UNIVAC 1100 RELOCATING LOADER

4.2. UNIVAC 1100 RELOCATING LOADER

4.2.1. Introduction

The Univac 1100 Series Collector is a very powerful tool for program development. MAP includes facilities for segmentation of large programs, including automatically loadable segments, dynamic segments, relocatable segments and special segments for use in the Operating System itself. MAP also provides facilities for producing programs consisting of one or more named program banks, with complete flexibility in the assignment of addresses to banks and in the ordering of elements and parts of elements within a bank or segment. In addition MAP provides facilities for combining relocatable elements into a single relocatable element, correcting relocatable elements, inserting SNAP dumps and controlling the results of library searches.

Unfortunately, this wealth of capability is not without its price. For very simple programs which use only the most basic Collector functions, the overhead introduced to support the sophisticated functions makes collection an expensive process. The extent to which MAP can be made to operate efficiently for small programs is limited by the need to provide for all the special cases which give MAP its power.

The purpose of the relocating loader, LOAD, is to provide an improved performance collection and execution facility for those programs which do not require some of the more expensive features of the OS 1100 collector. The loader was originally developed to satisfy the need for a quick execution (load-and-go) facility for the PL1 compiler. It has since been generalized and is now equally applicable to any program which meets the restrictions detailed in a later section. The basic function of the loader is to combine the relocatable elements in the run Temporary Program File, TPFS, with those of up to three libraries to produce an executable program, which is then executed. The restrictions of the loader result from its purpose:

- The program consists of a single D-bank; it can reference common banks, but may not itself be banked.
- The program is not segmented.
- The maximum size of the program is reduced by the size of the loader and its tables; the reduction is on the order of 16K words (32 core blocks).
- The collector's element selection and placement commands are not available.

Advantages

The advantages of the loader result from the same design considerations as the restrictions just noted. Because the program is loaded directly to memory, no scratch files are assigned or freed. Because the program is neither banked nor segmented, there is no output file to assign or free and no output need be performed prior to execution of the program. Because the loader has no source language, the overhead of acquiring and parsing the source input is absent, and decisions on element selection and placement are simplified.

The following comparison of collection and loading times is for a medium-sized PL/I program (1300 statements, 1900 lines). The comparison is between MAP level 29R1Q1 and LOAD level 3R1. The times given are SUP totals, scaled to hours, minutes and seconds, and core block SUPs, scaled to hours, minutes and seconds at a block size of 32768 words (64 core blocks). The last column gives the times for a 3R1 loader with the PL/I GET DATA, PUT EDIT and program initialization routines loaded as a nucleus, and, of course, do not include the time needed to build the nucleus.

	MAP	LOAD	LOAD (Nucleus)
TOTAL SUPS	2:17.092	0:27.545	0:10.334
CAU SUPS	0:04.272	0:01.530	0:00.957
I/O SUPS	1:57.568	0:16.234	0:05.272
CC/ER SUPS	0:15.251	0:09.784	0:04.559
32K BLOCK SUPS	1:19.317	0:48.098	0:24.829

4.2.2. LOAD Functional Characteristics

This section describes the functions of LOAD and the options available when it is used. The following subsections describe the input and output of the loader and the way in which loader options are specified.

4.2.2.1.1. Input Sources

The loader has four sources for relocatable elements:

- TPF$
- RUNLIB$
- SYS$*tttLIB$ (See TYPE option)
- SYS$*RLIB$

The loader reads all relocatable elements from TPF$; if there are none, the load is aborted. Next the library files are searched; naturally, the library files must be prepped. If a file with the internal name RUNLIB$ is assigned to the run, it is searched to resolve external references in the elements from TPF$. After RUNLIB$, the appropriate language library is searched. If no language is specified (see TYPE option), the SYS$*tttLIB$ file is not searched. The loader will search only one language library; if a second is needed, it must be assigned as RUNLIB$. After the language library, the loader searches SYS$*RLIB$ to satisfy any remaining external references.

4.2.2.1.2. Program Format

The program is loaded in a format different from that produced by the OS 1100 Collector. The I-bank contains the loader; on single PSR systems (1106, 1108, 1100/10, 1100/20), the loader is followed by its tables in the same I-bank. The main PSR D-bank contains the program. On dual PSR systems (1110, 1100/40, 1100/80), the tables are kept in a D-bank based on the utility PSR which starts at 0740000. Thus on single PSR systems, the loader and its tables reduce the address space available to the program; addresses less than 040000 are not available to the program. On dual PSR systems only the loader itself occupies program address space; the tables can be overlapped if the program expands via MCORE$.

The loading sequence also differs from that of the OS 1100 Collector. The order of location counters is:

- Odd location counters
- Common blocks
- Even location counters

Within these groups, elements occur in the order in which they are encountered by the loader. If the loader contains a preloaded library nucleus, the sequence is:

- Nucleus odd location counters
- Nucleus common blocks
- Nucleus even location counters
- Program/library odd location counters
- Program/library common blocks
- Program/library even location counters

4.2.2.1.3. LOADER Options

In order to leave the option letters available for communication with the loaded program, the loader uses the "file.element" fields of the processor call for option specification. The available option keywords are:

- MAP – Print a memory map.
- NUCLEUS – Create an initialized loader.
- TYPE – Specify language library.
- NODEBUG – Delete loader tables.

The options are specified in the element name fields of the @LOAD line, in any order. For example:

Format, Processor Call

```
@LOAD   MAP, NUCLEUS, TYPE/DGT
```

This processor call requests that a memory map be printed, that a new loader with a nucleus be generated, and that the second library searched be SYS$*DGTLIB$. The generated loader will be named LOAD and it will be left in TPF$.

4.2.2.1.3.1. MAP Option – Print Memory Map

The loader lists the names of the elements loaded with the first and last address assigned to each location counter. Following the list of elements, the program's common blocks are listed with the first and last address assigned to each. This listing is produced only if the MAP option is specified.

4.2.2.1.3.2. NUCLEUS Option – Create Initialized Loader

The loader writes an absolute element in TPF$ which contains a loader with the RB's from TPF$ (and all implied library routines) already loaded. The name of the created absolute element is specified as part of the option: "NUCLEUS/name". If name is omitted, the name of the absolute will be LOAD. When the NUCLEUS option is specified, the loaded program will not be executed. This facility can be very useful if a library nucleus can be identified which needs to be loaded for all, or most, programs using the loader. The generated loader is used in exactly the same way as the loader from the distribution tape. Thus, all the options and facilities described in this manual are available for a loader with a library nucleus as described here. Naturally, the time needed to load a program is considerably reduced if a large part of the library which it needs has already been loaded.

This facility can also be used to generate special purpose processors in which the main program and most of the library have already been loaded and only a few subroutines are to be provided when the processor is called. This can be useful in educational settings, where a student's routine can be tested in a standard environment. And since nearly all of the required library routines could be preloaded in such a setting, the cost for each student run could be significantly reduced.

4.2.2.1.3.3. TYPE Option – Language Specification

The TYPE option takes the form "TYPE/ttt; ttt is a three character language indicator which is used in the language library file name "SYS$*\tttLIB$." If the indicator is not three characters long, the TYPE option is ignored. If TYPE and NUCLEUS are specified together, the generated LOAD processor will assume the language specified.

4.2.2.1.3.4. NODEBUG Option – Delete Loader Tables

In the process of loading a program, LOAD creates a set of tables which correspond roughly to the diagnostic tables produced by the collector. However, these tables are resident in memory and occupy on the order of 10K words (20 core blocks). If they are not needed during execution of the program, they can be deleted by use of the NODEBUG option. Note that, while NODEBUG reduces the execution time size of the program, it does not increase the addressing space of the loaded program. The space occupied by the loader tables does not become available for use by the program.

4.2.2.1.4. SPECIAL SYMBOLS

The OS 1100 Collector defines fourteen special symbols; the Loader also defines these symbols, as well as five others. Certain of these symbols are always defined to be zero. Those are ENTRY$, COMMN$, XREF$, SLT$, FRSTI$ and LASTI$. The symbols FRSTD$, LASTD$, FIRST$, LAST$, BDI$, BDIREF$, BDICALL$, IBJ$ and DBJ$ all have their usual meanings, although restricted by the fact that the Loader creates a one bank program.

The additional symbols defined by the Loader are MIND$, DAVL$, LOAD$COMMON, LOAD$ENTRY and LOAD$ELEMENT. The first of these, MIND$, is not actually defined by the Loader; MIND$ is defined with an absolute value EQU by the program and used by the Loader. The value defined for MIND$ specifies the minimum amount of storage to be available following LASTD$; when the Loader expands the D-bank prior to reading the RB text, it will provide enough room for the space requested by MIND$, if possible. DAVL$ is then defined as the number of words between LASTD$ and the end of the D-bank. LOAD$COMMON, LOAD$ENTRY and LOAD$ELEMENT are the addresses of the first entry in the common block, entry point and element tables. If the NODEBUG option is specified, LOAD$COMMON, LOAD$ENTRY and LOAD$ELEMENT are zero. Note that these special symbols are not accessible to high-level language programs; they must be referenced through assembly language.

Because the Loader does not produce an OS 1100 Collector format absolute element, the OS 1100 RLIB$ routines for conversion of addresses, BDI's and segment indices do not function correctly with the Loader. These routines are described in Sections 2.2.2 through 2.2.8 of the Sperry Univac 1100 Series Executive System Programmer Reference, Volume 4, System Utility Programs, UP-4144.4. The routines involved are CRELAD$, CBN$, CSN$, CABSAD$, CBX$, CSX$ and CSYMVL$. They all require use of the absolute element's tables, and are thus not compatible with the Loader. Equivalent functions are provided by an element distributed with the Loader, called LOAD$DECODE. The names and calling sequences are the same as the RLIB$ routines. The functional differences result from the special execution environment provided by the Loader. Where UP-4144.4 indicates that an SLT$ index is returned, LOAD$DECODE always returns zero. CRELAD$ never takes the error return; there is no I/O to fail. CBN$ always returns "$IBANK" or takes the error return. CSN$ and CSX$ always take the error return. The reinitialization specified for CABSAD$ and CRELAD$ in the case of checkpoint restart is not necessary with LOAD$DECODE; it will cause no problems, so it should probably be specified if checkpoint restart is to be used. In order to avoid possible run time problems, LOAD$DECODE, which occupies about 100 words, should be in the nucleus of any preinitialized Loader.

4.2.2.1.5. RESTRICTIONS

The major restriction on use of LOAD lies in program debugging. Since there is no absolute element, and thus no diagnostic tables, PMD is limited in dumping programs loaded by LOAD. The storage addresses and values produced by PMD are correct, but element names and location counters are not displayed.

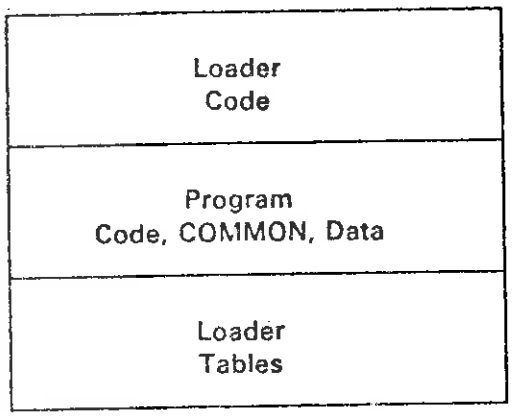
If a loaded program references a common bank, the common bank's address space must not overlap that of the program. This should present no problem on single-PSR machines or for users of PCIOS or the mathematical library common banks, but use of other common banks on dual-PSR systems may require remapping LOAD to provide an increased I-bank size limit.

4.2.3. Structure of the Loader

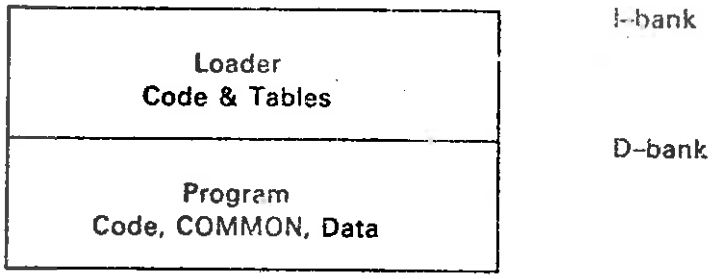
This section describes the organization and operation of LOAD in detail. The following sections will discuss the physical layout for both single-PSR and dual-PSR systems, the overall logic of the LOAD processor, and pertinent parts of the design details.

4.2.3.1.1. Organization

The LOAD processor can be divided into three main parts: the loader code and its static data, the loader dynamic table, and the loaded program. These parts are placed differently in memory depending on the number of banks which can be based simultaneously. On dual-PSR systems, each part occupies a seperate bank:



On single-PSR systems the Utility PSR is not available; the loader tables are moved to follow the loader in the I-bank:



A number of considerations went into the selection of this organization. The loader tables should reside at the end of a bank so that they can be deleted by use of LCORES if they are not needed by the program. This also makes it possible to initially reserve less than the maximum space for the tables and expand them dynamically for larger programs.

In the initial design, the loader was to have resided in the Main PSR D-bank starting at 0200000 with the program in the I-bank starting at 01000. Unfortunately, standard library common banks such as PCIOS require that some of the program's even location counters reside in a D-bank, so that they are addressable from the common I-bank. Because of the initial design, LOAD has been coded to be mappable at addresses greater than 0200000; while this capability is not currently used, it is being maintained. On dual-PSR systems the loader code is placed in the I-bank, rather than preceding the tables in the Utility D-bank, so that the loader code can be deleted before the program begins execution. On single-PSR machines the loader code is deleted only if the loader tables are deleted.

Buffers required for reading the table of contents, entry point table and relocatable element text are allocated in the Main PSR D-bank. When a loader is generated with a preloaded nucleus, the buffer addresses are adjusted to avoid overlaying the preloaded code.

4.2.3.1.2. LOGIC OVERVIEW

The logic flow of LOAD breaks down into the following steps:

- Initialization
- Read RB preambles from TPF$
- Scan entry point tables and read RB preambles from the library files.
- Assign addresses to location counters.
- Adjust D-bank size.
- Define undefined references from the nucleus.
- Read and relocate RB text.
- Free library files.
- If NUCLEUS is specified, save Waiting Reference list, write loader element and quit.
- Set up initial environment, release tables and loader code and start executing program.

4.2.3.1.3. DATA STRUCTURES

The loader is concerned with two primary groups of data structures: the 1100 system relocatable element and the LOAD processor tables. The relocatable element (RB) format is presented in the following section; much of this material is duplicated in UP-4144.4 (current version) in the section on the Relocatable Output Routine (ROR), but the information in UP-4144.4 is not complete. The second section following describes the loader table structure.

4.2.3.1.3.1. Relocatable Element Format

An 1100 Series relocatable element consists of two parts, each starting on a sector boundary on mass storage. The first part is the relocatable text, which contains the actual values to be loaded into the storage areas assigned to the element. The second part is the relocatable preamble, which describes the element. The preamble lists the location counters and undefined symbols used by the element and the entry points which it defines. The preamble also contains additional information describing special features of the location counters.

The preamble begins with a Base Table, which is followed by one or more of the four descriptive tables: the Location Counter Table (LCT), the Undefined Symbol Table (UST), the Entry Point Table (EPT) and the Control Information Table (CIT). The base table contains the index (word offset within the preamble) and number of items for each of the tables. The base table is always four words long as shown:

Index – LCT	Item Count – LCT
Index – UST	Item Count – UST
Index – EPT	Item Count – EPT
Index – CIT	Item Count – CIT

The Location Counter Table specifies the K-bit count, the minimum data area address and the length of each location counter. Each item in the LCT is one word long; the location counter number is the word offset in the table of the corresponding item. The format of the LCT is:

K-bit count	Length of LC 0
minimum data address	Length of LC 1
///////	Length of LC 2
///////	Length of LC n

The Undefined Symbol Table is a list of the symbols referenced by the element which must be defined as entry points by other elements. Each UST item is two words long, containing the symbol as a 12 character Fieldata name, left justified and space filled. An undefined symbol index in a relocation bit stream is the item number, counting from zero; it is multiplied by 2 to obtain a word offset in the UST. The maximum number of undefined symbols is limited by the K-bit count, which ranges from 1 to 10.

The Entry Point Table is a list of the symbols defined by this element for use by other elements, together with defining information for each symbol. The entry point name is 12 Fieldata characters, left justified and space filled. The descriptor word (the third word of an item) has two fields defined. Bit 27 is set to 1 if the entry point value is absolute, and to 0 if the value is to be relocated by a location counter address. Bits 18–26 (Q2) contain the location counter number when the entry point value is relocatable. The value word is just a 36-bit value. The format of a four-word Entry Point Table item is:

Entry Point Name (12 Fieldata characters)		
R-flag	Loc Ctr	///////
Value		

The Control Information Table is used primarily to hold descriptive information about common blocks; its other uses are not currently supported by the Loader. The common block name is 12 Fieldata characters, left justified and space filled. The group number determines the function of the CIT item. Group number 2 identifies named common blocks. Group number 4 identifies the blank-named common block; for blank common, the common block name may by zero, spaces, "BLANK$COMMON" or garbage. The minimum address is the lowest address at which the common block can be allowed to start. The location counter number identifies the location counter which is used in the RB text to identify the common block. The format of a 4-word CIT item is:

Common Block Name (12 Fieldata Characters)		
Group	//////	Minimum Address
Loc Ctr #	//////////	

The relocatable element text is written in fixed length blocks, currently defined by ROR to be 14 words. Each block consists of a sequence of relocation information bit streams, a sequence of text words and a sequence check word. The relocation information is described later. After a word group, if there is insufficient room for the relocation information and at least one text word of the next word group, padding words will be inserted between the relocation information bit streams and the text words to pad the block to 14 words. Padding words contain zero. Within a word group, the text words occur in the reverse of the order in which they are allocated to memory. Thus, within a text block the relocation information scan starts at the beginning of the block and the text word scan starts at the word preceding the sequence check word. T1 of the last word of each block contains two Fieldata characters, starting with "AA" in the first text block and incrementing within the alphabet. The format of a text block is:

Reloaction information for first word group	
Relocation information for second word group	
. . .	
Text words for second word group	
Text words for first word group	
Sequence	//////////

The relocation information for a word group starts with a series of fixed length fields followed by an optional variable length field. The relocation information always starts in the first two bits of a word; the relocation bit stream is padded on the right with zeroes if necessary to fill out its last word. The initial fields of the relocation information are:

Flag — 2 bits. The value of this field determines the type of word group. If this field is 10, this is a normal word group. If this field is 11, then the effective word group address defined by the following fields is the transfer address of the program. If the first bit of this field is 0, then there are no more word groups in this block. The end of a block is also detected when the word from which the relocation information would be taken overlaps the text words.

Address — 16 bits. The word offset from the beginning of the location counter at which the first text word is to be placed.

Count — 9 bits. The number of text words in this group. For flag = 11 (transfer address), count is always zero.

L — 2 bits. This field defines the location counter to be used in determining the address of the word group. The relocation function defines two special location counters: NC1 and NC2. The location counter associated with NC2 is always 2; the value of L determines which location counter is associated with NC1 and which location counter is used in computing the effective address, as follows:

00 NC1 = 0, code 14 → $(0)
01 NC1 = 1, code → $(1)
10 NC1 = 1, code → $(2)
11 NC1 = gc, code → $(gc)

Gc — K bits. The value of K is the K-bit count from the first word of the Location Counter Table in the preamble. This field is present only if the previous field (l) is 11. This field is then the location counter number to be used for NC1 and in which to place the text words.

This fixed relocation information header is followed by a sequence of variable length relocation bit streams. In the following discussion, some special symbols are used which are here defined:

RA — Bits 15–0 of a word.

LA — Bits 33–18 of a word.

RH — Bits 17–0 of a word (i.e., H2).

LH — Bits 35–18 of a word (i.e., H1).

(NC1) — The starting address of the location counter currently associated with NC1.

(NC2) — The starting address of the location counter currently associated with NC2.

u...u — A K-bit field whose value is to be used as an undefined symbol index.

c...c — A K-bit field whose value is to be used as a location counter number.

The relocation bit stream is interpreted by scanning the bits in order. There is a seperate relocation specification for each word in a group. There are two major possibilities for a relocation specification. If the first two bits of a specification are 00, then the specification is precisely two bits long and no relocation is applied to the text word. Otherwise, the specification consists of a sequence of subfields, of which the basic sequence is "ftvd".

The f subfield defines the field within the text word to which the relocation is to be applied. The contents of a field are always sign-extended to 36 bits before relocation arithmetic is performed. The possible values of f are:

1zzz — Field is RA, extend to 19 bits by appending zzz on the left, then sign extend to 36 bits.

0100zzz — Field is LA, extend to 19 bits by appending zzz on the left, then sign extend to 36 bits.

0101z — Field is RH, sign extend with z.

0110z — Field is LH, sign extend with z.

0111llllllrrrrrr

Field is bits llllll to rrrrrr of the word, sign extend with bit llllll.

The t subfield is a single bit which specifies the sign of the relocation. If t is 0, the address specified by v is added to the field; if t is 1, the address is subtracted.

The v subfield is either 2 or 2+K bits long; the value of v determines the address to be added to or subtracted from the field. The values defined for v are:

00	(NC1)
01	(NC2)
10u...u	The value of undefined symbol u.
11c...c	The starting address of location counter c.

The d subfield determines whether or not further relocation of the text word is required; d is either one or two bits long. The value defined for d are:

0	No more relocation; the next bit begins a new text word.
10	More relocation for this word; ftvd follow.
11	More relocation for this field; tvd follow.

4.2.3.1.3.2. Loader Table Format

Element List Item

ELEMENT NAME					1 2
ELEMENT CREATION DATE					3
ELT SEQ NUM	FL#	TEXT LENGTH			4
NEXT ELT LIST ITEM		TEXT ADDRESS			5
# UNDEFINED SYMB		# LOC CTRS			6
K-BIT COUNT		AFLG	UFLG	///	7
ADDR OF LOC CTR 0		LENGTH OF LOC CTR 0			8
. . .		. . .			
ADDR OF LOC CTR N		LENGTH OF LOC CTR N			
//////		ADDR OF UST ENTRY 0			

////// ////// //////	. . .	
//////	ADDR OF UST ENTRY M	

ELEMENT NAME

Words 1 & 2. This is the Fieldata name of the element, 12 characters left justified and space filled. The version name is not retained.

ELEMENT CREATION DATE

Word 3. This is the time and date the element was created, as recorded by ER PFI$.

ELT SEQ NUM

T1 of word 4. This is the sequence number of the element in the table of contents of the file it was taken from.

FL #

S3 of word 4. This is the number of the file from which the element was taken. This is used to determine the file name when the RB text is to be read.

TEXT LENGTH

H2 of word 4. This is the length of the RB text in sectors.

NEXT ELT LIST ITEM

H1 of word 5. This is a pointer to the next Element List item.

TEXT ADDRESS

H2 of word 5. This is the sector offset in its file of the RB text of the element.

UNDEFINED SYMB

H1 of word 6. This is the number (M+1) of undefined symbol slots in this Element List item.

LOC CTRS

H2 of word 6. This is the number (N+1) of location counter slots in this Element List item.

K-BIT COUNT

H1 of word 7. This is the K-bit count from the RB preamble. This is the length of location counter and undefined symbol number fields in the relocation bit streams of the RB text.

AFLG

S4 of word 7. This flag is set to 1 when addresses are assigned to the element.

UFLG

S5 of word 7. This flag is set to 1 when any one of the entry points defined by the element is referenced in the text of an element.

ADDR OF LOC CTR i

H1 of words 8 - 8+N. This is the address assigned to location counter i. This field is also used to temporarily hold the address of the corresponding Common Block Table item prior to address assignment.

LENGTH OF LOC CTR i

H2 of words 8 - 8+N. This is the length of location counter i, as taken from the RB preamble. This field can be set negative during address assignment to flag location counters for which addresses have already been assigned.

ADDR OF UST ENTRY j

H2 of words 9+N - 9+N+M. This is the address of the External Symbol Table item for the element's undefined symbol number j.

Common Block Table Item

COMMON BLOCK NAME		1 2
SIZE	ADDRESS	3

NEXT CBT POINTER	//////	4

COMMON BLOCK NAME

Words 1 and 2. This is the Fieldata name of the common block, 12 characters left justified and space filled.

SIZE

H1 of word 3. This is the size of the common block in words. This is the actual size of the common block; the sizes in the corresponding location counter slots are taken directly from the RB preambles and may thus be smaller than the actual block size.

ADDRESS

H2 of word 3. This is the address assigned to the common block. This address is also copied into all the corresponding location counter slots.

NEXT CBT POINTER

H1 of word 4. This is the address of the next item in the Common Block Table.

External Symbol Table Item

ENTRY NAME		1 2
VALUE		3
LEFT EST POINTER	RIGHT EST POINTER	4
LAST UST POINTER	NEXT UST POINTER	5

////////	ELT ITEM POINTER	6

ENTRY NAME

Words 1 and 2. This is the entry point's Fieldata name, 12 characters left justified and space filled.

VALUE

Word 3. Prior to address assignment this is the absolute part of the entry point value. Following address assignment this is the value of the entry point.

LEFT EST POINTER

H1 of word 4. This is the pointer to the next External Symbol Table item with an entry name less than the name in words 1-2 of this item.

RIGHT EST POINTER

H2 of word 4. This is the pointer to the next External Symbol Table item with an entry name greater than the name in words 1-2 of this item.

LAST UST POINTER

H1 of word 5. This points to the External Symbol Table item which precedes this one on the Undefined Symbol chain. This field is zero if the symbol has been defined.

NEXT UST POINTER

H2 of word 5. This points to the External Symbol Table item which follows this one on the Undefined Symbol chain. If the symbol has been defined, this field is broken into two subfields. Q3 is a relocation flag: 0 if relocation of the entry value is incomplete, 1 if the entry value is not relocatable and 2 if relocation of the entry value is complete. Q4 contains the location counter number used, or to be used, in relocating the entry value.

ELT ITEM POINTER

H2 of word 6. This points to the Element List Item for the element which defined this symbol.

Waiting Reference Item

T	LBIT	LEN	NEXT W. R. ITEM	1
UST POINTER			TEXT WORD ADDRESS	2

T

S1 of word 1. This field indicates the sign of the relocation. Zero means the entry value is to be added to the field; one means that the entry value is to be subtracted from the field.

LBIT

S2 of word 1. This is the number of the leftmost bit of the field (numbering 0 to 35, right to left, as usual).

LEN

S3 of word 1. This is the number of bits in the field.

NEXT W. R. ITEM

H2 of word 1. This is the address of the next Waiting Reference item, or zero at the end of the chain.

UST POINTER

H1 of word 2. This is the address of the External Symbol Table item for the symbol whose value is to be used in the relocation.

TEXT WORD ADDRESS

H2 of word 2. This is the address of the program word containing the field whose value is to be relocated.

4.2.3.1.4. INITIALIZATION

Initialization consists of the following steps:

a) Set up base registers.

b) Place loader defined tags in the External Symbol Table and MIND$ in the Undefined Symbol Table.

c) Save initial register values.

d) Read and validate option field.

e) If MAP is specified, print the sign-on line.

If the loader does not have a nucleus, execution starts at the tag BEGIN, where steps a and b are performed. If the loader has a nucleus, step b was performed when the nucleus was loaded; execution starts at the tag RESTART, which performs step a. For both cases, steps c-e are performed at the tag CSTART.

The routine at GETOPT, which performs step d, first clears the option flags (except for TYPE, which is carried over from the nucleus build). This routine simply scans the processor call subfields, looking for element name subfields. All subfields found in the scan which are not element names, or are not among the recognized option keywords, are diagnosed as incorrect options. For keywords which use the following version subfield as a parameter, the scan is automatically advanced past the version subfield.

4.2.3.1.5. READ TPF$

This process consists of three routines, ITPFS, FINDRB and PRMBL. First ITPF$ is called to read the TPF$ File Table Index and Table of Contents. Next FINDRB and PRMBL are called alternately to process all the relocatable elements in TPF$. ITPFS sets A15 to the number of TOC entries and X9 to the address of the first entry. FINDRB scans forward through the TOC until an undeleted relocatable element is found or A15 is decremented to zero. On return from FINDRB, X8 contains the address of the relocatable element TOC entry, or zero if the end of the TOC has been reached.

PRMBL read the preamble of a relocatable element, builds an Element List item for it and builds or updates Common Block Table items and External Symbol Table items.

Once the preamble has been read, PRMBL sets up the Element List item, leaving the location counter and undefined symbol slots empty. PRMBL maintains in MAXTXT a record of the longest RB text encountered, for use in buffer allocation during a later step.

The first preamble table scanned is the Control Information Table. Only group numbers 2 and 4 are processed; group 7 (even address) is ignored and group 8 location counters (diagnostic table information) are loaded as if they were data. Blank common is changed to named common with the name BLANKSCOMMON. The common blocks are processed individually. If a Common Block Table (CBT) item does not exist for a C.I.T. entry, it is created and filled in. If one does exist, its length is checked; unequal lengths are diagnosed and the greater length is used. (Note that no diagnostic is issued for unequal lengths for blank common.) In either case, H1 of the location counter slot in the Element List item is set to the address of the CBT item. At the end of this step, H1 of each location counter slot is either zero or the address of a CBT item.

Next the Location Counter Table in the preamble is scanned. The length of each location counter is moved to H2 of the location counter slot in the Element List item.

Next the Entry Point Table is scanned. For each entry, an External Symbol Table item is found or created. If the EST item is on the Undefined Symbol chain, it is removed from the chain, since it is now defined. The defining information is moved into the EST item.

Finally, the Undefined Symbol Table in the preamble is scanned. For each symbol, an EST item is found or created and its address is placed in the corresponding undefined symbol slot of the Element List item. If the EST item is created in this step, it is placed on the Undefined Symbol chain.

4.2.3.1.6. SCAN LIBRARIES

This is step 3 in the logic overview presented earlier. For each of the three libraries in turn, the library is assigned, its entry point table is read and selected elements are processed.

Three routines are used to assing the libraries and read the entry point tables: RUNLIB, TPLIB and RLIB. Each is entered via LMJ and each skips one instruction on return if the entry point table was not successfully read. In each case the LMJ is followed by an LMJ to SCANLB, the routine which scans the entry point table. Each of the three routines assigns or check assignment of a file, then branches to LIB001, where the entry point table is read.

Once an entry point table has been read, SCANLB is called to scan the Undefined Symbol chain and look up each symbol in the entry point table. SCANLB scans the Undefined Symbol chain, looking up each symbol in the entry point table. If the symbol is found, SCANLB checks for and diagnoses multiple definitions and selects and element. That element's TOC entry is read and passed to PRMBL (discussed in the preceding section, "Read TPFS"). One side effect of PRMBL is to remove the current symbol from the Undefined Symbol chain. Any new symbols added to the chain by PRMBL follow the current symbol, so they will be processed later against the same file. This process repeats for each item on the Undefined Symbol chain.

4.2.3.1.7. ASSIGN ADDRESSES

This is step 4 in the logic overview. Address assignment takes place in four parts: odd non-common location counters, common blocks, even non-common location counters and entry points. Once addresses have been completely assigned for an element, the flag ELTAFLAG is set in the Element List item. This flag is checked to avoid reassigning addresses for elements in the nucleus. During address assignment, the length and address parts (H2 and H1, respectively) of a location counter slot serve several functions. These fields are referred to here as LEN and ADDR. Initially LEN is the length from the location counter table in the RB, and ADDR is the address of the corresponding Common Block Table item, or zero. During part 1 the odd numbered location counters are assigned addresses in numerical order, with the first elements added to the Element List being processed first; common block location counters are skipped. When the address is placed in ADDR, LEN is complemented to show that ADDR is a data address.

During part 2 addresses are placed in the Common Block Table items.

During part 3 the Element List is again scanned as it was in part 1, but now all location counters are examined. Three cases are distinguished:

LEN < 0

An address has already been assigned in ADDR; LEN is set to -LEN.

LEN > 0 & ADDR > 0

This is a common block; ADDR is set from the Common Block Table item whose address is in ADDR.

LEN > 0 & ADDR = 0

This is a data location counter; an address is assigned and placed in ADDR.

During part 4 the elements of the binary tree which makes up the External Symbol Table are scanned and addresses are computed for items with relocatable definitions. Only one point presents any real complexity. H2 of word 5 of an EST item contains the UST pointer if a symbol is still undefined; this address is always greater than 01000 so Q3 is non-zero for all undefined symbols. Once an entry point has been defined, Q3 is 0 if the definition is relative to a location counter, 1 if the definition is absolute, and 2 if the definition is relative and the absolute address has been computed and placed in the item. Thus it is possible to determine that an EST item needs to have an address computed if Q3 of word 5 is zero. Note that the test for an undefined symbol could be made on word 6, which contains the address of the Element List item of the defining element.

4.2.3.1.8. ADJUST D-BANK

This step consists of determining the required D-bank size to accommodate the program plus a buffer as large as the largest relocatable element text and, if necessary, enlarging the D-bank to that size.

4.2.3.1.9. DEFINE WAITING REFERENCES

When a nucleus is being built and a relocation requires the value of an undefined external symbol, a Waiting Reference item is created. This step performs the relocations specified by the existing Waiting Reference list, if any. If an item references a symbol which is still undefined, the item is again deferred. The Waiting Reference item is copied into a new chain which follows the loader tables.

4.2.3.1.10. RELOCATE RB TEXT

This step first stores zero into all words assigned to location counters of this load; i.e., starting after the existing nucleus, if any. Then the Element List items are passed one at a time to the routine READRB, which reads and relocates the RB text.

The first step of READRB is to load the text length from the item, then to set the text length in the item to zero. If the loaded text length is zero, READRB returns immediately; this prevents attempts to reload the nucleus or to load the text of an empty element. READRB then reads the text into the buffer which follows the program space.

The text of an element is processed, starting with the first 14 word block and continuing until the end of the text is encountered, a transfer address is found, or the first word of the next block is zero. Within each block, the relocation bit strings are decoded left to right and the corresponding text words are loaded, relocated and stored in the appropriate word of the program. The end of a block is identified when the first word of the next relocation bit string is zero or when there are not at least two words left in the block at the end of a word group. READRB terminates, returning control to the routine which scans the Element List, when the last text block of the element has been loaded.