LINKING LOADER XVM UTILITY MANUAL

DEC-XV-ULLUA-A-D

XVM Systems

digital

LINKING LOADER XVM
UTILITY MANUAL

DEC-XV-ULLUA-A-D

digital equipment corporation · maynard. massachusetts

First Printing, December 1975

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM		TYPESET-11

CONTENTS

FIGURES

LIST OF ALL XVM MANUALS

The following is a list of all XVM manuals and their DEC numbers, including the latest version available. Within this manual, other XVM manuals are referenced by title only. Refer to this list for the DEC numbers of these referenced manuals.

BOSS XVM USER'S MANUAL	DEC-XV-OBUAA-A-D
CHAIN XVM/EXECUTE XVM UTILITY MANUAL	DEC-XV-UCHNA-A-D
DDT XVM UTILITY MANUAL	DEC-XV-UDDTA-A-D
EDIT/EDITVP/EDITVT XVM UTILITY MANUAL	DEC-XV-UETUA-A-D
8TRAN XVM UTILITY MANUAL	DEC-XV-UTRNA-A-D
FOCAL XVM LANGUAGE MANUAL	DEC-XV-LFLGA-A-D
FORTRAN IV XVM LANGUAGE MANUAL	DEC-XV-LF4MA-A-D
FORTRAN IV XVM OPERATING ENVIRONMENT MANUAL	DEC-XV-LF4EA-A-D
LINKING LOADER XVM UTILITY MANUAL	DEC-XV-ULLUA-A-D
MAC11 XVM ASSEMBLER LANGUAGE MANUAL	DEC-XV-LMLAA-A-D
MACRO XVM ASSEMBLER LANGUAGE MANUAL	DEC-XV-LMALA-A-D
MTDUMP XVM UTILITY MANUAL	DEC-XV-UMTUA-A-D
PATCH XVM UTILITY MANUAL	DEC-XV-UPUMA-A-D
PIP XVM UTILITY MANUAL	DEC-XV-UPPUA-A-D
SGEN XVM UTILITY MANUAL	DEC-XV-USUTA-A-D
SRCCOM XVM UTILITY MANUAL	DEC-XV-USRCA-A-D
UPDATE XVM UTILITY MANUAL	DEC-XV-UUPDA-A-D
VP15A XVM GRAPHICS SOFTWARE MANUAL	DEC-XV-GVPAA-A-D
VT15 XVM GRAPHICS SOFTWARE MANUAL	DEC-XV-GVTAA-A-D
XVM/DOS KEYBOARD COMMAND GUIDE	DEC-XV-ODKBA-A-D
XVM/DOS READER'S GUIDE AND MASTER INDEX	DEC-XV-ODGIA-A-D
XVM/DOS SYSTEM MANUAL	DEC-XV-ODSAA-A-D
XVM/DOS USERS MANUAL	DEC-XV-ODMAA-A-D
XVM/DOS V1A SYSTEM INSTALLATION GUIDE	DEC-XV-ODSIA-A-D
XVM/RSX SYSTEM MANUAL	DEC-XV-IRSMA-A-D
XVM UNICHANNEL SOFTWARE MANUAL	DEC-XV-XUSMA-A-D

PREFACE

This manual describes the operation of the Linking Loader Utility Program as it is used in the XVM/DOS environment.

It was assumed in the preparation of this manual that the reader is familiar with the operation of the XVM or PDP-15 equipment and the contents of the XVM/DOS Users Manual describing the features of the operating system.

Chapter 2, which describes the organization of relocatable binary program units as produced by FORTRAN IV XVM (FORTRAN AND MACRO XVM MACRO), is relevant not only to the Linking Loader but also to CHAIN, .SYSLD, UPDATE, and (in RSX systems) to Task Builder.

Manuals which relate to this manual are the following:

FORTRAN IV XVM Language Manual
FORTRAN IV XVM Operating Environment Manual
XVM/DOS Users Manual
XVM/DOS System Manual
XVM/DOS Keyboard Command Guide
DDT XVM Utility Manual
UPDATE XVM Utility Manual

CHAPTER 1

INTRODUCTION

1.1 GENERAL DESCRIPTION

The LOADER XVM (LOADER) is a program which operates under XVM/DOS. The Linking Loader loads and links both relocatable and absolute[1] binary program units as output from either the FORTRAN IV Compiler or MACRO Assembler. These program units consist of machine language instruction codes and special "loader codes" which tell the loader how to load the program. These program units can reside on the input device not only as separate files, but also as library files. The structure of the input data and description of the loader codes are provided in Chapter 2. Figure 1-1 illustrates the I/O functions of the Loader.

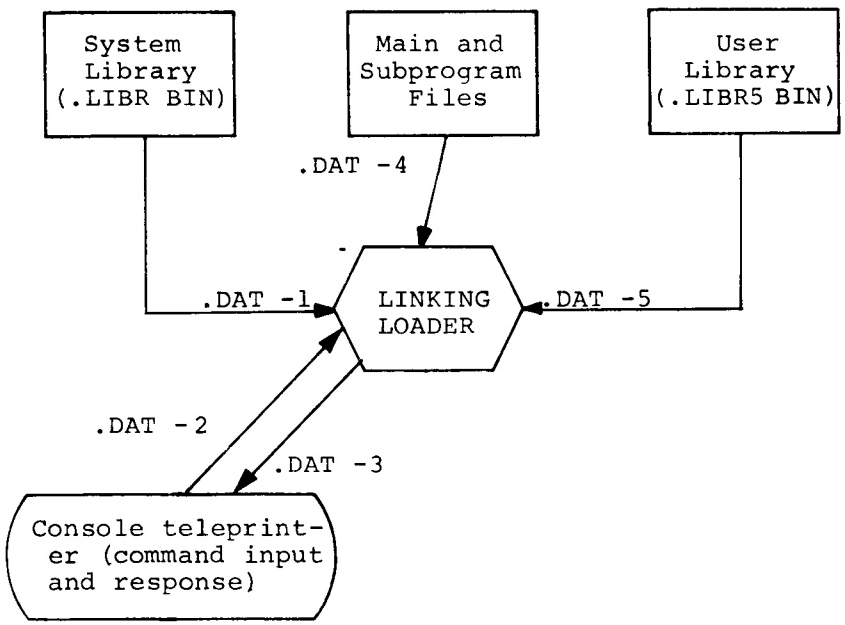


Figure 1-1
Linking Loader I/O Functions

[1] A MACRO assembled program headed by a .LOC statement, e.g., .LOC 100, is an absolute binary program and the binary is output in link loadable format. A program headed by an .ABS, .ABSP, .FULL, or .FULIP statement is output as absolute block binary and cannot be loaded by the Linking Loader.

Initially, the Loader loads all the program units named in the command string (see Operating Procedures, Chapter 3). The Loader then automatically loads and links all requested I/O handlers and library subprograms which have been implicitly requested.[1] The requested library subprograms are loaded from the external (user) library (if one exists) and then the system library (in that order). After both libraries have been examined for requested subprograms, the Loader prints the names of all subprograms which have not been found. I/O handlers that are already in core for the Loader's use will be retained if required for the user's program. The Loader also assigns COMMON data storage areas.

There are two modes of operation which affect how individual program units are loaded into core memory. When the system is running in PAGE mode, program units cannot exceed 4080 words in size and the loader relocates such programs so that they do not straddle a 4K memory page boundary. When the system is in Bank mode, program units cannot exceed 8176 words in size and the loader relocates them to avoid 8K memory bank boundaries. In XVM/DOS there are on-line monitor commands to select the desired mode of operation. If a program is written in absolute rather than relocatable binary, the loader does not check to see if a page or bank boundary is crossed.

Optionally, symbols and their absolute definitions are loaded into a program dictionary (symbol table) for use by the DDT (Dynamic Debugging Technique) Utility Program. The Loader also sets up, for use by DDT, the start execution address of the main program (in the System Communication Table) and the initial relocation value of all the program units (in the symbol table).

1.2 COMMON BLOCKS

The Linking Loader permits COMMON blocks and BLOCKDATA subprograms to overlap memory pages and banks. When operating in either Bank or Page Mode, the Loader allows COMMON block sizes up to 32,767 words (32K-1). Only non-COMMON arrays and variables are initialized to zero by the Loader.

---

[1] Implicit subroutine requests are made by global or symbolic linkages and are established through the use of .GLOBL pseudo-op in MACRO and the external function references and CALL statements in FORTRAN. Implicit I/O handler requests are made by use of the .IODEV pseudo-op in MACRO and the use of READ AND WRITE statements in FORTRAN.

If the system is running with XVM mode ON, the Loader allocates the uninitialized COMMON blocks in core above the bootstrap. The top of usable core above the bootstrap is determined by the system memory size. If there isn't enough space above the bootstrap, the Loader will use space below it. If XVM mode is OFF (which is necessarily the case in PDP-15 systems without XM15 hardware), the Loader loads everything below the bootstrap.

MACRO programs can be linked to COMMON areas defined by FORTRAN IV in two ways. The recommended method is to use the .CBS and .CBD pseudo-op in MACRO. However, for compatibility with earlier systems, the following is a second way. If any unresolved globals remain after the Loader has searched the user and system libraries and has defined COMMON blocks, the Loader tries to match those global names to COMMON block names. Wherever a match is made, the global becomes defined as the COMMON block. For example:

```
FORTRAN IV PROGRAM
   INTEGER A, B, C
   COMMON/NAME/C
   COMMON A, B
      .
      .
      .
MACRO PROGRAM
   .GLOBL NAME, .XX          /.XX IS NAME GIVEN TO BLANK COMMON
                             /BY THE FORTRAN COMPILER
   DZM* .XX                  /CLEAR A - NOTE INDIRECT REFERENCE
   ISZ .XX                   /BUMP COUNTER
   DZM* .XX                  /CLEAR B
   DZM* NAME                 /CLEAR C
```

Note that if the values are REAL (2 words) or DOUBLE PRECISION (3 words) the MACRO program must account for the number of words when accessing specific variables.

1.3 SPECIAL SYMBOLS

The following symbols, when used in this manual, are defined as follows:

Symbol	Meaning
↵	Carriage RETURN
→|	Horizontal Tab
␣	Space
[ ]	Optional Command Element
{ }	One of the enclosed command elements must be chosen.

CHAPTER 2
LOADER CODE DESCRIPTIONS

As mentioned in Chapter 1, the relocatable and absolute binary program units output by the FORTRAN IV Compiler and the MACRO Assembler contain both machine language instructions and loader codes. These codes are assigned by FORTRAN and MACRO to identify the various elements of the binary program. The Loader, in turn, interprets these codes to properly relocate, link, assign COMMON areas, preserve constants, etc. The paragraphs which follow provide descriptions of the physical organization of relocatable program units and library files and definitions of the loader codes.

2.1 INFORMATION UNITS

The binary output from the FORTRAN compiler and the MACRO Assembler consists of named files containing blocks of information units. Each information unit consists of a loader code (6 bits) and a data word (18 bits). The form of the object program at run time is determined by the content and the ordering of the information units. Several information units may be grouped to convey a single run-time instruction to the Loader.

Information units are grouped in blocks of four 18-bit machine words as shown in Figure 2-1.

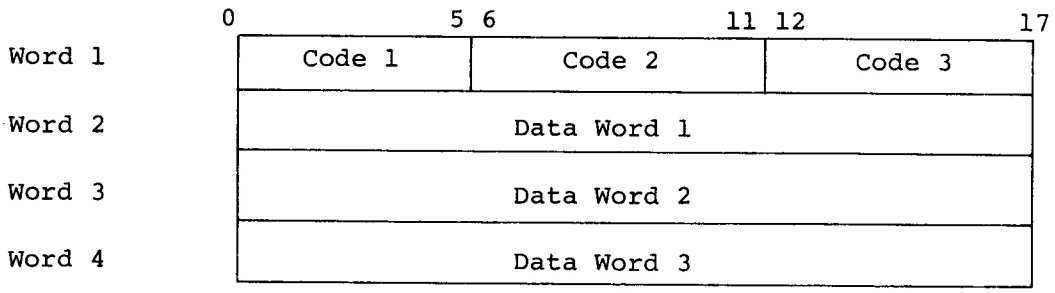


Figure 2-1
Information Unit Block Structure

IOPS binary records of 48 information words and a 2-word header are accepted by the Loader.

2.2 PROGRAM UNIT ORGANIZATION

A program unit consists of as many information units as are required to contain the binary program. The two basic types of program units are diagrammed in Figures 2-2 and 2-3.

PROGRAM SIZE (code 01) does not include COMMON blocks	
INTERNAL GLOBAL DEFINITIONS (code 12)	
PROGRAM NAME (codes 07, 10, 23, 33)	
Codes	
02	Program Load Address
03	Instructions
04	Constants
05	Transfer Vectors
06	Non-COMMON variables and arrays
07, 10	Symbols
11	External global symbol references
14, 15, 16	COMMON block declarations and references
24, 25	Strings
26	I/O Device Routine
31, 32	Relocation Mode
34,01	COMMON block initialization
END (code 27)	

Figure 2-2
Program Unit Organization

Loader Code Description

BLOCK DATA INDICATOR (code 13)
PROGRAM NAME (code 23)
COMMON STORAGE (codes 14, 15, and 16)
DATA INITIALIZATION CONSTANTS (codes 17, 20, 21, and 22)
END (code 27)

Figure 2-3
Block Data Subprogram Organization

2.3 LIBRARY FILE ORGANIZATION

Both system and user library files are structurally identical and are created and maintained by the UPDATE Utility Program. A library file consists of a number of program units (rather than just one) in which all end-of-file codes, except the last, have been removed. Figure 2-4 shows the complete structure of a library file. A library file must not contain BLOCKDATA subprograms.

2.4 IDENTIFICATION CODES

The identification code contained in each information unit tells the Loader how to interpret the associated data word. As mentioned earlier there is an implied order in which codes appear within a binary file.

Code	Loader Action
01	Program Unit or COMMON Block Size The data word normally specifies the number of machine words required by this program unit. This number does not include the required number of machine words for COMMON storage. The program size is used by the Loader to determine whether or not the program will fit within the unused locations of any available page or bank. This is the first information unit of the binary output. In absolute loads, no checking is made to prevent overlay of other program units; this is left to the user. The program size is also used to determine where to begin loading the executable code as loading proceeds from the bottom of the bootstrap downward in core. If this code immediately proceeds code 34, the data word declares the minimum size of the most recently named COMMON block.

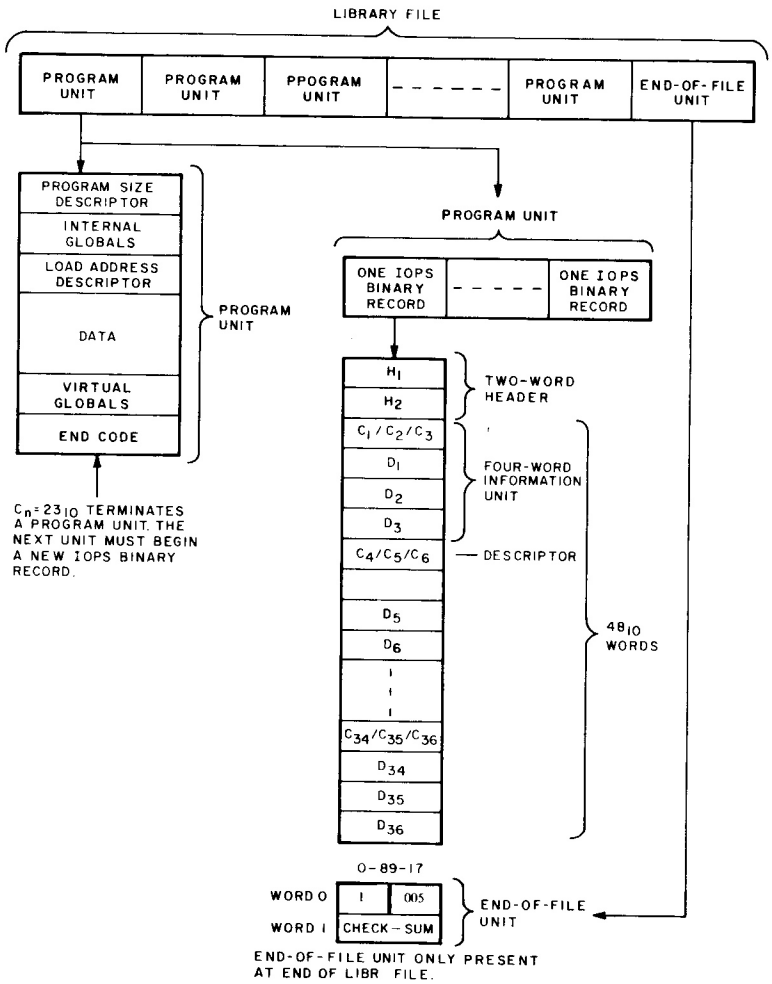
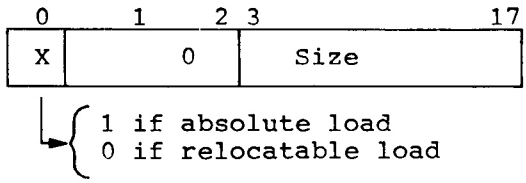


Figure 2-4

Library File Organization

Code	Loader Action

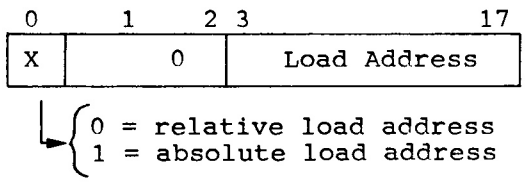
Data Word



02 Program or COMMON Block Load Address

The data word is an unrelocated memory address. This address normally specifies either an absolute or a relative storage address for program data words and is incremented by one for each data word stored (codes 03, 04, and 05). If the address is relative, it is initially incremented by the current relocation factor. If this code directly precedes a code 34 which causes an entrance into data initialization mode, the data word contains a COMMON block load address which is relative to the start of the COMMON block. Bit 0 of the data word is used to indicate an absolute address (bit 0 = 1) or a relative address (bit 0 = 0).

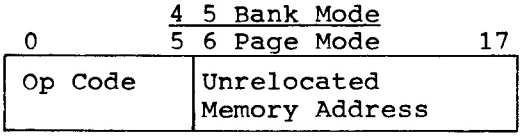
Data Word



03 Relocatable Instruction

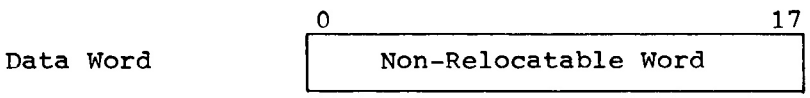
The data word is a memory referencing instruction. The address portion of the instruction is incremented by the current relocation factor (modulo 12 bits for page mode and 13 bits for bank mode). The instruction is stored in the location specified by the load address which is incremented by one after the word is stored.

Data Word

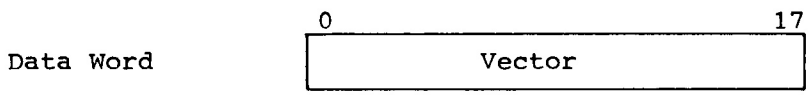


Loader Code Description

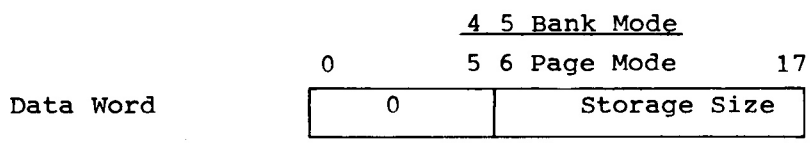
Code	Loader Action
04	Absolute Instruction/Constant/Address The data word is either a non-memory referencing instruction, a non-relocatable memory referencing instruction, an absolute address, or a constant. The word is stored in the location specified by the load address which is incremented by one after the word is stored.



Code	Loader Action
05	Relocatable Vector The data word contains a relocatable program address (vector). The word is incremented by the current relocation factor. The data word is stored in the location specified by the load address which is incremented by one after the word is stored.



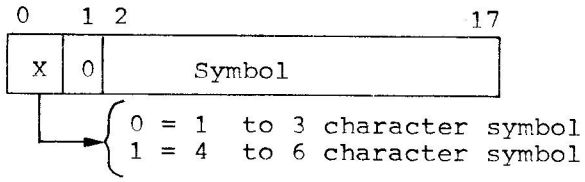
Code	Loader Action
06	Non-COMMON Storage Allocation The data word specifies the number of machine words required for non-COMMON variable and array storage. Storage allocation begins at the address specified by the load address. The load address is incremented by this number. This block of memory is cleared.



Code	Loader Action
07	Symbol - First Three Characters The data word contains the first three characters of a symbol in radix 50_8 format (see Appendix C). The data word is saved by the Loader for future reference.

Code	Loader Action

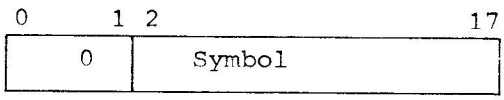
Data Word



10 Symbol - Last Three Characters

The data word contains the last three characters of a symbol in radix 50_8 format. The data word is saved by the Loader for future reference. This word is used only if in the code 07 data word bit 0 = 1.

Data Word

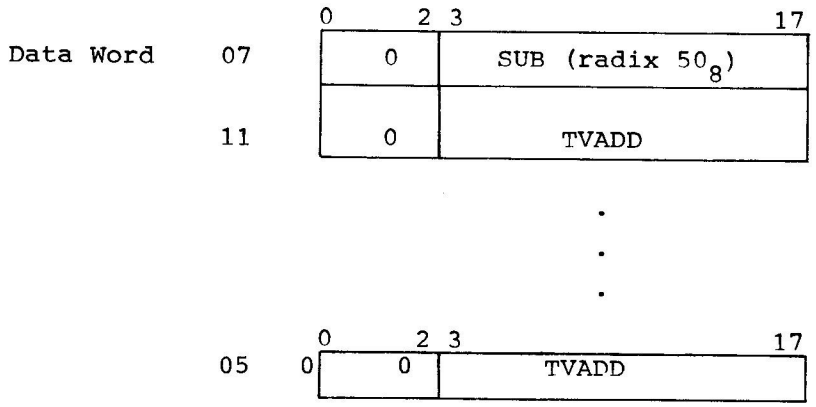


11 External Symbol Definition

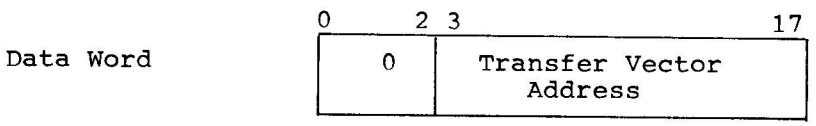
The data word contains the unrelocated address of the transfer vector for the subprogram named by the last symbol loaded (codes 07 and 10). If the external subprogram has already been loaded, the address (definition) of the symbol is stored into the specified relocated vector address. If the subprogram has not been loaded and this is the initial request, the symbol and the relocated transfer vector address are entered into the Loader symbol dictionary as a request for subprogram loading. This action automatically forces the Loader into a Library Search Mode after all programs mentioned explicitly in the command string have been loaded. The Loader remains in Library Search Mode until all unresolved globals have been resolved. If the subprogram has been previously requested (symbol in the dictionary) but not loaded, the Loader chains the reference locations. This chain, generated exclusively by the Loader, is followed when the external definition is encountered. (Unchained transfer vector locations must initially contain a reference address (code 04 or 05) to themselves.) For example, .GLOBL SUB where SUB is virtual causes the output of the following:

Code

Loader Action



SUB is defined internally as TVADD. Subroutine calls are made via JMS* SUB.

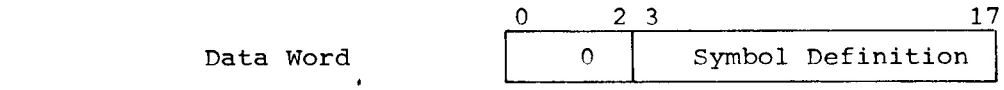


12 Internal Global Symbol Definition

The data word contains the unrelocated or absolute address (definition) of the last symbol loaded (codes 07 and 10). The last symbol loaded is a global symbol internal to the program unit which follows. In the Library Search Mode, if a request for subprogram loading exists (code 11) in the Loader dictionary, the relocatable or absolute definition is stored in the specified transfer vectors and the program unit is loaded. The definition also replaces the transfer vector address in the Loader dictionary. If no request for loading exists, the program unit is not loaded and the Loader continues to examine information units until the next internal global symbol definition is found (Library Search Mode). If the program unit is to be loaded, all internal global symbols following the one causing loading are automatically entered into the Loader dictionary as defined global symbols. If the symbol already exists in the dictionary and is defined (indicating that a program unit

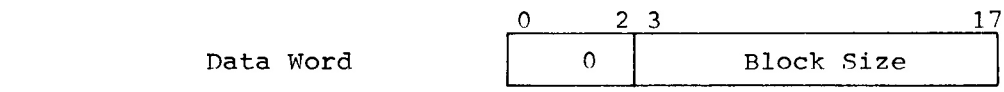
Code	Loader Action

with the same name is already loaded) the Loader does not try to load the program unit again.



13 Block Data Declaration

This information unit instructs the Loader that the COMMON blocks and data constants following are part of a block data subprogram.



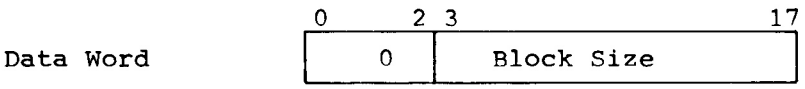
14 COMMON Block Definition

The data word specifies the number of storage words required for the COMMON block named by the last symbol loaded (codes 07 and 10). In general, the assignment of memory space for the COMMON block is deferred until all requested library subprograms have been loaded. The exception to this rule occurs when the block data declaration (code 13) has been encountered. In this case, the COMMON block name is treated as an internal global symbol, and the block is assigned to memory. After the block is assigned to memory, the starting address is entered into the Loader dictionary, and the starting address is saved by the Loader for future use (code 15). All symbols in the dictionary associated with the block are assigned addresses with respect to this starting address. All symbols which are yet to be loaded (via code 15 and 16) will also be assigned as they are encountered. When the block data flag is not set, the Loader enters the name and the size into the dictionary (if it is not already there) and also enters the word containing the next available dictionary entry address. This entry will contain the first symbol in this COMMON block and will be used as the head of the chain of all

Code	Loader Action

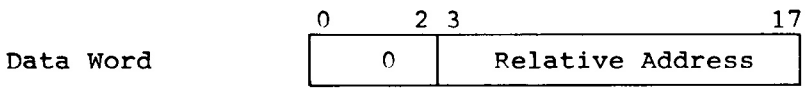
symbols in this COMMON block. The address of the head of chain is saved by the Loader so that the new set of symbols in the COMMON block may be added to the chain. The larger of the two block sizes is retained as the block size.

When the COMMON block has already been assigned memory locations, the respective lengths are compared. Loading terminates, with an appropriate error message, if the assigned block is smaller. When the assigned block is larger or both are equal, loading continues.



15 COMMON Symbol Definition

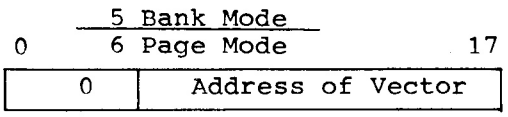
The data word specifies the relative location of the last symbol loaded (codes 07 and 10) in the last COMMON block (code 14). If the associated COMMON block has been defined (block data), the absolute address of the symbol is calculated (block address plus relative position) and placed in TV location (code 16). When the COMMON block has not been assigned, the relative address is entered into the Loader dictionary and chained to the symbols associated with the COMMON block.



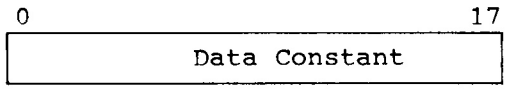
16 COMMON Symbol Reference Definition

The data word contains the unrelocated address of the transfer vector for references to the COMMON symbol named by the last symbol loaded (codes 07 and 10). The symbol definition (code 15) is stored in the relocated address specified when the associated COMMON block has been assigned (code 14). When the block has not been assigned, the relocated address is entered into the Loader dictionary along with the relative address (code 15) of the symbol.

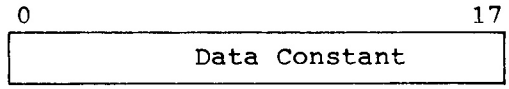
Code	Loader Action



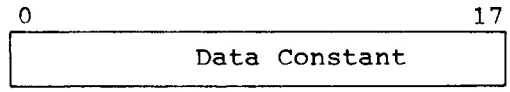
17 Data Initialization Constant - First Word
The data word contains the first machine word of a data initialization constant. It is saved by the Loader for future use (code 22).



20 Data Initialization Constant - Second Word
The data word contains the second machine word of a data initialization constant. It is saved by the Loader for future use (code 22).



21 Data Initialization Constant - Third Word
The data word contains the third machine word of a data initialization constant. It is saved by the Loader for future use (code 22).

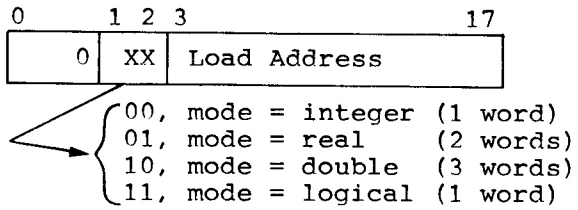


22 Data Initialization Constant Definition
The data word contains the relative load address of the last data initialization constant loaded (codes 17, 20, and 21) and a mode code identifying the constant (real, integer, double, logical). The load address is incremented by the current relocation factor if the constant initializes a non-COMMON storage element. When the constant initializes a COMMON storage element (indicated by the presence of the

Code | Loader Action

block data flag (code 13), the load address is incremented by the address of the last COMMON block loaded (code 14). The constant is stored according to mode and the relocated load address.

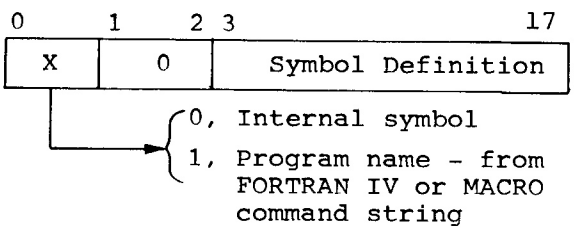
Data Word



23 Program Name or Internal Symbol Definition

The data word contains the unrelocated or absolute address (definition) of the last symbol loaded (codes 07 and 10). The symbol is strictly internal to the program being loaded and is entered conditionally (if DDT is also loaded) along with its relocated or absolute address into the DDT symbol dictionary. The program unit name is indicated by bit 0=1 of the data word.

Data Word



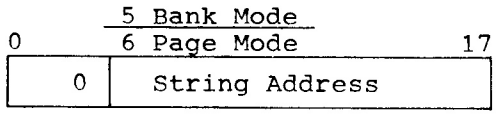
All symbols fall into this category.

24 String Code - First Half

The data word contains the unrelocated address of a data word whose address portion is to be replaced by another value. The relocated address is saved by the Loader for future use (code 25).

Code	Loader Action

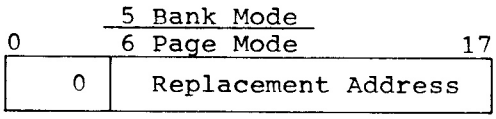
Data Word



25 String Code - Second Half

The data word contains an unrelocated address. The address portion of the data word specified by the first half-string code (code 24) is replaced with this address (relocated modulo 12 bits (page) or 13 bits (bank)).

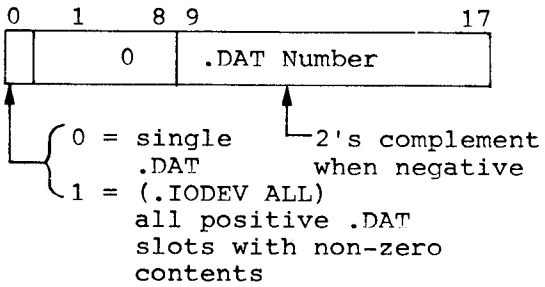
Data Word



26 Input/Output Device Routine Request

The data word specifies the .DAT slot number associated with a device level I/O routine. The Loader defers loading of any I/O handlers until all programs specified in the command string have been loaded. I/O handlers not already residing in memory (see Operating Procedures) are loaded from the disk <IOS> area.

Data Word



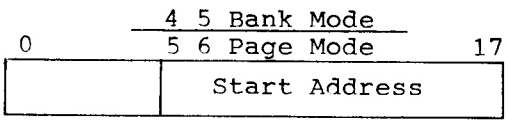
27 End of Program Unit

This is the last information unit of a program module. The data word contains the unrelocated or absolute start execution address of the program. The relocated or absolute start address is entered into the system communication tables to be used when control is given to the user. Only the first start address encountered is entered into the communication tables. (It is

Code | Loader Action

assumed that the first program unit specified in the command string is the main program.) The first address of the main program is used if the .END pseudo-op did not have a start address. Several program units, typically a main program and several subprograms, can be combined into a single named file in the same manner as a library file. The Linking Loader will load each program unit until an end-of-file is encountered. When loading from either the system or external libraries, the end unit causes the Loader to examine the next line buffer for the end-of-file (EOF) condition. When the EOF for the external library is obtained, the Loader automatically begins searching the system library to resolve any remaining globals.

Data Word



30 (Reserved)

31 Enable Bank Relocation
This code is output by the MACRO Assembler in response to the .EBREL pseudo-op. The Loader will relocate all 03 coded data words using 13 bit addressing. The associated data word is unused.

32 Disable Bank Relocation
This code is output by the MACRO Assembler in response to the .DBREL pseudo-op. The Loader will relocate all 03 coded data words using 12 bit addressing. The associated data word is unused.

NOTE

Loader codes 31 and 32 do not affect the execution of the relocated code but merely the size of the address field. Also, these codes are recognized only when the Loader is operating in Page Mode. Program units which use these codes are not allowed to overlap memory page bounds or be larger than 4K. These codes are intended for use with VT15 Display programs and should be used with caution. The instructions EBA

Code | Loader Action

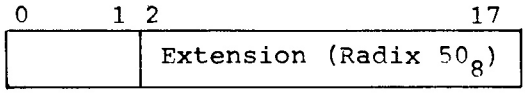
NOTE (Cont)

(enter bank addressing) and DBA (disable bank addressing) should be with code relocated via the .EBREL and .DBREL pseudo-ops.

33 Source File Extension

This code is output by the MACRO XVM Assembler. The data word contains the extension (in Radix 50 format) of the source file which produced this binary. The extension is typically a number indicating the revision level of the program module. It can be examined with the Linking Loader by selection of the P option in the loader's command string (see paragraph 3.3.1).

Data Word



34 COMMON Block Initialization

This code is output by the MACRO XVM Assembler in response to the .CBS and .CBE pseudo-ops. It directs the loader to enter or leave COMMON block initialization mode, depending upon the preceding symbol (codes 07 and 10). If the symbol is not blank, the loader will initialize the COMMON block identified by that symbol. If the symbol is blank, the loader will leave COMMON block initialization mode.

This code may be immediately preceded by code 01, program size, when entering COMMON block initialization mode. This is used to declare a COMMON block size.

Code 34 must always be followed by code 02, which denotes the COMMON block load address when entering COMMON block initialization mode, and the program load address when leaving COMMON block initialization mode. If entering COMMON block initialization mode, the COMMON block is then initialized via codes 03, 04, and 05. It should be noted that when in COMMON

Code	Loader Action
	block initialization mode, all relocation will be done with respect to the base of the COMMON block. In all cases, the data word associated with this code (code 34) is reserved for future use.

CHAPTER 3
OPERATING PROCEDURES

3.1 .DAT SLOT ASSIGNMENTS

Prior to calling the Loader, the user should perform all required device and UIC assignments for both the Loader and the program to be loaded. Space can be saved during loading if the same version of a device handler is used both for the Loader and for the program to be loaded.

All programs named in the command string must reside on the device and UIC associated with .DAT slot -4. At least one program must be loaded from this device. The system library (.LIBR BIN) must be accessible via .DAT slot -1. If a user-created library is to be used, it must be accessible via .DAT slot -5, and be named .LIBR5 BIN. If no user library is required, .DAT slot -5 must be assigned to NON; otherwise IOPS 13 (file not found) errors will occur.

3.2 CALLING THE LOADER

The Loader may be called using any of four commands, depending on the user's requirements, as shown:

Command	Meaning
LOAD↵	Load and Halt. Program execution is initiated by typing CTRL S.
GLOAD↵	Load and Go. Program execution begins automaticlly.
DDT[1]↵	Load user programs along with DDT. When loading is complete, control is given to DDT.
DDTNS[1]↵	Load user programs with DDT but do not build DDT symbol table. This provides more free core but limits debugging to octal numbers.

[1]Refer to the DDT XVM Utility Program Manual for operating instructions on DDT.

Type the desired command immediately to the right of the Monitor's $ as follows:

or $LOAD ↵

$GLOAD ↵

or

$DDT ↵

or

$DDTNS ↵

When running, the Loader identifies itself with one of the following messages, depending upon the addressing mode (Bank or Page).

Page Mode	Bank Mode
LOADER XVM Vnxnnn >	BLOADER XVM Vnxnnn >

3.3 COMMAND STRING

The Command String must be typed immediately to the right of the Loader's prompting symbol > in the format shown below:

>[options]←name1[,name2]...[,namen](ALT)

The option switches may be omitted as explained in Section 3.3.1. The left arrow (or underscore) must be present and so must at least one program name. The command must terminate with an (ALT) character. Below are a few examples of legitimate single line command strings (the > is printed by the Loader):

>P ← RUN1(ALT)

>CP←MAIN,SUBR1,SUBR2(ALT)

>←PROD,TABLE,CONVRT(ALT)

>UL:USERA,G,SL:SYSTEM←MAJOR,MINOR(ALT)

The Loader also accepts Carriage RETURN as a substitute for comma to separate program names. It would be used whenever a continuation line were needed. For example,

>P←PROG1,PROG2,PROG3,PROG4,PROG5(ALT)

is equivalent to

>P←PROG1,PROG2,PROG3,PROG4,PROG5(ALT)

3.3.1 Option Switches

Five option switches may be selected to obtain loader map output on the teleprinter and to specify alternate User or System Library names. If no options are selected, the default Library names apply, no map is output, and loading time is decreased. The switches are as follows:

P - Type program names and addresses

G - Type GLOBL symbols and addresses

C - Type BLOCK DATA names, COMMON block names and first addresses. (.XX is the name for BLANK COMMON) and indicate programs which initialize common blocks.

UL:ULNAME - The User Library name is specified by ULNAME instead of the default name .LIBR5.

SL:SLNAME - The System Library name is specified by SLNAME instead of the default name .LIBR.

The option switches may be typed in any order. The UL and SL options must be followed by a comma (or back-arrow if last option typed). The P, G, and C options can optionally be followed by commas.

In typing out the memory map, the Loader first types option switch character (P, G, or C) followed by the name or symbol and source file extension, followed by the address. To indicate a COMMON block initialization it types the option switch character ("C"), followed by the COMMON block name, followed by the abbreviation "INIT". To indicate a BLOCK DATA program, it types the option switch character ("P"), followed by the block data name, followed by the abbreviation "B DATA". For example:

P␣NAME1␣␣023␣037602	(The program created from source file NAME1 023 is loaded at 037602)
C␣.XX␣␣␣␣␣INIT	(The COMMON block .XX (blank common) is initialized by the preceding program)
G␣.DA␣␣␣␣␣␣␣␣027632	(The global symbol .DA is defined to be 027632 by the following program)
C␣.XX␣␣␣␣␣␣␣␣026000	(The base of COMMON block .XX (blank common) is 026000)

3.3.2 Program Names

Program names are standard six character (maximum) file names of the programs to be loaded. The Loader assumes that all programs have a BIN extension. The name of the main program (i.e., the program which is to obtain control first after loading) must be typed first. Alternatively, all programs (except BLOCKDATA subprograms) to be named in a command string could be combined into a library file under the name of the main program.

The name of the main program is followed by the names of all required subprograms which are not to be loaded from the system or user library. Subprogram names should be typed in order of program size, largest first and smallest last, to obtain optimum core utilization. The program names must be separated either by commas or by Carriage RETURNs. If program input is from a non-directoried device, program names are ignored and need not be typed. Simply type n-1 commas or Carriage RETURNs to load n programs. If program input is from a directoried device, a null file name within the command string is ignored. Thus additional commas and/or Carriage RETURNs can be typed between file names or after the final file name without affecting the semantics of the command.

3.3.3 ALT MODE (Terminates Command String)

An ALT MODE is the only legal command string terminator for the Loader. Once typed, program loading begins.

3.3.4 Command String Errors

Syntactical errors in command strings (e.g., omitting the back arrow (←)) cause the Loader to restart. Typing errors which occur prior to typing a Carriage RETURN or ALT MODE can be deleted through the use of the RUBOUT (delete character) and CTRL U (delete line) teleprinter editing features. The Loader can be restarted at any time prior to typing the ALT MODE terminator by means of the CTRL P command. On a directoried device, after the ALT MODE terminator, a check of the directory is made to verify the existence of each file. If any file named is not present or no names have been specified, the message "NAME ERROR" is output and the loader restarted to allow the correct file names to be input. Any other command string errors observed after an ALT MODE has been typed are unrecoverable and the user must return to the Monitor (via CTRL C) and reload the Loader.

3.4 OPERATION

Upon receipt of the command string, the Loader consecutively loads all explicitly named programs and builds a symbol table consisting of external (global) symbols, internal symbols (if DDT is to be loaded), COMMON block definitions, and COMMON block element definitions. Once all named programs have been loaded, a search is begun to resolve unsatisfied subroutine requests contained in the symbol table. The

Loader searches the IOS directory on the system disk for individual I/O handler files. Then a search through the user library (if present) followed by the system library is made to load subroutines previously referenced. If after a pass through both libraries there remain unresolved global symbol references, another pass is made. This is continued until a complete pass is made without resolving any global symbols. Any remaining unresolved references cause a .LOAD 3 error and loading terminates. Unresolved symbols are typed at the end of a loader map (when a map is requested) and have a load address of 0. The Loader's search normally terminates as soon as all global references are resolved. Executable program code is loaded as high in core memory below the bootstrap as possible. The normal progression of loading one program module after another is downward in core memory. However, because the Loader avoids placing executable code over page and bank boundaries (if PAGE mode is ON) and just bank boundaries (if PAGE mode is OFF), small modules may fit into holes in higher memory which could not hold larger modules loaded first. The first 20_8 locations in each page or in each bank are not loaded as a safeguard to avoid turning indirect memory references through words 10_8-17_8 of a program into autoincrement register references. COMMON blocks initialized with data[1] are allocated core below the bootstrap at the same time as is the executable code, but COMMON blocks are permitted to straddle page and bank boundaries.

Once executable code has been loaded, the Loader assigns core to the uninitialized COMMON blocks. If the system is running with XVM mode ON, the Loader allocates these blocks in core above the bootstrap. The top of useable core above the bootstrap is determined by the system memory size. If there is insufficient space above the bootstrap, the Loader will use space below it. If XVM mode is OFF (which is necessarily the case of PDP-15 systems without XM15 hardware), the Loader loads everything below the bootstrap.

When loading is complete, the Loader resets the lower free core pointers (SC.FRL and SC.FRL+1) to indicate that the area occupied by the Loader and its handlers, if they are not used

[1] COMMON blocks are initialized by BLOCK DATA subprograms in FORTRAN and the .CBS/.CBC/.CBE pseudo-ops in MACRO.

by the loaded program, is now free core. If XVM mode is ON, the Loader also sets SC.FRH and SC.FRH+1 to point to the lowest and highest locations of free core in memory above the bootstrap. If no free memory exists, both locations are set to -1. The Loader then passes control either to the user's main program or to DDT in one of several ways as follows:

a. If the Loader was called using the LOAD command, the Loader types ↑S and waits for the user to type CTRL S to start his program.

b. If GLOAD was used to call the Loader, the user's main program is automatically started.

c. If either DDT or DDTNS was used to call the Loader, control is given to DDT (refer to the DDT XVM Utility Manual for further information).

In all cases, whether transferring control to the user program or to DDT, the Loader enables XVM mode in the hardware if the XVM indicator (see SC.MOD) in the Monitor is ON. With hardware XVM mode enabled, user program indirect memory references can be to 128K of memory (17 bits) instead of the usual 32K (15 bits).

3.5 ERROR CONDITIONS

The following error codes are output by both the Linking Loader and the System Loader. When output by the Linking Loader, the errors are identified as shown below. When output by the System Loader, the errors are identified as ".SYSLD n" instead of ".LOAD n".

Error	Explanation
.LOAD 1	Memory overflow - the Loader's symbol table and the user's program have overlapped. At this point the Loader memory map will show the addresses of all programs loaded successfully before the overflow. Increased use of COMMON storage may allow the program to be loaded since COMMON can overlay the Loader and its symbol table, because it is not initialized until run time.
.LOAD 2	Input data error - parity error, checksum error, illegal data code, or buffer overflow (input line bigger than Loader's buffer).

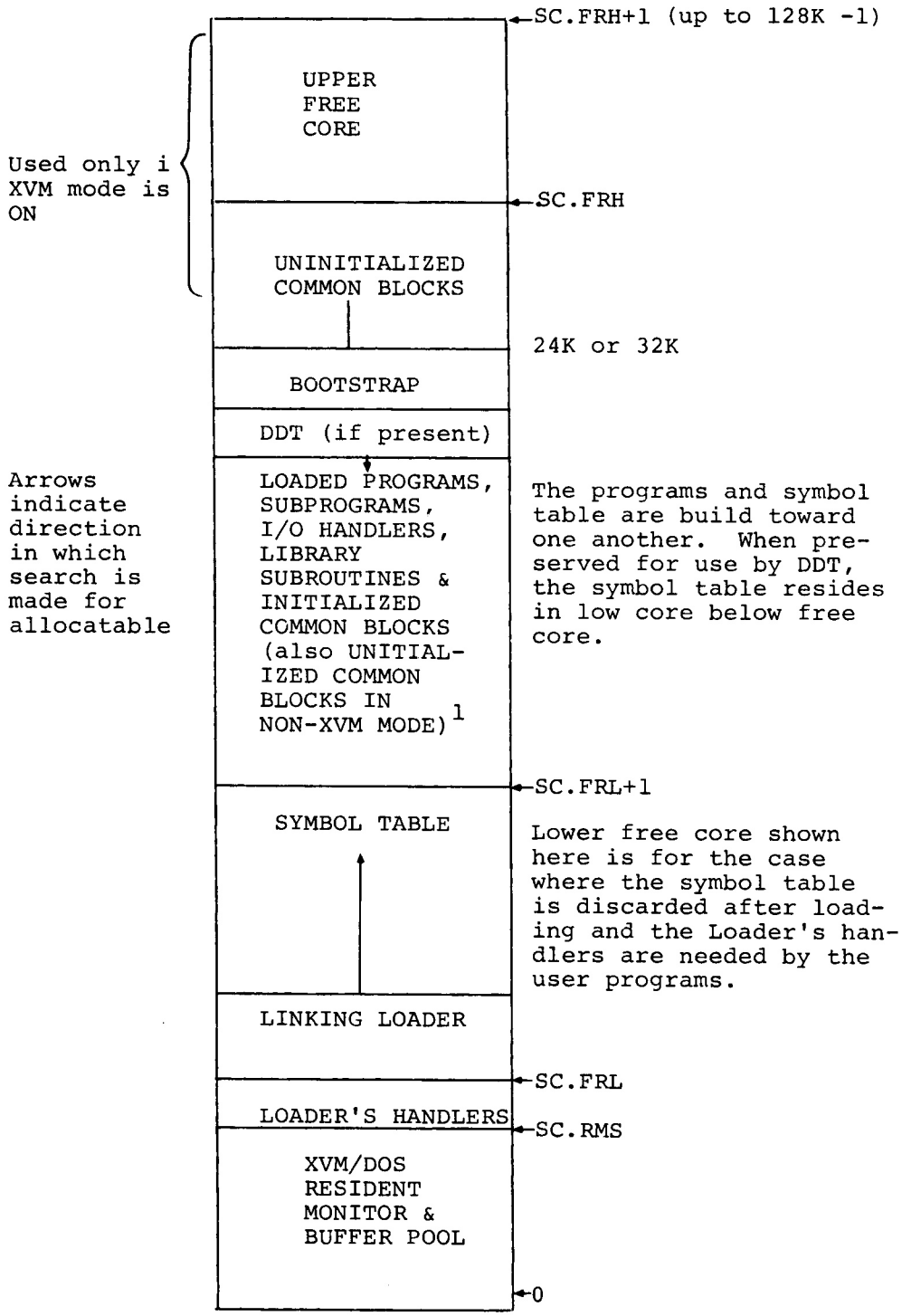


Figure 3-1
Linking Loader Core Map

1 Uninitialized COMMON blocks are loaded below the bootstrap in XVM mode only, if there is no core available above the bootstrap.

(Cont)

Error	Explanation
.LOAD 3	Unresolved Globals - any programs or subroutines required but not found, whether called explicitly or implicitly, are indicated in the memory map with an address of 00000. If any of the entries in the memory map has a 00000 address, loading was not successful; the cause of trouble should be remedied and the procedure repeated.
.LOAD 4	Illegal .DAT slot request - the .DAT slot requested was: a. Out of range of legal .DAT slot numbers; b. Zero; c. Unassigned; that is, was not set up at System Generation Time or was not set up by an ASSIGN command.
.LOAD 5	Program segment greater than 4K - the program segment being loaded in Page Mode cannot be placed in core without crossing a Page Bound.
.LOAD 6	COMMON Block Error - a. COMMON block of zero length or larger than 32K-1 words. A loader map line of the form: C blknam length where: blknam = COMMON block name length = attempted length will be output to the console Teletype[1] immediately prior to the line containing the error code. The program whose name was most recently typed out via the "P" (program loader address) loader map option contains the offending COMMON block reference. b. COMMON block requested length is greater than its length when first loaded. This error may be the result of loading the modules in the wrong order. It is remedied by defining all instances of the same COMMON block to be of identical length.
.LOAD 7	Global symbol with relocated value greater than 32K-1. A loader map line of the form: G Symnam value where: Symnam = global symbol name Value = attempted value

[1]Teletype is a registered trademark of the Teletype Corporation.

(Cont)

Error	Explanation
	will be output to the console terminal immediately prior to the line containing the error code.
.LOAD 8	Data initialization is attempted outside of the range of the program or COMMON block. This can be the result of an input error but is usually caused by an illegal data initialization attempt via the MACRO .CBC pseudo-op.
.LOAD 9 Device where device is a two letter code	Duplicate handler request - More than one handler for the same device has been requested. The first two letters of the duplicate file name are output along with the error message. Loading continues despite this message, but when the loader programs are executed, interrupts initiated by I/O requests may not properly distinguish among the multiple handlers, causing an error.

APPENDIX A
PROGRAMMING NOTES

1. Recommended practice for memory overflow (.LOAD 1) errors -- Apart from the obvious techniques of segmenting large programs and linking them via CALL or .GLOBL's, use of the CHAIN and EXECUTE programs will often allow loading of programs which overflow when using the Linking Loader. Since CHAIN creates XCT files on storage external to memory, space which would have been used by the Linking Loader is made available to the user during the chaining process.

2. Block data subprograms must be explicitly added after the main user program; that is, the name of the block data subprogram must be typed after that of the main program in the Loader command string as a separate file.

3. Care must be taken to assign the same handler to both .DAT slots -1 and -4 (user program input) to avoid possible incorrect linkage to one handler interrupt service when the initial I/O call was made to another handler.

   The assignment of the same handler to both the Linking Loader and user .DAT slots prevents the unnecessary loading of extra handlers which only take up more core. For example, if the program being loaded uses .DAT slots 1 and 2 for its I/O, a core-saving technique is to assign a common disk handler to .DAT slots 1, 2, -4, -1 (and -5 if a user library .LIBR5 BIN exists).

4. .INIT's to negative .DAT slots --
   An .INIT cannot be done to .DAT slots -4, -5, -1 and -7 since the Linking Loader uses these slots and clears them after loading.

APPENDIX B
TERMS AND DEFINITIONS

Term	Definition
Loadable Program Unit	A main program, subprogram, or block data subprogram.
Transfer Vector	A core location containing the address of a subprogram or an entity in COMMON. All references to subprograms and entities in COMMON are indirect.
Internal Global Symbol	A symbol defined in the current program unit and accessible to all programs.
External Global Symbol	A symbol which is referenced in the current program unit and defined in another.
Unresolved Global Symbol	An external global symbol reference which has not yet been resolved by replacement with an internal global symbol definition.
Relocation Factor	The amount added to relative addresses to form absolute addresses; initially, the first loadable core location. The relocation factor for programs following the first program unit is the next available load address.
Radix 50_8 Format	A method of symbol concatenation[1] utilizing 50_8 characters as a "number set", each with a unique value between and including 0 to 47_8. The symbol (number) is converted using standard base conversion methods (see Appendix C).

[1] i.e., linking together.

APPENDIX C
SYMBOL CONCATENATION[1] - RADIX 50_8 FORMAT

Radix 50_8 is a technique used by the MACRO Assembler and the FORTRAN IV Compiler to condense the binary representation of symbolic names in symbol tables. Three characters, plus two symbol classification bits, are contained in each 18-bit word. A symbol is defined as a string of one to six characters; i.e.,

C C C C C C
1 2 3 4 5 6

where any of the possible six characters (C_1 through C_6) can be defined as:

Character	6-bit octal code
Space	00
A	01
↓	↓
Z	32
%	33
.	34
0	35
↓	↓
9	46
#	47

The characters which make up a symbol are linked together in the following manner:

Word 1 $((C_1 * 50_8)+C_2)50_8+C_3$

Word 2 $((C_4 * 50_8)+C_5)50_8+C_6$

For example, the symbol SYMNAM would be entered in the Loader's symbol table as:

Word 1 $((23_8 * 50_8)+31_8)50_8+15_8 = 475265$[2]

Word 2 $((16_8 * 50_8)+1)50_8+15_8 = 053665$

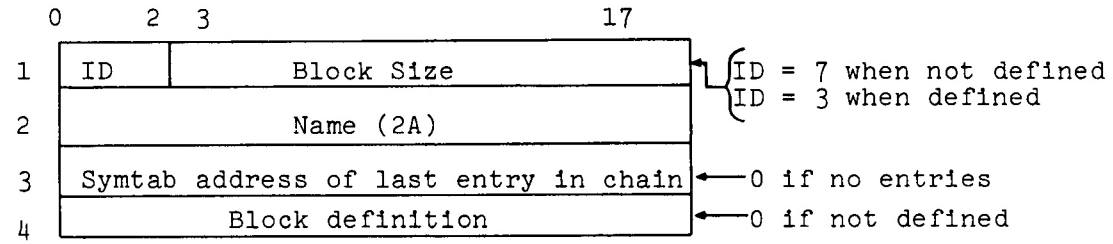
[1] i.e., linking together.

[2] The sign bit of WORD1 is set to 1 to indicate that this symbol consists of more than 3 characters and that the WORD2 is necessary.

APPENDIX D

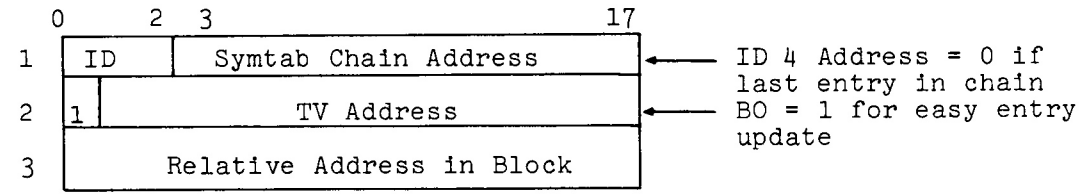
LOADER SYMBOL TABLE

COMMON BLOCK NAME



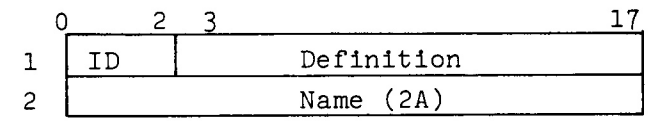
"Name" may require 2 words

COMMON NAME



If associated COMMON block was defined when code 14 is encountered, no entry is needed in the symbol table.

UNRESOLVED OR INTERNAL GLOBAL



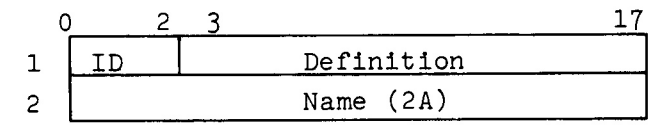
Definition (Unresolved) = Absolute Address of last TV in chain

Definition (Internal) = Absolute address of Symbol

"Name" may require 2 words.

Unresolved ID = 1
Internal ID = 5

INTERNAL NAMES



"Name" may require 2 words

ID=0 (If Program Name ID = 7) — Only entered into the symbol table during DDT loads

INDEX

Please cut along this line.

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you require a written reply, please check here. ☐

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

digital

digital equipment corporation

PRINTED IN U.S.A.