COMMON

Proceedings of the Chicago Meeting

April 8-10, 1968

Pick Congress Hotel

TABLE OF CONTENTS

TABLE OF CONTENTS/cont'd

TABLE OF CONTENTS/cont'd

SESSION REPORT

COMMON - Chicago

Session Number MON B1 Session Name 360 Full Project

Chairman A. Ragsdale Meeting

Time 10:30 to 12:00 Attendance (No.) 120

Speakers Mr. R. D. Pollit - IBM

"Green Words" in FORTRAN

OS/DOS Data Compability

Synopsis of Meeting "Green Words" in FORTRAN are becomming extinct.

It was stated that the main incompatibilities of data between OS and DOS were as follows: 1) Tape labels - OS does not support DOS user labels

2) Disk incompatibilities were:

a) Track initialization

b) Location and content of VTOC

c) Labels

d) File definition and method of deleting records fror Indexed Sequential Files.

Speaker: R. D. Pollitt

Topic: Greenwords in OS and DOS
OS/DOS Data Compatibility
Cross Use of DASD Utility Programs

Greenwords in OS and DOS FORTRAN

Greenwords existed in unformatted, sequential FORTRAN data sets through DOS Release 13 and OS Release 11. Because they were not compatible with any other S/360 function, they were eliminated in both DOS and OS.

A greenword was the vehicle for telling FORTRAN that a logical record spanned more than one physical record. Something was required because the maximum length of an unformatted physical record in DOS was 256 bytes. The greenword occupied the first 4 bytes of the record. Byte 1 indicated whether additional physical records were required to complete a logical record. Bytes 2 through 4 contained the record length. The first non-zero byte 1 encountered signaled the end of the logical record and the number of physical records.

New FORTRANs do not handle greenwords. Therefore, IBM has provided data conversion utilities to purge data sets of greenwords and establish 360 standard record length fields in their place.

OS/DOS Data Compatibility

I. General

A. Data records created by the same access method are generally the same, with some exceptions which will be described.

B. Most incompatibilities regarding tape files are associated with labels.

C. Disk incompatibilities involve labels, some elements of access methods and some record incompatibilities.

II. Tape Data Sets

A. Standard Labels

DOS tape files are written with one header (HDR) label. OS/360 tape files are written with two header labels. DOS will accept an OS created tape file as input and will automatically bypass the second label. OS will accept a DOS created tape as input, ignoring the absence of the second label provided that its information is provided by the DCB or DD statement. This information includes:

1. Record format
2. Block length
3. Record length
4. Tape density.

B. Non-Standard Labels

The DOS user provides routines in the individual program that references non-standard labeled tapes. OS/360 requires

that common routines be coded and included in the system at system generation time. These routines are:

1. Input header
2. Input trailer
3. Output header
4. Output trailer.

The appropriate non-standard label routine is selected, brought into main storage and executed whenever an OPEN, CLOSE, end-of-volume or end-of-data set condition occurs.

C. User Labels

DOS supports user labels while OS does not. OS will skip existing user labels and will not create new ones.

D. Unlabeled Tapes

DOS unlabeled tapes can have a tapemark before the first file. The languages handle the tapemark as follows:

Assembler	optional, specified by programmer
COBOL	generates tapemark
FORTRAN	no longer generates tapemark, will bypass if present
PL/I	optional, default option generates tapemark
RPG	generates tapemark

OS unlabeled tapes do not have the tapemark. A tape created under OS and used as input to a DOS program that expects a tapemark as the first record will read past the file looking for a tapemark.

If a tape created by DOS is used as input to an OS program and the first record is a tapemark, the user must state in the DD statement that the file is the second data set on the tape. This method applies only to a single volume data set. The data sets could then be concatenated and LABEL=2 can be coded in the DD statement for each data set.

E. Checkpoint Records

OS does not permit checkpoint records within data sets. DOS does permit them.

F. File Name

In both DOS and OS the file name may be 17 characters. In DOS the names can be any format, including embedded and trailing blanks. In OS the label data set name must have no embedded blanks and must not have more than 8 characters between periods.

Eg.	File Name	DOS	OS
	MASTER FILE	OK	NO
	MASTER.FILE	OK	OK
	MASTERFILE	OK	NO

III. Disk Data Sets

A. VTOC Differences

1. DOS permits the VTOC to be on cylinder 0, track 0. Under OS, the VTOC may not be on cylinder 0, track 0.
2. DOS does not build a Format 6 DSCB for split cylinder files. OS does and, therefore, OS could not process split cylinder files created by DOS.
3. OS maintains a Format 5 DSCB to describe unused space. DOS does not. OS uses this to provide automatic control of direct access space allocation and could not perform this function on a DOS generated pack.

B. Track Initialization

1. The BPS DASDI (Initialize Disk) used by DOS users to initialize a disk and the OS DASDI are quite dissimilar. Some of these differences are significant enough to prevent interchange of disks between the two systems.
2. Flagging of defective tracks during initialization is handled differently. In OS, the RO data length is set to 0. BPS DASDI sets the RO data length to 8. DOS interprets the OS data length of 0 as an end-of-file condition.

C. Direct Access Space Allocation

OS provides the capability of automatic control of direct access space allocation. The user must specify only the estimated minimum size of a data set, plus the size of one or more incre-

mental additions; the system will automatically allocate this space dynamically as the data set is generated and/or stored. If more increments were specified than were needed, the system will release the unneeded increments for other uses. DOS has no such capability.

D. Direct Access Method Differences

1. In OS, when the direct access data set is created, dummy records are written for those records which do not exist at creation time. The capacity record indicates that no space exists on the track. When a new record is created, the dummy record is overlaid with this new record.

   In DOS, only valid records are written when a Format F data set is created. The capacity record reflects the remaining available space. When new records are added to the DOS data set, the capacity record is interrogated, the data record written and the capacity record updated to reflect the addition of the new record.

   This difference forces the DOS user to convert such direct access data sets before using them in an OS system or to refer to them in OS as Format U data sets.

2. In DOS, direct organization files occupy absolute extents assigned by the programmer. In retrieving a record, the user must present IOCS with a specific track address of the desired record. In OS, direct organization files can occupy extents assigned by DASD space allocation routines. In retrieving a record, the user can present the access method with the track address relative to the beginning of the data set. BDAM will compute the actual address, dependent upon the current actual location of the data set.

E. Index Sequential Access Method

1. In DOS, the user must establish his own convention for indicating which records are delete records and he must also decide how and when to delete them.

2. OS provides a feature for deleting records automatically. The first byte of the data record is reserved for a delete code. This code is specifically "FF" hexidecimal. When sequentially retrieving, "FF" records are not retrieved. When randomly retrieving, "FF" records are returned and the user must check and delete. Deletion records are automatically dropped from a data set when:

   a. An addition to the track containing the record marked for deletion displaces this record from its prime track (it is dropped rather than displaced to an overflow track).

   b. The data set is reorganized.

F. A conversion process generally must be performed for DASD volumes created under DOS before they are usable with OS programs for the reasons stated above.

SESSION REPORT

COMMON - Chicago

Session Number **Mon B2** Session Name **1800 Soundoff and Update**

Chairman **R. W. Forstrom**

Time **10:30 to 12:00** Attendance (No.) **130**

Speakers

Synopsis of Meeting **Question and answer session between users and IBM. New SRL manuals available:**

C26-3720 MPX Subroutines

C26-3724 MPX Techniques

SESSION REPORT

COMMON - Chicago

Session Number MON B3 Session Name 1620 Project

Chairman H. B. Kerr

Time 10:30 to 12:00 AM Attendance (No.) 55

Speakers George German

Frank Bush

Synopsis of Meeting Well attended. Excellent presentations. Good questions and answers session. Space was far too small and cramped. Good visual aids were present and were used.

SESSION REPORT

COMMON - Chicago

Session Number MON B4 Session Name 1130 MONITOR Version II

Chairman P. J. Woodrow

Time 10:30 to 12:00 Attendance (No.) 218

Speakers Mr. Dudley Dinshaw and Mr. Gene Lester - IBM, San Jose

NOTE: This session was repeated at 5:30 PM Monday to about 30 people who could not fit into original room

Synopsis of Meeting This tutorial session, primarily on differances between the 1130 Disk MONITOR, Version I and Version 2, was split into two talks. Mr. Dinshaw discussed the new features of the assembler and FORTRAN compiler. Mr. Lester discussed changes in the Disk Utility Program and Supervisor. Numerous questions were asked (and most answered immediately) concerning various aspects of the new version of the IBM 1130 Disk Monitor System.

SESSION REPORT

COMMON - Chicago

Session Number MON B5 Session Name 360/20 Organizational Meeting

Chairman R. L. Mason

Time 10:30 to 12:00 Attendance (No.) 8

Speakers R. L. Mason - Westinghouse

Van Hettinger - IBM

Synopsis of Meeting The meeting was conducted to determine what interest there might be in establishing a separate project group within Common for the 360/20 users.

An informal discussion followed which indicated interst on IBM application packages for the Model 20. Three of the participants were representing mult-plant companies which in turn could provide a source of attendees at future meetings.

Tentative plans were made to conduct another sessions in Philadelphia to: 1) Further organize the group and 2) have a separate session for a Bill of Material Processor discussion.

SESSION REPORT

COMMON - Chicago

Session Number MON C1 Session Name 360 - Full Project Meeting

Chairman W. Norton

Time 1:30 to 3:00 PM Attendance (No.)

Speakers Mr. B. Ferguson - IBM, Chicago

Synopsis of Meeting Data Management and the Linkage Editor.

MON C1

OPERATING SYSTEM /360

DATA MANAGEMENT

OS DATA MGT. 3

CONTROL FACILITIES

- LOCATION OF DATA
  - LABELING
  - CATALOG OF DATA SET NAMES
  - AUTOMATIC VOLUME RECOGNITION
- DIRECT ACCESS SPACE MANAGEMENT
- PASSWORD PROTECTION
- DEVICE INDEPENDENCE

ACCESS METHODS

DATA SET ORGANIZATION	LANGUAGE CATEGORY	
	QUEUED	BASIC
SEQUENTIAL	QSAM	BSAM
INDEXED SEQUENTIAL	QISAM	BISAM
DIRECT		BDAM
PARTITIONED		BPAM
TELECOMMUNICATION	QTAM	BTAM

OS DATA MGT.

ACCESS FACILITIES

- DYNAMIC USAGE OF CORE
- EXCLUSIVE CONTROL OF RESOURCES (CPU, DATA SETS, RECORDS, PROGRAMS, ETC.)
- REORGANIZATION INFORMATION

OPERATING SYSTEM /360

LINKAGE EDITOR

OS LINKAGE EDITOR

15 K

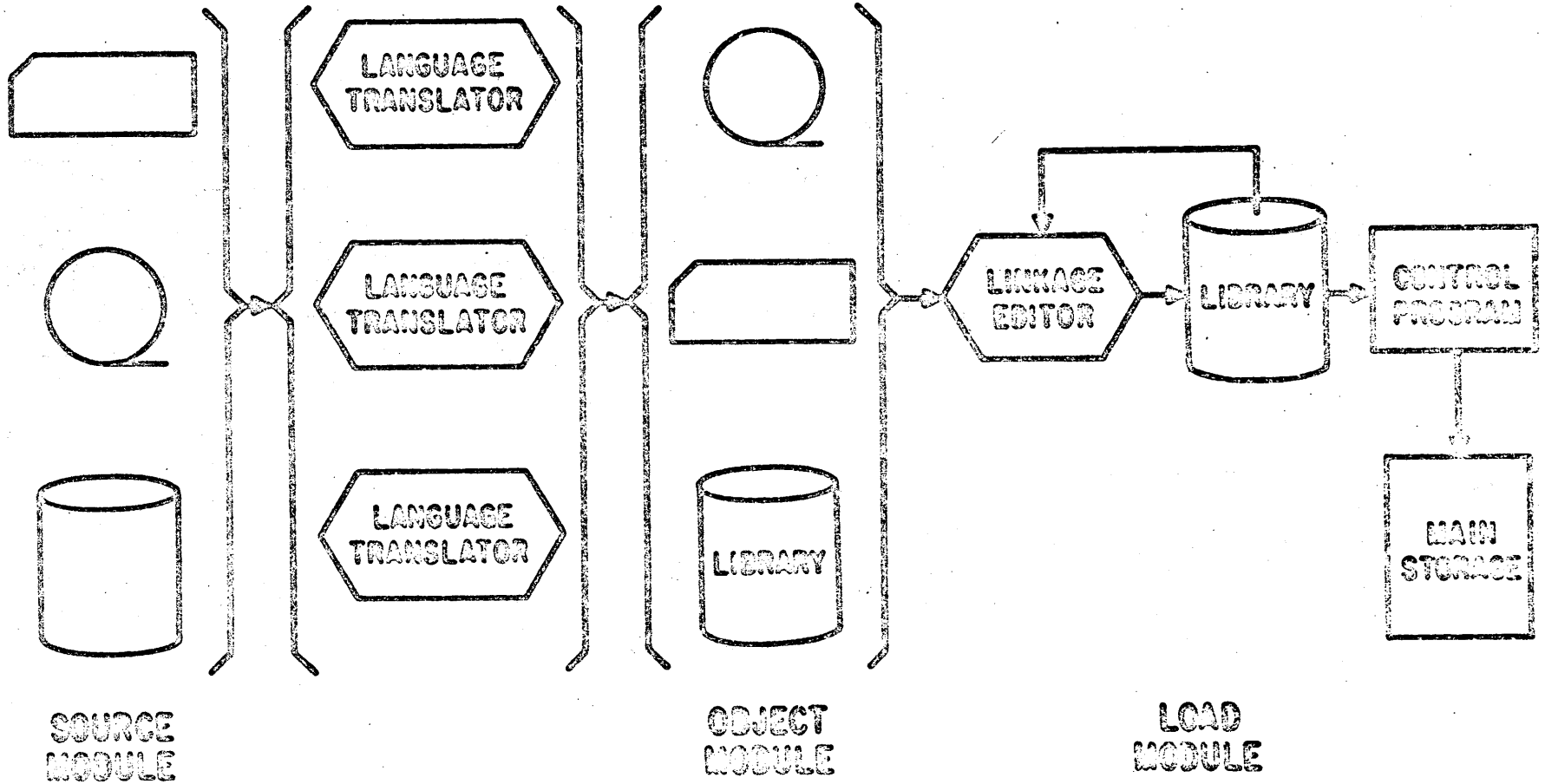
18 K

44 K

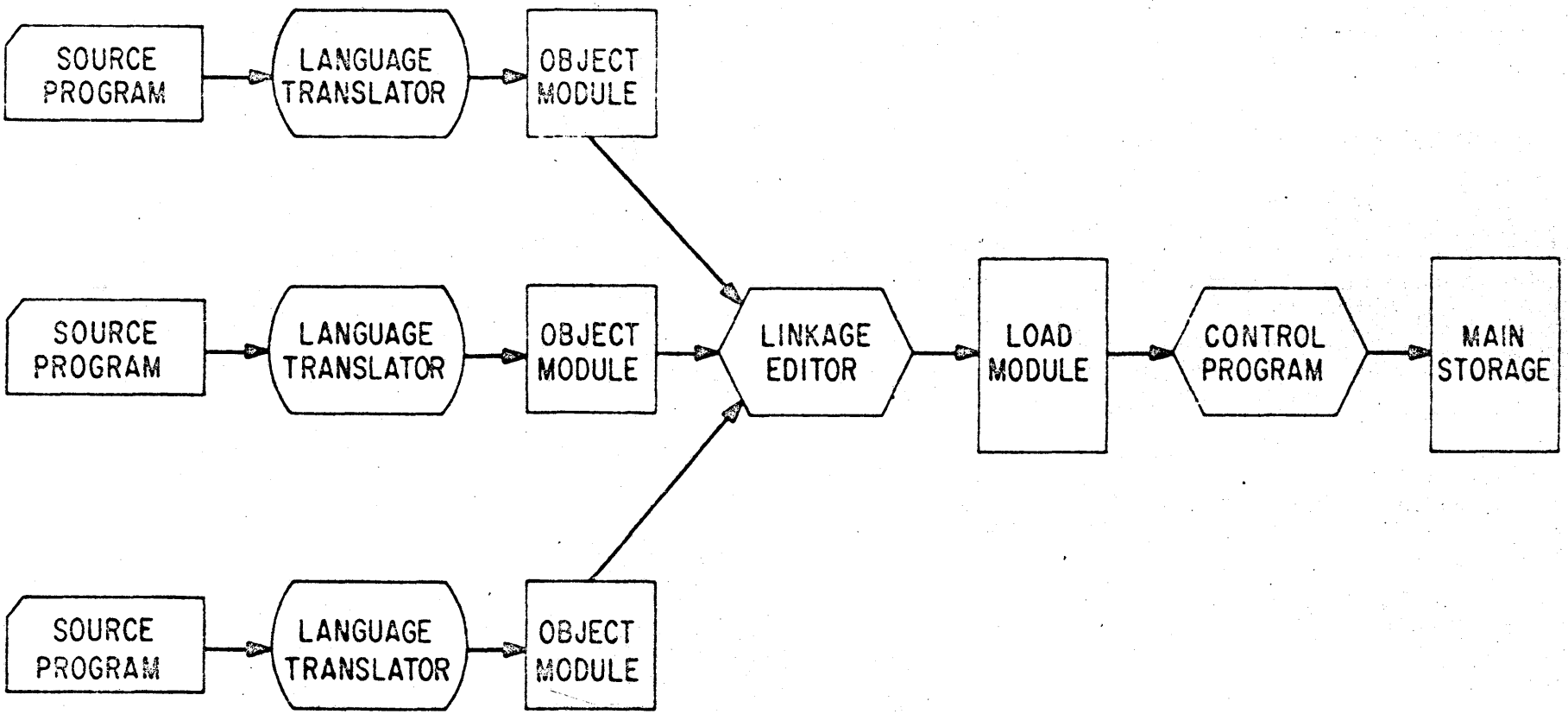
88 K

128K

PROGRAM PREPARATION



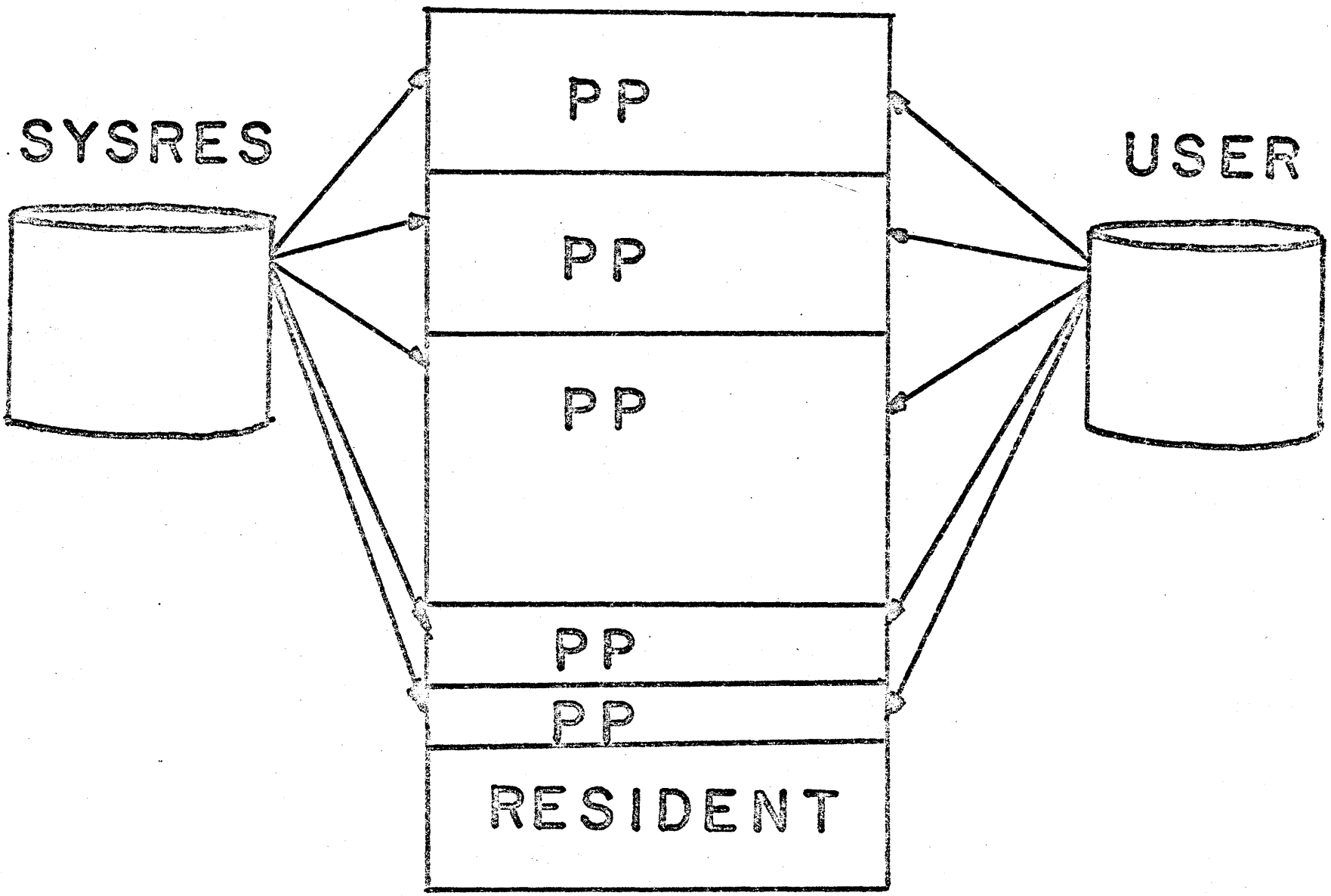
LINKAGE EDITOR PROCESSING
MODULE LINKAGE



OS LINKAGE EDITOR

LOAD MODULES

- RELOCATABLE
- EXECUTABLE
- PUT INTO PRIVATE LIBRARY
- ATTRIBUTES
  - ONLY LOADABLE
  - RE-ENTRANT
  - REUSABLE
  - SCATTER LOADABLE


SYSRES
PP
PP
PP
PP
PP
RESIDENT
USER

SESSION REPORT

COMMON - Chicago

Session Number MON C2 Session Name Use of CRT on 1800

Chairman R. W. Page

Time 1:30 PM Attendance (No.) 100

Speakers Jerry Gordan - Indiana University

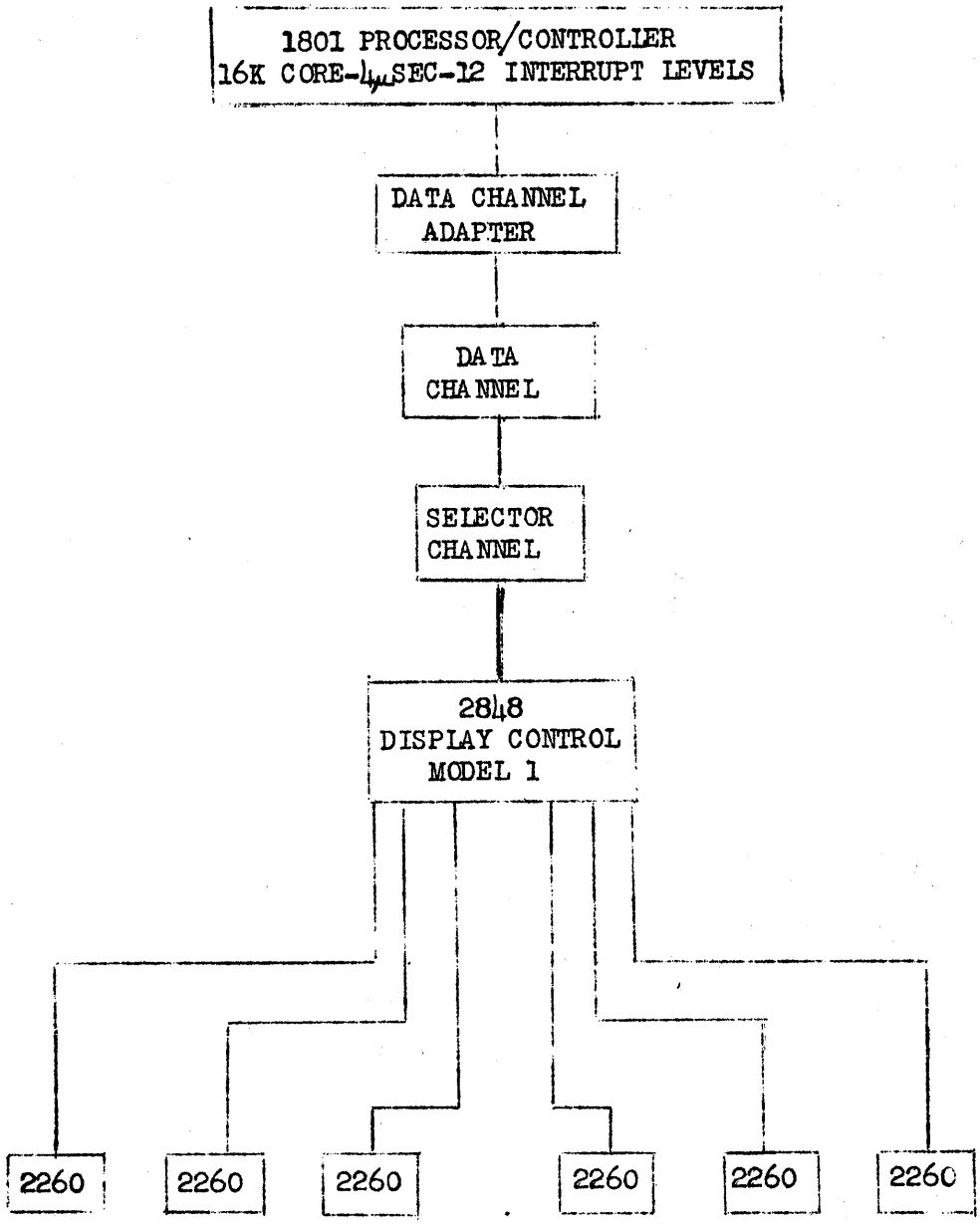
R. W. Page - New York State Electric & Gas Corp.

Synopsis of Meeting The application of 2260 Display Stations was explained. Mr. Gordon presented a specialized routine to handle limited conversation mode exchanges between various research "students" using the 2260 Display Unit. His paper also outlines in detail the CCW, CSW, IOCC and XIO formats used in 1800-2260 programming.

Mr. Page presented a generalized routine for handling 2260 on an 1800 system. This routine allows the user to fully use all the capabilities of the display units. A handout described the routine, its logic, use, and actual coding. Several slides were shown to indicate actual operation and applications.

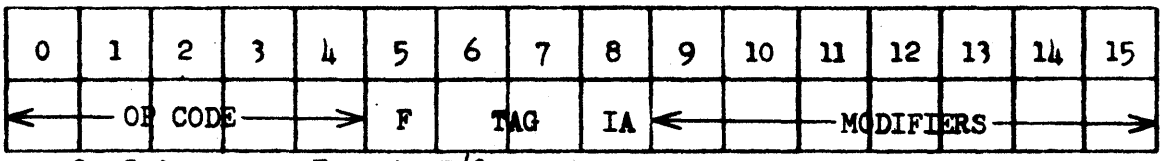
CONTENTS

Compliments of J. L. Gordon
Indiana University
Psychology Dept.
Bloomington, Indiana


1801 PROCESSOR/CONTROLLER
16K CORE-4μSEC-12 INTERRUPT LEVELS
DATA CHANNEL ADAPTER
DATA CHANNEL
SELECTOR CHANNEL
2848 DISPLAY CONTROL MODEL 1
2260
2260
2260
2260
2260
2260

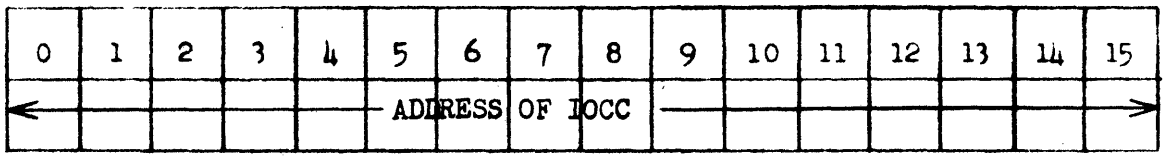
XIO AND IOCC INSTRUCTIONS

XIO (First Word)

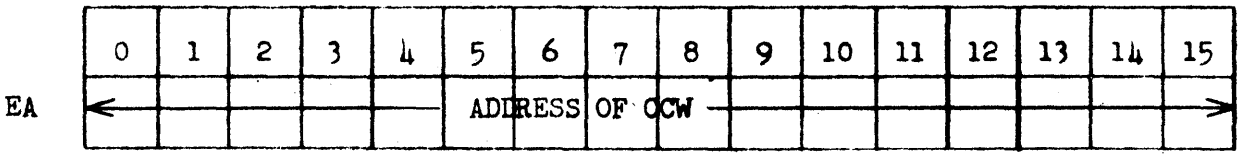


Op Code -Execute I/O
Flag -Specifies one or two word XIO instruction
Tag - Specifies register for address modification (0-3)
IA -Indirect Addressing option

XIO (Second Word)

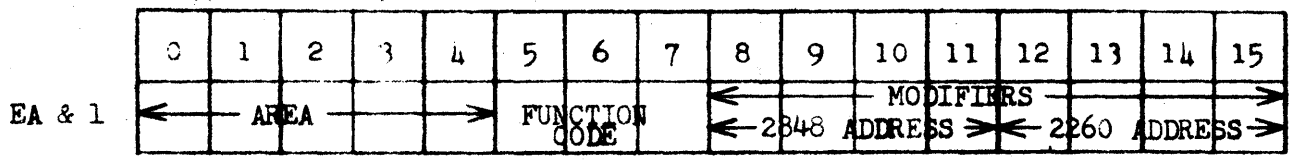


IOCC (First Word)



Address of CCW - Address of first word of CCW if the Function Code specifies Start I/0.

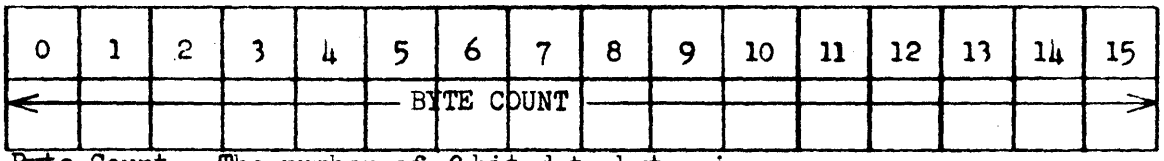
IOCC (Second Word)



Area - Selector Channel area code.
Function Code - (011) Sense interrupt level specified by modifiers.
(100) Halt I/0 on Selector Channel addressed by Area.
(101) Start I/0 on 2260 specified by modifiers.
(111) Sense Selector Channel status. Modifier bits 13 and 14 determine which of the four words to select. Bit 15 when on resets all status except status pending bit.
Modifiers- Bits 8-11 specifies 2848 address.
Bits 12-15 specifies 2260 address.

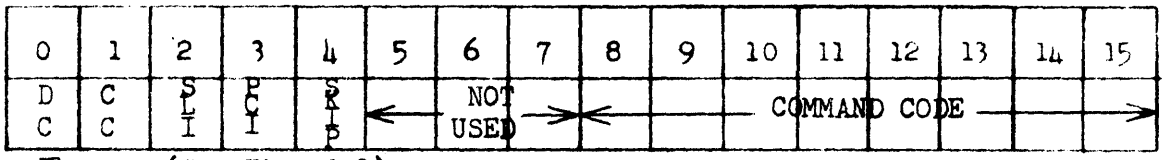
CHANNEL COMMAND WORDS (CCW)

BYTE COUNT (First Word)



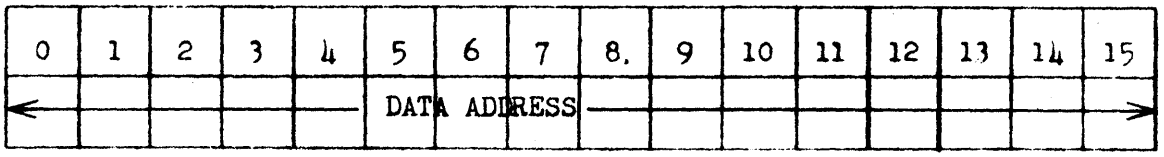
Byte Count - The number of 8 bit data bytes in I/O data area addressed in the third word of CCW.

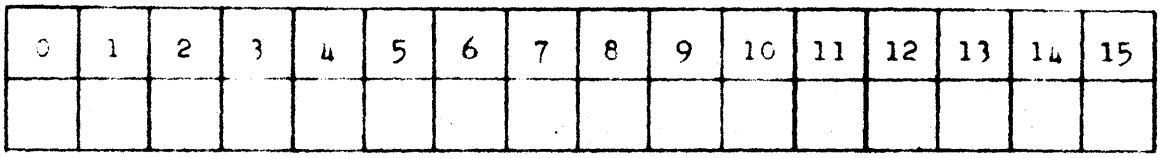
FLAGS AND COMMAND CODE (Second Word)



Flags - (See Fig. 3-2)
Command Code - (See Fig. 3-2)

I/O DATA AREA ADDRESS (Third Word)





FLAGS AND COMMAND CODE WORD (Second Word) of CCW

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DATA CHAINING	COMMAND CHAINING	SUPPRESS LENGTH INDICATION	PROGRAM CONTROL INTERRUPTION	SKIP	← NOT USED →			← COMMAND CODE →							
Selector Channel fetches the Data Address and Byte Count of the next CCW, ignoring the Command Code and Flags Word, when the byte count defined by the present CCW goes to zero when bit is on.	The operation specified by the command code in the next CCW initiated on normal completion of current operation when bit is on.	Incorrect length indication suppressed if bit is on. (See Selector Channel Status Word)	Channel generates an interrupt condition upon fetching CCW when bit is on.	Suppression of data transfer to storage during a read, read-backward, or sense operation when bit is on.				TEST I/O 0 0 0 0 0 0 0 0 The 2848 addressed, as specified in the Modifier field of the second word of the IOCC, will present its status to Selector Channel. WRITE DS BUFFER STORAGE 0 0 0 0 0 0 0 1 A Write operation is begun to the 2260 specified in the Modifier field of the IOCC. Data will be transferred from the I/O Data Area specified in the third word of the CCW until the byte count specified in the first word of the CCW is decremented to zero. WRITE DS LINE ADDRESS 0 0 0 0 0 1 0 1 A write operation is initiated to the 2260 on the line specified in the first data word of the I/O Data Area addressed in the third word of the CCW. (See Fig. 3-3). READ DS MANUAL INPUT 0 0 0 0 0 0 1 0 A Read operation is initiated to transfer manually entered messages and data from the 2260. All characters displayed between the start symbol and the cursor of the device will be transferred or until the byte count is decremented to zero.						

Fig. 3-2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
								← COMMAND CODE →							
								READ FULL DS BUFFER							
								0	0	0	0	0	1	1	0
								A Read operation is initiated to transfer all data displayed on the 2260 specified in the IOCC to the channel. Transfer continues until every available data position is transferred or until the byte count is decremented to zero.							
								NO OPERATION							
								0	0	0	0	0	0	1	1
								No data is transferred. The 2848 Control Unit responds to this command with Channel End and Device End in the Unit Address/Unit Status Word of the CSW.							
								ERASE DS BUFFER STORAGE							
								0	0	0	0	0	1	1	1
								All data displayed on the 2260 specified in the IOCC is erased and the cursor returned to the first position.							

Fig. 3-2 (cont'd)

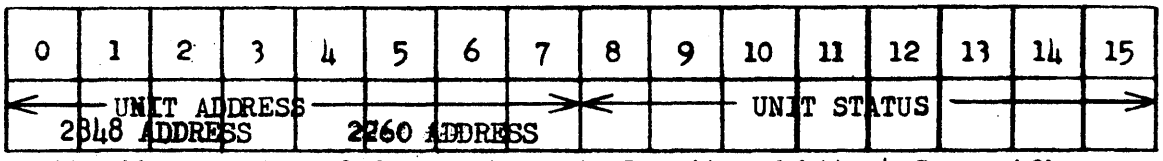
FIRST DATA WORD OF I/O DATA AREA SPECIFYING LINE ADDRESSING

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	0	0	0	0	(Line 1)							
1	1	1	1	0	0	0	1	(Line 2)							
1	1	1	1	0	0	1	0	(Line 3)							
1	1	1	1	0	0	1	1	(Line 4)							
1	1	1	1	0	1	0	0	(Line 5)							
1	1	1	1	0	1	0	1	(Line 6)							

Fig. 3-3

CHANNEL STATUS WORDS (CSW)

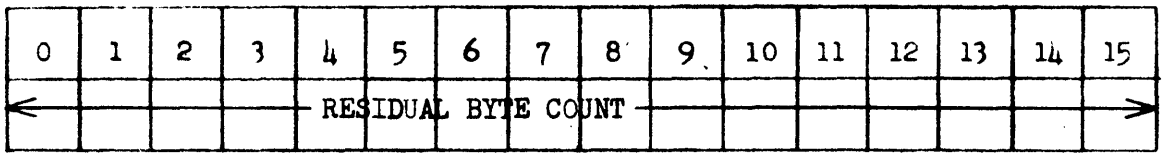
UNIT ADDRESS/UNIT STATUS (First Word)



Unit Address - Bits 0-3 specify control unit and bits 4-7 specifies 2260 in the last XIO executed. (See Fig. 7)

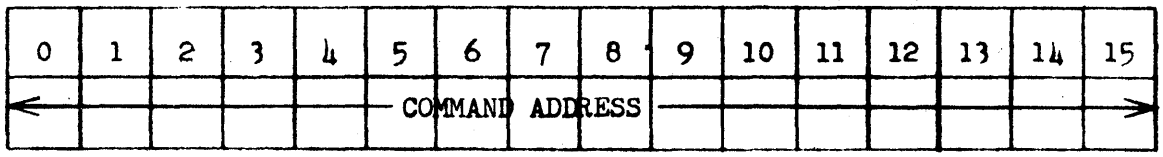
Unit Status - (See Fig. 4-2)

BYTE COUNT (Second Word)



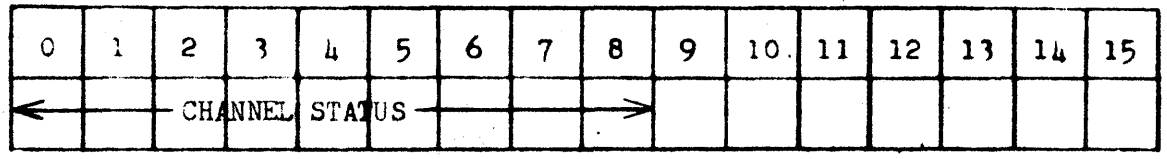
Byte Count - Residual byte count from last CCW.

COMMAND ADDRESS (Third Word)



Command Address - Specifies an address 3 higher than the last CCW used.

SELECTOR CHANNEL STATUS (Fourth Word)



Channel Status - (See Fig. 4-3)

Fig. 4-2

UNIT ADDRESS/UNIT STATUS WORD (First Word) OF CSW

15		
14	UNIT CHECK	Condition exists requiring programming investigation. Invalid command or flag, detection of a parity error in a command or data being transferred in a Read operation could cause this condition.
13	DEVICE END	This bit set at the completion of 2260 operation and indicates that it is free to perform another operation.
12	CHANNEL END	This bit set at the completion of 2848 operation.
11	BUSY	2848 in a busy status.
10	CONTROL UNIT END	Signifies that 2848 free to accept a new command.
9	STATUS MODIFIER	This bit set during a Short Control Unit Busy Sequence and indicated that 2848 busy.
8	ATTENTION	Notifies channel that an Enter key has been depressed at the 2260 specified and Manually Input data is ready to be transferred.
7	2260 ADDRESS	
6		
5		
4		
3		
2	2848 ADDRESS	
1		
0		

Fig. 4-3

SELECTOR CHANNEL STATUS WORD (Fourth Word) OF CSW

15		
14		
13		
12		
11		
10		
9		
8	UNIT OPERATIONAL	Channel executing a Start I/O operation. This bit is on until ending status is received. If this bit not on within 32μs after XIO, channel hung up.
7	ADAPTER BUSY	Channel executing a previous XIO instruction or servicing a 2260 or 2848 request.
6	INCORRECT LENGTH	Number of bytes specified in the byte count word of the CCW does not agree with the number of bytes read from or written to a 2260.
5	INTERFACE CONTROL CHECK	Malfunction of a device indicated. 2848 responded with an address other than that specified by the channel. 2 or more In-tags from devices occur simultaneously.
4	CHANNEL DATA CHECK	Channel detected parity error in data transferred or storage protect violation has occurred during a read or sense operation to data address in CCW.
3	*PROGRAM CHECK	Programming errors detected on the channel. Invalid data address specified in CCW outside the main storage. Parity error detected in the IOCW or CCW.
2	*PROGRAM CONTROL INTERRUPT	Occurs when channel fetches a CCW if the Program Control Interruption (PCI) bit is on in the CCW.
1	*UNIT STATUS PENDING	Occurs when a 2848 control unit has presented its ending, stacked, busy, or unusual condition status to the channel.
0	*NOT OPERATIONAL	2260 specified in modifier of IOCC not recognized by 2848 or 2848 specified in modifier of IOCC not on system.

* INDICATES INTERRUPT CONDITION

Fig. 5

2848 DISPLAY CONTROL SENSE WORD

15		
14		
13		
12		
11		
10		
9		
8		
7		
6		
5		
4		
3	EQUIPMENT CHECK	Buffer parity error discovered by 2848 during a Read DS Buffer operation.
2	BUS OUT CHECK	Parity error in command or incoming data byte detected by 2848.
1	INTERVENTION REQUIRED	1053 Printer not ready.
0	COMMAND REJECT	Illegal command or command with invalid flag in second word of CCW.

FIG. 6-1

INTERRUPT SERVICING SUBROUTINE

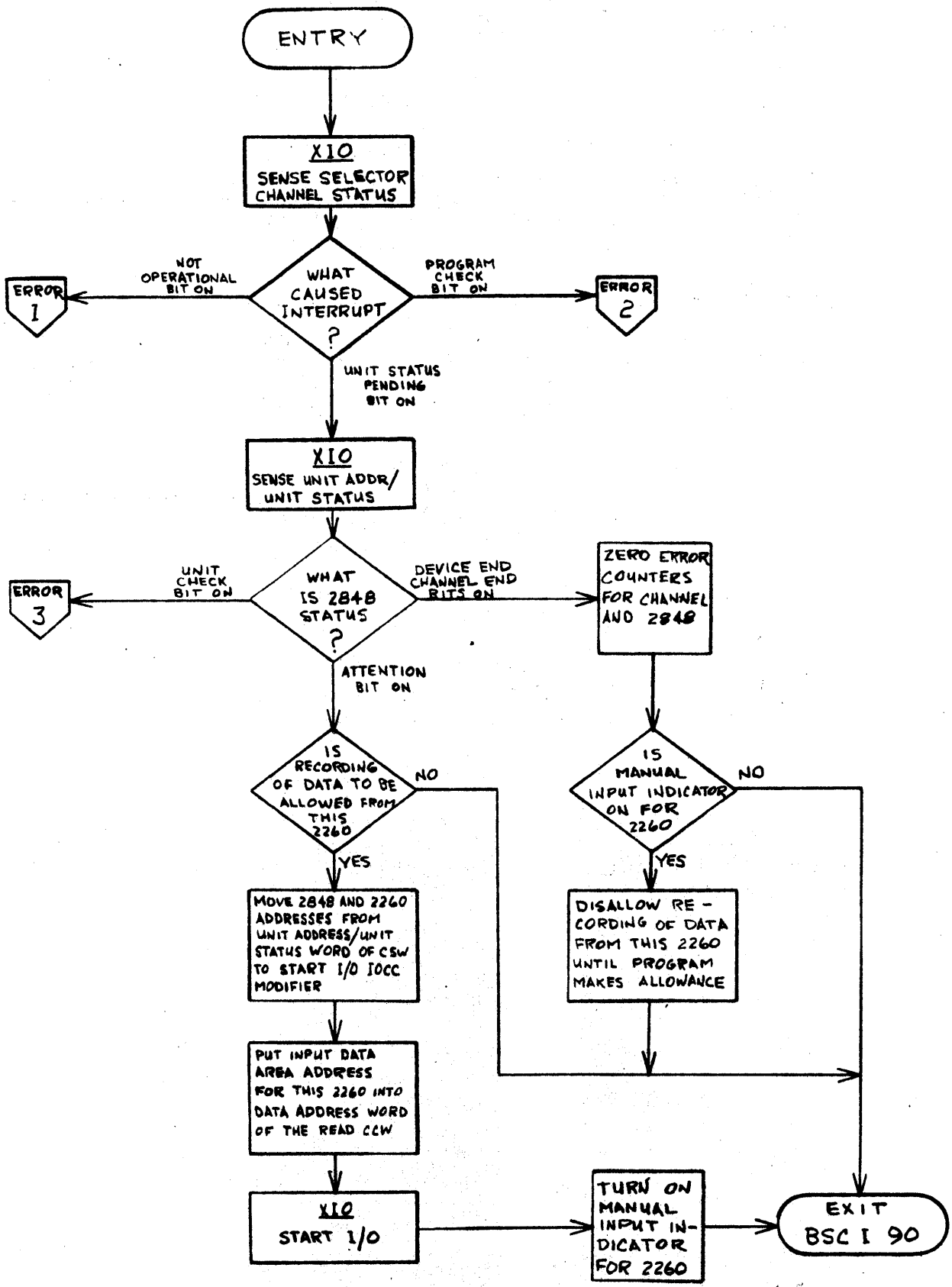
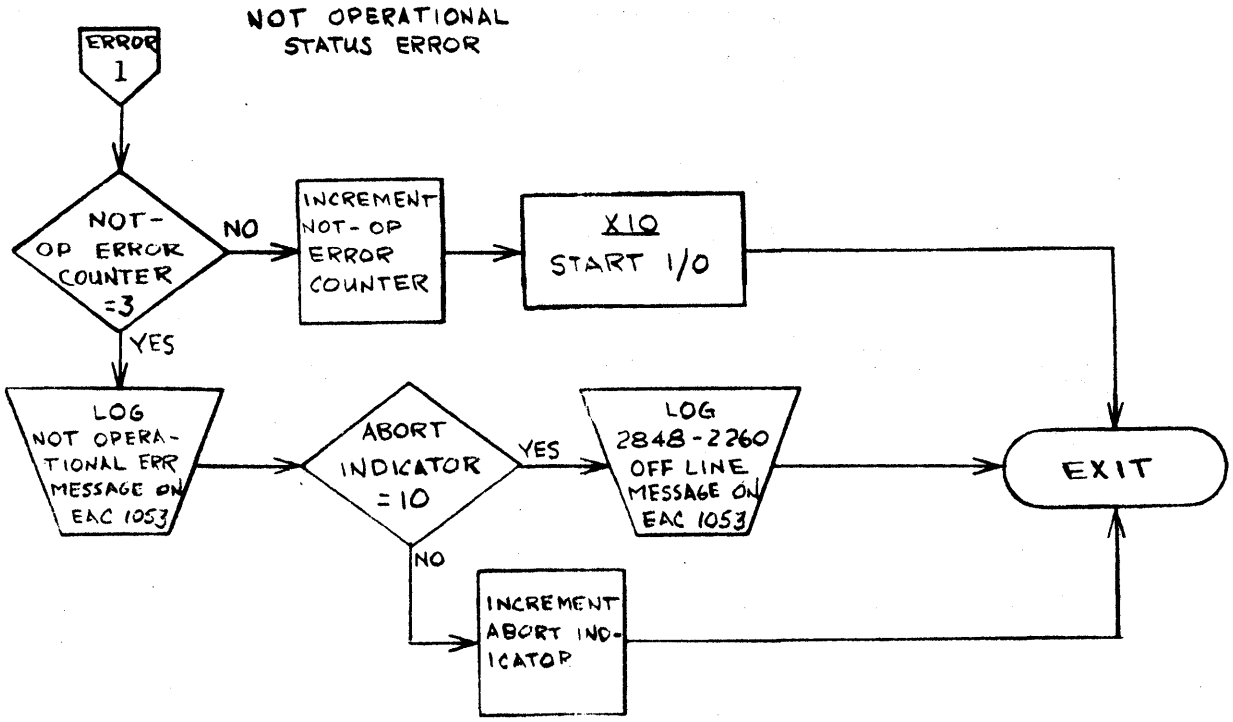


FIG. 6-2

ERROR ROUTINES



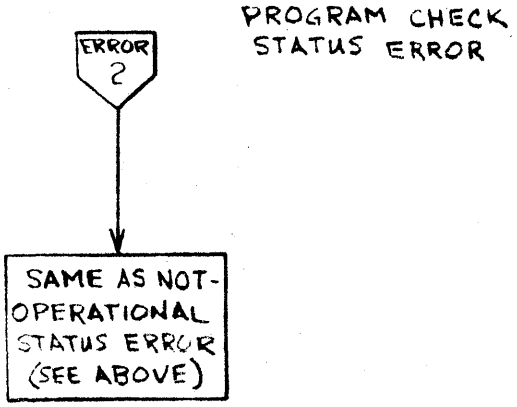


FIG. 6-2 (CONT'D)

ERROR ROUTINES (CONT'D)

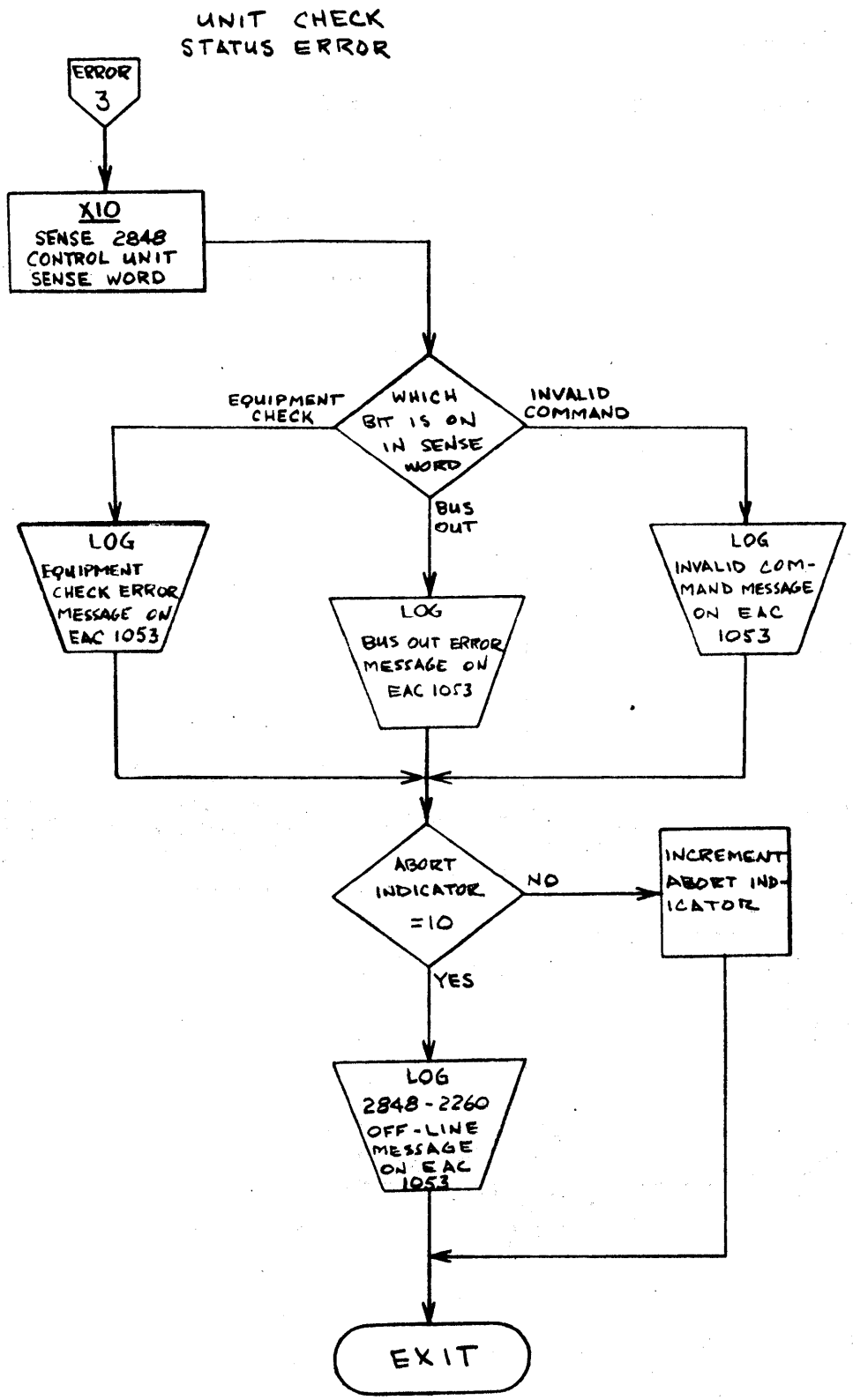


FIG. 6-3

WRITE/ERASE ROUTINE

CALL DSPLA(NOCRT, IARAY, LOA, LINNO)

CALL ERASE(NOCRT)

PARAMETERS: NOCRT - 2260 NUMBER

IARAY - OUTPUT ARRAY

LOA - LENGTH OF ARRAY

LINNO - LINE NUMBER (0 IF NO LINE ADDR)

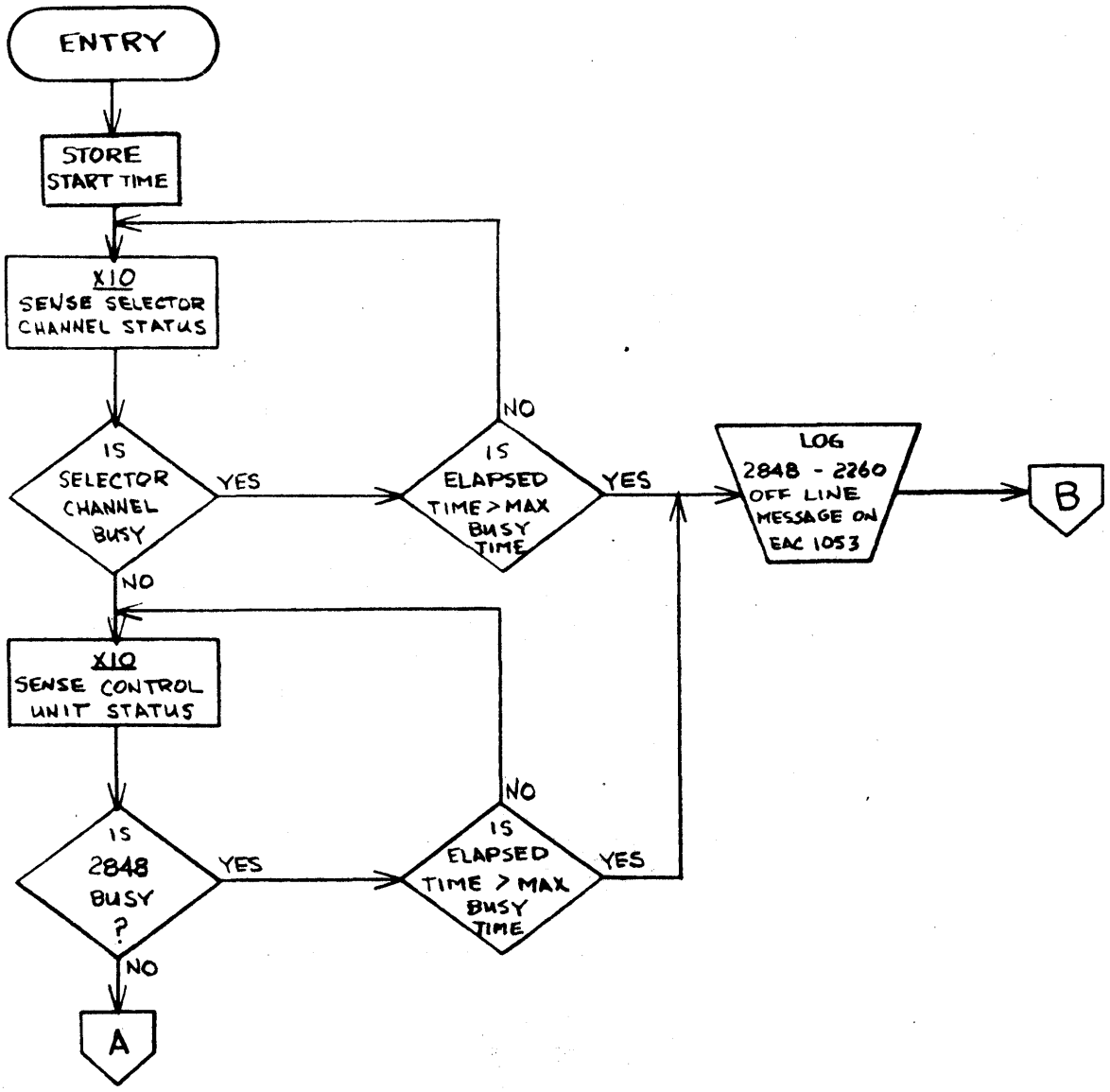
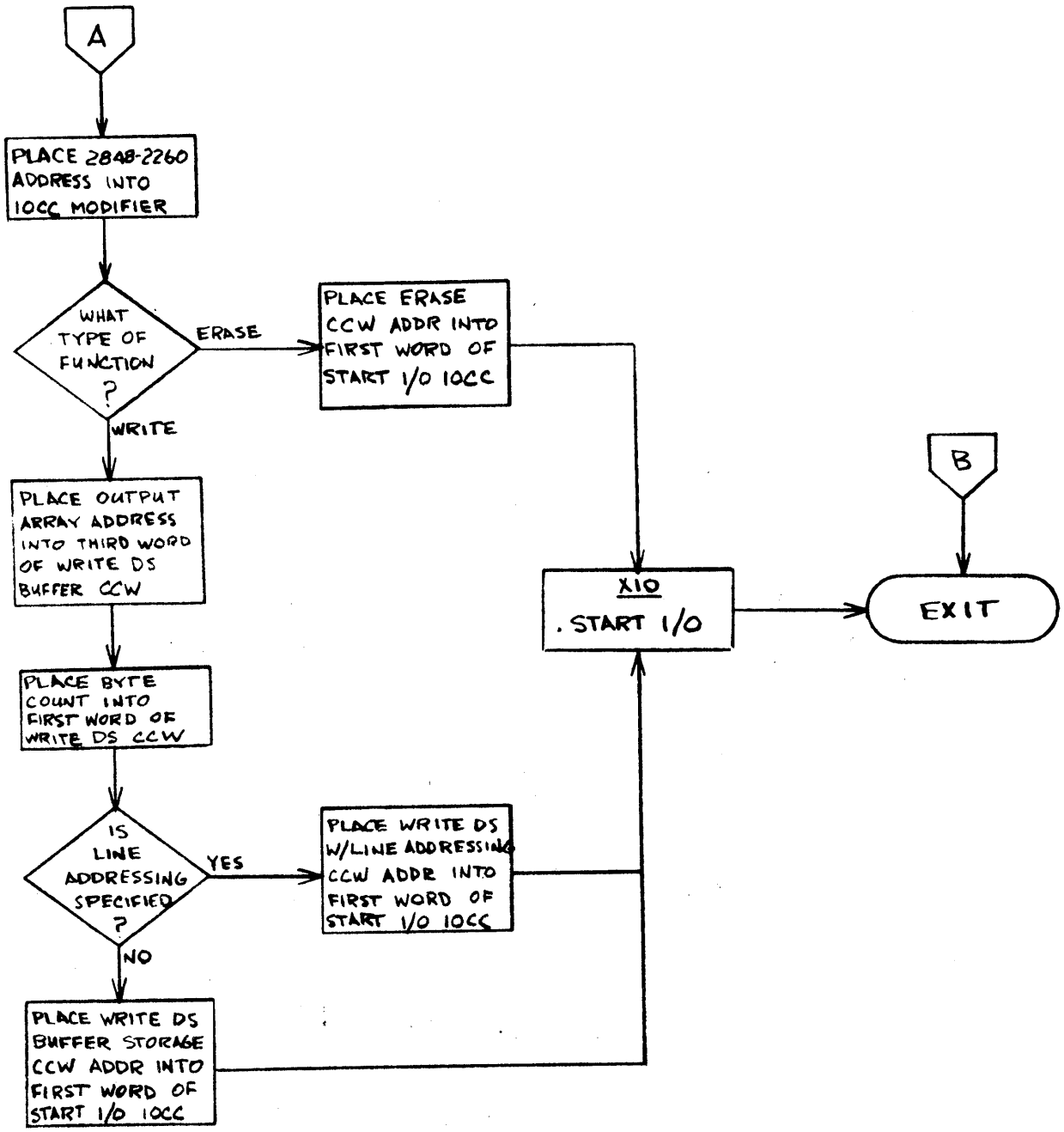


FIG 6-3 (CONT'D)

WRITE/ERASE ROUTINE (CONT'D)



2260 I/O OPERATION FOR THE IBM 1800 UNDER TSX

R. W. Page
New York State Electric & Gas Corp.
4500 Vestal Parkway East
Binghamton, New York 13902
April, 1968

DISCLAIMER STATEMENT

Although this program has been tested by its contributor, no warranty, express or implied, is made by the contributor as to the accuracy and functioning of the program and related program material, nor shall the fact of distribution constitute any such warranty, and no responsibility is assumed by the contributor in connection therewith.

2260 DOCUMENTATION

1. Disclaimer Statement

2. Table of Contents

Introduction:

The 2848/2260 I/O routine is designed to provide an easy means of handling I/O operations between 1800 programs and 2260 display terminals.

The routine was written in Assembler, resides in Skeleton, and occupies approximately 306 words plus one word in Skeleton Common.

The routine performs two major functions for the user:

1. Performs I/O operations when requested via CALL DSPLY statements in either Fortran or Assembler programs.
2. Handles attention (operator initiated) interrupts by posting in skeleton common a bit indicating which 2260 caused the attention interrupt, and then calls a user written attention handling routine by setting a program interrupt.

Reference Material

In addition to the 1800 System Reference Material the user of this 2260 I/O routine should have a working knowledge of the following material:

(1) IBM Selector Channel - Principles of Operation RPQ CO837 by J. B. Sampson and N. L. Gillette, Jr.

(2) IBM 2260 Display Station
IBM 2848 Display Control
Form A27 - 2700 - 1

Calling Sequences for DSPLY:

The 2260 I/O routine will be an INSKEL subroutine which is linked to by a standard TSX CALL statement. The CALL statement must pass either two (2) or five (5) parameters to the DSPLY routine depending on whether a test function or an I/O function is to be performed.

The Fortran CALL for a test function is:

```
CALL DSPLY (FUNCTION, RESPONSE)
```

The Assembler CALL for a test function is

```
CALL    DSPLY
DC      FUNCTION *
DC      RESPONSE *
```

* These must be address constants

The Fortran CALL for an I/O function is:

```
CALL DSPLY (FUNCTION, STATION, DATA, LENGTH, RESPONSE)
```

The Assembler CALL for an I/O function is

```
CALL    DSPLY
DC      FUNCTION *
DC      STATION *
DC      DATA *
DC      LENGTH *
DC      RESPONSE *
```

*These must be address constants

a) The FUNCTION parameter specifies what operation is to be performed by the DSPLY routine.

b) The STATION parameter must be the address of an integer 1 thru 8. This parameter specifies which 2260 the I/O operation should be performed on.

c) The DATA parameter must be the address of the left most word of the I/O area. I/O operations proceed from left.

d) The LENGTH parameter must be the address of an integer which specifies the length of the I/O area in bytes (2 bytes per word).

e) The RESPONSE parameter is the address of an integer which the DSPLY routine uses to communicate information back to the calling program.

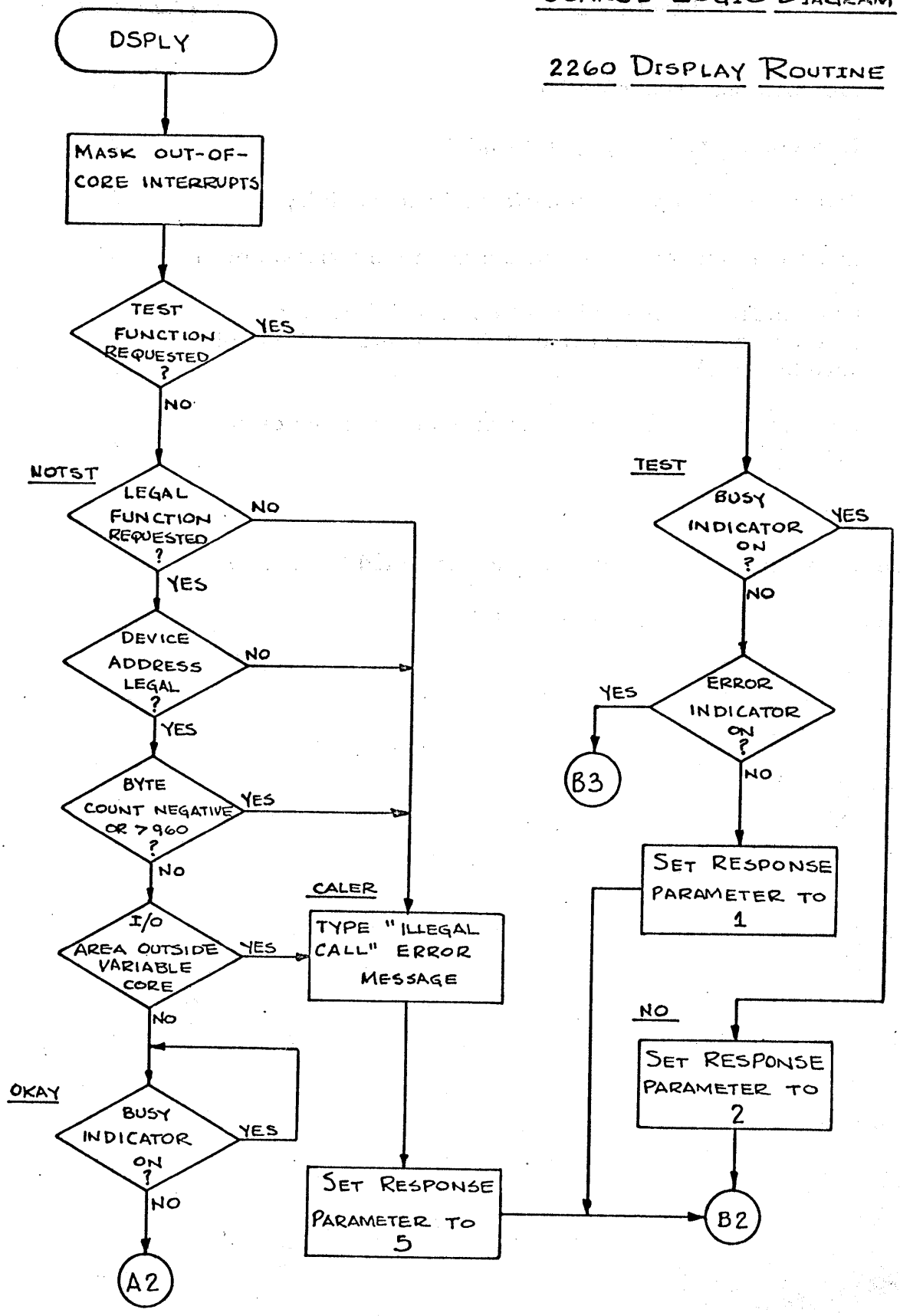
Operation	FUNCTION VALUE
Write DS buffer and wait for completion	1
Start write DS buffer	2
Write DS line address and wait for completion	3
Start write DS line address	4
Read MI buffer and wait for completion	5
Start read MI buffer	6
Read DS buffer and wait for completion	7
Start read DS buffer	8
Erase DS buffer and wait for completion	9
Start erase DS buffer	10
Test for completion of last operation	11

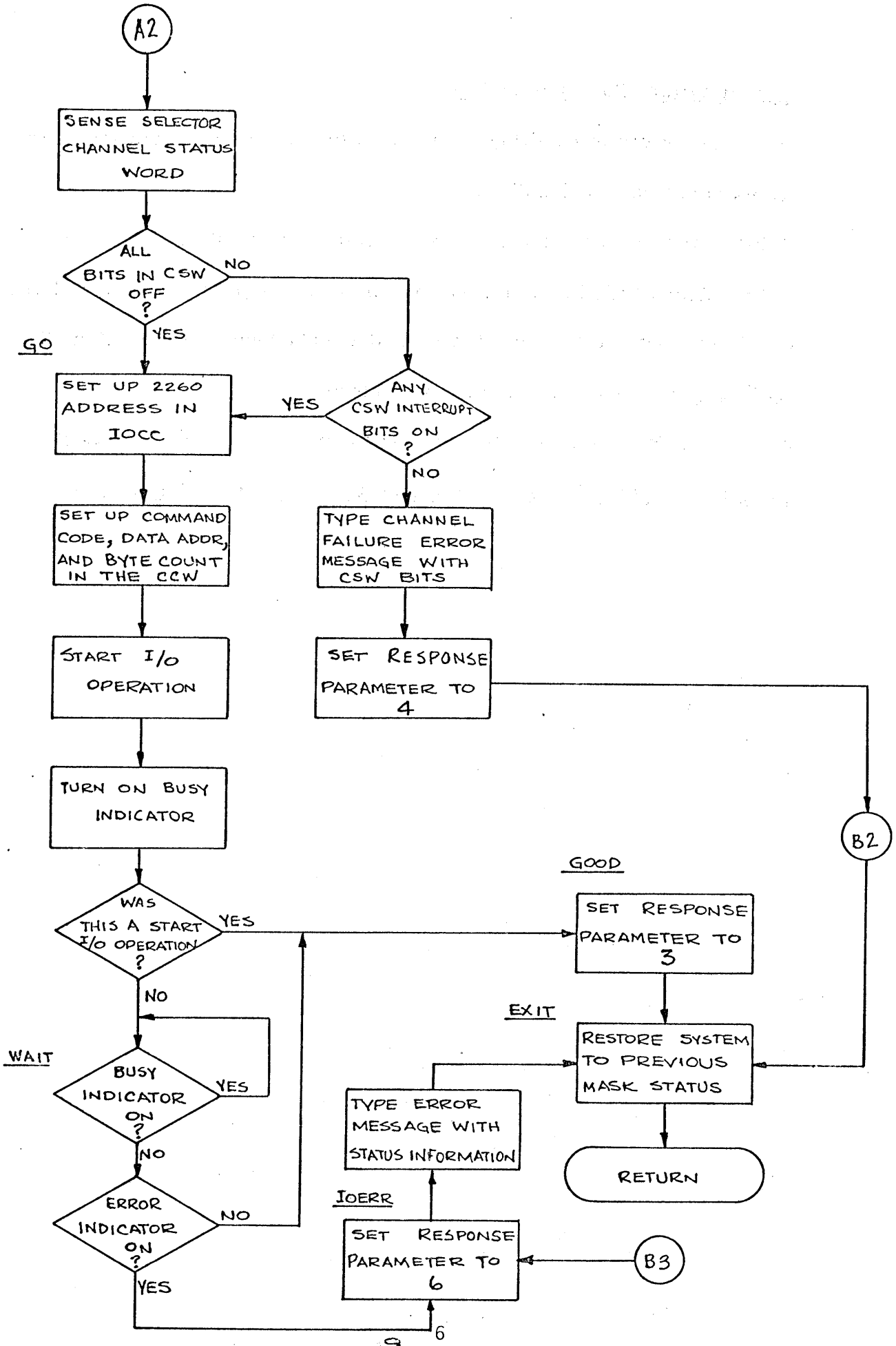
Condition	RESPONSE VALUE
Previous operation complete	1
Previous operation not yet complete	2
Requested I/O operation performed successfully	3
I/O operation cannot be started due to channel problem	4*
CALL is illegal (Invalid function, invalid station, byte count - or greater than 960 , I/O area outside of variable core)	5*
I/O operation is complete but an unrecoverable error occurred	6*

*Response codes 4, 5, and 6 are accompanied by an error message on the system printer (1816).

COARSE LOGIC DIAGRAM

2260 DISPLAY ROUTINE




A2
SENSE SELECTOR CHANNEL STATUS WORD
ALL BITS IN CSW OFF ?
NO
YES
GO
SET UP 2260 ADDRESS IN IOCC
YES
ANY CSW INTERRUPT BITS ON ?
NO
SET UP COMMAND CODE, DATA ADDR, AND BYTE COUNT IN THE CCW
TYPE CHANNEL FAILURE ERROR MESSAGE WITH CSW BITS
START I/O OPERATION
SET RESPONSE PARAMETER TO 4
TURN ON BUSY INDICATOR
B2
GOOD
WAS THIS A START I/O OPERATION ?
YES
SET RESPONSE PARAMETER TO 3
NO
EXIT
WAIT
BUSY INDICATOR ON ?
YES
RESTORE SYSTEM TO PREVIOUS MASK STATUS
NO
TYPE ERROR MESSAGE WITH STATUS INFORMATION
ERROR INDICATOR ON ?
NO
IOERR
RETURN
SET RESPONSE PARAMETER TO 6
B3
YES
6

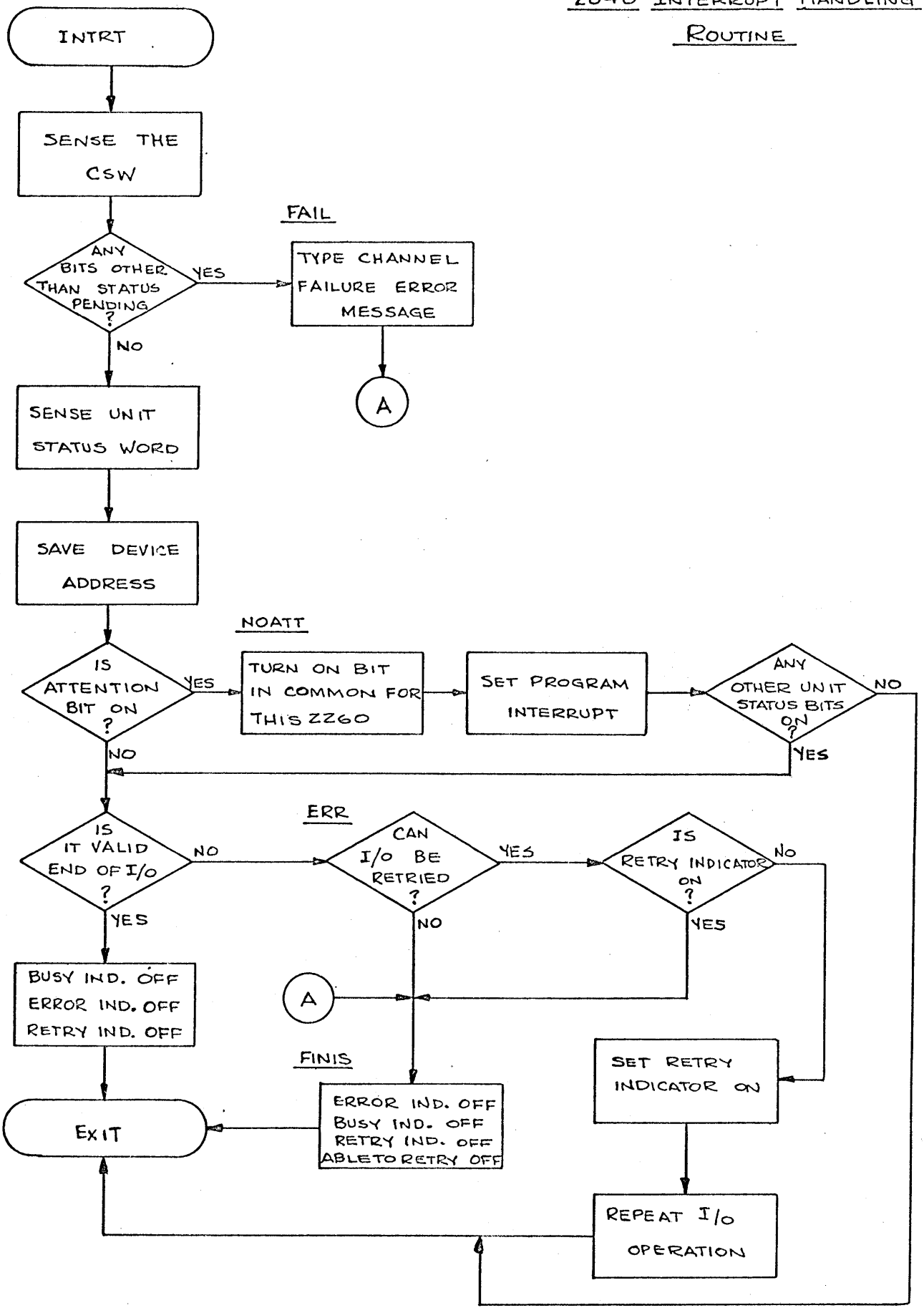
2848 Interrupt Handling Routine

The 2848 interrupt handling routine handles end of I/O operations, error retrys and attention handling.

Attention interrupts are recognized by turning on a bit in the first word (high address) of skeleton common. Bit 15 corresponds to 2260 address 0000, bit 14 to 0001, etc. If the word is 0000000001001110, then 2260's 1, 2, 3 and 6 have caused attention interrupts.

A user supplied routine is linked to when an attention interrupt is received. The routine that is called should reset the bits in skeleton common.

2848 Interrupt Handling Routine



Hardware Address Assignments

1. Selector channel area code (in IOCC) is 10010. (Area code is 18)

2. 2848 control unit address is 0001.

3. 2260 addresses run from 0000 to 0111.

4. The "first" word of skeleton common is reserved for the 2848 interrupt routine to post attention by unit. NOTE ! First word in FORTRAN common sense. (i.e. highest address or rightmost word of skeleton common)

5. The contents of core storage location 156_{10} which gives the "starting address of skeleton common" actually contains the address of the highest address word of skeleton common plus 1. (i.e. - if skeleton common runs from 7A0 to 7C0 then the contents of 156_{10} is 7C1). This is the way TSX - II is set up.

6. The program interrupt which is set on an attention interrupt is a program interrupt on level 11.

Notes on Use of 2260 Routine

1. The routine permits overlays of I/O with processing by allowing start I/O operations to be executed. A means of testing for completion is also provided. Note, however, that the I/O area is in variable core and as such must be maintained by the calling program. The calling program is responsible for insuring that the I/O area is not altered inadvertently until the I/O is complete.

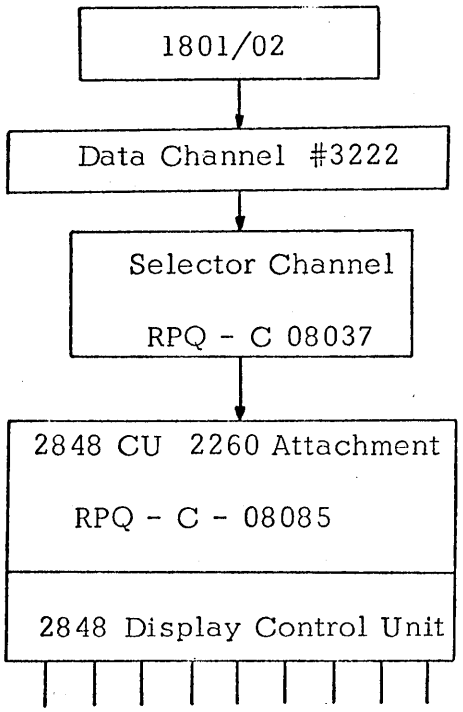
2. The routine should not be called from an interrupt level of higher priority than the selector channel.

3. The routine is not re-entrant. The routine masks out of core interrupts to eliminate the requirement for re-entrancy. The routine should not be called from an in core routine for this reason.

4. The CALL DSPLY must pass addresses as a FORTRAN CALL would.

```
CALL    DSPLY
DC      FUNC
DC      STATN       All are address constants
DC      DATA
DC      LNGTH
DC      RESP
```

MACHINE CONFIGURATION FOR 2260 I/O UNIT OPERATION



Approx. Monthly Rental - $1270 - Supports 6 2260 I/O units of 960 characters each with keyboards. Selector channel is also available to support many other system 360 I/O devices.

```
         HDNG    2260 I/O ROUTINE
*
*                2848/2260 I/O ROUTINE
*
*                 THIS ROUTINE HANDLES I/O OPERATIONS
*                TO IBM 2260 DISPLAY TERMINALS ON AN
*                1800 SYSTEM EQUIPED WITH THE RPQ
*                SELECTOR CHANNEL AND A 2848 DISPLAY
*                CONTROL UNIT.
*
*                  THE ROUTINE CONSISTS OF TWO MAJOR
*                SECTIONS.
*                  1. DSPLY ROUTINE. THIS SECTION IS
*                  LINKED TO BY A CALL STATEMENT AND
*                  IS USED TO PERFORM I/O OPERATIONS.
*                  2. INTRT ROUTINE. THIS SECTION
*                  HANDLES INTERRUPTS FROM THE SELEC-
*                  TOR CHANNEL.
*
         ISS   01 DSPLY
         DC       20          IAC FOR SELECTOR CHANNEL
         DC       INTRT       INTERRUPT ENTRY POINT
         ORG      *-2
*
*                THE FOLLOWING EQU STATEMENTS DEFINE
*                REFERENCES TO THE FIXED AREA OF CORE.
*
MSKRG    EQU      46         *IOCC FOR MASKING
OUT      EQU      108         OUT OF CORE INTERRUPTS
CORE     EQU      145         GIVES FIRST ADDRESS IN VAR
COMN     EQU      156         GIVES COMMON ADDRESS
IOXIT    EQU      90          EXIT ADDRESS FOR I/O ROUT.
X2000    DC       /2000      OP CODE FOR STORE STATUS
PI00     EQU      162        PI LEVELS 14-23
*
*                ENTRY POINT FOR A CALL DSPLY.
*
*                SAVE STATUS, ACCUMULATOR & IX REGS.,
*                MASK OUT OF CORE INTERRUPTS AND SET
*                UP FOR PARAMETER TEST.
*
DSPLY    DC       *-*         PLACE FOR PARAM. LIST ADRS
         STS   L  STATS       SAVE STATUS
         LD    L  STATS      GENERATE STORE STATUS INSTR
         OR    L  X2000
         STO   L  STATS
         STO   L  ACCUM       SAVE ACCUMULATOR
         STX   L1 SAVE1&1     SAVE INDEX REGISTER 1
         STX   L2 SAVE2&1     SAVE INDEX REGISTER 2
         LD    L  MSKRG       GET CURRENT MASK STATUS
         STO   L  RESTR       SAVE FOR LATER RESTORE
         OR    L  OUT        *TURN ON OUT-OF-CORE BITS
         STO   L  MSKRG       UPDATE CURRENT MASK STATUS
         XIO   L  MSKRG       MASK OUT OF CORE INTERRUPT
         LDX   I1 DSPLY       LOAD PARAM. LIST ADDRESS
*
*                TEST PARAMETERS FROM CALL FOR ERRORS
*
```

```
        LD    I1 0          *FUNCTION CODE TO A-REG.
        CMP      K11         COMPARE TO ELEVEN
        MDX      CALER       ERROR IF OVER 11
        MDX      NOTST       NOT TEST FUNC. IF BELOW 11
        MDX      TEST        TEST FUNC. IF EQUAL TO 11
NOTST   BSC  L   CALER,&     ERROR IF FUNC. NOT POS.
        LD    I1 1          *STATION CODE TO A-REG.
        CMP      K8          COMPARE TO EIGHT
        MDX      CALER       ERROR IF OVER 8
        MDX      *&2         OKAY IF EQUAL TO 8
        BSC  L   CALER,&     ERROR IF NOT POSITIVE
        LD    I1 3          *DATA LENGTH TO A-REG.
        CMP      K960        COMPARE TO 960
        MDX      CALER       ERROR IF OVER 960
        MDX      *&2         OKAY IF EQUAL
        BSC  L   CALER,&     ERROR IF NOT POSITIVE
        LD    L1 2          I/O AREA ADDRESS TO A-REG.
        CMP  L   CORE        COMPARE TO ETV END POINT
        MDX      OKAY        OKAY IF OVER
        NOP                  ERROR IF EQUAL OR LESS
CALER   LIBF     TYPEN       TYPE ERROR MESSAGE
        DC       /2001       CONTROL TO WRITE TO 1816
        DC       ERR1        ADDRESS OF MESSAGE
        DC       0
        LD       K5          LOAD ERROR RETURN CODE
        MDX      EXIT        GO TO EXIT ROUTINE
*
*            ROUTINE TO PERFORM TEST FUNCTION
*
TEST    LD       BUSY        LOAD BUSY INDICATOR
        BSC  L   NO,Z        BRANCH IF NOT COMPLETE
        LD       ERROR       LOAD ERROR INDICATOR
        BSC  L   IOERR,Z     BRANCH IF IN ERROR
        LD       K1          LOAD COMPLETE RETURN CODE
        MDX      EXIT        GO TO EXIT ROUTINE
NO      LD       K2          LOAD INCOMPLETE RETURN CD.
        MDX      EXIT        GO TO EXIT ROUTINE
*
*            WAIT FOR COMPLETION IF AN I/O OPERA-
*            TION IS CURRENTLY IN PROGRESS
*
OKAY    LD       BUSY        LOAD BUSY INDICATOR
        BSC  L   OKAY,Z      WAIT FOR BUSY TO GO OFF
*
*            CHECK SELECTOR CHANNEL STATUS BEFORE
*            ATTEMPTING TO START AN I/O OPERATION
*
        XIO      CSW         SENSE THE CSW
        BSC  L   GO,&-       SHOULD BE ZERO
        SLA      4           SHIFT OUT INTERRUPT BITS
        BSC  L   GO,&-       OKAY %INTERRUPT HANDLED
        LIBF     TYPEN       TYPE ERROR MESSAGE
        DC       /2001       CONTROL TO WRITE TO 1816
        DC       ERR2        ADDRESS OF MESSAGE
        DC       0
        LD       K4          LOAD ERROR RETURN CODE
        MDX      EXIT        GO TO EXIT ROUTINE
*
*            BUILD THE IOCC
```

```
*
GO      LD    I1 1          *STATION CODE TO A-REG.
        S       K1           CONVERT TO UNIT ADDRESS
        OR      BASE         ADD AREA, FUNC & 2848 ADRS
        STO     IOCC&1       STORE IN IOCC
*
*               BUILD THE CCW
*
        LD    L1 2           DATA ADDRESS FROM CALL
        STO     CCW+2       *TO CCW
        LD    I1 3          *BYTE COUNT FROM CALL
        STO     CCW         *TO CCW
        LD    I1 0          *FUNCTION TO ACCUMULATOR
        A       ADDR         ADD COMMAND TABLE ADDRESS
        STO     *&1          ADDRESS TO NEXT LD INSTR.
        LD    L *-*          COMMAND CODE FROM TABLE
        STO     CCW&1        TO CCW
*
*               START THE I/O OPERATION
*
        XIO     IOCC         START I/O
*
*               TURN ON BUSY INDICATOR
*
        LD      K1           1 TO A-REG
        STO     BUSY         BUSY ON
*
*               WAIT FOR COMPLETION IF THIS WAS NOT
*               A START FUNCTION
*
        SLA     16           MAKE A ZERO
        STO     STRT         INDICATE A WAIT FUNCTION
        LD    I1 0          *FUNCTION TO A-REG.
        BSC   L WAIT,E       EVEN FUNC INDICATES START
GOOD    LD      K3           RETURN CODE FOR SUCCESS
        MDX     EXIT         GO TO EXIT ROUTINE
WAIT    LD      BUSY         GET THE BUSY INDICATOR
        BSC   L WAIT,Z       LOOP UNTIL NOT BUSY
        LD      ERROR        GET THE ERROR INDICATOR
        BSC   L GOOD,&-      BRANCH IF NO ERRORS
*
*               UNRECOVERABLE I/O ERROR ROUTINE
*
IOERR   LIBF    TYPEN        TYPE ERROR MESSAGE
        DC      /2001        CONTROL TO WRITE TO 1816
        DC      ERR3         ADDRESS OF MESSAGE
        DC      0
        LD      K6           RETURN CODE TO A-REG
*
*               EXIT ROUTINE.
*               STORE RETURN CODE, RESTORE STATUS,
*               ACCUMULATOR, INDEX REGS. AND MASK,
*               CALULATE RETURN ADDRESS AND EXIT.
*
        STO     STRT         INDICATE START I/O ONLY
EXIT    STO   I1 4          *RETURN CODE TO CALLER
        LD    I1 0          *FUNCTION CODE TO ACCUM.
        CMP     K11          COMPARE TO 11
        MDX     ADD5         GO TO ADD 5 IF FUNCTION
```

```
        MDX       ADD5          IS NOT TEST
        LD        K2          SET UP TO ADD 2 FOR TEST
        MDX       ADD         GO TO ADD BASE
ADD5    LD        K5          SET UP TO ADD 5
ADD     A     L   DSPLY       ADD BASE
        STO   L   DSPLY       STORE RETURN ADDRESS
        LD        RESTR       GET PREVIOUS MASK STATUS
        STO   L   MSKRG       CHANGE CURRENT STATUS BACK
        XIO   L   MSKRG       RESTORE MASK STATUS
SAVE1   LDX   L1  *-*         RESTORE IX1
SAVE2   LDX   L2  *-*         RESTORE 1X2
        LD        ACCUM       RESTORE ACCUMULATOR
STATS   DC        *-*        *RESTORE STATUS
        BSC   I   DSPLY       RETURN TO CALLER
*
*               CONSTANTS AND DATA AREA FOR DSPLY
*
ACCUM   DC        0           PLACE TO SAVE ACCUMULATOR
RESTR   DC        0           PLACE TO SAVE MASK STATUS
K11     DC        11          USED TO CHECK FUNC. CODE
K8      DC        8           USED TO CHECK STATION CODE
K960    DC        960         USED TO CHECK BYTE COUNT
K5      DC        5           CONSTANT OF 5
ERROR   DC        0           PLACE FOR ERROR INDICATOR
BUSY    DC        0           PLACE FOR BUSY INDICATOR
K1      DC        1           CONSTANT OF 1
K2      DC        2           CONSTANT OF 2
K4      DC        4           CONSTANT OF 4
BASE    DC        /9500      *BASE OF IOCC 2ND WORD
ADDR    DC        TABLE-1     BASE OF COMMAND TABLE
K3      DC        3           CONSTANT OF 3
K6      DC        6           CONSTANT OF 6
        BSS   E   0           MAKE EVEN FOR IOCC S
CSW     DC        0           IOCC TO SENSE
        DC        /9701      *CHANNEL STATUS
IOCC    DC        CCW         IOCC TO PERFORM I/O
        DC        *-*          OPERATIONS
CCW     DC        *-*         CHANNEL
        DC        *-*            COMMAND
        DC        *-*               WORD
STRT    DC        0           PLACE FOR START I/O IND.
SVCSW   DC        0          *SAVE AREA FOR CSW CONTENTS
*
*               TABLE OF CCW COMMANDS FOR EACH
*               FUNCTION CODE
*
TABLE   DC        /2001       *WRITE DS BUFFER
        DC        /2001       *START WRITE DS BUFFER
        DC        /2005       *WRITE DS LINE ADDRESS
        DC        /2005       *START WRITE DS LINE ADR
        DC        /2002       *READ MI BUFFER
        DC        /2002       *START READ MI BUFFER
        DC        /2006       *READ DS BUFFER
        DC        /2006       *START READ DS BUFFER
        DC        /2007       *ERASE DS BUFFER
        DC        /2007       *START ERASE DS BUFFER
*
*               ERROR MESSAGES
*
```

```
ERR1   DC       10
       DMES     'RINVALID 2260 CALL 'E
ERR2   DC       8
       DMES     'RCHNL HDW ERROR'E
ERR3   DC       8
       DMES     'R2260 I/O ERROR'E
*****************************************************
*                                                   *
*       INTRT ROUTINE.  THIS SECTION                *
*       HANDLES INTERRUPTS FROM THE                 *
*       THE SELECTOR CHANNEL                        *
*                                                   *
*       THIS ENTRY POINT SERVICES ALL               *
*       2848/2260 INTERRUPTS                        *
*                                                   *
*       SAVE STATUS,ACCUMULATOR, AND XR 1           *
*                                                   *
*****************************************************
INTRT  STS      EXITI       SAVE STATUS
       STO      ICUM        SAVE ACCUMULATOR
       STX    1 ISAV1&1     SAVE IX 1
*
*              SENSE THE CHANNEL STATUS WORD
*
       XIO      CSW         CSW TO ACCUMULATOR
*
       STO      SVCSW      *SAVE CONTENTS OF CSW WORD
*              CHECK FOR CHANNEL FAILURE
*
       AND      BFFF        TURN OFF UNIT STATUS PEND
       BSC  L   FAIL,Z      ANY OTHER BITS IMPLY ERROR
*
*              SENSE THE UNIT STATUS WORD
*
       XIO      USW         UNIT STATUS TO ACCUMULATOR
       STO      UNIT        SAVE UNIT ADDRESS
       SLA      8           SHIFT OUT ADDRESS
       BSC  L   NOATT,-     CHECK FOR ATTENTION STATUS
*
*              TURN ON ATTENTION BIT FOR THIS 2260
*              IN SKELETON COMMON
*
       STO      USTAT       SAVE UNIT STATUS
       LD       UNIT        ADDRESS BACK TO A-REG.
       SLA      4           SHIFT OUT 2848 ADDRESS
       SRA      12          2260 ADDRESS TO LOW ORDER
       STO      TEMP        MOVE 2260 ADDRESS
       LDX  I1  TEMP          TO INDEX REGISTER 1
       LD       K1          PUT BIT IN LO-ORDER OF A
       SLA    1             CALCULATE BIT TO BE SET ON
       STO      TEMP        SAVE IT
       LDX  I1  COMN        ADDRESS OF COMMON TO IX1
       LD     1 -1          ATT. WORD FROM COMMON
       OR       TEMP        TURN ON BIT FOR THIS 2260
       STO    1 -1          STORE UPDATED ATT. WORD
*
*              SET PROGRAM INTERRUPT - LEVEL 17
*
       LD   L   PI00
```

```
         OR        PINTR
         STO   L   PI00
*
*                  CHECK FOR ANY OTHER STATUS BITS
*
         LD        USTAT          RELOAD STATUS BITS
         SLA       1              SHIFT OUT ATTENTION BIT
         BSC   L   EXITI,+-       GO TO EXIT IF ATT. ONLY
         MDX       NOATT&1        GO TO HANDLE OTHER BITS
*
*                  DETERMINE IF NORMAL END OF I/O
*
NOATT    SLA       1              SHIFT OUT ATTENTION BIT
         AND       E700           SHUT OFF CHNL END,DEV. END
         BSC   L   ERR,Z          GO TO ERROR ROUTINE
*
*                   SHUT OFF INDICATORS FOR NORMAL END
*
         SLA       16             MAKE A ZERO
         STO       BUSY           BUSY OFF
         STO       ERROR          ERROR OFF
         STO       RETRY          RETRY OFF
         LD        K1             GET A ONE
         STO       STRT           RETRYABLE OFF
         MDX       EXITI          GO TO EXIT ROUTINE
*
*                  RETRY I/O ON ERROR CONDITION IF
*                  POSSIBLE %IF IT WAS A WAIT FOR COM-
*                  PLETION FUNCTION
*
ERR      LD        STRT           LOAD RETRYABLE INDICATOR
         BSC   L   FINIS,Z        BRANCH IF NO RETRY
         LD        RETRY          LOAD RETRY INDICATOR
         BSC   L   FINIS,Z        BRANCH IF ALREADY RETRIED
         LD        K1             GET A ONE
         STO       RETRY          RETRY ON
         XIO       IOCC           RETRY
         MDX       EXITI          EXIT AND WAIT FOR COMPLETE
*
*                  SET INDICATORS FOR UNRECOVERABLE
*                  I/O ERROR CONDITION
*
FINIS    LD        K1             GET A 1
         STO       ERROR          ERROR INDICATOR ON
         STO       STRT           RETRYABLE OFF
         SLA       16             MAKE A ZERO
         STO       BUSY           BUSY INDICATOR OFF
         STO       RETRY          RETRY OFF
         MDX       EXITI          GO TO EXIT ROUTINE
*
*                  TYPE ERROR MESSAGE FOR CHANNEL ERROR
*
FAIL     LIBF      TYPEN
         DC        /2001          CONTROL TO WRITE TO 1816
         DC        ERR2           ADDRESS OF MESSAGE
         DC        0
         MDX       FINIS          GO TO SET INDICATORS
*
*                  EXIT ROUTINE
```

```
*               RESTORE STATUS, ACCUMULATOR AND IX1
*
EXITI LDS       *-*         RESTORE STATUS
ISAV1 LDX   L1  *-*         RESTORE IX 1
      LD        ICUM        RESTORE ACCUMULATOR
      BSC   I   IOXIT       RETURN TO MIC
*
*               CONSTANTS AND DATA AREAS FOR
*               INTERRUPT ROUTINE
*
ICUM  DC        0           PLACE TO SAVE ACCUMULATOR
BFFF  DC        /BFFF       MASK TO TURN OFF U.S. PEND
UNIT  DC        0           PLACE TO SAVE UNIT ADDRESS
USTAT DC        0           PLACE TO SAVE UNIT STATUS
TEMP  DC        0           TEMPORARY STORAGE AREA
E700  DC        /E700       MASK FOR CHNL,DEVICE END
RETRY DC        0           RETRY INDICATOR
      BSS   E   0           MAKE EVEN FOR IOCC S
USW   DC        0           IOCC FOR
      DC        /9703     * SENSING UNIT STATUS
PINTR DC        /1000     PI ON LEVEL 17
      END
```

```
*****************************************************
*    ASSEMBLER CALLING SEQUENCE FOR 2260 I/O        *
*****************************************************
START  CALL     DSPLY
       DC       $1          WRITE DS BUF
       DC       $1
       DC       OUTPT
       DC       $16
       DC       IRES
*
       LIBF     TYPEN
       DC       /2000
       DC       ONE
       DC       0000
*
       LD       LINE5
       STO      OUTPT
*
       CALL     DSPLY
       DC       $3          WRITE LINE ADDR
       DC       $1
       DC       OUTPT
       DC       $16
       DC       IRES
*
       LIBF     TYPEN
       DC       /2000
       DC       TWO
       DC       0000
*
       CALL     DSPLY
       DC       $0          INVALID FUNCTION
       DC       $1
       DC       OUTPT
       DC       $16
       DC       IRES
*
       LIBF     TYPEN
       DC       /2000
       DC       THREE
       DC       0000
*
       EXIT
$3     DC       3
$1     DC       3
$0     DC       0
$16    DC       16
LINE5  DC       /F440
IRES   DC       0
OUTPT  EBC      .    TEST MESSAGE.
ONE    DC       1
       DMES     'R1'E
TWO    DC       1
       DMES     'R2'E
THREE  DC       1
       DMES     'R3'E
       END      START
```

```
** 'TEST OF 2260 I/O CALLING SEQUENCE FROM A FORTRAN PROGRAM
C*************************************************************************
C ***                  2260 I/O FROM FORTRAN PROGRAMS                   ***
C ***                                                                   ***
C ***                      FORTRAN CALLING SEQUENCE                     ***
C ***                 CALL DSPLY (FUN,STA,DATA,LENGTH,RESPONSE)         ***
C ***          WHERE   FUN = FUNCTION TO BE PERFORMED                   ***
C ***                    1 = WRITE SCREEN                               ***
C ***                    3 = WRITE LINE ADDRESS                         ***
C ***                    7 = READ DISPLAY SCREEN                        ***
C ***                    9 = ERASE DISPLAY SCREEN                       ***
C ***                  STA = 2260 PHYSICAL UNIT NUMBER                  ***
C ***                    STA. NUMBERS FROM 1 THRU 8 ARE VALID           ***
C ***                  DATA = I/O AREA (LEFT MOST BYTE OR WORD)         ***
C ***                    IF I/O AREA REFERENCES AN ARRAY, THE DATA VARIABLE ***
C ***                    SHOULD REFER TO THE LAST ELEMENT OF THE ARRAY.  ***
C ***                    (FORTRAN ARRAYS ARE STORED BACKWARDS IN CORE)   ***
C ***                  LENGTH = NO OF BYTES IN I/O AREA (2 BYTES / WORD) ***
C ***                    TWO EBCDIC CHARACTERS / WORD                   ***
C ***                  RESPONSE = VARIABLE FOR COMMUNICATION OF INFORMATION ***
C ***                    BACK TO CALLING MAINLINE                       ***
C ***                                                                   ***
C*************************************************************************
      DIMENSION XMES1(5)
C ***  TEXT STORED BY DATA STATEMENT IN ARRAY FORM (BACKWARDS)           ***
      DATA XMES1/'E 1 ','SSAG','O ME',' 226','TEST'/
C ***                    LINE ADDRESSING PARAMETERS                     ***
C ***                IX IS THE LINE ADDRESSING PARAMETER                ***
C ***          IX FOR LINE 1 IS F0 (HEX)  (FIRST BYTE OF I/O AREA)      ***
C ***          IX FOR LINE 2 IS F1 (HEX)                                ***
      DATA IX/ZEF00/,I1/ZF000/
C *** ERASE SCREEN (FUN=9,UNIT=3,NO I/O AREA OR BYTE CT.,RESPONSE VAR = ITEST)**
    7 CALL DSPLY(9,3,0,0,ITEST)
C ***           WRITE MESSAGE ON EACH LINE OF DISPLAY SCREEN            ***
      DO 5 I=1,9
      IX=IX+256
      CALL DSPLY(3,3,IX,1,ITEST)
    5 CALL DSPLY(1,3,XMES1(5),20,ITEST)
      IX=IX-9*256
      CALL DSPLY(9,3,0,0,ITEST)
      DO 6 I=1,10
      CALL DSPLY(3,3,IX,1,ITEST)
      CALL DSPLY(1,3,I1,1,ITEST)
      IX=IX+256
    6 I1=I1+256
C ***  PAUSE - FOR OPERATOR ENTRY OF  DATA THRU 2260 KEYBOARD           ***
      PAUSE 2222
C ***           TEST SENSE SWITCH TO REPEAT TEST PROGRAM                ***
      CALL SSWTCH(1,J)
      GO TO (8,9),J
C ***   READ DISPLAY SCREEN - READ 20 CHARACTERS AND STORE IN ARRAY XMES1 ***
    8 CALL DSPLY(7,3,XMES1(5),20,ITEST)
      IX = IX -2560
      I1 = I1 - 2560
      GO TO 7
    9 CALL EXIT
      END
```

SESSION REPORT

COMMON - Chicago

Session Number MON C3 Session Name 1620 Project

Chairman H. B. Kerr

Time 1:30 to 3:00 PM Attendance (No.) 50

Speakers Charles Weingart

Paul McCollum

Synopsis of Meeting Well attended. Good presentations.

AN EXPERIMENTAL IBM 1620/DONNER 10-20 HYBRID SYSTEM FOR ENGINEERING EDUCATION

by

Prof. Paul A. McCollum

Dr. Jack M. Walden

Oklahoma State University
School of Electrical Engineering
Stillwater, Oklahoma

presented at the

COMMON meeting
The Users Group for Small IBM Computers
Chicago, Illinois
April 8, 1968

ABSTRACT

A hybrid computing and control system has assembled by interconnecting an IBM digital computer with a Donner 10/20 analog computer. The 1620 is equipped with an additional RPQ input/output channel, and a logic interface was designed and built locally for the purpose of coupling the I/O channel to a 9 channel A/D converter and a 5 channel D/A converter. Other features include programmable control and interrupt lines.

Use of the equipment is being introduced at the fourth and fifth year levels in the Electrical Engineering curriculum. The situlation of sampled-data control systems, and the hybrid solutions of problems making use of iterative techniques, are representative of the types of problems currently being solved on the system.

Introduction

A new facility, recently added to the 1620 computing system at the Oklahoma State University, is making possible the introduction of realistic problems and experiments in digital control and hybrid computation into certain courses in engineering. The project has been promoted by the School of Electrical Engineering, and the results are being introduced at the fourth and fifth year levels. It was never intended that this "hybrid" computer should be able to compete with a regular hybrid computing system. However, it does provide a low-cost system with which students and faculty can experiment and try out various ideas.

The Engineering Computing Laboratory at OSU is a facility maintained separately from the University Computing Center. The 1620 was purchased in 1962, and most of the digital equipment in the lab was obtained through the aid of matching funds from the National Science Foundation. The lab is operated on an open-shop basis, and a 24 hr/day, 7 day/wk schedule is maintained. The computer is, truly, a machine for the students. Figure 1 depicts the equipment in the laboratory, and attention is directed to the hybrid addition, which is the topic of this paper.

The decision to expand the 1620 system to include basic hybrid capability was dictated by the following reasoning, 1) the machine

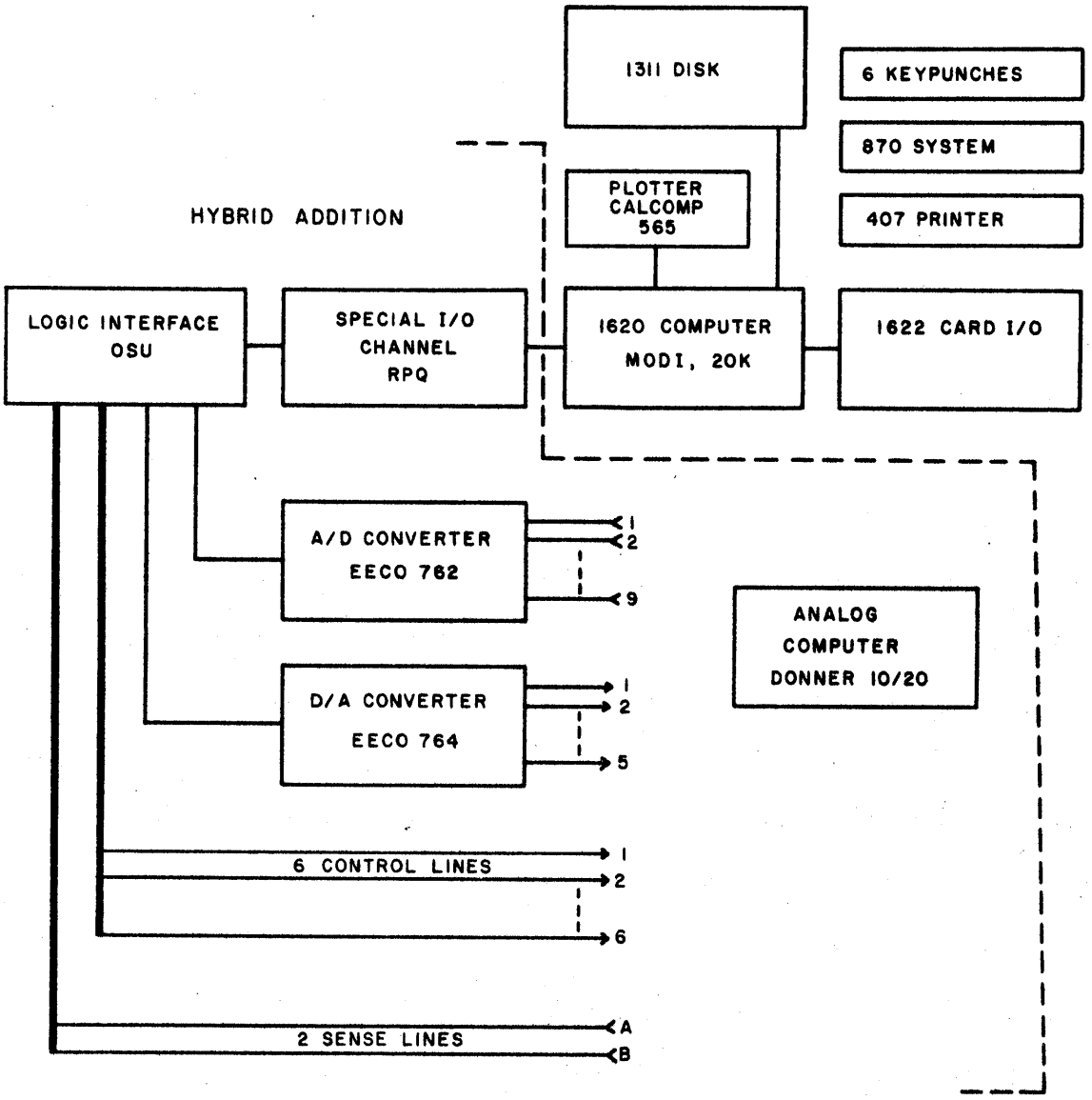


Figure 1. Block diagram of OSU computing system.

is owned by the school, 2) for teaching basic digital control and hybrid computation, a fast system is not necessary, since problems can be scaled to run in "slow-motion", 3) the design and construction of the logic interface would make an interesting and challenging project, and 4) the project could be undertaken at a rather low cost.

Now that the facility is at a usable stage of development, it can be said that the results of applications, thus far, have been most gratifying.

The special I/O channel, pictured in Figure 1, was purchased from IBM. This device provides for the input and output of BCD digits. In addition, it provides several other features, such as extra control functions and branch indicator functions. These features will be discussed later in more detail.

The 1620 special I/O device emits and accepts BCD data, serial by character, parallel by bit. Both of the external data converters handle BCD data parallel by character, parallel by bits. Thus, the OSU interface provides for the translation and buffering of the data as well as generating parity bits and necessary synchronizing control signals.

The Donner 10/20 analog computer is an ideal machine for the application at hand. It is a 100 volt, solid-state machine, with several features attractive to a hybrid project of this nature. These will be discussed further along.

The Special Input-output Channel

The Special Input-output (SIO) channel is an RPQ(#W97370) for the 1620 Model 1, obtained from IBM. Actually, it is considered to be no longer available since the 1620 is now an obsolete machine. With respect to the project at OSU, the SIO was shipped in "kit form", and the local IBM customer engineer performed the assembly.

The SIO provides the 1620 with the ability to communicate with external input-output devices in three modes; READ, WRITE, and CONTROL, without changing any of the standard computer functions. The particular mode is determined by the OP code (0_1 0_2) of the machine instruction. Numerous I/O devices may be interfaced to the SIO, and a particular device is selected with the Digit-Branch register contents (Q_8 Q_9) as a "device address". In the CONTROL mode, the contents of Q_{11} are also presented to the external device for selection of various control functions to be performed. Also, two additional indicators (90 and 91) are provided, which can be "SET" by external devices and testing by the usual BI or BNI instructions. These indicators can be set at any time by pulses or levels, and are reset by testing. They will be immediately set again by a continuing level input.

Along with the previously mentioned "working" I/O facilities, there are numerous timing, signaling, and control lines. They provide the necessary synchronization of operations in what is basically an asychronous "start-stop" mode.

The A/D and D/A Converters

Since the 1620 is a BCD(8421) machine, some of the problems were alleviated by obtaining A/D and D/A converters which utilized BCD(8421) for all digital data.

The EECO(Electronic Engineering Company of California) Model 762 A/D converter was purchased with 3-digit BCD and sign in signed-magnitude representation. Digital output is ± 999 for ± 5 volt analog input. Maximum conversion rate is 33,000 samples/second. In addition, the unit provides a randomly-addressable 10-input multiplexer, with input selection by an externally supplied BCD channel number.

The EECO 764 D/A converter is a companion unit, accepting ± 999 BCD signed-magnitude digital data, and delivering ± 100 volts analog output. The analog output is delivered to any 1 of 5 output channels, randomly addressable by an externally supplied BCD channel number. Nonaddressed channels are held at their previously set output value.

Interface Requirements

The 1620 SIO delivers and accepts signals at the IBM standard C levels.

+ C = + 2.0 volts

- C = - 3.5 volts

Output lines (from SIO) are "ON" (Logic 1) with + C levels, while input lines (to SIO) and "ON" (Logic 1) with -C levels.

The EECO units utilize digital and control levels of

ON (Logic 1) = 0.0 (±0.25)V

OFF (Logic 0) = - 9.0 (±3.0)V

Thus, in addition to the logic and control functions, the interface must provide and accept appropriate voltage levels.

Logic Functions

The 1620 SIO channel provides communication with the 1620 in a manner very similiar to that required for paper (or magnetic) tape input and output. Basically, it can be described as "parallel-by-bit" - "serial-by-character". For example, once the WRITE command is in E cycles (execution), the 8-4-2-1 coding of the first digit in the output record appears simultaneously on the output data lines. This is in the conventional paper tape X-0-C-8-4-2-1 code on 7 lines. When the external unit signals that it has "received" this character, the 1620 fetches the next character in the record, which again appears in parallel on the output lines. This continues, character-by-character, until three digits have been sent which terminates the WRITE operation. This de-activates the external device, and causes entry into I-cycles for the next instruction. Similar character-by-character sequences occur during the READ operation, except that the external device signals "end-of-data", which causes a record mark to be set in core, followed by device de-activation, and entry into I-cycles for the next instruction.

Thus, the interface must transmit and accept, in serial fashion, the BCD digits of the digital data.

Contrasted to this, the EECO equipment provides and accepts digital data in a parallel-by-character mode. When conversion is complete, the A/D converter presents all 3 BCD sigits simultaneously-holding them until a new conversion is initiated. The D/A converter requires that all 3 BCD

digits be present when conversion is initiated.

Thus, two functions of the interface are apparent:

1. In the A/D READ operation, it must "commutate" the output digits of the converter data to the 1620 in sequence as the digit-by-digit READ sequence progresses.
2. In the WRITE D/A operation, it must accept and store the output digits from the 1620 as they are presented serially, and present all digits in the register in parallel to the D/A converter when the WRITE terminate signals for the conversion.

In the process of handling this procedure, the logic must accept and generate many control signals, and initiate internal operations. A few of these are:

1. Acknowledge to the SIO that a READ, WRITE, or CONTROL function, which "addresses" units connected to the interface, has been sensed, and that operation can proceed.
2. In the WRITE operation -

   (a) Accept an SIO signal that the next digit is present on the output data lines.

   (b) Initiate storage of the data in the interface.

   (c) Signal the SIO that storage is accomplished, and the next digit can be set up.

   (d) Accept the "last digit" signal from the SIO, initiate D/A conversion, and signal SIO to continue to next instruction.
3. In the READ operation -

   (a) Accept an SIO signal that READ A/D is desired, and initiate conversion.

(b) Accept SIO signal that it is ready for next digit.

(c) Set up next digit on data input lines.

(d) Signal SIO that data is set up, and to proceed.

(e) After third digit, signal SIO to terminate READ operation.

In addition to the direct A/D READ and WRITE D/A operations, which are fundamentally data handling, a number of auxiliary functions must be performed:

1. Accept and decode the SIO CONTROL outputs, which are used to control miscellaneous external devices from the 1620. Some examples are

   (a) Operate a relay.

   (b) Switch analog computer to RESET (IC), HOLD, or COMPUTE.

   (c) Stop and/or reset analog computer digital clock.

   (d) Raise or lower plotter pen.

2. Provide for setting the 1620 indicators (90 and/or 91) which are used to control the 1620 from external devices, such as,

   (a) Relay closure,

   (b) Time pulse from analog computer digital clock,

   (c) Wait-loop in 1620 while analog in COMPUTE,

   (d) initiate 1620 READ (A/D) while analog is in COMPUTE,

   (e) Initiate 1620 WRITE (D/A) while analog is in RESET (IC).

3. Block any interface action or signal outputs except when specifically addressed - a "bug" that showed up because of 1311 disk operations that were not contemplated when the SIO was designed. The fetching of the disk-control field threw the interface into complete confusion without this feature.

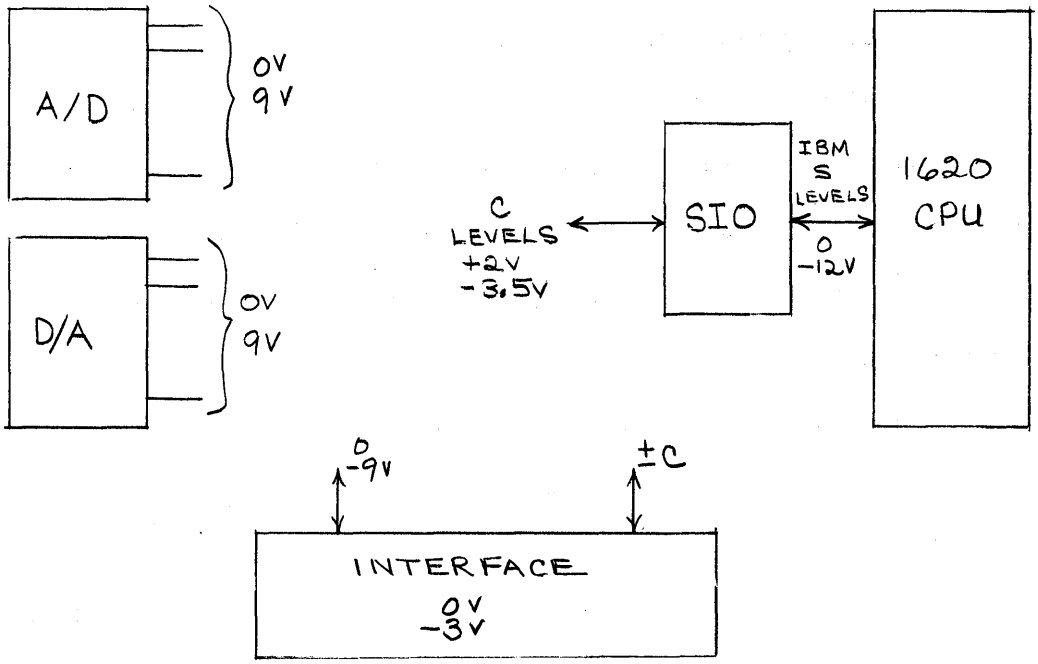


Figure 2. System data levels.

The interface unit was designed and constructed at OSU, using DEC (Digital Equipment Corporation) R-series logic module cards. Although the specified logic levels for these modules are 0 volts and -3 volts, it was found that as inputs they would accept both the IBM C levels (+2V., -3.5V.) and the EECO levels (0V., -9V.) without conversion. The DEC W602 Bipolar Output Converter was used to provide signals to drive the SIO. It was necessary to fabricate drivers locally to drive the EECO inputs. The DEC logic was very easy to work with, and few problems were encountered in obtaining desired functions with straightforward logic design techniques. The pulse actuated inputs require rise-times of the order of 40 nanosec., maximum, for reliable operation. This required some "sharpening" of SIO signals, which in most cases was adequately performed by simple inverters. In a few instances, Schmitt triggers were used for pulse shaping.

The interface was wired and tested over a 6-month period by a graduate student working part-time. The interface design follows the originally conceived organization with minor alterations. No areas where major significant improvements could be obtained with a different philosophy have be found.

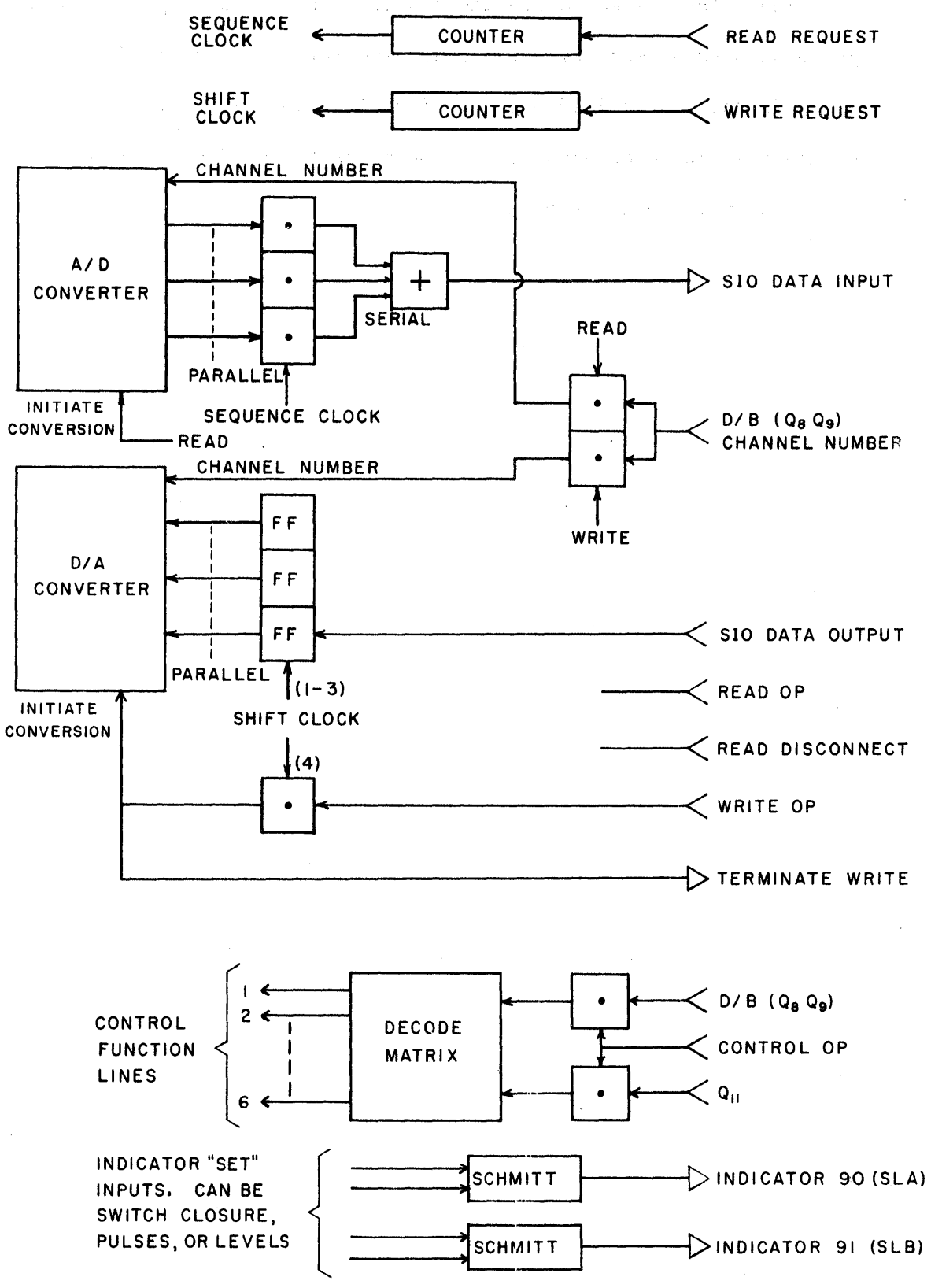


Figure 3. Simplified block diagram of the logic interface.

The Analog Computer portion of the System

The Donner 10/20 analog computer is ideally suited as the analog portion of a hybrid system such as the one being discussed. It is a solid-state machine with an operating range of ± 100 volts. The amplifiers are very stable, and the integrators are arranged for separate mode control. All computing resistors and capacitors are located inside the modules under oven control, and the setup of a problem is accomplished through the wiring of a removable problem board. The patch cords are compatable with IBM standard cords (407 type). There is room in the cabinet for nine modules of various selectable types, 24 potentiometers, and a tray underneath will accomodate 4 diode function generators. The complement of the OSU machine is shown in Figure 4.

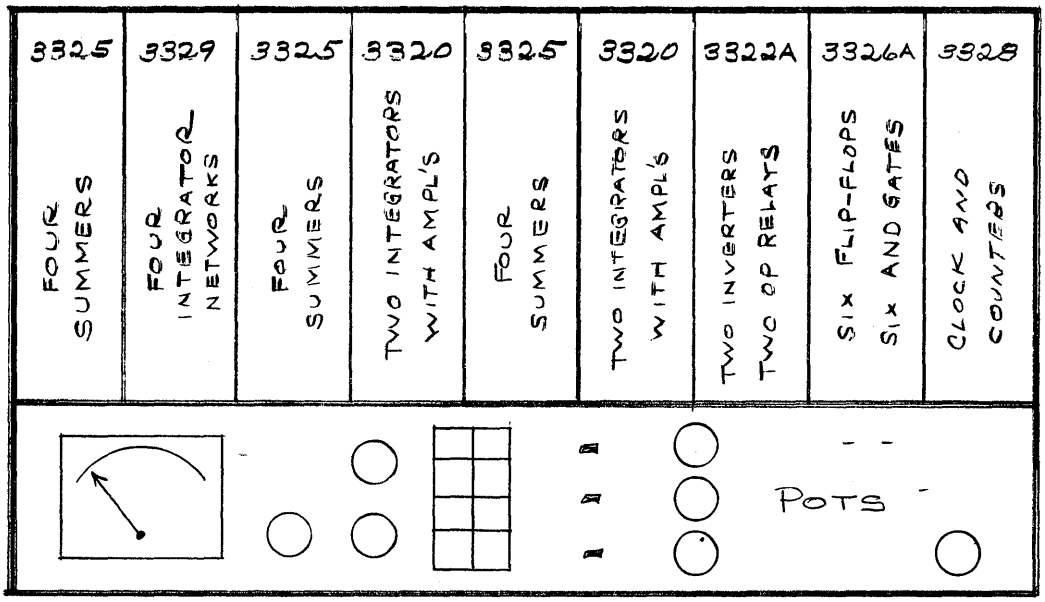


Figure 4. The analog computer module layout.

The four operational amplifiers in module no. 3325 can be used as summers, inverters or in conjunction with integrator networks. This module also contains patching terminals for a function generator,

four free diodes, potentiometers and operational switches.

Module no. 3329 contains networks and controls for four integrators. Any free amplifier may be used with these networks. This module also includes two multipliers (quarter-square), an operational relay, and terminals for more pots.

The no. 3320 integrator module contains two integrator networks and two operational amplifiers.

Module no. 3322A includes two operational amplifiers that can be used as inverters, and the facilities are included for their convienent connection as comparators. Two operational reed relays are also included as are four free diodes.

The logic module no. 3326A contains six RST flip-flops and six, three-input AND gates. Four solid-state relay drivers, capable of driving integrator mode relays, are included in the module.

The clock time event module no. 3328 contains a one-kilohertz crystal controlled clock with three decade dividers, three decade counters, and one gated relay driver. Through the use of the clock and the AND gates, time control can be accomplished in increments of 0.001 second to 100 seconds maximum.

The Hybrid Language Features

Communication with external devices, through the SIO channel, is achieved by the standard READ, WRITE, and CONTROL machine instructions. In general, this is accomplished by the addition of new "unit addresses" to the standard list.

For the READ-WRITE instructions, each command utilizes the Q_8Q_9 digits to address a particular I/O device, as follows:

01 - Typewriter

02 - Paper Tape Punch

03 - Paper Tape Reader

04 - Card Punch

05 - Card Reader

07 - 1311 Disk File

Added to this list are:

10 - A/D or D/A Channel 1

20 - A/D or D/A Channel 2

.

.

50 - A/D or D/A Channel 5

60 - A/D Channel 6

.

.

90 - A/D Channel 9

The A/D unit addresses apply only to READ instructions, and the D/A unit addresses only to WRITE instructions.

The SIO channel itself will accomodate both alpha-numeric and numeric READ-WRITE commands, with the accompanying data coded into the X-0-C-8-4-2-1 (paper tape) code by the 1620 internal code translation circuitry. However, for the A/D - D/A data, only numerics are possible, so that the

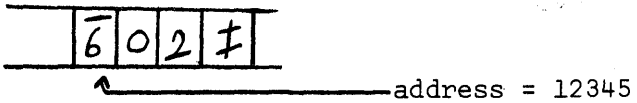
READ NUMERIC (RN, 36) and WRITE NUMERIC (WN, 38) instructions only are utilized in this application. In this mode, negative numbers are handled by the flag (X-bit) on the low-order digit.

READ - WRITE

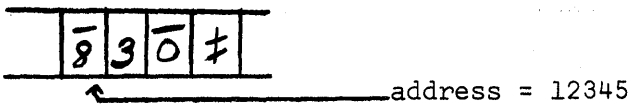
Typical machine-language instructions for A/D and D/A operations might be as follows. Consider the input instruction -

36 12345 01000 - Read A/D channel 1, and store at address 12345.

If the analog voltage on channel 1 input happened, for example, to be + 3.01 volt at the time of conversion, the stored result is:



For a - 4.15 volt input, the stored data is:



Notice that:

1. The field-definition flag in location 12345 is stored.
2. There is an inherent 2:1 scaling in the A/D converter input-output.
3. A negative number is flagged in location 12347.
4. The ‡ is set in 12348.
5. Exactly 3 digits (plus ‡) are transmitted from the A/D converter - no variation from this is possible.
6. The data is integer, with the"implied" decimal point between the first and second digits.

Now consider the output instruction ;

38 11111 03000 - Write data from address (record) 1111 on D/A Channel 3.

If the data stored is:

the analog output from D/A channel 3 is + 84.6 volts. If the stored data is:

the analog output is -54.3 volts. Notice that:

1. The field flag in 1111 is optional.
2. A negative number should be flagged in location 11113.
3. Exactly 3 digits must be in output record for proper conversion.

Because ±999 digital output from the A/D converter represents a ±5V input, and the same digital range of ±999 represents a D/A range of ±100V., there is an inherent "Scaling" of 20:1 in an A/D - D/A sequence.

CONTROL

The CONTROL instruction (K, 34) is utilized with additional "unit addresses" in Q_8Q_9 and control functions associated with Q_{11}. This instruction provides the capability for the 1620 program to exercise control over external devices. This control is not in any way directly associated with the A/D or D/A conversion - they are completely separate functions and may be utilized in any way that is appropriate.

As the SIO unit is presently wired, the Q_9 digit must be 0, and

only when it is 0 will the interface receive a correct unit address signal. Thus, the allowable Q_8Q_9 codes are 10, 20, 30, . . ., 90. The allowable Q_{11} codes are 1, 2, 3, . . ., 9.

In the interface, no particular distinction between Q_8 as a "unit-address" and Q_{11} as a "function" is maintained. The two are utilized as a two digit "function number". That is, the instructions are:

34 00000 01001 -Function 11

34 00000 02004 - Function 24

34 00000 09009 - Function 99

A conventional decoding matrix accepts the bits of Q_8 and Q_9 and output a logical 1 on the appropriate function line - all others remaining at logical 0.

In the current application, only 12 of the function circuits have been completed. They are not used directly individually, but serve to SET or RESET the six CONTROL LINES which go to external equipment such an an analog computer. The functions work like this:

Function 11 - SET CONTROL LINE 1

Function 12 - RESET CONTROL LINE 1

. . . . .

Function 23 - RESET CONTROL LINE 6

This is implemented by using the function signals to set or reset a flip-flop, whose output is the CONTROL LINE, and an accompanying relay closure.

Once a CONTROL instruction is transmitted to the interface by the SIO, the 1620 is in HOLD (clock pulses inhibited) until a signal is returned to the 1620 from the interface that the "operation" is completed. In the present use, this requires only enough time to set or reset the CONTROL LINE flip-flop. However, for instances where a Function initiates some external physical action, such as starting a 50 horsepower motor, further processing can be suspended until the action is complete, as signaled by a contact closure.

Sense Lines

External devices can control the 1620 by means of two additional indicator-codes for use with the conditional branch (BI, 46) and conditional no-branch (BNI, 47) instructions. These indicators, coded 90 and 91, can be set by the interface from switch closures, pulses, or levels, as desired. The indicators are reset by testing, unless the "set" signal is still present.

Typical 1620 instructions to utilize these indicators might be:

46 12345 09000 - Branch to 12345 if indicator 90 is SET (ON). Reset 90.

47 02402 09100 - Branch to 02402 if indicator 91 is RESET (OFF). Reset 91.

Programming

Simple procedures can be directly programmed in machine language using the appropriate instruction formats previously outlined. However, for most applications this is too tedious and time consuming to be practical. The SPS language can be utilized with its full capabilities. However, there are no direct mnemonic codes which will directly assemble the SIO instructions. This simply means that the "general" RN, WN, K, BI, and BNI mnemonics must be used. For example:

WN DATA,2000 will generate machine code 38 12345 0200.

Or, going one step further:

```
          WN DATA,DA2
             .
             .
             .
DA2       DS ,2000
```

or

```
          K  ,SETCL1
             .
             .
             .
SETCL1 DS  ,1001
```

will allow fully "symbolic" programming.

For more diverse applications, the FORTRAN language will allow for more powerful numeric processing and the use of floating-point arithmetic. A package of FORTRAN subroutine subprograms and Function subprograms is planned. These can be permanently stored on the disk file, and called into any application program. A typical list might be:

CALL GETADF(K,X) - Read A/D channel K, store at X, floating point.

CALL GETADI(K,N) - Read A/D channel K, store at N, integer.

CALL PUTDAF(K,X) - Write X, floating point, on D/A channel K.

CALL PUTDAI(K,N) - Write N, integer, on D/A channel K.

CALL SETCL(K) - Set CONTROL LINE K

CALL RSETCL(K) - Reset CONTROL LINE K.

CALL CFUNC(K) - Turn on FUNCTION K.

CALL TESTSL(K,N) - Test SENSE LINE K(90 or 91); if ON, N = 1,
if OFF, N = 0.

Function subprogram versions of these same routines would be:

```
X = READAD(K)

N = READAD(K)

Y = WRTDAF(K,X)      Y is "dummy"

Y = WRTDAI(K,N)      Y is "dummy"

Y = SETLN(K)         Y is "dummy"

Y = RSETLN(K)        Y is "dummy"

Y = FUNCTN(K)        Y is "dummy"

N = SENSL(K)
```

While expanding the digital computing capability and convenience, the use of FORTRAN may slow down computation considerably. Whether this will be a serious problem mainly depends on the particular application.

For the present system, a minimum machine-language program to READ A/D, store, increment storage address, test for top-of-core, repeat, will run at a speed of approximately 800 samples per second. A program to READ A/D, and WRITE D/A on the same data will operate at approximately 1000 samples per second. This procedure is a convenient and novel way to make a routine check of equipment operation. The input and output data waveforms (say a sine wave) can be viewed on a 2-channel oscilloscope. The preceding figures of speed are obviously far below the maximum conversion rates of the converters. They are held down by the 1620. However, for use in a simple hybrid, and model hardware control applications, it is quite adequate since the analog side can be time-scaled as necessary.

Applications of the Hybrid System

In order to show in more detail the nature of typical problems that are being demonstrated and solved on the equipment, two examples will be presented. The first problem deals with a sampled-data control system and the digital computer is used to generate a Z-tranform function for compensation purposes. The second problem utilizes an iterative technique to determine a parameter in a simple boundary-value situation. The physical system model is run on the analog computer and the digital computer is used to test the variables involved and to process the iterative scheme. The digital computer outputs successive parameter values to the analog computer until convergence is achieved.

As mentioned at the outset, the use of the equipment is being introduced at the fourth and fifth year levels in the School of Electrical Engineering. The courses involved at present are courses in control systems, electronic computers and analog and hybrid computation. The equipment is also available to graduate students who may wish to do a report or thesis relative to computer control of a physical system as well as any other hybrid computer application.

Example Problem No. 1 - Digital Compensation of Continuous System

Control theory tells us that the continuous system represented by the block diagram of Figure 5 is a "Type 0" system.

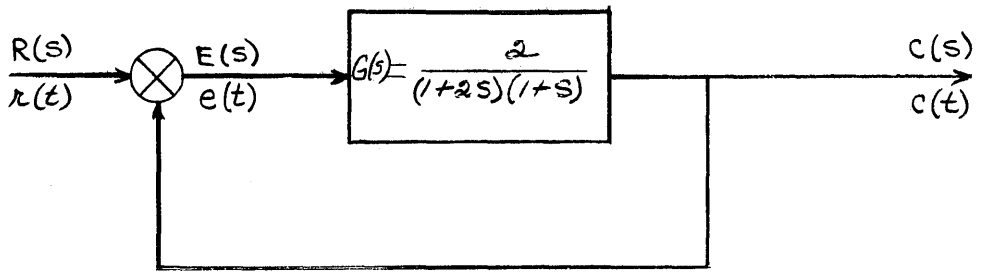


Figure 5. A Type 0 Control System

The principle characteristic of a Type 0 system is that of exhibiting a steady-state error when the input excitation is a step-function. More specifically, for a step-input, the steady-state error is

$$e_{ss}(t) = \frac{R}{1 + K_p} \qquad 1$$

K_p is the position constant or true gain of the system. For the system at hand, $K_p = 2$. If the system of Figure 5 was $r(t) = 50.U_{-1}(t)$, then

$$e_{ss}(t) = \frac{50}{1 + 2} = 16.6 \qquad 2$$

In order to study the characteristics of both analog and digital compensation, the given system was modeled on the Donner 10/20 analog computer. Figure 6 shows the analog model. 1:1 scaling was used for the amplitudes of all variables in the model and the analog model runs in real-time.

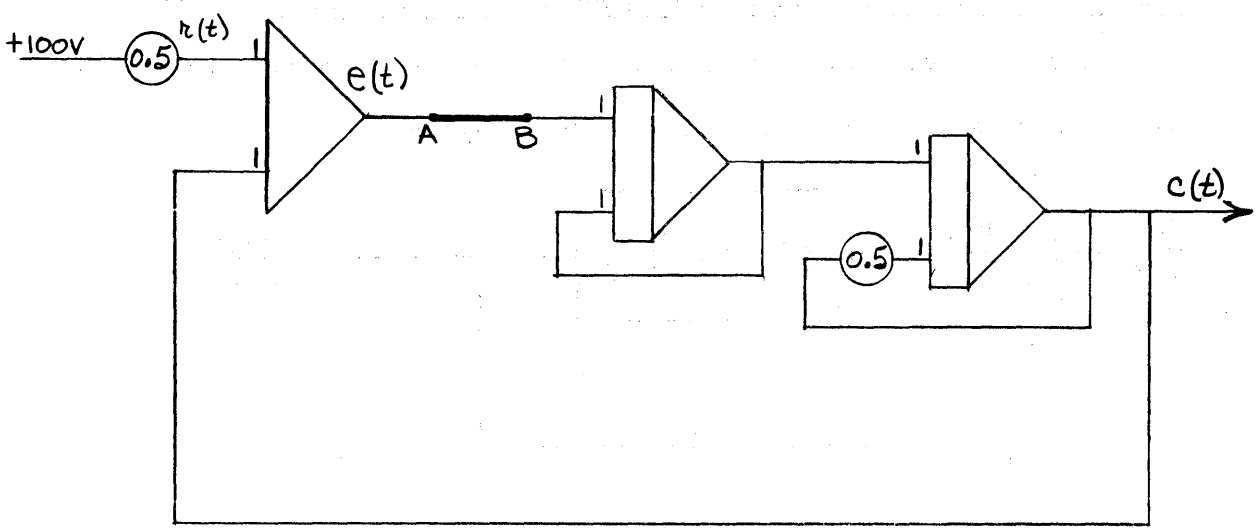


Figure 6. Analog model of a Type 0 system.

With a forcing function of $r(t) = 50.U_{-1}(t)$ volts, the response of the system, c(t), is shown as curve (1) in Figure 7. Note that there is a steady-state error of 16.6 volts as was predicted by Eq. (2). Curve (1) in Figure 8 shows the complete error response for the original system.

Again, control theory tells us that this steady-state error can be eliminated, for a step input, by raising the type classification of the system. For this system, this may be satisfactorily accomplished through the application of integral-error compensation. This was first done analog-wise by replacing the link A-B in Figure 6 by the analog model of Figure 9a which has for an output,

$$e(t) + a\int e(t).dt \qquad 3$$

The transfer function of this integral-error compensation circuit is recorded in Figure 9b.

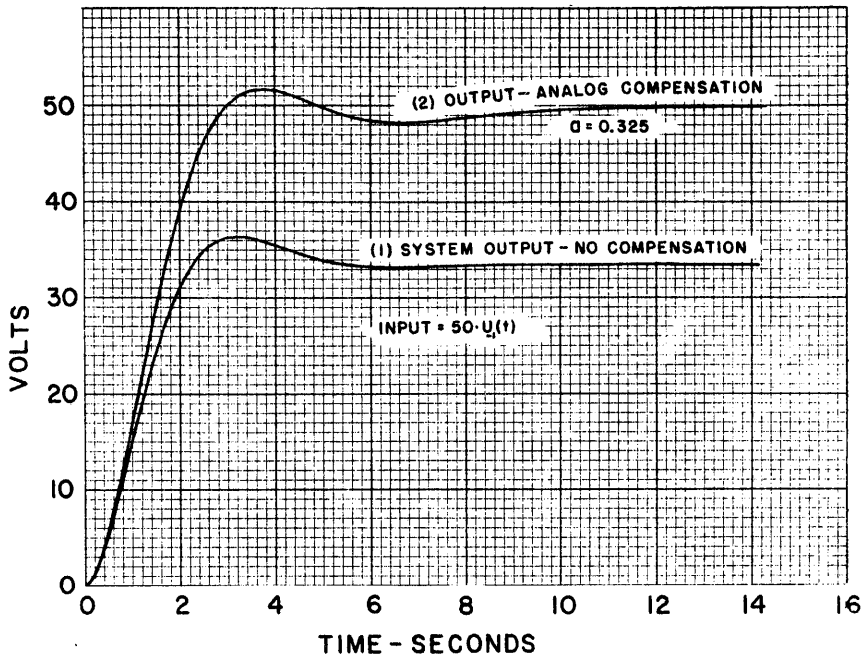


Figure 7. Analog System Output.

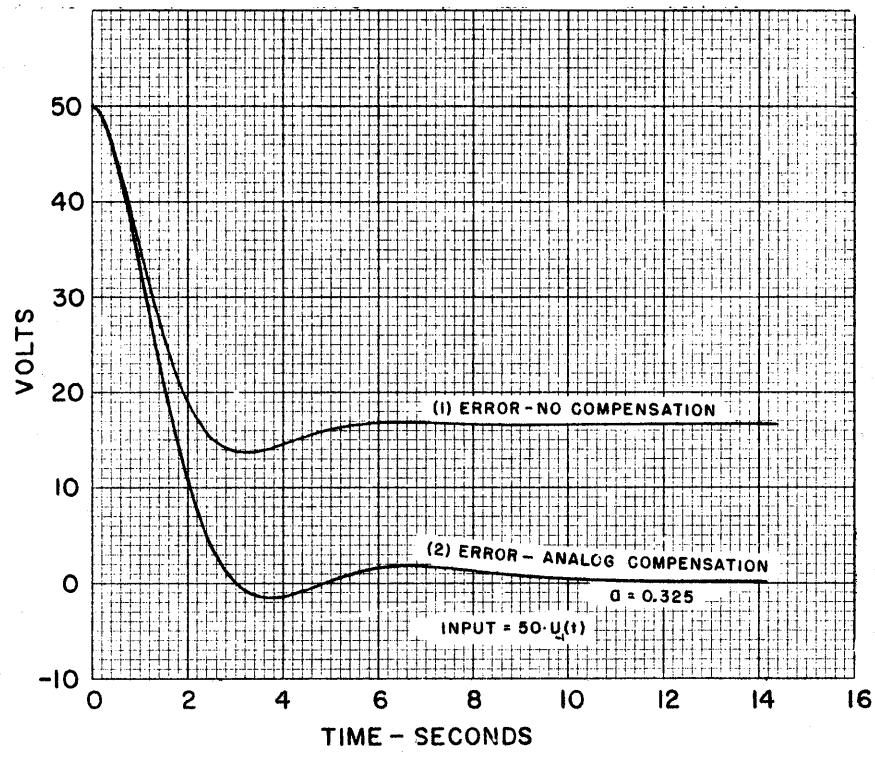


Figure 8. Analog System error response.

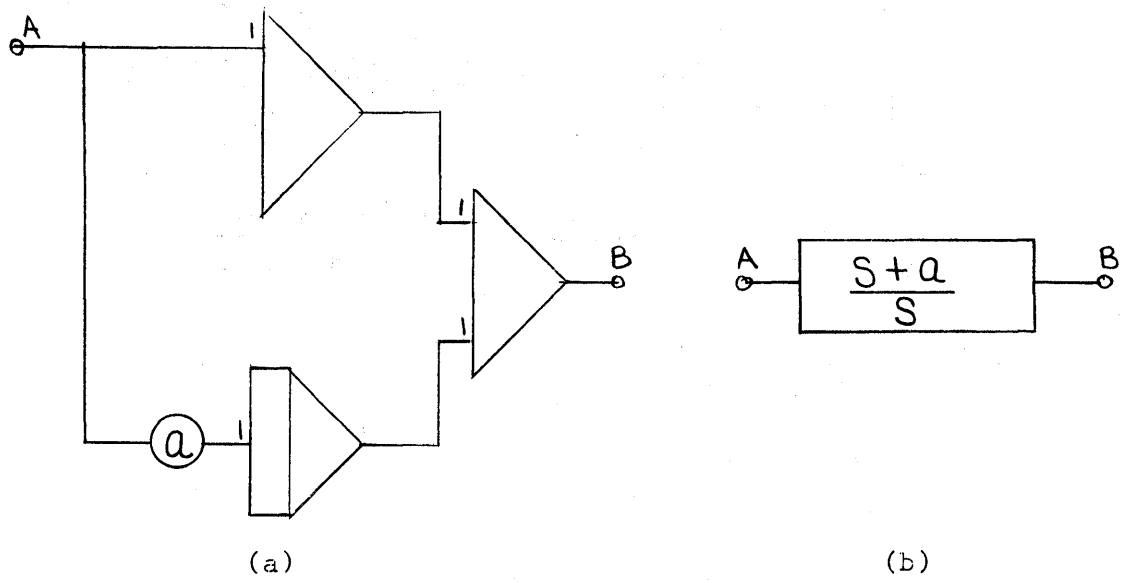


Figure 9. Analog model of integral-error compensation.

Through testing of the compensated model, the coefficient, a, was adjusted so that the transient response of the system, as evidenced by the overshoot property, was approximately the same as the original system. For a step-input of $50.U_{-1}(t)$ volts again, the output response of the compensated system is now seen as curve (2), Figure 7, and its error response is curve (2), Figure 8. These characteristics show that the objective of eliminating the steady-state error has been accomplished while retaining approximately the same nature of transient response.

Without going into an argument of analog compensation vs digital compensation, it was desired to demonstrate the use of a digital computer in the compensation of a continuous analog system. This time, the link A-B, referring to Figure 6, was replaced by the 1620 hybrid system. This new configuration is now pictured in Figure 10.

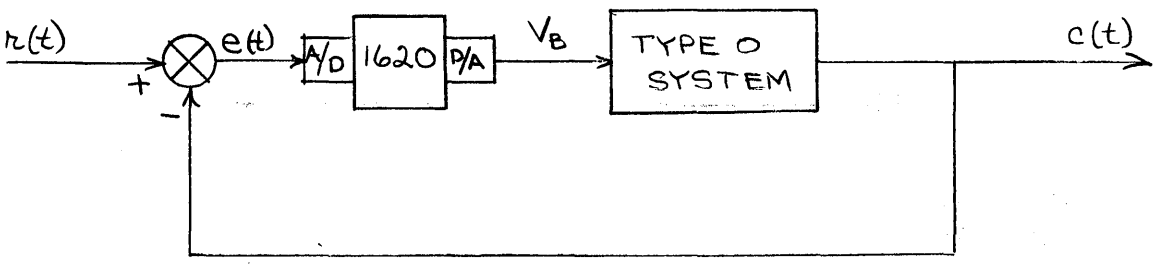


Figure 10. The analog system with the digital computer in the loop.

To accomplish integral-error compensation, the digital computer must approximate the function of Eq. (3). Due to the discrete nature of the digital data, the finite-difference function that was programmed was

$$V_B(nT) = e(nT) + aT \sum_{k=1}^{k=n-1} e(kT) \qquad (4)$$

a is the same coefficient as in (3), and T is the sample period in seconds. In this problem, T = 0.5 second.

The original program for use with this problem was written in machine language and is listed in Table 1. The sample period time, T, was regulated by adjusting the program execution time. It will be noted that this was controlled by the Transmitt-Record instruction at location 00522. This is a practical and very convienent method of time control when the program is short. To start the problem, the two computers can be started together, or the analog system can be turned on with no forcing function, the digital computer then being started, followed by the application of the system forcing function.

The output response of the system when digital compensation is used is recorded as curve (1), Figure 11. The error response of curve (3) again settles to zero in steady-state, thus indicating that the desired

Table I

Digital program for integral compensation

Instruction location			
00402	TFM	16 02000 00000	Zero the accumulator area
00414	RN	36 00900 00500	Read a.T into 900
00426	H	48 00000 00000	Pause
00438	RN	36 01004 01000	Read e(nT) on A/D channel 1
00450	M	23 01006 00903	e(nT) x aT
00462	A	21 02000 00099	Add to accumulator area
00474	MF	71 01996 02000	Move sign flag
00486	A	21 01006 01996	Add accum to output area
00498	MF	71 02000 01996	Move sign flag back
00510	WN	38 01004 02000	Write output on D/A channel 2
00522	TR	31 07750 07550	Kill time
00534	BNI	47 00438 00100	If SS1 is ON - repeat program
00546	B	49 00402 00000	Branch

Before loading the program, clear memory and set a record mark at 19999.

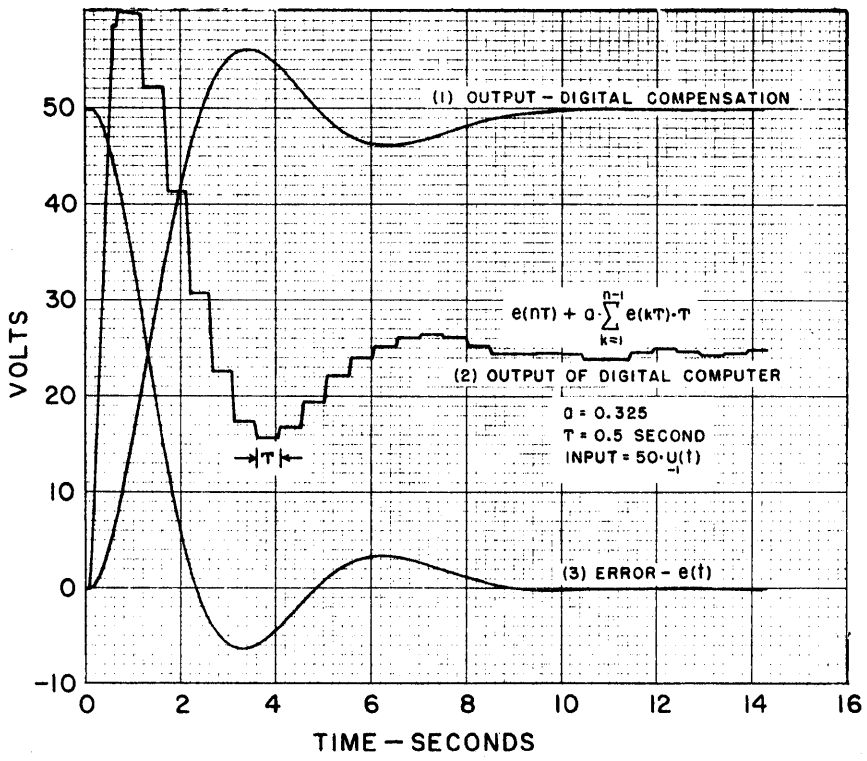


Figure 11. System response with digital compensation.

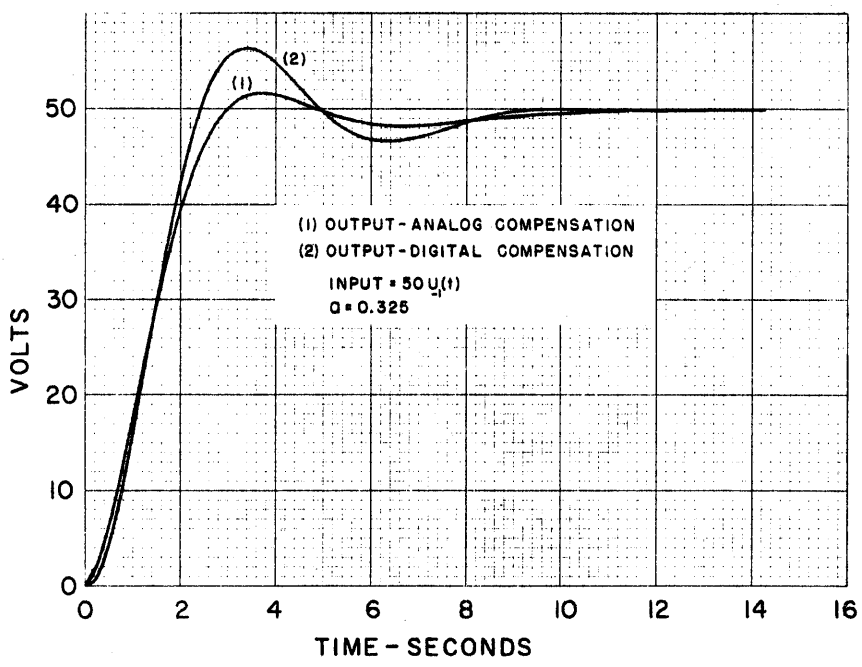


Figure 12. Comparison of system outputs.

effect has been accomplished. The same value of the coefficient, a, was used for both cases of compensation. Because of this, the output responses of the two cases are slightly different. A direct comparison of these output responses are made in Figure 12. It will be noted that the digitally compensated system has more overshoot in evidence in its transient period. For the same value of coefficient, a, this is to be expected. Control theory of sampled-data systems tells us that a zero-order data hold device has an effective lagging phase shift of $\omega T/2$ radians. Since the Type 0 system itself has a phase lag property, then the addition of more phase lag to the control loop can be shown to contribute to a more oscillatory response. By using a smaller value for a, the digital case could be made from the D/A converter. Approximately the same as the analog case. Curve (2) in Figure 11 represents the output of the digital computer. The zero-order hold property is very much in evidence in this data.

By appropriate programming, the 1620 hybrid system can be made to represent almost any conceivable Z-transform function. Realistic models of sampled-data systems can be set up on the IBM 1620/Donner 10/20 configuration. Classic cases, such as minimal prototype systems, can be modeled and studied, as well as many others.

Example Problem No. 2

A Split-boundary Value Problem

Given:

$$\frac{d^2x}{dt^2} + 4\frac{dx}{dt} + 16x = 1600.U_{-1}(t)$$

where $x(0) = 0$

and $\dot{x}(0)$ = to be determined

The problem is to determine that value of $\dot{x}$ (0) which will cause x(t) to have a desired value of 130.0 at t(time) = 0.5 second.

The problem is scaled so that time on the analog computer is 10 times slower than real time. Thus, the 0.5 second in real time becomes 5 seconds on the computer. Also, because of the magnitude scaling, the value of $S_0 \cdot x$ (t) becomes, xD = 65, on the model. The scaled analog model is pictured in Figure 13, and the timing control diagram is shown in Figure 14.

The algorithm for the solution of the problem is as follows: The starting value of $S_1\dot{x}$ (0) is set from the digital computer. Turn on the analog computer in RESET for 3 seconds. Put the analog in COMPUTE for 5 seconds. At the end of the COMPUTE period, put the analog in HOLD (3 sec.). Sample the value of $S_0x(t)$ with the A/D converter and compare to the desired value, XD = 65.0. If $S_0x(t) < XD$, then increment $S_1\dot{x}(0)$ (XDOT) by a DELTA of 10. Repeat the run. When $S_0x(t)$ becomes $>$ XD, then reset XDOT to its previous value and divide DELTA by 10. Repeat the problem procedure until $ABS(S_0x(t) - XD) \leq$ specified EPS or until DELTA = 0.1. Write out the answer for XDOT from the digital computer. The flow chart for the digital program is shown in Figure 15. The digital SPS program is listed in Table II.

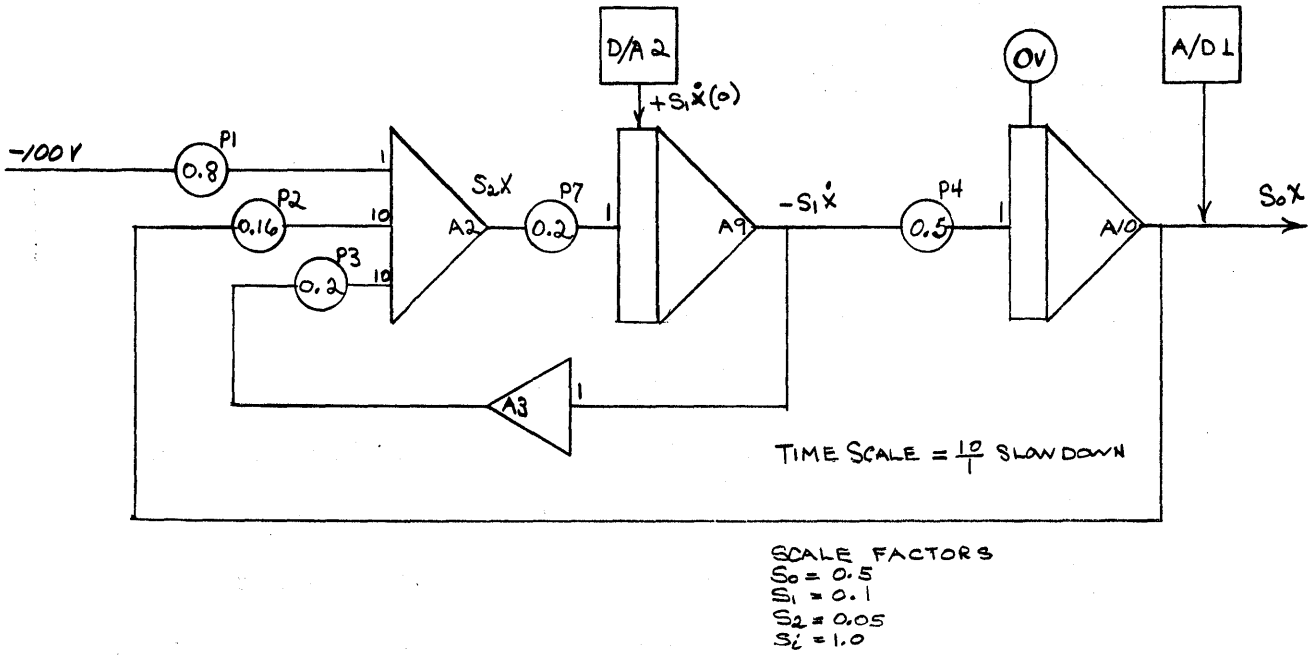


Figure 13. Analog model for problem 2.

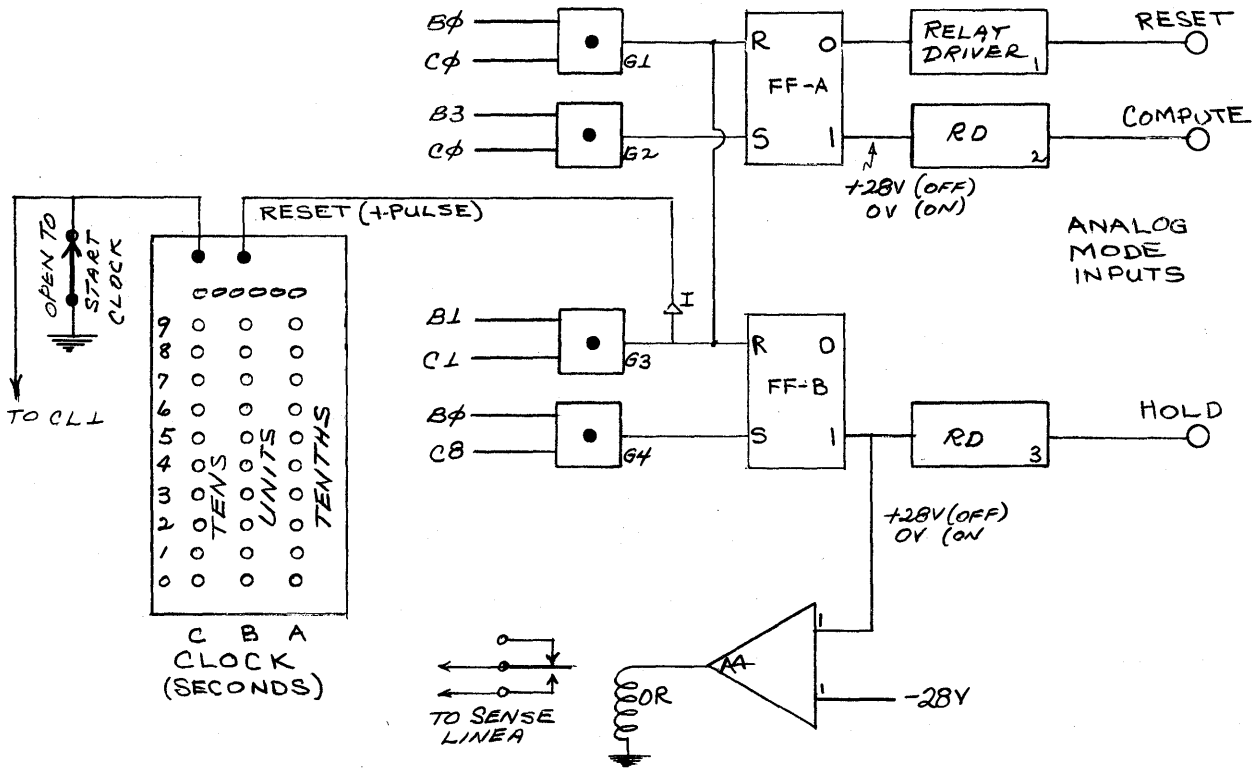


Figure 14. The timing system for problem 2.

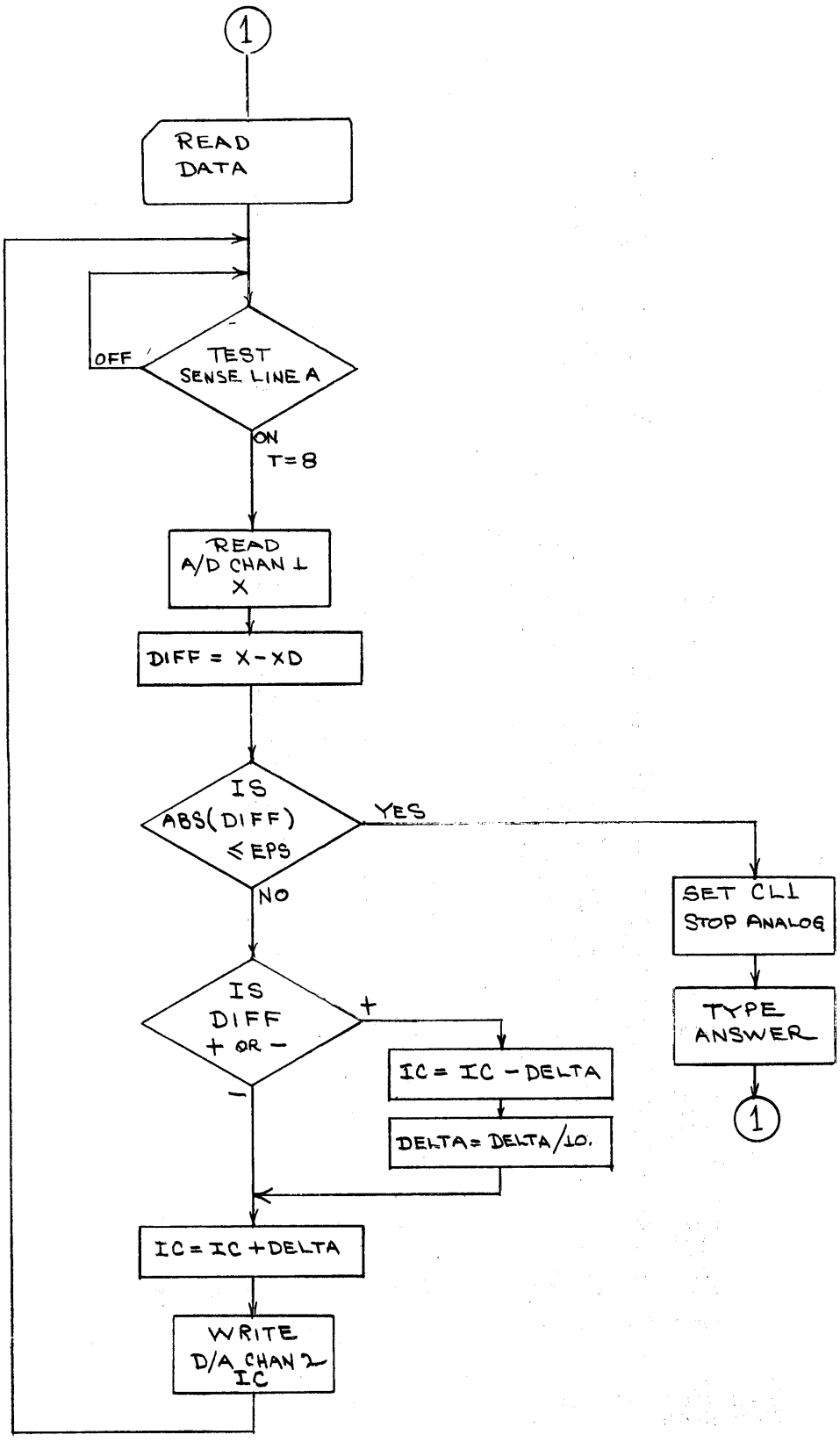


Figure 15. Flow chart for digital program.

Table II

```
*     SPLIT-BOUNDARY HYBRID PROBLEM NO. 1
*
START  H                        02402 48 00000 00000
       TFM   KOUNT,0,10         02414 16 03099 00000
       WN    ZEROES,2000        02426 38 03100 02000
       BI    *+12,9000          02438 46 02450 09000
       BI    *+12,9100          02450 46 02462 09100
       RNCD  INPUT              02462 36 03104 00500
       SF    INPUT              02474 32 03104 00000
       SF    INPUT+5            02486 32 03109 00000
       SF    INPUT+10           02498 32 03114 00000
       SF    INPUT+15           02510 32 03119 00000
       TF    IC,INPUT+7         02522 26 03186 03111
       WN    IC-2,2000          02534 38 03184 02000
WAIT   BNI   WAIT,9000          02546 47 02546 09000
       RN    X,1000             02558 36 03188 01000
       TF    DIFF,INPUT+2       02570 26 03194 03106
       S     DIFF,X+2           02582 22 03194 03190
       MF    SAVEF,DIFF         02594 71 03195 03194
       C     EPS,DIFF           02606 24 03116 03194
       BP    ENDER              02618 46 02762 01100
       BNF   STEP,SAVEF         02630 44 02726 03195
       CM    KOUNT,2,10         02642 14 03099 00002
       BE    ERROR              02654 46 03050 01200
       S     IC,DELTA           02666 22 03186 03121
       TF    DELTA,DELTA-1      02678 26 03121 03120
       CF    DELTA-1            02690 33 03120 00000
       TDM   DELTA-2,0,11       02702 15 03119 00000
       AM    KOUNT,1,10         02714 11 03099 00001
STEP   A     IC,DELTA           02726 21 03186 03121
       BI    *,9000             02738 46 02738 09000
       B     WAIT-12            02750 49 02534 00000
ENDER  K     0,1001             02762 34 00000 01001
       RCTY                     02774 34 00000 00102
       CF    INPUT              02786 33 03104 00000
       CF    IC-2               02798 33 03184 00000
       TD    MSG+14,INPUT       02810 25 03211 03104
       TD    MSG+16,INPUT+1     02822 25 03213 03105
       TD    MSG+20,INPUT+2     02834 25 03217 03106
       MF    SAVEF,IC           02846 71 03195 03186
       TD    MSG+38,IC-2        02858 25 03235 03184
       TD    MSG+40,IC-1        02870 25 03237 03185
       TD    MSG+44,IC          02882 25 03241 03186
       TD    MSG+66,X           02894 25 03263 03188
       TD    MSG+68,X+1         02906 25 03265 03189
       TD    MSG+72,X+2         02918 25 03269 03190
       TD    MSG+86,EPS-2       02930 25 03283 03114
       TD    MSG+88,EPS-1       02942 25 03285 03115
       TD    MSG+92,EPS         02954 25 03289 03116
       BNF   POS,SAVEF          02966 44 03026 03195
       TDM   MSG+31,2           02978 15 03228 00002
WRITE  WATY  MSG                02990 39 03197 00100
       H                        03002 48 00000 00000
       B     START              03014 49 02402 00000
POS    TDM   MSG+35,1           03026 15 03232 00001
       B     WRITE              03038 49 02990 00000
ERROR  RCTY                     03050 34 00000 00102
       RCTY                     03062 34 00000 00102
       WATY  ERMSG              03074 39 03293 00100
       B     ENDER              03086 49 02762 00000
KOUNT  DC    2,0                03099 00002 00
ZERO   DC    4,000@             03103 00004 000‡
ZEROES DS    ,ZERO-3            03100 00000
INPUT  DSS   80                 03104 00080
IC     DC    3,0                03186 00003 000
       DC    1,@                03187 00001 ‡
X      DSC   3,0                03188 00003 000
       DC    1,@                03191 00001 ‡
DIFF   DC    3,0                03194 00003 000
SAVEF  DC    1,0                03195 00001 0
DELTA  DS    ,INPUT+17          03121 00000
EPS    DS    ,INPUT+12          03116 00000
MSG    DAC   48,FOR XD=99.9, XDOT=+99.9, XACTUAL=99.9, EPS=01.00@
                                03197 00096
ERMSG  DAC   15,NO CONVERGENCE@
0                               03293 00030
       DEND  START              02402

END OF ASSEMBLY.
03322 CORE POSITIONS REQUIRED
00077 STATEMENTS PROCESSED

                        SYMBOL TABLE

ZEROES 03100    DELTA 03121    DIFF  03194    ENDER 02762    EPS   03116
 ERMSG 03293    ERROR 03050    IC    03186    INPUT 03104    KOUNT 03099
 MSG   03197    POS   03026    SAVEF 03195    START 02402    STEP  02726
 WAIT  02546    WRITE 02990    X     03188    ZERO  03103
```

To summarize:

$S_o x(0.5)$ = XD = 65.0

$S_1 \dot{x}(0)$ starting value = IC = 0.0

DELTA starting value = 10.0

EPS = 0.1

The problem solution, with this particular set of data, required 16 iterations. The final value for $S_1 \dot{x}(0)$ = XDOT = +11.2. This is the magnitude scaled value. The true value is XDOT/S_1 = 11.2/0.1 = 112.0.

Use of a Multiplier in place of a Servo set Potentiometer

In order to solve problems that require a problem variable to be multiplied by a constant, which in turn is to be set by the digital computer, a servo set pot is frequently used. Since the present system does not yet have servo pots, the multiplier connection depicted in Figure 16 has been used.

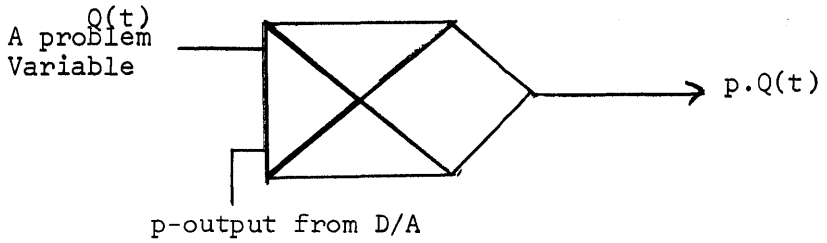


Figure 16. Use of a multiplier to control problem variable from 1620.

Future Plans for the Hybrid System

In order to expand the versatility of the system, several servo set potentiometers are being installed. These will be set through the use of CONTROL functions and D/A outputs.

Several projects which are planned will use the hybrid system to control actual hardware model systems, such as a model power generation plant and distribution network, and a model manufacturing process.

Subroutines for Set and Group Manipulations

by

Charles Weingart

Rockhurst College

COMMON meeting

April 8 - 10, 1968

At Rockhurst College the 1620 Computer is used primarily for educational or mathematical purposes. Therefore, it was decided to attempt to write programs which would help demonstrate certain ideas in various branches of mathematics. So, programs were written finding prime numbers, doing modular arithmetic, performing numerical integration, computing pi, and other similar programs. These programs used, for the most part, the arithmetic or computational abilities of the 1620.

However, the course in abstract algebra did not benefit; since, in this course, numbers are rarely used. Instead, only symbols or character-strings are used ~~an~~ and manipulated. Potentially, the 1620 is capable of doing such manipulations; but the languages available are not suitable. The SNOBOL language seemed most suitable for manipulating symbols; however, the present implementation for the 1620 operates much too slowly and consumes most of core storage. Finally, FORTRAN was chosen, because object programs execute quickly, have most of the storage available, and is probably the most widely used, especially at Rockhurst College. So, the programs were written as multiple augment subroutines for FORTRAN.

Before discussing the subroutines, some background is necessary. A Set is a named collection of elements, each element normally being a character string. In FORTRAN, this is easily represented by an array, usually, the elements are in "A" format, to keep the generality desired. A Mapping is a general function of the elements of the set. A mapping is a generalization of the functions Sin, Cos, etc., in FORTRAN.

A Product is a binary operation, or function of two elements at a time, giving a third element. If the elements of the set were numbers, this operation or "product" could be normal addition. In FORTRAN, if the Set is an array of order N, the product could be defined as a doubly dimensioned array, having N squared elements. If the elements of the set to be operated has subscripts I and J, then the product element could be subscripted (I, J). This approach gives the fastest operation for long tables and is easily implemented.

The product is said to be closed if all of the answers are in the original set. If there is an element in the set which does the same thing as a 0 (zero) for normal addition, this element is called the identity. If, given an element, there is another element such that the two, when operated by the product, give the identity, the elements are each other's inverse. That is, they correspond to positive and negative numbers.

The subroutines were written to test all of these properties and to check for symmetricity, count the non-duplicated elements in sets, and subroutines to look up products, mappings, or single elements.

These subroutines were written for a version of AFIT FORTRAN. This is a derivative of FORTRAN I with a larger symbol table and greater diagnostics than FORTRAN I. For these subroutines, the processor was patched to allow multiple arguments in functions, arrays may be mentioned without subscripts in function arguments, and the mode of the function is determined by the first letter, as in normal variables. The subroutine deck included an extra subroutine to extend and simplify input-output, and allowed one subroutine to call another.

Since the arrays are those of FORTRAN I, a simple formula enables the subroutines to locate an element in an array. If A is the base address transmitted in the argument of the subroutine, and I is the number of the element, it's address is A - 10 * I + 10. If B is the base of an N by N product table, the address of element (I, J) is B - ((I - 1) * N + J - 1) * 10. Fig. 3 This is because the elements are stored column-wise.

The tests are obvious. Closure checks if each element in the table is in the set specified. The identity test looks for an element whose row and column duplicates the original set. The inverse subroutine looks for the identity in the table. The group routine does these three tests, and checks that there are no duplicates in rows or columns.

The various tests are then directly implemented in S. P. S. and placed in the subroutine deck. It is not important to these routines whether the arrays are integer or floating point in nature.

Here are the descriptions of the individual subroutines. N is assumed to be an integer variable or constant. See fig. 5

NSTP (A)

This subroutine "strips" the mantissa of the argument from internal alphanumeric form to integer form. For example, from A format to I format. Consider this program. A maybe subscripted:

```
READ I,  A
FORMAT (A 4)
I = NSTP  (A)
```

See fig. 2

If 1234 is read in, A - .71727374
and I = 1234.

If 123 (blank) is read in, A = .71727300 and I = 1230 (Blanks are converted to zeroes)

If 123M is read in , A = .71727354 are I = - 1234. (Flag on input number makes it negative.)

FILL (N)

This subroutine performs exactly the opposite function as the NSTP subroutine. N may be subscripted.

If N = 1234, FILL (N) = .71727374 etc.

CLOS (N, SET, OP)

SET is a set name having N elements. OP is the name of an NxN product table, presumably of the elements of SET times themselves. This subroutine tests to see that every element of OP is in SET. If they are, the answer is 1. If not the answer is 0.

AIDN (N, SET, OP)

N, SET, and OP are under the same restrictions as in CLOS. This subroutine looks for an element of SET such that when multiplied by every member of SET, either element first, the product is that (other) member. The answer is that element found. If none is found, an answer of Q.EO

If none is found, an answer of O.EO is returned (all blanks in A 5 format).

If A is from SET, we want E to be the identity, such that A x E = E x A = A.

AINV (N, SET, OP)

N, SET, and OP have the same restrictions as in CLOS. This subroutine, given that the product table has an indentity (AIDN). Then if every element of SET is multiplied by another certain element, either element first, the

product of these is the identity, i.e. if A is from SET, then there is a B such that A x B = B x A = E, the identity. AINV automatically calls AIDN to get the identity. The naswer has the same form as the CLOS subroutine.

GRUP (N, SET, OP)

N, SET, and OP have the same restrictions as in CLOS. A group is a set and a product defined over the set, and having the qualities of closure (CLOS), identity AIDN, every element an inverse (AINV), and associative, i.e. Ax(BxC) = (AxB)xC. This subroutine performs all of these tests. The answer has the same form as CLOS.

NORD (N, SET)

SET is the name of a set and N is the maximum number of elements in it. The answer is the number of non-duplicate non-duplicate non-blank (O. EO) elements in SET. All duplicates are set to O. EO and moved to the end of the set.

NELT (N, SET, ELT)

N and SET have the same restrictions as NORD. ELT is compared against each element of SET, and the answer is the element it matched. If it matched SET (1), 1 is returned; SET (2), 2 is returned, etc. If no match was found, O is returned, N and ELT may be sub-scripted.

HINV (N, SET, OP, ELT)

N, SET, and OP have the same restrictions as CLOS; N and ELT the same as NELT, ELT is checked in the product table to see which element of SET is its inverse, i.e. such that ELTxB = BxELT = E, the identity. The The answer is the inverse. If ELT does not have an identity, or ELT is not in SET, and answer of O. EO is returned.

RIDN (N, SET, OP)

This subroutine performs the same function as AIDN, except only the right identity is checked for, i.e. and element E such that AxE = A for any A from SET, The answer is the same.

RINV (N, SET, OP)

This subroutine performs the same function as AIDN, except only a right inverse is check for, that there is a B so the ELT x B = E, the right identity. The answer is the same as for HINV.

GPRD (N, SET, ELT1, ELT2, OP)

N, SET, and OP have the same restrictions as in CLOS. If SET and OP form a group, the product of ELT1 x ELT2 is found and returned. If not, O.EO is returned.

RCD (A)

The RCD subroutine is different from all the other Algebraic Compiler Subroutines. Its purpose is to modify the Fortran input and output routines to be more adaptable for the type of operations necessary for the Algebraic compiler.

If A is - N. N = O, 1, 9

A card is read once then can be re-read N times by calling READ.

0. This resets all input modifications (-N, 1., 2., 3., 4.)
1. This restores the I/O Buffer see 2
2. This restores the I/O Buffer to allow an output operation while retaining free input.
3. This allows free-type input under Format control: Leading blanks are eliminated on A, E, F, G and I formats. New card read if only

blanks on present one.

4. There is no read on an input) useful with 3.

5. The I/O Buffer is not cleared on an output). New output data is added after old.

6. There is no output on a) if not the last. Useless without 5.

7. There is no output on last). The output record remains in the Buffer for future operations.

8. Restores all output modification. (5., 6., 7.)

   Unlike all other subroutines. This one may have various numbers of arguments. i.e.

   RCD (0. 3., 4.) RCD (-3., 3.)
   RCD (8., 5., 7.) RCD (-1.)
   RCD (2., 7.) RCD (0., 8.)

   These are all valid, and perform the specific function in the older of the arguments.

Here is a small program illustrating these subroutines. Statements 1 through 20 read in the set and the operation table. The remaining statements test for closure, identity, inverse, and group qualities. Granted, this program has limited applications other than educational, it illustrates the priciple described here. See fig. 6

Here is an assymetric operation table. The program tests and types the results within 5 seconds. ~~Below~~ Above it is a smaller, symmetric table, which is a group. This test is completed in shorter time. Figs 7, 4

Fig. 1

SET1...	A	B	C	D	E
SET2..	F	G	H	I	
	F	G	H	I	
A	A	F	K	P	
B	B	G	L	Q	
C	C	H	M	R	
D	D	I	N	S	
E	E	J	O	T	

Fig. 2

```
      READ 1, A
1     FORMAT(A4)
      I=NSTP(A)
```

INPUT	A	I
1234	A= .71727374	I= 1234
123	A= .71727300	I= 1230
123M	A= .71727354	I= -1234

Fig. 3

$$B - ((I - 1) * N + J - 1) * 10$$

```
SET..     A     E

PRODUCT...

          A     E

A         E     A
E         A     E
```

```
SET..     A     E

PRODUCT...

          A     E

A         A     A
E         A     E
```

```
SET..     A     B     C     D     E

PRODUCT...

          A     B     C     D     E

A         B     E     D     E     A
B         E     C     E     B     B
C         D     E     C     E     C
D         C     B     E     D     D
E         A     B     C     D     E
```

Fig. 4 set + Product examples

SUBROUTINE	LENGTH
NSTP (ANUM)	38
FILL (N)	50
CLOS (N,SET,OP)	404
AIDN (N,SET,OP)	620
AINV (N,SET,OP)	664
GRUP (N,SET,OP)	520
NORD (N,SET)	420
NELT (N,SET,ELT)	320
HINV (N,SET,OP,ELT)	780
ABEL (N,OP)	450
GPRD (N,SET,OP,ELT1,ELT2)	620
RIDN (N,SET,OP)	428
RINV (N,SET,OP)	588
RCD (ANUM)	762

Fig. 5

```
      DIMENSIONA(900),B(30)
C     READ SIZES
1     DUM=RCD(0.)
      READ,N
      NSQ=N*N
C     READ SET
      DUM=RCD(3.,4.,-0.)
      DO 10 I=1,N
10    READ 100, B(I)
100   FORMAT(A5)
C     READ PRODUCT TABLE
      DUM=RCD(-0.)
      DO 20 I=1,NSQ
20    READ 100, A(I)
C     CLOSURE TEST
      IF(CLOS(N,B,A)) 201, 202, 201
201   PRINT 200, 684562.
      GOTO 203
200   FORMAT(/9HCLOSURE  ,A4)
202   PRINT 200, 5556.
C     FIND IDENTITY
203   D=AIDN(N,B,A)
      PRINT 300, D
300   FORMAT(12HIDENTITY IS ,A5)
C     CHECK INVERSE
      IF(AINV(N,B,A)) 401, 402, 401
401   PRINT 400, 684562.
      GOTO 403
400   FORMAT(9HINVERSE  ,A4)
402   PRINT 400, 5556.
C     FINAL GROUP TEST
403   IF(GRUP(N,B,A)) 501, 502, 501
501   PRINT 500, 684562.
      GOTO 503
500   FORMAT(7HGROUP  ,A4,/)
502   PRINT 500, 5556.
503   STOP
      END
```

Fig. 6.

Sample Program

```
2
A    E
E    A    A    E


CLOSURE    YES
IDENTITY IS E
INVERSE    YES
GROUP   YES


2
A    E
A    A    A    E

CLOSURE    YES
IDENTITY IS E
INVERSE   NO
GROUP    NO


5
A      B      C      D      E
B      E      D      E      A      E      C      E      B
 B      D      E      C
E      C      C      B      E      D      D      A      B
C      D      E

CLOSURE    YES
IDENTITY IS E
INVERSE    YES
GROUP    NO
```

Fig. 7
Sample input and output for three products

SESSION REPORT

COMMON - Chicago

Session Number MON D1 Session Name 360 DOS

Chairman A. Ragsdale

Time 3:30 to 5:00 Attendance (No.) 88

Speakers Mr. Don Moeller - IBM (BTAM and QTAM)

Synopsis of Meeting BTAM - Basic Tellecommunications Access Method for high speed and low speed terminals. Capabilities of BTAM: 1) Terminal polling, 2) Message receiving, 3) Terminal addressing, 4) Message sending, 5) Terminal dialing, 6) Call answering, etc.

QTAM - Queued Telecommunications Access Method. A complete message handling language for low speed terminals. QTAM Capabilities: as well as all the BTAM capabilities we have 1) Dynamic buffering, 2) Checkpoint for communication lines, 3) Time and date stamping, 4) Message sequence checking, 5) Queuing of messages on disk or in core storage, 6) Logging of messages, 7) Rerouting of undelivered messages, etc.

MR. DON MOELLER - IBM

Chicago Field Systems Centre

214 North Michigan Avenue

Chicago, Illinois 60601

COMMON Chicago Conference

360 DOS Project

Session MON-D1

DOS BTAM and QTAM

Presentation To COMMON On DOS BTAM And QTAM

Foil 1. The two major IBM supplied programming support packages for DOS teleprocessing users are BTAM and QTAM. BTAM is an access method, i.e. it is a set of macro instructions which greatly simplify the programer's use of terminals as I/O devices. QTAM is an extensive message handling language.

Foil 2. DOS BTAM provides programming support for a wide range of terminals. Among these are tone transmission terminals such as the 1001, 1092 and touch-tone phones; start-stop terminals, including both IBM terminals (1050, 1030, 2740, 1060) and teletype terminals (83B3, 115A, TWX); higher speed synchronous terminals-the 2780, 1130 and S/360 model 20; and other S/360 systems. OS BTAM provides similar support, excepting tone transmission terminals and locally attached 2260 graphic terminals.

Foil 3. The overall environment in which the terminals operate is shown here. Moving inward from the remote end of the communication line we find the terminal with its various components such as keyboard, card reader, paper tape reader, printer, video display, etc. The terminal interfaces to the communication line via a data set or other line adapter. The communication line itself falls into one of two general categories-nonswitched or switched. A second data set or line adapter provides a line interface at the communication control unit (2701, 2702, 2703, 7770 or 7772). The control unit interfaces to the S/360 processor normally through the multiplex channel.

Foil 4. The first four BTAM capabilities relate to reading from and writing to terminals on nonswitched lines. The next two are required to place and accept calls automatically on switched lines. These six functions, together with error detection and recovery, provide normal access method capabilities. In addition, BTAM provides other facilities important to "real-time" systems. These include on line terminal testing, error statistics, terminal list modification, buffer pool management, and the ability to operate in a multiprogrammed environment.

Foil 5. Message processing is a user responsibility not only with BTAM but with any teleprocessing system. The other functions on this list must be coded by the BTAM user. We will find later that these are provided by QTAM.

Foil 6. The BTAM macro instruction set, though providing a powerful teleprocessing capability, is small in number. Training in the use and coding of these macro instructions can be accomplished in three or four days of classroom work.

Foil 7. Operation of BTAM in a multiprogramming environment depends on two factors. First, the user must code a multiple WAIT macro instruction in his BTAM program.

The DOS Supervisor in response to this coding will give CPU control to another partition. Second, the DOS Supervisor will immediately respond to an interrupt generated by the completion of a message sending or message receiving operation. The Supervisor will give CPU control back to the BTAM program. Thus, the BTAM program, when it has highest priority, will not relinquish control of the CPU until the programmer allows this to happen. It will regain control of the CPU as soon as it has useful work to perform. No other program running simultaneously can prevent BTAM from receiving CPU control.

Foil 8. BTAM is designed to manage a number of communication lines simultaneously. Once the read or write operation on each line has been started by BTAM, no further processing is required until one of these operations has completed. The CPU is free to be used for processing of a background job in a lower priority partition.

Foil 9. A typical BTAM inquiry program consists of two major elements. The first is the line control function which consists of BTAM reads and writes with a BTAM WAIT as the controlling element. The second is the logical processing performed on the inquiry after it arrives at the CPU. (This part of the program is outlined in dotted lines.)

Foil 10. The most straightforward way to write a BTAM program is as a unified, one partition program. If the total time required to process a message is less than the average interval between messages, this is completely satisfactory. However, there may be peak periods of message activity, such as between 4:30 and 5:00 P.M. Because the BTAM program cannot accept messages from the network any faster than they can be processed, the desired peak arrival rate may not be satisfied and messages may temporatily queue at the terminals.

Foil 11. A more flexible technique uses two partitions. Since the line control partition is not related to message processing speeds, it can react readily to transmission peaks. Message processing still proceeds at the same rate, but a disk queue now acts as a buffer at the times when message input rates exceed message processing rates.

Foil 12. A two partition system involving BTAM inquiry handling in the foreground and a batch processing job in the background is used for estimating core storage requirements.

Foil 13. The core storage requirement for BTAM control blocks, tables, macro instructions and subroutines necessary for a system of four communication lines and sixteen 1050 terminals is estimated as 6940 bytes. This is exclusive of message input/output areas.

Foil 14. Increasing the network size by a factor of five does not have an equivalent effect on BTAM core storage requirements. Instead, these storage requirements increase by less than 50%.

Foil 15. If the background partition is established at 14K, this two partition BTAM system is estimated to require approximately 42,000 bytes of core storage.

Foil 16. DOS QTAM also supports a wide range of terminals. It differs from DOS BTAM in that it does not provide support for the higher speed terminals-the 2780, 1130, S/360 model 20 and other S/360 processors.

Foil 17. A comparison of BTAM and QTAM capabilities shows the many additional message handling facilities provided by QTAM.

Foil 18. The QTAM macro instruction language can be divided into five major categories. The total is seventy macro instructions, with message handling macros accounting for about forty percent of the total.

Foil 19. The QTAM system always uses a multiple partition environment, as previously described in connection with foil 11. This provides an excellent generalized approach for communications systems. It keeps the line control function entirely independent of the message processing function.

Foil 20. A DOS QTAM system must have a message control program for communication line management. This must be in the highest priority partition. If messages are only being switched between terminals, a message processing program is not a requirement, except when it is time to shut down the message control program. If processing is required for all or some of the messages, one or two message processing programs may be multiprogrammed with message control. Two message processing programs may be used to improve system thruput by providing the capability for concurrent processing of two messages.

Foil 21. A message segment enters a core storage buffer area in the QTAM message control partition. The segment is processed by a set of message handling macro instructions referred to as the Line Procedure Specification. The message segment is usually written on a disk process queue. The segment is also placed in a core storage process queue of limited size. In response to a GET macro instruction issued by one of the QTAM message processing programs, the message segment is moved to a work area. If this segment is no longer available in the core storage process queue, it must be recalled from disk before the GET can be performed. After processing is completed any response is PUT from a work area in the message processing partition to core storage in the message control partition. It is then written to a disk output queue. When the line is available for transmission, the response is recalled from disk, processed through the output portion of the Line Procedure Specification and transmitted from an output buffer.

Foil 22. A typical QTAM environment which can be used for estimating core storage requirements consists of a QTAM message control partition, a QTAM message processing partition and a background batch job.

Foil 23. Core storage requirements for a QTAM message control program which manages four lines and sixteen 1050 terminals is estimated to be 21,310 bytes. This does not include communication line buffer areas.

Foil 24. A similar QTAM message control program for a network five times as large is estimated to require approximately seventy percent more core storage.

Foil 25. The message processing program requirement, exclusive of user coding, is estimated as 3935 bytes.

Foil 26. With 14K assigned to the background batch partition, the total core storage requirement for this three partition system is estimated to be approximately 58,000 bytes.

Foil 27. In reviewing the additional functions offered by QTAM and not found in BTAM we find that these functions fall into five general categories. These are:

1. An IBM supplied queuing technique.
2. A complete set of macro instructions for analysis of all fields in the message header.
3. Special macros for handling of undeliverable messages.
4. A ready-made line control program structure.
5. Checkpoint/restart facilities.

(1)

DISK OPERATING SYSTEM

SUPPORT FOR COMMUNICATION TERMINALS

BTAM

BASIC TELECOMMUNICATIONS ACCESS METHOD FOR HIGH SPEED AND LOW SPEED TERMINALS

QTAM

QUEUED TELECOMMUNICATIONS ACCESS METHOD - A COMPLETE MESSAGE HANDLING LANGUAGE FOR LOW SPEED TERMINALS

BTAM
TERMINAL SUPPORT

TERMINAL	DISK OPERATING SYSTEM	OPERATING SYSTEM
1001	X	
1030	X	X
1050 (SWITCHED)	X	X
1050 (NON-SWITCHED)	X	X
1060	X	X
1092/1093	X	
1130	X	X
2260 (LOCAL)	X	
2260 (REMOTE)	X	X
2740 I & II	X	X
2780	X	X
S/360 MODEL 20	X	X
S/360 MODELS 25 THRU 75	X	X
AT & T 83B3	X	X
W/U 115A	X	X
AT & T 33/35 (TWX)	X	X
TOUCH-TONE } PHONES	X	
TOUCH-CALLING } PHONES		

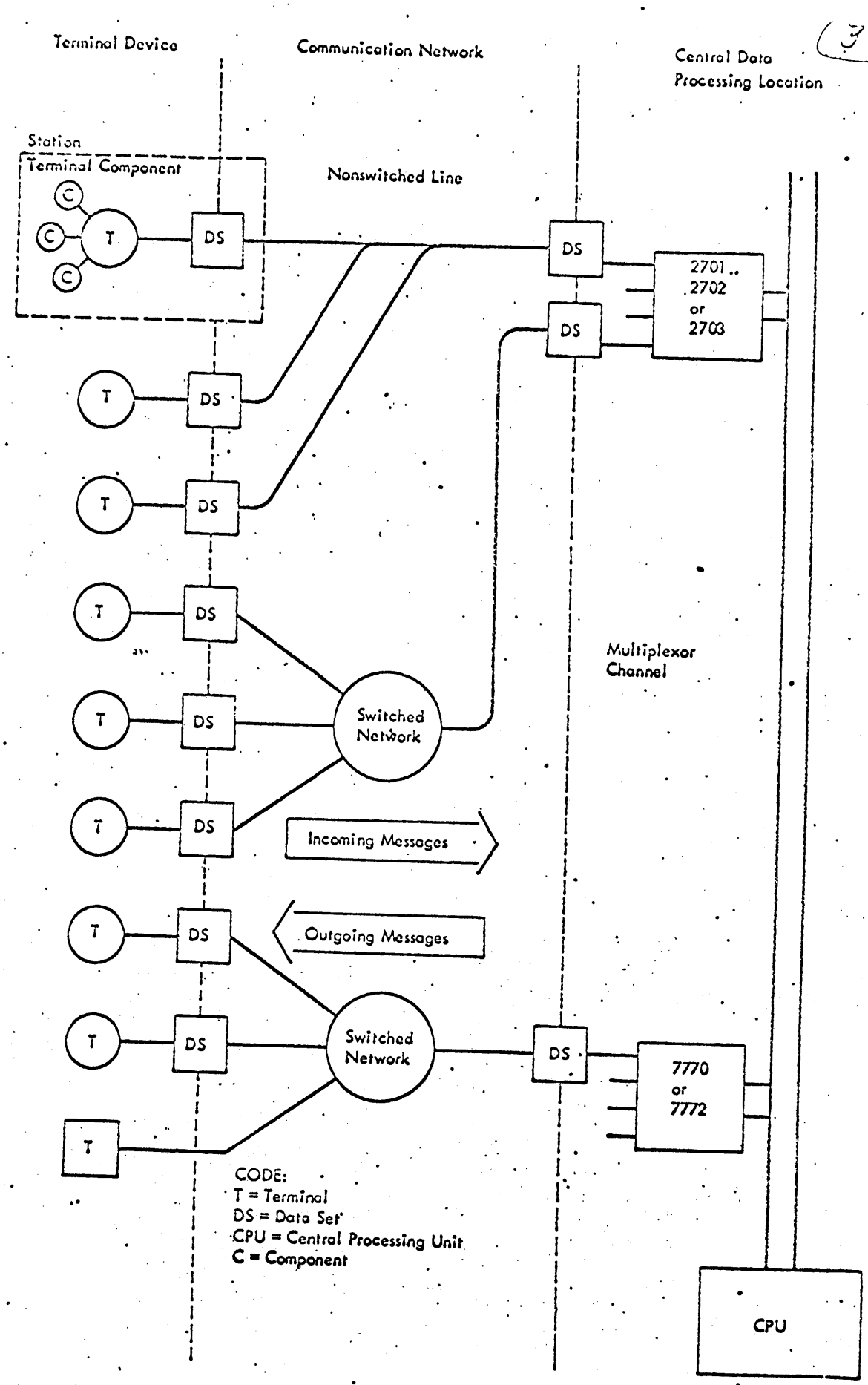


Figure 1. Configuration of a Teleprocessing Installation

BTAM CAPABILITIES

- TERMINAL POLLING
- MESSAGE RECEIVING
- TERMINAL ADDRESSING
- MESSAGE SENDING
- TERMINAL DIALING
- CALL ANSWERING
- ERROR DETECTION & RECOVERY
- ON LINE TERMINAL TESTING
- TERMINAL & LINE ERROR COUNTS
- TERMINAL LIST MODIFICATION (AT OBJECT TIME)
- BUFFER POOL MANAGEMENT
- CODE TRANSLATION
- CPU TIME SHARING WITH OTHER PARTITIONS

User Responsibilities

- Message Processing
- Queuing
- Logging
- Header Analysis
- Time & Date Stamping
- Handling of Undeliverable Messages
- Overlapping of T/P Program Functions
- Communication Between Partitions

BTAM MACROS

◇ Declarative Macros

BTMOD DFTRMLST
DTFBT LERB
ASMTRTAB

◇ Transient Macros

OPEN CLOSE
LOPEN

◇ Data Handling Macros

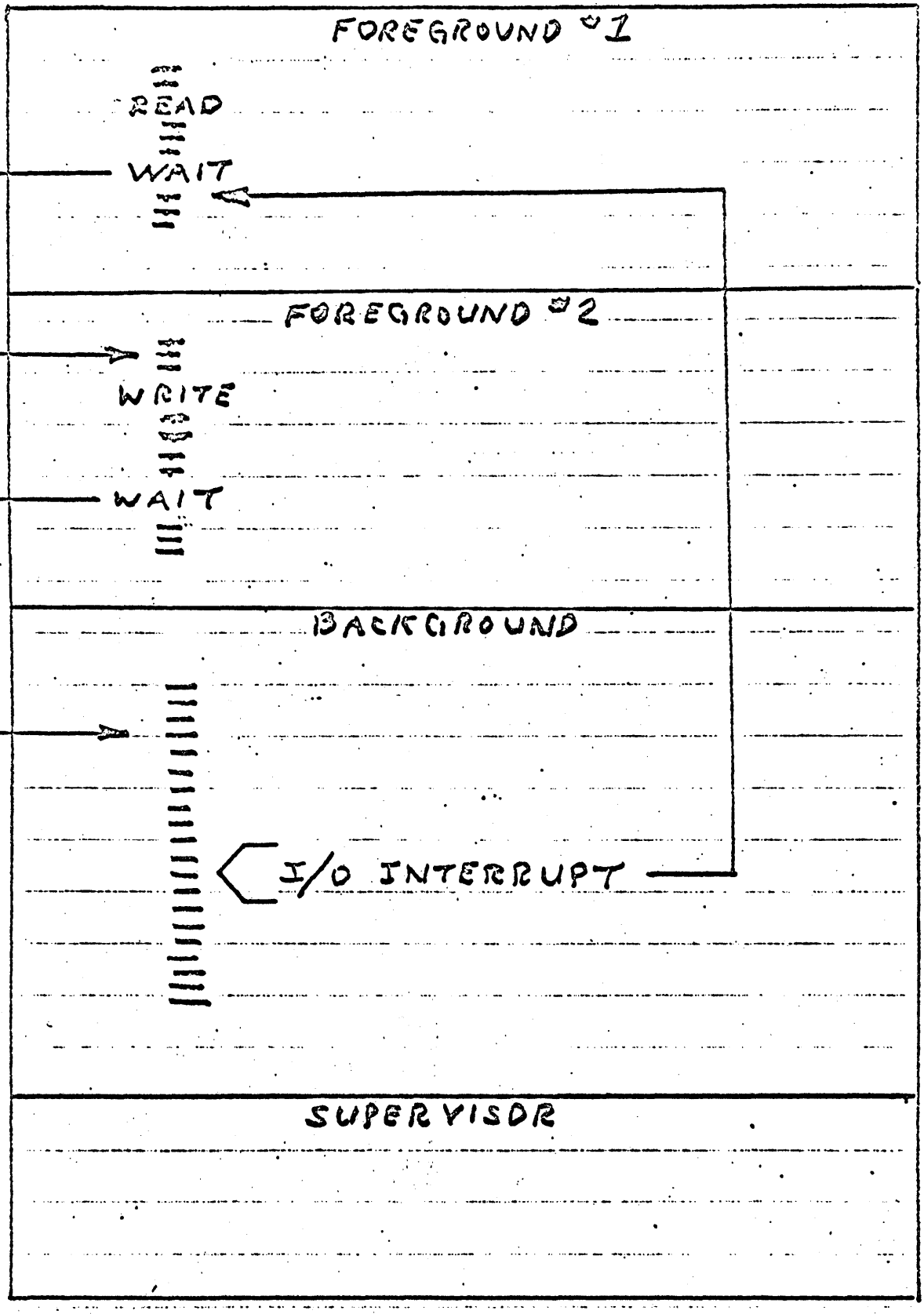
READ WAIT
WRITE TWAIT
CONTROL LERPRT
CHGNTRY TRANSLATE
RESETPL

◇ Buffer Management Macros

REQBUF RELBUF

TOTAL = 19

BTAM IN A MULTIPROGRAMMING ENVIRONMENT



BTAM OPERATION ON MULTIPLE LINES

READ ON LINE #1

READ ON LINE #2

WRITE ON LINE #3

READ ON LINE #4 — EOM — NEXT READ

START NEXT READ ON LINE #4 | PROCESS INPUT DATA

WRITE ON LINE #5

WRITE ON LINE #6

READ ON LINE #7

BACKGROUND PROCESSING — BACKGROUND PROCESSING

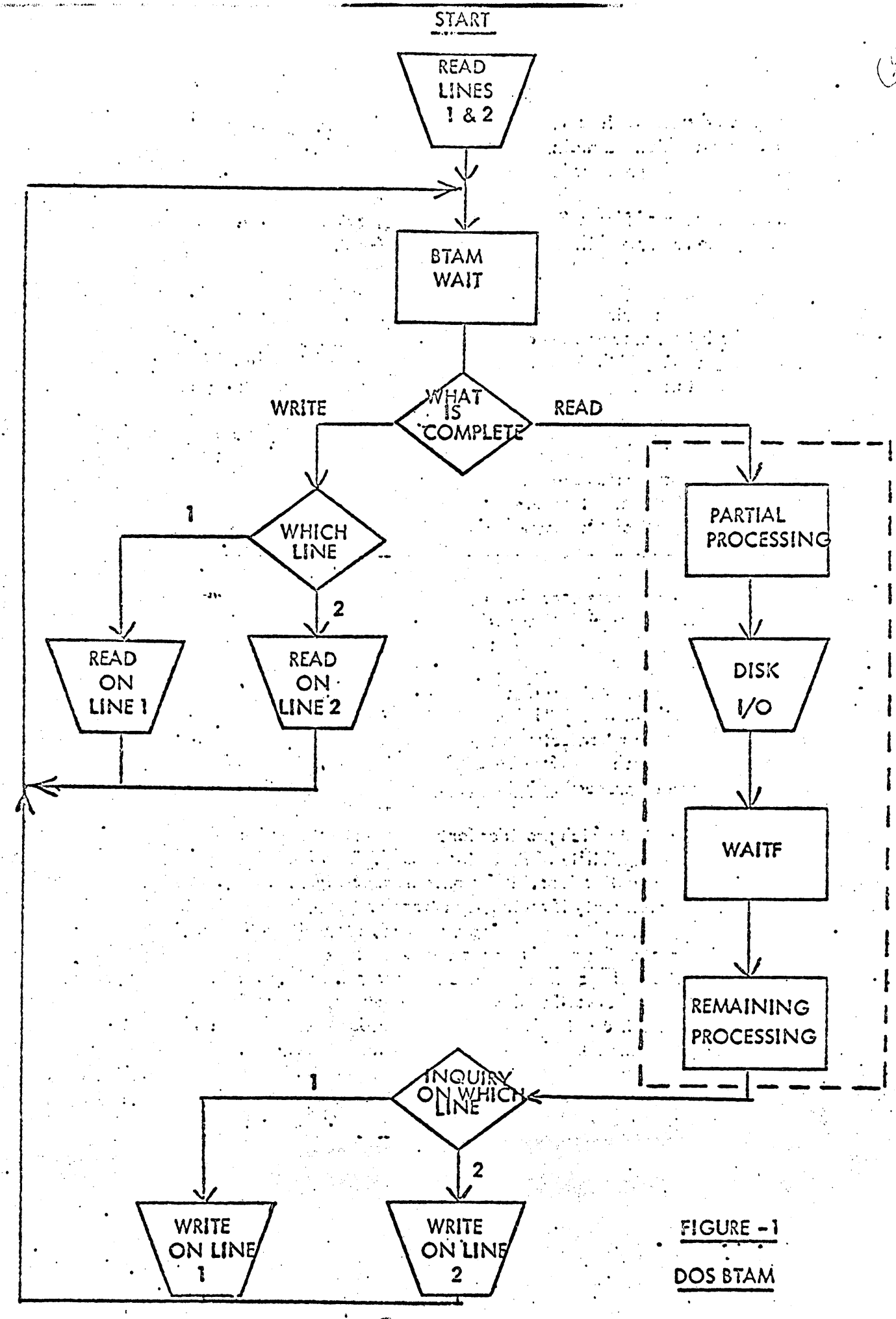
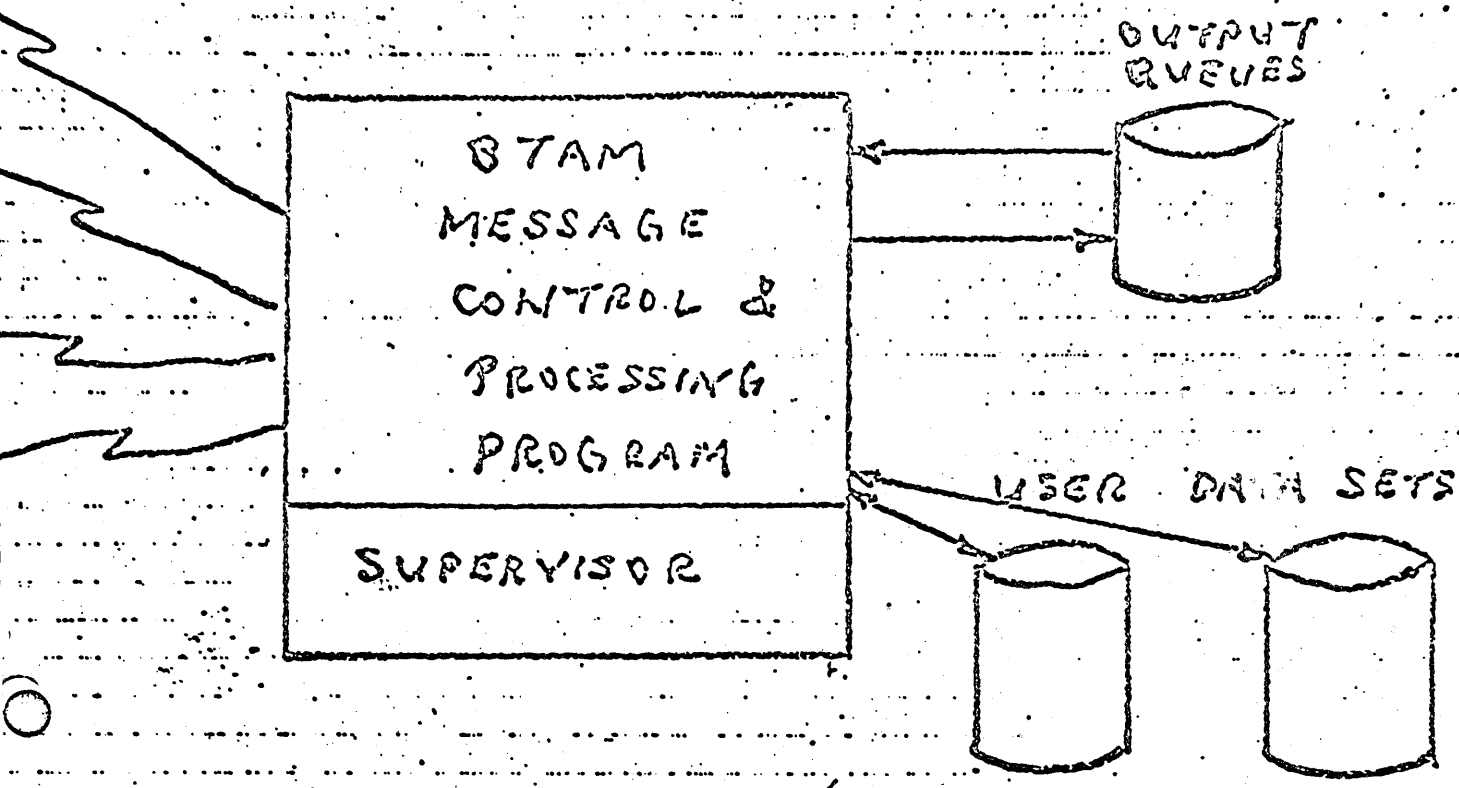


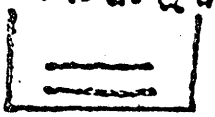
FIGURE -1

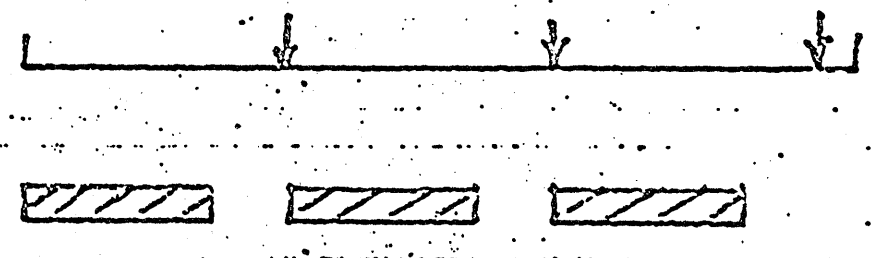
DOS BTAM

BTAM OPERATING ENVIRONMENT

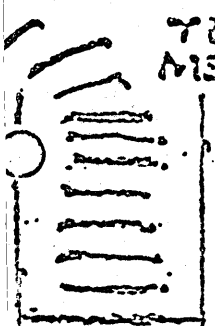


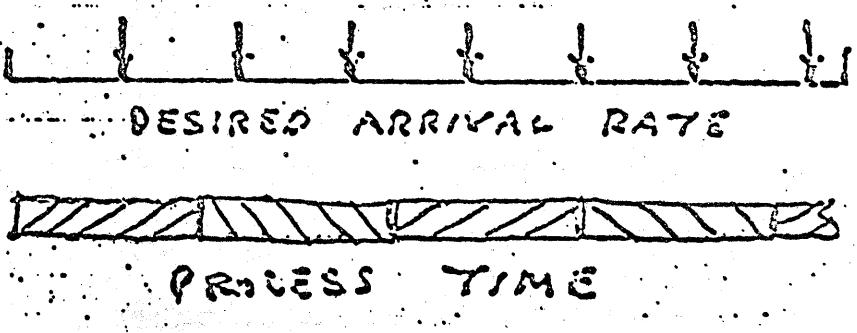
OK



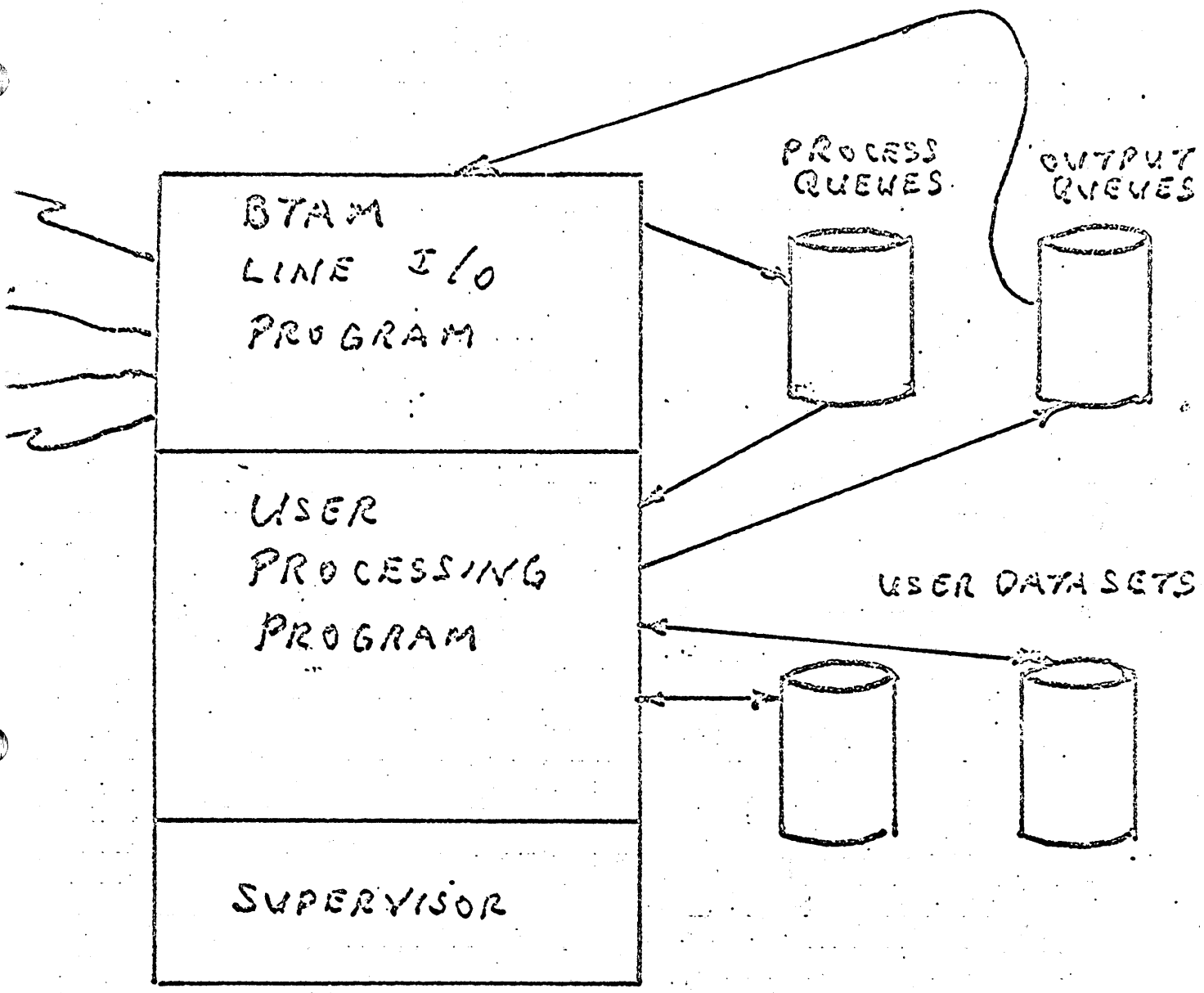


BAD

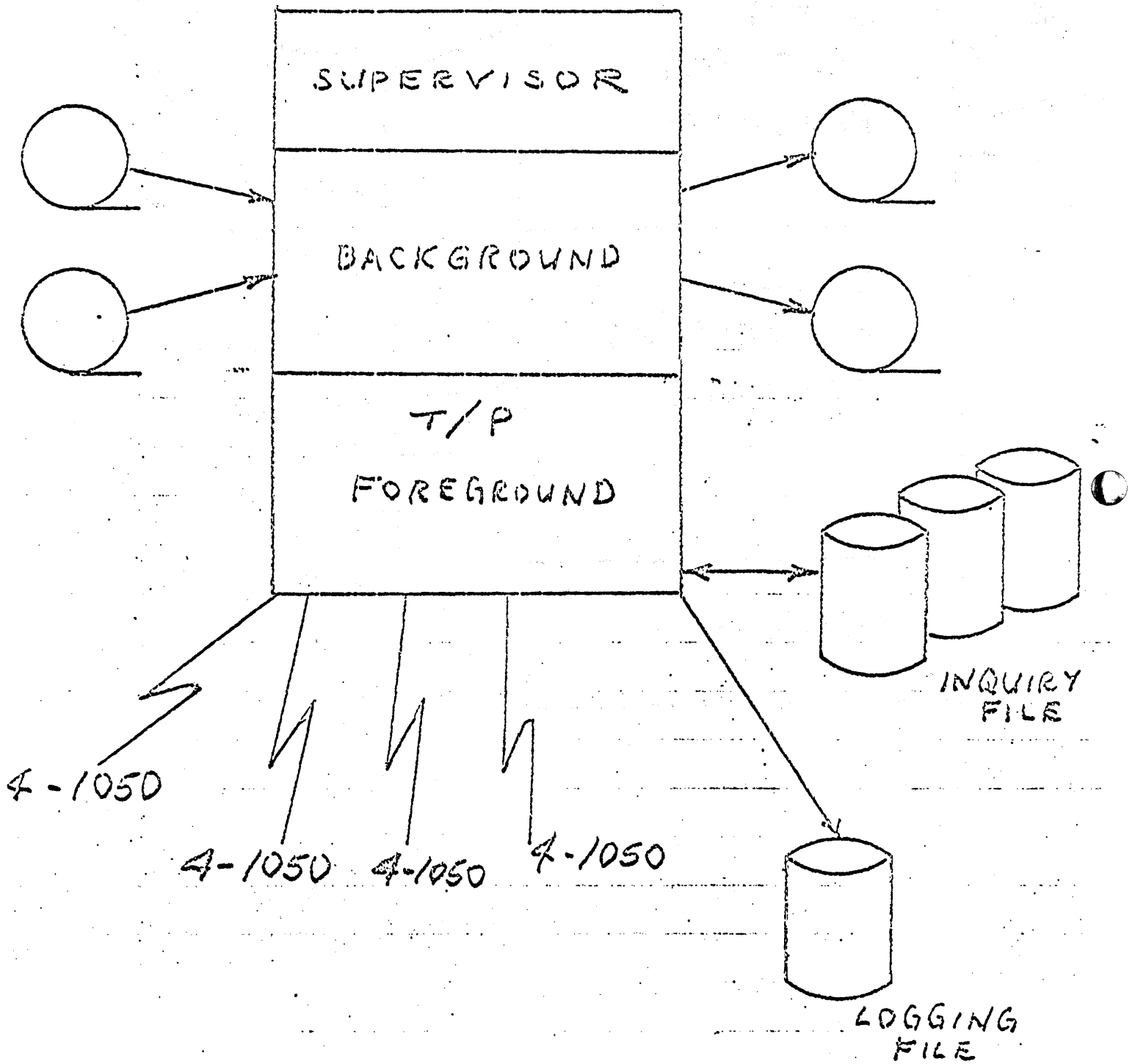




BTAM OPERATING ENVIRONMENT #2



CONFIGURATION FOR STORAGE ESTIMATES



BTAM CORE STORAGE ESTIMATE
(DOS)

BTMOD (BASIC MODULE)	2930
ERROR CORRECTION	2070
ERROR COUNTS	190
TERMINAL TESTS	690
MODEL CHANNEL PROGRAMS	84
CONTROL BLOCKS	600
BTAM MACRO INSTRUCTIONS	240
POLLING LISTS	56
ADDRESSING LISTS	80
	6940

4 LINES
16 1050's (4 PER LINE)

BTAM CORE STORAGE ESTIMATE
(DOS)

BTMOD (BASIC MODULE)	2930
ERROR CORRECTION	2070
ERROR COUNTS	190
TERMINAL TESTS	690
MODEL CHANNEL PROGRAMS	84
CONTROL BLOCKS	1240
BTAM MACRO INSTRUCTIONS	480
POLLING LISTS	280
ADDRESSING LISTS	400
► BUFFER MANAGEMENT	1470
	9834

20 LINES
80 1050's (4 PER LINE)

TOTAL SYSTEM - USING DOS BTAM
[CORE STORAGE ESTIMATE]

- FOREGROUND (BTAM) PLUS BACKGROUND (BATCH)
- 4 LINES AND 16 1050 TERMINALS

SUPERVISOR	10240
BACKGROUND PROGRAM	14336
BTAM	6940
IND. SEQ. LOGICAL IOCS	1190
SEQUENTIAL LOGICAL IOCS	820
IND. SEQ. I/O AREA	400
SEQUENTIAL I/O AREAS (2)	800
COMMUNICATION LINE BUFFERS (4)	520
INQUIRY PROCESSING PROGRAM	7000
	42246

QTAM
TERMINAL SUPPORT

TERMINAL	DISK OPERATING SYSTEM	OPERATING SYSTEM
1001	X	
1030	X	X
1050 (SWITCHED)	X	X
1050 (NON-SWITCHED)	X	X
1060	X	X
1092/1093	X	
1130		
2260 (LOCAL)	X	
2260 (REMOTE)	X	X
2740 I & II	X	X
2780		
S/360 MODEL 20		
S/360 MODELS 25 THRU 75		
AT & T 83B3	X	X
W/U 115A	X	X
AT & T 33/35 (TWX)	X	X
TOUCH-TONE PHONES	X	
TOUCH-CALLING PHONES		

BTAM - QTAM CAPABILITIES

	BTAM	QTA
• TERMINAL POLLING	X	X
• MESSAGE RECEIVING	X	X
• TERMINAL ADDRESSING	X	X
• MESSAGE SENDING	X	X
• BUFFER POOL MANAGEMENT	X	X
• "DYNAMIC" BUFFERING		X
• DIALING A TERMINAL	X	X
• ANSWERING A CALL ON DIAL FACILITIES	X	X
• CODE TRANSLATION	X	X
• ERROR DETECTION AND CORRECTION	X	X
• TERMINAL TESTS	X	X
• MAINTENANCE OF ERROR STATISTICS	X	X
• CHECKPOINT FOR COMMUNICATION LINES		X
• NETWORK CONTROL TERMINAL		X
• TIME AND DATE STAMPING		X
• MESSAGE SEQUENCE CHECKING		X
• MESSAGE TYPE RECOGNITION		X
• QUEUING OF MESSAGES ON DISK OR IN CORE STORAGE		X
• LOGGING OF MESSAGES		X
• INTERCEPTION OF MESSAGES FOR AN OUT OF SERVICE TERMINAL		X
• REROUTING OF UNDELIVERABLE MESSAGES		X
• ERROR NOTIFICATION TO TERMINALS		X
• CANCELLATION OF ERROR MESSAGES		X

QTAM MACRO LANGUAGE

- GROUP 1.
  I/O DEFINITION, INITIATION, TERMINATION - 8

- GROUP 2.
  CONTROL BLOCK, TABLE & BUFFER DEFINITON - 12

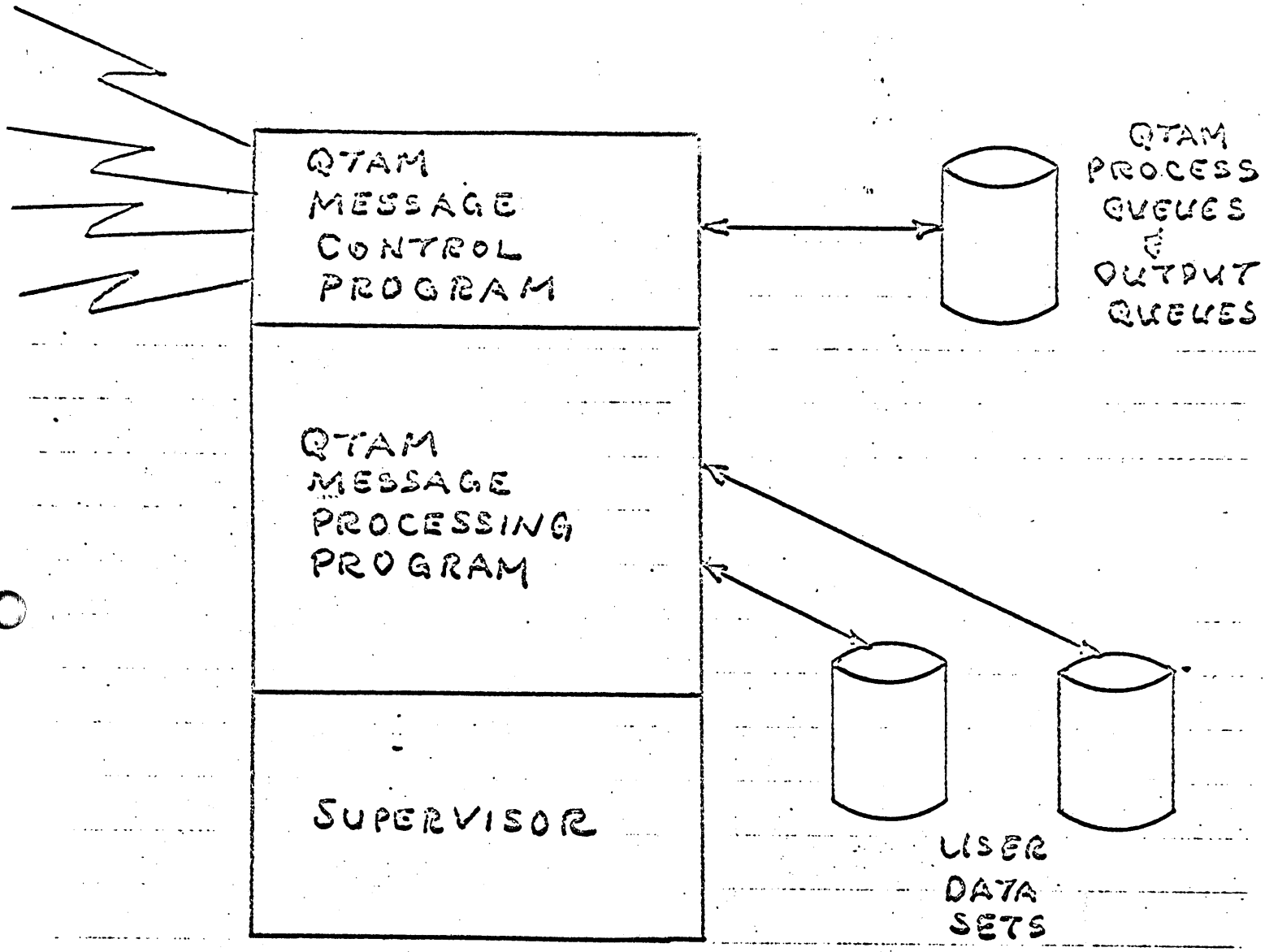
- GROUP 3.
  MESSAGE HANDLING - 27

- GROUP 4.
  DELIMITERS - 10

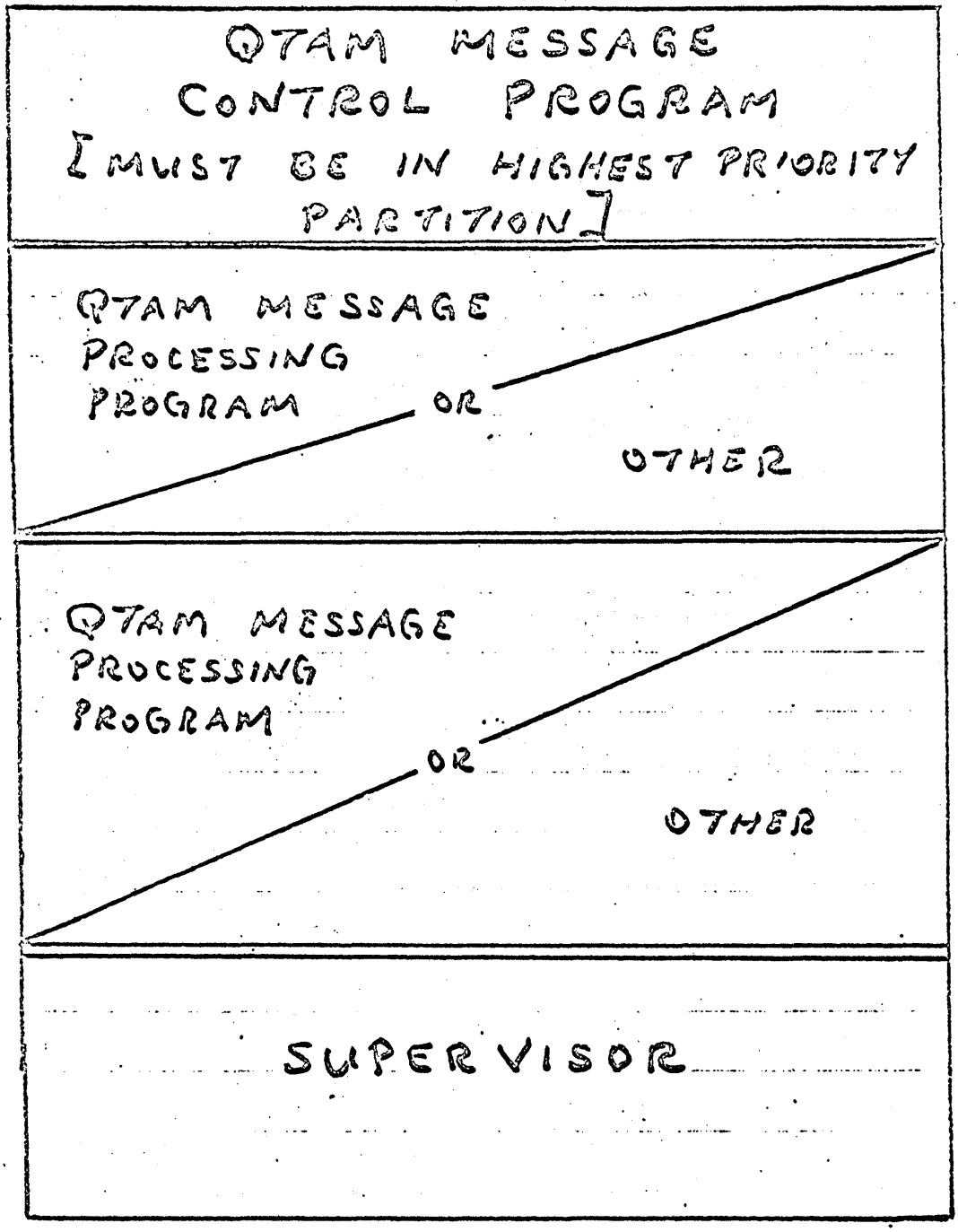
- GROUP 5
  NETWORK CONTROL - 13

TOTAL - 70

QTAM OPERATING ENVIRONMENT

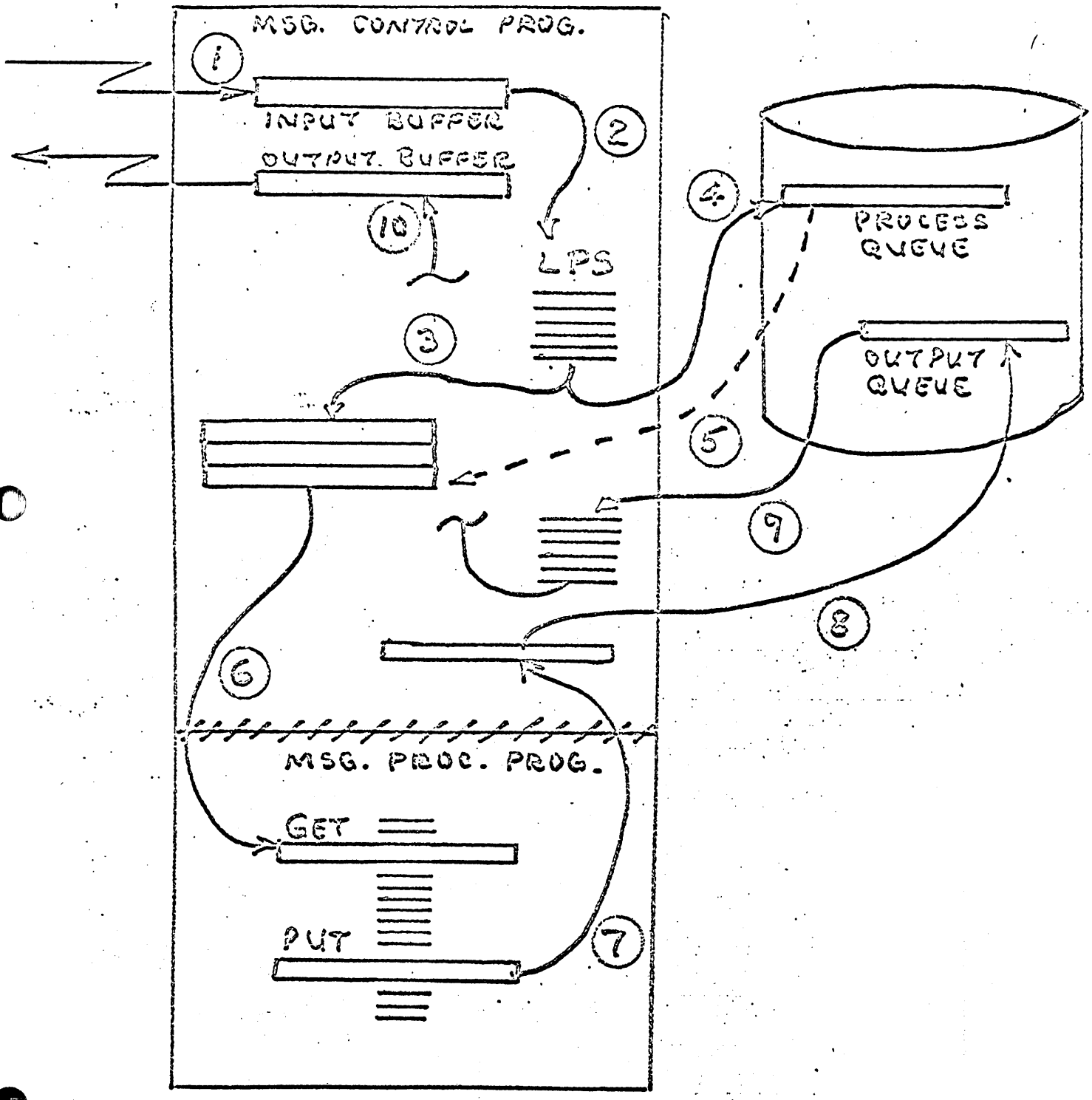


QTAM USE OF PARTITIONS

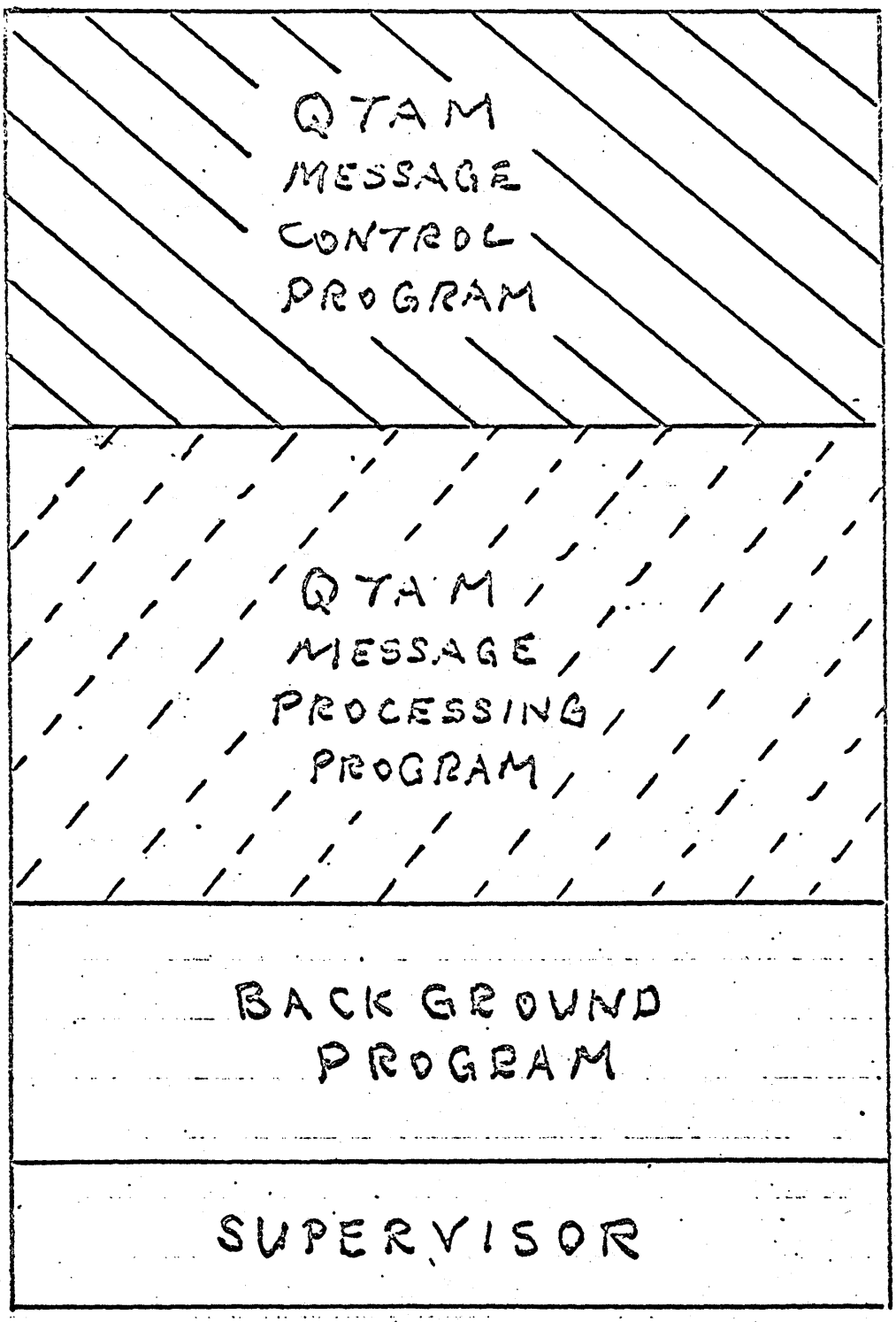


* QTAM MESSAGE PROCESSING PROGRAM MUST APPEAR IN ONE OF THESE PARTITIONS WHEN [illegible]

QTAM MESSAGE FLOW



QTAM Environment For Core Storage Estimates



DOS QTAM Core Storage Estimate
(Message Control Program)

Basic QTAM Modules	11435
Terminal Test Module	1378
Checkpoint Module	1190
Macro Instr. & Subroutines	3500
User Code	1000
Channel Programs	266
Control Blocks and Tables	1910
Checkpoint Area	631
	21310

4 Lines
16 1050s (4 per Line)

DOS QTAM Core Storage Estimate
(Message Processing Program)

Basic QTAM Modules	11435
Terminal Test Module	1378
Checkpoint Module	1190
Operator Control Module	3024
Macro Instr. & Subroutines	3650
User Code	2000
Channel Programs	1034
Control Blocks and Tables	8638
Checkpoint Area	2775
	35124

20 Lines
80 1050s (4 per Line)

DOS QTAM Core Storage Estimate
(Message Processing Program)

GET/PUT Macros & Subroutines	1014
Other Macros & Subroutines	2457
Work Areas	300
Control Blocks	164
	3935

** Plus User Coding **

TOTAL SYSTEM - USING DOS QTAM
[CORE STORAGE ESTIMATE]

- FOREGROUND 1 - QTAM MESSAGE CONTROL
  FOREGROUND 2 - QTAM MESSAGE PROCESSING
  BACKGROUND - BATCH
- 4 LINES AND 16 1050 TERMINALS

SUPERVISOR	1D240
BACKGROUND PROGRAM	14336
QTAM MESSAGE CONTROL	21310
BUFFER POOL	1272
QTAM MESSAGE PROCESSING	3935
USER CODE FOR MESSAGE HANDLING	7000
	56093

MAJOR REASONS FOR SELECTING QTAM OVER BTAM

- QTAM QUEUING TECHNIQUE
- HEADER ANALYSIS
- ERROR MESSAGE HANDLING FACILITIES

  ERRMSG
  CANCELM
  REROUTE
  INTERCPT

- TASK MANAGEMENT

  RD/WR TO COMMUNICATION LINES
  RD/WR TO DISK QUEUES
  WRITE TO TAPE OR DISK LOG
  ACQUIRE CORE BUFFERS FROM POOL
  RELEASE CORE BUFFERS TO POOL
  BUILD CHANNEL PROGRAMS
  PASS MESSAGES TO MESSAGE PROCESSING PROGRAM
  RECEIVE MESSAGES FROM MESSAGE PROCESSING PROGRAM
  PERFORM MESSAGE HEADER ANALYSIS

- CHECKPOINT/RESTART

DISK OPERATING SYSTEM

SUPPORT FOR COMMUNICATION TERMINALS

BTAM

BASIC TELECOMMUNICATIONS ACCESS METHOD FOR HIGH SPEED AND LOW SPEED TERMINALS

QTAM

QUEUED TELECOMMUNICATIONS ACCESS METHOD - A COMPLETE MESSAGE HANDLING LANGUAGE FOR LOW SPEED TERMINALS

Presented by Don Moeller
IBM Corp
Chicago Systems Design & Installation Center
80 E. Lake St., Chicago Ill.

Session MON D1

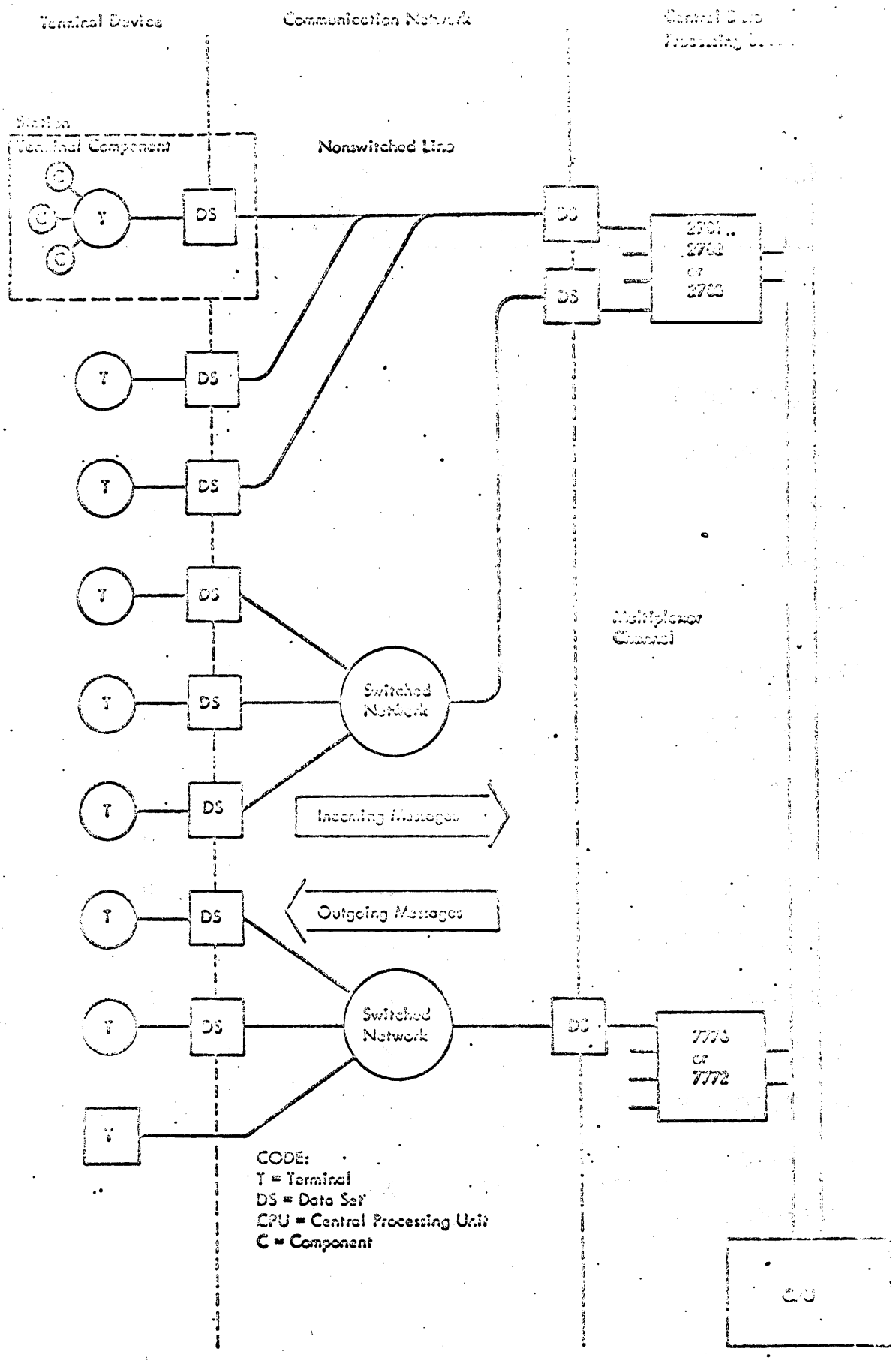


Figure 1. Configuration of a Teleprocessing Installation

BTAM
TERMINAL SUPPORT

TERMINAL	DISK OPERATING SYSTEM	OPERATING SYSTEM
1001	X	
1030	X	X
1050 (SWITCHED)	X	X
1050 (NON-SWITCHED)	X	X
1060	X	X
1092/1093	X	
1130	X	X
2260 (LOCAL)	X	
2260 (REMOTE)	X	X
2740 I & II	X	X
2780	X	X
S/360 MODEL 20	X	X
S/360 MODELS 30 THRU 75	X	X
AT & T 83B3	X	X
W/U 115A	X	X
AT & T 33/35 (TWX)	X	X
TOUCH-TONE PHONES TOUCH-CALLING	X	

BTAM CAPABILITIES

- TERMINAL POLLING
- MESSAGE RECEIVING
- TERMINAL ADDRESSING
- MESSAGE SENDING
- TERMINAL DIALING
- CALL ANSWERING
- ERROR DETECTION & RECOVERY
- ON LINE TERMINAL TESTING
- TERMINAL & LINE ERROR COUNTS
- TERMINAL LIST MODIFICATION (AT OBJECT TIME)
- BUFFER POOL MANAGEMENT
- CODE TRANSLATION
- CPU TIME SHARING WITH OTHER PARTITIONS

User Responsibilities

- Message Processing
- Queuing
- Logging
- Header Analysis
- Time & Date Stamping
- Handling of Undeliverable Messages
- Overlapping of T/P Program Functions
- Communication Between Partitions

BTAM MACROS

◇ Declarative Macros

BTMOD DFTRMLST
DTFBT LERB
ASMTRTAB

◇ Transient Macros

OPEN CLOSE
LOPEN

◇ Data Handling Macros

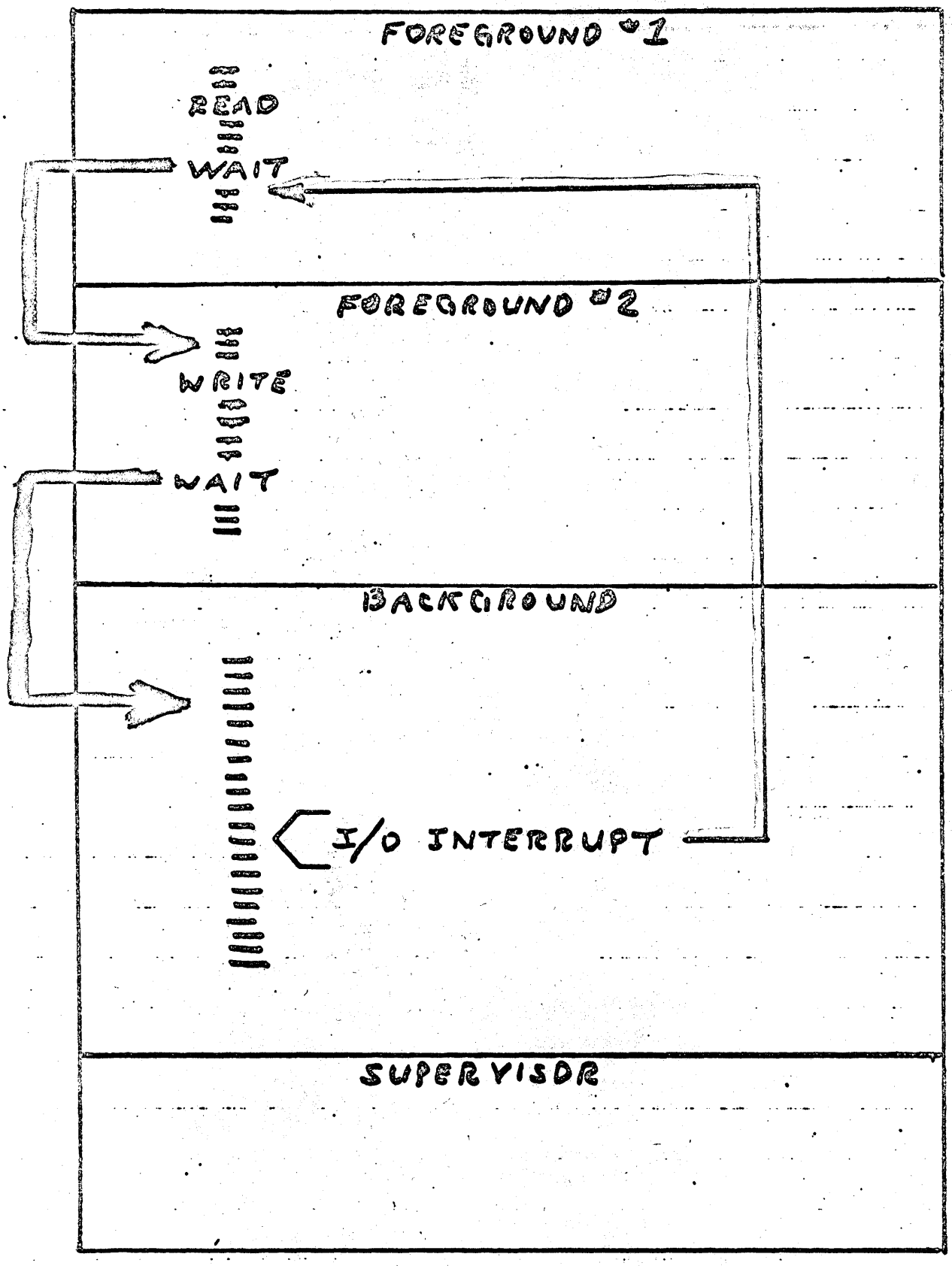
READ WAIT
WRITE TWAIT
CONTROL LERPRT
CHGNTRY TRANSLATE
RESETPL

◇ Buffer Management Macros

REQBUF RELBUF

TOTAL = 19

UTAM IN A MULTIPROGRAMMING ENVIRONMENT



BTAM OPERATION ON MULTIPLE LINES

READ ON LINE #1

READ ON LINE #2

WRITE ON LINE #3

READ ON LINE #4 — EOM — NEXT READ

START NEXT READ ON LINE #4

PROCESS INPUT DATA

WRITE ON LINE #5

WRITE ON LINE #6

READ ON LINE #7

BACKGROUND PROCESSING

BACKGROUND PROCESSING

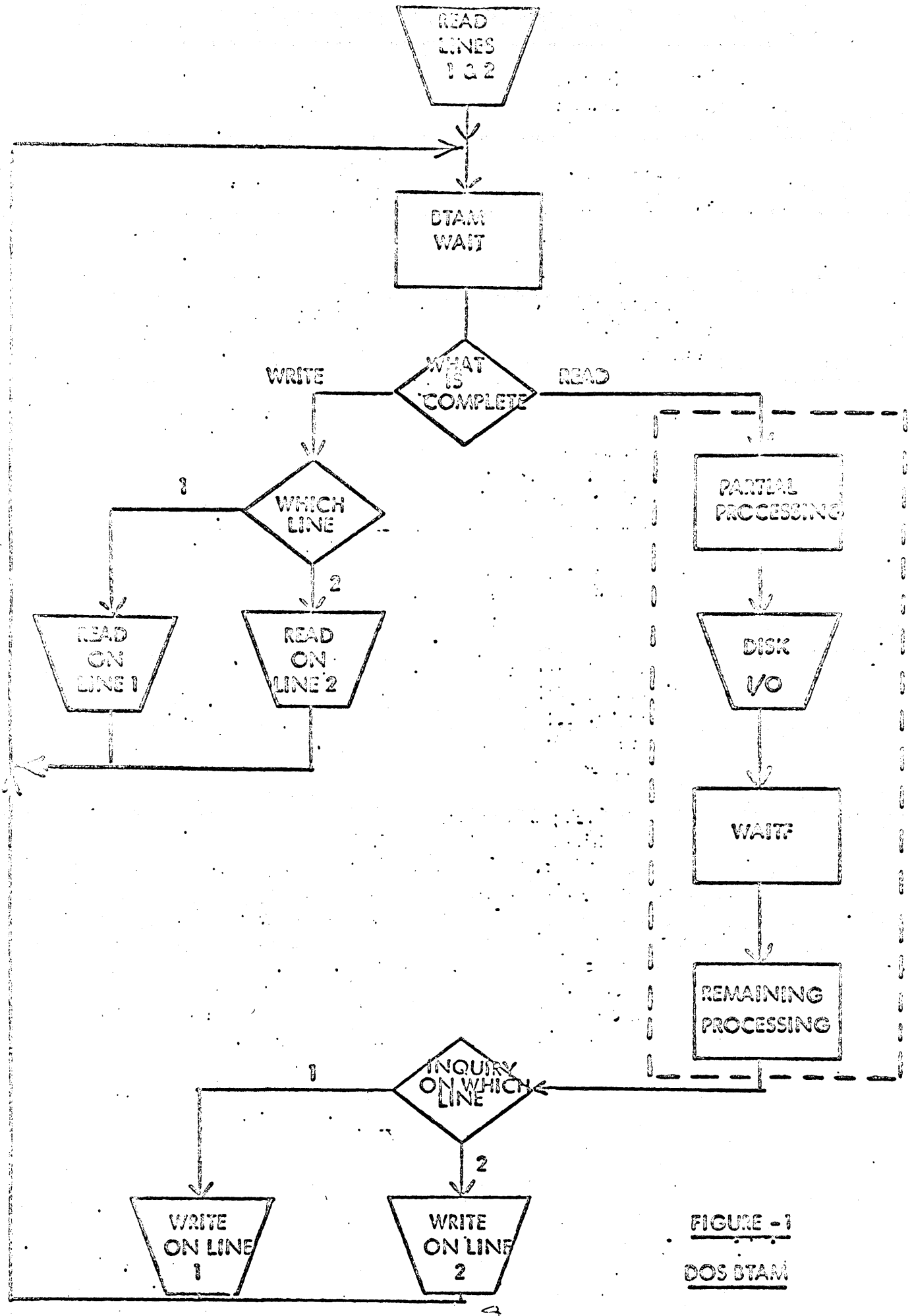
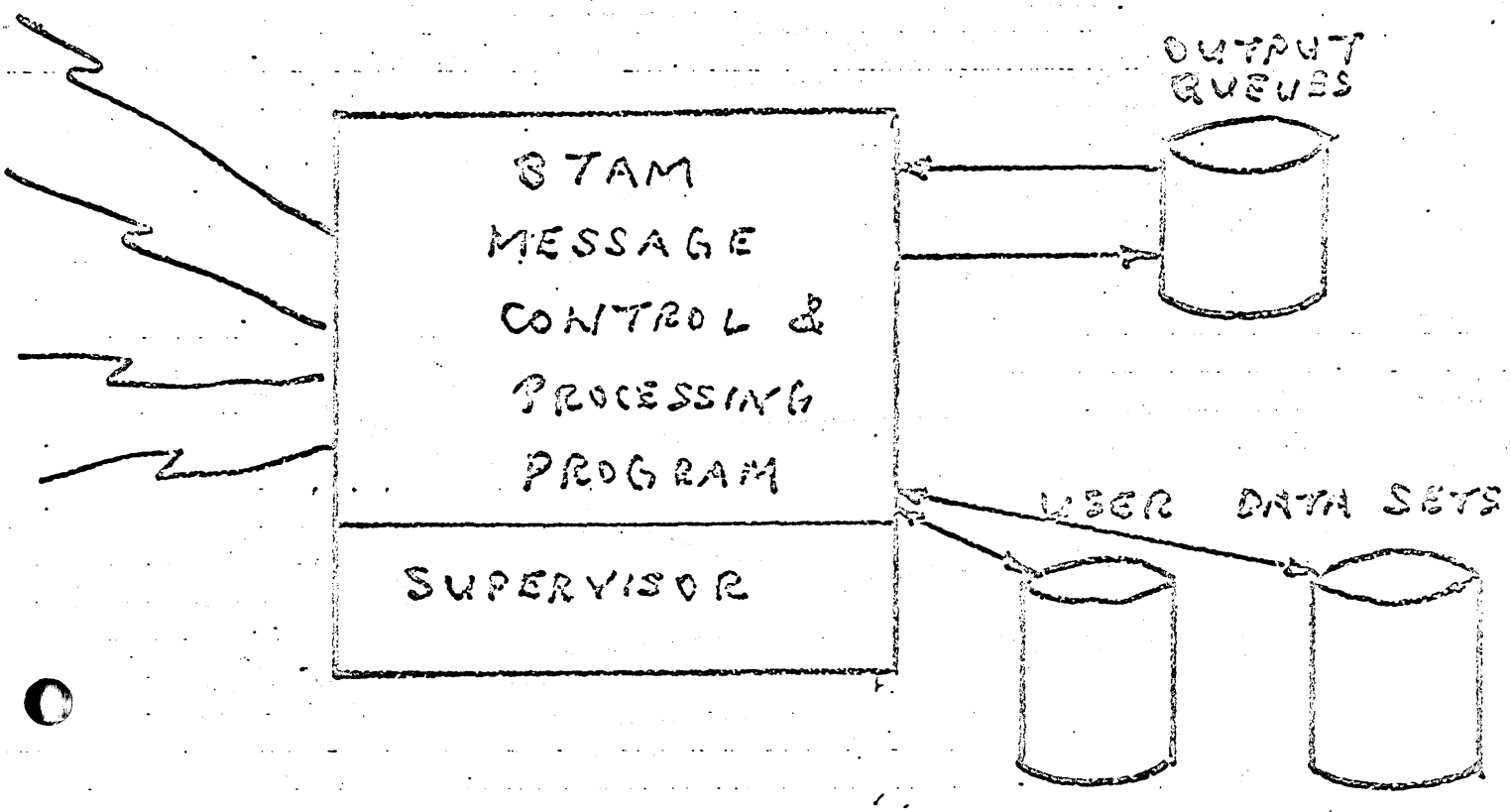


FIGURE - 1

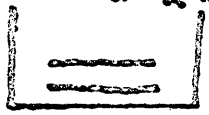
DOS BTAM

BTAM OPERATING ENVIRONMENT #1



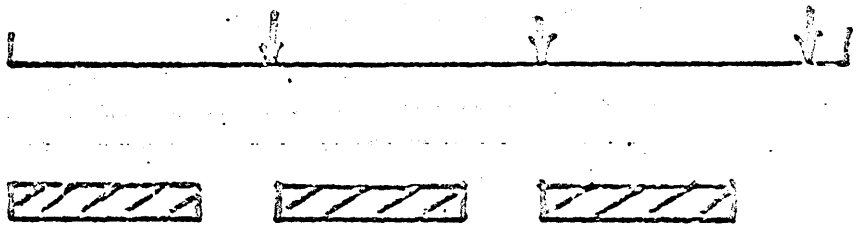
OK

TERMINAL MSG. QUEUE



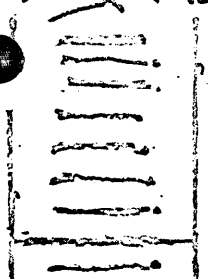
MSG. ARRIVAL RATE

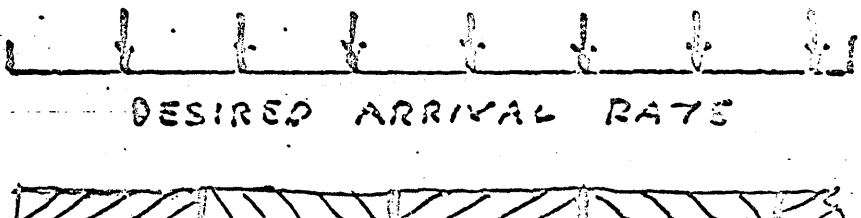
MSG. PROCESS TIME



BAD

TERMINAL MSG. QUEUE

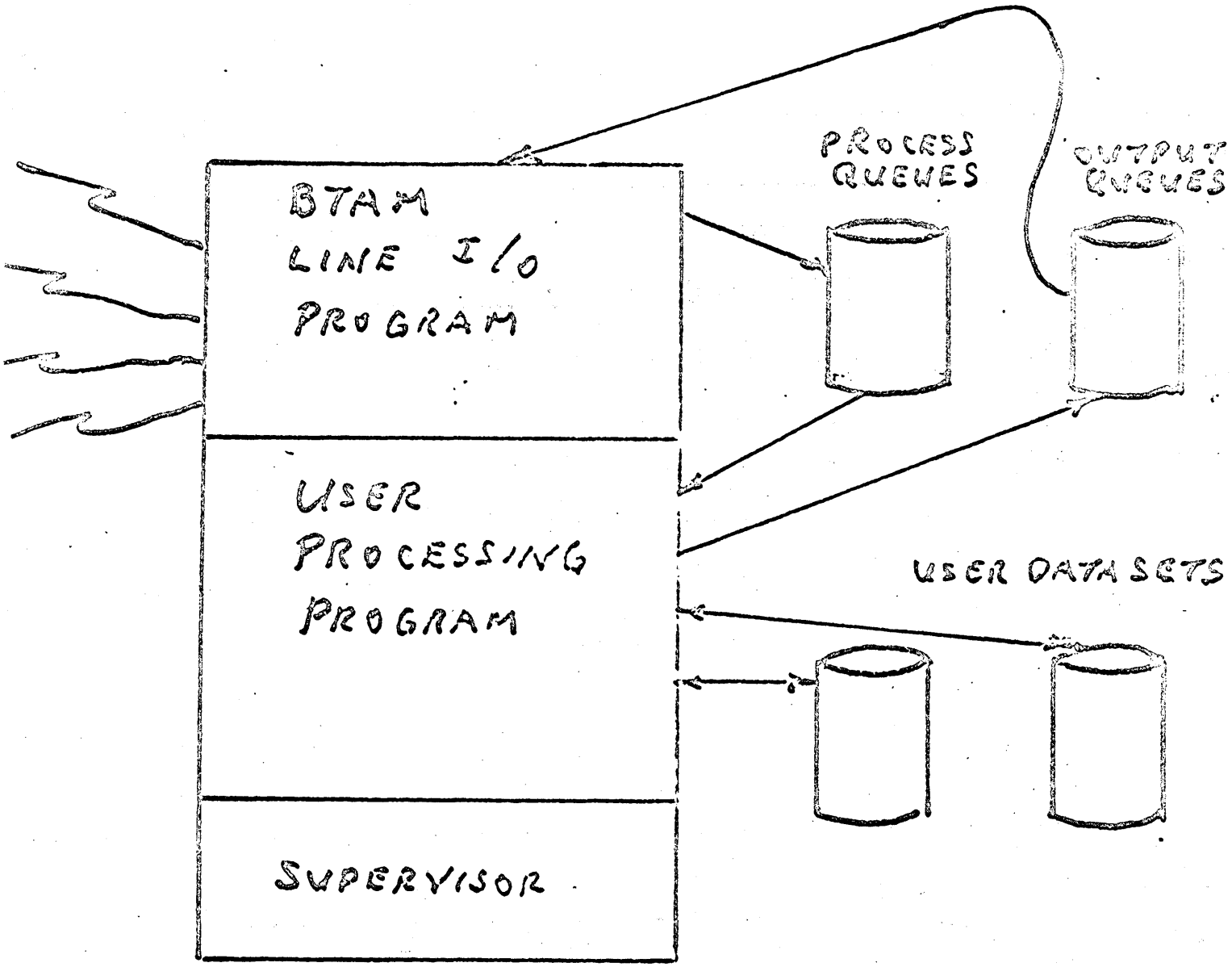




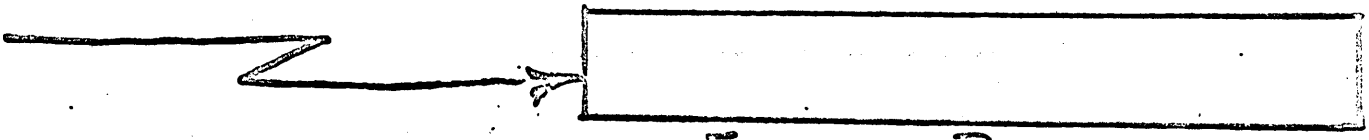
DESIRED ARRIVAL RATE

PROCESS TIME

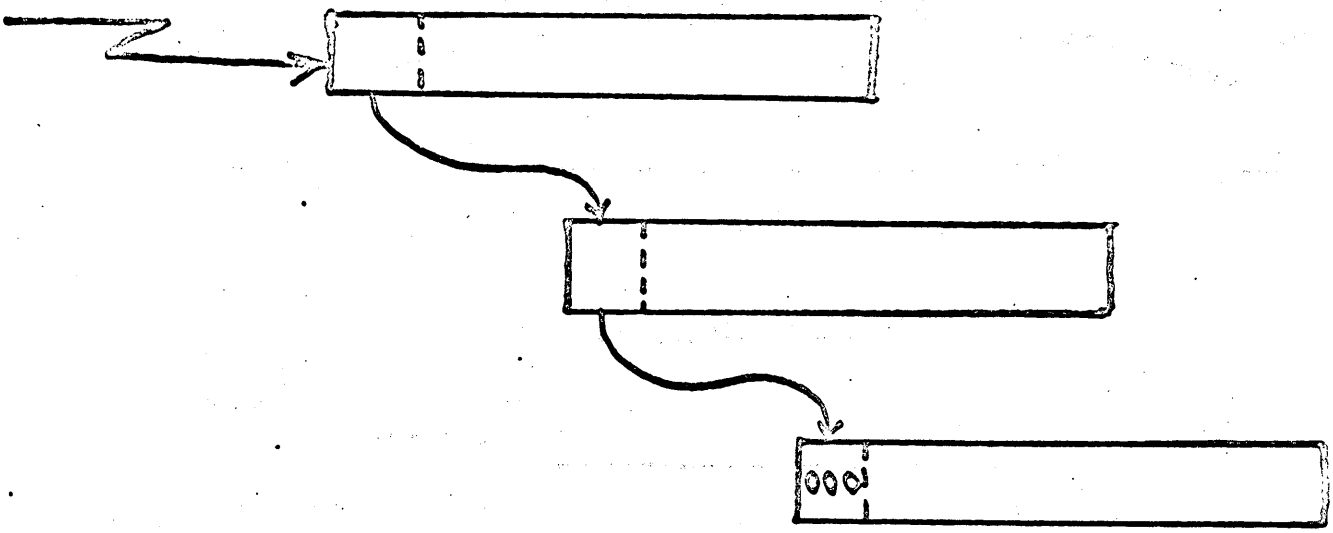
BTAM OPERATING ENVIRONMENT #2



BTAM Buffering Techniques

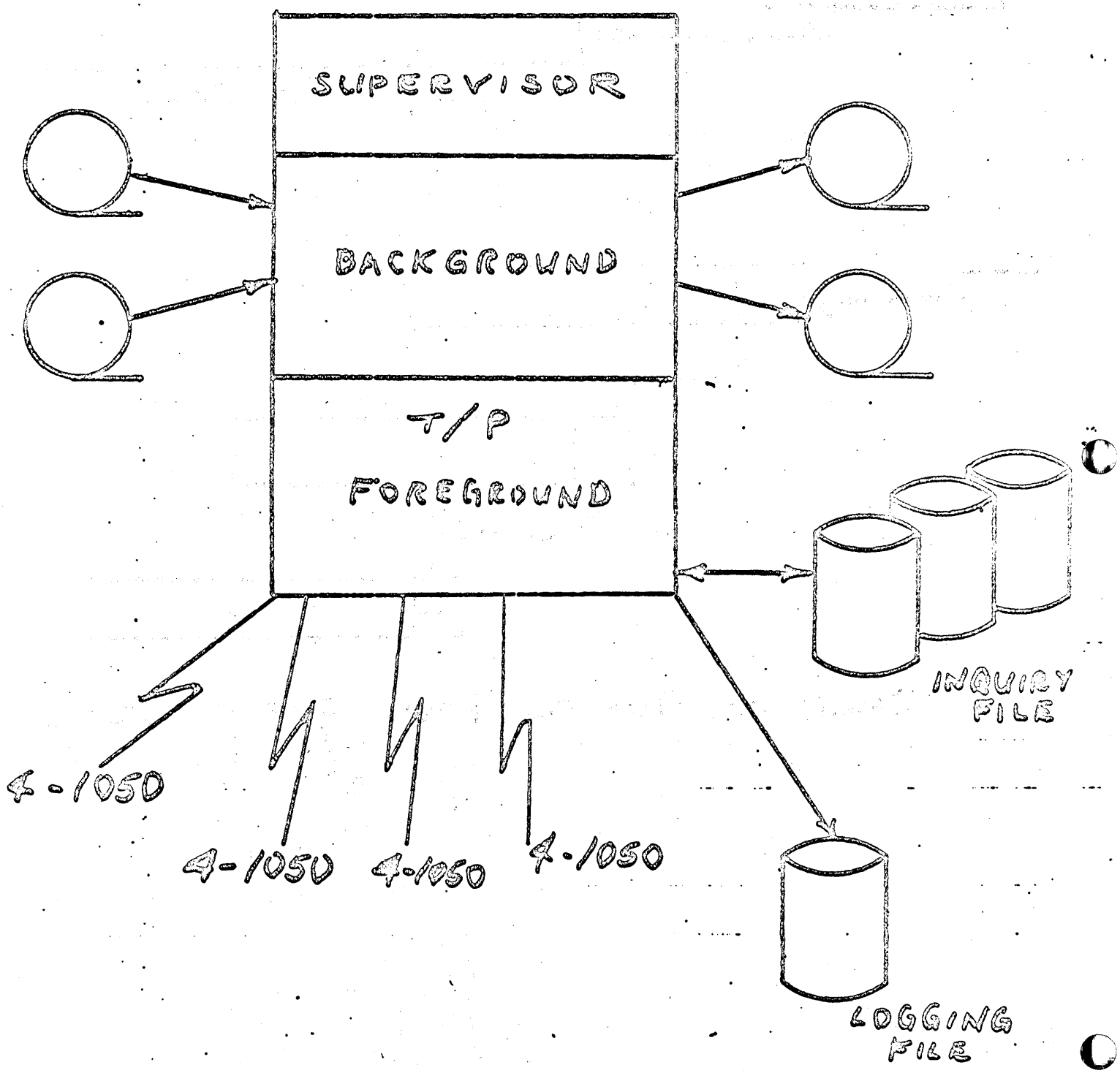


Fixed Buffer



Chain of Buffers from Buffer Pool

CONFIGURATION FOR STORAGE ESTIMATES



BTAM CORE STORAGE ESTIMATE
(DOS)

BTMOD (BASIC MODULE)	2900
ERROR CORRECTION	2070
ERROR COUNTS	190
TERMINAL TESTS	690
MODEL CHANNEL PROGRAMS	84
CONTROL BLOCKS	600
BTAM MACRO INSTRUCTIONS	240
POLLING LISTS	56
ADDRESSING LISTS	60
	6940

4 LINES
16 1050's (4 PER LINE)

BTAM CORE STORAGE ESTIMATE
(DOS)

BTMOD (BASIC MODULE)	2900
ERROR CORRECTION	2070
ERROR COUNTS	190
TERMINAL TESTS	690
MODEL CHANNEL PROGRAMS	84
CONTROL BLOCKS	1240
BTAM MACRO INSTRUCTIONS	400
POLLING LISTS	200
ADDRESSING LISTS	400
BUFFER MANAGEMENT	1470
	9834

20 LINES
80 1050's (4 PER LINE)

TOTAL SYSTEM - USING DOS BTAM
[CORE STORAGE ESTIMATE]

- FOREGROUND (BTAM) PLUS BACKGROUND (BATCH)
- 4 LINES AND 16 1050 TERMINALS

SUPERVISOR	8192
BACKGROUND PROGRAM	14336
BTAM	6940
IND. SEQ. LOGICAL IOCS	1190
SEQUENTIAL LOGICAL IOCS	820
IND. SEQ. I/O AREA	400
SEQUENTIAL I/O AREAS (2)	800
COMMUNICATION LINE BUFFERS (4)	520
INQUIRY PROCESSING PROGRAM	7000
	40198

QTAM
TERMINAL SUPPORT

TERMINAL	DISK OPERATING SYSTEM	OPERATING SYSTEM
1001	X	
1030	X	X
1050 (SWITCHED)	X	X
1050 (NON-SWITCHED)	X	X
1060	X	X
1092/1093	X	
1130		
2260 (LOCAL)		
2260 (REMOTE)	X	X
2740 I & II	X	X
2780		
S/360 MODEL 20		
S/360 MODELS 40 THRU 75		
AT & T 8333	X	X
W/U 115A	X	X
AT & T 33/35 (TWX)	X	X
TOUCH-TONE PHONES TOUCH-CALLING	X	

BTAM- QTAM CAPABILITIES

	BTAM	QTAM
• TERMINAL POLLING	X	X
• MESSAGE RECEIVING	X	X
• TERMINAL ADDRESSING	X	X
• MESSAGE SENDING	X	X
• BUFFER POOL MANAGEMENT	X	X
• "DYNAMIC" BUFFERING		X
• DIALING A TERMINAL	X	X
• ANSWERING A CALL ON DIAL FACILITIES	X	X
• CODE TRANSLATION	X	X
• ERROR DETECTION AND CORRECTION	X	X
• TERMINAL TESTS	X	X
• MAINTENANCE OF ERROR STATISTICS	X	X
• CHECKPOINT FOR COMMUNICATION LINES		X
• NETWORK CONTROL TERMINAL		X
• TIME AND DATE STAMPING		X
• MESSAGE SEQUENCE CHECKING		X
• MESSAGE TYPE RECOGNITION		X
• QUEUING OF MESSAGES ON DISK OR IN CORE STORAGE		X
• LOGGING OF MESSAGES		X
• INTERCEPTION OF MESSAGES FOR AN OUT OF SERVICE TERMINAL		X
• REROUTING OF UNDELIVERABLE MESSAGES		X
• ERROR NOTIFICATION TO TERMINALS		X
• CANCELLATION OF ERROR MESSAGES		X

QTAM MACRO LANGUAGE

- GROUP 1.
  I/O DEFINITION, INITIATION, TERMINATION - 8

- GROUP 2.
  CONTROL BLOCK, TABLE & BUFFER DEFINITION - 12

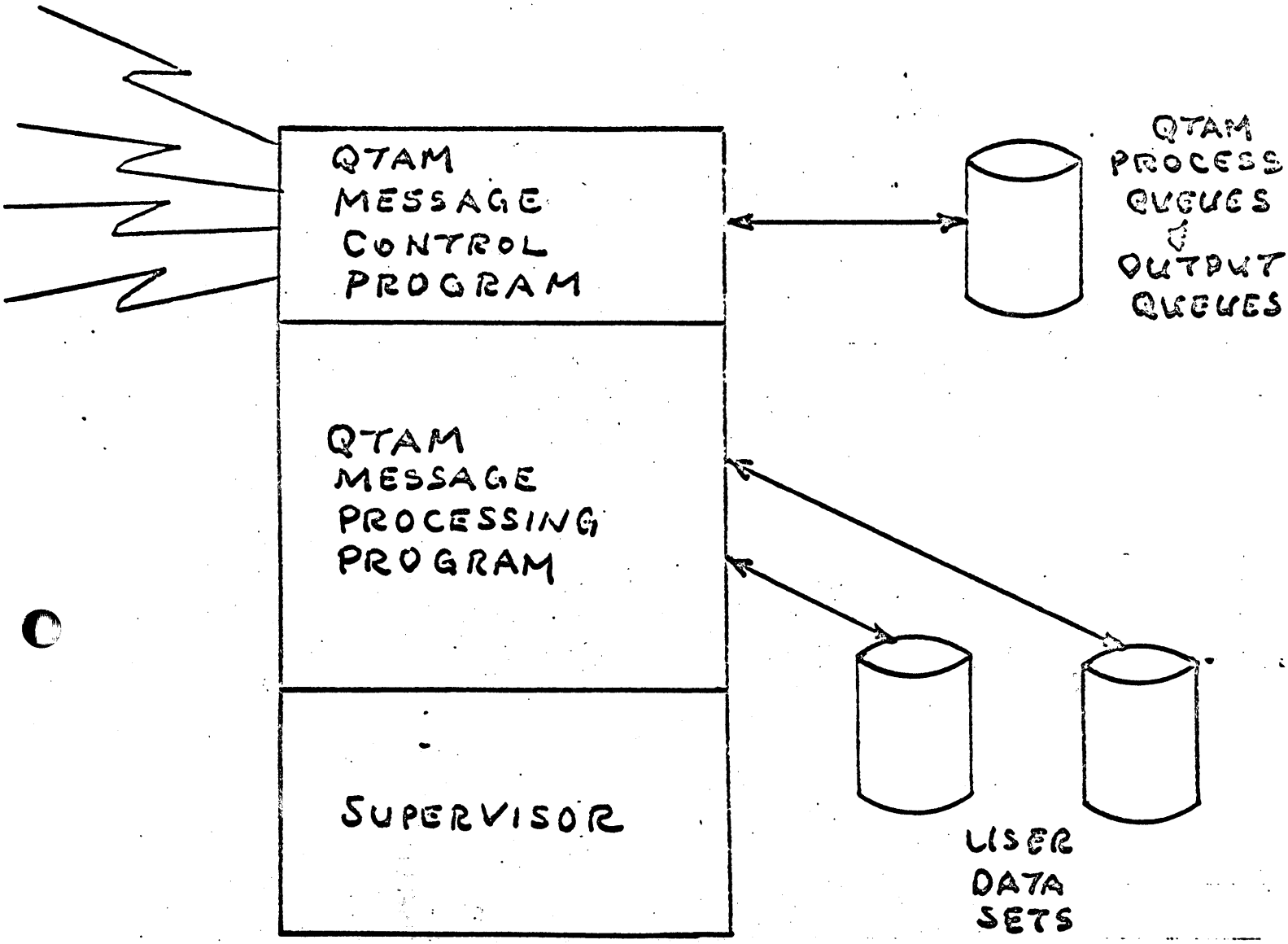
- GROUP 3.
  MESSAGE HANDLING - 27

- GROUP 4.
  DELIMITERS - 10

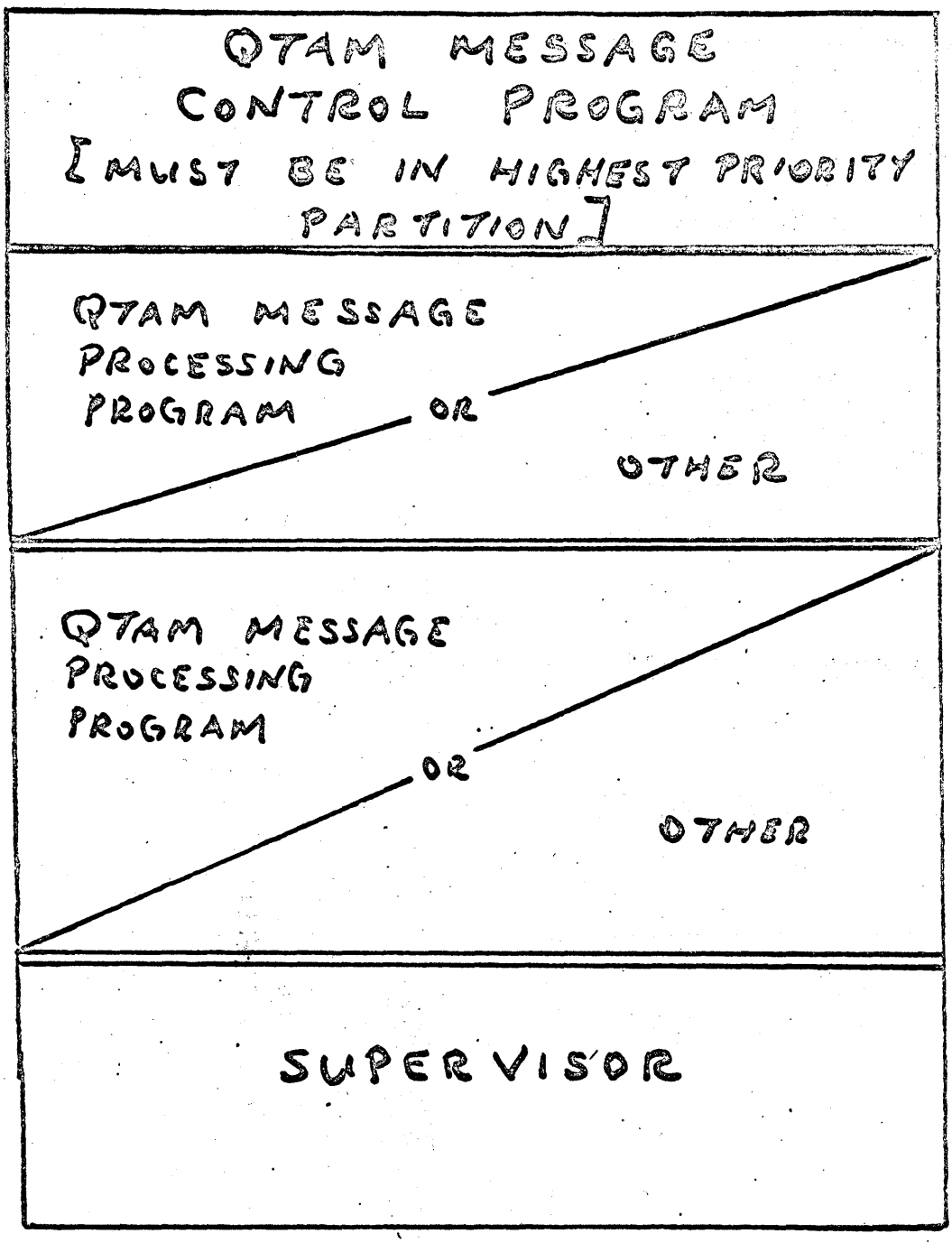
- GROUP 5
  NETWORK CONTROL - 13

TOTAL = 70

QTAM OPERATING ENVIRONMENT

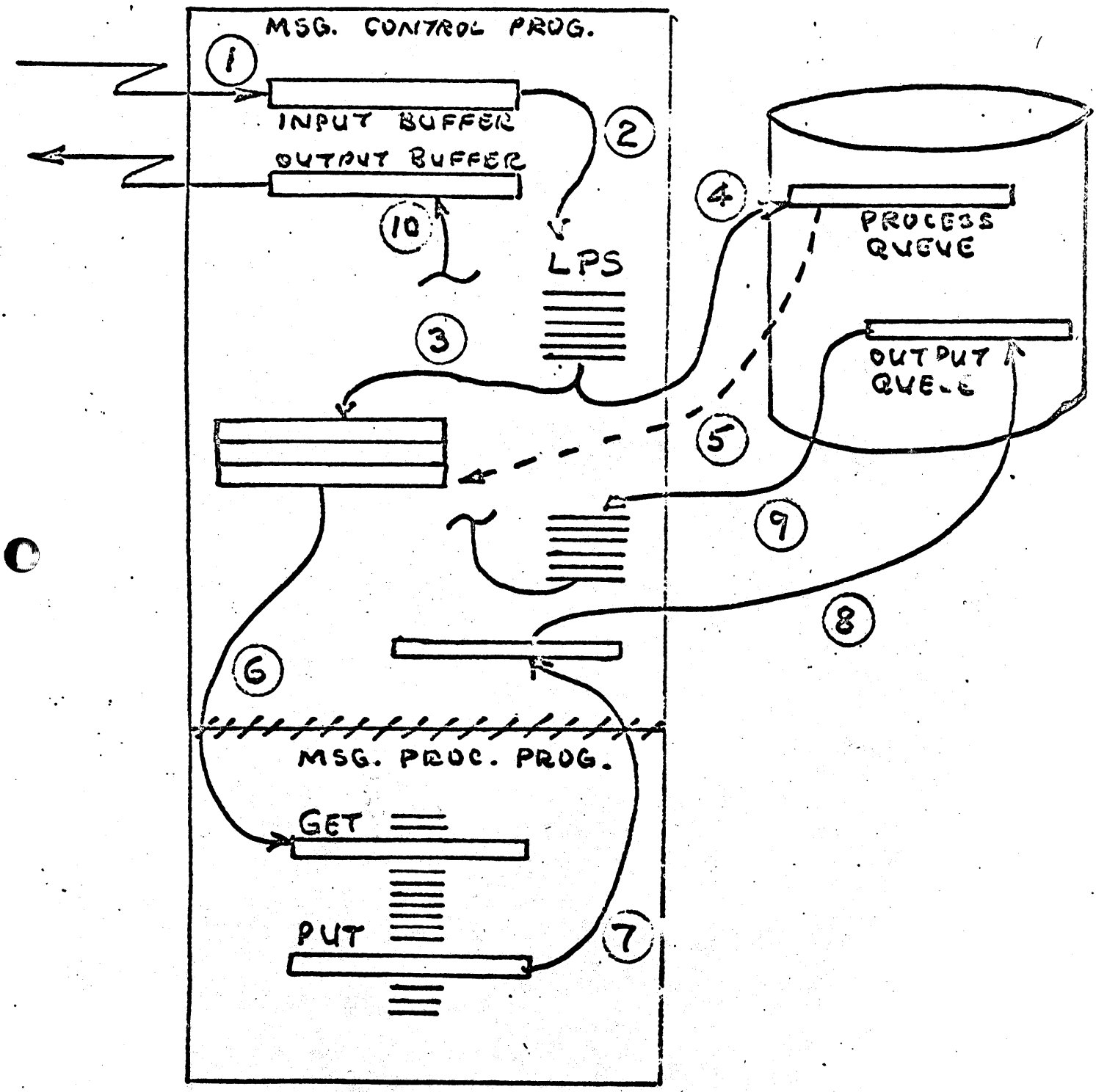


QTAM USE OF PARTITIONS

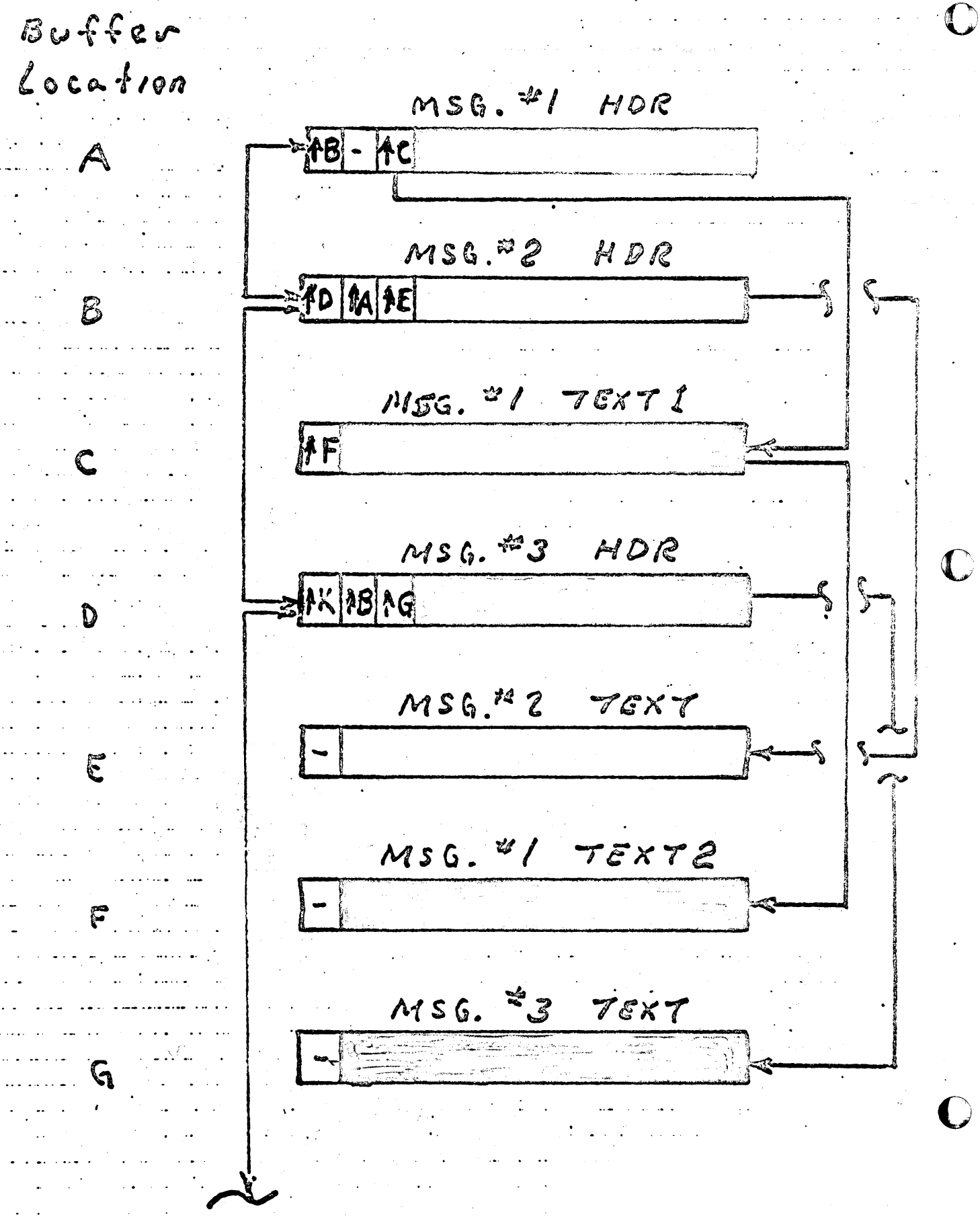


* QTAM MESSAGE PROCESSING PROGRAM MUST APPEAR IN ONE OF THESE PARTITIONS WHEN TERMINATING THE MESSAGE CONTROL PROGRAM.

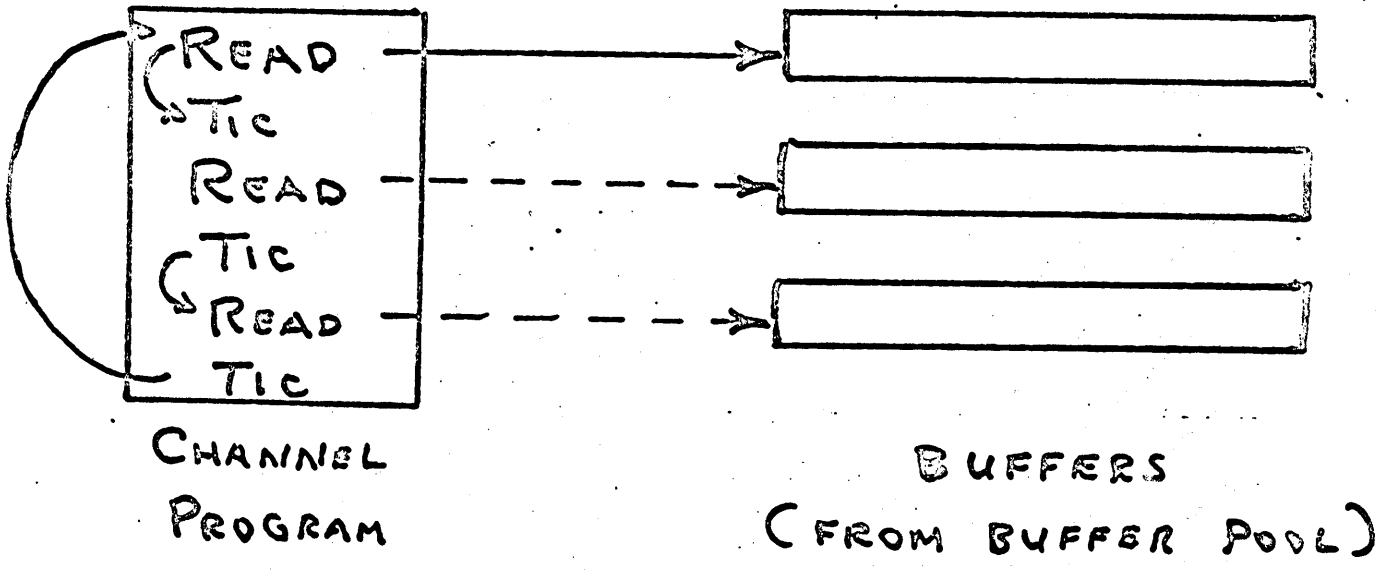
QTAM MESSAGE FLOW



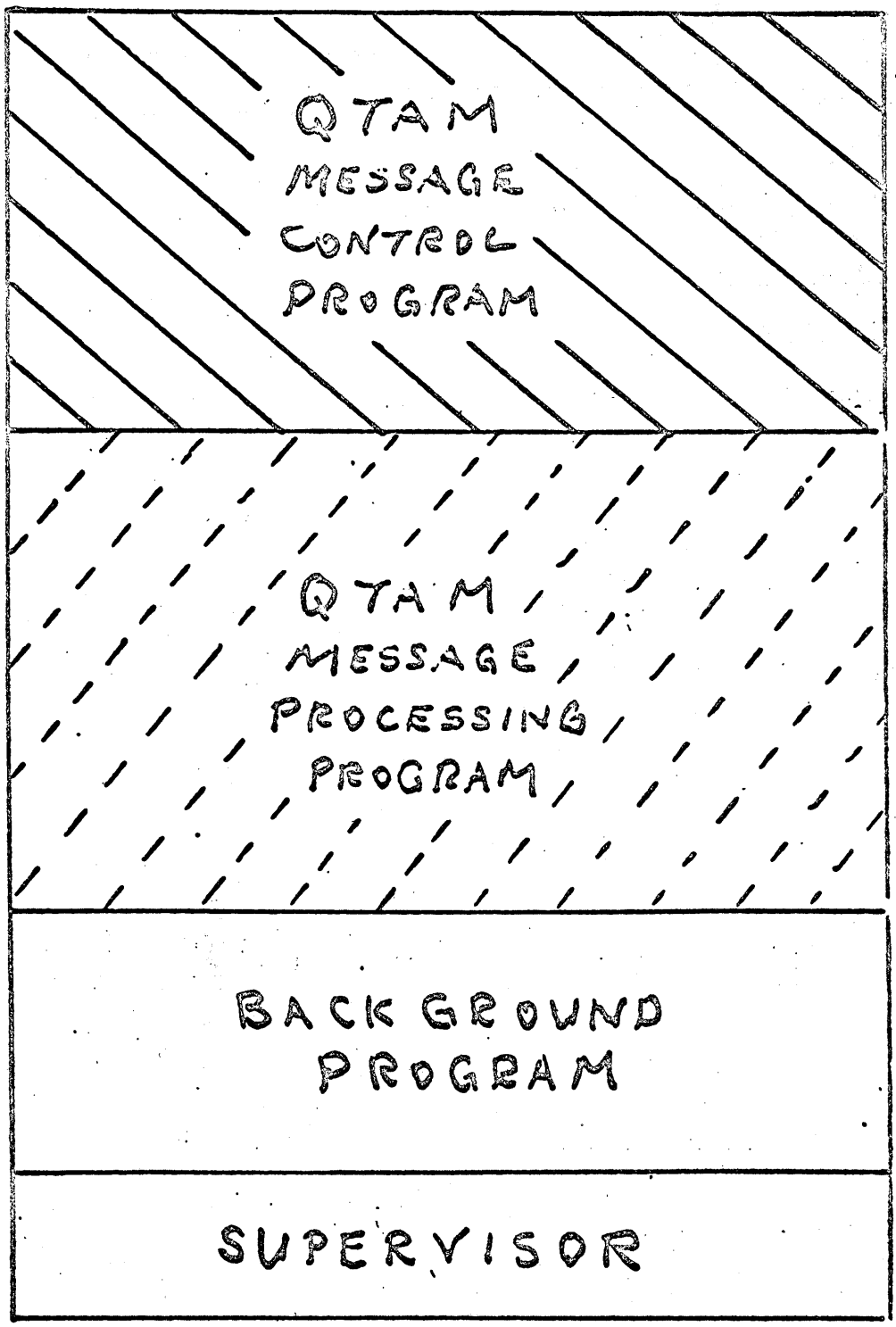
QTAM QUEUING



QTAM Dynamic Buffering



QTAM Environment For Core Storage Estimates



DOS QTAM Core Storage Estimate
(Message Processing Program)

Basic QTAM Modules	11435
Terminal Test Module	1378
Checkpoint Module	1190
Operator Control Module	3024
Macro Instr. & Subroutines	3650
User Code	2000
Channel Programs	1034
Control Blocks and Tables	8638
Checkpoint Area	2775
	35124

20 Lines
80 1050s (4 per Line)

DOS QTAM Core Storage Estimate
(Message Control Program)

Basic QTAM Modules	11435
Terminal Test Module	1378
Checkpoint Module	1190
Macro Instr. & Subroutines	3500
User Code	1000
Channel Programs	266
Control Blocks and Tables	1910
Checkpoint Area	631
	21310

4 Lines
16 1050s (4 per Line)

DOS QTAM Core Storage Estimate
(Message Processing Program)

GET/PUT Macros & Subroutines	1014
Other Macros & Subroutines	2457
Work Areas	300
Control Blocks	164
	3935

** Plus User Coding **

TOTAL SYSTEM - USING DOS QTAM
[CORE STORAGE ESTIMATE]

- FOREGROUND 1 - QTAM MESSAGE CONTROL
  FOREGROUND 2 - QTAM MESSAGE PROCESSING
  BACKGROUND - BATCH
- 4 LINES AND 16 1050 TERMINALS

SUPERVISOR	8192
BACKGROUND PROGRAM	14336
[QTAM MESSAGE CONTROL	21310
[BUFFER POOL	1272
[QTAM MESSAGE PROCESSING	3935
[USER CODE FOR MESSAGE HANDLING	7000
	56045

APPLICATIONS FOR BTAM AND QTAM

THE FOLLOWING TABLE ANSWERS QUESTIONS REGARDING THE ABILITY OF BTAM AND QTAM WITH OR WITHOUT USER WRITTEN PROCESSING ROUTINES TO SATISFY THE REQUIREMENTS OF VARIOUS T/P APPLICATIONS.

APPLICATION TYPE	PROGRAMMING APPROACH			
	BTAM ALONE	BTAM AND USER PROCESSING	QTAM ALONE	QTAM AND USER PROCESSING
DATA COLLECTION	NO	YES	YES	YES
MESSAGE SWITCHING	NO	YES	YES	YES
INQUIRY	NO	YES	NO	YES
"REAL TIME" PROCESSING	NO	YES	NO	YES

* QTAM PLUS A USER PROCESSING PROGRAM REQUIRES TWO PARTITIONS UNDER DOS.

MAJOR REASONS FOR SELECTING QTAM OVER BTAM

- QTAM QUEUING TECHNIQUE
- HEADER ANALYSIS
- ERROR MESSAGE HANDLING FACILITIES

  ERRMSG
  CANCELM
  REROUTE
  INTERCPT

- TASK MANAGEMENT

  RD/WR TO COMMUNICATION LINES
  RD/WR TO DISK QUEUES
  WRITE TO TAPE OR DISK LOG
  ACQUIRE CORE BUFFERS FROM POOL
  RELEASE CORE BUFFERS TO POOL
  BUILD CHANNEL PROGRAMS
  PASS MESSAGES TO MESSAGE PROCESSING PROGRAM
  RECEIVE MESSAGES FROM MESSAGE PROCESSING PROGRAM
  PERFORM MESSAGE HEADER ANALYSIS

- CHECKPOINT/RESTART

SESSION REPORT

COMMON - Chicago

Session Number MON D2 Session Name 1800-360 Communication

Chairman R. W. Page

Time 3:30 Attendance (No.) 100

Speakers Wayne Barnes - Pacific Gas and Electric Company

Synopsis of Meeting Mr. Barnes gave an excellent talk on 1800-360 Remote Job Entry. His system runs under TASK (Non-Process Mode) but could be run under TSX with slight modification. The session paper gives details on application and implementation including much of the coding.

An IBM 1800 - S/360 Remote Job Entry System

Presented at the

Chicago Meeting of COMMON

April 8-10, 1968

by

Wayne R. Barnes

Computer Systems Analyst

Pacific Gas and Electric Company

245 Market Street

San Francisco, California

LIST OF FIGURES

TABLE OF CONTENTS

I INTRODUCTION

The concept of entering data into a computer from a remote location is not new to the computer industry. Many early techniques were successfully developed, but in most cases they were not economically feasible. It is only during the last year or so that improvements in the field of multi-programming together with advancements in computer communication techniques have become advanced enough to make remote job entry economically attractive. Before describing to you the remote job entry system that was developed at Pacific Gas and Electric Company (P. G. and E.), I would like to briefly describe our facilities and relate a few pertinent statistics about our Company.

Pacific Gas and Electric Company is one of the largest combined gas and electric utilities in the country. Figure 1 is a map of our service area. As can be seen, it stretches from the Oregon border, south to Bakersfield, east to the Sierras and west to the Coast. This service area includes 47 counties and 94,000 square miles of territory. We service a total of approximately eight million people and have 22,000 employees.

Because of this far flung service area and the trend toward centralization of data files, remote job entry to a large central computer system is becoming an extremely important consideration in the planning and development of Company computer resources.

II BACKGROUND OF P. G. and E. COMPUTER SYSTEMS

EARLY SYSTEMS

The use of digital computers at P. G. and E. has evolved in the traditional manner of many large business corporations. The first use of a large scale in-house computing system was for data processing and was primarily dedicated to such applications as billing and general accounting functions. Early systems at P. G. and E. were therefore configured along the lines of the 705 - 7080 series computers with 1401's for peripheral support. Early attempts by our engineering and scientific personnel to make extensive use of these types of systems for "mathematical" calculations proved uneconomical. It was considered far more economical to send the Company's scientific and engineering type applications to service bureaus which on a contract basis leased time on their scientific type computers. I am sure that this sounds familar to many of you since this was a common practice in the late 1950's and early 1960's.

While our engineering and scientific computer requirement increased significantly over the years, it never reached the point where it could convincingly support the expense of a large scale stand alone installation for scientific computing.

SYSTEM 360/65

The procurement by our Company of a System 360 Model 65 temporarily reduced our dependence on outside computer time. This computer system which is shown in figure 3 can economically handle both scientific and

commercial applications. It has not, however, fully solved the problem of accommodating the needs of engineers who are located significant distances away from the central computer complex. In order for our Company engineers to make use of the System 360/65 it was necessary for them to use a courier service to pick up and deliver the work to the computer center. There soon developed a significant material handling problem which greatly reduced the "turn-around time" for jobs being submitted to the computer.

1800 COMPUTER SYSTEM

It was decided after a review of several alternatives that the lease of an appropriate satellite computer with provisions for remote job entry to our System 360 would significantly improve turn-around service to users, reduce our dependence further on outside computer services and most importantly alleviate a serious material handling problem. We selected an IBM 1800 computer for this task for the following reasons:

1. It can compile and execute Fortran programs.

2. It can support many I/O devices including paper tape, magnetic tape (1130 systems did not have this capability at the time of evaluation) and an incremental plotter.

3. It can interface with the System 360 via standard telecommunication lines.

4. It has the facilities for the possible development of a conversational time sharing system.

5. It is economically suitable.

Figure 2 is a schematic diagram of our 1800 computer system which consists of the following equipment:

UNIT	MODEL	DESCRIPTION
1802		C.P.U., 16K, 4 microseconds
2310	A1	Disk Storage - 512K
1442	6	Card Read-Punch - Read @ 300 cpm Punch @ 60 cpm
1443	2	Line Printer - 240 lpm
1816	1	Console - Keyboard
1627	2	Incremental plotter - 12" or 30"
--	-	Paper Tape Reader - 700 Chars/sec
--	-	Paper Tape Punch - 160 Chars/sec
2401	1	7 Track Magnetic Tape Drive - 30KC
2401	1	9 Track Magnetic Tape Drive - 30KC
2701	1	Data Adapter Unit

The 1800 computer system is presently being used by all technical computer users in the Company. Small and medium size applications and those requiring unique computer output facilities are processed directly on the 1800. The larger, production oriented applications requiring the higher power and software elegance of the System/360 are being transmitted to the System/360 by way of our remote job entry system. Figure 4 shows how this is accomplished schematically. It should be noted that the 1800 and System/360 communicate with one another over a 2400 BAUD, full duplex, voice grade communication line.

III REMOTE JOB ENTRY PROGRAMS FOR THE 1800

Our 1800 computer system is currently using the TASK operating system. Our programs are executed in an off-line mode by the non-process monitor. In order to develop a remote job entry system on the 1800 it was necessary to develop our own software routines for the 2701 Data Adapter. The routines that we developed could be easily modified to operate under TSX or MPX and probably, in some cases, be easier to implement. The routines for our remote job entry system on the 1800 consist of the following programs:

1800 TPSTR PROGRAM

TPSTR is an 1800 subroutine that provides interface between the 2701 STR device and the 1800 system. This routine is written in 1800 assembly language. A logic chart for this routine is shown in figure 6 and a complete program listing in figure 9. TPSTR is called via a standard LIBF and contains the following features.

1. Opens line for receiving or transmitting.
2. Provides an adapter busy routine.
3. Provides all line control and I/O instructions.
4. Closes line upon receipt of proper control record.
5. Sets up an address in the TASK Interrupt Branch Table and provides an interrupt routine to test for proper execution of I/O commands.
6. Provides an error retry routine and control messages for the 1800 console operator.
7. Contains a program switch routine to drop invalid records without losing line control.
8. Can operate in either a normal or closed loop test mode.

1800 RECEIVE PROGRAM

Figure 7 is a logic chart of the P. G. and E. 1800 Receive routine. This routine was written in 1800 assembly language and performs the following tasks.

1. Reads records from the line via the 1800 TPSTR subroutine.
2. Contains a LINEPR subroutine that translates the characters from 4/8 code to EBCDIC.
3. Expands the condensed records in the buffer to the proper format.
4. Logs System/360 "//JOB" records on 1800 console for accounting purposes.
5. Decodes System/360 records to provide the proper 1443 carriage control and print functions.
6. Overlaps the line routine with the 1443 printing to provide maximum throughput.

1800 SEND PROGRAM

Figure 8 is a logic chart of the P. G. and E. 1800 Send routine. This routine was written in 1800 assembly language and performs the following tasks:

1. Reads input from the 1442 card reader.
2. Removes extraneous blanks from the right hand side of records based on the type of input.
3. Contains a HOL48 subroutine which translates characters from 1442 card code to 4/8 line code.
4. Places the condensed records in a 320 work buffer. (2 chars per word).

5. Modifies the CARDN routine to accept System/360 "//JOB" cards and also sets up the Mask and IOCC word in the TASK Interrupt I.D. Table.

6. Logs the System/360 "//JOB" records on the 1800 console for accounting purposes.

7. Sends records to the System/360 via the 1800 TPSTR subroutine and overlaps line transmission and card reading to provide maximum throughput.

IV REMOTE JOB ENTRY PROGRAMS FOR S/360

Programming requirements for the System/360 R.J.E. package were greatly reduced through the use of the IBM Synchronous Transmit-Receive Access Method (STRAM) routines. STRAM is a MACRO language at the assembler level that provides:

1. Environment definition
2. Line control
3. Data transmission
4. Buffer Management
5. Data translation
6. Error procedures

STRAM routines require O/S option 2/MFT release 11 or later.

It is the responsibility of the user to write a tailored program, using the STRAM MACROS, to handle his requirements. Figure 5 is a logic chart of our routine which was written in 360 assembler language to perform the following tasks:

S/360 RECEIVE SUBROUTINE

1. Reads a 640 byte input record from the line via the STRAM routines.

2. Translates from 4/8 code to the BCD subset of EDCDIC code.

3. Deblocks the buffer, expands the records to 80 chars., and creates a O/S 360 SYSIN tape.

4. Writes Message records on the S/360 console.

5. On receipt of proper code closes the line and closes the SYSIN tape.

S/360 TRANSMIT SUBROUTINE

1. Reads the standard SYSOUT tape.

2. Removes entraneous blanks from right hand side of the records.

3. Translate records to 4/8 line code.

4. Creates a 640 cyte buffer record and sends records to the 1800 via the STRAM routines.

5. Closes the line and the SYSOUT tape upon receipt of the proper control record.

V USER EXPERIENCE

Early attempts (Spring 1967) to implement the communications routines on the 1800 proved frustrating. The channel adapter for the 2701 is an R.P.Q. device which means, of course, that it is not supported by the 1800 software systems. The early documentation that we were able to obtain was inadequate and in some cases incorrect. Even the valiant efforts of our IBM S.E.'s and C.E.'s proved futile and it was necessary to obtain the

help of plant technicians before we were able to get "on the air." After we obtained the proper instruction on the use of the selector channel we were soon able to transmit and receive data. The only other major problem that we encountered was a problem in duplication of transmitted records. This problem was solved with a hardware fix to the 2701.

On the 360 side problems of implementation were minimal. The System 360 fully supports the 2701 Data Adapter and provides the STRAM routines. All that was necessary to make an effective package was to include our user written subroutines. In addition the use of MFT with partitioned memory insures full utilization of our System 360/65. If there is any criticism to be noted it would be in the area of operator instruction. Most of the early "failures" on the System 360 were caused because the operator did not have proper written instructions. Documentation in this area is very limited and standard operating procedures must be developed as soon as possible to insure effective results.

Overall our experience with our 1800 - 360 R.J.E. system has been favorable. Aside from the problems which I have described we have been communicating with the System/360 since the summer of 1967 with excellent results.

VI CONCLUSIONS AND RECOMMENDATIONS

1. Our 1800 - 360 Communication System does what it was intended to do. We are able to enter jobs for the System/360 through the 1800 computer and transmit them to the 360 for processing. We are also able to transmit the results from the 360 back to the 1800 where they can be printed or plotted.

2. All Hardware and Software, with regard to 1800 - 360 communication, is currently working satisfactorily. There are very few I/O errors and "turn-around time" for the System 360 jobs has been greatly improved.

3. Voice grade line speed (2400 BAUD or 300 Chars/sec) is inadequate for high volume I/O in a remote job entry system.

4. Programming for I/O devices attached to the 1800 selector channel is not compatible with System/360 programming for the same devices.

5. It is not economically feasible to print all the 360 generated output on the 1800 - 1443 printer. Any job in which "turn-around time" is not a critical factor should use the 1800 - 360 R.J.E. system only to transmit the input data to the System/360.

6. Documentation of the IBM selector channel R.P.Q . C08037 should be revised. Our major problem in developing this system was caused by lack of the proper technical information on the use of the selector channel.

7. If the normal 1443 printer workload for the 1800 is quite heavy there is a need for an additional 1443 to be added to the system which could be dedicated to the printing of 360 output. This could also apply to 1442 usage.

8. The user should develop efficient record compacting routines in order to eliminate the transmission of unnecessary blanks.

9. Operator training for both the 1800 and the System/360 is essential. Console messages to the operators that can be placed in the normal job stream are extremely vital and should be used frequently.

10. Proper communication between the IBM C.E.'s and the telephone company technicians must be established early in order to avoid possible system failures do to "misunderstandings."

ACKNOWLEDGEMENTS

The Author wishes to acknowledge the following individuals for providing considerable advice and assistance in the design and development of this project.

Mr. G. A. Maneatis, Chief Computer Application Engineer
Pacific Gas and Electric Company

Mr. W. M. Pickslay, Jr., Chief Computer Application Engineer (Retired)
Pacific Gas and Electric Company

and

Mr. H. U. Brown III IBM Corporation

Mr. D. E. Eesley IBM Corporation

Mr. W. J. Fant, Jr. IBM Corporation

W. R. BARNES
3/11/68

WRB:tic

ELECTRIC SYSTEM

PACIFIC GAS AND ELECTRIC COMPANY
DECEMBER 31, 1966

THE COMPANY OPERATES AN INTEGRATED ELECTRIC SYSTEM EXTENDING INTO 46 CALIFORNIA COUNTIES

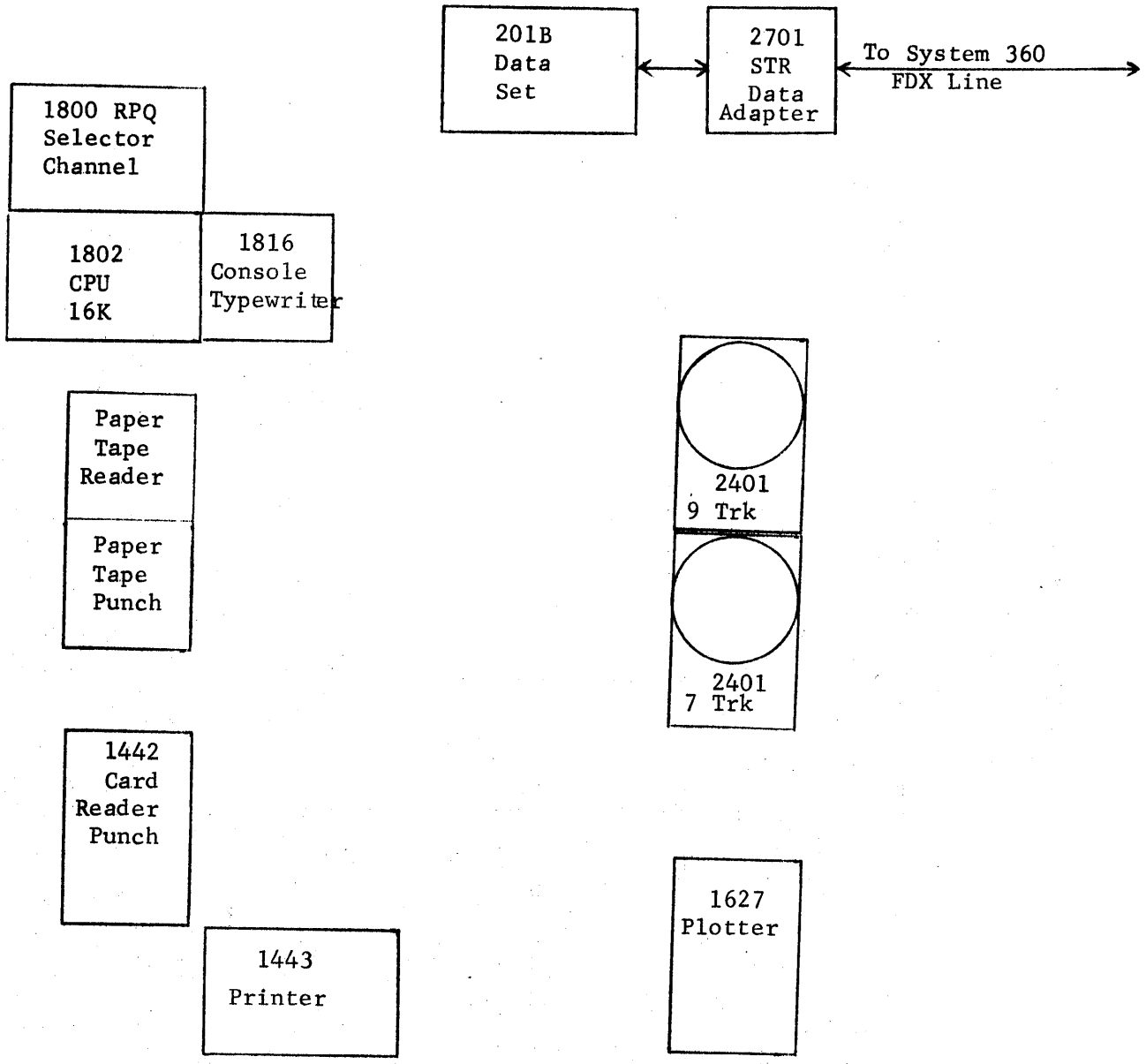
GENERATING PLANTS	NUMBER	CAPACITY
● STEAM	12	5,573,000 KW
HYDRO	66	2,378,000 KW
NUCLEAR	2	57,000 KW
● GEOTHERMAL	1	26,000 KW
TOTAL	81	8,034,000 KW
FROM OTHERS		1,007,000 KW
TRANSMISSION LINES		11,700 MILES
DISTRIBUTION LINES		70,500 MILES
TOTAL LINES		82,200 MILES

FIG. 1

5470f

Figure 2

Schematic Diagram
of
1800 Computer System



PACIFIC GAS AND ELECTRIC COMPANY

System/360 Model 65

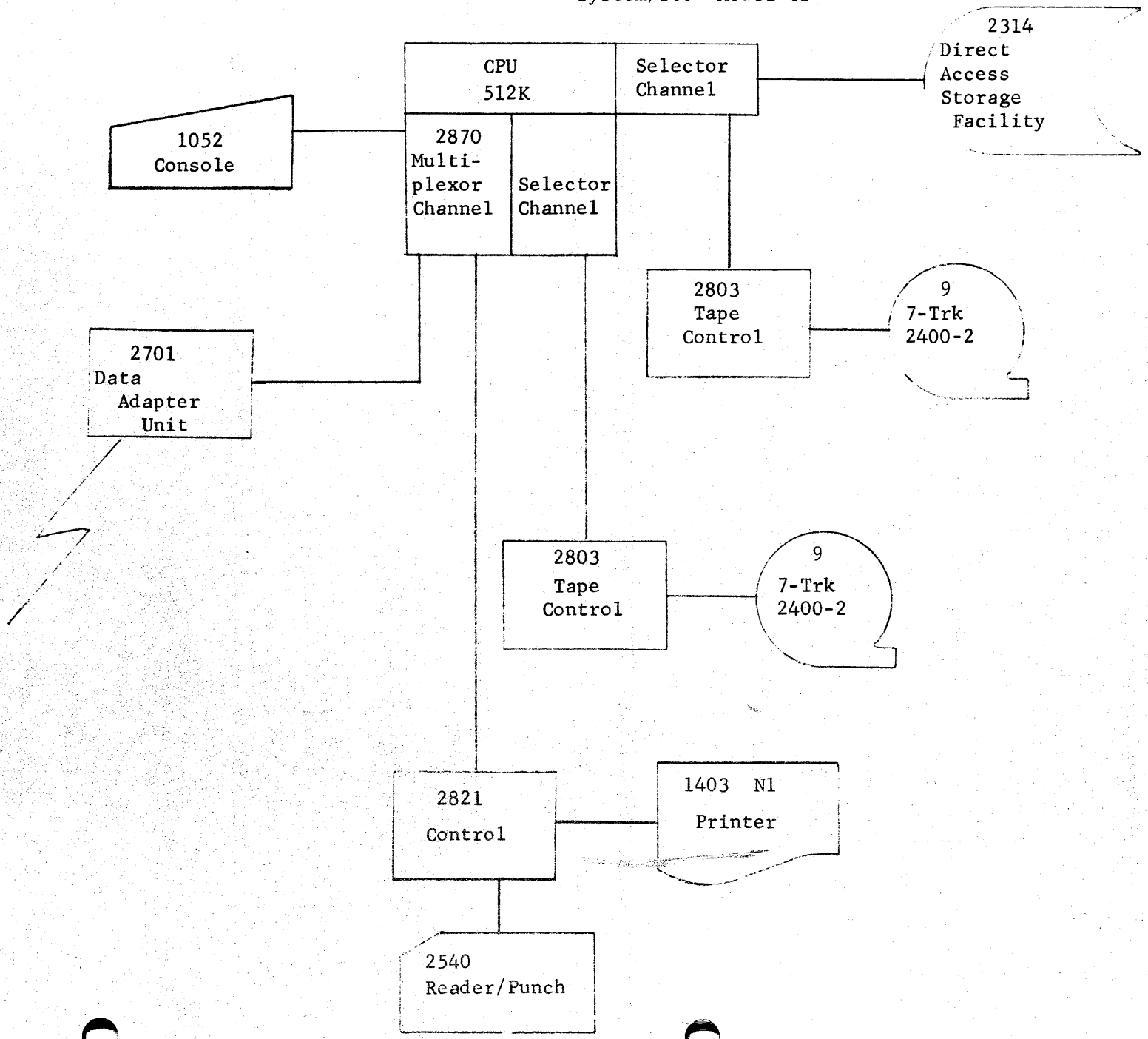
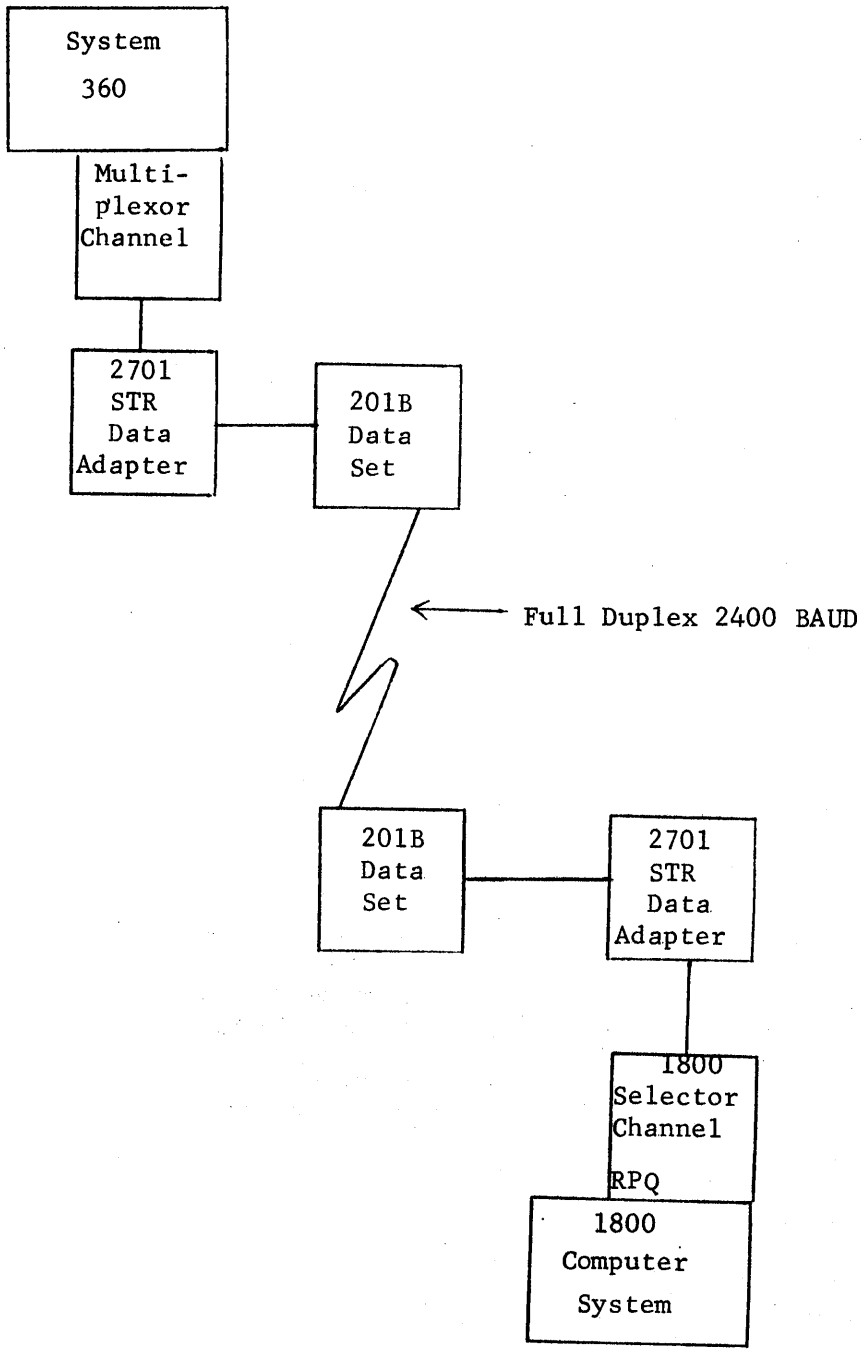
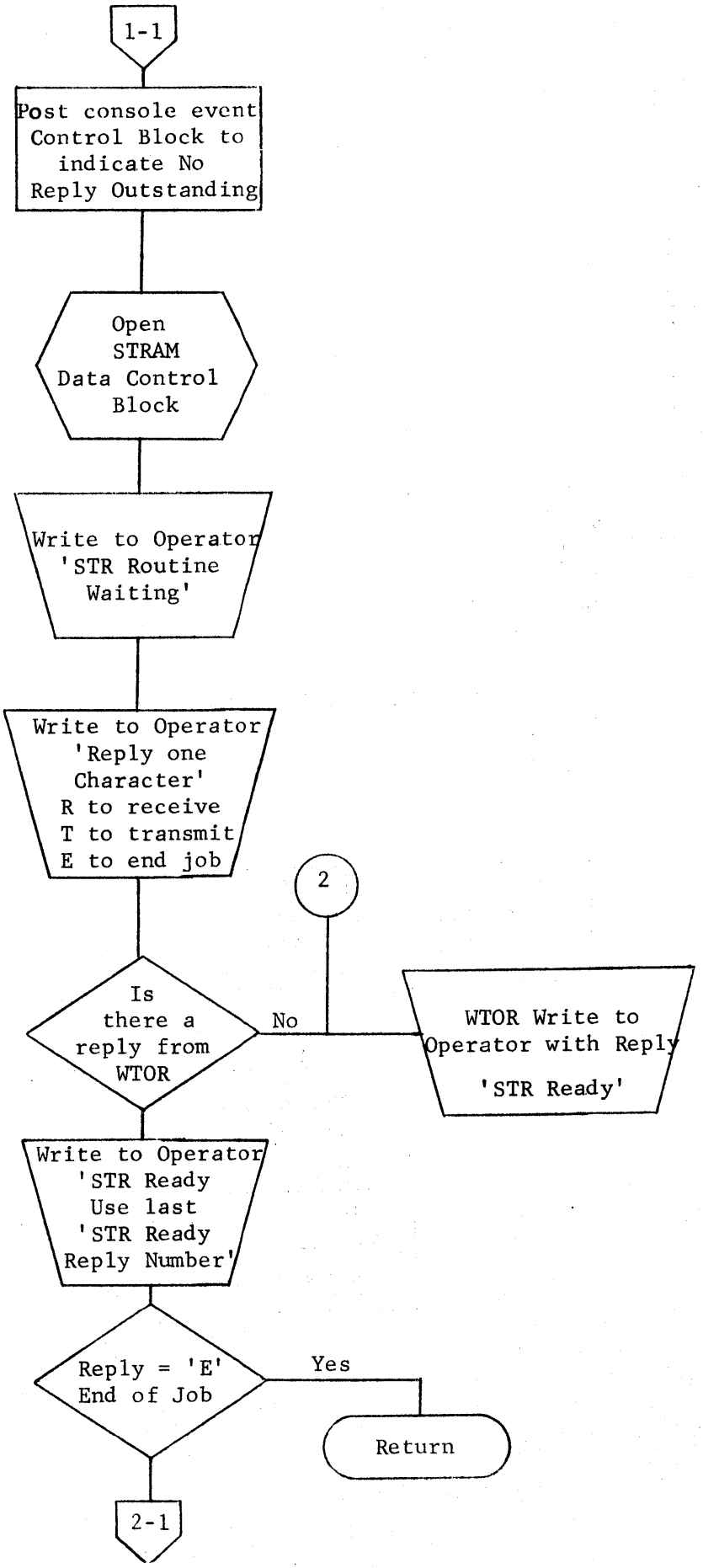


FIGURE 3

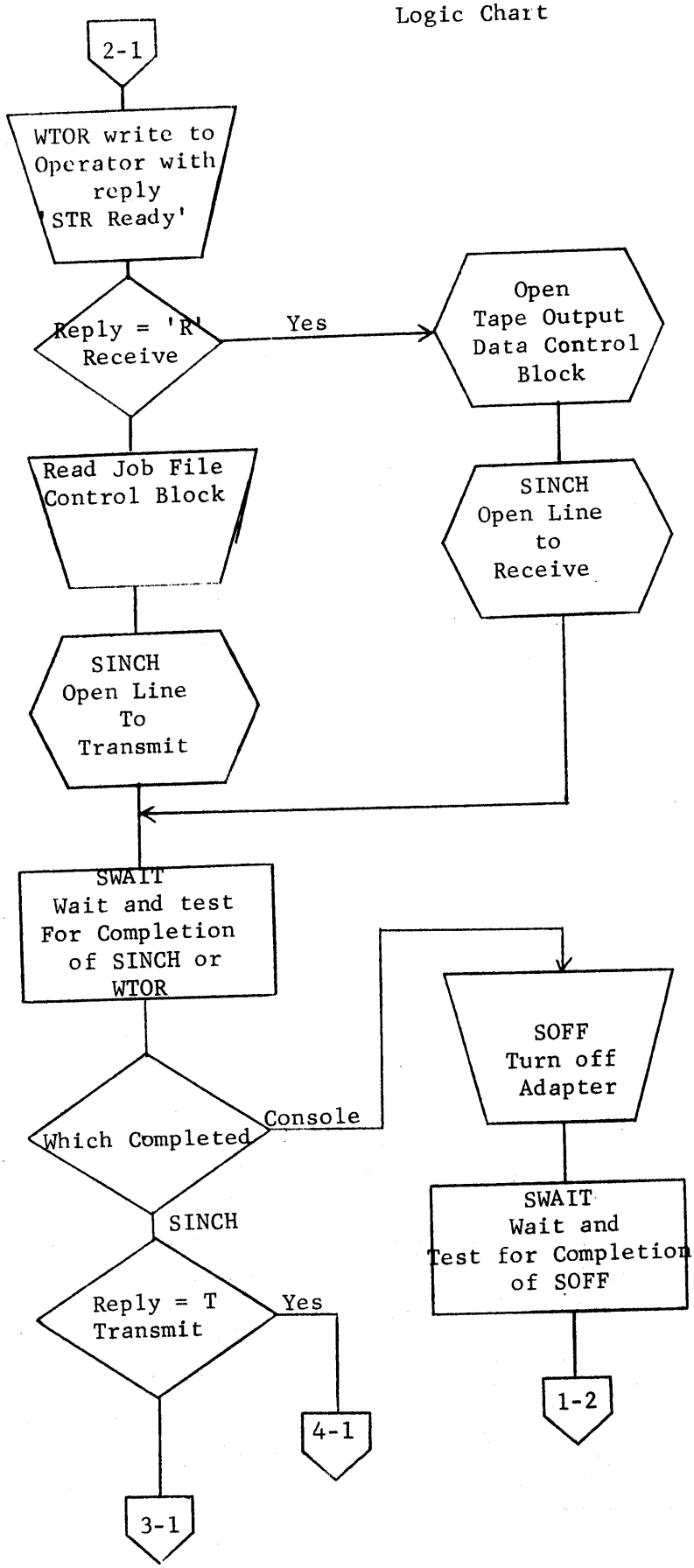
Schematic Diagram
of
1800 - S/360 Telecom System

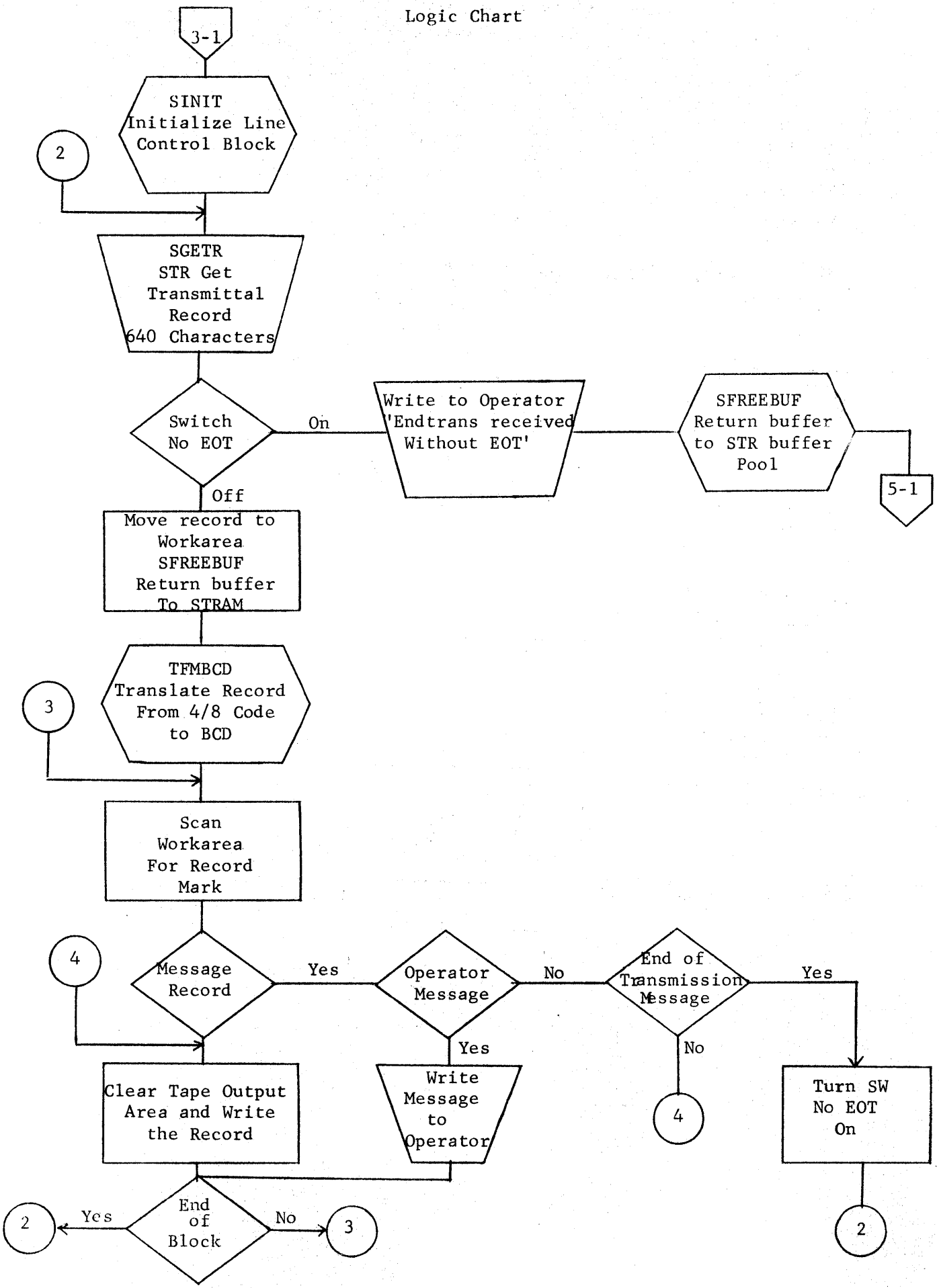
Figure 4

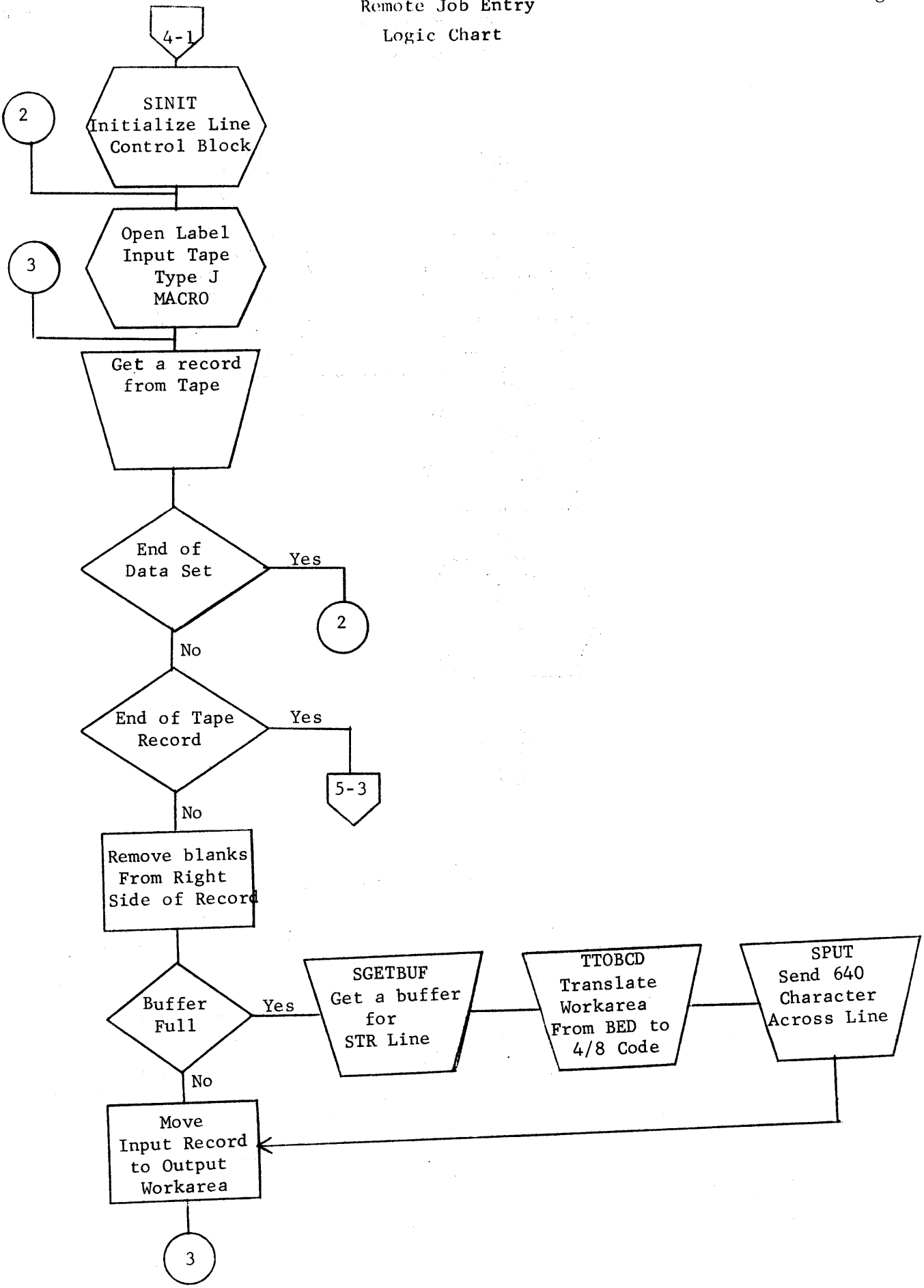


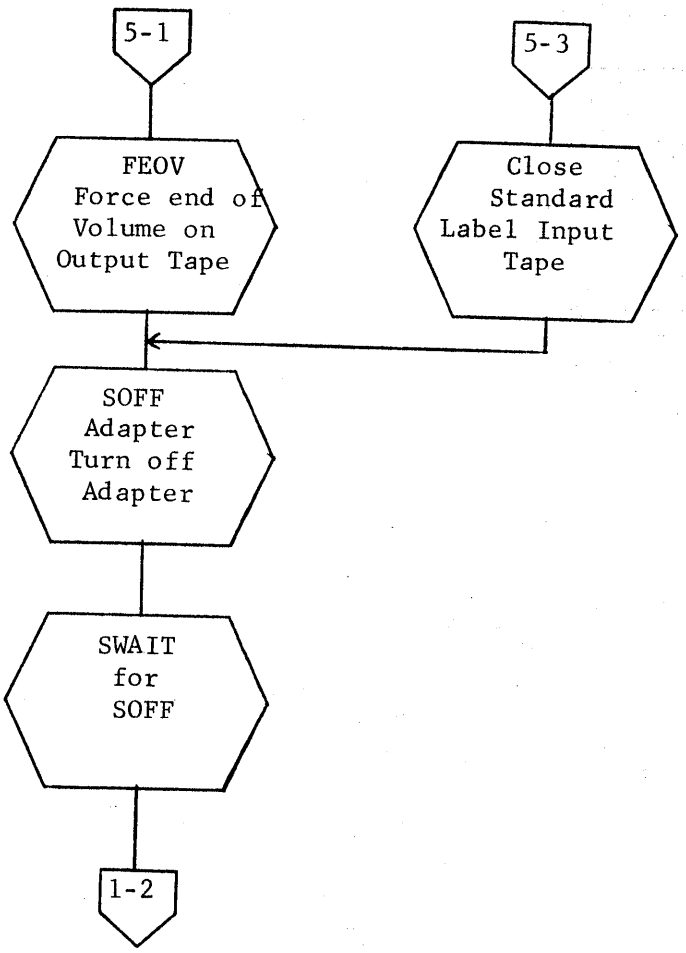

1-1
Post console event Control Block to indicate No Reply Outstanding
Open STRAM Data Control Block
Write to Operator 'STR Routine Waiting'
Write to Operator 'Reply one Character' R to receive T to transmit E to end job
2
Is there a reply from WTOR
No
WTOR Write to Operator with Reply 'STR Ready'
Write to Operator 'STR Ready Use last 'STR Ready Reply Number'
Reply = 'E' End of Job
Yes
Return
2-1

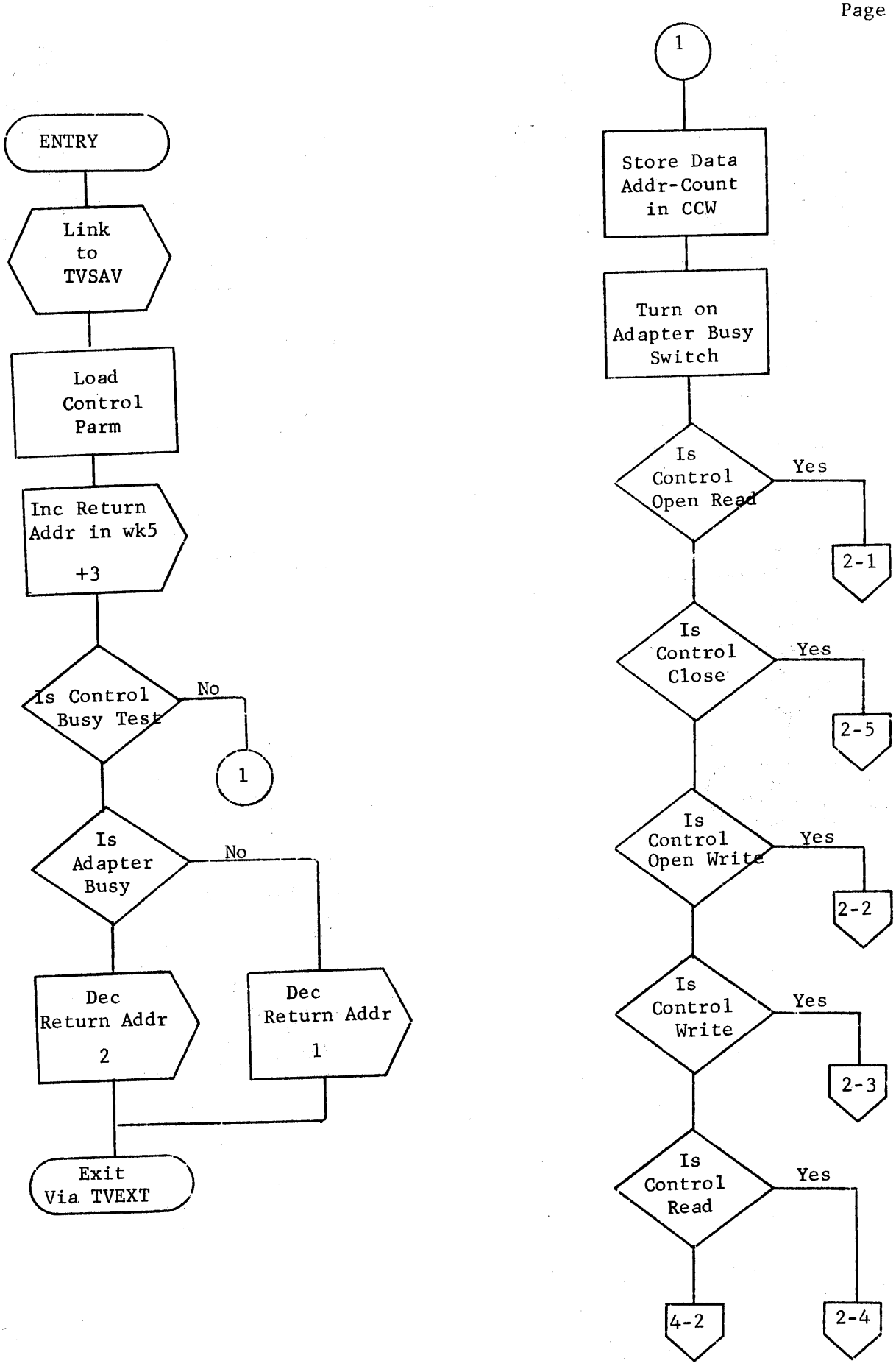
SYSTEM 360/65
Remote Job Entry
Logic Chart

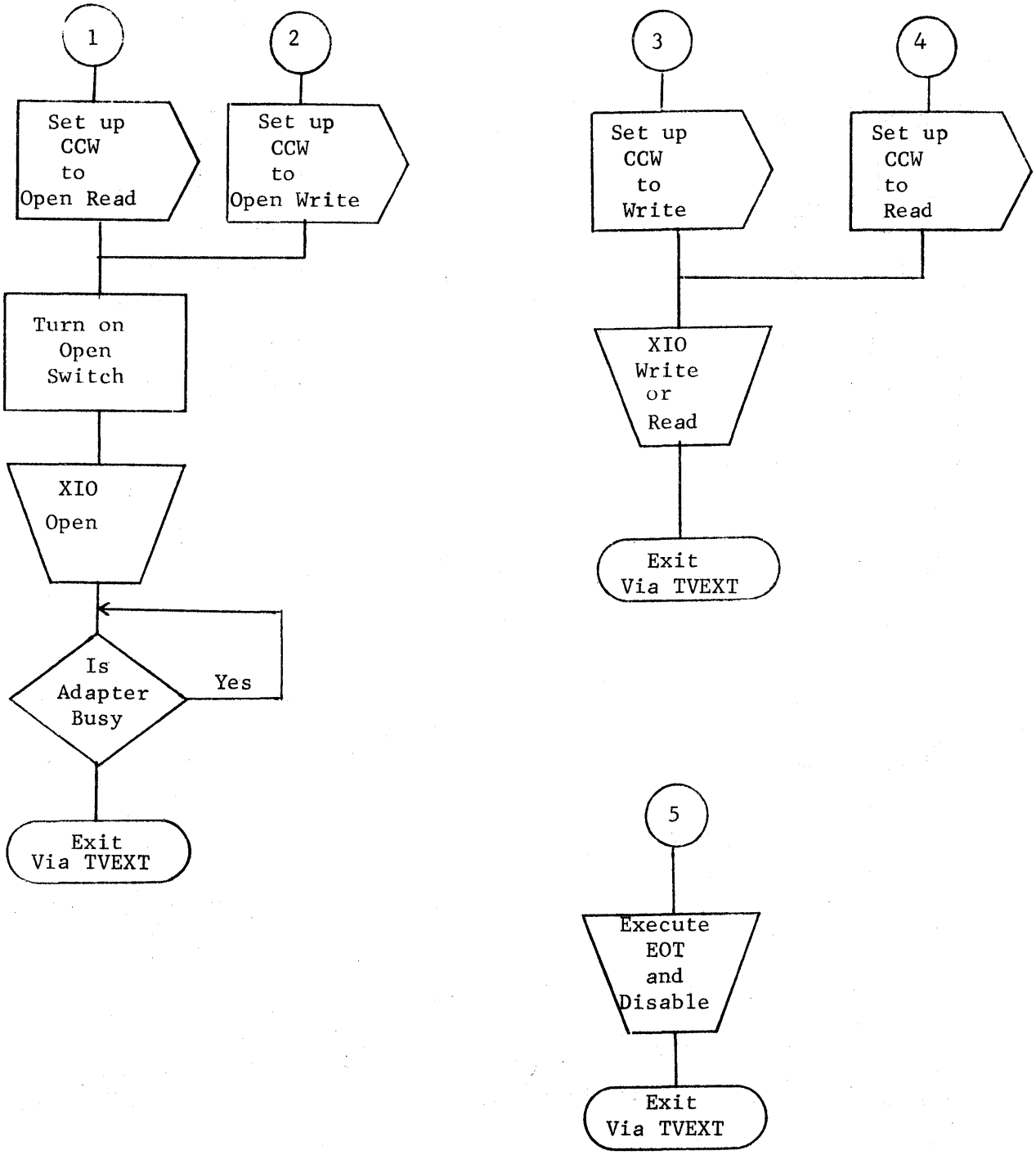



3-1
SINIT
Initialize Line
Control Block
2
SGETR
STR Get
Transmittal
Record
640 Characters
Switch
No EOT
On
Write to Operator
'Endtrans received
Without EOT'
SFREEBUF
Return buffer
to STR buffer
Pool
5-1
Off
Move record to
Workarea
SFREEBUF
Return buffer
To STRAM
TFMBCD
Translate Record
From 4/8 Code
to BCD
3
Scan
Workarea
For Record
Mark
4
Message
Record
Yes
Operator
Message
No
End of
Transmission
Message
Yes
Yes
No
Clear Tape Output
Area and Write
the Record
Write
Message
to
Operator
4
Turn SW
No EOT
On
End
of
Block
Yes
No
2
3
2

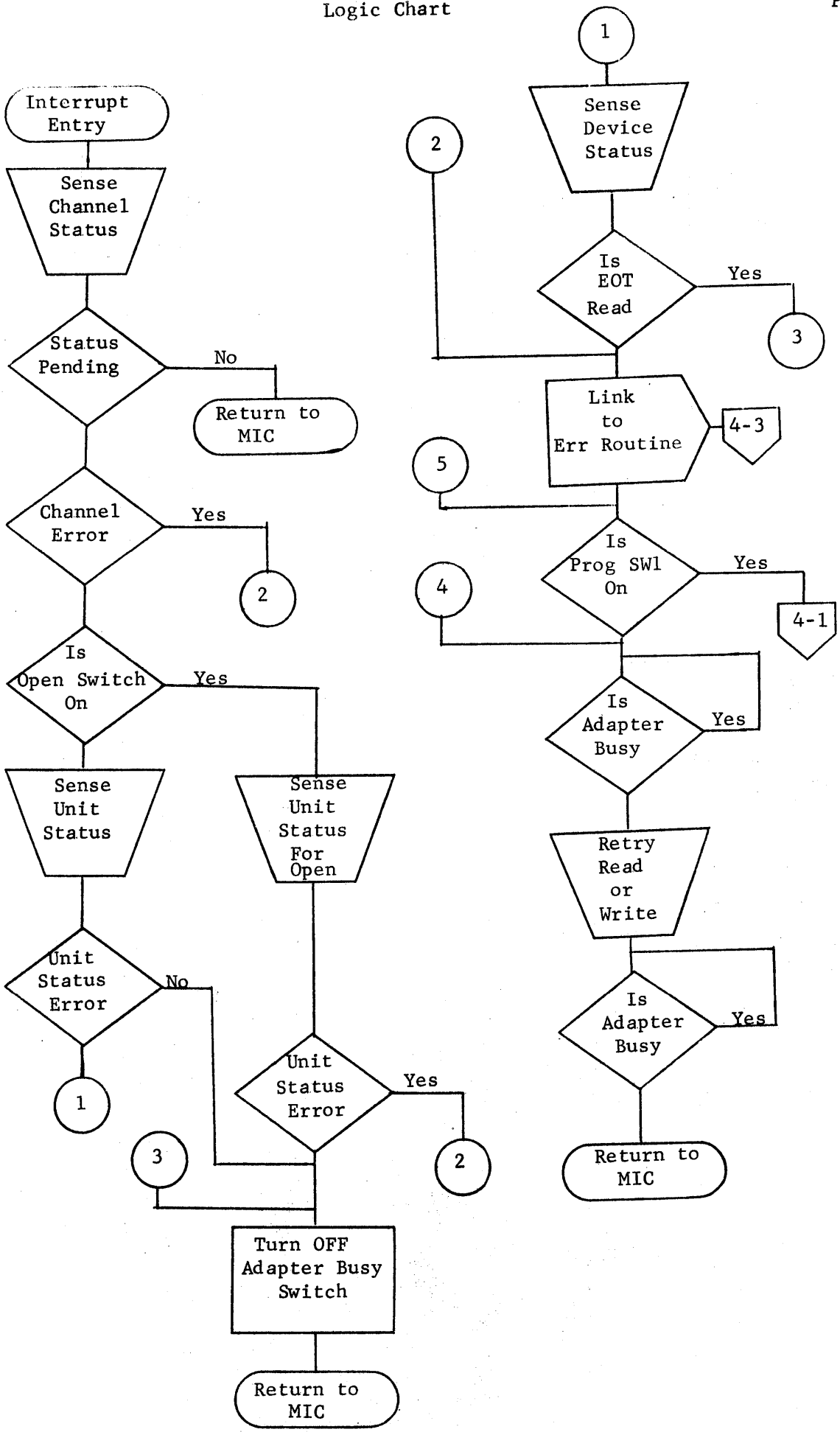

4-1
SINIT
Initialize Line
Control Block
2
Open Label
Input Tape
Type J
MACRO
3
Get a record
from Tape
End of
Data Set
Yes
2
No
End of Tape
Record
Yes
5-3
No
Remove blanks
From Right
Side of Record
Buffer
Full
Yes
SGETBUF
Get a buffer
for
STR Line
TTOBCD
Translate
Workarea
From BED to
4/8 Code
SPUT
Send 640
Character
Across Line
No
Move
Input Record
to Output
Workarea
3


5-1
5-3
FEOV
Force end of
Volume on
Output Tape
Close
Standard
Label Input
Tape
SOFF
Adapter
Turn off
Adapter
SWAIT
for
SOFF
1-2


ENTRY
Link to TVSAV
Load Control Parm
Inc Return Addr in wk5 +3
Is Control Busy Test
No
1
Is Adapter Busy
No
Dec Return Addr 2
Dec Return Addr 1
Exit Via TVEXT
1
Store Data Addr-Count in CCW
Turn on Adapter Busy Switch
Is Control Open Read
Yes
2-1
Is Control Close
Yes
2-5
Is Control Open Write
Yes
2-2
Is Control Write
Yes
2-3
Is Control Read
Yes
4-2
2-4


1
2
Set up CCW to Open Read
Set up CCW to Open Write
Turn on Open Switch
XIO Open
Is Adapter Busy
Yes
Exit Via TVEXT
3
4
Set up CCW to Write
Set up CCW to Read
XIO Write or Read
Exit Via TVEXT
5
Execute EOT and Disable
Exit Via TVEXT

1800 TPSTR SUBROUTINE
Logic Chart



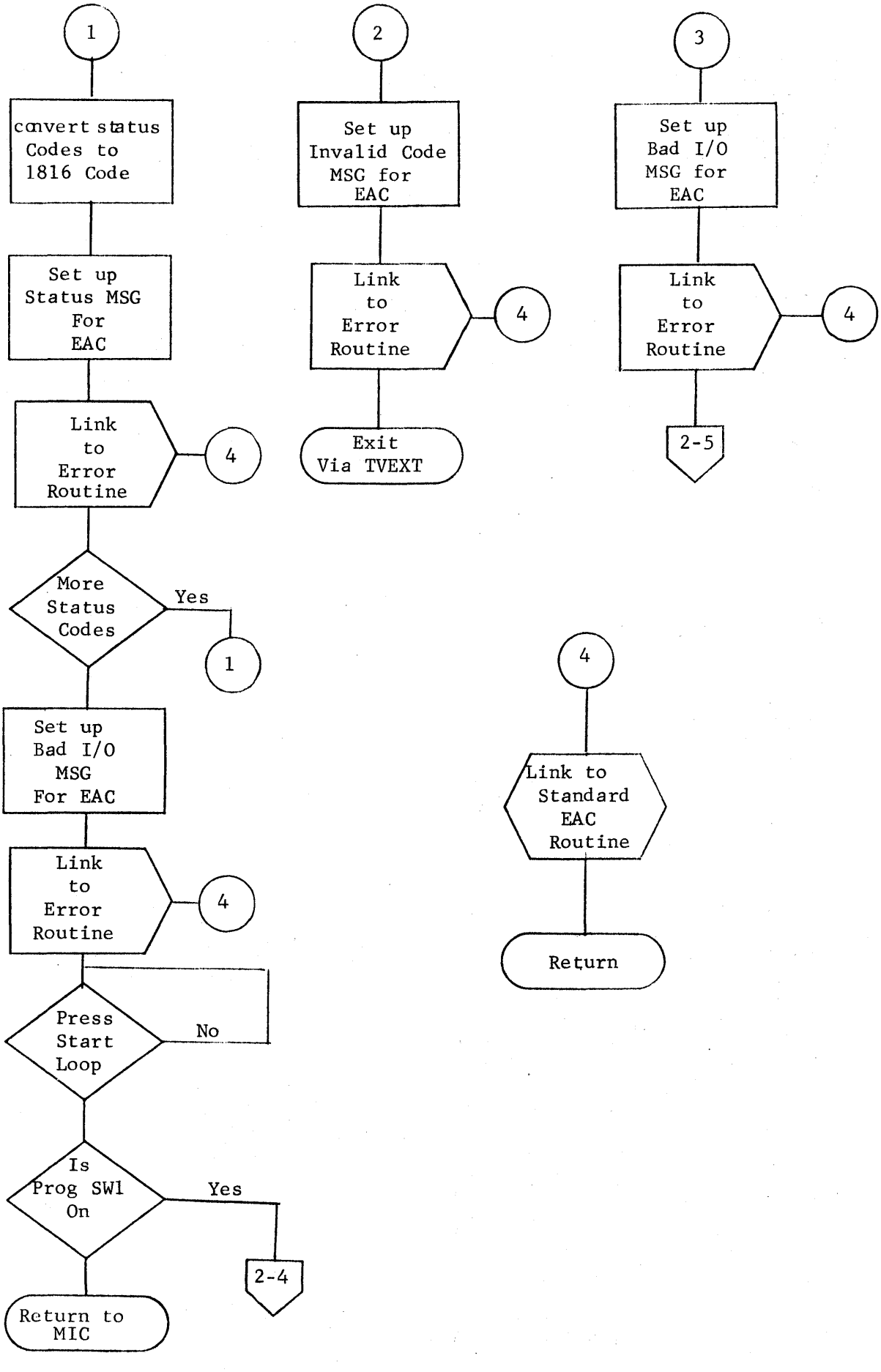
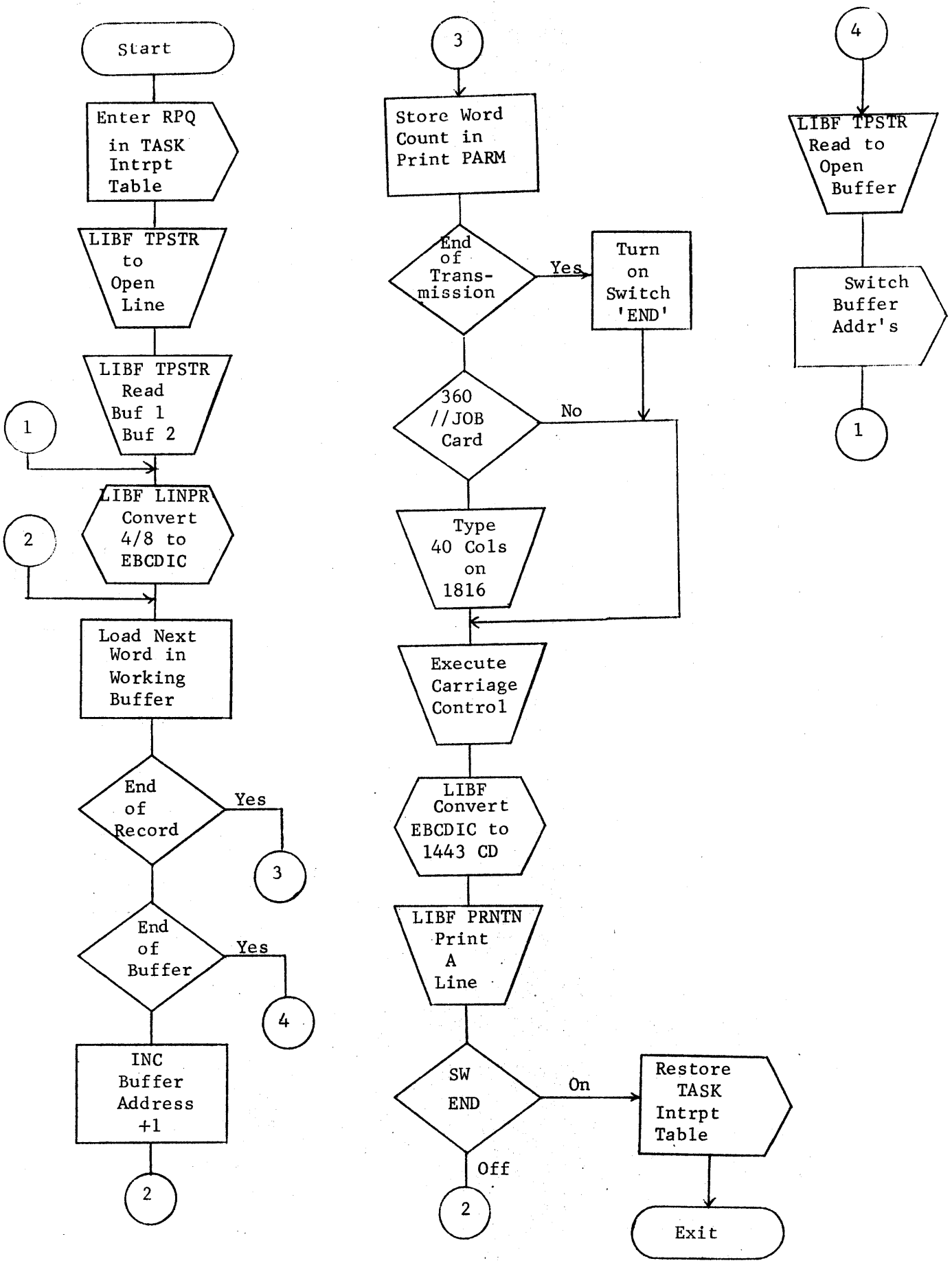

1
convert status Codes to 1816 Code
Set up Status MSG For EAC
Link to Error Routine
4
More Status Codes
Yes
1
Set up Bad I/O MSG For EAC
Link to Error Routine
4
Press Start Loop
No
Is Prog SW1 On
Yes
2-4
Return to MIC
2
Set up Invalid Code MSG for EAC
Link to Error Routine
4
Exit Via TVEXT
3
Set up Bad I/O MSG for EAC
Link to Error Routine
4
2-5
4
Link to Standard EAC Routine
Return

Figure 7

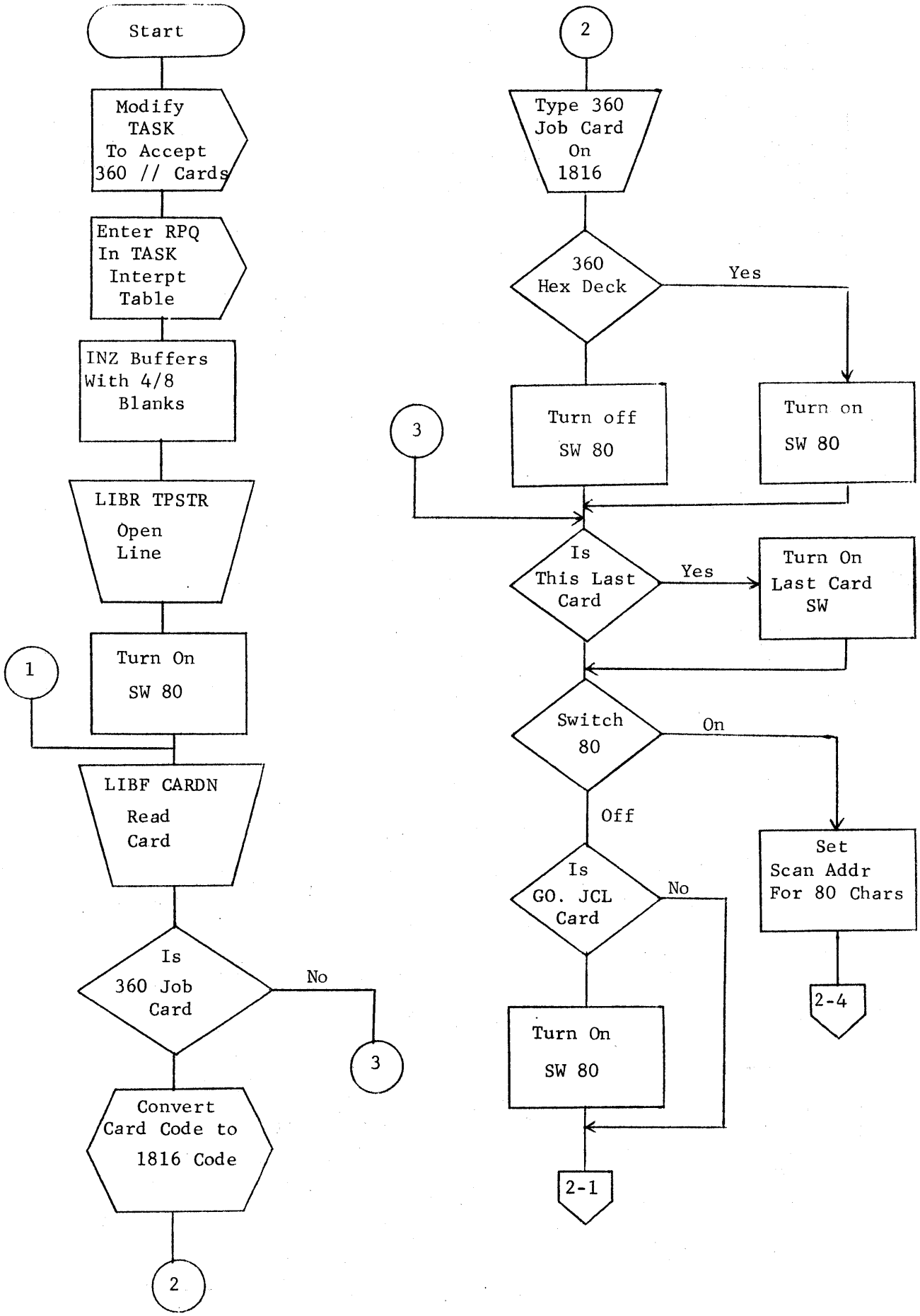
1800 Receive Routine

Logic Chart



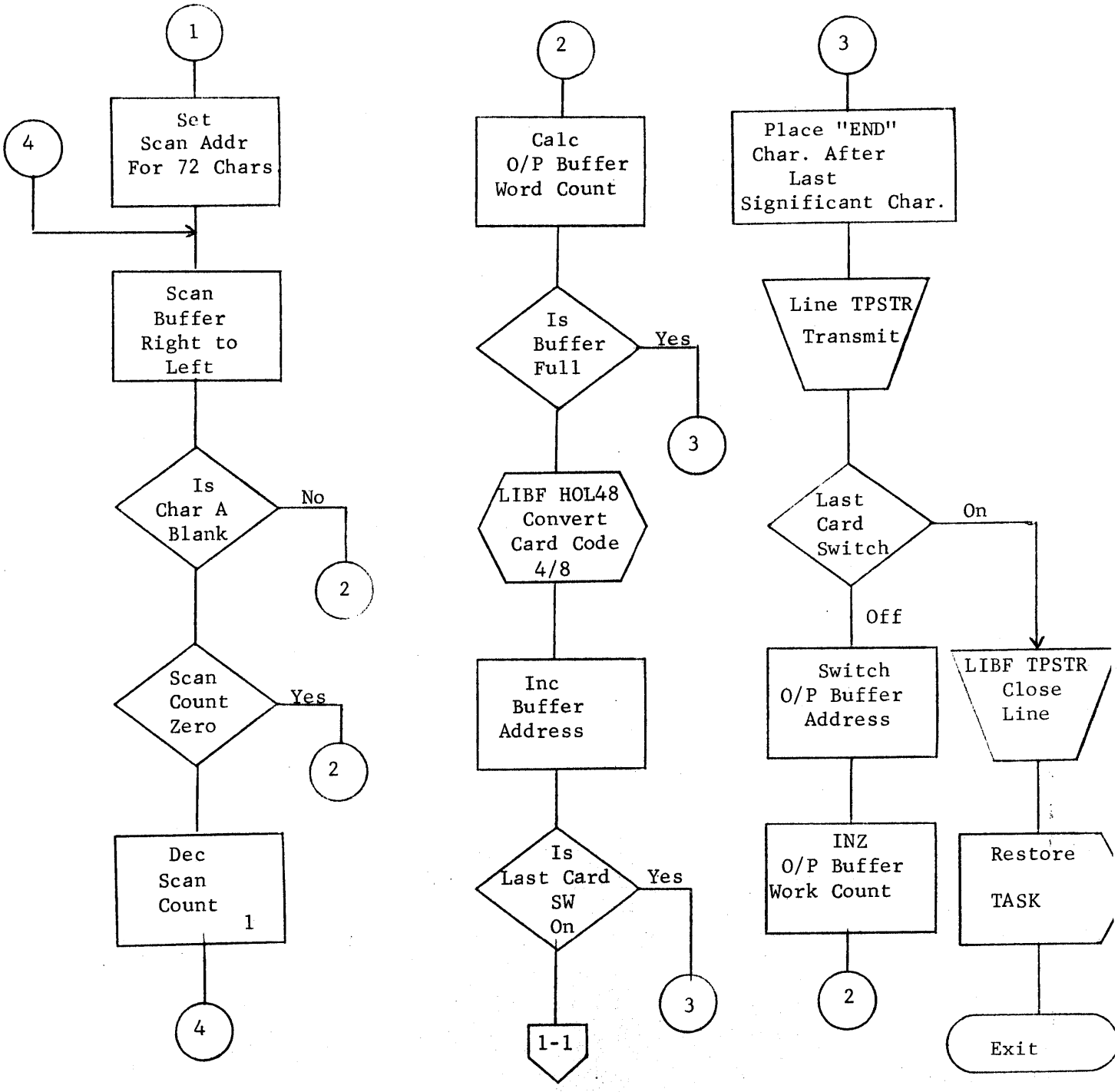
1800 Send Routine

Logic Chart



1800 Send Routine

Logic Chart



```
                        **************************************************
                        *
                        *CALL SEQUENCE AS FOLLOWS FOR READ OR WRITE
                        *
                        *       LIBF    TPSTR
                        *       DC      /AXXX       CONTROL
                        *       DC      AREA    I/O AREA
                        *       DC      NNNNN   DECIMAL CHARACTER COUNT
                        *
                        *       CONTROL CODES
                        *            3- CLOSE
                        *            4- OPEN
                        *            5- WRITE
                        *            6- RECEIVE
                        *
                        ******
                        *
                        *CALL SEQUENCE AS FOLLOWS TO TEST FOR BUSY
                        *       LIBF    TPSTR
                        *       DC      0
                        *       MDX     *-3
                        *
                        **************************************************
                        *
                              LIBR
0000     235E28D9             ISS  1  TPSTR
0000 0   0012                 DC      18          IA CODE
0001 1   0092                 DC      TPIN0       INTERUPT ROUTINE
0002                          ORG     *-2
                        *
00AC                    TVSAV EQU     172
00AD                    TVEXT EQU     173
0037                    WK5   EQU     55
                        *
                        **************************************************
                        *
0000 0   0000           TPSTR DC                  ENTRY POINT TO ROUTINE
0001 00  448000AC             BSI  I  TVSAV
0003 00  65800037             LDX  I1 WK5         XR1 POINTS TO PARAMETERS
0005 00  74030037             MDX  L  WK5,3       MODIFY RETURN ADDRESS
                        *
                        *ANALYSE CONTROL PARAMETER
                        *
0007 0   C100           TPT5  LD   X1 0           LOAD CONTROL TO ACC
0008 0   188C                 SRT     12          SHIFT I/O FUNCTION TO UNIT
0009 01  4C200017             BSC  L  TPT1,Z      IF NOT TEST BRANCH
                        *
                        *TEST FOR ADAPTER BUSY
                        *
000B 01  C40000BA       TPT6  LD   L  WBBSY       SENSE DEVICE BUSY
000D 01  4C200013             BSC  L  TPT2,Z      BRANCH IF BUSY
                        *
000F 00  74FF0037             MDX  L  WK5,-1      RETURN ADD NOT BUSY
0011 00  448000AD             BSI  I  TVEXT
                        *
```

```
0013 00 74FE0037     TPT2   MDX   L   WK5,-2       RETURN ADD BUSY
0015 00 448000AD            BSI   I   TVEXT
                     *
                     ********************************************************
                     *SET DATA ADDRESS AND COUNT INTO CCW FOR WR/RD
                     *IF NOT BUSY PREPARE FOR READ OR WRITE
                     ********************************************************
                     *
0017 0  C101         TPT1   LD    X1  1
0018 01 D4000236            STO   L   DADD         STORE DATA ADDRESS
001A 0  C102                LD    X1  2
001B 01 D4000234            STO   L   BCNT         STORE COUNT
001D 01 740100BA            MDX   L   WBBSY,+1
                     *
                     *
                     *ANALYSE CONTROL PARAMETER
                     *
                     *
001F 0  C100                LD    X1  0            LOAD PARAMETER
0020 0  6204                LDX    2  4            LOAD SHIFT COUNT
0021 0  1240                SLCA   2               SHIFT FOR TEST
0022 01 4E800023            BSC   I2  *-1
                     *
0024 1  00D4                DC        ERR2         CONTROL CODE  1
0025 1  0028                DC        TPT3         CONTROL CODE 2 OR 3
0026 1  002D                DC        TPT7         CONTROL CODE  4, 5, 6, 7,
0027 1  00D4                DC        ERR2         CONTROL CODE  8 OR MORE
                     *
                     *
0028 0  1001         TPT3   SLA       1
0029 01 4C100052            BSC   L   OPENR,-      OPEN READ ON 2
002B 01 4C00008E            BSC   L   CLOSE        CODE 3 - CLOSE
                     *
                     *
002D 0  1001         TPT7   SLA       1            CODES 4,5,6,7
002E 01 4C100035            BSC   L   TPT8,-       BRANCH ON 4 OR 5
0030 0  1001                SLA       1
0031 01 4C2800D4            BSC   L   ERR2,&Z      BRANCH ON CODE 7  INVALID
0033 01 4C000086            BSC   L   TPT4         CODE 6  READ
                     *
                     *
0035 0  1001         TPT8   SLA       1
0036 01 4C10003A            BSC   L   OPENW,-      OPEN WRITE ON 4
0038 01 4C000080            BSC   L   TPT9         BRANCH TO WRITE
                     *
                     ********************************************************* OPEN0
                     *                                                         OPEN0
003A 01 6500022A     OPENW  LDX   L1  IOCWA        SET UP OPEN TO SEND
003C 01 0C00022E            XIO   L   IOCWC        TEST SENSE SWITCH 0
003E 01 4C100049            BSC   L   NORML,-      IF OFF NORMAL MODE
0040 01 CC000262            LDD   L   TMODE 0 ON-LOAD TEST MODE AND
0042 01 DC00025E            STD   L   MODE         TEST WRITE
0044 01 C4000264            LD    L   TREAD        LOAD TEST READ
0046 01 D4000260            STO   L   CN02
0048 0  7013                MDX       TAG
```

```
0049 01 CC000266    NORML. LDD  L  NMODE       LOAD NORMAL MODE AND
004B 01 DC00025E           STD  L  MODE        NORMAL WRITE
004D 01 C4000268           LD   L  NREAD       LOAD NORMAL READ
004F 01 D4000260           STO  L  CN02
0051 0  700A               MDX     TAG
0052 01 CC000266    OPENR  LDD  L  NMODE       LOAD NORMAL MODE AND
0054 01 DC00025E           STD  L  MODE        NORMAL WRITE
0056 01 C4000268           LD   L  NREAD       LOAD NORMAL READ
0058 01 D4000260           STO  L  CN02
005A 01 65000232           LDX  L1 IOCWE       SET UP OPEN TO RECEIVE
005C 0  691C        TAG    STX   1 MODIO+1
005D 01 740100BB           MDX  L  SOPEN,+1
005F 00 C400005A           LD   L  90
0061 01 9400025D           S    L  CON8
0063 0  D003               STO     *+3
0064 0  D006               STO     *+6
0065 0  D007               STO     *+7
0066 00 2C400000           STS  L  /0000,/40   CLEAR STORAGE PROTECT BITS
0068 01 65000092           LDX  L1 TPINO
006A 00 6D000000           STX  L1 /0000
006C 00 2C410000           STS  L  /0000,/41   WRITE STORAGE PROTECT BITS
006E 01 740100BA           MDX  L  WBBSY,+1
0070 01 0C000224           XIO  L  IOCW5       DISABLE AND SET MODE
0072 01 C40000BA           LD   L  WBBSY
0074 01 4C200072           BSC  L  *-4,Z
0076 01 740100BA           MDX  L  WBBSY,+1
0078 00 0C000000    MODIO  XIO  L  *-*
007A 01 C40000BA           LD   L  WBBSY
007C 01 4C20007A           BSC  L  *-4,Z
007E 00 448000AD           BSI  I  TVEXT
                    *                                                  OPENO
                    *
                    ****************************************************
                    *
                    *SET WRITE OR READ COMMAND IN CCW
                    *
0080 01 C400025F    TPT9   LD   L  CN01        SET UP CCW FOR WRITE
0082 01 D4000235           STO  L  WR$RD
0084 01 4C00008A           BSC  L  TPW2
                    *
0086 01 C4000260    TPT4   LD   L  CN02        SET UP CCW FOR READ
0088 01 D4000235           STO  L  WR$RD
008A 01 0C00021E    TPW2   XIO  L  IOCW2       EXECUTE WRITE OR READ
008C 00 448000AD           BSI  I  TVEXT
                    *
                    *
                    ****************************************************
008E 01 0C000230    CLOSE  XIO  L  IOCWD
0090 00 448000AD           BSI  I  TVEXT
                    *
                    *                                                  BHTYP
                    **************************************************** BHTYP
                    **************************************************** TPINO
                    *TPINO IS INTERRUPT ROUTINE FOR TPSTR               TPINO
                    **************************************************** TPINO
```

```
                        *                                                    TPIN0
0092 01 0C00022C  TPIN0 XIO  L  IOCWB       CHANNEL STATUS W/O RESET
0094 01 D4000257        STO  L  WORK3
0096 0  1001            SLA     1           PENDING BIT
0097 00 4C90005A        BSC  I  90,-        BRANCH IF FALSE INTERRUPT
0099 0  1001            SLA     1
009A 01 4C20010F        BSC  L  TPIN6,Z     BRANCH IF CHANNEL ERROR
009C 01 0C000222        XIO  L  IOCW4       SENSE UNIT STATUS
009E 01 D4000258        STO  L  WORK4
00A0 01 C40000BB        LD   L  SOPEN
00A2 01 4C1800AE        BSC  L  TSTUS,+-
00A4 01 C4000258        LD   L  WORK4       UNIT STATUS
00A6 01 F40000BC        EOR  L  COND
00A8 01 4C20010F        BSC  L  TPIN6,Z
00AA 0  1010            SLA     16
00AB 01 D40000BB        STO  L  SOPEN
00AD 0  7006            MDX     TPEXT
00AE 01 C4000258  TSTUS LD   L  WORK4
00B0 01 F40000B9        EOR  L  CONUS
00B2 01 4C2000BD        BSC  L  IFEOT,Z     BRANCH IF UNIT STATUS ERR
00B4 0  1010      TPEXT SLA     16          NORMAL STATUS
00B5 01 D40000BA        STO  L  WBBSY       CLEAR BUSY SW
00B7 00 4C80005A        BSC  I  90          RETURN TO MIC
00B9 0  020C      CONUS DC      /020C       NORMAL UNIT STATUS
00BA 0  0000      WBBSY DC      0           BUSY SWITCH
00BB 0  0000      SOPEN DC      0           OPEN SWITCH
00BC 0  020D      COND  DC      /020D
00BD 01 0C000228  IFEOT XIO  L  IOCW7       SENSE DEVICE STATUS
00BF 01 C4000259        LD   L  WORK7       TEST FOR EOT ON RECEIVE
00C1 0  100B            SLA     11
00C2 01 4C10010F        BSC  L  TPIN6,-     BRANCH IF NOT EOT
00C4 0  70EF            MDX     TPEXT
                        *
                        ************************************************
                        *STANDARD CALL TO TSX EAC PRINTER
                        ************************************************
                        *
00C5 0  0000      ERRPT DC      *-*
00C6 00 0C000032        XIO  L  50
00C8 00 0C000034        XIO  L  52
00CA 00 65000000  ERPT1 LDX  L1 *-*
00CC 00 44800099        BSI  I  153
00CE 00 0C00002E        XIO  L  46
00D0 00 0C000030        XIO  L  48
00D2 01 4C8000C5        BSC  I  ERRPT       RETURN TO CALLING LOC+1
                        *
                        *
00D4 00 C4000037  ERR2  LD   L  WK5
00D6 01 9400025C        S    L  CON03
00D8 01 440000E4        BSI  L  BHTYP
00DA 01 DC0001B0        STD  L  EMS2B
00DC 01 C4000195        LD   L  EMES2
00DE 01 D40000CB        STO  L  ERPT1&1
00E0 01 440000C5        BSI  L  ERRPT
00E2 00 448000AD        BSI  I  TVEXT
```

```
                        ********************************************************
                        *CONVERT BIN IN ACC TO TYPEWRITER HEX IN A&Q            BHTYP
                        ********************************************************BHTYP
                        *                                                      BHTYP
                        *                                                      BHTYP
                        *                                                      BHTYP
00E4 0  0000    BHTYP   DC      *-*                                            BHTYP
00E5 0  1890            SRT     16                                             BHTYP
00E6 0  6204            LDX   2 4                                              BHTYP
00E7 0  18C4    BHTY    RTE     4                                              BHTYP
00E8 0  180C            SRA     12                                             BHTYP
00E9 0  D001            STO     *&1                                            BHTYP
00EA 00 65000000        LDX   L1 *-*                                           BHTYP
00EC 01 C50000FF        LD    L1 BHTAB                                         BHTYP
00EE 01 D60000FA        STO   L2 HBTA1-1                                       BHTYP
00F0 0  72FF            MDX   2 -1                                             BHTYP
00F1 0  70F5            MDX     BHTY                                           BHTYP
00F2 0  C00A            LD      HBTA1&2                                        BHTYP
00F3 0  1008            SLA     8                                              BHTYP
00F4 0  E809            OR      HBTA1&3                                        BHTYP
00F5 0  1890            SRT     16                                             BHTYP
00F6 0  C004            LD      HBTA1                                          BHTYP
00F7 0  1008            SLA     8                                              BHTYP
00F8 0  E803            OR      HBTA1&1                                        BHTYP
00F9 01 4C8000E4        BSC   I BHTYP                                          BHTYP
00FB 0  0000    HBTA1   DC      *-*                                            BHTYP
00FC 0  0000            DC      *-*                                            BHTYP
00FD 0  0000            DC      *-*                                            BHTYP
00FE 0  0000            DC      *-*                                            BHTYP
                        *                                                      BHTYP
00FF 0  00C4    BHTAB   DC      /00C4       0                                  BHTYP
0100 0  00FC            DC      /00FC       1                                  BHTYP
0101 0  00D8            DC      /00D8       2                                  BHTYP
0102 0  00DC            DC      /00DC       3                                  BHTYP
0103 0  00F0            DC      /00F0       4                                  BHTYP
0104 0  00F4            DC      /00F4       5                                  BHTYP
0105 0  00D0            DC      /00D0       6                                  BHTYP
0106 0  00D4            DC      /00D4       7                                  BHTYP
0107 0  00E4            DC      /00E4       8                                  BHTYP
0108 0  00E0            DC      /00E0       9                                  BHTYP
0109 0  003C            DC      /003C       A                                  BHTYP
010A 0  0018            DC      /0018       B                                  BHTYP
010B 0  001C            DC      /001C       C                                  BHTYP
010C 0  0030            DC      /0030       D                                  BHTYP
010D 0  0034            DC      /0034       E                                  BHTYP
010E 0  0010            DC      /0010       F                                  BHTYP
                        *                                                      BHTYP
010F 01 C4000173TPIN6   LD    L EMES7
0111 01 D40000CB        STO   L ERPT1&1
0113 01 440000C5        BSI   L ERRPT
0115 01 0C00022E        XIO   L IOCWC       TEST PROG SWITCHES
0117 0  1004            SLA     4
0118 01 4C280128        BSC   L TPIN8,&Z
011A 01 0C00022C RETRY  XIO   L IOCWB       SENSE CHANNEL STATUS
011C 0  1007            SLA     7
```

```
011D 01 4C28011A          BSC  L  RETRY,+Z    BRANCH IF BUSY
011F 01 0C00021E          XIO  L  IOCW2       SW ONE OFF RE-TRY I/O
0121 01 0C00022C          XIO  L  IOCWB
0123 0  1007              SLA     7
0124 01 4C280121          BSC  L  *-5,+Z
0126 00 4C80005A          BSC  I  90
                     *                                              TPINO
0128 01 0C000220    TPIN8 XIO  L  IOCW3
012A 01 9400025B          S    L  CON02
012C 01 D4000255          STO  L  WORK1       COMMAND ADDRESS
012E 01 C4800255          LD   I  WORK1
0130 01 D4000256          STO  L  WORK2       COMMAND
0132 01 C4000257          LD   L  WORK3       CHANNEL STATUS
0134 01 440000E4          BSI  L  BHTYP                              TPINO
0136 01 DC0001D0          STD  L  TST2B                              TPINO
0138 01 C4000258          LD   L  WORK4       UNIT STATUS            TPINO
013A 01 440000E4          BSI  L  BHTYP                              TPINO
013C 01 DC0001E2          STD  L  TST2C                              TPINO
013E 01 C4000259          LD   L  WORK7       DEVICE SENSE           TPINO
0140 01 440000E4          BSI  L  BHTYP                              TPINO
0142 01 DC0001F0          STD  L  TST2D                              TPINO
0144 01 0C000226          XIO  L  IOCW6       BYTE COUNT             TPINO
0146 01 440000E4          BSI  L  BHTYP                              TPINO
0148 01 DC0001FC          STD  L  TST2E                              TPINO
014A 01 C4000255          LD   L  WORK1                              TPINO
014C 01 9400025A          S    L  CON01       CALC CCW ADDRESS       TPINO
014E 01 440000E4          BSI  L  BHTYP                              TPINO
0150 01 DC000208          STD  L  TST2F                              TPINO
0152 01 C4000256          LD   L  WORK2       CURRENT COMMAND        TPINO
0154 01 440000E4          BSI  L  BHTYP                              TPINO
0156 01 DC000212          STD  L  TST2G                              TPINO
0158 01 C400025E          LD   L  MODE        EXPECTED MODE          TPINO
015A 01 440000E4          BSI  L  BHTYP                              TPINO
015C 01 DC00021C          STD  L  TST2H                              TPINO
                     *                                              TPINO
015E 01 C40001C5          LD   L  EMES5                              TPINO
0160 01 D40000CB          STO  L  ERPT1&1                            TPINO
0162 01 440000C5          BSI  L  ERRPT                              TPINO
0164 01 C4000184          LD   L  EMES8
0166 01 D40000CB          STO  L  ERPT1&1
0168 01 440000C5          BSI  L  ERRPT
016A 00 4480003E    WAITC BSI  I  62
016C 01 0C00022E          XIO  L  IOCWC
016E 0  1004              SLA     4
016F 01 4C10011A          BSC  L  RETRY,-     BR TO RETRY IF SW 1 OFF
0171 00 4C80005A          BSC  I  90          DROP RCDS -CONTINUE
                     *
0173 1  0174        EMES7 DC      *
0174 0  000F              DC      TST3C-TST3A
0175    000A        TST3A DMES    'RBAD I/O  'E
017A    0014              DMES    'RPROG SW 1 DETAILS'R'E
0184    0000        TST3C BES  0
0184 1  0185        EMES8 DC      *
0185 0  000F              DC      TST4B-TST4A
0186    000A        TST4A DMES    'RPROG SW 1'E
```

```
018B     000A              DMES    'ROFF RETRY'E
0190     000A              DMES    'RON IGNORE'E
0195     0000        TST4B BES     0
                    ********************************************************* TPIN0
                    *                                                         ERR40
                    ********************************************************* ERR40
                    *
0195 1   0196        EMES2 DC      *
0196 0   002E              DC      EMS2C-EMS2A
0197     001E        EMS2A DMES  0 'RINVALID CONTROL CODE IN TPSTR'
01A6     0012              DMES    LIBF LOCATED AT '2S'E
01B0     0002        EMS2B BSS   E 2
01B2     0026              DMES  'R'E
01C5     0000        EMS2C BES     0
                    *
                    *                                                         TEST1
01C5 1   01C6        EMES5 DC      *                                          TEST2
01C6 0   0057              DC      TST2I-TST2A                               TEST2
01C7     0012        TST2A DMES    'RCHANNEL STATUS  'E                      TEST2
01D0     0002        TST2B BSS   E 2                                         TEST2
01D2     001E              DMES    '5SUNIT ADDRESS AND STATUS  'E            TEST2
01E2     0002        TST2C BSS   E 2                                         TEST2
01E4     0018              DMES    'RDEVICE SENSE REGISTER  'E               TEST2
01F0     0002        TST2D BSS   E 2                                         TEST2
01F2     0012              DMES    '5SBYTE COUNT  'E                         TEST2
01FC     0002        TST2E BSS   E 2                                         TEST2
01FE     0012              DMES    'RCOMMAND ADDRESS  'E                     TEST2
0208     0002        TST2F BSS   E 2                                         TEST2
020A     0010              DMES    'RCOMMAND CODE  'E                        TEST2
0212     0002        TST2G BSS   E 2                                         TEST2
0214     0010              DMES    'REXPECTED MODE  'E                       TEST2
021C     0002        TST2H BSS   E 2                                         TEST2
021E     0000        TST2I BES     0                                         TEST2
                    *
                    *                                                         OPEN0
                    *********************************************************
                    *
                    *********************************************************
                    *IOCC AND CCW AREA
                    *********************************************************
                    *
021E     0000              BSS   E 0
                    *
021E 1   0234        IOCW2 DC      BCNT        WRITE OR READ IOCC
021F 0   9502              DC      /9502
                    *
0220 0   0000        IOCW3 DC      /0000       IOCC FOR SENSECOM ADD
0221 0   9705              DC      /9705
                    *
0222 0   0000        IOCW4 DC      /0000       IOCC FOR SENSE UNIT STATUS
0223 0   9703              DC      /9703
                    *
0224 1   0237        IOCW5 DC      CCWW1       IOCC TO START CHAINED CCW
0225 0   9502              DC      /9502       TO OPEN-WRITE
                    *
```

```
0226 0  0000        IOCW6  DC     /0000     IOCC FOR SENSE COUNT
0227 0  9707               DC     /9707
                    *
0228 1  0252        IOCW7  DC     CCWW7     IOCC FOR SENSE DVICE STAT
0229 0  9502               DC     /9502
                    *
022A 1  023D        IOCWA  DC     CCWW2     IOCC TO START CHAINED
022B 0  9502               DC     /9502       ENABLE,TEST SYNC8 SEND
                    *
022C 0  0000        IOCWB  DC     /0000     SENSE CH STAT W/O RESET
022D 0  9700               DC     /9700
                    *
022E 0  0000        IOCWC  DC     /0000     IOCC FOR SENSE PROGRAM SW
022F 0  0760               DC     /0760
                    *
0230 1  024C        IOCWD  DC     CCWW5     IOCC FOR CHAINED EOT
0231 0  9502               DC     /9502
0232 1  0246        IOCWE  DC     CCWWE
0233 0  9502               DC     /9502
                    *
0234 0  0000        BCNT   DC     /0000     BYTE COUNT FROM TPT1
0235 0  0000        WR$RD  DC     *-*       COMMAND CODE FROM TPT4-9
0236 0  0000        DADD   DC     /0000     DATA ADDRESS FROM TPT1
                    *
0237 0  0000        CCWW1  DC     /0000
0238 0  602F               DC     /602F     DISABLE
0239 0  0000               DC     /000
                    *
023A 0  0002               DC     /0002
023B 0  2023               DC     /2023     SET MODE
023C 1  025E               DC     MODE
                    *
023D 0  0000        CCWW2  DC     /0000
023E 0  6027               DC     /6027     ENABLE
023F 0  0000               DC     /0000
                    *
0240 0  0050               DC     /0050
0241 0  6033               DC     /6033     TEST SYNC
0242 0  0000               DC     /0000
                    *
0243 0  0003               DC     /0003
0244 0  203B               DC     /203B     SEND INQ
0245 0  0000               DC     /0000
                    *
0246 0  0000        CCWWE  DC     /0000
0247 0  6027               DC     /6027     ENABLE
0248 0  0000               DC     /0000
                    *
0249 0  0003               DC     /0003
024A 0  2006               DC     /2006     PREPARE TO RECEIVE DATA
024B 0  0000               DC     /0000
                    *
024C 0  0005        CCWW5  DC     /0005
024D 0  6037               DC     /6037     SEND EOT CHAINED TO DISABL
024E 0  0000               DC     /0000
```

```
                        *
024F 0  0000            DC      /0000
0250 0  202F            DC      /202F       DISABLE NO CHALN
0251 0  0000            DC      /0000
                        *
0252 0  0002    CCWW7   DC      /0002
0253 0  2004            DC      /2004       SENSE DEVICE STATUS CCW
0254 1  0259            DC      WORK7
                        *
0255 0  0000    WORK1   DC      *-*
0256 0  0000    WORK2   DC      *-*         CHANNEL STATUS WORD
0257 0  0000    WORK3   DC      *-*
0258 0  0000    WORK4   DC      *-*         UNIT ADD-UNIT STATUS WORD
0259 0  0000    WORK7   DC      *-*         DEVICE STATUS WORD
025A 0  0001    CON01   DC      /0001
025B 0  0002    CON02   DC      /0002
025C 0  0003    CON03   DC      /003
025D 0  0008    CON8    DC      /0008
025E    0000            BSS   E 0
025E 0  0000    MODE    DC      0           NORMAL OR TEST MODE STORED
025F 0  0000    CN01    DC      0           NORMAL OR TEST WRITESTORED
0260 0  0000    CN02    DC      0           NORMAL OR TEST READ STORED
0262    0000            BSS   E 0
0262 0  6D01    TMODE   DC      /6D01       TEST MODE
0263 0  2005    TWRIT   DC      /2005       TEST WRITE
0264 0  2012    TREAD   DC      /2012       TEST READ
0266    0000            BSS   E 0
0266 0  2D01    NMODE   DC      /2D01       NORMAL MODE
0267 0  2001    NWRIT   DC      /2001       NORMAL WRITE
0268 0  2002    NREAD   DC      /2002       NORMAL READ
026A                    END
```

SYMBOL TABLE

	BCNT	0234		BHTAB	00FF		BHTY	00E7		BHTYP	00E4		CCWWE	0246
	CCWW1	0237		CCWW2	023D		CCWW5	024C		CCWW7	0252		CLOSE	008E
	CN01	025F		CN02	0260		COND	00BC		CONUS	00B9		CON01	025A
	CON02	025B		CON03	025C		CON8	025D		DADD	0236		EMES2	0195
	EMES5	01C5		EMES7	0173		EMES8	0184		EMS2A	0197		EMS2B	01B0
	EMS2C	01C5		ERPT1	00CA		ERRPT	00C5		ERR2	00D4		HBTA1	00FB
	IFEOT	00BD		IOCWA	022A		IOCWB	022C		IOCWC	022E		IOCWD	0230
	IOCWE	0232		IOCW2	021E		IOCW3	0220		IOCW4	0222		IOCW5	0224
	IOCW6	0226		IOCW7	0228		MODE	025E		MODIO	0078		NMODE	0266
	NORML	0049		NREAD	0268		NWRIT	0267		OPENR	0052		OPENW	003A
	RETRY	011A		SOPEN	00BB		TAG	005C		TMODE	0262		TPEXT	00B4
	TPIN0	0092		TPIN6	010F		TPIN8	0128		TPSTR	0000		TPT1	0017
	TPT2	0013		TPT3	0028		TPT4	0086		TPT5	0007		TPT6	000B
	TPT7	002D		TPT8	0035		TPT9	0080		TPW2	008A		TREAD	0264
	TSTUS	00AE		TST2A	01C7		TST2B	01D0		TST2C	01E2		TST2D	01F0
	TST2E	01FC		TST2F	0208		TST2G	0212		TST2H	021C		TST2I	021E
	TST3A	0175		TST3C	0184		TST4A	0186		TST4B	0195	A	TVEXT	00AD
A	TVSAV	00AC		TWRIT	0263		WAITC	016A		WBBSY	00BA	A	WK5	0037
	WORK1	0255		WORK2	0256		WORK3	0257		WORK4	0258		WORK7	0259
	WR$RD	0235												

NO ERRORS IN ABOVE ASSEMBLY.

TPSTR

DUP FUNCTION COMPLETED

SESSION REPORT

COMMON - Chicago

Session Number MON D3 Session Name 1620 Project

Chairman H. B. Kerr

Time 3:30 to 5:00 PM Attendance (No.) 48

Speakers Peter Jutsum

John Powell and Richard LaRue

Lanny Hoffman

Synopsis of Meeting Well attended. Good presentations. Spirited discussion.

Using the 1620 as a Data Collection Device

by

J. Cooper and L. Hoffman
Guggenheim Labs
Forrestal Campus
Princeton University
Princeton, N.J.

Our interest in data collection with our 1620 Computer dates back to about 1963 when we asked IBM about an analog-digital converter (ADC) to connect to our 1620. At that time, the 1711 ADC was slow and expensive (about 20 samples per second). We needed at least two orders of magnitude faster than that. So we did nothing in the area of automatic data collection for several years. Then, in 1965, we found an entire stand-alone data collection system in the surplus listings. The system was an old CEC Millisadic contained in three large cabinets. It was a vacuum tube system with a multiplexor of 100 inputs, a clock, an analog-digital converter, control circuitry, an intermediate storage digital tape, and a serial to parallel converter for punching cards on a 523 summary punch. We were able to break the system into several parts which now constitute our present data collection system. The 1620 requires serial BCD data so that we were able to essentially substitute the 1620 in place of the Millisadic digital tape. The output of the Millisadic control unit is from a blocking oscillator, which is a highly damped oscillation from +150 to -250 volts. The input levels to the 1620 are 0 to -10 volts, so that this was the starting point of the interface. Mr. Cooper designed and built the present interface, which will be described in a later section. The output of the Millisadic control unit is BCD, so no code conversions were necessary. Also, there is a pulse available that occurs for every digit and a pulse that occurs at the end of 10, 15, or 20 samples or at the end of commutation of the multiplexor. The digit pulse is used to tell the 1620 that there is a digit on the input lines and the end of the scan or commutative pulse is used as an end-of-line (EOL) to terminate the read instruction for one mode of operation. The input to the 1620 is through the paper tape input lines. We have a paper tape reader but we have put a set of relays in the signal lines which will transfer the input from either the paper tape reader or the ADC. To use the paper tape inputs without having the real paper tape reader active,

certain control gates on the 1620 had to be artifically turned on. These are merely connected to the correct voltage when the relays are activated to switch input lines.

We have basically two modes of operation of the data collection system; record mode and burst mode. The record mode automatically samples only 10, 15, or 20 samples with one read instruction and then disconnects for processing by the program. Burst mode will continuously enter data into core since no EOL is present. The read instruction is terminated by manual operation by the "Release" key on the console of the 1620. Data can wrap around core if the operator does not watch the Memory Address Register (MAR). The core address of each digit is visible in the MAR so that the operator has an indication of how far the data is in core.

The basic differences in the two methods are:

1) burst mode has a limited total sample time (core) while record mode has an unlimited total sample time,

2) burst mode provides an equal sampling interval for as long as data may be taken, but record mode sampling interval depends on the amount of processing done on each record and can never provide equal sampling intervals due to the need for synchronization of the 1620 and the ADC.

To provide an idea of the data rates we use, several examples follow:

1) we can sample one channel of information at 1200 samples per second, so burst mode can only read about 8 seconds of data.

2) we can sample multiple channels at a rate of 400 samples per second, so that burst mode reads about 24 seconds of data.

3) record mode is used by one experiment where we sample for about six hours and store the information on disk.

4) in another experiment, the analog information is read from an analog tape recorder at such a slow rate that we can use the 1620 as a programmable card punch.

The synchronization of the ADC and the 1620 is quite simple; for burst mode, the 1620 must be started before the ADC (the 1620 waits for a "sync" pulse from the ADC), and for record mode, two sequential "read" instructions (RNPT) will insure that the data is placed in the correct core locations. (The first read can come at any point in the ADC scan since the

ADC is free-running, thus, the second read will start at the beginning of the scan.)

There are two general purpose programs in use for this system. One is a FORTRAN subroutine that returns an integer array of the data and the time of reading the data. This, of course, operates only in record mode. The other general purpose program is written in SPS and uses the typewriter for logging run numbers, titles, etc. and stores the information on disk. This program uses burst mode. Many runs may be stacked on a disk and operator messages keep the system and operator in synchronization. The data stored on disk is very basic; time and data are stored with no separation into channels since we do not want to occupy the experimental facilities any longer than necessary. A second program, which is rather specialized for each experiment, (but based on a single package) is then run to produce cards with channel separation if necessary. Each data card is of a standard format, 8 values of time and the data value at those times, with each card completely identified with a run number, tape position, channel number, and sequence number. Each set of cards also has a title card as the first card. Our plot routines and most data reduction programs have this format for the data input so that some very basic operations may be performed on the data with no special programming necessary.

Four programs are listed in Appendices I-IV. Appendix I is a short Fortran program to read 10 strip chart recorders and punch the data on cards on command. The subroutine "ADC" is an SPS subroutine listed in Appendix II. The arguments are time, data values, and number of channels, all integers. The number of channels read under record mode is 10, 15, or 20, depending upon certain switch settings in the A-D converter. Subroutine ADC determines the setting by checking the position of the record mark generated by the EOL. The subroutine is written to allow the use of non-standard "f and k" precision by changing only the subroutine header card. Appendix III is our general purpose data collection routine for burst mode data. The program should have an additional section added to allow decoding of any multiplexing arrangement. The checking for this is in the program but the decoding section has not been written. We have not found a need for decoding at data time because of the extra waiting time that the test cells would have while the 1620 punched cards. Core is

initialized to flag zeros so that it is possible to determine the amount of data entered. Appendix IV is a special program to sample and store data on the disk over approximately a six hour time period. Twenty channels of data are read and stored if there is any channel with a non-zero value. Four sets of samples are accumulated and stored on the disk. All processing for twenty channels must occur in less than one ADC record time, 50 milliseconds. Certain disk functions are broken into segments to keep processing time at a minimum. No loops are used since we need every machine cycle we can save. The timer is not used here because it is more accurate and faster to just number the records.

The ADC Interface System:

The ADC Interface System was designed and built in the summer and fall of 1965, before the era of inexpensive integrated circuits. Because IC's were not at that time competitive in the small quantities involved, the interface was designed and built using discrete components.

The Millisadic (ADC) Output

The Millisadic output lines of interest provide four digits (1, 2, 4, & 8) and an end-of-scan. A binary zero is indicated on these lines by the presence of a blocking oscillator pulse having a peak to peak amplitude of 225 volts and a total duration of 5 microseconds, as measured with a 20 kilohm load. (See Fig. 1.) A binary one is indicated by the absence of a pulse (ground level).

The 1620 Input Requirements

The 1620 input lines require a rectangular pulse having a rise and fall time of less than 5 microseconds, a nominal amplitude of 10 volts and a duration of more than 3 machine cycles (60 μsec.). A binary zero is indicated on these lines by a -9 volt (nominal) signal. A binary one is indicated by placing the lines at ground level (0 volts nominal). The 1620-1621 interconnection has 12 lines, of which eleven (eight data and three control) must be actuated in order to enter data into the 1620. The eight data lines are the five digits (0, 1, 2, 4, & 8), the check (C) bit, the X bit and the end-of-line (EOL) signal. The three control lines are the Tape Synch Exit (digit pulse), Nonprocess Runout (NPR) and the Reader Ready (RDRY). The twelfth line carries the Clutch Drive signal to the 1621 PTR and does not require interfacing.

Interface System Functions

The interface system must perform three functions: Convert the Millisadic output pulses into signals electrically acceptable to the 1620, generate the additional data and control signals required by the 1620, and provide for easy transfer of the 1620 input between the Millisadic and the 1621 PTR.

Interface System Construction

In designing the interface, the option existed to either derive the additional data and control signals using the vacuum-tube techniques employed in the Millisadic and then convert to the transistor logic levels required by the 1620, or to first convert the Millisadic output to the 1620 transistor logic levels and then generate the additional data and control signals. The later approach was selected in order to take advantage of the compactness, reliability and economies inherent in semiconductor circuitry.

The interface circuitry was contructed on 6-3/4" x 4-3/4" single-sided phenolic plug-in circuit boards. These are housed in a 6-3/4" x 19" relay rack card holder containing provisions for up to 17 circuit boards. Input, output and intermediate test points are available on the front panel along with the EOL control switch. Inexpensive commercial components were used, and no selection of components, including transistors, was required. The output stages drive the more than 50 feet of Belden #8753 cable (11 pair telephone type, unshielded) connecting the interface with the ADC-PTR transfer relay box (located near the PTR). Ten volt zener diodes are connected across all the interface output stages to prevent positive or excessive negative voltages at the 1620 input lines in the event of component failure. The interface power requirements of +12VDC @ 100 ma. and -12VDC @ 50 ma. are readily supplied from the 1620 "D" Gate power supply. The transfer relays draw an additional 300 ma from the "D" 12VDC Volt supplies when the ADC is in use. One switch on the 1620 console is used to select between the 1621 PTR and the ADC.

Interface System Analysis:

A functional block diagram of the ADC Interface System is included as Figure 2. The four digit (1, 2, 4, 8) and end-of-scan lines of the Millisadic output are fed to identical 75 μsec. "pulse conditioners". The output of these units is electrically acceptable to the 1620. They deliver a binary zero 75 microseconds long to the 1620 everytime an input pulse is received from the Millisadic. The conditioned end-of-scan signal is suitable for use as an EOL signal. By means of a front panel switch the operator can select the EOL signal or connect the 1620 EOL input line to a binary zero (-12VDC) for burst mode operation.

The 75 microsecond conditioner is designed to trigger from the fast, positive rise portion Millisadic output pulse. This is accomplished by differentiating the input pulse, selecting polarity, and limiting amplitude. This conditioned pulse is then used to trigger a 75 microsecond monostable multivibrator whose output is fed to an inverting line driver. The pulse conditioner schematic is shown in Figure 3.

The absence of a zero digit line from the Millisadic output requires generation of the zero bit within the interface. The outputs of the 1, 2, 4, and 8 pulse conditioners are fed to the zero bit generator which generates a binary one on the 1620's zero-bit line only when a binary zero is present on all of the other four digit lines. The zero bit generator is a four input TDL NOR circuit whose schematic is included as Figure 4.

The absence of a "C" bit from the Millisadic output requires it also be generated within the interface. The 1620 logic requires odd parity amongst the 1, 2, 4, 8, 0, X and C bit lines. Since only numbers are being entered into the 1620, the X bit line is set at a binary zero (-12VDC) in the ADC-PTR transfer relay box when ADC operation is selected. Since the X-bit is fixed during ADC operation, only the 1, 2, 4, 8, and 0 bit lines need be connected to the check bit generator.

The check bit generator is composed of four cascaded RTL EX-OR

circuits followed by an inverting line driver. A block diagram of the check bit generator is shown in Figure 5, and schematic diagrams are given in Figures 6 and 7. The check bit generator delivers a binary one to the 1620 C bit line when any even number of binary ones occur on the 1, 2, 4, 8 and zero bit lines, and a binary zero for any odd number of binary ones.

A special connection was brought out from the Millisadic so as to derive a synch (digit) pulse. This pulse occurs for every digit and can be used to tell the 1620 that there is a digit on its input lines. The 1620 synch line requires a binary one that becomes a zero halfway thru the machine cycle. The digit pulse from the Millisadic is fed to a 10 μsec. pulse conditioner followed by a non-inverting line driver. A schematic of the synch pulse conditioner is shown in Figure 8.

The remaining two 1620 control lines (NPR and RDRY) are automatically connected to the -12VDC bus at the transfer relay box when the ADC is selected. This supplies the binary zeros required on those lines for proper operation in the ADC position.

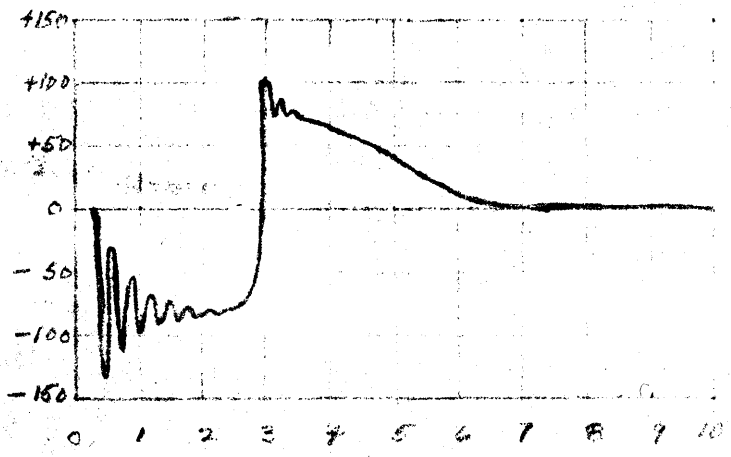
Switching between the ADC and PTR is accomplished in the transfer relay box by three 6-Pole double throw relays. Besides switching the 1620 input lines between the PTR and ADC, they also apply power to the ADC interface system. A schematic of the relay transfer box is shown in Figure 9.

A power supply board, used to provide power supply decoupling for the interface system, is shown in Figure 10.

The Interface System has been in operation since the winter of 1965. Its zero failure record during the past three years has given us confidence in its reliability, and we think the future holds great promise for us in the use of this equipment. Even though our equipment is old and somewhat hard to use, we feel that our students have a good understanding of data collection equipment when they finish a set of runs.

HORIZONTAL - 1 Microsecond / Division

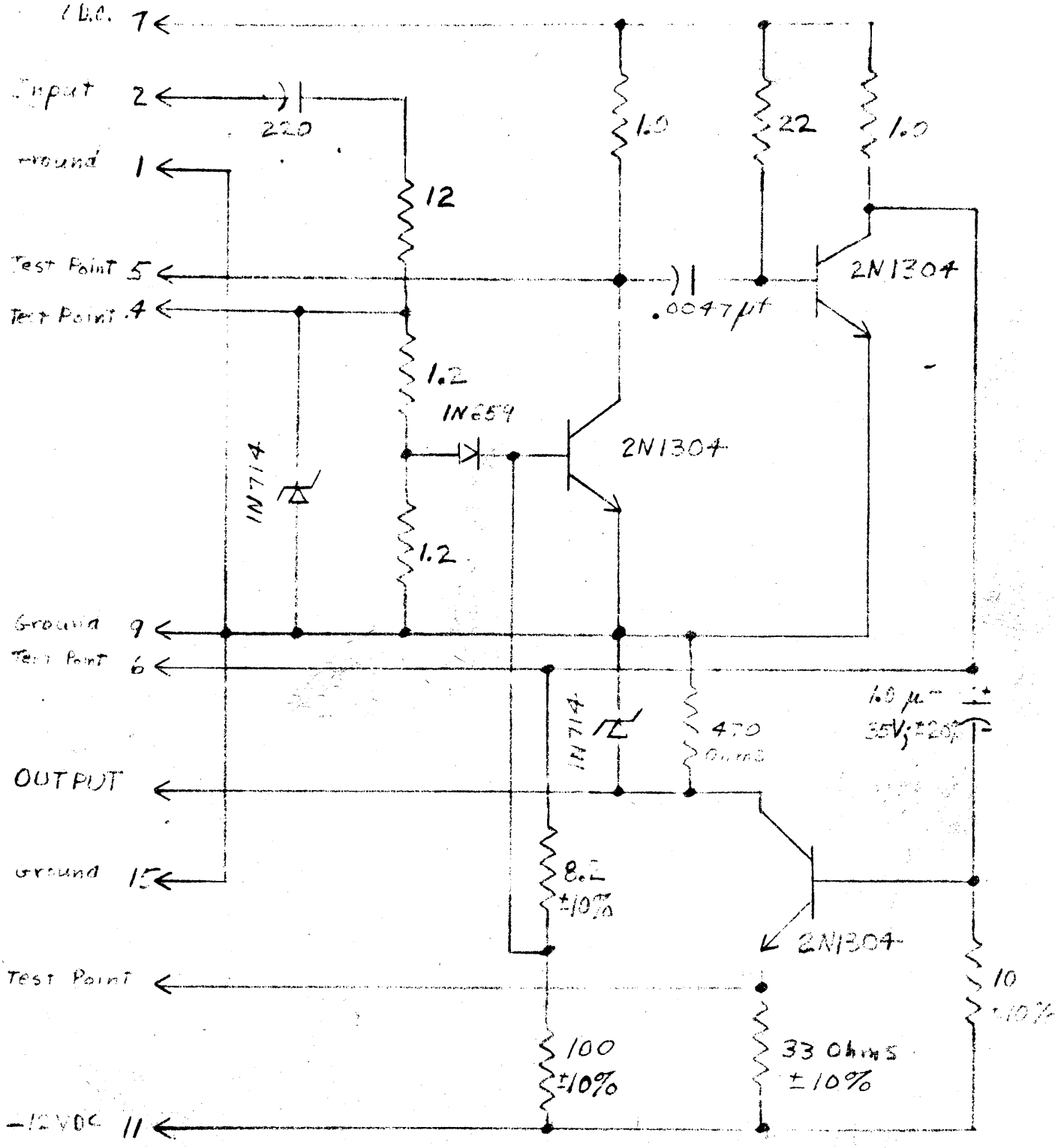
VERTICAL - 50 Volts / Division



Note: Output loaded with 50 ohms.

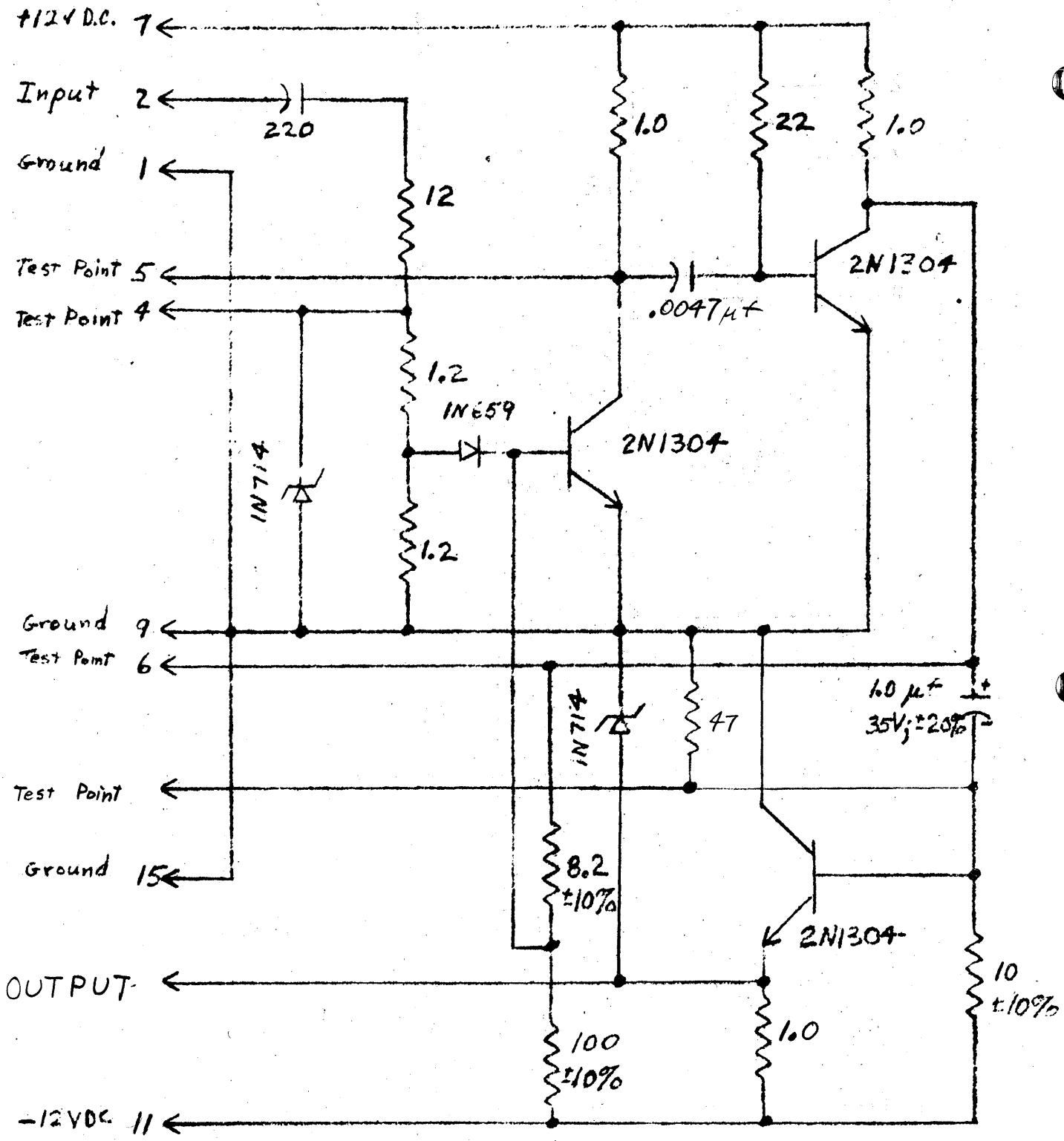
GUGGENHEIM LABORATORIES DEPARTMENT OF AERONAUTICAL ENGINEERING PRINCETON UNIVERSITY	TITLE MILLIMADIC OUTPUT PULSE		
	DRAWN BY [illegible]	DESIGN APPR.	ENG. APPR.
	DATE 10/16/[illegible]	SK. NO. FIGURE 1	

D & W BLUEPRINT CO. 3768-4



Notes:
1. All resistor & capacitor tolerances ±5% except as noted.
2. All resistors are ½ W. Composition with values in Kilohms except as noted.
3. All capacitors are silver mica with values in pf except as noted.

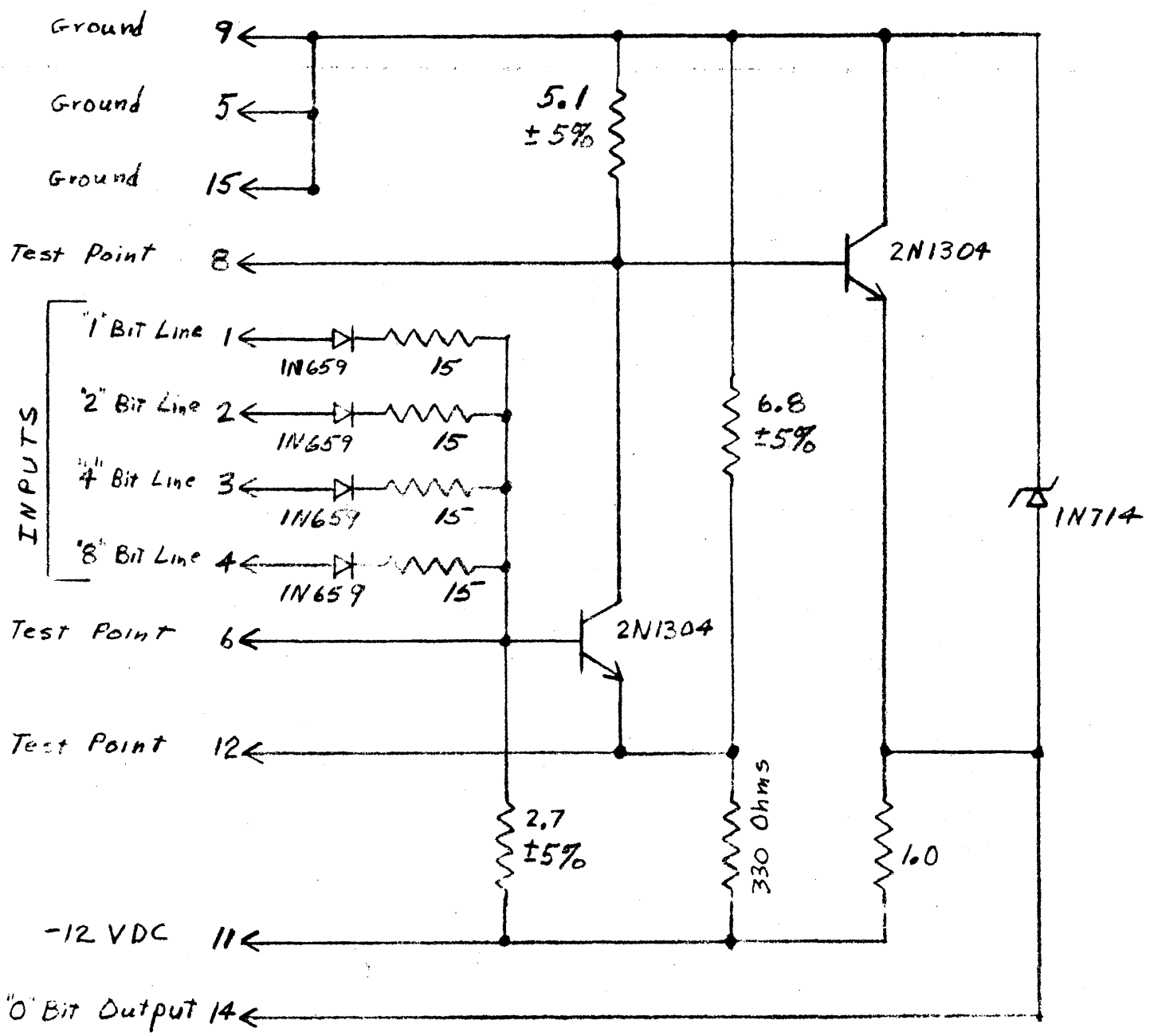
GUGGENHEIM LABORATORIES DEPARTMENT OF AERONAUTICAL ENGINEERING PRINCETON UNIVERSITY	TITLE 75° Microsecond Pulse Conditioner		
	DRAWN BY [illegible]	DESIGN APPR. [illegible]	ENG. APPR. [illegible]
	DATE 11/26/64	SK. NO. FIG. #3.	



Notes:
1. All resistor & capacitor tolerances ±5% except as noted.
2. All resistors are ½ W. Composition with values in Kilohms except as noted.
3. All capacitors are silver mica with values in μf except as noted.

GUGGENHEIM LABORATORIES DEPARTMENT OF AERONAUTICAL ENGINEERING PRINCETON UNIVERSITY	TITLE 75 μ sec EOL Pulse Conditioner	
	DRAWN BY [illegible]	DESIGN APPR. [illegible] ENG. APPR. [illegible]
	DATE 11/26/64	SK. NO. FIG. #3A

D & W BLUEPRINT CO. 8706-4

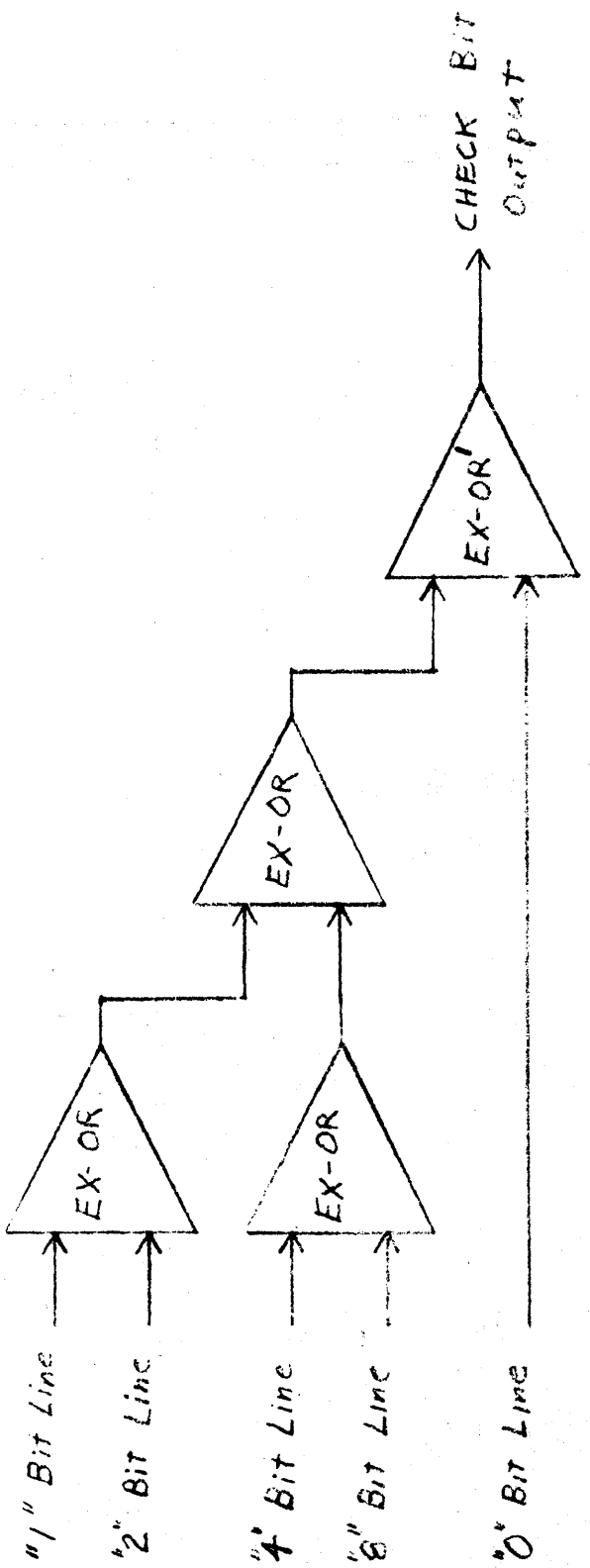


NOTES:

1. All Resistors are ½ Watt, ±10% Composition; Values given in Kilohms except as noted.

GUGGENHEIM LABORATORIES DEPARTMENT OF AERONAUTICAL ENGINEERING PRINCETON UNIVERSITY	TITLE ZERO BIT GENERATOR		
	DRAWN BY JBC	DESIGN APPR. JBC	ENG. APPR.
	DATE 12/64	SK. NO. FIGURE 4.	

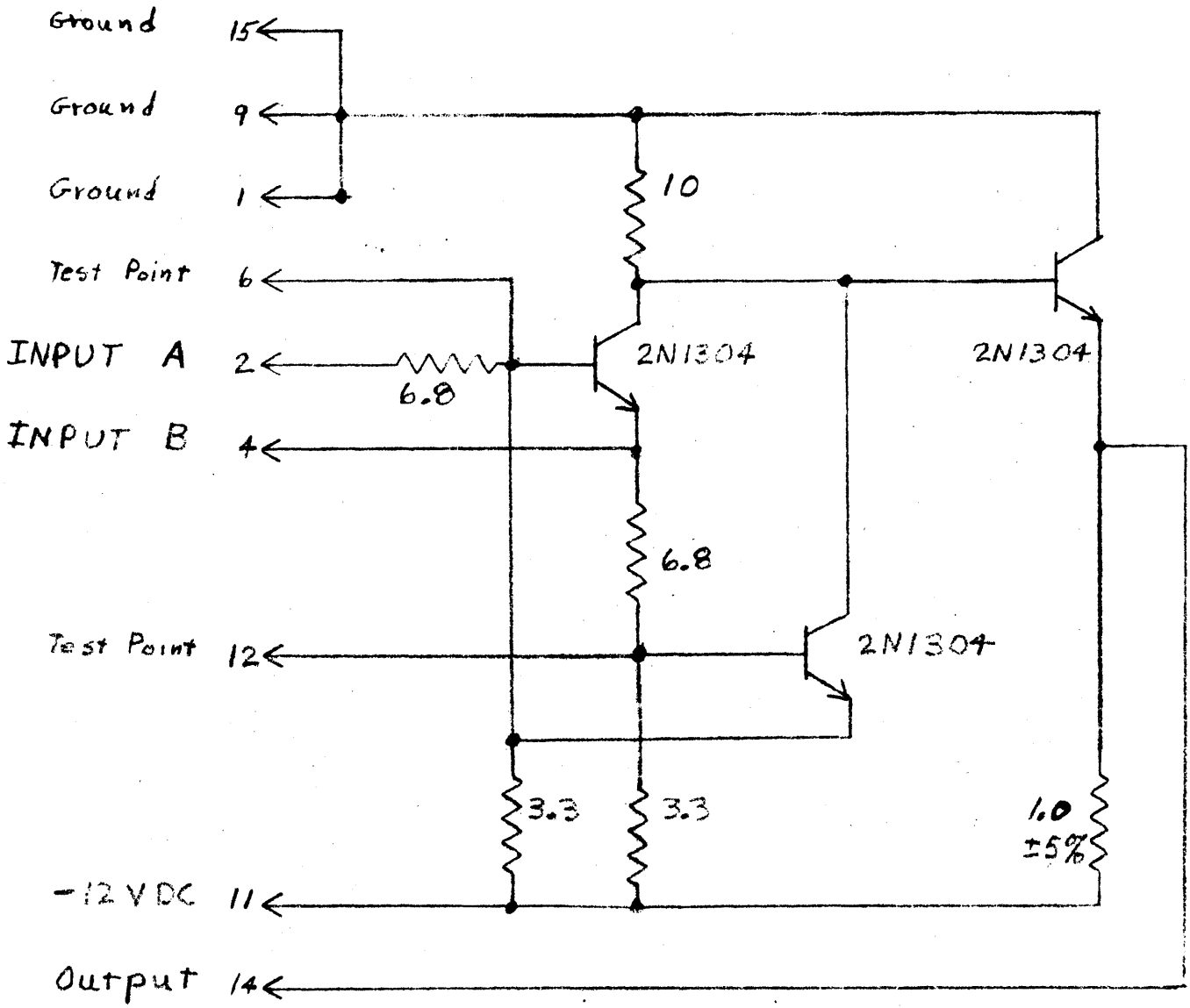
D & W BLUEPRINT CO. 3765-4



GUGGENHEIM LABORATORIES DEPARTMENT OF AERONAUTICAL ENGINEERING PRINCETON UNIVERSITY	TITLE CHECK BIT GENERATOR LOGIC			
	DRAWN BY		DESIGN APPR.	ENG. APPR.
	DATE 1/65	SK. NO. FIGURE 5.		

D & W BLUEPRINT CO. 3788-4

10140



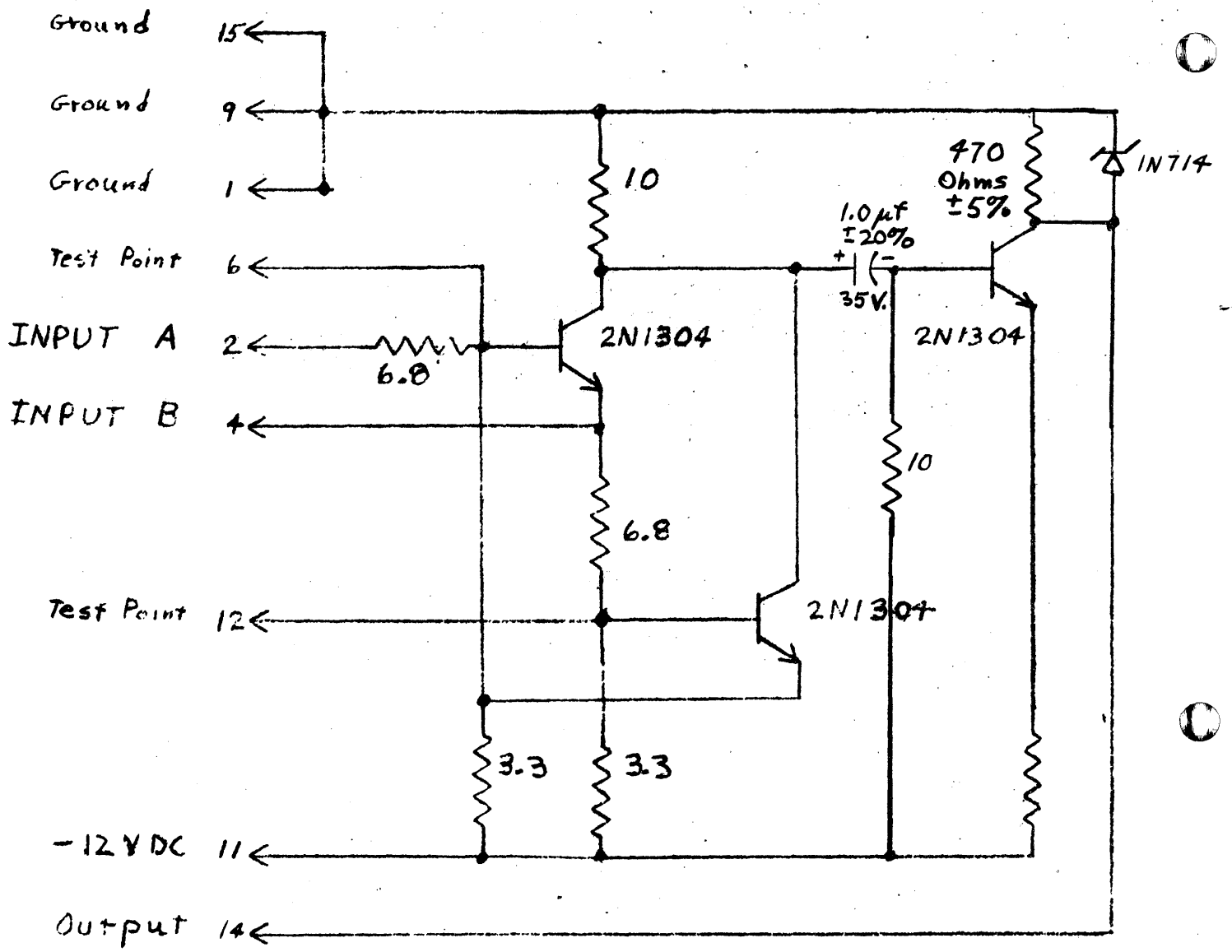
Notes:

1. All resistors are ½ Watt composition ±10%, Values in Kilohms except as noted.

TRUTH TABLE		
A	B	OUT
-10	-10	0
0	-10	-10
-10	0	-10
0	0	0

GUGGENHEIM LABORATORIES DEPARTMENT OF AERONAUTICAL ENGINEERING PRINCETON UNIVERSITY	TITLE RTL EX-OR CIRCUIT SCHEMATIC	
	DRAWN BY [illegible]	DESIGN APPR. [illegible] ENG. APPR. [illegible]
	DATE 1/65	SK. NO. FIGURE 6.

D & W BLUEPRINT CO. 8765-4



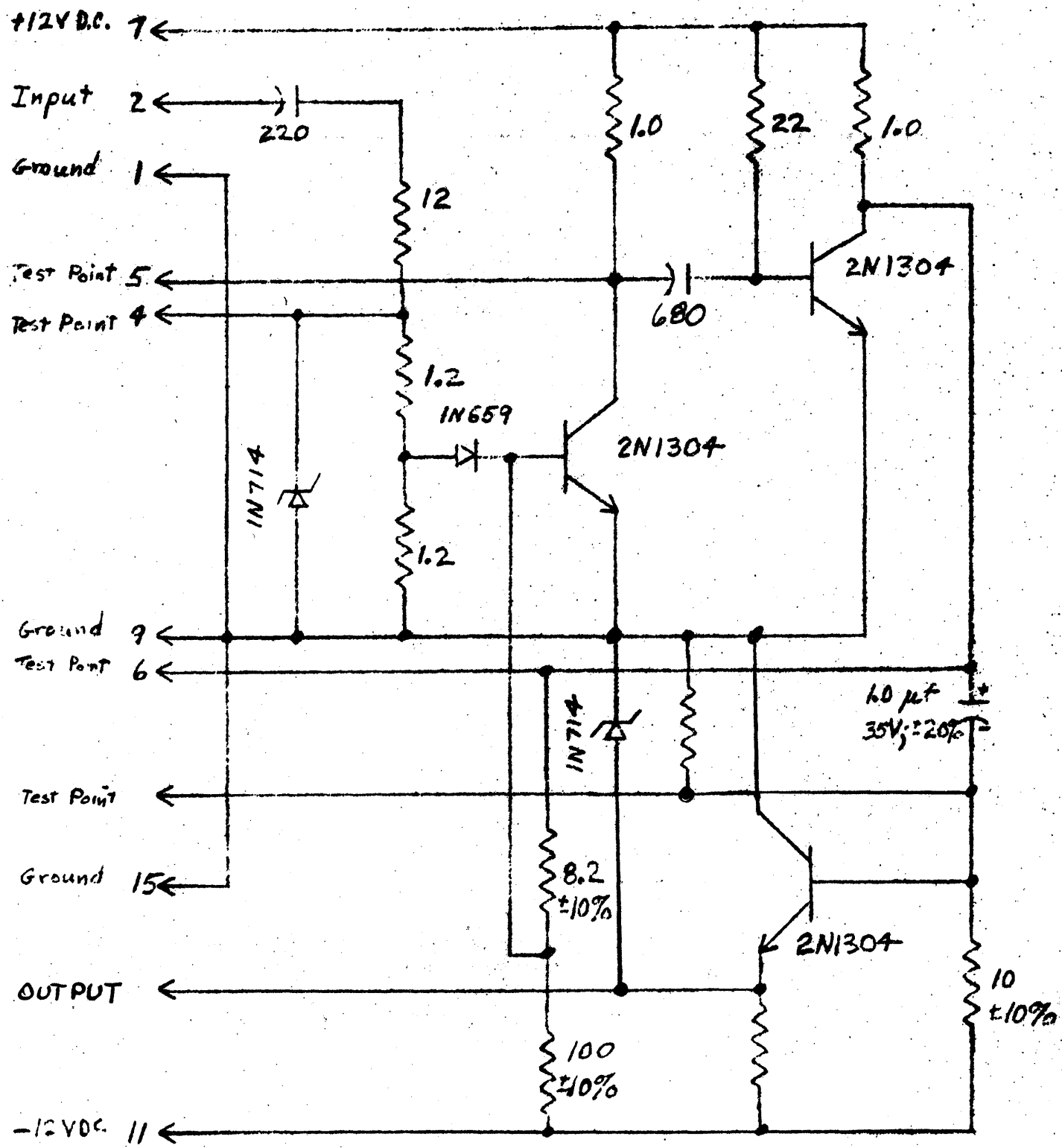
Notes:

1. All resistors are ½ Watt Composition ± 10%, Values in Kilohms except as noted.

TRUTH TABLE		
A	B	OUT
+10	-10	-10
0	-10	0
-10	0	0
0	0	-10

GUGGENHEIM LABORATORIES DEPARTMENT OF AERONAUTICAL ENGINEERING PRINCETON UNIVERSITY	TITLE RTL EX-OR CIRCUIT SCHEMATIC		
	DRAWN BY JBC	DESIGN APPR. JBC	ENG. APPR. JBC
	DATE 1/65	SK. NO. FIGURE 7.	

B & W BLUEPRINT CO. 8765-4



Notes:
1. All resistor & capacitor tolerances ±5% except as noted.
2. All resistors are ½ W. Composition with values in Kilohms except as noted.
3. All capacitors are silver mica with values in μf. except as noted.

744

GUGGENHEIM LABORATORIES DEPARTMENT OF AERONAUTICAL ENGINEERING PRINCETON UNIVERSITY	TITLE 10 μ Sec Synch Pulse Conditioner		
	DRAWN BY BP	DESIGN APPR. JBT	ENG. APPR. JBT
	DATE 11/26/64	SK. NO. FIG. #8	

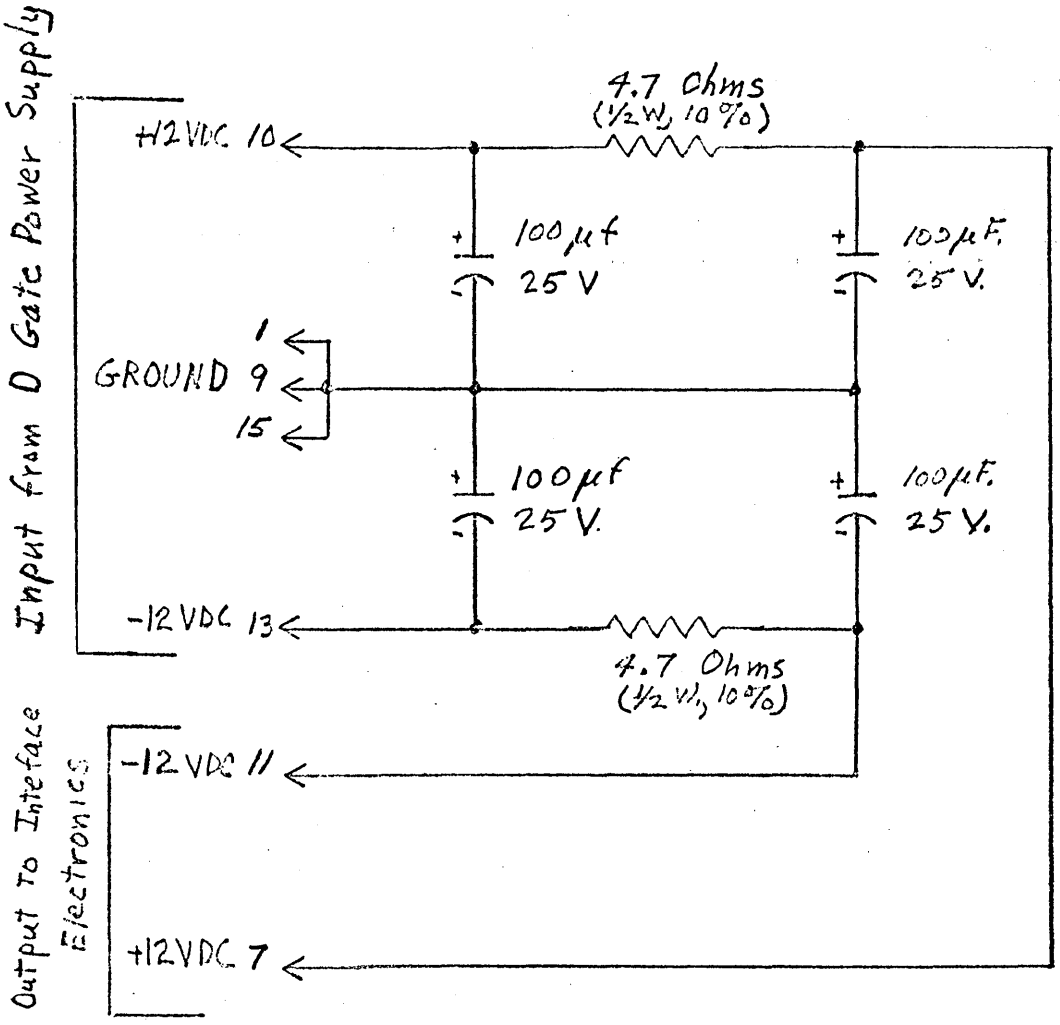
B & W BLUEPRINT CO. 6705-4.

INTERFACE ELECTRONICS

GUGGENHEIM LABORATORIES
DEPARTMENT OF AERONAUTICAL ENGINEERING
PRINCETON UNIVERSITY

TITLE	Millisadic (ADC) Interface Sys
DRAWN BY	
DESIGN APPR.	
ENG. APPR.	
DATE 7/21/62	SK. NO. FIGURE 2.

D & W BLUEPRINT CO. 3765-4



GUGGENHEIM LABORATORIES DEPARTMENT OF AERONAUTICAL ENGINEERING PRINCETON UNIVERSITY	TITLE POWER SUPPLY FILTER		
	DRAWN BY JLC	DESIGN APPR. JLC	ENG. APPR. JLC
	DATE 1/65	SK. NO. FIGURE 10.	

D & W BLUEPRINT CO. 3765-4

APPENDIX I

A SAMPLE PROGRAM TO USE A SUBROUTINE TO READ THE A-D CONVERTER AND PRESENT THE DATA IN AN INTEGER ARRAY.

```
C       PROGRAM TO READ TEN STRIP CHART RECORDERS.....................
        DIMENSION IX(20),IY(10)
        EQUIVALENCE (IX,IY)
        TYPE 4
    4   FORMAT(16H TURN SW. 1 ON .)
        PAUSE
    1   CONTINUE
C       IF SENSE SWITCH 1 IS ON , DO NOT READ CONVERTER.
        IF (SENSE SWITCH 1)1,2
    2   CONTINUE
        CALL ADC(ITIME,IX,IN)
C       PUNCH ONLY 10 RECORDERS UNDER ANY CIRCUMSTANCES.
        PUNCH 3,ITIME,IY
    3   FORMAT(I10,10I5)
        GO TO 1
        END
```

APPENDIX II

AN S.P.S. SUBROUTINE FOR FORTRAN TO READ
10, 15, OR 20 CHANNELS OF DATA, DETERMINE THE
NUMBER OF CHANNELS, AND PRESENT THE DATA
AND CONVERTER TIME IN A FORTRAN
INTEGER ARRAY.

```
AUTO        DORG11036
AUTO ADC    DS  5
AUTO ITIME  DS  5
AUTO IX     DS  5
AUTO IN     DS  5
AUTO        DS  7
     *              END OF ARGUMENT ADDRESSES
     *                START LINKAGE.
AUTO START  AM  START-1,5,010
AUTO        TF  ITIME ,START-1,0111
AUTO        BNF *+36,ITIME ,01
AUTO        CF  ITIME ,,0
AUTO        TF  ITIME ,ITIME ,0111
AUTO        AM  START-1,5,010
AUTO        TF  IX    ,START-1,0111
AUTO        BNF *+36,IX    ,01
AUTO        CF  IX    ,,0
AUTO        TF  IX    ,IX    ,0111
AUTO        AM  START-1,5,010
AUTO        TF  IN    ,START-1,0111
AUTO        BNF *+36,IN    ,01
AUTO        CF  IN    ,,0
AUTO        TF  IN    ,IN    ,0111
AUTO        AM  START-1,2,010
AUTO        B   AROUND,,0
AUTO        DORG*-3
     *              SYMBOL TABLE AND CONSTANTS HERE ...
     KK     DS  2,404
     INPUT  DS  1
            DS  120
     TIME   DC  10,0
     ZERO   DC  7,0
     FLOC   DS  5
     AROUNDRNPTINPUT,,0,, FIRST READ TO SYNCRONIZE A-D CONV.
            RNPTINPUT,,0,, SECOND READ TO GET DATA FROM ADC.
            BI  AROUND,00600,0,, READ AGAIN IF READ PARITY ERROR.
            TD  TIME,INPUT+35,01
            TD  TIME-1,INPUT+29,01
            TD  TIME-2,INPUT+23,01
            TD  TIME-3,INPUT+17,01
            TD  TIME-4,INPUT+11,01
            TD  TIME-5,INPUT+5,01
     *          FIND FLAG SET LOCATION IN FIELD TIME.
            TFM FLOC,TIME+1,017
            S   FLOC,KK,0
            SF  FLOC,,06
            TF  ITIME,TIME,016,, STORE FORTRAN TIME IN CORRECT LENGTH.
            A   IX,KK,0
            TF  TIME-3,ZERO,01,, SET UPPER TIME IEMP. TO ZERO.
            TFM LOOPSF+6,INPUT+2,017
            TFM COUNT+11,INPUT+4,017
```

```
LOOPSF SF
COUNT  TF   TIME,,0
       CF   TIME-2,,0
       SF   FLOC,,06
       TF   IX,TIME,016,, STORE FORTRAN SAMPLE IN ARRAY IX.
       A    IX,KK,0
       AM   COUNT+11,1,010
       BNR  *+20,COUNT+11,0111,, CHECK END OF SAMPLE.
       B    END,,0
       DORG*-3
       AM   LOOPSF+6,6,010
       AM   COUNT+11,5,010
       B    LOOPSF,,0
       DORG*-3
END    SM   COUNT+11,INPUT+59,017
       BZ   N10,,0
       SM   COUNT+11,30,010
       BZ   N15,,0
N20    TFM  TIME,20,08
       B    SFIN,,0
       DORG*-3
N15    TFM  TIME,15,08
       B    SFIN,,0
       DORG*-3
N10    TFM  TIME,10,08
SFIN   CF   TIME-3,,0
       SF   FLOC,,06
       TF   IN,TIME,016
       B    START-1,,06
       DEND
```

APPENDIX III

A GENERAL PURPOSE S.P.S. DATA COLLECTION PROGRAM FOR BURST MODE DATA FROM THE A-D CONVERTER. SEVERAL RUNS MAY BE STACKED ON THE DISK.

```
*          DATA COLLECTION PROGRAM FOR MULTICHANNEL OR SINGLE
*               CHANNEL OPERATION.
       DORG402
TYPEHDBTM HEADER,*+12,,  GO TO LABEL, TAB, DATE SUBROUTINE.
*                              SEND ADDRESS OF NEXT INSTRUCTION
       SF  NUMBLK-49,,,    SET FIELD LIMITER ON NUMERIC BLK. STRING.
       TFM TYPEHD+1,41,10,'NOP' THE HEADER ROUTINE.
       TF  LASTAD,MEMCAP,,ASSUME LAST FILLED ADDRESS IS UPPER MEMORY.
       TF  OLDSEC,NXTSEC,,, STORE BEGINNING SECTOR OF DATA IN OLDSEC.
*                           INITIALIZE INPUT RUN DATA AREAS.
       RCTY
       TBTY
       WATYLABEL
       TBTY
       TR  INPUT,FLGZER
       TR  INPUT+47,FLGZER
       RATYINPUT
       BC4 *-48
       TDM INPUT+92
       DC  1,',*
       TR  CARD,INPUT-1
       TDM CARD+93,,,          STORE A RMK IN CARD ALSO TO ALLOW LAST.
       DC  1,',*
       BTM OUTPUT
       TDM CARD+93,0,,,        REMOVE RMK IN OUTPUT AREA.
MEMINITFM COUNT,INPUT,,  BEGIN MEM. INITIALIZATION TO FLG. ZEROS.
       TR  COUNT,FLGZER,6,TRANSMIT 50 FLG. ZEROS AND ONE RMK.
       AM  COUNT,50,10,,  INCREMENT COUNTER BY 50 TO REMOVE RMK.
       C   COUNT,LASTAD,, HAVE ALL USED INPUT POSITIONS BEEN CLEARED.
       BP  INIOUT,,,       IF SO, LEAVE LOOP.
       B   MEMINI+12,,,
       DORG*-3
INIOUTRCTY,,,,             RETURN TYPE. CARR. TO ACCEPT RUN DATA.
       TFM RUN,0,7,,       RUN NUMBER (5 DIGITS)
       TFM TPOS,0,8,,      TAPE FOOTAGE (4 DIGITS)
       TF  CHANNL,ZER10,,  CHANNEL CONFIGURATION (10 DIGITS)
*                            INITIALIZED TO RMK'S.
       TFM SPERC,0,10,,    SAMPLES PER CARD (2 DIGITS)
       TFM CPERS,0,10,,    CARDS PER SECOND (2 DIGITS)
       BTM TYPEIN,RUN,,    ACCEPT RUN NUMBER (FROM 1-5 DIGITS)
       BTM TYPEIN,TPOS,,   ACCEPT TAPE FOOTAGE (1-4 DIGITS)
       BTM TYPEIN,CHANNL,,ACCEPT CHANNEL CONF. FOR CHAN. 0-9
*                            ALL 10 DIGITS MUST BE FILLED.
*                            IF A CHANNEL POSITION IS NOT USED,
*                            THAT POSITION MUST HAVE A RMK TYPED IN.
*            NOTE THAT IF ONLY ONE CHANNEL (NON-COMMUTATED) IS USED,
*            THEN ONLY ONE DIGIT NEED BE TYPED.
*
       BTM TYPEIN,SPERC,,  ACCEPT SAMPLES PER CARD (2 DIGITS)
*                            THIS IS USED TO FIND TIME LOCATION.
       BTM TYPEIN,CPERS,,  ACCEPT CARDS PER SECOND (2 DIGITS)
       CF  RUN-4,,,        REMOVE FLAGS FROM ALL RUN
       CF  TPOS-3,,,       DATA TO ALLOW A NUMERIC
```

```
       CF   CHANNL-9,,,     PUNCH OF ALL DATA AS
       CF   SPERC-1,,,      INTEGERS FOR FORTRAN.
       CF   CPERS-1,,,
       SF   CARD1-1
       TDM  CARD1+78,,,,
       DC   1,',*,,
       TF   CARD+79,CARD1+78,,, MOVE OUTPUT RECORD TO DISK I/O AREA.
       CF   CARD
       BTM  OUTPUT,,,,      HEADER CARD OF DATA IN CASE MEMORY IS BOMBED.
       TDM  CARD+79,0,, REMOVE RECORD MARK IDENT. FOR LABEL CARD.
       SF   RUN-4,,,        RESTORE FIELD FLAGS TO RUN DATA.
       SF   TPOS-3,,,
       SF   CHANNL-9,,,
       SF   SPERC-1,,,
       SF   CPERS-1,,,
HOP    RCTY,,,,             RETURN TYPE CARR. FOR OPERATOR MESSAGE.
       RCTY
       WATYOPINST,,,        OPER. INSTRUCTIONS.......
       H    ,,,             ALLOW OPERATOR TO DO ABOVE INSTRUCTIONS.
       BC1  *+32,,,         USE CONSOLE SWITCH TO CHOOSE CARD OR ADC
       RNPTINPUT,,,         SELECT A.D.C. AND WAIT FOR IT TO START.
       B    *+56,,,
       DORG*-3
*****************************************************************
       TFM  COUNT,INPUT,,      INPUT
       RNCDCOUNT,,6,,              CORE
       AM   COUNT,80 ,,                DUMPED
       BNLC*-24   ,,,                    CARDS
*****************************************************************
       BNI  *+44,00600,,    IF NO READ ERRORS, PROCESS.
       RCTY,,,              A READ ERROR HAS OCCURED.
       WATYRDERR,,,         NOTE THIS TO OPERATOR.
       B    HOP,,,          TRY AGAIN.
       DORG*-3
       BTM  BEGEND,0,10,,   FIND BEGINNING AND END OF DATA.
*
*
*          SORT CHANNELS, FIND TIME, OUTPUT TIME AND DATA
*              ACCORDING TO THIS FORMAT----------
*                                   FORMAT(8(I5,I3),3X,I5,I4,I1,I3)
*                            8(TIME, DATA POINT), RUN NO., TAPE FOOTAGE,
*                                 CHANNEL NO., AND SEQUENCE NO.
*                             BOTH TIME AND IT'S DATA POINT ARE OUTPUT
*                             TOGETHER BECAUSE OF THE GENERALITY OF
*                                    COMMUTATION SEQUENCING.
*
       TF   SEQNO-3,NUMBLK,,CLEAR LAST 53 COL. OF DATA CARD IN CASE
*                            THERE ARE NOT 20 SAMPLES.
*
       TFM  SEQNO,1,9,,     SET FIRST SEQUENCE NO. = 1
*           DECODE CHANNEL INFORMATION FOR SEPARATION.
*              CHECK IF ONLY ONE CHANNEL USED (NON-COMMUTATED)
       TFM  COUNT,CHANNL-8,, LOOK AT SECOND CHANNEL LOCATION.
       TDM  ONECHN,0,,      ASSUME MORE THAN ONE CHANNEL USED.
       BNR  *+24,CHANNL-8,,, IF NO RMK IN CHAN. IND. 2, THEN MORE THAN
*                                ONE CHANNEL USED.
       TDM  ONECHN,1,,      ONLY ONE USED.
*           ONECHN=1 IF ONLY ONE CHANNEL USED.....
*              FIND ALL DATA FOR CHANNELS 0-9
       TFM  CC,0,10,,       SET UP CHANNEL NO. 0
```

```
      NEXTCHTFM COUNT,CHANNL-9,, SET A COUNTER WITH FIRST DIGIT OF CHANNL
            TFM TEMP,TCHAN,,   SET A COUNTER WITH A CHANNEL MASK AREA.
            TDM NOCHAN,0,,     ASSUME NO CHANNEL OF THIS NO. USED
            TF  TCHAN,ZER10,,  INIT. MASK TO RMK'S TO BE REMOVED
      *                                  IF CHANNEL IS PROCESSED.
      CHANBRBNR COMPR,COUNT,11,, DO NOT EVEN CONSIDER THIS CHANNL IND.
      *                               IF THERE IS A RMK (CHANNEL NOT USED)
            AM  COUNT,1,,      INCR. CHANNL COUNTER
            AM  TEMP,1,,       INCR. MASK COUNTER
            CM  COUNT,CHANNL+1,, CHECK IF ALL IND. MASKED.
            BNE CHANBR,,,
            BD  PUNCH,NOCHAN,, GO TO PUNCH ROUTINE IF A CHANNEL HAS BEEN
            B   INCRCC,,,,                                        FOUND.
            DORG*-3
      COMPR TD  CCM,COUNT,11,, BRING CHANNL DIGIT TO COMPARE FIELD
            TDM CCM-1,0,11,,   FILL OTHER COMPARE FIELD DIGIT
            C   CC,CCM,,       COMPARE CHANNEL NO. AND IND.
            BNE CHANBR+12,,,   NO MATCH
            TDM TEMP,1,6,,     SET MASK TO MATCH INDICATOR
            TDM NOCHAN,1,,     REMOVE NO-CHANNEL IND.
            B   CHANBR+12,,,   CONTINUE SEARCH
            DORG*-3
      PUNCH NOP
            TFM LOOPCD+11,INPUT,,,   INIT. INPUT STRING COUNTER.
            BTM TIMER,TIME-5,,,STORE TIME OF FIRST SAMPLE.
      *                           GET TIME OF NEXT CARD INTO A TEMP. TIME
      *                             TO COMPUTE DELTA TIME.......
            A   BEGIN,CDL,,,   SET ADDRESS OF NEXT CARD
            BTM TIMER,TTIME-5,,GET TEMP. TIME
            S   BEGIN,CDL,,,   BACK TO ADDRESS OF LAST SAMPLE.
            SF  TTIME-5,,,     SET FIELD LIMITS
      SFTEM SF  TIME-5,,,      SET FIELD LIMITS
            S   TTIME,TIME,,   DELTA TIME /CARD
            TF  SFTEM+11,99,,  STORE CARD LENGTH
            LD  98,TTIME,,     SET DIVIDEND TIME
            D   93,SPERC,,     DIVISOR IS SAMPLES PER CARD.
            SF  91,,,
            TF  TTIME,96,,     STORE DELTA TIME FOR THIS CARD.
      *
      *       FORTRAN VERSION OF THIS SECTION.
      *
     *0606*
     C      *   READ S CORE DUMP INTO CORE AND STORES IN ARRAY FOR OUPT
     C      *   START WITH 000XXXT ON FIRST CARD
            *   DIMENSION IN(240),IA(40),ITEMP(6)
            * IF(SENSE SWITCH 9)99,99
          99*   CONTINUE
            *  READ 1,I1,I2,I3
            * PRINT 98,I1,I2,I3
          98*  FORMAT(1H1,36H   CORE DUMP PROCESSOR FOR CHANNELS  ,I2,1H,,I2,
          98* 1 1H,,I2/)
           7* CONTINUE
            * READ 1,IN
           1* FORMAT(80I1)
     C      * PULL TIME
            * J=0
            * DO 2 I=7,37,6
            * J=J+1
           2* ITEMP(J)=IN(I)
            * ITIME=0
```

```
 * IT=100000
 * DO 3 I=1,6
 * ITIME=IT*ITEMP(I)+ITIME
 * IT=IT/10
3* CONTINUE
 * J=1
 * DO 4 I=4,238,6
 * IA(J)=100*IN(I)+10*IN(I+1)+IN(I+2)
 * J=J+1
4* CONTINUE
 * PUNCH 5,I1,I2,I3,ITIME,IA
 * PRINT 5,I1,I2,I3,ITIME,IA
5* FORMAT(1X,3I2,5X,I6/20I4/20I4)
 * IF( SENSE SWITCH 9)99,7
 * END
 *
       BD   *+20,ONECHN,,  PROCESS ONE CHANNEL ONLY.
       B    MULCHN,,,      PROCESS MULTIPLE CHANNELS.
       DORG*-3
 *
 *                         ONE CHANNEL PROCESSOR.................
 *                         OUTPUT PHASE 1 - 1X,3I2,5X,TIME/40 SAMPLES.
 *
ONECH  TF   CARD+79,NUMBLK-18,,,  BLANK OUT CARD AREA.
       TF   CARD+48,NUMBLK,,,
       TDM  CARD+2,0,10,,  STORE CHANNEL NO. AS ZERO.
       TDM  CARD+4,0,10,,  STORE CHANNEL NO. AS ZERO.
       TDM  CARD+6,0,10,,  STORE CHANNEL NO. AS ZERO.
CCNT20DS   2,*-2
       TF   CARD+17,TIME,, STORE TIME IN CARD...
       CF   CARD+12,,,     CLEAR FLAGS FOR NUMAERIC OUTPUT.
CNT3   DS   2,*
       BTM  OUTPUT,,,,     OUTPUT CHAN. NO. AND TIME.
       TF   CARD+48,NUMBLK,,,  CLEAR OUT OLD TIME.
       TDM  CCNT20,0,,,    SET CARD COUNTER FOR FIRST CARD.
NEWCD  TFM  LOOPCD+6,CARD  ,,  START OUT ON BEGINNING OF CARD.
NEWDATAM   LOOPCD+6,1,10,, INIT. AND FILL TO I4 FORMAT
       TFM  CNT3,3,10,,    COUNTER FOR 3-DIGIT REMOVAL.
       AM   LOOPCD+11,3,10,, GO FOR NEXT SAMPLE.
LOOPCDTD   ,,,,            P AND Q FILLED BY PREVIOUS INST.
       BNR  *+24,LOOPCD+6,11,,  CHECK FOR 'RMK' IN DATA.
       TDM  LOOPCD+6,0,6,, REPLACE RMK WITH ZERO.
       AM   LOOPCD+11,1,,
       AM   LOOPCD+6,1,,,
       SM   CNT3,1,10,,
       BNZ  LOOPCD,,,      3 DIGITS NOT PULLED YET.
       CM   LOOPCD+6,CARD+79,,  CHECK END OF CARD.
       BNN  PCH1,,,        PUNCH THE CARD.
       B    NEWDAT,,,      FILL REST OF CARD.
       DORG*-3
PCH1   BTM  OUTPUT,,,,     OUTPUT CARD...
       C    LOOPCD+11,END,, CHECK END OF DATA SET.
       BNN  TYPEHD,,,      READY FOR NEW CONVERSIONS.
       BD   PUNCH1,CCNT20,, SEE IF FIRST OR SECOND CARD.
       TDM  CCNT20,1,,     SET SECOND CARD IND.
       B    NEWCD,,,
       DORG*-3
PUNCH1AM   BEGIN,240,9,,  UPDATE TIME LOCATION.
       BNC3PCHSS4,,,, CHECK S.S. 3 TO SEE IF RUN ABORT.
       RCTY
```

```
          WATYRABORT,,, NOTIFY OF RUN ABORT.
          TF   NXTSEC,OLDSEC,,, MODIFY DISK ADD. FOR NEW RUN.
          B    TYPEHD,,,, RESTART.
          DORG*-3
      PCHSS4BNC4PUNCH+24,,, FIND NEW TIME AND PUNCH IF NO ABORT.
          RCTY
          WATYPABORT,,,, LABEL TYPED PAGE WITH NOTE.
          B    TYPEHD
          DORG*-3
          DS   5
      OUTPUTNOP
          TF   SECTOR,NXTSEC,,, MOVE SECTOR ADDRESS TO DCF.
          SM   NXTSEC,1,10,,, POSITION NEXT SECTOR ADD. TO CORRECT SECTOR.
      **********                    420 CARDS PER MINUTE EQUIV. PUNCHING.
      DISK   WN   DCF,00702,,, TRY TO WRITE ON DISK.
          BNI  OKAY,03900,,, NO DISK CHECKS.
          BNI  DERR,03600,,, TRUE DISK ERROR.
          K    DCF,00701,,, SEEK CYLINDER.
          B    DISK,,,, TRY AGAIN.
          DORG*-3
      OKAY   NOP
          CM   NXTSEC,201,,,CHECK THAT CYL. 0 NOT VIOLATED.
          BNP  DERR,,,,,  ERROR , TOO MUCH WRITTEN ON DISK.
          BB   ,,,, GO BACK TO CALLING ROUTINE.
          DORG*-9
      DERR   RCTY
          WATYDERMES,,, NOTIFY OF DISK TROUBLE.
          H
          B    *-12
          DORG*-3
      MULCHNB    ONECH
*****         RCTY
*****         WATYNOMULT,,,,, NOTE THAT MULTIPLE CHANNEL OPERATION NOT IN.
*****         B    MULCHN
*****NOMULTDAC 44,MULTIPLE CHANNEL OPERATION NOT IMPLEMENTED.',,
      *                              ALL OF THIS CHANNEL HAS BEEN PROCESSED,
      *                               INCREMENT AND CHECK FOR LAST CHANNEL.
      INCRCCAM   CC,1,10,,       INCREASE CHANNEL NO.
          CM   CC,9,10,,        CHECK LAST CHANNEL NO.
          BNP  NEXTCH,,,
          H
      *                   CONSOLE DATA INPUT SUBROUTINE
      *                   -----------------------------
          DORG*+5
      TYPEINTR   TEMPIN,FLGZER+39,,INITIALIZE A TEMPORARY INPUT AREA.
          RNTYTEMPIN+1,,,     IMBED INPUT IN AN AREA OF FLAGGED ZEROS.
          BC4  *-24,,,        GOOF SWITCH
          BNR  ERRIN,TEMPIN+11,,IF NO RMK AT END OF TEMP, TOO MUCH TYPED
          TFM  COUNT,TEMPIN+10,,SET UP TRANSFER COUNTER.
      BNFSC  BNF  STRIN,COUNT,11,THIS LOC. HAS NOT BEEN TYPED IN.
          B    STRIN+24,,,    DO NOT STORE ANY INPUT DATA.
          DORG*-3
      STRIN  TD   TYPEIN-1,COUNT,611,, STORE A DIGIT.
          SM   TYPEIN-1,1,10, BUMP RECEIVE AREA COUNTER DOWN BY 1
          SM   COUNT,1,10,    BUMP INPUT AREA COUNTER DOWN BY 1
          CM   COUNT,TEMPIN,, AT BEGINNING OF INPUT AREA YET
          BNE  BNFSC,,,       IF NOT, CHECK NEXT LOWER LOC.
          TBTY,,,             TAB TYPEWRITER FOR NEXT INPUT DATA.
          BB   ,,,            RETURN TO CALLING PROGRAM.
          DORG*-8
```

```
ERRIN RCTY,,,              TOO MUCH INPUT TYPED........
      WATYERRIN1,,,        TELL OPERATOR.
      B   INIOUT,,,        GO BACK TO BEGINNING OF DATA INPUT.
      DORG*-3
ERRIN1DAC 37,YOU HAVE TYPED TOO MUCH, START OVER.',,
*
*            SUBROUTINE TO FIND BEGINNING AND END OF SAMPLED DATA.
*
      DORG*+2
BEGENDTFM COUNT,INPUT+3,,SET INPUT AREA COUNTER TO FIRST SAMPLE.
      MM  SPERC,6,10,,     FIND HOW FAR AWAY THE NEXT CARD BEGINS.
      SF  97,,,            DEFINE 3-DIGIT FIELD
      TF  CDL,99,,,        STORE CARD LENGTH
*
*          CHECK THIS CARD FOR A RMK IN HUNDREDS POS. OF EACH SAMPLE.
*
      TF  TEMP,COUNT,,     TRANSFER CARD BEGIN COUNT TO TEMP. COUNTER.
      A   COUNT,99,,       INCR. CARD BEGIN COUNTER BY SAM/CARD * 6
BNRIT BNR INCR,TEMP,11,,  IF NO RMK, INCREASE THIS CARD COUNTER.
      B   *-36,,,          THERE IS A RMK, FIND NEXT CARD.
      DORG*-3
INCR  AM  TEMP,6,10,,      FIND NEXT SAMPLE ON THIS CARD.
      C   TEMP,COUNT,,     AT END OF THIS CARD YET
      BNE BNRIT,,,         IF NOT, CHECK NEXT SAMPLE FOR RMK.
      S   COUNT,99,,       THIS CARD HAS NO RMK'S.
      TF  BEGIN,COUNT,,    SET BEGINNING ADDRESS FOR PROCESSING.
*
*            SEARCH FOR END OF CONVERSION.(FLAGS PRESENT).
*
      A   COUNT,99,,       LOOK ONLY AT FIRST SAMPLE.
      BNF *-12,COUNT,11,   IF NO FLAGS, INCREASE CARD COUNTER.
      TF  LASTAD,COUNT,,   SET END OF MEMORY INTIALIZATION.
      S   COUNT,99,,       A FLAG HAS OCCURRED, DO NOT PROCESS
*                            THE LAST CARD OF PARTIAL RESULTS.
      TF  END,COUNT,,,     SET END OF PROCESSING ADDRESS.
      BB  ,,,              RETURN TO CALLING PROGRAM.
      DORG*-8
*
*                              SUBROUTINE TO PULL TIME
      DORG*+5
TIMER TF  COUNT,BEGIN,,    SET UP FIRST SAMPLE COUNTER
      AM  COUNT,3,10,      INCR. TO FIRST TIME LOC.
TFMST8TFM *+8,7,10,,       SET NO. OF TIMES THROUGH LOOP COUNTER
      TF  TEMP,TIMER-1,,   LOAD TIME AREA COUNTER
      TD  TEMP,COUNT,611,  STORE TIME DIGIT
      AM  COUNT,6,,        INCR. DATA COUNTER TO NEXT TIME DIGIT
      AM  TEMP,1,,         INCR. TIME AREA COUNTER
      SM  TFMST8+8,1,10,   DECR. LOOP COUNTER
      BNZ *-48,,,          NOT DONE
      BB  ,,,              RETURN TO CALLING PROGRAM
      DORG*-8
OPINSTDAC 36,MAKE SURE A-D CONVERTOR IS STOPPED.',,
PABORTDAC 18,PUNCHING ABORTED.',,
RABORTDAC 30,RUN ABORTED, NO CARDS STORED.',,
RDERR DAC 43,READ PARITY ERROR IN CONVERSION,TRY AGAIN.',,
DERMESDAC 40,DISK OVERWRITE OR FATAL HARDWARE ERROR.',,,
LABEL DAC 6,LABEL',,
RMK   DC  2,0',,
MEMCAPDS  5
LASTADDS  5
```

```
COUNT DS   5
TEMP  DS   5
CDL   DS   3
CARD1 DNB  2
DATE  DC   6,0
      DNB  5
RUN   DC   5,0
      DNB  6
TPOS  DC   4,0
      DNB  5
CHANNLDC   10,0
      DNB  8
SPERC DC   2,0
      DNB  8
CPERS DC   2,0
      DNB  17
TEMPINDSS  12
ONECHNDS   1
CC    DS   2
CCM   DS   2
NOCHANDS   1
TTIME DS   6
TCHAN DS   10
FLGZERO    0,0,01234567891011
      0    0,0,01234567891011
      0    0,0,01234567891011
      0    0,0,01234567891011
      DC   2,-0
      DC   1,',
FLGRMKDSC  1,',
      DSC  1,',
      DSC  1,',
      DSC  1,',
      DSC  1,',
      DSC  1,',
      DSC  1,',
      DSC  1,',
      DSC  1,',
ZER10 DSC  1,',
BEGIN DS   5
END   DS   5
CARDN DNB  1
TIME  DC   6,0
DATA1 DS   1
      DS   69
SEQNO DS   3
NUMBLKDNB  50
      DS   1
      DAS  1
CARD  DS   1
      DS   79
      DSC  15,0
NXTSECDC    5,09999,,, SUPPLY SECTOR ADDRESS FOR NEXT WRITE.
OLDSECDC   5,09999,,,SAVE ADDRESS OF BEGINNING OF DATA SECTION.
********************  DISK CONTROL FIELD  ***********
      DAS  1
DCF   DC   1,1
SECTORDS   5
      DC   3,1
      DSA  CARD
```

```
**********                                    **********
      INPUT DAS 1
      *                    SUBROUTINE TO PRINT HEADER AND ONE-TIME INFORMATION.
            DORG*+5
      HEADERRCTY,,,              SET TABS
            WATYEOLOFF
            RCTY
            WATYABORT
            RCTY
            WATYMESLST
            RCTY
            WATYSETTAB,,,,       OPERATOR INST. TO CLEAR TABS.
            RCTY,,,
            H
            WATYBLK,,,           POSITION CARRIAGE EVERY 15 COL. FOR TABS...
            H
            WATYBLK,,,
            H
            WATYBLK,,,
            H
            WATYBLK,,,
            H
            WATYBLK,,,
            H
            RCTY
            WATYHDINS1,,,,       OPERATOR INSTRUCTIONS FOR DATE.
            TBTY
            BTM TYPEIN,DATE,,    ACCEPT DATE AS 6 DIGITS(MO. DAY YEAR)
            RCTY
            RCTY
            WATYHDINS2,,,,       OPERATOR INST. FOR RUN DATA.
            RCTY
            WATYHDINS3,,,,       CAUTIONS.
            RCTY
            WATYHDINS4
            RCTY
            WATYHD5,,,           TYPE RUN NO. LABEL
            TBTY
            WATYHD6,,,,          TYPE TAPE FOOTAGE LABEL
            TBTY
            WATYHD7,,,,          TYPE CHANNEL CONFIGURATION LABEL
            TBTY
            WATYHD8,,,,          TYPE SAMPLES/CARD LABEL
            TBTY
            WATYHD9,,,,          TYPE CARDS/SEC. LABEL
            B    HEADER-1,,6,,   RETURN TO CALLING PROGRAM.
            DORG*-3
      BLK   DAC 16,                  ',,
      EOLOFFDAC 47,TURN END-OF-LINE CHARACTER OFF FOR BURST MODE.',,
      ABORT DAC 45,SW. 4 ABORTS DISK WRITING, SW. 3 ABORTS RUN.',,,
      MESLSTDAC 46,FOR END OF RUN, TYPE A RECORD MARK FOR LABEL.',,
      SETTABDAC 49,CLEAR TYPEWRITER TABS, SET TABS WHEN TYP. STOPS.',,
      HDINS1DAC 39,TYPE DATE AS 6 DIGITS- MONTH DAY YEAR.',,
      HDINS2DAC 41,TYPE RUN DATA UNDER APPROPRIATE COLUMNS.',,
      HDINS3DAC 36,DO NOT SPACE OR TAB THE TYPEWRITER.',,
      HDINS4DAC 34,SPACING AND TABS ARE SET FOR YOU.',,
      HD5   DAC 11,RUN NUMBER',,
      HD6   DAC 13,TAPE FOOTAGE',,
      HD7   DAC 12,CHAN. CONF.',,
      HD8   DAC 13,SAMPLES/CARD',,
```

```
HD9    DAC 11,CARDS/SEC.',,
START  TR  39999,RMK-1,,,
       BLC *+12
       SF  FLGRMK,,,       SET A FLAG 10 RMK'S DOWN FROM ZER10.
       BNR *+32,0,,,       CHECK MEM. SIZE BY CONTENTS OF LOC. 0
       TFM MEMCAP,39999,,,40K CORE MEMORY SIZE
       B   *+20,,,
       DORG*-3
       TFM MEMCAP,59999,,,60K CORE MEMORY SIZE.
       B   TYPEHD,,,,      GO TO REAL START OF PROGRAM.
       DENDSTART
```

APPENDIX IV

A SAMPLE PROGRAM TO SHOW HOW DIFFERENT FUNCTIONS OF A DATA SYSTEM MAY BE SEPARATED. MINIMUM PROCESSING TIME RESULTS FROM EXPANSION OF LOOP STRUCTURES.

```
      DORG 402
SEEK  NOP
      SM   DCF+5,1,10
      K    DCF,00701,,,        SEEK TO FIND FIRST SECTOR.
      AM   DCF+5,1,10          ,,,,NOW ADDRESS OKAY.
START RNPT ADCIN,,, READ A-D CONVERTOR
      RNPT ADCIN,,, READ A-D CONVERTOR
      AM   RECNO,1,10,,, INCREASE RECORD NO.
      TF   TIME,RECNO,,,  STORE RECORD NO.
      TD   X-19,ADCIN+3,, TRANSFER   TENS   OF SAMPLE
      TD   X-18,ADCIN+9
      TD   X-17,ADCIN+15
      TD   X-16,ADCIN+21
      TD   X-15,ADCIN+27
      TD   X-14,ADCIN+33
      TD   X-13,ADCIN+39
      TD   X-12,ADCIN+45
      TD   X-11,ADCIN+51
      TD   X-10,ADCIN+57
      TD   X-09,ADCIN+63
      TD   X-08,ADCIN+69
      TD   X-07,ADCIN+75
      TD   X-06,ADCIN+81
      TD   X-05,ADCIN+87
      TD   X-04,ADCIN+93
      TD   X-03,ADCIN+99
      TD   X-02,ADCIN+105
      TD   X-01,ADCIN+111
      TD   X,ADCIN+117
      SF   X-19
      C    X,ZERO,,, CHECK IF RECORD NON-ZERO
      BE   START
      SM   DCF+5,1,10          ,,,, CORRECT SECTOR ADDRESS.
      BNC4*+24
      TDM TIME-4,,,,           STORE RMK IN TIME FOR END OF SCAN.
      DC   1,',*,,
YTART RNPT ADCIN,,, READ A-D CONVERTOR
      RNPT ADCIN,,, READ A-D CONVERTOR
      AM   RECNO,1,10,,, INCREASE RECORD NO.
      TF   YIME,RECNO,,,  STORE RECORD NO.
      TD   Y-19,ADCIN+3,, TRANSFER   TENS   OF SAMPLE
      TD   Y-18,ADCIN+9
      TD   Y-17,ADCIN+15
      TD   Y-16,ADCIN+21
      TD   Y-15,ADCIN+27
      TD   Y-14,ADCIN+33
      TD   Y-13,ADCIN+39
      TD   Y-12,ADCIN+45
      TD   Y-11,ADCIN+51
      TD   Y-10,ADCIN+57
      TD   Y-09,ADCIN+63
      TD   Y-08,ADCIN+69
      TD   Y-07,ADCIN+75
      TD   Y-06,ADCIN+81
```

```
        TD   Y-05,ADCIN+87
        TD   Y-04,ADCIN+93
        TD   Y-03,ADCIN+99
        TD   Y-02,ADCIN+105
        TD   Y-01,ADCIN+111
        TD   Y,ADCIN+117
        SF   Y-19
        C    Y,ZERO,,, CHECK IF RECORD NON-ZERO
        BE   YTART
*                                CHECK FOR SEEK TO OVERLAP OPERATIONS.
        C    DCF+3,SEEKAD
        BNN  *+36
        K    DCF,00701
        SM   SEEKAD,2,10
*                                ARMS SHOULD BE IN CORRECT CYL.
ZTART   RNPT ADCIN,,, READ A-D CONVERTOR
        RNPT ADCIN,,, READ A-D CONVERTOR
        AM   RECNO,1,10,,, INCREASE RECORD NO.
        TF   ZIME,RECNO,,,  STORE RECORD NO.
        TD   Z-19,ADCIN+3,, TRANSFER   TENS   OF SAMPLE
        TD   Z-18,ADCIN+9
        TD   Z-17,ADCIN+15
        TD   Z-16,ADCIN+21
        TD   Z-15,ADCIN+27
        TD   Z-14,ADCIN+33
        TD   Z-13,ADCIN+39
        TD   Z-12,ADCIN+45
        TD   Z-11,ADCIN+51
        TD   Z-10,ADCIN+57
        TD   Z-09,ADCIN+63
        TD   Z-08,ADCIN+69
        TD   Z-07,ADCIN+75
        TD   Z-06,ADCIN+81
        TD   Z-05,ADCIN+87
        TD   Z-04,ADCIN+93
        TD   Z-03,ADCIN+99
        TD   Z-02,ADCIN+105
        TD   Z-01,ADCIN+111
        TD   Z,ADCIN+117
        SF   Z-19
        C    Z,ZERO,,, CHECK IF RECORD NON-ZERO
        BE   ZTART
*                                CHECK FOR WRITE ON CYL. 0.
        CM   DCF+3,2,9
        BP   WTART
        RCTY
        WATYDONE
        H
        B    *-36
WTART   RNPT ADCIN,,, READ A-D CONVERTOR
        RNPT ADCIN,,, READ A-D CONVERTOR
        AM   RECNO,1,10,,, INCREASE RECORD NO.
        TF   WIME,RECNO,,,  STORE RECORD NO.
        TD   W-19,ADCIN+3,, TRANSFER   TENS   OF SAMPLE
        TD   W-18,ADCIN+9
        TD   W-17,ADCIN+15
        TD   W-16,ADCIN+21
        TD   W-15,ADCIN+27
        TD   W-14,ADCIN+33
        TD   W-13,ADCIN+39
```

```
        TD   W-12,ADCIN+45
        TD   W-11,ADCIN+51
        TD   W-10,ADCIN+57
        TD   W-09,ADCIN+63
        TD   W-08,ADCIN+69
        TD   W-07,ADCIN+75
        TD   W-06,ADCIN+81
        TD   W-05,ADCIN+87
        TD   W-04,ADCIN+93
        TD   W-03,ADCIN+99
        TD   W-02,ADCIN+105
        TD   W-01,ADCIN+111
        TD   W,ADCIN+117
        SF   W-19
        C    W,ZERO,,, CHECK IF RECORD NON-ZERO
        BE   WTART
DISK    WN   DCF,00702
        B    START
DERR    RCTY
        WATYDERROR
        H
        B    *-12
RECNO   DC   5,0
DERRORDAC  12,DISK ERROR.',
SEEKADDC   3,098
DONE    DAC  11,DISK FULL.',,
ZERO    DC   20,0
        DAS  1
TIME    DC   5,0
X       DS   20
YIME    DC   5,0
Y       DS   20
ZIME    DC   5,0
Z       DS   20
WIME    DC   5,0
W       DS   20
        DC   1,'
        DS   2
*********    DISK CONTROL FIELD..**********
        DAS  1
DCF     DC   1,1
*                                DISK ADD. 1 HIGHER THAN FIRST ACTUAL SECTOR
        DC   5,10000
        DC   3,1
        DSA  TIME-4
*************************************
ADCIN   DS   120
        DENDSEEK
```

A Data Processing System for a Regional Group of Seismic Stations.

The primary object of recording seismic waves at a network of stations is to determine the locations of the points at which the rock fractures occur; that is the hypocentres of the earthquakes. This is made possible by observing the arrival times of the elastic waves that originate at the hypocentre at several points on the surface of the Earth. Although there are many different types of wave originating from an earthquake, the two principal body waves are the ones used for hypocentre location. The first of these, the P-wave, is a compression wave whose velocity is known in terms of the density and elastic constants of the rocks through which it travels. This is the highest velocity body wave and it is the first arrival at any station. The S-wave or shear wave has a velocity that is also known in terms of the density and elastic constants but which is less than the P velocity by a factor which is nearly a constant for most rocks. Since the density and elastic constants of the material of the Earth vary with depth, the two velocities also vary and the paths of the waves are not straight lines. Thus, for accurate travel times, a table of time against surface distance to the epicentre and depth must be consulted even though an approximate value may be found by assuming constant velocities. The epicentre is the surface point immediately above the hypocentre.

The seismograph is an instrument which gives a continuous record of ground movement and it is from this record that the times of arrival of the seismic waves are found. The P- arrival will normally be measurable to a greater accuracy than the S- arrival as it occurs when the ground is

nearly stationary whereas the S- arrival may be obscured by the tail of the P-wave. For this reason, although the S-waves are used in the preliminary distance calculations, final hypocentre coordinates are based as far as possible on P-waves only. However, a second order equation gives a fairly accurate distance estimate in terms of the P-S interval.

If the P-S interval is measured at at least one station, the distance from that station, and hence the origin time, can be estimated. If the P- arrival time is known at at least two more stations the distance from these two stations can be estimated from the origin time and hence this represents a minimum amount of information upon which to base a hypocentre determination. In theory the P- arrival times at four stations would be sufficient information for a hypocentre determination, but the technique of the preliminary determination of the position presents a great many problems and since it is a very rare occurrence to observe P without S at as many as four stations, it is not a serious matter to ignore the possibility of the alternative approach.

When the position of the hypocentre of an earthquake has been established, measurements of the amplitude and frequency of the ground movement at known distances from it enable the amount of energy released to be calculated. The logarithm of the amount of energy is a measure of the magnitude of the earthquake. The distance and azimuth of the event from each of the observing stations can also be calculated. All this information constitutes the routine output of a regional seismic research organisation. The calculations represent a considerable amount of labour and automation of the system is clearly to be desired.

The regional group of stations for which the data processing system being discussed was devised is that of the Caribbean, with the main station in Trinidad. The stations which form the group, including those which contribute observations but which are not directly part of the group, form a chain through the islands of the Lesser and Greater Antilles together with one on the mainland of South America in Caracas. The arrangement of the stations presents some problems that would not apply to more satisfactory arrangements, but this is dictated by the absence of land in many of the desirable locations. The computer available is a basic card I/O 1620 with 20K storage and an off line card-sorter. The programmes were written in S.P.S. for this machine and had to be segmented in order that core storage would be sufficient. This is not a tremendous disadvantage. The discussion will be as far as possible independent of the machine configuration or programming language.

As is usually desirable, the data read from the seismograms is punched into cards at the earliest opportuniy. This is in the form of the actual length measurements on the seismograms without any preliminary calculation. These measurements are made on the relative positions of radio controlled time marks on the trace and the local clock marks, and they are made on the relative position with respedt to local clock marks, the amplitude and the frequency of seismic waves arriving at the station. Two types of card are punched. One enables a continuous series of clock corrections to be made and the other enables arrival times etc. for each phase arrival to be calculated. The overall processing system is illustrated in the flow-chart of figure 1.

The first part of the programme performs a very small amount of calculation, but the amount of data is quite considerable. A check is made for apparent sudden changes in the local clock rate at each station and for discrepancies which might result from the misidentification of a clock mark on the trace. The output is a set of preliminary phase cards for each station. These contain station identification, date and time of arrival, phase identification, ground amplitude, wave frequency, seismograph type and some other information which is not relevant to the discussion.

Mechanical sorting enables all the preliminary phase cards of all the stations to be merged in chronological sequence. Although a larger computer would probably enable this sorting process to be combined with the first part of the programme, this is not entirely desirable since the data from the several stations for the same period may not all be available at the same time.

The second part of the programme represents the most important section of the system. In it the phase cards are examined to find any grouping of observations that may have arisen from one event, and when these groups are found, the hypocentre position is calculated. The output is a set of hypocentre cards giving origin time, latitude, longitude and depth. The selection of phase cards to form groups must necessarily be done using arbitrary criteria and it is possible that this may result in some wrong grouping. Checking at later stages will generally eliminate errors from this cause. The arbitrary grouping is done on the basis that the first and last arrival time observed by any stations for the same event cannot be separated by lo-ger than the travel time between the two

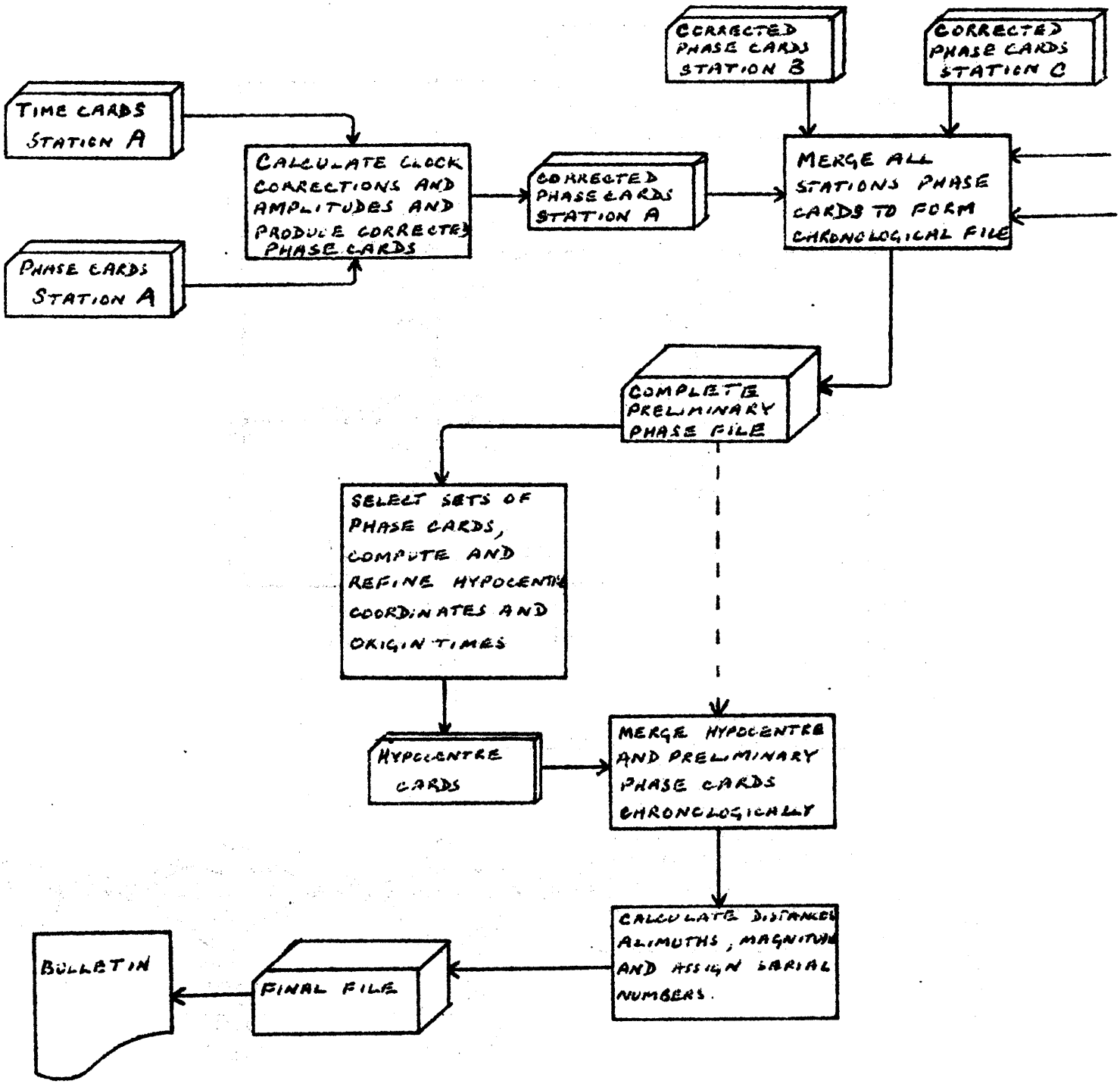
most distant stations, and also that the P- and S-waves observed at one station cannot be separated by more than a time corresponding to the limit of the region of interest. If a group is found for which at least 3 P-arrivals have been recorded and where there is at least one S- arrival, an attempt is made to compute a hypocentre. The flow-chart of this grouping process is illustrated in the first part of figure 2.

The most difficult part of the process of determining the position of a hypocentre is the finding of a plausible place from which to start. If a good guess is made of this position, the refinement process is almost routine. In large networks such as the worldwide systems, or in networks where the station locations form a desirable grid, this is not as serious a problem as it is in the case being considered. It usually is sufficient to assume that the hypocentre is very close to the station at which the first arrival was recorded and refine from there. But when the stations are constrained by geographical considerations to being arranged in a chain, especially when the events being observed are also near to that chain, a wrong choice of the starting position may easily lead to a false final hypocentre. The criterion of refinement of the hypocentre position is that the r.m.s. deviation of the individual station estimates of the origin time from the mean should be a minimum. It is not difficult to see that a hypocentre that is really a little to one side of the chain will produce an image minimum on the other side. The second part of figure 2 illustrates the programme that has been developed to find the best start point by an empirically satisfactory technique, and figure 3 interprets this technique graphically.

When the position of the hypocentre for which the r.m.s. residual

has been found it is tested to see whether the minimum value is acceptable. An r.m.s. residual of 0.5 seconds is hoped for and if this found it is concluded that a false grouping of phase arrivals has not occurred. If the r.m.s.residual is not as low as this, the residual of the worst station is examined to see whether this observation can reasonably be concluded to be unrelated to the set. If so, the refining process is continued with this station deleted, provided that it is possible to do so. It is not possible to continue refining if less than three stations are left. When no further improvement can be made to the r.m.s.residual, the hypocentre card is punched. It may happen that this is a false determination and a check is made for plausibility of all hypocentres found. So far no spurious one has turned up.

The final phase of the programme computes distances, azimuths, residuals and magnitudes and is more or less routine.



System Flow-Chart. Figure 1.

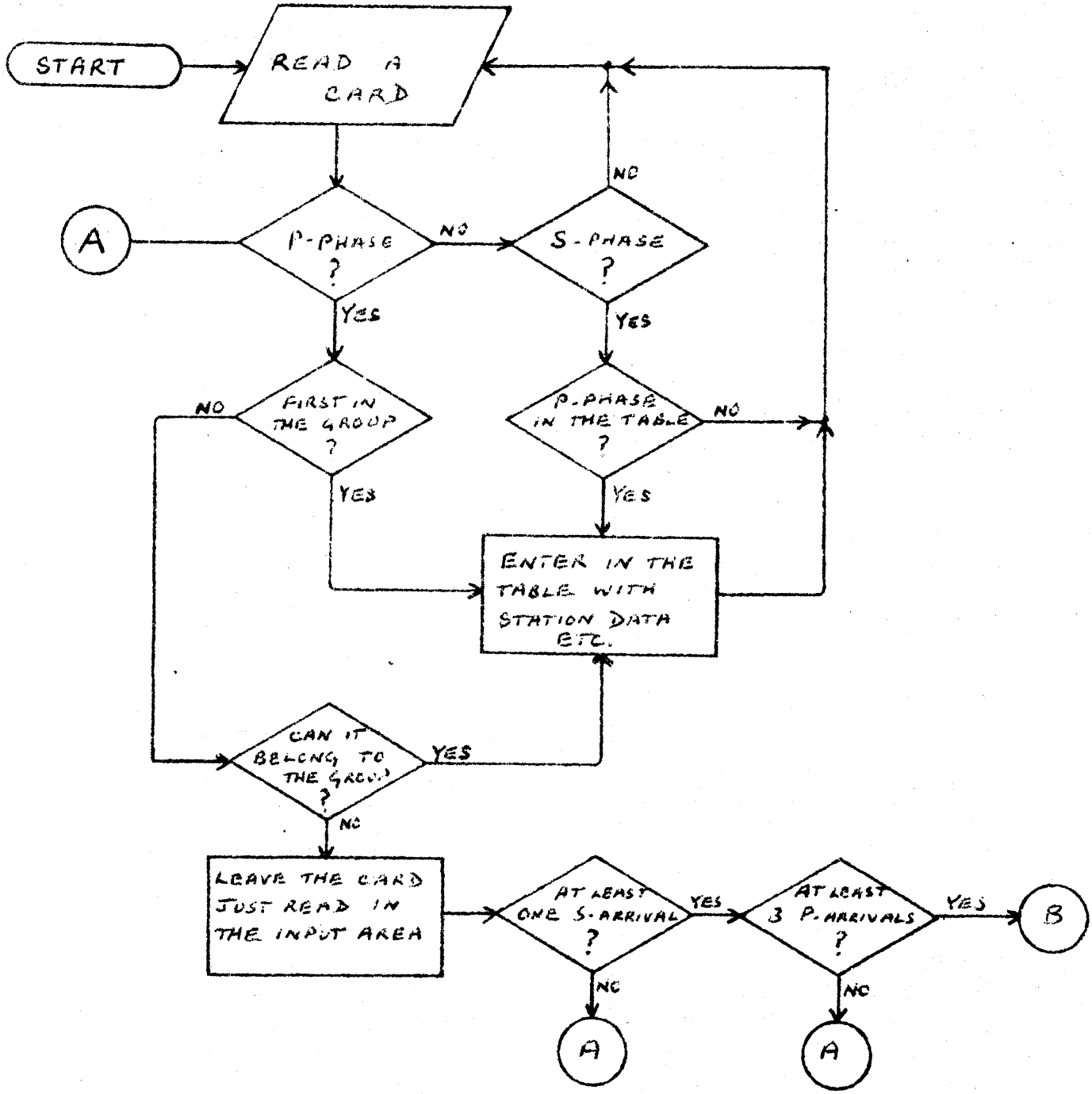
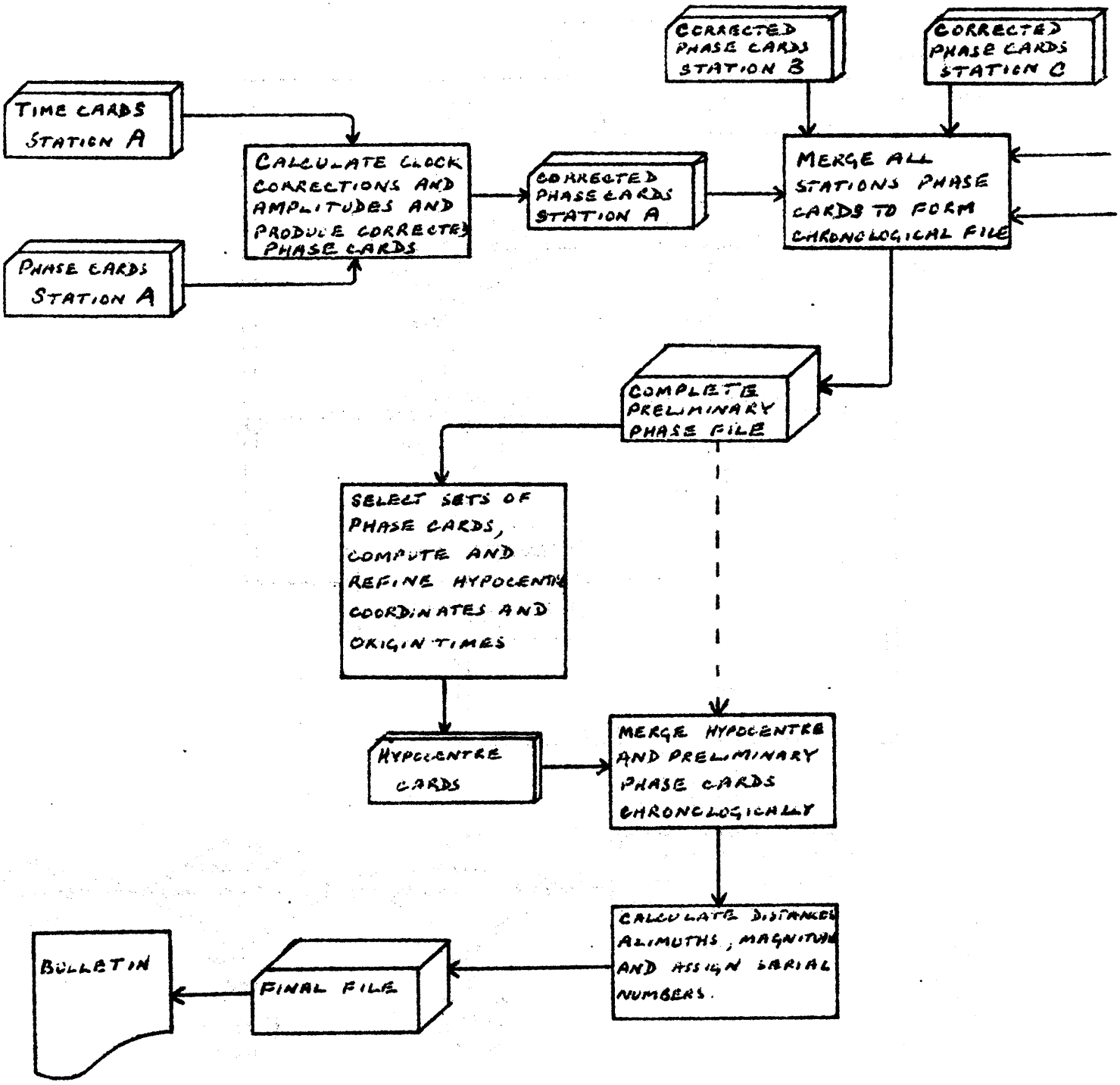


Figure 2, part A.



System Flow-Chart. Figure 1.

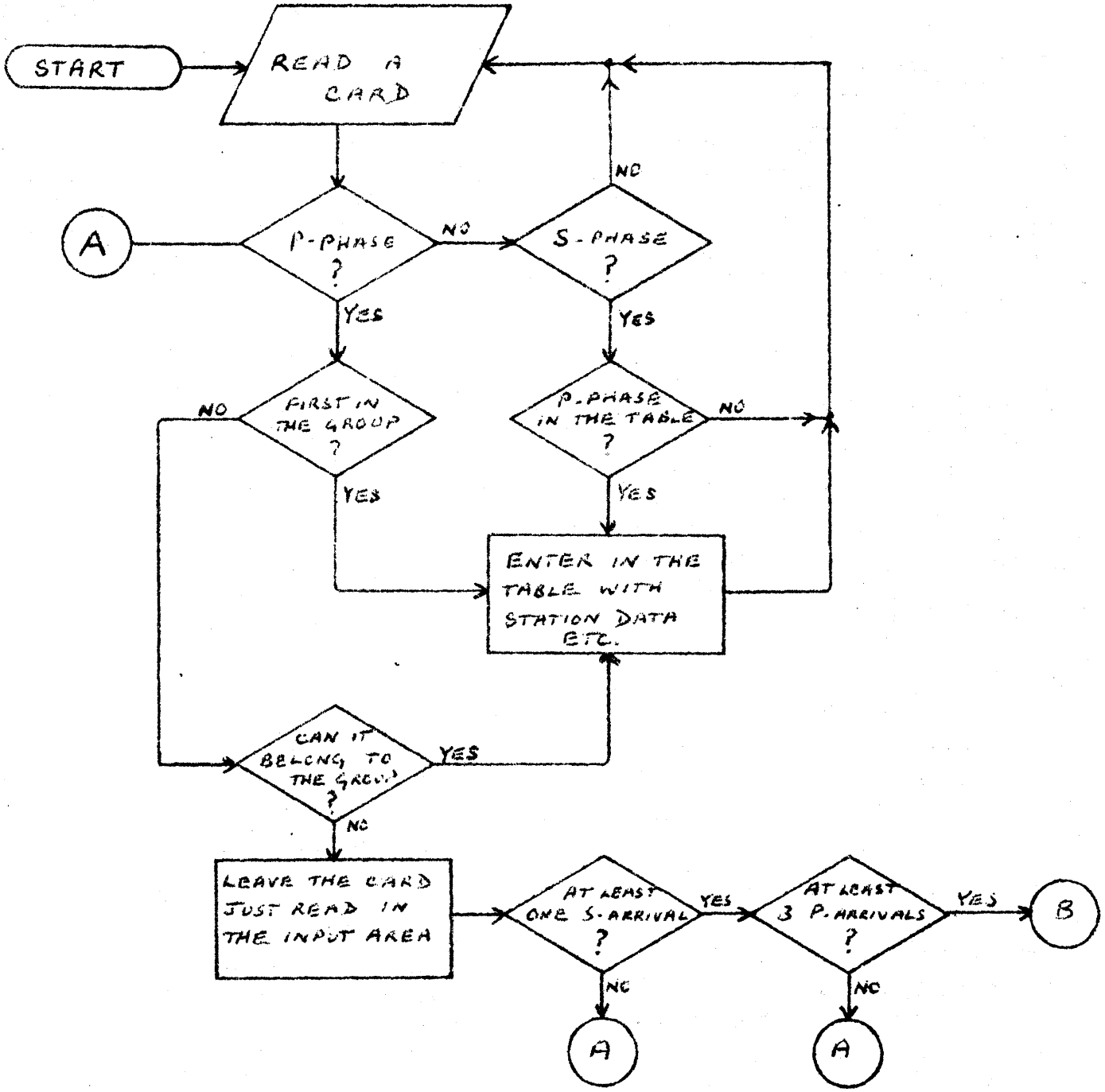


Figure 2, part A.

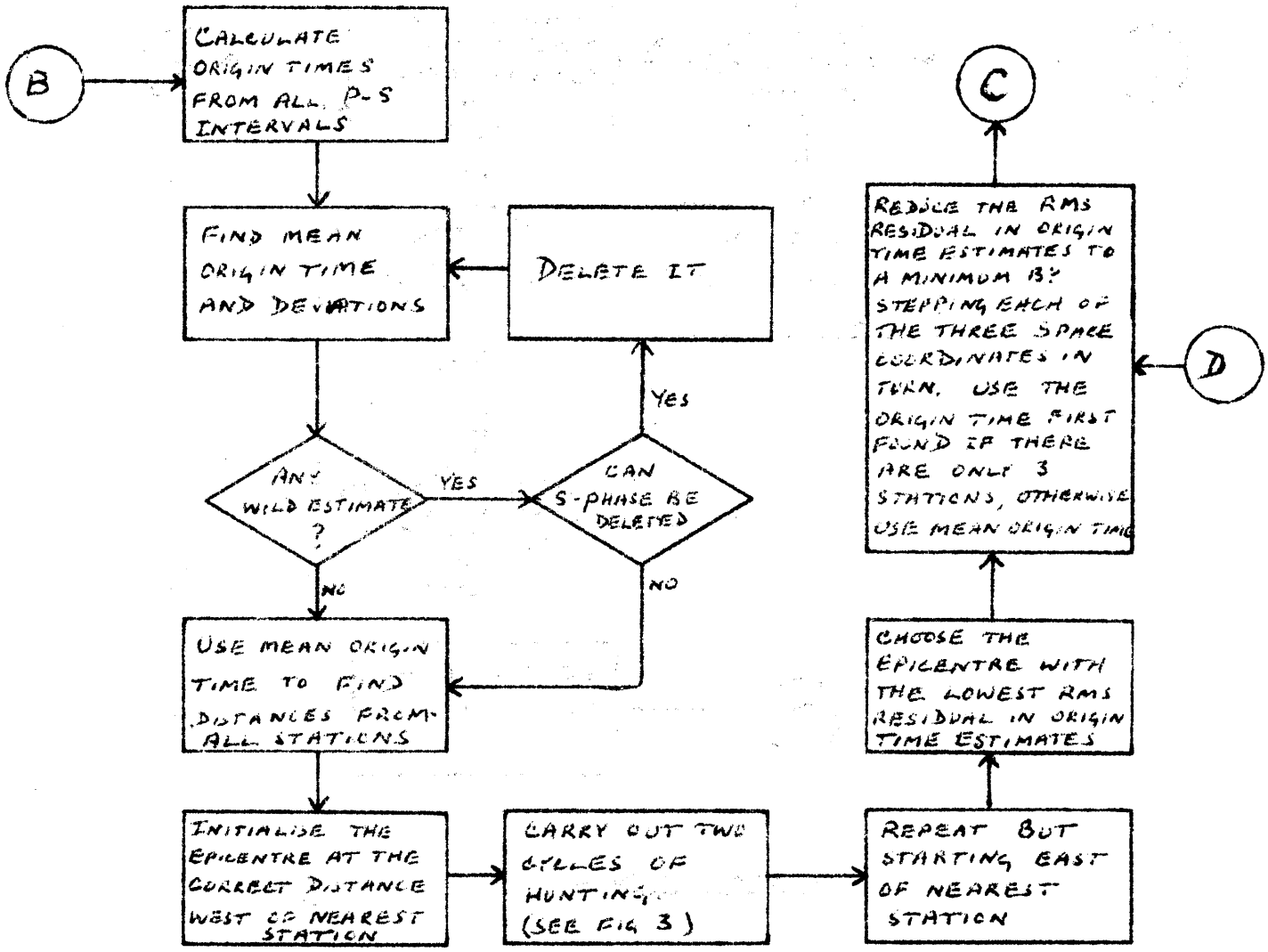


Figure 2, Part B.

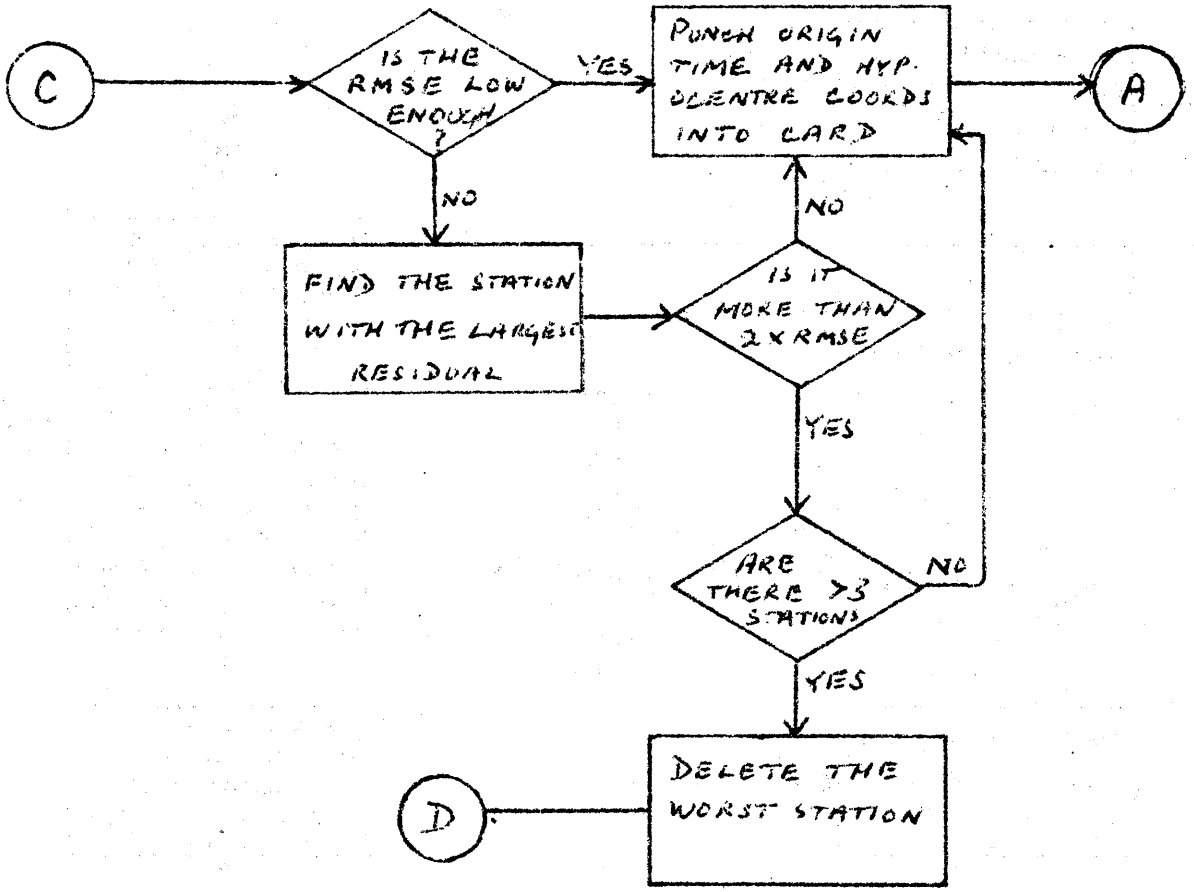


Figure 2, part C.

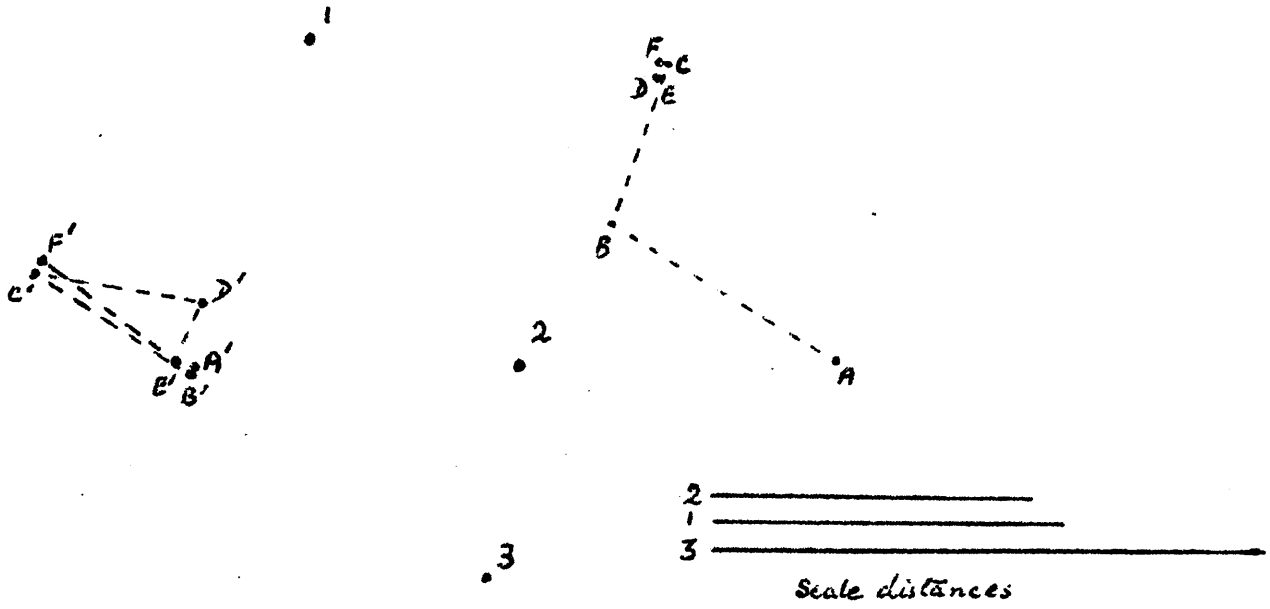


Figure 3.

The points 1,2 and 3 represent the positions of the observing stations. Preliminary calculations of distances are represented to scale at the bottom right. The points A and A' are obtained by measuring the appropriate distances to the East and to the West of the nearest station respectively. The point B (B') is obtained by measuring the appropriate distance along the line 1,A (1,A') from the point 1. The point C (C') is obtained by measuring the distance from 3 along 3,B. D, E and F are obtained similarly. It can be seen that, for the more appropriate starting point, the "hyposentre" converges to a stable position whereas for the other starting point this does not happen.

S.P.S. FUNCTION AND SUBROUTINE SUBPROGRAMS TO BE CALLED BY FORTRAN MAINLINE PROGRAMS.

AUTHORS:

RICHARD LA RUE & JOHN POWELL
THE UNIVERSITY OF SOUTH DAKOTA
VERMILLION, SOUTH DAKOTA 57069

REFERENCES:

1) IBM 1620 MONITOR I SYSTEM REFERENCE MANUAL FILE NO. 1620-36; FORM C26-5739-4
2) IBM 1620-1443 MONITOR I LISTINGS & FLOW CHARTS 1620-PR-033(CARD)

MACHINE REQUIREMENTS:

1) 1620 MODEL I, 20,000 DIGITS CORE STORAGE
2) (1) IBM 1311 DISK STORAGE DRIVE
3) INDIRECT ADDRESSING

OPTIONAL EQUIPMENT:

1) IBM 1443 PRINTER
2) ADDITIONAL INSTRUCTIONS: MF, TNS, TNF

S P S CONTROL CARDS NECESSARY TO ASSEMBLE AND STORE (AND LIST)

S P S SUBPROGRAMS

```
CARD COLUMN          1111111111222
                     12345678901234567890 12
                     *ASSEMBLE RELOCATABLE
                     *STORE RELOADABLE
                     *NAME XXXXXX
                     *LIST PRINTER
```

The Indicator Record (Monitor Manual page 134)

Each subprogram to be called by a FORTRAN program must contain a header record to identify the routine and to provide other essential information. The SPS instructions necessary to create this record are shown below:

```
line 1      S     DS    ,*+101
line 2            DC    6,987898,5-S
line 3            DAC   6,NAMEbb,7-S
line 4            DVLC  22-S,5,LAST,2,ff,2,kk,5,Entry Address-6,5,0
line 5            DC    30,0,66-S
line 6            DSC   17,0,0
line 7            DORG  S-100
```

Name you choose for Routine

Comments:

1) Lines 1 and 2 are standard and required.
2) Line 3 must contain 6 characters--the name of the program--left justified.
3) Line 4: The storage operand 22-S is standard
   LAST is the label of the last digit stored
   ff is the floating mantissa length
   kk is the fixed point word length
   Entry Address-6 is required and is the label of 1st instruction-6
   The 5,0 is standard and required
4) Line 5: The 30 digit DC refers digit for digit to the subroutine listed in Table 11 page 126.
   Place a 1 instead of a 0 in each position corresponding to a library subroutine to be called from the SPS subroutine.
5) Lines 6 and 7 are standard and required.

The following three statements must conclude each subroutine:

```
        DAC   1,0     forces next available position to be even
LAST    DC    1,@     required record mark (see LAST in DVLC statement)
        DEND  1       operand indicates number of entry points
```

CALL SORT2(A(1),J(1),N) will cause the FORTRAN compiler to generate the following:

```
                                          *+11
                                       17 XXXXX XXXXX
        BTM  SORT2,*+11      Address of A(1) XXXXX
        DSA  A(1),J(1),N     Address of J(1) XXXXX
                             Address of N    XXXXX
```

The following is now that portion of the SPS subprogram that moves these arguments (A(1),J(1),N) over to the subprogram and calculates the return address to the mainline which will be the next even position following the storage location of the address of the last argument (N).

```
MATRIX DS   5
COL    DS   5
N      DS   5
RETURN DS   5
SORT2  AM   RETURN,5
       TF   MATRIX,-RETURN,2
       AM   *-6,5
       CM   *-18,*-37
       BNE  *-48
       AM   RETURN,2
       TFM  SORT2+18,MATRIX
       TF   CHECK,-N
       SM   CHECK,1,10
       BNF  G01,MATRIX
       TF   MATRIX,-MATRIX
G01
```

The first three DS statements are to store the addresses of A(1), J(1), and N. The fourth is to store the return address to the mainline. The functions of the other instructions are described below.

```
SORT2  AM   RETURN,5
```

The instruction calculates the address of the address of A(1). The value at RETURN was *+11 or the Q_{11} digit of the BTM instruction generated by the CALL statement. If you add 5 to this we get the address of the units position of the first address in the DSA statement following the BTM instruction.

```
       TF   MATRIX,-RETURN,2
```

Brings the address of A(1) over to the area addressed by the symbol MATRIX.

```
        AM    *-6,5
```

Updates the P operand of the previous instruction by 5 thus making this the address assigned the symbol COL.

```
        CM    *-18,*-37
```

This compares the number in the P operand of the instruction TF MATRIX,-RETURN,2 with the number *-37 which is equal to the core address assigned to the symbol RETURN.

```
        BNE   *-48
```

If the result of the last compare was zero the addresses of all arguments have been moved in.

```
        AM    RETURN,2
```

Calculates the next even address following the addresses in the DSA statement generated by the CALL statement. This is the return address. (Note: We would add one if there was an even number of arguments.)

```
        TFM   SORT2+18,MATRIX
```

Restores the P operand of the instruction following SORT2 so that the subprogram can be called again by the FORTRAN program.

```
        TF    CHECK,-N
```

Moves the data whose address is at N to CHECK.

```
        SM    CHECK,1,10
```

Reduces CHECK by one.

```
        BNF   GO1,MATRIX
        CF    MATRIX
        TF    MATRIX,-MATRIX
```

If the actual argument used in the FORTRAN was A(K) instead of A(1) then MATRIX contains the address of the address of the argument, These three instructions will put the address of A(K) at MATRIX which is assumed in the instruction:

```
GO1     TF    X,-MATRIX
```

```
3400032007013600032007024902402511963611300102
**JOB
**DUP
*DELETSORT2
**JOB
**SPS
*LIST PRINTER
*ASSEMBLE RELOCATABLE
*STORE RELOADABLE
*NAMESORT2
**SPS SUBROUTINE TO SORT TWO COLUMNS - 1 - FIXED, - 1' - EITHER
S     DS   ,*+101
      DC   6,987898,5-S
      DAC  6,SORT2 ,7-S
      DVLC22-S,5,LAST,2,08,2,10,5,SORT2 -6,5,0,30,0
      DSC  17,0,0
      DORGS-100
*           MATRIX IS ADDRESS OF ARRAY TO BE SORTED (FIXED OR FLOATING)
*           COL IS ADDRESS OF ARRAY TO BE CARRIED ALONG (MUST BE FIXED)
*           N IS ADDRESS OF NUMBER OF ELEMENTS
*           RETURN IS ADDRESS OF NEXT INSTRUCTION IN MAINLINE
MATRIXDS  5
COL   DS  5
N     DS  5
RETURNDS  5
SORT2 AM  RETURN,5
      TF  MATRIX,-RETURN,2
      AM  *-6,5
      CM  *-18,*-37
      BNE *-48
      AM  RETURN,2
      TFM SORT2 +18,MATRIX,,RESET FOR NEXT PASS
*           SET NUMBER OF ELEMENTS AT CHECK
      TF  CHECK,-N
      SM  CHECK,1,10
      BNF GO1,MATRIX
      CF  MATRIX
      TF  MATRIX,-MATRIX
*           SEE IF DATA IS FIXED POINT OR FLOATING POINT
GO1   TF  X,-MATRIX
      BNF FIX,X-1
      B   FLOAT
FIX   M   CHECK,KK
      SF  95
*           PUT STARTING ADDRESSES AT ADDR1 AND ADDR4
      TF  ADDR1,MATRIX
      TF  ADDR4,COL
*           PUT ENDING ADDRESS AT ADDR3
      TF  ADDR3,ADDR1
      A   ADDR3,99
A1    TF  ADDR2,ADDR1
      TF  ADDR5,ADDR4
      A   ADDR2,KK
      A   ADDR5,KK
A2    C   -ADDR1,-ADDR2
      BNH A3
      TF  TEMP,-ADDR1
      TF  -ADDR1,-ADDR2
      TF  -ADDR2,TEMP
      TF  TEMP,-ADDR4
      TF  -ADDR4,-ADDR5
      TF  -ADDR5,TEMP
```

```
A3      A     ADDR2,KK
        A     ADDR5,KK
        C     ADDR2,ADDR3
        BNH   A2
        A     ADDR1,KK
        A     ADDR4,KK
        C     ADDR1,ADDR3
        BNE   A1
        B     =RETURN
*
*             FLOATING ROUTINE
*
FLOAT   TF    ADDR1,MATRIX
        TF    ADDR4,COL
        M     CHECK,FF
        SF    95
        TF    ADDR3,ADDR1
        A     ADDR3,99
B1      TF    ADDR2,ADDR1
        A     ADDR2,FF
        TF    ADDR5,ADDR4
        A     ADDR5,KK
B2      BT    TOFAC, ADDR1
        BT    FSB, ADDR2
        BTM   FMFAC,TEMP
        BNF   *+24,TEMP=2
        B     B3
        BT    TOFAC, ADDR1
        BTM   FMFAC,TEMP
        BT    TOFAC, ADDR2
        BT    FMFAC, ADDR1
        BTM   TOFAC,TEMP
        BT    FMFAC, ADDR2
        TF    TEMP,=ADDR4
        TF    =ADDR4,=ADDR5
        TF    =ADDR5,TEMP
B3      A     ADDR2,FF
        A     ADDR5,KK
        C     ADDR2,ADDR3
        BNH   B2
        A     ADDR1,FF
        A     ADDR4,KK
        C     ADDR1,ADDR3
        BNE   B1
        B     =RETURN
CHECK   DS    10
ADDR1   DS    5
ADDR2   DS    5
ADDR3   DS    5
ADDR4   DS    5
ADDR5   DS    5
X       DS    30
TEMP    DS    30
*             FF IS FLOATING MANTISSA LENGTH + 2
```

```
FF      DC    2,10
*             KK IS THE FIXED WORD LENGTH
KK      DC    2,10
FAC     DS    ,24C2
TOFAC   DS    ,34C8
FUFAC   DS    ,3452
FSB     DS    ,4066
        DAC   1,0
LAST    DC    1,0
        DEND1
```

```
340003200701360003200702490240251196361130010 2
#FORX52
*LIST PRINTER
      DIMENSION A(20,5),N(20,5),L(20)
      DO 10 I = 1,20
      DO 10 J = 1,5
      N(I,J)=-I*J
   10 A(I,J)=N(I,J)
      PRINT 101,((A(I,J),J=1,5),I=1,20)
  101 FORMAT(1H1/(5F10.2))
      DO 15 M = 1,5
      DO 5 I = 1,20
    5 L(I)=I
      CALL SORT2(A(1,M),L(1),20)
   15 PRINT 102,(L(J),A(J,M),J=1,20)
  102 FORMAT(1HO/(I5,F10.2))
      PRINT 103,((N(I,J),J=1,5),I=1,20)
  103 FORMAT(1H1/(5I10))
      DO 14 M = 1,5
      DO 25 I = 1,20
   25 L(I)=I
      CALL SORT2(N(1,M),L(1),20)
   14 PRINT 104,(L(J),N(J,M),J=1,5)
  104 FORMAT(1HO/(2I10))
      CALL EXIT
      END
```

SAMPLE PROGRAM USING SORT2 SUBROUTINE

```
**JOB
**DUP
*DELETFLOAT
**JOB
**SPS
*ASSEMBLE RELOCATABLE
*STORE RELOADABLE
*LIST PRINTER
*NAME FLOAT
      S      DS   ,*+101
             DC   6,987898,5⌴S
             DAC  6,FLOAT ,7⌴S
             DVLC22⌴S,5,LAST,2,08,2,04,5,FLOAT⌴6,5,0
             DC   30,0,66⌴S
             DSC  17,0,0
             DORGS⌴100
      ARG    DS   5
      RETURNDS  5
      FLOAT  AM   RETURN,5
             TF   ARG,⌴RETURN
             AM   RETURN,2
             TF   FAC,⌴ARG
             BTH  SFLOAT,0,10
             B7   ⌴RETURN
      SFLOATDS  ,4042
      FAC    DS   ,2492
             DAC  1,0
      LAST   DC   1,0
             DEND1
```

```
##JOB
##DUP
*DELETIFIX
##JOB
##SPS
*NAMEIFIX
*ASSEMBLE RELOCATABLE
*STORE RELOADABLE
*LIST PRINTER
     S      DS   ,*+101
            DC   6,987898,5=S
            DAC  6,IFIX   ,7=S
            DVLC22=S,5,LAST,2,08,2,04,5,IFIX=6,5,0
            DC   30,0,66=S
            DSC  17,0,0
            DORGS=100
     ARG    DS   5
     RETURNDS  5
     IFIX   AM   RETURN,5
            TF   ARG,=RETURN
            AM   RETURN,2
            BNF  *+36,ARG
            CF   ARG
            TF   ARG,=ARG
            BT   TOFAC,ARG
            BTM  FIX,0,10
            B7   =RETURN
     TOFAC  DS   ,3408
     FIX    DS   ,3854
            DAC  1,@
     LAST   DC   1,@
            DEND1
```

```
$$JOB
$$FORXSZ
*LIST PRINTER
**    DEMO FOR IFIX AND FLOAT--FUNCTION SUBPROGRAMS WRITTEN IN S P S
      X = 14.
      I = 2
      XI = I
      Y = X/XI
      PRINT 1,Y
      Y = 0.
      Y = X/FLOAT(I)
      PRINT 1, Y
      J = IFIX(X)/I
      PRINT 2, J
    1 FORMAT(1H0,F10.2)
    2 FORMAT(1H0,I8)
      END
```

Absolute Monitor Addresses to certain subroutines one might wish to use.

```
FAD     DS    ,4090,,Entry Address to Addition Subroutine
FSB     DS    ,4066,,Entry Address to Subtraction Subroutine
FMP     DS    ,4138,,Entry Address to Multiplication Subroutine
FD      DS    ,4162,,Entry Address to Division Subroutine
FIX     DS    ,3854,,Entry Address to Fix Subroutine
FLOAT   DS    ,4042,,Entry Address to Float Subroutine
FAC     DS    ,2492,,Low order position of FAC (Floating Accumulator)
TOFAC   DS    ,3408,,Entry Address to load FAC
FMFAC   DS    ,3452,,Entry Address to unload FAC
ENTLN   DS    ,2248,,Entry Address to log Subroutine
ENTEXP  DS    ,2253,,Entry Address to Expon. Subroutine
ENTSQT  DS    ,2318,,Entry Address to SQRT Subroutine
ENTABS  DS    ,2323,,Entry Address to ABS Subroutine
```

SESSION REPORT

COMMON - Chicago

Session Number MON D4 Session Name 1130 Users Experience Panel

Chairman W. C. Gray

Time 3:30 to 5:00 PM Attendance (No.) Estimate 275

Speakers W. C. Gray
Wagner Electric Corp.
11444 Lackland Road
St. Louis, Mo. 63141

W. F. Burggrabe Jr.
NOOTER, Inc.
140 South Third Street
St. Louis, Mo. 63166

Synopsis of Meeting I. Preventive Measures to keep from bombing pack.
1) Operator training, 2) Disk Maintenance, 3) Proper use of job cards.
II. Ease of recovery. Use "Darm" Type II program
III. How to repair "DCOM"
IV. How to repair Disk sector addresses. Procedures for III and IV will be submitted to CAST.
Mr. Gene Lester of IBM stated most of these procedures are being included in DCIP of Monitor Version 2.

SESSION REPORT

COMMON - Chicago

Session Number MON D7 Session Name 360 OS Committee

Chairman W. Norton

Time 3:30 to 5:00 PM Attendance (No.)

Speakers Eugene D. Fitzpatrick - Illinois Stat University

Synopsis of Meeting The speaker presented a paper entitled, "What a University Computing Facility Expects From the Operating System."

1. OS COMMITTEE

2. "WHAT A UNIVERSITY COMPUTING FACILITY EXPECTS FROM THE OPERATING SYSTEM"

3. EUGENE D. FITZPATRICK

4. ILLINOIS STATE UNIVERSITY

5. 106 MOULTON HALL, COMPUTER SERVICES, ILLINOIS STATE UNIVERSITY, NORMAL, ILLINOIS 61761

6. MONDAY, APRIL 8, 1968 AT 3:30 P.M. SESSION D-7

7. THREE PAGES OF TEXT.

WHAT A UNIVERSITY COMPUTING FACILITY EXPECTS FROM THE OPERATING SYSTEM

AN ADDRESS TO THE "SYSTEMS 360 OPERATING SYSTEM COMMITTEE"
BY EUGENE D. FITZPATRICK

WHEN YOUR CHAIRMAN, MR. WADE NORTON, INVITED ME TO PRESENT A PAPER TO THIS COMMITTEE I EXPLAINED THAT OUR UNIVERSITY HAS NOT AS YET INSTALLED OUR SYSTEM 360/40 COMPUTER. HIS RESPONSE INDICATED THAT THE COMMITTEE WAS MORE INTERESTED IN EXPECTATIONS THAN IN A RECITAL OF COMPLAINTS CONCERNING LIMITATIONS AND SHORTCOMINGS OF THE OPERATING SYSTEM. THIS ADDRESS THEREFORE WILL INCLUDE MANY FEATURES CURRENTLY INCORPORATED IN THE OPERATING SYSTEM PLUS ADDITIONAL FEATURES WHICH WE WOULD LIKE TO SEE INCORPORATED. WHILE ALL OF THE FEATURES LISTED ARE TECHNICALLY POSSIBLE, IT IS RECOGNIZED THAT SOME OF THESE FEATURES WILL BE DIFFICULT TO ACCOMPLISH AND MAY THEREFORE BE REJECTED BY THE IBM PROGRAM DEVELOPMENT DEPARTMENT.

THE UNIVERSITY WHERE I AM CURRENTLY EMPLOYED HAS AN ENROLLMENT OF APPROXIMATELY 12,000 STUDENTS AND IS SUPPORTED BY STATE REVENUE FUNDS. IN INSTITUTIONS SUCH AS OURS IT SEEMS TYPICAL THAT THERE ARE LIMITED FUNDS FOR INSTALLING COMPUTING FACILITIES; CONSEQUENTLY, WE FEEL FORTUNATE IF WE CAN AFFORD TO INSTALL A MODEL 40 WITH 128 K BYTES OF CORE MEMORY. THIS ONE MACHINE MUST SERVE THREE MAJOR PURPOSES IN THE ACADEMIC SETTING, (1) ADMINISTRATIVE DATA PROCESSING, (2) ANALYZING RESEARCH DATA FOR FACULTY AND/OR STUDENTS AND, (3) PROVIDING LABORATORY FACILITIES FOR THE INSTRUCTION OF STUDENTS. SINCE EACH OF THESE FUNCTIONS IS ESSENTIAL IT IS DESIRABLE TO OPERATE IN THE MULTI-PROGRAMMING MODE. SINCE THE UNIVERSITY IS LIKELY TO REFLECT THE NEEDS OF ALMOST EVERY OTHER TYPE OF INSTALLATION THE REQUIREMENTS ON THE OPERATING SYSTEM WILL BE MANY AND VARIED.

THE INSTRUCTIONAL NEEDS OF THE UNIVERSITY WILL COME FROM MANY AND DIVERSE DISCIPLINES SUCH AS MATHEMATICS, BUSINESS ADMINISTRATION, ACCOUNTING, CHEMISTRY, PHYSICS, PSYCHOLOGY, EDUCATION, SOCIOLOGY, ECONOMICS, AND MANY OTHERS. IN ORDER TO MEET THESE DIVERSE NEEDS IT IS DESIRABLE TO HAVE COMPILERS OF THE MOST FREQUENTLY USED LANGUAGES SUCH AS COBOL, FORTRAN, RPG, ALGOL, AND PL/1. ALTHOUGH MOST OF THE PROGRAMS WRITTEN BY STUDENTS ARE LIKELY TO BE BRIEF, IT WOULD BE NICE IF THE OPERATING SYSTEM HAD A PAGING TECHNIQUE SO THAT A GIVEN PROGRAM OPERATING IN A GIVEN PARTITION WOULD HAVE SEEMINGLY UNLIMITED CORE IN WHICH TO FUNCTION. MODULARITY AND CHAINING ARE PERHAPS DESIRABLE GOALS IN PROGRAMMING; HOWEVER, SUCH LIMITATIONS ARE A CONFOUNDED NUISANCE TO A GROUP OF NOVICES WHO ARE TRYING TO LEARN THE PRINCIPLES OF

COMPUTER OPERATIONS. IT WOULD ALSO BE A BOON TO THE OCCASIONAL USER IF THE SUBROUTINES WRITTEN IN A GIVEN LANGUAGE COULD BE CALLED BY ANY OTHER LANGUAGE. ALL OF THE SUBROUTINES WRITTEN IN A GIVEN LANGUAGE COULD BE CALLED BY ANY OTHER LANGUAGE. ALL OF THE SUBROUTINES ARE ULTIMATELY TRANSLATED INTO MACHINE LANGUAGE IN ORDER TO FUNCTION, SO IT WOULD SEEM THAT THEY SHOULD BE AVAILABLE TO ANY MAINLINE PROGRAM REGARDLESS OF THE LANGUAGE USED. ANOTHER VERY DESIRABLE FEATURE WOULD BE A COMMON DATA BASE IN THE FILE STRUCTURE. THE SECURITY OF RESTRICTED FILES IS OF COURSE PARAMOUNT, BUT THERE ARE MANY FILES THAT CAN BE OF VALUE TO MANY PEOPLE. IT WOULD SEEM DESIRABLE FOR SUCH FILES TO BE AVAILABLE TO THESE PEOPLE REGARDLESS OF THE LANGUAGE THEY ARE USING.

IN MANY UNIVERSITIES SUCH AS OURS THE DISCIPLINE OF INDUSTRIAL TECHNOLOGY IS BEING UPDATED TO TAKE ADVANTAGE OF NUMERICAL CONTROL OF THE VARIOUS MACHINES. THERE SHOULD BE SOME PROVISION FOR COMPUTER PROCESSORS OF NUMERICAL CONTROL PROGRAMS. MORE FLEXIBLE PROGRAMS FOR CONTROLLING INCREMENTAL PLOTTERS ARE ALSO NEEDED BY UNIVERSITY COMPUTING FACILITIES SO THAT THEY WILL BE ABLE TO SERVICE THE REQUESTS FROM MANY DIFFERENT DISCIPLINES SUCH AS CHEMISTRY, PHYSICS AND THE BEHAVIORAL SCIENCES, AS WELL AS INDUSTRIAL TECHNOLOGY.

IN OUR INSTITUTION LEVEL E COBOL WILL PROBABLY SATISFY MOST OF THE PROGRAMMING NEEDS FOR ADMINISTRATIVE DATA PROCESSING; HOWEVER, THERE WILL BE TIMES WHEN THE ADVANTAGES OF LEVEL F COBOL SHOULD BE EXPLOITED. ONE SUCH TIME IS AT THE END OF THE TERM WHEN THE PROCESSING OF GRADES BECOMES THE HIGHEST PRIORITY JOB. IT THEREFORE SEEMS DESIRABLE TO HAVE AN EASY METHOD OF REDEFINING THE PARTITIONS OF THE SYSTEM IN ORDER TO SWITCH FROM ONE SET OF PRIORITIES TO ANOTHER. IT WOULD ALSO SEEM DESIRABLE TO INCORPORATE WHAT MIGHT BE CALLED A "WARM START" IN ADDITION TO THE TRADITIONAL "COLD START" SO THAT JOBS REQUESTED IN ONE PERIOD OF PROCESSING COULD BE COMPLETED IN A LATER PERIOD OF PROCESSING. IN THE CURRENT "COLD START" PROCEDURES IT IS NECESSARY TO RESUBMIT REQUESTS FOR INCOMPLETE JOBS WITH A SUBSEQUENT WASTE OF VALUABLE COMPUTER TIME.

AS REMOTE TERMINALS ARE ADDED TO THE SYSTEM FOR TELEPROCESSING IT WOULD SEEM DESIRABLE FOR A TELEPROCESSING SYSTEM SUCH AS RAX TO BE OPERATING IN ONLY ONE OF THE PARTITIONS RATHER THAN CAPTURING THE ENTIRE PROCESSOR. IT WOULD ALSO SEEM TO BE DESIRABLE FOR LANGUAGES IN ADDITION TO FORTRAN AND BASIC ASSEMBLER TO BE ABLE TO OPERATE IN THIS TELEPROCESSING ENVIRONMENT. SUCH ADDITIONAL LANGUAGES SHOULD, OF COURSE, HAVE COMPILERS THAT WOULD FUNCTION IN CONVERSATIONAL MODE THROUGH THE REMOTE TERMINALS SO THAT WE CAN HAVE A MORE EFFECTIVE MAN/MACHINE INTERFACE.

AS TELEPROCESSING BECOMES INCREASINGLY FLEXIBLE THERE MUST BE GREATER EFFORT IN HAVING THE COMPUTER AUTOMATICALLY PROCESSING THE COMPLEX ASPECTS OF THE MAN/MACHINE INTERFACE SO THAT THE OPERATION OF THE REMOTE TERMINAL WILL BECOME A RELATIVELY SIMPLE AND EASY TASK. IT MUST BE REMEMBERED THAT MANY OF THE POTENTIAL USERS OF REMOTE TERMINALS WILL BE HIGHLY SKILLED IN DISCIPLINES OTHER THAN COMPUTER TECHNOLOGY AND WILL CONSEQUENTLY PREFER TO DEVOTE THEIR TIME AND ENERGIES TO MAKING DECISIONS IN THEIR AREA OF SPECIALTY RATHER THAN LEARNING TO BECOME COMPUTER TECHNOLOGISTS. IT WOULD THEREFORE SEEM THAT THE SUCCESS OF OUR ATTEMPTS TO HAVE PEOPLE AT THE ADMINISTRATIVE LEVEL TAKE ADVANTAGE OF THE INFORMATION THAT CAN BE SUPPLIED BY A COMPUTER WILL DEPEND PRIMARILY ON THE EASE WITH WHICH THEY CAN OBTAIN THIS INFORMATION.

I WISH TO THANK MR. NORTON AND THE COMMITTEE FOR INVITING ME TO DELIVER THIS PAPER. IT IS HOPED THAT BY DISCUSSING THE EXPECTATIONS OF A UNIVERSITY COMPUTING FACILITY WE CAN CONTINUE TO IMPROVE THE OPERATING SYSTEM FOR THE 360 COMPUTER AND ITS SUCCESSORS.

SESSION REPORT

COMMON - Chicago

Session Number MON E1 Session Name 360 DOS

Chairman A. Ragsdale

Time 5:30 to 7:30 PM Attendance (No.) 95

Speakers 1) Mr. Gerry Kaplan - IBM (Full FORTRAN Under DOS)

2) Mr. Otto Bufé - Arthur G. McKee Company (FORTRAN "Alter" System)

Synopsis of Meeting 1) Full FORTRAN Under DOS was announced by Mr. G. Kaplan of IBM. Availability was stated as the first quarter of 1969. The additional facilities provided will be: a) LOGICAL IF statement, b) DATA Statement, c) more than 3 dimensions, d) DEBUG facility, e) CALL passing a literal, f) Object time FORMAT statements, g) LABELED COMMON, h) Generalized TYPE statements, etc. The recommend reference manual is FULL FORTRAN C28-6515.

2) The User Paper "The FORTRAN 'Alter' System" was presented by Mr. Otto Bufé. This is a method of conviently updating FORTRAN source modules, and is well suited for use in testing and debugging procedures. Card handling and card reading are reduced to a minimum.

It is the intent of the IBM Corporation to provide a DOS/360 Fortran IV compiler. The language level of the compiler will be the same as the OS/360 Fortran IV (G) compiler. This presentation describes ~~the differences in~~

those FORTRAN IV Features

not in

BASIC FORTRAN IV as currently available in the DOS D level compiler.

PRESENTED BY
G. P. KAPLAN
IBM WHITE PLAINS

SESSION MON. E1

LOGICAL IF, LOGICAL

DATA, BLOCK DATA, LABELLED COMMON

DEBUG
COMPLEX
More than 3 DINENSIONS,
    ADJUSTABLE DIMENSIONS,
    GENERALIZED SUBSCRIPTS

GENERALIZED TYPE STATEMENT, IMPLICIT

CALL BY NAME, CALL ('LITERAL'),
    ENTRY, RETURN i

ASSIGN, ASSIGNED GOTO

OBJECT TIME FORMAT STATEMENTS,
PRINT, PUNCH, READ b, list, END, ERR
NAMELIST, PAUSE LITERAL, G, Z, L
FORMAT CODES

COMPLEX, LOGICAL, LITERAL, HEX CONSTANTS

* LOGICAL IF

IF (LOGICAL EXPRESSION) STATEMENT

```
IF (A.LE.0)     GO TO 25
IF (A.GT.0)     A = 0
```

*DATA INITIALIZATION STATEMENT

```
    DIMENSION A (5), B(3, 3), L (4)
    DATA A/5*1.01, B/9*2.0/,
        L/4*. TRUE./, C/'FOUR'/
```

*LABELED COMMON

```
    COMMON A, B, C /ITEMS/X, Y, Z
```

*BLOCK DATA SUBPROGRAM

```
    BLOCK DATA
    COMMMON . . .
    DIMENSION . . .
    DATA
    EQUIVALENCE
    TYPE
```

*DEBUG FACILITY

```
DEBUG SUBCHK (A), TRACE,
    INIT (B, C), SUBTRACE

AT
TRACE ON
TRACE OFF
DISPLAY   X, Y, Z
```

*COMPLEX DATA and ARITHMETIC

ARRAYS:

* More than 3 DIMENSIONS

* ADJUSTABLE DIMENSIONS

```
CALL SUB (A, 5, 4)

    SUBROUTINE SUB (A, I, J)
    DIMENSION A (I, J)

    A (K, L) = . . .
```

* GENERALIZED SUBSCRIPTS

```
A (I + J/4*K**2)

        + B (.3*A(M, N))
```

*GENERALIZED TYPE STATEMENT

One STATEMENT DESCRIBES
TYPE
LENGTH
DIMENSION
INITIALIZATION

```
REAL A(5, 5)/20*6.9,5*1./, B(100)
    /100*0./, TEST*8(5)
```

*IMPLICIT

```
IMPLICIT INTEGER*2(A - H),
    REAL *8 (I -K), LOGICAL (L, M)
```

*CALL BY NAME

ARRAYS, SUBPROGRAMS,
VARIABLES IN SLASHES

SUBROUTINE SUB (/x/, x)

*CALL ('LITERAL')

*ENTRY STATEMENT

*RETURN i

CALL SUB (A, &30, &40)

SUBROUTINE SUB (B, *, *)

RETURN

RETURN 1

RETURN 2

*ASSIGN, ASSIGNED GOTO

```
      ASSIGN   30 TO A
      GOTO     A (20, 30, 40)
```

*OBJECT TIME FORMAT STMTS

```
    READ (5, FMT) LIST
```

*READ b, LIST
PUNCH b, LIST
PRINT b, LIST

*READ (A, B, END = c, ERR = d) LIST

```
READ (4, 10, END = 300, ERR = 900) LIST
```

*NAMELIST

```
NAMELIST /NAMI/A, B, I, J
WRITE (6, NAMI)
      &NAMI  A = 4.00, B = 267.34 . . .
                        & END
```

*PAUSE 'DO NOT PROCEED'

*FORMATS

G - GENERALIZED
(INTEGER, REAL, COMPLEX, LOGICAL)

Z - HEXADECIMAL

L - LOGICAL

*CONSTANTS
COMPLEX, LOGICAL, HEX, LITERAL

FORTRAN "ALTER" SYSTEM

PRESENTED AT THE

COMMON MEETING

APRIL, 1968

CHICAGO, ILLINOIS

OTTO E. BUFE
PROCESS ENGINEER

ARTHUR G. McKEE & COMPANY
CLEVELAND, OHIO

FORTRAN "Alter" System

Summary

The FORTRAN "Alter" System is a method of conveniently updating FORTRAN source modules, and is well suited for use in testing and debugging procedures. Source modules are stored on disk in the Assembler sub-library and most of the operations are performed on disk, resulting in increased efficiency of operations. Card handling and card reading are reduced to a minimum.

The "Alter" system is comprised of 7 McKee utility programs as follows:

1. Card to disk
2. Sort
3. Update
4. Produce control cards for catalog function
5. Documentation program
6. Write last record on punch file
7. Disk to printer

The programs are written in E level FORTRAN IV, and are presently operating on a 64K Mod 30 under release #14 of DOS.

Background

Soon after operations were started on our 360, we realized that it was very inefficient to run a compile-and-go on most FORTRAN testing, due to the slow operation of the FORTRAN compiler. Furthermore, since

FORTRAN "Alter" System

DOS does not furnish the ability to compile directly from source module to relocatable library, many test runs could not be conveniently put into a job stream. We were able to stand this at first, since initially there was a fair amount of available time on the computer. However, as the work load increased, the time for testing and debugging was accordingly reduced, and we started thinking about procedures to make better use of the test time. Development of the "Alter" system was a step in the right direction.

Description

For maximum speed on our system (4 disk drives, 2 tape drives), most of the operations would have to be performed on disk, meaning that FORTRAN source modules must reside on disk. The DOS residence disk was selected to hold these modules rather than a separate disk. The Assembler sub-library in the Source Statement Library was used to store the FORTRAN source modules, due to lack of a FORTRAN sub-library. Because of our philosophy of operation, we are able to dedicate about 1/3 of the residence disk to the source library. All of our FORTRAN modules at McKee are presently stored in this sub-library.

Under the operation of the system, all FORTRAN source modules, when first coded and punched on cards, are put into the Assembler sub-library using McKee utility program #1 mentioned before. All modules

require a header card, giving a 4 character name and an 8 character name, which will correspond to the name of the module in source and relocatable libraries respectively. The program puts an identification on all cards in cols. 73-80, the first 4 characters being the source library name and the last 4 being an integer multiple of 10, indicating the card position within the module. This identification limits the effective number of cards per module to 999, but we have not found this to be a problem.

After the source module is in the source statement library, McKee's utility programs are used to perform various functions, such as updating the source and/or relocatable libraries, or performing documentation functions. Input to the first utility program (#2) in the job consists of the type of function to be performed, and the change cards for each module (if required). The program reads the cards and sorts the change cards (if present) to assure the correct sequence of records for updating. McKee utility program #2 also creates an input file for the source statement library service program (SSERV) to access the FORTRAN modules from the source statement library and an input file for McKee utility program #3.

McKee utility program #3 updates the source modules, if required, and creates input files for the FORTRAN compiler (if required) and the maintenance routines in DOS. If the source modules are updated, the source statement library is updated with the latest versions of the

modules. The procedure used in updating is similar to the one used in the QUIKTRAN operating system. In QUIKTRAN, any source statement in a module may be inserted, deleted, or replaced by specifying a line number and the altered coding, if applicable. In the McKee "Alter" System, the modules are renumbered as they are updated.

McKee utility program #4, used in conjunction with the FORTRAN compiler, produces the header card (CATALR name) necessary to catalog an object module to the relocatable library. The program accepts the header card for each module as input and produces it on the punch file. Note that several header cards may be produced in series if one or more compiles in the job are terminated. In this case, the valid header card is always the last one in the series. If the compile of any module is terminated, this utility program effectively deletes the object module in the relocatable library.

Some of you may be familiar with the documentation type program that we included in the "Alter" system as McKee utility program #5. The original program is available through COSMIC, an information center set up to distribute NASA computer programs, and produces a flowchart from FORTRAN source statements. We have made a few modifications to the program, one of which was to enable the program to arrange the statement numbers of a source module in ascending sequence, and revise the corresponding statement numbers in the transfer statements of the module. A control statement specifies the first statement number of the program,

if the statement numbers are to be arranged. A zero statement number in this card will perform the flowcharting function on the following cards of the module. We have found this program to be useful in debugging and testing as well as in documentation. If patches to a program are to be made involving new statement numbers, these numbers can be readily determined.

McKee utility program #6 is executed after the last module in the job has been read by the FORTRAN compiler (if object modules are to be catalogued to the relocatable library). The program writes several records on the punch file to effectively close the file and return system control to the card reader.

At this point, I would like to show a few overlays to illustrate the system. Overlay 1 is a display of a FORTRAN source module from the source library. Note that the header card is given an integer identification so it can be altered as well as any of the other statements. Overlay 2 is a display of the same FORTRAN module after the statement numbers have been arranged. McKee utility program #5 presently outputs all statement numbers in I3 format - hence the 3 digit statement numbers. Overlay 3 is a flowchart of the same FORTRAN module, made on a 1403 printer. The flowchart program requires no input other than the FORTRAN source statements.

Modifications to the Compiler

Before the "Alter" system could run successfully using disk files, it was necessary to modify the FORTRAN I/O module (IJTFIOS). The original

module writes 260 byte records on disk, which is unacceptable to either the FORTRAN compiler or maintenance programs, both of which accept 81 byte records. The modification consisted of revising the Channel Command Words for the READ and WRITE routines so they will accept 81 byte records. The number of records read or written per track was increased from 11 to 25.

Another modification to the IJTFIOS module was necessary for the flowcharting function of McKee utility program #5. Since a full printed line was necessary for this function, the 2nd modification to the IJTFIOS module consisted of changing only the Channel Command word for the READ routine to accept an 81 byte record, leaving the WRITE routine to accept a 260 byte record. McKee utility program #7 then could be used in conjunction with the documentation program to read several short records and print a full line.

Note that no modification of the IJTFIOS module is necessary if tape files are used in the operation of the "alter" system. In fact, the "alter" system was first implemented on a combination disk-tape system, where only the initial and final operation involved disk. However, our 2 tape units are the slowest and most error prone 2415's IBM makes, and we really get hung up waiting for the tapes to rewind! Needless to say, we converted to a complete disk operation as soon as possible.

FORTRAN "Alter" System

Techniques Used

We used some special techniques in the McKee "Alter" System to reduce the number of control cards to a minimum. Five extra disk files (in addition to the 4 standard files) were created by means of the Standard Label (STDLABEL) option. Four of these additional files were given the same extents and label information as one of the 4 standard files. One of the files created was a SYSIN file, which is assigned to various disk units at different times during the execution of the programs.

The two modifications of the IJTFIOS were catalogued into the relocatable library under different header names. There are now 3 object modules existing in our relocatable library with identical program names. By using an INCLUDE control card with the name of the modified IJTFIOS, we can override the original module.

We have been using the "Alter" system at McKee now for about 5 months and the FORTRAN programmers are well satisfied with it. Programmer efficiency has been increased because more can be accomplished with one computer run. There is a slight increase in compile speed, due to the input being taken from disk instead of cards, and also an increase in speed in cataloguing both to the source and to the relocatable libraries, all adding up to increased computer efficiency.

SESSION REPORT

COMMON - Chicago

Session Number MON E2 Session Name Solution of Large Simultaneous Equations

Chairman W. A. Pease, Jr.

Time 5:30 to 6:15 PM Attendance (No.)

Speakers Suresh R. Phansalker of the Badger Company

Synopsis of Meeting Solution of Large Systems of Simultaneous Equations of the LinearNonhomogeneous Type, which display the characteristics of symmetry and sparseness of the coefficient matrix is uneconomical, when it is necessary to hadle and store the entire matrix.

By using Cholesky's method for the half symmetric ban running diagonally across the overlapped area of the two subject matrices, much core space and operating time may be saved. The program is being presented to the library and the technique is in the paper presented.

"SOLUTION OF LARGE SYSTEMS OF LINEAR NONHOMOGENEOUS SIMULTANEOUS EQUATIONS WITH BAND SYMMETRIC COEFFICIENT MATRIX "

A paper presented by

Suresh R. Phansalkar at the

Chicago COMMON meeting on

April 8, 1968

Monday Session E2

SOLUTION OF LARGE SYSTEMS OF LINEAR NONHOMOGENEOUS SIMULTANEOUS EQUATIONS WITH BAND SYMMETRIC COEFFICIENT MATRIX

INTRODUCTION:

In many scientific applications, it is often necessary to solve large systems of simultaneous equations of the linear nonhomogeneous type which further display the characteristics of symmetry and sparseness of the coefficient matrix. It is uneconomical both in computation time and core storage if it is required to store the entire coefficient matrix and operate on all of its elements, most of which may be zeros.

Cholesky's method affords a very convenient means to secure both these objectives. For a general system of n equations, Cholesky's method[1] requires: n^2 + (a number of the order of n) operations; whereas the Gauss elimination requires: $\frac{n^3}{3}$ + (a number of the order of n^2) operations. Further economy is effected by taking into account the symmetry and sparseness of the coefficient matrix.

BASIC THEORY:

The theory underlying this method as well as the method itself has been well known for sometime. For a system of equations

$$AX = C \quad \text{----------} \quad (1)$$

the method consists of determining a lower triangular matrix L with which the system of equations (1) can be reduced to the unit upper triangular form, viz

$$TX = K \quad \text{----------} \quad (2)$$

where
- A = coefficient matrix in original system
- C = vector (s) of constants in original system
- T = unit upper triangular matrix
- K = derived vector(s) of constants
- X = vector of unknowns

Thus in essence, by pre-multiplying the system (2) by the lower triangular matrix L, system (1) is obtained.

Then:

$$L(TX - K) = AX - C = 0$$

This implies:

$$LT = A \text{ --------(3)}$$
$$LK = C \text{ --------(4)}$$

or, by adding (3) and (4)

$$L\left[T + K\right] = A + C \text{ --------(5)}$$

It is known [2] that factorization of any arbitrary square matrix, such as A, into a lower triangular and an upper triangular matrix, as is implied in equation (3) above, is uniquely possible if the elements on the principal diagonal of either L or T are given known values. In Cholesky's scheme, the elements on the principle diagonal of T, the upper triangular matrix, are all taken to be 1.

Once the unit upper triangular matrix T and the derived vector of constants K are obtained, the unknowns X are obtained from system (2) by a simple process of "backward sweep".

In practice the elements of L and the augmented T + K matrix are obtained from the parent A + C augmented matrix row after row, from left to right. The formulae to obtain the elements l_{ij} and t_{ij} respectively of the L and T matrices are:

a) $l_{i1} = a_{i1}$ --------i = 1---n n = no. of equations

Where a represents elements of augmented matrix $[A + C]$.
That is, first column of L is same as first column of $[A + C]$.

b) $t_{1j} = \dfrac{a_{1j}}{a_{11}}$ ------ j = 1 --------(n + no. of sets of right hand constants)

c) $l_{ij} = a_{ij} - \sum_{r=1}^{j-1} l_{ir}\, t_{rj}$

$j = 1,$ to i

d) $t_{ij} = \dfrac{1}{l_{ii}} \left[a_{ij} - \sum_{r=1}^{i-1} l_{ir}\, t_{rj}\right]$

$j = (i + 1),$ to (n + no. of sets of right hand constants)

c) and d) above clearly explain how elements of both L and T are obtained row wise.

For example, for a generalized coeff. matrix A (slide 2), an element, such as, say, l_{63} of the lower triangular matrix L is given by:

$$l_{63} = a_{63} - \left[l_{61}\, t_{13} + l_{62}\, t_{23}\right]$$

$$l_{66} = a_{66} - \left[l_{61}\, t_{16} + l_{62}\, t_{26} + l_{63}\, t_{36} + l_{64}\, t_{46} + l_{65}\, t_{56}\right]$$

$$t_{67} = \frac{1}{l_{66}} \left[a_{67} - (l_{61}\, t_{17} + l_{62}\, t_{27} + l_{63}\, t_{37} + l_{64}\, t_{47} + l_{65}\, t_{57}\right]$$

CASE OF BAND SYMMETRIC "A" MATRIX

Assume now that the matrix "A" is known to have symmetry @the principal diagonal and non zero elements only within the region bounded by the 2 off-diagonal lines. (Slide 1) For such a matrix, it can be shown that the upper triangular matrix T will also display the characteristic of non zero elements within the band width only.

Further, the L and T matrices for such a matrix display the additional property that;

$$l_{ij} = t_{ji} \times l_{jj} \qquad i \neq j$$

This can be readily seen from the illustrative example [Slide 3]

This then suggests that:

1) Only elements in the band width as defined in Slide 1 need be stored as the coefficient matrix, elements below the principle diagonal being unnecessary.

2) Only the diagonal elements l_{ii} and the elements of the upper triangular matrix T, t_{ij}, need be calculated, elements l_{ij} $(j<i)$ being computed only as an intermediate product and not required to be stored.

(4)

METHOD USED BY AUTHOR:

It is possible with a slightly different approach to the computation of the elements l_{ii} and t_{ij} to automatically eliminate operations on zero elements.

Consider the triangular wedge starting with row 4 and ending on row 8 (Slide 2) with t_{48}, l_{88} and t_{812} on its apex points.

Then if elements l_{88} and t_{8j} are being generated, the elements in row 4 i.e.; l_{44} to t_{47} will be skipped over and the product of

$$t_{48}\, l_{84} = t_{48}\,(t_{48} \times l_{44})$$

will be formed and retained as a progressive sum. Further, when elements in the 5th row are being processed, elements l_{55} to t_{57} will be skipped over and the product $t_{58} \times l_{85} = t_{58} \times (t_{58} \times l_{55})$ will be formed and added to the previous value. Thus, when the product

$$t_{78} \times l_{87} = t_{78} \times (t_{78}\, l_{77})$$

is formed and added to the previous sum, we can compute $l_{88} = a_{88}$ - SUM

Similar progressive products are formed and added and elements such as say, t_{810} are formed by:

$$t_{810} = \frac{1}{l_{88}} \left[a_{810} - \text{SUM}\right]$$

Thus, to compute the elements of the 8th row, l_{88} and t_{89} to t_{812}, only the elements in the triangular wedge directly above are required.

In the above example, if rows 4 to 7 are in core memory the elements of the 8th row in the triangularized matrix could be generated without any reference to disk storage.

In the program, the variable T is designated to hold NØRD rows, each row containing NB3 = NB1 + NLDC elements.

Where:
NB1 = band width
NLDC = No. of sets of right hand constants.

For a large band width, say 100, and 20 sets of right constants, each row of the upper triangular matrix T will contain 120 elements and NORD then equals $\frac{1200}{120}$ = 10 rows. Every time NØRD rows of the upper triangular matrix are generated, they are written on disk.

The auxiliary variable T1(1200) is used toa bring in the previous rows of the triangularized matrix, since a total of (NB1-1) previous rows of the T matrix (Slide 2) is required to generate the elements in the last row of T. It would have been possible to double the extent of T instead of having a separate variable T1 but then when newer previous rows are brought in T the previous contents of T (excepting the last row) will be lost. This means that it would be necessary to write the triangularized array row by row. It was, therefore, thought preferable to write NØRD rows of T at one time. Every time the last row is generated, rows are shifted up by 1. Thus after NORD cycles, NORD rows of array T will be written on disk.

USE OF ONE DIMENSIONAL ARRAYS:

If the array T were to be defined as T(12, 100), NØRD could not exceed 12. But if in a very large system, the bandwidth is say, just 10 and there are two sets of right hand constants, with one dimensional subscripting, it is possible to process NØRD = 100 rows in the array T. Thus, the triangularized matrix can be written at the rate of 100 rows at one time, instead of 12, for 2-dimensional subscripting. Incidentally even the object-time dimensioning facility in FORTRAN IV does not afford this flexibility, which can only be obtained by one dimensional subscripting.

Once the entire triangularized matrix (T + K) is written on disk, NBACK rows (starting from the last row towards the first) can be brought in memory and solutions obtained simultaneously to all the NLDC sets of constants, in clusters of NB1. The solutions are written on the disk in contiguous manner, so that all the n solutions for the unknowns corresponding to the 1st set of right hand constants are grouped together, then those for the next set of right hand constants and so on.

DISK WRITE OPERATIONS:

A) I. B. M. 1620:

The program written for I. B. M. 1620, uses a precision of 10 significant digits for floating point arithmetic. Thus, a sector on the 1620 disk would contain 8 words. Although the program uses one dimensional subscripting, triangularized matrix is written in units of 1 row each. In other words, every row containing (NB1 + NLDC) elements is written on an integral number of sectors to facilitate the read back operation.

The solutions to equations obtained in NLDC sets simultaneously in clusters of NB1 are written so that NB1 solutions of the first set occupy an integral number of sectors, (containing NNB1 values where NNB1 $\geq$ NB1) NB1 solutions of the next set occupy the same number of integral sectors, with enough space left in between to accommodate the remaining solutions of the first set.

B) I. B. M. 360/44

The author has a version of the program for an I. B. M. 360/44 also. The chief point of difference is that the minimum record size on the disk in this case is 360 bytes and care must be taken to prevent waste of disk space. A special DISKIO routine was, therefore, written which effects buffered writing in units of 90 (single precision) words. The in core calculations are performed in double precision but the triangularized matrix is written on disk in single precision to save disk space.

SUMMARY:

The I. B. M. 1620 version of the program, which is in the form of a main program, can solve about 250 equations with 10 sets of right hand constants. By redefining the working area on the disk and/or adding more disk drives to an installation, the program could be used to solve larger systems. The version for an I. B. M. 360/44 (with 2 single disk storage drives) can handle about 1000 equations with the use of one 2315 disk for work area, assuming that some disk space would be reserved for the program utilizing this routine.

The program output consists of the triangularized matrix, unknowns solved for in back sweep (NLDC sets of NNB1 values), sets of solutions fetched in memory (one at a time) and the rearranged actual solutions.

The augmented coefficient matrix $[A + C]$ is read in row by row from cards. (Only non zero elements above the principal diagonal are required.) The difference in input to a general purpose routine for simultaneous equations and that to the author's version is seen from Slides 3 and 4.

An illustrative example of 12 equations with 5 sets of right hand constants and a band width of 5 has been solved and the output of the author's program is attached as Appendix.

Cholesky's method, known to be the fastest [1] of the elimination methods, has thus been effectively adopted by the author to solve large systems of simultaneous equations with band symmetric coefficient matrix and a number of sets of right hand constants, so as to eliminate all operations on zeros outside the band width.

REFERENCES:

1. M. G. Salvadori, & M. L. Baron; Numerical Methods in Engineering

2. V.N. Faddeeva; Computational Methods of Linear Algebra

APPENDIX

(Solution of Large Systems of Linear Nonhomogeneous Simultaneous Equations with Band-Symmetric Coefficient Matrix)

1)	Program Output	Sheets 1 through 6
2)	Slides	Sheets 7 through 12

SESSION REPORT

COMMON - Chicago

Session Number MON G1 Session Name 1130 SOUND OFF

Chairman G. F. Schoditsch

Time 8:00 to 10:00 PM Attendance (No.) 175

Speakers Vann Hettinger and Gene Lester - IBM

Synopsis of Meeting: Members proposed questions to IBM regarding the MONITOR system version 1 and 2, hardware problems, type II packages, and other user questions and question areas. IBM took note of the following items: 1. Improve compatibility of 1130 and 360 FORTRAN

2. Improve Assembler Manual

3. Remove the restriction in DM2 that an integer constant used as a subscript may not point to an element outside the range specified in the DIMENSION.

4. Establish default option for ONE WORD INTEGERS

5. 1403 will space or skip even though READY.

6. Include 1403 and 1132 carriage clutch in not ready signal to 1130.

7. The "Green Sheet" (Bibliography) is 3 months late announcing DM 2.

8. Type 2 program notes:
   A. Time Delay Routine needed for CSMP
   B. Electric Load Flow should be Type II supported
   C. PCS should supprt Free Float
   D. ECAP should be Type II supported
   E. A routine similar to "360 Matlan" supported

SESSION REPORT

COMMON - Chicago

Session Number TUE A1 Session Name 360 Full Project

Chairman W. Norton Meeting

Time 8:30 to 10:00 AM Attendance (No.)

Speakers W. B. Garrison - IBM, Chicago

Synopsis of Meeting A presentation was made of the OS/360 Languages.

OS/360 Languages

I General Introduction - design points

ASSEMBLER E, F
COBOL E, F
FORTRAN E, G, H
PL/I F
ALGOL F
RPG E

II Characteristics of Individual Language Systems

A. Assembler - for maximum machine utilization -
- Symbolic, relatively free form, expressions
- Macro instructions - library and inline - (SYSGEN)
- Source 'COPY' library
- Data management & Control Program Services
- Testran debug facility
  - Testran macros
  - Testran Editor
- E (18K) and F (44K) versions
- Used by IBM to build OS/360
- Used by most users - degree varies widely
  - System programming
  - Augmenting High Level Languages
  - Via macros as an application language

B. COBOL E(17K) and F(80K)

- IBM COBOL - COBOL '65 subset with extensions
  - Canonical form of data declarations - picture
  - DASD, ISAM extensions
  - Transform verb - S/360 utilization
  - Debugging Source Statements
  - Floating point features
  - Subprogram facility
  - I/O Declaratives for exceptional conditions
- COBOL F adds:
  - Sort Verb
  - Report Writer feature
  - Extended COPY facilities. BASIS — Corresponding option
- Data management Access methods
  - BSAM / QSAM
  - BISAM / QISAM
  - BDAM
- USASI Standardization activity
  - multiple levels in 7 functional areas
- Primary language for half of OS users
  - used heavily by 2/3.
- Unrevised I/O facilities plus decimal arithmetic

C. FORTRAN E (15K) G (80K) H (200K)

- USASI (ASA) BASIC (E) and full (G, H)
- BCD (7000 series) & EBCDIC (S/360) Support
- E level extensions:
  ~~[illegible]~~ for READ, WRITE
  Double Precision arithmetic
  Mixed mode arithmetic
  Quotes in Format lists in lieu of 'H' spec's.
  T 'Tab' option.
  3 dimensional arrays
  P - scale factors
- H language:
  above plus extensions beyond full Fortran -
  IMPLICIT classification
  Extended type statement (length spec.)
  All purpose 'G' conversion
  Multiple entry points & non standard returns
  Arrays to 7 dimensions
  NAMELIST for I/O without explicit list & F'list.
  Hex constants & Format code
- G language:
  above plus debugging features ~~[illegible]~~
  ~~extensions to COBOL, PL/I.~~
- Recap E, G, H
  E Fast compile - limited storage
  G " " full language, debugging feat.
  H. Optional fast & optimizing compiles - large systems.

- Graphic Services for FORTRAN 2250 + L.P.
- Primary Language for 20%, used by half of OS users

D. PL/I F (44K)

- Language appropriate for both FORTRAN, COBOL & mixed application area - all data types
- Debugging facilities part of language
- 3rd generation facilities -
  - interrupts - enabling disabling - 'global' interrupts
  - supervisor linkages - time, date, tasking
- computational library plus all access methods (beyond COBOL)
- One general purpose compiler -
  - fast compilation
  - limited optimization
- Users report - 'programmer productivity'
- Subset on coding sheet & classroom
- 'Macro' facility - self extension.
- Usage primarily from assembler - commercial orientation.

E. ALGOL F (44K)

- ECMA + IFIP Subsets of ALGOL '60
- BSAM I/O Support
- Aimed at Algol user to ease conversion to S/360
- no expanded support planned -

F. RPG E (15K)

- Prior system RPG facilities plus:
  multiple input data sets
  table lookup
  calculation branch (looping)
  QSAM, BDAM + ISAM
- decimal oriented
- useful for reporting & limited computing.
- Usage light compared with DOS, BOS, M20
  System writers - use in COBOL, PL/I.

III Direction of future development

A. PL/I for areas not equipped - T.P., data base

B - Usability, efficiency improvements for FORTRAN & COBOL generally within standards.
- New requirements weighed against needs, costs.

C. RPG, Assembler, developments will continue.

SESSION REPORT

COMMON - Chicago

Session Number **TUES A2** Session Name **MPX I Introduction**

Chairman **R. P. Walker**

Time **8:30 to 10:00 AM** Attendance (No.) **91**

Speakers **Tom Candy - IBM**

Synopsis of Meeting **Tutorial Session**

PREFACE

The objective of this marketing guide is to provide you with a fundamental understanding of the new 1800 Multi-Programming Executive System (MPX), for the purpose of communicating this information to customers and prospects.

The guide is organized first to provide a general systems summary and then a more detailed specification.

The illustrations and diagrams in the guide were designed to help meet educational requirements, permitting view graph foils to be made directly from this material.

The word "area" is used frequently in the discussion of MPX in this guide, and is synonymous with the much used word "partition".

TABLE OF CONTENTS

TABLE OF CONTENTS (Con't)

MPX

FEATURING----------

- 26 PARTITION MULTIPROGRAMMING CAPABILITY
- COMPLETE INPUT/OUTPUT OVERLAP
- AUTOMATIC PROGRAM SCHEDULING
- MAXIMUM ON-LINE AVAILABILITY
- ON-LINE HARDWARE DIAGNOSTICS
- FLEXIBILITY TO FIT MANY APPLICATIONS

PHILOSOPHY

MPX is defined as a real-time multiprogramming operating system capable of maximizing the efficiency and throughput of the IBM 1800 Data Acquisition and Control System computer (figure 1). It is designed to asynchronously time-share several independent processes with concurrent background batch processing functions (figure 2). The increased throughput provided by MPX is accomplished through sophisticated input/output handling techniques, making the central processing unit available during all I/O operations. The capability exists for the MPX system to be configured into a maximum of 26 unique multiprogramming areas.

MPX control programs are designed modularly to provide extended flexibility in covering a wide spectrum of applications, thus minimizing the probability of user modifications due to unique application requirements.

Multiprogramming is achieved through the use of the programmed settable interrupt feature in the 1800 hardware, eliminating time consuming list searching techniques. The major objective within the design of MPX is the minimization of overhead, or unproductive processing.

HIGHLIGHTS

The System Executive within MPX provides an extremely flexible set of subroutines to manipulate the hardware and software interval timers. Periodic interrupts are now automatically generated for the user, thus alleviating the scheduling problem. It is also possible to create a specified time delay occurring between two instructions or statements within the same program. This interval of time is then allocated by the system to other programs requiring service. The ability to cancel any of the interval timers is also provided through a subroutine call.

Multiple programmed interrupts per level are provided, including a new interrupt handling concept which makes it possible to simulate the occurrence of an actual process interrupt.

Coreloads may automatically be queued in to designated areas with the occurrence of a process, programmed, or periodic interrupt, alleviating nearly all of the scheduling burden.

The background Batch Processing Monitor is automatically allocated idle real-time by the Time Sharing Control program. The entire background operation can be prioritized relative to other real-time programs queued into the same area of core, giving an override capability when batch processing is immediately required.

Source level distinction is minimized between real-time and non real-time programs providing an easy transition from program development in the batch stream to on-line execution as a real-time program.

Subroutines requiring immediate response to timed, process, or programmed interrupts are now built into a special coreload which is designed to reside in core until some modification is required by the user. This provides fast in-core response and user modification capability while the system remains on-line.

Complete flexibility in the handling of real-time errors is now possible through user intervention, and direction of error diagnostic and recovery options. Subroutine calls provide restart, reload, and abort functions. The error print routine provides information which helps in identifying the real source of the error and is directly callable by the user. A diagnostic dump analysis program clearly labels all pertinent areas of core to expedite checkout in the early phases of an installation.

The Disk Management Program (DMP) dynamically controls the allocation of bulk storage to MPX system programs, subroutines, user coreloads, and data files. An option exists for the user to strategically position data files and coreloads, giving better response and throughput.

A System Maintenance program operating under DMP permits on-line modifications to disk resident IBM system programs and library subroutines due to IBM supplied modifications. To complement this, On-Line Diagnostics are supplied for the 1053 printer, 1442 card read/punch, 2310 models A and C, and the Analog to Digital Converter in DPC mode.

A very useful set of utilities are available to perform disk copy, disk patch, dumps, and general system documentation.

FORTRAN READ/WRITE operations are overlapped to take advantage of the multiprogramming capability. Also, all input/output control routines are directly callable at the FORTRAN level, eliminating time consuming overhead of FORTRAN linkage subroutines.

System Generation is performed under complete control of the Batch Processing Monitor, providing streamlined operating procedures.

In summary, the MPX Operating System is designed with the objectives of maximizing on-line availability and throughput, and minimizing overhead.

TIME-SHARING

Time-sharing is a term most often misunderstood in its relationship to the application disciplines of control systems.

MPX TIME-SHARING

Time-sharing as defined under MPX is the allocation of resources by the central processing unit between several independent real-time applications while concurrently performing background batch processing functions.

Time-sharing under MPX should not be mis-construed as a massive terminal oriented time slicing system providing an increment of time by rotation to hundreds of users.

Figure 2 illustrates the interplay on a priority response basis between several real-time programs and a background batch processing function.

MPX COMPARED TO TSX

TSX is a single area operating system designed on a core exchange concept for the smaller user. Due to its one area design, disk I/O time cannot be overlapped during program loading, except randomly through higher priority executions. TSX operates very efficiently where the process or real-time programs reside in fixed core and the background is reserved for the Non Process Monitor. However, when real-time programs are disk resident, the time-sharing capability can get bogged down due to frequent core exchanges and non-overlapped disk I/O time.

MPX on the other hand is an "n" area multiprogramming system, where "n" is defined as twenty seven minus the number of I/O device levels. Assuming at least one level of I/O, its maximum number of areas is 26. In most cases three to five areas will be typical. The use of multiple areas of core to contain disk resident programs gives the advantage of executing one program while another is on its way to core.

MPX is designed for the large user who may have several processes to control and needs to squeeze the last available increment of time out of his 1800.

MPX THROUGHPUT ANALYSIS

Figure 3 illustrates a graphical comparison of the MPX and TSX philosophies. By utilizing area 2 for a real-time coreload, MPX can avoid two disk operations of saving and restoring core, while improving response time to the real-time coreload by one disk operation or 100 percent. Core exchanges occur in TSX whenever the transition is made between process and non-process, or for interrupt coreloads.

By overlapping the disk load time of the real-time coreload and internal I/O TIME such as analog input, and at the same time avoiding two disk operations, a substantial gain in throughput can easily be realized.

Figure 3 is not meant to unconditionally state that a 40% gain in throughput will be inherent in MPX. Careless MPX systems design can produce a negligible advantage while, on the other hand, the proper configuration of an MPX system can in fact improve throughput by large percentages as illustrated.

SYSTEM SPECIFICATIONS

MPX MINIMUM MACHINE REQUIREMENTS

A. 1801 or 1802 with 24 K words

B. 1053 printer

C. 1442 card read/punch

D. 2310 A2 (2 disks)

NOTE: A 16 K one disk system is feasible with MPX but will probably be severely limited in multiprogramming capability. This is due to minimum core requirements of 7.5 K words for the System Executive and 5120 words for the Batch Processing Monitor, leaving approximately 3 K words for extra coreload areas. The use of assembler language could perhaps make multiprogramming feasible in a 16 K 1800.

MPX CORE STORAGE ORGANIZATION

During system generation (SYSGEN) core storage is partitioned into a configuration which should best meet the response requirements of the processes under control. Core is organized into two basic functions; (FIGURE 4)

A. SYSTEM EXECUTIVE (core resident)

- Fixed Area (approximately 300 words)
- COMMON / INSKEL / (user defined)
- Executive I / O (disk, printer, etc.)
  includes Error Alert Control
- Executive Director
  includes MIC, ITC, PSC, TSC

} Minimum 7.5 K

- User and MPX included subroutines
- Patch area (whatever is left)
- Executive Linkage Tables (EBT, ETV, etc.)

B. CORELOAD PARTITIONS

- Special Coreload Area (SPAR)
  contains interrupt subroutines requiring fast response which are capable of being modified on-line.
- Coreload Areas (for real-time coreloads) 23 Maximum
- Variable Core (mimimum 5120 words)
  contains either real-time coreloads or the Batch Processing Monitor.

NOTE! A 26 area system includes; 23 coreload areas, SPAR, Variable Core, and the user subroutines in the System Executive.

MPX

EXECUTIVE

DIRECTOR

EXECUTIVE DIRECTOR

MASTER INTERRUPT CONTROL

- Automatically queues coreloads to any area
- Expanded programmed interrupts CALL LEVEL (LL, BB) 16 per level
- Process interrupts may be simulated
- Permits user written ISS routines
- Interrupt coreloads are interruptable by higher level interrupt coreloads
- Out of core interrupt handling is optional
- All interrupt subroutines end with RETURN
- All coreloads end with CALL EXIT
- Interrupt subroutines may be rebuilt on-line in the SPAR area.

Master Interrupt Control

New Interrupt Concept

MPX external and programmed interrupts are now handled consistently by a technique utilizing a pseudo PISW word. Process interrupt bits and programmed interrupt bits are both ORed into the pseudo PISW word when their respective interrupts occur. Often there are many unused bits within a PISW on each level. These bits may now be used for programmed interrupts. Also a program normally activated as the result of a process interrupt, may also be activated by the appropriate level and bit combination for a programmed interrupt. This is useful not only to simulate the occurrence of an actual process interrupt, but also applicable in certain real-time situations.

The new interrupt concept is extremely powerful since four unique types of programs may be activated under either condition;

A. In-core with the System Executive
B. SPAR resident subroutines
C. Automatically queued coreloads (any area)
D. Interrupt Coreloads.

EXECUTIVE DIRECTOR

PROGRAM SEQUENCE CONTROL (PSC)

- Directs Linking and Chaining of all coreloads

  CALL LINK (NAME)

- Standard exit procedure for all coreloads

  CALL EXIT

- System option exists for the following :

  CALL SPECL (NAME)

  CALL BACK

EXECUTIVE DIRECTOR

PROGRAM SEQUENCE CONTROL (PSC)

- Queues coreloads into specified areas on a priority basis

CALL QAREA (NAME, PRIOR, LEVEL, AREA, ERROR)

Where,

NAME = Coreload name

PRIOR = Sequence priority

LEVEL = Execution priority

AREA = Area number for coreload

ERROR = Variable which is set to :

1 = entered into queue
2 = not entered since queue was full

PROGRAM SEQUENCE CONTROL (PSC)

- Remove a coreload from the queue

CALL DEQUE (NAME, PRIOR, LEVEL)

Where,

NAME = Coreload name

PRIOR = Sequence priority

LEVEL = Execution priority

- Completely clear out a given level queue

CALL CLERQ (LEVEL)

Where,

LEVEL = the queue for the level specified

INTERVAL TIMER CONTROL (ITC)

- Sets a programmed interrupt at the completion of a time delay

CALL DELAY (LEVEL, BIT, TNO, INTVL)

Where,

LEVEL = Programmed interrupt level
BIT = Programmed interrupt bit (pseudo)
TNO = Timer number (I-II)
INTVL = Desired interval for delay

NOTE : (LEVEL, BIT) specifies either an interrupt subroutine which can reside in SPAR, or an automatically queued coreload.

INTERVAL TIMER CONTROL (ITC)

- Provides automatic periodic execution of a programmed interrupt subroutine or queued coreload on a specified cycle.

CALL CYCLE (LEVEL, BIT, TNO, INTVL)

Where,

LEVEL = Programmed interrupt level
BIT = Programmed interrupt bit
TNO = Timer number (I-II)
INTVL = Length of periodic cycle

NOTE : (LEVEL, BIT) is either a programmed interrupt subroutine or an automatically queued coreload executing on a periodic cycle.

INTERVAL TIMER CONTROL (ITC)

- Suspends execution of a coreload for a specified increment of time

CALL SUSPN (TNO, INTVL)

Where,

TNO = Timer number (I-II)
INTVL = Time increment for suspending the coreload prior to initiating the next statement or instruction

NOTE : The entire time interval is overlapped in a multi-programming mode.

INTERVAL TIMER CONTROL (ITC)

- Execute subroutine (NAME) at the completion of a time interval

CALL DEFER (NAME, TNO, INTVL)

Where,

NAME = Subroutine which executes on timer level
TNO = Hardware or software timer number (I-II)
where, 1, 2 = hardware timers
3-11 = software timers
INTVL = Desired time interval

INTERVAL TIMER CONTROL (ITC)

- Provides automatic periodic execution of a subroutine on timer level

CALL REPET (NAME, TNO, INTVL)

Where, NAME = Subroutine on timer level
TNO = Timer number (I-II)
INTVL = Periodic time interval

NOTE : This CALL would be normally issued at cold start. From then on, subroutine (NAME) will be periodically called into execution.

INTERVAL TIMER CONTROL (ITC)

- Cancel the operation of a specified timer.

CALL CANCL (TNO)

Where, TNO = Timer number (I-II)

- Read the real-time clock in hours, minutes, and seconds

CALL TIME (HH, MM, SS)

- Read the real-time clock in hours and thousandths

CALL CLOCK (H)

- Set the real-time clock in hours and thousandths

CALL SETCL (H)

EXECUTIVE DIRECTOR

TIME SHARING CONTROL (TSC)

- Automatically allocates idle CPU time to the Batch Processing Monitor if a CALL EXIT is made and the mainline level queue is empty.

- Automatically suspends the Batch Processing Monitor if a coreload is queued for the mainline area with a priority higher than that established for the Batch Processing Monitor.

- Queued coreloads with a priority lower than the Batch Processing Monitor are executed between batch jobs in the mainline area.

- The Batch Processing Monitor may be given top mainline level priority.

MPX

EXECUTIVE

I/O

and

CONTROL

ROUTINES

(IOCR)

EXECUTIVE I / O

- Queues requests for I / O when device is busy
- Initiates all I / O operations
- Suspends user coreloads during I / O
- Re-enables suspended coreloads when I / O is complete
- Transfers to user subroutines when I / O is complete
- Initiates previously queued requests when device is free
- IOCR directly callable from FORTRAN
- Overlap I / O within a coreload as in TSX
- Notifies the user of I / O errors
- Queued lists are dynamically storage protected

EXECUTIVE I / O

List Structure

The I / O list is the heart of MPX multiprogramming capability. The list structure format illustrated in the following diagram is basically the same for all I / O, with the exception of process I / O where extra control information is required.

The Link/Busy word is used to point to the next I/O list queued in some other area waiting to be serviced. When the I / O operation has successfully been completed, this word is set to zero and can thus be tested for a successful operation if desired.

The Exit Type word is used to determine the type of I / O call. If this word is non-zero, a type 2 call is assumed, and the address in this word denotes a user operation complete subroutine which is entered when I / O is terminated. If this word is zero and the call is from a coreload area, it is considered as a type 3 call and the coreload is suspended for the duration of the I / O operation. If the same call is made from Variable Core, it is treated as a type 1 call requiring a user busy test to determine I / O complete. This same procedure is used in TSX.

The next four words in the list are to be reserved for the system and are used to contain various addresses and data necessary to carry out the I / O operation.

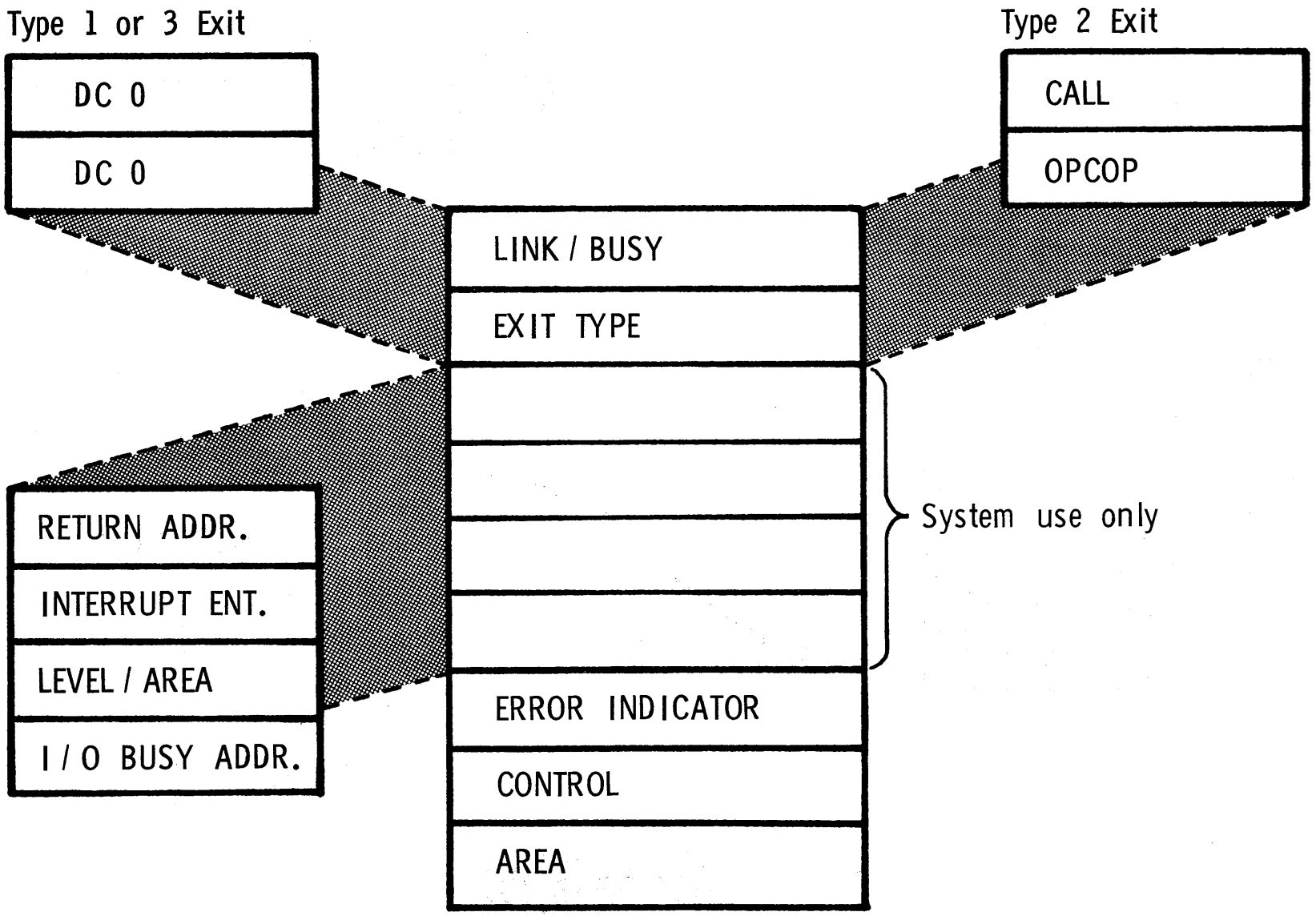
The Error Indicator word is set to various integer values depending on what happened during the I / O operation. A value of (1), tells the user that a successful operation occurred. Other values are set dependent on the type of error encountered.

The Control word is similar to TSX in that four hexidecimal digits are required to specify the type of I / O operation desired.

The Area word is also like TSX, specifying the location of the data input or output buffer.

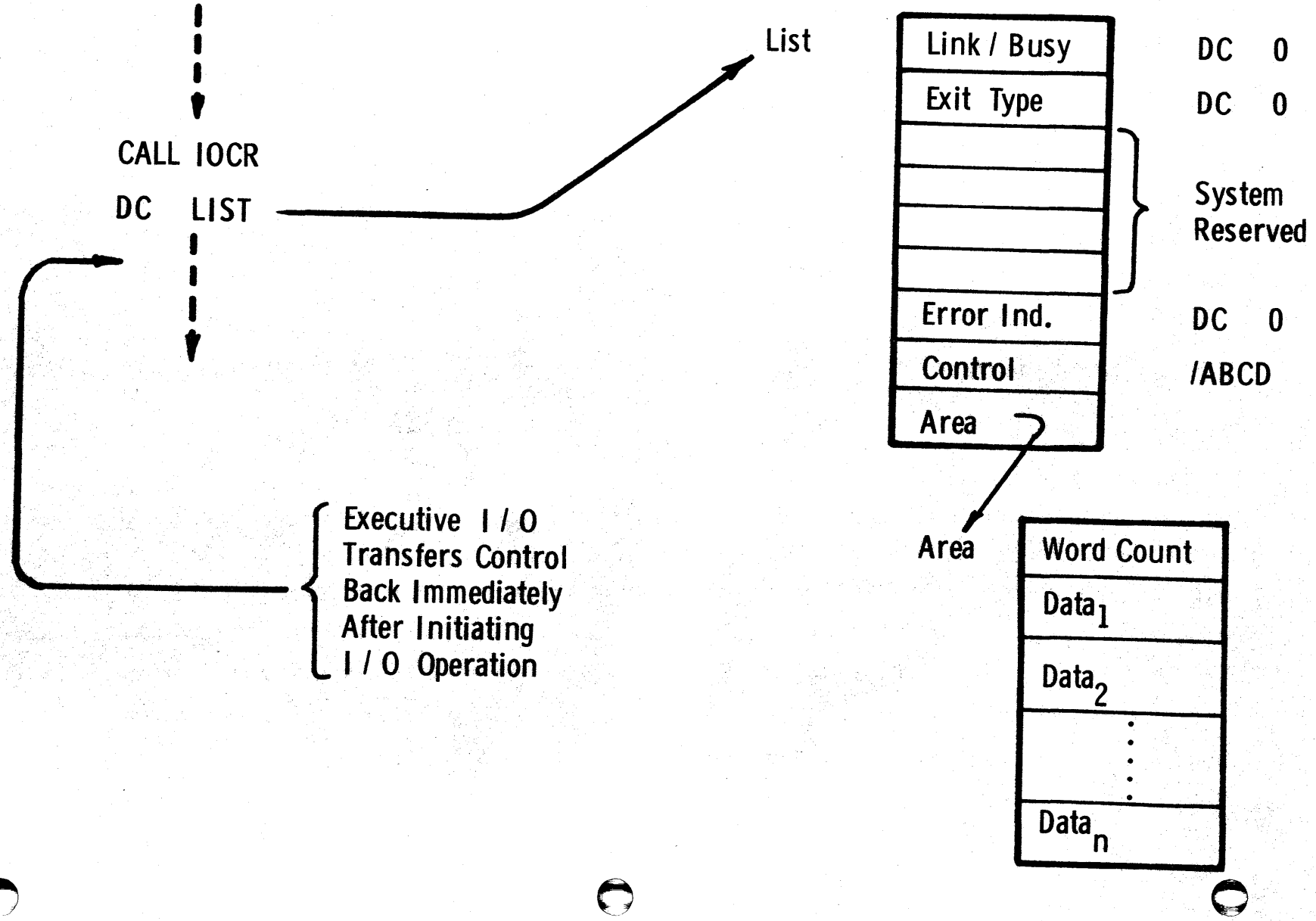
To maximize system integrity, the entire I / O list is dynamically storage protected only during the I / O operation.

EXECUTIVE I/O LIST STRUCTURE

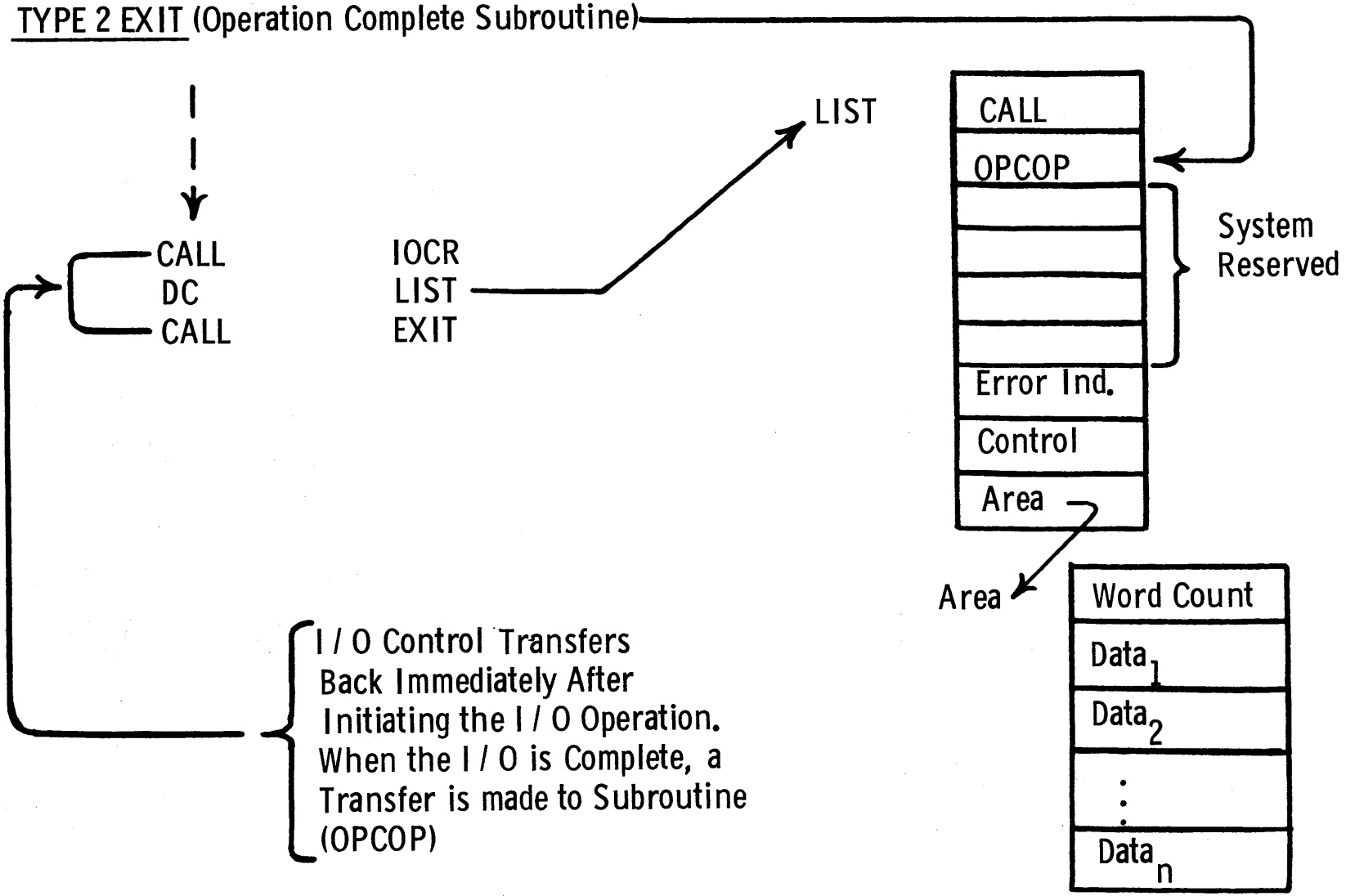


EXECUTIVE I / O

TYPE 1 EXIT (As in TSX)

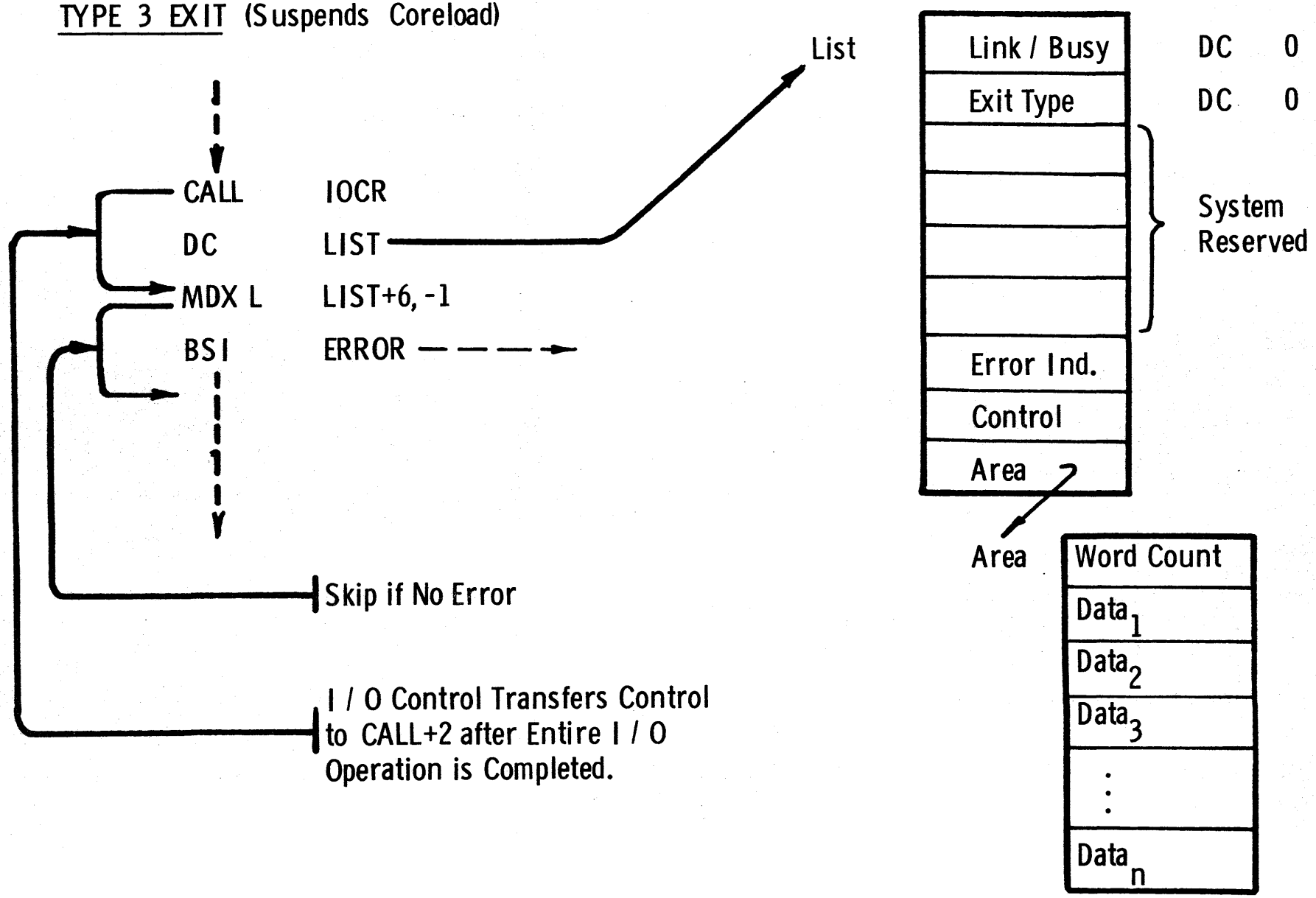


EXECUTIVE I / O



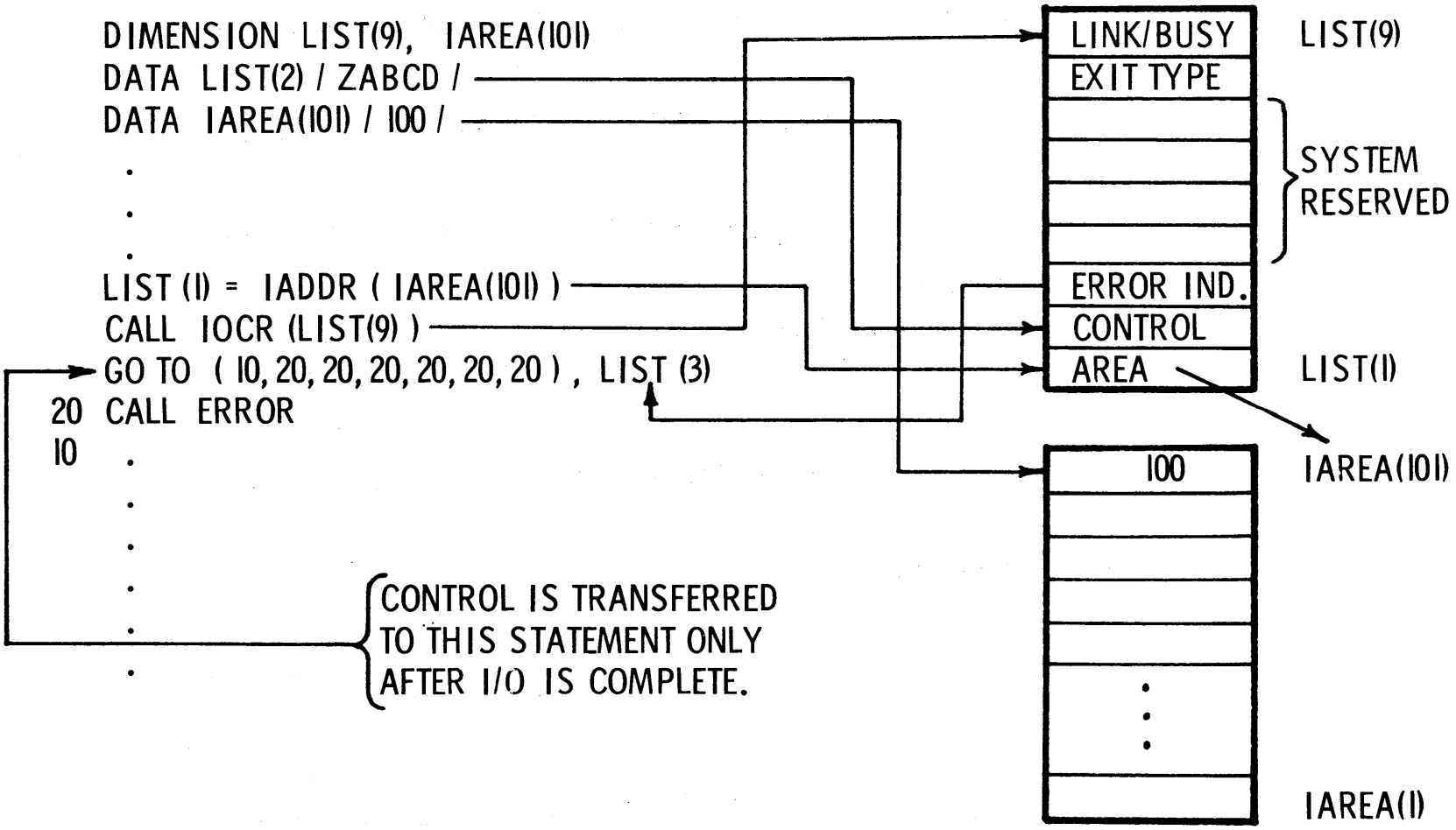
EXECUTIVE I / O

TYPE 3 EXIT (Suspends Coreload)



EXECUTIVE I/O

FORTRAN TYPE 3 EXIT



ERROR ALERT CONTROL (EAC)

- User intervention and direction for error recovery procedures

- Error Print Subroutine available to the user

- Subroutine calls available to :
  A. Restart or abort a coreload
  B. Re-load the system

- A Re-load coreload may be specified

- Error printing occurrs in unmasked mode.

- MPX programs notify user on a CALL basis of resultant errors

- A Diagnostic Dump Analysis program clearly labels all system and user coreload areas to expedite debugging.

MPX

BATCH

PROCESSING

MONITOR

BATCH PROCESSING MONITOR (BPM)

The MPX Batch Processing Monitor is very similar to the TSX Non Process Monitor in design. However many new features have been included under its control:

- On-Line hardware diagnostics for 1053 printer, 1442 card read/punch, 2310 model A and C disks, and the analog to digital converter for direct programmed control.

- Real-time multiprogrammed coreload execution under control of the supervisor.

- Streamlined SYSGEN procedures under complete control of the monitor (BPM).

- Re-entrant or Non Re-entrant subroutines selected by MPX builders or through control cards.

- The time of day is printed when encountering a // JOB card providing assistance for user job accounting.

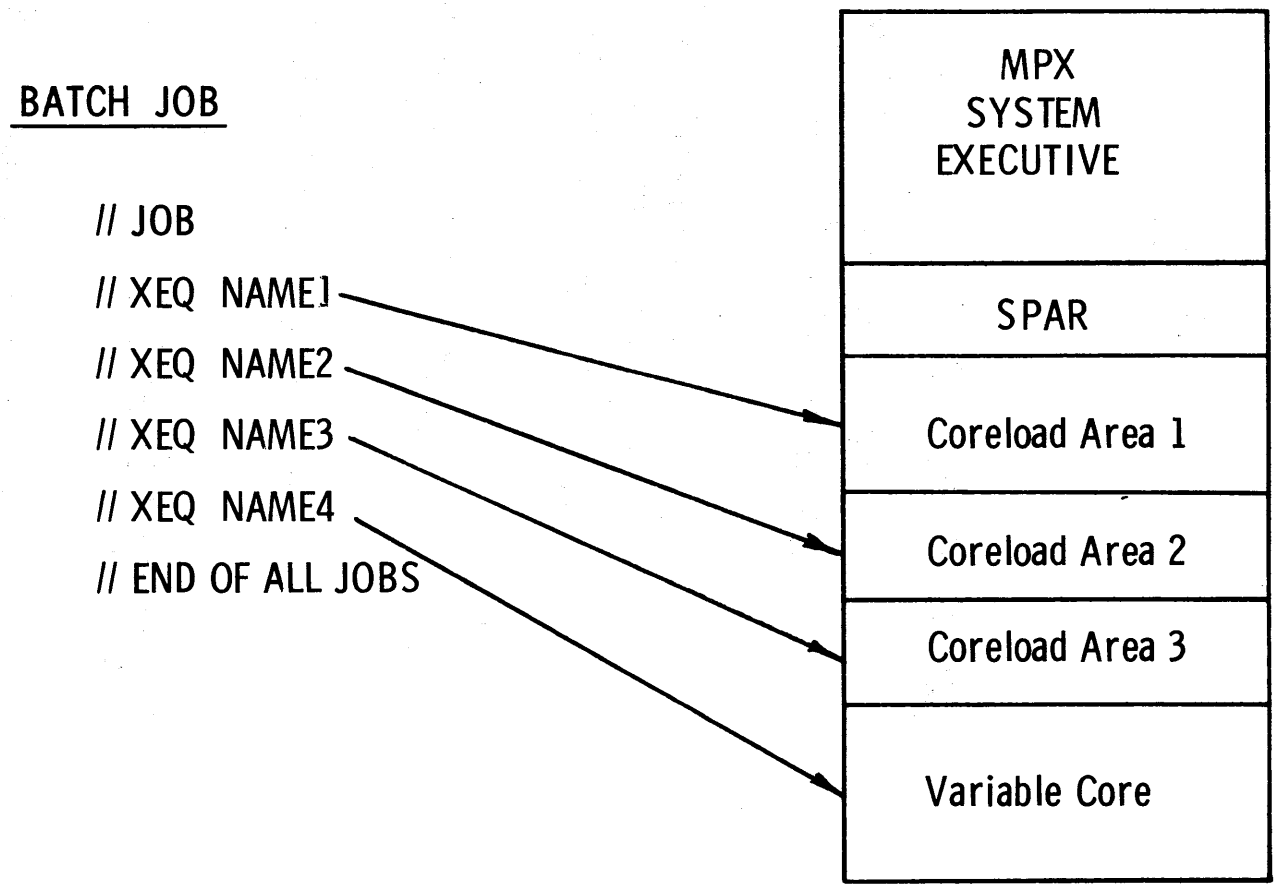
BATCH PROCESSING MONITOR

DISK MANAGEMENT PROGRAM

- Improved performance
- Defines sizes of various coreload areas
- User may specify actual disk locations for better optimization of coreloads and files
- Reserves file space without moving data
- Documentation printouts of coreload name tables and system executive tables
- On-line system program maintenance
- On-line disk patch, disk copy, dumps
- On-line modifications to coreloads and interrupt subroutines

BATCH PROCESSING MONITOR

- Real time coreloads may be executed in different areas from a batch job.



1800 MPX

FORTRAN IMPROVEMENTS

- FORTRAN I/O overlaps all READ/WRITE operations through the use of a Type 3 I/O exit.

- Input Output Control Routines (IOCR) are directly callable in FORTRAN eliminating the overhead of linkage subroutines.

- TSX - FORTRAN Compiler is used in MPX

- Multi-level shared FORTRAN I/O buffers

ASSEMBLER LANGUAGE

- TSX - Assembler is used in MPX

DYNAMIC STORAGE PROTECT SUBROUTINE

- This subroutine may be called by the user to protect and unprotect programs and/or data on a dynamic basis.

CALL STORP (FUNCT, ADDR, WDCNT)

Where, FUNCT = 0 Protect all words
= 1 Protect non-zero words
= 2 Unprotect all words

ADDR = Starting address
WDCNT = Total number of words

SYSTEM GUIDANCE

SYSTEM SIMPLICITY

The MPX System provides tools which will automatically assume the real-time programming burdens facing the TSX user.

In TSX if a coreload was to be queued as the result of an external interrupt, it was the users' responsibility to write the subroutine which performed the queueing and terminated time-sharing. Also in TSX if a coreload was to be executed periodically, it again was the users responsibility to write the scheduling subroutines to generate a periodic schedule. All of these user written subroutines were core resident and could not be modified unless the skeleton was rebuilt with the system taken off-line.

The MPX system has the facility to automatically queue coreloads into any multiprogrammed area upon the occurrence of an external interrupt, or an MPX supplied periodic interrupt. This greatly simplifies the job of programming and eliminates off-line rebuilds in order to change the scheduling parameters.

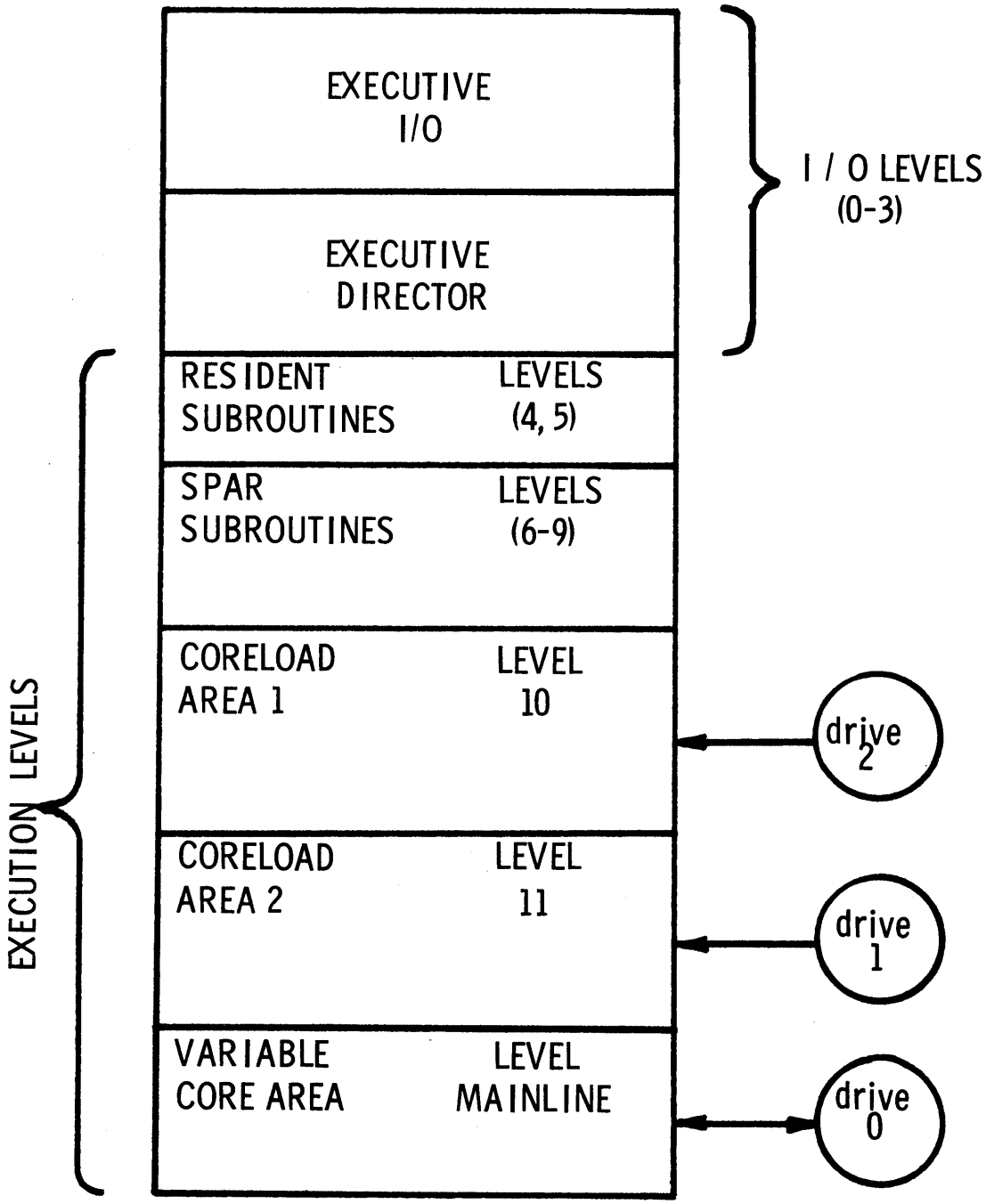
The flexibility of allowing the user to write his own scheduler remains in MPX. A Special coreload area (SPAR) is designed to contain subroutines of this type and permit on-line modifications to all subroutines within it.

MPX CONFIGURATION GUIDELINES

A very flexible system such as MPX must be carefully configured to avoid possible conflicts in interrupt level usage. The primary consideration to keep well in mind is to completely divorce I / O levels from execution levels. The disk and timers should be at the top two levels. Other considerations are; coreload levels should be unique to each area and assigned to the lowest priority interrupt levels defined in the individual system. The flexibility of multiple levels within a given area is designed to allow coreload areas to dynamically change priority relationships between themselves. This added flexibility will not however be a normal mode of operation. Resident and SPAR subroutines should not normally conflict with I / O and coreload levels.

A very efficient use of disk data channels is seen in the following illustration, allowing simultaneous loading of coreloads into all three areas. However, it is not necessary to uniquely assign a disk drive to a particular area since I / O is queued in cases where conflict exists.

TYPICAL CORE LAYOUT AND LEVEL ASSIGNMENT



CONVERTING FROM TSX to MPX

A conversion from TSX to MPX will require new systems design considerations. The conversion could result in no increased throughput if a straight conversion is attempted. However even a minimal conversion, placing all process coreloads in one coreload area and reserving variable core for batch processing, and an occasional optimization or log can produce excellent results. A more thoughtful design will produce the best results.

Interrupt coreloads in TSX could be assigned to a separate coreload area eliminating many disk operations. Interrupt subroutines in the TSX skeleton should be placed in the SPAR area of MPX giving the flexibility to modify them on-line.

Using the automatic queueing feature of MPX will greatly reduce the amount of small interrupt scheduling subroutines scattered throughout the TSX skeleton.

The careful use of the MPX IOCR routines will be the key in how much benefit can be obtained from multiprogramming capability.

SESSION REPORT

COMMON - Chicago

Session Number TUE A3 Session Name 1130 Systems

Chairman D. Gardner

Time 8:30 to 10:00 AM Attendance (No.) 150

Speakers (1) John Horn - IBM "Version 3 of CSP"

(2) Don Christians - Beloit Tool Corp. "Commercial Applications Using the 1130"

(3) Dave Dunsmore - ORVWSC "CSP Extras"

(4) Donald Gardner - General Foods Corp "FORTRAN-Coded Sorting Procedures"

Synopsis of Meeting (1) John Horn introduced Version 3 of the Commercial Subroutine Package. He also introduced the group to the 1130 USERS Guide which will be published by IBM around June 1, 1968.

(2) Don Christians explained some of the applications of CSP to the commercial environment in which he works. Sufficient interest was shown in his work to provoke him to promise submitting some of his programs to the library.

(3) Dave Dunsmore explained several subroutines he wrote to overcome some of the difficulties in using CSP. He will summarize these and submit abstracts to CAST for publication.

(4) Donald Gardner explained two FORTRAN-coded subroutines which are helpful in sorting applications. A single sector sorting subroutine forms the basis for the second subroutine.

"COMMERCIAL APPLICATION & THE 1130."

D. D. CHRISTIANS
MANAGER OF DATA PROCESSING
BELOIT TOOL CORPORATION
P. O. Box 38
So. BELOIT, ILLINOIS 61080
AREA 815-389-3461

GIVEN TUESDAY APRIL 9, 1968
SESSION #A3
COMMON CHICAGO, ILL. APRIL 7-10

PREFACE

1.) This abstract is divided into the following parts:

a.) Table of Programs
b.) Card Layouts
c.) Sample Forms & Cards
d.) System Flow Charts
e.) Source Program Listings
f.) System Description

2.) Beloit Tool Corporation was first founded in 1955. From that year until November 6, 1967 all data processing needs were satisfied by a "Tab Installation in House" consisting of a 403 Accounting Machine (Full Capacity), a 514 Reproducing Punch, Sorter, Collator, and Key Punches. There were also 5 major applications done "out of house" by various service bureaus because of tab equipment limitations.

3.) As of 1 March, 1968, all applications were running on the system with the exception of the Master Gross Sales Report which will be described later.

It is my ambition to get everything running smoothly and as soon as possible, then by June of next year to revise systems and programs into final state. By final state, if I may create a new word, I mean the "disk-ilation" of every possible application.

I welcome any remarks, suggestions, constructive criticisms and discovery of any "bugs" uncovered either in systems and/or programming area.

4.) The entire operation at Beloit Tool Corporation centers around a Model 2B, 8K 1130, Model 6 1442 Card I/O Unit, a Model 1 1132 Printer, and single disk system.
Computer rental totals $1594./Mo
EAM rental totals $412./Mo.

5.) A very note-worthy fact is the increase of time utilization after installation of the system: (Metered Hours).

	November	December	January
1130	69.32	110.11	134.31
1132	58.94	92.03	119.56
1442	68.01	101.23	129.22

D. D. Christians

TABLE OF PROGRAMS:

1.) Prepare Shipping Orders
2.) Invoicing
3.) Statements
4.) Overdue Orders & Backlog
5.) Daily Sales & Cash Receipts
6.) Invoiced Items Listing (4 card run)
7.) Product Class Report
8.) Product Detail Listing (3 card run)
9.) Controllers Trial Balance
10.) Sales Commissions
11.) Gross Sales Report
12.) Demonstration - Tour
13.) Master Mailing - Gross Sales File Edit
14.) Inventory

MISCELLANEOUS:

1.) 80/80 List
2.) Payroll
3.) Operator Efficiency Study
4.) Optional List & Count
5.) Reproducer & Sorter Simulator
6.) Prepare W-2 Forms
7.) Prepare FICA 941 Report
8.) 2-Up Mailing Labels
9.) Master Gross Sales Mailing Labels

SYSTEM DESCRIPTION

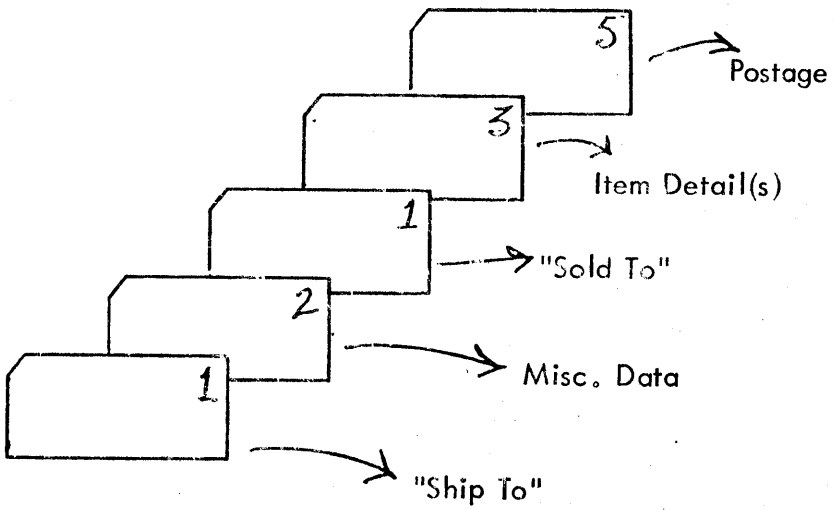
1. The series of applications about to be described is a "package" that could be adapted, with little or no modification, to any business.

   The strong points I feel to be of great interest are:

   a. All programs are in Fortran source.
   b. Use of commercial subroutines for overlap of I/O and ease of programming.
   c. Complete business "package" with numerous utility programs.
   d. Not tied to disk in concept.
   e. Very flexible A/R accounting.

2. The system begins with the receipt of a handwritten order, a teletype order, or a typed direct order from other plants. The orders are coded with a part number and then given to the Data Processing Center.

   Sold to and Ship to addresses are hand pulled from two tub files and form the order set. The key punch operator inserts the miscellaneous Data #2 card and #5 postage card as well as punching the order number in all cards. The order set is as follows:



   Names and addresses can consist of either 1 or 2 #1 cards giving the possibility of a four line address. "Sold to" address is optional. It will print the same as the "Ship to" if missing. There is no limit to the number of #3 item detail cards. The order sets are sorted on order number and entered to the computer for shipping orders to be run.

   After the shipping copy returns from the Shipping Department with quantity

shipped and postage changes written on it, the orders are hand pulled from the "Open Order" file, information punched, and entered to the computer for invoices to be run.

The invoicing run yields, besides the invoices, balance totals on gross, net, A/R, postage and IUT. It also produces A/R Summary cards and control listing as well as a #4 Summary Card file and control listing.

The A/R cards are merged twice weekly into the "A/R Open Order File" held by the Accounting Department. As payments are received the cards are relieved from the A/R file. Immediately after the last invoice for the month is run and A/R cards are merged, statements can be run. The statement program is at full printer speed and yields an aging breakdown as well as aged overall final totals.

The #4 Summary cards are kept for monthly balancing, product class report, commissions and gross sales reports, and quarterly inventory run. I might mention here that the weekly, as well as monthly balancing is done by means of the "Optional List-Count" program. This program will count any field as determined by the user and list or not list [redacted] concurrently.

The monthly commission report, as well as printed output and summary of totals, yields a monthly product summary card for each distributor. This card is merged with the Master Gross Sales Data and History File and used to list the Monthly Gross Sales Reprt. These two applications are by far the largest reports volume-wise and both operate at maximum printer speed.

3. Some other system programs of interest are:

   a. "Overdue Orders & Backlog": Scans the shipping order open order file and either extends and prints month-end backlog or orders overdue past a certain flexible date.

   b. "Daily Sales & Cash Receipts": Produces listing for payment record of the A/R file. Gang punches the date paid into the cards for future payment history records.

   c. "Product Class Report": Yields totals of sales by a 2-digit product class code. Also lists cause reports as well as totals of credits issued during the month.

   d. "Sales Commissions": Figures monthly commissions and detail listing of sales by distributor within state within territory. Summarizes sales by states and territories and prints summary listings of each. Produces summary card for each distributor and distributes sales by product class code.

e. "Gross Sales Report": (Program currently being written)..... Provides field salesmen and representatives with sales information from 1963 to present month by distributor. Also list company name and address along with 2 contact names. Provides summary sales information by distributor, states and territories.

f. "Demonstration-Tour:" Mainline and 5 option sub-programs to provide effective and fast demo to any interested people. Fifth option prints well known pictures of "Edith".

g. "Inventory": A set of two main programs, file maintenance and computation, to provide user with reorder points, economic order quantities, extensions and full report on a scientific basis.

h. "Payroll": Two Mainline and three sub-programs to compute payroll, print check, distribute amounts to accounts and punch a check reconciliation card. Routines include 4 different state routines as well as a city tax. Also includes profit sharing and salary bonus plans.

I. "Operator Efficiency Study": Determines shop machine operator efficiency using variables such as set-up times costs and time, pieces produced and classification of operations and machines. Output used to help determine employee merit rating.

j. "Optional List & Count": A extremely useful program to total on a variable field and either list or not list concurrently.

k. "Reproducer-Sorter-Collator Simulator": Provides the following operation at overlapped I/O speeds:

1. Move data from one field to another.
2. Gang punch.
3. Number cards sequentially (any starting value and increment).
4. 80/80 reproduce.
5. Multiply and extend any two fields.
6. Sort out any punch in any one column (stacker select).
7. Sequence check any field.

l. "Master Gross Sales Mailing Labels": Print mailing labels from Master Sales file working with the following types of cards:

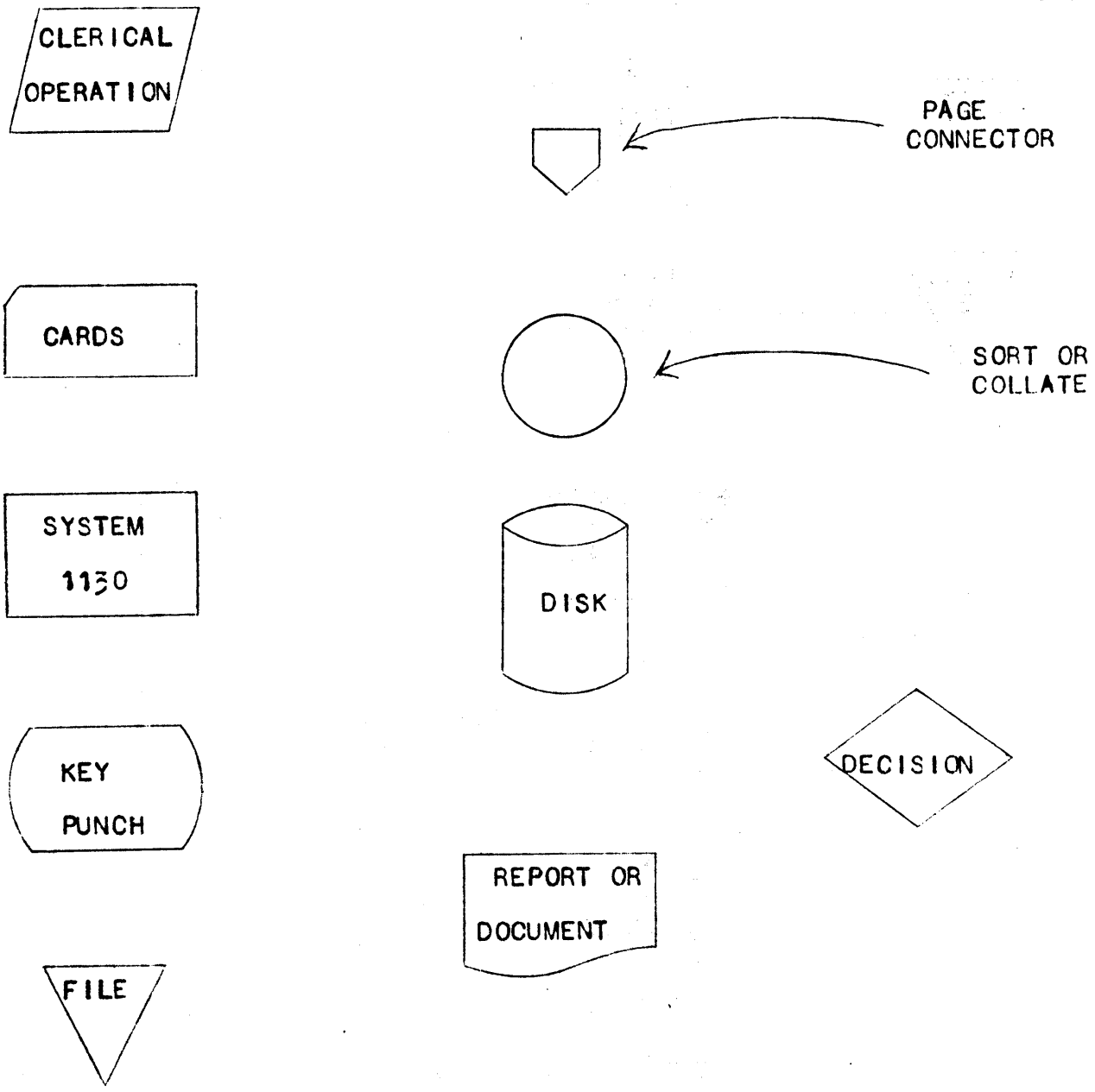
1. Company name and addresses.
2. Contact cards.
3. Company employee cards.

Program provides to either print company address or people home address,

which ever is present, follow postal regulations and places Zip Code on last line indented 2 spaces.

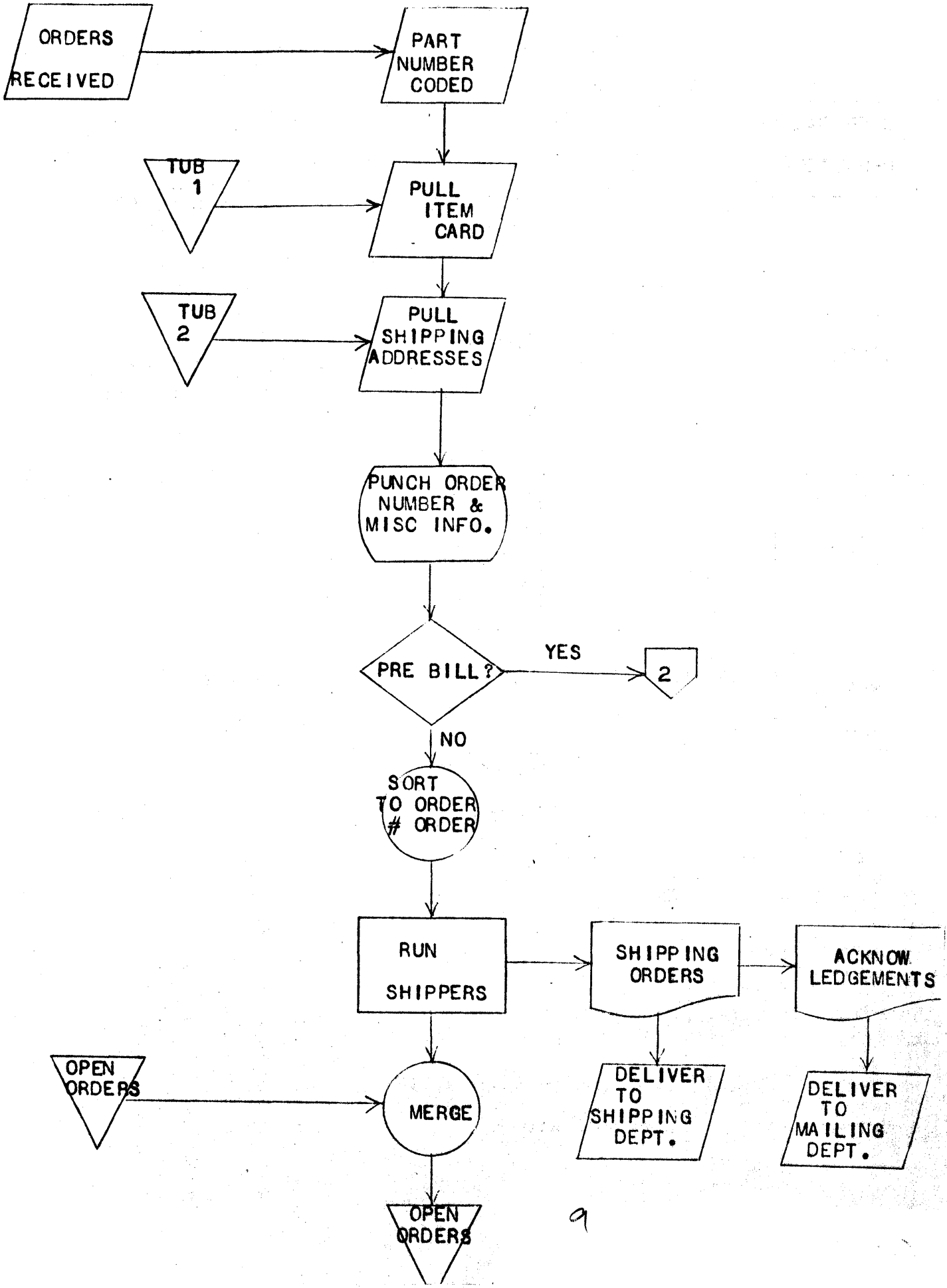
4. All programs are in Fortran Source. IBM commercial subroutines are utilized. Reference H20-0241-2- "1130 Commercial Subroutine Package (1130-SE-25X), Version 2 Program Reference Manual."

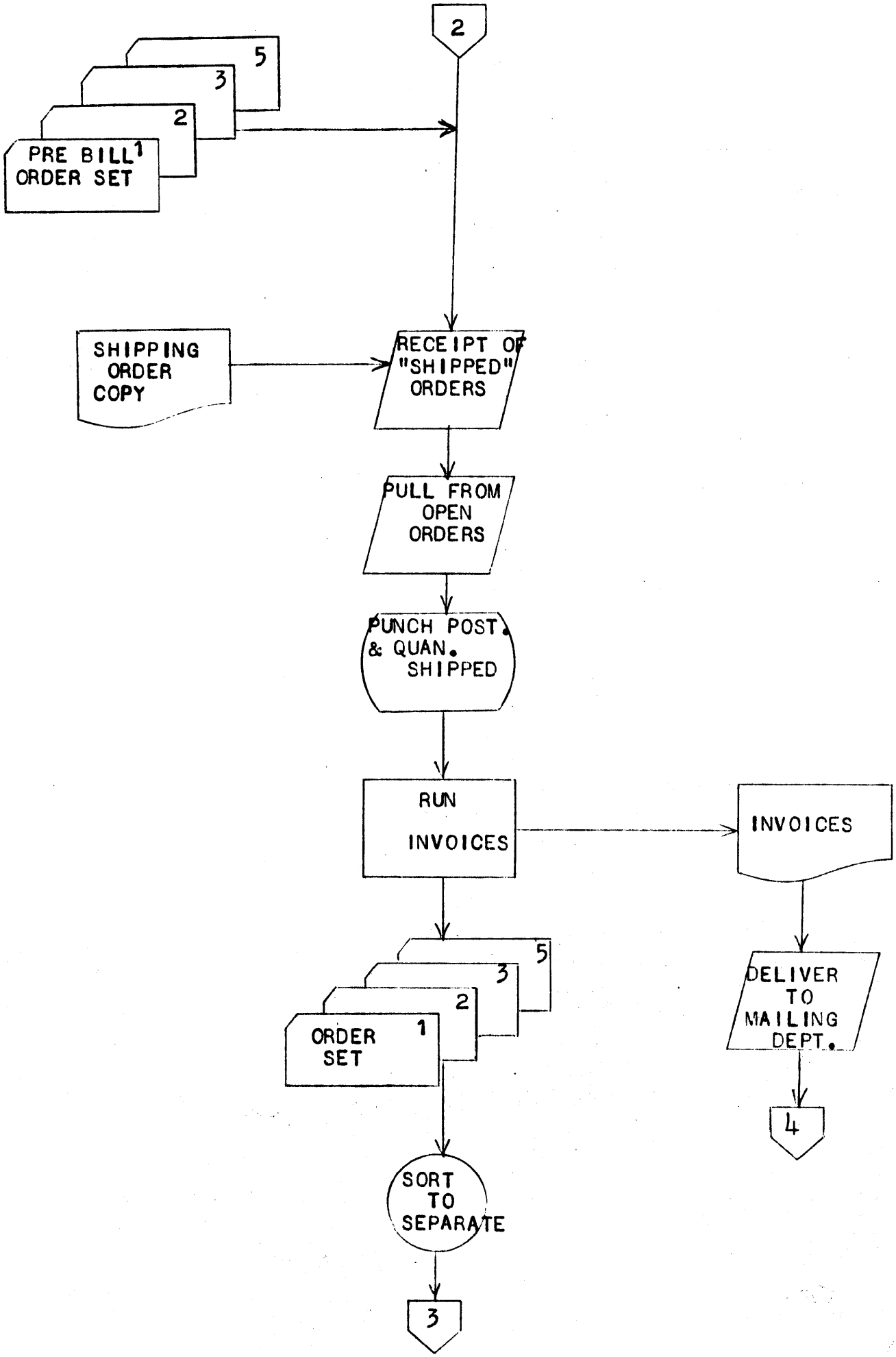
5. Beloit Tool Corporation will also be installing an IBM 633 Billing System in one of their Subsidiary companys, The Durst Corporation. They plan on using the card output for entry into our 1130 for application such as inventory, monthly statements, sales analysis and management reports.

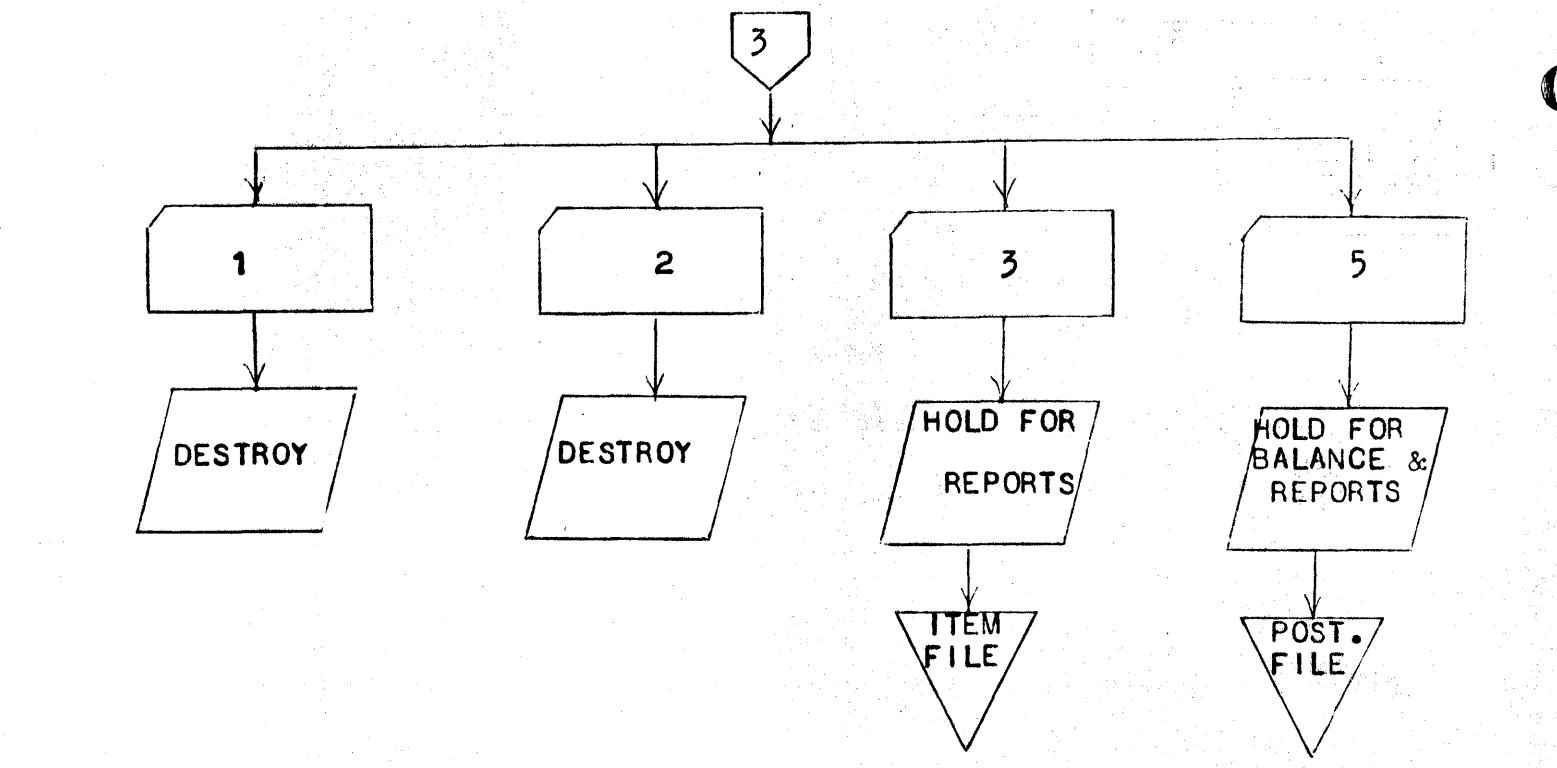


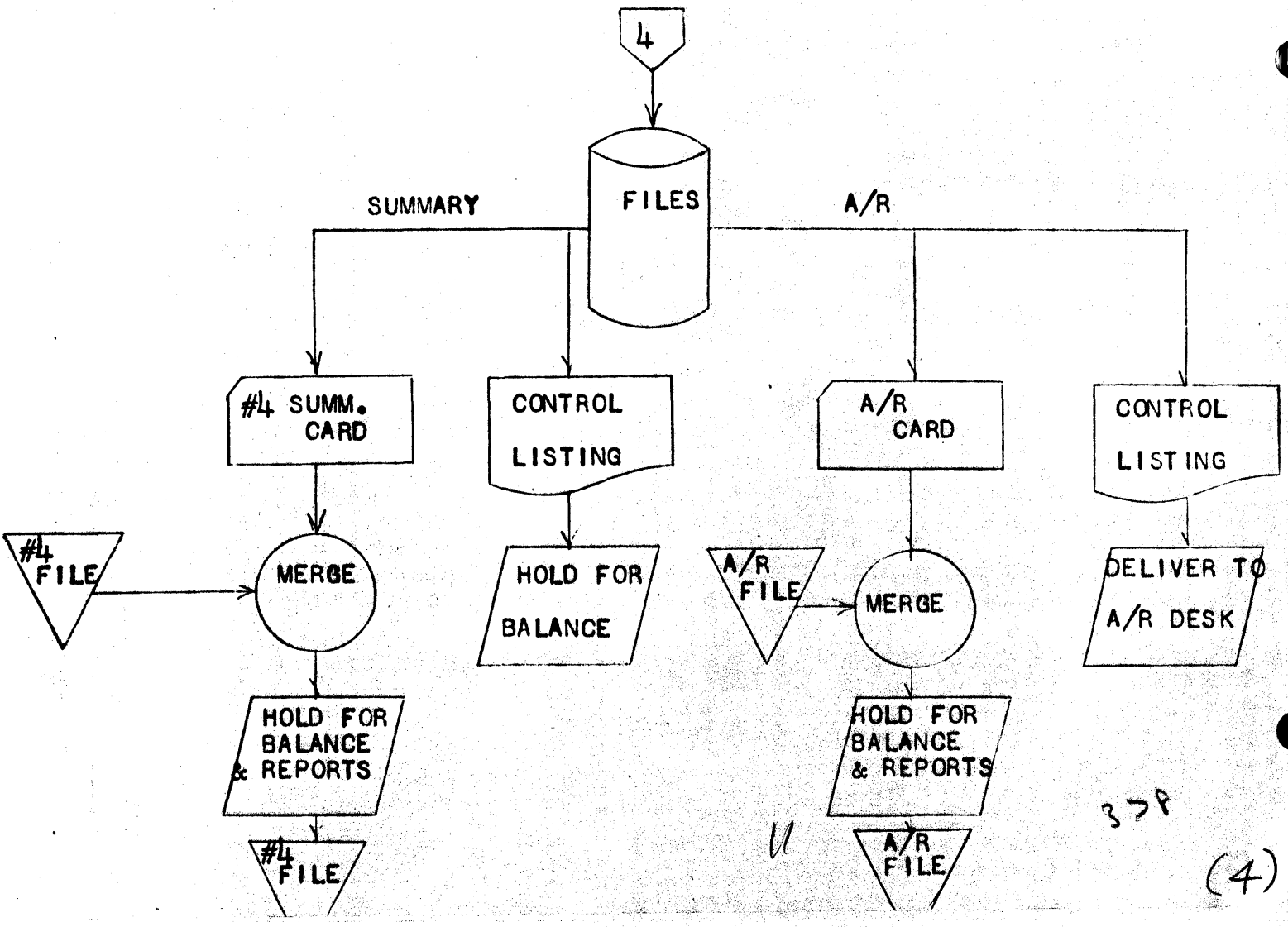
BELOIT TOOL CORPORATION SYSTEM

FLOWCHART SYMBOLS.


ORDERS RECEIVED
PART NUMBER CODED
TUB 1
PULL ITEM CARD
TUB 2
PULL SHIPPING ADDRESSES
PUNCH ORDER NUMBER & MISC INFO.
PRE BILL?
YES
2
NO
SORT TO ORDER # ORDER
RUN SHIPPERS
SHIPPING ORDERS
ACKNOW LEDGEMENTS
OPEN ORDERS
MERGE
DELIVER TO SHIPPING DEPT.
DELIVER TO MAILING DEPT.
OPEN ORDERS


2
5
3
2
PRE BILL 1
ORDER SET
SHIPPING
ORDER
COPY
RECEIPT OF
"SHIPPED"
ORDERS
PULL FROM
OPEN
ORDERS
PUNCH POST.
& QUAN.
SHIPPED
RUN
INVOICES
INVOICES
5
3
2
ORDER
SET 1
DELIVER
TO
MAILING
DEPT.
4
SORT
TO
SEPARATE
3


3
1
2
3
5
DESTROY
DESTROY
HOLD FOR REPORTS
HOLD FOR BALANCE & REPORTS
ITEM FILE
POST. FILE


4
FILES
SUMMARY
A/R
#4 SUMM. CARD
CONTROL LISTING
A/R CARD
CONTROL LISTING
#4 FILE
MERGE
HOLD FOR BALANCE
A/R FILE
MERGE
DELIVER TO A/R DESK
HOLD FOR BALANCE & REPORTS
HOLD FOR BALANCE & REPORTS
#4 FILE
A/R FILE

C S P E X T R A S

David A. Dunsmore

COMMON # 3428

Presented in Chicago

April 9, 1968

Session TUE-A3

CSP EXTRAS

David A. Dunsmore
COMMON # 3428

My presentation of five called subroutines, one functional subroutine and one program will be restricted to a synopsis or abstract of each of these programs because of time restrictions during this session. I will assume that all attendees have a general knowledge of FORTRAN, ASSEMBLER and CSP.

1. Subroutine DPUT (JCARD, J, JLAST, VAR, ADJST, N) causes a nine digit or less real variable to be placed in a ten digit or less output area with a decimal placed "N" positions to the left of the rightmost significant digit and logically suppresses leading zeros.

   The general use is the same as the CSP "PUT" subroutine except the "N" must be negative or the routine will act as through "PUT" had been called instead of "DPUT".

   This routine first fills the users defined output field with blanks, then computes the location of the decimal and places the decimal in the users field. Then leading zeros are suppressed and the remaining digits are "Edited" into the area defined by the user.

2. Subroutine JPUT (JCARD, J, JLAST, IAR, ADJST, N) causes an integer variable (IAR) to be placed in the users defined output area. This subroutine is identical to the "PUT" subroutine except for the fact that "PUT" handles real variables decimally truncated and "JPUT" handles integer variables.

3. Subroutine IZSUP (JCARD, J, JLAST) causes leading zeros in the defined users field to be suppressed. The basic advantage of this routine is that an edit mask is not required as would be the case if the CSP "EDIT" routine were used.

4. Subroutine READT (JCARD, J, JLAST, NER) causes paper tape records of variable length, but not exceeding 80 characters other than case shifts, deletes and new line (NL) to be read in the same manner that one would read a card of 80 columns or less. However, conversions are not overlapped because of some inherent difficulties with the library functions of both PAPTI and PAPTN. It is my belief that routines similar to "SPEED" will be required for paper tape if overlap during reading, or punching for that matter, is to be achieved.

   The general advantage of this routine is that users can read paper tape while reading cards or typing on the console keyboard and printer using other CSP routines. Current restrictions of FORTRAN READ or WRITE preclude this.

5. Function LEAP (YEAR) - YEAR is an integer variable. This functional subroutine determines if the current year is a leap year. If so, LEAP is set to 1, if not, LEAP is set to 2.

This routine requires the use of the following CSP or user submitted subroutines:

DPUT, FILL and NCOMP

General use of the routine might be:

```
      INTEGER YEAR
           .
           .
           .
      K = LEAP (YEAR)
      GO TO (10, 15), K
10    ROUTINE IF LEAP YEAR
           .
           .
           .
15    ROUTINE IF NOT LEAP YEAR
```

6. Subroutine PRINN (JCARD, J, JLAST, NERR) This subroutine is similar to the CSP "PRINT" subroutine and may be used in conjunction with "PRINT". The routine does not contain a "SKIP" subroutine. The basic advantage of this routine over the CSP "CALL PRINT" is that data listing may be performed at 120 lines/minute instead of 80 lines/minute. The data must not be alphabetic (A - Z) or, if it is alphabetic these characters will not be printed. Only digits zero thru nine and special printer characters such as COMMAS, DECIMALS, ASTERISKS, etc. will be printed. The basic advantage is the 50% increase in output speed to the 1132 Printer for numeric and special character printing.

This routine is used in the same way and in any place that the CSP "PRINT" subroutine is used. Existing user programs which print only numeric and/or special characters need only change the source program from the current "CALL PRINT" to "CALL PRINN".

7. "CDPRT" - Program to minimize I/O time from card to printer.

This program "CDPRT" will read cards from the 1442 card I/O and list the card image on the 1132 printer at either 80 or 120 lines/minute depending on the type of characters to be printed. If any characters of the alphabet (A thru Z) are present on the card to be listed, the listing takes place at 80 lines/minute. If no such characters exist, then the cards are listed at 120 lines/minute.

The program exits to Monitor after the last card has been read and printed.

No checking is done for Monitor Control Records or any channels detected while printing. Thus any card may be listed but no page skipping is attempted. The programming concepts employed in this program could be used by a user to write his own optimal print subroutine for any print job required in any CSP program.

After compilation and storage on the 1130's disk, the user need only use an "XEQ CDPRT" Monitor Control Card to list the deck of cards desired at the fastest print speed available on the 1132 printer. This program, as written, requires that the subroutine "PRINN" be available on the disk at execution time.

In summary, five CSP called subroutines, one functional subroutine and one program have been presented. A source listing with documentation has been included as an appendix to this presentation. Copies of the source listing and an abstract are available from this author. Inquiries should be sent to:

Ohio River Commission
414 Walnut Street
Cincinnati, Ohio 45202
Attention: David A. Dunsmore
COMMON User # 3428

```
// JOB
// ASM                                                                    CSP08190
* LIST (1132 PRINTER)                                                    CSP07390
* NAME PRINN                                                             (ID)
0041    17649555          ENT     PRINN      SUBROUTINE ENTRY POINT
                  * CALL PRINN (JCARD, J, JLAST, NERR3)
                  * PRINT JCARD(J) THROUGH JCARD(JLAST) ON THE           CSP08250
                  * 1132 PRINTER. PUT ERROR PARAMETER IN NERR3.          CSP08260
                  * ONLY PRINT NUMERIC
0000 0  0001      ONE   DC      1          CONSTANT OF 1                  CSP08300
0001 0  4000      SPACE DC      /4000
0002 0  0000      JCARD DC      0          JCARD J  ADDRESS               CSP08320
0003 0  0000      JLAST DC      0          JCARD JLAST  ADDRESS           CSP08330
0004    003D      AREA  BSS     61         WORD COUNT & PRINT AREA        CSP08340
0041 0  0000      PRINN DC      0          ADDRESS OF 1ST ARGUMENT
0042 20 176558F1  TEST  LIBF    PRNT1      CALL BUSY TEST ROUTINE         CSP08360
0043 0  0000            DC      /0000      BUSY TEST PARAMETER            CSP08370
0044 0  70FD            MDX     TEST       REPEAT TEST IF BUSY            CSP08380
0045 0  691A            STX   1 SAVE1&1    STORE IR1                      CSP08390
0046 01 65800041        LDX   I1 PRINN     LOAD 1ST ARGUMENT ADDRESS
0048 20 01647880        LIBF    ARGS       CALL ARGS ROUTINE              CSP08410
0049 1  0002            DC      JCARD      JCARD J  PICKED UP            CSP08420
004A 1  0003            DC      JLAST      JCARD JLAST  PICKED UP        CSP08430
004B 1  0004            DC      AREA       CHARACTER COUNT PICKED UP      CSP08440
004C 0  0078            DC      120        MAX CHARACTER COUNT            CSP08450
004D 0  C0B6            LD      AREA       GET CHARACTER COUNT            CSP08460
004E 0  80B1            A       ONE        HALF ADJUST                    CSP08470
004F 0  1801            SRA     1          DIVIDE BY TWO                  CSP08480
0050 0  D0B3            STO     AREA       STORE WORD COUNT               CSP08490
0051 0  C103            LD    1 3          GET ERROR WORD ADDRESS         CSP08500
0052 0  D012            STO     ERR&1      STORE IT IN ERROR ROUTINE      CSP08510
0053 20 195C10D2        LIBF    RPACK      CALL REVERSE PACK ROUTINE      CSP08520
0054 1  0002            DC      JCARD      JCARD J  ADDRESS               CSP08530
0055 1  0003            DC      JLAST      JCARD JLAST  ADDRESS           CSP08540
0056 1  0005            DC      AREA&1     PACK INTO I/O AREA             CSP08550
0057 20 176558F1        LIBF    PRNT1      CALL PRINT ROUTINE             CSP08560
0058 0  4000      WRITE DC      /4000      PRINT PARAMETER
0059 1  0004            DC      AREA       I/O AREA BUFFER                CSP08580
005A 1  0063            DC      ERROR      ERROR PARAMETER                CSP08590
005B 0  C0A5            LD      SPACE      LOAD PRINT WITH SPACE          CSP08600
005C 0  D0FB            STO     WRITE      STORE IN PRINT PARAMETER       CSP08610
005D 0  7104            MDX   1 4          INCREMENT OVER 4 ARGUMENTS     CSP08620
005E 0  6903            STX   1 DONE1&1    STORE IR1                      CSP08630
005F 00 65000000  SAVE1 LDX   L1 0         RELOAD OR RESTORE IR1          CSP08640
0061 00 4C000000  DONE1 BSC   L  0         RETURN TO CALLING PROGRAM      CSP08650
0063 0  0000      ERROR DC      0          RETURN ADDRESS GOES HERE       CSP08660
0064 00 D4000000  ERR   STO   L  0         STORE ACC IN ERROR PARAM       CSP08670
0066 0  1810            SRA     16         CLEAR ACC                      CSP08680
0067 01 4C800063        BSC   I  ERROR     RETURN TO PRNT1 PROGRAM        CSP08690
006A                    END                END OF PRINT SUBPROGRAM        CSP08840

     NO ERRORS IN ABOVE ASSEMBLY.
```

```
// JOB
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*NAME CDPRT
*IOCS(DISK)
      DIMENSION INPUT(80),IOUT(80)
C     NE IS ERROR PARAMETER FOR CARD READ ROUTINE (CSP)
      NE=-1
C     READ INPUT CARD (80 A1 - FORMAT)
    1 CALL READ(INPUT,1,80,NE)
C     MOVE INPUT RECORD TO OUTPUT AREA
      CALL MOVE(INPUT,1,80,IOUT,1)
C     CHECK ALL 80 COLUMNS OF CARD FOR ALPHA-NUMERIC CHARACTERS
      DO 2 I=1,80
C     LIST ALL NUMERIC AND BLANK CARDS AT 120 LINES/MIN
C     LIST ALL ALPHA-NUMERIC CARDS AT 80 LINES/MIN
      IF(IOUT(I)+4032)3,2,2
C     SKIP TO PRINT IF ANY ALPHA-NUMERIC
    2 CONTINUE
C     PRINT NUMERIC AT 120 LINES PER MINUTE
      CALL PRINN(IOUT,1,80,NER)
      GO TO 4
C     PRINT ALPHA-NUMERIC AT 80 LINES/MIN
    3 CALL PRINT(IOUT,1,80,NER)
CHECK FOR LAST CARD
    4 IF(NE)1,5,5
C     EXIT AFTER LAST CARD IS PRINTED
    5 CALL EXIT
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS
 IOCS

CORE REQUIREMENTS FOR CDPRT
 COMMON       0  VARIABLES    164  PROGRAM      66

END OF COMPILATION
```

```
// JOR
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
* NAME DPUT
      SUBROUTINE DPUT(JCARD,J,JLAST,VAR,ADJST,N)
      DIMENSION JCARD(10),JSAVE(9)
      IF(N)1,2,2
C     IF N IS NOT NEGATIVE, TREAT AS A REGULAR 'PUT'
    2 CALL PUT(JCARD,J,JLAST,VAR,ADJST,N)
      RETURN
C     HANDLE DECIMAL CONVERSION FROM FORTRAN TO CSP (A1-FORMAT)
    1 AN=-N
C     SHIFT DECIMAL TO EXTREME RIGHT END OF FIELD
      AVAR=VAR*(10.**AN)
C     PUT FIELD IN JSAVE WITHOUT DECIMAL
      CALL PUT(JSAVE,1,9,AVAR,0.5,0)
C     FILL USERS OUTPUT AREA WITH BLANKS
      CALL FILL (JCARD,J,JLAST,16448)
C     COMPUTE LOCATION OF DECIMAL FOR USER
      L=JLAST-AN
C     PUT DECIMAL IN USERS FIELD
      CALL FILL(JCARD,L,L,19264)
      L=9-AN
C     SUPRESS LEADING ZEROS
      DO3I=1,L
      IF(JSAVE(I)+4032)4,3,4
    3 CONTINUE
C     CHECK FOR NEGATIVE NUMBER
    4 IF(VAR)5,6,6
C     INSERT MINUS SIGN IF NEGATIVE
    5 CALL FILL(JCARD,J,J,24640)
C     PLACE VARIABLE (VAR) IN USERS OUTPUT AREA WITH DECIMAL
    6 CALL EDIT(JSAVE,I,9,JCARD,J,JLAST)
      RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR DPUT
 COMMON       0  VARIABLES     16  PROGRAM     150

END OF COMPILATION
```

```
// JOB
// FOR                                                                    CSP01810
*LIST SOURCE PROGRAM
*NAME JPUT
*ONE WORD INTEGERS                                                        CSP01840
      SUBROUTINE JPUT(JCARD,J,JLAST,IAR,ADJST,N)                          CSP0187
      DIMENSION JCARD(10)                                                 CSP01880
C-----PUT IAR INTO JCARD(J) THROUGH JCARD(JLAST).
C-----ADJST = A NUMBER TO HALF ADJUST THE VARIABLE VAR.                   CSP01900
C-----N = THE NUMBER OF POSITIONS THE DECIMAL POINT SHOULD BE MOVED LEFT  CSP01910
      DIGS=     (IABS(IAR)+ADJST)                                         CSP01920
      IF(N)   3,3,1                                                       CSP01930
1     DO 2 JNOW=1,N                                                       CSP01940
2     DIGS=WHOLE(DIGS*0.1)                                                CSP01950
C-----PUT DIGITS IN FIELD                                                 CSP01960
3     JNOW=JLAST                                                          CSP01970
4     DIGT=WHOLE(DIGS*0.1)                                                CSP01980
      JTEST=IFIX(DIGS-10.0*DIGT)                                          CSP01982
   11 IF(JTEST-10)9,10,10                                                 CSP01985
   10 JTEST=JTEST-10                                                      CSP01988
      DIGT=DIGT+1.0                                                       CSP01991
      GO TO 11                                                            CSP01994
    9 JCARD(JNOW)=256*JTEST-4032                                          CSP01997
      DIGS=DIGT                                                           CSP02000
      IF(JNOW-J)    6,6,5                                                 CSP02010
5     JNOW=JNOW-1                                                         CSP02020
      GO TO 4                                                             CSP02030
C-----PUT 11 PUNCH OVER LOW ORDER DIGIT IF NEGATIVE.                      CSP02040
6     IF(IAR)  7,8,8                                                      CSP02050
7     CALL NZONE(JCARD,JLAST,2,JNOW)                                      CSP02060
8     RETURN                                                              CSP02070
      END                                                                 CSP02080

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR JPUT
 COMMON       0  VARIABLES     10  PROGRAM     158

END OF COMPILATION
```

```
// JOB
// FOR
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*NAME IZSUP
      SUBROUTINE IZSUP(JCARD,J,JLAST)
      DIMENSION JCARD(120)
C     SUPRESS LEADING ZEROS IN USERS DEFINED AREA
      DO 1 I = J,JLAST
      IF(JCARD(I)+4032)3,2,3
    2 CALL FILL(JCARD,I,I,16448)
    1 CONTINUE
    3 RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR IZSUP
 COMMON       0  VARIABLES       2  PROGRAM      44

END OF COMPILATION
```

```
// JOB
// ASM                                                                    CSP06510
*LIST
** READ AND PUNCH SUBROUTINES FOR 1130 CSP PAPER TAPE                (ID) CSP06520
* NAME READT                                                          (ID) CSP06530
00CC     19141123          ENT     READT      SUBROUTINE ENTRY POINT     CSP06550
                         * CALL READ (JCARD, J, JLAST, NERR1)             CSP06560
                         * READ COLUMNS FROM BEGINNING OF CARD INTO JCARD(J) CSP06570
                         * THROUGH JCARD(JLAST). PUT ERROR PARAMETER IN   CSP06580
                         * NERR1.                                         CSP06590
0125     179150E3          ENT     PUNCT      SUBROUTINE ENTRY POINT     CSP06600
                         * CALL PUNCH (JCARD, J, JLAST, NERR2)            CSP06610
                         * PUNCH JCARD(J) THROUGH JCARD(JLAST) INTO THE   CSP06620
                         * BEGINNING OF A CARD. PUT ERROR PARAMETER INTO  CSP06630
                         * NERR2.                                         CSP06640
0000 0   0000        JCARD DC      0          JCARD J  ADDRESS           CSP06650
0001     0051        AREA  BSS     81         I/O AREA BUFFER            CSP06660
0052     0079        AREA1 BSS     121
00CB 0   0000        FLAG  DC      0          ERROR INDICATOR            CSP06670
00CC 0   0000        READT DC      0          FIRST ARGUMENT ADDRESS     CSP06680
00CD 0   6936              STX   1 SAVE1&1    SAVE IR1                   CSP06690
00CE 01  658000CC          LDX   I1 READT     GET 1ST ARGUMENT ADDRESS   CSP06700
00D0 0   403D              BSI     SETUP      GO TO SETUP                CSP06710
00D1 20  170578F1          LIBF    PAPT1
00D2 0   1000              DC      /1000      READ                       CSP06730
00D3 1   0052              DC      AREA1      AREA PARAMETER             CSP06740
00D4 1   0107              DC      ERROR      ERROR PARAMETER            CSP06750
00D5 20  170578F1          LIBF    PAPT1
00D6 0   0000              DC      /0000      BUSY TEST PARAMETER        CSP06900
00D7 0   70FD              MDX     *-3        REPEAT IF BUSY
00D8 0   6914              STX   1 SAVE+1     SAVE XR1
00D9 0   6189              LDX   1 -119
00DA 01  C50000CA    LD    LD    L1 AREA1+120 SEARCH INPUT AREA
00DC 01  4C2800E4          BSC   L  FOUND,+Z  FOR NEW LINE CHARACTER
00DE 0   1008              SLA     8
00DF 01  4C2800E4          BSC   L  FOUND,+Z
00E1 0   7101              MDX   1 1
00E2 0   70F7              MDX     LD
00E3 0   61FF              LDX   1 -1
00E4 0   7101        FOUND MDX   1 1          STORE BLANKS BEYOND
00E5 01  6D000001          STX   L1 AREA
00E7 0   C03C              LD      BLANK      NEW LINE FOUND
00E8 01  D50000CA    STO   STO   L1 AREA1+120
00EA 0   7101              MDX   1 1
00EB 0   70FC              MDX     STO
00EC 00  65000000    SAVE  LDX   L1 0         RESTORE XR1
00EE 20  17057213          LIBF    PAPHL      CALL CONVERSION ROUTINE
00EF 0   0000              DC      /0000      P.T. CODE TO CARD CODE
00F0 1   0053              DC      AREA1+1    FROM AREA1+1
00F1 1   0002              DC      AREA&1     TO AREA+1
00F2 0   0000        CN1   DC      0
00F3 20  225C5144    CONVT LIBF    SPEED      CALL CONVERSION ROUTINE
00F4 0   0010              DC      /0010      CARD CODE TO EBCDIC        CSP06770
00F5 1   0002              DC      AREA+1     FROM AREA
00F6 0   0000        JLAS1 DC      0          TO JCARD JLAST             CSP06790
00F7 0   0000        CNT1  DC      0          CHARACTER COUNT            CSP06800
00F8 0   C0D2              LD      FLAG       ERROR INDICATOR            CSP06810
00F9 01  4C1800FE          BSC   L  FINAL,&-  ALL DONE IF ZERO           CSP06820
00FB 0   1810              SRA     16         CLEAR ACC                  CSP06830
00FC 0   D0CE              STO     FLAG       CLEAR THE INDICATOR        CSP06840
00FD 0   70F5              MDX     CONVT      CONVERT AGAIN              CSP06850
00FE 20  22989547    FINAL LIBF    SWING      REVERSE THE ARRAY          CSP06860
00FF 1   0000              DC      JCARD      FROM JCARD J               CSP06870
0100 1   00F6              DC      JLAS1      TO JCARD JLAST             CSP06880
```

```
0101 0  7104          TEST  MDX   1 4          INCREMENT 4 ARGUMENTS          CSP06920
0102 0  6903                STX   1 DONE&1     STORE IR1                      CSP06930
0103 00 65000000      SAVE1 LDX   L1 0         RESTORE IR1                    CSP06940
0105 00 4C000000      DONE  BSC   L  0         RETURN TO CALLING PROGRAM      CSP06950
0107 0  0000          ERROR DC          0      START OF ERROR ROUTINE         CSP06960
0108 00 D4000000      ERR   STO   L  0         STORE ACC IN ERROR WORD        CSP06970
010A 01 740100CB            MDX   L  FLAG,1    SET THE FLAG INDICATOR         CSP06980
010C 01 4C800107            BSC   I  ERROR     RETURN TO INTERRUPT PROGRM     CSP06990
010E 0  0000          SETUP DC          0      START OF SETUP ROUTINE         CSP07000
010F 20 01647880            LIBF      ARGS     CALL ARGS SUBPROGRAM           CSP07010
0110 1  0000                DC        JCARD    GET JCARD J  ADDRESS           CSP07020
0111 1  00F6                DC        JLAS1    GET JCARD JLAST  ADDRESS       CSP07030
0112 1  0001                DC        AREA     GET CHARACTER COUNT            CSP07040
0113 0  0050                DC        80       MAX CHARACTER COUNT            CSP07050
0114 0  C0E1                LD        JLAS1    DISTRIBUTE JCARD JLAST         CSP07060
0115 0  D019                STO       JLAS2    INTO JLAS2                     CSP07070
0116 01 C4000001            LD    L   AREA     DISTRIBUTE COUNT               CSP07080
0118 0  D0DE                STO       CNT1     INTO CNT1                      CSP07090
0119 0  D017                STO       CNT2     AND CNT2                       CSP07100
011A 0  D0D7                STO       CN1
011B 0  D01A                STO       CN2
011C 01 D4000052            STO   L   AREA1
011E 0  C103                LD    1 3        GET ERROR WORD ADDRESS         CSP07110
011F 0  D0E9                STO       ERR&1    STORE INSIDE ERROR ROUTINE     CSP07120
0120 0  1810                SRA       16       CLEAR ACC                      CSP07130
0121 0  D0A9                STO       FLAG     CLEAR ERROR INDICATOR          CSP07140
0122 01 4C80010E            BSC   I   SETUP    RETURN TO CALLING PROG         CSP07150
0124 0  1010          BLANK DC          /1010
0125 0  0000          PUNCT DC          0        PUNCH ROUTINE STARTS HERE
0126 0  69DD                STX   1 SAVE1&1    SAVE IR1                       CSP07170
0127 01 65800125            LDX   I1 PUNCT     LOAD 1ST ARGUMENT ADDRESS
0129 0  40E4                BSI       SETUP    GO TO SETUP ROUTINE            CSP07190
012A 20 22989547            LIBF      SWING    CALL REVERSE ARRAY             CSP07200
012B 1  0000                DC        JCARD    FROM JCARD J                   CSP07210
012C 1  00F6                DC        JLAS1    TO JCARD JLAST                 CSP07220
012D 20 225C5144            LIBF      SPEED    CALL CONVERSION ROUTINE        CSP07230
012E 0  0011                DC        /0011    FROM EBCDIC TO CARD CODE       CSP07240
012F 0  0000          JLAS2 DC          0        FROM JCARD JLAST               CSP07250
0130 1  0002                DC        AREA&1   TO THE I/O AREA BUFFER          CSP07260
0131 0  0000          CNT2  DC          0        CHARACTER COUNT                CSP07270
0132 20 17057213            LIBF      PAPHL    CALL CONVERSION ROUTINE
0133 0  0011                DC        /0011    FROM CARD CODE TO P.T.
0134 1  0002                DC        AREA+1   I/O AREA BUFFER
0135 1  0002                DC        AREA+1   I/O AREA BUFFER
0136 0  0000          CN2   DC          0
0137 20 170578F1            LIBF      PAPT1
0138 0  2000                DC        /2000    PUNCH                          CSP07290
0139 1  0001                DC        AREA     I/O AREA BUFFER                CSP07300
013A 1  0107                DC        ERROR    ERROR PARAMETER                CSP07310
013B 20 170578F1            LIBF      PAPT1
013C 0  0000                DC        0        TEST FOR BUSY
013D 0  70FD                MDX       *-3      YES, REPEAT TEST
013E 0  70C2                MDX       TEST     NO, EXIT FROM ROUTINE
0140                        END                END OF READ SUBPROGRAM         CSP07330
```

NO ERRORS IN ABOVE ASSEMBLY.

7 11

```
// JOB
// FOR
*LIST SOURCE PROGRAM
* ONE WORD INTEGERS
*NAME LEAP
      FUNCTION LEAP(YEAR)
      INTEGER YEAR,JSAVE(7),KCARD(2)
      A=YEAR/4.
      CALL DPUT(JSAVE,1,7,A,0.005,-2)
C     PUT 'A' IN JSAVE STRING WITH TWO CHARACTERS BEYOND DECIMAL POINT
C     IF THE TWO CHARACTERS BEYOND THE DECIMAL ARE ZERO, THE YEAR IS LEAP YEAR
      CALL FILL(KCARD,1,2,-4032)
C      FILL MASK WITH EBCDIC ZEROS (-4032)
      IF(NCOMP(JSAVE,6,7,KCARD,1))2,1,2
C     IF LEAP YEAR, SET LEAP=1   IF NOT LEAP=2
    1 LEAP=1
      GO TO 3
    2 LEAP=2
    3 RETURN
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS

CORE REQUIREMENTS FOR LEAP
 COMMON       0  VARIABLES      14  PROGRAM      68

END OF COMPILATION
```

8

12

390

D. GARDNER

A-3

FORTRAN Coded Sorting Procedures

There are two sorting routines of interest - QSORT and MSORT - that I would like to talk to you about today.

QSORT is the FORTRAN-coded version of an ascending order sorting algorithm written by D.A. Shell originally described in Communications of the ACM 2(1959), 30-32. The algorithm is currently available in Collected Algorithms from ACM as Algorithm 201, SHELLSORT.

QSORT has several capabilities built into it which can be useful:

1. A second vector (optional) can be carried along with the vector being sorted.
2. The main vector may be sorted in ascending or descending order.
3. The starting point for sorting in the main vector can be specified (usually position 1, but could be any position).

QSORT is only one of three very similar routines differing only in the mode of the vectors being sorted. The following table shows the mode of each routine:

Subroutine Name	Mode of Main Vector	Mode of Secondary Vector
QSORT	Integer	Integer
XSORT	Real	Real
ZSORT	Real	Integer

Sorting times on the 1130 for QSORT (and XSORT) are shown below for vectors of three types: a randomly order vector, a vector in order, and a vector in reverse order (in each case a secondary vector was not carried along).

Routine	Made of Vector Being Sorted	Size of Vector	Time (in sec) Random Order	In Order	Reverse Order
QSORT	Integer	2000	36	14	22
"	"	1000	15	6	10
"	"	500	7	2	4-5
"	"	100	<1	<1	<1
XSORT	Real	2000	81	28	48
"	"	1000	34	12	22
"	"	500	15	6	10
"	"	100	1	1	1

MSORT is a subroutine developed for multiple vector sorting i.e., when one wants to sort more than one vector from an inner to an outer sort (similar to multiple field sorting on a card sorter). MSORT calls two subroutines, one of which in turn calls QSORT.

MSORT is one of two similar routines differing only in the mode of the vectors involved in the sorting. The following table describes the routines involved:

Main Subroutine	Mode of Vectors Involved	Other Necessary Subroutines
MSORT	Integer	ORDER LSORT QSORT
FSORT	Real	FORDR GORDR ESORT XSORT

A simple practice problem illustrates how MSORT works. The sheet labeled 'original data' shows four integer vectors (plus a sequence number at the far right), three of which are to be sorted in the order 3/2/1. Vector 4 is simply to be carried along.

The sheet labeled 'sample problem' shows a listing of a simple program written to do the above task. It is an interesting program because it illustrates the three basic steps involved in the use of a multiple (or singly) sorting routine: (1) read data, store on disk and generate the location vector NLOC, (2) sort, once or many times, and (3) print the results in the prescribed order. As stated above the disk was used to store the data and was referenced each time one of the vectors was needed for sorting. Note however, that all sorting is done in core and that the information on disk is in exactly the same order after the sorting as it was before.

The sheet labeled 'output' shows the results of the sort.

Also included in the handout are listings of all the programs mentioned above plus an abstract of the routine MSORT.

ORIGINAL DATA

1	2	3	4
0	0	6	0
7	7	6	4
3	4	3	2
6	0	6	4
3	1	6	1
6	5	4	0
1	0	4	9
2	7	4	2
9	0	6	4
7	4	3	9
4	2	4	8
6	7	5	5
8	5	7	9
2	1	5	1
7	2	2	9
1	7	4	3
8	7	0	1
6	5	4	7
5	0	9	1
2	6	3	7
7	4	5	8
8	8	2	5
7	0	7	1
2	3	7	6
7	0	1	2
1	8	5	0
3	6	1	9
4	6	6	8
4	9	5	6
9	6	4	3
0	8	0	9
8	6	6	1
9	6	1	6
2	9	2	7
4	6	0	2
9	0	5	0
7	3	0	5
9	7	3	2
0	0	5	0

Simple Program

```
// FOR
*ONE WORD INTEGERS
*IOCS(CARD,DISK,1132 PRINTER)
      DIMENSION NDATA(40),NLOC(40),NWORK(40),NDISK(4)
      DEFINE FILE1(40,4,U,IV1)
C
C        READ DATA AND PLACE ON DISK
C
    5 DO 1 I=1,40
      READ(2,400) NDISK
  400 FORMAT(4I5)
      NLOC(I)=I
    1 WRITE(1'I) NDISK
C
C        SORT DATA VECTORS FROM INNER SORT(3RD VECTOR) TO OUTER(1ST VECTOR)
C        THE 4TH VECTOR IS A *CARRY ALONG* VECTOR
C
      M=3
      DO 3 K=1,M
      DO 2 I=1,40
      READ(1'I) NDISK
      L=M-K+1
    2 NDATA(I)=NDISK(L)
    3 CALL MSORT(NDATA,NLOC,NWORK,40,1)
C
C        WRITE DATA VECTORS IN SORTED ORDER
C
      WRITE(3,401)
  401 FORMAT(1H1)
      DO 4 I=1,40
      L=NLOC(I)
      READ(1'L) NDISK
    4 WRITE(3,402) NDISK
  402 FORMAT(1H ,4I5)
      PAUSE9999
      GO TO 5
      END
// XEQ
```

OUT PUT

1	2	3	4	SEQ. NR.
0	0	6	0	02
0	3	0	9	3
0	3	9	1	0
0	6	3	7	21
1	0	4	9	0
1	7	4	3	17
1	8	5	0	27
2	1	5	1	15
2	7	4	2	09
2	8	7	6	25
2	9	2	7	35
3	1	6	1	06
3	4	3	2	04
4	2	4	8	12
4	5	6	8	29
4	6	0	2	36
4	9	5	6	30
5	9	9	1	20
6	0	6	4	05
6	5	4	0	07
6	5	4	7	19
6	6	6	1	33
6	7	5	5	13
7	0	1	2	26
7	0	7	2	24
7	2	2	9	16
7	3	0	5	3
7	4	3	9	1
7	4	5	2	22
7	7	6	4	03
8	0	5	0	40
8	5	7	9	14
8	6	1	9	28
8	7	0	1	18
9	0	6	4	10
9	5	5	5	23
9	6	1	6	34
9	6	4	3	31
9	7	3	2	39
9	9	5	0	37

```
// DUP
*DELETE              QSORT
// FOR
*ONE WORD INTEGERS
C
C         SORT AN INTEGER VECTOR INTO ASCENDING OR DESCENDING ORDER
C         CARRYING ALONG A SECONDARY VECTOR IN A ONE-TO-ONE
C         CORRESPONDENCE (OPTIONAL)
C
      SUBROUTINE QSORT(NA,NB,N,N1,NUPDN,NSTRT)
C
C         NA=VECTOR TO BE SORTED
C
C         NB=SECONDARY VECTOR
C
C         N=NUMBER OF ELEMENTS TO BE SORTED
C
C         N1=0   IGNORE NB
C           =1   NB CARRIED ALONG WITH NA
C
C         NUPDN=0   SORT IN DESCENDING ORDER
C              =1   SORT IN ASCENDING ORDER
C
C         NSTRT=STARTING POINT IN NA FOR SORTING (USUALLY 1)
C
      DIMENSION NA(1),NB(1)
      NDISP=NSTRT-1
      DO 1 I=1,N,I
    1 M=2*I-1
   20 M=M/2
      IF(M) 30,40,30
   30 K=N-M
      J=1
   41 I=J
   49 L=I+M
      II=I+NDISP
      LL=L+NDISP
      IF(NUPDN) 90,90,10
   90 IF(NA(II)-NA(LL)) 50,60,60
   10 IF(NA(II)-NA(LL)) 60,60,50
   50 NAS=NA(II)
      NA(II)=NA(LL)
      NA(LL)=NAS
      IF(N1) 70,70,80
   80 NAS=NB(II)
      NB(II)=NB(LL)
      NB(LL)=NAS
   70 I=I-M
      IF(I-1) 60,49,49
   60 J=J+1
      IF(J-K) 41,41,20
   40 RETURN
      END
// DUP
*STORE      WS  UA  QSORT
```

```
// DUP
*DELETE              XSORT
// FOR
*ONE WORD INTEGERS
C
C         SORT A REAL VECTOR INTO ASCENDING OR DESCENDING ORDER
C         CARRYING ALONG A SECONDARY VECTOR IN A ONE-TO-ONE
C         CORRESPONDENCE (OPTIONAL)
C
      SUBROUTINE XSORT(NA,NB,N,N1,NUPDN,NSTRT)
C
C         NA=VECTOR TO BE SORTED
C
C         NB=SECONDARY VECTOR
C
C         N=NUMBER OF ELEMENTS TO BE SORTED
C
C         N1=0   IGNORE NB
C           =1   NB CARRIED ALONG WITH NA
C
C         NUPDN=0   SORT IN DESCENDING ORDER
C              =1   SORT IN ASCENDING ORDER
C
C         NSTRT=STARTING POINT IN NA FOR SORTING (USUALLY 1)
C
      REAL NA(1),NB(1),NAS
      NDISP=NSTRT-1
      DO 1 I=1,N,I
    1 M=2*I-1
   20 M=M/2
      IF(M) 30,40,30
   30 K=N-M
      J=1
   41 I=J
   49 L=I+M
      II=I+NDISP
      LL=L+NDISP
      IF(NUPDN) 90,90,10
   90 IF(NA(II)-NA(LL)) 50,60,60
   10 IF(NA(II)-NA(LL)) 60,60,50
   50 NAS=NA(II)
      NA(II)=NA(LL)
      NA(LL)=NAS
      IF(N1) 70,70,80
   80 NAS=NB(II)
      NB(II)=NB(LL)
      NB(LL)=NAS
   70 I=I-M
      IF(I-1) 60,49,49
   60 J=J+1
      IF(J-K) 41,41,20
   40 RETURN
      END
// DUP
*STORE       WS  UA  XSORT
```

```
// DUP
*DELETE              ZSORT
// FOR
*ONE WORD INTEGERS
C
C         SORT A REAL VECTOR INTO ASCENDING OR DESCENDING ORDER
C         CARRYING ALONG A SECONDARY INTEGER VECTOR IN A ONE-TO-ONE
C         CORRESPONDENCE (OPTIONAL)
C
      SUBROUTINE ZSORT(NA,NB,N,N1,NUPDN,NSTRT)
C
C         NA=VECTOR TO BE SORTED(REAL)
C
C         NB=SECONDARY VECTOR
C
C         N=NUMBER OF ELEMENTS TO BE SORTED
C
C         N1=0  IGNORE NB
C           =1  NB CARRIED ALONG WITH NA
C
C         NUPDN=0  SORT IN DESCENDING ORDER
C              =1  SORT IN ASCENDING ORDER
C
C         NSTRT=STARTING POINT IN NA FOR SORTING (USUALLY 1)
C
      REAL NA(1)
      DIMENSION NB(1)
      NDISP=NSTRT-1
      DO 1 I=1,N,I
    1 M=2*I-1
   20 M=M/2
      IF(M) 30,40,30
   30 K=N-M
      J=1
   41 I=J
   49 L=I+M
      II=I+NDISP
      LL=L+NDISP
      IF(NUPDN) 90,90,10
   90 IF(NA(II)-NA(LL)) 50,60,60
   10 IF(NA(II)-NA(LL)) 60,60,50
   50 FAS=NA(II)
      NA(II)=NA(LL)
      NA(LL)=FAS
      IF(N1) 70,70,80
   80 NAS=NB(II)
      NB(II)=NB(LL)
      NB(LL)=NAS
   70 I=I-M
      IF(I-1) 60,49,49
   60 J=J+1
      IF(J-K) 41,41,20
   40 RETURN
      END
// DUP
*STORE       WS  UA  ZSORT
```

Subroutine MSORT

Purpose:

To sort an integer vector of numbers and produce the location (pointer) vector indicating the sort order. This subroutine would be used for multiple field sorting.

Usage:

```
Call  MSORT(NDATA,NLOC,NWORK,K,NUPDN)
```

Description of Parameters:

input	- NDATA =	vector of input data to be sorted
input/output	- NLOC =	location vector which must be initialized to the integers 1-K when calling MSORT for the first time. This vector is never reset and needs only to be initialized. After multiple sorts have been made, NLOC shows the index of the original records in the order according to the fields sorted.
input	- NWORK =	work vector of length K
input	- K =	length of NDATA,NLOC, and NWORK
input	- NUPDN =	0 descending order sort 1 ascending order sort

Remarks:

When using MSORT,NDATA is the data vector presented at any one time. When multiple sorting, the data representing the inner or minor sort must be presented first to MSORT, the next data vector in the sorting sequence etc. until the last data vector which represents the outer or major sort.

The following calls to MSORT would be used to sort in ascending order a record according to three data vectors NDAT1, NDAT2, and NDAT3 where NDAT3 indicates the inner sort and NDAT1, the outer. (Remember NLOC is set to the integers 1-K prior to calling MSORT the first time)

```
      DO 1  i=1,K
1     NDATA(i)=NDAT3(i)
      Call MSORT(NDATA,NLOC,NWORK,K,1)
      DO 2  i=1,K
2     NDATA(i)=NDAT2(i)
      Call MSORT(NDATA,NLOC,NWORK,K,1)
      DO3   i=1,K
3     NDATA(i)=NDAT1(i)
      Call MSORT(NDATA,NLOC,NWORK,K,1)
```

Subroutines Required:

ORDER, LSORT

Input-Output Devices Used:

None

Method:

Each data vector is re-ordered according to the location vector NLOC before being sorted into ascending or descending order. This has the effect of sorting a given field and carrying with it all the other associated fields in the record.

```
// DUP
*DELETE              MSORT
// FOR
*ONE WORD INTEGERS
      SUBROUTINE MSORT(NDATA,NLOC,NWORK,K,NUPDN)
      DIMENSION NDATA(1),NLOC(1),NWORK(1)
      CALL ORDER(NDATA,NLOC,NWORK,K)
      DO 1 I=1,K
    1 NWORK(I)=I
      CALL LSORT(NDATA,NWORK,K,NUPDN)
      CALL ORDER(NLOC,NWORK,NDATA,K)
      RETURN
      END
// DUP
*STORE       WS  UA  MSORT

// DUP
*DELETE              ORDER
// FOR
*ONE WORD INTEGERS
      SUBROUTINE ORDER(NA,NZ,NK,N)
      DIMENSION NA(1),NZ(1),NK(1)
      DO 1 I=1,N
      K=NZ(I)
    1 NK(I)=NA(K)
      DO 2 I=1,N
    2 NA(I)=NK(I)
      RETURN
      END
// DUP
*STORE       WS  UA  ORDER

// DUP
*DELETE              LSORT
// FOR
*ONE WORD INTEGERS
      SUBROUTINE LSORT(NA,NZ,N,NUPDN)
      DIMENSION NA(1),NZ(1)
      CALL QSORT(NA,NZ,N,1,NUPDN,1)
      KSTRT=0
      DO 1 I=2,N
      IF(NA(I)-NA(I-1)) 2,3,2
    2 IF(KSTRT) 1,1,4
    4 K=I-KSTRT
      CALL QSORT(NZ,NZ,K,0,1,KSTRT)
      KSTRT=0
      GO TO 1
    3 IF(KSTRT) 5,5,1
    5 KSTRT=I-1
    1 CONTINUE
      IF(KSTRT) 6,6,7
    7 K=N-KSTRT+1
      CALL QSORT(NZ,NZ,K,0,1,KSTRT)
    6 RETURN
      END
// DUP
*STORE       WS  UA  LSORT
```

```
// DUP
*DELETE                FSORT
// FOR
*ONE WORD INTEGERS
      SUBROUTINE FSORT(NDATA,NLOC,NWORK,K,NUPDN)
      REAL NDATA(1),NWORK(1)
      DIMENSION NLOC(1)
      CALL FORDR(NDATA,NLOC,NWORK,K)
      DO 1 I=1,K
    1 NWORK(I)=I
      CALL ESORT(NDATA,NWORK,K,NUPDN)
      CALL GORDR(NLOC,NWORK,NDATA,K)
      RETURN
      END
// DUP
*STORE       WS  UA  FSORT
```

```
// DUP
*DELETE                FORDR
// FOR
*ONE WORD INTEGERS
      SUBROUTINE FORDR(FA,NZ,FK,N)
      DIMENSION FA(1),NZ(1),FK(1)
      DO 1 I=1,N
      K=NZ(I)
    1 FK(I)=FA(K)
      DO 2 I=1,N
    2 FA(I)=FK(I)
      RETURN
      END
// DUP
*STORE       WS  UA  FORDR
```

```
// DUP
*DELETE                ESORT
// FOR
*ONE WORD INTEGERS
      SUBROUTINE ESORT(NA,NZ,N,NUPDN)
      REAL NA(1),NZ(1)
      CALL XSORT(NA,NZ,N,1,NUPDN,1)
      KSTRT=0
      DO 1 I=2,N
      IF(NA(I)-NA(I-1)) 2,3,2
    2 IF(KSTRT) 1,1,4
    4 K=I-KSTRT
      CALL XSORT(NZ,NZ,K,0,1,KSTRT)
      KSTRT=0
      GO TO 1
    3 IF(KSTRT) 5,5,1
    5 KSTRT=I-1
    1 CONTINUE
      IF(KSTRT) 6,6,7
    7 K=N-KSTRT+1
      CALL XSORT(NZ,NZ,K,0,1,KSTRT)
    6 RETURN
      END
// DUP
*STORE       WS  UA  ESORT
```

```
// DUP
*DELETE                GORDR
// FOR
*ONE WORD INTEGERS
      SUBROUTINE GORDR(NZ,FA,FK,N)
      DIMENSION NZ(1),FA(1),FK(1)
      DO 1 I=1,N
      K=FA(I)
    1 FK(I)=NZ(K)
      DO 2 I=1,N
    2 NZ(I)=FK(I)
      RETURN
      END
// DUP
*STORE       WS  UA  GORDR
```

SESSION REPORT

COMMON - Chicago

Session Number TUE A4 Session Name Control of Programming and Operations Costs

Chairman P. A. Bickford

Time 8:30 to 10:00 AM Attendance (No.)

Speakers

Synopsis of Meeting Two alternative methods of budgeting and accounting for systems, programming, and operating costs were outlined. Examples were given to show different management decisions arising from the application of these methods.

TUEAY

BUDGETING EDP COSTS *

INTRODUCTION

There are many reasons why companies install computing equipment. Some are valid - others are not.

In any case, the story often begins when a company decides to install a computer to handle expanding research, engineering and/or business oriented applications.

An individual is selected to manage the use of such equipment for one of two reasons:

1). He may have been the manager of a unit record installation,

or 2). He had some technical background associated with computer programming.

Seldom is any consideration given to the management background of the individual charged with the responsibility of utilizing the new computer. He soon finds himself subjected to rapidly increasing pressure:

1). He receives little management direction. Seldom is anyone in "Top Management" willing to become entwined with a new mysterious adventure. They don't understand it so they vote to go along with it.

2). He is so pre-occupied with learning how to make the new equipment work

*Presented at the COMMON USERS GROUP meeting in Chicago, Illinois, April 10, 1968.

that he devoted little or no time to real management problems.

3). When manufacturer proposals are considered, it is difficult for him to question promises of fantastic savings made to top management. After all, the potential importance of this new position looks far too rewarding.

4). The establishment of a new section, department or division with the prerogative to cross department lines does not win friends among other department heads. They begin to fear their own position as well as impending changes.

To top it all off, the top management of the company decides to handle the cost of this new effort the same way other costs have always been handled. The objective of this paper is to examine the different ways this has been done and their implications.

ALTERNATIVE METHODS

The method which should be used in an organization depends upon:

A. Organization structure.

B. Direct Top Management involvement in the review of possible areas of application.

C. The extent to which installed equipment is being used.

Each alternative will be discussed separately.

ALL COSTS CONSIDERED AS GENERAL ADMINISTRATIVE OVERHEAD

Here all costs associated with the computing activity are distributed as all other corporate, management

and staff costs. Usually, little effort is devoted to the development of direct costs by job. In this environment, the value of processing different projects is not questioned or evaluated.

Where the objective of the computing activity is to process large volume financial, inventory or Management Information Systems applications, this may be a satisfactory solution. However, where a variable mix of applications is being processed from several departments, this solution is not satisfactory. At this level of operation there is no comparison of actual costs with projected costs.

ALL VARIABLE COSTS CHARGED DIRECTLY - ALL FIXED COSTS CHARGED TO USING DEPARTMENTS ON A USE BASIS

The value of this approach to the problem depends upon the mix of applications being processed, the degree of computer use during different times of the year, and the top management support for the computing effort. This method is entirely satisfactory where the level of computer use is constant and there is top management support for the computing effort.

Where the level of computer use varies from period to period, any method for allocation of overhead costs is at best, approximate. Furthermore, if the main concern of top management is the control of costs within

cost centers, this method will result in cost center management using means other than the computing service for accomplishing major tasks. Under these conditions computing facilities will not be used to capacity and added corporate costs will be generated where cost center management options for major clerical efforts to accomplish major tasks.

Where a computing service is established at the corporate level, division and department heads may prefer to handle the processing of all tasks within departments where possible, to avoid:

1). Outside influence, questions or controls

or

2). Charges from a service department.

This can often happen where the comparative costs clearly favor the computing service.

VARIABLE COSTS CHARGED DIRECTLY - FIXED COSTS CHARGED AS GENERAL ADMINISTRATIVE OVERHEAD COSTS

Fixed costs are defined as those associated with the top administrator of the data processing effort and the basic costs of the computing equipment installed. Variable costs are those associated with all added personnel, supplies and maintenance required to complete processing for all applications.

The conditions where this method works best are obtained when:

1). The computer is not used to full capacity throughout the year.

2). Top management wants to maintain a cost center control and minimize total corporate costs.

and 3). The level of computer use varies during different times of the year.

Many organizations have major applications which require a minimum computer configuration. These organizations may use their equipment close to capacity during some periods of the year and less than capacity during the remaining periods. Under these conditions the equipment should be used for all applications where the increase in variable costs to do the job is less than the cost of accomplishing the job using alternative methods.

SESSION REPORT

COMMON - Chicago

Session Number TUE B1 Session Name 360 DOS

Chairman A. Ragsdale

Time 10:30 to 12:00 AM Attendance (No.) 118

Speakers 1) Mr. Bob White - IBM (DOS Version 3)

2) Mr. Gerry Kaplan - IBM (Comparison of FORTRAN to PL/1)

Synopsis of Meeting 1) Mr. Bob White of IBM announced the availability of DOS Version 3. He announced the following improvements in the DOS system: a) complete support of the 2314; b) simplified label handling; c) cylinder index in core for indexed sequential files (assembler language only); d) improved multiprogramming facilities; e) device independence for problem programs; f) libraries may be resident on other than SYSRES; g) disk initialize program now resident in DOS.

2) Mr. Gerry Kaplan of IBM presented the similarities between FORTRAN and PL/1. A good summation of the presentation may be found in the IBM Manual "A Guide to PL/1 for FORTRAN Users" - C20-1637.

DCS

PL/I AND FORTRAN: A COMPARISON

PART I STATEMENT SIMILARITY

PART II A SAMPLE PROGRAM

PART III CAPABILITIES OF PL/I BEYOND FORTRAN

PRESENTED BY
G. P KAPLAN
IBM WHITE PLAINS

SESSION TUES B1

PL/I AND FORTRAN: A COMPARISON

PART 1 - STATEMENT SIMILARITY

DATA DEFINITION

FORTRAN

```
DIMENSION A (50,50),  B(25,100),  C(2)
COMMON A
EQUIVALENCE (A, B)
DATA C/2*1.0/
INTEGER *2     A
REAL *8        B
COMPLEX        D
LOGICAL        E
```

PL/I

```
DECLARE A (50,50) BINARY (15,0) EXTERNAL,
        B(25,100) FLOAT BINARY (53) DEFINED A,
        C(2) INITIAL     ( (2) 1.0 ),
        D FLOAT BINARY COMPLEX,
        E BIT(1)
```

ASSIGNMENT

FORTRAN

```
A = B+C*SQRT(E)
```

PL/I

```
A = B+C*SQRT(E);
```

CONTROL STATEMENTS

FORTRAN

```
      GO TO 25

      GO TO (1,3,5), N

      N=2

      IF (X .EQ. Y .AND. Z .GT. Y)    A=B+13

      DO 100      I=1, 15,3

100 CONTINUE

      PAUSE 'END PHASE 1'
```

PL/I

```
      GO TO NEXT;

      GO TO L (I);

      I=3;

      IF X=Y & Z>Y   THEN A=B+13;

D100:DO        I=1 TO 15 BY 3;

      END D100

      DISPLAY ('END PHASE 1');
```

INPUT OUTPUT

FORTRAN

```
      READ (5,1)  X,Y,Z
    1 FORMAT  (F8 .2,2F4.1)
      WRITE  (6,2)
    2 FORMAT  (1HI,'HEADINGS')
      WRITE (8)  X,Y,Z
      ENDFILE  8
      REWIND  8
```

PL/I

```
GET EDIT(X,Y,Z)(F(8,2), F (5,1));
PUT LIST ('HEADING') PAGE;
WRITE FILE (SCRATCH) FROM (WORK)
DECLARE  1  WORK, 2X,2Y,2Z;
CLOSE FILE (SCRATCH);
```

SUBPROGRAM CONSTRUCTION

FORTRAN

```
CALL MATMPY  (A, B, C)

FUNCTION SPEC (A, B)

SUBROUTINE XTR (Y,Z)

ENTRY     XTRA(Q, R)

EXTERNAL     S1,S2, S3

RETURN  (A+B+C)
```

PL/I

```
CALL MATMPY  (A, B, C);

SPEC:  PROCEDURE  (A, B);

XTR:   PROCEDURE  (Y, Z);

XTRA:  ENTRY  (Q, R);

DECLARE   (S1, S2, S3) ENTRY;

RETURN  (A+B+C);
```

SAMPLE PROBLEM

QUADRATIC MODEL $A_0 + A_1 T + 1/2\ A_2\ T^2$

EXPONENTIALLY SMOOTHED COEFFICIENTS

READ IN DATA

UPDATE MODEL AND MAKE NEW FORCAST

PRINT NEW FORCAST

PUNCH UPDATED MODEL

PART II - COMPARISON OF SAMPLE PROGRAMS

FORTRAN MAIN PROGRAM

```
      DIMENSION ID(5)
      INTEGER *2    ID
      COMMON    OBS, PRJ, A0, A1, A2
      WRITE (6,100)
100   FORMAT (1H1)
      1  READ (5,101)  OBS, PRJ, A0, A1, A2, (ID(I),  I=1,15)
101   FORMAT  (F8.4, 4F10.4,15A2)
      CALL FORCAST
      WRITE  (6,102)  PRJ,  (ID(I), I=1, 15)
102   FORMAT  (1H,   F8.4, X1, I5A2)
      WRITE (7,103)  PRJ, A0, A1, A2, (ID(I),  I=1, 15)
103   FORMAT  (X8, 4 F10.4, I5A2)
      GO TO 1
      END
```

PL/I MAIN PROGRAM

```
SAMPL:  PROCEDURE OPTIONS (MAIN);
DECLARE     (OBS,PRJ,A0,A1,A2,ALPHA,BETA)
            FLOAT BINARY, ID CHARACTER (30) ) STATIC
            EXTERNAL;
ALPHA=.1;  BETA=.9;
PUT EDIT PAGE;
START: GET EDIT (OBS,PRJ,A0,A1,A2,ID)
            (F(8,4), 4(X(2), F(8,4) ), X(2), A(30) );
CALL FORCAST;
PUT EDIT (PRJ,ID) (F(8,4),X(2), A(30) ) SKIP (1);
PUT FILE (PUNCH) EDIT
            (PRJ,A0,A1,A2,ID)
            (X(8), 4(X(2),F(8,4) ),X(2),A(30) );
GO TO START;
END;
```

FORTRAN SUBPROGRAM

```
SUBROUTINE FORCAST
COMMON OBS,PRJ,A0,A1,A2,
REAL*4 ALPHA/.1/,BETA/.9/
ERR=PRJ-OBS
TEMP=A2-ALPHA**3*ERR
A1=A1+A2-1.5*ALPHA**2*(2.0-ALPHA)*ERR
A0=OBS+BETA**3*ERR
A2=TEMP
PRJ=A0+A1+.5*A2
RETURN
END
```

PL/I SUBPROGRAM

```
FORCAST: PROCEDURE;

ERR=PRJ-OBS;

TEMP=A2-ALPHA**3*ERR;

A1=A1+A2-1.5*ALPHA**2*(2-ALPHA)*ERR;

A0=OBS+BETA**3*ERR;

A2=TEMP;

PRJ=A0+A1+.5*A2;

RETURN;

END;
```

PART III

PL/I HAS SUPERIOR ARRAY HANDLING

EXAMPLES:

```
DO  I= 1 to 5, 7 to 10, 15, 17;

DO J = .8 to .5 by - .1

DO K = N+1 to P  by Q ** 3;

DO I = 1 to 50 (WHILE  ST ¬ ILL );

A = B + C/E + 5;
```

WHERE THE VARIABLES ARE ARRAYS

PL/I HAS CHARACTER AND BIT STRING PROCESSING

EXAMPLES

```
*     DECLARE (HEADACHE, FEVER) BIT (1):
      IF HEADACHE AND (¬ FEVER)
            THEN GO TO ASPIRIN;
            ELSE GO TO PENICILLIN;
*     DECLARE (X,A) CHARACTER (11),
      X = ' XYCOMABCMON';
      Y = INDEX (X, 'COM');
      Z = INDEX (X, 'MON');

      A = SUBSTR (X,Y,3) ||
                  SUBSTR (X,Z,3);
```

A WILL BE SET TO
COMMONbbbbb

PL/I HAS MORE EXTENSIVE

INPUT/OUTPUT FACILITIES

```
PUT SKIP EDIT (A,B,C) (F (10,2), F (8),
                  X (4), A (22));

PUT SKIP EDIT ('SIN (A)= b', SIN (A)) (A(8), F (10,6));

PUT SKIP EDIT (A,B)  (R ( FMT (I)));

FMT (1)...

FMT (2)...

PUT PAGE EDIT (A,B,C) ((K) F (N,M));

PUT SKIP FILE (OUT) LIST ('SQUARE ROOTS OF 100 INTEGERS',
     (X, SQRT (X), DO X + 1 to 100));

GET SKIP STRING (CHAR)
```

** AND ALL COBOL-LIKE FACILITIES VIA READ/WRITE

FROM/TO STRUCTURES

PL/I HAS MORE EXTENSIVE DATA EDITING

SOURCE	TARGET
00100	**100
10203	1 2 3
1234.56	1,234.56
12	1 2
001.23	$1.23
-123	$1.23CR
123	123^{+}
-123	123^{-}

PL/I HAS MORE BUILT IN FUNCTIONS

EXAMPLES

DATE	RETURNS CHARACTER STRING YYMMDD
TIME	RETURNS CHARACTER STRING HHMMSSTTT
SUM(X)	RETURNS SUM OF ALL ELEMENTS OF X
PROD(X)	RETURNS PRODUCT OF ALL ELEMENTS OF X

```
PL/I HAS POWER!!
*       TABLE (P) OF VALUES
        TO BE MODIFIED BASED ON PRESENT CONDITION

        P=P+ (.05 * P>1.0)
            + (.10 * P> 10.0)
```

IN ADDITION FULL PL/I OFFERS:

COMPILE TIME FACILITIES (MACROS)

LIST PROCESSING

MULTI TASKING

INTERRUPT CONTROL CAPABILITIES

SESSION REPORT

COMMON - Chicago

Session Number TUE B2 Session Name MPX I - Internals

Chairman R. P. Walker

Time 10:30 to 12:00 AM Attendance (No.) 91

Speakers Bruce Landeck - IBM

Paul Healey - IBM

Lex Arthur - IBM

Synopsis of Meeting B. Landeck introduced the speakers. Both speeches were tutorials. P. Healey spoke on Input/Output. L. Arthur spoke on system generation. Available material includes:

MPX System Introduction C26-3718-1

MPX Subroutine Library C26-3724-0

MPX Programmers Guide C26-3420-0

SESSION REPORT

COMMON - Chicago

Session Number Tues. B 3 Session Name "Conversion Experience & Tips"

Chairman R. J. Snailer Metropolitan Life (# 1495)

Time Tues., 10:30 a.m. Attendance (No.) 76

Speakers

Robert Wilkin, Hooker Chemical Company

James Bobay, Cummins Engine Co.

Robert Cornell, Federal Reserve Bank of Minneapolis

Don Forsyth, Continental Can Co.

Robert Therkildsen, I.B.M. - Chicago West

Reading Pollitt, I.B.M. - White Plains, N.Y.

Paul Koepsell, South Dakota State University

Synopsis of Meeting As it developed there was just sufficient time to accomodate the presentations of all the speakers. The talks contained a mixture of specific methods for meeting a particular conversion problem, various experiences in converting and an overall look at conversions in general. Programs for language conversions were described. Operating using emulation was commented on. Also discussed were training, load capacity, back-up and the need for a conversion plan which should be carefully prepared beforehand and adhered to.

4/15/68

COMMITTEE: Installation Management Division,
"Conversion Experiences and Tips" panel.

SUBJECT: Conversion Considerations

SPEAKER: Robert L. Cornell

COMPANY: Federal Reserve Bank of Minneapolis

ADDRESS: 73 South 5th Street
Minneapolis, Minnesota 55440

PHONE: (612) 333-0361, Ext. 418.

DAY, TIME, NUMBER OF SPEECH: Tuesday, April 9, 1968
10:30 - 12:00 A.M.
Tuesday Session - B3.

TEXT PAGES: 3

GRAPH PAGES: 0

Speaking for the Federal Reserve Bank of Minneapolis and aided by a great amount of hindsight, I should like to present a number of conversion considerations which may be helpful to others seeking "outside" ideas in this area. First I will present a summary of our current machine status, then view some of our impressions and ideas regarding conversion.

Prior to installing our present IBM System/360 Model 30E, our "shop" included a 20K 1620 Model I, two 402 accounting machines and a 416, one 514 and one 519 reproducing punch and a dozen keypunches. Our 360 was installed on July 1, 1967 with a 1620 compatibility feature attached. For a variety of reasons we are ordering an additional 32K for installation this summer.

Basically and quite briefly, our decision to convert to a larger machine was based on three important needs. First, the need had long existed for more raw computer power - faster processing speeds. With the diurnal bias of both programming and operating staff being mainly responsible, our operations had steadied to a 6:30 a.m. to 11:30 p.m. machine schedule, less and less of which was available for program testing or compiling.

Secondly, we recognized a need for larger disk file storage capacity and a desire to provide for storage of large data files on magnetic tape. Following a conservative policy on destruction of old card-data, we were faced with a continued accumulation of boxes of cards which would become unmanageable without taping facilities. Third, a near-future need exists to provide terminal facilities for direct data communications with other Feds and with the Board of Governors of the Federal Reserve System in Washington, D.C. A need to provide multi-programming environment required a machine change.

Hardware decisions are mainly a matter of selecting among the various promises as to what the manufacturers can and will provide for you. Decisions on specific readers and printers may be made based on the varying requirements of the installation. One tip to remember: In order to secure the best (and only good) bargaining position, written agreements regarding length and time of P.M. (Preventive Maintenance) should be made prior to installation date.

As in our case, the time schedule for delivery of the new machine may not allow time for extensive reprogramming prior to installation. When this occurs, it may be necessary to consider simulation or emulation as an aid to conversion. Having now had nearly one year of 1620 emulator experience on the 360/30, we would like to give a list of our impressions regarding this mode of operation.

1) Emulation is often fast and impressive. For identical jobs, it may even run faster than 360 DOS. Over our 1620 speeds we have reaped basically the gain in speed of our I/O devices; 1620 jobs can be expected to run 2 to 4½ times faster under Emulator.

2) Emulation causes problems in machine backup. Our nearest backup machine is in Des Moines, Iowa. The backup problem would be less crucial if our reprogramming effort were complete.

3) Emulation causes problems in hardware maintenance. In our area we have one IBM Field Engineer trained in 1620 Emulator.

4) Conversion should be swift. Any shop, subject as we are to programmer turnover, may find itself in the middle of a lengthy conversion with only one or two people capable of programming in the "old machine" language.

5) 1620 Emulation prevents any really full use of the 360/30. Multi-programming is not really practical, and the need to be forever switching back and forth between operating modes does not make for most efficient use of the machine.

6) Lastly, Emulation is a crutch. This is due to its reliability and speed, plus a programming viewpoint more concerned with getting new applications running and less concerned with converting old systems which do require reprogramming but which at least are now running under Emulation (faster too, remember!).

So we have run into a bottleneck in our reprogramming effort; complacency over Emulator speeds has certainly slowed this effort. The real question regarding emulation then becomes: Can sufficient resources be devoted to reprogramming prior to machine installation, or should emulation and a longer conversion period be considered?

Recall also that prior to installation date many of the programming staff will be spending at least part-time effort in learning about the new machine and even more probably, its new language. The choice of primary programming language will have an effect in at least two areas. One, because the level of basic knowledge about the way the machine functions varies greatly with the choice of language, this choice will have a bearing on the programmer competence in the area of assembler language which will be required of the individual. The problem-oriented languages are just that, and they can free the programmer from a need to know a lot about the intricacies of the machine.

Secondly, the decision on languages may have a bearing on the amount of core storage ordered for your machine. One of our primary reasons for ordering an additional 32K for our Mod 30 is the fact that we are a big COBOL user. COBOL certainly generates far more coding than could be written minimally in Basic Assembly Language.

One last area which we would mention is the area of Documentation Standards. Even for the shop where firm standards have already been established, a conversion period is a good time to consider updating or revamping those standards. We have found our gradual conversion to be beneficial in this respect, for it has given us a very good "handle" on exactly what should be required in the area of program documentation. To repeat, there is no better time to review documentation standards than at the time of a conversion.

In summary, a conversion should not be allowed to cause havoc in any area - whether in machine change-over, in re-programming, or in documentation. Reprogramming should be placed on a schedule to which it is reasonable to adhere. Documentation, as important as any area, should be used as a tool to guide programmers in the direction of a harmonious conversion.

Jim Bobay
Tue B·3

COMPUTER CONVERSION

INTRODUCTION

Planning a computer conversion passes through various stages of development similar to initial installation planning. Individual plans are greatly dependent upon the total computer status of the user. This presentation will review the planning process and installation highlights.

SYSTEM REQUIREMENTS

Feasibility

Company policy often directs the procedures to be used in feasibility studies of new or replacement equipment. Such a study would consist of a complete survey of present and potential applications. At this time, department heads should have the opportunity to make known their present and future needs. Upon the merit of the total requirement, equipment should be reviewed to accomplish the demand with allowance for growth. The growth factor, often unpredictable for immature computer systems, is generally compensated by improved technology.

One of the easiest feasibility studies attempted was the 1620 vs. the 1130 System. In most cases the cost alone satisfied the problem. However, other factors must also be considered in computer selection.

COMPUTER SELECTION

Collect and analyze information from all possible sources. Interview all potential users and discuss their needs. Study and evaluate each application and the equipment required to accomplish the different tasks. A practical performance plan should be developed. A data processing committee can be effective for outside direction and company priorities.

Consult other equipment users concerning the equipment under consideration. Find out the capabilities and limitations from experienced users. Study the various hardware configurations that might be considered feasible for

the applications. Arrange to make production test runs to form definite conclusions of the system capability.

Cost in most cases could be the most decisive factor of all. Similar to buying a car, after the basic equipment is selected, many accessories are added. The cost range of the 1130 begins at $695 and can increase to over $5,000 by adding auxiliary units and system improvements. Where to begin or stop must be determined in the feasibility study.

Performance specifications depend upon the applications to be processed. Such as, what records must be stored. Should disk, tape or card be used for required processing. Cycle alone will not give a true picture of processed through-put.

Training is always required in several different forms. Operators must be instructed on the proper use of all controls and how to handle routine processing and emergency stoppages. IBM should arrange training instructions to key personnel who can be ready to assist users at all times. Training should also provide orientation of non-data processing personnel concerning the capabilities of the new system.

Program conversion is probably the biggest in any change of equipment. Few systems are completely compatible. Extensive plans should be made to accomplish this task. Language changes must be known well in advance in order that programming personnel may make the necessary conversion changes on schedule. 1620 FORTRAN can easily be convered through the 1130 system FLIP program. Additional resources can be used if 1401 services are available. A 360 converter program (SIFT) is available for 1620 FORTRAN for either 1401 processing on the 360-1401 emulator. Compatibility from 1620 FORTRAN to the 1130 is very good, however, it is non-existent in symbolic programs.

~~[illegible]~~ Test allowance time at available IBM centers should be used.

Expansion will probably always be a part of almost every system. It can take various identities: storage, speed, auxiliary units or scheduling. For most data-processing installations it evolves into normal growth.

Communication in the computer field is perhaps the least developed field. However, spectacular things are sure to come in many forms of computer communication. Already terminals are available in different forms: typewriter, card, voice, tape, etc. The 1130 will soon be a terminal to the System 360 and perhaps it can also be predicted that there will be remote sources of communication to the 1130.

Load Capacity should always be a realistic part of every system. Of course, there are various methods of compensation from an overload. It can first be challenged by increasing the computer speed; other methods are: faster input-output units, larger core capacity for better programming efficiency and greater storage capacity. Beyond the system itself, scheduling can go into overtime or week-end hours. The 1130 8K core capacity is comparable to the system 360/30 32K and about equivalent to a 1620 40K.

Back-up is closely related to overload but it must be outside the system. In the case of a breakdown or overload, is there another system available? For some programs this is an essential element of a complete system. In the case of FORTRAN programs, another computer system should be sufficient, provided it is large enough. Down-time is a fact of life in computer activity and should fit in overall systems planning

Limitations are a part of all systems. A small computer should not be selected for major applications. Some projects are literally impossible on a small system. This should be discussed with other users to acquire an understanding of the problem. The system can be no better than the service turn-around it provides. User acceptance will greatly depend on service.

A conversion plan should be established as soon as possible with firm discipline in maintaining its schedule.

PHYSICAL REQUIREMENTS (A26-5914-2)

3-Months before delivery is a very critical stage, perhaps a go, no-go stage. It is a time for decision, first for the computer and then the environment factors. After determining a firm order the plant engineer should be called to discuss physical requirements.

1. Temperature (60 to 90° F)
2. Relative Humidity (10 to 80%)
3. Heat (Average 5900 BTU/Hr.)
4. Dust and Dirt (Normal Precaustions)
5. Fire Extinguisher System should be Non-Wetting
6. Power Supply - 115 Vac., 3-Cycle (2 Power, 1grd.)
7. Lightening Protection
8. Cables to Specified Length

[redacted]
The layout should be approved by IBM so servicing can be achieved.

If program conversion is taking place, a definite schedule should now be reviewed as well as orientation of concerned personnel to the new system. Another critical item at this stage is the ordering of the monitor system.

1-Week before delivery several units have probably arrived and are probably setting on the receiving dock, but it is time to make a check on environment requirements. Several administrative tasks are:

1. Issue operating procedures
   a. Time schedules
   b. Services
2. Training
   a. Computer concepts
   b. Programming classes [redacted]
   c. System review
   d. Language changes
   e. Review FORTRAN level (open-shop modifications)

PROGRAM DEMONSTRATION

TUE B3

INSTALLATION MANAGEMENT DIVISION, PERSONNEL PROJECT COMMITTEE

"Professional Programmers and Analysts: Problems in Performance Evaluation"

By: Arthur S. Gloster, II
Oak Ridge Associated Universities
P. O. Box 117
Oak Ridge, Tennessee

Thursday, 10:30 a.m.

Two pages of text.

PROFESSIONAL PROGRAMMERS AND ANALYSTS: PROBLEMS IN PERFORMANCE EVALUATION

Arthur S. Gloster, II

Oak Ridge Associated Universities is a nonprofit corporation in Oak Ridge, Tennessee. The primary function of the corporation's data processing center is to apply electronic data processing techniques, where feasible, to research and administrative projects. The center contains 27 employees of which 13 are analyst/programmers. The center is divided into three groups: 1) scientific applications consisting of 6 personnel and a group leader, 2) commercial applications consisting of 5 personnel and a group leader, and 3) operations section consisting of a supervisor and 10 other employees. Oak Ridge Associated Universities has on site an IBM 1800 disk/tape system which is utilized by approximately 40% of the programming personnel. Approximately 60% of the personnel use IBM 360-50, 360-75 and CDC equipment located in the area. The analysis and programming function comprise a significant portion of the operating costs of the ORAU data processing center.

After discussion with personnel from other installations, we found that there are no reliable standards by which costs can be calculated in advance, schedules established, and the performance of personnel evaluated. Although the methods we used to establish schedules and costs are subjective and arbitrary, they in no way approach the accuracy of the methods developed in the hardware area. For example, IBM has established rates for EAM equipment and has even produced a slide rule to use for estimating job times.

A means of evaluating the professional analyst/programmer's job and a means of evaluating his effectiveness is highly desirable but rarely accomplished. Such means would be helpful predictors in determining the staff needed for a particular application or a data processing installation. We have found that records of intangibles, such as the time for the application analysis and problem definition, flow charting, coding, debugging, checkout, and finally documentation would have to be maintained continuously to have a base for predicting analyst/programmer costs and for personnel evaluation. In predicting costs of computer programs and evaluation, we would like to be able to have a magic number representing the proper number of analysts or programmers that could be applied to a given situation and for our center as a whole, but we found this to be impractical because we were measuring intangibles by subjective means. We found that attempting to save expenses by minimizing or restricting the availability of professional personnel caused equipment to be used ineffectively. The more the programmer is annoyed with accounting for his time and the more detailed the account for nonproductive time, the less apt he will be in cooperating in a program that keeps up with all of the various functions he performs.

Programming personnel at ORAU are engaged in numerous types of jobs; therefore, standards of work evaluation could not effectively reflect the variety of tasks they encounter. One programmer may be responsible for coding X number of instructions with relatively small amounts of logic development, while another programmer may be responsible for extensive logic development with relatively few instructions. Thus, it becomes difficult to measure the amount of work required on each program, and the total work effort performed by a programmer cannot be assessed in standards of comparison with another programmer.

At ORAU, we believe that the group leader of either the scientific section or the commercial section, depending on the particular area, should look at the problem in advance and then meet with the data processing manager to establish reasonable target dates for each of the previously-mentioned phases on the basis of the nature of the problem and on their past experience. The manager or group leader must have a detailed knowledge of the problem under study because it is his responsibility to prepare the cost estimate and schedule. He must keep up with the allocation of funds and judge progress of the application. After giving several methods of job measurement trial, we have found that there is no substitute for experience in the area of predicting costs, measuring work, and evaluating programming personnel. Programmers respect a supervisor who gives them a job and can tell them what performance is expected. The supervisor is also respected by his staff if he remembers that good supervision is the least supervision needed to get the job done.

SESSION REPORT

COMMON - Chicago

Session Number TUE B5 Session Name N/C - Languages and Graphics

Chairman Joe Talkington

Time 10:30 to 12:00 Attendance (No.)

Speakers Mr. William Peterson

Synopsis of Meeting The speech was primarily on the APT Language and the need for computer support. Distinction was made between symbolic control and numerical control. Questions delt with the relationship of APT to small computer users.

REMARKS TO:
COMMON and CEPA
APRIL 9, 1968
PICK CONGRESS HOTEL, CHICAGO

Dr. John Porter
Director of Scientific Development
IBM Corporation
112 East Post Road
White Plains, New York

My topic today is: "The COMMON Challenge"

I chose this topic for two reasons: The first one is that I like puns. But the second reason is that what I want to say to you applies, I think, no matter how you interpret the word COMMON.

Whether you translate COMMON as the largest computer users group in the world, measured in number of member installations, or whether you use a dictionary definition; such as: "belonging or pertaining to the community", as long as you mean the computing community.

Because there is a challenge to the COMMON organization, and there is a challenge to all of us involved in data processing and computing, it's a complex challenge, not a simple one. This complex challenge that I see consists of three related parts.

The first part is to make our systems do what we know they can do -- for us, for our organizations, and for our society. The second part is to advance the state of the art to "dream the impossible dream", so to speak, and then make it happen. And the third part of this complex challenge is to use what we've learned -- about systems and languages, about machines and organizations, about human factors and problem solving techniques -- not only within our own organizations but in society at large.

Although the way I look at this challenge is my own -- and may be different from yours -- certainly its presence is nothing new to members of COMMON. Your formation of COMMON eight years ago was a forward looking response to this challenge. Your response to this challenge is what has brought us together in this meeting and at this luncheon.

Your response to this challenge -- hour by hour and month by month and year by year -- is what has given your eight full years of accomplishment, innovation, and leadership -- in your own disciplines and in the data processing and computing community at large. I daresay your response to this challenge has meant long days and short weekends -- the invention of algorithms while you were shaving; or, if you don't shave, while you were powdering your nose -- discovering bugs in programs while you were mowing your lawn.

COMMON's eight years as a leading-edge organization -- as a group that makes things happen -- is an outstanding example of how to do in the organizational world what Thomas Edison did in the industrial world -- organize for progress.

Isaac Newton is supposed to have said that if he had been able to see farther than others, it was because he had stood on the shoulders of giants. What Newton did as an individual, COMMON has done as an organization.

My feeling is that COMMON is on the right track.

My advice is to stay on it.
Present more papers.
Share more experiences.
Contribute more programs
Continue to prod IBM.

Although we don't always move in the direction you push us -- there are many other vectors that affect the resultant -- we want you to push us. And although we don't always move as fast as you like, we move. Your challenge is also our challenge.

Now I'd like to show you that IBM understands at least some of your requirements. Let me use my own shop as an example. Our primary concern is any use of information processing systems by engineers, scientists, and mathematicians. More specifically, we're interested in the techniques and disciplines which are commonly employed in the solution of problems in scientific computation. For example, we have produced a package of subroutines called the scientific subroutine package which provides the basic techniques of numerical analysis, statistics, and matrix manipulation. We have produced a series of linear programming codes -- in fact, one of my colleagues, Harry Muller, is on this meeting's agenda to present a tutorial on our 1130 LP Code. We have produced generalized network analysis programs, utilizing critical path and pert techniques. We have produced the cont. syst. modeling program, which allows the simulation of systems described by ordinary differential equations.

We have produced the general purpose simulation system which allows the simulation of discrete processes. Since this group represents the leaders in scientific computing, your inputs to us are quite significant in defining the requirements for these techniques. Let me add that we will continue to provide fully-supported application programs for these techniques and their variants. And for a reason that is very important to both of us. It is clear that the most important factor which will limit the growth of computing in the next 5-10 years is the shortage of skilled application and systems programmers. By providing tools and techniques which meet your requirements, we can free up this scarce resource, so that you can apply it in new, high-potential areas -- rather than squandering it by reinventing the wheel.

Now, in addition to the production of application programs, my shop is also working on longer-range projects. This activity is currently underway in the scientific centers. Since these centers are fairly new, most of you are probably unfamiliar with them; and therefore, I'd like to describe them in some detail.

There are six scientific centers located in Palo Alto, Los Angeles, Houston, Washington, D.C., New York and Cambridge. Each center is composed of about 15-20 professionals plus supporting personnel. Why, you ask, are there six small centers, rather than one large center? The answer is that we feel it's important to work closely with leading institutions around the country where

we have a common interest -- that is, pushing back the frontiers of computer science. And that brings me back to my topic --The COMMON Challenge. What is the challenge facing the scientific centers? We believe that the computer potential in solving man's problems has only begun to be realized. In recent years completely new opportunities have opened for the computer which we are only beginning to explore -- management science, environmental sciences, medicine, transportation systems, to name only a few. There is every reason to expect that the role of the computer will continue to grow, provided the correct environment is provided. Therefore, the challenge is to take the lead in identifying, monitoring, and developing these new areas of applications technology.

Let me describe briefly some specific projects now in process. One project in Palo Alto is concerned with on-line experimentation using an 1800 tied directly to multiple instruments. One facet of this work is a joint effort with Stanford University to control and monitor experiments using the two mile long Stanford linear accelerator.

In Los Angeles, we have sent an 1800 out to sea on an oceanographic research vessel from the Scripps Ocenaographic Institute. It is used for data acquisition and analysis and makes possible the use of satellite navigation to more precisely determine the position of the ship when data is being taken.

A Houston project has demonstrated the feasibility of generating holograms digitally. In addition to the implications for 3-dimensional displays, we feel that this technique could be used to improve the resolution of the electron microscope.

In New York we are working on extensions to linear programming. Such as integer programming, mixed integer programming, decomposition, and non-linear programming. Specific problems being addressed include airline crew scheduling and freight car scheduling.

Cambridge is concentrating on the development of new systems techniques to allow more efficient interaction between a user and a computer in a time-sharing environment.

This is just a small sampling of projects in the scientific centers. In addition, they produce internal technical reports; they publish regularly in the open literature; they hold special seminars on subjects where they have some expertise; and they are open for customer visits. I encourage you to find out more about these centers through your branch office and to take advantage of their special knowledge wherever possible.

In summary, then, I see a COMMON and a complex challenge facing all of us--

and a resultant need for closer cooperation in meeting that challenge. I see COMMON as a vehicle for coordinating our efforts in this regard. For what it's worth, I would like to give you my threefold prescription for continued success in the future.

First, as I've already said, support COMMON. In concrete terms, consider the coming election of officers. Don't wait until you're asked, volunteer. If you're asked, don't hesitate, accept. When you vote, vote for the individuals who have the clearest view of COMMON and what it should be.

Second, use COMMON as an interface with IBM. The 1966 reorganization of COMMON was designed to make this easier for you to do. Don't overlook the opportunity.

And third, keep up your inventiveness and creativeness. Society needs your help.

Thank you very much.

SESSION REPORT

COMMON - Chicago

Session Number TUE B8 Session Name OS Committee Meeting

Chairman W. Norton

Time 10:30 to 12:00 AM Attendance (No.)

Speakers J. A. Woodworth - Dow Chemical Company

Synopsis of Meeting In addition to committee business and organizational planning, a report on SHARE XXX was presented by Mr. Woodworth.

DIVISION: Systems

PROJECT: 360

COMMITTEE: O/S

SUBJECT: Report on SHARE XXX

SPEAKER: J. A. Woodworth

SESSION: B9 Tuesday, April 9, 1968

REPORT ON SHARE XXX

BY

J. A. WOODWORTH
USER 5155

DOW CHEMICAL COMPANY
CE&CS DIVISION
HOUSTON, TEXAS 77027

APRIL 9, 1968

Introduction

On Friday, February 23, 1968 at about 4:00 p.m., I received a long distance telephone call from Wade Norton, the Chairman of the O/S Committee, asking if I could attend the meeting of SHARE XXX which was starting in Houston the next Monday morning. Since I happened to be in my bosses office at the time I received the call, he gave me an okay and so I was able to attend the meeting. I have attempted to record as much information as possible about SHARE's organization and, in particular, the OS/360 project.

General Comments

The SHARE XXX meeting lasted five days of which the first two days were limited to closed project working sessions. Attendance at these sessions was limited to project and committee members and those who had obtained the permission of the project leader. Those interested in participating in committee and project work were instructed to contact project leaders before coming to the meeting.

Organization of SHARE

Since the current organization of COMMON was patterned after SHARE, there are, of course, a number of similarities. The main divisions are: Administration Committee, Advanced Planning Division, Applications Division, Graphics Division, Installation Management Division, and Systems Division.

Projects under the various divisions are:

1. Administration Committee--distribution and programming standards.

2. Advance Planning Division--data mapping.

3. Applications Division--applied management science, design automation, electrical analysis, general information systems, general file maintenance, list processing, mathematical programming, numerical analysis, statistical methodology and systems, symbolic mathematical computation, and systems simulation.

4. Graphics Division--applications, hardware, and software.

5. Installation Management Division--university installation management, and operations management.

6. Systems Division--ASP, assembler, COBOL, DOS/TOS, Fortran, HASP, OS/360, PL/I, TSS, Model 44/Model 91.

OS/360 Project

The current working groups on the OS/360 project are: Job management, data management, telecommunications, storage hierarchy, MFT improvements, hardware considerations, performance evaluation, and system management (i.e., job accounting).

The first order of business was a review of the resolutions passed at their December meeting. Some of those discussed which might be of interest to our committee are:

1. Ability to allocate and deallocate storage dynamically.

2. Ability to allocate and deallocate direct access storage during a job step.

3. Close and release a data set without terminating the job.

4. Nesting of catalog procedure calls.

5. Ability to add devices without going through SYSGEN.

6. Controlling tape error retries--now does 100.

7. Option--owner ID, creation dates for PDS.

8. Condense partitioned data sets more efficiently.

9. Access PDS from higher level languages.

OS/360 Project--Data Management Task Force

The data management group presented the following resolutions:

DM-01 Look into tape error retries--now 100.

DM-02 Recommend automatic catalog maintenance and library control.

DM-03 Allow use of direct access devices by two or more CPU's.

DM-04 Improved method to condense PDS data set.

DM-05 Ability to access PDS from higher level language.

DM-06 Allow reading file protected data from Fortran.

DM-07 Ability to define and use channel to channel feature as an I/O device.

DM-08 Request more information in PDS directory such as creation date, etc.

Other items were discussed such as changes in Release No. 14.

Fortran Project

A presentation was made by an IBM representative on data set compatibility between the following systems: DOS/TOS, TSS, M44, and OS. Data sets should be on tape unlabeled. The OS default scratch buffer size is 800 bytes. To speed up I/O, use array mode rather than lists or implied do's.

The Fortran project passed 12 resolutions. Voting was by active members of project only. Count was kept of the pro's, con's and abstentions. The resolutions were as follows:

68-1 Allow the typing of EXTERNAL function statements External SQRT (Real*4).

68-2 Allow variables as parameters to the define file statement. These could be passed as arguments to subroutines.

68-3 Output the buffer of the SYSOUT data set of ABEND.

68-4 Set precision of constants by context.

68-5 Fortran setting of condition codes.

68-6 Recommended parallel APAR handling between related Fortran compilers.

68-7 Eliminate problem of subroutine argument list when the number of arguments is eight.

68-8 Defeated.

68-9 DO LOOP indexing parameters--have compiler issue warning message on changing parameters inside of loop.

68-10 Fortran I/O requested improvements.

68-11 DEFINE FILE statement--clarity manuals to indicate that in overlay structures, the controlling DEFINE FILE must not be overlaid.

68-12 Clarify implications of double length precision extensions to Fortran for the Model 85.

These resolutions were then ranked by each committee member according to three priorities:

1. Paramount importance.

2. Soon.

3. In due time.

Format of Project Resolutions

Resolution No. _____ Date _____

Task Force _____

A. Requirement
B. Problem description
C. Resolution
D. Notes

Project Activity and Progress Report

Project _____ Division _____

Project Chairman _____ Installation Code _____

Date Submitted _____

(The following is preprinted on the form)

Please list the following:

1. Scope and objectives of project.
2. Recent project activities.
3. Planned project activities.
4. Manpower needs.
5. Membership qualifications.
6. Membership roster.

J. A. Woodworth
4-3-68:cd

Recommendations for COMMON OS/360 Committee Action

1. The COMMON OS/360 Committee should maintain a close relationship with the SHARE OS/360 project. This should probably be done by the Chairman.

2. Minutes of meetings and records of activities should be exchanged by the respective chairmen.

3. Specific items of significance be made available to active OS committee members or to the whole membership of COMMON depending on merits.

4. A representative from COMMON attend each SHARE OS/360 project meeting.

5. Formulate procedures for the documentation of the OS committee's activities and recommendations.

SESSION REPORT

COMMON - Chicago

Session Number TUE C2 Session Name Biomedical

Chairman Dr. James L. Grisell

Time 1:30 to 3:00 PM Attendance (No.) 50

Speakers Mrs. Joan Lukin - University of Kentucky Medical School

Dr. Emanuel Donchin - Ames Research Laboratory

Lukin - "Computerization in the Clinical Laboratory"

Donchin - " Acquisition and Analysis of EEG Evoked Responses"

Synopsis of Meeting Mrs. Lukins presented a system for the on-line analysis of auto analysers. These are used in performing clinical laboratory determinations. Dr. Donchin reviewed his work on eveoked responses with an emphasis on the difficulties of high speed data acquisition using IBM software. The discussion after the formal presentations showed that many users have had to find a variety of ways around the serious limitations of the system software when doing high speed data acquisition.

COMPUTERIZATION IN THE CLINICAL LABORATORY

by

J. Lukins, M. Ball, W. B. Stewart, N. Hill, and R. O'Desky

The purpose of this paper is to explain and illustrate the use being made of an IBM 1800 Computer by the Clinical Laboratory of the University of Kentucky Medical Center. We shall point out the need for the computer, explaining the task undertaken, and outlining the solution developed. We shall also touch briefly upon future plans for the computer.

COMPUTERIZATION IN THE CLINICAL LABORATORY

The clinical laboratory of the University of Kentucky daily handles a number of specimens from patients throughout the hospital. These specimens must be transported to the laboratory and analyzed, and the results returned to the patients' charts, a sequence of events which at present introduces the possibility of a number of errors of identification, technical inaccuracy, and clerical mistakes. Let us trace this sequence of events, and point out as we progress the areas in which error may be introduced. Because our efforts to date have been concentrated in the clinical chemistry area, we will consider only those analyses in this paper.

A patient entering the hospital is given a six-digit hospital number, representing the total number of admissions to date: if for example, he were the 368th admission, his number would be 0003684, the seventh digit being merely a check digit to assure the validity of the six-digit number. A non-removable wrist-band containing his hospital number is placed around his wrist. His attending physician, upon examination, writes a request in the physicians' order book for the required laboratory tests. A technician then completes a request slip (in triplicate) for the requested tests (see Figure 1), draws the required samples from the patient in his room, labels the specimen with the same accession number as the request slip, and carries both the request and the sample to the clinical laboratory. Already error may have been intro-

duced: the technician may have placed the wrong label on the specimen.

A technologist receives the sample in the clinical chemistry laboratory, and, matching the accession number of the request slip with the accession number of the sample, she gives both of them a new number - a laboratory number - the first specimen and request being labelled "1", the second "2", etc. Here, of course, is another possible source of error - it is possible that she may assign the wrong request slip to a given sample. Next, the patient's name and laboratory number are entered in the master log sheet (see Figure 2) and a notation is made under each test requested. The sample is then placed in the centrifuge, separating serum from cells, and the resulting supernate is decanted into a clean tube, having the same laboratory number as the sample tube (another possible source of error), and this tube is placed in a serum rack (see Figure 3).

After all specimens have been received, the technologist who is performing glucose analysis, for example, records on a separate log sheet the laboratory numbers of the patients requiring glucose analysis in the order in which they will be run (see Figure 4); the technologist performing blood urea nitrogen determinations does likewise for BUN's, etc. Each technologist then draws off small samples of the patients' sera, and places the samples in small autoanalyzer cups in the same order as they appear on her log sheet. She then places a series of standard solutions containing a known amount of the chemical being measured in front of all the patients' sera (see Figure 5.). Obviously, this series of transfers of numbers and sera introduces the possibility of additional errors.

At this point the autoanalyzer assumes the responsibility for analyzing the sample (see Figure 6). Each standard sample is aspirated in turn into the autoanalyzer, mixed with reagents, and carried through the colorimeter. Now, as light is passed through the resulting solution, some of it is absorbed by the solution: the greater the intensity of the color in the solution, the greater the amount of light absorbed, and, consequently, the smaller the amount of light that passes through the solution. A photocell receives the light that is transmitted, and the resulting voltage is used to drive the pen on a slowly moving strip chart recorder. The pattern produced is shown in figure 7.

The technologist must now wait for the entire pattern to be completed, then remove the strip chart, and calculate the patients' results by comparing the heights of their respective peaks to the heights of the standard peaks and interpolating to determine the individual results. Each answer is written on the individual log sheet by the technologist who performed the analysis. These results are transferred by the technologist to the master log sheet, and, finally, transferred to the original request sheet by another technologist. The results are then punched into cards, to be kept by the laboratory, and the original request slip is placed in the patient's chart for the physician's information. After discharge from the hospital, the entire chart is sent to the Medical Records Department. Here, again, is a series of manual transfers of data allowing for the possibility of numerous clerical mistakes. In fact, it was found that it was in this particular series of transfers that the majority of all laboratory errors occurred.

Therefore, the IBM Computer (see Figure 8) was introduced to improve the accuracy and precision of the laboratory tests, to provide greater speed of reporting, and especially, to eliminate as many of the above-mentioned clerical errors as possible.

Here is the way in which the system operates (use figure 9 as a flow diagram):

When the technologist has placed the samples on the autoanalyzer and has established a steady baseline voltage reading on the pen recorder, she is ready to begin the sampling. She sends a signal to the computer via the 1092 keyboard (see Figure 10) telling the 1800 which test is being performed (test code: each laboratory analysis has been assigned as unique six-digit number), on which autoanalyzer it will be running (device number: each colorimeter has been assigned a unique two-digit number), and that it should now initiate analog input reading on that particular device. The 1800 maintains a table internally which contains all the device numbers it is currently sensing. When the computer receives the "start analog reading" request from the 1092, it adds that device number to the table and establishes the direction of the peaks. Every few milliseconds, the voltage is sensed on each of the devices in the table. An analog to digital converter within the 1800 transforms this voltage signal to a numeric value; this value is then compared to the value last read from that device. If the Δx is within a certain predetermined small range, we know that we are very near to a peak value, and we save in another table (which we shall call RDAT (see Figure 11)) all of the values read from this device until the slope begins to change direction;

at this point we know that one of the values just previously read was the peak value.[1] Whenever a peak is found for any device, this series of digital values is placed in the file PEAK (see Figure 12).

This process continues until all of the tests have been entered. At that time, the technologist returns to the 1092 matrix keyboard, keys in the test code and device number, and signals the computer to stop analog input on that particular device. When analog input has ceased on all of the devices, indicating that all devices have completed their analyses, the routine COLAN takes over; taking each series of digital values in the file PEAK in turn, it finds the peak within the series, and converts it to a floating point number. COLAN sets up a new file, which we shall call COLOT (see Figure 13) into which it places these values, each entry representing one peak.

But let us backtrack for a moment to the point at which the sample and the request were given a laboratory number. We followed the fate of the serum sample, now let us follow the request slip.

The request slip is taken to the keypunch operator, who punches a card for <u>each test</u> being performed (see Figure 14). The cards for each test are then run through the card reader in the <u>same order</u> as the serum samples are placed on the autoanalyzer. The program CRLDI places the information contained on the card in the disk file PTST (see Figure 15), one record per patient test.

At this point, then, we have a "result" file (COLOT), and we have a "patient" file (PTST); now the two must be collated; this is the job of the program DLABI. Taking each test result in the file COLOT and the corresponding patient record from the file

PTST, it calculates the test results by interpolation against the standard values, [2] prepares a new record containing the patient information, test code, and test result, and places this record in a final disk file labelled FFLE (see Figure 16). When it has completed this job, and has emptied both the "patient" file PTST and the "result" file COLOT, it calls upon the program DSRT. DSRT prints the test results for each patient, as in Figure 17. This printed output is then sent to the patient's chart, eliminating the need for any manual data transfer.

In addition to the printed output, another function of DSRT is to reorganize the file just written by DLABI by gathering together all the data on any given patient, sorting the patients by hospital number, then writing these patients with their test results onto magnetic tape. If, for instance, John Smith had a BUN test and a blood glucose analysis, he will appear twice in the final disk file FFLE written by DLABI. Therefore, DSRT must now merge the two test results together into John Smith's single tape record.

We must not forget, of course, about the tests which are not run by autoanalyzers, e.g., magnesium analyses, lipid determinations, etc. These, too, must be entered into the computer, but in a different manner: patient information, test code, and test result are all keypunched into a card, one card for every test. The program DLAB reads these cards and enters the information into the final disk file FFLE, where it is ready to be used by DSRT, as outlined above.

One step remains, now, and that is the maintenance of the laboratory files. As was shown, a daily file is generated and recorded on tape. A master file is also maintained, containing all

of the laboratory test results of all patients currently in the hospital. Obviously, to keep this file current, each day's results must be added to this master tape file, and the data on patients discharged must be deleted from this file. The job of daily additions belongs to the program DLUPD, which merges the daily tape file with the master tape file, creating a new master tape file in proper numerical sequence by hospital number. The job of deletions falls to the program PURGE, which locates the patient records to be deleted, prints a complete summary of tests performed on those patients during their entire hospitalization, and punches cards containing this same information. These cards can then be used to create an historical file containing all laboratory data on all patients discharged from the medical center.

What improvements, then, have been made by the introduction of the 1800 computer in the clinical laboratory. First, and probably most important, is the elimination of many technical and clerical errors. Before the arrival of the 1800 it was determined that the majority of all laboratory errors were the result of numerous data transfer steps, most of which have been eliminated by the computer. Our results are now more accurate, more precise, and - far from a small consideration - more legible! The speed with which the entire process can be accomplished is another important factor - more technologist time is available, and fewer clerical hours are required, an important monetary consideration. From the physician's standpoint, aside from the obvious improvement of more accurate, legible results, there is also a decided advantage to the physician to see a weekly summary of his patients'

laboratory results, and, upon discharge, their total laboratory picture in a consolidated form, both of which are now possible.

Of course, as with any innovation, there are numerous obstacles still to be overcome, and many areas in which improvements can be made. We hope, for example, that in the future, laboratory requests will be punched into cards directly, at the time the test is ordered, thus eliminating the manual request slip altogether. We hope, also, to develop a system of identifying a serum specimen directly, rather than relying upon the order in which the specimens are analyzed. The computerization of other types of clinical laboratory equipment, e.g., coulter counters, densitometers, spectrophotometers, etc., is also in the near future. All of these objectives will eliminate technical and clerical error, and, ultimately, provide improved patient care. After these goals have been realized, we hope to further broaden our scope to include computerization of hematology procedures such as differentials, actual on-line monitoring of post-operative patients, and a complete, centralized hospital information system to collect and correlate patient data in all areas of the hospital.

All of this requires the cooperation of the physician, the laboratory personnel, and the hospital administration. But none of it, we feel, is outside the realm of possibility.

INDEX OF ILLUSTRATIONS

FIGURE 1
REQUEST SLIP

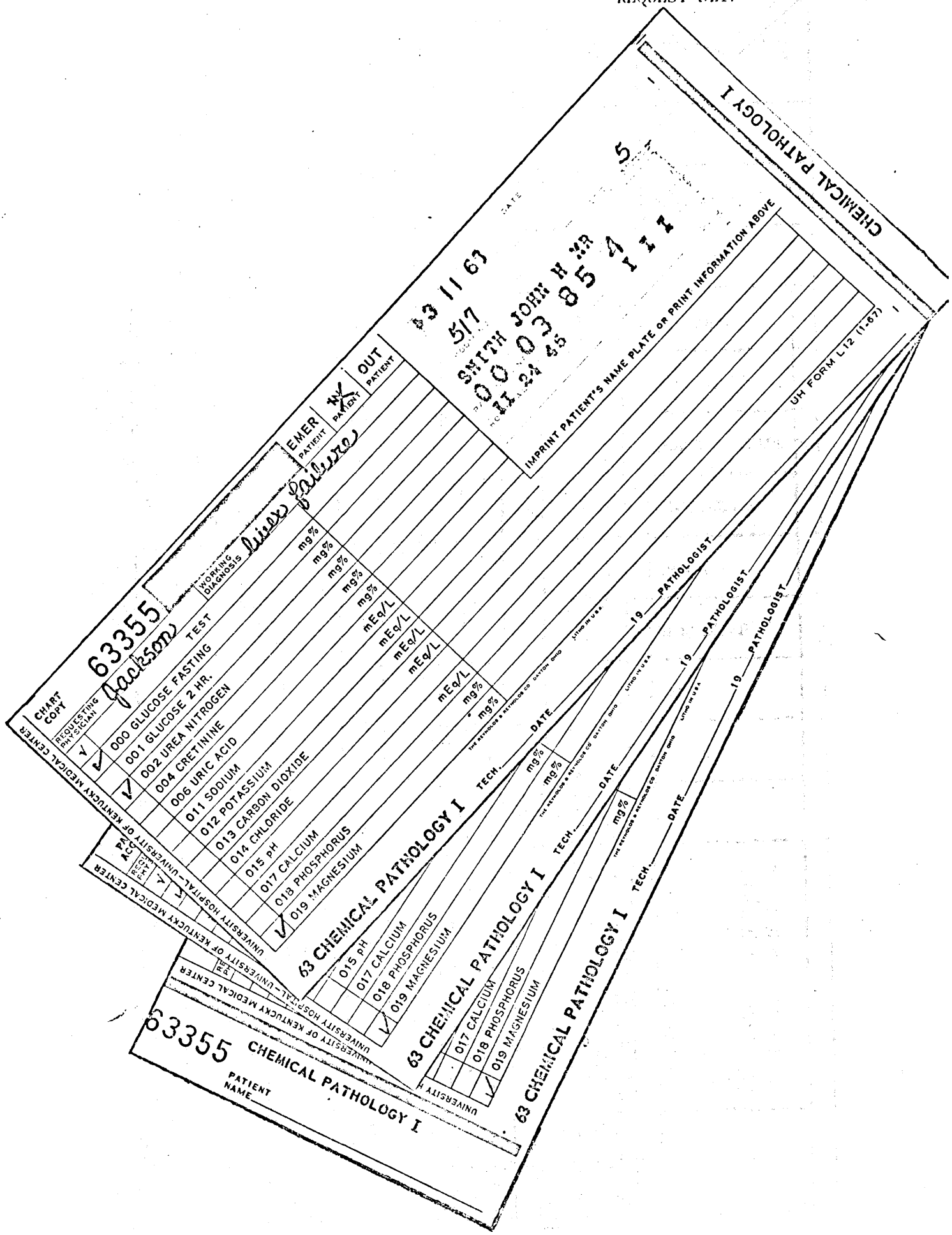
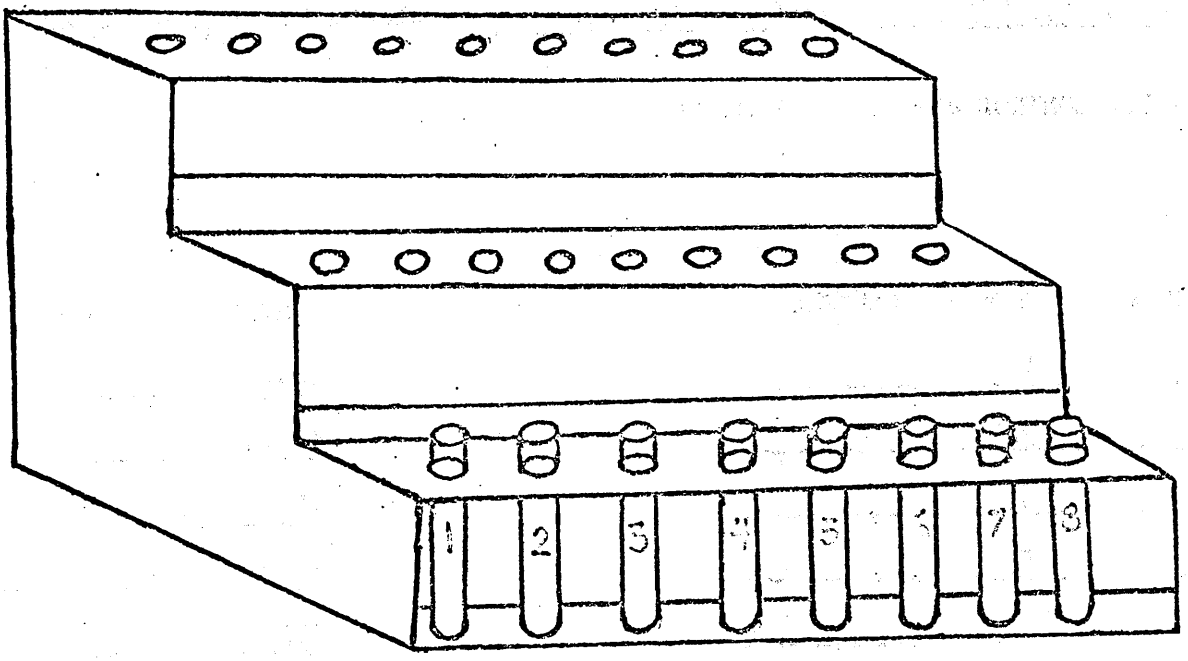


FIGURE 2
MASTER LOG SHEET

LAB NO.	CHEMISTRY MASTER SHEET DATE PATIENT NAME	ROOM NO.	GLUCOSE	BUN	CREATININE	SODIUM	POTASSIUM	CARBON DIOXIDE	CHLORIDE	MAGNESIUM	CALCIUM	PHOSPHOROUS	SGOT	SGPT
1.	Smith, John	517	✓	✓	✓					✓				
2.	Kelly, Sharon	632	✓	✓										
3.	Johnson, Bill	430		✓	✓									
4.	Brady, Edwin	575	✓	✓										
5.	Nakem, Carol	842	✓											

FIGURE 3
SERUM RACK



EACH RACK HOLDS 200 SPECIMENS

FIGURE 4
INDIVIDUAL LOG SHEET

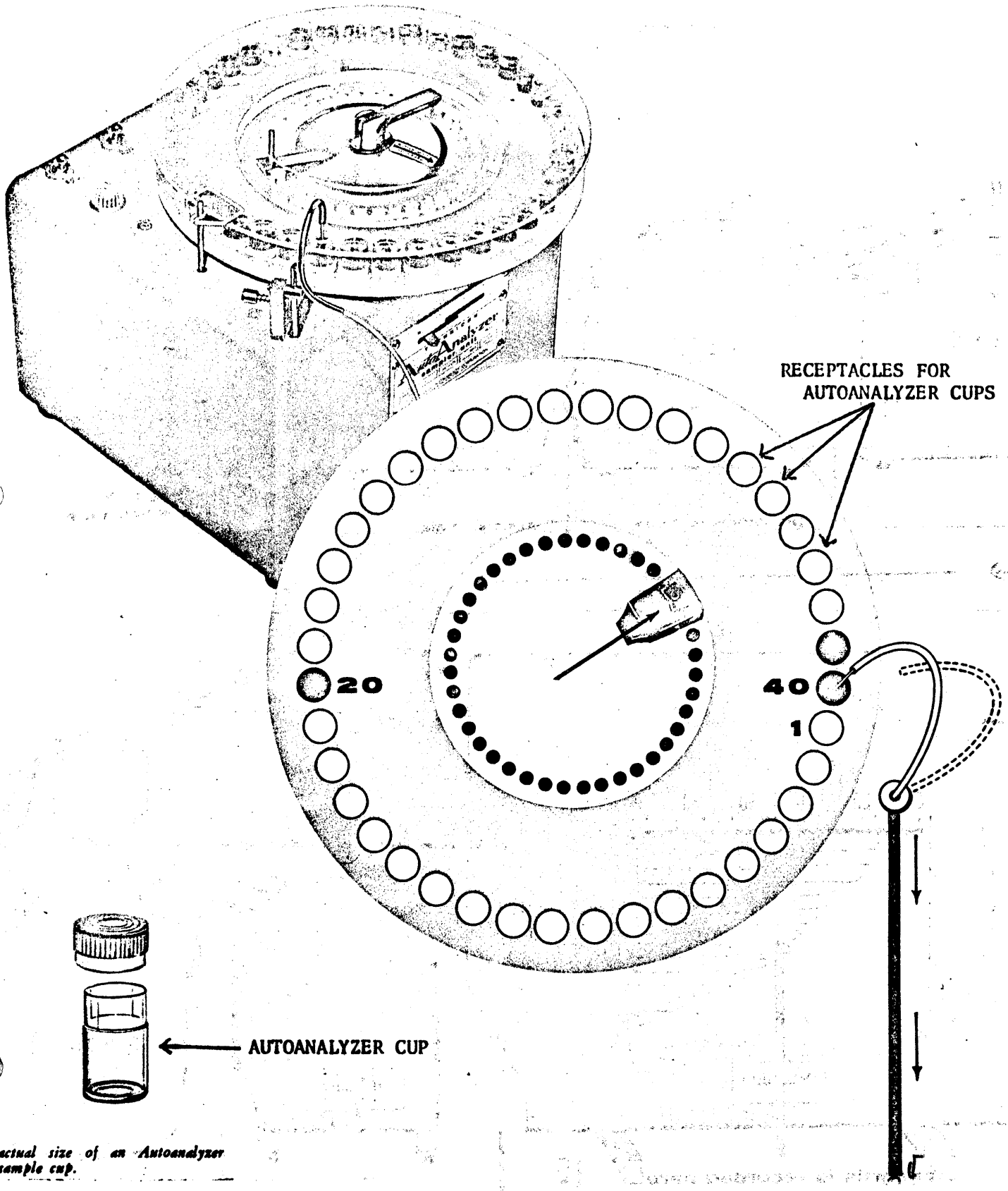
GLUCOSE

DATE 3-11-68 TIME 9 A.M. TECHNOLOGIST Jane Hagan

QUALITY CONTROL POOL GLUCOSE

SPEC.#	PT.#	GLUCOSE	SPEC.#	PT.#	GLUCOSE
1.	1.		16.		
2.	2.		17.		
3.	4.		18.		
4.	5.		19.		
5.			20.		
6.			21.		
7.			22.		
8.			23.		
9.			24.		
10.			25.		
11.			26.		
12.			27.		
13.			28.		
14.			29.		
15.					

FIGURE 5
AUTOANALYZER SAMPLER PLATE



actual size of an Autoanalyzer sample cup.

FIGURE 6
TECHNICON AUTOANALYZER

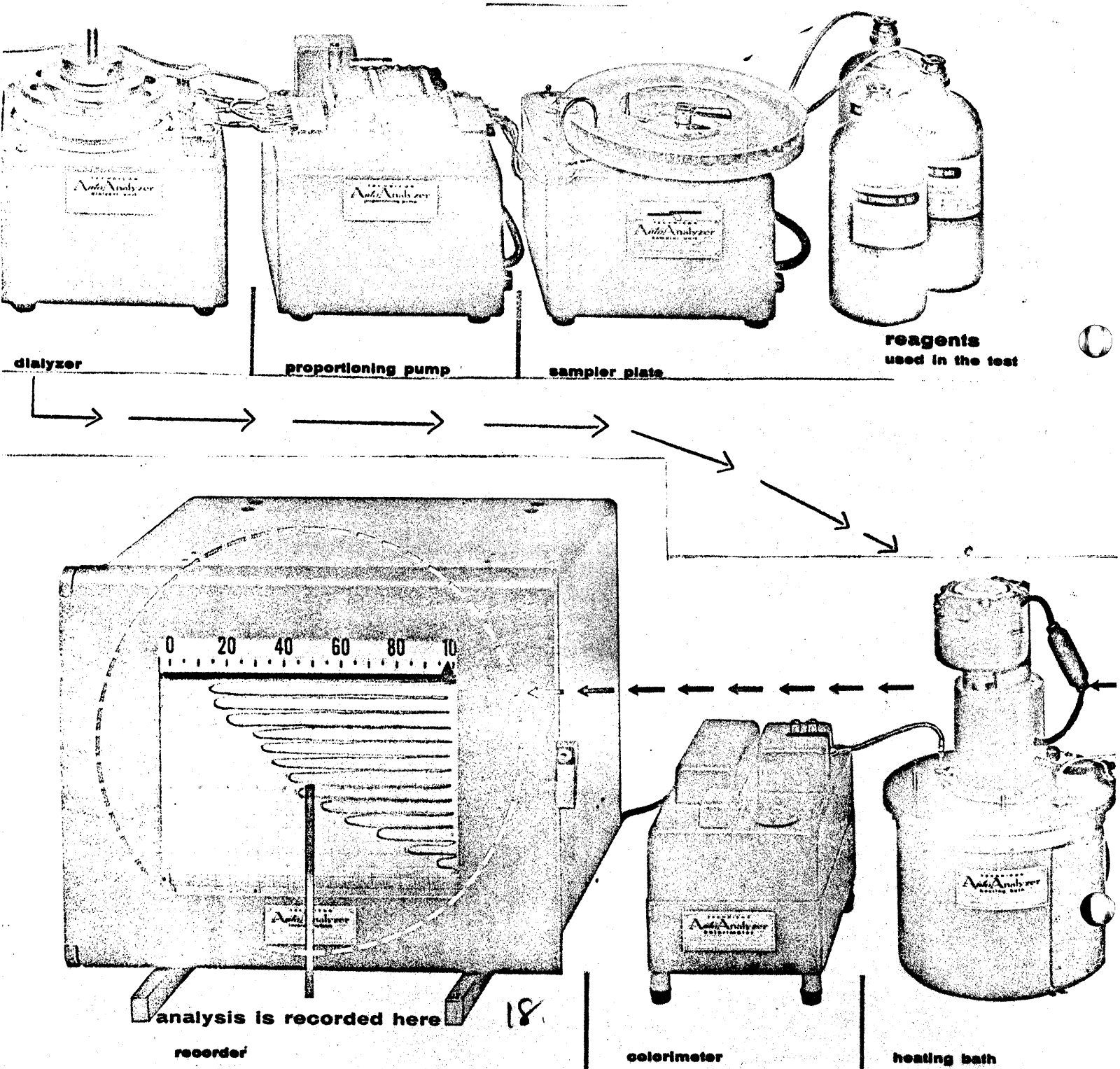


FIGURE 7
STRIP CHART RECORDING

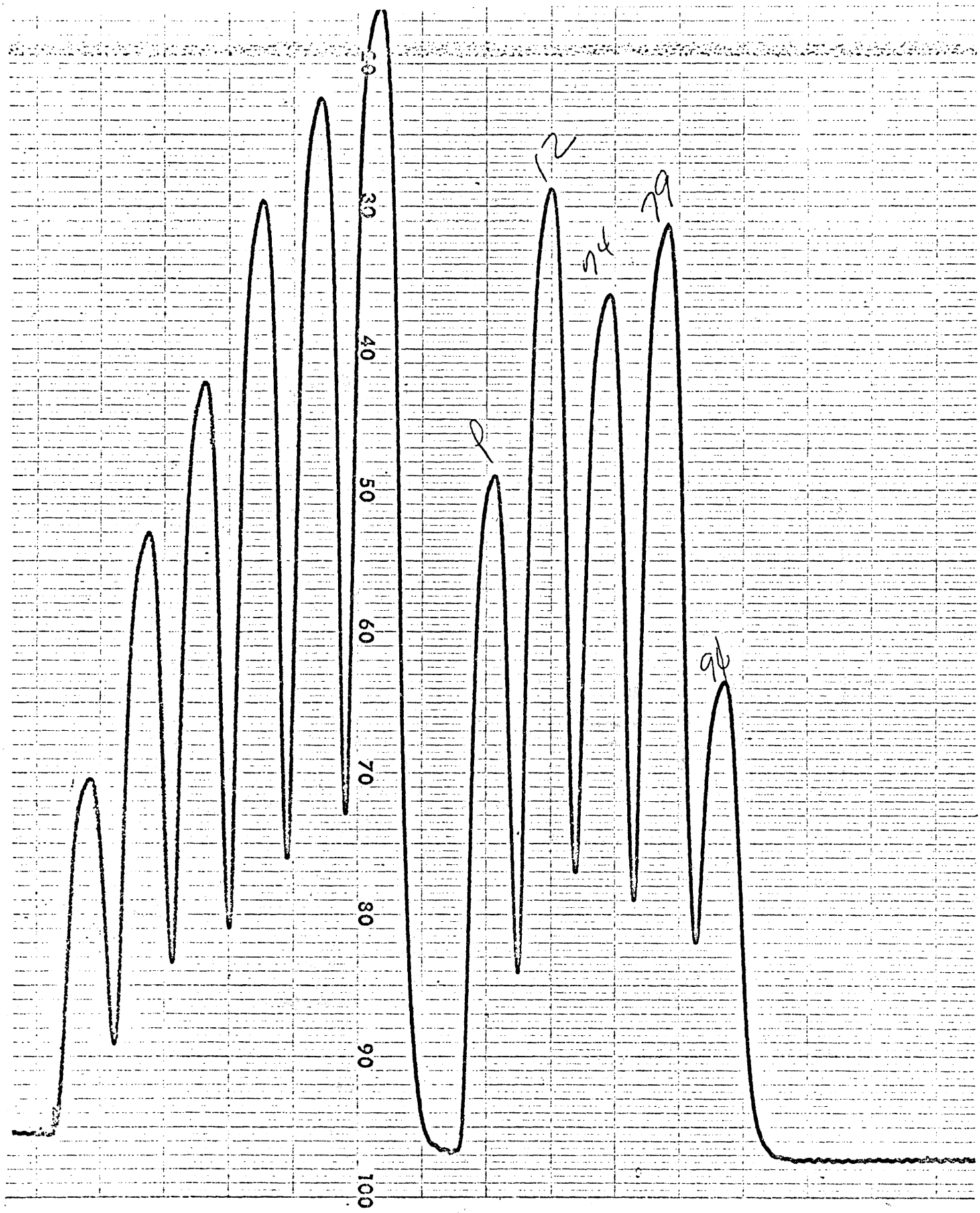
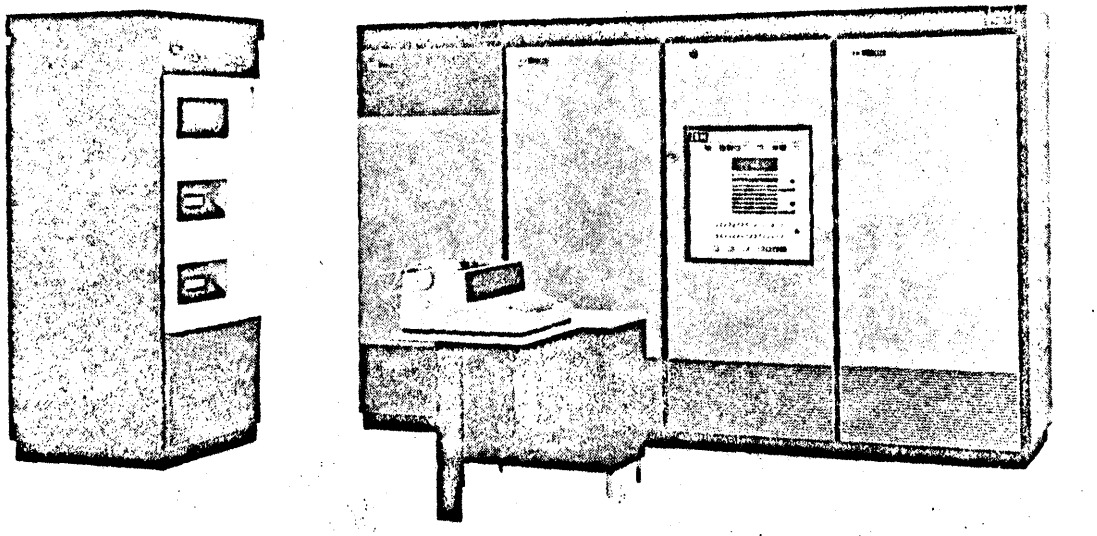


FIGURE 8
IBM 1800 COMPUTER



IBM 1800 Data Acquisition and Control System

FIGURE 9
FLOW DIAGRAM

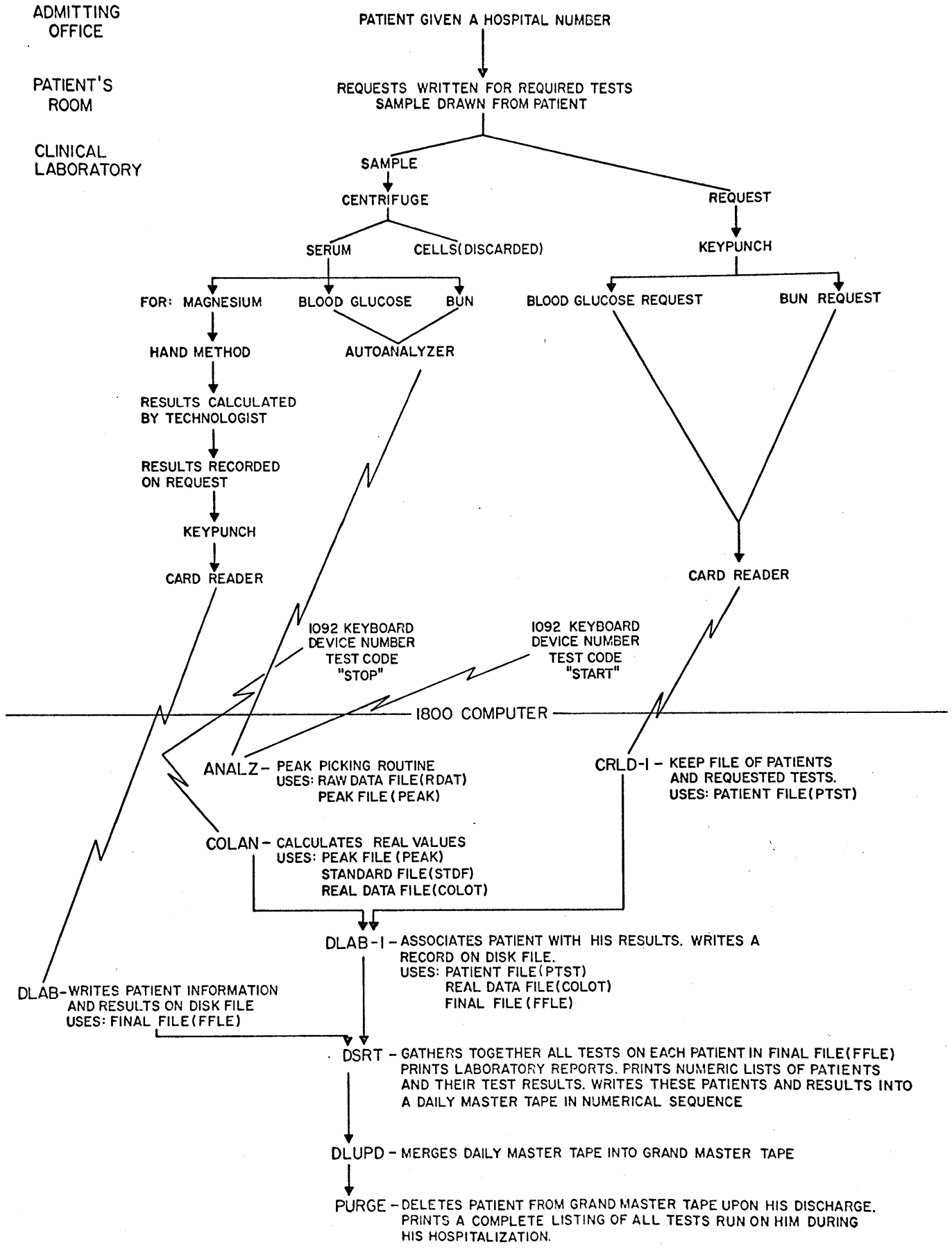


FIGURE 10
1092 KEYBOARD DEVICE

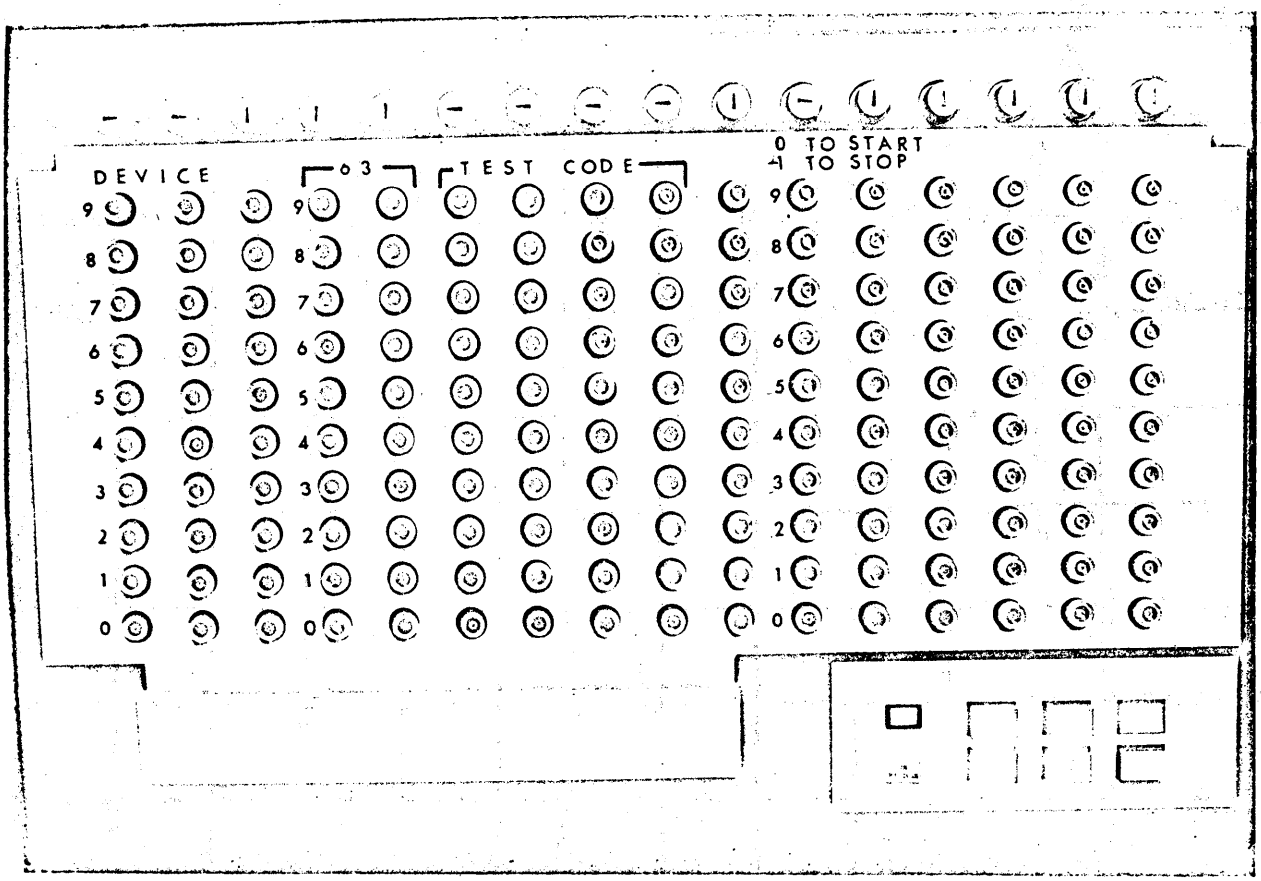


FIGURE 11
FILE: RDAT

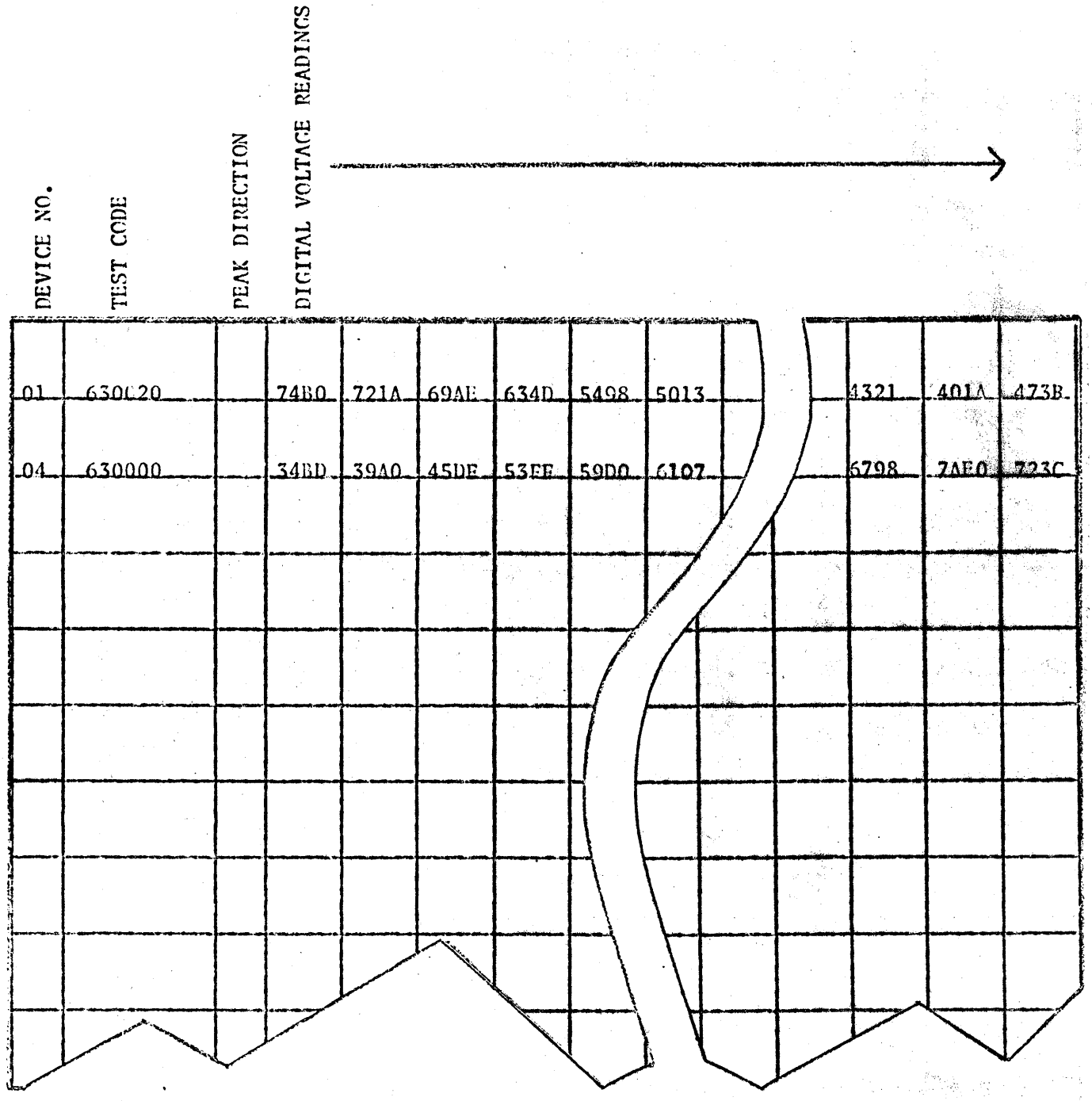


FIGURE 12
FILE: PEAK

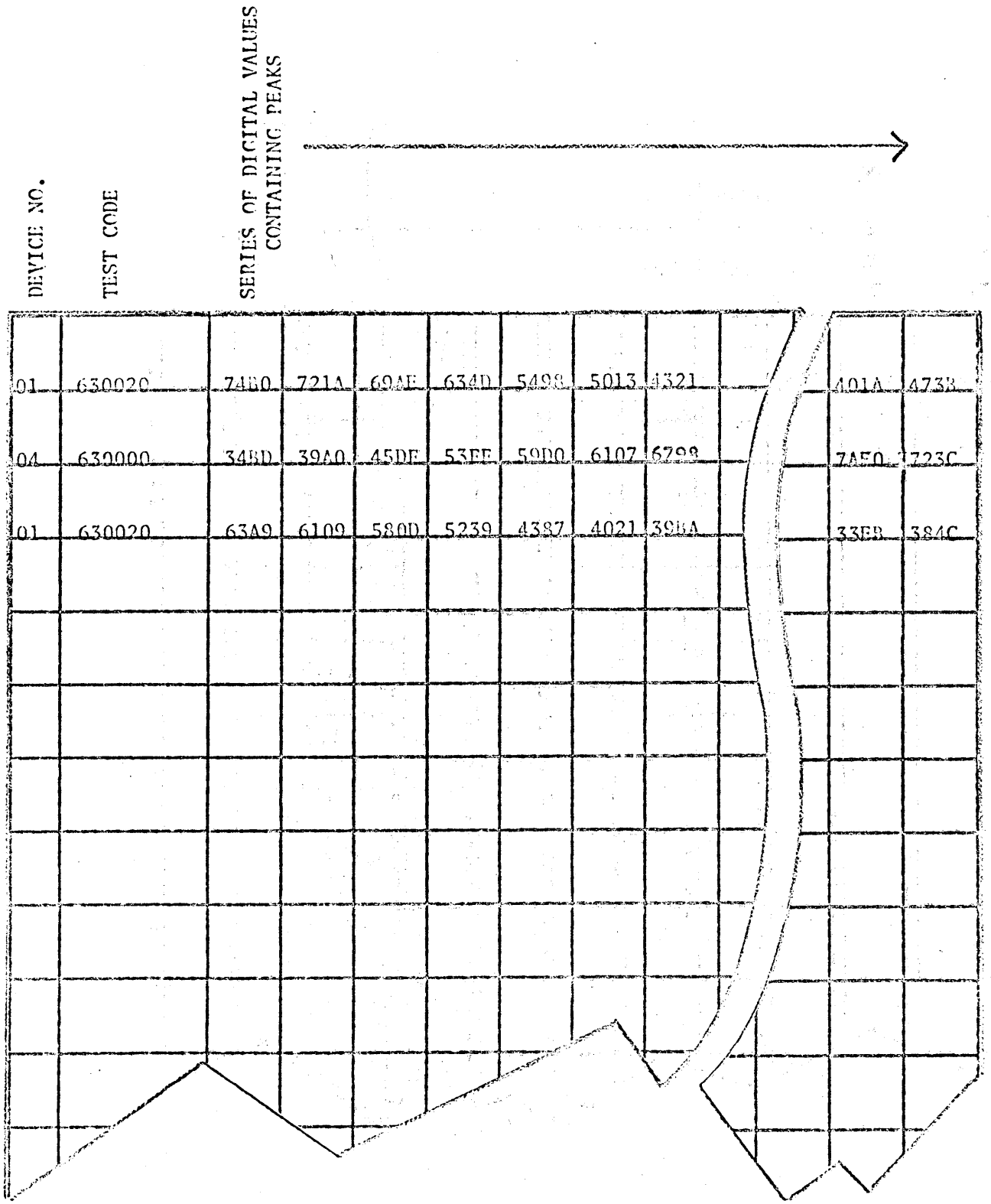


FIGURE 13
FILE: COLOT

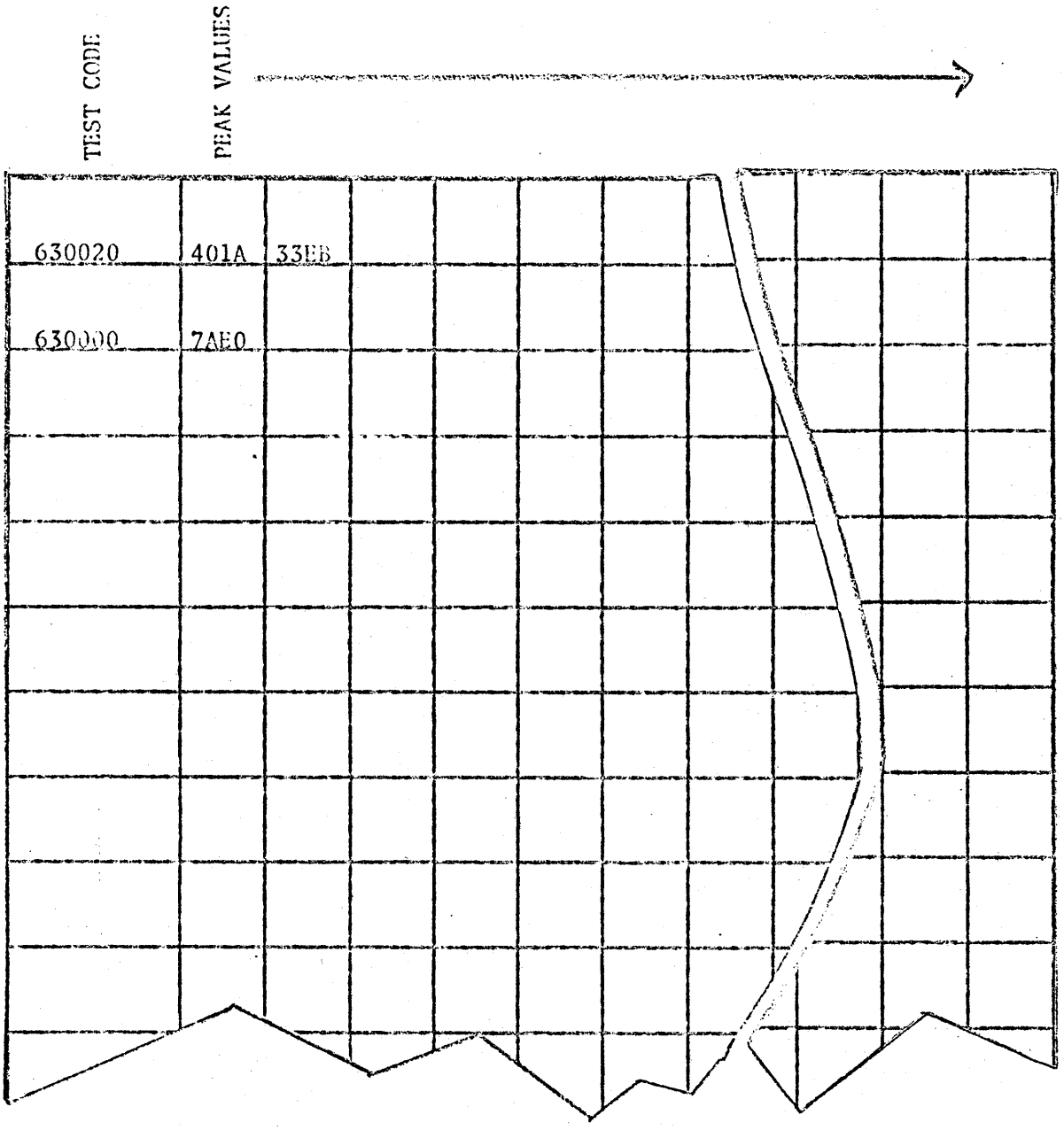


FIGURE 14
TEST CARD

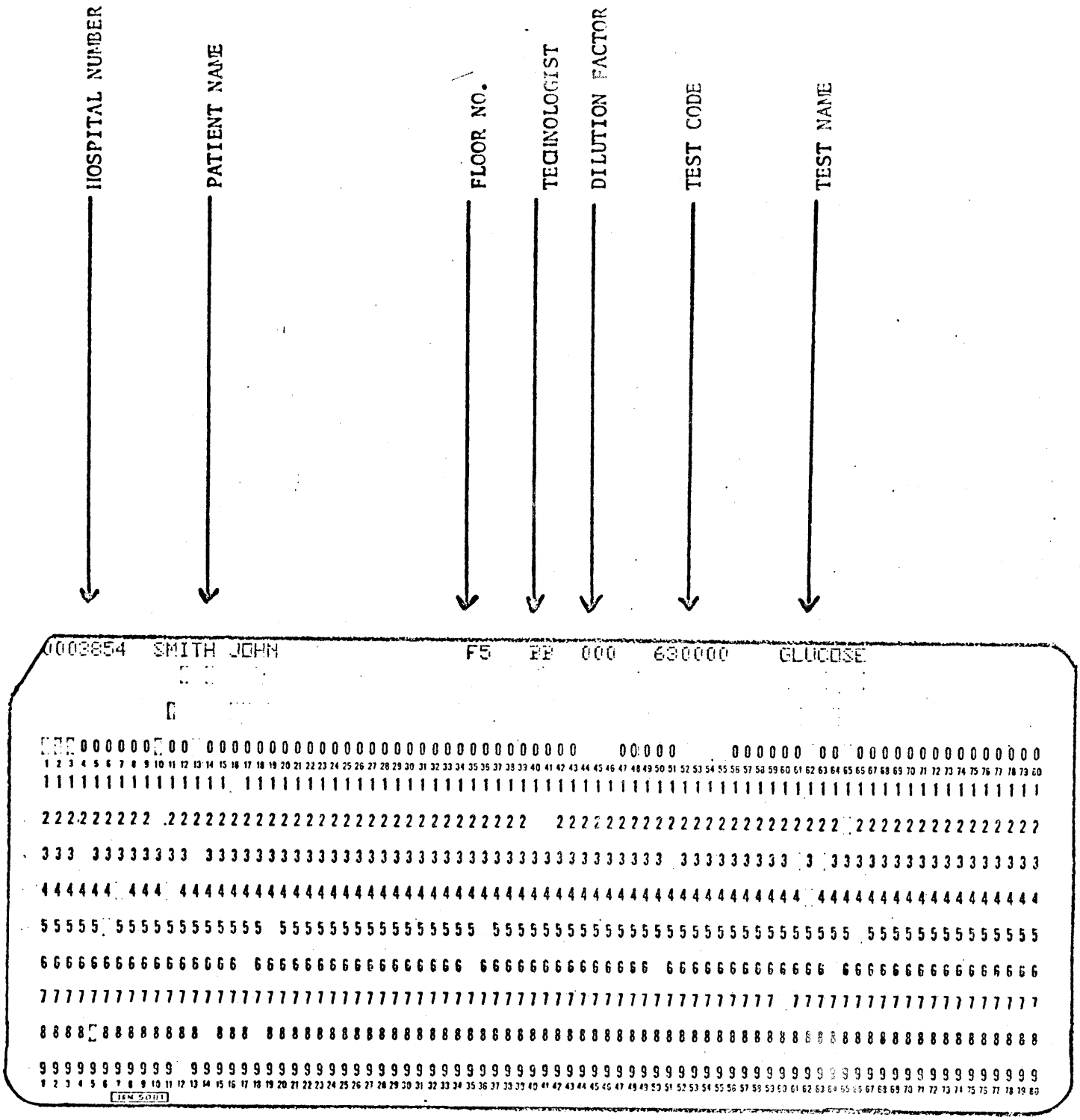


FIGURE 15
FILE: PTST

TEST CODE	TEST NAME	HOSPITAL NO.	PATIENT NAME	PATIENT LOCATION	TECHNOLOGIST	DILUTION FACTOR	UNITS OF TEST
630020	BUN	00-03-85-4	SMITH, JOHN	E5	JAH	0	MGS/100ML
630020	BUN	09-26-54-3	KELLY, SHARON	E6	JAH	0	MGS/100ML
630020	BUN	06-24-03-2	JOHNSON, BILL	E4	JAH	0	MGS/100ML
630000	BL GLU	00-03-85-4	SMITH, JOHN	E5	NRH	0	MGS/100ML
630000	BL GLU	09-26-54-3	KELLY, SHARON	E6	NRH	0	MGS/100ML
630000	BL GLU	07-91-82-1	LAMB, EDWIN	E5	NRH	0	MGS/100ML
630000	BL GLU	02-36-24-9	NAKEN, CAROL	E8	NRH	0	MGS/100ML
630040	CREAT	06-24-03-2	JOHNSON, BILL	E4	RST	0	MGS/100ML

FIGURE 16
FILE: FFLE

HOSPITAL NO.	PATIENT NAME	PATIENT LOCATION	DATE OF TEST	TEST CODE	TEST NAME	DILUTION FACTOR	TEST RESULT	UNITS OF TEST	TECHNOLOGIST
00-03-85-4	SMITH, JOHN	E5	3-11	630020	BUN	0	13.8	MGS/100	JAIL
09-26-54-3	KELLY, SHARON	E6	3-11	630020	BUN	0	20.3	MGS/100	JAIL
06-24-03-2	JOHNSON, BILL	E4	3-11	630020	BUN	0	18.7	MGS/100	JAIL
00-03-85-4	SMITH, JOHN	E5	3-11	630000	BL GLU	0	125.0	MGS/100	NPH
09-26-54-3	KELLY, SHARON	E6	3-11	630000	BL GLU	0	169.0	MGS/100	NPH
07-91-82-1	LAMB, EDWIN	E5	3-11	630000	BL GLU	0	88.0	MGS/100	NPH
02-36-24-6	NAKEN, CAROL	E8	3-11	630000	BL GLU	0	103.0	MGS/100	NPH
06-24-03-2	JOHNSON, BILL	E4			CREAT	0	3.2	MGS/100	PST

FIGURE 17
PRINTED OUTPUT

000385	SMITH JOHN		F5	11.6 HRS	03/11/68
000385		BUN	14	MG/100ML	JAH
000385		BL GLUC	125	MG/100ML	NRH
000385		MAGNESIUM	2.5	MG/100ML	BRL

PATHOLOGIST

079182	LAMB EDWIN		F5	11.6 HRS	03/11/68
079182		BL GLUC	88	MG/100ML	NRH

PATHOLOGIST

092654	KELLY SHARON		F6	11.6 HRS	03/11/68
092654		BUN	20	MG/100ML	JAH
092654		BL GLUC	169	MG/100ML	NRH

PATHOLOGIST

023624	NAKEN CAROL		F8	11.6 HRS	03/11/68
023624		BL GLUC	103	MG/100ML	NRH

PATHOLOGIST

FIGURE 17 CONTINUED
PRINTED OUTPUT

042068	LINDSEY BARBARA		F8	11.6 HRS	03/11/68
042068		HEMOGLOB	15.4	GM/100ML	RLT
042068		URINALYS			DRB
042068	URI PH	7.5	PH UNITS		
042068	URI GLUC	NEG	COMBISTIX		
042068	URI PROT	2+	SULFOSAL ACID		
042068	URI RBC	20-30	/HPF		
042068	URI WBC	10-15	/HPF		
042068	CASTS	RARE HYALINE CAST			
042068	MISC	FEW BUDDING YEAST SEEN			

PATHOLOGIST

062403	JOHNSON BILL		F4		11.6 HRS	03/11/68	
062403		BUN	19	MG/100ML		JAH	
062403		CREAT	3.2	MG/100ML		RST	
062403		CUL ROUT				JCL	
062403	SPECIMEN OF	SPUTUM	CONTAINS	STAPH AUREUS	COAG POS		
062403	KEFLIN	S	KANAMYCIN	S	PENICILLIN		R
062403	STAPHCILLIN	R	OXACCILLIN	R	POLYCILLIN		S
062403	OXYTETRACYCL	S	CHLORAMPHENI		ERYTHRONYCIN		S
062403	COLYMYCIN		NEOMYCIN	R	STREPTOMYCIN		S
062403							

PATHOLOGIST

FOOTNOTES

1. See the paper "Data Reduction Techniques in a Clinical Laboratory" by R. O'Desky, M. Ball, and W.B. Stewart (to be published).

2. The 1800 has the inherent capability of using more sophisticated interpolation schemes; however, linear interpolation between standards appears adequate for the laboratory's present needs.

3. When a complete Hospital Information System is introduced into the hospital, all of these files could be read directly from the 1800 computer into a larger, centralized hospital computing center.

SESSION REPORT

COMMON - Chicago

Session Number **TUE C3** Session Name **Direct Digital Control**

Chairman **R. P. Walker**

Time **1:30 to 3:00 PM** Attendance (No.) **23**

Speakers **R. Pomerance - IBM**

Synopsis of Meeting **Discussion of Direct Digital Control program developed by IBM. Documentation is presently in PID and should be released soon. Dick discussed the various options available and showed sample problems. He passed out the attached sheets showing the options. He also showed slides of an actual application by IBM in San Jose and emphasized how the process was controlled through an operator's panel which included an IBM 1892 console.**

PVR FIELDS COMMON TO ALL CALCULATIONS

LABEL	WD	BITS	DESCRIPTION
IDENT	0	00-15	PVR IDENTIFICATION
LPOUT	1	0	1 IF LOOP IN SERVICE IF NOT ,VALUES CAN BE DISPLAYED, BUT NO CALCULATIONS WILL BE MADE
ADCBT	01	01	1 = ADC READING REQ,D
GTRW	01	02-03	MEAS SOURCE IF NOT ADC RDG 0 = MEAS, THIS PVR 1 = MEAS, ANOTHER PVR 2 = OUTPT, ANOTHER PVR 3 = TABLE IN CORE
ALGOP	1	4-07	ALGORITHM OPTION CODE 0 = MEAS/CONVERSION ONLY 1 = MEAS/CONVERSION/LIMIT CHECK 2 = P+I CONTROL CALCULATION 3 = P+I+D CONTROL CALCULATION 4 = P+I+NL CONTROL CALCULATION 5 = P+I+ERR BIAS CONTROL CALC 6 = P+I+OUTPT BIAS CONTROL CALC 7 = RATIO CONTROL CONTROL CALC 8 = WGTD SUM CALCULATION 9 = MATERIAL INTEGRATOR A = USER FUNCTION B = ARITH CHECK CALCULATION C = OUTPUT TRACKING FUNCTION D = DAC/COS CHECK CALCULATION E = ADC CHECK CALCULATION F = COLD JUNCTION COMPENSATION
RECSZ	1	08-15	WORD COUNT OF THIS RECORD
POLLT	02	00-03	CYCLE TIME = 2**POLLT SECONDS
FAZE	02	04-07	PHASE TIME IN SECONDS
INFRQ	02	08	LINEARIZE FOR CONTROL CALC=1
INSUB	02	09-11	INPUT SUBROUTINE OPTION 0 = NO CONVERSION SUBR 1 = ZERO CHECK/ADJUST 2 = ZERO CHECK/FLOW CONVERSION 3 = PT-PTRH13 T/C CONV(50 MV) 4 = FE-CN T/C CONV (50 MV) 5 = CR-AL T/C CONV (50 MV) 6 = UNDEFINED 7 = UNDEFINED
DCN1	02	12	1 = DISPLAY CONSOLE 1
DCN2	02	13	1 = DISPLAY CONSOLE 2
ECN1	02	14	1 = ENTER CONSOLE 1
ECN2	02	15	1 = ENTER CONSOLE 2

PVR FIELDS COMMON TO ALL CALCULATIONS

LABEL	WD	BITS	DESCRIPTION
INMAD	03	00-15	MEAS ADDRESS WORD = MPX ADDRESS, IF ADCBT = 1. IF ADCBT = 0, USE THE FOLLOWING = PVR IDENT, IF GTRW =1,2. = TABLE INDEX, IF GTRW = 3.
EDTCH	05	00-02	DEC POINT FOR ENG DISP 0 = XXXX. 1 = XXX.X 2 = XX.XX 3 = X.XXX 4-7 = .XXXX
INDIT	05	03	1 = DISPLAY IN ENGINEERING UNITS
PUNIT	05	04-07	UNITS CODE 0 = BLANK 8 = LB/M 1 = PERCNT 9 = PSIG 2 = DEG. F A = RATIO 3 = GPM B = 4 = MCFH C = 5 = IN H2O D = 6 = IN HG E = 7 = MV F =
CONA	06	00-15	CONA = 2 * SPAN FOR ENGR SCALE
CONB	07	00-15	CONB = ZERO VALUE FOR ENGR SCALE
MEAS	08	00-15*	VALUE OF MEAS AFTER CONVERSION

PVR DESCRIPTION - TYPE 2 - P+I CONTROLLER CALCULATION

LABEL	WD	BITS		DESCRIPTION
INBAD	04	00-01	*	COUNTER FOR INPUT BAD SUBR
RTCH	04	02-03		RATE OF CHANGE LIMIT 0 = NO TEST 1 = 50 PCNT 2 = 10 PCNT 3 = 5 PCNT
FLTYP	04	04-07		FILTER OPTION 0 = NO FLTR 1 = DCU FLTR 2 = EXP. FLTR 3 = NO FILTR
FLCOT	04	08-15		FILTER CONSTANT = 256*BETA
SAUD	05	08	*	1 = HORN HAS BEEN SOUNDED, RESET ALARM BUTTON NOT PUSHED
LPALM	05	09	*	1 = VARIABLE IN ALARM
ALMOR	05	11	*	ALARM MESSAGE ORIGINATED
LPMWT	05	12-15	*	ALARM MESSAGE WAITING B12 1= NORMAL MESSAGE WAITING B13-14 0= NO ALM MSG WAITING 1= INPUT MESSAGE WAITING 2= ERROR MESSAGE WAITING 3= OUTPUT MESSAGE WAITING B15 0= LO ALARM MESS. 1= HI ALARM MESS
INAST	09	00-01	*	INPUT ALARM STATUS B0 = 1, INPUT IS IN ALARM B1 = 0, LOW ALARM B1 = 1, HIGH ALARM
INADB	09	02-03		INPUT ALARM DEADBAND 0 = 0.1 PCNT 2 = 5.0 PCNT 1 = 1.0 PCNT 3 = 10. PCNT
INAHI	09	04-07		ALARM ACTION, INPUT HIGH B4 = 1, PUT ON MANUAL B5 = 1, HOLD OUTPUT B6 = 1, SOUND AUDIBLE ALARM B7 = 1, PRINT ALARM MESSAGE
INALO	09	08-11		ALARM ACTION, INPUT LOW B8 = 1, PUT ON MANUAL B9 = 1, HOLD OUTPUT B10= 1, SOUND AUDIBLE ALARM B11= 1, PRINT MESSAGE
INCNT	09	14-15	*	COUNTER FOR LIMIT,ALARM CHECK
INHIL	10	00-15		INPUT HIGH LIMIT

PVR DESCRIPTION - TYPE 2 - P+I CONTROLLER CALCULATION

LABEL	WD	BITS	DESCRIPTION
INLOL	11	00-15	INPUT LOW LIMIT
SETPT	12	00-15	VALUE OF SETPT
SPCHI	13	00-15	VALUE SET CHG + LIMIT
SPCLO	13	00-15	VALUE SET CHG - LIMIT
ERRHI	14	00-15	ERROR HIGH LIMIT
ERRLO	14	00-15	ERROR LOW LIMIT
ERAST	15	00-01*	ERROR ALARM STATUS B0 = 1 ERROR IS IN ALARM B1 = 0, LOW ALARM B1 = 1, HIGH ALARM
ERADB	15	02-03	ERROR ALARM DEADBAND 0 = 0.1 PCNT 2 = 5.0 PCNT 1 = 1.0 PCNT 3 = 10. PCNT
ERAHI	15	04-07	ALARM ACTION, ERROR HIGH B4 = 1, PUT ON MANUAL B5 = 1, HOLD OUTPUT B6 = 1, SOUND AUDIBLE ALARM B7 = 1, PRINT ALARM MESSAGE
ERALO	15	08-11	ALARM ACTION, ERROR LOW B8 = 1, PUT ON MANUAL B9 = 1, HOLD OUTPUT B10= 1, SOUND AUDIBLE ALARM B11= 1, PRINT ALARM MESSAGE
OUTPT	16	00-15*	VALUE OF OUTPUT
OUTHI	17	08-15	OUTPUT HIGH LIMIT
OUTLO	17	00-07	OUTPUT LOW LIMIT
OUTAD	18	00-15	OUTPUT ADDRESS WORD IF MASTER LOOP,OUTAD HAS SLAVE ID IF OUTPUT TO COS,
COSGP	18	04-07	LOGICAL COS GROUP
COSBT	18	08-11	PULSE BIT FOR COS
XALB	18	08-11	EXTERNAL ALARM LIGHT BIT
XALW	18	12-15	LOGICAL ECO FOR EXTERNAL ALARM IND

PVR DESCRIPTION - TYPE 2 - P+I CONTROLLER CALCULATION

LABEL	WD	BITS		DESCRIPTION
OUAST	19	00-01	*	OUTPUT ALARM STATUS B0 = 1, OUTPUT IS IN ALARM B1 = 0, LOW ALARM B1 = 1, HIGH ALARM
OUADB	19	02-03		OUTPUT ALARM DEADBAND 0 = 0.1 PCNT 2 = 5.0 PCNT 1 = 1.0 PCNT 3 = 10. PCNT
OUAHI	19	04-07		ALARM ACTION,OUTPUT HIGH B4 = 1, PUT ON MANUAL B5 = 1, HOLD OUTPUT B6 = 1, SOUND AUDIBLE ALARM B7 = 1, PRINT ALARM MESSAGE
OUALO	19	08-11		ALARM ACTION, OUTPUT LOW B8 = 1, PUT ON MANUAL B9 = 1, HOLD OUTPUT B10= 1, SOUND AUDIBLE ALARM B11= 1, PRINT ALARM MESSAGE
OUTCH	19	12-13		OUTPUT CHANGE LIMIT 0 = 100 PCNT 2 = 5 PCNT 1 = 10 PCNT 3 = 1 PCNT
OTOK	19	14	*	1 = OUTPUT IN SERVICE
EXAM	19	15	*	1 = EXTERNAL CONTROL
MAUTO	20	00	*	1 = LOOP IN AUTO
MAULA	20	01	*	AUTO/MAN STATUS LAST TIME
STUPC	20	02		STARTUP OPTION 0 = START UP ON MANUAL 1 = START ON LAST CONDITION
CABSL	20	03		1 = CAN BE SLAVE
ISSLA	20	04	*	1 = IS SLAVE
MASTL	20	05		1 = MASTER LOOP
MASLK	20	06-07		OUTPUT LOCATION OPTION,MASTER PVR OUTPUT OF THIS LOOP GOES TO 0 = NOWHERE 2 = BIAS 1 = SETPT 3 = KP OF ANOTHER LOOP
REVAC	20	08		1 = REVERSE ACTION CONTROLLER
KPROP	21	00-15		PROPORTIONAL CONSTANT = GAIN*128
KINTG	22	00-15		INTEGRAL CONSTANT = GAIN*1024
RST1	23	00-15	*	WORD 1 OF RESET TERM
RST2	24	00-15	*	WORD 2 OF RESET TERM

SESSION REPORT

COMMON - Chicago

Session Number TUE C4 Session Name LP/MOSS Tutorial

Chairman W. A. Pease, Jr.

Time 1:30 to 3:00 PM Attendance (No.) 92

Speakers Muller of IBM

Synopsis of Meeting After a scramble for an overhead projector, the session proceeded with the presentation of the Aluminum Alloy Blending Problem from the LP/MOSS manual.

The presentation was too advanced for most of the newer users as only 29 returned for the continuation in Session Tues - E1 later.

SESSION REPORT

COMMON - Chicago

Session Number Tuesday C-5 Session Name Software Development

Chairman Paul Bickford House Repres. and Users

Time 1:30 to 3:00 P.M.

Speakers Evan Linick, Bob White, Don Weber, Roland Magee

Synopsis of Meeting Evan pointed to good reasons for permitting "Software houses" to make presentations without making COMMON a "Side Show". Bob pointed out the complexion of software development has changed and COMMON should permit Software houses to fill in what they cannot get with the traditional approach to developing systems. Ron raised the question of whether we should let software houses make Presentation when there is not time for member presentations. Don pointed out that one may want to give part of our program. The Meetings was well attended (75 people) and went smoothly. It was terminated by requesting those present who had opinions pro and con about the subject being discussed were asked to forward them to the Program Chairman.

Introductory statement made by Evan F. Linick, Manager, Research and Development, Software Resources Corporation at COMMON Panel Discussion (Session Tue-C5), April 9, 1968, Chicago, Illinois.

Statement of Topic

There has been some confusion regarding the exact topic that we are to discuss today. Although the meeting agenda indicates a slightly different subject, what had been agreed upon as a discussion topic was "Should software companies be allowed to make presentations at COMMON meetings." I believe this question is really part of a more general one: "What should be the relationship of COMMON to software companies (and perhaps even hardware companies) which offer products and services of direct interest to COMMON members?" My remarks will be directed toward this more general question.

Background

One of the reasons why we are discussing such a topic today has to do with the large number of software companies which have come into existence over the last several years. There are now at least 500, and perhaps well over 1,000 such companies specializing in programming services - principally contract programming. Of immediate interest to us is the latest innovation of these companies, the marketing of software packages. (Of course, the marketing of software packages today is not limited to software companies, many individual users are also attempting to market packages which they have developed.)

There is not time during this session to go into the details of why this has happened or the different types of packages which are currently available. Very briefly - there are today, many packages available in the areas of systems and applications software, at prices averaging about 10% of the cost of development.

While discussing this background, there are a few side comments which I would like to make. Today, there are many companies (and not all of these are software companies) whose existence depends on the programming and use of IBM computers and who do not qualify for membership in COMMON. COMMON may want to consider a distinction that perhaps did not exist previously - the distinction between being an IBM user group and an IBM customer group. Also, as mentioned previously, there are today many hardware companies offering useful products to IBM users. These products include such items as peripheral equipment, terminals, communications gear, etc. The remarks made here today relative to software companies may apply in many cases to these hardware companies as well.

How does this relate to COMMON?

COMMON is composed of organizations - companies, educational institutions, government agencies, etc. Attendees at COMMON sessions are representing these organizations and their interests. The basic purpose of COMMON and other organizations of its type has been to exchange information of mutual interest, to benefit by common experience, to save time and money, and to use computers more profitably. One way in which these basic aims have been pursued has been to provide a means of free exchange of computer programs. The question that arises now is whether the commercial availability of such programs conflicts with the free exchange promoted through COMMON. In this regard I would like to suggest that although exchange of programs has been of benefit in some cases, it has generally not been a successful method of achieving the desired goals. There are a number of reasons for this:

1. Users have discovered that programs obtained through exchange are not really free - there is often a considerable cost associated with making these programs operational.
2. Despite efforts that are sometimes (but not always) made, there are almost always serious problems associated with screening and verifying the quality of programs submitted to a user exchange library and assuring that adequate documentation is available.

3. There is usually little or no support available with programs obtained through exchange. In fact, there is often a great reluctance on the part of those having good programs to submit them to an exchange library because of the nusiance and cost involved in answering requests for additional support from those obtaining the system.
4. Many users with excellent programs don't want to give away something that cost them many dollars to develop. In almost all cases program libraries have not provided equal value to all of those participating.

The ability to purchase software packages is something that is quite new. It provides another possible answer to the problem presented by our luncheon speaker today. As he pointed out, the availability of good programs in a vast variety of areas is going to be a significant problem to all users over the next several years. Because of this, the commercial availability of package programs should be of extreme interest to all COMMON members and it is certainly appropriate for COMMON members at COMMON meetings to exchange information on this alternative resource.

What might be appropriate for COMMON?

There are a number of ways in which COMMON may appropriately attempt to exchange information about commercially available software packages. These include:

1. State-of-the-art papers or discussions.
   - Types of programs available.
   - Sources of packaged programs.
   - When to consider using a package.
   - How to evaluate a software package.
   - User experience with purchased software.
2. Current controversial items.
   - Protection of proprietary programs.
   - Separate pricing of software and hardware.

3. Presentations of specific packages.
   - Technical, non-sales oriented, presentations similar to those currently presented on user developed applications. (There is some precedent for this in Guide and Share as Bob White will discuss.)
   - Sales oriented presentations conducted concurrently with, but independently of, COMMON sessions.
4. Development of applications and systems specifications.
   - Development of specifications that could grow into joint ventures with software companies outside of COMMON.
   - Specifications submitted to IBM may or may not get a satisfactory response. Specifications made to a software company many result in a good product, well-documented and supported, available quickly, and because of these reasons worth the price associated with them. (Bob White's remarks should shed further light on this approach.)

How should this be done?

The specifics of how to proceed along these lines is up to COMMON. My main point today is that COMMON should not avoid the problem and hope it goes away - no matter what position COMMON may want to adopt, there should be some definite policy statements made as quickly as possible.

Some approaches that should be considered include:

1. Presentations made at COMMON meetings could be on request or pre-screened.
2. Presentations could be made in regular COMMON sessions, or in special sessions, or in private sessions held concurrently but independently of COMMON meetings.
3. A special bulletin board could be provided at COMMON meetings where information on commercial products may be posted or announcement of the availability and location of representatives of software companies could be posted.

Individual COMMON members would then be free to follow up these notices at their own discretion.

If COMMON does seriously consider these matters and act quickly to provide themselves and the software companies with specific guidelines, all of this can be done without over commercialization of COMMON meetings or unnecessarily distracting commercial side shows. Exchange of information on commercially available software packages if done in this way will certainly be in the better interests of COMMON, COMMON members, software companies and even IBM itself.

SHOULD COMMON ALLOW SOFTWARE HOUSES MAKE PRESENTATIONS AT OUR MEETINGS?

Session Tuesday C-5

R. H. Magee

I think there are probably quite a few reasons why we should not, but I am only going to discuss two of them. I believe these are practical reasons which should be considered.

The first is the question of time. I feel that we currently have enough topics to cover without trying to fit in other types of presentations. There were more papers presented for this meeting than could be used; and there is talk of extending the Philadelphia meeting to four days to cover the topics. I, for one, do want to see these meetings extended for four days.

The second reason seems to fit logically with the first and that is one of interest. We presently are committed for the 1620, 1130, 1800 and at least 4 models of the 360. To me, this is a lot of machines to cover. Not only hardware, but also software wise. If COMMON is to present meetings of interest to this wide range of installations, then I do not see how there is going to be time left for the software houses.

I do not see how a software house can make a presentation which would be of interest to enough users to justify the scheduling of such a presentation.

The program chairman has a hard enough job to get the revelant papers and topics in the schedule without placing a further burden on him.

I believe that these two reasons alone are enough to eliminate the presentations by Software Houses.

Thank you.

R. H. Magee

SESSION REPORT

COMMON - Chicago

Session Number TUE C7 Session Name Computer Graphics and Numerical Control

Chairman Joe Talkington

Time 1:30 Attendance (No.)

Speakers Joe Talkington

Ed Becker

Synopsis of Meeting Mr. Becker talked about IBM support for N/C users with IBM 1130's. Most of the presentation was on the Type 3 program entitled "Romance".

Joe Talkington discussed a government sponsored NDEA Institute entitled "Computer Graphics".

SESSION REPORT

COMMON - Chicago

Session Number TUE C8 Session Name Laboratory Auto-

Chairman G. Hertel mation - Petrochemical Project

Time 1:30 to 3:00 PM Attendance (No.)

Speakers B. Polishok, IBM and D. Hutchins, Procter and Gamble

Synopsis of Meeting (1) Polishok - "New 1800 Labs Monitor System"

(2) Hutchins - " Procter & Gamble's Lab Automation System"

Laboratory Automation talk given by B. H. Polishook at COMMON on April 9, 1968 to the Petrochemical Project

I am pleased to be here and have the opportunity of talking about what I believe to be one of the fastest growing areas of interest to the COMMON organization today. This is the area of laboratory automation wherein one can find numerous 1130's, 1800's, Model 44's and other System/360 computers interacting with laboratory instrumentation. The exciting thing about this whole field is that we are rapidly growing out of the data logging stage of just acquiring data and are beginning to do things in the laboratory which would have been impossible without the computer.

Due to the growing interest in laboratory automation, we felt a forum for users should be established. As a result, we have been trying to stimulate the interest of the COMMON organization to establish a continuing project in this area. Today's session represents the third time in a row that talks have been given on laboratory automation. The first time was in Cincinnati in September of 1967. There we raised the original question of establishing a continuing group in laboratory automation. Also, Ray Edwards of my staff outlined the general area of laboratory automation and discussed various approaches and progress to date. The second session was in San Francisco last December. There

Dr. Henry Gladney of IBM's San Jose Research Laboratory discussed his laboratory system. Again, Ray Edwards provided the group with a progress report.

You will note that the speakers at these sessions have come solely from IBM. We have always hoped that such support would be temporary to solicit initial interest. The activity must now become self-sustaining with the direction coming from the members of the COMMON organization. COMMON can offer an excellent forum for the users of laboratory automation systems. It can offer a point of contact in discussing programming approaches, application programming needs, and to exchange programs. This is vital to the growth of laboratory automation activities. I hope therefore that at the end of this COMMON Meeting you will discuss these sessions with your laboratory personnel and ensure that there is a continuing interest and desire to sustain laboratory automation activities in COMMON.

My purpose in being here today is not only to discuss the need for continued laboratory automation support within COMMON but also again to provide you with a progress report. As previously mentioned, progress reports were given in the last two sessions. One would suspect that progress cannot continually be made on such a short term basis. However, this is not the case in laboratory automation as we are presently seeing a revolution as opposed to an evolution. The most interesting

thing that has occurred over the last three months has been the 19th Annual Pittsburgh Analytical Chemistry and Applied Spectroscopy Conference. This is a technical conference which started out as a local event in Pittsburgh. Their success in putting on these annual conferences grew to the point where it became the conference of the year for presentations of original papers and for the announcement of new products. This year, due to labor relations problems, a new home had to be found for the conference. It was therefore placed in Cleveland. In looking at the computer papers given at the conference as a percentage of the total papers given, an interesting trend can be observed over the years---------in 1963, 1.8% of the papers were computer oriented; in 1964, 3.2%; in 1965, 9.3%; in 1966 it dropped to 3.5%. In 1967, it was up to 13.5% and in 1968 it was down to 8.4%. These statistics suggest that the computer within the laboratory is being recognized on a par with the instrumentation in the laboratory. In fact, one can look at the computer as if it were an instrument itself. In this particular case, it is an instrument which can enhance the data acquisition function of other instruments and can utilize sophisticated mathematical techniques to analyze data to provide meaningful information. It can manage this information for display, graphics or information retrieval purposes and finally can manage an experiment from one operating level to another based on previously analyzed data.

In 1966, two computers were shown at the conference, in 1967 there were about 12 computers and this year there were 19 computers including an 1130 and 1800, which were part of the IBM exhibit.

I would now like to discuss with you what the 1130 and 1800 were doing at the conference. The 1130 was used for infrared spectral data retrieval. Infrared spectroscopy is a most useful analytical technique to identify compounds and to determine their specific composition. The technique involves an infrared energy source radiating through gases, vapors, or thin film solids. The absorption of infrared energy at a particular wave length results from the presence of a particular (or a class of) chemical compounds. A typical result from such an infrared experiment would be a plot of percent transmittence of the radiation versus the wave length at which the transmittence occurred. The reciprocal of transmittence is, of course, the absorption of the infrared energy. In turns out that each compound has a characteristic transmittence curve which could be used as a fingerprint of the compound. Therefore, one need only match the curve of an unknown sample to that of a known sample to identify the unknown sample. However, the problem is not that simple. There are over 125,000 infrared standards in existence today and the task of matching is quite laborious.

The program on the 1130 was written by Dr. Duncan Erley of the Dow Chemical Company in Midland, Michigan. In this particular

system some 40,000 compounds and significant information about their spectral data characteristics were put on a disk. Each compound entry contained such information as its serial number, its elemental grouping, and finally several words devoted to the transmittence of infrared energy at specific wave lengths. The process of identifying a sample under investigation was started by its analysis on a spectrophotometer provided by Perkin-Elmer. This provided a strip chart recording showing transmittence versus wave length. Information on the transmittence curve such as where the absorbence occurred and where it did not occur was then keyed into the 1130. The matching process was then initiated. The searching rate is about 1,000 standards per second and so in about 40 seconds the experimentalist could determine the number of potential matches that occurred. If numerous matches occurred, several options allowing greater discrimination in the matching process could be initiated. Once the 1130 had either provided the user with a unique match or a small enough number for him to review personally, he would only have to look up the serial numbers of the matches on ASTM tables. In this way he could determine the identity of the sample under investigation. The ability to search a library of standards is probably one of the most important aspects of laboratory automation. Almost every instrumental technique, whether it be the gas chromatograph, infrared spectrometer, atomic

absorption spectrometer, mass spectrometer, NMR, etc., has a group of standards - each of which are characteristic of a specific compound.

At the Pittsburgh Conference the spectrophotometer was not on line. However, work is now going on to put it directly on line to the 1130. One method is through paper tape. A second method would be directly through the SAC channel. When the equipment is put on line, sophisticated data analysis can be introduced. The data can be smoothed to enhance the signal to noise ratio. This helps in determining the specific point of maximum absorption of energy as opposed to the observable point on the strip chart recorder. It also helps in decreasing the time required for the experiment. For example, the slower that one changes from one wave length to another concurrently with multiple data taking at each wave length, the more precise the initial raw data can be. However, this results in extremely long experiment times. On the other hand, if one sweeps through the experiment at a rapid rate the computer can be utilized to massage the data to provide precise information. This, of course, is one of the great benefits of utilizing the computer. In fact, the utilization of the computer may indeed have advanced instrument accuracy by at least an order of magnitude.

The second part of the IBM demonstration consisted of demonstrating the results of an exploratory effort between the IBM Palo Alto Scientific Center and Varian Associates, also of Palo Alto. This next section of the talk is based upon two of the papers that were available at the IBM exhibit at the Pittsburgh Conference - Papers attached: (1) "A Computer System for Automation of the Analytical Laboratory" by C. H. Sederholm, P. J. Friedl, T. R. Lusebrink, (2) Application Programs for 1800 Laboratory Automation System.

The third part of the IBM exhibit involved a new film covering various customer uses of IBM equipment in the laboratory. (Showing of film).

In summary, I have tried to discuss our interest in making COMMON a forum for laboratory automation computer users and to give you some idea of some of the recent progress in this field. Once again, let me request that each of you talk to your colleagues in your company and advise them of these activities at COMMON.

IBM

Application Programs for 1800 Laboratory Automation System

This paper describes the various application programs demonstrated at the March 1968 Pittsburgh Conference on Analytical Chemistry and Applied Spectroscopy held in Cleveland, Ohio* Although these programs were written to support specific analytical instruments (A-60 NMR Spectrometer, M-66 Mass Spectrometer and two Gas Chromatographs) and instrument-computer interfaces, all manufactured by Varian Associates, the programming system language and applications and control techniques are easily applied to instruments of other makes, models and functions.

Each instrument has associated with it a master program resident on an 1800 disk storage unit. When an experiment is to be performed the master program is loaded into the 1800's core memory by command from the 1800 console typewriter. The function of the master program is to provide control to the instrument-computer interface and to establish a data path between the instrument, the interface and the computer. The various push buttons and indicator lights on the interface enables the user to direct the master program to select and load into core memory specific application programs associated with his experiment. Later versions of the system will allow the user to select and load a master program from either the console typewriter or the interface.

---

*These programs operate under the 1800 Laboratory Automation System described in "A Computer System for Automation of the Analytical Laboratory" by C. H. Sederholm, P. J. Friedl, and T. R. Lusebrink.

Once the master program has been used to load application programs, it automatically is returned to disk storage. Subsequently, the parameters for a specific program can be entered through the use of thumb switches on the interface.

A. Nuclear Magnetic Resonance Spectrometer Programs

The NMR master program arms four push buttons on the interface. These buttons are used to select one or more of the following programs described below:

1. Time Averaging (TA) Data Acquisition Program

This program coordinates the data acquisition activity for the NMR, and allows several options regarding the mode of recording the spectrum. Appropriate parameters are entered by thumb switch on the interface. The parameters consists of the initial and final sweep positions in ppm, sampling rate, number of sweeps, and number of samples to be taken at each value of the field. Time averaging can be carried out by taking multiple readings of the NMR signal at given values of the magnetic field or by taking multiple sweeps.

The spectrometer scan is controlled by transmitting a sequence of digital values of the desired settings for the magnetic field from the 1800 to the interface. The interface converts these digital values to corresponding magnetic fields in the NMR instrument.

The resultant NMR signal is then digitized by the interface and sent back to the 1800.

2. Data Presentation

Several options are available during the data acquisition phase. For example, a sweep may be plotted on the NMR recorder simultaneously with a single sweep of the instrument. Or, if time averaging is being used, it is possible to stop the data acquisition function and have the averaged cumulative curve plotted in order to see whether the data have been sufficiently improved by the time averaging technique. If improvement is noted, the user may terminate the program; if not, the data acquisition function may be restarted.

3. Digital Smoothing Program

In addition to the time averaging technique described above, it is also possible to use digital smoothing methods to improve the signal-to-noise ratio (Savitsky and Golay, Anal. Chem., 36, 1627 (1964)). Upon execution of the smoothing program the data is taken from a disk storage unit and when smoothed written into another area of the disk storage unit.

4. NMR Field Homogeneity Adjustment Program

This program optimizes the curvature and y-gradient adjustments on the NMR instrument in essentially the manner described by Ernst (Pittsburgh Analytical Conference, March 1967). A course grid is calculated to cover ranges of curvature and y-gradient settings. The 1800 then positions the controls at these settings and records the height of a standard NMR peak for each setting. The coarse settings which yield the maximum peak intensity are used to calculate a finer grid of points about the first approximation. The points in this second grid which yield the best signal are then used as the basis for yet another finer grid, and so on until no further improvement is possible.

By pressing the appropriate button the interface, the user can execute this program at any time he chooses, or the program could be called into execution periodically by the NMR master program in order to monitor the field automatically at any desired intervals.

B. Mass Spectrometer Application Programs

The master program for the mass spectrometer arms four push buttons on the mass spectrometer interface. These buttons are used to select one or more of the following four programs:

1. Data Acquisition Program

Parameters relative to the initial mass, mass range, and sampling rate are entered by the user via thumb switches on the interface. The data acquisition program subsequently controls the scan rate and the recording of data. The resultant data are stored on a disk storage unit. An option allows the data to be simultaneously plotted on the spectrometer plotter.

2. Plotting Program

This program allows any portion of the above spectrum to be plotted back on the recorder, with the x-axis expanded to cover the entire recorder range. The initial and final masses are read in via the thumb switches on the interface.

3. Identification of Compound from Spectrum

This program utilizes the input parameter and the raw data on the disk to find all the nominal mass peaks with an intensity greater than a predetermined height. The five highest peaks are determined and normalized so the highest peak equals 100. These five peaks are then used to find the compound in a partial table taken from the ASTM Index of mass spectral data which is stored on the disk. If the compound matches a known spectrum, a report giving the name of the compound and the values for the five peaks is printed on the 1800 line printer.

4. Identification of Compound from Interface

This program allows the user to read in mass and intensity values of the five highest peaks via the thumb switches. The ASTM table is then searched and an identification is made in the same manner as in the previous program.

C. Gas Chromatography Programs

For the purpose of the demonstration, two gas chromatographs were coupled to the computer through a single interface. When the master program is loaded into the computer, it activates the interface so that initial parameters for the system (i.e., bot h chromatographs) may be entered via the thumb switches. These parameters are the total sampling rate for the columns and sensitivity factors for peak detection. The system parameters may be changed for either chromatograph by activating a sense switch which designates data sent from the interface as belonging to that instrument only. The data acquisition program implemented depends on the use of first and second derivatives to sense peaks and inflection points.

Signals from both chromatographs columns are continuously digitized by the interface once the master program has been

loaded into core memory. However, data are not saved until the sample is injected and the start button is activated for a chromatograph. When one or both chromatographs are started, the program sorts the data and stores the data for each instrument in separate areas on the disk storage units. The scan is terminated by depressing the end button.

Analysis of the chromatograms is then carried out by the two programs described below.

1. Peak Definition and Integrated Areas Program

This program uses first and second derivatives to sense the beginning, maximum and end of each peak. The peaks are defined as to type, i.e., single peaks or a leading, connected, or following peak in an unresolved group of peaks. The areas of the peaks are integrated to a base line of zero, so that if the actual base line is above zero or there is a base line drift, further corrections must be made. The area and position data pertaining to each peak are then stored on the disk storage unit for additional processing.

2. Peak Area and Base Line Correction Program

For this demonstration, only two options for the calculation of areas unresolved peaks have been implemented.* The first method is to drop a vertical line from the minimum between two peaks; a reasonably satisfactory estimate if the peaks are moderately well resolved and not too asymmetrical. When relatively small peaks follow a large asymmetrical peak, the user may select the option of an area "slice". This slice is roughly equivalent to smoothing out the larger peak and equating the area of the smaller peak to the area above this smoothed line.

This program also calculates the net peak area by subtracting the trapezoidal area between the actual base line and zero. The total area is then normalized to 100% and the area of each peak is calculated to give the weight percent of the sample.

---

* In order to provide significant improvement in peak resolution over the two options implemented, it would be necessary to analyze the line shapes in detail. This procedure is quite complicated (as the various line slopes are not related) and is best handled at this time by large scientific computers. Several programs for such analyses are available in the literature.

The output report is printed on a line printer and contains identification by job number and instrument number. Peaks are listed in order to emergence in the first column of the report. The second column lists the elution time in seconds while the third column lists the weight percent.

Additional Programs

When the previous application programs do not require all of the 1800's time, additional programs may also be run. These can consist of programs called by the real-time data acquisition programs to further process the data, or they may be programs which use data from other sources. Examples of such programs are: A spooling program which reads a deck of cards and prints a listing on the line printer, and a plotting program which plots a previously prepared logical tape on a digital plotter.

Another example involves a series of linked programs to process data from a Varian C-1024 time averaging device. A spectrum can be dumped from the C-1024 to punched cards as long as the integer numbers are to base 8. The first program reads the octal data from cards to disk in order to free the card reader as soon as possible. These data are then converted to decimal integers and stored on a logical tape. This program then calls a second

program and relinquishes its space in core. The accord program searches the logical tape for maximum and minimum points and shifts the base line to zero by subtracting the minimum from all points. This program then calls a smoothing program and relinquishes its space in core. The smoothing program carries out a nine-point digital smoothing calculation on the data in the logical tape and calls a plotting program before it relinquishes its space in core. The last program scales the smoothed data, "plots" the data on the line printer and then relinquishes core space. A minimum of core is required by this linking process because each program gives up its core as soon as it has called its succeeding program. Furthermore, the system is very flexible in that a program may call more than one program and have several programs executing simultaneously. For example a program doing analysis may initiate a data acquisition program. Thus, data may be analyzed and the spectrometer then activated to take additional data based on the results of the analysis of the original data.

Systems capability for handling background batch processing job streams are present. However, facilities for such processing have not as yet been implemented.

Text of a talk to be presented by C. H. Sederholm at the Pittsburgh Conference on Analytical Chemistry and Applied Spectroscopy, March 4-8, 1968

IBM

A COMPUTER SYSTEM FOR AUTOMATION OF THE ANALYTICAL LABORATORY

C. H. Sederholm
P. J. Friedl
T. R. Lusebrink

As was demonstrated at last year's conference, a large number of instrument manufacturers and a reasonably large number of instrument users have already taken the first steps toward automation. They have automated individual instruments with remarkably good results. A few of the more advanced laboratories have already started to automate entire laboratories. Surely widespread laboratory automation appears to be just around the corner.

In general, instrument automation has taken place by devoting and interfacing a small computer to an individual instrument. When one considers automation of an entire laboratory containing many such instruments, the question arises as to the advisability of acquiring many small computers, one attached to each instrument, or whether it would be preferable to have a single computer system which serves the entire laboratory.

There are three reasons for attacking the problem of laboratory automation in terms of successive instrument automation; i.e., using one small stand-alone computer per instrument. The most important consideration is one of isolation. Each individual user does not want to concern himself with any problems but his own. He does not want malfunctions of other instruments or their interfaces to jeopardize his interface to the computer system.

The programming considerations associated with his own instrument are sufficiently complex that he doesn't want to worry about other users' programming problems too. He is rightfully afraid of being forced to factor parts of his programming requirements into general purpose programs which serve the entire laboratory (programming by committee).

To a somewhat lesser extent, availability of the computer system encourages one to favor multiple computers, one per instrument. Waiting for a few seconds, or at most one or two minutes for computer facilities to become available would not be intolerable, but the prospects of having to schedule one's experiments and schedule the use of the computer facilities long in advance is very unpalatable to most users. In general, users prefer fewer facilities which are routinely available to larger facilities available by appointment only. Hence, the desire for availability tends to favor multiple instrument automation over laboratory automation.

Finally, the question of cost should be considered. A computer system which is capable of automating an entire laboratory will, of necessity, be more expensive, even in its most stripped-down version, than a computer system capable of automating a single laboratory instrument. Therefore, when one is taking the first step toward laboratory automation; that is, the automation of a single instrument, there is a very strong tendency to automate that instrument using one small computer. "Worry about laboratory automation after the first few instruments have been automated." However, if one considers automation of the entire laboratory as an ultimate goal

13

from the beginning, many of the considerations below imply that a single, shared, laboratory computer would give more performance per dollar.

The automation of a laboratory by the use of a single, shared computing facility has several outstanding advantages over multiple automated instruments. The purpose of automation is not only to acquire data but, also, to control the instrument during the data cquisition step, to process the data which has been acquired, to standardize it, to compare it against known parameters (e.g., to compare an unknown spectrum against a file of spectra of known compounds for identification), and finally to present the results in a form which is usable to the experimenter. The data acquisition and control steps can very often be adequately performed by a small computer. However, the data reduction steps, the comparison with data files, and the presentation of results in usable form, often require much larger compute capabilities. One solution to this is to record the raw data which has been acquired with a small stand-alone computer on a recording medium such as magnetic tape or punched paper tape. This data may then be processed on a large computer when time is available. However, if the raw data is at all voluminous, and magnetic tape must be used, the cost of the magnetic tape drive relative to the cost of the small computer can become very high. Hence, going this route, one is encouraged to minimize the quantity of data taken, often resulting in less precise results. Likewise, the turn-around time on very large computer facilities still is much longer than the individual investigator would like to wait between epochs of his experiment. That is, if he could have the data from the last epoch back

SVY

immediately, he could start planning the next experiment and executing it.

The larger shared laboratory computer then has three impelling reasons for its use in automation of laboratories. It can have sufficient core and sufficient processing capabilities to do reasonably large-scale data reduction and file look-up tasks at a modest expense per instrument attached to it. By having one central computer facility, one may take advantage of the fact that most analytical and spectrographic instruments have a low duty cycle; that is, much of the time is spent in sample preparation or with the instrument completely idle. Using a centralized computer facility, the computer need not be idle while an individual instrument is idle, since the computer can be used for other jobs. Finally, by using a larger centralized laboratory computer, the input/output devices may be shared by all the users of the system. Hence, for a given cost per user, each user may have the use of more sophisticated input/output and peripheral storage devices. E.g., it is possible to have a large disk file which can hold hundreds of thousands of words. This is most useful in the analysis of spectroscopic data. It is possible to have a line printer which increases the amount of printed output which a user can request and receive in a reasonable period of time. It is possible to have a card reader such that reasonably massive input data can be prepared in a reasonable format for input into the computer system. If one attached these input/output devices to each of the individual computers in a multiple instrument automation situation, the total expense for the input/output devices alone would be far greater than the cost of the laboratory automation facility.

The centralized laboratory automation facility seems to have several

important advantages. If the problems of unavailability and non-isolation of individual users could be successfully overcome using the centralized computer facility, it would seem to be a far more reasonable solution to automation of a laboratory. We believed that these problems could be overcome, and hence set out to construct a system designed to service multiple, isolated, asynchronous users, each of which had demands for closed-loop control, large-scale data reduction and file look-up programs, as well as smaller data acquisition and control programs. The specifications of this system were aimed at automation of a laboratory containing a group of analytical and/or spectrographic instruments. This system has been completed to a point, and has been demonstrated in the exhibition hall during this week.

The system features complete program independence and complete hardware independence of each instrument from all others. An application program for one instrument need in no way to take into consideration other programs running in the system simultaneously with it. An application program need not be modified as a result of a change in the total instrument mix attached to the computer system. Each instrument has its own individual data path directly to the core of the computer via a pair of data channels, so that there need be no sharing of the interfaces between various instruments.

This system uses an IBM 1800 computer. It is possible to operate this system on a computer with 16K words of core. However, it is only economically practical if the laboratory to be automated is sufficiently large to support a 24K or 32K word machine.

A new I/O device has been designed and built, specifically to support this laboratory automation system. This is a digital multiplexer channel for the 1800 which provides up to 32 discrete data paths between the laboratory and the core of the computer. This device allows data to be acquired from, or presented to, the individual instruments in a demand/response mode with a minimum of computer overhead. The reduction in computer overhead increases the allowable total data acquisition rate from all instruments by more than an order of magnitude. The demand/response mode of data acquisition is of great value in that it allows the instrument to say when data is available rather than the computer saying that data should be presented now. An example of the usefulness of demand/response data acquisition is in acquiring data from an infrared spectrometer which has automatic scan suppression. Since the scan rate is a function of the first derivative of the absorption, data acquired at equal intervals of time would not be at equal increments in wave length or wave number. However, with a demand/response interface, the instrument could be run with scan suppression, anddemands to take data could be made by the instrument at equal wave length or wave number increments.

The present operating system, known as the 1800 Laboratory System, requires approximately 13K of core which, with additional work, can be reduced. The rest of core (known as variable core) is broken up to 512-word blocks, called pages, which are assigned on a dynamic basis.

When a user wishes to sign on the system, he types a command at the 1800 console typewriter. This command causes the system to load a master program associated with that instrument into variable core and put that program into execution. Depending upon the size of a user's program, it may require one or multiple pages of variable core. If the program requires more than one page of core, the system loads the program into any available pages. The pages used need not be adjacent to each other.

This master program activates various buttons and indicator lights at the instrument interface. By pushing one of several buttons, the user may ask for one of several application programs associated with his instrument. When the master program recognizes which program or routine the user wishes to run, the master program asks the system to load the appropriate program into variable core and set it into execution. The master program then exists, releasing the core that it occupied back to the variable core pool. The specific application program may acquire parameters for the run by interrogating user-operated thumb switches at the instrument interface. Then the application program continues with its execution. At any time, an application program presently in execution may be aborted from the instrument interface. If the application program is very lengthy, the system provides facilities for chaining smaller segments together. The system also provides

facilities for disk-resident subroutines which only occupy core when they are called.

Many system subroutines are provided, and these are also loaded into pages of variable core when needed. If no user program presently in execution is using any of the subroutines on a given page, that page is returned to the pool of variable core. Because of this dynamic allocation of core, it is possible to use the system to execute a program requiring 15K of core, and a few minutes later, with no changes to the system, have the same computer executing 15 different programs simultaneously, each of which occupies 1K of core; i.e., a given instrument is not restricted to a fixed partition of core!

Presently, users' programs may be executed with two different priorities -- a foreground priority used for real-time data acquisition and control, and a middle-ground priority used for non-real-time calculations. All users' programs operating with foreground priority have the same priority. The system provides each program, in turn, with a 5 millisecond time slice for starting or stopping the data acquisition process and/or "massaging" current data and/or feedback control of instruments. Time slicing is also done for middle-ground programs, only the time slices here have 100 milliseconds. Only when there is no foreground program with processing to be done do the middle-ground programs receive time slices. If a foreground program finishes acquiring data and, hence, has some foreground processing to do (e.g., for instrument control) any active middle-ground time slice is interrupted to process the foreground program.

Once a data acquisition activity has been initiated in a foreground program, the new digital multiplexer channel provides for the activity at instrument rates. This activity procedes independently from and concurrently with the executions of time sliced programs.

Systems capability for handling background batch processing job streams are present. However, facilities for such processing have not been implemented yet.

Data stored on the disk file is referenced in terms of logical tapes. That is, each application program may be written as if there were available to it a large number of magnetic tape drives. Hence, an application program may ask the system to present it with 50 words of data off of logical tape SPEC starting with the 31,352nd word on the tape.

As far as the application programs are concerned, the entire system is oriented toward performing a variety of tasks, most of these tasks being associated with input/output. In general, it takes 12 words of a user's program to specify a task for the system. These tasks are accepted by the system and performed as soon as possible. Multiple tasks may be queued for a single input/output device; e.g., the line printer may have a current task in execution while 5 other tasks are waiting for the line printer to be free. A given application program may have several tasks outstanding simultaneously. For instance, a given application program may instruct the system to (1) acquire a block of data from a given instrument, (2) read a card in the card reader, (3) print a line of the line printer, (4) light a light at the

user interface, and (5) write a block of data on a logical tape. These tasks would be given to the system sequentially, but all five tasks would be set into execution before the first one was completed.

A partial list of tasks the system is capable of performing will demonstrate the power of the system:

1. Read a block of 50 words from logical tape DATA starting at word 5273.
2. Read the thumb switches at the NMR spectrometers.
3. Read 30,000 words from the gas chromotography columns at a data rate of 50 points per second and write these onto a logical tape called GCDAT.
4. Load and set into execution a smoothing program (SMUTH) in the foreground mode. (A common data area between the calling program and the called program is provided).
5. Release all core and system subroutines that this program presently occupies.
6. Read all the cards in the card reader and put them onto the logical tape FILE.
7. Print the logical tape OUT on the line printer.
8. Dump core presently occupied by program SMUTH.
9. Plot logical tape PLOT on the plotter.
10. When push button X is depressed, transfer control to entry point ENT in this program.

Having defined and constructed this system for use in the automation of an entire laboratory, we wished to investigate its actual usefulness. As

a result, we engaged in a joint study venture with Varian Associates of Palo Alto, California. Varian interfaced three of their analytical instruments to this system. These instruments included an M66 mass spectrometer, an A60 NMR, and a pair of aerograph gas chromatography columns. Multiple application programs for each of these instruments were jointly specified and written by Varian and IBM. Varian has had previous experience with the automation of individual instruments. Many of the programs previously written for stand-alone systems were modified and augmented to run on this system. For the A60 NMR, the following programs have been written: data acquisition and control, data smoothing, data catting, data presentation, and field homogeneity adjustment. The following programs have been written for the mass spectrometer: data acquisition and control, peak identification, and compound identification based upon the position of the five most intense peaks in a mass spectrum. For gas chromotography, a data acquisition and control program has been written. Peaks are detected, their areas are resolved and calculated, and retention time and areas are reported.

Once the system was defined and operational, the coding of the application programs went extremely rapidly. For the programs so far written, there has been no indication of unfavorable interaction between programs concurrently running on the system. Several of the programs which have been written, in particular for the mass spectrometer, could not have been written for a small stand-alone computer with no peripheral storage devices.

In conclusion, we believe that we have been able to produce a system for use in laboratory automation which provides each individual user the isolation, the availability, the real-time control responsiveness, and the price per instrument associated with multiple computers, one per instrument. In addition, this system provides sophisticated input/output devices, disk files, and a large amount of support for the I/O devices and the files, which would not be available on a small computer associated with a single instrument. Each user, then, has the impression that he has a large computer attached to his instrument and completely at his disposal.

#####

SESSION REPORT

COMMON - Chicago

Session Number TUE E1 Session Name LP/MOSS Tutorial

Chairman W. A. Pease, Jr.

Time 5:30 to 7:00 PM Attendance (No.) 29

Speakers Muller of IBM

Synopsis of Meeting Continuation of Tue C4, with reduced attendance. Mr. Muller asked that all inquired be sent to him. He also said that a specialized system was a better solution for problems with 60 or less equations. The way to get such a system is shown in the 1130 LP/MOSS manual on pages 11.2 and 11.3. Faster running and less work and overhead is the payoff here.

READING D. POLLITT
TUE EG

TOPIC: 1400 TO SYSTEM/360 MODEL 25 CONVERSION

I. Model 25

A System/360 Model 25 is upward compatible with other S/360 models within memory capacity and channel capabilities. The basic Model 25 is available as follows:

Core size --

16K, 24K, 32K, 48K

Native devices --

Up to 4, 2311

1 - 2540

1 - 1403-2 or 7

1 - 1052

Tape via channel --

2401/2803

2415

Channel --

Multiplexor or selector

Wide range of devices

II. Compatibility

1400 series compatibility is handled through use of a 16K reloadable control storage. Device relations are:

1401 --

1402 to 2540 native

1403 to 1403 native

1311 to 2311 native

tape to tape - selector channel 2401/2803 or 2415

1440 --

1442 to 1442 - channel

1443 to 1403 - native

1443 to 1443 - channel (144 print position)

1311 to 2311 - native

tape to tape - selector channel 2401/2803 or 2415

III. Channel Selection

An optional selector or multiplexor channel is available. The choice of channel is determined by device requirements. For example, tape requires the selector channel. If possible, unit record devices should use the multiplexor channel. Certain tape systems will require non-native unit record devices, in which case the selector channel must be specified. Non-tape configurations should specify the multiplexor channel option.

IV. Program Conversion

Native unit record devices provide greater throughput than do channel attached 1440 compatible devices. Consideration should be given to converting 1440 programs to 1401 mode wherever possible. The Type III library now has a conversion program to perform this conversion. While not 100%, it does provide an assist. Further field efforts are now underway. Significant throughput gains can be achieved just by the change in unit record devices now available on the Model 25 to the 1440 user.

V. Disk Data Conversion

File conversion is required to transfer data from 1311 files to 2311 files. To accomplish this conversion:

- Use 1401/1460 or 1440 utility program to dump 1311 files to card or tape on the 1400 system.
- Use the standard System/360 initialize disk program to initialize the 1316 pack.
- Use the 1401/1460 or 1440 clear disk utility in compatibility mode to format the 1316 pack for 1400 operations.
- Use the 1400 utility in compatibility mode to load card or tape to 2311.

SESSION REPORT

COMMON - Chicago

Session Number TUE E5 Session Name General Purpose

Chairman D. Dunsmore Plotting Program for the 1130

Time 5:30 to 7:00 PM Attendance (No.)

Speakers Peter J. Woodrow - ARAP

Synopsis of Meeting Mr. Woodrow presented a paper entitled, " A Small General Purpose Plotting System for the IBM 1130"

TUE ES

A SMALL GENERAL PURPOSE PLOTTING SYSTEM FOR THE IBM 1130

by

Peter J. Woodrow, Associate Consultant
Aeronautical Research Associates of Princeton, Inc.
Princeton, New Jersey

Presented at the Chicago COMMON Meeting, April 8-10, 1968

ABSTRACT

This paper describes the overall characteristics and capabilities of a small general purpose plotting program for use with the IBM 1130 and IBM 1627 or other incremental plotter. While specific details are contained in a reference manual, this paper describes the general features of the program. Also briefly described are general purpose subroutines that were developed for use with the program but can easily be used for other applications.

As is inevitable with such a program, a number of other programs and subroutines have also been developed, primarily for purposes of tabular input to the plotting program (and other programs) and providing facilities for very large amounts of data to be generated and plotted without exceeding disk capacity. These additional programs and subroutines are discussed in later sections of the paper.

Throughout the paper comments are made as to future modifications and additions to the package that are planned or desirable. Some of these modifications will require alteration of IBM 1130 systems programs or subroutines.

I. INTRODUCTION

Aeronautical Research Associates of Princeton, Inc., is a small independent research and consulting company engaged primarily in government sponsored research in various areas of aeronautical sciences. To provide computational facilities, ARAP has had some form of computer system since 1958. In June 1966, an IBM 1130 computer system (with a 1442 card read/punch, and 1132 line printer, 8K of core and disk) was installed to handle increased computational requirements. After the system had been operational for almost a year, company management (i.e., the president) was finally convinced that an on-line plotter would be a worthwhile investment even though it represented a capital expense (small companies rarely have much free capital). Accordingly, in September 1967, a Benson-Lehner Model 105 digital incremental plotter was added to the system.

One of the arguments that was used to convince management was the availability of very extensive, IBM supplied, software support for such a plotter in the form of the Data Presentation System (DPS). Unfortunately it quickly became clear that DPS had a number of drawbacks, in particular:

1) DPS disk file structure is incompatible with FORTRAN disk I/O. Thus, a program generating data to be plotted via DPS would either have to punch cards (to later be read via DPS) or use the DPS disk I/O routines. The latter solution would have required extensive modifications to existing programs, would result in reduced core availability, and, worst of all, would prevent the use of LOCAL/SOCAL's (an essential facility for most programs we run on the IBM 1130).

2) Even a minimal version of DPS requires at least 400 sectors on a disk cartridge. As we are unable to spare this much space on our master disk cartridge, extensive cartridge changing would have been required. The time wasted in changing cartridges would be prohibitive.

3) Use of DPS is not easy. Many of the users are engineers who, while competent, are unwilling to spend a number of hours learning a completely new language just to produce fairly straightforward plots.

4) While considerable effort has been made in DPS to reduce wasted (i.e., non-productive) computer time, the amount of time DPS requires for a typical plot is excessively large.

5) The current version of DPS is completely incompatible with the IBM 1130 Disk Monitor, Version 2. Since the new version offers a number of desirable features, we plan to start operating with the new version as our standard version as soon as Modification Level 1 has been installed (as delivered the new version has a large number of "bugs", mostly minor).

It was therefore clear that DPS would not provide the necessary software support for our new plotter. In order to start plotting, a few users decided to use the IBM supplied plotting routines designed for use with FORTRAN. These users soon discovered a new set of problems. When they tried to make plotting a part of their main program, they found themselves short on core space. In addition, the FORTRAN routines will not operate properly when SOCAL's are required. Thus, most of these initial users were forced to either abandon their efforts or output their plottable data on the disk and then write another main program to do plotting exclusively.

This latter approach caused a considerable amount of duplicated effort. In addition, the FORTRAN plotting routines require an extensive amount of programming in order to produce an acceptable graph complete with labels and titles. In fact, most of those attempting to use the FORTRAN plotting routines required at least three (and often as many as ten) recompilations before their plot was acceptable.

Thus it was decided to develop a reasonable general plotting program designed to operate within a disk file containing data to be plotted. A number of users were queried as to their desires

insofar as program capabilities were concerned. From this discussion evolved a basic set of requirements. The program described below was developed around these requirements, some of which were specifically oriented towards two very large and important programs that have between them accounted for more than 50% of the IBM 1130 usage. During development, additional features were added whenever it was obvious that such features were desirable and easy to accomodate within the basic design framework. Needless to say, a number of additional modifications are currently contemplated as a result of extensive experience with the program.

This small, but rather general, plotting program has proved exceptionally useful over the past three months it has been in use, despite a rather rigorous format and certain restrictions. As stated above, the program was developed primarily to satisfy the somewhat specific requirements of two large programs. Nevertheless, essentially every user has by now made use of the program at least once and most of them now make a habit of incorporating a plottable output file (i.e., a file in ASPP format) in all new programs.

Because of the availability of the ASPP plotting program, a number of supporting programs have been developed. One set of such programs is designed to input (from cards) one- or two-dimensional tables and to output a file which is plottable, as well usable as a table file in some other program. If these programs are used consistently to create all tabular data files, then the result is that, as a byproduct, all input tables to any program can be plotted if and when the user desires.

A second set of programs and subroutines arose from the desire to maintain a single disk cartridge operation. The two large programs mentioned above are both capable of producing very large amounts of plottable output during runs which may exceed 50 hours. In particular, one of these programs produced data at 1800 points (25 variables per point). A disk file of at least 140 sectors would have been needed to contain this data and, since the run was broken

into short segments with a total period of two weeks, this large file would have had to reside on the disk for at least two weeks, an intolerable situation with regard to our master disk cartridge. In addition, since the run took a total of 50 hours of computer time to produce the data, it was highly desirable to retain the plottable data on punched cards in case replots were needed at a later date.

To satisfy these requirements (as well as others) a set of programs and subroutines was developed to read and punch binary cards, each card containing a checksum, indicators, up to twenty-five two-word floating point (or integer) variables and card identification. These programs permitted the punching of a plot output file whenever it filled up, combining the decks from each segment, and reading selected variables onto the disk for plotting. In the problem described above, 1800 cards were sufficient to contain all the data and at no time were more than 30 sectors permanently allocated on the disk (if it had been necessary, even fewer sectors would have been required).

It is reasonable to expect that in the future, other programs will be written to support the basic plotting package. We believe that this open-ended approach will always require much less disk space than that needed by DPS. Already the system provides a number of features not available under DPS. As a final remark, it should be noted that the complete set of subroutines for the basic plotting program requires less than 33 sectors of disk space. Even with all of the support programs described above, it is unlikely that the requirements exceed 60 sectors.

The following sections describe in somewhat more detail (although insufficient detail for actual use) each of the programs and general purpose subroutines incorporated in the overall system as currently implemented. In addition, comments are frequently made in the discussion concerning desirable and/or planned modifications to the existing modules.

II. ASPP - ARAP SPECIAL PLOTTING PROGRAM

This section describes in some detail the features of the ARAP plotting program. An interested user should read the detailed reference manual for full instructions on use of this program. In order to enable the potential user to determine whether this program is at all suitable, the restrictions on the use of the program are stated below:

1) ASPP uses a special version of PAPTZ to provide a FORTRAN reread facility (see III.A.); therefore, any FORTRAN program stored on the same disk as ASPP must not use FORTRAN paper-tape I/O. This restriction will be removed when we are able to modify SFIO to incorporate the necessary reread facility within SFIO itself (using unit 0).

2) As currently implemented, ASPP can provide only linear (as opposed to log or log-log or monthly, etc.) plots. This fact has not yet caused problems, but there are plans to add log scales in the near future (the primary limitation now is core space). Also the program is incapable of providing multiple scales, reference grids, or any other "special effects". Some of these can now be accomplished via an appropriate data file and some may be added to the program, but most will not. The program is designed to be used by the average engineer without extensive training.

3) As is the case with DPS, data files to be plotted via ASPP may contain two-word floating point (real) variables only (i.e., extended precision cannot be used). If necessary, it is quite easy to write a combination FORTRAN/ASM program to convert a file containing extended precision numbers to standard precision; we have not yet needed such a program. In addition, all plot data values must be of type real (certain integers are, however, used as indicators). Programs may use one-word integers if some care is taken in creating the plottable data file.

These are the most serious limitations of ASPP; others, less significant, will become obvious below.

A. Plottable Data File Structure

Since all comunication (of data to be plotted) between the user's program and ASPP is accomplished via the data file on the disk, it is important to note the structure of this file. The file must be defined in the user's program as a two-word-per-record file with a maximum of 32,000 records in the file (200 sectors). The first record of the file must contain the number of variables (NV) per "block" in the file. This number must be between 1 and 25 inclusive and is stored as an integer in the first word of the record. The next NV records in the file must contain the "names" of each of the variables in a "block". These "names" are used for communication purposes only so that the user need not specify to ASPP the position of a desired variable within a block; he merely specifies the "name". Each "name" consists of 1-4 characters as read into a real variable using A4 format (or the DATA statement in Version 2 can be used).

Following the file header are a series of data blocks, each consisting of NV records. The order of the variables in each data block must be the same as the order of the "names" in the header. If the first word of the first record in a data block is the integer 1 or the integer 2 (easily distinguishable from a floating point number), it indicates that the record is a control record. The integer 1 signifies the end of data on the file and must be present somewhere in the file. The integer 2 is used to indicate a curve break; i.e., during plotting, the pen is lifted when this record is sensed and set down again at the next specified point in the file. If the record contains the integer 2, the record is skipped and the next data block is assumed to start with the next record after the curve-break-indicator record. Note again that integers should not appear within data blocks and that a data block starting with any other integer than 1 or 2 is assumed to be valid data; very clearly undesirable plots will result in either case.

The file structure described above is assumed by all programs within the system. All programs check the validity of NV and indicate an error if it is not between 1 and 25 inclusive. All programs also expect an end-of-data-indicator record. Because data files may be of variable length (each program assumes the maximum of 32,000 words and allows the loader to truncate the size for smaller files), SDFIO will indicate a F101 error if the end-of-data-indicator record cannot be found in the proper place (as the first record of a data block). All plotting systems programs are written in such a way that this record may be the last record in the file.

B. Plot Specification

Plot specification is accomplished by a somewhat arbitrary mixture of scale specification cards and plot specification cards (i.e., the program reads each card in A4 format, determines whether the card is a scale or plot specification card, and then "rereads" the card using a format appropriate to the card; however, each scale card for any variable specified on a plot card must have appeared before the plot card).

A scale specification card specifies the "name" of the variable to be scaled (the "name" must agree with one of those in the data file header), the scale textual label, special scale indication (whether the scale should be drawn at its normal position or zero of the opposite scale, and whether a reference line should be drawn at zero of this scale (with or without tickmarks every inch) parallel to the opposite scale), scale delta (= no. user units/inch) indication (whether the scale delta is specified on this card or is to be computed by ASPP from the data or is that specified in a previous card for some other variable), scale minimum indication (same options as for scale delta), length of this scale in inches, scale delta or "name" of variable providing scale delta (unless automatic scale delta specified), scale minimum data value or "name" of variable providing scale minimum (unless automatic scale

minimum specified), and linear transformation parameters (any variable may be transformed from x to x' = ax + b where a,b are specified on this card).

The plot specification card specifies the "names" of the X(abscissa) variable and Y(ordinate) variable, type of plot indication (point, linear, or "quadratic" smoothing between points), overlay plot indication (whether this plot is an overlay and whether next plot is to be an overlay), tagging specification (tag character and no. of data points between tags), and plot textual title (one or two lines of text).

The input program makes very extensive checks on the validity of the input specification data, printing an explicit error message as well as the card image for each error encountered. The program attempts to find as many errors as possible on each card. If a specification card is in error, the program ignores it and goes on to the next. At the end of specification input, the user has the option of making those plots which were specification-error-free or of reinputting all specification cards (after corrections have been made). The input program also provides the user (if he so desires) with a list of the maximum and minimum values of each variable in his data file and prints a warning message for any scale card specifying scaling such that data points will be off-scale; however, such an occurrence is not considered an error.

C. ASPP Capabilities and Plot Features

In order to fit the program into core,ASPP is divided into two links, ASPPI, which reads the specifications cards and outputs a specifications file and ASPPP, which does all of the actual plotting. While the loading time of ASPPI is annoying, the loading time of ASPPP provides the user with just about enough time to ready the plotter and insert the pen. In any case, the two loads are only required once for a whole series of plots (a typical 8 sector specification file can accomodate specifications for 25 scales and 24 plots) This mode of operation

also allows the user to just reexecute ASPPP if he desires multiple copies of plots or if there is a mechanical plotter failure during plotting.

As shculd be obvious from the description of specifications cards above, plots produced by ASPP have a rather rigorous format which is described below. Nevertheless, a number of features of a plot can be altered (see D.); nominal values for all variable elements are given below.

1) The X(abscissa) scale is drawn from the initial pen position parallel to the edge of the paper. Scale annotation and label are drawn to the right of the scale. Tick marks are provided every inch. One-half inch is needed for annotation and label.

2) The Y(ordinate) scale is drawn from the initial pen position perpendicular to the edge of the paper. Scale annotation and label are drawn to the left of the scale (i.e., outside the plotting area) with the annotation parallel to the X scale and the label parallel to the Y scale. Annotation and label take up 1-1/4" to the left of the scale. Tick marks are provided every inch.

3) The plot title is drawn such that the bottom of the last line of text is parallel to the X scale and 8.5" from the initial pen position.

4) At the end of a plot (including any overlay curves) the pen is positioned 4" beyond the end of the X scale and at the same Y position as the initial pen position.

5) The size of the scale annotation characters is .12", the scale label characters .18", and the title characters .24". The routines used provide eye-pleasing variable character widths. Each text field (the longest line of the title) is centered automatically on the appropriate scale.

6) Tags are plotted so as not to interfere with the curve (except that a special cross-hairs tag may be specified through which the curve passes).

7) When using "quadratic" interpolation between data points, the program actually plots a series of straight line segments approximately .10" in length.

8) The program automatically inhibits any plotting outside the plot region (the rectangle "formed" by the X,Y scales). The pen is lifted before the curve goes off-scale and is set down again when the curve comes back on-scale.

9) Points closer than .01" are not plotted.

10) Switches are used to specify "immediate" termination of current plot and setup for next plot, suppression of plot title, and suppression of scale information plotting.

11) ASPPP uses a modified version of the DPS UP,DOWN and DRAW for plotting. Versions can be supplied to work with either a .01" step or .005" step plotter. If the latter version is used, line start and end can only be specified to the nearest 1/100", but line segment plotting is done to an accuracy of .005" (i.e., the routine takes advantage of the finer step size to provide a more pleasing line).

12) If autoscale delta is specified, ASPPI selects a delta which is 1, 2, or 5 times an appropriate power of ten.

13) If autoscale minimum is specified, ASPPI selects a minimum which is an integral multiple of the scale delta and which is such as to cause approximate centering of the curve on the plot region.

D. ASPP Basic Parameter Modification

ASPP was designed to require the user to specify only those quantities he would be most likely to want to vary from plot to plot. However, wherever possible, other "fixed" parameters were treated as variables input from a basic parameter file which is set up by another program in the system. Thus, a number of the quantities given above can actually be changed if the user desires.

The parameters that can be easily altered to suit individual requirements are: origin, rotation, subdivision (tick mark) interval, and text rotation for X or Y scale; shortest distance allowed between points (points closer than this are ignored); size of space allowed for tag; X or Y displacement from appropriate scale for X or Y scale label; rotation of X or Y scale label; rotation of plot title; scale label X- or Y- font size; plot title X- or Y- font size; tag character X- or Y- font size; X or Y scale maximum length (for checking purposes); Y displacement of plot title from X scale; Y displacement per line of plot title text; maximum number of lines allowed in plot title; character used to denote special cross-hairs tag; up to twenty autoscale values (currently 1, 2, 5, 19); any character used to denote special action (e.g., P in column 1 currently denotes plot specification card and S scale specification card; these could be changed to U and V if so desired).

One parameter that is fixed by the DPS routine used to draw scale annotation is the size of the annotation characters; both X- and Y- font size must be .12". In addition, annotation is provided at intervals of no less than 1" regardless of what the subdivision interval happens to be.

E. Possible ASPP Program Modifications

Some of the possible modifications listed below are definitely planned; others are desirable but would involve extensive programming and are thus not currently anticipated. The order of the modifications listed below is of no consequence.

1) Replace the scale annotation routine (ANOTS from DPS) with one that provides a fixed number of digits after the decimal point for each annotation value.

2) Add the ability to specify a scale as logarithmic.

3) Provide for text specifications cards (optional) which could place specified text of specified X- and Y- font size and specified rotation anywhere within the plot region. Two types would be pro-

vided, one of which would cause text to be placed on every plot until a new specification is encountered, the other which would only apply to the most recent plot card processed.

4) Provide for user program-generated, plottable, text input, probably in the form of an additional (optimal) input file to ASPP. A specification card might be added to select a specific text field for plotting on the most recent plot.

5) Add two new types of plot, one for linear Y interpolation, quadratic X interpolation and the other for linear X interpolation, quadratic Y interpolation. The present system of quadratic interpolation provides some rather peculiar curves for sparse data, especially when one of the variables is known to be monotonic (such as time).

6) Provide automatic tag update whenever a curve break is encountered in the data file.

It is inevitable that any user of the system would suggest a number of desirable additions. However, it is important to keep the original design objective in mind, that of providing an easy to use plotting system requiring small amounts of disk space and designed primarily for plotting program-generated data for engineering uses. Again, we do believe that the current program satisfies these objectives very well.

III. GENERAL PURPOSE SUBROUTINES IN ASPP

Where feasible, routines developed for ASPP were designed to be of general use. Those that may be useful are described briefly below.

A. FORTRAN Reread Facility.

Once the basic specifications for ASPP had been settled, we found it necessary to provide the capability of rereading the same card with a different format. Since we do not have paper tape, this feature was added partly by replacing PAPTZ with our own special version. Whenever SIOBF is called in a FORTRAN program, it saves in a special area the current contents of the FORTRAN I/O buffer. Whenever unit 4 (paper tape) is used in a READ statement, the special PAPTZ transfers the current contents of the special buffer to the FORTRAN I/O buffer. SFIO proceeds as though a new "card" had just been read. A WRITE to unit 4 causes no I/O action and thus is an easy means of suppressing debug output. It can also be used to write information to the buffer in one format and read it with a different format with no actual I/O taking place.

B. BUFDR, BUFDW, BUFCL - Buffered Disk I/O

As any IBM 1130 user quickly discovers, FORTRAN disk I/O is notoriously inefficient. Thus, this multiple entry point routine was added. BUFDR and BUFDW work with a 161-word (real) buffer, up to one for each file being referenced. BUFDR "reads" a number of words (2-word variables; will not work with one-word-integers) from the disk, executing a disk read only if the data required is not in the specified buffer. BUFDW "writes" a number of words, executing a disk write only if the record to be written is not contained in the specified buffer. In addition, BUFDW sets a flag indicating buffer contents modified so that the buffer will be written out if a new sector is read from the disk. BUFCL is used to force output if the specified buffer has been modified.

These routines will probably become obsolete once we have replaced FORTRAN disk I/O (SDFIO) with a slightly modified version which will execute a disk read only if the necessary information is not contained in the sector in its buffer and will provide overlap with computation when writing on the disk.

C. Modified Version of DPS UP,DOWN,DRAW

We have modified this DPS routine by adding a new entry point, SPLTB, which is used to set up a specified length buffer for the routine. Thereafter,calls to any of the entries, UP, DOWN or DRAW, cause the command to be inserted in the buffer if the routine is busy. Thus, only if the buffer is full, will the routine wait until the previous operation is complete. We have found that this modification can significantly increase plotting speed for certain applications.

In addition, we have discovered a simple way to modify the routine so as to support a .005" step plotter such as we have.

D. CHTST - Character Test Routine

Since we decided to use a number of DPS subroutines with ASPP, and since ASPP uses variable width characters, we were forced to write a routine to compute the length of a text field. Thus, this routine scans a vector of text (as read in using A4 format), determines if any invalid characters are present, determines if an end-of-text delimiter is present, determines the number of lines in the text vector and the length of the longest line in DPS "X-units".

Some of these subroutines have found extensive use in other segments of our plotting system; some have only been used in ASPP at this point.

IV. TABULAR INPUT/OUTPUT FOR USE WITH ASPP

This section describes a number of programs and a subroutine designed to work with ASPP structured data files. The purpose of the program is to input tabular data for use with a user's program, but in such a format that the file is immediately plottable via ASPP. In addition, the header on the file provides a user with the means for verifying in his program that the correct file has been specified on the FILES card.

A. PLTBI - Plottable Tabular Input

This program provides the facility to read in data cards containing up to 14 dependent variables and 1 independent variable. The program makes up an ASPP-compatible file which may also be accessed by a user's program. If the first input card (specifying total number of variables and variable "names") is blank, the current contents of the data file are printed with appropriate page headings containing the "names" of the variables.

B. TDTBI - Two-Dimensional Tabular Input

This program provides the facility to read in data cards containing up to 8 dependent variables defined at intersections of 2 independent variables. The file created by the program is not directly plottable, but is useable with PLTDT and SRDTB (below). If the first input card is blank (specifying number of dependent variables, number of points of second independent variable, and "names" of all variables), the current contents of the file are printed with appropriate page headings.

C. PLTDT - Plottable Two-Dimensional Tabular Data

This program reads an input file produced by TDTBI and produces an output file that is plottable via ASPP. The plottable file contains a number of segments separated by a curve-break-indicator

record, one segment for each value of the first independent variable. Each segment specifies a number of curves versus the second independent variable, one for each dependent variable. Thus, ASPP will automatically produce for each plot a family of curves, one curve for each value of the first independent variable in the original file. It is up to the user to label the curves on the plot with the appropriate values of the first independent variable.

D. SRDTB - Search Disk Tabular Data File

This subroutine is designed to locate a specific data block within an ASPP-compatible data file (or a file produced by TDTBI). The first variable of each data block is assumed to be the independent variable. The subroutine will search forwards or backwards from the current position as necessary and will call a specified error subroutine with a specified integer error number as the parameter if it cannot bracket the input value with two data blocks contained in the data file (i.e., between the header and the end-of-data indicator record). A data file used with SRDTB cannot contain curve-break-indicator records. Output of the subroutine is the two data blocks bracketing the input value and a quantity that can be used in a linear interpolation scheme.

Eventually a routine will be written that can be part of the disk SOCAL overlay which will do the actual searching (at high speed since no overlay will take place during search; floating-point comparison will be built into this routine). SRDTB will then call this new routine and will compute the quantity to use in linear interpolation. It will, in addition, decide if search is necessary (to prevent unnecessary SOCAL overlay if input value is bracketed by current data blocks).

V. BINARY CARD I/O PROGRAMS AND OTHER SPECIAL PROGRAMS FOR USE WITH ASPP

As mentioned in the introduction, certain programs produce extensive quantities of plottable data while a run may require an extended period of time. Where the necessary data file has been deemed too large to reside on our master disk cartridge, these programs were modified to fill up a much smaller file, print a message to that effect, reset the record pointer to pointer at the first data block, and exit to the resident monitor after saving enough information in some other data file for later restart purposes. The contents of the plottable file are then punched using PFTBC, and the program restarted. When the run is complete, the binary card decks produced by PFTBC are combined into one deck which is fed into BCTPF. BCTPF puts selected variables from the master deck on the disk in a format suitable for plotting. Since the plots are made immediately, the data file can be considerably larger (the file is only temporary; it is destroyed as soon as plots are complete). This procedure has proved highly satisfactory for those users which require it, allowing them to run on the master cartridge intermixed with other user jobs. PFTBC and BCTPF, among other programs, are described briefly below.

A. PFTBC - Plot File to Binary Cards

This program uses FIMGR to read in the first data card containing from 0 to 8 characters of gang-punched deck identification. It then proceeds to punch the contents of an ASPP compatible file on cards using SBINP. Up to twenty-five variables are punched on each card along with a checksum, a binary card index, an indicator as to number of variables on card, an EBCDIC card index, and a deck identification character string. More than one data block may be punched on each card with the one proviso that a data block may not be split across cards. Card punching continues at maximum 1442

Card Read/Punch speed until the end-of-data indicator record is punched. The program then exits to the monitor.

B. BCTPF - Binary Card to Plot File

This program reads in three data cards using FIMGR. The first card specifies those variables which are to be checked for monotonicity from data block to data block, the second, those variables which are to be listed on the 1132 line printer, and the third, those variables which are to be stored on an ASPP-compatible and plottable file. The program then uses FBINR to read in binary card decks as punched by PFTBC. The program will accept multiple decks provided that the headers on every deck are identical. The first blank header card terminates input and, if the very first header card is blank, BCTPF uses the output file as input (i.e., it will list and/or contract an ASPP-compatible file). The program checks each card for proper card index, but will permit cards to be left out if user so specifies via data switches. This feature is useful for removing "bad points" before plotting.

C. EDTPF - Edit Plot File

This program edits (alters) specified records of an ASPP-compatible disk file. The record can be located either by record index (this is printed by BCTPF if file is listed) or by a specified variable being "closest" to a specified value (i.e., content-addressable file in some sense). Once a record is located, any or all of the variables in the block can be changed to new values. If any variables are changed, the contents of the data block before and after editing are listed on the line printer. This program provides even more facility than BCTPF for editing data prior to plotting.

D. AVGPF - Average Plot File

This program computes the "time" average of specified variables in an ASPP-compatible file. The first specified variable becomes the independent ("time") variable and the remainder of the variables are averaged over specified intervals in the independent variable by integrating (using trapezoidal integration) each dependent variable over the interval and then dividing by the interval size. The interval may either be specified (the actual interval, however, extends from one data point to another) or may be computed by the program as the interval between two minimum points of the first dependent variable. Output is in the form of line printer output, punched card output (optional), and an ASPP-compatible (plottable) file. This program is very useful for smoothing highly oscillatory program-generated data.

E. Binary Card I/O Subroutines

All of the binary card I/O subroutines are callable from a FORTRAN program which does not use FORTRAN card I/O (or FORTRAN Typewriter/Keyboard I/O in Version 1 of Disk Monitor System). The basic card format is the same for all; the first two words (16-bit words) on the card comprise a check sum, the next two 16-bit integers communicated from or to the user's program; the next 50 contain up to 25 two-word variables (real or two-word-integer). Columns 73-80 are available for card identification. Each of the routines has a parameter specifying whether overlapped I/O is desired. The salient features of the various routines are:

1) XBINP - LBINP - XBINP serves merely to LIBF to LBINP so that LBINP may be part of the I/O subroutine SOCAL overlay (overlap must not be specified if this is the case). LBINP is an entirely self-contained binary card punch routine. It thus cannot be used within the same program as any of the routines below. It also does not check cards to be punched for blanks in all columns. However, it is by far the shortest (including card I/O requirements) of any of the binary card punch routines.

2) FBINP - This subroutine is identical to XBINP - LBINP except that it uses CARD1 to read/punch each card. Each card is checked for blanks prior to being punched; if nonblank, an error wait is executed followed by a retry on the read.

3) SBINP - This subroutine is similar to FBINP except that it punches cols. 73-80 of each card with user provided deck information and card index.

4) FBINR - This routine is the binary card read counterpart of FBINP. It has one additional parameter which indicates the result of the comparison of the computed checksum with the checksum on the card.

F. SEQBC - Sequence Binary Cards.

This program essentially performs the functions of SBINP on a deck punched by LBINP or FBINP - namely, it adds deck information and card index to each card in cols. 73-80.

G. FIMGR - FORTRAN Card Image Read

This subroutine was designed for use with programs that use FBINR or SBINP. It reads a specified number of columns of a card and transfers them to a two-word-integer vector with no translation. Thus, the result is a set of integer words which contain card column images. The program provides deck identification input for programs using SBINP and switch input for programs using FBINP, SBINP or FBINR (e.g., BCTPF).

SESSION REPORT

COMMON - Chicago

Session Number TUE F1 Session Name 360 DOS

Chairman A. Ragsdale

Time 8:00 to 10:00 PM Attendance (No.) 63

Speakers A. Ragsdale - General Foods Ltd.

Mr. W. Sole - Polymer Corp.,Ltd.

Mr. B. White - IBM

Synopsis of Meeting This session consisted of a PAnel discussion and tutorial on multiprogramming facilities as they are available under DOS. The discussion ranged from SYSGEN requirements to implementation and operation. Application of Spooling and Multiprogramming were discussed at great length.

SESSION REPORT

COMMON - Chicago

Session Number WED A1 Session Name 360 Commercial Users

Chairman Fred A. Hatfield

Time 8:30 to 10:00 AM Attendance (No.) 20

Speakers Mr. Hatfield plus floor discussion

Synopsis of Meeting The meeting consisted of two parts. The first was supposed to be a continuation of the full 360 project to consider two resolutions which were presented by Wade Norton. Resolution # 1 was a request to IBM to make the SORT/MERGE control cards of OS And DOS compatable with each other or to specifically publish that they were not meant to be. Resolution Passed. Resolution # 2 was a request to IBM that format 6 labels of OS be accepted by DOS even though ignored. This resolution was referred to the DOS committee since partitioned data sets are not even known to DOS.

Part 2 of the meeting was discussion of COS for the most part. Fred Hatfield told of his company's work in expanding the 1405 being emulated to addresses greater than 99999, currently going to 124999 on five 2311 packs. Jean Louis (1724) told of work done to run multiple 1401 jobs concurrently. Al Cunningham (1628) told of changes to permit printing of HALTA registers on one line. Fred Hatfield also reported on a job recording scheme taken from EL-TAS, Lib # 360 D 00.04.009.

SESSION REPORT

COMMON - Chicago

Session Number WED A2 Session Name Introduction to IBM's

Chairman Walter Arntson Program Information Department

Time 8:30 Attendance (No.) 60

Speakers Mr. Dan Leeson, IBM

Synopsis of Meeting The functions of the IBM Program Information Department were described. The resources available to IBM customers were explained and certain significant points relative to the utilization of Type I and Type II programs. An hours discussion followed the half hour talk. Users complained of missing cards in program decks, time required to secure programs, why a complete deck could not be sent upon a revision of a program rather than the necessary cards, etc.

Good Morning Ladies and Gentlemen:

It is a great pleasure for me to have the opportunity to speak once again at COMMON.

The subject of my talk this morning will be on the Program Information Department which is located in Hawthorne, New York.

(SLIDE #1)

Most specifically, I would like to talk to the new users among you who have not had an opportunity to draw upon the services of this facility at any great length. Perhaps some of this material will be old hat to many of you, but for the purposes of establishing good rapport with the newcomers, I thought that this kind of talk would be satisfactory. At the conclusion of this discussion, please feel free to bring up any questions or comments which you may have.

For the newer members of COMMON, the questions relative to the Program Information Department that are probably going through your minds are:

1. What is it?
2. How do I discern what its contents are and what is applicable to me?
3. How do I obtain programs?
4. What do I do with them when I obtain them?

Basically, my talk will center around these points.

The first point I would like to address is what is PID?

The Program Information Department of the IBM Corporation is the official center of program distribution activity for the IBM Domestic Corporation. There are equivalents of PID for our World Trade customers. For instance, there is the European Program Library in Paris, the Asian Program Library in Tokyo, the Canadian Program Library in Toronto, the South American Program Library in Rio de Janeiro and the South Pacific Program Library in Sydney.

Programs received from the IBM development centers throughout the world enter into the distribution complex of the Corporation at the Program Information Department. It is there that the material is determined to be complete and reproducible, and it is also there to which orders for programs should be sent.

The material that PID distributes essentially falls into two categories. They are Type I/II programs and Type III/IV programs. Type I/II programs are developed by the IBM Corporation and have a range of support associated with them about which I am going to speak at some length toward the latter part of my talk.

Type III/IV programs are contributions made to this distribution center by both IBM employees and customers of IBM. In the case of Type III/IV programs we act as a distribution agent for these items and have no cognizance relative to the functionability of these programs. There are several types of standards that Type III/IV programs adhere to. As a matter of fact, COMMON was a party to the development of Type IV Standards, and the Type III Standards bear considerable similarity to those of the user groups. PID does act as agent for enforcement of these Standards, however.

Thus, I hope my first question of what is PID has been appropriately answered. In its simplest terms, PID is a building containing programs which you may order.

It is very important for you to know what the specific resources of PID are, which ones apply to you, and how this intelligence can be used to serve you.

Published several times a year are documents called the Catalog of Programs.

(SLIDE #2)

These Catalogs are published by system type; that is, there is a Catalog of Programs for the S/360, a Catalog of Programs for the 1130, 1620, etc. Within each Catalog is a discussion of the programs applicable to that system type. In general, you require the Catalog for only the system type of concern to you.

One of the most important items in the Catalog is the KWIC Index.

(SLIDE #3)

Here you will find whether or not a program is available based upon a keyword.

For instance, if you were interested in a program having to do with electrical power, note that either "power" or "electrical power" would be used as a keyword.

(SLIDE #4)

The entry in the Catalog of Programs reads "Load Flow Performance Calculations of Electrical Power." This particular program is for an 1130, whose configuration we will determine in a moment, and is described on page 22.

Turning to page 22

(SLIDE #5)

we find that the full title is

(SLIDE #6)

"Load Flow Performance Calculations on an Electrical Power System". This program number is 16.4.004, and the author and the availability date are also given within the abstract of the program. This particular program, as you can see, is a Type III program, and the configuration of the specific 1130 is given within the abstract. Having this information provides you with the intelligence to decide relative to its desirability.

Thus, through the array of Catalogs, you are able to discern the contents of PID. Through the specific Catalog in question, the Keyword-in-Context Index enables you to find a specific program by functional capability and the abstract of the program with associated information may then be located.

Catalogs of Programs for all systems are available from your local IBM representative. Once you are registered for a specific Catalog, maintenance to it will be shipped to you periodically enabling you to know how the contents of PID are altered as per your requirements. Also from time to time the Catalog will be completely reissued.

Once you are sure as to the nature, number, name, etc. of the program which you wish to order, you will find that you can place an order on several types of order forms.

(SLIDE #7)

The first of these is generally used for Type I/II programs only. The second, or tan colored card, should be employed for Type III/IV programs. The last one which is a rather large order form should be employed only when ordering S/360 operating systems. By this I do not mean OS/360 but rather any operating system with a collection of orderable items which is runnable on the compatible S/360 family. Specifically that includes OS, DOS, TOS and BOS.

There are also certain cards especially designed for user group members, and from time to time, you will have an opportunity to employ these cards.

(SLIDE #8)

The result of using an incorrect order form could conceivably be the rejection of the order so it is appropriate that you know which form for which program type. Upon completion of the order form for Type I/II programs, the material should be given to your local IBM representative who will authorize shipment of material to you and forward the order forms to us.

Please be cognizant of the nature of the distribution media for the programs that you wish because in some cases, it might be necessary for you to forward to us a reel of magnetic tape, a 2315 disk cartridge or a 1316 disk pack.

(SLIDE #9)

If the Catalog of Programs indicates that the program is available in cards or paper tape or Distribution Tape Reel (DTR), these will be forwarded to you with the appropriate data contained on them.

(SLIDE #10)

You are not obliged to take any action relative to these distribution media.

Once the order has arrived in PID, our policy is to have the material on the way to you within ten working days. Every attempt is made to guarantee the integrity of the data that is being shipped to you. In the event that you are concerned about the status of an order, you will find that the most rapid response can be obtained by communicating directly with your IBM representative.

Now I come to a portion of my talk which sounds almost obvious in its naivete; that is, what do you do with a program when you obtain it? There is, of course, the immediate reaction -- use it, of course! But I wish to comment about those things which will permit you to use the program in a much more effective fashion, and I wish to contain my comments exclusively to Type I and Type II programs.

When you receive such a program from PID, the first question that should be on your mind is what constitutes the totality of this program, or more specifically, what is "it"? A Type I/II program is a lot more than a deck of cards. It is an aggregation of documents and machine readable material, each of which has a specialized function and all of which are designed to permit maximum efficiency in the installation and running of the particular programming system in question. All of these items are described in a single document which is also part of the package you receive. In my experience, this document is generally thrown away shortly after receipt of the program.

This document is termed the Program Material List.

(SLIDE #11)

In my view, such a document is the most important item which you have received because with it the contents of the product are described, the volume of the material you have received and the currency of each item is specified. With it you will be able to keep your product current. Please note on the program material list which I have shown here that this particular product, which is the 1130 Card Fortran Compiler, consists of five items. One of these is a manual whose number is C26-3629-1.

The last digit, that is the -1, indicates the currency of this particular manual. Two of the items are decks of cards, the first of which is the Fortran Compiler itself at 933 cards, and the second is a sample Fortran program at 101 cards. There are also two items of informal documentation, each of which has an identification key and each of which is important in the scheme of this program. Please note that the top of this document describes a Fortran Compiler at a specific currency number; that is Version 1, Modification Level 1.

You will have such a program at that currency level only when you have all five of these items, each of which is at a specific level. If you had ordered this program when it was at Version 1 Modification Level 0 and subsequently had received Modification Level 1 and had applied it properly, your Fortran Compiler would then be at 933 cards. On the other hand, if you ordered this program at Version 1 Modification Level 1, you receive in effect Modification Level 0 Compiler with Modification Level 1 applied which is by definition 933 cards. As subsequent modifications are received by you and the machine readable material applied to the deck, the quantity of cards will alter and so should your material list. The alteration to the material list is itself described in a document accompanying the specific modification.

Instructions are sent with each modification which describe how the addition of that modification affects your current package; therefore, visualize yourself at some point in the future, let us say Version 1 Modification Level 3, and an error appears. The first question that is generally asked by the Field Engineer or Systems Engineer servicing your account is "What is the currency of your product?" The statement "I am at Version 1 Modification Level 3" is not within

itself sufficient. Rather what is your deck supposed to look like if you are at that modification? What other publications should there be with that modification level and what is the currency of each of those items? It is an appropriate set of answers to this type of question which enable you to determine whether or not you really have the product.

Please note that you will receive maintenance not only to the machine readable material contained in the product but to every item contained in the product. Should a subsequent operator's guide be published at a -2 level, you will receive it automatically. Should any of the informal documents be altered, you will either receive refreshed copies or instructions on the updating of the copies in your possession. Not only are new manuals sent to you when they are printed but should change pages, called Technical Newsletters (TNLs), be issued, you will receive them also.

Thus, getting back to my original comment of what do you do with the package when I obtain it, I submit first that you should identify it and by "it" I mean all portions of it. This is done through the material list. The second thing that you should do with your package is to be scrupulously careful about maintaining it. In the case of

machine readable material, it often presupposes that you never employ the program received from PID but rather a copy of it. In some cases the media of transmittal is too expensive for you to have this luxury but a backup copy of the machine readable material should always be made available. As an example, in the case of 1130 Disk Monitor Version II, the media of distribution is disk; and when the user punches out its contents, a copy of this deck should be made. The original punch out should be retained in the event of an unrecoverable error.

In my view the proper utilization of a program for its ultimate design consideration, that is solving problems, is best achieved by a careful attention to the contents of the package and the maintenance to each of these items.

In the case of Type III/IV programs, the mechanism for reporting errors is not as formal as in the case of IBM Type I/II programs. In this former case, communication should be directly with the author.

I have dealt at some length with Type I/II programs which are basic. By basic, I mean that material which is sent to you in fulfillment of our commitment to have a program of a nature such as we described.

There is also material available known as the optional program package which is generally the source code from which the basic package was developed. The media of distribution for optional packages varies with the size and complexity of the package. Generally, however, the availability is limited to magnetic tape. There is a program material list associated with optional packages which describes each item in the package and its currency.

The essential difference between a basic and optional package is that there is no automatic maintenance for optional program packages. We do assure that at any point in time the most current one is always available from PID. Thus, at any point in time should you request the optional package, you will receive the current one. However, the responsibility of ordering another or updated version of the optional package rests with you, the ultimate user.

One final point before we get off this subject is that there is generally a variety of literature called reference literature which is not shipped with part of the basic package. The fundamental decision associated with literature going with a program depends upon the anwer to the question "Does the user need this document in order to properly install and run this program?" When the answer to this question is

"yes", the document becomes a part of the basic package. If the answer to this question is "no", references are made to the document in several locations and ordering it is left to the discretion of the user.

Each of you will over the next few years have occasion to draw upon the resources of PID. We look forward to an opportunity to be of service to you and to provide those items which will permit an abundant utilization of your equipment. While we are scrupulously careful about production techniques, you may from time to time encounter difficulties directly associated with the packaging. We would be most grateful if you would communicate the nature of those difficulties to us on the special form which is also enclosed in the package shipped to you.

(SLIDE #12)

However, non-functionability of a program should never be reported to PID unless the cause is specific to the packaging. Very frequently we receive the type of document I have displayed here with a variety of technical questions on it. These are, of course, forwarded immediately to an appropriate department but the return of an adequate response is often delayed by this vehicle of communication.

In the case of technical difficulties, the most rapid response can be obtained by communicating the nature of the difficulty to your IBM representative.

Thank you for the opportunity to have spoken to you today, and now I would like to open the floor to any questions which you may have.

SESSION REPORT

COMMON - Chicago

Session Number Wed. A 3 Session Name Systems & Programming Project & Open Shop Disk File Maintenance

Chairman Paul Bickford

Time 8:30 - 10:00 A.M. Attendance (No.) 34

Speakers First Presentation: Pete Woodrow; Second Presentation: Larry Baker, Sam Lynch, Beryl Cording and Ed Hess

Synopsis of Meeting Open Shop Disk File Maintenance

Well attended and this topic was discussed in detail. Several questions were answered. Presentation time was 35 minutes. Panel discussion followed with each member's presentation.

There was considerable confusion about what the panel was going to present. The abstract was misleading and several people thought the whole session was devoted to Open Shop Disk File Maintenance.

A DISK FILE MAINTENANCE SYSTEM FOR THE IBM 1130

by

Peter J. Woodrow, Associate Consultant
Aeronautical Research Associates of Princeton, Inc.
Princeton, New Jersey

Presented at the Chicago COMMON Meeting, April 8-10, 1968

ABSTRACT

This paper describes a disk file maintenance system for the IBM 1130 consisting of two program segments and a strategy designed to maintain in general a single disk cartridge operating environment in an open shop (or closed shop) computer operation. The system is designed to give the computer operations management an up-to-date report on the user ownership of each user file on the disk as well as a listing by user of all current files, file lengths, file type, file creation date, and total disk words per user. Also included in this paper are detailed operating instructions for each of the programs in the system.

I. INTRODUCTION

A. Computer Operations Environment

Aeronautical Research Associates of Princeton, Inc., is a small independent research and consulting company engaged primarily in government sponsored research related to various areas of aeronautical sciences. In the research process we have used a computer since 1958. In June 1966, to increase our computing capability, we installed an IBM 1130 and have since built usage up to more than 250 hours per month.

Because the size of our programming staff has always been limited, a number of staff engineers have been encouraged and assisted to learn programming and computer operation. At present, approximately 30% of the staff is able to program and operate our IBM 1130 computer. Up to now we have operated the computer in a completely open shop mode and have permitted any capable engineer to operate the computer at any time of day, night or weekend. Until recently no restrictions were placed on usage. However, of late, increased usage has forced the imposition of a number of regulations for efficient computer utilization. One of these regulations restricted each user to a maximum of one half-hour each time he used the computer. The second regulation forced each user to account for each of his files on the disk.

B. File Maintance Problem

Efficient utilization of an IBM 1130 with a single disk drive requires that operation be restricted to a single disk cartridge as much as possible, especially when each user is restricted to a half-hour of computer time. Unfortunately, in a completely open-shop environment, a single disk cartridge is rapidly filled up. In a research organization in particular, many programs are short-lived and many of those that do enjoy a long-life are used relatively infrequently. However the IBM 1130 Disk Monitor System, unlike

many large computer systems, offers no facility for identifying the ownership of each file on the disk. In addition, most users are lazy and of short memory with regard to their old disk files.

There are two possible solutions to this problem within the framework of the Disk Monitor System, neither of them very successful The first is the use of the temporary job mode provided for by the monitor system. This solution will work whenever each subjob of a job executes successfully. However, if, for example, a compilation subjob results in compilation errors, any later execute subjobs will be bypassed. If a new job is initiated (temporary or not), any files previously created are destroyed (in Version II of the monitor even a cold start card initiates a new job). Another failing of this solution is that, even if binary cards are used, the card read time will cut down efficiency of computer utilization. The solution is, nevertheless, practical and necessary for programs using large temporary data files.

A second possible solution for limiting disk file buildup is to dump the contents of LET (Location Equivalence Table) at intervals and attempt to identify files that are no longer needed. This was the procedure we actually employed for some time. Unfortunately, the procedure is impractical because there is no easy way to transfer information from one LET dump to the next. Thus each user is required to spend an inordinate amount of time identifying his files. From experience it can be stated that the average user does not have the patience or time and will object strongly if his undeclared files are deleted by management.

C. Solution to Disk File Maintenance Problem

The ideal solution to the problem of identifying ownership of user files is to modify the monitor system to store in LET a user ID number which would appear on every JOB card. This solution is actually quite practical and might also help eliminate some problems

caused by two users coming up with the same file name (in a case of conflict, the user ID could be used to resolve the conflict). Unfortunately, this solution would require extensive, unsupported modifications to the monitor system and represents a commitment we are unwilling to undertake.

An acceptable solution had to satisfy a number of requirements, not the least of which was a small amount of programming. One key requirement was that each user should bear the primary responsibility for identifying his files without requiring an unreasonable amount of the user's time. Other requirements were that the system not require extensive amounts of computer time or disk space and that it be open ended to provide for system expansion. The adopted solution also had to provide a relatively easy way to specify the deletion of inactive files.

The solution finally arrived at was adopted in two phases, the first designed to provide up-to-date information on each file on the disk and the second designed to provide operations management as well as users with a listing by user of all current files on the disk. The completion of the second phase was actually forced by the arrival of Version II of the IBM 1130 Disk Monitor System. It enabled us to provide each user with a list of his current files on the Version I master cartridge so that he would know which of his files to transfer to the new master cartridge.

The first phase consists of a program which inputs a deck of cards containing file names, stacker selects cards specifying deleted files, and punches new cards for files not specified by the deck. All cards contain file name, file type, file length, and date card was punched. The first phase also consists of generous amounts of user cooperation (enforced by occasional threats to delete unidentified files).

Files cards are kept in a card tray near the computer. The cards are sorted into four categories,current-identified, current-unidentified, delete, and deleted. A memo circulated to all users stated

that on Friday morning of each week a list of new files (i.e., created since the previous Friday) would be posted. Users would be required to punch their identification (see below) into each card specifying a new file belonging to them sometime during the subsequent week. The memo also stated that on each Friday morning a list would be posted of files to be deleted sometime over the weekend unless the user took immediate action. This latter list consisted of all files for which either cards had been moved to the delete section or cards still remained in the current-unidentified section. Initially users were given two weeks to identify files cards (management took responsibility for punching identification on all cards specifying system files (basic IBM library plus ARAP library)).

Except for the inevitable occasional prodding of users, the system has worked quite well. Users have in general been cooperative, largely because the system has operated to their advantage in that the average user does not have to worry about changing disk cartridges. The primary value of phase one, per se, lies in the fact that a number of files, created by users, have a useful life of only a few days. Thus, rather than bother to specify identification, most users allow such files to be deleted for them. Thus, a number of files have an enforced lifetime of less than two weeks.

As some indication of the usefulness of phase one alone, when the system first went into operation, some 100 files were found to be unneeded (the deletion of which took over six hours). Whenever the program is run (usually at intervals of about two weeks) at least twenty (and often as many as forty), new files have been created and no more than five deleted in general. Of the new files, at least 25% remain unidentified or are specified as delete and are deleted when the need arises. Until quite recently, phase one alone was sufficient to provide enough room on our primary disk cartridge for all but commercial applications users (these have always been kept on a separate disk).

Unfortunately, phase one yielded insufficient information to prevent the primary cartridge from becoming overloaded. Thus, phase two was completed. It reads the current deck of files cards, sorts these by user, by project, and by file name, providing for each unique ID totals of the number of disk words used by that ID. With the use of this latter program at more infrequent intervals we expect to maintain single cartridge operation for the foreseeable future without undue hardship worked upon the users.

The following sections describe the two programs (LETUD and SPMDD) comprising the two phases of the system. Also included are some hints as to ways of keeping a cartridge relatively free of outdated files.

It should be mentioned at this point that, since all of the information resides on punched (interpreted) cards, additional programs (or modifications to existing programs) may be written to provide other forms of information. For example, we may find it necessary to modify LETUD to stacker select all files cards specifying files whose lengths have changed significantly from that specified on the files card. We may also write programs (or modify SPMDD) to provide information on only those files exceeding some specified length or with a "creation" date prior to some specified date. However, we have found that just knowing the "worst offender" in terms of large disk files is a great help in maintaining single disk cartridge operation.

II. LETUD-LET/FLET UPDATE PROGRAM

A. Design Objective

The objective in the design of this program was to provide a program which would read LET/FLET into core, sort the directory and then read a set of files cards, stacker selecting those no longer present on the disk and punching new cards for files not specified by the input deck. Since the program was to be run fairly frequently, it was important that reasonable amounts of computer time be sufficient for each run. It was also important that the program be relatively straight-forward so that programming costs would not be excessive.

As it turned out, all the objectives were met quite satisfactorily. There are, however, a few restrictions to simplify coding. These are:

1) The file type is indicated by a single digit (0 = DSF format, 2 = DCI format, and 3 = DDF format).

2) The maximum file length that can be handled is 99,999 words (if the actual length exceeds this, the first digit will be a 9 and remaining digits will be correct). It was felt that even a file of 99,999 words would be unreasonable for a master disk cartridge.

3) The length of the file as specified on an input card is ignored. Thus, even if the length has changed radically since the card was punched, no indication of this fact is given.

4) The version of LETUD designed to operate with version 1 of the monitor system only examines LET (not FLET). In addition, the length punched includes any padding necessary at the time the card is punched. This version of LETUD will also destroy the resident monitor (in core) if LET is "fouled up" in a particular fashion.

5) The version of LETUD designed to operate with version 2 of the monitor system examines both LET and FLET, but does not indicate

on the punched card which directory contained the file. In addition, this version ignores all padding (1DUMY) entries in LET/FLET. If LET/FLET is "fouled up" in a way such as to cause incorrect operation of this program, the program exits to the dump entry point in the resident monitor to dump all of core and then go on to the next job.

6) If LET/FLET is sufficiently "fouled up", either version will destroy core. In other words, both versions assume a maximum directory length which may be exceeded if pointers in LET/FLET are destroyed. Neither program checks for this condition.

7) Both versions require 8K of core and neither version could benefit by an increase in core.

8) The version of LETUD for monitor, version 2, does not support multiple disk drives. The program could however be easily (but crudely) modified to provide such support. The basic design is, however, such as to provide support for at most one drive each execution.

B. Program Design

In order to simplify design (with very little increase in running time), the program inputs all of LET/FLET twice, the first time to sort the file/entry names alone and the second to insert the file length or link to the main entry point. During the first pass over LET/FLET the program reads a sector at a time, removes the type indicator bits and length, and packs the result into as few words as necessary (1DUMY entries are also removed). An interchange sort is then done on the current sector. The result of the interchange sort is then merged with the master list in core. It turns out that 8K is sufficient to hold all of LET/FLET plus the program.

The second pass then rereads LET/FLET a sector at a time, looking up each name via a binary search on the master list, and

inserting file type and length (in disk blocks) or , if the name is a secondary entry point, the address of the primary entry point (with bit 0 set equal to 1). Both passes require a total of less than 15 seconds for an almost completely full LET (30 seconds for a full LET and FLET).

The third pass reads in files cards, one at a time, and looks up the file name in the master list in core. If the name is present, the program zeros the length/link word corresponding to the name. If the name is not present, the program stacker selects the card. The first card containing a blank in column 1 signifies the end of input and is immediately stacker selected. The master list is then scanned and for each name having a nonzero length/link word a card is read, checked for blanks, and punched containing the file name, file type, file length/link and any other information in the master card (see below). All punched cards are also selected to stacker 2; thus stacker 2 at job completion contains all cards specifying file status alteration. The program exits to the resident monitor when the master list scan is complete.

The card read pass proceeds at essentially card read speed because of the binary search (even for a full LET/FLET). As a result of the program design the cards may be in any order desired (except that blank cards must follow the first blank card). Because of this feature, cards stacker selected as a result of file deletion should be saved for awhile. It may be that this program was run at a time when the user had momentarily deleted the file. If all cards for supposedly deleted files are added to the end of the current deck, the cards will still be stacker selected if the files are still missing, but, if the file has been put back on the disk, the card will not be stacker selected and may thus be reinserted into the main deck, saving the user some time. As a final note, the program will accept more than one card with the same file name; it will cause no trouble (although reasons for its presence are not clear).

C. Program Operation

1) The XEQ card should specify DISKO (O in column 19).

2) The first card following the XEQ card is the master card. Any columns punched in this card are gang-punched into every card punched by LETUD. If the master card has nonblank columns where the program will insert variable data (see below), these columns will be overlaid by the variable data. The punching speed is regulated by the last nonblank column to be punched (whether variable data or gang-punch data). Any information may appear on the master card (except see III. SPMDD for restrictions if the cards are to be later processed by SPMDD); in particular, the master card may be used to gang-punch a system ID for all files in a freshly generated system disk cartridge, or the date on which the LETUD is being executed.

3) Following the master card comes the deck of files cards, the only constraint on these being that column 1 be nonblank and that any file name be left-justified (i.e., these cards need not have been punched by LETUD).

4) Following the input deck, and signifying the end of the input deck, is a blank card. This card is not punched, merely stacker selected. Thus, if it has a different corner cut than the remainder of the cards, it serves to separate files cards for deleted files from files cards for new files. Note that the program does not recognize monitor control cards.

5) Following the separator card comes a deck of blank cards. Each card is read to make sure it is blank before it is punched.

When the program finishes punching cards for each new file, it exits to the resident monitor. Each card punched by this program has the following format:

Columns	Contents
1-5	File name left-justified (consists of letters, numbers, and/or blanks)
8	File type (C=DSF format, 2=DCI format, 3 = DDF format)
10-14	File length in decimal words right justified with leading zeros punched OR name of primary entry point (DSF format only) if this card specifies name of secondary entry point.

All other columns contain information as punched on master card.

This program has only one internal programmed wait (other than the normal CARDO, DISKO waits) and that occurs if a card about to be punched contains a nonblank column. The version for use with monitor, version 1, will wait for each nonblank column and will then proceed to punch the card without rereading to check for blanks. The version of this program to be used with monitor, version 2, waits only once if there are any nonblank columns and then rereads the card, again to check for blanks, before finally punching the card.

III. SPMDD-SORT AND PRINT MASTER DIRECTORY DECK

A. Design Objective

The objective in the design of this program was to provide the ability to read a card deck containing information punched by LETUD and additional information punched by the user, sort the deck (internally) first by user ID and then by file name. The information was then to be printed on the 1132 line printer in a reasonably condensed format. As with LETUD this program was to be designed at minimum cost and to require minimal running time.

The design objectives have been essentially met, except that, as with LETUD, a number of restrictions were imposed to simplify the design and speed operation. Those which are not part of input/output format (described later) are detailed below:

1) The principal restriction is that in an 8K computer only a limited deck can be sorted and printed. The size depends on the number of different user ID's present in the deck. On an 8K machine all of LET may be sorted (maximum of 840 file names) if all ID strings are of less than 3 elements and there exist no more than 100 unique ID strings. However, this restriction will not be a burden, since procedures are available to omit from the sort all files belonging to selected ID strings (e.g., system library). In addition, a change of a control card in each of the links will allow for larger table area if more core is available.

2) The program is designed to operate under monitor, version 2, only, but since decks punched by LETUD are the same for both versions, this program can process a deck produced by LETUD operating under monitor, version 1. The program cannot be reasonably modified to assemble under monitor, version 1, as it makes extensive use of the new mnemonics available under version 2.

B. Program Design

The program was designed to accept identification strings for each file up to five elements (names). These elements form a hierarchy with the first element becoming the major sort key, the second the first minor sort key and so on, with the file name being the lowest sort key. The first element might be assigned to company name, the second to project, the third to program, and the fourth to programmer. The assignment is entirely up to the computer operations management making use of this program. Once fixed, however, the order cannot be changed. The first ID name on each input card is always the major sort key, etc.

As a result of simplifying program design, names should contain only letters, digits or blanks. Any other character will result in a blank on printed output and a "garbage" character on punched output. In addition, the program accepts and prints/punches only the first five characters of each ID name although the input card may contain up to ten characters per ID name.

To provide maximum processing speed, the program is divided into two links, SPMDD which processes all input (and sorts) and PRMDD which provides all printed or punched output. In addition, to economize on design effort and speed execution, the sort is done via a linked list structure in core "on the fly" as the cards are read.

The table area used by the program consists of a mixture of four-word ID name blocks and six-word file name blocks. The contents of these blocks are:

ID Name Block

Word	Contents
1	Address of next block in current ID string (next ID element in hierarchal order) or address of last block in file string corresponding to this ID string if this is last element in a valid ID string (bit 0=1 in this case) or 0 if no file string attached to this ID string yet.

Word	Contents
2	Address of block at same hierarchal level containing ID name in an ID string whose first "N" names are the same as those in current string. The new string sorts after the current string. If this word is zero, there is no such ID string yet.
3,4	ID name in name code (left-justified). If bit 0=1, this current string up to this point was specified as a control ID string (see C. Program Operation).

File Name Block

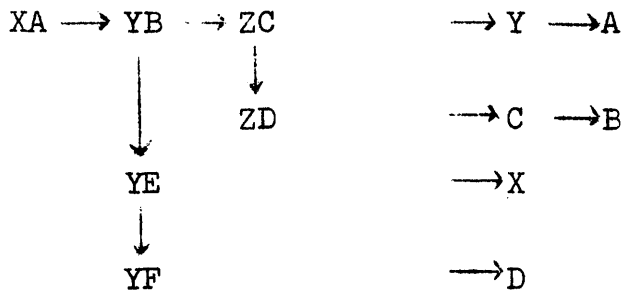
Word	Contents
1	Address of a file block to sort before this file name and attached to same ID string. If this word is zero, this is the last (sorted first) block attached to ID string.
2	Creation date (as specified on input card) in code - (((Year - 66)* 12 + Month - 1)* 31 + Day - 1).
3,4	File name in name code (left-justified). Bits 0,1 specify file type (0 to 3).
5,6	File length (in words) or name of primary entry (if this name is that of a secondary entry point) in name code (left-justified). In the latter case bit 0 is set to 1.

Example of Link Structure:

Input:

File Name	ID Names
A	XA YB ZC
C	XA YB ZD
B	XA YB ZD
D	XA YF ZC
X	XA YE ZD
Y	XA YB ZC

Link Structure:



This structural form internally provides quite rapid searching (not as fast as binary, but two passes would be needed to produce a binary table) to determine whether the same ID string has already been input. It also permits efficiency with variable length ID strings (variable no. of elements). Only as many blocks are used as there are nonblank ID name elements (except if XA, XA XB are both valid ID strings, then the program would force XA (blank), XA XB).

This scheme also permits sorting "on the fly" as input cards are read. Since a file string attached to an ID string is linked in reverse sort order, then, if the input deck is sorted by file name in normal order, no searching will be necessary to add the new file name to the file name list. Since the deck produced by

LETUD is in lexicographical order (and can be easily kept that way by appropriate filing of new cards), this procedure minimizes sort time. In fact, for an ordered (by file name) input deck, input proceeds at card-read speed (300 cards/minute).

Once input is complete it is a simple matter using a pushdown list to trace through the link structure and print/punch output in sorted order (the links in each file name string do have to be reversed, but it is desirable to scan the list to get the total length of all files prior to output anyway - this does not seem to slow down output in any way).

The current version of the program seems to provide all desired objectives with the possible exception of the ability to retain only those files created before (after) a certain date or only those files longer (shorter) than some specified length. For typical directory decks (with the exception of linking time) it operates at least at 1442, 1132 speeds.

C. Program Operation

1) Neither link uses disk I/O so that column 19 should be blank.

2) The first card following the XEQ card is the parameter card and specifies which options are desired.

Column	Contents
1	Blank - if the program should ignore all control ID strings (see below) Nonblank - if the program should process only those ID strings which start with control strings.
2	Blank - see column 3 Nonblank - no printed output
3	Blank - print all files Nonblank - print only ID strings and totals
4	Blank - punch ID string cards Nonblank - no punched card output

The next ten columns correspond to each of the five possible ID's, a two column set for each ID. The first column of a set specifies the lowest hierarchal level (highest-numbered) ID element sorted upon; i.e., if column 2*N + 3 is nonblank, but columns 2*I + 3 (I=1, N-1) are blank, then ID's N, N+1, etc., are ignored and thus not part of the sort.

The second column of a set specifies when page ejects occur; i.e., a page eject will occur whenever IDn takes on a new value (name) if columns 2*I + 4 (I=1,N) are blank.

3) Following the parameter card is a set of control ID string cards specifying strings either to be ignored (column 1 of parameter card blank) or to be processed (column 1 nonblank). It is important to note that, if, for example, XA YA and XA YB are valid ID strings, then the control string XA will apply to both XA YA and XA YB (i.e., both will be either ignored or processed). However, XA YA applies only to XA YA and not XA YB (but would, of course, apply to XA YA ZA, etc.). The format for these cards is the same as that for files name cards (see below) except that columns 1-29 are ignored and should be blank. An EOF card (any card containing a 7-9 punch in column 1) terminates the set of control ID string cards (the set may be empty, but the EOF card must be present).

4) Following the EOF card is the master directory deck containing cards of the following format:

Columns	Contents
1-14	Same as LETUD output format (columns 9 and 15 must contain a nonnumeric character and file length must have leading zeros punched).
20-27	File "creation date" as MM/DD/YY (where M=Month, D=Day, and Y=Year), MM must be between 1 and 12, DD between 1 and 31, and YY greater than 65. Leading zeros may be

	omitted (numbers must be right-justified),but not replaced by blanks; e.g., 1/12/68, 1/01/66, 1/1/69 are valid, but 1/_1/69 is not. The column following the last digit of the year must be nonnumeric. The results are unknown if this field is left-blank (except that it will not affect any other field).
30-34	First ID name (left-justified preferably)
40-44	Second " " " " "
50-54	Third " " " " "
60-64	Fourth " " " " "
70-74	Fifth " " " " "

Note: It is permissible to have all ID name fields blank.

Following the master directory deck must be another EOF card signifying end of input (if punched output is specified, blank cards should follow the EOF card).

5) After SPMDD has read all input cards (through the second EOF card), it links to PRMDD to perform all output functions. Output is a string of ID names with the total disk word count at the right of the line followed by a list of files (read from left to right), five files (with full identifying information) per line. As the ID string line is printed, an ID string card is punched in the same format as (and useable as) a control ID string card.

Note: Through judicious use of control ID strings it is possible to process practically any master directory deck. Any time, for example, a files card contains an ID string (which starts as one of the control ID strings) and the ignore "switch" is set, the card is skipped and no storage is wasted. If no file names are desired, then only one file block per unique ID string is needed (to contain total length which is accumulated as cards are read).

Programmed waits will occur if the program runs out of table space in SPMDD (program start will cause exit to resident monitor) or if card about to be punched in PRMDD is nonblank (program start will cause the card to be reread in a check for blanks before it is punched).

IV. CONCLUSIONS

We have found our Disk File Maintenance System for the IBM 1130 to be extremely useful. It has enabled us to uncover some very interesting facts:

1) SSP (Scientific Subroutine Package) requires approximately 60,000 words on the disk (nearly 200 sectors) or about 12.5% of the entire disk. We do not intend to put SSP on our monitor, version 2, master disk since less than 10% of SSP has been used to date. The user can put on what he needs.

2) By judicious deletion of unnecessary (for large original 1130 installation) files from the monitor, version 2, system library, the user can recover over 16,000 words of disk space (a saving of 40% over system as delivered).

3) We uncovered one user using almost 60,000 words of desperately needed disk space (however, we were unable to convince him that our needs were greater than his).

SYSTEMS AND PROGRAMMING MANAGEMENT*

Laurence H. Baker

The success of any computer installation depends upon many factors. Some of the more important ones are:

A. The objectives which the data processing effort is designed to fulfill.

B. Its place within the organization structure.

C. The organization of the systems and programming effort.

D. Top management involvement in the computer activity.

E. The mix of applications processed.

The key to success depends upon how well the systems and programming activities are managed. If these resources are planned and used properly, there will not be major problems in meeting the objectives of the data processing effort.

OBJECTIVES

Each objective of the computer activity must be studied and analyzed carefully relative to the resources required to accomplish it. If each objective can be analyzed and planned realistically, all other factors which can influence the success of the computing activity have been properly adjusted in the organization.

A variable mix of applications in different periods of the year can pose added problems requiring competent individuals and knowledgeable individuals in more than one area of application.

* Contribution to panel discussion on Systems and Programming Management at COMMON Users Group Meeting - April 10, 1968.

ORGANIZATION STRUCTURE

The place of the systems and programming effort in the organization can have a large measure of influence on its future. If they are established at the corporate level such that systems work can be carried out as a staff function, there may or may not be cooperation. Probably, this depends upon the involvement of top management in the review and encouragement of computer applications.

A useful alternative approach to the problem is to establish one or more key systems and computer liason personnel within each division. If these are key positions and report to the general manager or president of each division, they can play a very important role in the successful development of computer applications. A suggested organization chart is shown in Figure 1.

ORGANIZATION OF SYSTEMS AND PROGRAMMING EFFORT

This must be organized so each individual understands lines of responsibility, policies, their job and opportunities for self-improvement and advancement. A tool which can be used to achieve this end is a manual for Systems and Programming Personnel which each individual is given the day he joins the systems and programming staff.

The essential contents of such a manual are listed below:

A. Policy
B. Standards and Forms
C. Publications and Instruction
   Courses Available
D. Utility Program
E. Routine Support
F. Program Library
G. User Program Documentation

SYSTEMS AND PROGRAMMING PROJECT MANAGEMENT

I. Importance of controlling the operation whether it is a new installation, an upgrading of existing equipment, or the addition of new applications.

   A. The larger the undertaking, the more important the controlling and the scheduling of activities becomes.

   B. The goals should be clearly defined, the specifications should be well-documented, and should be rigidly adhered to throughout the entire span of the project.

II. A method of project control is essential to the successful completion - on schedule - of the effort.

   A. Manual methods lend themselves to small projects of a minimum number of inter-related activities over a relatively short period of time.

   B. Computer-oriented techniques are invaluable for analyzing and controlling complex projects that normally would tax manual approaches.

III. Scientific Management techniques for project management/control are not new although as time goes by more people are becoming aware of their availability and potential. The more complex the endeavor, and the more difficult it may appear to implement such a technique - the more desperate may be its need!

   A. Project FIRM, written to run on the 1401 or the S/360 Model 30 or Model 40 in compatibility, and PERT-CPM for the 1130, are two simple and yet very effective programs for controlling project schedules. These are techniques that are simple to use and easy to learn.

   B. More sophisticated programs are Project Control System/360 which runs under DOS/360 and Project Control System/1130 which runs under the Monitor. They are designed to handle more complex networks and the elements of resource allocation and cost factors are included.

   C. Project Management System/360, an extension of NASA PERT/Cost, runs under the OS/360 and has been developed for extremely complex projects.

IV. However the project is approached, however it is deemed wise to control its many facets - none should ever be started without a plan. The more comprehensive and thought-over the plan, the greater the chance for a successful completion -- on schedule. Management will respect our professionalism and be more amenable to our needs (additional manpower, overtime, etc.) if a well-developed and orderly schedule is used.

V. Expertise developed in the application of these Management Science techniques to the control of the Data Processing projects, can be readily applied to other programs within the corporation -- whether it be building construction, assembly line development, or engineering laboratory project control.

REFERENCES

1. Management Handbook for the Estimation of Computer Programming Costs

   E. A. Nelson

   System Development Corporation March 1967
   Clearinghouse for Federal Scientific and Technical Information

2. Toward Better Programming Management

   M. L. Rubin

   Journal of Data Management
   Vol. 5 (12) December 1967

3. A Technique for Improving the Management of a Computer Installation

   R. L. Patrick

   DPMA (Quarterly July 1965)

4. IBM DP Techniques

   Organizing the DP Installation (C20-1622-0)
   Mgmnt. of the Punched Card DP Dept. (C20-1611-0)
   DP STDS (S360) IBM United Kingdom

5. Programmer Selection and Evaluation

   A. COMMON Proceedings Dec 1967
   B. COMMON Proceedings Sept 1967
   C. IEEE Transactions on Human Factors in Electronics
      Vol. HFE-8 (1)
      March, 1967

      An Explanatory Investigation of Programmer Performance Under On-Line and Off-Line Conditions
      E. E. Grant & H. Sackman

6. EDP: Its Controls and Economics

J. V. Miccio

Journal of Data Management June 1967

7. Management by Crises

J. A. Campise

Journal of Data Management April 1967

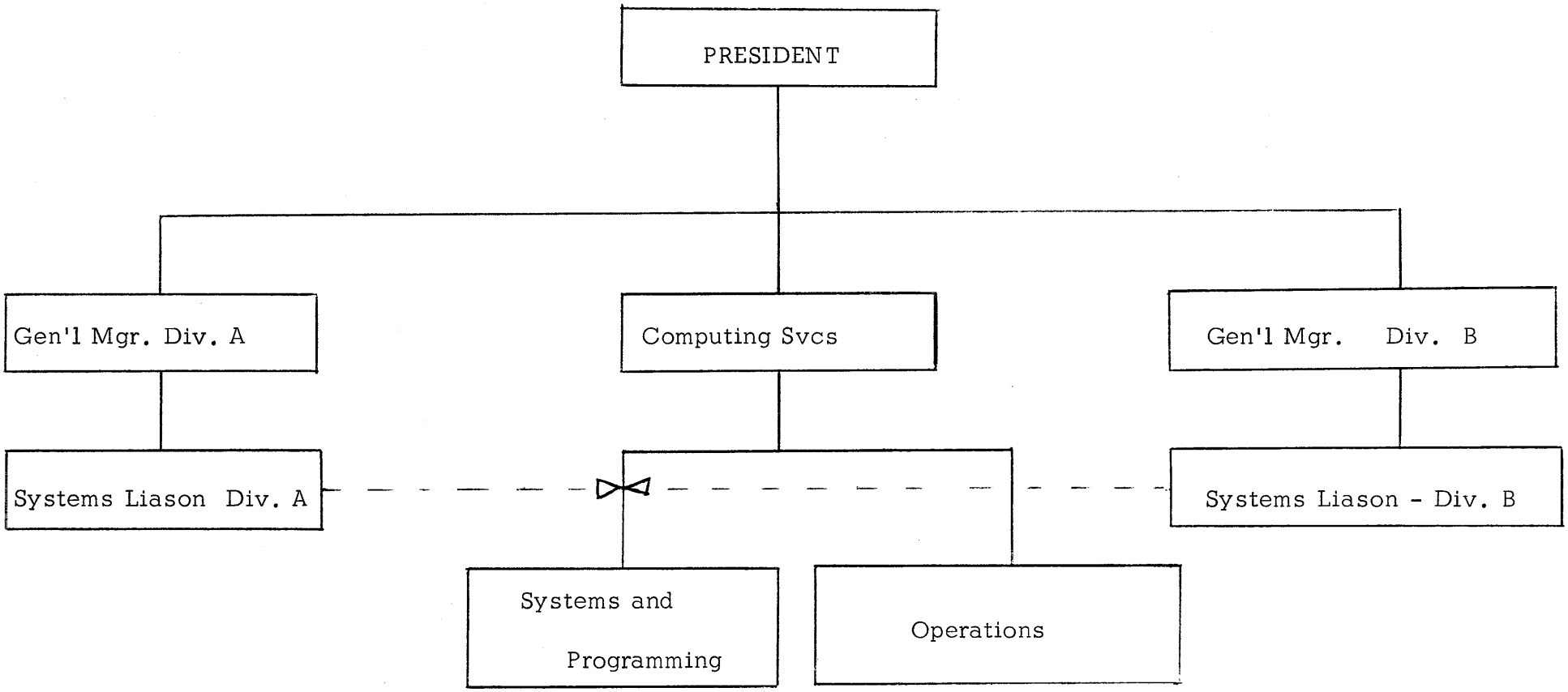

PRESIDENT
Gen'l Mgr. Div. A
Computing Svcs
Gen'l Mgr. Div. B
Systems Liason Div. A
Systems Liason - Div. B
Systems and
Programming
Operations

FIGURE I

SYSTEMS AND PROGRAMMING PROJECT MANAGEMENT

Panel Discussion Session III-A 3

By: Beryl Cording, Orlando Utilities Commission

The implementation of any project should devote a considerable effort to the definition of the problem and to the system design. It has been my experience that more effort should be expended in this area than in the actual writing of the program and its implementation. Considerable time should be spent studying the interplay of activity and the flow of records involved in the application. It is important to document this phase and to review it in detail with the affected personnel in the other divisions of the company. This review should establish that the activity analysis accurately reflects the operations now being conducted and should be approved not only by the involved operating personnel but the next higher level of management.

All of the activities and events which must take place to accomplish a conversion should be enumerated, scheduled, and checked off as they are accomplished. It is most important to arrange these activities in the proper order and establish a reasonable period of time of their completion. Comparisons of actual performance to the scheduled are very helpful to the project manager in focusing attention on the problem areas. The thoroughness of the preparation of this schedule will determine the ease with which the whole project can be managed and brought to a successful implementation. A conversion schedule or bar chart is the very least that is required for a simple application. At the other end of the scale, some form of critical path method, using the computer to evaluate the attitude or progress at any given point, is the most effective tool to manage any complex project.

When management is faced with a new application in a fully converted shop, it is even more important to follow a strict schedule since attention can be focused primarily on the new application. Management must also worry about system maintenance, the modification of operating programs, and other changes which involve the programming staff. To provide depth and flexibility, it is important that maintenance of the operating system be rotated among several staff members. Care must be taken to assure that adequate documentation of the utility and other IBM-supplied programs is provided. The use of a check-off sheet to keep track of the progress of the development of a new application is a very practical approach. Similar to a conversion schedule in scope, this sheet presents a listing of the activities that must take place from the general system design all the way through the establishment of operator instructions after the program has been thoroughly tested.

The management of a data processing installation must be planned and organized to delegate some of the responsibility to the appropriate staff members since no manager can do the entire job himself. The schedules and reports which are prepared by the staff are tools to be used in evaluating the position of the programming staff and the individual performance of the programmers. If this approach is taken and the staff has an understanding of why this is being done, projects and applications can be converted successfully and on time.

SESSION REPORT

COMMON - Chicago

Session Number WED A4 Session Name 1130 LP/MOSS Users Experiences

Chairman Dan Koster

Time 8:30 to 10:00 AM Attendance (No.)

Speakers G. Schoditsch #3438
Monsanto Co.
1700 S. Second St.
St. Louis, Missouri

Dr. Scott Hathorn
J. G. Boswell Co.
Corcoran, California

Synopsis of Meeting Dr. Hathorn spoke on two programs he has for generation of the cost row and constraints for his enterprise resource allocation model in a farming environment. The programs are to be released through PID.

Mr. Schoditsch spoke on optimum packings for gas liquid chromatography columns. The write-up follows. Mr. Muller of IBM issued a plea for feedback on Version 1 of LP/MOSS and suggestions for Version 2. He wants to know what we are interested.

USING LINEAR PROGRAMMING TO DETERMINE OPTIMUM GLC PACKING MIXTURES

To be presented at:

COMMON Meeting
Chicago, Ill.

April 8-10, 1968

G.F. Schoditsch

MONSANTO Company
St. Louis, Mo.

March 7, 1968

Using Linear Programming To Determine Optimum GLC Packing Mixtures

G.F. Schoditsch, MONSANTO Co., St. Louis, Mo.

ABSTRACT

Gas-Liquid Chromatograpy is a commonly used analytical tool in the chemical industry. GLC may be used for both qualitative and quantatative analyses (what is present and how much of each component is present in a sample). An important element in the effective use of GLC is the proper choice of packing or mixture of packings. Linear Programming has been applied to process of selecting the proper mixture of packings.

INTRODUCTION

In GLC, a vaporized sample is injected into a stream of carrier gas (usually helium) and adsorbed onto a packing. The packing is one or a mixture of high boiling components deposited on an inert substrate. As the sample/carrier stream passes over the packing, each of the components of the sample is 'adsorbed' onto the packing. As the sample/carrier stream becomes depleted of sample, the components of the sample 'desorb' from the packing. An instrument measuring the physical-chemical properties of the eluted components draws a graph (see Figure 1) that may be interpreted to determine the composition of the sample.

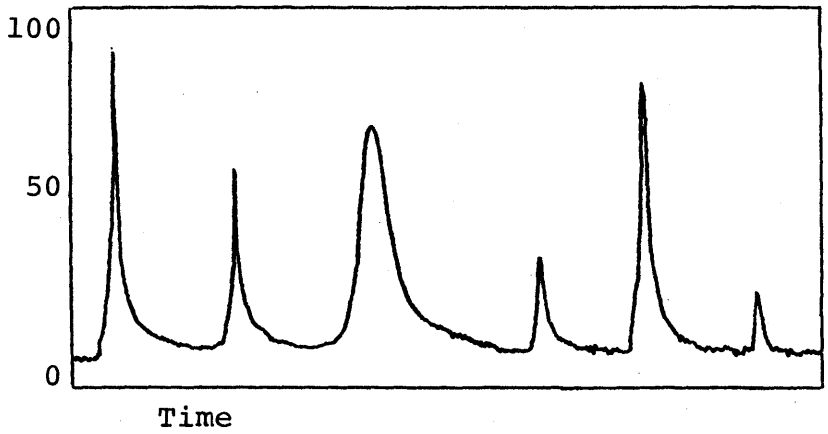


Figure 1 Typical GLC Chromatogram

Inspection of figure one shows that the output from the GLC instrument is a curve with several peaks. Each peak is associated with one component of the sample. The relative area under each peak is a measure of the amount of each component present. For a given packing material, the distance from a known reference point to each peak can be used to identify the component. As you can readily understand, if the peaks are too close together, not only does component identification become difficult, but calculation of the area under a peak is almost impossible. To improve the separation of peaks

(resolution), a mixture of packings is employed, since different packing materials give different resolution for different sample components. Figure 2 shows the relative retention times for a given sample using three different packing materials.

	PACKING NUMBER		
	1	2	3
Component			
1	1.00	1.00	1.00
2	1.95	1.79	2.41
3	2.20	1.93	4.23
4	4.90	4.20	3.84
5	5.10	6.10	3.62
6	8.10	6.39	7.04
7	7.40	9.94	6.82
8	9.20	8.73	8.48
9	20.60	24.85	11.61
10	24.50	28.15	24.88

Figure 2 Relative Retention Times

If difference between retention times for each of the components is tabulated (see Figure 3), these differences may be thought of as the effect that the packing produces when separating the material. The assumption of the LP approach to packing selection is that when several packings are combined, the net effect of each packing in the mixture is linearly additive. The object is then to find that combination of packings that will give the best resolution of peaks.

	PACKING NUMBER		
	1	2	3
Difference No.			
1	.95	.79	1.41
2	.35	.14	1.82
3	2.70	2.27	-.39
4	.20	1.90	-.22
5	3.00	.29	3.62
6	-.70	3.55	-.22
7	1.80	-1.21	1.66
8	11.60	16.12	3.13
9	3.90	3.30	13.27

Figure 3 Differences in Relative Retention Times

Other approaches to this problem[1] have been to assume a quasi-theoretical approach to the actual action within the packing column and calculate the expected separations for key components over the entire range of mixture of packings, usually in increments of about 5%. This method is usually limited to mixtures of two or three packings. This approach also makes several assump-

tions regarding the ideality of the sample mixtures. The linear programming approach uses a minumum of actual data from the actual sample to predict the optimum separation.

THE LINEAR PROGRAMMING APPROACH

Using the assumption of linear interaction between the packings, the following set of equations can be developed from the difference data of Figure 3:

$$
\begin{aligned}
D_1 &= 0.95X_1 + 0.79X_2 + 1.41X_3 \\
D_2 &= 0.35X_1 + 0.14X_2 + 1.82X_3 \\
D_3 &= 2.70X_1 + 2.27X_2 - 0.39X_3 \\
D_4 &= .20X_1 + 1.90X_2 - 0.22X_3 \\
D_5 &= 3.00X_1 + 0.29X_2 + 3.62X_3 \\
D_6 &= -0.70X_1 + 3.55X_2 - 0.22X_3 \\
D_7 &= 1.80X_1 - 1.21X_2 + 1.66X_3 \\
D_8 &= 11.60X_1 + 16.12X_2 + 3.13X_3 \\
D_9 &= 3.90X_1 + 3.30X_2 + 13.27X_3 \\
1.0 &= X_1 + X_2 + X_3
\end{aligned}
$$

The last equation simply states that the sum of the fractions of each packing must equal 1.0.
If equal proportions of each of the three packings are assumed, (that is $X_1 = 1/3$, $X_2 = 1/3$, $X_3 = 1/3$) the following set of differences are predicted:

$$
\begin{aligned}
D1 &= 1.05 \\
D2 &= 0.77 \\
D3 &= 1.53 \\
D4 &= 0.49 \\
D5 &= 2.30 \\
D6 &= 0.88 \\
D7 &= 0.75 \\
D8 &= 10.28 \\
D9 &= 6.82
\end{aligned}
$$

Inspection readily shows that D8 and D9 will be significantly larger than the other seven no matter what mixture is employed. These differences will be neglected.

Inspection also shows that D4 is the smallest difference of retention time. The object then is to maximize D4 with all other differences subjected to restraints of being equal to or greater than 0.5, 0.6, etc until either D4 becomes equal to or smaller than any of the other differences and the requested solution becomes infeaible.

Following this procedure, the results shown in Figure 4 were obtained using the LP-MOSS package on our 1130 computer. The runs were made with a lower limit on all differences (except 4) equal to 0.6, 0.7, and 0.8. The program indicated that no solution could be obtained with a lower bound of 0.8, or in terms of the real system, no mixture of real packings would give a relative retention time difference of at least 0.8 for each component. Examining the results when the limits were 0.6 and 0.7 led us to estimate that the optimum should occur (that is, the 'maximum' minumum difference would occur) when the lower limit was set to 0.688. Results of this run verified this estimate.

	VALUE OF RESTRAINT USED			
	0.6	0.7	0.8	0.688
Difference No.				
1	.991	1.026		1.022
2	.600	.700		.688
3	1.836	1.657		1.678
4	.765	.677	Solution	.687
5	2.087	2.224	Infeasible	2.208
6	1.058	.934		.949
7	.600	.700		.688
PACKING				
1	.386	.359		.363
2	.388	.352		.357
3	.225	.289		.280

Figure 4 Tabulated Results Of Computer Runs To Determine Optimum Packing Mixture

DISCUSSION AND COMMENTS

A packing mixture for the predicted optimum separation was prepared. The results of the actual mixture approximated those predicted by the computer (see Figure 5). Although one separation was not achieved, the method gave good general predictions. Unfortunately, the actual differences were not large enough to give the desired resolution. Other 'tricks of the trade' needed to be applied to achieve a satisfactory resolution. Unfortunately also, the original data were not generated using sample and operating conditions near enough to actual sample compositions and operating conditions. In any case, the predicted optimum is close to the actual optimum, and may be used as the starting point for further experimental study.

COMPONENT	PREDICTED	ACTUAL
1	1.0	1.0
2	2.0	2.1
3	2.7	2.2
4	4.4	4.9
5	5.1	4.9
6	7.3	7.9
7	8.2	8.4
8	8.9	9.6
9	20.0	22.3
10	26.3	25.7

Figure 5 Comparison of Actual and Predicted Relative Retention Times

The procedure shown here is applicable to any number of components, and any number of packing materials. All that is needed is relative retention times for a sample containing all the component for each of the packing materials. The original data should be generated using a sample with composition near that of the samples that will be routinely analyzed by the instrument.

Data for relative retention times is also given in Kovat's indices. Using the data from these tables a scheme may be worked out to generate relative retention times for a mixture without having to actually run the sample. This and several other alternates are being examined as ways of improving the method.

ACKNOWLEDGEMENT

Grateful acknowledgement is here-with given to Mr. Fred Stewart and Mr. John Hinchen of the Monsanto Co. Mr. Stewart supplied the orginal data for the problem and verified the results. Mr. Hinchen helped immeasureably in preparing the problem for solution using linear programming.

REFERENCES

1. "Computer Optimization of Mixed Liquid Phasis for Gas Chromatography", ANALYTICAL CHEMISTRY, Vol. 36, No. 2, February 1964, pp. 260-262.

SESSION REPORT

COMMON - Chicago

Session Number WED A6 Session Name 1130

Chairman D. Dunsmore

Time 8:30 to 10:00 AM Attendance (No.)

Speakers (1) Don Gardner - General Foods

(2) Marshall Hechter - IBM, Chicago

Synopsis of Meeting (1) Gardner - "Multiple Regression Program"

(2) Hechter - "1130 Assembler Programming Aids"

Wed A-6 By Don Gardner for Program Chairman

Multiple Regression Program
MRP/1130

Purpose:

The MRP/1130 performs standard multiple regression computations on a set of data consisting of a maximum of 60 variables and 9999 observations with residuals and predictions computations optional to the user. (This program was patterned after the Multiple Regression Program for the 1620, 1.6.043)

Machine Requirements:

An 8K 1130 with a 2315 Disk, 1442 Card Reader, and 1132 Printer.

Program Description:

MRP/1130 is coded completely in FORTRAN and consists of six mainline programs and seventeen subroutines for the multiple regression computations: and three mainline and eleven additional subroutines for residuals and predictions making a grand total of nine mainline programs and twenty-eight subroutines.

Output:

The standard output of MRP/1130 consists of the following:

1. Definition of variables
2. Average, variance, and standard deviation for each variable.
3. Pairwise correlation coefficients
4. Regression information
    a. the multiple correlation coefficient of each X-variable with other X-variables.

b. the b-coefficient for each X-variable

c. the standard deviation of each b-coefficient

d. the T-value for each X-variable (b/S_b)

e. the constant term, the multiple F-value, the degrees of freedom corresponding to numerator and denominator of F-value, the multiple correlation coefficient for the X-variables in the regression with the Y-variable, and the residual error.

The output of the Residuals and Predictions section of the program (if desired) is the following:

1. Definition of variables (optional here)
2. A statement of the regression equation (a listing of the variable number and the b-coefficient for each X variable in the model plus the constant term)
3. For each observation, the following are listed:
   a. the observed Y-value
   b. the predicted Y-value
   c. the residual value (the difference between a and b)
   d. the standard error of the prediction
   e. the normal deviation of the residual
4. A scatter plot of the residuals plotted against the observations
5. A scatter plot of the residuals plotted against the predictions

Input Deck Arrangement

The input deck is arranged as follows:

Calling Cards (// XEQ etc.)

MRP Header Card

MRP Variable Definition Cards

MRP Data Cards

MRP Trailer Card

Header Card

The format of the header card is as follows:

Cols	Description
1-4	Number of observations
5-6	Number of input variables/observation*
8	0 Regression Analysis
	1 Correlation Analysis (all input variables)
	2 Correlation Analysis (selected variables)

* The definition of 'input variable' is most important: an input variable is any value punched in the data cards which may or may not be included in the regression equation, i.e., one may have more variables on the data cards than one needs. The total number of input variables is punched in cols. 5-6)

Variable Definition Cards

These cards are used to define the X-variables (independent variables) and Y-variables (dependent variables) which are needed for the regression analysis(es).

These cards are either one of two basic formats:

1. Type 1 - used to define a variable directly
2. Type 2 - used to define a variable from previously defined variables.

Both X- and Y-variables may be defined directly or as a function of other previously defined variable(s). Some times a variable is defined and used only to generate another variable. This presents no problem since you can indicate to the program that the original variable is to be ignored and not used as part of the regression equation.

The variables defined directly can be coded or scaled by any amount or transformed by one of four available transformations (if transformations are important to the user, a total of nine (9) can be included for use.) They are

Square Root
Log (base 10)
Exponential
Reciprocal

A variable can be defined <u>both</u> as an X-variable and a Y-variable simultaneously. With the proper use of the variable deletion procedures, one can use a variable both ways. The important consideration is that the data need to be read only once under such circumstances. Proper handling of the trailer cards give the required results.

Data Cards

The program uses the subroutine DATAR to read the data in free format, that is, data may be anywhere on a card between columns 1-72 (the 72 may be changed to any limit) and any number of data cards may be used to represent an observation. This is particularly useful in regression work when one wants to merge two data decks together and the formats of each are different - no problem when using DATAR. The only other requirement is each piece of data must be separated from adjoining pieces of data by at least one blank column.

Trailer Card

Each regression analysis requires a trailer card to specify which Y-variable to use in the regression equation (one may specify more than one Y-variable on the variable definition cards, but of course only one at a time can be used). The trailer card is also used to specify which, if any, X-variables to delete before the inversion process. This feature allows one to define more X-variables than are needed for a given regression equation which can be useful in the following circumstances:

1. For a given dependent variable (Y) one would like to study various subsets of X-variables equal to or less than the total number of X-variables.
2. One has two or more dependent variables and the regression

equation for each involves a different set of X-variables. All X- and Y-variables for a problem like this can (and should) be defined at one time so the data do not have to be read more than once. An extreme example of this kind is when you have two problems: regress X_1-X_{25}, say, on Y_1 and regress X_{26}-X_{50} on Y_2. X_{26}-X_{50} can be eliminated for an analysis of Y_1, and X_1-X_{25} can be eliminated for an analysis of Y_2 by the use of trailer cards.

3. One has specified that automatic deletion of variables take place. If many X-variables are in the original equation, then after many successive deletions roundoff error can start the creep into the calculations.

This presents no problem since you can start over with just a trailer card eliminating those variables you do not want and obtain a 'clean' regression analysis. This happens because the unwanted variables are deleted before inversion.

The Use of the Disk

The following information is stored in the one permanent data file used by the program:

1. total number of observations
2. total number of variables (both X-and Y-variables)
3. total number of definition cards
4. index of Y-variable in equation just computed
5. residual variance

6. constant term
7. vector of subscripts for X-variables in regression
8. vector of averages and standard deviations for all X-and Y-variables
9. correlation matrix
10. inverse of correlation matrix
11. vector of b-coefficients for each X-variable in regression.

This file can be dumped on cards (149 cards) if more work has to be done (i.e., other analyses to be performed at a later time) with the particular set of variables. This feature is useful, again, so that the data only have to be read once. An example of its usefulness is the following situation: One has two Y-variables to be analyzed, but only time on the computer to analyze one. The data are read and the one analysis performed. Then the file is dumped on cards and re-entered onto the disk at a later time so that the second analysis can be done.

Residuals and Predictions

A residuals and prediction analysis is usually performed after a regression equation has been chosen. Sometimes all the original variables are in the equation and other times, a subset. In any case whenever a regression analysis has just been completed, a residuals analysis is available under sense switch control. All that is needed is a residuals and predictions header

card added in front of the complete data deck used in the regression analysis.

Highlights of MRP/1130

1. FORTRAN Coded - The nine mainline and twenty-eight subroutines are all coded in FORTRAN.

2. One Data File - the data file has been defined with the following statement:

   DEFINE FILE 1(3960,2,U,IV1)

   All variables, singly and in vectors, integer as well as real, are stored in this file. This is particularly convenient because of the need to use only one *DUMPDATA card when dumping the file on cards. More importantly, it allows for maximum transfer of vectors of data on and off the disk to and from core. Disk to core transfer for a vector of 1000 real variables is 1.2 sec.

3. Model Definition - The regression equation or model is very easy to: (1) define in terms of the original data and (2) define in terms of other variables. In addition the X-variable deletion procedures allow maximum control of model definition.

4. DATAR - DATAR is the format-free input subroutine which retrieves K pieces of data, converts them to floating point, and places them in the vector X when the statement CALL DATAR(K,X) is executed. This routine can be used in any program where format-free read capability is wanted.

5. MXINV - MXINV is the matrix inversion subroutine which has the following properties:

   a. uses the 'bordering' technique of inversion which is useful in a regression environment since variables are added one-at-a-time and a new inverse computed until all variables are in (1x1,2x2,3x3, ...,NxN). If at any stage the inverse becomes singular, the latest variable (which caused the singularity) will be ignored and computations will resume as if that variable never existed. The index of each variable ignored in this manner is printed out.

   b. operates upon an upper-triangular matrix in vector form using the SSP subroutine LOC.

   c. places the inverse on top of the input

matrix, thus minimizing the amount of core storage needed for matrix inversion.

d. Sample inversion times are listed below:

Size of Matrix	Time
10	3 sec
20	20 sec
30	63 sec
40	2 min 27 sec
50	4 min 40 sec
60	7 min 58 sec

Practice Problem

The practice problem shown on the following pages is in four parts -- four separate analyses on a single set of data. The general form of the desired regression equation is shown below:

$$Y = b_0+b_1(W_2/1000)+b_2W_3+b_3(\log W_4)+b_4(W_2/1000)(W_7)$$

where the W_i are the input values for each observation, the b_i are the regression coefficients to be estimated from the data, the b_0 is the constant term to be estimated from the data, and Y is the response variable. (In the practice problem we have two Y-variables, each of which will be used in the above regression equation).

To reduce the above equation to a more familiar form, $Y=b_0+\sum_{i=1}^{4} b_iX_i$, the following substitution is made:

$$X_1 = W_2/1000$$

$$X_2 = W_3$$

$$X_3 = \log (W_4)$$

and $$X_4 = (W_2/1000)W_7$$

A description of the four analyses performed on the set of data is shown below:

Analysis	Description
1	Read header card, X-and Y-definition cards, the data, and the trailer card and perform the indicated regression analysis of Y_1. (Automatic deletion of the least significant variable was done in this analysis.)

2 Read trailer card only and perform an analysis on Y_1 with full equation. Then, perform residuals and predictions analysis on same.

3 Read trailer card only and perform an analysis on Y_2 with full equation.

4 Read trailer card only and delete X_2 and X_4. Then do regression analysis with reduced equation.

DATA for Practice Pr

```
001701 0
03490  1000.
04192
07190                1
01304

06290  3000.
01190
```

Header Card analysis 1

X- and Y- definition cards

```
58.3    7107   21   129   52   3067   07
65.2    6373   22   141   56   2026   01
70.9    6796   22   153   29   2891   70
77.4    9206   20   166   23   2994   31
79.3   14792   25   193   40   3092   69
81.0   14564   23   198   14   3998   39
71.9   11964   20   178   96   3502   61
63.9   13526   23   196   94   3060   19
54.5   12656   20   190   54   3211   22
49.5   14119   20   187   37   3286   92
44.5   16691   22   193   42   3542   72
45.6   14571   19   206   22   3123   45
56.0   13619   22   199   28   3022   66
64.7   14575   22   192    7   2922   36
73.0   14556   21   191   42   3950   25
78.9   15573   21   200   33   4498   74
70.4   15618   22   200   92   3295   13
```

Data

01 Trailer Card - analysis 1

all these cards needed for analysis 1 only

01 Trailer Card - analysis 2

02 Trailer Card - analysis 3

010204 Trailer card - analysis 4

DEFINITION OF VARIABLES

```
X  1 = Z  1 = INPUT  2 /  1000.0001
X  2 = Z  2 = INPUT  3
X  3 = Z  3 =     LOG OF INPUT  4
       Z  4 = INPUT  7
X  4 = Z  5 = Z  1 * Z  4
X  5 = Y  1 = INPUT  6 -  3000.0004
X  6 = Y  2 = INPUT  1
```

AVERAGES, VARIANCES, AND STANDARD DEVIATIONS N= 17

X	AVG	VAR	STD
1	12.9004	12.4382	3.5267
2	21.4705	2.1397	1.4627
3	2.2562	0.0033	0.0575
4	642.4229	208619.0629	456.7484
5	303.7059	199539.3441	446.6983
6	64.8529	182.5224	13.5100

CORRELATION MATRIX

	1	2	3	4	5	6
1	1.000	0.105	0.909	0.656	0.630	-0.024
2	0.105	1.000	0.035	0.068	-0.088	0.437
3	0.909	0.035	1.000	0.564	0.409	-0.065
4	0.656	0.068	0.564	1.000	0.350	-0.224
5	0.630	-0.088	0.409	0.350	1.000	0.285
6	-0.024	0.437	-0.065	-0.224	0.285	1.000

X	RSQR X	B COEF	SE(B)	T	ANALYSIS OF Y 1
1	0.8612	219.8398	60.6129	3.62	
2	0.0329	-68.1118	55.3934	-1.23	
3	0.8327	-8261.5219	3389.6274	-2.44	
4	0.4375	-0.1698	0.2324	-0.73	

CONSTANT	MULT F	DF1	DF2	RSQR	RESIDUAL
17679.42583	4.86	4	12	0.618	101440.98455

DELETE X 4

X	RSQR X	B COEF	SE(B)	T	ANALYSIS OF Y 1
1	0.8325	201.5127	54.1786	3.71	
2	0.0326	-67.4125	54.3434	-1.24	
3	0.8308	-8001.6132	3305.9414	-2.42	

CONSTANT	MULT F	DF1	DF2	RSQR	RESIDUAL
17205.33208	6.54	3	13	0.601	97802.12518

DELETE X 2

X	RSQR X	B COEF	SE(B)	T	ANALYSIS OF Y 1
1	0.8271	189.5931	54.3360	3.48	
3	0.8271	-7397.2177	3332.1694	-2.21	

CONSTANT	MULT F	DF1	DF2	RSQR	RESIDUAL
14548.03127	8.71	2	14	0.554	101566.14060

DELETE X 3

X	RSQR X	B COEF	SE(B)	T	ANALYSIS OF Y 1
1	0.0000	79.8900	25.3772	3.14	

CONSTANT	MULT F	DF1	DF2	RSQR	RESIDUAL
-726.91211	9.91	1	15	0.397	128163.84396

Analysis 2 page 1

X	RSQR X	B COEF	SE(B)	T	ANALYSIS OF Y 1
1	0.8612	219.8398	60.6129	3.62	
2	0.0329	-68.1118	55.3534	-1.23	
3	0.8327	-8261.5215	3385.6274	-2.44	
4	0.4375	-0.1698	0.2324	-0.73	

CONSTANT	MULT F	DF1	DF2	RSQR	RESIDUAL
17679.42583	4.86	4	12	0.618	101440.98455

Analysis 2 page 2

VARIABLES IN MODEL - Y 1

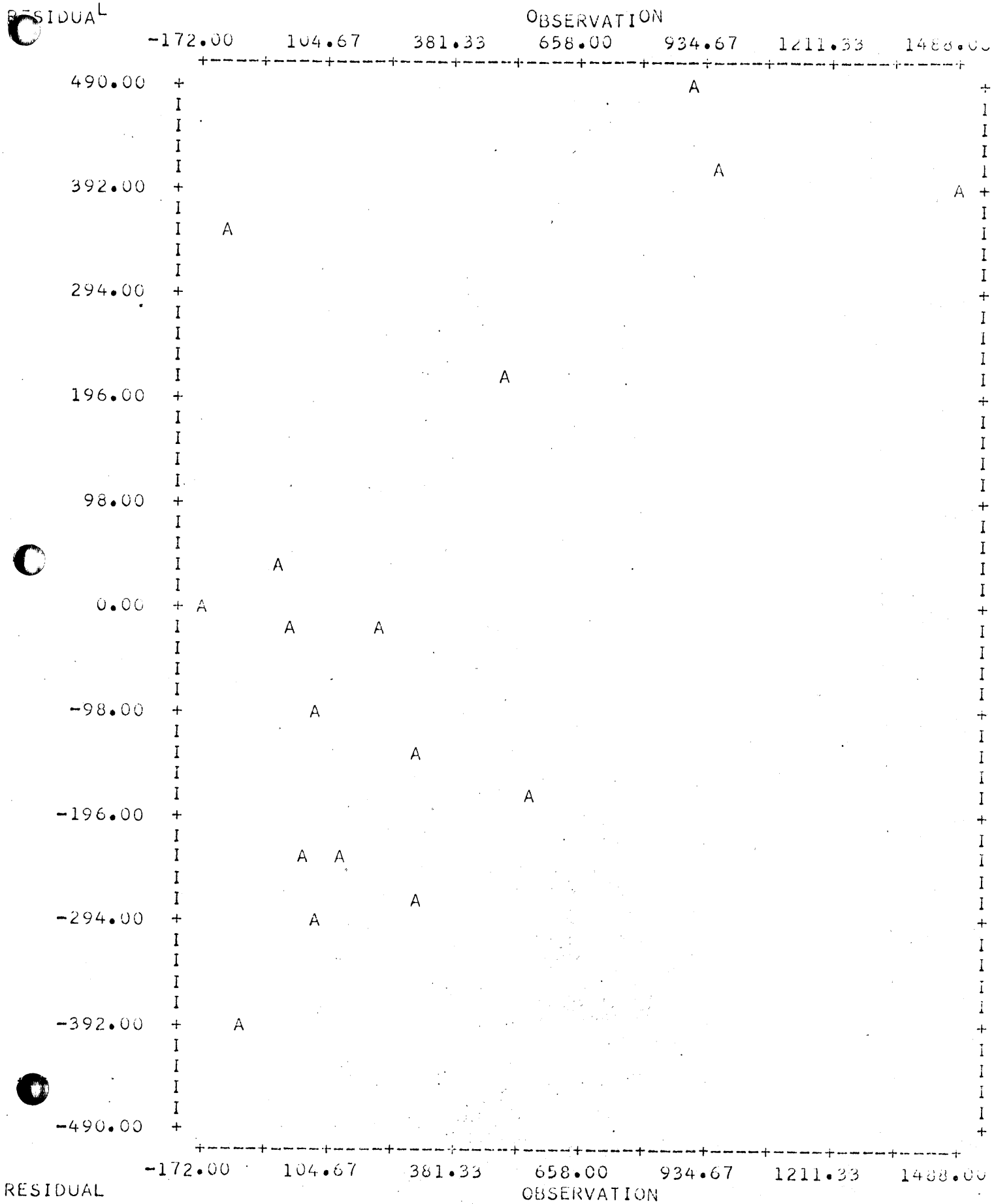
X	B COEF
1	219.8398
2	-68.1118
3	-8261.5215
4	-0.1698

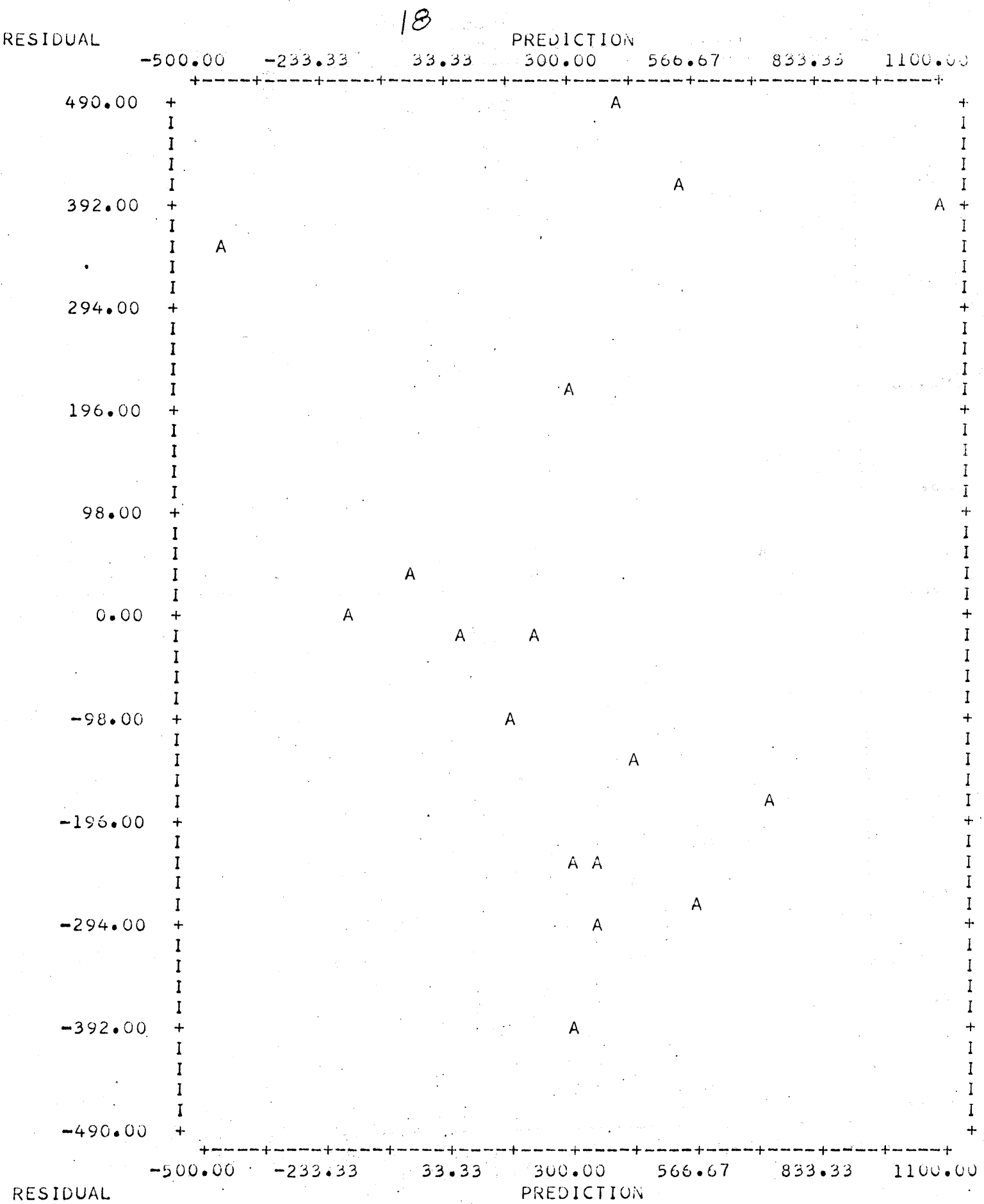
CONSTANT= 17679.42583

RESIDUALS AND PREDICTIONS

OBS	Y(OBS)	Y(PRED)	RESIDUAL	S.E.(Y)	NORM DEV
1	67.0000	366.3594	-299.3594	257.7266	-0.939
2	-172.0000	-174.8906	2.8906	175.8559	0.009
3	-109.0000	-454.6524	345.6524	199.5121	1.085
4	-6.0000	-48.4609	42.4609	144.1040	0.133
5	82.0000	173.0117	-91.0117	217.6426	-0.285
6	898.0001	411.1211	486.8789	127.6075	1.528
7	502.0000	292.5274	209.4726	118.5231	0.657
8	60.0000	293.1484	-233.1484	152.9842	-0.732
9	211.0000	226.2383	-15.2382	152.7211	-0.047
10	286.0000	431.6758	-145.6758	173.3133	-0.457
11	542.0001	727.0860	-185.0859	145.6507	-0.581
12	125.0000	361.1875	-236.1875	187.7821	-0.741
13	22.0000	48.3945	-26.3945	144.9513	-0.082
14	-78.0000	308.7539	-386.7539	142.3867	-1.214
15	950.0001	542.4766	407.5234	133.6047	1.279
16	1488.0002	1088.7775	399.2227	212.0325	1.253
17	295.0000	569.9454	-274.9453	185.0715	-0.863

16


RESIDUAL
OBSERVATION
-172.00
104.67
381.33
658.00
934.67
1211.33
1488.00
490.00
392.00
294.00
196.00
98.00
0.00
-98.00
-196.00
-294.00
-392.00
-490.00
RESIDUAL
OBSERVATION


RESIDUAL
PREDICTION
-500.00
-233.33
33.33
300.00
566.67
833.33
1100.00
490.00
392.00
294.00
196.00
98.00
0.00
-98.00
-196.00
-294.00
-392.00
-490.00

Analysis 3 page 1

X	RSQR X	B COEF	SE(B)	T	ANALYSIS OF Y 2
1	0.8612	1.2734	2.5234	0.50	
2	0.0329	4.0093	2.3644	1.73	
3	0.8327	-40.2371	140.9514	-0.28	
4	0.4375	-0.0111	0.0096	-1.14	-0.111097E-01

CONSTANT	MULT F	DF1	DF2	RSQR	RESIDUAL
60.28457	1.15	4	12	0.277	175.82247

Analysis 4 page 1

X	RSQR X	B COEF	SE(B)	T	ANALYSIS OF Y 1
1	0.8271	189.5935	54.3360	3.48	
3	0.8271	-7397.2422	3332.1694	-2.21	

CONSTANT	MULT F	DF1	DF2	RSQR	RESIDUAL
14548.08010	8.71	2	14	0.554	101565.98453

wed A-6 by Marshall ~~Hecht~~ Hechter for Program Chairman

ACCUMULATOR OPERATIONS

Operations		No X R	X R
LD LDD STO STD	SHORT	The operand may be either a number or a label representing a core address within 128 words of this instruction. This is the address of the data to be operated on.	The operand may be either a number within the limits ± 128 or a label representing a core address within the first 128 positions of core. The sum of this number or address and the contents of the specified index register is the address of the data to be operated on.
A AD S SD	LONG	The operand may be either a number or label representing an address anywhere in core. This is the address of the data to be operated on.	The operand may be either a number or a label representing an address anywhere in core. Add to this address the contents of the specified index register. The sum is the address of the data to be operated on.
M D AND OR EOR LDS* STS**	INDIRECT	The operand may be either a number or a label representing an address anywhere in core. This core location contains another address. This second address is the address of the data to be operated on.	The operand may be either a number or a label representing an address anywhere in core. Add to this address, the contents of the specified index register. The sum is the address of a core location which contains the address of the data to be operated on.
	X	The operand may be either a number within the limits ± 128 or a label representing a core address within the first 128 positions of core. The sum of this number or address and the address of the next instruction is the address of the data to be operated on.	Same as SHORT X R.

* Short format only.

** As a short instruction, store status stores the status of the carry and overflow indicators in the two low order bits of the word being operated on. As a long instruction store status writes or clears storage protect bits.

LOAD INDEX REGISTER OPERATIONS

Operation		No X R	X R
LDX	SHORT	The operand may be either a number within the limits ± 128 or a label representing an address within the first 128 positions of core. This address or number is the address of the next instruction that will be executed.	The operand may be either a number within the limits ± 128 or a label representing an address within the first 128 positions of core. The specified index register is loaded with this number or address.
	LONG	The operand may be either a number or a label representing an address anywhere in core. This address or number is the address of the next instruction that will be executed.	The operand may be either a number or a label representing an address anywhere in core. The specified index register is loaded with this number or address.
	INDIRECT	The operand may be either a number or a label representing an address anywhere in core. The contents of this address is the address of the next instruction that will be executed.	The operand may be either a number or a label representing an address anywhere in core. The specified index register is loaded with the contents of this address.
	X	Same as SHORT No X R.	Same as SHORT X R.

STORE INDEX REGISTER OPERATIONS

Operations		No X R	X R
S T X	SHORT	The operand may be either a number or a label representing a core address within 128 words of this instruction. The address of the next instruction is stored at this address.	The operand may be either a number or a label representing a core address within 128 words of this instruction. The specified index register is stored at this address.
	LONG	The operand may be either a number or a label representing a core address anywhere in core. The address of the next instruction is stored at this address.	The operand may be either a number or a label representing a core address anywhere in core. The specified index register is stored at this address.
	INDIRECT	The operand may be either a number or a label representing a core address anywhere in core. This core location contains an address of another core location. The address of the next instruction is stored at this second address.	Same as no X R but the contents of the specified index register is stored instead of the IAR. The operand may be either a number or a label representing a core address anywhere in core. This core location contains the address of another core location. The specified index register is stored at this second core location.
	X	The operand may be a number within the limits ± 128 or a label representing a core address within the first 128 positions of core. The sum of this number or address and the address of the next instruction is the address of a core location where the address of the next instruction will be stored.	Same as SHORT X R.

MODIFY INDEX REGISTER OPERATIONS

Operation		No X R	X R
M D X	SHORT	The operand may be either a number or a label representing a core address within 128 words of this instruction. This number or address is the address of the next instruction that will be executed. (Unconditional branch)	The operand may be either a number within the limits ± 128 or a label representing a core address within the first 128 positions of core. The specified index register is modified by this number or address. *
	LONG	Requires two operands. The first operand may be either a number or a label representing a core address anywhere in core. The second operand must be a number within the limits ± 128. The second operand is added to the data in the core address specified by the first operand. This is the only add to core operation in the instruction set. *	The operand may be either a number or a label representing a core address anywhere in core. The specified index register is modified by this number or address.
	INDIRECT	Do not use.	The operand may be either a number or a label representing a core address anywhere in core. The specified index register is modified by the contents of this core address. *
	X	The operand may be either a number within the limits ± 128 or a label representing a core address within the first 128 positions of core. The sum of this number or address and the address of the next instruction is the address of the next instruction that will be executed. (Unconditional branch)	Same as SHORT X R.

* If the contents of the core address or index register specified goes to zero or changes sign as a result of the modification, you skip one word in core.

BRANCH OR SKIP OPERATIONS

(Checking Accumulator Condition)

Operation		No X R	X R
B S C	SHORT	Requires only one operand consisting of the condition or conditions being tested. Any combination of the symbols C,O,Z,-,+ or E (not separated by commas) may be used.	C,O If a carry or an overflow condition is present you execute the next instruction; if not, you skip one core word. Z,-,+,E If any of the conditions specified in the operand are present you skip one word. If none of the conditions specified are present, you execute the next instruction.	Do not use.
	LONG	One Operand If only one operand is given, the instruction is an unconditional branch to a location anywhere in core.	The operand may be either a number or a label representing an address anywhere in core. The branch is to the address specified by the number or the label.	The operand may be either a number of a label representing an address anywhere in core. The branch is to an address equal to the sum of the number or address and the contents of the specified index register.
		Two Operands When two operands are given, the instruction is a conditional branch. The second operand specifies the condition or conditions to be tested. The first operand may be a number or label and specifies the branch address.	C,O If a carry or an overflow condition is present, branch to the address specified by the first operand. If not, execute the next instruction. Z,-,+,E If any of the conditions specified in the second operand are present, execute the next instruction. If none of the conditions specified are present, you branch to the address specified by the first operand.	Same as with No X R except the branch address is equal to the sum of the address specified by the first operand and the contents of the specified index register.

BRANCH OR SKIP OPERATIONS

(Checking Accumulator Condition)

Operation			No X R	X R
B S C	INDIRECT	One Operand If only one operand is given, the instruction is an unconditional branch to a location anywhere in core.	The operand may be either a number or a label representing an address anywhere in core. This address contains a second core address. This second address is the branch address.	The operand may be either a number or a label representing an address anywhere in core. The sum of the address specified by the operand and the contents of the specified index register is a second core address. The contents of this second address is the branch address.
		Two Operands When two operands are given, the instruction is a conditional branch. The second operand specifies the condition or conditions to be tested. The first operand may be a number or label representing an address anywhere in core.	C,O If a carry or overflow condition is present, branch. The branch address is the contents of the address specified by the first operand. If not, execute the next instruction. Z,-,+,E If any of the conditions specified in the second operand are present, execute the next instruction. If none of the conditions specified are present, branch. The branch address is the contents of the address specified by the first operand.	Same as No X R except for the branch address. The sum of the address specified by the first operand and the contents of the specified index register is a second core address. The contents of this second address is the branch address.
	X		Same as SHORT NO X R.	Same as SHORT X R.

BRANCH AND STORE OPERATIONS

(INSTRUCTION ADDRESS REGISTER)

Operation			No X R	X R
B S I	SHORT	Requires only one operand. The short instruction is an unconditional branch.	The operand may be either a number or a label representing a core address within 128 words of this instruction. The address of the next instruction is stored at this address. The next instruction that is executed is at the word following this address.	The operand may be either a number within the limits ± 128 or a label representing a core address within the first 128 positions of core. The sum of this address and the contents of the specified index register is a second core address. The address of the next instruction is stored at this second address. The next instruction that is executed is at the word following this address.
	LONG	One Operand If only one operand is given, the instruction is an unconditional branch to a location anywhere in core.	The operand may be either a number or a label representing a core address anywhere in core. The address of the next instruction is stored at this address. The next instruction that is executed is at the word following this address.	The operand may be either a number or a label representing a core address anywhere in core. The sum of this address and the contents of the specified index register is a second core address. The address of the instruction is stored at this second address. The next instruction that is executed is at the word following this address.
		Two Operands When two operands are given, the instruction is a conditional branch. The second operand specifies the condition or conditions to be tested. The first operand may be a number or label representing an address anywhere in core (branch address).	C,O If a carry or an overflow condition is present, the address of the next instruction is stored at the address specified by the first operand. The next instruction that is executed is at the word following this address. If no carry or overflow, the next instruction is executed.	Same as No XR except for the branch address. If the branch is executed, the sum of address specified by the first operand and the contents of the specified index register is a second core address. The address of the next instruction is stored at this second address. The next instruction that is executed is at the word following this second address.

BRANCH AND STORE OPERATIONS

(INSTRUCTION ADDRESS REGISTER)

Operation		No X R	X R
B S I	LONG	Z,-,+,E If any of the conditions specified in the second operand are present, the next instruction is executed. If none of the conditions specified are present, the address of the next instruction is stored at the address specified by the first operand. The next instruction that is executed is at the word following this address.	
	INDIRECT	One Operand If only one operand is given, the instruction is an unconditional branch to a location anywhere in core.	The operand may be either a number or a label representing an address anywhere in core. This address contains a second address. The address of the next instruction is stored at this second address. The next instruction that is executed is at the word following this second address.	The operand may be either a number or a label representing an address anywhere in core. The sum of the address specified by the operand and the contents of the specified index register is a second core address. This second core address contains a third address. The address of the next instruction is stored at this third address. The next instruction that is executed is at the word following this third address.
		Two Operands When two operands are given, the instruction is a conditional branch. The second operand specifies the condition or conditions to be tested. The first operand may be a number or label representing an address anywhere in core.	C,O If a carry or overflow condition is present, the branch is executed. The address specified by the first operand contains a second address. The address of the next instruction is stored at this second address. The next instruction executed is at the word following this second address. If no carry or overflow, the next instruction is executed.	Same as No X R except for the branch address. The sum of the address specified by the first operand and the contents of the specified index register is a second core address. This second core address contains a third address. The address of the next instruction is stored at this third address.

BRANCH AND STORE OPERATIONS

(INSTRUCTION ADDRESS REGISTER)

Operation		No X R	X R
B S I	INDIRECT	Z,-,+,E if any of the conditions specified in the second operand are present, the next instruction is executed. If none of the conditions specified are present, the branch is executed. The address specified by the first operand contains a second address. The address of the next instruction is stored at this second address. The next instruction that is executed is at the word following this second address.	The next instruction that is executed is at the word following this third address.
	X	Requires only one operand. The instruction is an unconditional branch.	The operand may be either a number within the limits ± 128 or a label representing a core address within the first 128 positions of core. The sum of this number or address and the address of the next instruction is a second address. The address of the next instruction is stored at this second address. The next instruction that is executed is at the word following this second address.	Same as SHORT X R

IBM

IBM 1130 Assembler
Coding Form

Form X26-5994-0
Printed in U.S.A.

Program DISK ASSEMBLER INSTRUCTIONS Date

Programmed by Page No. of

Label (21–25)	Operation (27–30)	F (32)	T (33)	Operands & Remarks (35–70)	Identification (75–80)
	SKP			+ SKIP IF ACC IS POSITIVE	
	SKP			+- SKIP IF ACC IS POSITIVE OR NEGATIVE	
	B			LOOP SHORT UNCONDITIONAL BRANCH	
	B	L		AGAIN UNCONDITIONAL BRCH ANYWHERE	
	BZ			ZERO BRCH IF ACC IS ZERO	
	BN			NEG BRCH IF ACC IS NEGATIVE	
	BP			POS BRCH IF ACC IS POSITIVE	
	BC			CARRY BRCH IF CARRY INDICATOR IS ON	
	BO			OVERF BRCH IF OVERFLOW INDICATOR IS ON	
	BOD			ODD BRCH IF ACC IS ODD	
	MDM			CORE,100 ADD 100 TO LOCATION CORE	
	XCH			EXCHANGE ACC AND EXT	

4.13.12

SHIFT INSTRUCTIONS

Short Form Only	ND XR	XR
	The operand is a number that becomes the shift count. If a shift count of Zero is encountered, the instruction is treated as a NOP. The rightmost 6 bits of the displacement are used to control the length of the shift.	The XR tagged provides the shift count. The rightmost 6 bits of the tagged register controls the length of the shift.
SLA	The Accumulator is shifted left the number of bit positions indicated by either the displacement (Tag = 00) or an index register. Low order (rightmost) bits of the accumulator are set to zero.	
SRA	The Accumulator is shifted right the number of bit positions indicated by either the displacement (tag = 00) or an index register. High order (leftmost) bits are filled with the value of the sign bit (bit position 0).	
SLT	The Accumulator and extension are shifted left as one 32 bit register (See SLA).	
SRT	The Accumulator and extension are shifted right as one 32 bit register . See SRA.	

SHIFT INSTRUCTIONS

Short Form Only	No XR	XR
SLCA	Tag bits of 00 cause this instruction to be executed as a SLA.	The Accumulator is shifted to the left, the number of positions indicated by the tagged Index register. The six low order (rightmost) bits of the indicated register are used to determine the shift count. This count is decremented by 1 for each bit position shifted.
SLC	Tag bits of 00 cause this instruction to be executed as a SLT.	The Accumulator and extension are shifted left as one 32 bit register. See SLCA instruction.
RTE	The shift count is in the operand field. The low order bits of the extension are shifted into the high order bits of the accumulator. Consider the accumulator and extension to be a continuous 32-bit register forming a loop.	The rotate count is found in the indicated index register.
XCH	Exchanges the contents of the Accumulator and extension. NO OPERAND is specified. This instruction acts as a RTE 16 instruction.	

XIO OPERATIONS

Operations	No X R	XR
SHORT	The operand may be either a number or a label representing a core address within 128 words of this instruction. This is the address of the first word of the IOCC. *	The operand may be either a number within the limits ± 128 or a label representing a core address within the first 128 positions of core. The sum of this number or address and the contents of the specified index register is the address of the first word of the IOCC.*
LONG	The operand may be either a number or label representing an address anywhere in core. This is the address of the first word of the IOCC.*	The operand may be either a number or a label representing an address anywhere in core. Add to this address the contents of the specified index register. The sum is the address of the first word of the IOCC.*
INDIRECT	The operand may be either a number or a label representing an address anywhere in core. This core location contains another address. This second address is the address of the first word of the IOCC.*	The operand may be either a number or a label representing an address anywhere in core. Add to this address, the contents of the specified index register. The sum is the address of a core location which contains the address of the first word of the IOCC.*
X	The operand may be either a number within the limits ± 128 or a label representing a core address within the first 128 positions of core. The sum of this number or address and the address of the next instruction is the address of the first word of the IOCC.*	Same as SHORT X R.

* The IOCC refers to the Input Output Control Command. The first word of the IOCC must be at an even boundary.

```
// *I/O INTERRUPT HANDLING...CARD READER   M.K. HECHTLR
SAV12   DC      0
        DC      6
CON1    DC      /8181
        DC      /9802
        DC      /72C2
        BSS     3
QTY1    DC      0
SAVE2   DC      0
        BSS  E  0
START   DC      /0000
        DC      /1404
READ    DC      CRDIN
        DC      /1200
CRDIN   BSS     80
SENS0   DC      /0000
        DC      /1701
SENS4   DC      /0000
        DC      /1702
AD0     DC      LEV0
AD4     DC      LEV4
ADCRD   DC      CRDIN
LOAD    LD   L  12
        STO  L  SAV12
        LDX  L1 LEV0
        STX  L1 8
        LDX  L1 LEV4
        STX  L1 12
BACK    XIO     START
        MDX     *-1
LCTST   LD      CRDIN
        BSC  L  EOJ,&-
        LIBF    DCBIN
        DC      CRDIN
        STO     QTY1
        LIBF    DCBIN
        DC      CRDIN&6
        A       QTY1
        LIBF    BINDC
        DC      CRDIN&12
        LIBF    HOLPR
        DC      /0000
        DC      CRDIN&12
        DC      CON1&3
        DC      6
        LD   L  SAV12
        STO  L  12
12      LIBF    WRTY0
11      DC      /2000
10      DC      CON1-1
9
8
7
6
5
4
3
2
```

```
TEST    LIBF      WRTYO
        DC        /0000
        MDX       TEST
        LDX   L1  LEV4
        STX   L1  12
        BSC   L   BACK
LEV0    DC        *-*
        STX   L2  SAVE2
        XIO       SENS0
        XIO   L   READ
        MDX   L   READ,&1
        LDX   I2  SAVE2
        BOSC  I   LEV0
LEV4    DC        *-*
        XIO       SENS4
        LD        ADCRD
        STO   L   READ
        BOSC  L   LCTST
EOJ     EXIT
        END       LOAD
```

HEXADECIMAL TO DECIMAL CONVERSION GUIDE

DISPLACEMENT VALUES

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	-128	-127	-126	-125	-124	-123	-122	-121	-120	-119	-118	-117	-116	-115	-114	-113
9	-112	-111	-110	-109	-108	-107	-106	-105	-104	-103	-102	-101	-100	-99	-98	-97
A	-96	-95	-94	-93	-92	-91	-90	-89	-88	-87	-86	-85	-84	-83	-82	-81
B	-80	-79	-78	-77	-76	-75	-74	-73	-72	-71	-70	-69	-68	-67	-66	-65
C	-64	-63	-62	-61	-60	-59	-58	-57	-56	-55	-54	-53	-52	-51	-50	-49
D	-48	-47	-46	-45	-44	-43	-42	-41	-40	-39	-38	-37	-36	-35	-34	-33
E	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
F	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Vol 1 - 6

By Marshall Hecht for Program Assistance

I B M 1130 OPERATION CODE TABLE
MACHINE CODE SEQUENCE

08	XIO	
09	XIO	1
0A	XIO	2
0B	XIO	3
0C	XIO	L
0D	XIO	L1
0E	XIO	L2
0F	XIO	L3
100	SLA	
107	SLCA	
108	SLT	
10C	SLC	
110	SLA	1
117	SLCA	1
118	SLT	1
11C	SLC	1
120	SLA	2
127	SLCA	2
128	SLT	2
12C	SLC	2
130	SLA	3
137	SLCA	3
138	SLT	3
13C	SLC	3
140	SLA	L
147	SLCA	L
148	SLT	L
14C	SLC	L
150	SLA	L1
157	SLCA	L1
158	SLT	L1
15C	SLC	L1
160	SLA	L2
167	SLCA	L2
168	SLT	L2
16C	SLC	L2
170	SLA	L3
177	SLCA	L3
178	SLT	L3
17C	SLC	L3
180	SRA	
188	SRT	
18C	RTE	
18D	XCH	
190	SRA	1
198	SRT	1
19C	RTE	1
1A0	SRA	2
1A8	SRT	2
1AC	RTE	2
1B0	SRA	3
1B8	SRT	3
1BC	RTE	3
1C0	SRA	L
1C8	SRT	L
1CC	RTE	L

1D0	SRA	L1
1D8	SRT	L1
1DC	RTE	L1
1E0	SRA	L2
1E8	SRT	L2
1EC	RTE	L2
1F0	SRA	L3
1F8	SRT	L3
1FC	RTE	L3
20	LDS	
28	STS	
29	STS	1
2A	STS	2
2B	STS	3
2C	STS	L
2D	STS	L1
2E	STS	L2
2F	STS	L3
30	WAIT	

CONDITION CODES

40	BSI		08=+	
41	BSI	1	09=+O	
42	BSI	2	0A=+C	
43	BSI	3	0B=+CO	
44	BSI	L	0C=+E	
45	BSI	L1	0D=+EO	
46	BSI	L2	0E=+EC	
47	BSI	L3	0F=+ECO	
48	BSC		00=NOP	1X=-
49	BSC	1	01=O	2X=Z
4A	BSC	2	02=C	3X=Z
4B	BSC	3	03=CO	
4C	BSC	L	04=E	
4D	BSC	L1	05=EO	
4E	BSC	L2	06=EC	
4F	BSC	L3	07=ECO	

60	LDX	
61	LDX	1
62	LDX	2
63	LDX	3
64	LDX	L
65	LDX	L1
66	LDX	L2
67	LDX	L3
68	STX	
69	STX	1
6A	STX	2
6B	STX	3
6C	STX	L
6D	STX	L1
6E	STX	L2
6F	STX	L3
70	MDX	
71	MDX	1

IBM 1130 OPERATION CODE TABLE
MACHINE CODE SEQUENCE

72	MDX	2	C1	LD	1
73	MDX	3	C2	LD	2
74	MDX	L	C3	LD	3
75	MDX	L1	C4	LD	L
76	MDX	L2	C5	LD	L1
77	MDX	L3	C6	LD	L2
80	A		C7	LD	L3
81	A	1	C8	LDD	
82	A	2	C9	LDD	1
83	A	3	CA	LDD	2
84	A	L	CB	LDD	3
85	A	L1	CC	LDD	L
86	A	L2	CD	LDD	L1
87	A	L3	CE	LDD	L2
88	AD		CF	LDD	L3
89	AD	1	DO	STO	
8A	AD	2	D1	STO	1
8B	AD	3	D2	STO	2
8C	AD	L	D3	STO	3
8D	AD	L1	D4	STO	L
8E	AD	L2	D5	STO	L1
8F	AD	L3	D6	STO	L2
90	S		D7	STO	L3
91	S	1	D8	STD	
92	S	2	D9	STD	1
93	S	3	DA	STD	2
94	S	L	DB	STD	3
95	S	L1	DC	STD	L
96	S	L2	DD	STD	L1
97	S	L3	DE	STD	L2'
98	SD		DF	STD	L3
99	SD	1	E0	AND	
9A	SD	2	E1	AND	1
9B	SD	3	E2	AND	2
9C	SD	L	E3	AND	3
9D	SD	L1	E4	AND	L
9E	SD	L2	E5	AND	L1
9F	SD	L3	E6	AND	L2
A0	M		E7	AND	L3
A1	M	1	E8	OR	
A2	M	2	E9	OR	1
A3	M	3	EA	OR	2

A4	M	L
A5	M	L1
A6	M	L2
A7	M	L3
A8	D	
A9	D	1
AA	D	2
AB	D	3
AC	D	L
AD	D	L1
AE	D	L2
AF	D	L3
CO	LD	

EB	OR	3
EC	OR	L
ED	OR	L1
EE	OR	L2
EF	OR	L3
FO	EOR	
F1	EOR	1
F2	EOR	2
F3	EOR	3
F4	EOR	L
F5	EOR	L1
F6	EOR	L2
F7	EOR	L3

IBM 1130 OPERATION CODE TABLE

SYMBOLIC SEQUENCE

80	A	
81	A	1
82	A	2
83	A	3
84	A	L
85	A	L1
86	A	L2
87	A	L3
88	AD	
89	AD	1
8A	AD	2
8B	AD	3
8C	AD	L
8D	AD	L1
8E	AD	L2
8F	AD	L3
EO	AND	
Ei	AND	1
E2	AND	2
E3	AND	3
E4	AND	L
E5	AND	L1
E6	AND	L2
E7	AND	L3

CONDITION CODES

48	BSC		QQ=NOP	1X=-
49	BSC	1	01=0	2X=Z
4A	BSC	2	02=C	3X=Z
4B	BSC	3	03=CO	
4C	BSC	L	04=E	
4D	BSC	L1	05=EO	
4E	BSC	L2	06=EC	
4F	BSC	L3	07=ECO	
40	BSI		08=+	
41	BSI	1	09=+0	
42	BSI	2	0A=+C	
43	BSI	3	0B=+CO	
44	BSI	L	CC=+E	
45	BSI	L1	OD=+EO	
46	BSI	L2	OE=+EC	
47	BSI	L3	OF=+ECO	

A8	D	
A9	D	1
AA	D	2
AB	D	3
AC	D	L
AD	D	L1

F5	ECR	L1
F6	EOR	L2
F7	EOR	L3
CO	LD	
C1	LD	1
C2	LD	2
C3	LD	3
C4	LD	L
C5	LD	L1
C6	LD	L2
C7	LD	L3
C8	LDD	
C9	LDD	1
CA	LDD	2
CB	LDD	3
CC	LDD	L
CD	LDD	LI
CE	LDD	L2
CF	LDD	L3
20	LDS	
60	LDX	
61	LDX	1
62	LDX	2
63	LDX	3
64	LDX	L
65	LDX	L1
66	LDX	L2
67	LDX	L3
AO	M	
A1	M	1
A2	M	2
A3	M	3
A4	M	L
A5	M	L1
A6	M	L2
A7	M	L3
70	MDX	
71	MDX	1
72	MDX	2
73	MDX	3
74	MDX	L
75	MDX	L1
76	MDX	L2
77	MDX	L3
E8	OR	
E9	OR	1
EA	OR	2
EB	OR	3

90	S	
91	S	1
92	S	2
93	S	3
94	S	L
95	S	L1
96	S	L2
97	S	L3
98	SD	
99	SD	1
9A	SD	2
9B	SD	3
9C	SD	L
9D	SD	L1
9E	SD	L2
9F	SD	L3
100	SLA	
110	SLA	1
120	SLA	2
130	SLA	3
140	SLA	L
150	SLA	L1
160	SLA	L2
170	SLA	L3
107	SLCA	
117	SLCA	1
127	SLCA	2
137	SLCA	3
147	SLCA	L
157	SLCA	L1
167	SLCA	L2
177	SLCA	L3
10C	SLC	
11C	SLC	1
12C	SLC	2
13C	SLC	3
14C	SLC	L
15C	SLC	L1
16C	SLC	L2
17C	SLC	L3
108	SLT	
118	SLT	11
128	SLT	2
138	SLT	3
148	SLT	L
158	SLT	L1
168	SLT	L2
178	SLT	L3
180	SRA	
190	SRA	1
1A0	SRA	2
1B0	SRA	3
1D0	SRA	L1
1E0	SRA	L2
1F0	SRA	L3
188	SRT	
198	SRT	1
1A8	SRT	2
1B8	SRT	3
1C8	SRT	L
1D8	SRT	L1
1E8	SRT	L2
1F8	SRT	L3
D8	STD	
D9	STD	1
DA	STD	2
DB	STD	3
DC	STD	L
DD	STD	L1
DE	STD	L2
DF	STD	L3
D0	STD	
D1	STD	1
D2	STD	2
D3	STD	3
D4	STD	L
D5	STD	L1
D6	STD	L2
D7	STD	L3
28	STS	
29	STS	1
2A	STS	2
2B	STS	3
2C	STS	L
2D	STS	L1
2E	STS	L2
2F	STS	L3
68	STX	
69	STX	1
6A	STX	2
6B	STX	3
6C	STX	L
6D	STX	L1
6E	STX	L2
6F	STX	L3
30	WAIT	
18D	XCH	
08	X10	
09	X10	1
0A	X10	2
0B	X10	3
0C	X10	L
0D	X10	L1
0E	X10	L2
0F	X10	L3

AE	D	L2
AF	D	L3
F0	EOR	
F1	EOR	1
F2	EOR	2
F3	EOR	3
F4	EOR	L
1DC	RTE	L1
1EC	RTE	L2
1FC	RTE	L3

EC	OR	L
ED	OR	L1
EE	OR	L2
EF	OR	L3
18C	RTE	
19C	RTE	1
1AC	RTE	2
1BC	RTE	3
1CC	RTE	L
1CO	SRA	L

SESSION REPORT

COMMON - Chicago

Session Number WED A8 Session Name 1620 Project

Chairman H. B. Kerr

Time 8:30 to 10:00 Attendance (No.) 40

Speakers Pat Emin

Scott Butler

Synopsis of Meeting Attendance dropped off since it was the last day. Papers were good with a small amount of discussion. A business session was held following the last paper. Present Co-Chairman resigned. New Chairman was elected. Plans were made for Philadelphia.

COMMITTEE: 1620 PROJECT

SUBJECT: Running FORTRAN II - D
In background mode with
spooling and check-point

SPEAKER: PATRICK P. EMIN
University of New Brunswick
Fredericton, New Brunswick
Phone: 506 - 475-9471

TIME: Wednesday, April 10, 1968
9:15 A. M.
Session A8

NO. OF PAGES(TEXT): 23

RUNNING FORTRAN II-D IN BACKGROUND MODE

WITH

SPOOLING AND CHECK-POINT

Prepared for presentation at
the COMMON Meeting, Chicago, Illinois,
April 8 - 10, 1968

by

Patrick P. Emin

UNIVERSITY OF NEW BRUNSWICK

COMPUTING CENTRE

Fredericton, N. B.

April 4, 1968

ABSTRACT

A description is given of modifications made to the MONITOR II system and some additional programming which permits the formation of a disk job queue, made up of Fortran II-D programs.

By the use of a sub-monitor, job-to-job transition can be controlled and switched alternately between card reader and disk. Provision is also made for spooling of I/O on disk, deferred printing or punching, and check-pointing of executing programs.

CONTENTS

RUNNING FORTRAN II-D IN BACKGROUND MODE

WITH SPOOLING AND CHECK-POINT

1. GENERAL

1.1 In a busy 1620 operating environment, programs with running times exceeding a few minutes usually cannot be completed during batch-processing express runs with heavy throughput. As a result, time-consuming programs are often returned to the user unfinished and usually with accompanying instructions concerning personal scheduling of additional time.

1.2 Consequently, a method has been devised by which, with the addition of a new control card, and one or two simple CALL satatements, FORTRAN II-D programs may be arranged to perform the following.

1.2.1 Executing programs may be automatically or manually interrupted, and automatically placed in a queue (waiting list) on 1311 magnetic disk storage for future resumption by a sub-monitor.

1.2.2 Programs may suppress and postpone punched card and/or printed line output, and store such output on 1311 magnetic disk. This permits the 1443 line printer and the 1622 card punch to be shut down and left unattended for the duration of a program e.g., overnight, and on completion, all output may be obtained in one continuous sequence

1.2.3 Card input data associated with a program may be read "in toto", in one continuous sequence at the beginning of the program. The card images are stored on 1311 magnetic disk, and thereafter may be read as required, either free-style or by FORMAT, each card as often as desired, and in any order. Thus, the 1622 card reader may also be shut down during program execution.

1.2.4 Any combination of 1.2.1, 1.2.2 and 1.2.3 is possible. All three together will, for our purposes, be referred to as "spooling".

1.2.5 Long running programs may be check-pointed by the programmer to guard against the possibility of machine or other malfunction. The status of the program as at check-point time is preserved on 1311 magnetic disk.

2. BACKGROUND

2.1 By including the FORTRAN II-D control card "*BACKGROUND" at the head of a source deck, the 1620 operator is provided with the facility for interrupting an executing program and having it stored automatically in one of the permanent storage areas on 1311 magnetic disk (6 at present). This is accomplished by turning sense switch 4 ON, consequently, like sense switch 1, this switch should not be used by the programmer. Use of the control card *BACKGROUND increases the length of the users object program by 8-13 percent, and causes loading of library subroutine number 22 at run time (length = 112 cores).

2.2 Subroutine DSTASH -- When the FORTRAN statement "CALL DSTASH" is encountered in a program the program is automatically interrupted and placed in one of the permanent storage areas on disk. If no permanent space is available, provision is made for the operator to resume the program or to store it in a temporary (pot-luck) area. It is NOT necessary to include the *BACKGROUND control card, but as a result, the program may be stored and resumed ONCE only, unless the statement CALL DSTASH can be reached from within any program loop when sense switch 4 is found to be ON. DSTASH uses library subroutine number 22 but itself takes up only 42 memory locations, therefore it is usually more economical than *BACKGROUND in respect to memory space allocation.

2.3 Program DSAVE[1]. This program is called into play by the sub-monitor via library subroutine 22, with *BACKGROUND whevever sense switch 4 is ON, or, by subroutines DSTASH or SPOOL when called by the programmer. It requires no memory space allocation by the programmer, since it overlays the contents of memory when in use. It provides the main facility for interrupting, storing and resuming programs in BACKGROUND, maintains the sub-monitor switches, indicators and records, and generally can be considered as the heart of the sub-monitor.

1. No. 7.0.078 in the 1620 program library.

DSAVE can also be activated manually by the 1620. operator through the use of library subroutine 19 (LDSAVE). However, its use in this manner is undesirable except for temporary program interruptions, since the stored program is not considered to be in proper BACKGROUND, and cannot participate fully in the Disk Job Queue (DISQUE) or Overnight Production Line (OPL).

2.4 SUBMON -- When the UNB Operating System disk packs are mounted, the 1620 is under control of the IBM MONITOR II system and the UNB sub-monitor (SUBMON). SUBMON is a sub-supervisor within the MONITOR II Supervisor which controls job-to-job transition between the card reader stack and the DISQUE. When no jobs are pending in the card reader, program DSAVE is called in to begin processing of programs stored in BACKGROUND. DSAVE extends the supervisory action of SUBMON by co-ordinating the functions of *BACKGROUND, DSTASH, DPRINT, MISER and SPOOL via a communications record on 1311 disk known as DATERA, which retains a running account of disk space allocation, SUBMON status, mode of operation and job-accounting information (current job number, date and user name).

3. Subroutine DPRINT

3.1 When the statement "CALL DPRINT" is executed in a FORTRAN II-D program, the normal operation of the FORTRAN arithmetic and input/output routines is modified in order to effect the following:

3.1.1 When sense switch 1 is ON (the normal setting) all print/punch output remains normal, being channeled to the appropriate output device.

3.1.2 If sense switch 1 is found to be OFF all print/punch output is channeled to the 1311 magnetic disk where it is stored in consecutive records beginning with the first available disk sector as indicated in DATERA.

3.1.3 When sense switch 1 is turned ON again, the next PRINT or PUNCH statement causes stored output(if any) to be channeled to the appropriate output devices in one continuous sequence up to and including the current record (s).

3.1.4 When "CALL EXIT" is executed, the 1620 will be interlocked with the 1443 printer or 1622 card punch in order to output all stored records UNLESS the program originated from the DISQUE, in which case it will return to the waiting list in BACKGROUND until the queue has completed one cycle.

3.2 Once DPRINT has been activated, all FORTRAN execution error messages go to the console typewriter.

This subroutine is most effective when used during 1443 printer repair and maintenance periods, and in the BACKGROUND mode for overnight scheduling. Subroutine DPRINT occupies 14 memory cores and requires the use of library subroutine 20 (length = 1930 memory cores).

3.3 DPRINT in BACKGROUND. A program which uses DPRINT and also has included an *BACKGROUND control card may be placed in the DISQUE at any time by the 1620 operator, or by the programmer with CALL DSTASH. The program will then be ready to participate in the Overnight Production Line. Such programs must run in the normal mode (sense switch 1 ON) prior to admission into BACKGROUND, or, at least have no output already stored on disk.

4. Overnight Production Line (OPL)

4.1 It is quite possible to submit a FORTRAN II-D program for diagnostic testing in a batch-processing express run, already prepared for overnight scheduling. However, the programmer must ensure that his/her program is already thoroughly tested and debugged, or at least will be tested during the limited diagnostic run it will be given in the batch. This is necessary, of course, to obviate CHECK-STOPS or other "hang-ups" which may occur during overnight production. (A program trace and interrupt routine is available from the author on request).

4.2 It requires three separate passes for the sub-monitor to complete a program in the OPL.

4.2.1 Pass 1 - the Assembly phase. The program is compiled, assembled and executed in the normal mode. There may be initial output for testing and monitoring purposes. CALL DPRINT is reached early in the program; during batch-processing runs this will occur within the limited time permitted. The 1620 operator will turn sense switch 4 ON to interrupt the program and transfer it to BACKGROUND, or, the programmer may use CALL DSTASH, whichever occurs first.

4.2.2 Pass 2 - the Production phase. The sub-monitor begins selecting jobs from the DISQUE when it senses the last card thru the 1622 card reader, or the 1620 operator may activate SUBMON. Each program in the queue executes to completion (with sense switch 1 OFF, 1443 printer and 1622 card punch OFF). All print/punch output is channeled to the 1311 disk where it is stacked sequentially for each program. As a program reaches completion, it returns to the DISQUE in BACKGROUND. When the queue has cycled once, all programs will be ready for output release and the printer or punch will be selected and interlocked with the 1620.

4.2.3 Pass 3 - the Output phase. When the 1620 is interlocked with a peripheral I/O device it requires human intervention, therefore pass 3 must await the arrival of an operator (usually the morning after). Once the required device(s) has been activated, the stored output for each program of the queue in turn will be released uninterrupted in one continuous sequence.

4.3 Programming instructions for the Overnight Production Line appear in Appendix A. Operating instructions may be found in Appendix B.

5. Subroutine MISER

5.1 Consider the following:

5.1.1 A program whose frequency and/or extent of data input is dependent on program execution, i.e., logical decisions resulting therefrom, may impose upon the 1622 card reader for the duration of the program.

5.1.2 It might sometimes be desirable or expedient to re-read one or more cards, or an entire data set, perhaps even under differing formats, without having to re-stack them in the card reader hopper.

5.2 Both of these obstacles may be overcome by the use of subroutine MISER. Card images can be stored on 1311 magnetic disk in the same order in which the cards appear in the reader, or even in a different order, according to the wishes of the programmer. The programmer has complete control over the location of the card-image records within the boundaries of the available disk space; it is merely necessary for him to keep an index of the records (physical or logical) within the program. The easiest method of course is to use sequential records beginning with number one.

5.3 In its simplest form, MISER functions as follows:

5.3.1 A card is read into the program, either free-style[1] or under format, e. g.,

```
READ, X, Y, Z, M, N
```

5.3.2 This is followed by a call to MISER with the positive value of the desired record number as argument, e. g.,

```
CALL MISER (1)
```

5.3.3 The card contents are stored on disk in record number one (1). The values read from the card for variables X, Y, Z, M, N may or may not be used and have no effect upon the operation of MISER or the storage of the card image.

5.3.4 The same card contents may be retrieved and reread many times, either immediately or at any time during the program, under any style, and only as much data extracted from it as desired. This is accomplished by first placing a call to MISER, with the negative value of the record number as argument, followed by the required READ statement, e. g.,

```
CALL MISER (-1)
READ, J
```

Note that any value from a card may be chosen at will and read into any variable, irrespective of mode, at different readings. In the last example, J will assume the value formerly associated with X on the first READ; the remainder of the card is ignored at this time.

5.4 Although the general method outlined above for using this subroutine provides the programmer with a prolific means of exercising control over his input data, a more powerful concept exists with the reading and storing of entire data sets in one fell swoop at the beginning of a program. The card images are then available on a permanent basis, either individually or collectively, for the duration of the program (also

continued.....

1. No. 01.02.019 in the 1620 Program Library.

LINK programs and succeeding ones in the job stream).

5.5 MISER in BACKGROUND. It is acceptable to use MISER in the BACKGROUND mode, however, like DPRINT, it must be under control of the sub-monitor and the program must enter the DISQUE for the Overnight Production Line. At present, only one program per queue can be scheduled in this manner, and of course it must also use DPRINT. At this point then, we will introduce the concept of "spooling" under the sub-monitor.

6. Simultaneous Peripheral Operations On-line (SPOOL)

6.1 Spool may be familiar to users of larger and faster computing systems (such as the IBM system/360).

Insofar as the 1620 is concerned, the concept could only be applied to the initial reading of cards and their storage on 1311 disk during the 1622 clutch cycle, concurrent with CPU processing. However this, together with the similar queueing of output and its subsequent channeling to a selected device, should help communicate to the novice some idea of the capability of the IBM System /360.

6.2 Subroutine SPOOL. When MISER and DPRINT are both required in a program for overnight scheduling, the statement "CALL SPOOL" will replace the previous call to DPRINT at the beginning of the program. Using SPOOL causes DPRINT and MISER to be loaded with the program initializes them for the sub-monitor during pass 1 of the OPL, queues all input data onto disk, and then kicks the program into BACKGROUND on the DISQUE. It is not necessary to include an *BACKGROUND control card.

Further details on the use of SPOOL in the OPL appear in Appendices A and B.

7. Check-point and Re-start

7.1 Another facility which will be familiar to users of the IBM system/360 is that of check-pointing a program during execution and subsequently re-strating it from one of the check-points. The purpose of such a device is to store a program complete with its status as it appears when the check-point is reached during program execution. The disk storage area used is transient, and will always yield a version of the program as it existed during the most recent check-pointing. Should

Continued.....

the 1620 malfunction, or should it be necessary to cease processing suddenly, or should it be desirable to follow alternate processing paths in a long-running program then the re-start procedure will resume the version of the program last stored at the check-point.

7.2 Subroutine CHKPT. Whenever the statement "CALL CHKPT" is executed in a program an image of the entrie 1620 memory is stored in the common check-point save area on 1311 disk. If sense switch 4 is found to be ON, the program in memory will be terminated on an end-of-job condition, otherwise processing will be resumed with the next statement following in the program.

7.3 Program RESTRT. This is a disk-stored program which, when executed, will re-start the processing of whatever program happens to be residing in the common check-point save area.

As an example, a programmer may decide that, instead of executing a time consuming program several times, he can place a CALL CHKPT at a strategic point in the program which perhaps requires considerable time to reach. This point may constitute a decision such as selection of additional input. Then, whenever the program is restarted, it may take different logical paths depending on the input it receives. Re-start is also available as a one-card utility loader.

8. Summary

8.1 Running FORTRAN II-D under control of the sub-monitor provides the facility for manual or automatic interrupt of executing programs, and storage on 1311 magnetic disk by DSAVE through the use of *BACKGROUND or CALL DSTASH. Such programs form a Disk Job Queue (DISQUE) and await their turn at continuation in cyclic order, perhaps many times before completion.

8.2 By placing the statement "CALL DRPINT" at the head of a FORTRAN II-D program, provision is made for bypassing immediate use of the 1443 printer or 1622 card punch and queueing of output on 1311 disk packs. Output may be channeled to the appropriate devices on completion of the program.

8.3 Time-consuming programs in BACKGROUND which make use of subroutine DPRINT, qualify for the Overnight Production Line (OPL) and are better-equipped for completion within a reasonable period of time.

8.4 Input data may also be queued on 1311 disk by the use of subroutine MISER. The same data may be read by the program many times and in any style, format or mode desired, without the use of the 1622 card reader.

8.5 A program may undergo "spooling" of input and output in the DISQUE Overnight Production Line. By calling upon subroutine SPOOL, the program utilizes subroutine DPRINT, subroutine MISER, library subroutine 22 (BACKGROUND) and program DSAVE in order to effect the necessary stages of queueing for successful completion of the program.

8.6 A program may preserve its state at a check-point within the program by executing the statement CALL CHKPT. Computer memory in stored intact in the common check-point save area on 1311 disk, and the program may be re-started from that point by executing the program "RESTRT" or by using the one-card re-start utility loader.

Appendix A - Overnight Production Line (OPL)

Programming Instructions

GENERAL

For the user who has no particular programing problem, or no severe memory space limitation, these general instructions should suffice.

1. Programs should be checked out thoroughly, and completely debugged before being submitted to the OPL. A program which check-stops or otherwise "hangs up" will hold up the remaining programs in the queue.

2. Include the control card "*BACKGROUND" at the head of the source deck with other FORTRAN II-D control cards.

Example 1

```
**JOB            surname
**FORX
 *BACKGROUND
 *LIST PRINTER
     etc.
```

3. Code the statement "CALL DPRINT" early in the program, at least within the first 5 minutes of execution. If possible, place at the beginning of the program.

4. Arrange to have all input data read at the beginning of the program, and at least within the first 5 minutes of execution.

5. Insure that the statement "CALL EXIT" is the last executed statement of the program (although not necessarily the last source card before the END statement). CALL EXIT must be reached.

Example 2 - vector addition

```
CONTROL CARDS | **JOB            surname
              | **FORX
              | *LIST PRINTER
              | *BACKGROUND <--

SOURCE        | CALL DPRINT <--
              | DIMENSION A(500), B(500), C(500)
```

Continued.....

```
1  READ,N,(A(I),I=1,N),(B(I),I=1,N)
   DO 2 I = 1, N
   C(I) = A(I) + B (I)
2  PRINT 3, A(I), B(I), C(I)
3  FORMAT (3 E 11.4)
   CALL EXIT
   END
```

6. The statement "PAUSE" and "STOP" must not be used.

7. Additional memory space required

subroutine DPRINT	=14 cores
library subroutine 20	=01930 cores
library subroutine 22	=00112 cores
*BACKGROUND	=8-13% of main program length
total	=2056 + 8-13% main.

Programs With Memory Space Restriction

Subroutine DSTASH

A programmer who uses most of the available computer memory for storing main program and subroutines may not be able to use the *BACKGROUND control card for fear of overflowing memory. In such a case, use should be made of subroutine DSTASH.

1. All the steps outlined for the General method should be followed with the exception that the *BACKGROUND control card is NOT included.

2. Code the statement "CALL DSTASH" at a convenient point early in the program, after the statement CALL DPRINT and if possible, after all input data has been read. Execution of this statement will cause an automatic interrupt and storage of the program on 1311 disk. This one call to subroutine DSTASH is sufficient for the OPL. since the submonitor provides all other interrupts necessary for the proper processing of the Disk Job Queue (DISQUE).

Example 3 - sorting a floating point array in ascending order.

```
##JOB
##FORX
*LIST PRINTER
CALL DPRINT
DIMENSION X (100)
READ, X
CALL DSTASH
```

```
1   MOVE = 0
    DO 2 I = 1, 99
    IF (X(I) - X(I + 1)) 2,2,3
3   TEMP = X(I)
    X (I) = X (I + 1)
    X (I + 1) = TEMP
    MOVE = 1
2   CONTINUE
    IF (MOVE) 4,4,1
4   PRINT 5, X
5   FORMAT (10(1XE 10.4))
    CALL EXIT
    END
```

3. A programmer who wishes to have contiguous or unseparated output should CALL DSTASH in his program ahead of all output statements.

4. Additional memory space required

subroutine DPRINT	=	14 cores
library subroutine 20	=	1930 "
library subroutine 22	=	112 "
subroutine DSTASH	=	42 "
Total =	=	2098 cores

Spooling I/O on the DISQUE
Subroutine SPOOL

For programs which cannot be arranged to read all input data at the beginning of execution, subroutine SPOOL in conjunction with subroutine MISER may be used in order to queue both input data and output results on 1311 magnetic disk.

1. Apply steps (1), (5) and (6) of the GENERAL method, pertaining to debugging, the use of "CALL EXIT", and avoidance of PAUSE and STOP.

2. Code the statement "CALL SPOOL" (in lieu of CALL DPRINT) at the beginning of the program.

Example 4 - simple linear correlation

```
$$JOB          surname
$$FORX
*FANDK 1204
*LIST PRINTER
```

Continued.....

```
      CALL SPOOL

    3 SUMY = 0.0
      SUMX = 0.0
      SUMXY = 0.0
      SXSQ = 0.0
      SYSQ = 0.0
      DO 4 I = 1, 500

      READ, X, Y
      IF (X) 6,5,5
    5 SUMX = SUMX + X
      SUMX = SUMY + Y
      SUMXY = SUMXY + X*Y
      SXSQ = SUMXSQ + X*X
      SYSQ = SUMYSQ + Y*Y
    4 CONTINUE
    6 FN = I - 1
      R = (FN * SUMXY - SUMX*SUMY)/SQRT(
     1 (FN*SXSQ-SUMX*SUMX) * (FN*SYSQ - SUMY*SUMY))
      RSQ = R * R
      PRINT 14, FN, R, RSQ
   14 FORMAT (13HbbSAMPEbSIZE=I5, 5X2HR=F12.8,
      15X10HR-SQUARED=F12.8)
      CALL EXIT
      END
```

3. Use of subroutine SPOOL causes what is known as "priority scheduling" of the program in the DISQUE. It will be first to be processed during pass 2 of the OPL.

4. Additional memory space required

subroutine SPOOL	=	532 cores
subroutine MISER	=	1128 "
library subroutine 20	=	1930 "
library subroutine 22	=	112 "
Total	=	3702 cores

Appendix B - Overnight Production Line (OPL)

Operating Instructions

GENERAL

The OPL is processed through three separate passes. The first pass may be carried out for each program separately, at different times during the day, normally by one of the Computing Centre staff. This must be under strict control to ensure that the program doing the largest "spooling" is added to the DISQUE first. The staff is also responsible for setting up the OPL data decks if any.

Pass 2 may be initiated by any operator, normally the last computer user at night preceding the scheduled OPL.

Pass 3 is controlled by the first morning operator and requires approximately 30 minutes to release a full disk of output.

In general, should any malfunction occur during OPL in hardware or software (machine or programs), when in doubt, type in a branch to the MONITOR supervisor (490079R/S). More explicit information is supplied in the instructions for each pass.

Pass 1 - Computing Centre Operators

General

1. Compile the program in the normal manner, and permit it to execute for a reasonable time. If *BACKGROUND is used, allow at least 5 minutes.

2. If the program does not execute an automatic interrupt (via subroutine DSTASH), turn sense switch 4 ON. When the message "*BACKGROUND" types out turn switch 4 OFF again, unless the last card has been read thru the 1622 reader and you do not wish to activate the sub-monitor(see Appendix C).

FULL DISQUE

3. If there is no room available in the DISQUE, then carry out one of the three following alternatives:

   a. Turn switch 3 ON, press start and type in a sector address to put the program in a pot-luch area. It may be possible to finish the program presently.

Continued.....

b. Turn switch 2 ON, press start and resume the program if time is currently available.

c. Press start to abort the job, and give it priority in the next OPL. This can be done by placing a "CALL SPOOL" at the head of the source.

Spooling

4. If the program is "spooling", follow the instructions given on the console typewriter. The input data for the program must be removed immediately from the 1622 reader, or well marked (by placing a coloured tag in front of the last card in the stacker). This will go to form part of the OPL job deck and will be the first data to enter the reader during pass 2.

5. Remember that the program using the largest disk space for SPOOL must be run first (this will be the program having the most input data), therefore it must be the last such program to enter the DISQUE.

Program Hang-up

6. Should a program check stop or otherwise hang-up, dispose of it in the usual manner by using the UNCLE post-mortem debugging facility.

Pass 2 - Night Operators

GENERAL

1. Ensure that the "CLOBBER" disk, pack no. 09999 is mounted on 1311 disk drive 0.

2. Turn all sense switches OFF.

3. Place the OPL job deck in the card reader and execute. The deck consists of the following:

```
##JOB                OPL
## (comments and instructions)
##PAUS
##XEQ OPL

(data optional)
```

If a "DSAVE countdown" occurs, allow it to continue thru to the selection of the first program in the DISQUE. Do not turn on switch 4.

4. After execution has begun, press STOP on the 1443 printer and 1622 card punch, advance the printer paper to a new page by pressing CARRIAGE RESTORE, and place a warning card over the printer switches stating that the printer must remain OFF unless attended.

5. Attach the OPL instruction sheet to the 1620 log, turn roomlights out and lock the door when leaving

Program or System Hang-up

6. Should a system error occur while DSAVE is restoring a program to memory, the computer will come to a HALT (48 in the OPERATION REGISTER) within 2 seconds of the message: "DSAVE i name RESUMED". Press CHECK RESET and START to re-try. After 9 tries, CLEAR MEMORY, remove any data cards remaining in the 1622 reader for the aborted program, and LOAD the MONITOR "cold start" card followed by the OPL job and any data remaining for the other programs in the DISQUE.

7. If a user program fails (check-stop) after it is resumed, remove its unused data, if any, from the card reader, replace any other remaining data and press READER START. place the 1620 in MANUAL, press INSERT and type in: "4900796R/S" to re-activate the sub-monitor.

Pass 3 - Morning Operator

GENERAL

1. At the completion of its scheduled time, the OPL should be finished with pass 2, and the 1620 interlocked with the printer or punch. The queued programs will have cycled thru execution once each, and the first one returned to memory for output release. Otherwise, GO TO STEP 8.

2. Turn sense switch 1 ON. Ensure that the 1443 printer is prepared on a fresh page of paper, and the 1622 card punch is prepared with blank cards in the hopper. Press PRINTER START and/or PUNCH START.

3. Attend the output devices until completion of the OPL which requires a maximum of 30 minutes. If you must leave the room, STOP the 1620 or the output devices. On completion, place the output and card deck in the "JOBS COMPLETED" box.

Program Hang-up During Pass 2 (overnight)

4. Should the 1620 be check-stopped or otherwise hung-up because of program malfunction during pass 2, it may be too late to continue with the remainder of the OPL. In this case:

a. Complete the preparations as in (2) and (3) above

b. Turn sense switch 4 ON

c. Press RELEASE-INSTANT STOP(SCE)-RESET-INSERT

d. Type in: "4900796R/S", then turn switch 4 OFF when instructed.

This effectively switches processing from the DISQUE to the card reader.

5. Execute pass 3 manually for each program in the DISQUE which had been resumed during pass 2, as indicated by the typewriter messages. This is done by reading in the following for each program:

```
ǂǂJOB                OPL
ǂǂXEQ DSAVE
(blank card)
(blank card)
```

When the message "ENTER AREA NO." appears on the console typewriter, type in the digit corresponding to i of the message "DSAVE i . name RESUMED" which had appeared on the typewriter during pass 2 of the OPL. This will release the stored output for that program.

Program Hang-up During Pass 3

6. If a program check-stop or other hang-up occurs after pass 3 has begun, merely RELEASE-INSTANT STOP-RESET-INSERT and type in "4900796R/S" to continue processing.

Running Overtime in Pass 2

7. If it becomes necessary to interrupt on OPL which is running overtime in pass 2, do so as follows:

a. Turn sense wwitch 1 ON. If no output occurs within a reasonable time, turn sense switch 4 ON.to return the program to BACKGROUND.

b. Prepare the output device (s) and make them READY as in steps (2) and (3) above.

c. If there is output, turn sense switch 4 ON. The program will return to BACKGROUND when output is complete.

d. Resume each of the programs in turn as indicated on the typewriter paper during pass 2, by manually activating DSAVE as outlined in STEP 5 above. (except the one which ran overtime)

1311 Disk Overflow

8. If the OPL is still in pass 2, i.e., the queue is still undergoing its first cycle, and yet the 1620 is interlocked with the printer or punch, this indicates that all the available disk space has been filled with program output, and the sub-monitor is attempting to accommodate more. Proceed as follows: (Otherwise, GO TO STEP 7)

a. Prepare the printer and/or punch, make them READY as in steps (2) and (3) above, turn sense switch 4 ON and allow output to continue until the program returns to BACKGROUND, or for a maximum time computed as follows:

Time (minutes) = 30/no. of programs already resumed.

b. If the time allowed is exceeded, press STOP (SIE) on the console then,

c. Press INSERT and type in "4900796R/S"

d. Resume each of the programs in turn which were processed during pass 2, by executing DSAVE as in STEP 5. (except the program already done above).

Appendix C - The Disk Job Queue Sub Monitor

1. GENERAL

The MONITOR II Supervisor Job Card Read Routine has been modified in order to perform the following additional functions:

a. When END OF JOB is reached, console switch 4 is examined. If ON, a new JOB is accepted from the 1622 reader. If OFF, the LAST CARD indicator is examined. (This indicator goes ON when data from the last card has been transferred to memory. It is turned OFF by interrogation.)

b. If LAST CARD indicator is ON, the sub-monitor goes into a 10 sec. countdown loop, at the end of which, DSAVE will be called into memory to begin automatic execution of saved programs. If switch 4 is turned ON before the countdown is complete, a new JOB will be accepted from the reader.

c. If LAST CARD indicator is OFF, the sub-monitor then examines the Disk Job Queue (DISQUE) indicator to determine if DSAVE has priority over new JOBs from the 1620 reader.

d. If the DISQUE indicator is ON, a countdown begins, which can be interrupted by turning console switch 4 ON before completion, otherwise DSAVE will go on to execute the next background program in the queue.

e. If the DISQUE indicator is OFF, a new job is accepted from the 1622.

Once a countdown loop is completed, DSAVE will select one of the available saved programs from the DISQUE without any operator action, and run it in the "background" mode. Each time DSAVE is entered, the DISQUE counter is incremented by ONE and the correspondingly numbered SAVE area (1,2,3,...) is executed. When all are completed, the sub-monitor awaits a new JOB from the card reader.

2. OPERATING PROCEDURES

a. Turning switch 4 ON before END OF JOB occurs will prevent the countdown and accept a JOB from the 1622 reader. Otherwise, SWITCH 4 ON will interrupt the countdown

.....continued

after it begins. After the message "TURN OFF SWITCH 4", and after switch 4 is turned OFF, it may be turned ON again immediately if desired.

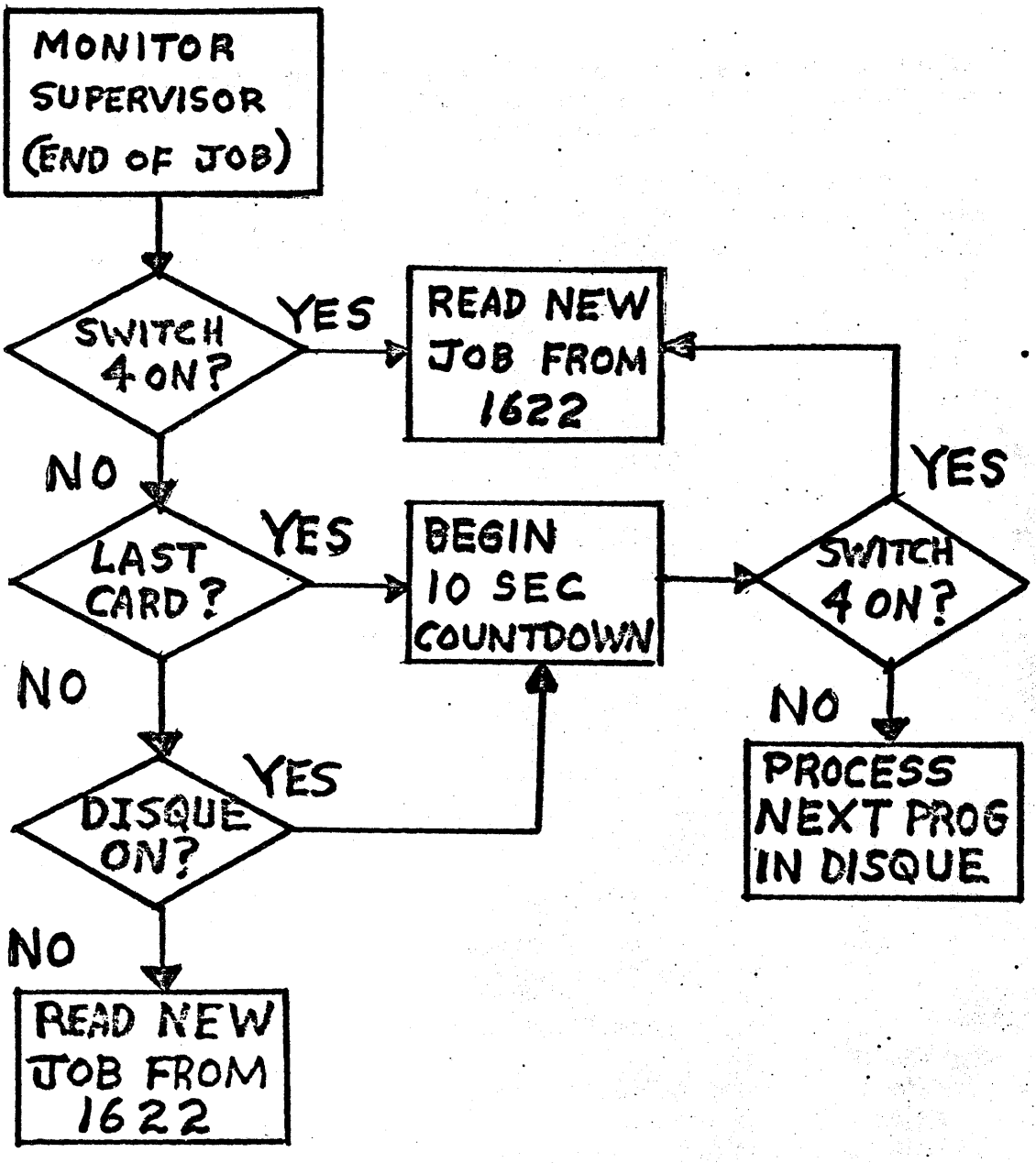
b. Placing TWO BLANK CARDS at the end of a deck will allow the card deck proper to be read completely without turning on the LAST CARD indicator. NON-PROCESS RUN-OUT does NOT turn on this indicator.

c. Once a saved program has resumed execution, it may be suspended again at any time by the operator in the usual manner.

d. Control may be passed from the MONITOR supervisor to the sub-monitor at any time by:

(1) Reading a blank card through the 1622 card reader, or

(2) Typing in a branch to 40048.



DISQUE SUB-MONITOR

FLOW CHART

Adaptation of an IBM 7094 Scientific Program for Use on the IBM 1620 II Computer

J. M. Read, Jr., R. W. Crecely, J. H. Goldstein and R. S. Butler

The scientific user with access to a 1620 can sometimes be confronted with a problem. Programs may exist which, while performing the calculations necessary to his field of research, have been written for other machines and thus are not usable by the scientist. Common programming languages such as FORTRAN to a great extent alleviate this problem but difficulties still exist. In particular, problems are encountered when one attempts to implement programs written for large, fast computers since the 1620 user may find that: (1) the program consumes an excessive amount of core storage; (2) the execution time is so slow that use of the program becomes impractical. These problems were encountered by our installation when the implementation of LAOCOON II (a 7094 FORTRAN IV program for the calculation of nuclear magnetic resonance spectra) was attempted. Our computer is a 1620 II with 40K, floating point hardware, one disk drive and paper tape IO.

LAOCOON II possessed a number of basic improvements over the program in current use at that time. It was thus highly desirable for the new program to be put into operation. Unfortunately the program was of such size that it could not be fit into core storage. Data storage alone required 32,000 core

positions in order for the full capabilities of the program to be retained. The decision was then made to rewrite LAOCOON in basic machine language but because of the stringent core storage requirements the MONITOR system SPS II-D, requiring 2402 core positions at execution time, was unacceptable. Instead SPS III was used.

The problems imposed by construction of a machine language equivalent outside the framework of the MONITOR system became immediately apparent. The translation of statements involving only arithmetic calculations was straight forward. The real problems were created by the absence of system routines. For instance, there were no input-output routines for either the typewriter or the disk. Also, routines to perform the normal MONITOR functions of loading program links during execution and loading compiled programs to disk storage did not exist. It is this facet of the project which we will primarily discuss.

The basic design philosophy of the system routines involved the construction of programs with enough generality to handle the various tasks required of the program but limited enough so that core requirements could be held to a minimum. Because of the restricted nature of the routines no attempt will be made to provide descriptions of internal program logic. Instead attention will be paid to specific examples of the use of the substitute FORTRAN system routines for the purpose of indicating one solution to the problems caused by the absence of an operating system.

Figure 1 shows an example of the format required for the use of the typewriter output routine. The routine is entered with a branch and transmit immediate containing as the Q field data the address of a two digit field following the instruction. This first field gives the total number of items to be assembled into the IO buffer. In this example there are four alphameric labels and three numbers, making a total of seven items. Following this is a DEFINE SYMBOLIC ADDRESS containing a minimum of two operands. The first gives the location of the format list; the last gives the address of the next instruction to be executed after the output operation has been completed. The intervening operands, if any, specify the addresses of the numbers to be assembled into the output buffer. They can be either fixed or floating point numbers completely interchangeably. The format list consists of a series of four digit codes. The third digit of the code is a record mark if control or alphameric information is to be processed. A flag over the second digit differentiates between the two, with alphameric data indicated by the presence of the flag. If there is no flag over the second digit and the first two digits of the field are zeros, a carriage return is executed. If the first two digits have a non-zero value, the indicated number of spaces is assembled into the output buffer. In our example the first code indicates a carriage return while the second indicates that five spaces are to be assembled into the output buffer. The next code contains a flag over the second digit indicating that alphameric data is to follow. In this

case the first two digits of the code indicate the length of the alphameric data field. The data must immediately follow the code and have a record mark assembledas the last character. If a record mark is not the third character of the code word, numeric data is indicated. A flag over the fourth digit indicates a floating point quantity; no flag indicates fixed point. The first two digits give the total width of the field while the last two indicate the number of places to follow the decimal point. If the number is too large to fit within the specified field width, an error is indicated and the offending value is displayed on the typewriter in internal floating point form. When the specified number of items have been assembled into the output buffer the line is typed and an exit occurs to the last address given in the DSA.

Perhaps the most important routine which had to be written was the disk input-output routine. The nature of the calculation of NMR spectra is such as to require the frequent movement of large amounts of data to and from disk storage. Hence a conveniently used disk IO routine was a necessity. Figure 2 shows an example of the input data required. After the entry to the program the first item required is a one digit code indicating whether an input or output operation is to be performed. Zero indicates input; one indicates output. Next comes the location of the sector address where reading or writing is to begin, followed by the return address to the mainline program. Each item in the IO list which follows is composed of two or more items. The first item is either a zero, one,

or two in a single digit field which indicates respectively whether a scalar, vector or matrix is to be transferred. A flag over this digit indicates that the items to be transferred are in floating point mode. For a scalar quantity the only additional information required is the address of the scalar. For a vector the addresses of the first element of the vector, the initial index value and the final index value follow the indicator code. The entry for a matrix consists of the indicator code, a three digit field containing the number of elements per column and a series of addresses which specify, respectively, the first element in the matrix, the initial and final row indices and the initial and final column indices. A record mark as the indicator character indicates that the IO list has been satisfied and causes an exit to the previously specified address. As in FORTRAN when control is returned to the mainline program the sector count has a value one greater than the last sector written or read.

The extensive matrix manipulations required by the program necessitated the frequent use of subscripted arrays. Because of the large number of times a subscript routine would be executed it was decided to make this operation as restricted as possible. Single subscripts were calculated in line (i.e. no subroutine was used for their calculation) by simply multiplying the index value by the appropriate word length and adding to the result the address of the first element in the array. The sum was then used as an indirect address. Implicit

in this method is the requirement that for N items the index run not from 1 to N but from 0 to N-1. This same restriction was carried over into the double subscripting routine. Input to the double subscripting routine consists of the address associated with a DEFINE SYMBOLIC ADDRESS declarative which has as operands the labels of the row and column indices. An increment is calculated and placed in core such that when the array address is added to the increment the result is the address of the desired element in the array. Before the subscripting routine can be used the column length of the array must be stored in a fixed location in core. The calculation of a subscript requires the execution of only six instructions not including entry into the subroutine.

Perhaps the best example of the gain in speed at the loss of generality is provided by the linkloading provisions. Once the program has been compiled the core locations of the various routines are fixed. This enables one to store the program links on the disk in core image format making subsequent loading very rapid. The "system loader" consists of a disk seek instruction, disk read instruction and a branch instruction located in an area of core storage that is never overlayed. When a new program link is to be brought in, possibly overlaying the calling program, a disk control field containing the appropriate information is tranferred to the location required by the disk instructions. Also the branch instruction is supplied with the address of the first executable statement in

the new program. When the program logic allows, the seek will be initiated by the calling program in order to allow for processing overlap. The entrance to the loader routine is then made at the disk read instruction. Under ordinary circumstances entrance is made at the disk seek. A routine also had to be written to load the assembled programs into disk storage. This task is accomplished through the use of TRA-TCD instructions which are used to load the program into core storage along with the information needed to store the program on the disk. The disk loader routine is first loaded to core storage. Information regarding the use of the program is typed and the first program is loaded into core by means of a TRA instruction. Each program is begun with a DEFINE ORIGIN which places within the load routine the program number (for identification purposes only), disk storage address, the number of sectors required by the program, and the locations in core storage where the program begins and ends. Another DEFINE ORIGIN is then used to place the program into the proper area. The last instruction in each program is a TCD instruction transferring control to the disk loader which stores the program on the disk and records certain information about the program on the typewriter. Depending upon a sense switch setting, the loader either halts or attempts to load the next program. A special code in the last program terminates loading.

The finished program consists of twenty-eight subprograms and contains over five thousand source statements. Some idea of the savings in core storage can be gained by noting that the

full capabilities of the 7094 program were retained. No accurate estimate of the increase in execution speed can be given since the FORTRAN version could not be run on the 1620. However, comparison of the machine language matrix diagonalization routine used and its FORTRAN equivalent indicates that execution speed is increased by a factor of three.

```
        BTM OUTPUT,*+13
        DC  2,7
        DSA LIST-2,WIDTHO,HIGHTH,SPECWD,R2P
        DS  1
LIST    DC  3,@
        DC  1,0
        DC  3,5@
        DC  1,0
        DC  3,-14
        DC  1,0
        DAC 14,LINE WIDTH =
        DC  4,-603
        DC  3,-7@
        DC  1,0
        DAC 7, HZ   @
        DC  4,-702
        DC  3,-25@
        DC  1,0
        DAC 25, CM/INT   SWEEP WIDTH = @
        DC  4,-702
        DC  3,-4@
        DC  1,0
        DAC 4, HZ@
R2P     NSI
```

FIGURE 1. TYPEWRITER OUTPUT FORMAT EXAMPLE

```
          BTM DISKIO,LIST
LIST      DC  1,1
          DSA SECTCT,RET
          DC  1,0
          DSA CNSTNT
          DC  1,1
          DSA VECTOR,INITAL,FINAL
          DC  1,2
          DC  3,7
          DSA MATRIX,IROW,FROW,ICOL,FCOL
          DC  1,0
RET       NSI
```

FIGURE 2. DISK INPUT-OUTPUT FORMAT EXAMPLE

SESSION REPORT

COMMON - Chicago

Session Number WED B1 Session Name Intermediate Course for Applications Programmers

Chairman L. H. Baker

Time 10:30 to 12:00 AM Attendance (No.) 57

Speakers Mr. Peter Savides
Corporate Systems Education
CHQ Dept. 684
IBM
Red Oak Lane (Soong Estate)
White Plains, New York 10604

Synopsis of Meeting The origin, content, availability and objectives of the ICAP program were reviewed. Key points reviewed were:

1) ICAP was suggested by the SET committee of Guide and developed by IBM.
2) It is designed to provide an outline of an intermediate or advanced course in programming techniques for people who can organize and teach the material in their organization.
3) It is available to COMMON members interested. They should write directly to Mr. Savides.

ICAP

INTERMEDIATE COURSE FOR APPLICATION PROGRAMMERS

This handout consists of some representative material from the ICAP package. The total package consists of 32 instruction modules, each covering a specific programming topic. Every module consists of the following elements:

1. Cover Page - Describes contents, objectives, prerequisites and hints for the instructor.
2. Learning Plan - Overview of the module and its contents. Suggested sequence and method of presentation.
3. Student Guide - Describes contents, relation of the module to the student's professional growth, and hints for the student.
4. Customizing Guide - Guideline suggestions to assist the instructor in tailoring the module for presentation.

In addition to this material, found in all modules, each contains some or all of the following supportive material:

1. Foils - In printed form that can be converted to overhead transparancies.
2. References - Abstracts of published material or specially written for ICAP to support the IBM publications.
3. Examinations and Special Supporting material.

The enclosed representative ICAP material is as follows:

1. A listing of ICAP modules.
2. Cover Page, Learning Plan and Student Guide for Module 24; Introduction to ICAP.
3. Cover Pages for following modules:
   - 5 Controls
   - 7 Standards
   - 8 Documentation
   - 18 Special Characteristics of COBOL F
   - 20 DASD Techniques
   - 26 Planning for Programming
   - 34 Arithmetic Statements and Efficiency
4. Cover Page for Module 10: Communication Techniques (Linkages).
5. Customizing Guide for Module 23; Overview and Review of Basics.

If you would like additional information on ICAP please contact:

MR. PETER SAVIDES
IBM - Corporate Systems Education
CHQ Dept. 684
Red Oak Lane (Soong Estate)
White Plains, N. Y. 10604
Phone: (914) 696-5962

ICAP MODULES

MODULE NUMBER	MODULE TITLE
1	Data definition
2	Program Logic Control
3	List Processing
4	Table and array techniques and the operation of the compiler
5	Controls
7	Standards
8	Documentation
9	JCL
10	Program communication
11	Data set manipulation
12	Testing
13	Programming efficiency
14	Systems
15	Report writer
16	Sort
17	Maintenance
18	COBOL F features
20	Direct access techniques
21	Cutover & conversion
22	Case study
23	Overview & review
24	Introduction to ICAP
25	Modular programming
26	Programming planning
27	Program generalization
28	Programming systems
29	MIS and data base
30	Multiprogramming and asynchronous processing
31	Program design
32	IBM Programming Support
33	Decision table coding
34	Arithmetic statements and the operation of the compiler

ICAP COVER PAGE MODULE 24: INTRODUCTION TO ICAP

ABSTRACT OF CONTENTS: Provides for the welcome and other logistics of the class. Gives the purposes of ICAP, its history, and content. Reviews the changing environment of application programming which, in part, led to the need for ICAP. Provides a frame of reference for the modules which follow. Includes the motivation of students.

OBJECTIVES OF MODULE -- The student will:

1. Know the purposes of ICAP.
2. Put extra effort into learning, because he is made to see the importance of ICAP to his professional development and, hence, to his career.
3. Obtain a frame of reference for the content of ICAP This frame of reference will include the objectives of ICAP, its schedule, and environment within which application programmers can expect to work in the future.
4. Be aware of the logistics of the class.
5. Understand the methods to be used in evaluating his performance in ICAP.
6. Use the self-evaluation aids provided in ICAP.

STUDENT PREREQUISITES --

Knowledge required: COBOL, Basic Programming, OS/360, JCL, DASD, Basic data processing
Experience required: 6-12 months programming in COBOL. Have completed one major program.
Material to be studied: ICAP References 24-1, 24-2, 24-3, 24-4, 24-5, 24-6, 24-7, 24-8

HINTS FOR THE INSTRUCTOR: The spirit in which you approach ICAP has a great deal to do with how successful your students are at achieving the objectives of the course. It is up to you to reveal to them the importance of ICAP to their future; it is your job to arouse their enthusiasm and obtain their commitment to achievement. During ICAP, you should always seek higher levels of performance than you would expect in day-to-day work. This striving toward excellence will help compensate for the inevitable slacking off and forgetting which occurs after a training period is over.

ICAP LEARNING PLAN MODULE 24: INTRODUCTION TO ICAP

OPENING: Welcome to ICAP, the Intermediate Course in Applications Programming. Our version of ICAP was adapted from the original course produced by IBM under the auspices of GUIDE. (Hand out copies of outline) HANDOUT

PREVIEW: This module handles the mechanics of the class. It sets the pace and tone for the following two weeks. It places the students' classwork in the context of requirements of their jobs and the environment of programming.

LOGISTICS Days and hours of class. (Two weeks recommended) Materials available. (See suggested list found in Reference 24-9) Other logistics related to local educational conditions.

PURPOSES ICAP is designed to begin the professionalization of the relatively inexperienced programmer. You can call your students' attention to the list of capabilities given in Reference 24-5. Achieving more of the capabilities of the intermediate applications programmer is only one of the purposes of ICAP. Using OPF 24-0 explain the goals of ICAP in your company. Have each student write his interpretation of the purposes of ICAP as they apply to him. OPF 24-0 EXERCISE

EVALUATION Measuring performance against purposes is perhaps even more important in ICAP than in most courses. Because ICAP is designed on a professional level, the student's evaluation of his own progress becomes central to evaluation. In order to do a good job of evaluation, he needs your help:

1. In making a committment to the purposes of ICAP.
2. In relating his learning to purposes.
3. In establishing and using criteria of accomplishment.
4. In determining and recording his accomplishment. (The evaluation forms provided with ICAP can be useful here). OPF 24-1 OPF 24-2 OPF 24-3
5. In assessing his progress relative to criteria of achievement in ICAP. (Examinations formed from the questions provided with ICAP can serve as one measuring instrument. You can provide other measures of achievement, such as effort required to complete problems.

PROGRAMMING ENVIRONMENT Application programmers find it necessary to adjust to frequent changes in their working environment. Some of these changes are as important as a move to the next generation of equipment and software; others are as minor as a change in the slashing of letters in coding. On OPF 24-4 explain the meaning of each change. On OPF 24-5 ask for comments. OPF 24-4 OPF 24-5 DISCUSSION

ICAP LEARNING PLAN MODULE 24

Point out how many modules within ICAP deal with problems raised by changes in the programming environment. Encourage participation by all students in the discussion.

Examine the details of changes in the programmers job as the environment changes. Use programming language change as an example.

OPF 24-6
OPF 24-7

ICAP STUDENT GUIDE MODULE 24: INTRODUCTION TO ICAP

ABSTRACT OF CONTENTS: Provides for the welcome and other logistics of the class. Gives the purposes of ICAP, its history, and content. Reviews the changing environment of application programming which, in part, led to the need for ICAP. Provides a frame of reference for the modules which follow.

OBJECTIVES: When you complete Module 24 you will:

1. Know the purposes of ICAP.
2. Have related those purposes to your own professional programming goals.
3. Know how to evaluate your own progress during class.
4. Be aware of how changes in the environment of programming prompted the development of ICAP.

RELATION OF MODULE TO YOUR PROFESSIONAL PROGRESS: ICAP is designed to prepare you to step up into the ranks of the more experienced programmers. It is intended to:

1. Consolidate your knowledge and experience.
2. Locate and fill gaps in your knowledge of the fundamentals of programming.
3. Introduce you to new concepts, techniques and skills.

ICAP provides you with an unusual opportunity to increase your professional competence as a programmer. Properly, used, these days in class can enhance the value of your experience and provide you with new competence to handle more difficult and rewarding assignments.

You have a responsibility to yourself to learn as much as you can in ICAP. The intermediate applications programmer should possess many capabilities. Some of these are listed in ICAP Reference 24-5.

HINTS FOR THE STUDENT: This is the beginning of your work on ICAP; success comes from good beginnings. It is extremely important that you begin by relating the goals of ICAP to your goals as a progessional programmer. This undoubtedly will require a searching examination by you of your career to date, your response to those experiences, and your hopes for the future. Such an examination cannot be concluded in one session; it will extend throughout ICAP and probably beyond. However, your instructor will ask you to make a first iteration of this analysis and to write down an initial approximation of the

relationship of your personal goals to the purposes of ICAP.

The ordinary educational practices of good study habits, evaluation of progress, etc. apply to ICAP as to any other course. ICAP differs in that your job experience before and after the classroom sessions is considered to be part of ICAP. You will work to integrate previous experience with new knowledge and prepare to make subsequent experience serve the purposes of ICAP.

ICAP COVER PAGE MODULE 5: CONTROLS

ABSTRACT OF CONTENTS: Covers the possibilities of loss of control, the techniques used to determine control conditions and the methods of restoring and retaining control. Special attention is given to the checking of the quality of input and output.

The underlying considerations: Create an appreciation for control requirements and the capability to satisfy them.

OBJECTIVES OF MODULE -- The student will:

1. Include sufficient validity checks in his programs.
2. Prepare a suitable controls plan for the case study.
3. Choose control techniques appropriate to an application and to his local requirements.
4. Build controls into his programs which will stand up to the severe test conditions imposed on them by the instructor.

STUDENT PREREQUISITES:

Knowledge required - some familiarity with the systems implications of controls

Materials to be studied - ICAP References 5-1, 5-2. IBM F20-0006, C20-1649 (Ch.9)

STUDENT ACTIVITIES DURING MODULE: 1. Prepare a controls plan, 2. Build controls into programs, 3. Attempt to circumvent the controls of fellow students, 4. Do exercises during lecture discussion

HINTS FOR THE INSTRUCTOR: Because management policy has a great deal to do with the type and extent of controls, you should adjust this module to reflect the policies of your management. Students and new programmers tend to over-control once they become convinced of the need for controls. This is normal, but can be tempered by emphasis on the economic evaluation of controls.

ICAP COVER PAGE MODULE 7: STANDARDS

ABSTRACT OF CONTENTS: The importance of installation standards is directly related to the benefits obtained from having them. The stake of management in standards is compared with that of programmers in standards. Programmer influence on the establishment and effectiveness of standards is reviewed.

OBJECTIVES OF MODULE -- The student will:

1. Use his local standards.
2. Prepare programs which adhere to standards.
3. Locate the need for standards and design standards.
4. Plan the inclusion of standards in his programs and his programming activities.
5. Have an appreciation of the importance of documentation as embodying significant standards.
6. Obtain the benefits which come from using standards properly.
7. Relate standards to his own effectiveness and to group effectiveness.

STUDENT PREREQUISITES --

Knowledge required: Know local standards

Experience required: Have worked with local standards.

Material to be studied: Local standards manual, ICAP References 7-1, 7-2

STUDENT ACTIVITIES DURING MODULE:

1. Analyze local standards.
2. Learn vital local standards.
3. Practice setting a standard.
4. Comply with standards in ICAP lab work.

HINTS FOR THE INSTRUCTOR: Standards imply compliance. This includes compliance in the classroom as well as in day-to-day work. In this module you will be attempting to get the student to reach a clear awareness of the direct ties between standards and his own best interests. Thus, he should be made to see that standards compliance favorably influences productivity and quality of output. Standards compliance reduces effort and frustration.

ICAP COVER PAGE MODULE 8: DOCUMENTATION

ABSTRACT OF CONTENTS: The standard programming documentation techniques such as flow charts, system descriptions, and comments in the coding are considered in detail. Guides to proper documentation are given. The importance of installation standards of documentation is emphasized. The underlying consideration: how to communicate with audiences at various technical levels.

OBJECTIVES OF MODULE -- The student will:

1. Produce adequate written communications for use of all the people interested in the program.
2. Successfully use various documentation techniques, particularly those provided by COBOL (F).
3. In the case study, integrate documentation with other programming actions.
4. Document problem programs so that another student may successfully debug one without questioning the originator.
5. Inspired to document and get others to document.
6. Pass an examination on the module.

STUDENT PREREQUISITES:

Knowledge required: Use of NOTE., Documentation standards of his installation.
Materials to be studied: Technical report on documentation and associated handouts.

STUDENT ACTIVITIES DURING MODULE:

1. Examine own documentation performance of past.
2. Document case study and problems in prescribed manner.
3. Use documentation prepared by other students.

ICAP COVER PAGE MODULE 18: SPECIAL CHARACTERISTICS OF COBOL F

ABSTRACT OF CONTENTS: Emphasis is on the features of COBOL F itself (their uniqueness in that they go beyond the stated features of COBOL). The extensions of COBOL F are covered and distinctions made between what is currently available and what will be available.

OBJECTIVES OF MODULE ---The student will:

1. Know the COBOL F features subset.
2. Know the extensions to COBOL F
3. Use the F features and extensions.

STUDENT PREREQUISITES:

Knowledge required: The features of COBOL E or COBOL for some system other than OS/360.

Experience required: Coding of at least 2 COBOL programs or major routines of a COBOL program.

Materials to be studied: COBOL F Language manual. COBOL F Programmer's Guide.

HINTS FOR THE INSTRUCTOR: As with most of the other modules, the important thing is that the student understand the concepts presented rather than have a fully detailed capability in the concepts' use. He should know what the features are and how they can be used - and where to set the details to code them.

ICAP COVER PAGE MODULE 20: DASD TECHNIQUES

ABSTRACT OF CONTENTS: This module covers the problems inherent in using disk storage (primarily the 2311). Although considerable emphasis is on review of device physical characteristics, the core of the module includes the various types of data set organization allowed and the access methods used to create and retrieve these files.

DIRECT ACCESS TECHNIQUES: Contains the concepts and COBOL facilities which the programmer will need in order to utilize and implement information on direct access devices with COBOL. The underlying considerations: facility in using direct access devices and non-sequential files.

OBJECTIVES OF MODULE ---The student will:

1. Know the physical characteristics of disk storage.
2. Know how to efficiently use disk storage.
3. Know the JCL requirements to effectively use the medium.
4. Use new COBOL F features associated with DASD.
5. Organize files on DASD.

STUDENT PREREQUISITES:

Knowledge required: Characteristics of disk storage devices (especially 2311) and file organization.

Experience required: Use of disk storage and associated JCL statements to specify disk areas, data set access, etc.

Materials to be studied: DASD manual JCL chart on disk parameters for DD cards.

HINTS FOR THE INSTRUCTOR: You will need to fill in the student's knowledge of disk storage as an auxiliary storage device in order to pave the way for showing him how to more effectively use the medium.

ICAP COVER PAGE MODULE 26: PLANNING FOR PROGRAMMING

ABSTRACT OF CONTENTS: Planning is an activity which should precede all aspects of programming. Module covers planning for each activity from program design to maintenance. Planning techniques reviewed.

OBJECTIVES OF MODULE -- The student will:

1. Prepare a series of programming plans for the case study.
2. Have a formal model of planning which he knows how to use.
3. Realize the connections between adequate planning and effective programming.
4. Pass an exam.

STUDENT PREREQUISITES:

Material to be studied: ICAP references 26-1, 26-2

STUDENT ACTIVITIES DURING MODULE:

Prepare programming plans

HINTS FOR THE INSTRUCTOR: The student is asked to plan in a much more formal manner than he is likely to follow on the job. This formality is deliberate to insure that he does all of the planning required, and so that you can check concrete details in his plans. The plan papers also provide criteria against which to measure performance.

ICAP COVER PAGE MODULE 34: ARITHMETIC STATEMENTS AND EFFICIENCY

ABSTRACT OF CONTENTS: This module covers the considerations related to the use of arithmetic statements --- the Data division definitions, the instructions operating on the data and the expected results of the various combinations. Through example, several precepts on the use of arithmetic statements are evolved.

OBJECTIVES OF MODULE: The student will understand the differences between decimal and binary operations --- the requirements and effects of each. As a result he will be able to make an intelligent choice when faced with the decision of choosing between them in his program.

STUDENT PREREQUISITES: An understanding of the COMPUTATIONAL clauses and their coding requirements.

HINTS FOR THE INSTRUCTOR: Emphasis should be placed on two things: Emphasizing the importance of Reference 34-1 and making the acceptance of these rules seem like obvious conclusions stemming naturally from the material in the Learning Plan.

ICAP STUDENT GUIDE MODULE 10 COMMUNICATION TECHNIQUES (LINKAGES)

ABSTRACT OF MODULE: Introduces the terminology of program linking and explains the processes involving the S/360 Operating System. Procedures necessary to effect linking is fully described.

RELATION OF MODULE TO YOUR PROFESSIONAL PROGRESS: Modularized programming is not a new concept. It is more feasible with the third generation computer ane its associated programming systems. Large programs can be placed and maintained in relatively inexpensive storage or they can be made up of many pieces (usually placed on secondary storage) and called when needed. It is important that the Professional Programmer understand and be able to use the conventions established to properly integrate the individual modules into a large executable program, and also to identify and use self contained re-enterable coded modules available for their use. Good communications among the programmers involved is essential for the effective use of these Communication Techniques.

HINTS TO STUDENTS: This module presents communication conventions and techniques. Modularized programming is a concept dependent on both need and efficiency. It should not be abused.

The following additional topics are suggested to fit Module 23 to your company:

1. Review of your programming organization.
2. Review of the programming cycle as it is carried out in your company.
3. Review of your position descriptions in the programming area with an interpretation of their meaning.
4. Review of your position descriptions in the systems area with an interpretation of their meaning.
5. Review of systems project and programming project management practices in your company.

This module will vary widely in its scope and depth from one group of students to another. It is imperative that you use a screening test based on the topics you nominate for the module. Failure of prospective students to show a sufficiently high level of understanding in a topic could result in one of these actions:

1. Remedial training prior to ICAP if only a few students show weakness.
2. Strengthening of the topic within the module if most students show weakness.
3. Elimination of students from the current ICAP class if they show numerous weaknesses which would take too long to correct before the start of ICAP.

Note that the Learning Plan advises adjusting the exact content of the module to class response. This deserves extra emphasis, because of the review nature of module. However, the module is also intended to introduce some new viewpoints for old concepts so that students will be better prepared to learn and accept the content of subsequent modules. For this reason it is suggested that you be very careful about rejecting the topic materials supplied with the module.

SESSION REPORT

COMMON - Chicago

Session Number **WED B4** Session Name **Petro-Chemical**

Chairman **G. Hertel**

Time **10:30 to 12:00 AM** Attendance (No.)

Speakers **J. R. Reese**

Synopsis of Meeting **"Petroleum Industry Graphics on the Geo Space DP 203 Platter"**

WED B4

ABSTRACT

Title: Petroleum Industry Graphics on the Geo Space DP 203 Plotter

by J. R. Reese

Because of its unusual versatility, the Geo Space DP 203 plotter fulfills the requirements of a wide range of computer graphics in the petroleum industry. This plotter is being actively used for creation of engineering drawings, piping isometrics, equilibrium charts, base maps, contour maps, seismic cross sections, and sonic logs. Potential applications include geologic cross sections, directional survey plots, critical path charts, process flow diagrams, and many others.

Two active applications, which require the variable intensity capability of the plotter and offer unusual long range opportunities, are sensor graphics and digital holography.

SESSION REPORT

COMMON - Chicago

Session Number WED B5 Session Name Applications COP

Chairman Larry Whelan

Time 10:30 Attendance (No.) 28

Speakers Len Schaider, IBM Chicago, "Control Optimization Program"

Bill Pease, W. Va. Pulp & Paper Co. "Simplex Optimization of Mixtures"

Synopsis of Meeting Mr. Schaider talked on the Control Optimization Program for the 1130 and 1800.

Mr. Pease talked on small computers and simplex optimization techniques for mixtures.

SMALL COMPUTERS AND SIMPLEX OPTIMIZATION TECHNIQUE FOR MIXTURES

W. A. Pease, Jr., W. G. Vardell

March 15, 1968

For Presentation at Chicago COMMON
April 10, 1968

Wed - B5

CHARLESTON RESEARCH LABORATORY

WEST VIRGINIA PULP AND PAPER COMPANY

CHARLESTON, SOUTH CAROLINA

ABSTRACT

This technique will optimize properties of any mix on a small computer, where the properties are functions of the ratios of the ingredients. Previous computer methods required plotters and larger core than utilized here. It is based on methods developed by Henry Scheffe and expanded by Gorman and Hinman.

Two things enable one to use a small computer. First, standard multiple regression techniques are used, simplifying programming and reducing core requirements. Second, an efficient program that generates the values for plotting the response surface has been written. One set of runs is needed for each property optimized.

Used as a screening device, the method is excellent for determining profitable areas of study.

TABLE OF CONTENTS

INTRODUCTION

In 1957, Henry Scheffe (2) pointed out that the properties of any mix could be predicted when these properties were functions of the ratios of the ingredients. His procedure is based on the use of a simplex as the plotting surface, within which the coordinates for any point sum to unity. Observed property values at specific points are plotted. These values are, in reality, projections of points existing on a response surface above the simplex. It is the equation of this surface that one attempts to derive.

Scheffe chose specific ratios so that the coefficients of the various terms in the mathematical model of the response surface could be readily calculated by hand. Later users (1) of this method continued to select Scheffe's points and method of coefficient determination even though the computation was now done by computer. Programs have also been published to plot the response contours on simplex coordinates using X-Y plotters (3).

Inflexible use of Scheffe's points and method of determining the coefficients has certain disadvantages, however. It requires more core than many small computers have available. It restricts the realm of study to a triangular area which may not be convenient. It also requires that the property must have a measurable value at each of the specific mix ratios on this triangular subspace. This often is not so.

Later users had overlooked that Scheffe reported his method as a form of polynomial regression. This fact was recognized by Dr. James Walker, Department of Mathematics, Georgia Institute of Technology, who was working with the authors, using a minimum configuration IBM 1620.

Use of multiple regression not only permits use of a smaller computer but gives much more flexibility to the experimenter since he is no longer tied to a given number of specific points. Any number above the minimum of evenly scattered points may be used. One may weight the equation for reliability in desired areas, expand the simplex area by addition of new points, or contract it for intimate exploration of the optimum. We are also permitted to examine subspaces other than triangular areas, though our coordinates remain triangular.

The present paper demonstrates the programming technique used to adapt this method of study to a small computer. The application of the method to a sample problem is also illustrated.

PROGRAMMING TECHNIQUE

Determination of coefficients on a small computer was no problem, since we were already using complex regression techniques at Westvaco. We were faced with the problem of generating the response points for plotting. The typewriter was slow, core was apparently insufficient to hold all the mathematical models desired, and economical segmentation was impractical without disk.

The typewriter problem was overcome by screening out points not on the simplex and outputting only effective points. Switching the order of some of the terms in the model equation was useful to avoid segmentation, since this dramatically reduced the number of decision statements required. This also made the generation of interaction terms faster. Core limitations were also met by using one general equation to cover all models, deleting undesired terms by reading in their coefficients as zero. The enclosed program will give points for the quadratic, special cubic, pure cubic, and special quartic math models (See Mathematical Models). There is room on the IBM 1130 for higher models with 8K.

Of particular interest from a programming point of view are the self-adjusting upper parameters for the nested DO-loops. The method is shown below:

```
      N5= -1                  (This makes the first subtraction a zero subtraction)
      DO 50 I= 1,N3
      N5= N5 + 1              (Zero on first pass, due to above)
      N6= N3 - N5
        .
        .
        .
      N7= -1                  (Same reason as N5 above)
      DO 50 J= 1,N6           (Note that N6 was defined in the next outer loop)
        .
        .
        .
   49 CONTINUE                (A separate CONTINUE is required for each inner loop)
   50 CONTINUE
```

In the above, N3 equals the number of intervals desired on the simplex border plus one. This is our chosen unit.

Variable N5 is the number of passes already made through the outermost loop.

Since the sum of the loop indices (I,J,K,L,M) must equal the chosen unit, here 11, the next loop index cannot exceed the value of (N3-N5). The same thing applies to the next inner loops, except that there is one additional index value over that of the previous outer loop to be subtracted in computing the current loop's upper index.

This results in continuously decreased upper limits for the inner loops as the outer loops progress. Inefficient passes made through the loops are minimzed.

Generally speaking the upper index for the ith inner loop equals ($N_{4+2i} = N_3 - N_{3+2i}$) and in each case N_{3+2i} is initialized to -1 prior to entry into ith loop.

This technique is only usable where the limits of the nested loops must sum to an integer, in this case 11, since the base for our unit is 10, and we need one more to get the zero values.

(The program is made to permit a constant being used where one forgets to repress it in the regression, and to make it flexible for other uses.)

Note how the zero levels are generated. A_i is made equal to the ith loop index minus one; thus when the index is one, A_i is zero. This is the reason N3 is N2, increased by one.

A more economical way of programming this from a core storage standpoint would be to put FORMAT 5 and the write loops into a subroutine as follows:

```
      SUBROUTINE PTWRT(MS,N1,BP,B,X)
C     MX = LOGICAL OUT PUT UNIT
      DIMENSION B(1),X(1)
      Y = 0.0
      DO 10 I = 1,N1
   10 Y = Y + (B(I)*X(I))
      IF (N1 - 3) 12,14,14
   12 Y = Y + (B(4)*B(4))
   14 Y = Y + B(42) + BP
      WRITE(MX,5)X(1),X(2),X(3),X(8),X(16),Y
    5 FORMAT (F9.2,3X,E16.9)
      RETURN
      END
```

Then a CALL PTWRT(3,N1,BP,B,X) statement could be substituted for statements 21, 26, 32, and 38. Statements after them, through their respective WRITE statements could be removed. These four calls would replace 19 of the present statements in the program. Expanding the WRITE statement in the subroutine would make it useable for more than five components.

In the program, A(I) is used to get the coordinates for the response points. Note how the zero points are generated. The reason for CONV in the program is that some experimenters like to work with unity, some ten, and some with one hundred as a base. This permits the printout of the coordinates to suit their particular desires.

AN in the program is our control to test CT, the constraint variable to insure that only points on the simple surface get printed.

METHOD OF APPLICATION

The steps employed in using this technique are as follows:

1. Define Study Area

The size of the area studied will depend on our knowledge of the system. For an exploratory run, make the simplex area of investigation large enough to detect unexpected optimums. Prior experience with a system would permit an optimization run within a restricted area of the simplex.

2. Select Model Equation

From the model equations shown in Appendix A, use the lowest order of equation expected to give a reasonable fit. Generally this order is one more than the number of points of inflection believed present on the response surface. The special cubic is a good place to start for systems of three or more components.

3. Select Experimental Points

From Table 1, determine the minimum number of points required for the model equation and number of components used. It is important to either replicate or exceed these minimums to insure significance for the derived equation.

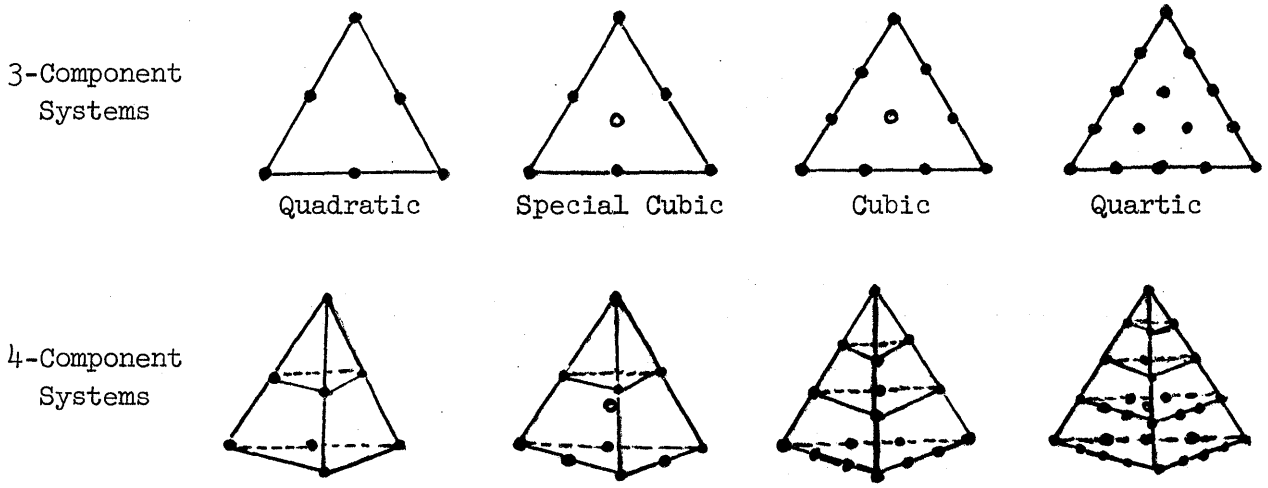
TABLE 1

Math Model Chosen / No. of Components	Quadratic		Special Cubic		Cubic		Special Quartic		Quartic		Special Quintic	
	P	T	P	T	P	T	P	T	P	T	P	T
2	3	2	-	-	4	2	-	-	5	2	-	-
3	6	3	7	3	10	3	-	-	15	4	-	-
4	10	4	11	4	20	4	31	5	38	5	-	-
5	18	5	19	5	40	5	58	6	84	6	97	6

The P-points are the experimental points. The T-points are the additional test-of-fit points, which must be made during initial runs to assure that the chosen mathematical model is sufficiently close to the true response equation. These points should also be run when switching to a different order equation in optimization runs.

It is important to distribute the points over the simplex study area with relatively equal spacing. For initial runs the Scheffe points shown in Figure 1 are mathematically very efficient and should be included where possible.

FIGURE 1



(The open points are centroids.)

Test-of-fit points should be equally spaced around the centroid of the field of study.

4. Making and Testing the Experimental Mixes

Mixes are prepared corresponding to the coordinates for the chosen points in the simplex area of study. For example, for a point whose (a, b, c) coordinates are (10, 60, 30), the mix would be 10% A, 60% B, and 30% C, where A, B and C may be either pure components or other mixtures.

Prepare the mixes in a randomized order and determine their properties.

5. First Computer Phase - Obtaining the Coefficients

The percentages of each component for each mix and the observed properties for that mix are fed into any standard multiple regression program. The following method is used:

a. Repress the constant normally generated by regression.
b. A separate run is made for each property, using the property response as Y.
c. Test-of-fit points are not included in initial runs.
d. Interaction terms are used according to the model equation chosen by the experimenter.

The coefficients derived are checked, using the test-of-fit points. If the variance is equal to or less than that for replication, the equation is acceptable. If not, a higher order equation must be used.

6. Second Computer Phase - Generation of Response Points

Upon achieving a satisfactory fit, program C-9655 (see Appendix B) is used to generate response points at the desired intervals. Input data is defined in the program listing. Again one run is made for each property.

The following comments regarding input should be helpful:

a. The value of "I" in B(I) is position of the accompanying variable in the program's equation, as listed in the heading. (B(I) is zero for unused variables.)
b. For more than three components, the experimenter must give the order desired for coordinate values. For example, if 0% D is to be the base of a four component tetrahedron, its coordinates must be fed in first. (i.e., D must be X1), then those for A, B and C.

The output of this program shows the simplex coordinates of the points of intersection of lines drawn from the interval points of one base to another, with the predicted value for the property at that intersection as the response.

7. Plotting the Response Points

For two component systems the responses are plotted on Cartesian paper using the Y-axis for the response and the X-axis for mix components.

With three components, triangular graph paper is used. The response value is written at the point represented by the mix coordinates.

For more than four components the simplex is examined like a three component mix at various interval levels, representing a constant value for the fourth and/or more component.

8. Interpretation for Optimum Area

Contour lines are drawn through points of equal response value. Examination of the resulting property map will reveal best areas for that particular property. Since it is unlikely that these will be in the same area for all properties, an added step is necessary to optimize the mix as a whole.

Superimposing different sets of contour lines on the same simplex will show where "optimums" for various properties come closest together. A new simplex is then drawn, with this point as the center, and reevaluated for better definition of the optimum area.

For example, if we had the following hypothetical situation:

There is to be a product made from a mixture of A, B and C. C is the most expensive ingredient, so we want to use as little as necessary. Properties 1 and 2 should be as high as possible, and property 3 must lie between 70 and 90. Using a special cubic model, the mixture has been evaluated, and the predicted responses have been plotted and superimposed as shown in Figure 2.

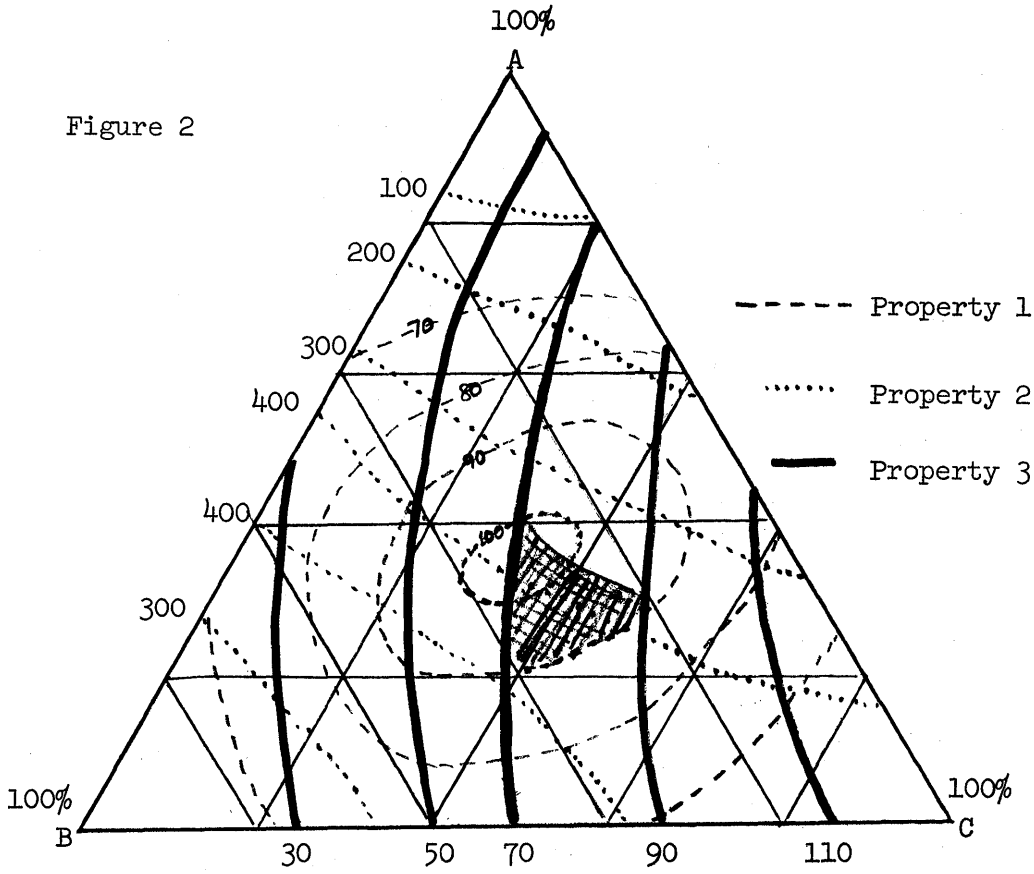
The shaded area in Figure 2 shows the apparent optimum area to be in the vicinity of 30% A, 30% B and 40% C.

The optimum area should then be defined more accurately by reducing the size of the simplex area under study. This has been done in Figure 3 by making a subspace simplex which encloses the apparent optimum. The new simplex is shown with its relationship to the old one represented by dotted lines. New points have been run with

additional mixtures made and evaluated in the immediate area of the optimum. Points from the earlier run may also be included if they fall within the area.

Figure 3 shows that reevaluation produced essentially the same optimum even though some of the porperties have changed slightly. These results, combined with a cost-analysis would give us a sound basis for choice of alternative product compositions.

Figure 2



Optimums

Property 1 - (Smallest Circle)

Property 2 - (Between 400 Lines)

Property 3 - (70 Prop 90)

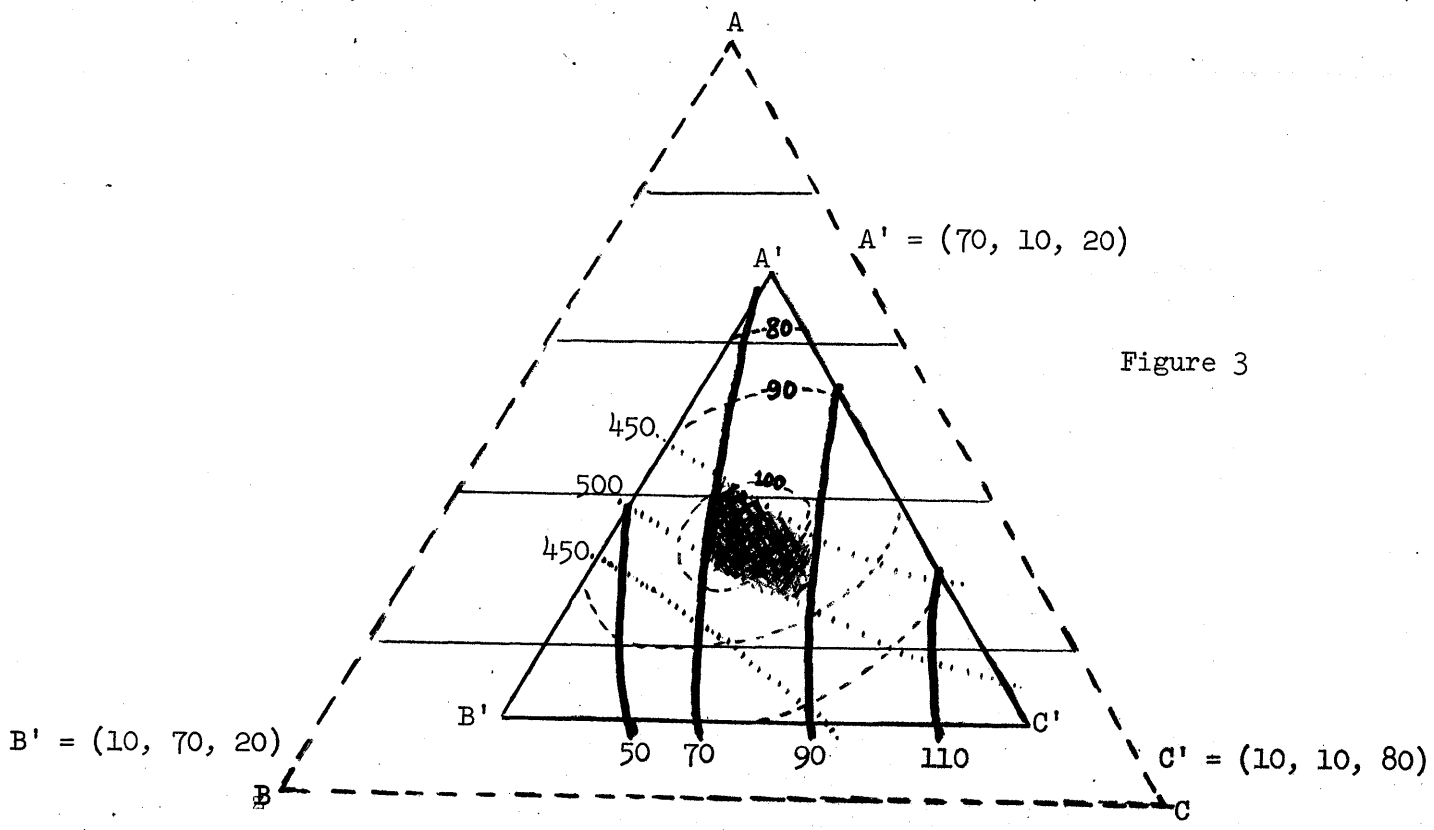


Figure 3

CONCLUSIONS

This technique can effectively contribute to the efficient screening of various alternatives involving mixtures. Afterwards, it may then be used to determine optimum formulations.

1620 time required is 15 minutes for a four-component system regression run. One run is made for each property to be optimized. The same computer needs 45 minutes for a response print out for the same system, using 10 as the interval on the simplex base line. Again one run is needed for each property. For the 1130 the times are 2 minutes and 5 minutes, respectively.

If a plotter is used, this will eliminate the time involved for manual plotting and the intermediate print-out of response points.

It is assumed in the above sample that the special cubic mathematical model is used.

Since many mix problems can be handled by this technique, it is felt that a general awareness of it will be profitable to user installations.

REFERENCES

1. Gorman, J. W. and Hinman, J. E.; "Simplex Lattices Designs for Multicomponent Systems," Technometrics 4:463-487(November 1962).

2. Scheffe, Henry; "Experiments with Mixtures," Journal of the Statistical Society. Series B, 20:344-369(1958).

3* Cruise, D. R.; "Plotting the Composition of Mixtures on Simplex Coordinates," Journal of Chemical Education, 43:30-31(January 1966).

4* Kurotori, I. S.; "Experiments with Mixtures of Components Having Lower Bounds," Industrial Quality Control, XXII(11):592-596 (May 1966).

*These were published after our report to Westvaco, upon which this paper is based, but are included for convenience of other users.

APPENDIX A

MATHEMATICAL MODELS USED

The following equations are the available models from which one may choose. The subscripts in the equations refer to the individual components. A multiple subscript indicates the presence, in the factor, of multiple components.

Models 1(a) and 1(b) will handle most situations.

1.(a) Quadratic: $y = \sum_{1\le i\le n} \beta_i X_i + \sum_{1\le i<j\le n} \beta_{ij} X_i X_j$ (n - No. of Components)

(b) Special Cubic: y = Same as 1.(a) $+ \sum_{1\le i<j<k\le n} \beta_{ijk} X_i X_j X_k$

2.(a) Cubic: y = Same as 1.(b) $+ \sum_{1\le i<j\le n} \alpha_{ij} X_i X_j (X_i - X_j)$

(b) Special Quartic: y = Same as 2.(a) $+ \sum_{1\le i<j<k<e\le n} \beta_{ijke} X_i X_j X_k X_e$

(The last term is used only when $n > 3$)

3.(a) Quartic: y = Same as 1.(a) $+ \sum_{1\le i<j<k\le n} \delta_{ij} X_i X_j (X_i - X_j)^2 + \sum \beta_{iijk} X_i^2 X_j X_k +$

$\sum \beta_{ijke} X_i X_j X_k X_e + \sum \beta_{ijjk} X_i X_j^2 X_k + \sum \beta_{ijkk} X_i X_j X_k^2 + \sum \alpha_{ij} X_i X_j (X_i - X_j)$

(b) Special Quintic: y = Same as 3.(a) $+ \sum_{1\le i<j<k<e\ m\le n} \beta_{ijke} X_i X_j X_k X_e + \sum \beta_{ijkem} X_i X_j X_k X_m X_e$

Any of the coefficients in the above equations may be zero, and it must be emphasized that the lowest order equation, consistent with a good fit, should be used. The choice of a more complex equation unnecessarily distorts the estimated response surface, and makes the optimum difficult or impossible to find.

There are no constants in the equation for we are examining differences* between the mixes, not what they have in common. It is important to remember this, since our correlation programs for the computer normally generate a constant. In this case, however, the constant must be repressed by the computer operator, otherwise the coefficients will not apply to the simplex. The simplex represents 100% of the mix, but if some portion is predetermined (for economic reasons, for example), then the simplex represents 100% of the flexible portion of the mix.

---

*Full advantage of this characteristic of simplexes is taken in examining systems with four or more components. The systems are examined at each level for three factors while the fourth and fifth factors are held constant at these levels. This permits examining a four dimensional simplex on two dimensional triangular pilots, which simplified interpretation.

```
      PROGRAM 9655-CHR
      GENERATION OF RESPONSE POINTS FROM SIMPLEX OF 2 TO 5 COMPONENTS,
         USING EITHER ABSOLUTE VALUES OR DIFFERENTIALS.
      MATH MODELS THROUGH SPECIAL QUARTIC INCLUSIVE.
      WRITTEN BY WM. A. PEASE,JR.                MARCH 2, 1967

      READ IN PARAMETER CARD CONTAINING N1,N2,N4,BP,NL,AND KB FIRST.
         THEN FOLLOW WITH NL HEADER CARDS TO LABEL AND SIX DATA CARDS
         CONTAINING B(I) VALUES FROM --
        B1(X1)+B2(X2)+B3(X3)+B4(X1X2)+B5(X1X3)+B6(X2X3)+B7(X1X2X3)+
        B8(X4)+B9(X1X4)+B10(X2X4)+B11(X3X4)+B12(X1X2X4)+B13(X1X3X4)+
        B14(X2X3X4)+B15(X1X2X3X4)+B16(X5)+B17(X1X5)+B18(X2X5)+B19(X3X5)+
        B20(X1X2X5)+B21(X1X3X5)+B22(X2X3X5)+B23(X1X2X3X5)+B24(X4X5)+
        B25(X1X4X5)+B26(X2X4X5)+B27(X3X4X5)+B28(X1X2X4X5)+B29(X1X3X4X5)+
        B30(X2X3X4X5)+B31(X1X2X3X4X5)+B32((X1-X2)X1X2)+B33((X1-X3)X1X3)+
        B34((X2-X3)X2X3)+B35((X1-X4)X1X4)+B36((X2-X4)X2X4)+
        B37((X3-X4)X3X4)+B38((X1-X5)X1X5)+B39((X2-X5)X2X5)+
        B40((X3-X5)X3X5)+B41((X4-X5)X4X5)+B42

      N1 = NO. OF DIFFERENT COMPONENTS (DIFFERENT X) IN MODEL.
      N2= NO. OF INTERVALS IN SIMPLEX BORDER (AT BASE).
      N4= BASE ON ORIGINAL SCALE. (UNITY,10,OR 100-- FOR PER CENT)
      BP= BASE FOR PROPERTY(SMALLEST VALUE OBSERVED-WHOLE NO.)
      NL= NO. OF LABELING HEADER CARDS
      B(I)= COEFFICIENTS IN MODEL EQUATION.
      KB= 1 FOR PURE CUBIC OR SPECIAL QUARTIC,= 0 IN ALL OTHER CASES.
      X(I)= EXPRESSION IN PARENTHESIS FOR CORRESPONDING B(I) IN MODEL.
      B(42)= CONSTANT IN EQUATION, WHEN USED.
      X(42)= 1 TO KEEP CONSTANT IN EQUATION,WHEN DESIRED.
      A(I)= BASIC COMPONENT RATIO BEFORE CONVERTING FOR BASE.
      CONV= CONVERSION FACTOR TO BRING COMPONENT RATIOS TO ACTUAL AMT.
      AN= CONSTRAINT ON SUM OF COMPONENT VALUES.
      CT= CONSTRAINT TEST TO KEEP POINTS WITHIN SIMPLEX MODEL

      DIMENSION B(48),X(48),A(5)
    1 FORMAT (3I4,F9.0,2I3)
    2 FORMAT (8F9.0)
    4 FORMAT (5X3H X1,6X3H X2,6X3H X3,6X3H X4,6X3H X5,7X9H RESPONSE )
    5 FORMAT (5F9.2,3X,E16.9)
    6 FORMAT (/I4,11H COMPONENTS,4X,I4,10H INTERVALS, I4 ,5H BASE /)
    7 FORMAT (72H
     1                    )
    8 FORMAT (1H1)
    9 FORMAT (30H WHAT IS NUMBER OF COMPONENTS.)
   20 FORMAT (40H NO. OF COMPONENTS EXCEEDS 5, TOO LARGE.//)

   10 WRITE (3,8)
      READ (2,1)N1,N2,N4,BP,NL,KB
      IF (N1) 14,15,16
   14 CALL EXIT
   15 WRITE (3,9)
      PAUSE 1
      GO TO 10
   16 IF (N1-5) 18,18,17
   17 WRITE (3,20)
      PAUSE 2
      GO TO 10
   18 DO 11 I=1,NL
      READ (2,7)
```

```
   11 WRITE (3,7)
C     INITIALIZATION OF VARIABLES
      DO 12 I=1,48
      X(I)= 0.0
   12 B(I)= 0.0
      N3= N2+1
      AN= N2
      AN4=N4
      CONV= AN4/AN
C
      WRITE (3,6)N1,N2,N4
      WRITE (3,4)
      DO 13 I= 1,41,8
   13 READ (2,2)B(I),B(I+1),B(I+2),B(I+3),B(I+4),B(I+5),B(I+6),B(I+7)
      X(42)= 1.
      N5= -1
      DO 50 I= 1,N3,1
      N5= N5+1
      N6= N3-N5
      A(1)= I-1
      X(1)= A(1)*CONV
      N7= -1
      DO 50 J= 1,N6,1
      N7= N7+1
      N8= N6-N7
      A(2)= J-1
      X(2)= A(2)*CONV
      X(4)= X(1)*X(2)
      IF (N1-3) 19,24,24
   19 CT= A(1)+A(2)
      IF (CT-AN) 50,21,50
   21 Y= 0.0
      DO 23 N= 1,2
   23 Y= Y+(B(N)*X(N))
      Y= Y+B(4)*X(4)+B(42)+BP
      WRITE (3,5)X(1),X(2),X(3),X(8),X(16),Y
      GO TO 50
   24 N9= -1
      DO 49 K= 1,N8,1
      N9= N9+1
      N10= N8-N9
      A(3)= K-1
      X(3)= A(3)*CONV
      X(5)= X(1)*X(3)
      X(6)= X(2)*X(3)
      X(7)= X(1)*X(6)
      IF (N1-4) 40,28,28
   25 CT= A(1)+A(2)+A(3)
      IF (CT-AN) 49,26,49
   26 Y= 0.0
      DO 27 N=1,7
   27 Y= Y+B(N)*X(N)
      Y=Y+B(42)+BP
      WRITE (3,5)X(1),X(2),X(3),X(8),X(16),Y
      GO TO 49
   28 N11= -1
      DO 48 L=1,N10
      N11= N11+1
      N12= N10-N11
      A(4)= L-1
      X(8)= A(4)*CONV
```

```
      DO 29 N=1,7
   29 X(N+8)= X(N)*X(8)
      IF (N1-5) 40,34,34
   30 CT= A(1)
      DO 31 N= 2,4
   31 CT= CT + A(N)
      IF (CT-AN) 48,32,48
   32 Y= 0.0
      DO 33 N= 1,15
   33 Y= Y+(B(N)*X(N))
      Y=Y+B(42)+BP
      WRITE (3,5)X(1),X(2),X(3),X(8),X(16),Y
      GO TO 48
   34 DO 47 M= 1,N12,1
      A(5)= M-1
      X(16)= A(5)*CONV
      DO 35 N= 1,15
   35 X(N+16)= X(N)*X(16)
      IF (N1-5) 40,40,17
   36 CT= A(1)
      DO 37 N= 2,5
   37 CT= CT+A(N)
      IF (CT-AN) 47,38,47
   38 Y= 0.0
      DO 39 N= 1,31
   39 Y= Y+(B(N)*X(N))
      Y=Y+B(42)+BP
      WRITE (3,5)X(1),X(2),X(3),X(8),X(16),Y
      GO TO 47
   40 IF (KB) 41,41,42
   41 GO TO (19,19,25,30,36),N1
   42 X(32)= (X(1)-X(2))*X(4)
      X(33)= (X(1)-X(3))*X(5)
      X(34)= (X(2)-X(3))*X(6)
      IF (N1-4) 25,43,43
   43 DO 44 LM= 1,3
   44 X(LM+34)= (X(LM)-X(4))*X(LM+8)
      IF (N1-5) 30,45,45
   45 DO 46 LM= 1,3
   46 X(LM+37)= (X(LM)-X(5))*X(LM+16)
      X(41)= (X(8)-X(5))*X(24)
      GO TO 36
   47 CONTINUE
   48 CONTINUE
   49 CONTINUE
   50 CONTINUE
      GO TO 10
      END

FEATURES SUPPORTED
 ONE WORD INTEGERS
 EXTENDED PRECISION
 IOCS

CORE REQUIREMENTS FOR
 COMMON       0  VARIABLES     352  PROGRAM    1318

END OF COMPILATION
```

APPENDIX C

REGRESSION AGAINST PROPERTY 5, SPECIAL CUBIC MODEL
09 19 66
PROB 02 15 VAR 00022 OBSER
(Includes Y)

DEP VAR = 05 (Position of Y in input sequence)
STD ERR Y.X = 37.74593
R SQUARED = .90220
SUM SQR RES = 05 .12822
IND VAR USED = 14

CONSTANT TERM = .00000

**VAR	*	COEFF	STD ERR	T RATIO	VARIABLE IN MODEL EQUATION	***
01	1	75.56728	37.70075	2.00439	X_1	2
02	2	277.43881	37.68200	7.36263	X_2	3
03	3	291.10783	37.70068	7.72155	X_3	8
04	8	186.13024	37.68184	4.93952	X_4	1
06	4	-28.26312	184.58658	-.15311	$X_1 X_2$	9
07	5	-228.92496	184.36437	-1.24169	$X_1 X_3$	10
08	9	-46.88036	184.58694	-.25397	$X_1 X_4$	4
09	6	228.86202	150.87728	1.51687	$X_2 X_3$	11
10	10	719.95126	991.68793	.72588	$X_2 X_4$	5
11	11	-79.79975	184.58658	-.43231	$X_3 X_4$	6
12	7	-514.56671	1022.15720	-.70570	$X_1 X_2 X_3$	14
13	12	-1955.80800	2771.44310	-.70570	$X_1 X_2 X_4$	7
14	13	-24.48535	1069.73940	-.02288	$X_1 X_3 X_4$ n	12
15	14	-989.20270	2784.63480	-.35523	$X_2 X_3 X_4$	13

When the T ratio is as low as that for Variable 14 shown above, the variable is considered as insignificant. (1.e. - Treated as though there coefficients were zero.)

Columns 2, 6 and 7 are not part of the printout.

---

*Order that coefficients are read into response program, normally
**Order that variables were read into multiple regression program. (05 = Dep. Variable)
***We would use the order on the far right, if we want to use 0% D as the base triangle in our plotting. (I.e. X1 = D, X2 = A, X3 = B, X4 = C)

RESPONSE FOR PROPERTY 1, SPECIAL CUBIC MODEL

4 COMPONENTS 10 INTERVALS 1 BASE

D X1	A X2	B X3	C X4	X5	RESPONSE
.0	.0	.0	1.0	.0	.1690330E+04
.0	.0	.1	.9	.0	.1630017E+04
.0	.0	.2	.8	.0	.1552197E+04
.0	.0	.3	.7	.0	.1456870E+04
.0	.0	.4	.6	.0	.1344036E+04
.0	.0	.5	.5	.0	.1213695E+04
.0	.0	.6	.4	.0	.1065846E+04
.0	.0	.7	.3	.0	.9004906E+03
.0	.0	.8	.2	.0	.7176276E+03
.0	.0	.9	.1	.0	.5172574E+03
.0	.0	1.0	.0	.0	.2993800E+03
.0	.1	.0	.9	.0	.1320748E+04
.0	.1	.1	.8	.0	.1238564E+04
.0	.1	.2	.7	.0	.1148577E+04
.	.	.	.	.	.
.0	.1	.9	.0	.0	.3002022E+03
.0	.2	.0	.8	.0	.9991548E+03
.0	.2	.1	.7	.0	.9648039E+03
.0	.2	.2	.6	.0	.8123553E+03
.	.	.	.	.	.
.0	.2	.8	.0	.0	.2976108E+03
.0	.3	.0	.7	.0	.7255498E+03
.0	.3	.1	.6	.0	.6287369E+03
.0	.3	.2	.5	.0	.5435310E+03
.	.	.	.	.	.
.0	.3	.7	.0	.0	.2916058E+03
.0	.4	.0	.6	.0	.4999332E+03
.0	.4	.1	.5	.0	.4103630E+03
.0	.4	.2	.4	.0	.3421044E+03
.	.	.	.	.	.
.0	.4	.6	.0	.0	.2821872E+03
.0	.5	.0	.5	.0	.3223050E+03
.0	.5	.1	.4	.0	.2496822E+03
.0	.5	.2	.3	.0	.2080758E+03
.0	.5	.3	.2	.0	.1974858E+03
.0	.5	.4	.1	.0	.2179122E+03
.0	.5	.5	.0	.0	.2693550E+03
.0	.6	.0	.4	.0	.1926652E+03
.0	.6	.1	.3	.0	.1466945E+03
.0	.6	.2	.2	.0	.1414449E+03
.0	.6	.3	.1	.0	.1769165E+03
.	.	.	.	.	.

for 2 components (at interval chosen) the printout would end here.

.0	.8	.2	.0	.0	.2103768ØE+Ø3
.0	.9	.0	.1	.0	.91676200E+02
.0	.9	.1	.0	.0	.1838902ØE+Ø3
.0	1.0	.0	.0	.0	.15399000E+Ø3
.1	.0	.0	.9	.0	.1415Ø846E+Ø4
.1	.0	.1	.8	.0	.13408807E+Ø4
.1	.0	.2	.7	.0	.12543193E+Ø4

← for 3 components, the printout would end here

.1	.0	.9	.0	.0	.3023786ØE+Ø3
.1	.1	.0	.8	.0	.1097748Ø2E+Ø4
.1	.1	.1	.7	.0	.1008617ØE+Ø4
.1	.1	.2	.6	.0	.91683288E+Ø3

.1	.1	.8	.0	.0	.3104194ØE+Ø3
.1	.2	.0	.7	.0	.82498584E+Ø3
.1	.2	.1	.6	.0	.73063186E+Ø3
.1	.2	.2	.5	.0	.64332980E+Ø3

.1	.2	.7	.0	.0	.3125983ØE+Ø3
.1	.3	.0	.6	.0	.59679728E+Ø3
.1	.3	.1	.5	.0	.50692535E+Ø3
.1	.3	.2	.4	.0	.43381006E+Ø3

.1	.8	.0	.1	.0	.12446268E+Ø3
.1	.8	.1	.0	.0	.2025718ØE+Ø3
.1	.9	.0	.0	.0	.1637174ØE+Ø3
.2	.0	.0	.8	.0	.11741224E+Ø4
.2	.0	.1	.7	.0	.1091176ØE+Ø4
.2	.0	.2	.6	.0	.10010235E+Ø4

.2	.0	.7	.1	.0	.4421388ØE+Ø3
.2	.0	.8	.0	.0	.3087384ØE+Ø3
.2	.1	.0	.7	.0	.9056174ØE+Ø3
.2	.1	.1	.6	.0	.8146881ØE+Ø3
.2	.1	.2	.5	.0	.7262562ØE+Ø3

.2	.1	.7	.0	.0	.3215495ØE+Ø3
.2	.2	.0	.6	.0	.6782714ØE+Ø3
.2	.2	.1	.5	.0	.5890636ØE+Ø3
.2	.2	.2	.4	.0	.5120577ØE+Ø3
.2	.2	.3	.3	.0	.4472535ØE+Ø3
.2	.2	.4	.2	.0	.3946588ØE+Ø3
.2	.2	.5	.1	.0	.3542498ØE+Ø3
.2	.2	.6	.0	.0	.3260504ØE+Ø3

.2	.3	.0	.5	.0	.4920844ØE+Ø3
.2	.3	.1	.4	.0	.4143031ØE+Ø3
.2	.3	.2	.3	.0	.3584281 2E+Ø3
.2	.3	.3	.2	.0	.3244594 6E+Ø3
			•		
			•		
			•		
.3	.5	.2	.Ø	.0	.2865399ØE+Ø3
.3	.6	.0	.1	.Ø	.1950824 8E+Ø3
.3	.6	.1	.Ø	.Ø	.2478805Ø E+Ø3
.3	.7	.0	.Ø	.Ø	.1984626Ø E+Ø3
.4	.Ø	.Ø	.6	.Ø	.7950476Ø E+Ø3
.4	.0	.1	.5	.Ø	.7100678Ø E+Ø3
.4	.0	.2	.4	.Ø	.6281795 2E+Ø3
			•		
			•		
			•		
.4	.4	.Ø	.2	.Ø	.2766815 2E+Ø3
.4	.4	.1	.1	.Ø	.2735495 6E+Ø3
.4	.4	.2	.Ø	.Ø	.3123280Ø E+Ø3
.4	.5	.0	.1	.Ø	.2329158Ø E+Ø3
.4	.5	.1	.Ø	.Ø	.2745076Ø E+Ø3
.4	.6	.Ø	.Ø	.Ø	.2234804Ø E+Ø3
.5	.Ø	.Ø	.5	.Ø	.6569350Ø E+Ø3
.5	.0	.1	.4	.Ø	.5786626Ø E+Ø3
.5	.Ø	.2	.3	.Ø	.5086314Ø E+Ø3
.5	.0	.3	.2	.Ø	.4468414Ø E+Ø3
			•		
			•		
			•		
.5	.4	.Ø	.1	.Ø	.2724314Ø E+Ø3
.5	.4	.1	.Ø	.Ø	.3037832Ø E+Ø3
.5	.5	.0	.Ø	.Ø	.2535950Ø E+Ø3
.6	.Ø	.Ø	.4	.Ø	.5531056Ø E+Ø3
.6	.0	.1	.3	.Ø	.4866902 8E+Ø3
.6	.Ø	.2	.2	.Ø	.4336658 4E+Ø3
			•		
			•		
			•		
.7	.3	.Ø	.Ø	.Ø	.3291146Ø E+Ø3
.8	.Ø	.Ø	.2	.Ø	.4482964Ø E+Ø3
.8	.Ø	.1	.1	.Ø	.4210442 8E+Ø3
.8	.0	.2	.Ø	.Ø	.4174824Ø E+Ø3
.8	.1	.Ø	.1	.Ø	.4010718 8E+Ø3
.8	.1	.1	.Ø	.Ø	.4075010Ø E+Ø3
.8	.2	.Ø	.Ø	.Ø	.3745196Ø E+Ø3
.9	.Ø	.Ø	.1	.Ø	.4473166Ø E+Ø3
.9	.Ø	.1	.Ø	.Ø	.4473706Ø E+Ø3
.9	.1	.Ø	.Ø	.Ø	.4250214Ø E+Ø3
1.Ø	.Ø	.Ø	.Ø	.Ø	.4806200Ø E+Ø3

Notice that only points whose coordinates add up to unity are printed by the computer. (The five pages from which this sample was taken were generated in 45 minutes on the 1620 computer.)

COP/1800

Control

Optimization

Program

For 1130 and 1800

One program for:

1130 Offline

1800 Offline

1800 Online

Manuals Available:

Users Manual	H20-0351
Application Description	H20-0208
System Manual	Y20-0110

Type II Program

PROGRAM ABSTRACT

General Purpose, Non-linear Optimization Program

Uses Sectional LP to Optimize Non-linear Model

Consists of 1130/1800 Assembler Programs

Operates under 1130 DMS or 1800 TSX

Uses Extended Precision FP Arithmetic

PROGRAM DESCRIPTION

User must supply model, IPSN, and limits

Model is set of equations relating X's and Y's

Model includes objective function to be optimized

Solution Method

Constraints and ObF. linearized about IPSN by calculating partials

Have matrix of partials

Result is: $\Delta Y = f(\Delta X)$

COP formulates and solves incremental LP (ΔY and ΔX)

Solution is set of ΔY's and ΔX's

After one loop, COP will re-linearize and optimize, etc.

Detection of Optimal

Optimum may be interior or exterior

Two convergence techniques

CVTI - reduce all LLIM
CVT2- selective reduction in LLIM

(Continued)

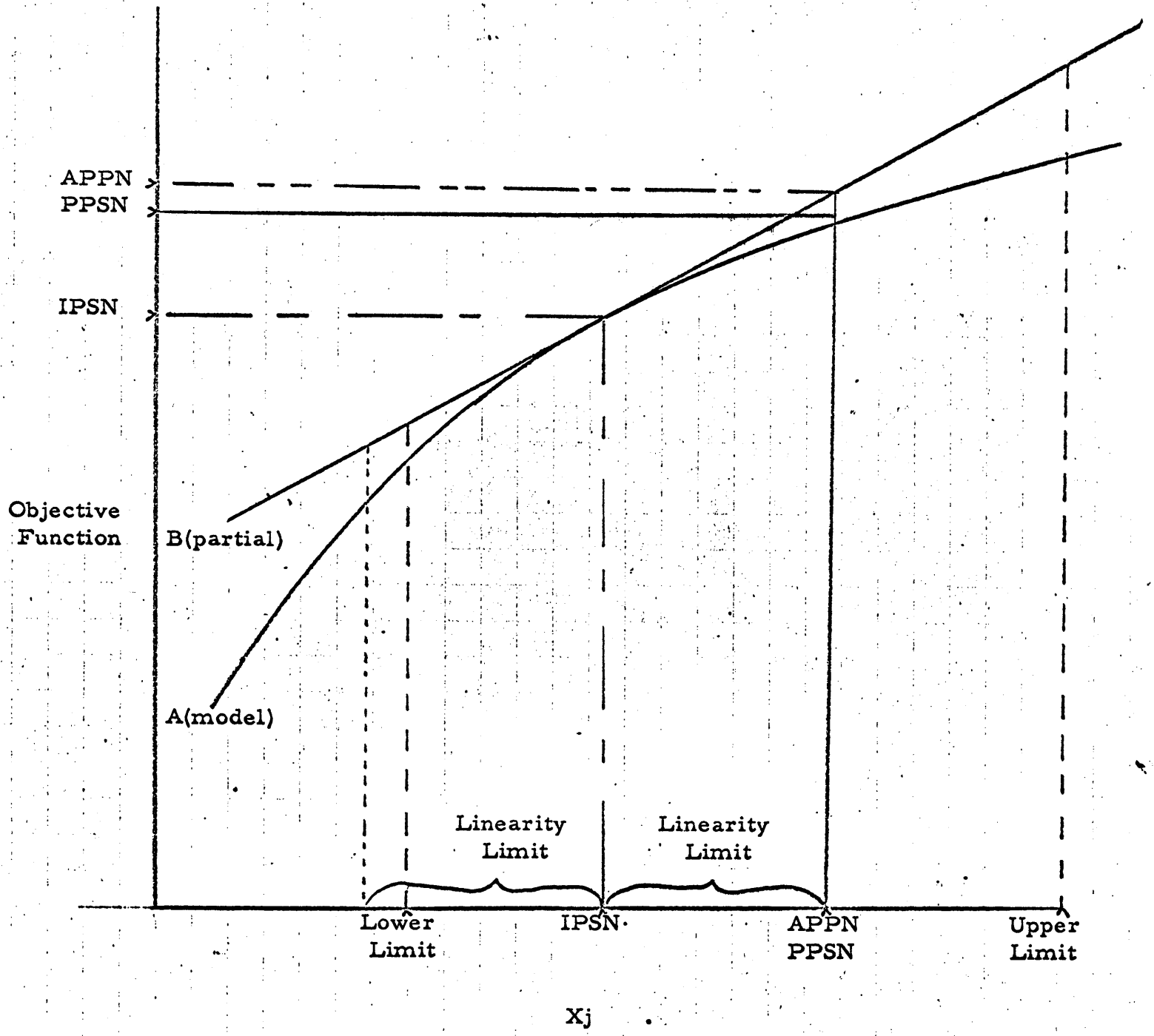
Infeasibilities

one or more variables outside limits

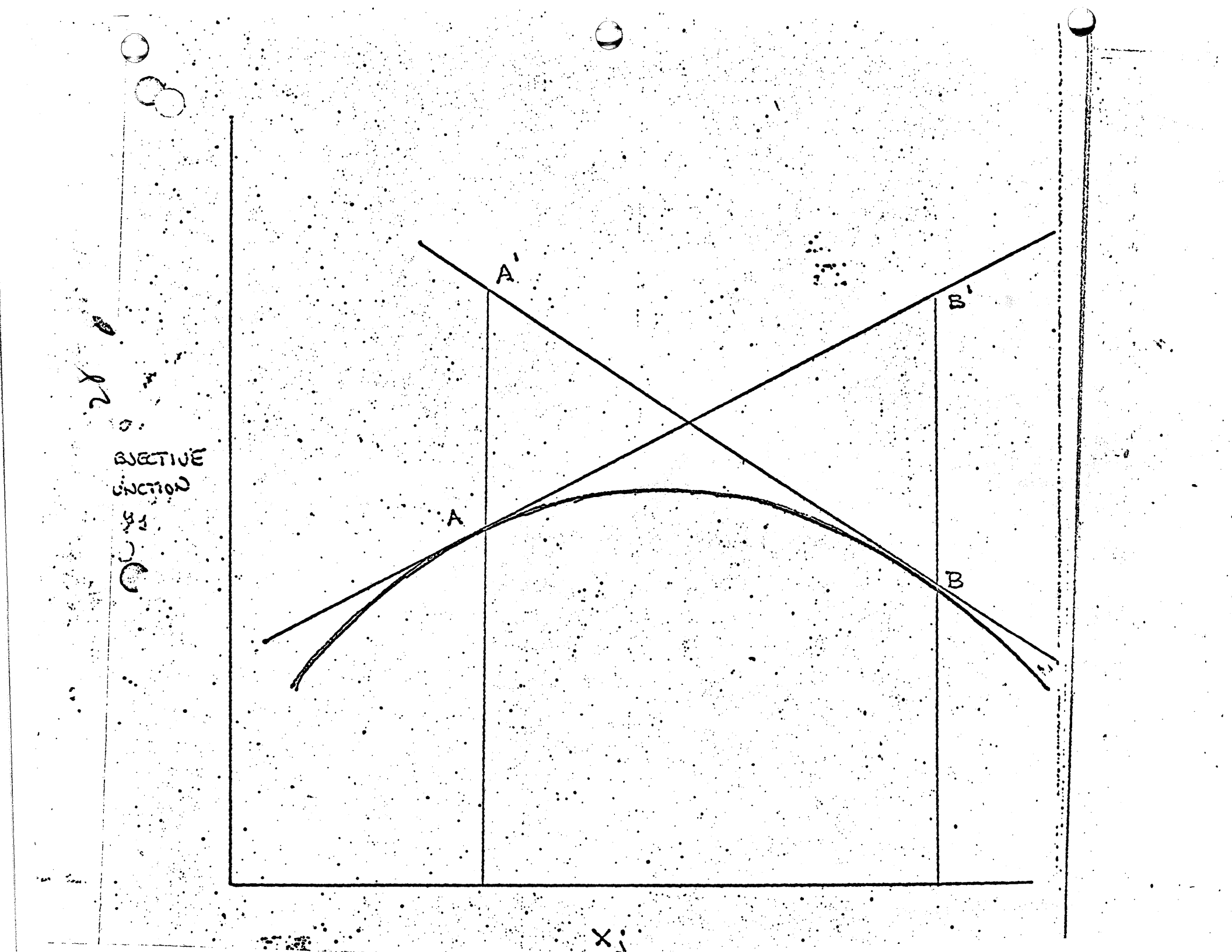
X: arbitrary move made

Y: dual algorithm employed

24



Graphical Representation of the Solution procedure


A'
B'
A
B
X1

SESSION REPORT

COMMON - Chicago

Session Number **WED B6** Session Name **1800 Plans and Soundoff**

Chairman **R. W. Forstrom**

Time **10:30 to 12:00** Attendance (No.) **80**

Speakers

Synopsis of Meeting

Plans for next meeting developed. Committees interested in:

(1) Documentation and Standards

(2) Data Acquisition

(3) MPX Review

Agreed to try to assemble committee or project interested in 1800-360 and 1130-360 Communication.

Agreed to establish informal news letter.

SESSION REPORT

COMMON - Chicago

Session Number WED B7 Session Name 360 Project

Chairman A. Ragsdale DOS Physical IOCS and FORTRAN

Time 10:30 to 12:00 AM Attendance (No.) 79

Speakers Mr. A. Saunders - Traveller's Research Center

Synopsis of Meeting Mr. Saunders presented his experiences and implementations while writing his own physical IOCS to handle tape and disk I/O under FORTRAN. Mr. Saunders has written his own REWIND, BACKSPACE, END FILE, READ And WRITE routines, etc. This has increased his performance significantly.

Alexander F. Saunders, Senior Analyst
Travelers Research Center, Inc.
250 Constitution Plaza
Hartford, Connecticut 06103

COMMON, Chicago, Illinois
360 Project, Session B7

April 22, 1968

"DOS Physical IOCS and FORTRAN"

Introduction

The reason for presenting the following material is to illustrate that user-written assembler language subroutines in support of DOS FORTRAN are not particularly formidable, and provide a means for performing unique functions faster and with greater flexibility and control. The use of physical IOCS routines for reading and writing binary tape records is the primary area of concern, although tape control functions, etc., will also be discussed. Whenever applicable, examples and performance comparisons with FORTRAN capabilities is provided. The routines to be described have been operational for over a year at the Travelers Research Center Computer Laboratory. The IBM 360/40 and 2402 mod II tape drives at that facility were used to obtain data for this report. The mode of operation was DOS, release 14.

1. The initial impetus for attempting a user-written tape I/O capability was provided by an historical requirement. Prior to obtaining its own computer, TRC had used the IBM 7090 series facilities at The United Aircraft Research Laboratories in East Hartford, Connecticut, and the I/O packages developed by their personnel. To maintain a minimum amount of re-programming effort, similar subroutines were desirable under our own system capability. It was also necessary to generate routines which (1) executed faster than their FORTRAN counterparts, (2) provided greater user control, (3) were easy to use, and (4) had a low core requirement.

2. Logical IOCS did not lend itself adequately to generalization, and, therefore, physical IOCS became the chosen method. As a brief description, PIOCS allowed performance of non-data operations and control of the transfer of data to and from external devices via four supervisor resident routines: start I/O, I/O interrupt, channel scheduler, and device error.

3. Eight subroutines were written to accomplish the specified control and read-write capability. Figure one contains a description of each. At this time the routines are general only to TRC users, and would probably need some revision to perform satisfactorily at another installation.

4. Figure two illustrates comparative execution times with FORTRAN, example one a subroutine used to measure the execution times, and example two demonstrates a typical rewind subroutine. Figure four, though not based on an I/O oriented procedure, was included to encourage assembler language programming in computational areas. It is especially attractive for production type programs when heavy FORTRAN indexing is involved.

5. After each routine having a response indicator as one of its arguments, a computed go to is usually employed to perform branching to the necessary statement number. Branch time was included in collecting data for figures two and three.

6. There are three factors involved in the performance differences indicated in figure two. FORTRAN generates 63 word data records preceded, in release 14, by two control words. Release 13 on down required one leading control word. Computation and testing of the control words and the movement of data to and from buffer areas also add considerably to the execution time. By using the PIOCS subroutines these time consuming functions were avoided. The advantage gained by writing one long record rather than a series of shorter ones may be demonstrated by referring to figure three. The time required to write a single 5000 word record is 0.35 seconds. If, instead,

one-hundred 50 word records were written, the time required jumps to 2.00 seconds. To write 5000 words with FORTRAN would take approximately 4.83 seconds.

7. At this time the core required by all eight routines is 850_{16}. They are presently being re-written to provide facility for handling records up to 16383 words in length, and to sense for not-ready and file protect status. Record size limitations now are within the range of 4 to 8191 words. Control routines RWD, WEF, and UNL are being changed to accept a variable number of arguments, thereby allowing the programmer to request multiple operations in one call statement. Incidently, the number of re-tries in event of I/O error has been programmed at five times reading, and fifteen writing. This has proven quite satisfactory.

8. Discussion to this point has been almost exclusively about magnetic tape I/O. Other external I/O devices (typewriter, printer, and card reader-punch) have also been programmed to perform unique functions. For instance, re-read type subroutines, typewriter communication, and job accounting procedures, to mention a few, have been implemented using physical IOCS.

9. In addition to FORTRAN, the routines described have linked and executed properly with PL/I and assembler language programs. The biggest problem in commencing PIOCS programming is in gathering the necessary material. Hopefully, example two, the references provided, your SE, and considerable reading and imagination will bridge the gap for interested programmers. The investment is well worthwhile.

References:

Source	Content
IBM System/360 Disk Operating System Supervisor and Input/Output Macros C24-5037-2	CCB (command control block) EXCP (execute channel program) WAIT
IBM System/360 Disk and Tape Operating Systems Assembler Language C24-3414-4	CCW (channel command word)
IBM System/360 Principles of Operation A22-6821-3	General
IBM System/360 Disk and Tape Operating System FORTRAN IV Specifications C24-5014-0	Subroutine Linkage
IBM System/360 Reference Data (Green Card) X20-1703-3	Channel Command Codes Channel Address Word Channel Command Word Channel Status Word
IBM System/360 Component Design 2400-Series Magnetic Tape Units and 2816 Switching Unit A22-6866-3	Sense Information
Programming the IBM System/360 Appendix F Staff of Computer Usage Co. John Wiley and Sons, Inc.	I/O Device Responses General Information

PIOCS Subroutine	FORTRAN Counterpart	Use
CALL RBT (A, NWDS, NT, NE)	READ (NT) (A(I), I=1, NWDS)	Read a physical record a known length or the first N words of a longer record.
CALL RBTX (A, NWDS, NT, NE)	-none-	Read a physical record of unknown length.
CALL WBT (A, NWDS, NT, NE)	WRITE (NT) (A(I), I=1, NWDS)	Write a physical record.
CALL RWD (NT)	REWIND NT	Tape rewind.
CALL UNL (NT)	-none-	Tape rewind and unload.
CALL WEF (NT)	END FILE NT	Write a tape mark.
CALL RSKP (NCT, NT, NE)	READ (NT) BACKSPACE NT	Skip forward or backward a given number of records. If an end of file is sensed, skipping is terminated regardless of the requested count.
CALL FSKPX (NCT, NT, NE)	-none-	Skip forward or backward a given number of files.

Symbol Definition

A Name of array or variable where data is to be read into or from in consecutive fashion
NWDS Number of words to be read or written. This is a returned value for RBTX.
NCT Number or records or files to skip. A negative value results in backwards motion.
NT FORTRAN unit assignment. Values of 10 through 14 are acceptable.
NE Response Indicator.

Table of Responses, NE

Routine \ Response	1	2	3	4	5
RBT	OK	end-of-file detected	I/O error	*	Wrong length record (longer or shorter
RBTX	"	"	"	*	
WBT	"	end-of-tape detected	"	*	
RSKP	"	end-of-file detected	*		
FSKPX	"	*			

* Word count (NWDS) or unit selected (NT) in error

FIGURE I

K+E 10 X 10 TO THE INCH 46 0780
7 X 10 INCHES MADE IN U.S.A.
KEUFFEL & ESSER CO.

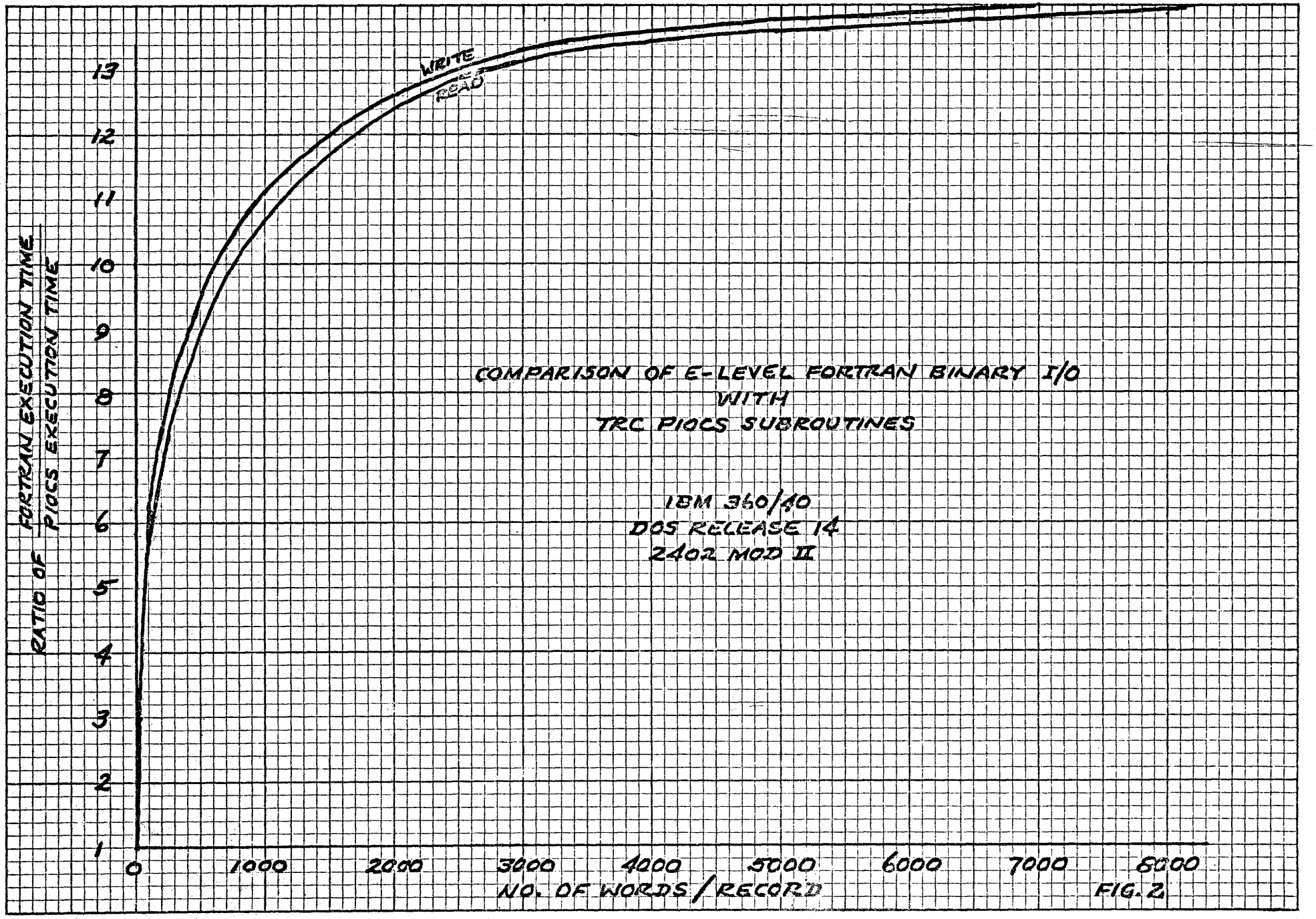


FIG. 2

K+E 10 X 10 TO THE INCH 46 0780
7 X 10 INCHES MADE IN U.S.A.
KEUFFEL & ESSER CO.

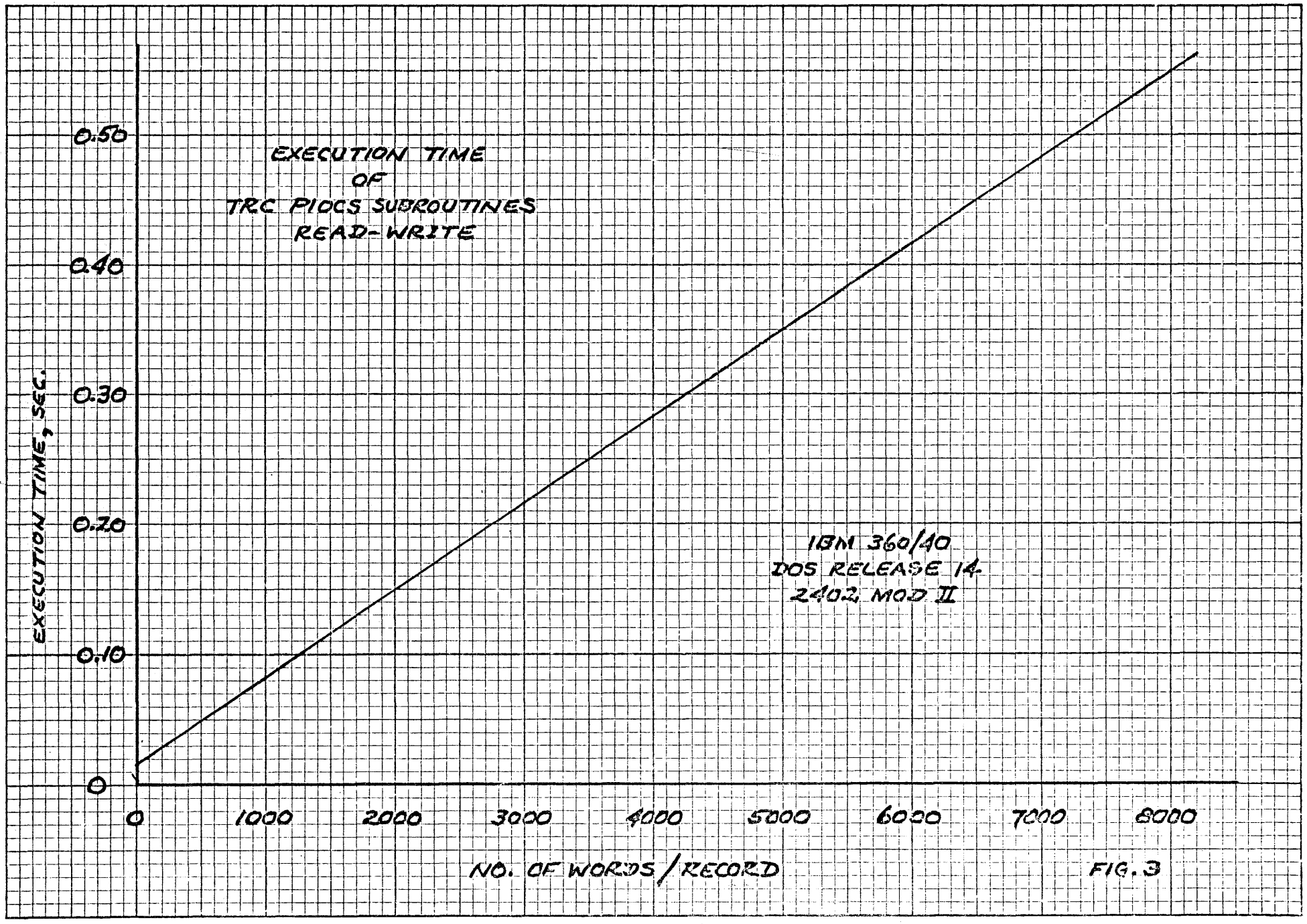


FIG. 3

MADE IN U.S.A.
KEUFFEL & ESSER CO.

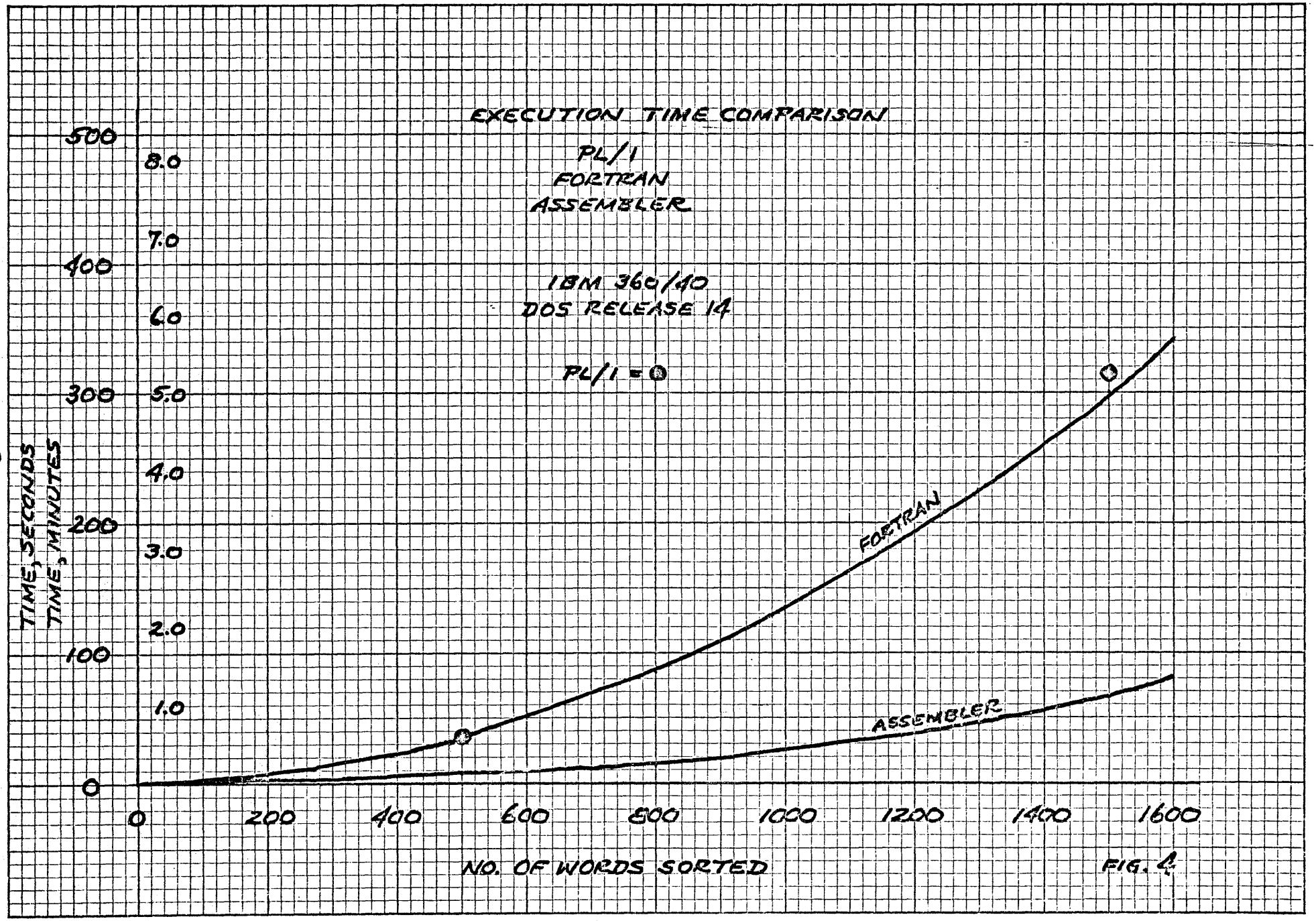


FIG. 4

```
   LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

000000                              1 TRCTSEC  START 0                CALL TIME(IT300)
                                    2          ENTRY TIME
000000                              3          USING *,15
                                    4 TIME     SAVE  (14,12)---
                                    5+* 360N-CL-453  SAVE      CHANGE LEVEL  2-0
000000 90EC D00C              0000C 6+TIME     STM   14,12,12+4*(14+2-(14+2)/16*16)(13)
000004 5851 0000              00000 7          L     5,0(1)
                                    8          GETIME TU              GET CLOCK TIME IN 1/300 SEC
                                    9+* 360N-CL-453  GETIME    CHANGE LEVEL  2-0
000008 9801 0050              00050 10+        LM    0,1,80 GET TIMER VALUE IN SEC/76800
00000C 8800 0008              00008 11+        SRL   0,8 TIMER IN SEC/300
000010 1F10                         12+        SLR   1,0 TIME OF DAY IN SEC/300
000012 5015 0000              00000 13         ST    1,0(5)
                                    14         RETURN (14,12)
                                    15+* 360N-CL-453  RETURN    CHANGE LEVEL  2-0
000016 98EC D00C              0000C 16+        LM    14,12,12+4*(14+2-(14+2)/16*16)(13)
00001A 07FE                         17+        BR    14
                                    18         END
```

01

EXAMPLE 1

```
   LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT

000000                             1 TRCRWDE  START 0               CALL RWD(NT)
                                   2          ENTRY RWD
000000                             3          USING *,15
                                   4 RWD      SAVE  (14,12)
                                   5+* 360N-CL-453  SAVE      CHANGE LEVEL  2-0
000000 90EC D00C            0000C  6+RWD      STM   14,12,12+4*(14+2-(14+2)/16*16)(13)
000004 58C1 0000            00000  7          L     12,0(1)         FORTRAN UNIT ADDRESS
000008 48BC 0002            00002  8          LH    11,2(12)        FORTRAN UNIT ASSIGNMENT
00000C 4BB0 F03C            0003C  9          SH    11,=H'3'        CONVERT TO SYS NUMBER
000010 42B0 F027            00027 10          STC   11,CONTROL+7    STORE IN CCB
                                  11          EXCP  CONTROL         EXECUTE REWIND
                                  12+* 360N-CL-453  EXCP      CHANGE LEVEL  2-0
000014 5810 F038            00038 13+         L     1,=A(CONTROL)
000018 0A00                       14+         SVC   0
                                  15          RETURN (14,12)
                                  16+* 360N-CL-453  RETURN    CHANGE LEVEL  2-0
00001A 98EC D00C            0000C 17+         LM    14,12,12+4*(14+2-(14+2)/16*16)(13)
00001E 07FE                       18+         BR    14
                                  19 CONTROL CCB   SYS000,TAPE
                                  20+* 360N-CL-453  CCB       CHANGE LEVEL  2-4
000020 0000                       21+CONTROL DC    XL2'0' RESIDUAL COUNT
000022 0000                       22+        DC    XL2'0' COMMUNICATIONS BYTES
000024 0000                       23+        DC    XL2'0' CSW STATUS BYTES
000026 01                         24+        DC    AL1(1) LOGICAL UNIT CLASS
000027 00                         25+        DC    AL1(0) LOGICAL UNIT
000028 00                         26+        DC    XL1'0'
000029 000030                     27+        DC    AL3(TAPE) CCW ADDRESS
00002C 00                         28+        DC    B'00000000' STATUS BYTE                    2-
00002D 000000                     29+        DC    AL3(0) CSW CCW ADDRESS
000030 0700003000000001           30 TAPE    CCW   7,*,X'00',1
                                  31         END
000038 00000020                   32                =A(CONTROL)
00003C 0003                       33                =H'3'
```

EXAMPLE 2

SESSION REPORT

COMMON - Chicago

Session Number 360 MON G1 and D1 Session Name 1130 PLAN

Chairman P. J. Woodrow

Time 1:30 PM and 3:30 PM Attendance (No.) 139

Speakers Mr. Jack Sams and Mr. Dick Weber - IBM

Synopsis of Meeting Mr. Sams devoted the first session to an overview of PLAN (Problem Language Analyzer), presenting the basic reasons and principles for this system. He announced that a new version of PLAN would be released in January 1969 whick would support Version 2 of the Disk MONITOR System. He also announced that PLAN would be available in the near future on the IBM 360 under BTAM AND QTAM. The second session was primarily devoted to questions from the users to Mr. Sams (principal designer of PLAN) and Mr. Weber (principal IBM user of PLAN for DPS (Data Presentation System)). A clear majority of those present indicated a desire to use PLAN and a subcommittee (of the 1130 Project) under Mr. Woodrow was formed to thoroughly investigate and report on PLAN.

The IBM Problem Language Analyzer

Based on successful experience with three current application programs, IBM has announced that a new development support package for problem-solving applications will be available January 30, 1969. This set of programs is known as the IBM Problem Language Analyzer (PLAN). At the same time, three more engineering applications based on PLAN were announced.

PLAN supports both the IBM 1130 and System/360. It interfaces with three monitor/operating systems (1130 Monitor vII, DOS, OS/360) to provide a uniform set of application services for FORTRAN-oriented installations. Using these services fully allows application modules to be exchanged at source level across system boundaries.

It should be emphasized that PLAN simply adds to the installation options. It is not a separate or dedicated system. The PLAN programs create an interactive environment within a standard batch monitor. There is no unusual effect on existing batch operations. PLAN operates ~~on~~ a JOB.

Within the PLAN JOB, PLAN provides several services:

- It loads program modules dynamically as directed by the user of the system (This control is usually indirect, implied by a problem statement).

- It creates and maintains problem language description tables. These allow installations to have their own POL's without writing compilers or interpreters for each different language and system.

- It uses the installation defined table to decode and execute each user's problem descriptive statements, one at a time, or in batches.

To programmers supporting PLAN-based applications, PLAN provides a library of over 60 subroutines that produce the same effect in different operating systems.

These perform program linkage, data management, and utility functions. Using these subroutines, and following the techniques suggested by PLAN allows an installation to produce modular programs that can be used in a dynamic loading environment.

Using these subroutines also lowers the total programming effort and maximizes the probability that a module can be reused in an unspecified future application.

The main theme of these subroutines is their emphasis on execution time definition of attributes that have usually been constants at source time. (Data file I/O record structure, program linkage, communication between user and program, etc.)

Application design and implementation under PLAN is quite different and much simpler than usual if the end-user of the application is honestly to be given a range of capability, or any degree of control at execution time.

Ordinarily, extensive pre-planning is required to identify the possible choices and control paths that are to be supported or prohibited. The complication factor is at least (N * N-1) where N is the number of options. It can approach (N!). It usually exceeds the programmers skill and foresight if N > 10. Adding or changing anything re-opens Pandora's box. Even successful efforts tend to age quickly and perish at machine boundaries.

PLAN assumes that the end-user will be subject to certain disciplines; but that he has the fundamental privilege to order sequences of processing that were not necessarily anticipated. He can make errors, as he can make errors with a calculator. He can also succeed in solving many unique, one-time problems that cannot be anticipated.

To allow the end-user to exercise control, PLAN supports installation definition of their own free format, problem oriented statements. Any meaningful combination of statements can then be processed by a PLAN JOB, one at a time or in batch mode, to give users results. Note that PLAN execution does not involve either compiling or interpreting computer programs. Input statements, like the higher level monitor control statements, direct execution of previously compiled and core imaged or link edited modules. These modules are obtained by standard FORTRAN or Assembler methods and have to be available before the PLAN JOB is executed.

Extending of the installation's PLAN language (collection of statement definitions) is a simple matter of ADDing or DELeting individual statement definitions. A single definition step serves all future users.

Extending an application under PLAN is a process of adding or deleting functional modules of code implied by POL statements. Changing, recompiling, and link-editing related code is not required. Growth and testing can be incremental, continuing, and far less costly.

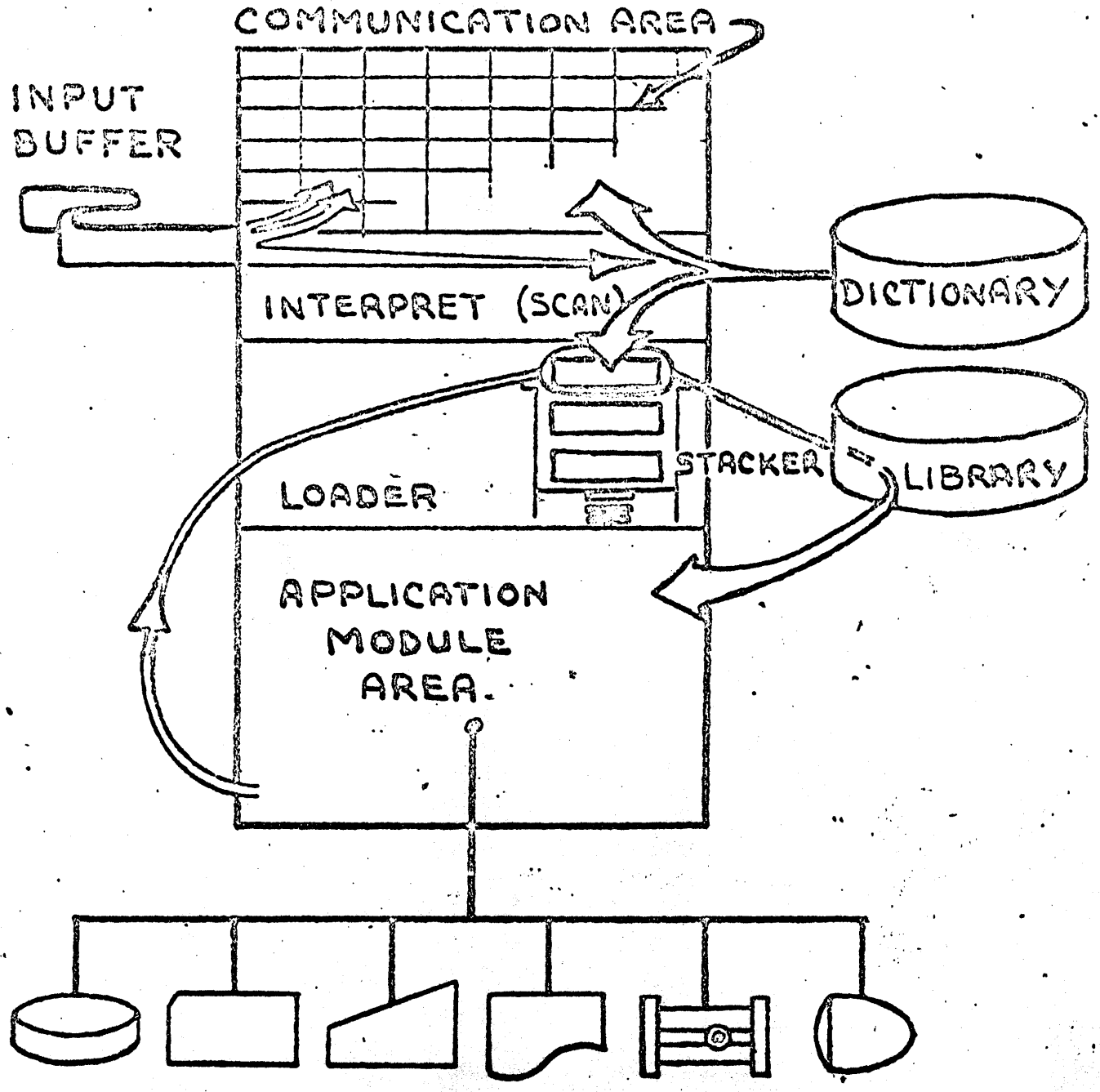
The attached diagram is a schematic of PLAN operation on a simple logical machine (real or a partition or a time slice). It does not reflect real core allocations. In practice, PLAN occupies 2560 bytes of storage in the problem program COMMON area. Otherwise, the problem space is as large as it normally would be in FORTRAN operations.

A second attachment indicates the general scope of the PLAN program. The subroutine argument lists are not given; and these calls should not be included in current programs. The users manual of the 1130 Data Presentation System provides programming detail on the subset now in use on the 1130 under Monitor I.

In summary, PLAN uses and augments three existing batch monitors to provide a problem-solving application environment. It promotes modularity and economy of programming. Most importantly, it offers a technique for open-ended development of application languages and programs at the installation level. Using PLAN techniques can extend profitable computer usage into the everyday world of problems we now spend a half day solving by hand, without requiring journeymen to become programmers.

Last, but not least, PLAN suggests and supports a new division of effort and responsibility among coders, system analysts, and users of problem-solving systems.

PLAN (STAND-ALONE PROCESSOR)



I/O & STORAGE DEVICES CONTROLLED BY "EXECUTE" PHRASE

UTILITY CALLS

IOCS	-	PLAN System Device Assignment
ERROR	-	Abort Statement with Diagnostic
ERRET	-	Continue after Diagnostic
ERRAT	-	Continue after Diagnostic, set Error Latch
ERREX	-	Abort Module after Diagnostic
ERLST	-	Print Pending Error Messages
INPUT	-	Give Caller the Text of the Last Input Statement
NDEF	-	Accumulator -,0,+ as Argument is False, True, Real
TRUE	-	Set True
FALSE	-	Set False
PARGO	-	Move Argument List to COMMON
PARGI	-	Collect Arguments from COMMON
BREAK	-	Convert A4 to Four Right Justified Integers
PACK	-	Extract Byte
PUNPK	-	Mask in Byte

LOADER CALLS

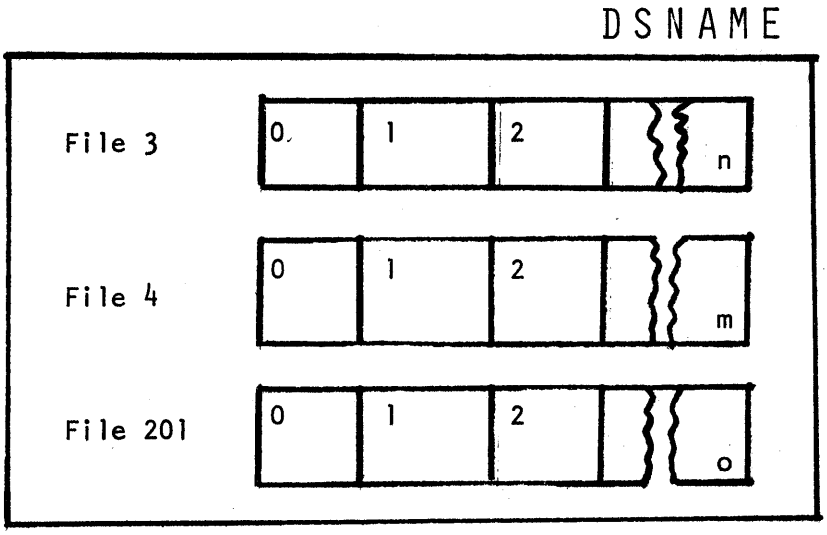
LEX	-	Load and Execute a Module
LIST	-	Add To/Remove
LISTB	-	Add/Remove from Bottom of Stack
LCHEX	-	Roll Out Current Module, Load and Execute New Module, Return to Old Module on Signal
LOCAL	-	Load and Enter Another Module, Return to Caller
LRET	-	Return to Calling Module
LNRET	-	Cancel Pending Returns
LREPT	-	Repeat Current Statement
PUSH	-	Execute a PLAN Statement from Core, rather than from the Input Stream

MODULES

PLAN	-	LOADER
PSCAN	-	INTERPRETER
PHRAS	-	LANGUAGE DEFINER
PERRS	-	ERROR PROCESSOR
FIOCS	-	INPUT/OUTPUT DEVICE SELECTION
PFIND	-	DISK SPACE ALLOCATOR
PSRTA	-	DISK SORT
PMRGA	-	DISK MERGE
PSTSV	-	STATEMENT SAVE
PTDMP	-	LANGUAGE TABLE DUMP
PFDMP	-	FILE PUMP
PPDMP	-	PERMANENT FILE DUMP
PCDMP	-	CORE DUMP
PIDMP	-	INPUT TRACE
PDIAG	-	DIAGNOSTIC MESSAGE MAINTENANCE

DATA MANAGEMENT CALLS

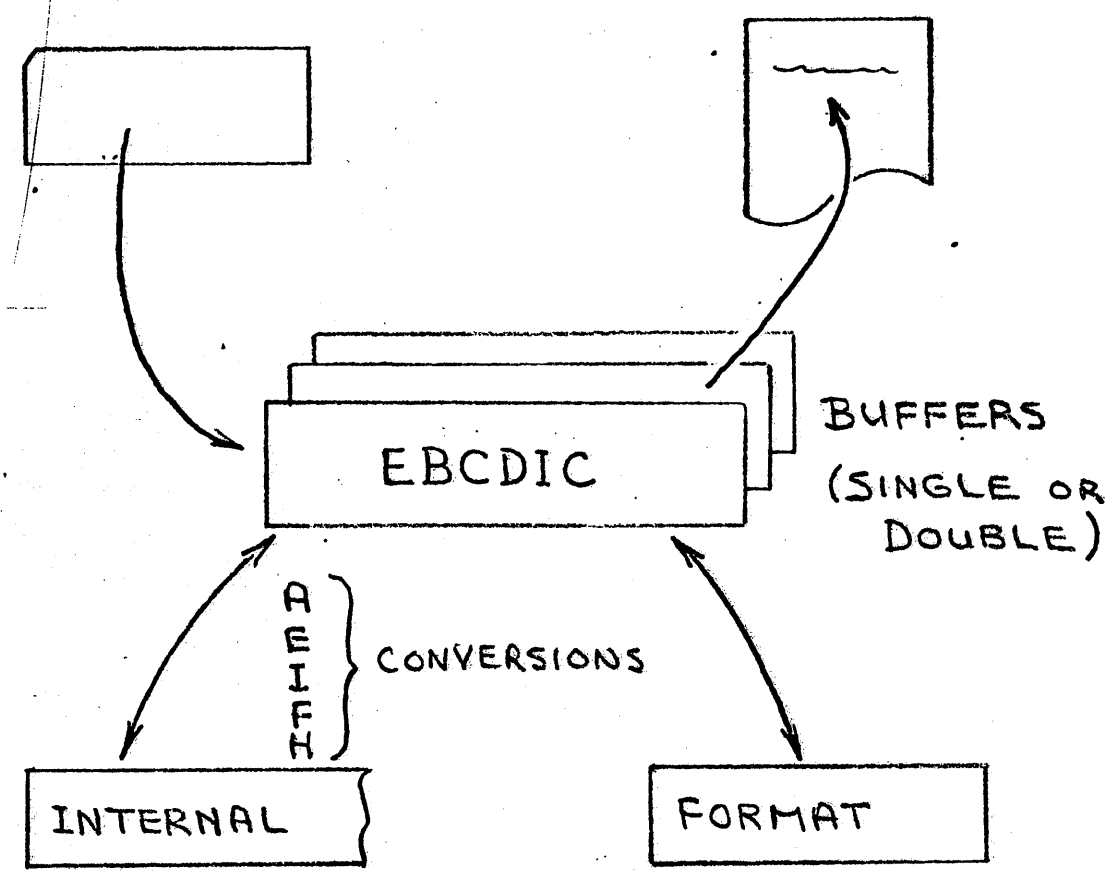
FIND	- Initialization
READ	- Disk to Core Array
WRITE	- Core Array to Disk
RELES	- Free Disk Space
PSORT	- Disk Sort in Place
PMERG	- Disk Merge in Place
GDATA	- Initialize for Files Written Outside PLAN
WDATA	- Core/Disk for GDATA Files
RDATA	- Disk/Core for GDATA Files
PFSPC	- Test for Available Space



UTILITY CALLS - (CONTINUED)

PCOMP	-	ARRAY COMPARISON
PHTOE	-	HEXADECIMAL TO EBCDIC
PBTST	-	BIT TEST ROUTINE

I/O TO U.R. DEVICES



ARRAYS IN PROGRAMS

- ALL ARGUMENTS ARE VARIABLES
- 30 SUBROUTINES

IMPLEMENTATION OF PLAN APPLICATION

OBJECTIVE: Make it easy for the man with the problems to describe them to the computer.

THREE GROUPS ARE INVOLVED:

USERS: Tell analysts what words and phrases are required. Define commands and data terms precisely.

ANALYSTS: Define words and phrases to the PLAN System. Determine what additional programmed functions are needed in the library. Specify details of required programs.

PROGRAMMERS: Write programs as specified by the analysts and add them to the PLAN library.

THEN: Users describe their problems in the language they have defined.

SESSION REPORT

COMMON - Chicago

Session Number WED C2 Session Name Education Project

Chairman _____

Time 1/30 to 3:30 PM Attendance (No.) _____

Speakers S. Lee

Synopsis of Meeting Dr. Lee presented a paper entitled, "The SL/1 Compiler for the IBM 1130"

WED C2

THE SL/1 COMPILER FOR THE IBM 1130

Authors: M. E. Jackson (IBM Canada)
E. S. Lee (University of Toronto)
P. I. P. Boulton (University of Toronto)

Because of the growth in the use of data processing equipment in recent years and because of the shortage of trained personnel, the Department of Education of the Province of Ontario is encouraging the teaching of data processing in the high schools of the Province. Some schools have installed keypunches and unit record equipment but during the last three or four years a number of small computers have been installed for teaching purposes. In some instances, a large school will acquire its own computer; in others, the local school board will obtain a machine to be used by several schools in the area. These machines are 1620s, 1130s or machines of comparable size, although some school boards are using System/360 Model 30s for both administration and teaching. Data processing is being taught as a vocation to students in the commerce stream and as a tool to students in the science and arts stream. In 1965, the Department of Education, in an effort to promote uniformity and standards, published curricula for teaching data processing and asked a committee of university professors for recommendations on a suitable programming language.
(SLIDE 1)
The report of this committee was released in 1966. It recommended a subset of PL/1, as PL/1 has scientific and commercial features, is suitable for modular learning and contains most of the features found in high level languages today. The subset specification was not rigid; it consisted of a list of PL/1 features and indicated which features were considered necessary, desirable or unnecessary. The report stressed the need for fast compilation and good diagnostics and recommended implementation on a small machine, since a small machine in each school is more satisfactory than courier service to a larger central machine and the cost is less than terminals linked to a medium scale computer. In 1966, a group in IBM, advised by Professors Lee and Boulton of the University of Toronto, undertook the development of a compiler meeting these specifications for the 1130.
(SLIDE 2)
The problems we faced were that we had to write a fast compiler, with good compile and object time diagnostics, with diagnostics in source program terms, for a sophisticated language and on a small machine with an instruction set ill adapted to either compiling or supporting the required language features. Furthermore, many features of PL/1 need the services of a much more comprehensive operating system than the 1130 Disk Monitor. Some of the requirements such as fast compilation and good diagnostics are conflicting. Factors in our favor were that the source programs would be trivial in size and complexity, and in execution time requirements; only one level of diagnostics was required, there would be no warning messages or attempts to correct errors made by the programmer; a disk was required to teach file concepts and was, therefore, available for compiling; language specifications were

flexible, enabling us to include, omit or modify features as the system developed; and finally, since it was to be a Type III program, we avoided all the red-tape associated with Type I and II programs.

(SLIDE 3)

The system is designed to run on an 8K, one disk system, with 1442 or 2501 reader and 1132 or 1403 printer. There is no support for paper tape, the console keyboard/printer, plotter or multiple disk drives. The system prepared for version two of 1130 monitor will not support a card punch. Additional core storage is used if available which permits larger programs to be compiled and executed but does not provide any speed increase.

(SLIDE 4)

The data types shown on the slide are supported by SL/1. Decimal arithmetic was implemented to illustrate the effects of variable length fields and to avoid confusing students with the conversion and truncation phenomena associated with binary fractions. Any of the data types shown may be in array form. Arrays may have one or two dimensions, and dimension bounds may be negative or positive. Character and bit strings may be combined in major and minor structures with a maximum structure level of four. Structures may be dimensioned but any data element must be accessible using, at most, four subscripts. Full or partially qualified names may be used for referencing structured data.

(SLIDE 5)

The data types shown on this slide are not implemented in SL/1, although provision was made in the design for some of them and the hooks are still there. Binary was not considered to be necessary. Picture was omitted with regret because of implementation problems, but it has been retained in format statements.

(SLIDE 6)

The major language features of SL/1 are shown on this slide. Internal and external procedures and begin blocks were included to facilitate teaching of subroutine concepts, scope of names and, to a limited extent, the effects of dynamic storage allocation. Although storage is not allocated dynamically, some of its affects are simulated by making data within a block unavailable when the block is inactive and by reinitialising all variables, except those with the static attribute, when a block becomes active. For those variables which do not have the initial attribute, this involves initialising with "garbage". It is in the implementation of I/O features that we have deviated most from PL/1. Because of lack of buffer space, each GET or PUT statement in SL/1 begins a new record instead of processing the file as a continuous data stream. The absence of an adequate operating system forced a lot of compromises in handling record I/O. Program control statements such as IF and GO TO are implemented as defined in PL/1. The ON statement is an exception; in SL/1 there is no return to the point of interrupt following execution of an ON-unit. Unless the programmer includes a GO TO statement in the ON-unit, execution of the ON-unit will be followed by program termination. Variables may be declared either explicitly or implicitly. Some of our users consider implicit declaration to be a major source of programmer errors and would like to see the feature eliminated. Perhaps an optional list of implicitly declared variables might be in order here.

(SLIDE 7)
Expressions of arrays and structures are not permitted. The assignment statement, however, allows a scalar variable to be assigned to all elements of an array, or an array to be assigned to another array with similar dimensions. A structure may be assigned to another structure provided their attributes are identical. For example, if A and B are both arrays or are both structures, then A=B is valid but A+B is not valid. We would have liked to include attribute factoring but, unfortunately, it used too much core and had to be dropped.
(SLIDE 8)
Implementation was kept as straight-forward as possible. The only options available to the user are: list/no list of source program; compile only/compile and go; and, execute an object program stored on disk. Only the source program is listed, no symbol table list, load map, etc., are provided. Compiler generation of in-line 1130 code was considered to be difficult and would make a fast compiler impossible. The alternatives, therefore, were to generate object programs consisting almost entirely of subroutine linkages or to generate interpretive code. We chose to generate interpretive code since it produces more compact object programs and is not significantly slower in execution than a series of subroutine linkages. The interpretive code is actually the machine language of a hypothetical machine. The instruction set of this hypothetical machine was designed so that many of the difficult source language features can be compiled as a single machine language instruction . For example, a DO statement, after generation of code to evaluate any expressions in the statement, compiles as a single instruction. The hypothetical machine also has registers specifically designed for handling SL/1 language features. This approach to the design of the interpretive code takes a large burden from the compiler and transfers it to the execution time interpreter. Our goal of a fast compiler was, therefore, more easily achieved, and the resulting poor execution times are justified by the high ratio of compilations to executions and the trivial nature of the programs in a student environment. The use of interpretive code and the decimal format of the data have made it impossible for SL/1 programs to link to routines written in FORTRAN or 1130 Assembler and vice versa, but this is not considered a serious drawback. It also forced us to write the function library routines SIN, SQRT, MAX, etc. in SL/1. These routines are, therefore, slow and, in some cases, not very accurate. Because the source programs we expect to compile are small, there is no provision for spilling symbol tables, attribute tables, etc., onto disk. This limits the size of program the compiler can handle but usually the object program runs out of space before the compiler tables start to overflow. No provision has been made for overlays in object programs.
(SLIDE 9)
Compiler performance is very sensitive to the format of the source program. If the source program is punched multiple statements per card, then the number of cards compiled per minute is low but the statements per minute are high. Punching the source program one statement per card improves the cards per minute. Programs punched one statment per card compile at about the same rate as the 1130 FORTRAN compiler process equivalent programs. Compiler performance is also affected by the language features used in the source program. Object program execution times depend very heavily on language

features used and, to some extent, on the order in which they are executed. Apart from the arithmetic routines, which are core resident, most of the interpreter is disk resident and is fetched into core in sections as it is needed. Thus, most of the execution time is devoted to fetching routines from disk. Probably the most meaningful performance figure is the through-put. This estimate was obtained by watching a class of beginners debugging four small programs each during a four-hour lab session. Since the system is used mainly for teaching commercial data processing and about 75% of the statements executed are I/O commands, the object time performance is not as bad as it looks.

SL/1 is being used in about 20 installations in Canada but apparently only for commercial applications. FORTRAN is still being taught to the science and arts students, and some schools prefer to teach COBOL or languages of their own invention. There is some provision in the system for adding more language features and given a 16K machine, the through-put could be considerably improved by making some or all of the disk resident routines core resident. At the present time, we have no plans for further development.

REASONS (Slide 1)

Standard language for all schools
Scientific and commercial
Easy to learn
Rich language
Fast compiler
Extensive diagnostics
Inexpensive machine

TRADE-OFFS (Slide 2)

Fast compile
Extensive diagnostics
Source language diagnostics
Rich language
Small core

Trivial programs
Unoptimised object code
All errors fatal
Flexible language specs
Disk available
Type III

HARDWARE (Slide 3)

8K (or more) core
1 Disk
1442/2501 Reader
1132/1403 Printer
48 char. set
BCD or EBCDIC

DATA TYPES SUPPORTED (Slide 4)

Character
Bit
Fixed decimal
Float decimal
Label

Arrays
Structures

DATA TYPES NOT SUPPORTED (Slide 5)

Binary
Pointer
Picture
Sterling
Complex

Task
Cell
Area
Event

LANGUAGE FEATURES SUPPORTED (Slide 6)

Internal/External procedures
Begin blocks
Stream I/O (List/Edit)
Record I/O
IF, GOTO, DO, ON, ETC
Explicit/Implicit declaration

LANGUAGE FEATURES NOT SUPPORTED (Slide 7)

Macros
Expressions of arrays/structures
Dynamic storage allocation
Multiprogramming
Attribute factoring
Recursive procedures
Pseudo variables

IMPLEMENTATION (Slide 8)

Limited options
Only source listed
Interpretive execution
Single language
No table spill to disk
No object program overlays

PERFORMANCE (Slide 9)

Compile:	20 - 50 cards/min.
	30 - 60 statements/min.
Execute:	25 - 1000 statements/min.
Thru-put:	50 programs/hour
Max program:	about 100 statements

Session Number Wed C-5 Session Name "Relationships Between Customer & S.E., F.E., & Marketing Reps."

Chairman R. J. Snailer

Time 1:30 - 3:00 p.m. Attendance (No.) 38

Speakers

Robert Lukeman - I.B.M. White Plains - Mark Rep.

Warren Gillis - I.B.M. White Plains - S.E.

Gerald Monjeau - I.B.M. White Plains - F.E.

Synopsis of Meeting Each speaker made a brief presentation dealing generally with I.B.M.'s policies, procedures and training of personnel with regard to his specific area. He then entertained questions from the floor. Some comments dealt with a particular company's dissatisfaction with an individual functioning in one of the capacities covered by this panel. There were favorable comments as well. Other questions promoted answers of a more general and informative nature. There was very active participation on the part of the audience. The questions dealt with topics such as "Unhooking", APAR's, PTF's, 1800-TSX, and methods of expediting action under emergency conditions.

The talks and following discussions brought out the team work required for obtaining "customer satisfaction". The customers in the audience seemed satisfied with this team's handling of their inquiries and problems.