PROGRAMMING GUIDE FOR THE MODEL 960 COMPUTER

VOLUME I
OPERATIONAL SOFTWARE

TEXAS INSTRUMENTS
INCORPORATED
DIGITAL SYSTEMS DIVISION
P.O. BOX 66027 HOUSTON, TEXAS 77006

PRINTED IN U.S.A.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

SECTION I

GENERAL INFORMATION

1-1 SCOPE OF PROGRAMMING GUIDE.

The Texas Instruments Programming Guide for the Model 960 Process Control Computer is divided into two volumes:

a. Volume I contains the operational software.

b. Volume II contains the performance assurance test descriptions.

Revisions and additions will be published whenever they are justified. Users who are on the authorized distribution list will receive published revisions and additions automatically.

The Programming Guide is broken into several sections according to subject. Refer to the Table of Contents to quickly locate the desired subject.

The user is given a general orientation to the Model 960 Process Control Computer in this section. Details of programming may be located also in the Model 960 Computer Programmer's Reference Manual. Detail information about the computer itself may be found in the Model 960 Computer Maintenance Manual.

1-2 COMPUTER CHARACTERISTICS.

The computer is a versatile tool for process automation which readily adapts to a wide variety of applications. These applications may be discrete or continuous operations. Typical applications include tool operation, fabrication and automatic assembly, material handling, environmental control, and data acquisition. The computer can perform inspections and issue status reports. Pertinent data can be displayed to an operator at the job site or relayed to a central computer facility to provide accurate data for management decisions.

Perhaps the most worthy characteristic of the computer is its real-time process control capabilities. Real-time process control requires a computer with fast efficient context switching, easy manipulation of bits and bit-fields, and easy exchange of data between the computer and external devices. The computer solves a great many automation problems with the following features:

a. Dual Mode Operation. The dual-mode feature permits fast context switching. While running in one mode with one set of registers and execution counter, control can be switched to a second mode with identical capabilities. This not only provides a new programming environment, but frequently avoids the need to save the status of the old environment. Mode switching can be accomplished under interrupt or programmed instruction control.

b. Real-Time Clocks. Many process control functions are time critical. The computer's optional interval timers perform many tight, time critical functions. These optional timers are desirable where process functions must occur at specific instants or must occur after a specific time delay.

c. Data Input/Output. The Communication Register Unit (CRU) provides a changeable, efficient interface with controlled external devices. Four CRU modules, each with 16 input and 16 output lines, can be installed inside the Central Processing Unit (CPU) enclosure. Additional CRU modules may be added to expand the CRU capacity to 256 modules in a separate enclosure. Total capability would be, therefore, 4096 input and 4096 output lines. A variety of CRU functions are available, such as binary data modules, analog-to-digital modules, digital-to-analog modules, stepping motor controller modules, and pulse generator modules.

The computer block diagram (Figure 1-1) shows the basic internal function relationships.

a. The core memory can store from 4096 to 65,536 17-bit words (16 data bits plus 1 parity bit).

b. The Central Processing Unit (CPU) can address the core memory, perform arithmetic and logic functions, and sequence and control the exchange of information between the core memory and other elements of the computer. The CPU features an arithmetic unit and a read-only memory controller.

c. The Communication Register Unit (CRU) controls the exchange of information between the computer and external operations.

d. The Direct Memory Access Channel (DMAC) interfaces the computer with high-speed automatic computer peripherals, such as disc storage units, line printers, and magnetic tape units. By using a separate controller for each device, concurrent operation of high speed peripherals is achieved.

Core memory built in modules of 4096 words each

Minimum storage capacity, 4096 words

Maximum storage capacity, 65,536 words

Data word length, 16 information bits plus 1 parity bit

Cycle time, 980 nanoseconds

Instruction word length, 32 bits

16 virtual registers for arithmetic, index, or mask operations

Execution time

Load: 5.0 microseconds

Store: 5.3 microseconds

Add: 6.0 microseconds

Set CRU bit: 4.6 microseconds

Load register in CRU: 7.0 to 12.0 microseconds (1-16 bits)

Memory protect system for variable amounts of memory

Single and double address logic

Direct addressing of entire core memory, by word or bit

Indirect addressing with pre-indexing or post-indexing

Displacement index registers

3 MegaHertz clock

Three levels of priority interrupt

Input/Output System

Communication Register Unit with 3 million bits per second burst rate, output; 1.5 million bits per second burst rate, input

Direct Memory Access Channel with 1 million words per second burst rate, input or output

1-3 SYSTEM CONFIGURATIONS.

The Model 960 Process Control Computer may be augmented with any or all of several peripheral equipments.

a. Electronic Data Terminal or Teletypewriter

b. Card Reader

c. Card Punch

d. High Speed Tape Reader

e. High Speed Tape Punch

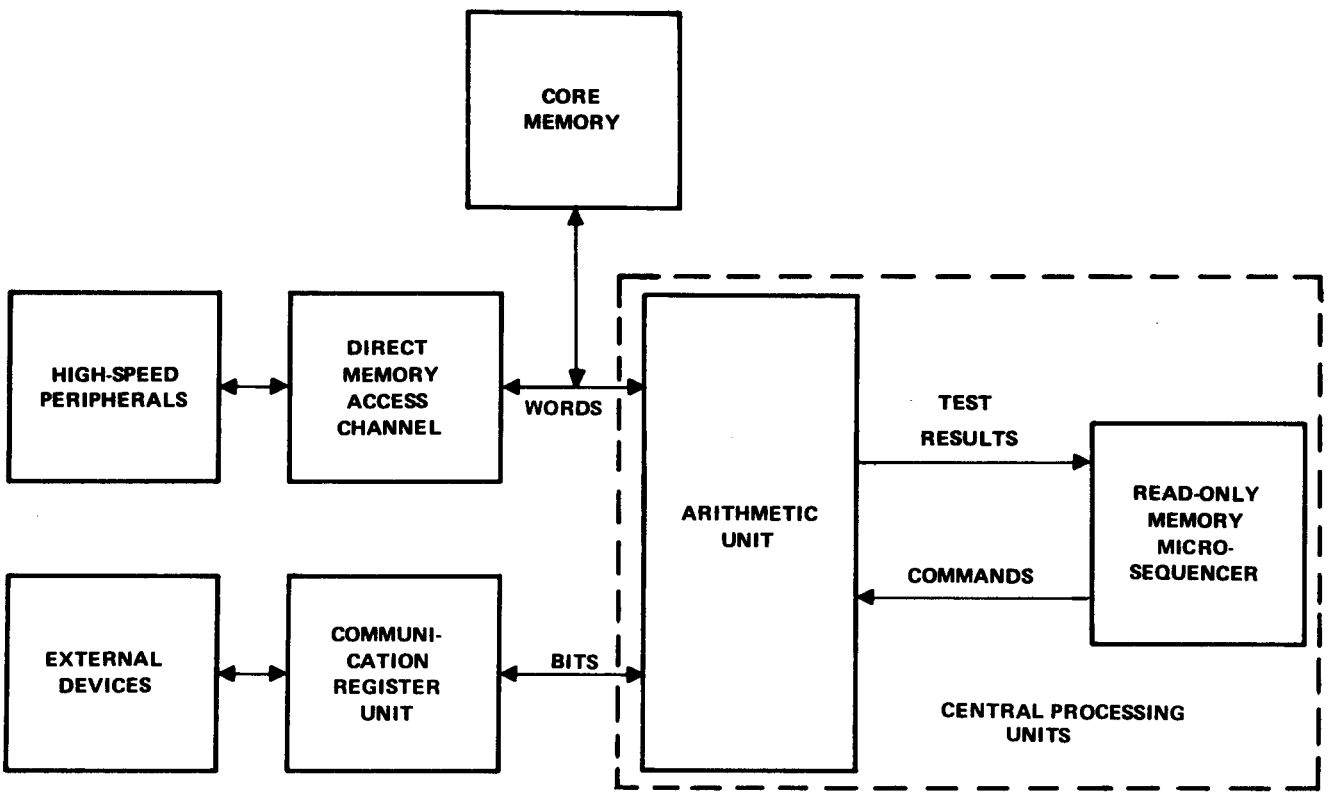


Figure 1-1 TI 960 Computer Block Diagram.

f. Magnetic Tape Transport

g. Storage Disc.

The software assumes that the computer will be equipped with an electronic data terminal with its associated tape reader and punch. This is the standard input/output device. Texas Instruments Silent 700 Electronic Data Terminal may be replaced with a Teletype Corporation Model ASR-33 TBE teletypewriter. The units are interchangeable as far as the software is concerned.

The computer's modular core memory has a capacity of 4096 words, commonly called a ". . . 4K module." The CPU has physical capacity for two core memory modules (8192 words). However, the system is available with any number of core memory modules, up to 65,536 words, maximum. The larger capacity is obtained by housing the additional core memory modules in a second cabinet. Standard cables and hardware fittings simplify assembly of computer systems to meet the user's needs.

Texas Instruments has anticipated that the user may desire the computer system to be arranged in more than one way. Therefore, the Model 960 Computer may be mounted into a table console or in traditional racks. Depending upon the application, the computer and its related peripheral equipments can be customized to a functional and pleasing arrangement for the user.

1-4 SOFTWARE CHARACTERISTICS.

The computer can be operated under control of two monitors or selected stand-alone programs. Two well developed monitors are initially available: Process Automation Monitor (PAM960) and Programming Support Monitor (PSM960).

The Process Automation Monitor is a real-time, multi-programming operating system. It uses an executive/worker method for program control and core resident worker tasks.

The Programming Support Monitor is a non-real-time, single-programming operating system. It uses a supervisor with interrupts for input/output devices. The I/O device service routines reside in core. Standard device service routines may be used or replaced by the user to perform more specific tasks. PSM is upwards compatible with PAM. Programs which are written to execute using PSM as the computer interface will also execute using PAM.

The Texas Instruments SAL960 Assembler is a unique program which has been written to simplify the user's programming task. Directives and programming functions can be expressed in assembly language and SAL960 will translate them into machine language. The assembler provides two outputs: an object program (tape or deck) and an assembly list. The assembly listing shows the original source program statements side by side with the object program instructions created from them.

Another feature of the software is the Linking Relocating Loader (LRL960). The LRL960 links separately assembled programs and program segments by combining the text and completing the assembly process for external symbols. LRL960 will optionally load a program as it is linked.

Utility programs are available to users of PAM960. These programs perform specific software control tasks.

SECTION II

SYMBOLIC CODING IN ASSEMBLY LANGUAGE

2-1 INTRODUCTION.

The first portion of this section discusses symbolic coding in detail. If the user is already familiar with symbolic coding techniques, he may proceed directly to paragraph 2-4.3 for a summary of the first portion. Explanation of the Model 960 Computer assembly language begins with paragraph 2-5.

The programmer's job is to:

a. Arrange input and output data.

b. Establish "work areas" in storage.

c. Create constants or text values used in calculations.

d. Choose and write the instructions that move data, perform appropriate tests and calculations, handle exception conditions, and arrange data in a format specified for output.

Assembly language with symbolic notation is one method by which this work is done.

2-2 PURPOSE OF ASSEMBLIES.

Programming in a symbolic language offers important advantages over programming in the actual language of the computer.

a. Mnemonic operation codes are more meaningful than machine language. For instance, the actual operation code for the instruction Store General Register in right justified hexadecimal is 12. The mnemonic operation code in assembly language is ST.

b. Addresses of data and instructions can be written in symbolic form. The programmer is thereby relieved of severe problems in the effective allocation of storage, and the resulting program is far easier to modify. Furthermore, the use of symbolic addresses reduces the clerical aspects of programming and eliminates many programming errors. If the symbols chosen are meaningful, the program is also much easier to read and understand than if written with numeric addresses.

c. Symbolic assembly directives and data generation statements permit the introduction of constants, reservation of space for results, definition of instructions, control of the assembly process, and manipulation of symbols.

The sum effect of these advantages is so great that it is virtually out of the question to program in actual machine language; that is, to write actual operation codes and numeric address displacements.

2-3 THE ASSEMBLY PROCESS.

An assembly language program cannot be executed directly by the computer. The mnemonic operation codes and symbolic addresses must be translated into machine language. This is the function of the Symbolic Assembly Language (SAL) processor.

The assembly process begins with a source program which is written by the programmer. Ordinarily, a special coding form is used (Figure 2-1). Cards or paper tape are punched from this form, one card or line for each line of coding making up the source program. This source program becomes the primary input to the assembly process. The Symbolic Assembly Language (SAL) processor controls the assembly process in the computer (Figure 2-2).

SAL generates two outputs. The first is an object program. Actual machine instructions in the object program correspond to the source program statements written by the programmer. The object program can be output on punched cards, paper tape, or disk. The second output is an assembly listing. This important document shows the original source program statements side by side with the object program instructions created from them. An example is shown in Figure 2-3.

Note the following in the example:

a. The items under A show the decimal line or sequence number of the source statement. It is printed to assist the programmer when correcting his source program. It has no effect on the object program.

b. The items under B show the hexadecimal addresses of the instructions, constants, and areas of storage specified by the programmer.

PROBLEM _____

_____ SYMBOLIC CODING FORM

PROGRAMMER _____

73 Identification 80

```
1        5        10       15       20       25       30       35       40       45       50
BCR2BP   PSEG                               BINARY CARD TØ BINARY TAPE
SUPENT   EQU    X'7F'
         REF    CRPRB,PPRB,EØF,ØN EXTERNAL REFERENCES
*  *FILL BUFFER WITH 80 CHARACTER BINARY RECØRD *
START    LA     3,CRPRB            PRB DISPLACEMENT
         SXBS   *SUPENT            CALL SUPERVISØR I/Ø
*  *ØUTPUT BINARY RECØRD BUFFER TØ TAPE PUNCH *
         LA     3,PPRB
         SXBS   *SUPENT
*  *CHECK FØR END ØF FILE RECØRD *
         BFNE   EØF,ØN,START
         LA     3,EØJ              END ØF JØB
         SXBS   *SUPENT            VIA SUPERVISØR
EØJ      EQU    X'1800'
         END
```

T1-12502

Figure 2-1. Sample Symbolic Coding Sheet

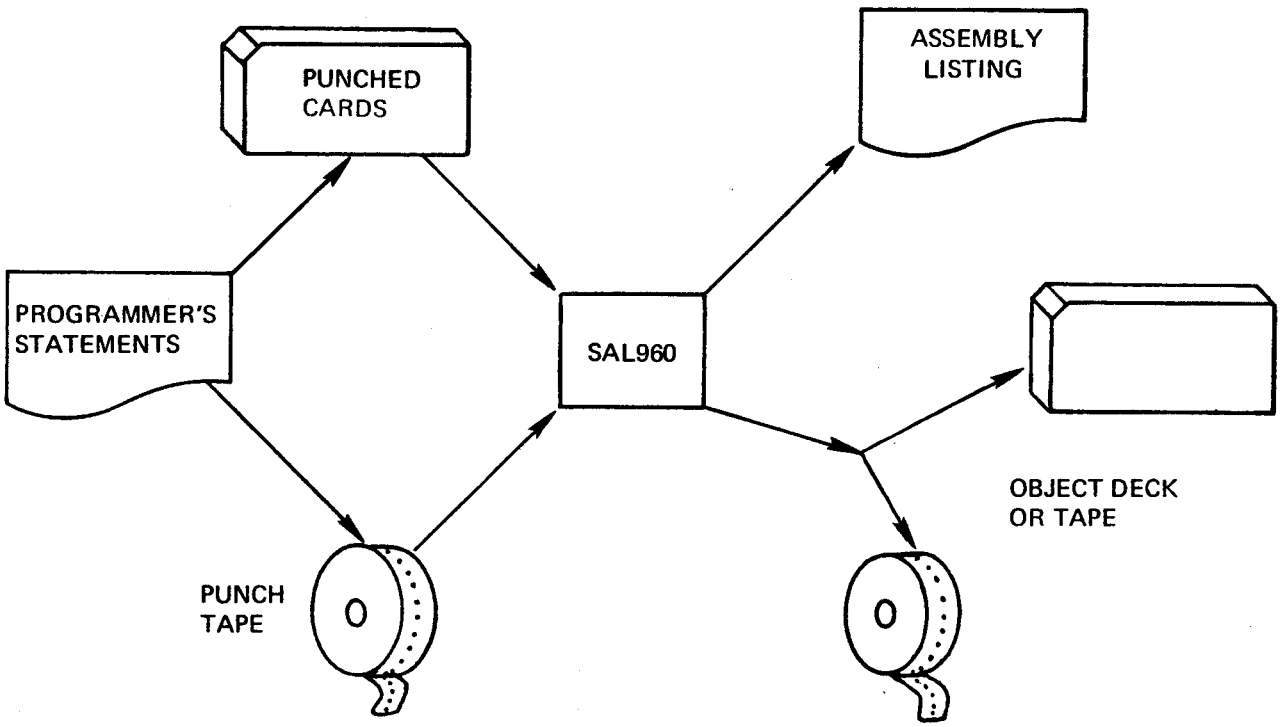


Figure 2-2. Schematic Representation of the Assembly Process

```
A     B      C          D
0001  0000               BCR2BP PSEG                BINARY CARD TO BINARY TAPE
0002  0000               SUPENT EQU  X'7F'
0003  0000                      REF  CRPRB,PPRB,EOF,ON EXTERNAL REFERENCES
0004                     *  *FILL BUFFER WITH 80 CHARACTER BINARY RECORD *
0005  0000 44830000      START  LA   3,*CRPRB       PRB DISPLACEMENT
0006  0002 7980007F             SXBS *SUPENT       CALL SUPERVISOR I/O
0007                     *  *OUTPUT BINARY RECORD BUFFER TO TAPE PUNCH *
0008  0004 44830000             LA   3,*PPRB
0009  0006 7980007F             SXBS *SUPENT
0010                     *  *CHECK FOR END OF FILE RECORD *
0011  0008 84000000             BFNE EOF,ON,START
0012  000A 44831800             LA   3,EOJ         END OF JOB
0013  000C 7980007F             SXBS *SUPENT       VIA SUPERVISOR
0014  000F               EOJ    EQU  X'1800'
0015  000E                      END
```

Figure 2-3. Example of Object Program

c. The items under C are a hexadecimal representation of the corresponding instructions and constants.

d. The items listed under D should be exactly the same as the handwritten entries on the coding sheet. This provides a good check on the accuracy of the keypunching and ample opportunity to comment on the function performed by the instruction.

2-4 SYMBOLIC CODING.

Each line of the coding sheet represents one symbolic statement. Each symbolic statement is used to tell SAL to assemble a machine language instruction, a data constant, or to do something during assembly time. Approximately 24 statements can be written on each coding sheet.

All instructions have a location in memory which they will occupy when the object program is being executed. Instructions also have an operation code and usually one or more operands. Take the case of a general Format II instruction which moves the contents of one memory location to another. This instruction would begin at some address in main storage, have a operation hexadecimal code of 05, and contain the addresses of two operands. The address of an instruction, its operation code, and the data addresses correspond respectively to the following fields on the coding sheet: Name, Operation, Operand. The entries on the coding sheet will be made symbolically, rather than in machine language.

The first six columns of the coding sheet are called the name field. This field gives symbolic names to the locations referred to by the program. For instance, if the program contains a routine to handle alarm scanning, it would be simpler if the machine address of the routine did not have to be remembered. After assigning a name to the first instruction in this routine, a symbolic branch instruction referring to that name can be written. Then SAL, in converting the program to machine language, will remember the machine address of the symbolic name and will use it in the object program whenever the programmer refers to it.

Symbolic names are also referred to as either symbols or labels. Some of the characteristics of symbolic names are:

a. Symbolic names are usually given to instructions, data fields, flags, or CRU lines referred to in programs.

b. Generally, a symbol cannot be used in the operand field unless it also appears in the name field of a symbolic statement. That is, a symbol in your program cannot be referenced unless it is used as the name of one of the instructions, directives, or data fields.

c. Symbols are restricted in length to six characters or less and must contain only letters or numbers. They must begin with a letter.

d. Within the preceding limitations, any symbol may be used.

Each line on a coding sheet is one symbolic statement. A symbolic statement can be a machine instruction, a data definition, or an assembly directive which gives some information to the assembler for use during the assembly process. The operation field on the coding sheet tells the processor whether the symbolic statement is an instruction or something else. Although the name field of a symbolic statement may be left blank, the operation field must contain a mnemonic that is recognizable in SAL. If the symbolic statement is an instruction, the mnemonic represents one of the computer's operation codes. Directives to the assembler are also recognized by a mnemonic.

For example, MLA instructs SAL to assemble a Shift Memory Left Arithmetic instruction. The mnemonic for the instruction is placed in the operation field of the coding sheet. Each instruction and directive has its own unique mnemonic. These are given in the Programmer's Reference Manual.

The operand field on the coding sheet contains the remainder of the instruction. That is, the operation code of an instruction is represented in the operation field by a mnemonic, while the locations of the data to be operated upon are put in the operand field.

The operand may be followed by a comment. This has no effect on the assembly.

The program coded using SAL is called the source program.

The sole function of the source program is to provide input data for the assembler. No instruction in the source program is executed during the assembly (translation) process. The output data of the assembler will be a machine language program called the object file. This may be paper tape, cards, or other media. The object file is the program converted into machine language. It can be loaded, either now or later, into the computer for execution. There is no need to reassemble the program each time it is executed. The object deck or tape can be used over and over again until changes are made in the program.

To obtain an object (machine language) program from the symbolic source program, the assembler must first be loaded into the computer's memory. As the assembler is being executed, it will read in the source program and convert it to the machine language that will be the object program. There are two outputs from the assembly process. One is the object file, while the other is an assembly listing.

2-4.1 LOCATION COUNTER. The computer, while executing the assembler program, acts as a clerk. One of the clerical tasks of the assembler is to assign machine addresses to symbolic names, and to remember these addresses and use them in the object program whenever the symbol is used in the operand of the source statement.

For instance, when the assembler encounters the following source statement, it must assign a machine address to the symbol BEGIN.

NAME	OPERATION	OPERAND
BEGIN	LA	1,THERE

The assembler must remember the address of BEGIN so it can insert that address when it encounters the following branch instruction.

NAME	OPERATION	OPERAND
	B	BEGIN

To be able to assign a machine address to a symbol, the assembler contains an internal counter. This counter is called the Location Counter and keeps track of the addresses in the source program as it is being assembled. The Location Counter is incremented as each symbolic statement is processed. The length, in words, of main storage area required by each statement determines how much the Location Counter is incremented. For instance, assume that the Location Counter is set to decimal 1000 when the following symbolic statement is read by the assembler.

NAME	OPERATION	OPERAND
BEGIN	LA	1,THERE

LA is the mnemonic for the Load Register with Effective Address instruction.

When the assembler encounters the preceding statement, it assigns the address of decimal 1000 to the symbol BEGIN and steps the Location Counter to decimal 1002.

Whenever the assembler finds an entry in the name field, it assigns the setting of the Location Counter to that name. It then increments the counter by the number of words required by the statement. The LA instruction in the above example is two words long, and the Location Counter steps from 1000 to 1002.

The Location Counter is just a data area within the assembler. The assembler gives it some initial setting and steps as required during the assembly process. Its main function is to be able to assign an address to symbols as they are encountered in the source program. The object program, when loaded into the computer later, might actually reside at locations different than those assigned at assembly time. This is known as program relocation and will be discussed separately later in this manual.

2-4.2 SYMBOL TABLE. The assembler uses its Location Counter to assign addresses to symbols. However, the assembler needs to retain the address it assigns to each symbol. These are stored in another data area within the assembler program. This area is referred to as the Symbol Table. When a symbol is encountered in the name field of a symbolic statement, that symbol, as well as the Location Counter setting, is placed in the Symbol Table. The area of storage used for the Symbol Table is limited by the memory size of the computer. SAL limits the length and quantity of symbols used in a program.

Whenever the assembler finds a symbol in the operand field, it searches for the symbol in the Symbol Table. When it locates the symbol, it obtains its machine address and uses it in computing the assembled instruction. Of course, the symbol must be defined somewhere in the source program.

SAL is a two-pass assembler. The first pass of a two-pass assembler does not produce an object program. It reads the source program and builds a complete Symbol Table.

If bulk storage is available, some versions of the assembler have an intermediate output. The intermediate output from the first pass is used as input data for the second pass. This eliminates the requirement on the user to physically enter the source data twice.

During the second pass, the assembler program uses the Symbol Table to complete the assembly of the statements. The output of the second pass is the object file and assembly listing.

2-4.3 CODING SUMMARY.

a. During assembly time, the assembler program is executed using a source program as input data.

b. The output data from the assembler consists of an object file and its assembly listing.

c. A Location Counter in the assembler keeps a record of the storage locations that are used by the object program.

d. When a source statement contains a name, the current setting of the Location Counter is given to the label.

e. Each label and the address assigned to it is placed in the assembler's Symbol Table.

f. SAL is a two-pass assembler.

g. During the first pass, the source program is read and the Symbol Table is generated.

h. During the second pass, the Symbol Table is used to complete the assembly, and produce the object program with its program listing.

2-5 SYMBOLIC LINE FORMAT.

The symbolic input line accepted by the assembler may contain a name field, operation field, operand field, and a comment field – or the entire line may be a comment. An input line is the first 60 characters read from a source record. The remainder of the record may contain a sequence number. On cards the sequence number is in columns 73 through 80. The sequence number is not used by the assembler but is important for the Source Maintenance Routine. Card columns 61 through 72 are left blank for compatibility with future additions to SAL.

In the case of punched tape, an input line is a string of characters. The first are the symbolic line characters and the last are a carriage return, transmitter off, line feed, and rubout characters (Figure 2-4). The input line must be no more than 60 characters, not including rubouts or the terminal characters. Input lines are free-form within the limits described. All rubouts are ignored in the source lines and do not affect the character count.

The character set acceptable for SAL includes:

Letters	A through Z
Digits	0 through 9
Special Characters	8 / - + , () ' @ $ #
Field Terminators	blank
Line Control	carriage return, line feed, x-off

2-5.1 COMMENT LINES. Comment lines provide the user with a place to annotate program listings. They are indicated by an initial character which is an asterisk (*). The remaining characters are arbitrary. The comment line in no way affects the assembly process. The line is merely reproduced in the printed output.

2-5.2 NAME FIELD. Names (also called symbols or labels) are provided for symbolic references to instructions, values, and data. A label is composed of from one to six characters, all of which must be letters or numbers. The first character of a label must be a letter (A-Z).

If a label is used, the first character must begin the input line. The label is terminated by the first space.

2-5.3 OPERATION FIELD. The operation field is used to describe the required action. It may be a mnemonic operation code, assembler directive, or data generation directive. The field consists of from one to four characters terminated by a space. The first character of the operation field must be preceded by at least one space.

2-5.4 OPERAND FIELD. The operand field consists of a list of expressions or sublists, separated by commas. The list is terminated by the first space or end-of-line character.

EXPRS1,EXPRS2,(EXPRS3,EXPRS4)

In the preceding example, (EXPRS3,EXPRS4) is a sublist. A sublist is a sequence of expressions enclosed in parentheses and arranged so each succeeding expression is connected with the preceding expression by a comma.

If fewer than the required number of fields appear in a source line, the remaining fields are assumed to be not applicable or zero. If the currency symbol ($) appears as an item in an expression, the current value of the assembler's location counter will be used as its numeric equivalent.

Expressions are strings of items separated by arithmetic operators and terminated by a space or end-of-line code.

Addition	+
Subtraction	-
Multiplication	*
Division	/

An item consists of a symbolic address, currency symbol ($), or a numeric value. Numeric items may be decimal or hexadecimal. If the first character of an item is not numeric, $, C', or X', it is assumed to be symbolic. A decimal item is a string of numeric characters (0-9), the first being non-zero. A hexadecimal item is indicated by an X followed by a string of hexadecimal digits (0-9 A-F), enclosed by apostrophes.

Expressions are evaluated left to right; i.e., all operations are performed in order of occurrence.

A symbol is defined for assembly purposes as relocatable if the value assigned must be modified at load time by the addition of a relocation constant.

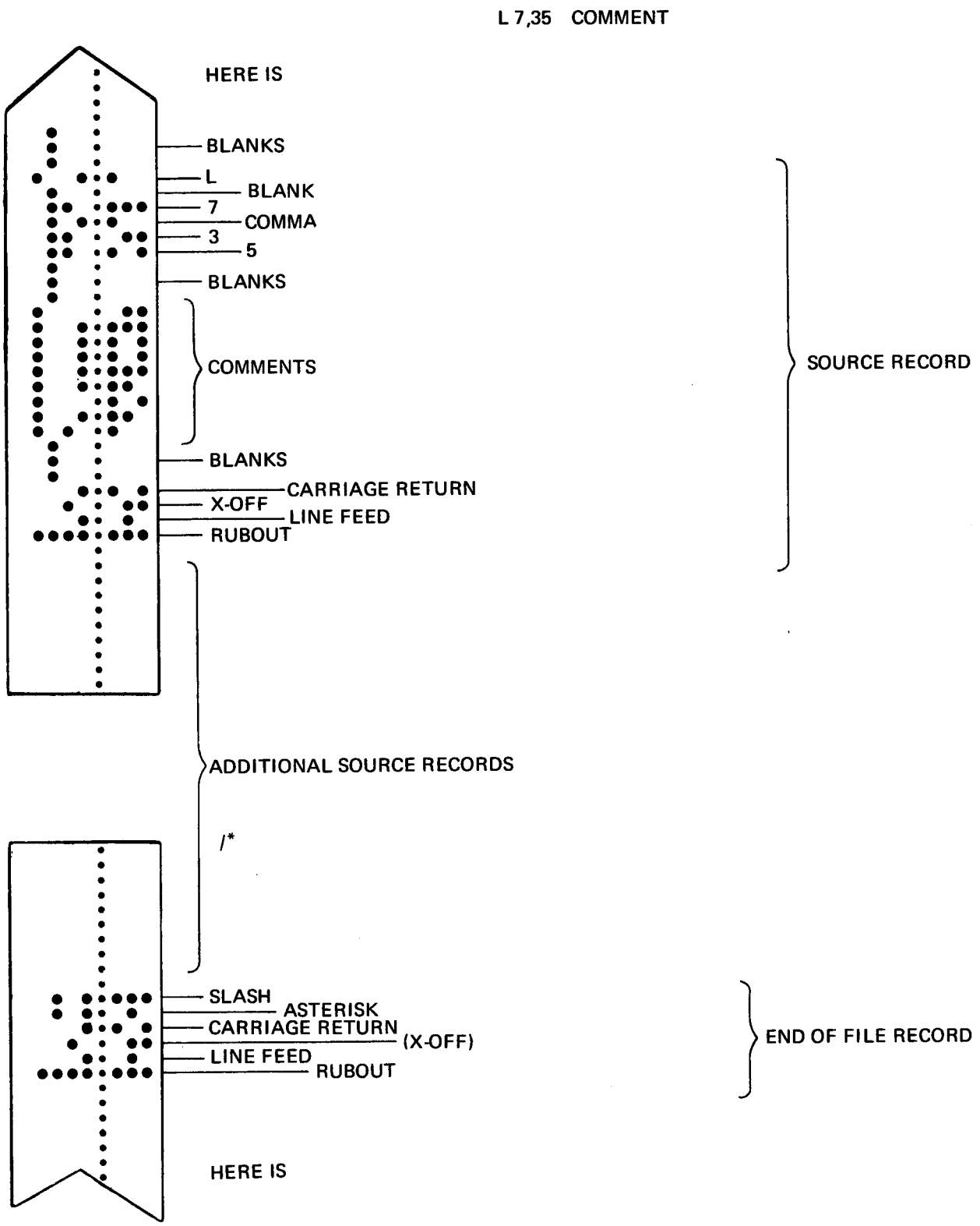


Figure 2-4. Format For Source Punch Tape

Relocated Symbol Value = Assembled Symbol Value + Relocation Constant

The only address values in this category are those which refer to memory directly; that is, 16-bit Format I memory address operands and address constants appearing as DATA directive operands.

Expressions containing relocatable symbols may also be relocatable, illegal, or nonrelocatable. When an expression consists of at least one relocatable item, the expression is:

Relocatable, if Δ, the sum of the added relocatable items minus the sum of the subtracted relocatable items is equal to 1.

Nonrelocatable, if $\Delta = 0$.

Illegal, if $\Delta < 0$, or if $\Delta > 1$, or if the expression involves any operations other than addition and subtraction upon two relocatable items.

Symbols used with an @ attribute are nonrelocatable.

The use of an external reference in an arithmetic expression is an error.

Sample expressions are:

JOE+TOM*3/BOB	Illegal if TOM and BOB are relocatable.
$+5	Legal
LEA-6	Legal
5034	Legal
XYZ+F4	Illegal if XYZ and F4 are relocatable.

PROGRAMMING NOTE

Due to the variety of address classes found in the computer, it is not always possible to determine the class attributes of an expression containing several symbols. In this case, the class attributes of the first symbol in the expression are used to define the attributes of the expression.

2-5.5 **COMMENT FIELD.** Comments may be written on any line. Any characters which appear between the space which terminates the operand field and the end-of-line character or card column 73 are treated as commentary. The comment field has no effect on the assembly process.

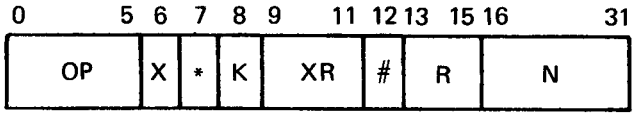
Typical symbolic statements are:

NAME	OPERATION	OPERAND
PT1	L	1,BUFFER+2
	ST	1,*ADDR
PT2	LA	0,1
	LA	2,-5
	ST	1,ANS,2
	ARB	0,$-2,2

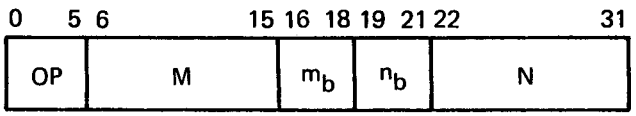
2-6 MACHINE INSTRUCTION FORMATS.

Machine instructions are implemented in three basic formats. Each instruction requires two consecutive memory locations. The formats are defined as follows:

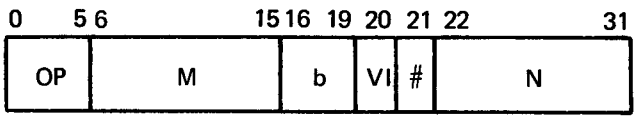
GROUP I



GROUP II



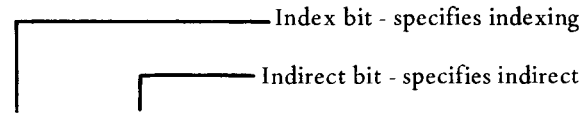
GROUP III



A description of each of the fields is as follows:

OP	The operation code field of an instruction.
*	The bit of the instruction used to specify indirect addressing.
X	The bit of the instruction used to specify that indexing is to be done.
K	Immediate operand indicator.
XR	The field of the instruction used to specify an index register.
#	The field of the instruction used to specify alternate mode registers.
R	The field of the instruction that specifies a register in the register file or specifies a shift count.
N	The field of an instruction used as an address field.
M	The field of an instruction used as an address field in two address instructions.
M_b	The field of instruction used to specify a base register to be used with the M address field.
N_b	The field of an instruction used to specify a base register to be used with N address field.
V_1	A bit used as an immediate value in flag and bit instructions.
b	The field used to specify a flag address within a memory word or the number of bits in a communication register.

The * and X fields, used to specify address modifications, have the following meanings:



X	*	
0	0	Direct operand
0	1	Indirect operand
1	0	Indexed operand
1	1	Indirect, indexed operand

Address modifications, as specified by the X and * fields, are performed using the XR and and the N fields. If indirect addressing and indexing are both specified, the indexing is done before indirect address if bit 5 (IM) of the status register is zero. If it is a one, the indexing is done after the indirect addressing. Any of the eight registers of each Mode Register File may be used as an index register. Supervisor index registers are used in supervisor mode and worker index registers are used in worker mode.

The index control bit in the Status Register permits optional indexing or post-indexing. This controls the relation of indexing to indirect addressing. If the index control bit is a one, indexing precedes indirect addressing. If the index control bit is zero, indexing follows indirect addressing. If indirect addressing is not involved, the two modes are equivalent.

The SAL programmer has at his disposal flexible address modification capabilities, a comprehensive set of machine instructions that make available the full capabilities of the machine hardware.

INDEXING – Indexing is specified by the presence of a register number (0-7) or symbol after the address operand.

EXAMPLE: L 2,N,3

NOTE

The underlined entries in the operand field of example instructions are used to identify the action described.

INDIRECT – Indirect addressing is specified by placing an asterisk (*) before the address operand.

EXAMPLE: L 2,*,N

IMMEDIATE – Immediate addressing is accomplished by using an alternate mnemonic of the operation, e.g., the immediate counterpart of the Load Register instruction (L) is LA (Load Address).

EXAMPLE: LA 2,N

RELATIVE – When the "at" symbol @ precedes the address operand, the relative address of the operand is used rather than the relocatable or absolute address.

EXAMPLE: L 2,@N,5

ALTERNATE MODE REGISTERS – The presence of the pound sign (#) preceding the operand list causes the general register of the inactive mode to be used in the execution of that instruction.

EXAMPLE: L #2,N

PROGRAMMING NOTE

The inactive mode general register 2 is loaded. Index register 3 from the active mode is used to compute the effective address.

For maximum flexibility in address modification, any combination of the attributes may be used. Examples of some of the possible combinations are as follows:

a. L #2,N

b. L 2,*N,3

c. L #2,*@N,3

d. LA 2,@N,3

2-6.1 FORMAT I MACHINE INSTRUCTIONS.

SUBSET A

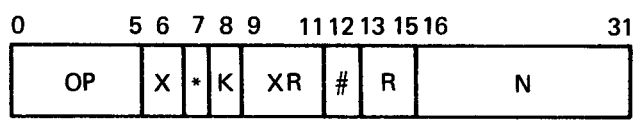
MNEMONIC	INSTRUCTION	OP CODE
L	Load GR from Memory	11
LA	Load GR with Effective Address	11
ST	Store GR into Memory	12
A	Add to GR from Memory	13
AA	Add Effective Address to GR	13
S	Subtract Memory Contents from GR	14
SA	Subtract Effective Address from GR	14
LOT	Load GR with Flag Tally of Memory Word	15
LOTA	Load GR with Flag Tally of Effective Address	15
N	And GR with Memory	16
NA	And GR with Effective Address	16
OR	Or GR with Memory	17
ORA	Or GR with Effective Address	17
SAT	Shift and Add Leading Zero's Tally of Memory Word to GR	1B
BL	Branch and Link to Subroutine	1D
*BL	Indirect Branch and Link to Subroutine	1D
MLAX	Shift Memory Left Arithmetic	1B
MRAX	Shift Memory Right Arithmetic	19
MRRX	Rotate Memory Right Logical	1A

TYPICAL SOURCE STATEMENT

LABEL OPER #GR,*@ADDRESS,XR COMMENTS SEQ

(Entries that are underlined are permitted, but not required.)

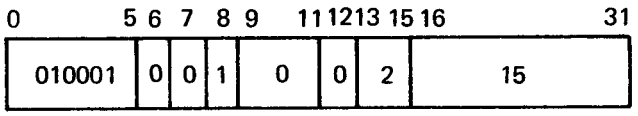
The corresponding instruction format is:



A specific Subset A instruction such as

LA 2,15 LOAD EFFECTIVE ADDRESS

would have the format:



which shows the immediate operand indicator, bit 8, set and the variable field R designates that 15 will be loaded into register 2.

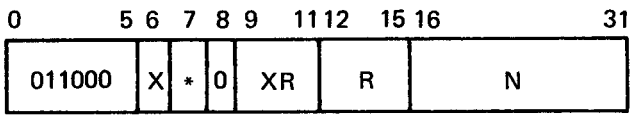
SUBSET B

MNEMONIC	INSTRUCTION	OP CODE
MLA	Shift Memory Left Arithmetic	1B
MRA	Shift Memory Right Arithmetic	19
MRR	Rotate Memory Right Logical	1A

TYPICAL SOURCE STATEMENT

LABEL OPER COUNT,*@ADDRESS,XR COMMENTS SEQ

This subset contains the memory shift instructions and has the following typical object format:



The illustration uses the Shift Memory Left operation as an example. For this subset, the variable field R contains the shift count.

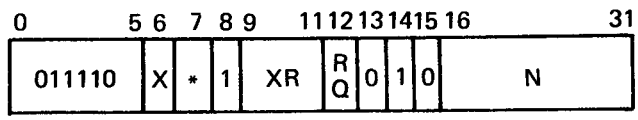
SUBSET C

MNEMONIC	INSTRUCTION	OP CODE
NOP	No Operation	1C
B	Branch	1C
*B	Indirect Branch	1C
XSB	Transfer Control, Branch to Supervisor	1C
*XSB	XC, Indirect Branch to Supervisor	1C
XWB	XC, Branch to Worker	1C
*XWB	XC, Indirect Branch to Worker	1C
XS	Transfer Control to Supervisor	1C
XW	Transfer Control to Worker	1C
SSB	Save Status, Branch	1E
SXBS	Save Status, XC, Branch to Supervisor	1E
SXBW	Save Status, XC, Branch to Worker	1E
SXS	Save Status, XC, to Supervisor	1E
SXW	Save Status, XC, to Worker	1E
SS	Save Status, Continue	1E
LDS	Load Status	1F

TYPICAL SOURCE STATEMENT

LABEL SSB *@ADDRESS,XR,RQ COMMENTS SEQ

The mnemonic SSB is the Save Status, Branch operation. This source statement has the following typical object format.



RQ is the relative address control bit.

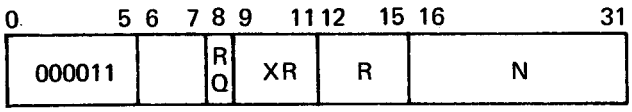
SUBSET D

MNEMONIC	INSTRUCTION	OP CODE
ARB	Add to Register and Branch	03

TYPICAL SOURCE STATEMENT

LABEL ARB ADDND,@ADDRESS,GR,RQ COMMENTS SEQ

The typical object format is:



The R field contains the signed addend and bits 6 and 7 are unused.

SUBSET E

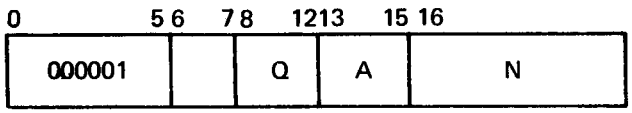
MNEMONIC	INSTRUCTION	OP CODE
ADAC	Activate Direct Access Controller	01

TYPICAL SOURCE STATEMENT

LABEL ADAC DEVADD,LISTAD,Q COMMENTS SEQ

The mnemonic ADAC is for the Activate Direct Access Controller operation.

TYPICAL OBJECT STATEMENT



The A field contains a device address and N field is the I/O command list address. The Q field is optional and may contain additional device address data.

2-6.2 FORMAT II MACHINE INSTRUCTIONS.

SUBSET A

MNEMONIC	INSTRUCTION	OP CODE
MOV	Move Memory Word to Memory Word	05
CM	Compare Memory with Memory	04
CML	Compare Memory with Limits in Memory	06

The typical source statement has two optional forms.

Option 1

LABEL MOV (DISP,GR),(DISP,GR) COMMENTS SEQ

LABEL CM

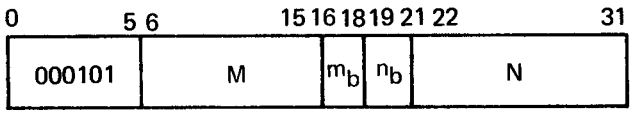
Option 2

LABEL MOV FROM,TO COMMENTS SEQ

PROGRAMMING NOTE

External references may not appear in Option 2 operands.

Typical Object Format:



m_b and n_b are explicitly defined base registers for Option 1, but are determined by the segment class in which the label is defined in Option 2.

SUBSET B

MNEMONIC	INSTRUCTION	OP CODE
BRRL	Branch Relative to Register and Link	QA

The typical source statement has two optional forms.

Option 1

LABEL BRRL LINK,(DISP,GR) COMMENTS SEQ

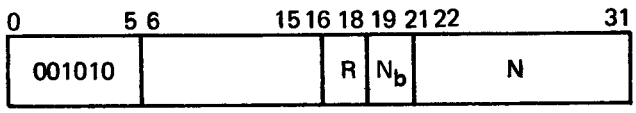
Option 2

LABEL BRRL LINK,THERE

PROGRAMMING NOTE

External References may not appear in the branch address of the Option 2 operand.

Typical object format:



In Option 1, the operand base, GR, is specified and is found in the N_b field of the format. In Option 2, the operand base is determined by the segment class in which the label is defined.

SUBSET C

MNEMONIC	INSTRUCTION	OP CODE
AMI	Add Immediate Value to Memory	08
CMI	Compare Memory with Immediate Memory	07

The typical source statement has two optional forms:

Option 1

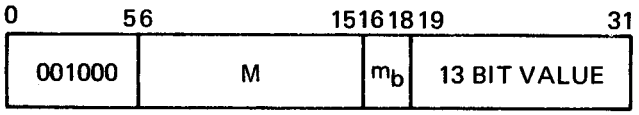
LABEL AMI (DISP,GR),VALUE COMMENTS SEQ

Option 2

LABEL AMI LOCAT,VALUE COMMENTS SEQ

PROGRAMMING NOTE
External References should not be used in the memory address part of the Option 2 operand.

Typical object format:



Option 1 allows explicit base register (m_b) definition and Option 2 base register is determined by the segment class in which the symbol is defined.

2-6.3 FORMAT III MACHINE INSTRUCTIONS.

SUBSET A

MNEMONIC	INSTRUCTION	OP CODE
XFNE	Transfer Control if Flag is Unequal to Operand	20
SETF	Set Flag	22

The typical source statement has two optional forms.

Option 1

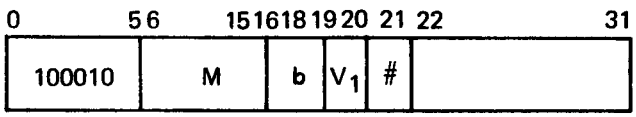
LABEL XFNE #FLAGN,F COMMENTS SEQ

LABEL SETF #FLAGN,F COMMENTS SEQ

Option 2

LABEL SETF #(WORD BIT),F COMMENTS SEQ

Typical object format:



The flag word address appears in M. The bit address within the word appears in b. Option 1 uses a flag name which has been defined by the assembly directive FLAG in a flag segment. Option 2 explicitly defines the appropriate flag bit. The symbol word must be nonrelocatable or appear in a flag segment. Self-defining terms (constants) may also be used in place of symbols in the Option 2 sublist. The value bit, V_1 is used to compare with the flag bit in memory. V_1 corresponds to F in the operand list.

SUBSET B

MNEMONIC	INSTRUCTION	OP CODE
BFNE	Branch if Flag Not Equal to Operand	21

The typical source statement is:

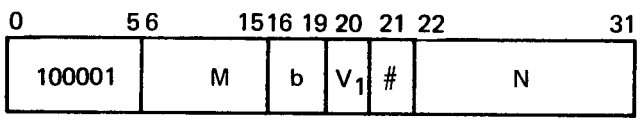
Option 1

LABEL BFNE #FLAGN,F,THERE COMMENTS SEQ

Option 2

LABEL BFNE #(WORD BIT),F,THERE COMMENTS SEQ

Typical object format:



The flag word address appears in M. The bit address within the word appears in b. The branch address appears in N. Option 1 uses the flag name designated in the flag segment of the assembly and Option 2 explicitly defines the flag bit address. The symbol WORD in the Option 2 sublist must be nonrelocatable or appear in a flag segment. Self-defining terms (constants) may also be used in place of symbols in Option 2 sublists. The value bit, V_1, is used to compare with the flag bit in memory. V_1 corresponds to F in the operand list.

SUBSET C

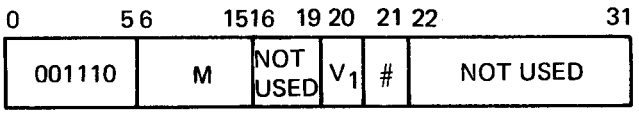
MNEMONIC	INSTRUCTION	OP CODE
SETB	Output Bit to CRU	0D
XBNE	Transfer Control if CRU Bit is Unequal to Operand	0E

The typical source statement is:

LABEL SETB #BITOUT,B COMMENTS SEQ

LABEL XBNE #BITIN,B COMMENTS SEQ

Typical object format:



M is the bit address in the CRU. b is not used. The value bit, V_1, is used to set or compare the addressed CRU bit. V_1 corresponds to B in the operand list.

PROGRAMMING NOTE

Self-defining terms used as bit addresses must be multiplied by the factor 16 in order to generate the correct CRU address. Symbols defined using a connect directive generate proper CRU addresses.

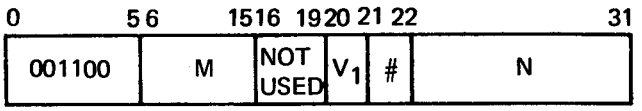
SUBSET D

MNEMONIC	INSTRUCTION	OP CODE
BBNE	Branch if CRU Bit is Not Equal to Operand	0C

The typical source statement is:

LABEL BBNE #BITIN,B,THERE COMMENTS SEQ

Typical object format:



M is the CRU bit address. V_1 is the immediate value operand corresponding to B in the operand list. N is the branch address.

PROGRAMMING NOTE

Self-defining terms used as bit addresses must be multiplied by the factor 16 in order to generate the desired CRU address. Symbols defined using a connect directive generate proper CRU addresses.

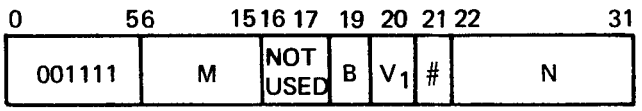
SUBSET E

MNEMONIC	INSTRUCTION	OP CODE
TSBX	Test input bit for comparison with operand bit, if input bit equals operand bit then set CPU output bit, or else transfer control to alternate mode.	0F

The typical source statement is:

LABEL TSBX #BITIN,B,BITOUT,OB

Typical object format:



M is the input bit address. V_1 corresponds to the input bit value to be tested (B in the operand list). N is the output bit address. B corresponds to the output bit value to be set (OB in the operand list).

PROGRAMMING NOTE

Self-defining terms used as bit addresses must be multiplied by the factor 16 in order to generate the desired CRU address. Symbols defined using a connect directive generate proper CRU addresses.

SUBSET F

MNEMONIC	INSTRUCTION	OP CODE
LDCR	Load CRU Register Memory	02
STCR	Store CRU Register into Memory	0B

The typical source statement has two options:

Option 1

LABEL LDCR #(LINE,FL),MEMORY COMMENTS SEQ

LABEL STCR #(LINE,FL),MEMORY COMMENTS SEQ

Option 2

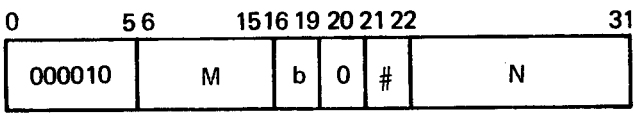
LABEL LDCR #CRFLD,MEMORY COMMENTS SEQ

LABEL STCR #CRFLD,MEMORY COMMENTS SEQ

PROGRAMMING NOTE

Symbols used to implicity reference CRU registers must be defined using the connect directive.

Typical object format:



M is the CRU starting line address. The b field contains the number of bits in the CRU register -1. N is the memory address of the data.

2-7 ASSEMBLER DIRECTIVES.

Assembler directives or *pseudo-ops* have formats similar to symbolic instructions, but do not directly cause code generation. Instead, they are directives to the assembler itself. Symbolic Assembly Language (SAL) directives are included to:

a. Identify symbols and program segments.

b. Allocate memory within a segment.

c. Define constants.

d. Pass data to the loader.

e. Supplement the source language.

If labels are used with assembler directives, they will be assigned the current location counter value unless otherwise specified. All assembler directives are listed in Table 2-1.

TABLE 2-1

ASSEMBLER DIRECTIVES

NAME	USE
PSEG	Procedure Segment
DSEG	Data Segment
FSEG	Flag Segment
BSEG	CRU Symbolic Address Segment
ABS	Absolute Program Declaration
MODE	Permit Use of Alternate Mode Registers
END	Segment Termination
REF	External Reference
DEF	Entry Point or External Symbol Definition
PST	Punch and List Symbol Table
RES	Reserve Memory
FLAG	Name Flag Bit
CON	Assign CRU Bit Address
EQU	Equate
DATA	Define Data
FRM	Format
PAGE	Page Eject
TITL	Identification

Certain directives are not used in all segment classes. The following list defines permitted usage of directives.

PSEG & DSEG	FSEG	BSEG
All except	FLAG	EQU
		TITL
FLAG and	EQU	CON
	PAGE	PST
CON	RES	END
	TITL	PAGE
	DATA	
	PST	
	END	

2-7.1 ABSOLUTE ATTRIBUTE DIRECTIVE: ABS.

General Form: ABS OPERAND

This statement instructs SAL to assign an absolute attribute to symbols defined subsequently until an END card is encountered. A segment identifier must follow an ABS statement. The label field is not required. The assembler ignores this field. The term in the operand field initializes the absolute assembly program counter.

2-7.2 ALTERNATE MODE REGISTERS DIRECTIVE: MODE.

General Form: MODE

This statement notifies SAL to permit reference to alternate mode registers using the #attribute. Note that any use of # when SAL is not provided a currently active MODE directive, will cause an assembly error. Label and Operand field entries are ignored.

2-7.3 TERMINATE SEGMENT DIRECTIVE: END.

General Form: END OPERAND

This directive terminates a segment and revokes an active ABS or MODE statement. A non-blank entry in the operand field will be passed to the loader identified as a transfer vector. Entries in the label field are ignored.

```
Example 1.              ABS X'0410'
            P1          PSEG
                        MODE
            PROC        LA 7,0
                        L #3,NUMBER
            START       DATA PROC,X'8000'
                        END START

Example 2.  P2          PSEG
                        .
                        .
                        .
                        END
            FS1         FSEG
```

In the first example, the label START is the address of a status block that is loaded to start the program. The START label in the operand of END directive tells the loader where to transfer control. The absolute directive, ABS, forces the first instruction to be located at location X'0410'. The MODE directive enables the use of alternate mode register 3. The END directive terminates the PSEG, the ABS directive, and the MODE directive.

In the second example, the END directive terminates the procedure segment. An ABS, MODE, PAGE, TITL, or a Segment Identifier Directive must immediately follow END, except when it is used to mark the conclusion of a program. In that case, END must be followed by an End-of-File record. (/*)

2-7.4 DIRECTIVE: REF.

General Form: REF $OPERAND_1 OPERAND_2$...

This statement identifies symbols appearing in the operand list as external references. These externally referenced symbols are passed to the loader with appropriate data for subsequent assembly. REF may be used only in Procedure and Data program segments. The use of an external reference in an arithmetic expression is an error.

2-7.5 DIRECTIVE: DEF.

General Form: DEF $OPERAND_1$,$OPERAND_2$...

$OPERAND_n$

The DEF statement identifies those symbols which will be defined within a segment and made available for reference or call from another segment. In order to REFerence a symbol in another program, that symbol must appear in a DEF directive operand in the program in which the symbol is defined. DEF is not used in a FSEG, BSEG, or DSEG. Such usage would be redundant since symbols in those segments are external by convention.

2-7.6 PUNCH SYMBOL TABLE DIRECTIVE: PST.

General Form: PST

This statement directs SAL to list the symbol table on the system listing device and punch the table on the system object device. Entries in the label and operand fields are ignored.

2-7.7 MEMORY ALLOCATION DIRECTIVE: RES.

General Form: LABEL RES OPERAND

This statement is used to reserve word locations in memory, the term in the operand field entry is added algebraically to the contents of the current location counter. An entry in the label field is optional.

Example: XLABEL RES 10

In the above example, the operand 10 specifies that ten consecutive words will be reserved in memory. The label XLABEL will be the address of the first word reserved. Subsequent words in this reserved area may be addressed using XLABEL and indexing by the proper integer (1 to 9). Use of a relocatable expression in the operand field is an error.

Example 1:

```
1.              ABS         1000
2. MAIN         PSEG
3.              REF         SINE
                .
                .
                .
4. ADSINE       BL          1,SINE
5.              END
```

Example 2:

```
1. SUB          PSEG
2.              RES         10
3.              DEF         SINE
                .
                .
                .
4. SINE         ST          1,SAVE
5.              L           1,SAVE
6.              B           2,1
7.              END
```

Line 3 of the first segment indicates that the label SINE will appear in a separately assembled segment. Line 3, in the second segment, declares the label SINE to be externally defined so that the first segment may have access to the value of the label which appears at line 4 when the two segments are linked by the Linking Relocating Loader.

2-7.8 ASSIGNMENT DIRECTIVE: FLAG.

General Form: LABEL FLAG $OPERAND_1$

$OPERAND_2$...$OPERAND_n$

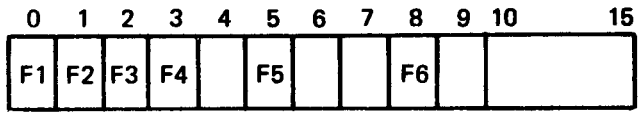
This directive allows naming of flag bit addresses in the Flag Segment (FSEG).

Flag addresses are assigned sequentially to the previously undefined symbols appearing in the operand list. The Current Flag Counter is started at bit 0 and is maintained modulo 16. Each time the counter passes through zero, the Flag Word Address Counter is incremented by one, thus addressing the next word.

For example:

```
XLABEL FSEG

FLAG F1,F2,F3,1,F4,F5,3,F6
```



Terms encountered in the operand list are added to the current location counter to appropriately advance the flag address. A constant or a symbol representing a constant appearing in the operand list specifies the number of flags to be skipped.

The FLAG directive may be used only in the Flag Segment. FLAG directives do not allocate memory and are used as a convenient method for generating flag addresses.

2-7.9 CONNECT DIRECTIVE: CON.

General Form: LABEL CON OPERAND,MODIFIER

The CONnect directive is used in the CRU Symbolic Address Segment (BSEG) to name the address of a CRU bit or the address of a series of CRU bits.

The modifier is optional and has a default value of 1.

Example 1:

```
COMM        BSEG        0
TAPE1       CON         18
TAPE2       CON         19,3
TAPE3       CON         20
TREG        CON         18
            END
```

Line 2 of the example assigns the name TAPE1 to bit location 18 of the bit program segment. Line 3 assigns a different name to the same location.

Line 3 assigns a name to bits 19, 20, and 21. These three bits may be referenced as a communication register. Line 4 assigns a label to bit 20. This bit is contained within the TAPE2 register, but may also be addressed as TAPE3.

Example 2:

```
COMM    BSEG    400
TAPE1   CON     418
TAPE2   CON     419,3
TAPE3   CON     420
TREG    CON     418
        END
```

The second example gives the identical bit addresses as the previous example because the difference between the BSEG operand and the CON operand determines the CRU bit address. This displacement value is limited by the number of bits in the instruction address field to $0 \leq M \leq 1023$. Addressing the CRU requires the use of base register 7. The value of the BSEG operand will normally be the value used as the CRU Base Address.

Note that the CON directive does not initialize any bits. CON is used only in a bit segment.

PROGRAMMING NOTE

A sequence of bits defined as a communication register with the CON directive (TAPE2 in the examples) may be addressed as a register by the LDCR and STCR instructions.

2-7.10 EQUATE DIRECTIVE: EQU.

General Form: LABEL EQU OPERAND

The EQU assignment directive assigns the value and attributes of the expressions in the operand field to the symbol appearing in the label field. Among other uses, this statement may be used to assign a name to a register.

Example: REGONE EQU 1

This directive would allow General Register 1 to be referred to as REGONE in subsequent instructions.

2-7.11 DATA DEFINITION DIRECTIVE: DATA.

General Form: LABEL DATA $OPERAND_1$

$OPERAND_2 ... OPERAND_n$

This directive is used to place specific values in memory. Values specified in the operand list are entered in successive core locations.

Four types of data are permitted. They are: decimal integer, hexadecimal, ASCII (plain language or Hollerith), and address.

A decimal integer list might appear as:

```
DATA 529,-3,65
```

A hexadecimal list:

```
DATA X'ABC0',X'A',X'3F10'
```

Note that for hexadecimal numbers, each entry is preceded by X and is enclosed by apostrophes. Neither hexadecimal nor decimal numbers may require more than 16 binary bits for internal representation. If a number (such as X'A') requires fewer than 16 bits, the number is right justified internally and the field is padded with leading zero's.

In whichever type a number is entered, its value (V) is limited by:

$$-2^{15} < V < 2^{15}-1.$$

An example of an ASCII list is:

```
DATA C'TEXAS INSTRUMENTS',C'960 SAL'
```

Note again that the data is enclosed in apostrophes and is preceded by C.

One memory word can store two ASCII characters so that the first constant, TEXAS INSTRUMENTS, requires nine storage words. There are 17 characters in the constant including the space between words. SAL left justifies ASCII characters and fills the right most half word with a blank if an odd number of characters is specified. The DATA directive may be used only in Procedure, Data, and Flag Segments.

The construction

```
DATA MDAT
```

where MDAT is a relocatable address constant is permitted in SAL960. Furthermore, the construction

```
DATA @MDAT
```

where @MDAT is a segment relative address constant is also allowed.

SAL can be expanded. New data types and their parameters will be added as they are developed.

2-7.12 SOURCE LANGUAGE EXTENSION DIRECTIVE: FRM.

General Form: LABEL FRM $OPERAND_1$,

$OPERAND_2 ... OPERAND_{32}$

New instructions and data structures can be designed using the format directive.

Field widths in bits are listed in the operand field.

```
XLAB FRM 4,2,10,5,5,6      Format Declaration

WORD XLAB 4,1,103,26,9,15  Format Reference
```

The programmer has the option to include a two-entry, parenthetic sublist in the FRM operand.

```
XLABEL FRM 6,3,7(X'FF',0)

    XLABEL 15,4,23
```

Should this option be exercised, a logical *and* will be performed between the first sublist entry and the final version of the FoRMatted word (or double word) and logical *or* performed between the second sublist entry and the formatted word. When the sublist is used, the number of binary bits (in the example, 16) required to represent each number of the sublist must not exceed the number of bits specified by the field width operands. Should the sublist number require less than the specified number of bits, the number will be right justified in a field with leading zero's added.

The number of bits specified in the operand list must equal either 16 or 32. Furthermore, when the 32-bit format is used, a field may not be defined that extends across the internal word boundary. An entry in the label field is required. After FoRMat has been defined, it is referenced by entering this label symbol in the operation code field. The label symbol may not exceed four characters in length. For example;

```
XLAB FRM 4,4,8

ALPHA XLAB 12,6,21

BRAVO XLAB 13,5,20
```

2-7.13 OUTPUT CONTROL.

Directive: PAGE

General Form: PAGE

The PAGE directive causes the listing output device to be advanced to *top of form*. The actual directive, PAGE will not be printed.

Directive: TITL

General Form: TITL OPERAND

This directive is used to specify the plain language (ASCII) characters to be used in program identification. These characters will be printed on the first line of each page of the list generated by the assembler.

Note that this does not refer to application output headings.

Example:

```
TITL TI 960 MONITOR SYSTEM
```

No further output instructions are required. The presence of TITL is sufficient to cause printing. Storage method and requirements are identical with ASCII DATA.

2-8 SYMBOLIC ASSEMBLY LANGUAGE.

Using SAL, the programmer may implement his programs as stand-alone units or may construct programs from one or more modules of four basic segment types. The four segment types are:

a. Procedures identified by the PSEG directive. The procedure segment is normally the main body of the program and contains computer instructions. It is the action part of a program. Symbol attributes in SAL assist the programmer in making procedures re-entrant.

b. Data (DSEG). Data segments are used to provide storage, I/O Buffers, and constants for procedure segments.

c. Flags (FSEG). A flag segment allows the programmer to symbolically address the memory bit by bit.

d. Communication Register Segment (BSEG). This segment simplifies the assignment and use of symbolic addresses for references to bit lines in the Communication Register Unit, both by register (field) and by individual bit.

There are many variations of this basic type of program segmentation possible. In most cases, the program segments permit very simple and quick convenient coding.

The computer feature which makes segmentation both convenient and efficient is the automatic use of specific base registers in Format III machine instructions. For example, when referring to a software flag in the Software Flag Segment (FSEG) with a Software Flag instruction, the value of the symbol representing the software flag is automatically biased with the contents of the Software Flag Base Register during execution of the instruction. This then

creates the bit address of the software flag. So, as the assembler is building an instruction that uses automatic base registers, the displacement of the symbol relative to the origin of the segment in which it was defined is placed in the instruction rather than the program relative address of the symbol. Format II instructions allow base registers to be specified in the instruction. For those instructions which do not use base registers, the segment relative value of the symbol rather than its program relative value may be specified with the use of the relative attribute, the "at" symbol, @. The relative attribute may also be used with the DATA directive. For example:

```
DATA @SYMBOL
```

This directive would cause a data word to be initialized with the value of the displacement of SYMBOL relative to the origin of the segment in which it was defined. If the relative attribute were not used in the example, the data word would be initialized with the program relative address of SYMBOL rather than its segment relative value. Example source statement:

```
L 1,@TEMP,4
```

If the symbol TEMP is defined in a Data Segment, the above use of the @ symbol effectively converts Format I instructions to the base-displacement addressing mode used in Format II instruction.

Segment Identifiers, PSEG, DSEG, and FSEG directives do not allow an operand field entry. A label field entry is required. The label and the assigned value are passed to the loader via the segment ID record. The operand field entry for the BSEG directive is an absolute bit address that is passed to the loader via an external definition. Symbols defined within a Flag segment, CRU segment, or Data segment will be passed to the loader as external symbols. The appropriate segment class attribute will be associated with symbols defined within a particular segment. Procedure and Data segments will normally reserve memory in some fashion or another. Flag segments may optionally reserve memory, but will normally be used for address assignment only. CRU symbolic address segments are used for address assignment.

Once the program is assembled by the SAL assembler, it may either be loaded into core memory for execution or combined with other segments in relocatable object format using the Linking Relocating Loader.

The SAL assembler creates the necessary data required by the LRL to perform its functions of linking the different segments, relocating segments, and completing the assembly process for external symbols.

Figure 2-5 shows the general structure of an example program. The program listing of paragraph 2-16 contains this specific example.

The flexibility of the computer usage is demonstrated by the segment structure of the program. Three independent process tasks are executed simultaneously under program and monitor control. Each process task has a data segment which is unique to the task being performed. The first 16 words form the worker task block for monitor communication. Because of independent task assembly, any task or tasks may be added or deleted without altering any part of the program except the relevant data segment(s). In the example, all data segments use the same execution logic provided by the re-entrant procedure segment. The one flag segment is also used by all tasks. The addresses of actual task process information input and output in relation to the computer is defined by the communication register symbolic address segment (BSEG), used by all three tasks.

2-9 OBJECT FORMATS.

The object program may be produced on either paper tape or cards, depending upon the peripheral equipment available.

2-9.1 OBJECT FORMAT. The object records for the Symbolic Assembler (or Linking Relocating Loader) have the following formats.

Each segment in an assembly will result in the output of an Identification Record and a Linkage Data Record(s) at the end of PASS 1.

During PASS 2 of the assembler, the text records are output. Text records contain the program text to be placed in memory and additional linkage data.

At the end of PASS 2, an end record is output.

Binary output records on paper tape will be separated by an x-off punch off and rubout or null code. Successive text records are punched as required until the end of the segment is reached. This is indicated by a segment end record. This record can contain a transfer location. The last character on the tape is an ASCII punch-off code.

All data on the object paper tape is punched 4 frames/16 bit word. Channels 1-4 of each frame contain one hexadecimal digit. Channel 5 is always zero. Channels 6 and 7 are always one. The following list displays the hexadecimal digits and the hexadecimal codes corresponding to the eight channels on the tape.

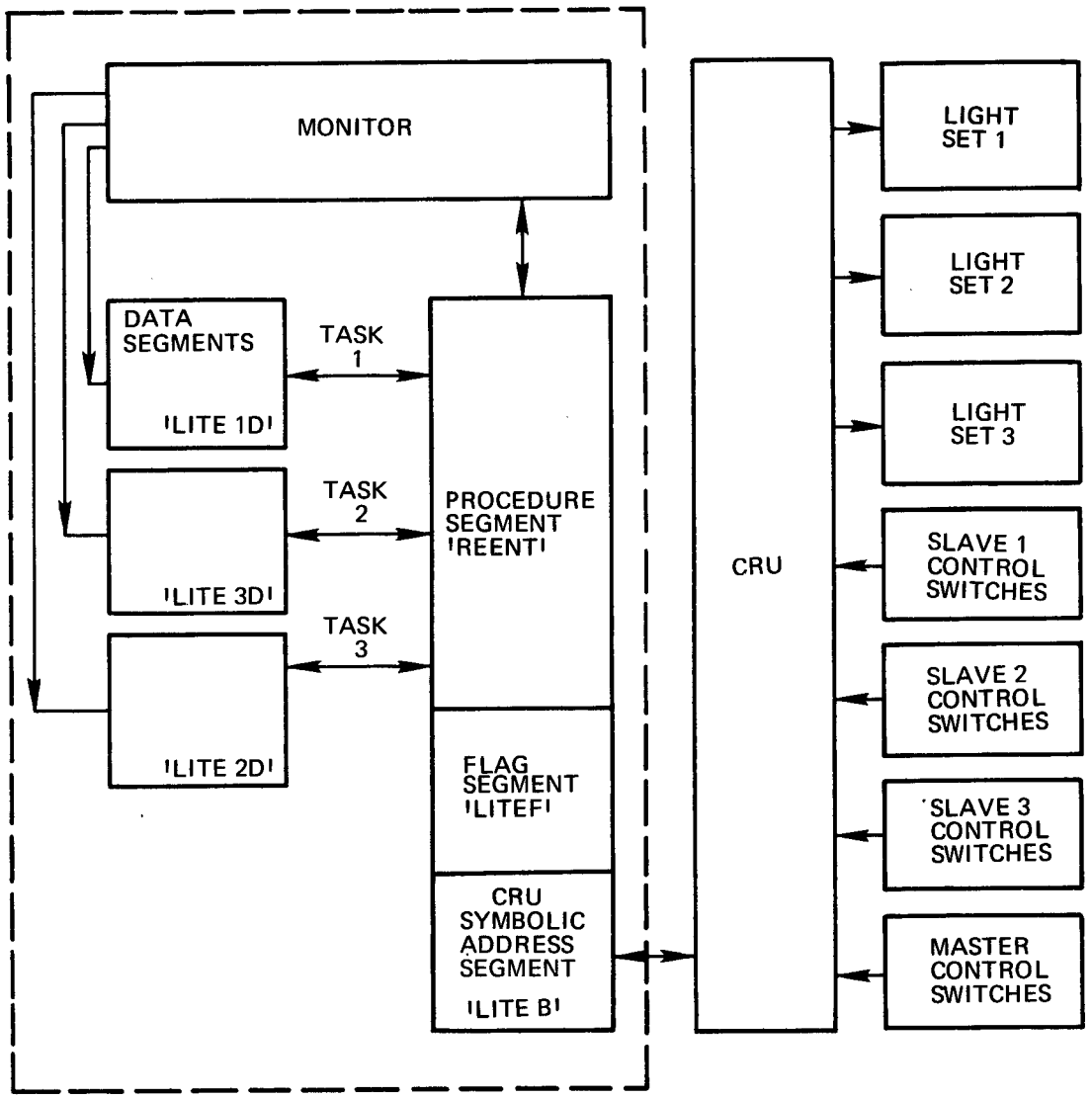


Figure 2-5. General Program Structure

ASCII Code Hexadecimal	Hex Digit
60	0
61	1
62	2
63	3
64	4
65	5
66	6
67	7
68	8
69	9
6A	A
6B	B
6C	C
6D	D
6E	E
6F	F

On cards each column represents one 8-bit character that is decoded according to the following list.

BINARY INTERNAL CODE TO
BINARY CARD CODE CONVERSION

Most Significant Digit	COMBINATION 12-11-0-9	Least Significant Digit	COMBINATION 8-(1,2,...,7)
0	blank	0	blank
1	9	1	1
2	0	2	2
3	0-9	3	3
4	11	4	4
5	11-9	5	5
6	11-0	6	6
7	11-0-9	7	7
8	12	8	8
9	12-9	9	8-1
A	12-0	A	8-2
B	12-0-9	B	8-3
C	12-11	C	8-4
D	12-11-9	D	8-5
E	12-11-0	E	8-6
F	12-11-0-9	F	8-7

Example – The binary card character for X 'CA' is 12-11-8-2.

All binary formats have several common features. The binary record is indicated by 17_{16} code in the first two frames. The two-bit record indicator code is defined for the four different types of binary records in the following table:

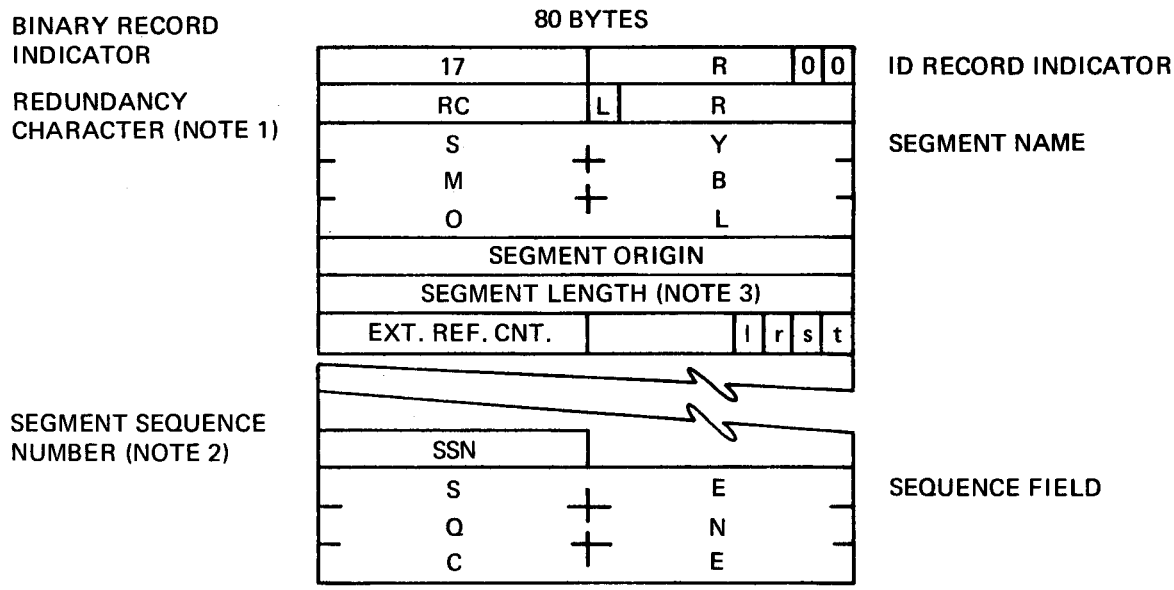
Binary Code	Record Type
00	Identification (ID) Record
11	Linkage Data (LD) or External Symbol Record
10	Text Record
01	End Record

The redundancy character is the sum modulo 256 of all bits equal to one contained within the record excluding the redundancy character itself.

The segment sequence number is increased by one for every new segment defined within any assembly containing multiple segments.

2.10 OBJECT FORMAT DEFINITION.

2-10.1 PROGRAM OR PROGRAM SEGMENT IDENTIFICATION (ID) RECORD.



NOTES

1. The redundancy character is the sum of all bits equal to one contained within the record excluding the redundancy character itself.

2. The segment sequence number is increased by one for every new segment defined within any assembly containing multiple segments.

3. The Segment Length is the value of the Segment Relative Program Location Counter when the segment is terminated.

l

0 Program Segment has been linked.
1 Linking is required – Unsatisfied References contained in Text.

r

0 Program Segment is absolute.
1 Program Segment is relocatable.

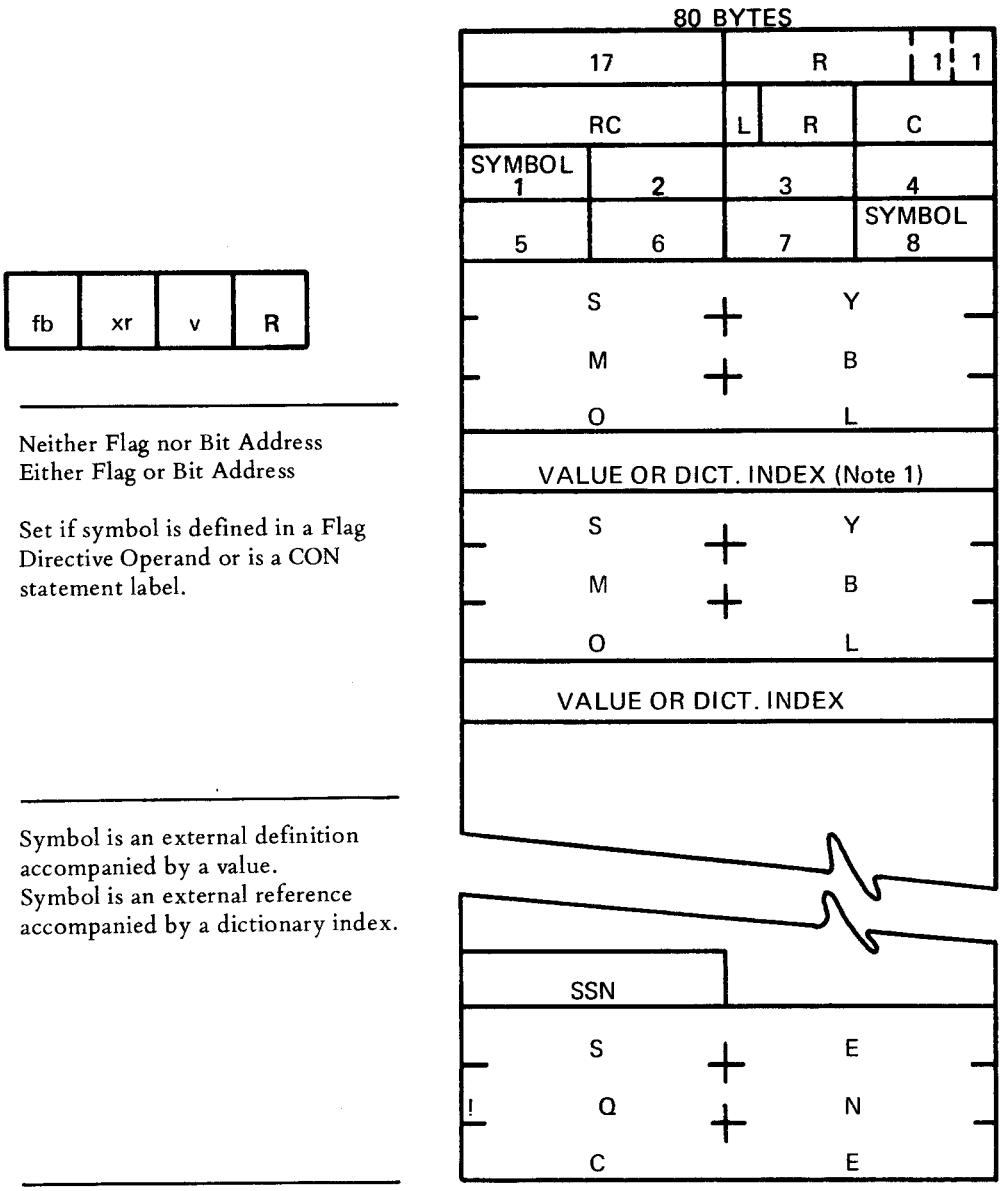
st

00 Procedure Segment
01 Data Segment
10 Flag Segment
11 CRU Symbolic Address Segment

L

0 Processed by SAL960
1 Processed by LRL

2-10.2 LINKAGE DATA (LD–EXTERNAL SYMBOL RECORD).



fb

0 Neither Flag nor Bit Address
1 Either Flag or Bit Address

Set if symbol is defined in a Flag Directive Operand or is a CON statement label.

xr

0 Symbol is an external definition accompanied by a value.
1 Symbol is an external reference accompanied by a dictionary index.

v

0 Symbol is assigned a relocatable value.
1 Symbol is assigned a self-defining value or a relative, i.e., is not relocatable.

NOTE

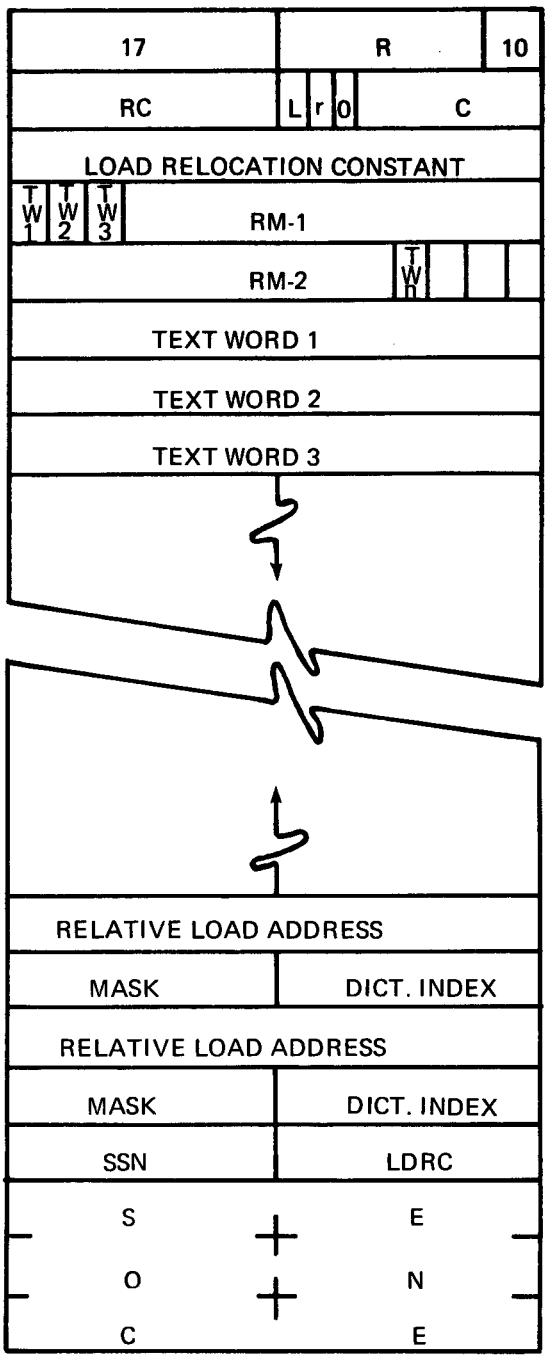
VALUE is the value assigned to an externally defined symbol. The DICTIONARY INDEX is an integer in the range 0-256 that is assigned to an external reference in the sequence in which it is declared in the program.

2-10.3 TEXT RECORD.

Binary Record Indicator
Redundancy Character
Text Word Address

Relocation Map:

Note: Relocation is required for Direct Memory Addresses only.



Text Record Indicator
Text Word Count

Text Word to be put in core by load function.

Map Bit

0 Not Relocatable
1 Relocatable

r

0 Program is absolute
1 Program is relocatable

Note 1: A MASK is composed of two hexadecimal digits.

M(1) Starting bit position of field

M(2) Field Width 0 = 16 bits

Example:

A4 - Field starts at bit 10 and is 4 bits wide.

Note 2: Special MASK values will be used to mark special cases.

Address Relative to Assembly Origin

Linkage Information for External Symbol References in Context

Linkage Data Record Count

Sequence Field

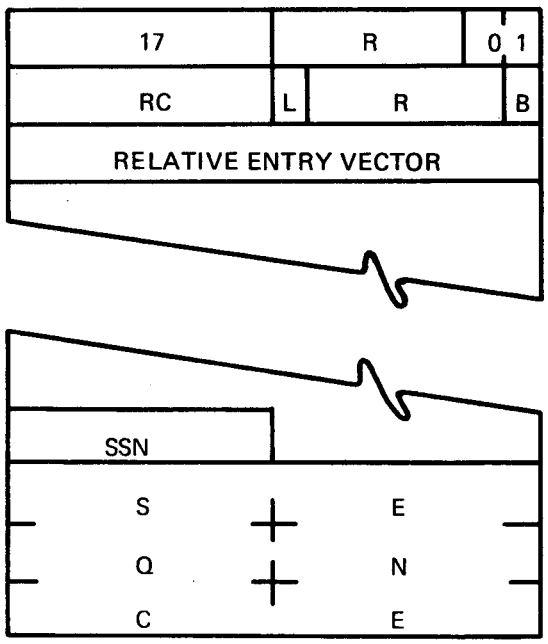
MASK Value	Meaning
FF	Flag reference
FE	CRU Bit Reference
FD	CRU Register Definition
FC	Relative Address required in Format I instruction

2-10.4 SEGMENT END RECORD.

Binary Record Indicator

Redundancy Character

Segment Sequence Number



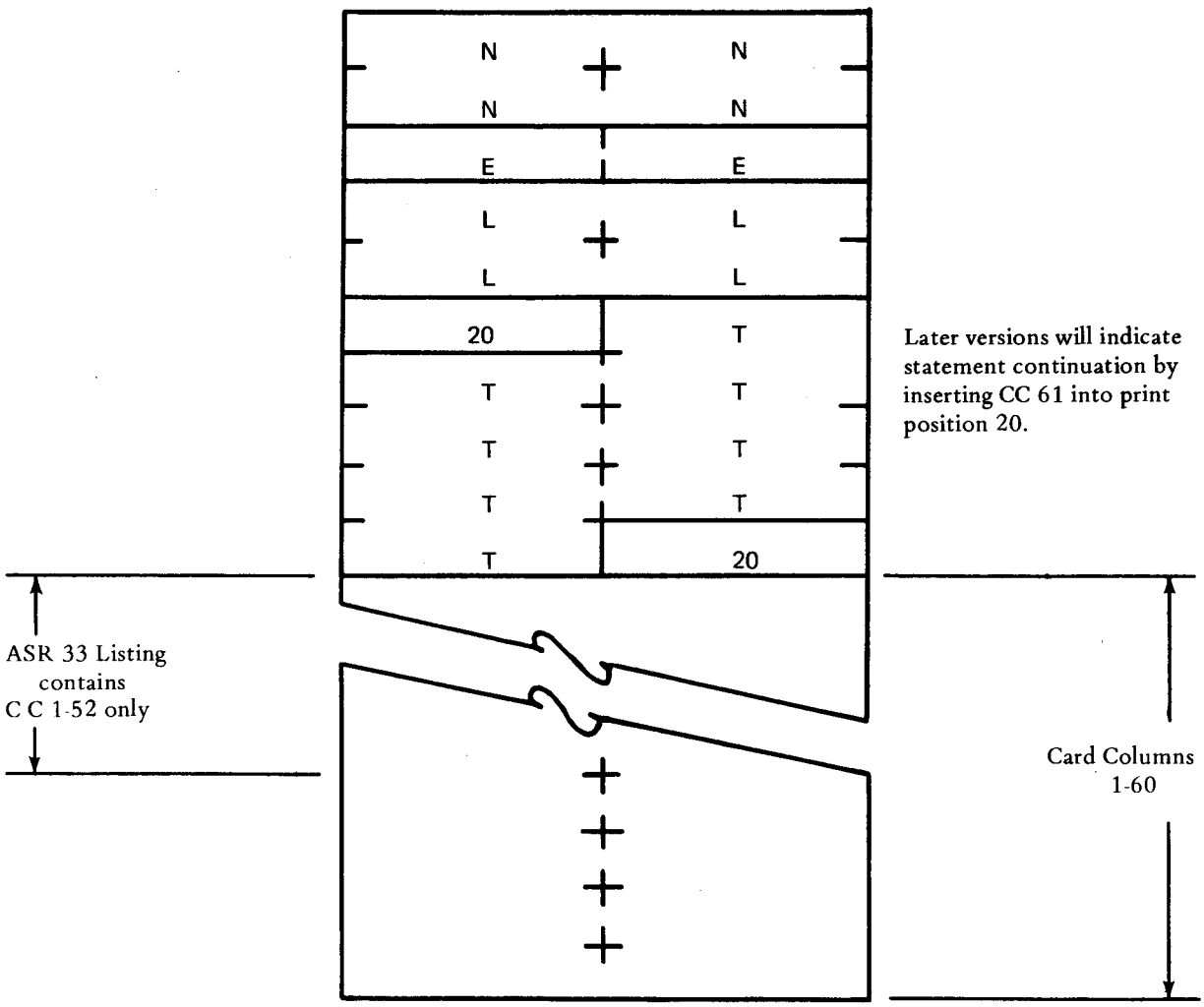
End Record Indicator

B	
0	No Branch Vector Follows
1	Branch Vector Follows

Sequence Field

R = Reserved

2-11 SAL LISTING RECORD FORMAT.



NOTE: For the 960/960 assembler with listing output on an ASR 33 lacking the punch/read without printing option the source statement will appear at the left of the page followed by the assembly data.

LEGEND:

N – Source statement line number in decimal with leading zero's suppressed.

E – Error Code.

L – Location counter in hexadecimal (Assembly Location Counter).

T – Text – Single words will be printed in 4 digits left justified and double words in 8 digits.

2-12 SAL960/960 WITH 8K MEMORY.

2-12.1 GENERAL. The 8K version of the assembler runs under control of the Programming Support Monitor. SAL960/960 is a two-pass assembler. During the first pass, the symbol, segment, and form tables are built. Presently, the assembler has the capacity to handle 100 symbols, 12 segments, and 5 forms. Since the common data block and several routines are common to Pass 1 and Pass 2, Pass 2 loading time is reduced by overlaying Pass 1 by Pass 2. Pass 2 reads a source card, processes it, and prints the object.

2-12.2 LOGICAL UNIT NUMBERS. Logical unit numbers for SAL960/960 are as follows:

LUN0		Typical Device
4	Punch ASCII (PST)	Teletypewriter
5	Read Source and Options	Card Reader
6	List	Line Printer
7	Punch Object	Card Punch

The assembler reads the source from cards, prints the object on the teletypewriter, and punches the output on paper tapes. If different devices are desired, logical unit numbers can be changed through the use of PSM job control.

2-12.3 OPTION CARDS. SAL960/960 accepts certain options specified on a card of the following format:

!*ab

where the combination ab can be one of the following:

a	b	PASS 1 FUNCTION	PASS 2 FUNCTION
L		List symbol and segment tables only.	List line number error code, APC, object, and card image only.
P		Punch the ID and LD records only	Punch text record only.
**B		No printed or punched output.	List line number error code, APC, and object only. Will not print card image. (See Note.)
L	P	List symbol and segment table and punch the ID and LD records.	List line number error code, APC, object, and card image and punch text records.
**B	P	Punch the ID and LD records	List line number error code, APC, and object and punch text records.

PROGRAMMING NOTE

B is used if input is on paper tape since the teletype prints as it reads. In the case of tapes, the output will appear as follows:

```
START LA 5,FLAG1

12 A0002 44850000
```

If no options are specified on the option card or no option card is found, the options LP are set. The option card will be recognized anywhere in the deck by Pass 1 but will be ignored by Pass 2.

2-12.4 LOADING AND RUNNING. After PSM is loaded and started:

a. Get Pass 1 ready to load and ready the load device and enter the letter L on the keyboard. Pass 1 loads.

b. Pass 1 execution is started by typing an X on the teletype.

c. When the assembler is ready to output the ID and LD records and, if a punch option is given, it will print the message

TURN ON OBJECT OUTPUT DEVICE

Turn on the teletypewriter and wait for a !*ON record to be read.

d. When Pass 1 is finished, the supervisor waits until one of the following characters is entered on the keyboard:

L If Pass 2 or Pass 2 option change deck is to be loaded.

X If Pass 1 is to be repeated.

If X is requested, repeat steps b-d again, since Pass 1 is designed to initialize itself without having to be loaded again.

e. If the options in Pass 1 are to be changed before running Pass 2, the letter L is entered on the keyboard and the Pass 2 Option Change program is loaded. The program is started by entering an X. The option record is input immediately. If something other than a !* record is read, the program says so and waits for another record. If no options are given, the options from Pass 1 will be retained and the program then branches to the supervisor, which in turn waits for the letter L to start loading in Pass 2.

f. If the Pass 2 Option Change program is not used, Pass 1 options will be retained and Pass 2 can be loaded by entering the letter L on the keyboard.

g. Pass 2 is started by entering an X. Pass 2 reads the first source record immediately.

h. If either the symbol or segment table is exceeded during Pass 1, the assembler prints out the symbol that caused the overflow and branches to the supervisor to wait for an L.

i. At the finish of Pass 2, the assembler branches to the supervisor to wait for an L or X.

j. The Option Change Deck can now be loaded and the options changed again or Pass 2 can be rerun. In either case, the tables from Pass 1 will not be destroyed.

2-13 SAL960/360 DOS.

2-13.1 GENERAL. SAL960/360 DOS is catalogued on the core image library and optionally in subprogram form on a relocatable library. The DOS system file extent for SYSLNK is used as an intermediate storage file (SYS007). During Pass 1, the source input file (SYS004) is copied to SYS007. SYS007 is the input for Pass 2. Pass 1 also outputs to the object file (SYSPCH) and the list file (SYS005). During Pass 2, the list file (SYS005) and the object file (SYSPCH) are output.

2-13.2 JOB CONTROL. Job control cards for a typical SAL960/360 DOS system are:

```
// JOB E7550871 MA PR 04 003   S960A MULTIPLY
                                          MARK II
// ASSGN SYS004,X'002' CARD READER FORTRAN
                                          UNIT7
// ASSGN SYS005,X'004' PRINTER FORTRAN UNIT 8
// ASSGN SYSPCH,X'006' CARD PUNCH FORTRAN
                                          UNIT 2
// ASSGN SYS007,X'135' DISC BULK STORAGE
// DLBL IJSYS07,'IJSYSLN'
// EXTENT SYS007,000135,1,0,3760,200
// EXEC SAL960
***SOURCE DECK***
/*
/&
```

This job can be entered into the IBM System/360 background control stream. SAL960/360 averages an assembly speed of 300 cards per minute on the 360/50. Twenty segments and 500 symbols are accommodated by SAL960/360 DOS.

2-14 SAL960/360 OS.

2-14.1 GENERAL. SAL960/360 OS is catalogued on the job library AG.ASS960. The disc area declared in the //G0.FT10F001 statement is used as an intermediate storage file. During Pass 1, the source input data set (FT05F005) is copied to FT10F001. FT10F001 is the input for Pass 2. Pass 1 also outputs to the object data set (FT07F001) and the list data set (FT08F001). During Pass 2, the list data set (FT08F001) and the object data set (FT07F001) are output.

2-14.2 JOB CONTROL. Job control cards for a typical SAL960/360 OS system are:

```
//SAL960 JOB
//JOBLIB          DD DSNAME=AG.ASS960,DISP=SHR
//SAL960   EXEC FORTGO,PARM='SAL960'
//GO.FT07F001     DD SYSOUT=C
//GO.FT10F001     DD UNIT=SPACE, SPACE=(CYL,(5,1)),
                     DCB=(RECFM=FB, BLKSIZE=80)
//GO.FT08F001     DD SYSOUT=A
//GO.FT05F001     DD *
/*
//
```

This job can be entered into the IBM system/360 through either the main terminal or any remote batch terminal connected through data lines.

SAL960/360 OS averages an assembly speed of 900 cards per minute on the 360/65. Thirty segments and 500 symbols are accommodated by SAL960/360 OS.

2-15 RELOCATABLE SAL960/PAM.

The following are the necessary job control cards to load and execute the Assembler beginning at location X'0100' under control pf PAM. This job stream will allow sequential loading and execution of Pass 1 and Pass 2. Re-execution of a pass will require user alteration of the job stream.

The deck provided is as follows:

```
!
$$LDTS**0100**0050    LOAD PASS ONE  ID=X'0050'
/*
   (pass 1 object is inserted here)
/*
$$INST**0050**00F0    INSTALL   PRIORITY=X'00F0'
$$ABLE**0050          ENABLE PASS ONE
$$DFI0**0004**0008    SYMBOL TABLE OUTPUT
$$DFI0**0005**0004    SOURCE INPUT
$$DFI0**0006**0005    LIST OUTPUT
$$DFI0**0007**0006    OBJECT OUTPUT
$$EXCT**0050          EXECUTE PASS ONE
```

(Insert source deck to be assembled along with Assembler option and control cards here)

PASS TWO

```
$$DLTS**0050          DELETE PASS ONE
$$LDTS**0100**0051    LOAD PASS TWO  ID=X'0051'
/*
   (pass 2 object is inserted here)
/*
$$INST**0051**00F0    INSTALL   PRIORITY=X'00F0'
$$ABLE**0051          ENABLE PASS TWO
$$EXCT**0051          EXECUTE PASS TWO
```

(Insert source deck to be assembled here)

CLEAN UP

The following data should be read through following execution of pass 2 in order to release logical units and delete the Assembler.

```
$$DLTS**0051  CLEAN UP    DELETE PASS TWO
$$RLI0**0004              RELEASE
$$RLI0**0005
$$RLI0**0006
$$RLI0**0007
$$JC0F**      END OF JOB
```

2-16 EXAMPLE SAL PROGRAM.

```
     SYMBOL TABLE DUMP                                          LIMSYM=   28
SYMBOL  SRLADI SSN FLAG REF RELOC DEFINED  EXT MULT

REENT    0000   1  F   F     T       T      T    F
LITFIG   0001   1  F   T     F       T      T    F
R7SAVE   0002   1  F   T     F       T      T    F
R7MAST   0003   1  F   T     F       T      T    F
R7SUB    0004   1  F   T     F       T      T    F
PATRN1   0005   1  F   T     F       T      T    F
PATRN2   0006   1  F   T     F       T      T    F
PATRN3   0007   1  F   T     F       T      T    F
PATRN4   0008   1  F   T     F       T      T    F
BIOTSK   0009   1  F   T     F       T      T    F
TTYPRB   000A   1  F   T     F       T      T    F
SUPER    007F   1  F   F     F       T      F    F
CRUREG   0000   3  T   F     F       T      T    F
CONTRL   0001   3  T   F     F       T      T    F
INDENT   0024   1  F   F     T       T      F    F
LINE1    0011   3  T   F     F       T      T    F
SW1      0018   1  F   F     T       T      F    F
LINE2    0021   3  T   F     F       T      T    F
SW2      001C   1  F   F     T       T      F    F
LINE3    0031   3  T   F     F       T      T    F
SW3      0020   1  F   F     T       T      F    F
LINE4    0041   3  T   F     F       T      T    F
ERROR    0026   1  F   F     T       T      F    F
IO       0000   2  T   F     F       T      T    F
EXIT     002C   1  F   F     T       T      F    F
LITEF    0036   2  F   F     T       T      T    F
DROP     0001   2  T   F     F       T      T    F
LITER    0000   3  F   F     F       T      T    F

     SEGMENT TABLE DUMP                                         LIMSEG=    3
  SEGNAM RELBIAS LENGTH REFCNT SSN LINK ABS SEGTYP

  REENT   0000     54     10    1   T   F   T   T
  LITEF   0036      0      0    2   T   F   F   T
  LITER   0000      0      0    3   T   F   F   F

** UNREFERENCED SYMBOL **  REENT

** UNREFERENCED SYMBOL **  LITEF

**    REFERENCED SYMBOL **  DROP

**    REFERENCED SYMBOL **  LITER

**     ASS 1 ERRORS **
```

```
SEQ FR LC   OBJECT                          SOURCE
  1  0000            REENT RSEG
  2                  *
  3                  * THIS EXAMPLE PROGRAM ACCEPTS CONTROL INPUT FROM 3 SETS OF
  4                  * 16 SLAVE SWITCHES AND 1 SET OF 16 MASTER SWITCHES AND
  5                  * PROVIDES 3 LEVELS OF CONTROL FOR THE OUTPUT OF OPTIONAL
  6                  * DEMONSTRATION PATTERNS ON EACH OF THE DEMONSTRATION
  7                  * LIGHT SETS. EACH LIGHT SET CONTAINS 16 LIGHTS WHOSE ON/OFF
  8                  * STATES CORRESPOND DIRECTLY TO 16 SLAVE SWITCHES WHEN
  9                  * INDEPENDENT OPERATION IS INDICATED BY MASTER CONTROL
 10                  * SWITCH 0'S OFF STATE. THE ON STATE OF THIS SWITCH
 11                  * INDICATES THAT EACH OF THE OUTPUTS TO THE LIGHT SETS ARE
 12                  * SLAVED TO 5 OF THE 16 MASTER SWITCHES. SETS 1,2,AND 3 ARE
 13                  * SLAVED TO MASTER SWITCHES 1-5,6-A, AND B-F RESPECTIVELY.
 14                  * IF THE FIRST SWITCH OF ANY 5 SWITCH GROUP IS OFF THE
 15                  * INDEPENDENT MODE OF OPERATION WILL BE IN EFFECT FOR THE
 16                  * CORRESPONDING LIGHT SET ONLY. IF THIS FIRST SWITCH IS ON,
 17                  * THE PROGRAM WILL OUTPUT ONE OF 4 PREDETERMINED LIGHT
 18                  * PATTERNS INDICATED BY THE HIGHEST PRIORITY(LOWEST NO.)
 19                  * SWITCH IN THE ON STATE WITHIN ITS GROUP.
 20                  *
 21                  **** RSEG EXECUTION BEGINS WITH THE CRU BASE REGISTER
 22                  * INITIALIZED WITH THE CRU BASE ADDRESS FOUND IN THE CURRENT
 23                  * WORKER TASK BLOCK ****
 24                  *
 25  0000                    REF  LITEIG,R7SAVE,R7MAST
 26  0000                    REF  R7SUB,PATRN1,PATRN2
 27  0000                    REF  PATRN3,PATRN4
 28  0000                    REF  BIDTSK,TTYPRB
 29  0000            SUPER   EQU  X'7E'
 30                  *
 31                  ****     INPUT SLAVE UNIT SWITCH CONFIGURATION ****
 32                  *
 33  0000 20200000           STCR CRUREG,LITEIG
 34  0002 4AC70000           ST   7,@R7SAVE,4              SAVE REGISTER 7
 35                  *
 36                  **** THE CRU BASE REGISTER IS CHANGED TO
 37                  * ACCESS THAT PORTION OF THE CRU CONNECTED TO THE MASTER
 38                  * SWITCHES ****
 39                  *
 40  0004 46470000           L    7,@R7MAST,4              MASTER BSEG BASE
 41  0006 30000824           BBNE CONTRL,1,INDENT          BRANCH FOR INDEP. MODE
 42                  *
 43                  **** THE CRU BASE REGISTER IS CHANGED TO
 44                  *ACCESS THAT PORTION OF THE CRU CONNECTED TO THE APPROPRIATE
 45                  *MASTER SWITCH GROUP ****
 46                  *
 47  0008 46470000           L    7,@R7SUB,4               MASTER GROUP BSEG BASE
 48  000A 30000824           BBNE CONTRL,1,INDENT
 49  000C 30010018           BBNE LINE1,0,SW1              DETERMINE SUBSTITUTE
 50  000E 3002001C           BBNE LINE2,0,SW2              LIGHT PATTERN.
 51  0010 30030020           BBNE LINE3,0,SW3
 52  0012 30040826           BBNE LINE4,1,ERROR            NO PATTERN,INDICATE ERROR
 53  0014 46400000           L    0,@PATRN4,4
 54  0016 70820022           B    INDENT-2
```

```
0018 46400000 SW1     L    ,0,@PATRN1,4
001A 70820022         B     INDENT-2
001C 46400000 SW2     L     0,@PATRN2,4
001E 70820022         B     INDENT-2
0020 46400000 SW3     L     0,@PATRN3,4
                      *
                      **** STORE SELECTED LIGHT PATTERN IN OUTPUT LOCATION ****
                      *
0022 4AC00000         ST    0,@LITEIG,4
0024 464700CC INDENT  L     7,@R7SAVE,4              RESTORE REGISTER 7
0026 84000820 ERROR   BENE  IO,1,EXIT
0028 44830000         LA    3,@TTYPRB                SHOULD ERROR BE IGNORED
002A 7980007F         SXBS  *SUPER                   NO,OUTPUT ERROR COMMENT
002C 08000000 EXIT    LOCB  CRUREG,LITEIG            OUTPUT LIGHT PATTERN
002E 46430000         L     3,@PIDTSK,4              GO TO NEXT TASK VIA
0030 7980007F         SXBS  *SUPER                   MONITOR BID ROUTINE
0032 44830400         LA    3,X'400'                 END CURRENT TASK
0034 7980007F         SXBS  *SUPER                   VIA MONITOR EOP ROUTINE
0036                  END

0036          LITEF   FSEG
              *
              **** THE FSEG DEFINES FLAG LABELS RELATIVE TO THE FLAG BASE
              * ADDRESS WHICH DIFFERS FOR EACH TASK.THE FLAGS ARE INITAL-
              *IZED IN THE TASK DATA SEGMENT BEGINNING AT THE BASE
              *ADDRESS ****
              *
0000                  FLAG IO,DROP,14
0036                  END

0000          LITER   RSEG  0
              *
              **** THE RSEG BASE ADDRESS DIFFERS FOR EACH TASK.DURING THE
              * TASK EXECUTION,THE BASE ADDRESS IS CHANGED FROM THE
              * INITIAL SLAVE SWITCH BASE TO THE MASTER SWITCH BASE AND
              * MASTER GROUP SWITCH BASE BY USING DISPLACEMENT OPERANDS
              * WITH GROUP 1 INSTRUCTIONS RELATIVE TO THE APPROPRIATE
              * DATA BASE ADDRESS. ****
              *
0000          CRUREG  CON   0,16
0001          CONTRL  CON   0
0011          LINE1   CON   1
0021          LINE2   CON   2
0031          LINE3   CON   3
0041          LINE4   CON   4
0036                  END
```

```
     SYMBOL TABLE DUMP                                          LIMSYM=   14
SYMBOL  SRLADT SSN FLAG REF RELOC DEFINED  EXT MULT

LITE1D    0000   1  F   F     T       T      T    F
REENT     0001   1  F   T     F       T      T    F
LITE1F    0024   1  F   F     T       T      T    F
LITFIG    0010   1  F   F     T       T      T    F
R7SAVE    0011   1  F   F     T       T      T    F
R7MAST    0012   1  F   F     T       T      T    F
R7SUB     0013   1  F   F     T       T      T    F
PATRN1    0014   1  F   F     T       T      T    F
PATRN2    0015   1  F   F     T       T      T    F
PATRN3    0016   1  F   F     T       T      T    F
PATRN4    0017   1  F   F     T       T      T    F
BIDTSK    0018   1  F   F     T       T      T    F
TTYPRB    0019   1  F   F     T       T      T    F
TTYBUF    001E   1  F   F     T       T      T    F
```

```
     SEGMENT TABLE DUMP                                         LIMSEG=   1
SEGNAM REBIAS LENGTH REFCNT SSN LINK ABS SEGTYP

LITE1D  0000     37      1   1   T   F   T  F

** UNREFERENCED SYMBOL **  LITFIG

** UNREFERENCED SYMBOL **  R7SAVE

** UNREFERENCED SYMBOL **  R7MAST

** UNREFERENCED SYMBOL **  R7SUB

** UNREFERENCED SYMBOL **  PATRN1

** UNREFERENCED SYMBOL **  PATRN2

** UNREFERENCED SYMBOL **  PATRN3

** UNREFERENCED SYMBOL **  PATRN4

** UNREFERENCED SYMBOL **  BIDTSK

** UNREFERENCED SYMBOL **  TTYPRB

** NO PASS 1 ERRORS **
```

```
SEQ ER LC    OBJECT                                          SOURCE
  1   0000            LITE1D DSEG                                FIRST LIGHT SET DATA SEGMENT
  2   0000                   REF   REENT
  3                   **** THE WORKER TASK BLOCK CONSISTS OF THE FIRST 16
  4                   * LOCATIONS ****
  5   0000 0000              DATA REENT,X'8000'                  EVENT COUNTER AND STATUS
      0001 8000
  6   0002 000B              DATA  11                            FLAGS/PRIORITY
  7   0003 0000              DATA  0,0,0,0                       WORKER REGISTERS 0 THROUGH 3
      0004 0000
      0005 0000
      0006 0000
  8   0007 0000              DATA  LITE1D                        DATA BASE ADDRESS,REGISTER 4
  9   0008 0000              DATA  REENT                         PROCEDURE BASE ADDRESS,REG.5
 10   0009 0024              DATA  LITE1F                        FLAG BASE ADDRESS,REGISTER 6
 11   000A 0030              DATA  X'30'                         CRU BASE ADDRESS,REGISTER 7
 12   000B 8000              DATA  X'8000'                       INITIAL STATUS WORD
 13   000C 0000              DATA  REENT                         ENTRY POINT
 14   000D 0000              DATA  0,0,0                         TIMER,LINK,IDENTIFICATION .
      000E 0000
      000F 0000
 15   0010 0000       LITE1G DATA  0                             INITIALIZE PATTERN STORAGE
 16   0011 0000       R7SAVE DATA  0                             REGISTER 7 STORAGE .
 17   0012 0060       R7MAST DATA  X'60'                         CRU MASTER CONTROL BASE .
 18   0013 0061       R7SUB  DATA  X'61'                         CRU SUB-CONTROL BASE .
 19   0014 0011       PATRN1 DATA  X'11'                         MISCELLANEOUS LIGHT PATTERNS
 20   0015 0012       PATRN2 DATA  X'12'                         AVAILABLE FOR OUTPUT TO SET 1
 21   0016 0013       PATRN3 DATA  X'13'                         UNDER MASTER GROUP CONTROL.
 22   0017 0014       PATRN4 DATA  X'14'
 23   0018 000C       BIDTSK DATA  12                            NEXT TASK TO BE EXECUTED
 24   0019 0000       TTYPRB DATA  LITE1D                        WORKER TASK BOLCK ADDRESS
 25   001A 001E              DATA  TTYBUF                        BUFFER ADDRESS
 26   001B 000B              DATA  11                            BUFFER LENGTH IN CHARACTERS
 27   001C 000B              DATA  11                            RECORD LENGTH IN CHARACTERS
 28   001D 0012              DATA  X'0012'                       FLAGS AND LOGICAL UNIT NUMBER
 29   001E 4552522C   TTYBUF DATA  C'ERR,GROUP 1'
      0020 47524F55
      0022 50203120
 30   0024 8000       LITE1F DATA  X'8000'
 31   0025                   END
```

```
     SYMBOL TABLE DUMP                                          LIMSYM=   14
SYMBOL  SRLADT SSN FLAG REF RELOC DEFINED  EXT MULT

LITE2D     0000   1  F   F     T       T      T    F
REENT      0001   1  F   T     F       T      T    F
LITE2F     0024   1  F   F     T       T      T    F
LITEIG     0010   1  F   F     T       T      T    F
R7SAVE     0011   1  F   F     T       T      T    F
R7MAST     0012   1  F   F     T       T      T    F
R7SUB      0013   1  F   F     T       T      T    F
PATRN1     0014   1  F   F     T       T      T    F
PATRN2     0015   1  F   F     T       T      T    F
PATRN3     0016   1  F   F     T       T      T    F
PATRN4     0017   1  F   F     T       T      T    F
BIDTSK     0018   1  F   F     T       T      T    F
TTYPRB     0019   1  F   F     T       T      T    F
TTYBUF     001F   1  F   F     T       T      T    F
```

```
     SEGMENT TABLE DUMP                                         LIMSEG=   1
SEGNAM RELBIAS LENGTH REFCNT SSN LINK ABS SEGTYP

LITE2D  0000     37      1    1    T   F   T  F

** UNREFERENCED SYMBOL **  LITEIG

** UNREFERENCED SYMBOL **  R7SAVE

** UNREFERENCED SYMBOL **  R7MAST

** UNREFERENCED SYMBOL **  R7SUB

** UNREFERENCED SYMBOL **  PATRN1

** UNREFERENCED SYMBOL **  PATRN2

** UNREFERENCED SYMBOL **  PATRN3

** UNREFERENCED SYMBOL **  PATRN4

** UNREFERENCED SYMBOL **  BIDTSK

** UNREFERENCED SYMBOL **  TTYPRB

** NO PASS 1 ERRORS **
```

```
SEQ  ER LC   OBJECT                          SOURCE
  1      0000          LITE2D DSEG
  2                    *
  3                    **** THIS DSEG IS IDENTICAL TO THE FIRST IN THAT DATA VALUES
  4                    * HAVING THE SAME DISPLACEMENT RELATIVE TO THE DSEG BASE
  5                    * WILL SERVE IDENTICAL LOGIC PURPOSES. THE ACTUAL VALUES
  6                    * MAY OR MAY NOT BE THE SAME . ****
  7                    *
  8      0000                 REF  REENT
  9      0000 0000            DATA REENT,X'8000'
         0001 8000
 10      0002 000C            DATA 12
 11      0003 0000            DATA 0,0,0,0
         0004 0000
         0005 0000
         0006 0000
 12      0007 0000            DATA LITE2D
 13      0008 0000            DATA REENT
 14      0009 0024            DATA LITE2F
 15      000A 0040            DATA X'40'
 16      000B 8000            DATA X'8000'
 17      000C 0000            DATA REENT
 18      000D 0000            DATA 0,0,0
         000E 0000
         000F 0000
 19      0010 0000      LITEIG DATA 0
 20      0011 0000      R7SAVE DATA 0
 21      0012 0060      R7MAST DATA X'60'
 22      0013 0066      R7SUB  DATA X'66'
 23      0014 0021      PATRN1 DATA X'21'
 24      0015 0022      PATRN2 DATA X'22'
 25      0016 0023      PATRN3 DATA X'23'
 26      0017 0024      PATRN4 DATA X'24'
 27      0018 000D      RIDTSK DATA 13
 28      0019 0000      TTYPRB DATA LITE2D
 29      001A 001F            DATA TTYBUF
 30      001B 000B            DATA 11
 31      001C 000B            DATA 11
 32      001D 0012            DATA X'0012'
 33      001E 4552522C TTYBUF DATA C'ERR,GROUP 2'
         0020 43524F55
         0022 50203220
 34      0024 0000      LITE2F DATA 0
 35      0025                 END
```

```
     SYMBOL TABLE DUMP                                          LIMSYM=   14
SYMBOL  SRLADT SSN FLAG REF RELOC DEFINED  EXT MULT

LITE3D     0000   1  F   F     T       T      T    F
REENT      0001   1  F   T     F       T      T    F
LITE3F     0024   1  F   F     T       T      T    F
LITFIG     0010   1  F   F     T       T      T    F
R7SAVE     0011   1  F   F     T       T      T    F
R7MAST     0012   1  F   F     T       T      T    F
R7SUB      0013   1  F   F     T       T      T    F
PATRN1     0014   1  F   F     T       T      T    F
PATRN2     0015   1  F   F     T       T      T    F
PATRN3     0016   1  F   F     T       T      T    F
PATRN4     0017   1  F   F     T       T      T    F
BIDTSK     0018   1  F   F     T       T      T    F
TTYPRB     0019   1  F   F     T       T      T    F
TTYBUF     001E   1  F   F     T       T      T    F
```

```
     SEGMENT TABLE DUMP                                         LIMSEG=    1
SEGNAM RELBIAS LENGTH REFCNT SSN LINK ABS SEGTYP

LITE3D  0000     37      1    1    T   F    T  F

** UNREFERENCED SYMBOL **  LITFIG

** UNREFERENCED SYMBOL **  R7SAVE

** UNREFERENCED SYMBOL **  R7MAST

** UNREFERENCED SYMBOL **  R7SUB

** UNREFERENCED SYMBOL **  PATRN1

** UNREFERENCED SYMBOL **  PATRN2

** UNREFERENCED SYMBOL **  PATRN3

** UNREFERENCED SYMBOL **  PATRN4

** UNREFERENCED SYMBOL **  BIDTSK

** UNREFERENCED SYMBOL **  TTYPRB

** NO PASS 1 ERRORS **
```

```
SEQ ER LC   OBJECT                          SOURCE
  1  0000           LITE3D DSEG
  2                 *
  3                 **** THIS DSEG IS IDENTICAL TO THE FIRST IN THAT DATA VALUE
  4                 * HAVING THE SAME DISPLACEMENT RELATIVE TO THE DSEG BASE
  5                 * WILL SERVE IDENTICAL LOGIC PURPOSES. THE ACTUAL VALUES
  6                 * MAY OR MAY NOT BE THE SAME . ****
  7                 *
  8  0000                   REF  REENT
  9  0000 0000              DATA REENT,X'8000'
     0001 8000
 10  0002 0000              DATA 13
 11  0003 0000              DATA 0,0,0,0
     0004 0000
     0005 0000
     0006 0000
 12  0007 0000              DATA LITE3D
 13  0008 0000              DATA REENT
 14  0009 0024              DATA LITE3F
 15  000A 0050              DATA X'50'
 16  000B 8000              DATA X'8000'
 17  000C 0000              DATA REENT
 18  000D 0000              DATA 0,0,0
     000E 0000
     000F 0000
 19  0010 0000      LITFIG  DATA 0
 20  0011 0000      R7SAVE  DATA 0
 21  0012 0060      P7MAST  DATA X'60'
 22  0013 006B      R7SUB   DATA X'6B'
 23  0014 0031      PATRN1  DATA X'31'
 24  0015 0032      PATRN2  DATA X'32'
 25  0016 0033      PATRN3  DATA X'33'
 26  0017 0034      PATRN4  DATA X'34'
 27  0018 000B      BIDTSK  DATA 11
 28  0019 0000      TTYPRB  DATA LITE3D
 29  001A 001F              DATA TTYBUF
 30  001B 000B              DATA 11
 31  001C 000B              DATA 11
 32  001D 0012              DATA X'0012'
 33  001F 4552522C  TTYBUF  DATA C'ERR,GROUP 3'
     0020 47524F55
     0022 50203320
 34  0024 8000      LITE3F  DATA X'8000'
 35  0025                   END
```

```
// JOB F7550871 DC PR 14 C05 960SAL*DEC
// ASSGN SYS004,X'002'              CARD READER UNIT 7
// ASSGN SYS005,X'004'              PRINTER UNIT 8
// ASSGN SYSPCH,X'006'              PUNCH UNIT 2
// ASSGN SYS007,X'135'              DISK STORAGE UNIT 10
// DLBL IJSYS07,'IJSYSLN'           DISK LABEL
// EXTENT SYS007,000135,1,0,3760,200 DISK LIMITS
// EXEC SAL960
```

2-17 ERRORS.

The following errors will be detected and output on the listing device at the end of pass 1.

1. Multiply defined symbols.
2. Undefined symbols.
3. Symbol Table Overflow.

Errors detected during Pass 2 will be flagged on the assembly listing. Space is provided for printing two errors per statement. The error codes printed will indicate the last two errors detected.

Error Code	Error Type
D	Multiply Defined Symbols
U	Undefined Symbols
A	Address Error
B	Branch Address Error
M	Illegal attempt to specify alternate mode registers
O	Undefined Operation Code
S	Syntax Error
T	Value was truncated to fit in operand field
E	Expression Error

SECTION III

PROCESS AUTOMATION MONITOR

3-1 SCOPE.

This section describes the Process Automation Monitor PAM (PAM960). PAM is an operating system for the Texas Instruments Model 960 computer. It is a real-time, multi-programming operating system using an executive/worker method for program control and it has core resident worker tasks.

LIST OF PAM960 SOFTWARE

NAME	SOURCE				OBJECT			
					LINKABLE		LOADABLE	
	CARDS	PT	LISTING	DESC	CARDS	PT	CARDS	PT
SUPERVISOR-M (SPB)	218420	218421	218422	218544*	218424	218424		
SUPERVISOR DATA-M (SDB)								
BASIC	218570	218571	218572	218573*	218574	218575		
HSPT MEDIA	218425	218426	218427	218545*	218428	218429		
CARD MEDIA	218430		218431	218546*	218432			
CRU INTERRUPT DECODER (CINTSG)	218433	218434	218435	218547*	218436	218437		
DMAC INTERRUPT DECODER (DMACSG)	218438	218439	218440	218548*	218441	218442		
INTERNAL INTERRUPT DECODER (ITSEG)	218443	218444	218445	218549*	218446	218447		
END OF RECORD PROCESSOR (RECEOR)	218448	218449	218450	218550*	218451	218452		
PROGRAM CONTROL SUPERVISOR SERVICES (TIMDLY)	218453	218454	218455	218551*	218456	218457		
GET DATA BLOCK SUPERVISOR SERVICE (BTDBLK)	218458	218459	218460	218552*	218461	218462		
INTERVAL TIMER AND CLOCK (TIMER)	218463	218464	218465	218553*	218466	218267		
I/O COMMON SUBROUTINES (GETCO1)	218468	218469	218470	218554*	218471	218472		
DP/UT SR300 DRIVER-M (CARDIN)	218473	218474	218475	218555*	218476	218477		
DP/UT SP 120 DRIVER-M (CDP000)	218478	218479	218480	218556*	218481	218482		
DP2310 LINE PRINTER DRIVER-CRU-M (LP000)	218483	218484	218485	218557*	218486	218487		
DP2310 LINE PRINTER DRIVER-DMAC-M (LPH)	218488	218489	218490	218558*	218491	218492		
REMEX RRS 304 DRIVER (HSRSEG)	218493	218494	218495	218559*	218496	218497		
TALLY 420 PUNCH DRIVER (PTP000)	218498	218499	218500	218560*	218501	218502		
DATA TERMINAL/ASR33 DRIVER (TTYSEG)	218503	218504	218505	218561*	218506	218507		
DIAGNOSTIC TASK	218508	218509	218510	218562*	218511	218512		
JOB CONTROL TASK	218513	218514	218515	218563*	218516	218517		
ON-LINE DEBUG TASK	218518	218519	218520	218564*	218521	218522		
PAM-PT MEDIA			218538≠	218568*			218539	218540
PAM-CARD MEDIA			218541≠	218569*			218542	218543
COMPLETE TIME & DATE SUPPORT (DTDSEG)	218576	218577	218578	218579*	218580	218581		
DUMMY TIME & DATE SUPPORT (DTPSEG)	218582	218583	218584		218585	218586		
DUMMY DMAC INT. DECODER (DMACDM)	218587	218588	218589		218590	218591		

SERVICE & UTILITY TASKS

NAME	SOURCE				OBJECT			
					LINKABLE		LOADABLE	
	CARDS	PT	LISTING	DESC	CARDS	PT	CARDS	PT
TASK STATUS DISPLAY	218523	218524	218525	218565			218526	218527
LOGICAL UNIT NO ASSIGNMENT DISPLAY	219528	218529	218530	218566			218531	218532
MESSAGE WRITER	218533	218534	218534	218567			218536	218537

PAM/PSM SERVICE ROUTINES

NAME	SOURCE				OBJECT			
					LINKABLE		LOADABLE	
	CARDS	PT	LISTING	DESC	CARDS	PT	CARDS	PT
MULTIPLY (MULTPY)	218260	218261	218262	218373*	218263	218264		
DIVIDE (DIVIDE)	218265	218266	218267	218374*	218268	218269		
BINARY/DECIMAL (CBDA)	218270	218271	218272	218375*	218273	218274		
BINARY HEXADECIMAL (CBHA)	218275	218276	218277	218376*	218278	218279		
DECIMAL/BINARY/ (CDAB)	218280	218281	218282	218377*	218283	218284		
HEXADECIMAL/BIN (CHAB)	218285	218286	218287	218378*	218288	218289		
CIRCULAR LEFT DOUBLE SHIFT (SCLD)	218290	218291	218292	218379*	218293	218294		
FIXED POINT SQ. ROOT (SQRT)	218295	218296	218297	218380*	218298	218299		
FLOATING POINT ARITH (FLTSEG)	218300	218301	218302	218381*	218303	218304		
TRIGONOMETRIC FUNCTIONS (TRSEG)	218305	218306	218307	218382*	218308	218309		

PAM/PSM UTILITY TASKS

NAME	SOURCE				OBJECT			
					LINKABLE		LOADABLE	
	CARDS	PT	LISTING	DESC	CARDS	PT	CARDS	PT
UNLOAD MEMORY	218397	218392	218393	218383			218394	218395
SOURCE MAINTENANCE	218396	218350	218351	218384			218352	218353
LINKED OBJECT TAPE EDIT	218354	218355	218356	218385			218357	218358

*Included within section 4 of this document.
≠Load Map.

STANDARD LOADERS

NAME	SOURCE				OBJECT			
					LINKABLE		LOADABLE	
	CARDS	PT	LISTING	DESC	CARDS	PT	CARDS	PT
LOADER-ASR33	218249	218250	218251	218386				218252[1]
LOADER-PTM	218253	218254	218255	218387				218256[1]
LOADER-CARD MEDIA	218257		218258	218388			218259[1]	
BOOTSTRAP LOADER				218391				
GENERAL FEATURES								
PRIMITIVE LOADER ASR33	218344	218345	218346	218389				
PRIMITIVE LOADER CARD MEDIA	218347	218348	218349	218390				
LINKING RELOCATING LOADER	218620	218621	218622	218623			218624	218625

[1] Primitive Loader Format

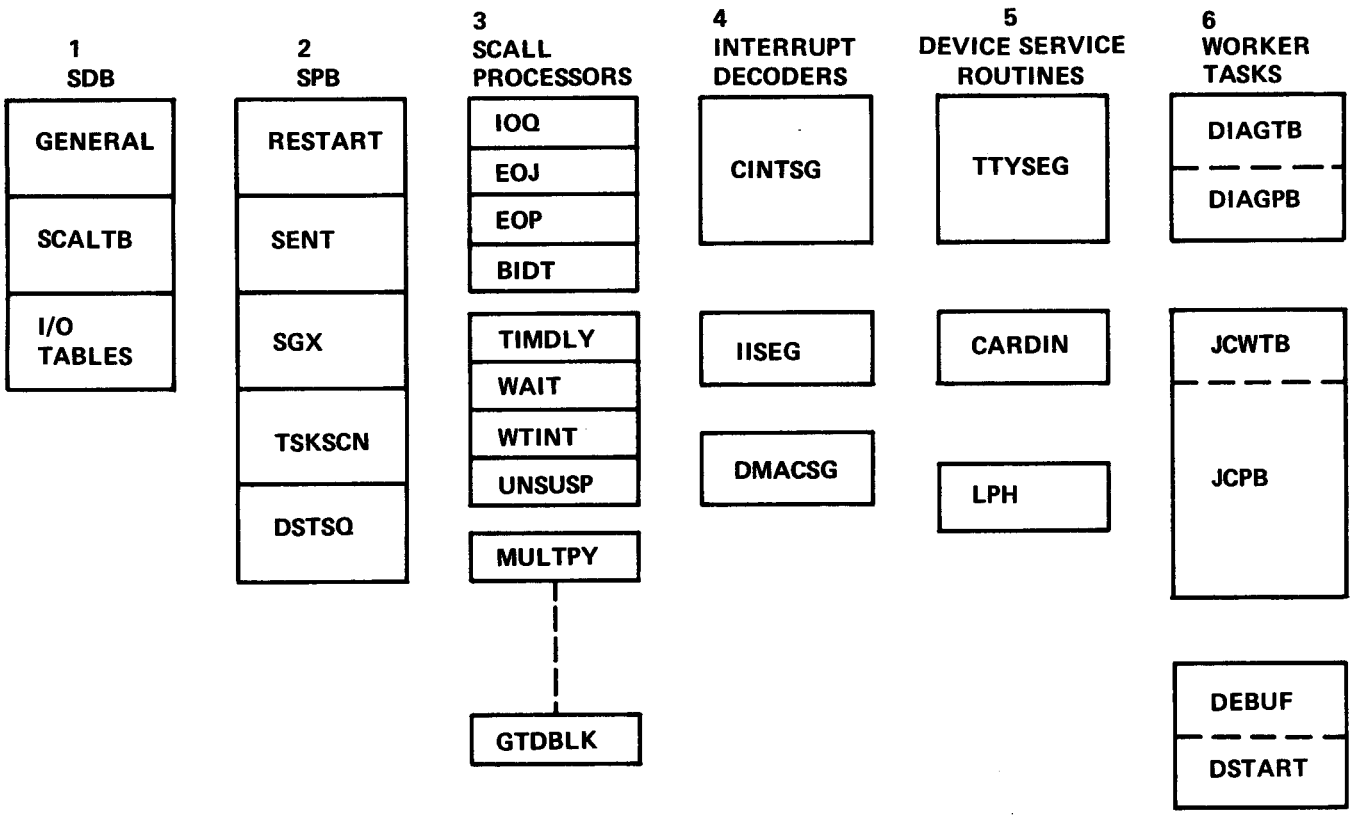


Figure 3-1 PAM960 Program Structure – Major Segments and Care Map.

3-2 PAM SYSTEM DESCRIPTION.

3-2.1. PAM PROGRAM STRUCTURE. PAM has six functional groups of programs (or data blocks). Their interrelationship is shown in Figures 3-1 and 3-2.

3-2.1.1 Supervisor Data Block. The Supervisor Data Block (SDB) contains all of the I/O tables, address tables, pointers to various tasks, interrupt status storage locations, and flags used by the rest of the supervisor. It is always the first part of the monitor in memory.

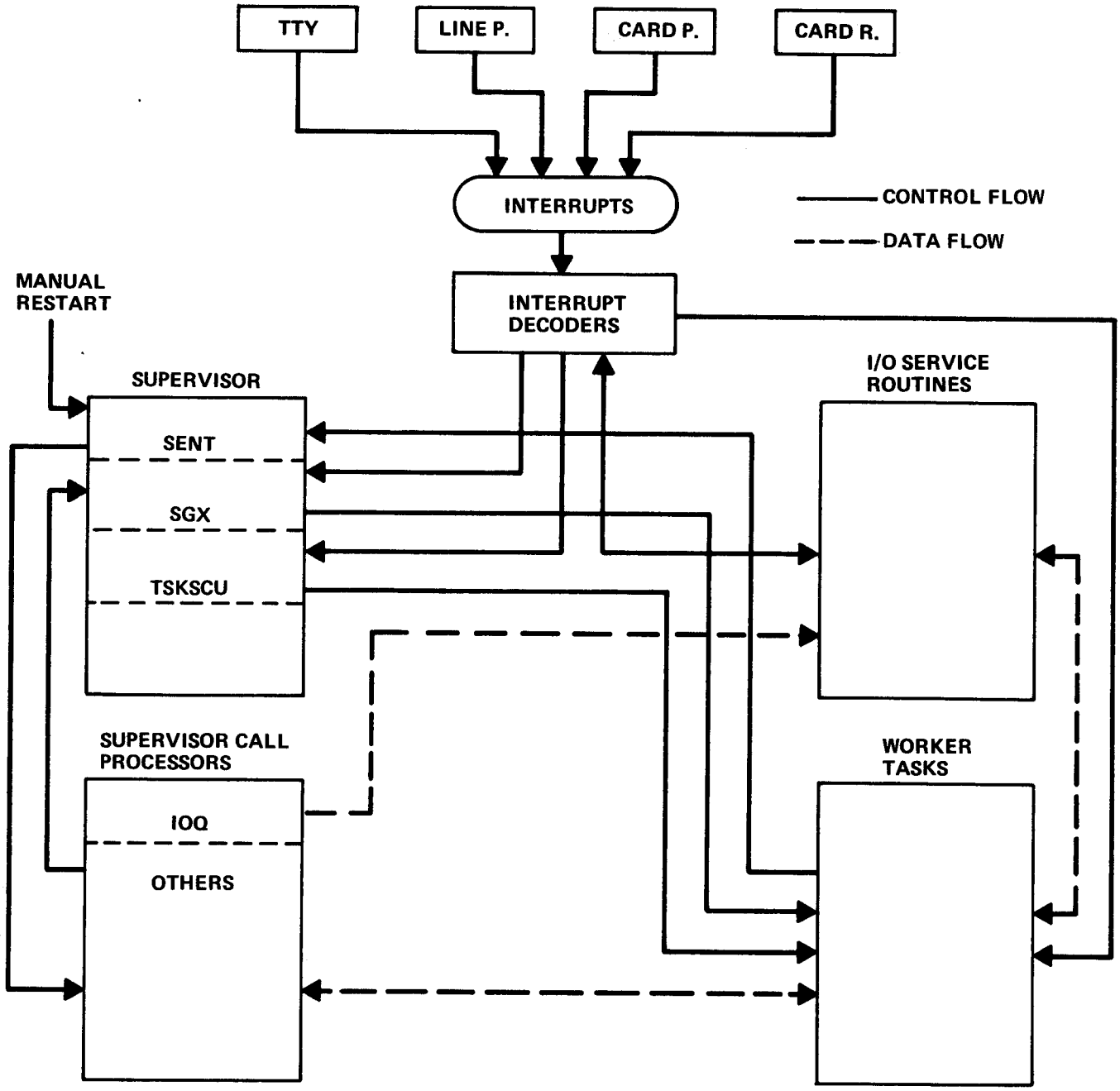


Figure 3-2 PAM960 Program Structure – Data and Control Flow.

3-2.1.2 **Supervisor Procedure Block.** The innermost part of PAM is the first section of the Supervisor Procedure Block (SPB).

a. The supervisor restart routine clears all I/O tables upon initial start-up.

b. The supervisor entry routine (SENT) decodes supervisor calls from a worker task.

c. The supervisor general exit routine (SGX) is used by all supervisor call processors to either return to the calling program or save the program's registers and enter the task scanner.

d. The task scanner (TSKSCN) finds the highest priority "able", "bid", and not "suspended", or not in time delay task; then loads the worker registers and the EC, and transfers control to the worker task.

e. The task disable processor disables a task which has an error and requests an error message printout.

3-2.1.3 Supervisor Call Processors. The Supervisor Call Processors perform requested functions for the worker programs. Some of these processors which are always resident (IOQ, EOP, etc.) are part of SPB. Others, both resident and optional, are separate segments and are linked together at system generation time. Refer to paragraph 3-3.9.

3-2.1.4 Interrupt Decoders. There are three interrupt decoders, one each for CRU, DMAC, and INTERNAL interrupt processing. If there is no DMAC on the computer, the DMAC interrupt decoder may be replaced by a dummy.

3-2.1.5 Device Service Routine. There is a Device Service Routine (DSR) for each type of peripheral equipment which is attached to the computer. These routines are reusable for multiple devices. That is, only one teletypewriter service routine is required even though there may be a number of teletypewriter devices. These routines are initially entered by the I/O supervisor call processor. They are subsequently entered by the interrupt decoders when the device generates an interrupt.

3-2.1.6 Worker Tasks. There are three worker tasks which are actually a part of PAM. Task zero, which is always at priority zero, is the diagnostic task (DIAGTB). Refer to Figure 3-3. Task one, the priority of which is determined by the user, is the on-line debug task (DEBUF). Task two, the priority of which is DEBUF plus one, is the Job Control task (JCWTB).

3-2.2 PAM PROGRAM FUNCTIONS. PAM provides the following general functions (refer to paragraph 3-3):

a. Control of all standard I/O devices.

b. Arithmetic and code conversion routines.

c. Interrupt processor.

d. Scheduling of multi-programmed worker pro grams based upon real time input stimuli.

e. Scheduling of batch processing tasks based upon computer operator inputs.

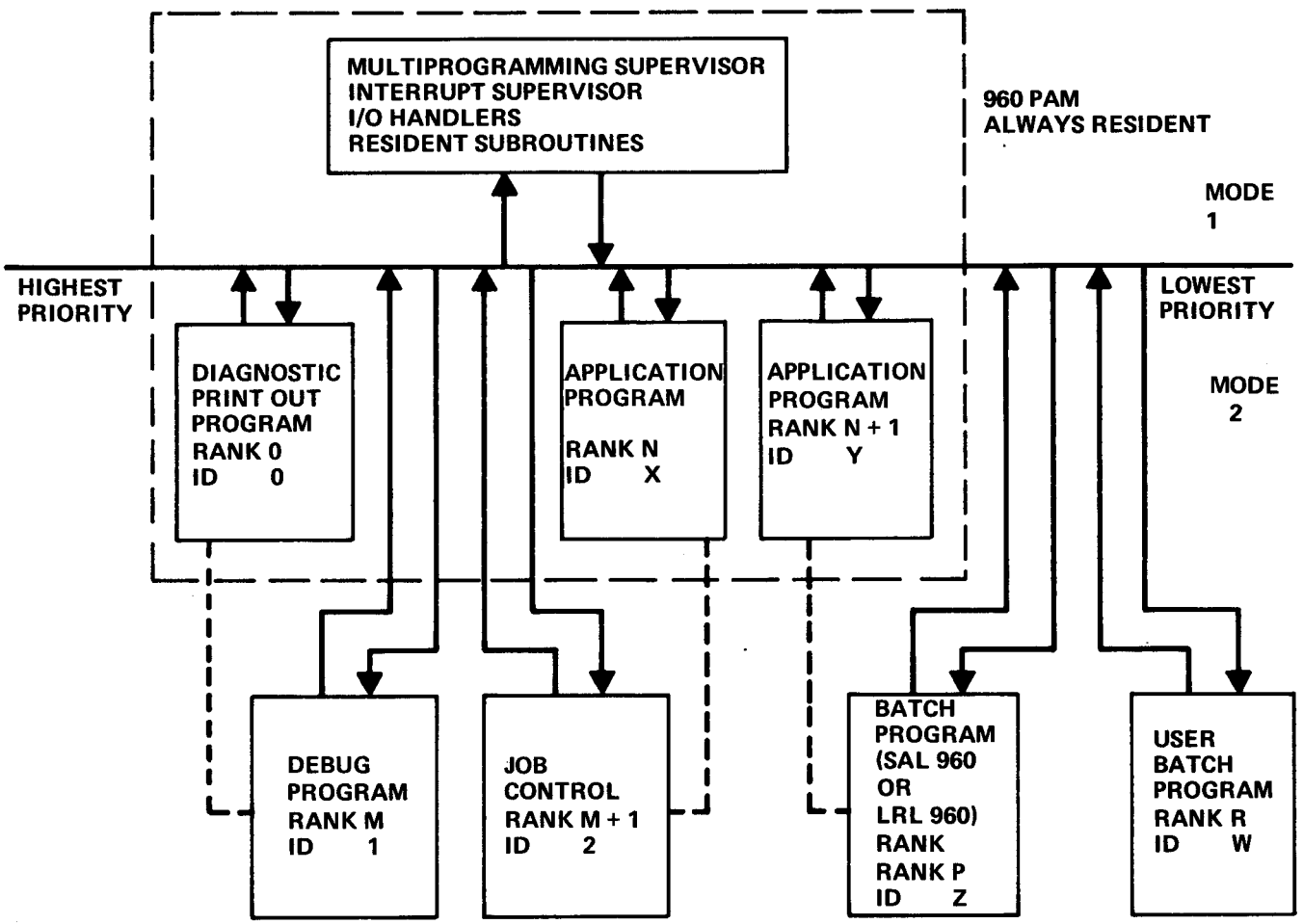


Figure 3-3 PAM960/Worker Program – Block Diagram.

f. Interval timer support, including worker program time delays and time of day and date.

3-2.3 EQUIPMENT CONFIGURATION. In order to perform effectively, PAM requires at least 8192 words of core memory, an ASR-33 teletypewriter or TI Silent 700 electronic data terminal, and an interval timer. The necessary additional devices are determined by the user. He constructs his particular PAM to support his configuration.

Three standard configurations are supplied. The basic configuration described above will be used as a starting point from which to build larger systems. A card media version supports a DP/UT SR300 card reader, DP/UT SP120 card punch, and a DP 2310 line printer. A high speed paper tape media version supports a Remex RRS 304 paper tape reader, a Tally 420 paper tape punch, and a DP 2310 line printer.

3-2.4 DATA STRUCTURE.

3-2.4.1 Supervisor Data Block. The Supervisor Data Block (SDB) can be divided into six sections, each of which is described below.

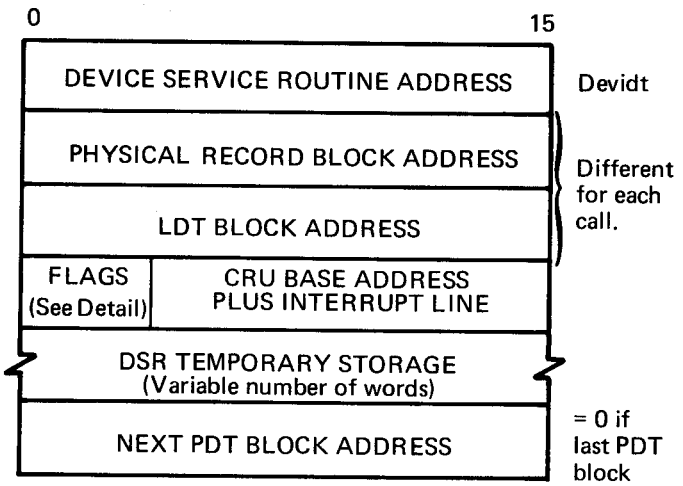
SDB-1. The first 36 words of SDB are common storage, data, flags, etc., used by all of PAM (Table 3-1).

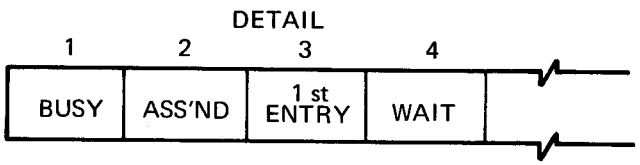
TABLE 3-1
SUPERVISOR DATA BLOCK

LABEL	NUMBER OF WORDS	USE
SENTRY	3	Supervisor Entry – object of SXBS by worker program
SUPRST	2	Pointer to Supervisor Restart – object of LDS instruction in location X'007D'.
(DATA LDTST)	1	Address of LDT table. Used by external support programs.
TSKLST	1	Pointer to first worker task.
DUTY, MDUTY	2	Current and maximum duty cycle.
YEAR-MILSEC	6	Time and date.
SEXS	1	Supervisor Exit. Switch – object of LDS* instruction to determine where supervisor will exit.
RUNWTB	1	Address of task which is currently running.
(SFLAGS)	1	Supervisor flag word. (See SFSEG listing.)
SST1-SST3	3	Supervisor Subroutine Temporary Storage
CRUINT	3	Object of SXNS instruction in location X'0094', CRU interrupt.
INTINT	3	Object of SXBS in X'0090', internal interrupt.
DMCINT	3	Object of SXBS in X'0092', DMAC interrupt.
DGFLAG	1	Diagnostic flags for general messages.
DTF, DTFQUE	3	Cause, etc. of a task disable.
SENTYA	1	Address of SENTRY (object of MOV instruction).
SXSCNA	1	Address of SXSCN (object of MOV instruction)
DCOUNT	1	Counter for duty cycle calculation.

SDB-2. The Supervisor Call Table (SCALTB) contains the address of all the routines which process supervisor calls. The order of this table determines the op code for each call. Thus, the user should not rearrange the table, although entries may be added to the bottom of the table when he adds more functions. If he does this, he must also add the appropriate REF statement.

SDB-3. The Physical Device Table (PDT) contains entries which identify each I/O device in the system to PAM. There is a PDT block for every device. All CRU devices come first, then the DMAC devices, if any, and finally the dummy device. Within the CRU group, the system teletypewriter must come first and the interval timer last. Each block has the following format.





BIT MEANING IF = 1

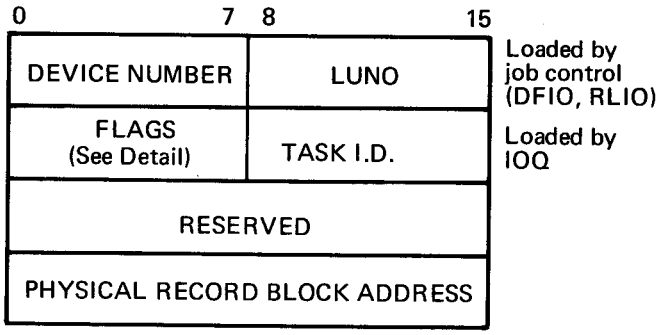
0 – Busy, device is currently processing a call.

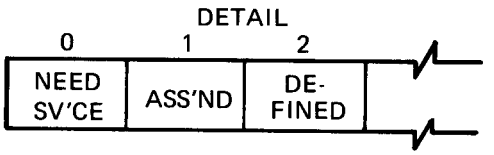
1 – Assigned, device is assigned to a specific LDT block.

2 – 1st Entry, initial entry for this call.

3 – Wait, device not ready – keep trying.

SDB-4. The Logical Device Table (LDT) contains entries which relate Logical Unit Numbers (LUNO's) to specific physical devices. A side effect of this table is to provide a one deep queue of I/O calls for each logical device. The table consists of a number of four-word blocks, one of which is pre-set to LUNO zero, Device zero, which is used by PAM. The number of blocks must be as great as the total number of LUNO's which may be in the system at any one time. PAM will require four of these in addition to the preset one. The blocks have the following format.





BIT MEANING IF = 1

0 – Need Service, this block has an unprocessed call.

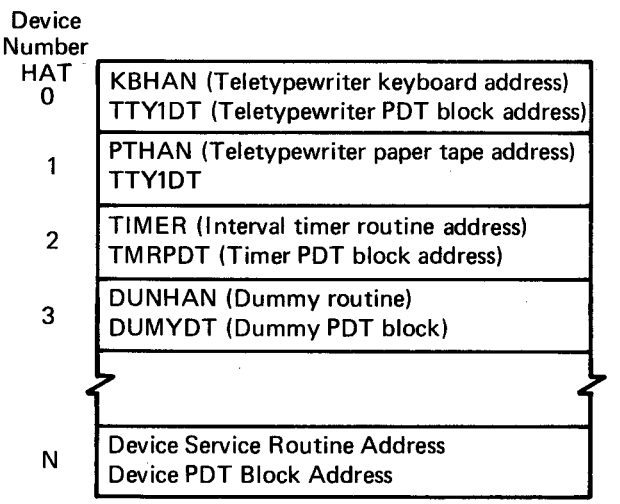
1 – Assigned, this block assigned to a particular task.

2 – Defined, this block not blank. Spare blocks have this flag cleared and word zero = X'FFFF'.

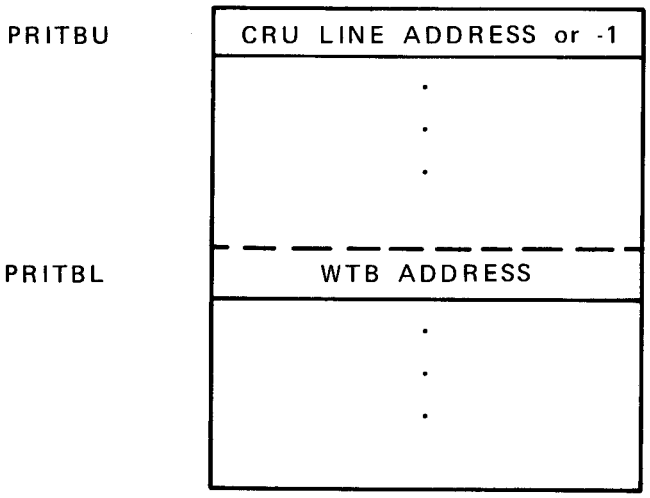
SDB-5. The Device Service Routine Address Table (DSRAT), called HAT in the listing, determines the physical device number for a particular device. The order of the table determines the device number. Devices zero through three are always present and should not be changed. There must be one entry (two words) in the table for each device.

NOTE:

A teletypewriter with paper tape equipment is considered to be two devices, although it has only one PDT block. The table's format is given below.



SDB-6. The Process Interrupt Table (PRITBU) contains entries which relate CRU line addresses and WTB addresses of tasks waiting for those lines to become a logical one. The CRU line addresses are stored in the first half of the table and the WTB addresses are stored in corresponding locations in the second half of the table. When a table slot is empty, the CRU line address is set to X'FFFF'. The number of words in the table must be twice the maximum number of tasks which can be waiting for an interrupt at one time.



3-2.4.2 Worker Task Block. The Worker Task Block (WTB) is the first 16 words of each task. It is used by PAM to:

a. identify the task and its procedure,

b. save the task's registers and EC upon suspension, and

c. keep track of the status of the task at any given time.

The format of the WTB is described in detail in paragraph 3-3.3.

3-2.4.3 Worker-Supervisor Call. When a worker program makes a supervisor call to PAM, worker register three contains the information necessary to define the call. Bits 0-5 contains an op code, and bits 6-15 usually contain an address relative to the contents of worker register four. The meaning of the op codes is given in paragraph 3-3.2.

3-2.4.4 Record Formats. Paper tape devices indicate an end of record by a reader-off code. Keyboard input devices indicate an end of record by a carriage return. Card devices consider a whole 80-column card to be a record. Output list devices recognize no end of record character. They merely utilize the character count (record length). All file oriented devices recognize a record to be an end of file if its first two characters are /*.

3-2.4.5 Support Programs. The Logical Unit Number Assignment Display task lists all LUNO's which are currently defined, lists their status and device assignment, and lists the numbers which are still available.

The Task Status Display task lists all tasks currently in the system, and their status. In addition it lists the time and date, and the duty cycle.

The Message Writer task services other tasks in the system by putting their output requests in a queue and then writing the messages.

3-3 OPERATING INSTRUCTIONS.

3-3.1 USING PAM. PAM is a multi-programming operating system utilizing an executive/worker method for program control and incorporating a multi-level priority scheme for program execution. It is the application programmer's job to write the worker tasks which will be executed under the supervision of PAM. He must determine the priorities of these tasks and their interaction with one another. He must be able to install these tasks into the computer, debug them, and bring the application *on line.* And before he can do this, he must generate the specific version of PAM which satisfies the particular needs of his application.

3-3.2 WRITING A WORKER PROGRAM.

3-3.2.1 Supervisor Calls. PAM provides I/O handlers, task sequencing, and subroutines to implement programming conveniences. The worker task requests PAM services by using supervisor calls.

The supervisor call format is independent of the function being accessed. The supervisor call format consists of one equate statement and two instruction statements.

```
            .
            .
SENTRY      EQU     127
            .
            .
            .
            .
            LA      3, Function Call Format
            SXBS    *SENTRY
            .
            .
```

Worker Register Three contains the supervisor operation code and optionally an address. Worker Register Three, bits zero through five, contain the operation code and bits six through fifteen, when used, contain an address.

The operation code is a number representing a specific function. The address is relative to the contents of Worker Register Four. The displacement value and the contents of

Worker Register Four are summed to obtain the effective address of a specific memory location.

FUNCTION CALL FORMAT

0	5 6	15
OP CODE VALUE	DISPLACEMENT VALUE RELATIVE TO WR4	
OPERATION CODE	DISPLACEMENT	

Functions marked (R) are always resident in PAM. Those marked (O) are optional. Those marked (P) are optional, but must be resident in any PAM which is to control 960 Programming System tasks. Those marked (T) are optional, but must be resident in any PAM which has full time and date support.

3-3.2.2 **Input/Output Function (00).** The input/output function effective address must point to the first memory location of the Physical Record Block (PRB) describing the I/O function to be performed (R).

PHYSICAL RECORD BLOCK

	0	1	2	3	4	5	6	7	8	15
PRB +0	WTB ADDRESS									
+1	DATA BUFFER ADDRESS									
+2	DATA BUFFER LENGTH									
+3	DATA RECORD LENGTH									
+4	BUSY	ERR	EOF	IN/EX*	C/C*	R/W*	B/A*	RES	LOGICAL UNIT NBR+	

* – SET BY CALLING TASK
\+ – LOGICAL UNIT NUMBERS 0 THRU F SYSTEM RESERVED; FF ILLEGAL

The Physical Record Block (PRB) is a five location data block containing the detailed information needed by the PAM for executing an input/output operation.

Relative Word Zero must contain the address of the first memory location of the associated WTB.

Relative Word One must contain the address of the first memory location of the associated data buffer area.

Relative Word Two must contain the length (character count) of the associated data buffer area. Input operations continue until the buffer is full, or an end of record character is read.

Relative Word Three must contain the record length (character count) of the associated data. Output operations will output this number of characters. Input operations will store the number of characters actually input in this word.

Relative Word Four is divided into two halves. The left half (bits zero through seven) is dedicated to input/output flags which signify operational state or status. The flag assignments are given in the table below. The second half (bits eight through fifteen) must contain the Logical Unit Number (LUNO) as selected by the user. Paragraph 3-5.1 tells how to define the LUNO to a specific I/O device prior to program execution.

PRB BIT		FLAGS (Word 4, Bits 0-7) USE
0	BUSY	1 = I/O in progress, 0 = I/O complete.
1	ERR	1 = Error on last operation, 0 = no error.
2	EOF	1 = End of File (/*) on last call, 0 = no EOF.
3	IN/EX	1 = Initiate Call (program returned to immediately), 0 = Execute Call (program returned to when I/O is complete).
4	CON	1 = Control Call (e.g. Punch Leader), 0 = Character I/O.
5	R/W	1 = Read (input), 0 = Write (output).
6	B/A	1 = Binary record, 0 = ASCII record.
7	RES	Reserved.

3-3.2.3 **End Of Program Function (01).** The end of program function does not use the displacement sector of Worker Register Three. The calling task is terminated and PAM enters the Task Scanner to search for another task to be executed (R).

3-3.2.4 **Bid A Task Function (02).** The task identified in the address field of register three will be executed when it becomes the highest priority bidding task. Bits six through fifteen of Worker Register Three contain the task ID itself, and not an address relative to Worker Register Four. The calling task may or may not be returned to immediately depending upon the relative priority of the calling task and the task which was bid (R).

3-3.2.5 **Multiply Function (03).** The multiply function effective address must point to the memory location containing the multiplicand. The multiplier must be in Worker Register Zero. The product will be placed in Worker Registers Zero and One (P).

3-3.2.6 Divide Function (04). The divide function effective address must point to the memory location containing the divisor. The dividend must be in Worker Registers Zero and One. The quotient will be placed in Worker Register One with the remainder placed in worker Register Zero (P,T).

3-3.2.7 Shift Memory Circular Left Double Function (05). The shift memory circular left double function effective address points to the memory address of the value to be shifted. This word and the next word in memory, treated as a 32-bit value, are rotated left the number of positions specified in Worker Register Two (P).

3-3.2.8 End Of Job Function (06). The end of job function does not use the address field of Worker Register Three. The calling task is terminated and Job Control is *bid.* Programs which are executed in a batch mode should use this function for termination rather than end of program (R).

3-3.2.9 Square Root Function (07). The effective address is not used. The argument (double precision integer) must be right justified in Worker Registers Zero and One. The integer square root is returned in Worker Register Zero (O).

3-3.2.10 Convert Binary to ASCII Coded Hexadecimal Function (CBHA) (08). The convert binary to ASCII coded hexadecimal function effective address must point to the first memory location of a two word buffer where the converted result is to be placed. Worker Register Zero must contain the binary value to be converted (R).

3-3.2.11 Convert Hexadecimal ASCII to Binary Function (CHAB) (09). The convert hexadecimal ASCII to binary function effective address must point to the first memory location of a two-location buffer containing the hexadecimal ASCII value. The binary result will be placed in Worker Register Zero (R).

3-3.2.12 Convert Binary to ASCII Coded Decimal Function (CBDA) (0A) The convert binary to ASCII coded decimal function effective address must point to the first memory location of a three-location buffer where the converted result is to be placed. Worker Register Zero must contain the binary value to be converted (P).

3-3.2.13 Convert Decimal ASCII To Binary Function (0B). The convert decimal ASCII to binary function effective address must point to the first memory location of a three-location buffer containing the decimal ASCII value. The binary result will be placed in Worker Register Zero (T).

3-3.2.14 Time Delay Function (OC). The time delay function effective address must point to the memory location containing the number of system clock counts minus one to be delayed before returning to the task. Paragraph 3-3.9.5 tells how to set the length of a system clock count (O). Note that this function, together with the next three functions, comprise the Program Control Supervisor Services segment. These functions are optional. However, if any one is needed, they must all be resident.

3-3.2.15 Wait (Unconditional) Function (00). The wait function does not use the address field of Worker Register Three. The calling task is suspended. In order for it to begin execution again, another task must release it by means of the activate suspended task function (O).

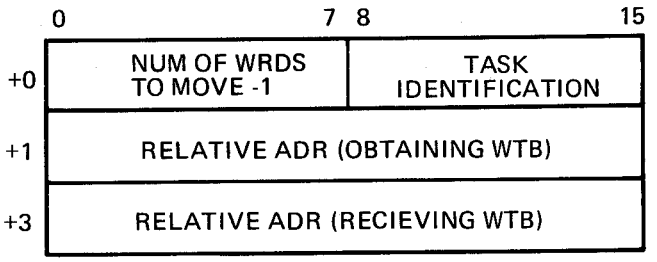
3-3.2.16 Activate Suspended Task Function (0E). The activate function address field in Worker Register Three is not used to generate an effective address. The value is a task identification number. The task specified in the address field will have its suspend bit cleared. Paragraph 3-3.3 tells about the task status bits (O).

3-3.2.17 Wait for Interrupt Function (0F). The wait for interrupt function effective address must point to the memory location containing the CRU line address of the interrupt. The task will be suspended until that interrupt occurs. The specified CRU line does not have to actually be an interrupt. If it is not, there is a delay of approximately one system clock count in responding to its change to a logical one state (O).

3-3.2.18 Get Date and Time Function (10). The get date and time function effective address must point to the first memory location of a five-location buffer. The year, day, hour, minute, and second will be placed in the five memory locations sequentially (O).

3-3.2.19 Get Data From Another Task Function (11). The get data from another task function effective address must point to the first memory location of a three-location buffer used to provide further information to PAM. The first relative location will be divided into two parts. The first part (bits zero through seven) will contain the number of locations to be moved minus one, and the second part (bits eight through fifteen) will contain the task identification number in which the locations to be moved are located. The second relative location contains the relative address value which is added to the Worker Task Block address of the task identified to obtain the address of the data to be moved. The third relative location contains the address displacement that is added to the Worker Task Block address of the operating task to obtain the effective destination address (O).

GET DATA FROM ANOTHER TASK
SUPERVISOR CALL BUFFER



3-3.2.20 **Convert Fixed Point To Floating Point (12).** The effective address is not used. The double precision integer argument must be in Worker Registers Zero and One. The floating point equivalent will be returned in Worker Registers Zero and One (O).

3-3.2.21 **Convert Floating Point To Fixed Point (13).** The effective address is not used. The floating point number in worker registers zero and one is converted to integer and returned in Worker Registers Zero and One (O).

3-3.2.22 **Floating Point Add (14).** The effective address points to a two-word block containing the floating point number to be added to the floating point number in Worker Registers Zero and One. The result is placed in Worker Register Zero and One (O).

3-3.2.23 **Floating Point Subtract (15).** The effective address points to a two-word block containing the floating point number to be subtracted from Worker Registers Zero and One. The result is placed in Worker Registers Zero and One (O).

3-3.2.24 **Floating Point Multiply (16).** The effective address points to a two-word block containing the floating point number to be multiplied by that in Worker Registers Zero and One. The result is placed in Worker Registers Zero and One (O).

3-3.2.25 **Floating Point Divide (17).** The effective address points to a two-word block containing the divisor. The dividend is in Worker Registers Zero and One. The quotient is returned in Worker Registers Zero and One (O).

3-3.2.26 **Convert Floating Point To Decimal ASCII (18).** The effective address points to a six-word buffer into which the results are placed (in E format). The number to be converted is in Worker Registers Zero and One. For example, if the floating point number had the value π, the conversion would produce an ASCII representation of:

+.314159E+01 (O).

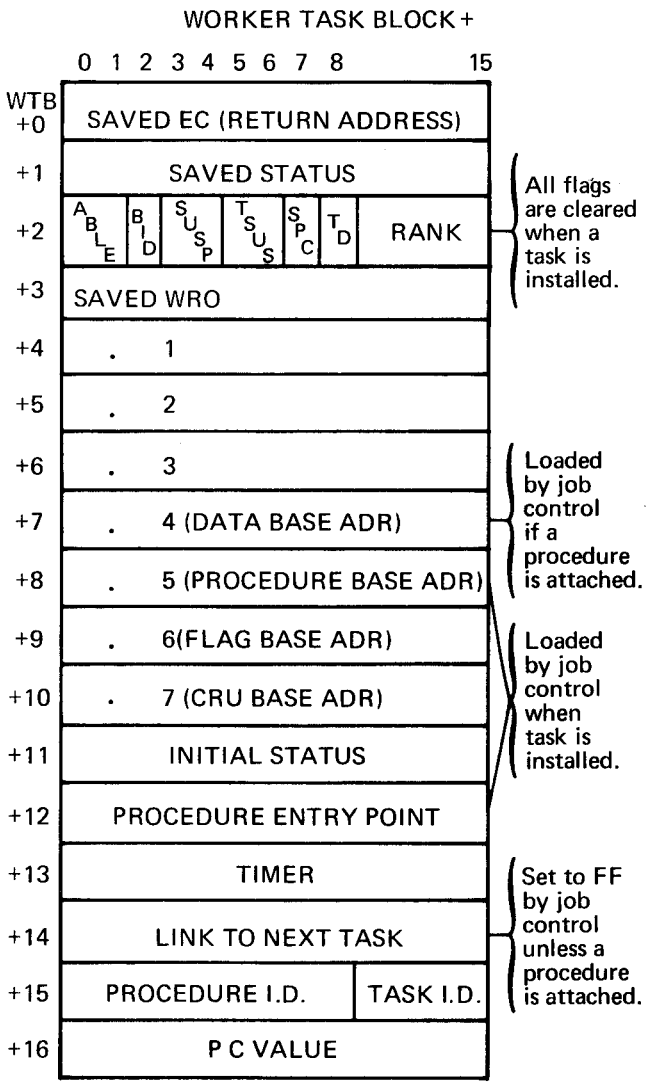
3-3.2.27 **Floating Point Sine (19).** The effective address is not used. The floating point representation of the angle X in radians is in Worker Registers Zero and One. The size of the angle must be $-\pi/2 \leqslant X \leqslant \pi/2$. The floating point representation of SIN(X) is returned in Worker Registers Zero and One (O).

3-3.2.28 **Floating Point Cosine (1A).** Arguments are the same as above except COS(X) is returned (O).

3-3.2.29 **Floating Point Arctangent (1B).** The effective address points to argument B (floating point). Argument A is in Worker Registers Zero and One. The angle value ARCTAN (A/B) is returned in Worker Registers Zero and One (O).

3-3.2.30 **Future Supervisor Calls.** Future supervisor calls will be created as needed. A maximum of sixty-four can exist (limited by the size of the operation code sector of Worker Register Three).

3-3.3 **CONSTRUCTING A WORKER TASK – THE WTB.** Tasks operating under PAM are executed in the worker mode. PAM must have access to the initial values of the EC, Status Register, and the eight worker mode registers. It must also have space in which to keep track of the current status of the task and to save the task's registers when it is suspended. The programmer must provide this space as the first 16 words of his task. These 16 words are called the Worker Task Block (WTB) and are normally the start of the DSEG. The use made of each word is described below.



\+ FIRST SIXTEEN CONSECUTIVE LOCATIONS OF TASK DATA SEGMENT

* EXISTENCE DEPENDENT ON USER DICTATES

WTB 0. The EC value (next instruction to be executed) will be saved here whenever the task is not active.

WTB 1. The Status Register contents are saved here whenever the task is not active.

WTB 2. The current status of the task is indicated by the flags in the left byte. The task rank (priority) is stored in the right byte. The flag definitions are given below.

BIT USE

0 – ABLE, set when the task is able to be executed.

1 – BID, set when the task is bid, or has been requested to execute.

2 – SUSP, set when the task is suspended (normally awaiting I/O completion).

3 – TSUS, task temporarily suspended, waiting for an I/O device to become available.

4 – SPC, set when the task wishes a value left in the PC.

5 – TD, set when the task is in a time delay.

6 – Reserved.

7 – Reserved.

WTB 3 through WTB 10. The eight worker registers are saved in these locations whenever the task is not active. If the programmer wants PAM to initialize his registers, he should place the desired values in WTB 3 through WTB 10. If the task is to be executed repetitively and if particular register initialization is desired, the registers should contain the desired values each time the task terminates. WTB 7 and WTB 8 are loaded with the WTB address and the Entry Point respectively when the task is first installed via job control.

WTB 11. The initial status register contents must be placed here by the programmer. Note that a bad value in this location could halt the machine. Normal entries would be either X'8000' or X'8200'. (Refer to Programmer's Reference Manual, paragraph 2-7.) When the task is attached to a procedure by job control and when task execution is completed, the contents of Relative Word Eleven are moved to Relative Word One.

WTB 12. The initial value of the EC (the procedure entry point) is stored here. If the procedure and task are assembled together, the programmer must put this value in WTB 12. When task execution is completed, or when a procedure is attached to the task via Job Control, the contents of WTB 12 are moved to WTB 0.

WTB 13. If the task is currently in a time delay, the number of system clock counts minus one remaining in the delay is stored here.

WTB 14. The address of the WTB for the next lower priority task is stored here. WTB 14 of the lowest priority task is zero.

WTB 15. This word is divided into two halves. The left half (bits zero through seven) contains a procedure identification number. The right half (bits eight through fifteen) contains the task identification number. The task identifier and procedure identifier are entered by Job Control when the task is installed and the procedure is attached respectively.

WTB 16. This word contains the value to be placed in the program counter if the task has flag four, Relative Word Two set to a logical one. Thus, when a mode change occurs, operational control in supervisor mode will be passed to the user at the specified location. WTB 16 provides for task operation in the supervisor mode. However, a task will always be entered by PAM in the worker mode. If WTB (4-2) is to be set to a logical one, it must be done at run time; not assembly time.

NOTE:

If the user elects not to use the above feature, relative word sixteen need not exist as a part of the WTB and WTB (4-2) should be set to zero.

3-3.4 USING THE MODEL 960 COMPUTER IN A CONTROL SYSTEM.

3-3.4.1 The Computer Advantages of the Model 960. The Model 960 computer is a carefully engineered approach to computer-controlled equipment automation. Some of the advantages are:

a. Economical and easily applicable electrical control system interface using the Communication Register Unit (CRU) and the variety of CRU function modules available.

b. Simplified methods for relating electrical connections from external equipment to symbols used in the programming language.

c. Computer programming instructions engineered to provide one instruction to one external function or event simplicity.

Digital output operation; uses one instruction

Digital input sense; uses one instruction

Analog input read; uses two instructions

Analog output; uses one instruction

Analog limit compare; uses one instruction

d. Memory utilization economy through program reusability for identical subsystem tasks. As many tasks as are necessary can share a reusable procedure program even though each task controls a physically separate device. Job Control supervises task loading and installation. Modular TI supplied supervisor functions can be added or deleted when PAM is generated.

e. Job Control to install programs on-line which lets the system grow as it is used. Typical functions:

Install tasks

Activate tasks

Delete tasks

Reorganize task priority

Change peripheral assignments

f. Complete set of off-line support programs.

960/960 Assembly program

960/IBM 360 Assembly program (OS or DOS)

960 Linking Relocating Loader

960 Performance Assurance Tests

960 Utility Programs

g. Complete documentation at two levels of detail for TI-furnished programs and systems.

User Guide – providing data for use, maintenance, and modification at the program module level.

Detailed Documentation – providing data for modification and customization at the machine instruction level.

h. All the expected conveniences.

Power Fail/Restart

Write Protected Memory with Violation Detection

Priority Interrupt Structure

Illegal Operation Detection

Interval Time Control

Time of Day and Date

Field Expansion for:

I/O Devices

Memory

Communication Register Unit

Fast Memory Speed

Direct Memory Access Channel

High Speed Peripherals

i. Low incremental cost of implementing any discrete or analog control loop for product improvement.

3-3.4.2 **Economic Justification.** Economic justification for automated equipment or products is usually derived from increased profit expectation. A better control system must, therefore, cost less to implement and perform the required tasks in an equal or superior manner. Model 960 configuration and programming flexibility allow construction of computer-control systems that are more cost effective in the above sense than many conventional analog, relay, or wired logic control systems, as well as other computer based control systems.

3-3.4.3 **Functional Specification.** All control systems, particularly those utilizing a computer, can be considered well started when the functional specification is completed.

Information typically included for the *computer system* is listed in the following outline:

I **PROJECT SCOPE.** This section

a. Contains a brief overall system description

b. List of pertinent standard practices, procedures, and reviews background, experience and expectation

c. Relates the project-to-company objectives

d. States the design goals

e. Outlines project schedule events.

II ECONOMIC JUSTIFICATION. The economic value is clearly identified.

III SYSTEM DESCRIPTION. Block diagrams of hardware configuration and programs necessary are presented.

IV PROGRAM DESCRIPTION. This section includes

a. A list of vendor software components and programs selected

b. A functional description of each program task, Model 960 programming techniques, and the Process Automation Monitor simplify task implementation. Basic tasks may be installed early in the project and others may be added as they are completed. Tasks may share reusable procedure segments.

c. Equations for calculated values and control algorithms

d. Input and output formats

e. Program timing analysis where appropriate

f. Forecast support computer requirements for program compilation and debugging.

g. On-line diagnostic program specifications.

V COMPUTER CONFIGURATION.

a. Complete description of computer components

Memory size

Peripherals and interface kits selected

CRU expansion size

CRU modules required and hardware address assignments

DMAC and port expander details, if necessary

Electrical interface details

b. Mechanical details

Installation layouts

Cable lengths and routing.

c. Operator interface or control panel functional description.

VI PERFORMANCE EVALUATION. Outline the unit and system test procedures selected to demonstrate the effectiveness of the finished system. Standard Performance Assurance Tests for the Model 960 computer and peripherals are listed. Other test programs are identified and described in detail.

VII SCHEDULE. Use the scheduling technique suited to project and company requirements. Target dates for important events are clearly established. Typical event dates to schedule are:

a. Detailed system design complete

b. Detailed programming design complete

c. Computer and vendor components order dates

d. Computer and component delivery dates

e. Input/output summary sheet and report formats 95% complete

f. Computer installation complete

g. Basic tasks installed and operating

h. All system tasks operating

i. System test complete

j. Documentation and drawings completed.

Control system schedules are often established by overall project requirements, but good planning within overall project constraints is essential for providing timely information availability and reasonable forecasts.

VIII DETAIL PROGRAM DOCUMENTATION. Detail program documentation consisting of flowcharts, task abstracts, job control statements, and program installation data are added to the functional specification after programming is completed.

IX OPERATING INSTRUCTIONS. Program loading and installation are included in this section. Detailed instructions for the use of any machine operator interface are also included.

3-3.4.4 I/O Summary. The I/O summary is simply a list of all computer input and output values. It may be expanded to include calculated values, and the same idea may also be used effectively to identify computer tasks.

Different systems require different I/O summary data. I/O summaries are typically the most referenced and universally useful document produced and should contain all pertinent data.

Sample I/O summary forms are shown in Figures 3-4 through 3-6.

DIGITAL I/O SUMMARY

OP ID	NAME	TYPE	SYSTEM USE				CRU DATA			REMARKS	SYS ID
			FUNCTION	LOGIC 1	LOGIC 0	NS	BIT PN	BIT S	BIT W		
1	2	3	4	5	6	7	8	9		10	11

1. OP ID. The unique symbol used to identify the input or output value.
2. NAME. A string of characters normally printed as part of an alarm or operation event.
3. TYPE. Input or Output.

4.,5.,6.,7. SYSTEM USE.

FUNCTION. Used to identify the event controlled by an output or sensed by an input.
LOGIC 1. Used to identify the meaning of a Logic 1.
LOGIC 0. Used to identify the meaning of a Logic 0.

NOTE: Decoding for Communication Registers several bits wide must be done separately or in REMARKS. Reference to the decoding method should be clearly stated.

8.,9. CRU DATA

BITPN. Program Name assigned to CRU bit address.
BIT. CRU address of bit. S = starting address. Communication registers can be defined. In this case W = register width in bits.

10. REMARKS.
11. SYS ID. System Identification.

Figure 3-4 Digital I/O Summary.

ANALOG OUTPUT SUMMARY

OP ID	OUTPUT NAME	OPERATING LIMITS		D/A OUTPUT BITS		D/A OUTPUT ENGR UNITS		AI	CP	CRU DATA			REMARKS	SYS ID
		LOW	HIGH	LOW	HIGH	LOW	HIGH			D/A PN	D/A S	D/A W		
1	2	3	4	5	6	7	8	9	10	11	12		13	14

1. OP ID. The unique symbol used to identify the output value.
2. OUTPUT NAME. A string of characters typically printed as part of an alarm output or value display.

3.,4. OPERATING LIMITS. The safe operating range of the output. Columns may also be required to indicate fixed or variable alarm limits or other special processing.

5.,6. D/A OUTPUT-BITS. The D/A converter output.

7.,8. D/A OUTPUT-ENGR UNITS. D/A converter output in engineering units.

9. AI. Algorithm Index. Used to identify the calculation to be performed. Other inputs and intermediate results from other algorithms are listed in REMARKS.
10. CP. Calculation Period or Interval.

11.,12. CRU DATA. Addresses used.

D/A PN D/A Converter Register Program Name.
D/A D/A Converter Communication Register Address. S = starting address. W = register width in bits.

13. REMARKS.
14. SYS ID. System Identification Number. Used if required to relate this output to identification numbers used elsewhere in the system.

Figure 3-5 Analog Output Summary.

ANALOG INPUT SUMMARY

OP ID	INPUT NAME	OPERATING LIMITS		INPUT ENGR UNITS		A/D OUTPUT BITS		CI	SP	CRU ADDRESSES						T/D TYPE	REMARKS	SYS ID
		LOW	HIGH	LOW	HIGH	LOW	HIGH			MUX PN	MUX S	MUX W	A/D PN	A/D S	A/D W			
1	2	3	4	5	6	7	8	9	10	11	12	13	13	14	15	15	16	17

1. OP ID. The unique symbol used by the computer operator to identify the input value.
2. INPUT NAME. A string of characters typically printed as a part of an alarm output or value display.
3.,4. OPERATING LIMITS. The safe operating range of the input. If alarms are implemented columns can be added for ALARM LIMITS.
5.,6. INPUT-ENGR UNITS. Transducer input range in Engineering Units.
7.,8. A/D OUTPUT-BITS. A/D Converter Output.
9. CI. Conversion Index. Used to identify the conversion equation and/or coefficients.
10. SP. Scan Period. The polling or scanning time period is specified here.
11.,12.,13.,14. CRU ADDRESS DATA. Columns may be added for sub multiplexor addresses and gain settings if required.
MUX PN. Multiplexor Register Program Name.
MUX. Multiplexor Communication Register Address. S = starting address. W = register width in bits.
A/D PN A/D Converter Register Program Name.
A/D. A/D Converter Communication Register Address.
15. T/D TYPE. Transducer or Input Type – Thermocouple, Pressure, etc.
16. REMARKS.
17. SYS ID. System Identification number. Used to relate input to identification used elsewhere in the system.

Figure 3-6 Analog Input Summary.

CRU connections listed in ascending order also assist the system designer and programmer. If the I/O Summary data is prepared on computer readable media, the summary lists are not only easy to reproduce and change but also make the data easy to present in many helpful formats.

The completed I/O summary and similar calculated value and task identification summaries are added to the functional specification and thus become a part of the final system documentation.

3-3.4.5 Installing the Computer Program. Two standard PAM's are available: the card media version and the paper tape version. The choice of either depends on the peripherals available. In addition, an almost endless variety of custom PAM's can be constructed using the SAL960 assembler and the Linking Relocating Loader. Custom versions of PAM can be constructed on a 960 with only a teletypewriter, but for convenience either high speed paper tape or card media should be available. The Programming Support Monitor (PSM) is used for generating PAM systems on Model 960 computers with 8K memory. PAM can be used for generating custom PAM versions in 960 computers with 12K memory or more.

Briefly, the system generation process is:

a. Select the optional modules required.

b. Assemble a Supervisor Data Block, if necessary.

c. Link the object modules in the prescribed sequence.

d. Load the new PAM.

e. Install Application Tasks using Job Control.

3-3.4.6 Defining Application Tasks. Tasks in the computer are items of work to be performed by the computer. The data for a task is usually unique to that particular task. The actual control program (procedure) may not be. If a reusable procedure is already resident in memory to control a particular device, then the task to control a second or even the 125th device of that same type is added by attaching the required data for the new device along with a Worker Task Block (WTB) to the device control procedure. This completes the adding of the task. If no procedure to perform the required function exists, then one is written and installed along with the data and a WTB.

Peripheral I/O performed by a task uses a Physical Record Block (PRB) to describe the I/O parameters. The programmer reads and writes a logical device so that peripherals can be assigned at the time the task is installed. This permits use of alternate I/O devices in the event the normal device is unavailable.

Process I/O is performed directly. This is an advantageous feature of the Model 960 computer. No subroutines are required to operate our external data via the CRU.

Task priority is determined at the time the task is installed. This permits system tuning by the variation of task priorities. Priorities can also be changed by temporarily deleting a task and immediately installing it once more (it

need not be reloaded) at a different priority.

3-3.5 INSTALLING JOB CONTROL PROGRAMS INTO THE SYSTEM.

3-3.5.1 Functions. Job Control provides the user a means by which he can direct the loading, installing, and executing of his task. This direction is obtained via the Job Control cards which tell PAM what specifically the user desires. Job Control is activated by a JCON command to the DEBUG program. Refer to paragraph 3-3.6.

LOAD TASK. The PAM loader loads a task or data and relocates it, beginning at the specified memory location. A bad redundancy character or an out of limits load memory location causes an error and the load is terminated. The load also terminates if a task with the specified identifier already exists. Another task should not be loaded until this one is installed, since this identifier would be lost. In order to identify to PAM a task which is already in memory, a dummy load function may be performed if no object is included. This is the way to identify the second of two tasks that have been assembled and loaded together.

LOAD PROCEDURE. The PAM loader loads a procedure and relocates it beginning at the specified memory location. A bad redudancy character or an out of limits load memory location causes an error, and the load terminates. The load also terminates if a procedure with the specified identifier already exists. Another procedure should not be loaded until this one has been attached to a task. Otherwise this identifier is lost. As above, a dummy load may be performed in order to identify a procedure which was loaded with a task.

INSTALL. The task, which was loaded previously is installed into the PAM task scan at the specified priority. If there is already a task at that priority, this request terminates with an error. A task which has just been deleted may be re-installed at another priority.

ATTACH. The specified procedure is attached to the task by placing the procedure entry point into the Relative Words Zero, Eight, and Twelve of the task's WTB. Refer to paragraph 3-3.3.

ENABLE. The able flag bit (WTB Relative Word Three) of the task WTB is set, the task entry point (WTB Relative Word Twelve) is placed in the WTB Relative Words Zero and Eight, and the WTB base address of the task is placed in the WTB Relative Word Seven. This function should be used prior to initial task execution and after a task has been disabled with an error. The task must not be re-enabled after an error until the error has been corrected.

EXECUTE. A task which has been installed and enabled is executed. This is accomplished by setting the bid flag bit (WTB Relative Word Three) of the task WTB and terminating Job Control. When the task becomes the highest priority bidding task, it begins execution. If it is desired for Job Control to be restarted after the task has finished execution, the procedure should terminate with an end of job supervisory call rather than an end of program supervisory call. Refer to paragraph 3-3.2.8.

DELETE A TASK. The task is removed from the priority structure and the PAM task scanning system. This function should *always* be used following execution of a task which will not be permanently core resident. If this task is to be re-installed at another priority, it should be installed prior to loading another task in order not to lose its identifier.

DEFINE INPUT/OUTPUT LOGICAL UNIT NUMBER. The Logical Unit Number (LUNO) is assigned to a particular input/output device. The function terminates with an error if there is no room in the PAM supervisor tables for another LUNO or if the LUNO has previously been defined.

RELEASE INPUT/OUTPUT LOGICAL UNIT NUMBER. The LUNO is removed from the PAM supervisor tables. The function should *always* be performed following execution of a nonpermanent task. The function should be performed for every LUNO that was defined for the task.

JOB CONTROL OFF. Job Control is terminated.

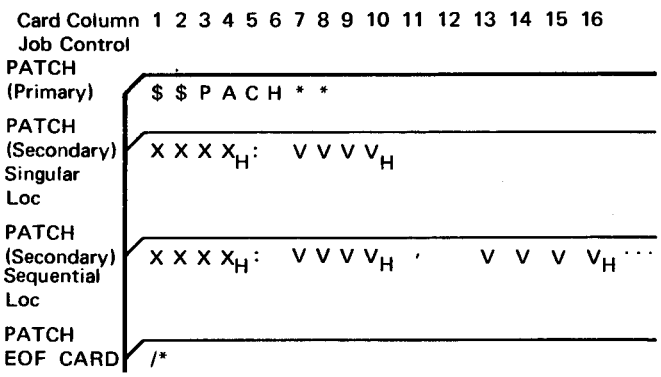
PATCH. The memory location contents of a previously loaded procedure, task, or data can be altered. An error occurs if a patch is attempted on protected memory locations, and PAM must be re-loaded.

3-3.5.2 FORMATS. The format of each control card is given below. Comments may be punched following column 18 if desired. If the input medium is paper tape or keyboard, rather than cards, the formats given are still valid, but a reader off or carriage return respectively must terminate each record.

Card Column	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Job Control	
LOAD A TASK	$$LDTS**SSSS**OOIII_T
LOAD A PROCEDURE	$$LDPR**SSSS**OOIII_P
INSTALL	$$INST**OOIII_T**OOPP
ATTACH	$$ATCH**OOIII_P**OOIII_T
ENABLE	$$ABLE**OOIII_T
EXECUTE	$$EXCT**OOIII_T
DELETE A TASK	$$DLTS**OOIII_T
DEFINE I/O LOG UNIT NO	$$DFIO**OOLL**OODD
RELEASE I/O LOG UNIT NO	$$RLIO**OOLL
JOB CONTROL OFF	$$JCOF**
END OF FILE	/*

Where: SSSS is starting Load Address
PP is priority (rank)
LL is Logical Unit Number
DD is Device Number
II_T is Task ID Number
II_P is Procedure ID Number

Patch cards have the following format. Again, comments may follow the data.

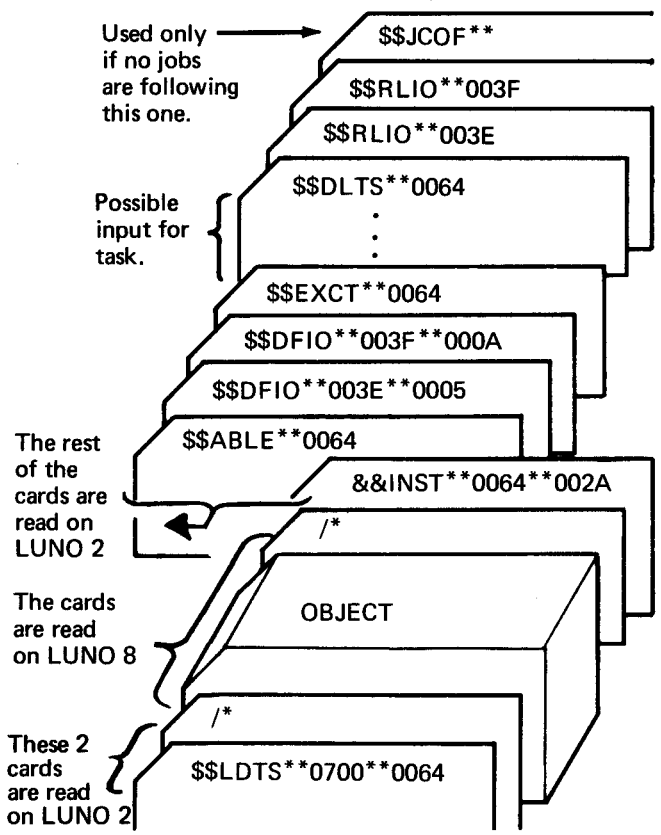


Where: $XXXX_H$ is the starting patch location (in hexadecimal)

$VVVV_H$ is the location contents (in hexadecimal)

3-3.5.3 Examples. The following are examples of the normal use of Job Control. Note that controls are input on LUNO 2, object (binary) is input on LUNO 8, and messages are output on LUNO 3. All of these *must* be defined (via DFIO commands to the DEBUG program) prior to activation of Job Control.

SINGLE BATCH TASK. The following deck loads and executes a single task with a self-contained procedure. The task uses LUNO's 3E and 3F. It is assumed that this is a batch type task and will be deleted after execution. It is given an identifier of 64 and a priority of 2A. It will be loaded at X'0700'.



TWO TASKS AND A PROCEDURE (SEPARATE). The following loads a procedure (ID 7A), load two tasks (ID's 2E, 2F) and then attach the procedure (presumably it is re-entrant) to both tasks.

```
$$LDPR**0100**007A
/*
     OBJECT
/*
$$LDTS**02F3**002E
/*
     OBJECT
/*
$$INST**002E**0015
$$LDTS**03C7**002F
/*
     OBJECT
/*
$$INST**002F**0016
$$ATCH**007A**002E
$$ATCH**007A**002F
$$ABLE**002E
$$ABLE**002F
$$JCOF**
```

The tasks are now ready to be executed when desired.

TWO TASKS AND A PROCEDURE (COMBINED). The following does the same thing as the previous example, but in this case the tasks and procedure are all assembled together (the same order and length as before).

```
$$LDPR**0100**007A
/*
     OBJECT
/*
$$LDTS**02F3**002E
/*
/*
$$INST**002E**0015
$$LDTS**03C7**002F
/*
/*
$$INST**002F**0016
$$ATCH**007A**002E
$$ATCH**007A**002F
$$ABLE**002E
$$ABLE**002F
$$JCOF**
```

SWAP PRIORITY. The following swaps the priority of the two tasks which were loaded in the previous example.

```
$$DLTS**002E
$$DLTS**002F (now 2E is lost)
$$INST**002F**0015
$$LDTS**02F3**002E
/*
/*
$$INST**002E**0016
$$JCOF**
```

CHANGE PRIORITY. The following will change the priority of task AB to priority 2F (2F must not be already assigned).

```
$$DLTS**00AB
$$INST**00AB**002F
$$JCOF**
```

PATCH. The following sets location X'0103' to X'0005', X'0ABC' to X'FFFF', X'0ABD' to 0, and X'0ABE' to X'ABCD'.

```
$$PACH**
0103:     0005            COMMENT
0ABC:     FFFF, 0000,     ABCD      COMMENT
/*
$$JCOF
```

3-3.6 DEBUG PROGRAM

3-3.6.1. The DEBUG program (segments DEBUF AND DSTART) allows the programmer to modify memory, take post mortem dumps, assign I/O devices, and initiate Job Control. To initiate the debug routine input a ! symbol on the system logging device. To terminate the debug routine, input ↑LFCR. After the routine has been initiated it responds with an OP message, meaning that it is waiting for one of the five presently defined four-letter function identifiers: LMHA, DMHA, JCON, DFIO, and RLIO. Any other inputs result in the error message BAD OP. Additional functions will be added as they become necessary.

3-3.6.2 LMHA Function (Load memory with hex numbers).

Format: LMHA $AAAA_{16}$ $NNNN_{16}$ $NNNN_{16}$ ... LFCR

Result: The string of four-digit hex numbers are converted to 16-bit binary numbers and stored consecutively in memory beginning at address AAAA. A ↑ LFCR instruction terminates the load operation.

3-3.6.3 DMHA Function (Print memory in hex)

Format: DMHA $XXXX_{16}$ $YYYY_{16}$

Result: The memory locations from XXXX to YYYY inclusive, are output onto LUNO 9 in lines of eight four-digit hex numbers along with the address of the first number on that line. At some time after loading PAM, but prior to using this function, the DFIO function must have been used to assign LUNO 9 to the teletypewriter or the line printer.

3-3.6.4 JCON Function (Activate Job Control).

Format: JCON

Result: The Job Control routine to be bid, and the debug routine to be terminated. The message J C BID is output. Prior to using this function DFIO must have been used to make the following assignments:

LUNO 2 to control input device (such as card reader)

LUNO 3 to a message output device (such as line printer)

LUNO 8 to the object input device (such as card reader)

3-3.6.5 DFIO Function (Define I/O).

Format: DFIO $OOUU_{16}$ $OODD_{16}$

Result: I/O specifying LUNO UU uses device DD. (Refer to paragraph 3-3.2.2, DEFINE I/O LUNO, and RELEASE I/O LUNO.

3-3.6.6 RLIO Function (Release I/O assignment).

Format: RLIO $OOUU_{16}$

Result: Refer to RELEASE I/O LUNO.

3-3.7 PERFORMANCE ASSURANCE TESTS. Two types of performance assurance tests are presently implemented: task errors and general messages.

3-3.7.1 Worker Task Error Message. When a worker task has a detectable error (which does not destroy the supervisor), it is disabled and a message of the following form is output on the system logging device:

TASK 3F, ERROR 03, AT 093F.

This message means that the task, the identifier being 3F, made an invalid supervisor call from location X!093F!. The meaning of the error codes is listed below:

ERROR	MEANING
00	Illegal Computer Instruction
01	Protected Memory Violation
02	Memory Parity Error
03	Invalid Supervisor Call
04	Invalid Logical Unit Number
•	
•	Future
0F	

3-3.7.2 General Messages. General messages are output to the system logging device in the following form:

DIAGNOSTIC 01

Their meanings are:

00 Card Reader Error (the card should be read again)

.
.
.
. Future
.

0F

3-3.8 LOADING PAM. To load the Process Automation Monitor via the Relocating Bootstrap Loader:

a. Determine desired load address of PAM. PAM should be loaded as high in the core as possible.

b. Select: HALT-RESET-CLEAR.

c. Place PAM object followed by /* record into object input device.

d. Select: OVER MEM PROT and:
Enter the following values into the specified addresses:

Absolute Hexadecimal Address	Value
7D	X'7C00'
7E	load bias + 3
7F	load bias

Select: OVER MEM PROT

e. Enter load address (a) into Supervisor register 0 (location X'0080').

f. Load Status register with X'01C0'.

g. Load Program Counter (PC) with 2.

h. Select RUN-START.

i. If input is on cards, start card reader.

After the /* has been read, PAM begins execution. If it contains full time and date support, it prints ENTER YEAR on the system logging device. Four digits should be input (e.g. 1970). Then the day, hour, and minute is requested in succession. The following shows the correct sequence for 1:30 PM, March 29, 1943.

ENTER YEAR	1943
ENTER DAY	0088
ENTER HOUR	0013
ENTER MINUTE	0030

Following this (or immediately following the load if there is no time and date support) the DEBUG should be initiated and the system LUNO's defined. Refer to paragraphs 3-3.6.3 and 3-3.6.4.

3-3.9 CONSTRUCTING PAM (SYSTEM GENERATION)

3-3.9.1. The PAM system has been structured to make system generation as easy as possible. When generating a system, source modification need be made to only one segment. Linkable object is available for all other segments. The only operations necessary are to assemble the one source segment and to link it to the other segments.

Linkable object exists for two different configurations of this segment. If either of these two is sufficient, then no assembly is necessary. Therefore any PAM (or PSM) system which is capable of running the Assembler and the Linking Relocating Loader can be used to generate another PAM system.

Six things affect the structure of a PAM System. They are:

a. The type and number of each type of I/O devices.

b. The structure of the service routine section (i.e., which service routines will be included or excluded).

c. The maximum number of logical device assignments possible.

d. The length of the system clock counts.

e. The priority of the DEBUG and JOB CONTROL tasks.

f. The maximum number of tasks which can simultaneously wait for an interrupt.

3-3.9.2 Type and Number of Each Type of I/O Device. To change the I/O device structure, source changes must be made to a Supervisor Data Segment (SDB) and a service routine for each type of I/O device must be included in the linking operation. All of the standard I/O device service routines available for PAM are written in a reusable manner. Thus, one device service routine can control as many devices of a particular type as desired.

a. Adding I/O devices. To add new I/O devices, two source changes must be made to SDB. They are:

(1) Add a Physical Device Table (PDT).

(2) Make an entry in the Device Service Routine Address Table (DSRAT).

The source listing of SDB is commented in such a manner that these changes can be done by following the direction of the comments. Turn in the listing to the title which says PHYSICAL DEVICE TABLE FOR XXXXX NO. N (where XXXXX is the name of the device to be added).

b. Deleting I/O devices. If at some time it is desired to delete some I/O devices from a system, three things should be done. (These changes are not necessary since the present system would be sufficient but they would reduce the size of that system.)

(1) Remove the Physical Device Table for that device from SDB.

(2) Replace the entry for that device in the DSRAT.

(3) If there is no other device of this type in the system, exclude the device service routine from the linking operation. The order of the DSRAT determines the physical device number of a device. If an entry is deleted from the middle of the table, then the physical device number for the entries following it changes. To prevent this, replace the entry with an entry which says DATA DUMHAN, DUMYDT. This keeps the table from being re-ordered and if this device number is referenced by a worker program, PAM ignores the I/O call and returns to the worker program.

3-3.9.3 Structure of the Service Routine Section. To add or delete a service routine from a system, just include or exclude the linkable object for that routine in the linking operation. Including a particular service routine in the linking operation causes the error message DBLDEF for that service routine name to be given by the Linking Relocatable Loader. Disregard this message. However, if one of these required routines is omitted by PAM (such as CHAB), the LRL gives the error message UNDEF. In this case another linking must be performed to include the routine. It is the responsibility of the user to observe all prerequisites when adding service routines. The LRL will not find errors caused by failing to observe the prerequisites. For example: CDAB requires DIVIDE, but the inclusion of CDAB and the omission of DIVIDE would not cause an error to be flagged by the LRL.

3-3.9.4 Logical Device Table Size. The basic PAM allows for a maximum of eleven logical device assignments. Five of these are used by PAM itself (one pre-defined and four defined by the operator). To increase this number add more DATA-1,0,0,0 statements to the logical device table (LDTST) in segment SDB.

3-3.9.5 System Clock Count. The length of the system clock count (paragraph 3-3.2.14 for the use of this in Time Delays) is set via the SYSCLK EQU X card in SDB. The value of X is the length of each count in milliseconds. Thus a value of 100 gives a 0.1 second clock, or ten counts per second. Values of SYSCLK greater than 1000 or less than 10 are not recommended.

3-3.9.6 Priority of Debug. The Diagnostic Task (DIAGTB) is always the highest priority task (0). The other two system tasks – DEBUG and JOB CONTROL – are at adjacent priorities, with that of DEBUG being set by the DEBPRI EQU X card in SDB. X may be any value between 1 and 253. In an application system it will probably be of lower priority than most application programs, but during system debugging operations, it should be at a high priority. Job Control is automatically at the priority immediately below DEBUG.

3-3.9.7 Process Interrupt Table Size. The basic PAM allows for a maximum of two tasks which can be waiting for an interrupt at any one time. To increase this number, add more DATA -1, -1 statements to the process interrupt table (PRITBU) in segment SDB.

3-3.9.8 PAM Segments Which Must Be In Any System.

Supervisor Data*	SDB
Supervisor	SPB
Flags	SFSEG
Converts	CHAB CBHA
Internal Interrupt	IISEG
DMAC Interrupt	DMACSG (real) or DMACDM (dummy)
CRU Interrupt	CINTSG
Record Subroutines	RECEOR
I/O Subroutines	GETCO1
Teletype	TTYSEG
System Clock	TIMER
Time & Date Support	DTDSEG (full) or DTPSEG (dummy)
Diagnostics	DIAGTB
Job Control	JCWTB
Debug*	DEBUF

*SDB must be first and DEBUF must be last. Thus, they must be at the beginning and end of the deck respectively when the linking operation is performed.

3-3.9.9 System Generation Summary. Steps to follow in generating a PAM system.

a. Read paragraph 3-3.9 carefully.

b. Decide what I/O devices will be included in the system.

c. Add a Physical Device Table for each I/O device to the Supervisor Data Segment.

d. Add an entry in the DSRAT for each I/O device in the Supervisor Data Segment.

e. If it is desirable to increase the maximum number of logical device assignments, add a DATA-1,0,0,0 to the Logical Device Table in the Supervisor Data Segment. The maximum number in the basic PAM is eleven. Each card added will increase this number by one.

f. Choose the system clock count and insert a SYSCLK EQU X card.

g. Choose DEBUG priority and insert a DEBPRI EQU X card.

h. If it is desirable to increase the maximum number of tasks which can simultaneously be waiting for an interrupt, add DATA-1,-1 card to PRITBU. Each card adds one possible task.

i. Assemble SDB.

j. Use the LRL to link all the segments together. They must include those listed in paragraph 3-3.9.8, plus routines for all I/O devices in the system, plus all additional desired service subroutines. Remember that SDB must be first and DEBUF last.

3-3.10 STANDARD VERSIONS OF PAM. Loadable object is supplied for two standard versions of PAM; a high-speed paper tape media version and a card media version. These are generated using two standard versions of SDB (paragraph 3-4.1 for device numbers). They both include the following optional segments:

NAME	FUNCTION
MULTPY	Multiply supervisor call
DIVIDE	Divide supervisor call
SCLD	Shift circular left double
*CBHA	Convert binary to hex ASCII
*CHAB	Convert hex ASCII to binary
CBDA	Convert binary to decimal ASCII
CDAB	Convert decimal ASCII to binary
TDLAY	Program Control Supervisor Services
GTDBLK	Get Data Block supervisor call
DTDSEG	Complete time and data support
DMACDM	Dummy DMAC Interrupt decoder
LPOOO	Line Printer Service Routine for CRU

*Required

The paper tape version also contains HSRHAN and PTPOOO. The card version contains CARDIN and CDPOOO.

3-4 SUBPROGRAMS.

3-4.1 SUPERVISOR DATA BLOCK (SDB). The structure of the data in SDB is described in detail in paragraph 3-3.4. Three standard versions of SDB are available.

The basic SDB is set up for teletypewriter only. However, its main use arises when building non-standard versions of PAM. This SDB is commented in a manner that the inclusion of any configuration of I/O devices may be easily accomplished by removing *'s from certain cards and re-punching a few other cards.

The card media SDB is set up to support a teletypewriter, card reader, card punch, and line printer.

The high speed paper tape media SDB is set up to support a teletypewriter, high speed paper tape reader, high speed paper tape punch, and line printer.

In the latter two versions, the following device numbers and CRU base addresses will apply.

DEVICE	NUMBER	CRU_{16}
TTY-KB	0	0000
TTY-PT	1	0000
TIMER	2	00B0
DUMMY	3	–
CARD READER	4	0400
LINE PRINTER (CRU)	5	0800
CARD PUNCH	6	0050
REMEX READER	7	0020

DEVICE	NUMBER	CRU_{16}
TALLY PUNCH	8	0040
LINE PRINTER (DMAC)	9	–

3-4.2 SUPERVISOR (SPB).

3-4.2.1 SUPRST is the first section of SPB. It is entered when PAM is initially started and each time it is restarted at location X'007D'. This section clears flags in LDT and PDT, initializes PAM worker tasks, and requests date and time initialization.

3-4.2.2 SENT (Figure 3-7) is entered when a worker program makes a supervisor call by executing a SXBS *SENTRY instruction. The function code in Worker Register Three is decoded. The op code is right justified in the A register and the effective address is placed in the E register. Then an indirect branch through SCALTB enters the appropriate processor. In PAM the following register conventions are observed: A=0, E=1, S=2, L=3, D=4, B=5, F=6, C=7.

3-4.2.3. The Supervisor General Exit routine (SGX) is entered by all the supervisor call processors when they are done (Figure 3-8). Previously the Supervisor Exit Switch (SEXS) has been set to either SENTRY or SXSCN. SGX executes an LDS *SEXS which either returns to the calling program or continues in the exit routine. If it continues, the worker program's EC is saved, and decremented if a temporary suspension is occurring. Then all worker registers are stored in the WTB and the task scan (TSKSCN) is entered.

3-4.2.4. The task scan (TSKSCN) is entered from SGX, from interrupt decoders, and from SUPRST (Figure 3-9). It searches the linked list (Figure 3-10) of all the tasks (pointed to by TSKLST) for a task which is ABLE, BID, and *not* SUSP, TSUS, or TD. Since the list is kept in order of priority, the first task found that meets the above conditions will be the highest priority one. If no such task is found, a loop is entered which increments a duty cycle counter. If such a task is found, its WTB address is stored in the current task location (RUNWTB) and the worker registers are loaded from its WTB. Worker mode is transferred to and the task is entered. The PC is left pointing to SCNXW. If the worker task does a "blind" transfer to supervisor mode, it normally returns immediately. However, if the SPC flag is set, control returns in the supervisor mode at the location specified by word 16 of his WTB.

3-4.2.5. When a task is disabled, many small sections of SPB are involved. The sequence of events is as follows:

a. Supervisor
SETF B
DFXXXX,ON TSKDSB
Set particular Error flag

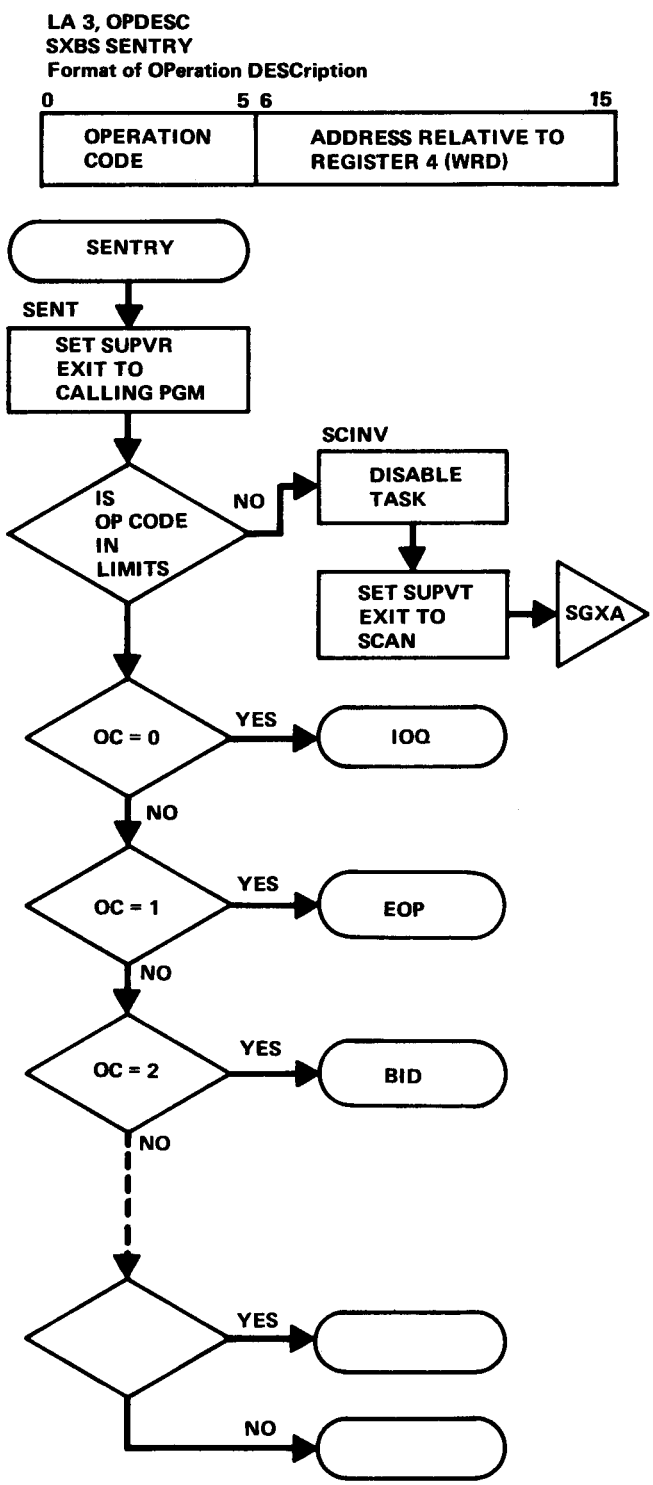


Figure 3-7 Supervisor General Entry Worker Sequence Routine.

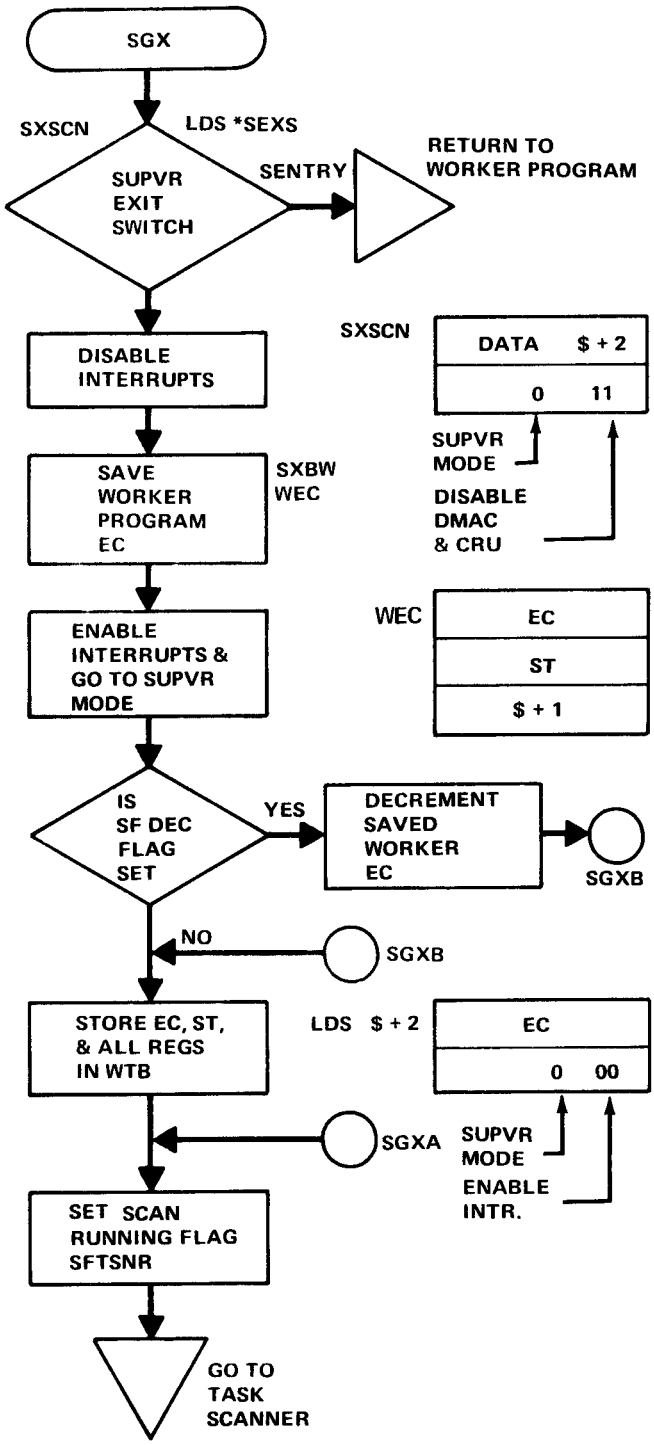


Figure 3-8 Supervisor General Exit Routine.

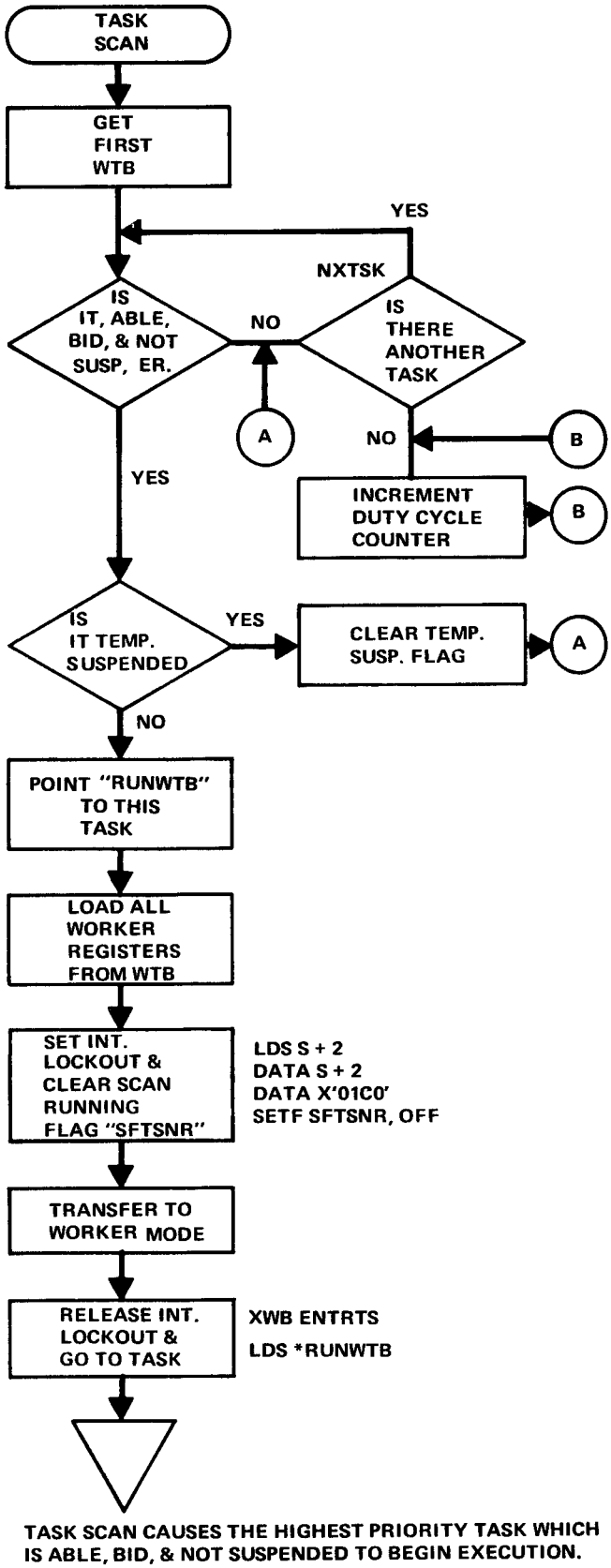


Figure 3-9 Task Scan Routine.

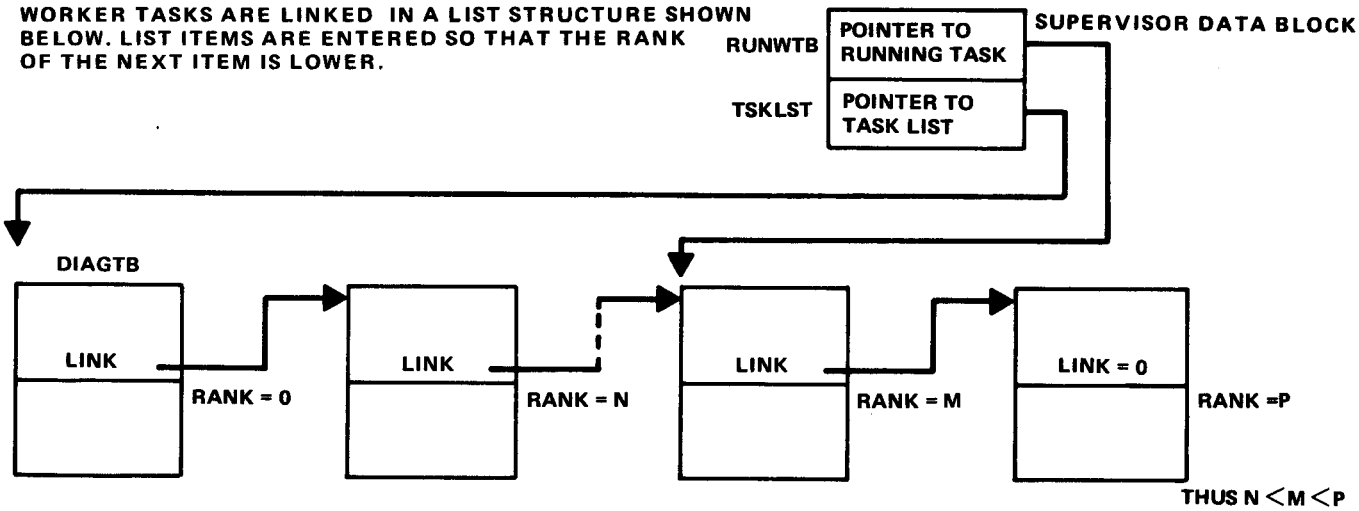


Figure 3-10 Worker Task Linking Routine.

b. TSKDSB
SETF SETF B
SFDSTS,ON ABLE,OFF EOP+2
Disable occurring Disable task

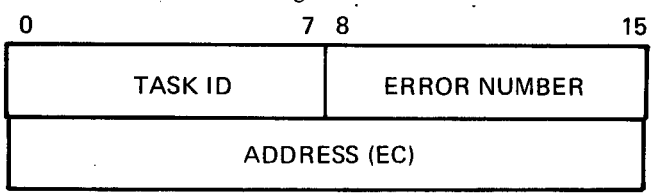
c. EOP
SETF Release assigned LDT's and PDT's LDS
BID,OFF SXSCN

d. SXSCN
Save EC, etc. in WTB B
DSTSQ

e. DSTSQ
Put task ID, Error # and EC in DTFQUE and bid for DIAGTB

f. DIAGTB
Print out task error message.

DTFQUE has the following form:



3-4.2.6. SGTWTB (get WTB subroutine) is called by bid task, get data block, unsuspend and other supervisor call processors. Upon entry E contains the task ID whose WTB is to be found. The call is SSB SGTWTB. Upon return F contains the WTB address minus one if found, otherwise F = -1. X is destroyed.

3-4.2.7. Bid task (BIDT) supervisor call processor, code 02, is entered from SENT with E containing the task ID (T) to be bid plus the contents of Worker Register Four. The ID is isolated; SGTWTB is called to find the WTB. If there is one, its BID flag is set and SGX is entered by way of IOQXSN. If none is found, the call is ignored, and again SGX is entered by way of IOQXSN. (IOQXSN restores registers B, F, and C and sets SEXS to scan.) Note that for a program to execute, it must be the highest priority program which is BID and ABLE, but not in a Time Delay, Suspension, or Temporary Suspension. Thus, if program T is of higher priority than the calling program, and it meets the above conditions, it will be executed. Otherwise, the task scanner will re-enter the calling task.

3-4.2.8. The end of program (EOP) supervisor call processor, code 01, is entered from SENT. The current program WTB address is obtained from RUNWTB and its BID flag is cleared. LDT blocks which are assigned to this task are searched for. For each are found, its assigned and need service flags are cleared, and if a PDT block is assigned to it, its assigned and busy flags are cleared. Then the task's initial status and entry point are obtained and SGX is entered at SGXB. This causes the registers and EC to be saved in the WTB and the task scan to be entered.

3-4.2.9 The end of job (EDT) supervisor call processor, code 06, is entered from SENT. The job control task (JCWTB) is bid, and then the EOP processor is entered.

Bid Task Time: *41 microseconds + $38N_T$ microseconds
Plus PAM overhead of 190 microseconds + 37 microseconds (Return to Task N)

3-4.2.10. The I/O supervisor call processor (IOQ), code 00, is entered from SENT with E containing the physical record block (PRB) address (Figure 3-11). The logical unit number (LUNO) is isolated and an LDT block with that LUNO is sought. If none is found, the task is disabled. If one is found but is busy (need service flag set), the task is temporarily suspended by setting the TSUS flag and SGX is entered. This decrements the EC and the worker re-executes the call later. The same thing occurs if the LDT block is assigned to another task.

If neither of the above conditions obtains, the LDT block is assigned to this task, the PRB address is inserted, and the need service flag is set. The device number is then isolated and the corresponding PDT block obtained. If the PDT block is busy or assigned to another task, then the calling task is suspended if it was an execute call, or returned to if it was an initiate call. If the PDT block is free, the LDT address, PRB address, and device service routine address are stored in it. Then the busy, assigned, and first entry flags are set. The LDT need services flag is cleared. Registers D and C are initialized, interrupts are masked, and the device service routine is entered.

Upon return, the task is suspended if it was an execute call. IOQXN sets SEXS to scan and exits to SGX.

3-4.2.11. SSBREG and SETBC are two system subroutines which are re-entrant and thus may be called by interrupt routines. I/O drivers, or the main supervisor. Both are called with a BL in register L.

SSBREG sets B = SPB, F = SDB, and C = O. It then returns (B 2,L).

SETBC is entered with F containing a PDT address. B is set to zero. D is set to (F) + 5. C is loaded from 3, F, and *anded* with X'0007' for a DMAC device or with X'0FF0'

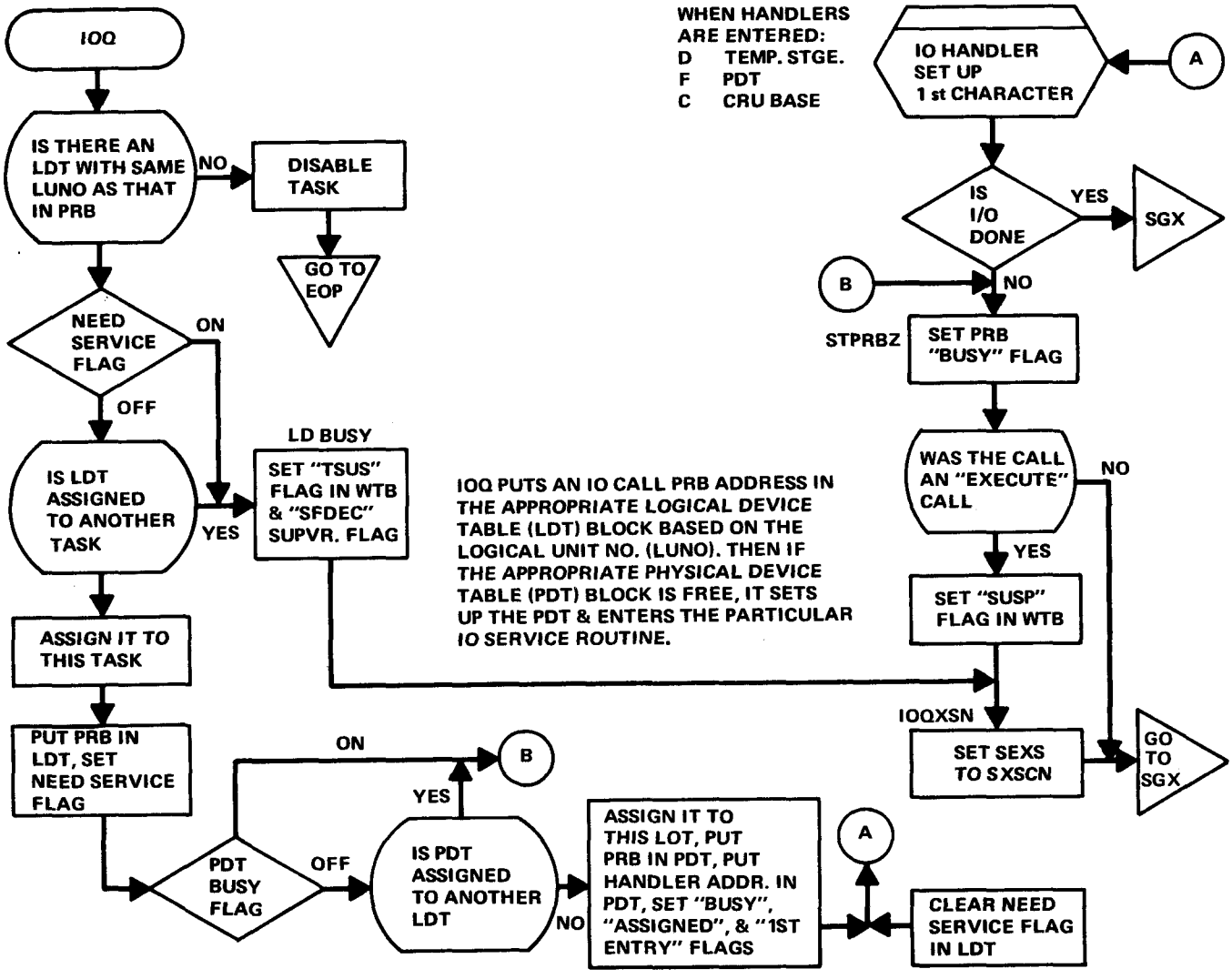


Figure 3-11 I/O Supervisor Call Processor Routine.

for a CRU device. Thus C equals the DMAC part number or the CRU base address. The routine then returns (B 2,L).

3-4.3 OPTIONAL SUPERVISOR CALL PROCESSORS. All supervisor call processors not included in SPB above are assembled as separate segments and must be linked with the other object decks when generating PAM. Each is given a separate document number and each may be used with PAM or PSM. Some of them (such as DIVIDE) contain subroutines as their main element. These subroutines may be called from other parts of the supervisor. For instance, CBDA calls the subroutine which is the major element of divide. Thus some of these processors are prerequisites for others.

All times given are for the routines themselves. The PAM overhead must be added to this. This is approximately 110 microseconds if SGX returns immediately to the calling task. It is 190 microseconds plus 37 N microseconds if the task scan is entered and returns to the Nth task.

3-4.3.1 Multiply. The multiply function effective address must contain the memory address of the multiplicand. The multiplier must be in Worker Register Zero. The product will be placed in Worker Registers Zero and One (B).

CODE	FUNCTION
03	Multiply

3-4.3.2 Divide. Te divide function effective address must contain the memory address of the divisor. The dividend must be in Worker Registers Zero and One. The quotient is placed in Worker Register One with the remainder placed in Worker Register Zero (B).

CODE	FUNCTION
04	Divide

3-4.3.3 Shift Memory Circular Left Double. The shift memory circular left double function effective address contains the memory address of the value to be shifted. This word and the next word in memory, treated as a 32-bit value, are rotated left the number of positions specified in Worker Register Two (B).

CODE	FUNCTION
05	Shift Memory Circular Left Double

3-4.3.4 Square Root. The effective address is not used. The argument (double precision integer) must be right justified in Worker Registers Zero and One. The integer square root returns in Worker Register Zero (O).

CODE	FUNCTION
07	Square Root

3-4.3.5 Convert Binary To ASCII Coded Hexadecimal. The convert binary to ASCII coded hexadecimal function effective address must contain the memory address of a two-word array where the converted result is placed. Worker Register Zero must contain the binary value to be converted (B).

CODE	FUNCTION
08	Convert Binary To ASCII Coded Hexadecimal

3-4.3.6 Convert Hexadecimal ASCII To Binary. The convert hexadecimal ASCII to binary function effective address must contain the memory address of a two-location array containing the hexadecimal ASCII value. The binary result will be placed in Worker Register Zero (O).

CODE	FUNCTION
09	Convert Hexadecimal ASCII To Binary

3-4.3.7 Convert Binary To ASCII Coded Decimal. The convert binary to ASCII coded decimal function effective address must contain the address of a three-location array where the converted result is placed. Worker Register Zero must contain the binary value to be converted (B).

CODE	FUNCTION
OA	Convert Binary To ASCII Coded Decimal

3-4.3.8 Convert Decimal ASCII To Binary. The convert decimal ASCII to binary function effective address must contain the memory address of a three-location array containing the decimal ASCII value. The binary result returns in Worker Register Zero (O).

CODE	FUNCTION
OB	Convert Decimal ASCII To Binary

3-4.3.9. The next six supervisor calls are defined for PAM only. They are treated as errors when encountered by PSM. They are listed here for reference only.

CODE	FUNCTION
OC	Time Delay
OD	Wait – Unconditional
OE	Activate "Waiting" Task
OF	Wait for Interrupt
10	Get Date and Time
11	Get Data Block from Another Task

3-4.3.10 Convert Fixed Point To Floating Point. The effective address is not used. The double precision integer argument must be in Worker Registers Zero and One. The floating point equivalent returns in Worker Registers Zero and One (O).

CODE	FUNCTION
12	Convert Fixed Point To Floating Point

3-4.3.11 Convert Floating Point To Fixed Point. The effective address is not used. The floating point number in Worker Registers Zero and One is converted to integer and returned in Worker Registers Zero and One (O).

CODE	FUNCTION
13	Convert Floating Point To Fixed Point

3-4.3.12 Floating Point Add. The effective address contains the location of a two-word block which contains the floating point number to be added to the floating point number in Worker Registers Zero and One. The result returns in Worker Registers Zero and One (O).

CODE	FUNCTION
14	Floating Point Add

3-4.3.13 Floating Point Subtract. The effective address contains the location of a two-word block which contains the floating point number to be subtracted from Worker Registers Zero and One. The result returns in Worker Registers Zero and One (O).

CODE	FUNCTION
15	Floating Point Subtract

3-4.3.14 Floating Point Multiply. The effective address contains the location of a two-word block which contains the floating point number to be multiplied by that in Worker Registers Zero and One. The result returns in Worker Registers Zero and One (O).

CODE	FUNCTION
16	Floating Point Multiply

3-4.3.15 Floating Point Divide. The effective address contains the location of a two-word block which contains the divisor. The dividend is in Worker Registers Zero and One. The quotient returns in Worker Registers Zero and One (O).

CODE	FUNCTION
17	Floating Point Divide

3-4.3.16 Convert Floating Point to Decimal ASCII. The effective address contains the location of a six-word array into which the results are placed (in E format). The number to be converted is in Worker Registers Zero and One. For example, if the floating point number had the value π, the conversion would produce an ASCII representation of the following:

+.314159E + 01 (O).

CODE	FUNCTION
18	Convert Floating Point To Decimal ASCII

3-4.3.17 Floating Point Sine. The effective address is not used. The floating point representation of the angle X in radians is in Worker Registers Zero and One. The size of the angle must be $-\pi/2 = X = \pi/2$. The floating point representation of SIN(X) returns in Worker Registers Zero and One (O).

CODE	FUNCTION
19	Floating Point Sine

3-4.3.18 Floating Point Cosine. Arguments are the same as in paragraph 4-4.2.21 except COS(X) returns (O).

CODE	FUNCTION
1A	Floating Point Cosine

3-4.3.19 Floating Point Arctangent. The effective address contains the location of Argument B (floating point). Argument A is in Worker Registers Zero and One. The angle of the value of which is ARCTAN (A/B) returns in Worker Registers Zero and One.

CODE	FUNCTION
1B	Floating Point Arctangent

3-4.3.20. Program Control Supervisor Services (TIMDLY).

LENGTH: 50 words

PREREQUISITES: None

PERTINENT INFORMATION: Four supervisor call processors are included in this segment – TIMDLY, WAIT, UNSUSP, WTINT.

Their descriptions follow.

TITLE: TIMDLY

PURPOSE: Suspend the calling task for N+1 system timer counts.

*TIME: 56 microseconds.

STORAGE: 8 words (set up subroutine only, processor requires 34 words).

PREREQUISITES: None
Part of program control supervisor services.

PERTINENT INFORMATION: TIMDLY puts the count N in WTB(13), sets the TD flag in WTB(2), and exits through IOQXSN. This will save the task's registers and EC in its WTB and then enter the task scanner.

Every time the interval timer causes an interrupt, the TIMER driver decrements WTB(13) by one for every task whose TD flag is set. When WTB(13) becomes negative, the TD flag is cleared. Then the task scanner will re-initiate the program.

ERROR RETURN: None

TITLE: WAIT (unconditional)

PURPOSE: Suspend the calling program. Assume another task will later use an UNSUSP call to restart it.

*TIME: 16 microseconds

STORAGE: 6 words

PREREQUISITES: None
Part of Program Control Supervisor Services.

PERTINENT INFORMATION: WAIT sets the SUSP flag in WTB(2) of the calling program and then exits to the task scanner through IOQXSN.

ERROR RETURN: None

TITLE: UNSUSP

PURPOSE: Unsuspend task T – Assumes the task is able, bid, and suspended. If so, unsuspending it will allow it to run.

*TIME: 30 microseconds

STORAGE: 14 words

PREREQUISITES: None
Part of Program Control Supervisor Services.

PERTINENT INFORMATION: UNSUSP calls SGTWTB to find the WTB for task T. It then clears the SUSP flag in WTB(2) and exits to the task scan through IOQXSN.

ERROR RETURN: If no task T, Worker Register Three is set to zero upon return.

TITLE: WTINT

PURPOSE: Suspend the task until the specified interrupt occurs.

LENGTH: 22 words

PREREQUISITES: Part of Program Control Supervisor Services.

PERTINENT INFORMATION: WTINT finds an empty slot in the PRITBU table and stores the CRU line address and the WTB address in it. It then exits through WAIT to suspend the task. If the CRU interrupt decoder ever finds the above CRU line set to a one, it will unsuspend the task.

ERROR RETURN: If there is no room in the table, Worker Register Three is set to zero and the calling program is returned to through SGX.

*Plus PAM overhead of 110 microseconds (For Return To Calling Task) or 190 microseconds + 37N microseconds (Return To Task N)

3-4.3.21 Get Data Block.

TITLE: GTDBLK

PURPOSE: Transfer a block of N_W words of data from task T (which is the N_T^{th} task) to the calling task.

*TIME: (94 microseconds) + (38 N_T microseconds) + (16 N_W microseconds)

STORAGE: 40 words

PREREQUISITES: None

PERTINENT INFORMATION: GTDBLK calls subroutine SGTWTB to find the WTB for task T. It finds the source data block by adding the address specified to the WTB address of task T. The receiving data buffer is the address specified plus the calling task's WTB address. N_W words are transferred to the receiving data buffer and control is returned to the calling task.

ERROR RETURN: If there is no task T, Worker Register Three is set to zero upon return. The data buffer is unchanged.

3-4.3.22 Get Date.

TITLE: GTDATE

PURPOSE: Move YEAR, DAY, HOUR, MINUTE, & SECOND in a five-word buffer to a five-word buffer in the calling task.

*TIME: 119 microseconds

STORAGE: 12 words

PREREQUISITES: Part of Time and Date Support – DTDSEG

PERTINENT INFORMATION: The contents of PAM's time and date locations are moved to the five-word buffer specified in the call. Note that if complete time and date support have not been included when PAM was generated, this call will be treated as an error and the calling task disabled.

Following completion of the call, control is immediately returned to the calling task.

ERROR RETURN: None

*Plus PAM overhead of 110 microseconds (For Return To Calling Task) or 190 microseconds + 37N microseconds (Return To Task N)

3-4.4 INTERNAL INTERRUPT DECODER (IISEG). When an internal interrupt occurs (Figure 3-12), a trap to location X'0090' is taken. SXBS INTINT instruction is sorted there and INTINT+2 points to IISEG. The status stored in INTINT+1 is examined to determine the cause of the interrupt and the mode when the interrupt occurred. If power failure has occurred, locations X'0090' and X'0091' are modified to cause the Power On routine to be entered when power is restored. The Power On routine restores locations X'0090' and X'0091' and returns to the place of interruption. For other causes, a particular diagnostic flag is set. If the machine was in the supervisor mode, a B $ instruction is executed. Otherwise, the worker task which caused the interrupt is disabled and the task scan entered.

Length: 66 words

Prerequisite: Locations X'0090' and X'0091' must have been set by the initialization routine in Job Control (JCRDBF).

3-4.5 CRU INTERRUPT DECODER (CINTSG). When a CRU interrupt occurs (Figure 3-13), a trap to location X'0094' is taken. An SXBS CRUINT instruction is stored there, and CRUINT+2 points to CINTSG. All supervisor mode registers are saved. Then, by loading F with each PDT block address and C with word three of each PDT block, the interrupt bit of each device whose busy flag is set is checked. The system teletypewriter is checked regardless, since it may accept unrequested input. For each interrupt bit which is on, the appropriate device service routine is entered.

After all devices have been processed, the Process Interrupt Table (PRITBU) is checked. If any of the specified CRU lines is a one, the corresponding task is unsuspended.

Next the conditions which prevailed prior to the occurrence of the interrupt are checked to determine the proper exit. The possibilities are enumerated on the flow chart on the next page.

Length: 123 Words

Prerequisites: Locations X'0094' and X'0095' must have been preset.

Approximate time: 250 microseconds

3-4.6 DMAC INTERRUPT DECODER (DMACSG). When a DMAC interrupt occurs (Figure 3-14), a trap to location X'0092' is taken. An SXBS DMCINT is stored there, and DMCINT+2 points to DMACSG. All supervisor mode registers are saved. Location X'0096' is then examined to see which port caused the interrupt. Prior to the interrupt, the DMAC set bits in location X'0096' corresponding to the port numbers of the devices waiting to interrupt. Bits 0-7 correspond to ports 0-7. If there is a DMAC PDT block corresponding to that port, the device service routine is

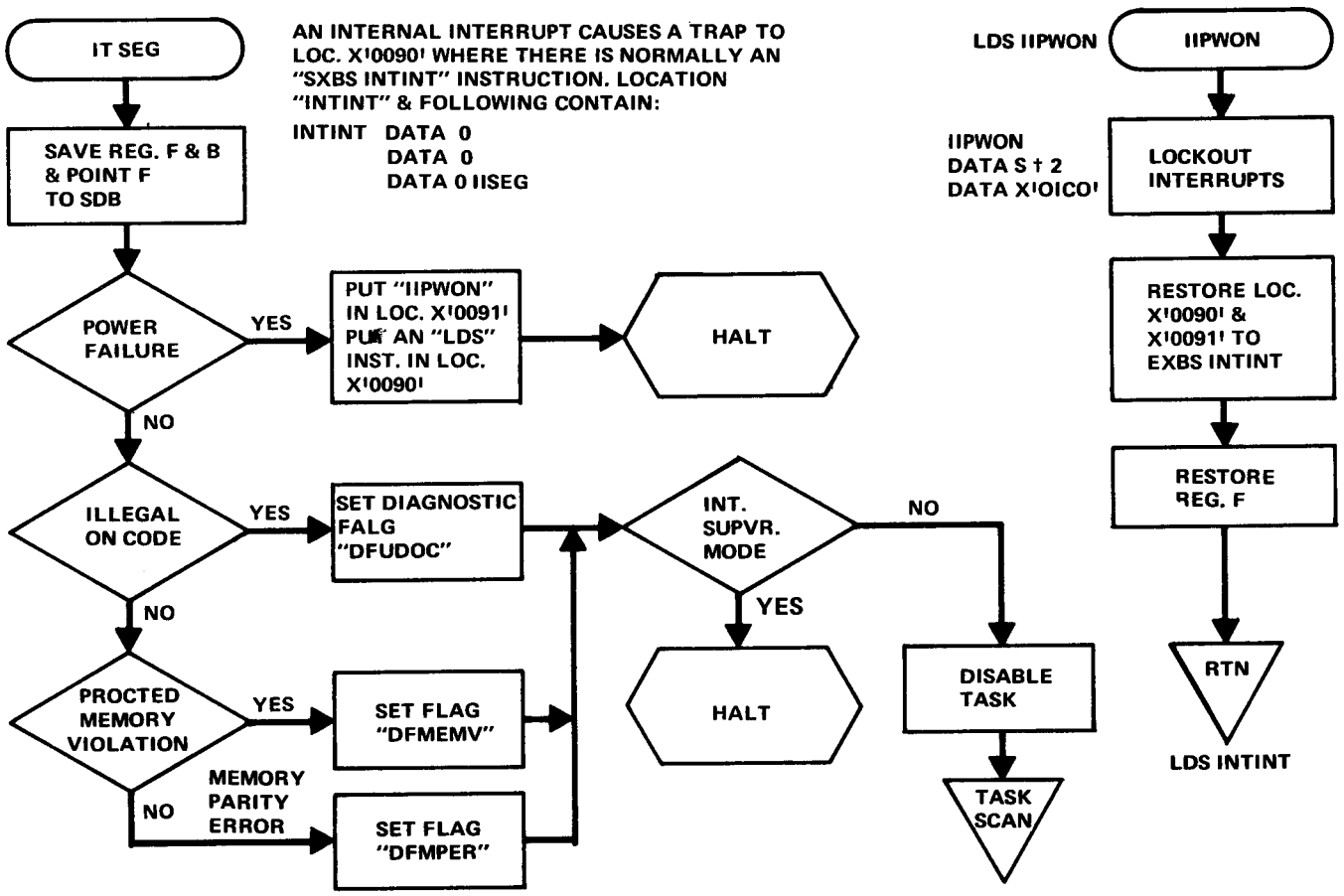


Figure 3-12 Internal Interrupt Service Routine.

entered. That routine then checks the device status, stored in location port number times two plus X'0098'.

After all interrupting ports have been serviced, status prior to the interrupt is checked to determine the correct exit. If the task scan was interrupted, it is restarted. If a worker was interrupted it will be halted, and the task scan restarted. If another part of the supervisor was interrupted, the registers are restored and it is returned to.

Length: 88 words

Prerequisites: Locations X'0092' and X'0093' preset.

3-4.7 END OF RECORD ROUTINES (RECEOR). When an I/O device completes a record (Figure 3-15), it branches to RECEOR or FILEOR depending on whether it is a record or file oriented device respectively. A file oriented device is used exclusively by a task until an end of file record is processed. These routines first use subroutine EORA which clears the PRB busy flag, clears the WTB suspend flag if it was an execute call, and sets the I/O done flag (SFCIOD). It then checks for end of file (/*) and, if it was not /*, returns to RECEOR or FILEOR. If there was a /* it sets the PRB EOF flag and clears the LDT assigned flag if it does not need service. It then enters LDTSCN (Figure 3-16).

When RECEOR is returned to with no EOF, it performs the above function on the LDT and enters LDTSCN.

When FILEOR is returned to with no EOF, if the LDT block is still assigned but does not need service, it returns to IOQ or CRU interrupt decoder. If it needs service the arguments are moved into the PDT and the device service routine is re-entered.

When LDTSCN is entered, it searches the Logical Device Table for an LDT block with the appropriate device number which needs service. (The LDT is effectively a one level deep queue of I/O requests, one for each LUNO.) If none is found, it returns. If it finds one it puts the appropriate data in the PDT and re-enters the device service routine.

Length: 123 words

3-4.8 I/O COMMON ROUTINES. This set of routines is used by many of the I/O device service routines. They

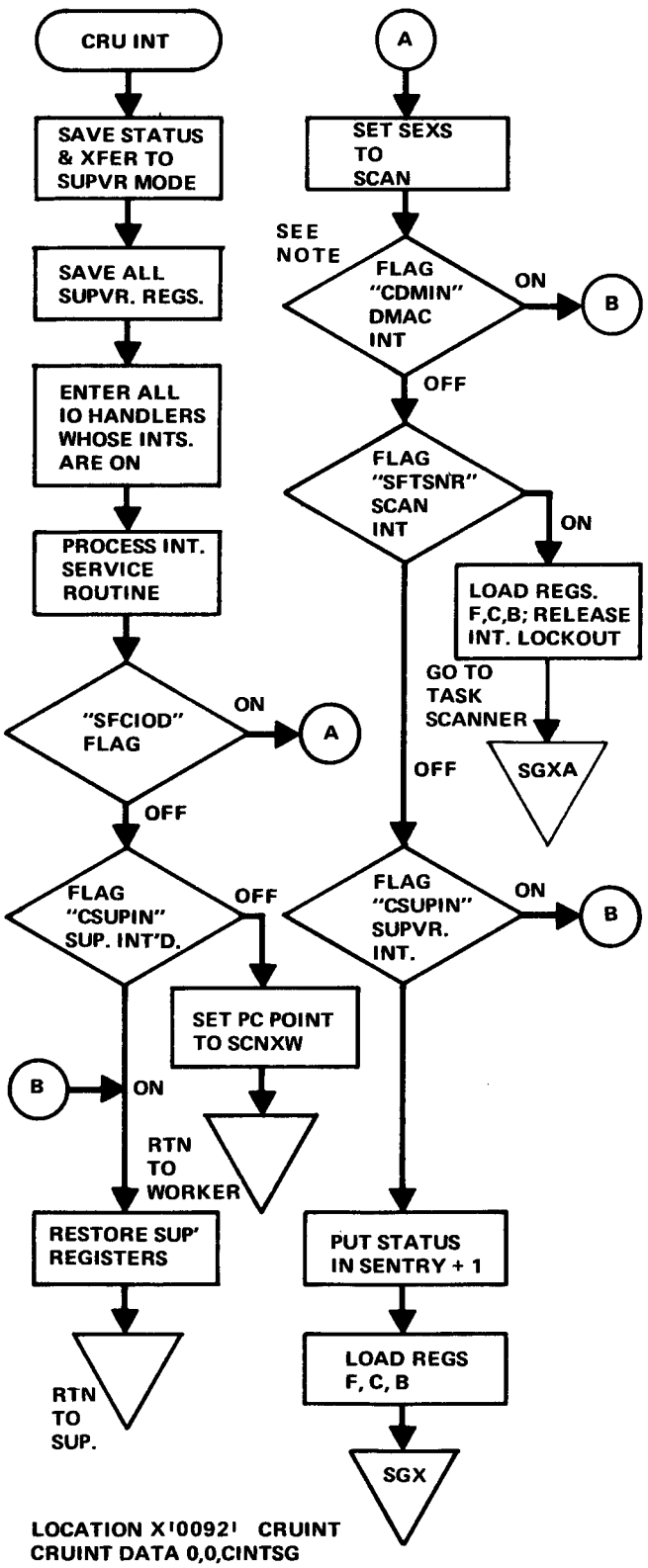


Figure 3-13 CRU Interrupt Service Routine.

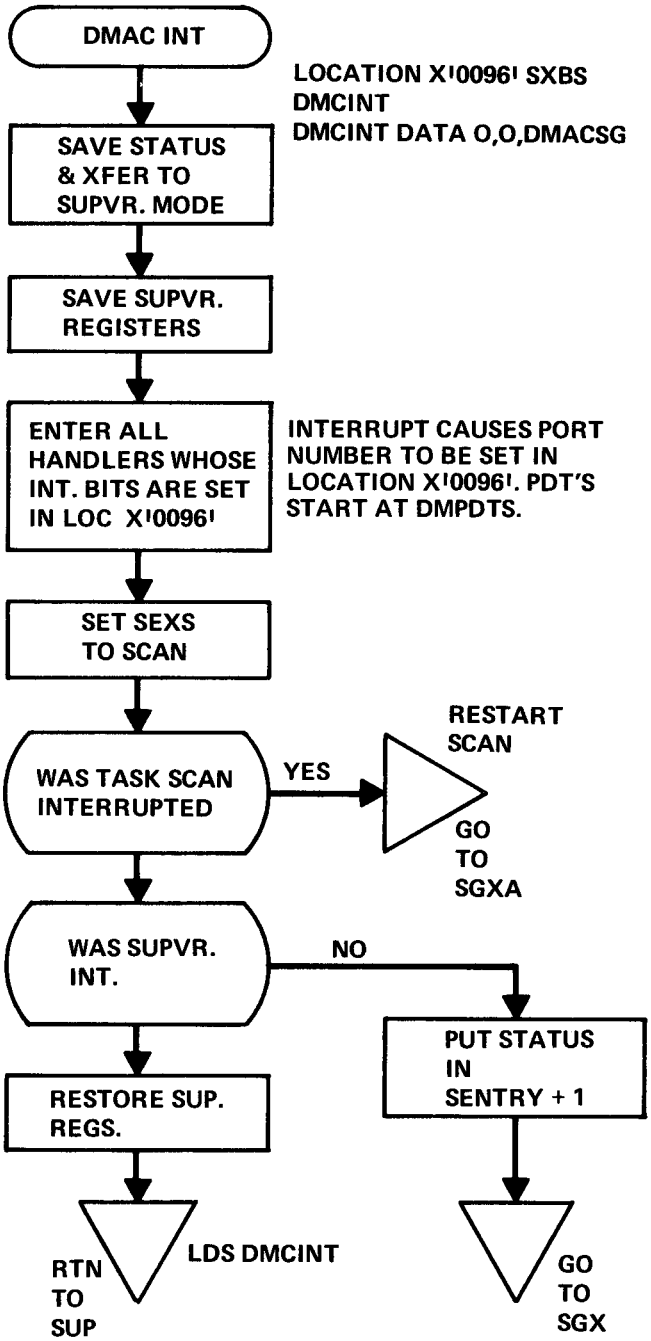


Figure 3-14 DMAC Interrupt Service Routine.

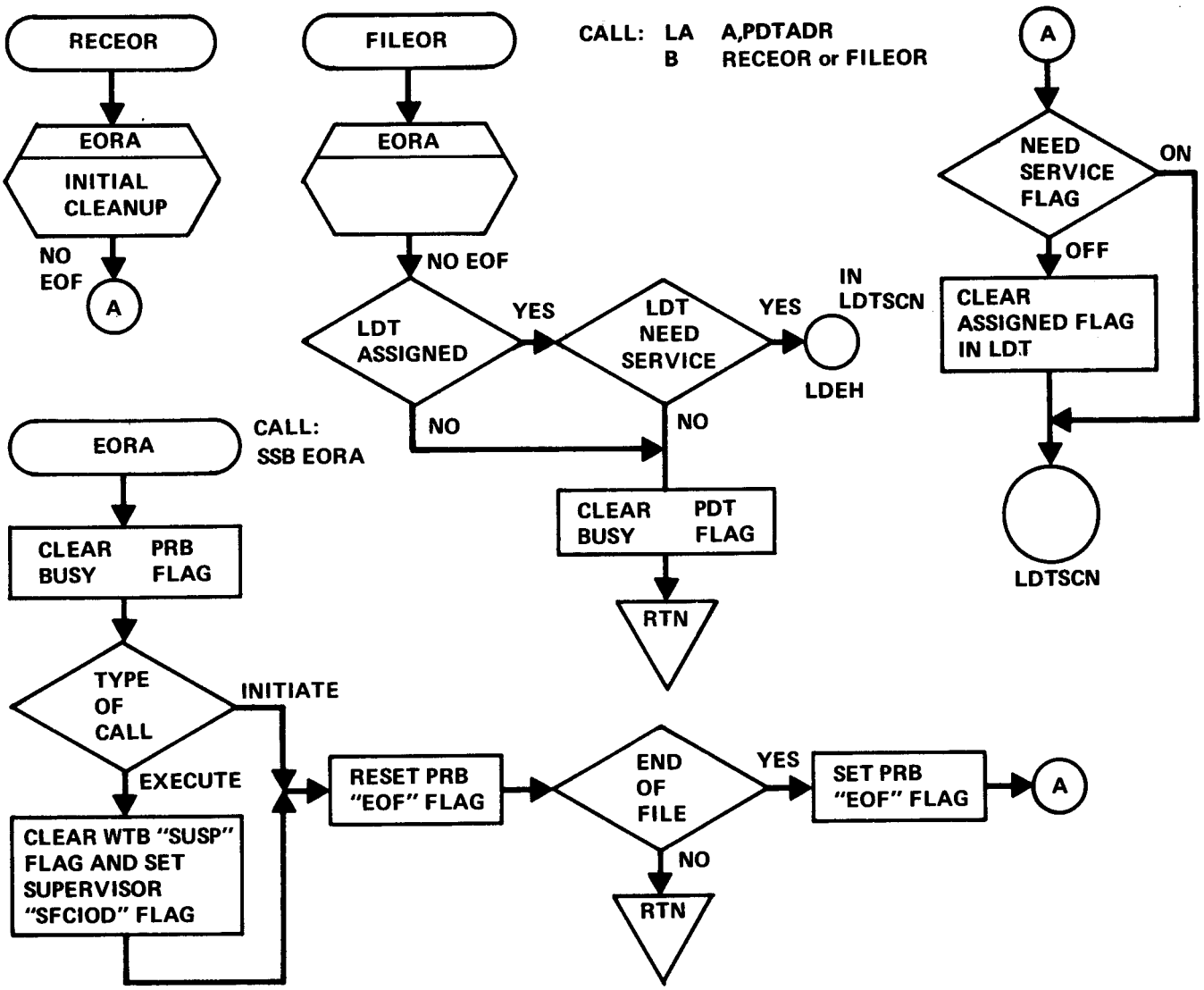


Figure 3-15 End of Record Routine.

assume a standard use of temporary storage locations. Word zero contains the character count and word one contains the current character.

GETCO1 is a routine to fetch the next character. Upon entry D must point to temporary storage and A to the PRB. The calling sequence is:

```
BL      L,GETCO1
DATA    EOR         End of record return address
- -                   Normal return location
```

The next character is fetched and placed in 1,D and the character count in O,D is incremented. If there are no more characters (character count = record length in PRB), the return is to the end of record location.

XORCOO is a routine which inverts the character in the right half of 1,D. The call is BL L,XORCOO.

PRBERR is a routine which sets the error flag in the PRB. On entry A must point to the PRB. The call is BL L,PRBERR.

Length: 44 words

3-4.9 CRU DEVICE SERVICE ROUTINES. The routines described in this section are the device service routines for devices connected to the CRU (Figure 3-17). They all have many common features. They all run in the supervisor mode. They use the PDT first entry flag to determine if they are being entered for the first time for a particular call. If the device is not ready, they set the PDT wait flag and

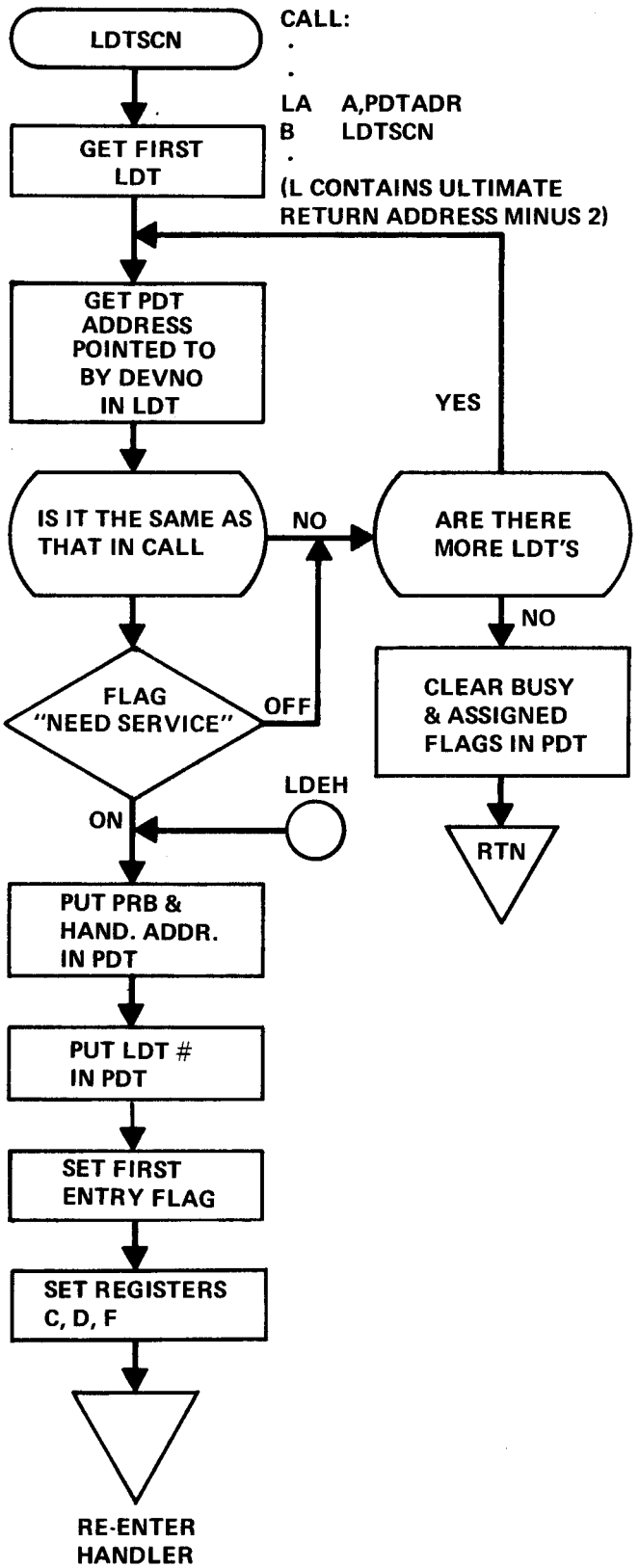


Figure 3-16 Logical Device Table Scan Routine.

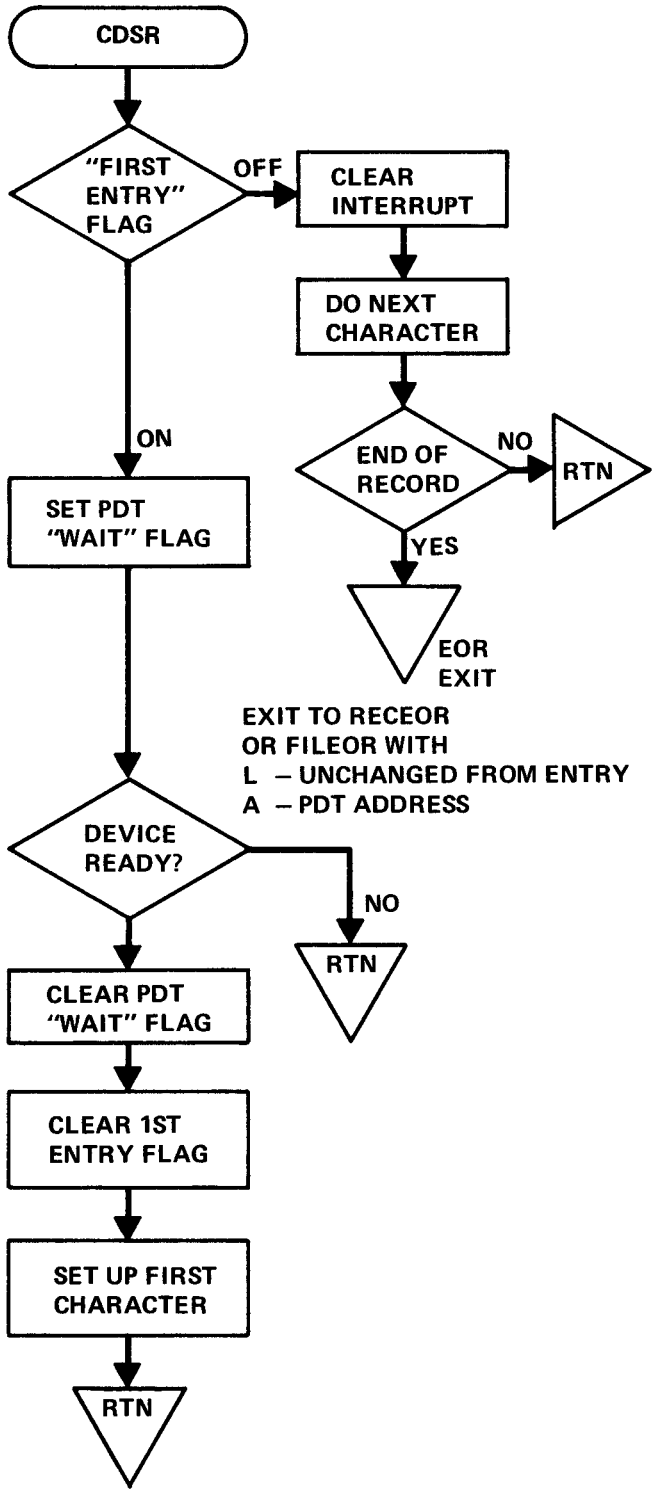


Figure 3-17 Typical CRU Device Service Routine.

exit; the TIMER routine will re-enter them periodically. Some of them which cannot run simultaneously call subroutine GONOGO to determine if they may run. They all exit to RECEOR or FILEOR after they have processed an entire record. The flow chart for a typical routine follows. Note that on entry:

L (Reg. 3) = Return address – 2

D (Reg. 4) = Temporary storage address

F (Reg. 6) = PDT block address

C (Reg. 7) = CRU base address

3-4.9.1 Interval Timer Service Routine (TIMER). The interval timer has an associated PDT block and an entry in the DSRAT but it is not a normal CRU device since it cannot be requested by a worker program (Figure 3-18). Its busy flag is always set and the routine is entered every time the timer counts through zero.

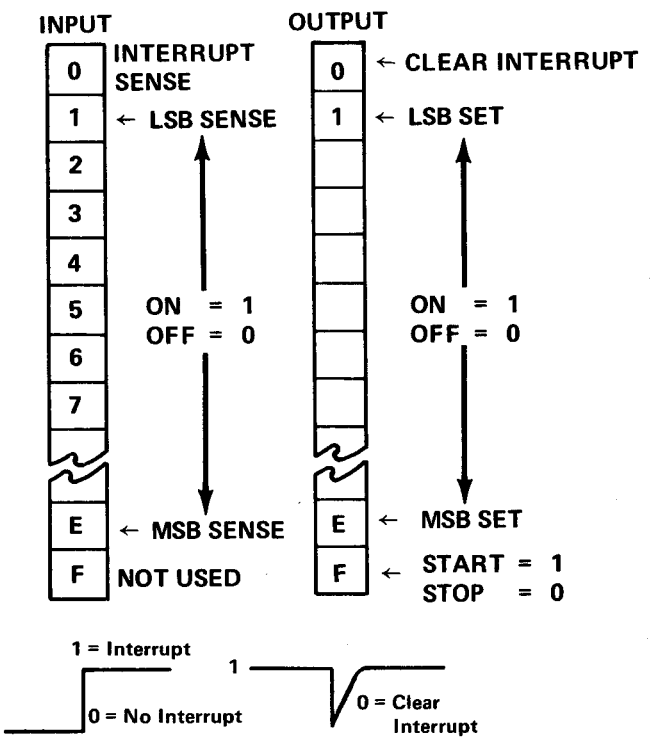


Figure 3-18 Interval Timer Operation.

The timer board itself can be preset with a count. This count is decremented by one every millisecond, and when the count becomes zero an interrupt occurs. In PAM, the timer is initially started by the time and date initialization portion of DIAGTB. The count is preset to SYSCLK.

When TIMER is entered it reads the timer count, adds SYSCLK to it enough times (N) to make it positive, and resets the count to that value. (Later N is used to decrement time delays.) The MILSEC counter is incremented by N*SYSCLK. This process prevents time loss when the timer interrupt is not answered immediately and it has counted below zero.

If the value MILSEC is now greater than 1000 (1 second is up), then it is decremented by 1000 and the time and date support package is entered. Otherwise, time delay processing is begun. Each task which is in a time delay has its TD count decremented by N. If this causes the count to go negative, then the TD flag in the WTB is cleared.

Next the wait flag in each PDT is checked and, if on, the corresponding device service routine is entered.

TIMER then returns to the CRU interrupt decoder.

Length: 101 words

App Time: $(183 + 23N_1 + 26N_2 + 21N_3)$ microseconds

where N_1 = Number of tasks

where N_2 = Number of tasks in a time delay

where N_3 = Number of PDT blocks

3-4.9.2 Electronic Data Terminal/Teletypewriter Service Routine (TTYSEG). The teletypewriter service routine, running under monitor interrupt control, will input or output binary or ASCII characters via the keyboard or paper tape (Figures 3-19 through 3-21). Required buffer and control information is taken from the user Physical Record Block.

The keyboard and paper tape hardware are each considered a physical device. This requires two device service routine entry points. Entry is controlled by IOQ.

Until the device service routine is called, the keyboard is open (bit 3 of temporary storage flag word is set). In this state, each character entered from the keyboard will be inspected and all rejected except an exclamation point, which will cause the DEBUG task to be bid.

On first entry for a keyboard input, the bell is rung to indicate ready for input.

On first entry to paper tape service routine for input, a reader-on character is sent.

On first entry to paper tape service routine for output, a punch-on character is sent.

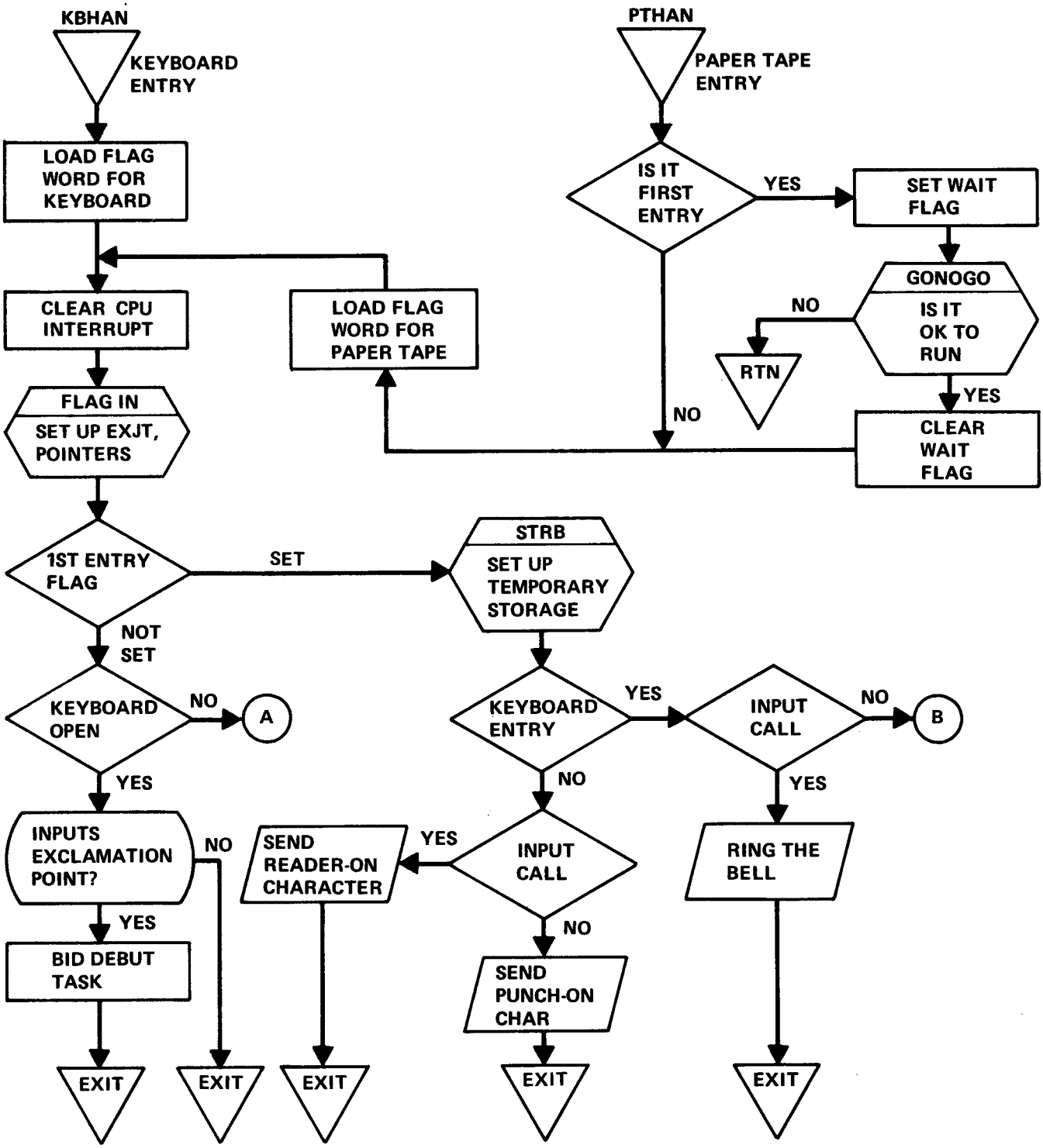


Figure 3-19 Teletypewriter Service Routine.

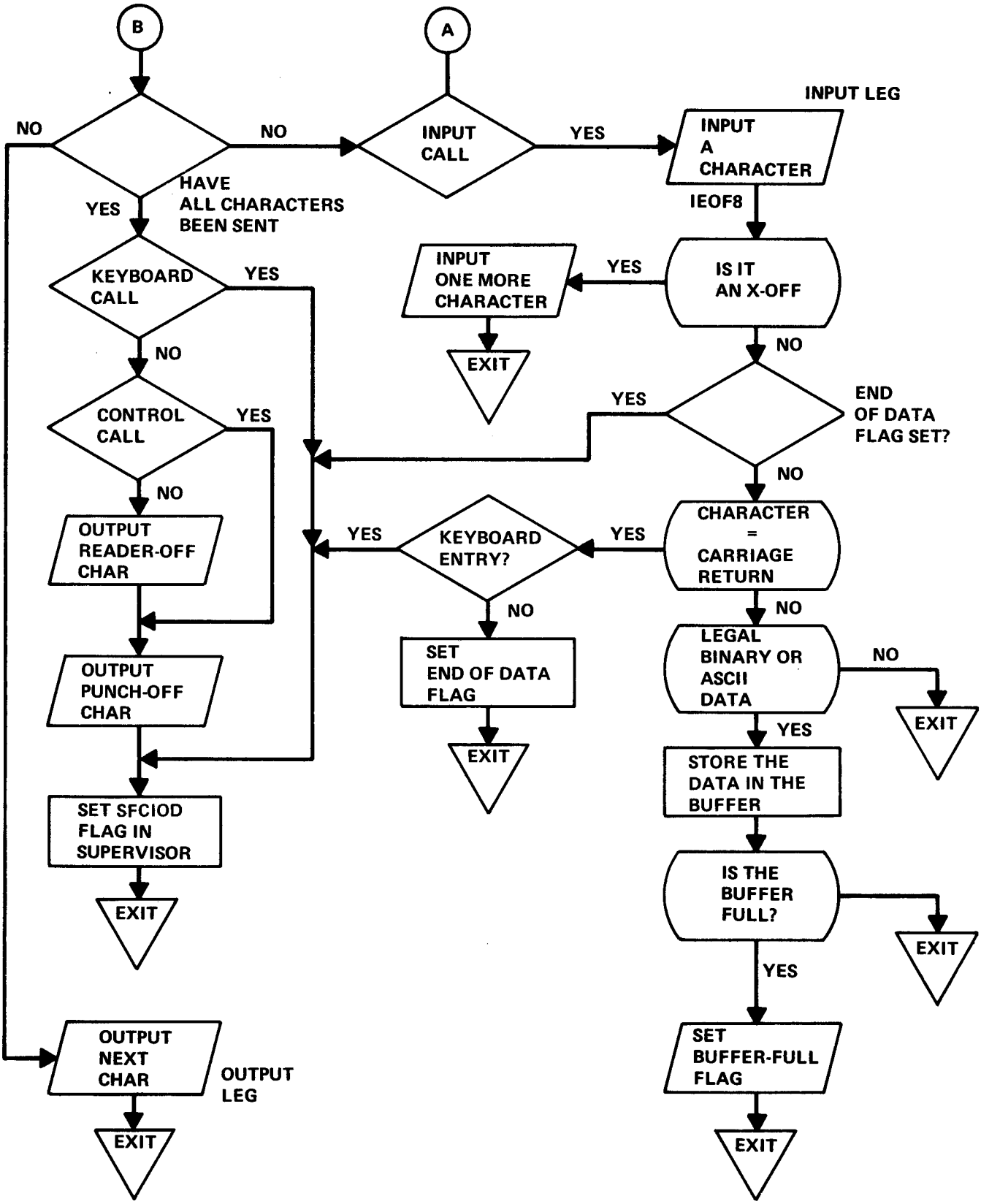


Figure 3-19 Teletypewriter Service Routine (cont'd.)

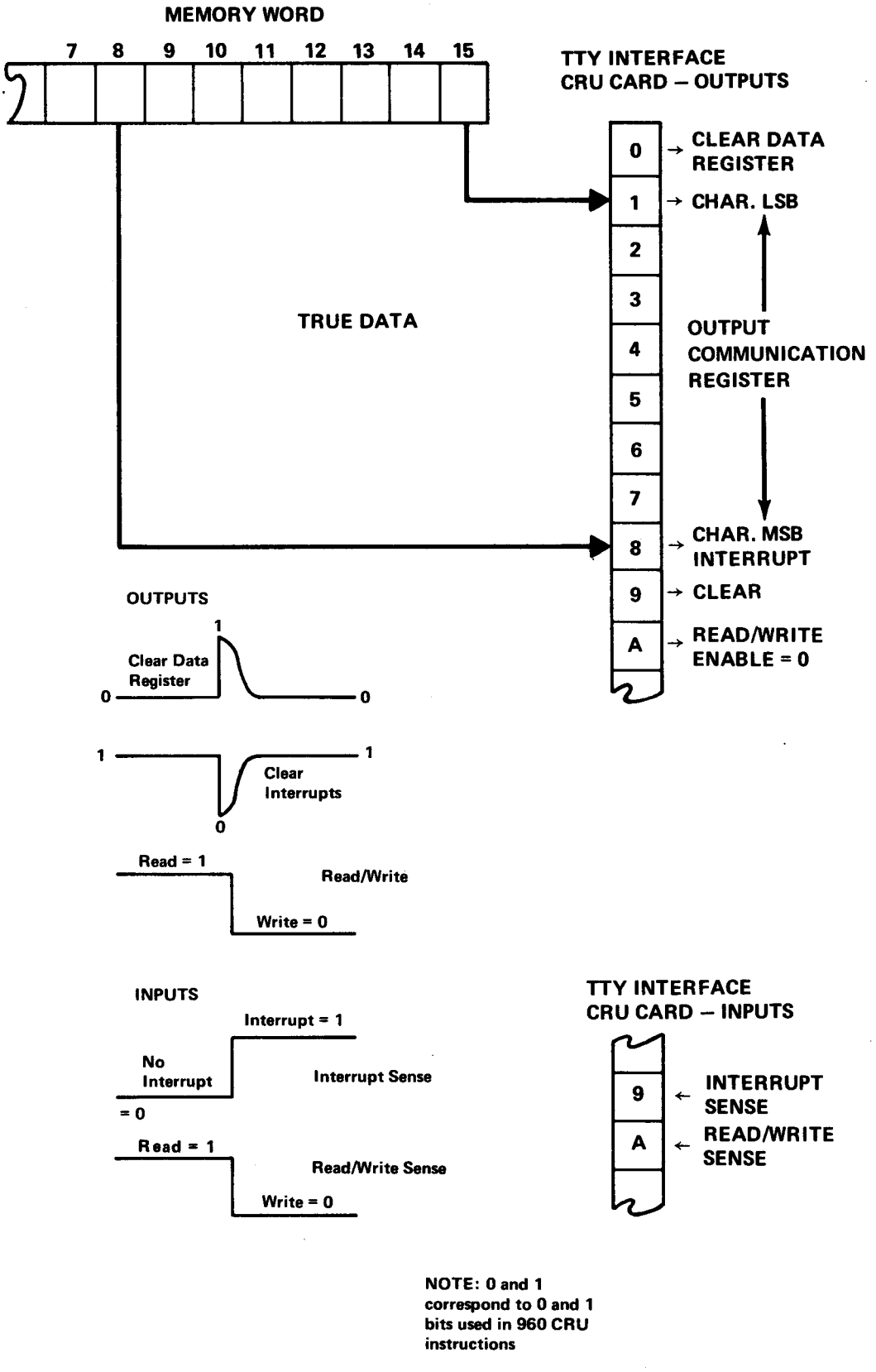


Figure 3-20 Teletypewriter Output Interface.

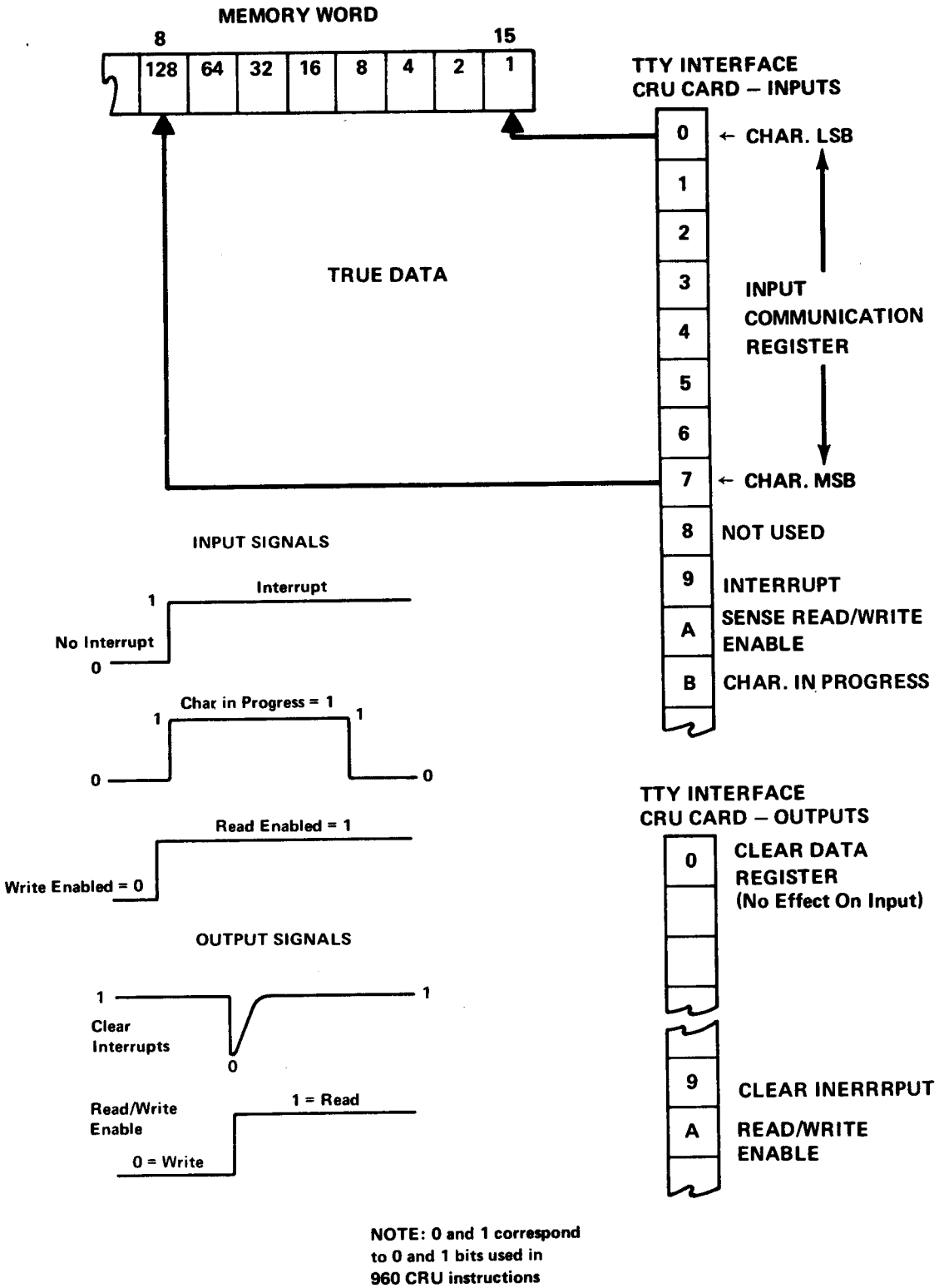


Figure 3-21 Teletypewriter Input Interface.

The following changes will be made to the user Physical Record Block by the device service routine:

a. Input buffer length will be truncated to X'03FF' or 1023_{10} characters.

b. At end of file, bit 2 of PRB flags is set to 1.

c. At end of record on input, the number of characters stored is placed in the PRB Record Length field.

No print line control is done by the device service routine. The user must send carriage returns and line feeds as needed. If a carriage return is sent as an ASCII output, a series of null characters are sent by device service routine to give the carriage time to complete its movement to prevent lost characters. If 72 characters is exceeded and no CR is sent, the device automatically carriage returns, causing a possible overprint.

Each physical device has its own four-word temporary storage area arranged as follows:

Word 0 = counter for characters input or output.

Word 1 = address of buffer word being processed.

Word 2 = user PRB flag temporary storage.

Word 3 = device service routine flag word. Initialized to 2001 for KB, 0001 for PT. DSR flag word bit arrangement:

Bit	Condition
0 = 1,	set when X-off sent at end of record on paper tape.
1 = 1,	set when P-off sent at end of record on paper tape.
2 = 1,	keyboard active, not a paper tape call.
3 = 1,	keyboard open, set to zero on first entry.
4 = 1,	All data stored on paper tape input. Wait for reader-off character.
5 = 1,	Remex (high speed paper tape reader) active
6 = 1,	Carriage return sent. Send four null characters for wait loop.
7 =	Not used.
8-11	Null character counter used with bit 6.
12-15	Counter used in packing and unpacking buffer words.

Buffer words for binary or ASCII input are packed. An ASCII word contains two characters; a binary word contains four characters. If a buffer word is not full, the data is left justified.

To punch leader, set control bit of PRB flags. Put number of frames wanted in PRB Record length field. Order of output on paper tape is PUNCH ON, LEADER, PUNCH OFF.

To punch binary data, set PRB flags for binary character output. Set Record length to number of characters* to be output, buffer address to address of first word of data block. Order of paper tape output is: PUNCH ON, DATA, READER OFF, PUNCH OFF. Each binary output digit is merged with X'0060'. For example, a binary one punch as X'0061'.

To input ASCII, set PRB flags for ASCII character input, set buffer length to the number of characters to be input, and set buffer address to the address of the first word of the storage area. Only data between X'0020' and X'005F' are stored. Carriage return or a full buffer halts input and terminates a keyboard call. If paper tape, tape continues to be passed for no data is stored until a reader-off character is sensed.

For binary input, set PRB flags for Binary Character input. Set buffer length to number of *characters to be stored. Set buffer address field to address of the first word of the storage buffer. Only data between X'0060' and X'006F' are accepted. A reader-off character or a full buffer stops data storage. Only a reader-off character stops tape movement. Any data between buffer full indication and a reader-off character are lost.

Length: 345 words

Average time per character: 200 microseconds

*A character is 8 bits, or 2 frames of tape.

3-4.9.3 HSRHAN – Punch Tape Reader Service Routine. HSRHAN (Figures 3-22, 3-23) is the entry point for the punch tape reader service routine. Since the data input by the device is subject to the same restrictions as that of the teletypewriter paper tape reader, and since it is assumed that all TI 960 systems have teletypewriters as the basic communication device, routines are shared where possible. This requires that the teletypewriter service routine be a part of a system using the high speed punch tape reader service routine.

The high speed punch tape reader service routine clears the interrupt, starts the tape movement, loads the data from the last read command, and stops the tape movement. There must be a delay of from 20 to 100 milliseconds between the start and stop commands. All other functions are handled by the teletypewriter service routine input routine.

Length: 38 words

Prerequisite: TTYSEG

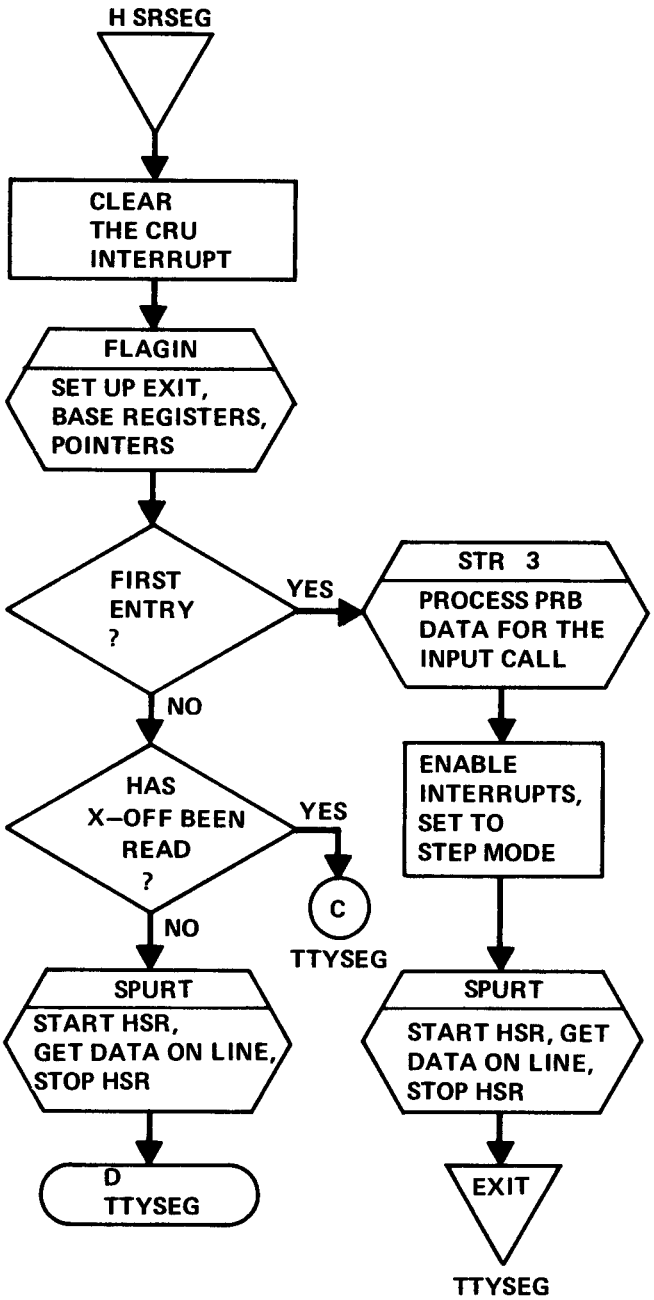


Figure 3-22 Punch Tape Reader Service Routine.

3-4.9.4 Card Reader Service Routine (CARDIN). The CARDIN portion of PAM is activated by IOQ (Figure 3-24). The CARDIN service routine performs either ASCII or binary read functions of any buffer length from 1 to 80 characters, inclusive. These options are specified in the Physical Record Block. A buffer length specification greater than 80 characters will be truncated in the device service routine to 80. A zero or negative buffer length will give unpredictable results and should be avoided.

The number of characters read from a card will be stored in the Physical Record Block (PRB) along with error and End-of-File (/*) flags.

When an ASCII read is being performed, any Hollerith code not found in the legal list (Figures 3-25, 3-26) will result in an error flag in the PRB and an ASCII block inserted in the buffer in place of the illegal character. This will not affect the reading of the remainder of the card.

Because of program structure, the card reader need not be READY prior to an I/O call. The device service routine waits for the operator to cause a ready state prior to its issuance of a feed card command.

The card reader requires approximately 185 milliseconds to read a card. During this interval, PAM masks CRU and DMAC interrupts.

If a feed or read error occurs, the message DIANOSTIC 00 will be output on the system logging device. The operator should reposition the card in the hopper and press the START button to re-read the card.

The GONOGO routine is used to prevent the card read from running (and masking interrupts) while other asynchronous devices are running (card punch or teletypewriter paper tape).

Length: 223 words

3-4.9.5 Card Punch Service Routine – CRU Interface (CDP000). The card punch service routine is first activated by the I/O call processor (IOQ) and subsequently operates under interrupt control (Figures 3-27, 3-28). Standard end of record processing returns program control to the worker task after the end of record is detected.

The number of characters transmitted as one record to the card punch is specified in the record length word of the PRB associated with the I/O call. End of record processing starts after the last character has been transmitted to the punch. Records should be at least one character long and no more than 80, otherwise results are unpredictable.

Data may be in either binary or ASCII format (See note) packed two characters per word. The PRB contains the appropriate data address.

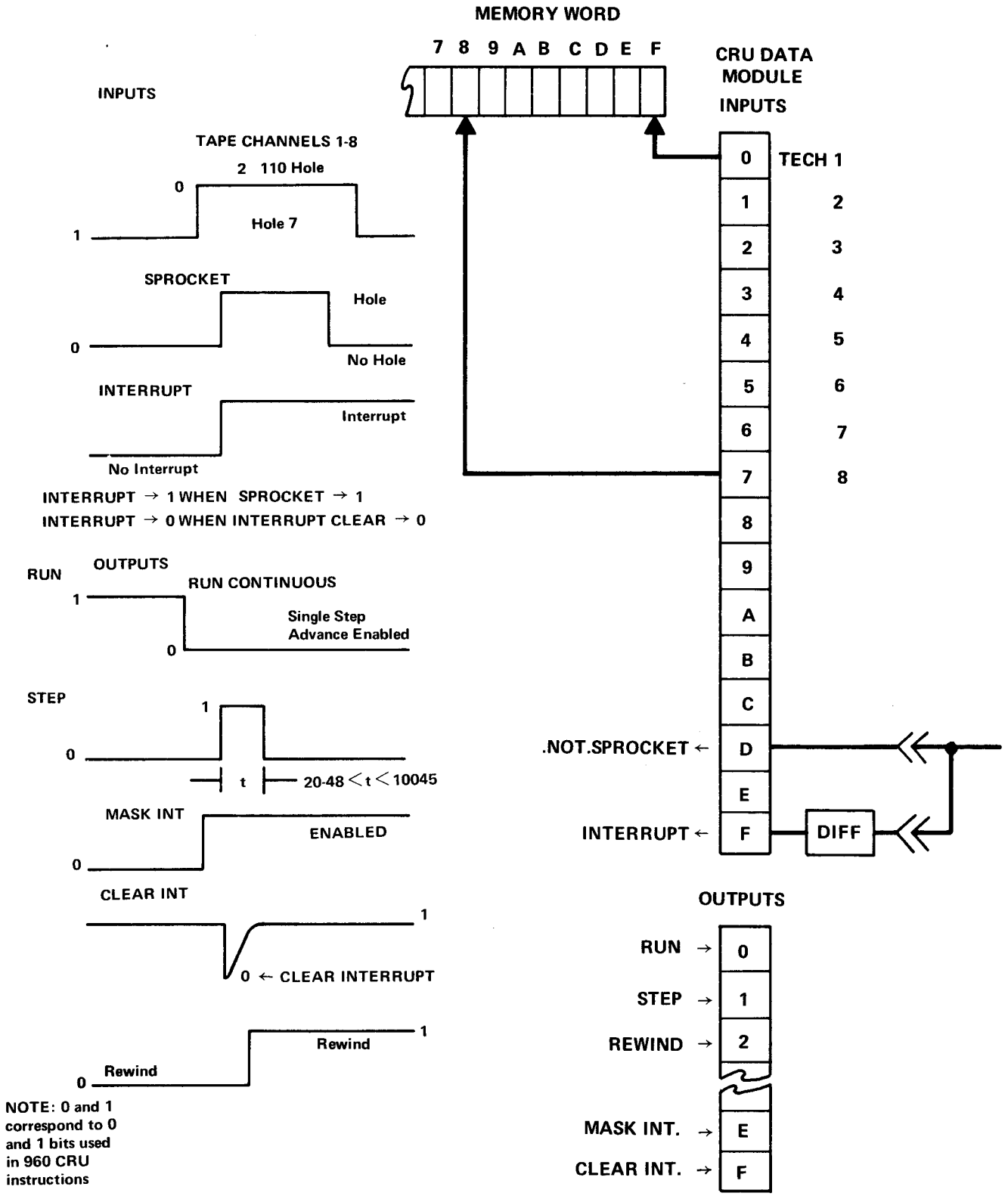


Figure 3-23 HSR Reader Interface.

ASCII	H. CODE	CHAR	ASCII	H. CODE	CHAR	ASCII	H. CODE	CHAR	ASCII	H. CODE	CHAR
20	No Punches	SP	30	0	0	40	8 · 4	@	50	11 · 7	P
21	11 · 8 · 2	!	31	1	1	41	12 · 1	A	51	11 · 8	Q
22	UD		32	2	2	42	12 · 2	B	52	11 · 9	R
23	8 · 3	#	33	3	3	43	12 · 3	C	53	0 · 2	S
24	11 · 8 · 3	$	34	4	4	44	12 · 4	D	54	0 · 3	T
25	UD		35	5	5	45	12 · 5	E	55	0 · 4	U
26	12	&	36	6	6	46	12 · 6	F	56	0 · 5	V
27	8 · 5	'	37	7	7	47	12 · 7	G	57	0 · 6	W
28	12 · 8 · 5	(	38	8	8	48	12 · 8	H	58	0 · 7	X
29	11 · 8 · 5	)	39	9	9	49	12 · 9	I	59	0 · 8	Y
2A	11 · 8 · 4	*	3A	8 · 2	:	4A	11 · 1	J	5A	0 · 9	Z
2B	12 · 8 · 6	+	3B	11 · 8 · 6	;	4B	11 · 2	K	5B	UD	
2C	0 · 8 · 3	,	3C	UD		4C	11 · 3	L	5C	UD	
2D	11	–	3D	8 · 6	=	4D	11 · 4	M	5D	UD	
2E	12 · 8 · 3	.	3E	UD		4E	11 · 5	N	5E	UD	
2F	0 · 1	/	3F	0 · 8 · 7	?	4F	11 · 6	O	5F	UD	

Figure 3-24 Legal Hollerith Codes.

PROGRAMMING NOTE:

ASCII character conversion is optional in the punch service routine. Use ASCII only if the capability is available.

The punch should be placed in the HOLD condition between files when no punch output is expected. This reduces punch mechanism wear and tear. It also prevents punch malfunctions and the need for frequent maintenance.

Mispunching of the first card can be avoided by always requesting a record to be punched before each file that can later be discarded. Punch malfunction during the first record is commonly caused by mechanical and electrical transients when the punch is started from the power off condition or from the HOLD condition.

Punching errors cannot be detected by the computer. Separate data verification is recommended such as use of a redundancy character or checksum.

Length: 72 words

Prerequisite: GETCO1

3-4.9.6 Paper Tape Punch Service Routine – CRU Interface (PTP000). The paper tape punch service routine (Figures 3-29, 3-30) is first activated by the I/O call processor (IOQ) and subsequently operates under interrupt control. Standard end of record processing returns program control to the worker task after end of record is detected.

Three punch functions are implemented and are requested by setting the appropriate bit in the PRB associated with the I/O call.

a. Punch ASCII, one frame per character.

b. Punch Binary, two frames per character.

c. Punch Leader, one null frame per character.

Data to be punched should be packed two characters per word. The number of characters is specified in the record length word of the PRB associated with the I/O call.

The punch motor power is off at all times when the punch is not in operation. When a record is punched the power is turned on. The punch power up sequence causes a delete code to be punched in the tape. No data is destroyed. Worker tasks should request leader before and after each file punched.

Length: 112 words

Prerequisities: GETCO1

3-4.9.7 Line Printer Service Routine – CRU Interface (LP000). The line printer service routine (Figures 3-31, 3-32) is first activated by the I/O call processor (IOQ) and subsequently operates under interrupt control. Standard end of record processing returns program control to the worker task after end of record is detected.

The number of characters transmitted as one record to the line printer is specified in the record length word of the PRB associated with the I/O call. The address of the data is also specified in the PRB. End of Record means the number of characters specified have been transmitted to the line printer.

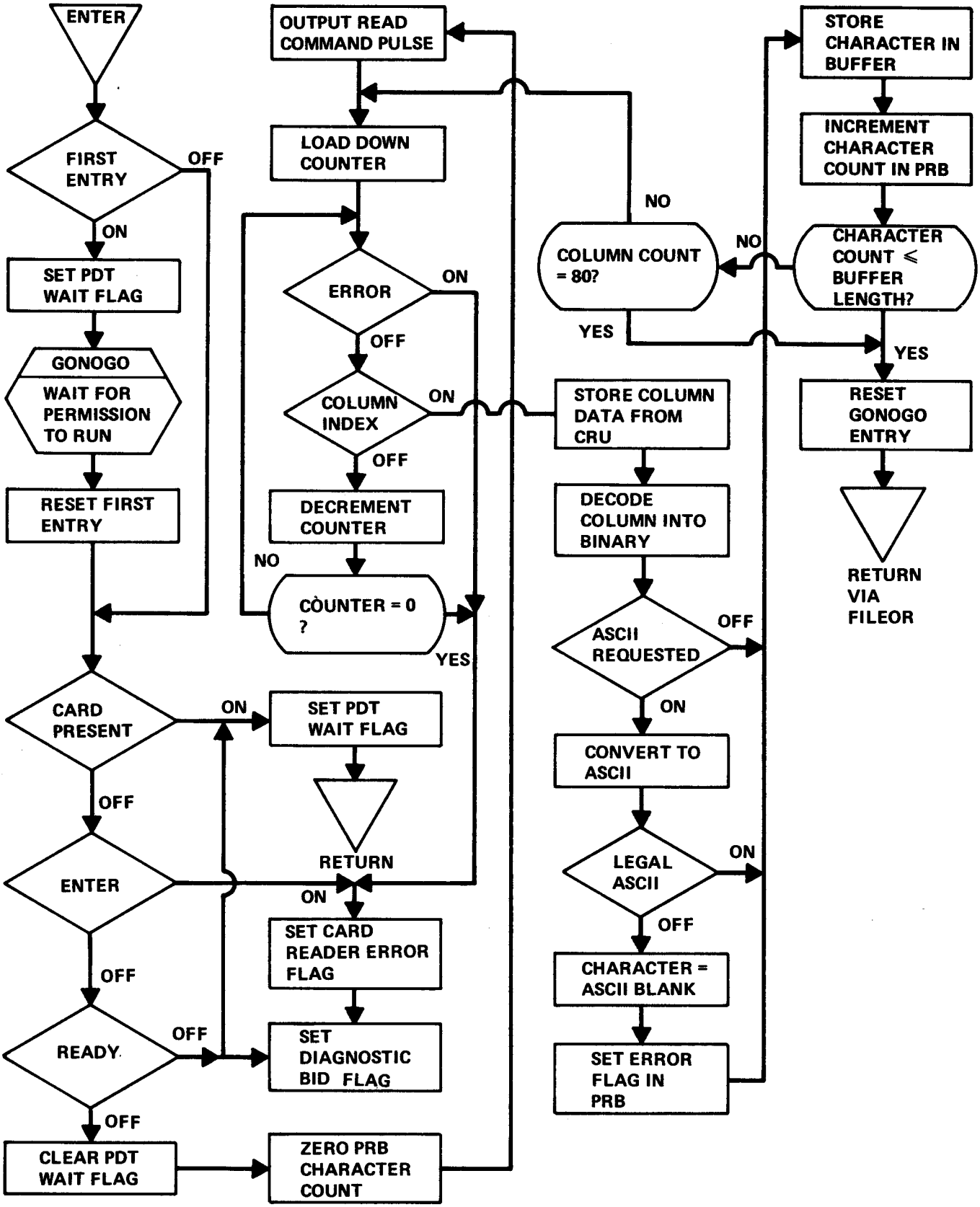


Figure 3-25 Card Recorder Service Routine.

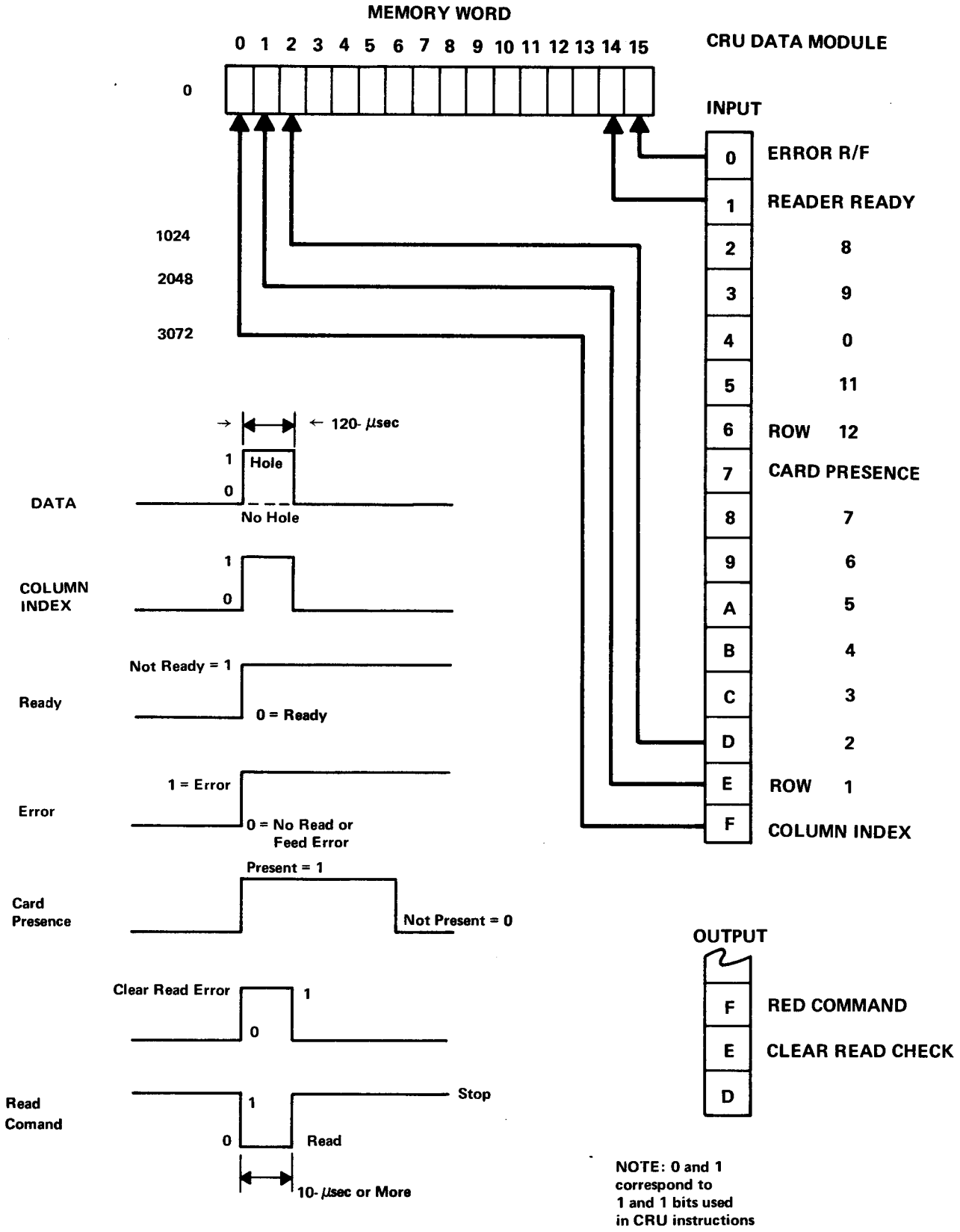


Figure 3-26 Card Reader Service Routine.

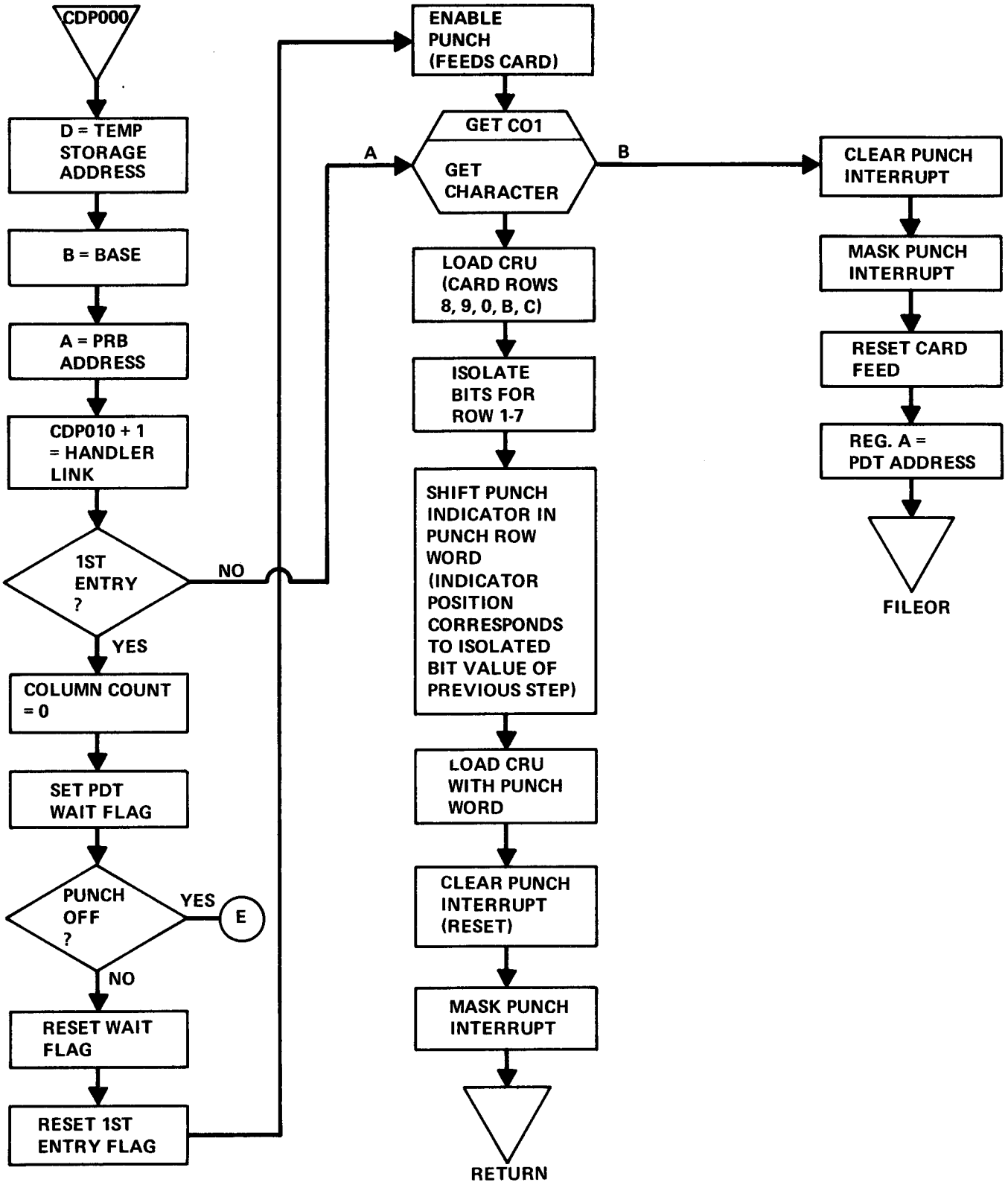


Figure 3-27 Card Punch Service Routine.

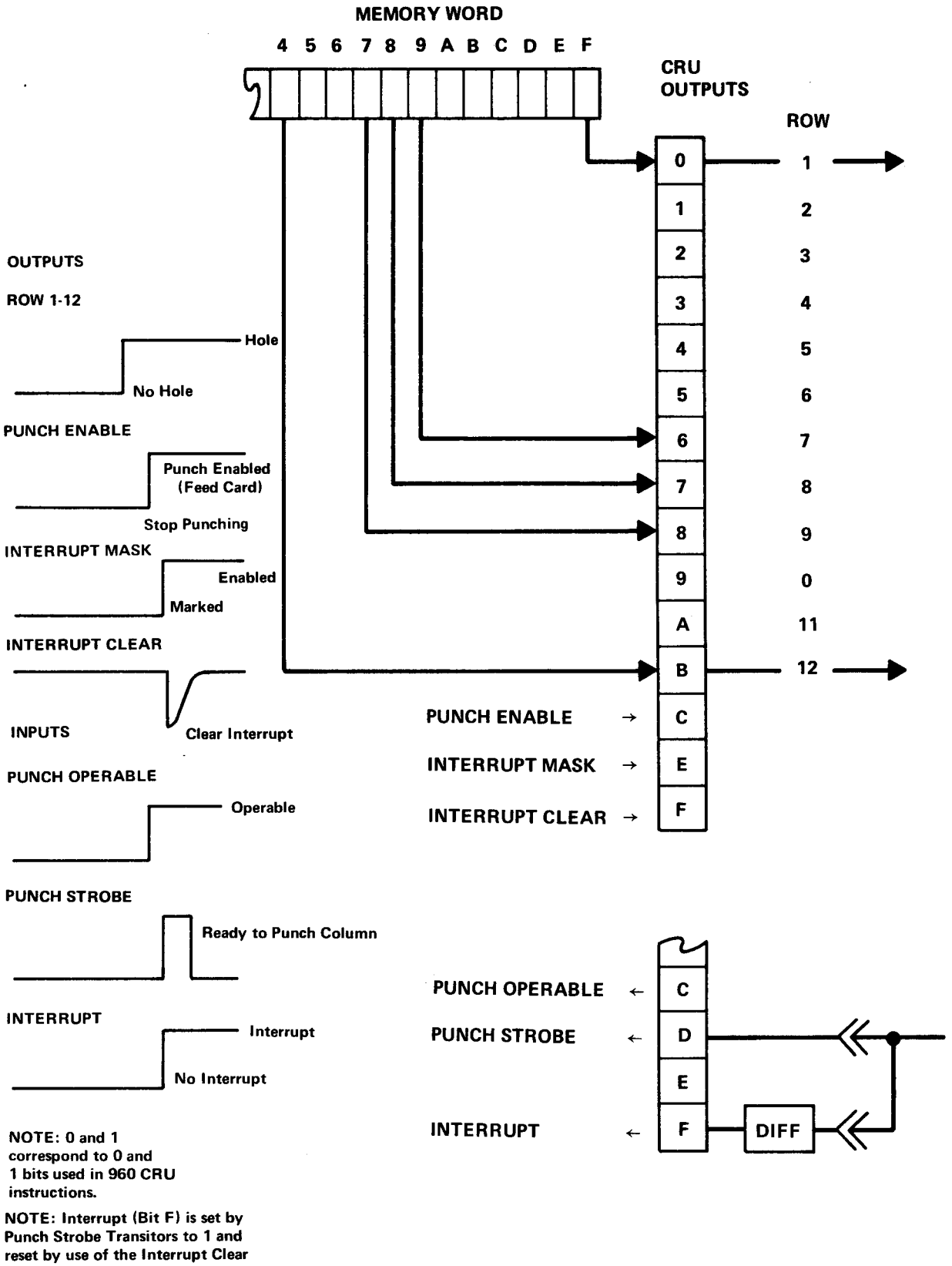


Figure 3-28 Card Punch Interface.

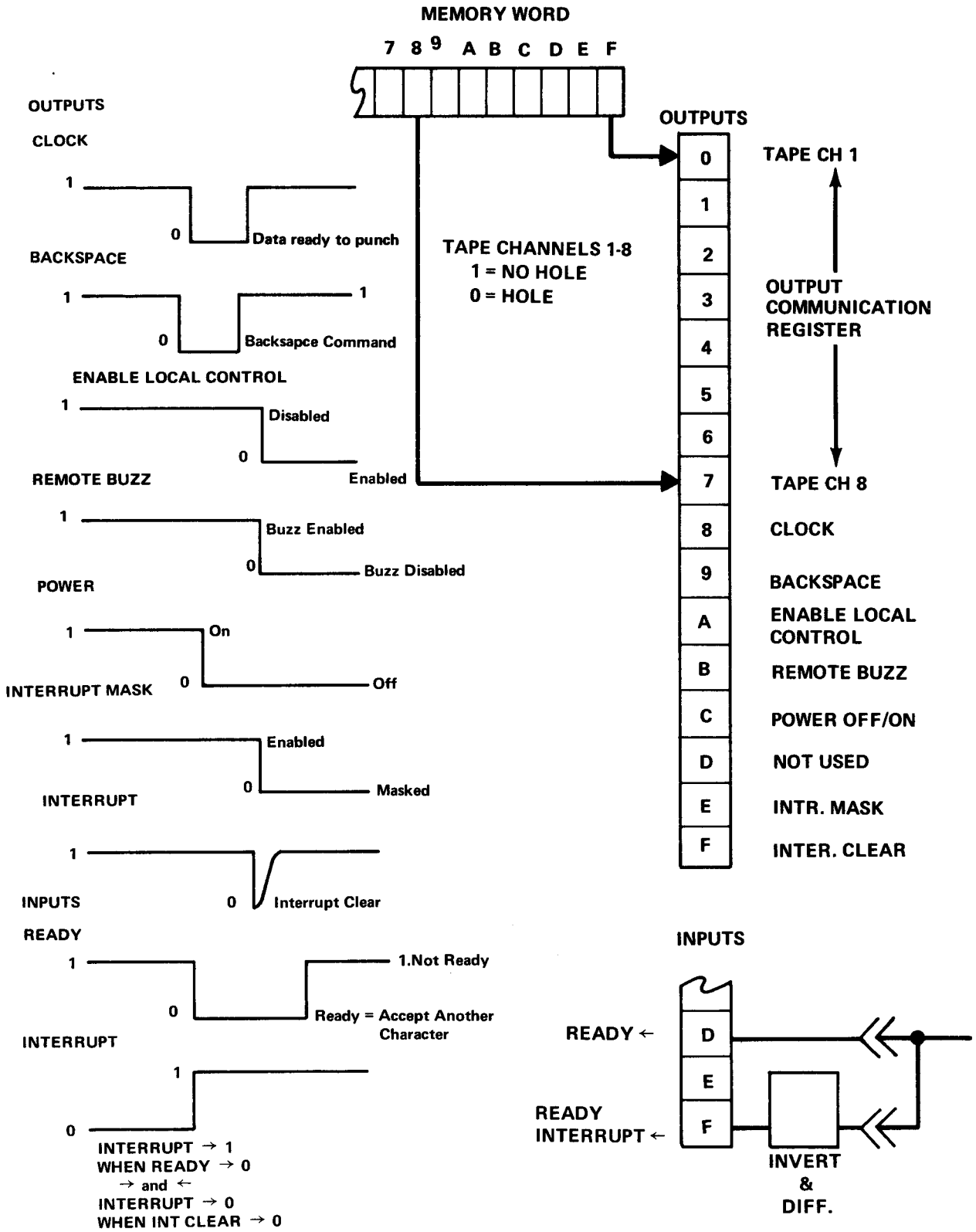


Figure 3-29 High Speed Paper Tape Punch Interface.

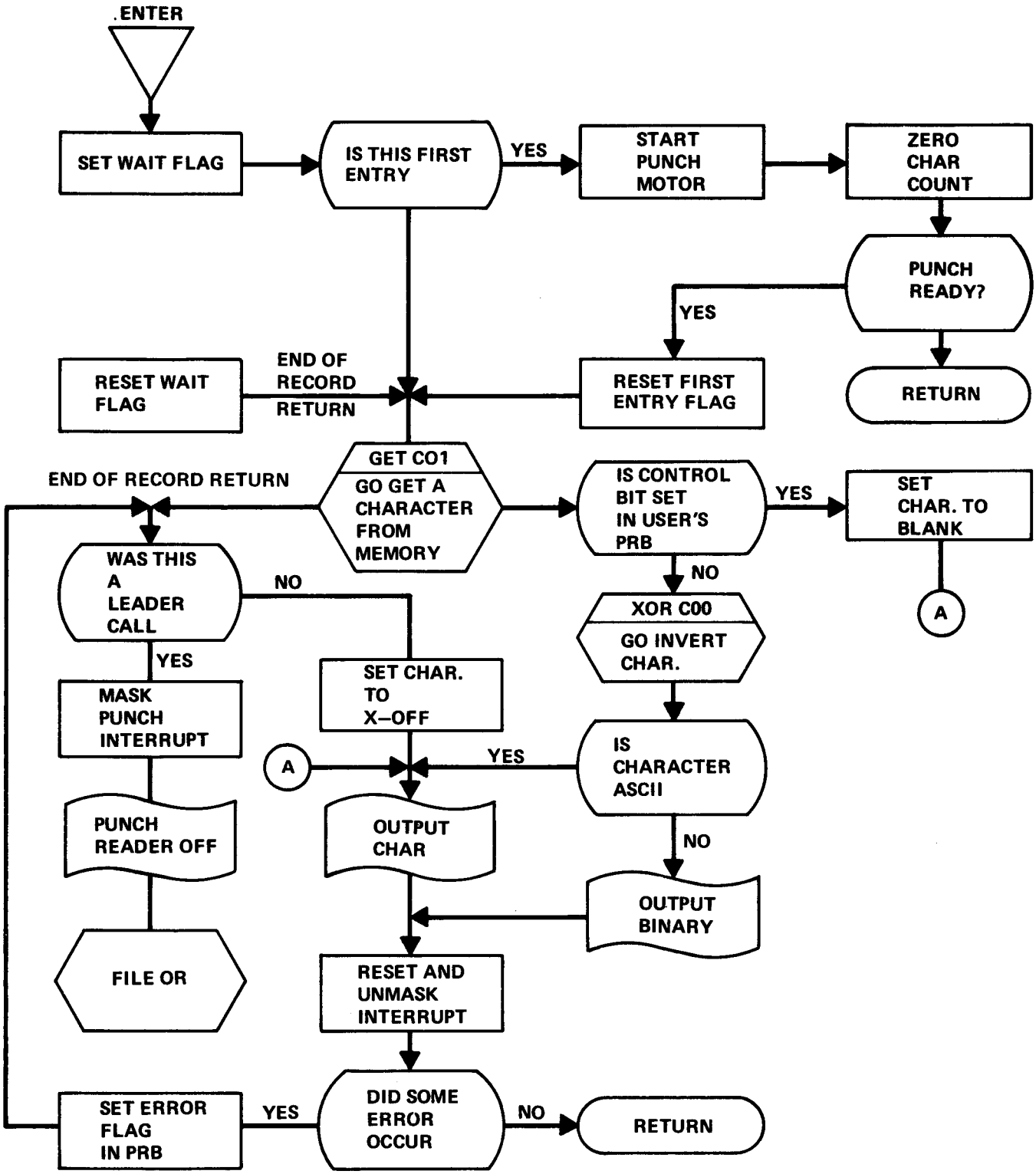


Figure 3-30 High Speed Paper Tape Punch Service Routine.

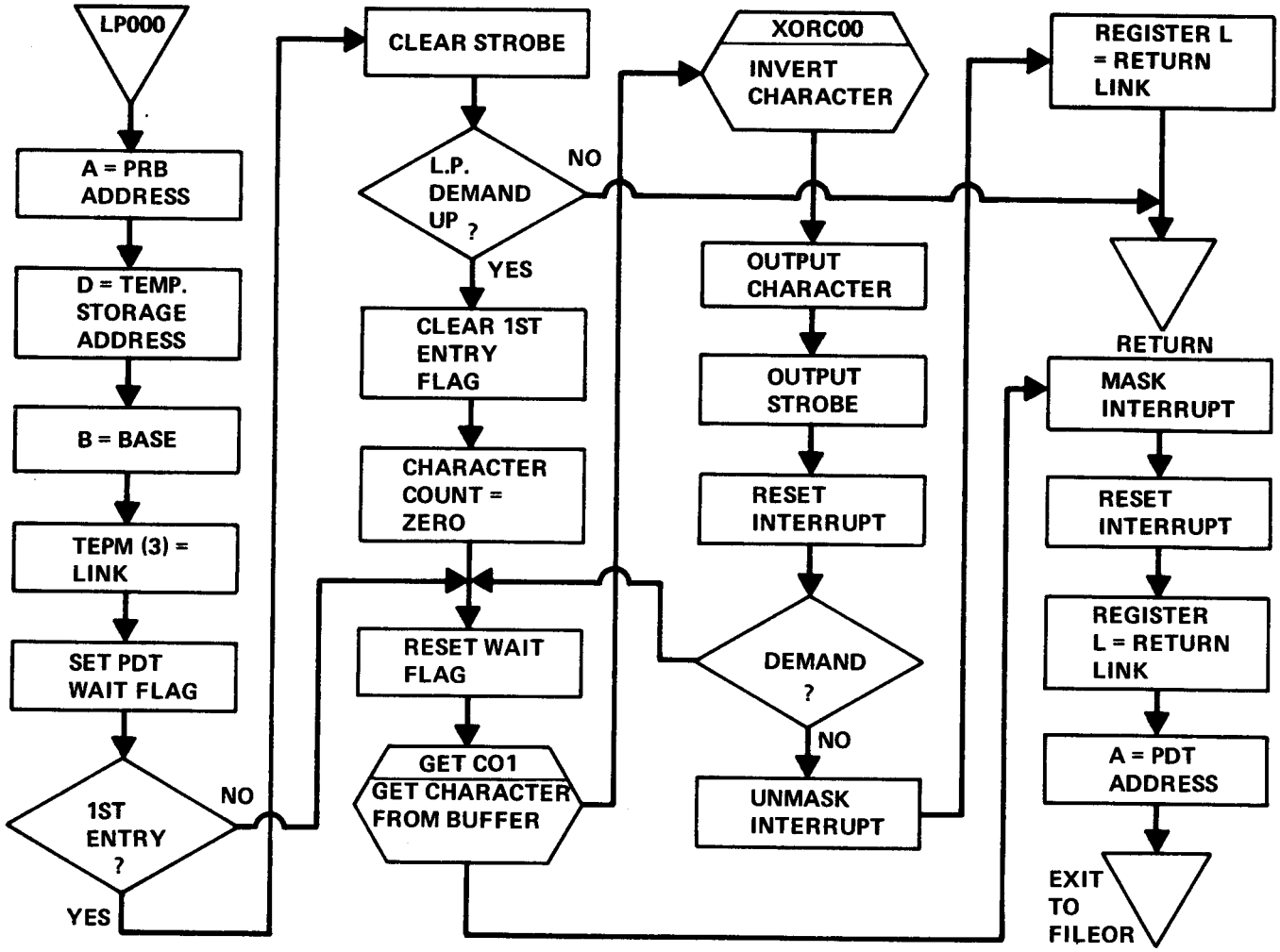


Figure 3-31 Line Printer Routine (CRU).

Data to be printed should be in ASCII format packed two characters per word. Line and format control characters can be inserted anywhere within the data to be printed. If more than 80 characters are sent between line feed control characters, overprinting will probably occur.

The line printer automatically skips to the top of the form after 62 lines have been printed.

Control characters are

ASCII Character	Function
X'000A'	Upspace one line and reset print position to first character.
X'008D'	Reset the print position to the first character.
X'000C'	Upspace the paper form to top of form.

If the line printer is off line when the line printer service routine is activated, the service routine sets the wait flag and exits. When the printer becomes ready, it continues.

Occasionally the line printer requires a Master Clear as well as an On-Line operation to obtain an operable condition.

Printing errors cannot be detected by the program.

Length: 51 words

3-4.10 DMAC SERVICE ROUTINES. The routines described in this section are for devices connected to the Direct Memory Access Channel (Figure 3-33). They have certain characteristics in common. Upon entry:

L (Reg. 3) = Return address – 2

D (Reg. 4) = Temporary storage address (if any)

F (Reg. 6) = PDT block address

C (Reg. 7) = DMAC part number (0-7)

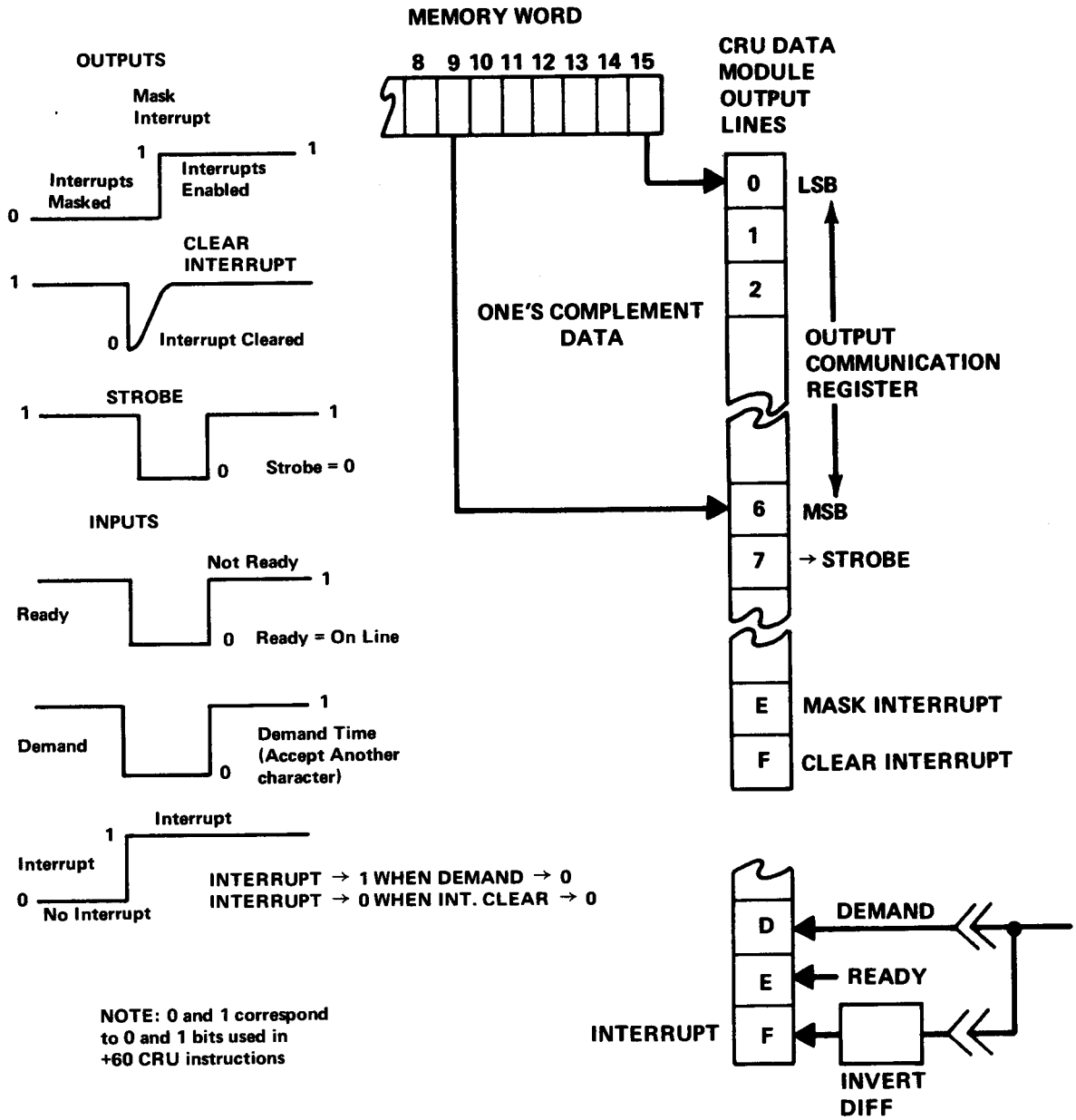


Figure 3-32 Line Printer Interface.

Device controllers attached to the DMAC process an entire record at a time. When finished they store a status word in location part number *2 plus X'0098'. Then a bit is set in location X'0096' corresponding to their part number and a DMAC interrupt causes a trap to location X'0092'. The DMAC interrupt decoder then enters the device service routine. This routine processes the end of record and exits through FILEOR.

3-4.10.1 Line Printer Service Routine – DMAC Interface (LPH). When the line printer is connected to the DMAC (Figure 3-34) its operation and the data transfer from the worker program are identical to the corresponding operations when using the CRU. However, the service routine is quite different. It merely sets up an initialization list and initiates the data transfer. When an interrupt is returned, it checks to see if the line printer was ready. If so, it assumes data transmission was correct. Otherwise, it sets the wait flag and attempts transmission again later.

Length: 44 words

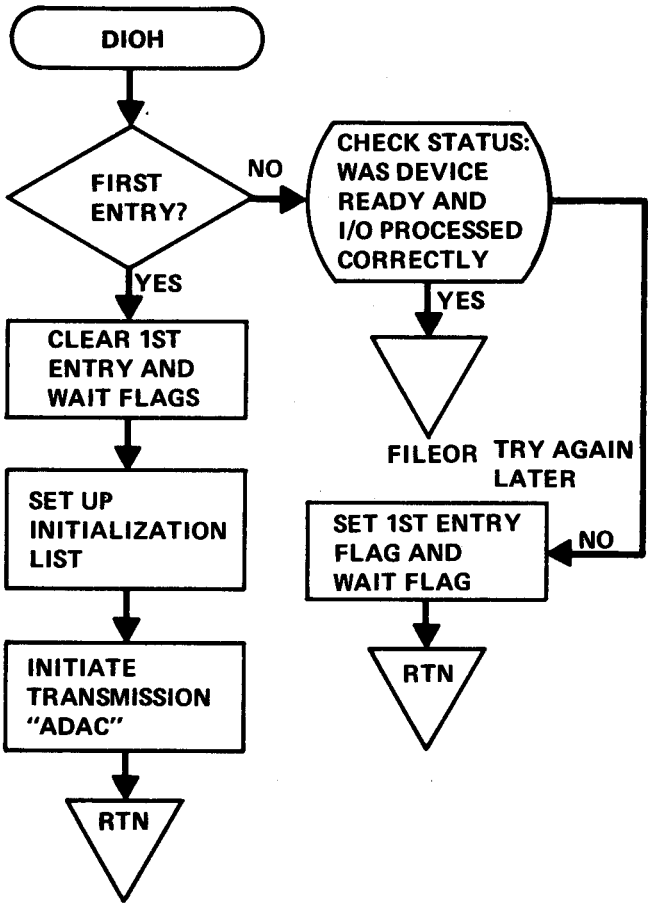


Figure 3-33 Typical DMAC I/O Service Routine.

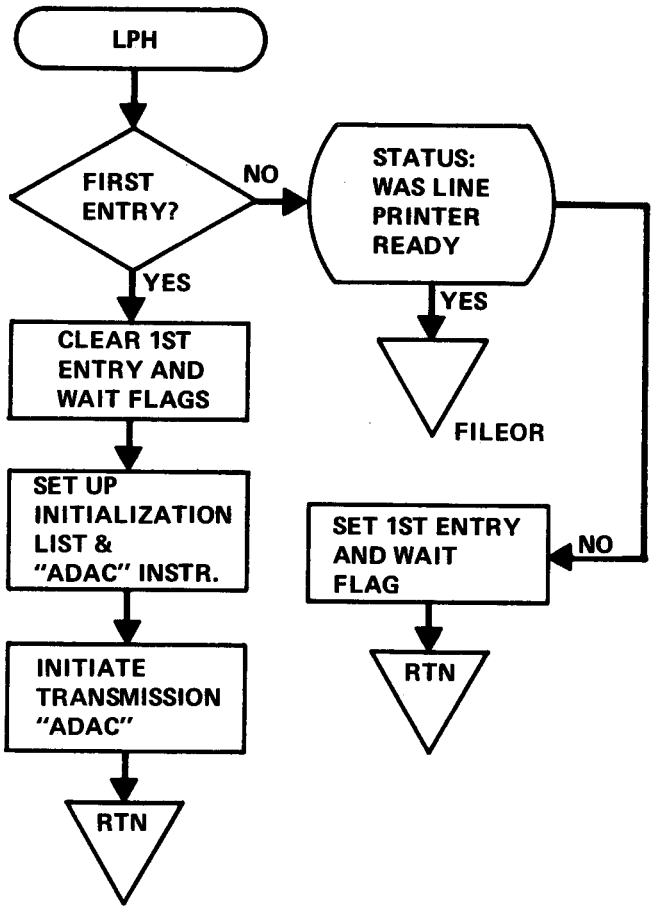


Figure 3-34 Line Printer Service Routine – DMAC Interface.

3-4.11 TIME AND DATE SUPPORT.

Real – DTDSEG Dummy – DTPSEG

The time and date support package contains three separate routines:

a. the time and date portion of the TIMER routine

b. the time and date initialization portion of DIAGTB

c. the get time and date supervisor call processor.

When the dummy segment is used, the routines are merely return branches.

The time and data portion of the interval timer routine (TMDAY) is entered once each second. It calculates machine duty cycle for that second and updates the maximum duty cycle if necessary. It then increments the SECOND counter and updates the MINUTE, HOUR, DAY, and YEAR counters as required. It then transfers back to TIMER for time delay processing (location TDLAY).

The initialization portion requests input of YEAR, DAY, HOUR, and MINUTE from the operator, converts them to binary and stores them. It then returns to DIAGTB at location DTRTN.

The get time and date supervisor call is described in paragraph 3-4.3.

Length: 167 words.

3-4.12 DIAGNOSTIC TASK (DIAGTB). The diagnostic task is always resident in PAM. Its task identifier is zero and its priority is always zero. Its three main functions are:

a. to initialize the time and date and start the interval timer

b. to print task error messages

c. to print general diagnostic messages.

If supervisor flag SETCLK is on when DIAGTB runs, it starts the interval timer and goes to DTPSEG for time and data initialization.

If any bits in word DGFLAG are set, it clears them and prints the message DIAGNOSTIC ON where N is the bit that was on.

If flag SFDSDG is on, a task has been disabled. The task identifier, error number, and address are obtained from DTFQUE and converted. A message of the form:

TASK XX, ERROR YY, AT ZZZZ

is printed.

Length: 149 words.

3-4.13 JOB CONTROL TASK (JCWTB). The job control task is always resident in PAM. Its task identifier is two and its priority is one greater than that of DEBUG. It performs all of the functions which are described in detail in paragraph 3-3.5. In general, the routines are straight forward, and only the main subroutines used are described here. Also, the interrupt trap location initialization routine is described.

JCRDBF is a 40-word buffer into which control cards are read. Prior to the reading of any cards, an initialization routine is stored here. When PAM is loaded, the bootstrap loader recognizes the end vector in the DEBUG routine, and transfers control to JCRDBF. This routine initializes locations X'0090' through X'0095', clears all device interrupts, and goes to SUPRST in SPB.

JCGWTB subroutine searches for a WTB with the specified task identifier or rank. If there is none, it returns to the calling location plus two. If it finds one, it returns to the calling location plus four with the WTB address in F and the next priority WTB address in E.

JCGPRO subroutine searches the task list for a procedure with the specified ID. If none is found it returns to the calling location plus two. If it finds one, it returns to the calling location plus four with the procedure entry point in E and the WRB address of a task to which it is attached in F.

JCLODR is a relocating loader which is used by both the load task and load procedure functions. It loads the object program in a similar manner to that of the bootstrap loader. In addition it verifies the redundancy character and checks load limits. It allows a load between location X'00A8' and the beginning location of PAM. End vectors are ignored.

DEFIOD is a subroutine (re-entrant to worker tasks) which is called both by Job Control and DEBUG to perform the DFIO function. It searches the Logical Device Table for an empty block, stores the device number and LUNO in it, sets the defined flag and returns to the calling location plus four. If the specified LUNO is already defined or the table is full, an error return is made to the calling location plus two.

RELIOD is similarly used to release LUNO. The block containing the specified LUNO has its defined flag cleared and its first word set to X'FFFF'. There is no error return. The routine always returns to the calling location plus two.

Length: 733 words.

3-4.14 DEBUG TASK (DEBUF). The debug task is always resident in PAM. Its task identifier is one. Its priority is set by the user via the DEBPRI EQU X card in SDB. It performs the functions described in detail in paragraph 3-3.6. All I/O except memory dump is performed on LUNO zero, the system logger. The memory dump is output on LUNO 9. The Job Control routines DEFIOD and RELIOD are utilized. There is no protection against illegal memory modification when using the LMHA function.

Length: 368 words.

3-5 CRU PROGRAMMING INFORMATION.

3-5.1 BASIC CRU. The Basic CRU consist of space (4 card slot) for 4 CRU Modules inside the Standard Central Processor unit chassis. Each Module has 16 input and 16 output line addresses. CRU Expansion is provided by connecting cables from the 4 basic card slots to CRU Expansion Chassis. Each Expansion Chassis provides space (16 card slots) for 16 CRU Modules. Up to four Expansion Chassis can be connected to each basic card slot in the central processor. A total expansion of 16 CRU Expansion Chassis or 4096 output and 4096 input line addresses results. Figure 3-35 illustrates the addressing scheme for the basic CRU and expanded CRU.

3-5.2 DATA MODULES – TTL COMPATIBLE. A data module provides direct input and/or output of data to external devices through the CRU. Each module has the capability of sixteen input lines and sixteen output lines (Figure 3-36). The line may be addressed individually or in fields from one to sixteen, either in input or output. A jumper modification can be made to the module to allow it interrupt capabilities. If this is done, input line fifteen is used to sense for interrupt presence, output line 14 is used for masking of the interrupt, and output line 15 is used for clearing the interrupt.

Outputing a logical 0 on a data line produces +5 volts on the output side of the module. A logical one produces 0 volts. On inputing or sensing lines, a +5 volts will cause a logical zero to be sensed or input and a 0 volts will cause a logical one to be sensed in input.

3-5.3 DATA MODULE – CONTACT CLOSURE INPUT AND OUTPUT. The contact closure module provides eight bits of relay buffered digital input and output CRU

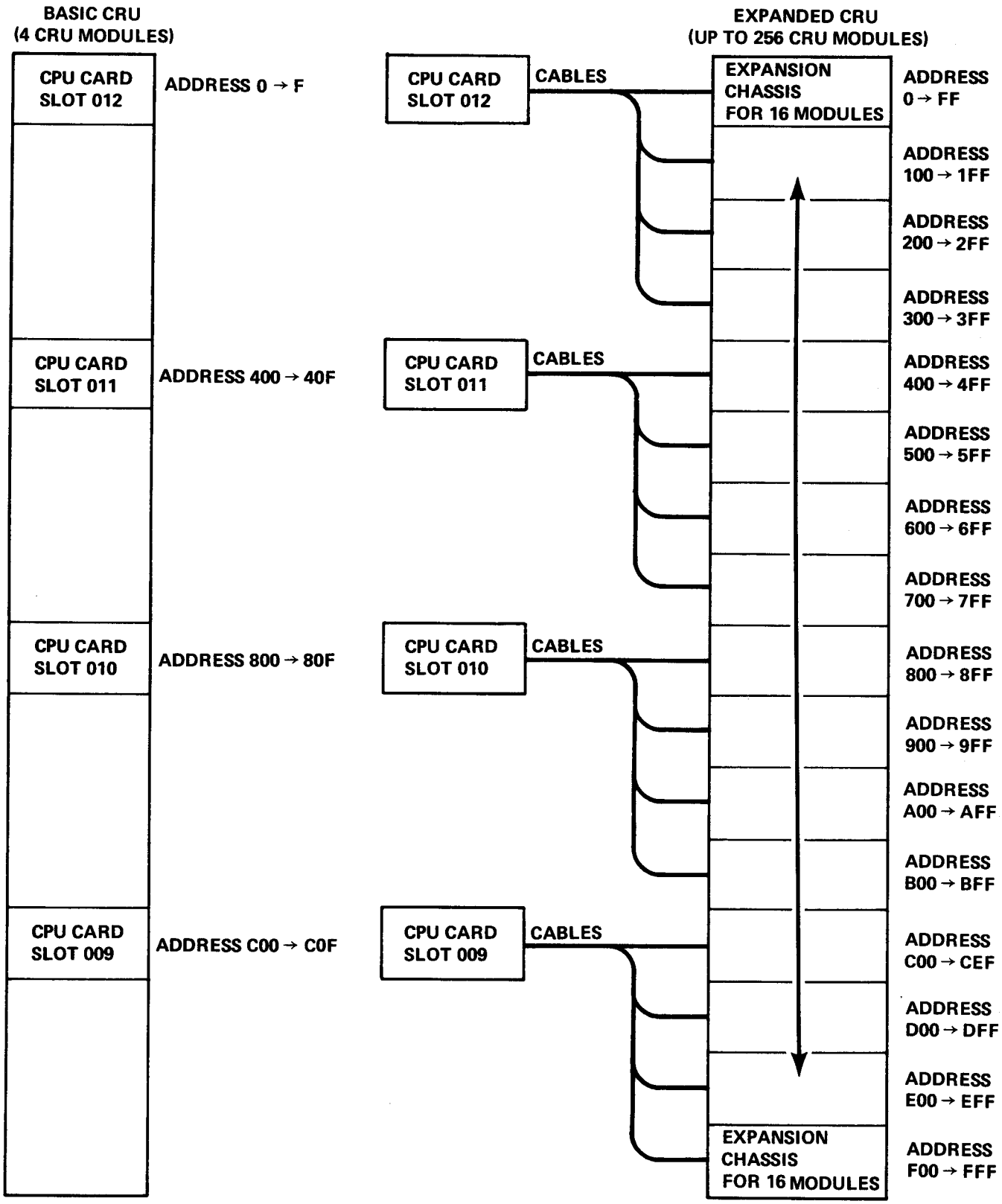


Figure 3-35 CRU Addressing Scheme.

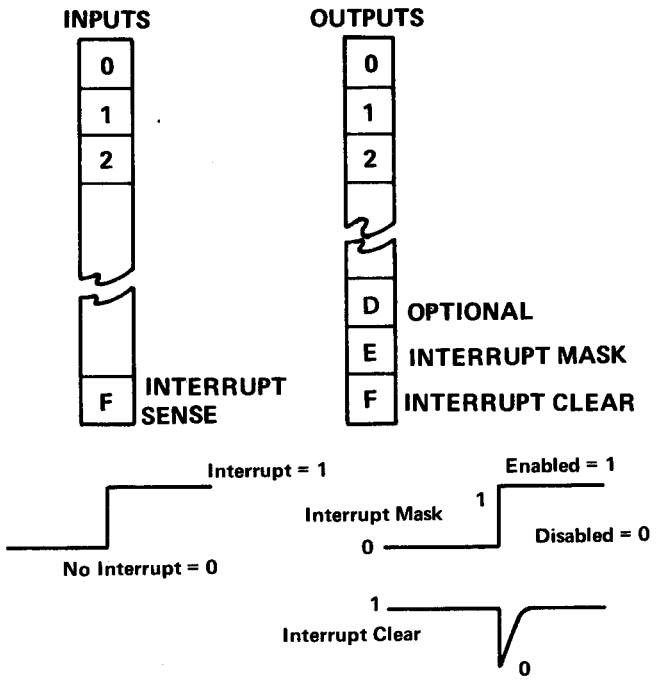


Figure 3-36 Data Modules.

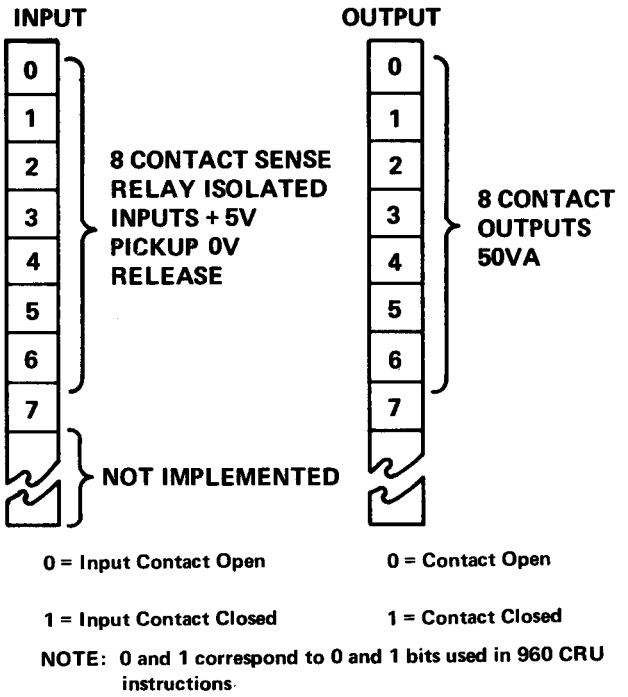


Figure 3-37 Data Modules – Contract Closure I/O.

addresses and operations are shown in figure 3-37. This module can be used in any expansion chassis.

Programmable functions are:

a. Sense Contact Input

Example

```
BBNE   C1,1,RDY1        IF C1 = 0 GO TO RDY1
```

b. Operate Output Contact

Example

```
SETB   OC1,1            CLOSE OUTPUT
                        CONTACT

or

SETB   OC1,0            OPEN OUTPUT
                        CONTACT
```

3-5.4 INTERRUPT MODULE. The interrupt module provides eight interrupts. CRU input and output addresses and functions are shown in Figure 3-38. This module can be used either in the CPU or in an expansion chassis.

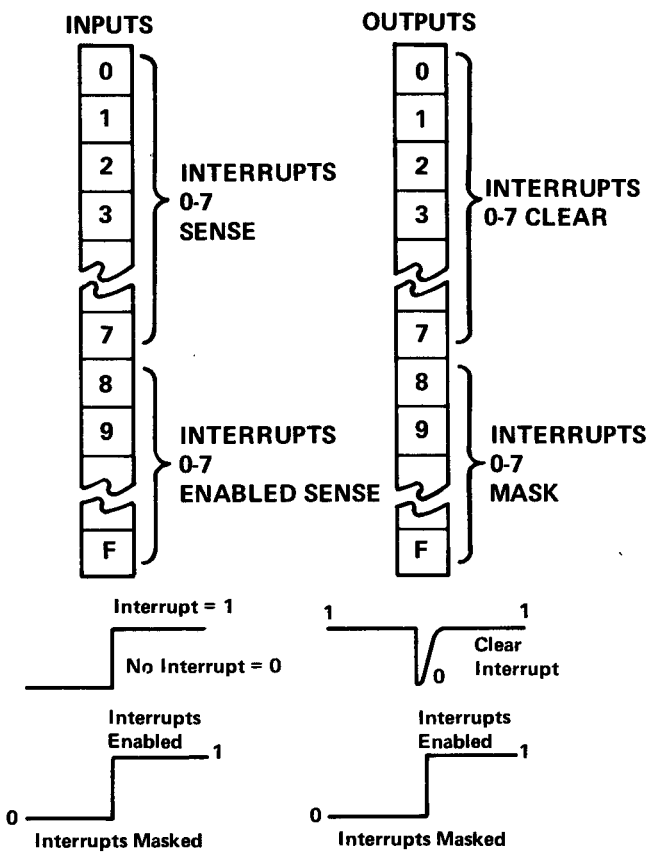


Figure 3-38. Interrupt Module.

Programmable functions are

a. Interrupt Sense

Example

```
BBNE   INT1,0, ISUB1   IF INTERRUPT
                       GO TO I SUB.
```

b. Interrupt Mask Sense

Example

```
BBNE   INT1M,0,MASK1 IF INTERRUPT
                     MASKED GO
                     TO MASK1
```

c. Interrupt Clear

Example

```
SETB   ICLR1,0         CLEAR INTERRUPT 1
```

d. Interrupt Mask

Examples

```
SETB   IMASK1,0        MASK INTERRUPT
```

or

```
SETB   IMASK1,1        UNMASK
                       INTERRUPT
```

3-5.5 ANALOG TO DIGITAL CONVERTER MODULE

The 960 A/D converter module provides a simplified means by which the programmer can converse with analog type equipment. To input a value from the module, a total of two instructions are required, one to request a sample and one to input the digital value. From a programming point of view, there are two configurations of the module, one which does not have input multiplexing capabilities and one which does. The same set of instructions can be used to program either, but on the module having input multiplexing capabilities, a specific input channel must be addressed to assure sampling of the desired analog input.

In the modules not having multiplexing capabilities, a sample and conversion is initiated with the outputing of a logical one to any one of the sixteen output addresses (0-15 plus CRU base address of the module). This may be done with either a SETB or LDCR instruction. In the modules having multiplexing capabilities, a sample and conversion is also initiated with a SETB or LDCR instruction, but the instruction must output a logical one to the output address corresponding to the desired input channel, e.g. a SETB instruction outputing a logical one addressed to output address six (plus CRU base address) will cause a sample to be taken on input channel six and a conversion to be made of that sample. During the sampling and conversion, the Busy/Ready Signal will go to a logic 0 and remain in that state until conversion is complete, at which time the signal will go to a logic 1. This signal is sensed by addressing input line fifteen (plus CRU base address). (See Figure 3-39). Conversion time for those modules without multiplexing capabilities is 70 microseconds maximum, with multiplexing requires 80 microseconds maximum. When the

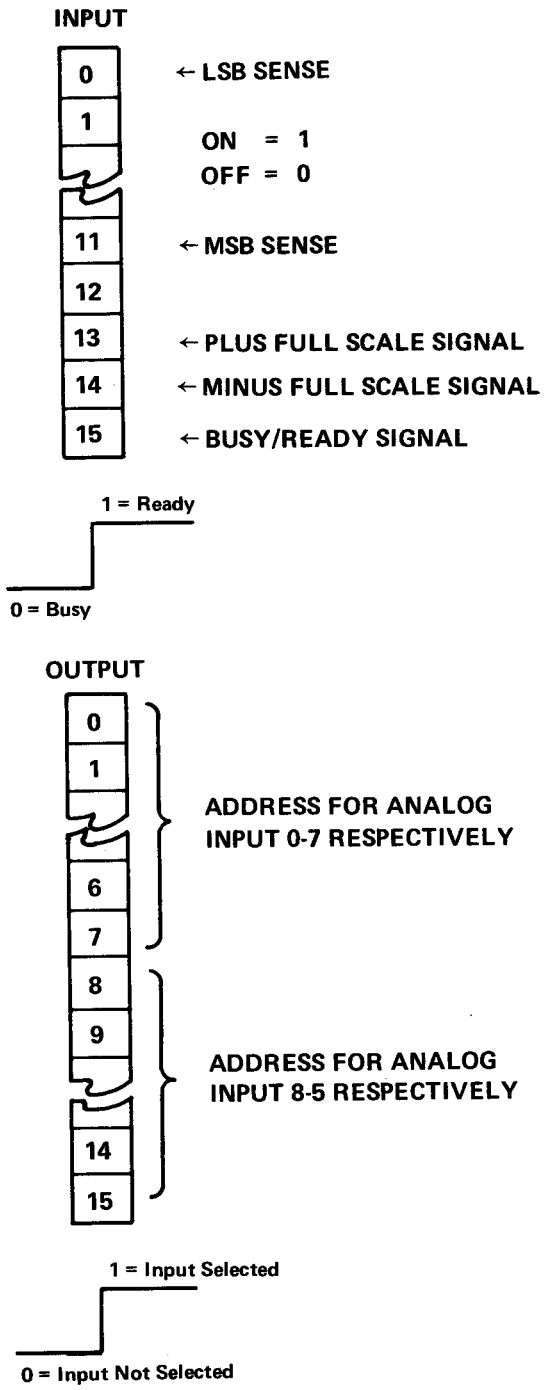


Figure 3-39 A/D Converter Module.

Busy/Ready signal goes to a logic one, indicating conversion complete, the digital value, a two's complement binary integer, can be input with a STCR instruction addressed to line zero with a field width of 12 bits. The number may be input anytime after the Busy/Ready signal has gone to a logic 1 and before another sample and conversion is requested.

Input line addresses 13 and 14 are used for signals which indicate a plus full scale or minus full scale binary output respectively. The signal at line address 13 will be a logic one if the A/D converter output contains the plus full scale binary output. The signal at line address 14 is logic one for a minus full scale binary output. If the converter number is some intermediate value, both signals will be a logic zero.

3-5.6 DIGITAL TO ANALOG CONVERTER MODULE The D/A converter module provides a simplified means by which the programmer can have a digital value converted to an analog output (Figure 3-40). The module can optionally have one, two, or three D/A converters, thus allowing multiple analog outputs with one instruction.

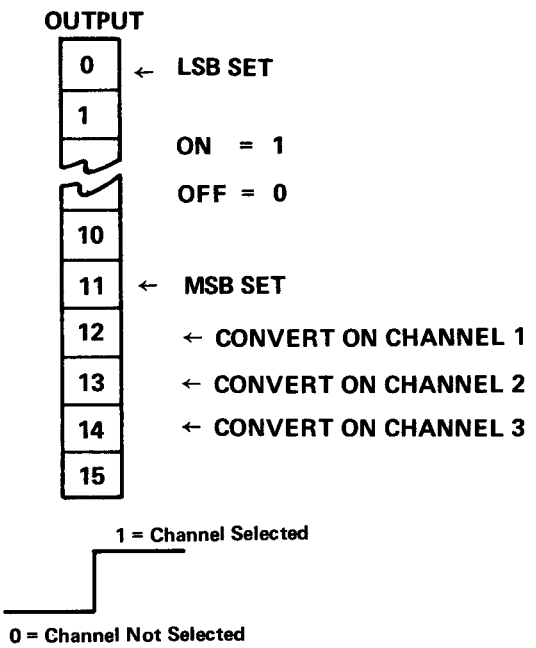
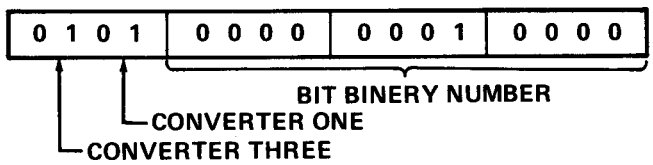


Figure 3-40 D/A Converter Module.

The output to the module is a 12 bit number containing a normal 960 two's complement binary number. To output the number and execute a conversion, a Load CRU (LDCR) instruction is used to output a 13, 14, or 15 bit field containing the 12 bit number to be converted. The CRU address for the instruction is line zero (plus CRU Base Address). To select the desired channel or channels, a logical one is output to the appropriate line or lines. For example, the following bit configuration would cause a 16 to be converted on converters one and three.



3-5.7 INTERVAL TIMER OPERATION. The interval timer is a 14 bit down-counter which can be initialized by the programmer. The count can be read at any time. It generates an interrupt when it passes through zero. The count will always be in the range -8,192 to +8,191. The time interval for each count is 1, 2, 4, or 8 milliseconds. It is selectable by a jumper wire but not by programming.

This module provides an easy means for the programmer to generate an accurate time window for enabling or masking system stimuli or to measure accurately the duration of an event.

Programmable functions – Refer to Figure 3-41.

a. Initiate count and clear interrupt

Example:

```
        LDCR    (0,16), STCLCK
         .
         .
         .
STCLCK  DATA    COUNT*2 + X'8000'
```

Note that using COUNT + 2 sets bit O to a 0, clearing the interrupt.

b. Read Count

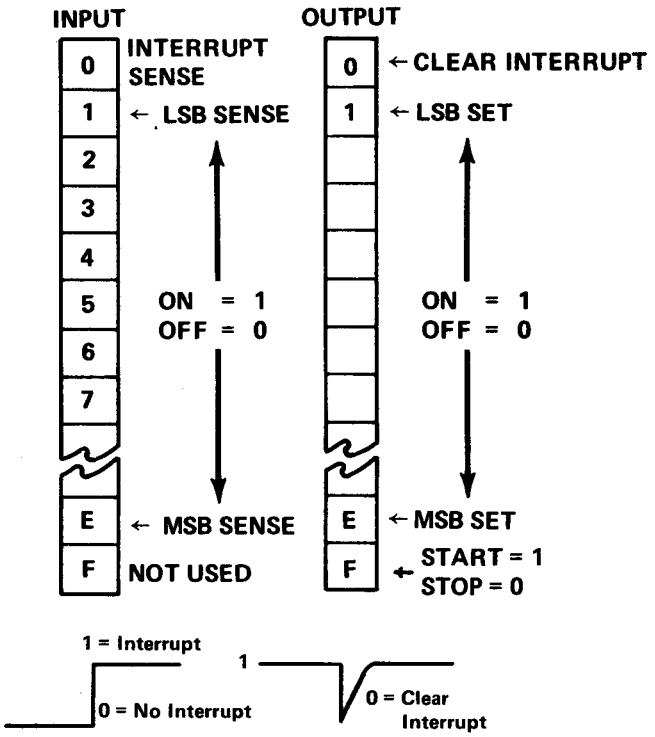
Example:

```
        STCR    (1,14), VALUE
         .
         .
         .
VALUE   RES     1
```

c. Reset count without stopping clock

Example:

```
        LDCR    (0,15), RESET
         .
         .
         .
RESET   DATA    COUNT*2
```



COUNT RATE = 1000 COUNTS/SEC INTERRUPT OCCURS ON ZERO COUNT. ALL FOURTEEN DATA BITS MUST BE OUTPUT AT THE SAME TIME. THE INTERRUPT CANNOT BE RESET WHEN THE COUNTER IS AT ZERO.

Figure 3-41 Interval Timer Module.

3-5.8 TELETYPE INTERFACE MODULE This CRU module provides the interface to Texas Instruments Silent 700/20 electronic data terminal. A functionally similar module is also provided for the Teletype Corp Model ASR-33 teletypewriter.

Programmable functions: Refer to figures 3-42 and 3-43.

1. Output to data terminal

Example:

SETB	TPO,1	MODE = SEND
SETB	TP9,0	RESET CHARACTER IN SIGNAL
LDCR	(1,8), CHAR	TRANSFER CHAR TO COMM REG
BBNE	TP9,1,$	WAIT FOR CHAR TO COMPLETE

2. Input from data terminal

Example:

SETB	TPO,0	MODE = RECEIVE
SETB	TYP9,0	RESET CHARACTER IN SIGNAL
BBNE	TP9,1,$	WAIT FOR CHARACTER IN SIGNAL
STCR	(0,8),CHAR	TRANSFER CHARACTER FROM COMMUNICATION REGISTER TO MEMORY

Bit No. 9 may also be used to generate an interrupt when the input or output function is complete.

3-5.9 PULSE ACCUMULATOR MODULE This module is designed to count pulses from external sources. The pulses may be accumulated on the basis of a selectable time interval, and an accumulated value can be held for interrogation by the CPU (Figure 3-44).

Programmable functions are:

Pulse Accumulator Input 0. (Mode I)

SETB	0,1	To select Pulse Input 0
STCR	(0,0),SAVE	To retrieve register count
SETB	0,0	To release Pulse Input 0

Pulse Accumulator Input 1. (Mode I)

SETB	1,1	To select Pulse Input 1
STCR	(0,0),SAVE	To retrieve register count
SETB	1,0	To release Pulse Input 1

Pulse Input 0, (Mode II)

SETB	2,1	To clear register count
SETB	0,1	Select

```
STCR    (0,0),SAVE      Retrieve

SETB    0,0             Release
```

Pulse Input 1 (Mode II)

```
SETB    3,1             Clear
SETB    1,1             Select

STCR    (0,0),SAVE      Retrieve

SETB    1,0             Release
```

Mode I and Mode II operation is a hardware option. For Mode I operation the pulse accumulator holding register is automatically cleared when the "release" instruction is executed.

A separate programmable "clear" command is used to reset the holding when Mode II is implemented.

3-5.10 MULTIPLY FUNCTION MODULES Two CRU Modules are used to implement the multiply function. There are designated "LEFT" and "RIGHT" respectively and are inserted in adjacent slots in a CRU expansion chassis. A sixteen bit (signed, 2's complement) number is output to "LEFT" and a second 16 bit number is output "RIGHT". After a short delay the product is available. The most significant 16 bits with sign extended is input from "LEFT" and the least significant 16 bits is input from "RIGHT" (Figure 3-45).

Example

```
LDCR    LEFT,       LOAD
        MPCND       MULTIPLICAND

LDCR    RIGHT,      LOAD
        MULT        MULTIPLIER

NOP                 *DELAY*

STCR    LEFT,       STORE
        MSPROD      PRODUCT MSP

STCR    RIGHT,      STORE
        L S PROD    PRODUCT LSP
```

NOTE

"LEFT" and "RIGHT" are defined as

```
MPY     BSEG    0
LEFT    CON     0,16
RIGHT   CON     16,16
  END
```

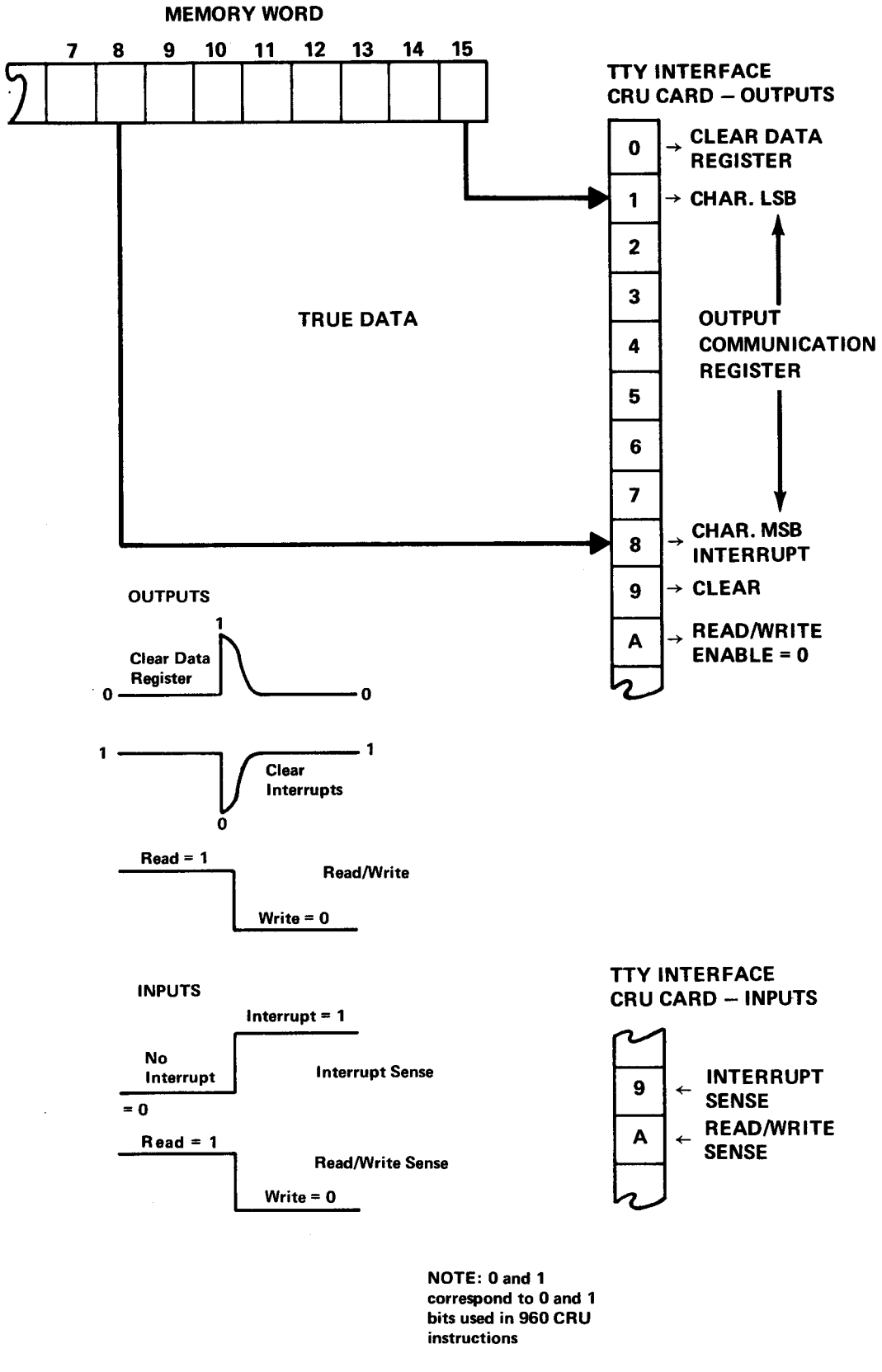


Figure 3-42 Teletypewriter Output

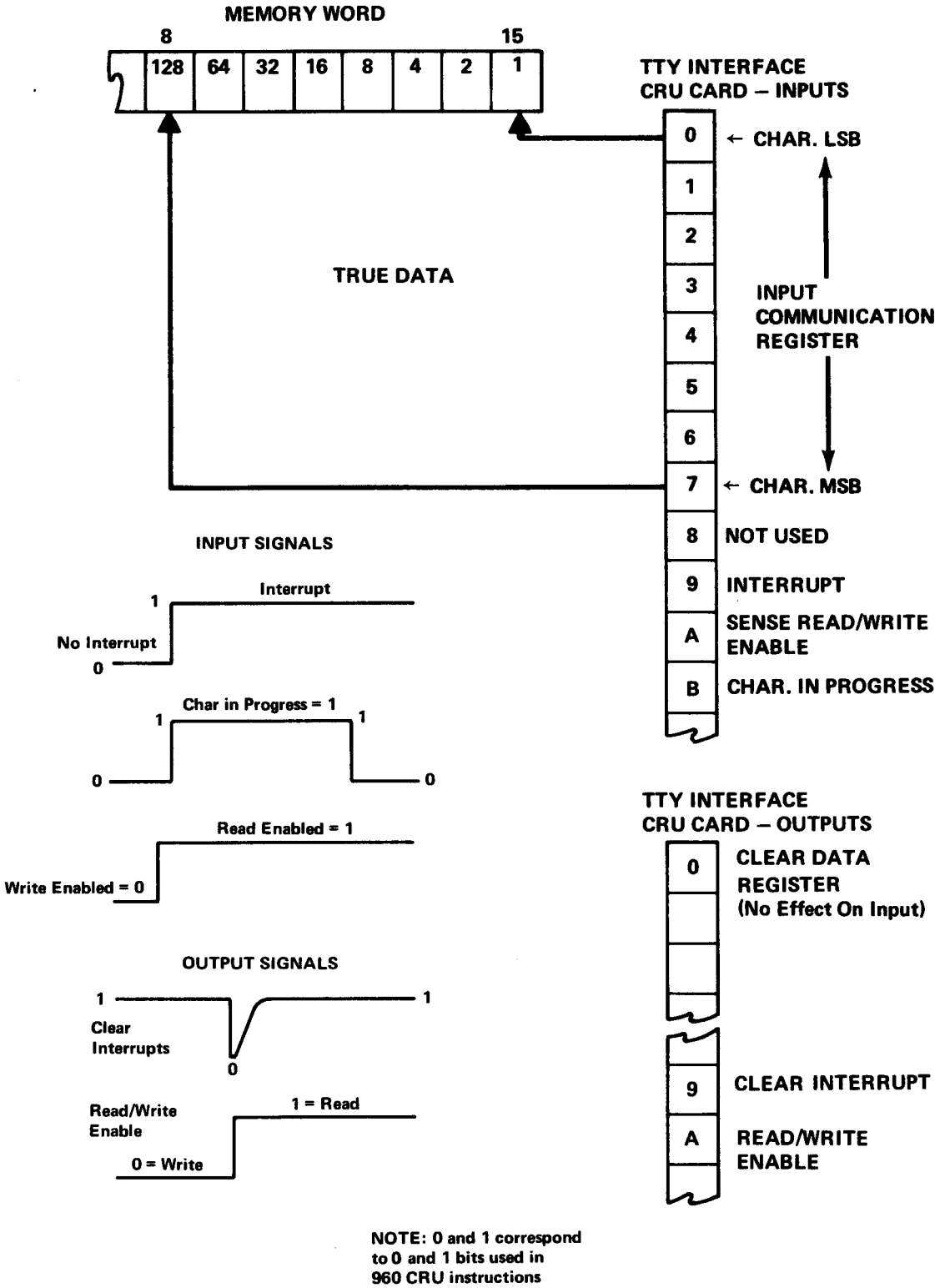


Figure 3-43 Teletypewriter Input.

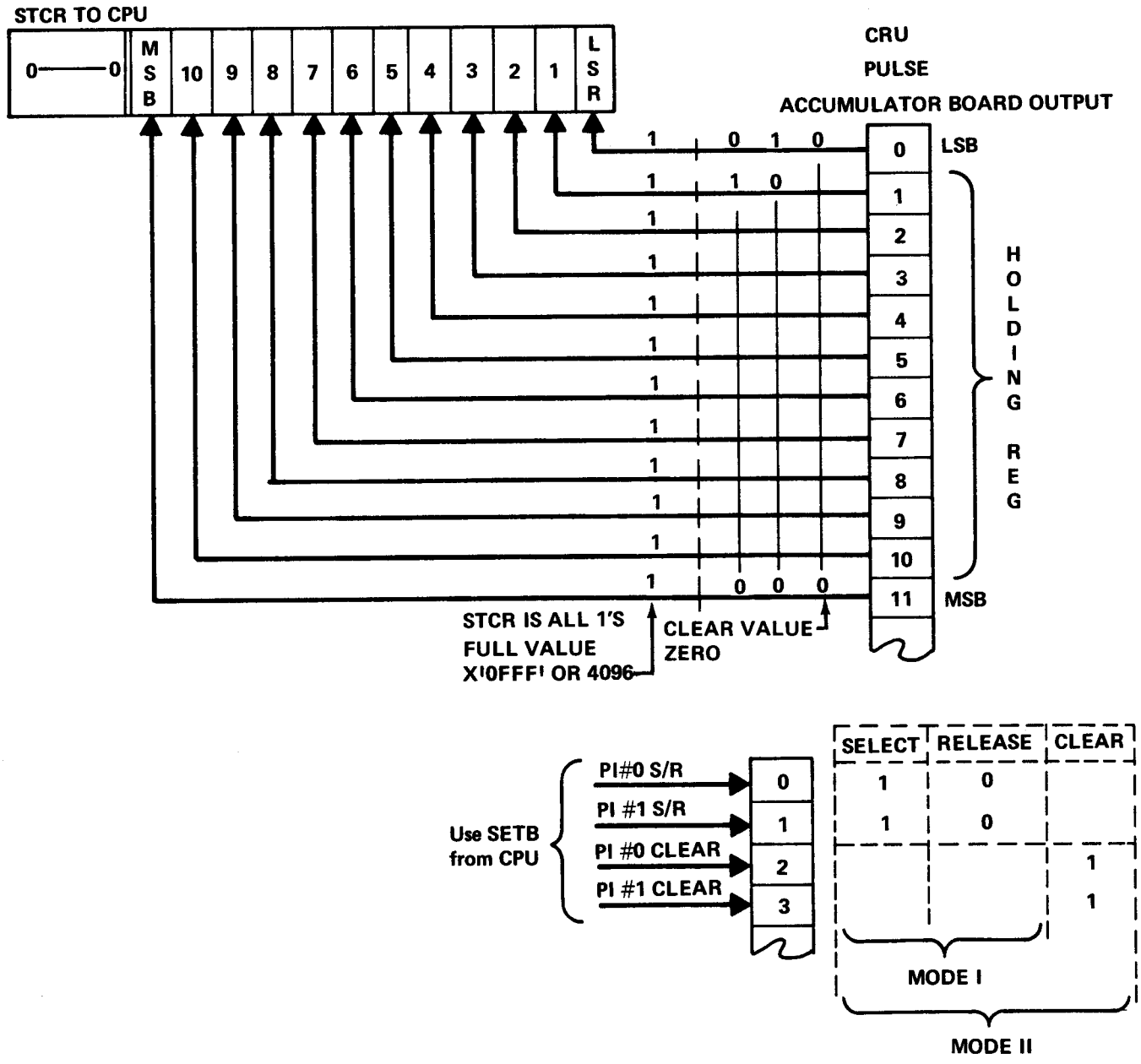


Figure 3-44 Pulse Accumulator Module.

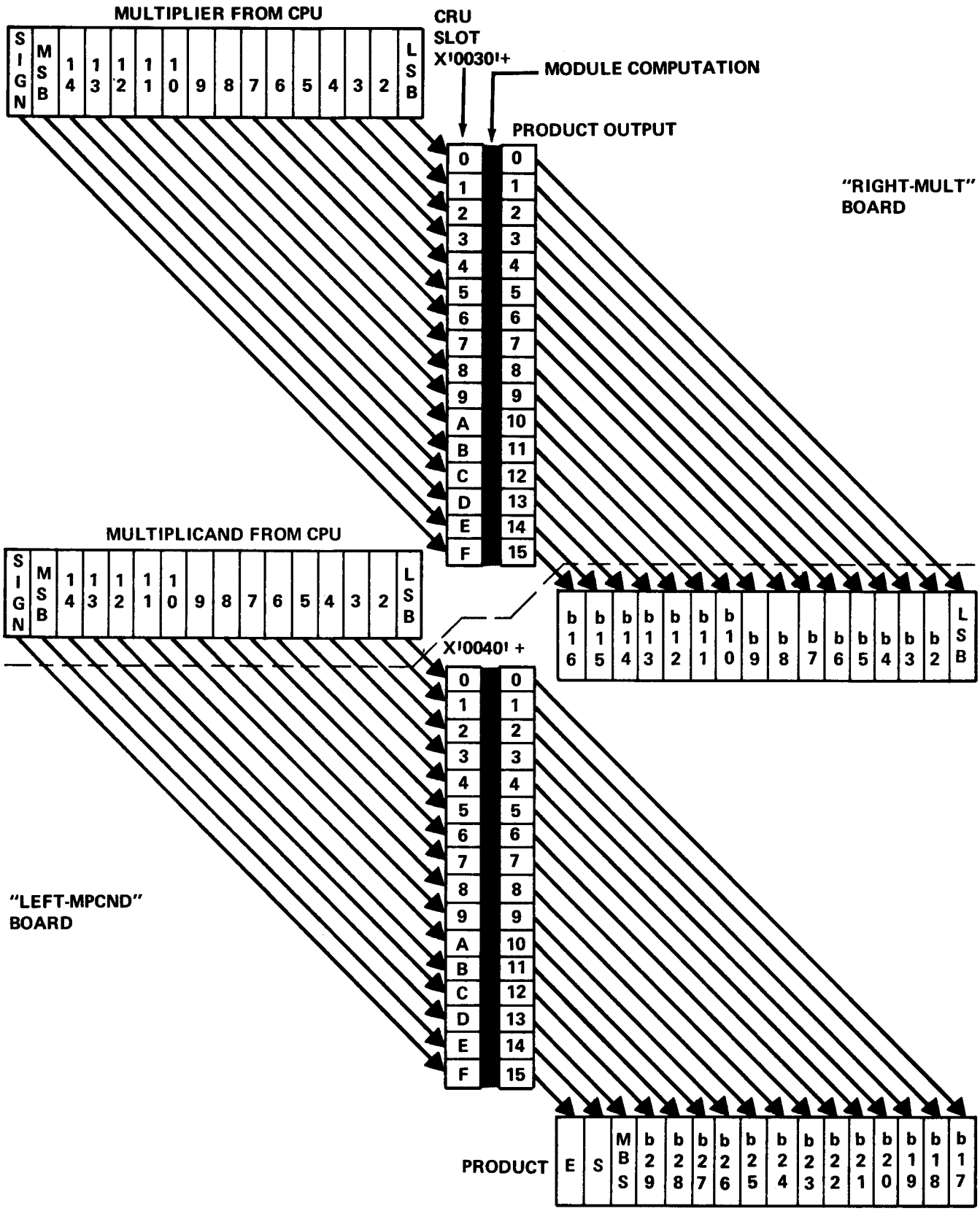


Figure 3-45 Multiply Functions Modules.

SECTION IV

PROGRAMMING SUPPORT MONITOR

4-1 INTRODUCTION.

The Programming Support Monitor (PSM) provides all the facilities required to operate the Model 960 programming system including

a. The assembler SAL960

b. The Linking Relocating Loader LRL960

c. The Source Maintenance Routine SMR960

d. The object program utilities

PSM is upwards compatible with the Process Automation Monitor (PAM). Programs that are written to execute using PSM as the computer interface will also execute using PAM.

Functions performed by PSM for the user include

a. Program Loading

b. Logical input/output device assignment

c. Control and operation of standard I/O equipment

d. Optional arithmetic and code conversion operations

e. Interrupt processing

f. Error detection and recovery.

Users with requirements for single program operation and particularly for non-real-time-program operation should consider the use of PSM as a system monitor because

a. PSM is upwards compatible with PAM

b. PSM is simple to use

c. PSM is expandable by the user

d. PSM memory requirements are minimal.

4-2 PSM SYSTEM DESCRIPTION.

The PSM Supervisor provides interrupt and supervisor call decoding together with interrupt subroutine linkage for operation of standard I/O devices and user linkage to system functions. The PSM user can solve applications problems without needing to become involved in the details of I/O interrupt processing and equipment control or the internals of system functions. (Multiply, Divide, Code Conversions, etc.)

Practically all PSM system functions are optional. The user may incorporate standard routines particularly suited to the solution of the application problem. The user where necessary can also supply and include special system functions of his own manufacture.

4-2.1 SUPERVISOR.

4-2.1.1 Interrupts. There are three interrupt routines:

a. Internal

b. CRU

c. DMAC.

When an internal interrupt occurs, PSM comes to an error halt (refer to paragraph 4-3.2). The DMAC interrupt is also a hardware or software malfunction and causes an error stop unless PSM is implemented on a machine with a DMAC. The internal interrupt is caused by a power failure or by a worker program which has not been debugged. Both the DMAC and internal interrupt device service routines may be easily replaced by the user with application-dependent interrupt processors.

The CRU interrupt routine decodes interrupts for I/O devices connected to the CRU and enters the I/O device service routine for each device requesting service. The device service routine processes the next character and returns to the CRU interrupt decoder.

NOTE

A device service routine that operates with interrupts masked is never entered in this manner.

PSM idles until each record requested is completed. At the end of the record control returns to the worker program.

4-2.1.2 Main Supervisor Call Decoder (SENT). A worker task makes a supervisor call by executing a SXBS *X'007F' instruction. X'007F' contains the address of the PSM Supervisor entry status block (SENTRY). At this time, Worker Register Three must contain an op code in bits 0-5 and an address relative to Worker Register Four in bits 6-15. The status is saved. Execution in the supervisor mode begins at location SENT. This routine decodes the call, puts the op code in Supervisor Register Zero (right justified) and the address in Supervisor Register One. If the op code is legal, the device service routine for that call is entered. Refer to paragraph 4-4 for a description of all legal calls. Many of the device service routines get arguments from the

worker registers and enter a system subroutine. When the requested action is completed, a LDS SENTRY instruction returns control to the worker program.

4-2.1.3 I/O Call Processor (IOQ). When the Supervisor call decoder (SENT) recognizes a zero op code, it branches to IOQ. Register One contains the address of a Physical Record Block (PRB). IOQ gets the Logical Unit Number (LUNO) from the PRB and searches the Logical Device Table (LDT) for a table entry with the same LUNO. If there is none, the PSM comes to an error halt (refer to paragraph 4-3.2). If the LUNO is found, the Physical Device Number is determined and the device service routine address is located in the Device Service Routine Address Table (DSRAT). The appropriate block in the Physical Device Table (PDT) is initialized with the device service routine address and PRB address. Register Five is loaded with the device temporary storage address. Register Seven is loaded with the device CRU base address. Next, the proper device service routine is entered, with interrupts masked. When the device service routine returns to IOQ (after processing the first character), interrupts are unmasked and control is transferred to the PSM idle location. When the entire I/O record is complete, control returns to the worker task.

4-2.2 DEVICE SERVICE ROUTINES. The input/output device service routines are reusable. One service routine can control as many devices of a particular type as there are physical device tables provided in the program.

4-2.2.1 Data Terminal/Teletypewriter Service Routine. The teletypewriter device service routine, running under interrupt control, inputs or outputs binary or ASCII characters via the keyboard or punched tape. Buffer and control information is taken from the user Physical Record Block. Refer to Figure 4-1.

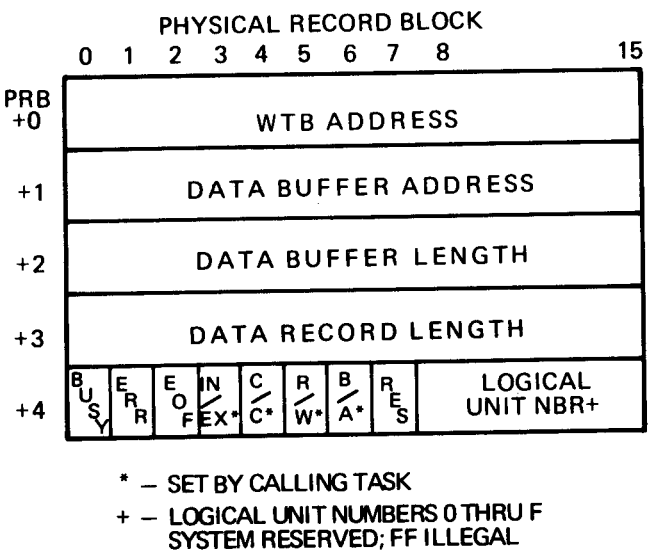


Figure 4-1 Physical Record Block.

The keyboard/printer is treated as one physical device. The paper tape reader/punch is considered a separate physical device. Two entry points are provided in the device service routines. Flags are set to permit use of common subroutines for processing teletype I/O for the

a. Keyboard

b. Printer

c. Paper Tape Reader

d. Paper Tape Punch

Half duplex operation is standard. Entry is from either IOQ or from the CRU interrupt decoder.

When the teletypewriter service routine is inactive, the keyboard is left open (bit 3 of temporary storage flag word is set). In this state, each character entered from the keyboard will be inspected and rejected except the letters L or X, which causes a task to be *loaded* or *executed.*

On first entry for a keyboard input, the bell rings to indicate ready for input.

On first entry for paper tape input, a reader-on character is sent.

On first entry for paper tape output, a punch-on character is sent. Teletypewriters with a manual punch will not respond to this signal.

The following changes will be made to the user Physical Record Block by the device service routine:

a. Input buffer length is truncated to $03FF_{16}$ or 1023_{10} characters.

b. At end of file, bit 2 of PRB flags is set to 1.

c. At end of record on input, the number of characters stored is placed in the PRB record length field.

No print line control is done by the device service routine. The user has complete control of page format. If a carriage return is sent as an ASCII output, a series of null characters is automatically sent by the device service routine to printing while the carriage is in motion. If 72 characters are sent with no carriage return, the device automatically returns the carriage, possibly causing an overprint.

Each physical device will have its own four-word temporary storage area arranged as follows:

Word 0 counter for characters input or output.

Word 1 address of buffer word being processed.

Word 2 user PRB flag word temporary storage.

Word 3 handler flag word. Initialized to 2001 for KB, 0001 for PT.

Device service routine flag word bit arrangement:

BIT	CONDITION
0	1, set when X-off sent at end of file on paper tape.
1	1, set when T-off sent at end of file on paper tape.
2	1, keyboard active, not a paper tape call.
3	1, keyboard open, set to zero on first entry.
4	1, All data stored on paper tape input. Wait for reader-off character.
5	1, Remex (high speed paper tape reader) active.
6	1, Carriage return sent. Send four null characters for wait loop.
7	Not used.
8-11	Null character counter used with bit 6.
12-15	Counter used in packing and unpacking buffer words.

Buffer words for binary or ASCII input are packed. An ASCII word contains two characters; a binary word contains four characters. If a buffer word is not full, the data is left justified.

To punch leader, set the control bit of PRB flags. Put the number of frames in the PRB record length field. Order of output on paper tape is PUNCH ON, LEADER, PUNCH OFF.

To punch binary data, set the PRB flags for binary *character output. Set the record length to number of characters to be output, buffer address to address of first word of data block. Order of paper tape output is: PUNCH ON, DATA, READER OFF, PUNCH OFF. All binary output frames contain a 6 in the upper digit. For example, a binary one will punch as a 61.

*A character is eight bits or two frames of tape.

To input ASCII, set the PRB flags for ASCII character input, set the buffer length to the number of input characters, and set the buffer address to the address of the first word of the storage area. Only characters with values between X'0020' and X'005F' are stored. Carriage return or a full buffer halts the input and terminates a keyboard call. If the input is from paper tape, the tape continues to pass after the carriage return is input until the reader-off character is sensed, but no data after the carriage return is stored.

For binary input, set the PRB flags for Binary Character input. Set the buffer length to the number of characters (two hex digits/character) to be stored. Set the buffer address field to the address of the first word of the storage buffer. Only data between X'0060' and X'006F' is accepted. A reader-off character or a full buffer stops the data storage. Only a reader-off character stops tape movement. Data between the buffer-full indication and reader-off character is lost.

4-2.2.2 Punch-Tape Reader Service Routine. HSRSEG is the entry point for the punch-tape reader service routine. Since the data input by the high-speed tape reader is subject to the same restrictions as that of the data terminal/teletypewriter punch tape reader, routines are shared where possible. The teletypewriter service routine must be a part of a system which includes a punch-tape reader.

The high speed reader service routine controls operation of the reader, processes reader interrupts, and uses the teletypewriter character processing routines for character processing.

4-2.2.3 Card Reader Service Routine. The card reader service routine is activated by an I/O call.

The device service routine performs either an ASCII or a binary read function of any buffer length from one to 80 characters inclusive as specified in the Physical Record Block associated with the call. A buffer length specification greater than 80 characters is truncated. A zero or negative buffer length gives unpredictable results.

The number of characters read from a card is stored in the Physical Record Block (PRB) along with error and End-of-File flags.

When an ASCII read function is being performed, any Hollerith code not found in Table 4-1 results in an error flag in the PRB and an ASCII blank inserted in the buffer in place of the illegal character. This does not affect the reading of the remainder of the card.

Because of program structure, the card reader need not be ready prior to an I/O call. The device service routines wait for the operator to ready the card reader before it issues a feed-card command.

The card reader requires approximately 185 milliseconds to read a card. During this interval, PSM masks external interrupts.

A card jam or misregistration causes the CPU to halt with

TABLE 4-1
LEGAL HOLLERITH CODES

ASCII	H. CODE	CHAR	ASCII	H. CODE	CHAR	ASCII	H. CODE	CHAR	ASCII	H. CODE	CHAR
20	No Punches	SP	30	0	0	40	8 · 4	@	50	11 · 7	P
21	11 · 8 · 2	!	31	1	1	41	12 · 1	A	51	11 · 8	Q
22	UD		32	2	2	42	12 · 2	B	52	11 · 9	R
23	8 · 3	#	33	3	3	43	12 · 3	C	53	0 · 2	S
24	11 · 8 · 3	$	34	4	4	44	12 · 4	D	54	0 · 3	T
25	UD		35	5	5	45	12 · 5	E	55	0 · 4	U
26	12	&	36	6	6	46	12 · 6	F	56	0 · 5	V
27	8 · 5	'	37	7	7	47	12 · 7	G	57	0 · 6	W
28	12 · 8 · 5	(	38	8	8	48	12 · 8	H	58	0 · 7	X
29	11 · 8 · 5	)	39	9	9	49	12 · 9	I	59	0 · 8	Y
2A	11 · 8 · 4	*	3A	8 · 2	:	4A	11 · 1	J	5A	0 · 9	Z
2B	12 · 8 · 6	+	3B	11 · 8 · 6	;	4B	11 · 2	K	5B	UD	
2C	0 · 8 · 3	,	3C	UD		4C	11 · 3	L	5C	UD	
2D	11	–	3D	8 · 6	=	4D	11 · 4	M	5D	UD	
2E	12 · 8 · 3	.	3E	UD		4E	11 · 5	N	5E	UD	
2F	0 · 1	/	3F	0 · 8 · 7	?	4F	11 · 6	O	5F	UD	

CONTROL CHARACTER

ASCII	H. CODE	CHAR	MEANING
17	0 · 9 · 6	EOB	BLOCK END

UD = UNDEFINED

X'3001' displayed in INSTRUCTION REGISTER A.

To recover this error:

a. Remove last card from card reader output stacker.

b. Place card at front of unread card deck in read hopper.

c. Restart card reader.

This halt does not set a PRB error flag since this procedure gives effective recovery.

4-2.2.4 **Card Punch Service Routine – CRU Interface.** The card punch service routine is first activated by the I/O call processor (IOQ) and subsequently operates under interrupt control. Standard end of record processing returns program control to the worker task after the end of record is detected.

The number of characters transmitted as one record to the card punch is specified in the record length word of the PRB associated with the I/O call. End of record processing starts after the last character has been transmitted to the punch. Records should be at least one character long and no more than 80; otherwise results are unpredictable.

Data may be in either binary or ASCII format (see note) packed two characters per word. The PRB contains the appropriate data address.

The punch does not operate when the feed hopper is empty, when the stacker is full, or when the chip box is full. These conditions can be corrected as follows:

a. Feed Hopper Empty

(1) HALT the computer using the operator's console.

(2) Place the punch in HOLD.

(3) Carefully riffle and place the cards in the feed hopper.

(4) Operate punch MASTER CLEAR.

(5) Select computer RUN, START.

(6) Release punch HOLD.

b. Stacker Full

(1) HALT the computer using the operator's console.

(2) Place the punch in HOLD.

(3) Remove the cards from the stacker.

(4) Operate punch MASTER CLEAR.

(5) Select computer RUN, START.

(6) Release punch HOLD.

c. Chip Box Full

(1) HALT computer using operator's console.

(2) Place punch in HOLD.

(3) Empty chip box.

(4) Operate punch MASTER CLEAR.

(5) Select computer RUN, START.

(6) Release punch HOLD.

The punch should be placed in the HOLD condition between files when no punch output is expected. This reduces punch mechanism wear and tear. It also prevents punch malfunctions and the need for frequent maintenance.

Mispunching of the first card can be avoided by always requesting a record to be punched before each file that can later be discarded. Punch malfunction during the first record is commonly caused by mechanical and electrical transients when the punch is started from the power off condition or from the HOLD condition.

Punching errors cannot be detected by the computer. Separate data verification is recommended such as use of a redundancy character or checksum.

PROGRAMMING NOTE

ASCII character conversion is optional in the punch service routine. Use ASCII only if the capability is available.

Punch malfunctions should be avoided by careful operating practices. Malfunctions can be recovered using the following procedure to eliminate the effect of unpredictable mechanical and electrical transients that occur when the punch fails to operate.

a. Non Pick

(1) HALT the computer using the operator's console.

(2) Place the punch in HOLD.

(3) Remove the damaged card from the feed hopper.

(4) Operate punch MASTER CLEAR.

(5) Display PSM load address +102.

(6) Enter value displayed in the program counter.

(7) Enter X'00C0' in the status register.

(8) Select RUN, START.

(9) Release Punch HOLD.

b. Jam

(1) HALT the computer using the operator's console.

(2) Place the punch in HOLD.

(3) Remove the damaged card from the punch mechanism.

(4) Perform steps four through nine above.

4-2.2.5 Paper Tape Punch Service Routine – CRU Interface. The paper tape punch service routine is first activated by the I/O call processor (IOQ) and subsequently operates under interrupt control.

Standard end of record processing returns program control to the worker task after end of record is detected.

Three punch functions are implemented and are requested by setting the appropriate bit in the PRB associated with the I/O call.

a. Punch ASCII – one frame per character.

b. Punch Binary – two frames per character.

c. Punch Leader – one null frame per character.

Data to be punched should be packed two characters per word. The number of characters is specified in the record length word of the PRB associated with the I/O call.

The punch motor power is off at all times when the punch is not in operation. When a record is punched the power is turned on. The punch power up sequence causes a delete code to be punched in the tape. No data is destroyed. Worker tasks should request leader before and after each file punched.

4-2.3 SYSTEM SERVICE ROUTINES.

4-2.3.1 Multiply Service Routine. This service routine obtains the integer product of the contents of worker register 0 and a 16 bit two's complement number in memory. The 32 bit two's product replaces the contents of Worker Registers Zero and One (16 magnitude bits in Register One and 15 magnitude bits and a sign bit in Register Zero).

TITLE: MULTPY

PURPOSE: To determine the product of two bit two's complement integer numbers.

TIMING: 930 microseconds (average)

STORAGE REQUIREMENTS: 119 words

PREREQUISITE: None

EXAMPLE: 0010 X FFFF = FFFF FFF0
FFFF X FFFF = 0000 0001
7FFF X 7FFF = 3FFF 0001

ERROR RETURN: None

4-2.3.2 Divide Service Routine. This service routine divides the 32-bit two's complement number in Worker Registers Zero and One by the 16-bit two's number. Its quotient is returned in Register One and the remainder in Register Zero. If the sign of the quotient is plus then the sign of the remainder is plus. If the quotient is minus, then the sign of the remainder is equal to the sign of the original dividend.

TITLE: DIVIDE

PURPOSE: To divide a 32 bit two's complement number by a 16 bit two's complement number.

TIMING: 1.2 milliseconds (average)

STORAGE REQUIREMENTS: 138 words

PREREQUISITE: None

EXAMPLE: $\frac{0001\ 000F}{0004}$ = 4003 with remainder = 0003

$\frac{FFFE\ FFF1}{0004}$ = BFFD with remainder = FFFD

$\frac{0001\ 000F}{FFFC}$ = BFFD with remainder = 0003

ERROR RETURN: If overflow occurs, that is, if the magnitude of the quotient is larger than 15 bits, then the overflow bit of the worker status register is set to 1.

4-2.3.3 Double Word Circular Left Shift Service Routine. This service routine shifts the two words addressed by Worker Register One circular left the number of places given by Worker Register Two, the old double word being replaced by the new shifted double word.

TITLE: SCLD

PURPOSE: To perform a double word circular left shift from 0 to 31 places.

TIMING: Approximately 141 microseconds + 1.6 Nμ (N = Shift Count).

STORAGE REQUIREMENTS: 81 words

PREREQUISITE: None

EXAMPLE: Shift 3F01, 1001 circular left 4 will result in F011, 0013.

ERROR RETURN: A shift count greater than 31 gives an unpredictable result, but no notification is given to the user.

4.2.3.4 Square Root Service Routine. This service routine calculates the square root of a 32-bit number in Worker Registers Zero and One (16 magnitude bits in Register One and 15 magnitude bits and a sign bit in Register Zero) and places the results in Worker Register Zero. If the number is minus (that is bit 0 of register 0 equals 1) then no operation is performed.

TITLE: SQRT

PURPOSE: To calculate the square root of a 32 bit fixed point number.

TIMING: 4.8 milliseconds

STORAGE REQUIREMENTS: 179 words

PREREQUISITE: Subroutine DIVIDE.

EXAMPLE:

Enter	Exit
3FFFF 0001	7FFF

ERROR RETURN: The maximum value which can be handled by this routine is 3FFF 0001_{16}. Any larger number results in an overflow in which case the worker status overflow bit is set to 1.

4-2.3.5 Convert Binary to Hexadecimal ASCII Service Routine. This service routine converts a 16-bit binary number contained in Worker Register Zero to a four-digit hexadecimal number and stores this converted value in a two-word buffer supplied by the worker program.

TITLE: CBHA

PURPOSE: To convert a 16-bit binary number to a four-digit hexadecimal ASCII number.

TIMING: 169 microseconds (average)

STORAGE REQUIREMENTS: 47 words

PREREQUISITE: The 16 word table BTOATB in service routine CBDA.

EXAMPLE:	Binary Value	Resulting Hex. ASCII
	7FFF	3746 4646

ERROR RETURN: None

4-2.3.6 Convert Hex ASCII To Binary Service Routine. This service routine converts a two-word hexadecimal ASCII array supplied by the worker task into a 16-bit binary value and returns it in Worker Register Zero. No validity checks are made. Significant hexadecimal digits should be right justified in the field with leading zeros or blanks padded.

TITLE: CHAB

PURPOSE: To convert a two-word hexadecimal ASCII array to binary.

TIMING: 346-2/3 microseconds + 1/3 microsecond for each character > 9.

STORAGE REQUIREMENTS: 48 words

PREREQUISITE: None

EXAMPLE: 41303746 hexadecimal ASCII converts to A07F.

ERROR RETURN: None

4-2.3.7 Convert Binary To Decimal ASCII Service Routine. This service routine converts a two's complement binary number in Worker Register Zero to a signed five-digit decimal ASCII number and stores this converted value in a three-word buffer supplied by the worker program.

TITLE: CBDA

PURPOSE: To convert a 16-bit two's complement binary number to a signed five-digit decimal ASCII number.

TIMING: 6.271 milliseconds

STORAGE REQUIREMENTS: 93 words

PREREQUISITE: Subroutine DIVIDE.

EXAMPLE:	Binary Value	Resulting Decimal ASCII
	7FFF	2B33 3237 3637

ERROR RETURN: None

4-2.3.8 Convert Decimal ASCII To Binary Service Routine. This service routine converts the three-word signed decimal ASCII buffer provided by the worker task into a 16-bit binary number returned in Worker Register Zero. Significant digits should be right justified in the field with leading zeros or blanks padded. A value which is not signed negative is assumed positive. No overflow or validity checks are made.

TITLE: CDAB

PURPOSE: To convert a 3-word signed decimal ASCII value to 16-bit binary.

TIMING: 55 microseconds

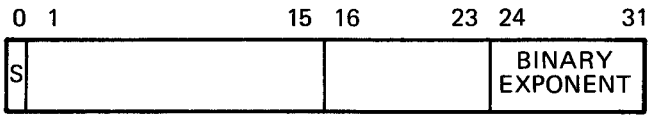
STORAGE REQUIREMENTS: 72 words

PREREQUISITE: None

EXAMPLE: +00329 read in ASCII format = 2B30 3033 3239. CDAB converts this buffer to 0149.
-00001 yields FFFF.

ERROR RETURN: None

4-2.3.9 Convert Fixed Point Number To Floating Point Service Routine. This service routine converts the double word fixed point number in Worker Registers Zero and One to a floating point number and places the resulting 32 bits in Worker Registers Zero and One. The floating point format is:



S = { 0 =plus / 1 = minus 23 bit magnitude Biased by X'0080'

TITLE: FIXFLT

PURPOSE: To convert a 32-bit two's complement fixed point number to floating point format.

TIMING: 555 microseconds

STORAGE REQUIREMENTS: (Floating Point Package) 851 words

PREREQUISITE: SCLD

EXAMPLE:

Fix Point (Hex)		Floating Point	Decimal Equivalent
0000	0001	40000081	1.0
FFFF	FFFF	C0000081	-1.0
0000	FFFF	7FFF8090	+65535.0
FFFF	0001	80008090	-65535.0

ERROR ERTURN: None

4-2.3.10 Convert Floating Point Number To Fixed Point Service Routine. This service routine converts a floating point number in Worker Registers Zero and One to fixed point format and returns the number in Worker Registers Zero and One.

TITLE: FLTFIX

PURPOSE: To convert a floating point number to a fixed point number.

TIMING: 460 microseconds

STORAGE REQUIREMENTS: (Floating Point Package) 857 words

PREREQUISITES: None

EXAMPLE:

Decimal	Floating Point (Hex)		Fixed Point Results	
1.0	4000	0081	0000	0001
0.5	4000	0080	0000	0000
-1.0	C000	0081	FFFF	FFFF
3.5	7000	0082	0000	0003

ERROR RETURN: None

4-2.3.11 Add Floating Point Numbers Service Routine. This service routine determines the floating point sum of a floating point number in Worker Registers Zero and One and a floating point number in memory. This sum replaces the contents of Worker Registers Zero and One. Refer to the floating point format.

TITLE: FLTADD

PURPOSE: To determine the sum of two floating point numbers.

TIMING: 900 microseconds

STORAGE REQUIREMENTS: (Floating Point Package)

PREREQUISITE: DUBADD, NORMAL, FINEXP, CLEXP

ERROR RETURN: None

4-2.3.12 Subtract Floating Point Numbers. This service routine determines the floating point difference of a floating point subtrahend in Worker Registers Zero and One and a floating point minuend in memory. This difference replaces the contents of Worker Registers Nine and One. Refer to the floating point format.

TITLE: FLTSUB

PURPOSE: To determine the difference of two floating point numbers.

TIMING: 1.0 millisecond

STORAGE REQUIREMENTS: (Floating Point Package) 851 words

PREREQUISITE: FLTADD, DUBCMP

ERROR RETURN: None

4-2.3.13 Multiply Floating Point Numbers Service Routine. This service routine determines the floating point product of a floating point number in Worker Registers Zero and One and a floating point number in memory. This product replaces the contents of Worker Registers Zero and One. Refer to the floating point format.

TITLE: FLTMUL

PURPOSE: To determine the product of two floating point numbers.

TIMING: 4.3 milliseconds

STORAGE REQUIREMENTS: (Floating Point Package) 851 words

PREREQUISITE: MULTPY, DUBADD, DUBRT, DUBCMP, NORMAL, FINEXP, CLEXP

ERROR RETURN: None

4-2.3.14 Divide Floating Point Numbers Service Routine. This service routine determines the floating point quotient of a floating point dividend in Worker Registers Zero and One and a floating point divisor in memory. This quotient replaces the contents of Worker Registers Zero and One. Refer to the floating point format.

TITLE: FLTDIV

PURPOSE: To determine the quotient of two floating point numbers.

TIMING: 5.6 milliseconds

STORAGE REQUIREMENTS: (Floating Point Package) 851 words

PREREQUISITE: Divide, Multiply

ERROR RETURN: None

4-2.3.15 Convert Floating Point Numbers Service Routine. This service routine converts a floating point number in Worker Registers Zero and One to output format and stores the resulting six words in a buffer supplied by the worker program. The output format is of the form 5_1.NNNNNNE5_2MM, where 5_1 is the sign of the number, .NNNNNN is the magnitude of the number in decimal, E divides the magnitude from the exponent, S_2 is the sign of the exponent, and MM is the decimal power of ten by which the number is multiplied.

TITLE: CONVBE

PURPOSE: To convert a floating point number to a format suitable for output.

TIMING: 240 milliseconds (maximum)

STORAGE REQUIREMENTS: (Floating Point Package) 851 words

PREREQUISITE: FLTDIV, FLTMUL, CBDA

EXAMPLE:

Floating Point Number	Output Format
1.0	+.100000E01
-479.01	-.479010E03

ERROR RETURN: None

4-2.3.16 Floating Point Sine Service Routine. This service routine calculates the SIN of the floating point number in Worker Registers Zero and One, which represents some angle between $-\pi/2$ and $\pi/2$, and returns the results in Registers Zero and One.

TITLE: FSIN

PURPOSE: To determine the floating point SIN of a floating point number.

TIMING: 39.12 milliseconds

STORAGE REQUIREMENTS: (Trigonometric Package) 468 words

PREREQUISITE: FLTSUB, FLTADD, FLTMUL, FLTDIV

ERROR RETURN: An angle greater than $\pi/2$ or less than $-\pi/2$ will produce an inaccurate result but no notification is given to the worker program.

4-2.3.17 Floating Point Cosine Service Routine. This service routine calculates the COS of the floating point number in Worker Registers Zero and One, which represents some angle between $-\pi/2$ and $\pi/2$, and returns the results in Registers Zero and One.

TITLE: FCOS

PURPOSE: To determine the floating point COS of a floating point number.

TIMING: 39.12 milliseconds

STORAGE REQUIREMENTS: (Trigonometric Package) 468 words

PREREQUISITE: FSIN

ERROR RETURN: An angle greater than $\pi/2$ or less than $-\pi/2$ will produce an inaccurate result but no notification is given to the worker program.

4-2.3.18 Floating Point Arctangent Service Routine. This service routine calculates an angle, between $-\pi/2$ and $\pi/2$, the SIN of which is contained in Worker Registers Zero and One, and the COS of which is addressed by the worker programs supervisor call. This resulting angle is placed in Worker Registers Zero and One.

TITLE: ARCTAN

PURPOSE: To compute the floating point arctangent of the ratio of two floating point arguments.

TIMING: 68.42 milliseconds

STORAGE REQUIREMENTS: (Trigonometric Package, including FSIN, FCOS, and ARCTAN) 468 words

PREREQUISITE: FSIN

ERROR RETURN: None

4-2.4 SYSTEM BOOTSTRAP LOADER. The system loader, which occupies locations 0-124, is used by PSM to process binary records from the SAL960 assembler and produce a relocated program in core memory. The loader has been written for three input devices. The input device determines which loader is used. Input devices are:

a. Card reader

b. Tape reader

c. Teletypewriter.

All loader variations have similar operating procedures, differing only in the manual control of the input device. Loader operating instructions are located in paragraph 4-5.

Loader restrictions and limitations are:

a. Error checking has been minimized in favor of saving memory.

b. Absolute programs (with a fixed load address) will not be loaded correctly unless the load address is manually set to zero before loading the program.

For these reasons, PSM will allow the user to supply his own loader program. To do this:

a. Using the system loader, load the new loader program at some location in memory.

These changes will not necessitate a new system generation.

4-2.5 LOGICAL DEVICE ASSIGNMENT. Worker programs address physical I/O devices by logical unit numbers. WDFIO is a worker program that assigns a logical unit number (LUNO) to a physical device (DEVNO). Loading instructions for WDFIO are found in paragraph 4-5.

Ten logical device assignments are permitted in the standard PSM. Logical Unit Zero is permanently assigned to the system input device. This leaves nine available for worker program logical unit assignment. All assignments, except Logical Unit Zero, are released each time WDFIO is loaded. Assignments for all I/O devices are required each time WDFIO is used. These assignments remain until WDFIO is executed again.

On entry, the program prints OP? on the system input device. There are three valid responses:

a. DFIO – is a request for I/O file definition.

b. ↑ – followed by a carriage return terminates the task.

c. ← – followed by a carriage return negates the last successful assignment.

Any other entry will print BADOP. This and all other error messages (Table 4-2) are followed by OP? and require one of the three legal responses.

If DFIO is entered, the system input device skips two spaces and waits for a four-digit hexadecimal LUNO to be entered in the format OOLL. If four zeroes are entered, UNIT ZERO NOT AVAILABLE is printed and the entry ignored.

If the LUNO entered has been previously assigned *during this program call,* the message ERROR.START OVER is printed and any assignments completed must be redone. If an entry is made after the nine available assignments have been completed, TABLE FULL is printed and the task is terminated. The number of assignment slots available to the worker program can be increased only at system generation time. Refer to paragraph 4-7.

If the LUNO is accepted, the system input device skips two spaces and waits for a hexadecimal DEVNO to be entered in the format OODD. If the DEVNO entered is greater than the highest DEVNO in PSM, DEVNO TOO LARGE is printed and the entry is ignored. All available logical unit numbers can be assigned to the same physical device number, if desired.

EXAMPLE: Assign logical unit 02 to Physical Device 04. The underlined portion is operator entry.

ENTRY REQUEST	OPERATOR ENTRY		
OP?	DFIO	0010	0002
OP?	DRIO	0020	0003
OP?	DFIO	0030	0002
OP?	DFIO	0040	0000
OP?	DFIO	0050	0004
OP?	DFIO	0060	0000
OP?	↑		

4-2.6 PHYSICAL DEVICE NUMBERS. All I/O devices controlled by PSM are given a physical device number. Each device has a unique number.

The numbers range from 0 to N-1, where N is the total number of I/O devices.

These numbers are determined by the order of the device service routine address table in the Supervisor Data Segment and are fixed at system generation time.

All PSM systems must have an I/O device called the System Logging device, such as the teletypewriter. This device always has a physical device number of zero.

TABLE 4-2
LDT TROUBLESHOOTING

ERROR MESSAGE	CAUSE	RESPONSE
BAD OP	Illegal response to OP?	Enter DFIO to continue, ↑ CR to terminate, ←CR to repeat last entry.
UNIT ZERO NOT AVAILABLE	Logical unit zero entered	Repeat entry with valid LUNO.
ERROR. START OVER	Duplicate LUNO entries	Start over. All prior entries cleared.
DEVNO TOO LARGE	No physical device with that DEVNO	Repeat entry with correct DEVNO.
TABLE FULL	All assignment slots full	The task is terminated.
ILLEGAL NUMBER	Character other than 0-F	Repeat the entry.

4-2.7 BASIC PSM SYSTEM. The following routines are a part of any standard PSM. The primary I/O device is assumed to be a data terminal/teletypewriter.

Supervisor (SPB)

Supervisor (SDB)

CRU Interrupt Decoder (CINTSG)

Multiply (MULTPY)

Divide (DIVIDE)

Circular Double Left Shift (SCLD)

Binary/Hexadecimal (CBHP)

Binary/Decimal (CBDA)

Hexadecimal/Binary (CHAB)

Data Terminal/Teletypewriter (DUOSEG & TTYSEG)

Loader Teletypewriter

Data Terminal/Teletypewriter (DUOSEG & TTYSEG)

If the equipment also includes a card reader, the following additional PSM routines are included:

DP/UT SR300 Driver (CARD READER)

Loader Card Reader

4-2.8 CORE DESCRIPTION. Figure 4-2 shows a typical core memory map. The user is advised that only one loader resides in protected core. Either the data terminal/ teletypewriter or card reader loader must be installed prior to attempting to load PSM or a worker program.

4-3 PSM CONTROL COMMUNICATION.

Simple control communication between the console operator and PSM is provided.

4-3.1 INPUT. There are only two valid inputs to the PSM system: L and an X. When no work program is in execution, the PSM is in an idle loop waiting for instructions from the operator. The letter L entered via the system input device will load the object program. An X will execute the program currently in core. This permits repetitive execution of a program.

4-3.2 OUTPUT. PSM halts when abnormal conditions are encountered. At this point, the contents of console Instruction Register A should equal 72D2, and the contents of console Instruction Register B should equal 0000. The contents of Supervisor Register Zero (memory location X'0080') can be examined to determine the exact error condition. Table 4-3 gives error codes: an explanation for each.

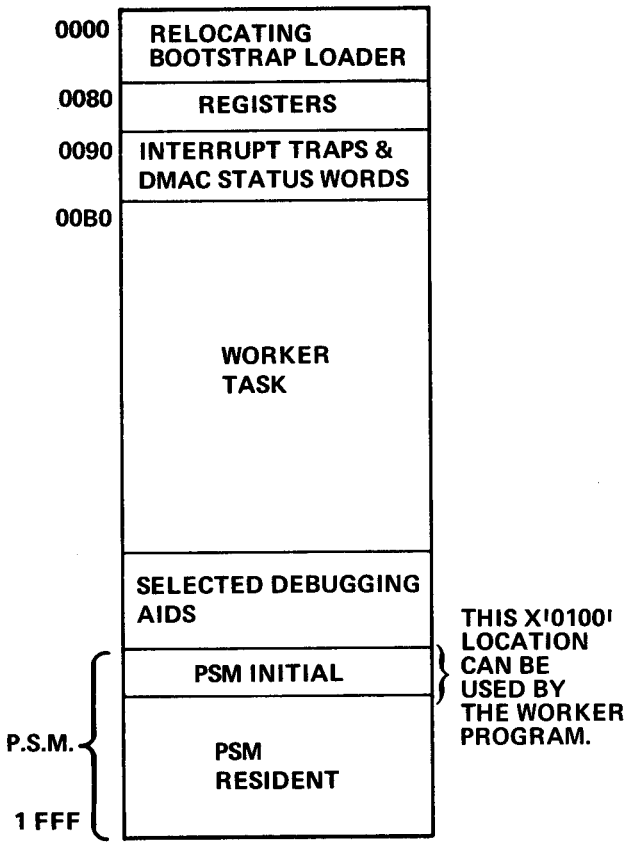


Figure 4-2 Typical 8K PSM Core Load

TABLE 4-3
ERROR CODES

CONTENTS OF X'0080'	ERROR EXPLANATION
0000	Illegal Interrupt – Either an internal or DMAC interrupt occurred. Memory locations K'0106' and K'0107' of PSM can be examined to determine the contents of the program counter (or event counter) and the STATUS REGISTER when the error occurred.
0001	Bad Supervisor Call – PSM was asked by the user to perform some illegal task.
0002	Illegal Logical Unit.
0003	Illegal Physical Device Number.

To recover from this error halt:

a. Set ST = X'0000'

b. Set PC = X'007D'

c. Select RUN, START

4-4 USING PSM.

4-4.1 WORKER TASK BLOCK. Tasks operating under PSM are executed in the worker mode. PSM must have access to the initial values of the EC, Status Register, and the worker mode registers. Both PSM and PAM (Process Automation Monitor) require the programmer to put the above information in the first 16 words of his task. This data is called the Worker Task Block (WTB). PSM does not utilize all 16 words, but they are reserved so that the task may run under either PSM or PAM.

WTB (0) – Procedure Entry Point

WTB (1) – Initial Status

WTB (2) – *

WTB (3-10) – Initial Value of worker registers 0-7

WTB (11) – Unused by PSM, but initial status put here for PAM

WTB (12) – Unused by PSM, but Entry Point put here for PAM.

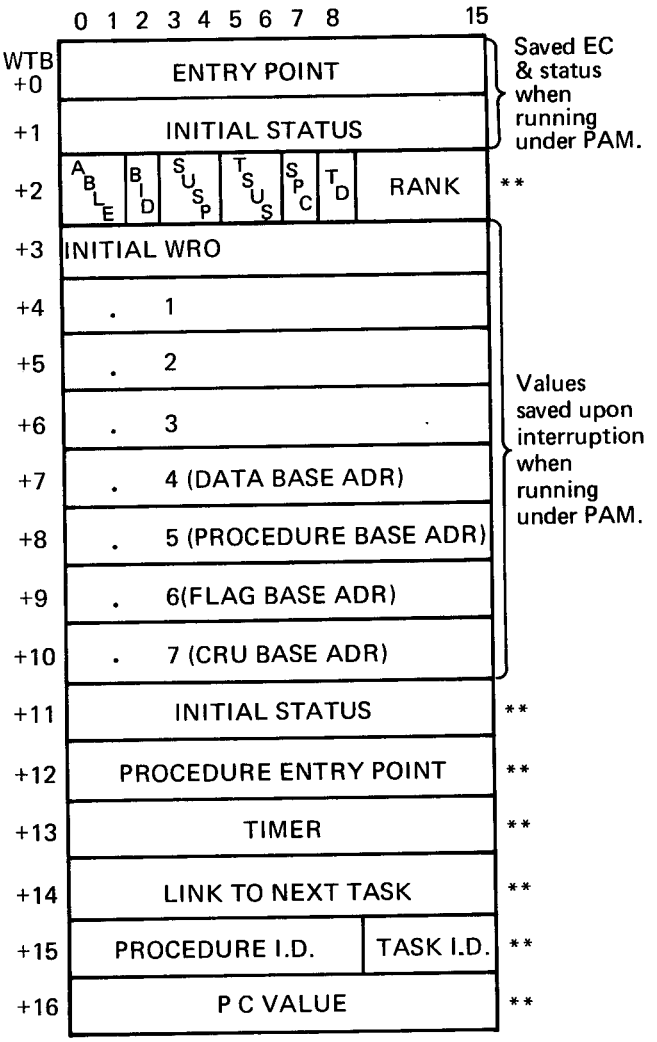
WTB (13-15) – *

*Unused by PSM but required for PAM.

Table 4-4 shows the make-up of the WTB for both PSM and PAM. Since this block is set up properly, the task may be executed under either PSM or PAM.

4-4.2 SUPERVISOR CALLS. PSM provides I/O device service routines, task sequencing, and subroutines to implement programming conveniences. The task invokes PSM operational functions using supervisor calls.

TABLE 4-4
WORKER TASK BLOCK +



\+ FIRST SIXTEEN CONSECUTIVE LOCATIONS OF TASK DATA SEGMENT

* EXISTENCE DEPENDENT ON USER DICTATES

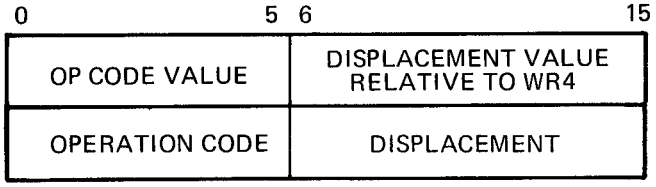
** NOT USED BY PAM

The supervisor call format is independent of the function being accessed. The supervisor call format consists of one equate statement and two instruction statements.

```
                .
                .
                .
                .
SENTRY  EQU     127
                .
                .
                .       Task Coding
                .
                LA      3,Function Call Format
                SXBS    *SENTRY
                .
                .
                .
                .
```

Worker Register Three contains the supervisor op code and, optionally, an address. Worker Register Three, bits 0-5, contain the op code and bits 6-15 when used, contain an address. The op code is a number representing a specific function. The address is relative to the contents of Worker Register Four. The displacement value and the contents of Worker Register Four are added to obtain the effective address of a specific memory location.

FUNCTION CALL FORMAT



Functions marked (R) are always resident in PSM. Those marked (O) are optional. Those marked (B) are optional, but they must be resident in any basic PSM system.

4-4.2.1 Input/Output. The input/output function effective address must point to the first memory location of the Physical Record Block (PRB) describing the I/O to be performed (R).

CODE	FUNCTION
00	Input/Output

4-4.2.2 End of Program. The end of program function does not use the displacement sector of Worker Register Three. The operational task is terminated. PSM returns to a state of waiting for a letter L or E to be input on the system logging device (R).

CODE	FUNCTION
01	End of Program

4-4.2.3 Bid A Task. Illegal call for PSM. This call applies to PAM only.

CODE	FUNCTION
02	Bid A Task

4-4.2.4 Multiply. The multiply function effective address must contain the memory address of the multiplicand. The multiplier must be in Worker Register Zero. The product will be placed in Worker Registers Zero and One (B).

CODE	FUNCTION
03	Multiply

4-4.2.5 Divide. Te divide function effective address must contain the memory address of the divisor. The dividend must be in Worker Registers Zero and One. The quotient is placed in Worker Register One with the remainder placed in Worker Register Zero (B).

CODE	FUNCTION
04	Divide

4-4.2.6 Shift Memory Circular Left Double. The shift memory circular left double function effective address contains the memory address of the value to be shifted. This word and the next word in memory, treated as a 32-bit value, are rotated left the number of positions specified in Worker Register Two (B).

CODE	FUNCTION
05	Shift Memory Circular Left Double

4-4.2.7 **End Of Job.** In PSM this function is the same as the end of program (R).

CODE	FUNCTION
06	End Of Job

4-4.2.8 **Square Root.** The effective address is not used. The argument (double precision integer) must be right justified in Worker Registers Zero and One. The integer square root returns in Worker Register Zero (O).

CODE	FUNCTION
07	Square Root

4-4.2.9 **Convert Binary To ASCII Coded Hexadecimal.** The convert binary to ASCII coded hexadecimal function effective address must contain the memory address of a two-word array where the converted result is placed. Worker Register Zero must contain the binary value to be converted (B).

CODE	FUNCTION
08	Convert Binary To ASCII Coded Hexadecimal

4-4.2.10 **Convert Hexadecimal ASCII To Binary.** The convert hexadecimal ASCII to binary function effective address must contain the memory address of a two-location array containing the hexadecimal ASCII value. The binary result will be placed in Worker Register Zero (O).

CODE	FUNCTION
09	Convert Hexadecimal ASCII To Binary

4-4.2.11 **Convert Binary To ASCII Coded Decimal.** The convert binary to ASCII coded decimal function effective address must contain the address of a three-location array where the converted result is placed. Worker Register Zero must contain the binary value to be converted (B).

CODE	FUNCTION
OA	Convert Binary To ASCII Coded Decimal

4-4.2.12 **Convert Decimal ASCII To Binary.** The convert decimal ASCII to binary function effective address must contain the memory address of a three-location array containing the decimal ASCII value. The binary result returns in Worker Register Zero (O).

CODE	FUNCTION
OB	Convert Decimal ASCII To Binary

4-4.2.13. The next six supervisor calls are defined for PAM only. They are treated as errors when encountered by PSM. They are listed here for reference only.

CODE	FUNCTION
OC	Time Delay
OD	Wait – Unconditional
OE	Activate "Waiting" Task
OF	Wait for Interrupt
10	Get Date and Time
11	Get Data Block from Another Task

4-4.2.14 **Convert Fixed Point To Floating Point.** The effective address is not used. The double precision integer argument must be in Worker Registers Zero and One. The floating point equivalent returns in Worker Registers Zero and One (O).

CODE	FUNCTION
12	Convert Fixed Point To Floating Point

4-4.2.15 **Convert Floating Point To Fixed Point.** The effective address is not used. The floating point number in Worker Registers Zero and One is converted to integer and returned in Worker Registers Zero and One (O).

CODE	FUNCTION
13	Convert Floating Point To Fixed Point

4-4.2.16 **Floating Point Add.** The effective address contains the location of a two-word block which contains the floating point number to be added to the floating point number in Worker Registers Zero and One. The result returns in Worker Registers Zero and One (O).

CODE	FUNCTION
14	Floating Point Add

4-4.2.17 **Floating Point Subtract.** The effective address contains the location of a two-word block which contains the floating point number to be subtracted from Worker Registers Zero and One. The result returns in Worker Registers Zero and One (O).

CODE	FUNCTION
15	Floating Point Subtract

4-4.2.18 Floating Point Multiply. The effective address contains the location of a two-word block which contains the floating point number to be multiplied by that in Worker Registers Zero and One. The result returns in Worker Registers Zero and One (O).

CODE	FUNCTION
16	Floating Point Multiply

4-4.2.19 Floating Point Divide. The effective address contains the location of a two-word block which contains the divisor. The dividend is in Worker Registers Zero and One. The quotient returns in Worker Registers Zero and One (O).

CODE	FUNCTION
17	Floating Point Divide

4-4.2.20 Convert Floating Point to Decimal ASCII. The effective address contains the location of a six-word array into which the results are placed (in E format). The number to be converted is in Worker Registers Zero and One. For example, if the floating point number had the value π, the conversion would produce an ASCII representation of the following:

+.314159E + 01 (O).

CODE	FUNCTION
18	Convert Floating Point To Decimal ASCII

4-4.2.21 Floating Point Sine. The effective address is not used. The floating point representation of the angle X in radians is in Worker Registers Zero and One. The size of the angle must be $-\pi/2 = X = \pi/2$. The floating point representation of SIN(X) returns in Worker Registers Zero and One (O).

CODE	FUNCTION
19	Floating Point Sine

4-4.2.22 Floating Point Cosine. Arguments are the same as in paragraph 4-4.2.21 except COS(X) returns (O).

CODE	FUNCTION
1A	Floating Point Cosine

4-4.2.23 Floating Point Arctangent. The effective address contains the location of Argument B (floating point). Argument A is in Worker Registers Zero and One. The angle of the value of which is ARCTAN (A/B) returns in Worker Registers Zero and One.

CODE	FUNCTION
1B	Floating Point Arctangent

4-4.3 FUTURE SUPERVISOR CALLS. Future Supervisor Calls will be written as needed. Sixty-four, maximum, can be used (limited by the size of the operation code sector of Worker Register Three).

4-4.4 I/O CALLS AND THE PHYSICAL RECORD BLOCK. The Physical Record Block (PRB) is a five-location array containing the detailed information needed by the PSM to execute an input/output operation.

4-4.4.1 PRB Relative Word Zero. Relative Word Zero must contain the address of the first memory location of the associated WTB.

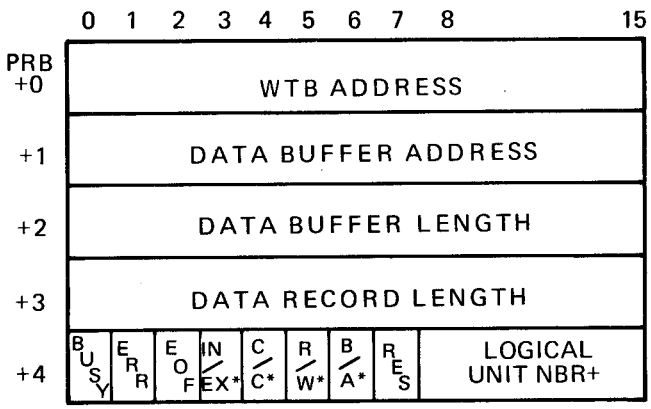
4-4.4.2 PRB Relative Word One. Relative Word One must contain the address of the first memory location of the associated data buffer area.

4-4.4.3 PRB Relative Word Two. Relative Word Two must contain the length (character count) of the associated data buffer area. Input operations continue until the buffer is full, or an end character is read.

4-4.4.4 PRB Relative Word Three. Relative Word Three must contain the record length (character count) of the associated data. The number of characters contained in Relative Word Three is output. Input operations store the number of characters actually input in this word.

4-4.4.5 Relative Word Four. Relative Word Four is divided in half. The left half (bits 0-7) is dedicated to input/output flags which signify operational status. The flag assignments are given in Table 4-5. The second half (bits eight through fifteen) must contain the Logical Unit Number (LUNO) as selected by the user.

PHYSICAL RECORD BLOCK



* – SET BY CALLING TASK

\+ – LOGICAL UNIT NUMBERS 0 THRU F
SYSTEM RESERVED; FF ILLEGAL

TABLE 4-5.
FLAG ASSIGNMENTS

PRB BIT		FLAGS (Word 4, Bits 0-7)
0	BUSY	1 = I/O in progress, 0 = I/O complete.
1	ERR	1 = Error on last operation, 0 = no error.
2	EOF	1 = End of File (/*) on last call, 0 = no EOF.
3**	IN/EX	1 = Initiate Call (program returned to immediately), 0 = Execute Call (program returned to when I/O is complete).
4	CON	1 = Control Call (e.g. "Punch Leader"), 0 = Character I/O.
5	R/W	1 = Read (input), 0 = Write (output).
6	B/A	1 = Binary record, 0 = ASCII record.
7	RES	Reserved.

** Bit 3 is used by PAM only. PSM treats all calls as execute calls.

4-4.5 End Vector. In order for the loader to return to PSM after loading a task, the task must contain a special end vector. This vector is the address of a two-word status block. The first word in the status block contains a X'007F' and the second word contains a zero. This allows the loader to return to PSM by executing a LDS instruction that addresses the status block supplied. The following example illustrates the end vector to be included on each task run under PSM:

```
        .
        .
        .
LABLE   DATA  X'7D',0
        END   LABLE
```

This end vector does not prevent running this task under PAM, for PAM will ignore it.

4-5 OPERATING PROCEDURES.

4-5.1 PRIMITIVE LOADERS. Two Primitive Loaders are provided: one for systems equipped with a card reader and the other for paper tape systems. Both are designed to load the system's Relocating Bootstrap Loader.

The Primitive Loader is entered into memory via the computer control panel.

4-5.1.1 Card Media Primitive Loader. The Card Media Primitive Loader reads a card file of any length. The first 32 columns of each record are read (each column containing a 1 or 0), and these binary values are entered sequentially in memory, two words (32 bits) per card read.

The loader occupies any 36 sequential memory words except the 16 virtual register locations or the area to be occupied by the bootstrap loader.

Table 4-6 contains the actual object, in hexadecimal, along with the word's relative address and a sample absolute address using X'0400' as a load bias.

TABLE 4-6
CARD MEDIA PRIMITIVE LOADER

RELATIVE ADDRESS	EXAMPLE ABSOLUTE ADDRESS	MEMORY CONTENT
00	400	340F
01	401	0800
02	402	3001
03	403	0002
04	404	340F
05	405	0000
06	406	4482
07	407	0001
08	408	7007
09	409	0000
0A	40A	340F
0B	40B	0800
0C	40C	4481
0D	40D	000F
0E	40E	6261
0F	40F	0000
10	410	300F
11	411	0810
12	412	300E
13	413	0816
14	414	8800
15	415	F800
16	416	300F
17	417	0016
18	418	0C9F
19	419	000E
1A	41A	46E6
1B	41B	0001
1C	41C	0CAF
1D	41D	000C
1E	41E	3007
1F	41F	001E
20	420	3000
21	421	0020
22	422	72D2
23	423	0000

Since no direct addressing is used in the Primitive Loader, the operator need not concern himself with relocating operand fields or constants.

4-5.1.2 Data Terminal/Teletypewriter Primitive Loader. The Data Terminal/Teletypewriter Primitive Loader reads a single paper tape record of any length consisting solely of the tape code acceptable to the primitive loader. Each frame of tape contains the code for a single hexadecimal character.

The loader accepts the codes shown in Table 4-7.

TABLE 4-7
TAPE CODES

PRIMITIVE LOADER CODE (hexa-decimal)	EQUIV-ALENT ASCII	BINARY CODE (hexa-decimal)	EQUIV-ALENT HEXA-DECIMAL DIGIT
40	(@)	60	0
41	(A)	61	1
42	(B)	62	2
43	(C)	63	3
44	(D)	64	4
45	(E)	65	5
46	(F)	66	6
47	(G)	67	7
48	(H)	68	8
49	(I)	69	9
4A	(J)	6A	A
4B	(K)	6B	B
4C	(L)	6C	C
4D	(M)	6D	D
4E	(N)	6E	E
4F	(O)	6F	F

Blank frames and rub outs are ignored. The end of record is denoted by X-OFF (X'0013').

The Data Terminal/Teletypewriter Primitive Loader occupies 28 sequential words of memory.

Table 4-8 contains the actual object in hexadecimal, along with the word's relative address and a sample absolute address using X'0400' as a load bias.

Since direct addressing is used, the operator need not be concerned with relocating operand fields or constants.

4-5.1.3 Primitive Loader Loading Instructions. Either Primitive Loader may be loaded via the operator control panel by following these steps:

a. Determine desired memory area for loading (load bias).

b. Select: HALT-RESET-CLEAR

c. Select: MEM ADD (memory address).

d. Enter Absolute Address on console DISPLAY REGISTER.

e. Select: ENTER

f. Select: MEM DATA (Memory Data)

TABLE 4-8
DATA TERMINAL
TELETYPEWRITER PRIMITIVE LOADER

RELATIVE ADDRESS	EXAMPLE ABSOLUTE ADDRESS	MEMORY CONTENT
00	400	340A
01	401	0800
02	402	4484
03	403	0000
04	404	4482
05	405	0003
06	406	4480
07	407	0000
08	408	3409
09	409	0000
0A	40A	3409
0B	40B	080A
0C	40C	2000
0D	40D	5081
0E	40E	3004
0F	40F	0008
10	410	3006
11	411	0808
12	412	6004
13	413	0080
14	414	5000
15	415	0081
16	416	0CAF
17	417	0008
18	418	4AE0
19	419	0000
1A	41A	0CE1
1B	41B	0000

ABSOLUTE ADDRESS = RELATIVE ADDRESS + LOAD BIAS

g. Select: CLEAR

h. Enter value corresponding to absolute address (step d) from MEMORY CONTENT column into console DISPLAY REGISTER.

i. Select: ENTER

Repeat steps c thru i until the Primitive Loader is loaded. To verify the contents of a location:

a. Select: HALT

b. Select: MEM ADD

c. Select: CLEAR

d. Enter absolute address of location to be displayed into console DISPLAY REGISTER.

e. Select: MEM DATA

f. Select: LOAD

The contents of the specified address are now displayed on the console DISPLAY REGISTER. A new value for that address may now be entered in the following manner:

a. Select: CLEAR

b. Enter desired value into console DISPLAY REGISTER.

c. Select: ENTER

4-5.2 RELOCATING BOOTSTRAP LOADER. Two Bootstrap Loaders are provided: One each for card and paper tape systems. Both Loaders perform the same function of loading relocatable object output from either SAL960 or the Linking Relocating Loader. Only one may be resident at any time.

To load either Relocating Bootstrap Loader:

a. Load the appropriate Primitive Loader.

b. Place the Bootstrap Loader special object file in input device.

c. Select: HALT-RESET-CLEAR.

d. Enter the first word address of Primitive Loader into Supervisor Register Five (location X'0085').

e. Clear (enter zero) Supervisor Register Six (location X'0086').

f. Enter the CRU Base Address of the object input device into Supervisor Register Seven (location X'0087').

g. Load Status Register with X'01C0' by:

(1) Select: CLEAR

(2) Select: ST (Status)

(3) Enter X'01C0' into console display register.

(4) Select: ENTER

h. Load Program Counter with Primitive Loader's first address by:

(1) Select: CLEAR

(2) Select: PC (Program Counter)

(3) Enter first address of Primitive Loader into Console Display Register.

(4) Select: ENTER

i. Select: OVER MEM PROT (Override Memory Protect)

j. Select: RUN-START

k. Start object input device.

l. The Primitive Loader is now active.

m. After the Bootstrap Loader has been read, select: OVER MEM PROT or optionally RESET.

4-5.3 PROGRAM SUPPORT MONITOR OPERATION. To load PSM via the Relocating Bootstrap Loader (PSM should be loaded at the highest part of core possible):

a. Determine desired load address of PSM.

b. Select: HALT-RESET-CLEAR.

c. Place the PSM object followed by /* record into object input device.

d. Select: OVER MEM PROT to unprotect memory and:

(1) Enter the following values into the specified addresses:

Absolute Hexadecimal Address	Value
7D	X'7C00'
7E	PSM load bias + X'0103'
7F	PSM load bias + X'0100'

(2) Select: OVER MEM PROT: memory is now protected.

e. Enter the load address (1) into Supervisor Register Zero (location X'0080').

f. Load Status register with X'01C0'.

g. Load Program Counter (PC) with 2.

h. Select run start.

i. If input is on cards, start card reader.

j. After the end-of-file record has been read, proper loading of PSM may be assumed if INSTRUCTION REGISTER A contains X'7082' and REGISTER B contains load bias + X'011B'.

k. If this condition does not exist, refer to paragraph 4-3.2. Re-execution of steps a-j may be necessary. Failure may have occurred because no end-of-file was present. If this is the case, read a /* record via the object input device.

4-5.4 WORKER TASKS. If the condition described in step j of 4-5.3 has been met, PSM is ready to load and execute a worker task. If this worker task needs any logical device assignment, it should be done before the task is loaded. Refer to paragraph 4-2.5. Worker Tasks are loaded starting at X'00B0' and PSM is structured so that the first X'0100' location can be used by the worker task. Thus, the worker task can have all of core from X'00B0' to PSM + X'0100'.

To load a task:

a. Place Worker Task object followed by an end-of-file (/*) record into object input device.

b. If the card reader is the input device, start the card reader.

c. Strike L on the system logging device keyboard.

Loading will start automatically at location X'00B0'.

At the conclusion of object input, INSTRUCTION REGISTERS A and B should contain X'7082' and load address + X'011B' respectively. If this is not the case, check that the worker task object deck was followed by an end-of-file (/*). If it was not, an end-of-file may be read at this time.

If an end-of-file has been read and the Instruction Registers do not contain the proper values, refer to paragraph 4-3.2.

The task is now loaded and is ready for execution. Any alterations to the task may be made at this time via the operator console.

To execute the task, strike X on the system logging device keyboard.

4-6 OPTIONAL SEPARATELY LOADABLE PROGRAMS.

There are four programs that perform useful functions and which run independent of PSM.

They can be used to:

a. Dump a variable part of memory in hexadecimal and ASCII onto the teletypewriter.

b. Dump a variable part of memory in hexadecimal and ASCII onto the line printer.

c. Patch memory from the card reader.

d. UNLOAD memory (binary punch) in reloadable format.

The first three of these tasks can be loaded with the Bootstrap Loader from the operating console (not by PSM, however, except following a worker task). They are entered with an SXBS instruction from a worker program, or they can be controlled from the operating console. If entered via an SXBS instruction, they will return to the worker program after completion. If operated from the operating console, they will halt after completion. The UNLOAD memory program must be loaded and executed from the control panel.

4-6.1 DUMP MEMORY ON TELETYPEWRITER. This program assumes the CRU base address of the teletypewriter is zero. If it is different from this, location X'000C' of it should be set equal to the CRU base address.

4-6.1.1 From The 960 Operating Console (Assume the task is loaded at ADDRES).

a. Set PC = ADDRES

b. Set ST = X'01C0'

c. Set memory address X'0080' = first word address to be dumped.

d. Set memory address X'0081' = last word address to be dumped.

e. Select RUN – START.

4-6.1.2 From A Worker Program.

```
LA      1,FWA       FWA = first word address to
                          be dumped
ST      1,X'80'
LA      1,LWA       LWA = last word address to
                          be dumped
ST      1,X'81'
SXBS    ADDRES +4
```

4-6.2 DUMP MEMORY ON LINE PRINTER. This routine assumes that the line printer has a CRU base address of X'0800'. If different from this, then set location X'000C' of the program equal to the proper CRU base address.

The instructions for operating this program are the same as those given in paragraph 4-6.1.

4-6.3 PATCH MEMORY FROM CARD READER. This routine assumes that the card reader has a CRU base address of X'0400'. If different from this, then set location X'000C' of the program equal to the proper CRU base address.

4-6.3.1 Card Format.

The card format is: (all numbers are hexadecimal)

Col 1

XXXX.NNNN,MMMM,OOOO,PPPP,...

Where XXXX is the memory address where NNNN will be placed. MMMM will go into address XXXX+1, 0000 will go into address XXXX+2, etc. The first blank space terminates a card.

These can be as many cards of this format as desired in a patch deck. the patching operation is terminated by a /* card.

Example of a patch deck.

```
0100,0104,2F00,1C00,FFFF
A102.F000,BCDE,0000,0000,0000
1402.0000
070F.0001,0020
/*
```

4-6.3.2 Operating Procedures From The 960 Operating Console (Assume the patch program is loaded as ADDRES).

a. Place patch cards in the card reader.

b. Turn on the card reader.

c. Set PC = ADDRES

d. Set ST = 01C0

e. Select RUN – START

When all cards have been read, the patch program will halt (B $).

4-6.3.3 Referencing The Patch Program From A Worker Program (Assume that the patch program is loaded at ADDRES). Execute an SXBS ADDRES +4 instruction.

The patch cards must be in the card reader and it turned on.

After all cards have been read, execution returns to the worker program.

4-6.4 UNLOAD MEMORY (BINARY PUNCH). UNLOAD is a worker mode program which punches selected portions of memory in a format which closely resembles that of output from the SAL960 Assembler. The only differences are:

a. The text records contains no linkage or relocation data.

b. Each text record contains an absolute starting address.

c. The end record contains no end vector.

These restrictions require that the segment unloaded cannot be linked with another segment and, when loaded, it must be assigned a load bias of zero.

UNLOAD runs with the assistance of either PSM or PAM, and uses two logical unit numbers which need assignment (paragraph 4-2.4) They are:

a. C – for I/O communication and must be assigned to the teletypewriter keyboard.

b. D – punch output device; card punch or tape punch.

Operating Procedures:

a. Using the bootstrap loader, load UNLOAD from the computer operating console.

CAUTION

Do not overwrite the program or data to be unloaded.

b. Ready the punch device.

c. UNLOAD will print the message INPUT SEGMENT NAME (6 CHARACTERS) and wait for the operator to enter the desired characters on the keyboard.

d. After the operator enters the desired data, the UNLOAD program prints INPUT PUNCH INTERVALS. Then the operator must enter the beginning and ending addresses of the core portion he desires to punch out. This input must not exceed ten intervals and must end with a ↑ symbol and a carriage return. The punch operation begins. Each punch interval consists of two four-digit hexadecimal numbers.

EXAMPLE: 0000 000F 024C 0FFF ↑

This means that core locations 0000 to 000F and 024C to 0FFF are punched.

e. UNLOAD terminates by returning control to the monitor.

f. If the output device is the card punch the first and last cards of a punched card deck should be discarded.

The core dump has no end vector. To load this data:

a. Place the object in the loading device.

b. Using the bootstrap loader, load with a load bias of zero.

c. Wait until loading is complete and set the program counter to X'007D'.

d. Select RUN and START.

4-6.5 SOURCE MAINTENANCE ROUTINES. The Source Maintenance Routines (SMR) are a series of routines combined into a single program which runs under monitor control. These routines can copy and sequence a source file, read a source file or parts of a source file into memory, list all or selected parts of a source file in memory, or insert or delete source statements from a file. One or more of the available options may be executed on any one program call on one source file by the appropriate responses to program questions. If maintenance is to be performed on a second source program, the SMR must be terminated and recalled.

SMR can have up to four file assignments which are made before SMR is loaded. If the letter-message N is printed in response to the program query HAVE I/O DEVICES BEEN ASSIGNED, the program terminates and must be called again after the assignments have been made. File logical unit numbers are:

Source Input File	Logical Unit 10
Source Output File	Logical Unit 20
Listing File	Logical Unit 30
Correction File	Logical Unit 40

Upon execution, the program types questions concerning the input and output devices, all of which can be answered with letter-message Y or N. The initialization routine also allows dynamic buffer allocation. All core between the end of the SMR and the start of the monitor is available for buffers. These limits are printed on the teletypewriter, but the upper limit may be lowered via keyboard input, if desired. The lower buffer limit cannot be changed.

Source input and/or output may be on either cards or paper tape. Corrections may be made via the keyboard, cards, or paper tape. All keyboard entries without an exclamation point as the first character are rejected unless an INSERT command has been entered. On keyboard entries, a ← symbol deletes the last character entered. If a source correction, a ↑ symbol deletes the current input line, carriage return, and waits for a corrected entry.

If the source is output to paper tape, the SMR punches leader before the first input and after the last output.

Commands accepted by SMR are:

!p,n — Reads source input into memory up to sequence number p, then skips records on source input file until sequence number n is read. When all source line buffers are filled, the records are output to the source output file if output was indicated. More input is read until the p and n requirements are met. If unsequenced records are being read, put 7FFF in both p and n fields. Source statements may be added, deleted, or listed after completion of this command.

!LSA — Lists all source statements in memory. The sequence number is listed on the left, followed by the source line.

!LSn,m — Deletes source lines in memory from sequence numbers n to m inclusive. Put delete code in proper records so they are bypassed on output.

!! — Terminates the program and returns control to the monitor. If output is indicated, writes any input records not previously output.

!RPn — Copies input file to output file and resequences, starting at sequence number zero and incrementing by n. No additions or deletions can be made since all input is automatically output.

All sequence numbers entered as part of a command must be four decimal digits in length.

If an illegal op code or sequence number is entered, ILLEGAL ENTRY will be typed and a new entry requested.

4-7 SYSTEM GENERATION

The PSM system has been structured and subdivided into segments in such a manner as to make system generation as easy as possible. When generating a system, source modification need be made to only one segment. Linkable object is available for all other segments. The only operations necessary are to assemble the one source segment and to link it to the other segments. Any PSM capable of running the SAL960 assembler and the Linking Relocating Loader can be used to generate another PSM.

Three things affect the structure of PSM. They are:

a. The type and number of each type of I/O devices.

b. The structure of the system service routine section (i.e., which systems service routines will be included or excluded).

c. The logical device assignment table capacity.

NOTE

Linkable object exists for three different configurations of this segment. If any of these three is sufficient, then no assembly is necessary.

4-7.1 TYPE AND NUMBER OF I/O DEVICES To change the I/O device structure, source changes must be made to a Supervisor Data Segment SDATA and a service routine for each type of I/O device must be included when the system is linked. All of the standard I/O device service routines available for PSM are written in a reusable manner, thus one device service routine can control as many devices of a particular type as desired (i.e., the teletypewriter device service routine can control any number of teletypewriters or data terminals.

4-7.1.1 Adding I/O Devices. To add new I/O devices, two source changes must be made to SDATA. They are:

a. Add a Physical Device Table.

b. Make an entry in the device service routine address table (DSRAT).

The source listing of SDATA is commented in such a manner that these changes can be accomplished by following the direction of the comments. Figure 4-3 shows how to add a line printer.

4-7.1.2 Deletion of I/O Devices. If at some time it is desired to delete an I/O device from a system, three things should be done.

a. Remove the Physical Device Table for that device from SDATA.

b. Replace the entry for that device in the device service routine address table (DSRAT.)

c. If there is no other device of this type in the system, exclude the device service routine from the linking operation.

```
0000    LP1PDT    RES    0
                  *
                  *              To add DP 2310 #1
                  *              3 things must be done
                  *
                  *        (1)   Replace the $ in the address field of the next card with the CRU base
                  *              address
0000    LP1CRU    FQU    $
                  *
                  *        (2)   Remove the * from col. 1 of the next 2 cards
                  *
                  *        REF   LP000
                  *        DATA  LP000.0.X'300F'.LP1CRU.$+5.0.0.0.0
                  *
                  *        (3)   Remove the * from col. 1 of the next card
                  *        DATA  LP000.LP1PDT
                  *
                  *              and place it in the devise service routine address table in this segment
                  *
                  *        Be sure to include segment LP000 in the linking operation
```

Figure 4-3 Physical Device Table for LP2310. Line Printer No. 1.

The order of the DSRAT determines the physical device number of a device. If an entry is deleted from the middle of the table, then the physical device number for the entries following it will change. To prevent this, replace the entry with DATA BADPDN,O. This keeps the table from being re-ordered and if this device number is referenced by a worker program, PSM will come to an error halt.

4-7.2 SYSTEM SERVICE ROUTINE STRUCTURE To add or delete a system service routine from a system, just include or exclude the linkable object for that routine in the linking operation.

Including a particular service routine in the linking operation will cause the error message *doubly defined symbol* for that system service routine name to be given by the Linking Relocating Loader. Disregard this message.

4-7.3 LOGICAL DEVICE ASSIGNMENT CAPACITY The basic SDATA allows for a maximum twelve logical device assignments. To add to this number add more DATA -1 statements to the Logical Device Table (LDTST) in segment SDATA.

4-7.4 PSM SEGMENTS REQUIRED

DESCRIPTION	NAME	CD	PT
Supervisor*	SPB	218224-0001	218225-0001
CRU Interrupt Decoder**	CINTSG	218242-0001	218243-0001
Teletype-writer Service Routine***	TTYSEG	218247-0001	218248-0001
Supervisor Data	SDATA	–	–
End of Record Routine	FILEOR	218407-0001	218407-0001

If the PSM being generated is not a teletypewriter only system, then the following segment must be included when the system is linked.

I/O Common Subroutine	GETCO1	218402-0001	218403-0001

*The SPB Supervisor must be the first segment in any PSM system. Therefore, it must be the first segment in the linking operation.

**This segment must be second in the linking operation.

***This segment must be last in the linking operation.

4-7.5 SYSTEM GENERATION SUMMARY.

a. Read all of paragraph 4-7 (through 4-7.5) carefully.

b. Decide what I/O devices are needed.

c. Add a Physical Device Table for each I/O device to the supervisor Data Segment (SDATA).

d. For each I/O device add an entry in the DSRAT in SDATA.

e. If the maximum number of logical devices assigned is to be increased, add DATA -1 to the SDATA Logical Device Table.

f. Assemble the SDATA.

g. Use the LRL to link the following segments.

DESCRIPTION	NAME	CD	PT
Supervisor*	SPB	218224-0001	218225-0001
CRU Interrupt Decoder**	CINTSG	218242-0001	218243-0001
End of Record Routine	FILEOR	218407-0001	218407-0001
Device Service Routines	–	–	–
System Service Routines	–	–	–
Supervisor Data***	SDATA	–	–
I/O Common****	GETCO1	218402-0001	218403-0001

*The SPB Supervisor must be the first segment in any PSM system. Therefore, it must be the first segment in the linking operation.

**This segment must be second in the linking operation.

***This segment must be last in the linking operation.

****Include this only if the PSM is not a teletypewriter-only system.

SECTION V

UTILITY PROGRAMS

5-1 INTRODUCTION

This section describes the utility programs available to the TI 960 PAM user. These utility programs are:

a. PAM Logical Unit Number Assignment Display

b. PAM Task Status Display

c. PAM Message Writer Task

5-2 PAM LOGICAL UNIT NUMBER ASSIGNMENT DISPLAY

This 240-word utility program lists all current logical unit assignments with their status on the logging device and the number of possible assignments remaining (Table 5-1). The loading procedure for this utility is the standard task loading procedure described in Section III. Logical unit assignments are unnecessary for task execution.

5-3 PAM TASK STATUS DISPLAY

This 434-word utility program outputs a list of currently installed tasks and their status on logical unit X'00F8'. The Task Status Display program (Table 5-2) also outputs the 960 duty cycle for the current second, the maximum duty cycle since PAM initiation, the time, and the date. This

TABLE 5-1
LOGICAL UNIT NUMBER ASSIGNMENT DISPLAY

LUN NUM	DEV NUM	TASK ID	PRB ADDR	NEED SVCE	ASSND TO TSK
09	05	01	2EFF	F	F
02	04	02	2BBF	F	F
03	05	02	2BFF	F	F
08	04	02	2BC4	F	F
00	00	20	0476	F	F

TABLE 5-2
PAM TASK STATUS DISPLAY

```
$$DFIO **00F8 ** 0005          DEFINE OUTPUT TO LINE PRINTER (5)
$$LDTS **0300 ** 0033               LOAD STAT DISPLAY
$$INST **0033 ** 0033          INST 33 AT PRI 33
$$ABLE **0033
$$EXCT **0033

TIME     15:  02
DATE    148:  1970

THE CURRENT 1 SEC DUTY CYCLE IS  8 PER CENT.
THE PREVIOUS MAXIMUM DUTY CYCLE WAS 17 PER CENT.
```

TASK RANK	TASK ID	PROC ID	WTB ADDR	ENTY PT	LAST EC	ABLE	BID	STATUS SUSP	TSUS	TD
00	00	FF	1A6E	1AA5	1AA5	T	F	F	F	F
01	01	FF	1DE0	1E5D	1E50	T	F	F	F	F
02	02	FF	1B03	1B85	1B85	T	F	F	F	F
33	33	FF	0300	03E2	04A2	T	T	F	F	F

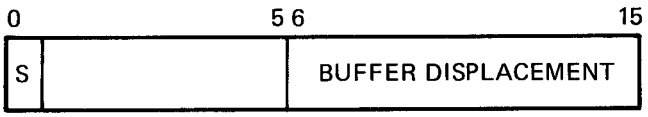
```
$$RLIO ** 00F8
$$DLTS ** 0033
$$JCOF **
```

program requires a version of PAM that includes the optional supervisor calls DIVIDE and CBDA and uses the standard task loading procedure (refer to Section III).

5-4 PAM MESSAGE WRITER TASK

This 428-word utility program is used by one or more worker tasks under PAM to output task messages on logical unit X'00FE' without suspending execution of more pertinent task functions and to minimize output buffer size within the user tasks.

The Message Writer Task service is requested by any task that sets bit zero of memory location 17 + load bias equal to one. Proper servicing requires that the last 10 bits of this location contain the displacement of the user's message buffer. This buffer is limited to a maximum of 36 words and the first word must contain the hexidecimal number of the ASCII characters in the message + 2 to define the buffer size.



S = 1 FOR SERVICE

The loading procedure for this task is non-standard in that a patch function (refer to Section III) is used with the standard control cards. This function is used to modify the Message Writer's user dictionary pointer location, load bias +X'00FE', to contain the address of the top ID of the user dictionary and to load into the dictionary the ID(s) of those user task(s) which are to be eligible for message service. User Task ID(s) fill all of the user dictionary from the address contained in the dictionary pointer location to the end of the message data file, load bias + X'01AC'. The ID format for the user dictionary is X'00––'.

In addition to the preceding required user modifications the user may optionally extend the data file length by storing in location "load bias + $E2_{16}$" the end address of the desired file. It is the user's responsibility to protect other tasks and the monitor when the file is extended. The need for file extension may occur when the message writer task has very low priority and/or many users.

The Message Writer Task requires a PAM version that includes the optional monitor Time Delay and Get Data From Another Task calls.

In the following example, three tasks, which execute repeatedly, request the message writer to output messages of the form Worker Task N, where N is the Task ID. The actual output is shown below:

```
WORKER TASK 11
WORKER TASK 12
WORKER TASK 14
WORKER TASK 14
WORKER TASK 11
WORKER TASK 12
WORKER TASK 14
WORKER TASK 11
WORKER TASK 12
WORKER TASK 14
```

SECTION VI

LINKING RELOCATING LOADER

6-1 INTRODUCTION

The LINKING RELOCATING LOADER (LRL960) links separately assembled programs and program segments by combining the text and completing the assembly process for external symbols.

6-2 LRL DESCRIPTION

6-2.1 GENERAL. Refer to Figure 6-1. The input and output object files are in 960 binary format. Control is assigned to an input device, normally a card reader or a teletype. The load map and error list are assigned to an output device such as a line printer or a teletype printer. LRL will optionally load a program as it is linked.

6-2.2 OPTIONS. Several options have been incorporated in LRL to provide for easy operation on most 960 systems:

a. List a Load Map. Flag External Symbols that are undefined or defined more than once.

b. List an Error Summary.

c. Output the binary object file only.

d. Load the program to an available core area as the program is linked.

6-2.3 EQUIPMENT CONFIGURATION. The minimum equipment configuration includes:

a. Model 960 computer with 4K Memory.

b. Device for binary object input file.

c. Device for binary object output file.

d. Device for control messages and input.

e. Device for Lists (Load Map and Error Summary).

LRL will operate with all files and control assigned to a single teletypewriter. Operation with only the teletype Corporation Model ASR-33 TBE (Manual Punch Control) is somewhat inconvenient. A program is available for separating the input binary records from the output records. Two teletypewriters are recommended as the minimum. I/O equipment configuration. Automatic punch control installed on a single teletypewriter will increase effectiveness. Optimum effectiveness is obtained with either of the following I/O equipment configurations.

CARD MEDIA

Card Reader
Card Punch
Data Terminal or Line Printer
Teletypewriter, if no Data Terminal

PUNCHED TAPE MEDIA

High Speed Paper Tape Reader
High Speed Paper Tape Punch
Data Terminal or Line Printer
Teletypewriter, if no Data Terminal

6-2.4 DATA STRUCTURE. Four types of 960 Binary Records are processed by LRL.

a. Segment Identification Records which contain

(1) Segment Name

(2) Origin, Length

(3) External Reference Count

(4) Segment Attribute Flags

b. Linkage Data Records which contain

(1) Number of symbols per card

(2) Symbol attribute Flags for each symbol

(3) External References accompanied by an index or External Definitions accompanied by a value.

c. Text Records which contain

(1) The number of text words in the record

(2) A Load address

(3) Relocation Map

(4) Linkage Data

(5) The number of Linkage Data Sets in the record

d. End Records which may contain an optional branch vector.

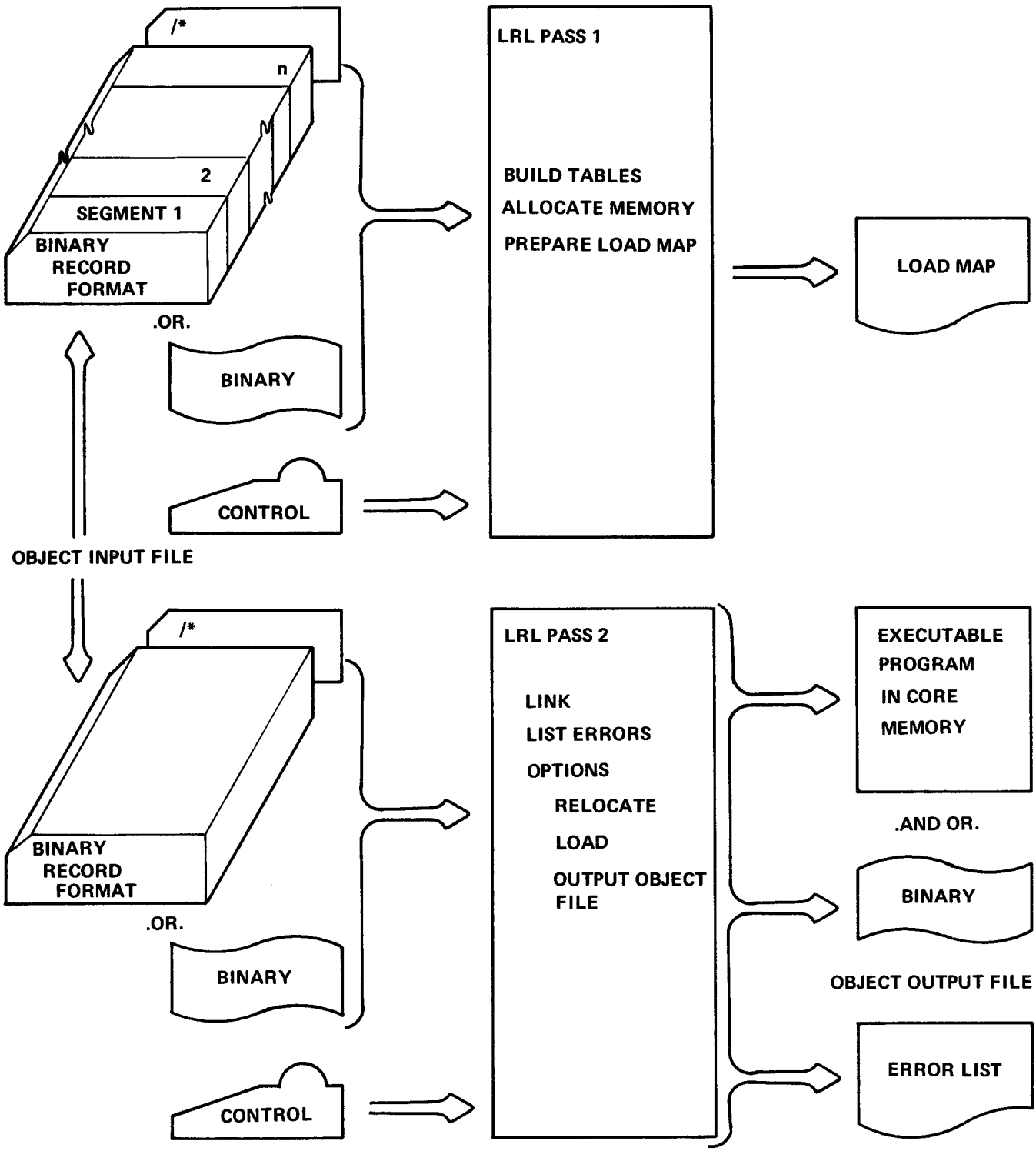


Figure 6-1 LRL960 – General Operating Procedure.

A detailed description of each binary record type can be found in Section II.

The segment name from each identification record processed is placed in a segment identification table (IDTAB). Space is allocated for one hundred segment names.

Symbols obtained from Linkage Data Records are processed by constructing a symbol table starting at one end of the symbol table work space. An External Symbol Reference list is constructed starting at the opposite end. Data in the symbol table is referenced by symbol name. The reference list is addressed using an index computed from the position of a given program segment in the input file and the external symbol index for a symbol within the given program segment.

Text Records are processed during pass two of the binary input file.

End Records provide separation between programs that have been assembled separately. The segment sequence number of the END record is retained for use in calculating indexes for the external symbol reference list.

LRL automatically uses the memory between LRL and PSM/PAM first address for the construction of the symbol table. WHEN USING PAM NO WORKER TASKS SHOULD RESIDE BETWEEN LRL AND THE FIRST PAM ADDRESS.

6-2.5. PROGRAM STRUCTURE. LRL program structure is divided into a control segment, first-pass segment, and second-pass segment. The control segment reads the input record and determines which record type is to be processed.

Only identification records and linkage data records are processed in the first pass. The option to print a load map may also be exercised.

6-2.6 LINKING LOADER OUTPUT TAPE EDIT. A link load performed with the binary input file assigned to a teletypewriter paper tape reader and the binary output file assigned to the same teletypewriter's paper tape punch (which has no automatic punch on/off) will cause records from the binary input file as well as records from the binary object to be punched on the tape. This tape must be edited before it can be loaded.

OBJMAT is a worker program which runs under control of either PSM or PAM that performs this edit task. It reads the LRL binary output file tape, deletes the extraneous data, checks redundancy characters and punches a new tape. The logical units used by this program are:

LUNO	ASSIGNED
000A	TTY Keyboard/printer
000B	TTY punch tape reader/punch

OBJMAT OPERATING INSTRUCTIONS:

a. Use WFDIO to make Logical Unit Number Assignments.

b. Load OBJMAT and execute.

c. Put the output tape from LRL into the TTY tape reader.

d. Respond to the OBJMAT message TURN OFF PUNCH AND RETURN CARRIAGE.

e. OBJMAT then reads the input tape, bypassing the extraneous data, and builds a 800-word buffer of good data.* If a redundancy character error occurs, the message REPOSITION TAPE IO LAST RECORD AND RETURN CARRIAGE is output to the teletype keyboard. If this message persists, the tape contains a bad record and cannot be processed.

f. When the buffer is full, the message TURN ON PUNCH AND RETURN CARRIAGE is given. The buffer is then punched.

g. Steps d, e, and f are repeated until an end record is encountered, at which time the partial buffer is punched and the program terminates.

*The size of this buffer (PCH BUF) can be changed at assembly time without affecting the operation of the program.

6-3 OPERATING INSTRUCTION

6-3.1 INPUTS/OUTPUTS. The input records, as defined in paragraph 6-2.4, are processed by LRL. During the first pass the identification records are used to build the segment name table (IDTAB) and the linkage data records are used to build the symbol table (SYMTAB). Examples of these inputs and their relationship to the Linking Loader system are shown in figure 6-2.

LRL processes text records during the second pass. Each processed text record is examined for any linkage data present. If there is no linkage data present in the card, the text record is output, or loaded as it is. If there is linkage data present, the text record data is linked as defined by the linkage data and output or loaded.

In the second pass only the text records and end records are processed. In this pass errors such as undefined labels, double defined labels, a flag or bit reference used illegally, illegal external references, and truncation errors (a value is too large to mask into the described field) are printed as an option. The second pass also produces the binary object output file, consisting of an identification record, text records, and an end record. The linked program can

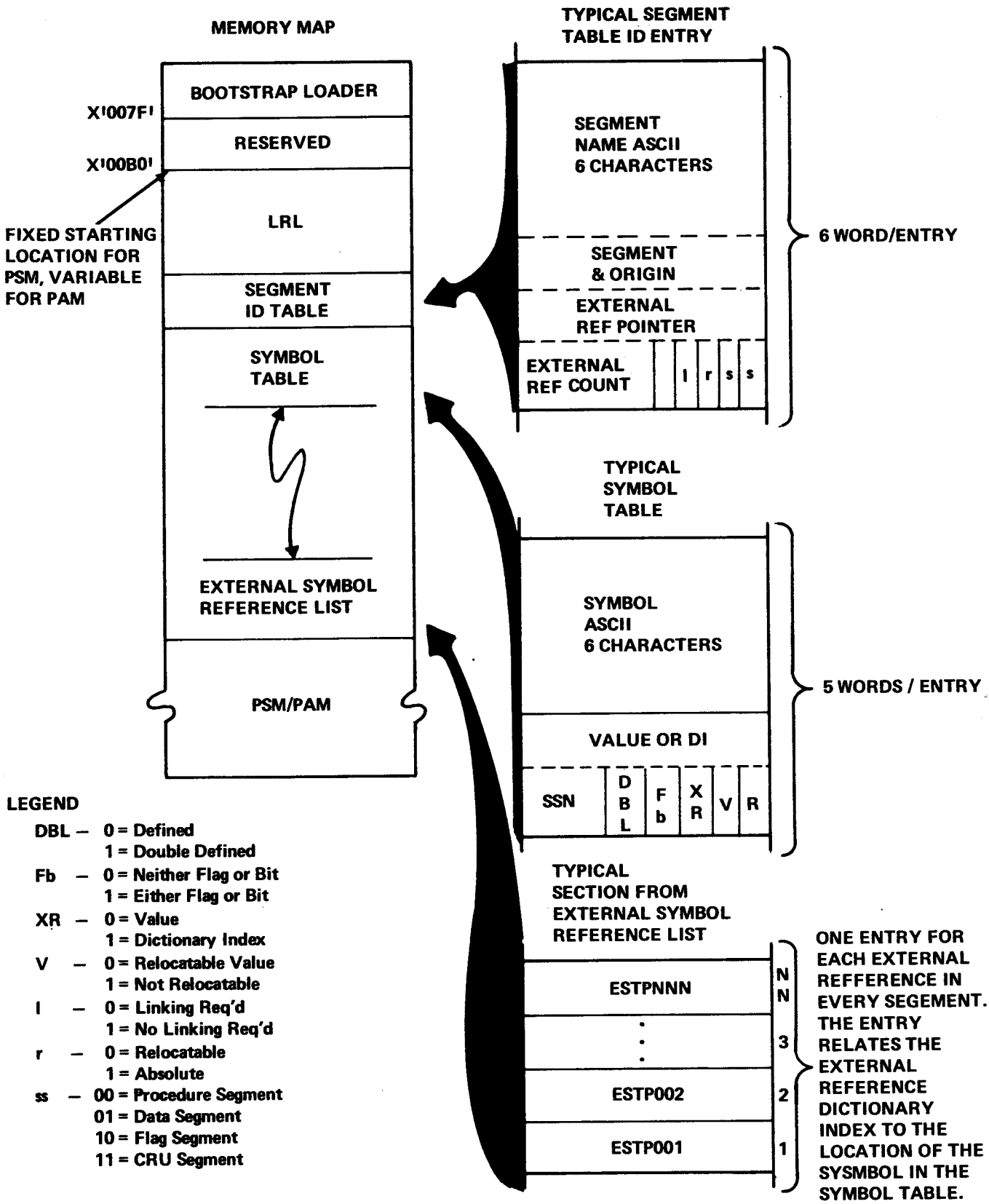


Figure 6-2 Memory Map.

optionally be loaded into any available space in memory.

When all text and end records are processed the Linking Loader encounters an END OF FILE (EOF) record and, after outputing the END and EOF record, the Linking Loader returns to the monitor system.

6-3.2 CONTROL FEATURES. LRL runs under PAM. Refer to Section III for help in using PAM Job Control to set up LRL jobs. LRL also runs under PSM. Refer to Section IV to find help in making PSM I/O assignments for LRL. The Linking Loader requires one option card to define what processes are necessary. LRL (LUNO 30) may be assigned to either the card reader or teletypewriter keyboard. The options are read after the messages PASS 1, LOAD PRG; or PASS 2; LOAD PRG? are output. After all passes are complete and an EOF card has been read, the program returns to the monitor system.

Error stops in the Linking Loader occur if the symbol table overflows; if the Linking Loader is trying to load a program that is out of limits; or if a redundancy character error is detected. Symbol table overflow is a non-recoverable error. The Linking Loader prints SYMBOL TABLE FULL and terminates the job. Loading out of limits is not a fatal error. The Linking Loader prints LOAD OUT OF LIMITS and turns off the loading process but continues with the remaining options.

If a redundancy character error is detected, REDUNDANCY ERROR, REPOSITION LAST RECORD ? will be printed. Error recovery requires the operator to position the last record read. He must type on the keyboard the letter Y and a carriage return if the desires to try again or the letter N and a carriage return if he desires to abort.

6-3.3 RESTRICTIONS. LRL input restrictions are:

a. The programs to be linked must be assembled according to Model 960 Reference Manual procedures.

b. There must be no change in the sequence of segments loaded for PASS 1 or PASS 2.

In assembly modules containing more than one segment the REFerence declearations for all externally referenced symbols in the entire assembly should be located in the first segment. This means that all REF statements are in the first segment of an assembly module.

6-3.4 LOADING PROCEDURES. LRL is loaded by Job Control under Pam or by the bootstrap loader under PSM. The PAM control cards needed to load and execute LRL are described in Figure 6-3. The PAM types OP ? on the teletypewriter when loaded. The correct answer to this is JCON and a carriage return.

6-3.5 OPERATING PROCEDURES. PAM or PSM must be loaded unless the monitor is already resident in core. LRL and its PAM job control are loaded via the card reader or the paper tape reader. Using PSM the command !L! from the teletypewriter activates the loading process.

Using PAM option card for LRL should be directly in front of the input binary file if options are to be read from a card. If options are to be entered in through the teletypewriter keyboard, the program segments to be linked must appear following the $$SEXCT** card. The option card format or teletypewriter entry is discussed in paragraph 6-3.4.

The LRL program prints:

PASS 1

LOAD PRG?

At this time the options are read and PASS 1 processing continues.

During PASS 1 each record in the binary input file is checked for a valid redundancy character. If an error is detected,

REDUNDANCY ERROR

REPOSITION LAST RECORD ?

is printed on the control message device, normally the teletype printer. Response to this is to reposition the last record read, ready the card reader, and enter on the keyboard:

Y cr try again.

N cr to abort.

LRL LOGICAL DEVICE ASSIGNMENT

NUMBER	USE	TYPICAL DEVICE TO USE
0010	Binary Input File	Card Reader, Punched Tape Reader
0020	Binary Output File	Card Punch, Paper Tape Punch
0030	Option Input	Card Reader, Data Terminal, TTY
0040	Messages -- LOG	Data Terminal, TTY Line Printer
0050	Load Map	Data Terminal, Line Printer, TTY
0060	Operator Input	Data Terminal Keyboard, TTY

EXAMPLE LRL LOG OUTPUT USING PSM

```
LX                                        Load, Execute
OP?   DFIO   0010   0002                  'WFDIO'
OP?   DFIO   0020   0003
OP?   DFIO   0030   0002                    IO Device
OP?   DFIO   0040   0000                    Assignment
OP?   DFIO   0050   0004
OP?   DFIO   0060   0000
OP?   ↑
LX                                        Load, Execute LRL

PASS 1                                      LRL Messages
LOAD PRG ?

PASS 2
LOAD PRG ?
X
```

OPTIONS FOR LINKING RELOCATING LOADER

```
CHARACTER NO.    1 2 3 4 5 6 7 8 9 0 _____ 8 0
```

/ *NN* *LLLL

Where NN is as follows:

ØA	Error Listing Only
ØB	Load Map Only
ØC	Load Map And Error Listing
ØD	Binary Output Only
ØE	Binary Output and Error Listing
ØF	Binary Output and Load Map
ØG	All Options
ØØ	No Options
L	in column 3 activates Load Option
LLLL	in the Load Bias in Hexadecimal

There must be an option card present as the first card of the binary deck (s) to be linked in PASS 1 and PASS 2 if card 3 is used to define Device 30 to the card reader.

NOTE

Binary decks should not be re-arranged after the linking process has begun.

When the notification on the teletypewriter that PASS 2 of the LRL has begun:

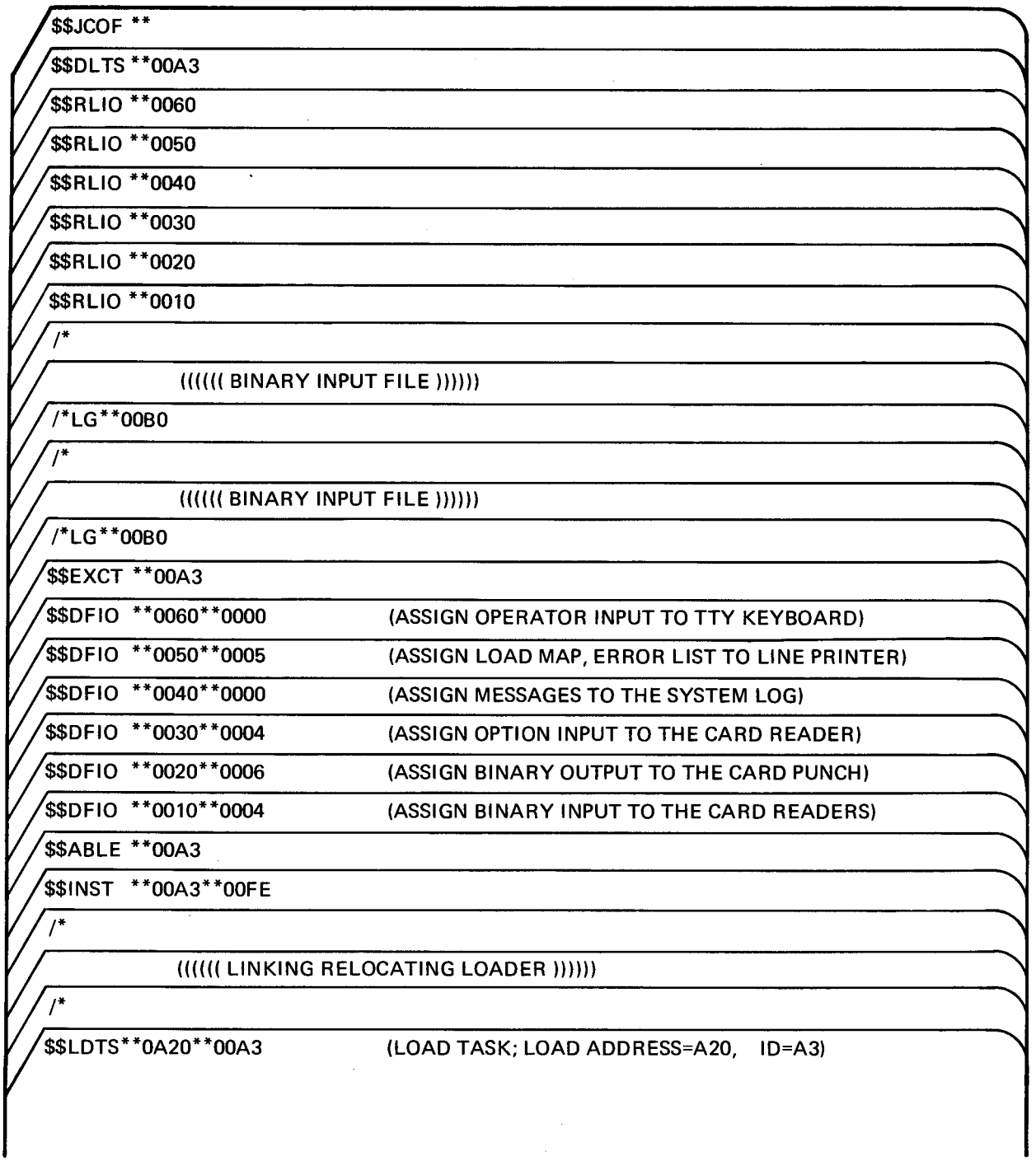


Figure 6-3 Typical LRL Job Setup Using PAM.

a. Card 12, the object programs to be linked, and Card 13 must be reloaded into the high speed card reader and the start button pushed. The LRL program begins processing the PASS 2 options and terminates after reading card 13.

If data read from the Linkage Data Record overflows the Symbol table during PASS 1 LRL prints.

SYMBOL TABLE FULL and terminates. This requires a change to the programs to be Linked (decrease number of symbols) or the symbol table of the Linking Relocating Loader must be lengthened. When all data has been read by the LRL and an EOF card (/*) has been recognized, the Linking Relocating Loader terminates PASS 1 processing.

If the option to print a load map is in effect, the LRL program prints the Local Map.

LOAD MAP (1)

INDENT TYPE RELOCATION CONSTANT (2)

EXTERNAL REFERENCES (3)

DUM001 – PSEG – 0000 (4)

SYM001 0001U SYM002 0002D SYM003 0000 (5)

DUM002 – DSEG – 0100 (6)

DUM003 – APSG – 0000 (7)

Lines 1 through 3 are header information. Line 4 identifies program segment DUM001 with an origin of 0000. Line 5 lists the external references for this segment. SYM001 0001U is an undefined reference and has a dictionary index of 1. SYM002 0002D is a double defined reference and retains the last value defined, 0002. SYM003 0000 is a defined symbol and has the value 0000.

Line 6 identifies the data segment DUM002 with the origin set at 0100. Line 7 identifies the absolute program segment DUM003 with an origin of 0000.

Five external references per line may be printed. One hundred program segments may be listed.

When the load map is completed, the LRL begins the second pass operations

PASS 2

LOAD PRG ?

is printed. The same option card loading procedure (or type in), linkable object deck, and EOF card is repeated for the second pass.

Three options are available during second pass processing. The first option is to list errors found while linking the text records. The errors format is DUM001 – PSEG – 0000.

The message UNDEF SYM001 0010 means that SYM001, an undefined symbol, was linked at relative location 0010 to Program segment DUM001.

The message DBLDEF SYM002 0011 means that SYM002, a double defined symbol was linked at relative location 0011 with the last defined value assigned.

The message TRUNC SYM002 0013 means that the symbol value for SYM002 was too large to attempt linkage in the instruction field at relative location 0013.

The message IL FLG OR BIT 0015 means that an illegal external reference for a FLAG or BIT SYMBOL was attempted at relative location 0015.

The second option is to generate the linked program object code. The punched records include an identification record, text records, end record, and an EOF record.

The third option is to load the linked program while the linking process is generating new binary records.

If the program is to be loaded and happens to require more than the available memory, LOAD OUT OF LIMITS is printed where the LRL program terminates the load option and continues with the remaining options.

The second pass terminates when the EOF card is recognized. The LRL returns to the monitor.