8080/8085

LINKING LOADER MANUAL

Microtec
P.O. Box 60337
Sunnyvale, CA. 94086
408-733-2919

TABLE OF CONTENTS

INTRODUCTION

This manual describes Microtec's 8080/8085 Linking Loader that accompanies the 8080/8085 Relocatable Assembler. The Linking Loader can be used to combine several independently assembled relocatable object modules into a single absolute object module. External references between modules are resolved with the final absolute symbol value being substituted for each reference.

The Loader not only provides for the linking of several modules and adjusting of the relocatable addresses into absolute addresses, but allows the program segment addresses to be specified, PUBLIC symbols to be defined, final load address to be specified and the order of loading of the program segments.

LOADER OPERATION

Many programs are too long to assemble as a single module. These programs can be subdivided into smaller modules and assembled separately to avoid long assembly time or to reduce the required symbol table size. After the separate program modules are linked and loaded by this program, the output module functions as if it had been generated by a single assembly.

The primary functions of the Linking Loader are as follows:

1. Resolve external references between modules and check for undefined references (linking)
2. Adjust all relocatable addresses to the proper absolute addresses (loading)
3. Output final absolute object module

To understand the loading process and to enable the user to use the assembler and Linking Loader (hereafter called Loader) effectively, the user should understand the various program segments and segment load addresses. Although described in the Assembler Manual the various segments are summarized below.

Absolute Segment - this is that part of the assembly program that contains no relocatable information but is to be loaded at fixed locations in the users memory. Absolute code is placed into the output object module exactly as it is read in the input modules.

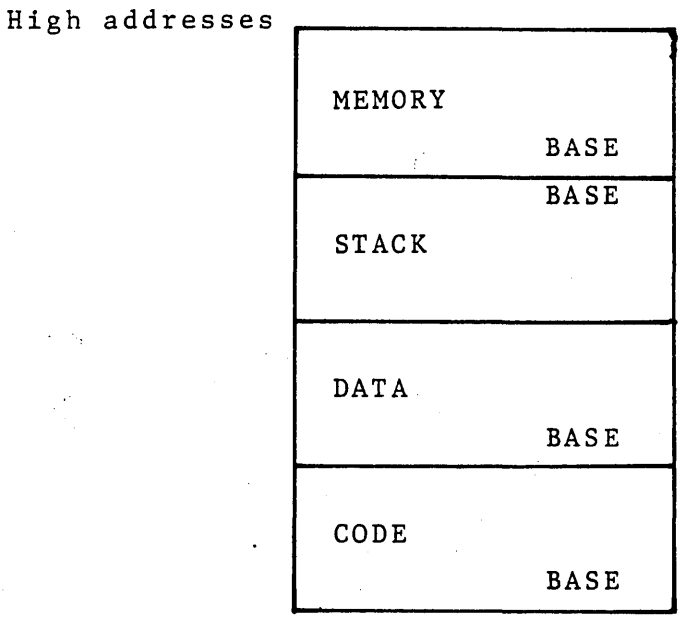
Code Segment - the code segment contains that part of the program which comprises actual machine instructions and which typically can be placed into ROM. Instructions in the code segment can make reference to any other segments.

Data Segment - the data segment contains specifications for that part of a users program that typically contain run time data and which usually resides in RAM. Of course this segment could contain actual machine instructions.

Stack Segment - the stack segment is used as the 8080/8085 run time stack during program execution.

Memory Segment - the memory segment is usually the high address portion of memory which is not allocated to any of the other segments. Data tables may expand into the memory segment but the assembler has no facility to cause instructions to be loaded into the Memory Segment. The start of the Memory segment is determined at Load Time.

The Loader allows the user to load the programs segments into a contiguous program module or to specify the starting address of any or all of the segments. The user may also specify the order in memory in which the segments will be placed. The default memory organization used by the Loader is shown below.



This is the typical memory organization used in most programs. Many users will want to place the STACK segment after the CODE segment so that the DATA segment can expand into the MEMORY segment during program execution.

The BASE addresses for all segments is the low address of the segment. When a user specifies the starting address of a segment via a Loader command, it is the BASE address that is being specified.

Relocation Types

The relocation type of any program segment is determined in the assembler by the CSEG and DSEG commands. The effect of the three relocation types in the Loader are explained below.

Byte Relocation - this implies that no operand was specified on the CSEG or DSEG directives. In this case the segment from the object module will be placed immediately after the same segment from the preceding object module and there will be no wasted memory.

Page Relocation - this relocation type is specified by the PAGE operand on the CSEG or DSEG directive in the Assembler. It implies that the program segment must begin on a page boundary (i.e. 0, 100H, 200H, ...). This code is placed by the Loader at the next available page boundary after the same segment type from the preceding object module.

Inpage Relocation - this is specified by the INPAGE operand on the CSEG or DSEG directive. It implies that the program segment must not cross a page boundary. If the loader determines that a program segment cannot fit within the current page it begins the segment on the next page boundary as though it was PAGE relocatable.

In the typical load sequence the Loader places all CODE segments contiquously in memory followed immediately by all DATA segments with no extra bytes between the segments. However, if any of the DATA segments specify PAGE or INPAGE relocation then the Loader must start the DATA segement at a page boundary so that relocation will be preserved. To avoid any wasted memory the user can always specify starting addresses. In the above case the same problem exists if the DATA segment is followed by the CODE segment and the CODE segment has specified any PAGE or INPAGE relocation.

When initially developing and debugging a program it is helpful to specify each segment in each assembly as PAGE relocatable. This will then force the starting address of each module to end in 00H and will make it easier for the user to follow the flow of the program since the assembler output listing contains the correct memory addresses except for an offset that must be added to the high order address byte.

LOADER COMMANDS

The Loader reads a series of Commands from the Command input device. The Commands may be read in an interactive or batch mode (see Loader Installation Notes). The last command must be an EXIT or an END command.

The object modules are read from the object module input device or files specified on LOAD command. The object modules may be read from the same input device as the Commands.

The output of the Loader consists of an absolute load module suitable for loading into an actual microcomputer. The output module is written to the object module output device and is described in the Loader Installation Notes.

All commands begin in column 1. Command arguments may begin in any column and must be separated from the command by at least one blank. Comments may be placed in the command, and are indicated by an asterisk in column 1.

The following pages describe the Loader commands. In the command descriptions brackets, { }, are used to indicate optional arguments. A summary of the commands is given below.

CODE	Set Code Segment Base Address
DATA	Set Data Segment Base Address
STACK	Set Stack Segment starting Address
MEMORY	Set Memory Segment Base Address
ORDER	Specify Segment Order
START	Specify Starting Output Module Address
STKLN	Specify Stack Length
NAME	Specify Output Module Name

LOAD	Load specified Object Modules
PUBLIC	Specify PUBLIC symbols
LIST	List specified elements
NLIST	Do not list specified elements
EXIT	Exit Loader
END	End command stream and finish final load
*	Comment

Command arguments that are numeric may be either decimal or hexadecimal. Hexadecimal constants are terminated by a H, e.g. 1FH, and need not have a leading zero if it starts with A-F.

Commands may be read in any order and the same command may be used more than once. The last use of a command determines and command parameters. Commands may be placed before or after the LOAD command except for the CODE,DATA,STACK, and MEMORY commands, which if specified must precede the first LOAD command.

CODE — Set Code Segment Base Address

The CODE command is used to specify the starting address of the Code Relocatable Segment. If not specified, the starting address is zero or begins after the preceding segment if this is not the first segment in memory.

Example:

```
CODE   400H
```

```
CODE   value
```

where:

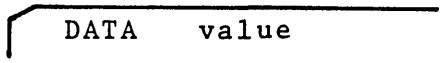
value - specifies the starting address of the CODE segment

DATA — Set Data Segment Base Address

The DATA command is used to specify the starting address of the Data Relocation Segment. If not specified the starting address follows the CODE segment or is zero if the DATA segment is the first segment in memory.

Example:

```
DATA    1000H
```



where:

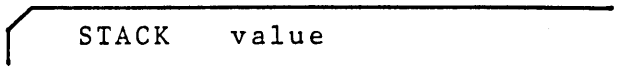
value - specifies the starting address of the DATA segment.

STACK — Set Stack Segment starting Address

This command is used to specify teh starting address of the STACK segment. The length of the STACK segment is specified by the STKLN command or is contained in the Load MOdule. If the Stack address if not specified it will start immediately following the preceding segment in memory or begin at zero if this is the first segment.

Example:

```
STACK    3FFH
```



where:

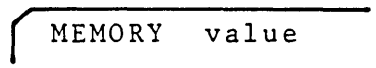
value - specifies the starting address of the STACK segment.

MEMORY — Set Memory Segment Base Address

The MEMORY command is used to specify the starting address of the MEMORY segment. The length of the MEMORY segment will be specified as zero on the load map but it is actually the length of available memory remaining in a user system after the other segments have been loaded. If not specified the starting address will start immediately following the preceding segment in memory or begin at zero if this is the first segment.

Example:

```
MEMORY   8000H
```



where:

value - specifies the starting address of the MEMORY segment.

ORDER — Specify Segment Order

As described under Loader Operation the normal order of the segments in memory is: CODE,DATA,STACK,MEMORY. The ORDER command is provided for users who do not need to specify starting addresses for each segment but would like to segments to be placed in memory in a different order. If the user specifies starting addresses for the segments the order of the segments is of no particular importance.

The user specifies the order of the segments separated by commas. All segments must be specified in the command or an error message is printed.

Example:

ORDER	C,S,D,M	would place segments in the order CODE,STACK, DATA and MEMORY

```
ORDER    seg,seg,seg,seg
```

where:

seg - specifies one of the four segment types as follows:

- C - CODE
- D - DATA
- S - STACK
- M - MEMORY

all four segment types must be included in the command.

START — Specify Starting Output Module Address

This command is used to specify the starting address to be placed in the terminator record of the object module. If not specified the starting address is obtained from the END record of the main program of the input modules. If no main program has been read the starting address will be zero.

Example:

```
START     8
```

```
START  value
```

where:

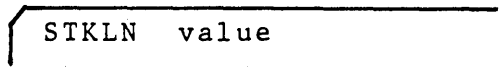
value - specifies the starting address to be used in the object module.

STKLN — Specify Stack Length

The STKLN command is used to specify the length of the STACK segment of the Loader. If not specified the stack length is determined by the sum of the stack segment lengths specified in the load modules.

Example:

```
STKLN   20H
```



where:

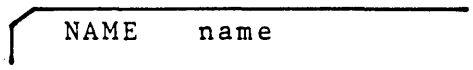
value - specifies the length of the STACK segment

NAME — Specify Output Module Name

The NAME command is used to specify the name of the final output object module. Currently this command performs no function for the output module as the module is in Intel's hexadecimal format and contains no name. It will be used when the output object module is in relocatable format. The user specified name may be any standard symbol and be up to 6 characters. If the user does not specify a name, the name of the output module will be taken from the first input module.

Example:

```
NAME    READER
```



where:

name - is a symbol that specifies the object module name

LOAD — Load specified Object Modules

The LOAD command is used to specify one or more input object modules to be loaded. If the command operand is a number, it is assumed that the input module is to be read from that logical device. If the command operand is not a number, it is assumed the name of a disk file is being specified, and the object module will be read from the file. Object modules may be read from a combination of peripheral devices and disk files. A user may use as many LOAD commands as needed.

The object modules are loaded in the order specified, with each module being loaded into memory immediately behind the preceding module.

Example:

```
LOAD    7,FILE1,FILE2,7
```

Four modules are to to loaded, the first form unit 7, FILE1 and FILE2 from disk and the fourth from unit 7 Unit 7 may be a paper tape reader for exampl

```
LOAD   module1{,module2, ..., modulei}
```

where:

$module_i$ - specifies the number of a logical input device or the name of a disk file on which the object module resides. Operands are separated by commas.

PUBLIC — Specify PUBLIC Symbols

This command is used to define and/or change the value of a PUBLIC symbol. If the symbol specified by this command is already a PUBLIC symbol (from an object module), the value of the symbol is changed to that specified by the user. If the symbol sepcified by this command is not already defined, it will be entered in the Loader Public symbol table along with the specified value and will then be available to satisfy external references from object modules.

This command is useful in that it allows the user to specify the value of some external symbols at Load time and possibly avoid any reassembly. To change the value of a symbol that is Public in a object module this command must be specified after the object module has been loaded via the LOAD command.

Example:

```
PUBLIC      INPUT=2FH,OUTPUT=200H
```

```
PUBLIC    sym1=val1{,sym2=val2, ...,symi=vali}
```

where:

sym_i - is user defined PUBLIC symbol

val_i - is the value of the symbol

LIST — List Specified Elements

The LIST command may be used to generate listings of the elements specified. The defaults are: no symbol tables are listed, an output object module is produced, no symbols are placed in the output object module, and local symbols are not purged from the input modules.

Example:

```
LIST    T          list symbol tables on
                   list device
```

```
LIST    O,P,S,T
```

where:

O - specifies that an object module is to be produced. (default)

P - specifies that any symbols present in the input modules be placed into the Loader symbol table. (default)

S - specifies that the local symbol table be written to the object module and thus may be used for debugging.

T - specifies that both PUBLIC and local symbol tables be listed on the list output device.

NLIST — Suppress Listing of the Elements Specified

The NLIST command is the opposite of the LIST command and is used to suppress the listing of the elements specified. The elements may be turned back on with the LIST command.

Example:

```
NLIST    O          don't produce an object
                    module
```

```
NLIST   O,P,S,T
```

where:

O - specifies that no output module is to be produced. This is useful to check for errors.

P - specifies that any symbol tables present in the input modules not be placed in the Loader symbol table. This is useful if many modules are being loaded and the symbol table may become full. Of course these local symbols may then not be listed in a symbol table.

S - specifies that the local symbol table not be written to the object module and thus may not be used for debugging.

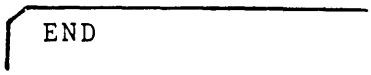
T - specifies that no symbol tables be listed on the output list device.

EXIT — Exit Loader

The EXIT command is used in the interactive mode to exit the Loader. This command is useful when the user finds an error that will require the exiting of the Loader to fix. It acts like an END command except the final load does not take place and an output object module is not produced. This command may also be used in the batch mode by making it the last command in the command stream. In this case the final load will not take place but the object modules and commands will be read and checked for errors.

END — End command stream and finish final load

The END command should be the last command in every Command stream except if the EXIT command is used as described under that command. It initiates the final steps in linking and loading the input modules. An exit is then made from the program.



Comment — Specify Loader Comment

An asterisk may be used to specify a comment in the command input stream. The asterisk should be in column one.

Example:

```
* LOADER EXAMPLE
```

HOW TO USE THE LOADER

The Loader

The Loader program is usually supplied as an unlabeled unblocked magnetic tape with 80 character card image records. Other media may be requested.

The Loader is written entirely in Fortran and is comprised of a main program and several subroutines. The main program appears first on the tape and the last subroutine is followed by a tape mark. The Loader is located after the assembler on the tape.

The Loader Installation Notes describe program installation and any modification that may have to take place for a particular computer. It is helpful to read these notes before installing the program.

Loader Execution

This is a two pass loader in which the commands and object modules are checked for errors during the first pass and a symbol table of PUBLIC symbols is formed. Errors detected during this phase of the program will be displayed on the listing. If the user is in batch mode any errors found during this pass will cause the loader to terminate with the message "LOAD NOT COMPLETED". If the user is in interactive mode, only those errors found in the object modules will cause termination of the loader.

During pass two of the Loader, the final object module is produced and any undefined externals are printed on the list device. A symbol table may also be listed.

When executing the Loader, the user should place the Loader Commands on the command input device expected by the program. Of particular importance is that the user specify the correct number of modules to be loaded and where they are loaded from on the LOAD command.

Loader Listing

The following pages show a sample listing from the Loader which is used to describe both the output listing and the Loading process.

The first page of the output listing lists all commands entered by the user along with any command errors. Following this would be any load module errors that occurred in the modules loaded via the LOAD command. If no fatal errors occur up to this point then a load map is displayed which lists the names of all input modules followed by the starting addresses of the CODE and DATA segments for that module. The ending address+1 for each segment is displayed at the end of all modules and is indicated by the //. Following this, the starting and ending addresses of the STACK and MEMORY segments are displayed. The ending addresses plus one are once again shown by the double slashs, //. When the starting and final addresses are the same, it implies that the length of the segment is zero.

Following the Load Map is a list of all PUBLIC symbols as well as local symbols if the user specified the "LIST T" command. PUBLIC symbols are those declared public in the assembler by the PUBLIC directive. Local symbols are those that were output by the assembler if the user had specified the "LIST B" directive. These may be used for debugging.

As shown on the example listing, the only other information that will be displayed on the listing after this point are any undefined externals found during final load.

The end of the Load program is indicated by the "LOAD COMPLETE" or "LOAD NOT COMPLETE" message.

```
  **LOADER COMMANDS

*
*
LIST   T,S
DATA    407H
CODE  605H
ORDER  C,S,D,M
STACK   A00H
STKLN  12
LOAD  5,5
LOAD  5
END



  **LOAD MAP**

    MODULE     CODE  DATA

    MAIN       0605  0407
    READ       063F  0458
    MODULE     0693  0500
    //         06A4  050F
    STACK      09F4
    //         0A00
    MEMORY     050F
    //         050F


  **PUBLIC SYMBOLS

TIN     061C      CRLF    0634      TOUT    0629      ECHO    0457
INBUF   0407      IBUFEN  0457      READ    063F


  **LOCAL SYMBOLS

BSPA    0008      BLNK    0020      ASCR    000D      TAB     0008
READ    063F      READ10  0644      READ20  0652      READ30  065F
READ40  0669      READ50  0673      READ60  067D      READ70  0680
READ80  0686

**MODULE   MAIN
   UNDEFINED EXTERNALS
   0011


  **LOAD COMPLETED
```

```
BSPA     0008H  BLNK     00020H  ASCR     0000DH  TAB      00008H
READ     0063FH  READ10  00644H  READ20  00652H  READ30  0065FH
READ40   00669H  READ50  00673H  READ60  0067DH  READ70  00680H
READ80   00686H
:1E06050031000ACD3F062107047EFE2023CA0E06CD000023C30506DB00E602CA1C0654
:1E062300DB00E67F47C9DB00E601CA290678D300C9060DCD2906060ACD2906C921078E
:1E06410004 1E00CD1C06FE18C25206CD3406C33F06FE0DC25F067BB7CA4406360DC9C7
:1E065F00FE7FC273067BB7CA44062B1D0608CD2906C38006FE08CA7D06FE20DA800613
:1E067D0077231C7BFE57CA69063A5704B7CA4406CD2906C34406002100003A0B05B715
:09069B00C2A006002F210E057615
:0F050000C3A006010B0580A0060B000506A00096
:00060501F4
```

LOADER EXAMPLE OUTPUT OBJECT MODULE

```
                                ; THE TERMINAL
                                ;
                                ; ENTRY PARAMETERS
                                ;   B          - CHARACTER TO OUTPUT
                                ;
                                ; EXIT PARAMETERS
                                ;   NONE
                                ;
                                ; REGISTERS USED
                                ;   A,B
                                ;
                                ;
0024  DB 00                     OUT8:   IN      USTAT           ;READ STATUS
0026  E6 01                             ANI     TRDY            ;CHECK IF READY
0028  CA 24 00        C                JZ      OUT8            ;NOT READY
002B  78                                MOV     A,B
002C  D3 00                             OUT     UDATOUT         ;OUTPUT DATA
002E  C9                                RET
                                ;
                                ; NAME - CRLF
                                ;
                                ; THIS ROUTINE OUTPUTS A CARRIAGE RETURN
                                ; AND LINE FEED
                                ;
002F  06 0D                     CRLF:   MVI     B,ASCR
0031  CD 24 00         C                CALL    OUT8
0034  06 0A                             MVI     B,ASLF
0036  CD 24 00         C                CALL    OUT8
0039  C9                                RET
                                ;
                                        DSEG                    ;SET DATA SEGMENT
0000                            INBUF:  DS      80              ;INPUT BUFFER
                                IBUFEND:                        ;END OF BUFFER
0050                            ECHO:   DS      1               ;ECHO FLAG
0000                            USTAT   EQU     0               ;USART STATUS
0000                            UDATOUT EQU     0               ;USART OUTPUT
0000                            UDATIN  EQU     0               ;USART INPUT
0001                            TRDY    EQU     1               ;TRANSMIT READY
0002                            RRDY    EQU     2               ;READER READY
000D                            ASCR    EQU     13
000A                            ASLF    EQU     10
0020                            BLNK    EQU     20H
0017                    C       TIN     EQU     IN8
0024                    C       TOUT    EQU     OUT8
0051                            END     MAIN
```

ASSEMBLER ERRORS = 0

```
 2                                      NAME    MAIN
 3                                      PUBLIC  INBUF,IBUFEND,TIN,TOUT,CRLF,ECHO
 4                                      EXTRN   READ,SCAN
 5                              ;
 6                              ;
 7                              ; THIS IS A SAMPLE PROGRAM THAT SHOWS MOST OF THE RELOCATABLE
 8                              ; FEATURES OF THE ASSEMBLER.  TWO MODULES ARE LINKED TOGETHER
 9                              ; TO FORM THE FINAL PROGRAM.  PUBLICS AND EXTRNALS ARE USED
10                              ; TO PERFORM THE LINK.
11                              ;
12                              ; BELOW IS THE MAIN PROGRAM AND THE I/O DRIVERS.  THIS IS
13                              ; LINKED TO A ROUTINE WHICH READS A LINE OF CODE AND WHICH
14                              ; ITSELF REQUIRES THE I/O DRIVERS.
15                              ;
16                              ;
17                                      CSEG                            ;SET CODE SEGMENT
18                              ;
19  0000  31 00 00      S      MAIN:   LXI     SP,STACK                ;SET STACK POINTER
20  0003  CD 00 00      E              CALL    READ                    ;READ NEXT LINE
21  0006  21 00 00      D              LXI     H,INBUF                 ;START OF BUFFER
22  0009  7E                   MAIN10: MOV     A,M
23  000A  FE 20                        CPI     BLNK                    ;CHECK FOR NON BLANK
24  000C  23                           INX     H
25  000D  CA 09 00      C              JZ      MAIN10
26  0010  CD 00 00      E              CALL    SCAN                    ;GET VALUE
27  0013  23                           INX     H
28  0014  C3 00 00      C              JMP     MAIN
29                              ;
30                              ; NAME - IN8
31                              ;
32                              ; THIS ROUTINE WILL INPUT A CHARACTER FROM THE TERMINAL
33                              ;
34                              ; ENTRY PARAMETERS
35                              ;   NONE
36                              ;
37                              ; EXIT PARAMETERS
38                              ;   A           - INPUT CHARACTER
39                              ;   B           - SAME AS A
40                              ;
41                              ; REGISTERS USED
42                              ;   A,B
43                              ;
44                              ;
45  0017  DB 00                 IN8:    IN      USTAT                   ;READ UART STATUS
46  0019  E6 02                         ANI     RRDY                    ;CHECK IF READY
47  001B  CA 17 00      C               JZ      IN8                     ;NOT READY YET
48  001E  DB 00                         IN      UDATIN                  ;READ DATA
49  0020  E6 7F                         ANI     127                     ;DELETE PARITY BIT
50  0022  47                            MOV     B,A
51  0023  C9                            RET
52                              ;
53                              ; NAME - OUT8
54                              ;
```

```
162E0002500006ECHO**00000006INBUF*00500005IBUFEN00E6
0614000100003100000C00000E7
240A0003030100C8
200C0003000000400CD
0640000106002100007EFE2023CA0900CD000023C30000080 0E602CA1700D300E67F69
2210000 30E0015001C006C
240A0002030700C6
200C0003010011009F
0638000122004 7C90B00E601CA24007 8D300C9060DCD2400060ACD2400C3FD
2210000329003200370039
040A0001010000F0
0E0200F0
```

MAIN OBJECT MODULE

```
                        CSEG                            ;SET CODE SEGMENT
                        LIST    X
                        LIST    B
                        PUBLIC  READ
                        EXTRN   CRLF,TIN,TCUT,ECHO,INBUF,IBUFEND
                ;
                ; NAME - READ
                ;
                ; THIS ROUTINE READS IN A LINE FROM THE TERMINAL AND
                ; PLACES IT INTO THE INPUT BUFFER.  THE FOLLOWING ARE
                ; SPECIAL CHARACTERS.
                ;   CR              - END OF CURRENT LINE
                ;   CONTROL X       - DELETE CURRENT LINE
                ;   DEL             - DELTE CHARACTER
                ; ALL DISPLAYABLE CHARACTERS BETWEEN BLANK AND Z AND
                ; THE ABOVE SPECIAL CHARACTERS ARE RECOGNIZED BY THER
                ; ROUTINE AS WELL AS THE TAB.  ALL OTHER CHARACTERS ARE
                ; IGNORED.  AN ATTEMPT TO INPUT MORE CHARACTERS THEN IS
                ; ALLOWED IN THE INPUT BUFFER WILL BE INDICATED BY A BACKSPACE.
                ;
                ; ENTRY PARAMETERS
                ;   ECHO        - ECHO FLAG, 0 = NO ECHO
                ;
                ; EXIT PARAMETERS
                ;   INBUF       - CONTAINS INPUT LINE
                ;
                ; REGISTERS USED
                ;   A,B,E,H,L
                ;
                ;
0000  21 00 00      E READ:   LXI     H,INBUF           ;INPUT BUFFER ADDRESS
0003  1E 00               MVI     E,0               ;SET CHARACTER COUNT
0005  CD 00 00      E READ10: CALL    TIN               ;READ NEXT CHARACTER
0008  FE 18               CPI     24                ;CHECK FOR CONTROL X
000A  C2 13 00      C         JNZ     READ20            ;NOT CONTROL X
000D  CD 00 00      E         CALL    CRLF
0010  C3 00 00      C         JMP     READ              ;START AGAIN
0013  FE 0D         READ20:   CPI     ASCR              ;CHECK IF CR
0015  C2 20 00      C         JNZ     READ30            ;NO
0018  7B                      MOV     A,E               ;GET COUNT
0019  B7                      ORA     A                 ;CHECK IF ANY INPUT
001A  CA 05 00      C         JZ      READ10            ;KEEP READING
001D  36 0D                   MVI     M,ASCR            ;PUT CR AT END OF LINE
001F  C9                      RET
0020  FE 7F         READ30:   CPI     127               ;CHECK FOR DELETE
0022  C2 34 00      C         JNZ     READ50
0025  7B                      MOV     A,E               ;GET COUNT
0026  B7                      ORA     A
0027  CA 05 00      C         JZ      READ10            ;NOT ENTRIES YET
002A  2B            READ40:   DCX     H
002B  1D                      DCR     E                 ;DECREMENT COUNT
002C  06 08                   MVI     B,BSPA            ;GET A BACKSPACE
002E  CD 00 00      F         CALL    TOUT              ;OUTPUT BACKSPACE
```

```
   55  0031  C3 41 00     C            JMP     READ70
   56  0034  FE 08                READ50: CPI     TAB             ;CHECK FOR A TAB
   57  0036  CA 3E 00     C            JZ      READ60
   58  0039  FE 20                     CPI     BLNK
   59  003B  DA 41 00     C            JC      READ70
   60  003E  77                   READ60: MOV     M,A             ;PUT CHARACTER INTO BUFFER
   61  003F  23                           INX     H
   62  0040  1C                           INR     E               ;INCREMENT COUNT
   63  0041  7B                   READ70: MOV     A,E             ;GET COUNT
   64  0042  FE 00        E               CPI     .LOW.IBUFEND    ;CHECK FOR END OF BUFFER
   65  0044  CA 2A 00     C            JZ      READ40          ;HAVE END
   66  0047  3A 00 00     E       READ80: LDA     ECHO            ;GET ECHO FLAG
   67  004A  B7                           ORA     A
   68  004B  CA 05 00     C            JZ      READ10          ;DONT ECHO CHARACTER
   69  004E  CD 00 C0     E            CALL    TOUT            ;ECHO CHARACTER
   70  0051  C3 05 00     C            JMP     READ10          ;CONTINUE
   71                             ;
   72  000D                       ASCR    EQU     13
   73  0008                       BSPA    EQU     8
   74  0020                       BLNK    EQU     20H
   75  0008                       TAB     EQU     08H
   76  0054                       END

ASSEMBLER ERRORS =    0
```

```
16120001070006READ**00PD
123C000090006BSPA**00200 0063LNK**C000006ASCR**00.   6TA ***003A
123C0001000C06READ**0005000 6READ10005130006READ20002000 6READ300C45
123C00012A0006READ400034C006READ50003E0006READ6000410006READ70008E
121200014700 06READ80004C
063C0001000C2100001E09CD0000FE18C21300CD0C00C30C00FE0DC220007BB717
22100003080011001600 99
201C000304000100010006 0000C00E00A7
064000011A00CA050036CDC9FE7FC234007BB7CA05002B1D0608CD0000C3410CFE0823
22140003180023002800 32002F
200C000302C02F00A0
062A00013600CA3E00FE20DA41007723 1C7BFE00CA2A0035
2210000337003C00450013
200C000105004300 8B
06220C0147003A000087CA0500CD0000C305003B
220C00034C00520031
20140003030048000 2004F002D
040A0000010000F1
0E0200F0
```

READ OBJECT MODULE

```
 1                                              CSEG
 2                                              LIST      X
 3  0000  00                                   NOP
 4  0001  21 00 00                              LXI       H,0
 5  0004  3A 0B 00          D                   LDA       DATA
 6  0007  B7                                   ORA       A
 7  0008  C2 0D 00          C                   JNZ       LAB1
 8  000B  00                                   NOP
 9  000C  2F                                   CMA
10  000D  21 0E 00          D   LAB1            LXI       H,DATA+3
11  0010  76                                   HLT
12                                         ;
13                                             DSEG      PAGE
14  0000  C3 0D 00          C                   JMP       LAB1
15  0003  01 0B 00          D                   LXI       B,DATA
16  0006  80                                   ADD       B
17  0007  0D 00             C                   DW        LAB1
18  0009  0B 00             D                   DW        .LOW.DATA
19  000B  05                    DATA            DB        5,6,.LOW.LAB1
20  000C  06
21  000D  0D
22  000E  00                                   NOP
23  000F                                       END

ASSEMBLER ERRORS =     0
```

```
220800030900CA
240E0002030500E0096
061A00020000C300000108008000D0075
220800030400CF
240E0001030100070OC2
06140002090CCB0005060D00B8
220800010900CC
240A0001010D00C3
040A0000010000F1
0E0200F0
```

MODULE OBJECT MODULE

```
  **LOADER COMMANDS


*
*
LIST   T,S
DATA    407H
CODE  605H
STRT   1000H
   INVALID COMMAND
ORDER  C,S,D,M
STACK   A00H
STKLN  12
LOAD  5,5


**MODULE   MAIN
   RECORD OUT OF SEQUENCE
   RECORD    5 - 240A0003030100CB


**MODULE
   HEADER RECORD ERROR
   RECORD    1 - 183E0006CRLF**0006TIN***0006TOUT**0006ECHO**0006INBUF*0006IBUFEN0058
LOAD  5
END


  **LOAD NOT COMPLETED
```

Loader Example

The preceding pages show three assembly listing of programs that will be combined by the Loader along with the output of the Loader. The main program contains references to a subroutine READ and SCAN which are not in the program but are declared external and will be found in another object module. The second assembly listing shows the READ routine which is required by the Main program and also shows that the READ routine requires I/O drivers TIN and TOUT which are declared external and will be found in the main program. The third program contains no links to the other programs but will also be loaded into the final module.

The Command stream shows that the user has specified the starting addresses of both the CODE and DATA segments and has changed the order of the segments to CODE,STACK,DATA, and MEMORY. The LIST command is then used to obtain a symbol table of all PUBLIC symbols used in the modules along with their final absolute addresses. Finally the LOAD command is used to read the three modules from the device shown.

The load map shown the starting and ending addresses of the three modules in the order loaded. Note that the third module had specified a "DSEG PAGE" direcitve in the assembly listing and the load map shows that the data segment for this module indeed starts on the next page boundary.

An undefined external is listed for the Main module and its address is specified. From the original listing it can be seen that the SCAN routine is not in any module. The user could have specified the address of the routine with a PUBLIC command.

Finally the symbol table of all PUBLIC symbols used in the program along with their absolute addresses is shown. The user can determine from the addresses as well as the final object module displayed on a subsequent page that the modules have indeed been linked together to form a final absolute module with all addresses adjusted to the correct values and any links between modules resolved.

Following the above example, a Loader run is displayed that contains many errors. Most of the load errors shown will not occur except under unusual conditions and they have been shown for information purposes only.

The final absolute object module from the example is also shown with the local symbols being part of the module.

APPENDIX A

LOADER MESSAGES

Messages from the Loader may be classified into Command Error Messages and Load Messages. Command errors are due to invalid commands or command parameters and always cause termination of the Loading process in the batch mode. Command messages are listed beneath the actual command. Load messages occur during the loading of object modules initiated by the LOAD command. These messages may be fatal or informative. For most load messages, the message is listed followed by the record number in the input module and the actual record in error. The module name is also listed at the start of the messages for a particular module.

Most load errors should not occur and if they do, the user is advised to first reassemble the program and attempt to reload.

Command Messages

Invalid Command -- a command specified by the user is not a legal Loader command.

Invalid Operand - an operand specified for a command contains invalid characters, does not exist, or is too large.

Command Not Allowed - this command is not allowed at this point in the program. Due to specifying a load address after a LOAD command has been specified.

Symbol Table Full - user specified a PUBLIC command and no more room exists in the symbol table.

Module Greater than 64K - At final load time the lengths of all program segments is greater then 64K memory size.

File Note Found - a file specified in the LOAD command does not exist or possible an invalid LOAD command operand.

Invalid Symbol - a PUBLIC command is specified that contains an invalid symbol

Load Messages

Invalid Hex Character - a character in the record shown contains an invalid hexadecimal character. Some records contains symbols as well as hexadecimal numbers. This message does not apply to those symbols in the record.

Invalid Checksum - the record has a checksum error and probably contains some changed characters.

Header Record Error - a header record was not the first record in the object module or a header record was found after the first record.

Record too large - a record specifies a record length that is greater than 72 characters.

Invalid Record Type - a record specifies a record type that does not exist in the Loader.

Invalid ID or type - some internal parameters on this record are invalid.

Address out of range - a relocation record specifies relocation at an address outside the range of relocation specified on the header record.

External Index out of Range - an External Reference is made to an external symbol that does not exist.

External Table Full - Current object module specifies more external symbols then may be contained in external table. Increase size of table.

Record Out of Sequence - a object module record was read that is out of sequence in the module or the user may have inadvertently mixed the records if they exist on cards.

Symbol Table Full - a PUBLIC object module record is being processed and the symbol table is full.

Undefined External - a reference is made to an external symbol that has not been defined in another module or by the user. The address of the external reference in the original module is listed.

Duplicate Public Name - a PUBLIC symbol is defined that has already been defined in another module. Loading will continue and the PUBLIC name will be listed.

Module Greater than 64K - during initial loading the sum of all segment lengths exceeds the 64K memory size.

Segment Overlap - due to user specified addresses one or more of the segments overlap. This is an informative message and loading continues.

LOADER INSTALLATION NOTES

These notes are desinged to help the user install the Loader and perform and modifications needed for a particular computer. The notes are separated into six sections: Program Installation, Program Modifications, Batch/Interactive Mode, Program Input/Output, Memory Requirements and Overlays, and NOVA Modifications.

A. Program Installation

1. The Loader should be compiled once and its object module stored on some secondary storage devie (disk). Compile the program in the usual manner, assigning it a name which can be refered to by an Execute or Run Statement. If upon loading the compiled program, it is discovered that not enough main memory is available to hold the entire program, refer to the section describing overlay structures.

B. Program Modifications

1. The variable IBIT corresponds to the number of bits per word in the host computer. IBIT is initially set to 16. This variable determines how many characters are packed into one host computer word for labels stored in the Loader symbol tables. The user may want to increase this variable if his machine has a longer word length. Increasing IBIT will allow a larger number of symbols to be stored in a fixed amount of memory. When initially installing the program, it is suggested that IBIT be left at 16 until the program is known to be operating correctly.

2. To increase the size of the symbol table and thus the number and length of the symbols the symbol table can hold, the user must change certain variables. The variables that must be changed depend on the number of bits per host computer word (see 2), the number of symbols in the symbol table, and the number of characters used to define a symbol. The variables that define these parameters

are described below.

IBIT - number of bits per host computer word (set by user)
MLAB - maximum label length in characters (set by user)
ICCNT - number of characters per host computer word (calculated)
IWORD - number of computer words per symbol (calculated)
LTAB - length of symbol table (set by user)

The user must change the following variables to reflect the size of the symbol table and the length of a symbol. The length of a symbol should correspond to the length set in the Assembler. The arrays to change are in COMMON, and therefore, the dimensions need to be changed in every subroutine.

ITAB(IWORD,LTAB)
ITABV(LTAB)
ITABS(LTAB)
NAME(IWORD)

where: IWORD = 1+(MLAB-1)/ICCNT
ICCNT = IBIT/8

C. Batch/Interactive Mode

1. The program is delivered with the Batch/Interactive flag, IBAT set to batch operation. In the Batch mode, commands are echoed to the listing device and all command errors are fatal, the final load does not occur. In the Interactive mode, commands are not echoed to the listing device, and some errors become non-fatal. The only fatal command errors are those that may cause some object modules to be loaded before an error is found on the LOAD command line.

D. Program Input/Output

1. The logical I/O device assignments assumed in the Loader Program are:

IPCH = 4 (object module output device, typically punch device)
ICRD = 5 (command input device, typically card reader)
IPRT = 6 (listing device, typically printer)
IMFLE = 7 (intermediate file, disk)

IFIL = 18 (input object module disk file number; when an input object module is on a file, the file name is equated to IFIL)

IRDR =__ (set dynamically during program execution to the input device specified by the LOAD command)

These device assignments may have to be changed for your system. This may be done either in the Job Control Stream or in the Program itself. If the assignments are to be changed in the program, the variables may be found in Subroutine INIT. Note the the intermedate file may be any sequential device such as a tape unit. If this is the case a REWIND IMFLE statement should be placed in the program. This statement is shown in the program near the bottom of the Main Program with a comment.

2. Reading and writing to a bulk storage device such as a disk is not standard in Fortran. See The Assembler Operation Notes for a discussion of the various methods.

3. All Program I/O activity except for generation of the output listing is handled in Subroutine INOUT. This includes the reads and writes for the intermediate file, reading the command input, reading the object module input, and writing the output object module.

4. Three are alternative ways of passing relocatable object modules from the Assembler to the Loader (see discussion in Assembler Notes). The Input devices or files that hold the object modules to be loaded by the Loader are specified as LOAD command arguments. When a disk file is specified as an argument, Subroutine EQUAT is used to equate the disk file name to the logical device, IFIL, so that the file may be read by the input statements in INOUT. There are two basic sections to the EQUAT subroutine. First, the file name is packed into a contiguous Hollerith string. The code

used to pack the characters of the file name into a string will work on any two's complement machine. For a one's complement machine, one line or code must be changed. The required change is marked with comments in subroutine EQUAT. Two variables in subroutine INIT must be set to the correct values for EQUAT to work properly. These are as follows:

ISBIT - actual number of bits in computer word. This may or may not be the same as IBIT.

ICHBT - number of bits per host computer character

The place to change these in INIT are marked with comments.

The second part of subroutine EQUAT consists of the code required to open the named disk file and equate it to the logical device number, IFIL. This code usually consists of one statement. The CALL ASSIGN statement that currently exists in the program is for a PDP-11. As mentioned in the Assembler Notes, some computers can read disk files without any special code to open the file. In this case Subroutine EQUAT may not be needed. The user will have to check the computer manuals to find out what the required statements are to perform the above functions.

5. Refer to the section on Input/Output in the Assembler Operation Notes, as many of the things discussed apply to the Loader.

6. The I/O statements needed to read in an object module may be different depending upon if the module is read from an I/O device or a file. The statements in subroutine INOUT at line number 200 have two I/O read statements, one for reading from a file and one from a device. For most machines these statements will be the same as shown. Some users may have to change one or the other. Comments in INOUT describe any changes necessary.

E. Memory Requirements and Overlays

1. The Loader program is smaller than the Assembler program. Overlaying should not be necessary. However, for users who may want to form their own Overlays or to Segment their programs, the following list shows each routine in the Loader and all the routines that call it.

```
MAIN  -
INIT  - MAIN
INOUT - MAIN,OBJ,OUT
OBJ   - MAIN
LABEL - MAIN,OBJ
SYMBL - MAIN,OBJ,LABEL
SCAN  - MAIN
NAMES - NAMES,OUT,MAIN
COMIN - MAIN
OUT   - OBJ
HEXIN - OBJ
VHEX  - OUT
AHEX  - MAIN,NAMES,ERROR
EQUAT - MAIN
ERROR - MAIN,OBJ
```

F. NOVA Modifications

When installing the Loader on a NOVA Computer, it is suggested the Fortran V be used. If Fortran IV is used, some additional program modifications have to be made.

1. Most versions of NOVA Fortran fill an H DATA specifications statement with zeros and not blanks, as is typically done. Therefore, characters read in under A formats must have the padded blanks stripped off. Insert the following statements after Fortran Statement 100 in INOUT.

```
      DO 105 I=1,80
      IN(I) = IN(I).AND.-256
105   CONTINUE
```

2. All variables initialized in DATA statements must be placed in Labeled Common. The variables are local to each Subroutine, so unique dummy labels may be used for the COMMON Block names.

3. The DEFINE FILE statement in the Main program must be replaced with a CALL OPEN statement similar to the one shown below.

```
CALL OPEN (7,"IDUM1",3,IER)
```

4. Binary READ and WRITE statements should be used for the intermediate file. To implement this change the Fortran source code in INOUT should be as follows:

```
300    READ BINARY (IMFLE)

400    WRITE BINARY (IMFLE)
```

A simplified EQUAT Subroutine for PDP-11 computers is shown below. This Subroutine may be used to replace the EQUAT Subroutine currently in the Loader.

```
      LOGICAL*1  JNAME(18)
```

REAL	leave REAL, INTEGER, and COMMON
INTEGER	statements in old Subroutine EQUAT
COMMON	in new Subroutine EQUAT

```
      IERR = 1
      K = 1
100   IF((INC(JCOL).EQ.IBLNK .OR. (INC(JCOL) .EQ. ICOMM)) GO TO 200
      IF(INC(JCOL).EQ. ICTAB) GO TO 200
      IF(K .GT. 18)  GO TO 900
      JNAME(K) = INC(JCOL)
      IPBUF(K) = INC(JCOL)
      K = K+1
      JCOL = JCOL+1
      GO TO 100
200   JNAME(K) = IBLNK
      IN(K) = IBLNK
      CALL CLOSE(IFIL)
      CALL ASSIGN(IFIL,JNAME,0,'OLD')
      IRDR = IFIL
      IERR = 0
900   RETURN
      END
```