Volume 4
System Utility Programs

Programmer Reference

UNIVAC
COMPUTER SYSTEMS

This document contains the latest information available at the time of publication. However, Sperry Univac reserves the right to modify or revise its contents. To ensure that you have the most recent information, contact your local Sperry Univac representative.

Sperry Univac is a division of Sperry Rand Corporation.

AccuScan, FASTRAND, PAGEWRITER, SPERRY UNIVAC, UNISCOPE, UNISERVO, UNIVAC, and ✧ are trademarks of the Sperry Rand Corporation.

This manual corresponds to CULL level 3R2, DOC level 4R1, FLAP level 4R1A, LIST level 3R1, PIRCB$ level 1.0, and SYSLIB level 73R1 of the 1100 Series Executive System.

PRINTED IN U.S.A.

PAGE STATUS SUMMARY

ISSUE: UP-4144.41

Section	Pages	Update	Section	Pages	Update	Section	Pages	Update
Cover/Disclaimer								
PSS	1							
Preface	1							
Contents	1 thru 5							
1	1 thru 2							
2	1 thru 110							
3	1							
4	1 thru 4							
5	1 thru 17							
6	1 thru 12							
7	1 thru 4							
User Comment Sheet								
Total: 159 pages and cover								

Preface

The SPERRY UNIVAC 1100 Series Executive System Programmer Reference manual is divided into four volumes. These volumes are titled as follows:

- SPERRY UNIVAC 1100 Series Executive System, Volume 1, Index, UP-4144.1

  Volume 1 references terms and subjects covered in the other volumes. The references are by volume number dash paragraph number, e.g., paragraph 3.7.4 of Volume 2 will be 2-3.7.4.

- SPERRY UNIVAC 1100 Series Executive System, Volume 2, EXEC Programmer Reference, UP-4144.2

  Volume 2 describes the overall control of SPERRY UNIVAC 1100 Series Systems by the Executive system.

- SPERRY UNIVAC 1100 Series Executive System, Volume 3, System Processors Programmer Reference, UP-4144.3

  Volume 3 describes the basic system processors.

- SPERRY UNIVAC 1100 Series Executive System, Volume 4, System Utility Programs Programmer Reference, UP-4144.41

  Volume 4 describes the system relocatable library and utility processors.

Cross references to subjects in other volumes are by volume number dash paragraph number, e.g., 2-3.7.4 is Volume 2, paragraph 3.7.4.

Contents

1. Introduction

This volume describes the routines included in the System Relocatable Library (SYSLIB). Also described are certain utility processors that are provided for the convenience of the user that are not essential to use the SPERRY UNIVAC 1100 Series Executive System. These processors are CULL, DOC, FLAP and LIST.

SYSLIB is contained in the system relocatable library file SYS$*RLIB$ along with other relocatable libraries. When these relocatable subroutines are referenced by user programs they are automatically included in the absolute program constructed by the Collector.

SYSLIB is composed of relocatable and procedure elements that can be grouped into six functional areas. These areas are listed below.

- System Standard Procedures and Definitions

AXR$	CBEPMDP$	CERU$	SSTYP$
CBEPFORV$	CBEPPIR$	ERU$	
CBEPMATH$	CBEPP2IO$	PROC$	

- Collector Interface Routines

DLOAD$	IDLD$
IDL$	IDLAD$
IDLA$	SNAP$

- Diagnostic and Debugging Routines

CABSAD$	XCREG$	XLGIC$
CRELAD$	XCW$	XMARK$
SNOOPY	XDRUM$	XMESG$
XBUFR$	XDUMP$	XSIZE$
XCOMN$	XFILE$	XTALY$
XCORE$	XFRMT$	XTAPE$

- Editing Routines

AEDIT$	EDIT$	EDIT$T
AEDIT$F	EDIT$F	EOUT$
AEDIT$T	EDIT$P	

- Processor Interface Routines (PIRs)

BSP$	INFOR$	PREPRO
GETPSF$	POSTPR$	ROR
IDLINE$	PREPF$	SIR$
IDONLY$	PREPRM	SOR

- Utility Routines

FDASC$	SDFI
HQUCNV	SDFO
HQUCRTS	TABLE$
MFDSP$	

The Collector interface and dynamic dump routines are described in Volume 3. The remaining routines are described in this volume.

2. System Relocatable Library (SYSLIB)

2.1. System Standard Procedures and Definitions

2.1.1. AXR$

AXR$ is a system procedure that contains the numeric definitions of the standard mnemonic designators for control registers, partial word designators, etc., which are used in assembly language coding.

Table 2-1 lists the mnemonic designators and absolute addresses of control registers and partial word designators.

2.1.2. Common Bank Entry Points

2.1.2.1. CBEPFORV$

CBEPFORV$ is the entry point to FORTRAN V I/O common bank and FORTRAN V/NUALG error common bank.

2.1.2.2. CBEPMATH$

CBEPMATH$ is the entry point to the 1100 Series Mathematical Library common bank.

2.1.2.3. CBEPMDP$

CBEPMDP$ is the entry point to the MACRO processor common bank.

2.1.2.4. CBEPPIR$

CBEPPIR$ is the entry point to the processor interface routine common bank.

Table 2-1. Control Register and Partial Word Designator Mnemonics and Addresses

Index Registers		Arithmetic Registers		R Registers		Staging Registers J Registers		Partial Word Designators	
Mnemonic	Absolute (Octal)	Mnemonic	Absolute (Octal)	Mnemonic	Absolute (Octal)	Mnemonic	Absolute (Octal)	Mnemonic For j	j-Value (Octal)
X0	0	A0	14	R1	101	SR1	103	W (or none)	0
X1	1	A1	15	R2	102	SR2	104	H2	1
X2	2	A2	16	R3	103	SR3	105	H1	2
X3	3	A3	17	R4	104	J0	106	XH2	3
X4	4	A4	20	R5	105	J1	107	XH1	4
X5	5	A5	21	R6	106	J2	110	T3	5
X6	6	A6	22	R7	107	J3	111	T2	6
X7	7	A7	23	R8	110			T1	7
X8	10	A8	24	R9	111			S1	15
X9	11	A9	25	R10	112			S2	14
X10	12	A10	26	R11	113			S3	13
X11	13	A11	27	R12	114			S4	12
		A12	30	R13	115			S5	11
		A13	31	R14	116			S6	10
		A14	32	R15	117			Q1	7
		A15	33					Q2	4
								Q3	6
								Q4	5
								U	16
								XU	17

2.1.2.5. CBEPP2IO$

CBEPP2IO$ is the entry point to ASCII FORTRAN I/O, ANSI sequential tape I/O, SDF sequential I/O, SDF direct I/O, common I/O error, PLUS/PLUS common I/O, ISAM and multi-keyed indexed I/O common banks.

2.1.3. CERU$

The relocatable element CERU$ contains the names of all system common banks and their associated BDI values.

2.1.4. ERU$

The relocatable element ERU$ consists of externally defined labels and generates no code in a user program. It consists of:

1. Mnemonic names of the Executive Requests (PRINT$, IOW$, et al.) which are equated to the ER indexes needed by the EXEC (see Volume 2-Section 4).

2. The mnemonic names for I/O functions (W$, WEF$, R$, et al.) which are equated to the value shown in Volume 2-Table 6-1.

3. A label RLIB$ which is equated to the Fieldata representation of the current level of SYS$*RLIB$.

4. The mnemonic names for RSI$ functions (RSBAT$, RSGET$, RSTRY$, etc.) which are equated to the values shown in Volume 2-8.6.

5. The mnemonic names for MSCON$ functions (DREAD$, DGET$, etc.).

6. The mnemonic names for INFO$ functions (see Volume 2-4.8.7.).

7. The following externalized labels:

   PCTBD$ - BDI value for user PCT bank.

   RPCTA$ - User entry point to PCT bank.

   CYCLIM$ - System standard symbolic element cycle maximum.

ERU$ is automatically included in a collection if any of the above symbols are referenced in an assembler program.

2.1.5. PROC$

PROC$ is a collection of Assembler Procedures as follows:

1. Procedures which generate calling sequences and packets for symbiont calls (see Volume 2-Section 5), I/O calls (see Volume 2-Section 6) and various other Executive Requests.

2. Procedures for EOUT$ editing routine calls (see 2.4.4).

3. Procedures which generate calls to the dynamic dump routines (X$DUMP, X$DRUM, et al.) (see Volume 3-Section 3).

4. A procedure which generates NTAB$ tables for FORTRAN V. (See SPERRY UNIVAC 1100 Series FORTRAN V Programmer Reference, UP-4060 (current version).)

5. The ELT$ PROC defines the fields of the file and element notation table (ELT$), prepared by the INFOR routine SELT$.

6. Procedures (ASCII and Fieldata) used with the assembler to change the radix for generation of character data.

7. A$SCTRL is a procedure which defines the mnemonics for the ASCII control characters corresponding to octal codes 0-37 and 177. These ASCII control characters are defined in Volume 2-Appendix D.2.1.

8. JREG function to aid in loading J registers (see UNIVAC 1100 Series Assembler (Fieldata) Programmer Reference UP-4040 (current version)).

9. SN$DEF PROC which defines SNOOPY flags (see 2.3.1).

2.1.6. Element Subtype Definitions (SSTYP$)

A non-executable element is available from the library which contains a table of the standard symbolic and omnibus element subtypes (processor codes). It has a single external definition, SSTYP$, which is the address of a word containing the length of the table which begins at SSTYP$+1. If the subtype number exceeds the contents of SSTYP$, no table entry exists, and the value must be edited in octal. Otherwise, the subtype number is used as an index relative to SSTYP$+1. Entries in the table are three characters per word, Fieldata LJSF (left-justified space-filled).

The subtype codes are contained in the element table items in the program file (See Volume 3-11.2.1.1).

The omnibus and symbolic element subtypes are defined in Table 2-2.

Table 2-2. Omnibus and Symbolic Element Subtypes

Mnemonic	Code	Processor Language
SYM	00	Symbolic (No subtype)
ELT	01	ELT
ASM	02	Assembler
COB	03	COBOL
FOR	04	FORTRAN
ALG	05	ALGOL
MAP	06	Collector
DOC	07	Document Processor
SEC	010	SECURE Processor
SSG	011	Symbolic Stream Generator
APL	012	A Programming Language
BAS	013	BASIC
LSP	014	LISP
PLS	015	PLUS
PL1	016	PL/I
CTS	017	Conversational Time Sharing
FLT	020	FLIT Processor
PNC	021	Panic
TCL	022	Traffic Control Language
MSM	023	Meta-Assembler (MASM)
MSD	024	MASM Definition Language
MAC	025	MACRO
APT	026	Automatic Program Tools

2.2. Collector Interface Routines

The Collector interface routines are described in Volume 3, Section 2.

2.3. Diagnostic and Debugging Routines

The dynamic dump routines are described in Volume 3, Section 3.

2.3.1. Program Trace Routine (SNOOPY)

SNOOPY is a program trace routine which is designed for use primarily with assembly-language programs. In batch mode, SNOOPY provides a straightforward accour.t of every instruction executed and its effect. In the demand mode, SNOOPY acts as a powerful diagnostic routine affording user control over the trace operation.

Two formats are available for calling SNOOPY:

```
SLJ     SNOOPY$ (or SNOOPY with pre-SYSLIB 73R1 releases)
+       mode-bits,termination-addr  .mode-word
```

and

```
SLJ     TON$
```

When the first format is employed, tracing begins with the instruction following the mode-word. Tracing continues until the termination address (termination-addr) is reached or until another termination condition is encountered.

If bit 18 of the mode word is set, quarter-word mode is simulated by SNOOPY for checkout of quarter-word sensitive programs on machines without quarter-word hardware. Otherwise, SNOOPY uses either third- or quarter- word mode depending on the mode set in the PSR on entry. Bit 19 of the mode word may be set to suppress the solicition of commands at the beginning and end of a trace when SNOOPY is used in demand mode.

When the second format is employed, tracing begins following the SLJ instruction and continues until a termination condition is encountered; quarter-word or third-word mode is determined by the mode set on entry.

When operating in the batch mode, tracing may be terminated by:

1. Reaching the specified termination address (program execution continues).

2. Executing an SLJ TOFF$ instruction (program execution continues). If an SLJ TOFF$ instruction is executed outside of the trace routine, it has no effect.

3. Performing an ER EXIT$ (see Volume 2-4.3.2.1.). This not only terminates SNOOPY but it also terminates the activity being traced.

4. Encountering a program contingency of types 1_8, 2_8, 7_8, or 12_8 for which standard system action has been specified (see Volume 2-4.9.). The activity being traced is terminated by EXIT$.

When operating in the demand mode, tracing may be terminated by the following methods in addition to those available for batch mode:

1. Using the TOFF$ command (see Table 2-3); program execution continues.

2. Using the EXIT$ command (see Table 2-3). This not only terminates SNOOPY but it also terminates the activity being traced.

The following restrictions apply to SNOOPY:

1. SNOOPY must be part of the program's main segment (see Volume 3-2.2.2.14.).
2. Tracing terminates if the main segment is reloaded.
3. Only one activity may be traced at any one time unless duplicate copies of SNOOPY are used.
4. The program cannot be running in real-time mode.
5. No activity contingency routine may exist for the activity being traced.
6. The only I/O Executive Requests permitted are IO$ and IOW$.
7. Reentrant processor Executive Requests will terminate the trace.
8. Byte instructions and J-register (character addressing) mode are not supported.

A program being traced functions the same as an untraced program except that:

1. Execution time is slower, which may affect I/O timing if that is relevant to the program being diagnosed.
2. Contingencies that would normally cause the activity being traced to teminate in error will result in EXIT$ termination.

When using SNOOPY, the u-field of the instruction is edited in octal if it does not refer to control storage. It is possible to provide a symbol table to SNOOPY giving Fieldata characters instead of octal values when the u-field falls within a certain range. This is done by storing a value into the externally-defined location SYMTB$. This may be done at any time, even while a program is being traced. The value stored in SYMTB$ has the following form:

H1	H2
nbr-of-entries	*table-addr*

The symbol table at the specified address is an array of three-word entries in the form:

	H1	H2
0	*high+1*	*low*
1	*Fieldata symbol name or address of long name*	
2	*bits*	*symbol address*

If the lowest bit of *bits* (bit 0) is set, word 1 of the entry is not taken to be a one-to-six-character Fieldata symbol, but is instead assumed to be the address of a seven-to-twelve-character Fieldata symbol. This allows accommodation of long labels.

If, for the last entry in the symbol table, bit 1 of *bits* is set, the next word after the last entry is assumed to be another symbol table descriptor word, in a format identical to that of SYMTB$. The symbol table to which it points will then be searched in the same fashion. This may be repeated to any depth desired; however, circularity must be avoided.

If the u-field of an instruction satisfies low ≤ u ≤ high, the u-field is edited as the symbol given by the second word of the entry, plus or minus an offset calculated from the symbol address in the third word. If the *nbr-of-entries* parameter of STMTB$ is zero, as it is initially, all u-fields outside control storage are edited in octal, except TOFF$ and a number of ER indices.

For batch users, operation is as just described. Users are cautioned that large quantities of printout may be produced when using SNOOPY.

Demand users are given a great deal of control over the behavior of SNOOPY. When entered, SNOOPY examines the program control table (PCT) to compute storage limits and the contingency routine address. At this time, the program type is checked; if it is demand, conversational control facilities are enabled.

When tracing a demand program, the amount of output produced by SNOOPY is reduced because of the low speed devices used for output. In particular, header and trailer messages are brief, registers are dumped only on request, and line length may be restricted. Further control over printing may be obtained through the use of commands as described in Table 2-3.

For demand programs, SNOOPY operates in two modes: trace mode, in which instructions are being traced, and command mode, in which commands are accepted that direct SNOOPY's operation. SNOOPY enters command mode under these circumstances:

1. On entrance, before tracing any instructions, except if mode bit 19 was set.

2. When the RBK contingency occurs (@@X C).

3. On completion of the number of instructions specified by a SKIP or a numeric command.

4. When a condition specified by the BREAK or TRAP command is encountered.

5. At trace termination, except if mode bit 19 was set.

In command mode, commands are solicited by the typeout "C—". Parameter fields required by the commands are solicited by a typeout indicating the nature of the parameter required. More than one command may be given on any line solicited in command mode (including parameter lines). The commands and parameters are separated by delimiters, where a delimiter is any character not in the set (A-Z, 0-9, $, or -). Excess blanks are ignored. If any other consecutive delimiters are encountered, the effect is the same as the STEP command; that is, if "C—" is answered with a line consisting of "////", it has the same effect as four STEP commands. In certain cases, specific delimiters are required. To omit a parameter entirely, the delimiter which terminated the command must be followed by another (nonblank) delimiter (for example: "PRINT /"). Trace mode is suspended, the last instruction is printed, and a soliciting message is made under the above five circumstances.

In each of the following circumstances, all commands on the line after the last one performed are ignored and can be reentered (if so desired):

1. trace termination (unless mode bit 9 was set).

2. the @@X C sequence is used to interrupt the trace.

3. an invalid command is encountered.

4. the CHANGE, TOFF$, or EXIT$ command is used, without a trailing asterisk.

A blank line (carriage return) in reply to the "C—" typeout has the same effect as the STEP command; a blank line repsonse to a parameter request may be erroneous or may have a special meaning depending on the nature of the command.

A command solicitation typeout may be answered by entering a line beginning with the character %. The remainder of the line should be an Executive control statement, exclusive of the initial @ character. The statement must be one of those legal for submission to the CSF$ Executive Request function (Volume 2-4.10.1.1.). SNOOPY will convert the initial % character to a @ and submit the resulting image to CSF$. If the requiest is processed normally, the status bits from A0 are displayed, and SNOOPY solicits the next command. Otherwise, the appropriate contingency is intercepted and the message "CSF$ ERROR" is printed, followed by command solicitation. For example, to assign a temporary file named T35, the user may respond to "C—" by typing "%ASG,T T35,F2".

When command mode is entered at trace termination, the trace may be completed only through use of the GO or STEP commands or a command with an equivalent effect. Once such a command is given, no further commands are executed, and the trace terminates. The activity then continues execution or exits, depending on the type of trace termination.

In all cases where number is called for, octal notation is assumed, unless otherwise indicated, a leading zero is not required. In general, SNOOPY uses octal notation everywhere except in register designations and in the instruction cycle count printed by SNOOPY.

Certain commands (TOFF$,EXIT$,CHANGE) clear SNOOPY's command buffer before reading further commands because of the potentially irreversible nature of the operation to be performed. If this is not desired, an asterisk may be affixed to the command (e.g. TOFF*). For example, to terminate a trace and continue an execution without typing in two lines, the sequence "TOFF$*GO" may be used.

All commands listed may be abbreviated to the first three characters; all commands except ALTPRT, TON$. RBK,RLIB, and STEP may further be abbreviated to the first character only.

In some cases particularly for complex programs, the set of commands provided by SNOOPY may not meet all the debugging requirements of the programmer. For this reason, a method is provided to allow a programmer to extend the command interpreter as his needs dicate. This is done by allowing SNOOPY to link to a user-supplied subroutine when a command is encountered which is not contained in the set of standard commands. (Because one-character abbreviations are permitted for most SNOOPY commands, a user command may not begin with the same letter as a SNOOPY command. Therefore, it is strongly suggested that user commands begin with the $ character.) A user command routine is identified to SNOOPY by storing its address in H2 of the externally defined cell SNUCM$. The user routine must return to 0,X11 if the command cannot be recognized; normal return is to 1,X11. The volatile register set may be used without saving and restoring it. On entry to the user command routine, A1-A2 contains the command in Fieldata, left-justified, space filled; X2 contains the current program address (simulated P-register); and X10 points to an area containing the program's simulated control registers, located offset corresponding to their absolute addresses (e.g., A0 contents at 014,X10, R5 contents at 0105,X10, and so on). The value in SNUCM$ may

be changed at any time by the user program, even while a trace is being performed. If it is set to zero, no user commands will be interpreted.

A user command routine may reference SNOOPY's command reading routine and lexical scanner by performing an SLJ to SNGTC$. A0 should contain 0 or a pointer to a TREAD$ packet before calling SNGTC$. If A0 ≠ 0, no input will be read; the remainder of the line containing the user command will be scanned for information. If A0 contains a TREAD$ packet pointer, the input buffer specified must be SNBUF$. On return from SNGTC$, A0 is unchanged, A1–A2 contains a Fieldata command, left-justified and space filled (unless A3 = 0), A3 contains the nonblank character count for A1–A2, and R1 contains the terminated delimiter. If an entirely blank line is encountered, A3 = 0 and R1 = '/'. SNOOPY commands conform to element name syntax (alphabetics, numerics, '–', and '$').

In some cases, it is not desirable to use SNOOPY to trace an entire program. Instead, one would like to have SNOOPY gain control only when a contingency occurs. A routine is available to provide this capability, it is called as follows:

```
SLJ     SNCNT$
```

Following this calling sequence, the program proceeds normally until a contingency is encountered. When a contingency occurs, SNOOPY will take control; the simulated P register will point to the contingency address + 1. A message indicating the nature of the error will be printed. In batch mode, if the contingency is fatal, SNOOPY will EXIT$. For nonfatal contingencies, the remainder of the program will be executed. In demand mode, the user is given control in command mode so that the program environment may be examined. No distinction is made between fatal and nonfatal contingencies in demand mode. Note that it is the user's responsibility to clear registers in case of arithmetic fault and to respond to the onsite operator in case of an II keyin. For IGDM, note also that the trapped address may or may not be the address of the offending instruction. The contingency interception mode may be reset by the call:

```
SLJ     SNCLR$
```

In frequently executed code, it is not always desirable to trace every iteration of a loop or every call of a subroutine; in particular, a bug may not appear until a routine has been executed some number of times. The alarm-clock mechanism provides a way to call SNOOPY selectively. The calling sequence is:

```
SLJ     TCNT$
+       count,start
+       until,every
+       mode-bits,termination-address
```

where mode-bits and termination-address are the same as for an SLJ to SNOOPY. The count field should be zero initially and will be incremented on each pass through the code. If start ≤ count < until and count – start = 0 (modulo every), SNOOPY will begin tracing following the mode word. Otherwise, execution will continue in normal (untraced) mode.

Table 2–3 lists the SNOOPY commands, their permissible abbreviations, and their functions.

Table 2-3. SNOOPY Demand Mode Commands

Command	Description
ABSAD ABS A-	Convert relative program addresses to absolute addresses. The parameters are: eltname,loc-counter,location. or symbol* If the program is segmented and the specified address is in a segment not in main storage, a message is printed. In case of symbols, only reference symbols defined by non-RLIB$ routines may be looked up unless the collection source language included a "TYPE EXTDIAG" statement, in which case all symbols may be looked up (see Volume 3-2.2.2.13). This command is useful in conjunction with the BREAK, CAPTUR, CHANGE, DUMP, JUMP, and TRAP commands.
ALTPRT ALT	Send all trace printout to an alternate print file, while command solicitation, command responses (as by DUMP, for example) and all print requests by the program are sent to the terminal as usual. One parameter may be given, which is the name of the file to be used as an alternate print file. If the file specified is not assigned, SNOOPY will automatically assign a temporary file. (If the file is to be printed by the @SYM statement, the user must assign it himself as catalogued rather than temporary.) The user must observe the restrictions on the maximum number of alternate files allowed to be open if the program being traced also uses alternate symbiont files. The ALTPRT command gives the user control over the disposition of his alternate print files. In general, when the ALTPRT command isgiven and an alternate file is in use, the current alternate file will be @BRKPTed. This action may be suppressed by employing an asterisk (*) as the trailing delimiter for the parameter; e.g., either "ALTPRT<filename>*" or "ALTPRT *". To obtain printout both at a terminal and in an alternate file, the ALTPRT filename should have a trailing exclamation point (!); e.g., "ALTPRT<filename>!". This is useful for obtaining a permanent record when working at a terminal which lacks a hard-copy capability. If an alternate file is already active, the command forms "ALTPRT !" and "ALTPRT ?" may be used to set and clear echo mode while leaving the same alternate file in use. Echo mode is always cleared when the trailing exclamation point is not used; therefore, to start a new alternate file without breakpointing the old file and with echo mode set, the command sequence "ALTPRT* ALTPRT<filename>!" must be used. If the parameter is omitted (as in "ALTPRT/"), printing will be directed to the terminal, as it is initially. This command is effective for all SNOOPY operations until another ALTPRT command is given or the program being traced terminates. This command forces the location counter-element name printout to be given, so that each piece of printout may be easily identified.
BREAK n1,n2... BREAK+ n1,n2... BREAK?	This command indicates a stopping point for SNOOPY when operating in trace mode. When any of the specified addresses is reached, the trace will stop and SNOOPY will enter command mode. This command is useful for running with printing turned off. The form BREAK+ is used to add entries to the existing table. A maximum of 16 addresses is permitted. The $ specification may be used to retrieve the last address returned by the ABSAD command. The form BREAK? is used to list the table of break addresses. The command BREAK may be abbreviated to just the letter B.

Table 2-3. SNOOPY Demand Mode Commands (continued)

Command	Description
	NOTE: This form of the break command is INCOMPATIBLE with the previous form. The change is being made for reasons of efficiency and utility; the break on element name and/or location counter was seldom used but introduced a considerable overhead for each instruction traced, while it was impossible to have more than one break active at the same time. The user may determine which form of the BREAK command is available in the SNOOPY in use by entering the BREAK command without parameters on the same line (nor a "/" to indicate no parameters). The old form of BREAK will solicit input with the typeout "ELT NAME—", whereas the new form will request input with the typeout "LOCN—".
CAPTUR	Enter SNOOPY from ordinary program execution mode whenever control reaches any of a specified set of addresses. The instructions at the specified addresses will be overlaid by a jump to a SNOOPY internal table area, where an SLJ TON$ will be performed, followed by the overlaid instruction. The overlaid instruction may not be referenced as data by the program, but there are no other restrictions. The syntax of command is CAPTUR <loc-1>,<loc-2>,... Up to 8 locations may be specified, with commas being the mandatory separators. Any existing list is replaced by the new list (with the overlaid instructions restored). If CAPTUR+ is used, the specifications are added to the existing table, again up to a total limit of eight. CAPTUR? may be used to print out the contents of the table. When using this command with segmented or multi-banked programs, care should be taken that the specified capture locations are in core when they are to be changed. The purpose of this command is to provide the fastest possible means of getting past a portion of the program which is irrelevant to the debugging task. The use of external symbols is subject to the restrictions noted for the ABSAD command. See CHANGE command for use of the symbol $.
CHANGE C	Allows the user to change the contents of control registers or main storage. The single parameter gives the location to be changed. After reading the location parameter, the contents of the specified location is displayed as if the location parameter had been given to a DUMP command; then the new value is solicited. The new value is stored and the new contents displayed. A void new value results in no change to the indicated main storage element. If the parameter is a register name, a number, or a number or register name preceded by an H or Q, the new value is to be entered as a single octal number. The CHANGE command allows the use of mnemonics and external symbols for I-format (instruction format) changes, as well as octal values. The first item given to the "NEW VAL—" typeout may be an op-code mnemonic instead of an octal number for the f-field of the instruction. Abbreviated forms such as L, LN, ANM, etc., are not permitted, however, LX, LA, LR, LNA, ANMA and so on must be used. For some instructions, the op-code mnemonic specifies values for the j-field and perhaps the a-field as well. In such cases, the next value given to the CHANGE command will be an a-field or an x-field. Mnemonics may also be used for the j-designator values (W,H1,H2,XH1,XH2,T1, T2,T3,S1,S2,S3,S4,S5,S6,Q1,Q2,Q3,Q4,U and XU) and for standard X-, A-, and R-register names. If a register name is used in the a-field of an instruction, its value will be adjusted appropriately. Truncation errors are NOT detected. Fields of an instruction are always expected to be entered in the order f,j,a,x,hi, and u. Note that the h- and i-fields are combined. An external symbol may be used for any field except the f-field, subject to the same restrictions noted for the ABSAD command. For specification of an address to the CHANGE command, the $ and $P have the following special meaning: $P The P register of the traced program.

Table 2-3. SNOOPY Demand Mode Commands (continued)

Command	Description
	$P-n Where n is an octal number, used to compute a value offset negatively from $P. $ The last value returned by the ABSAD command, or the last address plus one CHANGEd or DUMPed. This allows one to continue a dump or change operation immediately following the last area referenced, or to change a cell at an address to be calculated by ABSAD without retyping the absolute address. $-n Where n is an octal number, used to compute a value offset negatively from $. NOTE: The form $P (but not $P-n, $ or $-n) may also be used with the RELAD command.
DUMP D	Display the program status. Each parameter must be separated by a comma. If no parameter or an empty parameter (that is, two consecutive commas) is given, all registers and the carry and overflow designators are dumped. The parameters are: A,X, or R Dumps the indicated group of registers. To dump the contents of a single register, use the register mnemonic or the octal address. T Dumps the carry and overflow designators. L Display current storage limits. B Display names of active banks (in order main-I, main-D, utility-I, utility-D, with commas indicating place for unbased PSR portions), will indicate write-inhibited banks by an asterisk (*). Banks whose names cannot be found in diagnostic tables will be printed in octal numbers. S Display names of active segments. E Last contingency and address, if any. N Instruction cycle count Any dump specification which references an address may be given a trailing + sign followed by an octal number N. A dump is then taken of the N consecutive storage locations following the original address, in the same format as the first dump, resulting in a dump of N+1 locations. If a specification number is neither the address of a register, nor within the storage limits of the program, an error message is displayed. See CHANGE command for use of symbols $, $P. The contents of the specified location is printed in octal. The letter I preceding a number or register designator produces an instruction-format dump. The letter H preceding a number or register designator produces a Fieldata-character dump.

Table 2-3. SNOOPY Demand Mode Commands (continued)

Command	Description
	The letter Q preceding a number or register designator produces an ASCII-character dump.
EXIT$ E	Terminates the traced activity by means of an EXIT$ request. Trace mode is terminated and the last instruction is printed.
GO G	Return to trace mode command mode.
HDG	This command allows the insertion of print control information into SNOOPY's output, whether directed to a standard symbiont file or to an alternate file. Unless given as "HDG*", the command buffer will be flushed, and all other commands on the line will be ignored. In any case, the input for this command will be solicited by the printout "HDG- -". Only print control information may go on this line; no commands are permitted. The format is identical to that used for the PRTCN$/PRTCA$ ERs (see Volume 2-5.4). In particular, for heading information, the form is "H,opt,ps,txt", but any of the other print control functions may be used as well.
JHT	Print the jump history table, starting with the most recent jump-from address and continuing with the last eight jump instructions which causes a transfer of control. The table is cleared on entry to SNOOPY, so fewer than eight addresses may be printed early in a trace. A jump which has been executed several times in succession without any other jumps intervening (i.e., a loop) will produce a listing with a repeat count, rather than many identical entries in the table. On 1110 and 1100/40 systems, if SNOOPY is activated by a contingency after a call on SNCNTS, the JHT command may be used to retrieve the hardware jump history stack as captured at the time of the contingency. This applies only to hardware contingencies (IOPR, IGDM, IFOF, IFUF, and IDOF).
JUMP J	Transfers control to an address specified as an octal number or externally defined symbol. The current absolute and relative P register values are displayed. If the new value is within the program storage limits, that value is set into the P register. The new value is printed in relative form and the next command is executed. This command is the only way to get out of EXIT$ mode and do further tracing by means of the TON$ command. If TON$ is used in the EXIT$ mode without a JUMP command, the TON$ command is rejected and a message is displayed. If a JUMP command is used in EXIT$ mode termination, the termination mode becomes a TOFF$ termination; a TON$ command is required if tracing is to continue. The jump-to address specified for the JUMP command may be an external symbol as well as an octal value, subject to the restrictions on use of external symbols noted for the ABSAD command.
LINE	Adjust the length of the line printed by SNOOPY. If "LINE /" is entered, the line length set is the default value of 132 decimal = 204 octal. Otherwise, the parameter given to LINE must be an octal number denoting the line length for the device in use. As indicated the default value is 132 decimal = 204 octal for devices such as DCT 500 Terminals. For a UNISCOPE 300 Display Terminal, "LINE 100" should be used, while for use with alternate print files (see the ALTPRT command), "LINE 204" may be most convenient. The effect of a LINE command is cancelled by the next LINE command or by program termination. The previous value will be printed as "WAS nnn". This value is also printed when SNOOPY signs on.

Table 2-3. SNOOPY Demand Mode Commands (continued)

Command	Description
number	Has the same effect as a SKIP n GO sequence (where n is the number). See SKIP command. The number is in octal.
PRINT P	Allows modification of the amount of printing. The PRINT command recognizes only one parameter at any one time. If an invalid parameter or no parameter is specified, the F parameter is assumed. The parameters are: C — Produce a printout omitting extraneous spaces used for formatting. E — Produce an expanded printout (formatting spaces are included). The E mode is effective until a PRINT C is encountered. F — Produce a full printout consisting of each instruction, its location, and the contents of main storage and registers (in before/after form if the value changed). For certain Executive Requests, the contents of the associated packet is also dumped. This is the default mode. N — Suppress printout. This provides a means of skipping long sections of irrelevant code. P — Produce printout of the instructions but not referenced main storage or Executive Requests packets. If SNOOPY is in the N mode, the P mode is set automatically upon the occurrence of an RBK contingency or encountering a BREAK specified break condition. I,n — The value of n is an exponent of 2 indicating the frequency with which the instruction cycle printout is given. The default is 10, the minimum is 6. The printout may be turned off by using a value of 35. M,msk — The value of msk is a mask which determines which instructions are to be edited while tracing. The most useful value for msk is 400, which will suppress all instructions except for jumps, skips, and ERs. Other values for msk may be determined by examination of the code for SNOOPY (edit descriptor bits).
RBK	Allows the user to simulate an RBK contingency for the executing program; the actual RBK contingency is intercepted by SNOOPY and directs a return to command mode. This command provides the means for tracing a contingency routine. If the user program does not expect the contingency, an appropriate message is displayed.
RELAD R	Convert absolute program addresses to relative addresses. A list of locations separated by commas may be specified as parameters. See $P specification for DUMP command. Ambiguities are resolved in favor of elements residing in loaded segments in active banks.
RLIB param	The parameter specified may be either an L, an E, an X, an A, or an N. Anything else will be assumed to be N.

Table 2-3. SNOOPY Demand Mode Commands (continued)

Command	Description
	"RLIB L" will print the system type, system level identification, and site as well as the RLIB$ level used at collection time. "RLIB E, <element-list>" where <element-list> is a list of element names separated by commas specifies that the elements named are to be treated as if they were RLIB$ elements for the purposes of RLIB$ trace suppression. The list specified completely replaces any preceding list. If a '+' sign follows the E, the elements given are added to the current list. A maximum of 16 names may be specified. An empty list is specified by following the "E" with punctuation other than a comma, a plus sign, or a question mark. "RLIB E?" prints the current list. "RLIB X,<element-list >" has syntax identical to that for "RLIB E", but the effect is inverted. That is, any element not contained in the RLIB X list will be treated as an RLIB$ element. RLIB X+ may be used to extend the list. "RLIB A" is used to allow tracing of RLIB$ routines, while "RLIB N" is used to suppress trace printout of RLIB$ code. When tracing of RLIB$ is suppressed (which is the default mode for demand operations) and RLIB$ code is entered, the print mode in effect is saved, printing is turned off, and the RLIB$ code is interpreted until control leaves RLIB$ code, at which time the print mode is restored and normal execution continues. This command may be used at any time; if it is used to change the mode while the program is in RLIB$ code, the effect will be as if RLIB$ was just exited (from N to A) or entered (from A to N). This means that the print mode will be saved or restored, respectively. If a break interrupt happens to give the user control in RLIB$ (or pseudo-RLIB$) code and RLIB$ tracing is suppressed, the STEP command and the blank line command are disabled to prevent losing control due to the blank line transmitted by the break contingency under certain EXEC levels. Caution should be exercised to turn off printing or else allow RLIB$ tracing so that a runaway print situation can be avoided.
SKIP n S	Return to command mode after executing n number of instruction cycles. If n is omitted, any previously existing skip count is deleted and no skip interrupt occurs. Otherwise, an octal number is used to set the interrupt point. If the count is exceeded during an indirect addressing or execute remote cascade, the command mode is reentered when the instruction is completed.
STEP	Execute one instruction in trace mode and return to command mode.
TOFF$ T	Leave the trace mode and continue execution as if an SLJ TOFF$ command had been executed. Trace mode is terminated and the last instruction is printed.
TON$	Restart a trace that was to be terminated and execute one instruction. To compute the number of instruction cycles performed, use the TOFF$ command followed by the TON$ command. The TON$ command is not affected if the activity is about to terminate by means of an EXIT$ request; if it is desired to continue tracing from that point, a JUMP command must first establish a point from which execution will continue.
TRAP	Enter command mode from trace mode whenever one of a set of locations is referenced or altered except for ER operations. The locations may be octal numbers, register mnemonics, or external symbols. Entering the command "TRAP<loc-1>,<loc-2>..." will place up to sixteen locations in the trap table. Commas must be used as separators. If an asterisk immediately follows the "TRAP", the trap will occur only when a location's contents are changed. If the change occurs via an ER or asynchronously, the trap occurs at the next reference. The use of external symbols is subject to the restrictions noted for the ABSAD command. Each list specified completely replaces the preceding list, unless a + immediately follows the "TRAP" (intervening spaces not allowed), in which case, the specifications are added to the list. The command "TRAP?" will print out the current list. See CHANGE command for use of the symbol $.

The location of some of SNOOPY's control flags is externalized with the label SNFLG$. In addition, a PROC has been added to the standard library which will provide mnemonic tags for the various flags; this PROC is named SN$DEF. The flags defined are described in Table 2-4.

Table 2-4. SNOOPY Control Flags

Flag	Description
REMOTE	Clear for batch mode, set for demand mode. (SNFLG$,,S1)
CCALL	Set by SNCNT$, cleared by SNCLR$. (SNFLG$,,S2)
REMPRT	Set for partial print mode ("PRINT P"), clear for full print mode. (SNFLG$,,S3)
REMSKPIT	Set for suppress print mode ("PRINT N"), clear for full or partial print mode. (SNFLG$,,S4)
CONTNGF	Set when a contingency has occurred, otherwise clear. (SNFLG$,,S5)
REMBRK	Set when the break contingency has occurred or by encountering a condition set by the SKIP (or number) command, the BREAK command, or the TRAP command. When set, SNOOPY will enter command mode before interpreting the next instruction. (SNFLG$,,S6)
WASRLB	Set when executing an RLIB$ or pseudo-RLIB$ element. (SNFLG$+1,,S1)
CMODE	Set when tracing a contingency routine. (SNFLG$+1,,S2)
COMPRT	Set for compressed print ("PRINT C"), clear for expanded print. (SNFLG$+1,,S3)
RLIB$T	Set to enable RLIB$ tracing ("RLIB A"), clear to suppress RLIB$ tracing ("RLIB N"). (SNFLG$+1,,S4)
OLDSKP	Saved value of REMSKPIT when RLIB$T = 0 and WASRLB = 1. (SNFLG$+1,,S5)
ECHO	Set for echo mode of ALTPRT. (SNFLG$+1,,S6)
RLIBX	Set if RLIB list is in exception mode, otherwise clear. (SNFLG$+2,,S1)

2.3.2. CABSAD$

2.3.2.1. Compute Absolute Address (CABSAD$)

Purpose:

Computes the absolute address for a given address relative to a location counter.

Format:

```
L,U     A0,relative-address-to-be-converted
L,U     A1,location-counter-address-is-relative-to
DL      A2,(element-name-containing-address)
LMJ     X11,CABSAD$
        error return
        normal return
```

Registers Used: X11,A0-A5,R1-R3

Description:

When the normal return is taken, A0 contains the requested absolute address and A1 contains a pointer into SLT$. If A1 = 0, the program is not segmented or the element lies in the main segment, and the requested address is always in core. Otherwise, if A1>0, the instruction

```
TP SLT$,A1
```

will skip if the element is in core. On return, H1 of A2 contains the length of the location counter specified and H2 of A2 contains the starting address of that location counter, for possible further use.

If the error return is taken and A0<0, the requested address does not exist because either there is no such element in the executing program, that element has no such location counter, the location given is out of range for that location counter, or the program was collected with the Z option, so that there are no diagnostic tables. If A0>0, then A0 contains an I/O error status code and A1 contains the sector address of the error on FASTRAND mass storage in the file from which the program is loaded.

If a program using CABSAD$ is checkpointed and restarted, the restart contingency routine should contain the instruction

```
SZ  *CABSAD$-1
```

to reinitialize CABSAD$'s tables.

2.3.2.2. Initialize CABSAD$ (CAINIT$)

Purpose:

To initialize CABSAD$ to examine the diagnostic tables of an absolute element other than the one being executed.

Format:

```
DL      A0,file-name
L       A2,addr-of-header-tbl-of-abs-element
LMJ     X11,CAINIT$
        error return
        normal return
```

Registers Used: X11, A0-A5, R1-R3

Description:

This initializes the tables of CABSAD$ to point to the diagnostic tables of the absolute element indicated by the file name and header table start address indicated. Until re-initialized, all references to CABSAD$, CSX$, CBX$, and CSYMVL$ will refer to the absolute element specified. The error return is identical to a CABSAD$ error return. References to SLT$ are not valid, unless CABSAD$ is working on the absolute element being executed.

2.3.2.3. Compute Bank Index (CBX$)

Purpose:

Computes the BDI corresponding to a given bank name.

Format:

```
DL      A0,bankname
LMJ     X11,CBX$
        error return
        normal return
```

Registers Used: X11,A0-A5,R1-R3

Description:

The bank name must be in Fieldata, left-justified and space filled. At the error return, A0 = 0 indicates that the specified name could not be found in the diagnostic table, while A0>0 indicates that an I/O error has occurred. The contents of registers are the same as for an I/O error from CABSAD$. When the normal return is taken, A0 contains the requested BDI.

2.3.2.4. Compute Segment Index (CSX$)

Purpose:

Computes the segment index corresponding to a given segment name.

Format:

```
DL      A0,segment-name
LMJ     X11,CSX$
        error return
        normal return
```

Registers Used: X11,A0-A5,R1-R3

Description:

The segment name must be in Fieldata, left-justified, and space filled. Error conditions are the same as for CBX$. When the normal return is taken, A0 contains the requested segment index. A segment index may be converted to an SLT$ offset pointer by multiplying the index by 4. Segment indexes are the values used in references to ER LOAD$, except for the main segment.

2.3.2.5. Compute Symbol Value (CSYMVL$)

Purpose:

Computes the value of an externally defined global symbol.

Format:

```
DL      A0,symbol-name
LMJ     X11,CSYMVL$
        error return
        normal return
```

Registers used: X11,A0-A5,R1-R3

Description:

The symbol name must be in Fieldata, left-justified, and space filled. Error conditions are the same as for CBX$. When the normal return is taken, A0 contains the value of the symbol and A1 contains an SLT$ pointer as discussed for CABSAD$. If the symbol is absolute, A1 is always zero. If the TYPE EXTDIAG statement was used in the collection source, any externally defined symbol of the collection may be looked up; otherwise, unreferenced symbols and symbols defined in RLIB$ elements are excluded (see Volume 3-2.2.2.13). If the symbol is not a global symbol (i.e., is defined instead by a locally included element), one of the local definitions will be returned, although which one cannot be predicted.

2.3.3. CRELAD$

2.3.3.1. Compute Relative Address (CRELAD$)

Purpose:

Computes the element name, location counter number and address relative to that location number for a given absolute address.

Format:

```
L,U     A0,absolute-address-to-be-converted
LMJ     X11,CRELAD$
        error return
        normal return
```

Registers Used: X11,A0-A5,R1-R3

Description:

When the normal return is taken, A2-A3 contain an element name, left- justified, space filled, A1 contains a location counter number, and A0 contains an address relative to that location counter. These are the relative address values corresponding to the input absolute address. If the input value in A0 is not in range for the program (or segment), i.e., it is less than 01000, between the I- and D-banks, or beyond the end of the D-bank, A0 will be unchanged, A1 will contain 0, and A2-A3 will contain '*ABSOLUTE*' when the normal exit is taken. This will also happen if the diagnostic tables are missing from the absolute element (collection was done with the Z option), for then the necessary information to compute relative addresses is not available.

On normal return, A4 and A5 contain the location counter limits or zeros in case the address was absolute. Specifically, A4 equals the location counter lower limit - 1 and A5 equals the location counter upper limit. Thus, the instruction:

```
TW  A4,x
```

will skip if x contains an address in the same element and location counter range as the address on the call to CRELAD$ that computed A4 and A5. This information may be used to save the computation used by a call to CRELAD$ by using the same A1,A2,A3 and computing A0 = x-(A4+1).

The routine must be part of the main segment and cannot be used by more than one activity at a time.

If more than one element in different segments have addresses corresponding to the given absolute address, the one in main storage will be used. If there is none in main storage, an error condition exists.

The error return is taken if an I/O status other than 0 or 5 is encountered. In that case, A0 contains the status code, A1 the error address on FASTRAND, and A2-A3 the Fieldata characters 'INPUT ERROR', A4-A5 contains zero.

If a program using CRELAD$ is checkpointed and restarted, CRELAD$ must be reinitialized. This may be done by the instruction:

```
SZ  *CRELAD$-1
```

which should be included in the restart contingency code.

Alternate Format:

```
L       A0,(bdi,addr)
LMJ     X11,CRELAD$
        error return
        normal return
```

Registers Used: X11,A0-A5,R1-R3

This entry provides the same results as previously described. The only difference is that the user is permitted to specify a BDI in H1 of A0. This is necessary if the address given in H2 of A0 is in a bank which overlaps the bank containing CRELAD$ (usually the control bank) and the address is in the overlap area. Otherwise, when CRELAD$ performs an ER BANK$ to determine the BDI for the address, it will obtain the wrong BDI since the CRELAD$ PSR is then active.

2.3.3.2. Initialize CRELAD$ (CRINIT$)

Purpose:

To initialize CRELAD$ to examine the diagnostic tables of an absolute element other than the one being executed.

Format:

```
DL      A0,file-name
L       A2,addr-of-header-tbl-of-abs-element
LMJ     X11,CRINIT$
        error return
        normal return
```

Registers Used: X11, A0–A5, R1–R3

Description:

This initializes the tables of CRELAD$ to point to the diagnostic tables of the absolute element indicated by the file name and header table start address indicated. Until re-initialized, all references to CRELAD$, CSN$ and CBN$ will refer to the absolute element specified. The error return is identical to a CRELAD$ error return. References to loaded segments are not meaningful, unless CRELAD$ is working on the absolute element being executed.

2.3.3.3. Convert BDI to Symbolic Bank Name (CBN$)

Purpose:

Determines the symbolic bank name corresponding to the given BDI and returns it in A0–A1 (in Fieldata).

Format:

```
L,U     A0,bdi
LMJ     X11,CBN$
        error return
        normal return
```

Registers Used: X11,A0–A5,R1–R3

Description:

If A0 = 0 at the error return, the BDI was out of range or referred to a common bank. If A0 ≠ 0, an I/O error occurred and the registers contain the same values as for a CRELAD$ I/O error.

2.3.3.4. Convert Segment Index to Segment Name (CSN$)

Purpose:

Determines the symbolic segment name corresponding to the given segment index and returns it in A0–A1 (in Fieldata).

Format:

```
L,U     A0,seg-index
LMJ     X11,CSN$
        error return
        normal return
```

Registers Used: X11,A0-A5,R1-R3

Description:

The error and normal returns are as described for CBN$.

2.4. Editing Routines

2.4.1. Introduction

This subsection describes the three output editing packages EDIT$, AEDIT$ and EOUT$. EDIT$ and AEDIT$ are the newer editing packages and are more efficient and easier to use than EOUT$. It is recommended that EDIT$ or AEDIT$ be used; EOUT$ is documented solely as an aid to those users who are using EOUT$ in existing programs.

2.4.2. Image Composition Editing Package (EDIT$)

EDIT$ is a unified collection of routines which may be used to compose strings of Fieldata characters in an area specified by the user. EDIT$ may be used to produce:

- Images for the printer and the punch,
- Symbiont control images, and
- Control statements for CSF$.

EDIT$ has routines for editing decimal, octal, and floating-point (single- and double-precision) as well as routines for copying character strings and adjusting column settings. Routines are also available for editing the time and the date. A versatile collection of Assembler procedures makes it easy to code for EDIT$.

EDIT$ is reentrant and uses no D-bank storage. Working storage is provided by a six-word packet (10 words if floating-point is used) provided by the user. The format of this packet is given in 2.4.2.4.

The EDIT$ editing package consists of four elements:

- EDIT$ - general purpose editing routines
- EDIT$T - time and date editing routines
- EDIT$F - floating-point editing routines
- EDIT$P - procedure calls for editing routines.

EDIT$ does not reference the other routines. However, both EDIT$T and EDIT$F require the services of EDIT$.

2.4.2.1. EDIT$

Before using the EDIT$ package, edit mode must first be established by calling the EDIT$ routine. When edit mode is on, the index registers X1, X2, and X3 are used as pointers by EDIT$. The original values of these registers are saved in the packet to be restored when edit mode is turned off. The only other registers used by EDIT$ are X11, A0, A1, A2, A3, and R1. These registers are not saved or restored.

All EDIT$ routines are called with an LMJ on X11 and expect the parameters in A0, A1, and A2. A column pointer is maintained by EDIT$ which is advanced by the number of characters inserted whenever an editing function is performed. The first column is zero.

Table 2-5 describes the EDIT$ routines.

2.4.2.2. Time and Date Editing Routines (EDIT$T)

Table 2-6 describes the time and date editing routines.

Each of these routines accepts input in TDATE$ format; that is, in the form:

F m,d,y,t

where:

F = FORM 6,6,6,18

and

m = Month(1-12)
d = Day
y = Year (relative to 1964)
t = Number of seconds since midnight.

Table 2-5. General Purpose Editing Routines (In EDIT$)

Routine	Function	Operation
ECHAR$	insert a character	The character contained in the low-order six bits of A0 will be inserted in the image.
ECLEAR$	clear image	The image will be set to blanks and the column pointer will be set to the start of the image (column zero).
ECOLN$	compute the column number	Returns with the column number in A0.
ECOL$	position to a fixed column	The column pointer will be changed to the number specified in A0. The first character position of the image is column 0.
ECOPY$	copy a string	A0 is expected to contain the address of a string of characters. Number of characters in A1 will be copied from this area into the image.
EDCFZ$	edit decimal (fixed length with leading zero)	Edit A0 to a decimal number right-justified in a field of A1 characters. A leading '-' is added if A0 is negative. The result will overflow the field to the right if it does not fit in the specified field size. Leading zeros will be produced where EDECF$ would produce leading spaces. This routine is convenient for editing decimal fractions.
EDECF$	edit decimal (fixed length)	Edits A0 to a decimal number right-justified in a field of A1 characters. A leading '-' is added if A0 is negative. The result will overflow the field to the right if it does not fit in the specified field size.
EDECV$	edit decimal (variable length)	Edits A0 to a decimal number. A leading '-' is added if A0 is negative. The number of characters generated is a function of the sign and magnitude of A0.
EDITR$	reenter edit mode	Must be called with the address of the packet in A0. Edit mode will be reestablished and the column pointer (saved by EDITX$) will be restored.
EDITX$	leave edit mode	Will terminate edit mode. The column pointer will be saved in the packet. EDITX$ will return with the packet address in A0.
EDIT$	initialize and establish edit mode	Must be called with the address of the packet in A0. S3 of the first word of the packet is expected to contain the number of words in the image area and H2 of the first word of the packet is expected to contain the location of the image area. The image will be set to blanks and edit mode will be established. The column pointer will be set to the start of the image.
EFD1$	insert Fieldata (one-word)	The six characters in A0 will be inserted into the image. Blanks and master spaces will be ignored.
EFD2$	insert Fieldata (two-words)	The 12 characters in A0 and A1 will be inserted into the image. Blanks and master spaces will be ignored.

Table 2-5. General Purpose Editing Routines (In EDIT$) (continued)

Routine	Function	Operation
EMSGR$	message editor reentry	This routine performs the same operation as EMSG$ except that the pointer to the input string is taken from the packet. With these two routines it is possible to copy a string into the image, occasionally interrupting this process to perform other editing functions at certain selected points in the string.
EMSG$	message editor initial entry	Copies the characters starting at the address given in A0 into the image. This process stops when the character in S2 of the first word of the packet is found. The pointer to this string is saved in the packet and return is made to the user.
EOCTF$	edit octal (fixed length)	Edits A0 to A1 octal digits. Leading zeros are not suppressed. A1 must not exceed 12.
EOCTV$	edit octal (variable length)	Edits contents of A0 to octal. The number of digits edited will depend on the size of the number. If the number is greater than seven, a leading zero will be added.
EPACK$	copy and pack a string	Same as ECOPY$ except that master spaces will be ignored.
ESKIP$	advance the column pointer	The number given in A0 will be added to the column pointer. A0 may be negative.

Table 2-6. Time and Date Editing Routines (In EDIT$T)

Routine	Function	Operation
EDAY1$	Edit a date (format mm dd yy)	Edits the date portion of A0 into the format mm dd yy where mm, dd, yy are the two digits of the month, day and year, respectively.
EDAY2$	Edit a date (format dd mmm yy)	Edits the date portion of A0 into the format dd mmm yy where dd and yy are the two digits of the day and year, respectively, and mmm is a three character abbreviation for the month.
EDAY3$	Edit a date (format month dd, year)	Edits the date portion of A0 into the format month dd, year Where month is the name of the month (up to 9 characters) dd is the day (one or two characters) and year is the four digit year.

Table 2-6. Time and Date Editing Routines (In EDIT$T) (continued)

Routine	Function	Operation
EDAY4$	Edit a date (format yearmmdd)	Edits the date portion of A0 into the format yearmmdd where year is the four year digit year and mm and dd are the two digits of the month and day respectively.
EDAY5$	Edit a date (format yymmdd)	Edits the date portion of A0 into the format yymmdd where yy, mm and dd are the two digits of the year, month and day respectively.
ETIME$	Edit a time	Edits the time portion of A0 into the format hh:mm:ss where hh, mm and ss are the two digits of the hours, minutes and seconds of the time, respectively.

2.4.2.3. Floating-Point Editing Routines (EDIT$F)

The floating-point editing routines require three parameters in addition to the address of the floating point packet. The first two are called *x* and *y* and are supplied in S5 and S6 of A0. They specify the format of the edited number as described below. The third parameter is the single or double precision number to be edited. If single precision, the number is specified in A1; if double precision, the number is specified in A1 and A2.

Edited numbers may appear in one of the two general formats: "fixed decimal" format (no exponent) and "scientific" format. A number edited in "fixed decimal" format consists of a sign (if negative) followed by the digits of the number and an embedded decimal point. The sign is present only if the number is negative; a positive number has no sign character. The parameter *x* specifies the width of the edited field: the number of character positions the edited number will occupy, including sign and decimal point. If the value of *x* is given too small to represent the value being edited, it is ignored and as many character positions are used as is required to correctly represent the number. The parameter *y* specifies how many fractional digits are to be edited. This is the number of digits appearing to the right of the decimal point. The value of *y* may be any value greater than or equal to zero, but should always be less than the value of *x* to allow for the sign and decimal point characters.

The "scientific" format is similar to the "fixed decimal" format but has an exponent at the right. The exponent consists of an optional exponent character (usually 'E' for single precision and 'D' for double precision numbers) followed by the exponents's sign (either "+" or "-") and the exponent's digits. Exponents of single precision numbers are always edited as a two digit value, and those of double precision numbers are always edited as a three digit value. The parameter *x* specifies the total number of character positions to be occupied by the edited number, including signs, digits, decimal point, and exponent character. If the edited numbers require more character positions than is specified by the value of *x*, the parameter *x* is ignored and the required character positions are used. The parameter *y* specifies the total number of mantissa digits to appear in the edited number. It

includes the digits on both sides of the decimal point but not the decimal point itself. The placement of the decimal point within the *y* digits of the mantissa is determined by the value in the packet cell FPS. This is the number of digits to appear to the left of the decimal point. It is normally set to one.

Table 2-7 describes the floating-point editing routines.

Table 2-7. Floating-Point Editing Routines (In EDIT$F)

Routine	Function	Operation
EFLF1$	Single-precision fixed decimal format	A1 is edited to fixed decimal format with y digits following the decimal point, right-justified, in a field of size *x*.
EFLF2$	Double-precision fixed decimal format	A1, A2 is edited to fixed decimal format with *y* digits following the decimal point, right-justified, in a field of size *x*.
EFLG1$	Single-precision generalized format	Same as scientific format except that an attempt is made to move the decimal point in such a way as to reduce the exponent to zero. If this can be done the exponent is set to blanks. Otherwise scientific format is used.
EFLG2$	Double-precision generalized format	See EFLG1$ and EFLS2$.
EFLS1$	Single-precision scientific format	A1 is edited to scientific format with *y* significant digits in a field size of *x*. The exponent is two digits long.
EFLS2$	Double-precision scientific format	A1, A2 is edited to scientific format with *y* significant digits in a field size of *x*. The exponent is three digits long.

2.4.2.4. The Packet Format

The EDIT$ packet format is: (Italicized fields are filled by EDIT$)

0	ts	msg	il	iloc		
1	*cix*	*wix*	*cim*	*wim*		
2	fps	fpr	zero	*return*		
3	*ret*			*save1*		
4	*save2*					
5	*save3*					
6	dpc	spc	*ndp*	*ndf*	*sign*	*zero*
7	*fcol*			*scale*		
8	*value*					
9						

Word 0

ts – This area may be used as a Test-and-Set flag by the user.

msg – Signal character for EMSG and EMSGR$.

il – Image length in words.

iloc – Image location (18 bit address).

Word 2

fps – Number of digits to be placed to the left of the decimal point for scientific format floating-point editing.

fpr – If nonzero the floating-point rounding option is on: 5 is added to the first significant digit after the last significant digit to be printed.

Example:

Input 1.4356

3 digits printed

Result = 1.44

zero – Must be zero.

Word 6

dpc – If nonzero, specifies the character which is to separate the mantissa and the exponent for scientific format double-precision floating-point editing. Normally D if nonzero.

spc – If nonzero, specifies the character which is to separate the mantissa and the exponent for scientific format single-precision floating-point editing. Normally E if nonzero.

The following fields are maintained by EDIT$ and are of concern to the user only for debugging:

cix – Used by EDITX$ to save the character index for EDITR$.

wix – Used by EDITX$ to save the relative word index for EDITR$.

cim – Used by EMSG$ and EMSGR$ to save the character index to the input message.

wim – Used by EMSG$ and EMSGR$ to save the word index to the input message.

return – Used as the return point for the character store vector.

ret – Used to save the return point to the user during calls to other EDIT$ functions.

save1 – Used to save the modifier portion of X1 during edit mode.

save2 – Used to save X2 during edit mode.

save3 – Used to save X3 during edit mode.

ndp – Number of digits to be edited before the decimal point.

ndf – Number of digits to be edited following the decimal point.

sign – Nonzero if the floating-point number being edited is negative.

zero – Nonzero if the floating-point number being edited was not normalized.

fcol – Contains information to facilitate proper positioning of the edited result.

scale – Used to build the exponent.

value – Used to save intermediate results (two words). (Used only by floating-point.)

Two PROCs are available to generate a packet for EDIT$. The PROC E$PKT generates a six-word packet and the PROC E$PKTF generates a 10-word packet suitable for floating-point editing. The format of the PROC call is:

```
label   PROC-name   IL,ILOC   L1   L2  ...
```

The lists L_1, L_2, ... are used to override the assumed values to be placed in the fields msg, fps, fpr, dpc, spc. The assumed values are msg = '&', fps = 1, fpr = 1, dpc = 0 and spc = 0. The format of an override list L_1 consists of the name of the field enclosed in quotes, a comma, and the value to be used. For example, the following PROC reference will generate a packet which will edit floating-point numbers in a format similar to that used by FORTRAN:

```
PKT   E$PKTF   22,LINE   'FPS',0 'FPR',0 'SPC','E' 'DPC','D'
```

2.4.2.5. Procedures for EDIT$ (EDIT$P)

Each routine in the EDIT$ editing package has a corresponding procedure call. The name of the procedure may be found by deleting the '$' at the end of the routine name and placing a '$' between the first and second characters.

Some of the procedures do not require any parameters. These are E$DITX, E$CLEAR, E$COLN, E$MSGR, ETD, EDAT1, E$DAT2, E$DAT3, E$DAT4 and E$DAT5. The last six are special procedures which generate an executive request to TDATE$ and then call ETIME$, EDAY1$, EDAY2$, EDAY3$, EDAY4$ and EDAY5$ respectively.

The procedures EDIT, EDITR, E$CHAR, E$COL, and E$SKIP accept one parameter and generate:

```
LA,U        A0,<parameter>
LMJ         X11,<routine>
```

If there are no parameters, then the load instruction is not generated. The parameter may include indexing and may also specify a j-designator to override the implied "U". "*"(s) in the appropriate places will, of course, be honored. For example:

```
E$CHAR    'A'
```

will generate

```
LA,U        A0,'A'
LMJ         X11,ECHAR$
```

but

```
E$CHAR      *TAG,X6,S5
```

will generate

```
LA,S5       A0,*TAG,X6
LMJ         X11,ECHAR$
```

also

```
E$CHAR
```

will generate simply

```
LMJ         X11,ECHAR$
```

The procedures E$OCTV, E$DECV, E$FD1, E$FD2, EMSG, ETIME, E$DAY1, E$DAY2, E$DAY3, E$DAY4 and E$DAY5 will generate . . .

```
LA          A0,<parameter>
LMJ         X11,<routine>
```

with the exception of E$FD2 which will generate a Double Load A instruction. Again an index and a j-designator may be specified as well as indirect addressing and automatic index incrementation. If the parameter is missing, the load instruction will not be generated. For example:

```
E$DECV COUNT,,H2
```

will generate

```
LA,H2       A0,COUNT
LMJ         X11,EDECV$
```

but

```
E$DECV
```

will generate

```
LMJ         X11,EDECV$
```

The routines ECOPY$, EPACK$, EOCTF$, EDECF$ and EDCFZ$ require two parameters. The general form of the call for their procedures is:

```
<proc name>  P1,P2
```

and they generate

```
LA          A0,P2
LA,U        A1,P1
LMJ         X11,<routine>
```

except that ECOPY$ and EPACK$ generate a "U" j-designator on the first instruction. Again, indexing, an overriding j-designator, etc., may be specified. If P2 is missing the first instruction will not be generated. If both P1 and P2 are missing neither load instruction will be generated. For example:

```
E$COPY      50,LINE
```

will generate

```
LA,U        A0,LINE
LA,U        A1,50
LMJ         X11,ECOPY$
```

but

```
E$COPY      50,TAB,X6,W
```

will generate

```
LA          A0,TAB,X6
LA,U        A1,50
LMJ         X11,ECOPY$
```

also

```
E$COPY      COUNT
```

will generate

```
LA,U        A1,COUNT
LMJ         X11,ECOPY$
```

The PROC call

```
E$DECF      10,COUNT
```

will generate

```
LA          A0,COUNT
LA,U        A1,10
LMJ         X11,EDECF$
```

The procedures E$FLS1, E$FLS2, E$FLG1, E$FLG2, E$FLF1 and E$FLF2 have the following form for their procedure call:

```
<proc name>  x*/6+y,p
```

where x and y are defined in the section on the floating-point editing routines (see 2.4.2.3.). The procedures will generate:

```
LA          A1,p
LA,U        A0,x*/6+y
LMJ         X11,<routine>
```

except that the double-precision PROCs generate a Double Load A instruction, for the first instruction. Indexing, indirect addressing and automatic index incrementation may be specified for p. The dropout rules are the same as before. For example:

```
E$FLF2      30*/6+18,(1.234D)
```

will generate

```
DL          A1,(1.234D)
LA,U        A0,30*/6+18
LMJ         X11,EFLF2$
```

but

```
E$FLF1      20*/6+9
```

will generate

```
LA,U        A0,20*/6+9
LMJ         X11,EFLF1$
```

2.4.3. ASCII Image Composition Editing Package (AEDIT$)

AEDIT$ is a set of reentrant subroutines used for composing strings of ASCII characters in a user-specified area. The AEDIT$ package is very similar to the EDIT$ package used for Fieldata images. AEDIT$ is useful in preparing images for:

- Printed ASCII output (ER APRINT$)
- Punched ASCII output (ER APUNCH$)
- Other Executive Requests which require ASCII images.

2.4.3.1. The AEDIT$ Packet

AEDIT$ works from the following packet. Words 7–10 are used for editing floating-point numbers only.

0	[Test and Set]	*gwm*	image length	image address		
1	*char index*	*word index*	*AEMSG$ char.*	*AEMSG$ word (index)*		
2	fps	fpr	zero	*return address for char. store*		
3	*user's return address*			*save of original X1 modifier*		
4	*save of original X2 contents, or save of character pointer*					
5	*save of original X3 contents, or save of word pointer*					
6	AEMSG$ Stop char. (Q1)	[dpc] (Q2)		[spc] (Q3)	unused	
7	unused		*digits before*	*digits after*	*negative sign*	*not normalized*
8	*final column position*			*exponent's power of ten*		
9	*save area for intermediate floating-point results*					
10						

S2 of word 0 contains the PSR status for third/quarter-word mode and character addressing mode of the user on initial entry. Bit 0 of S2 will reflect the state of D4 (PSR bit 31 – character addressing mode) and bit 3 of S2 will reflect the state of D10 (PSR bit 17 – third/quarter-word mode) as defined for the LPD and SPD instructions. All other bits will be 0. When the user calls AEDITX$, qwm will be used to restore the user to the state he was in on entry to AEDIT$. In AEDIT$ mode all routines function in quarter-word mode.

All other items in the AEDIT$ packet carry the same meaning as they do in the EDIT$ packet. The following changes in location occur from the EDIT$ packet:

- The AEMSG$ stop character is stored in Q1 of word 6, instead of S2 of word 0 (for the EMSG$ stop character in the EDIT$ packet).
- The dpc and spc characters are stored in Q2 and Q3 of word 6 instead of S1 and S2 of word 6.
- Words 7–10 of the AEDIT$ packet are the same as words 6–9 of the EDIT$ packet.
- Q4 of word 6, and S1 and S2 of word 7 are unused in the AEDIT$ packet.

2.4.3.2. Generating the AEDIT$ Packet

The following PROC call generates a seven-word AEDIT$ packet (for non- floating-point routines), where msg='&' if field 2 is omitted.

```
A$EPKT image-length,image-address ['MSG','AEMSG$-stop']
```

The following PROC call generates an eleven-word AEDIT$ packet (for editing floating-point numbers), where the values msg = '&', fps =1, fpr = 1, dpc = 0, spc = 0 are inserted if the corresponding field is omitted.

```
A$EPKTF image-length, image-address ['MSG','AEMSG$-stop'] ;
    ['FPS',fps number] ['FPR',fpr number] ;
    ['DPC',dpc-char] ['SPC',spc-char]
```

2.4.3.3. ASCII Editing Routine Descriptions

The AEDIT$ routines use the same calling sequences as the corresponding EDIT$ routines, except for the routine names. To get the AEDIT$ routine name, simply add 'A' before the name of the corresponding EDIT$ routine (e.g., ECHAR$ becomes AECHAR$). To get the AEDIT$ PROC name, replace the leading 'E$' of the corresponding EDIT$ PROC name with 'A$E' (e.g., E$SKIP becomes A$ESKIP).

Table 2-8 describes the routines for initialization and termination of ASCII editing mode. Table 2-9 describes the general pupose ASCII editing routines, and Table 2-10 describes the ASCII floating-point editing routines.

Table 2-8. Initialization and Termination of ASCII Editing Mode

PROC	Routine	Description
A$EDIT	AEDIT$	Initial entry into ASCII edit mode. The image is space filled; the column pointer is set to column 0; quarter-word mode is set; X1-X3 are saved. The AEDIT$ package uses, but does not save or restore X11, A0-A3, R1.
A$EDITR	AEDITR$	Reentry into ASCII edit mode. Column pointer is restored.
A$EDITX	AEDITX$	Terminate ASCII edit mode. Restore X1-X3; save column pointer in packet.

Table 2-9. General Purpose ASCII Editing Routines

PROC	Routine	Description
A$ECHAR	AECHAR$	Edit an ASCII character. Insert the ASCII character from Q4 of A0 into the image.
A$ECLEAR	AECLEAR$	Set image to blanks and column pointer to start of image (column 0).
A$ECOL	AECOL$	Position the pointer to a fixed column.
A$ECOLN	AECOLN$	Obtain the current column number in A0.
A$ECOPY	AECOPY$	Copy a string into the image.
A$EDAY1	AEDAY1$	Edit the date portion of A0 into the format: mm/dd/yy (Use A$EDAT1 for the current date)
A$EDAY2	AEDAY2$	Edit the date portion of A0 into the format: dd mmm yy (Use A$EDAT2 for the current date)
A$EDAY3	AEDAY3$	Edit the date portion of A0 into the format: month dd, year (Use A$EDAT3 for the current date)
A$EDAY4	AEDAY4$	Edit the date portion of A0 into the format: yearmmdd (Use A$EDAT4 for the current date)
A$EDAY5	AEDAY5$	Edit the date portion of A0 into the format: yymmdd (Use A$EDAT5 for the current date)
A$EDCFZ	AEDCFZ$	Convert to ASCII decimal (fixed field length) with leading zeros.
A$EDECF	AEDECF$	Convert to ASCII decimal (fixed length field).
A$EDECV	AEDECV$	Convert to ASCII decimal (variable length field).
A$EFD1	AEFD1$	Insert ASCII (one word). Insert the contents of A0 (four ASCII characters) into the image, excluding any quarter-word whose value is 040 (ASCII space) or 000.
A$EFD2	AEFD2$	Insert ASCII (two words). This is the same as AEFD1$, except that it inserts the contents of A0 and A1 (eight ASCII characters) into the image.
A$EMSG	AEMSG$	Message editor (initial entry). Insert ASCII characters starting at the address in A0 into the image. This process stops when the AEMSG$ stop character (Q1 of word 6 of the packet) is encountered in the string.

Table 2-9. General Purpose ASCII Editing Routines (continued)

PROC	Routine	Description
A$EMSGR	AEMSGR$	Message editor (reentry).
A$EOCTF	AEOCTF$	Convert to ASCII octal (fixed length field).
A$EOCTV	AEOCTV$	Convert to ASCII octal (variable length field).
A$EPACK	AEPACK$	Copy and pack a string into the image. This is the same as AECOPY$, except that the quarter-word whose value is 000, although included in the character count, is not inserted into the image.
A$ESKIP	AESKIP$	Skip an area in the image (advance column pointer).
A$ETIME	AETIME$	Edit the time portion of A0 into the format: hh:mm:ss (Use A$ETD for the current time)

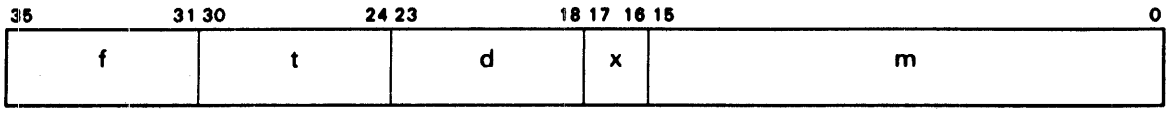
Table 2-10. ASCII Floating-Point Editing Routines

PROC	Routine	Description
A$EFLF1	AEFLF1$	ASCII single-precision fixed-point format. A1 is edited to fixed-point format with y digits following the decimal point, right-justified, in a field of size x, where the calling sequence is: L,U A0,x*/6+y L A1,addr.-of-number LMJ X11,AEFLF1$
A$EFLG1	AEFLG1$	ASCII single-precision generalized format.
A$EFLS1	AEFLS1$	ASCII single-precision scientific format.
A$EFLF2	AEFLF2$	ASCII double-precision fixed-point format. This is the same as AEFLF1$, except that the contents of A1 and A2 are edited.
A$EFLG2	AEFLG2$	ASCII double-precision generalized format.
A$EFLS2	AEFLS2$	ASCII double-precision scientific format.

2.4.4. EOUT$ (Generalized Output Editing Routines)

EOUT$ is an interpretive routine which performs editing functions for Fieldata output produced for a printer, communications terminal, card punch, or display console.

The interpretive instructions performed by the routine are constructed similarly to machine language instructions, the format is:



where:

f – Function code

t – Print position (printer) or character position (display console). Print position numbering starts at zero.

d – Decimal point location

x – Specifies indirect address and use of the simulated index register

m – Address (main storage location of data)

EOUT$ is called by:

```
LMJ  X11,EOUT$
```

There are two entry points to this subroutine. The normal entry point is EOUT$. The other, EOUTR$, is the point for reentry after the E$TERM (terminate) function (see 2.4.4.4.).

The addressed word in the m–field may be either in a control register or main storage. Any word, even if in a volatile register, is permissible; but if register X11 is addressed, the location of the interpretive word which references X11 is output. All registers, including volatile ones, are saved and restored. The x–field is used to specify indirect addressing and the use of the single simulated index register. Its permissible values (in octal) are:

00 -- No action
01 -- Use address indirectly
02 -- Apply simulated index register
03 -- Apply simulated index register then use address indirectly

Indirect addressing is permitted to one level only, and the x–, h– and i– field of the indirectly addressed word are ignored. It is possible, however, to indirectly address control storage. All modes may be used with indirect addressing.

The various functions are described in the following paragraphs. All may be called as procedures. Each of the procedure calls generates one word in the correct format. The parameters of these procedures are interpreted differently depending on the number written. A single parameter is taken m; two parameters m and x; three parameters t, d and m; and four parameters t, d, m and x. Any missing parameters are assumed to be zero.

Entry to EOUT$ may be obtained by the procedure E$OUT or E$OUTR, depending on the entry point desired. No parameters are required.

2.4.4.1. Editing Functions

These functions convert the information to be output. In all cases, except E$A (alphanumeric words), the field specifies the print position at which the rightmost digit, bit, or character is to be printed. The number given in parentheses following the procedure call is the octal function code.

E$D(01) – Decimal: The address word is treated as if it were a signed decimal integer and is edited without a decimal point unless a set function (see E$PNT – 2.4.4.3.) is in effect. Leading zeros to the left are suppressed and a minus sign, if any, is printed immediately to the left of the number (also see E$OVRP – 2.4.4.3.). If the value is zero, a single zero is printed. If a set point is in effect, the decimal number is assumed to have the started point specified by the set point, and the d–field specifies the number of decimal digits to be printed to the right of the decimal point. If a set field function (see E$FLD – 2.4.4.3.) with D = 0 is in effect, the specified field is treated as an unsigned decimal integer.

E$O(02) – Octal: The d low–order bits of the addressed word are edited and printed as (d+2)/3 octal digits, unsigned. For a full octal, binary, or alphanumeric character word, d must always be given as 36.

E$B(03) – Binary: The d low–order bits of the addressed word are edited as d binary digits unsigned.

E$C(04) – Alphanumeric Characters: The d low–order bits of the addressed word are edited and printed as (d+5)/6 alphanumeric characters in Fieldata code.

E$A(05) – Alphanumeric Words: the d words beginning with the addressed word are edited as 6*d characters in Fieldata code. For this editing function only, the t–field specifies the print position at which the leftmost character is printed.

E$E(06) – Floating–Point (FORTRAN E): The addressed word is edited as a floating–point number with d significant digits. Normally these are all printed to the right of the decimal point (see E$SCL – 2.4.4.3.). A decimal exponent consisting of a sign and two digits is inserted immediately to the right of the significant portion. If the floating–point number is negative, a minus sign is inserted immediately to the left of the number (see E$OVRP – 2.4.4.3.). If the addressed word is minus zero, there is no effect, and the field is left blank.

E$F(07) – Floating–to–Fixed (FORTRAN F): The addressed word is assumed to be a floating–point number and is edited to fixed point with d places following the decimal point. Negative numbers, including minus zero, are treated as in E$E.

E$DE(26) – Double–Precision Floating–Point: This editing function is the same as E$E with the addressed word and the addressed word plus one edited as a double–precision floating–point number. A decimal exponent consisting of a sign and three digits is inserted immediately to the right of the significant portion.

E$DR(27) – Double–Precision Floating–to–Fixed: This editing function is the same as E$F with the addressed word and the addressed word plus one edited as a double–precision floating–point number.

EIMG$ – The edited output of EOUT$ is sorted in a 22–word buffer in EOUT$'s D–bank following the externally defined label EIMG$.

2.4.4.2. Output Functions

The output functions serve to transmit the edited line to an output device. The device to be used is determined by the d-field:

Printer	d=0
Card Punch	d=1
Display Console	d=2

The word or character count is given in the t-field. This count must be given. (It is not assumed maximum if it is given as zero.) For the printer, the word count is normally 22; for the card punch, normally 14. For the display console, the t-field is a character count and cannot be more than 60.

For the printer, the m-field serves to specify the number of lines to be spaced. A value greater than the length of a logical page results in printing on the first line of the next page. For the punch and display console, the m-field is ignored. The number given in parentheses following the procedure call is the octal function code.

E$WT(10) - Write and Terminate: The edited image is transmitted to the specified device, and the routine returns control to the next instruction in machine language mode. The image is not reset to blanks.

E$W(11) - Write: The edited image is transmitted to the specified device and the routine continues in the interpretive mode. The image is reset to blanks.

E$WS(12) - Write and Save: The edited image is transmitted to the specified device and the routine continues to the next instruction in the interpretive mode. The image remains available for use by further output functions or further editing.

2.4.4.3. Modal Functions

The modal functions serve to enter information which affects the interpretation of one or more of the instructions which follow. The number given in parentheses following the procedure call is the octal function code.

E$SCL(13) - Set Scale: The contents of the address field is treated as a signed power of 10 to be applied to any floating-point or floating-to-fixed function which follows the set scale function. For floating-point, the scale is the number of digits to be printed to the left of the decimal point. The exponent field is reduced accordingly, so that the resulting value is the same as if no set scale function were in effect. Negative values of the address (the 16-bit ones complement) introduce leading zeros after the decimal point and increases the exponent field accordingly.

For floating-to-fixed conversion, the actual value of the resulting number is altered by multiplying it by the power of 10 indicated by the address. The set scale function remains in effect until it is countermanded by a new set scale. Upon initial entry to EOUT$, the scale is assumed to be 0.

E$PNT(14) - Set Point: The set point function specifies the position of the binary point for the next editing function to be encountered (presumably a decimal editing function). It remains in effect only for the single edit. The address of the set point gives the number of bits following the binary point. Negative values are permitted (see E$FLD).

E$FLD(15) – Set Field: The set field function is used to specify a subfield of the next word to occur (presumably decimal, octal, binary or alphanumeric character function). The t–field specifies the lefthand margin and the m–field the righthand margin. The bits of the machine word are numbered, for the purposes of this function, from left (00) to right (35). The d–field specifies extension of sign; if it is nonzero, the field is treated as sign. A set field function with d=0 and t=0 may be used to treat fields, including the sign bit, as unsigned unless m=35 (that is, a whole word must always be signed in the event a sign is applied).

The set field function remains in effect only for the next function encountered. If both a set field and a set point function are in effect when editing occurs, the set field function is applied first. In this case, the set point function specifies the binary point counting from the righthand end of the specified field.

E$INDX(16) – Set Index: The set index function is used to address a quantity in main storage which is to be loaded into the single simulated index register. For any function which addresses storage (including this one), the presence of a 1 bit in the increment (h) portion of the address causes the simulated index to be added to the specified address before access is made. The left half of the index register word is ignored. If the d–field is nonzero, the contents of the m–field (with sign extension) is loaded into the simulated index register. The set index function remains in effect until it is countermanded by another set index function.

E$OVRP(17) – Overpunch: The overpunch function specifies that any minus signs produced by the editing functions are to be removed from their positions in front of the edited numbers and placed as 11–punches over the low–order digits. In the case of floating–point editing, the sign of the mantissa is placed over the low–order digit of the mantissa and the sign of the exponent over its low–order digit. The space that would normally contain the sign of the exponent is omitted.

The overpunch function is initiated by its occurrence with address 1. It is countermanded by its occurrence with address 0. Upon initial entry to EOUT$, the overpunch mode is assumed to be off.

2.4.4.4. Control Functions

The control functions serve to introduce into the interpretive language some of the control operations available in machine language. The number given in parentheses following the procedure call is the octal function code.

E$TERM(20) – Terminate: The terminate function causes the routine to return to the next instruction in machine language. Upon reentry at point EOUTR$, all counters, modes in effect, interpretive subroutines, and any partial image are left undisturbed; control is returned to the next instruction in machine code. If reentry is made at EOUT$, these are all cleared and control is returned to the interpretive mode. Entry at EOUT$ is made by:

```
LMJ  X11,EOUTR$
```

E$LINK(21) – Link: The link function is used to form subroutines in the editing language. Its effective address specifies the location of the entry to a subroutine. Subroutines may be nested to a depth of 10.

E$JUMP(22) – Jump: The jump function with a nonzero effective address causes an interpretive transfer of control to the designated location. If the address is zero, the jump function serves as a subroutine exit. Transfer is to the interpretive function following that link control most recently executed for which no exit has been performed.

E$RPT(23) – Repeat: The repeat function causes the next single interpretive function to be repeated the number of times specified in the d–field of the repeat word. A repeat function preceding E$LINK is meaningless; for multiple execution of E$LINK, the routine EOUT$ should be called within a machine language loop. The t– and m–fields contain increments to the t– and m–fields of the instruction to be repeated for each execution. Any modes set by the modal functions which would be in effect for the first execution of a repeated instruction remain in effect for all executions.

E$CLEAR(24) – Clear: The clear function sets the image to blanks.

2.4.4.5. EOUT$ Calling Sequences

Several examples (The use of FORTRAN formats here is merely to indicate the format desired. The I/O functions in FORTRAN employ an editing scheme peculiar to themselves.) of typical calling sequences to EOUT$ follow:

Example 1:

The FORTRAN instruction:

```
      PRINT 100,A,I,N,B,c

  100 FORMAT (6X,E20.7, 120, 020, 1P.2F20.6)
```

is equivalent to the interpretive sequence:

```
E$OUT
E$E       26,7,A
E$D       46,0,1
E$O       66,36,N
E$SCL     1
E$F       86,6,B
ESF       106,6,C
E$WT      22,0,1
```

Next machine language instruction.

Example 2:

If this line were to be put out on the card punch, whose output code is 1, then the last interpretive instruction would be replaced by:

```
E$WS      14,1,0
E$WT      22,0,1
```

Only the first 80 columns of the image would be punched.

Example 3:

The FORTRAN statements

```
      PRINT 100 (J (I), K (I), L (I), M (I), I=1,4)
  100 FORMAT (2016)
```

are equivalent to the following interpretive sequences:

```
E$RPT    30,4,1
E$D      6,0,J,2
E$RPT    30,4,1,
E$D      12,0,K,2
E$RPT    30,4,1
E$D      18,0,L,2
E$RPT    30,4,1
E$D      24,0,M,2
E$WT     22,0,1
```

2.5. Processor Interface Routines (PIRs)

A set of routines, which are available in the system relocatable library to provide a standard interface with the operating system for all system and language processors. These routines simplify the task of incorporating additional processors into the operating system.

In general, processors assign input and output files, obtain source and correction input, and generate source and relocatable output. When using the processor interface routines, the processor need only be concerned with requesting the next input image and outputting a relocatable word.

The various processor routines are described briefly in the following paragraphs. Detailed descriptions follow. Note that the volatile register set (X11, A0–A5, R1–R3) is assumed available for the routines.

2.5.1. Preprocessor Routines (PREPRO,PREPRM)

The PREPRO routine is designed for use by processors which require source input, source output, and relocatable output parameters on the processor control statement. The PREPRM routine is designed for use by processors which require only source input and source output parameters.

2.5.1.1. Preprocessor Routine (PREPRO)

Purpose:

- Reads the INFOR from processor call card
- Standardizes its form in PARTBL (see Figure 2-1)
- Dynamically assigns the necessary files
- Obtains the options
- Obtains the 'next write location' for program files
- Verifies the element for tape files

Initial Conditions:

The processor must not have done an ER to READ$ prior to entering PREPRO.

PARTBL must be externally defined, at least 40 words long, and should be zero filled.

Format:

```
LMJ     X11,PREPRO$
        error return
        normal return
```

Description:

The user-provided PARTBL is filled with data necessary to process the files. The input/output files will be assigned as follows:

SI - assigned
RO - exclusively assigned
SO - exclusively assigned

The RO and SO assign will override the SI assign.

The error return is taken under the following conditions:

I/O error occurs (A0 = 0)

control card is in error
tape positioned improperly
program file is undefined

A0 ≠ 0

2.5.1.2. Preprocessor Routine (PREPRM)

Purpose:

Generates the source input and source output fields of PARTBL (see Figure 2-1). Performs dynamic assigns and/or program file searches on the source input and/or source output files.

Initial Conditions:

PARTBL must be externally defined, 27_{10} words long (or 40_{10} words long if relocatable output field is needed), and should be zero filled.

Format:

```
LMJ     X11,PREPRM$
        error return
        normal return
```

Description:

The information on the processor call statement is transferred to PARTBL (see Figure 2-1).

The input/output files will be assigned as follows:

SI - assigned
SO - exclusively assigned

The SO assign will override the SI assign.

On error return, an error message is printed.

2.5.1.3. PARTBL Description

All file, element and version names are left-justified and space filled.

Filenames are dynamically attached usenames (SI$, RO$, SO$). The default filename is TPF$. NAME$ is the default element name.

Word 0

file type
000 - Input from run stream
050 - Program file
076 - Tape file

options
Option letters are A-Z left to right, i.e., Z option, bit zero set to one.

Word 13

file information
Contains indicators for use by the routine POSTPR$ to restore program files to their original assignment state.
S1 - source input
S2 - source output
S3 - relocatable output

00 – no action
01 – @FREE,A remove attached name (SI$, SO$, RO$)
02 – @FREE,AX remove attached name and release exclusive-use
03 – @FREE,AR remove attached name and free file.

ITI – input termination indicator — Contains the SIR$ input termination indicator for purposes of processor reusability. The possible values are:

0 – Reuse of the processor is possible (input terminated by call to CLOSR$, or @EOF image, end of @ADD,E file, INFOR CLIST image, or transparent control image)
1 – Bit 35 end-of-file has been encountered
2 – @ENDX has been encountered

requested cycle — Contains source input element cycle requested. If none requested the latest cycle is used.

2.5.1.4. Reusable Processor Construction

Initial program load operations are expensive relative to swapping. A mechanism is provided to allow a program to make itself reusable, thereby avoiding the need to reload itself. This is applicable to programs called by processor call statements but not by @XQT. If the processor in question makes use of the standard processor interface routines (PREPRO, PREPRM, POSTPR$, ROR, SOR, SIR$), implementing reusability is not a major task. The steps involved are as follows:

1. Register the processor's name with INFOR CLIST, either ASCII or Fieldata, using a type 0 list (unknown control statement causes CLIST mode termination).

2. Become self-initializing or self-reinitializing. This may be done by explicitly setting all initialized variables to their initial value, or by reloading the main segment. The former method is recommended, unless the number of variables is excessive.

3. Check for potential reusability as indicated in PARTBL by SIR$ (S4 of PARTBL+13, see Figure 2-1), call REPRO$ or REPRM$ to read the next INFOR, if any, and reinitialize or terminate depending on which return is given by REPRO$/REPRM$.

Number one is described in detail in Volume 2-5.5. Number two simply means that any data areas which are expected to contain particular values at the beginning of the program must be initialized by a particular instruction sequence, rather than relying on static data initialization. Number three is elaborated as follows:

A. If S4 of PARTBL+13 > 0, call POSTPR, then quit.

B. Call REPRO$ or REPRM$, setting A0 as desired.

C. If absolute-eof-return, call POSTPR then quit.

D. If wrong-CLIST value, take action as defined for the particular processor in question.

E. If error return, treat same as error return from PREPRO or PREPRM.

F. If normal return, re-initialize processor (and SIR$, if used, by storing 0 in SIRP2$), and begin new processing operation (compilation, assembly, etc.).

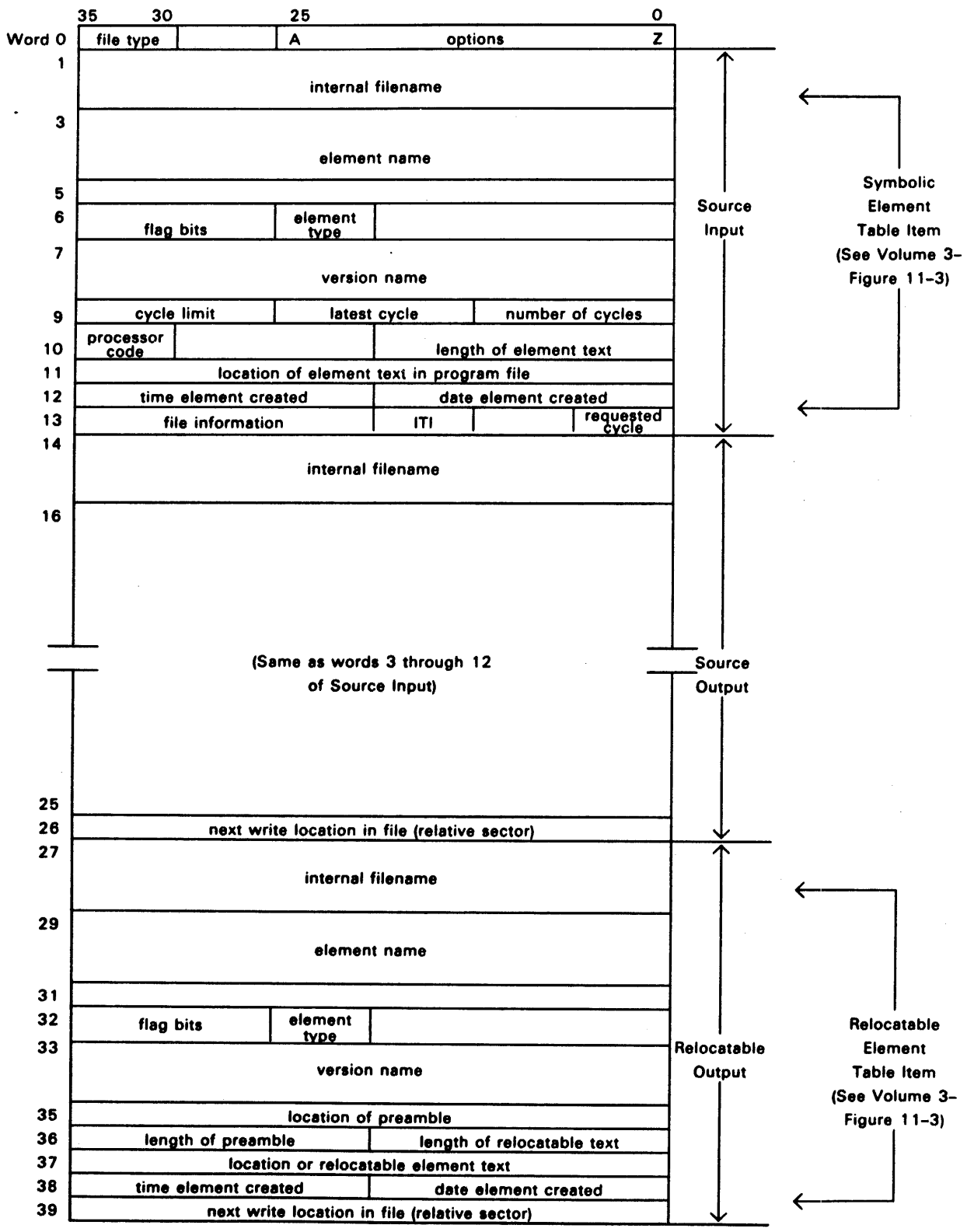


Figure 2-1. PARTBL Description

2.5.1.4.1. Reusable Processor Preprocessor Routine (REPRO$)

Purpose:

Reads INFOR from INFOR CLIST processor call card, provided INFOR$ has been set to zero.

Other purposes same as PREPRO.

Initial Conditions:

An INFOR CLIST must have been established, and on completion of SIR$, S4 of PARTBL+13 must be 0.

PARTBL must be externally defined and be at least 40 words long.

Format:

```
LA      A0,expected-CLIST-index-or-zero
LMJ     X11,REPRO$
        absolute-eof-return
        wrong-CLIST-index-return
        error return
        normal return
```

Description:

The error and normal returns are identical to those for PREPRO$. The absolute-eof-return indicates that an EOF condition has been encountered which cannot be bypassed; that is, a nontransparent control statement or @ENDX was encountered. In that case, the processor cannot reuse itself and must terminate. The wrong-CLIST-index return is optional for processors which have more than one item in their INFOR CLIST and only want to analyze one specific command automatically, which is specified by its CLIST index given in A0 prior to calling REPRO$. This return will never be taken if A0 is zero on entry to REPRO$. After the wrong-CLIST-index return, X11 will point to a reentry location if it is desired to continue with the preprocessor routine after all. (Note that the CLIST index encountered is found in S3 of INFOR$+1.) This may be done by the following instructions:

```
LMJ     X11,0,X11
        error return
        normal return
```

where the two returns are as for PREPRO$.

2.5.1.4.2. Reusable Processor Preprocessor Routine (REPRM$)

Purpose:

Reads INFOR from INFOR CLIST processor call card, provided INFOR$ has been set to zero.

Other purposes same as PREPRM.

Initial Conditions:

An INFOR CLIST must have been established, and on completion of SIR$, S4 of PARTBL+13 must be 0.

PARTBL must be externally defined and be at least 27 words long.

Format:

```
LA      A0,expected-CLIST-index-or-zero
LMJ     X11,REPRM$
        absolute-eof-return
        wrong-CLIST-index-return
        error return
        normal return
```

Description:

The error and normal returns are identical to those for PREPRM$. The absolute-eof return indicates that an EOF condition has been encountered which cannot be bypassed; that is, a nontransparent control statement or @ENDX was encountered. In that case, the processor cannot reuse itself and must terminate. The wrong-CLIST-index return is optional for processors which have more than one item in their INFOR CLIST and only want to analyze one specific command automatically, which is specified by its CLIST index given in A0 prior to calling REPRM$. This return will never be taken if A0 is zero on entry to REPRM$. After the wrong-CLIST-index return, X11 will point to a reentry location if it is desired to continue with the preprocessor routine after all. (Note that the CLIST index encountered is found in S3 of INFOR$+1.) This may be done by the following instructions:

```
LMJ     X11,0,X11
        error return
        normal return
```

where the two returns are as for PREPRM$.

2.5.1.5. Processor Field Retrieval (FLDGET)

Purpose:

This routine retrieves individual fields from the processor call card not normally processed by PREPRO or PREPRM.

Format:

```
        F66618          FORM        6,6,6,18
        F2466           FORM        24,6,6
```

the calling sequence is as follows:

```
L       A1,(F2466  0,l,f)
L       A0,(F6618  0,n,m,j)
LMJ     X11,FLDGETO$ (if using PREPRO$)

        or

LMJ     X11,FLDGETM$ (if using PREPRM$)
        error return
        normal return
```

where:

l – list number desired, currently 01 is only valid list number

f – field number desired

n – control bits

BIT 24 – if set field is to be used for output

– if not set field is to be used for input

BIT 25 – if set disallow tape for input

– if not set allow tape for input

BIT 26 – if set the routine will print any error messages

– if not set the routine will not print error messages

m – type of element (1–7); 070 for file only

j – address of 13 word data area

Description:

The information pertaining to the requested field is transferred from the INFOR table to the 13 word data area specified by *j*. Files are assigned if necessary and the element, if designated for input, is verified. A use name of the form 'F*xxyy* $' is attached to the file, where *xx* is the list number and *yy* is the field number. TPF$ is the default file.

It is possible to request either a file or an element by OR'ing 070 with the element type, i.e., 077 in field *m* indicates an omnibus element or a file. The request for an element has precedence over a request for a file.

When bit 24 of field *n* is set the routine does the following:

1. Assign file exclusively
2. Tape file not allowed
3. The element name, version, and type are stored in the 13 word data area

When bit 24 of field *n* is not set the routine does the following:

1. Tape file allowed, depending upon bit 25
2. Equipment code is stored in S4 of the first word of the 13 word data area.
3. An ER PFS$ is done on information supplied.
4. Element cycling is checked, relative cycling allowed.

When bit 26 is set, printing of the error message is suppressed when the field is not found and the first word of the 13 word data area is set negative.

The error return is taken when:

1. the file cannot be assigned correctly,
2. the input element cannot be found,
3. incorrectly coded field, and
4. the field specified cannot be found. (A0 will contain the same information SELT$ returned in A0 on a no find return).

2.5.2. Preprocessor Routine (PREPF$)

Purpose:

Generates the source input and source output fields of PARTBL (see Figure 2-1). Performs dynamic assigns of source input and output files. This routine is intended for processors which use files rather than elements.

Initial Conditions:

PARTBL must be externally defined and at least 27 decimal words in length.

Format:

```
LMJ     X11,PREPF$
        error return
        normal return
```

Description:

The information from the source input and source output parameters or the processor call statement, if given, is transfered to PARTBL. The options specified on the processor call statement governs how the information is placed in PARTBL.

no options
- if SI is given, the L option is assumed.
- if the SO field is given, an error message is printed.
- if both SI and SO fields are given, the files are assigned and no options are assumed.
- if neither SI nor SO fields are given, a normal return to the user is taken.

I option
- SI field is assumed to be the output file.
- if the SO field is given, an error message is output.
- if the SI and SO fields are not given, a normal return to the user is taken.

U option
- if the SI field is not given, an error message is printed and the error return taken.
- if the SO field is given, an error message is printed and the error return taken.

- the U option implies that the next higher f-cycle of the SI file is to be produced. PREPF$ assigns the next higher f-cycle. The name specified in the SI field cannot be a USE name.

It is important to remember that PREPF$ expects file names. If the user is not in DATA MODE then a period must be included in the file name.

2.5.3. INFOR Table Interface Routines (INFOR$)

NOTE:

When using the preprocessor routines, they must perform the first READ$ request and they must process the information from the processor control statement.

The INFOR table interface routine:

1. Reads the INFOR table.
2. Searches for a parameter subfield.
3. Retrieves a complete parameter.
4. Performs a dynamic @USE (see Volume 2-3.7.5).

Internal format (INFOR) is the table of information returned from the operand field of a processor call statement when a program is called for execution as a processor rather than by being named on an @XQT control statement. The INFOR is returned on the program's first request to READ$ (see Volume 2-5.2.1).

The INFOR call in contrast to @XQT is as follows:

@eltname,options operand-field

The operand-field may consist of many parameters, separated by commas, each of which may contain information of any kind, as long as it conforms to the syntax conventions of a file or element name. For example, an operand field of 23, REVISE/@7.,16/17 would create the following INFOR entries:

- Parameter 1 is treated as element name: '23 '
- Parameter 2 is treated as filename: 'REVISE '
  - read key: '@ '
  - write key: '7 '
- Parameter 3 is treated as element name: '16 '
  - version name: '17 '

The format of the INFOR table is flexible, with control words to indicate which components of a filename or element name are actually present (nonvoid). Figure 2-2 illustrates the INFOR table format. A set of routines is provided for reading and manipulating the INFOR table. These routines are described in following subsections.

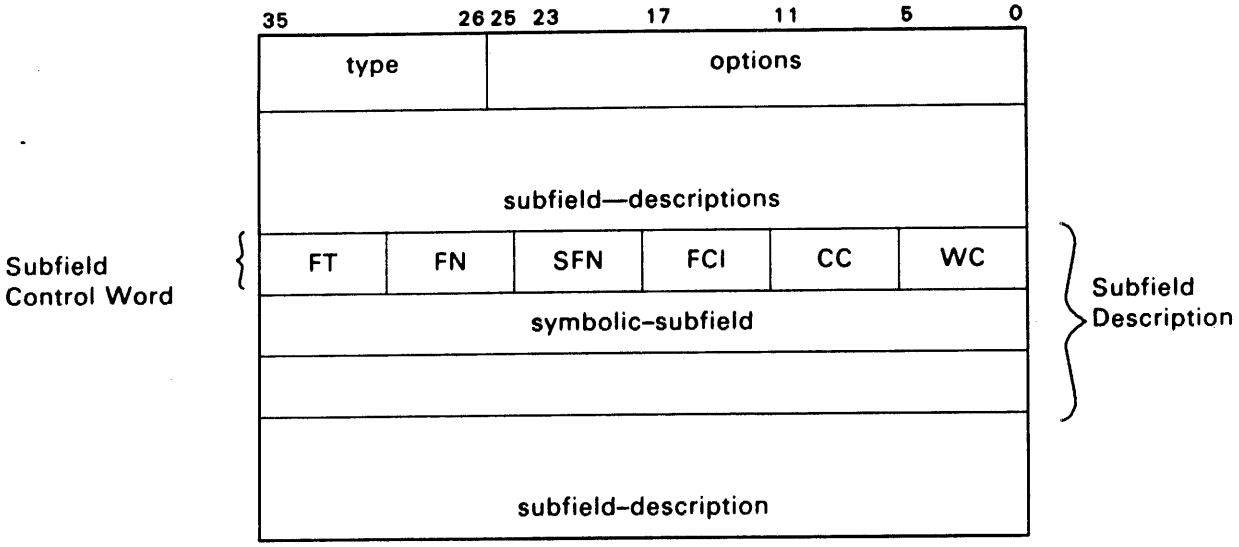


Figure 2-2. Format of INFOR Table

The fields of the INFOR table are:

type — The contents of type is right justified and specifies the control image (51_8 for processor; 100_8 for FURPUR).

options — A mask containing the specified options in master bit notation (bit 25 for A; bit 0 for Z).

A subfield-description represents each subfield specified on the control image. The subfield control word which is the first word of the subfield-description, has the following fields:

FT — Field type supplied by the following:

0 - options (or command) field
1 - specification field

FN — Field number within field type (numbered from one).

SFN — Subfield number within field (numbered from one).

FCI — File continuation indicator (used with element notation). Contains 075 octal if field was preceded by a period, otherwise zero.

CC — Specifies the number of characters in the last word of the symbolic subfield. The remaining characters are spaces.

WC — Specifies the number of words in the symbolic subfield.

The number of characters in the symbolic subfield is 6*(WC-1)+CC.

2.5.3.1. Element and File Notation

For element and file notation formats, see Figure 2-3. In file and element notation, if the (*) appears without a qualifier, the INFOR table contains a qualifier of twelve spaces. (see Volume 2-2.6) When the element name has the form:

```
.[N] [/[V]] [([C])]
```

the leading period causes the first subfield to contain 75_8 (Fieldata for period) in the FCI subfield. If this occurs in the command field of a control statement, a call for a user-defined processor in TPF$ is made. If this occurs in a specification field, the previous specification field is used (TPF$ is the first specification field).

2.5.3.2. Reading the INFOR Table

READ$ provides the processor an INFOR table as its first card images. The following bits in A0 control the reading of the INFOR table:

bit 31 - Image is part of the INFOR table

bit 30 - Another READ$ request is needed to complete the reading

bit 17-00 - Number of words transferred (normally 27)

2.5.3.3. Internal Format Routines

The outlines and functions available to INFOR$ are:

RINF$ - reads the INFOR table

SINF$ - searches for specification subfield

SELT$ - searches for a file or element notation

DUSE$ - performs a dynamic @USE (see Volume 2-3.7.5)

All of the code in INFOR$ is I-bank reentrant and quarter-word and third-word insensitive. INFOR$ uses registers A0 through A5, X11, R1, and R2.

In addition to the routine names, the following cells are externally defined:

INFOR$ - parameter supplied to RINF$

INFOR$ + 1 - contents of A0 after first READ$ performed by RINF$ (status bits, CLIST$ index, etc.)

FILE$ - edited filename from DUSE$ (8 words)

ELT$ - field description table from SELT$ (14 words)

2.5.3.4. Read INFOR Table (RINF$)

Purpose:

Reads the INFOR table into an area supplied by the caller.

Format:

```
L       A0, (word-count,starting-addr)
LMJ     X11,RINF$
        error return
        normal return
```

Description:

If INFOR$, an externally defined tag, is zero a read of INFOR is done. If INFOR$ is not zero, INFOR is assumed to have been read and is in the area described by INFOR$. Therefore on a reusability call INFOR$ must be set to zero in order to read INFOR.

Register A0 must contain the address in the modifier portion area and the length in the increment portion (length in the form $27n+1$). If no errors occur, the INFOR table is read into the specified area, followed by a zero word, and the normal return is taken.

When an error is encountered, register A1 contains the error type and register A0 contains the PRINT$ control word for the error message. The possible messages are given in Table 2-11.

Table 2-11. RINF$ Error Messages

Error Code	Error Message and Description
1_8	PROCESSOR CALL ERROR Data images were returned on the first call to READ$ indicating that no INFOR table was created for this program.
2_8	ABNORMAL RETURN FROM READ$ READ$ returned with an end-of-file or other status which indicates that this program has neither an INFOR table nor input data images.
3_8	TOO MANY SPECIFICATIONS The buffer area provided by the user is not large enough to hold the INFOR table.

2.5.3.5. Search Infor Table (SINF$)

Purpose:

Searches the INFOR table (previously read by RINF$, see 2.5.3.4) for a specification subfield.

Format:

```
L,U     A0,component-descriptor
LMJ     X11,SINF$
        no-find-return
        find-return
```

Description:

Register A0 must contain the specification subfield description (H1 of the control word).

The component descriptor is an octal value of ffsscc, where ff is the field number, beginning with 01, ss is the subfield number, beginning with 01, and cc is the component number with the components ranging from 01_8 to 10_8 as follows:

1_8	File qualifier (two-word entry)
2_8	Filename (two-word entry)
3_8	File cycle (one-word entry)
4_8	Read key (one-word entry)
5_8	Write key (one-word entry)
6_8	Element name (two-word entry)
7_8	Version name (two-word entry)
10_8	Element cycle (one-word entry)

To return the element name component of subfield 3, for example, the component descriptor would be 010306_8.

There is a special case where an element name is preceded by a period and given without a filename. This is normally interpreted to mean that the file for the present subfield is to be either TPF$ if it is the first subfield, or the same file as was used for the previous subfield. Following a find-return from SINF$ to get an element name component, the following test skips if this special file continuation case exists:

```
L,S4    A1,0,A2
TE,U    A2,'.'
```

If a file qualifier component of spaces is returned, a qualifier was implied by a leading asterisk (*) before the filename.

When a find-return occurs, the registers contain the following:

A0,A1	- Fieldata name (left-justified, space filled)
A2	- location of control word (S4 contains the FCI field; see Figure 2-2)
A3	- number of characters

2.5.3.6. Transfer to ELT$ Table From INFOR Table (SELT$)

Purpose:

Transfers a subfield of file or element notation from the INFOR table to the ELT$ table. A leading period (.) in element notation automatically causes file continuation; the element cycle is converted to binary.

Format:

```
F2466   FORM 24,6,6

        L       A0,(F2466 0,ff,ss)
                where:  ff is the field number desired
                        ss is the subfield number desired

        LMJ     X11,SELT$
                no-find-return
                find-return
```

NOTE:

The only field currently allowed is 01, if 00 is given a 01 is assumed. The specification of the field number is to accommodate future enhancements to the 1100 Series Executive System control statement interpreter.

Figure 2-3 describes the ELT$ table when a find-return occurs.

	S1	S2	S3	S4	S5	S6
ELT$+0	FQL	FNL	FCL	RKL	WKL	IQF
+1	ENL	EVL	ECL	CFN	ECC	BEC
+2	FQUAL (two words)					
+3						
+4	FNAME (two words)					
+5						
+6	FCYC					
+7	RKEY					
+8	WKEY					
+9	ENAME (two words)					
+11	EVER (two words)					
ELT$+13	ECYC					

Figure 2-3. Format of the ELT$ Table

The fields in the ELT$ table represent the following:

FQL	length (in characters) of file qualifier or 0
FNL	length (in characters) of filename or 0
FCL	length (in characters) of F–cycle or 0
RKL	length (in characters) of read key or 0
WKL	length (in characters) of write key or 0
IQF	implied qualifier flag
ENL	length (in characters) of element name or 0
EVL	length (in characters) of element version or 0
ECL	length (in characters) of element cycle or 0
CFN	file continuation field number or 0
ECC	element cycle signal character
BEC	binary element cycle
FQUAL	Fieldata file qualifier
FNAME	Fieldata filename
FCYC	Fieldata F–cycle
RKEY	Fieldata read key
WKEY	Fieldata write key
ENAME	Fieldata element name
EVER	Fieldata element version
ECYC	Fieldata element cycle

Description:

Since ELT$ is not cleared, the field lengths of the first two words should be interrogated to determine which Fieldata fields contain meaningful information. When defined, a Fieldata field is left–justified and space filled to one or two words.

A leading * sets the implied qualifier flag (IQF) to nonzero and file qualifier length (FQL) to zero.

If file continuation field number (CFN) is zero, then the file information was taken from the specification which was requested. If CFN is nonzero, then it contains the number of the field from which the file information was taken. This field is linked to the field being searched for by a sequence of element names with leading periods.

If the element cycle field is not specified, the element cycle signal character (ECC) equals a Fieldata minus sign (-) and the binary element cycle (BEC) equals zero. When it is specified, the numeric portion of the cycle is converted to binary and the result is placed in BEC. If the cycle is neither numeric nor signed numeric, or is greater than 63, the field ECC contains the Fieldata image E, indicating an error. Normally, ECC contains 0, Fieldata +, or Fieldata - when the first character of the cycle is numeric, +, or -, respectively.

Since ELT$ is not cleared, the character count must therefore be examined to determine which fields contain meaningful information.

When a no-find-return occurs, register A0 contains the address of the control word that stopped the search. If this word is zero, the rest of the INFOR table is empty.

As assembler procedure, ELT$, defines the field names in the ELT$ table (see Figure 2-3) as EQUF's.

2.5.3.7. Assign Attached Name to File Specified in INFOR Table (DUSE$)

Purpose:

Assigns an attached name to a file which was specified in the INFOR table by a CSF$ request naming a @USE control statement. DUSE$ uses the filename information in the ELT$ table and the attached name supplied by the programmer.

Format:

```
DL      A0,(attached-name)
LMJ     X11,DUSE$
        return
```

Description:

Registers A0 and A1 must contain the 12-character attached name, left-justified and space filled.

After the return, A0 contains the status bits from CSF$ (see Volume 2-4.10.1.1), and the 8-word area FILE$ contains the filename (edited, left-justified, and space filled). If a filename is not specified (FNL = 0), TPF$ is assumed.

2.5.4. Identification Line Routines (IDLINE$/IDONLY$)

IDLINE$/IDONLY$ is a routine that provides a standard identification line for 1100 Series System language and utility processors.

- Identification Line Format

  A standard processor identification line is generated in a processor supplied buffer called IDBUFF. All fields but the processor name and level field are generated. The element cycle information field is optional. All information is generated in Fieldata code.

  The format of the complete ID line is:

  PPPLLL RRRRRR MO/DD/YY HH:MM:SS (old,new)

where:

PPPLLL	=	processor name and level, 12 characters or less, supplied by the processor.
RRRRRR	=	System Relocatable Library level the processor is collected with. The form varies with the level number. For example, RL72R1, 73R1Q1.
MO/DD/YY	=	month/day/year at processor call.
HH:MM:SS	=	hour:minute:second at processor call.
(old,new)	=	optional (input,output) symbolic element cycle number.

- IDBUFF

The processor must supply a buffer with an externally defined label, IDBUFF. Prior to calling IDLINE$/IDONLY$ this buffer must be initialized with the name/level field (up to 12 characters) left-justified and the remainder of the buffer blank filled. This buffer must be at least 6 words long and up to 8 depending upon the name/level length and whether or not cycle information is requested.

The IDBUFF length is shown in Table 2-12.

Table 2-12. IDBUFF Length

Name and Level Length		
Call	1 Word	2 Words
IDLIN$/IDTIME$	7	8
IDONLY$/IDTOME$	6	7

A processor name and level > 12 characters may be used if the start of IDBUFF is defined as the word following the name and level.

- Register Usage:

IDLINE$/IDONLY$ uses registers A0-A5, X9, X11, R1-R3 (X9 is saved and restored), is reentrant and is not quarter/third-word sensitive.

2.5.4.1. IDLINE$

There are two entry points to the IDLINE$ routine, IDLIN$ and IDTIME$.

2.5.4.1.1. IDLIN$

Generates the complete ID line including symbolic element cycle information. The element cycle information is taken from PARTBL, which is filled in by the System Library routine PREPRM/PREPRO, so the call on PREPRM/PREPRO must precede the call on IDLIN$/IDTIME$.

The IDLIN$ calling sequence is:

```
      LMJ     X11,PREPRO$ . or PREPRM$
      J       ERROR       . error return from PREPRO$ or PREPRM$
      LMJ     X11,IDLIN$
      .
PF    FORM    12,6,18
      LA      A0,(PF  1,7,IDBUFF)
      ER      PRINT$
```

2.5.4.1.2. IDTIME$

This call is the equivalent of IDLIN$ except the caller furnishes the date (in register R2) and time (in register R3) in ER DATE$ format. IDTIME$ can be used by a processor that executes for a considerable time before it prints the ID line. In this case the actual date and time that the processor began execution would be saved and then passed to IDLINE$ in registers R2 and R3.

2.5.4.2. IDONLY$

There are two entry points to the IDONLY$ routine, IDONLY$ and IDTOME$.

2.5.4.2.1. IDONLY$

Generates the ID line without the symbolic element cycle information. IDONLY$ can be used by processors which do not create or update elements. An externally defined PARTBL is not required.

The IDONLY$ calling sequence is:

```
      LMJ     X11,IDONLY$
      .
PF    FORM    12,6,18
      LA      A0,(PF  1,6,IDBUFF)
      ER      PRINT$
```

2.5.4.2.2. IDTOME$

This call is the equivalent of IDONLY$ except the caller furnishes the date (in R2) and time (in R3) in ER DATE$ format.

2.5.5. Processor Scratch File Routine (GETPSF$)

As a matter of efficiency, a set of file names has been defined for common use by system processors, language processors, and user programs which require one or more scratch files. Based on the abbreviation PSF (Program Scratch File), these filenames are PSF$, PSF1$, PSF2$, etc. These files are generally to be assigned only once per run; the first program requiring a file will perform the

assignment and leave it assigned for subsequent use by other programs in the run. Moreover, a program should not assign the file by CSF$ if it is already assigned to the run; this can be determined by using the FITEM$ ER or by intercepting an IO 21 contingency. Each program is free to use as many PSF files as is needed; it is most economical to restrict the requirements to one or two. Since these files are used by the Collector and other complex programs, it is also desirable to assign them with a substantial granule maximum, such as 3000 tracks.

To facilitate the uniform assignment of these files, a library subroutine exists to obtain them in a manner consistent with the above requirements. This routine, GETPSF$, may be called as follows:

```
L,U     A0,n-1
LMJ     X11,GETPSF$
        error return
        normal return
```

where *n* is the number of files desired (in the range 1 to 10).

If the normal return is taken, the requested files are assigned to the run in FASTRAND format, track granularity, and with a granule maximum of 3000. For each file, a FACIL$ will be performed before attempting to assign the file, both to check equipment type for already assigned files and to prevent superfluous CSF$ requests.

An alternate entry point is available to assign only one specified file, as follows:

```
L,U     A0,n
LMJ     X11,GETPSFN$
        error return
        normal return
```

where *n* is the number of the file desired (*n* = 0 for PSF$, *n* = 1 for PSF1$, and so forth). The error conditions are the same as for the call to GETPSF$.

Upon normal return, A1 contains '$ $,', if one file was requested (0 if GETPSFN$) or '*n* $ *n* $,' ,where *n* is one less than the number of files requested (*n* if GETPSFN$). H1 of A1 is used for any @USE the processor may make on a PSF file (@USE *name*,PSF ('H1 of A1')).

When the error return is taken, A0 will contain an error code with auxiliary information in other registers depending on the nature of the error. The error codes are as follows:

A0 = 0	The input value in A0 is negative or exceeds 9. A1 contains the value.
A0 = 1	A file is assigned to the run which is not FASTRAND format. A1 points to the FACIL$ packet.
A0 = 2	A CSF$ request was rejected. A1 points to the FACIL$ packet and A2 contains the status bits returned by CSF$.

2.5.6. Source Input/Output Routine (SIR$)

The Source Input Routine (SIR$) is used by a processor to obtain the source images from the run stream, from a symbolic element in a program or element file, or from a SDF file. The routine can automatically merge corrections, list the corrections, and produce an updated symbolic element which is inserted into a program file. The symbolic element which contains the source input may be cycled; the desired cycle is specified in the processor call statement. The source input routine automatically passes to the processor only those images that pertain to the cycle requested.

SIR$ contains six user interface entries, OPNSR$, INISR$, GETSR$, GETAS$, GETNM$, and CLOSR$. These are described in following subparagraphs. The general capabilities of the source input routine are described below.

The information pertaining to source input and output elements or files (e.g., their names) is obtained from the respective entries in PARTBL. The information in PARTBL is usually produced by the PREPRM/PREPRO (see 2.5.1) or PREPF$ (see 2.5.2) routines from information specified on the processor call statement. The options that may be placed on a processor call statement that are used by SIR$ are presented in Table 2-13. Depending upon options used, one may direct the input and output of data in Fieldata or ASCII, input and check sequence numbers, insert symbolic elements, update and produce new cycles and list correction lines. The source-like correction statements used by SIR$ to modify symbolic elements are described in Volume 3-1.2.1.

The images processed and produced by SIR$ are in System Data Format (SDF). See Volume 3-11.2.3, for a description of SDF. SIR$ passes control images and label images of the SDF source language to the user, but does not count them as items when applying corrections.

When an updated or new source element is specified on the processor call statement, the corrected source is written to the program file. If a two-pass processor calls on SIR$ and no updated or new source element was specified on the processor call statement, the processor must provide a scratch file to hold the corrected source for the second pass. The internal filename of the scratch file must be in PARTBL+14,15, and PARTBL+16 must be set to zero prior to entering SIR$.

When a new element is being inserted into a program file from the run stream, SIR$ precedes the text images of the element with a label image.

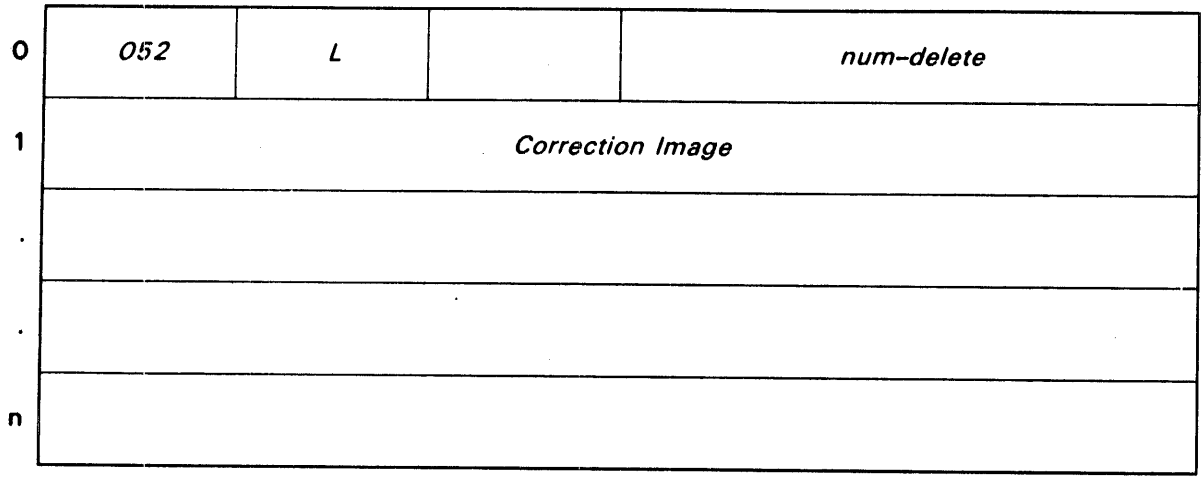
The format of this label control word and image is:

	S1	S2	S3		S6
0	*050*	*01*	*030*		*ASCII flag*
1	*SDFF*				

Word 0

S1 - *050* is the SDF label identifying this as an SDF element
S2 - indicates that a one-word image follows
S3 - the Fieldata character 'S' identifies this as a SIR$ created file or element
S6 - indicates the code type of the following text images, with a 1 indicating ASCII and a 0 indicating Fieldata.

If SIR$ applies corrections to the input, the output will contain *052* control images whose format is as follows:



where:

Word 0, S1

052 specifies that a correction image follows or that H2 of the *052* control word contains the number of images deleted before the next image on the last update.

Word 0, S2

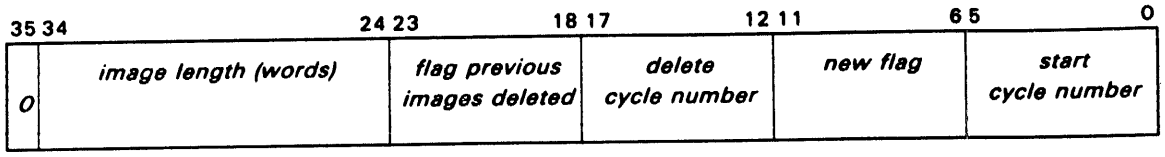
L specifies the length of the following correction image (*L* = 0 if this is a number of deleted images control image.)

Word 0, H2

num-delete Zero implies this is a correction statement control image. Nonzero implies H2 contains the number of images deleted before the next image on the last update. (This image is not returned to the user).

If the word specifies a correction statement image, the type of the image is the same as the current type of the input, Fieldata or ASCII. The "-n" correction image is placed in the output immediately after line n. The "-m,n", "-m-, and "-m,n" correction images are placed in the output immmediately after line m-1. The *052* control images in the SDF input element are discarded by SIR$ on its first pass.

A control word precedes each data image in the element. The format of this control word is:



S6 is the element cycle at which this image was added, and S4 is the cycle at which it was deleted. If the image was added on the latest cycle (after cycle 0), the new flag is set, so the calling processor can mark the line as "NEW". If this is not a new image, and any previous images were deleted on this cycle, S3 is nonzero. (See Volume 3-11.2.3.3.)

When the source input is from an element file (see Volume 3-11.2.2), the tape label block is read prior to entering SIR$, which then treats it as a program-file element. After SIR$ is closed, the tape is positioned in front of the new element label block.

2.5.6.1. SIR$ Control Options

Table 2-13 contains a list of those options used by the source input routine to control the input and output of the source language elements. Most language processors (FOR, ASM, COB, ALG, and so forth) supplied by Sperry Univac use the source input routine to obtain their input. Therefore, the listed options are generally applicable to language processors.

Table 2-13. Source Input Routine Options

Option Character	Description
G	Input is compressed symbolic in columns 1-80 of the card deck.
H	Input contains sequence numbers in columns 73-80 of the symbolic images.
I	Insert a new symbolic element into the program file.
J	Input contains compressed symbolic images in columns 1-72 of the cards and sequence numbers in columns 73-80. These sequence numbers are not checked by the K option.
K	Check sequence numbers in columns 73-80 of the symbolic images (valid only with H option).
P	Output symbolic element in Fieldata. (Compare with Q.)

Table 2-13. Source Input Routine Options (continued)

Option Character	Description
Q	Output symbolic element in ASCII. (If neither P nor Q is specified, code type of input element, if any, is used; otherwise, P is assumed.) If both P and Q are specified, output symbolic element with mixed images, Fieldata and ASCII (only if mixed mode code is turned on (NM = 1), otherwise output is the same as input type.)
U	Update and produce a new cycle of the symbolic element.
W	List correction lines.

2.5.6.2. Open Source (OPNSR$)

Purpose:

Initializes SIR$ to allow input/output.

Format:

```
LMJ     X11,OPNSR$
        error return
        normal return
```

Description:

(Either OPNSR$ or INISR$ may be called for initialization but not both on the same pass.)

Provides first or second pass initialization functions for the Get Source routines. The System Data Format Input routine (SDFI) is opened if the source input is from a program file or tape. The System Data Format Output routine (SDFO) is opened if the source output is specified.

If input is from an element and S3 of the label control word is nonzero and not the Fieldata character 'S', the message NOT SIR TYPE FILE will be printed and all cycling information in the control words will be ignored.

The error return is taken when:

1. SDFI was unable to open source input. A5 contains the I/O error status code from SDFI.
2. An attempt was made to process an invalid SDF file or element. A5 = 0. The associated error message is:

   'INVALID SDF LABEL WORD *w*'

   where *w* is the first two words of the element or file printed in Fieldata.

2.5.6.3. Initialize Source (INISR$)

Purpose:

Initializes SIR$ to allow input/output.

Format:

```
L       A4,<initialization-word>
LMJ     X11,INISR$
        error return
        normal return
```

Parameter:

initialization-word – Word used by SIR$ to specify which control images are not wanted by the calling processor and whether or not to create output to a scratch file if there are no corrections.

Description:

This entry point performs the same functions as OPNSR$ as well as initializing further information for SIR$. On entry, bits 0-31 of A4 specify which control images are not wanted by the calling processor. If bit *n* is set, (*n* is the octal bit position, right-most bit = 0), then control image *n* +040 will not be returned to the caller on a GET image call.

Example:

A4 = 043 on a call to INISR$ would prevent control image types 040, 041, and 045 from being returned to the caller. If bit 35 is set (A4 is negative) SIR$ will check for the following conditions to exist:

1. Output is to a file (PARTBL+16=0),
2. input is not from tape, and
3. there are no corrections

If these conditions exist, SIR$ will not do output to the processor's scratch file on the first pass. On subsequent passes SIR$ will reread the input element. If these conditions are met SIR$ will do approximately 75 percent fewer IO$'s on the first pass than would be done by calling OPNSR$.

Example:

```
@ASM,S <non-tape-si-element>,<ro-element>
@eof
```

This example would not create the unused output to the processor scratch file if the assembler had set A4 negative and called INISR$ instead of OPNSR$.

NOTE:

A processor whose purpose is to create an output file specified in the SO portion of PARTBL should not call INISR$ with A4 negative. Example: the @DATA processor.

2.5.6.4. Get Source Image in Fieldata (GETSR$)

Purpose:

Gets a source input image in Fieldata, and if source output is specified, generates a new source element image.

Format:

```
L       A0,(buffer-length, buffer-addr)
LMJ     X11,GETSR$
        error return
        EOF return
        normal return
```

Parameters:

buffer-length – Length of the buffer into which the images are to be read.

(If the user's buffer is longer than the image returned, the buffer will be blank-filled to the end. If the user's buffer is shorter than the image returned, only the part of the image that will fit in the buffer is returned – the last part of the image is lost.)

buffer-addr – Address of the buffer.

Description:

An image is transferred to the caller's buffer. GETSR$ will get the next image upon request. The user buffer will be spaced filled if the image returned is smaller than the buffer. Truncation will occur if the image exceeds the buffer size.

The error return is taken when either an I/O (unrecoverable) error occurs, a correction image sequence error occurs, or a partial line correction error occurs. In case of I/O error, A5 contains the I/O error status code. In case of sequence or partial line correction error, A5=0 and A0 contains a print control word for the image in error. The caller must do an ER PRINT$ to print the error message (see Volume 3-1.2.6).

The EOF return is taken when a SDF EOF image is encountered in the source input element or when READ$ gives an EOF return when reading images from the run stream.

The following information is returned to the caller when the normal return exit is taken:

A0 – (S6) == 0 (image returned is Fieldata)

A1 – SDF image control word. If A1 (bit 35) == 1 an SDF control image is being passed to the caller, A2 and A4 are undefined.

A2 – (S1) Always contains a Fieldata space (05).

(S2) Fieldata '#' symbol (03) if the sequence numbers in columns 73-80 are not sequential, otherwise a Fieldata space (05). Applies only with the H and K options.

(S3) Fieldata '*' symbol (050) if the image came from the run stream, otherwise a Fieldata space (05).

(H2) 3-digit Fieldata cycle number for the image.

A3 – Contains one of the following in Fieldata:

1. The characters 'NEW ' if this is a new image,
2. The characters '*xxxxxx*' where *xxxxxx* is a five or six digit decimal number left-justified space filled (preceded by a minus symbol if five digits) indicating the number of images deleted prior to this image.
3. Six space characters.

A4 – The current line number within the source input element. A4 = 0 when a new image from the run stream is passed to the caller. The line number for each image is the same for all passes.

2.5.6.5. Get Source Image in ASCII (GETAS$)

Purpose:

Gets a source input image in ASCII for the user and, if source output is specified, generates a new source element.

Format:

```
L       A0,(buffer-length,buffer-addr)
LMJ     X11,GETAS$
        error return
        EOF return
        normal return
```

Description:

The format and operation of the GETAS$ request are identical to those for the GETSR$ request (see 2.5.6.5) except that the images are returned in ASCII and the contents of the registers returned are altered as follows:

A0 – (S6) = 1 (image returned is ASCII)

A2 – (Q1) Contains the same information as S2 except in ASCII ('#' or ' ' = 043 or 040)

(Q2) Contains the same information as S3 except in ASCII ('*' or ' ' = 052 or 040)

(H2) Contains 2-digit ASCII cycle number for the image.

A3 – Contains either 'NEW ', '*xxxx*', or ' ' with *xxxx* being a three or four digit ASCII number similar to that for GETSR$.

2.5.6.6. Get Source Image In Native Mode (GETNM$)

Purpose:

Gets a source input image in whatever type it is input and if source output is specified, generates a new source element image.

Format:

```
L       A0,(buffer-length,buffer-addr)
LMJ     X11,GETNM$
        error return
        EOF return
        normal return
```

Description:

The format and operation of the GETNM$ request are identical to those for the GETSR$ request (see 2.5.6.5) except the images are returned as the type they were input, either from the runstream or SDF input. The registers are returned as described under the GETSR$ request if the image is returned in Fieldata or are returned as described under the GETAS$ request if the image is returned in ASCII. The contents of A0 may be used to determine the mode.

This entry point is available at site option. If this code is active the output may be of mixed type if the P and Q options were both specified on the processor call (see Volume 2-3.9). Fieldata and ASCII images may be intermixed in the output element with the appropriate SDF Fieldata/ASCII switch control words (0420...0/1) inserted.

2.5.6.7. Close Source (CLOSR$)

Purpose:

Closes the source input file, and if output is specified, closes the source output file.

Format:

```
LMJ     X11,CLOSR$
        error return
        normal return
```

Description:

Provides first or second pass termination functions for the Get Source routines. The System Data Format Input routine (SDFI) is closed if source output was specified. If first pass, and source output is an element, an entry is made in the source output program file through the Program File Input routine (PFI$).

The error return is taken when either SDFO is unable to close source or PFI$ gives an error return to SIR$. In the case of SDFO error, A5 contains the I/O error status code. In the case of PFI$ error, A5 = 0 and A2 contains the PFI$ status code (see Volume 3-Appendix B).

2.5.6.8. SIR$ Externalized Labels

SIR$ externalized labels are as follows:

- SIRIB$ – start of 448–word input buffer,
- SIROB$ – start of 448–word output buffer, and
- SIRP2$ – set to one in CLOSR$ to identify a second pass call to OPNSR$, INISR$, GETSR$, GETAS$, and GETNM$.

The user may not use SIRIB$ or SIROB$ buffers between calls to OPNSR$ or INISR$ and CLOSR$.

2.5.6.9. SIR$ Multipass Capability

SIR$ allows many passes to be made over the input element, if a scratch file or output element is made available in PARTBL+14,15 (except as noted under INISR$(2.5.6.4)). For reusable processors, SIR$ must therefore be told when to initialize itself to process the element designated on the next processor call statement read in by INFOR CLIST. This is done by storing zero into the externally defined cell SIRP2$ in SIR$. No other action is required.

2.5.6.10. Compressed Symbolic Elements

In order to minimize the number of cards required to contain a symbolic element, the FURPUR processor (see Volume 3–Section 4) can compress strings of blanks in symbolic images before punching the element. The source input routine can expand the compressed images on input.

A compressed symbolic image card deck is produced when the appropriate options are used on the FURPUR @PCH control statement (see Volume 3–4.2.12). The source input routine converts compressed card image decks to SDF images upon initial input when the appropriate options are used on the processor control statement (see Volume 2–3.9.).

The first card punched is an @ELT control statement (see Volume 3–5.2) with the appropriate options. Following the @ELT control statement are the cards which contain the compressed symbolic images.

The compressed image consists of a stream of characters in the following format:

xccc...cyxccc...cz

where:

x Number of characters C ($1 \leq x \leq 37_8$)

y 40_8 + Number of blanks ($41_8 < y \leq 77_8$)

z 40_8 = End–of–image

The number of characters in a string is limited to 37_8; the number of blanks is limited to 36_8. If either is larger, a new x or y is initiated.

In addition to the x, y, and z characters, two other special characters are used:

41_8 character – indicates the end–of–images in this element

0_8 character – A special character used in column 80 if a new x would begin in column 80. The x is moved to the next physical card and the 0_8 is placed in column 80.

The compressed images immediately follow one another on the physical card and continue to the next card when the end-of-card is reached. The punch routine begins each physical card with an x, y, or z by breaking an x string at the end of the card, and starting a new string on the next card. This guarantees a nonzero character in the first character position of the card. A compressed blank image would be represented by a 40_8 character. The physical card may contain compressed image characters in columns 1-80.

Compressed images are not retained in the program file. The source input routine expands the images, and stores them in the program file in SDF format.

Compressed image symbolic input is site dependent; for further information see the SPERRY UNIVAC 1100 Series Operating System Installation Reference (UP-8486).

2.5.7. Program File Basic Service Package (BSP$)

The Basic Service Package (BSP$) is an interface routine between the user and a program file Table of Contents. BSP$ is available in a common bank (PIRCB$) or as a relocatable element in SYSLIB. The user may, through selective calls to the BSP$, perform the following functions:

1. Read the file table index into a buffer
2. Read a selected program file table into a buffer.
3. Search a selected program file table for a specific entry.
4. Delete a specific entry in a program file table.
5. Locate a program file table entry from its sequence number in the table.
6. Add an entry to a program file table.
7. Write the last entry referenced.
8. Write a program file table.
9. Write the file table index.

Any file assigned and not previously written in may be prepared as a program file by requesting number (1) above. If not previously created, a table is created upon the first call to read it. The table is not written to the file until number (9) above is requested.

Following are some rules concerning the use of BSP$:

1. Reference PIRCB$ (section 3) for discription of common bank features.
2. The user attempting to create a program file in a file area previously written in will receive an error return, provided the first word encountered is not '**PF**' or the sector 0 is not all zeros.
3. Each table (except for the element table) on mass storage will initially begin at the system defined sector or at a greater sector if a previous table now occupies the defined sector.

4. The Table of Contents (TOC) is normally 1792 sectors in length.

5. Tables must be written back from the main storage area assigned, if the area is to be used by others. A diagnostic is given if a table cannot accept another item.

6. The user must provide a 34 word area (using BSP$) or a 47 word area (using PIRCB$) for the File Table Index (FTI) and main storage buffers for each table read. A request to read the FTI must be made before any other BSP$ functions are used. The FTI must be in core until all BSP functions that may be necessary have been executed.

7. No buffer may be smaller than 196 words, or an error diagnostic will result.

8. The File Table Index in main storage will be used by BSP$ to contain necessary information about the file.

9. Each table that has been altered in main storage requires a final call on the Write Program File subroutine in order to ensure all information changed reflected on mass storage.

10. The major register set is assumed available for BSP$ subroutines, although any major registers used are saved and restored.

The following descriptions of the BSP$ functions include the various entry methods. Entry points in the common bank are prefixed with the letter B .

The entries will be given in the following order:

(1) Auto switch (for systems which can base more than one bank simultaneously)

(2) Common Bank LIJ

(3) Relocatable Collection Method

2.5.7.1. Read File Table Index

1. Calling Sequence

```
        L,U     A0,FCT

(1)     LMJ     X11,BRFTI$-1

(2)     LXI,U   X11,PIRCB$
        LIJ     X11,BRFTI$

(3)     LMJ     X11,RFTI$
                error return
                normal return
```

FCT = Address of 34 or 47 word buffer; word 0, 1 contains the internal filename.

a. Error Return

A0 = 02	This file may not be used as a program file.
A0 = 7700000000xx	xx = Status from IOW$ on error return.

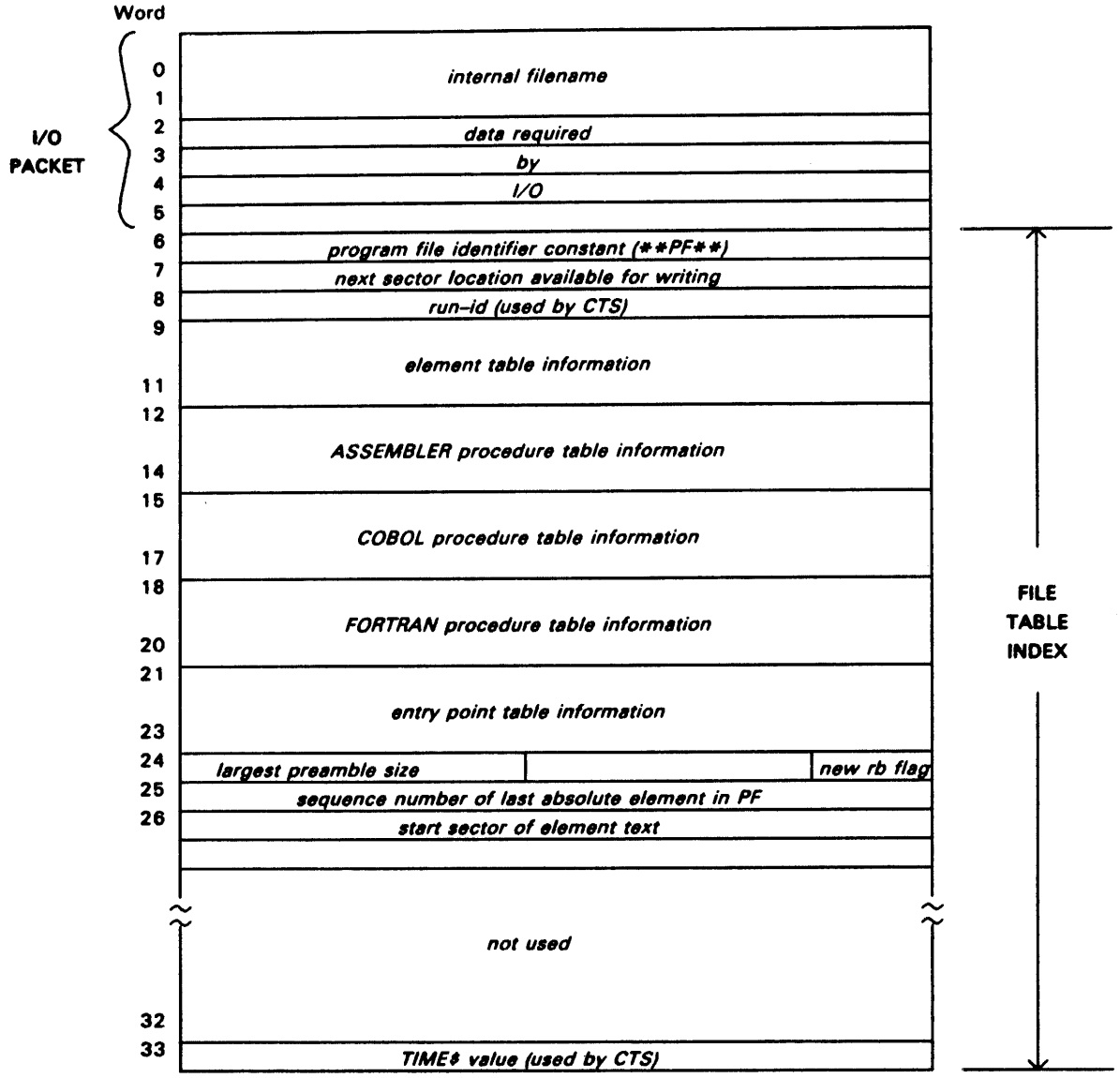
b. Normal Return

Words FCT+6 will contain the File Table Index

Words FCT+0 to FCT+5 contain the packet used last for I/O. If an 05 I/O status was returned on the read, the FCT, words 6-33, is initialized as an empty file.

2. Registers Used: A1,A2,A4,A5

ARRANGEMENT OF THE FILE CONTROL TABLE AFTER THE FILE TABLE INDEX IS READ



2.5.7.2. Read Program File Table

BRPFET$ – Read Program File Element Table
RPFET$

BRPFAPT$ – Read Program File Assembler Procedure Table
RPFAPT$

BRPFCPT$ - Read Program File COBOL Procedure Table
RPFCPT$

BRPFFPT$ - Read Program File FORTRAN Procedure Table
RPFFPT$

BRPFEPT$ - Read Program File Entry Point Table
RPFEPT$

1. Calling Sequence:

```
        L,U     A0,FCT
        L       A1,(buffer,length)

(1)     LMJ     X11,BRPFxxT$-1

(2)     LXI,U   X11,PIRCB$
        LIJ     X11,BRPFxxT$

(3)     LMJ     X11,RPFxxT$
                error return
                normal return
```

Buffer is the starting address of the buffer to be used for the table.

Length is the length in words of the area reserved for the table. It should be a multiple of 28 words. To minimize read before writes, it should be of the size 28(5+np) words, where p is the number of sectors in a physical disc record and n is some postive integer. The size can not be less than 196 words.

a. Error Return

A0 = 024	User buffer too small. (There must be at least 7 sectors, 196 words, available.)
A0 = 044	No room in file to create this table.
A0 = 012	User does not have FTI in main storage.
A0 = 7700000000xx	I/O Error.

b. Normal Return

Buffer is filled with the requested table, if the table does not exist the buffer is initialized.

2. Registers Used: A0 - A5

2.5.7.3. Search Table for Requested Item

BETIS$ - Element Table Item Search
ETIS$

BAPTIS$ - Assembler Procedure Table Item Search
APTIS$

BCPTIS$ - COBOL Procedure Table Item Search
CPTIS$

BFPTIS$ - FORTRAN Procedure Table Item Search
FPTIS$

BEPTIS$ - Entry Point Table Item Search
EPTIS$

1. Calling Sequence:

```
L,U     A0,FCT
L,U     A1,SPKT          SPKT = Address of SEARCH Packet
```

(1) `LMJ     X11,BxxIS$-1`

(2)
```
LXI,U   X11,PIRCB$
LIJ     X11,BxxIS$
```

(3)
```
LMJ     X11,xxIS$
        error or no find
        normal return
```

a. Error Return

A0 = 012 — No FTI

= 022 — Designated table not in main storage

= 7700000000xx — I/O Error

b. No Find

A0 = 001 Pointer table

= 011 Pointer link

= 021 Type link

= 041 Version link

A1 = 0

A2 = SEQNBR of last item referenced

c. Normal Return

A0 = ITEM Location

A1 = SEQNBR

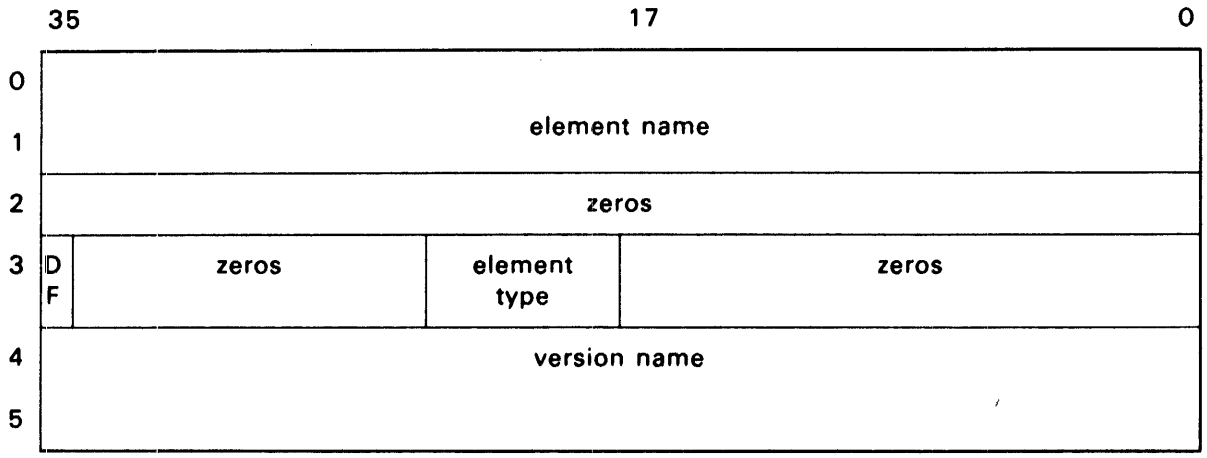
A2 - H1 = 001 Pointer Table
= 011 Pointer Link
= 021 Type Link
= 041 Version Link

H2 = SEQNBR of pointing item

2. Registers Used: A0 - A5

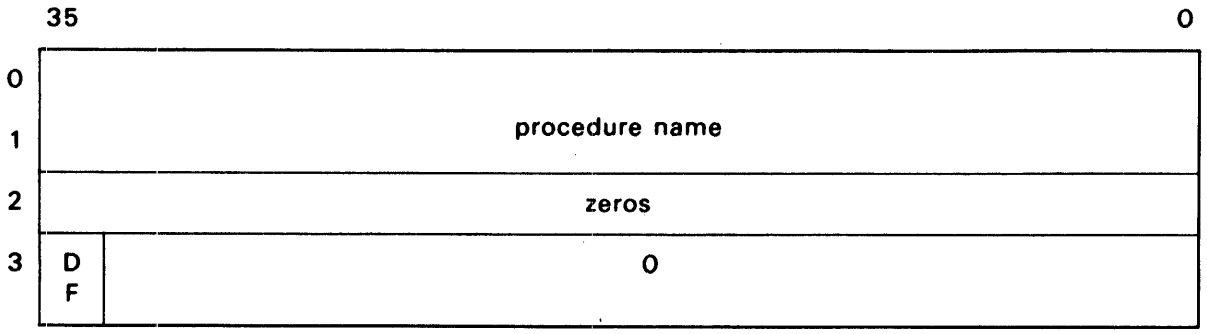
Packet Formats for Table Item Search:

1. Element Table Item Search Packet

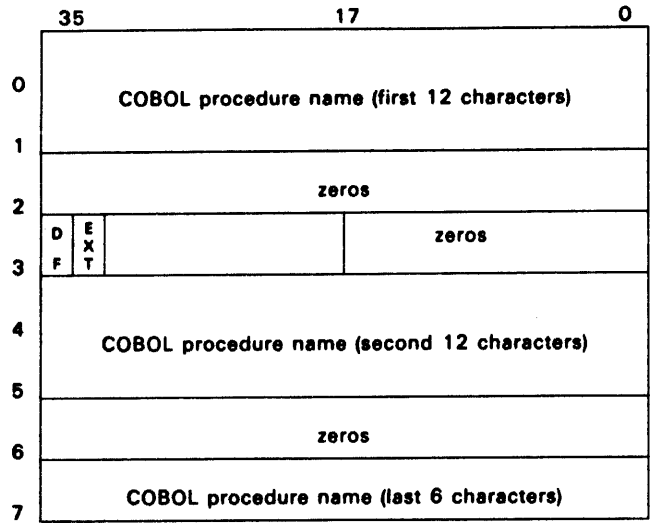


Element Type Code 1 may be used for any symbolic element. Search will consider all type codes less than 5 as symbolic elements.

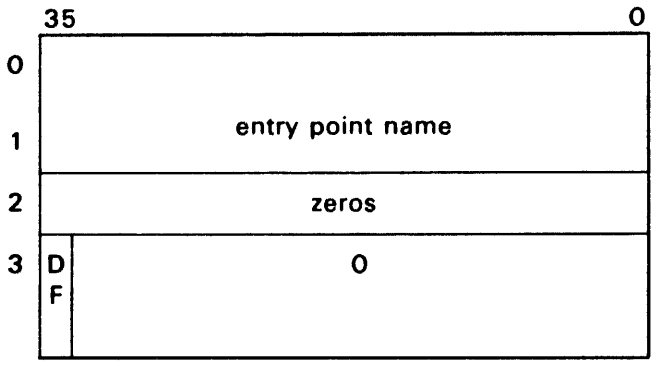
2. Assembler and FORTRAN Procedure Table Item Search Packet



3. COBOL Procedure Table Item Search Packet



4. Entry Point Table Item Search Packet



2.5.7.4. Delete Item From Requested Table

BETID$ – Element Table Item Delete
ETID$

BAPTID$ – Assembler Procedure Table Item Delete
APTID$

BCPTID$ – COBOL Procedure Table Item Delete
CPTID$

BFPTID$ – FORTRAN Procedure Table Item Delete
FPTID$

BEPTID$ – Entry Point Table Item Delete
EPTID$

1. Calling Sequence:

```
        L,U     A0,FCT
        L,U     A1,SPKT  (address of delete packet)

(1)     LMJ     X11,BxxID$-1

(2)     LXI,U   X11,PIRCB$
        LIJ     X11,BxxID$

(3)     LMJ     X11,xxID$
                error return
                normal return
```

a. Error Return

A0 = 012	FTI not in main storage
A0 = 022	Designated table not in main storage
A0 = 7700000000xx	I/O Error
A0 ≠ Above	Item could not be located

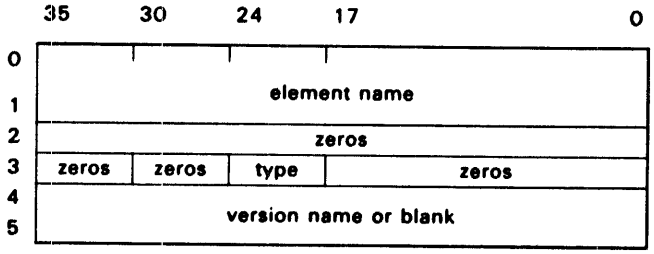
b. Normal Return

Requested item has been deleted from table.

2. Registers Used: A0 – A5

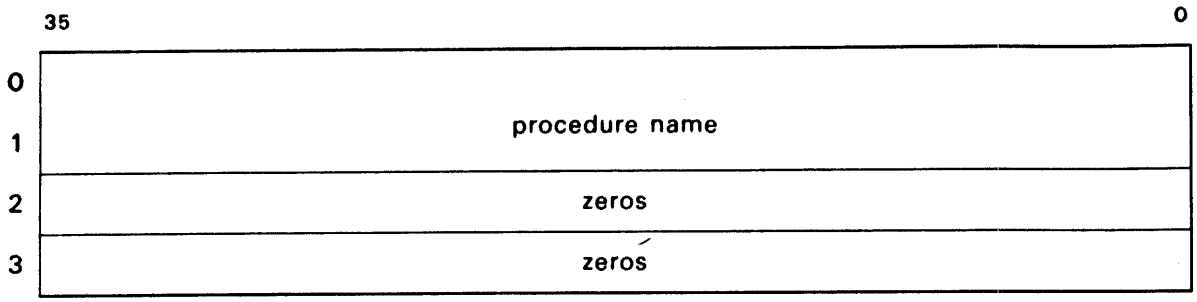
Packet Formats for Table Item Delete:

1. Element Table Item

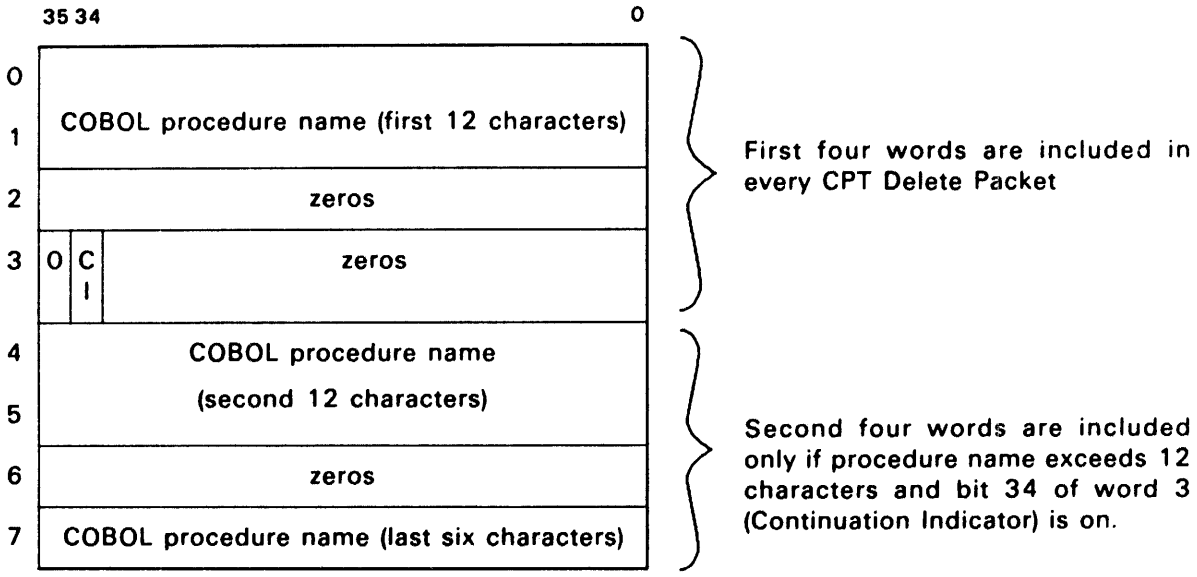


Element type 1 may be used for deleting procedures of types 2, 3, or 4.

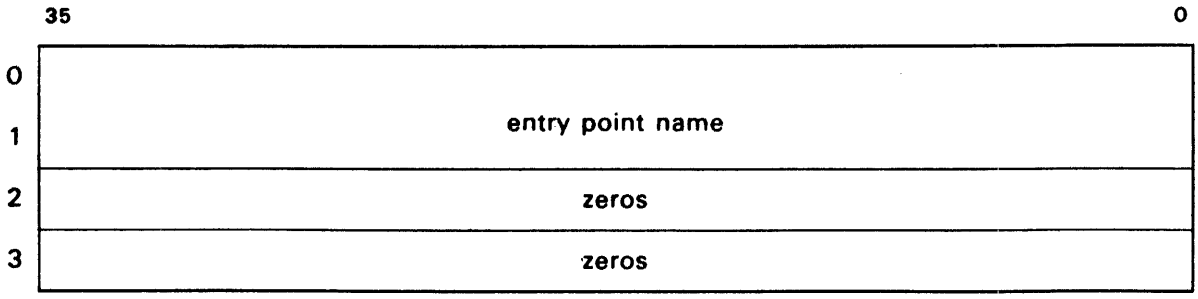
2. Assembler or FORTRAN Procedure Table Item Delete packet



3. COBOL Procedure Table Item Delete Packet



4. Entry Point Table Item Delete Packet



2.5.7.5. Entry Look-Up By Number

BETNL$ - Element Table Number Lookup
ETNL$

BAPTNL$ - Assembler Procedure Table Number Lookup
APTNL$

BCPTNL$ - COBOL Procedure Table Number Lookup
CPTNL$

BFPTNL$ - FORTRAN Procedure Table Number Lookup
FPTNL$

BEPTNL$ - Entry Point Table Number Lookup
EPTNL$

1. Calling Sequence:

```
L,U     A0,FCT
L       A1,SEQNBR
```

(1)
```
LMJ     X11,BxxNL$-1
```

(2)
```
LXI,U   X11,PIRCB$
LIJ     X11,BxxNL$
```

(3)
```
LMJ     X11,xxNL$
        error return
        normal return
```

a. Error Return

A0	= 014	Out of range
	= 012	FTI not in main storage
	= 022	Table not in main storage
	= 7700000000xx	I/O Error

b. Normal Return

A0 = Location of item

2. Registers Used: A0 - A5

2.5.7.6. Add Item to Requested Table

BETIA$ - Element Table Item Add
ETIA$

BAPTIA$ - Assembler Procedure Table Item Add
APTIA$

BCPTIA$ - COBOL Procedure Table Item Add
CPTIA$

BFPTIA$ - FORTRAN Procedure Table Add
FPTIA$

BEPTIA$ - Entry Point Table Item Add
EPTIA$

1. Calling Sequence:

```
L,U     A0,FCT
L,U     A1,SPKT  (address of add packet)

(1) LMJ     X11,BxxIA$-1

(2) LXI,U   PIRCB$
    LIJ     X11,BxxIA$

(3) LMJ     X11,xxIA$
            error return
            normal return
```

a. Error Return

A0 = 012	FTI not in main storage
A0 = 022	Designated table not in main storage
A0 = 7700000000xx	I/O Error
A0 = 044	No room for item

b. Normal Return

Requested item has been added to designated table. A1 has new SEQNBR. If the item being added is for the element table, and the item describes a relocatable element, the pointers to the entry point table in the File Control Table are cleared, effectively destroying the entry point table when the File Table Index is written back into the program file.

2. Registers Used: A0 - A5

Packet Formats for Table Item Add:

1. Symbolic or Procedure Element Add to Element Table

0	element name (Fieldata L.J.S.F.)		
1			
2	not used		
3	flag-bits	type	not used
4	version name or blanks (Fieldata L.J.S.F.)		
5			
6	cycle limit	latest cycle number	current number of cycles
7	sub-type	zero	length of element text
8	location of element text on mass storage		
9	time element added		date element added (or 0)

Element Type will be: 01 If Symbolic Element

02 If Assembler Procedure Element

03 If COBOL Procedure Element

04 If FORTRAN Procedure Element

Used only by Procedure Definition Processor

Element sub-types: See 2.1.6.

2. Relocatable Element Add to Element Table

0 1	element name (Fieldata L.J.S.F.)		
2	not used		
3	flag-bits	type = 5	zeros
4 5	version name or blanks (Fieldata L.J.S.F.)		
6	location of preamble		
7	length of preamble		length of relocatable text
8	location of element text on mass storage		
9	time element added (or 0)		date element added (or 0)

Length of element is total of 'length of preamble' and 'length of relocatable text' in sectors.

3. Absolute Element Add to Element Table

0 1	element name (Fieldata L.J.S.F.)		
2	not used		
3	flag bits	type = 6	zeros
4 5	version name or blanks		
6	bank information		
7	zeros		length of the element text
8	location of the element on mass storage		
9	time element added (or 0)		date element added (or 0)

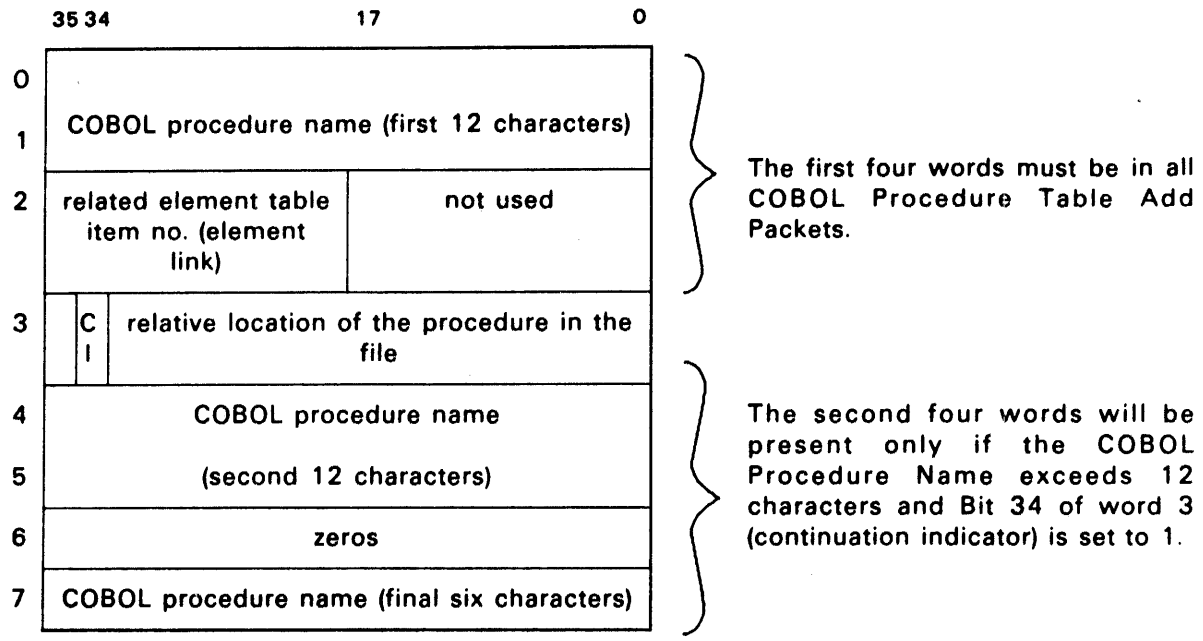
4. Omnibus Element Add to Element Table

0 1	element name (Fieldata L.J.S.F.)		
2	not used		
3	flag-bits	type = 7	zeros
4 5	version name or blanks (Fieldata L.J.S.F.)		
6	not used		
7	sub-type	not used	
8	location of element text on mass storage		
9	time element added (or 0)	date element added (or 0)	

5. Assembler Procedure or FORTRAN Procedure Item

0 1	procedure name	
2	related element table item number	not used
3	relative location of the procedure in the file	

6. COBOL Procedure Item Add to COBOL Procedure Table



7. Entry Point Table Item Add

0 1	entry point name	
2	related element table item number	not used
3	zeros	not used

NOTE:

If time and date are zero, BSP$ will insert the current time and date upon entry into a program file.

2.5.7.7. Write Last Item Referenced

BPTEWT$ – Part Table Element Table Write
PTEWT$

BPTATWT$ – Part Table Assembler Procedure Table Write
PTATWT$

BPTCTWT$ – Part Table COBOL Procedure Table Write
PTCTWT$

BPTFTWT$ – Part Table FORTRAN Procedure Table Write
PTFTWT$

BPTETWT$ – Part Table Entry Point Table Write
PTETWT$

1. Calling Sequence:

```
        L,U     A0,FCT

(1)     LMJ     X11,BPTxxWT$-1

(2)     LXI,U   X11,PIRCB$
        LIJ     X11,BPTxxWT$

(3)     LMJ     X11,PTxxWT$
                error return
                normal return
```

a. Error Return

A0	= 024	User error
	= 7700000000xx	I/O Error
	= 012	No FTI
	= 022	Table not in main storage

b. Normal Return

Last item referenced written.

2. Registers Used A0 – A5

2.5.7.8. Write Requested Table Back to Mass Storage

BWPFET$ – Write Program File Element Table
WPFET$

BWPFAPT$ – Write Program File Assembler Procedure Table
WPFAPT$

BWPFCPT$ – Write Program File COBOL Procedure Table
WPFCPT$

BWPFFPT$ – Write Program File FORTRAN Procedure Table
WPFFPT$

BWPFEPT$ – Write Program File Entry Point Table
WPFEPT$

1. Calling Sequence:

```
        L,U     A0,FCT

(1)     LMJ     X11,BWPFxx$-1

(2)     LXI,U   X11,PIRCB$
        LIJ     X11,BWPFxx$

(3)     LMJ     X11,WPFxx$
                error return
                normal return
```

The change indicator in S3 of word 0 of the File Table Index 3 - word item for the particular table to be written must be set, for the table to actually get written back to the program file. This will ordinarily have been done if the user has modified the table by doing an item add call (2.5.7.6) to BSP$.

a. Error Returns

A0	= 012	FTI not in main storage
	= 022	Requested table not in main storage
	= 7700000000xx	I/O Error

b. Normal Return

Designated Table, Pointer Table, and last segment left in main storage if changed, are written to mass storage.

2. Registers Used: A0 - A5

2.5.7.9. Write File Table Index

1. Calling Sequence:

```
        L,U     A0,FCT

(1)     LMJ     X11,BWFTI$-1

(2)     LXI,U   X11,PIRCB$
        LIJ     X11,BWFTI$

(3)     LMJ     X11,WFTI$
                error exit
                normal exit
```

a. Error Return

A0	= 042	At least one table has not been written back by a call on Write Program File Table
	= 7777777777xx	I/O Error

b. Normal Return

The File Table Index has been written back to mass storage.

2. Registers Used: A0 - A5

2.5.8. Relocatable Output Routine (ROR)

The relocatable output routine is used to produce a relocatable element that is used for input to the Collector. The relocatable element is produced from relocatable items generated by language processors.

The relocatable output routine contains user interfaces SROR$, ROR$, EROR$ and TBLWR$ which are described in following subparagraphs.

2.5.8.1. Start Relocatable Output Routine (SROR$)

Purpose:

Initializes ROR prior to output of any relocatable text words.

Format:

```
L,U     A0,K-bit limit
LMJ     X11,SROR$
        error return
        normal return
```

Parameters:

K-bit limit, the number of bits required to contain either the largest control counter used or the number of undefined symbols for the relocation, whichever is larger.

Description:

SROR$ saves the K-bit limit for the relocatable element and establishes the program file write location for the element via ER PFWL$. (See Volume 3-11.3.1.5.) SROR$ adjusts the output buffer to minimize read-before-write operations, if possible.

The error return is taken when a nonzero status code is encountered in A2. A2 contains the status code (see Volume 3-11.3.1.6) from ER PFWL$.

2.5.8.2. Generation of Relocatable Output (ROR$)

Purpose:

Formats relocatable items and outputs text to the relocatable element.

Format:

```
L,U     A0,item addr
LMJ     X11,ROR$
        error return
        normal return
```

Parameters:

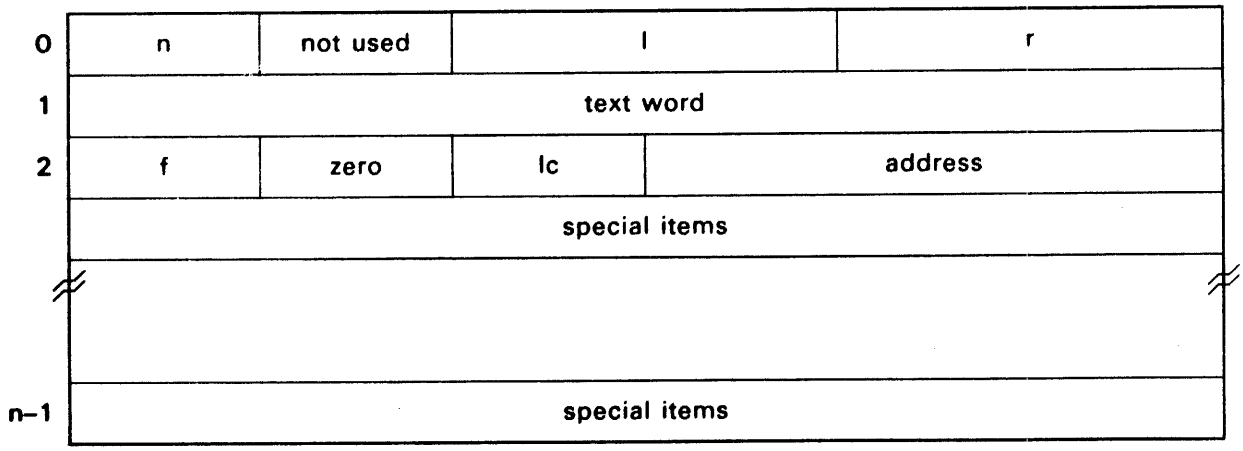
item addr – Address of item.

Description:

ROR is called for every text word (e.g., instruction or data word) to be inserted in the relocatable element. For each call the user will furnish the text word and its relocation information in an item.

The minimum length of the item is three words; however, it may be larger. Any relocation of a text word, other than a simple relocation of the right address under the same location counter that applies to the text word, must be handled through special relocation items. Any number of special relocation items may be appended to the item. The ROR item format is shown in Table 2-14.

Table 2-14. ROR Item



Word 0

n — item length, including any special relocation items

l — bits 16 and 17 of a left address field plus ten sign bits, or twelve sign bits for a left half-word field.

r — bits 16 and 17 of a right address field plus ten sign bits, or twelve sign bits for a right half-word field.

Word 1

text word

Word 2

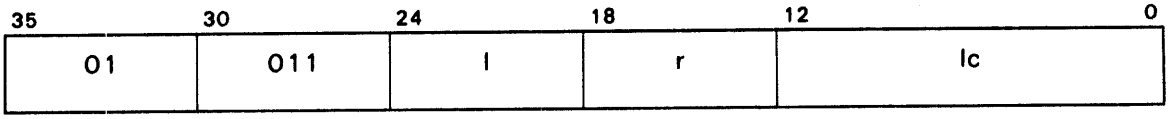
f — bit 32 = 1, right address (bits 0–15) relocatable
bit 33 = 1, left address (bits 18–34) relocatable.

lc — contains the location counter the text word is under and by which relocation specified by f takes place. May be overridden by a type 013 special item.

address — address of the text word.

Words 3, 4, etc. may contain special relocation items.

The Location Counter Relocation Item, type 011, format is:



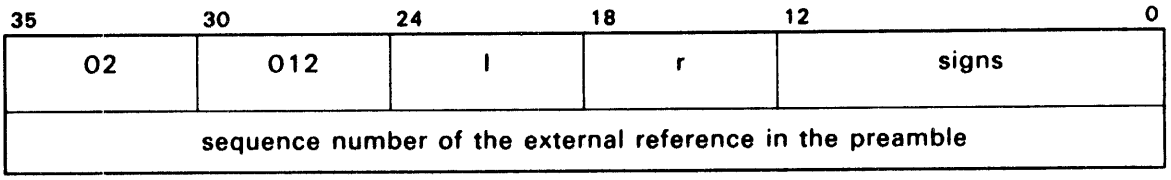
This item is used when relocation by a location counter other than the one specified in S3 of word 2 is desired.

l – the left margin of the field of relocation (a bit number in the range 0–35)

r – the right margin of the field of relocation (a bit number in the range 0–35, with r less than or equal to l)

lc – location counter number; a signed 12 bit integer. The starting address of the location counter is added if lc is positive or subtracted if lc is negative.

The External Reference Item (two words), type 012, format is:



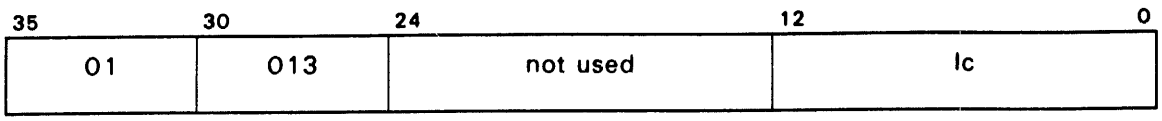
This item is used when relocation by the value of an externally defined symbol is desired.

l – the left margin of the field of relocation

r – the right margin of the field of relocation

signs – 12 bits of all zeros or ones, depending on whether the external reference is to be added or subtracted, respectively.

The Large Location Counter Item, type 013, format is:



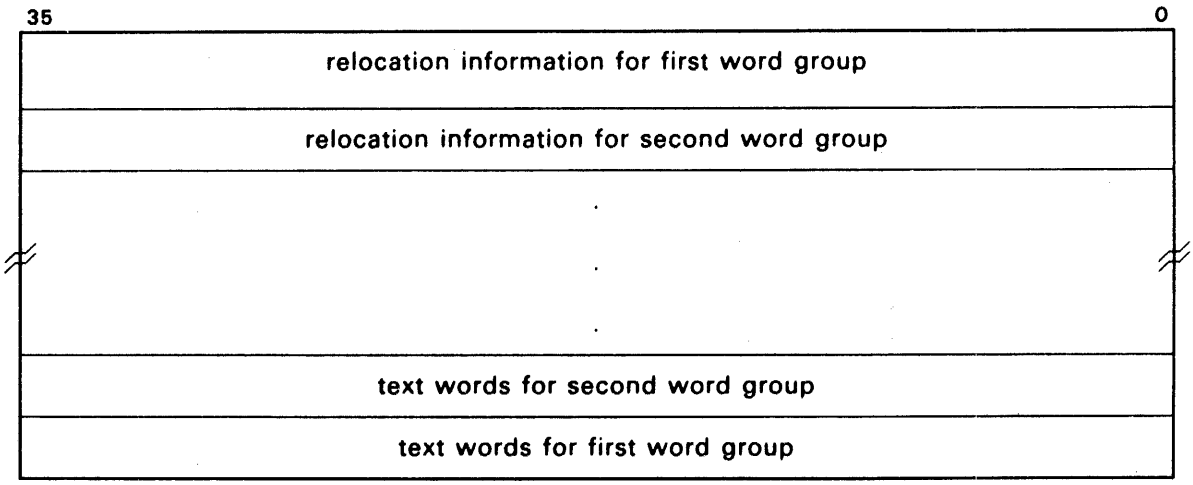
If this special item is used, it must be the first special item appended to the item.

This item overrides the lc in word 2 of the item. This allows location counters greater than 63.

lc – location counter number the text is to be under.

ROR formats the text words and the relocation information into blocks and outputs the blocks to the relocatable element. Blocks contain one or more word groups. Word groups represent text words that are allocated to sequential memory cells. Each word group consists of two separate parts; the relocation information and the text words themselves.

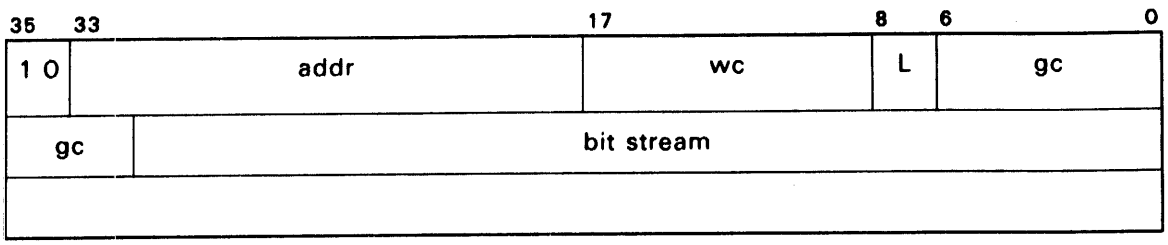
The relocation information for the word groups starts at the beginning of the block and the text words are at the end of the block. Word groups for relocatable information format is:



The text words are in the inverse order in which they are allocated to memory cells.

The relocation information for any word group is a stream of bits. The length of the stream is variable. The first bit of a stream is always the left-most bit of a word.

The stream of bits contain both fixed and variable length fields, the format is:



Bits 34-35 – an indicator which signals a new word group

addr – a fixed field containing the relative starting address of the first text word in the group.

wc – a fixed field containing the number of text words in the word group.

L – a fixed field indicating which location counter to use.

gc – an optional variable length field indicating a location counter. If L is 3 then this field is present. The field length is the K-bit count.

The rest of the bit stream is variable length and contains the relocation bits for the text words.

The error return is taken when:

1. an I/O error occurs, A1 will contain the status or
2. an internal inconsistency occurred in ROR, A1 will be zero.

2.5.8.3. End Relocatable Output Routine (EROR$)

Purpose:

Terminates ROR after the last text word has been processed.

Format:

```
L       A0,(transfer addr,transfer addr lc)
LMJ     X11,EROR$
        error return
        normal return
```

NOTE:

If no transfer location is specified (element may be a subroutine) then A0 must be negative.

Parameters:

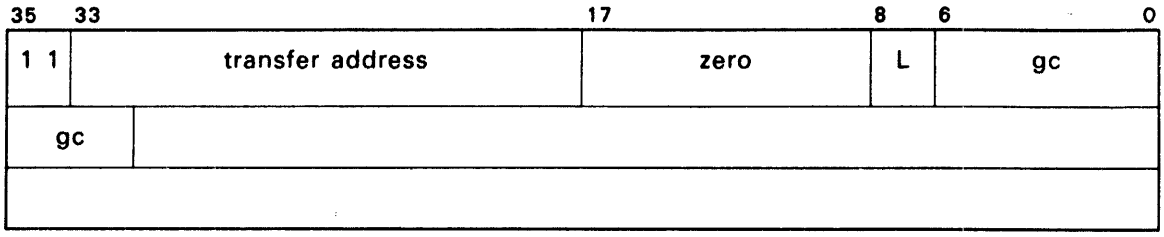
Transfer addr is the location of the first instruction to be executed in the main program.

Transfer addr lc is the location counter to which the address applies.

Description:

EROR outputs the last block built by ROR and generates a transfer image. The transfer image is a special word group.

The EROR format is:



EROR always generates a separate block for the word group:

The error return is taken when an I/O error occurs. The status is returned in A1.

2.5.8.4. Table Write Subroutine (TBLWR$)

Purpose:

Outputs the preamble constructed by the user to the program file containing the relocatable text. The text must have been previously entered in the program file through ROR and EROR must have been called.

Format:

```
L,U     A0,preamble addr
L,U     A1,preamble length
LMJ     X11,TBLWR$
        error return
        normal return
```

Description:

The preamble supplied by the user is written to the program file. The entries in PARTBL+29 through 38 (see Figure 2-1) are updated to reflect the preamble length, preamble location and the element type (5 = relocatable).

An ER PFI$ is done to update the file's table of contents.

The error return is taken when:

1. an I/O error occurs. A1 will contain the status.

2. an error occurs on the ER PFI$. A1 will equal 0 and A2 will contain the status.

The preamble consists of one to five tables. Each table will be discussed below. Only those tables necessary to a particular element, except the Base Table, are included in the preamble. The Base Table is always included.

The Base Table format is shown in Table 2-15.

Table 2-15. Base Table

0	index-location counter table	item count-location counter table
1	index-undefined symbol table	item count-undefined symbol table
2	index-entry point table	item count-entry point table
3	index-control information table	item count-control information table

The indices in bits 18-35 are relative to word zero of the Base Table. Bits 0-17 contain the number of items of the other tables. If the table is not present then the number of items will be zero.

The location in the table is the location counter number. The Location Counter Table (Table 2-16) is of variable length. The length is equal to the highest location counter number used plus one.

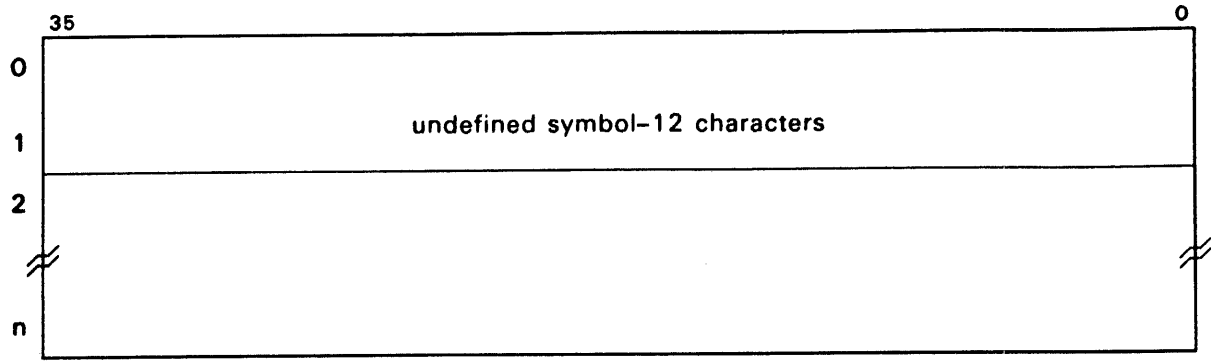
Table 2-16. Location Counter Table

0	K-bit count	length of location counter 0
1	minimum data area address	length of LC 1
N-2		length of LC N-2
N-1		length of LC N-1

K-bit count must be the same value supplied to SROR.

The undefined Symbol Table (Table 2-17) is a list of all symbols that are not defined within the element. The undefined symbols may be up to twelve characters in length, therefore each item is two words long. There may be up to 1024 undefined symbols, within the constraints of K-bit count.

Table 2-17. Undefined Symbol Table



The Entry Point format (see Table 2-18) is:

Each item is four words in length.

Words 0,1 - entry point name

Word 2 - descriptor word

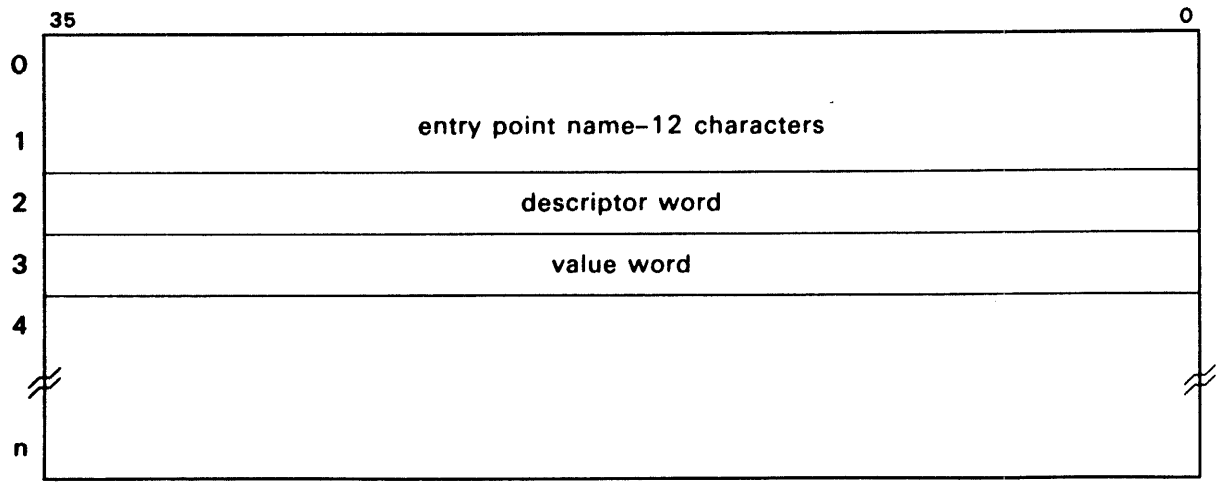
bit 27 = 1 - entry point is absolute and has value of value word

bit 27 = 0 - the value word is relative to the location counter

bits 18-26 - location counter

Word 3 - value word

Table 2-18. Entry Point Table



The Control Information format (see Table 2-19) is:

Each item is four words in length.

Words 0,1 - common block name

Word 2 - group number

designates named(2) and blank named(4) common blocks

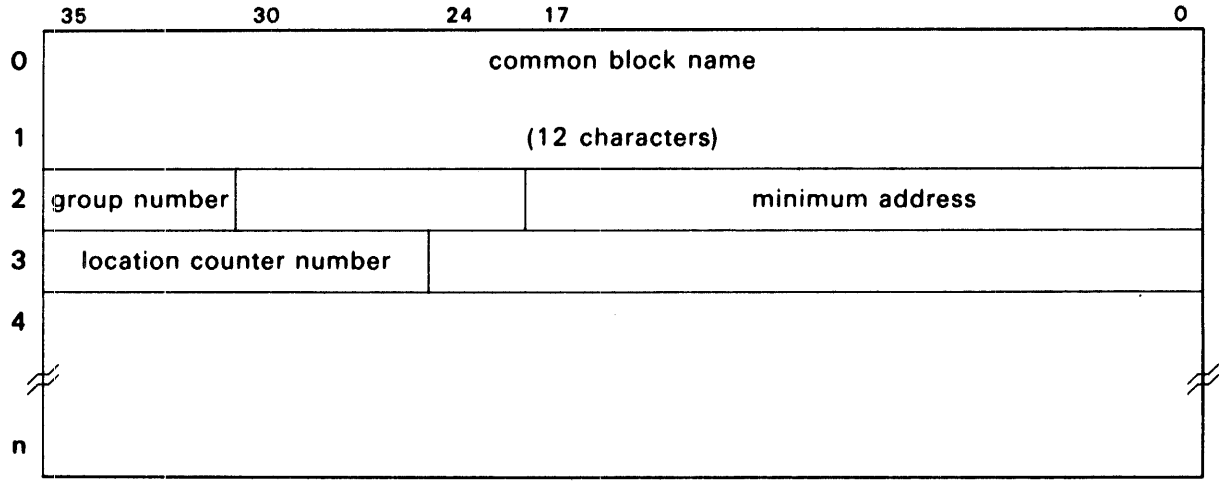
- minimum addr

minimum address at which this block will be addressable

Word 3 - location counter

location counter this common block will be under

Table 2-19. Control Information Table



2.5.8.5. Optimization Information

Both ROR and the Collector are optimized, as for storage space and relocation efficiency, for the instructions and data to be coded under control of location counters one and two, respectively, ROR has less processing to do and produces a more compact relocatable element. The Collector has a smaller bit stream to analyze.

2.5.9. Source Output Routine (SOR)

The Source Output Routine (SOR) is used to insert source images produced by a processor into a program file as a symbolic element. The symbolic element produced by SOR is suitable for input to another processor by SIR$. No other routine may be attempting to write into the designated program file. The first pass of SIR$ must therefore be completed before calling on SOR. (See Figure 2-1 for PARTBL description.)

PARTBL entries are as follows:

PARTBL+27,+28	-	internal filename
PARTBL+29,+30	-	element name
PARTBL+33,+34	-	version name

SOR inserts the following:

NOTE:

SOR uses the table item (13 words) normally used for relocatable output for symbolic output.

S2 of PARTBL+32	-	ASCII bit set or clear (bit 28) depending on mode of first image
S3 of PARTBL+32	-	element type number (01-symbolic)
T1 of PARTBL+35	-	cycle limit (system standard = 5)
T2 of PARTBL+35	-	latest cycle number (0)
T3 of PARTBL+35	-	current number of cycles (1)
S1 of PARTBL+36	-	processor code
H2 of PARTBL+36	-	length of text
PARTBL+37	-	location of text
PARTBL+38	-	Time and date element created. Initialized to zero at SSOR. User may insert time and date after SSOR call, otherwise current time and date will be used.
PARTBL+39	-	next write location in PF

The file, element, and version names may be entered in PARTBL by PREPRO.

File control images are constructed automatically by SOR.

The File Control Table (FCT) and the buffers provided by SOR for SDFO are similar to those required by SDFI. Therefore the external label SORFCT$ for the first word of the FCT has been provided. Initially zero, SORFCT$ is set non-zero upon the call to SSOR$ and reset to zero upon the call to ESOR$. If, at the time of a SSOR$ call, SORFCT$ is non-zero, a 01 (no find) A2 error condition is generated. User programs desiring to use the FCT and SOR buffers must do the following:

1. Check SORFCT$ to be zero before using.
2. Set SORFCT$ non-zero while using.
3. Reset SORFCT$ to zero when no longer required.

4. Be sure S2 of SORFCT$+3 is the correct I/O code.

NOTE:

SIR$ has its own input FCT and buffers for SDFI use.

2.5.9.1. Start Source Output Routine (SSOR$)

Purpose:

Initializes the SOR routine by setting up PARTBL and opening the program file.

Format:

```
LMJ     X11,SSOR$
        error return
        normal return
```

Description:

SOR is initialized and PARTBL entries are made for a symbolic element. The user receives control at the error return if ER PFWL$ (see Volume 3-11.3.1.5) encounters an error condition. On error return, A2 is nonzero (see Volume 3-Appendix B for status code).

2.5.9.2. Generation of Source Output (SOR$, SORA$, SORASC$, SORASCA$)

Purpose:

Outputs a source language from the user to a specified program file.

Format:

```
L       A0,(image length,image location)
LMJ     X11,SOR$ (or SORA$, SORASC$, SORASCA$)
        error return
        normal return
```

Description:

An image is transferred from the user area to the source output area. On the first entry, the label is also written. The user receives control at the error return if SDFO (see 2.6.4.2) encountered an I/O error (unrecoverable). A5 = I/O status code. SORASC$ is an alternate entry point for ASCII images.

The entry points SORA$ and SORASCA$ are provided for the cases when trailing blanks may be meaningful to the user and SOR should not adjust the image length. Non I/O error returns are generated for the two following cases:

1. A5 is zero for the illegal case of a zero image location with a non-zero image length.

2. The requested image length is over the 2047 word maximum allowable length, A5 (H1) = 0, and A5 (H2) = length.

2.5.9.3. End Source Output Routine (ESOR$)

Purpose:

Terminates the operations of the SOR routine.

Format:

```
LMJ     X11,ESOR$
        error return
        normal return
```

Description:

SOR is terminated, SDFO (see 2.6.4.2) is closed and the generated element is inserted in the program file (ER PFI$) (see Volume 3-11.3.1.1). If the user receives control on the error return from ER PFI$, A5 will equal 0. A2 will contain the descriptive code. If the error return is from SDFO an (unrecoverable) I/O error has occurred and A5 will contain the I/O status code.

2.5.10. Post Processor Routine (POSTPR$)

Purpose:

Removes all changes in the assignment status of a program file which were made by PREPRO, PREPRM or PREPF$.

Format:

```
LMJ     X11,POSTPR$
        error return
        normal return
```

Registers Used: X11,A0-A3

Description:

All files specified on the processor call statement are restored to their status preceding the processor call.

The error return is taken if any reference to CSF$ returns a nonzero status code.

2.5.10.1. Field Release (FLDREL$)

Purpose:

Remove changes in the assignment of program files which were made by FLDGETO$ or FLDGETM$.

Format:

```
L,U     A0,j
LMJ     X11,FLDREL$
        error return
        normal return
```

where:

j == address of data area given to FLDGET$.

Description:

Using the information saved in the first three words of the data area pointed to by j, the file is restored to its state prior to the calling of FLDGET$.

The error return is taken if the free returns a negative status.

2.6. UTILITY ROUTINES

The utility routines are available in the system relocatable library. Some of these routines (FDASC$, SDFI, SDFO) also are available in the common bank PIRCB$.

2.6.1. Master File Directory Service Package (MFDSP$)

Purpose:

Selectively retrieve items from a file produced by the ER MSCON$ (DGET$) and (DGETP$) functions.

Format:

```
L       A0,(size,address)
L,U     A1,function
LMJ     X11,MFDSP$
        error return
        normal return
```

where:

address - the address of an area in the user D-bank

size - the size of the above area in words

Registers Used: X11,A0-A5,R1-R3

Description:

The first two words of the area supplied by the user must contain the internal name of the file containing the output from a MSCON$, (DGET$) or (DGETP$) function. The user must have produced the file prior to calling MFDSP$.

The following is a list of the functions and their meanings:

00 Initialize for a file produced by a MSCON$ (DGET$) function. This function returns the address of the first lead item.

01 Initialize for a file produced by a MSCON$ (DGETP$) function. This function returns the address of the first main item.

02 Returns the core address of the next lead item.

03 Returns the core address of sector 1 of the above lead item.

04 If MFDSP$ was initialized with a function of 00, this function returns the core address of the next main item associated with the lead item retrieved by use of a function 00 or 02.

If MFDSP$ was initialized with a function of 01, this function returns with the next main item in the file.

05 Returns the core address of the next sector of the main item.

06 Returns the core address of the granule item associated with the main item retrieved by function 04 or 05.

07 Returns the core address of the next granule item associated with the main item retrieved by function 04 or 05.

When the normal return is taken, the address of the desired item will be in A1. If the item could not be found A1 will be set to 0.

When the error return is taken A0 will contain an error code. The codes and their meanings are as follows:

01	Function out of range
02	User supplied buffer too small
03	Illogical function sequence
04	I/O error, status will be in A5
07 x	Where x is an octal digit, indicating an internal error

The area size required by MFDSP$ is variable. MFDSP$ has 25 words of fixed storage. The dynamic storage is directly proportional to the number of different types of mass storage on which files are cataloged. A good rule of thumb is three words of storage for each different type of mass storage. If after allocation of the buffer to internal storage, the remaining buffer is less than the number of sectors in a physical disc record (prepping factor), the error return is taken with A0 set to 02. It is suggested that at least a track size buffer be used.

When MFDSP$ is initialized with a function of 01, functions 02 and 03 perform the same as function 04. A function of 01 may be used on a file produced by a (DGET$) MSCON$ function. The use of a function 01 causes MFDSP$ to utilize neither the look up table nor the lead items when performing subsequent functions.

2.6.2. Fieldata/ASCII Data Conversion (FDASC$)

The Fieldata/ASCII conversion routine consists of two elements: FDASC$ and TABLE$. FDASC$ is reentrant and does not assume quarter/third-word mode. See Section 3 for common bank usage.

2.6.2.1. Fieldata to ASCII Conversion Routine (FDASC$)

Purpose:

Converts Fieldata to ASCII

Calling Sequence:

```
        L,U     A0,input buffer word count
        L,U     A1,input buffer adrs
        L,U     A2,output buffer adrs

(1)     LMJ     X11,BFDASC$-1

(2)     LXI,U   X11,PIRCB$
        LIJ     X11,BFDASC$

(3)     LMJ     X11,FDASC$
```

Registers used: X11,A0-A5,R1

Description:

The Fieldata characters in the input buffer (six characters per word) are converted to ASCII and placed in the output buffer (four characters per word, with trailing space fill if necessary). The length of the output buffer in words must be at least 1 and 1/2 times the length of the input buffer. On returning control to the user program, A0 contains the length of the converted ASCII image (in words).

2.6.2.2. ASCII to Fieldata Conversion Routine (ASCFD$)

Purpose:

Converts ASCII to Fieldata

Calling Sequence:

```
        L,U     A0,input buffer word count
        L,U     A1,input buffer adrs
        L,U     A2,output buffer adrs

(1)     LMJ     X11,BASCFD$-1

(2)     LXI,U   X11,PIRCB$
        LIJ     X11,BASCFD$

(3)     LMJ     X11,ASCFD$
```

Registers used: X11,A0-A5

Description:

The ASCII characters in the input buffer (four characters per word) are converted to Fieldata and placed in the output buffer (six characters per word – with trailing space fill if necessary). The length of the output buffer in words must be at least 2/3 the length of the input buffer. On returning control to the user program, A0 contains the length of the converted Fieldata image (in words).

2.6.3. Fieldata/ASCII Conversion Table (TABLE$)

TABLE$ is 128 words long and has the entry point ASCFDASC$. H1 of the table is ASCII to FD translation. H2 of the table is FD to ASCII. H2 of the last 64 words of the table is unused.

Purpose:

Fieldata to ASCII conversion table used by FDASC$ routine. Entry point is 'ASCFDASC$'.

2.6.4. System Data Format Input/Output Routines (SDFI, SDFO)

The System Data Format Input (SDFI) routine and the System Data Format Output (SDFO) routine are two independent elements designed to handle files/elements that are in System Data Format (see Volume 3–11.2.3). Both routines are reentrant and become part of the user's program at collection time. For either routine the user must provide a File Control Table (FCT), an image and an image control word (SDFO) or image area (SDFI) and two equal length buffers. These buffers must be some multiple of 28 which is the length of a FASTRAND sector in size. Because disc physical records are most frequently four FASTRAND sectors (112 words) in size, SDFI and SDFO will run most efficiently when their buffers are a multiple of 112 words in length. If the buffer size is larger than 112 words but not a multiple, the excess (28, 56, or 84 words) will not be used in I/O operations, unless the I/O device is tape. For tape, the standard buffer size is 224 words. If tape is referenced, the buffer length should be the length of the tape block. A description of each routine and its corresponding File Control Table is given below. See section 3 for common bank usage.

2.6.4.1. System Data Format Input Routine (SDFI)

Purpose:

Inputs images one at a time to the user from a System Data Format (SDF) file/element on mass storage.

Format:

Open SDFI

```
     L,U     A0,FCT-addr

(1)  LMJ     X11,BSDFIO$-1

(2)  LXI,U   X11,PIRCB$
     LIJ     X11,BSDFIO$
```

```
(3)  LMJ     X11,SDFIO$
             error return
             normal return
```

The second entry point is provided which allows reading to begin at any word within the starting sector specified in the packet. The entry is referenced as follows:

```
     L,U     A0,FCT-addr
     L       A1,OFFSET

(1)  LMJ     X11,BSDFIOA$-1

(2)  LXI,U   X11,PIRCB$
     LIJ     X11,BSDFIOA$

(3)  LMJ     X11,SDFIOA$
             error return
             normal return
```

Input Image

```
     L,U     A0,FCT-addr

(1)  LMJ     X11,BSDFI$-1

(2)  LXI,U   X11,PIRCB$
     LIJ     X11,BSDFI$

(3)  LIJ     X11,SDFI$
             error return
             end-of-file return
             normal return
```

Close SDFI

```
     L,U     A0,FCT-addr

(1)  LMJ     X11,BSDFIC$-1

(2)  LXI,U   X11,PIRCB$
     LIJ     X11,BSDFIC$

(3)  LMJ     X11,SDFIC$
             error return
             normal return
```

Registers Used: X11,A0-A5,R1-R3

Description:

Images are retrieved from an SDF file/element on mass storage or tape until an SDF end-of-file image control word is encountered. The EOF control word contains an 077 in S1. SDFI recognizes 051 continuation control images and does not pass them to the caller.

SDFI is initialized by calling SDFIO$ or SDFIOA$. If the latter entry is referenced, the offset in A1 should satisfy $0 \leq A1 < 28$. Once the file has been opened for input, the routines SDFI$ and SDFIC$ may be used as usual. If A1 = 0, the effect of SDFIOA$ is identical to that of SDFIO$. Thereafter, for each image requested by the user, a call must be made to SDFI$. The two buffers provided by the user are alternately filled with input images. With each call on SDFI$ an image is transfered to the area specified by the user in the FCT, and its corresponding Image Control Word is set in FCT + 10. If the image is larger than the area provided by the user, the user's area will be filled and the rest of the image skipped. When reading from a mass storage file, SDFI maintains the mass storage address. This process is continued until an end-of-file image control word is encountered. At this point, the end-of-file return is taken by SDFI$. The user closes SDFI by calling SDFIC$.

The file control table for SDFI is described in Table 2-20. The table must be provided by the user and all information except that marked 'filled' must also be provided.

The error return from SDFI is taken when an unrecoverable I/O error status code occurs (see Volume 2-Appendix C.3). When an error return is made, A5 contains the I/O status code. The user program may examine the packet in words 0 through 5 of the FCT which is a standard I/O packet.

Table 2-20. File Control Table for SDFI

FCT+0	internal filename			
+1				
+2				
+3		020		
+4	buffer length (words)			*buffer address (filled)*
+5	Relative Mass Storage Address (Sector)			
+6	buffer number 1 address			buffer number 2 address
+7	buffer length (sectors)			length of image area
+8	01			image area location
+9	01			*buffer image location (filled)*
+10	*SDF Image Control Word (filled)*			

Words 0 through 5 are the I/O Packet

- H2 of FCT + 4 is filled by SDFI when filling a buffer from the SDF file or element.

- FCT + 5 is the location (sector) of the Image Control Word (ICW) of the image to be read.
- H2 of FCT + 9 is used by SDFI for the input buffer address for the next image.
- FCT + 10 is filled by SDFI with the ICW of the SDF image.

2.6.4.2. System Data Format Output Routine (SDFO)

Purpose:

Outputs images one at a time from the user to a System Data Format file/element on mass storage.

Format:

Open SDFO

```
     L,U      A0,FCT-addr

(1)  LMJ      X11,BSDFOO$-1

(2)  LXI,U    X11,PIRCB$
     LIJ      X11,BSDFOO$

(3)  LMJ      X11,SDFOO$
              normal return
```

Output Image

```
     L,U      A0,FCT-addr

(1)  LMJ      X11,BSDFO$-1

(2)  LXI,U    X11,PIRCB$
     LIJ      X11,BSDFO$

(3)  LMJ      X11,SDFO$
              error return
              normal return
```

Close SDFO

```
     L,U      A0,FCT-addr

(1)  LMJ      X11,BSDFOC$-1
```

```
(2)  LXI,U    X11,PIRCB$
     LIJ      X11,BSDFOC$

(3)  LMJ      X11,SDFOC$
              error return
              normal return
```

Registers Used: X11,A0–A5,R1–R3

Description:

Images are output to an SDF file/element either on tape, or on mass storage at the relative location specified in the FCT.

SDFO is initialized by calling SDFOO$. Thereafter, for each image that is to be output to the SDF file/element, a call must be made to SDFO$. With each call on SDFO$ an SDF image control word (ICW) must be provided in FCT + 10. The ICW is set in the output buffer and is immediately followed by the output image whose location is specified in H2 of word 8. When an image spans the output buffers, a partial transfer is made to the first buffer and the buffer is written out. The remainder of the image is then transferred to the next buffer. When writing to a mass storage file, SDFO maintains the write address. The process is continued until the user makes a call on SDFOC$. At SDFOC$ an end-of-file ICW is set in the output buffer and the last buffer is written out. If the write was to mass storage, SDFOC$ returns an adjusted mass storage address in FCT+5 so sector length of text may be computed.

The error return is taken from SDFO when an unrecoverable I/O error occurs (see Volume 2–Appendix C.3). When an error return is made, A5 contains the I/O status code. The user program may examine the packet in words 0 through 5 of the FCT which is a standard I/O packet.

Table 2-21 describes the File Control Table for SDFO. The table and all information except that marked *'filled'* must be provided by the user.

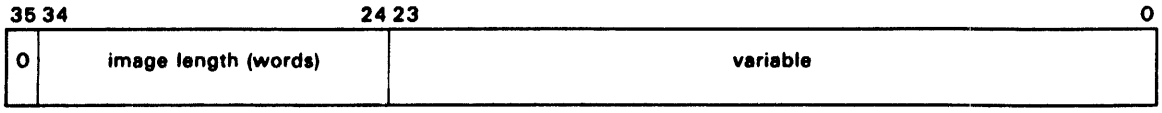
Table 2-21. File Control Table for SDFO

FCT+0	internal filename			
+1				
+2				
+3		010		
+4	*buffer length in words (filled)*		*buffer address (filled)*	
+5	Relative Mass Storage Address (Sector)			
+6	buffer number 1 address		buffer number 2 address	
+7	buffer length (sectors)		*length of image area (filled)*	
+8	01		image area location	
+9	01		*buffer image locator (filled)*	
+10	SDF Image Control Word			

Words 0 through 5 are the I/O Packet

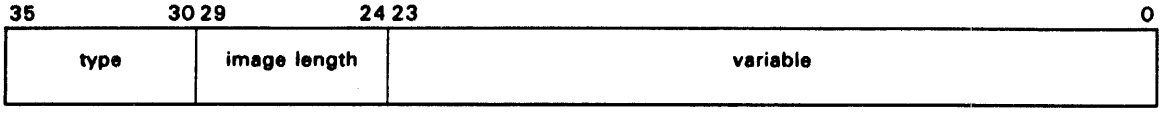
- H2 of FCT + 4 is filled by SDFO when writing out a full buffer.
- FCT + 5 is the location (sector) to which images are to be written.
- H2 of FCT + 9 is used by SDFO for the output buffer address for the next image.

The general format of the image control word located in FCT + 10 is:



Data Image Format

or



Control Image Format (bit 35 = 1)

For a general discussion of SDF see Volume 3-11.2.3. For a description of SIR$ generated SDF see 2.5.6.

For FORTRAN V formatted I/O, the label and data image control words are described in: SPERRY UNIVAC 1100 Series FORTRAN V Library, UP-7876 (current version) and SPERRY UNIVAC 1100 Series FORTRAN V Programmer Reference, UP-4060 (current version).

3. Processor Interface Routine Common Bank (PIRCB$)

PIRCB$ provides all the capabilities of the system relocatable library routines BSP$, FDASC$, SDFI, and SDFO.

The common bank entry points are defined in SYSLIB element CBEPPIR$. These entry points are the same as the ones in the relocatable routines except they are prefixed with the letter B. The common bank routines have dual referencing capability. They may be referenced via LMJ (auto switch), for systems which can base more than one bank simultaneously, or Common Bank LIJ. Three calling sequences are available for routines in a common bank. The type of call has been identified throughout this Volume as (1), (2), and (3). They are identified as follows:

(1) Auto Switch Method

(2) Common Bank LIJ

(3) Relocatable Collection Method

To utilize the LMJ referencing method the bank, PIRCB$, must be based on one of the PSRs when the routine is referenced (See Volume 3 Section 2). To prevent part of a program from becoming inaccessible, care must be exercised when collecting the PIRCB$ bank. The bank starts at address 01000 and occupies 03777 words.

4. CULL Processor

4.1. INTRODUCTION

This section describes the CULL processor which generates a listing of symbols, cross-referenced to the element and line in which they are found.

4.2. @CULL

Purpose:

Scans a collection of symbolic elements and produces a cross-reference listing of the symbols found, the elements, and the lines on which they occur. A list of symbols which are either to be omitted from the sort or the only symbols to be included in the sort may be specified. The CULL processor is called by the @CULL control statement.

All parameters in the @CULL control statement are optional.

Format:

```
@label:CULL,options   pro/scol(res),name-1,...,name-n.
```

Parameters:

options — Specifies available options for obtaining sorted symbolic listings (see Table 4-1).

pro — Specifies the symbolic elment being scanned/sorted. The possible values are:

- ALG – ALGOL elements
- ASM – 1100 Series assembly code
- COB – COBOL elements
- DATA – all symbolic elements
- DOC – DOC elements
- ELT – symbolic data element

FOR – FORTRAN elements

MAP – MAP elements

PLS – PLUS elements

If omitted, ASM is assumed.

scol — Specifies the column number where the scan automatically stops. If omitted, the following values are assumed:

ASM– 40

ALG, COB, FOR – 72

DATA, DOC, ELT, MAP, PLS – 80

res — Specifies the number of mass storage positions reserved for scratch files. If omitted, 2 is assumed (approximately 70,000 references).

names — Specifies element or filenames in standard notation (see Volume 2–2.6.6).

Description:

Each file must be assigned or catalogued. If filename is not given, TPF$ is assumed. All files are returned to their original status when the CULL processor terminates. If a file is not a FASTRAND–formatted program file, an error is noted, and the parameter is ignored.

If read/write keys are necessary, they must be given in the first reference to the file. Keys appearing anywhere else are ignored.

The set of elements to be scanned and sorted is formed in one of two ways:

- If an element name is a parameter, it is included in the scan/sort.
- If a filename is a parameter, the most recent cycle of each symbolic and procedure element is included in the scan/sort, unless the O and P options are specified.

Standard system dropout rules apply to both file and element name formats (see Volume 2–3.2.7).

Data images, following the @CULL control statement, are scanned for strings of characters separated by blanks; these strings are the special symbol table. If the M option is specified, only the symbols in this table are accepted; if omitted, the symbols in this table are ignored. If the S option is used, the special symbol table is printed. Processing does not begin until the end of the data is encountered (indicated by encountering a control statement).

Certain options specify that a symbol which is found under special circumstances is to be marked. When a symbol is to be marked, then all occurrences of that symbol elsewhere are to be marked. If the N option is not specified, only unmarked symbols are printed. If the N option is specified, then only marked symbols are printed.

The following special options are available:

- Special Options for Symbolic Data Elements

  The data scanner accepts strings of characters from the set A–X, 0–9, $, and period (.) up to 12 characters in length. See Table 4–1 for the special options available only to the data scanner.

- Scanner for Fortran Elements

  Action is the same as data scanner, except comment lines are ignored.

- Scanner for COBOL Elements

  Action is the same as data scanner, except symbols may be up to 30 characters.

- Special Options for Assembly Language Elements

  The assembler scanner accepts strings which are valid identifiers or numbers in the 1100 Series assembly language. These strings may include up to 12 characters, and the symbol $ is recognized. The assembler scanner recognizes an identifier when it is a label, directive, or operand; occurrences of labels and directives are followed by * and D, respectively. Labels defined at another level are marked with a double asterisk. The special options available to the assembler scanner are given in Table 4–1.

Table 4–1. @CULL Control Statement, Options

Option	Description
	General Options
C	Produces a 72–column listing; omits list of elements processed and summary information at end.
E	Does not eject the page. Otherwise, page eject for each time symbol begins with different character than previous symbol.
H	Coupled with the L option, produces page headings of the element name currently being listed.
L	Produces a full listing of each element scanned/sorted.
M	Only symbols in data images are accepted.
N	Marked symbols only are printed; otherwise only unmarked symbols printed.
O	Symbolic elements are not scanned.
P	Procedure elements are not scanned.
S	Symbols in data images are printed.
Z	Accepts symbolic elements with subtype = 0.

Table 4-1. @CULL Control Statement, Options (continued)

Option	Description
	Special Options for Symbolic Data Elements
A	Does not sort strings beginning with an alphabetic character.
D	Does not sort strings beginning with any character from the set: 0-9, $, and period (.).
	Special Options for Assembler Language Elements
A	Does not sort identifiers.
D	Sorts numbers and the symbol $.
N	Prints only marked symbols (otherwise, prints only unmarked symbols).
I	Any symbol that appears in the special internal table is not sorted. This table contains all symbols that are commonly used as directives, mnemonics, register names, and j designators. If the M option is specified, the internal table is used as a supplement to the special symbol table.
Q	Sorts data items from the set A-Z, 0-9, $, and period (.) which are enclosed in quotes. These references are followed by a Q.
U	Causes used symbols to be marked (and hence not printed in absence of N option, thus giving listing of symbols defined as labels but not otherwise referenced).
W	Marks a symbol when it appears in the label field.
X	Marks a symbol when it appears in the label field and is followed by an asterisk (*).
Y	Restricts the W and X options to elements explicitly named in the @CULL control statement.

Examples:

1. `@CULL`

2. `@CULL,D  ASM/80(30),EXEC1.,EXEC2.,EXEC3.,EXEC4.`

3. `@CULL,D  DATA/80(10),PRM/REV21`

4. `@CULL  ,Q*F.A,.B,.C,.D`

1. Sorts all of the elements in the Temporary Program File (TPF$) using the assembler scanner.

2. Used to obtain a complete scan and sort of the assmebly language files EXEC1 through EXEC4 and to reserve 30 positions of mass storage for use by the CULL processor.

3. Data element PRM/REV21 contains images for a document. This control statement is used to generate a glossary of words used.

4. Used to CULL elements A,B,C and D in file Q*F.

5. Document Processor (DOC)

5.1. INTRODUCTION

The Document Processor (DOC) is used to produce formatted printed listings (ASCII or Fieldata) of a document and to update a document. The document is simply a symbolic element in standard format and it may be manipulated by the FURPUR processor (see Volume 3- Section 4) just like any other symbolic element.

Control of the document is provided in the following ways:

- @DOC control statement options
- Control commands within the text that provide:
  - listing control
  - text control
- Editing commands that permit:
  - input line image editing
  - character string editing

5.2. @DOC FORMAT

Purpose:

The DOC processor is used to produce a formatted printed listing of a document or to update a document. The DOC processor is called by the @DOC control statement.

All parameters in the @DOC control statement are optional.

Format:

```
@label:DOC,options   elt-1,elt-2,altfile,form,form-params...
```

Parameters:

options — See Table 5-1.

elt-1 — Specifies the symbolic input element, output element for I option, or both for U option. May be omitted if input follows the control statement, and no output element is to be created.

elt-2 — Specifies the symbolic output element (not used with I or U options). May be omitted if output element is not created.

altfile — Specifies alternate print file to be used for output. If the file specified is not catalogued, a temporary file will be assigned. The version subfield of this specification may contain IDENT, in which case SIR corrections must be terminated by @EOF or @ENDX followed by an identification line, such as the name of the person to whom the listing is to be returned. This line will be solicited by the typeout 'MSG?'. To print the alternate file, the user must @FREE and @SYM the file himself, which can be done only if the file was catalogued. A temporary alternate file is suitable for examination by the text editor, for example.

If the filename is omitted but IDENT is used in the version subfield, DOC will catalogue, assign, write into, breakpoint, free, and queue for printing a file whose name will be chosen by DOC, based on the time of day and the generated run-id. This file should be unique. Use of the B option is not advisable in this case.

When this method of automatic file assignment is used, the qualifier field may specify the symbiont to be used for printing the file. (For example, "CSPPR1*./IDENT" in field 3.) This may be necessary if not all symbionts have lowercase printing capability.

form — Specifies the output device form. The various permitted values are given in Table 5-2. If omitted, the value HSP is assumed.

form-params — Specifies values to override standard form parameters. See Table 5-2.

Other specification fields are reserved for future expansion and should not be used.

Table 5-1. @DOC Control Statement Options

Option Character	Description
A	Not used.
B	Inhibit @BRKPT of alternate file when one has been specified by use of the altfile field.
C	Activates automatic lower-casing procedure to convert all-capitals document (produced by monocase device or created in Fieldata) to mixed uppercase and lowercase.
D	Inserts current data in title printed on each page.
E	Specifies global page numbering rather than numbering each chapter separately. With the E option, page numbers will not be reset to 1 for each new chapter. The table of contents will always be numbered separately.
F	Page numbers will be centered and printed at the foot of the page rather than in the top right corner.
G	Remove hyphenation at end of lines (may be overridden by UNHYPH directive; must not be used with H option).
H	Activates automatic hyphenation (may be overridden by HYPHEN directive; must not be used with G option).
I	Standard SIR option.
J	Not used.
K	Hard copy output with TKTRNX or TKTRN2 forms.
L	When right margin space is available (beyond rgd which must be nonzero), flags indicating the control directives encountered will be printed in the right margin. These flags will consist of the first two characters of the directive name, followed by the parameter specified, if any.
M	Produce modified listing (line numbers moved to within page boundaries).
N	Suppress listing (may be overridden by LIST and UNLIST directives).
O	When the R option is used, the listing produced will be right-aligned, but the output element will not. The O option is used to produce a right-aligned output element. The O option is meaningless unless the R option is used and an output element is specified.
P	Output element will be Fieldata instead of ASCII. This option is not needed if the input element is Fieldata. Also see remarks under Q below.
Q	Output element will be ASCII instead of Fieldata. This option is needed only if the input element is in Fieldata. Otherwise, the output element is always written in ASCII. If both P and Q are present, Q will override P.
R	Right margin alignment activated (may be overridden by RIGHTM directive.)

Table 5-1. @DOC Control Statement Options (continued)

Option Character	Description
S	Not used (single spacing is assumed mode).
T	Inverts the normal table of contents generation mode. For untitled forms, the table of contents will be generated, for titled forms, the table of contents will be suppressed.
U	Update (cycling is not available) when the U option is specified, the output element will be cycle 0 of a new element with the same name as the input element.
V	Page ejects for new chapters (level 1 counter change) are suppressed. This option implies the effect of the E option.
W	Standard SIR option.
X	Allows construction of the index file DOC$X when INDEX directives are specified in the text of the document (without the X option, the file will not be created), and printing of a sorted index following the table of contents.
Y	Not used.
Z	Not used (reserved for diagnostic purposes).

Table 5-2. @DOC Device Forms

Form	Description
HSP	Standard high-speed printer format.
HSP2	High speed printer format with two pages of text per printed page (60 character line width, maximum of 70 lines per page).
FR80	Modified HSP for use with FR80 microfiche output device.
TTY	72 character teletypewriter format (line numbers only with M option).
U10064	64 character line width UNISCOPE 100 Display Terminal format (line numbers only with M option).
U10080	80 character line width UNISCOPE 100 Display Terminal format (line numbers only with M option).
U100	Same as U10064.
DCT 500	132 character teletypewriter format.
DCT1K	DCT 1000 (same as DCT 500).

Table 5-2. @DOC Device Forms (continued)

Form	Description
TKTRNX	TEKTRONIX™ 4012/13 format, single column (same as TTY except that page erases are generated, and hard copy requests are made if the K option is used.
TKTRN2	TEKTRONIX 4012/13 format, two columns per page, otherwise same as TKTRNX (CAUTION: very short lines are printed).
FORM	User-defined form. The parameters defining the form are numbers given in fields five through eight of the DOC processor call statement, with the format as follows, starting from field four: "FORM,mulcol*ttlsw.datpos/pnrpos(ttlsp),lpp/cpdl,lmrg/llng,lgd*rgd.lnsw/lnpos" The fields have the following meanings: mulcol — One less than the number of columns per page. 0 for all standard form definitions, except HSP2, for which it is 1. ttlsw — Title switch: 1 for titled forms, 0 for untitled. datpos — Position of date in title when D option is used. pnrpos — Page number position in title and in table of contents entries. ttlsp — Number of blank lines to follow the title line. lpp — Number of text lines per page; does not include title or cutting guide information. cpdl — Characters per device line (hardware page width). lmrg — Left margin position of text. llng — Text line length. lgd — Left cutting guide position. rgd — Right cutting guide position. lnsw — Line numbering switch: 1 for line numbers printed, 0 for line numbers not printed. lnpos — Line number print position (meaningful only if lnsw is not zero). If pnrpos is zero, it will be computed as llng+lmrg-7. All print positions are measured in columns from 0 which is the left-most column. If rgd is 0, page quides will not be printed. Default values for omitted parameters are those for the HSP form. The M option is not meaningful when FORM is used. For convenience, the specifications of the form in use are printed as part of the DOC header, even when a standard form is being used. This will assist the user in defining his own forms. Any of these parameters may be specified for a standard form, in which case the normal value is replaced. Parameters not specified will retain their standard values.

Table 5-2. @DOC Device Forms (continued)

Form	Description
	NOTE: For sites which have configured 8 lines-per-inch printing instead of 6 lines-per-inch, when the HSP form is used, the recommended value of "lpp" is 70. Whenever the user supplies values for any of these parameters, there exists the possibility of describing a form which is internally inconsistent. DOC will check for this and make adjustments, including an error diagnostic, if necessary.
TKFORM	User-defined TEKTRONIX form. Same as FORM except that TEKTRONIX features are enabled.

The forms corresponding to interactive devices (everything except HSP and FR80) are generally referred to as untitled. For these forms, references to titling are not meaningful. Moreover, a table of contents will be generated for an untitled form only if the T option is used. Nevertheless, a title image must always be given, even for an untitled form.

5.3. OUTPUT LISTINGS

The first image of a document element is the title. It may begin in column 1, while its length is dependent on the form selected. The title of the document is printed at the top of each page. If the D option was used, the current date will be suffixed to the title, in the format dd mmm yy. The page number, sequenced within chapters, is printed flush with the right margin, following the title. Pairs of periods, to be employed as trimming guides, are printed below the title and after the last line of text on each page. For the HSP form, these periods are 8 1/2 inches apart. The precise position of the trimming guides may be affected by the M option.

The text appears below the title and page guides. The number of lines per page is device dependent, but is 50 for the HSP form. The line numbers of the output element are printed to the left of the text (for some forms, only if the M option is used). New material (introduced via SIR corrections) is indicated by line numbers with an asterisk suffixed.

Hyphenation at the end of a line may be obtained automatically through the use of the H option or the HYPHEN directive. When text justification requires it, hyphenation will be removed and the two parts of a word run together. This action may be forced for all hyphenation by use of the UNHYPH directive. If a word which normally contains a hyphen, such as 'right-handed' appears with the hyphen at the end of a line, to avoid losing the hyphen altogether, two consecutive hyphens should be used. Other than that, an input element may be hyphenated freely.

5.4. INTERNAL CONTROL DIRECTIVES

Internal control directives are distinguished from text by the appearance of a nonblank character in column 1. The types of control directives available are described in the following paragraphs.

5.4.1. Title Control

Title directives have a nonzero digit in column 1, followed by text, which is the title for that portion of the document. This title is not to be confused with the document title, which is specified by the first image of the input. Nine counters are maintained by the DOC processor in a hierarchical organization. The most significant of these is the level 1 counter, also called the chapter counter, which is controlled by title directives having a '1' in column 1. The other eight counters are ordered from level 2 to level 9, controlled by title directives having '2' through '9' in column 1, respectively. The number of counters printed corresponds to the level of counter being referenced. All counters at a lower numbered level are printed, and any that were zero are set to one. The counter at the level being referenced is incremented by one and printed. All counters at a higher level are set to zero and therefore are not printed. Titles for counters 2 through 9 will be printed with two vertical spaces before and after the line. A title for counter 1 will be printed on a new page, and will be followed by two vertical spaces. Level 1 directives also reset the page number (in the absence of the V or E options), table number, and figure number to 1. The test on a title directive, which may begin in column 2, is printed following the values of the various counters. The exact amount of text that may be printed will depend on the level being referenced, the form in use, and the values of the various counters, but a limit of 40 characters is a good guide.

The image inserted in the output element will contain the counter level digit in column 1 followed by the text of the section title, and will thus not correspond to the printed listing.

In addition to appearing in the text of the printed document in the appropriate place, the title directives are also used to produce a table of contents for the document which will be printed at the end of the listing. Untitled forms require the T option for the table of contents to be printed. The image inserted in the table of contents will contain the page numbers on which the title directives appeared.

In addition to chapter and section titling described above, tables and figures may also be titled and automatically numbered. This is done using the TABLE, FIGURE, and END directives (see Table 5-3).

5.4.1.1. Title Control Compatibility

Older versions of DOC accepted only four levels of title control counters, numbered 4 through 1, and 4 was the most significant level. This is not directly compatible with the scheme described above,so the following compatibiltiy transformation is made: If the first title directive encountered contains a '4' in column 1, the document is assumed to be in the old format. Digits 1 through 4 encountered in column 1 will then be transformed by the mapping as follows: 1-4, 4-1, 2-3, 3-2. This transformation will affect the output element, if one is specified. When the compatibility transformation is applied, an appropriate diagnostic message will be printed.

5.4.2. Input Case Control

For ease of conversion of a Fieldata DOC element to ASCII, or for use with a monocase input device (such as a teletypewriter or card reader, DOC contains a routine which will automatically convert text in uppercase only to mixed uppercase and lowercase. The method used is to leave in uppercase the first character of each word of a title, and also the first alphabetic character following a period, colon, question mark, or exclamation point. All other characters will be made lowercase. This means that proper nouns, abbreviations, and so on will require manual editing through a dual-case device.

Table 5-3. Title Control Directives

Directive	Description
TABLE text .	The text is the title of the table, which will be centered between the left and right margins. The chapter and table counters will be edited, preceded by the word 'Table' and followed by the specified text. Within a table, the status of the HYPHEN, UNHYPH, and RIGHTM modes is saved, and these modes are turned off. Their status will be restored when 'END' is encountered. Text composition will not be performed within a table, so the table must be constructed exactly as it is desired to appear. If text composition is desired within a table, the 'END' may immediately follow the 'TABLE' directive, followed by the text of the table. This will give the author the titling and numbering feature of the TABLE directive without suppressing text composition or the HYPHEN, UNHYPH, or RIGHTM modes.
FIGURE text	The discussion above applies fully to the FIGURE directive, except that a figure will be titled as a figure, and there is a separate counter for figures. Title will be centered between left and right margins.
FXFORM	This command provides the fixed format of TABLE or FIGURE for the text which follows, without, however, providing a title for the text. Fixed format is terminated by the END directive.
END	The END directive is used to terminate a table or figure, or to terminate fixed format mode. It restores the G, H, and R option settings and turns on text composition.
TITLE text	This directive replaces the existing title information (obtained from the first image of the document or a preceding TITLE directive) with the title specified by "text", a string beginning in column 7.
SUBHDG x,text	This directive allows the specification of subheadings to be printed at the top of each page, or the elimination of subheadings already being printed. The "x" field may be one of the following: N – Do not print subheadings ("text" is ignored) L – Print this subheading flush left C – Print this subheading centered R – Print this subheading flush right The "x" field must appear in column 8 and the "text" field begins in column 10. The "text" field specifies the contents of the subheading, except when X=N. Up to 4 subheadings may be in use simultaneously. It is not necessary to stop the use of subheadings at the end of a document before printing the table of contents; the table of contents printing is done with its own subheadings. Subheadings are not printed for untitled forms. The subheadings will be printed in the order encountered. Since subheadings are cumulative, the N form must be used before changing to a new set of subheadings.
SETCTR k,n	This directive allows setting any of 12 counters used by DOC to a specified value. The "k" field may be any of the following: digit 1-9 – specified section counter P – page counter F – figure counter T – table counter The "k" field must be in column 8, and the "n" field is an integer beginning in column 10 with a maximum of 999999. The specified counter will be set to n–1, so that at the next incrementation, it will be given the specified value. If a section counter is being set, all higher numbered section counters are set to zero, and any lower numbered section counters which are zero will be set to one.

Table 5-3. Title Control Directives (continued)

Directive	Description
.	This means that section counters should be set in increasing order if more than one is being set. This directive is most useful when composing a document in parts or for inserting material into an existing document.
TOCLVL n	This command is used to control the amount of section titling information which appears in the Table of Contents. The "n" field is a single decimal digit indicating the highest level of titling to appear. For example, the normal mode is 9, a setting of 0 prevents all entries from appearing, while a value of 2 would allow only first and second level titles to appear in the table of contents, and so on. This command is used to keep the Table of Contents from becoming cluttered.

The lower-casing routine is activated by the C option or by the CASE directive. The CASE directive is intended for use when a portion of a Fieldata document is to be left in uppercase, or when some portion of a document has been edited on a monocase device. Since the function of the CASE directive, once it has been performed, is not needed again in the output elment, the CASE directive images will not be written to the output element. The CASE directive is described as follows:

CASE *k,x* Controls lower-casing routine.

k = L *x* is not used. Turns the lower-casing routine on.

k = N *x* is not used. Turns the lower-casing routine off.

k = C *x* is used. Denotes *x* as a "shift character". When in lower-casing mode (set by C option or "CASE L"), all alphabetic characters between two shift characters will remain in upper case. Any ASCII character may be specified for *x*. If *x* is a blank, the shift character feature is inactive, which is the situation at the start of a document. The CASE command and all shift characters will be deleted from the final document. The shift character may be changed as often as desired.

5.4.3. Listing Control

Listing directives specify the appearance of the listing. The format is a directive word, beginning in column 1, extending no further than column 6. In some cases a specification field may follow. The available commands and their descriptions are given in Table 5-4.

Table 5-4. Listing Control Directives

Directive	Description
DOUBLE	Selects double spacing.
CENTER text	The text specified will be printed centered between the left and right margins. Since the LENGTH command affects the margin position, it will affect the positioning performed by this command. For proper positioning, exactly one blank should follow the R in the word CENTER, since leading blanks are not recognized. Trailing blanks will be stripped.
EJECT n	Ejects paper n times to begin a new page with n-1 intervening blank pages. If n is zero, there is no effect. If n is omitted, 1 is assumed.
FLUSHR text	The text specified will be printed flush with the right- hand margin. Except for the position on the line, all considerations specified for the CENTER command apply equally to the FLUSHR command.
HYPHEN mode	Turns the automatic hyphenator on or off, where mode is either ON or OFF, beginning in column 8. The heuristic used does not guarantee dictionary-correct results, which should not be expected.
INDEX text	The text specified, which begins in column 7, will be written to the index file project-id*DOC$X if the X option was specified on the @DOC processor call card. The level 1 and 2 section counters and the page number will be suffixed. A limit of about 40 characters should be observed. The DOC$X file is written in SDF (System Data Format). DOC$X may be attached as a @USE name tc some other file if the DOC user so desires. If the X option is on at the end of the document, the index items will be sorted in alphabetical order and printed following the table of contents. The INDEX command is transparent. This means that unlike other commands, it may occur at any point in the text of the document without affecting line composition. Because of the requirements of text composition, an INDEX command may not appear in the output element in exactly the same location it had in the input element.
INSERT text	Inserts the text, which begins in column 13, into the table of contents.
LIST	Turns on the listing, cancelling the effect of an UNLIST directive or the N option on the @DOC processor call statement.
OPTION k,mode	Allows clearing or setting any of the option letters from the DOC processor call statement (see Table 5-1), where "k" in column 8 specifies the option affected, and "mode" is either ON or OFF. This command can perform the same functions as LIST, UNLIST, RIGHTM, HYPHEN, and UNHYPH. Changing the SIR options, the C options or the M option is not meaningful. The B option is checked only at the end of the document, so the last setting is the only one used.
REMAIN n	Ejects paper for a new page if fewer than n lines remain on the page, where n is an unsigned integer in columns 8-9. If n is omitted or is zero, the paper is not ejected.
RIGHTM mode	Turns the right margin alignment on or off, where mode is either ON or OFF, beginning in column 8.
ROMAN mode	This command allows pages to be numbered with either Roman or Arabic numerals "mode" is ON or OFF, with the obvious meanings. It is suggested that Roman numerals be employed only with the F option set, since they take up a good deal more space in the title area than Arabic numerals.
SINGLE	Selects single spacing mode.

Table 5-4. Listing Control Directives (continued)

Directive	Description
SMCON text	Performs a symbiont control (APRTCN$/APRTCA$) function using the specified text. When used to request special forms or margins for a document, this command should immediately follow the title image (the first image). If several documents are being stacked in an alternate print file by use of the B option, this command is most useful in processing the first of the stacked documents, as the symbiont control function requested will then apply to the remainder of the alternate file.
SPACE *n* .	Spaces paper *n* lines, where *n* is an unsigned integer in columns 7–8. If *n* is omitted, 1 is assumed. A blank image encountered in the input element will be converted to a SPACE directive with *n* omitted. If *n* is 0, no spacing is performed. The SPACE 0 directive is used as a no–operation directive for control of text composition.
UNLIST	Turns off the listing.
UNHYPH mode	Turns the hyphenation removal routine on or off, where mode is ON or OFF, beginning in column 8.

When in the UNLIST mode, the EJECT, REMAIN, and SPACE directives are recognized for purposes of keeping the correct page numbering, but they do not cause any paper movement.

5.4.4. Text Control

The DOC processor assigns proper margins to the text that appears on the listing (and in the output element). This is accomplished by moving words (defined as strings of consecutive nonblank characters) and string of blanks from the input line to an output area until the output area is full. The output line is then printed and sent to the output element (if any). Control over this process is achieved by using the COLUMN, LENGTH, HYPHEN, UNHYPH, RIGHTM, and SPACE 0 directives, some of which have been described above.

The text movement and line composition are terminated under the following conditions:

- Start of a new paragraph
- Recognition of a directive

A new paragraph is recognized by the presence of either an indented line or a hanging line. An indented line is one on which the first nonblank character is to the right of the current left margin, while a hanging line is one on which the first nonblank character is to the left of the current left margin. All other text lines (i.e., those on which the first nonblank character is in the current left margin column) are called normal lines. Only normal lines are processed fully by the text compositon routine, although an indented or hanging line may be affected on the right by movement of words to or from a succeeding normal line.

In view of the above, it is necessary to maintain careful control of the left margin when text is indented, lest the output document be composed improperly. Similarly, when indenting text for an example, as the following:

    This is an example.

it is necessary to prevent the following normal text line from being moved onto the example line. This is done by using a SPACE 0 directive as a no–operation directive following the example line to force the next normal line to begin on a new line.

Control over the left margin is provided by the COLUMN directive, while LENGTH provides control over the right margin. These commands are described in Table 5-5.

Table 5-5. COLUMN and LENGTH Directives

Directive	Description
COLUMN n	Adjusts the left margin to column n, where n is an unsigned integer in columns 8-9. If n is omitted, the value 2 is assumed. Until a COLUMN directive is encountered, 2 is the default value at the start of a document. A value less than 2 is illegal, as are excessively large values; the permissible maximum depends on the form in use and the current LENGTH specification. Note that DOC expects that those input lines which are to be treated as normal text lines will begin in column n. Text beginning in column 2 will not be moved over to column n (n>2) but will be treated as a hanging line instead.
LENGTH n	Defines the current line length to be n where n is a signed or unsigned integer in columns 8-11. If n is unsigned, an absolute length is assigned. If n is signed, the new line length is relative to the line length previously in effect; a preceding plus sign increases the line length by n, while a preceding minus sign decreases the line length by n. Note that the LENGTH command pertains only to the listing and output element; input lines may be shorter or longer as desired (up to 132 characters maximum). It is suggested that the relative (signed integer) form of the LENGTH directive be used to obtain indentation of text on the right. Since there is a possibility of processing a document on various devices with standard line lengths of various sizes, the relative form will provide correct results in all cases.

The SPACE 0 directive (see Table 5-4) is used to indicate that the next normal line is to begin on a new line. A SPACE 0 directive will be generated automatically preceding any line which contains a 0 in column 1. The 0 will be replaced by a blank in column 1 in the output element.

5.4.4.1. Hyphenation Removal

If the automatic hyphenator is used when an output element is produced, the output element is hyphenated just as the listing shows. Since the hyphenator uses a heuristic, not a dictionary, the hyphenation may be performed at places that are not correct, according to common usage. For this reason, the G option and UNHYPH directive have been provided to remove all hyphenation at the end of lines. In occasional cases, a word at the end of the line will actually require the hyphen internally (such as 'right-handed'). To avoid losing the hyphen altogether, the hyphen should be replaced by two adjacent hyphens. Then, only one will be removed, leaving one to be retained inside the word.

Hyphenation removal will not be performed between TABLE and TABLE END directives, not between FIGURE and FIGURE END, unless it is specifically requested by the UNHYPH directive. The mode in effect when TABLE or FIGURE was encountered will be restored when TABLE END or FIGURE END is reached, independent of any changes inside the table or figure.

5.4.4.2. Right Margin Alignment

Documents being prepared for final distribution may be printed using the right margin alignment algorithm. This is activated by the R option on the @DOC Processor call statement or by the RIGHTM directive. Only the printed output is right-aligned, not the output element, unless the O option was specified on the @DOC Processor call statement.

The comments about hyphenation removal inside tables and figures apply equally to right margin alignment.

5.4.5. Editing Control

Line image editing is provided by standard correction lines in the format common to all processors (see Volume 2-3.2).

Character string editing is the manipulation of character strings on a particular line image of the input (text or control directive) in the run stream or input element. Character editing directives have an ampersand (&) in column 1, but are otherwise free form. They immediately follow the line to be altered.

Examples:

- To correct line 18 of the input element:

  [-18]

  &...

- To correct an image in the run stream.

  [<BAD line>]

  &...

The character correction lines have four formats. In the following format descriptions, the slash (/) is used as a character string delimiter; however, any nonblank character which does not appear in the text on the line may be used. Only one character may be used as the delimiter on a card. The first nonblank character on the card (after the ampersand in column 1) is the delimiter character for that line and any number of blanks may precede it.

Format 1:

&/<oldtext>/<newtext>/

The line being corrected is searched for the first occurrence of old text which is replaced by new text.

Format 2:

&/<oldtext>/<newtext>/*

Similar to format 1, except that every occurrence of old text is replaced by new text. The asterisk must immediately follow the third delimiter. If a character other than asterisk is used, format 1 is assumed. However, this does not preclude the use of the asterisk as the delimiter.

Format 3:

&//<column-nbr>/<newtext>/

The new text string is inserted in the line to be modified beginning at the column specified by column-nbr.

The insert overlays any previously existing characters. The first two characters after the ampersand are consecutive delimiters, no characters may separate them. The column-nbr may be stated or omitted. If it is omitted or contains a nonblank or nondigit character, the value assumed is the value on the most recent COLUMN directive.

Format 4:

```
& // <column-nbr>/
```

The first nonblank character, after column 1, on the line being altered is placed in the column specified by column-nbr. The entire character string is shifted to satisfy this specification. This format is similar to format 3, except that there must be exactly three delimiters on the card.

Character correction directives may alter the length of the line image. If it is shortened, the unused area is filled with blanks. If it is lengthened, the excess is checked to see if it is entirely blank. If so, it is ignored. Otherwise, the line is continued in such a way that a word is not broken between line images. This means that each line image ends with a string of blanks and the first word of each continuation line begins in column n, with columns 1 through n-1 containing blanks. The length of a word must not exceed 67 - n characters, where n is the column number on the current COLUMN n directive. A line may expand to a maximum length of 600 characters by using more than one character correction directive. The line to be changed is followed with as many character insertion directives as necessary with the changes made in the order of the directive encountered.

Examples:

- The following is a portion of a document:

```
43 ...UNARMED AND UNPREPARED TO BEET THEM BACK.  'TIS

44 SAID THAT RICHMOND IS THEIR ADMIRAL.  AND THERE THEY

45 HULL, AWAITING BUT ETH AID OF BUCKINGHAM TO

46 WELCOME THME ASHOR...
```

There are several typographical errors in the document which can be corrected as follows:

```
@DOC,DLMU       RICHARD/III
-43
&/BEET/BEAT/
-45
&/ETH/THE/
-46
&/THME/THEM/
&/ASHOR/ASHORE/
&//2/           (WILLIAM SHAKESPEARE)
&/WILLIAM/WM./
```

This results in the following output, assuming no line number changes:

```
43* ...UNARMED AND UNPREPARED TO BEAT THEM BACK.  'TIS

44 SAID THAT RICHMOND IS THEIR ADMIRAL. AND THERE THEY

45* HULL, AWAITING BUT THE AID OF BUCKINGHAM TO WELCOME

46* THEM ASHORE...

47      (WM. SHAKESPEARE)
```

- The following illustrates an appropriate use of the COLUMN directive to produce a blocked, left-justified description following each item in a list.

The input for a portion of a document is:

```
OPTIONS AVAILABLE ARE:
COLUMN 11
      S -       THIS OPTION IS USED TO INDICATE THAT
                THE FILE IS TO BE SENT TO A SPECIFIC DEVICE
SPACE 1
      C -       USED FOR A REMOTE PUNCH FILE.
```

The output generated is:

```
         C1       C10       C20       C30       C40       C50       C60 C66

   14  OPTIONS AVAILABLE ARE:

   16       S - THIS OPTION IS USED TO INDICATE THAT THE FILE IS TO BE
   17            SENT TO A SPECIFIC DEVICE.

   19       C - USED FOR A REMOTE PUNCH FILE.
```

The proper use of COLUMN and LENGTH directives is essential to the production of a document whose appearance will be satisfactory.

5.5. DOC PROCESSOR DIAGNOSTICS

5.5.1. Error Handling

In case errors occur, an error summary will be printed at the end of the listing. For each error, the summary will include the error type number, the line at which it occurred (which will be 1 for errors occurring during the first pass, before printing begins, such as SIR errors), a descriptive message, and an associated error code, which may be facility bits, I/O status, or etc. A maximum of 100 error messages will be listed; if that limit is exceeded, all subsequent errors will be ignored.

5.5.2. DOC Processor Error Messages

The DOC processor error messages are as follows:

SCRATCH/PRINT/INDEX FILE UNAVAILABLE

A file could not be assigned. This may be the internal scratch file DOC$, the index file DOC$X (if the X option was used and INDEX directives appear in the input), or the alternate print file specified by field 3 of the processor call card. The error code is the facility bits (see Volume 2- Appendix C for description).

PARTBL NOT INITIALIZED

The @DOC processor call card has an incorrect element specification in field 1 or 2, there is an option conflict, or an @XQT card was used to call DOC. There is no error code.

SIR ERROR RETURN

The Source Input Routine has encountered an I/O error. The error code is the I/O status code.

SOR ERROR RETURN

The Source Output Routine has encountered an I/O error. The code is the I/O status code.

INVALID CONTROL DIRECTIVE

A line with a nonblank in column 1 was encountered in the input, but the line does not specify one of the allowed directives. The code is the first four characters of the line.

COLUMN DIRECTIVE IN ERROR—ADJUSTED

A COLUMN directive specifies a number too large or too small. The value is adjusted to the largest or smallest permissible value, respectively. The code is the incorrect value.

SDF I/O ERROR

An I/O error has been encountered while transferring data to or from the internal scratch file. The error code is the I/O status code.

POSTPR$ ERROR

The Post Processor routine was unable to restore the files assigned by the Preprocessor routine to their original state. There is no error code.

UNDEFINED FORM—HSP ASSUMED

Field 4 of the @DOC processor call statement specified an unknown form. The HSP form is used. The error code is the first six characters of the incorrect specification.

DIRECTIVE SPECIFICATION INVALID

A directive such as LENGTH has specified a value which is not in the permitted range. The error code is the offending value.

'END' NOT PAIRED WITH 'TABLE/FIGURE'

An extra END has been inserted. There is no error code.

'TABLE/FIGURE' DIRECTIVE WHEN ALREADY IN TABLE/FIGURE MODE

An END has been omitted. There is no error code.

CONTINGENCY INTERRUPT

A contingency has occurred, probably due to an internal error, although the '@@X C sequence will also produce this message. The error code is the contents of the contingency packet. Submit a SUR if the error recurs after trying to remedy the cause (e.g. assigning a larger file in case IO 22 error).

NONBLANK STRING EXCEEDS LINE LENGTH OF FORM

A nonblank string was encountered which was too long to fit on a single line and could not be hyphenated by the heuristic normally used (either because the H option was not set or the string contained non-alphabetic characters). The string has been hyphenated at an arbitrary point. If this is not desired, change the line length either in the form definition on the DOC processor call card or through use of the LENGTH command.

TOO MANY SUBHDG COMMANDS—EXCESS IGNORED

More than four SUBHDG cards specifying additional subheadings have been used. This may be caused by the omission of "SUBHDG N". Otherwise, remove the excess SUBHDG card.

FILE OVERFLOW

The specified file is full. If it is the output file, (SO$ or SI$) DOC will continue processing but will not produce output. If a DOC internal file (PSF$ or DOC$X) overflow, DOC will terminate. Either of these files should then be reassigned with a larger granule maximum and the DOC operation repeated.

SORT ERROR

A sort error has occurred while processing the index. Consult the SPERRY UNIVAC 1100 Series Sort/Merge (Subroutine) Programmer Reference, UP-7621 (current version) for the meaning of the error code.

6. Flow Analysis Program (FLAP)

6.1. GENERAL

The Flow Analysis Package (FLAP) provides detailed information about the timing and frequency of the basic intervals of a program in its machine language form.

Within a program, those locations which contain instructions which branch, perform Executive Requests, or perform repeat instructions at some point during execution of the program are called the breakpoints for that execution. An interval is a sequential set of instructions which control enters as a result of an execution break (a jump destination or falling through an ER or repeated instruction) and leaves the sequence at a breakpoint. A basic interval is an interval with at most, one entry and one exit (breakpoint).

The basic interval is a natural unit for the analysis of program flow and timing since it is always entered at its first location and is completely executed upon each entry. The frequency of a basic interval is the number of times which it is entered during the execution of the program.

In software, most of the work is performed by a very small fraction of the program. By isolating these portions of the program and examining their actions, an economical means of determining the inefficiencies of a program is available. FLAP is a system for recording and summarizing the action of a program. By examining the summary, it is possible to easily recognize the areas of a program which are the best candidates for "tightening". In most instances, it is of more value to have a section of the program coded for generality or clarity rather than efficiency. However, most programs have sections which are so frequently used that the efficiency of these sections is of significant importance.

FLAP performs an analysis of instructions executed within a program, an analysis of the frequency and time spent doing executive returns (ERs), and optionally produces a report of I/O activity.

In the FLAP system time is measured in units which are approximately equal to the basic instruction time of a 1100 Series processor; e.g., the time for a load or store instruction referencing an alternate bank is one time unit. Since UNIVAC 1108 (1106,SPERRY UNIVAC 1100/10,1100/20, and 1110 (1100/40)) processors have different timing characteristics, there is a parameter to choose which machine timing is to be used. Instruction time extension due to referencing same bank or referencing sixth or third words is not measured.

The FLAP system is a pair of programs: FLOP is collected with and called from the program to be analyzed. FLOP interpretively executes the program and writes a very condensed description of the actions of the program to the output file FLAP$F. The second program, the FLIP processor, analyzes the information in the FLAP$F file and produces reports which describe in detail where the program is spending its time.

The operating characteristics of FLAP are reasonable enough so as not to discourage use of the system. FLOP is able to gather information about a ten second execution on one half reel of magnetc tape in seven minutes. This is usually sufficient time to analyze a compiler or assembler. Programs that run a long time usually are repeating the same process over many records of information. For these cases, the FLOP$STOP parameter can be set to the number of reels of FLOP output desired before program termination. The time required by FLIP to analyze the information is approximately equal to the time required by FLOP to generate the information.

6.2. FLOW OUTPUT PROCEDURE (FLOP)

FLOP is a program trace which is collected with and called from the program to be analyzed.

The trace is initiated by performing:

```
SLJ FLOP$
```

and normally continues until the activity terminates via an Executive Request to EXIT$, ERR$, EABT$ or ABORT$. When the activity terminates, FLOP closes the FLAP$F output file and always stops via a jump to the terminating ER. FLOP will also stop if an illegal instruction is encountered or if control goes to an address less than 0200.

If the user wishes to shut off the trace for a portion of his program, the call

```
SLJ FLOPE$
```

will stop the trace, but leave the output file open for a later call to start the trace.

If the user wishes to shut off the trace and end the report, but leave the FLAP$F file open for additional reports from the same absolute element, he can perform:

```
SLJ FLOPS$
```

If the user wishes to shut off the trace and close the FLAP$F output file before his activity terminates, he can perform:

```
SLJ FLOPC$.
```

An interface subroutine is available to allow FORTRAN or COBOL programs to call FLOP. The entry points are summarized in Table 6-1.

Table 6-1. FLOP Entry Points

ASM Entry	COB/FOR Entry	Description
FLOP$	FLOP	Open file if not already open. Start trace.
FLOPE$	FLOPE	Stop trace. Leave FLAP$F output file open.
FLOPS$	FLOPS	Stop trace. End report. Leave FLAP$F output file open.
FLOPC$	FLOPC	Stop trace. Close FLAP$F output file.

The output file for FLOP will always be named FLAP$F. It may be pre-assigned by the user as a FASTRAND-formatted file, or as magnetic tape (may be multi-reel). If FLOP receives control and FLAP$F has not been assigned, FLOP will perform a CSF$ request on:

```
@ASG,T  FLAP$F,F8/1/pos/300
```

Several parameters may be passed to FLOP as external definitions, these are:

- FSKIP$

  If zero, then test instructions which skip are treated as execution breakpoints, and thus, define the end of a basic interval. If nonzero, then test instructions are never treated as breakpoints. The recommended value is: FSKIP$ EQU 0

- FJUMPS$

  If nonzero and if a jump to an unconditional jump is encountered, then the unconditional jump is not treated as a basic interval in itself, but is treated as an extension of the previous interval. The recommended value is: FJUMPS$ EQU 0.

- FJUMP$

  If a jump is encountered to $+n where $1 \leq n \leq$ FJUMP$ and if the jump instruction does not contain indexing or indirect addressing, then the jump will not be treated as an execution breakpoint which terminates the current interval. The intention of this parameter is to allow the use of jump instructions when no test/skip instruction is appropriate. The recommended value is: FJUMP$ EQU 0.

- U1110T

  If U1110T = 0, timing is output using tables for 1106, 1108, 1100/10, and 1100/20 instructions. If U1110T = 1, timing is output using tables for 1110 and 1100/40 instructions. The setting is independent of the machine which FLOP is on, but should be set according to which computer the normal program is being measured for. The timing information is passed to FLIP via the FLAP$F file.

- FLOP$STOP

  If zero, FLOP will run normally. If n, and if FLAP$F is a tape, FLOP will stop the trace and close the output file after filling n reels.

- FTIO$

  If not zero, FLOP will output information from all I/O packets so FLIP can analyze IO. If FTIO$ is zero, no IO analysis will be done.

- CHK$SUM

  If not zero, FLOP will checksum each FLAP$F data block, which is 448 words long, by placing the sum of the last 446 words of the block in the first word of the block. FLIP will repeat the checksum and compare the result, treating an error as a read format error. If CHK$SUM = 0, no checksumming will be performed.

The following example is a set of MAP source statements which could be used in the collection of the program to be analyzed:

```
EQU FSKIP$/1,FJUMP$/2,FJUMP$/0
EQU U1110T/1              . USE 1110 TIMING VALUES
EQU FLOP$STOP/2           . STOP TRACE AFTER FILLING 2 REELS
EQU FTIO$/1               . TURN ON IO TRACE
```

The following restrictions apply to FLOP:

1. FLOP must be part of the program's main segment and never be overlayed by an RSEG.

2. FLOP must be located totally in the control bank, and must be based by the active PSR while tracing the program. (FLOP has only even numbered location counters.)

3. Tracing will be destroyed if the main segment is reloaded.

4. Only one activity may be traced at any one time.

5. No activity contingency routines will be traced.

6. The trace will not be active while code in a reentrant processor is being executed.

7. Timing figures may not be accurate if the program executes instructions which are different from or not contained in the absolute element.

8. The EDIT instruction (33–07) cannot be timed, but may be executed.

9. Programs which reference common banks can be traced by FLOP, but common bank code is described as an undefined area by FLIP.

10. Programs collected with no diagnostic tables (Z option on @MAP) can be traced, but only absolute addresses will appear in the reports produced by FLIP. Also, reports will be in terms of instruction counts rather than instruction timing.

11. IOXI$ and IOAXI$ ER's cannot be processed.

A program being traced, functions the same as an untraced program except that:

1. Execution time is slower – FLOP executes approximately 40 instructions for each user program instruction executed. This may affect I/O timing if that is relevant to the program being analyzed.

2. FLOP will terminate via ERR$ if the output file is the wrong device type, or cannot be assigned.

NOTES:

1. *Care should be used when FLAPPING a FORTRAN program which uses NTRAN. The first program call to NTRAN causes an ER FORK$, followed by an ER EXIT$. Thus, the CALL FLOP statement should normally appear after the first CALL NTRAN statement. Otherwise, FLOP will stop the trace when the ER EXIT$ in NTRAN is encountered.*

2. *When FLOP is active, the address of the instruction being processed may be found in the first cell under location counter $(6) in FLOP. It has the external definition PR$. This may be useful if an IGDM or similar error occurs while a program is being traced.*

6.3. FLOW INFORMATION PROCESSOR (FLIP)

Purpose:

The FLIP processor reads the information output by the trace routines and produces several reports.

Format:

@FLIP,options spec1

Parameters:

options — The options are:

- B Batch print format
- R Use alternate key for sorting intervals for report 2.
- G If the G option is set, gaps in the intervals listed in report 1 will be flagged by '**GAP-SIZE = nn'. A count of gaps will always be kept and printed in each section summary in report 1. Only non-executed code internal to each section can be detected and marked.
- M The M option inhibits a tape rewind before reading the input (FLAP$F) file, enabling multi-file tapes. If FLAP$F is a mass storage file, the M option is meaningless.
- P Print the P-address of the start and stop trace instructions.
- X If a fatal error occurs, stop via ER ERR$, with A15 containing the address of the instruction which jumped to the error routine. If the X option is not set, FLIP will stop via an ER EXIT$, enabling the remainder of a batch runstream to be processed.
- S Print summary I/O report
- L List all I/O references in order of occurrence, and print the I/O summary report.

spec1 — specifies the filename containing the data to be processed. If spec1 is absent, the name FLAP$F is assumed.

Description:

At the beginning of the reports generated by FLIP is a summary of the interval information. Included is the summation of (INTERVAL LENGTH)*(FREQUENCY OF INTERVAL) over all intervals. If the FLOP parameters, FSKIP$, FJUMPS$, and FJUMP$ were all set to zero during the FLOP trace, the value of the summation is equal to the number of instructions executed by the program while it was being traced.

FLIP sorts and merges the list of intervals in the FLAP$F file to produce another file containing a list of the intervals and the number of times they were executed. This file is then scanned to produce a list of the basic intervals and the frequencies with which they are executed. FLOP also provides a copy of the object program to FLIP, via the FLAP$F file. By examining the instructions in this copy of the object program, FLIP estimates the amount of time spent in each basic interval. After performing another sort, FLIP is ready to produce the first two reports.

The first report lists all of the basic intervals in the program, sorted according to the location of the basic interval within the program. The column headings are:

INTERVAL	cross reference number so that each entry may be referenced in subsequent reports.
RELATIVE	interval address limits relative to the location counter within an element.
ABSOLUTE	interval address limits.
LENGTH	number of instructions in the interval.
FREQUENCY	number of times the interval was executed.
RAW TIME	total number of time units spent in the interval.
% TIME	percentage of total program execution time spent in the interval.
TIME/INST	percentage of total time spent in the interval, per instruction, executing nonrepeated instructions. This value is a measure of how much work is performed by each instruction.
REP TIME	the percentage of the program time which was spent in repeat mode in the interval.
TOTAL % TIME	a running sum of % TIME. The % time used by any contiguous group of intervals can be found by subtracting the initial value from the final value from entries in this column.

NOTES:

1. *The columns listing RAW TIME or FREQUENCY can be used like a bar graph (each additional digit is an increase in time spent) to find clusters of intervals where a relatively large amount of time is spent.*

2. *The % of program time spent in an interval cluster can be found by subtracting the TOTAL % TIME column for the cluster from the last entry.*

3. *When FSKIP$ = 0 in FLOP, intervals which end with a TEST instruction can be examined to get conditional branch probabilities for optimal ordering of complex conditions.*

In the second report, the basic intervals are listed according to the average instruction timing. The basic intervals near the beginning are the one which if eliminated or modified, would make the greatest improvement in running time of the program. If FLIP is run with the R option set, this report is sorted by gross nonrepeat time. The intervals near the beginning are those in which most execution time is spent in the program. These intervals tend to be long, frequently executed intervals. Because of their length, the probability of being able to make an improvement in execution time for the interval, and thus, a significant improvement in total program execution time, is larger than for short intervals. The column headings for the second report are:

INTERVAL	interval number as listed in report 1.
TIME/INST	same as TIME/INST for report 1.
LENGTH	number of instructions in the interval.

TOTAL	a running sum of interval lengths.
TOTAL % INST	percent of total instructions executed represented thus far in this report.
% OF N.R.	percent of total nonrepeat time spent in this interval.
TOTAL	a running sum of % of N.R.
% OF PROG	percent of total execution time spent as nonrepeat time in this interval.
TOTAL	a running sum of % of PROG.

By comparing column 7 with column 5, it can easily be seen that normally a large percentage of execution time is spent executing a fairly small number of instructions.

The third report lists those intervals containing a repeat instruction (BT, SE, MSE, etc.). FLOP supplies FLIP with timing information about these instructions. Since these instructions are coded quite infrequently, this report is usually very short. The column headings are:

TIME/FREQ	average percent of total time used by the repeat instruction in the interval for one interval execution.
TIME	percent of total program time used by the repeat instruction in the interval.
TOTAL	a running sum of TIME

The fourth report lists execution time totals for each element. Only those location counters in which instructions were executed are listed. This report was designed mainly to assist in optimal memory placement of portions of code in the 1110 (1100/40). The column headings are:

ELEMENT NAME	name of relocatable element collected
COUNTER	location counter within element
BANK NAME	bank name specified (or implicit) in collection for this elt/counter
SUM (INT LNG)*(FREQ)	sum all items in the scalar (dot) product of interval length and frequency, for all intervals in this elt/counter. When FSKIP$, FJUMPS$, and FJUMP$ are all zero, this is the number of instructions executed.
RAW NON-RPT TIME	total number of time units spent in this location counter executing nonrepeat instructions.
% NRT OF TOTAL	percent of total program time spent in this elt/counter executing nonrepeat instructions.
% RPT OF TOTAL RPT	percent of total time spent doing repeat instructions in this elt/counter.
SUM % NRT OF TOTAL NRT	running sum of percent of total nonrepeat time spent executing nonrepeat instructions.

The rightmost column lists the interval numbers for the elt/counter which were assigned in report one.

The fifth report is an analysis of the frequency and time spent doing Executive Returns (ERs). The cost of an ER is taken from a table of values which appears in the EXEC 8 element AAERGTP and which is used to compute SUPs for a run. The cost can be compared on an equal scale with the raw times listed in reports 1 and 4.

Report 5 lists the ERs executed, sorted by raw cost. The column headings are:

ER	The mnemonic code for the function.
FREQUENCY	The number of times the ER located at the address listed was executed.
% ER COST	The ratio of the cost of all executions of this ER at this location, to the cost of all ERs executed in the program.
% OF PROG	Since ER times are not included in reports 1 through 4, this column lists the ratio of ER time charged compared to total program execution time plus total ER time.
ELEMENT	The name of the element the ER occurred in.
ADDRESS	Relative address within the location counter where the ER occurs. If the ER is in a location outside the original absolute element (e.g. in an area attached by MCORE$) this address will be only relative to the PSR, and not to any program element.
LC	The location counter number where the ER occurs.
BANK	The name of the program bank the ER was executed in.
SEGMENT	The name of the program segment the ER was executed in.
SUM % ER	A running sum of % ER COST.

Report 6 is a summary of subroutine calls within the traced program. All SLJ and LMJ instructions are assumed to be subroutine calls in this analysis. No other techniques for calling subroutines are considered. The report is sorted first by subroutine entry location, and then by frequency of the subroutine calls.

The column headings for this report are:

ENTRY ADRS	Relative address within an element of the subroutine entry point. The line of data containing the entry address describes the subroutine.
ELEMENT	Name of the element containing the subroutine entry point or the subroutine call.
LC	Location counter containing the subroutine entry point or the subroutine call.
BANK	Bank containing the subroutine entry point or the subroutine call.

SEGMENT	Segment containing the subroutine entry point or the subroutine call.
CALL ADRS	Relative address within an element of the subroutine call. The line of data containing the call address describes the subroutine call.
FREQUENCY	The frequency of execution of the described subroutine call.
SUM FREQUENCY	A running sum of the calls of the described subroutine.
TYPE	The instruction which was used for the subroutine call. (LMJ or SLJ)
ENTRY NAME	The externalized name of the subroutine entry point. This will be blank if no name can be found in the absolute element diagnostic tables.

Report 7 is a report of IO activity as seen by the FLOP trace. To get this report, the FLOP parameter FTIO$ must have been set non-zero and FLOP must have seen at least one IO request. The IO report has two parts: a time ordered list of every IO request seen by FLOP, and a summary of IO requests. The IO trace is selected by using the L-option on the FLIP processor call. The IO summary is selected by using the S-option (or the L-option) on the FLIP processor call. In order to make the IO Trace Report more compact, segment numbers and bank descriptor indices are printed instead of segment and bank names. To provide a simple correlation between names and number, the IO Trace Report is preceded by a simple listing of BANK NAME - BDI, and SEGMENT NAME - SEG#. IO requests using the arbitrary device handler are not reported.

The IO items are sorted by filename, function, requested transfer count, and actual transfer count. All matching entries are combined to compute total words transferred and frequency of read-before-write occurrences in the IO Summary Report.

The column headings for the IO Trace Report are:

FILENAME	Name from words 0,1 in IO packet.
FUNC	Mnemonic for IO operation performed. (See Volume 2 Table 6-1)
#-ACW	Word transfer request or number of S/G ACW's.
#-XFR	Number of words actually transferred by the IO request.
MS ADRS	Mass Storage Address or contents of word at IO packet +5.
RBW	An asterisk (*) appears if criteria for read-before-write are met. See discussion on FLIP parameter entry.
ER TYPE	Name of ER used to request the IO operation.
ER ADRS	Program relative address of the ER used to request the the IO operation. The SEG# and BDI for the ER are listed in the next two columns.
BUF ADR	Program relative address of buffer used for IO transfer or address of S/G access words. The SEG# and BDI for the buffer address are listed in the next two columns.

PKT ADR	Program relative address of the IO packet. The SEG# and BDI for the IO packet are listed in the next two columns.

The column headings for the IO Summary Report are:

FILENAME	Name from words 0,1 in IO packet.
FUNC	Mnemonic for IO operation performed.
#-ACW	Word transfer request or number of S/G ACWs.
#-XFR	Number of words actually transferred into or out of the users buffer space.
TOTAL XFR	Total number of words transferred.
FREQUENCY	Frequency of IO requests for this file, function requested and final word count.
#RBW	Frequency of RBW criteria match for this file, function and word count.

Read-Before-Write (RBW) Computation.

FLIP has the capability of recognizing a read-before-write (RBW) situation and printing all such instances as part of the IO Trace Report and IO Summary Report. The information needed to recognize RBW includes IO device type, prepping factor, and simulated device type (FASTRAND or word addressable drum format). If RBW computation is desired, the RBW criteria are passed to FLIP on parameter statements in the runstream following the FLIP processor call.

FORMAT:

Filename Prepfactor Adrs Mode

Parameter:

Filename	Is the internal filename used in the IO packets.
Prepfactor	Is the number of words in a disc record. If this value is omitted, the value 112 is assumed.
Adrs Mode	Is the number of file relative addresses in a disc record. If the file is FASTRAND formatted, this field may be omitted and the value will be assumed to be Prepfactor/28. If the file is formatted as word-addressable-drum, then this value must be entered as the same number as Prepfactor.

If no parameter statements are present, RBW will not be checked. If any parameter statements are present, files named will be checked for RBW according to the criteria specified and files not named will be checked for RBW according to the parameters specified on the last parameter statement. Since FLIP does not know which user files were tape and which were mass storage, improper RBW values may appear for tape files unless Prepfactor and Adrs Mode are set to one(1) for tape files.

Example:

```
FILEF       28
FILEW       56  56
FILET       1   1
FILED
```

where:

FILEF is on FASTRAND, FILEW is a word addressable drum file simulated on a disc prepped at 56 words/record, FILET is a tape, FILED and all other files are on a disc prepped at 112 words/record.

If all files used by a program are on a disc prepped at 112 words/record, a blank card following the FLIP processor call will be sufficient to cause RBW checking on all files.

6.4. ERROR MESSAGES PRODUCED BY FLOP

- COULD NOT LOCATE ABS ELT

  FLOP could not find the ABS of the program being executed in the program file pointed by the LOAD$ filename in the PCT.

- COULD NOT ASG FLAP$F OUTPUT FILE

  File FLAP$F was not assigned when FLOP got control, so FLOP did ER CSF$ with the image

  ```
  @ASG,T  FLAP$F,F8/1/POS/300
  ```

  status bits in A0 give reason for FAC REJECT.

- ILLEGAL OUTPUT DEVICE FOR FLAP$F

  Device assigned for file FLAP$F was not tape or FASTRAND formatted mass storage.

- ERR nn ON FILE filename

  Status = nn received from FLOP IO request on file named. This is a warning message only and FLOP will not stop.

6.5. ERROR MESSAGES PRODUCED BY FLIP

- PROCESSOR CALL ERROR

  FLIP must be called as a processor, not with @XQT.

- ABNORMAL RETURN FROM READ$

  FLIP must be called as a processor, not with @XQT.

- TOO MANY SPECIFICATIONS

  Too many specifications supplied on FLIP processor call.

- SPECIFICATION ONE SHOULD BE A FILE NAME

  Element name was specified for input file.

- UNABLE TO ASSIGN THE [INPUT/ABS.] FILE-STATUS: nnn

  Could not assign either the file named on the processor call or the FLAP$F file, or could not assign a scratch file to contain the absolute element of the program being FLAP'ed.

- INPUT FILE IS NOT FASTRAND FORMAT MASS STORAGE OR MAGNETIC TAPE

  FLAP$F or file specified on FLIP processor call is not on a usable device.

- COLLECT WITH FLOP LEVEL 4R1A OR LATER

  The levels of FLOP and FLIP used are incompatible.

- ERR nn ON FILE filename

  IO error status = nn occurred on file named. FLIP will continue unless 10 IO errors have occurred, in which case FLIP will error terminate.

- INTERNAL ERROR

  Bad data or bug in FLIP caused abnormal condition in FLIP. Save register dump and PMD for analysis. Register A15 contains relative address in the element FLIP where error was discovered.

- RANDOM FILE PACKAGE ERROR

  IO error on FLIP scratch file or bug in FLIP. Status code in A2 gives error type. See RFP$ symbolic listing for details.

- ERROR READING ABS ELT WRITTEN BY FLOP

  Unexpected formatting of absolute element being analyzed. Use standard collector.

- READ FORMAT ERROR ON FILE BUILT BY FLOP

  This is usually caused by improper usage of FLOP or a program bug causing data destruction in FLOP output buffers.

- HIT UNEXPECTED END-OF-FILE * SOME DATA MAY HAVE BEEN LOST

  Either FLIP encountered end-of-data due to FLOP$STOP parameter being set in FLOP, or FLIP encountered an IO error reading the FLAP$F file.

7. LIST Processor

7.1. INTRODUCTION

This section describes the LIST processor which produces an edited listing of any type element.

7.2. @LIST

Purpose:

Produces an edited listing of any type of element. The LIST processor is called by the @LIST control statement.

All parameters in the @LIST control statement are optional except eltname-1.

Format:

@lable:LIST,options eltname-1(cycle-1),...,eltname-n(cycle-n)

Parameters:

options — If neither A, O, R nor S is specified, S is assumed. All four options are allowed and all elements of the specified type found will be edited.

A - Absolute elements

O - Omnibus elements (dumped in octal)

R - Relocatable elements

S - Symbolic elements

The D option is used with A, R, O or S to dump the elements in octal. The dump is not edited.

If the R option is present C, E, L and X have the following meanings in relation to the relocatable's tables:

C – dump the Control Information Table

E – dump the Entry Point Table

L – dump the Location Counter Table

X – dump the External Reference Table

If the A option is present E, N, V and W have the following meanings in relation to the absolute's diagnostic tables:

E – dump Entry Point Name Table

N – dump Segment, Bank, Element and Location Counter Tables

V – dump Absolute Value Table

W – dump Static Diagnostic Walkback Table

In addition if I is specified with the A option and text is dumped, the instruction format editing will not be done to the text.

If the P option is present C, E, L, N, V, W and X are assumed.

If the T option is present the text will also be dumped.

If none of C, E, L, N, V, W, X, P or T is on, both P and T are assumed.

eltnames — Specifies the elements.

cycles — Specifies symbolic cycles whose line numbers are desired. This parameter is used only with the S option.

Description:

The edited listing contains the following information for each type of element:

- Symbolic elements
  - Every SDF image in the element, including control images, is printed with the length and relative word address of the image.
  - The line numbers of the symbolic images belonging to the specified cycle or, if none, to the most recent symbolic cycle are printed. The cycle information for all symbolic images is printed.
  - If the symbolic element is an Assembler, COBOL, or FORTRAN procedure, the appropriate procedure name table is printed.
- Relocatable Elements
  - Each text word is printed as 12 octal digits. The j-field (bits 29–26), a-field (bits 25–22), x-field (bits 21–18), and h–, i-fields (bits 17–16) are printed below the text word.

- The following abbreviations are used when the relocation information is printed:

  LA – Left address (bits 33–18)

  LC – Location counter

  LH – Left half (bits 35–18)

  RA – Right address (bits 15–0)

  RH – Right half (bits 27–0)

  XR – External reference

■ Absolute Elements

- Each text word is printed as 12 octal digits (see first entry under relocatable elements).
- The following abbreviations are used when the relocation information for the relocatable segments is printed:

  L – Left half relocated

  R – Right half relocated

■ Files

The edited listing contains the following information for files.

The listing of a file operates in an interactive mode. If on the initial read a @EOF is encountered, LIST will then dump the entire contents of the file in octal.

A blocking factor can be specified on the processor call in the version field of the file name, i.e., 'FILE./23'. The maximum blocking factor allowed is 896 words (half a track). If an illegal blocking factor is encountered the default is 28. (If the blocking factor is not specified on the call card, no blocking of the output will be done.)

```
@LIST  FILE1./56, FILE2., FILE3./100
```

FILE1 will be listed with blocking factor of 56. A @EOF will terminate listing of FILE1 and proceed with the listing of FILE2 (no blocking factor). A @EOF will terminate listing of FILE2 and proceed with listing of FILE3 with blocking factor of 100. A @EOF will then terminate LIST.

The command characters and explanation follow for the interactive LIST. All blanks are ignored.

L – Sets the line length printed in characters. The length must lie between 36 and 132 characters.

FORMAT: 'L D' – where D is a number (leading zero implies octal).

The format for the edit statement is as follows:

[#] [*e*] [*g*] *a* [/ [*g*] /]

\# – Prints the word address in the file of the first word of the line printed. (If not present the address is printed as sector and offset).

e – The type of editing to do on line. (The default type of editing is octal).

A = ASCII, D = DECIMAL, F = FIELDATA, O = OCTAL

g – Type of number following (default for address is sectors and for length is words).

S = SECTORS, W = WORDS

a – Address to edit (first word). The address must be specified and must be a number (leading zero implies octal).

/ – Implies a length to dump follows. (Default length is one).

l – Length to dump. L must be specified if '/' is present and must be a number (leading zero implies octal).

EXAMPLES:

0	– would dump word zero of the file in octal
F0	– would dump word zero of the file in FIELDATA
A0/2	– would dump word zero and word one in ASCII
O0/S1	– would dump sector zero in octal
L60	– would set line printed length to 60 characters
F0142000/S2	– would dump 2 sectors starting at sector 1792 in FIELDATA
#1792/S1	– would dump sector 1792 in octal with decimal word addresses.

A number of editing statements may be input on one line as follows:

F 0 * 1792 * F 0142001 * 0142002 * F 0142003 / 10 (The * is any character that is not recognized by LIST).

Assuming the above is a program file and the element at sector 1792 is in SDF format, the output will look something like this:

0	0	**PF**
1792	0	500130000000
1792	1	*SDFF*
1792	2	(Some SDF control image in octal – say 001200000000).
1792	3	(10 words printed out in FIELDATA format).

Two spaces are inserted between every word edited to the output.

USER COMMENT SHEET

Comments concerning the content, style, and usefulness of this manual may be made in the space provided below. Please fill in the requested information.

Requests for copies of manuals, lists of manuals, pricing information, etc. should be made through your 1100 Series site manager to your Sperry Univac representative or the Sperry Univac office serving your locality.

System: _____

Manual Title: _____

UP No: _____ Revision No: _____ Update: _____

Name of User: _____

Address of User: _____

Comments:

CUT

FOLD

FIRST CLASS

PERMIT NO. 21

BLUE BELL, PA.

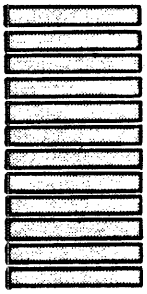
BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

SPERRY UNIVAC

SYSTEMS SUPPORT
ATTN: INFORMATION SERVICES M.S. 4533
P.O. BOX 3942
ST. PAUL, MINNESOTA 55165



CUT

FOLD