CCSYSTM-024-LL

CONTROL DATA
CORPORATION

Z-80 LINKING LOADER REFERENCE MANUAL

CONTROL DATA®
MP-32
COMPUTER SYSTEMS

Z80LDR Control Card Format

The Z80 Cross Loader (Z80LDR) is envoked by the following Control Card:

```
     *Z80LDR(I=10,L=20,H=22)
```

The table below describes the defaults and ranges of the various parameters. Parameters may be omitted, may stand alone, or may be equated to a numeric value in the range shown.

	ABSENT	ALONE	=XX	
I	63	56	1-63	INPUT
L	62	62	1-62	LISTING
H	8	8	1-60	ABSOLUTE HEX OUTPUT

All values above are logical unit numbers. The Absolute Hex Output is in INTEL format.

ABNOMALITIES:

1. Z80LDR produces 2 extra lines of output before starting the Absolute Hex output. The lines consist of 2 dollar signs followed by 2 blanks. Most processors of INTEL hex format asbolute loads will ignore lines that do not start with colon (:) and so the extra output is not serious.

CCS-A00X-02

PROFESSIONAL SERVICES DIVISION
a consulting service of
CONTROL DATA CORPORATION

MICROTEC

Z-80 LINKING LOADER MANUAL

FLEET NUMERICAL WEATHER CENTRAL
CONSOLIDATED COMMUNICATIONS SYSTEM

REVISION RECORD

REVISION	DESCRIPTION
A	Manual released.
(10-01-79)	
Publication No. CCS-A00X-02	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Professional Services Division
205 Montecito Avenue
Monterey, Ca. 93940

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

Page	Revision
Cover	–
Title Page	–
ii thru v	A
1-1	A
2-1 thru 2-4	A
3-1 thru 3-19	A
4-1 thru 4-18	A
A-1 thru A-3	A
B-1 thru B-2	A

Page	Revision

Page	Revision

TABLE OF CONTENTS

INTRODUCTION

This manual describes Microtec's Z80 Linking Loader that accompanies the Z80 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.

<u>Absolute Segment</u> - 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 object module exactly as it is read in the input modules.

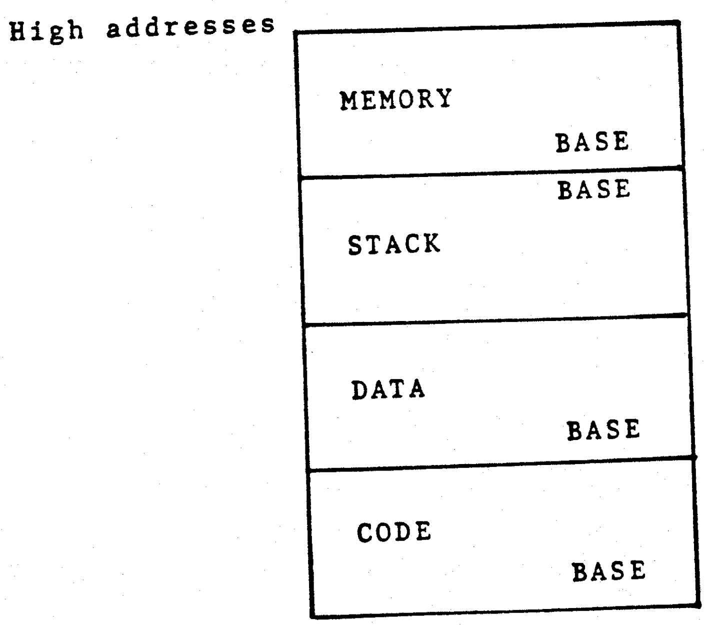
<u>Code Segment</u> - the code segment contains that part of the program which comprises actual machine instructions and which typically can be placed into ROM. Instructions in the code segment can make reference to any other segment.

Data Segment - the data segment contains specifications for that part of a users program that typically contains 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 Z80 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 program 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 address for all segments except the STACK segment 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. The BASE address for the STACK segment is the high address of the segment. This is done because during program execution the stack pointer typically moves toward lower addresses.

Relocation Types

The relocation type of any program segment is determined in the assembler by the CSEG and DSEG directives. 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 directive. 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 contiguously in memory followed immediately by all DATA segments with no extra bytes between segments. However, if any of the DATA segments specify PAGE or INPAGE relocation then the Loader must start the DATA segment 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 that starting address of each module to end in 00H and will make it easier for the user to follow the flow of the program. In this case the assembler output listing contains the correct memory addresses except for an offset that must be added to the high order address byte. This may also be accomplished in the Loader by the CPAGE and DPAGE commands.

LOADER COMMANDS

The Loader reads a sequence 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 and END command.

The object modules are read from the object module input device or files specified on the 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 stream 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 Base Address
MEMORY	Set Memor" Segment Base Address
CPAGE	Set Paging for Code Segment
DPAGE	Set Paging for Data Segment
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 the 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 Segments. 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 Segments. 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
```

```
    DATA      value
```

where:

value - specifies the starting address of the DATA segment.

STACK — Set Stack Segment starting Address

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

Note that the BASE address specified by this command is the high address of the Stack Segment.

Example:

```
                    STACK   3FFH
```

```
    STACK     value
```

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 segmnent will be specified as zero on the load map but it is actually the length of available memory remaining in a users 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
```

```
    MEMORY     value
```

where:

value - specifies the starting address of the MEMORY segment.

CPAGE — Set Paging for Code Segment

This command may be used to modify the relocation type of code segments in the input object modules. As explained under Relocation Types (page 2-3), the assembler indicates to the Loader the relocation type as byte, page, or inpage for each segment in each object module. This command allows the user to override that relocation type specified by the assembler.

The typical use of this command is to allow the user to begin each module on a page boundary for ease of debugging and then to specify the final program as byte relocation to avoid any wasted memory space. This command allows the user to avoid reassembling each module and changing only the relocation type.

This command allows the user to specify the code segment of each module to be byte or page relocatable regardless of the type of relocation specified by the assembler. Inpage relocation is not affected. Note that the command also allows the relocation type specified by the assembler to be used by the loader. This command may be changed for each module read by the Loader. The last CPAGE command will be used.

Example:

```
                    CPAGE   ON
```

```
     CPAGE       {blank,ON,OFF}
```

where:

blank	- specifies that the relocation type will be that specified in the Assembler. This is the Loader default.
ON	- specifies that the code segment of successive modules will be placed on a page boundary.
OFF	- specifies that the code segment of successive modules will be adjusted to byte relocation.

DPAGE — Set Paging for Data Segment

This command may be used to modify the relocation type of data segments in the input object modules. This command is used in the same way as the CPAGE command and allows the user to specify the data segment of each module to be byte or page relocatable regardless of the type of relocation specified by the assembler. Inpage relocation is not affected.

This command may be changed for each modules read by the Loader. The last DPAGE command will be used.

Example:

```
                    DPAGE   OFF
```

```
    DPAGE      {blank,ON,OFF}
```

where:

blank	- specifies that the relocation type will be that specified in the Assembler. This is the Loader default.
ON	- specifies that the data segment of successive modules will be placed on a page boundary.
OFF	- specifies that the data segment of successive modules will be adjusted to byte relocation.

ORDER — Sepcify 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 the segments to be placed in memory in a different order. If the user specifies starting addresses for the segments, the order of the segments if of no particular importance. If the user specifies starting addresses for only some of the segments, the remaining segments will be placed in the order specified by this command.

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 object 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 to 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
```

```
     STKLN     value
```

where:

value - specifies the length of the STACK segment.

NAME — Sepcify 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 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
```

```
     NAME       name
```

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 I/O device. If the command operand is not a number, it is assumed that the name of a disk file is being specified and the object module will be read from the file. If any operand is preceded by a minus sign, it indicates that the object modules should be read from the specified device or file until an end-of-file condition (EOF) is detected (see Installation Notes concerning modofications for EOF). In this case the user need not specify an operand for each object module.

Object modules may be read from a combination of files and peripheral devices and may or may not be read until the EOF. The object modules are loaded in the order specified with each module being loaded into memory at a higher address then the preceding module. A user may use as many LOAD commands as needed.

Example:

`LOAD    7,-FILE1,7`	Four modules are to be loaded. The first from unit 7, the next two from FILE1 until an EOF, and finally the last from unit 7.

LOAD module$_1${,module$_2$, ..., module$_i$}

where:

module$_i$ - specifies the number of a logical input device or the name of a disk file on which the object module resides. Any module specification preceded by a minus sign will read object modules until an EOF is detected on the device or file. 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 specified 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 an 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 $sym_1=val_i\{,sym_2=val_2, \ldots, sym_i=val_i\}$

where:

sym_i - is user defined PUBLIC symbol which may already be defined by some object module.

val_i - is the value given to 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 object module is produced, no symbols are placed in the output object module, and local symbols are not purged from the input modules. The user should note that placing both PUBLIC and local symbols into the output object module symbol table could cause duplicate symbols to exist in the module. Typically only the local symbols placed into the object modules by the assembler or PUBLIC symbols will be placed into the output object module. This is the case since using the B option in the assembler forms a symbol table which includes PUBLIC symbols.

Example:

```
                    LIST    T,X          list both local and
                                         PUBLIC symbol tables
```

```
      LIST      D,O,P,S,T,X
```

where:

D - specifies that PUBLIC symbols will be placed into the output object module.

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 the local symbol table be listed on the list output device.

X - specifies that the PUBLIC symbol table 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     D,O,P,S,T,X
```

where:

D - specifies that PUBLIC symbols will not be placed into the output object module. (default)

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

P - specifies that any local 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. (default)

T - specifies that the local symbol table not be listed on the list output device. (default)

X - specifies that the PUBLIC symbol table not be listed on the list output device. (default)

__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 commmand 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.

```
        EXIT
```

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. It initiates the final steps in linking and loading the input modules. An exit is than made from the program.

```
        END
```

<u>Comment</u> — 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:

```
*   SAMPLE LOADER PROGRAM
```

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 the the last subroutine is followed by a tape mark. The Loader is located after the assembler and assembler test program on the tape.

The Loader Installation Notes describe program installation and any modifications that may have to take place for a particular computer. It is extremely 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 along with their address in the object module. 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. It is extremely useful to use the read until EOF option on the LOAD command if the end-of-file can be detected on the particular computer.

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. This example is also used as the Loader Test Program.

The first page of the output listing lists all commands entered by the user along with any command errors that occur. 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 //. 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 slashes. When the starting and final addresses are the same, it implies that the length of the segment is zero. Following this is a list of all absolute segments in the object module along with the starting and ending addresses. It is possible that all absolute segments will not be shown if certain Loader tables become full.

Following the Load Map is a list of all PUBLIC symbols as well as local symbols if the user specified the appropriate

LIST 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 in the assembler. These may be used for debugging but serve no function to the Loader.

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. This is indicated by the name of the module that contains the undefined external, the address of the undefined external in the input object module, the segment type and name of the external.

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

```
                 Z8 LINKING LOADER VER 2.0

     **LOADER COMMANDS


   *
   * TEST PROGRAM FOR Z80 LOADER
   *
   * NOTE THE OBJECT MODULES ARE READ IN FROM THE SAME
   * DEVICE AS THE COMMAND STREAM.  TO READ THE OBJECT MODULE
   * FROM A DIFFERENT DEVICE THE LOAD COMMANDS MUST BE
   * CHANGED TO THE NEW DEVICE NUMBER.  ALSO IF THE USERS
   * COMMAND DEVICE IS NOT 5, THE LOAD COMMANDS MUST ALSO BE
   * CHANGED.
   *
   .IST  T,S,X
   DATA 407H
   CODE 605H
   ORDER C,S,D,M
   STACK A00H
   STKLN 12
   .OAD  5,5
   .OAD  5
   END
                          **LOAD MAP**

                            MODULE     CODE  DATA

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


                            ABSOLUTE SEGMENTS

                                       0008  000F
```

```
  **PUBLIC SYMBO .

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


  **LOCAL SYMBOLS

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

**MODULE   MAIN
  UNDEFINED EXTERNALS
  0011 C - SCAN

  **LOAD COMPLETED
```

LOADER EXAMPLE

```
$$
 ASCR     0000DH   BLNK     00020H   BSPA     00008H   READ     0063FH
 READ10   00644H   READ20   00652H   READ30   0065FH   READ40   00669H
 READ50   00673H   READ60   0067DH   READ70   00680H   READ80   00686H
 TAB      00008H
$$
:10060500031000AC03F062107047EFE2023C20E0362F
:10061500CD00023C30506DB00E002CA1C060DB000D
:10062500E67F47C9D800E001CA2906780300C90678
:10063500D0C02906000AC029060C92107041E00CDC0
:1006450001CC6FE16C2520EC0340EC33F06FE0DC277
:100655005F067BB7CA4406360DC9FE7FC2730670AB
:10066500B7CA44062B100608CD2900C38006FE0819
:10067500CA7000FE20DA800677241C7BFE57CA69F0
:0E0685000C3A5704B7CA4406CD2906C34400F8
:080698000E5C52A0B05C3000148
:1C069300002100003A0B05B7C2A000602F210E056A
:0106A30076E0
:0F050000C3A00601CB0580A0060B00506A00096
:00060501F4
```

LOADER EXAMPLE OUTPUT OBJECT MODULE

Loader Example

The following pages show three assembly listings of programs that will be combined by the Loader. The actual Load is shown on the preceding pages. 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 does contain some absolute code which will be used for a RST instruction during execution.

The Command stream on the preceding pages shows that the user has specified the starting addresses of both the CODE and DATA segments in addition to changing the order of the segments to CODE,STACK,DATA, and MEMORY. The LIST command is then used to obtain a symbol table listing of both local and PUBLIC symbols as well as placing the local symbols into the output object module. Finally the LOAD command is used to read the three modules from the device shown.

The Load Map shows the starting and ending addresses of the three modules in the order loaded. Note that the third module had specified a "DSEG PAGE" directive in the assembly 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, relocation type and name is specified. It can be seen that SCAN 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 and local symbols used in the program along with their absolute addresses is listed. 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 value and any links between modules resolved.

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

The final absolute object module from the example is shown along with the local symbols that were placed into the module.

```
ERR LINE  ADDR  B1 B2 B3 B4                Z80 ASSEMBLER VER . .R                          PAGE    1

     1                                         LIST      R
     2                                         NAME      MAIN
     3                                         PUBLIC    INBUF,INBUFEND,TIN,TOUT,CRLF,ECHO
     4                                         EXTRN     READ,SCAN
     5                                ;
     6                                ; THIS IS A SAMPLE PROGRAM THAT SHOWS MOST OF THE RELOCATABLE
     7                                ; FEATURES OF THE ASSEMBLER.  THREE MODULES ARE LINKED TOGETHER
     8                                ; TO FORM THE FINAL PROGRAM.  PUBLICS AND EXTERNALS ARE USED
     9                                ; TO PERFORM THE LINK.
    10                                ;
    11                                ; BELOW IS THE MAIN PROGRAM AND I/O DRIVERS.  THIS IS
    12                                ; LINKED TO A ROUTINE WHICH READS A LINE OF CODE AND WHICH
    13                                ; ITSELF REQUIRES THE I/O DRIVERS.
    14                                ;
    15                                ;
    16                                         CSEG                          ;SET CODE SEGMENT
    17                                ;
    18   0000  31 00 00      S        MAIN:    LD        SP,STACK            ;SET STACK POINTER
    19   0003  CD 00 00      E                 CALL      READ                ;READ NEXT LINE
    20   0006  21 00 00      D                 LD        HL,INBUF            ;START OF BUFFER
    21   0009  7E                     MAIN10:  LD        A,(HL)
    22   000A  FE 20                           CP        BLNK                ;CHECK FOR NON BLANK
    23   000C  23                              INC       HL
    24   000D  C2 09 00      C                 JP        NZ,MAIN10
    25   0010  CD 00 00      E                 CALL      SCAN                ;GET VALUE
    26   0013  23                              INC       HL
    27   0014  C3 00 00      C                 JP        MAIN
    28                                ;
    29                                ; NAME - IN8
    30                                ;
    31                                ; THIS ROUTINE WILL INPUT A CHARACTER FROM THE TERMINAL
    32                                ;
    33                                ; ENTRY PARAMETERS
    34                                ;   NONE
    35                                ;
    36                                ; EXIT PARAMETERS
    37                                ;   A           - INPUT CHARACTER
    38                                ;   B           - SAME AS A
    39                                ;
    40                                ; REGISTERS USED
    41                                ;   A,B
    42                                ;
    43                                ;
    44   0017  DB 00                  IN8:     IN        A,(USTAT)           ;READ UART STATUS
    45   0019  E6 02                           AND       RRDY                ;CHECK IF READY
    46   001B  CA 17 00      C                 JP        Z,IN8               ;NOT READY YET
    47   001E  DB 00                           IN        A,(UDATIN)          ;READ DATA
    48   0020  E6 7F                           AND       127                 ;DELETE PARITY BIT
    49   0022  47                              LD        B,A
    50   0023  C9                              RET
    51                                ;
    52                                ; NAME - OUT8
    53                                ;
    54                                ; THIS ROUTINE IS USED TO OUTPUT A CHARACTER TO
```

4-7

```
ERR LINE  ADDR  B1 B2 B3 B4

    55                          ; THE TERMINAL
    56                          ;
    57                          ; ENTRY PARAMETERS
    58                          ;   B          - CHARACTER TO OUTPUT
    59                          ;
    60                          ; EXIT PARAMETERS
    61                          ;   NONE
    62                          ;
    63                          ; REGISTERS USED
    64                          ;   A,B
    65                          ;
    66                          ;
    67  0024  DB 06             OUT8:   IN      A,(USTAT)        ;READ STATUS
    68  0026  E6 01                     AND     TRDY             ;CHECK IF READY
    69  0028  CA 24 00      C           JP      Z,OUT8           ;NOT READY
    70  002B  78                        LD      A,B
    71  002C  D3 06                     OUT     (UDATOUT),A      ;OUTPUT DATA
    72  002E  C9                        RET
    73                          ;
    74                          ; NAME - CRLF
    75                          ;
    76                          ; THIS ROUTINE OUTPUTS A CARRIAGE RETURN
    77                          ; AND LINE FEED
    78                          ;
    79  002F  06 0D             CRLF:   LD      B,ASCR
    80  0031  CD 24 00      C           CALL    OUT8
    81  0034  06 0A                     LD      B,ASLF
    82  0036  CD 24 00      C           CALL    OUT8
    83  0039  C9                        RET
    84                          ;                                ;SET DATA SEGMENT
    85                                  DSEG                     ;INPUT BUFFER
    86  0000                    INBUF:  DEFS    80               ;END OF BUFFER
    87                          IBUFEND:                         ;ECHO FLAG
    88  0050                    ECHO:   DEFS    1                ;USART STATUS
    89  0006                    USTAT   EQU     6                ;USART OUTPUT
    90  0006                    UDATOUT EQU     6                ;USART INPUT
    91  0006                    UDATIN  EQU     6                ;TRANSMIT READY
    92  0001                    TRDY    EQU     1                ;READER READY
    93  0002                    RRDY    EQU     2
    94  000D                    ASCR    EQU     13
    95  000A                    ASLF    EQU     10
    96  0020                    BLNK    EQU     20H
    97  0017                C   TIN     EQU     IN8
    98  0024                C   TOUT    EQU     OUT8
    99  0051                            END     MAIN

ASSEMBLER ERRORS =    0
```

```
                        CROSS REFERENCE

LABEL     VALUE        REFERENCE

ASCR        000D        79    -94
ASLF        000A        81    -95
BLNK        0020        22    -96
CRLF      C 002F         3    -79
ECHO      D 0050         3    -88
IBUFEN    D 0050         3    -87
IN8       C 0017       -44     46     97
INBUF     D 0000         3     20    -86
MAIN      C 0000       -18     27     99
MAIN1G    C 0009       -21     24
MEMORY    H 0000         0
OUT8      C 0024       -67     69     86     82     98
READ      E 0000         4     19
RRDY        0002        45    -93
SCAN      E 0001         4     25
STACK     S 0000         3
TIN       C 0017         3    -97
TOUT      C 0024         3    -98
TRDY        0001        68    -92
UDATIN      0000        47    -91
UDATOU      0006        71    -90
USTAT       0000        44     67    -89
```

```
022EuuuuбMAIN  u  uu13AuuuJJ231uuuu3u3uuuu0u3uuuuu0uJ3n0
1816uuu6READ**uuu6SCAN**uu35
162Euuu1:7uuJ6TIN***uu2FuuJ6CRLF**jG24uuJu6TOUT**uu33
162EuuG2buuuuu6ECHO**uuuuuuuu6INBUF*JJ5uuuJu6IBUFENuuE6
0614uuu1uuuu31uuuuCDJuuuE7
24uAuuG3u3u1uuCB
200CuuG3uuJuu4uJCD
u640Guu01uбuu21uJuu7EFE2C23C2u9uuCDuuuu23C3JuuuuBGuE6u2CA170JuBuuE67F71
2210uuu3uEuG15uu1CuuBC
24uAuuG2u3u7uuC6
20uCuuu3C1uuu1:JuBF
06380uuu122uC47C9DBuuE601CA24uu78D3u0C9060DCD24JJ060ACD24uJC9FD
2210uu03u9u032u037uJ39
04uAuuG1u1CuuuFJ
0EG2uuFJ
```

MAIN OBJECT MODULE

```
ERR LINE  ADDR  B1 B2 B3 B4
       1                                  NAME      READ
       2                                  CSEG                          ;SET CODE SEGMENT
       3                                  LIST      X
       4                                  LIST      B
       5                                  PUBLIC    READ
       6                                  EXTRN     CRLF,TIN,TOUT,ECHO,INBUF,IBUFEND
       7                          ;
       8                          ; NAME - READ
       9                          ;
      10                          ; THIS ROUTINE READS IN A LINE FROM THE TERMINAL AND
      11                          ; PLACES IT INTO THE INPUT BUFFER.  THE FOLLOWING ARE
      12                          ; SPECIAL CHARACTERS.
      13                          ;   CR          - END OF CURRRNT LINE
      14                          ;   CONTROL X - DELETE CURRENT LINE
      15                          ;   DEL         - DELETE LAST CHARACTER
      16                          ; ALL DISPLAYABLE CHARACTERS BETWEEN BLANK AND Z AND
      17                          ; THE ABOVE SPECIAL CHARACTERS ARE RECOGNIZED BY THIS
      18                          ; ROUTINE AS WELL AS THE TAB.  ALL OTHER CHARACTERS ARE
      19                          ; IGNORED.  AN ATTEMPT TO INPUT MORE CHARACTERS THEN IS
      20                          ; ALLOWED IN THE INPUT BUFFER WILL BE INDICATED BY A BACKSPACE.
      21                          ;
      22                          ; ENTRY PARAMETERS
      23                          ;   ECHO        - ECHO FLAG, 0 = NO ECHO
      24                          ;
      25                          ; EXIT PARAMETERS
      26                          ;   INBUF       - ONTAINS INPUT LINE
      27                          ;
      28                          ; REGISTERS USED
      29                          ;   A,B,E,H,L
      30                          ;
      31                          ;
      32  0000  21 00 00      E   READ:    LD       HL,INBUF            ;INPUT BUFFER ADDRESS
      33  0003  1E 00                      LD       E,0                 ;SET CHARACTER COUNT
      34  0005  CD 00 00      E   READ10:  CALL     TIN                 ;READ NEXT CHARACTER
      35  0008  FE 18                      CP       24                  ;CHECK FOR CONTROL X
      36  000A  C2 13 00      C            JP       NZ,READ20           ;NOT CONTROL X
      37  000D  CD 00 00      E            CALL     CRLF
      38  0010  C3 00 00      C            JP       READ                ;START AGAIN
      39  0013  FE 0D             READ20:  CP       ASCR                ;CHECK IF CR
      40  0015  C2 20 00      C            JP       NZ,READ30           ;NO
      41  0018  7B                         LD       A,E                 ;GET COUNT
      42  0019  B7                         OR       A                   ;CHECK IF ANY INPUT
      43  001A  CA 05 00      C            JP       Z,READ10            ;KEEP READING
      44  001D  36 0D                      LD       (HL),ASCR           ;PUT CR AT END OF LINE
      45  001F  C9                         RET
      46  0020  FE 7F             READ30:  CP       127                 ;CHECK FOR DELETE
      47  0022  C2 34 00      C            JP       NZ,READ50           ;NOT DELETE
      48  0025  7B                         LD       A,E                 ;GET COUNT
      49  0026  B7                         OR       A
      50  0027  CA 05 00      C            JP       Z,READ10            ;NO ENTRIES YET
      51  002A  2B                READ40:  DEC      HL
      52  002B  1D                         DEC      E                   ;DECREMENT COUNT
      53  002C  06 08                      LD       B,BSPA              ;GET A BACKSPACE
      54  002E  CD 00 00      E            CALL     TOUT                ;OUTPUT BACKSPACE
```

```
ERR LINE  ADDR   . 02 03 B4
    55   0031   C3 41 00     C            JP      READ70      ;CHECK FOR A TAB
    56   0034   FE 08              READ50: CP      TAB
    57   0036   CA 3E 00     C            JP      Z,READ60
    58   0039   FE 20                     CP      BLNK
    59   003B   DA 41 00     C            JP      C,READ70    ;PUT CHARACTER INTO BUFFER
    60   003E   77                 READ60: LD      (HL),A
    61   003F   24                        INC     H           ;INCREMENT COUNT
    62   0040   1C                        INC     E           ;GET COUNT
    63   0041   73                 READ70: LD      A,E         ;CHECK FOR END OF BUFFER
    64   0042   FE 00        E            CP      .LOW.IBUFEND ;HAVE END
    65   0044   CA 2A 00     C            JP      Z,READ40    ;GET ECHO FLAG
    66   0047   3A 00 00     E     READ80: LD      A,(ECHO)
    67   004A   B7                        OR      A           ;DONT ECHO CHARACTER
    68   004B   CA 05 00     C            JP      Z,READ10    ;ECHO CHARACTER
    69   004E   CD 00 00     E            CALL    TOUT        ;CONTINUE
    70   0051   C3 05 00     C            JP      READ10
    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 =    6
```

```
                              CROSS REFERENCE

LABEL     VALUE        REFERENCE

ASCR        000D         39    44   -72
BLNK        0020         58   -74
BSPA        0008         53   -73
CRLF      E 0000          6    37
ECHO      E 0003          6    66
IBUFEN    E 0005          6    64
INBUF     E 0004          6    32
MEMORY    M 0000          0
READ      C 0000          5   -32    38
READ10    C 0005        -34    43    50    68    70
READ20    C 0013         36   -39
READ30    C 0020         40   -46
READ40    C 002A        -51    65
READ50    C 0034         47   -56
READ60    C 003E         57   -60
READ70    C 0041         55    59   -63
READ80    C 0047        -66
STACK     S 0000          0
TAB         0008         56   -75
TIN       E 0001          6    34
TOUT      E 0002          6    54    69
```

```
022E0006REA[illegible]  .  .0015400030200000303000000304000000031C
183E0006CRLF**0000TIN***00006TCUT**00006ECHO**0006INBUF*00006IBUFEN0058
1012000100000006READ**0000D
123C00000008000006SPA**0020000068LNK**0000D00006ASCR**000000006TAB***0003A
123C00001000000006READ**00005J0006READ1000013C0006READ20000200006READ300045
123C000012A00006READ4000340006READ5C0003E00006READ6000041000006READ70008E
12126001470000READ8000040
063C000010000210000EA00CD0000FE18C213000CD00000C3000000FE00DC22000078B717
22100003000011000160099
201C00030400001C0010000000000000E30A7
064C000011A000CA05300300DC9FE7FC2340078B7CA05000281DG0008CD0000C341000FE0823
2214000320002300020003200002F
200C0000302002F00A0
062A000136000CA3E000FE200DA4100077241C78FE00CA2A00034
2210000337003C00045013
200C00001005004300C08
0622000147003A00000B7CA0500C000000C305003B
220C000034C000520031
201400003030040000200C4F0002D
040A00000C10000F1
0E0200F0
```

READ OBJECT MODULE

```
ERR LINE  ADDR  B1 B2 B3 B4                     Z80 ASSEMBLER VER . ..R                     PAGE    1

       1                                  LIST      X
       2                                  ASEG
       3                                  ORG       8                 ;RST 8
       4  0008  E5                         PUSH      HL
       5  0009  C5                         PUSH      BC
       6  000A  2A 0B 00      D            LD        HL,(DATA)
       7  000D  C3 00 01                   JP        100H
       8                                  CSEG
       9  0000  00                         NOP
      10  0001  21 00 00                   LD        HL,0
      11  0004  3A 0B 00      D            LD        A,(DATA)
      12  0007  B7                         OR        A
      13  0008  C2 0D 00      C            JP        NZ,LAB1
      14  000B  00                         NOP
      15  000C  2F                         CPL
      16  000D  21 0E 00      D    LAB1:   LD        HL,DATA+3
      17  0010  76                         HALT
      18                               ;
      19                                  DSEG      PAGE
      20  0000  C3 0D 00      C            JP        LAB1
      21  0003  01 0B 00      D            LD        BC,DATA
      22  0006  80                         ADD       A,B
      23                               ;
      24  0007  0D 00         C            DEFW      LAB1
      25  0009  0B 00         D            DEFW      .LOW.DATA
      26  000B  05                 DATA:   DEFB      5,6,.LOW.LAB1
      27  000C  06
      28  000D  0D
      29  000E  00                         NOP
      30  000F                             END

ASSEMBLER ERRORS =    0
```

```
                                   Z80 ASSEMBLER VER  . 4R              PAGE    2

                 CROSS REFERENCE

LABEL     VALUE       REFERENCE

DATA    D 0008           6     11     16     21     25    -26
LAP1    C 0000          13    -16     20     24     26
MEMORY  M 0000           0
STACK   S 0000           0
```

```
022E00C6MODUL..  0011100003C2DF0D02D3000003C4C00000313
0E1800C0C800E5052A0000C3000137
24CA000203080 0C2
062A00C10000002100000A0B00B7C20D00002F010E007E0F
22C800030900CA
240E00020305000E00B6
061A000200000C30D000010B00800D0075
220800030400CF
240E00C10301000700C2
061400020900080005060D00B8
22C800010900CC
240A0001010D00C3
04CA0000010000F1
0E0200F0
```

MODULE OBJECT MODULE

```
          80 LINKING LOADER VER 2.0

  **LOADER COMMANDS


*
* TEST PROGRAM FOR Z80 LOADER
*
* NOTE THE OBJECT MODULES ARE READ IN FROM THE SAME
* DEVICE AS THE COMMAND STREAM.  TO READ THE OBJECT MODULE
* FROM A DIFFERENT DEVICE THE LOAD COMMANDS MUST BE
* CHANGED TO THE NEW DEVICE NUMBER.  ALSO IF THE USERS
* COMMAND DEVICE IS NOT 5, THE LOAD COMMANDS MUST ALSO BE
* CHANGED.
*
LIST  T,S,X
DATA 407H
CODE 605H
ORDER C,S,D,M
STACK A00H
STKLN 12
LOAD  5,5


**MODULE   MAIN
   RECORD OUT OF SEQUENCE
   RECORD    5 - 240A0003630100C8
LOAD  5


**MODULE
   HEADER RECORD ERROR
   RECORD    1 - 0E1800000000E5C52A0B00C300137
END


  **LOAD NOT COMPLETED
```

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 batch mode. Command messages are listed beneath the actual command on the output listing. Load messages occur during the loading of the 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 than the 64K memory size.

File Not 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 contain 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 - an 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 name, relocation type, and address of the symbol 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. An absolute segment could also overlap a relocatable segment.

Unexpected end of Module - the user has used the EOF option on the LOAD command and an end-of-file condition has occurred before the current module end record. Possibly some of the information in the load module is out of order or not in the load module. User should reassemble the module and check that device or file contains proper load modules. Program termination occurs for this error.

APPENDIX B

OBJECT MODULE FORMATS

As part of the output processing, the Loader produces an absolute object module. This object module is a machine readable computer output in the form of punched cards, paper tape, etc. The output module contains specifications for loading the memory of the target microprocessor.

The object module produced by the Loader uses the standard Intel hexadecimal format. This was done for a number of reasons. The object module in this format contains its own load address. The user may easily create their own object records for patches. This is the format used by some Z80 manufacturers such as MOSTEK Finally the object module does not contain any special characters such as those used by the Zilog development system.

The object module is normally punched out on the device specified. However, through use of the NLIST and LIST directive the output module may be deleted.

The object module is produced as a series of card images on the output punch device. Each object record contains the load address and data specifications for up to 16 bytes of data. Symbol table information may also be included. The format of an object module is shown below.

```
$$
 symbol records
$$
 data records
```

A sample symbol record is shown below:

```
APPLE   00000H   LABEL1   0D0C3H   MEM     0FFFFH
```

As many symbols records as needed may be contained in the object module. At most 4 symbols per line are used but each line need not contain 4 symbols even if it is not the last line. A module may contain no symbol records in which case the "$$" records will still be contained in the module.

The format for a data record is shown below.

```
1  2  3  4  5  6  7  8  9  10 11 ... 40 41 42 43

:  byte     load     type   data      data checksum
   count  address
```

Column 1 contains the code for a colon. This marks the beginning of an object data record.

Column 2 and 3 contain the count of the number of data bytes on the record. If this field contains an "00" it signifies the end of the object module.

Columns 4 through 7 contain the load address expressed as hexadecimal digits. The first data byte is to be loaded into this address, subsequent data bytes into the next sequential addresses. Columns 4 and 5 contain the most significant byte of the address.

Columns 8 and 9 contain the record type. Presently two types are defined. "00" indicates a data record. "01" indicates a terminator record. In this case the byte count will also be zero and the load addresses will actually be the starting address.

Columns 10 to 41 (or less if less data) contain the hexadecimal specifications for up to 16 bytes of data.

The last two columns in the record contain a checksum. The checksum is the negative of the sum of all bytes on the record (except column 1(evaluated modulo 256.

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MN 55440
SALES OFFICE AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

CONTROL DATA CORPORATION