HP 2000

COMPUTER SYSTEM

SOURCES AND LISTINGS

DOCUMENTATION

HP PART NUMBER 22687-90020

PREFACE

This manual is a guide to the sources and listings of the HP 2000 Computer System. The source of the system can be used to make modifications to the system; the listings and this manual can be used to help understand the internal workings of the HP 2000 Computer System.

This manual is divided into three parts. Part I is a description of the physical format of the magnetic tapes (22703-10001-Listing, 22703-10002-Sources, 22703-11001-Listings on 1600 bpi magnetic tape, and 22703-11002-Sources on 1600 bpi magnetic tapes) and a description of the BASIC program supplied to extract portions of the listings. Part II is the internal maintenance specifications for the system processor. It includes details on the operation of the system master program, the system loaders, the BASIC language processor, and the operating system. Part III includes details on the Input/Output Processor program and its configurator.

PART I

SOURCES AND LISTINGS

Sources

The source listings are supplied as BASIC formatted files on a selective dump tape of account Z101. Each line of source is stored in one string. The files are:

NAME	APPROXIMATE NUMBER OF BLOCKS	MODULE
S2883	715	System processor loader with 2883 disc driver
S5ISS	725	System processor loader with 7905/2883 disc driver
S7900	715	System processor loader with 7900 disc driver
S7905	725	System processor loader with 7905/7900 disc driver
SASFH	228	ASFH (IOP module)
SBSTLD	26	Paper tape bootstrap loader
SC2883	346	Conversion program for 2883 disc system
SC5ISS	344	Conversion program for 7905/2883 disc system
SC7900	346	Conversion program for 7900 disc system
SC7905	344	Conversion program for 7905/7900 disc system
SCIO	135	CIO (IOP module)
SCRO	11	CRO (IOP module)
SCRC	85	CRC (IOP module)
SCRSLD	7	2100 Cross loader
SD04	52	D.04 (IOP module)
SD110	11	D.110 (IOP module)
DS11C	120	D.11C (IOP module)
DS120	12	D.120 (IOP module)
SD12C	107	D.12C (IOP module)
SD130	10	D.130 (IOP module)
SD13C	70	D.13C (IOP module)
SD140	10	D.140 (IOP module)
SD14C	68	D.14C (IOP module)
SD2741	115	D.2741 (IOP module)
SD274C	27	D.274C (IOP module)
SD274E	27	D.274E (IOP module)
SD340	13	D.340 (IOP module)
SD34C	214	D.34C (IOP module)
SD43	63	D.43 (IOP module)
SD50CD	278	D.50CD (IOP module)
SD51	551	D.51 (IOP module)
SD530	9	D.530 (IOP module)
SD53C	225	D.53C (IOP module)
SD61	332	D.61 (IOP module)

NAME	APPROXIMATE LENGTH	MODULE
SD62	62	D.62 (IOP module)
SD63	200	D.63 (IOP module)
SDUMP	29	DUMP (IOP module)
SHIO	62	HIO (IOP module)
SHL0	14	HL0 (IOP module)
SHLC	180	HLC (IOP module)
SHM0	59	HM0 (IOP module)
SHP0IH	12	HP0IH (IOP module)
SHR0	14	HR0 (IOP module)
SHRC	151	HRC (IOP module)
SICKH	255	ICKH (IOP module)
SIOC	298	IOC (IOP module)
SIOPC	1214	IOPC (IOP module)
SLP0	11	LP0 (IOP module)
SLPC	68	LPC (IOP module)
SLT0	8	LT0 (IOP module)
SLTC	286	LTC (IOP module)
SMCPRG	86	Master Program
SMEMRY	44	MEMRY (IOP module)
SMN	183	MN (IOP module)
SMNRCD	76	MNRCD (IOP module)
SMNRIH	117	MNRIH (IOP module)
SMUXH	68	(IOP module)
SPP0	11	PP0 (IOP module)
SPPC	72	PPC (IOP module)
SPR0	6	PR0 (IOP module)
SRP0	9	PP0 (IOP module)
SPPC	157	PPC (IOP module)
SSYNCD	214	SYNCD (IOP module)
SSYNIH	355	SYNIH (IOP module)
STHGH	16	THGH (IOP module)
STSK	7750	System Processor
SWSPRG	36	warmstart program

Approximate Total 19200 Disc Blocks

The source code is HP 2100 assembly language. HP supported assemblers are available for HP DOS and HP 3000 systems. An assembler is available from the HP contributed library for use on IBM 360/370 systems. Care must be taken to assure sufficient symbol table space exists, especially for the system processor.

LISTINGS

The system software listings are supplied as BASIC formatted files on a selective dump tape of account Z102. Each line of the listing is stored as a string. The first character of each string is actually a line printer CTL code which should be written to the line printer before the line in order to insure proper spacing on the output listing.

The program LISTER has been supplied to produce listings for the user in the proper format. To use this program, log on to account Z102. Then:

```
EXE-LISTER
LISTER

Enter name of module to be listed ? XXXXXX
Enter listing destination (Hit <CR> for terminal) YYY

DONE
```

XXXXXX is the name of the module to be listed, XXX is the file designator of the listing device.

The files are:

NAME	APPROXIMATE NUMBER OF BLOCKS	MODULE
L2883	1429	System processor loader with 2883 disc driver
L5ISS	1497	System processor loader with 7905/2883 disc driver
L7900	1429	System processor loader with 7900 disc driver
L7905	1454	System processor loader with 7905/7905 disc driver
LASFH	191	ASFH (IOP module)
LBSTLD	56	BSTLD (IOP module)
LC2883	798	Conversion Program for 2883 Disc System
LC5ISS	795	Conversion Program for 7905/2883 Disc System
LC7900	798	Conversion Program for 7900 Disc System
LC7905	797	Conversion Program for 7905/7900 Disc System
LCIC	128	CIO (IOP module)
LCR0	12	CR0 (IOP module)
LCRC	77	CRC (IOP module)
LCRSLD	13	2100 Cross Loader
LD04	42	D.04 (IOP module)

NAME	APPROXIMATE LENGTH	MODULE
LD110	9	D.110 (IOP module)
LD11C1	115	D.11C (IOP module)
LD120	11	D.120 (IOP module)
LD12C	92	D.12C (IOP module)
LD130	8	D.130 (IOP module)
LD13C	72	D.13C (IOP module)
LD140	9	D.140 (IOP module)
LD14C	59	D.14C (IOP module)
LD2741	93	D.2741 (IOP module)
LD274C	17	D.274C (IOP module)
LD274E	19	D.274E (IOP module)
LD340	11	D.340 (IOP module)
LD34C	180	D.34C (IOP module)
LD43	68	D.43 (IOP module)
LD50CD	221	D.50CD (IOP module)
LD50IB	214	D.50IB (IOP module)
LD51A	520	D.51 for 16 ports (IOP module)
LD51B	527	D.51 for 32 ports (IOP module)
LD530	6	D.530 (IOP module)
LD53C	189	D.53C (IOP module)
LD61	312	D.61 (IOP module)
LD62	68	D.62 (IOP module)
LD63	172	D.63 (IOP module)
LDUMP	27	DUMP (IOP module)
LHI0CD	52	HI0 for CDC (IOP module)
LHI0IB	54	HI0 for IBM (IOP module)
LHL0CD	14	HL0 for CDC (IOP module)
LHL0IB	14	HL0 for IBM (IOP module)
LHLCCD	157	HLC for CDC (IOP module)
LHLCIB	160	HLC for IBM (IOP module)
LHM0CD	56	HM0 for CDC (IOP module)
LHM0IB	53	HM0 for IBM (IOP module)
LHR0IB	13	HR0 for IBM (IOP module)
LHR0CD	14	HR0 for CDC (IOP module)
LHR0IB	14	HR0 for IBM (IOP module)
LHRCCD	125	HRC for CDC (IOP module)
LHRCIB	127	HRC for IBM (IOP module)
LICKH	251	ICKH (IOP module)
LISTER	1	Program to produce listings
LIOC	255	IOC (IOP module)
LIOPC	1069	IOPC (IOP module)
LLP0	11	LP0 (IOP module)
LLPC	65	LPC (IOP module)
LLT0	8	LT0 (IOP module)
LLTC	233	LTC (IOP module)
LMCPRG	163	Master Program
LMEMRY	37	MEMRY (IOP module)
LMN	165	MN (IOP module)
LMNRCD	67	MNRCD (IOP module)

```
NAME      LENGTH     MODULE
LMNRIB     101       MNRIB (IOP module)
LMUXH       65       MUXH  (IOP module)
LPP0        10       PP0   (IOP module)
LPPC        69       PPC   (IOP module)
LPR0         5       PR0   (IOP module)
LRP0         9       RP0   (IOP module)
LRPC       132       RPC    (IOP module)
LSYNCD     197       SYNCD (IOP module)
LSYNIB     328       SYNIB (IOP module)
LTBGH       15       TBGH  (IOP module)
LTSB      8257       System processor
LWSPRG      84       warm start program

Approximate Total 24987 Disc blocks.
```

PART II

INTERNAL MAINTENANCE SPECIFICATIONS

TABLE OF CONTENTS

INTRODUCTION.....1-2

Hardware Configuration.....1-2
Memory Map.....1-3

MASTER PROGRAM

Introduction.....2-1
Conventions for use of Id Records.....2-2
I/O Configurator Methodology.....2-3
Id Records.....2-5
Master Tape.....2-6
Patching the Master Tape.....2-7
Loading the Master Program on a 2100 based system.....2-8

LOADERS

Loaders.....3-1
General Description.....3-2
System Update.....3-3
Mag Tape Reload.....3-3
Disc Reload.....3-4
Access-1A Upgrade.....3-4
Disc Bootstrap.....3-5
Sleep and Hibernate.....3-5
System and Feature Level Codes.....3-5
Patch Date Codes.....3-7
Selective LOAD/DUMP/RESTORE.....3-7
Loader Routines.....3-11
Warm Start Program.....3-17
System Generation.....3-21
Disc Reload.....3-22
Mag Tape Reload.....3-23
System Update.....3-25
Disc Organization.....3-26
Disc Error Routines.....3-39
Cold Dump Program.....3-44
Magnetic Tape Format.....3-49
Sleep and Hibernate Tape Formats.....3-50
Dump Tape Format.....3-52

OPERATING SYSTEM

Scheduling....4-1
Communications....4-5
System Modules....4-6
Disc Driver....4-6
System Console Driver....4-8
Processor to Processor....4-10
Processor Interconnect....4-11
System processor to I/O processor....4-13
I/O processor to System processor....4-26
Power Failure and Recovery....4-32
System Tables....4-46
Directory....4-47
Direc....4-49
Id Table....4-50
Idec....4-50
Swap Areas Table....4-52
Adt....4-52
Locked Blocks Table....4-52
Fuss....4-53
Comtable....4-55
Loggr....4-61
Teletype Table....4-62
Equipment Table....4-69
Master Segment Table....4-71
Muerto....4-72
Moving Head Disc Table....4-73
Device Table....4-74

BASIC INTERPRETER

Notes On Basic....5-1
Syntax....5-2
Phase II....5-2
Compilation....5-2
Value Table....5-3
Decompilation....5-4
PRNST....5-4
Execution....5-5
Main Loop....5-5
Statement Execution....5-5
LET....5-5
IF....5-5
GOTO....5-6
GOSUB....5-6
FOR....5-6
NEXT....5-6
RETURN....5-7
INPUT....5-7

BASIC INTERPRETER (Continued)

ENTER....5-9
READ....5-9
LINPUT....5-9
PRINT....5-10
PRINT USING....5-11
RESTORE....5-11
MAT....5-11
END....5-13
CHAIN....5-13
ASSIGN....5-14
SYSTEM....5-14
CONVERT....5-15
LOCK/UNLOCK....5-15
CREATE....5-16
PURGE....5-17
ADVANCE....5-17
UPDATE....5-17
IMAGE, COM, DIM, DEF, DATA, FILES, REM....5-17
Formatter....5-18
Error Routines....5-19
BASIC Core Maps....5-21
Syntax....5-21
Compilation....5-23
Value Storage Allocation....5-24
Execution....5-26
Internal Representation....5-28
Variable Operands....5-28
Constant Operands....5-29
BASIC Operators....5-31
BASIC Statement Types....5-32
Pre-defined Function Table....5-33
Extended String Representation....5-34
Examples....5-35
Symbol Table....5-38
Files....5-39
File Table Entry (ASCII files)....5-39
File Table Entry (BASIC files)....5-40
File Table....5-41
File Contents....5-41
BASIC Formatted Files....5-42
Update Last Changed Date Routine....5-46
Run Time Stacks....5-47
Return Stack....5-47
For Stack....5-48
Operator/Operand Stack....5-49
Language Processor Tables....5-50
Flow Charts
Syntax....1
Compilation....1 thru 4
Decompilation....1
PRNST....1 thru 2
SSYMT....1

BASIC INTERPRETER (Continued)

ASYMT....1
RSTPT....1
ALCOM....1
VALUE....1 thru 5
Execution....1 thru 2

LIBRARY OVERLAY REGION....6-1

User Overlays....6-2
Append....6-2
Assign....6-2
Bye....6-3
Catalog....6-4
Chain....6-5
Csave....6-6
Delete....6-7
Device....6-7
Directory....6-8
Echo....6-10
Dump....6-10
Execute....6-14
File Command....6-15
Files Statement....6-17
Get....6-18
Hello....6-20
Purge....6-21
Length....6-22
Library....6-22
Group....6-22
List/Punch....6-23
Load....6-24
Message....6-26
Name....6-28
Create....6-28
Unrestrict....6-29
Protect....6-29
Lock/Unlock....6-30
Private....6-30
SWA....6-30
MWA....6-31
Pause....6-32
Renumber....6-32
Report....6-34
Save....6-35
Supersave....6-37
Tape Mode Clean-up....6-41
Time....6-42
Console Overlays....6-42
Announce....6-42
Assign....6-43
Awake....6-44
Banner....6-44
Bestow....6-45

LIBRARY OVERLAY REGION (Continued)

Break....6-45
Changeid....6-46
Copy....6-47
Hibernate....6-48
Divice....6-48
Directory....6-49
Dump....6-50
Disconnect....6-50
Killid....6-51
Mlock....6-52
Munlock....6-52
Newid....6-52
Supernew....6-53
Phones....6-53
Purge....6-53
Report....6-54
Reset....6-56
Roster....6-56
RJE....6-56
Sleep....6-57
Status....6-58

INTRODUCTION

The 2000 System consists of several separate programs which are run on two processors. The I/O Processor Program is responsible for handling all multiplexed I/O from user terminals, for I/O to optional non-shareable devices and for remote system data communications.
The system contains the BASIC interpreter, executive, and library routines and runs on the main processor. The Loader, which also runs on the main processor, is responsible for generating initial systems, backing up the system on mag tape, reloading the entire system and user library, and selectively reloading or backing up users' libraries. The Loader also contains the cold dump routine which isa used to dump the contents of core from both processors, as well as selected portions of the disc, to mag tape.

Hardware Configuration

I. SYSTEM PROCESSOR

10-11 Processor Interconnect
12 System console
13 Time base generator
14 First disc
17-20 Mag tape (can occupy any unused select codes)

II. I/O PROCESSOR

A. See the operator's guide (22687-90005) and I/O processor IMS for details.

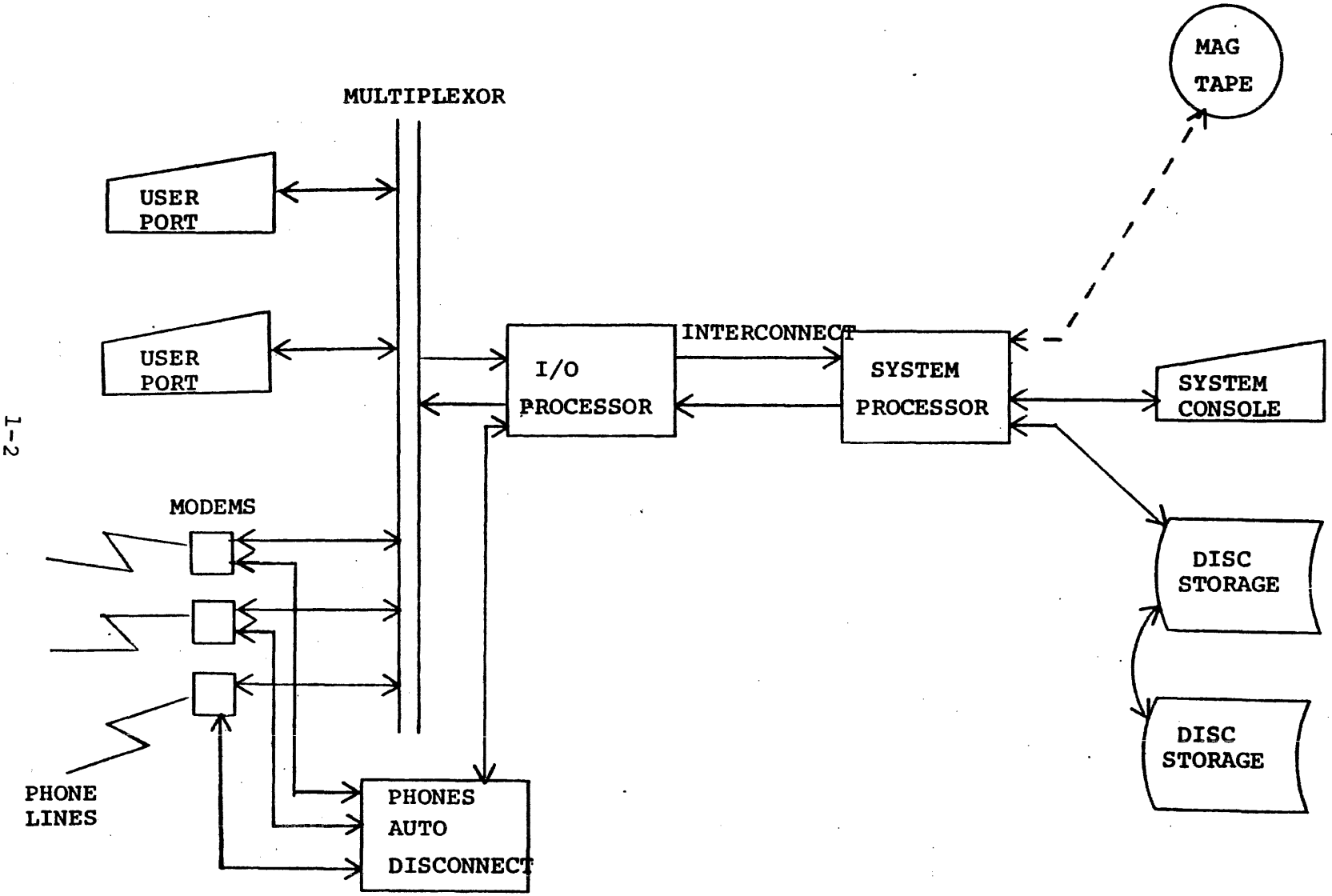


DIAGRAM OF S M HARDWARE

CORE_MAP

	APPROXIMATE OCTAL LOCATION	
	0	Interrupt linkage and uninitialized system variables
	100	Equipment table
	200	Constants and system variables
User	1231	Registers saved by clock
LIBUS	1235	User swap area and system library work area (10240 words from 2000 to 25777)
	24000	For disc resident salvage routines
	26000	Disc driver
	27700	Device table
	30401	IDEC
	30451	DIREC
	31571	ZDISC (disc driver driver)
	31605	BASIC
	54600	Formatter
	57414	Power fail/Restart routines
	60000	System console driver
	60447	Teletype tables
	62641	I/O processor communication drivers
	64000	Magnetic tape driver
	65417	System failure, retry, and salvage routines
	70000	Scheduler
	71200	SWAPR

	71670	Command table
	72242	System library subroutines
LIBRA	75000	System library programs swap area (512 words)
	77000	Cold dump

MASTER PROGRAM

I. INTRODUCTION

The Master Program (M.P.) supervises the generation or reconfiguration of Access systems invoking the desired configurator (IOP Configurator or TSB System Loader) and providing them with utility services. The M.P. is bootstrapped into the System Processor (SP) memory initially from the Master Tape and thereafter remains resident until loading of the SP program. Since it contains drivers for the system console and magnetic tape unit, the M.P. is stored onto the disc along with the TSB System Loader so that it can be recalled for service during system shutdown.

Programs intended to be co-resident with the M.P. must not alter the following areas of memory during their execution:

2-7:	Identical with the values used by the TSB System Loaders and the TSB System Program (except that the latter changes location 4 to enable powerfail recovery).
30(8)-77(8):	M.P. temporaries and linkage locations.
2000(8)-3777(8):	Contains the M.P. code and system console buffer.
MAGSC:	Contains the magnetic tape unit select code and system console type. This corresponds in location to part of the SP program's equipment table and will be in the range 100(8)-177(8).

Additionally, a call to the 'find id' routine (JSB 75B,I) will use a buffer origined at 300(8), potentially extending to 477(8). Complete safety can be assured by using only locations 10(8)-27(8), 1000(8)-1777(8), and 4000(8)-77677(8).

A program loaded in response to the "LOAD WHICH MODULE?" question is invoked by a JSB 4000B. Such a program can returncontrol to the M.P. after completion with a JMP 4000B,I. Certain errors detected by the M.P. will cause it to rewind the Master Tape and restart itself with the "LOAD WHICH MODULE?" question.

The following services are provided by the M.P.:

1. Magnetic tape unit driver (see listing for calling sequences) -invoked by a JSB 77B,I.
2. System console driver (see listing for calling sequences)- invoked by a JSB 76B,I.
3. 'Find id record' on Master Tape - invoked by a JSB 75B,I with the id number of the desired record in (A0 and zero in (B). (A -1 in (B) is used under special circumstances by the IOP Configurator only). Modules should assume that only a

single sequential pass will be made through the master tape.

4. 'read a data record' - invoked by a JSB 74B.I with a buffer address in (A). The M.P. will deliver the next record from the master tape id records are deleted from the stream of records delivered. This routine presupposes adjustment of the magnetic tape by some previous invocation of 'find id record'. The records thus read are presumed to have been generated by an HP Assembler and thus to be 64 or fewer words in length. Upon return (A) holds the record length in negative words or is zero if an end-of-file occurred (the routine automatically backspaces over the EOF. subsequent calls will return (A)=0 until 'find id record' is used to position the tape to another file). Checksums for both relocatable and absolute records are checked. Checksum errors will abort the process.

5. Location 73B holds the address of the end-of-tape flag. The word referenced will be zero unless the last magnetic tape operation finished past the EOT mark of the tape.

6. After return from an input request directed to the system console. location 72B will contain a character pointer to the response string (0 to 72 characters followed by a carriage-return). Since this word is reset with each call. modules can freely alter it as desired.

7. After return from a 'read data record' request (JSB 74B.I). location 71B will contain a copy of the buffer address supplied in (A) by the caller. Since this word is reset with each call. modules can alter it (for example. use it as an extraction pointer. advancing it after loading each word of the record read). Caution: a call to routine 'find id record' (JSB 75B.I) also alters this location.

II. Conventions for use of Id Records

Strict adherence to the conventions outlined here are not required for proper functioning of the master configuration strategy. but their use provides a consistent approach with maximum flexibility for future contingencies.

A. File format

1) All files except the master program and the Configured IOP program file should have the following format:
   a) File id record.
   b) One or more data groups (all absolute or all relocatable).
   c) End id record.
   d) End-of-file mark.

2) The Configured IOP program file must have the format:
   a) File id record (id number is 65000(10)).
   b) One or more data records in a format suitable for transmission to the IOP's protected loader or no records if no configured IOP program exists on the tape.
   c) End-of-file mark.

B. Patch tape format (recommended)
   1) Master id record (id number is 0).
   2) One or more data groups in ascending id number order (both absolute and relocatable data groups can be present on the patch tape).
   3) End id record (id number must be 65535 =177777(8)).

C. Data group format
   1) Group id record (bit 15 of info word = 1 if NAM record follows, 0 otherwise).
   2) None or more data records. If the data group is relocatable, the records from one complete program unit beginning with the NAM record and terminating with the END record; if the data group is absolute, each record is in the standard absolute binary form.

The use of id records for relocatable modules is straightforward. The assembler output for each separately assembled program unit is preceded by an id record, forming a data group. Such a group is a complete support of a patch tape or is combined with other groups into a file on the Master Tape.

The use of id records for absolute modules is somewhat arbitrary; the number and placement should be chosen to facilitate generation of correction records on a patch tape. Id records can be conveniently generated as part of the absolute assembly by situating them just prior to each ORG psuedo-statement of the program proper (i.e., just before the END statement). Of course, each group of absolute records on a patch tape should be preceded by an id record to indicate where they should be merged during generation of a new (i.e. patched) Master Tape.

III. I/O Configurator Methodology

To configure an IOP program:
1) Call 'find id' routine (JSB 758,I) with (B) = 0 and (A) = id number of standard modules file (2000(10)).
2) Read the file sequentially by calls to 'read data record' (JSB 748,I with (b) = buffer address) until the end-of-file is reached ((A0 = 0 on return).

3) Position the Master Tape to the optional modules file by a JSB 75B,I with (B) = 0 and (A) = 3000(10) (id number of optional modules file).
4) Read the file sequentially by calls to 'read data record' (JSB 74B,I with (B) = buffer address) until either the end-of-file is reached ((A) = 0 on return) or all desired modules have been read.

If a configured copy of the IOP program is to be written on the Master Tape, continue with step 5 below. Otherwise proceed to Step 9.

5) Position the Master Tape to the configured IOP file by a JSB 75,I with (B) = -1 and (A) = 65000(10) (id number of Configured IOP file).
6) Write the configured IOP program using direct calls to the magnetic tape driver (JSB 77B,I0. Before attempting to write each data record, check the end-of-tape flag (73B,I). If it is non-zero, issue a backfile/forward record request to purge the file written so far -

```
CCC                     backfile
CLB,INA                 forward one record
JSB 77B,I               request action
OCT 4                   (positioning request)
NOP                     these returns
NOP                       will not be taken.
```

Then issue a diagnostic and proceed to step 7 below.
Note: If the check for a write ring is made by a status call to the driver before performing step 5, the coding of the Master Program will allow dismounting the tape and remounting it with a write ring. A rewind should then be issued to ensure tape is at load point.

7) Write an end-of-file mark (this is to be done even if the end-of-tape flag is set) and backspace over it. (A) = a convenient id number (the id number of the IOP configurator's end id record, presumably 1999(10), will do nicely).

8) Return to the M.P. with a JMP 4000B,I.

To read a configured IOP program from the Master Tape:

1) Position the tape to the Configured IOP file by a JSB 75B,I with (B) = -1 and (A) = 65000(10).
2) Read the file using direct calls to the magnetic tape driver (JSB 77B,I). After the end-of-file mark is read, backspace over it. If no records were read, then no configured IOP program exists on the Master Tape; inform the operator.
3) Reposition the tape by a JSB 75B,I with (B) = -1 and (A) = a convenient id number (e.g., 1999(10) as in step 8 above).

4) Return to the M.P. with a JMP 4000B,I.

IV. ID RECORDS

An id record is exactly five words long and has the following form:

```
word 0:  1000B
word 1:  2001B
word 2:  ID number
word 3:  Info word
word 4:  Checksum
```

The id number is a 16-bit positive integer (0 < id < 65535). Each file on the master configuration tape must begin with an id record whose id value is a multiple of 1000(10) (allowing up to 65 distinct files on the Master Tape: bit 15 must be set if the id record precedes a NAM record (i.e., is the group id of a non-null relocatable data group). In all other cases it must be zero (absolute data group). This convention is used to determine which of two checksums to calculate as records of a module are read from the Master Tape. An absolute checksum check performed. No checksum check is performed on records of the configured IOP since these records are in core image format.

The following absolute program fragment will generate an id record:

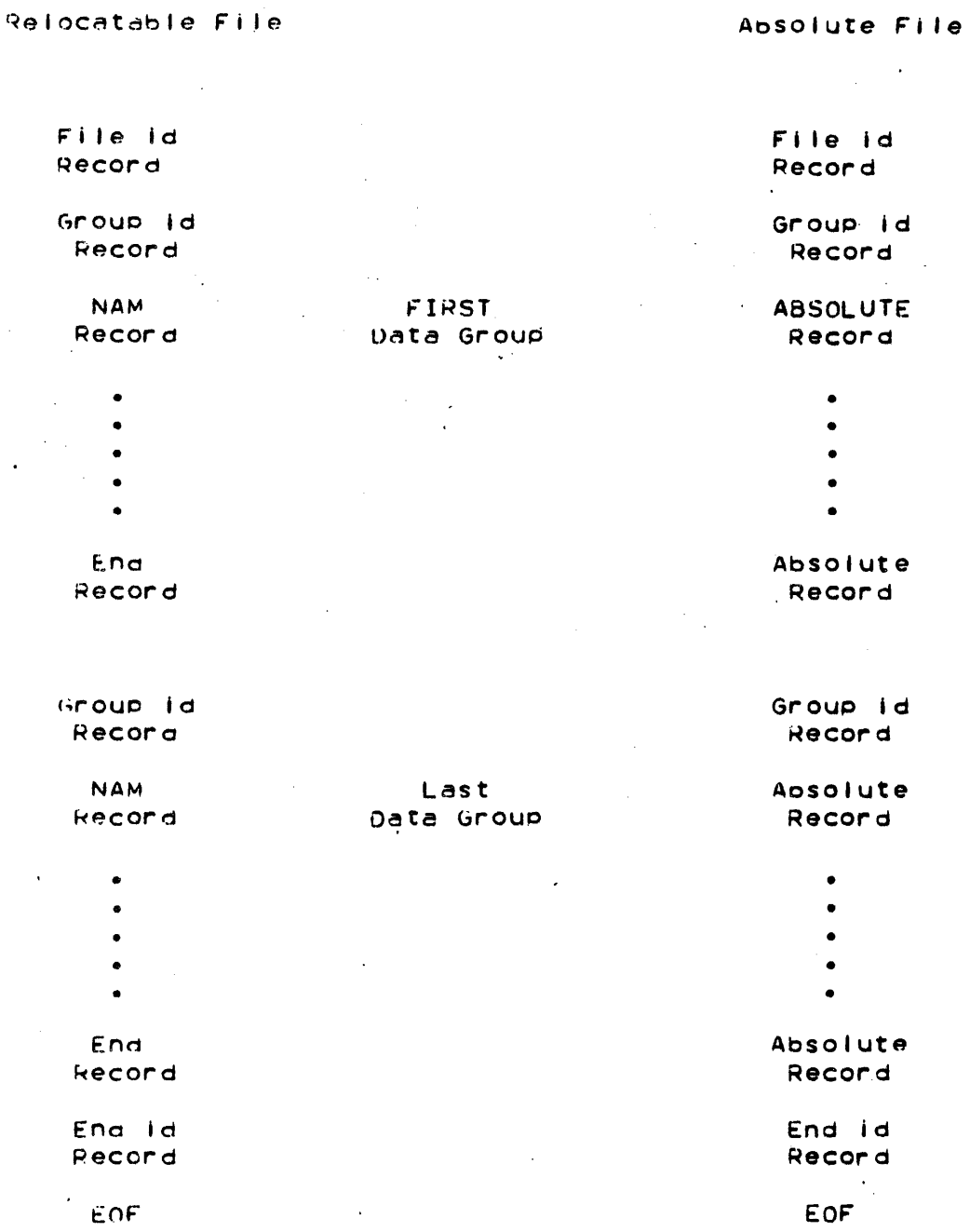
```
        ORG 2001B
        ABS IDVAL    IDVALUE
        DEC 0        INFO WORD
        BSS 1        FORCE END OF RECORD
```

Currently assigned file id numbers are as follows:

FILE	ID NUMBER (DECIMAL)	DATA FORMAT
IOOP CONFIGURATOR	1000	ABSOLUTE
IOP STANDARD MODULE	2000	RELOCATABLE
IOP OPTIONAL MODULES	3000	RELOCATABLE
WARMSTART	7000	ABSOLUTE
7900 LOADER	10000	ABSOLUTE
2883 LOADER	11000	ABSOLUTE
7905/7900 LOADER	12000	ABSOLUTE
7905/2883 LOADER	13000	ABSOLUTE
TSR SYSTEM	20000	ABSOLUTE
7900 FILE CONVERSION	25000	ABSOLUTE
2883 FILE CONVERSION	26000	ABSOLUTE
7905/7900 FILE CONVERSION	27000	ABSOLUTE
7905/2883 FILE CONVERSION	28000	ABSOLUTE

V. MASTER TAPE

The contents of the master tape for ACCESS consist of R+2 files, each followed by an end-of-file mark. The first file is the Master Program, composed of a sequence of records in absolute binary format (number of data words, memory origin, data words, and checksum). The M.P. must be loaded from mag tape; thereafter it provides the facilities for loading the configuration modules. The last file is initially empty except for the file id record. It can be overwritten with a configured IOP program through an option of the IOP Configurator. The middle n files are the IOP configurator, IOP program modules, warmstart, loaders, the ACCESS System, and the file conversion modules. Each of these files consists of a file id record, one or more data groups, and an optional, but recommended, end id record. A data group consists of a group id record followed by a relocatable group (NAM record through END record), or sequence of absolute binary format records, or a null record set (i.e., a group record may be followed by another id record or the file's end-of-file mark). All id records have the same format, the destinctions above are functional. Each id record contains a unique integer (id) which must be greater than the id of the physically preceding id record.



VI. Patching the Master Tape

The Master Program does not support patching of modules using a paper tape containing patches. However, the Master Tape can be directly patched in the following manner.

The old Master Tape may be merged with a patch tape creating a new Master Tape. Assuming that the patch tape follows the conventions for id records, the newly created Master Tape will contain the new patches. Hence when any module is loaded the patches have already been included. Note that the software to do this is not part of the ACCESS system itself and the Master Program can not be patched as it does not contain an ID record.

VII. Loading the Master Program on a 2100 based system.

Whenever it is desired to use the Master Program on a 2100 based system for whatever reason: system crash, changing IOPs, system update, etc. the following procedure should be followed.

The objective is to read the Master Program from mag tape using the mag tape bootstrap contained on paper tape. This bootstrap must be cross-loaded from the IOP to the system processor using the ACCESS cross loader since the system processor has no photo-reader.

The cross loader must be read from special ACCESS Basic Binary Loader. Hence to reload the system processor, the IOP must be clobbered with the crossloader. Obviously the IOP must be reloaded prior to reloading the system processor.

The mag tape bootstrap, once cross loaded into the system processor, may be used twice. Once to load or reload the IOP and once to reload the system processor.

2100 Cross Loader

The ACCESS Cross Loader binary resides on the first file of the Cross Loader/Bootstrap paper tape. The Cross Loader is read from the IOP's paper tape reader using the ACCESS protected Basic Binary Loader.

Once the Cross Loader has been read into the IOP's memory the contents of the A and B registers should not be altered as they contain the select codes of the paper tape reader and the interconnect kit. The ACCESS protected loader conveniently sets these registers for the Cross Loader.

To cross load the bootstrap (or any program), start the system processor's protected loader (reading from the interconnect kit). On the IOP, place the program in the photoreader and set P = 2 and press "run". A halt 77B on both processors signals a successful cross load. The Cross Loader may be restarted after a halt 77B by simply pressing "run" if it is desired to cross load another program.

2100 Mag Tape Bootstrap

The mag tape bootstrap is the second file on the crossloader/ bootstrap paper tape. The crossloader sends the bootstrap across the interconnect kit to the system processor where it is read and placed into memory byt the protected loader. The bootstrap may then be started at either 2000B or 4000B (2000B contains a jump to 4000B). However, if the bootstrap has just been used to reload or configure the IOP it must be started at 4000B the second time. This is required because the IOP configurator destroys the jump at location 2000B.

Once the execution of the bootstrap is started, the user is prompted at the console for the select code of mag tape unit 0. Valid values are 14B thru 26B.

The Master Program is then read off of mag tape and loaded into memory. At this point the tape is rewound, the select of the mag tape is placed into bits 6 thru 11 of the switch register and the execution of the Master Program begins.

2100 IOP Basic Binary Loader

The IOP on 2100 based systems contain a special ACCESS version of the protected Basic Binary Loader. Consult the Loader manual for procedures for loading this loader.

This loader has two starting addresses for each of three possible memory sizes. To read from the interconnect kit, start the loader at location X7700. To read from the paper tape reader start the loader at location X7750.

where X=7 for a 32 K IOP
X=5 for a 24K IOP
X=3 for a 16K IOP

In either case, successful execution is indicated by a halt 77B. If successful, the select codes of the photoreader and the interconnect kit are left in the A and B registers respectively. These register settings are required by the Cross Loader.

2000 ACCESS LOADERS

There are four loaders available for 2000 ACCESS systems. The specific loader to be used depends on the type of discs on the system. All loaders are functionally equivalent. There is a unique ACCESS loader for each of the followig combinations of discs:

7900 only
2883 only
7905 and 7900
7905 and 2883

Note that all other combinations of discs on a single system are explicity disallowed. All subsequent references to "Loader" imply all four, unless explicity noted.

The Loader is a separate program which runs on the system computer. It is explicitly loaded by the operator to perform the following functions:

1. Generate a new system from mag tape.

2. Update an existing disc-resident system from mag tape.

3. Reload a system from magnetic tape.

4. Reload a system from disc using the master tape.

5. Upgrade an ACCESS 1A system to an ACCESS 1B system.

The loader is implicitly loaded when the operator requests any of the following functions:

6. A normal load from disc of a slept or warmstarted system using the disc bootstrap loader.

7. A sleep or hibernate.

In addition, the loader contains the moving head disc driver for the system, the cold dump bootstrap, and cold dump program.

GENERAL DESCRIPTION

1. Initial System Generation

The Loader generates the system tables by asking the configuration option questions. If no options are specified then it sets the EQT to default values: 1 disc on the system, 6 directory track per disc, and 1 ID track. If the system discs are exclusively 7900s then the default is 1 directory track per disc.

Discs are checked for labels. The Locked Blocks Table is initialized to zeros only when the disc is not labeled for TSB and the operator wants it to be labeled.

The disc space claiming strategy is as follows: First build all ADT's in core from Locked Blocks Tables. Then claim disc space for the system, IDTs, ADTs and directory tracks using the ADTs in core, which are then written to disc. (The disc space required by system library routines and user swap areas are claimed after the first system library routine is read in because that routine is a table of the lengths of all the system library routines, according to which the disc space for the library can be computed and claimed.

The IDT, ADT and directory tracks are 32 contiguous blocks each and the swap tracks are 42 contiguous blocks each.

In order to minimize cylinder switching, the following algorithm is used for claiming disc space for the system tables and swap tracks:

a. Get the next piece of available disc space.
b. If it does not cross physical cylinder boundary, take it.
c. If it crosses physical cylinder boundary then take the next piece of available disc space which starts on physical cylinder boundary.

This single algorithm works for all disc types.

The rationale for being able to build all ADT's in core at 1400B words each and to have them remain memory-resident while claiming disc space is as follows:

A. Each ADT is built from Locked Blocks Table which is 1 block long (256 words) containing at most 128 entries, or 387 words (3 words/entry).

B. Therefore each initial ADT has at most 129 entries, or 387 words (3 words/entry).

C. When claiming space for the system segments, the number of ADT entries will not increase because they do not have to start on cylinder boundaries so that the ADT entries are either shortened or removed. Only claiming space for IDT, ADT and directory tracks may cause an increase of ADT entries because they have to meet the cylinder boundary conditions. The increase of ADT entries is at most:

   3(IDTs) + 8 (ADTs) + 10x8 (directory tracks) = 91 entries or 273 words.

D. Hence each ADT buffer needs at most 387 + 273 words = 660 (or 1224B) words.

E. Picking a nice even number, make it 1400B words long each. Then the buffer length for 8 ADTs is 14000B which can easily fit in memory starting at 32000B.

The system is read from mag tape and remains in memory. After the first system library routine is read, the system segments 1 and 2 occupying locations 32000B through 52000B are written out to the disc so that this portion of memory can be used for ADT buffer when claiming disc space for system library and swap areas. (The other system segments are written out after the COM6 table is set up.)

When all the system binaries are read, the remainder of the system is written to the disc. Then the pre-boot processor and final disc bootstrap are written to each disc; the system segments are restored in memory; the user swap areas are initialized and the DATE-TIME sequence is entered. Control is then transferred to TSB.

2. System Update

System update uses only the system binary load section of the load sequence. However instead of generating the EQT, it reads the EQT from disc. It returns the disc space occupied by the original system library to ADT and claims new disc space for the new library in order to allow expansion of system library as well as possible saving of disc space. It does not change any disc area other than the system code. Selective load and dump is processed before entering the DATE-TIME sequence.

3. Mag Tape Reload

Mag tape reload reads the EQT and DIREC from the hibernate or sleep tape rather than being initialized. Also, before the system is read, IDTs and directory tracks are read from tape and distributed evenly on disc.

After the system has been written to disc, the user library is loaded and stored on disc according to the allocation option specified.

Selective LOAD/DUMP is processed before entering the DATE-TIME sequence. It is described further in the Selective Load/Dump Section.

4. Disc Reloads

There are two procedures to reload a system from disc. The loader is explicitly loaded into memory for one procedure and implicitly loaded for the other. The latter will be explained in item 6.

The Master Tape must be used. If neither system generation or mag tape reload is specified, a disc reload is implied. Control is transferred directly to the section of the loader where it reads the disc bootstrap and SST from blocks 1 and 2. After checking for valid conditions, it reads in the system according to SST. Then it processes the configuration options, returns the old swap area to ADT, claims disc space for new swap areas, processes selective LOAD/DUMP commands and enters the DATE-TIME sequence. If the system (as read from disc) is not in a 'slept' state, warmstart may be attempted. Otherwise the reload is aborted.

5. ACCESS-1A Upgrade.

The LOADER (ACCESS-1B) knows that during a mag tape reload it may be reloading an ACCESS-1A system. If so, special processing is required. The following should have occured prior to attempting the reload.

1. Reconfigure the system processor and I/O processor hardware as required.
2. Configure an Access 1B I/O processor. (It must be running.)

During the mag tape reload, discs other than disc 0, must reconfigured using the DISC command. The differing system level codes are ignored. After the opportunity to perform LOAD or DUMP commands the system halts 77B, instead of going through the normal date and time sequence. At this point in time, the software is 1A (as far as the system processor is concerned), and the hardware and IOP are 1B.

In order to install the 1B software, set P=2000B and press run. The 1B master tape should be loaded and a system update specified. This completes the upgrade to ACCESS-1B.

6. Disc Bootstrap

The loader is entered implicitly through the bootstrap procedure:

A. The memory or ROM resident disc loader reads in sector 0 off disc 0 into memory starting at location 2055B, entering the pre-boot processor.

B. The pre-boot processor reads in the disc bootstrap and SST from blocks 1 and 2 of disc 0 into core starting at location 14500B and jumps into it.

C. The disc bootstrap then reads in the system according to SST and the same sequence of events occurs as outlined in item 4.

7. Sleep or Hibernate

For a normal sleep or hibernate, the loader is read into memory according to SST by the system library routine and control is transferred to the loader. It first copies the DIRECT and EQT to disc and then if DUMP was specified, it starts the mag tape dump procedure. If RELOAD was specified, a disc reload is performed (see item 4). If no option was specified, it just halts with 77B (SYSTEM SHUTDOWN).

The sequence of dumping is as specified in the sleep/hibernate tape format section. If sleep, the user files dumped are only those changed since the last hibernate; if hibernate, all user files are dumped.

8. System/Feature Level Codes

Each system and loader is assigned a system level code uniquely identifying that system (but not, in general, different versions). Each time a system is loaded, the loader will ensure that it is the correct one for that system.

A. On initial system generation or system update the loader will verify that the mag tape contains its system level code.
B. On magnetic tape reload the loader will verify that the set of tapes was generated by its system. The only exception is an ACCESS-1A upgrade (see item 5).
C. On disc reloads (bootstraps) the loader will verify that the system on the disc packs is its system.

If the system is not acceptable in any of the above, the loader will print ILLEGAL SYSTEM CODE. LOAD/DUMP ABORTED on the system console, and terminate loading. Of course, this should never occur. The system level codes assigned are:

2000C (High-Speed)	2000
2000F (Option 200/205)	3000
2000F (Option 210/215)	3500
2000 Access Release A	5000
2000 Access Release B	6000

The values are independent of the feature level codes described below.

Each system is also assigned a feature level code identifying the level of features it supports.

2000C (High-Speed)	200
2000F (Option 200/205)	200
2000F (Option 210/215)	200
2000 Access Release A	1000
2000 Access Release B	2000

The assumption is that all systems will be maintained as concentric subsets, but the scheme is flexible enough to abrogate this if necessary. The feature level code is used by the loader to ensure that a system is not loaded with a library containing programs with features it does not support. When selectively loading programs and files from magnetic tapes, if the tapes were produced by a system with a higher feature level the loader will print:

```
TAPE MAY CONTAIN FEATURES NOT SUPPORTED ON THIS SYSTEM
(xxx VS yyy)
DO YOU TAKE RESPONSIBILITY?
```

If the operator answers NO, the requested selective load is bypassed and the LOAD OR DUMP COMMANDS? message repeated to allow loading to continue. If the operator knows that in fact none of the programs or files to be loaded contain the extra features (xxx is the feature level code of the system, yyy is the feature level code from the magnetic tape), he may answer YES and the selective load will be performed. Any other answer will cause repetition of the message:

```
DO YOU TAKE RESPONSIBILITY?
```

When performing a system update, if the feature level code of the disc-resident library is higher than that of the system being loaded, the loader prints:

DISC MAY CONTAIN FEATURES NOT SUPPORTED ON THIS SYSTEM
(xxx VS yyy) DO YOU TAKE RESPONSIBILITY?

If the operator answers YES the system update proceeds anyway, retaining the higher level code. If he answers NO the message LOAD/DUMP ABORTED is printed and loading terminates. Any other answer will cause repetition of the message:

DO YOU TAKE RESPONSIBILITY?

9. Patch Date Codes.

In an effort to keep the system software and IOP software compatible, a scheme using date codes has been implemented. The initial IB software has a date code of 1624 for both the system and the IOP. Each time either processor software is updated the date code of both processors will be updated. Hence, both processors, should always have the same date code.

Each time the system is brought up, the dates codes are compared. If unequal the following message is printed.

IOP MAY CONTAIN FEATURES NOT SUPPORTED ON THIS SYSTEM
(XXX VS YYY) DO YOU TAKE RESPONSIBLITY?

Here xxx is the patch date code of the system and yyy is the patch date code of the IOP.

If the operator answers no, the load is aborted (must reload IOP, load aborted).

10. Selective LOAD/DUMP/RESTORE

Selective LOAD,DUMP, and RESTORE share most of the code with each other as well as with mag tape reload and mag tape dump.

It starts in the subroutine RLDC with the question "LOAD OR DUMP COMMANDS", processes the command, goes to mag tape reload section for LOAD or RESTORE and mag tape dump section for DUMP. When it finishes loading or dumping, it comes back to RLDC and asks again "LOAD OR DUMP COMMANDS?". This cycle repeats until the response is "NO" or a carriage return and then it exists from RLDC. LDFLG is set to non-0 for LOAD or RESTORE and 0 for DUMP.

SLFLG = 2 for LOAD all or DUMP all
= 1 for selective LOAD or
= 0 for mag tape reload
=-1 for selective RESTORE
=-2 for RESTORE ALL

word 10 of each directory entry is used as the "recovery" flag for that entry. The following values of the "recovery" flag are meaningful.

-1 -"must be recovered" during LOAD or RESTORE if the entry previously did not exist in the directory.
1 -"must be dumped" during DUMP. "should be recovered" during RESTORE (i.e. the program/file already exists in the directory).

The sequence of events for LOAD/RESTORE is:

a. Clear "recovery" flag for all directory entries.
b. Clear "constructed ID" table.
c. Ask the operator for a list of <ID> or <ID>.<FILE NAME> entries.
d. If an <ID> is entered, it is placed into the table -- call it "constructed ID table".
e. If <ID>.<NAME> is entered: set the directory entry's recovery flag.
f. Rewind the mag tape.
g. Read the label on load tape and check for valid feature level code.

h. Skip past the first EOF mark.

i. Read in first record which begins with the directory entry for that file.

j. If the entry is not in the directory go to step b. Otherwise, check the "recovery" flag: if not set, go to step m.

k. If the set "should be recovered" flag is set return space to the ADT.

l. Read in ADTs from disc one at a time to find the disc space for the file without removing space from the ADT yet.

m. Update the directory entry for this file with the "must be recovered" flag cleared.

n. Read a record at a time from mag tape and write to disc until the EOF mark.

o. Write updated directory track to disc.

p. Read in ADT again and call FSDAD to remove space from ADT.

q. If there are more files to load, go to step i. Otherwise, rewind and stand by.

r. Read in one ID track at a time. For each ID, read in one directory track at a time and compute the total disc space used by that ID and update the IDT. Enter RLDC again.

s. If the ID is in the "constructed ID table", put the file into the directory and go to step i; otherwise to step h.

The sequence of events for DUMP is:

a. Same as step a for LOAD/RESTORE
b. Same as step b for LOAD/RESTORE
c. Same as step c for LOAD/RESTORE
d. Same as step d for LOAD/RESTORE
e. Same as step e for LOAD/RESTORE
f. Same as step f for LOAD/RESTORE

g. Request "verify" option.

h. Set up tape label and write to mag tape.

i. Write EOF mark.

j. Read in one directory track at a time. For each entry, check the "recover" flag: if set, read the file from disc one record at a time and write to tape. The first record is preceded by the directory entry: the last record is followed by an EOF mark. If not set, check the "constructed ID table": if the ID is in the table, dump the file to mag tape.

k. When all the directory tracks are gone through, write an extra EOF mark, a final tape label and EOF mark. Rewind the tape.

l. Check verify flag. If not wanted go to step p.

m. Verify tape label and EOF mark.

n. Read in one directory track at a time. For each entry, check the "recovery" flag: if set, read the file one record at a time from disc into buffer at 32000B and read from tape into buffer at 52000B. Verify the contents of these two buffers with each other. If the flag is not set, check the "constructed ID table" and do the same verification if the ID is in the table.

o. Verify the EOF, trailer and EOF mark.

p. Rewind the stand by.

q. Print "DONE".

This section contains brief descriptions of some routines in the loader, arranged in alphabetical order.

BUMP This short routine bumps the pointer to next ADT track in the ADT buffer when all ADT tracks are core resident. It is called by CLAIM for setting up pointers to search next ADT track. (See paragraph on CLAIM).

CDB This routine is called by SYSLB to claim disc space for the system library and swap tracks. It calls FSDAD to find space for library which can be allocated anywhere on any disc. It calls CLMSW for claiming swap areas because they either start on physical cylinder boundary or do not cross physical cyclinder boundaries.

CFFW CFFW converts the first 4 words of a directory entry (ID and NAME) into printable ASCII format in a specified buffer.

CKFC CKFC checks for valid feature or patch date code. The feature code on tape or disc is considered invalid if it is greater than the feature code of the system. The IOP patch date code is considered invalid if it is greater than the patch date code of the loader. In these cases, the user may choose whether to take responsibilities or not; if not, then CKFC exists through P+2; otherwise exit to P+3.

CLAIM This routine finds a piece of disc space from ADT, which either starts on physical cylinder boundary or does not cross physical cylinder boundary, thus taking care of minimizing the interrupts resulting from switching of physical cylinders on both 7900 disc and 2883 disc.

The ADTs used by this routine may be either all in core buffer of 14000(8) words long or on disc to be read in one at a time. The subroutine which updates the ADTs and pointers is either BUMP or NUADT. The entry point of the appropriate subroutine is passed to CLAIM in the word CLMFL and CLAIM calls it indirectly through CLMFL.

CLMSW CLMSW claims swap areas of 40 blocks each and saves the disc addresses in the table TRKTb which is written to disc. Before entering the DATE-TIME sequence, the loader uses TRKTB to initialize the swap areas and TTY table.

DACV It converts a positive number of 6 digits from decimal to ASCII and puts into a 3-word buffer, right justified with leading zeros set to null characters.

DISCZ This routine provides the necessary environment for the moving head disc driver and transfers input/output/seek

requests to the actual driver.

In conjunction with the GMQBD routine, DISCZ provides a buffer for disc driver generated error messages and prints any such errors that occur.

When certain disc errors occur, DISCZ checks the disc error flag DSERZ. If it is zero, it halts, otherwise it gives control to the calling place through exit to P+2. DSERF is always cleared before DISCZ exits.

DREN The DREN routine is used to insert a 12 word directory entry into the directory. If the proper directory track is full, SUPDP is called to redistribute the dir. tracks. See the system documentation for supersave for a description of its operation.

EOTCH This routine checks for an end-of-tape (during Mag Tape Sleep operations). It is called when end of tape is not allowable (before first file mark). An end of tape prints a tape too short message and halts. Pressing run restarts the dump.

FNZSC FNZSC finds the next active disc (with non-zero select code) and returns with certain pointers set.

FSDAD FSDAD looks for disc space from the ADT. If insufficient room in ADT, it returns to P+1; otherwise it returns to P+2 with ADTEN set to point to the ADT entry and A-B registers set to the disc address. Disc space is not removed from ADT if DCFLG is -1 and is removed otherwise.

GMQBD This routine provides a placebox for the disc driver when it asks for an error message buffer. Either this routine or DISCZ will print out the message.

ISADT This routine inserts an entry into ADT and updates the ADT length should be in core buffer and certain points and parameters set.

ISADT This routine inserts an entry into ADT and updates the ATD length word in the EQT. On entry, the appropriate ADT should be in core buffer and certain pointers and parameters set.

LD101 This routine performs the following functions:

A. Reads back into core the four system segments that were overlayed by the ADT and IDT buffers.
B. Checks the IOP's date code in all cases except mag tape reload. If a mag tape reload is in progress, the IOP's date was checked previously.

C. Initialize the random number seed.
D. Write out the swapping portion of the language processor for each port. Prior to writing the swap track out for a port initialize the pointer to the ? ID word that port's teletype table entry.
E. Save the old device table in a buffer beginning at 25000B.
F. If upgrading from ACCESS 1A to 1B, set the system status into EQT to slept and go to step m.
G. Get the new device table from the IOP.
H. Update the device assignment word for each entry in the new device table that has a corresponding entry in the old device table. Thus maintain old device assignments in the new table.
I. Get the date from the operator.
J. Clear the switch register.
K. Get the time from the operator.
L. Copy the operators time and status of not slept into the EQT.
M. Save the disc address of the EQT and the IDEC table where the system processor can find them.
N. Write the EQT to disc.
O. Configure the mag tape driver.
P. If upgrading from ACCESS 1A to 1b halt (HLT 77B).
Q. Start the clock ticking every 100 milliseconds.
R. Print part number and date code on the console.
S. Enable console interrupts.
T. Enable power fail recovery.
U. Start IOP activity.
V. Disable RJE command if requested.
W. Begin timesharing.

LDR30 This routine checks each disc in the disc portion of EQT for a non-zero select code. For each disc found, the disc's label is read and verified (words 8 and 9 of the label which contain the system level and feature level codes are not verified). If any disc does not contain a valid label, the operator is given a change to have the disc labeled of TSB. Should the operator decide not to have such a disc labeled for TSB it is removed from the disc EQT and the MHTBL (which is reconfigured to reflect the removed disc). Disc 0 may never be removed. Should the operator decide to label a disc in question, the bad blocks table (block 3) is zeroed and the TSB label is written to the disc.

LDR50 This routine, starting at location 32000B, builds an ADT in core for each active (select code #0) disc. For each active disc in the disc EQT, the disc's locked blocks table is read into a buffer originating at 77300B. The ADT is built in core reflecting only the locked blocks for this disc. During the build process, the length of

the ADT for this entry (in the disc ADT descriptors part of the EQT) is update as each entry is added to the ADT. When the ADT for a particular disc is completed, the starting core address is bumped by 1400B words and another active disc is sought. This routine is terminated when no more active disc can be located.

LDR60 This routine claims disc blocks for:

A. the system
B. the ID table
C. the ADT
D. the directory

Space is claimed in such a manner that it will not cross a cylinder boundary unless the space also begins on a cylinder boundary. Spaced claimed for the system must be resident on disc 0. Also the disc addresses of the segments of the system are updated in the MST. The other spaces needed may be claimed from other discs if necessary. When the space for the directory tracks is claimed the initial pseudo entries are also written to disc. Finally the ADT's which have remained core resistent thus far are written to disc.

LDR75 This routine reads system and system library from mag tape and writes the overlays to disc.

Each record after being read from mag tape into a buffer at 77300B is processes as follows:

A. If end-of-file-write last overlay to disc, rewind the routine.
B. If the record is origined at location 4002B assume it is the Loader-System Linkage table and go to step D.
C. Check both the starting and ending location of the record. If either location corresponds to an area of memory that should not contain system code, abort the load.
D. Check the starting location of the record. If it begins at 75000B it must be a system overlay so call SYSLB to write the preceding overlay to disc.
E. Perform a checksum check on the record. Abort the load if the checksums conflict. Copy the record from the buffer to the appropriate core location.
F. Process the next record.

NUADT This routine is called by CLAIM indirectly through CLMFL to read a specified ADT into ADT buffer and to set DCADT to point to the beginning of the buffer and clear the flag LTMP1 for subsequent FSDAD call.

RDISL This subroutine is called each time the disc EQT is or might be changed. It sets the interrupt locations for all existing discs. Using the select code and bit 15 of each word in the disc EQT, this routine decides what type each disc is and reconfigures the MHTBL for discs 1 thru 7. All words except word 2 of a disc entry in the MHTBL may be changed. Also the value of MAXSC (truncated ninth entry) which contains a double integer specifying the first absolute sector number which does not map onto the system disc configuration.

RDLBL RDLBL calculates the disc address of a specified disc label, reads the 32-word label into the disc label buffer and checks the checksum for valid label.

RQINT This routine prints a specified prompt and interprets the user's input. An invalid input prints "ILLEGAL INPUT" and asks the question again. A simple CR or "NO" returns to P+2. Otherwise, return to P+1 with the integer (1 thru 10) in (A).

RTADT This routine returns any number of blocks of disc space to the ADT on disc. It checks to see which ADT the returning space belongs to and reads in the correct one. Then it decides whether to insert the new entry or to modify the existing entry/entries and updates the ADT to disc and updates the length word of ADT in EQT if necessary.

RTSWP This routine returns the swap areas to ADT according to the SWAP AREAS TABLE (SAT) on disc. It reads the table into core location 15500B and returns 40 blocks for every non-zero disc address in the table using RTADT. Then it zero's out the disc address and writes the table back to disc inorder to have the table kept up with the current status of ADT. It is called during disc reload and system update before claiming disc space for new swap areas.

SLDC This routine processes the LOAD/DUMP/RESTORE command. It only returns to the calling point when no more LOAD/DUMP/RESTORE commands are wanted. Otherwise, it takes the input line from the console and processes it. If only an ID is entered, it is placed into a table of ID's. If file names are entered as well, the entry is placed into the directory with the "recovery" set.

SUPDP This routine is used to balance all the directory tracks, with or without inserting a new directory entry as specified by the flag SUPTG.

SYSLB This routine is called by both system generation and

system update to write an overlay to disc. Mag tape reload calls SYSLH after loading each overlay. However both system generation and system update call SYSLB prior to loading each overlay, hence the first call in this case is always ignored. If bit 15 of the switch register is set, the load halts (HLT 15B) to allow for on-line patching prior to writing the overlay to disc (to continue simply press 'run').

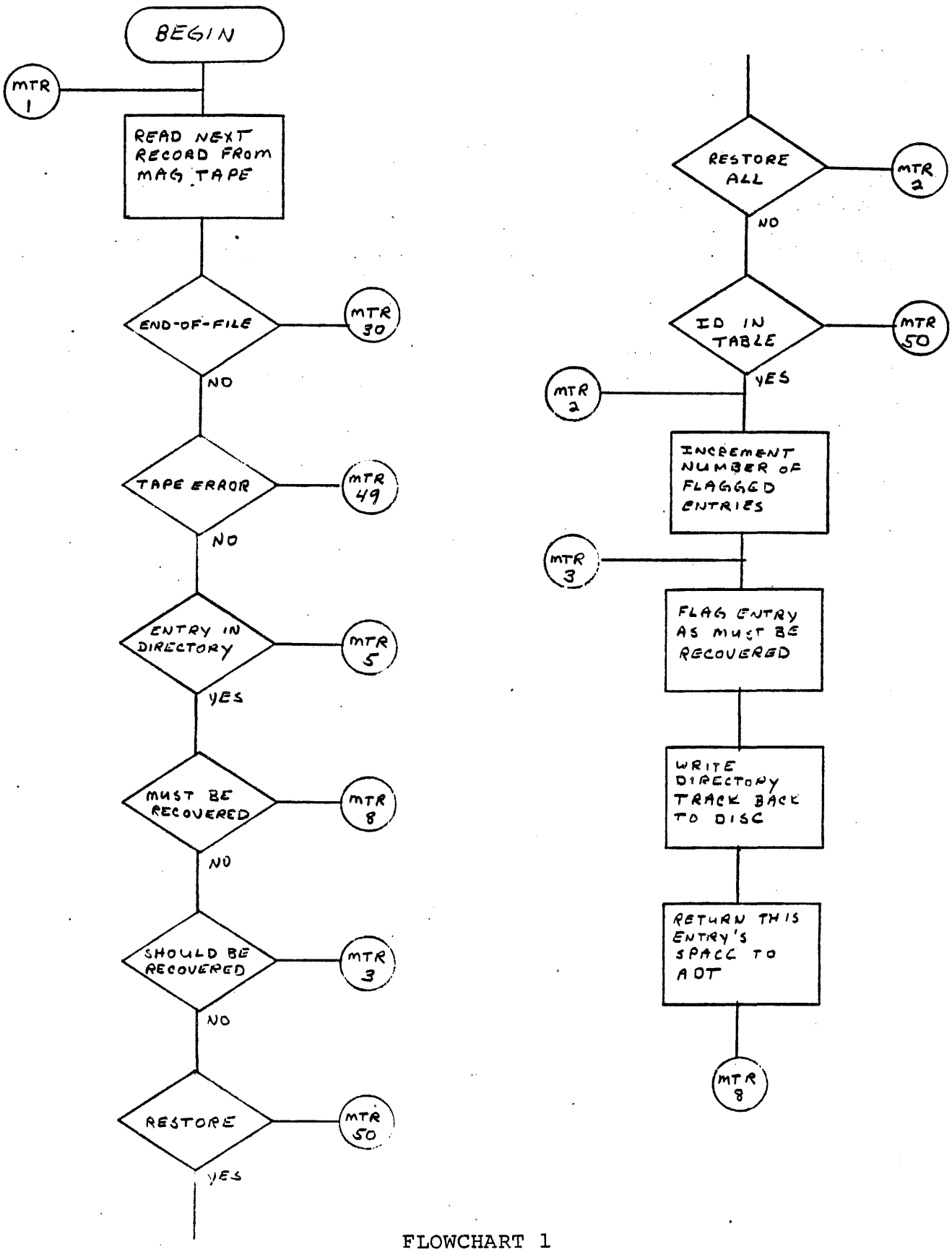
Prior to writing the first overlay to disc, the system and level codes are checked. Inconsistent feature level codes (excluding system update) allow the user to take responsibility.

Additionally, system segments one and two are written to disc to allow room from ADT buffers for space claiming purposes. The length of the overlay in blocks is calculated from the first overlay (library sizes table) and space for the overlays and the user swap tracks is claimed.

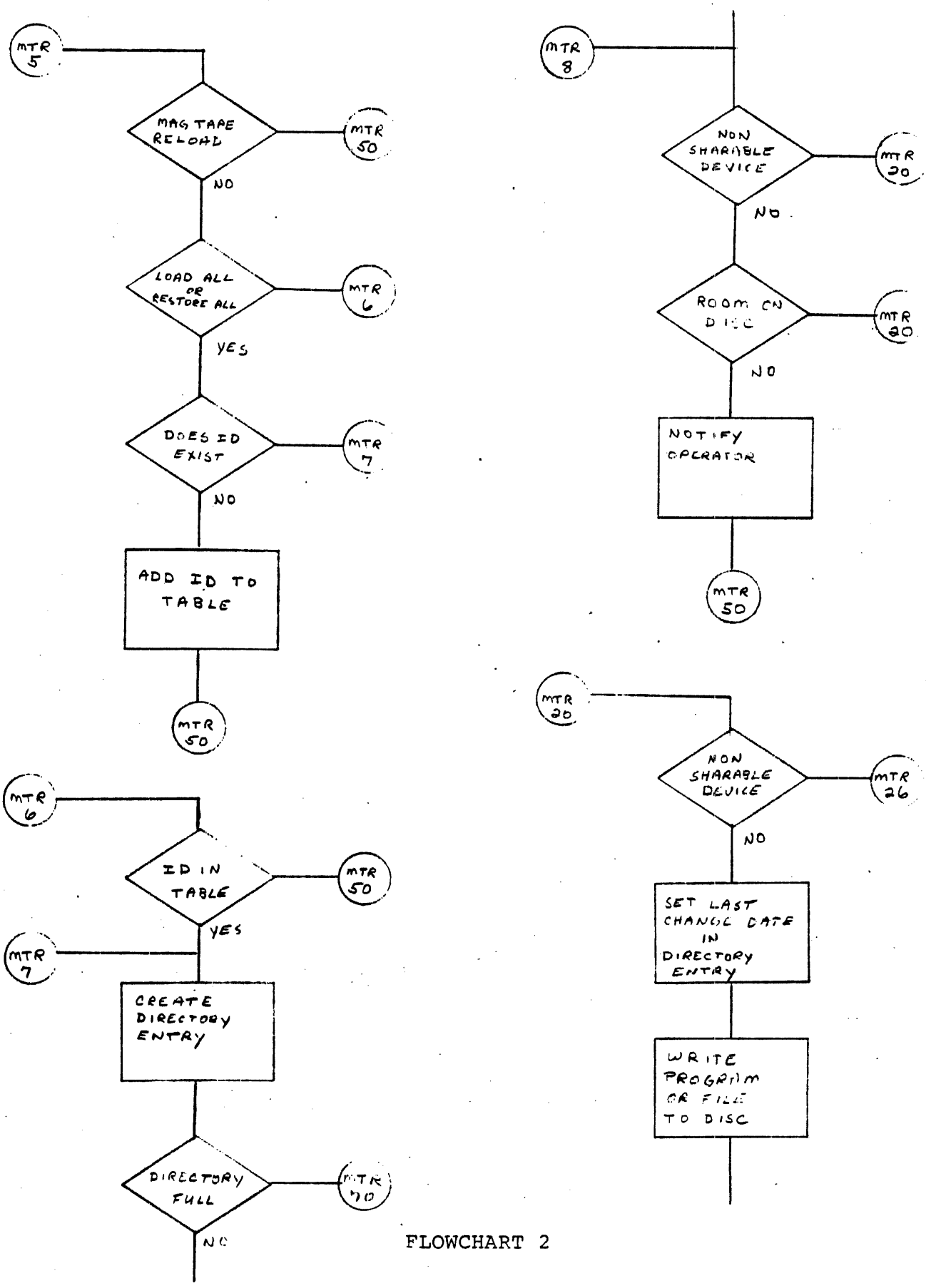
TPLR This routine reads a mag tape label into the first tape label buffer TLRT and makes sure that it has a TSB label and a reel number of one. Before reading, it initializes the system level and feature codes in the buffer to zeros because the TSB tapes prior to Option 200 did not have such codes and the tape levels were 7-words long instead of 9.

WDLTE This routine sets up the disc label, writes the label, pre-boot processor and the intermediate and final disc bootstrap onto each of the active.

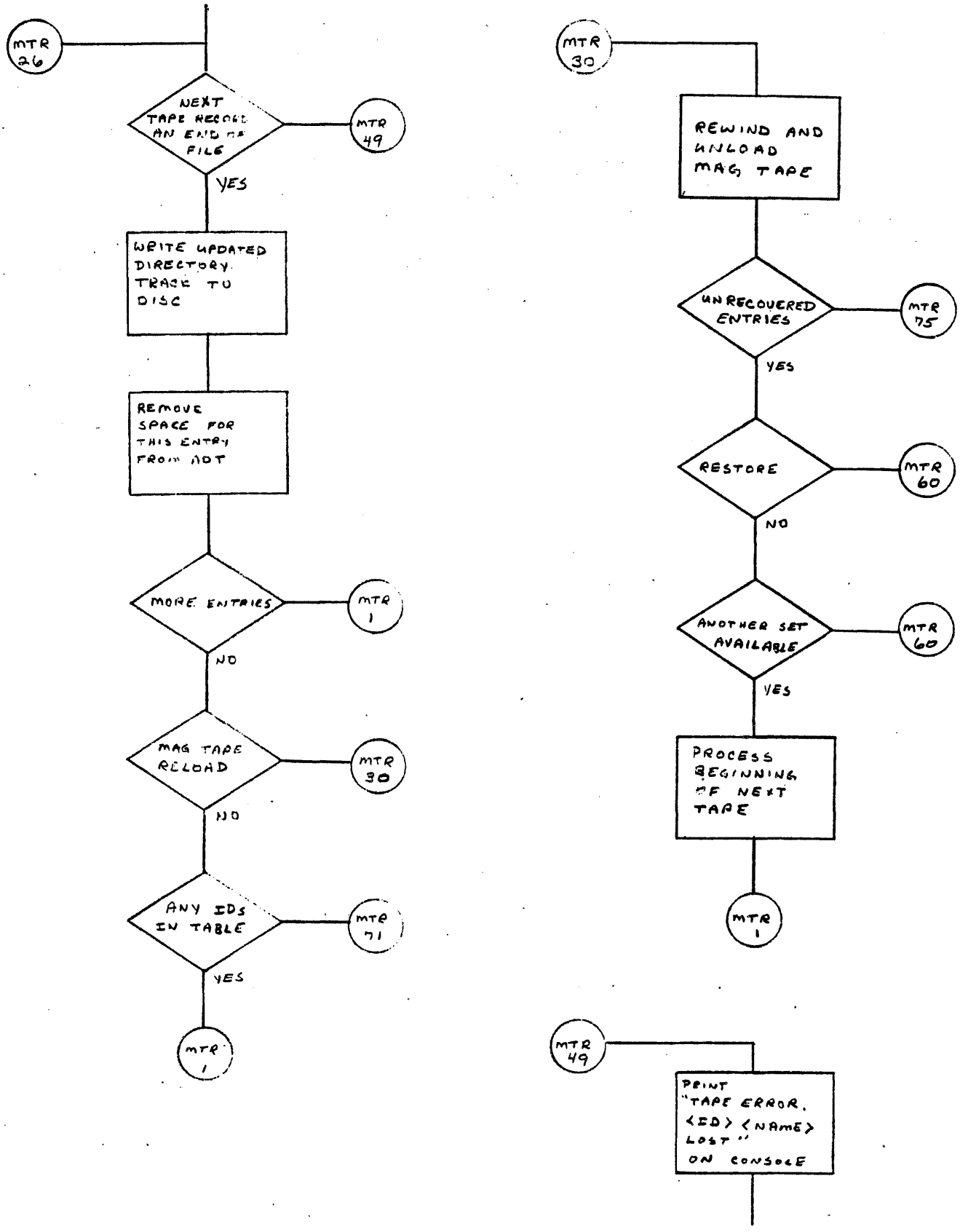
WTLBL This routine computes the checksum of the disc label and writes the entrie label sector onto the specified disc.


BEGIN
MTR 1
READ NEXT RECORD FROM MAG TAPE
END-OF-FILE
MTR 30
NO
TAPE ERROR
MTR 49
NO
ENTRY IN DIRECTORY
MTR 5
YES
MUST BE RECOVERED
MTR 8
NO
SHOULD BE RECOVERED
MTR 3
NO
RESTORE
MTR 50
YES
RESTORE ALL
MTR 2
NO
ID IN TABLE
MTR 50
MTR 2
YES
INCREMENT NUMBER OF FLAGGED ENTRIES
MTR 3
FLAG ENTRY AS MUST BE RECOVERED
WRITE DIRECTORY TRACK BACK TO DISC
RETURN THIS ENTRY'S SPACE TO ADT
MTR 8

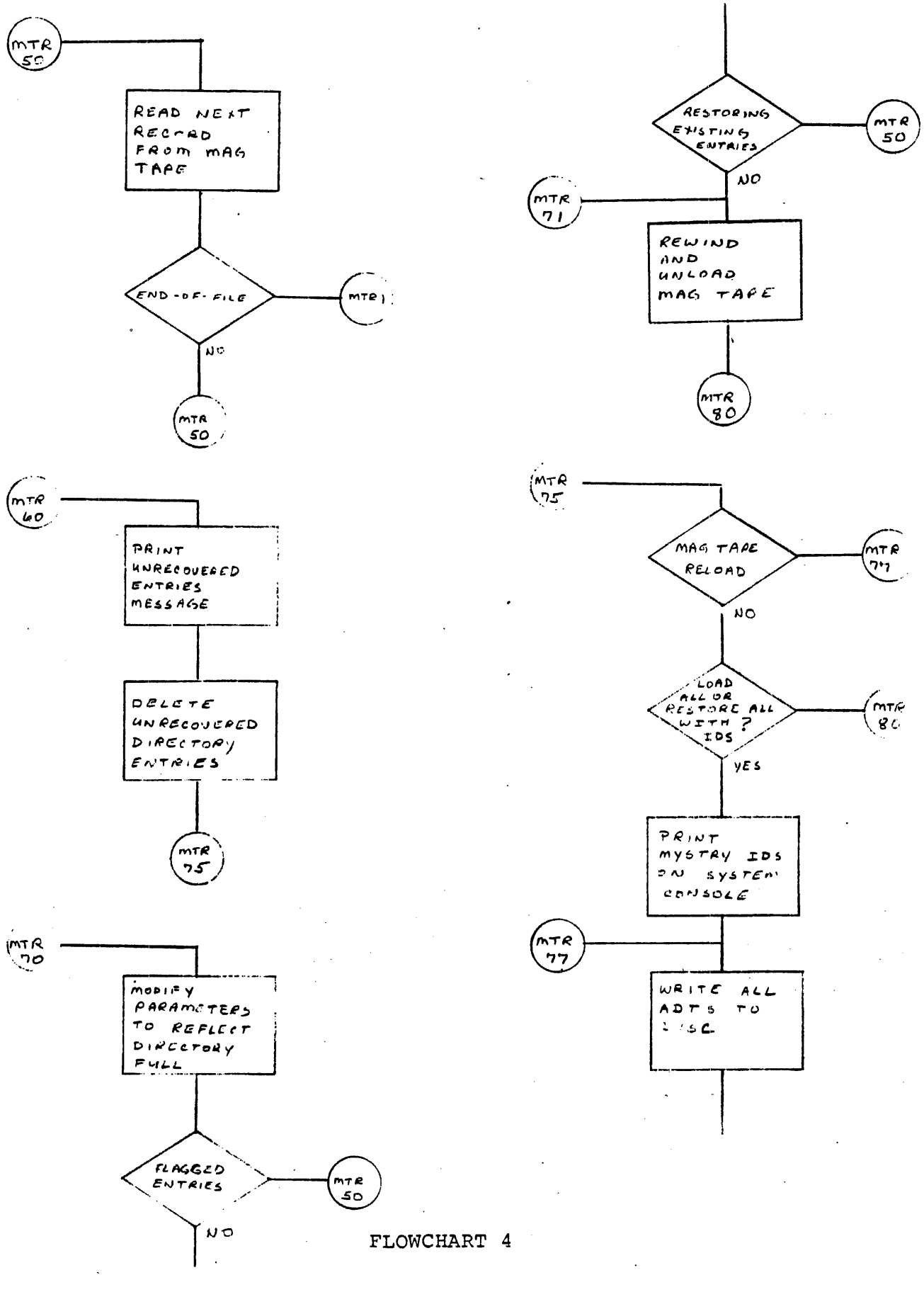
FLOWCHART 1


MTR 5
MAG TAPE RELOAD
MTR 50
NO
LOAD ALL OR RESTORE ALL
MTR 6
YES
DOES ID EXIST
MTR 7
NO
ADD ID TO TABLE
MTR 50
MTR 6
ID IN TABLE
MTR 50
YES
MTR 7
CREATE DIRECTORY ENTRY
DIRECTORY FULL
MTR 70
NO
MTR 8
NON SHARABLE DEVICE
MTR 20
NO
ROOM ON DISC
MTR 80
NO
NOTIFY OPERATOR
MTR 50
MTR 20
NON SHARABLE DEVICE
MTR 26
NO
SET LAST CHANGE DATE IN DIRECTORY ENTRY
WRITE PROGRAM OR FILE TO DISC

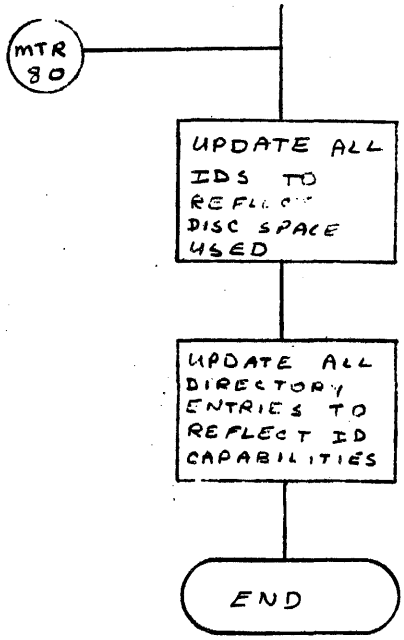
FLOWCHART 2


MTR 26
NEXT TAPE RECORD AN END OF FILE
MTR 49
YES
WRITE UPDATED DIRECTORY TRACK TO DISC
REMOVE SPACE FOR THIS ENTRY FROM ADT
MORE ENTRIES
MTR 1
NO
MAG TAPE RELOAD
MTR 30
NO
ANY IDS IN TABLE
MTR 71
YES
MTR 1
MTR 30
REWIND AND UNLOAD MAG TAPE
UNRECOVERED ENTRIES
MTR 75
YES
RESTORE
MTR 60
NO
ANOTHER SET AVAILABLE
MTR 60
YES
PROCESS BEGINNING OF NEXT TAPE
MTR 1
MTR 49
PRINT "TAPE ERROR, <ID> <NAME> LOST" ON CONSOLE

FLOWCHART 3


MTR 50
READ NEXT RECORD FROM MAG TAPE
END-OF-FILE
MTR 1
NO
MTR 50
RESTORING EXISTING ENTRIES
MTR 50
NO
MTR 71
REWIND AND UNLOAD MAG TAPE
MTR 80
MTR 60
PRINT UNRECOVERED ENTRIES MESSAGE
DELETE UNRECOVERED DIRECTORY ENTRIES
MTR 75
MTR 75
MAG TAPE RELOAD
MTR 77
NO
LOAD ALL OR RESTORE ALL WITH IDS ?
MTR 80
YES
PRINT MYSTRY IDS ON SYSTEM CONSOLE
MTR 77
WRITE ALL ADTS TO DISC
MTR 70
MODIFY PARAMETERS TO REFLECT DIRECTORY FULL
FLAGGED ENTRIES
MTR 50
NO

FLOWCHART 4



FLOWCHART 4
Continued

WARM_START_PROGRAM

EXTERNALS

To attempt to warm start a non-slept system, read the loader into core using the protected disc loader. The lineal disc bootstrap will detect that the system has not been slept and the loader will print the message "SYSTEM NOT SLEPT; FOR WARM START ATTEMPT, LOAD MASTER TAPE AND PRESS RETURN. OTHERWISE MUST RELOAD FROM MAG TAPE. Load the Master Tape on the tape drive and select -0 of the mag tape unit and press return on the system console. The warm start program will be read off the tape and will automatically begin execution.

The program starts by printing "**WARM START PROGRAM**". At the beginning of each major test, warm start will print the test's title. If the test is successful the program will print "OK", otherwise the reason for failure is printed followed by "WARM START FAILURE - ATTEMPT SALVAGE DUMP?". A response of "N-carriage return" or "carriage return" will result in a halt 77B and the system can then be reloaded from mag tape. A "Y-carriage return" response will result in a call to the salvage dump routine which will attempt to retrieve all of the files and programs listed in the directory and dump them to mag tape. Any other response will cause "ILLEGAL INPUT" to be printed followed by a repeat of the question "-ATTEMPT SALVAGE DUMP?"

If the warm start attempt is successful "WARM START CHECK COMPLETE" is printed at the console. Then the system slept flag is set (to slept of course) and control is returned to the intermediate disc bootstrap. Hence the loader is read back into memory and the system slept flag is checked again by the final disc bootstrap. This time the state of the system on disc is "slept" and the system is nearly up. The operator is given the opportunity to reply to the prompt "LOAD OR DUMP COMMANDS?" and then the date and time sequence is executed bring the system up.

The following is a list of the printed test titles and their associated diagnostic messages and their meanings:

I. ADT CHECK-
   - A. BAD TRACK LENGTH - an ADT track length is not a multiple of 3 or an ADT track length does not fall within the range 8192 to 0.
   - B. BAD DISC ADDRESS - the disc address for an ADT track is out of bounds.

II. IDT CHECK-

A. BAD TRACK LENGTH - an IDT track length as positive or not evenly divisible by 12.

B. TOO MANY ENTRIES ON TRACK - the number of entries exceeds 682.

C. BAD DISC ADDRESS - the disc address for an IDT track is out of bounds

D. INCONSISTENT ENTRY - the IDEC table entry's idcode is not the same as the first idcode on the corresponding IDT track.

E. ENTRY OUT OF ORDER - an idcode has been found out of alphabetical order.

F. BAD IDCODE - an impossible idcode has been found (i.e. an idcode >7999).

G. BAD DISC SPACE ENTRY - the amount of disc spaced used for an idcode entry exceeds the amount allowed.

H. BAD TRACK COUNT - the number of IDT track entries in the IDT table does not equal the number indicated by the Equipment Table entry.

III. DIRECTORY CHECK-

A. BAD TRACK LENGTH - a directory track length is positive or not evenly divisible by 12.

B. TOO MANY ENTRIES ON TRACK - the number of entries exceeds 682.

C. BAD DISC ADDRESS - the disc address for a directory track is out of bounds.

D. INCONSISTENT ENTRY - the first 4-word idcode-name sequence on a directory track is not equal to the track's corresponding sequence in the DIREC Table.

E. ENTRY OUT OF ORDER - an idcode-name sequence is out of alphabetical order.

F. BAD PROGRAM STARTING ADDRESS - a program entry has a starting address which does not fall within the range from 2000B to 20000B.

G. BAD PROGRAM LENGTH - a program entry has a length which does not fall within the range from -24000B to 0.

H. BAD FILE LENGTH - a file entry has a length which does not fall within the range from 32767.

I. BAD PSEUDO ENTRY - the directory entry is not the pseudo entry.

J. BAD TRACK COUNT - the number of directory track entries in the DIREC Table dose not equal the number of discs on the system times the number of directory tracks per disc.

IV. DISC SPACE USED CHECK-

A. INCONSISTENT - the total disc space used summed from the Directory Entries.

If a disc error occurs during the warm Start Program testing, the message "DISC ERROR - CORRECT AND RETRY" is printed and the computer will halt with 51B displayed.

INTERVALS

Warm Start functions as follows:

I. ADT-CHECK

A. Check the select code for each of eight possible discs in the core resident ADT Disc Address Table. If the select code is zero, zero the remaining two words of this entry. Otherwise, check the validity of the disc address and ensure that the length of the table is both a multiple of 3 and between 0 and -8192 words long.

II. IDT CHECK

A. Zero IDWRD

B. Get next IDEC table entry.

C. If disc address is zero, go to step G.

D. Increment track counter.

E. Length of track must be negative.

F. Length of track must be a multiple of twelve.

G. Track may contain no more than 682 entries.

H. Read in ID track from disc.

I. First ID on track must equal the corresponding ID in the IDEC table.

J. ID must be greater than IDWRD.

K. Save ID in IDWRD

L. Numeric part of ID must be in the range of 0 to 999.

M. Alpha part of ID must be in the range of A to Z.

N. Add number of blocks used by this ID to a double word quanity.

O. Update pointer to next entry.

P. If there are more entries on this track go to step J.

Q. The track counter must equal the number of ID tracks (NDIRT).

III. DIRECTORY CHECK

A. Set pointer to first entry of DIREC table.

B. Zero IDTTC and last entry encountered.

C. Length of track must be negative.

D. Length must be a multiple of twelve.

E. Number of entries on track can not exceed 682.

F. Read in directory from disc.

G. First four words of track must equal first four words of DIREC.

H. If initial pseudo entry has not been encountered check for it and go to step K.

I. This entry must be greater than last entry encountered.

J. Make last entry encountered equal to this entry.

K. If this entry is a file goto step P.

L. Program started address must be between 2000B and 26000B.

M. Program length must be between -24000B words and -3B words.

N. Convert program length to positive blocks.

O. Go to step Q.

P. Make sure length of file is positive.

Q. Add length in blocks to double word quanity.

R. Advance pointer.

S. If there are any entries left on this track go to step I.

T. Set pointer to next DIREC entry.

U. Increment track counter.

V. If pointer to next DIREC entry is not beyond the end of the table go to step C.

W. Last entry must be the final pseudo entry.

X. Track counter must equal the number of discs times the number of discs times the number of directory tracks per disc.

IV. DISC SPACE USED CHECK

A. The two double word quantities calculated in the IDT check (step N) and the DIRECTORY check (step Q) must be equal.

System Generation

1. Zero IDEC and DIREC tables.

2. Set system identification to blanks.

3. Set number of directory tracks per disc. Set NDIRT equal to one for 7900 loaders. In all other cases, set NDIRT equal to six.

4. Set number of ID tracks (NDIRT) equal to one.

5. Get system identification from operator.

6. Ask operator if configurations options are wanted. If so, solicit disc and format command.

7. Check discs for valid labels. (LDR30)

8. Resume configuration options. From the operator get the following:
   A. mlock and munlock commands
   B. number of directory tracks per disc
   C. number of ID tracks

9. Build ADT for each active disc (LDR40)

10. Claim disc blocks (LDR60)

11. Configure cold dump program.

12. Get the number of ports from the IOP.

13. Load the system and the system library from mag tape. This includes writing the system library to disc.

14. Rewind and unload the master tape.

15. Write the system to disc.

16. Perform the final sequence. (LD101)

DISC RELOAD

1. Check system level code of disc 0.
   A. If system level code of disc 0 is not equal to the system level code of the loader about the load.

2. Load the final disc bootstrap using the intermediate disc bootstrap.

3. Perform the final disc bootstrap.
   A. Read the loader and the system from disc 0
   B. Read each disc's label and verify correct unit number, system identification feature level and system level.

C. Restore configurator linkage (RDISL)
D. If system is not 'slept' allow user to attempt Warm Start
E. Get number of ports from IOP. If this number has changed, return unneeded tracks and claim one swap track for each port.
F. Rewind and unload Master Tape.

4. Processs Load or Dump Commands.

5. Balance directory tracks.

6. Write DIREC table to disc.

7. Zero Fuss Table and write to disc.

8. Perform the final sequence (LDIOI).

MAG TAPE RELOAD

1. Check IOP's date code (CKFC)

2. Check system level code of the first reel. It it is not the same as that of the loader (1B), abort the mag tape reload unless it is a 1A system level code in which case the system will be upgraded to a 1B.

3. Read the equipment table from mag tape and update the mag tape select code in the equipment table.

4. Restore disc linkage (RDISL)

5. Read the IDEC table from mag tape.

6. Read the DIREC table from mag tape.

7. Read the device table from mag tape.

8. Count and save the number of IDs.

9. Count and save the number of directory entries.

10. Zero IDEC and DIREC tables.

11. Zero ADT descriptors.

12. Ask operator if configuration options are wanted. If they are do the following.
A. get new system identification from operator
B. accept disc and format commands.

13. Check discs for valid labels (LDR30)

14. Resume configuration options.
    From the operator get the following:
    A. mlock and munlock commands
    B. number of directory tracks per disc.
    C. number of ID tracks.

15. Build ADT for each active disc (LDR50)

16. Claim disc blocks (LDR60)

17. If configuration options were wanted, ask if the alternate allocation option is wanted.

18. Distribute the ID entries evenly accross all allocated tracks updating IDEC in the process.

19. Distribute directory entries evenly accross all allocated tracks updating DIREC in the process.
    A. Set all real (i.e. not pseudo or not nonshareable device file) directory entries to "must be recovered".
    B. Uncount each nonreal entry from the number of directory entries as counted in step 9.

20. Configure cold dump program.

21. Get the number of ports from the IOP.

22. Read the system binary from mag table.

23. Restore disc linkage (RDISL)

24. Read the system library from mag tape and write all overlays to disc.

25. Write the system to disc.

26. Recover magnetic tape library (MTR)

27. Process load or dump commands.

28. Balance the directory tracks.

29. Write DIREC table to disc.

30. Zero FUSS table and write it to disc.

31. Perform final sequence (LD101)

1. Check the system level code of disc 0. If the system level code of disc 0 is not the same as the system level code of the loader, abort the update.

2. Read the Master Segment Table from block 1 of disc 0 in a buffer. Copy the disc address of each segment from the buffer to the appropriate entry in the loader's Master Segment Table.

3. Read the IDEC and DIREC tables from disc.

4. Read the device table from disc.

5. Read the equipment table from disc.

6. Restore disc linkage (RDISL)

7. Disallow disc reload (should the update fail) by setting the status of the system on disc to "not slept".

8. Return space from the system library to the ADT. (RTADT)

9. Return space from the swap tracks to the ADT. (RTADT)

10. Configure cold dump program.

11. Get the number of ports from the IOP.

12. Load the system and the system library from mag tape. This includes writing the system library to disc. (LDR75)

13. Rewind and unload master tape.

14. Write the system to disc.

15. Process load or dump commands.

16. Balance the directory tracks.

17. Write DIREC table to disc.

18. Zero FUSS table and write it to disc.

19. Perform final sequence (LD101).

DISC ORGANIZATION

DISC ORGANIZATION

The disc space available to the system is determined by the number and type of discs which exist on the system. The discs are divided into 256 word blocks. There are 29592 such blocks on a 7905 disc, 46690 blocks on a 2883 disc and 9744 on a 7900 disc.

The first 4 blocks of each disc are reserved for use by the system. Block 0 is a label, which looks like this:

WORD	
0	
1	"TS"
2	logical disc number
3-7	system identification
8	system level code
9	feature level code
10-30	0
31	checksum of words 0-30
32-127	pre-boot processor

Disc space for system usage is assigned as follows:

Resident	Variable
IDT	32 blocks/track
ADT	32 blocks/track
Directory	32 blocks/track
System library	Variable
Swap Area	42 blocks/track

The resident system is always on logical disc 0. In order to obtain good disc usage and easy expansion of the system library, the disc space occupied by the system library depends on the sizes of the library routines, each of which is either 1 or 2 blocks long. All the system library routines are contiguous to one another on disc.

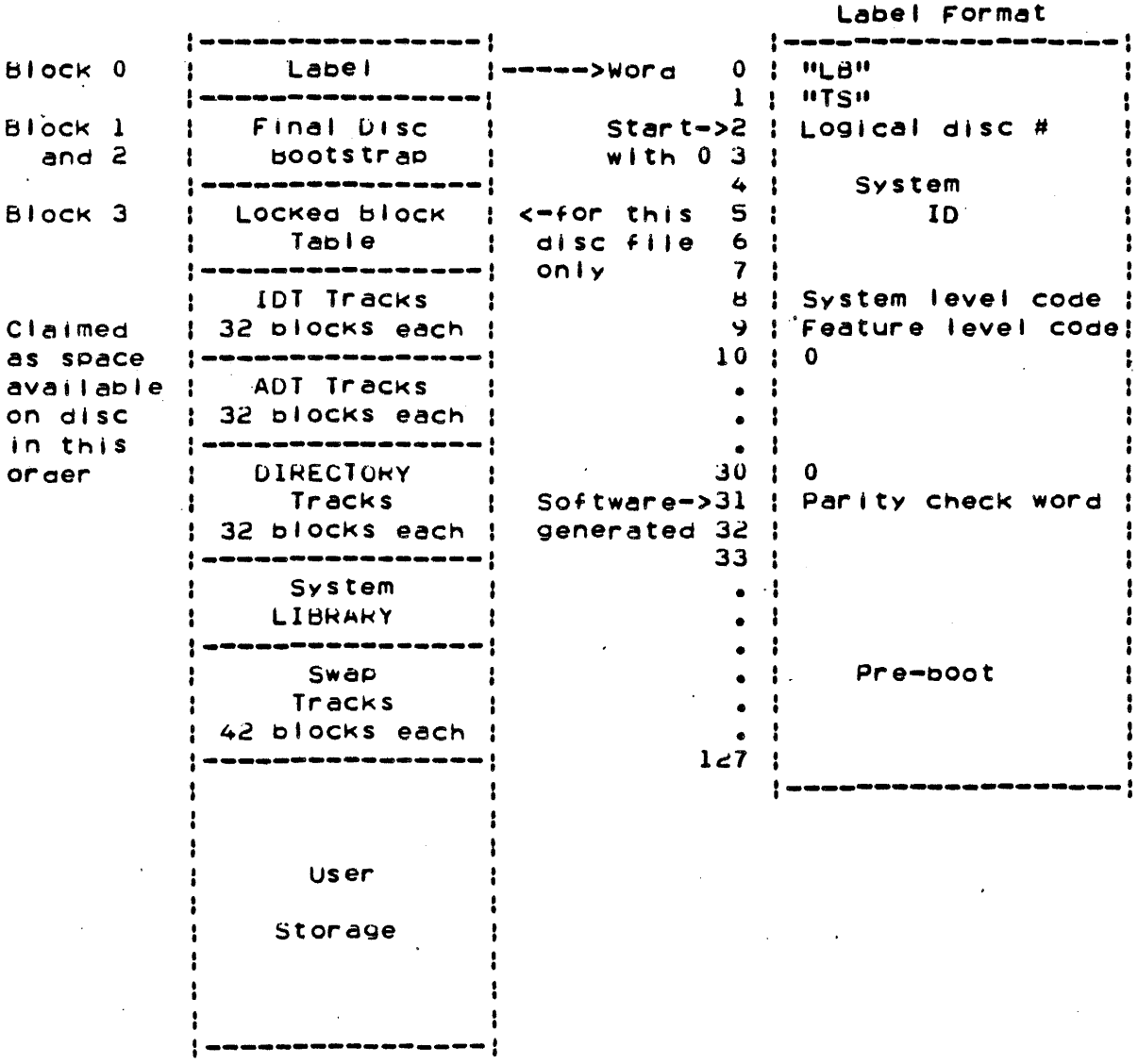
All remaining blocks are available for storage of user programs and files. Programs and files are each required to be stored as contiguous blocks of disc. Since the disc is allocated by blocks, each program may cause part of its last block to be wasted. When a program is stored (by the SAVE routine), it is first decompiled and is stored in that form. Only the encoded text is stored, so that a program may require as little as 3 words of disc space. When a program is stored (by the CSAVE routine) it is saved in a semi-compiled form, i.e. the form it is in after the symbol table is built. Both the encoded text and the symbol table are stored, plus 6 words of necessary information.

Files always occupy an integral number of records (1-32767), each file occupying a contiguous area on the disc. BASIC treats the individual records in the same logical sequence as the physical sequence.

TSB DISC FORMAT

2000 COMPUTER SYSTEM

DISC FORMAT



HP 2883 Disc

Logical Disc Number	First Block
0	0
1	46690
2	93380
3	140070
4	186760
5	233450
6	280140
7	326830

HP 2883 Disc

Cylinder Number	First Relative Block
0	0
10	2300
20	4600
30	6900
40	9200
50	11500
60	13800
70	16100
80	18400
90	20700
100	23000
110	25300
120	27600
130	29900
140	32200
150	34500
160	36800
170	39100
180	41400
190	43700
200	46000

HP 2883 Disc

Cylinder Number	First Relative Block
0	0
1	230
2	460
3	690
4	920
5	1150
6	1380
7	1610
8	1840
9	2070

HP 2883 Disc

Head Number	First Relative Block
0	0
1	11 1/2
2	23
3	34 1/2
4	46
5	57 1/2
6	69
7	80 1/2
8	92
9	103 1/2
10	115
11	126 1/2
12	138
13	149 1/2
14	161
15	172 1/2
16	184
17	195 1/2
18	207
19	218 1/2

HP 2883 Disc

Sector Number	Block
0	0
1	1/2
2	1
3	1 1/2
4	2
5	2 1/2
6	3
7	3 1/2
8	4
9	4 1/2
10	5
11	5 1/2
12	6
13	6 1/2
14	7
15	7 1/2
16	8
17	8 1/2
18	9
19	9 1/2
20	10
21	10 1/2
22	11

HP 7900 Disc

Logical Disc Number	First_block
0	0
1	9744
2	19488
3	29232
4	38976
5	48720
6	58464
7	68208

HP 7900 Disc

Cylinder Number	First Relative Block
0	0
1	48
2	96
3	144
4	192
5	240
6	288
7	336
8	384
9	432

HP 7900 Disc

Cylinder Number	First Relative Block
0	0
10	480
20	960
30	1440
40	1920
50	2400
60	2880
70	3360
80	3840
90	4320
100	4800
110	5280
120	5760
130	6240
140	6720
150	7200
160	7680
170	8160
180	8640
190	9120
200	9600

HP 7900 Disc

Head_Number	First Relative_Block
0	0
1	12
2	24
3	36

HP 7900 Disc

Sector_Number	Block
0	0
1	1/2
2	1
3	1 1/2
4	2
5	2 1/2
6	3
7	3 1/2
8	4
9	4 1/2
10	5
11	5 1/2
12	6
13	6 1/2
14	7
15	7 1/2
16	8
17	8 1/2
18	9
19	9 1/2
20	10
21	10 1/1
22	11
23	11 1/2

HP 7905 DISC

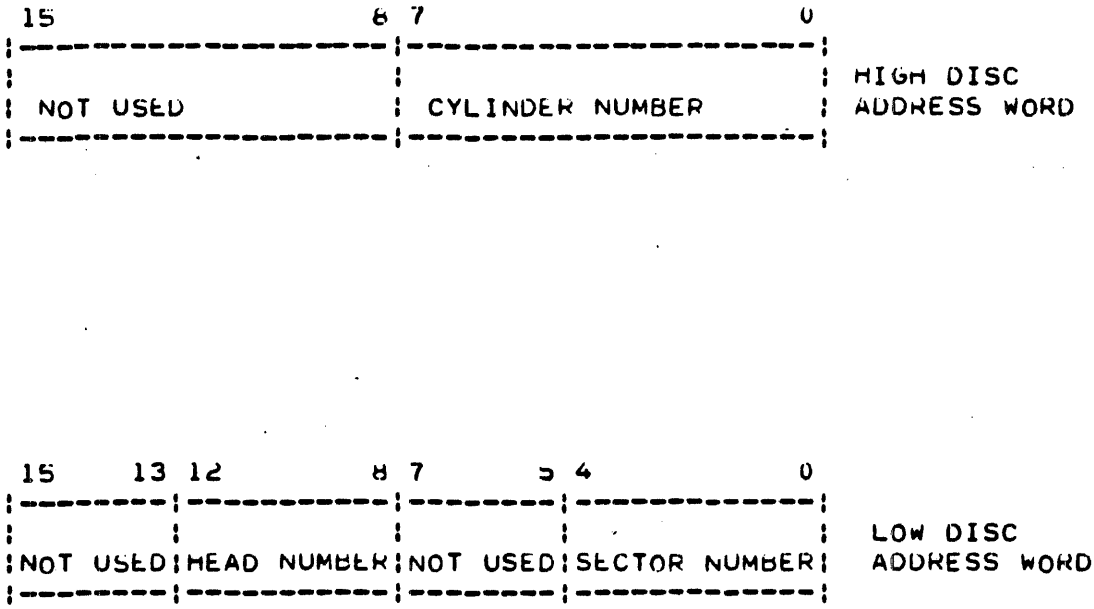
Logical Disc Number	First Block
0	0
1	29592(10)
2	49184(10)
3	88776
4	118368(10)
5	147960
6	177552
7	207144
8	236736(10)

Cylinder Number	First Relative Block
0	0
1	72
2	144
3	216
4	288
5	360
6	432
7	504
8	576
9	643
10	720
20	1440
30	2160
40	2880
50	3600
60	4320
70	5040
80	5760
90	6480
100	7200
110	7920
120	8640
130	9360
140	10080
150	10800
160	11520
170	12240
180	12960
190	13680
200	14400
300	21600
400	28800

HP 7905 Disc Relative Block Numbers

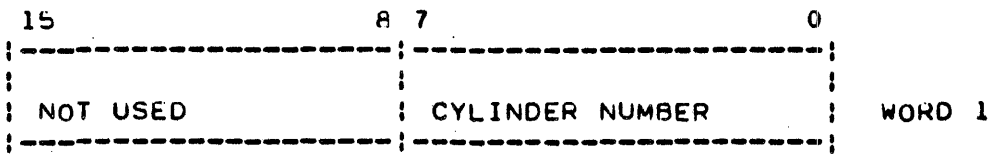
Head_Number	First Relative_Block
0	0
1	24
2	48

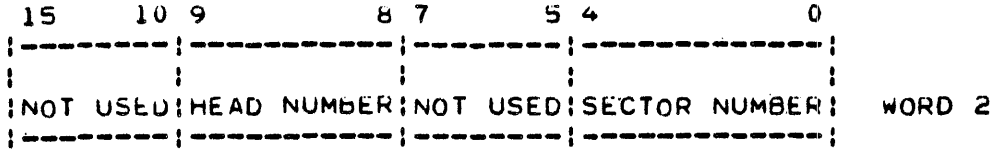
Sector Number	First Relative_Block
0	0
1	0 1/2
2	1
3	1 1/2
4	2
5	2 1/2
6	3
7	3 1/2
8	4
9	4 1/2
10	5
11	5 1/2
12	6
13	6 1/2
14	7
15	7 1/2
16	8
17	8 1/2
18	9
19	9 1/2
20	10
30	15
40	20



HARDWARE
DISC ADDRESSING
FORMAT FOR HP 2883
(32 bit Data Channel Word)

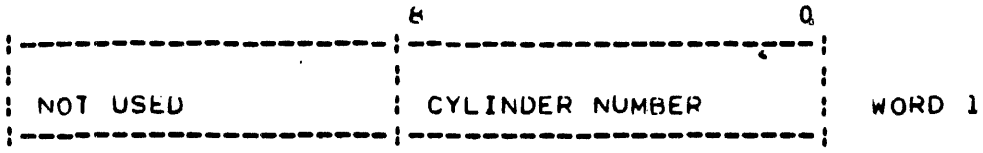
(DRIVE SELECT CODE IS
CONTAINED IN COMMAND WORD)

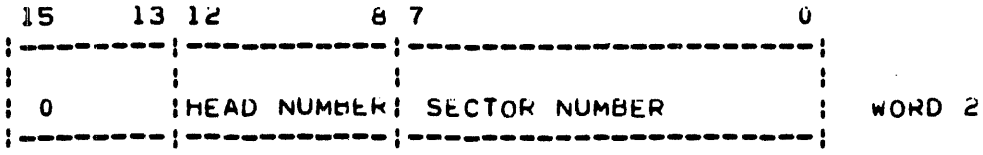




HARDWARE
DISC ADDRESSING
FORMAT FOR HP 7900
(32 Bit Data Channel Word)

(DRIVE SELECT CODE IS
CONTAINED IN COMMAND WORD)





HARDWARE
DISC ADDRESSING
FORMAT FOR HP 7905
(32 bit Data Channel word)

(DRIVE SELECT CODE IS
CONTAINED IN COMMAND WORD)

DISC ERROR ROUTINES

System Error Recovery

System error recovery is predicated on three different types of failure: 1) inability to read or write a portion of the disc or: 2) total system failure due to a complete disc failure, processor malfunction (especially memory parity errors), software failures, etc; or 3) IOP failure. Recovery from disc failures entails calling of a routine dependent on the type of entity involved (user swap track, directory track, IDT track, ADT track, command overlay, etc.). Recovery from a total system failure entails use of the warmstart program (detailed in the section of this document concerning the loaders) in conjunction with some prophylactic measures always taken by the system. Recovery from an IOP failure entails the use of the IOP PANIC routine.

Disc Failures

An unrecoverable disc error while swapping a user in or out of memory causes a call to the PORT ZAPPER routine (PTZAP). An error in reading an overlay causes a call to Emergency Sleep (SICKP). Inability to read a system track (directory, IDT, ADT, etc.) typically causes a call to SICKP. Occasionally, however, the message DISC ERROR, CAN'T DO IT is output when the error in reading the system track is related to a minor command. An error in writing a system track causes a call to the SALVAGE routine (SLVAG). Any non-recoverable read/write failures (e.g., port zapper and salvage are disc resident: a failure in reading them into memory is a non recoverable read failure) causes a call to the system death rooutine (DEADP). The routines port zapper, salvage, emergency sleep, and dead are discussed below.

Total System Failures

In order to allow recovery from a total system failure, the tables on the disc must be up to date and consistent. This is generally the case with the exception of the DIREC and IDEC memory resident tables and the equipment table which contains news about the system ADT entries. Therefore, each time a directory track is shortened or lengthened, an ID track is shortened or lengthened, or a system track is moved (see salvage, below) the equipment, the DIREC and the IDEC tables are updated on disc. This preventive measure ensures a good chance that all disc resident system tables are up-to-date and consistent. The warmstart program, which would be used in case of total system failure, thus has a good chance of finding the disc in good shape.

IOP_Failures

When the IOP fails, users are instructed to execute the IOP PANIC routine. This routine sends to the SP a buffer empty notification for each port. This ensures that the SP does not hang because of a buffer full bit set in a user's ?FLAG word. In addition, it responds to all requests from the SP with a 0. If any user required input, this 0 fakes a response (all system code which receives characters from the IOP treat a 0 as a carriage return - a null (ASCII 0) is never legal as an input character since nulls are stripped by the IOP). This 0 also satisfies any other SP requests (e.g., system sends transfer input buffer and IOP responds with 0. System interprets this as zero length transfer and is satisfied). Finally, the IOP is listening for send cold dump (KSN) commands so a cold dump can be performed when the IOP is in PANIC.

The net result of PANIC is to get the system processor to a point where a SLEEP (or HIBERNATE) command can be entered at the system console. The constant HFE's and zero responses from the IOP allow the system to be successfully slept.

Disc Error Routines

The dead routine informs all users that the system is going down, clears all non-shareable devices (to free the IOP), tells the IOP that the system is going down, informs the console operator of impending death, and, finally, halts the system.

The emergency sleep routine does the same thing as DEAD with two exceptions: memory resident tables are updated on the disc and the system slept flag is set to slept. Recovery is guaranteed from an emergency sleep.

Both jettison port (PTZAP) and salvage reside on the disc. They are read into the area between the user area and the disc driver. If a failure occurs when reading, we call DEAD.

Jettison port kicks a user off the system, making his port unavailable for further use. It first removes the user from the scheduler queue and clears MAIN in case it was set. The user's ?FLAG word is cleared and non-shareable devices are released. The user's area of the FUSS table is cleared to release any files he might have been using. Finally, we inform the user of the loss of his port, set the port's status to unavailable, tell the system operator of the error, and return to normal system operation minus one port.

Salvage is called on a failed write of a system table. It assumes that the track is in memory starting at LIBUS and that WORD contains the length of the track. It also assumes that STDAP contains the disc address where the track has failed to have been written. In the worst case an entire track (8192 words) and a table (such as the one created by the system operator's PURGE or MLOCK commands) is in memory. We are always guaranteed that the space occupied does not extend past 240000B. Execution is as follows:

1. Read approximately 1K (776B words (3 does not divide evenly into 1K)) chunks of the ADT into 240000B (throw away the ADT if we can't read it) and search for a free area large enough to hold the track. If no space is found and track to be salvaged was an ADT, throw it away and exit. Otherwise we must die because we can't move the track.

2. If space is found, substitute the new disc address in the memory resident table that points to the respective track.

3. Write track to its new location on the disc (if we can't, we die - goto DEATH).

4. Remove the space from the ADT for the new area just claimed, tell the operator that a track was moved, and return to normal processing.

COLD DUMP PROGRAM

Preparation for running the cold dump program

1. If the I/O processor is halted, note the register values and start it at "Panic" (location 2000B .).

2. Halt the main processor and record the value of the P-register.

3. Mount a mag tape with a write ring and select unit 0.

4. Set the P-register to 77000B (the starting location of the memory resident bootstrap that reads the cold dump program from disc.

   The cold dump bootstrap performs the following functions sequentially.

   1. turn off the interrupt system
   2. save A,B, E and 0 registers
   3. disable power fail recovery
   4. read the final disc bootstrap in memory starting at location 40000B.
   5. get the disc address and size of the Cold Dump Program from the Master Segment Table portion of the final disc bootstrap.
   6. read the Cold Dump Program into memory starting at location 40000B.
   7. jump to the Cold Dump Program.

   The cold dump program is assembled at location 24000B but is read into memory at location DSCDA (currently location 40000B).

   The cold dump program performs the following tasks sequentially:

   1. Checks that unit 0 is on line.
   2. Checks that the mag tape has a write ring.
   3. Rewinds tape.
   4. Writes cold dump label.
   5. Writes register values (A,B,E and 0).
   6. Writes system processor memory.
   7. Reads the final disc bootstrap leads into memory at location 32000B. The MST portion is needed as it contains the disc addresses of the disc resident portion of the system. The initial time the final disc bootstrap was read into memory, it was overlayed by the cold dump program after the disc address and length of the cold dump program was extracted. This

minimizes the amount of memory that gets clobbered prior to being dumped to mag tape.

8. If possible (if the IOP is "listening"), the IOP memory is dumped in chunks into a 127 word system processor buffer. After each transfer is completed, the buffer is dumped to mag tape.
9. Read system disc driver into memory.
10. Restore disc driver:
    A. Clear MBUSY
    B. Suppress disc error messages
    C. Restore pointer to the MHTBL
    D. Restore disc driver interrupt points
    E. Set interrupt locations for all discs.
11. Dump swap tracks:
    A. read system-loader linkage table from disc. On failure, forget about dumping swap tracks.
    B. From the linkage area get the maximum length of a user swap track, starting core address of user swap area, length of a TTY table, and the pointer to the disc address of the swap track for port 0.
    C. Dump all swap tracks, stoping after 32 have been dumped or a disc address of zero is encountered.
12. Dump library sizes table and FUSS table.
13. Dump directory tracks stopping after 80 tracks have been dumped or a disc address of zero is encountered.
14. Dump ID Table, stopping after 10 tracks have been dumped or a disc address of zero is encountered.
15. Dump Available Disc Table, stopping after 8 tracks have been dumped or a disc address of zero is encountered.
16. Terminate Cold Dump:
    A. Write 8 end-of-file marks
    B. Issue a rewind/standby to mag tape unit 0.
    C. Halt 77B

The following errors are correctable by correcting the problem and then pressing 'RUN'. Other errors are not recoverable.

Errors

HLT 22B - DISC 0 NOT READY
HLT 33B - NO WRITE RING ON MAG TAPE
HLT 44B - MAG TAPE UNIT 0 IS NOT ON-LINE
HLT 55B - MAG TAPE BAD OR TOO SHORT

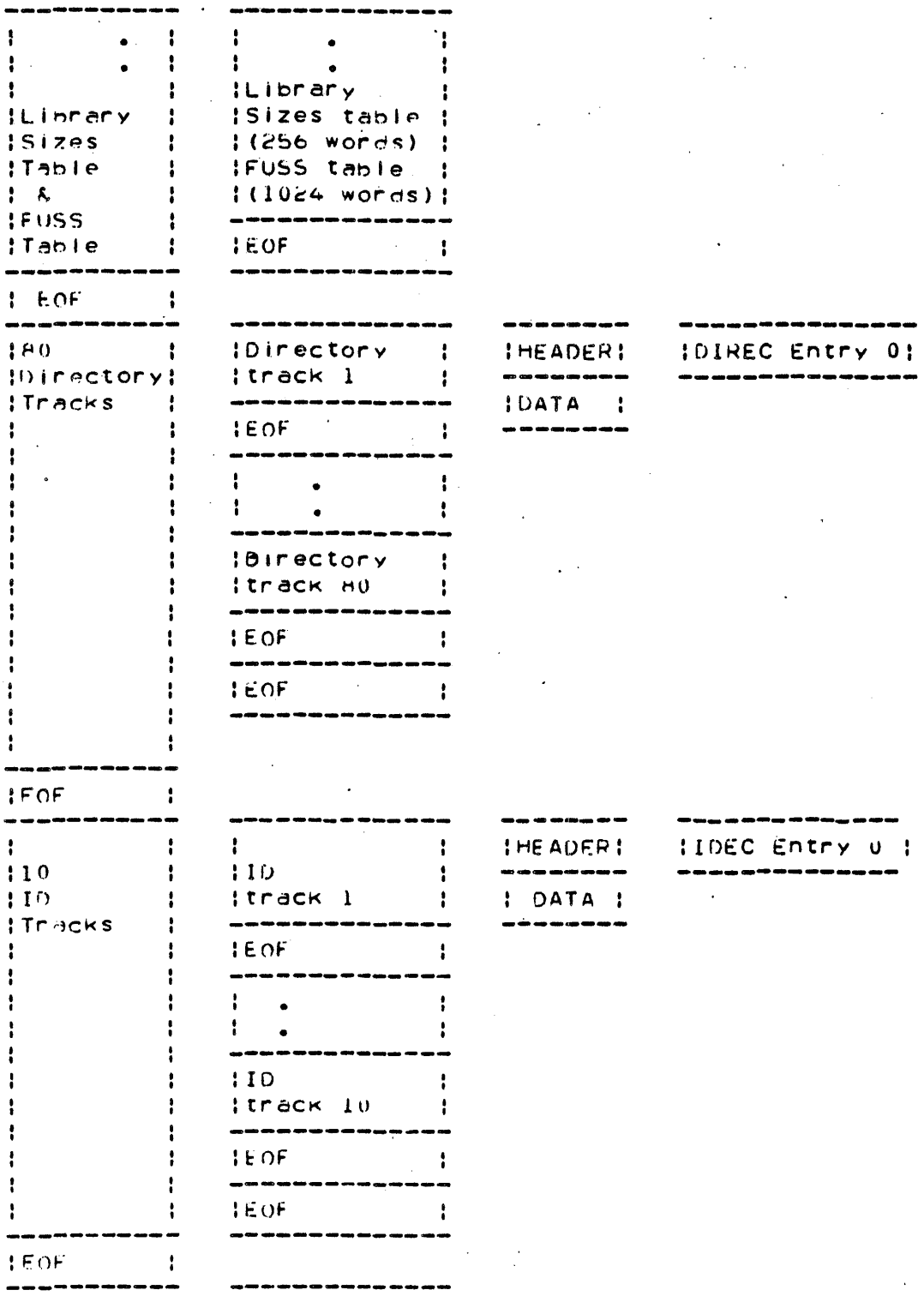
COLD DUMP TAPE FORMAT

The cold dump tape is written in variable length records with a maximum record length of 127/10 words. The tape is written

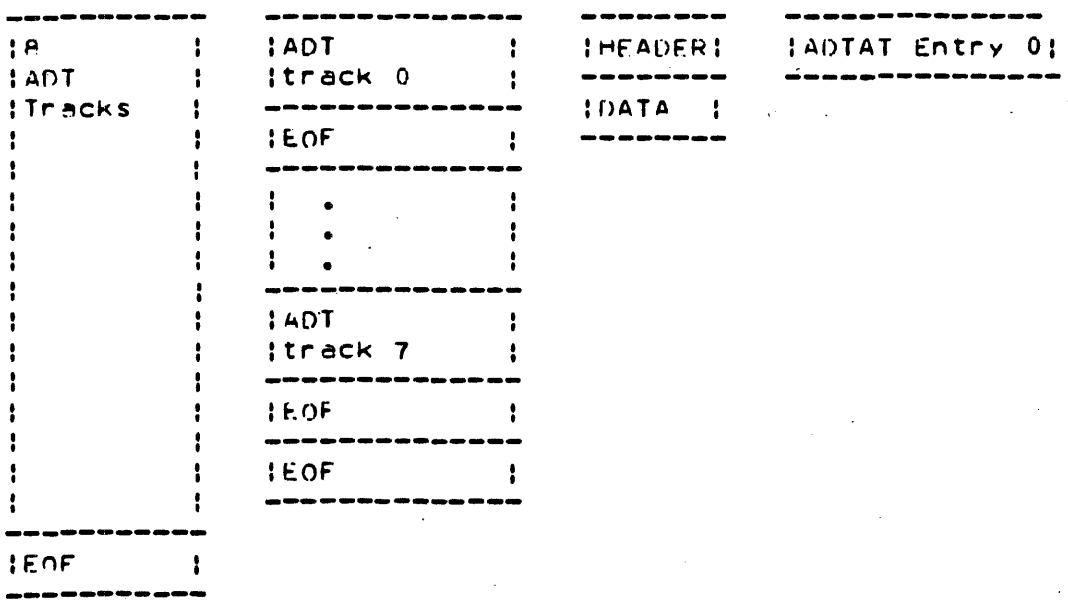
in the following format.

```
                                        CO
Label                                   LD
          A                             DU
Registers B                             MP
          EXXXX0      (Extend/          5000
System                Ovfl)                 10

Processor   loc 2

Memory
            loc 3

I/O         loc 77777

Processor

Memory
            loc 0
EOF
            loc 1
             .
Swap         .
             .
Tracks
            Swap        HEADER    1st 4
            Track        DATA     words
             0                    of TTY
                                  Table
            EOF
            Swap
            Track 1
            EOF
             .
             .
             .
            Swap
            Track 31
EOF         EOF
  .         EOF
  .
  .
```

COLD DUMP TAPE FORMAT (cont.)

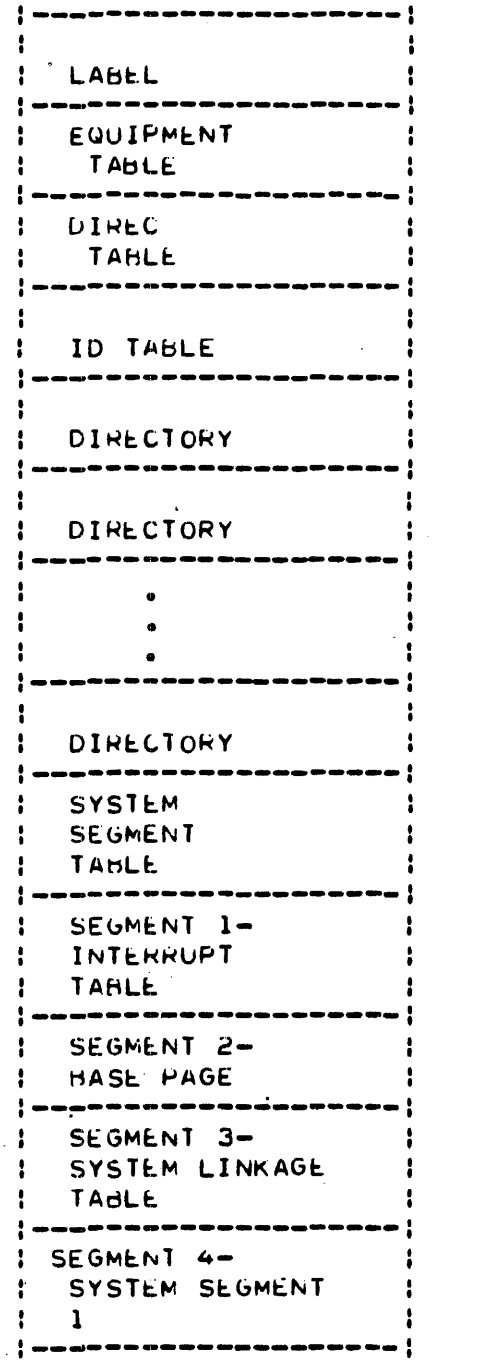


COLD DUMP TAPE FORMAT (cont.)



MAGNETIC TAPE FORMATS

2000 COMPUTER SYSTEM
SLEEP AND HIBERNATE
TAPE FORMATS



first word of first record only is the record word count (negative) placed there by the Mag Tape driver

Format the same as DUMP TAPE format

Lines indicate Record Mark

Dumped in 1024(10) word records, up to 80(10) records

Dumped in 1024(10) word records, up to 640(10) records

Last Directory record is <= 2000(8) words and >= 12 words

From loader MLTBL to MLTBE -- defines segments yet to be dumped

Core locations 2 to 27(8)

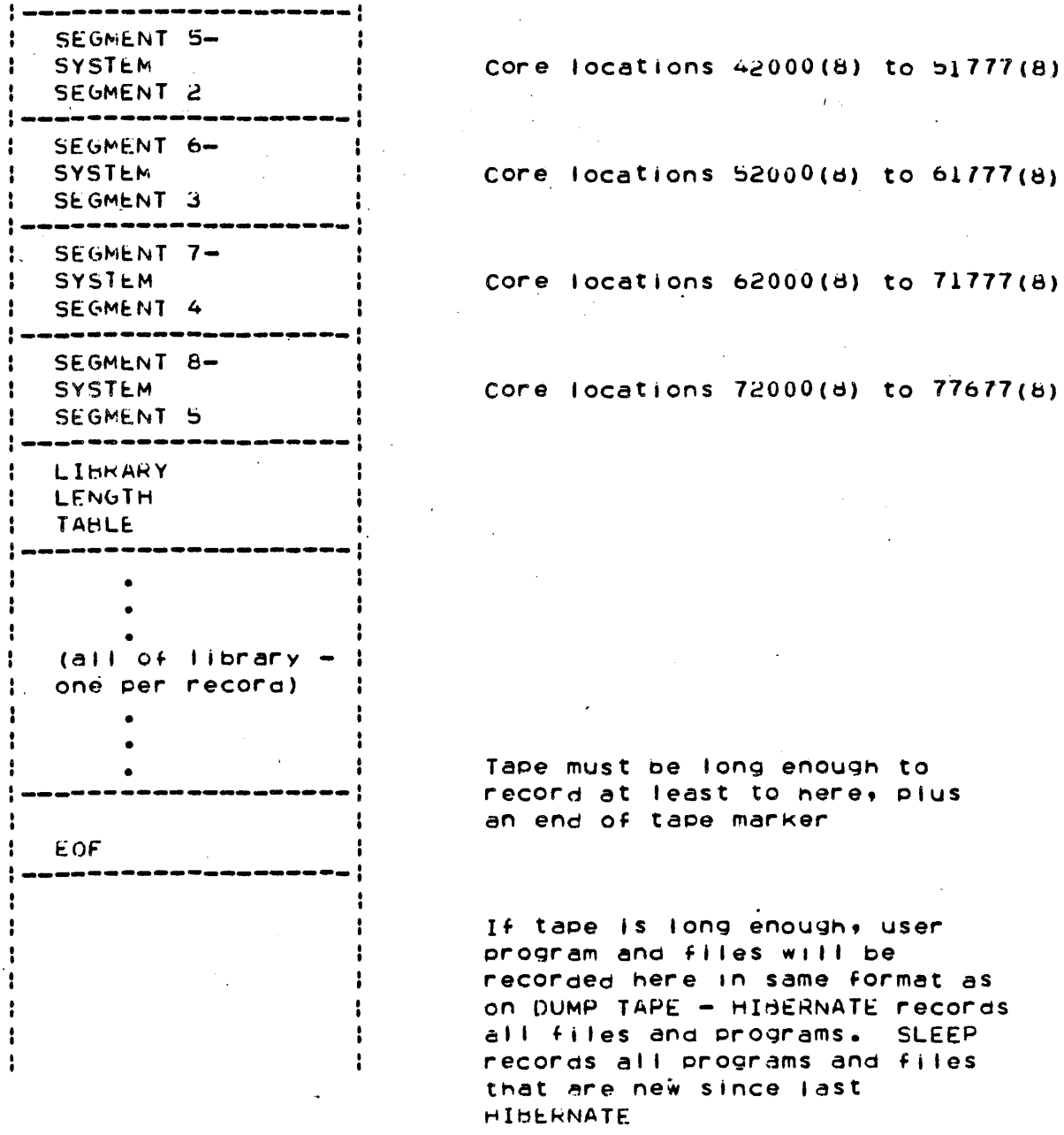
Core locations from end of Equipment Table to 1777(8)

Core locations containing loader - system linkage info-2002(8) to 2015(8)

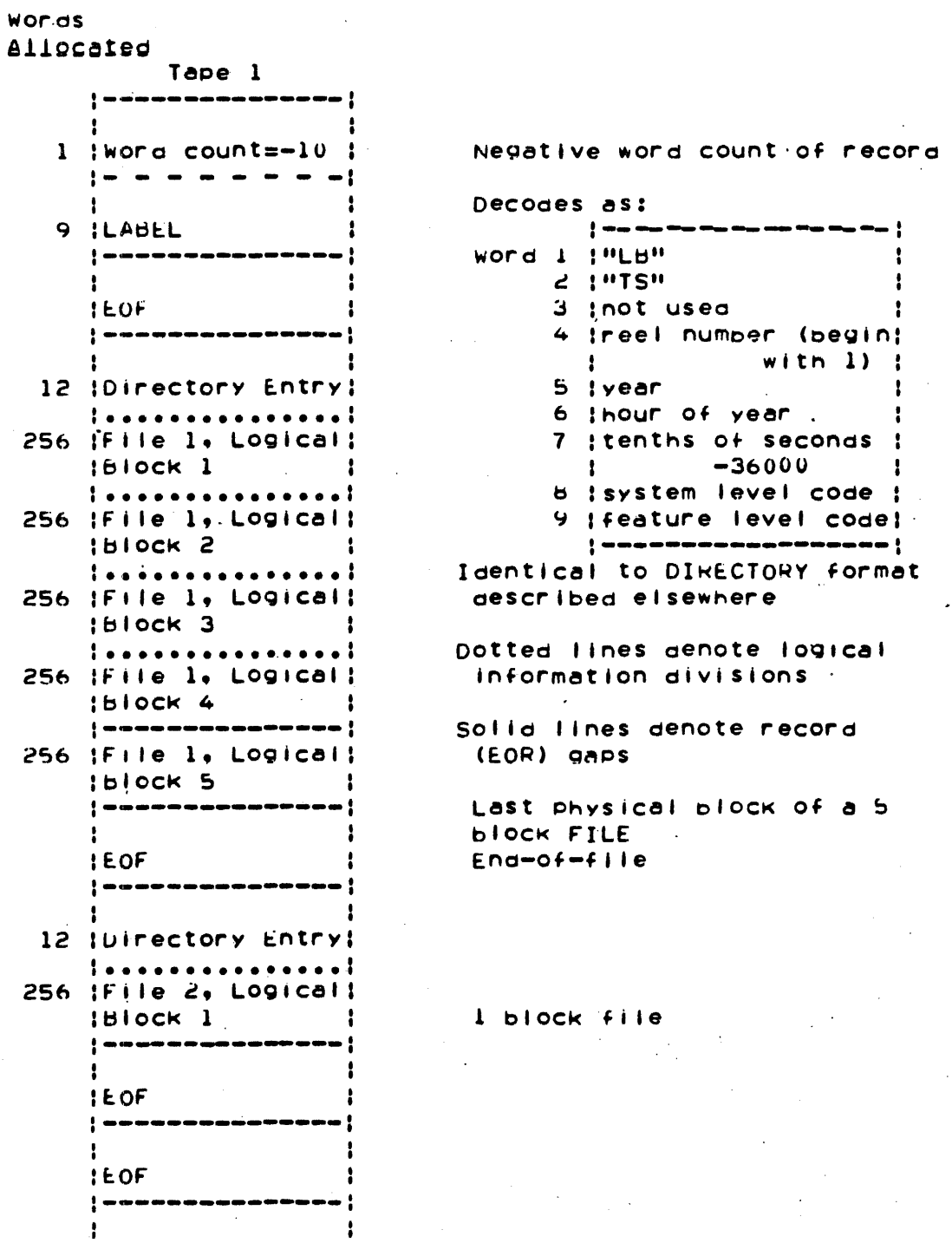
Core locations from end of DIREC Table to 41777(8)

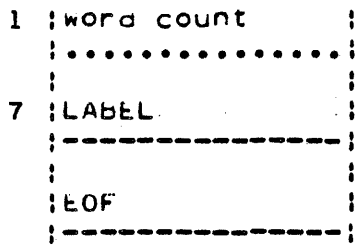
(Continued on Page 2)

2000 COMPUTER SYSTEM
SLEEP AND HIBERNATE
TAPE FORMATS
(Page 2)



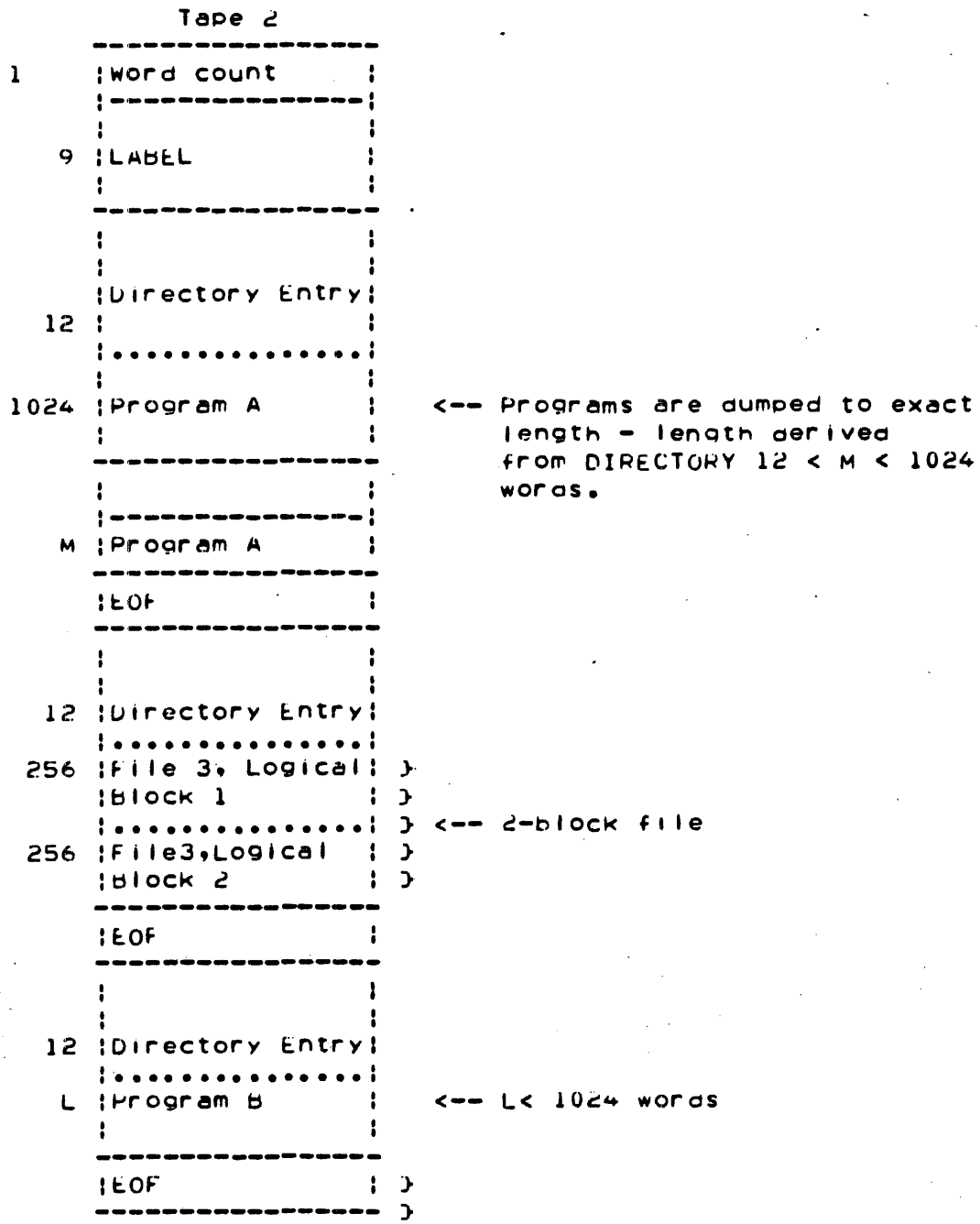
2000/COMPUTER SYSTEM
DUMP TAPE
FORMAT EXAMPLE 1

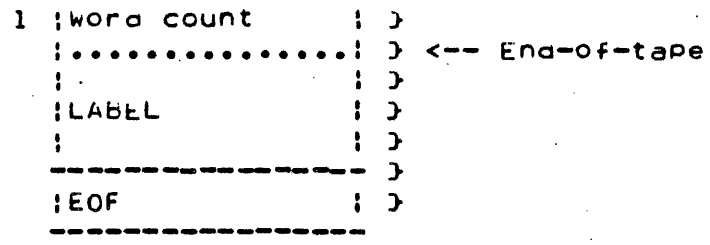




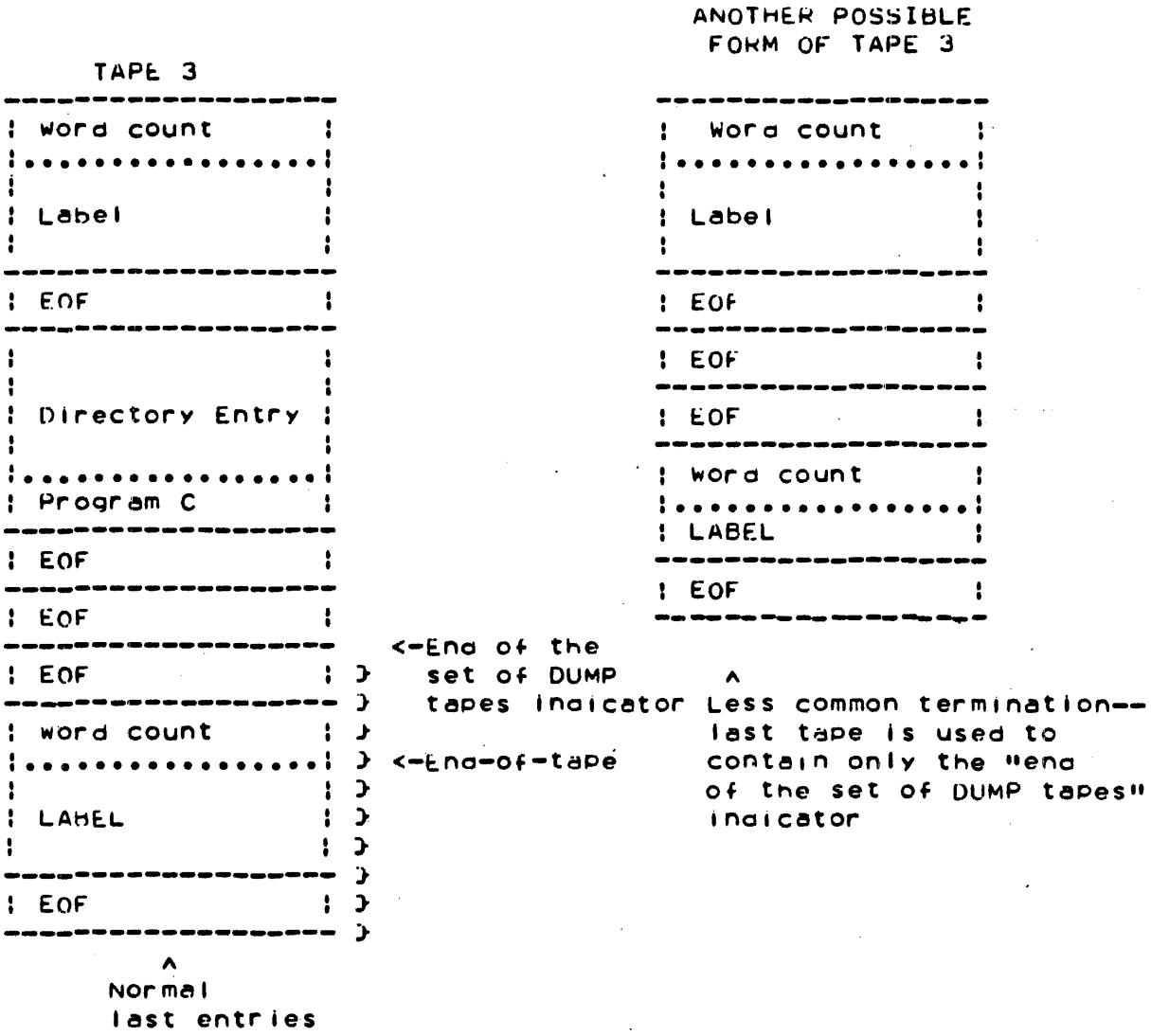
End-of-tape, but not end of entire dump

2000/COMPUTER SYSTEM
DUMP TAPE
FORMAT EXAMPLE 2

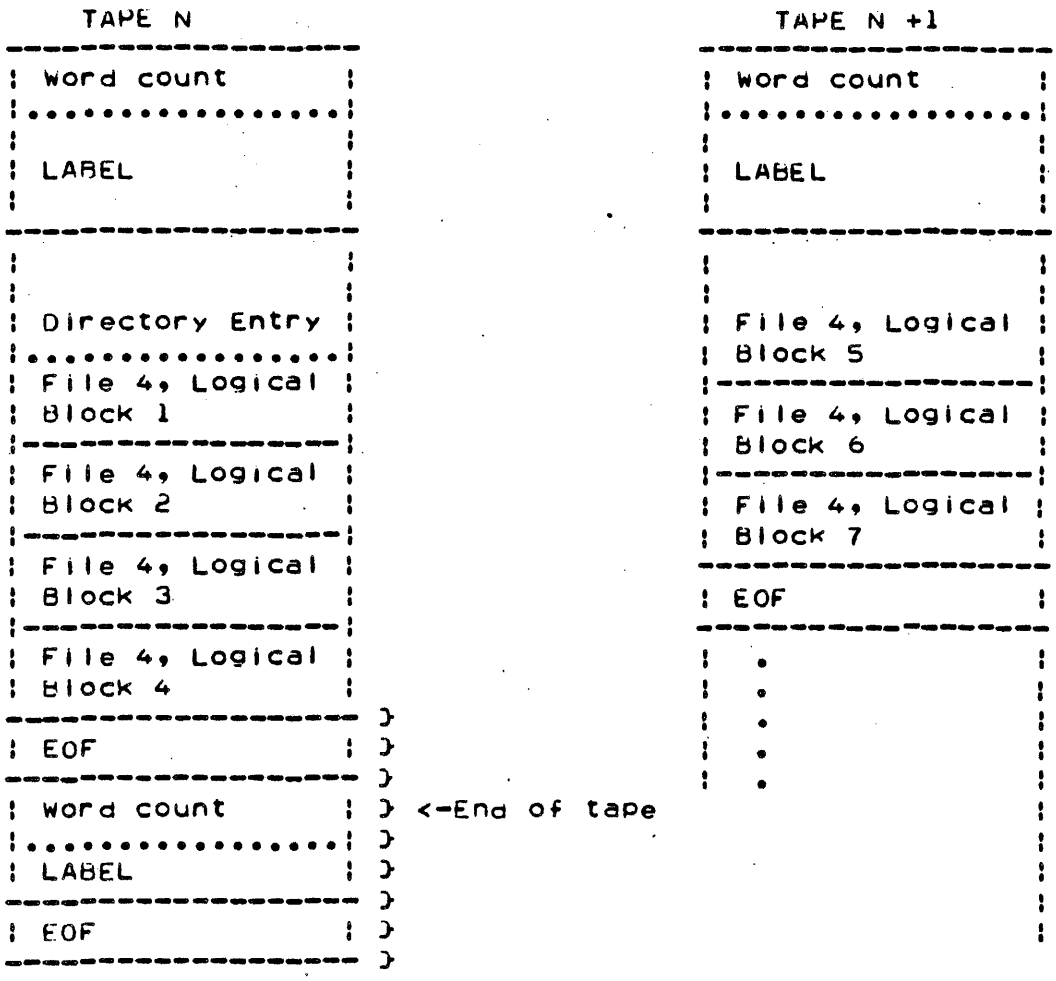



1
word count
LABEL
EOF
<-- End-of-tape

2000/COMPUTER SYSTEM
DUMP TAPE
FORMAT EXAMPLE 3



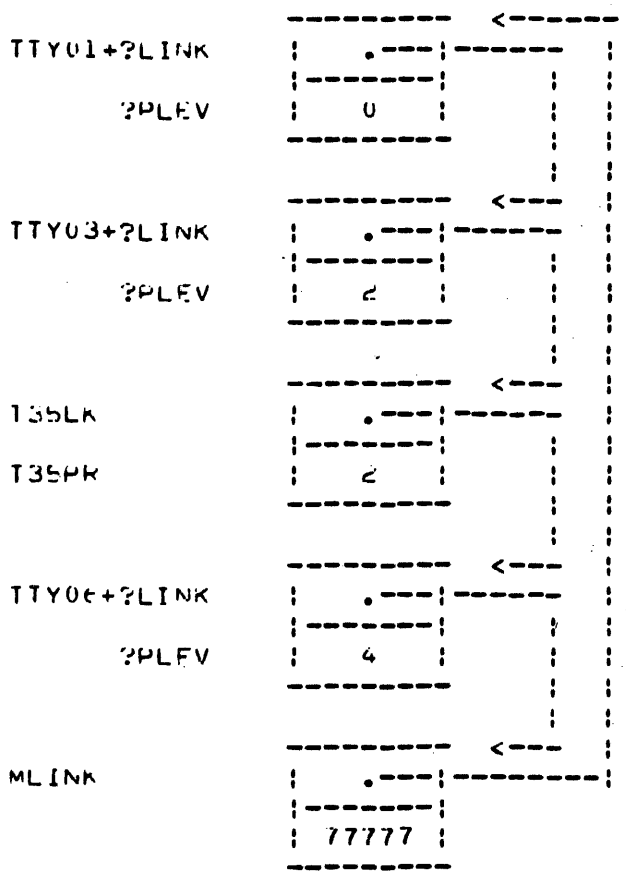
2000/COMPUTER SYSTEM
DUMP TAPE
FORMAT EXAMPLE 4



SCHEDULING

SCHEDULING

The basic philosphy of the TSB scheduling algorithm is to provide short response times for short, interactive jobs at the possible cost of delays in longer running jobs. The implementation of this involves a queue of jobs to run which is ordered according to a piority scheme. The queue is a linked list of from 1 to 34 entries, each entry pointing to the next entry, and the last entry pointing back to the first. The 34 possible entries in the queue are the 32 user LINK entries, a LINK word in a truncated TELETYPE table reserved for the system console, and a queue head. The queue head consists of the locations MLINK (0:2), and is always in the queue. The queue head has a priority of 77777(8), which is stored in location MLINK+2, and it is always the last entry in the queue. As an example of how this works, assume that users 1, 3 and 6 are on the queue in that order and so is the system console, in a position between users 3 and 6. Then the queue will have the following appearance:



Since the MLINK entry is always the last entry on the queue, MLINK+1 is a pointer to the first entry, which in this case is TTY01. In the case of an empty queue, MLINK+1 will point to itself, i.e., CONTENTS(MLINK+1) = CONTENTS(MLINK). Each entry on the queue has a priority no less in numerical value than that of the one its points to. When an entry is to be added to the queue, it is assigned a priority according to the rules described below, and then INSEQ is called. INSEQ computes a new priority using the following formula:

New priority = -(old priority)*(number of ports logged on +1)/4

The queue ordering is preserved by inserting the new entry just ahead of the first entry with a lower (more negative) priority number. Note that when the first entry in the queue has priority 0, it will maintain that position until it is removed from the queue entirely. To ensure that all jobs will eventually reach the top of the queue, the priority of the entry just behind the new entry is incremented by one. Note that by using the number of ports logged on the system as a factor in determining priority, we base the speed at which jobs reach the top of the queue on the system load.

The following rules are used to assign (and reassign) priorities by routines which call INSEQ:

1. Upon first entering the queue, jobs are assigned priorities as follows:

   SYNTAX lines and jobs returning from I/O suspend: 0
   BASIC commands (RUN, LIST, PUNCH, LOAD) : 1
   Commands for disc-resident routines (GET, BYE, Etc.): 2

2. Priorities of jobs are reassigned in the following way:

   Jobs of priority 2, when they reach the top of the queue, are reassigned priority 0.

   RUN jobs, when they exceed their time slice, are reassigned priority 4, and repositioned in the queue according to that priority. Each RUN job is assigned a time slice of two seconds, and if it exhausts that it, is assigned another. When executing a <CHAIN statement>, a <SYSTEM statement>, a <CREATE statement>, a <LOCK statement>, an <UNLOCK statement>, a <FILES statement>, or an <ASSIGN statement>, a RUN job is reassigned a priority of 0.

The CREATE command is reassigned a priority of 4 when it is suspended after writing file marks in 400 blocks.

After an abort during program execution a user is re-assigned a

priority of 0 to run the routine which updates the last change date for files.

LIB points to the locaion in the COMTABLE of the disc address of the library routine in core. LIB = 0 when none is present.

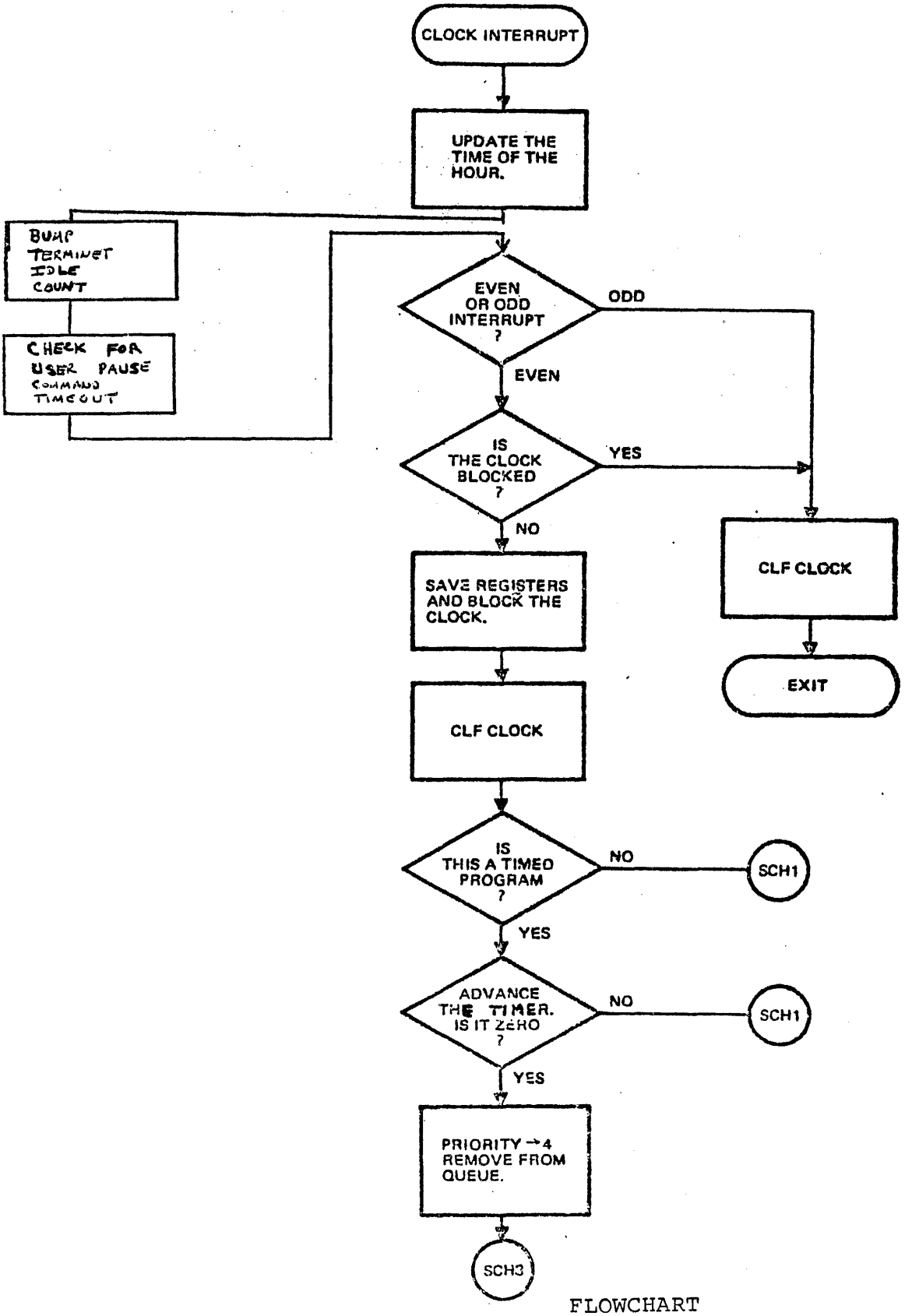
The following conditions must exist for the scheduler to permit execution:

A) for Syntax and BASIC commands;
   MAIN set to point to correct user table

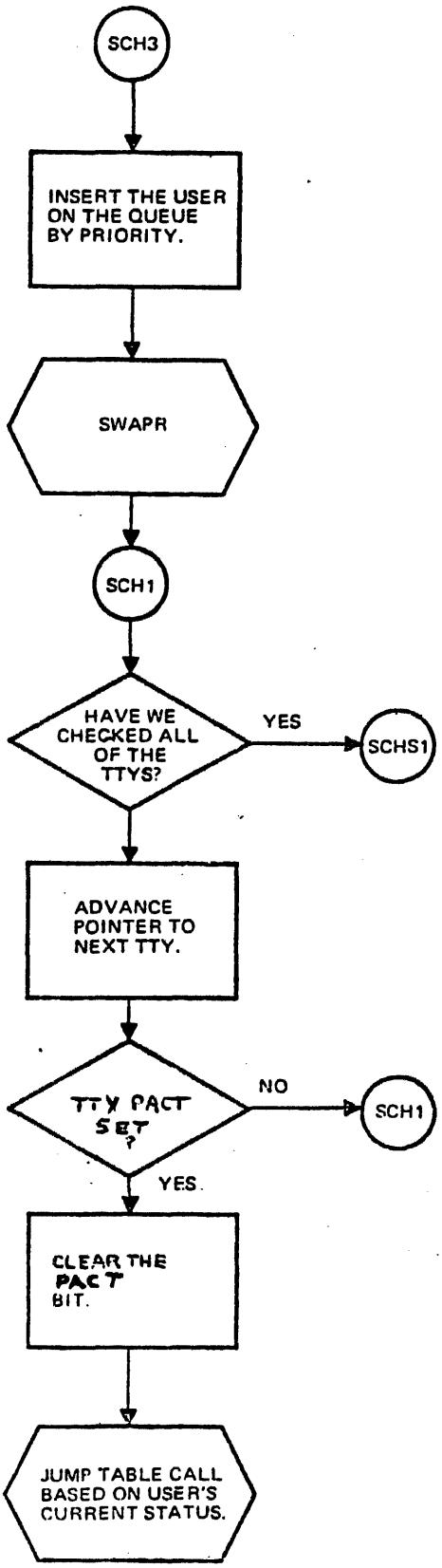
B) for disc resident commands:
   MAIN = 0
   LIB set to correct disc resident routine

The scheduler routine SWAPR is responsible for creating these conditions, and makes its decisions according to the values of MAIN, LIB, and the entry on top of the queue.

SCHEDULER

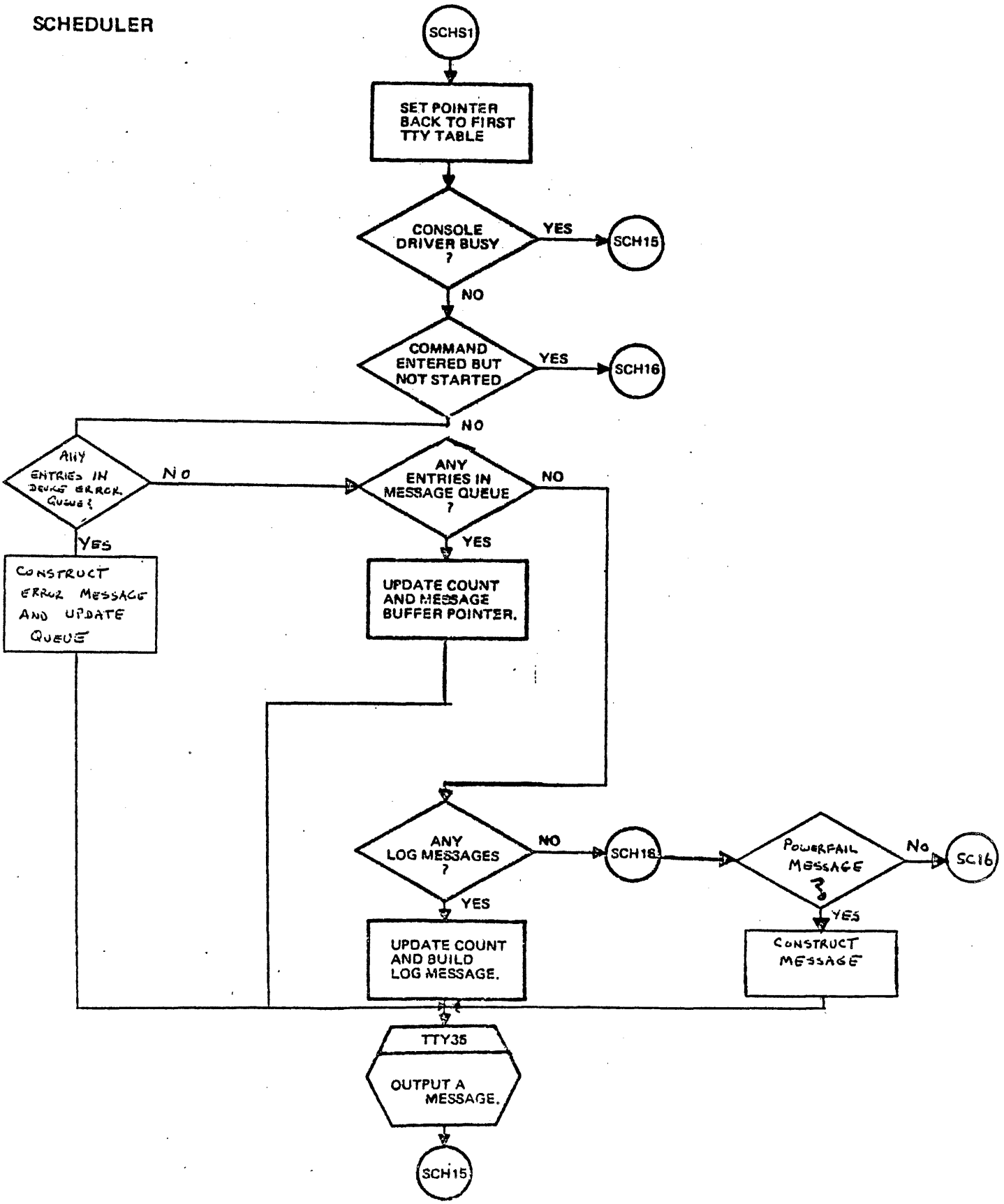

CLOCK INTERRUPT
UPDATE THE TIME OF THE HOUR.
BUMP TERMINET IDLE COUNT
CHECK FOR USER PAUSE COMMAND TIMEOUT
EVEN OR ODD INTERRUPT ?
ODD
EVEN
IS THE CLOCK BLOCKED ?
YES
NO
SAVE REGISTERS AND BLOCK THE CLOCK.
CLF CLOCK
CLF CLOCK
EXIT
IS THIS A TIMED PROGRAM ?
NO
SCH1
YES
ADVANCE THE TIMER. IS IT ZERO ?
NO
SCH1
YES
PRIORITY → 4 REMOVE FROM QUEUE.
SCH3

SCHEDULER



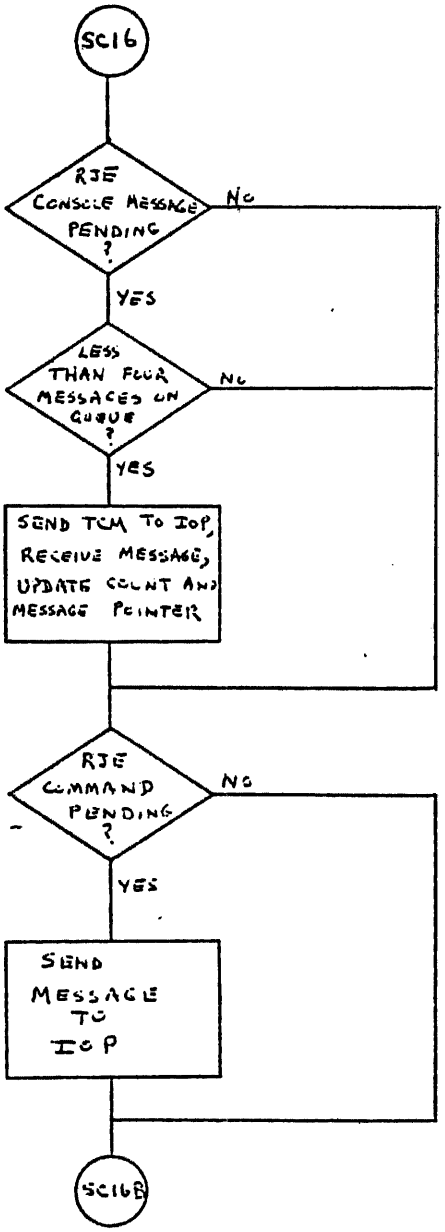
FLOWCHART
2 of 13

SCHEDULER



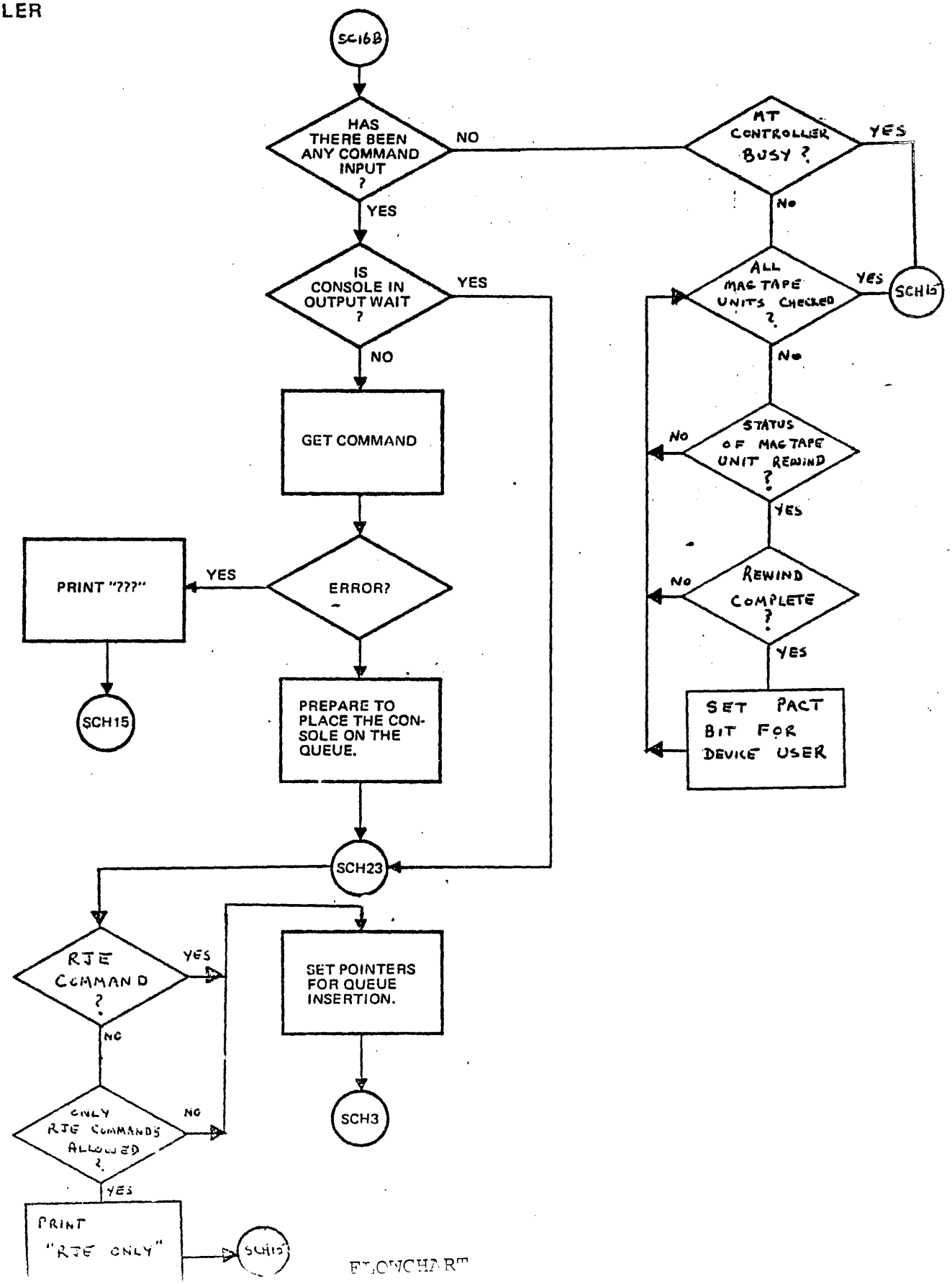
FLOWCHART

SCHEDULER



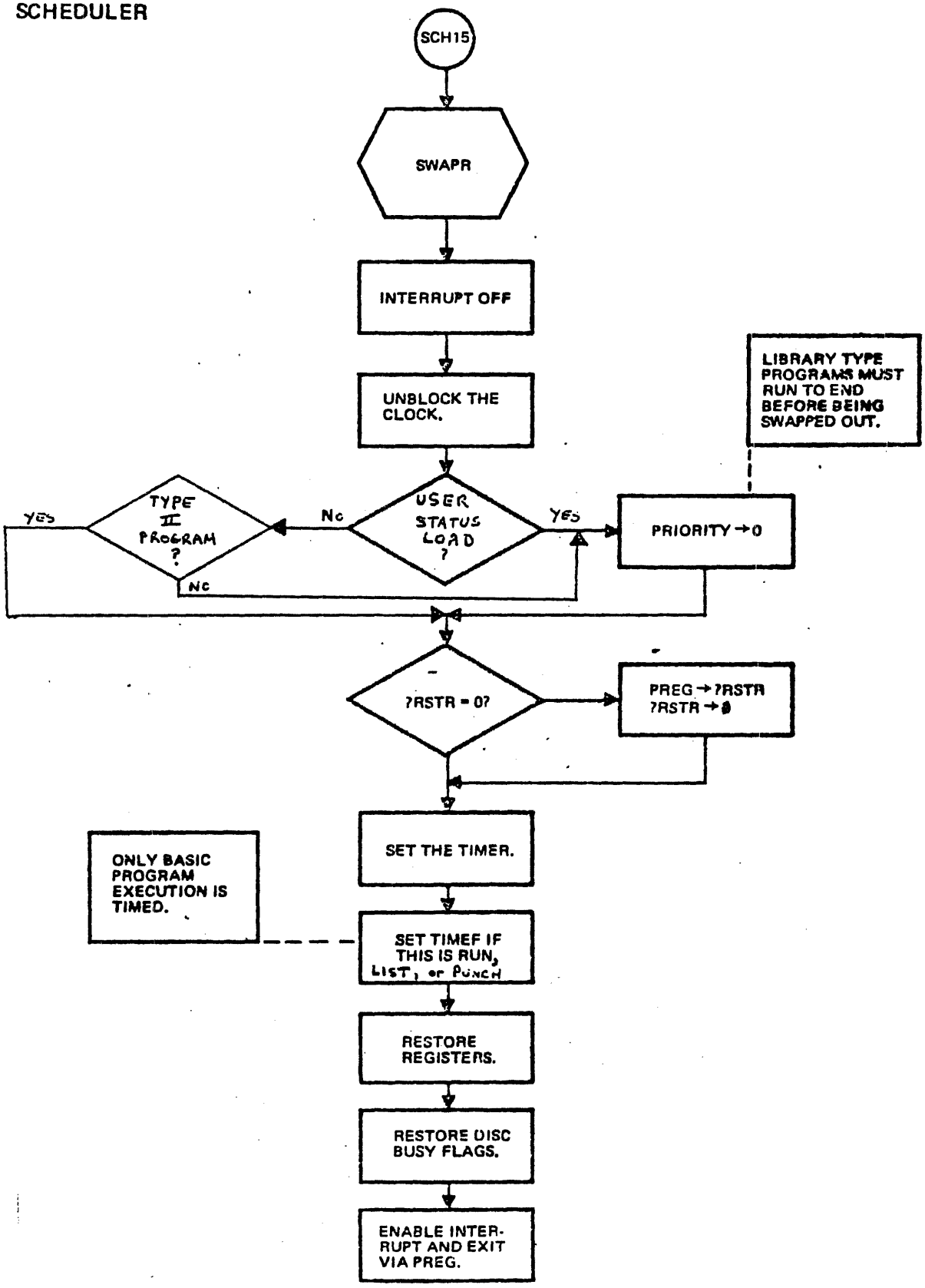
FLOWCHART
4 of 13

SCHEDULER

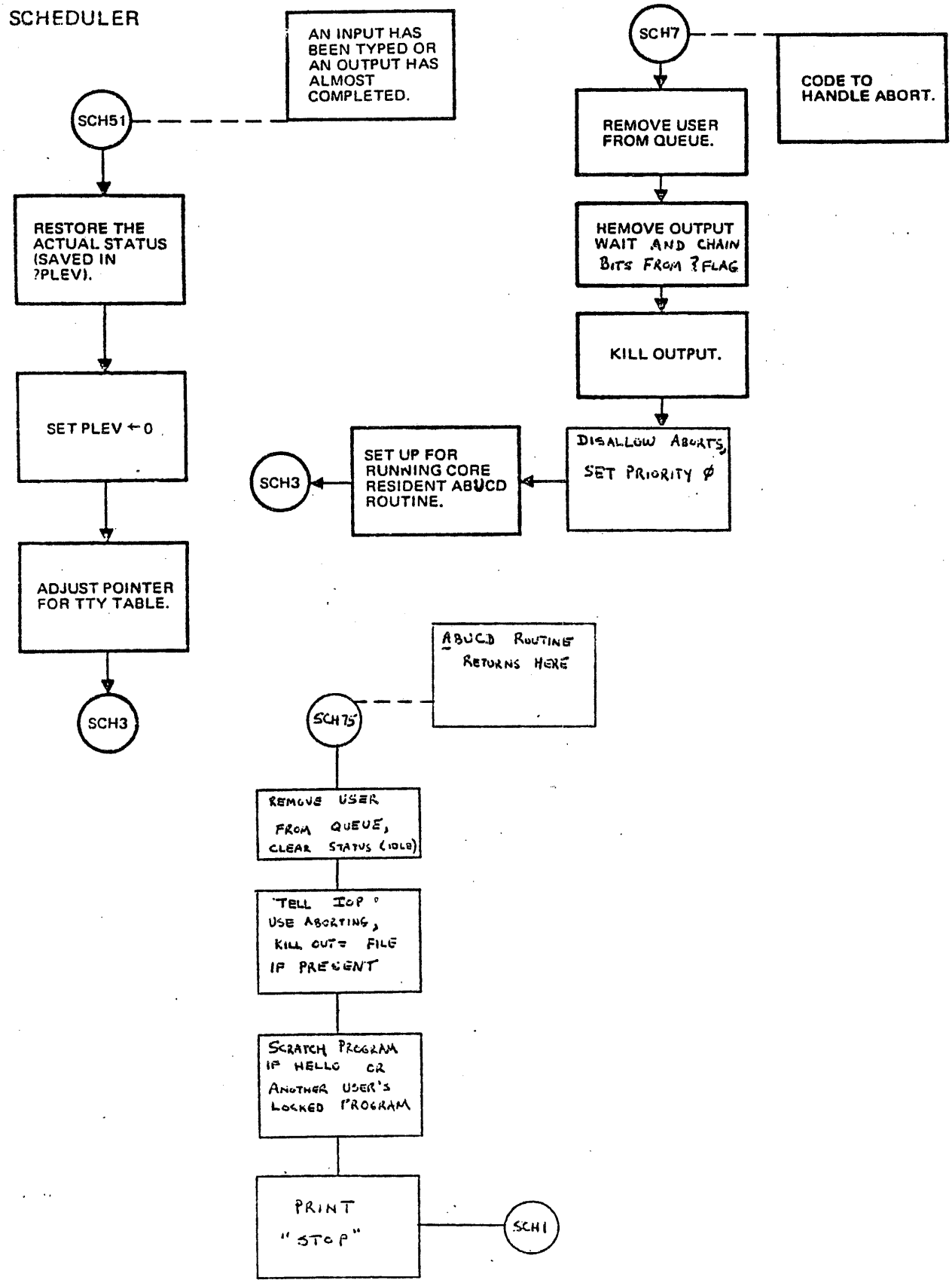


FLOWCHART

SCHEDULER

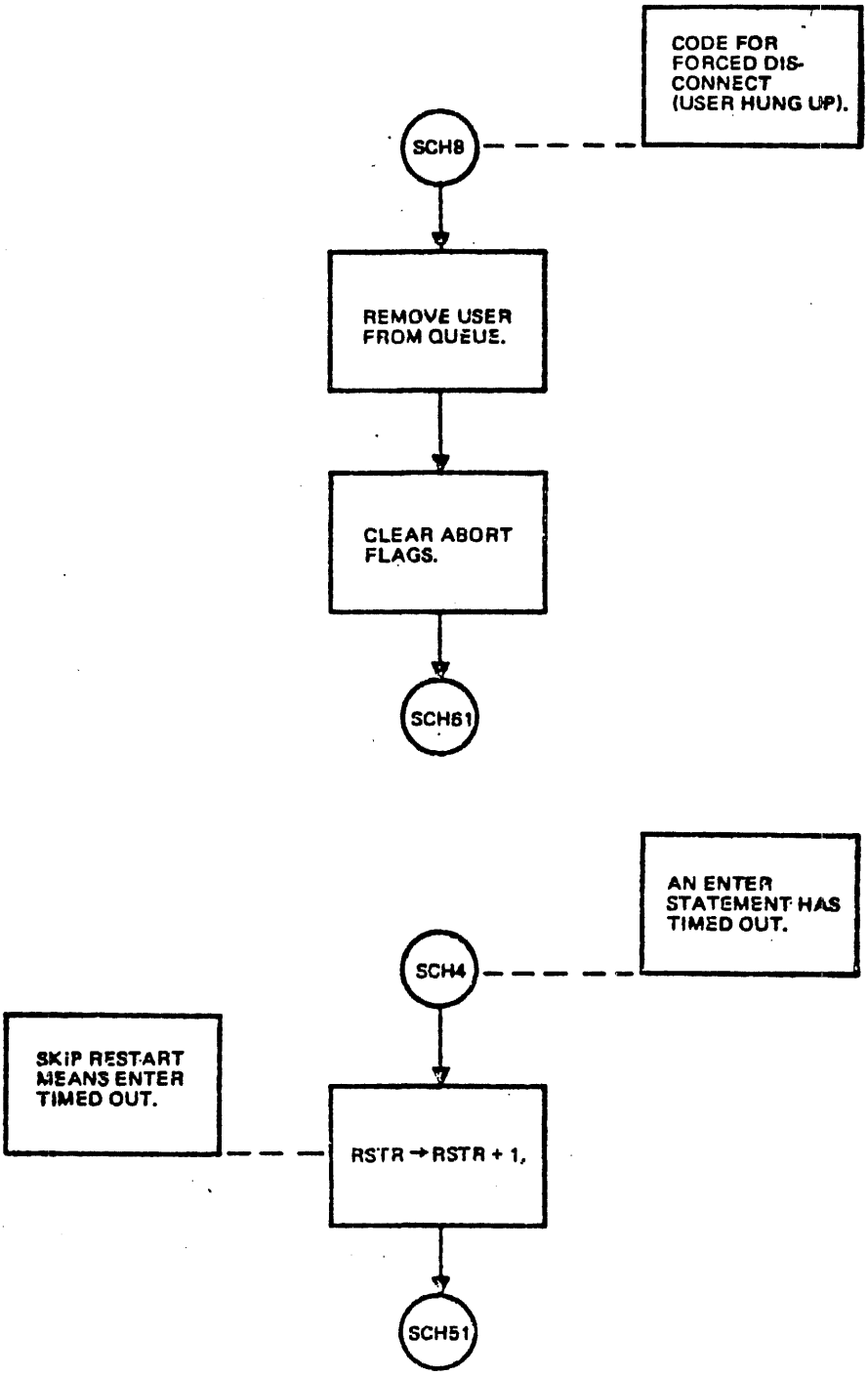


FLOWCHART

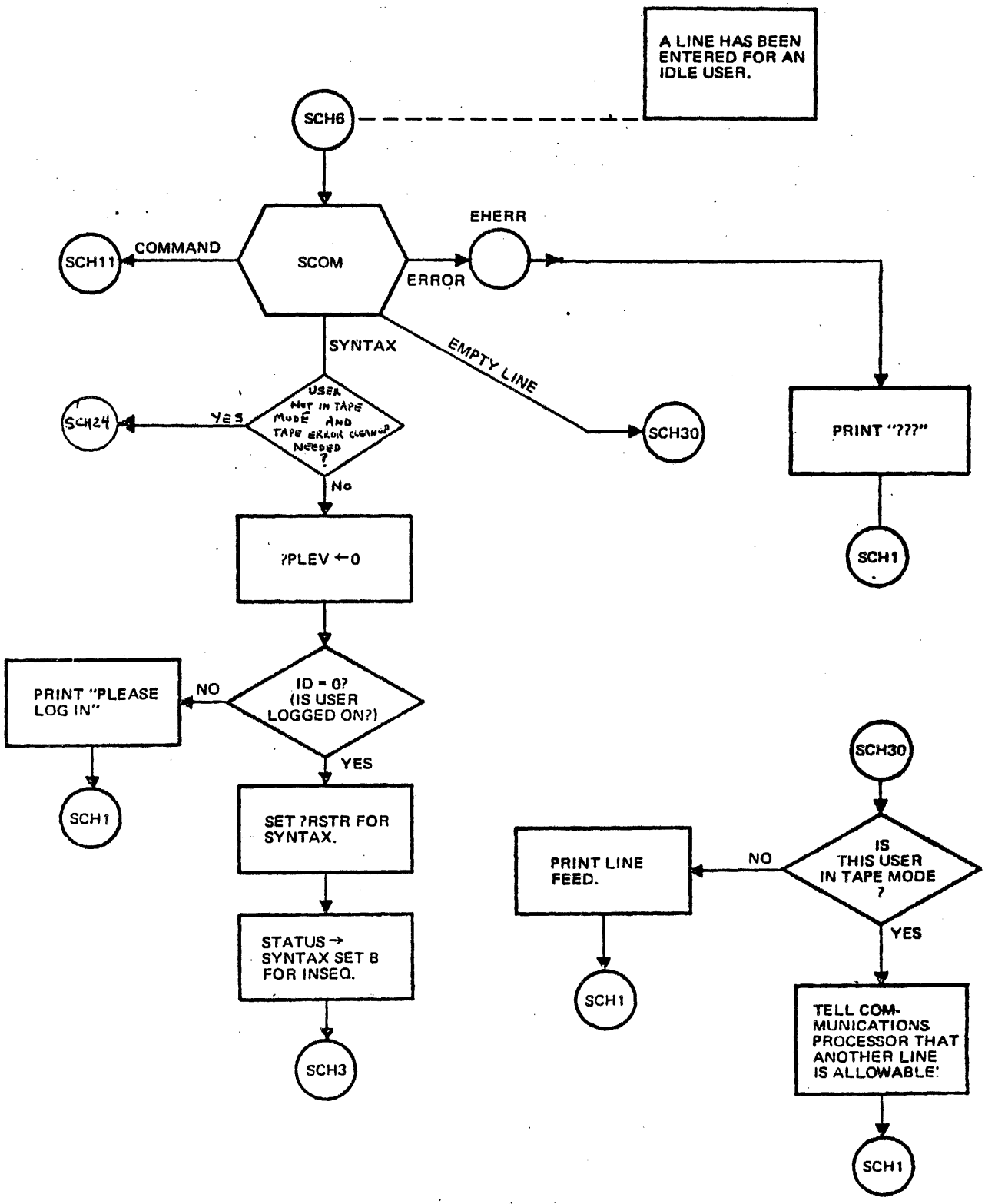

SCHEDULER
AN INPUT HAS BEEN TYPED OR AN OUTPUT HAS ALMOST COMPLETED.
SCH51
RESTORE THE ACTUAL STATUS (SAVED IN ?PLEV).
SET PLEV ← 0
ADJUST POINTER FOR TTY TABLE.
SCH3
SCH7
CODE TO HANDLE ABORT.
REMOVE USER FROM QUEUE.
REMOVE OUTPUT WAIT AND CHAIN BITS FROM ?FLAG
KILL OUTPUT.
DISALLOW ABORTS, SET PRIORITY ∅
SET UP FOR RUNNING CORE RESIDENT ABUCD ROUTINE.
SCH3
ABUCD ROUTINE RETURNS HERE
SCH75
REMOVE USER FROM QUEUE, CLEAR STATUS (IDLE)
TELL IOP ° USE ABORTING, KILL OUT= FILE IF PRESENT
SCRATCH PROGRAM IF HELLO OR ANOTHER USER'S LOCKED PROGRAM
PRINT "STOP"
SCH1

FLOWCHART

SCHEDULER

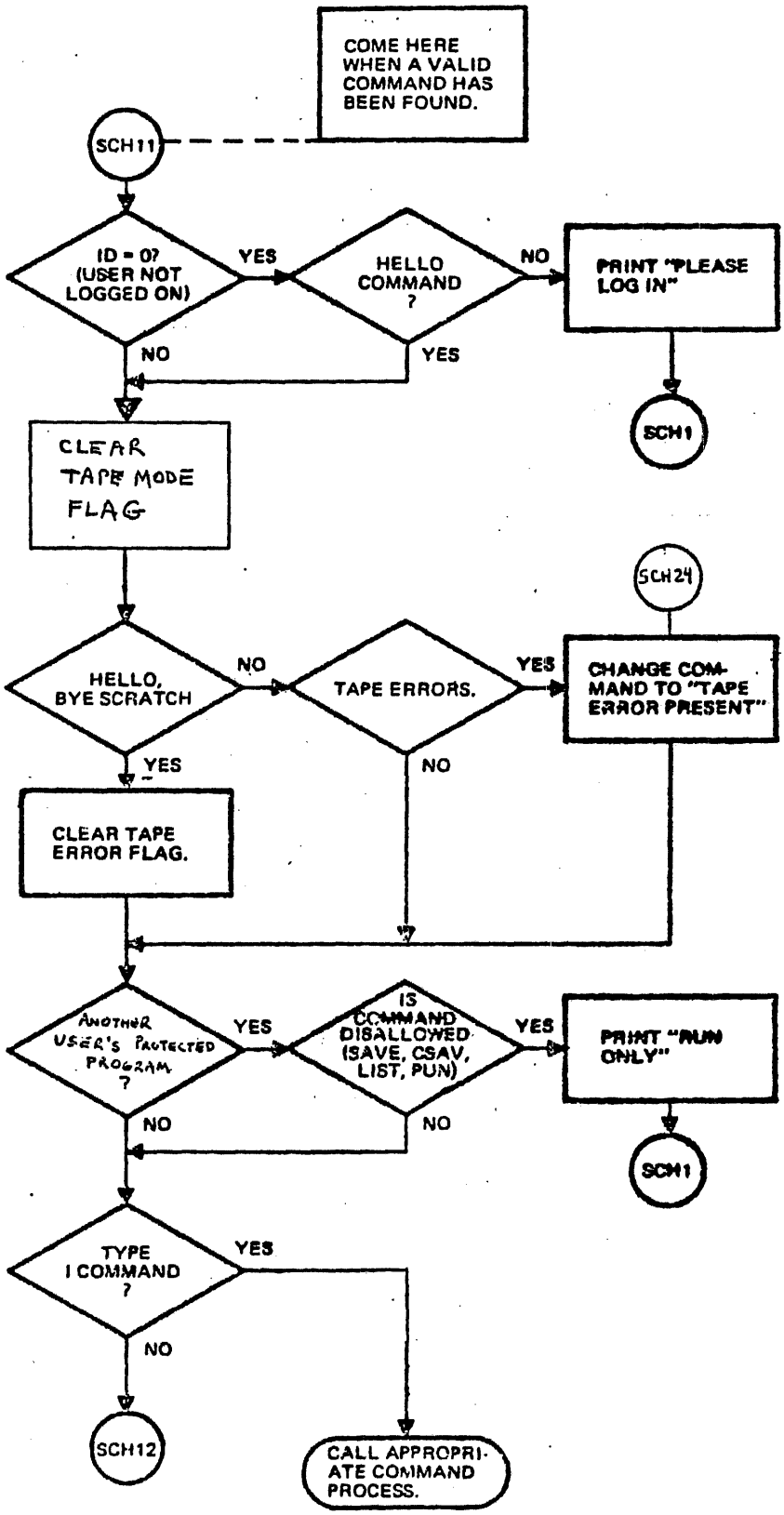

CODE FOR FORCED DIS-CONNECT (USER HUNG UP).
SCH8
REMOVE USER FROM QUEUE.
CLEAR ABORT FLAGS.
SCH61
AN ENTER STATEMENT HAS TIMED OUT.
SCH4
SKIP RESTART MEANS ENTER TIMED OUT.
RSTR → RSTR + 1.
SCH51

SCHEDULER

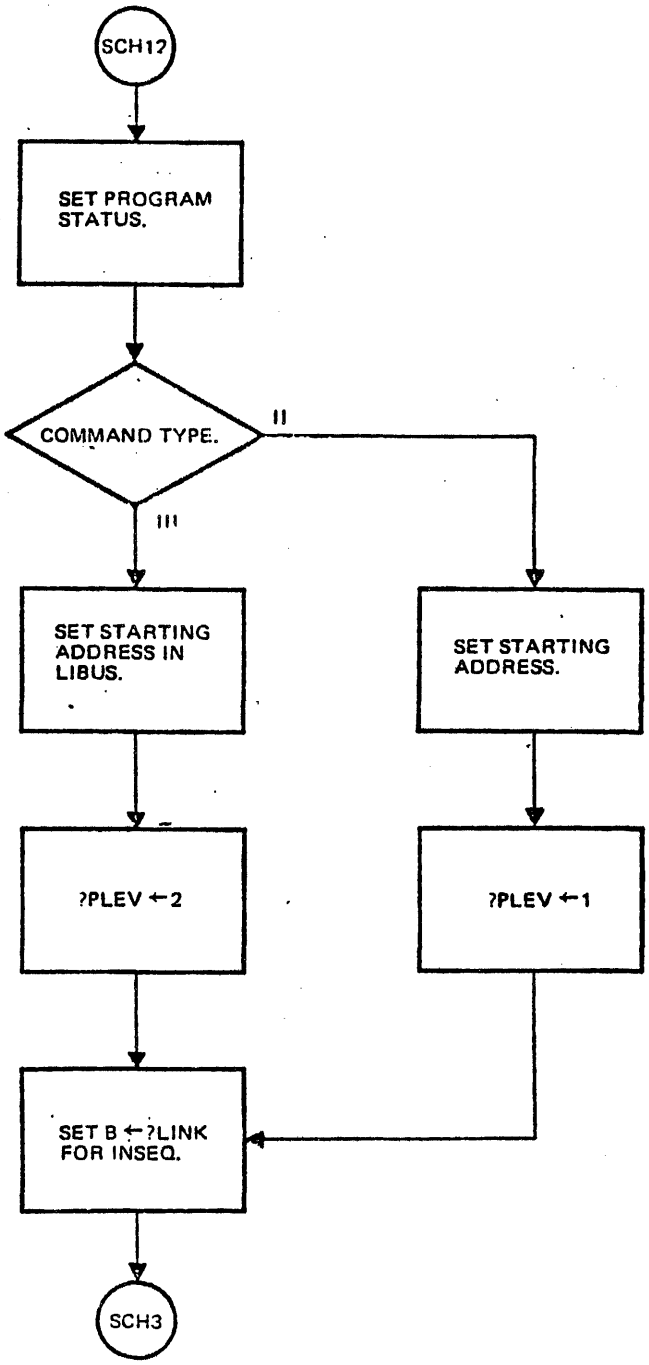


FLOWCHART

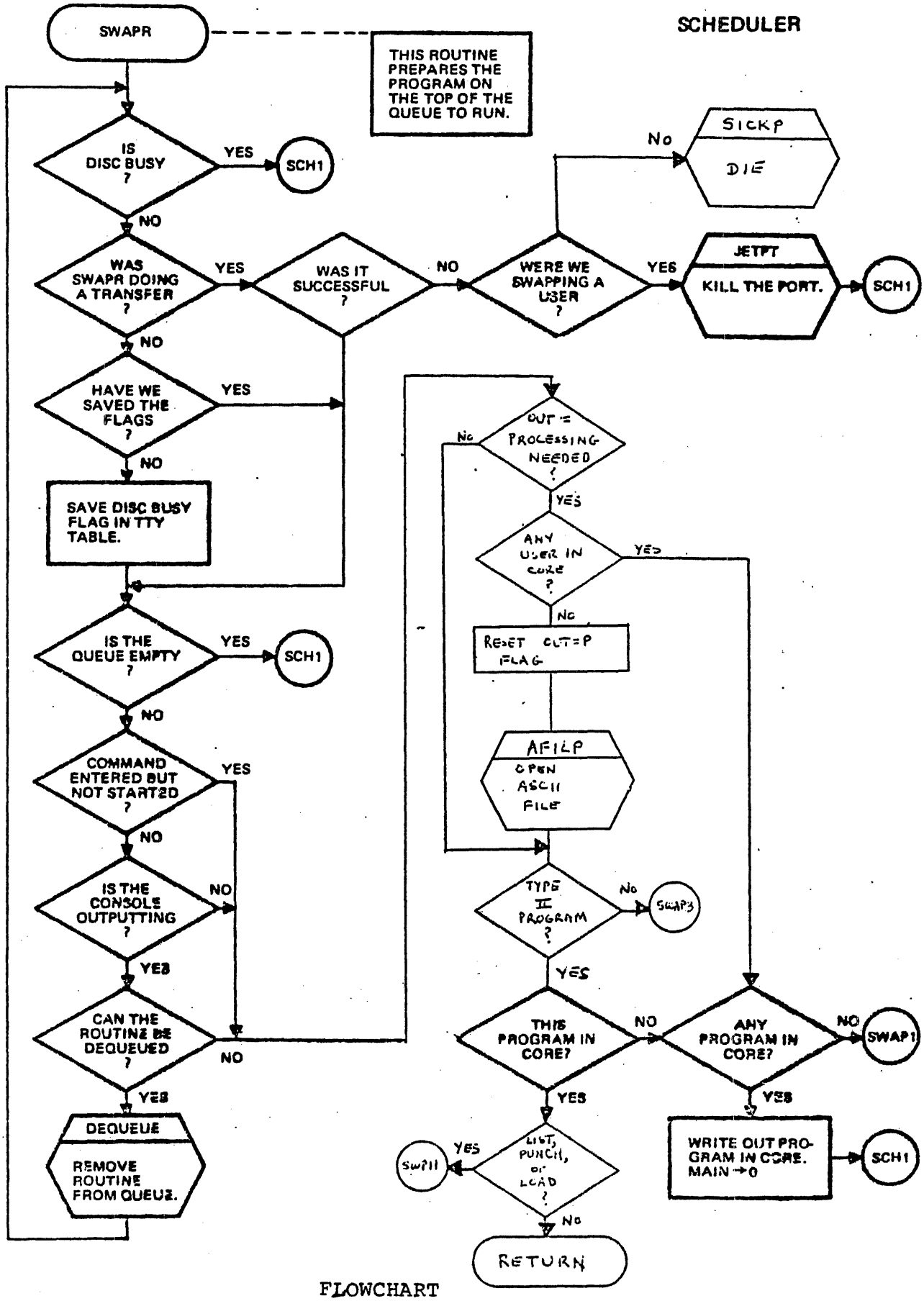
SCHEDULER


COME HERE WHEN A VALID COMMAND HAS BEEN FOUND.
SCH11
ID = 0? (USER NOT LOGGED ON)
YES
HELLO COMMAND ?
NO
PRINT "PLEASE LOG IN"
NO
YES
SCH1
CLEAR TAPE MODE FLAG
SCH24
HELLO, BYE SCRATCH
NO
TAPE ERRORS.
YES
CHANGE COMMAND TO "TAPE ERROR PRESENT"
YES
NO
CLEAR TAPE ERROR FLAG.
ANOTHER USER'S PROTECTED PROGRAM ?
YES
IS COMMAND DISALLOWED (SAVE, CSAV, LIST, PUN)
YES
PRINT "RUN ONLY"
NO
NO
SCH1
TYPE I COMMAND ?
YES
NO
SCH12
CALL APPROPRIATE COMMAND PROCESS.

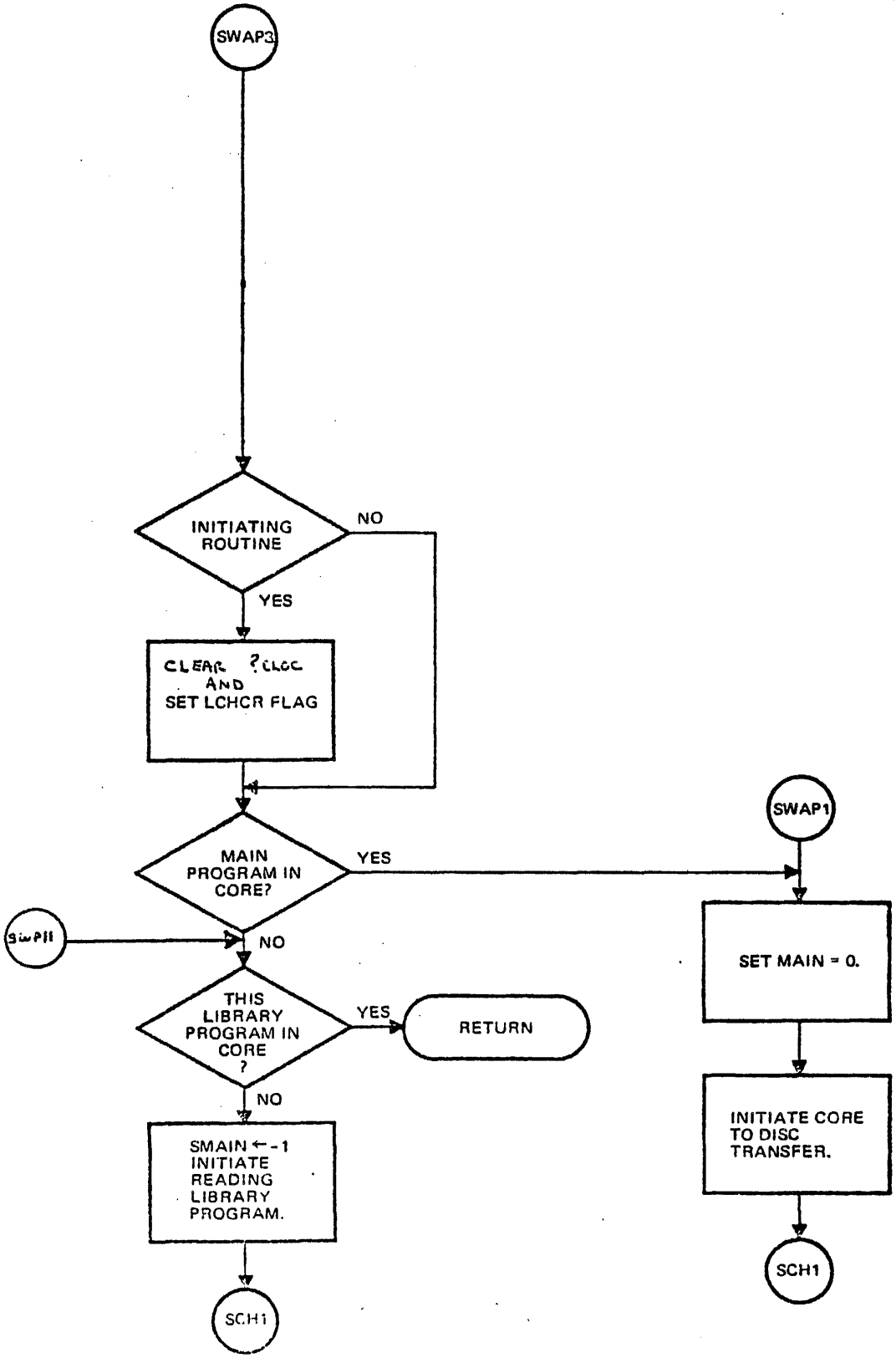
SCHEDULER



FLOWCHART



FLOWCHART
12 of 13


SWAP3
INITIATING ROUTINE
NO
YES
CLEAR ?CLOC AND SET LCHCR FLAG
MAIN PROGRAM IN CORE?
YES
SWAP1
SWAP11
NO
SET MAIN = 0.
THIS LIBRARY PROGRAM IN CORE ?
YES
RETURN
NO
INITIATE CORE TO DISC TRANSFER.
SMAIN ← -1 INITIATE READING LIBRARY PROGRAM.
SCH1
SCH1

FLOWCHART

COMMUNICATION BETWEEN SYSTEM MODULES

COMMUNICATION BETWEEN SYSTEM MODULES

There are six system modules that communicate with each other in various ways: the disc driver, I/O Processor driver, system console driver, scheduler, BASIC, and system library routines (HELLO, BYE, KILLID, etc.).

1. DISC DRIVER

Any section of the system may call the disc driver to perform a (moving-head) disc transfer. Three parameters are passed to the driver:

A = pointer to disc address	(the core address of a two word logical disc block number at which the transfer is to begin)
B = core address	(bits 14-0 = core address at which transfer is to begin; bit 15 = 1 for read from disc to core; bits 15 = 0 for write from core to disc)

The variable MWORD = the negative of the number of words to be transferred. If MWORD>=0, the driver will cause no transfer, but will position the appropriate disc unit at the specified block.

The disc driver is called by JSB DISCA,I.

The driver determines the logical disc on which the specified block lies, and, if that logical disc is present on the system, processes the requested transfer. While a request is being processed and transfer taking place, the driver busy flag, MBUSY, is set to -1. If the driver is called while MBUSY is so set, it will return without doing anything. If the disc block number passed to the driver does not lie on one of the discs present on the system, the driver will increment the return address by one and return without doing anything. If the driver accepts the request, it will increment the return address by two and return after processing of the request has been initiated.

A moving head disc transfer involves two steps: positioning the heads to the correct area of the disc and performing the actual data transfer. The disc driver returns to its caller while each of these is going on. Command channel interrupts return control to the driver when the operations are complete: the driver checks for successful completion of the operations before proceeding.

A single data transfer on a disc cannot automatically continue from one cylinder to the next. The 7900 disc has the further restriction that a transfer cannot cross the "mid-cylinder" boundary (between track 1 and track 2). When a data transfer is requested which crosses one or more of these boundaries, the disc driver breaks up the transfer to conform with the restrictions.

When the driver completes handling a request and returns to the caller, MRUSY is set to indicate the outcome of the transfer as follows:

0: The requested transfer has been successfully completed.
1: The transfer has failed; the seek (position) operation could not be completed.
2: The transfer has failed; the data transfer was unsuccessful.
3: The transfer has failed; part of the data lies on, or would be written to, a disc which is not present on the system.

A complete disc transfer can be performed by the following sequence:

```
JSB DISCA,I
    <return for driver busy>
    <return for disc not present>
    LDA MRUSY
    SSA
    JMP *-2
    SZA
    <process disc error>
    <process successful transfer>
```

The disc driver does not modify the contents of MWORD and the A and B registers. The system never suspends a program for a disc transfer.

SYSTEM CONSOLE DRIVER

The system console driver mainains three flags, T35F1, T35F2, and T35F3, which determines its status. The meaning of these flags are as follows:

T35F1: = -1 driver is busy, 0 otherwise
T35F2: Normally 0, it is set to -1 by the driver at the conclusion of input, and cleared to 0 externally.
T35F3: Normally 0, it is set to -1 by the driver at the conclusion of input, and cleared to 0 by the driver after output has been initiated.

The combined values of these flags are more significant:

F1	F2	F3	
0	0	0	Driver is accepting input
0	-1	-1	Input command received and is being processed, but output has not been initiated.
0	-1	0	Output terminated from a system command which is to be reinitiated.
-1	0	0	Outputting
-1	-1	0	Outputting, at the end of which the current system command will be reinitiated.

When F2 = -1, the driver will not accept any input. This guarantees system library programs that they will not be interfered with. These routines are responsible for clearing F2 when they call the driver for the last time. F2 and the console status (T355T) are cleared if a key is struck on the console during output for certain console routines. This will effectively terminate such routines as DIRectories, REPorts and STATuses.

When F3 = -1, log-on and log-off reports as well as the message queue are held off. This guarantees that these messages will not be interfered with by the system library program output.

F1 when set will hold off log-on and log-off reports as well as the message queue. F1 is set during output, F1 is also set for (50 sec) whenever the user presses a key (except CR).

IERMINET_SYSTEM_CONSOLE

The following variables and flags are also, maintained for a terminet console by the console driver. These are necessary because the console must be turned off after its been idle for approximately 50 seconds and turned on again when there is output to be printed. The terminet console also requires fill characters after a LF.

Bit 15 if set indicates a terminet console
TD: Normally 0, 1 when a motor shutdown is in progress
TDSU: Normally 0, 1 when a motor shutdown is in progress
TMTST: Contains the motor state 0 is off, 1 if on
TFCNT: Contain number of fills left to output

The calling sequence is:
A: bit 15 = 0 if CRLF is to be appended, bits (14:0) = # of chars.
B: bit 15 = 1 if punching is to take place in addition to printing, bits (14:0) = core address of output buffer.
JSR TTY35,1

The driver uses the 36 word buffer T35BF as an input buffer. Most of the library routines use it for output, and occasionally for temporary storage between lines of output.

SYSTEM PROCESSOR-I/O PROCESSOR COMMUNICATION

I. PROCESSOR INTERCONNECT

The block diagram for the Processor Interconnect is on the second page following. In the IDLE state, the interface cards are set up as follows:

C1(CH1)	CONTROL & ENDCODE:	SET
	FLAG & IRQ:	CLEAR
C2(CH2)	CONTROL & ENCODE:	CLEAR
	FLAG:	SET
	IRQ:	CLEAR

A data transmission operation occurs thusly:

1. Sending machine waits for flag to be set on C2(CH2) indicating that the previous transmission has been processed.

2. Sending machine places data word in output register of C2(CH2) thereby placing it on the input register of CH1 (C1). (OTA/B C2(CH2)).

3. Sending machine issues STC, CLF to C2(CH2) making the ENCODE LINE go high, setting FLAG on CH1 (C1), clearing ENCODE on CH1(C1), and strobing the data word into CH1(C1).

4. Sending machine issues CLC to C2(CH2) to prevent an interrupt from that card. The sending machine is now free to return to other tasks.

5. In the receiving machine, T5 will set the IRQ on CH1(C1). If the interrupt system is enabled and the priority line is high, the IRQ will cause an interrupt to a service routine.

6. The service routine does an LIA/B from CH1 and decodes the 16 Bit data word.

7. If a response is called for, the receiving machine can load the output register with a data word (OTA/B CH1(C1)).

8. When the receiving machine has completed its processing, it issues an STC,C(CLF) to CH1(C1) which restores the cards to the idle state.

The following is the resultant statuses of the sending two computers after a command has been sent, received and acknowleaged:

a) The flag is set on C2(CH2) of the sending computer indicating that another transfer is now allowed. This occurred when the receiving computer issued an STC,C to CH1(C1) after it had decoded and executed the command. The STC,C is the acknowledgement to the SEND computer that the RECEIVE computer did receive the transmission.

b) The control on C2(CH2) is cleared by the CLC to C2(CH2). This was done to inhibit the interrupt that normally would occur after the SEND computer outputted the command.

c) The control is set and the flag is cleared on CH1(C1) (from the STC,C acknowledgement) indicating readiness to receive another transmission.

SYSTEM PROCESSOR COMMUNICATIONS TO I/O PROCESSOR

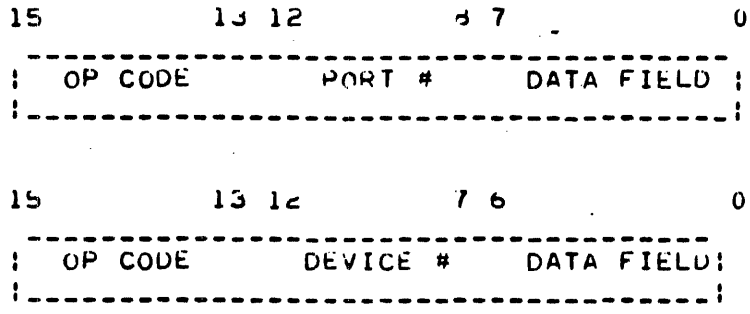
The system processor sends communications on I/O 11 channel and receives them on I/O channel 10. An exception is a communication sent by the system processor which requires a response, which will be received on I/O channel 11. Communications are initiated by JSB SDVRP,I with the communication in the A register.

The system processor can initiate requests and commands for the I/O Processor. Six of these are of general nature (i.e. not limited to a single user). They include PHS, PRR, RJE, SCI and TCM.

User commands associated with hardware control include ECO, ECF and STP. User commands associated with buffer control are POC, POS, BKS, FNC, PIS, RLB, SBP, and RBP. User commands associated with the general status are STE, WTP, UIR, UNR, IWT, HUU, ULO, TOP, NUC, KTO, ALI, OWT, IBA and ABT.

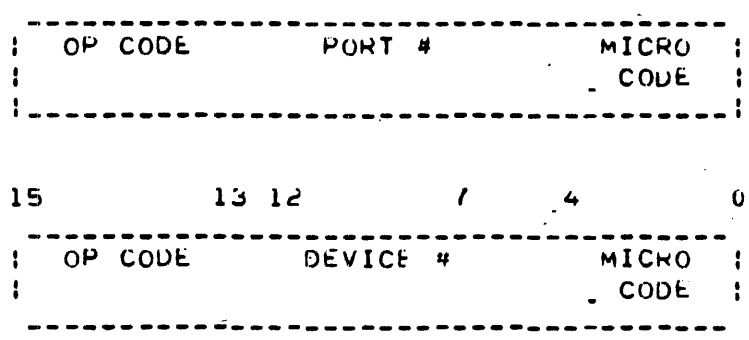
Commands concerned with control of ASCII devices are STR, ADV, RDV, ALB, XRB and KDO.

Code_Formats



The above format holds for command codes 0 thru 5

Command codes 6 and 7 are micro-coded and have one of the following formats:



Commands in the above format which must supply data are two word commands with the data supplied in the second word.

SYSTEM PROCESSOR --> I/O PROCESSOR

Numeric Value of OP_Code	Mnemonic	Description
0	POC	Process output character
1	STE	Start ENTER Timing
2	STP	Subtype information
3	PHS	Phones timing parameter
4	PCF	Perform control function
5	POS	Process output string
6	MIKRO 0	Micro code group 0
7	MIKRO	Micro code group 1

MIKRO 0

Microcode	Mnemonic	Description
0	(unused)	
1	WTP	What terminal type

MIKRO

Microcode	Mnemonic	Description
0	PRR	Initialize IOP/preempt request
1	UIR	User is running
2	UNR	User not running
3	IWT	Input wait
4	HUU	Hang user up
5	ULO	User logged on
6	ECO	Echo-on
7	ECF	Echo-off
10	TPO	Tape mode
11	STR	Start timed retries
12	NUC	New user called
13	KTO	Kill terminal output
14	ALI	Allow input
15	OWT	Output wait
16	IBA	Is buffer available
17	ADV	Allocate device
20	RDV	Release device
21	ALB	Allocate buffer
22	XRB	Transfer input buffer
23	BKS	Backspace terminal buffer
24	KDO	Kill device output
25	FNC	Fet next character
26	RJE	RJE command
27	ABT	User is being aborted
30	PIS	Process input string

31	(unused)	
32	SCI	Send core image
33	RLB	Release buffer
34	SSD	System shutdown
35	SBP	Save buffer pointer
36	RBP	Release buffer pointer
37	TCM	Transmit console message

POC

The process output character command contains a port number in bits 12 thru 8 and an ASCII character in bits 7 thru 0. The character is to be placed in the port's output buffer.

SIE

The start enter timing command contains the port number in bits 12 thru 8 and an integer between 1 and 255 in bits 7 thru 0. The integer specified is the maximum number of seconds the user port is to be enabled for entering a line of data. If the time elapses without the user completing a line, the IOP returns the command ETO. Otherwise the IOP returns the command HVL which contains the actual response time in bits 7 thru 0.

SIP

The subtype information command contains the port number in bits 12 thru 8 and the terminal type code in bits 7 thru 0. The terminal type code is an integer between 0 and 8 and specifies the terminal type of the port. This command is sent by the SP to the IOP when the SP is executing the HELLO command and has evaluated the terminal type parameter.

PBS

The phones timing parameter contains an integer value between 0 and 255 in bits 7 thru 0. This value is the number of seconds that a user is permitted for logging on. This command is sent when the system operator issues the PHONES command at the system console. If no command is given, the default logon time is 120 seconds.

PCE

The perform control function command contains a logical unit number in bits 12 thru 7 and a control code in bits 6 thru 0. The control code corresponds to the device control code in the table of CTL function values and indicates the operation to be performed on the device whose logical unit number is specified. The SP sends this command as the result of evaluating a CTL function in a PRINT statement list. The IOP returns a word response. See the section on I/O responses for possible return values.

POS

The process output string command contains a port number in bits 12 thru 8 and an integer character count in bits 6 thru 0. This command signals the IOP that the SP is ready to transmit an output string to the specified port. The IOP sends the SP a one word response - zero if ok to send the string, non-zero if not. If it's ok to send, the SP sends the string using DMA. If not, the IOP will send a BFE command to the SP when a buffer becomes available.

WIP

The what terminal type request contains a port number in bits 12 thru 8. The IOP sends the SP a one word response containing the integer value of the specified port's terminal type. This command is sent when a BASIC program executes the SYS command with parameter 4.

PBB

The initialize IOP/preempt request is sent by the loader and never occurs during normal system operation. See loader for additional details.

UIB

The user is running command contains a port number in bits 12 thru 8. This command is sent to the IOP whenever a user program begins execution on the specified port.

UNB

The user not running command contains a port number in bits 12 thru 8. This command informs the IOP that the user program on the specified port is no longer running due to normal termination or program error.

IWI

The input wait command contains a port number in bits 12 thru 8. This command is sent to the IOP to enable input on the specified port. The SP then suspends the port with status of input wait. The IOP will wake the user with a HVL, HVP, or HLL.

HUU

The hang user up command contains a port number in bits 12 thru 8. This command is sent whenever a user executes the BYE command or if the jettison port routine determines that the user swap tracks is unusable for the specified port.

ULO

The user logged on command contains a port number in bits 12 thru 8. This command informs the IOP that a user has successfully logged onto the system.

ECO

The echo-on command contains a port number in bits 12 thru 8. This command is sent to the IOP when the user at the specified port executes the ECHO-ON command. The effect is to initiate an echo for full duplex terminals.

ECE

The echo-off command contains a port number in bits 12 thru 8. It is sent to the IOP when the user at the specified port executes the ECHO-OFF command. The effect is to inhibit echo for half-duplex terminals.

IBO

The tape mode command contains a port number in bits 12 thru 8. This command is sent when the user on the specified port executes the TAPE command.

SIB

The start timed retries command contains a logical unit number in bits 12 thru 7. This command is sent after an unsuccessful I/O operation was attempted on the specified device to retry the operation. The IOP will return a WUU whenever the retry is successful.

NUC

The new user called command contains a port number in bits 12-8. This command is sent to inform the IOP that the user logged on the specified port is executing a HELLO command.

KIO

The kill terminal output command contains a port number in bits 12-8. This command is used to completely purge any output currently in the specified port's IOP buffers. It is sent when the user hits BREAK. if the system operator types SLEEP, or if the SP is shutting down the system due to a hardware failure.

ALI

The allow input command contains a port number in bits 12-8. This command is sent to the IOP to enable the specified port to enter lines of syntax when the terminal is in tape mode.

OWI

The output wait command contains a port number in bits 12 thru 8. This command tells the IOP to send a BFF command when the specified port has a buffer available. The SP sends this request for user terminal commands which generate several lines of output. This allows us to re-schedule the user after he has successfully output a line and a terminal buffer is once again available.

IBA

The is buffer available command contains a port number in bits 12 thru 8. This command is sent when the system operator is executing an ANNOUNCE and the IOP returns a one word response indicating either buffer available (response =0) or not available (response <> 0). When the response is affirmative the SP will send the message via POS.

ADV

The allocate device command contains a logical unit number in bits 12 thru 7. The second word of the command indicates the size of the buffer which should be allocated for the specified device. The IOP returns a one word response indicating success (response = 0) or device busy (response <> 0). This command is sent whenever an ASCII file is opened on a given IOP device.

RDV

The release device command contains a logical unit number in bits 12 thru 7. This command tells the IOP that the specified device is no longer needed and that all remaining buffers for the device should be written and then de-allocated. The IOP returns a one word response. See section on I/O responses for possible return values.

ALB

The allocate buffer command contains a logical unit number in bits 12 thru 7. The second word of the command contains the length of the output buffer in bytes. The IOP then returns a one word response. If the response is zero, then the SP sends the output buffer to the IOP via DMA. See section on I/O responses for other possible return values.

XBB

The transfer input buffer command contains a logical unit number in bits 12 thru 7. The IOP returns a one word response indicating whether it has a full input buffer on the specified device. If it does (response = 0), then the SP acknowledges the response (STC,CLF) and then the next word on the send channel is the size of the buffer (in bytes). After acknowledging the buffer size the SP transfers the input buffer using DMA. The SP acknowledges receipt of the buffer by another STC, CLF.

BKS

The backspace terminal buffer command contains a port number in bits 12 thru 8. The command causes the IOP to backspace the specified terminal's input buffer pointer one character.

KDO

The kill device output command contains a logical unit number in bits 12 thru 7. This command tells the IOP to completely purge and de-allocate any buffers or activity on the specified device. Usually sent due to unrecoverable I/O failures or abnormal program termination.

ENC

The fetch next character command contains a port number in bits 12 thru 8. The IOP returns a one word response which is the next character in the specified port's input buffer. The input buffer pointer is advanced to the next character.

RJE

The remote job entry command contains a word count in bits 12 thru 7. The command is sent when the system operator types an RJE command on the system console. The word count is the length of the message. The IOP always rejects the RJE command the first time. The IOP will send WRU when it is ready for the message. The RJE command is then guaranteed to be accepted and the message is then transmitted to the IOP via DMA.

ABI

The user is aborting command contains a port number in bits 12 thru 8. This command informs the IOP that the user program on the specified port being aborted due to the user hitting the BREAK KEY.

PIS

The process input string command contains a port number in bits 12 thru 8. The second word is the buffer length which the specified port has allocated for input. The IOP responds with the actual number of characters it has, which will be $\leq$ the specified buffer length. The SP acknowledges the buffer length with a STC, CLF and proceeds to transfer the input buffer via DMA. The transfer is acknowledged with an additional STC, CLF. If the IOP actually had more characters than would fit in the port's SP buffer, it will transfer one buffer full and remember the buffer position. A subsequent PIS request will receive characters starting from the position that the previous request finished.

SCI

The send core image command contains a word count in bits 12 thru 7. The second word of the command is the starting address. The IOP responds by sending the specified number of worrds, starting with the specified address. The SP receives the response via DMA. This command is sent when the user AUUU uses the DUMP command to dump the IOP memory.

RLB

The release buffer command contains a port number in bits 12 thru 8. This command is sent to force the IOP to print and release any output which is in a buffer for the specified terminal. This command is sent by the SP whenever it wishes to ensure that all user output has been printed (i.e. whenever a port is swapped out).

SSD

The system shut down command is sent to inform the IOP that the system is being shut down due to the system operator executing the SLEEP command.

SBP

The save buffer pointer command contains a port number in bits 12 thru 8. This command tells the IOP to save the current value of the terminal input buffer pointer on the specified port. The pointer value can be restored by the RBP command. These commands allow the SP to read the next few characters in the buffer (via FNC) and to return to the current position without sending multiple BKS's.

RBP

The restore buffer pointer command contains a port number in bits 12 thru 8. This command tells the IOP to restore the terminal input buffer pointer on the specified port to the value saved via a SBP.

ICM

The transmit console message command tells the IOP to send a console message to the SP. The transfer is handled via DMA. This command is sent by the SP after the SP has received an SCM command from the IOP.

SP. The possible values are as follows:

-3	No data available on RJE or LT
-2	end-of-file
-1	buffer not ready
0	operation successful
1	device not ready
2	device error
3	attention needed
4	read/write failure.

In the case of a -1 response, the IOP will send a WUU for the appropriate device when a buffer becomes available. If the response is greater than 0, the SP will eventually send either an STR (in which case the IOP will send a WUU when the condition has been fixed) or a RUO.

I/O PROCESSOR COMMUNICATIONS TO SYSTEM PROCESSOR

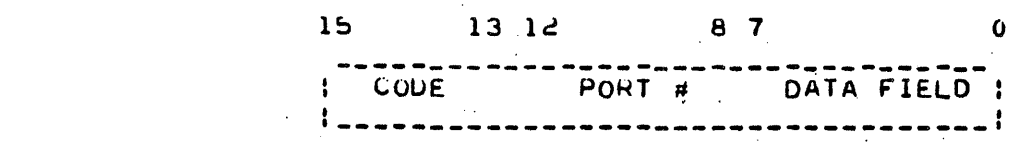
The system processor receives communications from the I/O processor on I/O channel 10. The system processor ignores inconsistent communications, e.g. accepts a line of input only when the user's status is IDLE or INPUT WAIT. The receive driver communicates with the scheduler by setting the PACT bit in the ?FLAG word of the port's teletype table and setting the appropriate status.

Communication requests initiated by the I/O Pocessor are divided into three general groups. User port commands affect one particular port and include HVL, HLP, HLL, ADR, BFL, BFE, ETO and UHU. Another group affects operation of the RJE facility and includes SCM, ADR, RDR, and WRU. The remaining commands, WUU, concerns operation of a non-shareable device.

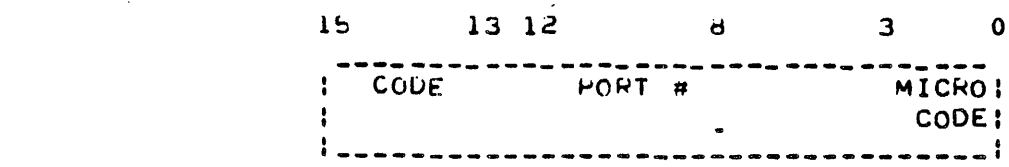
NUMERIC VALUE OF OP_CODE	MNEMONIC	DESCRIPTION
0	HVL	Have a line
1	HLP	Have a line - parity error
2	HLL	Have a line - lost character
3	(unused)	
4	(unused)	
5	(unused)	
6	(unused)	
7	MIKRO	Micro coded

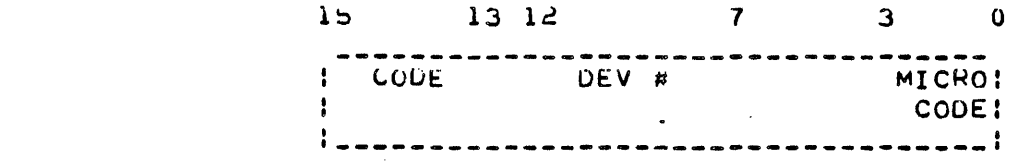
MIKRO_CODE	MNEMONIC	DESCRIPTION
0	ABK	User abort request
1	BFL	Buffer full
2	BFE	Buffer empty
3	ETO	Enter timed out
4	UHU	User hung up
5	SCM	Send console message
6	ADK	Allocate device for RJE
7	RDK	Release device from RJE
10	WUU	Wake user up
11	WRU	Wake RJE up
12	(unused)	
13	(unused)	
14	(unused)	
15	(unused)	
16	(unused)	
17	(unused)	

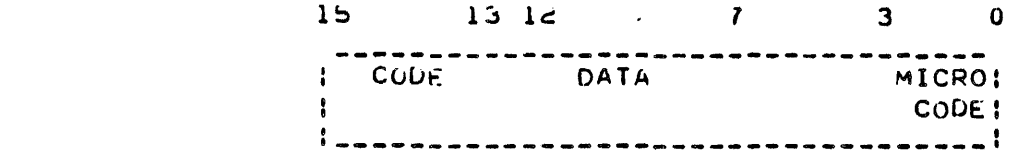
CODE_FORMATS



Command code 7 is micro-coded and has one of the following formats:







HVL,HVB,HVL

These commands contain a port number in bits 12 thru 8 and a response time in bits 7 thru 0. The commands indicate that the user on the specified port has input a complete line terminated by a carriage return. The response time field contains the number of seconds that it took the user to respond if the line was sent in response to an STE request by the SP. This value is saved in the user teletype table. The SP will set the PACT bit in the port's teletype table only if the user's status is IDLE or Input wait. The IRERR bit is set if HUP or HVL was received. Each of the commands causes the user to be re-scheduled at a different address.

ABR

The abort command contains a port number in bits 12 thru 8. The SP receives this command when the user on the specified port has hit the BREAK key. If the user has disabled break (PBFLG=1) and the system operator has not re-enabled it (CBFLG=0), the ABTRY bit in the user teletype table is set. ABTRY is also set if the system is executing critical code and has set the UNABT bit. A special check is made to allow the user to abort out of the LOAD command if his status is output wait or pause. If break is not disabled and UNABT is clear, then a check is made to see if the user is in an unabortable library program. If not, his status is set to ABORT and the PACT bit in the port's teletype table is set.

BEL

The buffer full command contains a port number in bits 12 thru 8. Ths command indicates that the specified port's output buffer is full. When this command is received, the SP sets the OUTPUT WAIT bit in the port's teletype table. This bit is always checked prior to issuing a POC for the port and the user is suspended pending BFE if it is.

BEE

The buffer empty command contains a port number in bits 12 thru 8. This command indicates that the specified port now has an output buffer available. Upon receipt of this command, the SP clears the OUTPUT WAIT bit in the port's teletype table. If the port's status is OUTPUT WAIT (indicating that the user is suspended due to a rejected POS or a delayed POC), then the PACT bit is set.

EIO

The enter timed out command contains a port number in bits 12 thru 8. This command indicates that the time period specified in and STE command has elapsed without the user at the specified port having typed a complete line. If the port's status is not currently ABORT, his restart address is incremented twice, his status set to TIMEOUT, and his PACT bit is set.

UHU

The user hung up command contains a port number in bits 12 thru 6. This command indicates that the user on the specified port has disconnected his terminal. If the user's status is currently syntax or library program running and he is at the head of the queue, HGUIS is set. Otherwise the user status is set to DISCONNECT and the PACT bit is set.

SCM

The send console message command contains a message length in bits 12 thru 7. This command informs the SP that the IOP has an RJE message for the system console. The SP saves the message length in CONML. At a later point in time the scheduler will request the message via a TCM command.

ADR

The allocate device for RJE command contains a logical unit number in bits 12 thru 7. This command is a request from the IOP to allocate the specified device. The SP checks the device table and if the device is available allocates it to RJE and returns a zero to the IOP. If the device is not available, a minus one is returned.

RDV

The release device from RJE request contains a logical unit number in bits 12 thru 7. This is a request from the IOP to release a device which was allocated in response to an ADR.

WUU

The wake user up command contains a logical unit number in bits 12 thru 7. The command indicates that the specified device has gone ready, an output buffer is available, or an input buffer is ready. This command is received to wake up the user controlling the device when he has been suspended after an STR command or after a -1 response was received for a PCF, ALD, XRB, or RDV command. The SP looks at the status of the port to which the specified device has been allocated. If that port has a status of either input wait or output wait then the OUTWT bit in the port's teletype table is cleared and PACT bit set.

WRU

The wake RJE up command has no parameters. When received it signifies to the SP that the IOP is now ready to accept the RJE command which was previously rejected. The SP checks that an RJE message is pending and sets RJEF6= -1 so that the scheduler can re-send the RJE request.

TWO PROCESSOR POWER FAIL CHARACTERISTICS

The two processors in the time share system have independent power supplies and consequently, power failure interrupts in either machine may occur at different times.

A problem arises if one computer is powered down, and the other machine attempts to send a transmission. Data will be lost as well as possible subsequent data transmissions. This is apparently caused by stray encode and data levels while power is coming up.

The external consequences of a lost data transmission are these:

1. A line (syntax, command, or input) being processed will be garbled.

2. Output characters will be lost. This problem will be hidden by the fact that the current output character is garbled (MUX quits sending during character).

3. Terminals on which a carriage return has come in may never have that line processed by the system processor. The terminal will not accept input and the break key must be used to re-establish communications when power is restored.

4. The system processor may lose the signal that indicates that the buffer for this user is almost empty. The terminal will stop typing and the program will remain in I/O suspend. The break key must be used to re-establish communications when power is restored.

5. If several users are typing on the I/O processor and the system processor is not running, all multiplexor activity may cease (I/O processor waiting for transmission to be acknowledged). This leads to the classic symptoms, i.e., no response to any struck key (even break), and termination of all output operations, perhaps with a space on the line (teletype chattering).

If the primary power source fails, the two machines will go down within milliseconds of one another and it is not so likely that any transmission will be in progress thereby being lost. If, however, only one processor's power is lost, one of the above symptoms is sure to occur if there is significant activity on the system.

To solve these problems, a cable is installed between the two systems. This cable causes the power to fail on both processors whenever a power fail is detected on one processor. Because one processor fails within nanoseconds of the other, the probability of data being lost or of a port lockout is minimized.

Unfortunately, some combinations of different hardware processor models do not permit use of the above mentioned cable. If the primary power source fails, both processors will power down almost simultaneously, mitigating the probability of the above problems occurring. If, however, only one processor loses power during normal system activity, there is almost certain to be some transmission loss. Consequently, if the system must be manually powered-down, either use a common power switch or turn off the I/O-Processor's power first. In this case the I/O Processor should be powered-up last when resuming system activity.

Logic of the ISB/G Power-fail/Power-up Routines

The TSB/G power-fail/power-up recovery routines in the two processors require close coordination to properly restore the interconnect kit. All of the following contingencies must be covered by the design: secondary power failures during one or both processor recovery routines, initiation of recovery by one processor before completion of the powerdown routine by the other processor; power failure during a DMA transfer on the interconnect, and power failure following a system shutdown but before the system restarts again. The strategy discussed below provides the mechanisms and restoration sequencing necessary to recover from any combination of these.

Each processor must maintain six words of information. One of these, POWFF, is a flag local to the power-fail/power-up routines. It is used to assist in identification of power failures occurring during recovery from a preceding one. RCHNO and SCHNO contain the last word sent (exclusive of a DMA transfer) on the interconnect receive and send channel respectively. These are obviously needed to restore the state of the interconnect. DMAFL is a state variable containing the DMA-on-interconnect status of the processor program. It is used to determine if the power failure disrupted a DMA transfer. The final two words record the length, starting memory address, and direction bit of the most recent DMA transfer on the interconnect. Their existence allows the recovery routine to restart an aborted transfer.

Since the order in which the processor will complete their recovery routines cannot be predicted, two sequencing requirements must be observed. First, each word placed on the interconnect is to be recorded in either RCHNO or SCHNO as appropriate before issuance of the 'STC n,C' which signals its presence to the other processor. Second, neither processor can proceed to its interconnect DMA routine until after execution of the last instruction affecting the system processor's send flag and I/O processor's receive flag, regardless of which processor will execute the instruction (i.e., one processor may have to wait for its flag to be set by the other processor). This condition is automatically met by the handshake requirements of POS, ALB, and RJE. It is also a fallout of the design for SCI, TCM, and the requests for the device table or COLD DUMP core image. However, use of XRB requires that the I/O processor wait for its receive flag to become set, after transmitting the transfer length word, before invoking its interconnect DMA routine.

The power-down routine distinguishes between four possible cases. 1) If power fails during normal time-sharing, POWFF = 0 and the restart address is not within the recovery procedure. Both processors must save their hardware registers and the flag states of the interconnect and interrupt system. The system processor (SP) also records the flag states of its peripheral devices. 2) If power fails while the SP is not time-sharing (during system shutdown, system loading, or after the SP has completed shutdown or failed), the SP will not save its state or attempt to recover. The I/O processor (IOP is normally still active (waiting for the SP to signal resumption of system activity) and does not distinguish this case from the preceding one. 3) POWFF = 1 identifies a power failure from the recovery routine prior to restoration of the interconnect. Since no system acitivity has occurred on either processor as yet, their current states are ignored and recovery begins anew from the information recorded at the initial failure. 4) If POWFF = 2 or the restart address lies within the recovery routine, then the interconnect was restored but the processor had not net completed the remainder of its recovery. This differs from case 3 in that the other processor might have completed its recovery and returned to its pre-failure activity. Having done so, it might further have executed an instruction which legitimately altered the state of the interconnect. Each processor is responsible for correctly restoring its send channel (simultaneously restoring the other processor's receive channel). Thus the processor in this situation need only determine if its send channel flag is not set (it could not have been cleared by the other processor) and, if so, overlay the previously recorded state. The state variable DMAFL allows detection of aborted DMA transfers on the interconnect. It must be set to zero whenever time-sharing begins in order to synchronize the processor states. Each entry to the interconnect DMA routine first saves the transfer parameters and then increments DMAFL to the next odd value. When the processor's hardware DMA flag comes set, it increments its DMAFL to the next even value, executing not less than three instructions (including the SFS 7) to ensure that the other processor has time to complete its current instruction, last cycle for DMA channel 7. If the last two bits of both processor's state variables are even (00 vs. 00 or 10 vs. 10) then no transfer is active. DMAFL will be odd (01 or 11) while a processor is in the critical portion of its transfer routine. The other processor's DMAFL will be one less (00 vs. 01 or 10 vs. 11) if it has not reached its transfer routine, or will be equal (01 vs. 01 or 11 vs. 11) if also in its transfer routine, or will be one more (10 vs. 01 or 00 vs. 11) if the transfer completed. The remaining combinations (00 vs. 10 or 01 vs. 11) cannot occur since the processors can never be more than one state apart.

The IOP power-up recovery routine is straightforward except for restoration of the interconnect. It begins by setting POWFF to 1 and then re-enabling power-fail interrupts. Since recovery requires an exchange of information, the IOP must clear its receive channel flag and then wait enough time for the SP to complete its power-down routine, clear its receive channel flag, and transmit its data. The IOP routine does this by incorporating a timing factor into its wait loop. If the loop times out, then the SP must not be active and the IOP should restore itself to the same state as it would be in following reception of a SSD from the SP. After clearing its receive channel flag, the SP will execute at least eighty instructions (allowing ample time for the IOP to reach its wait loop) and then send a word containing zero bits except for the direction bit of the last DMA transfer (1 if from the IOP, 0 if from the SP) in bit 0 and the last two bits of its DMAFL in bits 2 and 1. The recovery routines are not in synchronization. The IOP responds with its most recent DMA direction bit (0 if from the IOP, 1 if from the SP) and the low two bits of its DMAFL in the same format.

At this point both processors have a complete record of the interconnect's state at power-fail. The IOP automatically places the current value of RCHNO on its receive channel and sends it to the SP. If the IOP's DMAFL is odd and the SP's information indicates that the transfer did not complete, the IOP makes a note to reinitialize its DMA routine later in the recovery process. The IOP now waits for the SP to set the IOP's send channel flag. When this occurs, the IOP examines its record of the flag at the time when power failed. If it was clear, the IOP restores the word from SCHNO and issues the 'STC n,C' to transmit it. Finally, POWFF is incremented to 2 since the interconnect is not completely restored (the SP will not finish its recovery until well after the IOP finishes all of the above). The remainder of the IOP routine consists of ensuring that an aborted DMA transfer on the interconnect will be restarted, performing the out-of-line device recovery routines, resetting POWFF to 0 after returning to the power-up routine, restoring the hardware registers and interrupt system flag, and resuming time-sharing.

The SP recovery routine is somewhat more complex. It first sets POWFF to 1 and re-enables power-fail interrupts. The SP then reconstructs the possible nesting of power-fail, receive channel, system console, and time-base generator drivers. It does this by comparing the return address of each active routine (power-fail is of course active to initiate this search) with the boundaries of the lower-priority routines it might have interrupted. Each flag which was clear at the power failure is cleared during this process (the receive channel flag is always cleared here) and an interrupt is forced for each active routine whose flag was set.

to clear the flag buffer and prevent a false interrupt later. At least eighty instructions will be executed by completion of this activity. The SP assumes that the IOP has reached the wait loop of its recovery routine and transmits the information word described above on its send channel. The two processors are now in synchronization.

At this point both processors have a complete record of the interconnect's state at power-fail. The SP waits for the IOP to set the SP's send channel flag and is then ready to perform final restoration of the interconnect. If both processors were in their DMA routines or one was and the other had not yet reached its, then the aborted transfer must be restarted. The SP either clears its send channel flag if the transfer was from it to the IOP or leaves it set if the transfer was from the IOP. If the SP was in its DMA routine, and the transfer must be restarted, it restores the DMA parameters; if the transfer was completed it leaves the routine's length parameter at zero. If neither processor was in its DMA routine but the SP's send channel flag was clear, it restores the word from SCHNO and issues the 'STC n,C' to transmit it. Finally, the SP automatically places the current value of RCHNO on its receive channel, sends it to the IOP (which is waiting to restore its send channel), and increments POWFF to 2 to signal restoration of the interconnect.

The SP now completes preparations for restarting an aborted inerconnectterconnect DMA transfer. It checks the return address of each interrupt routine, except those which were identified as interrupting other interrupt routines in the logic above, to see if it lies within the critical portion of the transfer routine. For those that do, the return address is replaced with the transfer restart instruction's address. This will have no effect except for the one (if any) active routine which actually did interrupt from the transfer routine. If the transfer had completed, the length word was left at zero and the restart will not invoke DMA. If a transfer was aborted, then the parameters were restored above and the transfer will be repeated correctly. The remainder of the SP routine consists of performing the out-of-line device recovery routines, resetting POWFF to 0, restoring the hardware registers and interrupt system flag, and resuming time-sharing.

Possible DMAFL Combinations

SP	IOP	
00	00	Neither processor in its transfer routine.
00	01	IOP doing transfer, SP not yet to its transfer routine.*
01	00	SP doing transfer, IOP not yet to its transfer routine.*
01	01	Both SP and IOP in their transfer routines.*
01	10	SP in its transfer routine but transfer is complete.
10	01	IOP in its transfer routine but transfer is complete.
10	10	Neither processor in its transfer routine.
10	11	IOP doing transfer, SP not yet to its transfer routine.*
11	10	SP doing transfer, IOP not yet to its transfer routine.*
11	11	Both SP and IOP in their transfer routines.*
11	00	SP in its transfer routine but transfer is complete.
00	11	IOP in its transfer routine but transfer is complete.

*Transfer will be restarted. The SP must set its send channel flag/ IOP's receive channel flag according to the transfer direction.

Imposible DMAFL Combinations

SP:	00	01	10	11
IOP:	10	11	00	01

```
0001  *
0002  *  THE FOLLOWING ARE MODELS FOR ROUTINES TO SHUTDOWN AND RESTART
0003  *  THE TSB/G IOP.  THEY ARE MERELY SKELETONS ILLUSTRATING THE
0004  *  PROTOCOL AND TIMING NEEDED TO COORDINATE WITH THE SP'S ROUTINES.
0005  *  RC IS THE SELECT CODE OF THE INTERCONNECT RECEIVE CHANNEL.
0006  *  SC IS THE SELECT CODE OF THE INTERCONNECT SEND CHANNEL.
0007  *
0008  ***                                          ***
0009  **                                            **
0010  *  POWER-FAIL/POWER-UP RECOVERY ROUTINE       *
0011  **                                            **
0012  ***                                          ***
0013  *
0014  *  THIS ROUTINE PROVIDES AN ORDERLY SHUTDOWN OF THE I/O PROCESSOR
0015  *  WHEN POWER FAILS.  WHEN POWER RETURNS IT RESTORES THE PROCESSOR
0016  *  TO AN EQUIVALENT STATE, RESTARTS THE INTERCONNECT IN CONCERT WITH
0017  *  THE SYSTEM PROCESSOR, AND RESTARTS THOSE PERIPHERALS FOR WHICH
0018  *  RECOVERY IS POSSIBLE.  POWER FAILURES DURING A RECOVERY ABORT
0019  *  IT; THE POWER-UP PORTION IS RESTARTED AFRESH WHEN POWER RETURNS
0020  *  AGAIN.
0021  *
0022  POW    NOP
0023         SFC 4          IS INTERRUPT FROM POWER-UP?
0024         JMP POWUP      YES
0025  *                                  *
0026  **  SHUTDOWN THE I/O PROCESSOR    **
0027  *                                  *
0028         DST POWT1      TEMPORARILY SAVE
0029         ERB,RLS          (A) AND (B),
0030         SOC                (E) AND (O)
0031         INB                  REGISTERS
0032  *
0033  *  CHECK IF INTERRUPT OCCURRED WHILE RECOVERING FROM A PREVIOUS
0034  *  POWER FAILURE.  IF SO, THERE HAS BEEN NO SYSTEM ACTIVITY AND
0035  *  THE ORIGINAL SHUTDOWN STATUS IS STILL VALID WITH ONE POSSIBLE
0036  *  EXCEPTION.  IF THE SP COMPLETED ITS RECOVERY BEFORE THE NEW
0037  *  FAILURE AND RETURNED TO ITS RECEIVE DRIVER, IT MIGHT HAVE SET
0038  *  THE IOP'S SEND CHANNEL FLAG.  THIS CANNOT HAPPEN AS LONG AS
0039  *  POWFF=1.  THE FLAG'S VALUE WILL NOT OTHERWISE CHANGE.  THE
0040  *  IOP DOES NOT RECORD ITS RECEIVE CHANNEL FLAG, TRUSTING THE
0041  *  SP TO RESET IT CORRECTLY.
0042  *
0043         LDA POWFF      FAILED FROM POWER-UP ROUTINE
0044         SLA              BEFORE INTERCONNECT RESTORED?
0045         JMP POWD3      YES, RETAIN PREVIOUS STATUS
0046         SZA            NO, FROM OUT-OF-LINE RECOVERY?
0047         JMP POWD2      YES, CHECK SEND CHANNEL ONLY
0048         LDA POW        NO, COULD INTERRUPT
0049         ADA POWR1        BE FROM IN-LINE
0050         SSA                POWER-UP CODE?
0051         JMP POWD1      NO
0052         ADA POWR2      YES
0053         SSA            IS IT?
0054         JMP POWD2      YES, CHECK SEND CHANNEL ONLY
0055  *
0056  *
0057  *
0058  *
0059  *
```

```
0060  *
0061  *  POWER FAILED DURING NORMAL SYSTEM ACTIVITY.
0062  *  RECORD THE I/O PROCESSOR'S STATUS.
0063  *
0064  POWD1 STB POWEO      SAVE
0065        DLD POWT1        REGISTERS
0066        DST POWAR
0067        LDA POW        SAVE POWER-UP
0068        STA POWRA        RESTART ADDRESS
0069        LIA 1          SAVE THE
0070        STA POWSW        SWITCH REGISTER
0071        LDB CLF0       NOTE IF
0072        SFC 0            INTERRUPT SYSTEM
0073        LDB STF0           TO BE ON OR OFF
0074        STB POWIS            AFTER RECOVERY
0075  POWD2 CLA            SAVE THE STATE
0076        SFC SC           OF THE INTERCONNECT
0077        INA                SEND CHANNEL FLAG
0078        STA POWFL
0079  *
0080  POWD3 CLC 4          ENABLE INTERRUPT AND
0081        HLT 4            WAIT FOR POWER-UP
```

```
0001  *                                                     *
0002  **  RESTORE STATUS AND RESTART I/O PROCESSOR  **
0003  *                                                     *
0004  POWUP CLA,INA        NOTE INTERCONNECT STATUS NOT
0005        STA POWFF        RESTORED IF RECOVERY ABORTED
0006        STC 4          ENABLE POWER-FAIL INTERRUPT
0007        CLF RC         READY RECEIVE CHANNEL
0008   CLF0 CLF 0          INHIBIT INTERRUPTS
0009  *  THE SP REQUIRES A CERTAIN AMOUNT OF TIME TO SHUTDOWN, ENABLE
0010  *  A POWER-UP INTERRUPT, RESPOND TO IT, AND CLEAR ITS RECEIVE
0011  *  CHANNEL FLAG.  THE WAIT LOOP BELOW KEEPS THE IOP FROM SENDING
0012  *  ITS DMA-ON-INTERCONNECT STATUS UNTIL THE SP IS READY.  THE
0013  *  SP WILL THEN WAIT FOR THE IOP'S RESPONSE, SYNCHRONIZING BOTH
0014  *  INTERCONNECT RESTORATION ROUTINES.  IF THE WAIT TIMES OUT,
0015  *  THEN THE SP IS NOT CURRENTLY TIME-SHARING (SHUTDOWN OR
0016  *  CRASHED) AND THE IOP SHOULD RESTORE ITSELF TO THE IDLE
0017  *  (POST-SYSTEM SHUTDOWN) STATE.
0018  *
0019        CLB            SP 'NOT THERE'
0020        INB,SZB,RSS      TIME-OUT?
0021        JMP << ? >>    YES, ACT AS IF 'SSD' RECEIVED
0022        SFS RC         NO, WAIT FOR SP'S
0023        JMP *-3          DMA-ON-INTERCONNECT STATUS
0024        LIB RC         SAVE
0025        STB POWT1        IT
0026        OLD DMAI2      MOVE LAST DMA DIRECTION BIT
0027        RRR 15           TO A(0) AND DMA ROUTINE FLAG
0028        AND =7             (MODULO 4) TO A(2) AND A(1)
0029        OTA SC         INFORM SP OF IOP'S
0030        STC SC,C         DMA-ON-INTERCONNECT STATUS
0031  *
0032  *  THE SP WAITS FOR THE IOP TO RESTORE THE LAST WORD SENT ON THE
0033  *  IOP'S RECEIVE CHANNEL (OUTSIDE OF A DMA TRANSFER).  THE SP IS
0034  *  RESPONSIBLE FOR RESETTING THE IOP'S RECEIVE CHANNEL FLAG OR
0035  *  LEAVING IT CLEAR AS DICTATED BY THE POWER-DOWN STATUS.
0036  *
0037        LDB RCHNA,I    RESTORE THE LAST WORD SENT
0038        OTB RC           (OUTSIDE OF A DMA TRANSFER)
0039        STC RC,C           AND CLEAR THE RECEIVE FLAG
0040        LDB POWT1
0041        RRS            REMOVE DMA
0042        ARS              DIRECTION BITS
0043        XOR B
0044        SLB            WAS IOP'S DMA ROUTINE FLAG ODD
0045        RAR,SLA          & SP IN OR BEFORE ITS ROUTINE?
0046        CLA,RSS        NO
0047        CCA            YES, NOTE DMA TRANSFER
0048        STA POWT2        TO BE RESTARTED
0049  *
0050  *
0051  *
0052  *
0053  *
0054  *
0055  *
0056  *
0057  *
0058  *
0059  *
```

```
0060  *
0061  *   THE SP WILL RESTORE THE SP SEND/IOP RECEIVE FLAGS EITHER AS AT
0062  *   POWER-DOWN OR AS NEEDED TO RESTART A DMA TRANSFER.  THEN IT
0063  *   RESTORES THE LAST WORD SENT TO THE IOP'S SEND CHANNEL.  THE IOP
0064  *   MUST LEAVE ITS SEND CHANNEL FLAG SET OR CLEAR IT AGAIN, AS
0065  *   DICTATED BY ITS POWER-DOWN STATUS.  THE REMAINDER OF THE SP'S
0066  *   RECOVERY ROUTINE IS LONG ENOUGH TO ENSURE THAT THE IOP HAS TIME
0067  *   TO MAKE THIS DECISION BEFORE THE SP RESUMES TIME-SHARING.
0068  *
0069        LDA POWFL      LOAD STATE OF SEND CHANNEL FLAG
0070        SFS SC         WAIT FOR
0071        JMP *-1          HANDSHAKE
0072        SLA            WAS SEND FLAG CLEAR?
0073        JMP POWU1      NO
0074        LDB SCHNA,I    YES, RESTORE
0075        OTB SC           THE LAST WORD SENT
0076        STC SC,C           AND CLEAR THE FLAG
0077  POWU1 CLC SC         INHIBIT SEND CHANNEL INTERRUPTS
0078        ISZ POWFF      NOTE INTERCONNECT RESTORED
0079  *
0080  *   PERFORM DEVICE RECOVERY ROUTINES
0081  *
0082  *
0083  *   << APPROPRIATE CODE >>
0084  *
0085        ISZ POWT2      REPEAT DMA ON INTERCONNECT?
0086        JMP POWU2      NO
0087  < APPROPRIATE CODE> YES
0088  POWU2 CLA            BACK
0089        STA POWFF        IN-LINE
0090        LDA POWSW      RESTORE THE
0091        OTA 1            SWITCH REGISTER
0092        LDA POWEO      RESTORE
0093        CLO              (E)
0094        SLA,ELA            AND
0095        STO                  (O)
0096        DLD POWAB      RESTORE (A) AND (B)
0097  POWIS NOP            'STF 0' OR 'CLF 0'
0098        JMP POWRA,I    RESUME TIME-SHARING
0099  *
0100  POWR1 ABS -POWUP
0101  POWR2 ABS POWUP-POWR1
0102  *
0103  POWFF OCT 0          =0 WHEN NOT IN POWER-UP ROUTINE
0104  *                    =1 BEFORE INTERCONNECT RESTORED
0105  *                    =2 DURING OUT-OF-LINE RECOVERY
0106  POWT1 BSS 2
0107  POWT2 EQU *-1
0108  POWFL BSS 1          =1 IF SEND CHANNEL FLAG WAS SET
0109  POWSW BSS 1          SAVES SWITCH REGISTER
0110  POWRA BSS 1          SAVES POWER-UP RESTART ADDRESS
0111  POWEO BSS 1          SAVES (E) AND (O) REGISTERS
0112  POWAB BSS 2          SAVES (A) AND (B) REGISTERS
0113  *
0114  RCHNA DEF RCHNO      => LAST WORD SENT ON RC
0115  SCHNA DEF SCHNO      => LAST WORD SENT ON SC
0116  CLF0  CLF 0
0117  STF0  STF 0
```

```
0001  *                                  *
0002  **  INTERCONNECT DMA ROUTINE   **
0003  *                                  *
0004  *
0005  *  PERFORMS A DMA TRANSFER FROM (TO) THE SP TO (FROM) THE IOP.
0006  *  THE RECEIVE CHANNEL FLAG MUST BE CLEAR BEFORE ENTRY TO THIS
0007  *  ROUTINE IF THE TRANSFER IS FROM THE SP ( B(15) = 1 ).
0008  *  ON ENTRY (A) = TOTAL TRANSFER LENGTH (POSITIVE WORDS)
0009  *           (B) = BUFFER MEMORY ADDRESS (BITS 14-0) AND
0010  *                 TRANSFER DIRECTION (BIT 15 = 1 IF FROM SP)
0011  *  IF A POWER-FAIL ABORTS A TRANSFER, IT IS RESTARTED FROM
0012  *  THE BEGINNING.
0013  *
0014  DMAXR NOP
0015        CMA,INA        NEGATE TRANSFER LENGTH
0016        DST DMAT1      SAVE FOR POWER-FAIL RESTART
0017        ISZ DMAFL      NOTE IN ROUTINE (DMAFL NOW ODD)
0018  *
0019  *  RESTART AN INTERRUPTED TRANSFER BY 'DLD DMAT1' AND
0020  *  JUMPING TO THE FOLLOWING LABEL.
0021  *
0022  DMXRS SZA,RSS        ZERO-LENGTH TRANSFER?
0023        JMP DMAX1      YES
0024        STC 3          NO, OUTPUT DMA
0025        OTA 3            WORD COUNT
0026        CLC 3          OUTPUT DIRECTION BIT
0027        OTB 3            AND BUFFER ADDRESS
0028        LDA DMACW      ASK FOR 'STC' ON EACH WORD
0029        OTA 7            AND 'CLC' ON TERMINATION
0030        STC 7,C        START DMA
0031        SFS 7          WAIT FOR
0032        JMP *-1          COMPLETION   CLC 7
0033  DMAX1 ISZ DMAFL      ADVANCE ROUTINE FLAG TO EVEN
0034        NOP            (ROLLOVERS DO OCCUR)
0035        JMP DMAXR,I
0036  *
0037  *  DMAFL MUST FOLLOW DMAT2 FOR POWER-FAIL PURPOSES
0038  *
0039  DMAT1 BSS 2          SAVES TRANSFER LENGTH
0040  DMAT2 EQU *-1        SAVES BUFFER ADDRESS
0041  DMAFL OCT 0          ZERO WHENEVER SP SENDS 'INI'
0042  DMACW ABS 120000B+RC
```

Power-up Interconnect Restoration

```
    IOP                                  SP

CLF RC          ready receive channel    CLF CHI ready receive
 .                                        .
 .                                        .
 .                                        .
DLD DMATZ       prepare                  DLD DMAT4 prepare
RRR 15           interconnect            RRR 15     interconnect
AND =7            DMA info               AND .+7     DMA info
CLB                                      SES CH1   wait for
                                         JMP *-1     handshake
JMP *-5
OTA SC          send it
OLC SC
STC SC,C        ------DMA info-----      LIB CH1   load IOP's info
INB,SZB,RSS
<<time out>>
SFS RC          wait for
                 handshake

LIB RC                DMA info           OTA CHZ   send
                ------------------       STC CHZ,C  DMA info
STB POWT1       save SPLs info           SFS CHZ   wait for
LDB RCHNA,I     restore                  JMP *-1     handshake
OTB RC           receive channel
STC RC,C        last sent word on RC     CLE,ERB   check for
                ------------------
LDB POWT1                                          DMA restart
 .              Check for                {LDB SCHNA,I restore}
 .               DMA restart             {OTB CHZ      send  }
 .                                       {STC CHZ,C    channel}
 .                                        .
 .                                        .
 .                                        .
LDA POWFL
SFS 9C          wait for                 CLC CHZ
JMP *-1          handshake               LDB RCHNA,I restore
                                         OTB CH1      receive

SLA             last-sent word on CH1    STC CH1,C     channel
                <------------------
                                         ISZ POWFF note inter-
                                                   connect restored
LDB SCHNA,I     restore                   .
OTB SC           send                     .
STC SC,C          channel                 .
JMP *+4
CLC SC
```

```
ISZ POWFF          note interconnect
                    restored
  •
  •
  •
```

2000 SYSTEM TABLES

I. DIRECTORY

The Directory is a table which contains all necessary information about each program or file in the system library. It resides on the disc and may occupy from 1 to 80 tracks, depending upon how many discs there are specified (maximum is 10 per disc) by the operator at load time. A core resident table called DIREC contains information on the directory itself.

Each directory track occupies 32 contiguous blocks. A directory entry consists of 12 words. The entry format is detailed on the following page. The directory entries are kept sorted on words 0-3. Bit 15 of words 1 through 3 are not considered in the sorting. Names of fewer than 6 characters are filled out with spaces. The last reference date is the most recent date on which the program or file was referred to, while the last change date is the most recent date on which it was altered. The directory contains two pseudo entries which are the first and last entries in the table.

DIRECTORY
ENTRY
FORMAT

NORMAL ENTRY

First Entry	Last Entry	WORD	CONTENT
0	-1	0	USER ID
0	-1	1	PROGRAM — BIT 15 = 1 IF AN ASCII FILE
0	-1	2	OR FILE — = 1 IF A FILE, 0 IF A PROGRAM
0	-1	3	NAME — = 1 IF SEMI-COMPILED, 0 IF UNCOMPILED
0	-1	4	PROGRAMS - START OF PROGRAM POINTER/ FILES - RECORD SIZE
0	0	5	LAST REFERENCE DATE (YEAR - BITS 15:9, DAY - BITS 8:0)
-1	-1	6	LAST CHANGE DATE (HOUR OF THE YEAR)
0	0	7	PROGRAM STATUS BIT 15 DEVICE/FILE SUPPORTS INPUT 14 DEVICE/FILE SUPPORTS OUTPUT 13 PROGRAM/FILE HAS PFA 12 FILE HAS MWA ACCESS 11 PROGRAM/FILE'S OWNER HAS FCP 10 UNUSED 9 UNUSED 8 UNUSED 7 UNUSED 6 UNUSED 5 UNUSED 4 UNUSED 3 PRIVATE 2 LOCKED 1 PROTECTED 0 UNRESTRICTED
-1	-1	8	DISC ADDRESS IF — =1 INDICATES NON-SHAREABLE DEVICE
			OR
0	0	9	DISC FILE/PROGRAM — DEVICE DESIGNATOR, 3 5-BIT FIELDS
0	0	10	USED ONLY BY THE LOADER
0	0	11	LENGTH (-WORDS FOR PROGRAM, + RECORDS FOR FILE) (0 FOR NON-SHAREABLE DEVICE)

II. DIREC

DIREC is a 560 word memory resident table which contains information about the directory. DIREC entries consist of the first four words of the first entry on each of the directory tracks, as well as the track disc addresses and lengths. This facilitates rapid location of directory track entries by system programs using a method which is similar in concept to the use of the words at the top of dictionary pages.

The DIREC Table has the following structure:

DIREC +	0	-length in words of first directory track
	1-4	same as first four words of first directory track
	5-6	disc address of first directory track
	7-13	same as 0-6 but applied to 2nd directory track
	•	
	•	
	•	
	553-559	same as 0-6 but applied to 80th directory track

A disc address of 0 implies that there is no such directory track. When word 0 is 0, words 1-4 are meaningless.

When generating a system or reloading from mag tape, the system operator has the opportunity to specify the number of directory tracks per disc, in the range of 1-10, which is saved in NDIRT. The total number of directory tracks is this number times the number of discs on the system. Each directory track may contain as many as 8184 words = 682 directory entries.

III. ID TABLE

The ID table (IDT) is a disc resident table of from 1 to 10 tracks which contains one 12-word entry for each ID code on the system. The entries are kept sorted according to the ID codes. An entry has the following format:

WORD	CONTENT
0	USER ID
1	PASSWORD (FILLED
2	WITH 0'S IF FEWER
3	THAN SIX CHARACTERS)
4	TIME ALLOWED (IN MINUTES)
5	TIME USED (IN MINUTES)
6	DISC SPACE ALLOWED (IN BLOCKS)
7	DISC SPACE USED (IN BLOCKS)
8	USER CAPABILITIES - IF THE BIT IS SET, THE USER HAS THE CAPABILITY

BIT	MEANING IF SET
15	UNUSED
14	RP - READER PUNCH
13	JT - JOB TRANSMITTER
12	JL - JOB LINE PRINTER
11	JP - JOB PUNCH
10	JI - JOB INQUIRY
9	JM - JOB MESSAGE
8	MT - MAGNETIC TAPE UNIT
7	PP - PAPER TAPE PUNCH
6	PR - PAPER TAPE READER
5	LP - LINE PRINTER
4	CR - CARD READER
3	UNUSED
2	PFA - PROGRAM/FILE ACCESS
1	FCP - FILE CREATE/PURGE
0	MWA - MULTI WRITE ACCESS

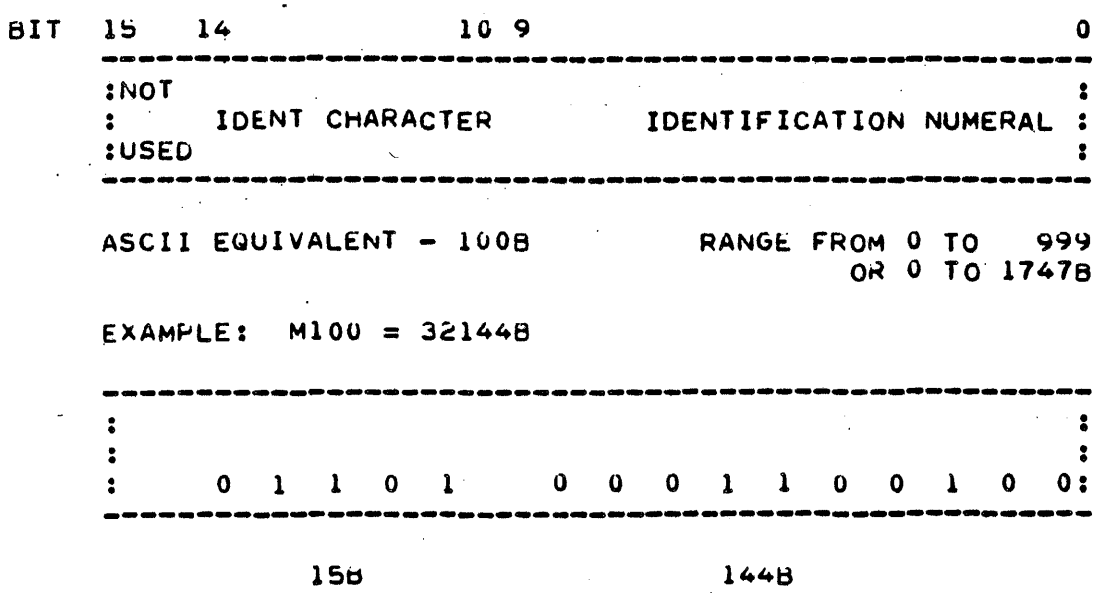
9-10	UNUSED

IV. IDEC TABLE

IDEC is a 40 word memory resident table which contains information about the id table. There is one 4-word entry for each id track and each entry has the following format:

word	
0	FIRST ID ON THE TRACK
1	DISC ADDRESS
2	OF THE TRACK
3	LENGTH IN -WORDS

ID FORMAT



15B + 100B = 115 : ASCII "M"
0144B = 100
10

IV. SWAP_AREAS_TABLE

The Swap Areas Table (SAT) is a disc resident table of 64 words long which contains 32 two-word entries, one for each swap track on the system. The 11b entry is the disc address of the 11b swap track. A zero entry indicates the absence of that swap track.

This table is built, used and updated by the Loader only.

V. ADT

The available disc table (ADT) is a disc resident table which contains one three-word entry for each area of the disc which is unallocated. Each ADT track occupies 32 contiguous blocks. An entry has the following form:

```
WORD    0     disc
        1       address
        2     length of area in blocks
```

There is one ADT track for each disc on the system and only entries for one particular disc appear on a track. The first 4 blocks of each disc are used by the system and are therefore always unavailable. Thus, there are no contiguous areas which overlap discs.

The 24-word available disc table address table (ADTAT) in the EQT table refers to ADT in the following format:

```
WORD    0-1   disc address of first track
          2   length of first track in - words
        .
        .
        .
       21-22  disc address of the 8th track
        23    length of the 8th track
```

VI. LOCKED_BLOCKS_TABLE

The Locked Blocks Table is a disc-resident table which resides in the 256 words of block 3 of each disc. It contains one two-word entry for each area of the disc that has been MLOCKED. An entry has the following format:

WORD	0	disc address relative to this disc
	1	length of area in blocks

The rest of the table is a zero filled.

The disc address is stored as if the disc were logical disc 0. The Locked Blocks Table is cleared only when it is determined during the loading procedure that the disc does not have a valid TSB label and the operator requests that one be written. This means that the packs "remember" which blocks are unavailable even if a different 2000 system is loaded.

VII. FUSS_TABLE

The FUSS table is a 1024 word disc resident table. It is divided into 32 sections of 32 words each. The 32 words in each section are the two word disc address of the 16 possible files currently being accessed by the user corresponding to that section. Addresses of 0 indicate no file or a non-disc ASCII file entry. The high word of the disc address may have one or two of the high bits set indicating:

bit	meaning if set
15	file is read-only to this user
14	file was locked by this user
13	WR restriction was imposed by this user
12	RR restriction was imposed by this user

If bits 13 and 12 are not set, an NR restriction was imposed by the user. The FUSS table is written to the disc in two halves by the LOADER. It must occupy two contiguous disc blocks.

The reasons for maintaining the FUSS table are:

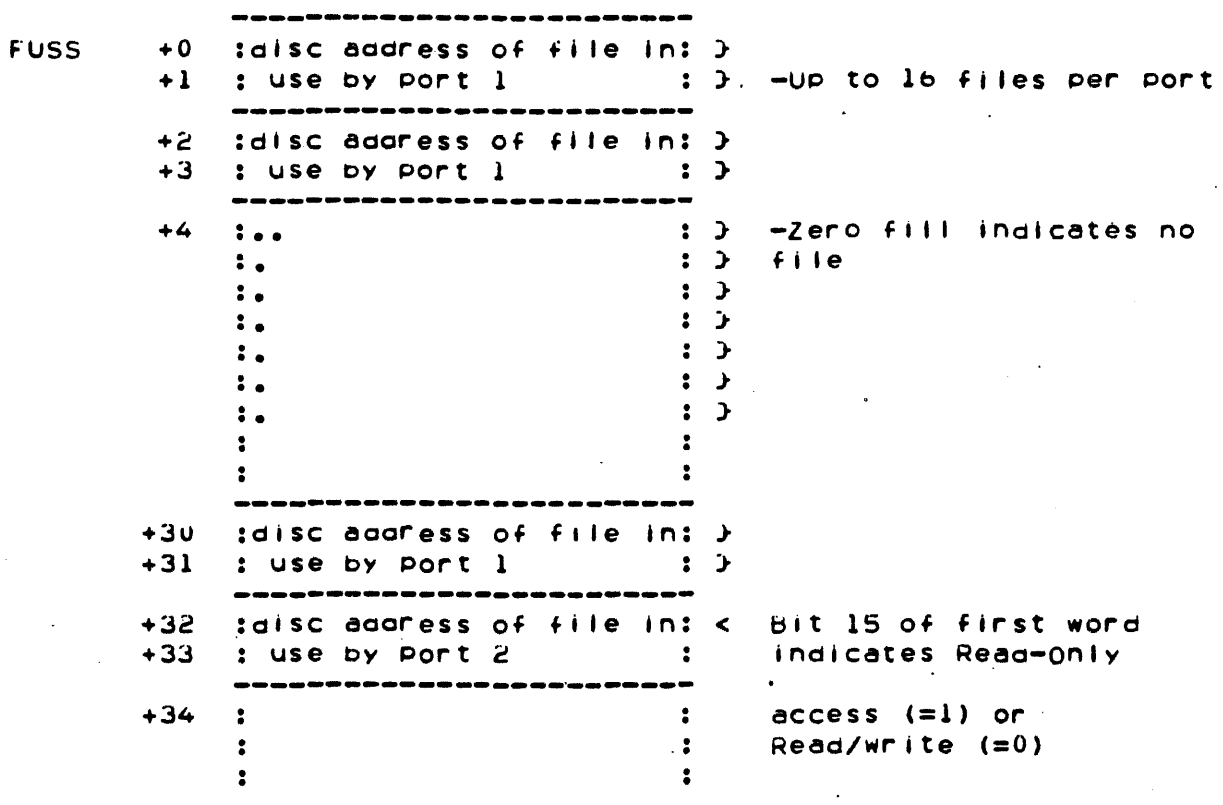
(1) To prevent simultaneous write-access by two or more users unless the file has MWA and no user has specified WR or RR on the file.

(2) To allow co-operating users of a file exclusive use of a file via the LOCK and UNLOCK statements.

(3) To pevent moving or removing an active file in the routines PURGE and MLOCK. No check is made for the routinesa COPY and BESTOW.

A user's FUSS (i.e. his section of the FUSS table) is set by the FILES routine, which is called from BASIC at the statement. Indivioual entries in a user's FUSS are changed by the execution of ASSIGN statements. The user's FUSS is cleared by HELLO, BYE, and normal or abnormal program termination.

FUSS TABLE




```
FUSS    BSS 32      Port 0 Entries  }
        BSS 32      Port 1 Entries  }
        BSS 32      Port 2 Entries  }            } Initialized to
           .                        } SECTION   } zeroes and disc
           .                        }   ONE     } space reserved
           .                        }           } just prior to
        BSS 32      Port 15 Entries }           } library being
        BSS 32      Port 16 Entries             } written on disc
           .                        } SECTION   } on disc to ensure
           .                        }   TWO     } sections one and
           .                        }           } two are written on-
        BSS 32      Port 31 Entries }           } to contiguous disc
                                    }             sectors--will be
                                                  read into user area
                                                  during later system
                                                  usage.
```

- Unprotected files stored under system or Group ID's will be accessed as read-only by other ID's.

VIII. COMTABLE

The COMTABLE is a list of all user and system commands containing their ASCII codings and disc locations or core addresses. The structure of the COMTABLE is as follows:

COM1	codes for commands which are executed immediately by the system	
COM2	codes for commands which are executed by BASIC	
COM3	user commands which are executed by disc resident programs	
COM4	system commands -- all are executed by disc resident programs	
COM5	starting addresses for those commands which are listed under COM1 AND COM2	
COM6	disc addresses for those commands which are listed under COM3 and COM4	(this section is filled by the loader)

Since each command is recognized only by its first 3 letters, the scanner converts each letter into a number from 0 to 31(8), and then packs the three codes into one word as three 5-bit bytes. In addition, bit 15 is set for system commands. Codes of -1 in tions 2, 3, and 4 do not correspond to any possible 3-letter code. Their purpose is to generate room in COM6 for disc addresses of routines that are called indirectly, or for tables like FUSS.

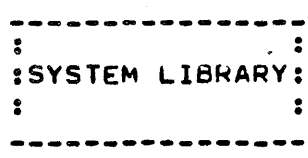
```
          15  14          10 9          5 4           0
COMMAND   ------------------------------------------
WORD      :      BYTE ONE      BYTE TWO    BYTE THREE :
FORMAT    ------------------------------------------
```

first 3 bytes of command name identify the command — set if command must be from system console.

BYTE = ASCII CHARACTER VALUE - 101B

COMMAND TABLE FORMAT

Program Name of Area	Description of Table Contents		
COM1	Part I -	SECTION I-	core resident user commands, executed immediately.
COM2	-	SECTION II-	core resident user commands, need user program in core before execution can proceed.
COM3	-	SECTION III-	disc resident commands --user commands
COM4			--system console commands
COM5	PART II	-	core starting addresses for PART I, SECTIONS I & II. 2 word disc addresses for all disc resident commands, and error messages.



CORE_RESIDENT_USER_COMMAND_
EXECUTED_IMMEDIATELY

SCRATCH	SCR
TAPE	TAP
KEY	KEY

CORE_RESIDENT_USER_COMMANDS
NEED_USER_PROGRAM_IN_CORE_BEFORE
EXECUTION_CAN_PROCEED

RUN	RUN
LIST	LIS
PUNCH	PUN
LOAD	LOA

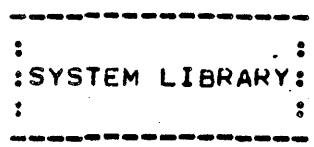
```
----------------
:              :
:SYSTEM LIBRARY:
:              :
----------------
```

DISC-RESIDENT
SYSTEM-COMMANDS-FROM-USER-CONSOLE

DUMP	DUM
DIRECTORY	DIR
REPORT	REP
STATUS	STA

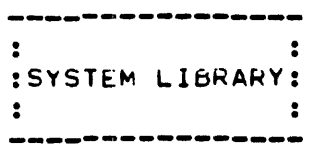
DISC-RESIDENT
SYSTEM-COMMANDS-FROM-SYSTEM-CONSOLE

ANNOUNCE	ANN
REPORT	REP
RESET	RES
CHANGE	CHA
DIRECTORY	DIR
STATUS	STA
SLEEP	SLE
HIBERNATE	HIB
NEWID	NEW
KILLID	KIL
MUNLOCK	MUN
MLOCK	MLO
COPY	COP
RESTOW	RES
PURGE	PUR
ROSTER	ROS
PHONES	PHO
BREAK	BRE
DEVICE	DEV
ASSIGN	ASS
RJE	RJE
DUMP	DUM
DISCONNECT	DIS
BANNER	BAN
AWAKE	AWA



DISC_RESIDENT_USER_COMMANDS

SAVE	SAV
CSAVE	CSA
GET	GET
APPEND	APP
HELLO	HEL
BYE	BYE
PURGE	PUR
RENUMBER	REN
NAME	NAM
CATALOG	CAT
LIBRARY	LIB
GROUP	GRO
DELETE	DEL
TIME	TIM
LENGTH	LEN
ECHO	ECH
MESSAGE	MES
EXECUTE	EXE
DEVICE	DEV
UNRESTRICT	UNR
PROTECT	PRO
LOCK	LOC
PRIVATE	PRI
SWA	SWA
MWA	MWA
PAUSE	PAU
CREATE	CRE



OTHER DISC RESIDENT BLOCKS

FILES PROCESSOR
ASSIGN STATEMENT PROCESSOR
SYSTEM STATEMENT PROCESSOR
LOCK/UNLOCK STATEMENT PROCESSOR

SAVE OVERLAY
STATUS (USER CONSOLE) OVERLAY
STATUS (SYSTEM CONSOLE) OVERLAY
MLOCK OVERLAY
DUMP OVERLAY

LENGTH TABLE
FUSS TABLE - 2 PARTS
ERROR MESSAGE TABLES

TAPE MODE CLEANUP

IX. LOGGR

LOGGR is a 64-word queue which contains codes for printing LOGON/OFF messages. Entries are placed on the queue by HELLO, BYE, and SLEEP. Each entry consists of 2 words, with the following format:

WORD 0: user id (BIT 15=0 for ON, 1 for OFF)
1: bits 15-5 = 60 x hrs + mins
bits 4-0 = terminal number

The representation of a user id is as follows:

BITS 14-10 = letter (A =1, B =2, ..., Z = 32(8))
BITS 9-0 = (0-999)

The following variables are relevant:

LOGCT = # of unprocessed entries in LOGGR
LOGP1 = points to word 1 of last processed entry
LOGP2 = points to word 1 of last unprocessed entry

Note that LOGCT = 0 <=> LOGP1=LOGP2

8.3 TELETYPE TABLE

This set of 32 tables, one for each user, contains relevant information about the various terminals. The structure of the tables is as follows:

Word Number	Word Name	Description
0	?FLAG	User bit flags
1	?TNUM	Port number in bits 12 to 6
2 3	?DISC	Address of Swap area on disc of program (2 words)
4	?PROG	Points to last used address of program when in core
5	?ID	User ID (=0 if no current user)
6 7 10	?NAME	} 6 character (or less) program name
11 12	?TIME	} Starting time (set to DATIM and DATIM+1 at LOGON)
13	?CLOC	User's RUN time time-out clock
14	?RSTR	Restart address for suspended programs
15	?STAT	User status
16	?LINK	Points to ?LINK of next port on Queue
17	?PLEV	Priority of user on Queue If user not in the queue, contains program type (temporary storage of ?STAT)
20	?OREC	Record size of OUT= file
21	?OUTP	Logical unit number or disc address of OUT= file
23	?NREC	Number of records in OUT= file
24	?PRID	Owner ID of program in swap area
25	?RTIM	Response time for ENTER

26	?TEMP	}	Temporaries
27		}	used by
30		}	Library routines

?FLAG BIT DEFINITIONS

Bit Number	Name	Description
0	TERR	Error detected on tape input by user
1	CFLAG	Program in swap area is compiled
2	HFLAG	HELLO is running
3	TAPEF	User is in tape mode
4	UNABT	Unable to abort user when set Is set - when compilation is occuring - when compiler executes a disc routine (allows port a chance to abort just prior to running of disc routine) - FILES - CHAIN - ASSIGN - when Update Last Reference Date for files is being run Is not set, but similar effect (i.e., unable to abort) when Library routines are running - except - LIBRARY - CATALOG - GROUP from user console { - DIRECTORY - REPORT - STATUS
5	OUTWT	User output buffer full-output wait
6	PACT	Port needs to be processed by scheduler
7	ABTRY	Abort attempt that has been delayed due to UNABT or should just be flagged due to PBFLG

8	DFCHK	Dirty files flag-used for last change
9	CHNFG	CHAIN is running-inhibit certain activities
10	SPACG	List has allocated an output buffer for the user
11	MBUST	Used to flag disc transfer errors
12	PBFLG	BRK function disabled the break key capability
13	CBFLG	System operator enabled the break key
14	OUT=P	The command running has an OUT= file
15	TRERR	Transmission error on port

?TNUM: Teletype number in bits 12-8; used for sending information to the I/O processor.

?DISC: Disc address of user's swap area

?PROG: When user is on the disc PROG points to the last core location used by the program. When the user is loaded into core, PROG is placed into PBPTR. When he is written back to disc, PBPTR is copied into PROG. BASIC is required to maintain PBPTR as a bound on the core it is using.

?ID: User's id, 0 if none.

?NAME: A three word entry containing the user's pogram name. It is set by the routine NAME & GET & CHAIN, and cleared by HELLO. When fewer than 6 characters are in the name, blanks are appended. Each word of the entry contains a flag in bit 15. Meaning of the flags, if set, are : (1) program locked (2) program private or (3) program protected

?CLOC: This is the time out clock used to determine the length of a user's time slice. See the discussion on scheduling for further information.

?RSTR: This is set, when a user is placed on the queue, to his starting address in core. When the user is actually initiated, RSTR is set to 0. Whenever RSTR = 0, the transfer address of the user can be found in location PREG.

?STAT: Indicates user's status. The user's status is as follows:

Numeric Representation	Name	Description
-4	%PUN	Port is unavailable due to hardware failure.
-3	%ENTO	An ENTER statement has timed out
-2	%DISC	Disconnect this user.
-1	%ABORT	User wants to ABORT
0	%IDLE	Idle - no user, or waiting for commands or syntax

1	%PAUS	Waiting for device attention or PAUSE command timeout
2	%INPT	Waiting for input from user
3	%OUTW	Waiting for output to user to complete
4	%SYNT	Processing user syntax or { FILES { CHAIN { library { ASSIGN { SAVE { CSAVE { LOCK/ { UNLOCK { SYSTEM { BYE { CREATE { PURGE
≥5		Processing a command - value will correspond to command table entries, e.g., 5 = RUN 6 = LIST 7 = PUNCH etc.

?LINK: The LINK words in the tables are used to form a queue of active users. All users whose status is ≥4 are in the queue.
See discussion on scheduling for further information.

?PLEV: This word gives the priority level of the user when the user is on the queue. When the user's status is set to 2 or 3, the previous value of ?STAT is copied into ?PLEV, and the user is removed from the queue. The possible values of ?PLEV are:

Set_by_Routine_Calling_Scheduler

Numerical_Value	----------Description----------
0	Syntax processing Return from input or output wait Return from ENTER time out When COMlIl commands (?PLEV=2) reach the head of the queue

OPEN command when it reaches head of queue
(from ?PLEV=4)
Core resident program to Update Last
Change Date for files
When compiler calls a disc resident
program, the running of which causes
the user ?CLOC to be incremented by one
FILES
CHAIN
ASSIGN

1 COMII Commands - RUN
LIST
PUNCH
XPUNCH

2 COMMIII Commands

4 Compute bound programs, when time-slice
has been exhausted
OPEN command after each group of 400
blocks has been initialized

Takes on value of ?STAT during input/output wait, ENTER wait, or when program to Update Files Change Date is running.

Information of Priority at Time of Entering Scheduler

NP = EP * NPORT/4
NP = new priority
EP = entering priority
NPORT = -(number of ports logged on +1)

?OREC: Logical record size for the ASCII out= file. Bit 15=1 if the file is on disc.

?OUTP: Either (a) logical unit number of the OUT= file, if a non-shareable device or (b) two word disc address if the OUT= file is on disc.

?NREC: Number of records in the OUT= file if disc file.

?PRIV: ID of the owner of the program in the user's swap area. bit 15 = 1 if user has FCP capability.

?RTIM: The length of the time in seconds that it took a user to respond to an <ENTER STATEMENT>.

?TEMP: Used (along with ?RTIM) to save variables when

OPEN, CATALOG, GROUP, LIBRARY, STATUS, DIRECTORY, and REPORT are swapped out.

Associated with each item in these tables is a symbol which is EQUated to the corresponding number of the item. For example:

```
?FLAG EQU 0
?TNUM EQU 1
?TEMP EQU 21B
```

These symbols are primarily used for adjusting pointers to the table. For example, if the B register contains a pointer to the LINK entry of some user, the instruction

```
        ADB .+?ID-?LINK
```

will point B to his ID entry.

. is a symbol located in base page at the 0 entry of a table of constrants from -26 to +51. A word containing the value N, where $-28 \leq N \leq 51$ can be referenced by .+N.

XI. EQUIPMENT_TABLE

The equipment table (EQT) is the area of memory which describes the resources available to the system. It resides in locations 100 - 176 as follows:

100	NIDT	number of ID tracks
101	NDIRT	number of directory tracks per disc
102-131	ADTAT	ADT address table
132-141	DKTBL	There is one word in this area for each of the 8 discs. When the word is zero, the particular disc does not exist. Otherwise, bits 15:8 contain the high priority select code and 7:0 the unit number.
142-143	DADSL	disc address of system library
144	DLNSL	length of system library in + blocks
145-151	SYSID	A ten-character system identification.
152	MAGSC	High priority select code for mag tape. Bit 15 is set if console is a terminet.
153	NPORT	Two's complement of the number of ports on the system. The ports available are numbers 0 thru -NPORT -1.
154	YEAR	Year of the century
155-156	DATIM	Time of year. The first word is the hour of the year, and the second is the number of 100 ms units in the hour minus 36000.
157	HDATE	Hour of year that the system was last hibernated.
160	SLEPT	0 says that the system has been slept, -1 that it has not. This word is modified only by the sleep and reload procedures and insures that the system may not be reloaded from disc if it has not been slept.

Following the equipment table, in locations 161-176 are another set of words which must correspond with the loader. They are defined as follows:

161		Core address in the loader of the final disc bootstrap.
162		Core address of the first loader segment in the Master Segment Table (MST).
163	ERIDX	Pointer to salvage portion of MST
164	MHAD	Core address of the Moving Head Disc Table (MHTBL)
165	GMQBP	Core address of the routine to get a buffer for disc error messages.
166	DISCA	Core address of disc driver entry point.
167	DISCB	Core address of disc driver interrupt entry point.
170	MBUSY	Disc driver busy flag.

171	MWORD	Word count for disc driver.
172	DREDP	Core address of disc driver auto restart entry point (used by powerfail/auto restart routine).
173-174	EQTAD	Disc address of the Equipment Table
175-176		Disc address of the IDEC Table.

XII. MASTER_SEGMENT_TABLE

The Master Segment Table (MST) is a 65-word table resident in the loader. It is the first portion of the bootstrap and is pointed to by the first word of the Equipment Table. The first word of the table contains - the number of system segments. Each group of 4 words following the first word has the following format:

WORD	1	length of segment in -words
	2	absolute, beginning core address of the segment
	3-4	disc address of the segment

There are 16 segments, ordered as follows:

SEGMENT	1	Interrupt locations (2(8) "to" 30(8))
	2	System base page (end of EQT "to" 1777(8))
	3	System linkage area (4002(8) "to" 4017(8))
	4	Disc Error Recovery area (25000(8) to (25777(8))
	5	System segment 1 (end of DIRECT "to" 41777(8))
	6	System segment 2 (42000(8) "to" 51777(8))
	7	System segment 3 (52000(8) "to" 61777(8))
	8	System segment 4 (62000(8) "to" 71777(8))
	9	System segment 5 (72000(8) "to" 77677(8))
	10	Equipment Table (100(8) "to" 176(8))
	11	Direct Table (30401(8) "to" 31530(8))
	13	Loader segment 2 (15500(8) to 23777(8))
	14	Loader Segment 1 (2000(8) "to" 14567(8))
	15	Cold Dump Segment (24000(8) "to" 24765(8))
	16	Disc driver (26000(8) "to" 27677(8))

Note that this includes all core resident portions of the loader and system except for locations 14500(8) "to" 15777(8). The first 1000(8) of these words comprise the disc bootstrap and are resident on blocks 1 and 2 of each disc. Locations 15500(8) "to" 15577(8) may only be used for temporaries.

XIII. MUERTO

MUERTO, as the name suggests, is a table of all system halts. All of the system halts can be referenced by the label "DEATH". MUERTO explains the meaning of each halt and gives an indication of possible recovery or failure. The following is a copy of the table:

2 -	Erroneous, non-recoverable system transfer has occurred.
5 -	A parity error has occurred; check the hardware.
11B -	Unexpected interrupt from the processor interconnect, take a cold dump of this unrecoverable system.
30B -	Disc driver is busy. Recovery not possible.
31B -	Disc called is not pesent. No way to recover.
32B -	Unable to read disc recovery routines into core. No way to recover.
34B -	Disc error causing the system tables to be incompatible. Recovery is out of the question.
36B -	Powerfail has occurred, check restart switch position.
37B -	Bad ADT disc address generated. Recovery - no way.
40B -	Impossible condition occured. No way to recover.
42B -	Owner of a directory entry is not in the IDT. No recovery.
44B -	IOP asked for non-existent RJE command. No recovery.

XIV. MOVING_HEAD_DISC_TABLE

WORD	0-1	Two-word absolute sector number of the first 128-word hardware sector on logical disc 0
	2	Points to select code/unit number in DKTBL for logical disc 0
	3	number of sectors/cylinder
	4	number of sectors/track
	5	Current cylinder position of heads for logical disc 0 (used only for 7900 discs)
	6-11	Same as 0-5 applied to logical disc 1
	•	
	•	
	•	
	42-47	Same as 0-5 applied to logical disc 7

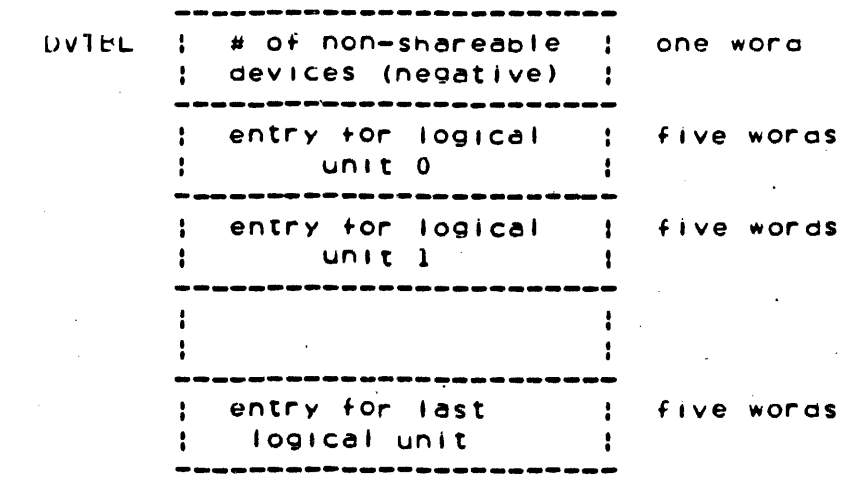
Note that the address in the first two words of each section of the table is a sector address and must be divided by two to obtain the block address. The last entry is followed by a truncated ninth entry, MAXSC, which always contains a double integer specifying the first absolute sector number which does not exist. The actual numbers for the 3 kinds of discs used on ACCESS are as follows:

	2883	7900	7905
WORDS 1-2	0	0	0
	93380(10)	19488(10)	59184(10)
	186760(10)	38976(10)	118368(10)
	280140(10)	58464(10)	177552(10)
	373520(10)	77952(10)	236736(10)
	466900(10)	97440(10)	295920(10)
	560280(10)	116928(10)	355104(10)
	653660(10)	136416(10)	414288(10)
3	460(10)	96(10)	144(10)
4	24(10)	48(10)	

DEVICE TABLE INITIALIZATION

The Non-shareable Device Table is a table of 321 words consisting of the number of non-shareable devices and a five word entry for each device. The I/O Processsor Configurator builds this table from the information supplied by the system operator.

The table can contain up to 64 entries. Each entry is internally referenced by a logical unit number 0 through 63. The table appears as follows:



During system initalization, the loader program will send the IOP a pre-emptory request with send device table sub-code. The IOP will respond with the device table, preceded by the number of ports configured and the number of non-shareable devices specified. The entries in the table have the following fields:

WORD	CONTENT	DESCRIPTION
0	device designator	The designator is composed of two letters and one or two digits. Octal 101 is subtracted from the ASCII value of each letter, allowing coding of the designator as 3 five-bit fields in bits 14-0. The number, which is between 1 and 31, is not coded.
1	select code	Bits 7-0 contain the select code of the device. Bit 15 is 1 if the device supports input, bit 14 is 1 if it supports output, bit 13 is set if the device user is trapping errors and an error occured on his last use of this device, bit 12 is 1 if it supports CTL functions but not output. If bit 12 is set, so is bit 14.

2	maximum record size	The maximum record size that can be requested when using the device.
3	assignment	This entry can have one of the following values: idcode - only users with this idcode can have access 1 - device reserved for RJE 0 - all users can have access -1 - no user can have access
4	tty table	If 0, the device is not busy; if 1, in use by RJE; otherwise it points to the ?FLAG entry of the controlling user.

The contents of the device table can be displayed by use of the DEVICE command.

SUPPLEMENTARY NOTES ON BASIC

I. SYNTAX

The general process of analyzing an input to the language processor is displayed in the flow charts. The annotations in the listing explain the actions of the subroutines, while the core map and sections on internal representation describe the objects/ structures being created or manipulated. The BASIC syntax, in conjunction with the listing, explains the method of identification and recognition of legitimate BASIC statements from the input string.

II. PHASE 2

A. Compilation

The preliminary section of CMPLE prepares for execution of the program following a successful compilation. Null programs require no processing. If a sequence number follows the RUN (e.g., RUN - 220) the interpreter's program counter is set to the first statement whose sequence number equals or exceeds the reference, otherwise it is set to the first statement of the user program. If the common area has not been allocated, ALCOM is called to compute the space needed and move the program accordingly. Common is initialized to -512. If the program is semi-compiled (SPTR=0, SYMTB<>0) we may skip building the symbol table. Otherwise FILTB is set to 0 so PRNST will not terminate compilation by mistaking it for decompilation.

The symbol table is then built as explained in the listing (Refer to the flow chart for general logic flow and to BASIC Variable Storage Allocation fr a visual example). Also, at this time statement number references are replaced by absolute addresses. This is facilitated by dividing the program into 32 parts and building a 64 word table in EKSEC (an unused storage block in the user area on base page) containing the first statement number and address of each part. During compilation SPTR points to the program word being processed. Pointers to <FILES statements> are stored in FLSTS and a count of them is kept in FILCT. An error in compilation will cause a call to DCMPL to restore the source form of the program followed by a call to the error routine. If after a successful compilation at least one <FILES statement> has been found, BASIC calls the system to build the file control blocks, filling all but the fifth, sixth, ninth, tenth, fifteenth, and seventeenth words of each entry.

The symbol routine has two entry points: SSYMT used for functions and simple variables, and ASYMT for array and string variables. Because the dimensionality of an array variable may not be known locally (e.g. MAT A=B) some symbols may have two entries. If this is the case, the "don't know" entry will always be farther down in the table (i.e., have a higher core address) than its dimensioned counterpart.

B. VALUE

VALUE is responsible for detecting deficiencies in the symbol table, allocating storage for the values of symbols initializing the values of all variables except those in common. The process of building the value table is described in the listing. Note that for arrays in common, the declared dimensions in the <COM statement> are checked against those in the common area. If they match and the dynamic dimensions are consistent (i.e. <= declared dimensions) then the values are left alone. Otherwise they are set equal to those in the <COM statement>. For strings, the physical length is checked against the declared length and the logical length tested to be less than or equal to the physical length. If these tests fail, the physical length is set to the declared length and the logical length is set to zero. Smple variables in common are left untouched.

Several errors may be encountered while building the value table. The occurrence of a null symbol (bit pattern of 0) in the symbol table means that an array symbol is used in the program, but never in such a way that its dimensionality can be determined. If the second word of a function entry is zero, no <DEF statement> for that function appears in the program. Arrays of more than 5000 elements are not allowed.

The next step is to allocate two buffers for program input and output. The output buffer has a pseudo file control block which consists of the first eleven words of a file table entry. If the output device is the user terminal, the third word of the block (logical unit number) is set to -2 and the record length set to 72 bytes. If RUN*OUT=file name* has been specified, then the ?OREC and ?OUTP entries of the user teletype table indicate the device and logical record size to be used for the program output device. A buffer large enough to hold one logical record is allocated, except for ASCII disc files which require a 256 word buffer to contain one physical block (ASCII files are blocked). The routine STFCB is responsible for filling the entries in the file control block. If the

program was CHAINed to, and the output device is an ASCII disc file, then the current disc address is set to the first disc block which was not written to by the CHAINing program.

The input buffer only requires a two word control block consisting of the tenth and eleventh words of a file table entry (current buffer pointer and logical end of record). The buffer allocated is the same size as the output buffer.

During program execution, the variables INFIL and OUTTB point to the first word of the "file table entry" for the program input/output devices. The non-existent portions of the entry are not needed and are never referenced. OUTTB points to the entry for the output device. Bit 15 is set if that device is the user terminal. INFIL is set to zero and points to the input device entry only during execution of an ENTER, LINPUT, or READ statement.

NOTE: See ASCII file handling for special use of INFIL and OUTTB

C. Decompilation

Programs are decompiled when an error occurs during compilation, building of the file table, or when the program is to be modified or saved in the user library. Since in the first of these only a portion of the program is compiled, the pointer SPTR is used to determine how much to be decompiled (a fully compiled program always has SPTR pointing to the first word following the program). The program is moved so that SPROG=PBUFF (no common area). The process is explained in the listing.

D. The Routine PRNST

PRNST is used by both CMPLE and DCMPL to scan the program and skip over those portions not affected by compiling. PRNST assumes responsibility for recognizing extra <FILES statements> and <COM statements> that are out of order. If such an error condition is encountered, SPTR is set to point before the statement which caused the error (it hasn't been compiled). Then PRNST calls DCMPL, which calls PRNST. The statement causing the error is not seen this time, so PRNST and DCMPL can exit correctly.

III. EXECUTION

A. Main Loop

Upon completion of the value assignment in phase 2, control transfers to XEC. After printing the program name (unless the program was CHAINed to) XEC proceeds to initialize the file table. A buffer the size of a logical record is allocated for each file, except ASCII disc files which require buffers of 256 words regardless of logical record size. Pointers to the word following the buffer are placed in words nine and ten of the file table. The first word of the disc address of the record in the buffer (word 5) is set to 100000 (octal) to indicate that no record is present. Word 17 is set to zero, indicating that no end-of-record/end-of-file exit has been specified. Word 15 is set to 0 as a null protect mask.

Following the preparation of files the initial execution status is set. The initial execution stacks are claimed from free user space and pointers are set to the first position counter (CHRCT) is set to zero by outputting a carriage return. Phase 2 has already set the BASIC program pointer (PRGCT) to the first statement to be executed.

Execution of a statement simulates the execution of an instruction on a 'BASIC machine'. The sequence number of the statement referenced by PRGCT is saved for possible use by the error routne. PRGCT is advanced to reference the following statement. The type of the current statement is used to branch to the appropriate routine via a jump table. Individual statement routines return to the top of the loop.

B. Statement execution

<LET statement> execution consists simply of evaluating the formula which is known to contain at least one assignment operator and to have type compatibility (numeric vs. string) by its acceptance by phase 1.

<IF statement> execution forks on the symbol following the IF. The construction 'IF END' causes the following: the file reference is evaluated and tested for existence as one of the program's requested files; if a legitimate reference, the statement reference following the THEN is placed in the end-of-file word of the file's table entry. The construction 'IF ERROR' causes

the statement reference following the THEN to be placed in the base page variable UTRAP (which is set to 0 on program termination, abort, or execution of a CHAIN statement). If not 'IF END' or 'IF ERROR', the decision formula is evaluated and if true the statement reference replaces the value of the interpreter's program counter, PRGCT, via the GOTO mechanism.

<GOTO statement> execution consists of choosing a statement reference to replace the program counter. For simple GOTO's this is done trivially; for multi-branch GOTO's this is done by evaluating the index formula and choosing the statement reference in the corresponding list position. If the index value lies outside the list of statement references, the program counter remains unchanged.

<GOSUB statement> execution follows the pattern for the GOTO except that after choosing the new value for the program counter, the old vaue is saved on the return stack (stack overflow generating an error condition).

<FOR statement> execution opens an active program loop. The for-stack is searched for an entry with the same for-variable; if found, the entry is eliminated (i.e., the previous <FOR statement> with this for variable is closed). A new entry is set on top of the for-stack (extending the for-stack by six words if no entry was eliminated) and a pointer to the for variable's value entry is put into word 1. Since the first formula in the FOR contains an assignment operator, the formula evaluator, FORMX, initializes the for-variable when it determines the initial value. A reference to the statement following the <FOR statement> is put into word 6 of the for-stack entry (the start-of-loop address). Words 2 and 3 save the result of evaluating the limit value formula. If a step size formula appears explicitly it is evaluated, otherwise 1.0 is taken as the step size. In either case the value of the step size is left in words 4 and 5 of the for-stack entry. The program counter is set to the statement following the associated <NEXT statement> and control transfers to the <NEXT statement> execution code to compare the initial and limit values (see flow chart).

<NEXT statement> execution decides whether to iterate a loop or close it. The for-stack is searched for an entry with the same for-variable. If none is found the statement is ignored and control passes to the following statement. If the entry is found, any entries above it (more recent entries) are eliminated; i.e., they are assumed to belong to nested loops wich were not closed

by exceeding their limit value but exited otherwise. The value of the for-variable is then incremented by the step size and the new value tested by subtracting the limit value and using the sign of the step size to determine whether a non-negative or non-positive result indicates 'success'. If the result is 'success', the program counter is loaded from word 6 of the for-stack entry (the reference to the statement following the <FOR STATEMENT>). If the result if not 'success', the for-stack entry is eliminated. At this point the program counter already points to the statement following the <NEXT statement> so exit is simply to the main execution loop.

<RETURN statement> execution merely loads the program counter from the top entry of the return stack. An error condition is generated if the return stack is empty.

<INPUT statement> execution assigns values to the input list for both INPUT and MAT INPUT. INITF=0 and MCNT is meaningless when executing an <INPUT statement>; for MAT INPUT, INITF = -1 and MCNT holds the number (in 2's complement) of elements of the current array as yet unassigned values. IFCNT holds the ordinal number of the current item in the current record. (Note that IFCNT is not cumulative over the entire execution of a statement requesting input unless the request is met entirely by one line from the teletype).

The <INPUT statement> executor is also used by the <READ statement> executor when READing ASCII files. The routines INCAL (used to get an input record) and GETCR (used to obtain a character from an input record buffer) operate on user terminal input or ASCII file input (either disc files or devices) depending upon the value of the variable INFIL. If INFIL=0, the user terminal is the input device. INCAL will initialize the terminal input buffer control block and set INFIL to point to this file control block (actually 9 words before the control block since we always point to the first word of file control blocks but in this case we have a pseudo control block of 2 words). Bit 15 of INFIL is set to indicate a user terminal instead of an ASCII file. INCAL then reads an input record from the appropriate device and GETCR retrieves characters from the buffer. Note that in the case of an <INPUT statement> we are always dealing with the user terminal. INFIL is cleared before proceeding to the next statement.

The general approach in execution is to determine the address and type of a variable in the input list and then attempt to satisfy it from the input record. When an error occurs in the above process, it is explained along with any necessary corrective action so that errors in the input record will not terminate program execution. For simple input if the next variable in the list is of numeric type its value table address is placed into SBPTR; for array input the base address of the is put into SBPTR. After filling a simple variable the next variable from the list is taken and a new address generated; after filling an array element SBPTR has been advanced to the next element by the numeric input routine so no new address need be calculated. When MCNT rolls over to zero (an array has been filled) control exists to the MAT INPUT code, which may return with another array's base address in SBPTR and MCNT reset appropriately. If the input record is empty but the variable list is not yet exhausted a request for additional input is made (signified by '??' rather than the initial '?'). SERR is needed as a flag to indicate if under/overflow ocurred while converting the latest numeric input, since the error message will have destroyed any additional information in the input record. When looking for a number, the input record is scanned for the first sign (+ or -), digit, or decimal point, which begins the number. Any other characters will be ignored except the ", which will generate a recoverable error.

String input requires fairly complicated analysis of the data transfer. If the string variable does not specify the transfer length (does not have a double subscript), then the next string in the input record is transferred in its entirety and the logical length of the variable set appropriately. If the next string variable specifies the transfer length then exactly that much of the next string in the input record will be transferred, either truncated or extended by blanks as necessary to achieve the specified length. The 'next string' in the input record begins with the next non-blank character or, if it is a ", the following character, blanks included. The string ends with first " (which is not part of the string) encountered or with the carriage return (also not part of the string) if no " appears.
Every data item in the input record must be followed by a comma or carriage return and a comma must be followed by another data item. Failure to observe the above will generate recoverable errors. INTMP holds the type of data being sought, INTMP = 0 for a number or INTMP <>0 for a string, and is used by the error recovery code to prepare for the entry.

<ENTER statement> execution assigns a value to a string variable or a simple variable. If a '#' follows the ENTER, the user's port number (0-31) is assigned to the first variable. The <ENTER statement> is timed and the length of time it took to respond (in seconds) is assigned to another variable. The input analysis proceeds much like an input statement with one variable, with the notable excetion that no error messages are printed. Instead, the response time variable is negated if an error occurs. If the user does not respond within the allocated time, the response time variable is set to -256. Parity error returns -257, lost character returns -258. This is non-ambiguous since response times are between 1 and 255 seconds inclusive. Also, for string input leading blanks are non stripped off and quote marks are allowed as characters.

The <ENTER statement> executor is also used to perform the assignment for the LINPUT statement. The common code occurs after the input record has been read. STINF is called to set up the pseudo file control block for the user terminal (in the case of LINPUT from another device, the file control block for that device is used). In either case, INFIL points to the file control block (or pseudo block) with bit 15 = 1 if we're using the user terminal. The GETCR routine is used to obtain characters from the input buffer. When the evaluation and assignment are complete, control is returned to the LINPUT executor if the statement is LINPUT (LFLAG = -1 for LINPUT, =0 for ENTER).

<READ statement> execution assigns values to variables in the <read variable list>. If a file reference is present it is evaluated by the routine VLFIL, which sets INFIL to point to the file control block if the file is of type ASCII (INFIL =0 otherwise). If the file is ASCII or if the file is file zero (corresponding to the user terminal), the <INPUT statement> executor is used to assign the values. Otherwise the routine FDATA is used to obtain values from either a BASIC formatted file or from <DATA statement>s. A mismatch in type between a variable in the <read variable list> and the next data item, or a string too long to fit into its designated destination, will generate an error and terminate program execution (see Notes on error routines for exception).

<LINPUT statement> execution assigns a value to a string variable. If a file reference is present it is evaluated by the routine VLFIL, which sets INFIL to point to the file control block if the file is of type ASCII (INFIL =0 otherwise). If there is no file re-

ference or if the file reference is for file 0, an input record is requested from the user terminal. If a file reference is present, a check is made to insure that the file is ASCII and not BASIC formatted (an error resits if the file is BASIC). An input record is read from the appropriate device. In any case, the <ENTER statement> executor is used to evaluate the input record and assign the values. The <LINPUT statement> sets LFLAG=-1 to indicate to ENTER that we are really doing LINPUT. The <ENTER statement> executor will return control to the <LINPUT statement> executor to perform necessary cleanup (including the clearing of INFIL, and LFLAG).

<PRINT statement> execution consists of identifying items in the print list and sending them to the user or other output device. A file reference is evaluated by VLFIL (absence of a file reference or a reference to file 0 indicates the user terminal). If the PRINT is to a BASIC formatted file then 0<=FILE#=15 where FILE#=file reference-1. If output is to the user terminal OUTTB has bit 15 set and OUTMP=0. If output is to an ASCII device then OUTMP points to the user terminal control block (and has bit 15 set) and OUTTB points to the file control block for the ASCII device.

If the PRINT is to a BASIC file, the end-of-line mode flag is turned off (EOL=-1). A user terminal or ASCII device print turns on the end-of-line mode flag (EOL=0). A comma or semicolon in the <print list> turns off EOL but a comma also generates enough blanks to advance to the next field of 15 characters if not a BASIC file. A literal string is written as a string of characters, less quotes, and turns on EOL if not a BASIC file. An END writes an end-of-file mark to an ASCII device or BASIC file.

Formula in the <Print list> are evaluated and the results examined. Formulae which are string variables evaluate to their contents, which are then treated as literal strngs. If not a string variable but within a print to a BASIC file, the floating point value of the formula evaluation is written in the file in two-word floating point format. If a print to an ASCII device or user terminal, the floating point value is converted to an ASCII character string of the decimal equivalent. LIN, SPA, TAB, and CTL are ignored for prints to BASIC files. For ASCII or terminal prints, evaluation of the function produces the desired action, so the return value is thrown away, along with the following delimiter if one exists. For terminal or ASCII device prints, all formulae turn on EOL. If EOL is on after processing the

last prnt item, a carriage return line feed is printed on a terminal, or the output record ended for an ASCII device. In addition, ASCII files must reset OUTTB and OUTMP and ASCII disc files set the last operation was a print bit in the file control block, unless a CTL(24) was the last item printed.

Before writing a quantity BASIC insures that sufficient space is available to accommodate it. CHRCT keeps track of the current print position on the teletype line (0-71). If the character string sent to the teletype would require non-blank characters to be printed past position 71, a carriage return-line feed is output first and CHRCT set to 0. Buffer contents are always sent to the IOP at the end of the print statement when doing terminal output. If an item sent to a file requires more words than remain in the current record, BASIC automatically advances to the next record if in serial mode or exits to the end-of-record code if in record mode.

<PRINT USING statement> execution is covered in the section on the formatter.

<RESTORE statement> execution resets the pointers to the DATA block, beginning at the statement specified, or at the first statement in the program if none is specified, the pointers are set to the first <DATA statement> found, or to the out-of-data condition if none is found.

<MAT statement> execution involves many disparate tasks. The forms of the <MAT statement> may be classified as array I/O, array assignment, array initialization, and the array functions TRN and INV. For conciseness in coding, all forms other than array I/O use some common program segments.

Array I/O prepares each array in the list in the same fashion. SBPTR is set to the dynamic dimensions of the array (base address -2) and the operator following the array identifier is picked up for examination. At this point MAT PRINT USING calls the formatter just as PRINT USING does. The EVEXP routine in the formatter takes care of picking up the elements of the array one by one, in rows. MAT PRINT follows a separate path than MAT READ and MAT INPUT. The following operator is noted as spacing the elements (comma or end-of-statement) or packing them (semicolon). VCHK examines the array and generates an error if any of its elements have value 'undefined'. The dynamic row and column lengths are saved in 2's complement. If the MAT PRINT references a BASIC file, the array elements are written one by one in rows, each element in its two-word binary form. If the MAT

PRINT references the user terminal or an ASCII file rows are double spaced and the elements within a row are spaced or packed as noted above, each element in its ASCII decimal form. Both MAT READ and MAT INPUT redimension the array, in 2's complement. MAT READ calls FDATA for element values while MAT INPUT transfers to the <INPUT statement> execution to obtain element values. MAT READ #0 is treated as a MAT INPUT statement. MTU acts as a flag for MAT INPUT, differentiating the first call for input from subsequent calls and saving the input record. After completing I/O on an array, a common section of code prepares the next array in the list or, if no more remain, terminates the statement execution. MAT INPUT returns to the input code to clean up there, MAT PRINT and MAT READ return directly to the main execution loop.

Array assignment consists of preparing the destination and source arrays and executing a loop which assigns the destination array elements one by one. The general procedure is to assign a jump to the element computation code to MOP, an exit address to MEXIT to use after completing the destination array, and a count of the elements to MCNT, in 2's complement. The code to compute an element returns to MLOP1, MLOP2, or MLOP3 depending on the number of arrays involved which require updating of the element address. Each operation checks the dimensions of the arrays involved to insure that the operation is well-defined; and all elements of the source matrices are checked to make sure none have value 'undefined'. Matrix multiplication does not use the element computation loop, instead it uses row and column counters to tell when it is done and computes destination array elements by inner products of the rows and columns of its source matrices.

Array initialization also uses the element computation loop. The initialization program first redimensions the destination array (if a matrix subscript is given) and then chooses the appropriate constant for the element values. IDN acts like ZER except that it insists that the destination array be 'square' and sets a special counter to choose 1.0 for the value of main diagonal elements.

TRN and INV are handled apart from the other matrix functions. For both of these, the elements of the source matrix are checked against the 'undefined value'. The source and destination matrices are then checked for transpositional compatibility. If TRN, then proceed to transfer the columns of the source matrix to the rows of the destination matrix.

INV uses the Gauss-Jordan algorithm with row pivoting. This procedure converts a copy of the source matrix into the identity matrix and converts an identity matrix into the inverse by applying the same set of operations to both. Since the source matrix is destroyed in the process, it is first copied into free user space and the copy treated thereafter as the source. A side effect of the copying produces the element of largest absolute value, which is used to compute a lower bound on the allowable magnitude of pivot elements. INV then calls IDN to set the destination matrix to an identity matrix, having the side effect of checking that the matrix is square.

Diagonalization of the source matrix and production of the inverse now proceeds on a row-by-row basis. The next unreduced column of the source is searched for the pivot element (the largest in magnitude). If necessary, rows are swapped to put the pivot element on the main diagonal (the corresponding rows of the destination matrix must also be swapped). If the pivot matrix is too nearly singular to invert execution is terminated. Otherwise, the pivot rows of both matrices are divided through by the pivot element.

<END statement> and <STOP statement> execution terminates the program run. User error trapping is terminated, files with "dirty" buffers in core are written, non-shareable devices released, last change date for each file is updated and the user's section of the FUSS table is cleared. The word DONE is sent to the user terminal and user not running (UNR) is sent to the IOP. PBFLAG, ABTRY, and CBFLAG bits in the user teletype table are cleared and control returned to the termination logic in the scheduler.

<CHAIN statement> execution begins by evaluating the address of the return variable (if present), the address of the program designator, and the optional expression. User error trapping is disabled (UTRAP=0) and a jump is made to the CHAIN library overlay.

The CHAIN overlay checks the validity of the program designator, performs a directory search on the name, and verifies user access to the program (PFA if needed, program not private, etc.). The program length is checked to insure it will fit, the last reference date is updated, and variables set for return to the in-core statement executor (protected, locked, and private bits set if needed). Return is made through the scheduler which first checks for abort or disconnect and remaining time on time-out clock.

The in-core executor then takes an error exit if an error occurred in the CHAIN overlay and no return variable was specified. If a return variable was specified, its value is set and program execution continues with the next statement.

If no error occurred in the overlay, any "dirty" file buffers are written, the last change dates updated, and any devices released. If an *OUT=* file is in use, a dirty buffer (if present) is written. If *OUT=* file is an ASCII disc file, the address of the next disc block is saved. The new program is then read into core. The CHNFG in the user teletype table is set, non-shareable device assignments cleared, and excess common area (if any) initialized to undefined. The executor then exits to the compile logic.

The compile logic for CHAIN differs only slightly from normal compilation. The program name is not printed, and the disc address of the ASCII disc *OUT=* file (saved above) is used instead of the disc base address.

<ASSIGN statement> execution changes the file referred to by a specified file number. The first operand is either a * or a file designator and the second is a file number, which is evaluated. The return variable address is evaluated and saved in ATMP+1 (a dummy address is used if no variable is specified). If the current file buffer is "dirty", it is written; and if it is a non-shareable device, it is released. The count of locked files is decremented if the file is locked. The file position is marked not in use by clearing the first word of the file control block and the protect mask is cleared. The protect mask (if present) is evaluated and the restriction (if present) is saved in TEMP7. Control is then transferred to the ASSIGN library overlay.

The ASSIGN overlay updates the directory entry for the current file, verifies user access to the new file, and sets up the file control block for the new file. If the old file is locked, it is unlocked and a check made to insure sufficient core space for the buffer of the new file. Errors are returned to the in-core executor through the variable ORUNO.

Upon return to the in-core ASSIGN executor, if no error occurred, the protect mask is evaluated and stored in the file control block.

<SYSTEM statement> execution consists of calling the SYSTEM library overlay. The overlay first evaluates the return variable address and notes whether its type is

numeric or string. The command string is evaluated and the command checked for validity. The type of the return variable is also checked against the type of the command response. The command parameter (that portion of the string following the hyphen after the command name) is moved into FILBF. The proper command overlay is then read into core replacing the SYSTEM library overlay. However, SYSFL has been set non zero to indicate that the SYSTEM statement called the command overlay.

When the command overlay is finished, it checks the value of SYSFL to see if it should call in the SYSTEM statement overlay for cleanup. The cleanup routine sets the return variable value (string or numeric) and control is returned to the in core (SYSTEM statement) executor which reports a string response overflow (if it occurred) or else proceeds to the next statement.

<CONVERT statement> execution involves conversion between string and numeric data. The first operand is evaluated to determine the direction of the conversion. If the first operand is numeric it is converted into a string using the same rules as in LIST. If the first operand is a string it is converted to a number. If an error occurs during the conversion, a check is made for an exit label. If one exists, execution continues there, otherwise a fatal error occurs.

<LOCK statement> and <UNLOCK statement> execution consists of setting PS0=0 for LOCK or -1 for UNLOCK and then calling the LOCK/UNLOCK library overlay after evaluating the file reference and saving it in ORDNO. The overlay then evaluates the address of the return variable - if none is present a dummy address is used. The variable ATMP is set to 0 if a return variable is present. A check is made to insure that the file is not ASCII. At this point the LOCK and UNLOCK statement executors diverge.

The UNLOCK executor insures that the file is locked, (by checking the lock bit in the file control block) then scans the lock queue to re-schedule the first user on the queue who is waiting to LOCK this file. The file locked bit in the FUSS table entry for this file is cleared, LCKFL is decremented (count of locked files for this user), and the locked bit in the file control block is cleared. The active buffer pointer for the file is bumped to the end of buffer and the buffer contents are written to disc if the buffer is dirty.

The LOCK executor checks that the file is not already locked by this user (thru the FCB lock bit). If it is, a fatal error exit is taken unless a return variable is specified, in which case a one is returned. The FUSS table is scanned to see if any other user has LOCKed the file. If the file is already locked and a return variable was specified the value one is returned (file already locked/unlocked). If no return variable was specified, the user is put on a LOCK queue and the disc address of the file is placed in the teletype table TEMP words. If the file is not currently locked, a check is made to see if this user currently has any other files locked (LCKFL=0). If so, a return variable must be present or a fatal error results. The FUSS entry for this file is marked locked, as is the FCB entry. The count of locked files (LCKFL) is incremented. If the user currently has a record in the buffer, it is written to disc (if dirty). The record is always read anew from the disc and the active pointer is reset to the beginning of the record. Control is then returned to the in-core statement executor through the scheduler.

The in-core executor reports fatal errors. If also takes care of releasing terminal output buffers for printing and placing the user in output wait if he tried to lock an already locked file.

The scheme of writting dirty reords on LOCK or UNLCOK and always re-reading the current record on a LOCK, insures the user that he always has the current disc record. It is also important to note that a user placed on the lock queue must not be swapped out by the scheduler before his status is changed to output wait - otherwise the user who has the file locked may unlock the file and reschedule the use by resetting the PACT bit. However, if the status isn't output wait the user will not be rescheduled (the PACT bit will be turned off and ignored).

<CREATE statement> execution starts by setting BTMP=-1 to flag the statement as a CREATE and not a PURGE which uses the same in-core executor but which uses a different library overlay. The return variable address is evaluated and saved, the file name is evaluated, and the file length is evaluated. The record size is evaluated if present. The CREATE library overlay is called after first setting SYSFL to indicate a programmatic call.

The CREATE library overlay is also used to process the CREATE command. The overlay creates the file with the specified parameters if room exists on the system and in the user's account. Each record of the file is initialized to an end-of-file. Success or failure is indicated via the return variable. See listing for additional details.

<PURGE statement> execution sets BTMP=0 to flag the statement as a PURGE. The return variable address is evaluated and saved and the file name is evaluated. SYSFL is set to indicate a programmatic call and the statement execution continues in the PURGE library overlay.

The user's library is searched for the file name and a check made to insure that the file is a data file and not a program file. The FUSS table is checked to determine whether the file is in use and, if not, the file is deleted. Success or faiure is indicated by setting the return variable. See listing for additional details.

<ADVANCE statement> execution first evaluates the file reference and the skip count. A check is made to insure that the file is not ASCII. Items in the file are skipped until either the skip count is satisfied or an end of file is encountered. The return variable address is evaluated and the number of items not skipped (if EOF was encountered) is placed in the variable.

<UPDATE statement> execution evaluates the file reference and checks that the file is not ASCII. The file control block is checked for write capability. The expression or string is evaluated. The type of the next item in the file is checked. If an EOF, an error exit is taken. If the file contains a number, the expression in the UPDATE statement must have evaluated to a number. If so, the number is placed in the file. If the file contains a string the expression must have evaluated to a string. The string is transferred, and is either truncated or blank-padded to the same length as the original file string. If an update was successfully performed, the dirty buffer bit is set in the file control block.

<IMAGE statement>, <COM statement>, <DIM statement>, <DEF statement>, <DATA statement> and <FILES statement> do not require statement executors as they only contain information which is required for compilation or during the execution of other statements. The <REM statement>, of course, requires no executor by definition.

The Formatter

The formatter is responsible for PRINT USING and MAT PRINT USING statements. It consists of three sections: the PRINT and MAT PRINT syntax routines, the EPRUS subroutine called from PRINT and MAT PRINT executors, and the formatter proper. IMAGE statement syntax and execution will not be explicitly discussed beyond mentioning that syntax merely stores the IMAGE string without any validity checks and that IMAGE statement execution is a no-op.

PRINT USING and MAT PRINT USING Syntax

After recording an optional file reference the PRINT and MAT PRINT syntax routines check for a USING keyword (subroutine USTCK). USTCK allows a string variable (USING A$), a string constant (USING "#,2D"), or a statement number (USING 1234). If end of statement, accept it, otherwise demand a semicolon and return to (P+2). For PRINT USING, we accept a print function (CTL, LIN, SPA, or TAB), a string variable, or an expression. Multiple items must be separated by commas. MAT PRINT USING only allows print functions or array identifiers, separated by commas.

PRINT USING and MAT PRINT USING Execution

After validating a possible file reference (VLFIL), a call to execute PRINT USING (EPRUS) is made from both execute print and execute mat print. EPRUS determines whether or not we have a PRINT USING or a MAT PRINT USING, sets FFLG to indicate the respective type of print statement, and then sets up a pointer to the format string. Finally, the length of the format string is calculated and the formatter is jumped to.

Formatter Execution

The formatter works in two passes. The first pass scans for a legal specification. After this, pass two is entered to either output a number or a string. Blanks are output on the fly in pass 2. After outputting a number or string according to specification, pass one is returned to and processing continued. The comments in the listing are unusually clear and should guide well.

NOTES ON THE ERROR ROUTINES

Errors are handled by routine SERR, reached by a jump through the base page error jump table SERRS. A JSB SERRS + I, I signifies detection of error I. The subdivisions SERRS, RERRS, FERRS and WERRS of the jump table correspond to sections of the error message table containing syntax, run-time, format and warning-only messages, respectively.

Syntax errors, except when detected in tape mode, are prompted by the message ERROR. Any response except a carriage return will cause the appropriate error message block to be read into the library overlay region. The message will be transferred to the ERSEC table and printed on the user terminal.

Syntax error detected while in tape mode are handled by accepting error psuedo-statements in place of the erroneous statements. These psuedo-statements will be replaced by any subsequently received statements with the same line number. Consequently, provision is made in FNDPS, which returns the location of a statement when given its sequence number, to decrement the error counter (ERRCT) whenever the statement found is an error psuedo-statement. (Note: An error psuedo-statement will only be found by FNDPS when another statement with the same sequence number is ready to replace it). Over/underflows detected during number conversions in syntax mode cause warning messages to be issued only after accepting the statement, if it is otherwise correct. Since no printing can be done while in tape mode, the routine CHOUF suppresses setting of the flag and these potential errors are not reported when in tape mode.

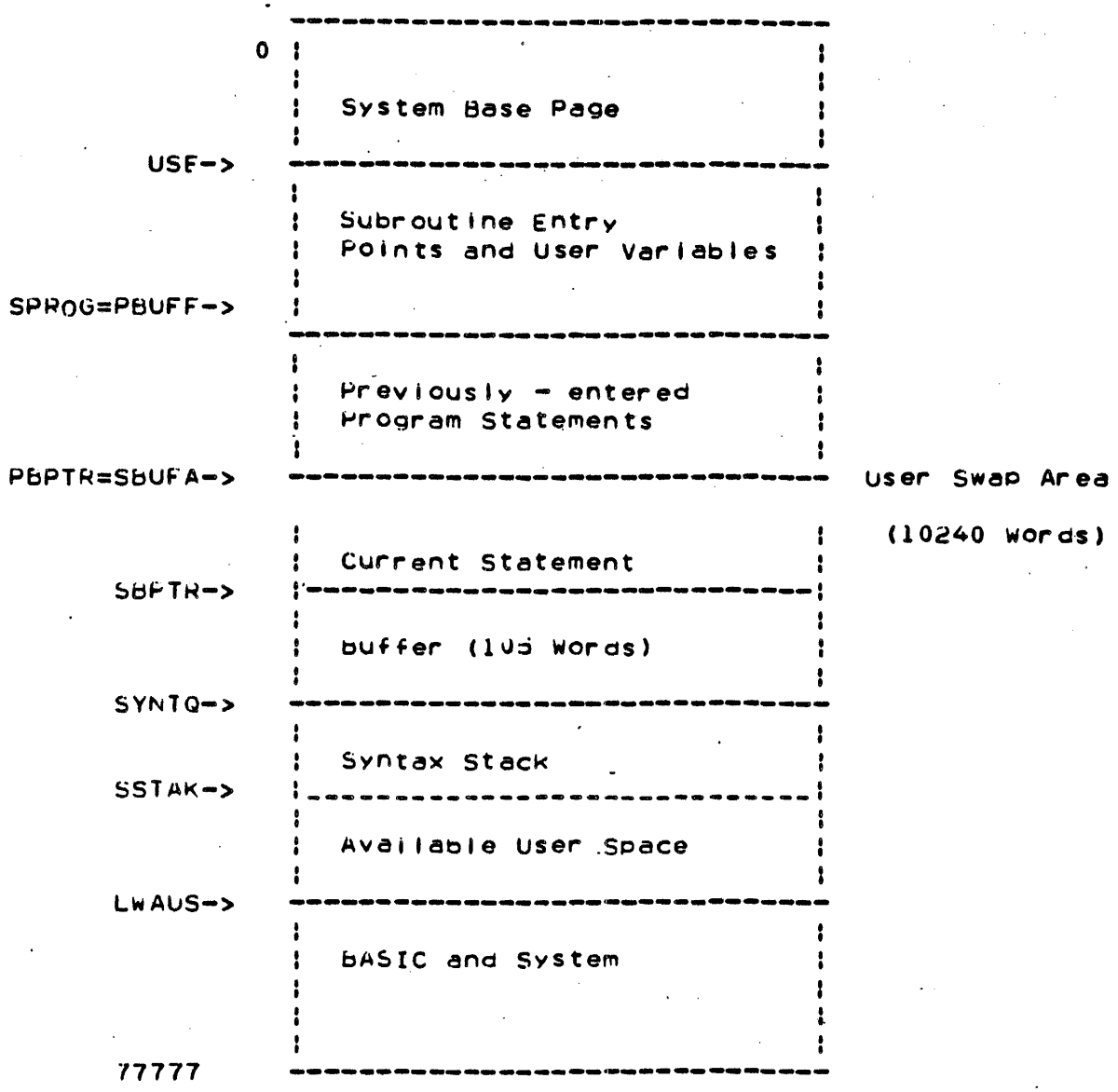
If the error is not a syntax error, a check is made to determine if the user has enabled program error trapping via the 'IF ERROR' statement. If he has, and the error is not either PROGRAM BAD or a compile error (which are not trapped), then the error number is saved in LERR and the line number in LSTAT. The user can access the values by use of the SYS function. If the error was only a warning, statement execution continues. If it was a run-time or format error. the current statement is terminated and program execution resumed at the line number specified in the 'IF ERROR' statement.

If the user is not trapping errors and the error is either a run-time or format error, the ASCII file print flag is turned off (if set), UNR is sent to the IOP, and PBFLAG, CBFLG, and ABTRY bits in the teletype table are cleared. The error message is printed (as in the case of syntax). If a buffer has been allocated for LIST, indicated by SPACQ=1 in the teletype table, it is de-allocated. The CHNFG and SPACQ bits are cleared and the OUT= file is killed (if one exists). All open ASCII files are killed. Any dirty buffers for BASIC formatted files are written to disc. The last changed dates are updated, the user's FUSS

cleared and a jump is made into the scheduler's termination logic. The user status will be set to idle and he will be removed from the queue.

BASIC Core Maps

SYNTAX (Phase 1)



Pointers

USE Fixed, first word of user swap area.

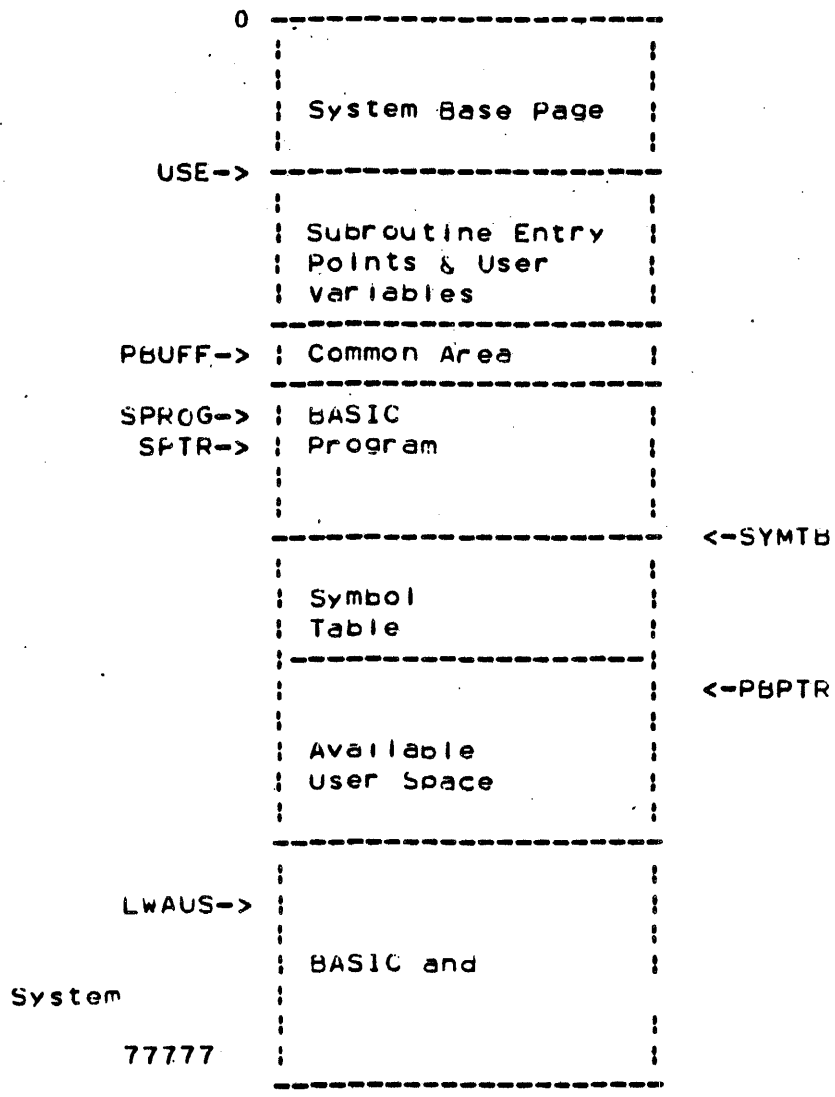
PBUFF Fixed, first word of program space.

SPROG Fixed, first word of program.

SBUFA	Variable, first word of statement being syntaxed.
PBPTR	Variable, first word of program space not used by previously accepted program statements.
SBPTR	Variable, first word not used by statement being syntaxed.
SYNTW	Variable, first word of syntax stack.
SSTAK	Variable, last word of syntax stack.
LWAUS	Fixed, first word not in user swap area.

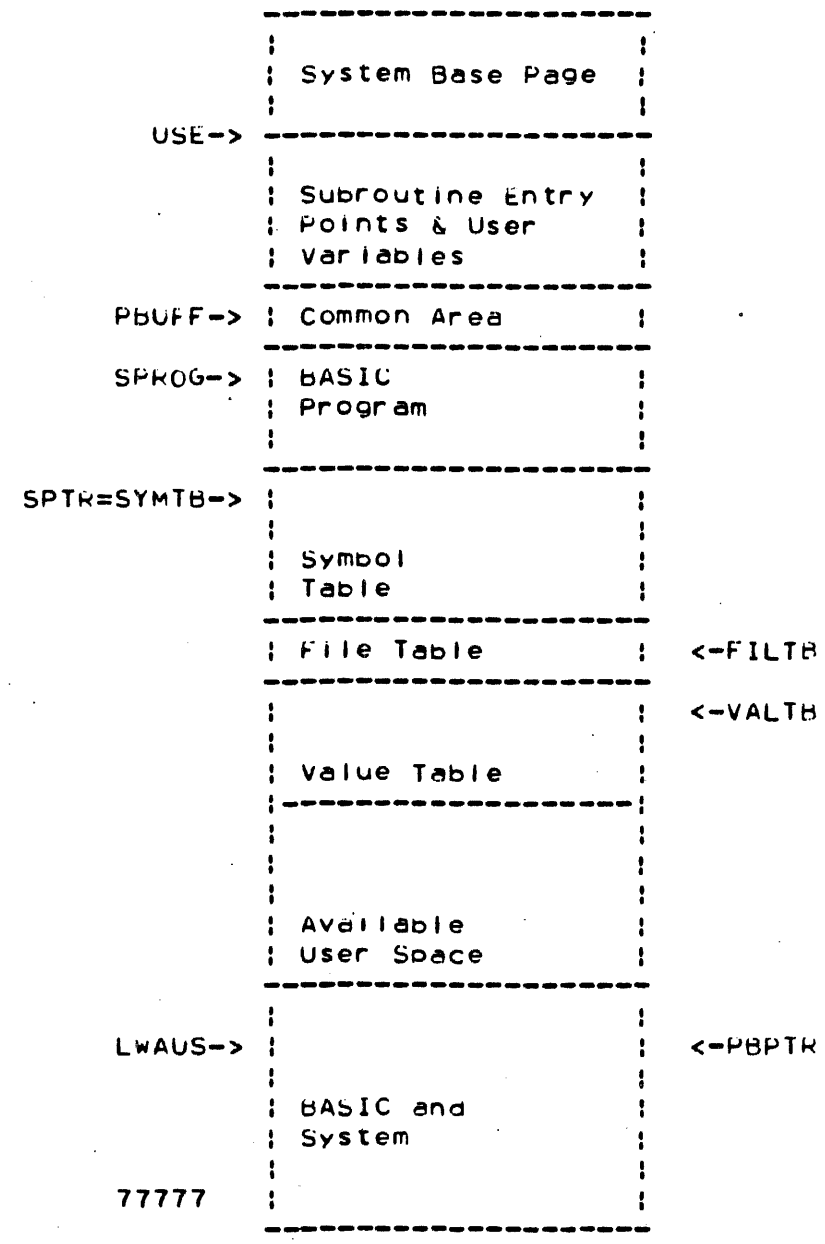
COMPILATION_(Base_III

Compilation



COMPILATION_(Phase_III) Continued

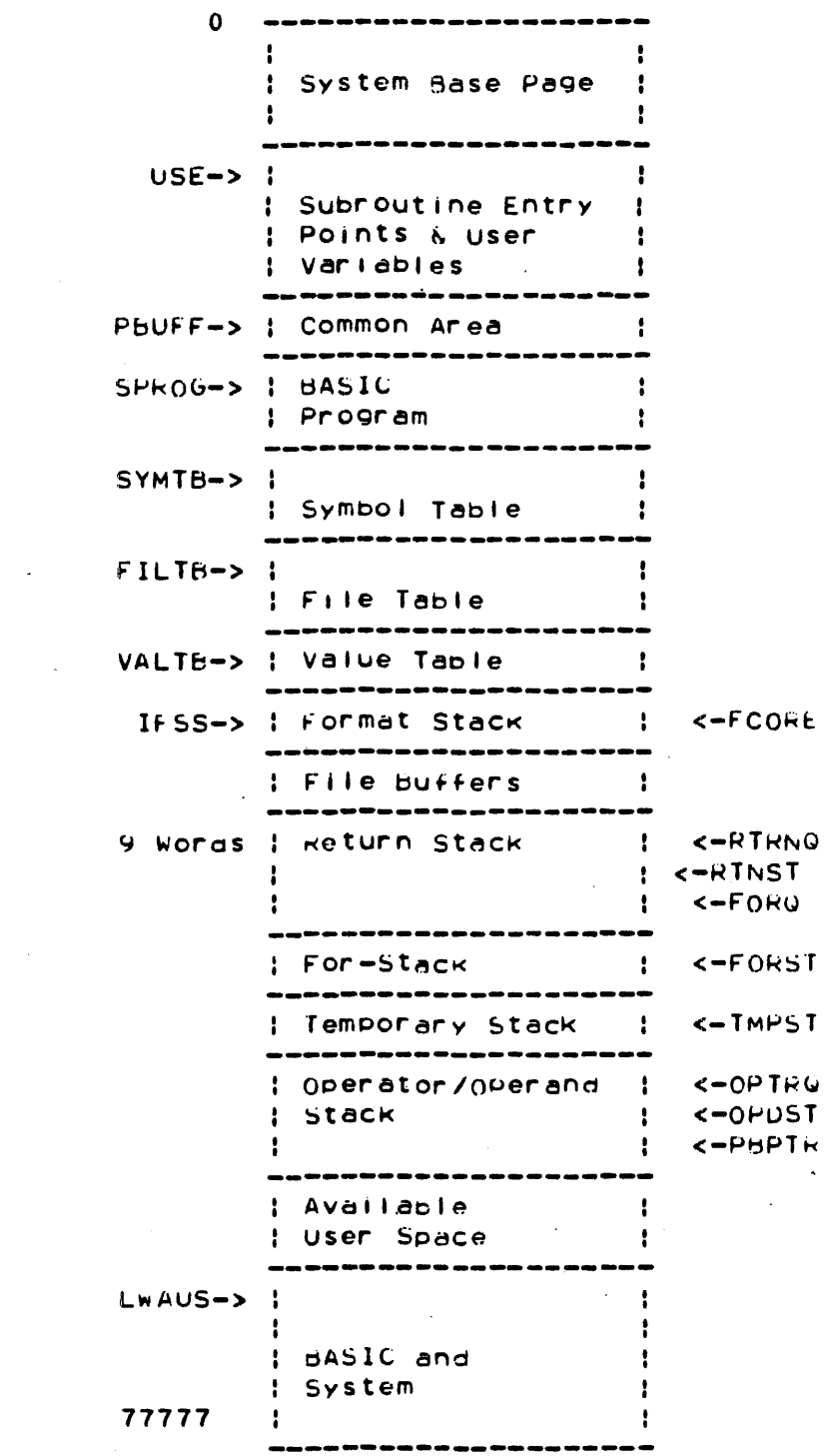
Value_Storage_Allocation



SPROG - Variable, first word of program.
SYMTB - Variable, first word of symbol table.
SPTR - Variable, word of program being processed.
FILTB - Variable, first word of file table.

VALTB - Variable, first word of symbol value table
(FILTB = VALTB if no <FILES statement> is in program)
PBPTR - Variabble, first word available of user space.
SYMTB and SPTR are not changed after compilation.
FILTB and VALTB are not changed after allocating value storage.

EXECUTION (Phase III)



EXECUTION_Phase_III_Continued

IFSS - Variable, first word of format stack
FCORE - Variable, first word not used by Phase II.
RTRNQ - Variable, bottom of return stack
return stack)
RTNST - Variable, top of return stack
FORQ - Variable, bottom of for-stack (sixth word preceding
for-stack)
FORST - Variable, top of for-stack (points to latest 6-word
entry)
TMPST - Variable, top of temporary stack (points to latest
2-word entry)
OPTRQ - Variable, bottom of operator stack.
OPDST - Variable, top of operand stack.
PBPTR - Variable, top of operator stack.

FCORE, RTRNQ, and FORQ are not changed after initiating execution.

Entries on the operator and operand stack are one word each and interleave (i.e., alternate words belong to one stack). All stacks beyond the return stack grow and shrink as needed so long as user space is available.

BASIC Internal Representation

BASIC statements are represented internally by the sequence number followed by the length in words (including the sequence number and length words) followed by the statement body. The statement body is composed almost entirely of operator-operand pairs which occupy from one to three words each. Null operands and operators are used when necessary to maintain the operator-operand correspondence. The operator resides in bits 14-9 of a word; the operand uses bit 15, bits 8-0, and sometimes whole additional words immediately following.

!Variable!_Operands

0	Operator	0	Null Operand

0	Operator	Name	0	String Variable

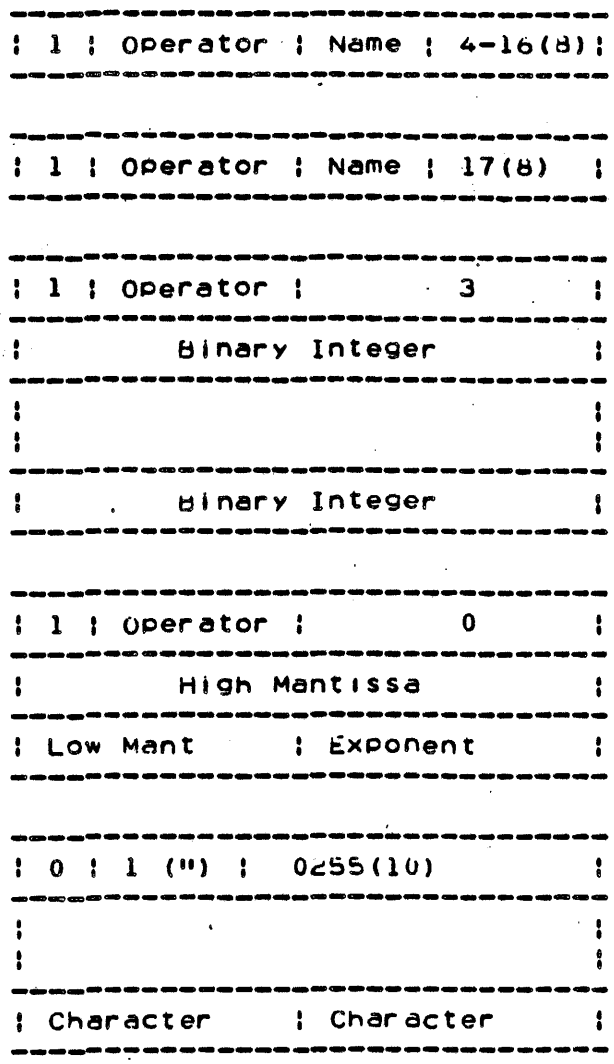
0	Operator	Name	1-3	Array Variable

0	Operator	Name	4-16(8)	Simple Variable

0	Operator	Name	17(8)	Function Variable

Bits 8-0 are generally divided into two fields as follows: a name field (bits 6-4) and a type field (bits 3-0). The name field holds a value between 1 and 32(8) corresponding to A-Z (for functions, corresponding to FNA through FNZ). A type of 0 identifies a string variable (e.g. 3,0 represents C$). Types 1 2 identify array variables of dimenstionality one and two respectively (e.g. 4,2 represents D[*,*] while type 3 identifies an array ariable whose dimensionality cannot be determined by its immediate context. Type 4 identifies a simple variable with no digit (e.g. 1,4 represents A) while types 5 - 16(8) identify simple variables whose names include the digit 0-9(10) respectively (e.g. 6,7 represents F2). Type 17(8) identifies a programmer-defined function (e.g. 32(8), 17(8) represents FNZ). Name field values of 33(8) thru 36(8) correspond to extended string representations (e.g. A0$ thru Z1$).

!Constant!_Operands



A parameter (which can only appear inside a <DEF statement>) differs from a simple variable only in that bit 15 is set. The name of a predefined function may range, in the standard system, from 0 to 31(8) or 33(8) to 37(8). (CTL to SYS or ZER to TRN). A flagged (bit 15 set) operand of 3 identifies either a formal dimension in a <DIM statement> or <COM statement> (value in following word) or a branch address list (one or more statement sequence numbers in the following words). A flagged operand of 0 indicates that the following two words hold a floating-point constant (all numerical constants within a program are so represented). The operator with internal code 1 is ", which signals the start of a string constant. The operand portion of the word has a value from 0 to 255(10), indicating the number of

characters in the constant. The string follows, two characters per word, and the closing " is not explicitly represented internally.

The table below gives the internal representation of the BASIC operators. Those operators which manipulate the formula evaluation stack during execution hae associated priorities. BASIC statement types are represented in the following table. They can be differentiated from BASIC operators because statement types must appear only as the first operator in a statement. All numbers are in octal notation.

BASIC Operators

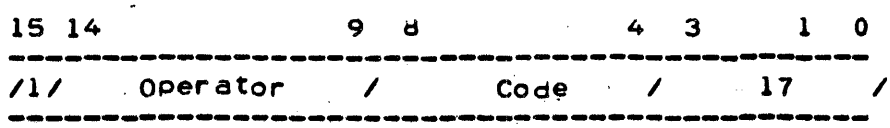
OP	PRIORITY	ASCII	OP	PRIORITY	ASCII
0	0	(end-of-formula)	31	4	(unused)
1	0	"	32	4	and
2		,	33	3	OR
3		;	34	6	MIN
4		# (FILE)	35	6	MAX
5		(unused)	36	5	<>
6		(unused)	37	5	>=
7		(unused)	40	5	<=
10	1	)	41	11	NOT
11	1	]	42	12	** (power)
12	13(1)	[	43		USING
13	11	(	44		PR
14	11	+ (UNARY)	45		WR
15	11	- (UNARY)	46		NR
16	2	, (subscript)	47		ERROR
17	2	= (assignment)	50-57		(unused)
20	7	+	60		END
21	7	-	61-62		(UNUSED)
22	10	*	63		INPUT
23	10	/	64		READ
24	12	^	65		PRINT
25	5	>	66-73		(UNUSED)
26	5	<	74		OF
27	5	#	75		THEN
30	5	= (equal)	76		TO
			77		STEP

BASIC Statement Types

OP	ASCII	OP	ASCII
0-31	(unused)	55	NEXT
32	SYSTEM	56	GOSUB
33	CONVERT	57	RETURN
34	LOCK	60	END
45	UNLOCK	61	STOP
36	CREATE	62	DATA
37	PURGE	63	INPUT
40	ADVANCE	64	READ
41	UPDATE	65	PRINT
42	ASSIGN	66	RESTORE
43	LINPUT	67	MAT
44	IMAGE	70	FILES
45	COM	71	CHAIN
46	LET	72	ENTER
47	DIM	73	'IMPLIED LET'
50	DEF	74	(unused)
51	REM	75	(unused)
52	GOTO	76	(unused)
53	IF	77	(unused)
54	FOR		

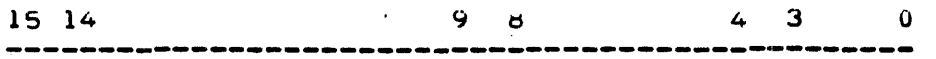
Pre-Defined Function Table

Operator - Operand Pair Format



CODE	ASCII	CODE	ASCII
0	CTL	20	SIN
1	TAB	21	COS
2	LIN	22	BRK
3	SPA	23	ITM
4	TAN	24	REC
5	ATN	25	NUM
6	EXP	26	POS
7	LOG	27	CHR$
10	ABS	30	USR$
11	SQR	31	SYS
12	INT	32	(unused)
13	RND	33	ZER
14	SGN	34	CON
15	LEN	35	IDN
16	TYP	36	INV
17	TIM	37	TRN

Extended String Representation



STRING VARIABLE	NAME	SUB-OP
A$-Z$	1-32	0
A0$-P0$	33	0-17
Q0$-Z0$	34	0-11
A1$-P1$	35	0-17
Q1$-Z1$	36	0-11

Some examples of BASIC statements in their internal form are given below. Note that actual function parameter formulas, <DEF statements> formulas, and subscript formulas appearing in <MAT statements> require end-of-formula operators to signal their end whereas most formulas end either with the first operator which does not manipulate the formula evaluation stack or with the end of the statement. Note also that constants are considered signed only within a <DATA statement>. ASCII numbers are decimal, internal numbers are octal in the presentation below.

```
10      LET W1 = Y = (B = C) ^ 3*A[I,J+K]

    12                          sequence number
    20                           length
0   46   27   6                  LET W1
0   17   31   4                  =    Y
0   17        0                  =
0   13    2   4                  (    B
0   30    3   4                  =    C
0   10        0                  )
1   24        0                  ^
060000                           3.0
000004
0   22    1   2                  * A
0   12   11   4                  [ I
0   16   12   4                  , J
0   20   13   4                  + K
0   11        0                  ]
```

```
20 DIM A[5], C[6,,12]

   24
   14
0 47    1   1
1 12        3
            5
0 11        0
0  2    3   2
1 12        3
            6
1 16        3
           14
0 11        0
```

```
30 DEF  FNC (X) = X + A0

        36
        10
0   50   3   17
1   13  30    4
0   10       0
1   17  30    4
    20   1    5
0    0       0
```

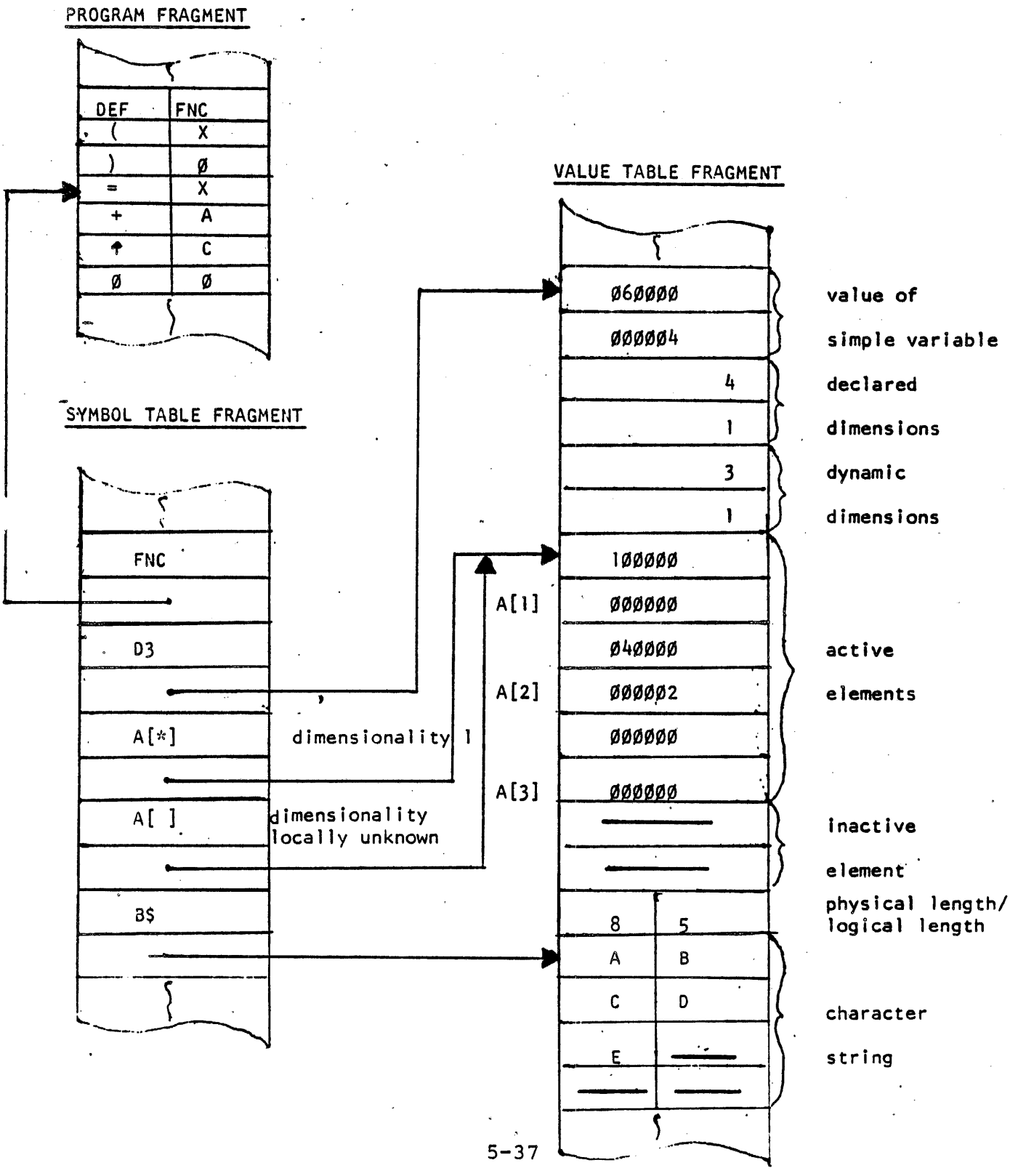
```
40  REM  ARK

        50
         5
0 51    40
```

```
50 GOTO    A OF  10, 20, 30

        62
         7
0   52   1    4
1   74        3
        12
        24
        36
```

```
60  DATA  -1, "ABC"

        74
        11
1 62     0
  100000
  000000
0 2      0
0 1      3

  040502
  041400
```

```
70 MAT READ #K:A[I]

    106
     11
0   67        0
0   64        0
0    4   13   4
0    3    1   1
0   12   11   4
0             0
0   11        0
```

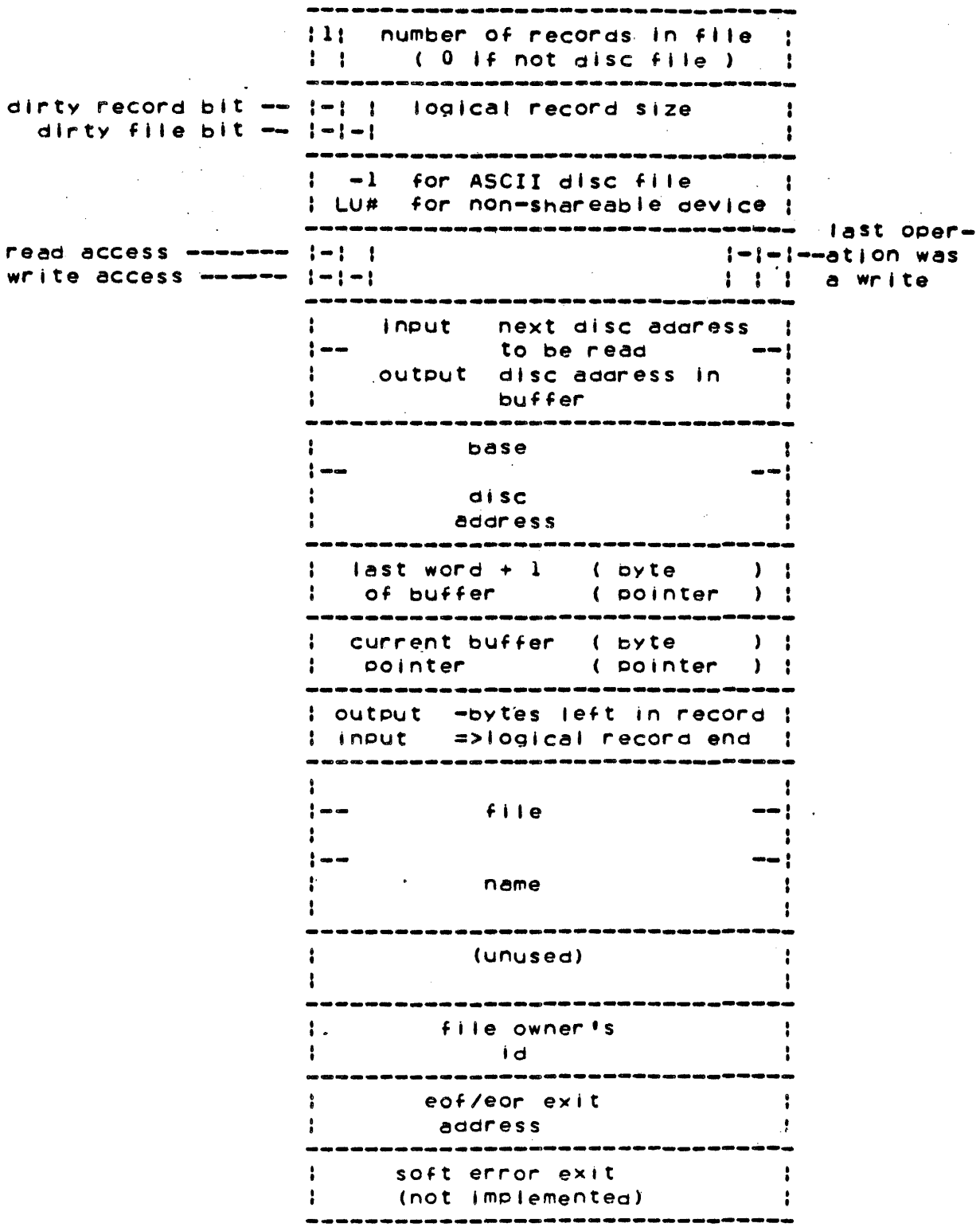
BASIC Variable Storage Allocation



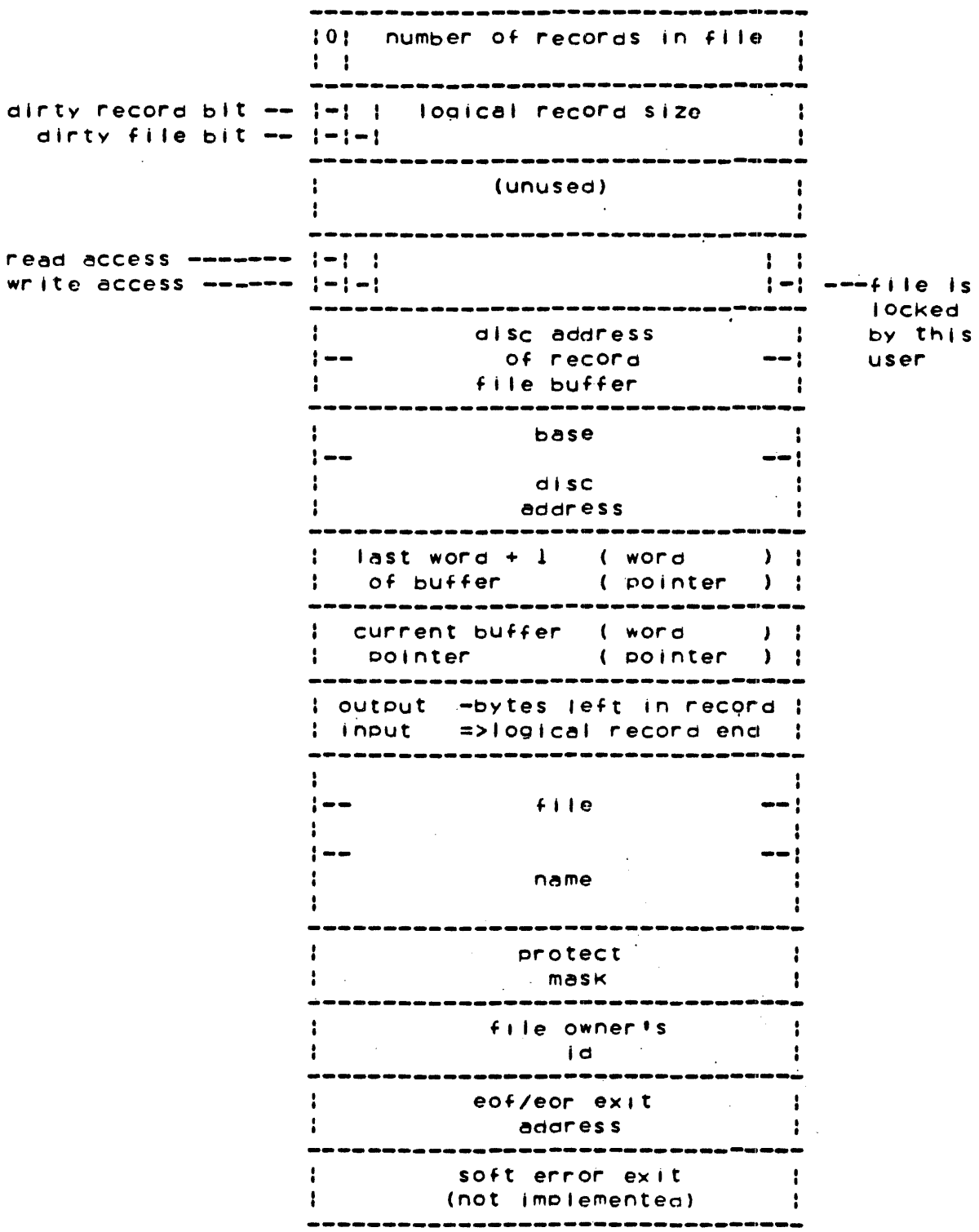
The symbol table consists of two-word entries, one for each unique symbol occurring in the user's program. The first word of an entry is the internal representation of the symbol as previously described. The second word of the entry is a pointer to the value of the symbol. For a programmer-defined function the value is the defining formula in the <DEF statement>. The value of a simple variable is a two-word floating point number. The value pointer of an array is its base address (i.e. the address of its first element); when an array is dynamically redimensioned to occupy less than its physically allocated storage, it occupies a contiguous block justified to the low core portion of its element space. Since array symbols may not have dimensionality locally defined (e.g. MAT A=B), array symbols may have a "don't know" entry in the symbol table in addition to the dimensioned entry. Both entries have the same value pointer. The declared and dynamic dimensions occupy the four words preceding the element space in the value table. The value of a string is also its base address. A string is a character array (packed two elements per word in contrast to the two words per element for numerical arrays). Its physical (declared) length and logical (dynamic) length occupy the word immediately preceding its value space.

The value table and common area are simply the concatenation of the values for the symbols in the program, excepting programmer-defined functions.

FILE TABLE ENTRY (ASCII files)



FILE TABLE ENTRY (BASIC files)



FILE_TABLE

The file table consists of one eighteen word entry for each file or place-holder ("*") in the FILES statement. Bit 15 of the first word distinguishes BASIC files from ASCII files. Bit 15 of the second word is set when an item is stored in the buffer, so that only records which are changed will be written to the appropriate device. Bit 14 is set when a record is written to the device and is used during program termination as a basis for updating the last changed date word in the file's directory entry. Bits 15 and 14 of word four are used to indicate that the user has read and/or write access to the file (if set). Bit 0 of the same word is set if the file is BASIC and the user has executed a LOCK statement. Bit 1 is used to indicate that the last operation on an ASCII disc file was a write.

A logical record-sized buffer is associated with each file table entry, and is accessed through pointers in the entry. An intra-buffer pointer designates the next portion of the record to be written or read. A fixed pointer to the first word not in the buffer acts as a bound on the intrabuffer pointer.

FILE_CONTENTS

There are 4 data types possible in a BASIC file. A string has bit 9=1 and the length in characters in the lowest 7 bits of the first word, followed by the string packed 2 characters per word. A two-word floating point number has the upper two bits of the first word different, except for a zero, which has both words zero. An end-of-file is a -1, and an end-of-record is a -2, in the first word.

Data written to or read from a file is first exclusively ORed with the fifteenth word of the file table entry. This has no effect, of course, unless that word is nonzero. It will be nonzero only if an ASSIGN statement has been used to specify the file, and the statement included a protect mask parameter. End-of-file marks, end-of-record marks, floating point zero, and the first word of strings are not masked. The entire buffer is masked or unmasked when written or read from the disc.

ASCII files contain only strings. ASCII disc files are blocked. Each record consists of a one word byte count followed by the actual character string. No carriage return or line feed is appended to the string - these are considered to be the record terminator only for user terminals. Non-disc ASCII files are not blocked. Each physical record equals a logical record and the length of the record is determined to be the length of the physical record.

2000/COMPUTER SYSTEM BASIC FORMATTED FILES

Block Size - logical size from 64 to 256 words
- default size is 256 words
- physical size is always 256 words

File Buffers in Core - one record's length buffer per file declared in program
- written to disc when different block of that file requested by user program
- written to disc by normal program termination
- buffers are swapped in and out with rest of user program

Access Modes - sequential
- like mag tape
- automatically pass over record marks until file mark encountered
- random
- logical blocks are addressable
- within a block, still sequential

Record and File Indicators - logical file mark (LEOF)
- creating a file places a logical EOF mark as first word in each block
- writing in that block erases that logical EOF mark
- user program may write LEOF using PRINT.....END (but continuing to write sequentially from that point will erase it)
- LEOF replaces LEOR that might otherwise be written
- logical record mark (LEOR)
- automatically written after last datum, if block not full, and if PRINT....., END not used
- physical record mark (PEOR)
- defines the absolute size a block; size defined by user creating the file
- physical file mark (PEOF)
- the absolute end of the last block of a file
- IF END.....statement will be executed in the circumstances described in the table on following page

	WRITE	READ
SERIAL	PEOF	PEOF LEOF
RANDOM	PEOF PEOR	PEOR LEOR PEOF LEOF

FILE_ITEM
DEFINITIONS

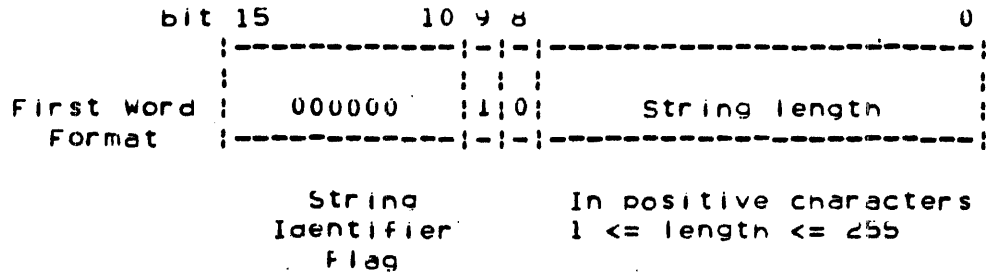
RECORD_MARKS

-2 = 177776(8) = logical record mark
logical record mark may also be detected as the physical end-of-record

FILE_MARKS

-1 = 177777(8) = logical file mark
logical file mark may also be detected as the last record mark of a file - i.e., the number of records in any file is pre-defined in the system

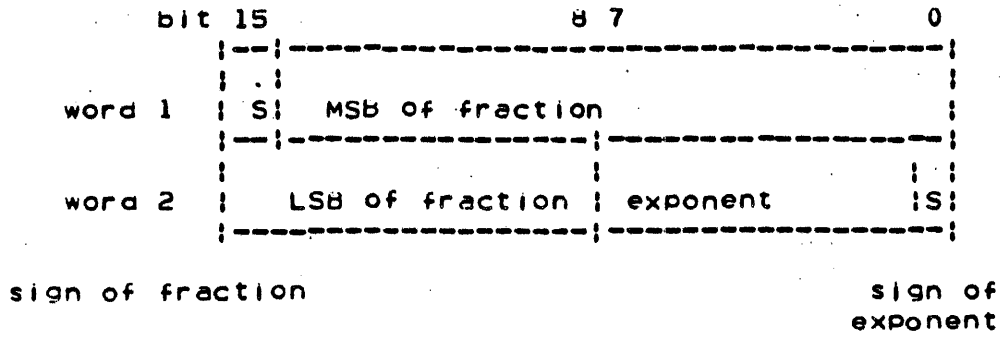
SIBINGS



ASCII String characters are then packed into consecutive words, two per word, left justified with byte wasted if odd length string.

FLOATING_POINT_NUMBERS

- identified by default (i.e., none of the above three)
- standard HP 2100 format



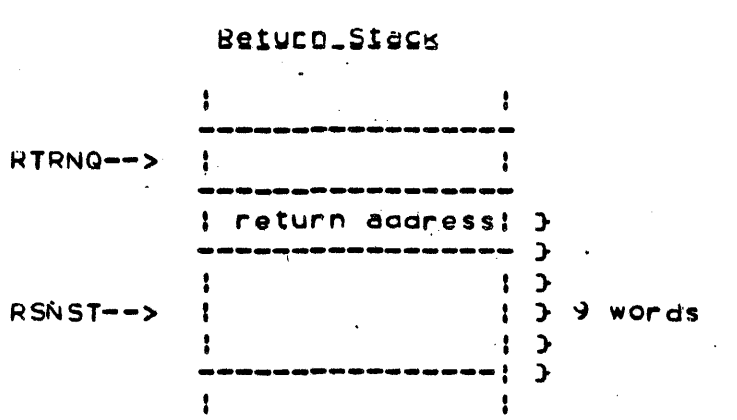
Update Last Change Date Routine

Each file and program entry in the directory has a word containing the hour of the year when the entry was last changed. It is necessary to update this word for files when a program is terminated for any of the following reasons: normal termination, CHAINing to a new program, error termination, abort and when a SLEEP or HIBERNATE command is issued.

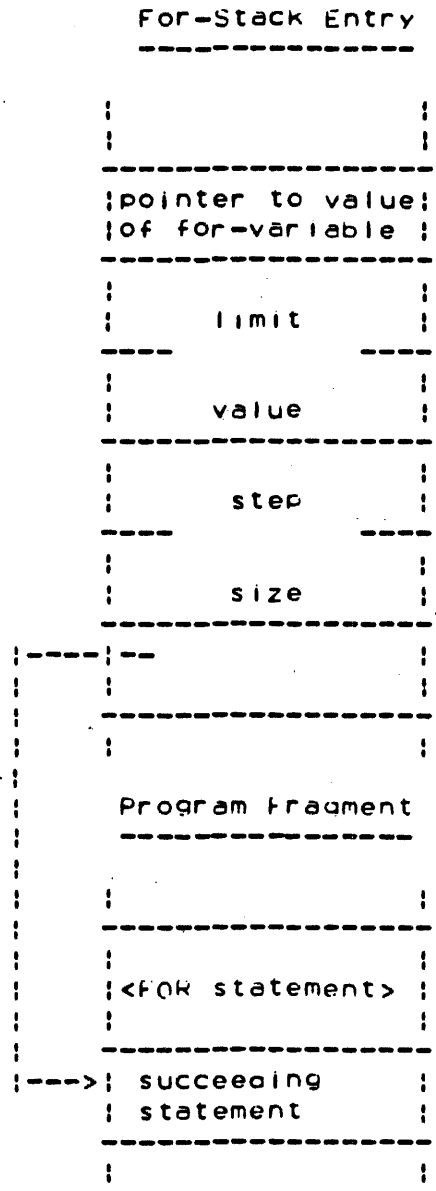
The DFCHK bit in the user's ?FLAG word in his TTY table is set to 1 if there were any files statements in the program. This determines whether the LCD routine will be called. When it is, each file table entry is examined. If bit 14 of word 2 = 1, the file has been written on, so the last change date must be updated. This bit is set by the WRBUF or OUTBF routines depending upon whether the file is BASIC or ASCII respectively.

The only abnormality in calling LCD occurs following an abort. The user is taken off the queue and re-inserted with priority 0 to run a core resident routine called ABUCD which writes the user to the swap track, calls LCD, and returns to the scheduler to finish aborting the user.

BASIC Run-Time Stacks

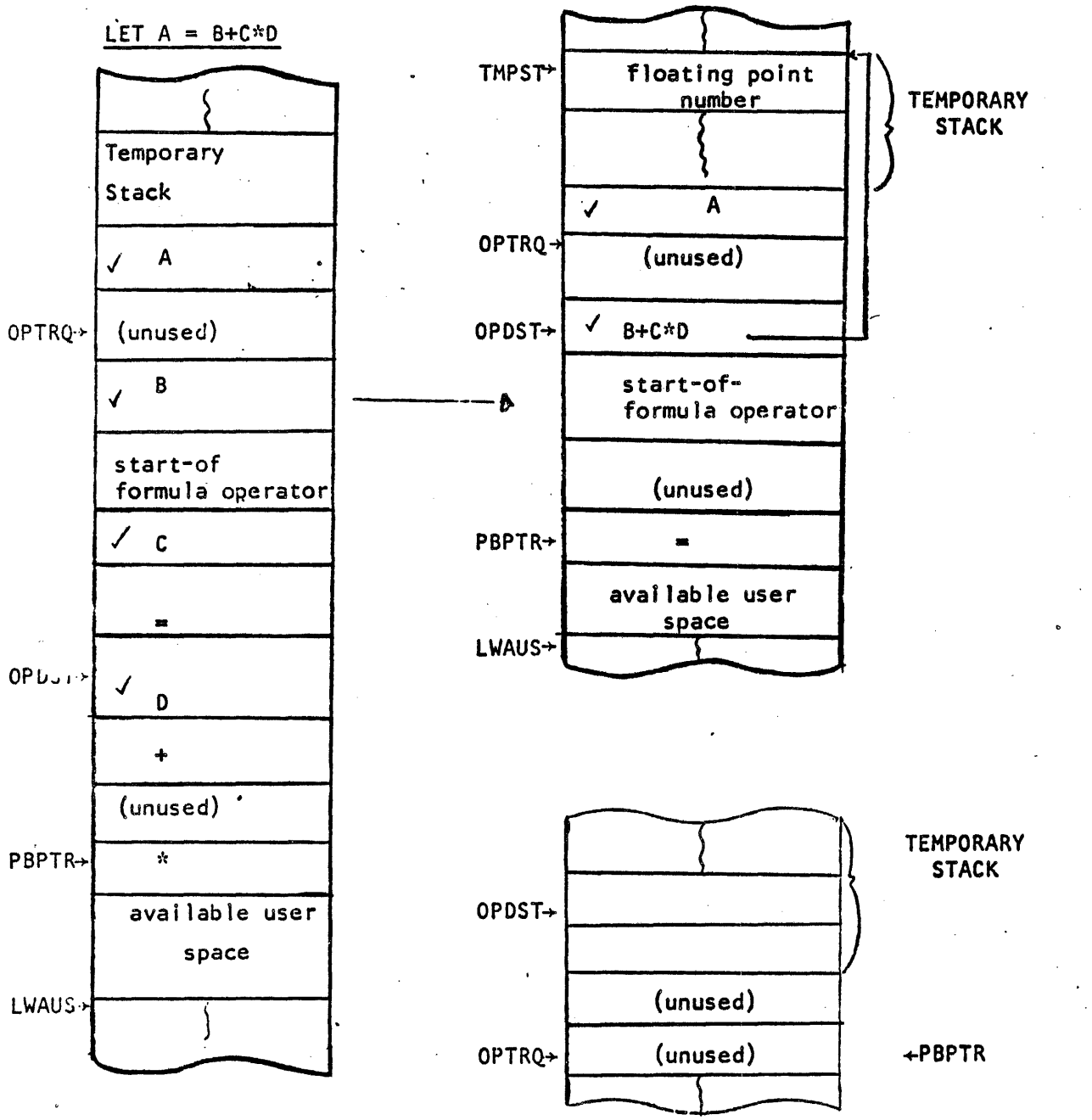


The return stack is of fixed length, holding from 0 to 20 one-word entries at any time. An entry is the absolute address of the statement following the GOSUB which placed the entry on the stack.



The for-stack is of variable length, containing one six-word entry for each for-loop which is currently active. Since the limit value and stop size are kept in the entry, they may not be changed within the for-loop. The value of the for-variable is the one kept in the value table, so this may be altered by statements within the for-loop.

OPERATOR/OPERAND STACK FRAGMENTS



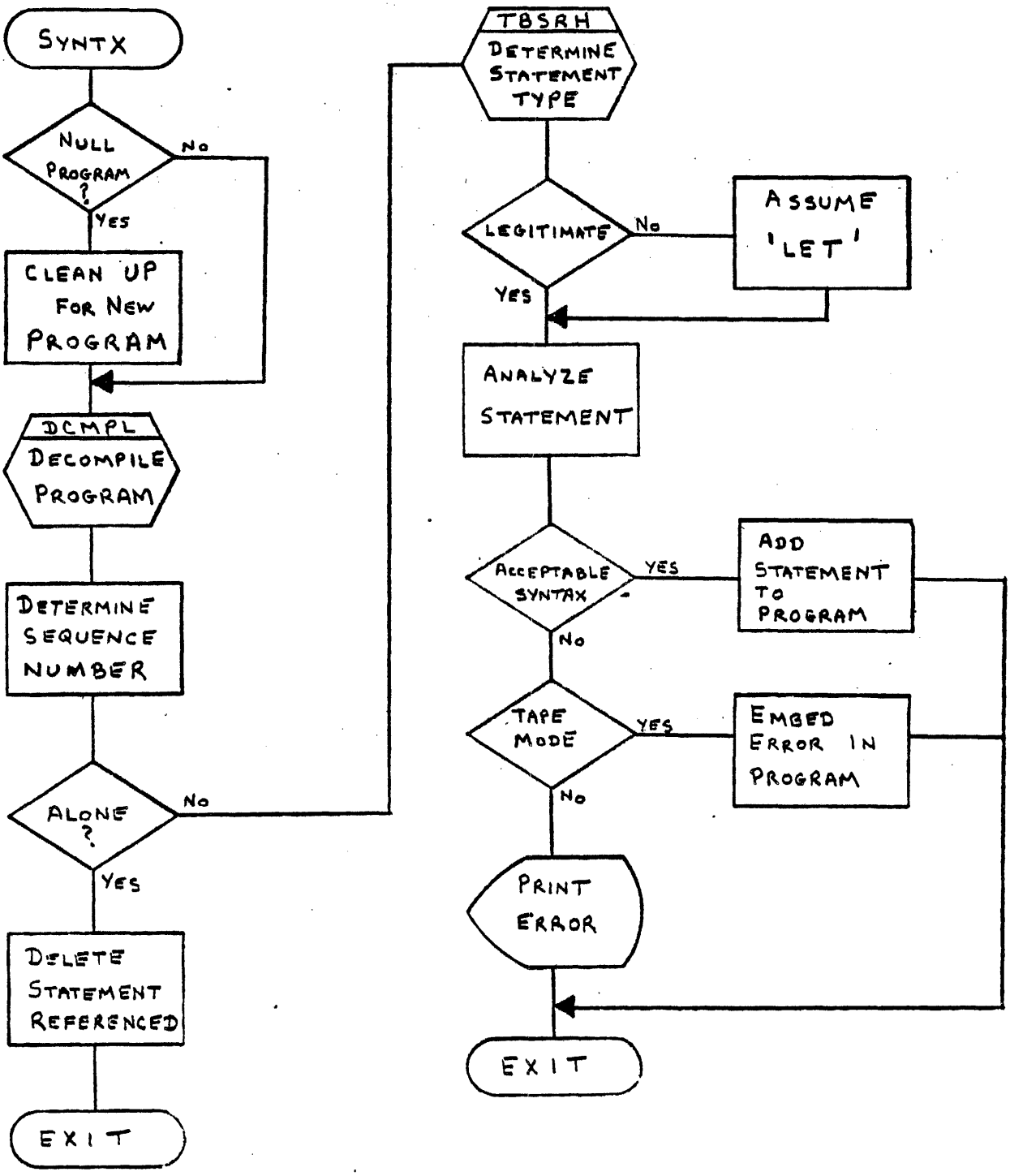
All operands (checked words) are addresses (i.e., C represents a pointer to the value of the simple variable C). Bits 7 - Ø of an operator entry contain the ors identifying code (See 'Basic Operators' Table) while bits 15-8 contain the operator's priority. Note the alternate-word structure of the stacks. The temporary stack holds intermediate values during the formula evaluation.

The two areas of core labelled SBJTB and USER contain the mechanism allowing different users to exercise different portions of the language processor without interference. The language processor makes its subroutine calls to the labels in the area beginning with USER. The word following a subroutine entry point is an indirect jump through the appropriate address in the area following SBJTB. When a user is displaced by the system, his registers are saved at USER and the area of core from USER to PBPTR.I inclusive is dumped onto his track of the disc. Thus, a complete record of the language processor's status with respect to him is preserved. The only thing particular to a user which remains when he is swapped out is his own teletype table.

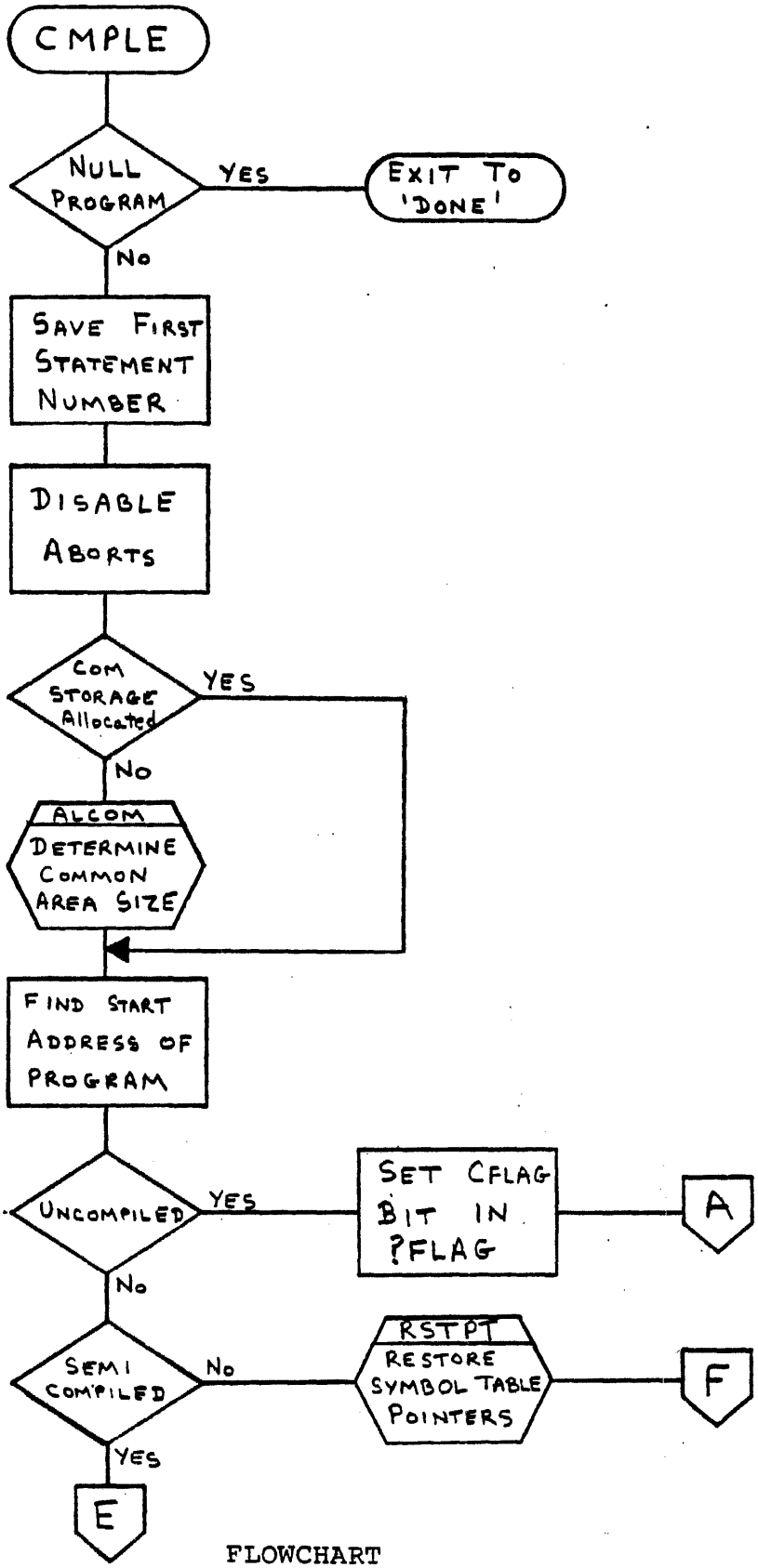
The tables headed by PDFTB (which must be in base page), SYNTb, XECTB, and FOJT are jump tables. The method in the last three cases is to compute a decision number, add the base address of the table, and transfer through the entry thus designated. The pre-defined function table is used by the formula evaluator to enter the code for evaluating pre-defned functions.

The tables headed by QUOTE and MCBOS have several uses. Their entries are explained in the listing and their use will be explained in those routines which access them. The Error Jump Table (at SERRS) is explained along with the error routines.

SYNTAX

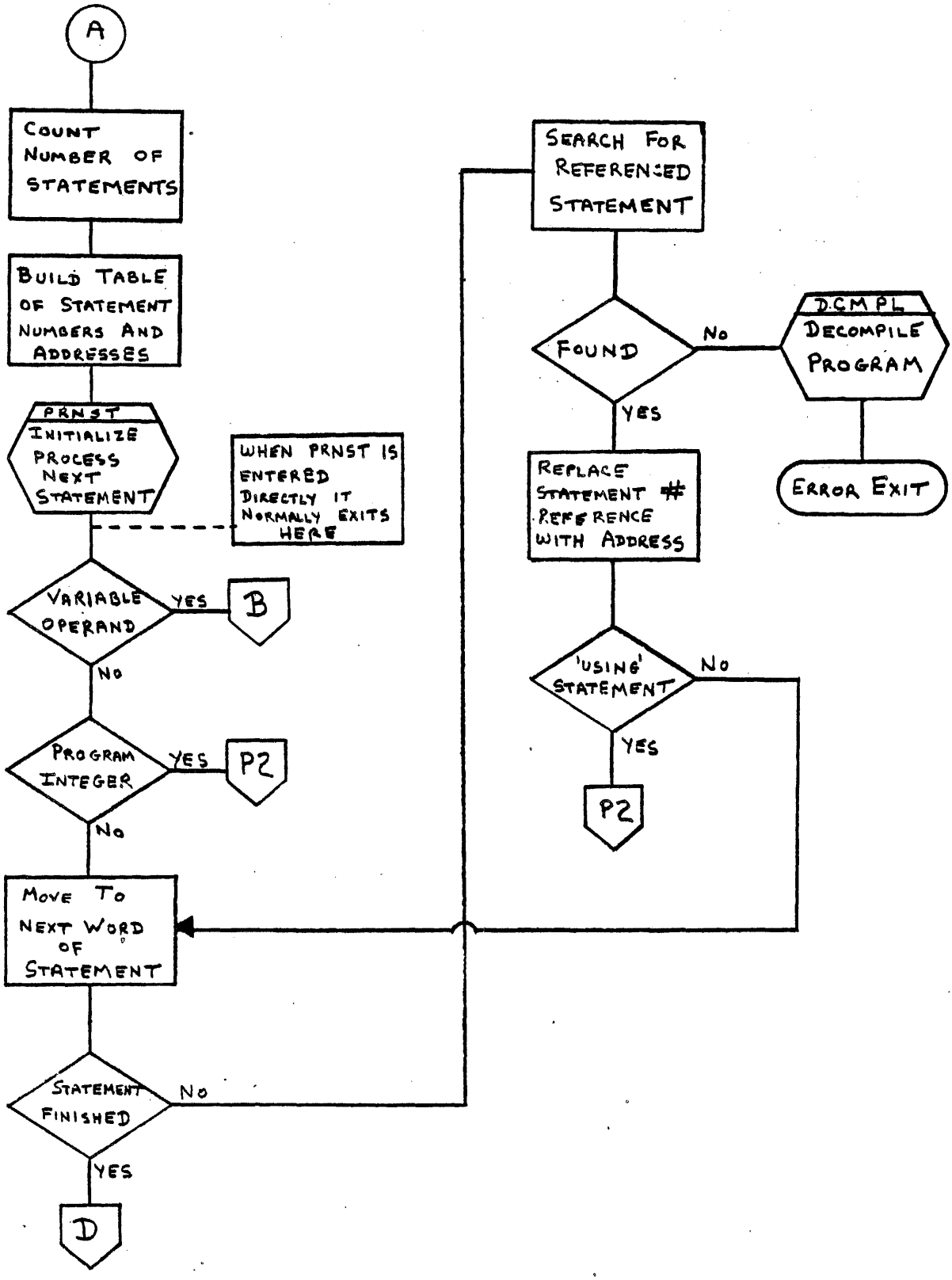

SYNTX
NULL PROGRAM ?
No
Yes
CLEAN UP FOR NEW PROGRAM
DCMPL
DECOMPILE PROGRAM
DETERMINE SEQUENCE NUMBER
ALONE ?
No
Yes
DELETE STATEMENT REFERENCED
EXIT
TBSRH
DETERMINE STATEMENT TYPE
LEGITIMATE
No
ASSUME 'LET'
Yes
ANALYZE STATEMENT
ACCEPTABLE SYNTAX
YES
ADD STATEMENT TO PROGRAM
No
TAPE MODE
YES
EMBED ERROR IN PROGRAM
No
PRINT ERROR
EXIT

COMPILATION



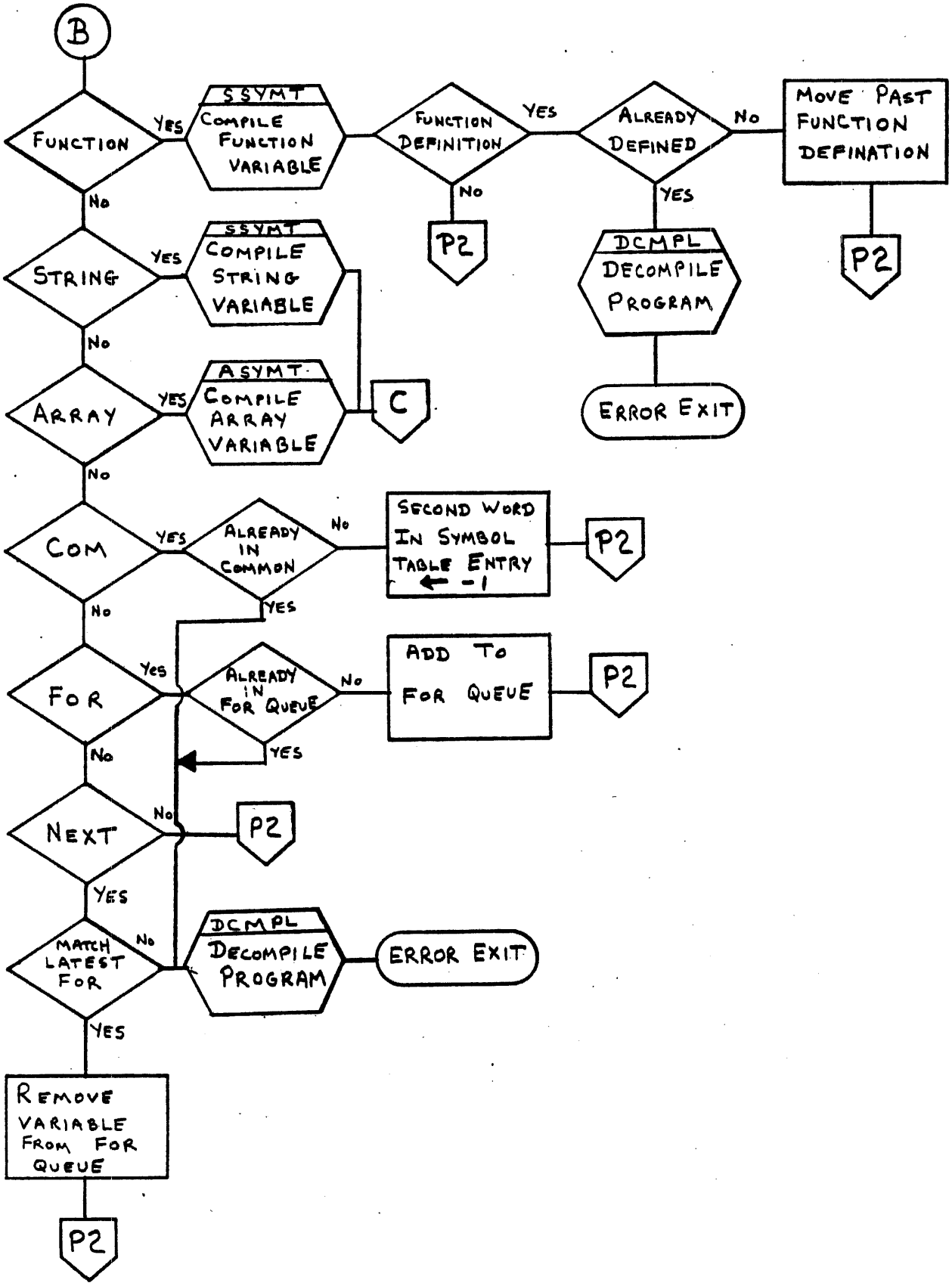
FLOWCHART
1 of 4

COMPILATION



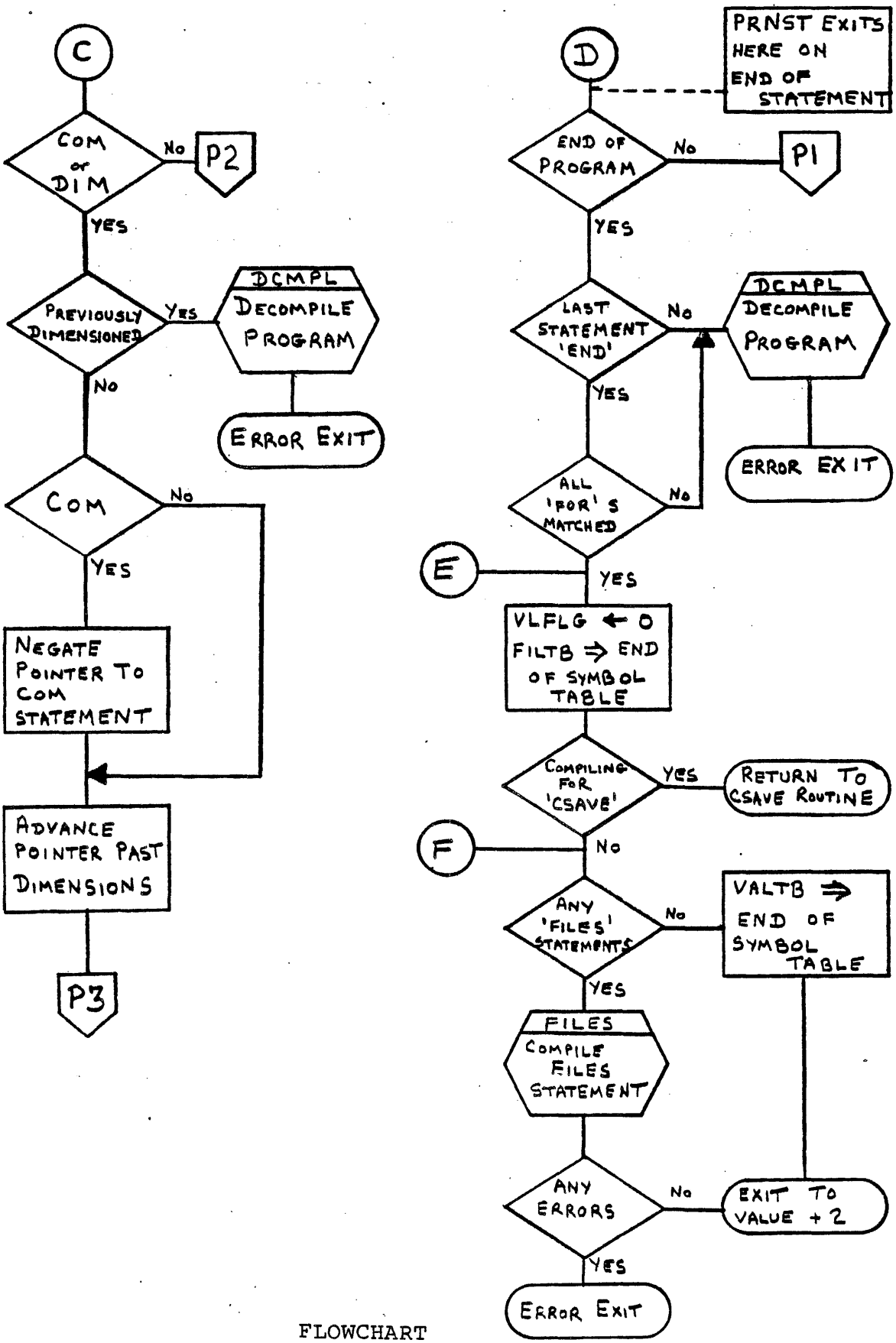
FLOWCHART
2 of 4

COMPILATION



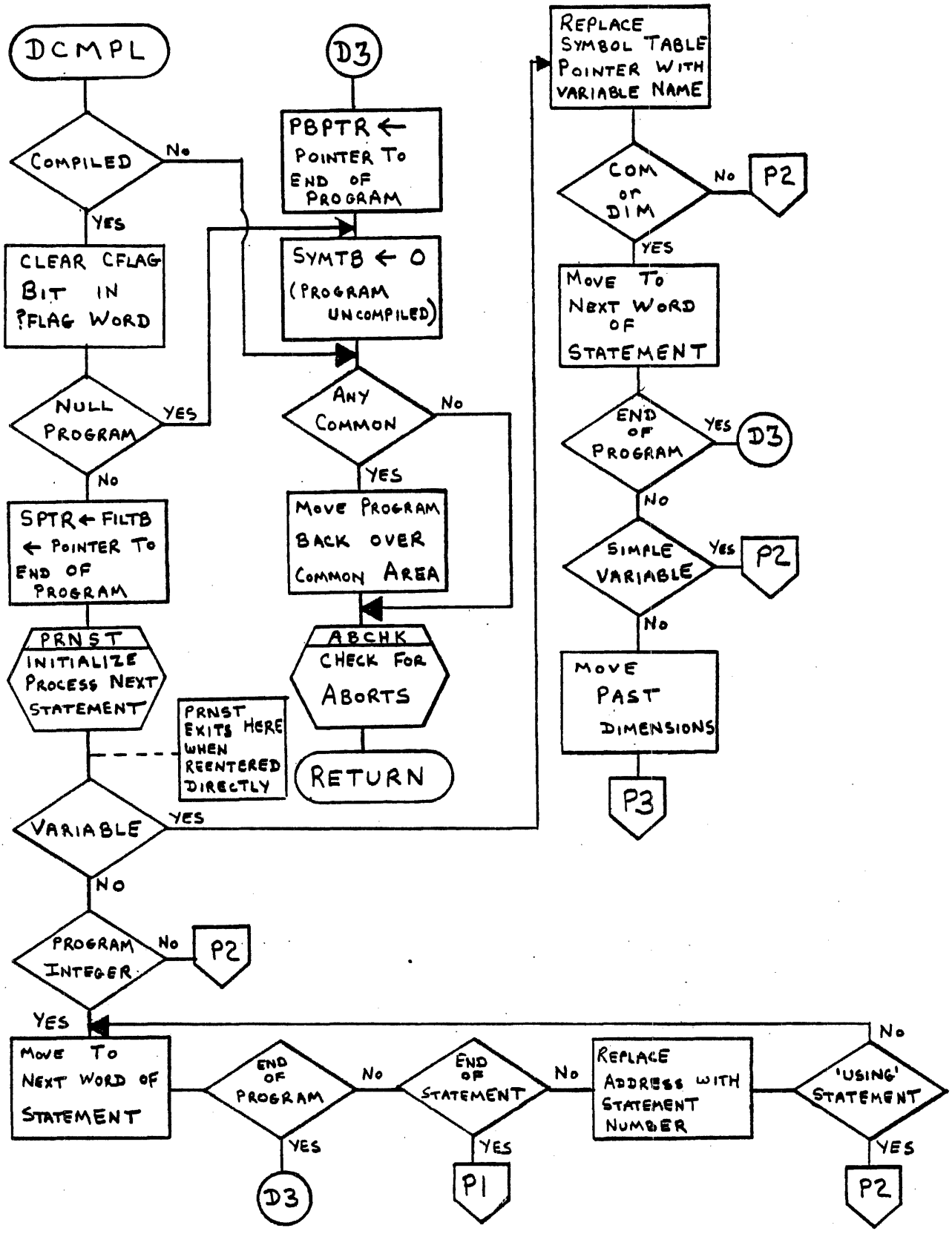
FLOWCHART
3 of 4

COMPILATION



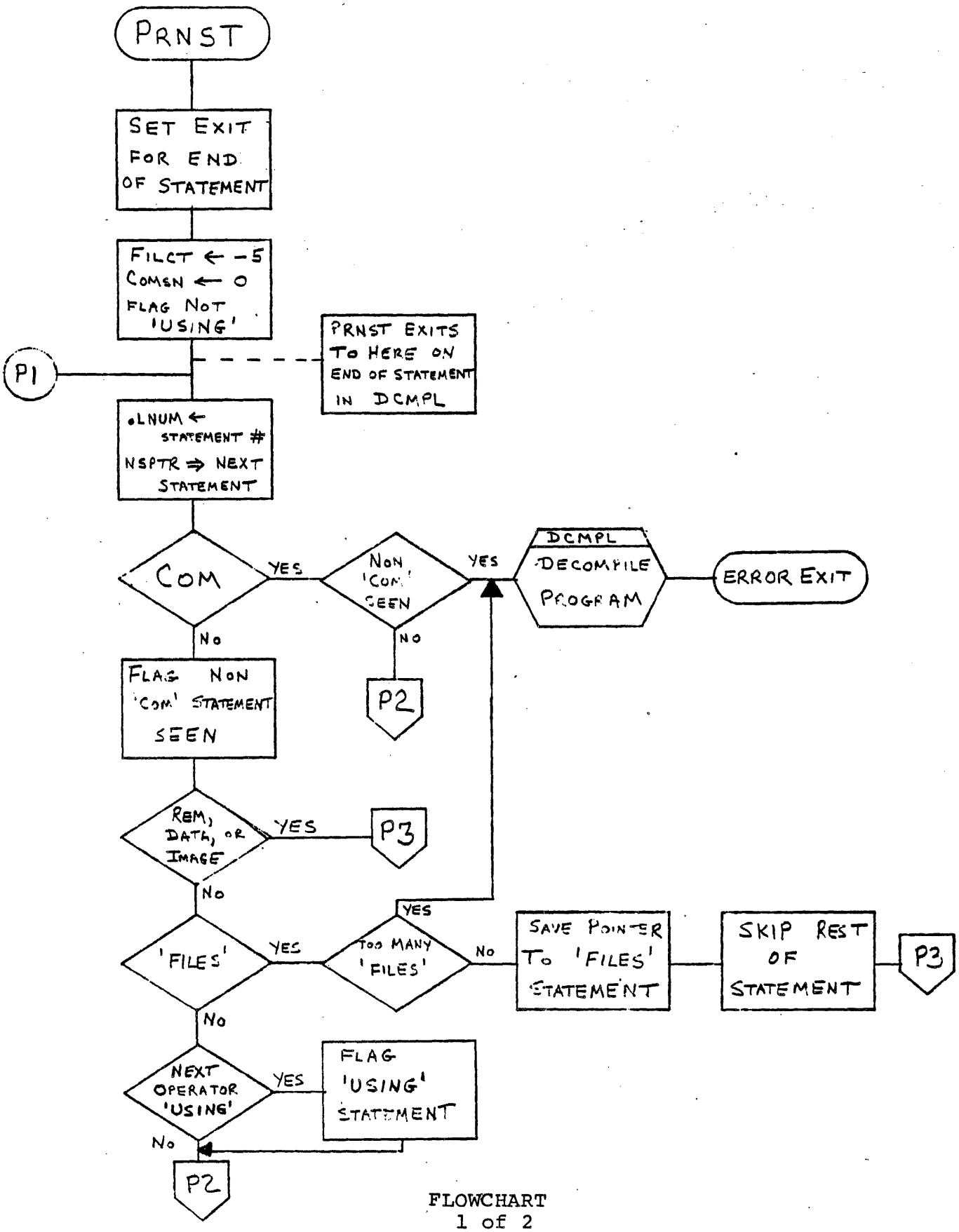
FLOWCHART
4 of 4

DECOMPILATION



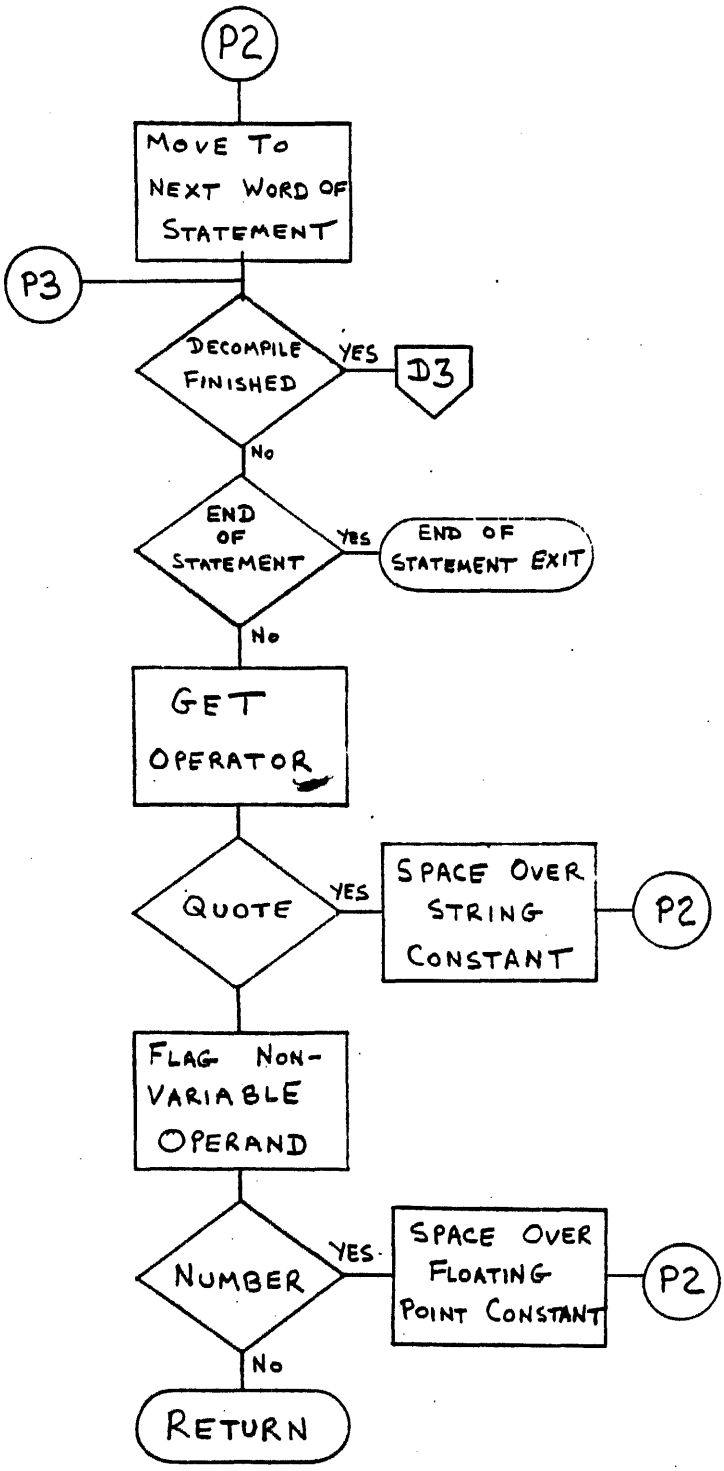
FLOWCHART
1

PRNST



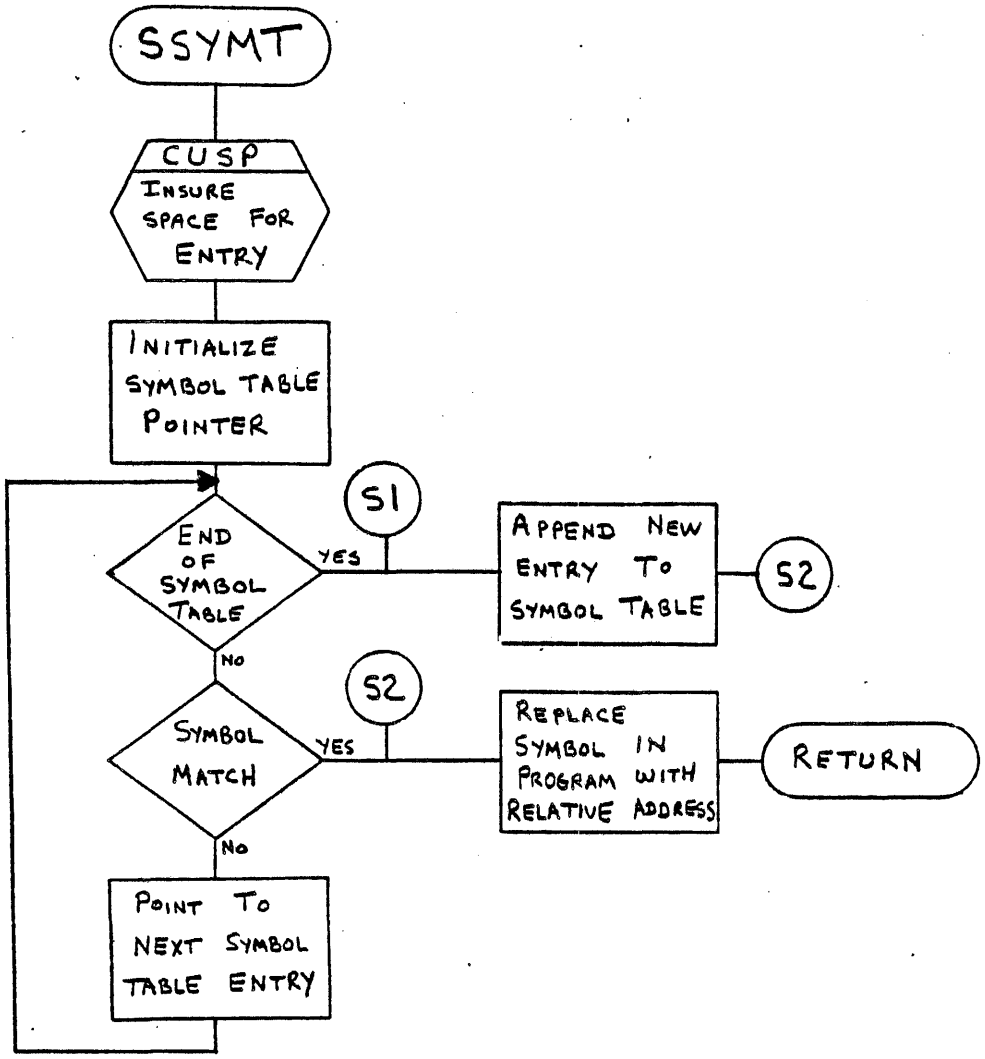
FLOWCHART
1 of 2

PRNST



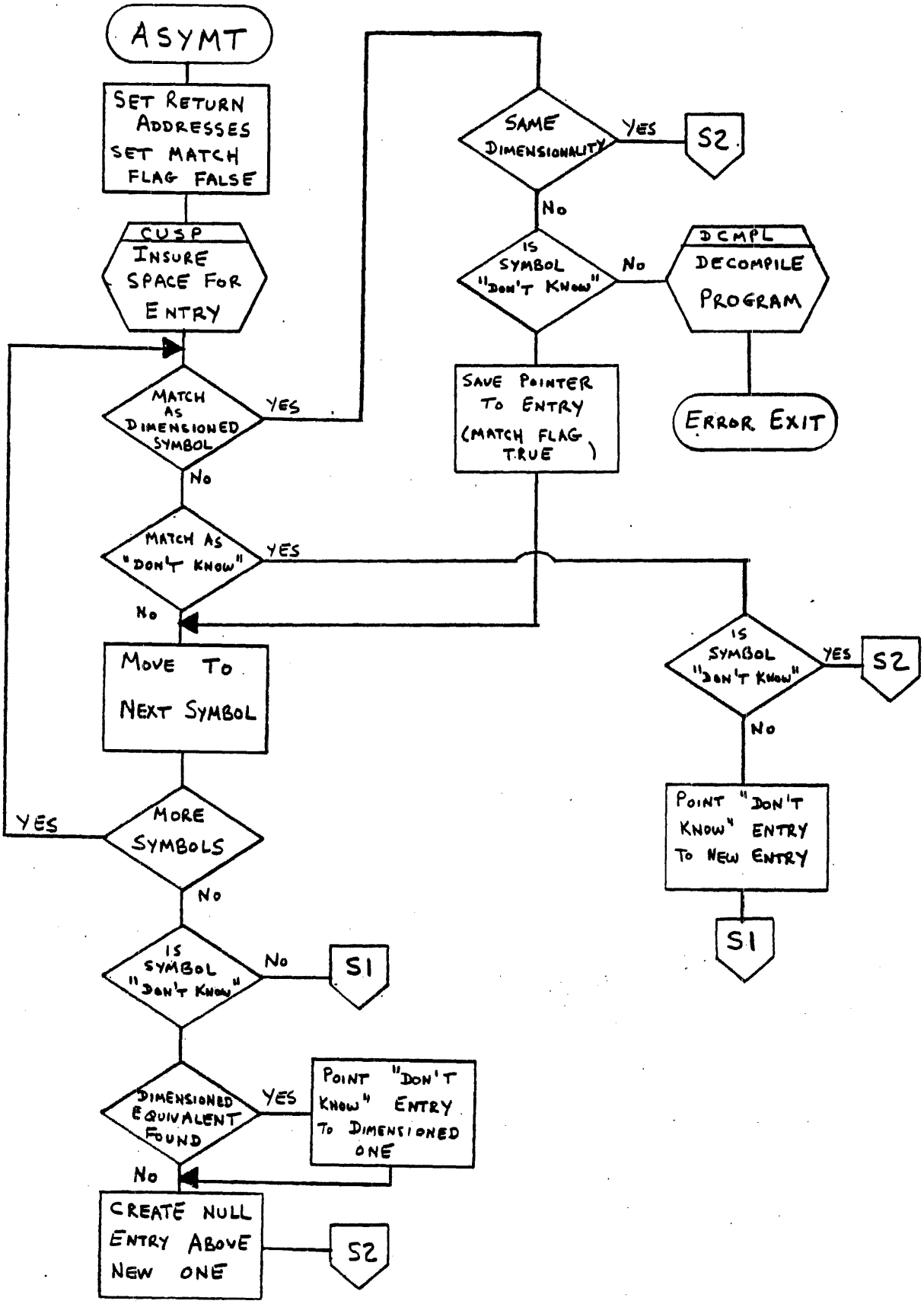
FLOWCHART
2 of 2

SSYMT



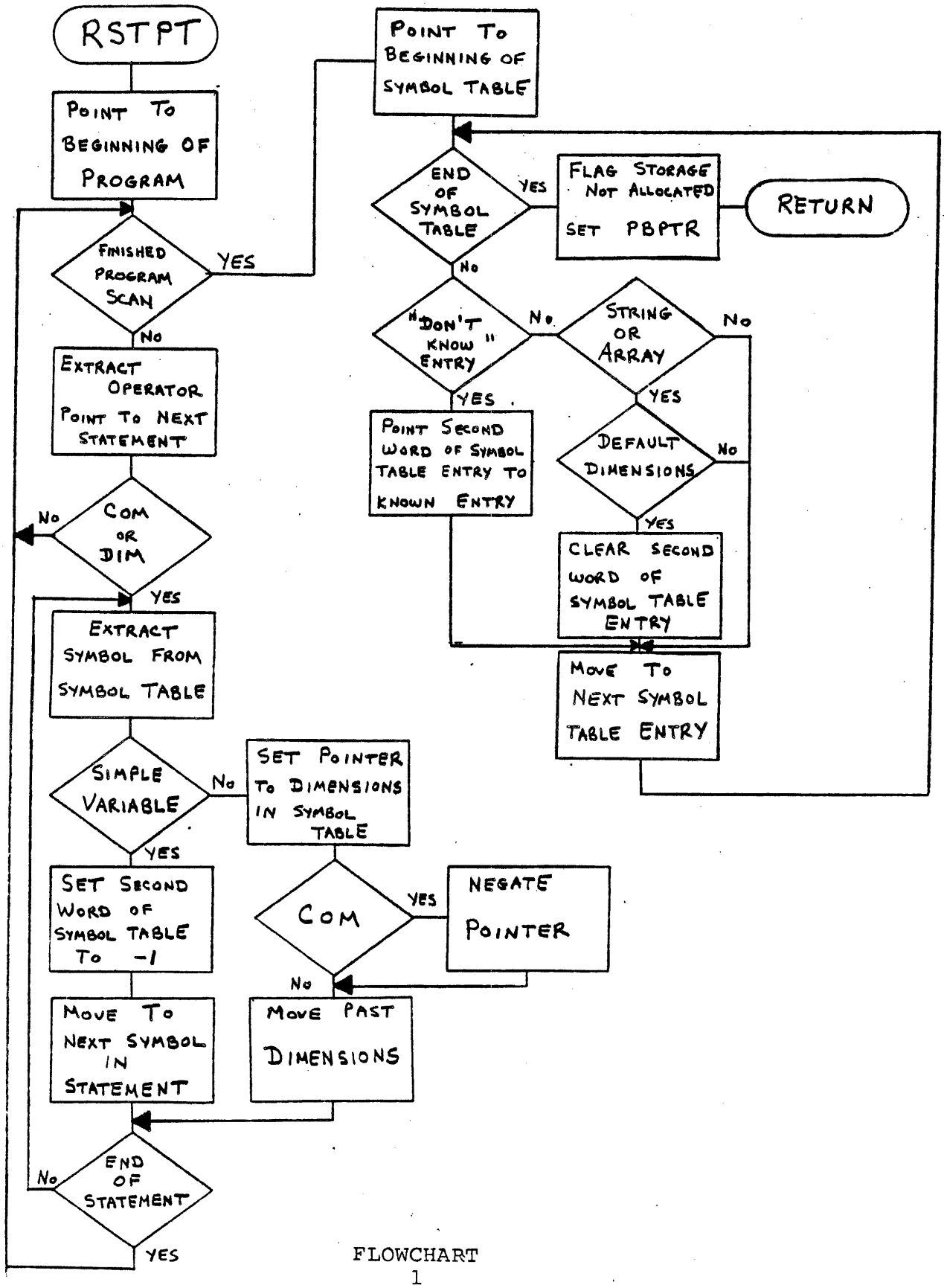
FLOWCHART
1

ASYMT



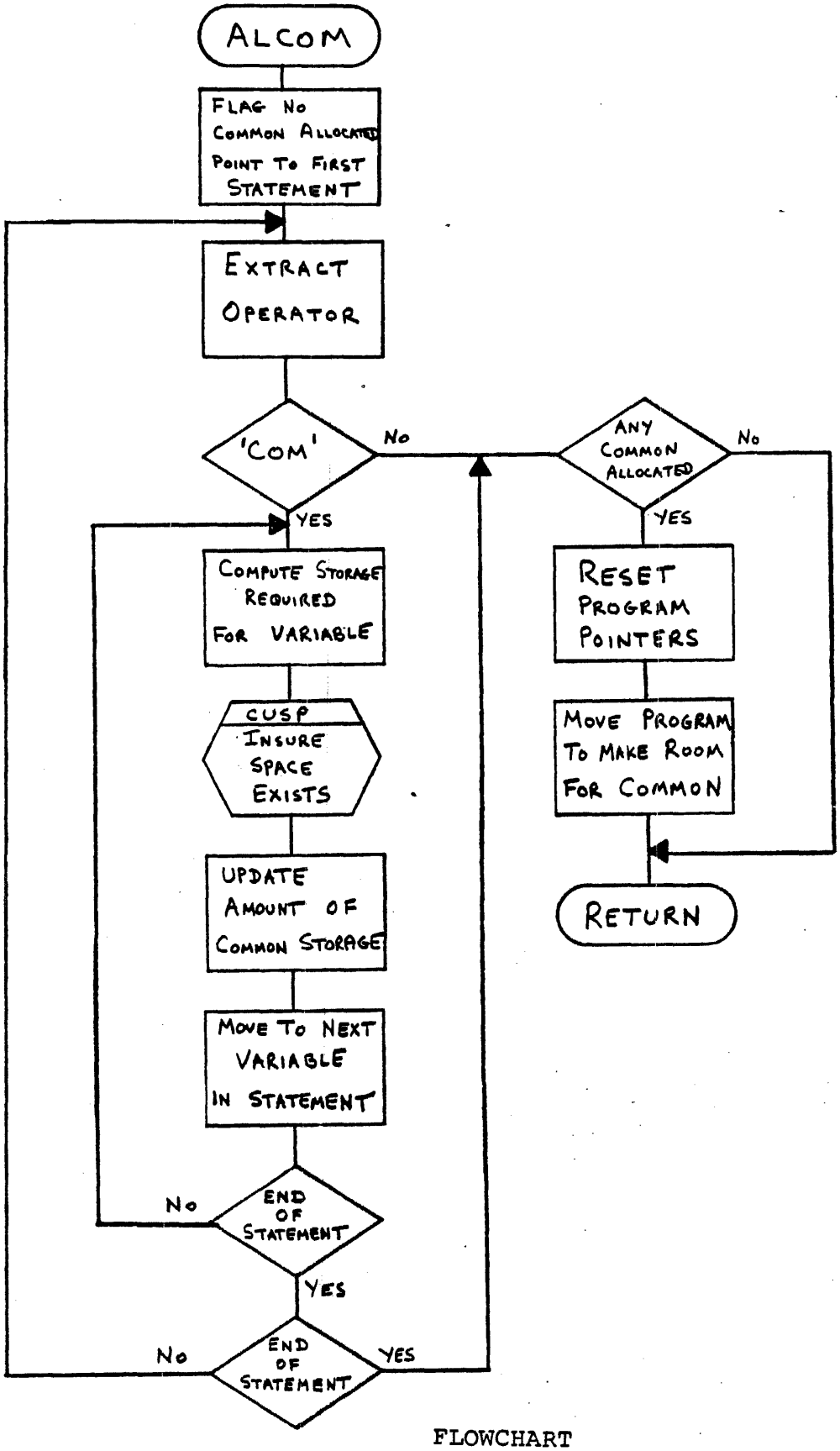
FLOWCHART

RSTPT



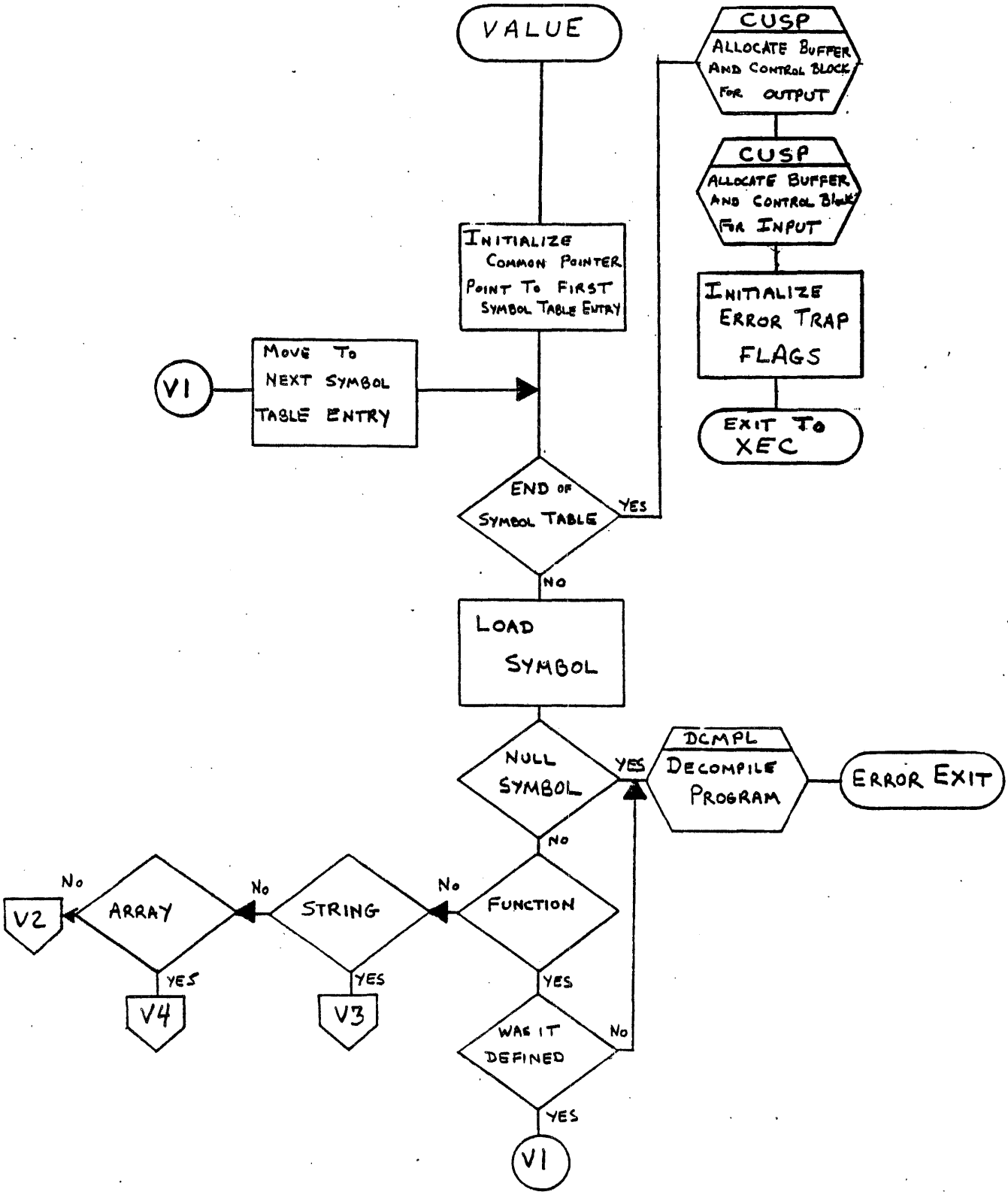
FLOWCHART

ALCOM



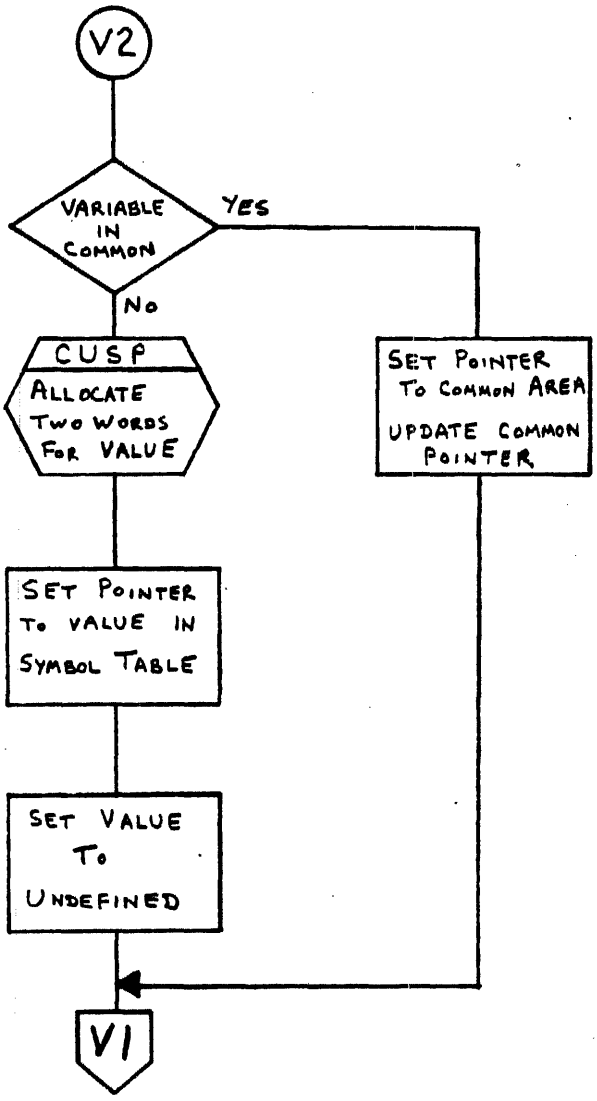
FLOWCHART
1

VALUE



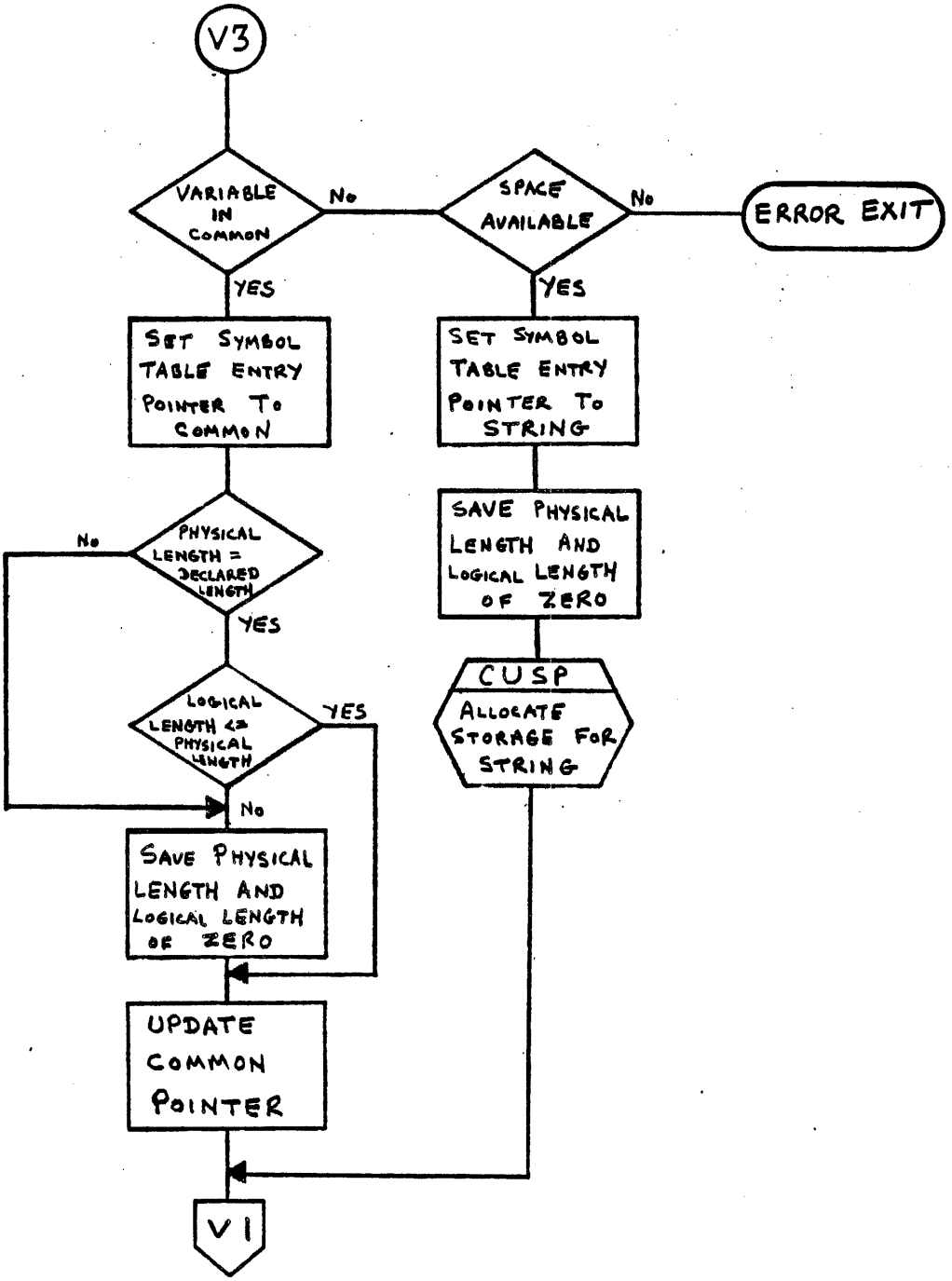
FLOWCHART
1 of 5

VALUE



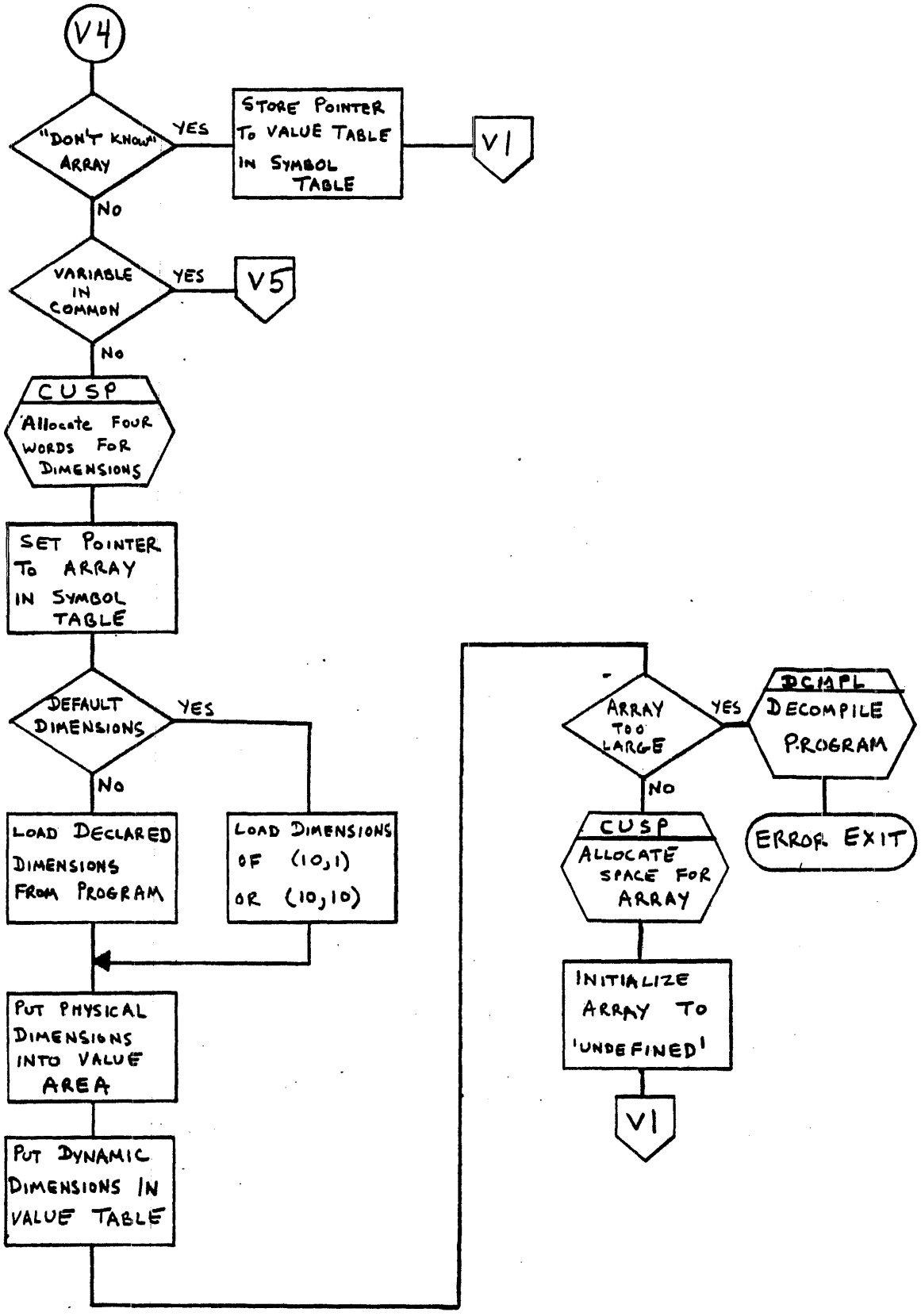
FLOWCHART
2 of 5

VALUE



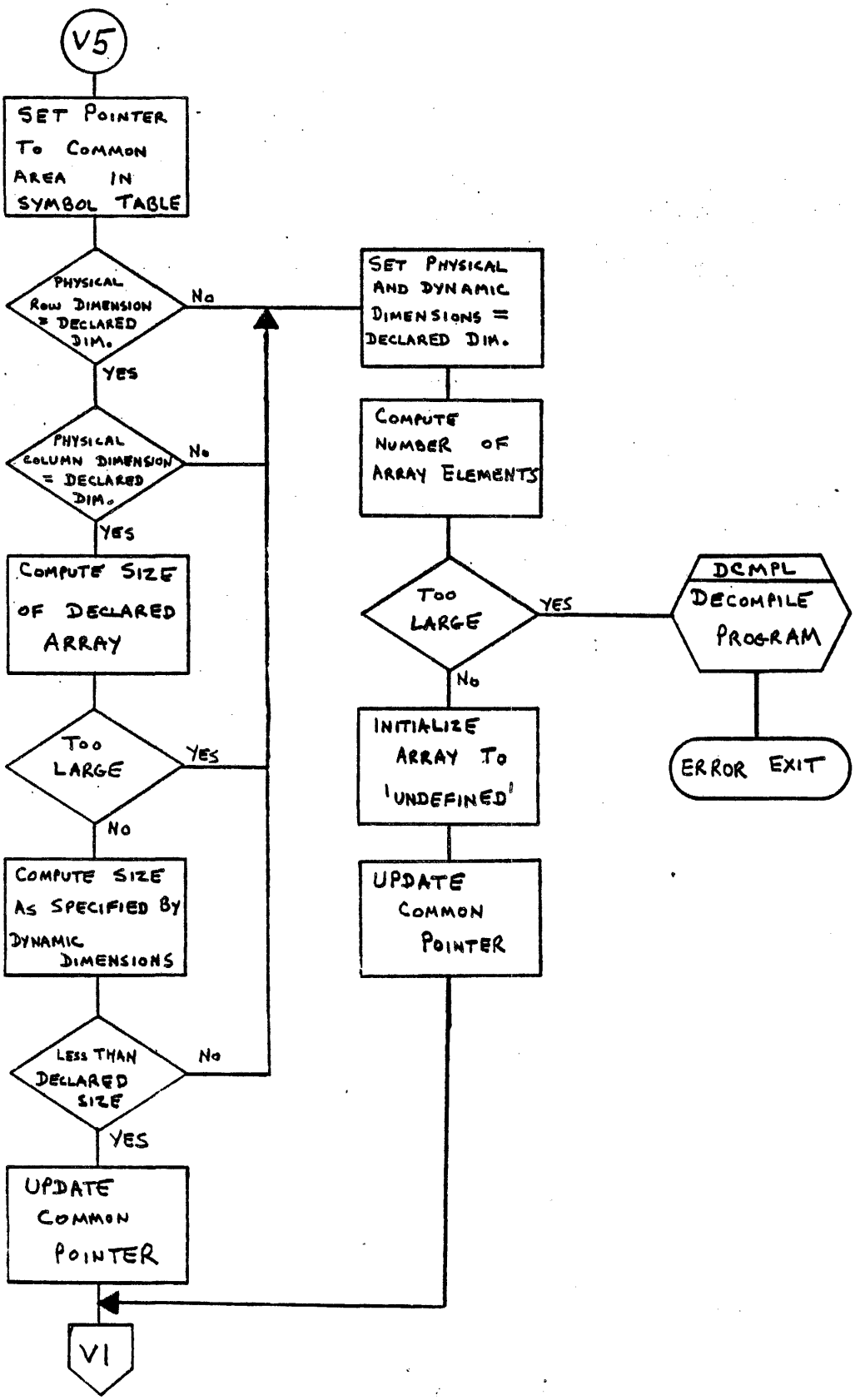
FLOWCHART
3 of 5

VALUE



FLOWCHART
4 of 5

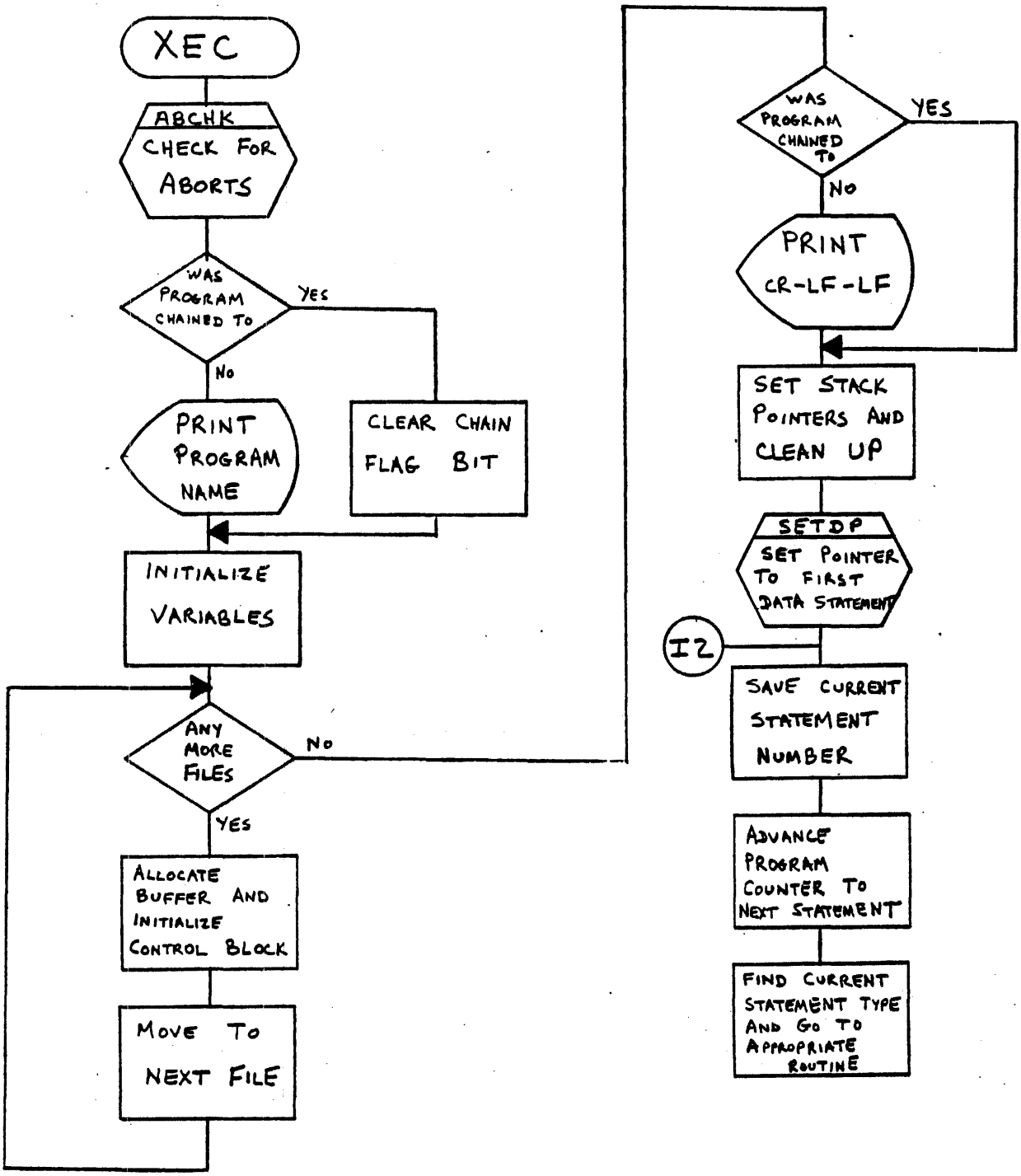
VALUE



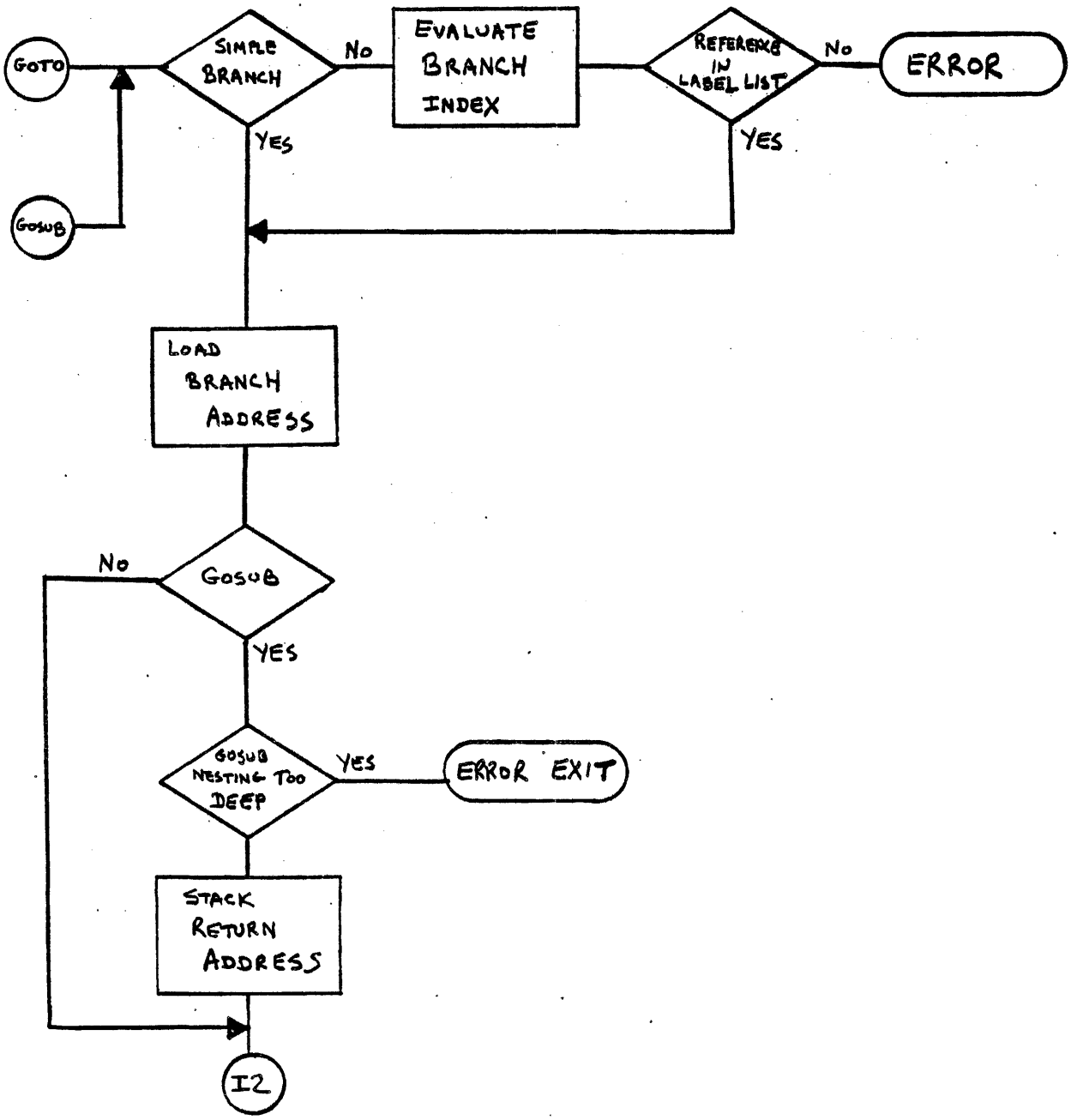
FLOWCHART
5 of 5

EXECUTION

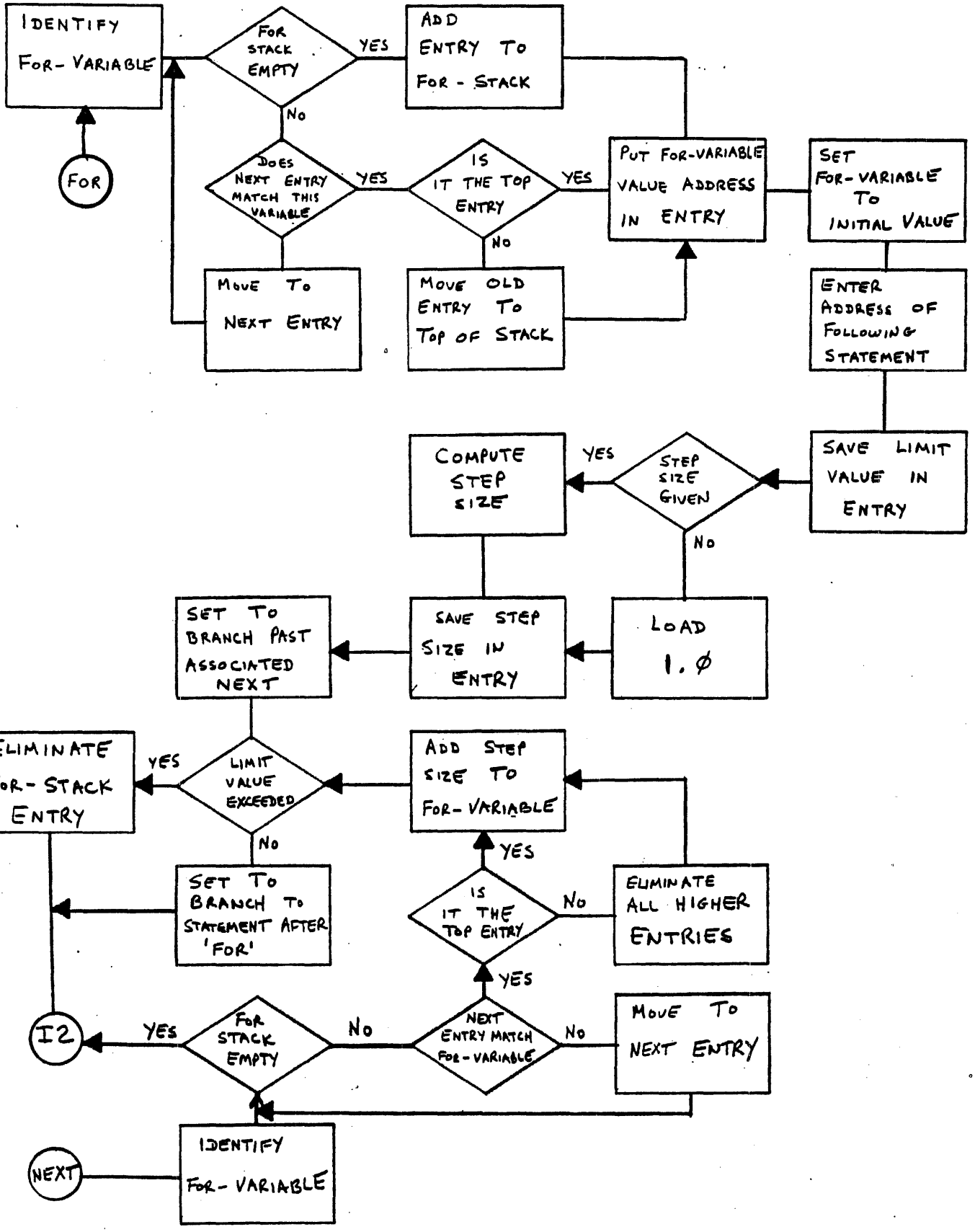
A. MAIN LOOP



B. SELECTED STATEMENT TYPES



FLOWCHART
2 of 2


IDENTIFY FOR-VARIABLE
FOR
FOR STACK EMPTY
YES
No
ADD ENTRY TO FOR-STACK
DOES NEXT ENTRY MATCH THIS VARIABLE
YES
IS IT THE TOP ENTRY
YES
No
PUT FOR-VARIABLE VALUE ADDRESS IN ENTRY
SET FOR-VARIABLE TO INITIAL VALUE
MOVE TO NEXT ENTRY
MOVE OLD ENTRY TO TOP OF STACK
ENTER ADDRESS OF FOLLOWING STATEMENT
SAVE LIMIT VALUE IN ENTRY
STEP SIZE GIVEN
YES
No
COMPUTE STEP SIZE
SAVE STEP SIZE IN ENTRY
LOAD 1.0
SET TO BRANCH PAST ASSOCIATED NEXT
ELIMINATE FOR-STACK ENTRY
YES
LIMIT VALUE EXCEEDED
No
ADD STEP SIZE TO FOR-VARIABLE
SET TO BRANCH TO STATEMENT AFTER 'FOR'
YES
IS IT THE TOP ENTRY
No
ELIMINATE ALL HIGHER ENTRIES
I2
YES
FOR STACK EMPTY
No
YES
NEXT ENTRY MATCH FOR-VARIABLE
No
MOVE TO NEXT ENTRY
NEXT
IDENTIFY FOR-VARIABLE

FLOWCHART
1

SYSTEM LIBRARY ROUTINES

APPEND

The APPEND routine is called by a user to append a library program to his current program. The operation is the same as GET for steps 1-8, and then continues as follows:

9. Check that the program to be appended is not semicompiled and has no common area. Set the date into word 5 of the directory entry and write it back.

10. Load user's current program and call DCMPL. Check that the program to be appended will fit, and if so, read it in at the end of the current program. If the read is unsuccessful, fail.

11. If the current program is not null, search it for the sequence number of the first statement of the appended program. If okay, update PRPTR.

12. If the appendee belongs to the user (?ID = ?PRID) then clear protection bits in the ?NAME words.

13. If the appendant's ID is not the same as the user's ID or ?PRID then put the appendant's ID into ?PRID. Also inclusive or the protect bit in the user's TTY table (the third word of ?NAME, bit 15) if the appendant is protected.

14. Set main and terminate.

ASSIGN

The ASSIGN overlay is used by BASIC to process an ASSIGN statement in a user's program. The function of the routine is to replace information in the file control block of the referenced file with information that pertains to the new file. We proceed as follows:

1. Find old file's directory entry and update it, if necessary.

2. Validate user's static access to the new file.

3. Write out directory entry of new file with updated last reference date.

4. Validate access to and allocate file if type ASCII.

5. Validate user's dynamic access to the file (use FUSS table).

6. Write out FUSS table with type of access granted.

7. Fill in new information into file control block. Adjust memory pointers if file buffer larger/smaller than that in this position.

8. Return to the language processor.

BYE

The BYE routine is used to log a user off. This routine may be called by the user as a command from his terminal or programatically, by using the SYSTEM statement. BYE is also called to log a user off when the IOP detects a disconnect on a port.

1. If there is no room in log table then requeue the user at the lowest priority. Restart at LIBRA.

2. If the user had an *OUT=* file then kill output.

3. Clear output wait bit in the flag word if set.

4. If ID is zero, the user is not logged on, then go to 15.

5. Clear ?PRID and ID.

6. If DFCHK=0 then there are no files that require their last change date updated. Go to 9. Otherwise read in the user program and call LCD to do it.

7. Release non-sharable devices user has in use.

8. Remove the user from the lock queue if he is on it.

9. Decrement the number of active port count.

10. Clear the user's FUSS table.

11. Put delog off entry into the log table.

12. Update the time used in the user in the IDT.

13. Send a log off message to the user via POSTR. If the IOP rejects the message POSTR will set the output wait bit in the user's flag word and take the reject return. Otherwise go to 16.

14. If the message was not rejected go to step 16. Otherise, loop to up to one minute while waiting for the output wait bit to be cleared by a buffer free (BFE) from the IOP. If timed-out go to step 15. If not go to step 13.

15. Send a POC to the IOP in the form of a LF to the terminal. This is required by the IOP to force it out of an undesirable state.

16. Send a hang user up (HUU) to the IOP.

17. Clear and set the unable to abort bit the user's flag word.

18. Clear main.

19. Exit to SCHED.

CATALOG

The CATALOG command lists all (or a line if called via the SYSTEM statement) of the programs and files stored on the system of a user. CATALOG lists, with three entries per line, the entry name a program/file descriptor, an access restriction program flag, the length and the record length.

NAME	- the name of the entry.
PROGRAM/FILE DESCRIPTOR	- A/F/M/C/BLANK ASCII FILE/FILE MWA FILE/CSAVED PROGRAM/PROGRAM
ACCESS RESTRICTION	- U/P/L/BLANK UNRESTRICTED/PROTECTED/ LOCKED/PRIVATE
LENGTH	- the length in blocks if a file or program, a device designator if it was a non-shareable device.
RECORD LENGTH	- print the record length in bytes for an specifiec device designator or in word for files #256.

The operation of the command is as follows:

1. Set the ?TEMP words to blanks
2. Check if a starting entry name has been specified. If so, put the name into the ?TEMP words.
3. Decrement the last byte of the name in ?TEMP by one.
4. If this was a call from the system then go to step 6.
5. Get a buffer from LOUT and build the heading in it. Call LOUT to print it.
6. Move the user's ID into LTEMP and move the entry name in ?TEMP into LTEMP (:3).
7. Get a buffer from LOUT to build the catalog line.
8. Call DLOOK to find the first directory entry beyond the

one sought in (LTEMP 0:3).
9. If the idcode of the new entry is the same as the one sought then go to step 13.
10. If the line contains at least one entry go to 12 to print the line and terminate.
11. Exit to the system statement clean up routine if catalog was called from the language processor. Otherwise, terminate.
12. Set the successful write return in LOUT to terminate (LLEND). Go to step 18.
13. Put the name, the program/file descriptor, the access restruction, the length and the record length into the buffer for the entry.
14. If this is not the third entry on the line then go get the next entry. If there are no more entries then go to 18.
15. Put the name of the third entry on the line into ?TEMP.
16. Set up LOUT to move LTEMP (1:3) to ?TEMP words if the write is rejected in LOUT. This will insure that catalog will resume processing at the correct entry.
17. Set the successful return of LOUT to step 4. The fail return is also step 4.
18. If catalog was called from the language processor then exit to the system statement clean up routine (SYSCU) with the line.
19. Print the line and suspend.

CHAIN

The CHAIN routine is used by BASIC to process a CHAIN statement in a user's program. The function the CHAIN routine is to find the program named in the CHAIN statement, retrieve it from the disc, and begin execution. It operates as follows:

1. Dump file buffers.

2. Update the last changed date entry in the directory for each file which was written on.

3. Translate name of program from CHAIN statement. Invalid names exit to error. If preceded by a"$", set up A000 search; if preceded by "#", set up group library search; otherwise set for searching on user's ID. Save the line number if any is specified.

4. Perform directory search. Exit to error if not found.

5. Check to make sure that the entry is a program, that it is not illstored, and that it will fit. If any of these are not true, exit to the appropriate error.

6. Update date entry in directory and write directory track back to disc.

7. Read in the basic portion of the previous program, including the common area and then append the new program. If the read is unsuccessful, read in the previous program again and exit to error. If successful, move the new program name into the user's table, and if this is a run only program, set the run-only bit, unless the program is in this user's own library. Call SEMIC, which sets up pointers for the language processor, dependent upon whether the program is uncompiled, or semicompiled. SEMIC also sets the "program unaltered bit" in the user's TTY Table.

8. Check if an abort was attempted during the previous steps, and if so, abort the user.

9. If a line number was specified, search the program for the statement and, if found, put its absolute address into PRGCT. If no line number was specified, set PRGCT equal to SPROG. If the program is null, or if the line number cannot be found, clear the chain bit in the flags word. In any case, exit to SCHDL.

CSAVE

The CSAVE routine is called by a user to save a program in semi-compiled form. This is the form it has after the symbol table is built. CSAVE operates like SAVE with the following exceptions.

3. If program is not compiled then allocate common if necessary, change the user status to syntax, initialize some compile routine variables and enter the compile routine. Return to step 3a when done compiling.

3a. Call RSTPT to restore the symbol table to its appearance just after it was built. Restore the user's status to CSAVE.

5. Check if program will fit in swap area (if the limit pointer (FILTP) plus 6 is ≤ LWAUS), if not print PROGRAM TOO LARGE and exit. Use this new value for space requirements calculations.

12. Fill in the six words after the symbol table:
    1. symbol table pointer
    2. # of file statements in the program
    3-6 the addresses of these file statements

15. Set the semicompiled bit in the directory entry template (LTEMP+3, bit 15).

DELETE

The DELETE command allows a user to delete a section of his program. He can specify two parameters, M and N. M refers to the first line to be deleted, N to the last. If N is not specified, the entire program is deleted, starting at line M. The operation is as follows:

1. Translate and check parameters. If N is not specified, set it to 9999.
2. Decompile program.
3. Locate range of statements to be deleted.
4. Move portion of program following deleted area up against portion preceding.
5. Reset PBPTR and exit.

DEVICE

The DEVICE command prints a list of devices on the system the user may access along with the device's select code and status. The status is N/A if the device is assigned to a name or to an ID not the same as the user's. It is busy if the device is being used.

1. Get the user's IDT entry and extract the capability word put into the first word of ?TEMP.

2. If there are no device table entries then terminate.

3. Initialize the device table entry-to-process pointer (stored in the 2nd word of ?TEMP) to the first entry of the device table. Also initialize a pointer (3rd word of ?TEMP) to the end of the device table +1.

4. Print the two line heading.

5. Copy the ?TEMP words to a three word buffer (pointed to by UDVTB).

6. Get the device table entry-to-process pointer. If this pointer goes beyond the device table (?TEMP+1=?TEMP+2) then terminate.

7. Get the device designator from the device table. If the user does not have the access capability then bump the device

table entry-to-process pointer to the next device entry and to to step 5.

8. Get a buffer from LOUT to build the data line.

9. Put the device designator, the select code and the status into the buffer along with the record terminator (XOFF, CRLF).

10. Set one of the parameters in the LOUT call to cause LOUT to swap ?TEMP with the 3 word buffer)pointed to by UDVTB) if the print is rejected. This will cause the routine to restart processing at the right device table entry.

11. Set the return address from LOUT to step 5 and call LOUT.

DIRECTORY

The DIRECTORY allows a user on A000 to list library programs and files according to the user idcodes. The line for entry include:

ID	- the id code of the owner, printed only for the first entry of account
NAME	- name of the program/file
PURGE DATE	- the last reference date
PROG/FILE FLAG	- A/F/M/C/BLANK ASCII FILE/FILE/FILE WITH MWA/ CSAVED PROG/PROG
PROTECTION FLAG	- U/P/L/BLANK UNRESTRICTED/PROTECTED/LOCKED/ PRIVATE
LENGTH	- in block of prog/file
DEV/ADDR	- disc address if disc file or device designator if non-shareable device
RECLEN	- include for files if its record length #256

The operation is as follows:

1. Check if the user's idcode is A000. If not, print PRIVILEGE COMMAND and exit.

2. If a starting idcode has been specified then decrement the idcode by one and store it in ?RTIM. Otherwise store 1 in ?RTIM.

3. Build the system heading consisting of the system id, the date and the time.

4. Try to output the heading and suspend. If the write was rejected go to 3.

5. Try to output the directory heading and suspend. If rejected go to 5.

6. Set LTEMP with the ID saved in ?RTIM and LTEMP(1:3) to -1s.

7. Set the previous write flag to successful (0) and entry found flag to not found (0).

8. Call DLOOK to find the entry (in LTEMP 0:3). Set the entry found accordingly.

9. Get a buffer from LOUT to build a new line of the directory.

10. If the previous write flag indicates the last write was successful then get the next entry.

11. Save the new entry's idcode in ?RTIM.

12. If the idcode is -1 (psuedo entry) then terminate.

13. Set bit 15 of ?RTIM if the new id code is not the same as the idcode of the previous entry (now in LTEMP).

14. This forces the idcode to be included in the data line.

15. Build the line of directory information. Include the idcode if bit 15 of ?RTIM is set. Insert the entry name. The program code, the protection code the length, the device/address and the record length.

16. Put the entry name in the ?TEMP words.

17. Try to output the line and suspend. If the write failed, set the previous write flag to fail (1) and go to step 19.

18. Set previous write flag to successful (0) and clear bit 15 of ?RTIM.

19. Set entry found flag to not found (0).

20. Move the idcode into LTEMP from ?RTIM and move the entry to LTEMP (1:3) from the ?TEMP words.

21. Go to step 6.

Echo

Some devices, most notably the teletype, have an independent keyboard and printer. The printer operates upon data from the computer only. In order to make it print a typed character, the computer must echo back the character. When bit 12 of a port's receive channel parameter is set (equal to 1), the port is operating in full duplex mode and the multiplexer will echo each character it receives on a bit by bit basis.

The ECHO command can be used to turn the feature on or off. The command determines if the user wants to turn echo on or off and sends to the IOP an ECN or an ECF. The format of the command is:

```
ECHO-ON for full duplex
ECHO-OFF for half duplex
```

DUMP User

The DUMP command allows a user on A000 to dump contents of system memory, IOP memory, ADT and swap tracks. The format of the command is:

```
DUMP-SYS [,STARTING WORD]
DUMP-IOP [,STARTING WORD]
DUMP-SWA [,PORT#][,STARTING WORD]
DUMP-ADT [,TRACK#]
```

The DUMP routine resides in two library overlays. The first overlay contains the routine to dump the system memory, the routine to dump the swap tracks, and utility subroutine. The second overlay, which overlays the first part of the first, contains the routine to dump the IOP memory and the ADTs. The second overlay uses some of the utility subroutines in the first overlay so it must end before the USO routine of the first overlay.

The operation of DUMP is as follows:

1. If the user is not on A000 then fail.
2. Check if DUMP is enabled. If not, fail.
3. Determine the command desired.
4. Put the starting word, port#, or track# if one was specified in VI. Put 0 in VI if not.
5. Branch to the appropriate routine, reading in the second overlay if necessary.

SYSTEM MEMORY DUMP

The contents of memory are listed. Each line consists of a five digit octal address followed by eight six digit octal numbers each representing a location in memory. The user swap area cannot be listed.

1. Print the report heading consisting of the systemid, the date and the time.

2. Print SYSTEM PROCESSOR MEMORY

3. Get the starting address (in V1).
   A. If the address is beyond the swap area then set the terminal address (VA) to the end of memory plus one. GO to step 4.
   B. If the address is in the swap area then set the starting address to the end of the swap area plus one. Print SWAP AREA CANNOT BE DUMPED BY USER. Set the terminal address (VA) to the end of memory plus one. Go to step 4.
   C. If the address is less than the beginning of the swap area then set the terminal address to the beginning of the swap area. Go to step 4.

4. Get a buffer from LOUT and build a line in it. During this process V4 is set to the first location past the last address dumped.

5. If the line is null then the starting address must have equaled the terminal address (V1=VA).
   A. If the terminal address points to the end of memory plus one then terminate.
   B. If the terminal address points to the beginning of the swap track then go to 3b. This will print a message and restart the listing after the swap area.

6. Put (V4) the first location past last one on the line into ?TEMP.

7. Set up LOUT to move the starting address (V1) into ?TEMP if the print is rejected.

8. Call LOUT to output the line and suspend.

9. Restore the starting address (V10 from ?TEMP.

10. Go to step 3.

DUMP_USER_SWAP_TRACK

The location from USE to ?PROG is listed for a port.

1. Get the port # to start (V2). Terminate immediately if not (0 <= port number <= 32).

2. Get the starting address. If it is less than the beginning of the swap area then set the starting address (13) to USE.

3. Determine if the port has ever been used.

4. If port has been used then go to step 6.

5. Bump to the next port and set V3 to USE. Terminate if no more ports. Otherwise go to 3.

6. Print SWAP AREA--PORT XX.

7. Use V3 the starting address of the next line, to determine the disc block address of the location to be dumped.

8. If the starting address is greater the ?PROG then go to step 5.

9. Read in two disc blocks in case the locations to be dumped span a disc block.

10. Get a buffer from LOUT and build a line of the dump in it.

11. Output and suspend.

12. Go to step 7 if the print was rejected.

13. If the print succeeded then bump V3 until it is divisable by 10B.

14. Go to step 7.

DUMP_IOP_MEMORY (resident on the overlay)

The user may optionally specify a starting address. The routine has no way of determining the size of the IOP memory so the dump is set to terminate at location 77777B. This means that for IOPs with less than 32K redundant information is dumped.

1. Print the report heading consisting of the systemid, the date and the time.

2. Put starting address in V3.

3. Set maximum address to the end of memory plus one.

4. Ask the IOP for eight words by sending a SCI then the address (in V3).

5. Transfer the word into memory at LIBD.

6. If the starting address is equal to the maximum then terminate.

7. Get a buffer from LOUT to build a line.

8. Print the line and suspend.

9. If the write succeeded then bump the starting address by up to 8 to make it divisable by 10B.

10. Go to step 4.

DUMP_ADI (resident on the overlay)

The system available disc tracks are dumped five entries per line, a track at a time. The user may optionally specify a starting track number.

1. Check if the track #(V1) specified is legal. Fail if not.

2. Print report heading consisting of the systemid the date and the time.

3. Initialize the last seen disc address (V2, V3) to zeroes.

4. If the track number is eight then terminate.

5. Calculate the track length. If null then bump the track number (V1) to the next track and go to step 4.

6. Print the message ADT-TRACK #n.

7. Calculate the track length again because it may have changed. If zero, go to step 4.

8. Read in the ADT track into the user swap area at LIBUS.

9. Save a pointer to the entry to process in V5.

10. Get the last seen disc address in V3 and V3+1.

11. Search the ADT track for the first entry with a disc address greater than the last seen disc address. Set V5 to point to the entry if found.

12. If the search failed, go to step 16.

13. Get a buffer from LOUT to put the output in.

14. Process the entry and put into the buffer.

15. If this was the fifth entry on the line go to step 17 to print the line.

16. If there are no more entries (the entry just processed was the last one on the track) then go to step to print the line if not null. If the line is null then bump V1 to the next track and go to step 4.

17. Prepare the print by
    A. Put the track number, in V1, into V4
    B. Set the last seen disc address, now pointed to by V5 into V5 and V6
    C. Put V4, V5 and V6 into ?TEMP to specify where to restart the list after the write.
    D. Set LOUT to move V1, V2 and V3 into ?TEMP if the write fails. This will cause the line to be rebuilt on return from LOUT.

18. Call LOUT to output the line and suspend. On return LOUT will transfer ?TEMP to V, V2 and V3.

19. Go to step 7.

EXECUTE

The EXECUTE command allows the user to get and run a program.

The operation of the command is as follows:

1. Decode name and save if PFA is required for the user to access this program.
2. Search the directory for the entry. If it is not found print NO SUCH PROGRAM and exit.
3. If the entry is a file then print ENTRY IS A FILE and exit.
4. Check the program entry to see if the owner has PFA. If not and it's required then print NO SUCH ENTRY and exit.
5. If the program owner is the user then go to step 6. Otherwise check if the program is private. Print NO SUCH ENTRY

if it is.

6. Check to see if there is enough room for the program. If it is too big print PROGRAM TOO LARGE and exit.
7. Update the last reference date of the program's directory entry and write the directory track out to disc.
8. Read in the user swap area to PBUFF.
9. Read in the program from disc. If the read fails print UNABLE TO RETRIEVE FROM LIBRARY and exit.
10. Set the protection bits in the ?NAME words. Set bit 15 of the first word if the program was locked, bit 15 of the second if private and bit 15 of the third if protected.
11. Set the program owner word (?PRID) in the user's TTY table. Set bit 15 if the owner has FCP and is a group librarian.
12. Call SEMIC to initialize compile variables before compiling (see the GET command for a more complete description).
13. Turn off the interrupt system and set the HELLO flag in the user's flag word and change his status to RUN.
14. Set the user up to run for two seconds and set the time flag to time this user.
15. Jump to the compile routine.

FILE Command

The FILE command serves two purposes. First, it is used to associate a file name (and create a directory entry) with either a specific or a class of non-shareable devices. Second, it is used to allocate space on the disc (and create a directory entry) for a disc ASCII file.

1. Begin building a directory entry by first extracting the proposed name of the entry.

2. Setting the user's ?CLOC word to -1 will cause suspension of the user if the command was called programmatically. This is done to keep users who are programmatically using the file command from tying up the system.

3. Extract the specified device designator and the proposed record size. Skip to step 6 for ASCII disc files before getting the third parameter.

4. Search the device table for the device. Demand that the device is available on the system and that the user has the capability to access it.

5. Insert the new directory entry in the proper directory track (calling supersave if the track is full) and update the DIREC, IDEC, and Equipment tables on disc. Exit routine.

6. For ASCII disc files, the next parameter is the file length

(in blocks). Extract it and the possible record size (set to 63 if not present) and validate these two parameters.

7. Check that there is sufficient account space and enough system disc space to accomodate the new file. Set disc address to appropriate spot on the disc and insert new directory entry. We don't set read/write access bits yet because we don't want user to have access before file is initialized.

8. Update the account's IDT entry and remove the necessary space from the ADT.

9. Using the user space, fill 400 blocks worth of space with EOF marks (user's have been known to play tricks to read IDT info from a badly created file).

10. If more than 400 blocks in the file, suspend with restart at disc initialization overlay (it lives in the MWA overlay). Otherwise, give the user read/write access by reading directory back into memory and then terminate.

FILES

The FILES routine is used by BASIC at run time to process FILES statements in a user's program. The function of the FILES routine is to translate the file names in the user's program into file control blocks for use during execution.

During operation of the FILES routine, a temporary buffer is used as a table to store intermediate data. Nine words of the buffer are used for each file. The operation is as follows:

1. Translate characters in FILES statements into the buffer table. FILES statements are pointed to by a four word table in the user swap area which is pointed to by DFILT. FILCT = -5+ # of FILES statements. There may be up to 4 such statements. Filenames are extended to six characters, if necessary, and those which are specified to be system library are marked by setting Bit 15 of their first word to 1. Those which are specified to be group library files are marked by setting bit 7 of their first word to 1. A "*" alone as a file name is a place holder. The buffer table for the entry is zeroed. Possible errors found in this step are:

   a. File name of 0 or > 6 characters
   b. More than 16 files requested

2. Perform directory search for each specified file. DIRWD points to the disc address of the directory track in core so that DLOOK doesn't have to read and write the directory for each file. Save the file's disc address, file length, and logical record size in its portion of the buffer table. The read-only bit is set if the file is a library file and the user is not the owner. An error occurs if the file is nonexistent or protected. Update the last reference date word in the directory entry for this file.

3. Validate access to and allocate file if type ASCII.

4. Test to make sure that there is sufficient room in the user area for the file control blocks and buffers.

5. Validate user's dynamic access to the file (use FUSS table).

6. Write out FUSS table with type of access granted.

7. Build file control blocks (see section on compilation for format).

GET Command

The GET command allows a user to get a program from the library.

COMMAND FORMAT	COMMENTS
GET-NAME	FULL ACCESS TO USER'S OWN ACCOUNT
GET-$NAME	SYSTEM LIBRARY PROGRAM REFERENCED. PROG MUST BE UNR OR PRO
GET-*NAME	GROUP LIBRARY PROGRAM REFERENCED. PROG MUST BE UNR OR PRO
GET-NAME.ID	REQUIRES OWNER HAVE PFA AND USER MUST HAVE STATIC ACCESS TO OWNER'S ACCOUNT

1. Decode name and note if program will need PFA. Leave ID and name in LTEMP 0-3.

2. Search for entry. Print NO SUCH PROGRAM if not found.

3. If file then print ENTRY IS A FILE and exit.

4. If the entry does not have PFA but is required print NO SUCH PROGRAM and exit.

5. If the user owns program goto 8.

6. If the program is private print NO SUCH PROGRAM and exit.

7. If the program is locked print EXECUTE ONLY and exit.

8. Check if the program will fit into the user area. This is necessary in case a program that was saved under an old version of the system can no longer fit with the current version. If it is too big print PROGRAM TOO LARGE and exit.

9. Update the last reference date of the directory entry and write back to disc.

10. Read in swappable base page.

11. Read in the program beginning at the starting address specified in the directory entry. If the disc transfer fails then print UNABLE TO RETRIEVE FROM LIBRARY and exit.

12. Move the name of the program from LTEMP+1 to LTEMP+3 into the ?NAME words in the user's TTY table entry.

13. Set bit 15 of the first word of ?NAME if the program is locked, the second word if private and the third word if protected.

14. Put owner's ID, from LTEMP+0, into ?PRID in the user's TTY table.

15. Call SEMIC which will for uncompiled programs, clear QFLAG, SYMTB, reset PBPTR to the end of the new program, clear the compile bit in the user's TTY table, set main and set SPROG. For a compiled program SEMIC will set PBPTR, and FILTB to the end of the program -6. SEMIC then moves the first word of the six after PBPTR to SYMTB; the second, the file statement counter, to FILCT; the next 4 words, file statement pointers, into FLSTS+0 to FLSTS+3. Then the compile bit in the user's TTY table is set, MAIN is set and SPROG.

16. Block the clock and initialize common if any, from PBUFF to SPROG, with -1's.

17. Exit.

HELLO

The HELLO command is used to log a user on the system. Its operation is as follows:

1. If the current ID is 0, there is no user to log off, so go Step 2. Otherwise, tell the I/O processor (service routine NUC) that a new user called. This will force the user to be disconnected if he does not successfully log on.

2. Read the IDT. If there is no user to be logged off, go to Step 3. If user has *OUT=* file then remove it.

3. Decrement the number of active ports by one.

4. Find the old user's IDT entry and update his total time used. Add an entry to LOGGR to be printed on the system console. Set the user's ID word to 0.

5. Translate the new ID code and search for it in the IDT. If not found, print an error message and terminate. Compare the password typed to the correct one, and fail if they disagree.

6. Check if a terminal type was input. If not, assume terminal type #0 and go to Step 7. Otherwise check if terminal type is in the Range 0 through 8. If not print an error message.

7. Tell I/O processor (service routine STP) which terminal is connected to the port. Check that the time used to date is less than the time allowed.

8. Check if user has anytime left. Fail if none left.

9. Initialize CoFLG bit in user's tty table ?FLAG word to 0.

10 Add a LOGON entry to LOGGR and set the starting time into the user's TTY table. Also insert the ID code, clear name, scratch the program. Tell the I/O processor of the successful Logon (service routine ULO).

11. Increment the number of active ports by one.

12. Print banner message if present. Ignore if it can't be read read from disc.

13. Search the directory for a program named HELLO in the library of user A000. If not found, or if it is a file, or if it will not fit in core, or if it cannot be read from disc, print READY and terminate.

14. Put name into ?NAME+0 to ?NAME+2 in user's TTY table. Set bit 15 of first word locked, second if private and third if protected.

15. Set ?PRID with program owner's ID. If owner has FCB then also set bit 15.

16. Read in the fixed user area and append HELLO. Call SEMIC, which sets program pointers as in SAVE. Change the user's status to RUN, set the chain flag, issue a LF, and transfer to BASIC.

PURGE Command

The PURGE command deletes a directory entry (program, file, disc ASCII file, or non-shareable device) and returns any space occupied by that entity to the system's ADT. The overlay can be invoked by a command, a BASIC statement, or some of its subroutines may be invoked directly from other overlays.

1. If called programmatically, skip this step as BASIC has already saved the entry name and owner's id in LTEMP (0:3). Otherwise, extract entry name and save in LTEMPs.

2. Force the user who programmatically invokes the command to be suspended at command completion by setting ?CLOC to -1. This keeps users doing a lot of sequential PURGEs from tying up the system.

3. If called programmatically and if it passes necessary criteria, allow a group master to purge a group member's locked BASIC formatted file.

4. Find specified directory entry. For files ensure that the entry is not in use.

5. Delete entry from directory track. Terminate routine if no space is to be returned to system.

6. Return space to ADT and update owner's IDT to reflect lost space. Update memory resident tables on disc (DIREC, IDEC and Equipment Table). Exit routine.

LENGTH Command

The LENGTH command prints the length of the user's program, in both words and disc records, the total disc space currently used in the account and the total disc space permitted. This command may be called programmtically by using the SYSTEM statement. The operation is as follows.

1. Read in program so we can access swappable base page variables.

2. The length of an uncompiled program in words is ?PROG-SPROG. ?PROG points to the last word if the program +1 and SPROG the beginning. For a compiled program the length in words is SYMTB-SPROG, SYMTB is the start of the symbol table which just follows the program. The length in records is the integer part of the sum of the length in words plus 255 divided by 256.

3. Convert the length in words and records into ASCII and put into output buffer.

4. Read in the IDT entry of the user to get the total space used (records) and the total allowed. Convert these to ASCII and also put into the output buffer.

5. If LENGTH was not called programmatically then go to step 6. Otherwise move the output buffer to FILBF so the SYSTEM overlay may be brought in. Exit to the language processor (SYSCU).

6. Print output buffer and terminate.

LIBRARY

The LIBRARY routine prints a list of all programs and files in the public library. Its operation is identical to that of CATALOG except that AUUU is used for directory searches instead of the user's id, and protected programs are not listed.

GROUP

The GROUP routine prints a list of all programs and files in the user's group library (the library of the idcode ending in 0U which has the same letter and first number as the user). Its operation is identical to that of LIBRARY except that the group librarian's idcode is used for directory searches instead of

A000.

LIST/PUNCH

The LIST and PUNCH commands are called by the user to LIST or PUNCH the program currently in core on the user's terminal. The In-core executor sets LTU=0 (LIST) or LTU=-1 (PUNCH).

1. If the program is null, terminate after printing a line-feed.

2. Evaluate the beginning and ending line numbers and determine whether the pagination option was specified.

3. A dummy file control block is established for the output device. If *OUT=* was specified, the block is initialized from the parameter ?OREC in the user teletype table. Otherwise it is set up for the user terminal. In either case, a buffer is allocated and the SPACQ bit is set in the user teletype table to indicate that the buffer is allocated.

4. The paginator flag is cleared if the output device is ASCII. If the command is PUNCH, a check is made that the output device is either the user terminal or a paper tape punch.

5. Output the program name followed by an empty line. If in the PUNCH command output feed frames for leader.

6. If at end of program, go to step 12. Otherwise get the line number and place in .LNUM. If this line is past the end line number, go to step 12. If a new page is required for the pagination option, output the required number of line feeds. Flag first position operator.

7. If at end of statement, print the current line and go to step 6. Otherwise go to step 8.

8. Extract next operator. If null go to step 9. If the first position operator flag is set then output the multicharacter operator (including the rest of the line if REM, FILES, or IMAGE) and go to step 7. If the operator is single quote go to step 11. If the operator is a single character, output the character and to to step 9. Otherwise output a multicharacter operator and go to step 9.

9. Extract next operand. If the constant flag is set, go to step 10. If null go to step 7. If program is compiled retrieve actual name from symbol table. If the operand is an extended string output the letter, digit (if present), and $ and go to step 7. If the operand is a user defined

function output FN followed by the letter and go to step 7. Otherwise output the letter followed by a digit (if present) or a $ (if a simple string variable). Go to step 7.

10. If the operand is a numerical constant, output the number and go to step 7. If a pre-defined function print the name and go to step 7. If parameter symbol print it and go to step 7. The remaining case is that of a program integer. If within a DIM or COM statement output the integer and and go to step 7. Otherwise output the integer statement reference. If at the end of the statement print the line and go to 6. If not at end of statement but within a USING go to step 8. Otherwise we're in a computed GOTO so emit a comma and keep printing program integers until statement end is found then go to step 6.

11. List a string constant, prefixed and suffixed by single quotes. Go to step 7.

12. Output trailing feed frames if PUNCH, write EOF if *OUT=* to ASCII disc file, de-allocate buffer, reset the SPACQ bit, output a line-feed and return.

LOAD

The LOAD command allows the user to input a program from an ASCII file.

1. Clear carriage return flag of LCHAR.

2. Block the clock so MAIN can be cleared and the user area written out to disc.

3. Decode filename and record if PFA is required.

4. Search for the file. If not found print NO SUCH ENTRY and exit.

5. If PFA is required but file owner does not have the capability, print NO ACCESS ALLOWED and exit.

6. Call SECHK to check if user has static access to file owner's account. If not, then print NO ACCESS ALLOWED and exit.

7. Call AFILE to find if the file is ASCII, the file not busy and the user has the capability to access it. If not, print an error message and exit. Set ?OREC, ?OUTP, and ?NREC with appropriate values if the checks succeed. Also update last reference and change date of the file and write out

directory track.

8. Read the user program back into core and set MAIN.

9. Initialize syntax variables and decompile program if compiled.

10. Check if there is enough space left in the user area for a device buffer (=record length), and a FCB (11 words). Print OUT OF STORAGE and exit if not.

11. Turn off the interrupt system and check the user's status word. If %ABRT then exit to scheduler (SCHL).

12. Set the unable to abort bit and set the tape mode bit in the user's ?FLAG word.

13. Reenable interrupt system and unblock the clock.

14. Move the user program into higher core so the FCB and the device buffer may be placed at PBUFF.

15. Set INFIL point to the FCB and call STFCB to fill the FCB.

16. Get a record from LINRD. If EOF detected go to step 21. If the record does not start with a digit then suspend by setting ?PLEV to 4 and restart to step 17, requeue and enter the scheduler at SCHL.

17. If the abort attempted bit is set then go to step 21.

18. Jump to syntax to process the record.

19. On return, suspend if the ascii file was not a disc file. Restart at step 16.

20. If the disc records in core have been exhausted then suspend restarting at step 16. Otherwise get a record and go to step 16.

The following steps describes the clean up necessary before exiting:

21. Set INFIL to zero and remove the FCB and device record buffer from the user swap area.

22. Adjust PBPTR.

23. Clear the unable to abort and the abort attempted bits in the users flag word.

24. Kill the ascii input device.

25. Clear the tape mode bit.

26. If the tape error bit was set then go to step 27 to schedule the tape mode cleanup routine. Otherwise terminate.

The following step schedules the tape mode clean-up routine.

27. Block the clock.

28. Set the restart address to the beginning of the tape mode clean-up routine.

29. Set the the user priority to 4.

30. Set the status to tape mode cleanup unless the user status is already %DISC.

40. Dequeue the user and requeue at the bottom of the queue.

50. Enter scheduler at SCH1.

MESSAGE

The MESSAGE command is used to send a message from a user console to the system console. The message is placed in a queue* and is ultimately output to the system console by the scheduler. The routine operates as follows:

1. Check if message queue is full. If so, fail.

2. Put a CR-LF and the ASCII representation of the user's port number in the message buffer.

3. Transfer the message from the user's teletype buffer in the I/O processor to the message buffer one character at a time (ignoring characters <40(8) and >177(8) except bell 7(8)). Stop when CR encountered or after 72 characters transferred.

4. Increment message counter and set pointer to next message buffer.

5. Terminate.

*This is a circular queue made up of five 39 word buffers. Relevant variable include, MSGCT, a positive count of full buffers, MSGP3, which contain the address of the next available buffer, and MSGP1, a pointer to the buffer being processed.

NAME

The NAME command allows a user to assign a name to his program. The program name is placed in his teletype table. The operation is as follows.

1. Get a character. If it is a carriage return then change it into a blank.

2. If the character is not a blank, letter or digit then zero name area, while preserving the protection bits, and take the illegal format exit (ILFER).

3. Save the character in the NAME words of the user's TTY table, and keep protection bits.

4. If less than 6 characters go to step 1.

5. If first word is blank then zero the word except for the protection bits that are set and get another input character. If the next character is a carriage return then exit. Otherwise output the warning message "ONLY 6 CHARACTERS ACCEPTED" before exiting.

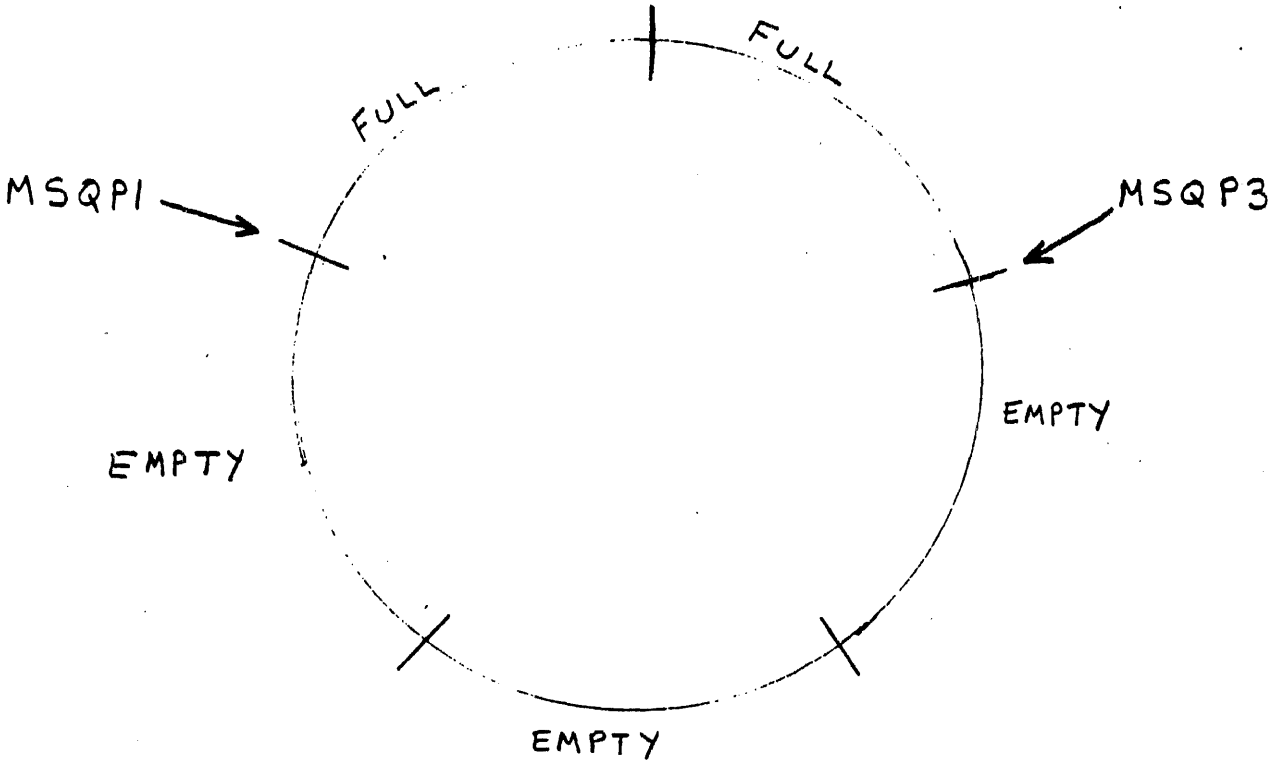
CREATE Command

The CREATE command allocates space on the disc for a BASIC formatted file, gives it a name, and creates a directory entry to point to it. The space on the disc is initialized with EOF marks to keep clever users from being able to read any left over tables that previously occupied memory. The overlay can be invoked by either a system command or by an executing BASIC program. In the latter case BASIC sets up the parameters in a directory template (LTEMP0..11) and sets SYSFL.

1. If called programmatically, continue at step 2. Otherwise, extract proposed file name, file length, and record length (optional parameter). Save in directory template (LTEMP0..11).

2. Set ?CLOC word in TTY table to -1 so, if called programmatically, user will be suspended at end of routine. This keeps users from inordinately tying up the system.

3. If name is qualified, demand that 1) we were called programmatically, 2) user is a group master, 3) user has FCP capability, and 4) potential owner id has PFA.


FULL
FULL
MSQP1
MSQP3
EMPTY
EMPTY
EMPTY

4. Verify sufficient space exists in account and system. Set disc address of space to be removed from ADT.

5. Insert new directory entry in the proper directory track (calling supersave if necessary). Update owner's IDT entry and remove space from ADT. Note that this order is important in the event of either being unable to insert directory entry or unable to update IDT entry. In these cases, no space has been removed from the ADT yet.

6. Initialize file with EOF marks. If more than 40 records, suspend user with restart at disc file initialization (MWA overlay resident). When done, read back directory and give user read/write access.

7. Note that DIREC, IDEC, and Equipment Table are updated on the disc as necessary.

UNRESTRICT

The UNRESTRICT command allows a user to unrestrict a program or file. An unrestricted program allows any user having static access to the owner's account to GET, EXE or CHAIN to it. An unrestricted file allows any user having static access to the owner's account to read/write on it (subject to dynamic restrictions). The operation is as follows.

1. Check for legal name. Output an error message if not. Search directory for program/file. Output an error message if not found.

3. Clear bits 0:3 in the status word and set bit 0, the unrestrict bit. in the directory entry.

4. Write the directory back out on disc and terminate.

PROTECT

The PROTECT command allows a user to protect a program or file. A protected file allows any user having static access to the file to read it. A protected program allows any user with static access to the owner's account to EXECUTE, CHAIN to or GET the program but not list, punch or save it. The operation of PROTECT is the same as UNRESTRICT but

3. Clear bits 0-3 in the status word and set bit 1, the protect bit, in the directory entry.

LOCK

The LOCK command allows a user to lock a file or program. Any user having static access to the owner's library may EXECUTE or CHAIN to the beginning of the program. A locked program from the owner's program owner's account may CHAIN to other than the beginning of the locked program. Any user having static access of the owner's library has read/write capability while executing a locked program from the owner's account. The operation of the LOCK command is the same as the UNRESTRICT except:

3. Clear bits 0-3 in the status word and set bit 2, the LOCK bit, in the directory entry.

PRIVATE

The PRIVATE command allows a user to place a file or program in the private state. A file or program in the private state may not be accesed by anyone but the owner. The operation of PRIVATE is the same of UNRESTRICT except.

3. Clear bits 0-3 of the STATUS word and set bit 3 the private bit, in the directory entry.

SWA

The SWA command allows a user to place a file in the SWA state. A file in the SWA state restricts write access to the file to at most one user.

The operation is as follows:

1. Get and check for legal name specified. If not, output an error message and exit.

2. Search the directory for the directory entry. Output an error message if not found or if a program and exit.

3. Clear MWA bit in the entry if set. Write the diectory back out to disc and terminate.

MWA_COMMAND

The MWA command allows a user to place a file in the multiple-write access state. This permits simultaneous access to the file (subject to dynamic restrictions) by any users who has static access to the owner's library. Only users with the MWA capability may execute the MWA command. The operation is as follows:

1. Check and save the parameter list of the command. If bad output a message and exit.

2. Check if the user has MWA. If not, output a message and exit.

3. Search directory for the entry. Output an error message if not found, a program, or an ascii file; exit.

4. Set MWA bit in entry, write back out to disc and terminate.

Programmatic_MWA

A file may be made MWA programtically by the use of the SYSTEM statement to execute the MWA command.

1. Check and save the parameter list of the command. If bad set return code to fail and exit.

2. Note if the file to be MWAed has been qualified (filename.ID) or not.

3. Check if the file owner has MWA. If not, set return code to fail and exit.

4. Search for the file. If the entry turns out to be an ASCII file or a program then set the return code to fail and exit.

5. Set the MWA bit in the files directory entry.

6. If filename was not qualified then go to 7. Otherwise check if file is locked, file owner has PFA, the file owner is in the group of the program running and the program owner's id is a group master. Print an error message and terminate if any of the checks failed.

7. Write out diectory track and exit.

PAUSE

The pause statement allows an executing program to suspend for a period of time from 1 to 32767 seconds. The user is suspended with the %PAUS status until the time expires or the console awakens him with the AWAKE command.

1. If user typed in PAUSE as a command then terminate.

2. Negate the time specified by the user and put into ?RTIME.

3. Clear the user's unable to abort bit so the user may break unless the he had disabled it (PBFLG=1).

4. Set the user's status to pause and suspend with the restart address set to PAURS-1. When restarted after the time has elapsed or awakened by the console, the program enters, with (B)=0 indicating success, the system statement clean up routine.

RENUMBER

The function of RENUMBER is to assign a new set of sequence numbers to a user program. The user may specify the sequence number of the first statement and the increment between statements. If unspecified, these are set to 10. He may also specify the first statment to be renumbered and the last statement to be renumbered. If unspecified these are set to the first statement of the program and the last statement of the program respectively.

There are actually two sets of numbers that must be modified. One set is the sequence numbers themselves, each of which occupies the first word of its statement. The other is the set of references, which are labels in CONVERT, GO TO, GOSUB, RESTORE, PRINT USING, MAT PRINT USING, and IF statements. Each of these also occupies one word. For programs in compiled mode, they are pointers to the statement they reference; in decompiled mode they are the actual statement number.

The primary technique used is to change all the references to absolute pointers (if in decompiled mode), then to change all the sequence numbers, and then (if in decompiled mode) to change the references to the new statement numbers. References to nonexistent labels are left unchanged.

Because the process of changing all the references to absolute pointers can become quite time consuming (due to the search that must be performed for each reference), a table is built in advance dividing the program into 32 parts, each containing the same number of statements. For large programs with many references, this effectively cuts the time down by a factor of 32.

The subroutine RENSK is used to scan for references. It maintains two pointers, RENP and RENQ. Whenever it is called, it moves RENP to the next reference, and sets RENQ to point at the statement following the one that RENP is pointing at. It takes advantage of the fact that any references within a statement are always the last word or words of the statement, except in the case of PRINT USING and MAT PRINT USING, in which case it takes advantage of the fact that there is only one statement number reference. All PRINT and MAT PRINT statements are scanned until either a USING is found or it is determined that no USING is in the statement. If a USING is found, a check is made for the statement reference. Before calling RENSK for the first time, RENQ is set to point at the first statement to be renumbered. RENP is set RENQ-1.

The operation of RENUMBER is as follows:

1. If null program, terminate immediately. Otherwise, read in user program.

2. Perform statement length check on all statements in program.

3. Translate and check parameters M and N. Set to default values if not present.

4. Translate parameters P and Q. Set RENBA = first statement to be renumbered, RENLA to last statement to be renumbered.

5. Set RENLA to point to the last sequence # $\leq$ RENLA. Also set RENBA to point to the first sequence # $\geq$ RENBA.

6. Insure that there will be no sequence number overlap at either end of the portion of the program to be renumbered and that the new sequence numbers will not exceed 9999.

7. If program is in compiled mode, go to Step 10. Otherwise, set up a table in ERSEC which divides the program into 32 parts. The result is that for each I from 0 to 31:

ERSEC[I] = sequence number of first statement in part I,
ERSEC[I+32] = absolute address of that statement

If there are 32K + L statements ($0 < L \leq 31$) in the program, ERSEC [I] is the sequence number of statement:

$(K + 1)*I + 1$, if $I < L$
$K*I + L + 1$, if $I \geq L$, $K \geq 0$
Undefined if $I \geq L$, $K=0$

Set RENQ=SPROG, RENP=RENQ-1. (SPROG points to the first statement).

8. Call RENSK to find the next statement reference. If there are none left, go to step 10. Find the largest I for which ERSEC [I] $\leq$ (RENP). If there is none, the statement referenced does not exist, so go to step 9. Otherwise, test all statements from (ERSEC [I + 32] to either (ERSEC [I + 33]) or PBPTR, depending upon whether $I < 31$ or $I = 31$. If found, set (RENP) to the location of the statement referred to, and repeat this step. Otherwise, go to step 9.

9. Set (RENP) = (RENP) + 100000(8) and go back to step 7.

10. Change the sequence numbers for all statements to be changed, according to the RENM, RENN, RENBA and RENLA values. If compiled mode, terminate. Otherwise, set RENQ = SPROG and RENP=RENQ-1.

11. Call RENSK to find the next statement reference. If none left, terminate. If (RENP) < 0, the reference was undefined, so set (RENP) = (RENP)-100000(8), and repeat this step.

REPORT User Console

The REPORT command allows a user on AUUO to list user IDT information. Each user ID is listed along with the time, disc used, its capabilities, and its device designators. The routine LOUT is used for listing to the terminal. This routine will suspend the user after each line.

The operation of the command is as follows:

1. Check if the user is on A000. If not, print PRIVILEGED COMMAND and exit.

2. Check if the user specified an ID to start with. If so, save it in the user's TTY table ?RTIM word otherwise save a null.

3. Print the first two lines of the heading.

4. Get the ID code from ?RTIM and search the ID tracks for it. If the ID code is greater than any on the system then terminate. If the ID code is not there, then choose the next one.

5. Build the line in FILBF, for the ID consisting of the idcode its time used, its disc space used, its capabilities and its device designators.

6. Check if the ID is the last one on the system. If so, set the return from a successful write to LEND otherwise set to step 4.

7. Try to output the line. If unsuccessful then go to 4. If successful exit to LLEND if there are no more IDs, otherwise bump the ID in ?RTIM then go to step 4.

SAVE

The SAVE routine is called by the user to save a program (uncompiled from SPROG to PBPTR) in library.

1. Test for the existence of a program name and a non-null program. Print NO PROGRAM NAME or NO PROGRAM if not and exit.

2. Read in user program from swap area.

3. If program is compiled then decompile it. Allocate common if any.

4. Write the program out to the swap area.

5. Allocated to CSAVE for numbering compatibility. See CSAVE.

6. Compute the space required to save the program in negative words and positive records. Put the negative word length into the directory entry template (at LTEMP to LTEMP+11) length word. Save the record length for the ADT search

later.

7. Get user IDT entry. Check the user capabilities and set the PFA and FCP bits in the directory entry template accordingly. Also the set private bit.

8. If there is not enough space left in the user's account then print LIBRARY SPACE FULL and exit.

9. Search the ADT for the space. In the process any ADTs encountered that could not be read because of disc errors are deleted from the system (length = 0 and the disc address = 0). If the search fails print SYSTEM OVERLOAD and exit.

10. Save the ADT disc address, the position on track of the entry, and the space remaining after the program space has been allocated. Put disc address of the space into the directory entry template disc address word.

11. Read the program back in from the swap area.

12. For CSAVE only.

13. Write the program out (from SPROG to PbPTR) to disc. If unsuccessful because of disc error the go to step 20.

14. Move name from TTY table (?NAME) to the directory entry template (LTEMP 1:3), stripping protection bits if present.

15. See CSAVE.

16. Search for a duplicate entry. If found, print DUPLICATE ENTRY and exit.

17 If the directory track that the new entry is to go on is full then call the SUPERSAVE routine to redistribute the directory. SUPERSAVE will perform step 18 and proceed to step 19. If SUPERSAVE fails then print SYSTEM OVERLOAD and exit.

18. Fill in the start of program pointer, the last reference date and the last change date in the directory entry template and insert into the directory.

19. Read the user's IDT entry and update the disc space used.

20. If the ADT track length is zero or the ADT could not be read from disc then go to step 21. Update the ADT and write it out again.

21. If the program was successfully written to the

library then update the system tables (EQT, DIREC, IDEC) on disc.

22. If the writer of the program to the user's library had failed then print UNSUCCESSFUL, TRY AGAIN.

23. Terminate.

SUPERSAVE

The SUPERSAVE routine is called by the SAVE, CSAVE, COPY, BESTOW, FILE and CREATE routines when they want to make a directory entry on a track that is already full. SUPERSAVE assumes that the following words are set properly:

(LTEMP:LTEMP+3) = first 4 words of entry.
(LTEMP+4) = pointer to DIREC entry for appropriate directory track
(LTEMP+5) = core address of entry which is to precede the new entry
(LTEMP+6:LTEMP+7) = disc address of entry
(LTEMP+8) = length of entry
(LTEMP+10) = start of program pointer/record size

Note that (LTEMP+4) and (LTEMP+5) are set correctly by DLOOK.

SUPERSAVE attempts to redistribute the directory tracks so that they will be as equal in length as possible. This will generally prevent it from being called very frequently. The operation is as follows:

1. Scan through DIREC and determine the total length of all directory tracks, and add 12 for the new entry. If all directory tracks are full, exit through failure location.

2. Divide total directory length by number of available disc tracks to determine their new individual lengths. Insert these in the table at (DEFNN+1:DEFNN+60) as negative.

3. Now squeeze all the directory entries to the last most of the available tracks. This is done by reading the tracks in reverse order and writing 1024 words on each track until we run out of directory entries. The following variables are used in this section:

SUPK1 points to the DIREC entry for track being read (initially DIREL)
SUPL1 points to the DIREC entry for track being written (initially DIREL)

SUPK2 = -# of words in core

4. If (# of words on track SUPK1-SUPK2) > 8184, go to 5. Otherwise update SUPK2 and read this track to the core buffer at location LULEN + SUPK2 (SUPK2 being negative). If SUPK2 # -8184, go to 8. Otherwise set the length into the SUPL1 DIRECT entry and write the 8184 word buffer to track SUPL1. Set SUPK2=0 and go to 7.

5. Set SUPES = (#of words on track SUPK1-SPUK2-8184)/256. That is the number of extra blocks on the track to be read. Set SUPEX = SUPES * 256. This is the number of extra words. Then read the last (# of words on track SUPK1-SUPEX) words from track SUPK1 to location (LULEN + SUPK2 - # of words on track SUPK1 + SUPEX) and update SUPK2.

6. Write 8184 words to track SUPL1 from location LULEN-8184 and set the length in the SUPL1 DIREC entry. Move the leftover -8184-SUPK2 words to the end of the buffer, resetting SUPK2. Then read the SUPEX words left on track SUPK1 to location LULEN+SUPK2-SUPEX. Set SUPK2=SUPK2-SUPEX.

7. Update SUPL1 to point to the next track to write.

8. Update SUPK1 to point to the next track to read. If we've finished all tracks, to to 9. Otherwise go to 4.

9. If SUPK2 = 0, go to 10. Otherwise write out the -SUPK2 words to track SUPL1 and set the length in the SUPL1 DIREC entry.

10. Zero out the lengths in the DIREC table of all those tracks that no longer have anything written on them.

11. Now redistribute the directory tracks. The basic idea of the algorithm is to fill the swap area with as much of the directory information as we can, reading from the beginning, and then to write out as much as we can, always making sure that when writing we don't overlay any portion that hasn't been read yet. The following variables are used:

SUPK1 points to the DIREC entry for track being read
(initially DIREC0)
SUPL1 points to the DIREC entry for track being written
(initally DIREC0)
SUPK2 = # of words read so far from track SUPK1
(initially 0)
SUPL2 = # of words written so far on track SUPL1
(initially 0)
SUP = # of words in core
(initially 0)
SUPP points to DEFNN entry, telling how many are to be

written on SUPL1.
SUPTG = 1 if we have already inserted the new entry.

12. If SUPL2=-(SUPP), we have completely written track SUPL1 so check for SUPL1 = DIREL. If it is, we've written all the tracks, so go to step 18. Otherwise, advance SUPL1 to the next directory track advance SUPP, set SUPL2 = 0, and repeat this step. If SUPL2=(SUPP) go to step 13.

13. If SUP ≥ 10232, we have read as much as we can, so go to step 15. If SUPK1 = DIREU, there is nothing left to read, so go to step 15. If SUPK2 = # of words on track SUPK1, we've read the entire track, so advance SUPK1 to the next track, set SUPK2 = 0, and repeat this step. Otherwise, compute the number of words we can read. If there is no room to read the balance of the track, we will, otherwise we will read the maximum number of full blocks possible. If this is zero, go to step 15. If it is not zero, read from block SUPK2/256 into core location LIBUS +SUP. Add the number of words read to SUP and to SUPK2.

14. If SUPTG = 0, determine if we can insert the new entry. To do this we first determine where the even entry boundary occurs in the core buffer since we may have read only part of an entry (12 does not divide 256 evenly). If the last entry in the buffer is greater than the entry we are inserting, the entry goes on this track. If this is not the case, go back to step 13. Otherwise, set SUPTG to 1, make a 12-word hole, insert the new entry, set SUP = SUP + 12, and go back to step 13.

15. WRITE_SECTOR. Set SUPS = 0. This is the number of words written.

16. Compute number of words we can write on track SUPL1. First set A = - number of words left to write on the track. If SUPL1 = SUPK1, we haven't finishing reading everything from track SUPL1, so if SUPL2 - A > SUPK2 change A to SUPL2-SUPK2, which is the number of words we can write without destroying any unread directory information. If SUP-SUPS<-A, we don't have as much in core as we are capable of writing so set A = - (((SUP-SUP) divide 256) x 256), an exact number of blocks.

17. If A = 0, we can't write anything, so if SUPS=0 slide the remain SUP-SUPS words in core up to location LIBUS, set SUPS = 0 and SUP = SUP-SUPS. Then go back to step 12.

If A#0, write -A words to block SUPL2 divide 256 of track SUPL1. If SUPL2 = 0, set the first 4 words of the SUPL1

DIREC entry to the first 4 words written. Set SUPL2 to SUPL2-A, SUPS to SUPS-A, and go back to step 16.

18. Set the new directory lengths into DIREC and go back to the calling program.

TAPE_MODE_CLEAN_UP

This overlay prints errors embedded in the program after the user has entered a program under tape mode of after a LOAD. The embedded errors are 3 word psuedo statements: statement number, length (=3) and error number.

This routine is called in place of any command typed if the tape error flag in the user's flag word is set. It is also called directly, if the error statement count (ERRCT) is non-zero, at the end of LOAD.

The operation of this routine is as follows:

1. Read in the user program from the disc.

2. If on entry the error statement count (ERRCT) is zero then go to 13. This is the case if the user aborted the print-out of the errors.

3. Strip out embedded error statements by scanning the program for error statements and moving each one found to the end of the program as two word entries (the length words are deleted). Decrement the error statement count by one for each one moved. Stop the scan when the count reaches zero.

4. Put into ?TEMP a pointer to the first two-word error entry to print.

5. Write out to disc the program and errors to update the disc copy of the user's swap area.

6. Get the statement number of the error.

7. Read in the appropriate error message block into the swap area starting at LIBUS.

8. Move the desired error message to LIBUS and append "IN LINE" and the line number.

9. Print the error message. The reader should note that in the print routine (LOUT) the user has output wait status for a brief period of time. This allows the user to break this, an otherwise unbreakable routine.

10. If the print fails (reject return from LOUT), read in the program, get the pointer to the error to print from ?TEMP and go to 6.

11. Read in the program again.

12. Bump the pointer (in ?TEMP) to the next error entry to print. If there are no more error entries then go to 13. Otherwise go to 7.

13. Set main; set the user's status to syntax; reset PBPTR to the end of the program; clear the tape error flag and exit to RERRS+33.*

*RERRS+33 prints the message LAST INPUT IGNORED, RETYPE IT and sets the user's status to idle.

TIME

The TIME command prints the user's ID, port number, time used during the session, the total time used to date and the total time permitted. The TIME command is callable from a program from a program via the SYSTEM statement.

The operation of TIME is as follows:

1. Get user's ID and port number and compute time used in the current session from the TTY table. Put these values into output buffer.

2. Read in the IDT to get total time used (excluding the current session) and the total time permitted.

3. Compute the total time used and put it into the output buffer along with the total time permitted.

4. Print it on the terminal or exit to SYSCU with the output buffer if it was a programmatic call.

ANNOUNCE

The ANNOUNCE is used by the system operator to send a message to a port or all the ports.

The operation of the command is as follows:

1. Determine if the message is to go to one port or all ports. Fail if the "ALL" was not typed or the port number is <0 or >31.

2. Save the number of ports and the first port to output the message to.

3. Put a CR, LF, LF into FILBF. Append the message and another CR, LF, LF.

4. Block the clock.

5. Get the status of the report to output. If user is to run a type III program and execution had not yet begun (?RSTR=LIBRA) then we must abort the program because the buffer which now contains the parameter list for the program will be used to hold the message. First clear the port's status, dequeue the port and clear the ports OUT=P bit if set and then send "STOP" to the terminal to abort the executing of the command.

6. If announce was to one port go to 7. Otherwise loop in step 5 until all ports are checked.

7. Send the message to the port(s).

8. Unblock the clock.

9. Check the status of the port(s) that had the message output to it. If a port's status is input wait then send an input wait command (IWT) to the IOP for that port.

ASSIGN Command

The ASSIGN command allows the system operator to assign a non-shareable device for exclusive access to all users, one user, no users, or to the RJE facility. Word 3 of the device's device table entry contains this information: -1, no user; 0, all users (default); 1, RJE only ; <idcode>, only that idcode.

1. Extract specific device designator.

2. Search device table for corresponding entry.

3. Determine whether assignment is to all, one user, none, or RJE.

4. Set word 3 of device table entry.

5. If assignment was to none, find if a user is currently in control. If so, set user's PACT bit and set status to abort (%ABRT). this will force the user off the device.

AWAKE Command

The AWAKE command is responsible for waking up a suspended user when the user has suspended for either an ATTENTION NEEDED (device error condition 3) or because of a programmatic execution of the PAUSE command.

1. Determine whether command is AWAKE-<port number> or AWAKE-<specific device designator>. If the latter, continue at 4.

2. Extract and validate port number; use port number to construct a pointer to user's part of TTY table.

3. If user is suspended (status = %PAUS) and it's because of a PAUSE command (sign of ?RTIM set) then set his PACT bit which will automatically wake him up. Exit routine.

4. Extract specific device designator and find device in device table. If the device is busy, the user is suspended with ?PAUS status, and if this device was the one that caused suspension, then we can awake him.

5. If device is not the mag tape we change his status to output wait (?OUTW) and send start timed retries (STR) to the IOP. The IOP will retry the device until it is ready at which point it will wake up the user.

6. If the device is the mag tape all we need to do is to set his PACT bit. When he is re-scheduled, he will automatically retry the operation.

BANNER Command

The BANNER command accepts a message entered by the system operator saves it on the disc in the last half of block 0 on the system disc. The format of the message is a word with a byte count followed by that many bytes of the message. If the message is null, the count word will be zero. This message, if any, is displayed to users when they log on (see HELLO command).

1. Read in block 0 of disc 0.

2. Initialize pointers to use byte filling routines.

3. Extract message from console. If non-null, terminate it with CR/LF.

4. Set the count word to the number of bytes in the message, write out block 0, and terminate.

BESTOW Command

The BESTOW command transfers ownership of library entries from one account to another. Optionally, we allow either all entries or a single entry. All entries may be transferred only when no users are logged on the system. Only protected or unrestricted entries which do not duplicate names in the target account are bestowed. The method is to scan for an entry in the directory, move it to the LTEMPS with a new account (adjusting MWA, PFA, and FCP bits if necessary), delete the old entry, and insert it where it now belongs. A running count of space deleted from source account and space added to target account is kept to update the IDT when the command completes.

1. Extract oldid, newid, and optional single entry.

2. Demand no users loged on if entire library to be bestowed.

3. Extract newid's capabilities, account space used, and account space allowed and save them away.

4. Find entry and determine feasibility of moving it.

5. Save and delete old entry.

6. Insert entry for newid.

7. Loop back to step 4 until no more entries left in oldid.

8. Update oldid and newid IDT entries for space used.

9. Output any notifications of things not done becuase of protection, no space, duplicate names, or no entries bestowed. Note that only one of these messages is output if more than one condition is met.

BREAK Command

The BREAK command overrides a user's disabled BREAK capability. There are two bits of interest in the flag (?FLAG) word of the user's TTY table: PBFLG and CBFLG. PBFLG is set when a user programmatically disables BREAK. The receive driver enables a BREAK attempt if PBFLG <> 1 or CBFLG <> 0. Thus, this command

can set CBFLG which will allow BREAK regardless of the state of PBFLG.

1. Extract port number, validate it, and ensure user is logged on.

2. Set CBFLG (which will override PBFLG, if set), clear ABTRY (so previous BREAK attempts will not cause trouble) and exit.

CHANGEID

The CHANGEID command allows the system operator to modify the parameters in an ID entry. The password, time allowed, disc space allowed and capabilities may be modified. The modification of the MWA, PFA and FCP may require ID's directory entries to also be modified.

1. Get the ID track the specified ID should be on. If no other parameters specified print ILLEGAL FORMAT and exit.

2. Search the ID track for the entry. If not found print NO SUCH NO ID and exit.

3. If a new password was specified then clear the password area in the ID entry and put the new one in its place.

4. Modify the disc and time limit if specified.

5. Save the initial states of MWA, PFA, and FCP.

6. Modify the ID's capability if any were specified.

7. Write out the updated ID entry out to disc.

8. If the ID had MWA but now doesn't or if the user's PFA or FCB capability had changed, the ID's directory entries must be modified, otherwise go to step 10.

9. Read in the ID's directory entries and modify the MWA, PFA and FCB bits in the status word appropriately.

10. Terminate.

COPY

The COPY command allows the system operator to copy a program/file from one id to another.

The operation of this command is as follows.

1. Get and save oldid and oldname. Put the ID and name into LTEMP 0:3 and also into a temporary buffer.

2. Search the directory for the oldname. If it does not exist print NO SUCH ENTRY and exit.

3. Move the old remaining part of the old directory entry. The new directory template.

4. If the entry is larger than 200 blocks and there are users logged on then print USERS LOGGED ON and exit. If the entry is LOCKED or PRIVATE then print PROPRIETARY ITEM NOT COPIES and exit.

5. If the entry is a file and the file has neither the read nor write capability then print PRIOPIETARY ITEM NOT COPIED. The file was in the pocess of being created.

6. Get the new ID and new name and put into LTEMP 0-3.

7. Copy the protection bits from the old directory entry to the name words of the directory template.

8. In the status word of the directory entry template clear the MWA bit if set if the new id does not have this capability. Set or clear the PFA and FCP bits according to the new ids capabilities.

9. Check if there is enough room in the newid's account for the new entry. If not print LIBRARY SPACE FULL and exit.

10. If the entry to be copied is non-shareable device go to step 12.

11. Search the ADTs for space for the new entry. Print SYSTEM OVERLOAD and exit if space could not be found for it. Save pointer to ADT entry and put the disc address into the new directory entry template.

12. Search the directory tracks for a duplicate entry. If there is one print DUPLICATE ENTRY and exit.

13. Copy if necessary the entry into the space found in 40 block chunks. Read the chunk first into the user area then

write to the new space.

14. Determine where in the directory the new directory entry should go. Call SUPERSAVE if necessary. SUPERSAVE will perform the next step and go to 16.

15. Set the last reference and last change date in the template, insert the entry into the directory, update the DIREC and write the directory track back to disc.

16. If new space was not allocated (entry copied was a non-shareable device) then go to 18.

17. Update the space used in the new id's IDT entry and update the ADT.

18. Update the system tables on disc and terminate.

HIBERNATE

The HIBERNATE command is identical to the SLEEP command except for the following additions/changes:

0. Set the current time into HDATE.
9. Set A = -1 (hibernate) and jump to the dump. (20000B of the loader).

DEVICE System Console

The DEVICE command lists information of all devices on the system. The device designator, select code (in octal), the record size assignment (all, a specific ID, "RJE" or "NONE"), and user (an ID and port or "RJE" or blank) are printed for each device.

The routine operates as follows:

1. Print first two lines of heading.

2. Get the number of device tables entries. If none then terminate.

3. Save pointer to the head of the device table.

4. Decode and put into the output buffer (T35BF) the device designator, the device select code the maximum record size, the assignment and the user (if the device is busy).

5. Print.

6. Bump the device table pointer. If there are no entries left then terminate, otherwise go to 4.

DIRECTORY System Console

The DIRECTORY command list on the system console, by ID, directory information. This information includes the ID, printed only with the first entry of an account, the entry name, the last reference date, the length, the device/address and the record length. The user may optionally specify a starting ID.

The operation of the command is as follows:

1. Determine if a starting ID had been specified. If so, save it in ID otherwise save 1 in ID.

2. Decrement the ID by one and save in T35BF+35.

3. Print the first heading line and suspend.

4. Move the second heading line into T35BF and print and suspend.

5. Retrieve the ID save in T35BF+35 put into LTEMP. Set LTEMP(1:3) TO 1777778. This will cause the list to start at the ID specified or the first one beyond if the ID does not exist.

6. Search the directory to find the first entry beyond the one sought. Save the ID of this entry in T35BF+35.

7. If the ID is -1 (psuedo entry) then terminate.

8. If the ID code was different from the previous (sought) ID then include the ID in the line we are building.

9. Continue building the line by adding the entry name, the last reference date, the length, the device/address and the record length.

10. Print and suspend.

11. Retrieve the ID code from T35BF+35 and the name of the entry from right byte of T35BF+4 to left byte of T35+7.

12. Put this into LTEMP(0:3). This is the new entry sought.

13. Goto 6.

DUMP Command (System Operator)

The DUMP command entered by the system operator either enables the use of A000's DUMP command or disables it. The routine sets the words DUMP1 and DUMP2 (which reside currently in the library subroutines area) to the time it will be when the DUMP capability will expire. The A000 DUMP command demands that the current time be less than this value.

1. Extract, convert to binary, and validate that time specified is between 0 and 60.

2. Add this time to the current time of day and save in DUMP2. Exit the routine.

DISCONNECT Command

The DISCONNECT command logs off the specified users from the system. It is done by merely changing the user status to disconnect (%DISC); this effectively simulates the situation when a user has physically disconnected from the system.

1. Determine whether command is DISCONNECT-ALL or DISCONNECT-(port number). Set LTEMP+1 to minus number of ports to be disconnected and LTEMP+3 to point to the first port's TTY table.

2. Looping on the counter (LTEMP+1), set each user's status to disconnect and set the user's PACT bit to force the scheduler to notice the disconnect.

KILLID

The KILLID command is used to remove ID codes from the system. KILLID first deletes the ID from the ID track. Then the directory is scanned backwards to delete entries belonging to the ID. A patch table is constructed at LIBUS + 8192 to aid the return of space to the ADT. The patch table holds three word entries: a two word disc address and the length. The routine RTADT which resides on disc is called to interogate the patch table and return the space to the ADT.

The operation of KILLID is as follows.

1. If users are logged on the system then print USERS LOGGED on and exit.

2. If the ID specified is A000 then print A000 NOT ALLOWED and exit.

3. Search the IDT for the ID. If not found print NO SUCH ID and exit.

4. Delete the IDT entry from the ID track and update the IDT and its IDEC entry.

5. Scan the directory backwards for an entry which belongs to the ID to be killed.

6. Put an entry in the patch table for the program or file.

7. If the patch table cannot hold another entry then go to 9.

8. Get the next entry on the directory track. If the next directory entry belongs to the ID to be killed then go to 6. If the next entry does not then go to 10.

9. Collapse the directory track, write it out, update the DIREC and return the space in the patch table to the ADT.

10. Go to the next directory track and go to 5.

11. Collapse the directory track, write it out, update the DIREC and return the space in the patch table to the ADT.

12. Update the disc copy of the DIREC, IDEC and EQT and terminate.

MLOCK Command

The MLOCK command allows the system operator to remove disc blocks from general usage either because they are bad or because the installation wishes to preserve some blocks for other purposes (one use could be a disc with both Access and RTE or DOS resident). MLOCK is one of those few commands where any additional documentation would be redundant. See listing for any additional details. Also see data structures for the format of the locked blocks table.

MUNLOCK Command

The MUNLOCK command frees disc blocks locked by the MLOCK command. See listing for additional details.

NEWID

The NEWID routine adds an entry to the IDT. The new IDT entry is built in NEWBF before inserting into the IDT.

1. Scan the parameter list for the ID and read in the ID track it should go on.

2. Get the password and put it into the ID entry template.

3. Get the disc space allowed and time allowed and put into template. If a CR follows then go to step 5.

4. Scan the parameter list for capabilities and device designators. Call the attribute search routine (SAT) to get the bit position for each parameter. Set these bits in the capability word of the template.

5. Put the new ID into the template.

6. If the ID track is full then call SUPERNEW to redistribute the ID tracks. Go to step 7.

7. Search the ID track for a duplicate entry. If not found then insert the entry, update the IDEC entry for the track and write the track back to disc.

8. Terminate.

SUPERNEW

This is a routine in the NEWID overlay that is used to evenly redistribute the IDs on the ID tracks. SUPERNEW operates very similarly to SUPERSAVE except that it does not build a table to save the new length of each track or insert the new ID entry into the IDT.

The routine first calculates the size each track should be when redistributed if the new ID entry had already been inserted. A fail exit is taken if there is no room for the new entry. Next the IDs are packed into the last most ID tracks. Then the tracks, starting with the first one is filled to the required length. The track the new entry is to go on is filled with one less entry and the first word of the track's IDEC is set to the new ID, this is so the FIDT routine will find this track, to insert the new ID, when SUPERNEW returns to step 7 of NEWID.

PHONES Command

The PHONES command informs the IOP of the number of seconds that a user is allowed to have for log on attempts before we hang up the phone. The IOP starts this timing when a port initially brings up data set ready. The time is initially set to 120 seconds at IOP generation. It can be set to 0 to effectively prohibit access to the system (the IOP does this by setting a nominal time of .2 seconds which is too short to allow anyone to log on).

1. Extract, convert to binary, and validate the time value.

2. Send this value to the IOP and exit the routine.

PURGE Command (System Operator)

The PURGE command deletes any programs, files, or non-shareable device designators that have not been referenced on or after the date specified. The method is to scan the directory for qualified entries, deleting them as they are found. As the entries are found, we build a table of them immediately beyond the directory track in memory. Whenever the table fills, we temporarily suspend processing the directory to call in an overlay which returns the space occupied by purged entries to the IDT and the ADT.

1. Interpret and validate the date given.

2. Reject command if any files are in use by scanning the TTY tables for any OUT= files and for any FUSS table entries.

3. Update the last reference date of the HELLO progam (if any) so it does not disappear.

4. Scan all directory tracks on system, deleting entries when they qualify. Add entry to patch table.

5. When patch table is full, remove space from IDT and ADT.

6. When last directory track has been scanned, remove space from IDT and ADT if any entries in patch table (see user's PURGE command for a description of RTIDT and RTADT). RTIDT/RTADT will exit back to the system if last call.

REPORT System Console

The REPORT command prints IDT information on the system console. For each IDT entry, the user ID, the time used, the disc space used, its capabilities, and its device designators. Note that the time used does not include time consumed of active users.

1. If an ID was specified get it and save it in REPLN. If not save 1 in REPLN.

2. Print the first heading line and suspend.

3. Retrieve the ID in REPLN and find the ID or the first ID passed it. If the ID is beyond the last one on the system then print the second line of the heading and terminate. Otherwise set REPLN = -number of words from the ID entry to the end of the track and set REPPT to point to the -length of the ID track in the IDEC table.

4. Print the second line of the heading and suspend.

5. Determine from REPLN and REPPT the block on the disc the ID entry resides.

6. Read in the ID entry and set REPID to point to it in memory.

7. Build the report line in T35BF.

8. Bump REPLN by 12. If the result is zero (the entry just processed was the last one on the ID track) go to 9. Otherwise print the line, suspend and go to 5.

9. If there are no more ID tracks then print the line and terminate.

10. Reset REPPT to point to the ID track's-length and REPLN to the -length of the new track. Print the line, suspend and go to 5.

RESET

The RESET command modifies the time used to a date of a user's IDT entry. It operates as follows:

1. Set ID = REST = 0.

2. If the idcode = "ALL" go to 3, otherwise set ID equal to the specified ID code and read the proper IDT track.

3. If no time specified, go to 4. Otherwise set REST equal to specified time.

4. If ID = 0, go to 5. Otherwise search for the specified idcode. Fail if not found. If found, set its time entry to REST, write the IDT track back and terminate.

5. Set the time entry for all the idcodes on this track to REST and write it back to disc.

6. Move to the next IDT track. If all are finished, terminate. Otherwise read the IDT track and go to 5.

ROSTER

The ROSTER routine prints a listing of the idcodes of all active users. These are obtained from the ?ID word in the 32 TTY TABLES. The absence of a user is indicated by the word being zero.

RJE Command

The RJE command extracts an RJE message from the console buffer and notifies the IOP that a message is pending. At some later point in time, the IOP is supposed to send the system a wake RJE up (WRU) command. This sets a flag which the scheduler interrogates and, if a message is pending, will send the message to the IOP. The RJE interconnect kit command serves two purposes. Within this command it only notifies the IOP that a message is pending; we do not bother to check for acceptance since we know that it is always rejected the first time (this is incredibly bad programming and should be changed at the first opportunity). In the scheduler we actually check for acceptance and, if accepted, send the RJE message.

1. Reject command if previous message is still pending.

2. Extract message and place into RJE command buffer (currently resident following the library overlay area).

3. Save message length in first word of buffer.

4. Nofity IOP that RJE command is pending. We do not need to pick up the reply code since it will sit on the interconnect kit without harm. The STC. C done by the IOP serves as our acknowleagement that the command was accepted.

SLEEP

The SLEEP command is used for system shutdown. It operates as follows:

1. Remove all users from the queue and make sure they can't get back by:

   a. Clearing each user's ?FLAG word except for the DFCHK bit, in his TTY table.
   b. Setting all status words to -2 (%DISC).
   c. Sending a KTU to all active terminals (?ID =0).
   d. Setting T35LK to point to MLINK+1.

2. Kill all busy devices and clear the fifth word of each device table entry.

3. Output the sleep message to all active users, preceeded and followed by a CRLF.

4. Tell the I/O processor were going down by sending a SSD.

5. Call LCD to update the last change date for files for each port that has its DFCHK bit still set. Clear the DFCHK bits when done with each port.

6. Update the IDT entry for each active user and create a logoff entry in LOGGR.

7. Wait for the console to finish outputting.

8. Read in the loader, turn off all the I/O and the interrupt system, set power fail to halt.

9. Set A = 0 (sleep) and jump to the dump (200008 in the loader).

STATUS - SYSTEM CONSOLE

The STATUS routine prints a summary of the various system resources and the extent of their utilization on the system console. It operates as follows:

1. Print the heading consisting of system id, date and time and suspend.

2. Print the logical unit, select code, unit number, first block and last block of the discs on the system.

3. Print a list of those disc blocks which have been LOCKED.

4. Print the disc addresses and lengths of the IDT, ADT and Directory tracks.

5. Print the disc addresses and lengths of the system segments.

6. Print the user swap track disc addresses.

7. Terminate.

PART III

I/O PROCESSOR PROGRAM

HEWLETT-PACKARD 2000 SYSTEM I/O PROCESSOR

CONTENTS

I. OVERVIEW

II. SYSTEM SERVICES AND DATA STRUCTURES

III. TERMINALS

IV. NON-SHAREABLE DEVICES

V. RJE

VI. PROBLEM ISOLATION

VII. IOPC

I. Introduction

I/O processor software in the HP2000 ACCESS system provides two concurrent and ostensibly independent functions: The simultaneous support of up to 32 time share users and a Remote Job Entry (RJE) facility for access to batch systems in IBM 360 and 370 computers. This access is gained through a line printer and card reader attached to the I/O Processor (IOP) of the HP2000 ACCESS system. The TSB user also has access to the remote batch system via a remote file transfer capability.

The RJE function utilizes the IBM multileaving communications protocol. This protocol was originally only available with IBM operating systems including HASP but is now also supported by IBM's ASP, VS1-JES, VS2-JES2, and VS2-JES3. The protocol provides for concurrent operation of multiple data streams, both transmitted and received, and includes data streams specifically used for remote operator control. This multiple stream capability will be specifically exploited in future system enhancements.

The time-share function includes more comprehensive error detection, automatic speed detect, and terminal user access to non-shareable devices attached to the IOP (e.g. card readers and line printers).

II. Design Overview

Design assumptions

Certain elements of hardware required by this project are new to a 2000 series time share system. Microprogramming is used to improve performance through special purpose instructions. The RJE communications I/C support is based on a 12618 synchronous transmit/receive interface. A variety of card readers, line printers and paper tape punches may be used as peripherals available to both the TSB and RJE user.

Operational assumptions

The RJE function typically requires specific attention on the part of an operator. He is required to activate and deactivate the function as well as tc control its activities (operator control and commands, reader and printer operaticns, and communications line management). Although it is not required that the operator be knowledgeable of host system (IBM 360 or 370) job processing, he should be familiar with remote operational procedures. This knowledge can cnly be acquired in cooperation with IBM and personnel at the host site.

It is expected that this software will be used in an environment having a requirement for both time sharing and remote batch processing but having no need for interaction between the two. Remote file transfer capability provides the TSB user access to the RJE function.

Performance of the system is designed around the assumption that the TSB user and his respcnse times are more important than the RJE function. For this reason, the RJE function will generally receive lower priority than TSB and may show degraded performance when TSB activity is heavy.

Design summary

The structure of the I/O Processor software is an interrelated collection of mcdules each performing a predefined task. Two groups cf modules exist. One group consists of the IOC (I/O Control) module and all I/O drivers. Based on the Hewlett-Packard Basic Control System (BCS), this group of modules encompasses all I/O service, scheduling, and control. The second group of modules makes up the actual data processing system. These modules are divided into three general classes: supervisor, manager, and function handler. The supervisor sectien provides run time control of which modules are executed; the manager

section provides services utilizable by any other module in the system; and function handlers control devices and/or major processes in the system.

The supervisory task is carried out by the Dispatcher, which controls the execution of function handlers on a priority basis. Function handlers are dispatched when an I/O operation, initiated by them, completes; or when explicitly scheduled through the Queue Manager. This explicit scheduling of a handler is called "priming." All such execution control is performed by the Dispatcher and control is always returned to the Dispatcher when a handler completes its processing.

Managers provide communication and support services for the handlers. Interhandler communication is provided through the Queue Manager (QM). The QM is the central message switching routine in the system; maintaining work queues, buffer routing, and priming of handlers when work is queued. The Buffer Manager (BM) controls all buffer acquisition and release, and administers all buffer pools in the system. The Allocate/Deallocate Manager (ADM) controls the use of non-shareable resources by the IOP. Through its services a non-shareable resource, such as the line printer, may be assigned to a particular task, such as RJE. In addition to these managers, several general purpose data handling routines and common subroutines are included.

Function handlers control devices or major processes in the system. Associated with each handler is a queue or queues. These queues are the source of work for the device or process which the handler controls. Handlers typically initiate I/O operations to devices (through the use of queued IOC) and perform necessary processing when the operation completes. Other handlers never perform I/O. Instead they perform major processing functions and are driven strictly by Dispatcher priming. Through the services of the QM, handlers inform one another of work to be performed by making entries on work queues. The handlers also acquire and release buffers and the use of non-shareable resources where appropriate. The term "handler" or "function handler" should not be confused with "driver" at the BCS IOC level. The function handlers in current use are:

1. Interconnect Kit Handler
2. Multiplexor Handler
3. ASCII File Handler
4. Time Base Generator Handler
5. Line Printer Handler
6. Card Reader Handler
7. Synchronous Communications Handler

8. Console I/O Handler
9. Host Inquire Compression Handler
10. Host Message Decompression Handler
11. Host Reader Compression Handler
12. Host List Decompression Handler
13. Paper Tape Punch Handler
14. Photo-reader Handler
15. Reader/Punch/Interpreter Handler

Time-Share Function

The time-share function of the I/O processor is graphically illustrated in figure 1. Two distinct types of data flow are implicit in this function: the terminal user/system dialogue and communications with ASCII file devices. Several handlers participate, some on a shared basis, in these communications; specifically, the interconnect kit, the ASCII file, the multiplexor, the card reader, the paper tape punch, the photo-reader, the reader/punch/interpreter, and line printer handlers.

Major module relationships

The user/system dialogue is conducted exclusively through the interconnect kit, serving as an interface with the system processor, and the multiplexor, functioning as the terminal/user interface. Although each handler is served by a single queue, the connection does provide the capability for 32 concurrent and independent data streams (one for each terminal user). Neither handler is allocatable by the Allocate/Deallocate Manager since they are permanently associated with each other. Operations on any data stream are controlled by the interconnect kit handler at the dictate of the system processor. This control is exercised by the dispensation of buffers by the interconnect kit for reading, writing, and control operations. The only exception to this structure is the ability of the multiplexor handler to request special processing for unusual conditions, i.e., line break or break character received.

ASCII file communications are carried on between the system processor via the ASCII file handler, and an external I/O device via the handler dedicated to its operation. The ASCII file handler is served by a single work queue supporting multiple data streams as in the case of the interconnect kit handler. Such is not the case, however, for the external device handlers which are single queue, single data stream processes. The information

DATA FLOW OF TIME SHARE FUNCTION

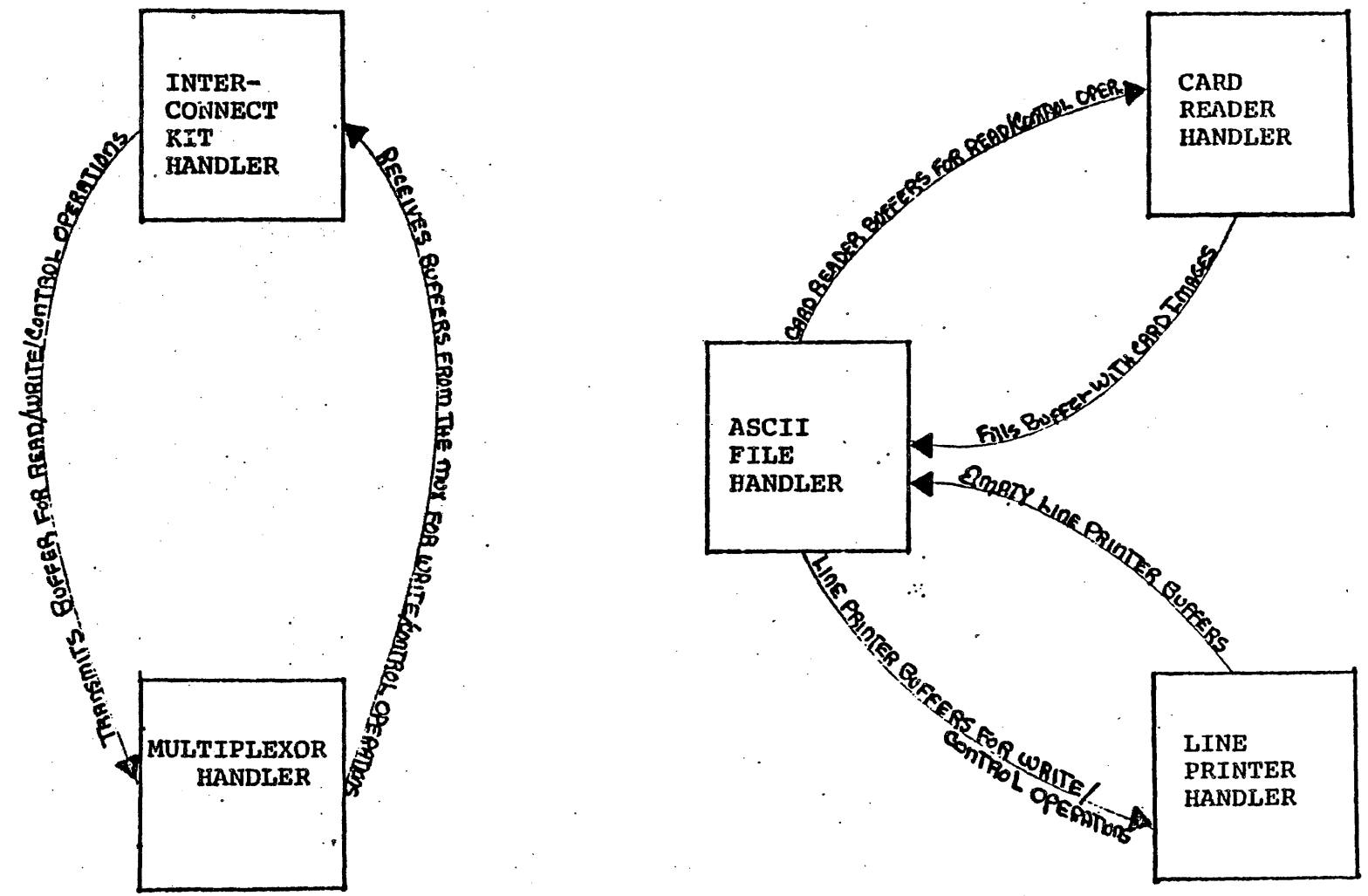


FIGURE 1

necessary for data stream identification is solely a function of the ASCII file handler and need not concern the external handlers. The connections between handlers performing ASCII file operations are not permanent, since the devices themselves, the card reader and line printer, are also required by the RJE function. While the connections are established, however, control is vested in the ASCII file handler as it is in the interconnect kit for terminal traffic. That is, control is exercised by dispensing buffers for reading, writing, and control operations.

Data flow

The following message flow is possible among handlers for the time-share function:

Interconnect kit - transmits buffers to the multiplexor for read, write, and control operations. Receives buffers from the multiplexor for write and control operations and empty buffers for discretionary use.

Multiplexor - transmits empty buffers to the interconnect kit for its discretionary use and also sends buffers for write or control operations. Receives buffers from the interconnect kit to be used as directed; read, write, or control.

ASCII files - transmits buffers to the card reader and line printer to be used as directed; read, write, or control. Receives buffers from these handlers for transmittal to the SP (write or control) or for discretionary use.

Card reader - transmits card images to the ASCII file handler and receives buffers from the ASCII file handler to be used as directed (read or control).

Line printer - transmits empty buffers to the ASCII file handler for its discretionary use and receives buffers from the ASCII file handler to be used as directed (write or control).

RJE Function

Figure 2 illustrates the RJE function components. Some of these components represent non-shareable, allocatable resources for which the RJE function competes along with TSB users. Some software modules depicted in figure 2, therefore, are the same ones shown in figure 1. Three general levels of software exist. Central to the design is the synchronous communications handler. Here are isolated (as much as possible) details of line protocol and control. All line I/O occurs at this level. This module is also adaptable to the configuration of the remaining two levels. At system initialization time, it determines that configuration and subsequently deals with it. Thus, for example, the number of input data streams need not be known to this level in advance.

At the second level are all modules which interface directly with the level one communications module. Some are input related modules which acquire and preprocess data to be transmitted. In the case of multileaving, preprocessing involves blocking and compression of data. Other modules are output related. They accept blocks of data from the synchronous communications module and prepare it for output. With multileaving this means deblocking and decompressing. At system initialization time, the synchronous communications handler at level one determines how many input and output modules exist at level two. At run time it then provides instructions to these modules about current running conditions.

Level three modules are concerned with actual I/O operations. These modules may represent allocatable resources (e.g., a line printer) which a level two module must acquire. These acquisitions are typically initiated by operator commands or other signals from the communications line. Once the acquisition is complete, the associated level two module either provides data to or accepts data from the level three module. These so-called level three modules are not unique to the RJE function. They are in fact the same modules which will be used by the ASCII files handler. In the case of remote file transfer, the level three module role is played by the ASCII files handler.

Console communications require special attention. The console (ASR-35) is attached to the system processor, not to the I/O processor. For this reason, minor changes to the system processor have been designed to allow transfer of messages to and from the I/O processor. These messages either originate from or arrive at a level three module concerned with console I/O. Due to the often unique nature of console messages, required interaction

between this module and any other module in the system is made possible.

Major Module Relationships

Seven level three modules will be used. The line printer handler will be the same module used by the ASCII file handler. It does represent an allocatable resource and will therefore have no permanent connection to the level two module (host list decompression) with which it communicates during the RJE function. A card reader handler will also exist. It represents a potentially allocatable resource. Like the printer handler, it will have no permanent connection to the level two module (host reader compression) with which it communicates. The card reader handler is being designed for general purpose future use and will be able to read data in several modes. The third level three module is a console I/O module. This module is not allocatable since it represents a resource shared by everyone. Any module may present output messages to it for display on the console. Also, it will route messages received from the console to the intended recipient of the message. The fourth level 3 module is a paper tape punch handler. It also represents a potentially allocatable resource and will have no permanent connection to the level two module (host punch decompression handler). The fifth level three module is the photo-reader handler. It represents an allocatable device and will have no permanent connection to the host reader compression handler. The sixth level three module is the reader/punch/interpreter handler. It also represents a potentially allocatable resource and may be temporarily connected to the host reader compression handler or the host list or punch decompression handlers. The seventh level three module is the ASCII files handler. While it normally plays a level two role in behalf of a TSB program which is accessing some peripheral, it here plays a level three role. It may communicate with either the host list decompression, host punch decompression, or host reader compression handlers to perform the job list, job punch, or job transmit functions for the TSB user.

Five level two modules will be a part of the design. One host list decompression module and one host punch decompression module will exist. The module is directed by the synchronous handler to obtain a printer or punch. Following such acquisition, data is accepted, deblocked, decompressed, and translated for delivery to the associated printer or punch handler. A similar function is performed by a single host message decompression module. One difference is that no allocation takes place as the console is always available. One host reader compression module also exists. It is activated by the synchronous handler and

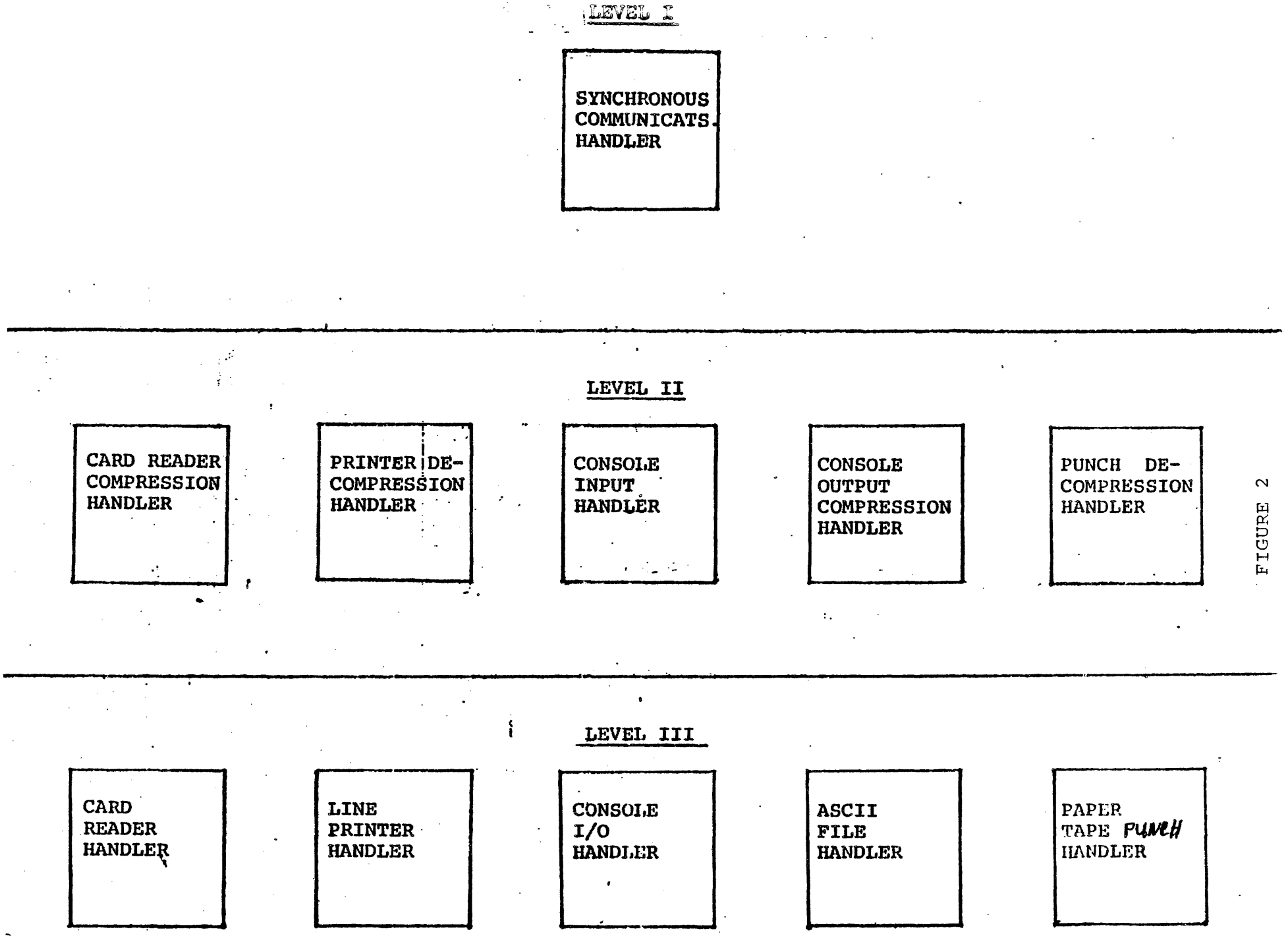

LEVEL I
SYNCHRONOUS COMMUNICATS. HANDLER
LEVEL II
CARD READER COMPRESSION HANDLER
PRINTER DE-COMPRESSION HANDLER
CONSOLE INPUT HANDLER
CONSOLE OUTPUT COMPRESSION HANDLER
PUNCH DE-COMPRESSION HANDLER
LEVEL III
CARD READER HANDLER
LINE PRINTER HANDLER
CONSOLE I/O HANDLER
ASCII FILE HANDLER
PAPER TAPE PUNCH HANDLER

FIGURE 2

operator commands in order to allocate a reader for reading, compressing, and blocking of data. These data blocks are then delivered to the synchronous handler. A similar module performs preparation of console input messages. Centralized compression and decompression subroutines reduce the redundancy of sections of these modules.

Some special comments about the design of the host list decompression and host reader compression modules at level two are needed. In anticipation of future requirements for multiple streams, these level two modules will be coded in a "serially reusable" manner. This means that all potentially volatile data associated with a particular data stream will be isolated from the code itself. In essence, the code may be viewed as a subroutine which operates on a data structure provided to it. Then the actual level two function handler consists only of the block of data for a stream plus those instructions needed to invoke this "serially reusable" subroutine. To add a second or third card reader stream would require only the replication of this data structure component. Again, due to the unique nature of console activity and requirements, level two console modules will not follow this design.

The synchronous communications handler at level one is unique in that it is driven by nine distinct input work queues. Fifteen distinct output work queues are also supported. The number of queues actually used is determined at system initialization time. One console input stream and up to seven card input streams make up eight of the input work queues. The ninth is a general purpose work queue used to receive operator control commands and other items such as buffers. The fifteen possible output work queues are a console output work queue, seven printer output work queues and seven punched card output work queues. Obviously very few of these will be utilized initially. Communications line activity is initiated and deactivated by this handler as dictated by operator control commands.

Data flow

The following message flow is possible among handlers in the RJE function:

Console I/O - Sends reader control commands to the host reader compression module, communications line control commands to the synchronous handler, and remote inquiry commands to the host inquire compression handler, and remote inquiry replies to the ASCII files handler for the job message function. Receives console messages from virtually anyone.

Host inqure compression - Sends blocks of compressed remote inquiries to the synchronous handler. Receives remote inquiries from console I/C and operational control signals from the synchronous handler.

Host message decompression - Sends messages to console I/O and control information to the synchronous handler. Receives blocks of console messages from the synchronous handler.

Card reader - Sends card images to host reader compression. Receives control signals from card reader compression.

Host reader compression - Sends blocks of card images to the synchronous handler, reader activity messages to console I/O, and control signals to the level 3 handler (card reader handler or ASCII files handler). Receives control signals from the synchronous handler, reader control commands from console I/O, and card images from the level 3 handler.

Printer - Sends control signals to host list decompression. Receives print images from host list decompression.

Host list (or punch) decompression - Sends control signals to the synchronous handler, print or punch images to the level 3 handler (printer handler, paper tape punch handler, or ASCII files handler), and printer activity messages to console I/O. Receives control signals from the level 3 handler and synchronous handler, and blocks of output images from the synchronous handler.

Synchronous handler - Sends messages to console I/O in response to connection requests, blocks of output data to all level two output modules (host message decompress and host list decompress), and control signals to all second level modules. Receives input blocks from all level two input modules (host inquire compression and host reader compression), commands from console I/O, and control signals from output modules.

ASCII files handler - When playing its level three handler role, the ASCII files handler can be the source of data for the host inquire and host reader compression handlers. At the same time, it can be the sink for the data provided by the host message and host list and host punch decompression handlers. In this capacity, the module is the link between the TSB user and the RJE subsystem.

Function handlers

All system function handlers are described in the following sections. Names associated with these function handlers are as follows:

Name	Level	Handler
ICKH	1	Interconnect Kit Handler
MUXH	1	Multiplexor Handler
ASFH	2/3	ASCII Files Handler
TBGH	1	Time Base Generator Handler
LPRH	3	Line Printer Handler
CRH	3	Card Reader Handler
SFH	1	Synchronous Communications Handler
CIO	3	Console I/O Handler
HIO	2	Host Inquire Compression Handler
HMO	2	Host Message Decompression Handler
HRO	2	Host Reader Compression Handler
HLO	2	Host List Decompression Handler
HPO	2	Host Punch Decompression Handler
PPH	3	Paper Tape Punch Handler
PRH	3	Photo-reader Handler
RPH	3	Reader/Punch/Interpreter Handler

Handlers are roughly classified into three levels. Level one handlers are typically in control of major processing areas. Level two handlers also are involved with major processing areas and are often very closely tied to a level one handler. (An example is the activity between HRO or HLO and SFH.) However, the main point which distinguishes a level two handler from a level one handler, is that a level two handler will acquire and control a level three handler to perform work on its behalf. In order that a given level three handler can be used by <u>any</u> level two handler (e.g. ASFH-CRH or HRC-CRH), a definite message and control flow protocol is defined for using a level three module. This protocol is defined as follows:

1. A level two module (L2) allocates a level three module (L3) using the ADR. L2 sends a "start" command to L3. This command is sent in a control buffer and is used to activate the connection. In this buffer are parameters needed by both L2 and L3 to maintain the connection as well as to initiate it:

	word	Bit(s)	Function
BCW	1	15-11 7-4	May be used by L2 to inform L3 about any desired operational modes

	2	13-8	Informs L3 of data stream identifier to be used in BCW2 of all future message exchanges
	3	14-0	Queue name of L2 to which future messages are returned
	7	15-0	Length of required buffers in positive bytes
	8	15	1 = RJE module issued start 0 = ASPH issued start
		14-2	May be used as needed
		1-0	Indicates basic operational modes For input: 00=read ASCII 10=read EBCDIC For printer output: 00=Space then print 01=Print then space For paper tape punch output: 00=punch mode in Data word 1 01=punch mode changed by control operation
Data	1	15-0	Number of buffers needed for the connection (Zero is not a legal number)

Upon receipt of this message, L3 is expected to:

a. Retain supplied information.
b. Perform any initial I/O operations (such as emitting punched tape leader).
c. Acquire the number of buffers specified in data word 1, interlock them, and return them to L2 with BCW3 containing the queue name found in the start command, and BCW2 containing a "nop" command and data stream identifier as noted above.
d. Free the control buffer containing the "start" message.

2. L3 accepts read or write commands from L2. These will appear in the buffers supplied to L2 by L3 as noted in (1) above. BCW word 7 is used by L2 to indicate the desired positive length in bytes (characters) of reads or writes. Upon completion of the operation, the following occurs:

a. If a read, the command is changed to write.
b. If a write, the command is changed to nop.

c. The buffer is returned to L2 with a .PUTQ if a write or with .PRIQ if a nop.

If an operation completes in error, the following occurs:

a. The original operation is held by L3 for later retry either after a timed pause or at operator direction.
b. A control buffer is acquired by L3 and is set with an error command (6), appropriate BCW word 8 general error indicators, and stream identification. This buffer is sent to L2.
c. For all errors, a retry must be indicated by receipt of a type 9 command. This may come directly from an operator or may be generated internally by L2. For a type 1 or type 2 error (see BCW word 8 description), a timed retry is performed by L3. Only when a device makes a transition from ready to not ready should L3 report the error to L2. A continued not ready condition after timed retry is not reported. For type 3 errors, an immediate retry is performed upon receipt of the type 9 command.

3. L3 may receive control commands from L2. These are defined by L3, and the buffers are discarded by L3.

5. A "purge" command from L2 to L3 may appear in a buffer. The command will be in a non-interlocked buffer which should be released by L3. In response to the purge, L3 should:

a. Purge any pending operations and return the buffers to L2 as nop's.

5. A "stop" command from L2 to L3 may appear in a buffer. The stop command must be in an interlocked buffer for eventual return to L2. L3 must do the following in response to the stop:

a. Complete any pending operations.
b. Perform any final task (such as emitting a punch tape trailer).

c. Return the "stop" message to L2 <u>after</u> having returned all read or write buffers to L2. It is always the responsibility of L2 to release buffers. Read and write buffers active at the time of a stop are considered purged. It is not necessary to complete operations with them, and they may be returned as nops.

SECTION II

SYSTEM SERVICES AND DATA STRUCTURES

HEWLETT-PACKARD 2000 SYSTEM I/O PROCESSOR

SYSTEM SERVICES AND DATA STRUCTURES

CONTENTS

1. IOC — QUEUED IOC
2. BCW'S — GLOBAL USAGE OF BCW WORDS
3. QM & QIT — QUEUE MANAGER AND QUEUE INFORMATION TABLE
4. .COM. & IODT — DISPATCHER/COMMUTATOR AND I/O DISPATCHING TABLE
5. BM,SCOL,BPRT — BUFFER MANAGER, SUBPOOL CONTROL LIST, BUFFER PENDING REQUEST TABLE
6. ADM — ALLOCATE/DEALLOCATE MANAGER
7. DAM & DAT — DEVICE ASSIGNMENT MANAGER AND DEVICE ASSIGNMENT TABLE
8. MISCELLANEOUS OTHER MN SERVICE ROUTINES
   SYSTEM STARTUP/RESTART
   CENTRALIZED CONSOLE OUTPUT
   DATA CONVERSION AIDS
   DECOMPRESSION SERVICE
   COMPRESSION SERVICE
9. DEVICE TABLE
10. D.04 — POWER FAIL AUTO-RESTART MODULE
11. D.43 — TBG DRIVER
12. TBGH — TBG HANDLER
13. IOP MICROPROGRAM DEFINITIONS

I. Introduction

Queued .IOC.

Queued .IOC. is the input/output control subroutine within the I/O subsystem of the IOP program for the HP2000 ACCESS system. It provides for scheduling of I/O requests and notification of the completion of these requests. Queued .IOC. differs from existing versions of .IOC. (non-buffered and buffered) in that it will maintain queues of I/O requests for individual devices and a priority queue of completed requests for all devices. Also, facilities are available to interrogate the queue of completed requests and to manipulate the scheduling of I/O. DMA use and conflicts for use are also managed and scheduled by Queued .IOC.

Queued .IOC. is being developed to make viable the use of BCS type operating system in the I/O processor of an HP2000 ACCESS system. However, Queued .IOC. need not be restricted to this use. This document will deal primarily with the features of Queued .IOC. which are different from current versions of .IOC. Reference to existing documents on BCS and .IOC. is suggested for answers to questions not covered here.

II. Design Overview

Objectives

The major goal for Queued .IOC. is to generate a group of new optional services which will enable BCS to function in an environment having time dependencies and a requirement for priority handling of events.

Special requirements

Queued .IOC. will require an equipment table (EQT) of larger size than previous versions of .IOC. The special microcode developed for use in the IOP software of the HP2000 ACCESS systems is also required.

Design summary

Queued .IOC. will be modular in organization. The functions to be performed by Queued .IOC. are very distinct so that modularity is quite natural. In addition, processing which is common to several functions can be implemented as subroutines.

Three primary modules will exist. First is the module associated with the .IOC. entry point. This module receives control from the user program and processes the request appropriately. Secondary modules will process the various types of requests -- read, write, control, status, etc. Several of these modules will process the request directly or queue it for later processing by the I/O driver. In other cases the I/O driver will be entered to process or start processing the request.

The second primary module is that associated with the .BUFR entry point. Here the completed requests may be placed on a prioritized queue with other completed requests (the CRQ or completed request queue). If another operation has been requested and queued for the device, it will be initiated at this point.

The third primary module is that associated with the .UNS. entry point. .UNS. allows a driver to place into the completed request queue, an entry containing information about some unsolicited asynchronous event. Such events are unsolicited in that no .IOC. call brings them about. (An example might be the receipt of a BREAK character from a communications line.) By placing such events into the completed request queue, information can be passed back to the user program for action at that level.

Flow of control

It is imperative that the capabilities available to the user program through Queued .IOC. be understood. A summary of the capabilities follows.

Perhaps most important is the fact that all capabilities formerly available through non-buffered .IOC. are still available and in an identical form. This is also true of buffered .IOC. with the exception that multiple buffered output requests will not be possible with Queued .IOC. A reading of the remainder of this section will reveal that multiple output requests can be done but in a slightly different manner.

A significant new capability exists to allow scheduling of multiple read, write, or control requests. Coupled with this is the ability to handle the completion of these requests on a priority basis. The flow of control is as follows--

1. The calling program uses an .IOC. parameter word having bit 15 set to indicate a queued request. (May be read, write, or control.)

2. For reads and writes, the buffer supplied for the call is assumed to be preceded by 5 queueing and control words for use by Queued .IOC.
3. The .IOC. call is made. .IOC. may enter the I/O driver immediately (if the driver is idle) or queue the request for later processing by the driver. This latter queue is referred to as the device request queue or DRQ.

   Certain specialized I/O drivers do not depend on the device request queue but handle queueing of multiple requests themselves. An example is the D.51 multiplexer driver. The EQT associated with such a driver is marked as using no device request queue. All requests are immediately presented to the driver in this case. (See the section entitled "Self-queuing I/O drivers" elsewhere in this document.)
4. The driver completes the requested operation and exits through .BUFR (an entry point in .IOC.).
5. .BUFR places the completed request into the completed request queue. This queue is arranged in priority order. The logical unit number is used as a default priority. However, the priority for a given completed request may optionally be selected by the user program. This capability is discussed below.
6. The user program uses an interrogate request to inspect the completed request queue. In this way the observance of the completed request is made.

Since available DMA channels are shared by all I/O drivers needing DMA, contention for these channels exists. An I/O driver which needs DMA but which is unable to acquire a DMA channel will reject the I/O call with the B register set to 1 (DMA needed but unavailable). For queued requests only, Queued .IOC. will resolve this conflict. A reject of this sort will cause the I/O operation to remain queued and the EQT to be marked with a flag indicating "wait for DMA." As other I/O operations complete, Queued .IOC. will note the availability of DMA channels and will reschedule an EQT suspended for this reason. Note that non-queued I/O requests are not handled in this manner and will still result in the traditional reject at the .IOC. call.

The interrogate request is a special .IOC. call which asks for notification of a completed request or unsolicited event. The unit reference number in this .IOC. call designates the lowest priority to be inspected. In other words, if it is desired to inspect for completed requests at priority x or higher (<x), then x is the unit reference number used in the .IOC. call. A

rejection return is used to indicate that no completed requests at the designated priorities were found. For a successful return, registers are set to supply information about some completed request or unsolicited event.

Three other capabilities exist. One allows the user program to supply the address of a subroutine which .BUFR will use to choose the priority at which a completed request is to be queued. Between steps 4 and 5 noted above, this subroutine is activated. The priority it returns for the completed request will over-ride the default priority (logical unit number) for the device. Another capability allows a device request queue to be purged. All pending requests are immediately moved to the completed request queue. Finally, an .IOC. call is available to activate a device request queue which has been suspended due to an I/O error. Such suspension is indicated as necessary to .IOC. whenever an I/O driver indicates a need for operator attention in the EQT status word.

III. Design Structure

Organization of queues

Queued .IOC. is concerned with two queues. One is the Device Request Queue (DRQ), and the other is the Completed Request Queue (CRQ).

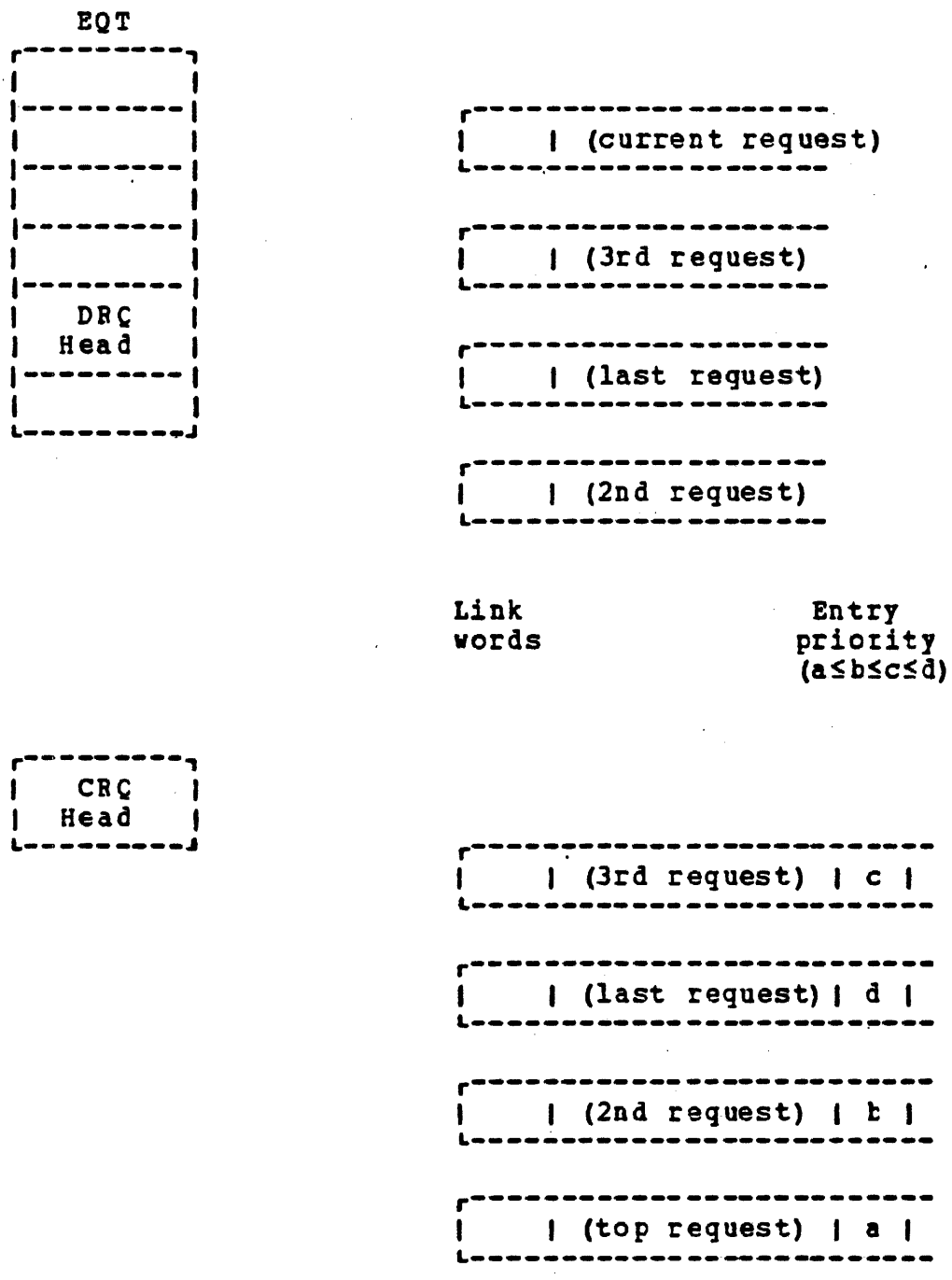
For each EQT, a DRQ can be maintained to hold multiple I/O requests for the unit. The organization of the DRQ is based on two things. One is a word in the EQT called the DRQ head. It holds the address of the current request. The second thing is the set of five queue control words associated with each request to be queued. For read or write requests these control words are provided by the user program. (See "Buffer use with queued requests" under the section on "Data formats and .IOC. calls".) For control requests the control words are obtained by .IOC. Using the DRQ head and the five control words, the DRQ is organized as shown in the illustration on the next page.

The CRQ has a very similar organization. The CRQ head is location within the .IOC. module which addresses the highest priority completed request. Other completed requests are then chained in priority order. The same five control words used for requests on the DRQ are used for the CRQ. CRQ organization is also illustrated on the next page.

As shown in the illustration, the first of the queue control words is used to link requests together. The appropriate head

addresses the frist request and each raquest in turn addresses its successor. The last request has a zero link word. In the case of the CRQ, the priority of the request is stored in another of the five words (the last).

Illustration of queue organization



Q U E U E D I O C

QUEUED .IOC. IS THE INPUT/OUTPUT CONTROL ROUTINE WITHIN THE I/O SUBSYSTEM. IT IS RESPONSIBLE FOR THE SCHEDULING OF I O REQUESTS AND THE NOTIFICATION OF THE COMPLETION OF THESE REQUESTS. THE FOLLOWING TWO QUEUES ARE MAINTAINED BY QUEUED .IOC TO AID THE SCHEDULING OF THESE TASKS:

DEVICE REQUEST QUEUE (DRQ)

FOR EACH QUEUING DRIVER EQUIPMENT TABLE (EQT). A DEVICE REQUEST QUEUE IS MAINTAINED TO HOLD MULTIPLE I/O REQUESTS EACH DEVICE REQUEST QUEUE HAS A DRQ HEAD (EQT WORD 4) WHICH CONTAINS THE ADDRESS OF THE CURRENT I/O REQUEST AND A LINKED LIST OF FIVE QUEUE CONTROL WORDS ASSOCIATED WITH EACH REQUEST TO BE QUEUED. FOR CONTROL REQUESTS THE QUEUE CONTROL WORDS ARE OBTAINED BY THE .IOC. SUBROUTINE VIA A CALL TO .GET. FOR READ AND WRITE REQUESTS THE CONTROL WORDS ARE SUPPLIED BY THE CALLING MODULE AS FOLLOWS:

EACH READ AND WRITE REQUEST HAS A BUFFER ASSOCIATED WITH IT. A BLOCK OF NINE BUFFER CONTROL WORDS BCW WORDS) IS ATTACHED TO THE BEGINNING OF EACH BUFFER. BCW WORDS 5 THROUGH 9 ARE USED BY .IOC. AS THE SET OF DRQ CONTROL WORDS.

THE DRQ QUEUE CONTROL WORDS ARE DEFINED AS FOLLOWS

WORD 0 (BCW WORD 5) - QUEUE LINKAGE
WORD 1 (BCW WORD 6) - .IOC. I/O REQUEST PARAMETER
WORD 2 (BCW WORD 7) - BUFFER SIZE (N/A FOR CONTROL REQUESTS)
WORD 3 (BCW WORD 8) - NOT USED
WORD 4 (BCW WORD 9) - NOT USED

COMPLETED REQUEST QUEUE (CRQ)

THE CRQ IS A PRIORITIZED QUEUE OF ALL COMPLETED I O REQUESTS AND OF ALL UNSOLICITED EVENT REQUESTS (UNSOLICITED IN THAT NO .IOC. CALL BRINGS THEM ABOUT). THE CRQ HEAD IS LOCATED WITHIN .IOC. AND IT CONTAINS THE ADDRESS OF THE HIGHEST PRIORITY CRQ ENTRY. OTHER REQUESTS ARE LINKED IN PRIORITY ORDER. THE SAME FIVE QUEUE CONTROL WORDS USED FOR DRQ ENTRIES ARE USED FOR CRQ ENTRIES. QUEUE CONTROL WORDS FOR UNSOLICITED EVENTS ARE OBTAINED BY THE .UNS. MODULE VIA A CALL TO .GET. THE COMPLETION PRIORITY OF A CRQ ENTRY FOR SOLICITED EVENTS IS THE UNIT REFERENCE NUMBER OF THE DEVICE FOR WHICH THE ENTRY IS BEING MADE. THE COMPLETION PRIORITY FOR UNSOLICITED EVENTS CAN BE SPECIFIED BY THE CALLING DRIVER OR THE UNIT REFERENCE NUMBER CAN BE USED AS A DEFAULT PRIORITY. (SEE .UNS. DESCRIPTION) THE CRQ CONTROL WORDS ARE DEFINED AS FOLLOWS:

ALL CRQ ENTRIES EXCEPT UNSOLICITED EVENTS

WORD 0 (BCW WORD 5) - QUEUE LINKAGE
WORD 1 (BCW WORD 6) - .IOC. I/O REQUEST PARAMETER
WORD 2 (BCW WORD 7) - TRANSMISSION LOG FROM EQT WORD 2
WORD 3 (BCW WORD 8) - STATUS FROM EQT WORD 1
WORD 4 (BCW WORD 9) - COMPLETION PRIORITY

CRQ ENTRIES FOR UNSOLICITED EVENTS

WORD 0 - QUEUE LINKAGE
WORD 1 - TSB LOGICAL UNIT NUMBER/UNIT REFERENCE NUMBER
WORD 2 - NOT USED
WORD 3 - EVENT DATA
WORD 4 - COMPLETION PRIORITY

A CRQ ENTRY IS RECOGNIZED AS BEING FOR AN UNSOLICITED EVENT BY THE FACT THAT THE COMMAND BITS OF CRQ WORD 1 (BITS 14:12) ARE ZERO. IN THIS CASE CRQ WORD CONTAINS THE EVENT DATA WHICH IS DEFINED BY THE DRIVER PRESENTING THE UNSOLICITED EVENT.

QUEUED .IOC. IS MADE UP OF THREE PRIMARY MODULES AND THEIR SUPPORTIVE SUBROUTINES. EACH MODULE HAS AN EXTERNAL ENTRY POINT. A DESCRIPTION OF THE WORK PERFORMED BY THESE BIG THREE FOLLOWS:

. I O C .

THIS MODULE RECEIVES CONTROL FROM CALLING HANDLERS TO PROCESS VARIOUS TYPES OF I/O REQUESTS SUCH AS READING, WRITING, CONTROL, STATUS, ETC. SOME REQUESTS ARE IMMEDIATELY PROCESSED BY .IOC. AND OTHERS RESULT IN A CALL TO THE APPROPRIATE I/O DRIVER FOR PROCESSING. IN ADDITION .IOC. IS RESPONSIBLE FOR QUEUING READ, WRITE, AND CONTROL REQUESTS TO BUSY, QUEUING I/O DRIVERS ON THE DEVICE REQUEST QUEUE (DRQ). SELF-QUEUING DRIVERS ARE ALWAYS CALLED AS THEY MAINTAIN THEIR OWN QUEUES. THESE QUEUED ENTRIES WILL BE RESCHEDULED BY .IOC. AFTER THE DRIVER BECOMES NOT-BUSY.

WHEN A BUSY DRIVER COMPLETES AN I/O OPERATION, IT CALLS .BUFR AS ITS LAST ACTIVITY PRIOR TO EXITING (AND THUS BECOMING NOT-BUSY). ONE OF THE OPERATIONS .BUFR PERFORMS IS TO CHECK THE DRQ OF EACH DRIVER THAT COMPLETES AN I/O REQUEST. IF ANY EVENT IS QUEUED, .BUFR WILL SET THE "PENDING DRQ ACTIVITY FLAG (P FLAG) IN THE DRIVER'S EQT FLAGS WORD AND IT WILL INCREMENT THE PENDING DRQ ACTIVITY COUNTER. ALL .IOC. CALLS EXIT THROUGH THE COMMON ROUTINE, RETN. IF THE PENDING DRQ ACTIVITY COUNTER IS NON-ZERO, EVERY EQT IS CHECKED FOR PENDING I/O. WHEN PENDING ACTIVITY IS FOUND, THE P FLAG IS CLEARED AND THE I/O IS INITIATED (SEE .IOC. I/O RESCHEDULING)

.IOC. ALSO MAINTAINS THE SCHEDULING OF DRIVERS WHICH REQUIRE THE USE OF DMA. IF AN I/O REQUEST CANNOT BE HONORED BECAUSE THERE IS NO FREE DMA CHANNAL, .IOC. WILL SET THE "REQUIRES DMA" FLAG (R FLAG) IN THE DRIVER'S EQT FLAGS WORD AND THEN INCREMENT THE PENDING DMA ACTIVITY COUNTER. WHEN AN .IOC. REQUEST EXITS THROUGH RETN, A CHECK IS MADE OF THE PENDING DMA ACTIVITY COUNTER. IF IT IS NON-ZERO AND IF A DMA CHANNEL IS FREE, A ROUND ROBIN SEARCH OF THE EQT'S IS MADE FOR ONE AWAITING DMA USAGE. WHEN ONE IS FOUND, ITS R FLAG IS CLEARED AND THE I/O REQUEST IS RE-ISSUED TO THE DRIVER. (SEE .IOC. I/O REQUEST RESCHEDULING)

.IOC. I/O REQUEST RESCHEDULING

THE DRIVER'S EQT IS CHECKED TO SEE IF IT IS SELF-QUEUING (EQT WORD 0, BIT14). IF IT IS SELF-QUEUING, THE DRIVER'S DRQ HEAD (EQT WORD 4) IS CLEARED AND CONTROL IS RETURNED TO THE CALLER. HENCE SELF-QUEUING DRIVERS MUST NOT ONLY SCHEDULE THEIR OWN I/O REQUESTS THEY MUST ALSO RESCHEDULE THEIR OWN PENDING DMA USAGE. OTHERWISE THE I/O REQUEST IS RETRIEVED AND THEN SENT TO THE DRIVER. NORMALLY THE DRIVER WILL RETURN BUSY AND THEREFOR I/O REQUEST RESCHEDULING IS DONE. IF THE

REQUEST IS REJECTED. THE BUFFER IS MARKED AS PURGED EQT WORD 1 = 140377). THE ENTRY IS REMOVED FROM THE DRQ AND PLACED ON THE CRQ, AND THE DRQ IS SUSPENDED (EQT WORD 1, BIT13=1). IF THE REQUEST IMMEDIATE COMPLETES, THE ENTRY IS REMOVED FROM THE DRQ AND PLACED ON THE CRQ. IF ATTENTION IS REQUIRED (EQT WORD 1, BIT14=1), THE DRQ IS SUSPENDED. OTHERWISE, THE DRQ HEAD IS AGAIN CHECKED FOR MORE PENDING I/O. IF THE DRQ IS NOT EMPTY, THE CYCLE WILL RESTART.

.IOC. REQUEST PARAMETER

A SINGLE WORD PARAMETER IMMEDIATELY FOLLOWING THE JSB TO .IOC. DEFINES THE TYPE OF I/O REQUEST AND DEVICE TO BE USED IT HAS THE FOLLOWING FORMAT:

```
15  14    12   11        6   5             0
Q : FUNCTION : SUBFUNCTION : UNIT REFERENCE
```

Q= QUEUED REQUEST INDICATOR

1 REQUEST IS TO BE QUEUED FOR PROCESSING BY THE I/O DRIVER. THIS INCLUDES SUBSEQUENT QUEUING OF THE OPERATION IN THE COMPLETED REQUEST QUEUE WHEN DRIVER PROCESSING IS FINISHED.

0 NO QUEUING OF THE I/O REQUEST AS IT IS TO BE PROCESSED IMMEDIATELY BY THE DRIVER.

NOTE: MIXED USE OF QUEUED AND NON QUEUED REQUESTS WILL RESULT IN REJECTION OF THE REQUEST BY .IOC. MIXED USE IS POSSIBLE ONLY IF ALL REQUESTS FOR THE DRIVER HAVE BEEN COMPLETED. ONLY READ, WRITE, AND CONTROL REQUESTS CAN BE QUEUED.

FUNCTION=

A NUMERIC VALUE INDICATING A GENERAL CLASS OF PROCESSING. LEGAL VALUES ARE:

0 CLEAR
1 READ
2 WRITE
3 CONTROL
4 STATUS
5 INTERROGATE COMPLETED REQUEST QUEUE
6 SPECIAL

CLEAR - USED TO TERMINATE I/O AND CLEAR THE HARDWARE INTERFACE. .IOC. ISSUES THE CLEAR REQUEST TO THE DRIVER AND AFTER RETURNING IT RELEASES ANY PENDING DMA USAGE AND PURGES ANY OUTSTANDING QUEUED I/O REQUESTS FOR THE DRIVER THIS IS DONE BY REMOVING THE ENTRIES FROM THE DRQ, MARKING THEM AS PURGED, AND PLACING THEM ON THE CRQ.

READ - USED TO REQUEST ANY READ OPERATION. .IOC. PUTS QUEUED READS ON THE DRIVER'S DRQ AND IF THE QUEUE IS EMPTY, IT ISSUES THE READ REQUEST TO THE DRIVER. NON-QUEUED READS ARE IMMEDIATELY ISSUED. FOR ACTIVITY PERFORMED AFTER THE DRIVER RETURNS TO .IOC., SEE "READ/WRITE/CONTROL REQUEST RETURNS".

WRITE - USED TO REQUEST ANY WRITE OPERATION. .IOC. PUTS QUEUED WRITES ON THE DRIVER S DRQ AND IF THE QUEUE IS EMPTY, IT ISSUES THE WRITE REQUEST TO THE DRIVER. NON-QUEUED WRITES ARE IMMEDIATELY ISSUED. FOR ACTIVITY PERFORMED AFTER THE DRIVER RETURNS TO .IOC., SEE "READ/WRITE/CONTROL REQUEST RETURNS".

CONTROL - USED TO PERFORM DRIVER DEFINED CONTROL OPERATIONS SUCH AS A READER/PUNCH FEED REQUEST. .IOC. PUTS QUEUED CONTROL REQUESTS ON THE DRIVER'S DRQ AND IF THE QUEUE IS EMPTY, IT ISSUES THE CONTROL REQUEST TO THE DRIVER. NON QUEUED CONTROL REQUESTS ARE IMMEDIATELY ISSUED TO THE DRIVER. FOR ACTIVITY PERFORMED AFTER THE DRIVER RETURNS TO .IOC., SEE "READ/WRITE/CONTROL REQUEST RETURNS".

STATUS - USED TO OBTAIN THE LAST RECORDED STATUS FOR A DEVICE. .IOC. RETURNS WITH
A = STATUS (EQT WORD 1)
B = TRANSMISSION LOG (EQT WORD 2)

INTERROGATE - USED BY THE COMMUTATOR TO INTERROGATE THE COMPLETED REQUEST QUEUE. FOR A COMPLETE DESCRIPTION SEE "INTERROGATION REQUESTS".

SPECIAL - USED TO PERFORM UNIQUE OPERATIONS WHICH ARE DETERMINED BY THE SUBFUNCTION. THESE OPERATIONS ARE:

0 PRIORITY APPENDAGE - USED TO SUPPLY .IOC. THE ADDRESS OF A MODULE TO BE USED TO DETERMINE THE PRIORITY AT WHICH A COMPLETED I/O REQUEST IS TO BE PLACED ON THE CRQ. IOC. RETRIEVES THE APPENDAGE ADDRESS AND PLACES IT IN EQT WORD 5.

1 PURGE - USED TO REMOVE QUEUED REQUESTS FOR A DRIVER. THE PURGE REQUEST IS SENT TO THE DRIVER AND UPON RETURN, ANY PENDING DMA USAGE IS RELEASED AND ALL PENDING I O REQUESTS ARE PURGED BY REMOVING THEM FROM THE DRIVER'S DRQ, MARKING THEM AS PURGED, AND PLACING THEM ON THE CRQ. PURGE DOES NOT INCLUDE CLEARING THE HARDWARE INTERFACE.

2 RELEASE - USED TO ACTIVATE A SUSPENDED DEVICE. SUSPENSION OCCURRS WHEN A DEVICE TERMINATES AN OPERATION WITH AN INDICATION THAT SPECIAL ATTENTION IS REQUIRED. IF THE DRQ IS NOT SUSPENDED, NO ACTION IS TAKEN IF IT IS SUSPENDED, THE SUSPENSION BIT IS REMOVED (EQT WORD 0, BIT13=0). THEN IF THERE IS AN I/O REQUEST ON THE DRQ, IT IS ISSUED TO THE DRIVER. IF THE REQUEST IMMEDIATE COMPLETES WITHOUT ATTENTION NEEDED, THE DRQ IS AGAIN CHECKED FOR MORE ENTRIES AND THE CYCLE IS REPEATED.

SUBFUNCTION= A NUMERIC VALUE INDICATING OPTIONS FOR THE FUNCTION. THESE OPTIONS ARE ALL DRIVER DEFINED EXCEPT FOR THE SPECIAL FUNCTION AND THE FOLLOWING UNIQUE FUNCTION/SUBFUNCTION COMBINATIONS:

0/0 SYSTEM CLEAR*
4/0 SYSTEM STATUS*
3/0 DYNAMIC STATUS

* MUST ALSO HAVE A UNIT REFERENCE OF 0.

SYSTEM CLEAR - USED TO TERMINATE ALL DRIVER I/O AND PURGE ALL PENDING I/O REQUESTS.

SYSTEM STATUS - USED TO DETERMINE IF ANY DEVICES ARE ACTIVE. .IOC. WILL RETURN WITH THE A-REG, BIT15 = 1 IF AT LEAST ONE DEVICE IS BUSY.

DYNAMIC STATUS - USED TO OBTAIN THE ACTUAL CURRENT DEVICE STATUS. IF THE REQUEST IS NON-PRIORITY (I/O REQUEST, BIT9=0), THE REQUEST IS PUT ON THE DRQ AND THE COMPLETION IS PUT ON THE CRQ. PRIORITY DYNAMIC STATUS REQUESTS ARE NOT PUT ON EITHER QUEUE.

NOTE: BIT 9 OF QUEUED READ, WRITE OR CONTROL REQUESTS INDICATES PRIORITY. SUCH A REQUEST WILL BE QUEUED IN FRONT OF CURRENT REQUESTS.

UNIT REFERENCE= A NUMBER USED TO REFER TO A SPECIFIC DEVICE.

READ/WRITE/CONTROL REQUEST RETURNS

WHEN A CALLED DRIVER RETURNS TO .IOC. AFTER PROCESSING A READ, WRITE, OR CONTROL REQUEST, THE A AND B REGISTERS CONTAIN STATUS INFORMATION WHICH RESULTS IN THE FOLLOWING ACTION:

A-REG	B-REG	MEANING
1	BIT0=0	(DRIVER REJECT RETURN)
1	BIT0=1	(DMA REQUIRED RETURN)
BIT15=1		(IMMEDIATE COMPLETION RETURN
0		(NORMAL RETURN)

QUEUED DRIVERS MAKING QUEUED REQUESTS

(DRIVER REJECT RETURN) - THE DRIVER'S EQT STATUS WORD IS FLAGGED AS REQUEST REJECTED (EQT WORD 1 = 140377 THE REQUEST IS REMOVED FROM THE DRQ AND IS PLACED ON THE CRQ THE DRQ IS THEN SUSPENDED (EQT WORD 0, BIT 13 = 1). CONTROL IS RETURNED TO THE CALLER.

(DMA REQUIRED RETURN) - THIS DRIVER IS SCHEDULED FOR DMA USAGE BY SETTING THE "REQUIRES DMA" FLAG (R FLAG IN HIS EQT FLAGS WORD (EQT WORD 0, BIT 12 = 1) AND THEN CONTROL IS RETURNED TO THE CALLER.

(IMMEDIATE COMPLETION RETURN) - THE REQUEST IS REMOVED FROM THE DRQ AND IS PUT ON THE CRQ. THE DRIVER'S EQT STATUS IS CHECKED FOR REQUIRED ATTENTION (EQT WORD 1, BIT1 IF ATTENTION IS REQUIRED (BIT = 1) THE DRQ IS SUSPENDED (EQT WORD 0, BIT13=1). CONTROL IS THEN RETURNED TO THE CALLER.

(NORMAL RETURN) - CONTROL IS RETURNED TO THE CALLER.

SELF-QUEUING DRIVERS AND NON-QUEUED REQUESTS

(DRIVER REJECT RETURN) - CONTROL IS RETURNED TO THE CALLER AT HIS .IOC. REQUEST REJECT RETURN POINT.

(DMA REQUIRED RETURN) - SAME AS DRIVER REJECT RETURN.

(IMMEDIATE COMPLETION RETURN) - CONTROL IS RETURNED TO THE CALLER.

(NORMAL RETURN) - CONTROL IS RETURNED TO THE CALLER.

INTERROGATION REQUESTS

THE COMMUTATOR USES THIS REQUEST TO INTERROGATE THE COMPLETED REQUEST QUEUE. EACH DRIVER IN THE I/O PROCESSOR HAS AN .IOC INTERROGATION REQUEST CALLING SEQUENCE IN THE COMMUTATOR. THE UNIT REFERENCE NUMBER. CONTAINED IN THE .IOC. I/O REQUEST PARAMETER REPRESENTS THE LOWEST PRIORITY TO BE EXAMINED IN THE CRQ. THE HIGHEST PRIORITY COMPLETED REQUEST OR UNSOLICITED EVENT IS AT THE TOP OF THE CRQ. IF THIS ENTRY'S PRIORITY IS NOT EQUAL TO OR GREATER THAN THE UNIT REFERENCE NUMBER IN THE INTERROGATION REQUEST. .IOC WILL TAKE THE REJECT RETURN. IF THE REQUEST IS SUCCESSFUL. THE ENTRY IS REMOVED FROM THE CRQ AND THE NORMAL RETURN IS TAKEN WITH THE FOLLOWING INFORMATION IN THE A AND B REGISTERS:

	COMPLETED REQUEST ENTRY	UNSOLICITED EVENT ENTRY
A-REG	I/O REQUEST PARAMETER	BIT 15 SYSTEM OVERLOAD FLAG 14-12 ZERO 11-6 TSR LOGICAL UNIT NUMBER 5-0 UNIT REFERENCE NUMBER
B-REG	BUFFER ADDRESS	EVENT DATA

NOTE: THE SYSTEM OVERLOAD FLAG IS SET IN THE EVENT THAT ALL AVAILABLE SYSTEM QUEUE SPACE IS IN USE. THE FLAGGED ENTRY WAS OBTAINED FROM EMERGENCY STORAGE AND INDICATES THAT SOME SUBSEQUENT UNSOLICITED EVENTS MAY HAVE BEEN LOST.

.BUFR

THIS MODULE IS CALLED FROM THE CONTINUATOR SECTION OF ALL DRIVERS WHEN AN I/O REQUEST HAS COMPLETED. IF THE REQUEST WAS NOT QUEUED (PUT ON THE DRIVER'S DRQ), THE DRQ HEAD WILL BE ZERO AND A RETURN TO THE CALLING DRIVER IS IMMEDIATELY MADE. (HOWEVER SELF-QUEUING DRIVERS CAN MAKE USE OF THE CRQ BY FOLLOWING THE STEPS OUT-LINED IN THE SECTION "SELF-QUEUING I/O DRIVERS.) OTHERWISE THE REQUEST IS PLACED ON THE CRQ AND IS REMOVED FROM THE DRQ.

THE EQT FOR THE COMPLETED I/O REQUEST IS NOW CHECKED IF THE EQT INDICATES THAT THE DRIVER IS SELF-QUEUING (EQT WORD BIT14=1), THE DRQ HEAD (EQT WORD 4) IS SET TO ZERO AND CONTROL IS RETURNED TO THE CALLER. FOR A QUEUING DRIVER THE EQT STATUS IS CHECKED. IF THE EQT STATUS INDICATES THAT ATTENTION IS NEEDED EQT WORD 1, BIT14=1), THE DRQ IS SUSPENDED (EQT WORD 0, BIT13= AND .BUFR WILL RETURN TO THE CALLER. OTHERWISE, THE DRIVER'S DRQ IS CHECKED FOR ANY PENDING I/O REQUESTS. IF THE DRQ HEAD IS ZERO, .BUFR WILL RETURN TO THE CALLER. IF THE DRQ HEAD IS NOT ZERO THE "PENDING DRQ ACTIVITY" FLAG IS SET IN THE EQT FLAGS WORD (EQT WORD 0, BIT11=1) AND THE PENDING DRQ ACTIVITY COUNTER IS INCREMENTED. CONTROL IS THEN RETURNED TO THE CALLER.

SELF-QUEUING I/O DRIVERS

A SELF-QUEUING I/O DRIVER IS ONE ASSOCIATED WITH AN EQT HAVING BIT 14 OF EQT WORD 0 SET. SUCH AN I/O DRIVER IS UNIQUE IN THAT NO I/O REQUEST IS EVER QUEUED FOR IT BY QUEUED .IOC. THAT IS, NO DRQ IS MAINTAINED. INSTEAD, ALL I/O REQUESTS ARE IMMEDIATELY PASSED TO THE DRIVER AND IT MUST MAINTAIN ITS OWN REQUEST QUEUING INTERNALLY. THIS DEFINITION IS TO ALLOW FOR DRIVERS WHICH HANDLE MULTIPLE DEVICES, ESPECIALLY MULTIPLE DEVICES SERVICED VIA A SINGLE HARDWARE I/O INTERFACE SUCH AS THE HP 12920 ASYNCHRONOUS MULTIPLEXER. THIS TYPE OF DRIVER DOES NOT WANT THE I/O QUEUING SERVICES OF QUEUED .IOC. SINCE REQUESTS MAY BE DIRECTED TO DIFFERENT SUBUNITS OR SUBCHANNELS.

SELF-QUEUING DRIVERS DO, HOWEVER, HAVE ACCESS TO MOST OF THE CAPABILITY OF QUEUED .IOC. SPECIFICALLY, THE COMPLETED REQUEST QUEUE AND THE UNSOLICITED EVENT FACILITY ARE AVAILABLE. USE OF THE UNSOLICITED EVENT FACILITY REQUIRES NO SPECIAL INFORMATION, BUT USE OF THE COMPLETED REQUEST QUEUE MUST BE PLANNED. QUEUED .IOC. REQUIRES FIVE WORDS OF STORAGE IMMEDIATELY PRECEDING THE BUFFER ASSOCIATED WITH A READ, WRITE, OR CONTROL REQUEST. HENCE, IF A NON-QUEUING DRIVER PLANS TO USE THE COMPLETED REQUEST QUEUE, I/O REQUESTS TO IT SHOULD, IN TURN, EXPECT THE CALLING PROGRAM TO SUPPLY THESE FIVE WORDS. (NOTE THAT INCIDENTALLY THESE FIVE WORDS COULD BE USED BY THE I/O DRIVER TO PERFORM ITS OWN REQUEST QUEUING.) WITH THESE FIVE WORDS AVAILABLE, THE I/O DRIVER CAN HAVE A REQUEST ADDED TO THE COMPLETED REQUEST QUEUE BY PERFORMING THE FOLLOWING STEPS:

1. UPDATE THE EQT WITH ALL NECESSARY STATUS AND TRANSMISSION LOG INFORMATION.

2. COPY THE .IOC. REQUEST PARAMETER INTO THE SECOND WORD OF THE FIVE PREFIX WORDS. (THE LOGICAL UNIT NUMBER IS THE MAIN REQUIREMENT.) THIS SHOULD PROBABLY BE DONE IN THE INITIALOR SECTION OF THE DRIVER WHEN THE I/O REQUEST IS MADE.

3. PLACE THE ADDRESS OF THE BUFFER (NOT THE ADDRESS OF THE FIVE WORD PREFIX) INTO EQT WORD 4 (THE DRQ HEAD).

4. CALL .BUFR IN THE NORMAL WAY.

.UNS.

THIS MODULE IS CALLED BY DRIVERS FOR UNSOLICITED EVENT NOTIFICATION PROCESSING. .UNS. CALLS .GET TO OBTAIN THE FIVE QUEUE CONTROL WORDS NECESSARY FOR COMPLETED REQUEST QUEUE ENTRIES. IF NO MORE SPACE IS AVAILABLE, .UNS. WILL ATTEMPT TO USE THE FIVE WORDS OF EMERGENCY STORAGE. IF THIS TOO IS UNAVAILABLE, THE LOST ENTRY COUNTER IS BUMPED AND THE ENTRY IS IGNORED. ONCE THE ENTRY STORAGE IS OBTAINED, .UNS. WILL BUILD THE CRQ ENTRY AND INSERT IT ON THE CRQ ACCORDING TO ITS COMPLETION PRIORITY. THE COMPLETION PRIORITY CAN BE COMPUTED BY A SPECIAL DRIVER SUPPLIED PRIORITY APPENDAGE (SEE BELOW) OR IT CAN BE SUPPLIED BY THE CALLING DRIVER OR THE DEFAULT PRIORITY CAN BE REQUESTED BY SUPPLYING A PRIORITY OF OCTAL 77. THE DEFAULT PRIORITY WILL BE THE DRIVER'S UNIT REFERENCE NUMBER. AFTER ENQUEUING THE CRQ ENTRY, .UNS. WILL RETURN CONTROL TO THE DRIVER.

PRIORITY APPENDAGE

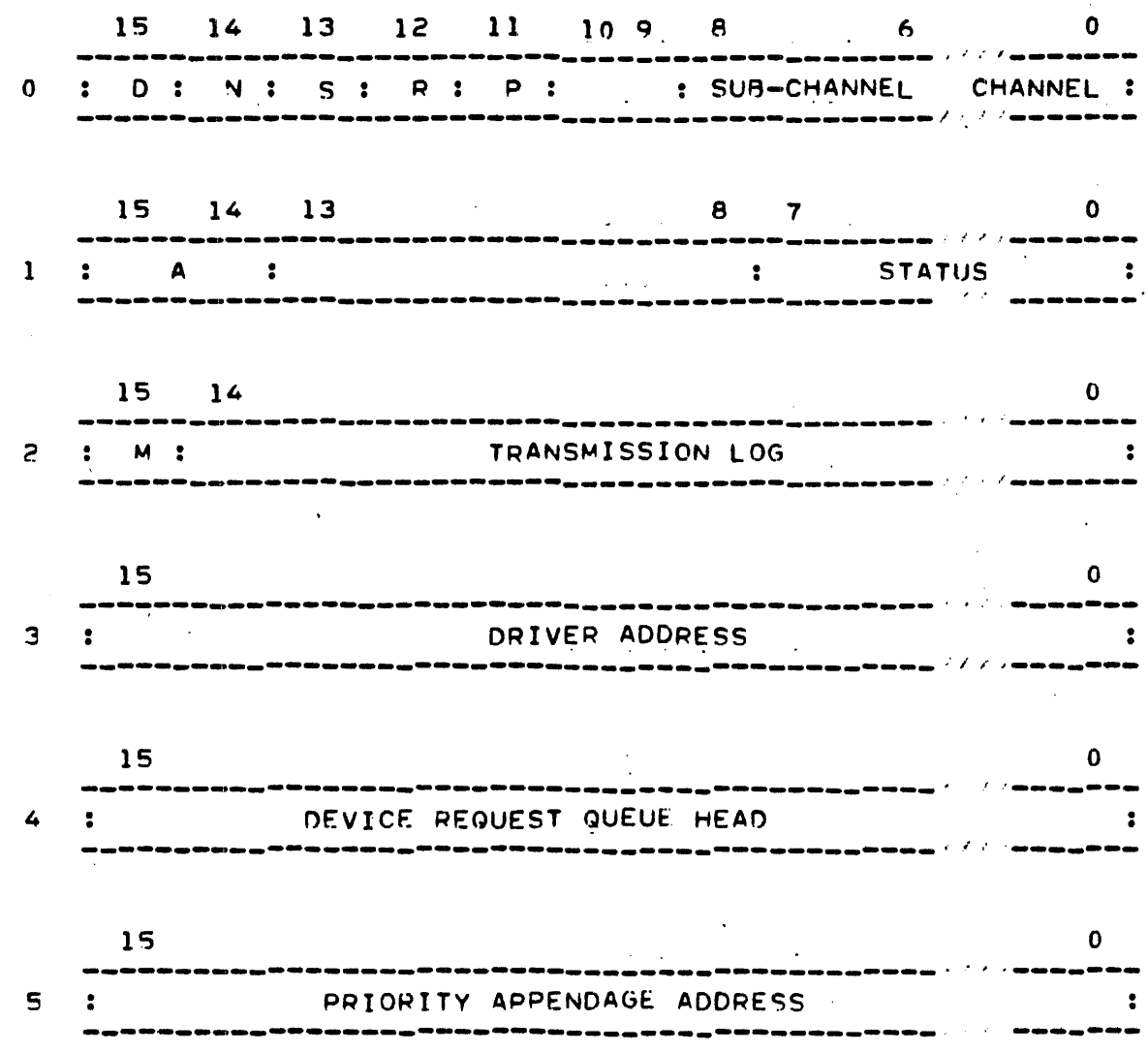
IF A PRIORITY APPENDAGE IS ESTABLISHED FOR A GIVEN DEVICE, IT WILL BE ENTERED AS A SUBROUTINE WHENEVER AN OPERATION FOR THE DEVICE COMPLETES. WHEN THE ROUTINE IS ENTERED, THE A AND B REGISTERS WILL CONTAIN THE FOLLOWING INFORMATION:

A-REG = ADDRESS OF BUFFER
B-REG = ADDRESS OF EQT

THE ROUTINE IS EXPECTED TO RETURN WITH THE A-REGISTER CONTAINING THE PRIORITY AT WHICH THE BUFFER IS TO BE PLACED ON THE COMPLETED REQUEST QUEUE. ZERO IS THE HIGHEST PRIORITY AND OCTAL 77 IS THE LOWEST PRIORITY.

A PRIORITY APPENDAGE IS ESTABLISHED BY MAKING AN .IOC PRIORITY APPENDAGE REQUEST WITH THE ADDRESS OF THE APPENDAGE SUPPLIED AS A PARAMETER. AN ADDRESS OF ZERO WILL DEACTIVATE THE PRIORITY APPENDAGE.

F Q T T A B L E



WORD 0 D - DIRECT MEMORY ACCESS CHANNEL INDICATOR
1 DMA REQUIRED FOR ALL DATA TRANSMISSIONS
0 DMA NOT REQUIRED

N - NON-QUEUING INDICATOR
1 NO DEVICE REQUEST QUEUE IS USED. THE I/O DRIVER ACCEPTS ALL I O REQUESTS IMMEDIATELY
0 A DEVICE REQUEST QUEUE IS TO BE MAINTAINED BY .IOC.

S - DRQ SUSPENSION INDICATOR
1 DEVICE REQUEST QUEUE IS SUSPENDED
0 DEVICE REQUEST QUEUE IS NOT SUSPENDED

R - REQUIRES DMA INDICATOR
1 THE DRQ IS SUSPENDED BECAUSE A DMA CHANNEL IS NEEDED BUT WAS NOT AVAILABLE AT THE TIME THE DRIVER WAS ENTERED
0 THE DRQ IS NOT SUSPENDED FOR DMA

P - PENDING DRQ ACTIVITY
1 THE DRIVER IS NO LONGER BUSY AND SO THE I/O REQUEST ON TOP OF THE DRQ CAN BE INITIATED
0 THERE IS NO PENDING DRQ ACTIVITY

SUB-CHANNEL - THE UNIT NUMBER OF A DEVICE ON A MULTI-DEVICE CONTROLLER

CHANNEL - THE CHANNEL NUMBER (SELECT CODE) FOR THE DEVICE INTERFACE

WORD 1 A - AVAILABLILITY OF DEVICE:
0 AVAILABLE - PREVIOUS OPERATION COMPLETE WITHOUT ERROR
1 AVAILABLE - PREVIOUS OPERATION COMPLETED WITH ERROR
2 NOT AVAILABLE - I/O REQUEST IN PROGRESS
3 PURGED REQUEST - THE DRIVER REJECTED THE LAST I/O REQUEST AND THE REQUEST WAS REMOVED FROM THE DRQ AND PUT ON THE CRQ OR ALL I/O REQUESTS FOR THIS DEVICE WERE PURGED. IN EITHER EVENT, THE CRQ ENTRIES FOR PURGED I/O REQUESTS WILL CONTAIN A COPY OF THIS STATUS

STATUS - STATUS OF THE DEVICE. IF THE AVAILABILITY VALUE IS 3, THE STATUS WILL BE ZERO IF THE REQUEST WAS PURGED OR IT WILL BE 77 IF THE REQUEST WAS REJECTED

WORD 2 M - MODE OF TRANSMISSION
1 BINARY
0 ASCII OR BCD
TRANSMISSION LOG - NUMBER OF CHARACTERS OR WORDS TRANSMITTED

WORD 3 DRIVER ADDRESS - ABSOLUTE ADDRESS OF THIS DEVICE'S I/O DIVER

WORD 4 DEVICE REQUEST QUEUE HEAD - ADDRESS OF CURRENT BUFFER BEING PROCESSED IF QUEUED REQUEST FOR QUEUING DRIVER. THIS ADDRESS MUST BE SUPPLIED BY SELF-QUEUING DRIVERS IF THE REQUEST IS TO BE PLACED ON THE COMPLETED REQUEST QUEUE.

WORD 5 - PRIORITY APPENDAGE ADDRESS - THE ABSOLUTE ADDRESS OF THE PRIORITY APPENDAGE ROUTINE.

```
**********************************************************

  REQUEST DESCRIPTIONS:

      - DATA TRANSMISSION (READ/WRITE)
        FUNCTIONS = 1/2

          (P)         JSR    .IOC.
          (P+1)       (Q.FUNCTION,SUBFUNCTION,UNIT)
          (P+2)       (REJECT RETURN)
          (P+3)       (BUFFER ADDRESS)
          (P+4)       (BUFFER LENGTH)
          (P+5)       (NORMAL RETURN)

      - FUNCTION SELECT
        FUNCTION = 3

          (P)         JSR    .IOC.
          (P+1)       (FUNCTION,SUBFUNCTION,UNIT)
          (P+2)       (REJECT RETURN)
          (P+3)       (NORMAL RETURN)

      - STATUS/CLEAR
        FUNCTIONS = 4/0

          (P)         JSR    .IOC.
          (P+1)       (FUNCTION,UNIT)
          (P+2)       (NORMAL RETURN)

      - INTERROGATION
        FUNCTION = 5

          (P)         JSR    .IOC.
          (P+1)       (FUNCTION,UNIT)
          (P+2)       (REJECT RETURN)
          (P+3)       (NORMAL RETURN)

      - PURGE/RELEASE
        FUNCTION = 6

          (P)         JSR    .IOC.
          (P+1)       (FUNCTION,SUBFUNCTION,UNIT)
          (P+2)       (NORMAL RETURN)

**********************************************************
```

III. Design Structures

Global data structures

In the following sections, precise bit-level definitions of system-wide or global data structures are given. In the strictest sense of the word, only the BCW's are truly global data structures since they are known to virtually every module of the system. None of the other data structures outlined here are known to any modules except the system managers or supervisory components. Hence they are not global. However, because all of these data structures are of major importance to the total system design, they are gathered together here for completeness and clarity's sake.

Buffer Control Words (BCW's)

Nine buffer control words are contained in the first 9 words of each buffer. Words 5 through 8 are required by Queued .IOC. for I/O scheduling, but these words may also be used for interhandler communication whenever no intervening I/O would destroy the contents. Word 9 is used for queueing and is therefore volatile at the interhandler level. However, word 9 may be used to communicate between a handler and a driver. Words 1 through 4 are used for inter- and intrahandler communication and for retaining such fixed information as buffer pool number and size. These words are never destroyed by IOC.

Word 1

Bit(s)	Function
15-11	Handler defined status flags
10	Reject =0 - accepted =1 - message rejected
9	Purge =0 - no purge =1 - message purged
8	Interlock =0 - no interlock =1 - return buffer to sender
7-4	Handler defined status flags
3-0	Buffer Pool number

Word 2

Bit(s)	Function
15-14	(Reserved for future use)
13-8	Data stream id number
7-4	(Reserved for future use)

	3-0	Buffer type indicator =0 - no operation =1 - write =2 - read =3 - enable or start =4 - disable or stop =5 - purge =6 - abort or error =7 - operator command or directive =8 - control operation =9 - start error retry operation =10-14 - reserved for future use =15 - allocated buffer
Word 3		
	Bit(s)	Function
	15	Unused
	14-0	Name of sender*
Word 4		
	Bit(s)	Function
	15-0	Buffer length (words)
Word 5		Used by .IOC. only
Word 6		Used by .IOC. only
Word 7		Used by .IOC. and by handlers to communicate message length
Word 8		Used by .IOC. and for inter-handler status information
	Bit(s)	Function
	15-3	Defined by individual handlers
	2-0	General error type indicator (as used by ASCII files handler, et. al.) =0 no error =1 not ready condition (type 1 error) =2 read check error (type 2 error) =3 data check error (type 3 error) =7 end of file
Word 9		Used by handlers to communicate this information to I/O drivers:
	Bit(s)	Function
	15-14	Driver specific flags,
	13-8	Stream identifiers (to identify

	one of several devices controlled by a driver such as the 12920 multiplexor driver)
7-0	Driver specific flags
	Used by the Queue Manager as the chain or link word when linking buffers onto a work queue

The appropriate BCW words are set by the sending handler before queueing for another handler. It should be noted that .IOC. will not disturb words 1 through 4 of the BCW's.

*The term "name" is used for clarity's sake (See QIT). Here it refers to the address for the QIT entry of the sender.

System Managers

In the following sections are discussed the system managers: Queue Manager, Buffer Manager, and Allocate/Deallocate Manager.

Queue Manager (QM)

Functional description

The Queue Manager provides the communications link between function handlers. Much work created for a function handler is by queue entries, and the Queue Manager manipulates the proper queues to route commands, messages, etc., in buffers to and from function handlers.

Submodule functions

The Queue Manager has several entry points each supplying a service to calling function handlers. The entry points are:

1. .PUTQ which will enqueue work for a given function handler.
2. .PRIQ which is the same as .PUTQ except that the entry is placed first in the designated queue.
3. .GETQ which returns a queue entry for processing to the calling handler and dequeues the entry.
4. .SEEQ which is the same as .GETQ except that the entry is not removed from the queue so that on subsequent .SEEQ. calls the same queue entry is returned to the handler.
5. .PURQ which removes all entries from a queue and returns buffers to the proper location.

All uses of the QM depend on identification of a specific queue. The queue in question will be the one whose QIT entry address appears in register A. This parameter is referred to as the queue name.

Upon receiving control from a .PUTQ or .PRIQ call, the queue manager uses the destination name to access an entry in the QIT. The buffer is then chained into the work queue associated with the QIT entry. If the queue was empty, the prime gate is set to dispatch the function handler that is to process the queue entry. If the queue was not empty, the QM returns to the caller.

Upon receiving control from a .GETQ call, the QM uses the calling handler name to dequeue the first entry in the queue. The remaining queue depth count is placed in register A, and the address of the queue element is placed in register B. In case of a .SEEQ call the action is the same as .GETQ except the queue

entry is not removed from the queue. If the designated queue has been purged, both .GETQ and .SEEQ will return this indication with a zero value in register B. This is to be considered a "logical" work entry. For this reason, .SEEQ will continue to return this indicator until it is relieved via .GETQ. If a queue entry is not present, the prime gate for the calling handler is closed to prevent dispatching the handler again, and the handler is notified of the empty queue condition.

.PURQ uses .GETQ to empty a queue, routing buffers back to the sender (if the BCW word 1 interlock bit is set), or freeing them through the Buffer Manager.

Interfacing

Calling Sequences

1. .PUTQ or .PRIQ
   Calling sequence is:

```
LDA   NAME      Load destination queue name
LDB   BUFAD     Load buffer address for enqueue
JSB   .PUTQ     Call QM
```

2. .GETQ or .SEEQ
   Calling sequence is:

```
LDA   NAME      Load calling handler queue name
JSB   .GETQ     Call QM
JMP   NOQ       No queue entry available
STA   QCNT
STB   BUFAD     Save address of queue elememt
                    returned
```

Note that if a queue entry is not available, a return to the calling handler is to the word immediately following the jump-to-subroutine instruction; when a queue entry is available a skip is taken to the second word following the JSB instruction. On return to the caller, the QM places in Reg A the current depth of the queue (number of work entries currently in the queue).

In case the queue was purged previous to the .GETQ call, return is with a skip, however, register B is zero.

3. .PURQ

Calling sequence is:

```
LDA    NAME        Name of queue to be purged
JSB    .PURQ
```

Local data structures

The Queue Manager maintains all work queues as linked lists chained by address. QIT word one points to the first queue element and the link word of that queue element points to the next queue element and so on. QIT word two points to the last queue element in the linked list. The link word for each element is BCW word 9.

Queue Information Table (QIT)

The Queue Information Table contains one entry for each work queue in the system. Generally, one work queue exists for each function handler, although multiple work queues could conceivably be associated with a single function handler. It is through these queues that one function handler gives work to another via the Queue Manager.

The term "name" is associated with the address of a QIT entry. A name is obtained from the Allocate/Deallocate Manager and is used by a producing function handler to designate the consuming or receiving function handler.

The QIT consists of two segments. First are all of those entries which represent non-allocatable resources. The second segment consists of all those entries which represent allocatable resources. This second segment is ordered by TSB logical unit number (not to be confused with logical unit numbers for ICP .IOC. calls).

Each entry consists of 7 words as follows:

Word 1

Bit(s)	Function
15-0	Address of first buffer on the work queue or zero if no work is queued. This word may be thought of as the queue head.

Word 2

Bit(s)	Function
15-0	Address of last buffer on the work queue or the address of this word if no work is queued. This word may be thought of as the queue tail.

Word 3

Bit(s)	Function
15-0	Queue entry count (number of buffers on the work queue)

Word 4

Bit(s)	Function
15	Allocatable resource flag =0 - not allocated =1 - allocated

14	Queue status flag =0 - queue normal =1 - queue has been purged
13-8	.IOC. logical unit number
7-6	Unused
5-0	TSB logical unit number

Word 5

Bit(s)	Function
15-0	Address of associated handler's prime gate

Word 6

Bit(s)	Function
15-0	Address of associated handler's initialization entry point or zero if no initialization is required

Word 7

Bit(s)	Function
15-0	Device name (class and number) for device associated with this QIT (see Allocate/Deallocate Manager)

Supervisory Components

In the following sections, supervisory components of the system are discussed.

Dispatcher

Functional description

The Dispatcher module is the master interface between queued .IOC., the Queue Manager, and the various function handlers in the IOP. The Dispatcher must perform scheduling of function handlers in two cases, upon I/O event completion and when primed by the Queue Manager.

Interfacing

Function handlers are modules which perform major processing tasks. Usually they are associated with an I/O device. These handlers have two sources of work: completed I/O operations which they have previously scheduled and work given to them by other handlers via their work queue. For this reason, handlers have two types of entry points in addition to an initialization entry point. One is for dispatching when I/O operations complete, and the other is for dispatching when work appears on a work queue. The following naming conventions for these entry points are used where XXX is a basic module name:

XXXHP	-	scheduled or prime entry
XXXHC	-	I/O completion entry

The HP entry point is always entered via JSB. The HC entry point is entered via JMP. All exits from these modules following a dispatch (i.e. entry at either type of entry point) must be done with a JMP to the external symbol .COM. which is the beginning of the dispatcher's commutator.

Processing algorithms

The Dispatcher has two parts: a commutator and an I/O dispatcher.

The commutator consists of several levels of dispatching software. Each level has associated with it a software priority. The levels of the commutator execute in priority order. At the end of the commutator, following the last level, is a JMP back to the top of the commutator. Thus the entire IOP system will loop in the commutator. As work becomes available, it will be dispatched from its appropriate level by leaving this loop.

Dispatching software at a commutator level consists of an .IOC. completed event request and one or more function handler prime gates. The .IOC. call removes completed I/O events from IOC's completed event queue. These are then turned over to the I/O dispatcher. Function handler prime gates are simple two instruction sequences of this form:

```
GATE  RSS
      JSB ENTRY
```

The JSB is to a prime entry point in a function handler. The "GATE" is generally an RSS instruction to prevent entry to the handler. The Queue Manager manipulates this gate, making it either an RSS or a NOP, depending on available work. The gate is opened (NOP) when .PUTQ's or .PRIQ's of work are done and is closed (RSS) when the work queue has been emptied via .GETQ's. Note that due to the use of the JSB, a handler is able to open or close its own gate.

The I/O dispatcher receives control from all commutator levels whenever a completed I/O event is located. Using the logical unit number for the I/O operation, the I/O dispatcher performs a table look-up and dispatches the associated handler at its I/O completion entry point.

Local data structures

Since the system is based on a BCS environment, an .IOC. logical unit number is associated with each I/O device. Also, a specific function handler will exist to control each I/O device. Therefore, association between the logical unit numbers and the function handlers is necessary in order to dispatch the proper function handler to process the completed I/O events for the device. Thus, a table of the function handler I/O complete entry point addresses is kept. The table is ordered by logical unit number and is called the I/O Dispatching Table or IODT.

I/O Dispatching Table (IODT)

The IODT is a table of function handler entry point addresses. These are the entry points for the sections of the handlers which handle completed I/O events. This table is maintained in .IOC. logical unit number order so that an association between I/O events and function handlers can be made. The IODT is used by the dispatcher to enter a function handler when a completed I/O event is found. The IODT format is:

Word 1	Address of I/O completion entry point for logical unit number 0
Word 2	Address of I/O completion entry point for logical unit number 1
.	
.	
etc.	

Since no BCS standard unit numbers (logical unit numbers 0 through 6) are used in the system, words 1 through 7 contain the address of .COM. and yield no effective dispatching.

Buffer Manager (BM)

Functional description

The Buffer Manager is a module which provides a central control point for obtaining and releasing buffers by function handlers. It manages the centralized buffer storage area or pool. The buffer pool is divided into subpools of two general types. Structured subpools are prebuilt with a fixed number of fixed size buffers. These subpools are typically used when hardware or other system requirements demand a guaranteed number of buffers. A single unstructured subpool is also available. From it, buffers of varying size can be obtained. All storage not used by program modules or fixed data structures is available to this subpool. The Buffer Manager has two entry points: .GETB and .FREB. .GETB provides a buffer from a subpool and .FREB allows a handler to release a buffer to a subpool.

When called, the Buffer Manager uses the buffer subpool number to map to a Subpool Control List (SCOL). The SCCL holds necessary information about the subpool, such as what buffers are free and how many requests for these buffers may be outstanding. As well as maintaining the buffer subpools a provision is made to stack .GETB requests for handlers when .GETB requests fail due to buffer unavailable conditions. A Buffer Pending Request Table (BPRT) is associated with each SCOL to record information about handlers with buffer pending requests. These requests will be eventually satisfied when the buffers become available through .FREB. When the buffer manager is entered via .FREB and the SCOL for the buffer being returned shows a handler waiting for a buffer, that handler will be given the buffer via a .PRIQ. In this case BCW word 3 will be cleared and BCW word 2 will be set with a command code of 15. (See Buffer Control Words.)

The BPRT retains information about handlers which request buffers when no buffers are available. The number of slots in the BPRT is equal to the number of work queues in the QIT. Each BPRT slot then corresponds to a specific work queue. The contents of the slot indicate the number of buffers required by the associated work queue. Initially this BPRT has all zero entries. If a handler requests a buffer from the associated subpool and none is available, the following processing occurs:

1. The queue name of the caller is used to generate an index to the associated BPRT slot. This slot is

incremented to show the requirement of the caller for a buffer.

2. A master count of required buffers in the SCOL (word three) is also incremented. If this master count was zero, the SCOL fetch pointer is set to address this newly incremented BPRT slot. The handler's prime gate is close to prevent further processing until the requested buffer is available. A "no buffer" return is effected.

When a buffer is returned to the subpool via .FREB, the following processing occurs:

1. If BCW word 1 bit 8 is set, then the buffer is interlocked. In this case a true .FREB is not done. Rather, a .PUTQ to the name in BCW word 3 is done.

2. If the master count is zero, no further processing occurs.

3. The fetch pointer in the BPRT is used to calculate the queue name index. The queue name is then generated, and the free buffer is assigned to that queue. The count for that queue name is reduced as is the master count.

4. If the master count is now non-zero, the fetch pointer is advanced to the next non-zero BPRT slot. This fetch pointer is always used in a round robin fashion. That is, each BPRT entry will be examined equally often so that a handler requesting multiple buffers will not receive all of them prior to other handlers. This is to prevent lockout conditions by handlers making heavy demands of this subpool.

Interfacing

Calling Sequences

.GETB - obtain buffer

Register A must contain the name of the calling handler. Register B must contain the desired buffer subpool number. The calling sequence is:

```
LDA   NAME
LDB   PULNO
JSB   .GETB
JMP   NOBUF
STB   BUFAD
   .
   .
```

.FREB - free a buffer

Register A must contain the buffer address of the buffer to be released. The calling sequence is:

```
LDA   BUFAD
JSB   .FREB
   .
   .
   .
```

Variable length buffer subpool

Requests for buffers from subpool zero are for buffers of length supplied by the caller. The calling sequence to .GETB for this subpool is modified as follows:

```
LDA    NAME        QUEUE NAME OF CALLER
LDB    LNGTH       LENGTH OF REQUIRED BUFFER
CMB,INB            COMPLEMENT LENGTH
JSB    .GETB       REQUEST BUFFER
JMP    NOBUF       RETURN IF NO BUFFER
STB    BUFAD       NORMAL RETURN
```

It should be noted that this calling sequence to .GETB differs from other calls to .GETB in its use of register B. Normally, register B supplies the number of the subpool from which the allocation is to occur. In this case, register B indicates the length of the buffer required. The negating of this length then implies use of subpool zero. This is the only subpool for which this is true. The length should not include the 9 BCW words. These are automatically supplied by the Buffer Manager.

In order to handle these requests, the buffer manager will use the BPRT for this subpool in a different manner. In the event no buffer is available, the requested length will be saved rather

than a count. The module MEMRY will be used to perform allocations. If the requested length is such that the buffer can never be supplied, control will be returned to the calling handler as in a normal return, and the B register will contain the maximum size of buffer that can ever be requested. The A register will contain a -1 value. Note that the subpool zero BPRT can only retain information about one size of request at a time by a given handler. If multiple outstanding requests occur, they will all be satisfied with the last requested size. It is assumed that this is the mode of use for this subpool.

The use of the SCOL fetch pointer when a buffer is returned to the subpool is as follows:

1. The buffer is returned using MEMRY.

2. The BPRT entry addressed by the SCOL fetch pointer is used to obtain the length of a buffer required by a waiting handler.

3. If a buffer of this length is available, it is given to the requesting handler. The fetch pointer is adjusted, and processing continues at step 2 for other possible buffer allocations. At the first occurance of non-availability, processing ceases.

If any requests for buffers occur when other handlers are waiting on the BPRT, those requests are not satisfied (they are added to the BPRT). The combination of this technique and the one of satisfying BPRT entries in order guarantees that all requests will eventually be satisfied. No lockouts can occur. A master count of pending requests is maintained in the SCOL. This is used to detect pending requests at the time of new requests.

For this subpool, BCW word 4 is especially important. It holds the length of a buffer allocated from subpool zero. It is needed by the Buffer Manager in releasing the buffer to the subpool. Therefore, no user of such a buffer may modify BCW word 4.

Subpool Control List (SCOL)

One Subpool Control List is associated with each buffer subpool and is used by the Buffer Manager to control buffer allocation/deallocation. It is also used in conjunction with its associated Buffer Pending Request Table (BPRT) to control delayed allocation of buffers when requests cannot be immediately satisfied.

A pointer list to the SCOL's is maintained at location PULPT as follows:

```
PULPT DEF   SCOL0
      DEF   SCOL1
      .
      .
      etc.
```

Each SCOL has the following format:

Word 1 *	At load time - contains the address of the 1st word of the buffer subpool. At run time - contains the address of the 1st free buffer (or zero if none are available.)
Word 2 *	At load time - contains the address of the data area (1st word after 9 BCW words) of the last buffer in the subpool. At run time - contains the address of the last free buffer (or the address of this word if none are available).
Word 3	Holds a master count of pending buffer requests.
Word 4	Holds address of next BPRT slot containing a pending request.
Word 5	Holds the address of the 1st BPRT slot.
Word 6	Holds the address of the last word of the BPRT plus one.

* For subpool zero (variable size buffers), words 1 and 2 of the SCCL are initially zero and are not used.

The term "address of buffer" always means the address of the first data word of the buffer. It is assumed that the preceding words are the BCW words. BCW word 9 is used by both the Buffer Manager and Queue Manager to link buffers together on the free list (SCOL) or work queue (QIT entry) respectively.

The following subpools are currently in use:

0 Variable length buffers
1 Control buffers (available for general use)
2 Console message buffers

(Terminal user buffers are taken permanently from subpool zero during system initialization.)

Buffer Pending Request Table (BPRT)

One Buffer Pending Request Table is associated with each SCOL. It retains information about outstanding requests for buffers. Each slot in the BPRT is one word long and is equated with one QIT entry, retaining information about pending buffer requests for that associated QIT name. Hence, the number of slots in the BPRT is the same as the number of QIT entrys. The format of all BPRT's except the one for subpool zero is as follows:

```
Word 1      Number of pending requests for 1st QIT entry
Word 2      Number of pending requests for 2nd QIT entry
.
.
etc.
```

For subpool zero, the BPRT has this format:

```
Word 1      Number of required buffers for 1st QIT entry
Word 2      Size of required buffer for 1st QIT entry
Word 3      Number of required buffers for 2nd QIT entry
Word 4      Size of required buffer for 2nd QIT entry
.
.
etc.
```

Allocate/Deallocate Manager (ADM)

Functional description

The Allocate/Deallocate Manager provides a facility to assign and release handlers and queues for non-shareable resources such as line printers, card readers, and paper tape punches. All such allocation is done in cooperation with the system processor of the HP2000 ACCESS system. This is to prevent uncontrolled interference between the TSB and RJE functions. The ADM is also used by all handlers to acquire queue names for appropriate QIT entries so that .PUTQ's, .GETQ's, etc., among handlers can be done.

Designators are associated with each of these resources. These designators reflect the class or type of device and the number of the device (in case more than one is present). The designators are formed from a two character class designator and a single digit device number. For example:

LP1 - line printer class - number 1
CR0 - card reader class - number 0

A device class and number are represented in one 16 bit computer word as follows:

Bit 15 Zero
Bits 14-10 (1st class character) minus 101 (octal)
Bits 9-5 (2nd class character) minus 101 (octal)
Bits 4-0 (device number) minus 57 (octal)

To request allocation of any device from a given class (no digit supplied), bits 15-5 are as above, and bits 4-0 are zeros. The following designators are currently in use:

Designator	Octal value	Associated queue and handler
IK0	(20501)	Interconnect Kit Handler
MX0	(31341)	Multiplexor Handler
TG0	(46301)	Time Base Generator Handler
SC0	(44101)	Synchronous Communications Handler
CI0	(4401)	Console I/O Handler
HI0	(16401)	Host Inquire Compression Handler
HM0	(16601)	Host Message Decompression Handler
SI0	(44401)	Synchronous Input Queue Handlers(s)
HL0	(16541)	Host List Decompression Handler(s)
HR0	(17041)	Host Reader Compression Handler(s)
HP0	(16741)	Host Punch Decompression Handler(s)
AF0	(241)	ASCII Files Handler

LP0	(26741)	Line Printer Handler(s)
CR0	(5041)	Card Reader Handler(s)
PP0	(36741)	Tape Punch Handler(s)
PR0	(37041)	Phcto-reader Handler(s)
RP0	(42741)	Reader/Punch/Interpreter Handler(s)

Submodule functions

Three entry points are provided:

1. .ALL - Allocate a resource
2. .DEAL - Deallocate or release a resource
3. .FIND - Request information about a resource

An allocation request uses the device class supplied by the caller to request the use of a device of that class from the system processor. If successful allocation occurs, the SP returns the TSB logical unit number of the device to the Allocate Manager which then passes this information on to the requestor as well as the appropriate QIT name. Communication between the SP and the Allocate Manager is via an .IOC. call to the interconnect kit.

A deallocation request uses the QIT name supplied in the .DEAL call to release the device. The TSB logical unit number in the QIT entry is passed to the SP through an .IOC. call to the interconnect kit driver. This shows the deallocation to the SP.

A .FIND request is used to obtain information about a device or class of device. QIT name and/or class information is returned to the caller.

Interfacing

1. .ALL

Calling Sequence is:

```
LDA   DEVCL      Device Class desired
JSB   .ALL
JMP   NODEV      No device available return
STA   LOGUN      Save logical unit #
STB   NAME       Save QIT name
```

2. .DEAL

Calling Sequence is:

```
        LDA    NAME        QIT name
        JSB    .DEAL
```

3. .FIND

Calling Sequences are:

```
        CLA                Zero A-Reg
        LDB    LOGUN       Logical unit #
        JSB    .FIND
        JMP    NODEV       No device by that #
        STA    DEVCL       Save device class
        STB    NAME        Save QIT name
```

or

```
        LDA    DEVCL       Device class & number
        CLB                Clear B-Reg
        JSB    .FIND
        STA LOGUNI         No such device
        STB    NAME        Save QIT name
```

NOTE: The two .FIND calling sequences are required to extract information by supplying either logical unit number or device class and number.

The Allocate/Deallocate Manager depends upon the second segment of the QIT being ordered by TSB logical unit number for all non-shareable devices.

The manager forms an IOC call to the ICK driver of the following form for Allocation:

```
        STA    DEVCL       Store device class
        JSB    .IOC.
        OCT    220XX       Write control
        JMP    NODEV
        DEF    PARM
        DEC    2
* THE ALLOCATED TSB LOGICAL UNIT NUMBER WILL
* REPLACE DEVCL
            .
            .
            .
PARM    OCT    160006
DEVCL   BSS    1
```

For Deallocation the IOC call is:

```
        LDA     LOGUN
        ALF,ALF
        RAR
        IOR     RDR
        STA     PARMD
        JSB     .IOC.
        OCT     220XX
        HLT     0B      (cannct happen)
        DEF     PARMD
        DEC     1
        .
        .
PARMD   BSS     1
RDR     OCT     160007
```

Device Assignment Manager (DAM)

Functional description

The Device Assignment Manager provides the mechanism for establishing and activating linkages between handlers. Requesting handlers obtain permission to activate linkages and the necessary information to do so through calls to the manager. The required information is contained in the Device Assignment Table (DAT) which is created at system configuration time (see LOCAL DATA STRUCTURES). The manager is also responsible for modifying the contents of the DAT as directed by the system operator through use of the Device Assignment (DA) command.

Submodule functions

Three entry points to the Device Assignment Manager are provided:

1. .DA - processes Device Assignment commands.

2. .BSY - processes requests to activate linkages between handlers.

3. .UBSY - processes directives to deactivate linkages between handlers.

A device assignment request (.DA) uses the parameters input with the DA command to modify the contents of the Device Assignment Table. If the host function specified is not currently active, the DAT entry for that function is updated to reflect the new assignment and a skip return is made to the caller. If the host function is busy, the entry is not altered and a reject return is made to the caller.

An activate linkage request (.BSY) uses the device name [1] of the caller to determine if the linkage can be established. This determination is made by first obtaining the device name of the assigned handler from the DAT. If the assigned device is a real device (i.e. card reader, line printer, etc), then the manager attempts to allocate the device. If the allocation is unsuccessful, a reject return is made to the .BSY caller. If the allocation is successful the QIT name and logical unit number of the assigned handler are returned to the .BSY caller. For other devices the DAT is searched for other assignments of the same device. If none exist or if existing ones are marked as not active, a FIND call is made to obtain the Qname and logical unit number of the assigned device which is returned to the .BSY

caller. Otherwise a reject return is made to the caller. Before making a successful return the caller's entry in the DAT is marked active.

A deactivate call (.UBSY) uses the device name of the caller to locate the appropriate DAT entry and mark it as not active. If the assigned device is real, a deallocate call is made before returning to the requestor.

Interfacing

1. .DA

```
Calling sequence is:

LDA DNAME Host function name
LDB HNAME Device name to be assigned
JSB .DA
JMP NAMSG command not appropriate return
 .
 .
 .
```

2. .BSY

```
Calling sequence is:

LDA DNAME   caller's device name
JSB .BSY
JMP PJCT    reject return
STA LOGUN   save assigned LU #
STB QNAME   save assigned Q name
```

3. .UBSY

```
Calling sequence is:

LDA DNAME   caller's device name
JSB .UBSY
 .
 .
 .
```

Note: Reject returns from .BSY and .UBSY will affect the extend (E) register as follows:

```
E=1 - call was for a pseudo device
E=0 - call was for a real device
```

The Device Assignment Manger issues calls to the allocation/ deallocation manger as follows:

For allocation:

```
LDA ANAME    assigned device name
JSB .ALL
JMP NØDEV    device not available return
STA LOGUN    save logical unit #
STB QNAME    save Qname
```

For deallocation:

```
LDA LCGUN    logical unit number
JSB .DEAL
 .
 .
 .
```

For obtaining qname and LU#:

```
LDA ANAME    assigned device's name
JSB .FIND
JMP NODEV    no device return
STA LØGUN    save LU#
STB QNAME    save qname
```

Local Data Structures

Device Assignment Table (DAT)

The Device Assignment .Table contains one 2-word entry for each "Host function" defined at system configuration time. The allowable host function are:

1. HR1 through HR7 for host reading functions.
2. HL1 through HL7 for host line printing functions
3. HP1 through HP7 for host punching functions.

The DAT is prefixed by one word, which is a pointer to the second.

Each entry is definded as follows:

WORD1 BITS-0-15 host function name[1]

```
WORD2 BIT 15 -=1 if function is active
             =0 if function not active
     BITS 0-14 - assigned device name[1]
```

The last two words of DAT are -1.

```
CRn - card reader n
LPn - line printer n
JTn - job transmitter n
JLn - job line printer output n
JPn - job punch output n
```

[1] the function and device names are encoded as follows:

```
BITS 14-10 1st character of device name
           minus 101 octal
BITS 9-5   2nd character of device name
           minus 101 octal
BITS 4-0   device number minus 57 octal
```

System Startup Routine

Functional description

This routine is the starting point for the IOP software. It also contains an entry point for system restart and a panic routine. The purpose of SYSTR is to dispatch all handlers during system startup or restart and then to start the dispatcher. SYSTR's panic routine provides normal responses to the SP in the event of an IOP failure.

Submodule functions

The only difference between the START and RSTRT routines is the setting of location STATE. The routine START sets STATE=0 and RSTRT sets it to 1. Subsequent processing is the same in both cases and consists of searching through QIT for handler initialization entries. Each handler is called by:

```
LDA   STATE
JSB   entry,I
```

After all entries are completed the routines exit by a

```
JMP .COM.
```

PANIC simply provides the necessary responses to the SP so it may be SLEPT in case of an ICP failure.

Interfacing

Calling sequences:

At system startup time SYSTR is entered at location START. Entry is made by setting the START address in the P-reg and pushing .RUN. This routine exits via a JMP .COM.

At system restart time SYSTR is entered at location RSTRT by a JMP RSTRT. Exit is through a JMP .COM.

The entry point PANIC provides for automatic responses to the SP in the event of an IOP crash. The routine is started by entering the PANIC address in the P-reg and pushing START. It will continue to run until HALT is pushed.

Called routines:

During the startup and restart processing, SYSTR calls all device handlers. Control is passed by a JSB to the indicated address with A reg=0 on a system startup and A reg≠0 on a restart.

Local data structures

None (see QIT in Global data structures).

Centralized console output

Description and function

This system service routine (.CCO.) performs the task of transferring a message to the console output queue. Any handler may contain a group of console messages. The handler can then mark a message for output and call the centralized console output routine. This routine will acquire a console buffer, locate the marked message, move the message to the console buffer, remove the mark from the message, and place the console buffer on the console output work queue. In the event that no console buffer is available, the routine leaves the message marked and returns to the calling handler. In this case, the calling handler will eventually receive a console buffer on its work queue. This buffer can be provided to the centralized console output routine to finally dispose of the marked message.

Interfacing

Handlers using this centralized console output routine must construct their message lists similar to the following example:

```
MSG0  ABS   MSG1-MSG0-2
      OCT   x
      ASC   n,message 0 text
MSG1  ABS   MSG2-MSG1-2
      OCT   x
      ASC   n,message 1 text
MSG2   .
       .
       .
MSGL  OCT   0
```

The first word of each individual message contains the length (in words) of the message text. This value must be positive. The word following the last message must be zero. The second word is used to construct BCW word 8 in the console buffer for the message. In this way a handler can present special flags in BCW word 8 to the console output module. Bits 3-0 of BCW word 2 will be set to the value of 1.

A handler marks a message by setting bit 15 of the length word.

Calling sequences are:

1. If no console buffer is supplied:

```
CLA                 (indicates no buffer)
LDB   QN            (calling handler's queue name)
JSB   .CCO.
DEF   MSG0          (address of 1st message in a list)
JMP   X             (return if no console buffer available)
...                 (normal return)
```

2. If a console buffer is supplied:

```
LDA   BA            (address of supplied buffer)
JSB   .CCO.
DEF   MSG0          (address cf 1st message in a list)
NOP                 (always a skip return)
```

.CCO. may be unable to dispose of a message because of unavailability of console buffers. This is ususally nc problem since a buffer will later be made available to the handler that invoked .CCO. However, one problem does exist. A handler may be required to issue a message again, and it may still be marked from some previous call to .CCO. The handler has two options. It may ignore this problem, in which case the duplicate messages are lost. A second option is to temporarily suspend processing if the message is already marked. In this latter case, a console buffer will eventually become available to relieve the handler of the message. Then the message can be remarked and reissued. Obviously, the second option is the better choice when message content may change or when the number of responses shculd equal the number of events which elicit responses.

Algorithm

On entry to this routine, three parameters are provided. The first is the address of a grcup of one or more console messages. The second is the address of a console buffer if cne can be supplied or a zero to indicate that no console buffer is being supplied. Parameter three is the name of the calling handler's work queue.

If no console buffer is supplied, a .GETB is issued to obtain one using the calling handler's work queue name. If this attempt is unsuccessful, the routine returns to the caller.

Next the list of messages is searched for a marked message. If none is found, the console buffer is returned via .FREB, and a return to the calling handler is made. If a marked message is located, the mark is removed, and the message is transferred to the console buffer (i.e. the one just acquired or the one provided as a parameter).

Finally, the console buffer is placed on the conscle output queue. A return to the calling handler is made.

Data ccnversion aids

Several general purpose system services have been designed which aid in data conversion. Since the RJE function is based on the EBCDIC code, much conversion between ASCII and EBCDIC takes place. There are also requirements for conversion between binary integer values and their printable ASCII counterparts. These conversions are handled by the following routines:

.ATOE - converts a string of ASCII characters to EBCDIC. The conversion is done in place. The B register provides the address of the string, and the A register provides the length.

.ETOA - converts a strirg of EBCDIC characters to ASCII. The conversion is done in place. The B register provides the address of the string, and the A register provides the length.

.BTD. - converts an integer value to a string of characters consisting of one space and five decimal numerics. If the integer is negative, the space is replaced by a minus sign. The A register provides the integer value, and the B register provides the byte address of a six byte area to receive the string.

.DTB. - converts a strirg of ASCII decimal numeric characters to an integer value. The B register provides the byte address of the string of characters, and the A register provides the length of the string. The result is returned in the A register. The detection of a length greater than five or any illegal non-numeric character will cause a nc-skip return. A five digit value greater than 32767 will also cause a no-skip return. The normal return is "plus cne."

All of the above routines are used as follows, where "name" is one of the four routine names:

```
        EXT   name
        .
        LDA   (A register parameter)
        LDB   (B register parameter)
        JSB   name
```

Only the .DTB. routine has an alternate return.

.DCOM - decompression service

Description and function

.DCOM provides a data decompression service to expand IBM multileaving compressed records into uncompressed format for unit-record devices. It is typically called by output decompression handlers and is thought to be an RJE related module.

Interfacing

The routine is entered via a JSB with calling parameters in registers and following the call. It will return two values in registers. The calling sequence is:

```
LDA   <from addr>
LDB   <to addr>
JSB   .DCOM
DEC   <buffer length>
```

<from addr> is the byte address of the data to be decompressed. This must be the address of the first SCB in the data.

<to addr> is the byte address of the destination of the decompressed data.

<buffer length> is the positive byte length of the buffer to receive the decompressed record.

Upon return the registers will contain the following:

A-register: Length of decompressed record (0 if EOF record).

B-register: Pointer to next RCB in source buffer.

Note: If there is more data in the compressed record than is available in the buffer, excess data will be truncated. The length returned will never be greater than <buffer length>.

Miscellaneous system service routines

The following service routines are available for use by any modules:

.COMP - compression service

Description and function

.COMP provides a data compression service to compact records according to the IBM multileaving format. It is typically called by input compression handlers and is thought to be an RJE related module.

Interfacing

The routine is entered via a JSB with the parameters in registers and following the call. It returns a value in a register. The calling sequence is as follows:

```
LDA   <from addr>
LDB   <from length>
JSB   .COMP
DEF   <to addr>
```

<from addr> is the byte address of the data to be compressed.
<from length> is the number of bytes of data in the block to be compressed.
<to addr> is the byte address of the destination of the compressed data. The RCB and SRCB are not supplied by .COMP. The caller is assumed to have already inserted these characters. This address must point to the location where the first SCB is to go.

Upon return, the B-register will contain the byte address of the next location in the destination buffer following the compressed data.

Device Table

The Device Table is constructed by the I/O Configurator Program to provide information about non-shareable devices supported by the total system. The format of the table is as follows:

```
.DVTB DEF *+1
      DEC n          Contains # of supported TSE ports
      DEC m          Contains # of following entries
      OCT a,b,c,d,e  Five word entry for 1st device
      OCT f,g,h,i,j  Five word entry for 2nd device
      .
      .
      etc.
```

A total of "m" five word device entries will exist. The format for each is as follows:

Word 1	Device designator or name (Refer to Allocate/Deallocate Manager)
Word 2	Device select code in bits 5-0 Bit 15 is a one if the device is capable of input and bit 14 is one if the device is capable of output. Bit 12 is set if the only output alloved is CTL.
Word 3	Maximum record size for device
Word 4	Zero
Word 5	Zero

A TSB logical unit number is associated with each device. Device Table entry number corresponds tc TSB logical unit number 0 and so forth.

I. Introduction

D.04 Power Fail/Restart Module

This module will store the state of the system in the I/O Processor (IOP) at power failure time. At power recovery time, working closely with all I/O drivers and cooperating with the System Processor (SP), the saved information will be used to restore the software to its cperating state. It is a basic design point of this module that sufficient recovery information is maintained to insure the completion of all scheduled I/O operations. However, due to the impact of power loss at the actual I/O devices, some I/C operations will necessarily ccmplete in error or with data loss. This is a point to be addressed by the individual I/O drivers.

II. Design Overview

Two primary processing sections make up D.04. Both are triggered by an interrupt through interrupt vector 4. The direction of the interrupt (power failure cr power restart) is inspected in order to dispatch the appropriate processing section. Power failure processing saves register contents, I/C device states, and DMA processing states and then halts the CPU. (This routine does take into account the possibility of a power failure during some previous power recovery operation.)

Power recovery processing includes a cooperative restoration of interconnect kit states with the System Processor. Then device states are restored, and I/O driver power fail appendages are given ccntrol to correctly restcre their states relative to their respective devices. Finally, registers and interrupt system states are restored, and the operating system regains ccntrol.

III. Design Structures

Relationship of D.04 to the System Processor

Proper restoration of the system following a power failure requires close cooperatien with the System Processor. Specifically, the correct operational state of the interconnect kit must be maintained, since it is more properly viewed as an extension of each central processing unit than as a peripheral. An auxiliary cable between the two CPU's ensures that each will receive the power loss signal at the same point in time. This technique constrains their power failure interrupts to occur no

more than one instruction apart and insures that the state of the interconnect kit device flags can be saved. However, the status of any DMA transfer over the interconnect kit must be reconstructed. (DMA is inhibited by a power failure interrupt.) Also, power failures occurring during previous power recovery attempts must be carefully noted, so as to prevent improper changes in the recorded system state. Obviously, D.04 processing related to the System Processor is very dependent on the D.61 Interconnect Kit I/O driver. Much of what is stated here is further documented in the D.61 Base Design Specifications.

Each processor must maintain six words of information. One of these, POWFF, is a flag local to the power-fail/power-up routines. It is used to assist in identification of power failures occurring during recovery from a preceding one. RCHNO and SCHNO contain the last words sent (exclusive of a DMA transfer) on the interconnect receive and send channels respectively. These are obviously needed to restore the state of the interconnect. DMAFL is a state variable containing the DMA-on-interconnect status of the processor program. It is used to determine if the power failure disrupted a DMA transfer. The final two words record the length, starting memory address, and direction bit of the most recent DMA transfer on the interconnect. Their existence allows the recovery routine to restart an aborted transfer.

Since the order in which the processors will complete their recovery routines cannot be predicted, two sequencing requirements must be observed. First, each word placed on the interconnect is to be recorded in either RCHNO or SCHNO as appropriate before issuance of the 'STC sc,C' which signals its presence to the other processor. Second, neither processor can proceed to its interconnect DMA routine until after execution of the last instruction affecting the system processor's send flag and I/O processor's receive flag, regardless of which processor will execute the instruction (i.e., one processor may have to wait for its flag to be set by the other processor). This condition is automatically met by the handshake requirements of POS, ALB, and RJE. It is also a fallout of the design for SCI, TCM, and the requests for the device table (SDT) or cold dump core image (KSN). However, use of XRB requires that the I/O processor wait for its receive flag to become set, after transmitting the transfer length word, before invoking its interconnect DMA routine. (Note: POS, ALB, etc. are all mnemonics for commands processed by D.61 in conjunction with the System Processor.)

Special microprogramming for power fail/restart

Three special microprograms will be written to aid I/O drivers in power recovery processing. There are two basic questions which an I/O driver will need to answer in determining correct power recovery action:

1) Has a specific I/C instruction been executed?

2) Is an interrupt section engaged in processing an interrupt at the time of a power failure?

The first question can indicate whether an interrupt was being anticipated at the time of a failure or whether specific I/O interface conditions had been established. The best way to answer this question is to set a software flag which marks the fact that an associated I/O instruction has occurred. However, using the basic instruction set, a power failure could occur between the I/O instruction and the instruction which sets the software flag. This would result in an inability to answer question one easily. For this reason, a microprogram will exist that will incorporate the execution of the I/O instruction with the setting of the software flag. Its calling sequence will be as follows:

```
PFRIO EQU 221E
      .
      .
      RAM PFRIO
IC    STC sc,C
FLAG  OCT 0
```

IO can be any I/O instruction which does not involve A or B. FLAG will be set to 1 by PFRIO. It is assumed that FLAG will be cleared by an interrupt handler or other point where the effect of the I/O instruction has been noted.

The second question must be answered in order to determine the proper action to occur relative to a pending or existing interrupt condition. Through the use of two special microprograms, the value of the interrupt entry point can be used to ascertain the activity of a driver's continuator segment. For purposes of discussion, let I.XX denote an entry point for a driver continuator section. An interrupt typically invokes a JSB to I.XX. Hence if I.XX is assembled as zero, the interrupt will force I.XX non-zero. Thus, a test of I.XX can indicate activity of the segment. The only remaining problem is to incorporate a clearing of I.XX when exiting through I.XX. The two microprograms

perform this function. One, in addition, incorporates the PFRIO microprogram noted above. Calling sequences are:

```
PFREX EQU 223B
      .
      .
      RAM PFREX
      DEF I.XX

PFREI EQU 222B
      .
      .
      RAM PFREI
IC    STC sc,C
FLAG  OCT 0
      DEF I.XX
```

In both cases, control will be transferred to the location whose address is found in I.XX (as in a JMP I.XX,I). Also, I.XX will be set to zero.

I/O driver considerations

During power recovery processing, each I/O driver must be given an opportunity to note the failure and take possible corrective action. D.04 assumes that each I/O driver contains a subroutine entry point, P.XX, in addition to its D.XX and I.XX entry points (that is, its initiator and continuator entry points). After restoring the state of the interconnect kit, D.04 will enter each of the P.XX subroutines. It is assumed that P.XX will do some or all of the following things:

1. Since it is possible for power to fail during a power recovery operation, P.XX may not be able to complete. In addition, should this occur, P.XX will be restarted not continued. For this reason, P.XX must not alter any data or variables which it needs to perform its work.

2. I/O interface control word registers may need to be restored. An example would be the speed and other parameters for the ports on the asynchronous multiplexer interface.

3. Power fail or other status may need to be recorded in the EQT status word. (This is only necessary if needed by other portions of the driver or as a part of status returned in IOC calls.)

4. Due to the fact that the correct nesting of active interrupt sections (continuators) must be restored at power recovery time, D.04 requires that no continuator allow itself to be interrupted by lower priority continuators.

5. When power is restored to the system, all devices flags are set and all control flip/flops are clear. That is, an interrupt is pending. In addition, the state of device flags at power failure time will be saved. Each P.XX appendage must restore its driver relative to these flags:

State of I/O driver interrupt section	State of saved flag	Action to be performed
Idle	Clear	Clear device flag if appropriate. Reissue any I/O instructions which may have been lost (see PFRIO microprogram).
Idle	Set	If an interrupt is pending, issue an STC instruction to allow it. Otherwise, clear the flag. Reissue any lost I/O instructions.
Active	Clear	Allow an interrupt to occur in order to restore the proper nesting of interrupts (see D.04? below). Then clear the device flag. Reissue any lost I/O instructions.
Active	Set	Allow an interrupt to occur. Reissue any lost I/O instructions.

As noted above, in the discussions of the PFREX and PFREI microprograms, the activity of an interrupt section can be determined from the value of its entry point. Testing the saved state of the device flag and allowing an interrupt for proper interrupt nesting are two functions provided by D.04. A subroutine called

D.04? may be called by any P.XX appendage with the select code of a device in register A. D.04? always returns a non-zero A if the saved state of the device flag is set and a zero A otherwise. In addition, if register B is set to a -1 value, D.04? will allow an interrupt on the select code in order to restore its nested interrupt condition.

Power down processing section

The power-down routine distinguishes between four possible cases. 1) If power fails during normal time-sharing, POWFF = 0 and the restart address is not within the recovery procedure. Hardware registers are saved (A, B, P, S, E, O, and F). The states of the interrupt system and all device flags are saved. 2) If power fails while the SP is not time-sharing (during system shutdown, system loading, or after the SP has completed shutdown or failed), the SP will not save its state or attempt to recover. The I/O processor (IOP), however, is normally still active (waiting for the SP to signal resumption of system activity) and does not distinguish this case from the preceding one. 3) POWFF = 1 identifies a power failure from the recovery routine prior to restoration of the interconnect. Since no system activity has occurred on either processor as yet, their current states are ignored and recovery begins anew from the information recorded at the initial failure. 4) If POWFF = 2 or the restart address lies within the recovery routine, then the interconnect was restored but the processor had not yet completed the remainder of its recovery. This differs from case 3 in that the other processor might have completed its recovery and returned to its pre-failure activity. Having done so, it might further have executed an instruction which legitimately altered the state of the interconnect. Each processor is responsible for correctly restoring its send channel (simultaneously restoring the other processor's receive channel). Thus the processor in this situation need only determine if its send channel flag is now set (it could not have been cleared by the other processor) and, if so, overlay the previously recorded state.

The state variable DMAFL allows detection of aborted DMA transfers on the interconnect. It must be set to zero whenever time-sharing begins in order to synchronize the processor states. Each entry to the interconnect DMA routine first saves the transfer parameters and then increments DMAFL to the next odd value. When the processor's hardware DMA flag comes set, it increments its DMAFL to the next even value, executing not less than three instructions to ensure that the other processor has time to complete its current instruction, possibly steal one cycle

for DMA channel 6, and then perform the last cycle for DMA channel 7. (All interconnect DMA activity occurs on channel 7.) Note that this is only a convention based on the assumption that DMA channel 6 can not steal all cycles. If channel 6 did steal all cycles then the required three instruction sequence would fail to insure that the other processor had in fact completed its DMA transfer. (No procedure can guarantee completion if DMA channel 6 performs in this manner.) If the last two bits of both processor's state variables are even (00 vs. 00 or 10 vs. 10) then no transfer is active. DMAFL will be odd (01 or 11) while a processor is in the critical portion of its transfer routine. The other processor's DMAFL will be one less (00 vs. 01 or 10 vs. 11) if it has not reached its transfer routine, or will be equal (01 vs. 01 or 11 vs. 11) if also in its transfer routine, or will be one more (10 vs. 01 or 00 vs. 11) if the transfer completed. The remaining combinations (00 vs. 10 or 01 vs. 11) cannot occur since the processors can never be more than one state apart. This assumes that both processors reset DMAFL to zero at system initialization time.

Possible DMAFL Combinations

SP	IOP	
00	00	Neither processor in its transfer routine.
00	01	IOP doing transfer, SP not yet to its transfer routine.*
01	00	SP doing transfer, IOP not yet to its transfer routine.*
01	01	Both SP and IOP in their transfer routines.*
01	10	SP in its transfer routine but transfer is complete.
10	01	IOP in its transfer routine but transfer is complete.
10	10	Neither processor in its transfer routine.
10	11	IOP doing transfer, SP not yet to its transfer routine.*
11	10	SP doing transfer, IOP not yet to its transfer routine.*
11	11	Both SP and IOP ir their transfer routines.*
11	00	SP in its transfer routine but transfer is complete.
00	11	IOP in its transfer routine but transfer is complete.

* Transfer will be restarted. The SP must set its send channel flag/IOP's receive channel flag according to the transfer direction.

Impossible DMAFL Combinations

SP:	00	01	10	11
ICP:	10	11	00	01

Power recovery processing section

The IOP power-up recovery routine is straightforward except for restoration of the interccnnect. It begims by setting POWFF to 1 .and then re-enabling power-fail interrupts. The WCS module is restored. Then the P.61 interconnect kit driver power recovery appendage is entered. P.61 performs as follows: Since recovery requires an exchange of infcrmation, the IOP must clear its receive channel flag and then wait enough time for the SP to complete its power-down routine, clear its receive channel flag, and transmit its data. The IOP routine does this by incorporating a timing factor into its wait loop. If the loop times out, then the SP must not be active and the IOP should restore itself to the same state as it would be in following reception of an SSD from the SP. After clearing its receive channel flag, the SP will send a word containing zero bits except for the direction bit of the last DMA transfer (1 if from the IOP, 0 if from the SP) in bit 0 and the last two bits of its DMAFL in bits 2 and 1. The recovery routines are now in synchronization. The IOP responds with its most recent DMA direction bit (0 if from the IOP, 1 if from the SP) and the low two bits of its DMAFL in the same format.

At this point both processors have a complete record of the interconnect's state at power-fail. The IOP automatically places the current value of RCHNO on its receive channel and sends it to the SP. If the IOP's DMAFL is odd and the SP's information indicates that the transfer did not complete, the IOP makes a note to reinitialize its DMA routine later in the recovery process. The IOP now waits for the SP to set the IOP's send channel flag. When this occurs, the IOP examines its record of the flag at the time when power failed. If it was clear, the IOP restores the word from SCHNO and issues the 'STC n,C' to transmit it. Finally, POWFF is incremented to 2 since the interconnect is now completely restored (the SP will not finish its recovery until well after the IOP finishes all of the above). The remainder of the IOP routine consists of ensuring that an aborted DMA transfer on the interconnect will be restarted, performing the out-of-line device recovery routines, resetting POWFF to 0 after returning to the power-up routine, restoring the hardware registers and interrupt system flag, and resuming time-sharing.

I. Introduction

The D.43 driver provides timer facilities for both I/O drivers and the user level program.

D.43 is a somewhat unusual driver in that it may be called directly by other drivers as well as by standard .IOC. calls from a user program. For convenience, driver calls will be designated as Line Timer Requests and .IOC. calls as System Timer Requests. The following functions are available from this driver:

1. "Start" - Enables clock interrupts. Must be executed at system startup to enable timing facilities.

2. "Set Timer" - Request by caller to be notified after specified time interval expires.

3. "Cancel Timer" - Cancel previous timing request. Returns residual time interval.

II. Design Overview

Design summary

The driver is a module of the IOP software for the HP2000 ACCESS system. The driver will provide .1 second timing resolution for requests. Hardware requirements are a 2100 computer and time base generator 12539A. The HP2000 ACCESS I/O processor microcode is required.

III. Design Structures

Data structures

The term TQE is used to refer to a Timer Queue Element. The address of an area to be used as a TQE must be passed to the driver as a parameter in all calls. The form of a TQE is shown below. (Only the first two words must be preset by the caller. These words are not disturbed by D.43.)

TQE for Line Timer Request

```
LTQE  DEC   Number of .1 second ticks to time
      DEF   Exit address upon timer completion
      BSS   1     Flag/Link word used by driver
                  Bit 15      =0 - system timer request
```

```
                                    =1 - line timer request
                        Bits 14-0   link address
        BSS    1        Expiration clock value for TQE
```

TQE for System Timer Request

```
STQE    DEC    Number of .1 second ticks to time
        DEF    Exit address for timer completion
        BSS    1        Flag/link word used by driver
                        Bit 15      =0 - system timer request
                                    =1 - line timer request
                        Bits 14-0   link address
        BSS    1        Expiration clock value for TQE
```

Major functional modules

Start

The start request puts the Time Base Generator "on line" and enables all timing requests and must be executed by the user program to enable all timing functions.

The start algorithm is outlined below:

1. Enable I/O board interrupts.

2. Return.

The calling sequence is as follows:

```
JSB    .IOC.
OCT    XX     where XX is logical unit number.
```

Set Timer

The set timer request enables timing to cause asynchronous return after caller specified time interval. The execution sequence for this routine follows:

1. Insert request in timer queue.

2. Return

The calling sequence for this routine is as follows:

1. Line Timer Request

```
ATQE  DEF   TQE
      LDA   ATQE  address of TQE to "A"
      JSB   .LTS. Request Line Timer Set
```

2. System Timer Request

```
      JSB   .IOC.
      OCT   2ØØXX Where XX is logical unit number
      JMP   REJCT
      DEF   STQE  Address of TQE
      BSS   1
```

When the time interval expires for a line timer request, the address of the TQE is placed in register B, and a JSB to the exit routine specified in the TQE is made. The system is disabled during this exit. For system timer requests, the address of the expired TQE is returned through .IOC. as an unsolicited event.

Cancel Timer

The cancel timer request is used to cancel a previous timing request. The driver will return the number of ticks remaining to expiration in word four of the TQE. The execution alogrithm follows:

1. Search timer queue for specified TQE. If not found, take reject address.

2. Remove TQE from queue, and store residual tick count in TQE word 4. Note that normal TQE expiration results in a zero word 4.

3. Return.

The calling sequences for this routine are as shown below.

1. Line Timer Requests

```
      LDA   ATQE  Load A with address of TQE
      JSB   .LTC. Request cancel
      JMP   ERROR Reject address
```

2. System Timer Requests

```
JSB    .IOC. IOC call
DEF    1ØØXX Request cancel
JMP    REJ   Reject address
DEF    STQE  TQE address
BSS    1
```

A reject indicates the specified TQE was not in the timer queue. This may indicate the timer has already expired before the call was made to cancel it.

Power recovery appendage

The P.43 appendage receives control from D.04 at power recovery time. The time base generator decade value is reset. If the I.43 interrupt handler was active, then D.04? is invoked to restore the interrupt condition of the interface. Otherwise, an STC is issued to reactivate the timer.

Time Base Generator Handler

Functional description

This handler (TBGH) provides system timer service through the D.43 time base generator driver. In one sense TBGH is not a true handler since it does not communicate with other handlers through the medium of work queues and the queue manager. Instead its services are utilized through JSB calls. TBGH does use .IOC. to drive the time base generator and is dispatched by DSPCH upon completed event notifications. It passes control to other modules also through JSB calls. Specifically, TBGH provides calling modules the ability to:

1. set a timer
2. cancel a timer
3. provide a routine to be called when a timer expires

Submodule functions

During system initialization TBGH is entered at location TBGHI. A START call to the TBG driver is issued and the routine exits.

For timer requests TBGH is entered at entry point TBGHQ. Processing consists of a SET TIMER request to the TBG driver using the TQE provided by the caller and exit to the caller.

For purging of timer requests the entry point is TBGHP. Processing consists of a CANCEL TIMER request to the TBG driver with the TQE pointer provided and an exit to the caller.

Expired timer request processing occurs at entry point TBGHC. Word 2 of the expired TQE contains the address of a processing routine. It is entered via a JSB call, return is to the location following the JSB. Exit is to the dispatcher.

Interfacing

calling sequences:

TBGH is entered at location TBGHC whenever a timer request expires. The calling sequence is:

```
     LDA   param1      copy of IOC parameter word
     LDB   param2      address of TQE
     JMP   TBGHC
```

The exit procedure is:

```
     JMP   .COM.
```

To queue a timer request TBGH is entered at location TBGHQ by;

```
     LDA   param1      address of TQE
     JSB   TBGHQ
```

The routine exits by:

```
     JMP   TBGHQ,I
```

To purge an outstanding timer request TBGH is entered at location TBGHP by:

```
     LDA   param1      address of TQE
     JSB   TBGHP
```

The exiting procedure is:

```
     JMP   TBGHP,I     if the TQE cannot be located
```

or

```
     ISZ   TBGHP       if the TQE is purged
     JMP   TBGHP,I
```

TBGH is given control during system initialization at location TBGHI by a

```
     JSB   TBGHI
```

control is returned through a

```
     JMP   TBGHI,I
```

called routines:

TBGH makes requests of the time base generator through standard .IOC. calls. Timer completion notifications are processed by a caller provided routine. The routine is accessed by:

```
RAM   LAI+TQEW2   address of service routine to reg A
JSB   A,I         call routine (reg B contains TQE addr)
JMP   .COM.       exit after return
```

Local data structures

Every timer request is communicated to TBGH through a Timer Queue Element (TQE). The TQE is 4 words long and is defined as follows:

```
WORD 1
15                0
  timer interval
```

number of .1 second ticks in requested interval.

```
WORD 2
15                0
  process pointer
```

address of a routine to be given control at time interval expiration

```
WORD 3
15      =0 - system timer request
        =1 - line timer request
14-0    link word
```

```
WORD 4
15      clock expiration value
```

two system link words to be used by the device driver. Following any cancel or expiration, the fourth TQE word contains the number of ticks remaining (0 in the latter case). This is referred to as the interval residual.

I. Product Identification

Microprogram components for HP2000 ACCESS I/O processor

Microprogramming will make a significant contribution to the project. Of primary concern is efficiency. Microprograms are being designed and implemented to reduce processing time for certain frequent high-overhead instruction sequences. A secondary gain will be made in the areas of flexibility and storage utilization.

II. General Design Overview

Microprogram components

Save registers

This microprogram saves the contents of the registers, which is a common activity in interrupt handling, subroutine entry, etc. A, B, E, and O are "pushed" onto the stack addressed by register F. (It is assumed that this stack pointer is initially loaded using an OTA instruction to select code 5.) Pushing is in the direction of higher memory addresses. The contents of A, B, E, and O are not changed.

This microprogram has been assigned to macro 105362. An example of its use follows.

```
SAVE  EQU   362B        DEFINE SAVE MACRO
SUB   OCT   0           SUBROUTINE ENTRY POINT
      RAM   SAVE        SAVE REGISTERS ON ENTRY
      .
      .
      etc.
```

Since a stack mechanism is used for saving of registers, nesting of saves is possible. Note that the save microprogram does no checking for exceeding the size of the stack area. The size of this area must be carefully chosen to handle the maximum possible number of nested saves.

Restore registers

This microprogram complements the save register program. It "pops" A, B, E, and O from the stack addressed by register F.

This microprogram has been assigned to macro 105340. An example of its use follows.

```
RESTR EQU   340B        DEFINE RESTORE MACRO
SAVE  EQU   362B        DEFINE SAVE MACRO
SUB   CCT   0           SUBROUTINE ENTRY POINT
      RAM   SAVE        SAVE REGISTERS ON ENTRY
      .
      .                 SUBROUTINE PROCESSING
      .
      RAM   RESTR       RESTORE REGISTERS
      JMP   SUB,I       EXIT FROM SUBROUTINE
```

Read F register

This microprogram loads the A register with the current content of the F register. This will allow saving the value of F accross power failures.

This microprogram has been assigned to macro 105220. An example of its use follows.

```
READF EQU   220B        DEFINE READ F MACRO
      RAM   READF       LOAD A FROM F
      STA   SAVEF       STORE F VALUE
      .
      .
SAVEF BSS   1
```

Load byte

This microprogram allows loading of bytes (8 bits) into the A register. Character handling, message handling, and buffer unpacking can be facilitated by this microprogram. A byte address is defined as a word address shifted left one bit into bits 15-1. Bit 0 is then used to address byte 0 in the word (bits 15-8) or byte 1 (bits 7-0). The load byte microprogram loads an addressed byte into A register bits 7-0 and clears A register bits 15-8. The byte address is supplied to the microprogram in the B register. The microprogram increments B to the next byte address after loading A.

This microprogram has been assigned to macro 105320. An example of its use follows.

```
LBYTE EQU   320B        DEFINE LOAD BYTE MACRO
      LDB   BYTEA       GET ADDRESS OF WORD
      RBL               DEVELOP ADDRESS OF BYTE 0
      RAM   LBYTE       LOAD THIS BYTE
      STB   NEXTA       SAVE ADDRESS OF BYTE 1
      .                 (A NOW CONTAINS 000377)
      .
BYTEA DEF   DATA
```

```
DATA   OCT    177401
NEXTA  ESS    1
```

Store byte

This microprogram is the store counterpart to load byte. It will be useful in buffer packing and other kinds of character handling. As in the case of load byte, the B register is used to address individual bytes. A register bits 7-0 are stored into the specified location without modifying the accompanying byte in the same word. A register bits 15-8 are ignored, and the contents of A are not altered. B is incremented to the next byte address.

This microprogram is assigned to macro 105300. An example of its use follows.

```
SBYTE  EQU    300B          DEFINE STORE BYTE MACRO
       IDB    BYTEA         GET ADDRESS OF WORD
       RBL                  DEVELOP ADDRESS
       INB                   OF BYTE 1
       LDA    ONE           GET CHARACTER "1"
       SAM    SBYTE         STORE THE "1"
       STB    NEXT          SAVE NEXT BYTE ADDRESS
       .                    (DATA NOW CONTAINS "A1")
BYTEA  DEF    DATA
DATA   ASC    10,A
NEXT   BSS    1
ONE    ASC    1, 1
```

Enque

Many operations in the project involve the use of queues which are maintained as linked lists of elements. For each such queue, a pair of queue control words is maintained. The first word addresses the first element in the queue (the head), and the second word addresses the last element in the queue (the tail). An empty queue is defined such that the head is 0 and the tail points to itself.

The enque microprogram places an element on a queue. Register A addresses the pair of queue control words and register B addresses the element to be added to the queue. (the link word for an element is assumed to be at the address in B minus one.) The microprogram causes a skip whenever the element is added to a non-empty queue. Normally, the enque microprogram places the element at the end of the linked list (via the tail). However, the element may optionally be placed at the beginning of the linked list. This is referred to as a priority enque. Registers A and B are not altered.

This microprogram is assigned to macro 105240. An example of its use follows. Macro 105257 provides the optional priority enque.

```
ENQ     EQU     240B        DEFINE ENQ MACRO
        LDA     QCWA        ADDRESS OF HEAD, TAIL
        LDB     ELA         ADDRESS OF ELEMENT
        RAM     ENQ         ENQUE THE ELEMENT
        JMP     FIRST       (QUEUE WAS EMPTY)
        .                   (QUEUE WAS NOT EMPTY)
        .
QCWA    DEF     HT          ADDRESS OF HEAD, TAIL
ELA     DEF     ELEM        ADDRESS OF NEW ELEMENT
```

Deque

The deque microprogram removes an element from a queue. A skip is performed whenever an element is successfully removed from the queue (queue was non-empty). As in the case of the enque microprogram, register A addresses the head and tail words. The B register is used to return the address of the dequeued element. Register A is not altered.

This microprogram is assigned to macro 105260. An example of its use follows.

```
DEQ     EQU     260B        DEFINE DEQ MACRO
        LDA     QCWA        ADDRESS OF HEAD, TAIL
        RAM     DEQ         GET AN ELEMENT
        JMP     EMPTY       (QUEUE IS EMPTY)
        .                   (QUEUE WAS NOT EMPTY)
        .
QCWA    DEF     HT          HEAD, TAIL ADDRESS
```

In the event the queue is empty (no-skip return), the B register will contain zero.

Load A indexed

This microprogram facilitates access to control blocks, tables, etc. It uses the B register as the base address of an area of storage up to 32 words long. An index value from -16 to 15 may then be added to the contents of B in order to load A from any of these 32 locations. The index value is supplied as a signed four bit number in the low order five bits of the macro which invokes the microprogram. B is not altered by the microprogram.

This microprogram is assigned to macros 105020 to 105057. Note the special use of bits 4-0. An example of its use follows.

```
LAI     EQU     040B        DEFINE LOAD A INDEXED MACRO
        LDB     ATAB        SET BASE ADDRESS
        RAM     LAI+1       LOADS CONTENTS OF W1 INTO A
        .
        .
        RAM     LAI+14      LOADS CONTENTS OF W14 INTO A
        .
ATAB    DEF     W0
W0      BSS     1
W1      BSS     1
        BSS     12
W14     BSS     1
```

Store A indexed

Store A indexed is exactly the store counterpart of load A indexed. Refer to that description for basic details.

This microprogram is assigned to macro 105060 to 105117. Note the special use of bits 4-0.

CRC generation

This microprogram computes cyclic redundancy check characters required in telecommunications error detection procedures. The CRC result is developed by operating on each character of the message with this microprogram. A character is supplied to the microprogram in the A register bits 7-0. The CRC result is developed in the word following the macro invoking the microprogram.

This microprogram is assigned to macro 105150. An example of its use follows. A and B are not modified. The initial value of the CRC result should be zero.

```
CRC     EQU     150B        DEFINE CRC MACRO
        LDA     NEXTC       NEXT CHARACTER TO BE APPLIED
        RAM     CRC         APPLY TO CRC RESULT
RSULT   OCT     0           CRC RESULT
        .                   RETURN TO HERE
```

Word move

Word move transfers blocks of "n" words of storage from one area of memory (source) to another area (destination). The A and B registers supply the source and destination addresses respectively, and the count or "n" is supplied as an inline parameter in the location following the invoking macro. The count must be a positive integer. A negative or zero value will result in an effective NOP. The microprogram is interruptable. For this

reason, it requires a second inline parameter word for interrupt use which must be assembled as zero. A and B will be modified.

This microprogram is assigned to macro 105200. An example of its use follows.

```
WMOVE EQU   200B        DEFINE WORD MOVE MACRO
      LDA   FROM        LOAD SOURCE ADDRESS
      LDB   TO          LOAD DESTINATION ADDRESS
      RAM   WMOVE       MOVE DATA
      DEF   FIVE        ADDRESS OF AMOUNT TO MOVE
      CCT   0
      .
      .
FROM  DEF   *+1
      ASC   5,ABCDEFGHIJ
TO    DEF   *+1
      BSS   5
FIVE  DEC   5
```

Byte move

Byte move is like the word move microprogram with these additional facts. Bytes rather than words are moved. A byte is 8 bits. The source and destination addresses in A and B must be byte addresses. (A byte address is defined as: bits 15-1 contain the word address and bit 0 contains the byte number.)

This microprogram is assigned to macro 105120. An example of its use follows.

```
BMOVE EQU   120B        DEFINE BYTE MOVE MACRO
      LDA   FROM        GET SOURCE ADDRESS
      CLE,ELA           CREATE BYTE ADDRESS
      LDB   TO          GET DESTINATION ADDRESS
      CLE,ELB           CREATE BYTE ADDRESS
      RAM   BMOVE       MOVE THE BYTES
      DEF   NINE        ADDRESS OF AMOUNT TO MOVE
      CCT   0
      .
      .
FROM  DEF   *+1
      ASC   5,A1B2C3D4E5
TO    DEF   *+1
      BSS   5
NINE  DEC   9
```

Following the above execution, the characters "A1B2C3D4E" will have been moved. The byte following the E in the destination area will not have been modified.

Translate

This microprogram performs a translation of a string of characters according to a translate table. If Y is the byte address of the translate table and X is a character (8 bits) to be translated, then:

X <-- character at location Y+X

The A register provides the word address of the translate table, while B supplies the byte address of the string to be translated. An inline parameter provides a positive count of the number of characters to be translated. The microprogram is interruptable. Hence, the inline count is volatile and may be modified. A and B are modified.

The microprogram is assigned to macro 105160. An example of its use follows.

```
TRSLT EQU   160B        DEFINE TRANSLATE MACRO
      LDA   MAPAD       LOAD TRANSLATE TABLE ADDRESS
      LDB   SAD         LOAD ADDRESS OF STRING
      CLE,ELB           MAKE IT A BYTE ADDRESS
      RAM   TRSLT       TRANSLATE THE STRING
      DEC   5           AMOUNT TO TRANSLATE
      .
      .
MAPAD DEF   *-101B
      ASC   5,1234567890
SAD   DEF   *+1
      ASC   3,AIEBCG
```

Following the above execution, the characters "AIEBCG" will have been replaced by the characters "19523G".

Indirect address list

This microprogram generates a list of indirect address pointers useful in accessing unique storage areas. Such a technique has been used in the programming of serially reuseable modules to access individual control blocks. The address in register A is deposited into the location addressed by an inline address list pointer. A is then incremented as is the location pointer, and the process is repeated for a specified number of pointers. The B register is left untouched. A will be updated by the number of pointers generated.

The microprogram is assigned to macro 105000. An example of its use follows:

```
IAL     EQU     000B        DEFINE IAL MACRO
        LDA     ILIST       LOAD ADDRESS OF STORAGE AREA
        RAM     IAL         GENERATE ADDRESS LIST
        DEF     PTRS        ADDRESS OF ADDRESS LIST
        DEC     3           NUMBER OF POINTERS
        .
        .
ILIST   DEF     A
A       BSS     3
PTRS    BSS     3
```

Following execution of the above example, PTRS will contain values equivalent to:

```
        DEF     A
        DEF     A+1
        DEF     A+2
```

Power fail I/O

This microprogram is used to simultaneously execute an I/O instruction and mark the occurance of this execution. This is done by using a macro followed by the desired I/O instruction and a flag word which will be set non-zero by the microprogram. I/O instructions are limited to those not involving A or B.

The microprogram is assigned to macro 105221. An example of its use follows:

```
PFRIO   EQU     221B        DEFINE THE MACRO
        RAM     PFRIO       EXECUTE STC AND FLAG SAME
        STC     10B
        OCT     0           FLAG WORD
```

Following execution of this example, the control bit of the selected interface will have been set (normal STC), and the OCT 0 will have been changed to a one.

Power fail exit

This microprogram allows a module to exit from a subroutine and simultaneously clear the subroutine entry point location. This is especially useful in interrupt handlers to allow for the simple inspection of the value of the interrupt entry point in order to determine if an interrupt is active. This is a necessary decisicn for power recovery processing.

The microprogram is assigned to macro 105223. An example of its use follows:

```
PFREX EQU   223B        DEFINE THE MACRO
      RAM   PFREX       EXIT THROUGH I.10
      DEF   I.10
```

Control is transferred to I.10 indirectly just as in a JMP I.10,I. However, in addition, the I.10 location will be cleared.

Power fail exit with I/O

This microprogram incorporates the actions of power fail I/O and power fail exit. That is, an I/O instruction is executed and flagged, and an indirect exit is made with the entry point being cleared.

The microprogram is assigned to macro 105222. An example of its use follows:

```
PFREI EQU   222B        DEFINE THE MACRO
      RAM   PFREI       EXECUTE THE MACRO
      STF   0B
      OCT   0
      DEF   I.10
```

The STF instruction is executed and the inline OCT 0 flag is set non-zero. Control is given to the location whose address is in location I.10. I.10 is cleared.

III. Design Structures

Microprogram structure

Some of the efficiency of the microprograms is due to the design assumption that the WCS module is module one. It is therefore the macro "entry" module. This can eliminate some wasted microprocessor cycles for jump tables.

In the remainder of this section, each of the individual microprograms is discussed in detail.

Save and restore registers

These microprograms will use the F register as a stack pointer. This has two impacts. First, memory protection can not be used since the two uses of the F register would conflict. Second, no other microprograms may use the F register indiscriminantly. This use of the F register is justified on the basis of efficiency. Memory protection does not appear to be a future requirement, and other microprograms do not seem to be heavily restricted in not using F. It should be noted that the

basic instruction set never modifies F. Should this use of F become a problem, other forms of save and restore can be implemented. However, all other forms will suffer an increase in execution time over the current design.

Save registers

The save microprogram depends on the use of a macro having bit 1 set and bit 0 clear (105xx2 or 105xx6). This is because the IR is used to develop the saved value of the O register. Since saving of the registers requires three core writes (one for A, one for B, and one for E and O), it is desireable to have the microprogram initiate the first write as soon as possible. By assigning the macro 105362 to this microprogram, the primary jump table can be avoided. Therefore, the first microinstruction for this microprogram can initiate the first core write and does not have to be a JMP microinstruction. The microprogram thus fully executes in the minimum number of storage cycle times.

While the core writes for A and B register values are occuring, the values of E and O must be developed. This is done using the CNTR. Bit 1 of the CNTR is set if O is clear by using bit 1 of the IR as a constant (hence the requirement for 105xx2 or 105xx6). Also, if E is set, bit 0 of the CNTR is set. Hence, the net result of this macro is as follows:

1. A is written to the location addressed by F.
2. F is incremented.
3. CNTR is cleared.
4. CNTR(1) is set if O is clear. (Thus, CNTR(1) saves the complement of O.)
5. B is written to the location addressed by F.
6. E is transferred to FLG.
7. F is incremented.
8. CNTR(0) is set if E is set.
9. CNTR value is written to the location addressed by F.
10. E is restored from FLG.
11. F is incremented.

Restore registers

This microprogram performs nearly the inverse operation of the save registers microprogram.

1. F is decremented.
2. A core read for E and O is started.
3. O is assumed set.
4. FLG is set with bit zero from the data read from core. (This is the saved value of E.)

5. F is decremented.
6. A core read for B is started.
7. E is loaded from FLG.
8. O is cleared if bit one of the data from (2) is set. (This is the complement of the saved value of O.)
9. B is loaded from the data read from core.
10. F is decremented.
11. A core read for A is started.
12. A is loaded from the data read from core.

Read F register

1. F is transferred to A.

Load byte

This microprogram must initially have a default NOP in the IR. Therefore, IR is cleared by the JMP in the primary jump table This microprogram is also used as a subroutine by some other microprograms. In this case, the JSB clears the IR.

1. A 32 bit logical right shift of B and A places B(15-1) in S1(14-0) and B(0) in A(15). (Address of word containing byte in S1 and byte number in A(15).)
2. A read of the word addressed by S1 is started.
3. If A(15) is clear, an ALF instruction is built in the IR.
4. The data read is placed in Q as operated on by either two NOP's or two ALF's. (Thus the desired byte is placed in Q(7-0).)
5. B is incremented.
6. Q is transferred to A with A(15-8) cleared.
7. The microprogram either returns to byte move or exits phase three.

Store byte

This microprogram must initially have a default NOP in the IR. Therefore, the IR is cleared by the JMP in the primary jump table. This microprogram is also used as a subroutine by some other microprograms. In this case, the JSB clears the IR.

1. B(15-1) are shifted to S1(14-0). FLG is set with B(0). (Address of word containing byte to S1 and byte number to FLG.)
2. A read of the word addressed by S1 is started.
3. B is incremented.

4. If FLG is clear, an ALF instruction is built in the IR.
5. The data read is placed in Q as operated on by either two NOP's or two ALF's. (Thus the byte to be saved is moved to Q(15-8).)
6. Q(7-0) are cleared and Q is moved to S2.
7. A(7-0) and S2(15-8) are merged and positioned either with two NOP's or two ALF's.
8. A core write is initiated.
9. The merged value is written to core.
10. The microprogram either returns to byte move or exits phase three.

Enque

1. The address of the head pointer is placed in S2.
2. The CNTR is set from the IR for use in detecting the priority enque request.
3. The new element link word address is placed in S3. If the request is not priority, S2 is changed to the tail pointer.
4. A read of the appropriate pointer is started using S2.
5. The tail pointer address is developed in S1.
6. Q receives the old element address just read. S4 receives the proper link word content for the new element.
7. The carry flip/flop is set if the request is priority and the head was clear.
8. A core write of the head or tail pointer is started.
9. The contents of B are written into the pointer. (Tail or head now addresses new element.)
10. If the request is not priority, then S1 is changed to rewrite the old element link rather than the tail.
11. The link word of the new element is written from S4.
12. If a priority enque and the list was not empty, the remainder of the microprogram is a NOP.
13. B is written into the core location addressed by S1. (Old element now points to new tail element or for priority with an empty list, the tail points to the new element.)
14. The carry flop is set for non-priority requests to reflect the emptiness of the list.

Deque

1. The head pointer is read into B.
2. If B is clear, an exit is made. (P is not incremented. The list is empty.)

3. The link word of the dequeued element is develcped in S1 and is read.
4. P is incremented. (The list is not empty.)
5. The link word value just read is placed in S4.
6. The head is rewritten with the value in S4. (Head pointer now addresses new head element.)
7. If the head pointer is going to zero, the tail pointer is rewritten to address itself.

Load and store A indexed

These microprograms are very similar and will be described together.

1. Bits 4-0 of the instruction register represent a signed index to be added to B. Due to the function of the ADR S-bus micro-order, bits 9-0 are added to B first. The result goes to Q. If X represents bits 9-0 of the instruction register with bits 4-0 clear, then this step produces: Q=B+X+N where N is the desired index.
2. X must be removed. Two subtraction steps now occur to remove X from Q. The final result (B+N) goes to S2. The value X is based on the macro assignment for the microprograms.
3. For load, a core read is started to the address in S2. For store, a write is started.
4. A is appropriately loaded from or stored to the memory location in question.

CRC generation

The CRC computation is a division of a message by an error detecting polynomial. The remainder of the division is the CRC result. By beginning with a clear CRC result and applying each successive bit of the message to the division process, the CRC result may be computed. This microprogram handles a character of the message and applies each of the 8 bits of the character successively. The polynomial involved here is:

$$X^{16}+X^{15}+X^{2}+1$$

This is the standard CRC polynomial for IBM binary synchronous communications. In these communications, the CRC result is maintained with its low order bits in high order positions of the computer memory word. The division is carried out with 1 bit shifts and modulo 2 addition (an exclusive or).

For more details on the subject of error detecting codes and on the technique used by this microprogram, refer to the following publication:

James Martin, Teleprocessing Network Organization, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1970, pages 76-95. (especially pages 85 and 91-93)

1. A and B are saved in S4 and Q respectively.
2. CNTR is set for an 8 cycle loop using a constant value of 10 (octal) from the IR. (Hence, the requirement for a macro assignment of 105150.)
3. A constant (polynomial) and an RAR instruction are built.
4. The supplied character is moved from A to B.
5. The current CRC result (addressed by P) is read and rotated right into S2. A is cleared.
6. B and A shift right together. B(0) becomes A(15). S2 is then exclusive or'ed with A. This applies the constant (or 1) part of the polynomial. Result goes to A.
7. If the result of 7 is a 1 in A(15), then the remainder of the polynomial is applied.
8. S2 is set with the new rotated value of A.
9. CNTR will cause cycling on steps 7 through 9 for each of the 8 bits in B. A is cleared on JMP's to step 7.
10. The CRC result is rewritten to core.
11. P is incremented. A and B are restored.

Word move

Normal processing

1. The count addressed by the inline parameter is read and the result is loaded into Q (-count-1). If the result is not a positive value, the microprogram exits at step 10.
2. The destination address in B is moved to S2.
3. A source word is read as addressed by A.
4. A and B are incremented.
5. S3 (or S4 if used in control store module 0 or 2) receives the source word content.
6. The count is decremented in Q. When zero a skip occurs to an exiting JMP to step 10.
7. The destination word write is started using the address in S2.

8. The source content in S3 (or S4) is written to the destination. If an interrupt is pending, proceed at step 11.
9. B is transferred to S2 as the new source address. The microprogram continues at step 3.
10. P is incremented past the inline parameters and end of phase is set.

Interrupt processing

11. The count in Q is decremented, and the microprogram continues at step 10 if the count goes to zero.
12. If more moving is required, the current count is rewritten to the location addressed by P and P is decremented to cause the macro to be re-executed.

Byte move

This microprogram uses the load and store byte microprograms as subroutines.

Normal processing

1. The count acquisition code in the move words microprogram is used as a subroutine to obtain the count.
2. B is saved in S4 as the destination address. A is moved to B as the first source address.
3. Load byte is used to obtain the next byte.
4. S3 retains the next source address returned by load byte in B.
5. B is restored with the next destination address from S4.
6. S4 is used to retain Q accross store byte which also uses Q.
7. The store byte is performed and Q is restored. The next destination address provided by store byte in B is saved in S4. If an interrupt is pending, proceed as in move words.
8. B receives the next source address saved in S3.
9. The count is decremented, and the microprogram exits if it goes to zero. Otherwise proceed at step 4.
10. P is advanced beyond the inline count. End of phase is set.

Translate

This microprogram uses the load and store byte microprograms as subroutines.

Normal processing

1. The count is read into Q in two's complement form. If the result is not negative, an immediate exit occurs.
2. The byte address of the translate table is created in S4.
3. The byte addressed by B is loaded (function byte).
4. The address of this byte is retained in S3.
5. The function byte is added to S4 to generate the address of the object byte within the table.
6. The object byte is loaded.
7. The address of the function byte is restored to B from S3. Q is saved in S3. (Store byte uses Q.)
8. The object byte is stored over the old function byte.
9. S3 is incremented into Q. (Count check.) If a zero results, an exit is begun.
10. If an interrupt is pending, interrupt processing begins.
11. Processing continues at step 3 until the count goes zero when P is incremented to skip the inline count. Then end-phase is set.

Interrupt processing

12. A is restored. The positive counterpart of Q is rewritten as the inline count.
13. P is decremented for reinvocation of the microprogram.

Indirect address list

1. The inline parameter, which is the address of the indirect address list, is read from memory.
2. P is incremented past this inline parameter. The B register is saved in register S1.
3. The data read in 1 above is now moved to register B.
4. A memory read is started to obtain the inline parameter which is the address list size. This information is transferred to the Q register, and P is incremented past the parameter.
5. A memory write is started using the address contained in the B register. This is the address of a cell in the indirect address list. The contents of the A register are deposited into this location.

6. Both A and B are incremented.
7. The count is decremented in Q. When this count reaches zero, B is restored from S1 and end phase is set. Until this occurs, processing continues at step 5.

Power fail I/O

1. The first inline parameter is read using the P register. This is the I/O instruction which is to be executed. P is incremented to skip this parameter, and the IR is loaded with the instruction as read from memory.
2. A memory write is started to the next inline parameter. This is the flag which will mark the occurance of the execution of the I/O instruction. A constant one is deposited into this location.
3. P is incremented to skip the inline flag. Then the IOG decoder is enabled to execute the I/O instruction in the IR.
4. If the microprogram is in use by power fail exit with I/O, then a return from subroutine is effected. Otherwise, this is a nop and end phase is set.

Power fail exit

1. The inline parameter addressed by the P register is read from memory. This is the address of the word which contains the return address. The contents are deposited into S2.
2. A read is now started using the address in S2. This will obtain the return address to which control is to be transferred. The value in that location is deposited into P.
3. The location addressed by S2 is now written with a zero. This marks the fact that the exit has been taken and that no processing is active in the associated module.
4. End phase is set.

Power fail exit with I/O

1. The power fail I/O microprogram is invoked as a subroutine.
2. The microprogram then falls into the power fail exit microprogram.

IV. Implementation

All microprograms are tested and debugged using WCS. However, it is expected that in the final HP2000 ACCESS systems, PROMs will be burned with the microprograms described herein. These PROMs will become a standard part of the CPU for the HP2000 ACCESS I/O processor.

SECTION III

TERMINALS

HEWLETT-PACKARD 2000 SYSTEM I/O PROCESSOR

TERMINALS

CONTENTS

1. D.61 INTERCONNECT KIT DRIVER
2. ICKH INTERCONNECT KIT HANDLER
3. MUXH MULTIPLEXOR HANDLER
4. D.51 MULTIPLEXOR DRIVER

I. Introduction

The D.61 Interconnect Kit Driver is designed for use in the I/O Processor of the Hewlett-Packard HP2000 ACCESS Time Share Basic System. It will provide buffering of input and output from/to the System Processor and proper channelling of control-type information throughout the system.

II. Design Overview

Design assumptions

The hardware required for operation of the program includes a 2100A processor used as the I/O Processor on a HP2000 ACCESS system. Also required is a processor interconnect kit (part number 12875B).

The software environment is assumed to contain the HP2000 ACCESS I/O processor microcode. Also assumed are special purpose HP2000 ACCESS driver extensions in the D.51 multiplexer driver.

Design summary

The design approach uses as a starting point an analysis of the functions required of the I/O Processor and the buffered I/O requirements of Queued .IOC.

Many of the functions requested of the IOP are of an asynchronous nature, unrelated to activity on the IOP. An example is the PHONES command execution. To optimize performance, this driver will process these by dispatching appropriate appendages in co-existing drivers and the base level program. Base level appendages are also occassionally dispatched using the technique of unsolicited event notices.

Another subset of the requirements for the driver is a buffered I/O interface with Queued .IOC. This requires synchronization of an input from the System Processor with .IOC. READ calls and output to the SP with .IOC. WRITE calls. It is noted in the SP connection protocol that all WRITE calls must be immediately followed by READ calls, so a special WRITE/READ operation is provided.

Major modules

Command Dispatcher - This module will interpret commands from the SP and pass control to the module/routine responsible for execution of the command.

External Appendages - This "module" consists of a collection of routines scattered throughout the system responsible for execution of asynchronous control/information commands from the system processor. Since these routines are coded in different, external modules, no specific outline of their functioning will be made here. However, Table 3 contains a list of the routines, entry parameters, and expected returns to the SP where appropriate.

Input Buffering - This module processes POC and POS commands to buffer data from the SP to the IOP. It also notifies the SP of Buffer Full and Buffer Empty conditions.

Data Output - This service module is used to output commands to the SP.

Output Transmission - This module handles transmission of data to the SP. It also is responsible for execution of Backspace and Save/Restore Buffer Pointer commands.

READ Interface - This module interfaces an .IOC. READ request with the Input Buffering module. It must reference and adjust table information to reflect input buffer availability.

WRITE/READ Interface - This module interfaces an .IOC. WRITE/READ request with the Output Transmission module. It also notifies the SP of data availability.

WRITE CONTROL Interface - This module interfaces an .IOC. WRITE CONTROL request with the SP. Control information from the user program is passed directly to the SP.

III. Design Structures

Major data structures

One of the key features of the driver is simultaneous control of up to 32 data streams through a single .IOC. logical unit number. Implementation of this feature requires control tables in the driver for each of the 32 streams. This table, called a Logical Unit Table (LUT) is illustrated in Table 1.

Major functional modules

The Command Dispatcher is entered to obtain commands from the SP and transfer control to the appropriate routine. It will relinquish control via a JSB with the following parameters set:

A-register: Logical Unit Number
B-register: Copy cf SP Command

The algorithm used is as outlined below.

1) Use bits 15-13 as offset into major command JMP table. If 15-13 ≠ 7, goto routine via JSB. On return goto step 3.

2) If 15-13 = 7, use bits 4-0 as offset into micro-command JMP table. Goto routine via JSB.

3) On return clear flag to allow further SP interrupts. Terminate interrupt processing.

See table 1 for a list cf SP commands and where control is transferred for the various commands.

External Appendages are a collection of device specific routines used to quickly execute device specific SP commands. An example of such a command is ABT, which requires the MUX driver to change status of the user. No description of the processing requirements for each External Appendage is given here, but is given within the Base Design Specification for the module in which it is coded.

The Input Buffering module is used to receive characters from the SP and assemble them into a message buffer. It obtains control to process the following SP commands: POC, KTO, IWT, ALI, STE, OWT, IBA, RLB, UIR, UNR, TPO, ABT, and POS. It may also transmit the following commands to the SP: BFL. And it may ultimately pass control to the terminator section of the READ Interface module. References to the LUT use the component name as shown in Table 1. Implicit in every LUT reference is subscripting by the appropriate device number. The Input Buffering module is also responsible for generating all unsolicited event notices. SP commands which unconditionally generate an unsolicited event notice are INI, SSD, HUU, ULO, and NUC. The notification and interpretation of these commands are included as footnotes to Table 2. Buffer Full, SP Input Killed, User Running, User Not Running, and User Aborted status indications may also occur in unsolicited event notices if no processing is in progress when these status' are obtained.

Data Output consists of the single routine OMTOM. This module is entered for all IOP to SP command transfers. Upon entry, the command should be in the A-register. Enter via a JSB OMTOM.

The Output Transmission module has the responsibility of feeding the SP from a WRITE buffer one character at a time. It obtains control from the Command Dispatcher for the following commands: FNC, SBP, RBP, BKS. It does not transmit any commands to the SP.

The READ Interface module is responsible for initiating READ requests from .IOC. and queuing READ completions through .IOC. The user interface is through two .IOC. calls, as described below.

READ REQUEST

```
JSB .IOC.
OCT 100XX   (XX is logical unit number)
Reject Address
Buffer Starting Address
Buffer Length
```

The high-order byte (bits 12-8) of the word immediately preceding the buffer is assummed to contain the logical stream number (sub-unit number) for the request. The driver will reject if a READ request is already pending. Status returns for the READ are given in Table 2. It should be noted that these status returns, as everything else in this driver, are specific to the HP2000 ACCESS system.

The algorithms for this module consist of one for READ request execution and one for READ completion processing. READ request execution will adjust pointers in the Logical Unit Table for the Input Buffering module and if necessary transmit a BFE command to the SP to re-enable data transfer. READ completion processing will copy current status to the equipment table log and queue the input by a JSB .BUFR. No special handling is required by this routine.

The WRITE/READ Interface is used to request data output to the SP, and

turnaround of the device upon WRITE completion to READ status. The form of this call is as follows:

WRITE/READ REQUEST

```
JSB .IOC.
OCT 2ℓ2XX
Reject Address
Buffer Start Address
Buffer Counts          (high byte is
                        write count, low
                        byte is read count)
```

Again, the high-crder byte of the word immediately preceding the buffer is assumed to contain the logical stream number for the request. Further, bit 13 shculd be set if a parity error occurred and bit 14 should be set if a character was lost. Both should not be set, but parity errors will override lost characters. The driver will reject if a WRITE request is already pending. Status returns for the WRITE are given in Table 2.

The algorithm for the WRITE interface is very straightforward, as outlined below.

1) If WRITE/READ request already queued, reject.
2) Store WRITE buffer information in LUT for Output Transmission module.
3) Transmit HVL command to SP.
4) Exit.

Upon WRITE completion initiate a READ operaticn by direct entry to the READ interface.

The WRITE CONTROL Interface module is used to transmit the following "special" commands to the SP: ABR, ETO, and UHU. These may all be sent via the follcwing .IOC. call.

WRITE CCNTROL REQUEST

```
JSB .IOC.
OCT 2ℓ4XX
Reject Address
Buffer Address
DEC 1 (ccnstant)
```

A reject will never occur. The "buffer address" should point to one word, consisting of the command to be sent to the SP. Upon return, the request will be completely executed; hence no completion notice will be given.

The execution algorithm consists of loading the command to transmit, calling the Data Output routine, then returning. For reference, ABR is an Abort request, code OCT 16UUØØ (UU IS Logical Stream number); ETO is ENTER time out, OCT 16UUØ3; UHU is User hung up, OCT 16UUØ4; and ADR is Allocate Device for RJE, OCT 16UN06 where UN is the logical unit number.

Power recovery processing

Much information about restoration of the interconnect kit following a power failure is outlined in the D.04 Power Fail/Restart module specifications. (See "Relationship of D.04 to the System Processor.") There can be found details on the data exchanged between the IOP and SP during power recovery as well as a description of interface flag restoration.

Table 1
LOGICAL UNIT TABLE DEFINITION

.IB1.	Input Buffer Pointer
.IL1.	Input Buffer Length
.ISTA	Input Status *
.ISAV	Input Save Word
.OB1.	Output Buffer Pointer
.OL1.	Output Buffer Length
.CSTA	Output Status **
.OSAV	Output Save Word
.OSAL	Output Save Length
.ICP	Input Current Pointer
.ICL	Input Current Length
.OCP	Output Current Pointer
.CCL	Output Current Length

* .ISTA bits have the follcwing definitions:

7-0	Copy of EQT Status for stream. Cleared by operation completions; if any bits set on READ call, operation terminates immediately.
8	"BFE" must be sent to start read.
9	Input character saved in .ISAV.
10	RLB received for message.
11	Character 2 saved in .ISAV.
12	IBA indicated to base program.

** .OSTA bits have the following definitions:

15	BPSAV - Buffer Pointer is saved.
14	WRITE/READ operation in progress.

Table 2
EQT STATUS BIT DEFINITIONS

7	6	5	4	3	2	1	0
Buffer Full	Start ENTER Timing*	Allow Mux Input	SP Input Killed	User Running	User Not Running	Tape Mode On	User Aborted

Note: A bit on indicates condition is true. Bit off has no meaning.

The definitions of the above and associated actions required are listed below:

Buffer full -- This indicates a READ operation has terminated under conditions requiring another READ to follow. Such conditions are READ buffer filled or OWT state entered. The next operation scheduled should be a READ operation. It may also occur in an unsolicited event notice if the SP attempts to transfer data while no input buffer is assigned.

Start ENTER timing -- This bit is set if ENTER timing is requested by the SP at the termination of a READ operation. It cannot occur unsolicited. The ENTER time parameter, in seconds, will be returned as the last character in the buffer. After receiving this notice a MUX READ should be initiated and the ENTER timing should be started.

Allow MUX input -- This indicates the SP has requested a MUX READ operation. It cannot occur unsolicited. A MUX READ should be started.

SP Input killed -- This occurs as the termination of a READ operation and indicates the data should be purged instead of output to the MUX. It may occur unsolicited, also, in which case current MUX output should be purged, followed by a MUX READ.

User running -- This may occur at the completion of an operation. It indicates a state transition from conversational to RUN mode.

User not running -- This also may occur at the completion of an operation. It indicates a state transition from RUN to conversational mode.

Tape mode on -- This may occur only in the completion of a READ operation. It indicates a state transition to TAPE mode, and processing should be adjusted accordingly.

User aborted -- This is set when an ABORT is indicated by the SP. It implies the User is Not Running; it does not purge active I/O on the device. It may occur at operation completion or as an unsolicited event.

Unsolicited event extensions

In the case of an unsolicited event notice, the above status will be returned in bits 7:0 of the B-register. Additional status returns are defined as follows:

Bit 12 -- System Shut Down
Bit 13 -- Hang User Up requested by SP.
Bit 14 -- User Logged On indicated by SP.
Bit 15 -- New User Called indicated by SP.

Bits 11:6 of the A-register will contain the device number.

An unsolicited event with status of all zeros indicates an INIT request from the SP.

Table 3
SP COMMAND DEFINITIONS

Bits 15:13 of all commands received from the SP contain the command value. Represented in octal, these bits give a range of 00 to 16. Obviously, more SP commands are defined than this range will permit. Therefore, one of these commands is interpreted as a "micro-coded" command with the actual command indicated by other bits. In many cases, both primary and micro-coded commands use bits 12:8 to indicate the TSB port number. Several other commands use bits 12:7 to indicate a TSB logical unit number. Certain other commands use these and other bits for entirely different purposes. In the summary which follows, these exceptions are noted. Several commands are followed by a second parameter word from the SP. This mechanism is used to supply length and other such indicators. Most commands do not require a response to the SP. However, for those which do, the responses are outlined as notes.

Primary commands

Command	Mnemonic	Function	Parameter <bits>	Notes
00	POC	Process output character	Character <7:0>	
02	STE	Start enter timing	Time in seconds <7:0>	
04	STP	Subtype information	Subtype <7:0>	
06	PHO	Phones timing	Time in seconds <7:0>	
10	PCF	Perform control function	Function <6:0>	1
12	POS	Process output string	String length <7:0>	2
14	TIN	Terminal information	Port<12:8> Micro-command <5:0>	
16	---	Micro-coded command	Micro-command <5:0>	

Microcommands

Command	Mnemonic	Function	Notes
00	---	Micro-micro-command	3
01	UIR	User is running	
02	UNR	User is not running	
03	IWT	Input wait	
04	HUU	Hang user up	
05	ULO	User logged on	
06	EON	Echo on	
07	EOF	Echo off	
10	TPO	Tape mode on	
11	STR	Start timed retry (ASCII files)	

12	NUC	New user called	
13	KTO	Kill terminal output	
14	ALI	Allow input	
15	OWT	Output wait	
16	IBA	Is buffer available	4
17	ADV	Allocate device (ASCII files)	5
20	RDV	Release device (ASCII files)	6
21	ALB	Allocate output buffer (ASCII files)	7
22	XRB	Transfer input buffer (ASCII files)	8
23	BKS	Backspace	
24	KDO	Kill device output (ASCII files)	
25	FNC	Fetch next character	
26	RJE	RJE command	9
27	ABT	User aborted	
30	PIS	Process input string	
31	NOP	(Reserved for future use)	
32	SCI	Send core image	10
33	RLB	Release buffer	
34	NOP	(Reserved for future use)	
35	SBP	Save buffer pointer	
36	RBP	Restore buffer pointer	
37	CBE	Console buffer empty	

TIN Microcommands

00	WTP	What terminal type
01	SBL	Send buffer length

Notes

1. PCF is an ASCII files related command. PCF returns one of the following responses to the SP:

   0 - command accepted
   1 - device not ready
   2 - illegal data input from device
   3 - hardware read error on device
   4 - unrecoverable device error
   -1 - no buffer available
   -2 - device end of file
   -3 - no data available from device

   In the non-zero cases, the response is simply an echo of the current device status.

2. POS responds to the SP as follows:

0 - command accepted
-1 - no buffer available

3. The following micro-micrc-coded commands are defined by bits 9:7 of the command:

0 INI Initialize IOP
1 KSN Cold dump
2 SNP Send number of ports
3 SDT Send device table
4 SSD System shutdown

For the INI command, the fcllowing SP response is given:

0 - RJE is present in system
1 - RJE is not present in system

Other commands yield obvicus respcnses.

4. IBA responds to the SP as fcllows:

0 - buffer is available
-1 - no buffer available

5. ADV is a two word command. The second word received from the SP contains the buffer length to be associated with the device being allocated.

6. RDV responds to the SP as follows:

0 - device was released
-1 - device is still busy

7. ALB is a two word command. The second word received frcm the SP contains the buffer length to be allocated. ALB then responds the SP as follcws:

0 - buffer was allocated
¬0 - same as PCF (see ncte 1)

8. XRB responds to the SP as fcllows:

0 - buffer available
¬0 - same as PCF (see ncte 1)

9. RJE responds to the SP as follows:

0 - buffer available

-1 - no buffer available

10. SCI is a two word command. The second word received from the SP contains the start address for the area of the ICP to be returned to the SP.

Interconnect Kit Handler

Functional description

The interconnect kit handler (ICKH) manages communications between the system processor (SP) and the I/O processor (IOP) at the user program level. Those communications consist of the dialogue between a separate handler, managing time share terminals, and the system processor. By definition the interconnect kit has characteristics which require special consideration.

1. Although the interconnect kit is a single physical device, it must support 32 separate and independent data streams (one for each potential user of the time-share system).

2. The identity of these separate data streams must be maintained in its interactions with other handlers as well as with .IOC.

3. Each of the data streams is bidirectional, used for messages from the SP to the external handler and for messages from the external handler to the SP.

4. For reasons of performance the ICKH and the handlers with which it interacts will not utilize the dynamic buffering facilities offered by the buffer manager.

5. The system processcr "controls" each data stream. The nature of communication is that every input tc the SP evokes a response in the form of a command or output.

The impact of these characteristics on the ICKH handler is outlined below:

1. Control and status information must be maintained for individual data streams (see Local data structures below).

2. Stream identification is embedded in all communications with external routines. The necessary information is maintained in the control words attached to each buffer.

3. Since information flow between handlers is bidirectional, contrcl information in the form of a

command code and status information is attached to every buffer queued.

4. Buffers are managed by the ICKH. All data buffers used are dedicated to the handler. The ICKH queues empty buffers for other handlers to fill; or alternatively, queues full ones as output to another handler and receives them back as work in its own queue. This is in lieu of the dynamic facilities of the buffer manager.

5. Buffering and processing techniques are dictated by the SP through its responses to inputs it receives. These are communicated to the handler through completed event notifications.

A data stream can be in one of three operating states. The states and buffering techniques associated with each are:

1. Conversation state - all data received from another handler and transmitted to the SP is single buffered. Every transmission to the SP is followed by a transmission from the SP. If a transmission from the SP is terminated with an indication of more output to follow, double buffering is used. Outputs terminated with a transmit complete indication cause input from the other handler tc be enabled.

2. Running state - input from the other handler is enabled only at the direction of the SP and is single buffered. Outputs from the SP to the other handler are double buffered under the same conditions as conversation state. Outputs terminated with a transmit complete indication cause input from the SP to be enabled.

3. Tape state - input from the other handler to the SP is double-buffered. There is no output to the other handler in tape state.

State transitions occur at the direction of the SP either through an explicit command or implicitly due to an unexpected occurrance.

Submodule functions

Implicit to this discussion is the understanding that each data stream is independent of all others. Processing requirements are determined by the "state" of the stream as described previously. The "initial" or native state of each data stream is conversation. State transitions occur because of an explicit directive from the SP or due to the occurence of an unexpected event. The valid state transitions and causes are:

1. conversation to running - occurs when the SP indicates UIR (user is running)

2. conversation to tape - occurs when the SP indicates TPO (tape mode)

3. running to conversation - occurs when the SP indicates UNR (user not running) or ABR (user aborted)

4. tape to conversation - occurs on the first data output from the SP.

5. tape to running - occurs when SP indicates UIR (user is running).

Because of performance considerations relative to the SP software and because the ICKH (and the IOP) are in effect driven by the SP, the sequence of operations is normally a WRITE to the SP immediately followed by a READ terminating with output from the SP and/or an indication of action to be performed. The indicators, as documented in Base Design Specifications of the Inter-connect Kit Driver are:

1. Buffer full (BFL) - the SP has data to send

2. Start ENTER timing (ENT) - the SP requests a timed read from the terminal

3. Allow MUX input (ALI) - the SP requests an untimed read from the terminal

4. SP Input Killed (SIK) - all output operations to the mux should be purged and an input is requested.

5. User running (UIR) - directs a state transition from conversation or tape to running

6. User not running (UNR) - directs a state transition from running to conversation

7. Tape mode on (TMO) - directs a state transition from conversation to tape

8. User aborted (ABR) - directs a state transition from running to conversation

9. no indicators on at the termination of a READ or WRITE is a valid return subject to state dependent processing

Unsolicited events can occur to direct:

1. Buffer Full (BFL) - indicates SP output pending

2. SP Input Killed (SIK) - indicates a request to purge terminal output

3. User Aborted (ABR) - directs a state transition from running to conversation

4. Hang user up (HUU) - the SP requests a user be disconnected

5. User logged on (ULO) - the SP indicates a user has logged on

6. New user called (NUC) - the SP indicates a new user has called

7. System Shutdown (SSD) - the SP is shutting down and all terminals should be disabled.

8. Initialization (INI) - the SP is directing reinitialization of the system

In addition there is one event which is processed as an appendage to the ICK driver:

1. Phones (PHO) - a phones timing parameter has been received

ICKH consists of four major segments; an I/O complete segment, a Primer segment, an Initialization segment, and a Timer segment. Each of these is outlined in some detail below and in the accompanying flow charts.

The initialization segment (ICKHI) is entered at system startup time (i.e., IOP load). Processing consists of:

1. initializing all SIB indicators

2. acquire and store identification for the ICK and MUX handlers.

3. exit to the calling routine

The initialization segment is reentered at system INIT time (i.e., SP startup) and the processing proceeds as follows:

1. purge the queues for the ICK and MUX handlers
2. issue a Clear call to the interconnect driver.

3. empty the ICK queue and make buffers available.

4. purge all TQE's and set stream status word to zero.

5. queue an ENABLE operation on each data stream.

6. exit to the calling routine.

The Primer entry (ICKHP) is activated whenever another handler has queued work for the ICKH. Processing proceeds as follows:

1. obtain queue entry from queue manager

2. locate proper SIB

3. branch to appropriate processing state routine

Conversation state processing (PCON)

1. determine queue entry type and process accordingly

2. Terminal input - issue WRITE/READ call to the ICK driver and exit. All writes contain status as communicated by the mux handler. If the line connect flag in BCW8 is set, start LOGON timing.

3. Empty buffer return (NOP) - mark buffer as available if flags W=I=O=P=C=D=C. Otherwise use the tuffer as directed by those flgas and exit.

4. Control (break or line drop) - issue a WRITE CCNTROL to the ICK driver with the appropriate indicator set and exit.

5. Control (line ccnnect) - queue READ through the queue manager, queue timer element, L=1, exit.

Running state processing (PRUN)

1. determine queue entry type and process accordingly.

2. terminal input - if ENTER timing is in progress, purge the TQE and insert time in BCW9. Always issue WRITE/READ call to ICK driver, then exit. All writes contain status as ccmmunicated by the mux handler.

3. empty buffer return (NOP) - same as conversation state with the follwoing exceptions. If W=1, queue a zero length write to MUXH with Bit 15, BCW1 set to 1, clear W and exit. If BCW1 Bit15=1, start ENTER statement timing, queue a READ to the terminal, and exit.

4. Control (break or line drop) - issue a WRITE CONTROL to the ICK driver with the appropriate indicator set, then exit.

Tape state processing (PTAP)

1. All work queued is input from the terminal. Issue a WRITE/READ to the ICK driver, then exit. All writes contain status as ccmmunicated by the mux handler.

2. or CONTROL (lire drop) - issue a WRITE CONTROL indicating user hung up and exit.

The exit procedure for ICKHP is a JMP .COM.

The I/O complete segment (ICKHC) is activated by completed event notifications for the ICK. ICKHC processing is state sensitive and proceeds as follows:

1. obtain I/O complete entry

2. locate appropriate SIB

3. observe state change indicators, change state if necessary and branch to processing state routine.

Conversation state processing (CCON)

1. determine event type and process as follows

2. SP Output - if character count is non-zero, queue output as WRITE through queue manager, else free buffer. Then,
   a. If BFL=1, issue a READ to the ICK driver if a buffer is available else set 0=1. Exit.
   b. If ALI=1 or ALI=BFL=SIK=0, queue a buffer with READ requested through the queue manager if a buffer is available, else set I=1. Exit.
   c. If SIK=1, queue an available buffer with CONTROL-PURGE requested and set I=1. If no buffer is available set P=1. Exit.

3. Initialize (INI) - jump to system restart routine

4. Hang user up (HUU=1) - queue a buffer with control disable. Queue a timer element with .1 sec interval, set U=L=E=0, and exit

5. User logged on (ULO=1) - set L=0,U=1, if T=1 purge the timer queue element and exit.

6. New user called (NUC) - set L=1,U=1, queue timer element with interval=PHONE, and exit.

7. System Shutdown (SSD) - disable timer routine, I/O complete segment, and all data streams. Reset to native state with no operations pending.

Running state processing (CRUN)

1. Determine event type and process accordingly

2. SP Output - if character count is non-zero, queue output as WRITE through queue manager, else free the buffer and then,

   a. If BFL=1, or ALI=BFL=SIK=ENT=0, issue a READ to the ICK driver if a buffer is available, else set O=1. Exit.

   b. ALI=1, queue a READ through the queue manager if a buffer is available, else set I=1. Exit.

   c. SIK=1, queue a CONTROL-PURGE through the queue manager if a buffer is available and set I=1. Else set P=1. Exit.

   d. ENT=1, set E=1 queue zero length WRITE to terminal and save time parameter. If no buffer is available for writing set W=1 and exit.

3. Hang User UP (HUU) - queue a DISABLE request, set a timer with a one second expiration, and set U=E=L=S=0.

Tape state processing (CTAP)

1. determine entry type

2. must be ALI=1, queue READ through queue manager, I=1, exit. If ALI=0, then return to Conversation state.

The exit procedure for ICKHC is a JMP .COM.

PROCESSING STATE / IO COMPLETE INDICATORS	CONVERSATION READ	CONVERSATION UNSOL	RUNNING READ	RUNNING UNSOL	TAPE READ	TAPE UNSOL
BFL	X	X	X	X	*	*
ENT			X	X		
ALI	X		X	X	X	
SIK	X	X	X	X		
UIR	*			*	*	
UNR			*			
TMO	*					
ABR			*	*		
HUU		X		*		
ULO		X				
NUC		X				
RLB (BFL=ALI=SIK=0)	X		X			
INI		X				
SSD		X				

X - occurence possible
* - IMPLIES STATE TRANSITION

The timer segment (ICKHT) is used as a subroutine by the other segments and the TBG handler. There are three parts:

Create timer queue element (TSTR)

1. get TQE address from A-reg
2. set T=1 in SIB
3. JSB to timer enque routine
4. JMP TSTR,I

Purge timer queue element (TPRG)

1. get TQE address from A-reg
2. JSB to timer dequeue routine
3. set T=0
4. JMP TPRG,I

Process timer expiration (TEXP)

1. locate SIB, set T=0
2. If L=1, queue a CONTROL-DISABLE (=) purge), exit.
3. If E=1, queue a CONTROL-PURGE through the queue manager. Exit.
4. If L=E=0, queue a CONTROL-ENABLE through the queue manager. Exit
5. JMP TEXP,I

Interfacing

Calling Sequences:

ICKH has four entry points. The primer entry, ICKHP, is entered from the dispatcher when work is queued for ICKH. The calling sequence is

```
NOP                 switch location
JSB   ICKHP
```

Exit is via:

```
JMP   .COM.
```

The I/O complete entry, ICKHC, is called by the dispatcher whenever an event completes on the interconnect kit. The calling sequence is:

```
LDA
LDB
JMP   ICKHC
```

and it exits through:

```
JMP   .COM.
```

A third entry point, ICKHI, is used, only during system startup. The calling sequence is:

```
JSB   ICKHI
```

and it exits by

```
JMP   ICKHI,I
```

The fourth entry point, ICKHT, is used for timer services. The calling sequence is:

```
LDA   TQE           TQE address
JSB   ICKHT
```

and exit is via

```
JMP   ICKHT,I
```

Called Routines:

The interconnect kit handler uses the services of the queue manager, buffer manager, allocate/deallocate manager, and the TBG handler. The handler manages the interconnect kit through .IOC. and the ICK driver.

Local data structures

Associated with every data stream is a block containing status and control information, buffer pointers, and a timer queue area. The stream information blocks (SIBs) are ordered by stream id number (0-31) into a stream information table (SIT). Each block is 8 words long so that any SIB address can be built by the computation

SIBadd=SITadd+8*(stream id)

The SIB entries are defined as follows:

```
WORD 1          STATUS INDICATORS
15 14 13 12 11 10 9 8 7 6 5 4 1-0
P  O  I  C  D  W  E L Z T U R  S
```

P - purge flag
=1 the SP has requested all output be killed
=0 no purge in progress

O - output flag
=1 output from SP pending
=0 no output pending

I - input flag
=1 the SP is waiting for input
=0 no input request pending

C - enable flag
=1 enable operation pending
=0 no enable pending

D - disable flag
=1 disable operation pending
=0 no disable pending

W - special write flag
=1 zero length write to terminal pending
=0 no write pending

E - ENTER flag
=1 ENTER statement timing in progress
=0 no ENTER statement timing active

L - Logon flag
=1 Phones timing in progress
=0 no Phones timing in progress

```
Z -   disable timing flag
      =1  disable timing in progress
      =0  no disable timing active

T -   timer flag
      =1  timer element queued
      =0  timer element not queued

U -   user activity flag
      =1  user signed on
      =0  no user active

R -   read activity
      =1  read active on ICK
      =0  no read active

S -   state flag
      =0  conversation state
      =1  running state
      =2  tape input state

WORD 2 - BUFFER ACTIVITY FLAGS

5  4  3 2  1  0
          B2 B1

B2  - buffer allocation flag
      =1  buffer is allocated (see BS2)
      =0  buffer 2 is available

B1  - buffer 1 allocation flag
      same as B2 definition

WORD 3 - Buffer 1 Address

15                              0
POINTER TO BUFFER #1

WORD 4 - Buffer 2 Address

15                              0
POINTER TO BUFFER #2

WORDS 5-8 - System Timer Queue Element

  15                                    0
5  TIME INTERVAL IN 1/10 ths OF SEC
6  ADDRESS OF TIMER EXPIRATION PROCESSOR*
```

7 TBG DRIVER LINK WORD
8 TBG DRIVER LINK WORD

*ICKHT

The queuing and control information required by the queue manager, handlers, drivers, and .IOC. is contained in a set of buffer control words (BCW's) appended as a prefix to each buffer. Descriptions are found under "Global data structures." Word 8 and Word 1 of this prefix are jointly defined by the interconnect bit and multiplexor handlers for communication of status information. All other uses conform to the descriptions under GLOBAL DATA STRUCTURES.

Word 8 - Event Status

Bit	Usage
15	inclusive OR of bits 14-10
14	record received with lost data error
13	record received with parity error
12	line connect
11	line disconnect
10	break character received
8	last operation indicator 1=terminal output 0=terminal input

Word 1 - Buffer Type

Bit	Usage
15	special zero length write to terminal or timed read completing

Multiplexor Handler

Functional description

The multiplexor handler (MUXH) manages communications between those devices attached to the 12920A multiplexors and the interconnect kit handler. Because of its close interelationship with ICKH and because of the nature of the physical device it controls, MUXH has some unique characteristics:

1. Since the 12920A multiplexor is a single physical device which runs multiple logical devices, all communication must be specific to a logical device. For this reason, all communications directed to or emanating from MUXH must contain device indentifiers. This is true for both IOC and QM traffic.

2. The logical devices attached to the mux are bidirectional. How they are used is dictated by ICKH; therefore, all communications between MUXH and ICKH contain explicit directives on actions to be performed.

3. Each logical device (or data stream) is independent of every other data stream. Separate status and control information is maintained for each.

4. Buffering is maraged by ICKH and all buffers are interlocked. MUXH disposes of empty buffers by queuing them to ICKH and is ignorant of all processing states as defined for ICKH.

Submodule functions

As alluded to earlier the mux handler is comprised of three segments; an initialization segment called at system startup, a Primer section which processes work queued for the mux handler and an I/O complete section which processes I/O event complete notifications. Implicit to the subsequent discussion of mux handler processing is the understanding that each data stream is separate and independent.

The Primer section (MUXHP) processes all work queued for MUXH by other handlers. Work consists of data and/or control requests as defined by the buffer control words prefixing each queue entry. MUXHP processing proceeds as follows:

1. obtain queue entry from QM

2. establish stream id and save

3. obtain command code and process as follows:

   a. WRITE - issue WRITE .IOC. call to driver, and exit.

   b. READ - issue READ .IOC. call to driver, and exit.

   c. CONTROL-PURGE - issue PURGE .IOC. call to driver, return buffer as a NOP, and exit.

   d. CONTROL-ENABLE - configure stream for auto-speed detect and issue ENABLE .IOC. call to driver, and exit.

   e. CONTROL-DISABLE - issue DISABLE .IOC. call to driver, return buffer as NOP, and exit.

   f. ALLOCATED BUFFER - use as directed by the first non-zero entry in the Port Table (PTAB) and zero the entry.

The I/O complete section (MUXHC) processes completed event and unsolicited event notifications. Notifications consist of a copy of the ICC parameter word, and the queued request entry (see GLOBAL) MUXHC processing is as follows:

1. establish stream id and save

2. if notice was solicited go to 3; else

   a. if line drop, acquire control buffer and queue event notification through QM, then exit

   b. if break received, acquire control buffer and queue notification through QM, and exit.

   c. if unable to obtain a control buffer, save event status in PIAB and exit.

3. If the operation was purged, set CMD=0, queue the buffer, and exit

4. Write complete - set CMD=0, queue the buffer, and exit

5. Read complete - set CMD=1, save completion status, queue the buffer and exit

6. Enable complete -set CMD=1, indicate line connect, queue the buffer and exit

7. Disable complete - set CMD=0, queue the buffer and exit

8. Purge complete - set CMD=0, queue the buffer, and exit

The initialization section (MUXHI) is entered at system startup. Processing requirements are:

1. obtain identification for MUX and ICK handlers.

2. issue CLEAR .IOC. call for all data streams

3. exit

Interfacing

calling sequences:

MUXH consists of three segments, an Initialization section (MUXHI), a Primer section (MUXHP), and an I/O complete section (MUXHC). MUXHP is activated by the QM whenever another handler has queued work for MUXH, it is entered from the system dispatcher by

```
NOP          active gate set by QM
JSB   MUXHP
```

The primer exits by a JMP .COM.

MUXHC is entered from the system dispatcher for notification of completed events on mux devices or to inform the handler of unsolicited occurrences on the devices. The calling sequence is

```
LDA          IOC pointer
LDB          buffer pointer/status word
JMP   MUXHC
```

and the completed event section exits

```
JMP   .COM.
```

The initialization section (MUXHI) is entered at system startup by

```
JSB   MUXHI
```

and returns control via

```
JMP   MUXHI,I
```

called routines:

MUXH calls upon the queue manager (.GETQ,.PUTQ,&.SEEQ) and the services of .IOC.

Local data structures

Attached to each buffer queued to MUXH is a nine word prefix of control words. These BCW's as well as those mentioned above are described under "Global data structures." Word 8 of the BCWs is jointly used by the multiplexor and interconnect kit handlers to communicate event status information. All other uses conform to the descriptions under "Global data structures."

Word 8 - Event Status

Bit	Usage
15	inclusive OR of bits 14-10
14	record received with lost data error
13	record received with parity error
12	line connect
11	line disconnect
10	break character received

A Port Table (PTAB) is maintained with a one-word entry for each port configured (the entries are in port number order). A non-zero entry indicates that an allocated control buffer is expected for that port. The content of the entry is the event status which caused the buffer request.

I. Functional Specifications

The multiplexer data interface driver provides an interface for input/output operations between an HP2100 computer and asynchronous devices attached tc the 12920A multiplexer.

The multiplexer driver will handle up to two 12920A multiplexers simultaneously (a maximum of 32 I/O channels).

The following functions are available thru calls tc .IOC.:

Function	Sub Function	Operation
ØØ	Ø	Reserved
Ø1	ØØ	Read characters from the designated channel into the designated buffer. Terminate on CR.
Ø1	1Ø	Reserved for future use.
Ø1	2Ø	Enable the designated channel; i.e. allow a user to dial up on the designated channel. Terminate when a user dials up.
Ø2	ØØ	Write the designated number of characters from the designated buffer. Terminate at end cf buffer.
Ø2	2Ø	Control the operation of the multiplexer by outputting the given parameter word.
Ø2	3Ø	Disable the designated channel. Do not allow a user to dial up the designated channel. No further operations are allowed tc the channel until an Enable is executed.
Ø2	4Ø	Purge the current read or write or both operation(s).

II. Calling Sequences

General calling sequences following the form for regular .IOC. calls:

```
EXT                 .IOC.
  .
  .
  .
JSB                 .IOC.
OCT                 function   subfunction   unit-reference
JSB or JMP          reject address    (error return)
DEF                 buffer address
DEC or OCT          buffer length in plus bytes
(normal return)
```

the format of word 2 is in general: (PVM = ø at all times)

```
15        1211            9 8 7 6 5              0
function     subfunction    P V M  unit-reference
```

Note: The queued version of .IOC. requires each buffer to be prefixed with buffer control words. This is assumed to hold in the following description. Also note that if more than one multiplexer is attached to the computer the "unit reference" is the select code of the first (lowest) multiplexer. Buffer control word 5 is specified as follows for all .IOC. calls.

```
BITS   6 - 7      reserved for PURGE call.
BITS   8 - 15     channel number at which the command is
                  directed.
```

Read:

```
     JSB                .IOC.
     OCT                function   subfunction   unit reference
     JSB or JMP         reject address    (error return)
     DEF                buffer address
     DEC or OCT         buffer length in plus bytes
```

function = Ø1
subfunction = ØØ
unit reference = nn
    where nn is the unit reference of the multiplexer.
buffer address = address of the area into which characters are read.
buffer length = maximum number of characters to be read.

Returns: If there is no error in the calling sequence and the read has been successfully iritiated the normal return is taken. If the reject address is taken then registers A and B contain pertinent information.

```
                 15 14 13                        8 7      10
     A-register = a                 equipment type   status
     B-register = d---------------ZERO------------------C
```

See the status IOC call for register A definations.

d=1 The device driver is busy
c=1 A DMA channel is not available
d=c=0 The function or subfunction is not valid for this device

Asynchronous return: When the read operation is terminated by the occurance of a carriage return or some unusual condition, a completed event notice is passed to .IOC., which then informs the user level program.

Status: (not currently supported)

```
     JSB                .IOC.
     OCT                function   subfunction   unit reference
     DEF                buffer address
     DEC                1
     (normal return)
```

function = 01
subfunction = 10
unit reference = nn
Where nn is the unit reference of the multiplexer for which status is requested. If nn is zero then system status is returned.

Returns: No errors are possible.

The contents of the A and B registers contain the status information.

```
                15 14 13              87      0
     A-register= a     equipment type  Status

     B-register = m     transmission log
```

a=0: The device is available; the previous operation is complete.
a=1: The device is available; the previous operation complete but a transmission error has been detected.
a=2: The device is not available for another request; a requested operation is in progress.
a=3: Previous operation was purged.

equipment type: These six bits indicate the device type
The value is 77

Status bit definition
Bit 0=0: A character was received
=1: A character was transmitted
Bit 1=0: No characters lost cn read operation
=1: One or more characters have been lost during the read operaticn
Bit 2=0: No break detected during current operation
=1: A break character was received
Bit 3=0: Diagnose mode is not set
=1: Diagnose mode is set
Bit 4=0: The seeking bit is not set
=1: The seeking bit is set
m=0
Transmission lcg is the positive count at the end of the operation.

Write:

```
     JSB              .IOC.
     OCT              function   subfunction   unit reference
     JSB or JMP       reject address    (error return)
     DEF              buffer address
     DEC or OCT       buffer length in plus bytes
```

function = 02
subfunction = 00
unit reference = nn
     Where nn = is the unit reference of the multiplexer
buffer address = address of the output buffer
buffer length = the number of characters to be transmitted to the designated channel

Returns: If there is no error in the calling sequence and the operation has been successfully initiated (queued IOC) the normal return is taken. If an error is detected in the calling sequence or the operation can't be initiated the reject exit is taken.

Asynchronous return: When the write operation is complete (buffer empty) or purged by the user .IOC. notifies the user level program by an entry in the completed event queue.

NOTE: If a read command is active when a write command is passed to the driver, the read is purged before the write is initiated.

Enable

```
     JSB              .IOC.
     OCT              function  subfunction  unit reference
     JSB or JMP       reject address (error return)
     DEF              buffer address
     DEC or OCT       1 (constant 1)
     (normal return)
```

function = 01
subfunction = 20
unit reference = nn
Where nn is the unit reference of the multiplexer.
buffer address = address of a one word buffer

Returns: If there is no error in the calling sequence and the operation has been successfully initiated the normal exit is taken. Otherwise the reject exit is taken.

Asynchronous Return: The enable is posted complete when a connection is established for the requested channel. The driver assumes a connection is established when either of two conditions is met:

1. Data set ready is asserted by the data communications equipment.
2. A carriage return is received from the data terminal equipment.

Either of the above conditions causes an outstanding enable operation to be completed. Condition one (1) allows proper operation over dialup or dedicated lines with modems. Condition two (2) allows operation with hardwired (three wire) lines.

NOTE: Be sure that all current operations are complete before issuing the enable. It is best to precede the enable by a disable or purge operation. The current driver implementation purges all operations before processing the enable request. Further note that not all terminals will operate over three wire (send, receive, signal ground) circuits without special jumpers attached to the connector at the data terminal end; some terminals require the presence of data set ready, and/or clear to send and/or carrier detect.

Control:

```
     JSB              .IOC.
     OCT              function   subfunction   unit reference
     JSB or JMP       reject address
     DEF              buffer address
     DEC or OCT       1     (constant 1)
     (normal return)
```

function = ø2
subfunction = 2ø
unit reference = nn
Where nn is the unit reference of the multiplexer.

Returns: Like the write call.

Operation: The parameter word at the given buffer address is output to the designated channel.

Parameter word definition:

Bit 15 = 1 must be set to one
Bit 14 = 0 output parameters to receive channel
If Bit 14 = 0, then the following definitions hold:
Bit 13 = 0 disable receive interrupts
Bit 13 = 1 enable receive interrupts
Bit 12 = 0 do not Echo received characters
Bit 12 = 1 Echo received characters
Bit 11 = 0 disable diagnose logic
Bit 11 = 1 enable diagnose logic

Bit 14 = 1
If Bit 14 = 1, then the following definitions hold:
Bit 13 = 0 disable transmit interrupts
Bit 13 = 1 enable transmit interrupts
Bit 12 = 0 generate even parity for 8 bit ASCII
Bit 12 = 1 generate odd parity for 8 bit ASCII
Bit 11 = 0 disable diagnose logic
Bit 11 = 1 enable diagnose logic

Regardless of the setting of bit 14 the following definitions hold:

Bits 8-10 (character length - 5) MOD(12)
Bits 0-7 (14,400/Baud rate) - 1

For a control function the channel number (Bits 8-15 of Buffer prefix word 5) have a special meaning.

Bit 13 = 1 diagnostic channel configuration
Bit 14 = 0 first mux
Bit 14 = 1 second mux

Disable:

```
     JSB              .IOC.
     OCT              function   subfunction   unit reference
     JSB or JMP       reject address
     DEF              buffer address
     DEC or OCT       1 (constant 1)
     (normal return)
```

function = Ø2
subfunction = 3Ø
unit reference = nn
     Where nn is the unit reference of the multiplexer.
buffer address = address of a one word buffer.

Returns: If there is no error in the calling sequence and the operation has been successfully completed the normal exit is taken. Otherwise the reject exit is taken.

NOTE: Disable will purge all outstanding read and write operations before being initiated. The data set control interface for the requested multiplexer channel is set to cause a "ring forever" condition; this can only be cleared by an enable being issued to the same multiplexer channel.

Purge:

```
     JSB                .IOC.
     OCT                function   subfunction   unit reference
     JSB or JMP         reject address
     DEF                buffer address
     DEC or OCT         1 (constant 1)
     (normal return)
```

function = Ø2
subfunction = 4Ø
unit reference = nn
Where nn is the unit reference of the (first) multiplexer.
buffer address = address of a null (Ø length) buffer.
Only the buffer control words (1-5) are passed to, and used by the driver. The word at buffer address-1 contains the following information:

```
     BITS:  7  6
            0  0       Reserved
            0  1       Purge current read operation
            1  0       Purge current write operation
            1  1       Purge current read and write operations
```

As usual, bits 8-15 contain the channel number on which the purge operation is to take place.

III. Unsolicited Event Notifications

There are two conditions which can cause an unsolicited event notice to be passed from the 12920 A driver to the base level program. In both cases pertinent information is returned in A and B.

The first is a "line drop" or " connection broken" on the TP line.

A register
- Bits 0-5 = unit reference number of the Multiplexer
- Bits 6-11 = stream identifier (terminal number)

B register
- Bits 0-15 = Zero

The second unsolicited return is caused by the detection of an unusual condition during a data transmission.

A register
- Bits 0-5 unit reference number of the multiplexer
- Bits 6-11 stream identifier (terminal number)

B register
- Bit 0=0 current operation is a read
- Bit 0=1 current operation is a write
- Bit 1=0 no characters lost
- Bit 1=1 one or more characters lost during current operation
- Bit 2=0 no break detected
- Bit 2=1 break detected
- Bit 3=0 diagnose disabled
- Bit 3=1 diagnose enabled

IV. Design Overview

D.51 The 12920A multiplexer driver

The multiplexer driver provides input, output and data link control facilities for asynchronous devices attached to the 12920A multiplexer. The driver is modular in design and provides an interface between the 12920A hardware and Queued .IOC.

The driver is cognizant of the communication line status at all times. All lines must be enabled before any read or write operation is issued. When the link is established notification is passed to the user level program. When the operation (read/write) is completed notification is passed to the user level.

Design assumptions

The software driver for the 12920A multiplexer assumes the existence of the HP2000 ACCESS I/O processor microcode.

Since the driver is able to handle more than one multiplexer the select codes must be assigned as follows: first mux lower data select code; first mux upper data select code, first mux data set control board select code, second mux lower data select code, second mux upper data select code, second mux data set control board select code. This restriction is imposed because there is only one equipment table (EQT) regardless of the number of multiplexers attached to the system. The driver also assumes that these select codes are consecutive.

Design summary

Although the multiplexer is physically a single I/O device filling three I/O slots in the 2100 CPU, logically it must be viewed as 16 full duplex I/O channels. This implies that the driver must maintain information for each of the 16 channels attached to each multiplexer. This information is contained in Logical Unit Tables (LUT's) internal to the driver. Each LUT is composed of three sections, one for read commands, one for write commands, and one for terminal configuration. The structure of the LUT is outlined below and displayed in Table 1.

Like all BCS drivers the mux driver has an initiator (D.51) and a continuator section (I.51). The initiator

section is responsible for converting a requested operation into the appropriate actions. It also queues requests for channels which are busy when the request is made. There is a read queue and a write queue associated with each LUT. The continuator section is primarily an interrupt handler and event complete processor.

Key algorithm descriptions

Automatic speed detection -- The multiplexer driver software contains logic to determine at what baud rate (speed) an attached terminal is transmitting characters. The driver assumes the transmit speed is the same as the receive speed. Speed detection is facilitated by the hardware features of the multiplexer interface. Each interface has 16 full duplex input/output channels. Each of the channels is capable of operating independently of the others at any baud rate from about 50 to 2400 bits per second. In addition to the 16 main channels there are 5 diagnostic channels. By setting the diagnose bit in a main channel receive parameter, the 5 diagnostic channels can be tied to the main channel. Each of the diagnostic channels can be configured to receive at a baud rate different from the main channel and the other diagnostic channels. Speed detection operates as follows. The main channel is configured to receive characters at 2400 baud. The five diagnostic channels are configured to receive at 1200, 600, 300, 150, and 110 baud respectively. The user at his terminal strikes a known key (in the case of this driver a carriage return which is an octal 15). The main channel receiving at the 2400 baud rate interrupts before any associated diagnostic channels do. The software examines the received character. If it is a carriage return, the main channel is operating at the correct speed (2400 baud), and the only configuration that need be done is to turn off the diagnose bit in the receive channel parameter and turn off the speed detect bit in the user status word. If a carriage return is not seen at 2400 baud, the driver exits and awaits diagnostic channel interrupts. Eventually, the diagnostic channel configured for the same speed as the transmitting terminal will interrupt with a carriage return as the received character. The main channel is then reconfigured (from 2400 baud) to the rate associated with the diagnostic channel which had the interrupt; again the speed detect bit in the user status word is turned off. In all cases, whenever the correct speed is sensed (carriage return seen), the "parity detection in progress" bit of the user status word is set. This bit is used in the automatic parity detection algorithm.

Speed sensing for 2741 type terminals is handled in a slightly different fashion. Since 2741 terminals operate at 134.5 baud with 6 bit characters, their carriage return character would not be recognized using the algorithm described above. Every 2741 emits a special line control character, called a circle C, after each carriage return. Furthermore, this circle C is seen as an octal 100 when received at 300 baud and as an octal 174 when received at 150 baud. When the diagnostic channel which is configured to receive at 300 baud (channel 18) receives an octal 100, a specific flag ("selectric possibility" flag) is set. The next diagnostic channel interrupt (at 150 baud) will be an octal 174 if the terminal is indeed a selectric. If the octal 174 is not received at 150 baud, the "possibility" flag is reset.

Automatic terminal parity detection -- Automatic parity determination is accomplished in connection with automatic speed detection. When speed is correctly determined, two things are done. First, the "parity detection in progress" bit is turned on in the user status word. This bit tells the driver that the next character to accept must be a line feed (octal 12); all other characters are ignored. Secondly, the parity (bit 7) of the correctly received carriage return is saved in the channel status word. When the waited for line feed is received, its parity (bit 7) is also recorded in the channel status word. Since carriage return and line feed have opposite parity sense, the parity with which the transmitting terminal is operating can be determined as shown in the following table:

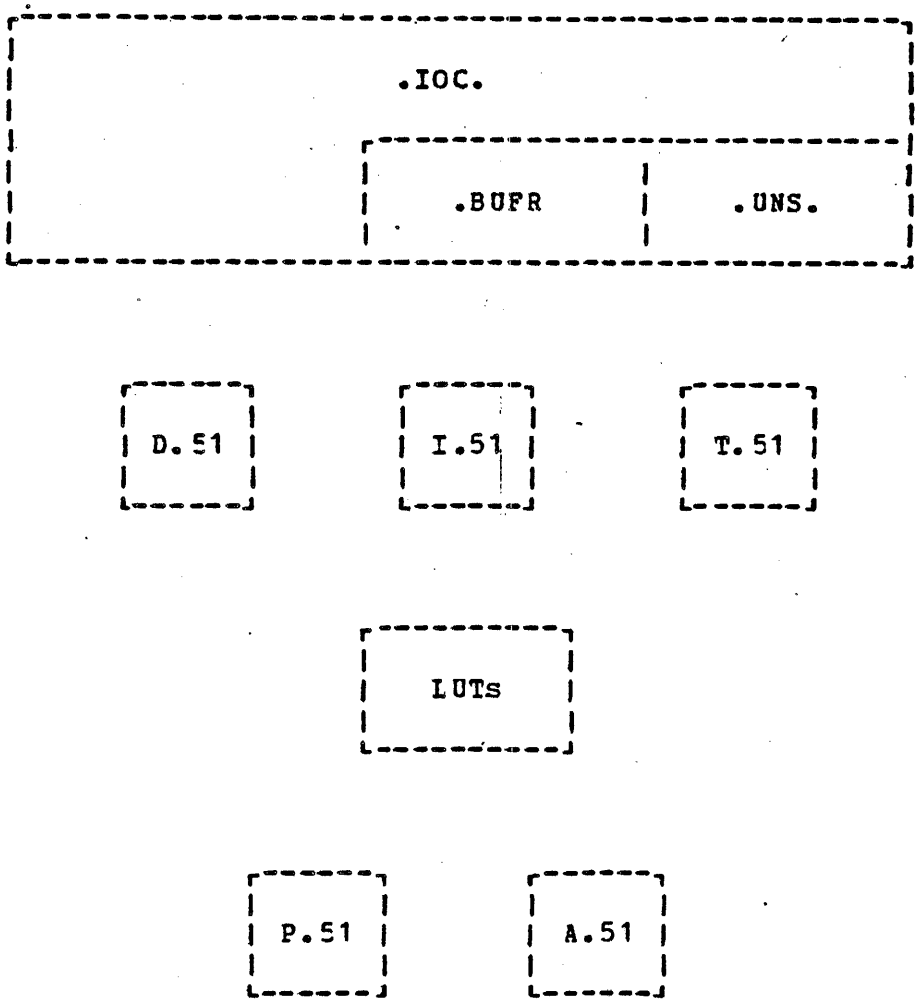
Received parity of		Terminal parity
CR	IF	
0	0	None - space for parity bit
0	1	Odd parity
1	0	Even parity
1	1	None - mark for parity bit

Note: An underlying assumption in both automatic speed and parity detection is that the carriage return and line feed are received without error. If speed and/or parity are improperly determined due to an error, the user will not be able to correctly log on the system. In this case, his port will be disabled and re-enabled after the PHONES parameter time limit has expired; always less then 255 seconds. Therefore, a port cannot be locked out indefinitely because of incorrect speed/parity determination.

Major module relationships

The module relationships are indicated in Figure 1. Data and control flow are indicated in the same figure. The continuator section of the driver follows the philosophy of the 2000F IOP multiplexer driver quite closely; indeed, much of the 2000F code for selectric processing remains intact.

FIGURE 1



Lines to the LUTs are intended to indicate data access only. All other lines indicate flow of control.

Module	Method of entry
D.51	.IOC. call
I.51	Interrupt from multiplexer channel
T.51	Interrupt from data set control board
P.51	JSB from D.C4 following power recovery
A.51	JSB from D.61 for special SP commands

V. Design Structures

The Logical Unit Table

The logical unit table (LUT) contains all the information essential tc the operation of the driver. The LUT contents are accessed or modified by all modules in the driver. There is one LUT per multiplexer channel and each LUT has a read section, a write section, and a terminal configuration section. Table 1 shows the contents cf an LUT.

TABLE 1

LUT CONTENTS

READ_SECTION

RBFAD	Read buffer address
RBYCT	Read byte count
RRCNT	Read request count
RBYAD	Read byte address
RQHED	Read queue head
RQEND	Read queue tail

WRITE_SECTION

WBFAD	Write buffer address
WBYCT	Write byte count
WRCNT	Write request count
WBYAD	Write byte address
WQHED	Write queue head
WQEND	Write queue tail

TERMINAL_CONFIGURATION_SECTION

TYPE	Terminal type
CDLY,LFDLY	Carriage return, line feed delay
DCNT	Delay counter
STAT	User status word
STAT2	Port status word
PPRM	Data set control board parameter
RPRM	Receive channel parameter
TPRM	Transmit channel parameter
SCNT	CR-LF delay counter for selectrics

TIMER_QUEUE_ELEMENT

OCT 24	20 clock ticks (2 seconds)
DEF DSTE,I	Exit routine for timer expiration
OCT 0,0	Timer queue link and save word
OCT nn	Port number in bits 13-10 (Bit 15 set if 2nd multiplexer)

LUT BIT DEFINITIONS

```
STAT -  USER STATUS WORD
 10000  User is in tape mode
     1  Speed detect in progress (not detected)
     4  Control X was entered
    10  Reserved
    20  Line drop timing in progress
    40  Line feed needed for tape mode prompt
   100  Special write active (/CRLF) using receive buffer
   200  User is in Run mode
   400  An operation is active
 40000  Input is not allowed
100000  Purge in progress (ignore interrupts)
  4000  X-OFF was input from terminet
 20000  Parity detection is in progress
  2000  Input configuration needed (Reserved for future use)
 20000  264X ENQ transmitted
  1000  Preceding input was ENQ


STAT2 - PORT STATUS WORD
     1  A character was received/transmitted
     2  A character was lost (input)
     4  A break (input) was detected
    10  The port is in diagnose mode
    20  The seeking bit is set
    40  A parity error (input) has occurred
   100  The channel requires parity
   200  Channel parity (odd=1, even=0)
   400  Parity of carriage return
  1000  Parity of line feed
 10000  )
   .    ) Teletype subtype
100000  )


TYPE - TERMINAL TYPE PLUS OTHERS
     2  Code determined (EBCDIC or CALL/360)
     4  Upper case mode
    10  "CENT" character
    20  "CENTC" character
    40  "CR" bit (used for output only)
   100  Control X was entered
   200  Circle c sent
   400  Circle d (Psuedo) Transmit interrupt
  1000  SYNC       "        "        "
```

2000	Space	"	"	"
4000	Space	"	"	"
10000	Space	"	"	"

PPRM - PHONES PARAMETER WORD (See ERS fcr definition)
RPRM - RECEIVE PARAMETER WORD (See ERS for definition)
TPRM - TRANSMIT PARAMETER WORD (See ERS for definition)

VI. Major Functional Modules

The mux driver consists of five major modules D.51, I.51, A.51, T.51, P.51, the initiator section, the continuator section (driver control), the appendage section, telephone line control, and power fail/restart section. Each section has specific functions distinct from each of the others.

D.51 The Initiator Section

The initiator section D.51 receives control from .IOC. as a result of a user call. D.51 checks the requested device to see if it is operable and the user requested operation and user supplied parameters are valid. If these conditions are satisfied, D.51 queues the request for processing. Purge and control requests are not queued; they are executed immediately.

In the event the user request can't be satisfied, the B register indicates the reason:

B= 100000 The device is busy or inoperable.
B= 000000 The request is not legal.

In these two cases the A register is set to 1. If the request was successfully initiated (this may mean successfully queued) the A register is zero and the B register contents are urdefined.

I.51 The Continuator Section

The continuator section I.51 receives control whenever one of the multiplexers attached to the IOP causes an interrupt, or it may be entered directly by a call from D.51 to start an operation.

Upon receiving control I.51 saves A, B, and E. Next I.51 determines whether the cause of the interrupt derives from a solicited request (read or write) on an unsolicited (asynchronous) event (i.e., break key struck). If the interrupt is due to a read/write request a character is appended to/removed from the buffer associated with the current operation(s) for the interrupting channel. Note that the channel number (0-15) must be associated with the correct multiplexer (0 or 1) to arrive at a user "stream number" (0-31). The appropriate code and/or protocol conversions are done on a per-character basis.

Interruptions due to unsolicited events are handled in a special way. Notification is returned to the user level via an unsolicited entry in the completed events queue of .IOC.. Included in the notification is the cause of the unsolicited interrupt. Certain unsolicited interrupts are transparent to the user level program. The occurrence of any of these special events causes driver dependent actions to occur. An example is the backspace character. The user level program is not informed of the receipt of a BS character; rather the current byte pointer is decremented and normal processing continues.

The continuator section is also responsible for notifying .IOC. when operations are complete and for initiating the next queued request (if any).

A.51 The Appendage Control Section

The Appendage Control Section A.51 receives explicit control from the ICK driver via a JSB instruction. The purpose of A.51 is to respond to certain system processor initiated requests for information which is contained in the LUT for a given channel. A.51 decodes the request, obtains and formats the data from the LUT, and returns to the ICK driver.

It is desirable to make this code as logically independent of the mux driver as possible.

T.51 The Telecommunications Line Control Section

The Telecommunications line control section T.51 is responsible for monitoring the status of user connections (phone lines or hardwired) to the IOP. When a change in any line status is detected, the user level program is made aware of the change via an unsolicited entry in the completed event queue of .IOC.. The two possible line states are "connection established" and "connection broken." T.51 will time short line drops to determine if the line is permanently or temporarily down. Any line drop of 2.0 seconds or greater duration on an established line will cause a "connection broken" to be passed to the user level program.

T.51 will access the LUT's for the channel and modify certain status information. When an initial connection is established T.51 tells the driver control section, I.51, to begin auto-speed detect for the newly connected channel.

P.51 The Power Fail/restart Control Section

P.51, the power fail recovery section of the driver receives control from D.04 after power is restored to the CPU following a power failure. P.51 is responsible for restoring the multiplexer interface and data set control interface cards to the status they were in when the power failure occurred. In addition to restoring the device flag and control status, the proper input and output channel parameters must be restored in the multiplexer interface memory. The dataset control board interface parameters must also be restored, and any interruptions pending for either the multiplexers or data set control interfaces must be handled appropriately. Any multiplexer channels which were actively engaged in output at power fail time are restarted following the last character previously output. Any multiplexer channels which were in the input wait state have their receive operations terminated with a character lost indication; this allows the BASIC program to reissue the input operation.

SECTION I

OVERVIEW

SECTION IV

NON-SHAREABLE DEVICES

HEWLETT-PACKARD 2000 SYSTEM I/O PROCESSOR

NON-SHAREABLE DEVICES

CONTENTS

1.	D.63	INTERCONNECT KIT DRIVER (ASCII FILES)
2.	ASFH	ASCII FILE HANDLER
3.	CRH	CARD READER HANDLER
4.	D.11	CARD READER DRIVER
5.	LPH	LINE PRINTER HANDLER
6.	D.12	LINE PRINTER DRIVER
7.	PPH	PAPER TAPE PUNCH HANDLER
8.	D.13	PAPER TAPE PUNCH DRIVER
9.	PRO/CRH	PAPER TAPE READER HANDLER
10.	D.14	PHOTO READER DRIVER
11.	RPH	READER PUNCH INTERPRETER HANDLER
12.	D.34	READER PUNCH INTERPRETER DRIVER
13.	LTH	SERIAL LINK TERMINAL HANDLER
14.	D.53	SERIAL LINK TERMINAL DRIVER

```
*          D.63 is the ASCII FILES portion of the interconnect
*      kit driver.  The documentation on this module is broken
*      into the following sections:


*            I.  UNIT CONTROL TABLE

*           II.  SYSTEM PROCESSOR REQUESTS

*          III.  IOC REQUESTS

*           IV.  UNSOLICITED EVENTS

*            V.  BCW USAGE

*           VI.  IOP - SP PROTOCAL
```

I. UNIT CONTROL TABLE (UCT)

IOPC builds a table for D.03 called the unit control table. The UCT contains one 6 word entry for each Ascii File Device configured into the IOP. Each 6 word entry is called a unit control block (UCB). A UCB consists of the following elements:

1. UQHED head of request queue for this device

2. UFOOT tail of request queue for this device

3. USTAT logical status of the device (relative to SP and ASFH)

   BIT 15 device online
   set when ADV received
   cleared when RDV, KDU received
   BIT 14 WUU - WAKE USER UP MUST BE SENT
   set when RDV, ALB, XRB rejected
   set when STR received
   cleared when WUU sent or KDU received
   BIT 13 RDV - RELEASE DEVICE RECEIVED
   set when RDV first received
   cleared when RDV finally accepted
   BIT 12 UNIT TYPE
   set up by IOPC and never modified
   =0 normal (real) device
   =1 pseudo device
   BIT 11 ALB/XRB UNSOLICITED EVENT FLAG
   set when an allocate buffer (ALB) or transfer input buffer (XRB) unsolicited event is sent to the Ascii File Handler
   cleared when a read or write buffer arrives at the driver as a result of the unsolicited event
   BIT 10-0 NOT USED

4. UNUM TSB logical unit number

5. UBLEN buffer length

6. UDSTS physical device status

   -3 RJE not ready
   -2 EOF read
   -1 no buffer available
   0 device ready
   1 device not ready
   2 device error
   3 attention needed
   4 read/write failure

II. REQUESTS FROM SYSTEM PROCESSOR

1. ADV - Allocate Device

The system processor sends ADV when a BASIC program has assigned an Ascii File Device. ADV will eventually get passed to the device as a start command. Any preparation at the device driver level should be done at this time.

D.63 always accepts this command. An unsolicited event is put on the CRQ for the ASFH. The online bit is set in the unit status (USTAT).

2. XRB - Transfer Input Buffer

The system processor sends this command when it wants to have data read from a device.

A check is first made of the pending request queue for this device.

If a write to the SP for the device is pending, an IOP to SP transfer of the data is done via DMA.

If a write/control for the device is pending, it is executed. A check is then made of the newly updated device status (UDSTS). If the device status is good (zero), we go back to the beginning and check for more pending requests. If the status is bad, the command is rejected by sending the bad status across the receive channel. If the bad status was minus three (RJE not ready), no other processing takes place. For all other device errors, WUU is set in the unit status (USTAT).

If a read from the SP for the device is pending, it is forced to completion and processing continues as if there was no write queued.

If there is not a write to the SP queued, the device status (UDSTS) is checked. If the status is bad, the same bad status processing described previously for write/control is invoked. If the status is good, an XRB unsolicited eventis sent to the ASFH and the XRB/ALB bit is set in the unit status. The SP command is then rejected by sending a minus three across the receive channel for pseudo devices, and a minus one across for real devices. If the device is real, the WUU bit in the unit status is also set.

3. ALB - Allocate Buffer

The system processor sends this command when it has data to be written to the specified device.

A check is first made of the pending reqest queue for the device.

If a read from the SP for the device is pending, an SP to IOP data transfer is done via DMA.

If a write/control for the device is pending, it is executed. A check is then made of the newly updated device status (UDSTS). If the device status is good (zero), we go back to the beginning and look for more pending requests. If the status is bad, the command is rejected by sending the bad status across the receive channel. If the bad status was minus three (RJE not ready), no other processing takes place. For all other device errors, WUU is set in the unit status (USTAT).

If a write to the SP is queued, it is forced to completion and processing continues as if there were no read queued.

If there is no read queued, the device status is checked. If the status is bad, the same bad status processing described previously for write/control is invoked. If the status is good, an ALB unsolicited event is sent ot the ASFH and the XRB/ALB bit is set in the unit status. The SP command is then rejected by sending a minus three across the receive channel for pseudo devices, and a minus one across for real devices. If the device is real, the WUU bit in the unit status is also set.

4. PCF - Perform Control Function

The SP sends this command when it executes a CTL function in a BASIC program.

The device status (UDSTS) is checked. If the status is bad, the command is rejected by sending the bad status across the receive channel. If the bad status is minus three (RJE not ready), no other processing occurs. For all other device errors, WUU is set in the unit status (USTAT).

If the device status is good, the command is accepted by sending zero across the receive channel and the a PCF unsolicited event is sent to the ASFH. The control function to be performed is included in the unsolicited event data sent. The device status is then set to minus three for pseudo devices and minus one for real devices. The ASFH will later send a write/control down for the device to clear this status.

5. STR - Start Timed Retries

The SP eventually sends this command when the previous SP request was rejected due to a device error. It tells the IOP to periodically retry the current I/O request until the device comes up.

A STR unsolicited event is sent to the ASFH and the WUU bit is set in the unit status (USTAT).

6. RDV - Release Device

The SP sends this command when it is de-assigning the device from the current user.

First, the request queue for the device is purged. Purging is described in the IOC requests section of this document.

For pseudo devices, the command is always accepted and an RDV unsolicited event is always sent to the ASFH.

For real devices, processing differs. The first time the command is received, it is always rejected, a RDV unsolicited event is sent to the ASFH, and RDV and WUU are set in the unit status (USTAT). On succeeding RDV's, no unsolicited event is presented, the command is rejected if the device is down (UDSTS # 0) and accepted if the device is up.

7. KDO - Kill Device Output

The System Processor sends this command when it is aborting the current user of the specified device.

The device request queue for the device is first purged. Purging is described in the IOC requests section of this document. A purge unsolicited event is then sent to the ASFH and then both the device status (UDSTS) and unit status (USTAT) are cleared.

III. IOC REQUESTS

Control word	Function	Subfunction	Operation
0000XX	0	00	Init/Clear
1100XX	1	00	Read
1200XX	2	00	Write
1220XX	2	20	Write Control
1240XX	2	40	Purge

XX = IOC Unit Reference Number of ICK for Ascii Files

1. Init/Clear

IOPC puts a word in front of the UCT which is a count of the Ascii File Devices configured into the IOP. The Init/Clear processor uses this as the limit to a loop which does a purge of the request queue for each device. Purging is described in detail later in this section.

2. Read

A read requests an SP to IOP transfer of data for a device.

A read request is made as a result of the SP sending ALB for the device. The buffer is queued on the request queue in the UCB for the device. The device status (UDSTS) is cleared. ALB/XRB in the unit status (USTAT) is cleared. If the NUU bit in the unit status is set, Wake User Up will be sent to the SP. This will usually result in the SP resending the ALB which will now be accepted since the buffer is there.

3. Write

A write requests an IOP to SP transfer for a device.

A write request is made as a result of the SP sending XRB for the device. The same processing is done as is done for reads.

4. Write/Control

A write/control requests the device status (UDSTS) in the UCB be updated to the value specified.

The buffer is queued on the request queue in the UCB for the device. Actual execution of write/controls (ie. actual updating of UDSTS) does not happen when the request is queued. Write controls are executed on reception of PCF, ALB, and XRB commands from the system processor, and during purge processing. See the description of the related routine for details.

5. Purge

A purge requests the clearing of all requests from the specified devices request queue.

Besides being executed as a result of an IOC call, the Purge processor is also called by the KDO and RDV processing routines.

Purged status is stuffed in D.63's EQT status word. Each existing entry on the request queue for the device is dequeued. Any Write/Controls are executed. Other operations are placed on the CRQ via .BUFR. They will be be marked as purged by .BUFR because of the purge EQT status. When all operations have been purged, the EQT status is restored.

IV. UNSOLICITED EVENTS

Unsolicited event notices from D.63 will give the following information.

A Register

Bits 15-12		Not Used
Bits 11-6		TSB Logical Unit Number for the device
Bits 5-0		IOC Unit Reference Number of ICK for Ascii File

B Register

ADV	Bit	0	Set
	Bits	15-1	Negative buffer length for device, without sign bit
KDB	Bit	1	Set (octal 2)
XRB	Bits	3-2	Set (octal 14)
ALB	Bit	3	Set (octal 10)
PCF	Bits	15-8	Control Information
	Bit	4	Set
PDV	Bit	5	Set (octal 40)
STR	Bit	6	Set (octal 100)

```
*      V.  BCW USAGE
*
*
*          Although there are nine BCW words prefixing each buffer
*      only the last five are considered at the IOC/Driver level.
*      Furthermore, since D.63 is a self-queing driver handling
*      many possible streams,  some things are handled differently
*      than normal.  IOC does not set up the BCW's before calling
*      the driver - the driver does that itself.  Also, before
*      calling .BUFR, the driver must set up the DRQ and transmis-
*      sion log in the EQT with information that is relevant to
*      the completing stream.
*
*
*                BCW'S  -  INCOMING BUFFERS
*
*      WORD 1   (BCW 5)      Queuing word          Set up by driver
*      WORD 2   (BCW 6)      IOC request           Set up by driver
*      WORD 3   (BCW 7)      Request length        Set up by driver
*      WORD 4   (BCW 8)      Device status on      Set up at handler
*                              Write/Controls
*      WORD 5   (BCW 9)      TSB LU # in left      Set up at handler
*                              byte
*
*
*                BCW'S  -  OUTGOING BUFFERS
*
*      WORD 1   (BCW 5)      Queuing word          Set up by .BUFR  (CRQ)
*      WORD 2   (BCW 6)      IOC request           As above
*      WORD 3   (BCW 7)      Transmission log      Set up by .BUFR
*      WORD 4   (BCW 8)      Status                Set up by .BUFR
*      WORD 5   (BCW 9)      TSB LU # in left      As above
*                              byte
```

VI. IOP/SP PROTOCAL

For every Ascii File Device related command the SP sends the IOP gives back a response across its RECEIVE channel. Following are two tables listing the IOP responses and expected SP action as a result. The first table covers the user not using IF ERROR. The second covers the user using IF ERROR.

IOP/SP PROTOCAL - NORMAL

RESPONSE	MEANING	EXPECTED SP ACTION*
-3	RJE not ready	Try again later (SP dequeues and then requeues the guy)
-2	EOF	Release Device (RDV)
-1	Record not ready	Swap guy out, requeue when IOP sends Wake User Up (WUU)
0	Command accepted	Continue processing
1	Device Not Ready	Send Start Timed Retries (STR)
2	Device Error	Send Start Timed Retries (STR)
3	Attention Needed	Send Start Timed Retries when operator AWAkes device
4	Read/Write Failure	Send Kill Device Output (KDO)

* Besides the specified expected action, a KDO may come across at any time if the user hits break.

IOP/SP PROTOCAL - IF ERROR

RESPONSE	MEANING	EXPECTED SP ACTION*
-3	RJE not ready	Try again later (user can do other things and retry the I/O later in his program)
-2	EOF	Release Device (RDV)
-1	Record not ready	Swap guy out, requeue when IOP sends Wake User Up (WUU)
0	Command accepted	Continue processing
1	Device Not Ready	Send Start Timed Retries if the user retries the I/O, send Kill Device Outut if the user de-assigns the Ascii File
2	Device Failure	Same as Device Not Ready
3	Attention Needed	Same as Device Not Ready
4	Read/write Failure	Same as Device Not Ready

* Besides the specified expected action, a KDO may come across at any time if the user hits break.

ASCII File Handler

Functional description

The ASCII file handler coordinates communication between the System Processor (SP) and the handlers for those devices designated as ASCII files on the HP2000 ACCESS system. Like the interconnect kit, the ASCII file handler will support multiple data streams (up to 64) through a single physical device. It is to be understood that individual streams are independent of one another. ASFH functions both as a level 2 and a level 3 handler, dependent upon the characteristics of the particular data stream. Therefore, it may acquire buffers which are in effect controlled by another handler (ASFH=level 3) or it may direct another handler to acquire buffers which are controlled by ASFH (ASFH=level 2).

The handler maintains control and status information for each data stream (see Local data structures) as well as embedding the control information in the Buffer Control Words (BCW) (see Global data structures). Buffering and processing techniques are determined by the device class as returned by the Allocation/Deallocation manager. Buffering is generalized to n-levels where n is determined by the device class. All buffers will be dedicated for the duration of activity on the device.

The handler must respond to a number of events, both scheduled and unsolicited, which emanate from the SP. In addition it will be served by a single queue through which it receives work from other handlers.

The following unsolicited notifications occur:

1. Perform Control Function (PCF) - carry out the specified control function on the indicated device.

2. Allocate Device (ADV) - the indicated device and data stream should be initialized. This may imply generating a start directive to another handler or acquiring and dispatching buffers.

3. Release Device (RDV) - the indicated device and data stream should be deactivated. This implies generating a stop directive to another handler.

4. Allocate Buffer (ALB) - a buffer is required for data from the SP on the indicated device.

5. Transfer Input Buffer (XRB) - a record for transfer to the SP is required from the indicated device.

6. Kill Device Output (KDO) - all operations on the indicated device and data stream should be purged (implies RDV).

Scheduled event notifications occur in response to READ, WRITE, and WRITE/CONTROL requests made to the driver.

Queue entries direct the ASCII File Handler to perform work at the request of other handlers in the system. The type of processing will be determined by the Buffer Control Words (see BCW in Global data structures). Allowable indicators and their meanings are:

1. No operation - an empty buffer is being returned for reuse.

2. Write - a full buffer is available for transfer to the SP.

3. Read - for level 3 devices a request for data from the SP has been made

4. Stop - an end of file condition has been reached on an I/O device.

5. Error - a device error has occurred.

6. Allocated buffer - a previously requested buffer has become available.

7. Start - this only applies to level 3 devices and implies that stream is now available for use

Likewise the ASCII File Handler queues work to other handlers in the system and specifies processing through the BCW's. Valid queue entry types are:

1. Write - output the data to the I/O device.

2. Read - input data from the I/O device and send to the ASCII File Handler.

3. Start - perform necessary initialization for the device.

4. Purge - purge all current operations on the device.

5. Control - a control request (device specific) is being made of the device.

6. Stop - perform necessary functions to deactivate the stream

Submodule functions

The ASCII File Handler (ASFH) is comprised of three sections. The initialization section, ASFHI; the I/O complete section, ASFHC; and the primer section, ASFHP.

The initialization section is entered at system startup time (i.e. IOP load) and at system INIT time (i.e., SP ready). At startup time ASFHI must:

1. Configure all ICC calls with the logical unit number for ASCII files.

2. Do a FIND on all logical units to obtain Q names and device types to complete the SIB entries.

3. Exit to caller.

At system INIT time ASFHI is entered so that it can:

1. Purge all device queues that are not idle and mark them as idle.

2. Purge its own queue and empty it, returning any buffers to the buffer manager.

3. Exit to caller.

The primer section, ASFHP, is activated when an entry exists in the queue for ASFH. Processing consists of:

1. Obtain entry from the queue manager.

2. Locate proper SIB entry.

3. Determine entry type and branch to processing routine.

4. NOP entry processing

   a) if device is idle, free the buffer and exit.
   b) for input files, queue buffer with read request to external handler.
   c) for output files, issue a read call to the driver.
   d) for input/output files, use the buffer as indicated by the stream status flags (i.e., read from SP, read from external device, or show buffer as available).

6. Read entry process - if the device is active issue a read call to the driver, else free the buffer and exit.

7. Write entry process - if the device is active issue a write call to the driver, else free the buffer and exit.

8. Stop entry process - issue a write/control call to the driver.

9. Error entry processing - if the device is active issue a write/control request indicating the error condition, else free the buffer and exit.

10. Allocated buffer received - if a data buffer give it to requestor by queueing as NOP through QM. If more buffers are required for this unit issue requests until satisfied or suspended. If all requests are satisfied, search SIB's for units with pending requests and attempt to obtain buffers for them. If the buffer is a control buffer restart the suspended device and use the buffer as directed.

11. Start entry process - mark the device as available for use. If the device is a level 3 handler and an ADV has been received, enter the buffer acquisition routine.

All of the above processes exit through •COM•. The IC complete section, ASPHC, will be dispatched whenever an I/O operation completes on the logical unit for ASCII files. Processing follows this sequence:

1. Obtain and save I/O complete entry.

2. Locate designated SIB.

3. Determine event type and dispatch appropriate processing routine.

Unsolicited

4. Perform control function - obtain a control buffer and queue the required operation to the indicated handler.

5. Allocate Device - mark device as active; for level 2 devices issue a START to device handler. For level 3 devices acquire buffers if a START has been received, else exit.

6. Transfer Buffer - If there is an unused buffer available (read/write device) queue it for reading to the indicated handler. If none available set the Read requested flag. For level 3 devices present a type 1 device error if the host system is not connected. In all other cases no processing is necessary.

7. Allocate Buffer - If there is an unused buffer available (read/write device) use it to read from the interconnect kit. If none is available, set the output pending flag. For level 3 devices, present a type 1 device error if the host system is not connected. In all other cases, no processing is necessary.

8. Kill Device Output - purge the queue of the indicated handler and set the SIB to show device inactive. Generate a stop command for associated handler. For a L3 device turn off the active flag and exit.

9. Release device - Generate a stop command and queue to the associated handler. Mark the SIB entry as inactive. For a I3 device turn off the active flag and exit.

Scheduled

10. READ complete - if the stream is inactive or KILLIO status is given, free the buffer. Otherwise queue the buffer for writing to the indicated handler.

11. WRITE complete - if the stream is inactive or KILLIO status is returned, free the buffer. If the device is a read/write device perform the action indicated by the SIB I and O flags (i.e., read from external handler, read from SP, or show buffer as available). For other devices (read only) queue the buffer to the indicated handler for reading.

12. WRITE/CONTROL complete - if the buffer is non-dedicated, free it. Otherwise process like a WRITE completion. In the case of a level 3 device STOP completing, the buffer is queued back to the level 2 handler as a STOP.

Interfacing

The calling sequences and exit procedures for the three entry points to ASFH are:

```
ASFHI
entry:
        LDA         SRFLG       start/restart flag.
        LDB         LUTNO       logical unit for ICK
        JSB         ASFHI
exit:
        JMP         ASFHI,I

ASFHP
entry:
        LDB         ENTAD       entry address
        NOP
        JSB         ASFHP
exit:
        JMP         *COM*

ASFHC
entry:
        LDA         PRMWD       IOC parm word
        LDB         BUFAD       buffer address/status
        JMP         ASFHC
exit:
        JMP         *COM*
```

The ASCII file handler utilizes the services of the buffer manager, queue manager, and allocate/deallocate manager. All device operations are carried out through the services of QIOC and the ASCII file extension of the D.61 driver.

Local data structures

Associated with every ASCII file data stream is a block containing status and control information. The stream information blocks are ordered contiguously by stream id number (0-63). Each block is 6 words long so that any SIB address can be computed by

SIBnn = SIB00 + 6 * (stream id)

The SIB entries are defined as follows:

WORD 1 - STATUS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	-	0
F	O	I	T	W	R	S	A	B	P	X				BC	

F - action required flag
= 1 inclusive OR of bits 13 & 14
= 0 otherwise

O - output requested flag
= 1 if SP output is pending
= 0 otherwise

I - input requested flag
= 1 if SP input is needed
= 0 otherwise

T - level 3 translate mode
= 1 no translation to EBCDIC
= 0 translation to EBCDIC

W - level 3 wait for buffer
= 1 if level 2 module controls buffer when RDV received
= 0 if ASPH had buffer or regained buffer following RDV

R - restart flag
= 1 stream is awaiting allocated control buffer
= 0 stream not held

S - start flag
= 1 a start command has been sent or received
= 0 a stop or purge command has been sent or received

A - activity flag
= 1 an ADV has been received
= 0 an RDV or KDO has been received

B* - buffer availability flag
   = 1 buffer avail
   = 0 buffer in use

P - buffer request pending
   = 1 this stream has a buffer request pending
   = 0 otherwise

X - start timed retry underway for read/write device
   = 1 if yes
   = 0 if no

BC - buffer count
     number of buffers in use for this stream

WORD 2 - Queue name

name of queue to which operations for this device are directed.

WORD 3 - Device Requirements

```
15 - - - - - - - - -12   9 - 8   7   6 - 5   4   3 - 2   1 - 0
 RCD                      DBC    J           L     M       T
```

RCD - restart code for allocated central buffers

     3 - issue START
     4 - issue STOP
     5 - issue PURGE
     6 - issue ERROR
     7 - issue WRITE/CONTROL
     8 - issue CTRL
     9 - issue RETRY

DBC- device buffer count
     number of buffers needed for stream operation.

J - JI device flag
  = 1 - this is the JI device

T - device type flag
  = 0 - read/write CTL only
  = 1 - write only
  = 2 - read only
  = 3 - read/write**

M - mode
  = 0 - for card reader

= 1 - for printer

L - device class flag
= 0 device class is level 2
= 1 device class is level 3

WORD 4 - Buffer Address*

address of buffer for this data stream**
or buffer size

WORD 5 - restart status if waiting for buffer

status word for allocated control buffer

WORD 6 - ASFH code restart address

holds address of location within ASFH to receive control when buffer is obtained

Queuing and control information for inter-handler and handler-driver communications are contained in the buffer control words (BCW's) prefixing all buffers. Content descriptions are found under "Global data structures."

* significant for read/write (type=3) devices only.

**read/write (type 3) devices are always single buffered.

Card Reader Handler

Description and function

This card reader handler (CRH) can be activated by other handlers in order to read cards. The card images and any error indications are returned to the activating handler. CRH supports the reading of cards in ASCII or EBCDIC modes. The programming for this module is serially reusable. CRH is a level 3 function handler and responds to the defined level 2 - level 3 protocol.

Interfacing

CRH accepts these work entry commands:

1. Start - received from a level 2 handler. Causes CRH to activate.

2. Read - received from a level 2 handler. Causes CRH to read a card for the level 2 handler.

3. Stop - received from a level 2 handler. Causes CRH to commence deactivation.

4. Retry command - may be received from the level 2 handler. Causes CRH to retry reading after being previously suspended due to data errors.

5. Allocated buffers - received from the buffer manager. These are buffers which were previously requested but not available at the time of the request.

6. Control - may be received from the level 2 handler. Causes CRH to set device specific actions or options.

CRH returns these commands to its activator:

1. Nop - an allocated read buffer (following a start) is being returned. Also read buffers outstanding at the time of a stop will be returned as nop.

2. Write - a completed read is being returned.

3. Stop - the received stop command is returned following the return of all read buffers and deactivation.

4. Error - an indication that a read error has occurred.

Local data structures

The start command is defined by the following BCW words:

BCW word 2 -	Bits	13-8	stream identifier
		3-0	command=3
BCW word 3 -	Bits	14-0	activating handler queue name
BCW word 7 -	Bits	15-0	buffer length
BCW word 8 -	Bits	15-2	(unused)
		1-0	00 - ASCII 10 - EBCDIC
Data word 1	Bits	15-0	number of buffers

Card images returned by CRH contain the following BCW definitions:

BCW word 2 -	Bits	13-8	stream identifier
		3-0	command=1
BCW word 7 -	Bits	15-0	Number of data bytes (excluding trailing blanks)
BCW word 8 -	Bits	15-2	(unused)
		1-0	00=no errors 11=end of file

All other commands returned to the level 2 handler contain the appropriate stream id in BCW2-bits 13-8. For the error command, BCW8-bits 1-0 indicate the correct general error type.

These symbolic indirect pointers to local data are also used by CRH:

RTQE -	a read retry TQE (timer queue element)
RWQ -	holds name of associated work queue
RW2T2 -	holds the W2T2 queue name or is zero if inactive
RIOCP -	holds configured IOC read parameter
RACBC -	holds the current active buffer count
RSIZE -	retains length in negative bytes for read operations
RQUIT -	holds address of stop buffer if a stop is pending, otherwise zero
RID -	holds stream id in bits 13-8, error condition bits in bits 1-0, and ready/not ready transitions in bit 15.
RTYPE -	zero if a card reader; one if a photo-reader (see description of photo-reader handler)

Algorithm

CRH has four entry points:

CRCHI - handler initialization, invoked by system initialization

CRCHP - receives control from system dispatcher when primed by system queue manager

CRCHC - receives control from system dispatcher when scheduled read operations have completed

RTEX - receives control from the time base generator handler when a timed pause expires (for read retries following not ready states)

At each of these entry points, register A contains the address of the local data structure for some card read function. Subroutine RADDS is used by all four sections to generate a list of indirect addresses for access to the actual data. References to the local data names herein should be viewed as indirect access to the actual data locations.

CRCHI receives the BCS logical unit number for its associated card reader from the system initialization module. This number is merged into RIOCP for use as an IOC parameter word throughout the CRH module. CRCHI then issues a .FIND for its own queue name. This is needed by CRCHP to make .GETQ requests. The queue name is retained in RWQ.

The entire handler is initially in an idle state. It is removed from this state by being primed with a start command containing the queue name of an activating handler as well as that handler's desired mode of reading. Read commands are accepted following the start. Once started in this manner, CRH continues until a stop command appears. Then CRH returns to its idle state. The activating handler can re-activate CRH with a new start command.

CRCHP processes all entries on its queue once primed. It then exits with its prime gate closed. These types of entries may be processed:

1. start - the activator's queue name is retained as well as the indicated stream identifier in RW2T2 and RID. The reading mode is configured into RIOCP. The read

length is saved in RSIZE. Then the desired number of buffers are acquired and returned to the activator.

2. read - using the RSIZE length, a read operation is queued. The active buffer count in 5ACBC is incremented.

3. stop - any pending timed retry is cancelled. Pending reads are purged. If the active buffer count is zero, the stop is immediately returned. Otherwise it is saved in RQUIT for future return.

4. operator command - the device request queue is released.

5. allocated buffer - a read buffer is given to the activator (if still alive). A control buffer is used to dispose of an error message to the activator.

6. control - undefined control parameters are ignored. Others allow switching of read modes.

CRCHC processes completed read operations. The active buffer count is decremented. If a stop is pending, the buffer is returned as a NOP. If any abortive error occurred (purged DRQ), the buffer is returned as a NOP. If no error has occurred, the buffer is returned as a write with end-of-file possibly indicated. If an error has occurred, the buffer is placed back on the work queue for CRCHP to retry. An error message is sent to the activator, and if a not ready type, a timed retry is scheduled.

I. Product Identification

Product abstract

The D.11 card reader driver is designed for use with the HP2000 ACCESS system. It supports both the 7261 and 2892 card readers. (The older 2767 card reader uses an interface identical to that of the 7261. Thus it may also be used. However, this document claims no specific support.) The module is written in a serially reuseable manner. Hence, it will support multiple readers of any mix. (The HP2000 ACCESS system will support up to seven.)

II. Design Overview

Design assumptions

The module assumes a 2100 or 21MX series computer. It also requires the HP2000 ACCESS I/O processor microcode. The 2892 hardware end-of-file feature is supported.

Design summary

The complete driver consists of two distinct modules (D.110 and D.11C). The D.110 module defines the actual driver initiator, continuator, and power recovery entry points associated with any given card reader. This is the only module which must be replicated to support multiple readers. The D.11C module, which contains the actual code, is used by all copies of D.110. In addition, D.110 contains all data of a varying or potentially volatile nature which is required by each reader. This includes related I/O instructions, buffer pointers, reading modes, flags, etc.

Cards may be read by the reader and optionally converted to ASCII or EBCDIC character codes. The original column binary card image may also be requested. All reading is done with DMA.

Power recovery processing allows for proper termination of all scheduled reads even though reader power may be lost. Error status will be appropriately set in this case.

Error handling allows the detection of pick fail, hopper full, data conversion, and other errors. Where the 7261 and 2892 differ, errors are converted to a common base. The occurrance of a "::" (colon-colon) sequence in columns 1 and 2 of a card is treated as end-of-file. For the 2892, hardware end-of-file may be substituted for use of the "::" card.

Design approach

As stated above, the driver is written in a serially reuseable manner. Thus, the one copy of code, D.11C, serves all readers. D.11C accesses a given D.110 storage block through a series of indirect pointers. The use of these pointers is seriallized by D.11C. However, the driver will accept multiple card reader interrupts, queueing each such interrupt for eventual service.

The method of distinguishing between a 7261 and 2892 is via the subunit number (bits 7:6) in EQT word one. Subunit 0 designates a 2892, subunit 1 a 7261.

Module organization

D.11C contains code for I/O initiation (D.11C), I/O continuation (I.11C), and power recovery (P.11C). Since calls to the D.11C section may originate both in the base level program (a JSB to .IOC.) and from IOC (a result of a call to .BUFR), D.11C is forced into serial use by a CLF 0 instruction. In this section, the call parameters are analyzed and a read is initiated. Allocation iof a DMA channel also occurs. If no DMA channel is available, the call is rejected. (Queued .IOC. manages this condition by later rescheduling.)

I.11C, the continuator section, consists of three segments. The first accepts interrupts and queues them for service. It will also dispatch the second segment for the first interrupter only. The second segment processes an interrupt. The third segment dequeues a processed interrupt and reenters the second segment to process additional queued interrupts if any. When no interrupts remain, I/O retires are rescheduled as required. Note that only segments one and three need to run disabled.

The P.11C power recovery section is entered for each card reader (via P.110) from D.04. If the associated I.110 section was active, D.04? is used to restore this condition. However, if the reader was not having an interrupt processed, but a read had been scheduled, an interrupt is forced to cause normal I.11C error processing to effect recovery.

Interrupt service

The second segment of I.11C first establishes the driver's indirect pointer list to access the D.110 storage block for the reader whose interrupt is being processed. Next, error detection is done using a different status examination procedure for each

reader type. For the 7261 only, up to 10 retries are attempted for pick failures before the error is considered "hard." The 2892 hardware end-of-file status is recorded. (It may be overridden by simultaneous occurrence of a "::" card.)

Next the significant data count is obtained using the DMA length control word. Note that this allows the 40 column feature of the 7261 to be supported. Since the 7261 and 2892 column images are oppcsite, the 7261 image is reversed at this point. Row 9 will appear in computer word bit 0 and the "twelve" row in word bit 11.

The reading mode is examined next. Note that reading does not occur directly into the caller's buffer. Rather, an intermediate 80 word buffer in D.110 is used. The data is now moved and converted from this intermediate buffer into the caller's buffer. The caller's length is honored and overrides the actual significant read lergth. In all cases, the returned transmission log reflects the number of columns read less any trailing blanks. This count may be less than the requestor's read length. Data is moved to the requestor's buffer after possible conversion to ASCII or EBCDIC character representations. The detection of any erroneous punch combinations in this case will result in a data error status. The completed read operation is now signalled via .BUFR, and the used DMA channel is released.

III. Design Structures

Interface descriptions

Clear

Any current operation is aborted. DMA is released. The driver enters an idle, available state.

```
JSB .IOC.
OCT 0000xx          (xx=logical unit #)
```

Read

A card is read and transferred to the supplied buffer. Optional data conversion may occur.

```
JSB .IOC.
OCT 010nxx          (xx=logical unit #)
JMP REJCT
DEF BUFFR
DEC m
```

m=positive number of desired card columns
n=0 - ASCII read mode
  1 - column binary read mode
  2 - EBCDIC read mode

Status information

The returned transmission log always reflects the number of card columns requested less any trailing blanks. The status word includes these bit definitions:

```
14 = error
1:0 = 0 - no error
      1 - pick fail or other not ready condition
      2 - data error
      3 - end-of-file
```

Line Printer Handler

Description and function

The Line Printer Handler is responsible for normal flow of print lines to the printer and notification of its caller of printer errors and printer completion (end-of-file; obtained from calling program). The Line Printer Handler will effect the emptying of buffers through the queue manager and calls to .IOC. to print user output on the line printer. The handler will adhere to the defined level 2-level 3 protocol. This module is written in a 'serially reusable' manner with all locally used data accessed from a unique storage block. Thus, additional printers may be added by replicating this storage block.

The handler has three entry points; a "prime" entry which is used when a queue entry exists and is to be processed and an "IO complete" entry which is used when an operation initiated by the "prime" entry, has finished and the queue entry processed must be returned to the proper place; and an initialization entry called during system startup and restart.

The "prime" entry will obtain an entry from the input queue and attempt to do a .IOC. call to process the buffer.

The "IO complete" entry will examine the returned information from a completed operation and determine if an error occurred. If an error occurred, the handler will issue a priority .IOC. call to reschedule the request and inform the level 2 handler of the error. If no error occurred, the entry will be returned to the proper handler queue.

Interfacing

Input queue entries may be one of the following:

1. Start - received from a level 2 handler in a control buffer. It activates the handler.

2. Write - received from a level 2 handler. Contains data to be output to the printer.

3. Stop - received from a level 2 handler in a print buffer. It causes the handler to begin termination processing.

4. Allocated buffer - received from the buffer manager. The buffer is received as a queue entry if no buffers were available at the time of the initial request.

5. Control operation - recieved in a control buffer by a level two handler. It is used to perform special carriage control operations when in the print-then-space mode. The control buffer is freed by this handler.

6. Start timed retries - received from a level 2 handler when the printer has indicated a not ready condition.

7. Purge - received from a level 2 handler. A clear IOC call is made to purge any outstanding write requests.

Output queue entries may be any of the following:

1. NOP - an allocated write buffer or a completed write buffer is being returned to the caller.

2. Stop - the stop command received by this handler is being returned to the caller to indicate termination processing is complete.

3. Error - a notification that a write error has occurred.

The start command is defined as follows:

BCW word 2 -	Bits 13-8	stream identifier
	3-0	command=3
BCW word 3 -	Bits 14-0	activating handler queue name
BCW word 7 -	Bits 15-0	buffer length
BCW word 8 -	Bits 1-0	00=space then print
		01=print then space
Data word 1	Bits 15-0	number of buffers

The print buffer is defined as follows:

BCW word 2 -	Bits 3-0	command=1
BCW word 7 -	Bits 15-0	length
Data word 1-	Bits 15-0	print-then-space, not used
		space-then-print, control
	Bits 1-0	zero

Algorithm

ERSWT is set when a printer error occurs and is not reset until a line is successfully printed. This mechanism is used to provide only one error notification to the original sender when a printer malfunction occurs. PRBUF is used to store the address of the printer buffer in case suspension due to buffer not available occurs during error processing.

The four entries to the handler are: LPHI, the initialization entry; LPHP, the primer entry; LPHC, the I/O complete entry; and LPHT, the timer expiration entry. At each entry point, subroutine PADDS is used to create a list of indirect addresses to the local data structure.

In the initialization section LPHI, the logical unit number of the device is saved and the handler's queue name is found and saved.

LPHP handles one queue entry at a time when it is primed. The queue entries are processed as follows:

1. start - the activator's queue name, the stream ID, and the print mode are saved. The required number of buffers are obtained and returned to the activator. In print-then-space mode, a write IOC call is made with control code equal to one to cause a skip to top-of-form.

2. print - in space-then-print mode, the PCF control code is moved from Data word 1 to BCW9 and the IOC call is issued. In print-then-space mode, the control code saved in LPCON is moved to BCW9 and the IOC call is issued. LPCON is then reset to the default control code (space 1).

3. control - the PCF control code is saved at LPCON and the control buffer is freed.

4. stop - the print buffer containing the stop command is used to issue an IOC call with length equal to 0 and BCW9 control code equal to 3 (space 1).

5. allocated buffer - the buffer may be a control buffer or a print buffer. The control buffer is used to send an error indication to the activator when a not ready condition occurs. The allocated print buffer is made

interlocked, the stream ID and NOP command are placed in BCW2 and it is sent to the activator.

6. start timed retry - a timer is started and the buffer is freed. In the timer exit, a release IOC request is made to reinitiate the operation.

7. purge - a clear IOC request is made and the buffer is freed.

When control is received with indication of an I/O complete event the following sequence of events occurs: The status of the request is checked to determine success of this operation. If successful; bits 15 & 14 of register A or word 3 of the buffer prefix will be zero. In this case, the buffer is returned to the activator with command code=0. If availability is set to one the device was not ready and the handler will issue a priority .IOC. call to requeue the unprocessed buffer. A control buffer is obtained and sent to the activator with command code=6 to indicate the occurrence of the error.

I. Product Identification

Product abstract

The D.12 line printer driver is designed for use with the TSB/2000G System and will support both post-spacing and pre-spacing print modes for the 2607, 2610, 2613, 2614, 2618, 2767, and 2778 line printers.

II. Design Overview

Design assumptions

This driver is designed to run on a 2100 series computer and requires any one or more of the interfaces for the printers listed above. It also requires the microcode described in the Phase II Base Design.

Design summary

Two modes of operation are supported. In the pre-spacing print mode, the carriage control will be issued and then the characters in the buffer will be transferred to the printer. This line will not actually be printed until the next control operation is performed (carriage control for the next line). Therefore, it will be necessary to issue a control operation to the driver when end-of-file is reached.

A post-spacing mode will also be supported. In this mode, the characters in the buffer will be transferred to the printer before the control operation is issued.

All printer types listed above will be supported by this driver. The appropriate control function routine is determined by the unit number in the EQT.

The following functions will be performed by D.12:

CLEAR - reset all status words and issue a clear control request to the printer.

WRITE - transfer a block of data to the printer and issue a control operation either before or after the characters are transferred.

Design approach

This driver is written in a serially reusable manner. In order to simplify the addition of printers to the system, the driver is composed of two separate and distinct modules. The first contains the unique entry points and the local data storage for a given printer. When another printer is added to the system, only this code is replicated.

The second mudule is the serially reusable code used by all copies of the first module. This common code, D.12C, is composed of three basic units; the initiator section, the continuator section, and the power recovery section. The initiator section is entered on an IOC call and is all disabled code. This is necessary because it may be entered by .BUFR on the completion of a print line for any printer driver. The continuator section handles one printer interrupt at a time. If another interrupt occurs while this section is in use, the interrupt will be queued and processed when the active one is completed. In the power recovery section, the status of the printer driver is checked and a set control is issued if required to create an interrupt.

Design structures

Major functional units

CLEAR

Description

The printer is cleared with a CLC instruction and all status words are cleared.

Interface description

```
JSB .IOC.
OCT 0000XX        XX is the logical unit number
```

No rejects are possible.

Algorithm description

1. Get printer status.
2. Issue CIC to the printer.
3. Update status word in the EQT.

4. Clear driver status flags.

5. Return to caller.

WRITE

Description and function

Transfer a block of data to the printer. The control operation is issued before or after the characters are transferred depending on the subfunction code in the IOC call.

The control operation is translated to the appropriate format control code required by the printer interface. The proper routine for the translation is determined by the unit number in the EQT. The EQT unit numbers are defined as follows:

Unit Number	Printer Type
0	2607, 2610, 2614
1	2613, 2618
2	2767
3	2778

The control code for each printer line is located in Bits 15-8 of BCW 9. Control codes are defined as follows:

Code	Action Requested	Normal Effect
0	end-of-file	ignored by driver
1	skip to channel 1	top of form
2	skip to channel 2	bottom of form
3	skip to channel 3	next line
4	skip to channel 4	next double line
5	skip to channel 5	next triple line
6	skip to channel 6	next half page
7	skip to channel 7	next quarter page
8	skip to channel 8	next sixth page
9	skip to channel 9	next line
10	skip to channel 10	next line
11	skip to channel 11	next line
12	skip to channel 12	next line
13	suppress spacing	suppress paper advance
14	true single space	same
15	true double space	same
16	true triple space	same

Channels 9 through 12 exist on line printer models 2613 and 2618 for optional installation definition of additional VFU capabilities. They are identical tc channel 1 on the factory-supplied VFU tape. These channels do not exist on the 2607, 2610, and 2614 models and these ccdes will result in the 'next line' action for these printers.

Interface specifications

```
JSB .IOC.
OCT 020YXX          where XX is logical unit number
                    and Y=0 for pre-spacing print mode
                        Y=1 fcr post-spacing print mode
Reject Address
Buffer Address
Buffer length       (b;  s, ..gati.e)
```

The call may be rejected if the driver is busy. If so, the return is made with:

```
A=1
B=100000
```

The transmission log will reflect the number of characters transmitted to the printer.

Algorithm description

If the driver is not busy, the print operation is started in the initiator section of the driver. Either the format control word (pre-spacing) or the first character in the buffer (post-spacing) is output to the printer.

In the continuator section, the next character is output to the printer. When the buffer is exhausted, the format ccntrcl word is output (post-spacing mcde) to the printer and then .BUFR is called. Interrupts that cccur while this section is in use are queued for later processing. Before exiting, this queue is checked for work and all queued interrupts are processed.

All commands to the printer are channel type commands. Single and double spacing are accomplished with 'skip to channel 3' and 'skip to channel 4' ccmmands respectively.

Paper Tape Punch Handler

Description and function

The Paper Tape Punch Handler is responsible for normal flow of output to the paper tape punch and notification of its caller of errors and completion (end-of-file; obtained from calling program). The Paper Tape Punch Handler will effect the emptying of buffers through the queue manager and calls to .IOC. to punch user output on the paper tape punch. The handler is written in a 'serially reusable' manner and will adhere to the defined level 2-level 3 protocol.

The handler has four entry points; a "prime" entry which is used when a queue entry exists and is to be processed; an "IO complete" entry which is used when an operation initiated by the "prime" entry has finished and the queue entry processed must be returned to the proper place; a timer entry used when a timer expires; and an initialization entry called during system startup and restart.

The "prime" entry will obtain an entry from the input queue and attempt to do a .IOC. call to process the buffer.

The "IO complete" entry will examine the returned information from a completed operation and determine if an error occurred. If an error occurred, the handler will issue a priority .IOC. call to reschedule the request and inform the level 2 handler of the error condition. If no error occurred, the entry will be returned to the proper handler queue.

Interfacing

Input queue entries may be one of the following:

1. Start - received from a level 2 handler in a control buffer. It activates the handler.

2. Write - received from a level 2 handler. Contains data to be output to the paper tape punch.

3. Stop - received from a level 2 handler in a write buffer. It causes the handler to begin termination processing.

4. Allocated buffer - received from the buffer manager. The buffer is received as a queue entry if no buffers were available at the time of the initial request.

5. Start timed retries - received from a level 2 handler when the paper tape punch has indicated a not ready condition.

6. Purge - received from a level 2 handler. A clear IOC call is made to purge any outstanding write requests.

Output queue entries may be any of the following:

1. NOP - an allocated write buffer or a completed write buffer is being returned to the caller.

2. Stop - the stop command received by this handler is being returned to the caller to indicate termination processing is complete.

3. Error - a notification that a write error has occurred.

The start command is defined as follows:

BCW word 2 - Bits	13-8	stream identifier
	3-0	command=3
BCW word 3 - Bits	14-0	activating handler queue name
BCW word 7 - Bits	15-0	buffer length
BCW word 8 - Bits	1-0	00=punch mode in Data word 1
		01=punch mode changed by control operation
Data word 1 Bits	15-0	number of buffers

The write buffer is defined as follows:

BCW word 2 - Bits	3-0	command=1
BCW word 7 - Bits	15-0	length - positive bytes
Data word 1- Bits	15-8	punch mode - RJE
	7-0	zero

Algorithm

The algorithm for processing is given below:

ERSWT is set when a punch error occurs and is not reset until a line is successfully printed. This mechanism is used to provide only one error notification to the original sender when a device malfunction occurs. BUFFR is used to store the address of the

write buffer in case suspension due to buffer not available occurs during error processing.

The four entries to the handler are: PPCHI, the initialization entry; PTHP, the primer entry; PPCHC, the I/O complete entry; and PPCHT, the timer expiration entry.

In the initialization section PPCHI, the logical unit number of the device is saved and the handler's queue name is found and saved.

PPCHP handles one queue entry at a time when it is primed. The queue entries are processed as follows:

1. start - the activator's queue name, the stream ID, and the mode are saved. The required number of buffers are obtained and returned to the activator. An ICC call is made to create the tape leader.

2. write - an IOC call is made to output the characters in the buffer.

3. stop - the output buffer containing the stop command is used to issue an IOC call which causes the driver to emit a trailer.

4. allocated buffer - the buffer may be a control buffer or a write buffer. The control buffer is used to send an error indication to the activator when a not ready condition occurs. The allocated write buffer is made interlocked, the stream ID and NOP command are placed in BCW2 and it is sent to the activator.

5. start timed retry - a timer is started and the buffer is freed. In the timer exit, a release IOC request is made to reinitiate the operation.

6. purge - a clear IOC request is made and the buffer is freed.

When control is received with indication of an I/C complete event the following sequence of events occurs: The status of the request is checked to determine success of this operation. If successful; bits 15 & 14 of register A or word 3 of the buffer prefix will be zero. In this case, the buffer is returned to the activator with command code=0. If availability is set to one the device was not ready and the handler will issue a priority .IOC. call to requeue the unprocessed buffer. A control buffer is

obtained and sent to the activator with command code=6 tc indicate the occurrence of the error.

I. Product Identification

Product abstract

The D.13 tape punch driver is designed for use with the HP2000 ACCESS system. It supports the HP 12926A paper tape punch subsystem. The module is written in a serially reuseable manner. Hence, it can support multiple tape punches. (The HP2000 ACCESS system will support up to seven.)

II. Design Overview

Design assumptions

The module assumes a 2100 or 21MX series computer. It also requires the special microcode develcped for use in the I/O Processor of the HP2000 ACCESS system.

Design summary

The complete D.13 driver consists of two distinct modules (D.130 and D.13C). The D.13C module defines the actual driver initiator, continuator, and power recovery entry points associated with any given tape punch. In addition, it contains all storage (constants, pointers, I/O instructions, etc.) which is unique to that given tape punch. The D.13C module contains the actual code used to drive the device. It is called by the D.130 module and then utilizes the unique storage provided by the caller to associate itself with a specific tape punch.

The operation of the tape punch includes provisions for punching leaders and trailers and for punching data in record mode or transparent mode. In the record mode, each buffer of data is punched and an X-OFF, carriage return, line feed sequence is then appended by the driver. Each record may be punched with even, odd or no parity. The parity punches are supplied by the driver. (Except for no parity for which no alteration of level eight occurs.) In the transparent mode, no modifications to the data are made.

Power recovery operation includes the punching of a record delete sequence (control-X, carriage return, line feed). Also, any record currently being punched is restarted.

Error handling allows only for a "not ready" error indication. A non-process timeout is a provision of the driver intended to make up for the general lack of status bits in the tape punch interface. Each time a tape frame is punched (i.e.

started), a 1 second timer is started. If the timer expires prior tc a punch complete interrupt, the punch is considered down and a "not ready" error is set. The occurrence of a low tape supply also causes a "not ready" error.

Design approach

As stated above, the driver is written in a serially reuseable manner. This means that a single copy of D.13C (the code) is used by multiple copies (and, hence, multiple tape punches) of D.130. Access tc the storage in a given D.130 module by D.13C is done via indirect addressing through an indirect pointer list. This pointer list can only address one D.130 module at a time. D.13C is thus forced to run serially to preserve the integrity of this list. Later descriptions will show, however, that this does not impare the driver's ability to handle multiple interrupts from different tape punches.

The following documentation will confine itself to a description of the D.13C module.

Module organization

D.13C ccntains code for I/O initiation (D.13C), I/O continuation (I.13C), power recovery (P.13C), and ncn-process timeout handling (T.13C). Since calls to the D.13C section may originate both in the base level program (a JSB to .IOC.) and from IOC (a result of a call to .EUFR), D.13C is forced into serial use by a CLF 0 instruction. In this section, the I/O call parameters are analyzed and established, and the initial I/O operation is begun.

I.13C, the continuator section, consists of three segments. In the first segment, an interrupt is accepted, and the D.130 storage for the associated punch is enqueued to a linked list of storage blocks which require interrupt service. This is done without disturbing any of the module's indirect pointer list. If this enqueueing process determines that the interrupting punch is not the first (at least one other storage block is already enqueued), then an interrupt exit is immediately made. This is possible because the first storage block to be enqueued will cause entry to the second segment cf I.13C which will begin to process all such interrupts.

In the second segment, the top D.130 storage block on the interrupt service queue is processed. The indirect pointer list is set for this block, and the interrupt is processed. (See discussion of actual interrupt service below.) Following

interrupt service, the third I.13C segment receives control. It disables the system and removes the top D.130 storage block from the interrupt service queue. If other blocks are still queued, the second segment of I.13C is entered again. When all D.130 storage blocks have been serviced and dequeued, required additional I/O is begun for all punches needing same. Then an exit from the original (very first) tape punch interrupt is made.

The P.13C power recovery section of the module is entered for each tape punch (via P.130) at power recovery time. It is called as a subroutine from D.04 and takes appropriate action to restore service. P.13C always emits a record delete sequence consisting of control-X, carriage return, line feed. This is to cancel the effect of any spurious punches associated with power restoration at the punch itself. (This sequence is punched with even parity, since it may be impossible to determine the mode in current use.) Finally, the proper state of the device's flag and control bit settings is restored.

Since the driver uses timed punch operations to detect "down" punches, a common T.13C non-process timeout exit is a part of the driver. It receives control from the D.43 time base generator module. T.13C first checks for a set flag on the punch. This would indicate that an interrupt is pending but has not yet been allowed. No error is indicated in this case. However, if the flag is clear, the non-process timeout error bit is set, and an interrupt is forced in order to effect recovery by the interrupt processor.

Interrupt service

The second segment of I.13C processing is the interrupt service segment. Interrupts occur either because of a scheduled operation completing, or because an interrupt is forced when a non-process timeout occurs.

The first action of the interrupt service code is to acknowledge the interrupt with a CLC instruction. Next, errors are checked. If a non-process timeout or low tape supply exists, the operation is immediately terminated with a call to .BUFR. If no errors have occurred, the power recovery flag is tested. If set, the original I/O operation parameters are restored so as to effect a complete restart of the current operation in the remainder of the processing. Finally, the character output routine is invoked to select and output the next appropriate frame. If no more data remains, the operation is instead terminated via .BUFR.

The character output routine is used both by D.13C to output the first frame and by I.13C for subsequent frames. The routine first checks for operation end, updating buffer pointers and lengths in the process. Then the mode is checked. Either a feed frame is selected or a record datum. In the case of record data, parity is applied as required. Additionally, at record end for ASCII modes, an X-OFF, carriage return, line feed sequence is appended. When the character has been selected and output, the type of entry is checked. If entered from D.13C, an STC instruction and its timer are invoked to start the punching. If an I.13C entry, the STC instruction is enqueued for future execution by the third segment of I.13C. This is to prevent an interrupt on a punch whose D.130 storage block is engaged in servicing other tape punch interrupts. (That is, it was the original interrupter.)

III. Design Structures

Interface descriptions

Clear

Any current operation is aborted. The driver enters an available and idle state.

```
JSB .IOC.
OCT 0000xx          (xx=logical unit #)
```

Write

A block of data is transferred to paper tape. Optionally, a leader (or trailer) may be produced. All such operations are scheduled with a basic write command with appropriate mode settings for the various punch types.

```
JSB .IOC.
OCT 02nnxx          (xx=logical unit #)
JMP REJCT
DEF DATA
DEC m
```

m=number of bytes of data to be output (or number of leader/trailer frames) in plus bytes

nn=00 - ASCII mode with even parity
01 - ASCII mode with odd parity
02 - ASCII mode with no parity
03 - transparent mode

20 - leader/trailer mode

In the nn=20 case, the data buffer address is insignificant. For the 00 to 02 modes only, X-OFF, carriage return, line feed is appended by the driver to all records. No parity means that no alteration of the eigth level cccurs. It is forced neither high nor low. It remains as supplied by the caller.

Status information

The transmission log always returns the original length of the caller's request. This is to facillitate retries. The status bits include bit 14 which indicates the occurance of some error and bit 0 which is one to indicate the occurance of a punch "not ready" conditicn.

Photo-Reader Handler

Description and function

The programming for this handler is the same as that for the card reader handler. Where it is necessary to distinguish between a card reader and a photo-reader, the code tests a variable in the respective local storage to determine the appropriate action. Actions specific to the photo-reader are outlined below.

Algorithm

In CRCHI, the photo-reader type causes an alternate device designator to be used in finding the associated queue name.

In CRCHP, these alternate command processes occur:

1. start - differing read mode indicators are established in the .IOC. parameter word

2. control - different set of control parameters is allowed to set varying .IOC. parameters.

In CRCHC, if EBCDIC reading is selected, the translation from ASCII to EBCDIC occurs at this level.

I. Product Identification

Product Abstract

The D.14 paper tape photo-reader driver is designed for use with the 2000 ACCESS system. It supports the 2748 reader. Because it is written in a serially reusable manner, it will support multiple readers. (The 2000 ACCESS system will support a maximum of seven)

II. Design Overview

Design Assumptions

The module assumes a 2100 or 21MX series computer. It also requires the 2000 ACCESS Firmware option (13206A or 13207A).

Design Summary

The complete driver consists of two distinct modules (D.140 and D.14C). The D.140 module defines the actual driver initiator, continuator, timer, and power recovery entry points associated with any given reader. This is the only module which must be replicated to support multiple readers. The D.14C module, which contains the actual code, is used by all copies of D.140. In addition, D.140 contains all data of a varying or potentially volatile nature which is required by each reader. This includes related I/O instructions, buffer pointers, flags, etc.

The driver supports four modes of tape reading. Three modes provide for varying types of ASCII data formats, and the fourth mode allows transparent reading. The only difference between the ASCII modes is in the form of parity testing (even, odd, or none). In the transparent mode, any and all frame punch combinations are valid.

Power recovery processing allows for proper termination of all scheduled reads even though reader power may be lost. Error status will be appropriately set in this case.

Error handling allows the detection of parity errors for appropriate reading modes. Since the reader interface has no inherent "not ready" status, all read operations are timed, and a read timeout is used to effect not ready errors. For the ASCII reading modes, an end of file condition is defined by the occurrence of 80 or more consecutive null frames.

Design Approach

Since the driver is written in a serially reuseable manner, one copy of code (D.14C) serves all readers. D.14C accesses a given D.140 storage block via a series of symbolic indirect pointers. The use of these pointers is serialized by D.14C. However, the driver will accept multiple reader interrupts, queueing each for eventual service.

Module Organization

D.14C contains code for I/O initiation (D.14C), I/O continuation (I.14C), power recovery (P.14C), and timer expiration (T.14C). Since calls to the initiator section may originate from both a base level program (a JSE to .IOC.) and from IOC (a result of a call to .BUFR), D.14C is forced into serial use by a CLF 0 instruction. In this section of the driver, the call parameters are analyzed and a read may be initiated.

I.14C, the continuator section, consists of three segments. The first segment runs disabled and enqueues the interrupting I.140 module for eventual interrupt service. The first such module enqueued causes the second segment to be entered. Segment two removes an enqueued I.140 module and establishes the indirect symbolic pointers to the module data. Then the system can be enabled while the interrupt is serviced. Interrupt service includes accepting the input datum and adding it to the caller's buffer. Parity, end of file, and other tests are perfcrmed. At the end of interrupt service, segment three gains control. The system is disabled, and the module is dequeued. If other modules require service, segment two will be re-entered.

P.14C gains control from the D.04 power fail/restart module for each P.140 entry point. If the associated I.140 section was active, D.04? is used to restore this condition. However, if the reader was inactive (I.140 iractive) but a read had been started, an interrupt is forced to cause I.14C to effect recovery.

T.14C gains control if a given T.140 timer has expired. This indicates a failing read. Urless an interrupt is pending, one is forced to cause normal recovery.

Module Detail

These symbolic indirect pointers are used in D.14C to access D.140 data:

PREQT - holds related EQT address

```
PRIOR - holds address of last .IOC. parameter list
PFLAG - driver flags:

                  15 - busy driver
                  14 - skipping for CR
                  13 - resume reading after control-X
                  12 - 1=data has been read
                10:8 - device status
                              0=ready
                              1=read time out
                              2=unrecoverable read error
                              3=data error
                              7=end of file
                 2:1 - 0=even parity
                       1=odd parity
                       2=no parity
                   0 - 1=transparent mode (bits 2:1 ignored)
                       0=ASCII mode (bits 2:1 apply)
PBUFR - holds byte address of input
PLENG - holds negative read length
PLL   - holds original read length
PEOFL - holds negative count for EOF detection
PSTC  - STC subroutine (uses PFRIO and therefore has STC flag
        word)
PTQE - a four word TQE for read timing
PCLC - associated CLC instruction
PLIA - associated LIA instruction
PSFS - associated SFS instruction
PSTF - associated STF instruction
```

These data are specific to D.14C:

```
PIOQ - a queue for pending STC instruction
PEXIT- pointer to interrupting I.140 entry point
PINQ - a queue for interrupt service
I.14 - pointer to 1st interrupting I.140 entry point
.BUFP- .BUFR EQT parameter
```

The following "structured" description provides internal module detail.

```
D.140:    JSB to D.14C;
D.14C:    disable; save registers;
          establish indirect pointers to D.140 storage
          using contents of D.14C as base address;
          transfer contents of D.140 to D.14C;
          restore registers;
          store A into PREQT and B into PRIOR;
```

```
        load and isolate bits 14:12 of I/O request
        parameter in register A (PRIOR indirect);
        if A=0 (clear) then do;
              issue CLC to stop interface;
              purge timer using .LTC.; disable on return;
              clear driver flags (PFLAG);
              set EQT status to zero;
              exit via D.14C;
              end;
        if PFLAG bit 15 is set then do;
              set A=1 to show reject;
              set B=100000 to show device busy;
              enable interrupts;
              exit via D.14C;
              end;
        if A=1 (read) then do;
              load B from PRIOR;
              load buffer address into A(LAI+2);
              convert to byte address and save in PBUFR;
              load read length into A(LAI+3);
              copy into PLL;
              negate and save in PLENG;
              load request parameter (LAI+0);
              isolate bits 8:6
              and align to A register 2:0;
              merge A into PFLAG;
              set PFLAG bit 15 to show driver busy;
              clear PFLAG bit 12 to show no
              data yet read;
              set EQT word 2 bit 15 to show EQT busy;
              set PEOFL to -80 for possible EOF detection;
              JSB to R.14C to start read operation;
              clear A to show read started;
              enable interrupts;
              exit via D.14C;
              end;
        set A=1 to show reject;
        set B=0 to show illegal request;
        enable interrupts
        exit via D.14C;

R.14C:  RSS if D.14C in control via .IOC. call;
        JMP TO R.14E (if D.14C in control while I.14C active);
        JSB to PSTC;
        call .LTS. with PIQE;
        disable on return from .LTS.;
        return;
```

```
R.14E:    load address of PIOQ into A;
          load address of PSTC into B;
          enqueue PSTC to PIOQ;
          return;

I.140:    JSB to I.14C;
I.14C:    disable interrupts; save registers;
          move address of I.140 to PEXIT;
          clear PSTC flag for power recovery;
          generate address of associated TQE in register A;
          call .LTC. to cancel timer, and disable interrupts on return;
          load contents of I.14C into B and decrement B;
          load address of PINQ into A;
          enqueue I.140 storage to PINQ;
          if not first storage enqueued then do;
                              restore registers;
                              exit using PFREI to restore
                              interrupts and exit via PEXIT;
                              end;
          move PEXIT to I.14;
          change RSS at R.14C to nop;

E.14C:    enable interrupts;
          load address of enqueued I.140 storage (PINQ);
          establish contents of indirect pointers;
          issue CLC to acknowledge interrupt (PCLC);
          if PFLAG bits 9:8=1 or 2 then go to E.14C;
          if PFLAG bit 0 is set then do;
                use PLIA to load datum;
                load B from PBUFR;
                store byte from A via B and store B into PBUFR;
                increment PLENG;
                if zero then go to E.14C;
                JSB to R.14C for next frame;
                go to F.14C;
                end;
          else do;
                use PLIA to load datum;
                if PFLAG bit 12 is clear and A=0 then do;
                      increment PEOFL;
                      if zero then do;
                            set PFLAG bits 10:8;
                            go to E.14C;
                            end;
                      JSB to R.14C;
                      go to F.14C;
                      end;
                if PFLAG bit 2 is clear then do;
```

```
                calculate number of bits in A;
                if even and PFLAG bit 1 is set
                or if odd and PFLAG bit 1 is clear
                then do;
                      set PFLAG bits 9:8;
                      end;
          mask A to seven bits;
          if A=carriage return then go to E.14C;
          if PFLAG bit 14 set then do;
                JSB tc R.14C;
                go to F.14C;
                end;
          if A=0 [illegible] A=177
          then do;
                JSB [illegible];
                go to F.14C;
                end;
          if A=30 then do;
                restore original length;
                set original buffer pointer;
                set PFLAG bits 14 and 13;
                JSB tc R.14C;
                go to F.14C;
                end;
          if A=10 then do;
                reduce length;
                reduce buffer pointer;
                JSB tc R.14C;
                go to F.14C;
                end;
          load B from PBUFR;
          store byte from A and restore B to PBUFR;
          set PFLAG bit 12 to show data read;
          increment PLENG;
          if zero then do;
                set PFLAG bit 14;
                end;
          JSB to R.14C;
          go to F.14C;

E.14C:    load PFLAG into A; if PFLAG bit 13 is set
          then do;
                clear PFLAG bits 14 and 13;
                JSB to R.14C;
                go to F.14C;
                end;
          isolate bits 10:8 of A into bits 2:0 of A;
          if A not 0 then set A bit 14;
```

```
            store A into EQT status word;
            add PLL and PLENG and move to
            EQT transmission lcg;
            move PREQT to .BUFP;
            JSB to .BUFR with return to F.14C;

F.14C:      disable interrupts;
            dequeue I.140 storage from PINQ;
            if PINQ not empty then go tc B.14C;

G.14C:      dequeue PSTC from PIOQ;
            if none then do;
                        restore RSS at R.14C;
                        restore registers;
                        exit using PFREI to restore
                        interrupts and exit via I.14;
                        end;
            JSB to the dequeued PSTC;
            call .LTS. to start associated timer;
            disable on return frcm .LTS.;
            go to G.14C;

T.140:      JSB to T.14C;
T.14C:      use TQE address tc generate address of
            asscciated SFS;
            execute SFS;
            if flag set then exit via T.14C;
            generate address cf PFLAG;
            set PFLAG bit 8;
            generate address cf PSTC and execute same;
            generate address cf PSTF and execute same;
            exit;

P.140:      JSB to P.14C;
P.14C:      set PFLAG bit 9;
            if I.140 not 0 then do;
                        set B=-1;
                        load PLIA into A;
                        call I.04?;
                        exit;
                        end;
            if PSTC flag = zerc then exit;
            load PSTC instructicn;
            change to STC (no clear flag);
            execute in-line;
            exit;
```

III. Design Structures

Interface Descriptions

Clear

Any current operation is aborted. The driver enters an idle, available state.

```
JSB    .IOC.
OCT    0000xx       (xx=logical unit #)
```

Read

A record is read from paper tape and transferred to the supplied buffer.

```
JSB    .IOC.
OCT    010nxx       (xx=logical unit #)
JMP    REJCT
DEF    BUFFR
DEC    m
```

m=positive number of frames to be read
n=0 - even parity ASCII mode
  1 - transparent mode
  2 - odd parity ASCII mode
  4 - no parity ASCII mode

Status Information

The returned transmission log is always equal to the number of data frames read. The status word includes these definitions:

```
 14 - error
2:0 - 0 no error
      1 reader not ready
      2 power failure
      3 parity error
      7 end of file
```

Reader Punch Interpreter Handler

Description and Function

The reader/punch/interpreter handler (RPH) manages all I/O for the reader punch driver. It may be activated by other handlers to read cards, punch cards, or print on cards. When activated by the ASCII files handler it may be used for both read and write operations. When activated by an RJE module it will be used either as a card reader handler or a card punch handler, but not both simultaneously.

The handler supports the reading of cards in column binary, ASCII, or EBCDIC. It supports punching cards in Hollerith or column binary. The handler is written as a serially reusable module. It is a level 3 function handler and responds to the defined level 2-level 3 protocol.

Interfacing

RPH has one input queue for all work entries. These commands may appear on the input work queue:

1. Start - received from a level 2 handler. Activates the handler.

2. Read - received from a level 2 handler. The handler uses the buffer to read a card.

3. Write - received from a level 2 handler. The buffer contains data to be punched and/or printed.

4. Control - received from a level 2 handler in a control buffer. It contains a PCF code in the data word.

5. Stop - received from a level 2 handler and causes the handler to begin termination processing.

6. Retry - received from a level 2 handler and causes the handler to retry the read or write operation.

7. Purge - received from a level 2 handler. A clear call is made to IOC to purge any outstanding I/O requests.

8. Allocated buffers - received from the buffer manager in delayed response to a request for a buffer.

The following commands may be returned to the activator:

1. NOP - a completed write buffer is being returned to the caller, or an allocated buffer is being sent to the activator.

2. Write - a completed read buffer is being returned to the caller.

3. Stop - the received stop command is being returned following the completion of all outstanding I/O requests.

4. Error - an indication that a read or write error has occurred.

Local Data Structures

The start command is defined by the following BCW words:

```
BCW word 2 - Bits 13-8 stream identifier
BCW word 3 - Bits 14-0 activating handler queue name
BCW word 7 - Bits 15-0 buffer length
BCW word 8 - Bit 15    start for read only or write only
                       (RJE mode)
                 1-0   read mode (if bit 15 on)
                       00=ASCII
                       01=column binary
                       10=EBCDIC
Data word 1 - Bits 15-0 number of buffers
```

Card images returned to the activator have the follwcing BCW definitions:

```
BCW word 2 - Bits 13-8 stream identifier
                  3-0  command=1
BCW word 7 - Bits 15-0 number of data bytes
                       (excluding trailing blanks)
BCW word 8 - Bits 15-2 unused
                  1-0  00=no errors
                       11=end of file
```

For the error command, BCW2-bits 13-8 contain the stream identifier and BCW8-bits 1-0 contain the general error type.

The following define indirect pointers to local data:

```
TQE   TQE
QNAME RPH queue name
```

```
AHQNM activating handler queue name
RIOCP read IOC parm
WIOCP write IOC parm
CIOCP control IOC parm
STAT  error status
STID  stream ID
RLEN  read length
BUFFR buffer address
STOPB stop buffer address
ACTBC active read buffer count
FLAGS flags
      bit 15 not ready error condition
          14 purge active
          11 stacker selection
          10 hopper selection
           0 1=RJE mode
             0=TSB mode
```

Algorithm

The reader/punch/interpreter handler has four entry points:

RPHHI - handler initialization, invoked by system initialization

RPHHP - receives control from system dispatcher when primed by system queue manager

RPHHC - receives control from system dispatcher when scheduled read, write, or control operations have completed

RPTAP - receives cotnrol frcm the time base generatcr handler when a timed pause expires (for read or write retries follcwing not ready states)

The following "structured" description provides internal module detail.

```
RPOHI:      entry point fcr initialization section.
            JSB to commcn code (RPHHI)
            return through entry point

RPHHI:      initialization section of common code;
            if restart, return;  if start, merge logical unit
            number with icc calls;  use .FIND to determine queue
            name;  exit;
```

```
RPOHP:      entry point for prime section;  JSB to common code;
            return through entry point;

RPHHC:      prime section of common code.
            establish addressing for local storage;
            check work queue;
            if no work, exit via .COM.
            else; if queue was purged,
                  use PAGES subroutine to clear device;
                  exit to commutator;
            save work entry address;
            use command type to enter appropriate
            routine (START, READ, WRITE, CNTRL, STOP,
            PURGE, RETRY, ALLBF)

START:      routine to handle start command.
            save stream ID;
            save activating handler queue name;
            if read only mode or write only, indicate RJE mode;
                    set default modes for stacker 1, hopper 1;
                    set Hollerith punch mode;
                    set EBCDIC read mode;
                    set to punch, print same data
                    issue control IOC call to set stacker
                    overflow mode;
                    continue START processing at STBUF
            if read/write mode, indicate TSB mode;
                    set default modes for stacker 1, hopper 1,
                    set ASCII read mode;
                    set Hollerith punch mode;
                    set to punch, print same data;
                    issue control IOC call to set stacker
                    select mode;
STBUF:      acquire buffers;
            give to activating handler as NOP;
            free start buffer;
            exit to commutator;

READ:       read command processing.
            create IOC parm from flags and read mode;
            zero out read buffer;
            issue IOC call.
            indicate read buffer active;
            exit to commutator;

WRITE:      write command processing
            if RJE mode, get punch mode from buffer;
```

```
                else get punch mode from write IOC parm;
                create IOC parm with flags and punch mode;
                issue IOC call;
                exit to commutator;

CNTRL:          control command processing.
                if immediate type control, set mode;
                      free control buffer;
                      exit to commutator;
                else; set up control call to driver;
                      issue IOC call;
                      free control buffer;
                      exit to commutator;

STOP:           stop command processing.
                purge timer if running;
                if read buffers active, issue CLEAR IOC call;
                else; issue clear wait station IOC call;
                indicate stop in progress;
                save address of stop buffer;
                exit to commutator;

RETRY:          retry command processor.
                start a timer;
                free retry command buffer;
                exit to commutator;

PURGE:          purge command processing.
                purge timer if running;
                purge device request queue;
                free purge command buffer;
                indicate purge processing;
                exit to commutator;

ALLBF:          allocated buffer processing.
                if no longer running, free the buffer;
                else; if control buffer; use for error report to
                      activating handler;
                   else; give to activating handler as NCP;
                exit to commutator;
```

The following modules make up the I/O complete section of the handler.

```
RPOHC:          entry point for I/O complete section.
                JSB to common code;
                end;
```

```
RPHHC:      I/O complete section of common code.
            establish addressing for local storage;
            if call was clear wait station, then;
                  return stop buffer to activating handler;
                  exit to commutator;
            if call was read, enter CREAD;
            else call was write;

CWRIT:      handle write complete.
            if no error, return to activating handler as NOP;
            else; save buffer address;
            if first time, then;
                  turn on error flag;
                  send error command to activating handler;
                  issue priority IOC call with same buffer;
                  exit;
            else; start timer;
            issue priority IOC call with same buffer;
            exit;

CREAD:      handle read complete.
            decrement active buffer count;
            if stop in progress, give buffer to activating
                  handler as NOP;
            else; check status;
            if no error; give buffer to activating handler;
                  turn off error flag;
                  exit to commutator;
            else; handle error;
            if first time or EOF, send error command to
                  activating handler;
                  exit to commutator;
            else; do start timer;
                  put buffer back on handler's queue;
                  exit to commutator;

RPTAP:      timer appendage.
            establish addressing for local storage;
            issue release IOC call to release request queue;
            return;
```

I. Product Identification

Product Abstract

This driver is designed for use with the HP2000 ACCESS system to support the HP12989A reader/punch/interpreter subsystem. The module is written in a serially reusable manner in order to support multiple (up to 7) reader/punch/interpreters.

II. Design Overview

Design Assumptions

This driver assumes a 2100 or 21MX series computer and the HP2000 ACCESS Firmware Option (13206A or 13207A).

Design Summary

The reader/punch/interpreter driver consists of two separate modules, D.340 and D.34C. The D.340 module contains the initiator, continuator, and power recovery entry points and all potentially volitile local storage for a given reader/punch/interpreter. Only this module must be replicated to support additional devices. The other module, D.34C, contains the actual driver code and is used as a subroutine by all copies of D.340.

The D.34 driver is a read/write driver and will support the following functions:

Clear- any card in the visible wait station is ejected to the selected stacker and the device status is cleared.

Read- Cards may be read in ASCII, EBCDIC or column binary format.

Write- cards may be punched in Hollerith or column binary format. ASCII data may be printed on the top edge of the card. The print data may be different from the punch data.

Control- control operations supported are
-feed a card from selected hopper
-set stacker select mode
-set stacker overflow mode
-clear the visible wait station by ejecting card to selected stacker.

The power recovery routine allows for the proper termination of any read/write operations that were active at the time of power failure.

The error handling section provides for the detection of empty hopper, full stacker, data conversion errors, pick fail, eject fail, or not ready conditions. The occurrance of a '::' sequence in columns 1 and 2 of a card is treated as end of file.

Design Approach

The driver is written in a serially reusable manner. That is, one copy of D.34C serves multiple copies of D.340. Access to local data storage for each device is through an indirect pointer list. Interrupts from multiple devices are queued and handled serially by the driver.

Module Organization

D.34C contains code for the initiator section (D.34C), the continuator section (I.34C) and the power recovery section (P.34C). In the initiator section, the call parameters are analyzed, initial conditions are established, and a read, write, or control operation may be initiated. Because the initiator section may be entered from base level programming (.IOC.) or the continuator section (.BUFR), the initiator section is entirely disabled.

I.34C, the continuator section, consists of three parts. The first part, which runs disabled, queues the interrupt for service. If this interrupt is the first to be queued, the second part is entered. Otherwise, an exit is made immediately.

In the second segment of I.34C, the interrupt is handled. A detailed description of the interrupt handling process is given below. When interrupt service is completed, the third segment is entered.

In the third segment, which also runs disabled, the storage block just serviced is removed from the queue. If more storage blocks remain in the queue, the second segment is entered again. Otherwise, an exit is made from the original interrupt.

The P.34C section is entered for each reader/punch/interpreter device at power recovery time. If the assocaited I.340 section was active, D.04? is entered to restore the condition. If the interrupt section was not active, but an

operation had been scheduled, an interrupt is forced to cause I.34C to effect recovery.

Module Detail

The following define the indirect pointers in D.34C used to access the local data in D.340.

EQTA	EQT address
BUFF	buffer address
BBUFF	byte address of buffer
NLEN	negative read/write length
LNGTH	original request length
PPOFF	print/punch buffer offset
IOREQ	I/O request word
FLAGS	driver flags
	bit 15 driver busy
	14 stacker mode
	=0 stacker select mode
	=1 stacker overflow mode
	13 write operation started
	12 feed after write
	11 first input character read
	10-8 not used
	7 inhibit input
	6-2 not used
	0-1 read/punch mode
	=0 ASCII/Hollerith
	=1 Column Binary
	=2 EBCDIC
CHAR	first input character
STCL	set control instruction for low select code
STCH	set control instruction for high select code
CLCL	clear control instruction for low select code
CLCH	clear control instruction for high select code
LIAL	LIA instruction for low select code
LIAH	LIA instruction for high select code
OTAL	OTA instruction for low select code
OTBH	OTB instruction for high select code
PSTAT	pseudo status word
	bit 15 not ready condition
	14 stacker full
	8 hopper 1 empty
	7 hopper 2 empty
	0 card in wait station
POPT	print/punch options for write request
SSTC	STC subroutine with power/fail instruction

The following define data specific to D.34C

```
SQADR       Queue for STC instructions
IQADR       Queue for interrupts
TEMP1       Temporary
TEMP2          storage
TEMP3        for continuator Section
```

The following "structured" description provides internal module detail.

```
D.340:      entry point for initiator section.
            JSB to common initiator code D.34C;
```

The following modules make up the initiator section of the common code.

```
D.34C:      entry into initiator section of common code.
            disable interupts; save EQT address;
            save EQT pointers;
            establish addressing for local data;
            establish real return address and
            save at D.34;
            get user request code;
            if clear, enter CLEAR routine;
            if driver already busy, then;
            set bit 15 in B-reg;
            enter REJCT;
            set to ignore int on low channel;
            save request code status;
            if read request, enter READ routine;
            if write request, enter WRITE routine;
            if control request, enter CNTRL routine;
            indicate request code reject (B=0);

REJCT:      indicate reject return (A=1);
            enable interrupt;
            exit;

READ:       routine to start the read operation.
            JSB to RWSTF to set up initial conditions;
            zero the transmission log;
            if device input buffer is full,
                  set to handle interrupts on data channel;
                  use LIA subroutine to read first character;
                  save at PRCHC;
                  indicate character read in driver flags;
                  issue STC,C on low channel;
```

```
                take normal exit (NEXIT);
          use FEED subroutine to feed card;
          if +1 return from FEED, take not ready exit (NREDY)
          if +2 return, take normal exit (NEXIT)

WRITE:    routine to initiate punch or print operation.
          use RWSTR subroutine to set up initial conditions;
          copy write length to transmission log in
          case immediate return;
          if punch request, set to enable punch;
          if print request, set to enable print;
          if separate print/punch data, set SPD bit,
          save print/punch options;
          if card in wait station, then;
                set B-reg with print/punch options;
                indicate write started in driver flags;
                skip next instruction;
          clear B;
          use FEED subroutine to start operation;
          if +1 return, take not ready exit (IMRET).
          if +2 return, take normal exit (NEXIT).

RWSTR:    subroutine to set up initial conditions
          for read or write operations.
          get buffer address and save (RPBUF);
          get buffer length and save (RPLEN);
          negate and save for count (RPLNG);
          get IO request word and save (RPREQ);
          make driver busy (EQT and driver flags);
          return;

CNTRL:    routine to handle control requests.
          get subfunction code;
          if feed request then;
                JSB to FEED subroutine;
                take not ready exit (IMRET);
                skip return--take normal exit (NEXIT);
          if clear work station request, then;
                enter EJECT subroutine;
                take not ready exit(IMRET);
                take normal exit (NEXIT);
          if set stacker select mode, then;
                set indicator in driver flags;
          if set stacker overflow mode, then;
                set indicator in driver flags;
IMRET:    set bit 15 in A-reg, clear B=reg;
          enable interrupts;
          exit;
```

```
NEXIT:      clear A-reg;
            enable interrupts;
            exit;

CLEAR:      routine to handle clear requests.
            zero transmission log;
            use STATS subroutine to get device status;
            if +1 return because not ready, then;
                  issue CLC to both channels;
                  clear status in EQT;
                  clear device flags;
                  exit via D.34;
            use EJECT subroutine to eject card;
            exit via D.34;
            exit via D.34;

EJECT:      subroutine to [illegible] the visible wait station;
            set to inhibit input on feed operations;
            clear B-reg;
            enter FEED subroutine;
            return;
            increment return address; return;
```

The following are subroutines used by both the initiator and continuator sections of the common code.

```
FEED:       subroutine to feed a card.  When
            entered, the B-reg contains print/punch
            options.  A plus one return is made if
            the device is not ready.  Otherwise, a
            plus two return is made.
            enter subroutine MTINB to clear device
            input buffer;
            clear buffer full flag in data channel;
            enter STATS subroutine to check status;
            return if not ready;
            if stacker full, then return;
            if selected hopper is empty, then return;
            set command channel control word by
                  merging B-reg with hopper/stacker
                  selection bits in IO request;
            output word to command channel;
            enter FILL subroutine to fill a waiting
            output buffer if necessary;
            if in initiator section, then issue STC;
            else queue the STC request;
            increment return address;
            return;
```

```
MTINB:      subroutine to empty the device input buffer.
EAT:        use LIA subroutine to read character;
            if buffer empty, return;
            JMP to EAT to empty the buffer;

LIA:        subroutine tc read data channel.
LIAL:       read the low select channel;
            if buffer empty, return;
            if data not ready, JMP to LIAL;
            else issue STC to get more data;
            issue CLC tc low select channel;
            return;

FILL:       subroutine to fill a waiting output buffer.
            indicate first time through;
FLUP:       enter CKRDY subroutine to see if device is ready
            fcr command;
            return to caller on plus one return;
            read data channel;
            if device flag is set,
            or first tise through;
            issue STC tc fill buffer;
            JMP to FLUP tc repeat;

CKRDY:      subroutine to check the ready for command bit.
            read the command channel;
            if not ready, then increment return address;
            else return;

STATS:      subroutine to check and update the device status;
            read low select channel;
            if status nct available then check high (CKHIS);
            if input check, then;
                  indicate not ready condition;
            if read check, then;
                  indicate data error condition;
            if hopper 1 empty, then;
                  indicate hopper 1 empty;
            if output check, then;
                  indicate not ready condition;
            if stacker full, then;
                  indicate not ready condition;
            save low channel status;
SKHIS:      read high select channel;
            if hopper 2 empty, then;
                  indicate hopper 2 empty
            if card in wait station, then;
                  indicate card in wait station;
```

```
                if not ready, then;
                      indicate not ready;
                merge low and high status;
                update EQT status word;
                if no not ready condition, then increment return
                address;
                return;
```

The following routines make up the continuator section of D.34.

```
I.340:      entry point for continuator section.
            JSB to common continuator code (I.34C);

I.34C:      entry point into continuator section of common code.
            disable interrupts; save registers;
            indicate interrupt received in local data;
            queue the interrupt;
            if not first on queue, then;
                  get return address;
                  restore registers;
                  exit with enable;
            else set up return address for exit;
            set switch in FEEL and LIA subroutines to
            indicate continuator in control;

I.GO        establish addressing for local data;
            establish EQT pointers;
            enable interrupts;
            set for no interrupts on data channel;
            get IC request word;
            if completion from CLEAR,
                  go to I.END;
            if completion from CNTRL,
                  go to I.END;
            if completion from WRITE,
                  go to IWRIT;
            else completion from READ;

IREAD:      handle read interrupt.
            if read column binary requested, then;
            IRCBI:      if first character already in,
                        use subroutine WRISZ to store character
                        and increment counters;
            IRCBL:      use GETLIA subroutine to read character;
                        use WRISZ subroutine to store character
                        and increment counters;
                        JMP IRCBL to loop until done;
```

```
            if ASCII read requested, then
                  set ASCII conversion table address;
                  enter IRASC routine;
            if EBCDIC read requested, then
                  set EBCDIC conversion table address;
                  enter IRASC routine;

IRASC:      subroutine to read ASCII or EBCDIC.
            change buffer add to byte address;
            if first character already read, skip
            over reading character;

IRASL;      use subroutine CRLIA to read character;
            use ACRT or ECVRT subroutine to convert character;
            use BRISZ subroutine to store character
            and increment counters;
            JMP IRASL to loop until done;

IREND:      routine to complete read.
            use MTINB to empty buffer if necessary;
            put # of characters less trailing blanks in transmission log;
            enter I.BUFR;
            go to I.END;

IWRIT:      routine to handle write interrupt.
            if feed before write, then;
                  set print/punch options (PPOPT);
                  enter FEED subroutine to start write;
                  go to I.END;
            if write completion, then
                  set transmission log;
                  set EQT status;
                  go to .BUFR to complete write;
                  go to I.END;
            else set temporary counter to -80 for punch operation;
            if punch binary request, then
                  set print buffer offset to -80;
                  get buffer address;
                  load data word;
                  output to device using OTA subroutine;
                  increment buffer printer;
                  use CWISZ subroutine to count down;
                  loop until done;
            else for ASCII punch;
                  set print buffer offset to -41;
                  convert buffer address to byte address;
                  get character from buffer;
                  use CATCH subroutine to convert character;
```

```
                output character;
                use CWISZ to count down;
                loop until done;

I.CND:      make driver not busy; clear driver flags;

I.END:      disable system;
            remove storage block from queue;
            if queue not empty, go to I.GO;
            else; dequeue pending STC;
            if any on STC queue, then;
                  issue the STC;
                  check for more entries on STC queue;
            else;

I.DUN       restore requests;
            exit with enable;
            end
```

The following subroutines are used by the read routines.

```
CRLIA:      subroutine to get character from device input buffer.
            use LIA subroutine to read character;
            if data present, then;
            return;
            else; enter IREND to complete read;

BRISZ:      routine tc store character and increment counters.
            store the byte in user buffer;
            increment character count;
            return;

WRISZ:      routine to store word and increment counters.
            store word in user buffer;
            increment buffer address;
            increment character count;
            return;

ACVRT:      subroutine converts to ASCII .
            if blank, load ASCII blank; return
            else convert card column punch to ASCII;
            if illegal punch, then enter IRDCK;
            indicate significant character;
            return;

ECVRT:      subroutine converts column binary to EBCDIC.
            if blank, load EBCDIC blank; return;
```

```
          else convert to EBCDIC;
          if illegal, then enter IRDCK;
          count significant character;
          return;

IRDCK:    use MTINB to clear device input buffer;
          set status in EQT;
          GO TO I.BFR;
```

The following subroutines are used by the write routines.

```
CWISZ:    subroutine to count down on punching.
          increment count of characters;
          count down using write length;
          when done, return plus one; else return plus 2;

OTA:      subroutine to output character.
          output character to buffer;
          read data channel;
          if not ready, loop until ready;
          else execute STC instruction;
          return;

CATOH:    subroutine to convert ASCII to Hollerith.
          check for non-valid character;
          if not valid, set to output blank;
          else get Hollerith from table;
          return;

WFILL:    routine entered at end of punch.
          fill up output buffer if necessary;
          if separate print data requested, then;
                get buffer address;
                add print data offset;
                convert to byte address;
                set temporary counter to -80;
                output byte;
                loop on output until done;
          disable interrupt;
          indicate feed after write;
          go to I.END;
```

The following is a description of the power recovery routine.

```
P.340     entry point for power recovery routine
          JSB to P.34C;
```

```
P.34C     common code for power recovery routine
          get and save return address;
          get address of interrupt entry point and save;
          get address os STC instruction and save;
          get address of STC flag and save;
          if interrupt section was active, then
                enter D.04? tc restore condition;
                exit;
          if STC instruction was executed, then
                issue STC instruction to cause interupt
          exit;
```

III. Design Structures

Interface Description

Clear

Any current operation is aborted. Any card in the visible wait station is ejected to the selected stacker. Device status bits are cleared and the driver enters an idle state.

```
JSB   .IOC.
OCT   0000xx      xx=logical unit number
```

Read

A card is read and transferred to the supplied buffer. Optional data conversion may occur. If the device read buffer is full (card was fed by previcus operation), no card movement takes place. Otherwise a card is fed from the selected hopper. If the device read buffer is empty, but a card is in the visible wait station, it will be ejected to the selected stacker when the next card is fed.

```
JSB   .IOC
OCT   parameter word
JMP   REJCT
DEF   BUFFR
```

The parameter word is defined as follows:

```
Bit 15      queued request
   14-12    function
            =1 read
    11      stacker select
            =0 stacker 1
```

```
          =1 stacker 2
10        hopper select
          =0 hopper 1
          =1 hopper 2
9-8       not used
7-6       read mode
          =0 ASCII
          =1 column binary
          =2 EBCDIC
5-0       unit reference number
```

Write

Data is punched and/or printed on a card. The punch data may be Hollerith or column binary format. The print data must be ASCII format. The print/punch data may be the same or different data.

```
JSB   .IOC
OCT   parameter word
JMP   REJCT
DEF   BUFFR
```

The parameter word is defined as follows:

```
bits 15      queued request
   14-12     Function
             =2 write
     11      stacker select
             =0 stacker 1
             =1 stacker 2
     10      hopper select
             =0 hopper 1
             =1 hopper 2
     9       not used
     8-7     punch/print options
             =00 punch,print same data
             =01 punch only
             =10 print only
             =11 punch,print separate data
     6       punch mode
             =0 Hollerith
             =1 column binary
     5-0     logical unit number
```

Control

The control request is used to perform special

functions and set device modes.

```
JSB   .IOC.
OCT   parameter word
JMP   REJCT
```

The parameter word is defined as follcws:

```
bit 15      queued request
    14-12   function
            =3 control
     11     stacker select
            =0 stacker 1
            =1 stacker 2
     10     hcpper select
            =0 hopper 1
            =1 hopper 2
      9     not used
     8-6    subfunction
            =0 feed card from selected hopper
            =1 enable stacker control mode
            =2 enable stacker overflow mode
            =3 clear visible wait station
     5-0    logical unit number
```

Status Inforamtion

On a read operation, the transmission log contains the number of card columns read less trailing blanks. On a write cperation, the transmission lcg contains the number of characters printed and/or punched.

The status word is defined as follows:

```
bits 14     error
     2-0    =C no error
            =1 not ready condition
               hopper empty
               stacker full
               pick fail
               eject fail
            =2 data conversion error on read
            =3 power fail
            =7 end of file
```

Serial Link Terminal Handler

Description and Function

The Serial Link Terminal Handler is responsible for normal flow of input, output and control requests to the Serial Link Terminals and notification of its caller of error and completion conditions.

This handler is written in a serially-reusable manner. In order to simplify the addition of new terminals to the system, the handler is composed of two separate and distinct modules. The first module contains the unique entry points and local data storage for a given terminal, so that when another terminal is added only this code is replicated. The second module is the serially-reusable code used by all copies of the first module.

This handler adheres to the defined level-2 level-3 protocol.

Interfacing

LTH accepts these work entry commands :

START - received from a level-2 handler in a control buffer It activates the handler.

WRITE - received from a level-2 handler in a data buffer. Contains a line of characters to be output to a terminal.

READ - received from a level-2 handler in a data buffer. Causes LTH to obtain data or information from a terminal.

STOP - received from a level-2 handler in a control buffer. Causes the handler to begin termination processing.

ALLOCATED BUFFERS - received from Buffer Manager. These are buffers which were previously requested but not available at the time of the request.

START TIMED RETRIES - received from a level-2 handler in response to error notifications. Causes the handler to restart the erroneous operation after a pause.

PURGE - received from a level-2 handler in a control buffer. Causes all outstanding requests to be purged and starts termination processing.

CONTROL OPERATIONS - received from level-2 handler in a control buffer. It is used to initiate special control operations on the terminal.

LTH returns these commands to its activator :

NOP - an allocated buffer, a completed write or a purged data buffer is being returned.

WRITE - a completed read is being returned.

STOP - the received stop command is returned following the return of all data buffers and de-activation.

ERROR - a notification that an error has occurred is sent.

Local Data Structures

The buffer in the input queue are defined as follows :

START command :

BCW-2 Bits	13-8	stream identifier
	3-0	command code (3)
BCW-3 Bits	14-0	activating handler queue name
BCW-7 Bits	15-0	buffer length
Data word-1		number of buffers (1)

WRITE/READ commands :

BCW-2 bits	3-0	command code (1 or 2)
BCW-7 bits	15-0	length of data in characters

CONTROL commands :

BCW-2 bits	3-0	command code (8)
Data word-1		control function code

The buffers in the output queue are defined as follows :

WRITE type :

BCW-2 Bits	13-8	stream identifier
	3-0	command code (1)
BCW-7 Bits	15-0	number of characters in buffer
BCW-8 Bits	15-2	unused
	1-0	no error (0)

ERROR type :

BCW-2 Bits	13-8	stream identifier
	3-0	command code (6)
BCW-8 Bits	1-0	error type

NOP type :

BCW-2 Bits	13-8	stream identifier
	3-0	command code (0)

STOP type :

BCW-2 Bits	13-8	stream identifier
	3-0	command code (4)

These symbolic pointers to local data are used :

```
TERMA - address of terminal on Serial Link
TALKA - address of current input unit
        bits 15-14 : not used
              13-5 : terminating mode of special keys
               4-0 : input unit address
STATU - status of terminal
        bits   15 : terminal busy
               14 : power fail occured on request
             13-3 : not used
              2-1 : input completion type
                0 : service requested at terminal
LIGHT - state of prompting lights on terminal's display
CONFG - configuration options
        bits 15-9 : not used
                8 : HPIB istrument option present
                7 : CRT unit state
                6 : CR[illegible]
                5 : not used
                4 : Card/Badge reader option present
                3 : Printer unit state
                2 : Printer option present
                1 : Display unit state
LUN   - .IOC. logical unit number
QUENM - name of associated work queue
AHQUE - name of level-2 activating handler
SID   - stream identifier
FLAGS - internal logic conditions
        bits  15 : (REQON) request in progress
              14 : (ERFLG) error detected during command
              13 : (CTBEX) control expected back
              12 : (INPEX) read data expected
              11 : (PRIEX) print of data expected
              10 : (RDIEX) read status information expected
               9 : (STBON) status request in progress
               8 : (SRION) service request identification on
               7 : (PFAIIL) power failed during command
               6 : (IBCEX) HPIB command expected
               3 : (LITEX) light numbers expected
               2 : (RESTO) environment being restored
               1 : (AUTRD) automatic read selection in effect
               0 : (PUGRQ) termination procedure in progress
BYTEI - information status byte
        bits   15 : type of information (address or status)
             14-6 : not used
              5-0 : information part (address of unit or status)
TCTBF - control buffer address for use in delegation of control
BUFFR - head of wait queue
STOPB - tail of wait queue
TQE   - timer queue elements
```

Algorithm

LTH has four entry points :

LTHHI : "handler initialization" : invoked by system initialization module. It issues a call to FIND routine for its own queue name. The queue name is retained in QUENM and will be needed for issueing GETQ requests.

LTHHP : "handler priming" : receives control from System Dispatcher when primed by system Queue Manager. PADDS is used to generate a list of indirect addresses for access to the local storage area. Initially the handler is in an idle state. It is removed from this state by being primed with a START command containing the queue name of the level-2 activating handler. All other commands are accepted following the START. PURGE, START-TIMED-RETRY and ALLOCATED BUFFER commands are processed as soon as they are received. READ, WRITE, CONTROL and STOP commands are queued on the handler local wait queue. If the wait queue is empty upon reception of a command that command is immediately processed. LTH stays in the active state until a STOP or a PURGE command is processed. Then it returns to its idle state and a new START command is needed to re-activate it.

LTH handles one queue entry at a time when it is primed. The queue entries are processed as follows :

START : the activator's queue name and the stream identifier are saved in AHQUE and SID respectively. The rest of the local storage area is initialized. The START buffer is returned to the Buffer Manager and one data buffer of appropriate size is requested from the Buffer Manager. Once obtained this buffer is used to issue an IFC call to .IOC. which causes the driver to initialize the terminal.

WRITE : the data in the buffer is interpreted according to the value of selected bits in FLAGS :

a) if bit-3 is set the buffer contains a list of light numbers. The appropriate sequence of requests to .IOC. is issued to correctly position the prompting lights. LIGHT is updated to reflect the state of each light on the display.
b) if bit-4 is set the buffer contains a list of special function keys. The appropriate sequence of requests to .IOC. is issued to correctly set the terminating mode of the keys. SFKS is

updated to reflect the mode of each special function key.

c) if bit-6 is set the buffer contains an HPIB message. That message will be decoded and emulated according to its meaning.

d) otherwise the buffer contains ASCII characters. An .IOC. call is issued to cause the driver to print a line of characters on the terminal.

READ : if an input unit is defined, that is if bits 4-0 TALKA word are not all set to one a read request is issued to enable the current input unit on the terminal and start an input operation. If no input unit is defined the read buffer is used to issue a WAIT on SRQ request to IOC. Then when the SRQ line goes LOW the address of the input unit will be determined and a normal read issued to this unit.

STOP : the data buffer containing the command is used to issue an IFC call to .IOC. which causes the driver to reset the terminal to its initial state. PUGRQ condition is set in FLAGS and any pending operation is cancelled.

ALLOCATED BUFFER : the buffer may be a control buffer or a data buffer. The control buffer is used to send an error notification to the activator. The data buffer is used to issue an IFC call to .IOC. to cause the driver to initialize the terminal. Subsequently the buffer is made interlocked. The stream identifier and NOP command are placed in BCW2 and it is sent to the activator.

START TIMED RETRY : a timer is started and the buffer is freed. When the timer finally exits, the operation at the top of the wait queue is re-initiated.

PURGE : an IFC call is issued to .IOC. which causes the driver to reset the terminal to its initial state. PUGRQ condition is set in FLAGS.

CONTROL : it is used to perform special operations on the terminal and its optional units. Those special operations are emulated as a series of write requests to the associated driver. Control functions 71 to 81 are related to the display, keyboard, printer, badge and CRT units of a terminal. At the end of the actual operation the control buffer is released by this handler. Control function 70 is related to instruments connected to a terminal via its internal HPIB.

Once a control request with code 70 has been received the normal control functions (codes 71 to 81) become ineffective. The control buffer is kept in TCTBF word for use in case of the HPIB "delegate control" command.

The CALL routine is invoked whenever a call to .IOC. is necessary. It prepares the required code sequence and saves in word BCW5 the address of the local storage area associated with this terminal.
The IOCMD and IOCM2 routines format special command write requests to the driver (pure HPIB commands are issued that way).

LTHHC : "handler I/O completion" : receives control from System Dispatcher when scheduled read or write requests have completed.
The address of local data storage is found in word BCW7 of the returned buffer. PADDS routine is used then to generate the correct pointers to the local area associated with the terminal. If PUGRQ condition holds, that is if a PURGE command has been previously received, the buffer is returned to level-2 handler as NOP (if a data buffer) or freed (if a control buffer).

The following error detection/correction procedure is used when the error bit is set by the driver :

- if ERFLG is already on, that is if an error was detected on that same operation during the previous attempt, a timer is started for 1 second and the request is requeued at the head of the wait queue.
- otherwise ERFLG condition is set and a control buffer is requested to notify the level-2 handler of a type-2 or type-3 error condition. The buffer is requeued at the top of the wait queue and will be reactivated when a timed retry is scheduled.

If no error has occured the command type is analysed to determine the next action to undertake. One of the following may take place :

- the operation has not completed and the buffer is used to send the next request to the driver.
- the operation has completed and the data buffer is returned as NOP (if a WRITE command), as WRITE (if a READ command)) or as STOP (if a STOP command), while the control buffer is freed by this handler. If PUGRQ condition holds the wait queue is emptied and all the buffers returned to the system.
  If the wait queue is empty control is passed to the System Dispatcher. Otherwise, the top entry of the wait queue is obtained and passed to LTHHP section for processing.

LTHT : "timer-expiration" section : receives control from the time base generator handler when a timed pause expires (for operation retries following not ready states).
Upon entry the content of the A register (address of timer block) is used to generate the pointers to the local area associated with this terminal.
The buffer at the top of the wait queue is obtained and passed to LTHHP for re-processing.

Special cases
------- -----

1) Restarting a Read operation through the SRQ alarm button.

When the presence of an SRQ is sensed at the end of a Read request the following takes place :

- the SRQ is acknowledged
- the terminal is restored to the state recorded in handler's local tables (lights, keys, input/output units)
- the read operation is automatically restarted.

If the request could not be acknowledged at the beginning of the procedure then a type-3 error is notified (operator intervention).

2) Power fail condition.

The handler receives in STATU an indication of the occurence of power-failures (it records this condition in PFAIL flag).

If at the end of an operation PFAIL condition holds, the following takes place :

- the system is notified of an error if the operation was a read since one character may have been lost.
- the operation is retried if it was a WRITE of prompting light codes.
- the operation terminates normally in all other cases.

3) Instruments in use.

When a control function 70 has been executed the terminal is viewed as a set of HPIB compatible instruments (keyboard is an input unit with address 29, display is an output unit with address 29, ...).
No recovery procedure is attempted in this case as the handler does not keep track of the state of the complete HPIB. The SRQ condition is no longer interpreted as an input restart indication.

HP 2000 ACCESS I/O PROCESSOR
BASE DESIGN SPECIFICATIONS
D.53 SERIAL LINK CONTROLLER DRIVER

I. Product identification.

I.1 Product abstract.

The D.53 Serial Link Controller driver is designed for use with the HP2000 Access System. It provides an interface for input/output operations between an HP21MX series computer and 3070 data collection terminals connected to a 40280A controller via a Serial Link.
The D.53 driver will handle up to 8 Serial Link Controllers simultaneously whith up to 31 terminals on each Serial Link (the system will limit to 31 the total number of 3070 terminals configured in the I/O processor). Transmission rates between 25 and 250 characters are supported by each controller.

II. Design overview.

II.1 Design assumptions.

The driver module assumes a 21MX series computer and requires the HP2000 Access I/O processor microcode.

II.2 Design summary.

The complete driver consists of two distinct modules (D.530 and D.53C). The D.530 defines the actual driver initiator, continuator and power fail recovery entry points associated with any given Serial Link Controller. This is the only module which must be replicated to support multiple controllers. It includes all data associated with a specific controller such as related I/O instructions, list pointers.....
The D.53C which contains the actual code is used by all copies of D.530 to drive the 3070 terminals controlled by a Serial Link Controller.

II.3 Design approach.

II.3.1 Controller logic.

Although the Serial Link Controller is physically a single I/O board, it may be viewed as a multiplexer for up to 31 data collection terminals of the 3070 family.
The controller uses a polling technique to communicate with the terminals. A polling cycle consists of a series of elementary messages conforming to the Serial Link Protocol and constructed from a table of informations passed by the driver before starting a cycle. The format of an information word is described in Table-1.
Upon the end of the cycle the controller will interrupt the CPU if one of the two following conditions holds :

1) an interrupt has been requested by the driver
2) at least one operation programmed in the cycle has completed (either a character has been accepted or one character has been received).

If none of the conditions is fulfilled the controller automatically repeats the cycle.

II.3.2 Driver logic.

D.53c is written in a serially reusable manner and serves all Serial Link Controllers.
It accesses a given D.530 storage block through a series of indirect pointers. In addition it has access to the calling handler storage block where it finds information about a specific terminal. D.53C finds in BCW-5 the address of the data block associated with the terminal to which the request is directed. The structure of the part of the data block that D.53C accesses has been detailed in Table-2.
For each controller the driver maintains a linked list of data blocks called the Active Terminals List. The head of the list is part of that controller local data structure (.ACTQ) while the elements are the calling handlers data blocks.
Each time a request is accepted by the driver, the calling handler data block is inserted into the Active List of that controller according to the terminal number (contained in TERMA word).
Normally the driver will reject a request if the previous one has not completed. The only exception to that rule is the "initialize terminal" request which will cause the current request to be aborted.
Upon request completion by the driver, the data block is unlinked from the Active List.

II.3.3 Driver algorithm description.

Like all BCS drivers the Serial Link Controller driver has an initiator section (I.53C), a continuator section (D.53C), a power-fail/restart control section (P.53C) and a time out exit (T.53C).

The initiator section is responsible for converting a requested operation into appropriate actions.
The following prologue takes place at request initiation in D.53C :

1) validity of request is checked
2) handler data block is located, initialized and linked to the Active List
3) if the Controller is already active an interrupt is requested at the end of the current polling cycle and step 4 is skipped.
4) if the Controller is not active the table of information is constructed, passed to the controller which starts a cycle.

The continuator section is primarily an interrupt handler and event complete processor. The following procedure is activated upon acknowledgment of an interrupt from a controller :

1) the table of informations is obtained from the interrupting controller
2) for each terminal present in the Active List the op-code and the step-number are used to determine the next action i.e :
   - either a transition to the next step takes place and the next element of the information table is prepared
   - or the request is completed if it has reached its final step.
3) if not all terminals have completed their requests the table of informations prepared in step 2 is used to start the next cycle.

If another interrupt occurs while this section is in use, that interrupt is queued and processed only when the active interrupt is completed.

The power fail recovery section receives control from D.04 after power is restored to the CPU following a power failure.
P.53C is responsible for restoring the controller interface board to the status it was in when power failed.

III. Design Structures.

III.1 Handler ---> Driver Interface structure.

The handler and the driver levels of software interface thru calls to .IOC.. The following functions are issued by the handler :

Function	Sub-function	Operation
01	00	Normal read of data
	01	Transparent read of data
02	00	Normal write of data
	01	Terminal initialization
	02	Special write of HPIB commands
	03	Remote Control Enabling
	04	Remote Control Disabling
	05	Service Request Condition Monitoring
	06	Service Request Identification

III.2 Driver ---> Handler interface structure.

The driver returns requests to the handler level thru calls to .BUFR.. The transmission log information is passed in that handler data block (BUFL word) while the address of that data block is placed in BCW-7 of the returning buffer upon request completion.

When a call is rejected the following return codes will appear in the A and B registers :

1) A =1 , B =0 : if the request parameters are illegal (code, arguments...)
2) A =1 , B =100000 : if the terminal is busy executing the previous request.

III.3 Major functional modules.

The D.53C driver is modular in its internal architecture. Each module may be viewed as a "function processor" (modules which are responsible for executing a particular request) or as a "common module" (modules or subroutines which aid function processors).

Normal Read

Description and function

Inputs a line of data from the current input unit of a terminal. A line is considered terminated when a terminating character is received (LF, EOI line LOW, SRQ line LOW or any special function key defined as a terminator in TALKA).

Interface Description

```
        JSR   .IOC .
        OCT   0100XX      XX is the LU of a Controller
        JMP   REJCT       Exit to reject address
        DEF   BUFFR       Start address of buffer
        DEF   LEN         Length of buffer in + characters
```

The input unit is specified in TALKA bits 4-0.

Algorithm description

1. Put terminal in input mode
2. Enable input unit
3. Input characters until terminator detected
4. Disable input unit
5. Update STATU, BUFL and return buffer.

Transparent Read

Description and function

Inputs characters from the HPIB, where it assumes that an input unit has been previously defined by a local controller. Request terminates upon receipt of any normal terminating character, any command message (whith ATN line LOW) or when buffer is full.

Interface Description

```
JSB   .IOC.
OCT    0101XX        XX is the LU for the Controller
JMP    REJCT         Exit to reject address
DEF    BUFFR         Start address of buffer
DEC    LEN           Length of buffer in + characters
```

Algorithm Description

1. Input characters until termination condition is detected
2. Update STATU, BUFL and return buffer.

Normal Write

Description and Function

Sends characters to the output units of a terminal. The line of data is terminated by a LF character unless the buffer ends with a "<-".

Interface Description

```
        JSB   .IOC.
        OCT   0200XX        XX is the LU of the Controller
        JMP   REJCT         Exit to reject address
        DEF   BUFFL         Start address of buffer
        DEC   LEN           Length of data in + characters
```

Algorithm Description

1. Output characters in successive polling cycles
2. End with LF (unless buffer terminated with "<-")
3. Return buffer.

Initialize terminal

Description and Function

Resets terminal to a well-defined initial state (display cleared, lights off).

Interface Description

```
        JSB   .IOC.
        OCT   0201XX        XX is the LU of the Controller
        JMP   REJCT         Exit to reject point
        DEF   BUFFR         Start address of buffer
        DEC   LEN           Not used
```

Algorithm Description

1. Send command message with IFC LOW
2. Send command message with IFC HIGH
3. Send Device Clear command
4. Return buffer.

The following error detection procedure is used :

- if terminal does not respond in steps 1, 2 or 3 the error bit is set in BCW-8 word
- if terminal is busy, the current request is aborted and terminal initialization is started.

Command Write

Description and Function

Sends HPIB command messages constructed from data passed in buffer.

Interface Description

```
        JSB   .IOC.
        OCT   0202XX        XX is the LU of the Controller
        JMP   REJCT         Exit to reject address
        DEF   BUFFR         Start address of buffer
        DEC   LEN           Number of commands to send
```

Commands are packed 2 per word in the buffer.

Algorithm Description

1. Format command messages by adding the ATN bit to the data bytes and send them one per polling cycle
2. Return buffer.

Remote Control Enable

Description and Function

Sets terminal and associated instruments in remote control mode.

Interface Description

```
        JSB   .IOC.
        OCT   0203XX          XX is the LU of the Controller
        JMP   REJCT           Exit to reject point
        DEF   BUFFR           Start address of buffer
        DEC   LEN             Not used
```

Algorithm Description

1. Set REN bit in TERMA word,
2. Send message with REN line LOW
3. Return buffer.

Disable Remote Control

Description and Function

Resets the terminal and its associated instruments in their local control mode.

Interface Description

```
        JSB   .IOC.
        OCT    0204XX         XX is the LU of the Controller
        JMP   REJCT           Exit to reject point
        DEF   BUFFR           Start address of buffer
        DEC   LEN             Not used
```

Algorithm Description

1. Zero out REN bit in TERMA word
2. Send command message with REN line HIGH
3. Return buffer.

Wait Until Service Requested

Description and Function

Watches the SRQ line until a unit on the terminal requests service.

Interface Description

```
        JSB   .IOC.
        OCT   0205XX        XX is the LU of the Controller
        JMP   REJCT         Exit to reject point
        DEF   BUFFR         Start address of buffer
        DEC   LEN           Not used
```

Algorithm Description

1. Put terminal in input mode
2. When SRQ line goes LOW return buffer.

Identify Unit Requesting Service

Description and function

Determines which unit on the terminal requested service. Units to explore are taken from a list of units specified in the buffer.

Interface Description

```
        JSB   .IOC.
        OCT   0206XX      XX is the LU of the Controller
        JMP   REJCT       Exit to reject point
        DEF   BUFFR       Start address of buffer
        DEC   LEN         Minus number of units to explore
```

Algorithm Description

1. Disable all output units
2. Disable all input unitss
3. Enable Serial Poll Procedure
4. For each unit to be explored :
   4.1 Enable input unit
   4.2 Read status from unit and store into buffer
   4.3 If SRQ bit set in status byte proceed with step 5
5. Disable all input units
6. Disable Serial Poll Procedure
7. Check whether SRQ gone
8. Update STATU word and return buffer.

Get-New-Word Subroutine (GETNW)

Description and function

Builds a new word of the information table according to the contents of TERMA word bit 15-6 (operation-code and step number). If the information table shows an input operation that has successfully completed, the character of the table is stored in the buffer.
The step-number part of TERMA word is updated and the current request is completed if the very last step has been reached. The structure of the step number information is described in Table-3.

Send-New-Word Subroutine (SEND)

Description and Function

Once the information table has been built, this routine outputs it to the controller. The transfer is programmed using the skip on flag technique and only the relevant part of the information table is passed to the controller (the last word has its STOP bit set to 0).
The following error detection/correction procedure is used :

1) if power failure occured just after the table has been sent, the transfer is restarted
2) if the flag on the controller board is not comming back after the transfer, the power fail condition is checked as in 1.
3) after the whole table has been sent, a time-out of 2 seconds is programmed to be sure to come-back even if power fails during the polling cycle and prevents the controller from interrupting.

Table-1 : Information word format.

Driver ---> Controller.

1. Terminal in input mode :

Bits	Interpretation
15	Last word of table indicator (STOP) : 0 : last word 1 : not the last word
14	Not used
13-12	00 for input mode
11-0	Not used

2. Terminal in output mode :

Bits	Interpretation
15	Last word of table indicator
14	Not used
13-12	01 for output mode
11	IFC line state
10	REN line state
9	EOI line state
8	ATN line state
7-0	Data byte lines

3. Terminal in idle mode :

Bits	Interpretation
15	Last word of table indicator
14	Not used
13	1 for idle mode
12-0	Not used

Controller ---> Driver.

1. Terminal in input mode :

Bits	Interpretation
15	Last word of table indicator
14	Transmission error flag : 0 : transmission error detected 1 : no error during transmission
13-12	00 for input mode
11	Not used
10	Data byte present flag : 0 : no data byte transmitted 1 : valid data byte transmitted
9	EOI line state if data present SRQ line state if no data present
8	ATN line state if data present Not used if no data present
7-0	Data byte if data present Not used if no data present

2. Terminal in output mode :

Bits	Interpretation
15	Last word of table indicator
14	Data byte accepted flag : 0 : data byte not accepted 1 : data byte accepted
13-12	01 for output mode
11-0	Message being output by driver

3. Terminal in idle state :

Bits	Interpretation
15	Last word of table indicator
14	Not used
13-12	10 for idle mode
11-0	Not used.

Table-2 : Data block structure.

In D.53n (n=0 to 7 stands for controller number) module :

```
   .ACTQ     - head of Controller Active Terminals List
   PFLAG     - power-fail/recovery indicator
                 1 : recovery from power failure occurred during cycle
                 0 : no power failure during cycle
```

In LTxxH (xx stands for the terminal number) module :

```
   LINK      - link in Active List
   TERMA     - terminal address specifications
               bits   4-0 : address of terminal on Serial Link
                        5 : not used
                        6 : start of request flag
                        7 : REN line state (1 if LOW)
                    11-8 : step reached in request execution
                   15-12 : op-code of current operation
   TALKA     - terminal input unit specifications :
               bits   4-0 : input unit number (37 if none)
                     13-5 : special function keys mode indicators
                              0 : key is not a terminator
                              1 : key is defined as a terminator
                       14 : SRQ line terminating mode
                              0 : SRQ detection will not iend input
                              1 : SRQ detection terminates input
                       15 : not used
   STATU     - terminal status word
               bits    0 : service requested indicator (SRQ)
                              0 : SRQ line was HIGH
                              1 : SRQ line was LOW
                     2-1 : read request completion condition
                              0 : normal completion
                              1 : HPIB command message received (ATN L
                              2 : buffer has been filled
                     7-3 : unit identified as requesting service
                    4-13 : not used
                      14 : power fail indicator :
                              0 : power did not fail during request
                              1 : potential error due to power failure
                      15 : terminal busy indicator :
                              0 : terminal not busy
                              1 : terminal busy executing a request
```

Table-3 : Step number information structure.1

Bits	Interpretation
15	Source of next information word : 0 : computed by routine whose address is in bits 14-0 1 : word is encoded in remaining bits
14-13	Modifications to apply to information word: 0 : not any modification 1 : merge with TALKA bits 4-0 2 : merge with BUFA word
12-0	Bits 12-0 of the information word before applying the modifications.

SECTION V

RJE

HEWLETT-PACKARD 2000 SYSTEM I/O PROCESSOR

RJE

CONTENTS

1.	D.62	INTERCONNECT KIT DRIVER (RJE)
2.	CIO	CONSOLE INPUT/OUTPUT HANDLER
3.	SYN	SYNCHRONOUS COMMUNICATIONS HANDLER
4.	D.50	SYNCHRONOUS COMMUNICATIONS DRIVER
5.	HLO	HOST LIST DECOMPRESSION HANDLER
6.	HRO	HOST READER COMPRESSION HANDLER
7.	HMO	HOST MESSAGE DECOMPRESSION HANDLER
8.	HIO	HOST INQUIRE COMPRESSION HANDLER
9.	Appendix A	MULTILEAVING SPECIFICATIONS
10.	CDC RJE	

I. Introduction

These modifications serve as an extension of the D.61 Driver to allow RJE Console operations. Functions implemented are READ, WRITE, and INIT.

II. Design Overview

Design assumptions

This driver is designed to run in the HP2000 ACCESS environment.

Design summary

The modifications will allow the RJE Console feature to be implemented as a Buffered (Queued) I/O device. READ ASCII, WRITE ASCII, and INIT/CLEAR requests will be accepted. The processor-to-processor interface provides for operation synchronization with a "request-to-send," "permission-to-send" approach.

Major modules include READ request processing, WRITE request processing, INIT request processing, and SP command processors. All routines have access to the output routine OMTOM.

III. Design Structures

Major functional modules

The INIT processor must be invoked to initialize RJE Console processing. The calling sequence is as follows:

```
INIT REQUEST

JSB .IOC.
OCT 0000XX (XX is logical unit number)
```

This request saves the EQT address of the unit for later reference.

It may also be used to CLEAR operations active on the driver. An active operation will be reported by a .BUFR call, creating a CRQ entry with "a"=3.

The READ processor is entered to process READ requests from an .IOC. call. The calling sequence is:

```
READ REQUEST

JSB .IOC.
OCT  Ø1ØØXX (11ØØXX for queuing)
<reject address>
DEF BUFR
<buffer length>
```

The <reject address> is entered if the driver is busy. <buffer length> is the number of characters in the buffer. Only ASCII READ's with byte data (positive buffer length) are valid. READ processor execution is summarized in the accompanying flowcharts.

The WRITE processor is entered to process .IOC. WRITE requests. The calling sequence is:

```
WRITE REQUEST

JSB .IOC.
OCT  Ø2ØØXX (12ØØXX for queuing)
<reject address>
DEF BUFR
<buffer length>
```

Again, only ASCII records with byte data are valid. The execution sequence is given in the accompanying flowcharts.

The SP command processors process the 2 SP-to-IOP commands. These are:

```
160011      Wake RJE up
160026      Have console message
160037      Console buffer empty
```

Have console message results in an unsolicited event being placed on the CRQ. Its format is:

```
A:   ØØØØXX (where XX is console logical
             unit number)
B:   ØØØØØØ
```

A READ should be issued when such a request is received.

A write command to D.62 results in an IOP to SP command being sent (16LL05 where LL is length of message). Upon receipt of the "console buffer empty" command from the SP, the message is transmitted to the SP via DMA.

A read command is issued when the SP indicates it has data for the RJE subsystem through a "have console message" command. Upon getting the read command from the RJE subsystem, an IOP to SP command is issued to tell the SP to send the data. That command is "wake RJE up" (160011).

Console Input/Output Handler

Description and function

The console input/output handler (CIO) is responsible for initiating all input and output requests to the console. It accepts console messages from various handlers and issues queued write requests through .IOC.

The handler responds to an attention request and issues a priority read to the console via queued .IOC.

Another function of this handler is to identify the messages read from the console. This is necessary so that the message may be transferred to the appropriate handler.

The handler must also recognize the responses to requests for the 'System Connect' (SC command) and 'System Disconnect' (SD command). It will then be possible to filter out inappropriate RJE commands before they are transferred to other handlers.

Interfacing

The console input/output handler can receive console messages from several different handlers. The buffer is freed when the write request is complete with two exceptions. If the console message was received from the host message decompression handler, it is given to the ASCII files handler for the job message log. If it was received from ASCII files (job inquire function), it is given to HIO.

The unsolicited event indication received by this handler will be a request to read from the console, an indication of system shut down, or an indication of system initialization.

This handler will transfer messages to other handlers as follows:

SC,SD,ER	synchronous handler
SR#,TR#,RR#	host reader compression handler
RC-msg	host inquire compression handler

The DA command is handled within this routine as described below. Any other command is assumed to be an inquiry to the host and is sent to HIO.

If the command from the console is to be transferred as a block of data to the synchronous handler (RC-msg), it is first

converted from ASCII to EBCDIC via a call to the central conversion routine (.ATOE).

Local data structures

INBUF	address of buffer used for console input
QNAME	console input/output queue name
SHNAM	synchronous handler queue name
CICNM	host inquire compression queue name
SYSGT	'system activity' gate is set to 'nop' if the RJE system is connected and set to 'skip' it is not

The buffer control words associated with messages to be printed on the console are defined as follows:

	Bit(s)	Function
Word 2	3-0	command=1
Word 7	15-0	message size
Word 8	15-13	Synchronous handler run status
	12-0	(unused)
Data word 1	15	message is from HMO
	14	message is from ASCII files handler

The buffer control words associated with messages transferred from the console to other handlers are defined as follows:

	Bit(s)	Function
Word 2	15-4	(unused)
	3-0	command=7
Word 7	15-0	message size

The unsolicited event indication is defined as follows:

A-reg -	15	1-system overflow flag 0-normal unsolicited event
	14-12	zeroes
B-reg -	15	system shut down
	0	system initialization

Algorithm

The console input/output handler has 3 entry points:

CIOHI -	initialization section
CIOHP -	scheduled primer section

CIOHC - completed event section

In the initialization routine, (CIOHI), system service .FIND is used to determine the queue names of all handlers with which communication is required. On entry, it saves the logical unit number and merges this number into all of its .IOC. call parameters. A clear .IOC. call is made to purge any previous outstanding requests and to initialize the ICK driver.

In the scheduled section (CIOHP), all queue entries are inspected and appropriate .ICC. calls are made. If the entry is a buffer, a read request is queued and if it is a message to print, a write request is queued. SYSGT is opened or closed when the handler recognizes a 'COMMUNICATIONS ESTABLISHED' or 'SYSTEM DISCONNECTED' message, respectively. When communications are established, start commands are sent to ASCII files handler to start the job message and job inquiry functions. At disconnect time, stop messages are sent to terminate these functions. While RJE is active, console messages received from ASCII files will be sent to HIO and messages received from HMO will be sent to ASCII files handler.

There are three main parts of the IO complete section (CIOHI). When an unsolicited request for a read occurs, a buffer is obtained and a priority read request is made. When system shut down occurs, the queue is purged and all console messages are ignored until system initialization is received.

When an IO complete indication is received, it is first necessary to determine if the completed event was a read or write. If a write, the buffer is then freed or passed on to HIO or ASCII files as required.

If the completed event was a read, the type of console message must be determined so that it may be transferred to the appropriate handler. If the RJE system is not connected, RJE commands are inappropriate and an 'RJE COMMAND NOT APPROPRIATE' message is printed on the console.

In the case of a 'RJE-DA,hf<dev' or 'RJE-DA,hf>dev' command, the text is scanned for syntactic errors. If correct, CIO uses the device assignment manager to make the assignment. If the assignment is not successful, an appropriate error message is printed.

If the message is a 'RJE-DA' command, the DAT is scanned and messages are printed on the console reflecting current device assignments.

Synchronous Communications Handler

Description and function

The synchronous communications handler (SH) manages all I/O on the communications line. Once activated, it manipulates and controls the transmission and receipt of data within the defined IBM multileaving protocol. SH is activated at the request of the operator via the SC command. Activity then continues until an SD command or abortive communications line error occurs. SH will report data regarding line condition to the operator when directed to do so with ER.

IBM's multileaving protocol provides for the support of up to seven input streams (card image) and fourteen output streams (seven print and seven punch card image) as well as a remote operator console input/output stream. These 23 streams are supported and managed by SH using 23 work queues. The number actually used depends on the software and hardware configuration of which SH is a part. At system initialization time, these 23 work queues are enabled or disabled as required by the configuration. One other control work queue is also associated with SH for use in buffer acquisition, etc.

When a system connection is made, the enabled work queues are attached to other function handlers which either produce data (input streams) or consume data (output streams). SH then interfaces with the host system via the multileaving protocol in order to control the flow of data appearing on these queues. In appendix A of this base design document, is a specification of the multileaving protocol. Some terms which appear here (such as RCB and FCS) are fully defined in appendix A.

Interfacing

SH interfaces directly with the console I/O handler as follows:

1. SH accepts the SC, SD, and ER commands at any time. These must appear in console buffers and on the SH control work queue.

2. SH places messages on the console I/O work queue via .CCO.

SH interfaces with <u>any</u> one of its input modules as follows:

1. SH places a 'SH running' message on the handler's work queue. This message informs the handler of current SH parameters and indicates that the handler may proceed. It is also used to inform the handler that it may continue. (This latter point applies when the handler suspends itself after having reached the queue limit prescribed by SH.) Contained in the message are:

   RCB - stream identification which the handler is to use in constructing its data blocks.

   Buffer size - current communications buffer size

   Queue name - name of SH input work queue on which the handler is to place its data blocks

   Queue limit - maximum number of data blocks the handler should place on the queue. At the limit, the handler should suspend itself (or at least refrain from further data handling) until a new 'SH running' signal is received.

   Mode - a flag to indicate when the system is in ACCESS-to-ACCESS mode.

2. SH accepts a start command from the handler (except for the host inquire stream). SH then procedes to initiate the transmission of this stream to the host. The producing handler is expected to quiesce until this has occurred. SH will issue a new 'SH running' command to the handler once the transmission has been initiated.

3. SH accepts data blocks from the handler on the specified work queue. A BCW definition for the final block in a series of blocks must be adhered to.

4. SH will purge the handler's work queue if an abortive line error occurs or when a normal disconnect occurs. The handler should view this as an 'SH not running' signal.

SH interfaces with any one of its output modules as follows:

1. SH places a request to receive data signal on the handler's work queue. This signal will be contained in a communications buffer. When the handler has acquired all needed resources it is expected to return this buffer to SH in order to signal permission for the

request. ACCESS-to-ACCESS mode will be indicated in this request to receive message. (This entire process does not apply to console output.)

2. SH places data blocks on the handler's work queue. When the handler disposes of a block, it must be returned to SH to signal a reduction in that handler's queue level.

3. SH may purge the handler's work queue if an abortive error occurs and will definitely purge when SH performs a signoff. This should be viewed as a signal to quiesce. In this case, blocks of data which are partially disposed of, should not be returned to SH.

Local data structures

These local data are used by SH:

SNWE	a switch used to indicate whether or not to .FREB the current SH work queue entry
SWEAD	holds address of current work entry
SOL	online/offline switch plus signon active/not active flags plus signoff activity flags plus ACCESS-to-ACCESS mode flags plus auto restart flags
STFCS	current trarsmit FCS
SRFCS	current receive FCS
SRSVB	holds address of reserve receive buffer, when negative a buffer is pending
SACQ	used to determine if any input queues are active
SSTBY	holds address of buffer available after I/O completion
SCTRL	holds control queue name
SSIZE	holds specified communications buffer size
SIHBT	holds number of outstanding console messages which must be disposed of before continuing

SINQS holds number of input work queues actually available

SACQO holds number of active output queues

Definition of 'SH runnirg' message which appears in a control buffer:

BCW	Word 2	Bits	15-4	Zero
			3-0	command=3
	Word 3	Bits	15	(unused)
			14-0	Queue name (SH input work queue)
	Word 7	Bits	15-0	communications buffer size (negative 2's complement number of words)
	Word 8	Bits	15-1	(unused)
			0	0 = normal mode 1 = ACCESS-to-ACCESS mode
Data	Word 1	Bits	15-8	Stream RCB
			7-0	Queue limit

Note: this message is always placed on the recipient's work queue using .PRIQ (priority .PUTQ).

Definition of blocks appearing on input work queues is as follows. Most of these definitions are internally generated by SH:

BCW	Word 1		15	control transmissions (permissions/requests)
			14	signon transmission
			13	signoff transmission
			12	BCB and FCS are to be left alone
			11	BCB is to be left alone
	Word 2	Bits	3-0	command=2
	Word 7	Bits	15-0	length of the input data in positive bytes (see below)
	Word 8		15	0=not end-file 1=end-file

Note: the position of the data in input blocks must be as shown below. The communications buffer size noted above in the 'SH running' message allows for this format and size. The maximum

size of the input data section of this format will be dictated by the operator's specification in the SC command.

Data	Word 1	**reserved**
	Word 2	**reserved**
	Word 3	beginning of input data
	.	
	.	
	Word n	end of input data (length of words 3 through n is reflected in BCW word 7)

The first 5 bytes (characters) of the input data are supplied by the synchronous handler but must be included in the length. The last byte must contain an ETB character (octal 46).

Definition of messages given to output handlers by SH. Message appears in a communications buffer:

BCW	Word 2	Bits 3-0	command=3 (request to receive) command=1 (data block)
	Word 3	Bits 15	(unused)
		14-0	Queue name for response
	Word 8	15-8	RCB stream id
		7-1	(unused)
		0	0 = normal mode 1 = ACCESS-to-ACCESS mode

Note: these buffers are not interlocked. Response to the queue name in word 3 is cooperative not mandatory. The console output decompression handler will never receive a buffer with a command of 3.

The responses to SH by output handlers must follow these definitions:

BCW	Word 2	Bits 3-0	command=3 (permission to receive) command=5 (data block finished)
	Word 8	15-8	RCB stream id
		7-0	(unused)

When messages are queued for the console I/O handler, these BCW definitions are made. Their purpose is to allow console I/O to note the running state of SH:

BCW	Word 2	Bits 3-0	command=1
	Word 7	15-0	message length
	Word 8	15-14	00=normal message 01=communicatcns established 10=disconnectcn
		13-0	(unused)

Tables are defined which contain appropriate data regarding each input and output stream. The tables allow for suppcrt of the maximum possible number of each type of data stream. However, the number of streams actually present is determined at system initialization time, with thcse not present being disabled.

The Synchronous Transmit Stream Table (STST) defines each possible input stream:

STST entry 0 - console input
1 - card reader number 1
2 - card reader number 2
.
7 - card reader number 7

STST entry format:

Word	Bits	Contents
0	15-0	Stream FCS definition
1	15-0	Stream FCS state
2	15	Permission to transmit requested
	14-2	(unused)
	1	'SH running' message needed
	0	Permission to transmit granted
3	15	Stream presence or absence
	14-0	Queue name for producing handler
4	15-8	Stream RCB
	7-0	Queue depth limit
5	15	(unused)
	14-0	SH input queue name

The Synchronous Receive Stream Table (SRST) defines each possible output stream:

```
SRST entry  0 - console output
            1 - printer number 1
            .
            .
            7 - printer number 7
            8 - punch number 1
            .
            .
            14- punch number 7
```

SRST entry format:

Word	Bits	Contents
0	15-0	Stream FCS definition
1	15-0	Stream FCS state
2	15	Permission to transmit requested
	14-1	(unused)
	0	Permission to transmit granted
3	15	Stream presence or absence
	14-0	Queue name for consuming handler
4	15-8	Stream RCB
	7-0	Queue depth limit (the maximum number of blocks which will be received prior to suspending the stream via FCS)

Algorithm

Two local data structures are of special significance. These are the Synchroncus Transmit Streams Table (STST) and the Synchronous Receive Streams Table (SRST). The eight entries in the STST define the possible input work queues. The fifteen entries in the SRST define the possible output work queues. Contained in the tables are all parameters (flags, queue names, etc.) needed to use these queues plus all parameters needed to control the producers and consumers of these queues. Entries in these tables are used sequentially in a priority fashion. For example, the STST is ordered as follows: console input, card stream 1, ..., card stream 7. Thus, transmissions for card stream n will tend to take precedence over card streams greater than n. The console will always have greatest priority. This is, of course, subject to the metering of data by the line protocol. SRST entries are used in a similar manner.

The system initialization entry point to SH is SYNHI. The SH control queue name is found and saved in SCTRL. Finally, SYNHI "configures" SH by initializing the STST and SRST tables. This

involves attempting a .FIND for all possible work queues and associated producers and consumers. Those table entries for which the .FIND's are successful are enabled. Remaining table entries are disabled. On a system restart, if the auto-restart flag is on, a restart is attempted.

The prime entry to SH is SYNHP. All work queues associated with SH activate this same prime entry. SYNHP consists of three main sections. The first is the entry point section which removes work from the control queue and invokes processing of the work entries. Section two consists of a group of processing sections, each of which processes a control queue work entry. The third section is an I/O scheduler which is also used by the I/O complete entry point to SH. The I/O scheduler performs the task of examining the STST entries for transmission or other I/O scheduling.

SYNHP is entered by a system prime. It then invokes processing of each work entry found on the control work queue. The routines which handle these work entries all return to a common point where the current work entry is released to its buffer pool if appropriate. Other work queue entries are processed in turn until none remain. At this point the I/O scheduler is entered for possible I/O initiation.

These messages may appear on the SH control work queue:

SC, SD, or ER commands - the command is received from the console I/O handler and is processed by routine SRJEC. The work entry is in a consele buffer.

empty output buffer - this is a communications block which was previously received, has been disposed of via one of the output work queues, and is now being returned to SH by the associated consuming hardler. Its function for SH is to signal a reduction in the number of blocks on that output work queue. SH uses this technique to monitor the flow of data and use of communications buffers. SQCHK processes these entries.

printer allocation signal - received from a consuming data handler to indicate that all of its needed resources are available and that SH may commence receipt of data for the output stream. SPALL processes these entries. This work entry appears in a communications buffer.

allocated communications buffer - received from the buffer manager in belated response to a buffer request made by the I/O scheduler. SCOMB processes this work entry.

allocated console buffer - received from the buffer manager in belated response to some previous request for a console buffer. SCONB processes the entry.

allocated control buffer - received from the buffer manager in belated response to a previous request for a control buffer. SCTLB processes the entry.

Work entry processing by SYNHP is as follows:

SPALL - locates the SBST for the associated output stream. When in ACCESS-to-ACCESS mode, the RCB received is temporarily changed from an input type RCB to a printer type RCB in order to locate the SRST entry. A "permission to transmit" control record for the stream is constructed in the work entry buffer (a communications buffer). This buffer is placed at the beginning of the console input work queue (highest priority STST entry) for subsequent transmission. This is really just a trick to allow these high priority messages to be selected for transmission as soon as possible.

SQCHK - locates the SRST for the associated output stream and determines the current work queue depth. When in ACCESS-to-ACCESS mode, the RCB received is temporarily changed from an input type RCB to a printer type RCB in order to locate the proper entry in the SRST. If below the SRST defined limit, the stream control bit in the send function control sequence (FCS used in multileaving protocol) is enabled.

SCOMB - if the system is not online, the buffer is released. Otherwise, its address is saved in SRSVB for use by the I/O scheduler.

SCTLB - if the system is not online, the buffer is released. Otherwise, it is supplied to SHRUN for the issuance of "SH running" messages to available producers of input work queue data.

SCONB - the buffer is supplied to .CCO. for console output.

SRJEC - for an ER command, the data is acquired and the messages are constructed and issued.

For an SC command:

1. If SH is online or not online but engaged in going online, the 'SYSTEM ALREADY CONNECTED' message is issued.

2. If SH is not online and is not going online, the SC data (buffer size and signon record) are processed. Then the signon activity signals are set, the 'SYSTEM READY' message is issued, and the data line enable I/O operation is started.

For an SD command:

1. If a signoff is already in progress, the command is ignored.

2. If SH is not online or engaged in a signon, the 'SYSTEM DISCONNECTED' message is issued.

3. Otherwise, the signoff record is processed. Signoff conditions are established and the auto-restart flag is turned off to disable the restart feature.

All I/O scheduling (except for the initial data link enable and terminal clear) is performed by SIO. SIO is entered by SYNHP when control work queue is exhausted. SIO is also used by SYNHC to initiate succesive I/O operations. If I/O is active when SIO is entered, it exits to the central system commutator. Otherwise it performs the following sequence of steps:

1. If SH is no longer online, SPRGE is entered to clean up all queues, etc.

2. If the host has temporarily disabled transmissions to it (WABT or wait-a-bit), processing continues at step 7 in order to attempt a receive only operation. This will not occur when in ACCESS-to-ACCESS mode.

3. A scan of all input work queues now occurs:

   A. The first input work queue (normally console input) may contain permission to transmit control records. If so, these are immediately selected for transmission.

B. A local variable, SACQ, is used in the remainder of the scan to note the presence of active transmit streams.

C. If a data block appears on an input work queue, then:

a. If the stream's current FCS state is disabled (host has temporarily suspended this stream), then no further processing of this stream occurs.

b. Otherwise, SACQ is set to show at least one active input work queue.

c. If permission to initiate transmission has been received from the host, the data block is removed from the work queue. If the queue depth drops below the STST defined limit, an 'SH running' command is re-issued to the associated handler so that it may continue. If the block contains end-file, the STST transmit permission is cleared. (If SIO is currently active because of SYNHC and a secondary communications buffer is available, it is now released.) Transmission of the block is scheduled.

d. If no permission to transmit exists but has been requested, no further processing occurs. Otherwise, if a buffer is available, the request is constructed and scheduled for transmission.

4. This point is reached if no WABT exists and no input work queue transmissions can be scheduled. If SACQ shows active input work queues, processing continues at step 7 to attempt a receive operation.

5. If there are no active input queues, the output queues are checked. If some are active, a receive operation is attempted as at step 7.

6. If no work queues are active and a signoff is pending, the transmission of the signoff record is scheduled. Otherwise, a receive operation is attempted.

7. Processing continues at this point if no transmit operations can be scheduled. If a buffer is available, a receive only operation is scheduled. Otherwise an exit occurs.

8. I/O scheduling always involves application of the current SH transmit FCS which informs the host about the desired receipt of data by SH. I/O is then begun, and I/O activity is flagged. Finally, an exit occurs.

SIO utilizes a subroutine called SIOBF. This subroutine provides, if possible, the address of a communications buffer to its caller. The buffer may come from either of two sources. A standby buffer is one that SYNHC has determined is of no further use. This one, if available, is always given away first by SIOBF. Otherwise, if a primary buffer is available, it will be supplied. If there is no primary buffer, an attempt is made to acquire one. If this fails, one will appear later as handled by SCOMB.

SYNHC is the I/O complete entry point to SH. It processes as follows:

1. I/O activity flags are cleared. If any abortive error has occured, SPRGE is entered to clean up.

2. If the operation completing is a signoff transmission, a clear I/O operation is performed and SPRGE is invoked.

3. If the operation is a successful connection following the SC command's data line enable, BCW8 is checked to determine the mode of operation. If in normal mode, or ACCESS-to-ACCESS/slave mode, the sign-on record transmission is begun. If in ACCESS-to-ACCESS/master mode, the handler waits to receive the sign-on record from the slave system.

4. If the operation is signon completion, SH is placed online and the 'COMMUNICATIONS ESTABLISHED' message is issued. If a signoff has been requested, the signoff transmission section of SIO is invoked. Otherwise, the 'SH running' control signals are issued to all associated input handlers using SHRUN.

5. If the operation is a completed transmit/receive sequence, the received FCS is processed.

6. The RCB in the received data is then examined. In ACCESS-to-ACCESS mode, an input type RCB is changed to a printer-type RCB in order to locate the proper entry in the SRST. They type of RCB determines the processing as follows:

   A. If a request to transmit, and a signoff is pending, the request is ignored. Otherwise, the associated output handler is activated.

   B. If a permission to transmit, the STST is marked. In this case and in the case where step A leads to ignoring the request, the buffer is established as a secondary buffer for SIOBF. Then SIO is entered.

   C. If data, the block is enqueued to the proper output work queue. If that queue limit is reached, the associated FCS bit for the stream is disabled.

SHRUN is a subroutine used to present signals to associated input handlers informing them of SH activity and parameters. It searches for marked STST's requiring this signal, and transmits a message to the handler containing its intended RCB, input work queue name, queue limit, and current communications buffer size. SHRUN is invoked either by SYNHC following a signon or by SCTLB in belated response to the request for control buffers. It can also be invoked by SIO when an input work queue drops below its queue limit.

SPRGE purges all input and output work queues and associated handler queues. It resets SH to initial conditions, and issues the 'SYSTEM DISCONNECTED' message. Finally, it clears the purge indication from all SH work queues and closes the SH prime gate. If auto-restart is requested, a new sign on is attempted. Otherwise, processing is terminated.

Note: This document assumes knowledge of IBM's Binary Synchronous Communications procedure, and the HASP Multileaving conventions. For those persons unfamiliar with the Multileaving discipline, Appendix A of the HP2000 ACCESS Base Design Specifications for the base level program describes the HASP Multileaving protocol.

I. Product Identification

Product abstract

The D.50F Driver interfaces 2100 computers to telecommunications devices using the subset of IBM's Binary Synchronous Communications required by HASP Multileaving. In addition to the standard facilities of Multileaving, an auto answer capability is included in the program.

II. Design Overview

Design assumptions

The program requires the following for operation: time base generator driver (D.43), and the 12618 interface boards for synchronous transmission. System must also have appropriate common carrier equipment.

Additional software required by this program includes the HP2000 ACCESS I/O processor microcode.

Design summary

D.50F is an input/output driver designed to operate in the HP2000 ACCESS I/O processor. It will perform the following functions:

1. "Clear" - puts I/O board in "on-hook" condition. Resets status words.

2. "Write" - Transmits a data block. No error correction is done. No reception from remote is initiated. Used primarily to transmit a final block such as a signoff.

3. "Write/read" - Sends a block of data to the remote and then waits for acceptance/rejection from remote. If block is rejected, it will be resent. If accepted, any incoming data overlays the buffer.

4. "Answer" - Allows data line to be established by answering phone or by operator manual dial. It then initiates signon protocol with the remote.

5. "Extended Status" - Establishes a table for collection of statistics on device performance during a session.

Because the Multileaving protocol requires the EBCDIC code, the driver supports only the EBCDIC transmission code.

With one exception the driver always operates in a Wait-Before-Transmit Acknowledgement mode. WABT allows the processor to indicate a "temporary not ready to receive" condition to the transmitting station. When in ACCESS-to-ACCESS mode, the driver will not respond with WABT when it receives WABT in a normal WRITE/READ operation. Instead the driver will respond with ACK0 and continue the read operation.

The driver also supports Transparent mode, which allows data link control characters to be transmitted as data without taking on control meaning.

Design approach

D.50F is modular in its internal architecture. Each module may be classified as a "function processor" (modules are responsible for executing a particular function; i.e., "read", "write") or as a "common module" (modules or subroutines which aid function processors).

III. Design Structures

Major functional modules

NOTE #1: When a call is rejected, the following return codes will appear in the "A" and "B" registers:

A-REGISTER	B-REGISTER	Reason for reject
1	C	Illegal request
1	1	Local terminal is "off-line"
1	100000	Driver busy

Clear Request

Description and function

Puts I/O board in "on hook" condition (drops line). Clears status words except "extended status".

Interface description

The following calling sequence should be used:

```
JSB  .IOC.
OCT  XX          where XX is logical unit number.
```

This call is never rejected.

Algorithm description

1. Turn off I/O boards (drop telephone line).
2. Clear status word in EQT entry.
3. Return.

Programming hints:

Do not use the clear function with the intent of merely clearing the "present operation" since it will drop the communications line.

Read Request

Description and function

This request is not directly available. It is a part of WRITE/READ processing. Read a block of data from the remote terminal. Execute error detection and correction on the new data block.

The following error detection/correction procedures are used by the program:

1. The driver will re-read blocks of data received in incorrect parity up to 25 times. If still in error, a "DLE EOT" sequence will be sent to the remote and the line dropped.

2. If more than 3 seconds pass without receipt of a "SYN" character from the remote, the driver will consider the block in error and utilize the error processing in (1).

3. If a "NAK" is the first control character received, the driver will resend its acknowledgement sequence.

The following optional operational mode may be used:

1. The driver will automatically detect transparent text, delete "DLE" characters, and examine the characters following "DLE" for control characters.

Interface specification

The calling sequence for this request is discussed under WRITE/READ below.

The transmission log will reflect:

a) Characters received.

b) Include STX, SOH, and ETX characters.

Algorithm description

1. Process write request. (See below.)

2. Receive data block from remote.

3. Check parity on received block, requesting retransmission if the block was received in error.

4. Check for WABT received. If so, respond with ACKØ and continue the read operation. Else enable "WABT" time delay generation and terminate the read.

Write/Read Request

Description and function

Sends a block of data to the remote then waits for an acceptance or rejection of the block. A data block may be read as an acknowledgement, as well as an ACKØ. If the block is rejected by a NAK, it will be retransmitted.

The following error detection/correction facilities are implemented:

1. If remote fails to respond within three seconds after the block has been sent, a "NAK" is transmitted to request retransmisssion of the response.

2. If the remote's reply is invalid, a "NAK" is sent and the driver set to receive a new reply. This procedure will be repeated up to 25 times, whereupon a "DLE EOT" will be sent and the line dropped.

3. If the remote "NAK"s the block 25 times, a "DLE EOT" sequence will be sent and the line dropped.

4. If DATA SET READY signal status from the local modem drops, the I/O board will be turned off and the driver freed.

5. A reply of "DLE EOT" will cause the I/O board to be turned off, and operation ended.

The following cptional operations may be requested:

1. Transparent mode allows all data, including the normally restricted data-link line-control characters to be transmitted as data. Transparent mode is indicated by a "DLE STX" at the beginning of the block.

Interface description

The following calling sequence should be used:

```
JSB  .IOC.
OCT  22ØXX  where XX is the logical unit number
Reject Address
Start Address of Buffer
Length of buffer to be sent
```

Buffer:

WORD #1: Length of receive buffer.

WORD #1: Start of receive buffer.

WORD #3: Start of data to transmit.

The call may be rejected for the following reasons:

1. Local terminal "off-line".

2. Driver busy.

The following considerations apply when using a WRITE Request:

1. User program must place the following data-link control characters in the block to be sent:

   a) Place SOH, STX or DLE, STX at beginning of data.

   b) Place ETB at end of buffer.

2. A WRITE only operation can be specified by a request code of 2ØØXX. This call should be used only for the final write of a session and should be followed by a CLEAR request.

Algorithm description

1. Send data block to the remote.

2. Send block check characters.

3. Wait for remote to reply.

4. Take appropriate action for remote's reply.

Reply	Action
ACKØ, DATA BLOCK	Enable wait-before-transmit generator and end operation
NAK	Resend the block.
WABT	Reply with ACKØ and continue the read operation.

Answer Request

Description and function

Establishes line for local terminal. Allows either manual connection or auto answer connection.

The following error detection/correction facilities are utilized:

1. If remote fails to respond to the "SOH, ENQ" within three seconds, another "SOH, ENQ" will be sent. The process will be repeated up to 25 times, after which a "DLE EOT" will be sent and the line dropped.

2. If the remote replies with anything other than "ACKØ", processing will continue as if a timeout had occurred in (1) above.

Interface description

The following calling sequence should be used:

```
JSB  .IOC.
OCT  3ØØXX  where XX is the logical unit number
Reject Address
```

The call may be rejected for the following reasons:

1. Local terminal already "on-line".

2. Driver busy.

Algorithm description

1. Prime I/O board to send.

2. Wait for carrier from modem.

3. Send "SOH, ENQ" to remote.

4. Wait for response from remote.

5. If response is ACKØ, then we are in normal mode. Proceed to number 9.

6. If response is SOH/ENQ, then we are in ACCESS-to-ACCESS mode and will take on the role of slave. Proceed to number 8.

7. If response is ACK1, then we are in ACCESS-to-ACCESS mode and will take on the role of master.

8. Indicate ACCESS-to-ACCESS mode and master/slave role in EOT status word complete the answer request.

9. Activate "Wait-before-transmit" generator.

Extended Status Request

Description and function

The usage of this call is to pass statistics on the device's performance during a session.

Interface description

The following calling sequence is used:

```
JSB     .ICC.
OCT     120XX          where XX is the logical unit number
Reject  Address
DEF     Buffer
DEC     7

Buffer  BSS  7
```

After this call statistics on device performance will be maintained in the buffer specified by the requestor. The call should be made before beginning a session. The call will never be rejected and will always yield an immediate completion return.

Algorithm description

1. The following statistics are maintained in the requestor's buffer:

WORD 1	Number of buffer overruns.
WORD 2	Number of timeouts.
WORD 3	Number of "NAK"s received.
WORD 4	Number of data checks on incoming data.
WORD 5	Number of unrecognized responses.
WORD 6	Carrier losses.
WORD 7	Error code - contains reason that the line was dropped.

Value	Reason

0	Dropped due to CIEAR request.
1	Message sent/received 25 times with line errors each time.
2	Modem dropped ready.
3	Received "DLE ECT" from remote.
4	(Not used.)
5	Power failure

Status Request

Description and function

This request is used to pass to the user WORD #2 (device status) and WORD #3 (transmission log) of the device's entry in the equipment table.

Interface description

The calling sequence for this function is as follows:

```
JSB   .IOC.
OCT   4ØØXX        where XX is logical unit number.
```

Upon return the A and B register will contain status information as outlined below:

A - register contains status.
B - register contains transmission log.

Status Bits:

Bit	Set	Reset
14	Line dropped due to abortive error	No error

Power recovery processing

The P.50 power recovery appendage is called from the D.04 module. It first restores the interrupt section states (I.50 and J.50) using D.04? if necessary. Finally, the occurance of a power failure is marked for possible use by DIE as a disconnect type.

The above processing is inadequate if interrupt routines are not active. This is because the driver may be involved in a receive or transmit operation and power loss to the modem will cause these operations to fail to complete. However, this problem is overcome in the receive state by the receive timeout which is a standard part of all receive operations. The timeout will discover loss of data set ready and will lead to a disconnect.

For the transmit state, a 15 second "watchdog timer" is employed to monitor all transmission attempts. This timeout should never occur unless a power failure (or other modem catastrophe) keeps a transmission from completing. The timeout forces an interrupt leading to a disconnect.

Status Information

The transmission log reflects the number of characters received in a write/read operation. On answer completion, the status word is defined as follows:

```
bits  14  =0 no error
          =1 error
       1  =0 slave role in ACCESS/ACCESS mode
          =1 master role in ACCESS/ACCESS mode
       0  =0 normal mode
          =1 ACCESS/ACCESS mode
```

Host List Decompression Handler

Description and function

The host list decompression handler (HL0) has as its main responsibility the transformation of compressed data blocks from the synchronous handler to print or punch lines which are passed to the appropriate level 3 handler. It is also responsible for performance of the following control functions: Accept printer or punch allocation requests and perform the desired allocation, printing the RJE AWAITING PRINTERn or RJE AWAITING PUNCHn messages if necessary. Accept disconnect requests and force printer or punch disconnection. Accept printer or punch error notices and produce ATTENTION REQUIRED messages. Accept device end-of-file notices and release the device. This handler does not have an I/O Complete section, since it does not issue IOC calls. The module is coded in a "serially reusable" fashion, with all locally used data accessed from a unique storage block. Thus, additional printer or punch streams may be added by replicating this storage block.

Interfacing

HL0 interfaces with the level 3 handler as follows:

1. A start command is placed on the level 3 handler's work queue to activate it. It contains the stream RCB, the communications buffer size, the name of the queue to receive the response, and a flag to indicate normal or ACCESS-to-ACCESS mode.

2. Allocated print or punch buffers and emptied print or punch buffers are accepted from the level 3 handler. These are filled with decompressed data and given to the level 3 handler. At termination they are freed by HL0.

3. Error message buffers from the level 3 handler are used to report error conditions to the operator.

4. A stop command is sent to the level 3 handler at end-of-file or synchronous handler disconnect.

5. A 'start timed retries' command is sent to the level 3 handler when HL0 is notified that the printer or punch is not ready.

HLO interfaces with the synchronous handler by accepting blocks of compressed data for the printer or punch.

No BCW definitions are made by this handler. It conforms to those defined by the synchronous handler and printer handler.

Algorithm

A significant feature of the inputs to the routine is that they include a mixture of normal messages entered by .PUTQ calls (print blocks, allocate requests, purge queue request) and priority messages entered by .PRIQ calls (printer error, printer EOF, and buffer messages). As a result, upon entry to the handler the queue must be inspected to detect priority calls. Hence, a .SEEQ call is used to create a latch to different message processors. These message processors are discussed below. When a processor has finished it will execute a .GETQ to remove its entry and return to the .SEEQ latch or .COM., depending on the type of processing performed.

DISCO is entered for purge-queue requests, which only result from a communications disconnect. If a device is allocated, the routine deallocates the device, issues a purge command to the level 3 handler, releases any unused buffers, and resets local storage.

ALLOC is entered for printer or punch allocation requests. If the device is already allocated to this handler, the allocation request is delayed. Otherwise the routine attempts to allocate the device, sends an "AWAITING PRINTER" or "AWAITING PUNCH" message to the console, if necessary, and sets a timer loop to retry allocation. A pathological case of special interest is when a buffer cannot be obtained for the console message. In this case the entry latch for the routine is reset to allow no further actions until the buffer is received as a message. This technique is used elsewhere in the handler, but is not further elaborated in this document.

PRBLK is entered for print or punch block queue entries. A buffer is obtained from the free list and entry is made to .DCOM to decompress lines for output. The routine also sets carriage control, if necessary, and then ships lines to the level 3 handler. It is also responsible for indicating print or punch block processed to the Synchronous Handler when finished. Again, suspension may occur due to unavailability of a buffer, and the latch-restart technique is used to continue processing.

PRFLT is entered for printer or punch faults. PRERR is intered when the printer is not ready. PRERR issues an "ATTENTION REQUIRED" message to the console and is done. Again, a latch-restart for buffer not available suspension is used.

PREOF is entered when the stop command is returned by the level 3 handler. All print or punch buffers are freed by HLO and the device is deallocated.

PRBUF is entered for buffer messages. It stores the buffer message to allow proper operation of the latch-restart routines and then goes to .SEEQ to allow continuation of the suspended operation.

The only other entry to this handler is the initialization entry. This simply calls .FIND to find its own queue name for future reference.

Submodule descriptions

PDCC is the functional module in the handler which receives control after a printer buffer is obtained and determines the carriage control character to insert in the buffer. Print-then-space mode is always utilized by this handler, so control is always executed before printing. In ACCESS-to-ACCESS mode, the value in the SRCB (PCF control code) is moved intact to the high-order byte of the first data word. In normal mode, PDCC maintains the following data to determine the spacing and convert multileaving control characters to PCF control codes:

STATE	State indicator 0 - Last control was immediate (I) 1 - Last control was after print (A)
CH	Control hold Stores last control for control after print operations.

Input SRCB's may be considered as ordered pairs of data: (W,C), where W is when the control function is to be performed, I - immediate, A - after print, and C is the control operation. We may now define the operation of PDCC in terms of possible (STATE,(W,C)) combinations.

STATE	(W,C)	Operations
0-I	(I,C)	Carriage control <- C Decompress data and queue

0-I	(A,C)	Carriage control <- space 0 CH <- C; STATE <- A Decompress data and queue
1-A	(A,C)	Carriage control <- CH CH <- C; Decompress data and queue
1-A	(I,C)	Carriage control <- CH STATE <- I Data <- one blank; Queue data Adjust pointers to reprocess this block

Host reader compression handler

Description and function

This handler (HR0) allocates and manages a card reader handler in order to pre-process the card data for the synchronous communications handler. Pre-processing involves compressing and blocking card images into the acceptable communications format. The code for this handler will be written as a separate "serially reusable" module. The actual handler will consist of a block of storage for the local data structure and appropriate calling sequences to the code. HR0 is a level 2 function handler, and its interfacing to a card reader handler follows the level 2 - level 3 protocol.

Interfacing

HR0 utilizes its allocated reader as follows:

1. A start buffer is placed on the reader's work queue to initially activate it or to reactivate it. The form of this buffer is as prescribed by the level 2 - level 3 protocol.

2. Allocated reader buffers are accepted from the card reader handler. They are set to a read command and returned. These same buffers return as writes to HR0 with usable data. They are always given back to the allocated reader as reads except when terminating, in which case they are freed.

3. Error commands from the allocated reader are used to report the error to the operator.

4. HR0 sends a stop command to the allocated reader when HR0 is deactivating due to a TR command or synchronous handler disconnect.

HR0 interfaces with the synchronous handler by:

1. Placing compressed blocks of card images on the work queue indicated in the 'SH running' message. If the limit of this queue is reached, HR0 suspends until reactivated with a new 'SH running' message.

2. End-of-file signals from the reader are indicated to the synchronous handler as prescribed by its base design.

Local data structures

The following symbolic names are associated with the local data structure used by HR0. In fact, these names are used as indirect pointers to the local data and do not contain the data.

CCCPG holds the address of the prime gate for the active function handler

CCQNM hold the work queue name for the active function handler

CCFLG contains several flags:

CCSHR synchronous handler status (+=running, 0=not active, -=running but CRC is at queue limit)

CCPCR set = ok to process card data

CRDTA set = all compression resources available (communications buffer and not at queue limit)

CCEOF set = at end-of-file with not ready reader

CCTRC set = TR command is pending

CCWST set = Waiting for level 3 handler to stop

CCSPN set = start reader is pending

CCATA set = ACCESS-TO-ACCESS mode

CCRDR holds queue name of allocated reader

CCSHQ holds queue name to which compressed data is delivered

CCCCB holds address of communications buffer

CCCBP current position within communications buffer

CCCBE address of end of communications buffer

CCCBS holds size of communications buffer

CCRCB holds stream RCB for input to CCSHQ

CCSRC holds carriage control for SRCB when in ACCESS-to-ACCESS mode.

CCQLM holds queue limit for maximum number of buffers to place on CCSHQ

CCRET holds return address for delayed returns from subroutine CGCB

No buffer definitions are forced by HRO. It conforms to those required by the synchronous handler and the level 2 - level 3 protocol.

Algorithm

When this module is entered, register A is loaded with the address of the block of local storage for the specific compression function handler which is being primed or initialized. A routine named CADDS generates a series of indirect address pointers for the local storage in question.

HROHI is the system initialization entry point to HRO. The only function performed is to .FIND the HRO work queue name. This name is stored in CCQNM.

HROHP is the prime entry to HRO. Work queue entries processed include:

SH running - a signal from the synchronous handler that it is running. This message is received in a control buffer and informs HRO that it may initiate processing at any time. (It also informs HRO that it may continue if suspended due to queue limitations.) HRO will not do any work until this message is received. In the message are parameters needed to operate: the stream RCE, the communications buffer size, the name of the queue to receive data from HRO, the queue limit for this queue, and a flag to indicate if the system is in ACCESS-to-ACCESS mode. HRO is designed to include SR command processing as an inherent part of the 'SH running' message processing.

SR, TR, and RR commands - these messages are received from the console I/O handler as operator requests to activate and control use of a reader. These messages are not valid unless preceded by an 'SH running' message.

Reader data - card images from the reader allocated by CRC are blocked and compressed. Error indications from the reader are also processed.

Purged queue - the HRO work queue can be purged by the synchronous handler due to abortive errors or normal termination procedures. In any case, HRO deactivates its allocated reader, deallocates same, and quiesces until the next 'SH running' message is received. A stop command sent to the reader as a part of purge processing will also return to HRO later.

Control, console, and communications buffers - these may be received from the buffer manager in belated response to requests for buffers.

Control operations - when in normal mode, a control 0 represents end-of-file and all other controls are ignored. When in ACCESS-to-ACCESS mode, control 0 represents end-of-file and all other controls are for carriage control. The value is saved at CCSRC and is passed along in the SRCB for the next data to be compressed.

Allocation and management of the card reader handler is affected by three sources: activity signals from the synchronous handler, operator directives and device status signals from the card reader handler itself. The synchronous handler informs HRO (as well as other modules) that it is functioning by placing an 'SH running' work entry on the HRO work queue. Only when this signal has been processed can other signals be effective. Similarly, a purged HRO work queue (purging done by the synchronous handler) indicates that the synchronous handler is ceasing to function.

Between the "running" and "purged" signals, operator directives (Start Reader, Restart Reader, and Terminate Reader) are accepted. Note that the synchronous handler connection signal internally causes an SR command to be processed. A purge signal also implies a TR command. The SR command causes an allocation of a card reader handler to be attempted. A message indicating the result is issued in either case. If successful, initial conditions are established and a start buffer is acquired. This buffer is put on the card reader handler work queue with EBCDIC reading mode and the HRO name indicated. This will activate the card reader handler as described in its base design. An RR command is routed to the allocated reader for actual handling. The TR command inspects an HRO activity indicator and either deallocates the associated handler immediately if no cards are

being processed or flags the TR condition to be effected at end-of-file.

Once allocated and started, the received card images are compressed by the central compression module and put on the associated synchronous handler work queue. Two problems can interrupt this compression flow. When a communications buffer is not available, an indicator of this condition is set. If a card image appears on the HR0 work queue prior to the needed communications buffer, that card image is replaced on the HR0 queue (via .PRIQ) and the HR0 handler exits after closing its prime gate. This prevents further processing of card images until the needed communications buffer becomes available. Any errors indicated by the card reader handler to HR0 will result in the issuing of an error message. An end-of-file signal is propogated through the central compression routine to the synchronous handler. Following the processing of an end-of-file, the TR flag is inspected and appropriate action is taken. Read buffers are always returned to the allocated reader immediately following processing of the data in them. This technique insures that a read operation is continually being attempted and is the way in which the "hot reader" feature is implemented.

Host Message Decompression Handler

Description and function

The function of the host message decompression handler (HMO) is to receive blocks of compressed data from the synchronous handler, decompress the data, and transfer it to the console input/output handler.

Interfacing

This handler communicates with the synchronous handler and the console input/cutput handler. Its input queue entries are either blocks of compressed data from the synchronous handler or console output buffers requested by the handler itself.

Input buffers received from the synchronous handler are returned when the entire block has been decompressed. This will serve as a signal to the syrchronous handler that this handler is able to accept more work.

Local data structures

INBUF	address of the input data block.
OUTBF	address of the output buffer.
NDATA	address of the next data block to be decompressed. NDATA will be zero when there is no input blcck being processed.
SIZE	retains the size (number of bytes) of a decompressed message.
KDDC1	normally an unconditional skip - set to 'nop' if a decompressed message must be folded into a second conscle buffer.
KDOBF	normally unconditional skip - set tc 'nop' if the handler is suspended beca use it cannot obtain a buffer.
QNAME	console output decompression handler queue name.
CIONM	console input/output handler queue name.

Buffer control words associated with input queue entries are as defined by the synchronous handler.

Buffer control words associated with output queue entries are as defined by the synchronous handler (exhausted input buffers) and by the console I/O handler (output messages).

Algorithm

This handler has two entry points: an initialization section (HMOHI) and a main queue entry processing section (HMOHP). At initialization, the queue names of this handler and the console input/output handler are found via .FIND and saved.

The host message decompression handler receives blocks of compressed console messages from the synchronous handler. One block may contain several console messages. It is therefore necessary to determine whether or not a block of data is already being processed. If so, the output buffer processing section of the handler is entered. If not the input queue processing section is entered.

In the input queue handling section, a queue entry is obtained. It will be either an input block or a buffer. If it is a buffer, the decompress routine is entered. If it is an input block, and the handler is waiting for a buffer, the entry is put back on the top of the queue. Otherwise the output buffer processing section is entered.

In the output buffer processing routine, a check is made to determine if a buffer has already been requested. If so, the input queue handling section is entered. If not, a console buffer is requested. If the request is satisfied, the decompression section is entered. If not, the handler suspends itself until a buffer is acquired.

In the decompression section, one line of data is decompressed into an intermediate buffer using .DCOM. Depending on the size of the decompressed message, either one or two console buffers receive the text and are sent to the console I/O handler. If the input block is completed, it is returned to the synchronous handler. If not, no further processing takes place and the handler simply exits.

Host Inquire Compression Handler

Description and function

The primary function of the host inquire compression handler (HIO) is to receive remote command messages from the console input/output handler, compress them, and transfer them to the synchronous handler.

Interfacing

The host inquire compression handler will communicate with the console input/output handler and the synchronous handler.

Queue entries received from the console input/output handler will be RC-msg type commands. The message may or may not have 'RC,' in front of it. This handler will compress the messages and give them to the synchronous handler.

The only queue entry received from the synchronous handler is a control buffer indicating a 'SH running' command. The buffer will contain the synchroncus handler queue name, queue limit, communications buffer size, and the RCB for the console.

Buffers received as input will be communicatcns buffers requested by this handler.

Local data structures

One constant is established at initialization time and is never modified.

QNAME — console input compression queue name.

Two constants are used in the calling sequence of the centralized compression routine.

INBUF — address of the input buffer.

OUTBF — address of the output buffer.

Three constants are used only by the conscle input compression handler.

BFLMT — established when an output buffer is obtained and is used to determine whether or not another entry will fit into the output buffer.

QSIZE established when an input queue entry is obtained. If there are no more entries on the queue, the cutput buffer will be immediately transferred to the synchronous handler.

SFLAG normally zero. It is set to -1 if the handler is suspended because the SH queue limit has been reached.

Four constants are passed to the handler by the synchronous handler with the 'SH running' command.

QLIM the maximum number of entries that may be in the synchronous handler's input queue.

SHNAM synchronous handler queue name. When this value is zero, the handler is not operational and will discard any input queue entries.

CBSIZ communicatiens buffer size. This value is required in the request for a communications buffer.

CRCB RCB for the console

Algorithm

The console input compression handler has two entry points. In the initialization section, HIOHI, the handler's own queue name is found via .FIND and saved.

The operation of this handler is controlled by the synchronous handler. It will not receive any input queue entries until communications have been established and it has received a 'SH running' command from the synchronous handler. If communications are broken, the input queue will be purged by the synchronous handler and all queue entries will be discarded until the next 'SH running' command is received. If the synchronous handler input queue limit is reached, the host inquiry compression handler will suspend itself. Any queue entries received while the handler is suspended will be put back at the top of the queue. The handler will not process any entries until the next 'SH running' command is received.

The HIO handler will process as many queue entries as possible at one time. When there are no more queue entries, it

will complete the output buffer and transfer it to the synchronous handler even if it contains only one console message.

Appendix A

Multileaving Specifications

Description

"Multileaving" is a term which describes a computer-to-computer communication protocol originally developed by IBM for use with its HASP system. IBM's ASP, VS1-JES, VS2-JES2, and VS2-JES3 now also support the multileaving protocol. In a gross sense, multileaving can be defined as the fully synchronized, pseudo-simultaneous, bi-directional transmission of a variable number of data streams betweeen two or more computers utilizing binary synchronous communications facilities. It must be emphasized that this is not a full duplex protocol, although full duplex equipment may be used. Binary synchronous communications implies bi-directional transmissions, but in only one direction at a time. This protocol is as viable on half duplex communications links as it is on full duplex.

Major components

At the heart of the protocol is a technique commonly referred to as "conversational acknowledgement." Many communications protocols provide for acknowledging the correct receipt of data with special control sequences (ACK or ACK0 and ACK1). These protocols have the disadvantage of wasted time in the sense that the communications line must be "turned around" to transmit the acknowledgement. The acknowledgement is then transmitted, and the line is "turned around" a second time for more data transmission. Multileaving acknowledges the correct receipt of data with data. Thus, the protocol is conversational--transmit data, receive data, transmit data, receive data, etc. Only when an error occurs which requires retransmission, or when one of the two parties has no data to transmit, does this flow break. In these cases, error correction or hand-shaking procedures are initiated. It should be obvious that this conversational technique has a potential for greater communications line utilization. It also gives the participating computers greater potential since processing of received data may be overlapped with the transmission of data.

Another aspect of the multileaving protocol is its support of a variable number of independent data streams in each direction. Each block of data transmitted from computer A to computer B may contain one or more individual messages or records. Each message is tagged with its unique data stream identifier so that the receiver (computer B) can sort the messages out and route them to appropriate destinations. An additional feature of the protocol

allows the receiver of messages (computer B) to control its initial and continued receipt of the messages for each data stream. This is done as a part of the conversational transmissions back to the sender of the messages (computer A). It should be remembered that this protocol is bi-directional. Therefore, each computer may play both the roles of sender and receiver. So the statements just made about data streams in the A to B direction are also simultaneously true about those data streams in the B to A direction. It must now be apparent that the data making up each transmission typically contains at least three things: the inherent acknowledgement of any previously received data, one or more messages, and control information regarding data streams which may be returned.

A third important aspect of the multileaving protocol is its use of data compression. Each of the messages included in a transmission is compressed in order to remove character redundancy via encoding. This, of course, can also improve line utilization by reducing the actual number of characters in the transmissions.

Transmission philosophy

The basic element of each multileaved transmission is the character string. One or more character strings are formed from each message input to the computer for transmission. Messages are usually of the classical types (card images, printed lines, magnetic tape records, etc.). For efficiency in transmission, each message is reduced to a series of character strings of two types. These are (1) a variable length nonidentical series of characters, and (2) a variable number of identical characters. Due to the high frequency of blank characters, a special case is made in (2) above when the duplicate character is blank. An eight bit control field, termed a String Control Byte (SCB), precedes each character string to identify its type and length. Thus a string as in (1) is represented by an SCB followed by the nonduplicate characters. A string as in (2) above can be represented by an SCB and a single character (the SCB indicates the duplication count and the character following is the character to be duplicated). In the case of an all blank string, only an SCB is required to indicate both the type and the number of blank characters. A message to be transmitted is therefore a group of character strings, each preceded by its respective SCB. A special trailing null (zero) SCB delimits the message.

In order for the receiver of messages to be able to associate each message with a given data stream, an additional eight bit control field precedes the group of character strings representing the original message. This field, the Record Control Byte (RCB),

indicates the general type, function, and unique identification of each data stream. Note that the RCB also provides for the grouping of multiple messages of various types intc a single transmission block. A null (zero) RCB delimits the block. A second eight bit control field, the Sub-Record Control Byte (SRCB) is also included immediately following the RCB. This field may provide additional information concerning the message to the receiving program. For example, in the transmission of messages to be printed, the SRCB can be used for carriage control information.

For actual multileaving transmissions, a variable number of messages may be combined into a variable sized block (i.e. RCB, SRCB, SCB1, SCB2, ..., SCBN, RCB, SRCB, SCB1, ... etc.). The protocol allows for the conversational exchange of such transmission blocks. To allcw optimum use of this capability, however, the participating ccmputers must have the ability to control the flow of particular data streams while continuing normal transmission of others. This requirement becomes obvious if cne considers the case of the simultaneous transmissicn of two data streams tc a system for immediate transcription to physical I/O devices of different speeds (such as two print streams). It is also an obvious need in the case where one of the transcribing devices becomes not ready, and it is necessary to suspend the associated data stream. To provide for metering and suspension of the flow of individual data streams, a Function Control Sequence (FCS) is added to each transmission block. The FCS, which is receiver oriented, contains a sequence of bits, each associated with a particular data stream. The receiver of several data streams can temporarily suspend a stream by setting the corresponding FCS bit off in its next transmissicn to the sender of that stream. To resume the stream, the FCS bit can be turned on. In transmissions from A to B, the FCS sent by A tells B about those streams A wants to receive. The FCS received by A tells A about those streams B wants to receive.

For error detection and correction purposes, a Block Control Byte (BCB) is added as the first character of each transmitted block. The BCB, in addition to control information, contains a modulc 16 block sequence count. These counts are maintained and verified by both sending and receiving systems in order to exercise positive control over lost or duplicated transmission blocks.

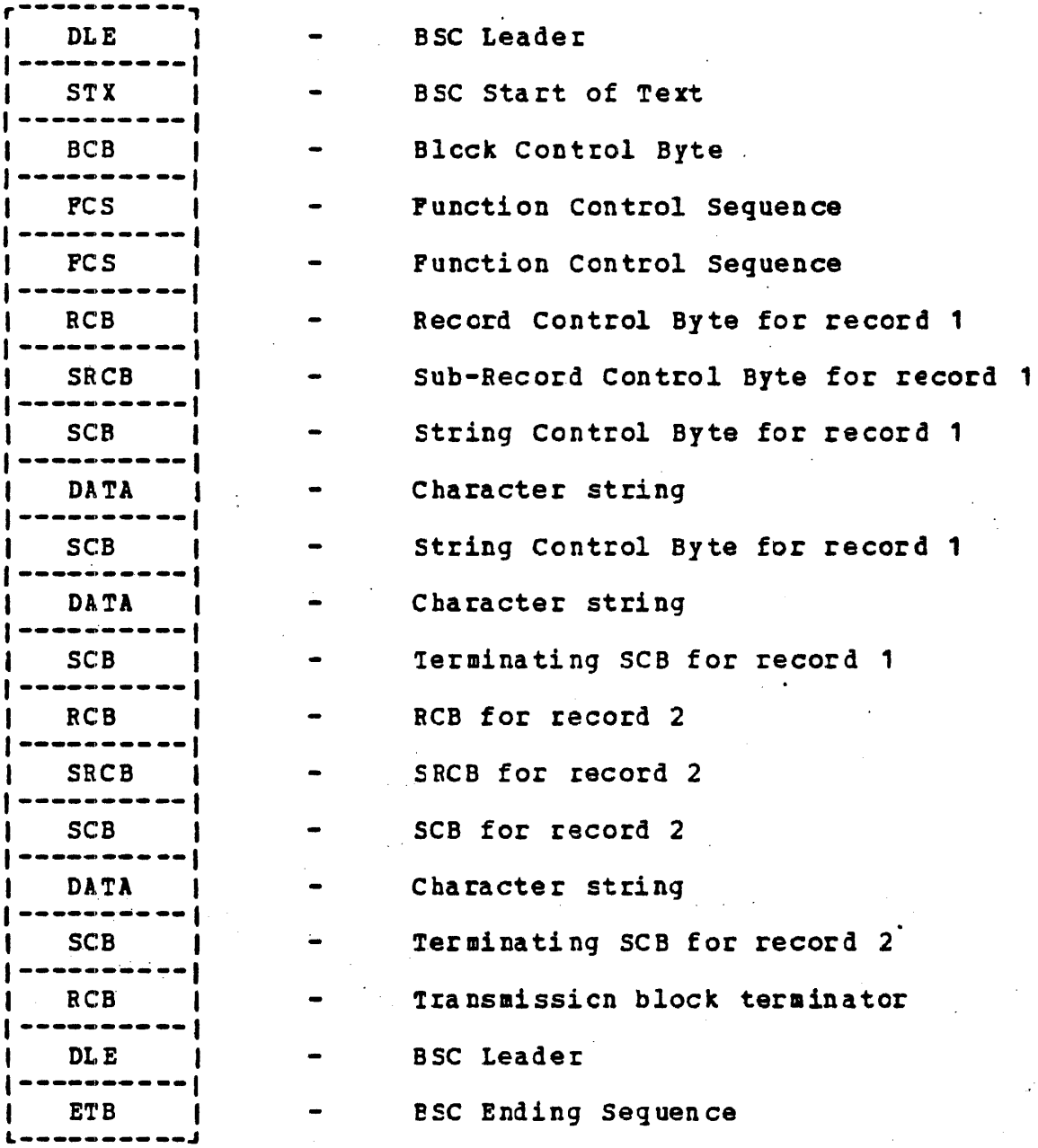
Transmission blocks are delimited by standard binary synchronous communications ccntrol characters (STX, ETB, etc.). Multileaving also uses the ACK0 and NAK control sequences. ACK0 is used as time fill for handshaking when data is not available

for transmission. Normally, handshaking occurs at two second intervals. NAK is used as the only negative response and indicates that the previous transmission was not successfully received.

Figure A illustrates a typical multileaving transmission block.

Figure A

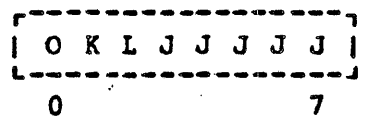
Typical multileaving transmission block



Multileaving control field specifications

Following is a complete specification of multileaving control fields. The reader should keep in mind that not all of the capabilities implied by these definitions have been implemented by IBM. At the end of this appendix, specific omissions in the implementation are noted.

String Control Byte (SCB)



Usage: This field identifies the type and length of a character string. One or more of such character strings represent a record or message for transmission.

Definition:

O = 0 = End of record (KLJJJJJ=0)
O = 1 = All other SCB's

K = 0 = Duplicate character string

L = 0 = Duplicate character is blank
L = 1 = Duplicate character is nonblank (and follows SCB)

JJJJJ = Duplication count

K = 1 = Nonduplicate character string

LJJJJJ = Character string length

Notes:

1. If KLJJJJJ = 0 and O = 1, SCB indicates record is continued in next transmission block.

2. Count units are normally 1 but may be in any other units. The units utilized may be indicated dynamically in the SRCB or other methods of indication could be designed.

Record Control Byte (RCB)

Usage: This field identifies each record type within the transmissior block. Further, it uniquely identifies individual data streams when the record types are the same.

Definition:

O = 0 = End of transmission block (IIITTTT=0)
O = 1 = All cther RCB's

III = Stream identifier used to identify streams of multiple identical functions (i.e. multiple print streams to a multiple printer system, etc.)

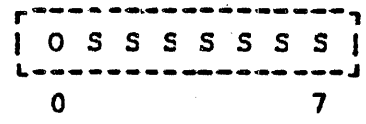
III = Control information if TTTT = 0 (ccntrol record)

= 000 = Reserved for future expansion
= 001 = Request to initiate a function transmission (Prototype RCB for function in SRCB)
= 010 = Permission to initiate a function transmission (RCB for function contained in SRCB)
= 011 = Reserved
= 100 = Reserved
= 101 = Available for local modification
= 110 = Available fcr local modification
= 111 = General control record (type indicated ir SRCB)

TTTT = Record type identifier

= 0000 = Control record
= 0001 = Cperator message display request
= 0010 = Cperator ccmmand
= 0011 = Normal input record
= 0100 = Print record
= 0101 = Punch record
= 0110 = Data set record
= 0111 = Terminal message routing request
= 1000 - 1100 = Reserved for future expansion
= 1101 - 1111 = Available for local modifications

Sub-Record Control Byte (SRCB)

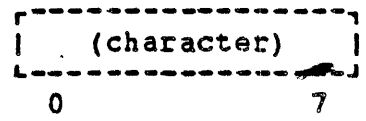


Usage: This field provides supplemental information about a record.

Definition: O = 1 (Must always be on)

SSSSSSS = Additional information - actual content is dependent on record type. Several examples are listed below.

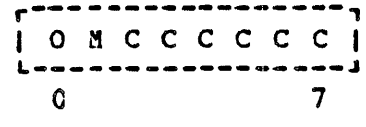
SRCB for General control record



Usage: Identifies the type of generalized control record.

Definition: character = A = Initial terminal signon
= B = Final terminal signoff
= C = Print initialization record
= D = Punch initialization record
= E = Input initialization record
= F = Data set transmission initialization
= G = System configuration status
= H = Diagnostic control record
= I - R = Reserved
= S - Z = Available for local modification

SRCB for Print records



Usage: Provides carriage control information for print records.

Definition: O = 1 (Must always be on)

M = 0 = Normal carriage control
= 1 = Reserved for future use

CCCCCC = Carriage control information
= 1000NN = Space immediately NN spaces
= 11NNNN = Skip immediately to channel NNNN
= 0000NN = Space NN lines after print
= 01NNNN = Skip to channel NNNN after print
= 000000 = Suppress space

SRCB for Punch records

Usage: Provides additional information about punch records.

Definition: O = 1 (Must always be on)

SS = Punch stacker select information

B = 0 = Normal EBCDIC card image
= 1 = Column binary card image

M = 00 = SCB count units = 1
= 01 = SCB count units = 2
= 10 = SCB count units = 4
= 11 = Reserved

RR = Reserved for future expansion

SRCB for Input records

Usage: To provide additional information for input records.

Definition: O = 1 (Must always be on)

M = 00 = SCB counts = 1
= 01 = SCB counts = 2

= 10 = SCE counts = 4
= 11 = Reserved

B = 0 = Normal EBCDIC card image
= 1 = Column binary card image

RRRR = Reserved

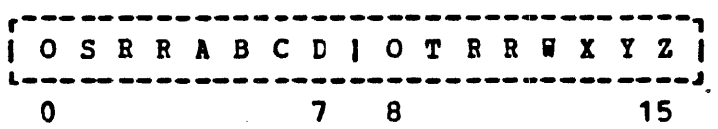
SRCB for Terminal message routing records

Usage: Indicates the destination of a terminal message.

Definition: 0 = 1 (Must always be on)

TTTTTTT = Remote system number (1≤T≤99)
= Remote system group (100≤T≤127)
= Broadcast to all remote systems (T=0)

Function Control Sequence (FCS)



Usage: Controls the flow of individual function streams.

Definition:

O = 1 (Must always be on)

S = 1 = Suspend all stream transmission (Wait-a-bit)
(A handshaking response of ACK0 implies the clearing of this bit.)
= 0 = Normal state

T = Remote console stream identifier

R = Reserved for future expansion

ABCD...WXYZ = Various function stream identifiers (oriented only to recipient.)

- Normal print (or input) = A, B, C, ...
- Normal punch streams = Z, Y, X, ...
- Other functions = ... - ...

Note - a bit on = continue function transmission
- a bit off = suspend function transmission

Note: Bits for the print and punch streams may have overlapped meanings for terminals with multiple devices. For example, bit "W" specifies function continue/suspend for both printer 5 and punch 4.

Block Control Byte (BCB)

Usage:

0 = 1 (Must always be on)

CCCC = Modulo 16 block sequence count

XXX = Control information as follows --

= 000 = Normal block
= 001 = Bypass sequence count validation
= 010 = Reset expected block sequence count to CCCC
= 011 = Reserved
= 100 = Reserved
= 101 = Available for user modification
= 110 = Available for user modification
= 111 = Reserved for future expansion

Initiating a multileaving connection

The procedure for establishing the connection of computer A to computer B is performed as fcllows:

1. A transmits a two character sequence (SOH,ENQ) to B.

2. B responds with ACK0.

3. A responds with the transmission of a block containing a single message which must be a signon general control record:

   DLE,STX,BCB,FCS,RCB,SRCB,signon message,DLE,ETB

   RCB must be 11110000. SRCB must be 11000001. The signon message is as defined by system B and does not appear in compressed form.

4. Assuming acceptance of the signon by B, normal processing may ensue. (Otherwise, further transmissions from B to A will usually cease.)

Performing a data stream transmission

The procedure for initiating and performing a data transmission from A to B is as fcllows. In the example, let X be the unique RCB associated with the data stream in questicn:

1. A transmits to B a request to initiate a transmission:

   DLE,STX,BCB,FCS,RCB,X,0,0,DLE,ETB

   RCB must be 10C10000. SRCB must be X (the prototype RCB for which permission is being requested).

2. B responds with a permission:

   DLE,STX,BCB,FCS,RCB,X,0,0,DLE,ETB

   RCB must be 10100000. SRCB must be X (the RCB for which permission is being granted).

3. A now proceeds to transmit the data stream messages. (Refer to "Transmission philosophy.")

4. A indicates end of a data stream (end-of-file) by using a terminal message which has a zero SCB for its first

SCB. Note that an all blank message cannot be transmitted as a null message.

NOTE For console streams, permission is never sought nor granted. End-of-file is never used. Only the processing noted in step 3 occurs.

Error detection and correction

Each transmission block is initially checked using convential cyclic redundancy check techniques. NAK is used to request retransmission when the check yields an error. NAK is also used when no data is received for a period of three seconds.

Several other logical error conditions can also occur:

BCB sequence check - To insure that data blocks are received in order and that no blocks are missed, the following BCB testing is done: Let x be the received BCB and Y be the expected BCB.

If X-Y>0 then a block has been missed and a sequence error is detected.

If X-Y=0 then the proper block has been received and it is processed. (Y is incremented as teh next expected X.)

If X-Y<0 then:

a. If 16+X-Y>2, then a sequence error is detected.

b. If 16+X-Y≤2, then the received block is a duplicate and is discarded.

If a sequence error is detected, the following transmission is used:

DLE,STX,BCB,FCS,RCB,Z,DLE,ETB

BCB usually indicates reset block check count. RCB is 11100000 to show the error. Z reflects what the expected BCB was (Y in above example). The receiver of this message may reset his transmitted BCB to Z. However, data has probably been lost, and restarting the current transmissions or aborting the connection may be more appropriate.

Duplication of requests to transmit - these may be ignored. However, the IBM HASP system aborts the connection in this case.

Non-existent RCB's or prototype RCB's - these are ignored. Any unprocessed messages in the transmission are ignored.

Special notes

Certain features of the multileaving protocol, though defined, have not been implemented by IBM. These include:

1. Record types other than print, punch, card input, console, and control are not supported.

2. Only the signon general control record is used.

3. SCB counts other than one are not used.

4. No "continued" SCB's are used.

5. No support for non-EBCDIC transmissions exists.

6. Although the protocol allows a mixing of RCB types in a given transmission block, this is not done in practice. Consider messages A, B, C in a transmission block each for a different data stream (mixed RCB's). If the resources to handle message A are not available, then messages B and C might be locked out as well. Thus, data streams B and C are being artificially inhibited. By keeping like RCB's in a transmission block, stream A could be suspended while transmissions for B and C could continue.

I. Introduction

HP2000 ACCESS CDC RJE Option

As outlined in the Product Requirements of September 25, 1974, this project will be an option to the currently defined IBM RJE component of the HP2000 ACCESS system, a CDC compatible RJE component. This RJE function, emulating the CDC 200 User Terminal, will provide remote access to a CDC machine. Concurrent with this RJE capability, the I/O Processor will continue to support the 32 terminals accessing the TSB system.

II. Functional Specifications

Following the implementation of these enhancements, two distinct functions will exist in the HP2000 ACCESS system. The obvious function is the TSB software which will continue to exist and operate as in the past. In addition, the I/O Processor will be able to function as an RJE station. In this way the HP2000 ACCESS system may serve an installation for time sharing at the same time it provides access to a remote CDC system (referred to as the host system).

The RJE function will be based on the CDC 200 User Terminal synchronous telecommunications protocol. It will operate on data in the ASCII or BCD codes. Job input to the host is in the form of 80 column card image data. Job output from the host is in the form of printed lines. Host operating systems generally refer to these types of inputs and outputs as readers and printers respectively. (There is also input and output in the form of console data.) The design of this software, however, is such that there need not be a fixed association with physical devices. It will be possible through HP2000 ACCESS operator commands to designate the actual devices to be used. In addition to real card readers and line printers attached to the I/O Processor, data streams between the TSB system and the I/O Processor will be available. Functioning as ASCII files available to BASIC programs, these data streams will also be available for communication with the host system. Thus, transmission of data to the host from a BASIC program might be substituted for transmissions from a real card reader. Similarly, output might be directed to a BASIC program rather than to a real line printer. There will also be BASIC ASCII file access to the console data streams. Since the CDC 200 User Terminal will operate with several CDC host operating systems, no attempt is made in this document to outline host-specific operating procedures. Appendix A of this document does provide a list of some CDC manuals which may be useful.

TSB users

The 32 users attached to the TSB system through the I/O Processor will continue to utilize the system exactly as described in the HP2000 ACCESS user's manual. Use of the special ASCII files for host communications will follow normal ASCII file rules. The names for these ASCII files will be JT0, JL0, JI0, and JM0 (Job Transmitter, Job Line printer output, Job Inquiry, and Job Messages). The term "job" is used by CDC host systems to refer to a unit of work performed for a user of such a system. The job inquiry and message functions correspond to console input to the host and console output from the host respectively.

Except where necessary, the remainder of this document will deal primarily with the RJE function as it relates to the use of real devices. For information on the use of the ASCII files, refer to appropriate sections of the HP2000 ACCESS user's manual.

TSB operator

The TSB operator assumes an added role as a result of the enhancements. In addition to managing the TSB system, he becomes an operator of the RJE station as well. He is both a TSB operator and an RJE operator. In this capacity he has these new duties:

1. Initiate the connection of the system to a host CDC system.
2. Assign input and output devices to the host reading and printing functions.
3. Operate the input devices in order to transmit jobs to the CDC system.
4. Monitor the line printer in order to separate listings and load forms.
5. Make remote operator inquiries and/or control the job flow via remote operator commands.
6. Initiate disconnection of the system from the CDC system.
7. Request a report of communications errors.

Before describing the operational details of these new duties, it is appropriate to describe the function of the operator console. The operator console is physically attached to the

System Processor, not to the I/O Processor. Communications between the two processors is via a computer to computer interconnect kit. Enhancements to the System Processor will allow messages (commands or inquiries) destined for the remote CDC system to be entered on the System Processor's console. These will be sent to the I/O Processor via the interconnect kit for actual handling or transmission. In addition, operator messages received from the CDC system will be sent through the interconnect kit to the System Processor for display on the system console. It is important to note that messages received as a result of TSB use (inquiry via JIO) will also be logged on the system console.

III. Interface Specifications

In this section, all commands for the RJE function are described. Specific responses to each command are discussed. However, if the RJE command can not be interpreted as one of the legal commands shown, the following error message will be received:

RJE COMMAND ERROR

Another possible error is the use of commands at inappropriate times. Some commands are inappropriate if entered when the RJE function is not connected to a host CDC system. If this fact applies to a command, it will be noted in the description of the command. The following general message is then used in response to a command entered at an improper time:

RJE COMMAND NOT APPROPRIATE

Note that all commands and all related responses begin with the characters "RJE" which allows easy visual separation of RJE functional activity from TSB activity. The RJE function will take a "back seat" to TSB users if the system becomes heavily loaded with work. Should this occur, responses to commands may be delayed, and activity on the associated I/O devices may degrade. This is done to prevent interference with the TSB user.

System connection

If the operator wishes to connect (or re-connect) the HP2000 ACCESS system to a host CDC system, he does the following:

1. Enter the following command on the system Processor console:

RJE-SC,n,m (SC=System Connection)

where n is the site address (octal 160 to 177). This number is determined by the host CDC system.

m is the code (ASCII, IBCD, or BCD) to be used for the duration of the connection to the host CDC. Default specifications of n and m will be:

n=160
m=ASCII

Subsequently, they will default to the last values used. Either operand may be omitted. If n is omitted, the form of the command must be:

RJE-SC,,m

After entering this command, the operator may receive the message:

RJE SYSTEM ALREADY CONNECTED

This message indicates that a previous connection is still in effect. The operator may then accept this existing connection or force a disconnection (see below) and retry the system connection described here.

Normally, the following message will be received:

RJE SYSTEM READY

The operator should proceed to step 2.

2. Ready the communication line. This will typically involve dialing the CDC system, waiting for an answer, and then placing the data set in data mode. If non-switched communications facilities are in use (a "dedicated" or "leased" line), no action is usually required to ready the line. If it is possible for the CDC system to call its remote stations and the data set in use has an auto-answer capability, the TSB operator may elect to ready the communications line by enabling the auto-answer. When the CDC system calls, the RJE function will then continue automatically.

3. As soon as the connection has been established, the TSB operator will receive this message:

RJE COMMUNICATIONS ESTABLISHED

Device assignment

A data source or destination must be associated with each host reading and printing function. When the I/O Processor is loaded and started, a default assignment of these sources and destinations will occur. However, the operator may wish to change this assignment from time to time, so a command is available for this purpose.

Physical devices attached to the I/O Processor and the TSB ASCII files for host communications are designated by a two character name and a number used to distinguish multiple like devices:

CRn	-	card readers
JTO	-	job transmitter
LPn	-	line printers
JLO	-	job line printer output

All ASCII file names are numbered relative to zero. Hence, the "n" value in these names is also relative to zero.

Similarly, the individual host reading and printing functions are designated as follows:

HRn	-	host reading function
HLn	-	host line printing function

For these names, the "n" values start with one. This is to provide for better association with the names commonly used by the host systems to refer to these functions.

For completeness' sake it should be noted that there also exist host inquiry (HIO) and host message (HMO) functions. These functions are, however, always associated with the system operator's console and the Job inquiry/Job message ASCII files. Therefore, they are not assignable by operator command. For the operator to make a device assignment, he needs to designate an association between one of the physical devices or ASCII files and one of the host functions. Naturally, some assignments would not make sense, for example a card reader with a printing function. Therefore, in order to assist the operator in making the proper assignments, an "arrow" ("<" or ">") is used in the command to

show the direction of the data flow. The general form of the command is as follows:

```
        RJE-DA,hf<dev
or      RJE-DA,hf>dev
```

where hf designates one of the host functions and dev designates one of the available devices. The first form is used to assign devices to the reading function (the arrow indicates flow of data from the device to the host: RJE-DA,HR1<CR0). The second form is used to assign devices to the printing function (the arrow indicates flow of data from the host to the device: RJE-DA,HL1>LP0).

These assignments may be made at any time. However, they will only become effective at the time the use of a device begins. For input devices, use begins at the time of the SR command and ends at the time of a TR command (see below). For output devices, use begins with the initial receipt of data from the host and ends when that output is complete. Obviously, no use of devices can occur except between a system connection and disconnection.

When the I/O processor is initially loaded, default assignments will be made. These defaults are determined by the operator at the time his system is configured.

One other form of the command is available to be used for display of the current device assignments:

```
RJE-DA
```

Responses will be of the form:

```
        RJE hf<dev
or      RJE hf>dev
```

where hf and dev are as discussed above.

Device characteristics

The aforementioned devices have the following characteristics:

Designator	Description	Max Buffer size	Function
CRn	Card reader	40 words	read only
JT0	Job transmitter	40 words	write only
LPn	Line printer	66 words	write only
JL0	Job line printer	67 words	read only

JIO	Job inquiry	36 words	write only
JMO	Job message	60 words	read only

For the JL device only, the first word of each buffer is used to supply carriage control information to the BASIC program. This information is in the form of a carriage control character followed by a null character. The actual print data begins in the second word of the buffer. Carriage control characters for JL are equivalent to those for the BASIC CTL construct. They will generally require use of the NUM function in a BASIC program for proper use.

Reading operations

Let us refer to the device assigned to the host reading function as a reader. Two commands are available for activating or deactivating the reader with respect to the RJE function. To attach the reader, the following command is used:

RJE-SR1 (SR=Start Reader)

If the reader is or can be assigned to the RJE function, the following message is received:

RJE READER1 AVAILABLE

If the reader is presently active with some other function, the following message will be received:

RJE READER1 NOT AVAILABLE

To detach the reader from the RJE function, issue the following command:

RJE-TR1 (TR=Terminate Reader)

As soon as the reader is finished reading any remaining data, the following message will be received:

RJE READER1 NOT AVAILABLE

These commands are also generated internally by the system at certain points. A start reader command will be attempted automatically at the same time as the connection to the host CDC is completed. A terminate reader command will be done automatically at the same time a system disconnection is done. In these cases the appropriate messages will be shown to the operator. That is, he will see the "RJE READER1 AVAILABLE" or "RJE READER1 NOT AVAILABLE" message. After the reader is once

started, it is always active and attempting to read. Thus, the operator need only ready the reader for the data to be read. This action is equivalent to depressing the "LOAD" key on a CDC 200 User Terminal.

Since some card readers do not have an "end-of-file" feature, it is necessary to use a special card image to delimit the end of a set of data read by the reader. The format of this card is:

:: (two colons)

The two colons begin in column one of the card image. The accidental appearance of the "::" image at any point other than the end of a set of data is considered an operator error. If this occurs, the image will be ignored.

Two types of problems may occur at the reader which require operator attention. One type of problem includes the normal physical errors (such as card jams, etc.) and failure to terminate a set of data with the "::" card image. Should any of these situations occur, the following message will be issued:

RJE READER1 ATTENTION REQUIRED (bell)

The operator should correct the problem and ready the reader.

The second problem area is the detection of illegal characters in the data read from a card. (This problem can occur only with card reader devices.) Only valid Hollerith codes can be read and transmitted to the host system. If any illegal characters are detected, the reader will halt and the following message will be issued:

RJE READER1 DATA ERROR (bell)

The operator will find the offending card last in the stacker. He may correct the card and replace it in the hopper, or he may ignore the card. After taking either of these actions, he must issue the following command to cause the reader to continue:

RJE-RR1 (RR=Restart Reader)

All three reader commands (SR, TR, and RR) are inappropriate if used when the RJE function is not connected to a host. Use at such times will result in the "NOT APPROPRIATE" response.

One additional RJE command is related to reader operations. Occasionally, when using a 200 User Terminal, the host will terminate card reading because of some erroneous data found on the

cards being read. The usual response is for the operator to correct the deck of cards, reposition the cards in the input hopper, depress the 200 User Terminal "LOAD" key, and then enter a command to the host to restart reading. Obviously, the use of the "LOAD" key is optional, depending on whether card repositioning is necessary or not. For this HP2000 ACCESS RJE software, the "LOAD" key is emulated with the following command:

RJE-LO (LO=Load)

It is intended for use only in correcting problems as noted above. The command is not needed to effect any initial reading of cards. Use of the command at inappropriate times will result in the discarding of any cards already read but not yet transmitted to the host. Of course, this is exactly the desired action when correcting problems as noted above.

Two special codes are used in card images for the 200 User Terminal. These are the 6/7/8/9 and 7/8/9 punches. Use of these codes from a real card reader is straightforward. However, if the JTO ASCII file capability is being used, the codes must be entered as follows. For the 6/7/8/9 code, substitute the extended string literal '28. For the 7/8/9 code, substitute the extended string literal '30.

Output Operations

Let us refer to the device associated with host output as a printer. Whenever data is ready for transfer from the host to the HP2000 ACCESS system, it will be accepted unless some other process (such as a TSB user) currently controls the printer. In the case of such contention, the RJE function will wait until the printer is released by the user before accepting the data from the host CDC system. Whenever the RJE function is unable to allocate the printer, this message will be issued:

RJE AWAITING PRINTER1

Except to correct errors and to load forms, etc., the operator's function is largely passive. If any paper jams or other "not ready" conditions occur at the printer, the operator's attention will be required. In this case, he will be notified of the problem with the following message:

RJE PRINTER1 ATTENTION REQUIRED (bell)

A CDC host does not necessarily indicate the end of a print operation to the 200 User Terminal. For the HP2000 ACCESS system this could mean that deallocation of a line printer by the RJE

function would be delayed. For this reason, two methods are used to release the printer. Whenever the host switches from printer to console output, the printer will be deallocated. In addition, the operator may force deallocation by using the following command:

RJE-TP1 (TP=Terminate Printer)

One additional comment about printed output should be made. The RJE software allows a maximum of two blocks of print lines to be received from the host at any given point in time. If for some reason, the output device (JL0 or PR0, etc.) does not dispose of this output, degradation of other RJE functions (card reading) may occur. However, the operator will always be able to transmit console input to the host. This generally causes a host to interrupt printed output and will prevent deadlock situations if serious output problems should occur.

Remote Inquiries and Commands

The CDC system generally has available a set of commands and inquiries which the RJE operator may use. It is assumed that the operator has access to or knowledge of the commands he may use. For the TSB operator, such commands and inquiries are entered through the System Processor console as follows:

RJE-RC,m (RC=Remote Command)

where m is a string of characters making up one of the legal CDC commands or inquiries.

Any responses from the CDC host or any other operator messages received at any time from the CDC host will be sent to the System Processor for display on the console as follows:

RJE m

where m is the string of characters making up the message received.

The RC command is inappropriate if the RJE function is not connected to a host. The "NOT APPROPRIATE" response would be received. The "INT" key function of a CDC 200 User Terminal is not directly emulated by this RJE software. This is because access to the console is always possible even when card reading or line printing is active. Thus, the RC command may be used at any time.

System Disconnection

When the TSB operator wishes to discontinue the RJE function, he performs the following steps:

1. Enter the following command on the System Processor console:

   RJE-SD (SD=System Disconnect)

2. Receive the message:

   RJE SYSTEM DISCONNECTED (TYPE n)

The "n" in the above message indicates the type of disconnection that occurred. This message may be received at times unrelated to the use of an SD command. For example, simply hanging up the data set without entering SD first would result in a type 1 disconnect. These types of disconnections are possible:

0 normal disconnection (always invoked by the RJE operator)

1 line break or other failure in the communications equipment

2 apparent host failure (no reception from host for two minutes)

3 power failure

Following any system disconnect, a new system connection may be performed. Any repeated use of this command will result in a repeat of the above response.

Communications Errors Report

To aid in observing communications line quality, a report of communications errors may be requested as follows:

RJE-ER,R (ER=Error Report)

where R is an optional parameter described below. The following report will be received:

```
RJE n1 ERRONEOUS STATION ADDRESSES
RJE n2 ERRONEOUS CONTROL CODES
RJE n3 CHARACTER PARITY ERRORS
RJE n4 MESSAGE PARITY ERRORS
```

RJE n5 CARRIER LOSSES
RJE n6 MISSING WRITE CONTROL CODES

Each n is a count of the particular error. Counters may optionally be reset to zero when the report is requested by specifying the R (reset) parameter. The error types are:

ERRONEOUS STATION ADDRESSES -- the station address contained in the message was incorrect or for a non-existent station.

ERRONEOUS CONTROL CODES -- an invalid operation from the host.

CHARACTER PARITY ERRORS -- bad parity on a received character.

MESSAGE PARITY ERRORS -- LRC character is incorrect.

CARRIER LOSSES -- carrier on signal dropped before ASCII end-of-text was recognized.

MISSING WRITE CONTROL CODES -- write message with no E1, E2, or E3.

This command may be issued at any time and as often as desired.

Command pauses

A real 200 User Terminal uses a CRT for display of console output. This is obviously a much faster device than the HP2000 ACCESS system console. The CDC host, not aware of this problem, could flood the TTY console with output. For this reason, the RJE software introduces pauses which allow the operator to enter commands. This is, in most cases, an action analogous to the effect of the "INT" key used at the 200 User Terminal. The pauses are approximately 15 seconds long.

Power Fail/restart

If a power failure occurs, some or all of these things will happen:

1. The communications equipment (modem) may lose power.

2. Terminal ready signals from the computer to the modem will drop.

3. The host system will be unable to communicate with the RJE function of the HP2000 ACCESS system and will

initiate error recovery procedures. This typically involves disconnecting the attached remote station.

For these reasons, if the RJE function is active at the time of a power failure, a system disconnection will be performed on a restart following power failure.

IV. Operating Specifications

Hardware Requirements

The HP2000 ACCESS RJE component requires the following hardware:

12618 Synchronous Data Set Interface Kit

Instructions for Loading

The I/O Processor is configured and loaded by the I/O Configuration program.

V. Appendices

Appendix A -- Representative list of useful CDC manuals

Appendix A

Representative list of useful CDC manuals

For information regarding the CDC 200 User Terminal, refer to the following manual. It contains a description of the terminal hardware and general operating procedures. Analogous capabilities have been included in the HP2000 ACCESS RJE component.

CDC 200 User Terminal (Publication No. 82128000)

For CDC host system operating information related to the CDC 200 user Terminal, refer to the following manuals. In most cases, these manuals will contain further references to other useful CDC literature.

Computer Systems Reference Manual (Publication No. 60100000)
INTERCOM Reference Manual (Publication No. 60307100)
SCOPE 3.4 Reference Manual (Publication No. 60307200)
KRONOS General Information Manual (Publication No. 60407100)

Synchronous Communications Handler

Description and function

The synchronous communications handler (SH) manages all I/O on the communications line. Once activated, it manipulates and controls the transmission and receipt of data within the defined CDC U200 Communications protocol. SH is activated at the request of the operator via the RJE-SC command. Activity then continues until an RJE-SD command or abortive communications line error occurs. SH will report data regarding line condition to the operator when directed to do so with RJE-ER.

The U200 provides for the support of one card image input stream, one printer image output stream, and remote console operator input/output streams. The input streams are managed via two distinct SH input work queues. Communications between SH and its data producers and consumers follows conventional handler work queue exchanges.

When a system connection is made, handlers are enabled to either produce data (input streams) or consume data (output streams). SH then interfaces with the host system via the U200 protocol in order to control the flow of data appearing on these queues. Additional information about the U200 protocol may be found in the D50CD synchronous driver base design.

Interfacing

SH interfaces directly with the console I/O handler as follows:

1. SH accepts the SC, SD, and ER commands at any time and the LO command during an RJE session. These must appear in console buffers and on the SH control work queue.

2. SH places messages on the console I/O work queue via .CCO.

SH interfaces with <u>any</u> one of its input modules as follows:

1. SH places a 'SH running' message on the handler's work queue. This message informs the handler of current SH parameters and indicates that the handler may proceed. Contained in the message are:

Queue name - name of SH input work queue on which the handler is to place its data blocks.

2. SH accepts data blocks from the handler on the specified work queue. (A BCW definition for the final block in a series of blocks is adhered to by the card image input handler.)

3. SH may purge the handler's work queue if an abortive line error occurs and will definitely purge when a normal disconnect occurs. The handler should view this as an 'SH not running' signal.

4. SH will place a TR command (in a console buffer) on the handler's work queue if a signoff is to occur. (This is not true for the console input handler.)

5. SH will purge the card compression handler's work queue when it receives an LO command. It will the place an "SH running" message on the card compression handler's queue to restart the reader.

SH interfaces with any one of its output modules as follows:

1. SH places data blocks on the handler's work queue. When the handler disposes of a block, it must be returned to SH to signal a reduction in that handler's queue level.

2. SH may purge the handler's work queue if an abortive error occurs and will definitely purge when SH performs a signoff. This should be viewed as a signal to quiesce.

3. SH will place a TP command on the printer output work queue if it has been active and the host starts writing to the console instead of the printer.

Local Data Structures

These local data are used by SH:

CTLQ holds control queue name

CQENT holds address of current work entry from prime gate

DQENT holds address of current work entry from I/O completion

SIOSW indicates current host communications activity

SIO I/O activity flag

Definition of 'SH running' message which appears in a control buffer:

BCW	Word 2	Bits	15-4	Zero
			3-0	command=3
	Word 3	Bits	15	(unused)
			14-0	Queue name (SH input work queue)

Note this message is always placed on the recipient's work queue using .PRIQ (priority .PUTQ).

Definition of 'end-file' flag for blocks appearing on input work queues:

BCW	Word 2	Bits	15-4	(unused)
			3-0	command=2
	Word 8	Bits	15	0=not end-file 1=end-file

Definition of messages given to output handlers by SH. Message appears in a communications buffer:

BCW	Word 2	Bits	15-8	device escape code ("E" code)
			7-4	(unused)
			3-0	command=1 (data block)
	Word 3	Bits	15	(unused)
			14-0	Queue name for response

The responses to SH by output handlers must follow these definitions:

BCW	Word 2	Bits 15-8	device escape code ("E" code)
		7-4	(unused)
		3-0	command=2 (data block finished)
	Word 8	Bit 15	for console output handlers only, this bit set indicates that a command pause has been terminated by console input. SH uses the indication to suspend console output until after the console input transmission.

When messages are queued for the console I/O handler, these BCW definitions are made. Their purpose is to allow console I/O to note the running state of SH:

BCW	Word 2	Bits 15-14	00=normal message 01=communications established
		13-4	(unused)
		3-0	command=1
	Word 7	15-0	message length

Tables are defined which contain appropriate data regarding each input and output stream.

For input the Synchronous Transmit Stream Table (STST) defines each stream:

STST entry 0 - console input
1 - card reader

STST entry format:

Word	Bits	Contents
0	15	Permission to transmit requested
	14-2	(unused)
	1	'SH running' message needed
	0	permission to transmit granted
1	15	(unused)
	14-0	Queue name
2	15-8	Queue depth limit
	7-0	Stream Escape Code
	15	(unused)
3	14-0	Queue name for producing handler

The Sychronous Receive Stream Table (SRST) defines each possible output stream:

SRST entry 0 - console output
1 - printer

SRST entry format:

Word	Bits	Contents
0	15	suspend console output pending a console transmission
	14-2	(unused)
	1	handler suspended because of q depth
	0	(unused)
1	15	(unused)
	14-0	Queue name for consuming handler
2	15-8	Queue depth limit (the maximum number of blocks which will be received prior to suspending the stream via FCS)
	7-0	Stream Escape Code
3	15-0	(unused)

Algorithm

Two local data structures are of special significance. These are the Synchronous Transmit Streams Table (STST) and the Synchronous Receive Streams Table (SRST). The entries in the STST define the possible input work queues. The entries in the SRST

define the possible output work queues. Contained in the tables are all parameters (flags, queue names, etc.) needed to use these queues plus all parameters needed to control the producers and consumers of these queues. Entries in these tables are used sequentially in a priority fashion. For example, the STST is ordered as follows: console input, then card input. Thus, the console will always have greatest priority. SRST entries are used in a similar manner.

To insure availability of communications buffers in order to prevent deadlocks and insure performance, four communications buffers are an assembled part of SH. These four buffers are interlocked to SH with the SH conrol queue name in BCW word 3. The buffers are managed within SH using a simple linked list for free buffers. All I/O with the CDC host is scheduled into and from these buffers. They are given to the data producers (console input/card input) and consumers (console output/printer output) by SH, and these handlers always return the buffers to SH. SH attempts to double buffer to the printer and from the reader but will always hold one of the buffers in reserve for console communications with the host. In this way, the operator is assured that he can intervene at any time.

The system initialization entry point to SH is SYNHI. The SH control queue name is found and saved in CTLQ. Finally, SYNHI "configures" SH by initializing the STST and SRST tables. This involves issuing a .FIND for all work queues for associated producers and consumers. The table entries are then initialized.

The prime entry to SH is SYNHP. SYNHP consists of three main sections. The first is the entry point section which removes work from the control queue and invokes processing of the work entries. Section two consists of a group of processing sections, each of which processes a control queue work entry. The third section is an I/O scheduler which is also used by the I/O complete entry point to SH. The I/O scheduler performs the task of examining the STST entries for transmission or other I/O scheduling.

A second prime entry point to SH is SINHP. This entry point receives control whenever any data is placed on either input stream work queue (console or card transmissions). The SH response is to enter the I/O scheduler for possible I/O initiation.

SYNHP is entered by a system prime. It then invokes processing of each work entry found on the control work queue. The routines which handle these work entries all return to a common point where the current work entry is released to its

buffer pool if appropriate. Other work queue entries are processed in turn until none remain. At this point the I/O scheduler is entered for possible I/O initiation.

These messages may appear on the SH control work queue:

LO, SC, SD, or ER commands - the command is received from the console I/O handler and is processed by routine SRJEC. The work entry is in a console buffer.

empty output buffer - this is a communications block which was previously received, has been disposed of via one of the output work queues, and is now being returned to SH by the associated consuming handler. Its function for SH is to signal a reduction in the number of blocks on that output work queue. SH uses this technique to monitor the flow of data and use of communications buffers. SQCHK processes these entries and then returns the buffer to the SH free buffer list.

free communications buffer - communications buffers released due to purge processing at system disconnect time are added to SH's free buffer list. SCOMB processes this work entry.

allocated console buffer - received from the buffer manager in belated response to some previous request for a console buffer. SCONB processes the entry.

allocated control buffer - received from the buffer manager in belated response to a previous request for a control buffer. SCTLB processes the entry.

Work entry processing by SYNHP is as follows:

SQCHK - locates the SRST for the associated output stream and determine the current Q depth. If the stream is not suspended, release the buffer. If the stream is suspended, decrement the count and use this buffer to signal "go ahead" to the host.

SCOMB - the buffer is added to the free buffer list.

SCTLB - if the system is not online, the buffer is released. Otherwise, it is supplied to SHRUN for the issuance of "SH running" messages to available producers of input work queue data.

SCONB - The buffer is supplied to .CCO. for console output.

SRJEC - for an ER command, the data is acquired and the messages are constructed and issued. For an LO command, the reader handler is purged and restarted (with "SH running") to effect a fresh card transmission.

For an SC command:

1. If SH is online or not online but engaged in going online, the 'SYSTEM ALREADY CONNECTED' message is issued.

2. If SH is not online and is not going online, the SC data station address and mode are processed. Then the signon activity signals are set, the 'SYSTEM READY' message is issued, and the data line enable I/O operation is started.

For an SD command:

1. If a signoff is already in progress, the command is ignored.

2. If SH is not online or engaged in a signon, the 'SYSTEM DISCONNECTED' message is issued.

3. Signoff conditions are established and a disconnect is forced immediately.

All I/O scheduling (except for the initial data link enable and terminal clear) is performed by SIO. SIO is entered by SYNHP when control work queue is exhausted. SIO is also used by SYNHC to initiate successive I/O operations. If I/O is active when SIO is entered, it exits to the central system commutator. Otherwise it performs the following sequence of steps:

1. If SH is no longer online, SPRGE is entered to clean up all queues, etc.

2. Now SIOSW is inspected to determine if the host is waiting on input. If not proceed to step 3. Else:

   A. inspect the input work queues (console first) for a data block to transmit. If none is found, then issue a read request.

   B. initiate transmission of the first available data block and assign new communications buffer to the

reader. If an end of file is indicated the STST transmit permission flag is cleared, then exit.

3. If no work queues are active and a signoff is pending, a disconnect is performed. Otherwise, a read is issued.

SYNHC is the I/O complete entry pcint to SH. It processes as follows:

1. I/O activity flags are cleared. If any abortive error has occured, SPRGE is entered to clean up.

2. If the operation completing is a disconnect, SPRGE is invoked.

3. If the operation is a successful connection, SH is placed online and the 'COMMUNICATIONS ESTABLISHED' message is issued. If a signoff has been requested, the disconnect section of SIO is invoked. Otherwise, the 'SH running' control signals are issued to all associated input handlers using SHRUN.

4. If it is a completed transmit/receive sequence, the request type is then examined.

   A. if the host is requesting input, set SIOSW and enter SIO.

   B. if the host has output data, locate the SRST entry and enque the data block to the appropriate handler's work queue. Then if q depth limit has been reached set the suspended flag and exit. If the maximum depth has not been reached, tell the host to go ahead by:

      a. transmitting a pending console message.

      or

      b. acquiring a communications buffer and initiating a transmit/receive operation.

   Then exit.

5. If the I/O event is an ALERT or an unsolicited attempt to write by the host, a communications buffer is acquired and a receive only operation is scheduled.

SHRUN is a subroutine used to present signals to associated input handlers informing them of SH activity and parameters. It searches for marked STST's requiring this signal, and transmits a message to the handler containing its intended input work queue name. SHRUN is invoked either by SYNHC following a signon or by SCTLB in belated response to the request for control buffers.

SPRGE purges all input and output work queues and associated handler queues. It resets SH to initial conditions, and issues the 'SYSTEM DISCONNECTED' message. Finally, it clears the purge indication from all SH work queues, closes the SH prime gate, and exits.

Card reader compression handler

Description and function

This handler (CRC) allocates and manages a card reader handler or other similar data source in order to pre-process the card data for the synchronous communications handler. Pre-processing involves blocking card images into the acceptable communications format. CRC is a level 2 function handler, and its interfacing to a card reader handler follows the level 2 - level 3 protocol.

Interfacing

CRC utilizes its allocated reader as follows:

1. A start buffer is placed on the reader's work queue to initially activate it or to reactivate it. The form of this buffer is as prescribed by the level 2 - level 3 protocol.

2. Allocated reader buffers are accepted from the card reader handler. They are set with a read command and returned to the card reader handler. These same buffers return as writes to CRC with useable data. They are always given back to the allocated reader as reads except when terminating, in which case they are freed.

3. Error commands from the allocated reader are used to report the error to the operator. For hardware error types (type 1 or type 2), a retry command is immediately sent to the allocated reader. For a type 3 data error, the retry is signalled by the operator's use of the RR command. This also then results in a retry command being sent to the reader.

4. CRC sends a stop command to the allocated reader when CRC is deactivating due to a TR command or synchronous handler disconnect.

CRC interfaces with the synchronous handler by:

1. Placing blocks of card images on the work queue indicated in the 'SH running' message.

2. End-of-file signals from the reader are indicated to the synchronous handler as prescribed by its base design.

Local data structures

The following symbolic names are associated with the local data structure used by CRC.

CCQNM holds the work queue name for the active function handler

CCFLG contains several flags:

CCSHR synchronous handler status (+=running, 0=not active, -=running but CRC is at queue limit)

CCPCR set = ok to process card data

CRDTA set = all resources available (communications buffer and not at queue limit)

CCEOF set = at end-of-file with not ready reader

CCTRC set = TR command is pending

CCRDR holds queue name of allocated reader

CCSHQ holds queue name to which compressed data is delivered

CCCCB holds address of communications buffer

CCCBP current position within communications buffer

CCQLM holds queue limit for maximum number of buffers to place on CCSHQ

CCRET holds return address for delayed returns from subroutine CGCB

No buffer definitions are forced by CRC. It conforms to those required by the synchronous handler and the level 2 - level 3 protocol.

Algorithm

CRCHI is the system initialization entry point to CRC. The only function performed is to .FIND the CRC work queue name. This name is stored in CCQNM.

CRCHP is the prime entry to CRC. Work queue entries processed include:

SH running - a signal from the synchronous handler that it is running. This message is received in a control buffer and informs CRC that it may initiate processing at any time. CRC will not do any work until this message is received. In the message are parameters needed to operate: name of the queue to receive data from CRC. CRC is designed to include SR command processing as an inherent part of the 'SH running' message processing.

SE, TR, and RR commands - these messages are received from the console I/O handler as operator requests to activate and control use of a reader. These messages are not valid unless preceded by an 'SH running' message.

Reader data - card images from the reader allocated by CRC are blocked and placed into the communications buffer. Error indications from the reader are also processed.

Purged queue - the CRC work queue can be purged by the synchronous handler due to abortive errors or normal termination procedures. In any case, CRC deactivates its allocated reader, deallocates same, and quiesces until the next 'SH running' message is received. A stop command sent to the reader as a part of purge processing will also return to CRC later. Buffers returned by the allocated reader at this time are released.

Control, console, and communications buffers - these may be received from the buffer manager in belated response to requests for buffers.

Allocation and management of the card reader handler is affected by three sources: activity signals from the synchronous handler, operator directives and device status signals from the card reader handler itself. The synchronous handler informs CRC (as well as other modules) that it is functioning by placing an 'SH running' work entry on the CRC work queue. Only when this signal has been processed can other signals be effective. Similarly, a purged CRC work queue (purging done by the

synchronous handler) indicates that the synchronous handler is ceasing to function.

Between the "running" and "purged" signals, operator directives (Start Reader, Restart Reader, and Terminate Reader) are accepted. Note that the synchronous handler connection signal internally causes an SR command to be processed. A purge signal also implies a TR command. The SR command causes an allocation of a card reader handler to be attempted. A message indicating the result is issued in either case. If successful, initial conditions are established and a start buffer is acquired. This buffer is put on the card reader handler work queue with ASCII reading mode and the CRC name indicated. This will activate the card reader handler as described in its base design. An RR command is routed to the allocated reader for actual handling. The TR command inspects a CRC activity indicator and either deallocates the associated handler immediately if no cards are being processed or flags the TR condition to be effected at end-of-file.

Once allocated and started, the received card images are added to the communications buffer after being compressed using the .COMP module and put on the associated synchronous handler work queue. Two problems can interrupt this flow. When a communications buffer is not available, an indicator of this condition is set. If a card image appears on the CRC work queue prior to the needed communications buffer, that card image is replaced on the CRC queue (via .PRIQ) and the CRC handler exits after closing its prime gate. This prevents further processing of card images until the needed communications buffer becomes available. Any errors indicated by the card reader handler to CRC will result in the issuing of an error message. An end-of-file signal is propogated through to the synchronous handler. Following the processing of an end-of-file, the TR flag is inspected and appropriate action is taken. Read buffers are always returned to the allocated reader immediately following processing of the data in them. This technique insures that a read operation is continually being attempted and is the way in which the "hot reader" feature is implemented.

Printer Decompression Handler

Description and function

The printer decompression handler (PD) has as its main responsibility the transformation of compressed data blocks from the synchronous handler to print lines which are passed to the appropriate level 3 handler. It is also responsible for performance of the following control functions: Accept printer allocation requests and perform the desired allocation, printing the RJE AWAITING PRINTER1 message if necessary. Accept disconnect requests and force printer disconnection. Accept printer error notices, produce ATTENTION REQUIRED messages, and notify the synchronous handler of the printer error. Accept device end-of-file notices and release the device. This handler does not have an I/O Complete section, since it does not issue IOC calls. The module is coded in a "serially reusable" fashion, with all locally used data accessed from a unique storage block. Thus, additional printer streams may be added by replicating this storage block.

Interfacing

PD interfaces with the level 3 handler as follows:

1. A start command is placed on the level 3 handler's work queue to activate it.

2. Allocated print buffers and emptied print buffers are accepted from the level 3 handler. These are filled with decompressed data and given to the level 3 handler. At termination they are freed by PD.

3. Error message buffers from the level 3 handler are used to report error conditions to the operator.

4. A stop command is sent to the level 3 handler at end-of-file or synchronous handler disconnect.

5. A command type 9 is sent to the level 3 handler to initiate timed retries when the printer is not ready.

PD interfaces with the synchronous handler by accepting blocks of compressed data for the printer or punch.

No BCW definitions are made by this handler. It conforms to those defined by the synchronous handler and printer handler.

Algorithm

The detailed processing algorithm is given in flowchart form. A summary of the algorithm is given below.

A significant feature of the inputs to the routine is that they include a mixture of normal messages entered by .PUTQ calls (print blocks, purge queue request) and priority messages entered by .PRIQ calls (printer error, printer EOF, and buffer messages). As a result, upon entry to the handler the queue must be inspected to detect priority calls. Hence, a .SEEQ call is used to create a latch to different message processors. These message processors are discussed below. When a processor has finished it will execute a .GETQ to remove its entry and return to the .SEEQ latch or .COM., depending on the type of processing performed.

DISCO is entered for purge-queue requests, which only result from a communications disconnect. If a device is allocated, the routine deallocates the device, issues a stop command to the level 3 handler, releases any unused buffers, and resets local storage.

ALLOC is entered for printer allocation requests. If the printer is already allocated to this handler, the routine waits until the printer is deallocated before it attempts another allocation. Otherwise the routine attempts to allocate the device, sends an "AWAITING PRINTER" message to the console, if necessary, and sets a timer loop to retry allocation. A pathological case of special interest is when a buffer cannot be obtained for the console message. In this case the entry latch for the routine is reset to allow no further actions until the buffer is received as a message. This technique is used elsewhere in the handler, but is not further elaborated in this document.

PRBLK is entered for print block queue entries. If a print buffer is received and a printer has not yet been allocated, ALLOC is entered to perform the allocation. If a printer has been allocated, a buffer is obtained from the free list and entry is made to .DCOM to decompress lines for output. The routine also sets carriage control, if necessary, and then ships lines to the level 3 handler. It is also responsible for indicating print block processed to the Synchronous Handler when finished. Again, suspension may occur due to unavailability of a buffer, and the latch-restart technique is used to continue processing.

PRFLT is entered for printer faults. When the printer is not ready, PRERR issues an "ATTENTION REQUIRED" message to the console. Again, a latch-restart for buffer not available suspension is used.

PREOF is entered when the stop command is returned by the level 3 handler. All print or punch buffers are freed by PD and the device is de-allocated.

PRBUF is entered for buffer messages. It stores the buffer message to allow proper operation of the latch-restart routines and then goes to .SEEQ to allow continuation of the suspended operation.

The only other entry to this handler is the initialization entry. This simply calls .FIND to find its own queue name for future reference.

Submodule descriptions

PDCC is the functional module in the handler which receives control after a printer buffer is obtained and determines the carriage control character to insert in the buffer. Print-then-space mode is always utilized by this handler, so control is always executed before printing.

.COMP - compression service

Description and function

.COMP prepares buffers of card images for transmission under the CDC U200 protocol.

Interfacing

The routine is entered via a JSB with the parameters in registers and following the call. It returns a value in a register. The calling sequence is as follows:

```
LDA   <from addr>
LDB   <from length>
JSB   .COMP
DEF   <to addr>
```

<from addr> is the byte address of the data to be added to the buffer.
<from length> is the number of bytes of data in the record to be added to the buffer.
<to addr> is the byte address of the destination buffer. This address must point to the first available location in the buffer.

Upon return, the B-register will contain the byte address of the next available location in the destination buffer. Compression consists primarily of moving the card image to the destination buffer. The occurance of 200 User Terminal end-of-record or end-of-file codes in the first byte of the source card image are encoded with the proper preceding escape code before moving the card image to the buffer.

.DCOM - decompression service

Description and function

.DCOM provides a data decompression service to expand CDC compressed records into uncompressed format for unit-record devices.

Interfacing

The routine is entered via a JSB with calling parameters in registers and following the call. It will return two values in registers. The calling sequence is:

```
LDA   <from addr>
LDB   <to addr>
JSB   .DCOM
DEC   <buffer length>
```

<from addr> is the byte address of the data to be decompressed. This must be the address of the first character following the format control character

<to addr> is the byte address of the destination of the decompressed data.

<buffer length> is the positive byte length of the buffer to receive the decompressed record.

Upon return the registers will contain the following:

A-register: Length of decompressed record (Ø if EOF record).

B-register: Pointer to next format control character in source buffer.

Note: If there is more data in the compressed record than is available in the buffer, excess data will be truncated. The length returned will never be greater than <buffer length>.

NOTE: This document assumes knowledge of the HP2000 ACCESS Queued IOC module and Control Data Corporation's (CDC) telecommunications protocol used with the 200 User Terminal (200UT). The following CDC manuals are recommended for additional information on the protocol. These manuals describe in detail the actual hardware emulated by this software.

200 User Terminal Hardware Reference Manual (Pub. No. 82128000)
217-2 Equipment Controller (Pub. No. 82128100)

I. Product Identification

Product Abstract

The D50CD driver interfaces 2100 series computers to a remote Control Data Corporation CPU via a dial-up or dedicated communications line. The line discipline is bit serial synchronous and the link protocol is that employed by the CDC 200 User Terminal. The driver assumes the use of the 12618 synchronous interface kit.

II. Design Overview

Design assumptions

The program requires the microcode designed for use in the HP2000 ACCESS I/O Processor. The driver also requires the D.43 time base generator driver.

NOTE: In order to avoid loss of characters on input/output operations, the interface boards should be placed at a relatively high interrupt priority. This does not mean the driver uses a large portion of available CPU time, indeed only a few percent of CPU cycles are taken by D50CD. It does mean that the interrupts do need to be processed quickly.

Design summary

D50CD is an input/output driver designed to operate in the HP2000 ACCESS I/O Processor. The following functions will be performed by D50CD.

1. CLEAR — resets all status words. Puts I/O board in "on-hook" condition.

2. READ — Accept a block of data from the host system. Error detection and reporting to the host are automatically handled.

3. WRITE — Send a block of data to the host. As in the READ operation, error detection and reporting to the host are automatically handled.

4. ENABLE — Allows the communications line to be established by answering the phone or by operator manual dial. Certain operating parameters are passed to D50CD via the ENABLE call.

5. EXTENDED STATUS - Accumulates statistics on device performance during a terminal session.

NOTE: The WRITE operation may be either a straight write to the host or a write/read; in the case of a write/read operation, after data is written to the host, the buffer is used to receive data from the host.

Design approach

The driver is composed of two fundamental units; the initiator section and the continuator section. Each of these two sections is further modularized to handle the various driver functions. Each module performs some specific function ("read" or "write") or common "service functions" (code conversion, establish synchronization, send character, receive character, etc.).

The initiator section sets up service requests for the continuator section. The requests are for either input (read), output (write), or output/input (write/read). The continuator section receives messages from the host and when possible, satisfies the requests made from the initiator section. It must be noted that the driver is responsible for maintaining the line protocol. It is the responsibility of the calling program to ensure the proper message content and sequence; D50CD has no knowledge of the message content.

When a request is made from the calling program to the initiator section of D50CD, a check is made to see if there is a pending request. If so, the call is rejected. If there is no pending request, the request is passed to the continuator section for action. (Passing the request consists of setting flags in a common data structure. Obviously, this must be done in a disabled state.)

The continuator section of D50CD monitors data source (host system) messages and polls and satisfies the initiator section request appropriately. In the case of an input operation (read), the continuator acknowledges any host write operation and fills

the caller's buffer. In the case of an output (write) operation, the data is sent to the host on the next POLL. In all cases where there is no input buffer, host write messages are rejected. Likewise, when there is no output buffer, host POLLs are rejected.

For a complete understanding of D50CD, the operation of the continuator section of the driver must be understood. It is here that most of the emulation of the 200 User Terminal communications protocol is done. The operation of the continuator or interrupt handler portion of D50CD is completely independent of almost all other activities in the system. It is totally interrupt driven and communicates with the host system all by itself, even when no I/O has been requested by the calling program. This interrupt handler switches back and forth between transmissions and receptions, controlling both interface boards as required.

All activities begin at the label INPUT where receive operations are initiated. A real 200 User Terminal is normally in a receive mode, waiting for the host to indicate the next desired action. That is, the protocol is a polling/selection type of protocol, and, therefore, the terminal (driver in this case) must be directed through each action by the host. During any of the actions being performed asynchronously by this interrupt handler, the initiator section of D50CD may cause various buffer addresses and lengths, etc. to be set. These will be noted by the interrupt handler for action at appropriate times.

When a receive operation from the host begins, the common message header is first analyzed character by character. If the incoming message contains a site address other than the one designated in the ENABLE operation, the reception is ignored. This would make multidrop operation possible. Normally, however, the message is accepted, and the message control code will be detected. This causes dispatching of a routine designed to handle each incoming message type.

POLL -- If the initiator section has scheduled a read operation, then it is out of phase with the host. Such a read is terminated with EOT status of 1. (This feature of the driver allows the calling program to attempt to anticipate host input at times when no other I/O is indicated.) If the initiator section has scheduled a write operation, the correct message header is transmitted to the host, followed by the data supplied by the caller. Finally, the trailing ETX and message parity codes are transmitted. If the calling program has indicated a write/read operation, the read is now enabled.

WRITE -- Diagnostic writes from the host are not supported by this driver. All other forms of write messages are processed identically. If the calling program has not scheduled a read operation, the incoming message is still scanned to determine the associated "E" code. This is then passed to the upper level program in the form of an unsolicited event. Then the host is responded to with a REJECT message. However, if a read operation is active, the incoming data is buffered and checked. If no errors occur, a completed I/O operation is returned via .BUFR. Then the host is sent an ACK message. The detection of any error results in the transmission of an ERROR message to the host.

ALERT -- If a calling program read operation is active, the host is sent an ACK message. If no read operation is active, the upper level program is shown the occurrence of the ALERT with an unsolicited message.

Code conversions for 200UT emulator

The 200UT emulator operates internally in ASCII mode. All buffers transferred between the driver and the caller are in ASCII. The central system may use BCD (internal or external) or ASCII). The former requires code conversions be done on received and transmitted data.

Received data (INTERNAL or EXTERNAL BCD) is translated via translate tables (INTAS, EXTAS) to ASCII.

Transmitted data is translated from ASCII via translate tables (ASINT, ASEXT) to Internal or External BCD.

In addition, conversion of inbound codes following an escape code are converted to ASCII using tables IASES and EASES if a non-ASCII code is in use.

III. Design Structures

Major Functional Units

CLEAR

Description

Puts the I/O board in "on hook" condition (drops data terminal ready). Clears all status words except "extended status".

Interface Description

```
JSB   .IOC.
OCT   0000XX  where XX is logical unit number
```

No rejects are possible.

Algorithm Description

1. Turn off data terminal ready.
2. Clear the driver and reset to idle state.
3. Clear status word in EQT.
4. Disable I/O interface.

NOTE: CLEAR is issued to terminate all activity on the communication link. The CLEAR should only be issued after all operations are complete (no outstanding reads or writes).

READ

Description and Function

Read a block of data from the host system. Execute error detection on the incoming data block.

The following conditions will result in an error message being sent to the host system.

1. Nonexistent Station
2. Station address of 140 or 160 in any message other than POLL.
3. Unrecognized message control code.
4. No E code or erroneous E code in message.
5. Word parity error (in any character).
6. Message parity error.
7. Carrier dropped before EOT or LRC character received.

Interface Specification

```
JSB .IOC.
OCT 0100XX              where XX is logical unit number
Reject Address
Buffer Address
Buffer Length           (positive bytes)
```

The call may be rejected for the following reasons:

1. Terminal "off-line" (E value not complete)
2. Driver is busy.

Notes:

a) The transmission log will reflect the number of characters received.

b) The transmission log includes the control code after the site address through the ETX character.

c) A buffer overrun will be treated as a carrier loss error.

d) If an operation other than WRITE or ALERT is received from the host, the read operation is terminated with a status of 1 in the D50CD EQT. This indicates that the read operation is inappropriate for host activity and allows the caller to alter its I/O operations. Thus, when no other I/O can be scheduled, a read operation is always safe.

Algorithm Description

1. Await a host POLL operation.

2. Receive and buffer the data from the host, generating message parity on a per character basis.

3. Check LRC on receiving message parity character. If there is a parity error or any other error mentioned above, send an error message to the host.

Read is the operation issued in response to a host system WRITE message or ALERT message.

WRITE

Description and Function

Sends a block of data to the host system.

Interface Description

```
JSB    .IOC.
OCT    0200XX          where XX is the logical unit number
Reject Address
Address of Buffer
Length of Buffer
```

The call may be rejected for the following reasons:

1. Terminal "off-line"
2. Driver busy

NOTE: The caller must supply only data in the buffer through the associated "E" code. The driver will supply the message prefix and trailing EIX code.

WRITE/READ

Description and Function

As the WRITE operation except the write buffer is used for a read operation at the termination of the WRITE.

Interface Description

```
JSB    .IOC.
CCT    0210XX              where XX is the logical unit number
Reject address
Write Buffer address
Length of write buffer
```

The call may be rejected because the driver is busy or the terminal is "off-line". The read length is always assumed to be 1040 characters. The read buffer is the same as the write buffer. Therefore, any incoming data overlays the original write data.

Algorithm Description

The write will send data to the host in response to a POLL from the host. After all data is sent a read is performed. The read accepts data from an incoming host WRITE message.

NOTE: The driver does not support write diagnostic operations.

ENABLE

Description and Function

Establishes the communication line for the terminal. Allows either manual connections or auto answer connection.

Interface Description

```
JSB   .IOC.
OCT   0220XX      where XX is the logical unit number
Reject address
Buffer address
Buffer length
```

Buffer:

```
Word #1:    BIT 0 = 1   Operate in ASCII mode
            BIT 1 = 1   Operate in Internal BCD mode
            BIT 2 = 1   Operate in External BCD mode
            BIT 8 - 15  SITE ADDRESS
```

The call can be rejected for the following reasons:

1. The terminal is already "on-line".
2. The driver is busy.

Algorithm Description

1. Prime the I/O board.
2. Wait for carrier from modem.
3. Enable host receive operations.

EXTENDED_STATUS

Description_and_Function

The use of this call is to pass statistics on the interface's performance during a terminal session.

```
JSB   .IOC.
OCT   0120XX          where XX is the logical unit number
Reject address
Buffer address
DEC   6
```

After this call statistics are kept in the buffer addressed by the word in the calling parameter list.

The call should be made before the ENABLE call is issued. The caller will never be rejected.

Algorithm_Description

1. The following statistics are maintained in the storage addressed by the requestor:

WORD 1	erroneous station addresses
WORD 2	erroneous control codes
WORD 3	character parity errors
WORD 4	message parity errors
WORD 5	carrier losses
WORD 6	missing write control codes
WORD 7	code indicating type of line drop (0=clear request, 1=loss of data set ready condition, 2=failure to receive any data from host for 2 minute period)

(see the project ERS for amplification).

Notes: Whenever a call to D50CD is rejected the following return codes will appear in the "A" and "B" registers:

A	B	Reason for Reject
1	0	Illegal request
1	1	Terminal is offline
1	100000	Driver is busy

Unsolicited Events

The following unsolicited events are defined for D50CD.

ALERT received	A=logical unit number B=000001
WRITE received	A=logical unit number B=000002 (also, bits 15:8 contain the write message "E" code)
LINE DROP OCCURED	A=logical unit number B=000000

In the case of an unsolicited write operation the driver will reject writes (station busy) until a read operation is issued.

The driver is completely responsible for line quality monitoring. The monitoring is accomplished by the alternating station addresses in messages received from the host.

All data passed to D50CD is assumed to be ASCII. All data passed to the caller is also ASCII. Code conversions are done by D50CD from/to ASCII and INT/EXT BCD.

Summary of D50CD IOC calls

FUNCTION	SUBFUNCTION	OPERATION	RETURN
00	00	CLEAR	Immediate
01	00	READ	Asynchronous
01	20	EXTENDED STATUS	Immediate
02	00	WRITE	Asynchronous
02	10	WRITE/READ	Asynchronous
02	20	ENABLE	Asynchronous

SECTION VI

PROBLEM ISOLATION

PROBLEM ISOLATION

Unfortunately, there is no one way of localizing any problems which might occur in the IOP. However, tracks do exist which could help you in localizing the bug or possibly solving it. Outlined in the following paragraphs are essentially those tracks which you can check to, if nothing else, eliminate certain modules as suspects.

1) Looking at the entry point of .GETQ
This will give you the handler who last got something from his work queue.

2) In addition, entry points of .PUTQ, .PRIQ, .FREB and .GETB give you the same type of information.

3) Due to the nature of power fail recovery, the I.XX entry points of all drivers will be cleared. Hence, if a HLT 2 should occur and the I.XX entry point of a driver is non-zero, then he was the last one in control and most likely the cause of the problem.

   There is one exception to this rule. The interconnect kit driver D.61 contains the code to process a cold dump request. As a result, if a cold dump is taken, the I.61 entry point and some DMA related information get wiped by the dumping process. Since this data is recognized as valuable, the dumping routine saves it before executing. The information can be found within D.61 as follows:

KSNI.	contents of I.61
KSNI.+1	DMA routine return address
KSNI.+2	DMA buffer address
KSNI.+3	DMA buffer length

4) There exists a trace table of SP to IOP communications, (entry point D.61S in D.61). This will give you the last 50 ICK transmissions. You cannot conclude that the last thing received from the SP was the culprit, remember many things go on asynchronously. Its just another source of information on whats happening in the system at the time of the crash.

5) The entry point of .IOC. will show who was the last to cross the boundary from system (handler level) to I/O (driver level).

6) QIT entries will show you what's pending on the work queues of all the handlers. what is on the work queue can almost be eliminated from the cause of the problem by the fact that the processing they invoked has not yet taken place.

7) The CRQ will show you what I/O events have completed but have not yet been processed.

8) EOT entries will yield information about all DRQ's: whether a device is suspended, whether the last event for that device completed, and finally, the hardware status.

9) QTOP in D.43 points to those events which are being timed.

10) Buffer pending queues will show if there is a bottleneck in the returning of buffers. If the count for each handler becomes large, then someone is hogging the buffers and not returning them. This "hogging" should never occur.

11) The main trace table traces completed events (both solicited and unsolicited) and all calls to .PRIQ and .PUTQ. This table has room for fifty entries (100 words). The trace routine is found in the module MN. TTPTR will be pointing to the next slot in which an event will be traced. The previous slot will contain the last event put on the trace table. Events are traced as follows:

    a) .PUTQ/.PRIQ: A reg - address of QIT to receive buffer
                    B reg - address of the buffer

    b) Completed events:
       solicited - A reg - original I/O call parameter
                   B reg - address of the buffer
       unsolicited - A reg - function plus unit reference #
                     B reg - event data driver generated

NOTES:

    a) .PUTQ and .PRIQ call the trace routine immediately after they are called. This means the trace entry is made when the PUTQ/PRIQ happens. The COMPLT processor calls the trace routine to trace completed I/O. The COMPLT processor is jumped to from the commutator when an IOC interrogate request has revealed completed I/O for some handler to process. This means that the trace entry is made after the event has completed at the driver level and after the event has been removed from the CRQ, but immediately before the HC entry point of the related handler is called.

b) Decoding the main trace table sometimes appears a fine art to the uniniitiated. Using the following algorithm will help.

1. LOOK ONLY AT THE A REGISTER

2. Compare the A register contents to the QIT addresses. If the A register points to the beginning of a QIT entry, this is a .PUTQ/.PRIQ to the handler represented by that QIT entry.

3. If it is not a .PUTQ/.PRIQ, then it should be a completed I/O event. The last six bits of the A register give you the IOC unit reference number for the device. The Equipment Table (EQT) is in unit reference number order (first entry is unit reference # 7). Use the unit reference # to index into the EQT and find the select code of the device. This will usually tell you what device it is. If you don't know the configuration, go to the Device Table. It contains both the select code and designator for each device configured in the IOP. Now that you know the device, go to the documentation for the driver of same and find out what kind of IOC requests/unsolicited events it handles.
Some IOC requests will be obvious. Bit 15 tells if it is a queued request or not. Bits 14-12 give the function. The most common functions are read (1) and write (2). Some examples are:

```
010010 non-queued read request - unit ref # 7
020011 non-queued write request - unit ref # 11
020210 non-queued write request - unit ref # 10
012011 non-queued read request - unit ref # 11
110015 queued read request - unit ref # 15
120015 queued write request - unit ret # 15
```

If it's not some type of read or write, it's most likely an unsolicited event. The B register contains the relevant unsolicited event data. See the documentation on the driver presenting the unsolicited event to decode the event data.

12) The interrupt trace table is another trace of past IOP activity . .TINT (found in module MN) is called from every I.XX entry point. The address of the I.XX entry point and the contents of that location (ie. what module got interrupted) are recorded. The interrupt trace table has room for 50 entries (100 words). TIPTR will be pointing to the next slot in which an enterrupt will be traced. The previous slot will be the last interruot that occured.

13) A list of addresses and other paramaters is generated by IOPC and loaded into the IOP at configuration time. The address of this list is moved to location 3 of the IOP at system start-up. If any IOP module detects a serious problem, a system DEATH routine is called which saves the environment in this same list. Similarly, if the operator executes the IOP PANIC loop and the environment has not already been saved, it is saved as above. This list exists primarily for the cold dump analyzer's use, but is very helpful if no analyzer is available.

COLD DUMP ANALYZER LIST

WORD	CONTENTS
0	IOP date code
1	A register
2	B register
3	Flags - bit 2 = int sys status bit 1 = E register bit 0 = O register
4	F register (register save stack pointer)
5	p register = 0 info saved when PANIC executed # 0 info saved when DEATH executed and this is address from which it was called
6	DEF .DAT address of device assignment tabl
7	DEF .DVTB address of device table
8	DEF QITAD address of QIT's
9	DEF PULPT addrsss of SCOL list
10	DEF QIOP address of D.43 TQE chain head
11	DEF LUT51 address of D.51 unit tables
12	DEF LUT61 address of D.61 unit tables
13	DEF UCT00 address of D.63 unit tables
14	DEF SIB1K address of 1CKH unit tables
15	DEF SIBAF address of ASFH unit tables
16	DEF D.61S address of D.61 trace pointer
17	DEF TTPTR address of main trace pointer
18	DEF CRQ address of .IOC. CRQ head
19	DEF QHEAD address of .IOC. free storage
20	DEF STST address of SYN(IB/CD) tables
21	DEF PNTR address of MEMRY free list
22	DEF EQT address of equipment table
23	Copy of IOPC's .MOPT word
24	Address of first buffer in pool 1
25	Address of last bufer in pool 1
26	Address of first buffer in pool 2
27	Address of last buffer in pool 2

If a cold dump analyzer is available, it will use the above list to print out almost all the tables and traces in the IOP in a neatly formatted listing. The interrupt trace table is not broken out by the analyzer. The other two trace tables are, with the most recent event printed at the top of the table.

GET COLD DUMP ANALYZER

SECTION VII

IOPC

I. Product Identification

I/O Processor Configurator (IOPC)

The purpose of this program is to configure I/O Processor (IOP) programs for the HP2000 ACCESS systems. Since the IOP program can contain various components and features, a technique is required to selectively load and merge needed modules to make up the IOP system desired by the user. Some of the possible features which may be configured are: number of TSB ports, IOP memory size, 2741 support, ASCII files support (line printers, card readers, tape punches, etc.), and RJE access to various host systems.

A secondary purpose of IOPC is to perform loading (reloading) of the IOP.

II. Design Overview

Design assumptions

IOPC is not a stand alone program, although it is written in absolute form. It is one of several programs which are used to configure complete HP2000 ACCESS systems (the others being the 7900 loader, 2883 loader, etc.). The MCP or Master Configurator Program controls the loading and invocation of the various configuration routines. The MCP contains I/O drivers for the system console and magnetic tape units. IOPC utilizes these drivers and several other parameters and subroutines in the MCP. One special capability of the MCP is the application of "patches" or "fixes" to the program binarys which it reads. While this is transparent to IOPC, it does allow the application of such patches to the IOP program which IOPC creates.

The entire HP2000 ACCESS system is distributed on a master magnetic tape. File one of the tape is the MCP. A small bootstrap program is used to load and initiate the MCP. The MCP then can load IOPC. Each master tape file after the first (MCP) is identified by a file id record. Within the file, groups of binary records are identified by group id records. The entire file is terminated by a file end id record. The purpose of these id records is to allow MCP to identify files for tape positioning and to identify binary records for the application of patches. IOPC is only concerned with the file id records for the files which it uses. One file is the standard modules file. The second is the optional modules file. Third is the IOP core image file. A complete outline of the files and file id records assumed by

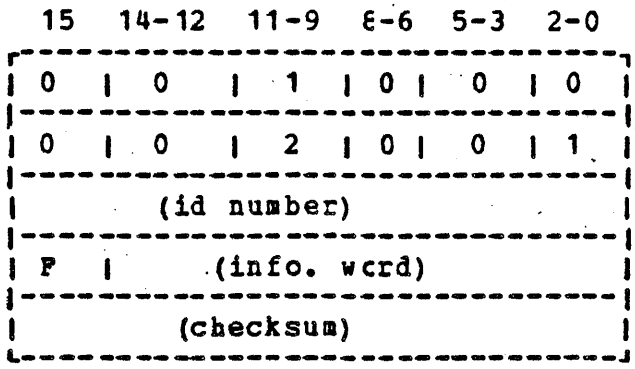
IOPC may be found in Table 1. Further details on the use of these files will follow.

Other design assumptions for IOPC are:

1. Select code 11 is assumed to be the transmit channel interconnect kit interface to the IOP.
2. Select code 12 is assumed to be the system console.

Table 1

ID records - An ID record is exactly 5 words long and has the follcwing form:



The ID number is an unsigned 16 bit positive integer ($0 \leq n \leq 65535$). File ID numbers must be a multiple of 1000. The information word does not apply to ID records appearing on the master tape. This word is used on patch tapes only (see MCP). The "F" bit designates a relocatable (set) or absolute (clear) group of binary records. An ID record can be assembled as follows:

```
ORG 2001B
ABS IDVAL
DEC 0
BSS 1
```

Master tape organization - At the time of this document, the master tape had the following file organization:

File	Contents	File ID	Format
1	MCP	none	absolute
2	IOPC	1000	absolute
3	Std. IOP modules	2000	relocatable
4	Opt. IOP modules	3000	relocatable
5	7900 Loader	10000	absolute
6	2883 Loader	11000	absolute
7	7905 Loader	12000	absolute
8	TSB System	20000	absolute
9	IOP copy file	65000	absolute

IOPC files - At the time of this document, the master tape organization of IOPC related files was:

File ID	Group ID	Contents
1000	1010-?	IOPC program

```
2000        2010        MN
            2020        DUMP
            2030        TBGH
            2040        ICKH
            2050        MUXH
            2060        MEMRY
            2070        IOC
            2080        D43
            2090        D61
3000        3010        D51B
            3020        D51A
            3030        D63
            3040        D62
            3050        D50IB
            3060        D50CD
            3070        D110
            3080        D120
            3090        D130
            3100        D140
            3110        D340
            3120        D11C
            3130        D12C
            3140        D13C
            3150        D14C
            3160        D34C
            3170        .2741
            3180        .274E
            3190        .274C
            3200        MNRIB
            3210        MNRCD
            3220        WCSL
            3230        D04
            3240        ASFH
            3250        SYNIB
            3260        SYNCD
            3270        CIO
            3280        CICIB
            3290        CICCD
            3300        CODIB
            3310        CODCD
            3320        CC0IB
            3330        CC0CD
            3340        PD0IB
            3350        PD0CD
            3360        PP0
            3370        CR0
            3380        PR0
            3390        RP0
```

3400	LP0
3410	PU0IB
3420	CCCIB
3430	CCCCD
3440	PDCIB
3450	PDCCD
3460	CRC
3470	LPC
3480	PPC
3490	RPC

IOPC assumptions - The order of the binary groups in the standard and optional modules files (IDs 2000 and 3000) is not random. Modules in the standard modules file are always loaded by IOPC. This is true even if the module can not be identified by IOPC (see .NAMT). Further, it is required that the first module in the standard modules file be the MN or managers module. This is to insure its loading at location 2000 which, in turn, assures the documented start addresses for PANIC (2000) and the IOP to IOPC unloader (2002).

Not all needed modules are in the standard modules file. Even a minimum system requires loading of the appropriate D.51 module and D.04 module from the optional modules file. Generally, the modules in the optional modules file are ones which are variant in some way. Some details about this file are:

1. D.04 must follow all driver modules.
2. Variant RJE modules are assumed to have names (NAM records) with the last two characters indicating the version (e.g. IB for IBM, CD for CDC, etc.)

Design approach

IOPC has three distinct processing sections. First is the system configuration dialogue. During this section all questions about configuration are asked. The operator's responses are analyzed. The outcome is the establishment of various configuration parameters and control blocks which will determine further processing.

The second section of IOPC is essentially a relocating loader. Modules are selected from the master tape as required. These are relocated and linked together to create the IOP program. In addition, a number of binary module modification processes are also involved. The purpose of these processes is to contruct needed IOP control blocks, buffers, etc. Also, such things as the merging of select codes into I/O instructions can be done. A final function of section twc is to create an optional ccpy of the configured IOP program.

Section three of IOPC produces a memory map and report of the configured IOP program.

III. Design Stucture

Data structures

Several control blocks and parameter lists are used during the configuration process. These are outlined here.

Master Configuration Table (.MCT.)

This table, located cn base page, contains most of the configuration parameters. While it does contain some default values, most of this table is filled in as a result of operator specifications during the system configuration dialogue section.

Word(s)	Name	Content
0	.MEMS	highest memory address in IOP
1	.MPN	zero relative number of ports (default is 31)
2-33	.MPBS	TTY buffer size for each port in words (default is 60)
34	.MOPT	options flags:

Bit	Meaning
15	2741 EBCD code
14	2741 CALL 360 code
12	IBM RJE

		11 CDC RJE 1 create master tape IOP copy 0 create separate tape IOP copy
35	.MSCM	highest used select code in IOP
36	.MHRD	number of host readers (default is 1)
37-43	-----	device assignments for host readers
44	.MHLD	number of host line printers (default is 1)
45-51	-----	device assignments for host line printers
52	.MHPD	number of host punches (default is 0)
53-59	-----	device assignments for host punches
60	.MACR	number of card readers
61	.MALP	number of line printers
62	.MAPU	number of tape punches
63	.MARP	number of reader/punch/interpreter
64	.MAPR	number of photo-readers
65	.MCNT	number of IOP modules to be loaded
66	.JT#	number of job transmitters
67	.JL#	number of job transmitters
68	.JP#	number of job transmitters
69	.MAFN	negative number of ASCII files
70-?	.MASF	3-word ASCII file apecifications: Word Content 0 device designator 1 15:12 characteristics 7:6 subtype 5:0 select code 2 record size in words

Memory Allocation Table

This table keeps track of IOP memory as it is allocated.

Word	Name	Content
0	FWBP	holds address of 1st word of base page links
1	FWABP	holds address of 1st word of available base page
2	LWABP	holds address of last word of available base page
3	FWAM	holds address of 1st word of available program memory
4	LWAM	holds address of last word of available program memory

ASCII Files Dialogue Control Table (.ACT.)

This table controls the system configuration dialogue for ASCII files. It also provides basic parameters for the

establishment of the .MASF entries which are generated from the ASCII files dialogue. Each eight word entry in .ACT. is as follows:

Word	Contents
0	Device type name (2 ASCII characters)
1	Basic device designator
2	Minimum number of this device allowed
3	Maximum number of this device allowed
4	Default record size in words or a negative value
5	Maximum record size or address of device subtype table if word 4 is negative (see note below)
6	Device characteristics and flags: Bit 15 = 1 if input device Bit 14 = 1 if output device Bit 12 = 1 if only output allowed is CTL. Bit 7 = 1 if RJE pseudo device (causes this entry to be bypassed in no RJE. Also prevents request for select code.) Bit 6 = 1 if SP device Bits 5-0 contain select code for SP devices and number of associated select codes if IOP device.
7	Address of list of addresses of words to receive number of these devices to be configured (list is terminated by minus one value)
8	Address of word to receive address of first .MASF entry corresponding to this type of device

Note: for devices which come in several flavors, word 5 addresses a subtype table. This is indicated by a negative word 4. In this case, word 4 represents the maximum subtype expressed negatively. All subtypes are zero relative. A subtype table is as fcllows:

Word	Contents
0	Default (and maximum) record size for subtype 0
1	Same for subtype 1
2	Etc.

New types of ASCII files can be included by simply adding new entries to this table. A negative one word terminates the table. The order of the table must be as follows: all system processor devices, all RJE pseudo devices, all IOP devices.

Module Loading Table (.NAMT)

This table contains all information required to make load/noload decisions about modules encountered in the binary

input files. In addition, other information is present to be used in configuring various tables (EQTs, etc.). The order of .NAMT entries for drivers and handlers indicates relative IOC and TSB logical unit numbers which must be derived from this order and other information such as that in .MCT. Entries in .NAMT for standard file modules must contain a copy count of 1. Other copy counts are established during the initial dialogue. The format of .NAMT is as follows:

Word(s)	Contents
0-2	Module name
3	Module characteristics: Bit 0 = 1 if module is handler Bit 1 = 1 if module is driver Bit 2 = 1 if module is varying RJE module Bit 3 = 1 if module is RJE module Bit 4 = 1 if module is ASCII file module For handlers: Bit 15 = 1 if module has prime entry point Bit 14 = 1 if module has I/O complete entry point Bit 13 = 1 if module has initialization entry point Bit 12 = 1 if module represents allocatable resource For drivers: Bits 15-14 are EQT characteristics
4	Number of copies to load
5	Designator name (if handler) Driver number in bits 7:0 and number of associated select codes in bits 15:8 (if driver)
6	DBL exit address
7	ENT exit address
8	END exit address
9	Module index number (for use as index to special processing routines)
10	For handlers, this word contains the logical unit number associated with same. If multiple copies of the module are loaded, this will be the first of the assigned unit numbers. For drivers, the basic select code of the module is contained here. For drivers which are loaded multiple times for use as ASCII files, this word contains the address of the first .MASP entry which corresponds. For several basic modules, this word initially contains the number of associated select codes.

The order of entries in .NAMT is as follows: all drivers, all handlers, all other modules. The drivers and handlers sections

are in relative logical unit number order both for IOC and TSB logical unit numbers. A negative one value delimits the table.

Pseudo Base Page

One memory page of IOPC is reseved for use as a pseudo base page. On this page is contructed the base page which is eventually loaded into the ICP. Some locations are assembled into this page which are assumed by the IOP program. These are:

Location	Content
2	JMP 3,I for system start
5	HLT 5 for memory failure
6	CLC 6 for DMA channel 6
7	CLC 7 for DMA channel 7

Location 3 is later filled by IOPC with the last (hopefully only) transfer address encountered in the processed binarys. Location 4 is configured by the D.04 module modification appendages. Locations 10 through 77 are initially zero to indicate that none of the select codes have yet been allocated. They are set non-zero as they are allocated during the system dialogue processing.

Module Map Table (.MMT.)

Each 5 word entry in this table saves the name and beginning and ending addresses of each module loaded into the IOP:

Word(s)	Content
0-2	Module name
3	Module starting address
4	Module ending address

The table is terminated by a word containing a -1 value.

Entry Point list

Each 5 word entry in this list contains the name of an IOP program entry point plus its entry point address and base page link address. This list is kept in a reverse order from high memory downward:

Word	Contents
4	1st word of name
3	2nd word of name

2	3rd word of name
1	address of entry point
0	address of base page link

In addition, a count of all such entries is kept in variable LST. The LSTP routine described below, will establish five indirect pointers to a given entry with LST1 addressing word 4, LST2 addressing word 3, etc.

Functional specifications

Section 1 - System Configuration Dialogue

Subroutines

Before discussing the actual dialogue process, several subroutines will be outlined. All are used by section 1. The READ routine is used as a central console read subroutine. In addition to requesting a console read via the MCP console driver, a line feed is output to acknowledge the input. Also, the first input character is tested for control-G (BELL). The occurrance of this code is the operator's abort signal. READ then issues an abort message and terminates IOPC. Normally, however, READ simply returns to the caller.

The LDBYT subroutine extracts characters from the console input buffer and inserts them into the A register. The high order 8 bits of A are not disturbed. LDBYT advances the buffer pointer as characters are extracted. If a carriage return is detected, a space character is returned and a "+0" return is made. Normally, return is to "+1".

A YESNO subroutine processes responses intended to be yes or no. The caller provides the address of the question which YESNO outputs. Next, the response is read and the first character is input. A NO response ("N" or null input) causes a "+0" return. A yes response ("Y") causes a "+1" return. Any other response causes the question to be asked again.

SCREA is a select code input and analysis subroutine. Input is a value in A which represents the number of total select codes associated with the basic select code to be input. The B register inputs the address of a message part which describes the desired select code. SCREA first outputs the select code question. Then the OCTIN routine (see below) analyzes the input. Any error results in an appropriate diagnostic and reasking of the question. Correct octal input is validated as a legal select code. If

valid, the given code plus all associated codes are checked for previous specification. This prevents overlap. As SCREA allocates the select codes, it marks their allocation in corresponding locations on pseudo base page. Before returning, the select code is checked against, .MSCM (maximum select code), and .MSCM is updated if necessary.

DECIN and OCTIN process numeric input and return an internal binary value plus an indication of the terminator of the numeric string. Both use a common routine but set up a different conversion radix.

Dialogue processing

IOPCM is the entry point which receives control from MCP (via the IOPC entry point). IOPC is identified to the operator, and the optional reload question is asked. A yes answer leads to the reload routine described in appendix B. Normally, a series of configuration questions is asked. Processing of these is as follows. Note that in many cases, the outcome of the processing is to set .MCT. values. Other important details include the setting of copy counts and names in the module loading table. These settings will determine the eventual module selection.

DATE?

Any input, except null input, is accepted and moved to the header message which is part of the ouput of section 3 processing.

MEMORY SIZE?

Decimal input is accepted and compared to a list of valid specifications. The corresponding actual high memory address is established in .MEMS.

NUMBER OF PORTS?

The decimal number of ports is input and validated. Depending on the specification, the required version of the multiplexer driver is selected (A or B). This is done by setting the appropriate name in the module loading table. Also, the associated number of select codes is set at either 3 or 6.

BUFFER SIZE OPTION?

A no answer to this question results in no further processing, and the .MCT. default TTY buffer sizes are used. A yes answer allows input of alternate sizes. Any of the three formats of buffer size

specification are input and analyzed repetitively until an 'END' is input. Each specification is validated, and a range of ports is found (may be one port). The given size is then set in the corresponding slots of the .MCT. table's .MPBS entries.

XXX SELECT CODE?

SCREA is used to input each of the select codes for the time base generator, interconnect kit and multiplexer. The module loading table for the associated drivers initially contains the number of associated select codes (at offset .NAUN). This value is input to SCREA and is then overlaid by the select code returned from SCREA.

2741 TYPE TERMINALS?

A yes answer causes the copy count to be set to one in the module loading table entry for the 2741 module. At least one 2741 code must then be selected also. These questions are asked and the code module copy counts are also set as required.

INCLUDE RJE FUNCTION?

A no response causes this section to be skipped. A yes response leades to two other questions asking which version of RJE is to be included. Next a scan of the module loading table occurs. All RJE related modules receive a copy count of one. In addition, those which are variant (IBM vs. CDC) have the module names modified to reflect the appropriate required version. Then the SCREA routine is invoked to obtain the synchronous modem interface select code. For the IBM version only, the number of each type of host function is input. These values are stored in respective .MCT. cells (.MHRD, .MHLD, and .MHPD) and in respective .ACT. cells for specification of the maximum number of JT, JL and JP ASCII file types. Finally, the copy counts for HR, HI, and HP modules are set since these may be loaded in multiple. Next the ASCII files question is skipped since RJE requires ASCII files.

NON-SHAREABLE DEVICES?

For the yes answer to this question, all processing is controlled by the .ACT. table. For each .ACT. entry, the following is done:

1. If no RJE component is to be included and the file type is RJE related, then the .ACT. entry is skipped.

2. If only one possible quantity of the file type can be specified, then the quantity question is skipped. Otherwise it is asked.

3. The specified number is checked against maximum and minimum. If acceptable, the module loading table entries which are related receive appropriate copy counts. The total number of ASCII files is updated (.MAFN).

4. For each file of the type requested, the following is done:

   a. A .MASF entry is started with the correct device designator.
   b. Device characteristics from the .ACT. entry are moved to the .MASF entry. (This includes select code for SP devices.)
   c. For real IOP devices, the select code is input and added to the .MASF entry.
   d. If a subtype is indicated by the .ACT. entry, it is obtained and merged into the .MASF entry.
   e. Finally, the record size is input and added to the entry.

At the end of this processing, common ASCII file modules are set with appropriate copy counts (unless only SP devices are requested). An important fact about the above processing is that it results in the generation of .MASF entries in an order opposite that required for the IOP device table (.DVTB). This was only done as a matter of convenience here. However, it forces back to front processing of the table during later sections.

XX DEFAULT DEVICE ASSIGNMENT?

This question is processed only if RJE is included. First, validity checks of the number of ASCII files to the number of host functions is done. This is to insure that not more host functions exist than devices to be potentially associated with them. Next, the question is asked for each defined host function. The response is validated as a legal file type for association with the type of host function. If valid, a designator is contructed and stored in the appropriate .MHRD, .MHLD., or .MHPD entry. These are later used to construct the IOP .DAT table.

MAGNETIC TAPE COPY?

The correct IOP copy option is set in .MOPT.

Several final activities are now performed by section 1.

1. The select code for D.61 is copied to associated drivers D.62 and D.63.
2. For all unspecified select codes from 10 up to the maximum IOP select code, a HLT instruction is configured into the corresponding pseudo base page select code location.
3. The total number of modules to be loaded is calculated and saved.
4. Logical unit numbers are calculated and inserted into module loading table entries. This is done only for function handlers which are to be loaded and which have I/O complete entry points.
5. The first available base page location (.MSCM+1) is set.
6. The standard modules file is selected via the MCP.
7. The starting of the IOP loader is requested.
8. The IOP is loaded with zeros.
9. Section 2 is entered.

Section 2 - Relocating Load

Subroutines

Several subroutines which are a major part of section 2 processing are described first. Note that some of these routines (e.g. DIAG) are used by sections 1 and 3.

LINK is a routine used to allocate base page indirection pointers (link words). These link words are required for the resolution of off-page references of locations by memory reference instructions and DEFs not on the same page. LINK accepts input in register A. If A is zero, then a new base page link is forced to be allocated. If A is non-zero, a scan of existing links is made to see if one already exists. In any case, register B is used to return the address of the required base page link. LINK maintains its base page image on the pseudo base page but returned register B values are actual base page addresses. An overflow of base page results in an abort.

LSTI and LSTP are routines used to access the entry point list. LSTI prepares LSTP for a scan of the list by setting a controlling count and an initial set of pointers. Five pointers are used to allow indirect references to a given list entry. These pointers are adjusted by LSTP. LSTP returns "+0" when the list is exhausted. Normal return is at "+1".

Several terminal error exits are defined for common error types. All have entry points of the form LERx. All produce an appropriate message and then abort. Most add special information to the message to be output.

DIAG is a general console output routine. An inline DEF to the message is the only required input. If the DEF is indirect, this implies that IOPC is to be terminated and is not really an indirect DEF. In this case, DIAG returns to the MCP after properly repositioning the master tape. Messages are preceded by a positive count of the number of words in the message.

PACK, PUNCH, PPPP, and CKSS are routines used to prepare absolute load records for transmission to the IOP protected loader (see IOPLD). PACK adds a word to the absolute record and updates the checksum. PUNCH completes the checksum using CKSS and then calls IOPLD to load the record. PPPP is then called to reset pointers for new PACK calls.

IOPLD accepts input in register B indicating that a record is to be loaded into the IOP (B=0) or that the IOP protected loader is to be forced to stop (B=-1). A timout technique is used to insure that the IOP loader is running. Failure to receive a response from the IOP prior to the timeout will result in an IOPC abort (unless B=-1 input requested this action). Normally, each 8 bit byte of A is output to the IOP via the interconnect kit (bits 15:8 first).

Section 2 overview

Section 2 processing is essentially that of a relocating loader like the one used in ECS or MTS systems. However, there are additional functions. The overall flow of section 2 is:

1. Load modules from the standard modules file.
2. Position to the optional modules file. Select and load required optional modules (there are always some).
3. Construct IOP system tables, control blocks, etc. (.DAT, .DVTB, .COM., QIT's, et. al.)
4. Create a magnetic tape copy of the IOP program if required.

Several important extensions to the normal type of processing performed by a relocating loader are a standard part of steps 1 and 2 above:

1. The module loading table is used to select modules to be loaded. Unneeded modules are discarded.

2. Modules which are to be replicated are copied to an IOPC in-core buffer and repetitively processed from that area. This is to prevent the need for master tape rereading and to allow singular application of patches by the MCP.
3. Select codes and IOC logical unit numbers are merged into the relocated modules.
4. Optionally, module modification appendages can be invoked to perform special processing of the relocated image of a module being loaded. Also, special processing at the end of module loading can be done. These appendages are indicated in the module loading table entries. Three are defined as follows:

ENT exit - For each module entry point (except .SCxx or .LUxx), an exit is made. Registers are set as follows:

A = address of ENT entry as it appears in the ENT record

B = base address of the module being loaded

The exit routine is expected to return as follows:

+0 - normal return, retain the entry point

+1 - optional return, discard the entry point

DBL exit - Following the relocation of each module location, the exit is made. This is done prior to loading the location into the IOP. Registers are set as follows:

A = relocated value of location

B = location address

On return, B is of no consequence. However, A is used as returned. Thus, the exit may modify the value to be loaded. There are no optional returns.

END exit - After relocation and loading of the module, this exit is taken. No register values are provided to the exit, and no optional returns may be made. The exit has access to FWAM and an indication of the next available location and may use the ENT exit to define needed addresses within the loaded module. FWAM can be modified up or down by this exit as required.

Relocating loader

Processing of both the standard and optional module files is sequential. Routine LDRIN reads through the current file and performs minor validity checks on each record read. Each record is a single relocatable binary record as produced by an assembler. NAM, ENT, EXT, DBL, and END records are accepted. Further, it is assumed that all ENT and EXT records precede any DBL records. (Note that LDRIN can be forced to read from an in-core copy buffer for modules which are repetitively loaded).

LDRIN invokes the NAM record processor in all cases. This is to allow for the selection decision. Other record type processors are also invoked unless the NAM selection process dictates that the module is to be skipped. The PLFLG variable (program loading flag) is controlled by the NAM processor.

LDRIN includes END record processing. At END processing, any transfer address is set as the IOP program start address on pseudo base page. Note that only one such transfer address is assumed. Next, the entry point list is scanned, and the sixth byte of all names is set to zero. Originally this is an external ordinal which is used in each binary module to associate the "nth" EXT definition with any given external reference in a memory reference instruction or DEF. Next, the end appendage for the module is invoked. Following this, the module map table is updated for the end address of the module. The total module count is then adjusted. If more modules are to be loaded, LDRIN continues. Otherwise, control is passed to section 2 table construction processing.

LDRIN will position to the optional modules file for further module selection at the end of the standard modules file.

NAM processing

The main purpose of NAM processing is to make a load/noload decision about the current module. Note that all module selection is based on NAM records. Other actions include recording the module in the module map table, defining the module relocation base, and copying the module to an in-core buffer if a repetitive load.

NAMR first tests for an in-progress repetitive load. If a previously selected module is to be loaded more than once, the copy count is reduced. When it reaches zero, the binary is finally discarded. Otherwise, the unit number if any is

established (see below). Then the in-core copy is modified to give it the appearance of a new module:

1. The third or fifth character of the module name is incremented (third if a handler and fifth if a driver). For example, CR0 becomes CR1 and D.110 becomes D.111.
2. All ENT entries having names of the form XXdXX (handlers) or XXXXd (drivers), where "d" is a digit, have the d incremented. Processing continues at label NM3.

For new occurrances of modules, the module modification appendages are set to a dummy exit. Then the module loading table is scanned for the name in question. If not found but the standard modules file is current, the module is force loaded. This will allow the inclusion of special routines such as line printer core dumps. If not found and the optional modules file is current, the module is skipped. If the module is located, the following is done:

1. If the copy count is zero, the module is skipped.
2. If only one copy is needed, the module unit number is set as are its appendages. Processing continues at NM3.
3. If multiple copies are required, the module is copied to an in-core buffer. Then the module unit number and appendages are set.

At the NM3 label, module loading is set. The module name is copied to the module map table. Pointers to the table entry for recording module size are set. The program relocation base is set and possible memory overflow is checked. Control returns to LDRIN.

Two routines in the NAM processor are of concern. NMBUN is used to establish the existence of any select code or logical unit number which might be associated with the module to be loaded. This is used by the DBL processor for merging into I/O instructions and parameters.

CORED is used to perform in-core reads in place of calls to the magnetic tape driver. This is done for replicated modules.

ENT processing

All ENT names are first tested for the forms .SCxx or .LUxx. These entry points define locations in a module to which the

module's unit number (select code or IOC logical unit number) must be added. The occurance of such an ENT is discarded from further processing after calculating the address of the location in question. These addresses are counted and added to a special list used by the DBL processor.

Normal ENT processing is as follows. The ENT exit is taken, and on return, the entry point may be discarded. If kept, the previous occurrance of the name in the entry point list is checked. If not found, the entry point is added to the list. No base page link is generated at this time since no external references to it are yet indicated. Thus, the entry point list entry word 5 is set to zero.

If the name is found and the entry point address (word 4) is non-zero, then the entry point occurred previously. This is a duplicate erroneous entry point and IOPC is aborted. Otherwise, some previous reference to the entry point has occurred. In this case, word 4 in the entry point list is completed with the entry point address. The associated base page link also receives the entry point address.

EXT processing

The EXT name is first locked up in the entry point list. If not present, it is added to the list. The proper external ordinal is set. Also, a base page link is defined. This link will be used to resolve references to the external name with indirect references via base page.

An external ordinal is a non-zero value assigned to each external symbol defined by an EXT statement. This definition is done by an assembler. The assembler then uses this value when assembling memory reference instructions and DEFs which reference the external. That is, assembled with the instruction or DEF is an indication that the reference is external plus the associated external ordinal.

If the EXT name already exists in the entry point list, only the external ordinal is changed in the entry.

DBL processing

Words to be relocated occur in 5-word groups in DBI records. These groups are preceded by a word containing 5 3-bit indicators. Each defines the type of relocation to be performed on the respective subsequent 5 values:

Indicator	Type of relocation
0	Absolute value
1	Program relocatable
2	Base page relocatable (unsupported)
3	Common relocatable (unsupported)
4	External reference
5	Memory reference instruction: This is a two word entry. Word 1 contains the op code plus an indicator of the type of reference -- 0 - program relocatable 1 - base page relocatable (unsupported) 2 - common relocatable (unsupported) Word 2 contains the 15 bit address of the referenced location.

The DBL processor loops on these 5-word groups and performs relocation as noted below. When the relocated value is ready, a scan of the unit merging table is done to see if the word requires addition of a select code or logical unit number. Next the DBL exit is taken. Finally, the PACK routine is called to add the value to the absolute record. PUNCH is called whenever the absolute record is filled.

For type 0 and type 1 relocation, the value in the DBI record is simply added to zero or the module relocation base respectively.

For external references, the entry point list is scanned for the corresponding external ordinal. When found, the base page link address is merged into the word and the indirect reference bit is set.

For memory reference instructions, the address of the referenced location is generated. An on-page references causes merging of the page offset bits into the instruction. For off-page references, a base page link is generated. It receives the address of the referenced location and an indirect bit if the instruction is an indirect reference. The instruction in this case is made to indirectly reference the allocated base page link.

Appendages

Specific module modification appendages are outlined below.

D.61 - The ENT exit is used to locate words which the DBL exit will set (RJE indicator, DMA linkage words, and memory size). The END exit is used to return storage from unused LUT tables.

D.63 - The END exit is used to construct the ASCII file unit control tables.

D.51 - The END exit is used to release unused TTY tables.

All replicated handler modules- The ENT exit locates a word which the DBL exit fills with the relative copy number of the loaded module.

D.04 - The power fail interrupt linkpage is set on pseudo base page. The END exit then constructs the power recovery appendage list and count. It is assumed that D.04 is loaded after all drivers.

MUXH - The END exit deletes unused tables.

ICKH - Unused unit tables are deleted. Then the TTY buffers are allocated, constructed and loaded.

ASPH - The ENT exit locates the file characteristics list. The DBL exit saves a copy of these. The END exit overlays the characteristics list with constructed file tables

IOC - The DMAC2 variable is set to permanently allocate channel 7 to D.61.

MN - Interconnect kit logical unit number is merged to IOC parameter lists in the allocate manager.

Table construction

Many IOP tables and control blocks are built as follows:

.COM., QITs, and IODT

A pass is made on the handler entries of the module loading table. During this pass, the commutator is constructed and loaded directly into the IOP. The IODT and QIT's are built in SP storage for later loading into the ICP.

The .COM. entry point is defined. An indirect JSB to IOC is built for use in building the .COM. CRQ calls. An indirect JMP to the CMPLT routine is built, also for use in building CRQ calls.

The seven entries of IODT which handle erroneous references to units 0 through 6 are built.

For each handler entry in the module loading table corresponding to a loaded module this is done:

1. The first 3 characters of the module name are saved for use in constructing handler entry point names.
2. If the handler has a prime entry, a commutator gate and JSB to the entry point are constructed. The gate address is saved for use in building the QIT.
3. If the handler has an I/O complete entry point, the IODT copy is updated.
4. A QIT is constructed.
5. Steps 2 through 4 are repeated for replicated modules.
6. A .COM. CRQ call is built if the handler contains an I/O complete entry point.

Finally a catch-all CRQ call is added to the .COM. as well as a final JMP to .COM.

The IODT is completed and is transferred to the IOP.

Next, the QIT's are tranferred to the IOP. Several special DEFs are also contructed as follows:

```
QITAD     DEF   first QIT
QITND     DEF   QITND (follows QITs)
QITNR     DEF   first allocatable QIT
QITMX     ABS   max TSB unit number
QITAF     DEF   ASCII files handler QIT
```

BPRTs, Buffer Pools, and SCOLs

Addresses of each of the three BPRTs is saved as each is set to zeros and loaded into the IOP.

Buffer subpools 1 and 2 are constructed. Addresses of the first and last buffers in each pool are saved for inclusion in the SCOLs. The number of control buffers constructed is either five or 1/2 the number of TSB ports, whichever is greater. Five console buffers are always built.

Each SCOL is built using addresses saved from building the BPRTs and subpools.

Finally, the buffer manager's PULPT list is overlayed with the three SCOL addresses.

.DVTB

The .DVTB entry point is defined and loaded with a DEF *+1. The negative number of ports is loaded followed by the negative number of ASCII files. Finally, the .MASF entries are processed in reverse order tc yield each 5-word .DVTB entry.

.DAT

The .DAT entry point is defined and loaded with a DEF *+1. then the .MHRD, .MHLD, and .MHPD lists are used to construct the .DAT entries. Two -1 values are added to terminate the .DAT.

EQTs and Interrupt Linkages

A scan of the driver entries in the module loading table is done. For each loaded driver, the following is done:

1. The driver number is converted to ASCII character form for use in name lcokup.
2. The number of related select codes is saved as are the driver characteristics (used to build EQT word one).
3. The related select code is derived and saved.
4. The number of EQTs is incremented. The EQT is constructed in SP storage using saved information.
5. For each related select code, an interrupt vector value is established. It is assumed that I.XX is the first interrupt entry point and that J.XX, K.XX, etc. are related to susequent select codes. If a given entry point is not defined, a halt is installed in the interrupt vector. Note that the related undefined entry point is skipped. These vectors are built on the pseudo base page as are the related interrupt linkages.

The XEQT entry point in IOC is now overlaid with the 1st address for the EQTs. This address is alsc saved for section 3 report generation use. Finally the EQTs are loaded into the IOP.

.MEM.

The memory table is overlaid with a copy of the memory allocation table and address of the cold dump analyzer list.

Cold Dump Analyzer List

A list of addresses and other parameters is generated and loaded into the IOP. The address of this list is moved to location 3 of the IOP by the IOP software at system start. Undefined addresses result in zero values. The content of this list is:

Word	Contents
0	DEF .DAT
1	DEF .DVTB
2	DEF QITAD (address of QITs)
3	DEF PULPT (address of SCOL list)
4	DEF QTOP (address of D.43 TQE chain head)
5	DEF LUT51 (address of D.51 unit tables)
6	DEF LUT61 (address of D.61 unit tables)
7	DEF UCT00 (address of D.63 unit tables)
8	DEF SIBIK (address of ICKH unit tables)
9	DEF SIBAF (address of ASFH unit tables)
10	DEF D.61S (address of D.61 trace pointer)
11	DEF TTPTR (address of MN trace pointer)
12	DEF CRQ (address of .IOC. CRQ head)
13	DEF QHEAD (address of .IOC. free storage)
14	DEF STST (address of SYNxx tables)
15	DEF PNTR (address of MEMRY free list)
16	DEF EQTs (address of EQT tables)
17	Copy of .MOPT
18	Address of 1st buffer in pool 1
19	Address of last buffer in pool 1
20	Address of 1st buffer in pool 2
21	Address of last buffer in pool 2

Base page loading

After all table construction, the pseudo base page is loaded into actual IOP base page. If any required modules were not loaded, a warning message is issued.

Tape Copy

If no tape copy is requested, section 3 is entered. For a master tape copy, the presence of a write ring is insured. Then the master tape is positioned to the last file of the tape which has file ID 65000. This is the optional copy file.

For a separate tape, the master tape is unloaded. The operator is asked to mount the separate tape. When this is done, the presence of a write ring is insured. Next, a loader program is written to the tape followed by a file mark. (See Appendix A for a description of the loader program).

The operator is asked to start the IOP at location 2002. This causes a JMP to an unloader program which is part of the IOP D.61 driver. The unloader transfers all of the IOP core image back to the SP. This data is formatted back into absolute records and written to the tape. Finally, the report generator is entered.

Section 3 - Report Generator

The report produced by IOPC is described below. First a summary of the sources of the data for this report is given.

I/O linkage list - the pseudo base page select code vectors are scanned in order. For each one, a line of output is produced (potentially 3 for the interconnect kit, a special case). The linkage is shown, and if the select code can be related to an EQT, the EQT information is added.

Base page links, memory bounds - the memory allocation table provides the data for this output. Also, an estimate of needed free storage is made, and a warning is issued if:

Available storage $\leq 100+100*R+(HR+HL+HP)+100*A$

where R=0 if no RJE
1 if RJE
Hx=repective host functions
A=number of ascii files

It must be pointed out that this estimate is very very rough. It attempts to reflect the need for RJE buffers by one-half of all host functions concurrently and all ASCII files concurrently. However, the formula assumes a 400 byte communications buffer which may be incorrect for some IBM systems and is certainly incorrect for CDC host systems.

ASCII files - a summary of the .MASF entries is produced

host function assignments - the .MHRD, .MHLD, and .MHPD tables are summarized

module map - The contents of the module map table is listed

QIT summary - The in-core copy of the QITs previously built is used to calculate QIT addresses and designators.

Entry point list - The entry point list is sorted and then printed.

Report Outline

The report produced by IOPC supposedly contains sufficient information to assist the analyst in resolving program problems. It also gives certain information of value to the user. This would be information about the size of a given configuration as well as available and/or unused memory space. Such information can be valuable in making decisions about system performance as well as upgrades which may be considered. The report is outlined below.

I/O PROCESSOR MEMORY MAP DATE=8/6/75

This line of the report is the first line output. It identifies the map and also identifies the particular configuration with the information supplied by the operator to the DATE? question of the initial dialogue sequence. It is not necessary that that information be restricted to a date. In fact, some other identification of the system type might be advisable.

I/O LINKAGE

S.C.	DRIVER	I.L.	I.E.	U.N.	EQT
A	B	C	D	E	F

The I/O linkage list gives a complete summary of the relationship of select codes to interrupt entry points, .IOC. logical unit numbers, and equipment tables (EQTs). Each line of the report corresponds to a given select code, and the select codes are all reported from 10 through the maximum select code used (.MSCM). Not all of the A, B, C, etc. values will appear in each line. This is true in those situations where multiple select codes are associated with a given EQT. The contents of each line are as follows:

A Select code

B Name of driver (same as driver's initiator section entry point name). This field of the report may contain blanks if the select code is one of several associated with a given driver. The field may also contain the word "HLT" if there is no interrupt entry point associated with the select code. In this case, the actual location will contain a HLT instruction with a halt value corresponding to the select code.

C Base page address which is object of indirect JSB located in select code vector. The content of this base page location is the address of the associated interrupt entry point. These locations are collectively referred to as interrupt linkages.

D Interrupt entry point (same as entry point name of driver's continuator section).

E .IOC. logical unit number.

F Address of EQT associated with this select code. This value and the E value will appear only for the first select code (base select code) associated with a given EQT. Note that for multiple select code devices, the IOP program currently assumes that the select codes are contiguous.

The select code associated with the interconnect kit is a special case any may result in one, two, or three lines in this report. This is because in addition to the basic D.61 driver, two other drivers may also be loaded into the system (D.62 and D.63).

BASE PAGE LINKS - X-Y

Here are reported the first and last base page addresses used as indirect linkages. All other base page locations from Y+1 to 1777 are available.

AVAILABLE MEMORY - X-Y

Unused memory from location X to location Y is reported. Note that this area of memory and any unused base page are the free areas available for dynamic buffer allocation.

ASCII FILES

L.U.	DESIGNATOR	SELECT CODE	RECORD SIZE
A	B	C	D

This report appears only if at least one ASCII file is defined into the system. Each line of the report describes one ASCII file as follows:

A Logical unit number of the ASCII file as used by BASIC programs.

B ASCII file device designator and possible device subtype indicator. Subtypes appear only for those devices which come in several flavors such as line printers and card readers.

C The select code of the device is given if applicable. No select codes are associated with RJE pseudo devices. Note that the select code for MT and PR devices are SP select codes.

D Default record size in words which will be allocated for use with the device. This value may be over-ridden by the BASIC programmer.

HOST FUNCTION	DEFAULT DEVICE
A	B

This report is not produced if no RJE component is included in the system. The report shows the default assignment of ASCII files to host functions which will be made in a freshly loaded system. In other words, this report reflects the initial content of the .DAT table.

MODULE MAP

Each line of this report may contain up to three entries. Each entry describes the name of a loaded module as well as its beginning and ending addresses. Note that if any module modification appendages adjust the image of the loaded module, the size may not correspond to the assembled module size. One example is the interconnect kit handler. For this module, the END exit discards unused tables and then constructs all TTY buffers. These memory occupied by these buffers will be reflected in this report as a part of the interconnect kit handler.

QIT SUMMARY

For each function handler in the IOP program, a QIT control block is constructed. This major control block is used to hold work in the form of buffers for the handler. In addition, various handler entry points are noted in the QIT. Each QIT contains a name very much like the ASCII files device designators. In fact, some names correspond directly, while others are unique (e.g. the one for the interconnect kit handler). This report gives the designator associated with each QIT as well as the QIT address.

ENTRY POINT MAP

Here are listed all entry points in the system together with their entry point location values. If any entry points are discarded by ENT exits, these will not appear. Also, the .SCxx and .LUxx entry points will nct appear. The list is alphabetized. If an entry point is referenced but is rever defined, the report will reflect this fact by using "------" rather than an entry point lccation value. Certain of these unresclved references are acceptable. An example is the reference to the 2741 modules by D.51. It is, in fact, the case the D.51 detects the presence of the 2741 modules by observing the resolution cf the reference.

IV. Appendices

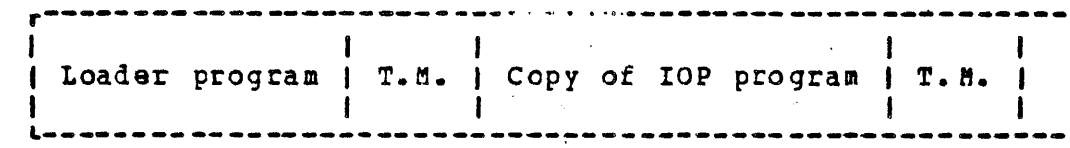
The following appendices are attached to this document:

Appendix A -- Separate tape IOP lcader description
Appendix B -- Optional IOP reload procedure
Appendix C -- Maintainerce guidelines
Appendix D -- Restrictions on relocatable modules

Appendix A

Separate tape IOP loader description

When the optional copy of the configured IOP program is transcribed to a magnetic tape different from the master tape, a special IOP loader program is also written to the tape. This loader program is suitable for loading by the HP2000 ACCESS bootstrap program which is distributed in paper tape form. When this loader program has been loaded, it is given control by the bootstrap program and proceeds to load the configured IOP program into the IOP. The format of a tape written for this purpose by IOPC is as follows:



T.M. = Tape Mark

The loader program reads the second file and transfers it to the IOP protected loader. This is similar to the IOPC optional reload process which reads the special master tape IOP copy file and transfers it to the IOP protected loader.

The loader program is initially an assembled part of IOPC itself. The memory page which it occupies in IOPC is the same memory page which it will occupy when loaded by the HP2000 ACCESS bootstrap. This is to say that when IOPC copies the loader to the first file of the separate tape, its absolute record images specify this given memory page. Additional instructions are added to cause a JMP to the loader from location 4000 which is the assumed start address for all programs loaded by the bootstrap.

The internal flow of the loader program is as follows. The program contains three small I/O drivers each of which is non-interruptable (SFS logic). The first of these drivers is a simple console output driver. This is used to output operator assistance messages. An interconnect kit driver is also included. This driver is used to output the absolute records read from the magnetic tape to the IOP protected loader. This driver contains a timeout procedure which is used to detect a non-operational IOP loader. The third driver is, of course, the magnetic tape driver.

This driver employs DMA to read the copy of the IOP program from the magnetic tape.

The loader receives control from the bootstrap. The S register is preset by the boctstrap with the magnetic tape select code. This select code is configured into all magnetic tape I/O instructions. Next, the operator is asked to start the IOP protected loader and to signal the completion of this action by pressing CR. (Actually, any input is accepted.) A program loop is now entered in which tape records are read from the second file of the tape and transferred tc the IOP. (Note that no tape positioning is necessary, since the bootstrap program will have advanced to the second file by reading the first file which is the loader program.) The length of the records read from the tape is used to determine the amount of data to be transferred to the IOP. If any tape errors occur, an error message is issued, and the loader halts. No timeout by the interconnect kit driver should occur prior to the occurarce of the tape mark at the end of the IOP program copy file. If one does occur, an error message is issued, and the loader halts. At the end of the file, zeros are loaded into the IOP until a timeout does occur. This will insure that the IOP protected loader will go to a normal halt.

Messages and halt codes used by the loader program are as follows:

START IOP PROTECTED LOADER. PRESS RETURN
This message is issued tc the operator to inform him that the loader program is ready to proceed with loading. The operator should at this point insure that the IOP protected loader is running and then press CR to signal that the loader may continue.

IOP IS NOT RESPONDING
This message indicates that the IOP protected loader has stopped accepting data from the loader prematurely. This message is followed by a HLT 66. No recovery is possible short of restarting the entire loading process from scratch.

TAPE ERROR
Some tape error has occurred such as parity, timing, etc. The message is followed by a HLT 22. No recovery is possible short of restarting the entire loading process from scratch.

Appendix B

Optional IOP reload procedure

A yes answer to the RELOAD? question of the initial IOPC dialogue section indicates that the optional IOP copy file is to be loaded into the IOP. Such a copy can be created by IOPC at an earlier point in time for use in restoring the IOP program for production use. The flow of control of the reload process is outline below.

A message is issued to the operator instructing him to start the IOP protected loader. When this is done, the operator signals the completion of the action by pressing CR. The reload program then positions to the optional copy file on the master tape. This file has ID 65000. Next a loop is entered in which records are read from the tape and transferred to the IOP. The magnetic tape driver in the MCP and the interconnect kit driver in IOPC (IOPLD) are used to accomplish these actions. The records read from tape are in the form of absolute binary records. The length values in these records dictates the amount of data to be transferred to the IOP. Several error possibilities exist. If the IOP should fail to accept the data, an appropriate message is issued, and control is returned to the MCP. If no IOP copy exists on the tape (indicated by immediate tape mark), an appropriate message is issued, and control is returned to the MCP. It is also assumed that the magnetic tape driver in the MCP will take appropriate corrective action for any tape errors. Normal completion of the load process is indicated by forcing the IOP protected loader to go to a normal halt. This is done by sending zeros to the IOP until it fails to respond. Control is then given to the MCP.

Appendix C

Maintainence guidelines

There are some minor and/or obscure details about IOPC and/or the configured IOP systems which are of importance to maintainence personnel. As many of these details which can be remembered are outline below:

1. All modules in the standard modules file are loaded into the IOP by ICPC. This is true even if the module cannot be located in .NAMT. Only those modules from the optional modules file will be loaded which are required. This can be a handy tool to force-load specialty programs such as stand alone memory dumps, performance analysis tools, etc.

2. A simple technique tc effect patches in configured IOP programs exists. The use of unused base page for the location of patches is generally simple. In crder to place patches on base page, simply patch the second word of the .MEM. table to reserve as much base page as necessary. This location defines the last word of available base page for use in satisfying dynamic buffering requests. If this location is modified prior to starting the system, the area will never be made available to any cther part of the system.

3. Patches can be made in the configured prcgram prior to copying the IOP program to the magnetic tape. This is done prior to starting the IOP at location 2002 when directed to do so by IOPC.

4. The cold dump analyzer list is outlined elsewhere in this document. When the IOP program is started, the address of this fixed length list is placed in location 3. It can be used tc format cold dumps as well as to examine important control blocks, et. al. in a running system (with DUM).

5. Although there are several control blocks which are not assembled, these same control blocks can te easily located. Examples are the .COM., QITs, EQTs, and SCOLs. Finding most of these control blocks is done as follows:

   .COM. is found via its .COM. entry point.

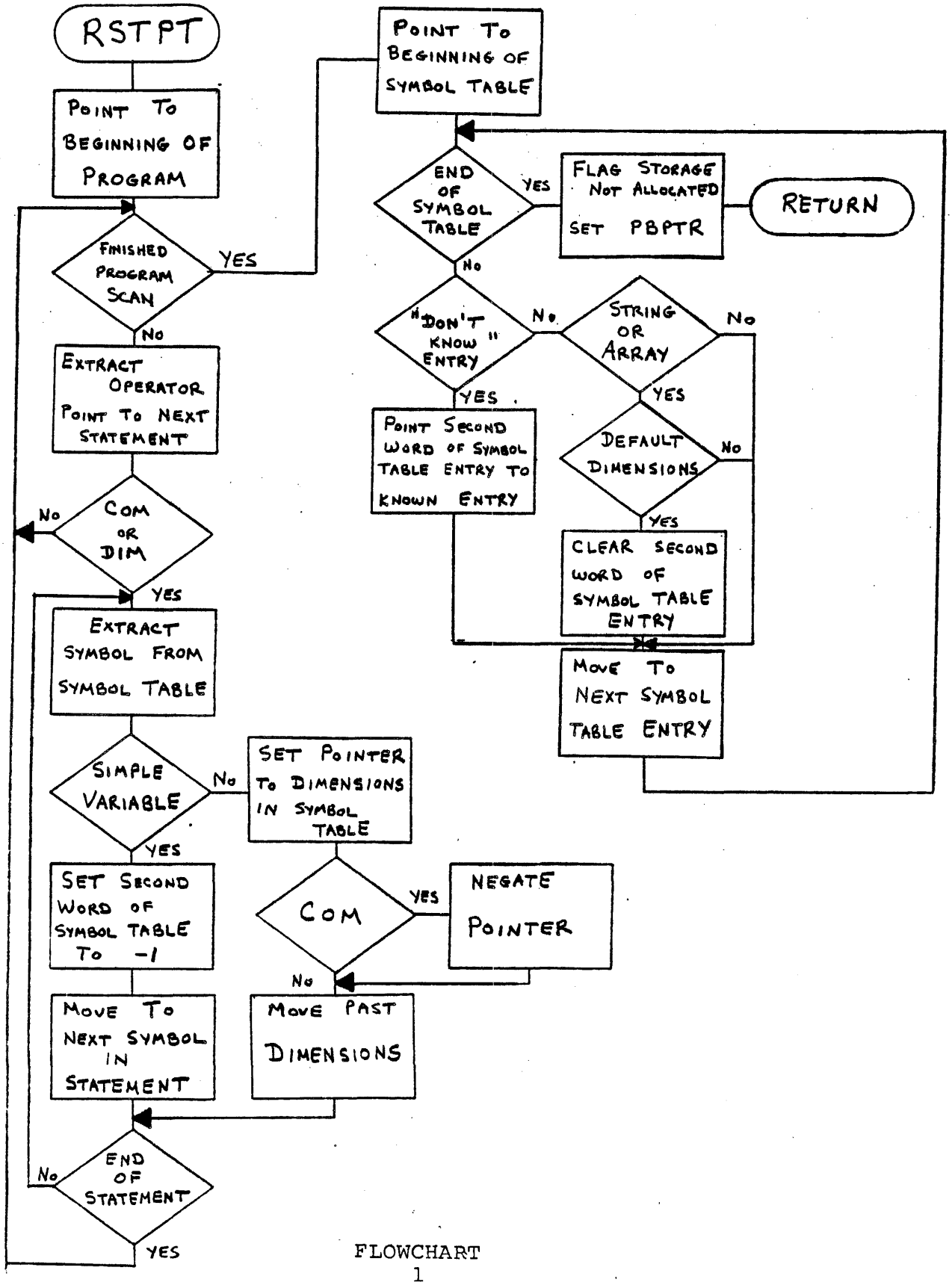
QITs can be located using the QIT summary in the IOPC report.

EQTs can be located using the I/O linkage list in the IOPC report.

SCOLs and BPRTs are located by finding the PULPT entry point. At that location are the three addresses of the SCOL tables. These, in turn, address the BPRTs.

The .DVTB and .DAT tables may be found via their respective entry points.

RSTPT



FLOWCHART

Appendix D

Restrictions on relocated modules

The IOP program modules, which IOPC configures into a running system, are expected to be assembled following certain guidelines. These are:

1. If the module is a driver or handler which depends on some I/O unit reference number (select code or IOC logical unit number), this number can be merged into I/O instructions and parameter words by IOPC. This will allow the module to be written excluding initialization code normally intended to perform this activity. The method by which this is done is to assemble each such instruction or parameter word with its relative select code or IOC unit reference number. Most drivers, for example, which use only one select code will assemble all I/O instructions using a zero select code. Those which depend on two or more should be assembled using 0 for the first, 1 for the second, and so on. In addition, all such instructions must be identified with an entry point of the form ".SCxx" or ".LUxx" for select codes and logical units respectively. (The "xx" may be any unique characters.) IOPC will add the base select code or unit number to these instructions when configuring the IOP program.

2. The names assigned to program modules are very important. All module selection is on the basis of the name found in the NAM statement. In addition, certain parts of names are expected to have certain forms for IOPC use:

   Variant RJE modules are expected to reflect themselves via different names consisting of a basic module name and a version indicator. For example, the synchronous function handler is expected to have the name "SYN" followed by either "IB" for the IBM version or "CD" for the CDC version. Thus, "SYNIB" or "SYNCD" will be appropriately selected by IOPC.

   The first three characters of all function handler module names are used to generate corresponding module entry point names for prime, I/O complete, and initiator entry points. For example, the SYNIB module is expected to have entry points with names "SYNHP", "SYNHC", and "SYNHI". Of course, these entry points

are not needed if the module does not require the respective type of entry point.

For handlers which may be replicated, the third character of the name as well as the third character of entry point names is expected to be a digit. Thus, for example, the card reader handler module has a name of CR0 and entry points CR0HI, CR0HC, and CR0HP. IOPC will increment these digits each time a module is replicated.

For drivers which may be replicated, the fifth character of the name as well as the fifth character of entry point names is expected to be a digit. Thus, for example, the card reader driver module has a name of D.110 and entry points of D.110, I.110, and P.110. IOPC increments these digits for each copy of the module.