Y27-7116-1

Program Logic

IBM System/360 Conversion Aids: The 1620 Simulator for IBM System/360

Program Number 360C-SI-752

This document describes the internal logic of the IBM System/360 Simulator for the IBM 1620 Model 1 and Model 2.

Program Logic Manuals are intended for use by IBM systems engineers involved in program maintenance, and by systems programmers involved in altering the program design. Program logic information is not necessary for program operation and use; therefore, distribution of this manual is limited to persons with program maintenance or modification responsibilities.

Restricted Distribution

PREFACE

This document describes the structure and functions of the Simulator: its components, their functions, and the control flow among them. The organization of each component and the instructions used to implement its functions are described in the program listing.

The manual consists of six sections. Section 1 is an introduction to the Simulator and includes a description of its overall structure and input/output flow. Section 2 describes the simulation program, SIM20; Section 3 describes EDITOR, the program used to adapt SIM20 to the 1620 system simulated; Section 4 describes the disk initialization program, DSKINT; Section 5 describes the updating program, UPDT20; and Section 6 describes the common subprograms used by EDITOR and UPDT20.

The reader should be familiar with the contents of the following publications:

IBM System/360 System Summary, Form A22-6810
IBM System/360 Principles of Operation, Form A22-6821
IBM 1620 Central Processing Unit, Model 1, Form A26-5706
or
IBM 1620 Central Processing Unit, Model 2, Form A26-5781
IBM System/360 Conversion Aids: The 1620 Simulator for IBM System/360, Form C28-6529

Second Edition (September 1966)

This is a major revision of, and makes obsolete, Form Y27-7116-0. The present edition of this publication should be reviewed in its entirety.

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

A form for reader's comments appears at the back of this publication. Address any additional comments concerning the contents of this publication to: IBM France, Centre d'Etudes et Recherches, Programming Publications, Department 841, 06 - La Gaude, France.

FIGURES

TABLES

CHARTS

The 1620 Simulator distributed by IBM is a set of four programs, plus a group of general subprograms used by two of them.

The four programs which make up the Simulator are:

1. SIM20, the program which contains the routines to simulate the 1620

2. EDITOR, a program used to adapt SIM20 to the particular 1620 system being simulated and to the System/360 used for simulation, and to create a self-loading version of SIM20

3. DSKINT, a disk initialization utility program used to simulate 1620 systems with disk storage drives

4. UPDT20, the program used to maintain and modify the Simulator

The subprograms used by EDITOR and UPDT20 are:

1. ABSLOD, an absolute loader used to load the other four programs or SIM20 into System/360 main storage

2. CONTPR, a control program used to supervise System/360 interruptions, to perform physical I/O operations, and to communicate with the 1052 Printer-Keyboard

3. IOPACK an I/O support package used to perform logical I/O operations on devices for Simulator support functions

4. INIT, an initialization program used to initialize CONTPR, IOPACK, and RELLDR

5. RELLDR, a relocating loader used to load EDITOR and UPDT20 into System/360 main storage

The way in which these programs and subprograms perform the various functions of the Simulator is outlined in the following sections and is described in more detail in the sections devoted to the individual programs and subprograms.

OVERALL LOGIC OF THE SIMULATOR

Chart AA shows the overall logic of those parts of the Simulator using the common subprograms. This relationship, in conjunction with the System/360 main storage allocation, is described in the following section.

SYSTEM/360 MAIN STORAGE ALLOCATION

Figures 1 through 6 illustrate the allocation of System/360 main storage to the programs and subprograms which make up the Simulator. Each step corresponds to a phase of the loading or initialization procedure.

Figure 1 shows the allocation of System/360 main storage to the components of EDITOR; Figure 2, to those of SIM20 when disks are not simulated; Figure 3, to those of SIM20 when disks are simulated, and the I/O simulation routines are permanently in main storage; Figure 4, to those of SIM20 when disks are simulated, and the I/O simulation routines are resident on disk; Figure 5, to those of DSKINT; Figure 6, to those of UPDT20. The System/360 addresses given below are approximate, and give an idea of the amount of main storage allocated to each component.

EDITOR

The System/360 main storage allocation for EDITOR is given in the following steps and follows the logic shown in Chart AA.

Step 1 (IPL): When the load key on the system control panel is pressed, the IPL sequence is loaded into the first bytes of System/360 main storage, and ABSLOD is loaded into an upper part of main storage.

Step 2 (Absolute Load): ABSLOD loads CONTPR, IOPACK, INIT, and RELLDR. After having loaded these subprograms, it sends the addresses of CONTPR, IOPACK, and RELLDR to INIT.

Step 3 (Subprogram Initialization): Control is then transferred to INIT, which reads the DEV360, DEVSUP, and CALL control cards; from the information in these cards, it builds up the appropriate channel and unit control blocks and initializes CONTPR, IOPACK, and RELLDR.

Step 4 (Relocating Load): After initialization, control is transferred to RELLDR, which loads EDITOR into System/360 main storage. It uses IOPACK for I/O operations, and uses certain facilities of CONTPR.

```
Step 1
┌──┬─────────────────────────────────────────────────┬──────┬───────────────────┐
│I │                                                 │ABSLOD│                   │
│P │                                                 │      │                   │
│L │                                                 │ (1K) │                   │
└──┴─────────────────────────────────────────────────┴──────┴───────────────────┘

Step 2
┌──────┬──────┬──────────────────────────────────┬──────┬──────┬─────┬──────┬─────┐
│CONTPR│IOPACK│                                  │ INIT │ABSLOD│     │RELLDR│     │
│      │      │                                  │      │      │     │      │     │
│ (3K) │ (3K) │                                  │ (4K) │ (1K) │     │ (5K) │     │
└──────┴──────┴──────────────────────────────────┴──────┴──────┴─────┴──────┴─────┘

Step 3
┌──────┬──────┬───────┬──────────────────────────┬──────┬──────┬─────┬──────┬─────┐
│CONTPR│IOPACK│  I/O  │                          │ INIT │ABSLOD│     │RELLDR│     │
│      │      │Control│                          │      │      │     │      │     │
│ (3K) │ (3K) │Blocks │                          │ (4K) │ (1K) │     │ (5K) │     │
└──────┴──────┴───────┴──────────────────────────┴──────┴──────┴─────┴──────┴─────┘

Step 4
┌──────┬──────┬───────┬──────┬───────────────────────────────────────────────────┐
│CONTPR│IOPACK│  I/O  │EDITOR│                                                   │
│      │      │Control│      │                      Unused                       │
│ (3K) │ (3K) │Blocks │ (5K) │                                                   │
└──────┴──────┴───────┴──────┴───────────────────────────────────────────────────┘
```

Figure 1. System/360 Main Storage Allocation for EDITOR

SIM20

The System/360 main storage allocation for SIM20 depends on the version of the I/O simulation section used.

No Disk Simulation

When the 1620 installation being simulated has no disk storage drives, the loading procedure is as described in the following paragraphs and as illustrated in Figure 2 (see Chart AB).

Step 1 (IPL): Same as step 1 for EDITOR.

Step 2 (Absolute Load): ABSLOD loads a condensed SIM20 version of CONTPR into addresses 00000 through 02299; it then loads SIM20. CPU and console simulation routines are loaded into addresses 02300 through 09699, and I/O simulation routines are loaded into addresses 09700 through 10999.

```
Step 1 (Same as in Figure 1)

Step 2
┌───────┬──────────┬──────────┬──────────────────────────┬──────┬────┐
│ 1620  │ CPU and  │   I/O    │                          │ABSLOD│    │
│Control│ Console  │(Non-disk)│                          │      │    │
│Program│Simulation│Simulation│                          │ (1K) │    │
└───────┼──────────┼──────────┼──────────────────────────┴──────┴────┘
        │          │          │
      02300      09700      11000

Step 3
┌───────┬──────────┬──────────┬──────────────────────────────────────┐
│ 1620  │ CPU and  │   I/O    │             Simulated                │
│Control│ Console  │(Non-disk)│             1620 Core                │
│Program│Simulation│Simulation│              Storage                 │
└───────┼──────────┼──────────┼──────────────────────────────────────┘
        │          │          │
      02300      09700      11000
```

Figure 2. System/360 Main Storage Allocation for SIM20 (No Disks)

Step 3 (Setting Simulated 1620 Core Storage): Control is then transferred to SIM20, which sets simulated 1620 core storage to zero, starting at address 11000. Storage is set to zero through address 30999 to simulate a 20,000-position 1620; through address 50999 to simulate a 40,000-position 1620; through address 70999 to simulate a 60,000-position 1620.

Storage-Resident I/O (Including Disk) Routines

Step 1 (IPL): Same as step 1 for EDITOR.

Step 2 (Absolute Load): The same as step 2 for a 1620 with no disk storage drives, except that the disk simulation routines are loaded into addresses 12300 through 14199 (see Figure 3).

Step 3 (Setting Simulated 1620 Core Storage): Control is then transferred to SIM20, which sets simulated 1620 core storage to zero, starting at address 14200. Storage is set to zero through address 34199 to simulate a 20,000-position 1620; through address 54199 to simulate a 40,000-position 1620; through address 74199 to simulate a 60,000-position 1620.

Disk-Resident I/O (Including Disk) Routines

Step 1 (IPL): Same as step 1 for EDITOR.

Step 2 (Absolute Load): ABSLOD loads a condensed SIM20 version of CONTPR into addresses 00000 through 02299; it then loads SIM20. CPU and console simulation routines are loaded into addresses 02300 through 09699, and I/O simulation routines (other than those for disks) are loaded into a 2,000-byte reserved area at addresses 09700 through 11699 (see Figure 4).

Step 3 (Setting Simulated 1620 Core Storage): Control is then transferred to SIM20, which sets simulated 1620 core storage to zero, from address 11700 through 31699.

When a 1620 disk instruction is encountered, the disk simulation routines are read into System/360 main storage in the 2,000-byte reserved area from address 09700 through 11700, thus overlaying the other I/O simulation routines.

DSKINT

The System/360 main storage allocation for SIM20 when a 1620 installation with disk storage drives is being simulated depends on whether or not the disk simulation routines are permanently in main storage. In either case, DSKINT must be used to place certain initial information on the 2311 Disk Storage Drive.

The main storage allocation for DSKINT is given in the following steps and is illustrated in Figure 5 (see Chart AC).

Step 1 (IPL): Same as step 1 for EDITOR.

Step 2 (Absolute Load): ABSLOD loads a condensed SIM20 version of CONTPR into addresses 00000 through 02299; it then loads SIM20 and DSKINT. CPU and console simulation routines are loaded into addresses 02300 through 09699, I/O simula-

```
Step 1 (Same as in Figure 1)

                                12300
Step 2                            |
┌───────┬──────────┬──────────┬─────┼──────────┬──────────────────────┐
│ 1620  │ CPU and  │   I/O    │     │   Disk   │                      │
│Control│ Console  │(Non-disk)│     │Simulation│                      │
│Program│Simulation│Simulation│     │          │                      │
└───────┼──────────┼──────────┼─────┴──────────┼──────────────────────┘
        |          |          |                |
      02300      09700      11000            14200

                                12300
Step 3                            |
┌───────┬──────────┬──────────┬─────┼──────────┬──────────────────────┐
│ 1620  │ CPU and  │   I/O    │     │   Disk   │      Simulated       │
│Control│ Console  │(Non-disk)│     │Simulation│      1620 Core       │
│Program│Simulation│Simulation│     │          │       Storage        │
└───────┼──────────┼──────────┼─────┴──────────┼──────────────────────┘
        |          |          |                |
      02300      09700      11000            14200
```

Figure 3. System/360 Main Storage Allocation for SIM20 (Storage-Resident Disk Simulation Routines)

tion routines (other than those for disks) are loaded into addresses 09700 through 10999, disk simulation routines are loaded into addresses 12300 through 14199, and DSKINT is loaded into addresses 16400 through 24599.

<u>Step 3 (Disk Initialization):</u> Control is then transferred to DSKINT, which on operator request:

1. Places certain initial information on a specified 2311 Disk Storage Drive and establishes the format of information on this storage drive

2. Loads SIM20 onto the same 2311 Disk Storage Drive, in a self-loading format

System/360 main storage is then dumped onto disk, as follows:

- CONTPR and CPU and console simulation routines onto cylinder 00

- The I/O simulation routines (other than those for disks) onto cylinder 01

- The disk simulation routines onto cylinder 02

DSKINT is not loaded onto disk; in order to be used again, it must be re-loaded from cards.

For a subsequent SIM20 run, SIM20 alone is loaded from disk, using an IPL procedure.

<u>Note:</u> "Console Simulation" includes the Insert and Automatic Load operations.

UPDT20

System/360 main storage allocation for UPDT20 is given in the following steps and is illustrated in Figure 6 (see Chart AA).

<u>Steps 1 through 3:</u> Same as steps 1 through 3 for EDITOR.

<u>Step 4 (Relocating Load):</u> After initialization, control is transferred to RELLDR, which loads UPDT20 into System/360 main storage. It uses IOPACK for I/O opera-

```
Step 1 (Same as in Figure 1)

Step 2
┌───────┬──────────┬──────────┬────────────────────────────────────────┐
│ 1620  │ CPU and  │   I/O    │                                        │
│Control│ Console  │(Non-disk)│                                        │
│Program│Simulation│Simulation│                                        │
└───────┼──────────┼──────────┼────────────────────────────────────────┘
        │          │          │
      02300      09700      11700


Step 3
┌───────┬──────────┬──────────┬───────────────────────────┬────────────┐
│ 1620  │ CPU and  │   I/O    │         Simulated         │            │
│Control│ Console  │(Non-disk)│         1620 Core         │            │
│Program│Simulation│Simulation│          Storage          │            │
└───────┼──────────┼──────────┼───────────────────────────┼────────────┘
        │          │          │                           │
      02300      09700      11700                       31700
```

Figure 4. System/360 Main Storage Allocation for SIM20 (Disk-Resident Disk Simulation Routines)

```
Step 1 (Same as in Figure 1)
                                     12300          16400
Step 2                                 │              │
┌───────┬──────────┬──────────┬────┼──────────┬────┼──────────────┬───┐
│ 1620  │ CPU and  │   I/O    │    │   Disk   │    │              │   │
│Control│ Console  │(Non-disk)│    │Simulation│    │    DSKINT    │   │
│Program│Simulation│Simulation│    │          │    │              │   │
└───────┼──────────┼──────────┼────┴──────────┼────┴──────────────┼───┘
        │          │          │               │                   │
      02300      09700      11000           14200               24600
```

Figure 5. System/360 Main Storage Allocation for DSKINT

tions, and uses certain facilities of CONTPR.

LAYOUT OF THE SIMULATOR ON TAPE

The Simulator system tape distributed by IBM contains the four programs (SIM20, EDITOR, DSKINT, and UPDT20), the five common subprograms (ABSLOD, CONTPR, IOPACK, INIT, and RELLDR), and the sample program described in the publication IBM System/360 Conversion Aids: The 1620 Simulator for IBM System/360, Form C28-6529. EDITOR, UPDT20, and the common subprograms are in binary, SIM20 and DSKINT are in symbolic form, and the sample program is in BCD code.

The way in which these programs and subprograms are placed on tape is shown in Figure 7. There is a tape mark at the end of each program and at the end of the common subprograms. Following the last binary program (EDITOR) is a one-card binary file, labeled SYSINEND, which marks the end of the binary system tape. This file is followed by a tape mark, SIM20, DSKINT, and another tape mark. At the end of the system tape is the sample program, in BCD code, followed by two tape marks.

Steps 1 through 3 (Same as in Figure 1)

Step 4

```
┌──────┬───────┬────────┬───────┬──────────────────────────────────────────────┐
│CONTPR│IOPACK │  I/O   │ UPDT20│                                              │
│      │       │Control │       │                                              │
│ (3K) │ (3K)  │Blocks  │  (7K) │                                              │
└──────┴───────┴────────┴───────┴──────────────────────────────────────────────┘
```

Figure 6. System/360 Main Storage Allocation for UPDT20

```
┌──────┬────────┬───────┬───────┬──────┬──────┬──────┬─┬────────────────────┬───────┬┐
│IPL  +│        │       │       │      │      │      │ │                    │Sample ││
│      │ CONTPR │IOPACK │ INIT  │RELLDR│UPDT20│EDITOR│ │    SIM20+DSKINT    │       ││
│ABSLOD│        │       │       │      │      │      │ │                    │Program││
└──────┴────────┴───────┴───────┴──────┴──────┴──────┴─┴────────────────────┴───────┴┘
                                                      ↑
                                                      │
                                                  SYSINEND
                                             End of System Tape
```

Figure 7. Layout of Programs on the System Tape

Chart AA. Overall Logic of EDITOR/UPDT20

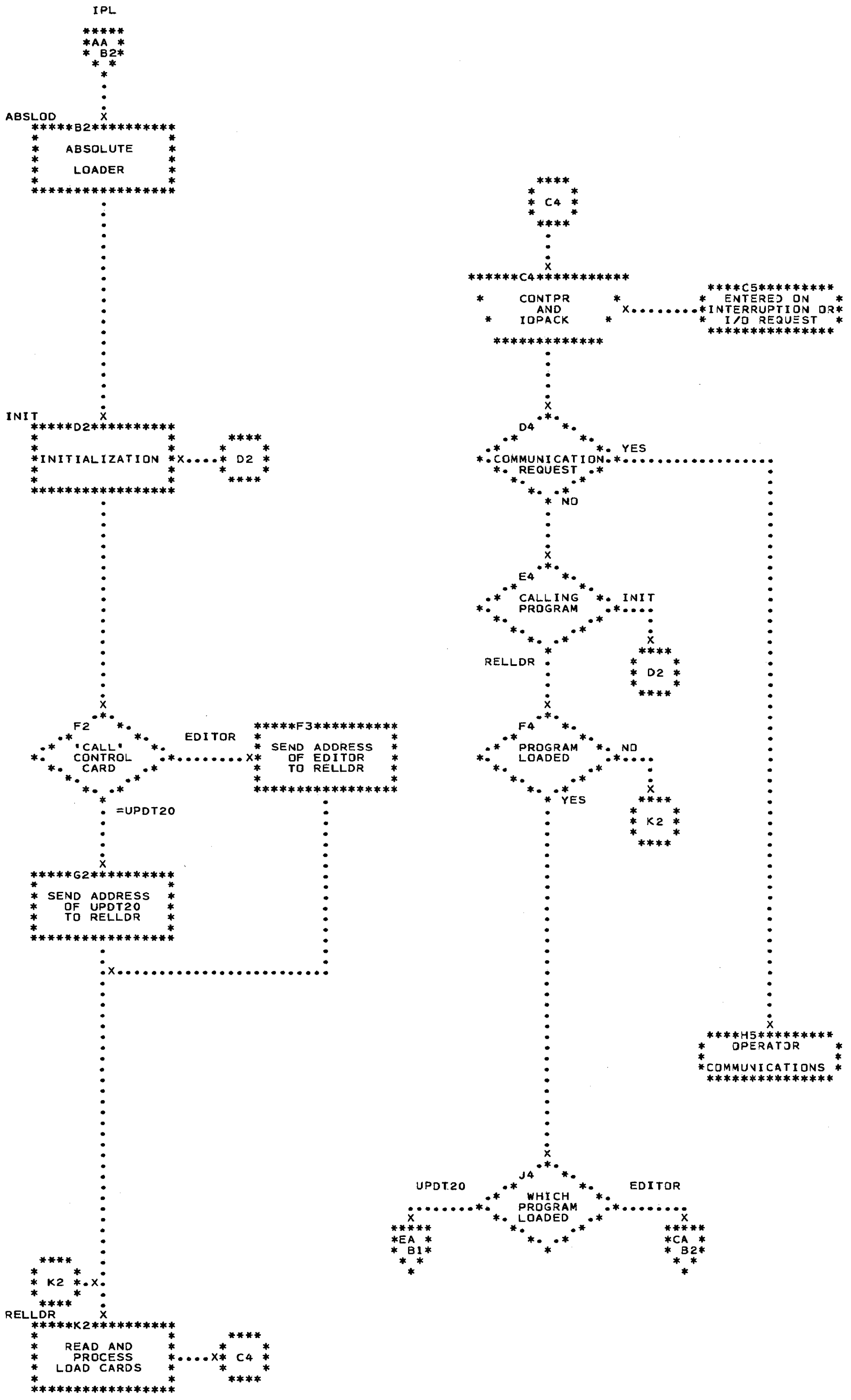


Chart AB. Overall Logic of SIM20 (No Disk Simulation)

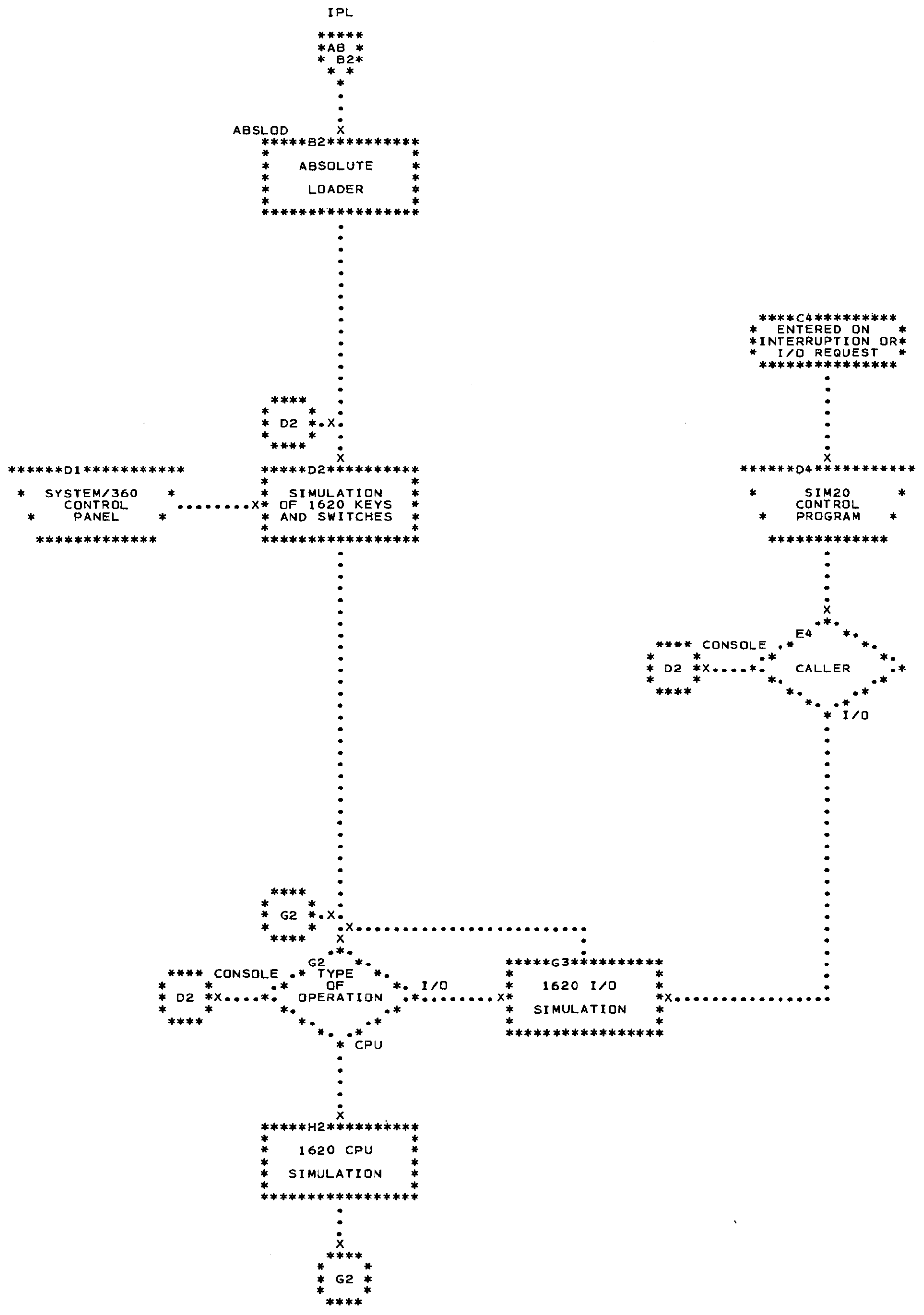
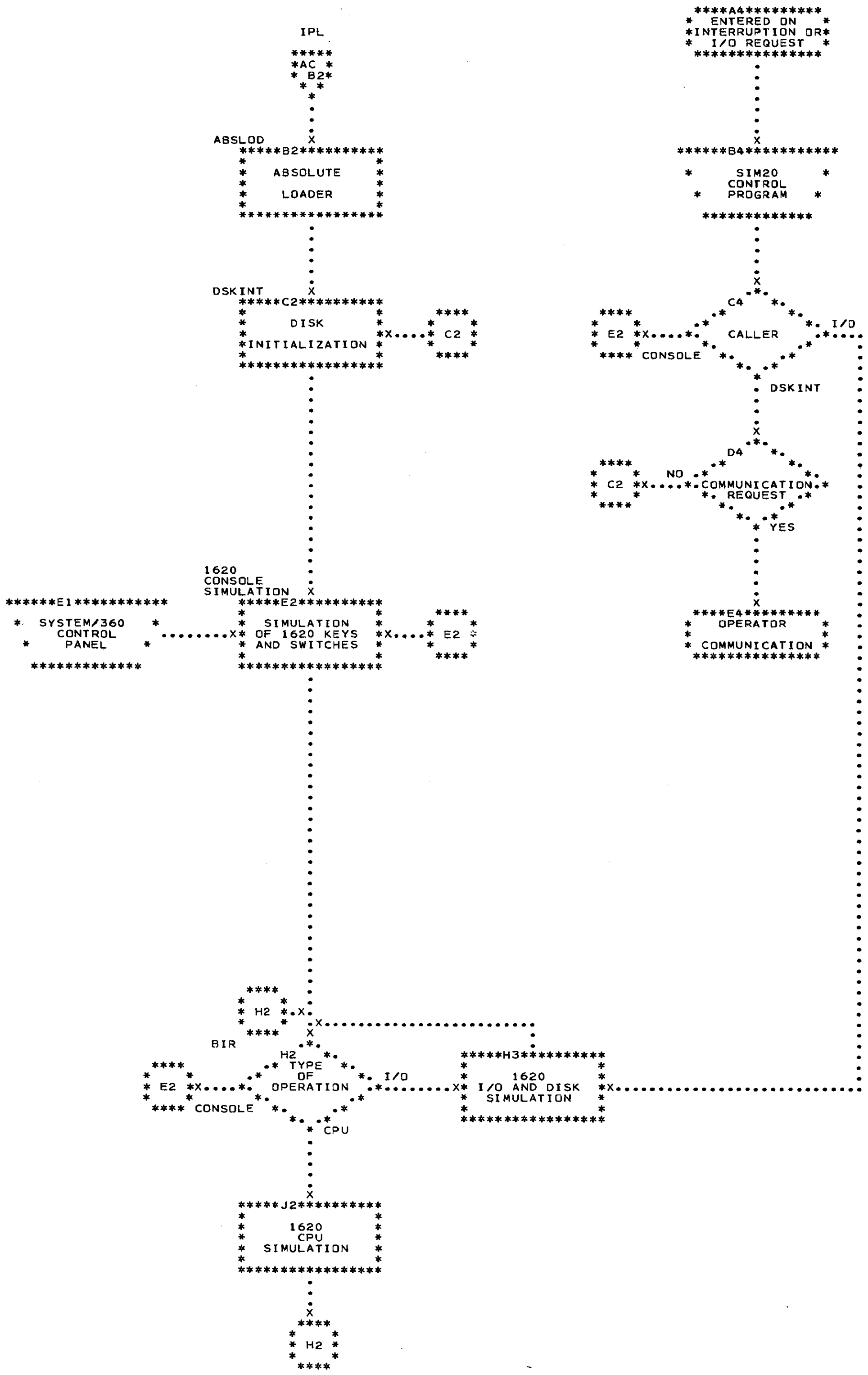


Chart AC. Overall Logic of SIM20 (With Disk Simulation)



SIM20 consists of the following logical sections:

1. CPU simulation routines, which simulate the 1620 CPU instructions

2. I/O simulation routines, which simulate 1620 I/O instructions

3. Console simulation routines, which simulate the functions of the 1620 Console; that is, keys, switches, and Automatic Load

The way in which these logical sections perform the various functions of SIM20 is described in detail in the text devoted to the individual logical sections. Charts AB and AC show the overall logic of SIM20; that is, the relationship among the logical sections and CONTPR.

When SIM20 has been loaded into System/360 main storage, control is transferred to the console simulation section, which simulates the functions of the 1620 Console and allows operator communication with SIM20.

The CPU simulation section simulates 1620 CPU instructions. It gives control to the I/O simulation section or to the console simulation section whenever it encounters an I/O or console instruction, or whenever an I/O or console interruption has been signaled by CONTPR.

The I/O simulation section simulates 1620 I/O instructions, calling on CONTPR to perform any necessary physical I/O operations.

CPU SIMULATION

This logical section of SIM20 is made up of the following parts:

1. Simulated 1620 core storage

2. Simulated 1620 index registers (Model 2) and storage registers

3. Basic Interpretive Routine

4. Subroutines for address conversion

5. Operation routines to simulate 1620 instructions

6. Tables

SIMULATED 1620 CORE STORAGE

Each six-bit position of 1620 core storage is simulated by one byte in System/360 main storage.

Each 1620 instruction is represented by 12 bytes. The first two bytes contain the operation code of the instruction, the next five bytes contain the P address, and the last five bytes contain the Q address. If one or both addresses are missing, the corresponding 5- or 10-digit fields may be used by the 1620 program for storing data, as is usual in 1620 programming.

Representation of 1620 Positions

Each six-bit position is simulated by a System/360 byte, as follows:

```
                      ┌─┬─┬─┬─┬─┬─┐
1620                  │C F│8 4 2 1│
                      └─┴─┴─┴─┴─┴─┘

              ┌─┬─┬─┬─┬─┬─┬─┬─┐
System/360    │8 4 2 1│8 4 2 1│
              └─┴─┴─┴─┴─┴─┴─┴─┘
                zone    numeric
```

The 1620 parity-check bit is simulated by the parity bit of the byte. The zone part of the byte can contain either a hexadecimal D, which represents the flag, or a hexadecimal F, which indicates an unflagged character.

The meaning of the flag (D) depends on its position in the field. The different positions and their meanings are shown below.

Position	Meaning
In the rightmost address of a field	Minus sign
In the leftmost address of a field	Field mark
In the rightmost byte of an address field	Indirect addressing
In the thousands, hundreds, or tens position of an address field	Address indexing
In an entry in the Add table in simulated 1620 core storage (Model 1 only)	Carry

Simulated 1620 numeric characters and numeric special characters are given in Table 1, and simulated 1620 alphameric characters and alphameric special characters are given in Table 2.

For all 1620 operations in the numeric mode, 1620 characters are represented in simulated core storage by the hexadecimal characters F0 through FF and D0 through DF, and are processed directly. Numeric special characters, however, are truncated at processing time in arithmetic and compare operations, though they are not modified in simulated core storage. For example, FA is truncated to F0 when placed in a general register to be added to another digit; but it retains the value FA in simulated core storage.

The rule used for truncating special characters is: the 8 and 2 bits are suppressed. Thus,

FA is truncated to F0	DA is truncated to D0
FB is truncated to F1	DB is truncated to D1
FC is truncated to F2	DC is truncated to D2
FD is truncated to F3	DD is truncated to D3
FE is truncated to F4	DE is truncated to D4
FF is truncated to F5	DF is truncated to D5

Addressing of Fields

A field is defined by the address of its rightmost byte, and is processed from right to left until a byte with a hexadecimal D in its zone part is encountered. The address of the byte being processed is placed in general register RP or RQ. As the bytes of a field are addressed consecutively, the contents of RP or RQ are decremented by one each time a byte is processed.

Addressing of Records

A record is defined by the address of its leftmost byte, and is processed from left to right until a byte with a hexadecimal A in its numeric part is encountered. The address of the byte being processed is placed in general register RP or RQ. Since the bytes of a record are addressed consecutively, the contents of RP or RQ are incremented by one each time a byte is processed.

Table 1. 1620 Numeric Characters and Numeric Special Characters

1620 CHARACTER	SYSTEM/360 REPRESENTATION	
	UNFLAGGED	FLAGGED
0,1,2,...9	F0,F1,F2,...F9	D0,D1,D2,...D9
Record Mark	FA	DA
Group Mark	FF	DF
Numeric Blank	FC	DC
Minus Zero	------	D0

Table 2. 1620 Alphameric Characters and Alphameric Special Characters

1620	System/360	1620	System/360	1620	S/360
A	F4F1	R or -9	F5F9	8	F7F8
B	F4F2	S	F6F2	9	F7F9
C	F4F3	T	F6F3	.	F0F3
D	F4F4	U	F6F4	)	F0F4
E	F4F5	V	F6F5	+	F1F0
F	F4F6	W	F6F6	$	F1F3
G	F4F7	X	F6F7	*	F1F4
H	F4F8	Y	F6F8	-	F2F0
I	F4F9	Z	F6F9	/	F2F1
J or -1	F5F1	0	F7F0	,	F2F3
K or -2	F5F2	1	F7F1	(	F2F4
L or -3	F5F3	2	F7F2	=	F3F3
M or -4	F5F4	3	F7F3	@	F3F4
N or -5	F5F5	4	F7F4	Blank	F0F0
O or -6	F5F6	5	F7F5	Flagged Zero	F5F0
P or -7	F5F7	6	F7F6	Record Mark	F0FA
Q or -8	F5F8	7	F7F7	Group Mark	F0FF

SIMULATED 1620 REGISTERS

Although most 1620 registers are not directly simulated, their functions are simulated by System/360 general registers and by elements of System/360 main storage. The following description gives the correspondence between 1620 register functions and the System/360 elements used for their simulation.

1620 Storage Registers

The function of the 1620 storage register IR-1 is simulated by the System/360 general register CNTR.

The functions of registers IR-2 and PR-1 are simulated by two full-words labeled IR2 and PR1.

The functions of the following 1620 storage registers are performed by working registers (general registers WR1 to WR6):

OR-1
OR-2
OR-3
OR-4
OR-5
PR-2
PR-3
MAR
MBR
MDR
OP
CR-1
Multiplier-Quotient
Digit and Branch

Since they are used only by the 1710 Control System, the storage registers IR-3 and IR-4 are not simulated.

1620 Model 2 Index Registers

Each of the two bands (Band 1 and Band 2) of seven index registers is simulated by numeric core storage positions at simulated 1620 addresses 300 through 339 (Band 1) and 340 through 379 (Band 2).

Simulator Registers

In addition to the general registers used to simulate 1620 registers, certain general registers are used for SIM20 functions. These registers and their functions are given in Table 3.

The two SIM20 base registers are not included in Table 3. The functions of these registers are described in the following paragraphs.

Table 3. Simulator General Register Assignment

SYMBOLIC NAME	FUNCTION
MAPORG	Contains the address of the first byte of simulated 1620 core storage
SIZE	Contains the address of the last byte of simulated 1620 core storage
RP	Contains the converted P address of the current 1620 instruction
RQ	Contains the converted Q address of the current 1620 instruction
R1, R2	Contain subroutine return addresses
WR1 to WR6	Used as working registers

SIM20 is divided into three logical parts in System/360 main storage. The first part is contained in the bytes with addresses 00000 through 04095, and any element in this part can be addressed directly. The second part is contained in the bytes with addresses 04096 through 08191, and the third part in the bytes with addresses 08192 through 12287.

The two base registers SIMB1 and SIMB2 are used in such a way that the address of any element of the second or third part of SIM20 can be expressed as the contents of SIMB1 or SIMB2 plus a displacement.

SIMB1 always contains the value 4096, and is used to address the second part of SIM20. SIMB2 always contains the value 8192, and is used to address the third part.

When disk simulation routines are permanently in main storage (on a System/360 with 65,536 or more bytes of main storage), SIMB2 may temporarily contain the value 12288, but is reset to 8192.

BASIC INTERPRETIVE ROUTINE

The Basic Interpretive Routine (BIR) is entered each time a 1620 instruction is encountered (see Chart BA). It first checks the value of the bit which simulates the Start/Stop key. If the value is one, the Stop key is simulated, and the simulated 1620 stops and enters the Manual mode. If the value is zero, the Start key is simulated, and the simulated 1620 enters the Automatic mode.

When the simulated 1620 is in the Manual mode, all the bits which simulate console keys and switches are scanned, the appropriate operations are executed, and the BIR is re-entered. When the simulated 1620 is in the Automatic mode, according to the value of the operation code of the current 1620 instruction (contained in the operation code table), the following operations are performed:

1. The standard address conversion subroutine is called, and the P address or the Q address, or both, are converted to binary.

2. Depending on the type of 1620 instruction, the indirect addressing or index address conversion subroutine, or both, are called.

ADDRESS CONVERSION SUBROUTINES

SIM20 contains three address conversion subroutines (see Chart BB): a standard subroutine used for all 1620 instructions, an indirect addressing subroutine used by those 1620 instructions for which indirect addressing is specified, and an index subroutine used when a band has been selected (1620 Model 2 only).

These subroutines are associated with the BIR, but, since not all 1620 instructions require both P and Q addresses, are not incorporated into the BIR. Furthermore, the subroutines are divided into three sections. P-address conversion only, Q-address conversion only, and P- and Q-address conversion.

Standard Address Conversion Subroutine

This subroutine converts the P address or the Q address, or both, into binary and adds the result to the contents of general register MAPORG. The contents of this register are then compared with the contents of general register SIZE to determine whether the address falls within the bounds of simulated 1620 core storage. If it does, control is transferred to the operation routine which simulates the execution of the current 1620 instruction; if it does not, the message MAR CHK is printed on the 1052 Printer-Keyboard, and simulation is stopped (the BIR enters the 1620 Manual mode).

Indirect Addressing Subroutine

This subroutine is entered from the standard conversion subroutine, and tests for the presence of a flag in the units position of the address in general register R1. If there is no flag, control is transferred to the operation routine which simulates the execution of the current instruction. If a flag is present, the indirect address is fetched from simulated core storage, and the procedure described under "Standard Address Conversion Subroutine" is executed until no further flag is found.

Index Subroutine (1620 Model 2)

This subroutine is entered from the standard conversion subroutine, and tests whether a band has been selected. If not, control is transferred to the indirect addressing subroutine. If so, the address of the index register is computed from the flag pattern of the address field of the instruction. The corresponding core storage field is packed, converted to binary, and added to the previously converted address. The result is compared with the contents of general register SIZE to determine whether the address falls within the bounds of simulated 1620 core storage. If the resulting 1620 address is greater than the address of the last byte of simulated core storage, it results in a MAR CHK indication. If the resulting 1620 address is negative, it is replaced by the System/360 effective address corresponding to the ten's complement of this negative address. Control is then transferred to the indirect addressing subroutine.

CPU SIMULATION PROCESS

Chart BA gives the overall operation of the CPU simulation section of SIM20. The following paragraphs describe this operation.

The BIR is entered each time a 1620 instruction is encountered. After having decoded the instruction, it places the simulated 1620 in either the Manual or the Automatic mode. When it is in the Manual mode, the bits which simulate the 1620 console keys and switches are scanned, and control is returned to the BIR. When it is in the Automatic mode, the execution of the current 1620 instruction is simulated.

When the instruction has been executed, the indicators involved in the operation are updated, the instruction counter CNTR is incremented by 12 (except for 1620 Branch instructions, where CNTR contains the P address), and control is returned to the BIR.

The simulation of 1620 CPU instructions normally consists of three distinct steps:

1. Converting 1620 addresses into effective System/360 binary addresses

2. Simulating the proper function of the instruction

3. Updating indicators (if necessary)

Charts BC, BD, and BE illustrate the logic of three typical routines used to process arithmetic operations in either fixed or floating point. The FIXADD routine adds two fields, digit by digit; the MULT routine multiplies two fields; and the FIXDIV routine divides two fields.

FIXADD Routine

The FIXADD routine (see Chart BC) adds two fields, using either the simulated 1620 Model 1 core storage tables (1620 addresses 300 through 399) or 1620 Model 2 direct addition. In both cases, each digit to be processed is converted before addition, using internal table look-up. This method allows the use of the FIXADD routine for subtraction as well as addition, using complement tables.

MULT Routine

The MULT routine (see Chart BD) directly simulates the 1620 multiplication process. Each partial product is computed from the multiply tables at simulated 1620 addresses 100 through 299, and is added to the product area. A carry is propagated from right to left as many times as necessary. The process is repeated for each digit of the multiplier field, the partial product being added each time to the preceding result.

When the leftmost flag in the multiplier field is encountered, the process is terminated and the indicators are updated. In fixed-point multiplication, the product is left in the product area; in floating-point multiplication, it is shifted (if necessary) and moved to the P field.

FIXDIV Routine

The FIXDIV routine (see Chart BE) directly simulates the 1620 automatic divide process. The Q field is subtracted (that is, added in complement form), digit by digit, from the dividend. This subtraction loop is repeated until a leftmost carry of zero is obtained, indicating that one operation too many has been performed. The divisor field is then shifted one position to the right, and the subtraction loop is repeated. The division is terminated when the rightmost address of the divisor corresponds to the rightmost address of the dividend. The indicators are then updated.

TABLES

There are two types of tables in the CPU simulation section of SIM20: an operation code table and code conversion tables.

Operation Code Table (OPTBL)

OPTBL is made up of 154 half-words, containing the absolute addresses in System/360 main storage of the routines which simulate the 1620 instructions. This table allows a maximum of 99 operation codes, of which some may be invalid on the particular 1620 system being simulated, and of which some are always invalid since they correspond to no existing 1620 operation code.

The Basic Interpretive Routine uses this table in order to transfer control to the appropriate routine to simulate a particular 1620 instruction. The BIR packs the operation code, multiplies it by two, and uses the result as an index in OPTBL. The half-word at the index is placed in general register R1, and control is then transferred to the address in R1.

Code Conversion Tables

Code conversion tables are used by the routines which simulate 1620 I/O operations (other than those for disk). Code conversion is not necessary for disk operations, since the code on disk is the same as that in main storage.

Code conversion is performed by the MASK subroutine: after filling the buffer, and before transferring the data to simulated storage (input operations); and after transferring data from simulated storage to the buffer, and before emptying the buffer (output operations). The I/O simulation routine which calls the MASK subroutine places the address of the appropriate code conversion table in a working register.

Each table is made up of a series of character strings. The last character string in the table is followed by a delimiter character (either X'EE' or X'EF', depending on the type of table). This delimiter character is used to stop table loading.

Each character string can contain a variable number of characters. The first character is a relative address index which indicates at which address in the zone the string should be loaded. The second character contains the length (in the form L-1) of the character string. The data in the string starts at the third character and contains L characters.

I/O SIMULATION

This section of the PLM contains a general description of the I/O simulation section of SIM20. Among the topics discussed are the simulation of 1620 I/O devices, buffer formats, and the general technique of I/O simulation.

CHANNEL CONTROL BLOCKS

One channel control block (CCB) in System/360 main storage is associated with each System/360 channel used by SIM20. SIM20 assumes that three CCBs have been created during an editing run; these CCBs correspond to multiplexor channel 0 and selector channels 1 and 2. The CCB for a particular channel contains the address of the first byte of the unit control block for each device attached to the channel.

Since the 1620 system has no channels, there is no restriction on the assignment of 1620 I/O devices to System/360 channels. The assignment of I/O devices to channels conforms to System/360 Operating System standard addressing, and is set up by DEVICE control information during an editing run.

The detailed structure of a CCB is given in Table 4.

Table 4. Structure of a CCB

WORD	CONTENTS
1	Always contains zeros
2	Contains the address of the first byte of this CCB
3 . . n	One word for each System/360 device attached to this channel, broken down as follows: The first byte contains the System/360 hexadecimal address of the device The last three bytes contain the address of the first byte of its associated UCB

UNIT CONTROL BLOCKS

One unit control block (UCB) in System/360 main storage is associated with each 1620 I/O device to be simulated. UCBs are created from DEVICE control information during an editing run. A maximum of ten 1620 I/O devices, each with a corresponding UCB, can be simulated, as follows:

- One console typewriter
- One paper tape reader
- One card reader
- One card punch
- One printer
- Four disk storage drives

When System/360 Operating System standard addressing is used, two of the disk storage drives are attached to selector channel 1, two to selector channel 2, and the rest of the devices to multiplexor channel 0.

The detailed structure of a UCB is given in Table 5.

Table 5. Structure of a UCB

SYMBOLIC NAME	CONTENTS
DEVTYP	Four bytes, containing the type of System/360 device (for example, 1442, 1052)
DEV360	Two bytes, containing the System/360 address of the device
DEVSPF	One byte, denoting the special features of the device
BORCH	One byte, giving the device status: 00=available 11=busy 01=chained 10=busy, unit check encountered
DEVSVC	Three bytes, containing the address of the SVC calling sequence
DEVCHN	Three bytes, containing, if chained, the address of the next UCB on the chain
DEVINT	Three bytes, containing the return address in case of an interruption
SENSE	Three bytes, containing sense information
INVST	One byte, containing any invalid status bits

CHANNEL TABLE

The Channel Table (CHTABL) is a block of four consecutive full-words in System/360 main storage. The first three full-words correspond, in order, to System/360 channels 0, 1, and 2; each contains the address of the first byte of the CCB associated with the corresponding channel. The fourth full-word must contain zeros.

BUFFERS

Only one I/O buffer is used by SIM20 since the processing of 1620 CPU instructions is interrupted only during the transfer of data from simulated 1620 core storage to the buffer, or from the buffer to simulated core storage.

The length of this buffer is set at 210 bytes, based on the following considerations:

1. I/O operations on the console typewriter and on paper tape devices are executed with groups of 100 characters, or fewer when a record mark or an end of block is detected.
2. I/O operations on a card read-punch device are executed with groups of 80 characters.
3. Output operations on the printer may process up to 144 characters at a time.
4. I/O operations on disk storage drives use two fields of 105 bytes each.

For 1620 I/O operations in the Alphameric mode, the expansion of each character read (or the reduction of each two-digit field to a written character) is performed, one character at a time, during code conversion. Thus, a double-length I/O buffer is not necessary.

LOGIC OF I/O SIMULATION

Depending on the characteristics of the 1620 system to be simulated, EDITOR produces one of two versions of the I/O simulation section.

1. To simulate 1620 systems that have disk storage drives and are too large to be simulated on a System/360 with 32,768 bytes of main storage, SIM20 is edited in the following way:
   a. I/O simulation routines for 1620 disk operations are placed on cylinder 2 of a 2311 Disk Storage Drive; I/O simulation routines for console typewriter, paper tape device, card read punch, and printer operations are placed on cylinder 1; and the rest of SIM20 is placed on cylinder 0. At the beginning of simulation, the contents of cylinders 0 and 1 are read into System/360 main storage.
   b. During simulation, when a 1620 disk operation is to be performed, SIM20 transfers the I/O simulation routines for disks from the 2311 into System/360 main storage, overlaying the I/O simulation routines (except Insert and Automatic Load) for the other 1620 devices. The 1620 disk operation is then simulated.
   c. Later, when a 1620 I/O operation is to be performed on a device other than a disk storage drive, the disk simulation routines are replaced in System/360 main storage by the I/O simulation routines for the other 1620 devices.

   The overall logic of this version of the I/O simulation section is shown in Chart BA.
2. Few 1620 systems require the above-described I/O simulation technique. When simulating most 1620 systems, all the I/O simulation routines remain permanently in System/360 main storage, and all 1620 I/O operations (including disk operations) are performed as soon as requested.

I/O SIMULATION ROUTINES (OTHER THAN DISK)

The BIR analyzes the operation code of the I/O instruction and, according to its value, transfers control to the appropriate simulation routine (see Charts BF, BG, and BH). The method of simulation depends on whether the instruction requests a read or a write operation.

Read Operation

A read operation is performed in the following way:

1. Information is read from an I/O device into the I/O buffer. The physical transfer of data from the device to the buffer is performed as follows:
   a. SIM20 submits an I/O and continue (SVC 2) or an I/O and interrupt at channel end (SVC 1) request to CONTPR.
      SVC 1 is used in card read punch and printer simulation; SVC 2 is

used in typewriter, paper tape, and disk simulation.

b. Control is transferred to CONTPR.

c. The physical read operation is begun.

d. Control is returned to SIM20 as soon as the request is accepted.

2. The MASK subroutine is called and, if so indicated by the command-check byte, loads the code conversion table. When the table has been loaded, the "lock switch" in the I/O request sequence is turned on. At channel end, CONTPR tests for exceptional conditions that may have been detected during the read operation. If none have been detected, control is returned to SIM20.

3. The VALIN subroutine converts the information in the buffer into SIM20 internal code and checks the validity of all characters.

4. The information is transferred from the I/O buffer into simulated 1620 core storage.

5. The 1620 I/O indicators and switches associated with the operation are updated, and control is passed to the BIR to analyze the next 1620 instruction.

Chart BG shows the overall logic of a read operation.

Write Operation

The simulation of a write operation is similar to that of a read operation; it is performed in the following way:

1. The VALOUT subroutine converts the information in the buffer from SIM20 internal code into the appropriate output code and checks the validity of all characters.

2. The information is transferred from simulated 1620 core storage into the I/O buffer.

3. The physical write operation is performed.

Chart BH shows the overall logic of a write operation.

Exceptional Conditions

The following exceptional conditions may occur during the execution of an I/O operation.

Unrecoverable Errors: I/O errors involving a channel or a control unit are classed as unrecoverable errors. When such an error is detected, a message is sent to the operator, and control is transferred to the BIR. The simulated 1620 is placed in the Manual mode.

Intervention Required: When intervention is required at an I/O device, a message is sent to the operator, and control is passed to the BIR.

The operator message simulates the status of a 1620 I/O indicator. The simulated 1620 enters the Manual mode, and the operator must perform a 1620 Start operation in order to resume simulation of the instruction.

Unit Exception: The response of SIM20 to a unit-exception condition depends on the type of read operation being performed. This condition may, for example, denote a last-card or cancel indication.

Unit Check: When a unit-check condition occurs, a sense operation is performed at the device; according to the resulting sense information, the 1620 I/O indicators concerned are set "on". If no indicators are involved in the operation, the standard error recovery procedures are performed.

DISK SIMULATION ROUTINES

The following sections present the technique used in the I/O simulation section of SIM20 to simulate 1620 disk operations.

Sector Arrangement

On the disk packs for 1311 Disk Storage Drives, the 20 sectors of any one track are arranged consecutively from 0 to 19. On the disk packs for 2311 Disk Storage Drives, the sectors are rearranged as shown in Figure 8.

0	7	14	1	8	15	2	9	16	3	10	17	4	11	18	5	12	19	6	13	20

Figure 8. Disk Sector Arrangement

This rearrangement of sectors allows efficient simulation of 1620 disk operations involving a number of consecutive 1311 sectors in logical sequence. For example, the processing of a series of sectors during a 1620 Read Disk operation is simulated in the following way:

1. The first sector in the series is read into the I/O buffer.

2. A certain amount of CPU processing is performed to check the results of the operation.

3. The data is transferred from the I/O buffer into simulated 1620 core storage.

4. A counter is incremented in order to read the next sector in the series.

While steps 2, 3, and 4 are being executed, the two sectors immediately following the sector just read pass under the read/write head of the disk unit. For this reason, the sectors are arranged in the order:

n,n+7,n+14,n+1,n+8,n+15,...

The result is that when SIM20 is ready to accept the next sector in logical sequence, that sector is the next one to pass under the read/write head.

This rearrangement of sectors reduces the processing time for multi-sector operations from 4.25 ms per sector (1311 Disk Storage Drive) to 3.57 ms per sector.

Each sector on a 2311 track is made up of the following elements:

- Identification
- Key length (=0)
- Count (=105 bytes)
- 105 bytes of data (the first 5 bytes contain the sector address, and the last 100 the contents of the sector)

An additional 21st sector is placed at the end of each 2311 track; it contains the 1620 addresses of the 20 preceding sectors. Since each 1620 sector address is 5 bytes long, this sector contains 100 bytes. It is used by SIM20 to check the addresses of the sectors on the track; when the sector addresses are modified as a result of a 1620 disk operation, its contents are changed accordingly.

Disk Indicators

The 1620/1311 indicators

36 (address check)
37 (WLRC/RBC)
38 (cylinder overflow)

and the read- and write-check indicators 06 and 07 are simulated by bits in System/360 main storage. These indicators are set by the SIM20 sequences which perform address comparison on the 21st sector, check cylinder overflow by arithmetic computation, and perform a wrong-length record check by character comparison.

Write Address Switch

The write-address switch is simulated by bit 5 of the byte at System/360 address 00001. When "on", it allows 1620 programs to modify the addresses of sectors on 2311 tracks; when the addresses are modified, the contents of the 21st sector of the track are changed accordingly.

Protection Flag

As on the 1620, write protection in SIM20 consists in placing a flag on the leftmost position of the sector address; that is, in the first of the 105 data bytes.

Disk Addresses

SIM20 determines the 2311 cylinder and head numbers by converting the disk control field specified by the 1620 disk instruction. The addresses thus defined are placed in System/360 main storage so that they can be used by further 1620 disk instructions.

Disk Read, Write, and Check Instructions

When one of these 1620 disk instructions is encountered, the Disk Operation Entry routine (see Chart BM) is called. This routine makes a number of tests:

- Read, write, or check operation (Charts BP,BQ,BR)
- Track or sector mode
- With or without WLRC/RBC (wrong-length-record check, read-back check)

Depending on the result of these tests, it then passes control to the appropriate routine.

Sector Mode Operations: A loop processes, one at a time, the sectors of a sequence, regardless of the length of the sequence. The routine increments a counter as necessary, and checks the results of the operation from sector to sector and track to track.

Track Mode Operations: These are also performed sector by sector (including the processing of sector addresses) until the 20th sector has been processed. This requires three rotations because of the disk sector arrangement chosen.

Seek Operations

The simulation of a 1620 seek operation (see Chart BN) is performed in the following steps:

1. Conversion of the contents of the disk control field
2. Issuance of a System/360 Seek Cylinder and Head command

CONSOLE SIMULATION

SIMULATED KEYS, SWITCHES, AND INDICATORS

All 1620 console keys, switches, and indicators are simulated by bits in a double-word at System/360 address 00000, since SIM20 tests these bits when the simulated 1620 is in the Manual mode. The functions of simulated keys, switches, and indicators are simulated only when the simulated 1620 is in the Manual mode. The operator can change the settings of keys and switches directly by manipulating System/360 control panel switches. SIM20 can also display the status of 1620 indicators in such a way that the operator can read them directly on the system control panel.

Simulated Console Keys

1620 Model 1 and Model 2 console keys are simulated by seven bits in the byte with System/360 address 00000.

- The Start and Stop keys (bit 0 of the byte) are simulated by the same System/360 control panel switch:

  OFF=Start key (bit contains zero)
  ON=Stop key (bit contains one)

- The Automatic Load key of the 1622 Card Reader is simulated by bit 6 of the byte. Its function is simulated only when the simulated 1620 is in the Manual mode. Setting the bit to one corresponds to pressing the Automatic Load key.

The correspondence between simulated 1620 console keys and the bits of byte 00000 is given in the listing of SIM20.

Simulated Console Switches

1620 Model 1 and Model 2 console switches are simulated by eight bits in the byte with System/360 address 00001. The first four bits of this byte correspond to 1620 program switches 1, 2, 3, and 4; the following bit, to the disk-check switch; and the last two, to the I/O-check switch (read check, write check) and the overflow switch (arithmetic check, exponent check).

The function of the 1620 parity bit is simulated by the parity bit of the appropriate System/360 byte; therefore, the 1620 parity-check switch is not simulated.

Bit 5 of byte 00001 simulates the write-address switch, which is used by 1620 programs that modify sector addresses on 1311 Disk Storage Drives. Its effect is simulated only during 1620 disk operations.

The correspondence between simulated 1620 console switches and the bits of byte 00001 is given in the listing of SIM20.

Simulated Indicators

1620 Model 1 and Model 2 indicators are simulated by 17 bits in System/360 main storage at addresses 00002 through 00007. Indicators 19 and 39, which summarize the other indicators, are not simulated. When a BI or BNI instruction requests that indicator 19 or 39 be tested, the corresponding detailed indicator bits (indicators 06+07+25+36+37+38) are tested simultaneously.

The third bit of byte 00002 is used as the "paper tape switch." If this bit contains a 1, the 1621 Paper Tape Reader is simulated by a card reader; if it contains zero, the 1621 is simulated by the 2671 Paper Tape Reader.

LOGIC OF KEY SIMULATION

The simulation of 1620 console keys (except the Stop key) is effective only when the simulated 1620 is in the Manual mode.

The BIR, which is given control whenever a 1620 instruction has been completely processed, tests the value of the start/stop bit (bit 0) at System/360 address 00000. If this bit contains a zero, simulation continues with the next 1620 instruction in sequence. If this bit contains a one, control is passed to a closed loop, which tests successively the value of the save, reset, check reset, insert, modify, and automatic load bits. If all these bits contain zeros, the sequence loops endlessly. As soon as a value of one is encountered, the corresponding functions are performed. These bits can be set by storing them in System/360 main storage through the system control panel.

When one of the save, check reset, or insert operations is performed, control is returned to the closed loop, and the simulated 1620 remains in the Manual mode. The only way to exit from this loop is to perform a Start or Automatic Load operation; that is, to set the start/stop bit to zero. Control is then returned to the BIR, which resumes simulation of 1620 instructions.

When the automatic load bit contains a one, control is passed to the Read Numerically (Card) sequence, which reads the first card into simulated 1620 core storage, at address 00000, sets the contents of the simulated instruction counter to 00000, and sets the start/stop bit to zero, withdrawing control from the closed loop. Loading then begins immediately.

When the modify bit contains a one, the corresponding function is performed (all simulated 1620 core storage is set to zero), provided that the start/stop bit contains a zero. When simulated core storage has been cleared, the start/stop bit is set to one, and the closed loop is re-entered.

The logic of key simulation is shown in Charts BA, BK, and BL.

MESSAGES AND COMMANDS

All I/O operations on the 1052 Printer-Keyboard are performed by the TYPIO subroutine, which may be given control at any time, provided that it has been given the necessary information: a device address, a read or write command, a count, and the address of a buffer. This general purpose routine is used by:

- The MESSAG subroutine, which prints all SIM20 messages on the 1052 Printer-Keyboard
- The ALARM sequence, which performs the error recovery procedures for the 1052
- The INSERT, RNTY, RATY, WNTY, WATY, and DNTY sequences, which simulate 1620 console typewriter operations

MESSAG Subroutine

The MESSAG subroutine is given control whenever a part of SIM20 must display a message on the 1052 Printer-Keyboard. It contains three options:

1. Send a message and continue simulation.
2. Send a message and stop the BIR.
3. Send an INTERVENTION REQUIRED message (such as Reader No Feed), stop the BIR, and return control to the BIR without incrementing the simulated instruction counter. The operator can then re-start simulation on the same 1620 instruction as soon as the required device is ready.

The MESSAG subroutine processes every message, provided it is given the message address (that is, the first character of the string representing the message). This character must contain the length of the message proper, which begins with the next byte.

RNTY Sequence

The RNTY sequence simulates the 1620 Insert, Read Numerically (Typewriter), and Read Alphamerically (Typewriter) instructions. All three instructions are processed using modifier switches.

An Insert operation issues a read command on the 1052, with a count of 100 bytes. These 100 bytes are converted into SIM20 internal code and placed in simulated 1620 core storage, starting at address 00000. The closed loop of the BIR is then re-entered, ready for a start operation.

A Read Numerically or Alphamerically operation is processed by sending commands to read 100 bytes to the 1052, converting these bytes, and placing them in simulated 1620 core storage. This process is repeated until an end-of-block indication is encountered.

Note that an Insert, a Read Numerically, or a Read Alphamerically operation must not be terminated by a Release and Start operation, as is done on a 1620, because the end-of-block indication is the only one that can release the 1052 Printer-Keyboard, by sending channel-end and device-end indications to SIM20.

Write Numerically and Alphamerically Operations

These operations are performed in the same way as the insert and read operations: by sending commands to the 1052 to write strings of 100 bytes until a record mark is detected in simulated 1620 core storage.

Chart BA. SIM20 Internal Logic

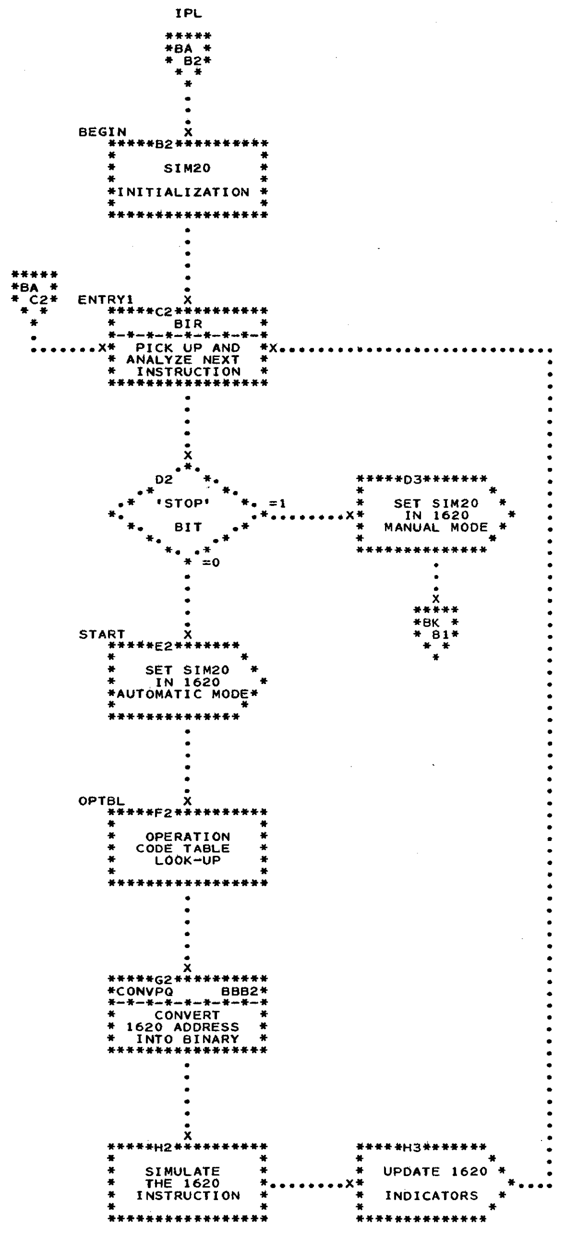
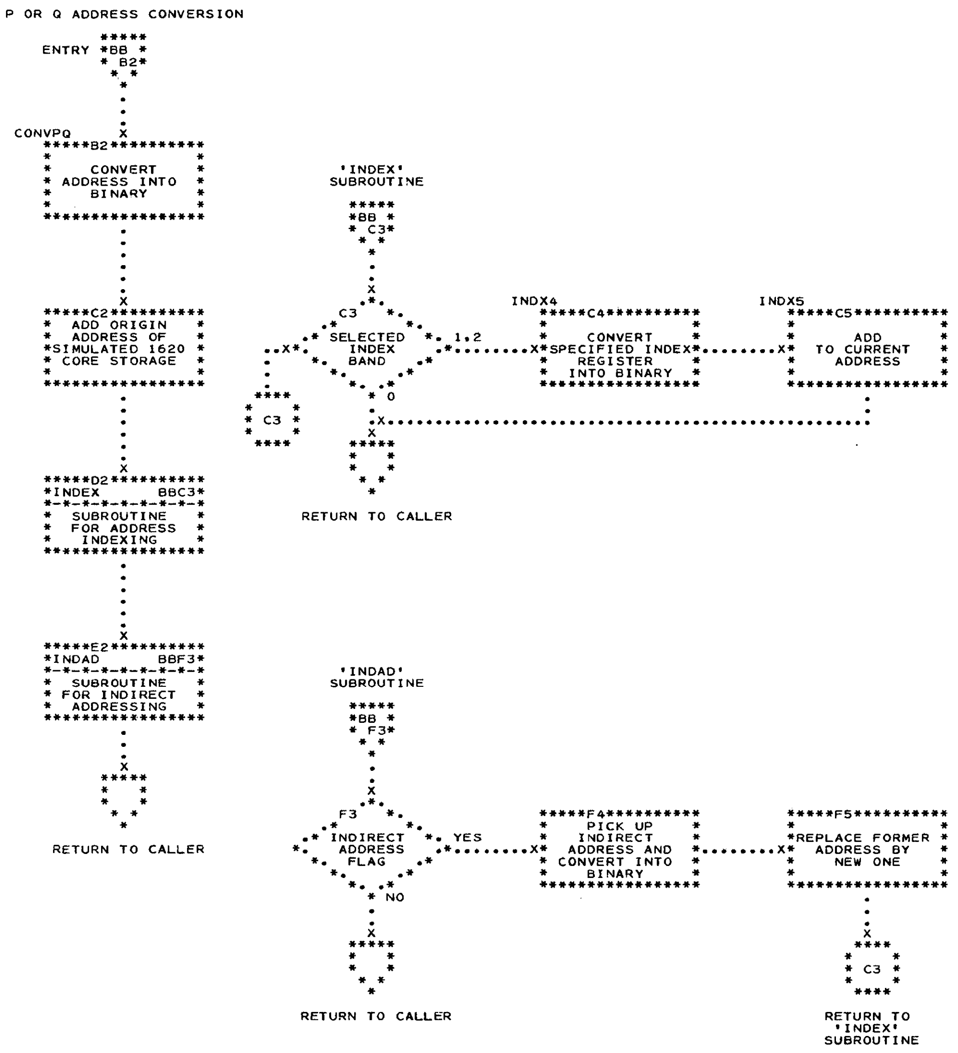
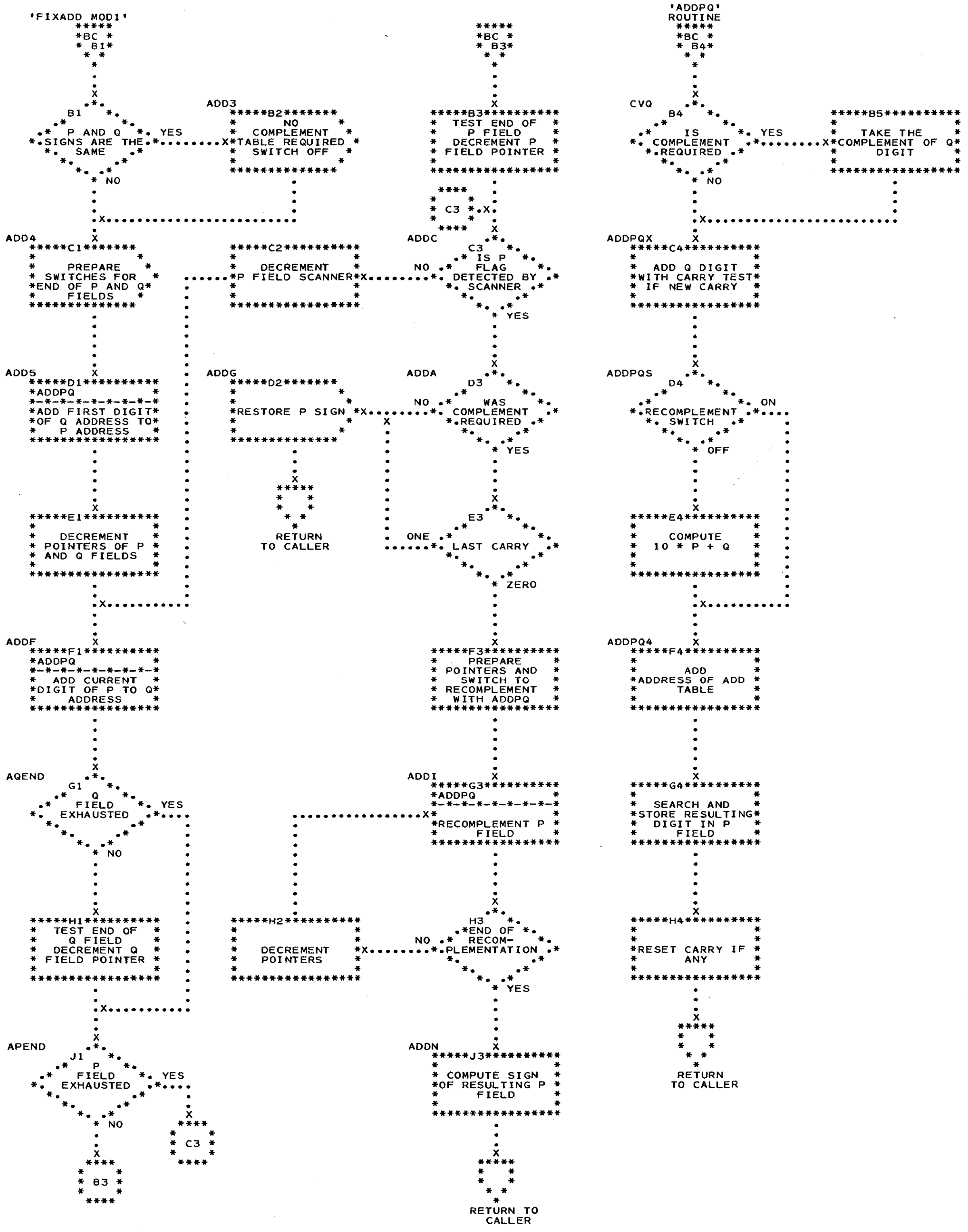

IPL
BA B2
BEGIN
B2
SIM20 INITIALIZATION
BA C2
ENTRY1
C2
BIR
PICK UP AND ANALYZE NEXT INSTRUCTION
D2
'STOP' BIT
=1
=0
D3
SET SIM20 IN 1620 MANUAL MODE
BK B1
START
E2
SET SIM20 IN 1620 AUTOMATIC MODE
OPTBL
F2
OPERATION CODE TABLE LOOK-UP
G2
CONVPQ BBB2
CONVERT 1620 ADDRESS INTO BINARY
H2
SIMULATE THE 1620 INSTRUCTION
H3
UPDATE 1620 INDICATORS

Chart BB. Address Conversion Subroutines


P OR Q ADDRESS CONVERSION
ENTRY BB B2
CONVPQ
B2 CONVERT ADDRESS INTO BINARY
C2 ADD ORIGIN ADDRESS OF SIMULATED 1620 CORE STORAGE
D2 INDEX BBC3 SUBROUTINE FOR ADDRESS INDEXING
E2 INDAD BBF3 SUBROUTINE FOR INDIRECT ADDRESSING
RETURN TO CALLER
'INDEX' SUBROUTINE
BB C3
C3 SELECTED INDEX BAND
1,2
0
C3
INDX4
C4 CONVERT SPECIFIED INDEX REGISTER INTO BINARY
INDX5
C5 ADD TO CURRENT ADDRESS
RETURN TO CALLER
'INDAD' SUBROUTINE
BB F3
F3 INDIRECT ADDRESS FLAG
YES
NO
F4 PICK UP INDIRECT ADDRESS AND CONVERT INTO BINARY
F5 REPLACE FORMER ADDRESS BY NEW ONE
C3
RETURN TO CALLER
RETURN TO 'INDEX' SUBROUTINE

NOTE = THIS FLOWCHART CORRESPONDS TO THE MAXIMUM POSSIBLE 1620 CPU CONFIGURATION. IN CERTAIN SMALLER CONFIGURATIONS, 'INDEX' OR 'INDAD' SUBROUTINES MAY BE ABSENT.

Chart BC. FIXADD Routine



'FIXADD MOD1'
BC B1
B1 P AND Q SIGNS ARE THE SAME
YES
ADD3
B2 NO COMPLEMENT TABLE REQUIRED SWITCH OFF
NO
ADD4
C1 PREPARE SWITCHES FOR END OF P AND Q FIELDS
ADD5
D1 ADDPQ ADD FIRST DIGIT OF Q ADDRESS TO P ADDRESS
E1 DECREMENT POINTERS OF P AND Q FIELDS
ADDF
F1 ADDPQ ADD CURRENT DIGIT OF P TO Q ADDRESS
AQEND
G1 Q FIELD EXHAUSTED
YES
NO
H1 TEST END OF Q FIELD DECREMENT Q FIELD POINTER
APEND
J1 P FIELD EXHAUSTED
YES
NO
C3
B3
C2 DECREMENT P FIELD SCANNER
ADDG
D2 RESTORE P SIGN
RETURN TO CALLER
H2 DECREMENT POINTERS
BC B3
B3 TEST END OF P FIELD DECREMENT P FIELD POINTER
C3
ADDC
C3 IS P FLAG DETECTED BY SCANNER
NO
YES
ADDA
D3 WAS COMPLEMENT REQUIRED
NO
YES
E3 LAST CARRY
ONE
ZERO
F3 PREPARE POINTERS AND SWITCH TO RECOMPLEMENT WITH ADDPQ
ADDI
G3 ADDPQ RECOMPLEMENT P FIELD
H3 END OF RECOM- PLEMENTATION
NO
YES
ADDN
J3 COMPUTE SIGN OF RESULTING P FIELD
RETURN TO CALLER
'ADDPQ' ROUTINE
BC B4
CVQ
B4 IS COMPLEMENT REQUIRED
YES
NO
B5 TAKE THE COMPLEMENT OF Q DIGIT
ADDPQX
C4 ADD Q DIGIT WITH CARRY TEST IF NEW CARRY
ADDPQS
D4 RECOMPLEMENT SWITCH
ON
OFF
E4 COMPUTE 10 * P + Q
ADDPQ4
F4 ADD ADDRESS OF ADD TABLE
G4 SEARCH AND STORE RESULTING DIGIT IN P FIELD
H4 RESET CARRY IF ANY
RETURN TO CALLER

Chart BD. MULT Routine

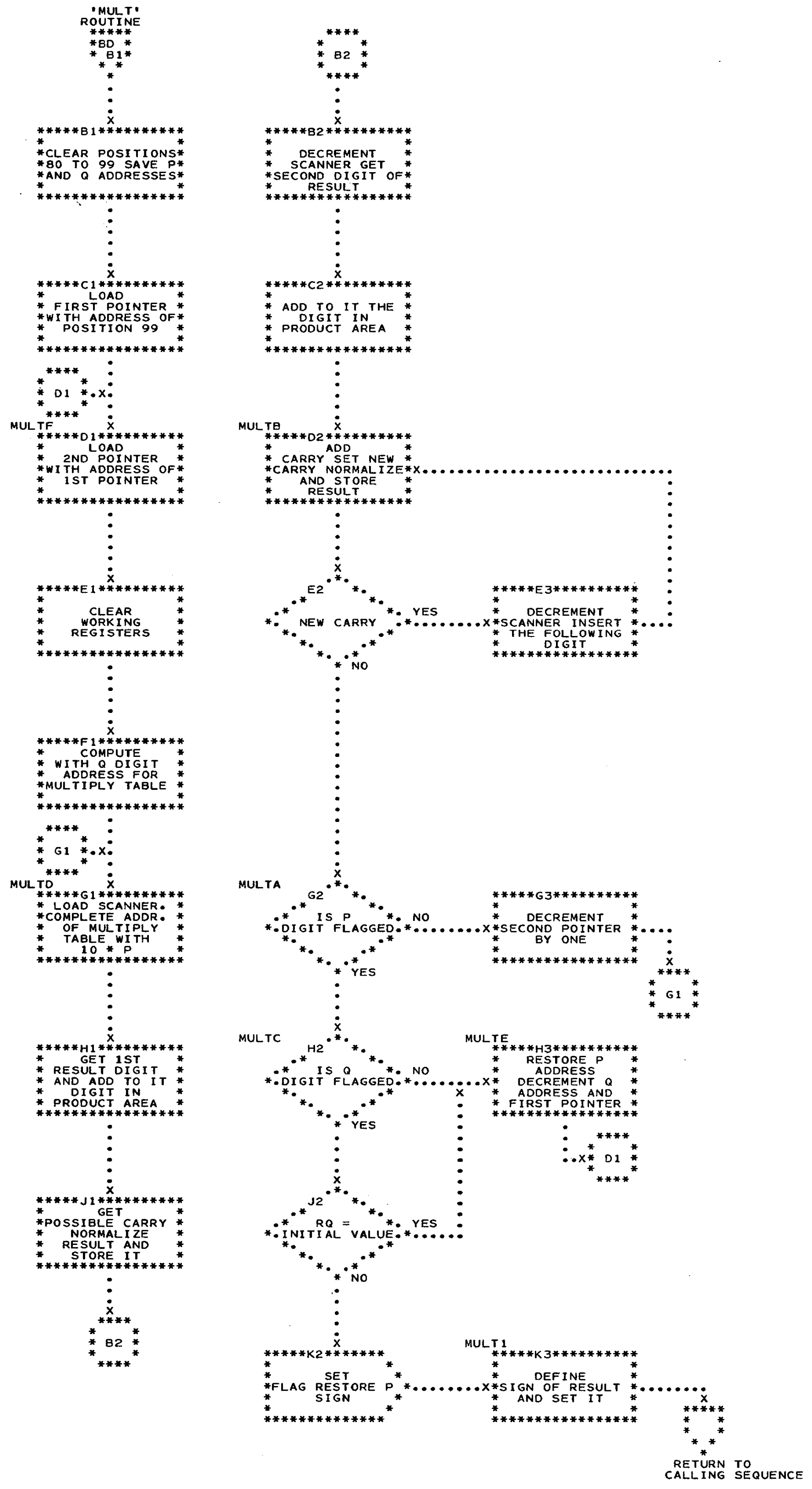
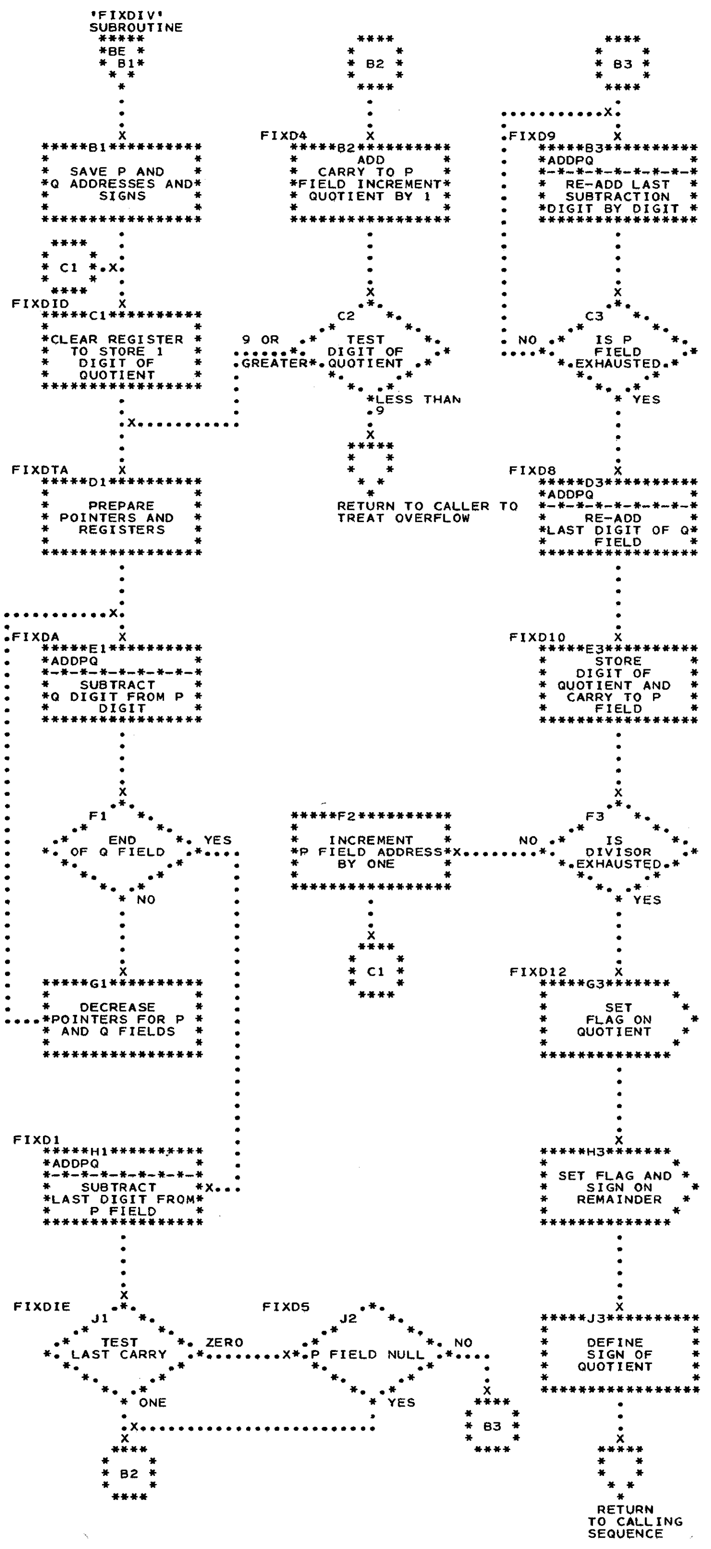

'MULT' ROUTINE
BD B1
B1
CLEAR POSITIONS 80 TO 99 SAVE P AND Q ADDRESSES
C1
LOAD FIRST POINTER WITH ADDRESS OF POSITION 99
D1
MULTF
D1
LOAD 2ND POINTER WITH ADDRESS OF 1ST POINTER
E1
CLEAR WORKING REGISTERS
F1
COMPUTE WITH Q DIGIT ADDRESS FOR MULTIPLY TABLE
G1
MULTD
G1
LOAD SCANNER. COMPLETE ADDR. OF MULTIPLY TABLE WITH 10 * P
H1
GET 1ST RESULT DIGIT AND ADD TO IT DIGIT IN PRODUCT AREA
J1
GET POSSIBLE CARRY NORMALIZE RESULT AND STORE IT
B2
B2
B2
DECREMENT SCANNER GET SECOND DIGIT OF RESULT
C2
ADD TO IT THE DIGIT IN PRODUCT AREA
MULTB
D2
ADD CARRY SET NEW CARRY NORMALIZE AND STORE RESULT
E2
NEW CARRY
YES
NO
E3
DECREMENT SCANNER INSERT THE FOLLOWING DIGIT
MULTA
G2
IS P DIGIT FLAGGED
NO
YES
G3
DECREMENT SECOND POINTER BY ONE
G1
MULTC
H2
IS Q DIGIT FLAGGED
NO
YES
MULTE
H3
RESTORE P ADDRESS DECREMENT Q ADDRESS AND FIRST POINTER
D1
J2
RQ = INITIAL VALUE
YES
NO
K2
SET FLAG RESTORE P SIGN
MULT1
K3
DEFINE SIGN OF RESULT AND SET IT
RETURN TO CALLING SEQUENCE

Chart BE. FIXDIV Routine



'FIXDIV' SUBROUTINE
BE B1
B2
B3
B1 SAVE P AND Q ADDRESSES AND SIGNS
FIXD4 B2 ADD CARRY TO P FIELD INCREMENT QUOTIENT BY 1
FIXD9 B3 ADDPQ RE-ADD LAST SUBTRACTION DIGIT BY DIGIT
C1
FIXD1D C1 CLEAR REGISTER TO STORE 1 DIGIT OF QUOTIENT
C2 TEST DIGIT OF QUOTIENT
9 OR GREATER
LESS THAN 9
RETURN TO CALLER TO TREAT OVERFLOW
C3 IS P FIELD EXHAUSTED
NO
YES
FIXDTA D1 PREPARE POINTERS AND REGISTERS
FIXD8 D3 ADDPQ RE-ADD LAST DIGIT OF Q FIELD
FIXDA E1 ADDPQ SUBTRACT Q DIGIT FROM P DIGIT
FIXD10 E3 STORE DIGIT OF QUOTIENT AND CARRY TO P FIELD
F1 END OF Q FIELD
YES
NO
F2 INCREMENT P FIELD ADDRESS BY ONE
C1
F3 IS DIVISOR EXHAUSTED
NO
YES
G1 DECREASE POINTERS FOR P AND Q FIELDS
FIXD12 G3 SET FLAG ON QUOTIENT
FIXD1 H1 ADDPQ SUBTRACT LAST DIGIT FROM P FIELD
H3 SET FLAG AND SIGN ON REMAINDER
FIXD1E J1 TEST LAST CARRY
ZERO
ONE
FIXD5 J2 P FIELD NULL
NO
YES
B3
B2
J3 DEFINE SIGN OF QUOTIENT
RETURN TO CALLING SEQUENCE

Chart BF. Simulation of I/O (Other Than Disk) Operations

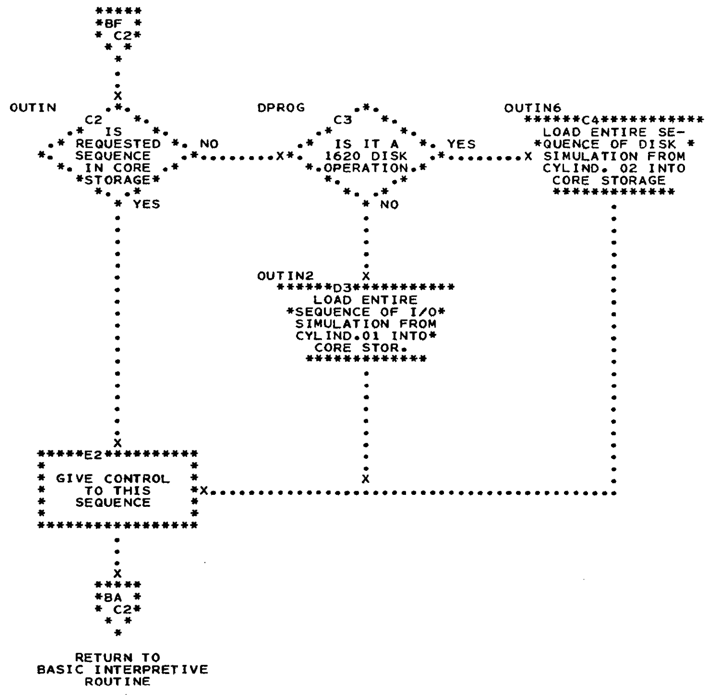

BF C2
OUTIN
C2
IS REQUESTED SEQUENCE IN CORE STORAGE
NO
YES
DPROG
C3
IS IT A 1620 DISK OPERATION
YES
NO
OUTIN6
C4
LOAD ENTIRE SE- QUENCE OF DISK SIMULATION FROM CYLIND. 02 INTO CORE STORAGE
OUTIN2
D3
LOAD ENTIRE SEQUENCE OF I/O SIMULATION FROM CYLIND.01 INTO CORE STOR.
E2
GIVE CONTROL TO THIS SEQUENCE
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE

Chart BG. Logic of Read Operations

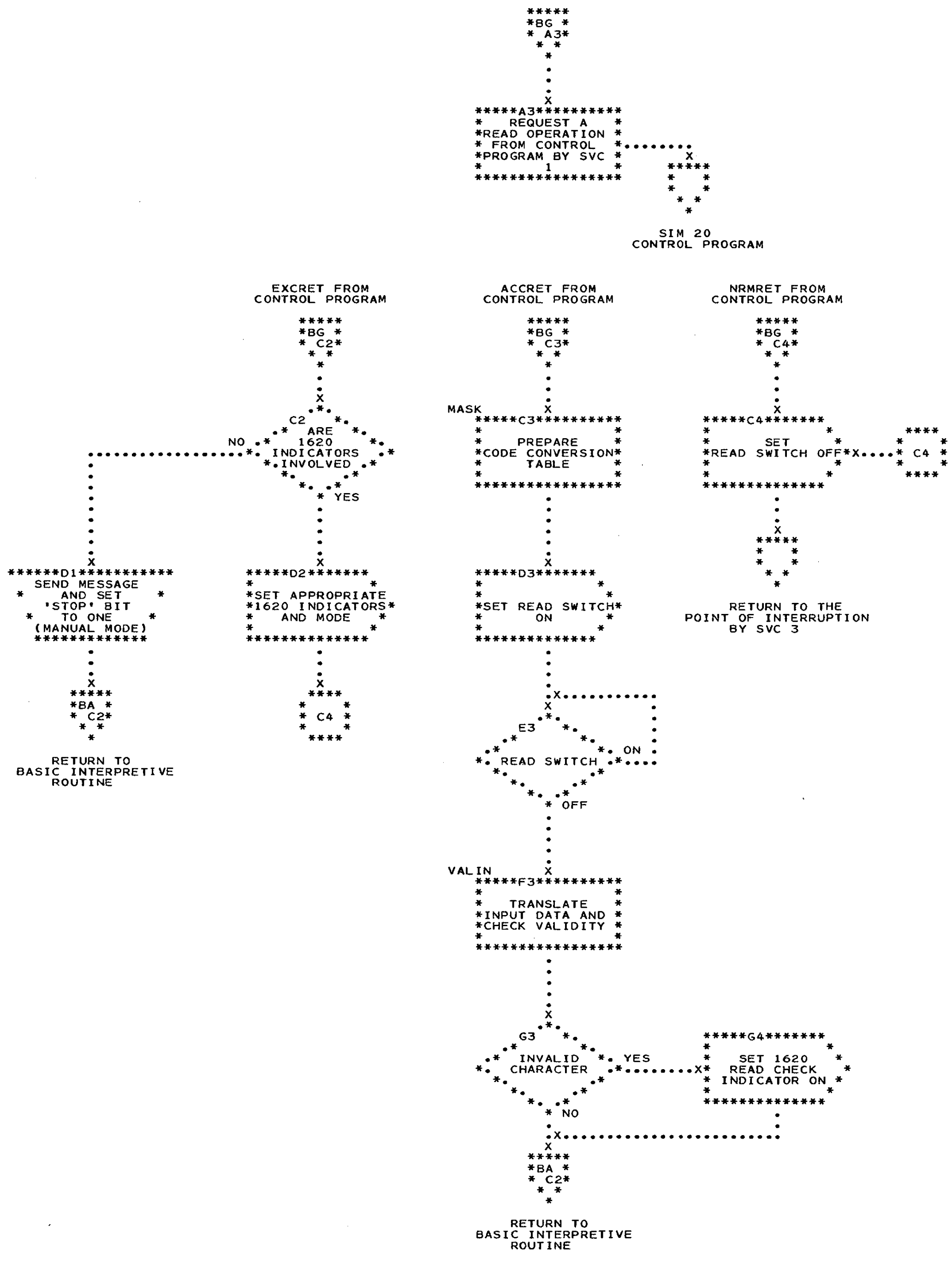

BG A3
A3 REQUEST A READ OPERATION FROM CONTROL PROGRAM BY SVC 1
SIM 20 CONTROL PROGRAM
EXCRET FROM CONTROL PROGRAM
BG C2
C2 ARE 1620 INDICATORS INVOLVED
NO
YES
D1 SEND MESSAGE AND SET 'STOP' BIT TO ONE (MANUAL MODE)
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
D2 SET APPROPRIATE 1620 INDICATORS AND MODE
C4
ACCRET FROM CONTROL PROGRAM
BG C3
MASK
C3 PREPARE CODE CONVERSION TABLE
D3 SET READ SWITCH ON
E3 READ SWITCH
ON
OFF
VALIN
F3 TRANSLATE INPUT DATA AND CHECK VALIDITY
G3 INVALID CHARACTER
YES
NO
G4 SET 1620 READ CHECK INDICATOR ON
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
NRMRET FROM CONTROL PROGRAM
BG C4
C4 SET READ SWITCH OFF
C4
RETURN TO THE POINT OF INTERRUPTION BY SVC 3

Chart BH. Logic of Write Operations

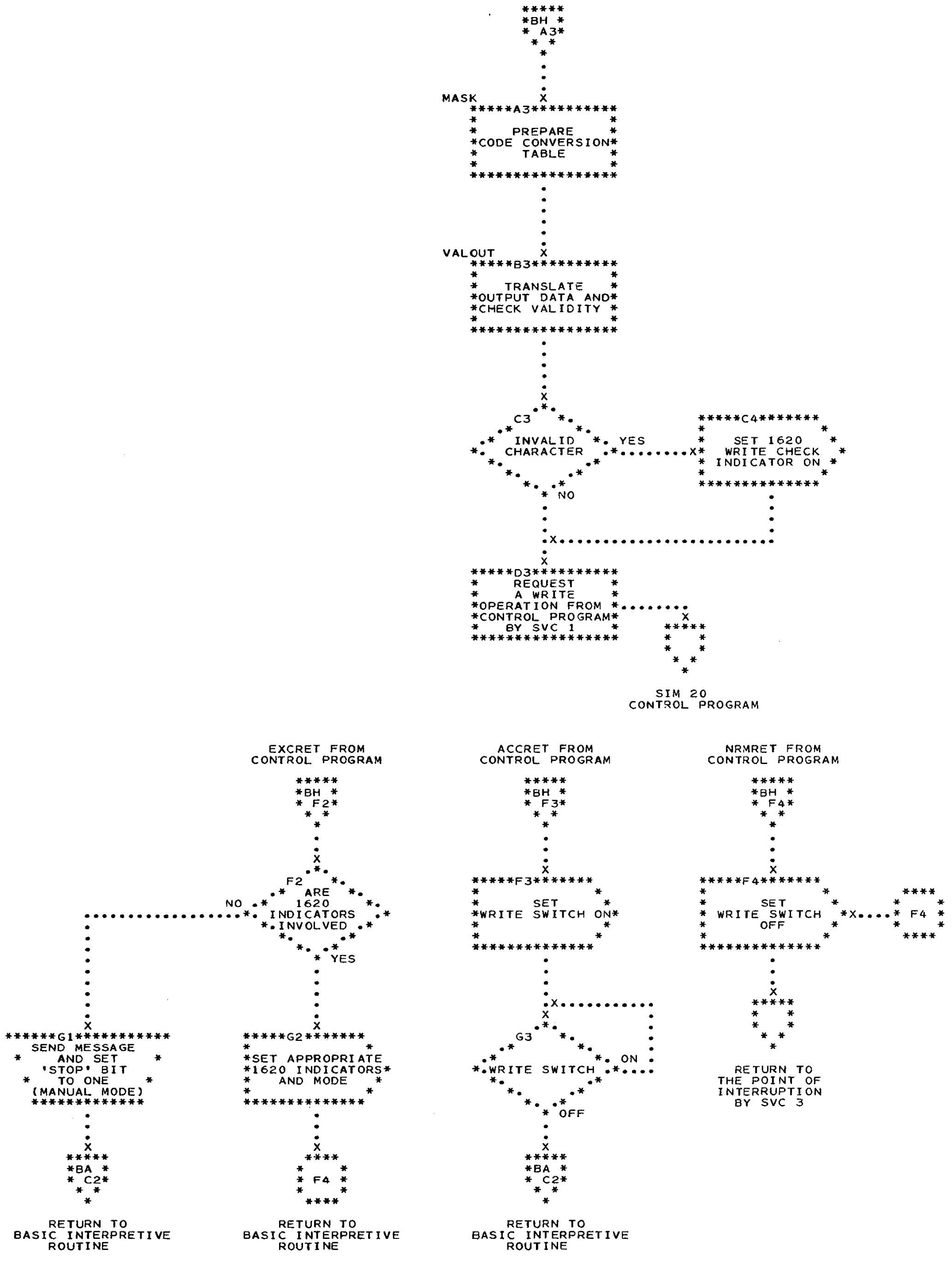
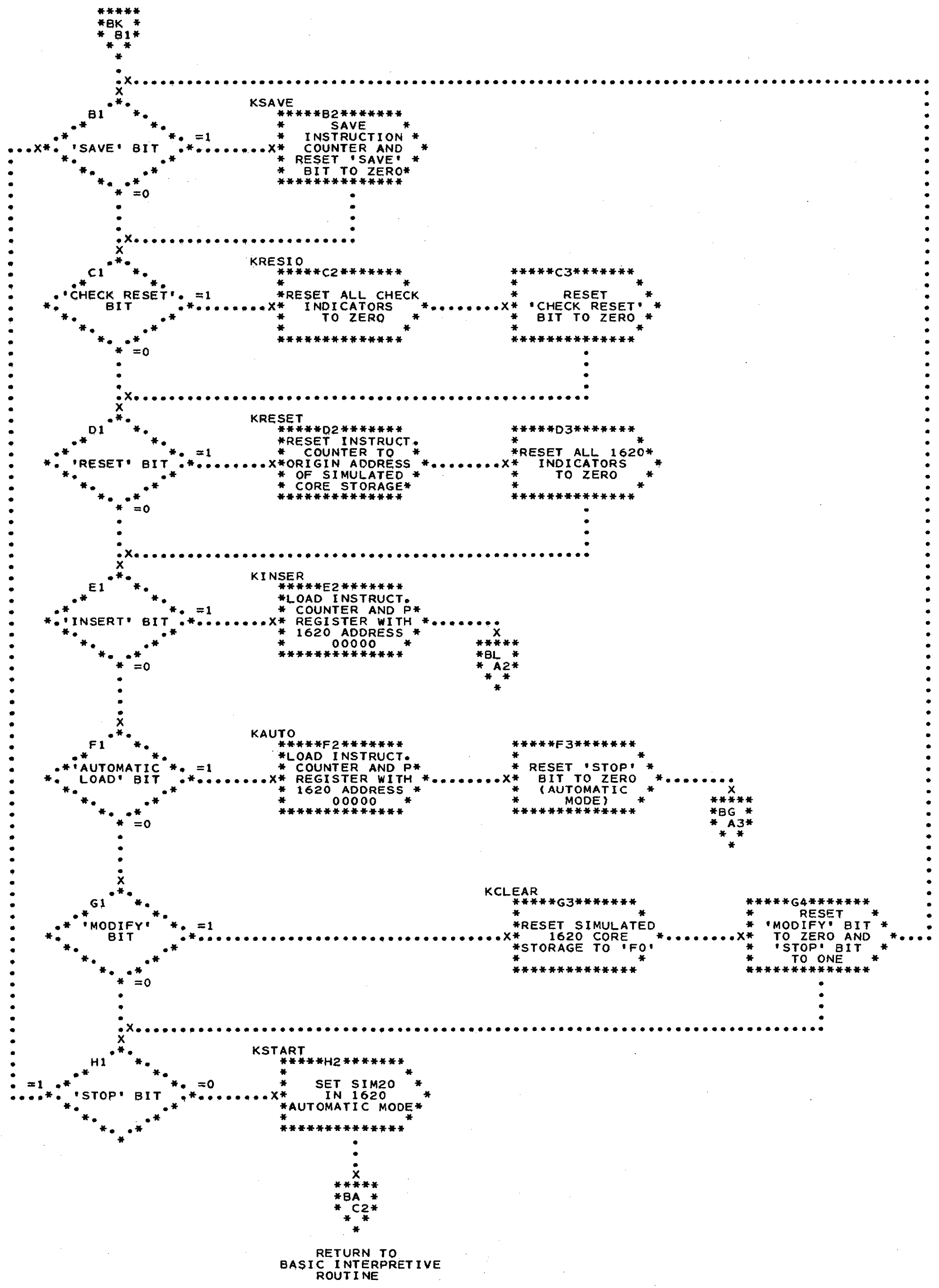
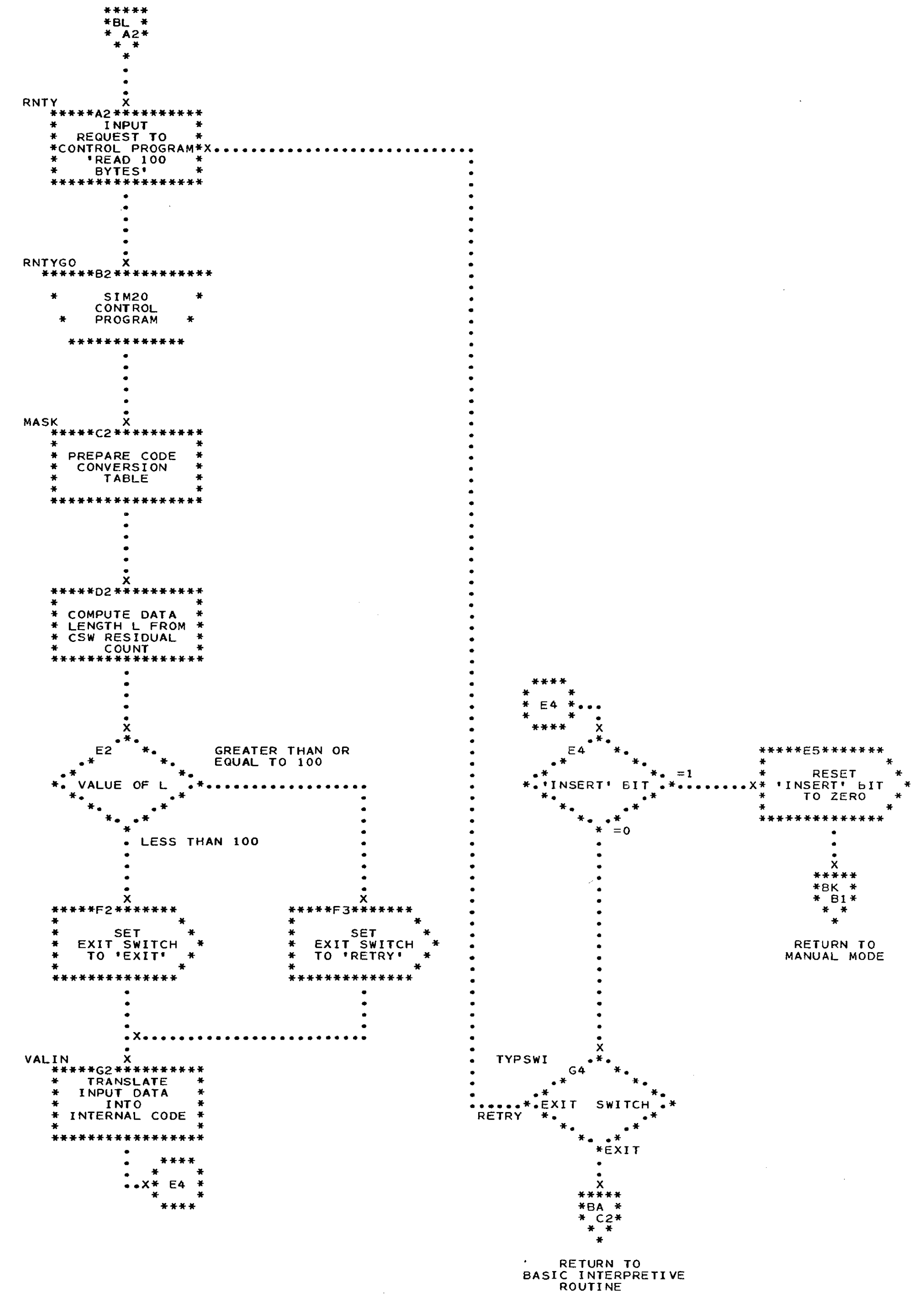

BH A3
MASK
A3 PREPARE CODE CONVERSION TABLE
VALOUT
B3 TRANSLATE OUTPUT DATA AND CHECK VALIDITY
C3 INVALID CHARACTER
YES
NO
C4 SET 1620 WRITE CHECK INDICATOR ON
D3 REQUEST A WRITE OPERATION FROM CONTROL PROGRAM BY SVC 1
SIM 20 CONTROL PROGRAM
EXCRET FROM CONTROL PROGRAM
BH F2
F2 ARE 1620 INDICATORS INVOLVED
NO
YES
G1 SEND MESSAGE AND SET 'STOP' BIT TO ONE (MANUAL MODE)
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
G2 SET APPROPRIATE 1620 INDICATORS AND MODE
F4
RETURN TO BASIC INTERPRETIVE ROUTINE
ACCRET FROM CONTROL PROGRAM
BH F3
F3 SET WRITE SWITCH ON
G3 WRITE SWITCH
ON
OFF
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
NRMRET FROM CONTROL PROGRAM
BH F4
F4 SET WRITE SWITCH OFF
F4
RETURN TO THE POINT OF INTERRUPTION BY SVC 3

Chart BK. Console Simulation Logic



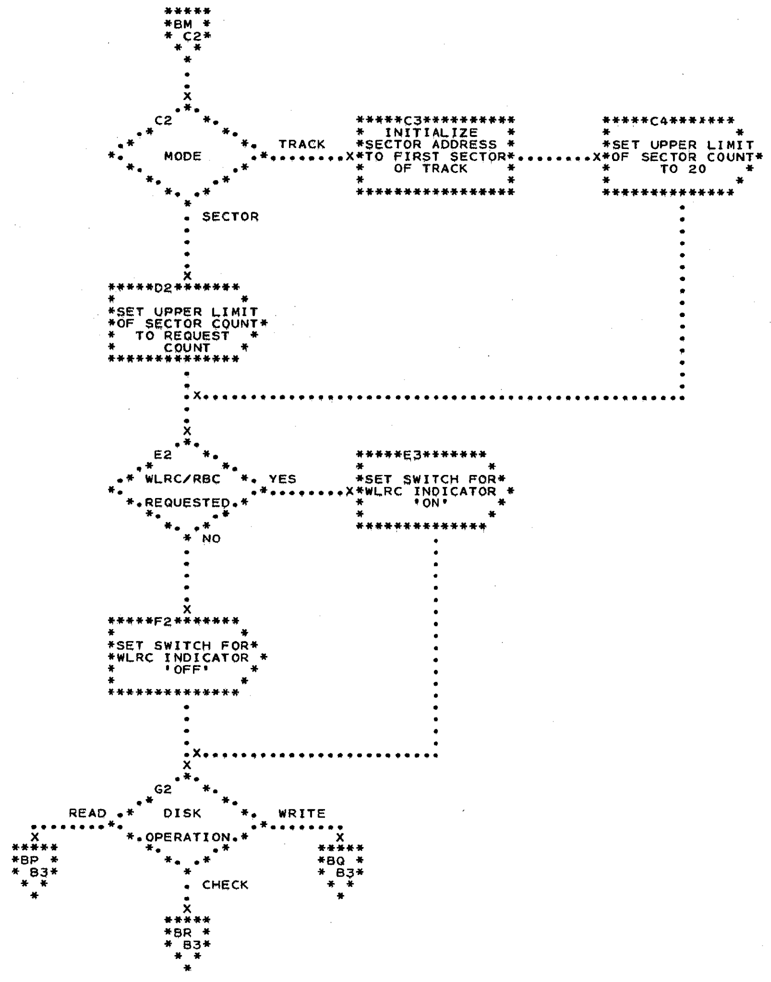
BK B1
B1 'SAVE' BIT =1 =0
KSAVE B2 SAVE INSTRUCTION COUNTER AND RESET 'SAVE' BIT TO ZERO
C1 'CHECK RESET' BIT =1 =0
KRESIO C2 RESET ALL CHECK INDICATORS TO ZERO
C3 RESET 'CHECK RESET' BIT TO ZERO
D1 'RESET' BIT =1 =0
KRESET D2 RESET INSTRUCT. COUNTER TO ORIGIN ADDRESS OF SIMULATED CORE STORAGE
D3 RESET ALL 1620 INDICATORS TO ZERO
E1 'INSERT' BIT =1 =0
KINSER E2 LOAD INSTRUCT. COUNTER AND P REGISTER WITH 1620 ADDRESS 00000
BL A2
F1 'AUTOMATIC LOAD' BIT =1 =0
KAUTO F2 LOAD INSTRUCT. COUNTER AND P REGISTER WITH 1620 ADDRESS 00000
F3 RESET 'STOP' BIT TO ZERO (AUTOMATIC MODE)
BG A3
G1 'MODIFY' BIT =1 =0
KCLEAR G3 RESET SIMULATED 1620 CORE STORAGE TO 'F0'
G4 RESET 'MODIFY' BIT TO ZERO AND 'STOP' BIT TO ONE
H1 'STOP' BIT =1 =0
KSTART H2 SET SIM20 IN 1620 AUTOMATIC MODE
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE

Chart BL. Insert Key Simulation


BL A2
RNTY
A2 INPUT REQUEST TO CONTROL PROGRAM 'READ 100 BYTES'
RNTYGO
B2 SIM20 CONTROL PROGRAM
MASK
C2 PREPARE CODE CONVERSION TABLE
D2 COMPUTE DATA LENGTH L FROM CSW RESIDUAL COUNT
E2 VALUE OF L
GREATER THAN OR EQUAL TO 100
LESS THAN 100
F2 SET EXIT SWITCH TO 'EXIT'
F3 SET EXIT SWITCH TO 'RETRY'
VALIN
G2 TRANSLATE INPUT DATA INTO INTERNAL CODE
E4
E4
E4 'INSERT' BIT
=1
=0
E5 RESET 'INSERT' BIT TO ZERO
BK B1
RETURN TO MANUAL MODE
TYPSWI
G4 EXIT SWITCH
RETRY
EXIT
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE

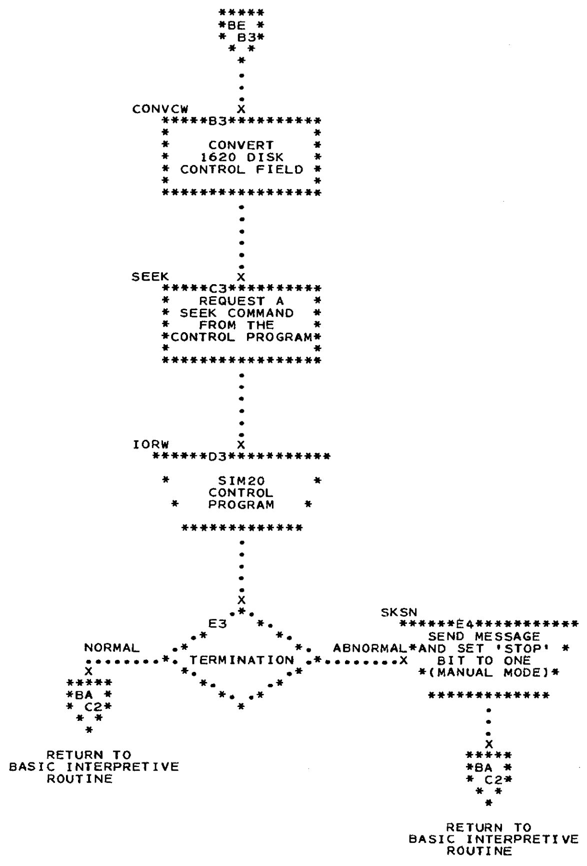
NOTE= THIS FLOWCHART IS ALSO VALID FOR
THE 'READ TYPEWRITER' OPERATION.

Chart BM. Disk Operation Entry Routine



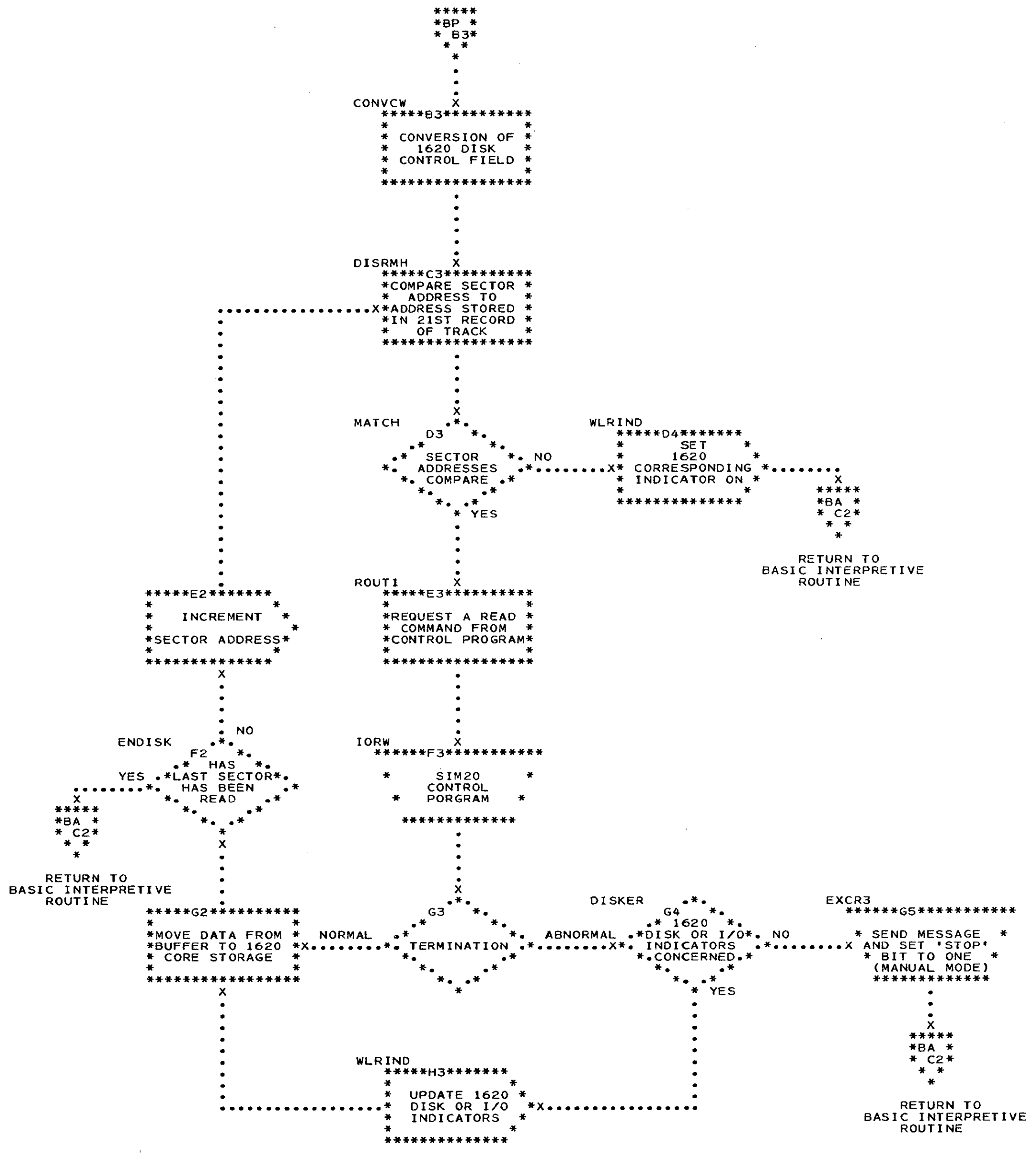
BM C2
C2 MODE
TRACK
C3 INITIALIZE SECTOR ADDRESS TO FIRST SECTOR OF TRACK
C4 SET UPPER LIMIT OF SECTOR COUNT TO 20
SECTOR
D2 SET UPPER LIMIT OF SECTOR COUNT TO REQUEST COUNT
E2 WLRC/RBC REQUESTED
YES
E3 SET SWITCH FOR WLRC INDICATOR 'ON'
NO
F2 SET SWITCH FOR WLRC INDICATOR 'OFF'
G2 DISK OPERATION
READ
WRITE
CHECK
BP B3
BQ B3
BR B3

Chart BN. Seek Disk Operations



BE B3
CONVCW
B3
CONVERT 1620 DISK CONTROL FIELD
SEEK
C3
REQUEST A SEEK COMMAND FROM THE CONTROL PROGRAM
IORW
D3
SIM20 CONTROL PROGRAM
E3
TERMINATION
NORMAL
ABNORMAL
SKSN
E4
SEND MESSAGE AND SET 'STOP' BIT TO ONE (MANUAL MODE)
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE

Chart BP. Read Disk Operations



BP B3
CONVCW
B3
CONVERSION OF 1620 DISK CONTROL FIELD
DISRMH
C3
COMPARE SECTOR ADDRESS TO ADDRESS STORED IN 21ST RECORD OF TRACK
MATCH
D3
SECTOR ADDRESSES COMPARE
NO
YES
WLRIND
D4
SET 1620 CORRESPONDING INDICATOR ON
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
E2
INCREMENT SECTOR ADDRESS
ROUT1
E3
REQUEST A READ COMMAND FROM CONTROL PROGRAM
ENDISK
F2
HAS LAST SECTOR HAS BEEN READ
NO
YES
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
IORW
F3
SIM20 CONTROL PORGRAM
G2
MOVE DATA FROM BUFFER TO 1620 CORE STORAGE
NORMAL
G3
TERMINATION
ABNORMAL
DISKER
G4
1620 DISK OR I/O INDICATORS CONCERNED
NO
YES
EXCR3
G5
SEND MESSAGE AND SET 'STOP' BIT TO ONE (MANUAL MODE)
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
WLRIND
H3
UPDATE 1620 DISK OR I/O INDICATORS

Chart BQ. Write Disk Operations

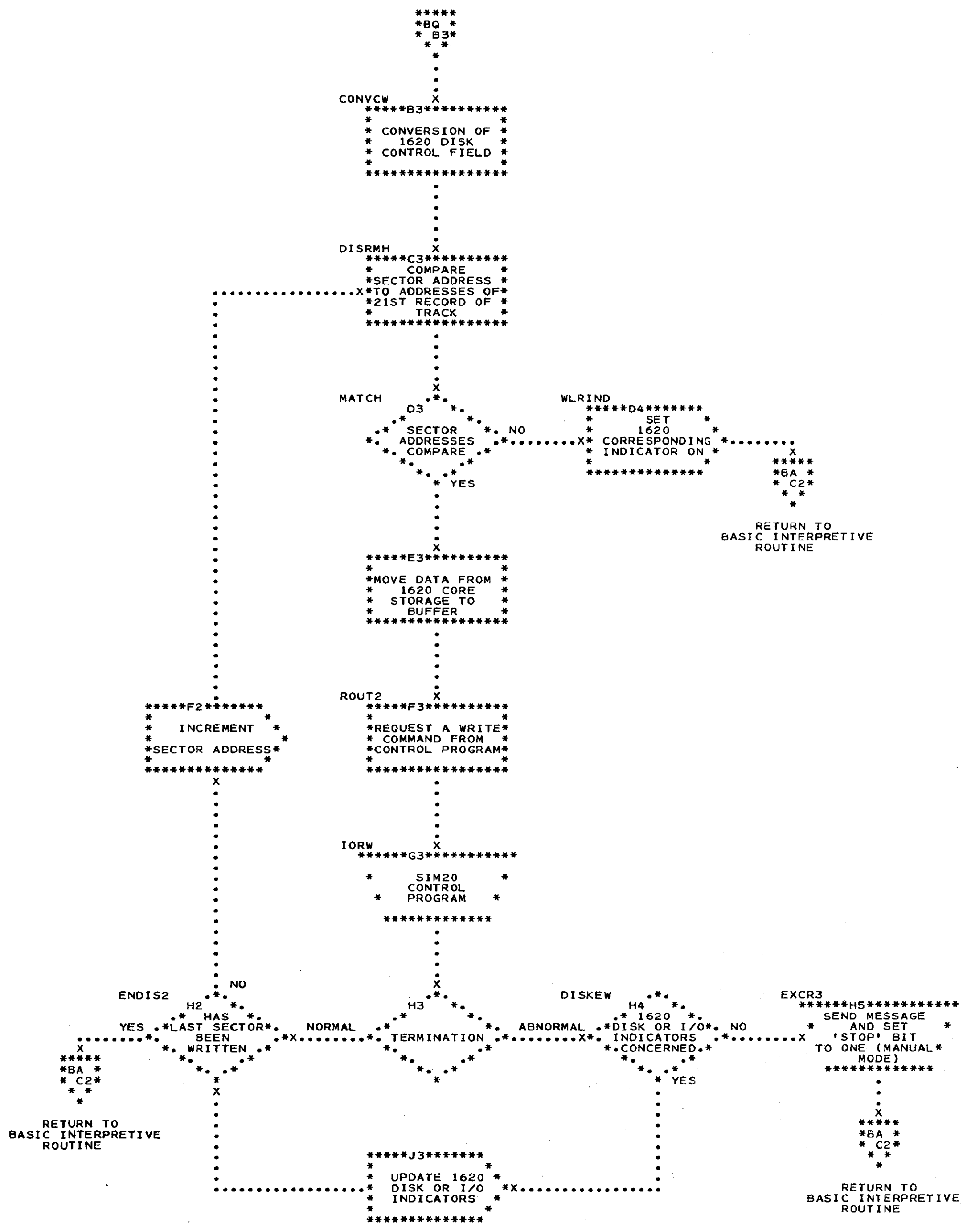
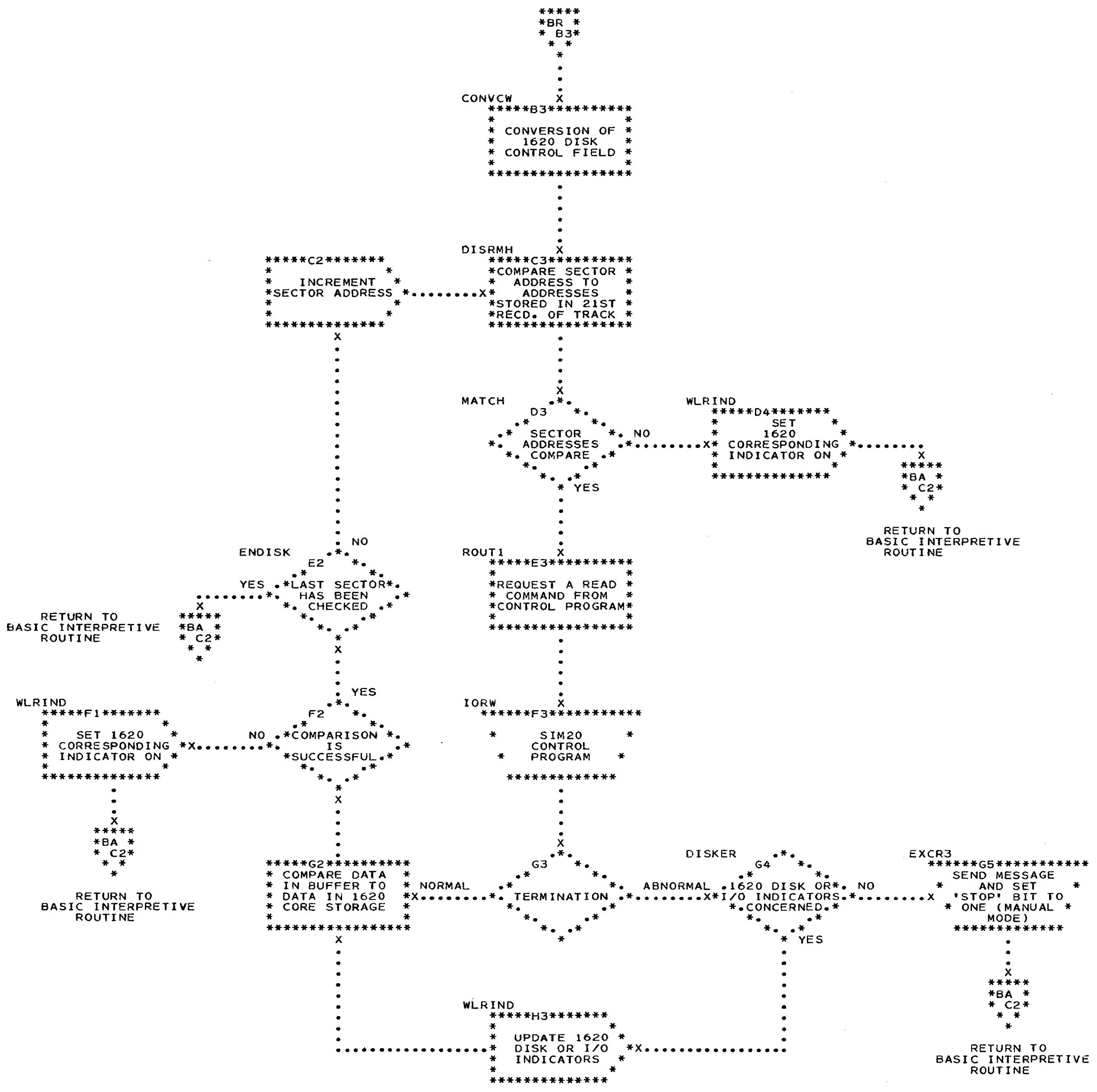

BQ B3
CONVCW B3
CONVERSION OF 1620 DISK CONTROL FIELD
DISRMH C3
COMPARE SECTOR ADDRESS TO ADDRESSES OF 21ST RECORD OF TRACK
MATCH D3
SECTOR ADDRESSES COMPARE
NO
YES
WLRIND D4
SET 1620 CORRESPONDING INDICATOR ON
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
E3
MOVE DATA FROM 1620 CORE STORAGE TO BUFFER
F2
INCREMENT SECTOR ADDRESS
ROUT2 F3
REQUEST A WRITE COMMAND FROM CONTROL PROGRAM
IORW G3
SIM20 CONTROL PROGRAM
ENDIS2 H2
HAS LAST SECTOR BEEN WRITTEN
NO
YES
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
H3
TERMINATION
NORMAL
ABNORMAL
DISKEW H4
1620 DISK OR I/O INDICATORS CONCERNED
NO
YES
EXCR3 H5
SEND MESSAGE AND SET 'STOP' BIT TO ONE (MANUAL MODE)
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
J3
UPDATE 1620 DISK OR I/O INDICATORS

Chart BR. Check Disk Operations


BR B3
CONVCW
B3
CONVERSION OF 1620 DISK CONTROL FIELD
DISRMH
C3
COMPARE SECTOR ADDRESS TO ADDRESSES STORED IN 21ST RECD. OF TRACK
C2
INCREMENT SECTOR ADDRESS
MATCH
D3
SECTOR ADDRESSES COMPARE
NO
YES
WLRIND
D4
SET 1620 CORRESPONDING INDICATOR ON
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
ENDISK
E2
LAST SECTOR HAS BEEN CHECKED
NO
YES
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
ROUT1
E3
REQUEST A READ COMMAND FROM CONTROL PROGRAM
WLRIND
F1
SET 1620 CORRESPONDING INDICATOR ON
F2
COMPARISON IS SUCCESSFUL
NO
YES
IORW
F3
SIM20 CONTROL PROGRAM
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
G2
COMPARE DATA IN BUFFER TO DATA IN 1620 CORE STORAGE
NORMAL
G3
TERMINATION
ABNORMAL
DISKER
G4
1620 DISK OR I/O INDICATORS CONCERNED
NO
YES
EXCR3
G5
SEND MESSAGE AND SET 'STOP' BIT TO ONE (MANUAL MODE)
BA C2
RETURN TO BASIC INTERPRETIVE ROUTINE
WLRIND
H3
UPDATE 1620 DISK OR I/O INDICATORS

EDITOR is an independent program used to create a symbolic version of SIM20, adapted to the particular 1620 system being simulated, and to the System/360 model on which SIM20 is to be run (see Charts CA and CB).

Using information from control cards, EDITOR extracts from the symbolic SIM20 tape distributed by IBM those routines needed to simulate a given 1620 system. Five types of control cards are used by EDITOR:

• CPU1	defines the 1620 CPU to be simulated.
• CPU2	defines the corresponding System/360 CPU.
• FEATURE	defines a 1620 special or optional feature to be simulated.
• DEVICE	defines a 1620 I/O device to be simulated and a corresponding System/360 I/O device.
• START	indicates the end of the control card deck and begins the editing process.

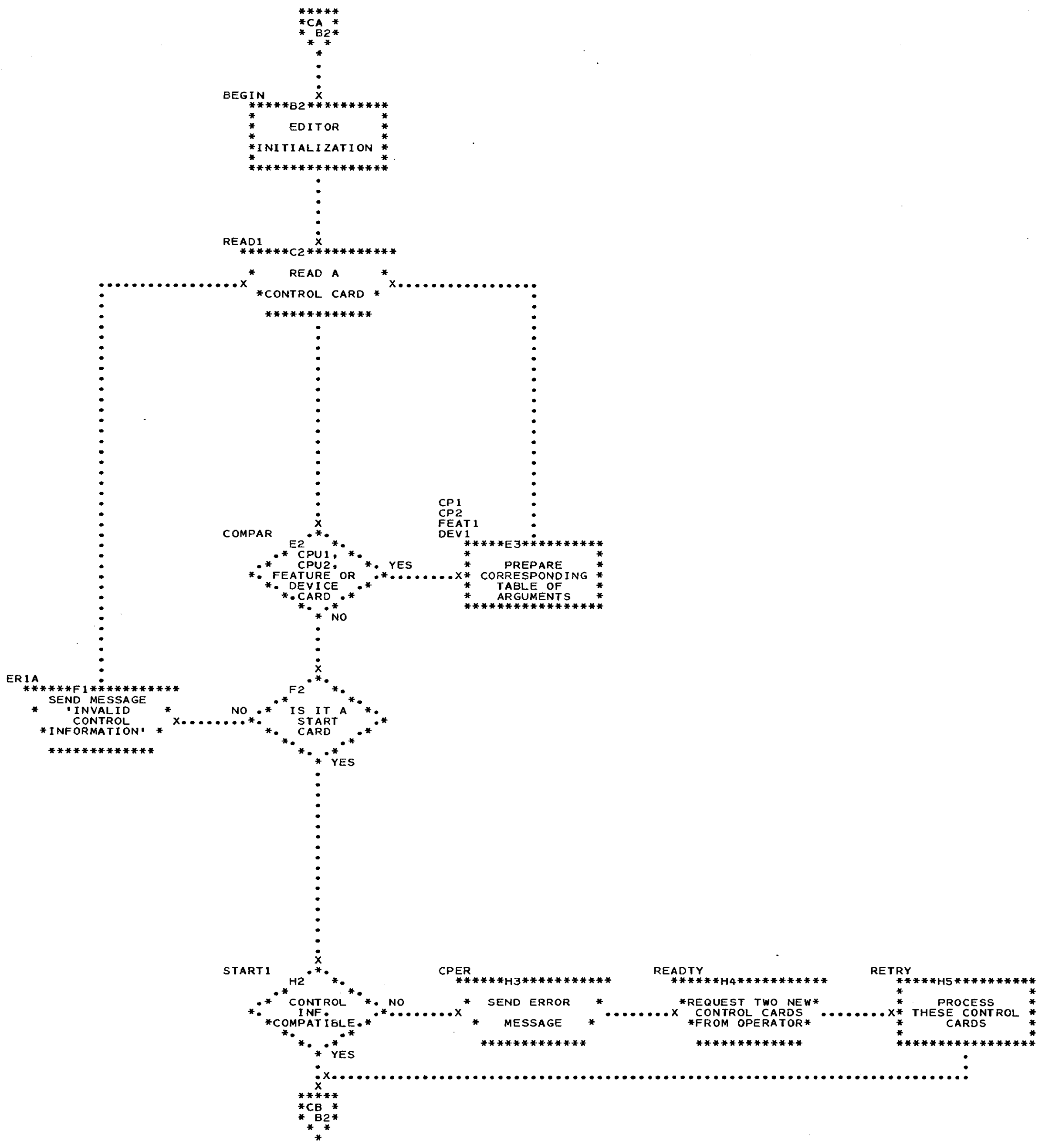
OVERALL LOGIC OF EDITOR

The following steps give the overall logic of EDITOR and follow the logic shown in Charts CA and CB.

1. Control cards are read, one at a time, and the control information is used to build up a table of arguments, later used to select routines from the symbolic SIM20 tape.
2. When the START card is encountered, processing begins; EDITOR checks the compatibility of:
   a. CPU1 and CPU2 cards, to ensure that the 1620 defined in the CPU1 card can be simulated on the System/360 defined in the CPU2 card
   b. FEATURE and CPU2 cards, to ensure that the optional or special features defined in the FEATURE card can be simulated on the System/360 defined in the CPU2 card
   c. DEVICE cards and the table of accepted devices contained in EDITOR

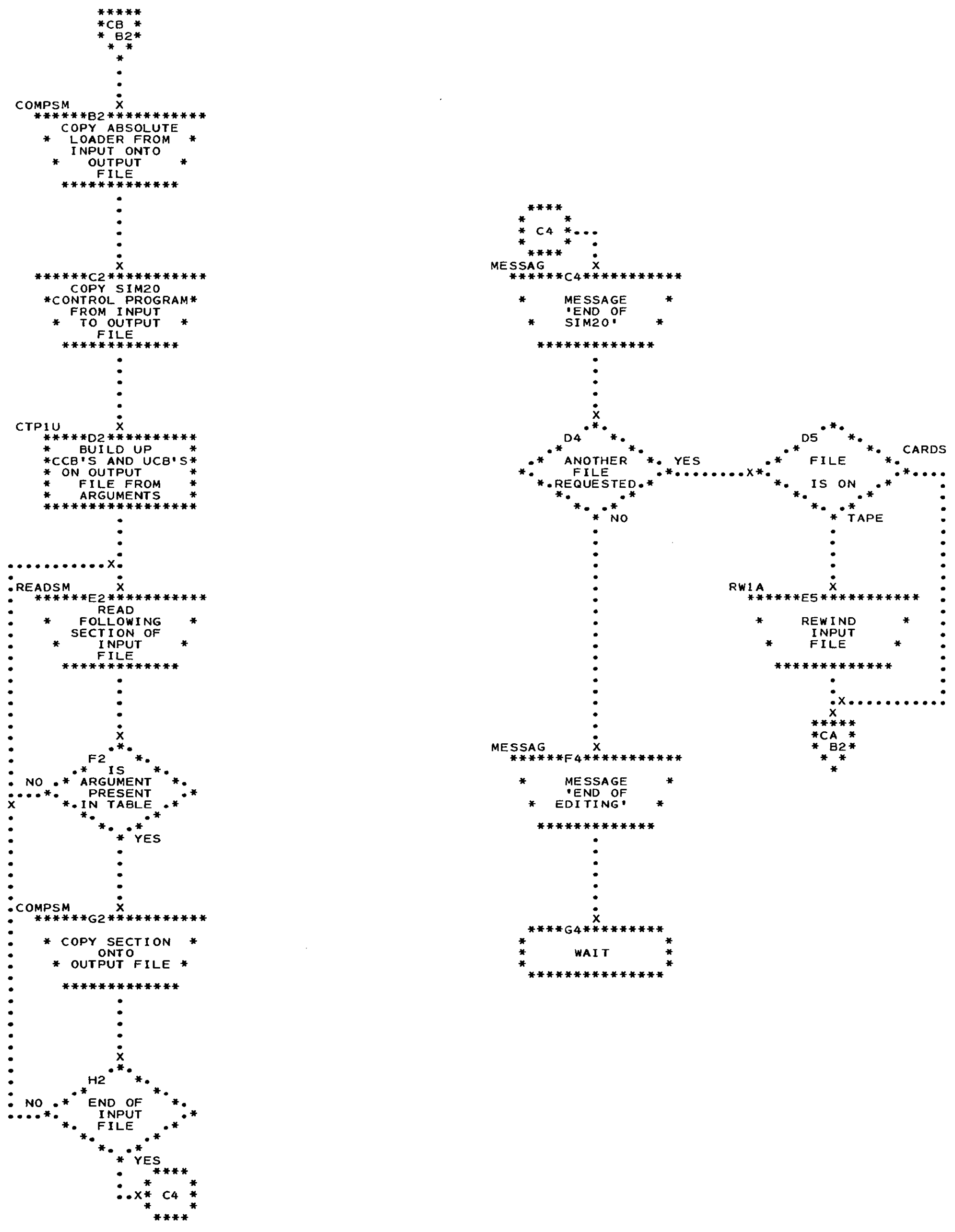
   When an incompatibility is found, a message is sent to the operator requesting him to enter two correct control statements or to resume the editing process.
3. The SIM20 version of CONTPR is written on the output device.
4. EDITOR then builds up the CCBs and UCBs from the information in the DEVICE control cards, selects routines from the SIM20 tape according to the table of arguments, and writes the selected routines on the output device.
5. When the end of the SIM20 tape is reached, a message is sent to the operator requesting him either to stop the editing procedure or to provide new control cards.
6. The SIM20 tape is then rewound, ready for the next editing run.

Chart CA. Logic of EDITOR (Part 1)



CA B2
BEGIN
B2 EDITOR INITIALIZATION
READ1
C2 READ A CONTROL CARD
COMPAR
E2 CPU1, CPU2, FEATURE OR DEVICE CARD
YES
NO
CP1
CP2
FEAT1
DEV1
E3 PREPARE CORRESPONDING TABLE OF ARGUMENTS
ER1A
F1 SEND MESSAGE 'INVALID CONTROL INFORMATION'
F2 IS IT A START CARD
NO
YES
START1
H2 CONTROL INF. COMPATIBLE
NO
YES
CPER
H3 SEND ERROR MESSAGE
READTY
H4 REQUEST TWO NEW CONTROL CARDS FROM OPERATOR
RETRY
H5 PROCESS THESE CONTROL CARDS
CB B2

Chart CB. Logic of EDITOR (Part 2)


CB B2
COMPSM B2 COPY ABSOLUTE LOADER FROM INPUT ONTO OUTPUT FILE
C2 COPY SIM20 CONTROL PROGRAM FROM INPUT TO OUTPUT FILE
CTP1U D2 BUILD UP CCB'S AND UCB'S ON OUTPUT FILE FROM ARGUMENTS
READSM E2 READ FOLLOWING SECTION OF INPUT FILE
F2 IS ARGUMENT PRESENT IN TABLE
NO
YES
COMPSM G2 COPY SECTION ONTO OUTPUT FILE
H2 END OF INPUT FILE
NO
YES
C4
MESSAG C4 MESSAGE 'END OF SIM20'
D4 ANOTHER FILE REQUESTED
YES
NO
D5 FILE IS ON
CARDS
TAPE
RW1A E5 REWIND INPUT FILE
CA B2
MESSAG F4 MESSAGE 'END OF EDITING'
G4 WAIT

DSKINT is automatically edited by EDITOR when a 1620/1311 system is requested in control information. The output file of EDITOR contains both SIM20 and DSKINT in symbolic format. After assembly, the binary deck obtained is self-loading and immediately gives control to DSKINT.

LOGIC OF DSKINT

DSKINT (see Chart DA):

1. Sets the device address of the System/360 devices used by SIM20.
2. Relocates the non-disk I/O simulation routines when the special disk-resident version of SIM20 has been requested.
3. Requests from the operator the address of the 2311 Disk Storage Drive which must be initialized (the disk addresses given by the operator are checked to ensure that they conform to the UCBs in SIM20).
4. Requests from the operator whether the disk formats are to be set up on the 2311 Disk Storage Drives. If they are, the program writes the home address, the record 0, and 21 records on the 2311. The first 20 records correspond to the 20 sectors of the 1311 and contain the System/360 identification followed by 105 data bytes (5 for the address, 100 for data); the 100 data bytes all contain X'F0'. Sector addresses are numbered successively from 00000 through 19999.

   The last, or 21st, record summarizes sequentially the preceding 20 addresses (for example, for track 0, cylinder 0, the 21st record contains sector addresses 00000, 00001, 00002, ...00019).
5. Requests from the operator whether SIM20 is to be loaded from System/360 main storage. If so, it is loaded onto cylinders 00, 01, and 02, in the following manner:

   a. The first track of cylinder 00 is an IPL track, and the following tracks contain parts of SIM20 (CPU and console simulation routines).
   b. Cylinder 01 is loaded with the I/O simulation routines (other than those for disks).
   c. Cylinder 02 is loaded with the disk simulation routines.
6. Signals the end of these two operations by messages END OF FORMAT and END OF LOADING, and then branches back to step 3.

Note: An exceptional condition not cleared by an error recovery procedure places the System/360 in the wait state. Operator commands such as END or NO in response to the message SIMULATOR LOADING NEEDED also place the System/360 in the wait state.

Loading SIM20 onto disk in a self-loading format is necessary when the user has defined a 1620/1311 system when editing SIM20. The DSKINT binary card deck, though it contains SIM20, cannot give control directly to SIM20. A SIM20 run can be started only by an IPL from the disk unit.

Formatting is necessary when the disk storage drive must be used to simulate a 1620/1311 system. Formatting is not necessary when the disk storage drive is used only as SIM20 residence.

BUFFERS AND TABLES

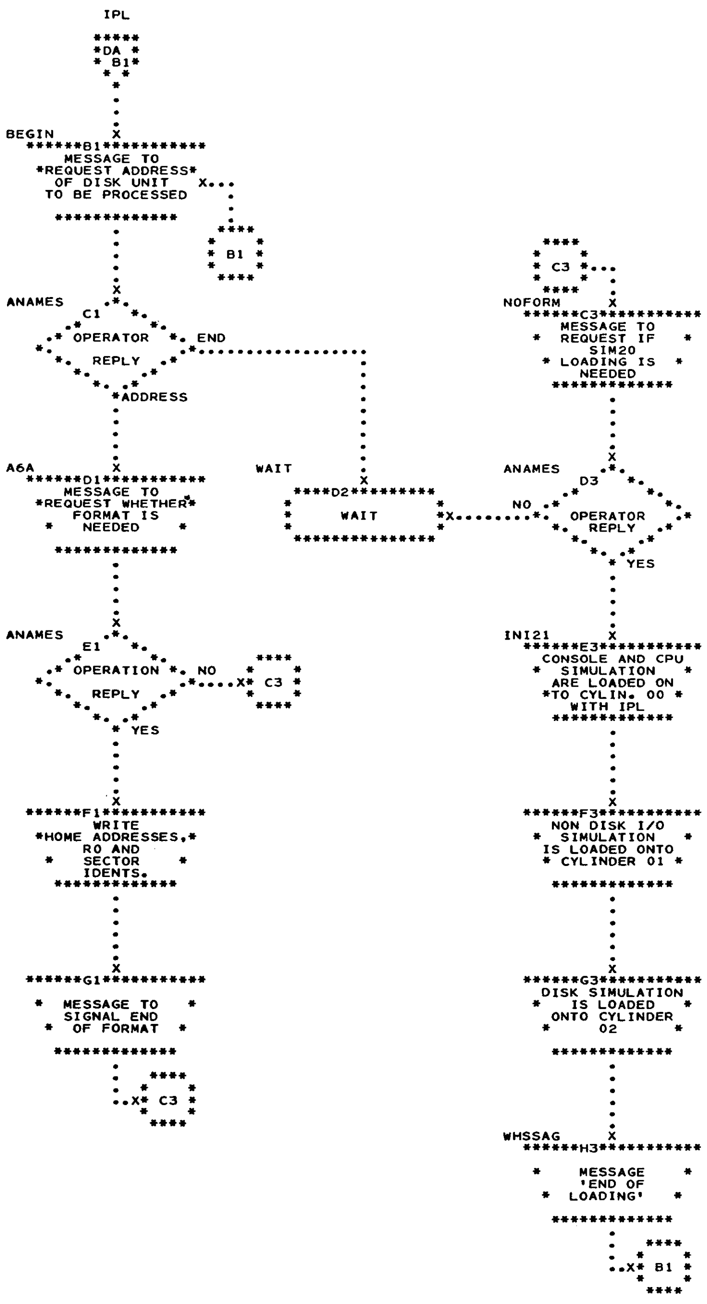
DSKINT contains three buffers:

1. An input/output buffer IOBUFF, used to write messages or to enter commands
2. A 3,400-byte buffer located at address X'6000', used to load the CPU and console simulation routines onto cylinder 00
3. A 2,100-byte buffer located at address X'6E00', used to load the disk simulation routines onto cylinder 02 (no buffer is necessary to load the other I/O simulation routines onto cylinder 01)

The program also contains two blocks of main storage:

1. A field for each sector, containing:
   The home address (4 bytes)
   The record 0 (1 byte)
   The key length (1 byte -- always = 0)
   The data length (2 bytes -- always = 105)
   The data field (105 bytes)
2. An IPL field containing the IPL sequence for 2311 Disk Storage Drives

Chart DA. DSKINT


IPL
DA B1
BEGIN B1 MESSAGE TO REQUEST ADDRESS OF DISK UNIT TO BE PROCESSED
B1
ANAMES C1 OPERATOR REPLY
END
ADDRESS
A6A D1 MESSAGE TO REQUEST WHETHER FORMAT IS NEEDED
ANAMES E1 OPERATION REPLY
NO
C3
YES
F1 WRITE HOME ADDRESSES, R0 AND SECTOR IDENTS.
G1 MESSAGE TO SIGNAL END OF FORMAT
C3
WAIT D2 WAIT
C3
NOFORM C3 MESSAGE TO REQUEST IF SIM20 LOADING IS NEEDED
ANAMES D3 OPERATOR REPLY
NO
YES
INI21 E3 CONSOLE AND CPU SIMULATION ARE LOADED ON TO CYLIN. 00 WITH IPL
F3 NON DISK I/O SIMULATION IS LOADED ONTO CYLINDER 01
G3 DISK SIMULATION IS LOADED ONTO CYLINDER 02
WHSSAG H3 MESSAGE 'END OF LOADING'
B1

UPDT20

UPDT20 (see Chart EA) reads from the device with symbolic name UPDTOLD a version of the Simulator system tape to be corrected, makes the corrections indicated in the information read from the device with symbolic name UPDTCORR, and writes the corrected version of the system tape on the device with symbolic name UPDTNEW.

To simplify the presentation, this description refers only to a system tape, but it is also valid when the Simulator is on punched cards.

LAYOUT OF THE SYSTEM TAPE

The system tape on UPDTOLD is made up of several files:

- A file contains one or several modules which, in turn, contain one or several records.
- Each file is followed by a tape mark.
- The end of the system tape is indicated by two tape marks.

Note: When the Simulator is distributed on cards, the card deck contains only one file composed of several modules.

IDENTIFICATION OF SYSTEM TAPE COMPONENTS

The three components of the system tape are identified as follows:

- A file is identified by the identification of its first module.
- A module is identified by the identification of its first record.

  A module is composed of all the consecutive records having the same identification in columns 73 through 75 for symbolic card-image records, or in columns 73 through 76 for binary card-image records.
- A record is identified by its serial number in columns 77 through 80 for binary records, or in columns 76 through 80 for symbolic records.

  Record numbers in each module must be in ascending order.

The identification of system tape components is illustrated in Table 5A.

Table 5A. Identification of Simulator System Tape

FILES	MODULES	RECORDS	COMPONENTS
A21B	A21B	A21B0001 A21Bnnnn	COMMON SUB-PROGRAMS composed of modules: A21B A22B A23B A24B A25B A26B
	A22B	A22B0001 A22Bnnnn	
	. . .	. . .	
	A26B	A26B0001	LDT card
TM			Tape mark
A2UB	A2UB	A2UB0001 A2UBnnnn	UPDT20
	A27B	A27B0001	LDT card
TM			Tape mark
A2EB	A2EB	A2EB0001 A2EBnnnn	EDITOR
TM			Tape mark
A2ZB	A2ZB	A2ZB0001	SYSINEND
TM			Tape mark
A2S	A2S	A2S00001 A2Snnnnn	SIM20
TM			Tape mark
A2P	AP2[1]	A2P00000 ....[1]	SAMPLE PROGRAM
TM TM			Tape marks Data end

[1]Identification field not significant.

UPDATING FUNCTIONS

Updating can be done at file level, at module level, or at record level. For corrections at module or record level, the file containing the module or record to be corrected must first be defined.

The main updating functions are:

Updating at File Level

- Copy an old file onto UPDTNEW
- Correct and copy an old file onto UPDTNEW

Updating at Module Level

- Copy an old module onto UPDTNEW
- Replace an old module by a new one

Updating at Record Level

- Delete an old record or a set of consecutive records
- Insert a new record or a set of consecutive records
- Replace an old record or a set of consecutive records by a new one
- Re-number a set of consecutive records

Note: All undefined records are copied without modification.

Other updating functions such as deleting, inserting, and listing all or only the corrected modules on UPDTNEW at file level, or deleting, inserting, re-numbering, and re-identifying a module at module level, or re-identifying a record or a set of consecutive records at record level, are explained in detail in the listing of the UPDT20 program supplied with the Program Logic Manual.

UPDATING THE SYSTEM TAPE

The correction data used to update the system tape may be on cards or on tape. The correction cards (or card images) required for UPDTCORR are divided into control cards and modification cards.

Since correction cards are not sorted by the program, the sequence of the files, modules, and records required for correction must exactly correspond to the sequence of the files, modules, and records of the tape distributed by IBM. Any corrections the user may receive will be in the correct order.

Control Cards

Three types of control card are used for the main updating functions.

- The / UPDATE card defines the old file to be corrected or copied onto UPDTNEW. If this control card has not been specified for an old file, this file is ignored by UPDT20.
- The RIS mode C card defines each module to be replaced in the file defined by the / UPDATE card.
- The RIS mode R card defines the record, or set of consecutive records, to be corrected in the file defined by the / UPDATE card.

Modification Cards

The modification cards contain the replacement data for the new modules or the insertion or replacement data for the new records to be written on UPDTNEW.

The identification in columns 73 through 75, or in columns 73 through 76 is that of the module defined in the corresponding RIS card.

Control Card Formats

The format of the / UPDATE card is illustrated in Figure 9. The format of the RIS mode C card is given in Figure 10, and that of the RIS mode R card in Figure 11.

Column 1 3 15 18 80	
/ UPDATE	Defines card as / UPDATE control card
XXX	Identification of first symbolic module of the file
XXXB	Identification of first binary module of the file

Figure 9. Format of the / UPDATE Card

Column 12	8	41	44	80	
*RIS			C		Defines card as RIS mode C card C = mode C
		R			R = replace module
	XXX XXXB				Identification of module to be replaced (same for old and new module)
* = 12-2-9 punch Note: This card is required only for modules to be replaced.					

Figure 10. Format of the RIS Mode C Card

Column 12	8	24	41	59	67	80	
*RIS							Defines card as RIS mode R card Column 44 must be blank.
	XXXnnnnn XXXBnnnn						Identification of first old record to be replaced, deleted, or renumbered, or the record after which the first new record is to be inserted
		XXXnnnnn XXXBnnnn					Identification of last old record to be replaced, deleted, or renumbered
			R I S N				Replace Insert Suppress (delete) Re-number
			nnnnn[1]				Identification number required for first new record
					nn[1]		Increment value of identification
* = 12-2-9 punch [1]Leading zeros can be ignored.							

Figure 11. Format of the RIS Mode R Card

CODING EXAMPLES

The following examples show how the control cards should be punched to perform the updating functions described in the section "Updating the System Tape."

```
 1.   / UPDATE      A21B

 2.  *RIS   A2EB              R  C

 3.  *RIS   A2S01230        A2S01240         S

 4.  *RIS   A2S01375                         I            01376     001

 5.  *RIS   A2S02300        A2S02380         R            02300     005

 6.  *RIS   A2S08000        A2S09810         N            10010     005

 7.  / UPDATE      A21B
     / UPDATE      A2UB
     / UPDATE      A2EB
     *RIS   A2EB              R  C
     ┌
     │ new module (replacement cards)
     └
     / UPDATE      A2ZB
     / UPDATE      A2S
     *RIS   A2S01230        A2S01370         S
     *RIS   A2S03750                         I             3751     001
     ┌
     │ insertion cards
     └
     *RIS   A2S02300        A2S02380         R            02300     005
     ┌
     │ replacement cards
     └
     / UPDATE      A2P
```

*represents a multiple punch of 12-2-9 in column 1.

Updating a File

In example 1 above, file A21B is to be copied onto UPDTNEW with or without a modification at module or record level.

Updating a Module

In example 2, the whole binary module identified by A2EB is to be replaced.

Updating a Record

In example 3, records A2S01230 through A2S01240 of the file defined by the / UPDATE card are to be deleted.

In example 4, new records are to be inserted after record A2S01375 of the file defined by the / UPDATE card. The identification of the first new record is to be A2S01376; the numbering step is 1.

In example 5, records A2S02300 through A2S02380 are to be replaced by a record or set of records. The identification of the first replacement record is 02300; the numbering step is 5.

In example 6, records A2S08000 through A2S09810 are to be re-numbered beginning with number 10010; the numbering step is 5.

Updating the System

Example 7 shows a sequence of correction cards used in updating the system tape. The first part covers updating the system tape and replacing module A2EB. The second part covers updating the system tape and deleting, inserting, replacing and renumbering records of file A2S.

INITUP ROUTINE

The INITUP routine initializes UPDT20 by:

- Reading the first record of the system tape on UPDTOLD and the first / UPDATE card
- Transferring control to the File Processing routine (FLTGA)

FILE PROCESSING ROUTINE (FLTGA)

This routine selects the operations to be performed on a file.

Entry Points: The FLTGA routine has two entry points, first from the INIT routine, and then from the CSTGL routine after processing a file.

Input: When the routine is entered, the first record of the system tape to be corrected (or the last tape mark on the tape) and a / UPDATE card (or the last of the correction cards) have been read.

Operation: This routine analyzes the contents of the / UPDATE card and of the record, and transfers control to the routine to perform the requested operation.

Exits: Control is transferred to one of the routines NLSTP, SKLDM, SKCRDM, SKLDN, or CSTGB. When the routines SKCRDM, SKLDN, and CSTGB are entered, the last record read has been specified in the / UPDATE card.

NLSTP Routine

This routine is entered at the end of the run. It writes the last tape mark on the system tape on UPDTNEW.

SKLDM Routine

If the record just read has not been specified in the / UPDATE card, this routine suppresses a file on the system tape on UPDTOLD. All the records on this system tape, up to the next tape mark, are suppressed; that is, they are not written on the system tape on UPDTNEW. Control is then transferred to the CSTGC routine (end of processing of a file).

SKCRDM Routine

If the / UPDATE card is followed by a modification card, this routine skips over the correction cards for a file. All the modification and RIS cards following this / UPDATE card are read, but not processed. Control is then transferred to the SKLDN routine.

SKLDN Routine

If the / UPDATE card is followed by another / UPDATE card or by no card at all, this routine copies a file from the system tape on UPDTOLD onto the tape on UPDTNEW. The record just read, and all those that follow, up to the next tape mark, are copied onto the tape on UPDTNEW. Control is then transferred to the CSTGC routine (end of processing of a file).

MODULE PROCESSING ROUTINES (CSTGA AND CSTGB)

These routines select the operations to be performed on a module (control section), provided that the / UPDATE card is followed by an RIS mode C card.

Entry Points: The CSTGB routine is entered from the FLTGA routine, and the CSTGA routine is entered from the routines SKCRDA, SKLDA, RISN (after an entire module has been processed), and RCTGB (after all the records in a module have been corrected).

Input: When the routine is entered, the first record of a module from the system tape on UPDTOLD (or a tape mark after all the modules in a file have been processed) and an RIS card (or a / UPDATE card or the last of the correction cards) have been read.

Operation: The routine analyzes the parameters in the RIS card and the contents of the first record of the module. Control is then transferred to the appropriate subroutine to process the module.

Exit: When the module has been processed (when the program has encountered a tape mark on the system tape on UPDTOLD, a / UPDATE card, or the last of the correction cards on UPDTCORR), control is transferred to the CSTGC routine.

SKCRDA Subroutine

If the RIS card is invalid, this routine skips over the correction cards for the module. This card and all the MODIF and RIS cards for this module are read, but not processed. Control is then returned to CSTGA to process the next module in the file.

SKLDA Subroutine

If the RIS card has not specified the record just read, this routine copies a module from the system tape on UPDTOLD onto the tape on UPDTNEW. Control is then returned to CSTGA to process the next module in the file.

RISN Subroutine

If the RIS card has specified the record just read, and if this RIS card is a mode C card, this routine replaces, inserts, suppresses, or re-numbers an entire module. Control is then returned to CSTGA to process the next module in the file.

END OF FILE PROCESSING ROUTINE (CSTGC)

This routine is entered from the routines CSTGA, CSTGB, SKLDM, and SKLDN. It writes the end-of-file tape mark on the tape on UPDTNEW and transfers control to the FLTGA routine to process the next file.

RECORD PROCESSING ROUTINE (RCTGB)

This routine selects the operations to be performed on the records of a given module, provided that the RIS card has specified the record just read, and that the RIS card is a mode R card.

Entry Points: This routine is entered from the CSTGA routine, and from the routines SKCRDA, SKLDB, and RISN.

Input: When the routine is entered, a record of the module being processed (or the first record of the next module, or a tape mark after the last record of a module has been processed) and an RIS card referring to this module (or an RIS card referring to another module, or a / UPDATE card) have been read.

Operation: This routine analyzes the parameters of the RIS card and the contents of the record just read. Control is then transferred to the appropriate subroutine to process the record.

Exit: When all the records in a module have been processed, control is transferred to the CSTGA routine to process the next module.

SKCRDA Routine

If the RIS card is invalid, this routine skips over the correction cards for a module. This RIS card and all the correction cards for the module are read, but not processed. Control is then returned to RCTGB to process the next record.

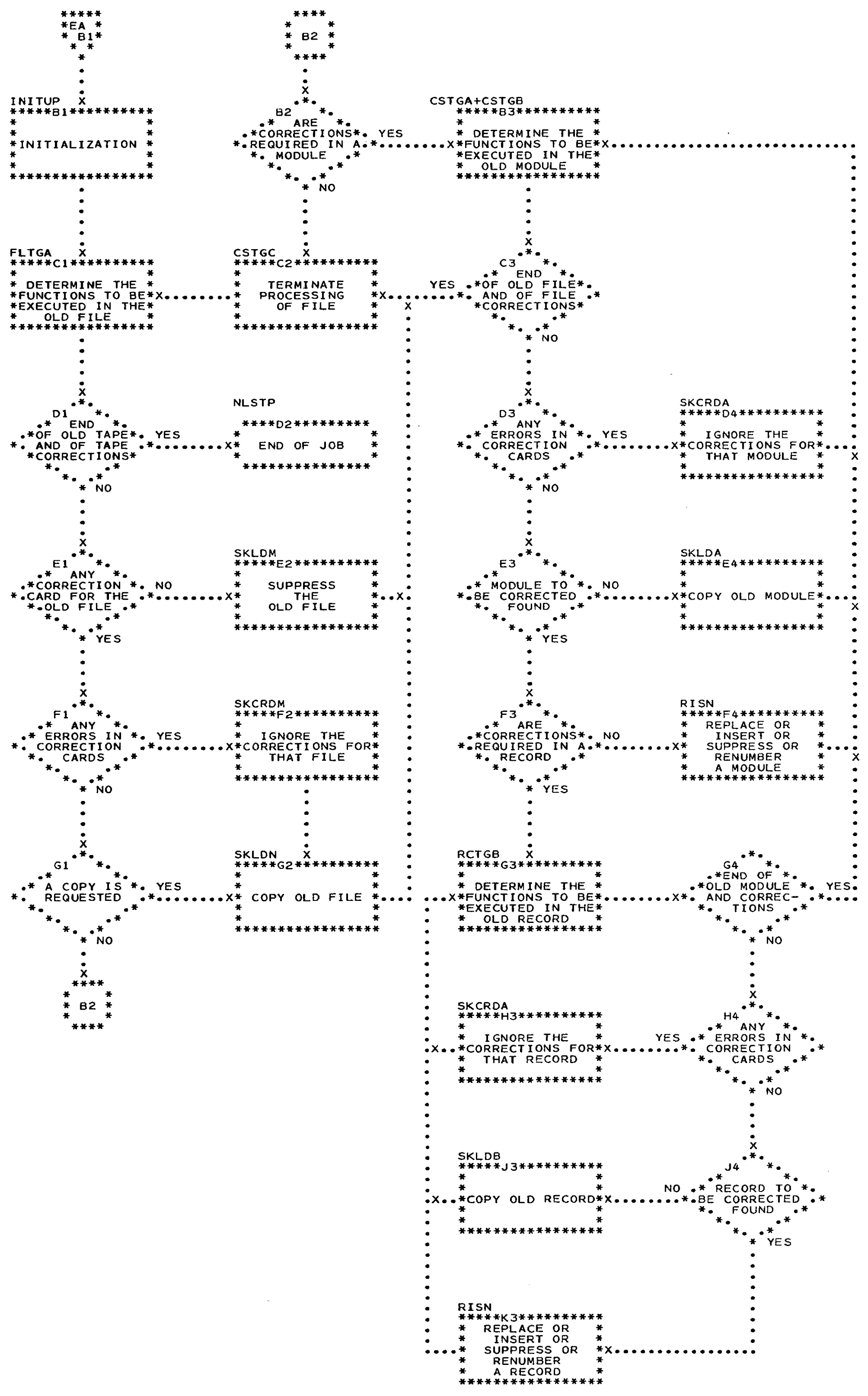
SKLDB Routine

If the RIS card has not specified this record, this routine copies a record from the system tape on UPDTOLD. Control is then returned to RCTGB to process the next record.

RISN Routine

If the RIS card has specified the record or records and if the RIS card is a mode R card, this routine replaces, inserts, suppresses, or re-numbers a set of records. Control is then returned to RCTGB to process the next record.

Chart EA. Overall Logic of UPDT20


EA B1
B2
INITUP B1 INITIALIZATION
B2 ARE CORRECTIONS REQUIRED IN A MODULE
YES
NO
CSTGA+CSTGB B3 DETERMINE THE FUNCTIONS TO BE EXECUTED IN THE OLD MODULE
FLTGA C1 DETERMINE THE FUNCTIONS TO BE EXECUTED IN THE OLD FILE
CSTGC C2 TERMINATE PROCESSING OF FILE
C3 END OF OLD FILE AND OF FILE CORRECTIONS
YES
NO
D1 END OF OLD TAPE AND OF TAPE CORRECTIONS
YES
NO
NLSTP D2 END OF JOB
D3 ANY ERRORS IN THE CORRECTION CARDS
YES
NO
SKCRDA D4 IGNORE THE CORRECTIONS FOR THAT MODULE
E1 ANY CORRECTION CARD FOR THE OLD FILE
NO
YES
SKLDM E2 SUPPRESS THE OLD FILE
E3 MODULE TO BE CORRECTED FOUND
NO
YES
SKLDA E4 COPY OLD MODULE
F1 ANY ERRORS IN CORRECTION CARDS
YES
NO
SKCRDM F2 IGNORE THE CORRECTIONS FOR THAT FILE
F3 ARE CORRECTIONS REQUIRED IN A RECORD
NO
YES
RISN F4 REPLACE OR INSERT OR SUPPRESS OR RENUMBER A MODULE
G1 A COPY IS REQUESTED
YES
NO
SKLDN G2 COPY OLD FILE
RCTGB G3 DETERMINE THE FUNCTIONS TO BE EXECUTED IN THE OLD RECORD
G4 END OF OLD MODULE AND CORRECTIONS
YES
NO
B2
SKCRDA H3 IGNORE THE CORRECTIONS FOR THAT RECORD
H4 ANY ERRORS IN CORRECTION CARDS
YES
NO
SKLDB J3 COPY OLD RECORD
J4 RECORD TO BE CORRECTED FOUND
NO
YES
RISN K3 REPLACE OR INSERT OR SUPPRESS OR RENUMBER A RECORD

ABSOLUTE LOADER (ABSLOD)

ABSLOD is used to load the following subprograms into the System/360 storage locations assigned by the assembler (see Chart FA):

- CONTPR
- IOPACK
- INIT
- RELLDR

ABSLOD also accepts assembled programs intended to be loaded by a relocating loader, but with the limitations given in the section "Additional ABSLOD Functions."

ABSLOD is used to load the assembled SIM20 and DSKINT programs.

ABSLOD CARDS AND FUNCTIONS

Five types of load card are recognized by this loader: TXT and END cards which were generated by the assembler, any REP cards which may be inserted by the user, and the LDR and LDT cards which were supplied by the IBM programmer. The load cards of the subprograms listed above have been converted, in the correct order, to card images on magnetic tape and form the first portion of the Simulator system tape.

The functions of ABSLOD and the cards associated with each function are listed in Table 6.

Card Sequence

Each subprogram, or control section, in the Simulator system tape includes at least two types of card: TXT and END, in that order. The LDR card is placed between the END card of IOPACK and the first card of INIT. The last card in the deck is an LDT card.

If the user wants to make any changes in the assembled program, he must insert the appropriate REP cards after the last TXT card of the control section concerned and before the END card.

As each TXT or REP card is read, ABSLOD places the contents of the card in storage at the absolute address given in that card; therefore, the addresses in the cards need not be in increasing order of value. It is also possible to overlay a section which has already been loaded.

The value of the highest address loaded is recorded by the loader in a location counter (LOCCTR). Each time the location counter is incremented, its value is checked to make sure that the loader program is not overlaid. If it is, loading is interrupted and the System/360 enters the wait state.

Card Formats

Values in load cards produced by the assembler are represented in IBM extended card code; for example, the decimal value 20 (represented in one byte as 0001 0100) becomes an 11-9-4 punch in one card column.

In contrast, the programmer uses the more convenient hexadecimal code if REP cards are used. The hexadecimal equivalent of decimal 20 is 14; this is a 1 punch and a 4 punch in two successive card columns, representing the contents of one byte.

Table 6. ABSLOD Functions

FUNCTIONS	CARDS
<u>Loading</u>: Places the instructions or constants, or both, of a control section into the storage locations assigned by the assembler.	Text (TXT)
<u>Correcting</u>: Allows changes to be made to the instructions or constants in the program at load time.	Replace (REP)
<u>Transferring Control</u>: Ends loading of the control section and transfers control to some location within the section.	Load End (END) Load Terminate (LDT)

Text Card

The Text (TXT) card is generated by the assembler and contains, in IBM extended card code, the following:

1. The address at which the assembled instructions and constants in the card are to be inserted

2. The number of bytes of information contained in the card

3. The text itself, up to a maximum of 56 bytes

The contents of the TXT card fields are defined in Table 7.

Table 7. Text Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	TXT. Identifies the type of load card.
5	Blank.
6-8	The address, in extended card code, at which the information on the card is to be loaded.
9-10	Blank.
11-12	Number, in extended card code, of bytes of text in the card.
13-14	Blank.
15-16	Information for RELLDR. The contents of these columns is ignored by ABSLOD.
17-72	From 1 to 56 bytes of text (instructions or constants assembled in extended card code).
73-80	Not used by the loader.

Replace Card

Replace (REP) cards are supplied by the programmer and must be placed in the control section immediately after the last TXT card. Assembled instructions or constants, or both, are replaced byte for byte by the instructions or constants punched in the card in hexadecimal code. A REP card may contain a minimum of two bytes (one half-word) and a maximum of 22 bytes.

The programmer cannot replace a two-byte instruction by a four-byte instruction through the load program. Instead, he must either re-assemble his source program or patch; that is, replace the incorrect or old entry with a branch instruction to some storage location into which the replacement will be loaded. Replacement must be made byte for byte.

The contents of the REP card fields are defined in Table 8.

Table 8. Replace Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	REP. Identifies the type of load card.
5-6	Blank.
7-12	Address, in hexadecimal, of the area to be replaced. It must be right-justified in these columns. Unused leading columns are filled with zeros. The address must specify a half-word boundary.
13-16	Blank.
17-70	A maximum of eleven 4-digit hexadecimal fields, separated by commas, each replacing one previously loaded half-word (two bytes). The last field must not be followed by a comma.
71-72	Blank.
73-80	Not used by the loader.

Load End Card

The Load End (END) card is generated by the assembler when it encounters the END instruction. This card ends the loading of a control section and may specify a location within the section to which control is to be transferred.

The contents of the END card fields are defined in Table 9.

Table 9. Load End Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	END. Identifies the type of load card.
5	Blank.
6-8	Address (may be blank), in extended card code, of the point in the control section to which control may be transferred at the end of the loading process. See the priority conditions discussed under the LDT card.
9-14	Blank.
15-16	Information for RELLDR. The contents of these columns is ignored by ABSLOD.
17-72	Blank.
73-80	Not used by the loader.

Processing of END Cards by ABSLOD

When an END card is found by ABSLOD, parameters pertinent to this card are stored in System/360 main storage, then transferred to registers or return addresses when an LDT card is found.

Four types of END Cards may be found by ABSLOD:

1. CONTPR END card

   The address of the last byte of CONTPR, which is contained in the location counter, is stored in System/360 main storage. When an LDT card is found, this address is stored in System/360 general register 2.

2. IOPACK END card

   The address of the byte following IOPACK, which is contained in LOCCTR+1, is stored in System/360 main storage. When an LDT card is found, this address is stored in System/360 general register 1.

3. INIT END card

   The entry point in INIT, specified in this card, is stored in a PSW. It is used as a return address to INIT at the end of ABSLOD.

4. RELLDR END card

   The entry point in RELLDR, specified in this card, is stored in System/360 main storage. It is used as a return address to RELLDR at the end of INIT.

LDR Card

Of the four other subprograms included in the Simulator system tape, two (CONTPR and IOPACK) are designed to reside permanently in storage with whatever other program is being used: EDITOR or UPDT20, and are therefore loaded starting at address 0. The two other subprograms (INIT and RELLDR) are used only to load and initialize the Simulator programs; thereafter they are overlaid. These two subprograms are therefore loaded into storage after address 56000.

The LDR card, which is placed immediately after IOPACK, terminates incrementing of the location counter (LOCCTR) at this point, so that the Simulator programs may be loaded from this address onwards. The location counter is therefore not incremented when INIT and RELLDR are loaded.

The contents of the LDR card fields are defined in Table 10.

Table 10. LDR Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	LDR. Identifies the type of load card.
5-72	Blank.
73-80	Not used by the loader.

Load Terminate Card

The Load Terminate (LDT) card is placed at the end of the input deck. It has two uses:

1. It ends the loading process.
2. It causes control to be transferred to some location within the section or sections loaded.

The location to which control is transferred is determined according to the following order of priority:

1. Control is always transferred to any location specified in the LDT card.
2. If the LDT card does not specify a location, control is transferred to the first location specified by an END card encountered during the current loading process.
3. If neither the LDT card nor any END cards specify a location, control is transferred to location 0, resulting in an error halt.

The contents of the LDT card fields are defined in Table 11.

Table 11. Load Terminate Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	LDT. Identifies the type of load card.
5	Blank.
6-8	Address (may be blank), in extended card code, of the point in the program to which control is to be transferred.
9-72	Blank.
73-80	Not used by the loader.

The LDT card at the end of RELLDR is blank and control is transferred to the address specified in the END card at the end of INIT.

ABSLOD prepares the following data in the general registers for INIT:

- The address of the last byte of CONTPR
- The current value of the location counter
- The address in RELLDR to which control must be transferred when initialization is completed

ABSLOD initializes the machine-check and program-check new PSWs to cause the System/360 to enter the wait state if a machine check or a program check occurs. Before loading, ABSLOD also sets core storage to zero from location X'180' (384 decimal) to the end of storage, except for the area occupied by ABSLOD.

After control is transferred to the loaded program, the Simulator operates in the problem state, disabled for all interruptions except machine check or program check until the PSWs for these interruptions are altered by the programs loaded.

ADDITIONAL ABSLOD FUNCTIONS

ABSLOD also accepts assembled programs intended to be loaded by a relocating loader, but with the following limitations:

1. Cards of other types than those listed above (SLC, ICS, ESD, and RLD) are ignored, as is any information on TXT, REP, and END cards meaningful only to RELLDR.
2. Control sections are not linked. Should one control section refer to instructions or data in a section assembled separately, absolute addresses are used.

CONTROL PROGRAM (CONTPR)

CONTPR (see Chart FB) is used by EDITOR, UPDT20, and, in a reduced form, by SIM20 and DSKINT. To simplify the presentation, this description refers only to EDITOR, but is valid for all the programs. When a part of CONTPR is not used by SIM20 and DSKINT, it is so stated.

CONTPR consists of routines to:

- Process machine-check interruptions
- Process supervisor-call interruptions
- Process program interruptions
- Process I/O interruptions
- Verify the characteristics of I/O devices
- Process I/O requests
- Set up the standard SEREP interface
- Communicate with the 1052 Printer-Keyboard

CONTPR operates in the supervisor state, whereas EDITOR operates in the problem state. Any attempt to execute a privileged instruction within EDITOR causes a program interruption.

Machine-Check Interruptions

When a machine-check interruption occurs, CONTPR is entered. It responds by setting up the standard SEREP interface for a machine check.

Supervisor-Call Interruptions

Supervisor-call (SVC) interruptions are processed by means of an SVC table of 20 full-word entries corresponding, in order, to the allowed values (0 through 19) of the interruption code in an SVC instruction. These codes and their corresponding functions are given in Table 12.

If the interruption code is greater than 19, a program interruption is artificially created; the interruption code portion of the program old PSW is set to indicate an operation exception. Otherwise, control is passed to the appropriate routine via the SVC table.

For most of its functions, CONTPR must be given a number of parameters, whose values are set up in the bytes immediately following the SVC instruction. An SVC instruction, together with its necessary parameters, is referred to as an SVC calling sequence.

The general and floating-point registers may contain any value when an SVC calling sequence is presented to CONTPR. When control is returned to EDITOR, the contents of these registers remain unchanged.

Only the SVC instructions with interruption codes 1, 2, 3, 7, 8, and 9 are used by SIM20 and DSKINT.

Program Interruptions (SVC 6)

When a program interruption occurs, CONTPR is entered.

When CONTPR is loaded into System/360 main storage, program interruptions are processed in the following way:

1. A PSW is loaded, for which I/O and external interruptions are enabled, and in which the wait state bit is one and all the interruption code bits are zeros.

2. The wait light on the system control panel is turned on, and, except when processing I/O and external interruptions, operator intervention is awaited.

Table 12. SVC Table

SVC CODE	FUNCTION
0	I/O device verification[1]
1	Submit an I/O request and interrupt at channel end[2]
2	Submit an I/O request and continue[2]
3	Return to the point of interruption
4	Write a message[1]
5	Set command parameters[1]
6	Set return address for a program interruption[1]
7	Set up SEREP interface
8	Disable I/O and external interruptions
9	Enable I/O and external interruptions
10	Set return address for an external interruption[1]
11	Submit an I/O request and wait[1]
12	Dump System/360 main storage[1-2]
13	Rewind a specified 2400-Series Magnetic Tape Unit[1]
14	Rewind and unload a specified 2400-Series Magnetic Tape Unit[1]
15	Disable console (ignore attention interruptions)[1]
16	Enable console (accept attention interruptions)[1]
17	Set parameters for a logical I/O request to IOPACK[1]
18	Submit a logical I/O request to IOPACK[1]
19	Set the wait state bit "on" in the current PSW[1]

[1]These SVC codes are not used by SIM20 and DSKINT.
[2]These SVC codes are not used by EDITOR and UPDT20.

The processing of subsequent program interruptions may be changed by submitting an SVC 6 calling sequence to CONTPR. This calling sequence has the following form:

```
         CNOP     2,8
I        SVC      6
         DC       A(PRRET)
PRPSW    DS       D
I+14     Any instruction
```

As a result of this calling sequence, when any subsequent program interruption occurs, the program old PSW is placed in the double-word with address PRPSW, I/O and external interruptions are disabled, and control is returned to EDITOR at address PRRET.

This SVC calling sequence is not used by SIM20 and DSKINT.

External Interruptions (SVC 10)

When an external interruption occurs, CONTPR is entered.

In the following cases, external interruptions are ignored:

- When CONTPR is loaded into System/360 main storage
- When an external interruption occurs because of an external signal

In these cases, the external old PSW is loaded into the PSW.

External interruptions related to the timer or to the interrupt key on the system control panel can be processed by submitting an SVC 10 calling sequence to CONTPR. This calling sequence has the following form:

```
         CNOP     6,8
I        SVC      10
I        DC       A(TIMINT)
I        DC       A(KEYINT)
EXTPSW   DS       D
I+18     Any instruction
```

As a result of this calling sequence, when any subsequent timer or interrupt-key interruption occurs, the external old PSW is placed in the double-word with address EXTPSW, I/O and external interruptions are disabled, and control is returned either to the instruction with address TIMINT (timer interruption) or to that with address KEYINT (interrupt-key interruption).

If the value of TIMINT or KEYINT is zero, the interruption is ignored.

This SVC calling sequence is not used by SIM20 and DSKINT.

Disable I/O and External Interruptions (SVC 8)

The SVC 8 calling sequence causes I/O and external interruptions to be disabled; that is, the system mask is set to the value X'00'.

The disabled state may be set up either by the SVC 8 calling sequence or as a result of an interruption.

Enable I/O and External Interruptions (SVC 9)

The SVC 9 calling sequence causes I/O and external interruptions to be enabled; that is, the system mask is set to the value X'FF'.

I/O DEVICE VERIFICATION (SVC 0)

CONTPR verifies that the device at a given System/360 address is of the type "tttt" and has special features corresponding to "ss". The value of "tttt" and the bit structure of the special-features byte "ss" for the devices supported by CONTPR are presented in Table 13.

The SVC 0 calling sequence has the following form:

```
         CNOP     0,4
I        SVC      0
DEV360   DC       X'0ddd'
TYPE     DC       C'tttt'
FEATURE  DC       X'ss'
         DC       AL3(ERROR)
I+12     Any instruction
```

If the device at System/360 address X'0ddd' corresponds to the device specified in the SVC 0 calling sequence, control is returned to address I+12; if not, control is returned to address ERROR.

This SVC calling sequence is not used by SIM20 and DSKINT.

I/O REQUESTS

Three types of I/O request can be submitted to CONTPR. These are:

- I/O Request and Wait: CONTPR returns control to EDITOR only when all activity related to the I/O operation has terminated. (This type of I/O request is not used by SIM20 and DSKINT.)

Table 13. Device Verification Table

DEVICE TYPE	SPECIAL-FEATURES BYTE
1442 Card Read Punch tttt = 1442	Bit 7 = 0 No Card Image feature Bit 7 = 1 Card Image feature
2501 Card Reader tttt = 2501	Bit 7 = 0 No Data Mode 2 Bit 7 = 1 Data Mode 2
2520 Card Read Punch tttt = 2520	Bit 7 = 0 No Data Mode 2 Bit 7 = 1 Data Mode 2
2540 Card Read Punch tttt = 2540	Bit 7 = 0 No Column Binary feature Bit 7 = 1 Column Binary feature
1403 Printer tttt = 1403	Bit 7 = 0 100 print positions Bit 7 = 1 132 print positions
1443 Printer tttt = 1443	Bit 7 = 0 120 print positions Bit 7 = 1 144 print positions
2400-Series Magnetic Tape Unit tttt = 2400	Bit 7 = 0 Nine-track tapes Bit 7 = 1 Seven-track tapes Bit 6 = 0 No data converter Bit 6 = 1 Data converter Bit 5 = 0 Seven-track tapes Bit 5 = 1 Nine-track tapes
1052 Printer-Keyboard	No special features
2671 Paper Tape Reader	No special features
2311 Disk Storage Drive	No special features

- I/O Request and Continue: CONTPR returns control to SIM20 as soon as possible after having accepted the request. I/O interruptions related to such a request interrupt SIM20 and transfer control to CONTPR. CONTPR preserves all information related to the I/O interruption and, if this information indicates that all the I/O activity related to the request has terminated, returns control to SIM20 at a predetermined location. Otherwise, control is returned to SIM20 at the point of interruption.

- I/O Request and Interrupt at Channel End: This request is similar to the I/O request and continue, except that, in the absence of unusual conditions, the channel-end condition also causes CONTPR to return control to SIM20 at a predetermined location.

SVC Calling Sequence Parameters

The SVC calling sequences for I/O requests contain the following parameters:

- DEV360 gives the System/360 address of the device for which the request is intended.

- CAWADD gives the address of the first CCW to be executed. (There is no restriction on the CCWs that can be presented. In particular, a string of CCWs connected by either data chaining or command chaining is permitted.)

- STATUS is treated by CONTPR as two hexadecimal digits, STRTBT and ERRTYP. On receipt of the I/O request, both STRTBT and ERRTYP are set to zero.

  STRTBT is set to one only when the physical I/O operation has been initiated at the device. When control is returned to SIM20 at address ACCRET (request accepted), STRTBT indicates whether or not the physical I/O operation has been initiated. Also, at initial selection, if an error condition precludes the initiation of the operation, STRTBT is set to one.

ERRTYP indicates whether or not an exceptional condition has occurred and, if so, the type of condition.

- SNSADD denotes the address of the first of three bytes used to accumulate the first three bytes of sense information during a sense operation performed as a result of a unit-check condition detected during the execution of an I/O request. On receipt of an I/O request, these bytes are set to zero.
- SVCCSW denotes the address of a double-word used to accumulate channel status information. On receipt of an I/O request, the contents of this double-word are set to zero. If channel and device status information is generated on more than one occasion during the execution of a chain of I/O commands, CONTPR accumulates the logical "OR" of this status information in the appropriate bytes of SVCCSW.
- SVCPSW denotes the address of a double-word in which is placed the I/O old PSW generated by the last I/O interruption related to the request. (This parameter is not present in the I/O request and wait calling sequence.)

I/O Request and Continue (SVC 2)

The SVC calling sequence used to submit an I/O request and continue has the following form:

```
        CNOP    4,8
I       SVC     2
DEV360  DC      X'0ddd'
CAWADD  DC      A(CCWADD)
STATUS  DS      C
SNSADD  DS      3C
SVCCSW  DS      D
SVCPSW  DS      D
        DC      A(NRMRET)
        DC      A(EXCRET)
ACCRET  Any instruction
        (Address I+36)
```

If the associated channel, subchannel, control unit, or device is busy, precluding initiation of the I/O request, CONTPR places this request in a queue until all parts of the device path are free.

Control is returned to SIM20 under any of the following conditions.

- Physical I/O operation started:

  STRTBT=1
  Return to address ACCRET

- Device path (channel, subchannel, control unit, or device) busy:

  Add I/O request to request queue
  STRTBT=0
  Return to address ACCRET

- Physical I/O operation started and terminated with no exceptional conditions:

  STRTBT=1
  Place OSVPSW at SVCPSW
  Address ACCRET to address part of SVCPSW
  Return to address NRMRET with all I/O and external interruptions disabled

- Exceptional condition has prevented the starting of the I/O operation or the I/O operation has started and terminated with an exceptional condition:

  STRTBT=1
  Place OSVPSW at SVCPSW
  Address ACCRET to address part of SVCPSW
  Return to address EXCRET with all I/O and external interruptions disabled

An I/O interruption related to this request interrupts the Simulator and gives control to CONTPR. If examination of the interruption indicates that not all the activity related to the request has terminated, control is returned to the point of interruption. Otherwise, control is returned to one of the addresses NRMRET or EXCRET, according to the conditions described under "Exceptional Conditions." The I/O old PSW is placed in the double-word with address SVCPSW, and all I/O and external interruptions are disabled.

An I/O request and continue calling sequence for a device in the busy or chained state is not allowed when SIM20 is in the disabled state.

This SVC calling sequence is not used by EDITOR and UPDT20.

I/O Request and Interrupt at Channel End (SVC 1)

The SVC calling sequence used to submit an I/O request and interrupt at channel end has the following form:

```
         CNOP   4,8
I        SVC    1
DEV360   DC     X'0ddd'
CAWADD   DC     A(CCWADD)
STATUS   DS     C
SNSADD   DS     3C
SVCCSW   DS     D
SVCPSW   DS     D
         DC     A(NRMRET)
         DC     A(EXCRET)
ACCRET   Any instruction
         (address I+36)
```

This calling sequence performs the same functions as the I/O request and continue, except for the following additional facility offered by the I/O request and interrupt at channel end.

When a channel-end condition occurs without the device-end condition, a test is made for the presence of a unit-exception or unit-check condition.

If neither of these conditions is present, the I/O old PSW is placed in the double-word with address SVCPSW, and control is returned to SIM20 at location NRMRET with all I/O and external interruptions disabled. Otherwise (channel end accompanied by unit check or unit exception), control is returned to SIM20 at the point of interruption. Thus, on devices for which channel end and device end occur separately, there can be two returns to NRMRET.

This SVC calling sequence is not used by EDITOR and UPDT20.

I/O Request and Wait (SVC 11)

The SVC calling sequence used to submit an I/O request and wait has the following form:

```
         CNOP   4,8
I        SVC    11
DEV360   DC     X'0ddd'
CAWADD   DC     A(CCWADD)
STATUS   DS     C
SNSADD   DS     3C
SVCCSW   DS     D
EXCRET   Any four-byte instruction
         (address I+20)
NRMRET   Any instruction (address I+24)
```

STRTBT is significant only in the I/O request and continue/interrupt at channel-end calling sequences. In the I/O request and wait, it always contains the value one (physical I/O operation initiated at the device) when control is returned to EDITOR.

I/O and external interruptions related to other I/O requests are allowed to occur while CONTPR is waiting for the I/O request and wait to terminate. Such interruptions are processed normally.

An I/O request and wait calling sequence is not allowed when EDITOR is in the disabled state.

This SVC calling sequence is not used by SIM20 and DSKINT.

Exceptional Conditions

Control is returned to SIM20 at address NRMRET with ERRTYP=0 (no exceptional conditions encountered) or at address EXCRET for the following conditions:

- No unit control block exists for this device (ERRTYP=2).
- The device or its associated control unit, subchannel, or channel is not operational (ERRTYP=1).
- A program-check or protection-check condition has been detected by the channel (ERRTYP=3).
- A unit-check condition has occurred (ERRTYP=0).

  A sense operation is performed on the device, and a maximum of three bytes of sense information are stored, starting at address SNSADD.
- A unit-exception or chaining-check condition has occurred (ERRTYP=0).

Note: The first two of these exceptional conditions are mutually exclusive, but the last three may occur concurrently. In this case, ERRTYP is set to the value corresponding to the first exceptional condition detected.

Channel Status Information

Information from the CSWs which can be generated as a consequence of the execution of an I/O request is accumulated in the double-word SVCCSW. On receipt of an I/O request, CONTPR sets the contents of SVCCSW to zero.

The execution of a chain of I/O commands produces, at most, one non-zero value each for the command address and count parts of a CSW. (CONTPR ignores any SVC in which the program-controlled interruption is the only status bit present.) The values of these quantities are set into the appropriate bytes of SVCCSW.

If non-zero values of the two status bytes are produced during the execution of a chain of I/O commands, CONTPR accumulates the logical "OR" of this status information in the appropriate bytes of SVCCSW.

If, when a chain of I/O commands has terminated, a unit-check status bit is present in SVCCSW, CONTPR performs a sense operation and places a maximum of three bytes of sense information, starting at address SNSADD.

When control is returned to SIM20 with ERRTYP=1 or 2, SVCCSW always contains zero, indicating that no I/O operation has started for the I/O request.

When control is returned with ERRTYP=3, the I/O operation has terminated, and SVCCSW (and, in the case of unit check, the bytes at SNSADD) describes the state of this termination.

I/O PROCESSING WITHIN CONTPR

The following paragraphs contain an outline of the techniques used by CONTPR in scheduling input/output operations. These techniques are explained with particular reference to the I/O request and continue (SVC 2) calling sequence.

Control Blocks

CONTPR associates a block of 28 bytes of System/360 main storage, called a unit control block, with each System/360 device. Each unit control block contains information giving the address, characteristics, and status of a device.

With each System/360 channel, CONTPR associates a block of eight bytes of System/360 main storage. This block is called a channel control block and is used to control the chaining of I/O requests for the associated channel (see "Chaining I/O Requests").

Three general registers (I, J, and K) are assigned to contain the addresses of the first byte of an SVC calling sequence, of a unit control block, and of a channel control block, respectively. Hence, using I, J, or K, any element in an SVC calling sequence, in a unit control block, or in a channel control block may be expressed as a displacement augmented by the contents of I, J, or K.

Processing an SVC 2 Calling Sequence

Charts FC and FD show how an SVC 2 calling sequence is processed by CONTPR. This processing is divided into two parts, as follows:

Part 1:	This part is entered once to initiate the physical I/O operation, if possible.
Part 2:	This part may be entered more than once. It is entered once for each I/O interruption associated with the physical activity initiated by part 1.

GETUCB Routine

The GETUCB routine uses the System/360 device address to generate the index pair (J,K). It uses the device table DEVTAB. There is one DEVTAB table for each System/360 channel. Each DEVTAB table contains a set of consecutive full-word entries corresponding to the devices attached to a channel.

Bits S through 7 of each word contain the System/360 address of the device, excluding the channel part.

Bits 8 through 31 of each word contain the address of the associated unit control block.

STRTIO Routine

The STRTIO routine tries to initiate an I/O operation and, depending upon the conditions encountered, has one of the following exits:

TERM	The operation has been started and terminated immediately without unit check, unit exception, or other error conditions.
DEVBSY	The device is busy with some previous I/O request.
PATHB	The associated channel, subchannel, or control unit is busy, or the operation must be delayed because some outstanding sense requests have not yet terminated.
EXCEPT	The operation cannot be started because of a unit-check or unit-exception condition on the device. The operation has been started and terminated with a unit-check or unit-exception condition. The device is not operational. A program-check condition has occurred.
CHEND	The operation has produced an immediate channel-end condition.
START	The operation has started without any immediate status conditions.

For the exit EXCEPT, if a unit-check condition has occurred, the STRTIO routine calls the SENSE subroutine.

SENSE Subroutine

The SENSE subroutine carries out a sense operation and places the first three sense bytes in the SVC calling sequence, and the last three sense bytes in the unit control block.

IOINT Routine

The IOINT routine is entered following an I/O interruption, and has one of the following exits:

TERM	The operation has terminated without unit check, unit exception, or other error conditions.
EXCEPT	The operation has terminated, and a unit-exception, chaining-check, or protection-check condition has occurred.
SENSE	The operation has terminated, and a unit-check condition has been detected.
CHEND	A channel-end condition has been detected.
OTHER	None of the above.

In the case of the exit SENSE, a request to carry out a sense operation for this device is added to the chain of waiting I/O requests for the associated channel. Furthermore, CONTPR has a parameter, labeled SNSCNT, whose value is equal to the number of outstanding sense requests on all System/360 channels. If any I/O request is attempted when SNSCNT is non-zero, the STRTIO routine returns to exit PATHB. This is done to avoid destroying sense information by a subsequent I/O request.

Chaining I/O Requests

The state of a given device at any moment is determined by CONTPR from information in its associated unit control block. CONTPR treats each device as being in one of the following three states:

• Busy	CONTPR has started activity for some I/O request, and this activity has not yet terminated.
• Chained	Not busy; an SVC 1 or SVC 2 calling sequence for the device has been received, but cannot yet be executed.
• Available	Not busy and not chained.

Any I/O interruption, except an attention interruption, received for a device which is in the available or chained state is ignored.

Available State

I/O REQUEST AND WAIT: If the device for which the request is received is in the available state, CONTPR tries to start the corresponding I/O operation. If the status of the channel, subchannel, control unit, or device precludes initiation of the operation, CONTPR cycles[1] on the SVC calling sequence until the request is accepted. Otherwise, the operation is started, the busy state is set, and CONTPR cycles on the SVC calling sequence until all related I/O interruptions have been received and processed.

I/O REQUEST AND CONTINUE/INTERRUPT AT CHANNEL END: If the device for which the request is received is in the available state, CONTPR tries to start the corresponding I/O operation. It sets either the busy state (operation started) or the chained state (operation waiting) and returns control to SIM20 at address ACCRET (request accepted).

Busy or Chained State

Any I/O request received for a device in the busy or chained state causes CONTPR to cycle on the new SVC calling sequence.

Adding a Request to a Chain

The channel control block with, for example, address K for a particular System/360 channel contains two full-word quantities labeled IOQBEG(K) and IOQEND(K), used in chaining I/O requests for this channel. Furthermore, each unit control block with, for example, address J attached to this channel contains a full-word quantity labeled DEVCHN(J).

Initially, when no requests are chained:

- IOQBEG(K) contains zero.
- IOQEND(K) contains the address of IOQBEG(K).

---

[1]To "cycle" means that CONTPR places the address of the SVC instruction into the address part of the supervisor-call old PSW and then loads this PSW. Thus, CONTPR returns to the SVC instruction, which is repeated until the operation can be initiated.

- DEVCHN(J) contains zero for all the unit control blocks on the channel.

To add a request to a chain, the following steps are carried out:

1. Extract the address contained in IOQEND(K).
2. Place at this address the value of J associated with the I/O request.
3. Place DEVCHN(J) at the address IOQEND(K).

Then, if two values of J (for example, J_1 and J_2) are added to the chain:

- IOQBEG(K) contains J_1.
- DEVCHN(J_1) contains J_2.
- DEVCHN(J_2) contains 0.
- IOQEND(K) contains DEVCHN(J_2).
- DEVCHN(J) contains 0 for all other devices on this channel.

Types of Requests Chained

Two types of request may be added to a channel chain:

1. SVC 1, SVC 2, SVC 13, and SVC 14 requests for which the exit PATHB is taken when the STRTIO routine is called
2. Sense operation requests for which the exit SENSE is taken when the IOINT routine is called

(A parameter in the unit control block enables CONTPR to distinguish between these two types of request.)

UNSTAK Routine

The UNSTAK routine (see Chart FE) attempts to initiate as many I/O operations on a designated channel as possible. The routine is entered with one input parameter, the channel index K.

Any unit control block for which an I/O operation is started (or is inhibited owing to exceptional conditions) is removed from the chain of requests for this channel.

SETTING UP THE SEREP INTERFACE (SVC 7)

If certain unrecoverable conditions are encountered during the execution of an I/O request, there is no return from CONTPR to EDITOR. Instead, the standard SEREP interface is set up.

During the execution of an I/O request, one of the following conditions may occur:

- One or more of the channel status indications (channel control check, interface control check, channel data check) is detected.
- A channel or device status indication which should not occur is detected.
- A sense operation cannot be performed on a device. (Such a sense operation is attempted each time the execution of an I/O request gives rise to a unit-check condition.)

In these situations, CONTPR sets up in main storage the elements necessary for the standard SEREP interface. It loads a PSW in which I/O and external interruptions are disabled, in which the wait state bit is one, and for which all the interruption code bits are ones.

In all other cases, control is returned to EDITOR.

EDITOR may find it necessary, as a result of the conditions under which an I/O request has terminated, to set up the SEREP interface. (For example, the condition ERRTYP=1 may be interpreted as a SEREP condition.)

The following calling sequence should be used in EDITOR to request that CONTPR set up the standard SEREP interface:

```
             CNOP     2,4
   I         SVC      7
   TYPE      DC       X'tt'
             DC       AL3(IOREQ)
```

tt
: denotes the type of interface which is required. Thus:

tt=0F	indicates a channel failure.
tt=1F	indicates a device failure.
tt=3F	indicates a device-not-operational condition.

IOREQ
: denotes the address of the SVC instruction in the calling sequence of the I/O request which gave rise to this SEREP condition.

CONSOLE COMMUNICATION

Two types of console communication are handled by CONTPR. The first type allows a message to be sent from EDITOR to the 1052 Printer-Keyboard, and the second allows transmission of a command from the 1052 to EDITOR in response to an attention interruption from the operator. There are no facilities for processing queues of messages or commands. SIM20 and DSKINT do not use CONTPR for console communication.

Write Message (SVC 4)

A request to write a message can be submitted by EDITOR to CONTPR using an SVC calling sequence of the following form:

```
            CNOP      2,4
   I        SVC       4
   N        DC        X'nn'
            DC        AL3(BUFF)
   I+6      Any instruction
```

The bytes to be printed are taken from locations

BUFF+1,BUFF+2,...BUFF+X'nn'

CONTPR sends the contents of these bytes to the 1052 Printer-Keyboard, using a Write Inhibit Carrier Return command. Consequently, if a new line is required at the end of the message, the "new line" character should be set up in location BUFF+X'nn'.

If, when a write message calling sequence is submitted, CONTPR is busy with a read or write request for the printer-keyboard, it cycles on the calling sequence until the previous request has terminated. When it accepts the calling sequence, it sets the contents of the byte at address BUFF to X'00', initiates the writing of the message, and returns control to EDITOR at address I+6.

When the request has terminated, CONTPR sets the byte at address BUFF to some non-zero value. Thus:

- A programming error has been detected. This probably indicates that part of CONTPR has been overwritten (BUFF= X'03').

- A device error has been detected during the printing of the message. CONTPR repeats the message; if a second error occurs, a control alarm is issued (BUFF=X'01').

  If no second error occurs, BUFF=X'07'.

- A device error has prevented the printing of the message. CONTPR tries to repeat the operation. If the failure occurs again, a control alarm is issued, and the SEREP interface is set up.

  If the failure does not occur again, BUFF=X'07'.

- The message was written without error (BUFF=X'07').

When EDITOR is in the disabled state, a write message request cannot be submitted unless the disabled state was caused by an interruption resulting from an operator command at the 1052 Printer-Keyboard.

This SVC calling sequence is not used by SIM20 and DSKINT.

Command Input (SVC 5)

When the attention key on the 1052 Printer-Keyboard is pressed, EDITOR is interrupted and CONTPR is entered. In response to this interruption, CONTPR sets up and executes a read command. Information is read from the 1052 into a command buffer. When the reading operation has terminated, control is returned to EDITOR at a predetermined address.

Before information can be transmitted from the 1052 to the Simulator, an SVC calling sequence of the following form must be submitted:

```
            CNOP      6,8
   I        SVC       5
   N        DC        X'nn'
            DC        AL3(BUFF)
   COMLEN   DC        X'00'
            DC        AL3(COMRET)
   COMPSW   DS        D
   I+18     Any instruction
```

This calling sequence need be presented to CONTPR only once, and the parameters which it contains are used, as described below, in conjunction with all the commands from the operator. (Any attention interruptions which occur before this calling sequence is submitted are ignored.)

The byte at address COMLEN contains the number of characters read.

X'nn' denotes the maximum number of characters that can be read. Hence, the number of characters, COMLEN, can never exceed X'nn'.

The characters of any command are placed in locations

BUFF+1,BUFF+2,...BUFF+COMLEN

The following termination conditions may be associated with the reading of a command:

- A device error has been detected during the reading of the command. CONTPR issues an error message for the operator and returns control to the point of interruption. Thus, the command is ignored.

- A control alarm is issued and the SEREP interface is set up. This may be the result of one of the following conditions:

  1. A device error has prevented the reading of the command. CONTPR has retried the operation and the failure has occurred again.

  2. The error message to the operator, in the case of a device error during the execution of a read command, cannot be written.

- A programming error has occurred. This probably indicates that part of CONTPR has been overwritten (BUFF=X'03').

- The command has been read without error (BUFF=X'07').

For the last two of these termination conditions, control is returned to EDITOR at location COMRET with all I/O and external interruptions disabled. The PSW of EDITOR at the point of interruption is placed in location COMPSW.

To avoid the possibility of overwriting the information in the command buffer by a subsequent command from the operator, the sequence starting at location COMRET should have completely processed this information before returning to the point of interruption.

The cancel condition at the 1052 Printer-Keyboard is treated normally; that is, a new request to read from the 1052 is issued.

This SVC calling sequence is not used by SIM20 and DSKINT.

<u>Disable Console (SVC 15)</u>

The SVC calling sequence

```
I     SVC       15
```

causes attention interruptions (resulting from an operator command on the 1052 Printer-Keyboard) to be ignored.

This SVC calling sequence is not used by SIM20 and DSKINT.

<u>Enable Console (SVC 16)</u>

The SVC calling sequence

```
I     SVC       16
```

causes such attention interruptions to be accepted if an SVC 5 calling sequence (set command parameters) has been previously submitted.

This SVC calling sequence is not used by SIM20 and DSKINT.

SIM20 INTERRUPTION AND RETURN (SVC 3)

When an interruption occurs, control is given to CONTPR, which may return control either to the point of interruption or to a predetermined location. In the latter case, the old PSW at the point of interruption is stored in a double-word at a predetermined address. In addition, all I/O and external interruptions are disabled.

Control may be returned to the point of interruption by using an SVC calling sequence of the form:

```
          CNOP     2,4
I         SVC      3
          DC       A(RETPSW)
```

where RETPSW denotes the predetermined address at which CONTPR has stored the old PSW.

The current PSW is replaced by the contents of the double-word with address RETPSW, thus returning control to the point of interruption.

REWIND AND REWIND-AND-UNLOAD CALLING SEQUENCES (SVC 13 AND 14)

When an I/O request and continue calling sequence is used to rewind or to rewind and unload a 2400-Series Magnetic Tape Unit, the operation is normally terminated (and EDITOR interrupted) only when the device-end signal is received from the tape unit.

The two SVC calling sequences given below enable CONTPR to terminate the operation when the channel-end signal is received. In this case, I/O interruptions for the tape unit which occur after the channel-end signal has been received are ignored.

The following SVC calling sequences are used for the rewind and rewind-and-unload functions:

```
            CNOP     4,8
   I        SVC      13 (Rewind)
or
   I        SVC      14 (Rewind-and-Unload)
   DEV360   DC       X'0ddd'
   CAWADD   DC       A(CCWADD)
   STATUS   DS       C
   SNSADD   DS       3C
   SVCCSW   DS       D
   SVCPSW   DS       D
            DC       A(NRMRET)
            DC       A(EXCRET)
   ACCRET   Any instruction
            (address I+36)
```

When a channel-end condition occurs without device end, the following tests are made.

Rewind: Has a unit-exception or unit-check condition occurred?

Rewind-and-Unload: Has a unit-exception condition occurred?

If not, control is returned to EDITOR at location NRMRET and the device-end condition is ignored. Otherwise, the termination of this operation is identical to that of the I/O request and continue operation.

CONTPR makes no check for the validity of the command code in the CCW provided by EDITOR. Thus, in EDITOR, a command code corresponding to the operation to be performed must be placed in the CCW. If it is not, CONTPR treats the calling sequence as an I/O request and continue calling sequence, but terminates the operation as a rewind or rewind-and-unload.

These two SVC calling sequences are not used by SIM20 and DSKINT.

SET WAIT STATE (SVC 19)

The SVC calling sequence

```
   I    SVC      19
```

sets the wait state bit "on" in the current PSW. ALL I/O and external interruptions are enabled.

When an I/O or external interruption occurs, CONTPR is entered. The wait state bit is set "off" in the old PSW at the point of interruption, and control is returned either to the point of interruption by loading the old PSW, or to a predetermined location. The old PSW at the point of interruption is also stored at a predetermined location.

This SVC calling sequence is not used by SIM20 and DSKINT.

DUMP SYSTEM/360 MAIN STORAGE (SVC 12)

This SVC calling sequence is not used by the 1620 Simulator.

INTERFACE WITH IOPACK

The two following SVC calling sequences are used to request that an I/O operation be performed on an EDITOR support device. On receiving the calling sequences, CONTPR transfers control to IOPACK.

These two SVC calling sequences are not used by SIM20 and DSKINT.

Assign a System/360 Device to a Simulator Support Function (SVC 17)

Before a request for an I/O operation by an EDITOR support device can be submitted to IOPACK, an SVC 17 calling sequence must be submitted to CONTPR.

Execute an I/O Operation on a Simulator Support Device (SVC 18)

To execute an I/O operation on an EDITOR support device, an SVC 18 calling sequence must be submitted to CONTPR.

I/O SUPPORT PACKAGE (IOPACK)

IOPACK (see Chart FF) is a subprogram consisting of a set of routines which perform logical I/O operations on System/360 I/O devices used for EDITOR support functions. It also performs logical I/O operations for UPDT20; but, to simplify the presentation, this description refers only to EDITOR.

The I/O operations which IOPACK is designed to perform and the associated System/360 devices are given in Table 14.

All these routines are designed for non-overlapped operation. Thus, program execution is suspended until the I/O operation has terminated.

IOPACK examines the error conditions which can occur when operating the devices given in Table 14, and takes the action prescribed by System/360 standards. Operator message facilities are provided via the 1052 Printer-Keyboard.

SYSTEM/360 DEVICE ASSIGNMENT (SVC 17)

When an SVC 17 calling sequence is submitted to CONTPR, control is transferred to IOPACK.

Table 14. Logical I/O Operations

OPERATION	SYSTEM/360 DEVICE
Read a card	1442 Card Read Punch, Model N1 2501 Card Reader, Model B1 or B2 2520 Card Read Punch, Model B1 2540 Card Read Punch
Punch a card (optional)	1442 Card Read Punch, Model N1 2520 Card Read Punch, Model B1 2540 Card Read Punch
Write a message	1052 Printer-Keyboard
Read a command	1052 Printer-Keyboard
Print a line	1403 Printer 1443 Printer
Print a line and skip to the first line on the next page	1403 Printer 1443 Printer
Read a tape record	2400-Series Magnetic Tape Unit Model 1, 2, or 3
Write a tape record	2400-Series Magnetic Tape Unit Model 1, 2, or 3
Write a tape mark	2400-Series Magnetic Tape Unit Model 1, 2, or 3

The SVC 17 calling sequence has the following form:

```
          CNOP     0,4
I         SVC      17
SYMBOL    DS       8C
DEV360    DC       X'0ddd'
TYPE      DC       C'tttt'
IOTYPE    DS       C
          DC       AL3(ERROR)
I+20      Any instruction
```

This calling sequence assigns the System/360 device address given by DEV360 to the symbolic name SYMBOL.

SYMBOL is a symbolic name assigned by EDITOR to a System/360 device. This name may contain from one to eight characters, being any combination of alphabetic and numeric characters. The first character must be alphabetic, the symbolic name is left-adjusted, and all remaining characters in the eight-byte field must be blank.

IOTYPE is one character, I or O, which specifies the type of operation (input or output) to be performed on the device named SYMBOL. "ddd" denotes the System/360 address and "tttt" the type of this device.

The types of device and the operations accepted on them are as follows:

2540,2520,1442	I (O optional)
2501	I
1403,1443	O
2400	I and O
1052	I and O

With each SYMBOL,DEV360 group is associated a block of control information in a table called SYMTAB.

IOPACK verifies the following conditions:

- SYMTAB is not full.

If the table is full, IOTYPE is set to X'01'.

- A routine exists for the operation to be performed and for the device to be used.

  If not, IOTYPE is set to X'02'.

- A unit control block in CONTPR exists for this device.

  If not, IOTYPE is set to X'03'.

In the above cases, when IOTYPE is set to X'02' or X'03', control is returned to EDITOR at location ERROR. Otherwise, the SYMBOL,DEV360 group is placed in SYMTAB and control is returned to EDITOR at location I+20.

SYMTAB can contain a maximum of 10 SYMBOL,DEV360 groups. Once an entry is placed in the table, it cannot be removed. Therefore, the SVC 17 calling sequence either adds a new SYMBOL,DEV360 group to the table (if the table is not full), or assigns a different System/360 device to a symbol already in the table.

SYMTAB is created when IOPACK is initialized, before EDITOR is loaded. The contents of the table remain unchanged when control is transferred from RELLDR to EDITOR.

Control Card Entry: The functions of the SVC 17 calling sequence may be performed by entering a control card at the time of program initialization. This control card has the following format:

```
/ DEVSUP SYMBOL=X'ddd',tttt,IOTYPE
```

where SYMBOL, "tttt", "ddd", and IOTYPE denote the same quantities as in the SVC 17 calling sequence. The blanks before and after DEVSUP must be respected.

EXECUTE A LOGICAL I/O OPERATION (SVC 18)

The SVC calling sequence used to request a logical I/O operation on an EDITOR support device has the following form:

```
        CNOP      0,4
I       SVC       18
SYMBOL  DS        8C
COUNT   DC        FL2'nn'
BUFFER  DC        A(BUFF)
I+16    Any instruction
```

SYMBOL denotes the same quantity as in the SVC 17 calling sequence. COUNT contains the number of bytes of data to be processed, and BUFFER contains the address of the I/O buffer for the device being used.

The data is fetched from or placed in locations

BUFF+1,BUFF+2,...BUFF+X'nn'

For an output operation on the 1403 or 1443 Printer, the EBCDIC character in location BUFF+1 specifies the type of print command, as follows:

The character "1"	Write and skip to channel 1 after printing.
Any other character	Write and space one line after printing.

Thus, the data is fetched from locations

BUFF+2,BUFF+3,...BUFF+X'nn'

For an output operation on a 2400-Series Magnetic Tape Unit, it may be necessary to write a tape mark (particularly after a unit exception has occurred, denoting the end of tape). To write a tape mark, COUNT must contain one (nn=1) and BUFF+1 must contain a 7-8 punch (hexadecimal 7F).

The I/O operation is performed using an SVC 11 calling sequence (I/O request and wait). CONTPR cycles on the SVC 11 calling sequence until the request has terminated. The request may terminate in any of the following ways:

- An unrecoverable error has occurred. Control is returned either from CONTPR to IOPACK, or from IOPACK to EDITOR. In the first case, the standard SEREP interface is set up. In the second case, a message is issued requesting that a System/360 dump program be loaded (a part of the system has probably been over-written), or that the standard SEREP program be loaded (a machine malfunction has been detected).

- The device SYMBOL is unknown to IOPACK. It has not been defined by a control card, nor by an SVC 17 calling sequence. The byte at address BUFF is set to the value X'01'.

- A device malfunction has been detected during the execution of the I/O request, and a message has been issued to inform the operator of the malfunction. IOPACK has received a command to terminate the I/O operation. The byte at address BUFF is set to the value X'02'.

- A unit-exception condition has occurred during a read or write operation on a magnetic tape unit. A message is issued and control is returned to EDITOR, with the byte at address BUFF set to the value X'03'.

- None of the above conditions has occurred; that is, the I/O operation has terminated with no exceptional conditions. The byte at address BUFF is set to the value X'07'.

In the last of these cases, IOPACK returns control to EDITOR at location I+16.

INITIALIZATION PROGRAM (INIT)

This subprogram (see Chart FG) initializes CONTPR, IOPACK, and RELLDR for an EDITOR or UPDT20 run. To simplify the presentation, this description refers only to EDITOR, but is valid for both programs. Initialization is performed in the following manner:

CONTPR: It initializes the 1052 Printer-Keyboard Read/Write routine and creates the channel and unit control blocks.

IOPACK: It creates SYMTAB, which assigns System/360 devices to the symbolic names of EDITOR support devices.

RELLDR: It selects the program to be loaded, defines the length of the loader tables, the output device to be used by the Self-Loading Program Generator routine (if it is required), and the names of any control sections which must not be loaded.

Three types of control card are used by the program for the above functions; these are, respectively, DEV360, DEVSUP, and CALL cards. The format and contents of these cards are described in the publication IBM System/360 Conversion Aids: The 1620 Simulator for IBM System/360, Form C28-6529. The program translates the mnemonic operand terms in the cards by means of a dictionary (DICT) which contains, against each operand, the action to be taken and any data required for this action.

PROGRAM STRUCTURE

To simplify the presentation, the program may be divided into five phases.

Phase 1

ABSLOD has loaded CONTPR, IOPACK, INIT, and RELLDR, and has prepared the following data in the general registers:

- The address of the last byte of CONTPR
- The current value of the location counter (address of the first byte following IOPACK)
- The address in RELLDR to which control must be transferred at the end of INIT

Control is transferred to INIT, which needs to read control information but does not know the address of the device on which to read it. It cannot issue a message to the operator since the address of the 1052 Printer-Keyboard is also unknown, so it sets the system in the wait state. The operator then presses the request key on the 1052, causing an attention interruption. The program now inserts in a CCB and a UCB at the end of the program the address of the 1052 which is recorded in the I/O old PSW, and issues a message to the operator requesting the address of the control information input device.

When the operator has typed a command indicating the type and address of the input device, a UCB (together with a CCB if the device is not on the same channel as the 1052 Printer-Keyboard) is completed for this device.

Phase 2

Each control card or card image (on tape) is read, listed on the 1052 Printer-Keyboard, and analyzed with the aid of the dictionary (DICT); the result is then entered, in condensed form, in a table (TABLE). This table is created in front of INIT during program execution and overlays the first routine (initialization) of this program. The table is filled backwards; that is, the first element is contiguous to INIT, and the table is extended to the front as new elements are added. The condensed DEV360 card images are sorted in order of increasing channel-unit addresses and are placed in the first part of the table; the condensed DEVSUP card images are placed after the last DEV360 card image, in the order in which they are read. The length of the table is adjusted as each card image is entered.

The last card in the deck is the CALL card and it is processed as follows:

1. The name of the program to be loaded by RELLDR is placed in the dictionary for later use.
2. If the selective loading feature is to be used, flags are set in the list of optional control sections against those which will be required.
3. If the term LIST is present, a flag is set on to inform RELLDR that, later, it must print loading messages.

4. If the term "INIT=nnnnnn" is present, the symbolic name "nnnnnn" is saved to inform RELLDR on which output device the self-loading program must be generated.

5. A card punching routine, which may be used by the Self-Loading Program Generator routine, is included at the end of IOPACK. If the output device called by the term "INIT=nnnnnn" is not a card punch, this card punching routine will not normally be required; therefore, the location counter is decremented by the length of this routine to save storage space. Should the card punching routine be required in another program (UPDT20, for example), the term PUNCH must be added to the operand field of the CALL card to prevent the location counter from being decremented.

Phase 3

The program uses the data contained in the first part of the table (TABLE) to build up channel and unit control blocks in storage, starting at the address contained in the location counter.

One channel control block is created for each available System/360 channel, and one unit control block for each available device. The format of channel and unit control blocks is discussed in the section "I/O Simulation."

As each control block is created, the location counter is incremented and, at the end of this phase, it points to the first byte following the last unit control block created.

Phase 4

With the data stored in the second part of the table (TABLE), the program creates SYMTAB in IOPACK by means of SVC 17 calling sequences. This calling sequence is described in the section "I/O Support Package."

If the EDITOR support device which will be used to load the program specified in the CALL card is not defined at this point, the program stops after issuing an error message, and cannot continue.

Phase 5

The program now prepares to transfer control to RELLDR. The name of the program to be loaded, which is in the dictionary, is used to check that the file about to be read is the correct one. The first card or card image in the file is read. This first card is the PROGNAME card, which contains the name of the program and the size of the loading tables required to load it.

It is assumed that in the case of a program on cards, the first card read will be the correct one; that is, that unwanted programs will have been removed. Should this not be the case, an error message will be printed and the program will stop.

In the case of a program on tape, if the name in the PROGNAME card image is not the name required, the file will be skipped, and the next PROGNAME card image will be read and checked. This action is repeated until the correct file is found or until the SYSINEND card image is met. In the latter case, an error message will be printed, and the program will stop.

INIT prepares a list of parameters for RELLDR which contains, in all cases, the following items:

- The size of the loader tables needed to load the specified program
- The current value of the location counter
- A flag to indicate whether or not loader messages should be issued

The list of parameters may also contain one or more of the following items, depending on which terms were present in the CALL card:

- The symbolic name and the address of the output support device
- The name(s) of the control section(s) to be ignored in the program about to be loaded

If the term "INIT=nnnnnn" was present in the CALL card, and if the 1052 Printer-Keyboard used before the self-loading program is created is not the same as the one to be used afterwards, then the parameters prepared for RELLDR must include the address of the 1052 to be used after the self-loading program has been created, and the address of its unit control block.

Once the list of parameters is complete, INIT transfers control to RELLDR at the address which was specified by ABSLOD. The system then operates in the problem state, disabled for all interruptions except a machine check or a program check.

CARD SEQUENCE

DEV360 and DEVSUP cards may be mixed and in any order, but the last card must be the CALL card since, in addition to the functions indicated above, it marks the end of control information input. Should the CALL card not be the last, the DEV360 and DEVSUP cards placed after it will be ignored and will cause an error at some time during program execution.

If two DEV360 cards define different devices at the same address, only the latter definition is retained. Similarly, if two DEVSUP cards assign the same symbolic name to two System/360 devices, only the latter assignment is retained.

LINKAGE WITH CONTPR AND IOPACK

INIT, after it has initialized CONTPR during phase 1, uses both that program and IOPACK to read the control information and to issue messages. The linkage between these programs is discussed in the sections "Control Program" and "I/O Support Package."

MESSAGES

Messages are printed by INIT on the 1052 Printer-Keyboard to inform the operator of any errors detected while the control information is read or during the initialization itself (phases 3, 4, and 5). These messages are listed and explained in the publication IBM System/360 Conversion Aids: The 1620 Simulator for IBM System/360, Form C28-6529.

RELOCATING LOADER (RELLDR)

RELLDR (see Chart FH) is used to load EDITOR or UPDT20, whichever is specified in the CALL control card.

The distinguishing feature of this loader, as opposed to ABSLOD, is its ability to load control sections into storage at addresses other than those assigned by the assembler; that is, to relocate them, and to complete linkage among the sections by means of special tables.

RELLDR uses the location counter (LOCCTR) to determine where control sections will be loaded. Initially, LOCCTR indicates the first byte that follows the last unit control block created by INIT. Thereafter, it is incremented by the number of bytes indicated in an ESD type 0 term (see "ESD Type 0 Term (Control Section Name)"), or by the length indicated on an ICS card (see "Include Control Section Card"); or it may be set to a definite value by an SLC card (see "Set Location Counter Card"). Each time LOCCTR is incremented, the new value is compared with the low-order address of the loader tables to prevent these tables and the loader program from being overlaid. If an attempt is made to overlay the tables and loader program, an error halt occurs. After loading, however, when control has been transferred to the program loaded, the space occupied by the loader tables and program is available and may be overlaid.

SPECIAL RELLDR FUNCTIONS

RELLDR has not only the three functions of ABSLOD, that is, loading, correcting, and transferring control, but also the special functions described with their associated load cards in Table 15.

Table 15. Special RELLDR Functions

FUNCTIONS	CARDS
Relocating: Can place the instructions and constants of a control section into storage locations other than those assigned by the assembler; that is, relocate them.	Set Location Counter (SLC) Include Control Section (ICS) External Symbol Dictionary (ESD type 0 term) Text (TXT), Replace (REP)
Linkage: Loads two or more control sections one after the other and completes linkage among them so that one control section may refer to constants or instructions, or both, within another; makes any changes necessary to evaluate address constants of up to four bytes which are used by the control section.	External Symbol Dictionary (ESD type 1 and 2 terms) Relocation List Dictionary (RLD) Replace (REP)
Transferring Control: Ends loading and causes control to be transferred according to the priority noted in the discussion of the LDT card.	Load End (END) Load Terminate (LDT)
Note: The function of the REP card is essentially the same as in ABSLOD. The END card remains an essential part of each control section, but is subordinate in function to the LDT card.	

LOADER TABLES

To relocate assembled addresses and to link the various modules or control sections, the loader uses three tables, referred to as the Dictionary, the Reference Table, and the Relocation List. These tables are built before storage just before RELLDR, overlaying ABSLOD, which is no longer required.

The Dictionary is used to list the symbolic names of all the control sections, entry points, and external symbols as they are encountered during the entire loading process, and their relocated addresses when they are known.

The Reference Table is used to relocate all the assembly addresses of a control section and to calculate, when possible, the value of the load constants in that section. Any load constant whose value cannot be calculated because the relocated address of the symbol to which it refers is not yet known, is placed in the Relocation List until it can be calculated.

When an END card is encountered, the Reference Table is cleared in preparation for the next control section, and the Relocation List is scanned to calculate the value of any load constants for which the relocated address of the related symbol is now known.

LOADER CARDS

The formats of the eight types of load card recognized by RELLDR are described in detail in the following sections, together with the manner in which each type of card is processed by the loader.

Set Location Counter Card

The Set Location Counter (SLC) card sets the location counter to an address indicated in one of three ways:

1. Any absolute address specified as a hexadecimal number punched in card columns 7-12.

2. Any symbolic address already defined as a control section name or entry point. This is specified by a symbolic name punched in card columns 17-22.

3. The sum of the absolute address in card columns 7-12 and the internal address of the symbolic name in card columns 17-22, if both these fields are specified.

If only one field is used, the other must be left blank. If both fields are blank, the SLC card is ignored and a warning message is issued. If the SLC card causes LOCCTR to be decremented, a warning message is also issued as LOCCTR is set to the new value.

The SLC card is normally placed in front of the control section to which it applies, but it will be recognized at any point within an assembled control section.

The contents of the SLC card fields are defined in Table 16.

Table 16. Set Location Counter Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	SLC. Identifies the type of load card.
5-6	Blank.
7-12	Address in hexadecimal (to be added to the value of any symbol specified in columns 17-22). The address must be right-justified in these columns. Unused columns are filled with zeros.
13-16	Blank.
17-22	Symbolic name (may be blank) whose internal assigned location will be used by the loader. The symbol must be left-justified in these columns. Unused columns are left blank.
23-72	Blank.
73-80	Not used by the loader.

Include Control Section Card

The Include Control Section (ICS) card is used to reserve storage space for a control section which will be loaded later. The card specifies the name and the length of the control section.

Control sections are loaded only on double-word boundaries. The loader automatically makes this adjustment before loading any given control section, in the following manner:

1. The location counter is adjusted, if necessary, to the next double-word boundary.

2. The symbolic name is stored in the Dictionary, with the current value of the location counter.

3. The length of the control section is added to the value of the location counter and the latter is set to the resulting sum to reserve the storage area.

Later, when the loader encounters a reference to this control section, its location is already known; and when the control section is loaded, it will be placed in the area of storage reserved for it.

The loader does not retain the length of the control section given by the assembler in the ESD type 0 term; therefore, the length specified in the ICS card must not be less than that specified in the ESD type 0 term. If it is, the control section concerned will overlap the next one in storage. Storage space may be reserved for REP cards by specifying a greater length in the ICS card. A warning message is issued if the length stated in the ICS card differs from that in the ESD type 0 term.

ICS cards are normally placed before the first card of a control section, but they will be recognized at any point within an assembled control section.

The contents of the ICS card fields are defined in Table 17.

Table 17. Include Control Section Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	ICS. Identifies the type of load card.
5-16	Blank.
17-22	Name of control section, left-justified.
23-24	Blank.
25-28	Length (in bytes) of the control section, in hexadecimal notation, right-justified. Unused leading columns are filled with zeros.
29-72	Blank.
73-80	Not used by the loader.

External Symbol Dictionary Card

The External Symbol Dictionary (ESD) cards are generated by the assembler. These cards may contain three types of term:

1. The ESD type 0 term defines the name, the assembled starting address, and the length of a control section. It is produced by the assembler when it encounters a START instruction. Only one ESD type 0 term is produced per control section, and it is assigned an external symbol identification (ESID) of 01.

2. The ESD type 1 term defines an entry point within the control section to which another section may refer. One ESD type 1 term is produced by the assembler each time it encounters an ENTRY instruction.

3. The ESD type 2 term points to a name within another control section to which this section may refer. One ESD type 2 term is produced by the assembler each time it encounters an EXTRN instruction, and is assigned an ESID of from 02 onwards, in the order in which it is encountered among the external symbols of the control section being assembled.

The variable field on the ESD card may contain up to three terms, which may be of the same or of mixed type. The contents of the common fields of the ESD card are defined in Table 18; the contents of the variable field will be discussed under each type of term.

ESD Type 0 Term (Control Section Name)

This term defines the name, or entry point, of the control section. It is produced by the assembler when it encounters a START instruction. If the START instruction does not specify a control section name, blanks will be placed in the Dictionary to define that "name."

The assembler assigns an external symbol identification of 01 (ESID 01) to the control section. This number is used by the loader as a pointer to the Reference Table entry. This entry is created by the loader when it processes the ESD type 0 term, and contains the address of the control section name in the Dictionary and the address at which it was assembled. The ESID 01 appears in the ESD type 0 term, in all the ESD type 1 terms, and in all TXT, RLD, and END cards produced by the assembler. The loader can thus calculate the

Table 18. External Symbol Dictionary Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	ESD. Identifies the type of load card.
5-10	Blank.
11-12	Number of bytes in the variable field (card columns 17-64), in extended card code.
13-14	Blank.
15-16	ESID of the first ESD type 0 or 2 term, if any, in the card.
17-64	Variable information field containing from one to three 16-column terms (see Tables 19, 20, and 21). Unused columns are left blank.
65-72	Blank.
73-80	Not used by the loader.

control section's relocation factor whenever it is needed; this factor is the difference between the address where the control section is loaded (recorded in the Dictionary) and that at which it was assembled (recorded in the Reference Table).

The address at which the control section will be loaded is determined by the following conditions:

1. If the name of the section defined by the ESD type 0 term is already in the Dictionary, then the section will be loaded at the location specified in the Dictionary, and no adjustment is made to the location counter.

2. If the name of the control section defined in the ESD type 0 term is not in the Dictionary, then:

   a. The location counter is adjusted, if necessary, to the next doubleword boundary.

   b. The control section name is placed in the Dictionary, with the current value of the location counter.

   c. The location counter is incremented by the length of the control section, and the section will be loaded at the value now specified in the Dictionary.

The loader loads only one control section at a time and clears all entries in the Reference Table when it encounters an END card. Since it does not save the ESIDs from one section to another, there is no conflict in the Reference Table when the next section is assigned the same number (ESID 01).

The contents of an ESD type 0 term are defined in Table 19.

Table 19. ESD Type 0 Term

COLUMN	CONTENTS
1-6	Control section name.
7-8	Blank.
9	Extended card code 12-0-1-8-9 (hexadecimal 00), identifying this as an ESD type 0 term.
10-12	Address, in extended card code, of the first byte of the control section as assigned by the assembler.
13	Blank.
14-16	Length, in bytes, of the control section (extended card code).

ESD Type 1 Term (Entry Point)

This term defines an entry point within the control section to which another section may refer. One such term is produced by the assembler each time it encounters an ENTRY instruction. All ESD type 1 terms are assigned the same ESID as that of the ESD type 0 term of the same control section.

The loader processes ESD type 1 terms by scanning the Dictionary to see whether the entry point has already been defined as an external symbol in another control section. If it has, the relocated address is now calculated and placed in the Dictionary against the name; if not, a new entry containing the name and the relocated address of the program entry point is created.

The contents of an ESD type 1 term are defined in Table 20.

Table 20. ESD Type 1 Term

COLUMN	CONTENTS
1-6	Name of entry point.
7-8	Blank.
9	Extended card code 12-1-9 (hexadecimal 01), identifying this as an ESD type 1 term.
10-12	Address, in extended card code, of the entry point as assigned by the assembler.
13-14	Blank.
15-16	ESID, in extended card code, assigned to the control section in which the entry point occurs.

ESD Type 2 Term (External Symbol)

This term points to a name within another control section to which this section may refer and is produced by the assembler when it encounters an EXTRN instruction. One term is produced for each external symbol thus defined and assigned an ESID of from 02 onwards as it is encountered in the program. This number is used as a pointer to the Reference Table entry and appears in the RLD card associated with that external symbol.

The loader processes an ESD type 2 term by scanning the Dictionary to see whether the external symbol has already been defined as an entry in another control section. If it has not, the symbol is entered in the Dictionary but the address is left blank. A Reference Table entry is then created which contains the address of the symbol in the Dictionary.

The loader loads only one control section at a time and clears all entries in the Reference Table when it encounters an END card. Since it does not save the ESIDs from one section to another, there is no conflict in the Reference Table when the next section is assigned the same numbers (02, 03, etc.).

The contents of an ESD type 2 term are defined in Table 21.

Text Card

The Text (TXT) card is generated by the assembler. It contains the instructions and constants of the program to be loaded, and the address at which the first byte of text in the card is to be loaded. Each card contains a maximum of 56 bytes of text in extended card code.

Table 21. ESD Type 2 Term

COLUMN	CONTENTS
1-6	Name of external symbol.
7-8	Blank.
9	Extended card code 12-2-9 (hexadecimal 02), identifying this as an ESD type 2 term.
10-12	Extended card code 12-0-1-8-9 (hexadecimal 00), three times. The address assigned to an external symbol by the assembler is always zero.
13-16	Blank.

The loader relocates the address in the card by the relocation factor of the control section to which the card belongs, and stores the contents of the card at that address.

The contents of the TXT card fields are defined in Table 22.

Table 22. Text Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	TXT. Identifies the type of load card.
5	Blank.
6-8	Address, in extended card code, at which the information on the card is to be loaded.
9-10	Blank.
11-12	Number of bytes of text in the card, in extended card code.
13-14	Blank.
15-16	External Symbol Identification (ESID) assigned to the control section in which the text occurs (in extended card code).
17-72	A maximum of 56 bytes of instructions and constants assembled in extended card code.
73-80	Not used by the loader.

Relocation List Dictionary Card

The Relocation List Dictionary (RLD) card is produced by the assembler when it encounters a DC instruction or the second operand of a CCW instruction which defines an address as a relocatable symbol or expression. This may be the address either of an internal symbol which occurs only within the control section or of an external symbol belonging to another section.

The contents of the RLD card fields are defined in Table 23.

The loader uses position and relocation headers (see Table 23) to enter the Reference Table and the Dictionary. It calculates the relocated address of the load constant and the value of the expression. If the latter cannot be computed because the relocation header refers to a symbol which has not been loaded yet, the loader places the loading address and the relocation headers of the load constant in the Relocation List. The loader scans the Relocation List at the end of each control section and finishes processing load constants which refer to symbols defined in that control section.

Table 23. Relocation List Dictionary Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	RLD. Identifies the type of load card.
5-10	Blank.
11-12	Number of bytes of information in the variable field (card columns 17-72), expressed in extended card code.
13-16	Blank.
17-72	Variable field in extended card code, consisting of the following subfields: <u>Relocation Header</u>. Two-byte ESID of the symbol in the load constant. The ESID is 01 if the symbol is internal to the control section, and greater than 01 if it is external. <u>Position Header</u>. Two-byte ESID assigned to the control section in which the load constant appears. <u>Flag Byte</u>. (Bits 0 to 3 are not used.) This byte contains three items: 1. <u>Size</u>. Bits 4 and 5 indicate the length in bytes of the load constant as follows: 00 - one byte 01 - two bytes 10 - three bytes 11 - four bytes 2. <u>Complement flag</u>. When bit 6 is a one, the value of the symbol must be subtracted from the expression in which it occurs. When bit 6 is zero, the value must be added. 3. <u>Continuation Flag</u>. When bit 7 is a one, it means that this is one of a series of expressions to be derived from the value of one symbol. When bit 7 is a zero, it means that this is the only expression, or the last expression, to be derived from the value of one symbol. <u>Address</u>. Three-byte address of the expression given by the assembler, in extended card code. The flag-byte and address fields may be repeated for other expressions as long as the continuation flag is on in the current four-byte entry.
73-80	Not used by the loader.

Replace Card

Replace (REP) cards are produced by the programmer to substitute new text for portions of assembled text. They must be placed immediately after the last TXT card. Assembled instructions or constants, or both, are replaced byte for byte by the instructions or constants punched in the card in hexadecimal code. A REP card may contain a minimum of 2 bytes (one half-word) and a maximum of 22 bytes. The assembled address of the first byte of text to be replaced, which must be stated in the card, will be relocated by the loader.

If additions made by REP cards increase the length of a control section, an ICS card must be placed at the front of the control section to define the new length of the section.

The contents of the REP card fields are defined in Table 24.

Table 24. Replace Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	REP. Identifies the type of load card.
5-6	Blank.
7-12	Address, in hexadecimal, of the area to be replaced. It must be right-justified and unused leading columns must be filled with zeros. The address must specify a half-word boundary.
13	Blank.
14-16	External Symbol Identification (ESID), in hexadecimal, assigned to the control section in which the replacement is to be made. If this number is not known, the field must be filled with zeros.
17-70	A maximum of eleven 4-digit hexadecimal fields, separated by commas, each replacing one previously loaded half-word (2 bytes). The last field must <u>not</u> be followed by a comma.
71-72	Blank.
73-80	Not used by the loader.

<u>Load End Card</u>

The Load End (END) card is produced by the assembler when it encounters the END instruction; it ends loading of the control section and may specify a location within the section to which control should be transferred. When the loader encounters this card, it clears the Reference Table and scans the Relocation List to finish processing any load constants related to symbols which have been defined in the control section just loaded.

The contents of the END card fields are defined in Table 25.

Table 25. Load End Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	END. Identifies the type of load card.
5	Blank.
6-8	Address (may be blank), in extended card code, of the point in the control section to which control may be transferred at the end of the loading process. See priority conditions discussed under the LDT card.
9-14	Blank.
15-16	External Symbol Identification (ESID) of the control section.
17-72	Blank.
73-80	Not used by the loader.

<u>Load Terminate Card</u>

The Load Terminate (LDT) card must be placed at the end of the input deck. It has two uses:

1. It ends the loading process.

2. It causes control to be transferred to some location within the section or sections loaded.

The contents of the LDT card fields are defined in Table 26.

When the loader encounters the LDT card, it scans the Dictionary to see whether all the symbols have a loading address; that is, whether they have been defined, and if they have not, it issues a warning message.

The loader then checks whether any errors have been detected during loading, and if so, it causes a message to be printed and the system to enter the wait state.

Table 26. Load Terminate Card

COLUMN	CONTENTS
1	Load card identification (12-2-9 punch). Identifies this as a card acceptable to the loader.
2-4	LDT. Identifies the type of load card.
5-16	Blank.
17-22	Name of the symbolic entry point to the loaded program in standard card code, left-justified. This field may be blank.
23-72	Blank.
73-80	Not used by the loader.

If no errors have occurred during loading, the loader transfers control to a location determined by the following order of priority:

1. The Self-Loading Program Generator routine if the CALL card specified that an initialized self-loading version of the program be produced on magnetic tape or punched cards
2. Any location specified in the LDT card
3. The first location specified by an END card encountered during the current loading process
4. The first byte of the first control section loaded by RELLDR

After control has been transferred to the loaded program, the system operates in the problem state, enabled for all interruptions. If a program check occurs, the program new PSW causes the loader to issue a message and to set System/360 in the wait state. This is true only as long as RELLDR has not been overlaid, nor the program new PSW altered, by the program being executed.

CARD SEQUENCE

The following list shows the sequence of cards in a series of control sections ready to be loaded by RELLDR; it does not show all the permissible combinations.

SLC	sets the location counter at an absolute address.
ICS	defines control section B as a section to be loaded later and specifies the length to be reserved for it.
ESD	defines the name and length of section A (type 0 term), any entry points in section A to which section B may refer (type 1 term), and any external symbols in section B to which section A refers (type 2 term).
TXT	contains instructions and constants of section A.
REP	contains changes or additions to section A.
RLD	contains information for evaluating relocatable addresses in section A.
END	can contain an address within section A to which control will be transferred after loading if no address is given in the LDT card.
ESD	defines the name and length of section B (type 0 term), any entry points in section B to which section A may refer (type 1 term), and any external symbols in section A to which section B refers (type 2 term).
TXT	contains instructions and constants of section B.
RLD	contains information for evaluating relocatable addresses in section B.
END	can contain an address within section B to which control will be transferred after loading if no address is given in the LDT card nor in the previous END card.
LDT	ends the loading process. If this card specifies an address for transfer of control, this overrules any address saved or specified by an END card.

OTHER FEATURES

In addition to its basic functions, RELLDR can be used for absolute loading.

RELLDR cannot implement the overlaying-load procedure for programs larger than available storage.

Loading in Absolute Form

If the ESD cards are removed from the control section before loading, RELLDR operates in a manner similar to ABSLOD.

The loader will load one or more control sections, either in absolute form or in both absolute and relocatable form, until it encounters an LDT card. However, the following restrictions apply:

1. The loader will not record in the loader tables the presence of a control section loaded in absolute form.
2. No linkage is provided with any section loaded in absolute form; therefore, if the user wishes to load a section at the location assigned by the assembler and have linkage with another section, he must specify the starting address with an SLC card and leave in the ESD cards.
3. If two or more control sections are loaded in absolute form, any common addresses in these sections will be overlaid.

The location counter setting depends on the SLC, ICS, and ESD (type 0 term) cards read during loading. If the control section to be loaded in absolute form is the first one, the location counter contains the address of the first storage location following the unit control blocks created by INIT. To avoid overlaying programs that have already been loaded, an absolute loading address must not be lower than the one contained in the location counter, nor greater than that of the start of the loading tables.

Selective Loading

The selective loading function saves storage space by loading only those parts of the Simulator which are necessary to simulate a particular installation.

Optional sections have two control section names, one defining the complete control section, the other defining a dummy or a shorter version of the section. Depending on the options specified in the CALL card, which are translated into the appropriate control section names by INIT, RELLDR will load one or the other section. When the loader is initialized, it creates a loading table (LDOPT) of control sections that must not be loaded.

Both control sections of any optional portion of the program must conform to the following requirements:

1. Their names must be different and no reference must be made to these names in any other portion of the Simulator.
2. They must contain the same number of entry points with the same names to avoid leaving any external symbols in other sections undefined.
3. They must be assembled separately, each one on its own.

SELF-LOADING PROGRAM GENERATOR ROUTINE

This routine is entered from RELLDR if the term "INIT=nnnnnn" was present in the CALL card processed by INIT ("nnnnnn" being the symbolic name of the output device). It creates a punched card deck or a magnetic tape file in a form which can be loaded by the System/360 IPL procedure. This card deck or tape file contains the contents of main storage between locations 24 and the last byte loaded by RELLDR (LOCCTR-1); that is:

- CONTPR
- IOPACK
- The program just loaded (specified in the CALL card)

Since CONTPR and IOPACK are needed to create the card deck or tape, the routine first copies this storage area into main storage beyond location LOCCTR.

The magnetic tape file consists of:

1. A 24-byte IPL record containing a PSW and two CCWs
2. A variable-length record containing a copy of the storage area concerned

The card deck consists of:

1. A 24-byte IPL card containing a PSW and two CCWs
2. A 56-byte IPL card containing a bootstrap loader made up of seven chained CCWs
3. As many TXT cards as are required to contain the storage area concerned
4. An END card to exit from the bootstrap loading loop and terminate the IPL procedure

The Self-Loading Program Generator routine ends by issuing one of the following messages and setting System/360 in the wait state:

1. A212A END OF INITIALIZATION - This is the normal end of the routine.
2. A213W INITIALIZATION ERROR, CANNOT CONTINUE - This message is issued if the area of storage between LOCCTR and

the beginning of this routine in main storage is too short to contain the copy of locations 24 to LOCCTR-1.

LINKAGE WITH CONTPR AND IOPACK

RELLDR uses CONTPR and IOPACK for all input/output operations, including messages. The linkage between these subprograms and RELLDR is discussed in the sections "Control Program" and "I/O Support Package."

RELLDR MESSAGES

RELLDR can produce off-line messages for the programmer if the parameter LIST is added to the CALL card after the other parameters it may contain. Under normal conditions, no messages are needed, or printed; the parameter LIST should be inserted in the CALL card only if loading does not terminate correctly to determine why during another attempt; in this case, the CALL card has the following format:

```
/ CALL EDITOR,LIST
```

The messages are recorded on the support device identified by the symbol SIM2PRNT, defined in a DEVSUP control card before EDITOR was loaded, and are classified under three headings:

Information	These messages are for information only, giving, for instance, the addresses at which the various control sections start.
Warning	These messages draw attention to possible recoverable errors which may occur but do not interrupt loading.
Error	These messages indicate any errors such that loading cannot continue.

In the following messages "xxxxxx" represents one of the following:

- A symbolic name
- The identification number punched in card columns 73-80
- The serial number of the card in the deck if columns 73-80 are blank

The term "card" refers either to an actual card or to a tape record in the form of a card image.

<u>Informative Messages</u>

RL00I xxxxxx CONTROL SECTION LOADED AT.....
The first byte of the control section identified by the symbol "xxxxxx" is loaded at the address stated.

RL01I * INITIAL PSW......
This is the PSW which transfers control to the program loaded.

<u>Warning Messages</u>

RL02I xxxxxx ILLEGAL CARD IN LOADER INPUT
The card indicated was not recognized by the loader and was ignored.

RL03I xxxxxx TXT FOLLOWS REP OR RLD CARD
The TXT card indicated was out of sequence. The loader processed the card, but part of the control section loaded may have been overlaid.

RL04I xxxxxx ADDRESS OUTSIDE C.S. OR C.S. ALREADY LOADED
The address contained in the TXT or REP card indicated is beyond the end of the control section being loaded or in a preceding control section already loaded. The card was ignored.

RL05I xxxxxx TXT CARD CONTAINS MORE THAN 56 BYTES
The loader stores as many bytes as are stated in card columns 11-12, but this may cause a program error at a later stage. This message will occur only if an error is made when correcting or repunching a TXT card.

RL06I xxxxxx TEXT OVERLAYS LOADER TABLES
The control section being loaded is longer than stated in the ESD card (type 0 term) or the ICS card and overlays the start of the loader tables. This message can occur only if the end of the control section has been modified by REP cards. The card indicated was ignored.

RL07I xxxxxx ESD CARD FOLLOWS TXT CARD
The card indicated was out of sequence and was ignored.

RL08I xxxxxx USED AS ENTRY AND CONTROL SECTION NAME
The symbol in the card indicated, already defined in an ESD type 1 term, has been found in an ESD type 0 term or vice versa. This might be a source of error.

RL09I xxxxxx CONTROL SECTION DEFINED WITH 2 LENGTHS
The control section defined in the

card indicated has been previously defined with a different length by an ESD type 0 term or an ICS card. The new definition was ignored.

RL10I xxxxxx LDT CARD NOT PRECEDED BY END CARD
The last control section loaded did not contain an END card. The loader assumed that there was one but that it did not specify any transfer address.

RL11I xxxxxx EXTERNAL SYMBOL HAS NO REAL DEFINITION
The symbol shown, which was recorded as an external, does not correspond to any entry point or control section name. This message is printed after the LDT card has been read.

RL12I xxxxxx BLANK OR COMMA MISSING IN REP CARD
The REP card indicated does not conform to the standard format and was ignored.

RL13I xxxxxx ADDRESS OF SYMBOL IN SLC CARD NOT RELOCATED
The symbol in the SLC card indicated has not been defined yet, has not been relocated yet, or does not exist. In the first two cases, the SLC card is out of sequence; in the last, the symbol is erroneous. The card was processed as though no symbol were specified.

RL14I xxxxxx NEITHER NAME NOR ADDRESS IN SLC CARD
Since the SLC card indicated contained no data, it was ignored.

RL15I xxxxxx SLC HAS SET LOC. CNTR. TO VALUE ALREADY LOADED
The updated value of the location counter is smaller than the previous one, therefore all or part of the control section previously loaded may be overlaid.

RL16I xxxxxx CHARACTER IN CARD NOT HEXADECIMAL
The card indicated, which may be a REP, an SLC, or an ICS card, contains a non-hexadecimal character in a field which must be hexadecimal. The card was ignored.

RL17I xxxxxx ENTRY POINT IS REPEATED
The symbol contained in the card indicated has already been defined as an entry point. The card was ignored.

RL18I xxxxxx ENTRY POINT NOT RELOCATABLE
The address of the entry point on the card indicated cannot be relocated within the limits of the control section to which it belongs. This can arise only if the card was punched by hand and a mistake was made in calculating or punching the address. The loader relocates the entry point at address 0.

RL19I xxxxxx ADDRESS NOT RELOCATABLE
This message applies only to RLD cards and means that the address of a constant in the card cannot be relocated within the limits of the control section to which it belongs, or that the control section has already been loaded. The first case is due to an error in hand-punching a card, the second indicates that the control section has been repeated by error on the tape or in the card deck.

RL20I xxxxxx EOF BEFORE END OF LOADING
No LDT card was found at the end of the file. The loader assumes that there was one but that it did not contain a transfer address, and ends loading accordingly.

Error Messages

RL21W xxxxxx INSUFFICIENT SPACE FOR LOADER TABLES
The area of main storage reserved for the loader tables is too small. The length of this area was specified in the card preceding the first module or control section to be loaded. This message occurs only if the user has modified the Simulator system delivered by IBM.

RL22W xxxxxx INSUFFICIENT SPACE AVAILABLE FOR THE PROGRAM
This message may be due to an SLC card with too high an address, or it may occur if the space for the loader tables has been increased by the user to such an extent that there is not enough storage left to accommodate the program. In the latter case, if the loader table size cannot be reduced, RELLDR will have to be moved further on in storage.

RL23W xxxxxx PROGRAM ERROR
This message occurs only if part of the support programs (CONTPR or IOPACK) has been accidentally overlaid during loading, for instance as a result of an erroneous SLC card. The card which was being loaded at the time the program error occurred is identified in the message.

Chart FA. Overall Logic of ABSLOD

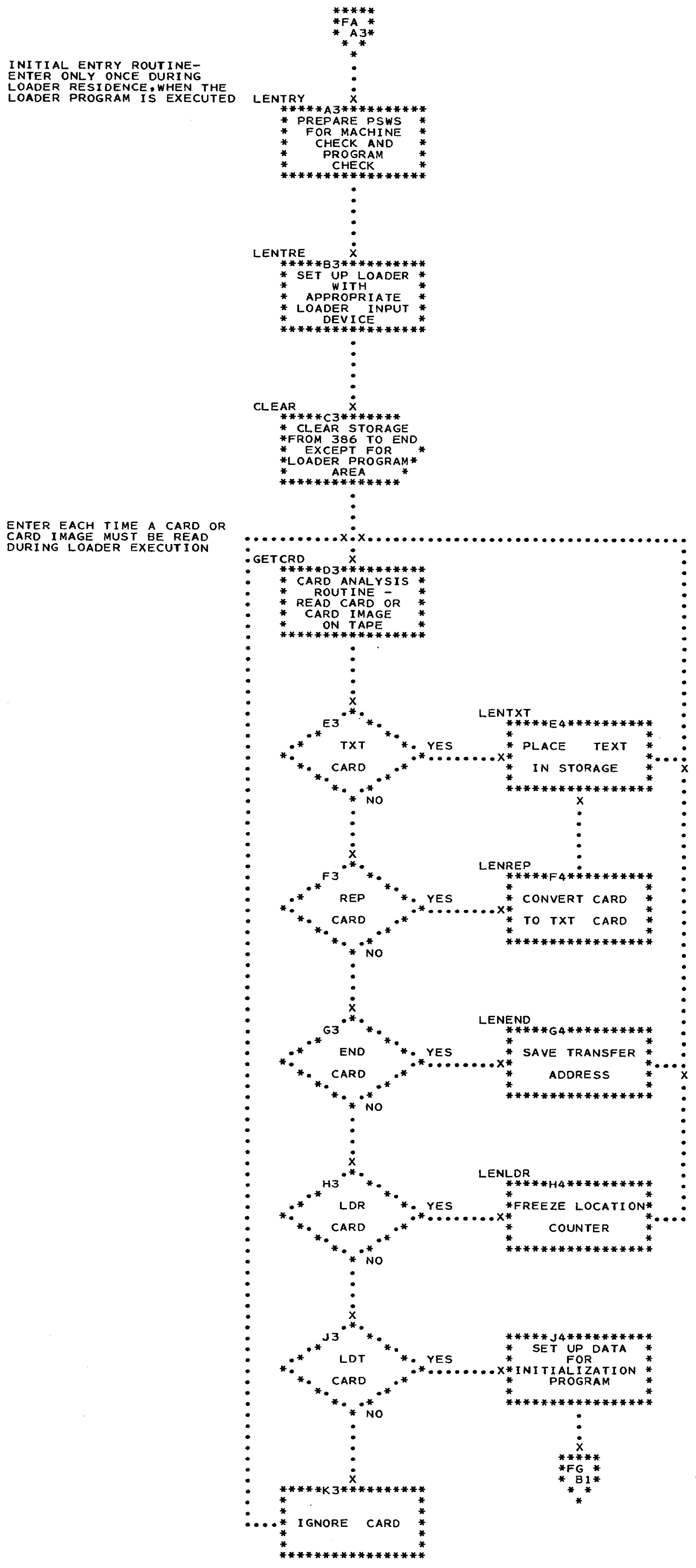

FA A3
INITIAL ENTRY ROUTINE-ENTER ONLY ONCE DURING LOADER RESIDENCE,WHEN THE LOADER PROGRAM IS EXECUTED
LENTRY
A3 PREPARE PSWS FOR MACHINE CHECK AND PROGRAM CHECK
LENTRE
B3 SET UP LOADER WITH APPROPRIATE LOADER INPUT DEVICE
CLEAR
C3 CLEAR STORAGE FROM 386 TO END EXCEPT FOR LOADER PROGRAM AREA
ENTER EACH TIME A CARD OR CARD IMAGE MUST BE READ DURING LOADER EXECUTION
GETCRD
D3 CARD ANALYSIS ROUTINE - READ CARD OR CARD IMAGE ON TAPE
E3 TXT CARD
YES
NO
LENTXT
E4 PLACE TEXT IN STORAGE
F3 REP CARD
YES
NO
LENREP
F4 CONVERT CARD TO TXT CARD
G3 END CARD
YES
NO
LENEND
G4 SAVE TRANSFER ADDRESS
H3 LDR CARD
YES
NO
LENLDR
H4 FREEZE LOCATION COUNTER
J3 LDT CARD
YES
NO
J4 SET UP DATA FOR INITIALIZATION PROGRAM
FG B1
K3 IGNORE CARD

Chart FB. Overall Logic of CONTPR

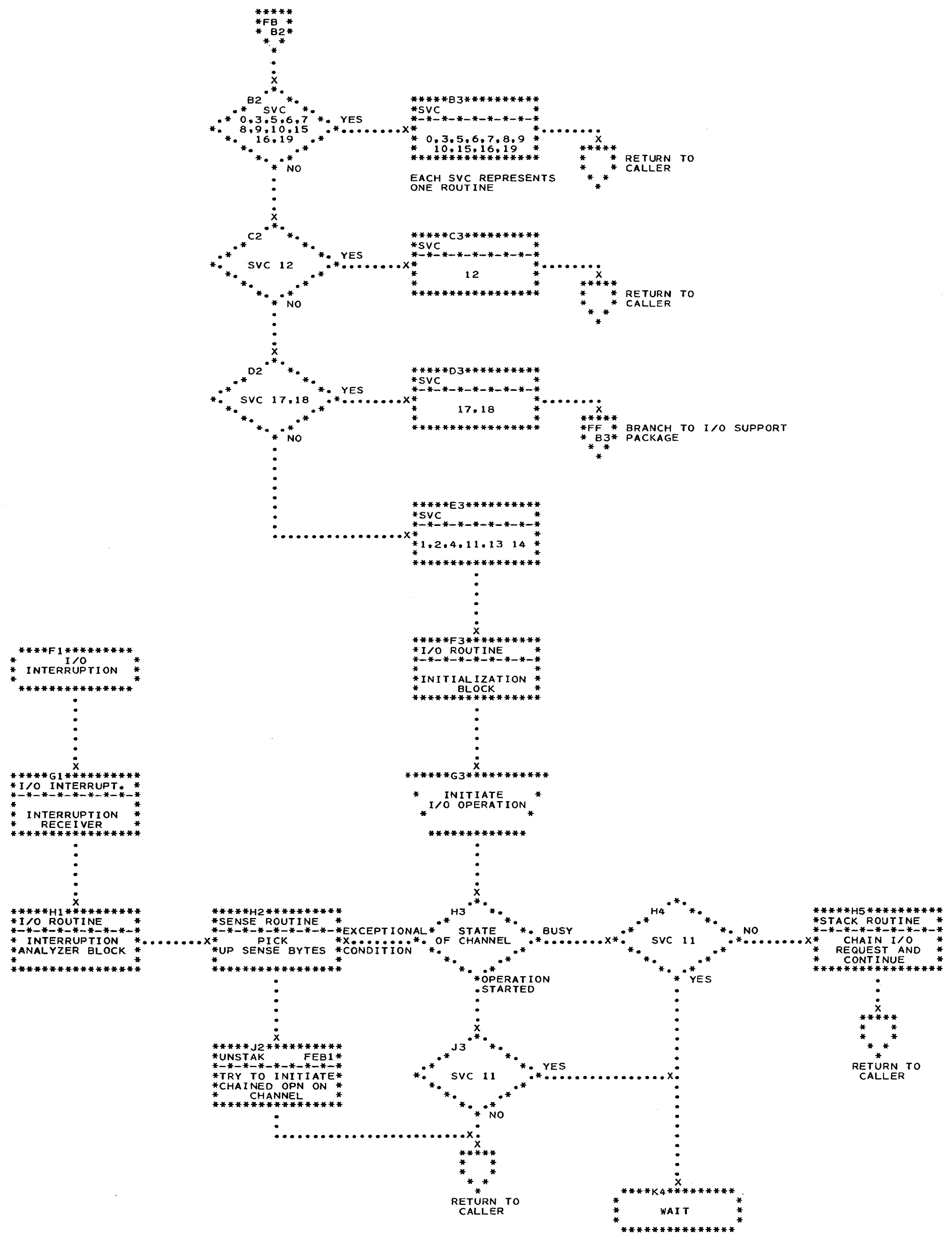
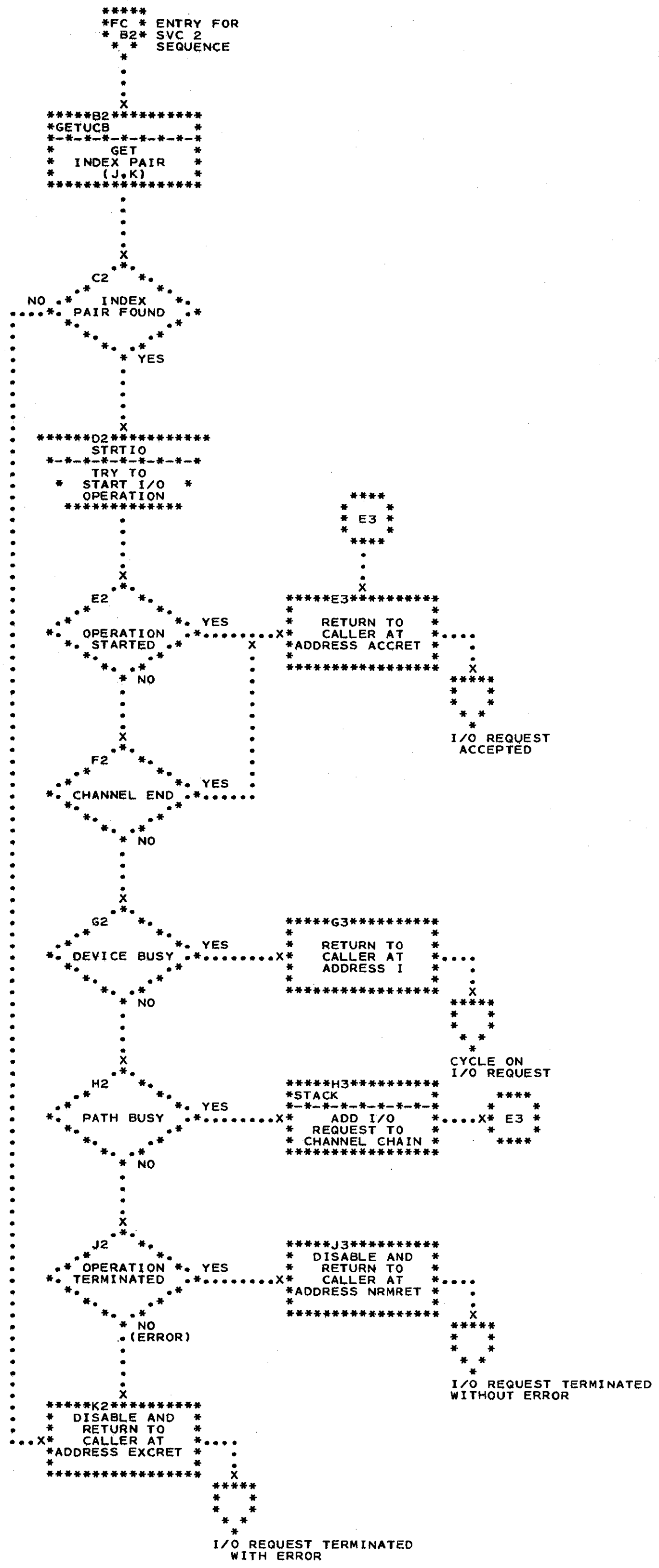

FB B2
B2 SVC 0,3,5,6,7 8,9,10,15 16,19
YES
NO
B3 SVC 0,3,5,6,7,8,9 10,15,16,19
EACH SVC REPRESENTS ONE ROUTINE
RETURN TO CALLER
C2 SVC 12
YES
NO
C3 SVC 12
RETURN TO CALLER
D2 SVC 17,18
YES
NO
D3 SVC 17,18
FF B3 BRANCH TO I/O SUPPORT PACKAGE
E3 SVC 1,2,4,11,13 14
F3 I/O ROUTINE INITIALIZATION BLOCK
F1 I/O INTERRUPTION
G1 I/O INTERRUPT. INTERRUPTION RECEIVER
G3 INITIATE I/O OPERATION
H1 I/O ROUTINE INTERRUPTION ANALYZER BLOCK
H2 SENSE ROUTINE PICK UP SENSE BYTES
EXCEPTIONAL CONDITION
H3 STATE OF CHANNEL
BUSY
OPERATION STARTED
H4 SVC 11
NO
YES
H5 STACK ROUTINE CHAIN I/O REQUEST AND CONTINUE
RETURN TO CALLER
J2 UNSTAK FEB1 TRY TO INITIATE CHAINED OPN ON CHANNEL
J3 SVC 11
YES
NO
RETURN TO CALLER
K4 WAIT

Chart FC. I/O Request and Continue (Part 1)



FC B2 ENTRY FOR SVC 2 SEQUENCE
B2 GETUCB GET INDEX PAIR (J,K)
C2 INDEX PAIR FOUND
NO
YES
D2 STRTIO TRY TO START I/O OPERATION
E3
E2 OPERATION STARTED
YES
NO
E3 RETURN TO CALLER AT ADDRESS ACCRET
I/O REQUEST ACCEPTED
F2 CHANNEL END
YES
NO
G2 DEVICE BUSY
YES
NO
G3 RETURN TO CALLER AT ADDRESS I
CYCLE ON I/O REQUEST
H2 PATH BUSY
YES
NO
H3 STACK ADD I/O REQUEST TO CHANNEL CHAIN
E3
J2 OPERATION TERMINATED
YES
NO (ERROR)
J3 DISABLE AND RETURN TO CALLER AT ADDRESS NRMRET
I/O REQUEST TERMINATED WITHOUT ERROR
K2 DISABLE AND RETURN TO CALLER AT ADDRESS EXCRET
I/O REQUEST TERMINATED WITH ERROR

Chart FD. I/O Request and Continue (Part 2)

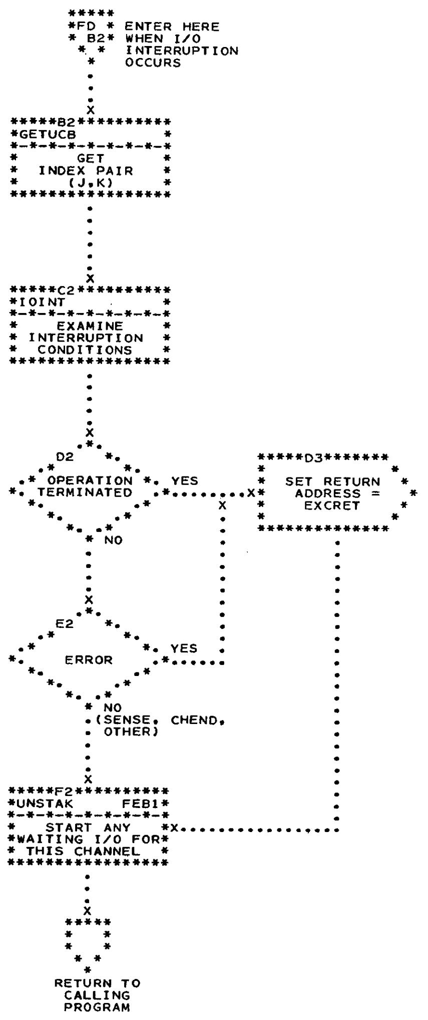

FD B2
ENTER HERE WHEN I/O INTERRUPTION OCCURS
B2
GETUCB
GET INDEX PAIR (J,K)
C2
IOINT
EXAMINE INTERRUPTION CONDITIONS
D2
OPERATION TERMINATED
YES
NO
D3
SET RETURN ADDRESS = EXCRET
E2
ERROR
YES
NO (SENSE, CHEND, OTHER)
F2
UNSTAK FEB1
START ANY WAITING I/O FOR THIS CHANNEL
RETURN TO CALLING PROGRAM

Chart FE. UNSTAK Routine

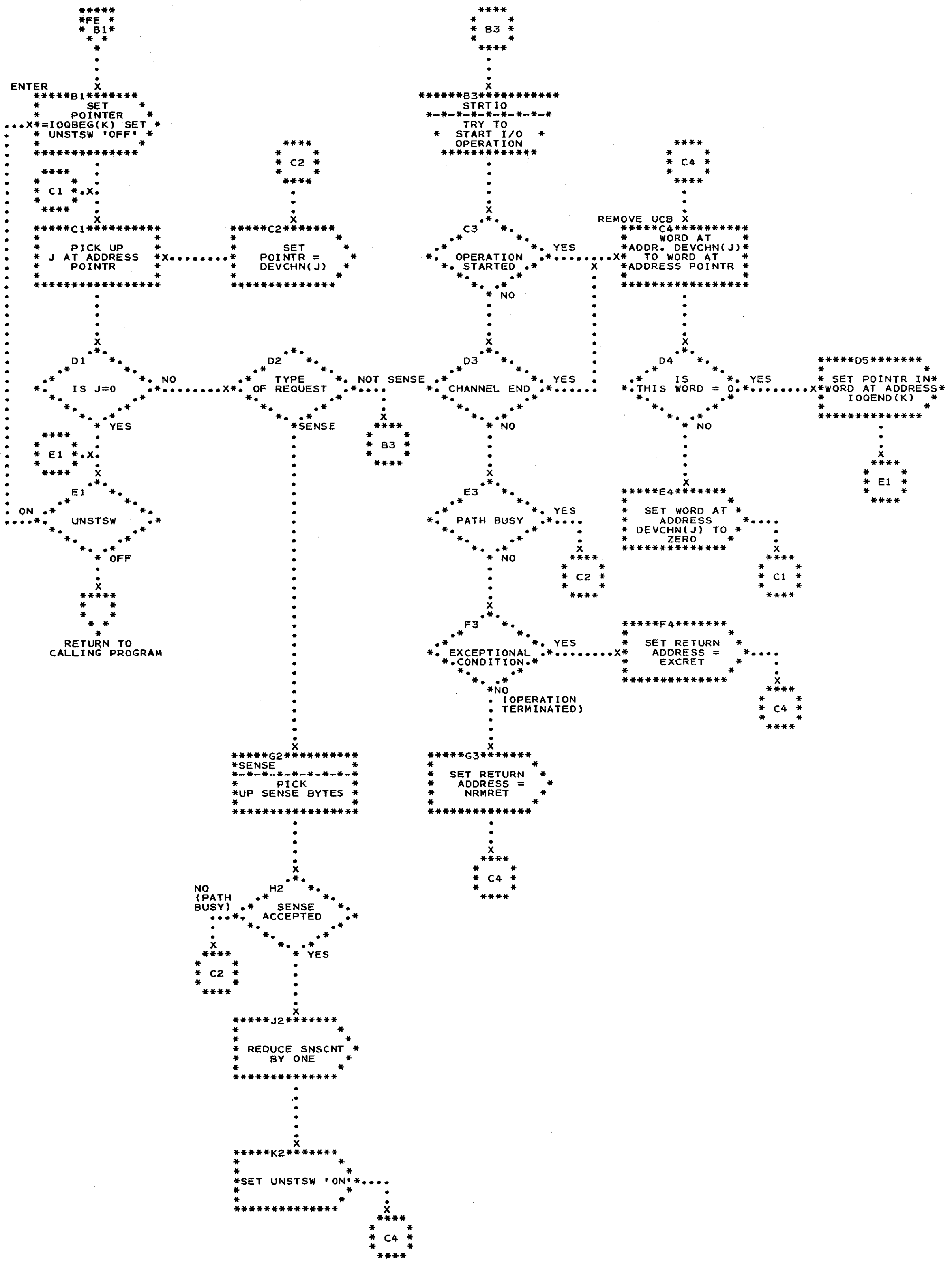

FE B1
ENTER
B1 SET POINTER =IOQBEG(K) SET UNSTSW 'OFF'
C1
C1 PICK UP J AT ADDRESS POINTR
D1 IS J=0
NO
YES
E1
E1 UNSTSW
ON
OFF
RETURN TO CALLING PROGRAM
C2
C2 SET POINTR = DEVCHN(J)
D2 TYPE OF REQUEST
NOT SENSE
SENSE
B3
G2 SENSE PICK UP SENSE BYTES
H2 SENSE ACCEPTED
NO (PATH BUSY)
C2
YES
J2 REDUCE SNSCNT BY ONE
K2 SET UNSTSW 'ON'
C4
B3
B3 STRTIO TRY TO START I/O OPERATION
C3 OPERATION STARTED
YES
NO
D3 CHANNEL END
YES
NO
E3 PATH BUSY
YES
C2
NO
F3 EXCEPTIONAL CONDITION
YES
NO (OPERATION TERMINATED)
G3 SET RETURN ADDRESS = NRMRET
C4
F4 SET RETURN ADDRESS = EXCRET
C4
C4
REMOVE UCB
C4 WORD AT ADDR. DEVCHN(J) TO WORD AT ADDRESS POINTR
D4 IS THIS WORD = 0.
YES
NO
E4 SET WORD AT ADDRESS DEVCHN(J) TO ZERO
C1
D5 SET POINTR IN WORD AT ADDRESS IOQEND(K)
E1

Chart FF. Overall Logic of IOPACK

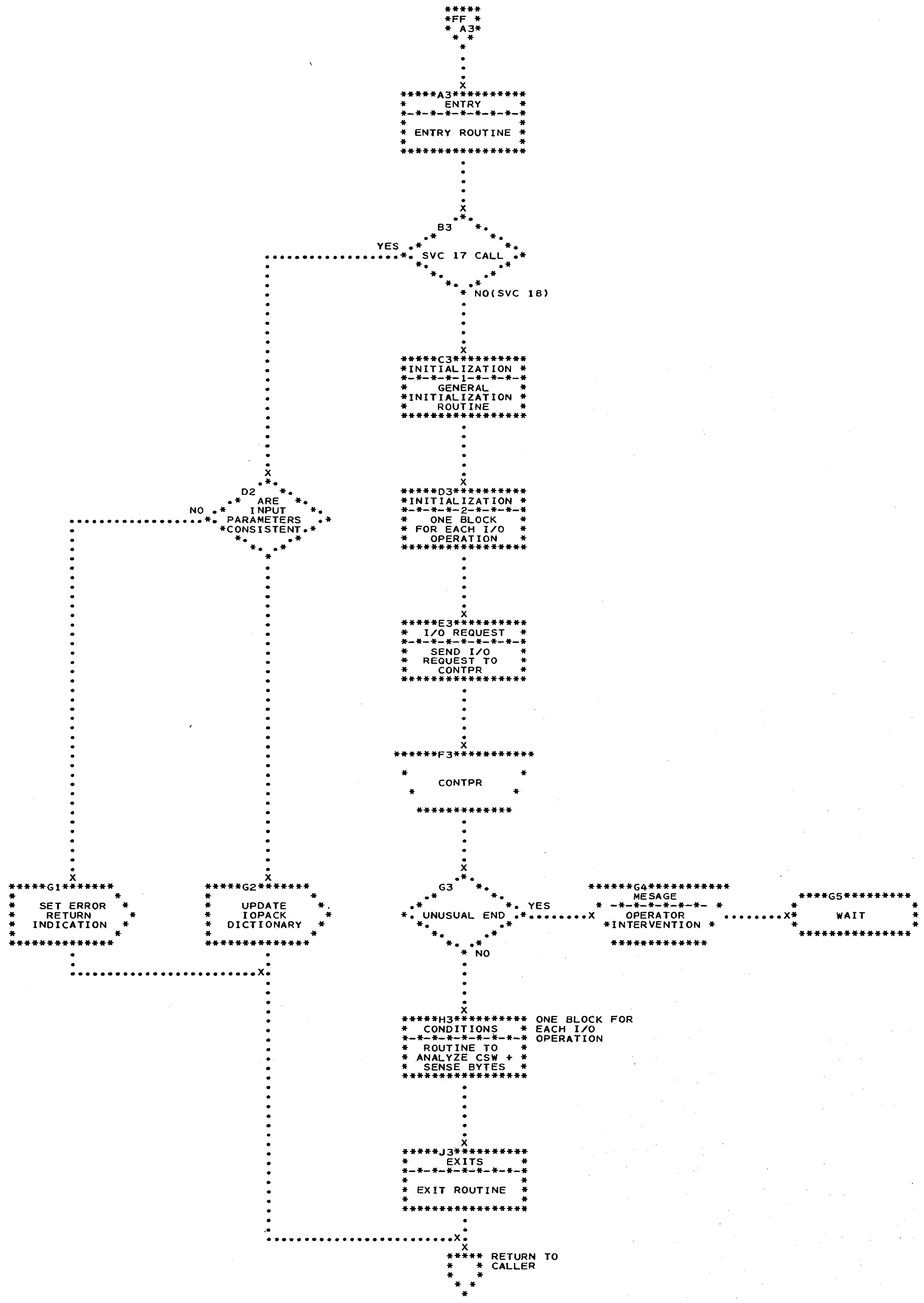

FF A3
A3 ENTRY
ENTRY ROUTINE
B3 SVC 17 CALL
YES
NO(SVC 18)
C3 INITIALIZATION 1
GENERAL INITIALIZATION ROUTINE
D2 ARE INPUT PARAMETERS CONSISTENT
NO
D3 INITIALIZATION 2
ONE BLOCK FOR EACH I/O OPERATION
E3 I/O REQUEST
SEND I/O REQUEST TO CONTPR
F3 CONTPR
G1 SET ERROR RETURN INDICATION
G2 UPDATE IOPACK DICTIONARY
G3 UNUSUAL END
YES
NO
G4 MESAGE
OPERATOR INTERVENTION
G5 WAIT
H3 CONDITIONS
ROUTINE TO ANALYZE CSW + SENSE BYTES
ONE BLOCK FOR EACH I/O OPERATION
J3 EXITS
EXIT ROUTINE
RETURN TO CALLER

Chart FG. Overall Logic of INIT

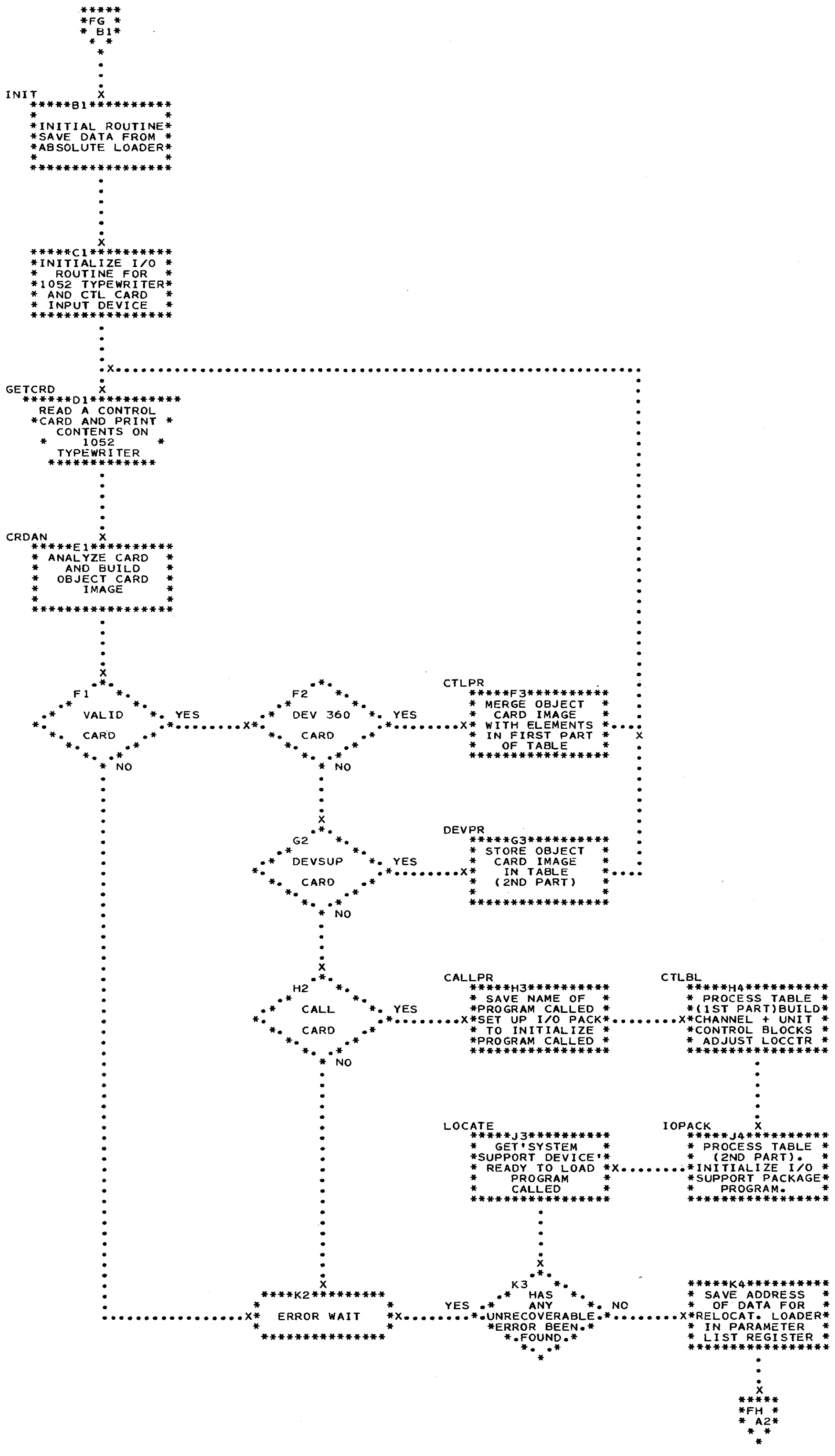
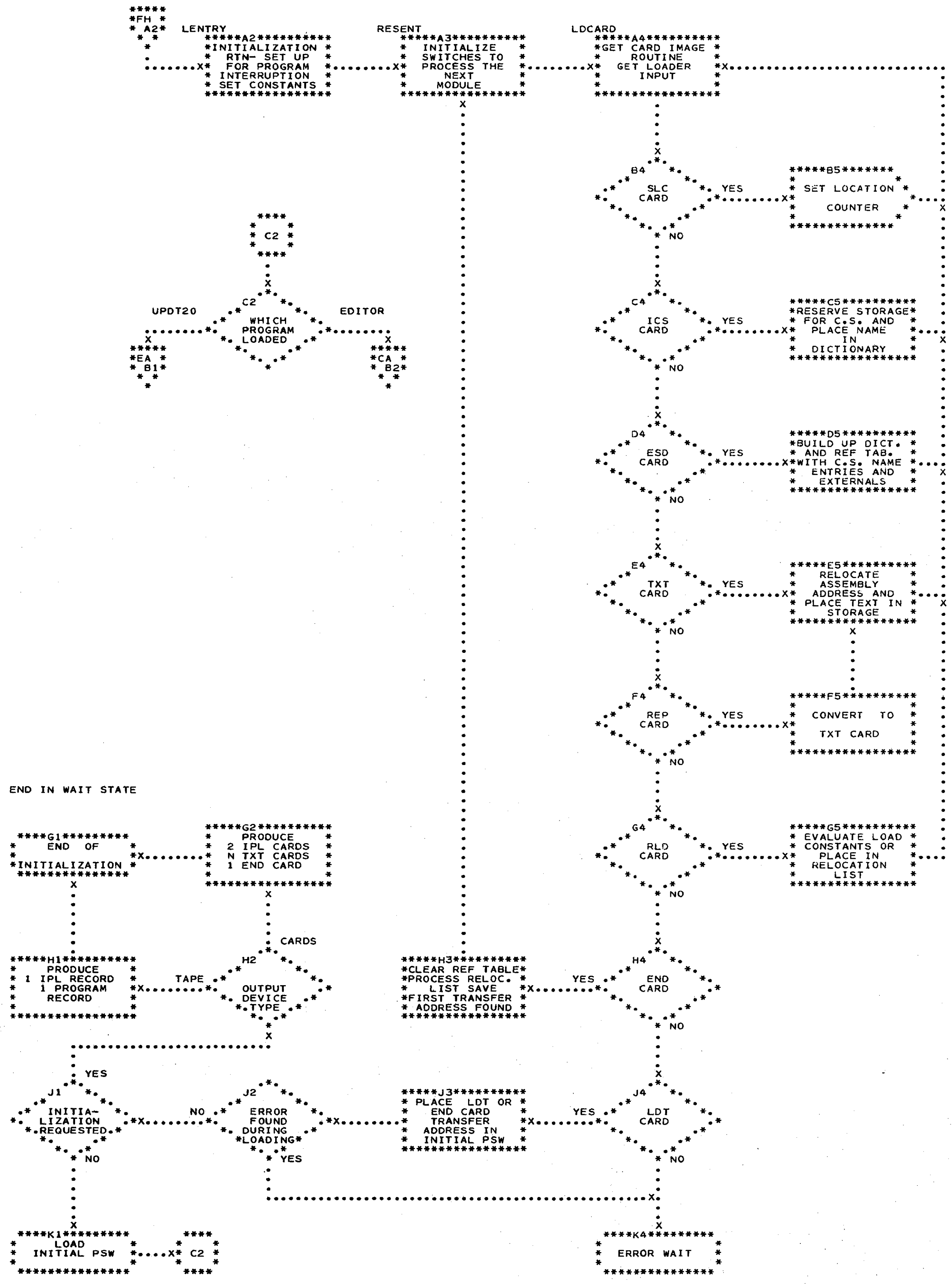

FG B1
INIT B1 INITIAL ROUTINE SAVE DATA FROM ABSOLUTE LOADER
C1 INITIALIZE I/O ROUTINE FOR 1052 TYPEWRITER AND CTL CARD INPUT DEVICE
GETCRD D1 READ A CONTROL CARD AND PRINT CONTENTS ON 1052 TYPEWRITER
CRDAN E1 ANALYZE CARD AND BUILD OBJECT CARD IMAGE
F1 VALID CARD
YES
NO
F2 DEV 360 CARD
YES
NO
CTLPR F3 MERGE OBJECT CARD IMAGE WITH ELEMENTS IN FIRST PART OF TABLE
G2 DEVSUP CARD
YES
NO
DEVPR G3 STORE OBJECT CARD IMAGE IN TABLE (2ND PART)
H2 CALL CARD
YES
NO
CALLPR H3 SAVE NAME OF PROGRAM CALLED SET UP I/O PACK TO INITIALIZE PROGRAM CALLED
CTLBL H4 PROCESS TABLE (1ST PART)BUILD CHANNEL + UNIT CONTROL BLOCKS ADJUST LOCCTR
LOCATE J3 GET'SYSTEM SUPPORT DEVICE' READY TO LOAD PROGRAM CALLED
IOPACK J4 PROCESS TABLE (2ND PART). INITIALIZE I/O SUPPORT PACKAGE PROGRAM.
K2 ERROR WAIT
K3 HAS ANY UNRECOVERABLE ERROR BEEN FOUND
YES
NO
K4 SAVE ADDRESS OF DATA FOR RELOCAT. LOADER IN PARAMETER LIST REGISTER
FH A2

Chart FH. Overall Logic of RELLDR


FH A2
LENTRY
A2 INITIALIZATION RTN- SET UP FOR PROGRAM INTERRUPTION SET CONSTANTS
RESENT
A3 INITIALIZE SWITCHES TO PROCESS THE NEXT MODULE
LDCARD
A4 GET CARD IMAGE ROUTINE GET LOADER INPUT
B4 SLC CARD
YES
B5 SET LOCATION COUNTER
NO
C2
C2 WHICH PROGRAM LOADED
UPDT20
EDITOR
EA B1
CA B2
C4 ICS CARD
C5 RESERVE STORAGE FOR C.S. AND PLACE NAME IN DICTIONARY
D4 ESD CARD
D5 BUILD UP DICT. AND REF TAB. WITH C.S. NAME ENTRIES AND EXTERNALS
E4 TXT CARD
E5 RELOCATE ASSEMBLY ADDRESS AND PLACE TEXT IN STORAGE
F4 REP CARD
F5 CONVERT TO TXT CARD
END IN WAIT STATE
G1 END OF INITIALIZATION
G2 PRODUCE 2 IPL CARDS N TXT CARDS 1 END CARD
G4 RLD CARD
G5 EVALUATE LOAD CONSTANTS OR PLACE IN RELOCATION LIST
CARDS
TAPE
H1 PRODUCE 1 IPL RECORD 1 PROGRAM RECORD
H2 OUTPUT DEVICE TYPE
H3 CLEAR REF TABLE PROCESS RELOC. LIST SAVE FIRST TRANSFER ADDRESS FOUND
H4 END CARD
J1 INITIA-LIZATION REQUESTED
J2 ERROR FOUND DURING LOADING
J3 PLACE LDT OR END CARD TRANSFER ADDRESS IN INITIAL PSW
J4 LDT CARD
K1 LOAD INITIAL PSW
C2
K4 ERROR WAIT

APPENDIX. LIST OF SIM20 ROUTINES

The following list contains the symbolic names and functions of all major SIM20 routines.

Basic and Console Simulation Routines

ALARM	Issues alarm commands on the 1052 Printer-Keyboard
BIR	Basic Interpretive Routine: decodes the operation codes of 1620 instructions
EXCRET	Selects exceptional returns from all I/O requests
MASK	Builds up all code conversion tables required by I/O operations
MESSAG	Processes all output messages on the 1052 Printer-Keyboard
OUTIN	Selects the sequence corresponding to any 1620 I/O instruction and, in the case of the disk-resident version, checks its presence in core storage
TYPIO	Processes all physical I/O requests
VALIN	Performs code conversion and validity checking in all input operations

CPU Simulation Routines

ARCHK	Processes arithmetic overflow and underflow
COMP	Compares P and Q fields in 1620 add and subtract and floating add and subtract operations
CONVP	Converts the P address only
CONVPQ	Converts both P and Q addresses
CONVQ	Converts the Q address only
EXCHK	Processes exponent overflow and underflow
EXPOW	Checks for exponent overflow and underflow
FIXADD	Adds or subtracts P and Q fields in 1620 add and subtract and floating add and subtract operations
FIXDIV	Divides P and Q fields in 1620 divide and floating divide operations
INDAD	Processes indirect addresses
INDEX	Processes address indexing (1620 Model 2)
INDIC	Updates HP/EZ indicators after all arithmetic operations
MULT	Multiplies P and Q fields in 1620 multiply and floating multiply operations
SHIFT	Shifts P or Q fields, when the P and Q exponents are different, in 1620 floating add and subtract operations

I/O Simulation Routines

GETEOR	Scans the P field to detect a record mark in output operations
VALOUT	Performs code conversion and validity checking in all output operations

Disk Simulation Routines

CONVCW	Converts the 1620 disk control field into data consistent with System/360 commands
DCFADD	Checks the sequence of sector addresses and updates the counter
DISKER	Handles all exceptional returns from I/O request and wait in read and check operations
DISKEW	Handles all exceptional returns from I/O request and wait in write operations
DISRMH	Prepares data for the MATCH subroutine
ENDISK	Detects the end of a disk operation by checking the sector count against zero
IORW	Processes all physical read and write operations on disk

MATCH	Checks all sector addresses submitted against the contents of the 21st sector
READ21	Reads the 21st sector in core storage for the MATCH subroutine
TESTGM	Scans data to detect group marks
TRAKED	Performs all checks and counts in track mode operations
WLRCSB	Tests for a group mark after the last sector in operations with WLRC

ABSLOD Routines (Module A21)

GETCRD	Analyzes loader cards
IPLCTL	Loads ABSLOD into System/360 main storage
LDREAD	Reads from tape or cards
LENEND	Processes END cards
LENLDR	Processes LDR cards
LENLDT	Processes LDT cards
LENREP	Processes REP cards
LENTRY	Initialization routine
LENTXT	Processes TXT cards
LHEXB1	Converts hexadecimal to binary

CONTPR Routines (Module A22)

COMAND	Reads commands from the 1052 Printer-Keyboard
INTUCB	Determines the device index J of a unit control block and the channel index K of a channel control block, given the device address
IOCONT	Processes I/O request and continue calling sequences
IOINT	Processes I/O interruptions
IOWAIT	Processes I/O request and wait calling sequences
MESAGE	Transmits messages to the 1052 Printer-Keyboard
SENSE	Performs sense operations
SEREP	Sets up the standard SEREP interface for I/O failures
STACK	Chains I/O request and continue calling sequences
STRTIO	Performs physical I/O operations
SVCINT	Processes SVC interruptions
UNSTAK	Initiates as many I/O operations as possible on a designated channel chain
VERIFY	Verifies the type and characteristics of a device

IOPACK Routines (Module A23)

CALLA	Processes I/O request calling sequences
CONSLE	Transmits messages to the 1052 Printer-Keyboard
CVRTM	Converts binary to hexadecimal
INFACT	Calls the write routine for IOPACK messages
IOPACK	I/O support package entry routine
NRMRET	Sets up exits for I/O request calling sequences
PN1442/ PN2540	Punches cards
PRNT	Printer output
RD1442/ RD2540	Reads cards
STRTIO	Initializes I/O request calling sequences
SWEXT	Processes input commands
TP70P/ TAPERD/ TAPEWR	Reads or writes magnetic tapes
TYPRD	Reads operator commands
TYPWRT	Writes operator messages

INIT Routines (Module A24)

CALLPR	Processes CALL cards
CRDAN	Analyzes control cards
CTLBL	Builds channel and unit control blocks
CTLPR/ DEVPR	Stores control information table

GETCRD	Reads control cards
INIT	Initialization routine
IOPACK	Creates SYNTAB in the I/O support package
LOCATE	Exit routine
LOCERR	Prints error messages
MSDG	Writes messages

RELLDR Routines (Module A25)

LDCARD	Reads card image
LDEDIT	Self-Loading Program Generator routine
LDEND	Processes END cards
LDESD	Processes ESD cards
LDICS	Processes ICS cards
LDLDT	Processes LDT cards
LDMSDG/ LPRINT	Prints loading messages
LDREP	Processes REP cards
LDRIN	Loader initialization routine
LDRLD	Processes RLD cards
LDSLC	Processes SLC cards
LDSWT	Analyzes loader cards
LDTXT	Processes TXT cards

Where more than one reference is given, the first page number indicates the major reference.

READER'S COMMENTS

IBM System/360 Conversion Aids: The 1620 Simulator
for IBM System/360; Program Number 360C-SI-752

Y27-7116-1

Your comments will help us to produce better publications for your use. Please check or fill in the items below and add explanations and other comments in the space provided.

Which of the following terms best describes your job?

¤ Programmer	¤ Systems Analyst	¤ Customer Engineer
¤ Manager	¤ Engineer	¤ Systems Engineer
¤ Operator	¤ Mathematician	¤ Sales Representative
¤ Instructor	¤ Student/Trainee	¤ Other (explain) ________________

Does your installation subscribe to the SRL Revision Service? ¤ Yes ¤ No

How did you use this publication? ____

¤ As an introduction
¤ As a reference manual
¤ As a text (student)
¤ As a text (instructor)
¤ For another purpose (explain) ____

Did you find the material easy to read and understand? ¤ Yes ¤ No (explain below)

Did you find the material organized for convenient use? ¤ Yes ¤ No (explain below)

Specific Criticisms (explain below)

Clarifications on pages
Additions on pages
Deletions on pages
Errors on pages

Explanations and Other Comments

Y27-7116-1

FOLD FOLD

PAR AVION

AFFIX POSTAGE

IBM WORLD TRADE LAB
CENTRE D'ETUDES ET RECHERCHES
06-LA GAUDE (ALPES-MARITIMES)
FRANCE

ATTN: PROGRAMMING PUBLICATIONS
DEPARTMENT 841

FOLD FOLD

Printed in U.S.A. Y27-7116-1

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Printed in U.S.A. Y27-7116-1

IBM Technical Newsletter

File Number	S360-35
Re: Form No.	Y27-7116-1
This Newsletter No.	Y33-7002
Date	April 22, 1967
Previous Newsletter Nos.	None

IBM SYSTEM/360 CONVERSION AIDS: THE 1620 SIMULATOR FOR IBM SYSTEM/360 PROGRAM LOGIC MANUAL

This technical newsletter amends the publication IBM System/360 Conversion Aids: The 1620 Simulator, Program Logic Manual, Form Y27-7116-1. The attached pages replace pages in the publication. Corrections and additions to the text are noted by vertical bars to the left of the change. A dot (•) next to a page number indicates that the entire page should be reviewed.

Pages to be Inserted	Pages to be Removed
1 and 2	1 and 2
45, 46, 46.1 through 46.4	45 and 46

Summary of Amendments

Section UPDT20, page 46, has been modified. The layout of the system tape is described, and the identification of its components is illustrated in Table 5A.

The chapter "Corrections" has been replaced by a description of the updating functions, correction card formats, and examples of updating.

On page 47, under "SKCRDA Subroutine," the words "MODIF cards" should be changed to "modification cards."

Note: Please file this cover letter at the back of the publication. Cover letters provide a quick reference to changes, and a means of checking receipt of all amendments.

RESTRICTED DISTRIBUTION

IBM France, Centre d'Etudes et Recherches, Programming Publications, Department 841, 06 La Gaude, France

PRINTED IN U.S.A.