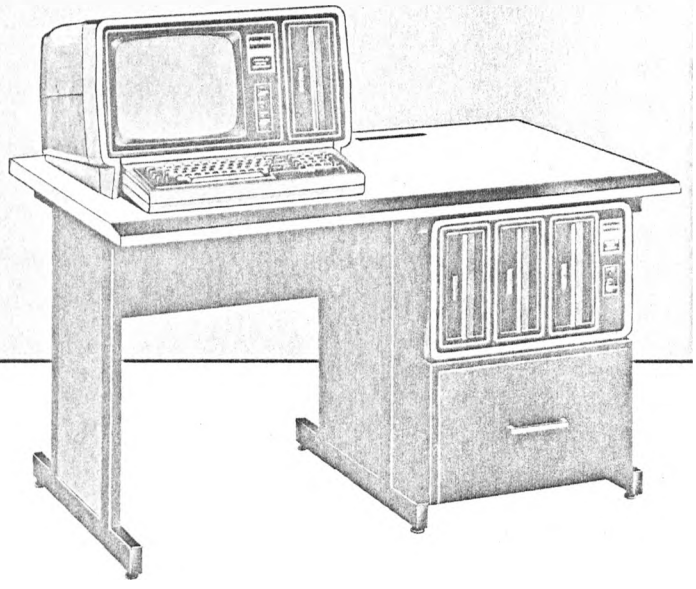
Radio Shack®

TRS-80™ Model II
Assembly Language Development System



CUSTOM MANUFACTURED IN THE U.S.A. BY RADIO SHACK, A DIVISION OF TANDY CORPORATION

ALDS

ASSEMBLY LANGUAGE

DEVELOPMENT SYSTEM

TO OUR CUSTOMERS,

The Assembly Language Development System (ALDS) is a powerful tool for developing Z80 programs for the TRS-80 Models I, II, III, and the Model 16 in the Model II mode.

It contains these five systems:

ALEDIT, a Text Editor, for writing and editing source programs.

ALASM, an Assembler for converting source programs to Z80 object code. The Assembler contains more than:

- 50 powerful directives. Among many features, they allow you to build relocatable program sections, macro sections, index sections; generate a length byte for text storage; and control the assembly listing format
- 30 arithmetic, logical and relational operators.
- 10 "extended" Z80 mnemonics, which expand into an entire group of Z80 mnemonics.

ALLINK, a Linker, for linking relocatable program sections into absolute object files

ALBUG, a Debugger, for debugging a program in memory or altering a file on disk. ALBUG is comprised of six program files: ALBUG, ALBUG/SYS, ALBUG/OVL, ALBUGX, ALBUGRES/REL and ALBUG/RES.

ALTRAN, a File Transfer System, for transferring a file between the Models I, II, III, and 16.

For your convenience, it also contains:

Three source files for referencing Models I, II, or III Supervisor Calls (see REF, in Chapter 8, to find out how):

- SVCNUM1/SRC, for Model I Supervisor Calls
- SVCNUM2/SRC, for Model II Supervisor Calls
- SVCNUM3/SRC, for Model III Supervisor Calls

XSAMPLE/SRC, a source file documenting all the Z80 mnemonics and extended mnemonics.

The Model II ALDS Package comes on three diskettes, allowing you to create a program on the Model II and transfer it to the Models I and III:

- the Model II ALDS diskette contains all the systems described above.
- the Model I diskette contains ALTRAN only.
- the Model III diskette contains ALTRAN only.

ABOUT THIS MANUAL

This manual assumes you already know Z80 assembly language programming and have used an editor/assembler. It contains three sections:

Section I, USING ALDS, begins with a sample session which shows how to create a modular program for the Models II and III using all five systems. Following this session are reference chapters on each system.

Section II, ALDS ASSEMBLY LANGUAGE, references the source language acceptable to the ALDS Assembler. Chapter 7 outlines the syntax for writing source lines. The remaining chapters reference all the directives, Z80 mnemonics, and extended Z80 mnemonics available.

Section III, ERROR MESSAGES, lists the error messages that may be generated by the ALDS programs.

If you are new to Z80 assembly language programming, we suggest you read:

TRS-80 Assembly-Language Programming by William Barden, Jr. (Radio Shack Catalog Number 62-2006)

NOTATION KEY

The manual uses these notational conventions:

DOT MATRIX	to represent what you will see on the screen or should type
<KEY>	to represent a specific key you should press
italics	to represent a value you should specify
H	to represent a hexadecimal number (for example, 4233H represents the hexadecimal number 4233.)
$	to represent the current value of the Assembler's location counter (this is actually a convention of the Assembler)
filespec	to represent a valid TRSDOS file specification. (See your TRSDOS manual for a definition of filespec.)

TABLE OF CONTENTS

SECTION I/ USING ALDS

Chapter 1/ Sample Session.....11
Chapter 2/ The ALDS Editor.....19
Chapter 3/ The ALDS Assembler.....37
Chapter 4/ The ALDS Debugger.....43
Chapter 5/ The ALDS Linker.....67
Chapter 6/ The ALDS File Transfer System....71

SECTION II/ ALDS ASSEMBLY LANGUAGE

Chapter 7/ Assembly Language Syntax.....91
Chapter 8/ Directives.....1Ø1
Chapter 9/ Z8Ø Mnemonics.....161
Chapter 1Ø/ Extended Z8Ø Mnemonics.....347

SECTION III/ ERROR MESSAGES

APPENDICES

Appendix A/ Undocumented Z8Ø Instructions...385
Appendix B/ Memory Map.....391
Appendix C/ Converting a Series I Editor Assembler Program.....393
Appendix D/ ALDS Object Code Format.....394
Appendix E/ Model I TRSDOS 2.3B.....398
Appendix F/ Numeric List of Z8Ø Instruction Set.....4Ø6
Appendix G/ Alphabetic List of Z8Ø Instruction Set.....412
Appendix H/ Z8Ø Hardware.....418

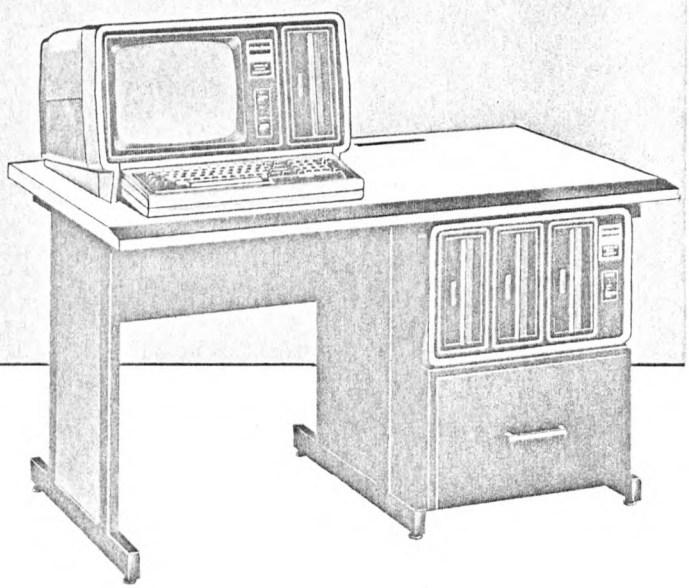
TABLES

Table 1/ ALEDIT Insert Control Functions.....22
Table 2/ ALEDIT Insert Mode Special Keys 23
Table 3/ ALEDIT Line Edit Mode Subcommands....24
Table 4/ ALEDIT Line Edit Mode Special Keys...26
Table 5/ ALEDIT Command Mode Keys.....28
Table 6/ ALEDIT Commands.....29
Table 7/ ALASM Switches.....39
Table 8/ Debugger Commands.....51
Table 9/ Baud Rate Change Table.....73
Table 1Ø/ Operators.....95
Table 11/ Complex Expressions Allowing
Relocatable or External Symbols.....98

Radio Shack®

Section 1
Using ALDS

Editor, Assembler, Debugger,
Linker, and File Transfer System



CUSTOM MANUFACTURED IN THE U.S.A. BY RADIO SHACK, A DIVISION OF TANDY CORPORATION

SECTION I/

USING ALDS

Chapter 1/ SAMPLE SESSION

This chapter is for those of you who want to try a session using the entire ALDS package. It demonstrates how to link separate program sections for the Models II and III.

The session is for demonstration only. To find out how and why each system works the way it does, you will need to refer to specific chapters in this manual.

CREATING A SOURCE FILE

In this session, you need to create five source program files. To do this, use the ALDS Editor. In the TRSDOS READY mode, type:

```
ALEDIT <ENTER>
```

this loads the ALDS Editor. After it displays its heading, type:

```
I
```

the insert command (Do not press <ENTER>). The Editor clears the screen and prints NONAME/SRC in the upper right-hand corner. You are now in the insert mode and can insert the first source program.

1. MAIN PROGRAM

To insert the first program, named MAIN, type:

```
MAIN    PSECT
        PUBLIC    BEGIN
        EXTERN    PRINT,TRSDOS
BEGIN   LD        HL,MSG1
        CALL      PRINT
        LD        HL,MSG2
        CALL      PRINT
        JP        TRSDOS
MSG1    DEFT      'YOU WILL BE ABLE TO LINK THIS'
        DEFB      0DH
```

```
MSG2    DEFT    'AS EITHER A MODEL II OR III PROGRAM'
        DEFB    0DH
        END     BEGIN
```

(Press <TAB> between columns; press <ENTER> at the end of each line.)

When you are finished press <BREAK>. This puts you in the Editor command mode. If you made mistakes, you can use the Editor commands to edit the program. They are all listed in Chapter 2, The ALDS Editor.

After pressing <BREAK>, save this source program on disk by typing this Editor command:

```
W MAIN <ENTER>
```

this saves the program as a source file named MAIN/SRC (the Editor changes the top right-hand corner display to MAIN/SRC.) Clear the edit buffer by typing:

```
K <ENTER>
```

the kill command and answer Y <ENTER> to the prompt. The screen will then clear.

Now repeat the same procedures for inserting and saving MODII, MODIII, PROGII, and PROGIII. (Insert all of these programs on the Model II -- even MODIII and PROGIII.)

2. MODII PROGRAM

```
MODII   PSECT   ;model II print routine
        PUBLIC  PRINT,TRSDOS
VDLINE  EQU     9
JP2DOS  EQU     36
PRINT   LD      B,(HL)
        LD      C,0DH
        INC     HL
        SVC     VDLINE
        RET
TRSDOS  SVC     JP2DOS
        END
```

3. MODIII PROGRAM

```
MODIII    PSECT     ;model III print routines
          PUBLIC    PRINT,TRSDOS
VDLINE    EQU       021BH
JP2DOS    EQU       402DH
PRINT     INC       HL
          CALL      VDLINE
          RET
TRSDOS    JP        JP2DOS
          END
```

4. PROGII PROGRAM

```
PROGII    PSECT
          EXTERN    BEGIN
START     JP        BEGIN
          LINK      'MAIN/REL'
          LINK      'MODII/REL'
          END       START
```

5. PROGIII PROGRAM

```
PROGIII   PSECT
          EXTERN    BEGIN
START     JP        BEGIN
          LINK      'MAIN/REL'
          LINK      'MODIII/REL'
          END       START
```

When you have finished inserting all five source files, exit the Editor by typing:

```
Q <ENTER>
```

which returns you to TRSDOS READY.

ASSEMBLING A FILE

You should now have stored five source files:

```
MAIN/SRC
MODII/SRC
MODIII/SRC
PROGII/SRC
PROGIII/SRC
```

To see that they are all on your diskette, check the disk directory by typing DIR <ENTER>.

These files contain three types of instructions:

- Z80 mnemonics (LD, CALL, INC, and RET), which the Assembler converts into Z80 object code. Chapter 9 describes Z80 mnemonics.

- An extended mnemonic (SVC), which the Assembler converts into a group of Z80 instructions. Chapter 10 describes extended mnemonics.

- Directives (PSECT, EXTERN, DEFT, PUBLIC, EQU, LINK and END), which are instructions to the Assembler or the Linker. Chapter 8 describes directives.

To assemble the source files, use the ALDS Assembler (ALASM). In the TRSDOS READY mode, type:

```
ALASM MAIN/SRC MAIN/REL <ENTER>
```

The Assembler processes the source file MAIN/SRC into an object file named MAIN/REL. If it displays any errors, edit or re-insert MAIN/SRC and re-assemble it. (An explanation of the Assembler error messages is in the Error Messages Section of this manual.)

You can assemble the other source files in the same way. Note: You can omit the /SRC and /REL extensions. The Assembler knows to append them:

```
ALASM MODII MODII <ENTER>
ALASM MODIII MODIII <ENTER>
ALASM PROGII PROGII <ENTER>
ALASM PROGIII PROGIII <ENTER>
```

When finished, the Assembler produces these object files:

```
MAIN/REL
MODII/REL
MODIII/REL
PROGII/REL
PROGIII/REL
```

The extension REL means that the files are relocatable. That is, they do not have absolute load and execution addresses. Because of this, they cannot be loaded and executed in their present form.

The Assembler converts them into relocatable rather than absolute files because of the the PSECT directives. See Chapter 8 for more information on the directives. See Chapter 3 for information on operating the Assembler.

LINKING A RELOCATABLE FILE

Two of the relocatable files created by the Assembler are:

PROGII/REL
PROGIII/REL

which consist solely of LINK directives. They are for the ALDS Linker to process. Type:

```
ALLINK PROGII/REL PROGII $=4000 <ENTER>
```

This causes the Linker to:

(1) process the LINK directives, LINKing MAIN/REL and MODII/REL to PROGII/REL.

(2) assign absolute addresses beginning with 4000H to PROGII/REL

(3) save the resulting absolute object code as PROGII.

You can link PROGIII/REL in the same way. (Notice that you can optionally omit the /REL extension, since the Assembler will automatically append it.) Type:

```
ALLINK PROGIII PROGIII $=4000
```

Using the same processes as above, the Linker creates PROGIII, an absolute object file, composed of MAIN/REL and MODIII/REL.

Chapter 5, The Linker, discusses the Linker itself. Chapter 8, Directives, discuss the directives which control the Linker.

EXECUTING A FILE

The Linker created two absolute object files:

```
PROGII
PROGIII
```

which are actually two versions of the same main program. PROGII runs on the Model II; PROGIII is for the Model III.

To run PROGII, simply type (in the TRSDOS READY mode):

```
PROGII <ENTER>
```

TRANSFERRING A FILE

You will, of course, need to transfer PROGIII to the Model III before you can execute it. If you have a Model III and an appropriate modem or cable, you can transfer PROGIII with the ALDS File Transfer System. It will produce a Model III disk file of PROGIII.

Connect the two systems (see Chapter 6, The ALDS File Transfer System for instructions).

Load the ALTRAN program on both the Model II and Model III by typing:

```
ALTRAN <ENTER>
```

After ALTRAN displays its menu, type:

```
1 <ENTER>
PROGIII <ENTER>
```

on the Model II, and:

```
2 <ENTER>
PROGIII/CMD <ENTER>
```

on the Model III.

This transfers PROGIII to the Model III diskette and names it PROGIII/CMD (Model III executable programs must have the /CMD extension.)

ALTRAN re-displays its menu when it has finished the transfer. Press <BREAK> to exit the ALTRAN program and return to TRSDOS READY. You can then execute PROGIII on the Model III in the same way you executed PROGII above. Type:

```
PROGIII <ENTER>
```

DEBUGGING A FILE

You can debug any of the object files with the ALDS Debugger. Type:

```
ALBUG <ENTER>
PROGII <ENTER>
```

The Debugger starts up at the beginning of PROGII. You can use any of the Debugger commands listed in Chapter 4 to debug it.

Chapter 2/ THE ALDS EDITOR (ALEDIT)

The ALDS Editor allows you to enter and edit an assembly language source program. You can save this program on disk as a source file to be assembled into Z80 object code.

This section describes the use of the Editor itself. For information on how to write an assembly language source program, see Section II, "ALDS Assembly Language".

LOADING THE EDITOR

This command, typed in the TRSDOS READY mode:

ALEDIT source filespec

loads the Editor and then loads the specified source filespec into the Editor. The source filespec is optional. For example:

ALEDIT <ENTER>

causes the Editor to load and display its heading:

TRS-80 Text Editor Version v.r.
Copyright (c) 1982 Tandy Corp.

(v.r. is the version and release numbers.)

ALEDIT SORTER <ENTER>

causes the Editor to load, display the above heading, then load a source file named SORTER/SRC.

If the source filespec does not contain an extension, the Editor appends /SRC to it.

The Editor loads into all of the memory above TRSDOS. It reserves approximately the top 40K bytes as an "edit buffer" for inserting your programs. However, if you have also loaded one of these TRSDOS utilities -- DO, HOST, SPOOL, SETCOM, or DEBUG -- the edit buffer will be smaller.

USING THE EDITOR

The following pages define the three modes in which you can use the Editor:

- the insert mode
- the line edit mode
- the command mode

THE INSERT MODE

The I command gets you into the insert mode. Type:

```
I
```

(Do not press <ENTER>.) The Editor clears the screen and positions the cursor at the upper left-hand corner. You can now insert source lines into the edit buffer.

Do not use line numbers. The Assembler will consider them syntax errors.

Each source line may have up to 78 characters. After typing the line, press <ENTER> to insert it. To cancel it and return to the Editor command mode, press <BREAK>. For example:

```
;THIS IS THE FIRST SOURCE LINE <ENTER>
;THIS IS THE SECOND <ENTER>
;THIS IS THE THIRD <BREAK>
```

inserts only the first two lines in the Editor's memory; then returns to the Editor command mode.

While inserting lines, you might find it convenient to use the <TAB> key. The Editor has tabs set every eight columns.

The Editor offers certain control functions for quick insertion. To activate a control function, press the <CTRL> key at the same time you press the function key. For example, pressing these two keys at the same time:

<CTRL> <D>

causes the Editor to insert a semicolon and the current date in the text and then position the cursor on the next line.

<CTRL> <E>

causes the Editor to insert ":", tab to the next tab stop, insert "EQU", and then tab again to the next tab stop.

If the line becomes full while inserting the control function, the Editor stops and awaits the next insert mode instruction.

Table 1 lists all the insert control functions.

Table 2 lists the special control keys available in the insert mode.

NOTE: When the edit buffer is full, it will give you a buffer full message and return to the command mode.

Table 1/ ALEDIT Insert Control Functions

FUNCTION	INSERTS
<CTRL> <D>	;current date <ENTER> (i.e. ;Aug 18, 1981 <ENTER>)
<CTRL> <E>	: <TAB> EQU <TAB>
<CTRL> <G>	<TAB> GLOBAL <TAB>
<CTRL> <J>	<TAB> HEADER <TAB> '
<CTRL> <L>	<TAB> INCLUDE <TAB> '
<CTRL> <N>	; <TAB> ENTRY: <TAB>
<CTRL> <P>	<TAB> PUBLIC <TAB>
<CTRL> <R>	; <TAB> EXIT: <TAB>
<CTRL> <S>	;**********... <ENTER> (semicolon followed by 64 asterisks)
<CTRL> <U>	; <TAB> USES: <TAB>
<CTRL> <X>	<TAB> EXTRN <TAB>
<CTRL> <Y>	displays the tab positions. Nothing is inserted.
<CTRL> <Z>	;----------... <ENTER> (semicolon followed by 64 dashes)

Table 2/ ALEDIT Insert Mode Special Keys

<BACKSPACE>	moves cursor back one space and deletes a character
<ENTER>	ends current line, carriage return, and goes to next line still in "I" mode. Note: <ENTER> inserts a blank line if executed by itself.
<BREAK>	cancels current line, and returns to CMD-mode with the cursor on the next line.
<LEFT ARROW>	erases current line, and still in "I" mode, returns to column 1 on current line.
<TAB>	moves to next tab position on the line. Note: <BACKSPACE> will reverse tab.

THE LINE EDIT MODE

The E command enters the line edit mode for editing characters within the current line. When you enter this mode, the Editor displays the line in reverse video. You can then use any of the edit subcommands listed in Table 2 or the special edit keys listed in Table 3.

For example, assume the cursor is on the following line:

```
;THIS IS THE FIRST LINE
```

To change the word FIRST to THIRD from the command mode, type:

```
E
```

(Do not press <ENTER>.) The Editor will display the line in reverse video. You are now in the line edit mode.

Use the <SPACEBAR> to position the cursor at the F in FIRST and type:

5CTHIRD <ENTER>

This will store the change and return to the Editor command mode.

Table 3/ ALEDIT Line Edit Mode Subcommands

COMMAND	DESCRIPTION
A	Clears all changes and re-enters the edit mode for the current line.
nCstring	Changes the next n characters to the specified string. If n is omitted, only one character is changed. (Press ESC to exit the change early)
nD	Deletes n characters. If n is omitted, one character is deleted.
E	Exits the edit mode and stores changes.
Hstring	Deletes the remaining characters, enters the insert mode and allows you to insert a string.
Istring	Allows you to insert material beginning at the current cursor position on the line. Pressing backspace will delete characters from the line. The line may be up to 78 characters in length.
nKcharacter	Kills all characters preceding the nth occurrence of the character.* If n is omitted, the first occurrence is used. If no match is found, the rest of the line is killed.

L	Moves cursor to beginning of line.
Q	Quits the edit mode, cancelling all changes.
nScharacter	Positions the cursor at the nth occurrence of character.* If no match is found, positions the cursor at the end of the line.
Xstring	Moves the cursor to the end of the line, enters the insert mode, and allows you to insert a string.

* The compare begins on the character following the current cursor position.

Table 4/ ALEDIT Line Edit Mode Special Keys

<SPACEBAR>	Moves cursor one position to the right.
<ESC>	Returns to edit command mode from the I, X, C, or H subcommands.
<TAB>	Moves cursor to next tab position (or the end of the line) while in the I, X, or H subcommand mode.
<left arrow>	Identical to the L subcommand.
<right arrow>	Identical to <SPACEBAR>.
<ENTER>	Identical to the E subcommand.

THE COMMAND MODE

When you first load the Editor, it is in the command mode. While in this mode, you can use any of the special keys listed in Table 5 or the commands listed in Table 6.

All commands except I and E return to the command mode after executing. To return to the command mode from I (insert mode) or E (line edit mode), press <BREAK> or <ENTER> respectively.

When you enter an Editor command, it creates a blank "work line" and points to the line just beneath it. To redisplay the screen after an error message and delete the work line, use the N command.

Sample Use

For an example of using the command mode, use the I command to insert this program:

```
;THIS IS THE FIRST LINE <ENTER>
;THIS IS THE SECOND <ENTER>
;AND HERE IS ANOTHER <ENTER>
;AND ANOTHER <ENTER>
```

Press <BREAK> to return to the command mode. Type:

T

the cursor moves to the top of the text. Type B to move it to the bottom. Press <up arrow> and <down arrow> to move it to specific lines.

Move the cursor to the third line and type:

1

The < appears to the left of the line. This specifies the beginning of a block. Move the cursor to the last line and type:

2

The > appears to the left of the line. This specifies the last line in the block. Move the cursor up to the second line and type:

O

which is the O command. This inserts the block between the first and second line. Move the cursor to the last line and type:

D

the delete command (executes without pressing <ENTER>). The last line is now deleted.

To save this program on disk you can use the W command. Type (it does not matter which line the cursor is positioned at):

W TEST <ENTER>

This saves this program on disk as a file named TEST/SRC. You can exit the Editor by typing:

Q <ENTER>

the quit command.

Q will exit the Editor without writing the text to disk. If you forgot to save the text first, type ALEDIT * <ENTER> to re-enter the Editor. Your text will be retained.

Be sure you use the ALEDIT * command immediately after you exit the Editor. It will not work predictably after you run a command which modifies memory. Also, be sure you type one blank space between ALEDIT and the asterisk (*).

Table 5/ ALEDIT Command Mode Keys

<left arrow>	clears the command and returns to the command mode.
<down arrow>	positions the cursor down one line (ignored if the cursor is not in first column)
<up arrow>	positions the cursor up one line. (ignored if the cursor is not in the first column)
<F1>	positions the cursor to the top of the screen.
<F2>	positions the cursor to the bottom of the screen or to the first line after the last line of text.
	displays the current line sequence number. This number will change as you insert and delete lines.
#line<ENTER>	postions the cursor to the specified line sequence number and moves that line to the top of the screen.
<BREAK>	cancels any command being executed and returns to the command mode.
<ESC>	cancels the current command line if you have not yet pressed <ENTER>.

Table 6/ ALEDIT Editor Commands

Description of terms:

current line
the line where the cursor is currently positioned.

del
(stands for delimeter) One of the following characters which marks the beginning and ending of a string:
! " # $ % & ' ( ) * + , - . / : ; < = > ?

string
one to 37 ASCII characters.

text
the source program or text currently in RAM.

A <ENTER>
Re-executes the last executed command that contains more than one letter.

B
Moves the cursor to the bottom of the text.

C del stringl del string2 del occurrence <ENTER>
Changes stringl to string2 for the number of occurrences you specify. Occurences must range from 1 to 255. The changes begin at the current line and are made only to the first occurrence on a given line.

If you omit occurrence, only the first occurence of stringl is changed. You may specify occurrence with an asterisk, in which case the change is made to the first occurrence of stringl in all the remaining lines.

For example:
C/TEXT/FILE/3 <ENTER>
changes the first 3 occurrences of TEXT to FILE.
C?TEXT?FILE?* <ENTER>
changes all occurrences of TEXT to FILE. (Change acts on only the first occurrence within a line.)

After executing the command, the cursor positions itself at the last change or, at the top of the file if changes went through the whole file.

D
Deletes the current line or block of lines. To delete a block, position the cursor at the first line in the block and type <1>. Then position it at the last line and type the D command. (The block may be on several pages.) The cursor must be positioned on a line within the file.

For example:

```
                    LD   A,B
      <1>           ADD  A,1
                    ADD  A,3
      <D>           ADD  A,4
                    DEC  B
```

deletes all but the following:

```
                    LD   A,B
                    DEC  B
```

You can cancel a block deletion after pressing <1> but before typing D. To do this, press <3>.

E
Allows you to edit the current line using line edit mode subcommands. The line will appear in reverse video. See the edit mode for a listing of subcommands.

F del string del occurrence <ENTER>
Finds the specified occurrence of string. If you omit occurrence, finds the first occurrence of string. If you omit string, the last string specified is found. Occurences must range from 1 to 255. For example:

F/TEXT/2 <ENTER>
finds the second occurrence of TEXT.
F/TEXT/ <ENTER>
finds the next occurrence of TEXT.
F <ENTER>
finds the next occurrence of the last specified string.
F% % <ENTER>
finds the next occurrence of five blank spaces.

The Editor will search for only one occurrence of the string in each line.

G <ENTER>
Deletes all text from the current line to the end. You will first be prompted with:
"Are you sure?"
Type Y <ENTER> to delete; N <ENTER> to cancel.

H <ENTER>
Prints the entire text if entered as the first command or the specified block on the printer. To print a block, move the cursor to the first line of the block and type <1>. Move the cursor to the last line of the block and type <H>. For example:

```
          LD   A,B
<1>       ADD  A,1
          ADD  A,3
<H>       ADD  A,4
          DEC  B
```

prints a block of ADD instructions.

You can cancel a block printing after pressing <1> but before typing H. To do this, press <3>.

Press <HOLD> to stop the printing; press <HOLD> again to continue. Press <BREAK> to terminate printing. If the printer is off-line or goes off-line during printing, some characters may be lost.

I
Enters the insert mode for inserting lines just before the current line. See "Insert Mode" for more information.

J
Displays current size of text and how much memory remains. Memory size does not include a small work area when the buffer is full, but the text size may reflect some of this work area.

K <ENTER>
Deletes ALL text. (Does not delete text from the disk file, only from the edit buffer. Before deleting your text, the Editor will ask you "Are you sure". Type Y <ENTER> to execute the command; N <ENTER> to not execute it.

L filespec $C <ENTER>
Loads filespec into the Editor. $C is optional. If specified, the Editor chains the new filespec to the end of the text currently in memory. If not specified, the new filespec overlays the current text.

For example:
L TEST <ENTER>
loads TEST/SRC into the Editor.
L TEST $C <ENTER>
chains TEST/SRC to the end of the text currently in memory.

The Editor will load either variable length record (VLR) files or fixed length record (FLR) files with a record length of one. If the file is fixed length, each line must be ended with a carriage return.

M
Moves the specified block just ahead of the current line. Use <1> and <2> to specify the block. The Editor displays a line count as it moves each line.

For example:

```
                ADD  A,B
        <1>     PUSH DE
                PUSH HL
                PUSH IY
        <2>     PUSH BC
                LD   A,8
        <M>     ADD  A,10
```

Moves the block of PUSH instructions just ahead of the last line:

```
                ADD  A,B
                LD   A,8
                PUSH DE
                PUSH HL
                PUSH IY
                PUSH BC
                ADD  A,10
```

You can cancel the block after specifying it but before typing M. To do this press <3>.

N
Updates the display. You might want to use this after executing the J command or cancelling the G command.

O
Copies the specified block just above the current line. (Use <1> and <2> to specify a block as described in the M command.

P
Moves the cursor to the next page (which is 24 lines from the top of the screen).

Q <ENTER>
Exits the Editor. If you forgot to save the file first, type ALEDIT * <ENTER> immediately upon exiting the Editor. The Editor will load with your text retained in memory.

R <ENTER>
Deletes the current line and enters the insert mode. Using the J command, if there is 0000 memory left in the buffer, executing the R command will delete the line but will not allow it to be replaced with new text.

S command <ENTER>
Executes a TRSDOS command and returns to the Editor. For example:

S DIR <ENTER>

displays a directory and then returns to the Editor.

You should only use the TRSDOS library commands (not TRSDOS utilities or user programs). Press N to redisplay the editor screen. The program is back in command mode at this point with the blinking cursor as prompt.

T
Moves the cursor to the top of the text.

U
Moves the cursor to the previous page (which is the 24 preceding lines).

V
Scrolls current line to the top of the screen.

W filespec $option1... <ENTER>
Saves all text on disk as filespec. filespec is optional; if omitted, it is the filespec you used to load the file. The Editor appends /SRC to filespec unless it already includes an extension.

If you specify a filespec which is the same name you used to load the file, the Editor saves the old file as a "backup". The backup file will have the extension /BAK. The Editor will create this backup file only once for each load command or initial loading of the editor. On subsequent Write commands, it updates the specified filespec only.

The options are:

N Negates the creation of a backup file. (Useful if the disk is nearly full or if you do not want the previous backup erased.)

E Exits the Editor after saving the file unless there is an error.

L Saves the file as a variable length record (VLR) file with line numbers in this format: line length/ASCII line number/dummy TAB/text. This way, it can be loaded with Radio Shack's Series I Editor Assembler (Catalog Number 26-4713). The ALDS Assembler considers line numbers to be syntax errors.

M Saves the file as a fixed length record (FLR) file with an LRL of 1 in this format:

text/carriage return

This way, you can use ALEDIT to edit a "DO-file" created with the TRSDOS "BUILD" command and save it in a format which can be loaded by the TRSDOS "DO" command. In editing DO-file, do not remove the "P" character at the top left-hand corner of the screen. This character identifies the file as a DO-file.

ML or LM Saves the file as a fixed length record (FLR) file with an LRL of 1 and line numbers in this format:ASCII line number/dummy TAB/text/ carriage return. This way, it can be loaded with Radio Shack's Macro Editor/Assembler (Catalog Number 26-4202).

For example:

W SAMPLE <ENTER>

saves all text as a file named SAMPLE/SRC.

W SAMPLE $NE

saves text as SAMPLE/SRC. The Editor will not create any backup and will exit back to TRSDOS READY after saving the file.

Without using the L or the M options, the Editor saves the file in the format required by the ALDS Assembler:

- Each record is variable length (VLR) as described in the TRSDOS Reference Manual.
- Each character is saved exactly as it appears on the display.
- No carriage returns or end of text code is saved.
- Each line is saved in this format: length/text/

X del stringl del string2 del occurrence

Same as the C command, but prompts before making the change. Occurence must range from 1 from 255.

Y filespec

Same as the L command with one exception. It compresses a file that contains blank space characters into tab characters. Text that does not begin on tab boundaries may be shifted out of alignment.

Chapter 3/ THE ALDS ASSEMBLER (ALASM)

The ALDS Assembler produces Z80 object code. It does this by inputting a source file -- composed of Z80 instructions, assembler language directives, and data -- and assembling it into Z80 code.

In this Section, we'll show how to use the Assembler. For information on the source file, see the sections on the ALDS Editor, Assembler Language Directives, and Z80 Instruction Set.

THE ASSEMBLER COMMAND

This command, typed in the TRSDOS READY mode, loads and executes the Assembler:

ALASM filespec1 filespec2 {switches}

filespec1 is the source file you want assembled. If you do not specify an extension, the Assembler will assign it /SRC. filespec1 must not be read protected. Do not specify a password.

filespec2 is optional. It stores the assembled object code. You can specify filespec2 with an asterisk (*). If so, the Assembler will assign it filespec1's name (less the extension).

If the program is relocatable and filespec2 does not have an extension, the Assembler will assign it /REL. (The Assembler uses the PSECT directive, discussed in Chapter 8, to determine whether the program is absolute or relocatable.)

filespec2 overrides any OBJ directive you have in your program. filespec1 and filespec2 must be in the standard TRSDOS filespec notation.

Examples:

ALASM TEST TEST <ENTER>

assembles TEST/SRC and saves the object code as TEST if the program is absolute or TEST/REL if it's relocatable.

```
ALASM TEST * <ENTER>
```

does the same.

```
ALASM TEST/PAY * <ENTER>
```

assembles TEST/PAY and saves the object code as TEST or TEST/REL.

```
ALASM TEST/PAY FILE/ACC <ENTER>
```

assembles TEST/PAY and saves the object code as FILE/ACC.

```
ALASM TEST <ENTER>
```

assembles TEST/SRC. No object file is produced unless TEST/SRC contains an OBJ directive.

SWITCHES

You may specify one or more switches to create a listing or control the assembly output. If you do not specify filespec2, you must enclose the switches in braces. For example:

```
ALASM TEST * L <ENTER>
```

assembles TEST/SRC into TEST or TEST/REL and displays a listing (L) of the assembly.

```
ALASM TEST * LXP <ENTER>
```

does the same as the above and also creates a cross reference listing (X) and prints it all on the printer (P).

```
ALASM TEST {L} <ENTER>
```

assembles TEST/SRC and creates a listing. Since filespec2 is omitted, the braces are required.

The details of all the available switches are in Table 7:

Table 7/ ALASM Switches

L (Listing)

Generates a complete listing on the video display. Figure 1 shows a sample assembly listing.

The Assembler prints a character to the left of a line number if the line is affected by one of these special conditions:

Character	Condition
-	the symbol in symbol field is never referenced
p	the symbol in symbol field is PUBLIC
g	the symbol in symbol field is GLOBAL
+	a symbol in operand field is defined in global file
x	a symbol in operand field is defined in an external file
r	some or all the object data is relocatable

X (Cross Reference)

Generates an alphabetical cross reference listing of all symbols defined in the program. Figure 2 shows a sample cross reference.

P (Printer)

Outputs the listing on the printer in addition to the video display. Use this option with the L option. You may not use this switch with the Assembler D switch, nor can you use it with the TRSDOS SPOOL command's "capture file" option (the "N" option). Be sure that the printer is on-line.

W (Wait On Errors)

Causes the Assembler to stop the listing at each assembly error. Press <ENTER> to continue the listing.

T (Truncate the Listing)

Truncates the listing output to the printer so that you can use 80 column paper.

Ddrive number (Store Listing on Disk)

Stores the listing in a disk file named filespec1/LST. Use this option with the L option. If the listing will not all fit on the diskette, the Assembler will close the file and prompt you to change diskettes. Do so and press <ENTER>. (Be sure the diskette you remove does not contain the source, object, or ALASM files.)

The Assembler will store the remainder as filespec1/LSU on the newly inserted diskette. If this diskette also becomes full, the listing will go to the next diskette as filespec1/LSV.

The Assembler will repeat this process until it has saved the entire listing. Each time it creates a new listing file, it will increment the third character in the extension:

filespec1/LST, filespec1/LSU, ...filespec1/LSZ,
filespec1/LSA, filespec1/LSB, ...filespec1/LSS

You may optionally omit the drive number. If you do so, the Assembler will output the listing file to the lowest numbered write-enabled drive (usually drive 0) and continue the listing in the next drive. This is not a good method to use, since the Assembler will probably run out of work space before completing the listing.

Files created with the D option should be printed with the PRINT command with FORMS P=66, L=66 set. (If your printer handles form feeds, FORMS P=0, L=0 will also work.)

The D switch overrides the P switch.

G (Go)

Executes the program after assembling it. The program must be absolute and have no errors.

F (Memory image)

Causes the assembled object file to be in memory image form, rather than the TRSDOS program file format. The program must be absolute and have no errors. See the NOLOAD directive in Chapter 8 for more information.

Examples:

```
ALASM SOURCE OBJTST LDX <ENTER>
```

assembles SOURCE/SRC into OBJTST/REL or OBJTST. Displays a listing and a cross reference of this assembly and saves these in one or more files named SOURCE/LST, SOURCE/LSU, SOURCE/LSV, etc.

```
ALASM TEST * G   <ENTER>
```

assembles TEST/SRC into TEST or TEST/REL, then executes the program (unless it is relocatable or has errors).

```
ALASM MOD1 PROG/CMD LPW <ENTER>
```

assembles MOD1/SRC into PROG/CMD. Generates a listing which is printed on the video display and the printer. Each time the Assembler encounters an error, it stops the listing.

```
ALASM XYZ/COD TST/ABC:2 LD3 <ENTER>
```

assembles XYZ/COD and stores it as TST/ABC on the diskette in drive 2. The Assembler generates a listing which it displays and saves as XYZ/LST on the diskette in drive 3.
If the drive 3 diskette becomes full, you will be prompted to insert another diskette which will hold XYZ/LSU, a continuation of the listing.

NOTE: Be sure the CLOCK is not turned on (CLOCK {OFF}) while running the Assembler.

Chapter 4/ THE ALDS DEBUGGER

The ALDS Debugger is an easy-to-use system for debugging absolute object code programs. It includes all the features found on the DEBUG utility program of your TRSDOS disk. In addition, it includes several new, powerful debugging tools.

To allow you the maximum memory space for programs, the Debugger is on your diskette in three modules. These are ALBUG, the program which calls ALBUG/SYS, the main module, and ALBUG/OVL, an overlay which contains certain Debugger commands that load into memory only when necessary.

ALBUG, ALBUG/SYS, and ALBUG/OVL reside in the high memory of your Model II starting at EFØØH.

ALBUGX, ALBUGRES/REL and ALBUG/RES are explained in the addendum.

Among many other features, the ALDS Debugger allows you to:

- set both permanent breakpoints with pass counts and temporary breakpoints (see the J and B commands in Table 8).
- execute one or more instructions at a time (see the I and E commands in Table 8).
- specify a memory address as an offset. This is useful in debugging a program which you assemble in the relocatable mode (see the O command in Table 8).

What you can debug with the ALDS Debugger :

You can debug any absolute program. The program must lie in memory between 28ØØH and EEFFH.

In addition, you can use the Debugger to change the contents of disk files, using the DISK ZAP mode (see the Z command).

The DISK ZAP mode uses memory locations EEØØH through EEFFH as well as the normal Debugger memory. Because of this, you must be sure not to store one of your programs in that part of the memory if you want to use DISK ZAP, or else it may be overwritten.

LOADING THE ALDS DEBUGGER

Turn on the ALDS Debugger by typing (from TRSDOS READY):

ALBUG <ENTER>

The Debugger clears the screen and displays the message:

ALBUG is now ON

and returns control to TRSDOS READY. Now to debug your program, type (if your program uses any parameters, you may include them in the command line):

filespec <ENTER>

The Debugger display appears on your screen with the cursor flashing in the left-center. You are now in the Debugger command mode and can use any of the commands listed in Table 8. The PC contains the entry point of your program.

If you want to enter the Debugger without loading one of your programs (for instance, to enter the DISK ZAP mode), type:

ALBUG ON <ENTER>

from the TRSDOS READY mode, and the Debugger program begins execution.

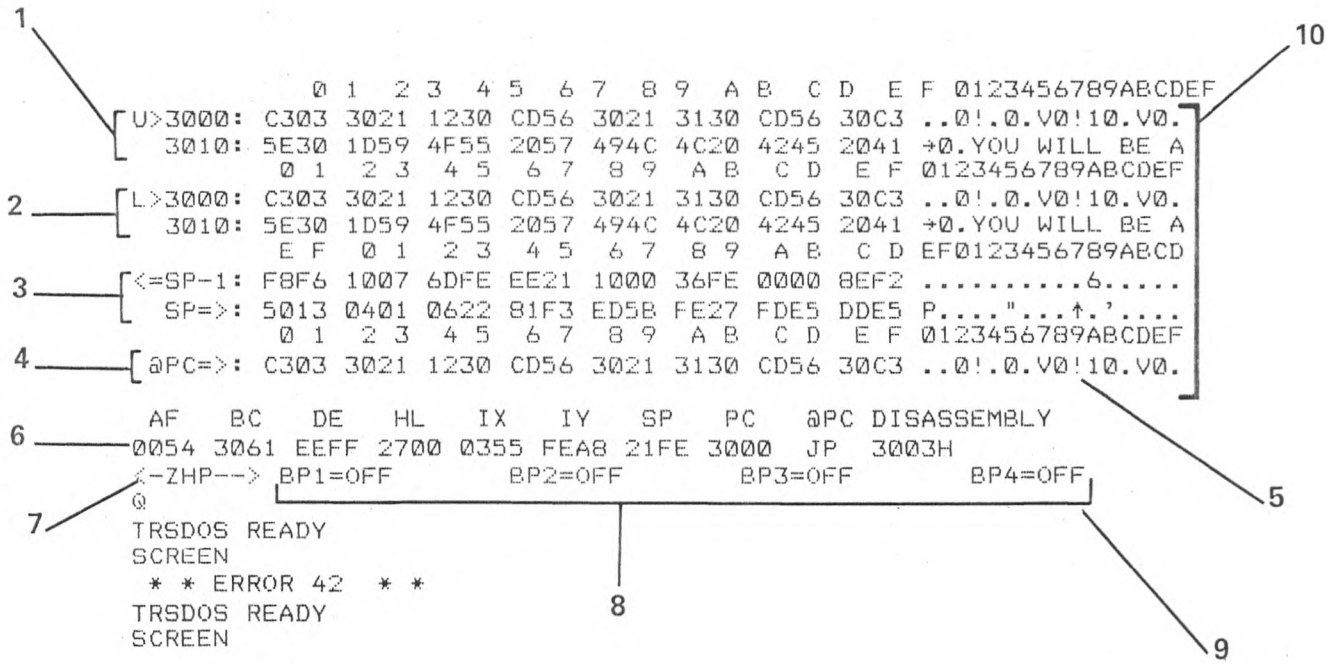
ALBUG remains ON (activated) until either you enter <Q> from the Debugger, or you type:

ALBUG OFF <ENTER>

from the TRSDOS READY mode.

THE DEBUGGER DISPLAY

This is a sample Debugger display.



1. Upper Dump. This is a 32 byte section in the memory. U stands for Upper Dump. The 3ØØØ signifies that the memory address of the first byte in that row is 3ØØØH. To the right of the 3ØØØ are the contents of memory locations 3ØØØH through 3ØØFH. To the right of the 3Ø1Ø are the contents of memory locations 3Ø1ØH through 3Ø1FH. Above these lines are numbers which represent the memory address of the data listed below them. For example, the byte under the 7 and in the row marked as 3ØØØ, is the memory location 3ØØ7H.

2. Lower Dump. This is another 32 byte section of memory. It is arranged exactly like the Upper Dump.

3. The memory location pointed to by one of the register pairs (in this case SP) is displayed here along with the 15 bytes immediately following it. The label of this line is SP=>. Directly above it is a line labeled <SP-1. It contains the 16 bytes preceding the memory location pointed to by the register pair. (The byte on the far left of the line is at the address (SP)-16 and the byte on the far right is at the address (SP)-1.)

4. The memory location pointed to by the PC is on this line, marked @PC=>. It is followed by the contents of the next 15 bytes in memory. Above this line is the memory location of the respective bytes.

5. This line (here shown blank) displays certain information such as base register addresses and math function results. When you enter a new command, it is erased.

6. These lines show the contents of the Z8Ø registers. At the right side of the lower line, below " @PC DISASSEMBLY ", is a Z8Ø instruction. The address pointed to by the PC contains this instruction in machine code, and the Debugger has disassembled it into an assembly level instruction. The Debugger uses as many bytes following the PC address as necessary to make a complete instruction. This means that what is disassembled can be one to four bytes long.

7. <SZHPNC> are the condition codes set in the F register. The codes are:

S	sign
Z	zero
H	half carry
P	parity
N	BCD condition
C	carry

When a condition bit is set (i.e. when it is equal to 1) the Debugger encloses the letter within the < > characters. Otherwise it simply displays a hyphen (-). For example, <-Z---C) shows that the zero (Z) and the carry (C) bits have been set and all other bits have not.

8. This area lists the status of the permanent breakpoints. BP1=28ØØ/ØØØC translates as breakpoint 1 set at 28ØØH with the pass counter set at 12 decimal passes. BP2=OFF means that breakpoint 2 is not set. (See Table 7 for more information).

9. When you first enter the Debugger, this line gives version and copyright information. Thereafter, it displays commands and prompts used in debugging code.

1Ø. This area displays the ASCII value of any hexadecimal data to its left. If the hex number has no printable value, a period (.) is displayed.

ENTERING COMMANDS

Table 7 lists all the Debugger commands. You can execute most of them by simply pressing the appropriate letter. By pressing <BREAK>, you can abort any command in the middle of execution and return to the command level.

Most commands prompt you to specify a register or data (the prompt is in area 9 of figure 1). The prompts use these abbreviations:

Adr	Address
Asc	ASCII
BP1	Breakpoint 1
CHR	Character
C(lr)	Clear
DEC	Decimal
<E>	<ENTER>
Eadr	End address
H(ex)	Hexadecimal
Pas	Pass counter
Reg	Register
SAdr	Start address
Str	String

The commands usually prompt you for a certain number of parameters. If you fail to provide enough parameters, or if you use an invalid number as a parameter (e.g. hex when a decimal number is expected), you receive the message:

Invalid Parameter

and the Debugger returns to the command mode.

SPECIFYING REGISTERS

Certain commands require you to input a register or register pair. For example, the Debugger might prompt you with:

C,E,L,A(F),B(C),D(E),H(L),(I)X,(I)Y,S(P) or P(C)

To enter a single register, you simply press the appropriate letter. To enter a register pair, you must press the letter NOT shown in parenthesis. For example, <C> enters the C register, but <B> enters the BC register pair.

SPECIFYING DATA

As a Constant

Some commands require constants. When entering a hexadecimal constant, you must follow it with H. For example, "1Ø" indicates the decimal number 1Ø, while "1ØH" stands for the hexadecimal number 1Ø (the decimal number 16).

As an Address

Other commands require addresses. These must be in hexadecimal. There is no need to follow hex addresses with an H.

You can specify an address by referring to a register pair which contains that address. For example, if BC contains the number 6ØØØH, you can enter $B instead of the address 6ØØØH. The register abbreviations for this type of addressing are:

AF	$A
BC	$B
DE	$D
HL	$H
IX	$X
IY	$Y
SP	$S
PC	$P

You can also specify any address as an "offset" to a base register. This is useful if you assemble the program in the relocatable mode. It allows you to use a relocatable location to specify an address. (See the O command in Table 8).

The ALDS Debugger is for debugging your own code. Hence, you cannot enter an address which is in the system memory (i.e., below 27ØØH). In addition, the Debugger protects itself by not allowing you to interfere with the memory above EFØØH. If you enter an invalid address the Debugger returns to the command mode.

BREAKPOINTS

The Debugger allows you to set "breakpoints" within your code. Breakpoints are commands causing the execution of your program to stop at a given point. There are two types of breakpoints, temporary and permanent.

You can assign temporary breakpoints with the J command (Jump to an address and execute). They apply only to this one execution of J. With them you can execute a short section of code, or determine which way control goes at a branch statement. (See the J command in Table 8).

With the B (Breakpoint) command, you can set permanent breakpoints. They remain in your program until you leave the Debugger or clear them. Permanent breakpoints may have a pass count associated with them.

You must be cautious when setting breakpoints. Set them only at the first byte of an instruction. If you are writing a self-modifying code where the first byte of an instruction may change during the course of running the program, be careful not to place a breakpoint at that instruction.

Table 8/ Debugger Commands

n; (semicolon)
Advances the memory location of the Upper Dump. The default advance is 16 bytes. You can precede the semicolon with n, a decimal number, which changes the default to n bytes, until you press <BREAK>, when the default returns to 16 bytes.

n: (colon)
Advances the memory location of the Lower Dump. The default advance is 16 bytes. You can precede the colon with n, a decimal number, which changes the default to n bytes, until you press <BREAK>, when the default returns to 16 bytes.

n= (equal sign)
Decrements the memory location of the Upper Dump. The default decrement is 16 bytes. You can precede the equal sign with n, a decimal number, which changes the default to n bytes, until you press <BREAK>, when the default returns to 16 bytes.

n+ (plus sign)
Decrements the memory location of the Lower Dump. The default decrement is 16 bytes. You can precede the plus sign with n, a decimal number, which changes the default to n-bytes, until you press <BREAK>, when the default returns to 16 bytes.

B
Sets or clears permanent breakpoints and their pass counters. After you press <B>, a prompt appears:

```
1,2,3,4 or C(lr)?
```

You can now choose to set or alter any of the four breakpoints, or clear all four. To set breakpoint 1, for example, press <1>. The Debugger prompts with:

```
Ø <E> or [Adr][,P(as)]<E>?
```

You can now select the address where you want the breakpoint. You must set it at the first byte of an instruction. You can not place a breakpoint on top of an existing breakpoint.

Each permanent breakpoint is associated with a pass counter. Pass counters are useful to stop execution after an instruction has been executed a given number of times. A pass count is specified by following the breakpoint address with a with a comma and then the pass count value.

To set the breakpoint at 3ØØØH, with a pass of 12 type:

```
3ØØØ,12<ENTER>
```

You can clear the breakpoint by entering a value of Ø:

```
Ø <ENTER>
```

To clear all four of the breakpoints, press <C> in response to the first prompt. The Debugger asks you:

```
Are You Sure (Y/N)?
```

to allow you to change your command (the Debugger accepts only Y or N). The status of all four breakpoints is displayed in area 8 of figure 1.

When you set each breakpoint, the Debugger saves the contents of the breakpoint address, and replaces it with an RST 3Ø instruction which assembles into ØF7H. Now, in typing Y to remove the breakpoints, the Debugger restores the memory addresses to their original contents.

The contents of the pass counter can be updated without respecifying the address of the breakpoint. For example, if you had previously set a permanent breakpoint at 3ØØØH, you can update the pass count to 24 by typing:

```
,24 <ENTER>
```

in response to:

```
Ø<E> or [Adr][,P(as)]<E>?
```

Whenever ALBUG executes a program instruction which is associated with a permanent breakpoint with a nonzero pass count, the count is decremented and execution resumes. Execution halts when a permanent breakpoint with a pass count of zero is reached. ALBUG is designed so that once execution is halted by reaching a pass count of zero, you may single step over a permanent breakpoint.

The permanent breakpoint remains in the program until it is explicitly cleared with the <B> command or until ALBUG is exited with the <Q> command. Note: if a return to DOS is executed in your program, the premanent breakpoints remain intact and ALBUG can be re-entered by typing ALBUG.

Typing ALBUG OFF will remove all breakpoints (restoring your original code) and deactivate ALBUG. If you load another program, and then clear breakpoints with ALBUG OFF or the breakpoint clear command, the original code at the breakpoints from the first program will be restored. This is probably not what was intended. Turn ALBUG OFF or clear permanent breakpoints before loading another program.

ALBUG uses RST 3Ø instructions to handle all breakpoint processing. If ALBUG encounters a RST 3Ø instruction which you placed in your program, execution will halt. To resume execution, the program counter must be reset using the <R> command.

C
Copies one section of memory to another. After you press <C>, a prompt appears:

```
Start Adr,End Adr,To Adr <E> ?
```

Type the appropriate start, ending, and destination addresses. For example, type:

```
28ØØ,282F,3ØØØ <ENTER>
```

to copy the data contained in addresses 28ØØH-282FH to addresses 3ØØØH-3Ø2FH.

D
Dumps the data contained in the address pointed to by a register pair to the Debugger display. (See area 4 in figure 1). The data on either side of this address is also displayed. After you press <D>, the Debugger displays:

```
Reg Dump  B(C),D(E),H(L),(I)X,(I)Y,S(P) or P(C)?
```

To see the data referenced by the IX register pair, respond with:

```
<X>
```

The screen updates to display the new dump.

nE
This command is identical to the <I> command with one exception: If the current instruction is a call the debugger executes the entire routine.

nF
Searches for a string within a given range in the memory. After you press <F>, a prompt appears:

Sadr,Eadr <E> or <E>?

After you enter a valid start and end address, the Debugger asks you:

H(ex) or A(scii)?

Depending on whether you enter <H> or <A>, the Debugger then prompts you with:

Hex Str <E>?
or
Asc Str <E>?

When you enter the appropriate type string the Debugger searches through the given memory for it. If the Debugger finds a matching string, the Lower Dump is set to display this part of memory. If no match is found, the Debugger returns to the command level.

To find the next occurence of the string, you need only to press <F> from the command level and respond to the prompt with <ENTER>. You can continue to search for matching strings until you reach the ending address (EAdr) or until there are no more string matches in the specified range.

To specify which occurrence of the string you want to find, precede the F command with n, a decimal number between 1 and 255. For example, to find the fifth occurrence of 1FH, start by entering 5F.

The F command will find an ASCII string of up to 24 characters or a HEX string of up to 12 digits.

You may also specify a new range for the current string. Enter the new range, abort the <F> command with the <BREAK> key at the 'H(EX) or A(SCII)? ' prompt, and press 'F <ENTER>'.

G

Examines a 256 byte area in memory. After you press <G>, a prompt appears:

U(pper) or L(ower)?

Depending on your answer, the Debugger displays the 256 byte multiple of memory which contains the address of the Upper or Lower Dump. For example, if the Upper Dump starts at 28Ø7H and you press <G> and then <U>, your screen changes so that it now contains a dump of memory starting with 28ØØH.

The <n;>, <n:>, <n=>, and <n+> commands may be used in this display mode as they were in the partial screen display mode, except that the value of n is always rounded up to a multiple of 256.

Press <BREAK> to return to the regular Debugger display.

nI

The <I> and <E> commands are ALBUG's single step instructions. The <I> command executes the current instruction in your program (the instruction pointed to by the PC register.) ALBUG then increments the PC register to the next instructions address and returns to the command mode.

By preceding <I> with n, a decimal number, you can indicate the number of times it is to be repeated. For example, if you type:

1ØI

the <I> command is executed 1Ø times.

There are a couple of considerations you should be aware of when single stepping. ALBUG will not place a breakpoint in a protected area. This implies that an attempt to single step an instruction in a protected area will cause a jump to that instruction. Single stepping a call to a protected area will cause the entire call to be executed at full speed. These precautions are necessary since many of the system calls such as video and disk I/O will work properly only when executed at full speed.

J
Executes a specific section of your program. After you press <J>, a prompt appears:

```
J [ADR][,BP1][,BP2][,BP3][,BP4]<E>?
```

The start address (ADR) is optional. If you omit it, the execution begins at the contents of the PC. BP1-BP4 are temporary breakpoints and are also optional. You can include any or all of them.

The first temporary breakpoint encountered causes the execution to terminate. This clears all temporary breakpoints. The execution also terminates if a permanent breakpoint with a pass of zero is encountered.

For example, suppose you want to execute the instructions between 28ØØH and 282lH, inclusively. After pressing <J>, you would type:

```
28ØØ,2821 <ENTER>
```

Temporary breakpoints are often useful near branch points. If you set breakpoints at the possible jump locations, you can see which way

your program goes. For example, say you have a set of conditional jumps which could go to 3Ø4ØH, 3Ø8ØH or 3ØFØH. When you enter:

```
28ØØ,3Ø4Ø,3Ø8Ø,3ØFØ <ENTER>
```

your program will begin at 28ØØ, and terminate after jumping. You can then examine the PC to see which breakpoint caused the execution to stop (i.e., which way the jump went).

K

Allows you to convert between decimal, hex, and ASCII characters. With this command, you can also perform addition and subtraction. After you press K, a prompt appears:

```
Enter value or equation ?
```

You can then enter a value or equation. For example, to find out the ASCII character for 32H, type:

```
32H <ENTER>
```

The display (in area 5 of figure 1) is then:

```
HEX String = ØØ32 DEC String = ØØØ5Ø CHR String= ".2"
```

To do addition or subtraction, simply type in the equation. You can mix decimal, hex, or character constants in the equation. Only single characters are allowed, and unprintable characters are output as periods (.); all characters must be preceded by a quote mark ("). For example, if you type this equation:

```
1124-4ØH+"Z <ENTER>
```

the Debugger displays:

```
HEX String = Ø47E DEC String = Ø115Ø  CHR String = ".."
```

the result must lie between Ø and FFFFH, or else the number is represented modulo FFFFH. For example, -1H is represented as FFFFH, and 1ØØØ1H as 1H.

L
Loads a given range of memory with a constant value. After you press <L>, a prompt appears:

```
SAdr,EAdr,Value <E> ?
```

When you enter a start address, end address, and value, the area in memory is filled inclusively with the value. For example:

```
6ØØØ,6FFF,FFH<ENTER>
```

fills addresses 6ØØØH to 6FFFH with FFH.

```
6ØØØ,6FFF,16<ENTER>
```

fills addresses 6ØØØH to 6FFFH with 1ØH (the hexadecimal equivalent of decimal 16).

M
Changes values in user memory. After you press <M>, a prompt appears:

```
Address =?
```

Enter a hexidecimal address and press <ENTER>. The Debugger then displays a 256 byte block of memory and puts the cursor on the specified memory location. The numbers along the left-hand side are the memory addresses for the first byte in their respective lines.

You may reposition the cursor with the up, down, left, and right arrow keys when entering data. Press <ENTER> to return to the debugger display.

N
Toggles the Debugger display between the primed and unprimed register set.

O
Sets values for offset base registers. You can use these offset registers for debugging a program you assembled in the relocatable mode. When you press <O> a prompt appears:

1,2,3,4,5,6,7,8 or <E>?

If you press <ENTER> the Debugger displays the values of the base registers in area 5 of the screen (see figure 1). There are eight offset base registers. They supply the "base" or start address of the program or O module.

After you set an offset address, you can specify an address as a relocatable location, followed by a colon, followed by the number of the offset register. (Your Assembler listing gives the relocatable locations of each instruction.

For example, if an instruction in the assembly listing is at relocatable ØØ1A, and you linked the program using an absolute start address of 6ØØØH, press <1> in response to the above prompt, and you will receive:

Base Adr <E>?

type:

6ØØØ <ENTER>
This sets base register 1 to 6ØØØH. Then, an address 1AH bytes after the beginning of 6ØØØH can be entered as 1A:1.

P
Prints what is currently displayed on your screen. If your printer is not ready, the Debugger displays the error number.

Q
Exits the Debugger and returns to the TRSDOS READY mode. All existing breakpoints are cleared. The Debugger is turned off.

R
Alters the contents of any of the registers. When you press <R>, a prompt appears:

```
C,E,L,A(F),B(C),D(E),H(L),(I)X,(I)Y,S(P) or P(C)?
```

After you press the appropriate letter, the Debugger prompts you for a value to put in the register. For example, if you are changing the C register, a prompt appears:

```
(C =## or # <E>) C = ?
```

To change the register to FFH, type:

```
FF <ENTER>
```

The screen is updated and the C register now contains FFH.

You can also change register pairs. For example, if you were changing the contents of the HL register pair to AØ64H, after you press <R>, respond to the register prompt by pressing <H>. You are then prompted with:

```
(HL =#### or ### or H =## or # <E>) HL = ?
```

To complete the change, simply type:

```
AØ64 <ENTER>
```

If you are changing a register pair and you input only 3 digits, the Debugger assumes leading zeros.

By using the N command first, you may alter the contents of the prime register set.

The stack pointer and the program counter may not be changed to point at the protected areas. Keep in mind when changing the stack pointer that ALBUG uses the stack. To be safe allow for a stack size of 256 bytes.

S

Executes a TRSDOS system command. Enter the system command after the S. For example:

S DIR <ENTER>

returns the directory of drive Ø, and then prompts you with:

<ENTER> to continue

V

Changes the start address of the Upper or Lower Dump. When you press <V>, a prompt appears:

(U)pper or (L)ower?

Depending on which you press, <U> or <L>, you will be prompted with either:

U Address =?

or

L Address =?

For example, to change the start address of the Upper Dump to 6ØØØH, respond to "U Address =?" with:

6ØØØ <ENTER>

Z

Enters the DISK ZAP mode, allowing you to debug disk files. See the explanation following.

THE DISK ZAP MODE

The DISK ZAP mode allows you to change the contents of your fixed-length record disk files. When you press the <Z> command, the screen clears, and you are prompted:

```
ALDS DISK ZAP Utility

Enter Filespec Name?
```

After you enter the filespec, DISK ZAP asks you whether you want to see your file based on relative sectors or records:

```
Enter S(ector) or R(ecord) ?
```

SECTOR MODE

If you enter S, you are then asked to enter the relative sector number:

```
Enter Relative Sector Number (# <E> or <E>) ?
```

You can specify a sector number, or just press <ENTER>. If you press <ENTER>, the DISK ZAP displays the first disk sector containing your file (relative sector Ø).

The display for the sector is similar to what the M (Modify memory) command displays. Except that the relative sector and starting byte numbers are listed along the left side. For example, the number ØØ11ØØ refers to sector 11 and byte ØØ.

You can move from sector to sector by using the semicolon (;) which advances the display to the next sector. The equal sign (=) decrements the display to the previous sector. If you cross a file boundary (that is, if you go to a sector not used by your file), you will return to the DISK ZAP, filespec prompt.

You can modify the data in your file much like you modify memory. When you press <M>, the Debugger puts the cursor onto the first byte of the sector. You can then position the cursor to the correct byte with the up, down, left, and right arrows. After you have completed your change, press <ENTER> to write the change to disk. If you don't want the change written, press the <BREAK> key.

RECORD MODE

If you respond to the Record/Sector prompt with <R>, DISK ZAP prompts you to:

Enter Record Number (# <E> or <E>) ?

You can default to the beginning record (record Ø) by pressing <ENTER>, or you can choose a specific record in your file. DISK ZAP displays only the record you specify. If the record length is less than or equal to 16, one line is shown and the irrelevent bytes are filled with zeros. If the record length is between 17 and 32 two lines are displayed, between 33 and 48 three lines, and so on.

If you want to display the previous record, press the equal sign (=). The next record of a file is displayed when you press the semicolon (;). If you go beyond the last record, or go before the first record (record Ø), the Debugger returns to the DISK ZAP filespec prompt.

Bytes of the record are modified just like they are in the Sector Mode. After you press <M>, position the cursor and make the change.

Technical Note: Decimal numbers in ALBUG are treated modulo 65536 (Ex. A number entered as 65537 will be treated by ALBUG as 1). TRSDOS 2.Øa files may contain 65536 records. Thus, the largest file with LRL=1 is 256 sectors. Specifying a sector number greater than the maximum size allowable for that logical record length will give a displacement into that file based on modulo 65536 records.

TRSDOS-II files may be larger than 65536 records, but DISK ZAP cannot access beyond 65536 records.

DISK ZAP ERRORS

If you get an error message while using DISK ZAP, it is a TRSDOS error message. See your TRSDOS Owner's Manual for an explanation.

LEAVING THE DISK ZAP MODE

Pressing <BREAK> at the DISK ZAP filespec prompt returns you to the Debugger. Pressing <BREAK> from any other level of DISK ZAP returns you to the original DISK ZAP filespec prompt.

Note: DISK ZAP uses memory starting at EEØØH rather than EFØØH, so any data you have stored there will be over-written.

Chapter 5/ THE ALDS LINKER (ALLINK)

The ALDS Linker converts a relocatable object file into absolute object code.

Unlike many linkers, ALDS Linker receives its commands through directives in your program. You can use these directives to get the Linker to link in external program sections and use external symbols. The Linker directives are:

PSECT	- begins a program section and determines its mode (absolute or relocatable)
PUBLIC	- declares symbol definitions PUBLIC so that other program sections can use them
EXTERN	- brings in external symbols
GLOBAL	- creates a global symbol file
GLINK	- brings in global symbols
LINK	- links an external absolute or relocatable program section

For information and examples on how to write a relocatable program containing Linker directives, see Chapter 8.

THE LINKER COMMAND

This command, typed in the TRSDOS READY mode, loads and executes the Linker:

ALLINK filespec1 filespec2 {options}

filespec1 is the relocatable file you want converted. If you do not specify an extension, the Linker will assign it /REL.

filespec2 is optional. If specified, it stores the converted absolute object file. If not, the Linker will still process the file so that you can test for undefined symbols, missing files, or generate a listing.

On the Models I and III, filespec2 must have the extension /CMD to load and execute). You can use an asterisk (*) to specify filespec2. If so, the Linker assigns it filespec1's name less the extension.

You can specify one or more of these options, separated by a blank space:

$=nnnn specifies the absolute hexadecimal start address of the program. If omitted the start address is 3ØØØH (Model II) or 52ØØH (Model I/III).

MAP prints each PSECT name, its absolute start address, and the start, end, and transfer address of the program.

SYM prints the absolute address of each PUBLIC and GLOBAL symbol, sorted alphabetically by symbol. You cannot use this option with the XREF option

XREF prints an alphabetical cross-reference of each PUBLIC and GLOBAL symbol, its absolute address, and all addresses which reference it. This option overrides the SYM option, if both are specified

DISK saves the listing requested by the MAP, SYM, or XREF options on disk. The resulting disk file will have the same name as filespecl with the extension /MAP.

PRT directs the listing requested by the MAP, SYM, or XREF options to the printer. The Linker ignores this option if your printer is not connected or ready for printing.

Examples:

```
ALLINK PROG/REL PROG $=7ØØØ MAP SYM DISK
```

assigns absolute addresses beginning with 7ØØØH to PROG/REL and stores the resulting file as PROG. The Linker displays a PSECT MAP, and a table of absolute symbol definitions and stores this listing in a file named PROG/MAP

```
ALLINK PROG DONE
```

assigns absolute addresses beginning with 3ØØØH (Model II) or 52ØØH (Models I or III) to PROG/REL and stores the resulting file as DONE.

```
ALLINK PROG *
```

assigns absolute addresses beginning with 3ØØØH or 52ØØH to PROG/REL and stores the resulting file as PROG.

TECHNICAL INFORMATION

OPERATION

The Linker processes the file in two passes. In pass 1, the Linker:

- processes any LINK directives by linking in the specified program sections.
- assigns the file absolute addresses. It does this by offsetting the relocatable locations (assigned by the Assembler) to the absolute start address.
- processes any LINK directives by linking in the specified program sections (PSECTs). If the PSECT to be LINKed is relocatable, the Linker assigns it addresses which immediately follow the last relocatable PSECT. If it is absolute, the Linker will assign it the same addresses the Assembler assigned it.
- processes any PUBLIC or GLOBAL directives by inserting the declared symbols and their corresponding definitions in a Linker symbol table.
- processes any GLINK directives by inputting the specified global file's symbols into the Linker symbol table.

In pass 2 the Linker:

- fills in the addresses of any EXTERNal symbols, and generates error messages for all undefined symbols.
- if filespec2 is specified, saves the resulting absolute file.
- processes any GLOBAL directives, by creating a global file.

MAXIMUM SIZES:

The Linker will link up to 2ØØ external program sections (PSECTs).

The Linker Symbol Table will hold at least 2,ØØØ external symbols. If the symbols are smaller than the maximum size of ten characters the Table will hold more.

The maximum absolute object file which the Linker creates can be as large as TRSDOS will load. See your TRSDOS manual.

Chapter 6/ ALDS FILE TRANSFER SYSTEM (ALTRAN)

The ALTRAN program transfers files created under the ALDS package between any two TRS-8Øs (Model I, II, or III) by either hardwire or modem. It transmits or receives object code, source code or data files.

Since ALTRAN was developed specifically for files created with the ALDS package, we cannot guarantee that it will accurately transfer files created with other software.

SET-UP

You can use two types of connections in ALTRAN: modem or hardwire.

Modem

The standard RS-232-C Interface is appropriate if you plan to transfer files via a modem. You can use any TRS-8Ø modem provided that both ends can use the same baud rate and can communicate with each other (i.e. both can't be originate only or answer only modems).

See your Radio Shack modem operation manual for installation instru tions.

On the Model II, ALTRAN automatically disables the channel you're using, then resets it with it's own parameters. Certain "smart" modems automatically hang-up if the channel it's on is disabled. To avoid this hang-up, start ALTRAN first, then turn on the modem. ALTRAN has no effect on the channel not being used.

Hardwire

If you plan to hardwire computers, you will need:

Model I/III to Model I/III	26-14Ø8 RS-232-C Cable 26-1496 Adapter Box 26-1497 12" Extension Cable
Model II to/from Model I/III	26-1496 Adapter Box 26-14Ø8 RS-232-C Cable

To hardwire a Model I to a Model II:

1. Connect cable from the Model I RS-232-C Interface to the female plug of the 26-1496 Adapter Box, and connect the other end to the Model I Expansion Interface.
2. Directly connect the male plug of the Adapter Box to Serial Channel B on your Model II.

To hardwire a Model I to a Model III:

1. Connect cable from the Model I RS-232-C Interface to the female plug of the 26-1496 Adapter Box, and connect the other end to the Model I Expansion Interface.
2. Attach the other end of the Adapter Box to the female plug of the 26-1497 12" Extension Cable.
3. Connect the male plug of the Extension Cable directly to the RS-232 port of the Model III.

To hardwire a Model II to a Model III:

1. Connect the male plug of the Adapter Box to Serial Channel B of the Model II.
2. Attach the other end of the Adapter Box to the RS-232-C Interface of the Model III.

BAUD RATE

The factory sets the baud rate at 3ØØ for all ALTRAN packages. As a general rule with most systems, the quality of transmissions is directly proportional to the ratio of distance versus baud rate. In other words, the higher the baud rate, the shorter the distance allowed.

If you want to change the factory-set baud rate, you can use the PATCH utility. The patch for the Model I and III is:

```
PATCH ALTRAN/CMD (ADD=52ØØ,FIND=55,CHG=nn)
```

where nn is the value in Table 9.

The patch for the Model II is:

```
PATCH ALTRAN {A=3ØØØ,F=Ø3,C=Øn}
```

where Øn is the value in Table 9.

Model I customers can set the baud rate in one of two ways:

1. Use the patch indicated above.

2. Perform the patch: PATCH ALTRAN/CMD (ADD=52ØØ, FIND=55,CHG=FF), and adjust the Dip Switches on the RS-232-C Interface board as listed in Table 3. Note that only switches S4, S6, S7, and S8 are used and that, like the Model II and III, you can't change the other parameters. These settings are for ALTRAN and do not necessarily match switch settings for any other terminal programs. 'FF' is a signal to the program that the desired baud rate is not software overwritten and must be patched back if you want to use the software option mentioned above.

(ALTRAN will function properly only under Model I TRSDOS 2.3B -- not Model I TRSDOS 2.3. Appendix E describes the differences between the two versions of TRSDOS.)

Table 9/ BAUD RATE CHANGE TABLE

Baud Rate Desired	Model III Patch	Model II Patch	Model I Patch	Model I Switches S4	S6	S7	S8
75	11	N/A	11	O	O	O	C
11Ø	22	Ø1	22	O	C	O	O
134.5*	N/A	N/A	33	O	C	O	C
15Ø	44	Ø2	44	O	O	C	O
3ØØ	55	Ø3	55	O	O	C	C
6ØØ	66	Ø4	66	O	C	C	O
12ØØ	77	Ø5	77	O	C	C	C
18ØØ	88	N/A	88	C	O	O	O
2ØØØ	N/A	N/A	99	C	O	O	O
24ØØ	AA	Ø6	AA	C	C	O	O
36ØØ	BB	N/A	BB	C	C	O	C
48ØØ	CC	Ø7	CC	C	O	C	O
72ØØ	DD	N/A	DD	C	O	C	C
96ØØ	EE	N/A	EE	C	C	C	O

*listed in the MENU as 134
N/A = not available
C = Closed O = Open

The Model II package uses channel B for operations. To use channel A, perform the PATCH:

```
PATCH ALTRAN {A=3ØØ1,F=42,C=41}
```

If you want to change both the baud rate and the channel on the Model II, you may wish to use this patch:

```
PATCH ALTRAN {A=3ØØØ,F=Ø342,C=nn41}
```

where nn is the appropriate value from the change table above (Ø1 to Ø7).

The following table shows the recommended maximum distance (hardwired) versus baud rate for high quality transmissions. The factors that govern this table are for worse case non-modem situations.

Note: all values are approximate

Baud Rate	Maximum Model I/III Distance	Maximum Model II Distance
75 - 3ØØ	5ØØ feet	5ØØ feet
6ØØ - 12ØØ	5Ø feet	75 feet
18ØØ - 36ØØ	25 feet	35 feet
48ØØ +	1Ø feet	2Ø feet

LOADING ALTRAN

To load ALTRAN from TRSDOS READY, type:

```
ALTRAN <ENTER>
```

The program immediately displays the menu of operations and the settings of the RS-232-C parameter list (including the channel being used on the Model II).

Figure 1 shows the menu of a Model II. The menus for Model I and Model III vary slightly in format. The operations and numbers are the same.

Channel 'B', 3ØØ baud, 8 data bits, no parity, 1 stop bit

1 - Transmit OBJECT file
2 - Receive OBJECT file

3 - Transmit SOURCE file
4 - Receive SOURCE file

5 - Transmit DATA file
6 - Receive DATA file

7 - Transmit via COMMAND file
8 - Receive via received COMMAND file or WILDCARD mask

9 - Enter 'Mini-Terminal' Mode

W - Transmit via WILDCARD mask

Figure 1. THE ALTRAN MENU

Operations 1, 3, 5, 7 and W are the transmission modes. The one you select depends on the type of file you want to transfer.

Operations 2, 4, 6 and 8 are the receiving modes. Again, the one you select depends on the type of file you'll be receiving.

You can use operation 9, 'Mini-Terminal', for terminal to terminal communications.

See COMMAND FILE for instructions on creating a command file.

The W operation is only available on Model II ALTRAN. See "WILDCARD MASK" later in this section.

OPERATION

Once you load ALTRAN, as a final test to ensure both transmitting and receiving stations are operational, send a test message via Operation 9 - 'Mini-Terminal' mode in both directions. ALTRAN must be able to communicate in both directions to function properly.

Beginning the Transmission

1. Determine the type of file you want to transfer.

   Use operations 1 and 2 (OBJECT file) for:

   - ALDS object files (both executable and relocatable)

   Use operations 3 and 4 (SOURCE file) for:

   - ALDS source files
   - Series I Editor/Assembler source files (the file transfer system will write the file to the receiving station in ALDS source file format).

   Use operations 5 and 6 (DATA file) for:

   - fixed length record files with an LRL of other than 256 (assembler global files, application program data files, assembler listing files, and non-ALDS source files such as BASIC.)

     Please note some non-ALDS Model I and Model III files, with an EOF byte which is not zero (as displayed in the directory) may not transfer properly. This is because ALTRAN will change the EOF byte to zero, thereby changing the length of the file.

   - Model II variable length files. (You cannot transfer a variable length file to a Model I or III, since these two systems do not support variable length files.)

   Note: When transferring files from one model to another, you must consider the differences between systems. It is unlikely that the same object file can run on all three models due to the difference in ROM and RAM addresses, etc. In addition, we can't guarantee successful transfer of file formats not used by ALDS, even though some files may transfer.

2. Select an operation.

   The number of the operation you choose depends on the type of file you want to transfer and whether you're the transmitting or receiving station. If you are the transmitting station and plan to send an OBJECT file, type 1 <ENTER> in response to the Which? prompt. The receiving station enters a 2 in answer to the Which? prompt. (The order in which the stations enter their operations doesn't affect the transfer, i.e. the receiving station can specify operation 2 before the transmitting station specifies operation 1.)

3. Specify a file.

   After each station selects an operation, ALTRAN prompts for a filespec with File Name?.....

   Both stations should enter the name of the file. Be sure to include the extension and drive number (if not the system drive).

   If you choose Operation 2 and are using a Model I or III, be sure to specify a file which has the extension /CMD. Model I and III object program files must have this extension.

   If you choose Operation 7 or W, ALTRAN prompts the the transmitting station with File Name? (See COMMAND FILE later in this section on how to create one.)

   Using Operation 8, ALTRAN prompts the receiving station with Drive Number?. To avoid the possibility of accidently writing over a file, the receiving station should use a blank diskette. Remember, the Model I and III diskettes do not hold as much data as the Model II.

4. Specify a record length (Model I data transfers only)

   In operation 5, if you are the transmitting station on a Model I, ALTRAN prompts you with "File's LRL?" Your response must be the LRL set when the file was created. To find this information, use the DIRectory command under TRSDOS .

During the Transmission

When operation actually begins, the transmitting station immediately sends the first block of the file. During transmission, the display reads:

'Transmitting Block 1 '

As each block is sent, it increments the block number by one. (Depending on the baud rate and LRL, this increment may take from a fraction of a second to about a minute.) This message is not displayed if you are transferring a null file (EOF and no other information).

At the same time, the message:

'Receiving Block 1'

appears on the receiving station's video display. This indicates that the station is ready to receive the first block of the file, and is not necessarily receiving it. After each file is received, the block number displayed is one more than what was actually received.

This message may not come on immediately in operations 5 and 6 because the transmitting station must first send the file type and the logical record length of the file before the receiving station can be readied to receive the first block of the file.

After receiving each block, ALTRAN increments the block number by one, then stores that block to disk under the filespec named in step 3.

If the receiving station is not ready, the transmitting station keeps trying to transmit a block until it receives an acknowledgement or until the <BREAK> key is pressed.

Once transmission actually takes place, the receiving station expects a block until it receives an EOF marker or until the <BREAK> key is pressed. If the <BREAK> key is pressed during transmission of a file, the file won't be valid or useable.

On SOURCE file transfer only, prior to transmission, ALTRAN at the transmitting station checks the first line of the file for an existent line number. If there is none, it automatically adds line numbers to the entire file before sending the file.

The receiving station strips the bytes corresponding to the line number from all lines of the transferred file as it stores them.

In the 'Mini-Terminal' mode, you can transmit any character except <ESC> (<up arrow> on the Model I/III) and <BREAK> and the receiving station will output the character to the screen. However, not all of the TRS-8Ø models (at the receiving station) interpret the characters in the same way. One model may interpret the control characters differently and display a character other than what was transmitted. On other models, certain characters may activate features such as dual routing, reverse video, or 4Ø-character mode. And, the Models I and III won't output tabs.

Ending the Transmission

After all transmissions are complete for operations 1 through 8, ALTRAN returns to the menu, unless you are sending a wildcard or a command file ending with operation 9.

To escape from the menu, type Q <ENTER>. To exit the 'Mini-Terminal' mode, press <ESC> on the Model II and <up arrow> on the Model I/III. The COMM drivers are left turned on. You may wish to turn them off with SETCOM after a series of transfers.

If you want to transfer another file, return to Step 2.

When an Error Occurs

If an error occurs at one station (not including 'Unknown or unuseable baud rate was patched' which automatically returns to TRSDOS READY), ALTRAN will cease transmission, close the file, return a descriptive error message (including error number if a TRSDOS error on the Models II or III), and display the following:

```
Further transmission not possible
Press <ENTER> to go into Mini-Terminal mode
Press <ENTER> <ESCAPE>* to return to menu
Press <BREAK> to exit to TRSDOS READY

* <up arrow> on Model I/III
```

When an error occurs, the computer making the error will send a cancellation message (<CTRL><X> or 18H) which the other computer will display minus the descriptive error message.

Under certain circumstances, such as transmitting or receiving the LRL, a byte of data, or the checksum, this feature is disabled so that a legitimate 18H won't cause a cancellation and an error message won't be displayed. Therefore, if your computer remains idle for a period of time (the length depending upon your baud rate), you can assume an error has occurred. Press <BREAK> to return to TRSDOS READY.

Note: It is always a good idea for both stations to arrange to go to 'Mini-Terminal' mode if an error occurs. Because the station not causing the error isn't always informed of an error, you should return to 'Mini-Terminal' mode if your computer locks up for an unusual length of time.

COMMAND FILE

A command file is an automatic input file. This file executes a series of operations with one command. By building a command file, you will be able to transmit several files with this one command.

You must enter the Editor to create a command file. The procedure is:

1) Load the Editor
2) Enter the Insert Mode
3) Enter the filespec you are sending
4) Tab over one position and enter the operation code number used for transmitting the file (1, 3, or 5)
5) Repeat steps 3 and 4 until all files are entered.
6) If you want to invoke Mini-Terminal mode, enter it last. A dummy filespec must precede it.
7) Exit insertion mode
8) Write the command file to disk. Do NOT use the line numbers option.

Example

At TRSDOS READY, type:

```
ALEDIT <ENTER>
```

to enter the Screen Editor. Then type I to enter the insertion mode.

In the insertion mode, type:

```
FILE1/SRC      3  <ENTER>
FILE2/OBJ      1  <ENTER>
FILE3/DAT      5  <ENTER>
DUMMY          9  <ENTER>
```

to create the command file.

When run, this command file transmits three files in a row with one input command. It transmits the first filespec, FILE1/SRC, as a source file, the second, FILE2/OBJ, as an object file, and the third, FILE3/DAT, as a data file. The last file, DUMMY, isn't transmitted. It invokes the 'Mini-Terminal' mode.

If for some reason you don't have ALEDIT, you may download the Command File from another computer, using the SOURCE File Transfer.

USING A WILDCARD MASK (Model II only)

The wildcard mask allows you to specify a collection of files by using a "wildcard" field in place of the filenames and/or extensions. For a complete definition of a wildcard and how to use it, see your Model II Owner's Manual.

The wildcard feature only scans the first 96 directory entries. Under TRSDOS-II, disks may have more than 96 files. Files beyond the 96th directory entry will not be transmitted. (Use Function 1, 3 or 5 for thes files.)

Example

You can use a wildcard to transfer related files with the same extension. For example, if you want to transfer all of the object files on a certain disk with the extension /PAY, you can use the W command. All the files you transfer must have this extension (i.e., ACCT/PAY:1, MERCH/PAY:1, etc.), be on the same drive, and be the same type (i.e., all are object files, source files or data files). To transmit the files, select the W option. When the computer prompts you for the wildcard mask, type:

```
*/PAY:1 <ENTER>
```

The computer then prompts you for the function number. Type 1 for Object File, 3 for Source File or 5 for Data File.

As it transfers each, the computer displays the name of the file. When the transfer is complete, the computer returns to the menu.

Remember when writing your mask:

1. Drive number must be specified (even if drive Ø).
2. No imbedded blanks are allowed.
3. All files must be the same type.
4. * means zero or more characters. It doesn't matter what they are. Beware of specifying files you didn't want to transfer.
5. Do not specify passwords in this mask. The files must not be read protected.

TECHNICAL INFORMATION:

DEFINITIONS:

ACK = Acknowledgement of receipt of correct block or inquiry and request to transmit next block. (code Ø6H)

NAK = Acknowledgement of receipt of incorrect block and request for retransmission. (code 15H)

WAK = Acknowledgement of receipt of correct block, but wait before transmitting next block (so the computer may write out block). (code 1BH)

EOT = End of transmission of this file. (code Ø4H)

ENQ = Enquire for a ready to receive. (code Ø5H)

ETX = End of text. (code Ø3H)

CAN = Cancellation (aborts current transfer) (code 18H)

ALGORITHMS

OBJECT FILES

ALTRAN transmits and receives OBJECT files as 256 byte, fixed length record (FLR) blocks.

It uses this algorithm to transmit OBJECT files:

1 open file for read
2 read a sector into a buffer
  if end of file, send EOT, receive ACK, and return to menu
3 display xmit block number
4 send ENQ
5 receive ACK
6 output sector
7 output checksum

```
8     receive ACK or NAK or WAK
          repeat block if NAK
          if WAK, wait for ACK
9     goto 2, "read a sector"
```

It uses this algorithm to receive OBJECT files:

```
1     open file for write
2     display received block number
3     receive ENQ
          if EOT, send ACK, close file and exit
4     send ACK
5     receive sector
6     receive checksum
7     output ACK,NAK,WAK
          repeat receive if NAK
8     send WAK
9     write sector
1Ø    send ACK
11    goto 2, "display block number"
```

SOURCE FILE

ALTRAN transmits SOURCE files as variable length records (VLR) in the Model II, or fixed length records (FLR) 256 on Models I/III.

It uses the following algorithm to transmit the SOURCE file:

```
1     open file for read
2     read in a line (if MOD I/III strip bit 7 from
          line numbers).  if a line number is not
          present on the first byte of the line,
          add a line number.  Be sure the source
          does not have numbers in column 1.  They
          may be accidentally deleted.  If end of
          file, send EOT and receive ACK.
3     display xmit block number
4     send ENQ
5     receive ACK
6     send line length
7     output the line
8     output the checksum
9     receive ACK, or NAK, or WAK
          repeat line if NAK
          if WAK, wait for ACK
1Ø    goto 2, "read in a line"
```

It uses this algorithm to receive the SOURCE file.

```
1    open file for write
2    display receive block number
3    receive ENQ
          if EOT, send ACK, close file and exit
4    send ACK
5    receive line length
6    receive the line
7    receive the checksum
8    send ACK, or NAK, or WAK
          repeat receive if NAK
          send WAK
          write the line, without the line number
9    goto 2, "display block number"
```

DATA FILE

The ALTRAN program sends DATA files as variable length records (VLR) or as fixed length records (FLR), depending on whatever created the file, on the Model II or as fixed length records (FLR) on the Models I/III.

It uses the following algorithm to transmit the DATA file:

```
1    (if Model I, ask for LRL)
2    open file for read
3    send file type (V/F) and file's LRL
4    read in one record of data
          if end of file, send EOT, receive ACK,
          and exit.
5    display xmit block number
6    send ENQ
7    receive ACK
8    send data record length
9    send data
1Ø   send checksum
11   receive ACK or NAK or WAK
          repeat xmit if NAK
          if WAK, wait for ACK
12   goto 4, "read in one record"
```

It uses this algorithm to receive the DATA file:

```
1    receive file type (V/F) and file's LRL
2    open file for write with those parameters
3    display receive block number
4    receive ENQ
          if EOT, send ACK, close the file and
          exit
5    send ACK
6    receive data record length
7    receive data
8    receive checksum
9    send ACK or NAK or WAK
          repeat receive of NAK
          send WAK, write data record
1Ø   send ACK
11   goto 3, "display block number"
```

INDIRECT COMMAND FILE

ALTRAN uses this algorithm to transmit the COMMAND file:

```
1    open IND file for read
2    build a text line
          if end of file, send ETX, wait for ACK,
          and return to menu.
3    send ENQ
4    receive ACK
5    send file name and function
6    send checksum
7    receive ACK or NAK or WAK
          if NAK, goto send ENQ
8    display file name
9    transmit file through functions 1, 3, or 5
1Ø   goto 2, "build a text line"
```

It uses this algorithm to receive the COMMAND file:

```
1    receive ENQ or ETX
          if ETX, send ACK and return to main menu
2    send ACK
3    receive file name and function
4    receive checksum
5    send ACK or NAK or WAK
          if NAK, goto receive ENQ or ETX
6    display file name
7    receive file through functions 2, 4, or 6
8    goto 1, "receive ENQ or ETX"
```

WILDCARD MASK (Model II only)

ALTRAN uses this algorithm to transmit a wildcard mask:

1 check for a drive number, if none, then error
2 get the directory of the drive
3 check each filespec against the wildcard mask
4 store each filespec that matches the mask
  into an internal command file
5 after all filespecs are checked, goto
  appropriate transmitting mode

MINI-TERMINAL MODE

ALTRAN uses the following algorithm to transmit and receive keyboard characters:

1 scan keyboard for character
  if escape character, exit mini-terminal mode
  if character, then display and output to RS-232-C
2 scan RS-232-C input
  if character, then display
3 goto 1, "scan keyboard"

BUILDING AN ADAPTER CONNECTION

If you want to, you have the option to build your own adapter connection instead of buying a Radio Shack Adapter Box (Catalog Number 26-1496).

Required Materials

Model I	RS-232-C Interface Board RS-232-C Cable DB-25 Male Connector (2)
Model II	DB-25 Connector (2) Cable Wire (minimum of 9 wires)
Model III	RS-232-C Interface Board RS-232-C Cable DB-25 Male Connector (2)

When hardwiring for Model I/III to Model I/III, the pin connections are as shown below:

Figure 2. / Model I/III Pin Connections

For Model I/III to Model II, the pin connections are as shown:

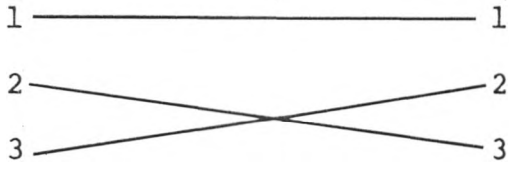
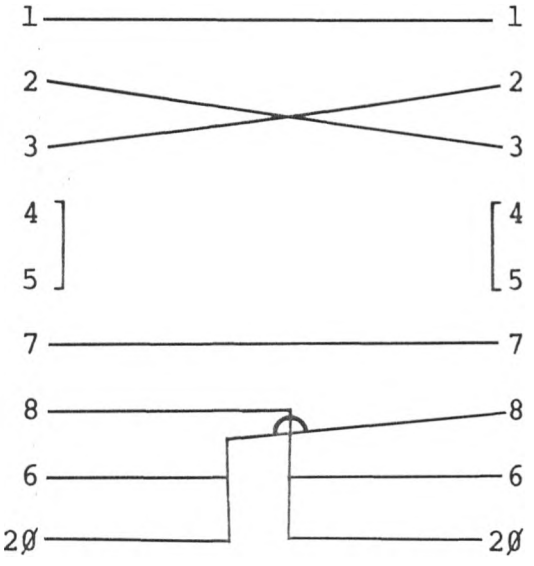


Figure 3. / Model II Pin Connections

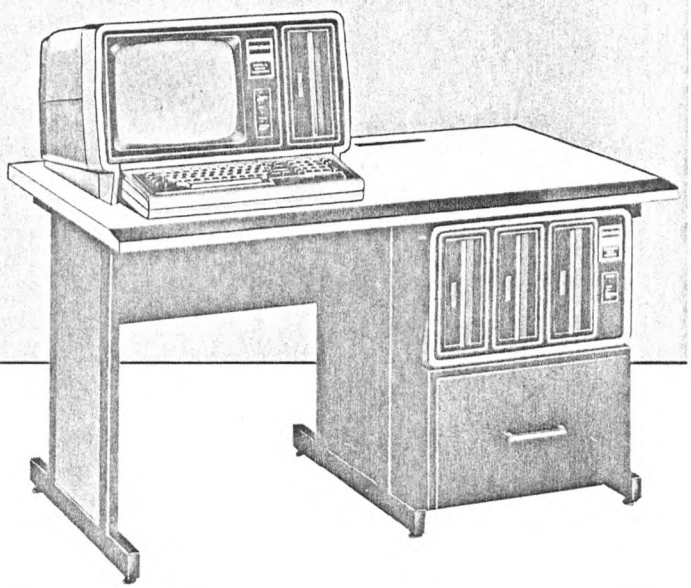


(Note: This cable works from any model to any other model. You can eliminate the connections involving pins 4, 5, 6, 7, 8, and 2Ø on the Model I/III end if you wish.)

Radio Shack®

Section 2
ALDS Assembly Language

Syntax, Directives,
Z80 Mnemonics, Extended Z80 Mnemonics



CUSTOM MANUFACTURED IN THE U.S.A. BY RADIO SHACK, A DIVISION OF TANDY CORPORATION

SECTION II

ALDS ASSEMBLY LANGUAGE

Chapter 7/ ALDS ASSEMBLY LANGUAGE SYNTAX

This chapter describes how the ALDS Assembler interprets source lines. The next chapters list all the instructions available with ALDS.

An ALDS assembly language source line can contain up to four fields. They are:

- the label
- the instruction
- the operands
- the comment

THE LABEL

The label is optional. It is a symbol which defines the location of the instruction immediately following it. For example:

```
NAME      LD   A,5
```

NAME is a symbol used as a label. The Assembler uses it to store the location of the LD A,5 instruction. For example, if LD A,5 is at location 3ØØØH, the Assembler assigns the value 3ØØØH to NAME and stores this in the symbol table.

The label must begin in column one (the first character in the line) or be followed by a colon. For example, this line produces a syntax error:

```
 NAME     LD   A,5
```

since the label NAME is not in column one.

However, this is acceptable:

```
 NAME:    LD   A,5
```

since NAME is followed by a colon.

VALID SYMBOLS

A symbol can consist of up to ten of the following characters:

alpha characters
: (A-Z) in either upper or lower case (the Assembler treats upper and lower case letters differently. "NAME", for example, is a different symbol than "Name").

numeric characters
: (Ø-9) (the symbol cannot begin with a number).

special characters
: the underscore (_)
: the question mark (?)
: the dollar sign ($)
: the @ character

It may not contain a space character. These are examples of valid symbols:

Date? $B__? AlD2 B2345678

The following are reserved words. You cannot use them as ordinary symbols, since this conflicts with the way the Assembler notes register names, branch conditions, or the location counter value:

$	A	B	C	D	E	H	L	F	Z	P
M	I	R	V	AF	BC	DE	HL	SP	IX	IY
XH	XL	YH	YL	NC	NZ	PE	PO	NV		

Reserved words are reserved in both upper and lower case. For example, SP, sp, Sp, and sP are all reserved.)

THE INSTRUCTION

The instruction is usually required. It can be either:

a Z8Ø mnemonic
: (Chapter 9), which is an instruction to the microprocessor. The Assembler converts into a Z8Ø operation code.

an assembler directive
(Chapter 8), which is an instruction to the Assembler itself.

an extended Z8Ø mnemonic
(Chapter 1Ø), which the Assembler expands into a group of Z8Ø mnemonics.

a macro call
(Chapter 8), which the Assembler expands into one or more of the above types of instructions.

You can begin the instruction anywhere but in column one. If the line contains a symbol, there must be at least one space, tab, or colon between the instruction and the symbol.

For example, the Assembler interprets LDIR as an instruction in all of these lines:

```
SYMBOL LDIR
SYMBOL    LDIR
 LDIR
          LDIR
```

However, in these two lines:

```
SYMBOLLDIR
LDIR
```

the Assembler interprets LDIR as part of the symbol field.

You can use either upper or lower case to indicate the instruction. For example, you can indicate the LDIR instruction as:

```
ldir
```

Of course, in the case of a macro call, you must be careful that you use the same case that you used when you defined the macro.

THE OPERANDS

Many instructions allow you to specify data as operands. Some instructions allow you to use a register name or a flag as an operand. Some allow you to indicate a specific value.

You must use at least one space or tab to separate the operands from the instruction. In these examples, A and 3 are operands:

```
SYMBOL LD A,3
 LD           A,3
 LD   A,3
```

However, this line produces an error:

```
SYMBOL LDA,3
```

since there is no space between the instruction and the operands.

EXPRESSIONS

When specifying a certain value as an operand (such as "3" in the above example), you must use a valid assembler expression. The expression can consist of one or more terms connected by operators.

Terms

A term can be:

a number
: The Assembler assumes the number is decimal (base 1Ø) unless you use a base suffix or the RADIX directive. Changing number bases is described in the next chapter.

an ASCII character
: You must enclose the character in single quotes. The Assembler will assemble it into its ASCII code.

a symbol
: The Assembler fills in its value using the symbol table.

$ (the dollar sign character)
The Assembler interprets this character as the location counter's current value.

For example, each of these are valid terms:

152

which represents the decimal number 152 (unless you have used the RADIX directive described in the next chapter).

'A'

which represents the ASCII character code of decimal 65 or hexadecimal 41.

SYMBOL

which represents the value of SYMBOL.

$

which represents the current value of the Assembler's location counter.

OPERATORS

The operators and their functions are listed on Table 1Ø. If an asterisk (*) follows the function, the operator is unary (acts on one operand). Otherwise it is binary (acts on two operands).

Table 1Ø/ OPERATORS

OPERATOR	FUNCTION	PRIORITY
+	unary plus*	1
-	unary minus*	1
.NOT.	logical not*	1
.HIGH.or.MSB.	high order byte*	1
.LOW.or.LSB.	low order byte*	1
.BIT.	bit* (one shifted n bits to the left)	1

** or ^	exponentiation	2
*	multiplication	3
/	integer division	3
.MOD.	modulo	3
.SHR.	logical shift right	3
.SHL.	logical shift left	3
.RR.	logical rotate right	3
.RL.	logical rotate left	3
+	addition	4
-	subtraction	4
.AND.	logical and	5
.OR.	logical or	6
.XOR.	logical exclusive or	6
.ABS.	absolute value*	7
.EQ. or =	equals	7
.GT. or >	greater than	7
.GE.	greater than or equal to	7
.LT. or <	less than	7
.LE.	less than or equal to	7
.RES.	result* (ignore overflow)	7
.SGN.	sign*	7
.UGT.	unsigned greater than	7
.UGE.	unsigned greater than or equal to	7
.ULT.	unsigned less than	7
.ULE.	unsigned less than or equal to	7

Examples:

```
4321H.SHL.3
```

returns the number 4321H shifted three bits to the left.

```
4321H.SHL.1
```

returns the number 4321H shifted one bit to the left.

```
.RES.(7FFF*7FFF)
```

multiplies 7FFFH by 7FFFH and returns the result. (The RES. operator causes the Assembler to ignore the overflow error this operation would normally cause.)

```
.SGN.SYMBOL
```

returns a -1 if SYMBOL is negative, Ø if it's zero, or 1 if it's positive.

Priority of Operators

When you use multiple operators, the Assembler evaluates them using the priority number indicated. If two operators have the same priority, the Assembler evaluates them from left to right.

You can use parentheses to change the priority of operators.

Examples:

```
4+4/2
```

The division is performed first. (Division is priority 3; addition is priority 4.)

```
(4+4)/2
```

The addition is performed first.

```
4*4/2
```

The multiplication is performed first.

Note: You must use parentheses to separate two operators which are both enclosed in periods. For example:

```
LD HL,5.AND..ABS.-4      is illegal
LD HL,5.AND.(.ABS.-4)    is valid
```

Using Relocatable or External Symbols in Complex Expressions

When using complex expressions, i.e., expressions using more than one term, you need to be careful about using symbols which are:

- external (defined in an external program section), or
- relocatable (defined in a relocatable program section).

Table XX shows which types of complex expressions allow relocatable or external symbols, and the type of value which the Assembler will return. If the expression is not on this table, you cannot use a relocatable or external symbol. Under no conditions can you use relocatable and external symbols within an absolute program.

TABLE 11/ Complex Expressions Allowing Relocatable or External Symbols

Definition of Terms:

ABS is an absolute constant, symbol or expression
EXT is an external symbol or expression
REL is a relocatable symbol or expression
ALL is any of the above

Complex Expression	Resulting Type
EXT+ABS	EXT
ABS+EXT	EXT
EXT-ABS	EXT
REL+ABS	REL
ABS+REL	REL
REL-REL	ABS
REL-ABS	REL
ALL.EQ.ALL**	ABS
REL.GE.REL	ABS
REL.GT.REL	ABS
REL.LT.REL	ABS
REL.LE.REL	ABS

REL.UGE.REL	ABS
REL.ULE.REL	ABS
REL.UGT.REL	ABS
REL.ULT.REL	ABS
.HIGH.REL	*
.MSB.REL	*
.LOW.REL	*
.LSB.REL	*
.HIGH.EXT	*
.MSB.EXT	*
.LSB.EXT	*
.LOW.EXT	*

* these expressions cannot be used as a term in a larger expression. Also, they must be used only where an 8-bit quantity is expected.

** the terms must be of the same type (absolute, external, or relocatable) in order to be equal. Two externals are never equal, including the special case of comparing an external to itself.

Other Special Conditions
Regarding Relocatable or External Expressions

These are some additional considerations you need to be aware of when using relocatable or external expressions:

- If you attempt to fit a relocatable or external value outside of the range of -256 to 255 into an 8-bit field, you will not get an error message. The Assembler will store the low order byte into this field. (Absolute values outside this range generate an error message.)

- You can use the .HIGH., MSB., .LOW., or .LSB. operators only where an 8-bit value is expected. If you use one of these operators where a 16-bit value is expected, the Assembler will either give you an error message or unpredictable results.

. If you use the .HIGH. or .MSB. operator, the Assembler saves the entire value in the object code so it can properly compute the carry into the high order byte (which might result from adding the load address to the expression value during linking)

THE COMMENT

The comment is an optional way to document your program. The Assembler ignores it.

To insert a comment at the end of a line, you must precede it with a semicolon. For example, all of these lines contain comments:

```
NAME      LD   A,3;This is a comment
 LDIR;AND SO IS THIS
;and here is another comment
          LD   A,3  ;and another
```

The Assembler ignores every character following the semicolon. However, this line produces a syntax error:

```
NAME      LD   A,3  This is an illegal comment
```

since there is no semicolon preceding the comment.

Another way to insert a comment is by typing an asterisk (*) in column one. The Assembler ignores all lines which follow until it encounters another * in column one.

For example:

```
     LD   A,3
*This begins a comment section
which the Assembler will ignore.
comment, comment
comment, comment
This is the last line in the comment section
*
     ADD  B
```

the Assembler ignores all lines between LD A,3 and ADD B.

Chapter 8/ ASSEMBLER DIRECTIVES

Assembler directives are commands to the Assembler or, in a few cases, the Linker. They are not instructions to the Z-8Ø Microprocessor and are not a part of your executable program. Generally, you can type them in the same form as the Z8Ø mnemonics and insert them throughout the program.

This chapter contains two parts. Part A is a tutorial. It describes the different types of directives -- what their purpose is and how they inter-relate to each other in the program.

Part B is a reference. It contains an alphabetical listing of each directive. Each listing gives the syntax, a definition, and an example use.

INTRODUCTION TO ASSEMBLER DIRECTIVES

ALDS assembler directives allow you to:

- Change Number Bases
- Define Symbols
- Define Data
- Define Storage
- Initialize the Location Counter
- Manipulate the Location Counter
- Terminate or Hold the Assembly
- Use External Symbols
- Create Index Sections
- Define Macros
- Create a Conditional Section
- Control the Assembly Listing

CHANGING NUMBER BASES

The Assembler recognizes number bases 2 (binary), 8 (octal), 1Ø (decimal) and 16 (hexadecimal). The default is base 1Ø.

You can change the default with the RADIX instruction. For example:

```
RADIX     8
```

tells the Assembler to evaluate all subsequent numbers as base 8.

Using a base suffix identifies a base for a particular number. The base suffixes are:

H	Hexadecimal
d	Decimal
b	Binary
Q or O	Octal

For example, in this instruction:

```
LD        A,33H
```

the 33 is evaluated as a hexadecimal number, regardless of which default base you are in.

You can use upper case "d" and "b" suffixes. Be careful with this, though, since the hexadecimal base interprets "D" and "B" as numbers. For example, in base 16, "1b" is a binary 1; "1B" is hexadecimal 1B.

DEFINING SYMBOLS

Defining symbols allows you to refer to data or memory addresses symbolically. This makes the program easier to read and revise.

ALDS allows you to use a symbol to label the location of any Z8Ø instruction and most directives. It also contains these directives which define symbols:

- EQU - equates a symbol to a constant value
- DEFL - defines a symbol to a variable value

For example:

```
NUMBER    EQU       12          ;EQUates NUMBER to 12
LOOP      LD        A,NUMBER    ;loads A with 12
            .
            .
          LD        HL,LOOP     ;loads HL with LOOP
```

This program uses NUMBER and LOOP as symbols. The first line EQUates NUMBER to 12. The next line uses NUMBER as an operand.

LOOP will define the location of LD A,NUMBER. The last line uses LOOP to specify this location.

DEFINING DATA

Data definition directives insert data into RAM. ALDS contains these data definition directives:

- DEFM - defines string data
- DEFE - defines "encrypted data"
- DEFT - defines data and includes a length byte
- DEFB - defines a byte
- DEFW - defines a word
- DEFR - defines a Roman Numeral
- DATE - defines the current date
- TIME - defines the current time

For example:

```
            LD      HL,TABLE
            LD      B,27
            SVC     9       ;print TABLE on video screen
              .
              .
TABLE       DEFM    'THIS BEGINS A TABLE OF DATA'
```

DEFM inserts the ASCII codes for THIS BEGINS A TABLE OF DATA in the next 27 locations. The symbol TABLE defines the beginning of this location.

Note: SVC is an ALDS extended Z8Ø mnemonic (see Chapter 1Ø). It expands into instructions which call the Model II VDLINE Supervisor Routine:

```
            LD      A,9
            RST     8
```

DEFINING STORAGE

Defining storage reserves an area of RAM which you can use for such functions as inputting and outputting data. ALDS contains these storage definition directives:

- DEFS - reserves RAM
- FILL - sets the "fill mode" so that DEFS will fill the reserved area with zeroes
- NOFILL - ends the fill mode

For example:

```
                LD          HL,BUFFER
                LD          B,2Ø
                SVC         5       ;keyboard input into
                                    ;BUFFER area
                 .
                 .
                FILL
     BUFFER     DEFS        2Ø      ;reserves the next 2Ø bytes
                NOFILL
```

FILL sets the fill mode. DEFS reserves the next 2Ø locations for storage and fills them with zeroes.

(Here, SVC 5 expands into the instructions which call the Model II KBLINE routine.)

INITIALIZING THE LOCATION COUNTER

The Assembler contains a "location counter" which it uses to:

- assign locations to each executable instruction, and
- define the symbols which identify these locations

The locations it assigns are either absolute or relocatable depending on how you initialize the counter.

INITIALIZING THE LOCATION COUNTER TO AN ABSOLUTE LOCATION

To initialize an absolute location, you must use PSECT:

```
     START      PSECT       4ØØØH
4ØØØ  NUM       LD          A,5     ;begin assembling at 4ØØØH
4ØØ2            PUSH        A
4ØØ3            LD          A,6
.                .
.                .
                END         NUM
```

This program section initializes the counter to an absolute 4ØØØH. The Assembler then assigns all the instructions absolute locations, beginning with 4ØØØH.

The Assembler saves this assembly on disk as an "absolute object file". You can load it in the TRSDOS READY mode simply by typing the filespec followed by <ENTER>. Each instruction will load into the same (or "absolute") memory location the Assembler assigned it.

Many other assemblers, such as the Series I, use ORG rather than PSECT to accomplish the same task. If you want to assemble such a program with ALDS, you need to change the first ORG to PSECT. (See Appendix C for more details on converting Series I programs.)

INITIALIZING THE LOCATION COUNTER TO A RELOCATABLE LOCATION

PSECT without an argument initializes the location counter to a relocatable ØØØØ (the ' signs indicates that the locations are relocatable, rather than absolute):

```
                PSECT
ØØØØ'  NUM      LD          A,5    ;begin assembling at
                                   ;relocatable zero
ØØØ2'           PUSH        A
ØØØ3'           LD          A,6
.                .
.                .
                END         NUM
```

The Assembler saves this assembly on disk as a relocatable, rather than absolute, file. You cannot load a relocatable file. You need to use the Linker to convert it into an absolute file.

For example, if the name of the assembled relocatable file is PROG/REL, this Linker command:

```
ALLINK PROG PROG $=4ØØØ <ENTER>
```

assigns absolute locations beginning with 4ØØØH to all the instructions in PROG/REL. It does this by adding 4ØØØH to each relocatable location. The resulting program is saved as an absolute file named PROG.

MANIPULATING THE LOCATION COUNTER

There are several instructions which manipulate the counter within a program section. They are:

- . ORG - changes the value of the counter
- . LITORG - changes the value of the counter and allows room for literal operands
- . SETLOC - manipulates the counter for symbols only
- . RESLOC - ends the SETLOC manipulation

For example:

```
                PSECT     3ØØØH
3ØØØ BEGIN      LD        A,5    ;begin assembling at 3ØØØH
3ØØ2            LD        B,2
     SECOND     ORG       4ØØØH
4ØØØ            LD        HL,ADD  ;increment counter to 4ØØØH
4ØØ2            PUSH      AF
                .
                .
                END       BEGIN
```

This program section initializes the counter to an absolute 3ØØØH. The Assembler begins assigning consecutive absolute addresses until it reaches ORG, which changes the value of the counter to 4ØØØH. The Assembler assigns 4ØØØH to the next instruction and continues again sequentially.

Since the above program is absolute, ORG's parameter sets an absolute location of 4ØØØH.

In the relocatable mode, ORG's parameter sets a relocatable location of 4ØØØH. This means that when you link the program, 4ØØØH serves as an offset to the program's absolute start address.

For example, assume you assemble the same program in the relocatable mode. The Assembler assigns it these locations:

```
                PSECT
ØØØØ' BEGIN     LD        A,5   ;begin assembling at
                                  ;relocatable zero
ØØØ2'           LD        B,2
     SECOND     ORG       4ØØØH
4ØØØ'           LD        HL,ADD  ;increment counter to
                                    ;relocatable 4ØØØH
4ØØ2'           PUSH      AF
                .
                .
                END       BEGIN
```

Now assume you link the relocatable file to the absolute start address of 5ØØØH. The Linker assigns it these addresses:

```
                PSECT
5ØØØ BEGIN      LD        A,5    ;begin assembling at
                                   ;relocatable zero
5ØØ2            LD        B,2
     SECOND     ORG       4ØØØH
9ØØØ            LD        HL,ADD    ;increment to
                                      ;relocatable 4ØØØH
9ØØ2            PUSH      AF
                 .
                 .
                END       BEGIN
```

Notice that here, ORG 4ØØØH offsets the absolute start address of 5ØØØH. This causes the absolute address following ORG to be 9ØØØH (5ØØØH + 4ØØØH).

ASSEMBLY TERMINATION OR HOLD INSTRUCTIONS

ALDS contains several directives which terminate or hold the assembly. They are:

- . END - ends the assembly and saves the output object file
- . QUIT - quits the assembly
- . NOEND - ends assembly of a non-executable "load-only" program
- . STOP - temporarily halts the assembly

For example, all of the above programs contain an END directive. This tells the Assembler to end the assembly, store the assembled file, and return to TRSDOS READY.

In most programs, you'll want to use a parameter with END to specify the transfer address (the address of the first executable instruction in the program). The Assembler then stores the transfer address so that when loaded, the program immediately begins execution.

PROGRAM SECTIONS

All the above programs are "program sections". You can store several relocatable program sections in the same file. For example:

```
     MAIN      PSECT
ØØØØ'BEGIN     LD        A,3        ;begin first PSECT
.                .
Ø5ØØ'          RET
     SUB1      PSECT
ØØØØ'          LD        HL,DATA    ;begin second PSECT
.                .
Ø1ØØ'          RET
     SUB2      PSECT
ØØØØ'LOOP      LD        B,1Ø       ;begin third PSECT
.                .
Ø2ØØ'          SVC       36
               END       BEGIN
```

Since each section is independent, it must declare its symbols "PUBLIC" (discussed below) for another section to use them. Otherwise, two sections may not share the same symbols. (Only the MAIN program can use BEGIN; only SUB2 can use LOOP; and DATA must be defined in SUB1.)

Notice the Assembler initializes each program section to a relocatable ØØØØ. Now assume you link the program to an absolute start address of 3ØØØH:

```
3ØØØ BEGIN     LD        A,3        ;begin first PSECT
.                .
35ØØ           RET

35Ø1           LD        HL,DATA    ;begin second PSECT
.                .
36Ø1           RET

36Ø2 LOOP      LD        B,1Ø       ;begin third PSECT
.                .
38Ø2           SVC       36
               END       BEGIN
```

The Linker assigns each relocatable program section an address immediately following the preceding one.

USING EXTERNAL SYMBOLS

ALDS allows two or more program sections to share the same symbols. For example, you could write and test several independent subprograms -- such as PAYROLL, PAYABLES, RECEIVABLES, and INVENTORY. You could then mix and match them into separate application packages.

ALDS offers two ways of doing this:

1. By linking the programs into one file
2. By creating a "global symbol file"

The first is more common. The second is for special applications such as overlays where you want to use only the symbol definitions of an external program, but not the entire program itself.

1. COMBINING PROGRAM SECTIONS

For combining program sections, ALDS offers these directives:

- PUBLIC - declares symbols public
- EXTERN - declares symbols external
- LINK - appends an outside program file

These are actually directives to the Linker, as well as the Assembler.

As an example, assume you want to combine a subprogram named PAYROLL with a main program named ACCTG. You want both programs to share the same symbols. This is how you could go about it:

a. Declare the symbols you want shared.

You do this by using the PUBLIC or EXTERN directives at the beginning of your program. In the PAYROLL subprogram:

```
PAYROLL     PSECT
            PUBLIC    SUBPAY,MENU    ;SUBPAY and
                                     ;MENU are for
                                     ;PUBLIC use
                                     ;
            EXTERN    STORE1         ;STORE1 is in
                                     ;an EXTERNal
                                     ;program
                                     ;
SUBPAY      SVC       7              ;defines SUBPAY
                                     ;and clears
                                     ;screen
            LD        HL,MENU
            LD        B,23
            LD        C,ØDH
            SVC       9              ;print MENU
            LD        HL,STORE1
            LD        B,9
            SVC       9              ;print STORE1
            SVC       36             ;jump to TRSDOS
            RET
MENU        DEFM      'THIS BEGINS PAYROLL FOR'
                                      ;defines
                                      ;MENU
            END
```

The definitions for the symbols SUBPAY and MENU are declared PUBLIC. This means another program can use the same definitions.

The definition for STORE1 is declared EXTERNal. This means that although the existing program uses STORE1, an external program defines it.

In the ACCTG program:

```
ACCTG     PSECT
          PUBLIC    STORE1          ;STORE1 is for
                                    ;PUBLIC use
                                    ;
          EXTERN    SUBPAY,MENU     ;SUBPAY and
                                   ;MENU are in
                                   ;EXTERNal programs
                                   ;
MAIN      CALL      SUBPAY
STORE1    DEFM      'ABC DRUGS'
*
This part of the program defines other stores
*
          LINK      'PAYROLL/REL' ;insert
                                  ;PAYROLL/REL file
          END       MAIN
```

STORE1 is declared PUBLIC. This means that this program defines STORE1 and another program can use STORE1's definition.

SUBPAY and MENU are declared EXTERNal. They are used in this program but are defined in an external program (namely, PAYROLL).

If you want to try this exercise, use the ALDS Editor to insert the above two program files. Save the first as PAYROLL/SRC and the second as ACCTG/SRC.

b. Insert a directive to combine the programs

Notice LINK at the end of the ACCTG program. This tells the Linker to link the assembled code of PAYROLL at the end of ACCTG.

c. Assemble the programs

Assemble both the PAYROLL and ACCTG source program files in the normal way. In the TRSDOS READY mode, type:

```
ALASM PAYROLL PAYROLL <ENTER>
ALASM ACCTG ACCTG <ENTER>
```

The Assembler creates two relocatable files -- PAYROLL/REL and ACCTG/REL.

The Assembler marks every occurrence of the PUBLIC, EXTERN, and LINK directives, as well as every occurrence of EXTERNal symbols. However, you will need to use the Linker to complete the processing of these directives.

d. Link the programs

To link PAYROLL to ACCTG, you can use this Linker command at TRSDOS READY:

```
ALLINK ACCTG/REL ACCTG $=3ØØØ <ENTER>
```

The Linker processes the LINK, PUBLIC, and EXTERN directives and assigns the entire file absolute addresses beginning with 3ØØØH. This is done in two passes. In pass 1 the Linker:

- processes the LINK directive by linking PAYROLL/REL to the end of ACCTG/REL
- assigns the entire file absolute addresses
- creates a Linker Symbol Table which contains the definitions of all the symbols declared PUBLIC.

In pass 2, the Linker:

- fills in the values of all EXTERN symbols (using the Linker Symbol Table created in pass 1)
- saves the resulting program as ACCTG, an absolute object file.

e. Executing the program

You now have an absolute file, ACCTG, which consists of both ACCTG/REL and PAYROLL/REL. To execute it, type at TRSDOS READY

```
ACCTG <ENTER>
```

2. CREATING A GLOBAL FILE

Creating a global file is useful if you want to conserve memory by "overlaying" one program on top of the other. To create and use a global file, ALDS offers these directives:

- GLOBAL - declares symbols global
- EXTERN - declares symbols external
- GLINK - tells the Linker to use a global file
- EXT - tells the Assembler to use a global file

As an example, assume you want to create a file name MAIN which consists of a number of subroutines, such as printing lines on the display.

You also want to create several accouting system files, one of which is LEDGER. Users will use only one of these accounting systems at a time. However, each accounting system uses routines from MAIN.

It is therefore necessary to have MAIN and LEDGER in memory at the same time. However, there is not enough room in memory for both programs.

The alternative is to "overlay" one program on top of the other. In this example, MAIN loads LEDGER. When loaded LEDGER overlays sections of MAIN which it will not use.

These procedures clarify how this is done:

a. **Declare the symbols you want shared.**

This time, you do this with GLOBAL and EXTERN directives. In the MAIN program:

```
MAIN      PSECT
          GLOBAL    PRINT
BEGIN     SVC       7          ;clear screen
          CALL      ROUTINE
 ;
 ;load LEDGER routine begins here
 ;
          LD        HL,LEDGER
          LD        B,6
          SVC       38         ;load LEDGER file
 ;
 ; PRINT routine begins here
 ;
PRINT     LD        B,(HL)
          INC       HL
          LD        C,ØDH
          SVC       9          ;print contents of
                               ;register HL
          RET
LEDGER    DEFM      'LEDGER'
ROUTINE
 ;
 ;This part  of the program contains 18ØØØ bytes
 ;of subroutines which only MAIN uses.
 ;Since LEDGER does not need them
 ;LEDGER will load into this area
 ;
          RET
          END       BEGIN
```

The definition for PRINT is declared GLOBAL. When you assemble this program, the Assembler will create a global file named MAIN/GBL which contains PRINT's definition.

Notice that this program loads LEDGER. Also notice that it intends to load LEDGER on top of the ROUTINEs at the end.

This is the beginning of the LEDGER program:

```
LEDGER    PSECT
          EXTERN    PRINT
BEGIN     LD        HL,MENU
          CALL      PRINT
          SVC       36          ;jump to TRSDOS
MENU      DEFT      'THIS BEGINS THE GENERAL LEDGER MENU'
          ; the rest of the very long
          ; LEDGER program goes here
          ;
          GLINK     'MAIN/GBL'
          END       BEGIN
```

The definition for PRINT is declared EXTERN. Another program (MAIN) defines it.

(If you want to try this exercise, use the Editor to insert and save the first file as MAIN/SRC and the second as LEDGER/SRC.

b. Insert a directive to search the global file

Notice the GLINK directive in the above program. This tells the Linker to look for PRINT's definition in a global file named MAIN/GBL.

c. Assemble the programs

Assemble MAIN and LEDGER in the normal way:

```
ALASM MAIN MAIN <ENTER>
ALASM LEDGER LEDGER <ENTER>
```

The Assembler creates MAIN/REL and LEDGER/REL.

d. Link the program which creates the GLOBAL file

You must link MAIN/REL before linking LEDGER/REL. This is because MAIN/GBL contains a GLOBAL symbol that must be available to link LEDGER/REL. Type:

```
ALLINK MAIN MAIN $=3ØØØ <ENTER>
```

The Linker assigns absolute addresses to MAIN/REL beginning with 3ØØØH and saves the resulting absolute file as MAIN.

It also processes the GLOBAL directive. This causes it to create a global file named MAIN/GBL. This file contains only a symbol table defining PRINT.

e. Link the program which uses the GLOBAL file

After creating MAIN/GBL, you can link LEDGER. Type:

```
ALLINK LEDGER LEDGER $=31ØØ <ENTER>
```

The Linker processes the EXTERN directive. This tells it to look for PRINT's definition in an outside file.

It then processes the GLINK directive. GLINK tells the Linker to look for PRINT's definition in a file named MAIN/GBL.

The Linker also assigns absolute addresses to LEDGER/REL beginning 31ØØH.

f. Executing the program

You now have two absolute program files:

```
MAIN and LEDGER
```

Type:

```
MAIN <ENTER>
```

MAIN loads beginning at address 3ØØØH and begins executing. It then loads LEDGER beginning at address 31ØØH, which overlays the last portion of MAIN.

NOTES AND OPTIONS

ALDS offers several alternatives for linking programs:

- You can use INCLUDE rather than LINK. If you do this, you must include a source file rather than a relocatable object file. INCLUDE is a directive which the Assembler processes at assembly time. (See INCLUDE)

- You can use REF to reference only the symbol definitions of a source file only. (See REF)

- You can create indirect LINK files composed solely of LINK directives. By doing this, you can create several files containing different combinations of program sections. An example of this is PROGII and PROGIII in Chapter 1.

- You can use EXT rather than GLINK to combine absolute, as well as relocatable symbols. EXT is a directive to the Assembler (whereas GLINK is a directive to the Linker)

INDEX SECTIONS

ALDS contains directives which allow you to create an index section. They are:

- ISECT - begins an index section
- ENDI - ends an index section
- USING - associates an index register with an index section
- DROP - drops the index association established by USING

An index section is for EQUating symbols you want to use as offsets from an index register. For example:

```
PROG    PSECT   5ØØØH
        .
        ISECT   1       ;begins index section 1
DATA    EQU     1ØH
        ENDI            ;ends index section 1
        .
        LD      IX,4ØØØH
        USING   1,IX        ;associates IX
                            ;with the symbol
                            ;in index
                            ;section 1
```

```
LD        A,(DATA) ;loads A indexed
                   ;with IX, which
                   ;will be (IX+
                   ;DATA) or
                   ;(4ØØØH+1ØH)
.
DROP      1         ;drops association
                    ;of IX and index
                    ;section 1
.
LD        A,(DATA)  ;loads A with (DATA)
                    ;which is (1ØH)
```

Index section 1 (ISECT 1) equates DATA to 1ØH. USING associates all the symbol equations from ISECT 1 with index register IX. This means any time a symbol from ISECT 1 appears in the program, the Assembler generates an instruction to access memory with the indexed addressing mode (IX + the displacement value).

Later in the program, the Assembler encounters the symbol DATA (defined in ISECT 1.) The Assembler sets DATA as an offset to the IX register so that when you run the program, the processor will add DATA to the contents of register IX (The contents of register IX remains unchanged.)

Then the Assembler DROPs the association between IX and ISECT 1. After DROPping the association, the Assembler interprets DATA as simply DATA.

You can temporarily clear a USING association and return to it later with:

- APUSH - saves the current USING associations in an Assembler stack
- APOP - restores the USING status saved with APUSH by "popping" it from the Assembler stack

For more information, see the individual definitions of each directive.

MACRO SECTIONS

ALDS allows you to define your own "macro" symbol as a group of Z8Ø instructions. Whenever the Assembler encounters this macro symbol, it expands it into its defined Z8Ø instructions.

For example:

```
START     PSECT     3ØØØH
DISPLAY   MACRO     #L          ;begins macro
                                ;section defining
                                ;DISPLAY #L
                                ;(#L is a dummy
                                ;parameter)
          LD        HL,#L
          LD        B,(HL)
          INC       HL
          LD        C,ØDH
          SVC       9           ;display ViDeo LINE
          ENDM                  ;ends macro section
;
BEGIN     DISPLAY   FIRST       ;call DISPLAY and
                                ;pass it FIRST
;
          DISPLAY   SECOND      ;call DISPLAY and
                                ;pass it SECOND
;
          SVC       36          ;Jump to TRSDOS SVC
;
FIRST     DEFT      'THIS IS THE FIRST SENTENCE'
SECOND    DEFT      'AND THIS IS THE SECOND'
          END       BEGIN
```

The MACRO section begins with MACRO and ends with ENDM and in this example defines a MACRO named DISPLAY which displays a dummy parameter named #L.

The program then calls the DISPLAY macro and passes it the parameter FIRST. The Assembler expands this DISPLAY instruction into its macro definition, substituting FIRST for #L:

```
          LD        HL,FIRST
          LD        B,(HL)
          INC       HL
          LD        C,ØDH
          SVC       9
```

Next, the program calls the DISPLAY macro passing it the parameter SECOND. This expands into:

```
LD      HL,SECOND
LD      B,(HL)
INC     HL
LD      C,ØDH
SVC     9
```

When you assemble this program, notice that the macro SECTION (not the macro CALL) is for the Assembler's memory only. It is not assembled as part of the executable program.

For more information on macros, see MACRO.

IF SECTIONS

An "IF" section is a section of your program you only want assembled if a certain condition is true. ALDS offers these directives for conditional sections:

- IFT assemble if operand is a true expression
- IFF assemble if operand is a false expression
- IFZ assemble if operand equals zero
- IFNZ assemble if operand does not equal zero
- IFP assemble if operand is positive
- IFM assemble if operand is negative
- IFDEF assemble if operand is a defined symbol
- IFUND assemble if operand is an undefined symbol
- ELSE assemble if IF condition is false
- ENDIF end conditional section

For example, assume you want to create two versions of a program -- a Model II version and a Model III:

```
START   PSECT   7ØØØH
MODII   EQU     Ø           ;defines MODII
                            ;(any value will do)
BEGIN   LD      B,3
;
        IFDEF   MODII       ;assemble the following
                            ;IF MODII is DEFined
                            ;
```

```
SVC       36        ;Model II SVC call
                    ;
ELSE                ;assemble the following
                    ;if MODII is NOT defined
                    ;
JP        4Ø2DH     ;Model III SVC call
                    ;
ENDIF               ;END the IF section

END       BEGIN
```

IF the program defines the symbol MODII, the Assembler processes SVC 36 or ELSE it processes CALL 4Ø2DH.

The above program defines MODII. The Assembler processes SVC 36, thereby producing a Model II version of the program. To have the Assembler produce a Model III version, delete the MODII EQU Ø directive.

ASSEMBLER LISTING COMMANDS

Assembler listing commands change the way the Assembler processes the listing.

ALDS offers these listing commands:

- EJECT - ejects the printer listing to the next page
- VERSION - prints the time on the second line
- TITLE - prints a title on the third line
- HEADER - prints a heading on the fourth line
- PRINT - prints or does not print what you specify

See each directive listing for more information

OTHER ASSEMBLER COMMANDS

The remaining Assembler commands are:

- ADISP - displays or prompts you for information
- NOLOAD - assembles in memory image form
- OBJ - specifies the object file name to use
- PATCH - fills the remaining bytes in a sector with FF's to create a patch area

ASSEMBLER DIRECTIVES REFERENCE

The following pages list the syntax and a brief definition of the assembler directives available with ALDS. This is a definition of the terms used in the syntax:

expression
a valid assembler expression. (See Chapter 7.)

absolute expression
an expression with an absolute (non-relocatable, non-external) value. This can include a relocatable symbol as long as the resulting value is absolute. See Chapter 7.

expression list
one or more expressions, separated by commas

location
an expression designating an assembly location

filespec
a TRSDOS file specification (see your Owner's Manual).

string
a string of ASCII characters. The entire line must be 78 characters or less.

symbol
a one to ten character name which you may reference in your program.

symbol list
one or more symbols, separated by commas

ADISP

```
ADISP 'string^symbol'
ADISP 'string~symbol'
```

Displays or inputs certain parameters during the assembly of your program. You can specify one or both of these parameters:

(1) a string to be displayed
(2) a symbol to be displayed or input
- ^ <SHIFT> <6> causes the Assembler to display the symbol's value
- ~ <CTRL><6> causes the Assembler to prompt you to input the symbol's value

The Assembler executes ADISP during pass one only.

Example:

```
ADISP 'THE VALUE OF START IS ^START'
```

causes the Assembler to display: THE VALUE OF START IS followed by the value of the symbol START.

```
ADISP 'WHAT IS THE VALUE OF START ~START'
```

displays WHAT IS THE VALUE OF START? .... You can then input a hexadecimal value for START.

```
ADISP 'This is my Message'
```

displays the message.

```
ADISP '^$'
```

displays the current address of the PC (program counter) register.

```
ADISP     'NEW ORIGIN ~STARTLOC'
ORG       STARTLOC
```

displays NEW ORIGIN? and prompts you to input a value for STARTLOC. The next instruction resets the location counter to the value you input. Note that ADISP 'NEW ORIGIN ~$' does not accomplish the same thing.

APOP

```
APOP USING
APOP PRINT
APOP PRINT,USING
```

Restores the PRINT or USING status which was saved by a previous APUSH instruction.

Example:

```
APOP USING
```

restores the USING status.

```
APOP USING,PRINT
```

restores both the USING and PRINT status.

APUSH

```
APUSH PRINT
APUSH USING
APUSH PRINT, USING
```

Pushes the current PRINT and/or USING status into an assembly stack. Use APOP to get this current status back from the stack.

You may nest APUSH only one level deep. That is, you can not use APUSH twice without an APOP in between.

Examples:

```
APUSH USING
```

saves the USING status.

```
APUSH USING,PRINT
```

saves both the USING and PRINT status

APUSH is useful when you want the Assembler to treat a certain section of your program differently. For example:

```
          .
MAIN      PRINT     ON
          .
          PRINT     CON
          .
          PRINT     SHORT
          .
          .
          APUSH     PRINT
          PRINT     OFF
          CALL      SUB1
          APOP      PRINT
          .
```

When the Assembler encounters APUSH PRINT, the current status of PRINT is ON, CON, SHORT (print the first 6 bytes of all source lines, including conditionals).

The Assembler PUSHes this status into an assembly stack and turns PRINT OFF. This causes it not to print any lines in SUB1.

The Assembler then POPs the PRINT ON, CON, SHORT status back from the stack, which causes it to restore the printing status.

DATE

symbol DATE

Stores the current date in memory beginning with the current address. The optional symbol labels this address.

The Assembler stores the date as a string in the form of Day of Week, Month Date, Year (Model II) or MM/DD/YY (Model I/III).

For example, if today's date is Saturday, February 29, 1981:

DATE

stores SAT FEB 29, 1981 in Model II memory, or Ø2/29/81 in Model I/III memory.

DEFB

symbol DEFB expression
symbol DEFB absolute expression list
symbol DEFB absolute repeat count% absolute expression

Stores one or more one-byte expressions in memory beginning with the current address. The optional symbol labels this address. The optional repeat must be in the 1-255 range and will repeat a single absolute expression only.

```
TCONV     DEFB     NUM
```

stores NUM in the current memory address, defined as TCONV. NUM must be in the range of one byte numbers (-256 to +255 decimal).

If you use multiple expressions, all of them must be absolute. For example:

```
QSYM:     DEFB  7,9BH,BTABLE+3
```

stores decimal 7 at QSYM, the current memory address. Hexadecimal 9B and BTABLE+3 are stored in the next two bytes. None of these bytes can be relocatable. BTABLE must be defined in the existing program unit.

```
DEFB 128%'*'
```

fills the next 128 bytes with the character '*'.

You can substitute BYTE or DB for DEFB.

DEFE

symbol DEFE 'string'

Stores an "encrypted" string in memory beginning with the current memory address. The optional symbol labels this address.

Using DEFE makes it difficult for users to read the string by listing the object code. The first byte contains the unencrypted length of the string. The following bytes contains each character code XOR'd with 55H.

Example:

```
MESSAGE   DEFE  'hidden data'
```

stores 'hidden data' in the next 12 bytes and names the first byte MESSAGE. The first byte contains an ØBH (decimal 11). The next bytes contain codes for 'hidden data'.

DEFL

symbol DEFL expression

Defines symbol as expression. DEFL allows you to redefine a symbol in the same program. For example:

```
IMMED       DEFL  5
            ADD   A,IMMED
IMMED       DEFL  12
            ADD   A,IMMED
```

defines IMMED as 5 and adds it to the contents of register A. The next instruction defines IMMED as 12 and adds this to the contents of A.

Once you define a symbol with DEFL, you should not attempt to define it with EQU, EXTRN, or use it as a label.

DEFM

symbol DEFM 'string'

Stores string in memory beginning with the current address. The optional symbol labels this address. For example:

```
MESSAGE    DEFM 'THIS IS THE MESSAGE'
```

stores 'THIS IS THE MESSAGE' in the next 19 bytes and names the first byte MESSAGE.

You can use these two special characters in the string:

- the tilde "~" (typed as <CTRL> 6) to store a carriage return (hexadecimal ØD).
- the circumflex "^" (typed as <SHIFT> 6) to toggle the high bit (8ØH) on and off.

For example:

```
TEXT       DEFM '^J^OHN BROWN~M STREET'
```

stores JOHN BROWN <carriage return> M STREET in the next 19 bytes and flags the letter J by setting the high bit. J is stored as ØCAH, the code for J, plus 8ØH.

You can substitute ASCII for DEFM.

DEFR

symbol DEFR 'decimal number'

Converts a decimal number into a Roman numeral string and stores it in memory beginning at the current address. The first byte contains the hexadecimal length of the Roman numeral string. The following bytes contain the ASCII codes for the Roman numerals.

The decimal number must be in the range of 1 to 65535. The optional symbol allows you to name the first address.

For example:

```
DEFR '1981'
```

stores MCMLXXXI in the next 9 bytes. The first byte contains 8, the length of the Roman numeral string.

DEFS

symbol DEFS absolute expression

Reserves expression bytes, beginning with the current address, for storage. The optional symbol names this storage area.

This Assembler will not insert anything in the reserved area unless the FILL mode is in effect (see FILL).

Example:

```
          ORG  3ØØØH
BUF1      DEFS 1ØØH
BUF2      DEFS 5ØH
BUF3      DEFS 1Ø
START     LD   HL,BUF1
```

assigns BUF1 to location 3ØØØH, BUF2 to 31ØØH, and BUF3 to 315ØH. START begins execution at location 316ØH, loading HL with 3ØØØH.

You can substitute DS or BLOCK for DEFS.

DEFT

symbol DEFT 'string'

Stores string in memory, beginning with the current address. The optional symbol labels this address. The first byte contains the length of the string. You may use the two special characters described under DEFM (the tilde and the circumflex).

For example:

```
MESSAGE     DEFT   'this is my message'
```

stores the number 12H (decimal 18) in the next byte of memory and 'this is my message' in the following 18 bytes; then assigns the name MESSAGE to the address of the first byte.

DEFW

```
symbol DEFW expression
symbol DEFW absolute expression list
symbol DEFW absolute repeat count%
             absolute expression
```

Stores one or more two-byte expressions in memory beginning with the current memory address. The optional symbol labels this address. The least significant byte is stored first, followed by the most significant byte. The optional repeat must be in the 1-127 range and will repeat a single absolute expression only.

Examples:

```
MAXCNT    DEFW      1ØØØ
```

stores decimal number 1ØØØ in the next two bytes and labels that location as MAXCNT. Since 1ØØØ decimal is Ø3E8H, the first byte contains E8H and the second byte contains Ø3H.

```
          DEFW      3333,VAL
```

stores 3333 and VAL in the next four bytes. The same rules that DEFB uses for multiple expressions apply here. VAL must be defined in the existing program sections. Relocatable and external expressions may be used only if DEFW has a single, non-repeated expression.

```
          DEFW  3Ø%1ØØØ
```

fills the next 6Ø bytes with decimal 1ØØØs, repeated 3Ø times.

You can substitute DW or WORD for DEFW.

DROP

```
DROP 1
DROP 2
DROP
```

Terminates the index register association, specified by USING, with ISECT 1, ISECT 2, or all the ISECTs. This allows you to change USING associations. For example:

```
DROP    1
```

The index register is no longer associated with ISECT 1.

```
DROP
```

The index register is no longer associated with any of the ISECTs.

EJECT

```
EJECT
```

During the assembly listing, causes the printer to go to the next page before listing the next instruction. The EJECT instruction will not appear in the listing.

END

```
END address
```

Ends the assembly of the source program. The optional address causes the Assembler to store the entry address of the program.

Examples:

```
END       7FFFH
```

ends assembly and stores address 7FFFH in the assembled file as the entry point of the program. When you load the assembled file, it will immediately begin execution at address 7FFFH.

```
END       BEGIN
```

ends assembly and stores the address defined by BEGIN as the entry address.

```
END
```

ends assembly of the program. Since no entry point is specified, the Assembler stores it as absolute zero. This is an invalid entry point for TRSDOS. Therefore, you will be able only to load this program with the LOAD command -- not execute it.

ENDI

```
ENDI
```

Marks the end of an index section, initiated by ISECT.

ENDM

```
ENDM
```

Ends a macro definition, initiated by MACRO.

EQU

```
symbol EQU expression
```

Equates a symbol to an expression. For example:

```
START EQU 32ØØH
```

causes the symbol START to be equal to hexadecimal 32ØØ.

```
POINT     EQU  15+START
```

equates POINT to 3215, the sum of 15 and START.

Symbols defined by EQU may not be defined elsewhere in the program.

EXT

EXT 'filespec'

Tells the Assembler that the absolute definitions for certain symbols in your program are contained in the specified global file (created by GLOBAL). Since these symbols will have an established value at assembly time, you should not declare them EXTERNal or define them elsewhere in the program.

You can specify only one filespec per EXT instruction. It must have a /GBL extension. If you omit /GBL, the Assembler will automatically append it.

The EXT statement allows the programmer to have several absolute object files "talk" to each other. This requires considerable prior planning, but is useful and powerful.

Since EXT includes only the symbol definitions of the external program and not the program code, you will need to load the external program before attempting to use code in it.

For example:

```
EXT 'PROG1/GBL'
EXT 'PROG2'
```

tells the Assembler that your program contains symbols which are defined in PROG1/GBL and PROG2/GBL.

EXTERN

EXTERN symbol list

Declares that one or more symbols are not defined in the existing main program. They are defined externally in either:

- an external program section (which contains a corresponding PUBLIC instruction), or
- an external global file (which was created by a corresponding GLOBAL instruction).

For example:

```
     EXTERN      LOOP1,LOOP2
```

declares that LOOP1 and LOOP2 are defined externally.

You may substitute EXTRN for EXTERN.

FILL

     FILL

Causes any subsequent storage areas, initiated by DEFS, to be filled with zeros. Use NOFILL to turn it off.

For example:

```
          FILL
  BUF1    DEFS      1ØØ
          NOFILL
  BUF2    DEFS      2ØØ
```

BUF1 is filled with zeros. BUF2 is not filled with zeros.

You can use FILL only with DEFS instructions which reserve 255 or less bytes.

GLINK

     GLINK 'filespec'

Tells the Linker that the absolute definitions for certain symbols in your program are contained in the specified global file (created by GLOBAL). Your program must also contain an EXTERN instruction for each of the symbols referenced, to avoid undefined symbol errors.

You can specify only one filespec per GLINK instruction. It must have a /GBL extension. If you omit /GBL, the Linker will automatically append it.

GLINK accomplishes the same function as EXT, except it is an instruction to the Linker, rather than the Assembler. Because of this you need not have the external file written at assembly time, but you must have it loaded when you link the program.

For example:

```
GLINK 'PROG1'
GLINK 'PROG2'
```

tells the Linker that your file contains certain symbols which are defined in PROG1/GBL and PROG2/GBL.

GLINK must be the last instruction in your program before LINK, END, or another GLINK.

GLOBAL

GLOBAL symbol list

Declares one or more symbols as global and stores their values in a "global" file. Like PUBLIC, this permits another program section to use the same symbols. GLOBAL, however, goes one step further. It stores these symbols in a global file.

The global file will contain a symbol table only. It will define the absolute values of all the global symbols. If your program is absolute, the Assembler will create this global file. If your program is relocatable, the Linker creates it.

For example:

```
            PSECT     3ØØØH
            GLOBAL    DATA
DATA        DEFM      'THIS STARTS A DATA TABLE'
```

declares that DATA is a global symbol and stores DATA's value, hexadecimal 3ØØØ, in a global file. Since this program is absolute, the Assembler will create the global file.

```
PSECT
GLOBAL    LOOP1,LOOP2
```

declares that LOOP1 and LOOP2 are global symbols to be stored in a global file. Since this program is relocatable, the Linker will create the global file.

The global file will have the same name as the assembled object file with the extension /GBL. You will be able to access this file with any other program, provided it has these two instructions:

(1) GLINK, which specifies that some symbols in the global file should be used, and
(2) EXTERN, which specifies which global (or external) symbol definitions should be used

or simply:

(1) EXT, which tells the Assembler to look for the definitions of some symbols in the global file

Symbols declared PUBLIC or GLOBAL must be defined on both passes, that is, not defined with REF, ASISP, or EXT. The Linker may flag these symbols as undefined.

Symbols defined with DEFL more than once should not be declared PUBLIC or GLOBAL. The Linker will flag these symbols as multiply defined.

HEADER

HEADER '<u>string</u>'

Prints the specified <u>string</u> on the fourth line of each page in the assembly listing until the Assembler encounters a new HEADER instruction. HEADER starts a new page.

For example:

```
HEADER  'Electronics'
```

causes the Assembler to print "Electronics" on the fourth line of each page in the assembly heading.

For the header string to appear on the first page, HEADER must precede all listed instructions in the program. Otherwise, it ejects to the next page before printing the header string. TITLE, HEADER, and PRINT instructions are not listed.

You must specify a string when using HEADER. You may substitute HEADING for HEADER.

IFDEF

symbol IFDEF symbol

Assembles the following source lines IF the symbol is defined. IF NOT, the Assembler goes to the next ELSE or ENDIF directive. The optional symbol labels this directive.

```
IFDEF   SYMBOL
```

assembles the next lines IF the program defines SYMBOL. If not, the Assembler goes to the next matching ELSE or ENDIF. If the symbol is defined at all, it must be defined before the IFDEF.

The Assembler will not print the IF sections (instructions beginning with an IF directive and ending with ENDIF) unless PRINT CON is in effect. (See PRINT.)

All IF directives are nestable to six levels.

IFF

symbol IFF expression

Same as IFDEF except the expression must be false for the next lines to be assembled. For example:

```
IFF   5.GT.SYMBOL
```

assembles the next lines if 5 is not greater than SYMBOL.

IFM

symbol IFM expression

Same as IFDEF except the expression must be negative for the next lines to be assembled. For example:

```
IFM    SYMBOL
```

assembles the next lines if SYMBOL is a negative number.

IFNZ

symbol IFNZ expression

Same as IFDEF except the expression must not equal zero for the next lines to be assembled. For example:

```
IFNZ SYMBOL
```

assembles the next lines if SYMBOL does not equal zero.

IFP

symbol IFP expression

Same as IFDEF except the expression must be positive for the next lines to be assembled. For example:

```
IFP  SYMBOL
```

assembles the next lines if SYMBOL is a positive number.

IFT

symbol IFT expression

Same as IFDEF except the expression must be true (that is, bit Ø must be 1) for the next lines to be assembled.

For example:

```
IFT   5.GT.SYMBOL
```

assembles the next lines IF 5 is greater than SYMBOL.

IFUND

symbol **IFUND** symbol

Same as IFT except the symbol must not be defined for the next lines to be assembled. For example:

```
IFUND   SYMBOL
```

assembles the next lines if the program does not define SYMBOL. If the symbol is defined at all, it must be defined before the IFDEF.

IFZ

symbol **IFZ** expression

Same as IFDEF except the expression must equal zero for the next lines to be assembled. For example:

```
IFZ  SYMBOL
```

assembles the next lines if SYMBOL equals zero.

INCLUDE

INCLUDE '**source filespec**'

Inserts filespec at the point where INCLUDE appears in the program. The Assembler will assemble the INCLUDEd file before processing the next instruction.

The optional END instruction of the INCLUDEd file tells the Assembler to continue assembling the main program. The END of the main program will terminate the assembly.

You may specify only one filename per INCLUDE. You may use as many INCLUDE instructions as you want.

For example:

```
INCLUDE 'PROG1'
```

inserts and assembles PROG1, a source file, before processing the next instruction.

```
INCLUDE 'PROG1'
INCLUDE 'PROG2'
```

inserts and assembles PROG1; then inserts and assembles PROG2; then proceeds with the next instruction.

INCLUDE is nestable to five levels. That is, file 1 can call file 2; 2 can call 3; 3 can call 4; and finally, 4 can call 5. But at no time can a called file (file 5) call a calling file (file 4). This results in an Error 37 -- Open attempt for a file already open.

ISECT

ISECT name

Begins an "index section" of EQU instructions, terminated by ENDI. If you wish, you can name the section 1 or 2 (no other names are allowed).

Using an index section allows you to specify certain index symbols. You can then use the index symbols to offset an index register.

For example, this is an index section named ISECT 1:

```
          ISECT 1
SYMBOL1   EQU  5
SMBL3     EQU  3
SMBL26    EQU  26
SYMBL     EQU  1ØØ
          ENDI
```

It specifies four index symbols. Whenever the Assembler encounters one of these index symbols enclosed in parentheses, it evaluates it as the expression:

(the contents of an index register + index symbol)

You must specify which index register to use with the USING instruction. For example:

```
LD        IY,4ØØØH
USING     1,IY
LD        A,(SYMBOL1)
```

The Assembler evaluates this as:

```
LD        IY,4ØØØH
USING1,   IY
LD        A,(IY+SYMBOL1)
```

You cannot use a register name or a flag condition to name an index symbol.

LINK

```
LINK 'filespec'
LINK 'filespec(symbol)'
```

Tells the Linker to insert filespec, an absolute or relocatable object file, at the point where LINK is encountered in the current program. This instruction is similar to INCLUDE, except it applies only to the Linker. It allows you to link one or more files together.

LINK must be at the end of your program section. (Only END, GLINK, or another LINK can follow it.) Each LINK instruction can specify only one filename. You can use as many LINK instructions as you want.

For example:

```
LINK      'FILE1'
LINK      'FILE2'
END       PROG
```

inserts FILE1 and then FILE2 at the end of your main program. FILE1 and FILE2 must both be assembled object files.

```
LINK      'TAX(TABLE)'
```

inserts a program section named TABLE which exists in a file named TAX at the end of your program. TAX must be an object file. TABLE is a PSECT label.

The LINK statement is nestable to five levels. That is, file 1 can call file 2, 2 can call 3, 3 can call 4, and finally, 4 can call 5. But at no time can a called file (file 5) call a calling file (file 4).

LITORG

symbol LITORG location

Allows you to specify where to place literals used as operands. LITORG should be used only once per assembly and placed in the same PSECT as all references to the literals, and after the last reference.

If you omit the optional location, the Assembler stores the literals in the current location. If you include it, LITORG resets the location counter (in the same way that ORG does) and stores the literals at the newly reset location.

The optional symbol labels this location. The Assembler assigns the remaining instructions locations immediately following the literals.

All literal operands must be preceded by an equal sign (=) and surrounded with single quotes ('). For example:

```
        LD      HL,='INPUT THE ITEM NUMBER'
```

This instruction uses INPUT THE ITEM NUMBER as a literal operand. Here is how you could use it in a program:

```
START   PSECT   3ØØØH
BEGIN   LD      HL,='INPUT THE ITEM NUMBER'
        LD      B,(HL)
        INC     HL
        SVC     9
        SVC     36
        LITORG
        DEFM    'THIS IS A LONG TABLE OF PROMPTS'
        DEFM    'INPUT THE ITEM NUMBER'
        DEFM    'INPUT THE PRICE'
        DEFM    'IS THERE A DISCOUNT?'
        DEFM    'INPUT THE DISCOUNT'
        END     BEGIN
```

Notice that INPUT THE ITEM NUMBER is defined by DEFM later in the program. The Assembler stores it in two locations: (1) the location where LITORG appears in the program, and (2) the location where DEFM 'INPUT THE ITEM NUMBER' appears.

Note that if literals are used and the program ends with a LINK or GLINK, LITORG is mandatory to place the literals before the LINK or GLINK statement.

MACRO

name **MACRO** dummy parameter list

Begins a section of the program which defines a macro name. Use ENDM to end this macro definition.

The optional dummy parameter list allows you to pass parameters to the macro. You may use up to ten dummy parameters separated by commas. Each can be only one character and must be preceded by a # sign.

Defining a macro allows you to "call" an entire block of instructions with a single program line. This is useful when you will be using the same block many times in your program.

For example, this is a macro definition:

```
SCROLL    MACRO
          LD        B,1Ø
          SVC       27
          ENDM
```

which defines a macro named SCROLL. Every time the Assembler encounters SCROLL, it "expands" SCROLL into the LD B,1Ø and SVC 27 instructions. That is, if this is your source program:

```
          .
          LD        A,3
          SCROLL
          LD        HL,DATA
          .
```

The Assembler will interpret SCROLL as a macro call and expand it into the appropriate instructions:

```
          .
          LD        A,3
          LD        B,1Ø
          SVC       27
          LD        HL,DATA
          .
```

The next example defines a macro named ADNUM which acts on four dummy parameters named #Ø, #1, #2, and #3:

```
ADNUM     MACRO     #Ø,#1,#2,#3
          ADD       A,#Ø
          ADD       A,#1
          ADD       A,#2
          ADD       A,#3
          ENDM
```

This definition allows you to "pass" four values to ADNUM when you call it. For example:

```
ADNUM     B,1Ø,NUMB,LST
```

calls ADNUM and passes four values to it. The Assembler expands this macro call into:

```
ADD       A,B
ADD       A,1Ø
ADD       A,NUMB
ADD       A,LST
```

Notice that B, the first value, replaces #Ø, the first parameter; 1Ø replaces #1; NUMB replaces #2; and LST replaces #3.

When using a macro, remember that you must define it before you use it. You might want to put all the macro definitions in one file and then INCLUDE or REF them at the beginning of your main file.

We do not recommend that you use a macro name which is the same as an extended mnemonic or directive name. If you do this, the Assembler will use the definition you assigned the macro. This will of course give undesirable results.

When using dummy parameters, be sure not to insert them inside quoted strings. If you do this, the Assembler will treat them as ordinary characters.

A macro cannot call another macro.

NOEND

NOEND

Ends the assembly of a non-executable program. The Assembler marks the assembled code as load-only and will not execute the file when used as a TRSDOS command. This command is useful for creating overlays to be loaded with the DOSCMD supervisor call.

NOFILL

NOFILL

Terminates the mode initiated by FILL.

NOLOAD

NOLOAD

Assembles the program sequentially in memory image form, rather than in the standard TRSDOS object format. You must use NOLOAD as the first line of the main source file (before coments, titles, PSECT, etc.), otherwise some TRSDOS object code load headers may be placed into the file.

You cannot use NOLOAD with these features:

- the relocatable mode
- EXTERNal, or PUBLIC symbols
- LINK or GLINK

If you want the file to contain an accurate memory image of the program, you must also avoid these instructions:

- DEFS (unless the FILL mode is on)
- ORG
- more than one PSECT

(These instructions change the value of the location counter but do not output object code. This causes the load address and location counter to differ.)

OBJ

OBJ '<u>filespec</u>'

Tells the Assembler that it should write the assembled <u>filespec</u> to disk. The Assembler will ignore this instruction if you specify an object filespec in the assembly command line.

Example:

```
OBJ  'ACCOUNTS'
```

Unless you specify an object filespec in the assembly command line, the above instruction saves the assembled object program as ACCOUNTS.

ORG

symbol ORG location, boundary

Resets the Assembler's location counter to the specified location. For example, in an absolute program:

```
ORG      4ØØØH
```

resets the location to an absolute 4ØØØH.

In a relocatable program:

```
ORG      4ØØØH
```

resets the location counter to a relocatable 4ØØØH. Assuming you link the program to an absolute start address of 5ØØØH, the Linker determines the effective address to be 9ØØØH, the sum of 5ØØØ and 4ØØØ.)

The second parameter allows you to reset the location counter to a boundary divisible by decimal 2, 4, 8, 16, 32, 64, 128, or 256. For example, if the value of the counter is currently 4ØØ5H:

```
ORG      $,4
```

resets the counter to 4ØØ8H, which is the next highest number divisible by decimal 4.

Unlike many other assemblers, ORG will not initialize the location counter. You need to use PSECT for this purpose.

ORG will not change the location counter from the relocatable to the absolute mode, or vice versa. You must assemble absolute and relocatable programs as different files.

location may not be an external symbol.

PATCH

PATCH

Fills the remaining bytes in the last sector in the assembled object file with FF's. This reserves an area for patches.

The Assembler will print a message on pass 2 giving the address and length of the patch area (if the file produces object code).

This must be the last command prior to the END directive. You cannot use it with LINK, and it is for use with absolute assemblies only.

PRINT

PRINT command list

Controls what is printed or not printed in the assembly listing. You may use one or more of the following commands, separated by commas or blank spaces:

ALL - print all source lines (Same as ON,MAC,CON)

ON - print all normal open code source instructions

OFF - do not print anything except error messages and diagnostics until (1) the end of the assembly or (2) a PRINT ON command

MAC - print all source lines generated in macro expansions (except those which might be overridden by other PRINT options).

NOMAC - do not print source lines generated by macro expansions. Only the macro instruction itself will appear in the listing file.

CON - print all conditional assembly source lines, whether they generate code or not.

NOCON - print only the conditional assembly source lines that generate code.

LST - output the listing, regardless of what was on the command line. The listing will be printed on the video, and if the D or P options were specified, the listing will also go to disk or to the printer. You cannot save this option with APUSH.

NOLST - do not output a listing, regardless of what was on the command line.

SHORT - print only the first 6 bytes of object code generated by each line.

LONG - print all of the object code generated, even if it requires several lines.

For example:

```
PRINT MAC,SHORT
```

prints all the macro expansions in the assembly listing. It limits printing to the first six bytes of object code for each line.

Only PRINT instructions specifying OFF, NOMAC, and NOCON will appear in the listing.

You can use comments with PRINT.

PRINT defaults to ON, MAC, NOCON, LONG.

PSECT

symbol PSECT location

Initializes the Assembler's location counter to a relocatable zero or to the absolute location you specify. The Assembler assembles all subsequent instructions sequentially throughout the program.

The optional symbol labels the program section and can be up to six characters. This symbol is for the Linker, and will be listed on the Linker map. The symbol will not be defined by the Assembler and cannot be used in expressions.

PSECT begins an independent, executable "program section". You can have several relocatable program sections in one program file. One program section cannot use symbols from another program section unless you declare them EXTERN and PUBLIC.

For example:

```
            PAYROLL   PSECT
ØØØØ'       BEGIN     LD        A,3
                        .
            PAYABLE   PSECT
ØØØØ'                 PUSH      A
                        .
                      END
```

This program has two sections: "PAYROLL" and "PAYABLE". Both begin with a relocatable ØØØØ. When you link this file, the Linker assigns "PAYABLE" addresses which immediately follow "PAYABLE". Since no symbols are declared PUBLIC and EXTERNal, "PAYROLL" and "PAYABLE" cannot share the same symbols.

The following instructions do no have to be part of a program section:

- comments
- index sections
- conditional assembly instructions
- macro sections
- macro instructions (which will not affect the location counter)
- EQU or DEFL (as long as they do not reference the location counter)
- assembler directives (which do not affect the location counter)

You can define symbol (with EQU, for example) prior to your first PSECT. This permits you to use a conditional assembly such as:

```
          IFT       RELOC
XYZ       PSECT
          ELSE
XYZ       PSECT     3ØØØH
          ENDIF
```

which starts a relocatable PSECT if RELOC equals 1, and an absolute PSECT if RELOC equals Ø. Doing this will create two PSECTs with the same name, one being zero-length. This will appear on the Linker map but it will not affect the assembly.

The PSECTs within an assembly must either be all relocatable or all absolute. Relocatable and EXTERN expressions cannot be used in absolute assemblies.

The PSECT location you specify cannot be an external value.

PUBLIC

PUBLIC symbol list

Declares one or more symbols as "public". This permits another program section to use the same symbols.

When you assemble a program with public symbols, the Assembler will mark all their definitions. Then, when you link it to an external progam section, the Linker will insert these definitions in the Linker Symbol Table.

For example:

```
PUBLIC    LOOP1
```

declares LOOP1's definition to be public.

Another program can use the public symbol definitions provided it contains a corresponding EXTERN directive.

You can substitute ENTRY for PUBLIC.

Symbols declared PUBLIC or GLOBAL must be defined on both passes, that is, not defined with REF, ADISP, or EXT. The Linker may flag these symbols as undefined.

Symbols defined with DEFL more than once should not be declared PUBLIC or GLOBAL. The linker will flag these symbols as multiply defined.

QUIT

QUIT

Quits the assembly and returns to TRSDOS READY. This Assembler only recognizes this instruction at the second pass of a listing (specified by the L assembly option). It will not save the object file.

RADIX

RADIX expression

Specifies expression as the default number base. That is, the Assembler will interpret any numbers without a base suffix in the default base.

You may use any expression with a value of 2, 8, 10, or 16. Without RADIX, the Assembler defaults to 10 (decimal).

For example:

```
RADIX     16
```

causes the Assembler to interpret all the numbers which do not have "b" or "d" suffixes as hexadecimal numbers.

Remember that the Assembler uses the current default base to evaluate your RADIX instruction. For example, if you want to change the default base of 16 to 1Ø, use RADIX 1Ød or ØA, not RADIX 1Ø. While in base 16, the Assembler would evaluate the 1Ø as a hexadecimal 1Ø.

Example:

```
RADIX     1ØH   ;Use Hexadecimal
DEFB      1B     ; This is 1B (hex) = 27 (decimal)
DEFB      1b     ; This is 1 (binary)
DEFB      25     ; This is 25 (hex) = 37 (decimal)
RADIX     1Ø     ; Radix is still hex (1Ø hex =
                 ; 16 decimal)
RADIX     1ØD    ;ERROR 1ØD hex = 269 decimal --
                 ; too large.
RADIX     1Ød    ; Radix is now decimal
DEFB      1B     ; This is a 1 binary
DEFB      1b     ; This is also a 1 binary
DEFB      25     ; This is 25 (decimal) = 19 (hex)
```

REF

REF 'source filename'

Includes only the symbol definitions from the specified source file. This is useful for referencing a file of EQU directives or MACROs.

REF tells the Assembler to INCLUDE the source file during Pass 1 only. After processing the source file, the Assembler restores the location counter to its original value. Thus, the Assembler uses the referenced file's symbols, but not its assembled code.

For example:

```
REF  'TEST/SRC'
```

The Assembler will define macros and symbols contained in TEST/SRC. It will not insert the code for TEXT/SRC.

The Assembler will not report any errors in the referenced file. Also, if there is a conflict between symbols of the referenced file and the main program, the first definitions will be used with no error message. You might want to use INCLUDE instead of REF until all conflicts have been resolved.

Symbols defined in the REF file should not be declared PUBLIC or GLOBAL. The Linker may flag these symbols as undefined.

RESLOC

RESLOC location

Resets the location counter to the location computed as:

the value of the counter prior to executing SETLOC	+	the number of bytes of code generated by the SETLOC block

For example, assuming the value of the location counter was 3ØØØH prior to SETLOC and there are two 3-byte instructions following SETLOC:

```
RESLOC
```

resets the location counter to 3ØØ6H.

SETLOC

SETLOC location

Temporarily changes the location counter's value to the absolute location specified. The Assembler uses this changed location for defining symbols only. It does not use the changed location for assembling the instructions.

For example:

```
4ØØØ            LD        A,3
                SETLOC    3ØØØH
3ØØØ POS        PUSH      AF
```

The actual PUSH AF instruction is not stored at location 3ØØØH. Rather, it is stored at 4ØØ2H, the location which immediately follows LD A,3. However, the Assembler defines POS, the symbol which labels the location of PUSH AF, as 3ØØØH.

SETLOC is useful anytime you are writing a routine which you want to load in one location, and then move and execute at a different location. By using SETLOC, the Assembler defines this routine's symbols as if they were already in their execution location.

For example, you might want to run a memory test from a very low memory address. You cannot load it on top of TRSDOS. However, after loading it, you can move and execute it in that location. Since TRSDOS will be overwritten, the memory test must do its own input/output.

Using SETLOC, you could write the routine this way:

```
                    PSECT     3ØØØH
                      .
31ØØ        MOVE    EQU       $           ;SETLOC block begins
                    SETLOC    5ØØH
25ØØ        LOOP    LD        A,3
                      .                   ;code for memory
26ØØ                  .                   ;test
                    RESLOC
32ØØ        LDBLOCK EQU       $-MOVE      ;SETLOC block ends
                      .
                    LD        HL,MOVE     ;move SETLOC block
                    LD        DE,LOOP     ;to its proper loop
                    LD        BC,LDBLOCK
                    LDIR
                    JP        LOOP
                      .
```

Here, the Assembler defines LOOP as though it were at address 5ØØH -- the address the program will eventually move it to. However, it actually assembles the code for LOOP at address 31ØØ.

MOVE defines where the actual assembled code of the SETLOC block (ended by RESLOC) begins. LDBLOCK defines the length of the SETLOC block by subtracting MOVE from the current contents of the PC register. (The $ sign indicates the current value of PC).

LDIR then moves the SETLOC block from location 31ØØ, defined by MOVE, to location 5ØØ. Since LOOP has already been defined as if it were at location 5ØØ, you do not have to redefine it.

> Note: If your program is relocatable, SETLOC still sets an absolute location. You need to avoid using these instructions within the SETLOC block: ORG, DEFS (unless the FILL mode is in effect), PSECT, and relocatable
> and external expressions.

STOP

STOP

Stops the assembly listing. Press any key to continue the listing. Press <BREAK> to abort it.

TIME

symbol **TIME**

Stores the time in memory as a string beginning at the current address. The optional symbol labels this address. For example if the time is 1:45 p.m. and 55 seconds when the Assembler reaches this instruction:

TIME

it will store the string 13.45.55 (Model II) or 13:45:55 (Model III) in the next eight bytes of memory.

TITLE

TITLE 'string'

Prints the specified string on the third line of each page in the assembly listing. For example:

```
TITLE     'THIS IS THE TITLE'
```

prints THIS IS THE TITLE on the third line of every page.

If you are using both TITLE and HEADER, TITLE should precede HEADER (otherwise the TITLE will not appear until the next page).

USING

USING index section name, index register
USING index register, expression
USING index register

Associates an index register -- IX or IY -- with the index sections. For example:

```
USING     IX
```

associates IX with all the ISECTS.

You can optionally specify one (but not both) of the following:

- an index section name (1 or 2), as the only section to be associated with the register
- an expression to be loaded into the register

For example:

```
USING     1,IX
```

associates the IX register with ISECT 1 only.

```
USING     IX,DCB
```

loads IX with the value of DCB, then associates IX with all the ISECTs.

The index sections are specified with the ISECT instruction.

USING does not apply to any external program sections.

VERSION

```
VERSION
```

Prints the current time on the second line of the assembly listing heading.

*** (block comment)**

```
*
```

Turns on and off the block comment function. The asterisk must be in the first column.

When the Assembler encounters a line beginning with an asterisk, it begins interpreting the lines as comments rather than instructions. The next asterisk ends the block comment.

For example:

```
*
The following program is a ...
.
.
.
.
*
```

Be careful when using the asterisk. One asterisk out of place near the beginning of your program can cause the Assembler to treat most of your program as a comment.

Chapter 10/ EXTENDED Z80 INSTRUCTIONS

The ALDS Assembler contains a number of extended Z80 instructions. You can use them the same way you use other Z80 instructions.

An extended instruction is actually an internally defined macro. When you assemble the instruction, the Assembler expands it into a group of Z80 instructions. A description of macros is in Chapter 8.

NOTATIONS

In addition to the notations described in Chapter 9, this chapter uses:

xx	a register pair
yy	a register pair
[ ]	optional value

FORMAT OF EACH INSTRUCTION

This chapter uses the same format for the instructions as Chapter 9, with the following exceptions:

- many of the instruction formats show different combinations of operands. These combinations are listed under "Operands"
- following the description of each instruction is a breakdown of how the instruction expands when assembled
- the operation is not shown
- the object code is not shown

CPR operand ComPare double Register

Mnemonic: CPR Operand: xx (where xx=BC, DE, HL, or SP)

Description:

Compares the contents of the operand to the contents of HL. If they compare, the Z bit is set.

Example: If register pair BC contains an A0H and HL contains and A0H.

```
CPR BC
```

sets the Z bit.

Expansion of: CPR xx

```
PUSH    HL
OR      A
SBC     HL,xx
POP     HL
```

CMPD operand1,operand2,[length] CoMPare with Decrement

Mnemonic:	CMPD	Operands:	nn1,nn2,n	length is n
			nn1,nn2	length is contents of BC
			nn1,nn2,(nn3)	length is contents of nn3
			nn1,(nn2)	length is last byte of the string beginning at operand2

Description:

Compares the string beginning at operand1 and ending at (operand1 - length) with the string beginning at operand2, and ending at (operand2 - length). The Z bit is set according to the result of the comparison. Zero length strings are equal.

If a mismatch occurs, HL and DE will contain the addresses preceeding that mismatch.

Example: If memory location 4000 - 4006 contains the string1 "develop" and location 5000 - 5006 contains the string2 "envelop, the operation

```
CMPD 4006H,5006H,7
```

starts the comparison of the two strings with the last byte, in this case the 'p'. A mismatch occurs at the second letter. Because of this mismatch, the address of the preceding 'n' is now in register HL and the address of the preceding 'e' in register DE.

Exit Conditions:
all registers modified

Expansion: CMPD nn1,nn2,n

```
LD      DE,nn1
```

```
      LD      HL,nn2
      LD      BC,n
X2:   LD      A,B
      OR      C
      JR      Z,X1
      LD      A,(DE)
      CP      (HL)
      JR      NZ,X1
      LDD
      JR      X2
X1:
```

Expansion: CMPD nn1,nn2

```
      LD    DE,nn1
      LD    HL,nn2

X2:   LD    A,B
      OR    C
      JR    Z,X1
      LD    A,(DE)
      CP    (HL)
      JR    NZ,X1
      LDD
      JR    X2
X1:
```

Expansion: CMPD nn1,nn2,(nn3)

```
      LD    DE,nn1
      LD    HL,nn2
      LD    A(nn3)
      LD    C,A
      LD    B,0
X2:   LD    A,B
      OR    C
      JR    Z,X1
      LD    A(DE)
      CP    (HL)
      JR    NZ,X1
      LDD
      JR    X2
X1:
```

Expansion: CMPD nn1,(nn2)

```
      LD    DE,nn1
      LD    HL,nn2
      LD    C,(HL)
      LD    B,0
      INC   HL
X2:   LD    A,B
      OR    C
      JR    Z,X1
      LD    A(DE)
      CP    (HL)
      JR    NZ,X1
      LDD
      JR    X2
X1:
```

Note: The symbols used in the expansion are shown for clarity and are not actually defined for use by other statements.

CMPI operand1,operand2,length CoMPare with Increment

Mnemonic:	CMPI	Operands:	nn1,nn2,n	length is specified.
			nn1,nn2	length in BC.
			nn1,nn2 (nn3)	length is contents of nn3.
			nn1,(nn2)	length is first byte of nn2

Description:

Compares the string beginning at operand1 with the string beginning at operand2 for the given length. Depending on the operands, length can be specified as a constant, the contents of an address, or the contents of the BC register. If a match does not occur, HL and DE will contain the addresses following that mismatch. The Z bit is set according to the result of the comparison. Zero length strings are equal.

Example: If memory location 4000 - 4006 contains the string1 "develop" and location 5000 - 5006 contained the string2 "envelop":

```
CMPI 4000H,5000H,7
```

starts the comparison of the two strings beginning with the first byte (in this case, the 'd' in string1 and the 'e' in string2). A mismatch occurs at the first letter. The address of 'd' is now in register DE and the address of 'e' is now in register HL where the comparison failed.

Exit Conditions:
all registers modified

(1) Expansion: CMPI nn1,nn2,n

```
     LD   DE,nn1
     LD   HL,nn2
     LD   BC,n
X2:  LD   A,B
```

```
      OR    C
      JR    Z,X1
      LD    A(DE)
      CP    (HL)
      JR    NZ,X1
      LDI
      JR    X2
X1:
```

(2) Expansion: CMPI nn1,nn2

```
      LD    DE,nn1
      LD    HL,nn2
X2:   LD    A,B
      OR    C
      JR    Z,X1
      LD    A,(DE)
      CP    (HL)
      JR    NZ,X1
      LDI
      JR    X2
X1:
```

(3) Expansion: CMPI nn1,nn2,(nn3)

```
      LD    DE,nn1
      LD    HL,nn2
      LD    A,(nn3)
      LD    C,A
      LD    B,0
X2:   LD    A,B
      OR    C
      JR    Z,X1
      LD    A,(DE)
      CP    (HL)
      JR    NZ,X1
      LDI
      JR    X2
X1:
```

(4) Expansion: CMPI nn1,(nn2)

```
      LD    DE,nn1
      LD    HL,nn2
      LD    C,(HL)
      LD    B,0
```

```
      INC   HL
X2:   LD    A,B
      OR    C
      JR    Z,Xl
      LD    A,(DE)
      CP    (HL)
      JR    NZ,Xl
      LDI
      JR    X2
Xl:
```

Note: The labels used in the expansion are shown for clarity and are not actually defined for use by other statements.

TZ operand Test register for Zero-TZ

Mnemonic: TZ Operand: xx (where xx= BC, DE, HL, IX, or IY)

Description:

Compares the contents of xx to zero. If true, the Z bit is set.

Example: If the contents of BC contains a 00H then the operation

TZ BC

sets the Z bit. Any other value (i.e. A0H) sets the NZ bit.

```
**************************************************************
*  NOTE:  TZ IX and TZ IY are instructions which are not     *
*  documented by ZILOG.  Although they should assemble       *
*  properly, Radio Shack does not guarantee that they will   *
*  work on all processors.  You should test them in your     *
*  own environment to ensure their validity.                 *
**************************************************************
```

Expansion: TZ xx

```
        LD    A,high order byte of xx
        OR    low order byte of xx
```

EX operand **EXchange double register with (SP)--EX**

Mnemonic: EX operand: (SP),xx where xx=AF, BC, or DE

Description:

Exchanges the low order byte contained in xx with the contents of the memory address specified by the contents of the stack pointer (SP). The high order byte of xx is exchanged with the next highest memory address (SP + 1).

Example: If the contents of the register pair BC is 3978H and the stack pointer (SP) and its next byte (SP+1) contains 2357H:

```
EX (SP),BC
```

causes the register pair BC to contain 2357H and the top address of the stack to contain 4978H.

(1) Expansion for: EX (SP),xx where xx=AF or BC

```
EX    (SP),HL
PUSH  xx
PUSH  HL
POP   xx
POP   HL
EX    (SP),HL
```

(2) Expansion for: EX (SP),DE

```
EX    DE,HL
EX    (SP),HL
EX    DE,HL
```

EX operand1,operand2 **EXchange double register**

Mnemonic: EX Operand: xx,yy where xx and yy are any register pairs listed under "Expansion" below.

Description:

Exchanges the two-byte contents of xx with the contents of yy.

Example: The contents of BC is 6789H and the contents of DE is 1234H. After the execution of:

```
EX BC,DE
```

the values are exchanged so that BC contains 1234H and DE contains 6789H.

(1) Expansion for:

```
EX    AF,BC
EX    AF,DE
EX    BC,DE
```

```
PUSH   1st Operand
PUSH   2nd Operand
POP    1st Operand
POP    2nd Operand
```

(2) Expansion for: EX xx,yy (xx = AF, BC or DE; yy = IX or IY)

```
PUSH    1st Operand
EX      (SP),2nd Operand
POP     1st Operand
```

(3) Expansion for:

```
EX    HL,xx    (xx = IX or IY)
EX    IX,IY
EX    xx,HL, (xx=AF of BC)
```

```
PUSH    1st Operand
EX      (SP),2nd Operand
POP     1st Operand
```

(4) Expansion for: EX (SP), xx (xx=AF, BC)

```
EX        (SP), HL
PUSH      2nd Operand
PUSH      HL
POP       2nd Operand
EX        (SP), HL
```

(5) Expansion for: EX (SP), DE

```
EX        (SP), HL
```

```
EX          DE, HL
EX          (SP), HL
```

LoaD double register--LD

Mnemonic: LD Operand: xx,yy
(xx),yy
xx,(yy)
(xx),(yy)

Description:

Loads the first operand with the second operand. The numbers shown in the tables (1-14) represent the coded expansions for the pair of operands. Details of each expansion follow the tables (i.e. BC,AF refer to expansion description #1).

Example: The operation:

```
LD HL,DE
```

copies the contents of DE to HL.

First Operand	< BC	< DE	< HL	< (BC)	Second Operand (DE)	(HL)	> (IX+DD)	> (IY+DD)
(BC)	4	4	6	--	9	9	9	9
(DE)	4	4	7	9	__	9	9	9
(HL)	5	5	8	9	9	9	9	9
(IX+DD)	5	5	5	9	9	9	--	9
(IY+DD)	5	5	5	9	9	9	9	--

First Operand	< AF	< BC	< DE	< HL	> IX	> IY	> A
AF	1	1	1	1	1	1	--
BC	1	3	3	3	1	1	2
DE	1	3	3	3	1	1	2
HL	1	3	3	3	1	1	2
IX	1	1	1	1	1	1	2
IY	1	1	1	1	1	1	2

First Operand	< (BC)	< (DE)	< (HL)	> (IX+DD)	> (IY+DD)
BC	11	12	10	10	10
DE	12	11	10	10	10
HL	13	13	14	10	10

(--) - indicates operand pairs not applicable

(1) Expansion of:: LD xx,yy where xx and yy are any of the following operand pairs:

```
AF,AF ; AF,BC ; AF,DE ; AF,HL ; AF,IX ; AF,IY
BC,AF ; BC,IX ; BC,IY
DE,AF ; DE,IX ; DE,IY
HL,AF ; HL,IX ; HL,IY
IX,AF ; IX,BC ; IX,DE ; IX,HL ; IX,IX ; IX,IY
IY,AF ; IY,BC ; IY,DE ; IY,HL ; IY,IX ; IY,IY
```

```
PUSH    2nd Operand
POP     1st Operand
```

(2) Expansion of: LD xx,yy where xx and yy are any of the following operand pairs:

```
          BC,A ; DE,A ; HL,A ; IX,A ; IY,A

     LD     Low order byte of register pair,A (accumulator)
     LD     High order byte of register pair,0

*************************************************************
*  NOTE:  LD IX,A  and LD IY,A  are instructions which are  *
*  not documented by ZILOG.  Although they should assemble  *
*  properly, Radio Shack does not guarantee that they will  *
*  work on all processors.  You should test them in your    *
*  own environment to ensure their validity.                *
*************************************************************
```

(3) Expansion of: LD xx,yy where xx and yy are any of the following operand pairs:

```
          BC,BC ; BC,DE ; BC,HL
          DE,BC ; DE,DE ; DE,HL
          HL,BC ; HL,DE ; HL,HL

     LD     High order byte 1st Operand, High order byte 2nd Operand
     LD     Low order byte 1st Operand, Low order byte 2nd Operand
```

(4) Expansion of: LD xx,yy where xx and yy are any of the following operand pairs:

```
          (BC),BC ; (BC),DE
          (DE),BC ; (DE),DE

     PUSH   1st Operand
     EX     (SP),HL
     LD     (HL),Low order byte 2nd Operand
     INC    HL
     LD     (HL),High order byte 2nd Operand
     EX     (SP),HL
     POP    1st Operand
```

Side Effect: First operand register is incremented by 1.

(5) Expansion of: LD xx,yy where xx and yy are any of the following operand pairs:

```
          (HL),BC ; (HL),DE
```

(IX+DD),BC ; (IX+DD),DE ; (IX+DD),HL
(IY+DD),BC ; (IY+DD),DE ; (IY+DD),HL

```
LD      (1st Operand),Low order byte 2nd Operand
INC     Register of 1st operand
LD      (1st Operand),High order byte 2nd Operand
```

Side Effect: first operand register is incremented by 1.

(6) Expansion of: LD (BC),HL

```
PUSH    AF
LD      A,L
LD      (BC),A
INC     BC
LD      A,H
LD      (BC),A
POP     AF
```

Side Effect: Register BC is incremented by 1.

(7) Expansion of: LD (DE),HL

```
PUSH    AF
LD      A,L
LD      (DE),A
INC     DE
LD      A,H
LD      (DE),A
POP     AF
```

Side Effect: Register DE is incremented by 1.

(8) Expansion of: LD (HL),HL

```
PUSH    AF
LD      A,H
LD      (HL),L
INC     HL
LD      (HL),A
POP     AF
```

Side Effect: Register HL is incremented by 1.

(9) Expansion of: LD xx,yy where xx and yy are any of

the following operand pairs:

(BC),(DE) ; (BC),(HL) ; (BC),(IX+DD) ; (BC),(IY+DD)
(DE),(BC) ; (DE),(HL) ; (DE),(IX+DD) ; (BC),(IY+DD)
(HL),(BC) ; (HL),(DE) ; (HL),(IX+DD) ; (HL),(IY+DD)
(IX+DD),(BC) ; (IX+DD),(DE) ; (IX+DD),(HL);(IX+DD),(IY+DD)
(IY+DD),(BC) ; (IY+DD),(DE) ; (IY+DD),(HL);(IY+DD),(IX+DD)

```
LD      A,(2nd Operand)
LD      (1st Operand),A
```

Side Effect: Register A is changed.

(10) Expansion of: LD xx,yy where xx and yy are any of the following operand pairs:

BC,(HL) ; BC,(IX+DD) ; BC,(IY+DD)
DE,(HL) ; DE,(IX+DD) ; DE,(IY+DD)
HL,(IX+DD) ; HL,(IY+DD)

```
LD      Low order byte 1st Operand,(2nd Operand)
INC     Contents of 2nd Operand, register
LD      High order byte 1st Operand,(2nd Operand)
```

Side Effect: 2nd operand Register is incremented (HL,IX or IY)

(11) Expansion of: LD xx,(yy) where xx and (yy) are either of the following operand pairs: BC,(BC) ; DE,(DE)

```
PUSH    Contents of 2nd Operand
EX      (SP),HL
LD      Low order byte of 1st Operand,(HL)
INC     HL
LD      High order byte of 1st Operand,(HL)
POP     HL
```

(12) Expansion of: LD xx,(yy) where xx and (yy) are either of the following operand pairs: BC,(DE) ; DE,(BC)

```
PUSH    Contents of 2nd Operand
EX      (SP),HL
LD      Low order byte of 1st Operand,(HL)
INC     HL
```

```
LD      High order byte of 1st Operand,(HL)
EX      (SP),HL
POP     Contents of 2nd Operand
```

Side Effect: 2nd operand register is incremented by 1.

(13) Expansion of: LD xx,(yy) where xx and (yy) are either of the following operand pairs: HL,(BC) ; HL,(DE)

```
PUSH    AF
LD      A,(2nd Operand)
LD      L,A
INC     Contents of 2nd Operand
LD      A,(2nd Operand)
LD      H,A
POP     AF
```

Side Effect: 2nd operand Register is incremented by 1.

(14) Expansion of: LD HL,(HL)

```
PUSH    AF
LD      A,(HL)
INC     HL
LD      H,(HL)
LD      L,A
POP     AF
```

MOVD operand1,operand2,length MOVe with Decrement

Mnemonic: MOVD	Operand:		
		nn1,nn2,n	length is specified
		nn1,nn2	length is in BC
		nn1,nn2,(nn3)	length is contents of nn3 (byte)
		nn1,(nn2)	length is first byte of nn2.

Description:

Moves a string of a given length (implied in the operand) from the address of operand2 to the address of operand1. MOVD starts at the end of the string and moves backward starting at the address of operand2.

You can specify the length as a constant, the contents of an address, or the contents of the BC register.

Example: If the address 4000 contained the string "develop":

MOVD 5000H,4000H,7

moves "develop" from address 3FFA-4000 to 4FFA-5000 starting with the end of the string, (i.e. 'p') which would be located at address 5000H .

(1) Expansion: MOVD nn1,nn2,n

```
     LD   DE,nn1
     LD   HL,nn2
     LD   BC,n
     LDDR
```

(2) Expansion: MOVD nn1,nn2

```
     LD   DE,nn1
     LD   HL,nn2
     LD   A,B
     OR   C
     JR   Z,X1
     LDDR
X1:
```

(3) Expansion: MOVD nn1,nn2,(nn3)

```
     LD   DE,nn1
     LD   HL,nn2
     LD   A,(nn3)
     LD   C,A
     LD   B,0
     OR   A
     JR   Z,X1
     LDDR
X1:
```

(4) Expansion: MOVD nn1,(nn2)

```
     LD   DE,nn1
     LD   HL,nn2
     LD   C,(HL)
     LD   B,O
     INC  HL
     LD   A,B
     OR   C
     JR   Z,X1
     LDDR
X1:
```

MOVI operand1,operand2,length **MOVe with Increment**

Mnemonic: MOVI Operands: nn1,nn2,n length is specified.
nn1,nn2 length is in BC
nn1,nn2,(nn3) length is contents of nn3
nn1,(nn2) length is first byte of nn2.

Description: Moves a string of the given length from the address of operand2 to the address of operand1. MOVI starts at the beginning of the string and moves forward.

You can specify the length as a constant, the contents of a memory address, or the contents of the BC register.

Example: If location 4001H contains the string "develop", the instruction:

```
MOVI 5000H,4000H,7
```

moves "develop" from address 4001H to 5000H starting with d, the first letter.

(1) Expansion: MOVI nn1,nn2,n

```
     LD   DE,nn1
     LD   HL,nn2
     LD   BC,n
     LDIR
```

(2) Expansion: MOVI nn1,nn2

```
     LD   DE,nn1
     LD   HL,nn2
     LD   A,B
     OR   C
     JR   Z,X1
     LDIR
X1:
```

(3) Expansion: MOVI nn1,nn2,(nn3)

```
      LD    DE,nn1
      LD    HL,nn2
      LD    A,(nn3)
      LD    C,A
      LD    B,0
      OR    A
      JR    Z,X1
      LDIR
X1:
```

(4) Expansion: MOVI nn1,(nn2)

```
      LD    DE,nn1
      LD    HL,nn2
      LD    C,(HL)
      LD    B,0
      INC   HL
      LD    A,B
      OR    C
      JR    Z,X1
      LDIR
X1:
```

POP

Mnemonic: POP Operand: none

Description:

Increments the stack pointer one full word.

Example: If the stack pointer contains the byte 39H on top and 45H in the next location

POP

increments the stack pointer past these two bytes to the next point.

Expansion:

```
INC    SP
INC    SP
```

RSTR operand ReSToRe

Mnemonic: RSTR Operand: n where n=

none	restores HL,DE BC
4	restores HL,DE BC and AF
I	restores HL,DE BC,AF,IX,IY
P	restores HL,DE, BC,AF,IX,IY,HL' DE',BC'
A	restores HL,DE, BC,AF,IX,IY,HL' DE',BC',AF'

Description:

Restores the registers specified by the operand after a SAVE (see extended instruction). This is often used after a return from a subroutine.

Example: If registers HL, DE, BC are saved (See SAVE),

```
RSTR
```

restores them to their original values.

(1) Expansion: RSTR

```
POP     HL
POP     DE
POP     BC
```

(2) Expansion: RSTR 4

```
POP     HL
POP     DE
POP     BC
POP     AF
```

(3) Expansion: RSTR I

```
POP     HL
```

```
        POP     DE
        POP     BC
        POP     AF
        POP     IY
        POP     IX
```

(4) Expansion: RSTR P

```
        POP     HL
        POP     DE
        POP     BC
        POP     AF
        POP     IY
        POP     IX
        EXX
        POP     HL
        POP     DE
        POP     BC
        EXX
```

(5) Expansion: RSTR A

```
        POP     HL
        POP     DE
        POP     BC
        POP     AF
        POP     IY
        POP     IX
        EXX
        POP     HL
        POP     DE
        POP     BC
        EXX
        EX      AF,AF'
        POP     AF
        EX      AF,AF'
```

SAVE operand

Mnemonic: SAVE Operand: n where n=

none	saves HL,DE,BC
4	saves HL,DE,BC AF
I	saves HL,DE,BC, AF,IX,IY
P	saves HL,DE,BC AF,IX,IY,HL' DE', BC'
A	saves HL,DE,BC, AF,IX,IY,HL', DE',BC',AF'

Description:

Copies the contents of the registers specified by the operand. This is useful before executing a subroutine. The registers are restored with RSTR (see extended instruction).

Example:

SAVE

saves the contents of registers HL, DE, BC. to free them for use, then execute an SAVE.

(1) Expansion: SAVE

```
PUSH    BC
PUSH    DE
PUSH    HL
```

(2) Expansion: SAVE 4

```
PUSH    AF
PUSH    BC
PUSH    DE
PUSH    HL
```

(3) Expansion SAVE I

```
      PUSH    IX
      PUSH    IY
      PUSH    AF
      PUSH    BC
      PUSH    DE
      PUSH    HL
```

(4) Expansion: SAVE P

```
      EXX
      PUSH    BC
      PUSH    DE
      PUSH    HL
      EXX
      PUSH    IX
      PUSH    IY
      PUSH    AF
      PUSH    BC
      PUSH    DE
      PUSH    HL
```

(5) Expansion: SAVE A

```
      EX      AF,AF'
      PUSH    AF
      EX      AF,AF'
      EXX
      PUSH    BC
      PUSH    DE
      PUSH    HL
      EXX
      PUSH    IX
      PUSH    IY
      PUSH    AF
      PUSH    BC
      PUSH    DE
      PUSH    HL
```

SuperVisory Call--SVC

Mnemonic: SVC Operand: n

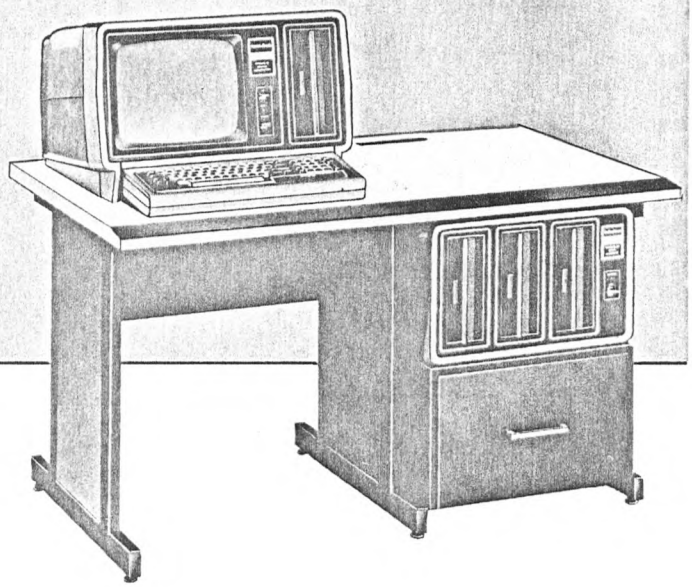
Performs the supervisory call specified by n.

Expansion:

```
LD      A,n
RST     8
```

Radio Shack®

Section 3
Error Messages



CUSTOM MANUFACTURED IN THE U.S.A. BY RADIO SHACK, A DIVISION OF TANDY CORPORATION

SECTION III

ERROR MESSAGES

EDITOR ERROR MESSAGES

BAD FILE FORMAT

The file is not a type ALEDIT can load, either fixed LRL 1 or Variable, and with record length not greater than 256 bytes.

BAD FILENAME FORMAT

The filename is too long or incorrectly formatted on a load or a write command.

BAD PARAMETERS

The ASCII line number converted to hexadecimal is greater than 65535 decimal (for line number request).

The change string is zero or the length of the line to be changed is zero (for Change command).

BUFFER FULL

There is no more room in the edit buffer. Program returns from any mode back to the command mode. Note that the edit buffer is about 4K smaller if DO, HOST, COMM, SPOOL, DEBUG or ALBUG are on.

LINE LENGTH TOO LONG, TRUNCATING LINE

You are loading a file that has lines longer than 78 characters.

LINE NUMBER TOO LARGE

The line number is larger than the last line number in the file.

The editor does not recognize your command. Re-type it.

NO TEXT
The edit buffer is empty, the only commands which are effective are:

K, L, Y, I, Q, J, S

OCCURRENCE TOO LARGE
In the Find and Change commands the occurrence is greater than 255.

SEARCH ARG TOO LONG

The string you want to search for is longer than 37 characters.

SYNTAX ERROR

The command is improperly specified.

TOTAL LINE LENGTH TOO LONG

The new line created by a Change command is greater than the acceptable Line Length.

If the Editor returns an error code, it is a TRSDOS error message. You can identify it, by simply typing in the error number. For example, at TRSDOS READY type:

ERROR 19 <ENTER>

or at the Editor command mode, type:

S ERROR 19 <ENTER>

and your computer answers you with the correct identification:

IMPROPER FILE NAME (filespec)

You can do this any time your computer identifies an error which you are not aware of.

HIT ANY KEY TO CONTINUE

If there is an error in the load or write routines, the Editor waits for the user to read the entire error message.

ASSEMBLER ERROR CODES

CODE	MEANING
A	Arithmetic Overflow -- result of a multiplication is outside the range of -65536 - +65535
B	Balance Error of Brackets
C	Condition Error ELSE outside an IF ... ENDIF pair Unterminated IF ENDIF without matching IF Macro defined after a macro was expanded
D	Macro Definition Error ENDM outside a macro definition Macro not terminated when END statement was reached. Parameter substitution (i.e. "#9") specified in the body of the macro for a parameter not listed in the heading. Macro body too long.
E	Missing END statement Missing ENDM statement
F	Include files nested too deeply
I	Illegal character Control character in source file.
L	Maximum Line Length Exceeded. The limit is 254 characters a line
M	Multiple Definition of a Symbol This includes defining a symbol and declaring it EXTRN
O	Stack Overflow -- expression too complicated
P	Phase Error -- Symbol appears or changes value after Pass 1. This is often caused by using symbols in the operand field of EQU, DEFS, or ORG before those symbols are defined.

R	Range Error in Relative Addressing. Use a JP instead of JR, or rearrange code.
S	Syntax Error Illegal operation code Too few, too many, or the wrong type of operands Use of an external symbol or relocatable expression where it is not allowed Use of an instruction generating object code within an ISECT Use of an instruction before a PSECT Instruction illegal after a LINK directive
T	Mixing of absolute and relocatable PSECTs
U	Undefined Symbol
V	Illegal Value Value too large to fit in a single byte (-256 - +255 permitted) Illegal combination of relocatable or external symbols
W	Reserved word used as a symbol. Do not use a register name or branch condition as a symbol

LINKER ERROR MESSAGES

Symbol Table Overflow

There are too many external symbols to fit in memory. Reduce the number of symbols declared public or global by assembling several modules together, or using shorter names.

Multiply Defined Entry Symbol

The indicated symbol has been defined more than once (and declared public and/or global). The two or more definitions may be in the same object file (the assembler will output an 'M' error) or in different files. Note that using the same name for a public or global symbol in one file and for a local symbol (not declared PUBLIC, GLOBAL or EXTRN) in another file is permitted.

Address Different from Pass 1

The indicated symbol changed values between Pass 1 and Pass 2. Normally this error is preceeded by a "Multiply Defined Entry Symbol" message and the cause is the same. This error may also be caused by changing disks in the middle of a link, inserting a disk with a different version of the same object file in a lower drive number during the link, or linking corrupted object files.

The two addresses are the values from Pass 1 and Pass 2 respectively. These values and the PSECT map may be used to locate the modules containing the definitions, assuming that the value falls withing the code area of the module.

Undefined External Symbol

The indicated symbol is declared EXTRN in at least one module and is never defined and declared PUBLIC or GLOBAL in any module included in the link. This is usually caused by failing to declare a label PUBLIC, omitting files that should have been included in the link, or linking incomplete programs to test just the implemented parts. In the last case, if the instructions referring to the undefined symbol are never used, the error may be ignored.

Missing External Transfer Address

The main program ends with NOEND, or the object file has been corrupted. The main program should terminate with END and a transfer address.

Illegal Addressing

The load address being computed by the linker wraps around from FFFF to 0000. Reduce the size of your program or use a lower load address.

Invalid Parameter

The LINKs are nested too deeply; an illegal character was specified in a filename on the command line, LINK, or GLINK instruction, the source filename is missing, or errors were found in the $=XXXX parameter.

LINKER TRSDOS ERRORS

ERROR 24 (File Not Found)

Object file not found.

Note: default extension is /REL.

ERROR 34 (Attempt to Use a Non-Program File As a Program)

The file used is incomplete or in NOLOAD format, or is not an object file.

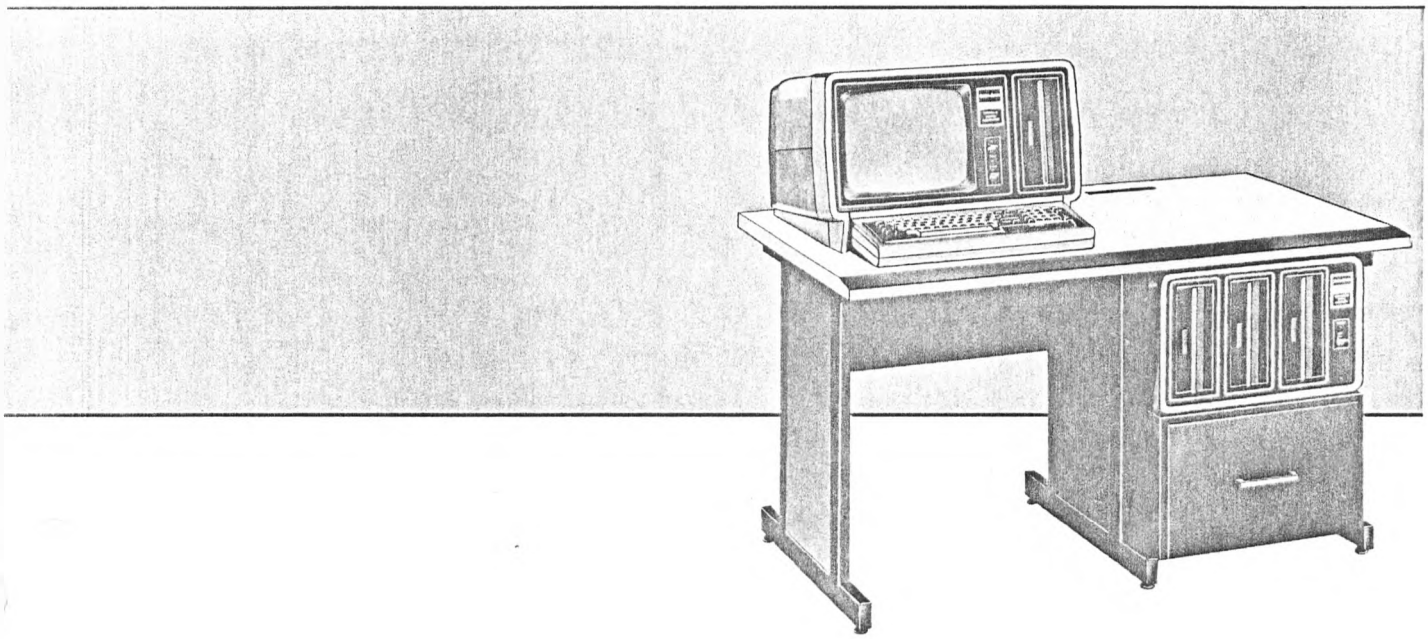
ERROR 37 (Open Attempt For a File Already Open)

Another file, directly or indirectly, attempted to include itself with a LINK directive.

Note: default extension is /REL. Also, other errors may include: disk read/write errors, password protection, illegal disk change, disk full etc.

Radio Shack®

Section 4
Appendices



CUSTOM MANUFACTURED IN THE U.S.A. BY RADIO SHACK, A DIVISION OF TANDY CORPORATION

Appendix A/ UNDOCUMENTED Z80 INSTRUCTIONS

```
**************************************************************
*  These instructions are not documented by                  *
*  ZILOG.  Radio Shack does not guarantee that they will     *
*  work on all processors.  You should test them in your     *
*  own environment to ensure their validity.                 *
**************************************************************
```

SHIFT/LOAD INSTRUCTIONS

```
**************************************************************
*  These instructions are not documented by                  *
*  ZILOG.  Radio Shack does not guarantee that they will     *
*  work on all processors.  You should test them in your     *
*  own environment to ensure their validity.                 *
**************************************************************
```

In the following list, the undocumented instructions on the left perform the same function as the corresponding instructions on the right, except that the memory location data is shifted or rotated and stored in both the register and the memory location.

RLCLD	r,m	RLC	m
RLLD	r,m	RL	m
RRCLD	r,m	RRC	m
RRLLD	r,m	RRL	m
SLALD	r,m	SLA	m
SLOLD	r,m	SLO	m
SRALD	r,m	SRA	m
SRLLD	r,m	SRL	m

r is one of the following registers: A,B,C,D,E,H, or L
m is one of the following: (IX+d) or (IY+d)

The operation of the condition code bits and instruction timing is believed to be the same as for the corresponding shift or rotate instruction.

OBJECT CODE:

1	1	X	1	1	1	0	1

DD for (IX+d)
FD for (IY+d)

1	1	0	0	1	0	1	1

CB

d	d	d	d	d	d	d	d

0	0	n	n	n	r	r	r

n =		r =	
	RLCLD 0		111 A
	RLLD 2		000 B
	RRCLD 1		001 C
	RRLLP 3		010 D
	SLALD 4		011 E
	SLOLD 6		100 H
	SRALD 5		101 L
	SRLLD 7		

BIT SET/LOAD AND BIT RESET/LOAD INSTRUCTIONS

```
*****************************************************************
*  NOTE:  These instructions are not documented  by             *
*  ZILOG.  Radio Shack does not guarantee that they will        *
*  work on all processors.  You should test them in your        *
*  own environment to ensure their validity.                    *
*****************************************************************
```

In the following list, the undocumented instructions on the left perform the same function as the corresponding instructions on the right except that the resulting data after the bit operation is loaded in both the memory location and the register.

RESLD	r,n,m	RES	n,m
SETLD	r,n,m	SET	n,m

r is one of the following registers: A,B,C,D,E,H or L
n is a bit number with value between 0 and 7, inclusive
m is either (IX+d) or (IY+d)

OBJECT CODE:

1	1	X	1	1	1	0	1

DD for (IX+d)
FD for (IY+d)

1	1	0	0	1	0	1	1

CB

d	d	d	d	d	d	d	d

x x	n n n	r r r

x = 10 RESLD
11 SETLD

n = bit number

r =

111	A
000	B
001	C
010	D
011	E
100	H
101	L

INDEX REGISTER HALF INSTRUCTIONS

```
************************************************************
*  NOTE:  These instructions are not documented by         *
*  ZILOG.  Radio Shack does not guarantee that they will   *
*  work on all processors.  You should test them in your   *
*  own environment to ensure their validity.               *
************************************************************
```

The upper and lower bytes of the index registers IX and IY may be manipulated individually. To use these instructions, the following register names are used:

XH	High Byte of IX
XL	Low Byte of IX
YH	High Byte of IY
YL	Low Byte of IY

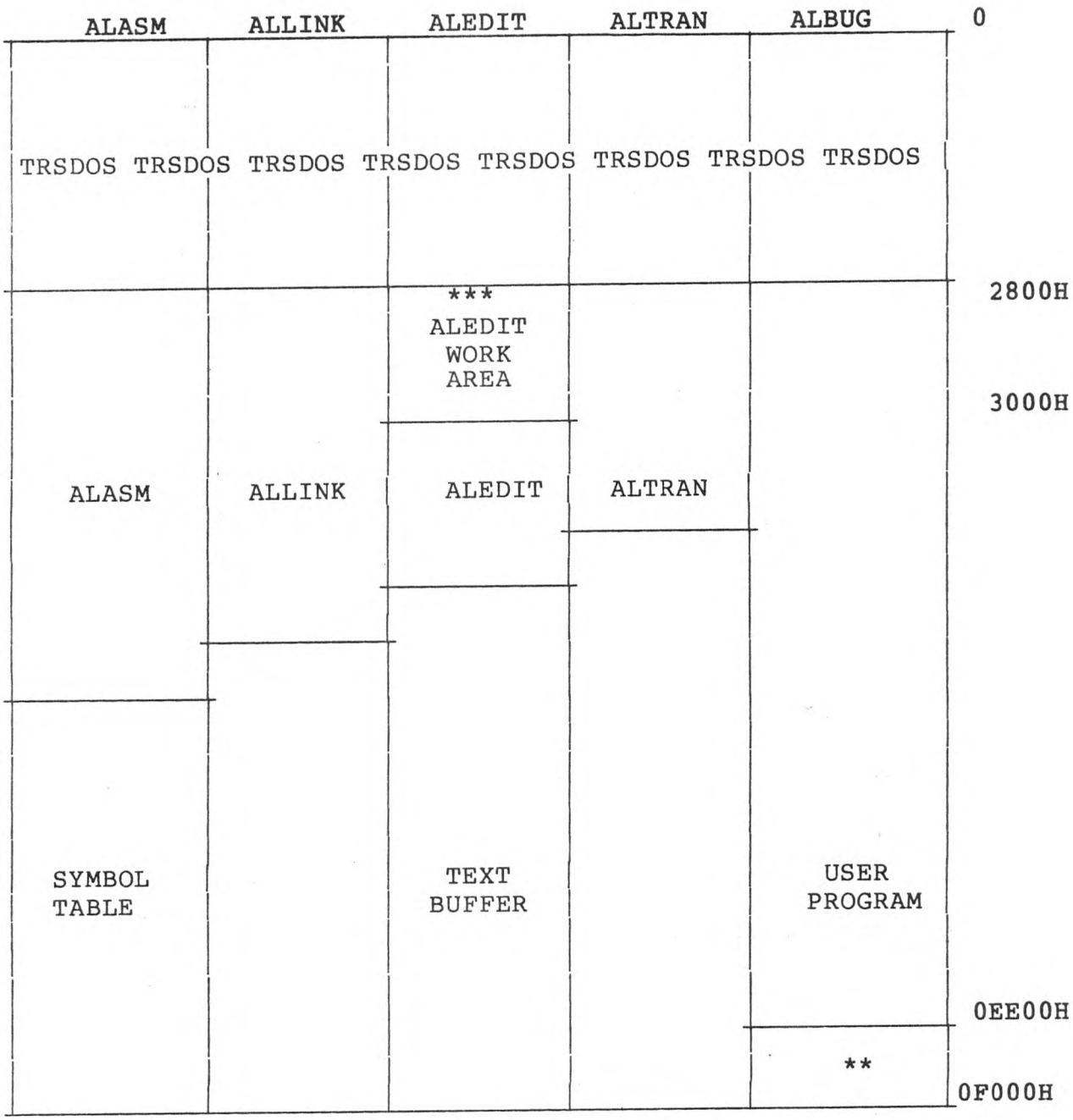
The object code generated has a prefix byte of DD or FD (for the halves of the IX or IY register) and otherwise is the same as the corresponding instructions with the H or L register used in place of the high or low byte of an index register.

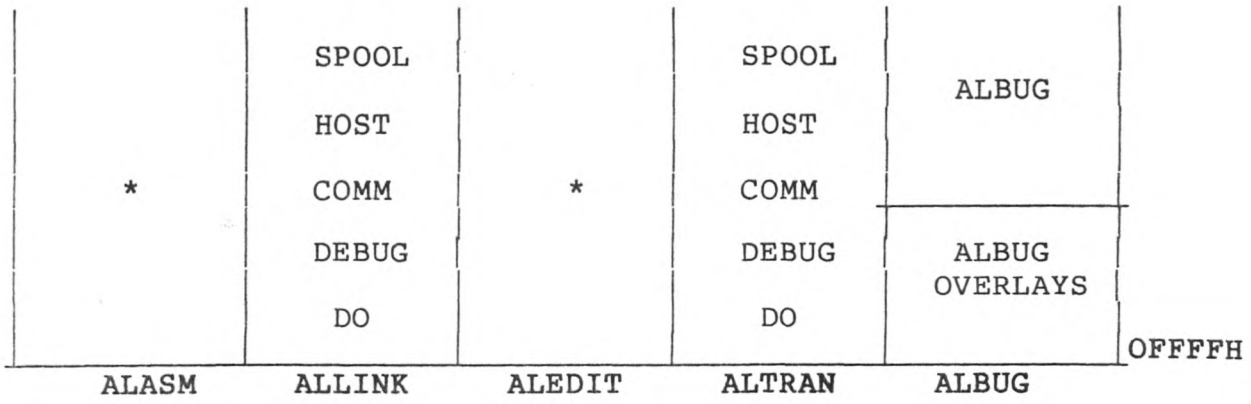
The XH, XL, YH and YL registers may be used in the following instrusssons:

```
ADC   A,XH          LD    r,XH
ADD   A,XH          LD    XH,r
AND   XH            LD    XH,n
CP    XH            OR    XH
DEC   XH            SBC   A,XH
INC   XH            SUB   XH
                    XOR   XH
```

r = A, B, C, D, or E

Appendix B/ MEMORY MAP





* - Used by ALASM and ALEDIT if SPOOL, HOST, COMM, DEBUG, and DO are not present.

** - Used only during DISK ZAP command and when first loading ALBUG. User programs may write over it.

*** - Shared by the Editor and programs run with the Editor "S" command. The Editor assumes other programs will use this area and sets it up when an "S" command completes.

Appendix C/ CONVERTING SERIES I EDITOR-ASSEMBLER PROGRAMS

To convert programs created for the Series I Editor-Assembler, so that they can be used with ALDS, follow these steps:

1. Load the Series I Editor-Assembler program into

   ALEDIT:

   ```
   ALEDIT    filespec/ext
   ```

2. Using the editor, make the following changes:

   a) Change the first ORG statement to PSECT, with the same operand.

   b) Change all '*LIST ON' statements to ' PRINT ON'

   c) Change all '*LIST OFF' statements to ' PRINT OFF'

   d) If the < operator is used for a shift operation, change

   ```
   value < n to value .SHL. n       (n positive)
   value <-n to value .SHR.(-n)     (n negative)
   ```

   e) Be sure the program has an END statement

   f) Remove all lines that begin with '*' or change the '*' to a ';' .

3. Write the file to disk in ALDS format (no option characters to specify the file format).

4. Assemble the program with the ALDS assembler.

Appendix D/ ALDS OBJECT CODE FORMAT

Each record is a variable number of bytes, packed consecutively in an LRL 256 file. Records may span sector boundaries. The file is terminated by a record with an 02 or 03 header. For further information, see the Model II Owner's Manual.

	HEADER (1)	LENGTH (1)			
Object Code	01	n + 2	Load address (2)	Data bytes (n)	
*Absolute Entry	02	02	Absolute Entry Point (2)		*One of These Records Terminates Each Object File
*Relocatable Entry	02	03	Relocatable Entry Offset (2)		
*Load - only	03	02	0 0 0 0 (2)		
*External Entry	03	0D	FLAGS 01000011	Object (2)	External Name(10)
Relocatable Object Data	04	03	FLAGS 00001xxx	Object (2)	
External Object Data	04	0D	FLAGS 01001lxx	Object (2)	External Name(10)
Public Label w/Object	04	0F	FLAGS 100xlxxx	Public Label Offset (2)	

Object (2)	Public Label Name (10)

Public Label w/o Object	04	0F	FLAGS 100x0011	Public Label Offset (2)
			0 0 0 0 (2)	Public Label Name (10)
Public Label w/External	04	19	FLAGS 110x11xx	Public Label Offset (2)
			Object (2)	Public Label Offset (2) / External Name (10)
LINK	09	n + 1	FLAGS 00100000	File Name (n)
GLINK	09	n + 1	FLAGS 00110000	File Name (n)

Numbers given under flags are in binary. X = varies depending on particular situation.

FLAGS FOR 03, 04, 07 RECORDS

7	0 = No public name is present (bit 4 = 0) 1 = Public name is present
6	0 = External name is not present 1 = External name is present (bits 3, 2, = 1, 1)
5	0 Reserved
4	0 = Address of public label is relocatable or not present, or this is an absolute file 1 = Address of public label in a relocatable file is absolute. (bit 7 = 1)
3	0 = No object present (bits, 1, 0 = 0, 1, 1) 1 = Object code is present
2	0 = Object is absolute or not present 1 = Object is relocatable (bit 3 = 1)
1	00 = Illegal combination 01 = Use only MSB of result (bit 3 = 1) 10 = Use only LSB of result (bit 3 = 1) 11 = Use both LSB and MSB of result (if bit 3 = 1) or object not present (if bit 3 = 0)
0	

If object is absolute (bit 2 = 0) Result = object
If object is relocatable (bit 2 = 1)
Result = object + PSECTS origin (if bit 6 = 0)
or Result = object + External name value (if bit 6 = 1)

FLAGS for 05/06 Records

7	1	= File contains relocatable object
6	1	= file contains externals
5	0	Reserved
4	1	= File contains public records
3	1	= File contains a link or glink file name
2	0	Reserved
1	0	Reserved
0	0	Reserved

Appendix E/ MODEL I ALTRAN

The MODEL I ALTRAN diskette in this package contains a NEW version of TRSDOS which is not compatible with OLD versions of Model I TRSDOS, see below for further details.

UPGRADE UTILITY ON TRSDOS 2.3B

OLD Model I TRSDOS diskettes to be used under the NEW Model I TRSDOS MUST be UPGRADEd before use. Once UPGRADEd, a system or data diskette becomes a NEW Model I TRSDOS data diskette.

OLD diskettes used under NEW TRSDOS without UPGRADEing, may cause extraneous information to be read at the end of files, giving a false End Of File (EOF) indication. Some programs will not function properly under these conditions.

NEW diskettes used under OLD TRSDOS, may not access all data and/or NEW programs may not run correctly.

If you determine that you need to use the UPGRADE utility, see page titled "TIPS ON USING THE MODEL I TRSDOS 2.3B UPGRADE UTILITY" contained in this addendix.

NOTE: When changing from one TRSDOS to the other you must use the RESET switch each time the diskette in drive 0 is changed.

RADIO SHACK APPLICATION PROGRAMS WHICH WERE DELIVERED ON AN OLD TRSDOS DISKETTE SHOULD NOT BE UPGRADED.

OLD:	TRSDOS 2.1, 2.2, and 2.3.
NEW:	TRSDOS 2.3B.
file:	A collection of information stored as one named unit in the directory.
program:	A file which causes the computer to perform a function.
data:	Information contained in a file which is used by a program.
system diskette:	A diskette containing TRSDOS. When this diskette is placed in drive 0 and the RESET switch is pressed, TRSDOS will begin to run.

data diskette:	A diskette which does not contain TRSDOS. If this diskette is placed in drive 0 and the RESET switch is pressed, the screen will clear and "NO SYSTEM" will be displayed.
UPGRADE:	A program contained on the TRSDOS 2.3B diskette.

DIFFERENCES BETWEEN MODEL I TRSDOS 2.3B and TRSDOS 2.3

Differences between TRSDOS 2.3B and TRSDOS 2.3 are:

1. Variable length records have been corrected, in all aspects.

2. In most cases, your computer will not "hang up" when you attempt use of a device which is not connected and powered up.

3. The DEVICE command has been deleted.

4. The following commands have been added:

   CLS
   This command clears the display and puts it in the 64-character mode.

   PATCH 'filespec' (ADD=aaaa,FIND=bb,CHG=cc)
   This command lets you make a change to a program file. You need to specify:

   - 'aaaa' - a four byte hexadecimal address specifying the memory location of the data you want to change
   - 'bb' - the contents of the byte you want to find and change. You can specify the contents of more than one byte.
   - 'cc' - the new contents to replace 'bb'

   For example:
   PATCH DUMMY/CMD (ADD=4567,FIND=CD3300,CHG=CD3B00)
   changes CD3300, which resides at memory location 4567 (HEX) in the file named DUMMY/CMD, to CD3B00.

If this command gives you a STRING NOT FOUND error message, this means that either 'bb' does not exist, or else 'bb' crosses a sector boundary. If 'bb' crosses a sector boundary, you must patch your file one byte at a time. For example:

```
PATCH DUMMY/CMD (ADD=4568,FIND=33,CHG=3B)
```

replaces the contents of the second byte in the above example.

TAPE (S=source device,D=destination device)
This command transfers Z-80 machine-language programs from one device to the other. You must specify the 'source device' and 'destination device' using these abbreviations:

T - Tape
D - Disk
R - RAM (Memory)

The only valid entries of this command are:

```
TAPE (S=T,D=D)   TAPE (S=T,D=R)   TAPE (S=D,D=T)
```

For example

```
TAPE (S=D,D=T)
```

starts a disk-to-tape transfer. TRSDOS will prompt you for the diskette file specification and ask you to press <ENTER> when the cassette recorder is ready for recording.

CAUTION: When doing a tape-to-RAM transfer, do not use a loading address below 6000 (Hex), since this would write over TRSDOS or the tape command.

5. These commands have been slightly changed:

BACKUP now checks to see if the diskette which will be your backup copy is already formatted. If it is, BACKUP will ask you if you want to REFORMAT it.

CLOCK will no longer increment the date when the time goes beyond 23:59:59.

COPY now works with only one-drive. For example:

```
COPY FILE1:0 to FILE3:0
```

duplicates the contents of FILE1 to a file named FILE3 on the same diskette.

KILL will now allow you to kill a protected file without knowing its UPDATE or protection level. To kill this kind of file, type an exclamation mark (!) at the end of the KILL command. For example:

```
KILL EXAMPLE !
```

kills the UPDATEd or protected file named EXAMPLE. (Note the mandatory space between the file name and the exclamation mark.)

LIST only lists the printable ASCII characters.

PROT no longer allows you to use the UNLOCK parameter.

DIR is now in this format:

```
Disk Name: TRSDOS      Drive: 0      04/15/81
Filename       Attrb   LRL   #Rec   #Grn   #Ext   EOF
JOBFILE/BLD     N*X0   256      1      1      1     1
TERMINAL/V1     N*X0   256      5      2      1   126
LOADX/CMD       N*X0   256      5      2      1     0
*** 171 Free Granules ***
```

---

1. Disk name is the name which was assigned to the disk when it was formatted.
2. File Name is the name and extension which was assigned to the file when it was created. The password (if any) is not shown.
3. Attributes is a four-character field:
   a. the first character is either I (Invisible file) or N (Non-invisable file)
   b. the second character is S (System file) or * (User file)
   c. the third character is the password protection status of the file:
      - X - the file is unprotected (no password)
      - A - the file has an access word but no update word
      - U - the file has an update word but no access word
      - B - the file has both update and access word

d. the fourth character specifies the level of access assigned to the access word:
   0 - total access
   1 - kill the file and everything listed below
   2 - rename the file and everything listed below
   3 - this designation is not used
   4 - write and everything listed below
   5 - read and everything listed below
   6 - execute only
   7 - no access

4. Number of Free Granules - how many free granules remain on the diskette.
5. Logical Record Length - the record length which was assigned to the file when it was created.
6. Number of Records - how many logical records have been written.
7. Number of Granules - how many granules have been used in that particular file.
8. Number of Extents - how many segments (contiguous blocks of up to 32 granules) of disk space are allocated to the file.
9. End of File (EOF) - shows the last byte number of the file.

TIPS ON USING THE MODEL I TRSDOS 2.3B UPGRADE UTILITY

If you determine that you need to use the UPGRADE utility then proceed as indicated below.

Insert your TRSDOS 2.3B system diskette in drive 0, press the RESET switch, and when TRSDOS READY is displayed type UPGRADE <ENTER>. Your screen will display:

```
TRSDOS DIRECTORY UPGRADE UTILITY

FOR CONVERSION OF TRSDOS 2.1, 2.2, OR 2.3 TO
TRSDOS 2.3B DIRECTORY FORMAT.

ONCE UPGRADE HAS BEEN EXECUTED, YOUR DISKETTE SHOULD
NOT BE USED UNDER TRSDOS 2.1, 2.2, OR 2.3 AGAIN.
```

DO YOU WISH TO CONTINUE (Y/N/Q)?

This means that the directory format on your TRSDOS 2.1, 2.2, or 2.3 diskette will be converted to the TRSDOS 2.3B format. Once you type Y to continue, the screen will display:

INSERT DISKETTE TO BE UPGRADED IN DRIVE 1.
PRESS <ENTER> WHEN READY.

Insert the diskette you want to convert in drive 1 and press <ENTER>. After successful conversion, the screen will display a CONVERSION COMPLETE message. If you are attempting to convert a diskette which has already been converted, the screen will display a DISKETTE IS ALREADY A 2.3B error message.

TECHNICAL NOTE

For all files indicated in the directory that have an End Of FIle (EOF) not equal to zero, UPGRADE will change the number of records to be one less than the previous record count. Note that in FILE1, the number of records indicated has been changed from 10 to 9 after UPGRADE. For FILE2 the records indicated remain the same since EOF=0.

BEFORE UPGRADE TRSDOS 2.1, 2.2, 2.3	AFTER UPGRADE TRSDOS 2.3B
FILE1 EOF=9 10 RECORDS	9 RECORDS
FILE2 EOF=0 10 RECORDS	10 RECORDS

If the TRSDOS 2.1, 2.2, or 2.3 diskette is a system diskette, part of the conversion process will prohibit accidental usage under the TRSDOS 2.1, 2.2, or 2.3 by killing the files listed below:

SYS0/SYS	SYS1/SYS	SYS2/SYS
SYS3/SYS	SYS4/SYS	SYS5/SYS
SYS6/SYS	FORMAT/CMD	BACKUP/CMD
BASICR/CMD	BASIC/CMD	

TIPS ON GETTING OBJECT FILES FROM TRSDOS 2.3B ONTO TRSDOS 2.1, 2.2, OR 2.3 DISKETTES

As an alternative to UPGRADEing your entire diskette, you can use these procedures to move an object file from 2.3B to 2.1, 2.2 or 2.3:

1) Insert your TRSDOS 2.3B system diskette that contains the object file in drive 0. Press the RESET switch.

2) At TRSDOS READY enter the command:
   LOAD object filespec

3) Remove your TRSDOS 2.3B diskette.

4) Insert your TRSDOS 2.3 diskette in drive 0 and press the RESET switch.

5) At TRSDOS READY enter the command:
   DUMP object filespec (START=start address,
   END=end address,TRA=transfer address)

6) The object file on this diskette may now be used as needed under TRSDOS 2.1, 2.2, or 2.3.

IMPORTANT NOTES

Please note the following concerning communications with Model I ALTRAN:

1. Radio Shack Application programs on TRSDOS 1.1, 1.2, 2.1, 2.2, or 2.3 were tested on the particular version of TRSDOS for which they were purchased.

   No guarantee is implied that these programs will work correctly after being UPDATEd to MODEL I TRSDOS 2.3B.

2. If you are transferring from Model I to Model I, both Model I's must be running under TRSDOS 2.3B.

3. You cannot run BASIC programs because TRSDOS 2.3B does not contain DISK BASIC.

Following is a listing of object codes in numerical order in column two followed by the nmemonic or source statement in column four.

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0000	00	1	NOP
0001	018405	2	LD BC,NN
0004	02	3	LD (BC),A
0005	03	4	INC BC
0006	04	5	INC B
0007	05	6	DEC B
0008	0620	7	LD B,N
000A	07	8	RLCA
000B	08	9	EX AF,AF'
000C	09	10	ADD HL,BC
000D	0A	11	LD A,(BC)
000E	0B	12	DEC BC
000F	0C	13	INC C
0010	OD	14	DEC C
0011	0E20	15	LD C,N
0013	0F	16	RRCA
0014	102E	17	DJNZ DIS
0016	118405	18	LD DE,NN
0019	12	19	LD (DE),A
001A	13	20	INC DE
001B	14	21	INC D
001C	15	22	DEC D
001D	1620	23	LD D,N
001F	17	24	RLA
0020	182E	25	JR DIS
0022	19	26	ADD HL,DE
0023	1A	27	LD A,(DE)
0024	1B	28	DEC DE
0025	1C	29	INC E
0026	ID	30	DEC E
0027	1E20	31	LD E,N
0029	1F	32	RRA
002A	202E	33	JR NZ,DIS
002C	218405	34	LD HL,NN
002F	228405	35	LD (NN),HL
0032	23	36	INC HL
0033	24	37	INC H
0034	25	38	DEC H
0035	2620	39	LD H,N
0037	27	40	DAA
0038	282E	41	JR Z,DIS
003A	29	42	ADD HL,HL
003B	2A8405	43	LD HL,(NN)
003E	2B	44	DEC HL
003F	2C	45	INC L
0040	2D	46	DEC L
0041	2E20	47	LD L,N
0043	2F	48	CPL
0044	302E	49	JR NC,DIS
0046	318405	50	LD SP,NN
0049	328405	51	LD (NN),A
004C	33	52	INC SP
004D	34	53	INC (HL)
004E	35	54	DEC (HL)
004F	3620	55	LD (HL),N
0051	37	56	SCF
0052	382E	57	JR C,DIS
0054	39	58	ADD HL,SP
0055	3A8405	59	LD A,(NN)
0058	3B	60	DEC SP
0059	3C	61	INC A
005A	3D	62	DEC A
005B	3E20	63	LD A,N
005D	3F	64	CCF
005E	40	65	LD B,B
005F	41	66	LD B,C
0060	42	67	LD B,D
0061	43	68	LD B,E
0062	44	69	LD B,H(NN)
0063	45	70	LD B,L
0064	46	71	LD B,(HL)
0065	47	72	LD B,A
0066	48	73	LD C,B
0067	49	74	LD C,C
0068	4A	75	LD C,D
0069	4B	76	LD C,E
006A	4C	77	LD C,H
006B	4D	78	LD C,L
006C	4E	79	LD C,(HL)
006D	4F	80	LD C,A
006E	50	81	LD D,B
006F	51	82	LD D,C
0070	52	83	LD D,D
0071	53	84	LD D,E
0072	54	85	LD D,H
0073	55	86	LD D,L
0074	56	87	LD D,(HL)
0075	57	88	LD D,A
0076	58	89	LD E,B
0077	59	90	LD E,C
0078	5A	91	LD E,D
0079	5B	92	LD E,E
007A	5C	93	LD E,H
007B	5D	94	LD E,L
007C	5E	95	LD E,(HL)
007D	5F	96	LD E,A
007E	60	97	LD H,B
007F	61	98	LD H,C
0080	62	99	LD H,D
0081	63	100	LD H,E
0082	64	101	LD H,H
0083	65	102	LD H,L
0084	66	103	LD H,(HL)
0085	67	104	LD H,A
0086	68	105	LD L,B
0087	69	106	LD L,C

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0088	6A	107	LD L,D
0089	6B	108	LD L,E
008A	6C	109	LD L,H
008B	6D	110	LD L,L
008C	6E	111	LD L,(HL)
008D	6F	112	LD L,A
008E	70	113	LD (HL),B
008F	71	114	LD (HL),C
0090	72	115	LD (HL),D
0091	73	116	LD (HL),E
0092	74	117	LD (HL),H
0093	75	118	LD (HL),L
0094	76	119	HALT
0095	77	120	LD (HL),A
0096	78	121	LD A,B
0097	79	122	LD A,C
0098	7A	123	LD A,D
0099	7B	124	LD A,E
009A	7C	125	LD A,H
009B	7D	126	LD A,L
009C	7E	127	LD A,(HL)
009D	7F	128	LD A,A
009E	80	129	ADD A,B
009F	81	130	ADD A,C
00A0	82	131	ADD A,D
00A1	83	132	ADD A,E
00A2	84	133	ADD A,H
00A3	85	134	ADD A,L
00A4	86	135	ADD A,(HL)
00A5	87	136	ADD A,A
00A6	88	137	ADC A,B
00A7	89	138	ADC A,C
00A8	8A	139	ADC A,D
00A9	8B	140	ADC A,E
00AA	8C	141	ADC A,H
00AB	8D	142	ADC A,L
00AC	8E	143	ADC A,(HL)
00AD	8F	144	ADC A,A
00AE	90	145	SUB B
00AF	91	146	SUB C
00B0	92	147	SUB D
00B1	93	148	SUB E
00B2	94	149	SUB H
00B3	95	150	SUB L
00B4	96	151	SUB (HL)
00B5	97	152	SUB A
00B6	98	153	SBC A,B
00B7	99	154	SBC A,C
00B8	9A	155	SBC A,D
00B9	9B	156	SBC A,E
00BA	9C	157	SBC A,H
00BB	9D	158	SBC A,L
00BC	9E	159	SBC A,(HL)
00BD	9F	160	SBC A,A
00BE	A0	161	AND B
00BF	A1	162	AND C
00C0	A2	163	AND D
00C1	A3	164	AND E
00C2	A4	165	AND H
00C3	A5	166	AND L
00C4	A6	167	AND (HL)
00C5	A7	168	AND A
00C6	A8	169	XOR B
00C7	A9	170	XOR C
00C8	AA	171	XOR D
00C9	AB	172	XOR E
00CA	AC	173	XOR H
00CB	AD	174	XOR L
00CC	AE	175	XOR (HL)
00CD	AF	176	XOR A
00CE	B0	177	OR B
00CF	B1	178	OR C
00D0	B2	179	OR D
00D1	B3	180	OR E
0002	B4	181	OR H
00D3	B5	182	OR L
00D4	B6	183	OR (HL)
00D5	B7	184	OR A
00D6	B8	185	CP B
00D7	B9	186	CP C
00D8	BA	187	CP D
00D9	BB	188	CP E
00DA	BC	189	CP H
00DB	BD	190	CP L
00DC	BE	191	CP (HL)
00DD	BF	192	CP A
00DE	C0	193	RET NZ
00DF	C1	194	POP BC
00E0	C28405	195	JP NZ, NN
00E3	C38405	196	JP NN
00E6	C48405	197	CALL NZ,NN
00E9	C5	198	PUSH BC
00EA	C620	199	ADD A,N
00EC	C7	200	RST 0
00ED	C8	201	RET Z
00EE	C9	202	RET
00EF	CA8405	203	JP Z,NN
00F2	CC8405	204	CALL Z,NN
00F5	CD8405	205	CALL NN
00F8	CE20	206	ADC A,N
00FA	CF	207	RST 8
00FB	D0	208	RET NC
00FC	D1	209	POP DE
00FD	D28405	210	JP NC,NN
0100	D320	211	OUT ,NA
0102	D48405	212	CALL NC,NN
0105	D5	213	PUSH DE
0106	D620	214	SUB N
0108	D7	215	RST 10H
0109	D8	216	RET C
010A	D9	217	EXX
010B	DA8405	218	JP C,NN
010E	DB20	219	IN A,N
0110	DC8405	220	CALL C,NN
0113	DE20	221	SBC A,N
0115	DF	222	RST 18H
0116	E0	223	RET PO
0117	E1	224	POP HL
0118	E28405	225	JP PO,NN
011B	E3	226	EX (SP),HL
011C	E48405	227	CALL PO,NN
011F	E5	228	PUSH HL

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0120	E620	229	AND N
0122	E7	230	RST 20H
0123	E8	231	RET PE
0124	E9	232	JP (HL)
0125	EA8405	233	JP PE,NN
0128	EB	234	EX DE,HL
0129	EC8405	235	CALL PE,NN
012C	EE20	236	XOR N
012E	EF	237	RST 28H
012F	F0	238	RET P
0130	F1	239	POP AF
0131	F28405	240	JP P,NN
0134	F3	241	DI
0135	F48405	242	CALL P,NN
0138	F5	243	PUSH AF
0139	F620	244	OR N
013B	F7	245	RST 30H
013C	F8	246	RET M
013D	F9	247	LD SP,HL
013E	FA8405	248	JP M,NN
0141	FB	249	EI
0142	FC8405	250	CALL M,NN
0145	FE20	251	CP N
0147	FF	252	RST 38H
0148	CB00	253	RLC B
014A	CB01	254	RLC C
014C	CB02	255	RLC D
014E	CB03	256	RLC E
0150	CB04	257	RLC H
0152	CB05	258	RLC L
0154	CB06	259	RLC (HL)
0156	CB07	260	RLC A
0158	CB08	261	RRC B
015A	CB09	262	RRC C
015C	CB0A	263	RRC D
015E	CB0B	264	RRC E
0160	CB0C	265	RRC H
0162	CB0D	266	RRC L
0164	CB0E	267	RRC (HL)
0166	CB0F	268	RRC A
0168	CB10	269	RL B
016A	CB11	270	RL C
016C	CB12	271	RL D
016E	CB13	272	RL E
0170	CB14	273	RL H
0172	CB15	274	RL L
0174	CB16	275	RL (HL)
0176	CB17	276	RL A
0178	CB18	277	RR B
017A	CB19	278	RR C
017C	CB1A	279	RR D
017E	CB1B	280	RR E
0180	CB1C	281	RR H
0182	CB1D	282	RR L
0184	CB1E	283	RR (HL)
0186	CB1F	284	RR A
0188	CB20	285	SLA B
018A	CB21	286	SLA C
018C	CB22	287	SLA D
018E	CB23	288	SLA E
0190	CB24	289	SLA H
0192	CB25	290	SLA L
0194	CB26	291	SLA (HL)
0196	CB27	292	SLA A
0198	CB28	293	SRA B
019A	CB29	294	SRA C
019C	CB2A	295	SRA D
019E	CB2B	296	SRA E
01A0	CB2C	297	SRA H
01A2	CB2D	298	SRA L
01A4	CB2E	299	SRA (HL)
01A6	CB2F	300	SRA A
01A8	CB38	301	SRL B
01AA	CB39	302	SRL C
01AC	CB3A	303	SRL D
01AE	CB3B	304	SRL E
01B0	CB3C	305	SRL H
01B2	CB3D	306	SRL L
01B4	CB3E	307	SRL (HL)
01B6	CB3F	308	SRL A
01B8	CB40	309	BIT 0,B
01BA	CB41	310	BIT 0,C
01BC	CB42	311	BIT 0,D
01BE	CB43	312	BIT 0,E
01C0	CB44	313	BIT 0,H
01C2	CB45	314	BIT 0,L
01C4	CB46	315	BIT 0,(HL)
01C6	CB47	316	BIT 0,A
01C8	CB48	317	BIT 1,B
01CA	CB49	318	BIT 1,C
01CC	CB4A	319	BIT 1,D
01CE	CB4B	320	BIT 1,E
01D0	CB4C	321	BIT 1,H
01D2	CB4D	322	BIT 1,L
01D4	CB4E	323	BIT 1,(HL)
01D6	CB4F	324	BIT 1,A
01D8	CB50	325	BIT 2,B
01DA	CB51	326	BIT 2,C
01DC	CB52	327	BIT 2,D
01DE	CB53	328	BIT 2,E
01E0	CB54	329	BIT 2,H
01E2	CB55	330	BIT 2,L
01E4	CB56	331	BIT 2,(HL)
01E6	CB57	332	BIT 2,A
01E8	CB58	333	BIT 3,B
01EA	CB59	334	BIT 3,C
01EC	CB5A	335	BIT 3,D
01EE	CB5B	336	BIT 3,E
01F0	CB5C	337	BIT 3,H
01F2	CB5D	338	BIT 3,L
01F4	CB5E	339	BIT 3,(HL)
01F6	CB5F	340	BIT 3,A
01F8	CB60	341	BIT 4,B
01FA	CB61	342	BIT 4,C
01FC	CB62	343	BIT 4,D
01FE	CB63	344	BIT 4,E
0200	CB64	345	BIT 4,H
0202	CB65	346	BIT 4,L
0204	CB66	347	BIT 4,(HL)
0206	CB67	348	BIT 4,A
0208	CB68	349	BIT 5,B
020A	CB69	350	BIT 5,C

LOC	OBJ CODE	STMT	SOURCE STATEMENT
020C	CB6A	351	BIT 5,D
020E	CB6B	352	BIT 5,E
0210	CB6C	353	BIT 5,H
0212	CB6D	354	BIT 5,L
0214	CB6E	355	BIT 5,(HL)
0216	CB6F	356	BIT 5,A
0218	CB70	357	BIT 6,B
021A	CB71	358	BIT 6,C
021C	CB72	359	BIT 6,D
021E	CB73	360	BIT 6,E
0220	CB74	361	BIT 6,H
0222	CB75	362	BIT 6,L
0224	CB76	363	BIT 6,(HL)
0226	CB77	364	BIT 6,A
0228	CB78	365	BIT 7,B
022A	CB79	366	BIT 7,C
022C	CB7A	367	BIT 7,D
022E	CB7B	368	BIT 7,E
0230	CB7C	369	BIT 7,H
0232	CB7D	370	BIT 7,L
0234	CB7E	371	BIT 7,(HL)
0236	CB7F	372	BIT 7,A
0238	CB80	373	RES 0,B
023A	CB81	374	RES 0,C
023C	CB82	375	RES 0,D
023E	CB83	376	RES 0,E
0240	CB84	377	RES 0,H
0242	CB85	378	RES 0,L
0244	CB86	379	RES 0,(HL)
0246	CB87	380	RES 0,A
0248	CB88	381	RES 1,B
024A	CB89	382	RES 1,C
024C	CB8A	383	RES 1,D
024E	CB8B	384	RES 1,E
0250	CB8C	385	RES 1,H
0252	CB8D	386	RES 1,L
0254	CB8E	387	RES 1,(HL)
0256	CB8F	388	RES 1,A
0258	CB90	389	RES 2,B
025A	CB91	390	RES 2,C
025C	CB92	391	RES 2,D
025E	CB93	392	RES 2,E
0260	CB94	393	RES 2,H
0262	CB95	394	RES 2,L
0264	CB96	395	RES 2,(HL)
0266	CB97	396	RES 2,A
0268	CB98	397	RES 3,B
026A	CB99	398	RES 3,C
026C	CB9A	399	RES 3,D
026E	CB9B	400	RES 3,E
0270	CB9C	401	RES 3,H
0272	CB9D	402	RES 3,L
0274	CB9E	403	RES 3,(HL)
0276	CB9F	404	RES 3,A
0278	CBA0	405	RES 4,B
027A	CBA1	406	RES 4,C
027C	CBA2	407	RES 4,D
027E	CBA3	408	RES 4,E
0280	CBA4	409	RES 4,H
0282	CBA5	410	RES 4,L
0284	CBA6	411	RES 4,(HL)
0286	CBA7	412	RES 4,A
0288	CBA8	413	RES 5,B
028A	CBA9	414	RES 5,C
028C	CBAA	415	RES 5,D
028E	CBAB	416	RES 5,E
0290	CBAC	417	RES 5,H
0292	CBAD	418	RES 5,L
0294	CBAE	419	RES 5,(HL)
0296	CBAF	420	RES 5,A
0298	CBB0	421	RES 6,B
029A	CBB1	422	RES 6,C
029C	CBB2	423	RES 6,D
029E	CBB3	424	RES 6,E
02A0	CBB4	425	RES 6,H
02A2	CBB5	426	RES 6,L
02A4	CBB6	427	RES 6,(HL)
02A6	CBB7	428	RES 6,A
02A8	CBB8	429	RES 7,B
02AA	CBB9	430	RES 7,C
02AC	CBBA	431	RES 7,D
02AE	CBBB	432	RES 7,E
0280	CBBC	433	RES 7,H
0282	CBBD	434	RES 7,L
0284	CBBE	435	RES 7,(HL)
0286	CBBF	436	RES 7,A
0288	CBC0	437	SET 0,B
02BA	CBC1	438	SET 0,C
02BC	CBC2	439	SET 0,D
02BE	CBC3	440	SET 0,E
02C0	CBC4	441	SET 0,H
02C2	CBC5	442	SET 0,L
02C4	CBC6	443	SET 0,(HL)
02C6	CBC7	444	SET 0,A
02C8	CBC8	445	SET 1,B
02CA	CBC9	446	SET 1,C
02CC	CBCA	447	SET 1,D
02CE	CBCB	448	SET 1,E
02D0	CBCC	449	SET 1,H
02D2	CBCD	450	SET 1,L
02D4	CBCE	451	SET 1,(HL)
02D6	CBCF	452	SET 1,A
02D8	CBD0	453	SET 2,B
02DA	CBD1	454	SET 2,C
02DC	CBD2	455	SET 2,D
02DE	CBD3	456	SET 2,E
02E0	CBD4	457	SET 2,H
02E2	CBD5	458	SET 2,L
02E4	CBD6	459	SET 2,(HL)
02E6	CBD7	460	SET 2,A
02E8	CBD8	461	SET 3,B
02EA	CBD9	462	SET 3,C
02EC	CBDA	463	SET 3,D
02EE	CBDB	464	SET 3,E
02F0	CBDC	465	SET 3,H
02F2	CBDD	466	SET 3,L
02F4	CBDE	467	SET 3,(HL)
02F6	CBDF	468	SET 3,A
02F8	CBE0	469	SET 4,B
02FA	CBE1	470	SET 4,C
02FC	CBE2	471	SET 4,D
02FE	CBE3	472	SET 4,E

LOC	OBJ CODE	STMT	SOURCE STATEMENT
0300	CBE4	473	SET 4,H
0302	CBE5	474	SET 4,L
0304	CBE6	475	SET 4,(HL)
0306	CBE7	476	SET 4,A
0308	CBE8	477	SET 5,B
030A	CBE9	478	SET 5,C
030C	CBEA	479	SET 5,D
030E	CBEB	480	SET 5,E
0310	CBEC	481	SET 5,H
0312	CBED	482	SET 5,L
0314	CBEE	483	SET 5,(HL)
0316	CBEF	484	SET 5,A
0318	CBF0	485	SET 6,B
031A	CBF1	486	SET 6,C
031C	CBF2	487	SET 6,D
031E	CBF3	488	SET 6,E
0320	CBF4	489	SET 6,H
0322	CBF5	490	SET 6,L
0324	CBF6	491	SET 6,(HL)
0326	CBF7	492	SET 6,A
0328	CBF8	493	SET 7,B
032A	CBF9	494	SET 7,C
032C	CBFA	495	SET 7,D
032E	CBFB	496	SET 7,E
0330	CBFC	497	SET 7,H
0332	CBFD	498	SET 7,L
0334	CBFE	499	SET 7,(HL)
0336	CBFF	500	SET 7,A
0338	DD09	501	ADD IX,BC
033A	DD19	502	ADD IX,DE
033C	DD218405	503	LD IX,NN
0340	DD228405	504	LD (NN),IX
0344	DD23	505	INC IX
0346	DD29	506	ADD IX,IX
0348	DD2A8405	507	LD IX,(NN)
034C	DD2B	508	DEC IX
034E	DD3405	509	INC (IX + IND)
0351	DD3505	510	DEC (IX + IND)
0354	DD360520	511	LD (IX + IND),N
0358	DD39	512	ADD IX,SP
035A	DD4605	513	LD B,(IX + IND)
035D	DD4E05	514	LD C,(IX + IND)
0360	DD5605	515	LD D,(IX + IND)
0363	DD5E05	516	LD E,(IX + IND)
0366	DD6605	517	LD H,(IX + IND)
0369	DD6E05	518	LD L,(IX + IND)
036C	DD7005	519	LD (IX + IND),B
036F	DD7105	520	LD (IX + IND),C
0372	DD7205	521	LD (IX + IND),D
0375	DD7305	522	LD (IX + IND),E
0378	DD7405	523	LD (IX + IND),H
037B	DD7505	524	LD (IX + IND),L
037E	DD7705	525	LD (IX + IND),A
0381	DD7E05	526	LD A,(IX + IND)
0384	DD8605	527	ADD A,(IX + IND)
0387	DD8E05	528	ADC A,(IX + IND)
038A	DD9605	529	SUB (IX + IND)
038D	DD9E05	530	SBC A,(IX + IND)
0390	DDA605	531	AND (IX + IND)
0393	DDAE05	532	XOR (IX + IND)
0396	DDB605	533	OR (IX + IND)
0399	DDBE05	534	CP (IX + IND)
039C	DDE1	535	POP IX
039E	DDE3	536	EX (SP),IX
03A0	DDE5	537	PUSH IX
03A2	DDE9	538	JP (IX)
03A4	DDF9	539	LD SP,IX
03A6	DDCB0506	540	RLC (IX + IND)
03AA	DDCB050E	541	RRC (IX + IND)
03AE	DDCB0516	542	RL (IX + IND)
03B2	DDCB051E	543	RR (IX + IND)
03B6	DDCB0526	544	SLA (IX + IND)
03BA	DDCB052E	545	SRA (IX + IND)
03BE	DDCB053E	546	SRL (IX + IND)
03C2	DDCB0546	547	BIT 0,(IX + IND)
03C6	DDCB054E	548	BIT 1,(IX + IND)
03CA	DDCB0556	549	BIT 2,(IX + IND)
03CE	DDCB055E	550	BIT 3,(IX + IND)
03D2	DDCB0566	551	BIT 4,(IX + IND)
03D6	DDCB056E	552	BIT 5,(IX + IND)
03DA	DDCB0576	553	BIT 6,(IX + IND)
03DE	DDCB057E	554	BIT 7,(IX + IND)
03E2	DDCB0586	555	RES 0,(IX + IND)
03E6	DDCB058E	556	RES 1,(IX + IND)
03EA	DDCB0596	557	RES 2,(IX + IND)
03EE	DDCB059E	558	RES 3,(IX + IND)
03F2	DDCB05A6	559	RES 4,(IX + IND)
03F6	DDCB05AE	560	RES 5,(IX + IND)
03FA	DDCB05B6	561	RES 6,(IX + IND)
03FE	DDCB05BE	562	RES 7,(IX + IND)
0402	DDCB05C6	563	SET 0,(IX + IND)
0406	DDCB05CE	564	SET 1,(IX + IND)
040A	DDCB05D6	565	SET 2,(IX + IND)
040E	DDCB05DE	566	SET 3,(IX + IND)
0412	DDCB05E6	567	SET 4,(IX + IND)
0416	DDCB05EE	568	SET 5,(IX + IND)
041A	DDCB05F6	569	SET 6,(IX + IND)
041E	DDCB05FE	570	SET 7,(IX + IND)
0422	ED40	571	IN B,(C)
0424	ED41	572	OUT (C),B
0426	ED42	573	SBC HL,BC
0428	ED438405	574	LD (NN),BC
042C	ED44	575	NEG
042E	ED45	576	RETN
0430	ED46	577	IM 0
0432	ED47	578	LD I,A
0434	ED48	579	IN C,(C)
0436	ED49	580	OUT (C),C
0438	ED4A	581	ADC HL,BC
043A	ED4B8405	582	LD BC,(NN)
043E	ED4D	583	RETI
	ED4F		LD R,A
	ED5F		LD A,R
0440	ED50	584	IN D,(C)
0442	ED51	585	OUT (C),D
0444	ED52	586	SBC HL,DE
0446	ED538405	587	LD (NN),DE
044A	ED56	588	IM I
044C	ED57	589	LD A,I
044E	ED58	590	IN E,(C)
0450	ED59	591	OUT (C),E
0452	ED5A	592	ADC HL,DE
0454	ED5B8405	593	LD DE,(NN)

LOC	OBJ CODE	STMT	SOURCE STATEMENT
045A	ED60	595	IN H,(C)
045C	ED61	596	OUT (C),H
045E	ED62	597	SBC HL,HL
0460	ED67	598	RRD
0462	ED68	599	IN L,(C)
0464	ED69	600	OUT (C),L
0466	ED6A	601	ADC HL,HL
0468	ED6F	602	RLD
046A	ED72	603	SBC HL,SP
046C	ED738405	604	LD (NN),SP
0470	ED78	605	IN A,(C)
0472	ED79	606	OUT (C),A
0474	ED7A	607	ADC HL,SP
0476	ED7B8405	608	LD SP,(NN)
047A	EDA0	609	LDI
047C	EDA1	610	CPI
047E	EDA2	611	INI
0480	EDA3	612	OUTI
0482	EDA8	613	LDD
0484	EDA9	614	CPD
0486	EDAA	615	IND
0488	EDAB	616	OUTD
048A	EDB0	617	LDIR
048C	EDB1	618	CPIR
048E	EDB2	619	INIR
0490	EDB3	620	OTIR
0492	EDB8	621	LDDR
0494	EDB9	622	CPDR
0496	EDBA	623	INDR
0498	EDBB	624	OTDR
049A	FD09	625	ADD IY,BC
049C	FD19	626	ADD IY,DE
049E	FD218405	627	LD IY,NN
04A2	FD228405	628	LD (NN),IY
04A6	FD23	629	INC IY
04A8	FD29	630	ADD IY,IY
04AA	FD2A8405	631	LD IY,(NN)
04AE	FD2B	632	DEC IY
04B0	FD3405	633	INC (IY + IND)
04B3	FD3505	634	DEC (IY + IND)
04B6	FD360520	635	LD (IY + IND),N
04BA	FD39	636	ADD IY,SP
04BC	FD4605	637	LD B,(IY + IND)
04BF	FD3E05	638	LD C,(IY + IND)
04C2	FD5605	639	LD D,(IY + IND)
04C5	FD5E05	640	LD E,(IY + IND)
04C8	FD6605	641	LD H,(IY + IND)
04CB	FD6E05	642	LD L,(IY + IND)
04CE	FD7005	643	LD (IY + IND),B
04D1	FD7105	644	LD (IY + IND),C
04D4	FD7205	645	LD(IY + IND),D
04D7	FD7305	646	LD (IY + IND),E
04DA	FD7405	647	LD (IY + IND),H
04DD	FD7505	648	LD (IY + IND),L
04E0	FD7705	649	LD (IY + IND),A
04E3	FD7E05	650	LD A,(IY + IND)
04E6	FD8605	651	ADD A,(IY + IND)
04E9	FD8E05	652	ADC A,(IY + IND)
04EC	FD9605	653	SUB-(IY + IND)
04EF	FD9E05	654	SBC A,(IY + IND)
04F2	FDA605	655	AND (IY + IND)
04F5	FDAE05	656	XOR (IY + IND)
04F8	FDB605	657	OR (IY + IND)
04FB	FDBE05	658	CP (IY + IND)
04FE	FDE1	659	POP IY
0500	FDE3	660	EX (SP),IY
0502	FDE5	661	PUSH IY
0504	FDE9	662	JP (IY)
0506	FDF9	663	LD SP,IY
0508	FDCB0506	664	RLC (IY + IND)
050C	FDCB050E	665	RRC (IY + IND)
0510	FDCB0516	666	RL (IY + IND)
0514	FDCB051E	667	RR (IY + IND)
0518	FDCB0526	668	SLA (IY + IND)
051C	FDCB052E	669	SRA (IY + IND)
0520	FDCB053E	670	SRL (IY + IND)
0524	FDCB0546	671	BIT 0,(IY + IND)
0528	FDCB054E	672	BIT 1,(IY + IND)
052C	FDCB0556	673	BIT 2,(IY + IND)
0530	FDCB055E	674	BIT 3,(IY + IND)
0534	FDCB0566	675	BIT 4,(IY + IND)
0538	FDCB056E	676	BIT 5,(IY + IND)
053C	FDCB0576	677	BIT 6,(IY + IND)
0540	FDCB057E	678	BIT 7,(IY + IND)
0544	FDCB0586	679	RES 0,(IY + IND)
0548	FDCB058E	680	RES 1,(IY + IND)
054C	FDCB0596	681	RES 2,(IY + IND)
0550	FDCB059E	682	RES 3,(IY + IND)
0554	FDCB05A6	683	RES 4,(IY + IND)
0558	FDCB05AE	684	RES 5,(IY + IND)
055C	FDCB05B6	685	RES 6,(IY + IND)
0560	FDCB05BE	686	RES 7,(IY + IND)
0564	FDCB05C6	687	SET 0,(IY + IND)
0568	FDCB05CE	688	SET 1,(IY + IND)
056C	FDCB05D6	689	SET 2,(IY + IND)
0570	FDCB05DE	690	SET 3,(IY + IND)
0574	FDCB05E6	691	SET 4,(IY + IND)
0578	FDCB05EE	692	SET 5,(IY + IND)
057C	FDCB05F6	693	SET 6,(IY + IND)
0580	FDCB05FE	694	SET 7,(IY + IND)
0584		695 NN	DEFS 2
		696 IND	EQU 5
		697 M	EQU 10H
		698 N	EQU 20H
		699 DIS	EQU 30H
		700	END

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
00DE	DDCB0576	111	BIT	6,(IX + IND)
00E2	FDCB0576	112	BIT	6,(IY + IND)
00E6	CB77	113	BIT	6,A
00E8	CB70	114	BIT	6,B
00EA	CB71	115	BIT	6,C
00EC	CB72	116	BIT	6,D
00EE	CB73	117	BIT	6,E
00F0	CB74	118	BIT	6,H
00F2	CB75	119	BIT	6,L
00F4	CB7E	120	BIT	7,(HL)
00F6	DDCB057E	121	BIT	7,(IX + IND)
00FA	FDCB057E	122	BIT	7,(IY + IND)
00FE	CB7F	123	BIT	7,A
0100	CB78	124	BIT	7,B
0102	CB79	125	BIT	7,C
0104	CB7A	126	BIT	7,D
0106	CB7B	127	BIT	7,E
0108	CB7C	128	BIT	7,H
010A	CB7D	129	BIT	7,L
010C	DC8405	130	CALL	C,NN
010F	FC8405	131	CALL	M,NN
0112	D48405	132	CALL	NC,NN
0115	CD8405	133	CALL	NN
0118	C48405	134	CALL	NZ,NN
011B	F48405	135	CALL	P,NN
011E	EC8405	136	CALL	PE,NN
0121	E48405	137	CALL	PO,NN
0124	CC8405	138	CALL	Z,NN
0127	3F	139	CCF	
0128	BE	140	CP	(HL)
0129	DDBE05	141	CP	(IX + IND)
012C	FDBE05	142	CP	(IY + IND)
012F	BF	143	CP	A
0130	B8	144	CP	B
0131	B9	145	CP	C
0132	BA	146	CP	D
0133	BB	147	CP	E
0134	BC	148	CP	H
0135	BD	149	CP	L
0136	FE20	150	CP	N
0138	EDA9	151	CPD	
013A	EDB9	152	CPDR	
013C	EDA1	153	CPI	
013E	EDB1	154	CPIR	
0140	2F	155	CPL	
0141	27	156	DAA	
0142	35	157	DEC	(HL)
0143	DD3505	158	DEC	(IX + IND)
0146	FD3505	159	DEC	(IY + IND)
0149	3D	160	DEC	A
014A	05	161	DEC	B
014B	0B	162	DEC	BC
014C	0D	163	DEC	C
014D	15	164	DEC	D
014E	1B	165	DEC	DE
014F	1D	166	DEC	E
0150	25	167	DEC	H
0151	2B	168	DEC	HL
0152	DD2B	169	DEC	IX
0154	FD2B	170	DEC	IY
0156	2D	171	DEC	L

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
0157	3B	172	DEC	SP
0158	F3	173	DI	
0159	102E	174	DJNZ	DIS
015B	FB	175	EI	
015C	E3	176	EX	(SP),HL
015D	DDE3	177	EX	(SP),IX
015F	FDE3	178	EX	(SP),IY
0161	08	179	EX	AF,AF'
0162	EB	180	EX	DE,HL
0163	D9	181	EXX	
0164	76	182	HALT	
0165	ED46	183	IM	0
0167	ED56	184	IM	1
0169	ED5E	185	IM	2
016B	ED78	186	IN	A,(C)
016D	DB20	187	IN	A,N
016F	ED40	188	IN	B,(C)
0171	ED48	189	IN	C,(C)
0173	ED50	190	IN	D,(C)
0175	ED58	191	IN	E,(C)
0177	ED60	192	IN	H,(C)
0179	ED68	193	IN	L,(C)
017B	34	194	INC	(HL)
017C	DD3405	195	INC	(IX + IND)
017F	FD3405	196	INC	(IY + IND)
0182	3C	197	INC	A
0183	04	198	INC	B
0184	03	199	INC	BC
0185	0C	200	INC	C
0186	14	201	INC	D
0187	13	202	INC	DE
0188	1C	203	INC	E
0189	24	204	INC	H
018A	23	205	INC	HL
018B	DD23	206	INC	IX
018D	FD23	207	INC	IY
018F	2C	208	INC	L
0190	33	209	INC	SP
0191	EDAA	210	IND	
0193	EDBA	211	INDR	
0195	EDA2	212	INI	
0197	EDB2	213	INIR	
0199	E9	214	JP	(HL)
019A	DDE9	215	JP	(IX)
019C	FDE9	216	JP	(IY)
019E	DA8405	217	JP	C,NN
01A1	FA8405	218	JP	M,NN
01A4	D28405	219	JP	NC,NN
01A7	C38405	220	JP	NN
01AA	C28405	221	JP	NZ,NN
01AD	F28405	222	JP	P,NN
01B0	EA8405	223	JP	PE,NN
01B3	E28405	224	JP	PO,NN
01B6	CA8405	225	JP	Z,NN
01B9	382E	226	JR	C,DIS
01BB	182E	227	JR	DIS
01BD	302E	228	JR	NC,DIS
01BF	202E	229	JR	NZ,DIS
01C1	282E	230	JR	Z,DIS
01C3	02	231	LD	(BC),A
01C4	12	232	LD	(DE),A

LOC	OBJ CODE	STMT	SOURCE STATEMENT
01C5	77	233	LD (HL),A
01C6	70	234	LD (HL),B
01C7	71	235	LD (HL),C
01C8	72	236	LD (HL),D
01C9	73	237	LD (HL),E
01CA	74	238	LD (HL),H
01CB	75	239	LD (HL),L
01CC	3620	240	LD (HL),N
01CE	DD7705	241	LD (IX + IND),A
01D1	DD7005	242	LD (IX + IND),B
01D4	DD7105	243	LD (IX + IND),C
01D7	DD7205	244	LD (IX + IND),,D
01DA	DD7305	245	LD (IX + IND),E
01DD	DD7405	246	LD (IX + IND),H
01E0	DD7505	247	LD (IX + IND),L
01E3	DD360520	248	LD (IX + IND),N
01E7	FD7705	249	LD (IY + IND),A
01EA	FD7005	250	LD (IY + IND),B
01ED	FD7105	251	LD (IY + IND),C
01F0	FD7205	252	LD (IY + IND),D
01F3	FD7305	253	LD (IY + IND),E
01F6	FD7405	254	LD (IY + IND),H
01F9	FD7505	255	LD (IY + IND),L
01FC	FD360520	256	LD (IY + IND),N
0200	328405	257	LD (NN),A
0203	ED438405	258	LD (NN),BC
0207	ED538405	259	LD (NN),DE
020B	228405	260	LD (NN),HL
020E	DD228405	261	LD (NN),IX
0202	FD228405	262	LD (NN),IY
0216	ED738405	263	LD (NN),SP
021A	0A	264	LD A,(BC)
021B	1A	265	LD A,(DE)
021C	7E	266	LD A,(HL)
021D	DD7E05	267	LD A,(IX + IND)
0220	FD7E05	268	LD A,(IY + IND)
0223	3A8405	269	LD A,(NN)
0226	7F	270	LD A,A
0227	78	271	LD A,B
0228	79	272	LD A,C
0229	7A	273	LD A,D
022A	7B	274	LD A,E
022B	7C	275	LD A,H
022C	ED57	276	LD A,I
022E	7D	277	LD A,L
022F	3E20	278	LD A,N
0231	46	279	LD B,(HL)
0232	DD4605	280	LD B,(IX + IND)
0235	FD4605	281	LD B,(IY + IND)
0238	47	282	LD B,A
0239	40	283	LD B,B
023A	41	284	LD B,C
023B	42	285	LD B,D
023C	43	286	LD B,E
023D	44	287	LD B,H
023E	45	288	LD B,L
023F	0620	289	LD B,N
0241	ED4B8405	290	LD BC,(NN)
0245	018405	291	LD BC,NN
0248	4E	292	LD C,(HL)
0249	DD4E05	293	LD C,(IX + IND)
024C	FD4E05	294	LD C,(IY + IND)
024F	4F	295	LD C,A
0250	48	296	LD C,B
0251	49	297	LD C,C
0252	4A	298	LD C,D
0253	4B	299	LD C,E
0254	4C	300	LD C,H
0255	4D	301	LD C,L
0256	0E20	302	LD C,N
0258	56	303	LD D,(HL)
0259	DD5605	304	LD D,(IX + IND)
025C	FD5605	305	LD D,(IY + IND)
025F	57	306	LD D,A
0260	50	307	LD D,B
0261	51	308	LD D,C
0262	52	309	LD D,D
0263	53	310	LD D,E
0264	54	311	LD D,H
0265	55	312	LD D,L
0266	1620	313	LD D,N
0268	ED5B8405	314	LD DE,(NN)
026C	118405	315	LD DE,NN
026F	5E	316	LD E,(HL)
0270	DD5E05	317	LD E,(IX + IND)
0273	FD5E05	318	LD E,(IY + IND)
0276	5F	319	LD E,A
0277	58	320	LD E,B
0278	59	321	LD E,C
0279	5A	322	LD E,D
027A	5B	323	LD E,E
027B	5C	324	LD E,H
027C	5D	325	LD E,L
027D	1E20	326	LD E,N
027F	66	327	LD H,(HL)
0280	DD6605	328	LD H,(IX + IND)
0283	FD6605	329	LD H,(IY + IND)
0286	67	330	LD H,A
0287	60	331	LD H,B
0288	61	332	LD H,C
0289	62	333	LD H,D
028A	63	334	LD H,E
028B	64	335	LD H,H
028C	65	336	LD H,L
028D	2620	337	LD H,N
028F	2A8405	338	LD HL,(NN)
0292	218405	339	LD HL,NN
0295	ED47	340	LD I,A
0297	DD2A8405	341	LD IX,(NN)
029B	DD218405	342	LD IX,NN
029F	FD2A8405	343	LD IY,(NN)
02A3	FD218405	344	LD IY,NN
02A7	6E	345	LD L,(HL)
02A8	DD6E05	346	LD L,(IX + INI
02AB	FD6E05	347	LD L,(IY + IN'
02AE	6F	348	LD L,A
02AF	68	349	LD L,B
02B0	69	350	LD L,C
02B1	6A	351	LD L,D
02B2	6B	352	LD L,E
02B3	6C	353	LD L,H
02B4	6D	354	LD L,L

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
02B4	6D	354	LD	L,L
02B5	2E20	355	LD	L,N
	ED4F		LD	R,A
02B7	ED7B8405	356	LD	SP,(NN)
02BB	F9	357	LD	SP,HL
02BC	DDF9	358	LD	SP,IX
02BE	FDF9	359	LD	SP,IY
02C0	318405	360	LD	SP,NN
02C3	EDA8	361	LDD	
02C5	EDB8	362	LDDR	
02C7	EDA0	363	LDI	
02C9	EDB0	364	LDIR	
02CB	ED44	365	NEG	
02CD	00	366	NOP	
02CE	B6	367	OR	(HL)
02CF	DDB605	368	OR	(IX + IND)
02D2	FDB605	369	OR	(IY + IND)
02D5	B7	370	OR	A
02D6	B0	371	OR	B
02D7	B1	372	OR	C
02D8	B2	373	OR	D
02D9	B3	374	OR	E
02DA	B4	375	OR	H
02DB	B5	376	OR	L
02DC	F620	377	OR	N
02DE	ED8B	378	OTDR	
02E0	EDB3	379	OTIR	
02E2	ED79	380	OUT	(C),A
02E4	ED41	381	OUT	(C),B
02E6	ED49	382	OUT	(C),C
02E8	ED51	383	OUT	(C),D
02EA	ED59	384	OUT	(C),E
02EC	ED61	385	OUT	(C),H
02EE	ED69	386	OUT	(C),L
02F0	D320	387	OUT	N,A
02F2	EDAB	388	OUTD	
02F4	EDA3	389	OUTI	
02F6	F1	390	POP	AF
02F7	C1	391	POP	BC
02F8	D1	392	POP	DE
02F9	E1	393	POP	HL
02FA	DDE1	394	POP	IX
02FC	FDE1	395	POP	IY
02FE	F5	396	PUSH	AF
02FF	C5	397	PUSH	BC
0300	D5	398	PUSH	DE
0301	E5	399	PUSH	HL
0302	DDE5	400	PUSH	IX
0304	FDE5	401	PUSH	IY
0306	CB86	402	RES	0,(HL)
0308	DDCB0586	403	RES	0,(IX + IND)
030C	FDCB0586	404	RES	0,(IY + IND)
0310	CB87	405	RES	0,A
0312	CB80	406	RES	0,B
0314	CB81	407	RES	0,C
0316	CB82	408	RES	0,D
0318	CB83	409	RES	0,E
031A	CB84	410	RES	0,H
031C	CB85	411	RES	0,L
031E	CB8E	412	RES	1,(HL)
0320	DDCB058E	413	RES	1,(IX + IND)
0324	FDCB058E	414	RES	1,(IY + IND)
0328	CB8F	415	RES	1,A
032A	CB88	416	RES	1,B
032C	CB89	417	RES	1,C
032E	CB8A	418	RES	1,D
0330	CB8B	419	RES	1,E
0332	CB8C	420	RES	1,H
0334	CB8D	421	RES	1,L
0336	CB96	422	RES	2,(HL)
0338	DDCB0596	423	RES	2,(IX + IND)
033C	FDCB0596	424	RES	2,(IY + IND)
0340	CB97	425	RES	2,A
0342	CB90	426	RES	2,B
0344	CB91	427	RES	2,C
0346	CB92	428	RES	2,D
0348	CB93	429	RES	2,E
034A	CB94	430	RES	2,H
034C	CB95	431	RES	2,L
034E	CB9E	432	RES	3,(HL)
0350	DDCB059E	433	RES	3,(IX + IND)
0354	FDCB059E	434	RES	3,(IY + IND)
0358	CB9F	435	RES	3,A
035A	CB98	436	RES	3,B
035C	CB99	437	RES	3,C
035E	CB9A	438	RES	3,D
0360	CB9B	439	RES	3,E
0362	CB9C	440	RES	3,H
0364	CB9D	441	RES	3,L
0366	CBA6	442	RES	4,(HL)
0368	DDCB05A6	443	RES	4,(IX + IND)
036C	FDCB05A6	444	RES	4,(IY + IND)
0370	CBA7	445	RES	4,A
0372	CBA0	446	RES	4,B
0374	CBA1	447	RES	4,C
0376	CBA2	448	RES	4,D
0378	CBA3	449	RES	4,E
037A	CBA4	450	RES	4,H
037C	CBA5	451	RES	4,L
037E	CBAE	452	RES	5,(HL)
0380	DDCB05AE	453	RES	5,(IX + IND)
0384	FDCB05AE	454	RES	5,(IY + IND)
0388	CBAF	455	RES	5,A
038A	CBA8	456	RES	5,B
038C	CBA9	457	RES	5,C
038E	CBAA	458	RES	5,D
0390	CBAB	459	RES	5,E
0392	CBAC	460	RES	5,H
0394	CBAD	461	RES	5,L
0396	CBB6	462	RES	6,(HL)
0398	DDCB05B6	463	RES	6,(IX + IND)
039C	FDCB05B6	464	RES	6,(IY + IND)
03A0	CBB7	465	RES	6,A
03A2	CBB0	466	RES	6,B
03A4	CBB1	467	RES	6,C
03A6	CBB2	468	RES	6,D
03A8	CBB3	469	RES	6,E
03AA	CBB4	470	RES	6,H
03AC	CBB5	471	RES	6,L
03AE	CBBE	472	RES	7,(HL)
03B0	DDCB05BE	473	RES	7,(IX + IND)
03B4	FDCB05BE	474	RES	7,(IY + IND)

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
03B8	CBBF	475	RES	7,A
03BA	CBB8	476	RES	7,B
03BC	CBB9	477	RES	7,C
03BE	CBBA	478	RES	7,D
03C0	CBBB	479	RES	7,E
03C2	CBBC	480	RES	7,H
03C4	CBBD	481	RES	7,L
03C6	C9	482	RET	
03C7	D8	483	RET	C
03C8	F8	484	RET	M
03C9	D0	485	RET	NC
03CA	C0	486	RET	NZ
03CB	F0	487	RET	P
03CC	E8	488	RET	PE
03CD	E0	489	RET	PO
03CE	C8	490	RET	Z
03CF	ED4D	491	RETI	
03D1	ED45	492	RETN	
03D3	CB16	493	RL	(HL)
03D5	DDCB0516	494	RL	(IX + IND)
03D9	FDCB0516	495	RL	(IY + IND)
03DD	CB17	496	RL	A
03DF	CB10	497	RL	B
03E1	CB11	498	RL	C
03E3	CB12	499	RL	D
03E5	C813	500	RL	E
03E7	CB14	501	RL	H
03E9	CB15	502	RL	L
03EB	17	503	RLA	
03EC	CB06	504	RLC	(HL)
03EE	DDCB0506	505	RLC	(IX + IND)
03F2	FDCB0506	506	RLC	(IY + IND)
03F6	CB07	507	RLC	A
03F8	CB00	508	RLC	B
03FA	CB01	509	RLC	C
03FC	CB02	510	RLC	D
03FE	CB03	511	RLC	E
0400	CB04	512	RLC	H
0402	CB05	513	RLC	L
0404	07	514	RLCA	
0405	ED6F	515	RLD	
0407	CB1E	516	RR	(HL)
0409	DDCB051E	517	RR	(IY + IND)
040D	FDCB051E	518	RR	(IY + IND)
0411	CB1F	519	RR	A
0413	CB18	520	RR	B
0415	CB19	521	RR	C
0417	CB1A	522	RR	D
0419	CB1B	523	RR	E
041B	CB1C	524	RR	H
041D	CB1D	525	RR	L
041F	1F	526	RRA	
0420	CB0E	527	RRC	(HL)
0422	DDCB050E	528	RRC	(IX + IND)
0426	FDCB050E	529	RRC	(IY + IND)
042A	CB0F	530	RRC	A
042C	CB08	531	RRC	B
042E	CB09	532	RRC	C
0430	CB0A	533	RRC	D
0432	CB0B	534	RRC	E
0434	CB0C	535	RRC	H

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
0436	CB0D	536	RRC	L
0438	0F	537	RRCA	
0439	ED67	538	RRD	
043B	C7	539	RST	0
043C	D7	540	RST	10H
043D	DF	541	RST	18H
043E	E7	542	RST	20H
043F	EF	543	RST	28H
0440	F7	544	RST	30H
0441	FF	545	RST	38H
0442	CF	546	RST	08H
0443	9E	547	SBC	A,(HL)
0444	DD9E05	548	SBC	A,(IX + IND)
0447	FD9E05	549	SBC	A,(IY + IND)
044A	9F	550	SBC	A,A
044B	98	551	SBC	A,B
044C	99	552	SBC	A,C
044D	9A	553	SBC	A,D
044E	9B	554	SBC	A,E
044F	9C	555	SBC	A,H
0450	9D	556	SBC	A,L
0451	DE20	557	SBC	A,N
0453	ED42	558	SBC	HL,BC
0455	ED52	559	SBC	HL,DE
0457	ED62	560	SBC	HL,HL
0459	ED72	561	SBC	HL,SP
045B	37	562	SCF	
045C	CBC6	563	SET	0,(HL)
045E	DDCB05C6	564	SET	0,(IX + IND)
0462	FDCB05C6	565	SET	0,(IY + IND)
0466	CBC7	566	SET	0,A
0468	CBC0	567	SET	0,B
046A	CBC1	568	SET	0,C
046C	CBC2	569	SET	0,D
046E	CBC3	570	SET	0,E
0470	CBC4	571	SET	0,H
0472	CBC5	572	SET	0,L
0474	CBCE	573	SET	1,(HL)
0476	DDCB05CE	574	SET	1,(IX + IND)
047A	FDCB05CE	575	SET	1,(IY + IND)
047E	CBCF	576	SET	1,A
0480	CBC8	577	SET	1,B
0482	CBC9	578	SET	1,C
0484	CBCA	579	SET	1,D
0486	CBCB	580	SET	1,E
0488	CBCC	581	SET	1,H
048A	CBCD	582	SET	1,L
048C	CBD6	583	SET	2,(HL)
048E	DDCB05D6	584	SET	2,(IX + IND)
0492	FDCB05D6	585	SET	2,(IY + IND)
0496	CBD7	586	SET	2,A
0498	CBD0	587	SET	2,B
049A	CBD1	588	SET	2,C
049C	CBD2	589	SET	2,D
049E	CBD3	590	SET	2,E
04A0	CBD4	591	SET	2,H
04A2	CBD5	592	SET	2,L
04A4	CBD8	593	SET	3,B
04A6	CBDE	594	SET	3,(HL)
04A8	DDCB05DE	595	SET	3,(IX + IND)
04AC	FDCB05DE	596	SET	3,(IY + IND)

LOC	OBJ CODE	STMT	SOURCE STATEMENT	
04B4	CBDA	599	SET	3,D
04B6	CBDB	600	SET	3,E
04B8	CBDC	601	SET	3,H
04BA	CBDD	602	SET	3,L
04BC	CBE6	603	SET	4,(HL)
04BE	DDCB05E6	604	SET	4,(IX + IND)
04C2	FDCB05E6	605	SET	4,(IY + IND)
04C6	CBE7	606	SET	4,A
04C8	CBE0	607	SET	4,B
04CA	CBE1	608	SET	4,C
04CC	CBE2	609	SET	4,D
04CE	CBE3	610	SET	4,E
04D0	CBE4	611	SET	4,H
04D2	CBE5	612	SET	4,L
04D4	CBEE	613	SET	5,(HL)
04D6	DDCB05EE	614	SET	5,(IX + IND)
04DA	FDCB05EE	615	SET	5,(IY + IND)
04DE	CBEF	616	SET	5,A
04E0	CBE8	617	SET	5,B
04E2	CBE9	618	SET	5,C
04E4	CBEA	619	SET	5,D
04E6	CBEB	620	SET	5,E
04E8	CBEC	621	SET	5,H
04EA	CBED	622	SET	5,L
04EC	CBF6	623	SET	6,(HL)
04EE	DDCB05F6	624	SET	6,(IX + IND)
04F2	FDCB05F6	625	SET	6,(IY + IND)
04F6	CBF7	626	SET	6,A
04F8	CBF0	627	SET	6,B
04FA	CBF1	628	SET	6,C
04FC	CBF2	629	SET	6,D
04FE	CBF3	630	SET	6,E
0500	CBF4	631	SET	6,H
0502	CBF5	632	SET	6,L
0504	CBFE	633	SET	7,(HL)
0506	DDCB05FE	634	SET	7,(IX + IND)
050A	FDCB05FE	635	SET	7,(IY + IND)
050E	CBFF	636	SET	7,A
0510	CBF8	637	SET	7,B
0512	CF9	638	SET	7,C
0514	CBFA	639	SET	7,D
0516	CBFB	640	SET	7,E
0518	CBFC	641	SET	7,H
051A	CBFD	642	SET	7,L
051C	CB26	643	SLA	(HL)
051E	DDCB0526	644	SLA	(IX + IND)
0522	FDCB0526	645	SLA	(IY + IND)
0526	CB27	646	SLA	A
0528	CB20	647	SLA	B
052A	CB21	648	SLA	C
052C	CB22	649	SLA	D
052E	CB23	650	SLA	E
0530	CB24	651	SLA	H
0532	CB25	652	SLA	L
0534	CB2E	653	SRA	(HL)
0536	DDCB052E	654	SRA	(IX + IND)
053A	FDCB052E	655	SRA	(IY + IND)
053E	CB2F	656	SRA	A
0540	CB28	657	SRA	B
0542	CB29	658	SRA	C
0544	CB2A	659	SRA	D
0546	CB2B	660	SRA	E
0548	CB2C	661	SRA	H
054A	CB2D	662	SRA	L
054C	CB3E	663	SRL	(HL)
054E	DDCB053E	664	SRL	(IX + IND)
0552	FDCB053E	665	SRL	(IY + IND)
0556	CB3F	666	SRL	A
0558	CB38	667	SRL	B
055A	CB39	668	SRL	C
055C	CB3A	669	SRL	D
055E	CB3B	670	SRL	E
0560	CB3C	671	SRL	H
0562	CB3D	672	SRL	L
0564	96	673	SUB	(HL)
0565	DD9605	674	SUB	(IX + IND)
0568	FD9605	675	SUB	(IY + IND)
056B	97	676	SUB	A
056C	90	677	SUB	B
056D	91	678	SUB	C
056E	92	679	SUB	D
056F	93	680	SUB	E
0570	94	681	SUB	H
0571	95	682	SUB	L
0572	D620	683	SUB	N
0574	AE	684	XOR	(HL)
0575	DDAE05	685	XOR	(IX + IND)
0578	FDAE05	686	XOR	(IY + IND)
057B	AF	687	XOR	A
057C	A8	688	XOR	B
057D	A9	689	XOR	C
057E	AA	690	XOR	D
057F	AB	691	XOR	E
0580	AC	692	XOR	H
0581	AD	693	XOR	L
0582	EE20	694	XOR	N
0584		695 NN	DEFS	2
		696 IND	EQU	5
		697 M	EQU	10H
		698 N	EQU	20H
		699 DIS	EQU	30H
		700	END	

F/Z-80 CPU Register and Architecture

This section gives information about the actual Z80 chip including the Central Processing Unit (CPU) Register configuration.

Z-80 CPU Architecture

A block diagram of the internal architecture of the Z-80 CPU is shown in **Figure 2**. The diagram shows all of the major elements in the CPU and it should be referred to throughout the following description.

CPU Registers

The Z-80 CPU contains 208 bits of R/W memory that are accessible to the programmer. **Figure 3** illustrates how this memory is configured into eighteen 8-bit registers and four 16-bit registers. All Z-80 registers are implemented using static RAM. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or in pairs of 16-bit registers. There are also two sets of accumulator and flag registers.

Special Purpose Registers

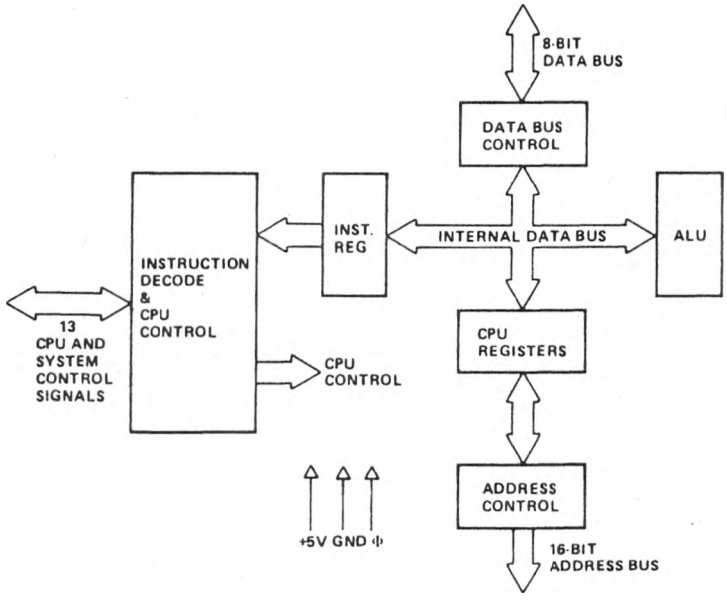


Figure 2, Z-80 CPU Block Diagram.

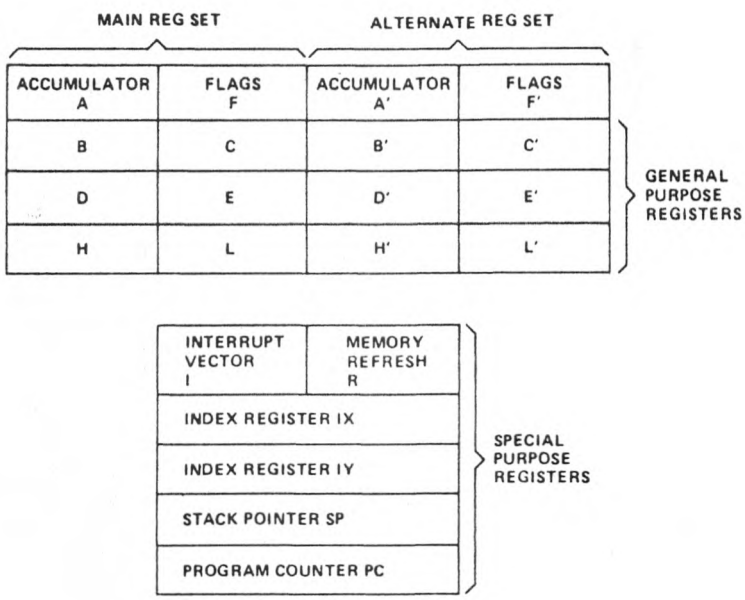


Figure 3, Z-80 CPU Register Configuration.

1. **Program Counter (PC).** The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs the new value is automatically placed in the PC, overriding the incrementer.

2. **Stack Pointer (SP).** The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file.

   Data can be pushed onto the stack from specific CPU registers or popped off of the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.

3. **Two Index Register (IX & IY).** The two independent index registers hold a 16-bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especially where tables of data are used.

4. **Interrupt Page Address Register (I).** The Z-80 CPU can be operated in a mode where an indirect call to any memory location can be achieved in response to an interrupt. The I Register is used for this purpose to store the high order 8-bits of the indirect address while the interrupting device provides the lower 8-bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with absolute minimal access time to the routine.

5. **Memory Refresh Register (R).** The Z-80 CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. Seven bits of this 8 bit register are automatically incremented after each instruction fetch. The eighth bit will remain as programmed as the result of an LD R, A instruction. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer. During refresh, the contents of the I register are placed on the upper 8 bits of the address bus.

Accumulator and Flag Registers

The CPU includes two independent 8-bit accumulators and associated 8-bit flag registers. The accumulator holds the results of 8-bit arithmetic or logical operations while the flag register indicates specific conditions for 8 or 16-bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects the accumulator and flag pair that he wishes to work with a single exchange instruction so that he may easily work with either pair.

General Purpose Registers

There are two matched sets of general purpose registers, each set containing six 8-bit registers that may be used individually as 8-bit registers or as 16-bit register pairs by the programmer. One set is called BC, DE and HL while the complementary set is called BC,' DE and HL.' At any one time the programmer can select either set of registers to work with through a single exchange command for the entire set. In systems where fast interrupt response is required, one set of general purpose registers and an accumulator/flag register may be reserved for handling this very fast routine. Only a simple exchange command need be executed to go between the routines. This greatly reduces interrupt service time by eliminating the requirement for saving and retrieving register contents in the external stack during interrupt or subroutine processing. These general purpose registers are used for a wide range of applications by the programmer. They also simplify programming, especially in ROM based systems where little external read/write memory is available.

Arithmetic & Logic Unit (ALU)

The 8-bit arithmetic and logical instructions of the CPU are executed in the ALU. Internally the ALU communicates with the registers and the external data bus on the internal data bus. The type of functions performed by the ALU include:

Add	Left or right shifts or rotates (arithmetic and logical)
Subtract	Increment
Logical AND	Decrement
Logical OR	Set bit
Logical Exclusive OR	Reset bit
Compare	Test Bit

Instruction Register and CPU Control

As each instruction is fetched from memory, it is placed in the instruction register and decoded. The control sections performs this function and then generates and supplies all of the control signals necessary to read or write data from or to the registers, control the ALU and provide all required external control signals.

Z-80 CPU Pin Description

The Z-80 CPU is packaged in an industry standard 40 pin Dual In-Line Package. The I/O pins are shown in **Figure 4** and the function of each is described below.

A_0-A_{15} (Address Bus)	Tri-state output, active high. A_0-A_{15} constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges. I/O addressing uses the 8 lower address bits to allow the user to directly select up to 256 input or 256 output ports. A_0 is the least significant address bit. During refresh time, the lower 7 bits contain a valid refresh address.
D_0-D_7 (Data Bus)	Tri-state input/output, active high. D_0-D_7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.
$\overline{M_1}$ (Machine Cycle one)	Output, active low. $\overline{M_1}$ indicates that the current machine cycle is the OP code fetch cycle of an instruction execution. Note that during execution of 2-byte op-codes, $\overline{M_1}$ is generated as each op-code byte is fetched. These two byte op-codes always begin with CBH, DDH, EDH or FDH. $\overline{M_1}$ also occurs with $\overline{IORQ}$ to indicate an interrupt acknowledge cycle.

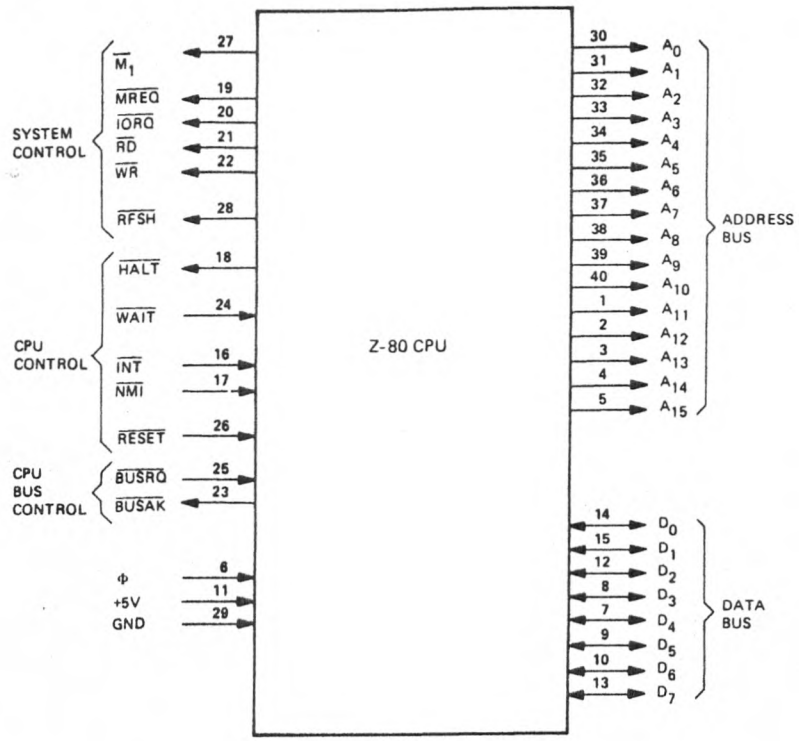


Figure 4, Z-80 Pin Configuration.

$\overline{\text{MREQ}}$ (Memory Request)	Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.
$\overline{\text{IORQ}}$ (Input/Output Request)	Tri-state output, active low. The $\overline{\text{IORQ}}$ signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An $\overline{\text{IORQ}}$ signal is also generated with an $\overline{M_1}$ signal when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus. Interrupt Acknowledge operations occur during M_1 time while I/O operations never occur during M_1 time.
$\overline{\text{RD}}$ (Memory Read)	Tri-state output, active low. $\overline{\text{RD}}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.
$\overline{\text{WR}}$ (Memory Write)	Tri-state output, active low. $\overline{\text{WR}}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.

$\overline{\text{RFSH}}$ (Refresh)	Output, active low. $\overline{\text{RFSH}}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current $\overline{\text{MREQ}}$ signal should be used to do a refresh read to all dynamic memories.
$\overline{\text{HALT}}$ (Halt state)	Output, active low. $\overline{\text{HALT}}$ indicates that the CPU has executed a HALT software instruction and is awaiting either a non maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.
$\overline{\text{WAIT}}$ (Wait)	Input, active low. $\overline{\text{WAIT}}$ indicates to the Z-80 CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active. This signal allows memory or I/O devices of any speed to be synchronized to the CPU.
$\overline{\text{INT}}$ (Interrupt Request)	Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled and if the $\overline{\text{BUSRQ}}$ signal is not active. When the CPU accepts the interrupt, an acknowledge signal ($\overline{\text{IORQ}}$ during M_1 time) is sent out at the beginning of the next instruction cycle.
$\overline{\text{NMI}}$ (Non Maskable Interrupt)	Input, negative edge triggered. The non maskable interrupt request line has a higher priority than $\overline{\text{INT}}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{\text{NMI}}$ automatically forces the Z-80 CPU to restart to location 0066_H. The program counter is automatically saved in the external stack so that the user can return to the program that was interrupted. Note that continuous WAIT cycles can prevent the current instruction from ending, and that a $\overline{\text{BUSRQ}}$ will override a $\overline{\text{NMI}}$.
$\overline{\text{RESET}}$	Input, active low. $\overline{\text{RESET}}$ forces the program counter to zero and initializes the CPU. The CPU initialization includes: 1) Disable the interrupt enable flip-flop 2) Set Register I = 00_H 3) Set Register R = 00_H 4) Set Interrupt Mode 0 During reset time, the address bus and data bus go to a high impedance state and all control output signals go to the inactive state.

$\overline{\text{BUSRQ}}$ (Bus Request)	Input, active low. The bus request signal is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these buses. When $\overline{\text{BUSRQ}}$ is activated, the CPU will set these buses to a high impedance state as soon as the current CPU machine cycle is terminated.
$\overline{\text{BUSAK}}$ (Bus Acknowledge)	Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.
Φ	Single phase TTL level clock which requires only a 330 ohm pull-up resistor to +5 volts to meet all clock requirements.

Z-80 CPU Instruction Set

The Z-80 CPU can execute 158 different instruction types including all 78 of the 8080A CPU. The instructions can be broken down into the following major groups:

- Load and Exchange
- Block Transfer and Search
- Arithmetic and Logical
- Rotate and Shift
- Bit Manipulation (set, reset, test)
- Jump, Call and Return
- Input/Output
- Basic CPU Control

* 159
8 bit load grp 17Ø-187
LD r,r' 17Ø
LD r,n 171
LD r,(HL) 172
LD r,(IX+D) 172-173
LD r,(IY+d) 174
LD (HL),r 175
LD (IX+d),r 175-176
LD (IY+d),r 176-177
LD (HL),n 177-178
LD (IX+d),n 178
LD (IY+d),n 179
LD A,(BC) 18Ø
LD A,(DE) 18Ø-181
LD A,(nn) 181
LD (BC),A 182
LD (DE),A 182-183
LD (nn),A 183-184
LD A,I 184
LD A,R 185
LD I,A 185-186
LD R,A 186-187
8 bit arithmethic and logical 227-251
ADD A,(HL) 229
ADD A,(IX+d) 229-23Ø
ADD A,(IY+d) 23Ø-231
ADD A,n 228
ADD A,r 227-228
ADD A,S 231-233
AND s 237-239
CP s 244-245
DEC m 249-251
INC (HL) 247
INC (IX+d) 247-248
INC (IY+d) 248-249
INC r 246
OR s 239-241
SUB s 233-235
SBC A,s 235-237
XOR s 241-243
16 bit load grp 188-2Ø8
LD dd,nn 188-189
LD IX,nn 189
LD IY,nn 19Ø
LD HL,(nn) 19Ø-191
LD dd,(nn) 191-192
LD IX,(nn) 192-193
LD IY,(nn) 193-194
LD (nn),HL 194-195
LD (nn),dd 195-196
LD (nn),IX 196-197
LD (nn),IY 197-198
LD SP,HL 198-199
LD SP,IX 199
LD SP,IY 2ØØ
16 bit arithmetic group 262-271
ADD HL,ss 262
ADC HL,ss 263
ADD IX,pp 265
ADD IY,rr 266
DEC ss 269
DEC IX 27Ø
DEC IY 27Ø-271
INC ss 267
INC IX 267-268
INC IY 268-269
SBC HL,ss 264
Absolute Assembly 1Ø5-1Ø6
ADC A,XH 39Ø
ADC HL,ss 263
ADD A,(HL) 229
ADD A,(IX+d) 229-23Ø
ADD A,(IY+d) 23Ø-231
ADD A,n 228
ADD A,r 227-228
ADD A,S 231-233
ADD A,XH 39Ø
ADD HL,ss 262
ADD IX,pp 265
ADD IY,rr 266
Address Different from Pass 1 381
ALASM (see Assembler)
ALBUG (see Debugger)
ALEDIT (see Editor)
ALLINK (see Linker)
ALTRAN
Model I 398-4Ø5
(see File Transfer)
AND s 237-239
AND XH 39Ø
APOP 125
APUSH 125
Arithmetic Operators 95-97
ASCII (see DEFM)
Assembler
Command 37
Description 37-42*
Directives 1Ø1-159

Errors 379
Expressions 94-95
Labels 91-92
Object Code Format 394-397
Operands 94-1ØØ
Operators 95-1ØØ
Switches 38-42
Symbols 91-92
Assembler Listing
Description 37-42
EJECT 132
HEADER 137-138
PRINT 149-15Ø
QUIT 153
STOP 157
TITLE 158
USING 158
VERSION 159
VERSION 159*
Attempt to Use a Non-Program File as a Program 383
Bad File Format 376
Bad Filename Format 376
Bad Parameters 376
BLOCK (see DEFS)
block comment 159
BIT b,(HL) 299-3ØØ
BIT b,(IX+d) 3ØØ-3Ø1
BIT b,(IY+d) 3Ø1-3Ø2
BIT b,r 298-299
Bit,set,reset, and test grp 298-3Ø8
BIT b,(HL) 299-3ØØ
BIT b,(IX+d) 3ØØ-3Ø1
BIT b,(IY+d) 3Ø1-3Ø2
BIT b,r 298-299
RES b,m 3Ø7
SET b,r 3Ø2-3Ø3
SET b,(HL) 3Ø3-3Ø4
SET b,(IX+d) 3Ø4-3Ø5
SET b,(IY+d) 3Ø6-3Ø7
Buffer Full 376
BYTE (see DEFB)
CALL cc,nn 322-324
CALL nn 321-322
Call and return group 321-33Ø
CALL cc,nn 322-324
CALL nn 321-322
RET 324-325
RET cc 325-327
RETI 327-328
RETN 328-329
RST p 329-33Ø
CCF 256
CMPD operand1,operand2, [length] 349-351
CMPI operand1,operand2, length 352-254
comment 1ØØ,159
Conditional Sections (see If Sections)
CPD 224-225
CPDR 225-226
CPI 221-223
CPIR 223-224
CPL 254-255
CP s 244-245
CP XH 39Ø
CPR operand 348
DAA 253-254
Data
Defining 1Ø4
DEFB 127
DEFE 128
DEFM 129
DEFR 129-13Ø
DEFT 13Ø-131
DEFW 131-132
DATE 126-127
DB (see DEFB)
Debugger
Description 43-65*
Loading 44
Display 45-47
Registers 48
Data 49
Breakpoints 4Ø,51-54
Disk Zap 62-65
DEC IX 27Ø
DEC IY 27Ø-271
DEC m 249-251
DEC ss 269
DEC XH 39Ø
DEFB 127
DEFE 128
DEFL 128
DEFM 129
DEFR 129-13Ø
DEFS 13Ø
DEFT 13Ø-131
DEFW 131-132
DS (see DEFS)
DW (see DEFW)
DI 258-259
Directives 1Ø1-159
Introduction 1Ø2-122
Reference 123-159

Disk Zap 62-65
DJNZ e 319-32Ø
DROP 132
Editor
Description 19-25
Errors 376
Loading 19
Insert Mode 2Ø-23
Control 22
Special Keys 23
Line Edit Mode 23-25
Subcommands 24
Special Keys 26
Command Mode 26-36
Special Keys 28
Commands 29-36
Compatibility with other Editors 35
EI 259
EJECT 132
END 132-133
ENDI 133
ENDM 133
ENTRY (see PUBLIC)
EQU 133
ERROR 24 383
ERROR 34 383
ERROR 37 383
Error Messages 375-383
EX AF,AF' 2Ø9-21Ø
Exchange,Search, and Transfer 2Ø9-226
CPI 221-223
CPIR 223-224
CPD 224-225
CPDR 225-226
EX DE,HL 2Ø9
EX AF,AF' 2Ø9-21Ø
EXX 21Ø-211
EX (SP),HL 211-212
EX (SP),IX 212-213
EX (SP),IY 213-214
LDI 215-216
LDIR 216-218
LDD 218-219
LDDR 219-221
EX DE,HL 2Ø9
EX operand 356-358
Expressions 94-95
EX (SP),HL 211-212
EX (SP),IX 212-213
EX (SP),IY 213-214
EXT 134
Extended Z8Ø Mnemonics 347-374
CPR operand 348
CMPD operand1, operand2, [length] 349-351
CMPI operand1, operand2, length 352-254
TZ operand 355
EX operand 356-358
LD double register 359-364
MOVD operand1, operand2, length 365
MOVI operand1, operand2, length 367-368
POP 369
RSTR operand 37Ø-371
SAVE operand 372-373
SVC 374
EXTERN 134-135
External Symbols 98
EXTERN 134-135
EXT 134
GLINK 135-136
GLOBAL 136-137
LINK 142-143
PUBLIC 152-153
EXX 21Ø-211
FILL 135
GLINK 135-136
GLOBAL 136-137
File Transfer
Set-Up 71
Loading 74
Errors 8Ø
Command File 8Ø-81
Connector 87-88
Technical 83-86
Object Files 83
File Not Found 383
Hit Any Key to Continue 378
General purpose arithmetic and CPU control grps 253-261
CCF 256
CPL 254-255
DAA 253-254
DI 258-259
EI 259
HALT 258
IMØ 26Ø

IM1 26Ø-261
IM2 261
NEG 255-256
NOP 257-258
SCF 257
Global File 114-118
HALT 258
HEADER 137-138
If Sections 121-122
IFDEF 138
IFF 138
IFM 139
IFNZ 139
IFP 139
IFT 139
IFUND 139
IFZ 139
IFUND 139
IFZ 139
IFDEF 138
IFF 138
IFM 139
IFNZ 139
IFP 139
IFT 139
Illegal Addressing 382
IMØ 26Ø
IM1 26Ø-261
IM2 261
IN A,(n) 331
INC (HL) 247
INC IX 267-268
INC IY 268-269
INC (IX+d) 247-248
INC (IY+d) 248-249
INCLUDE 14Ø-141
INC r 246
INC ss 267
INC XH 39Ø
IND 336-337
Index Sections 118-119
ISECT 141-142
APOP 125
APUSH 125
DROP 132
INDR 337-338
INI 333-334
INIR 334-336
Initializing
Location Counter 1Ø5
Input and output
group 331-346
IN A,(n) 331
IND 336-337
INDR 337-338
INI 333-334
INIR 334-336
IN r,(C) 332-333
OUT (C),r 339-34Ø
OUT (n),A 338-339
OUTD 343-344
OUTI 34Ø-341
OTIR 341-342
IN r,(C) 332-333
Invalid Parameter 382
ISECT 141-142
JP cc,nn 31Ø-311
JP (HL) 317
JP (IX) 318
JP (IY) 318-319
JP nn 3Ø9
JR C,e 312-313
JR e 311-312
JR NC,e 313-314
JR NZ,e 315-316
JR Z,e 314-315
Jump group 3Ø9-32Ø
DJNZ e 319-32Ø
JP cc,nn 31Ø-311
JP (HL) 317
JP (IX) 318
JP (IY) 318-319
JP nn 3Ø9
JR C,e 312-313
JR e 311-312
JR NC,e 313-314
JR NZ,e 315-316
JR Z,e 314-315
Labels 91-92
LD A,(BC) 18Ø
LD A,(DE) 18Ø-181
LD A,I 184
LD A,(nn) 181
LD A,R 185
LD (BC),A 182
LD dd,nn 188-189
LD dd,(nn) 191-192
LD (DE),A 182-183
LD (HL),n 177-178
LD HL,(nn) 19Ø-191
LD (HL),r 175
LD I,A 185-186
LD (IX+d),n 178
LD (IX+d),r 175-176
LD IX,nn 189
LD IX,(nn) 192-193
LD (IY+d),n 179
LD (IY+d),r 176-177
LD IY,nn 19Ø
LD IY,(nn) 193-194

LD (nn),A 183-184
LD R,A 186-187
LD r,n 171
LD r,r' 17Ø
LD r,(HL) 172
LD r,(IX+D) 172-173
LD r,(IY+d) 174
LD (nn),dd 195-196
LD (nn),HL 194-195
LD (nn),IX 196-197
LD (nn),IY 197-198
LD SP,HL 198-199
LD SP,IX 199
LD SP,IY 2ØØ
LDD 218-219
LD double register 359-364
LDDR 219-221
LDI 215-216
LDIR 216-218
LD r,XH 39Ø
LD XH,r 39Ø
LD XH,n 39Ø
Line Length Too Long, Truncating line 376
Line Number Too Large 376
LINK 142-143
Linker
Command 67-68
Technical 69
Errors 381-383
LITORG 143-144
Location Counter 1Ø5-1Ø8
MACRO 144-145
Macro Editor Assembler Compatibility 35
Macro Sections 12Ø-121
ENDM 133
MACRO 144-145
Memory Map 391-392
Missing External Transfer Address 382
Model I
ALTRAN 398
MOVD operand1,operand2, length 365
MOVI operand1,operand2, length 367-368
Multiply Defined Entry Symbol 381
NEG 255-256
NOEND 146
NOFILL 147
NOLOAD 147
NOP 257-258
No Text 377
Number Bases 1Ø2-1Ø3
OBJ 147-148
Occurrence Too Large 377
Open Attempt For a File Already Open 383
Operands 94-1ØØ
Operators 95-1ØØ
ORG 148-149
OR s 239-241
OR XH,n 39Ø
OTIR 341-342
OUT (C),r 339-34Ø
OUT (n),A 338-339
OUTD 343-344
OUTI 34Ø-341
overflow 161
parity odd 161
parity even 161
PATCH 149
POP 369
POP IX 2Ø5-2Ø7
POP IY 2Ø7-2Ø8
POP qq 2Ø4-2Ø5
PUSH IX 2Ø2-2Ø3
PUSH IY 2Ø3-2Ø4
PUSH qq 2ØØ-2Ø1
PRINT 149-15Ø
Program Section 1Ø9-11Ø
PSECT 151-152
Pseudo Ops (see Directives)
PUBLIC 152-153
QUIT 153
RADIX 153-154*
1Ø2
REF 154-155
Relocatable 1Ø6, 1Ø9-11Ø
Operators 98
RES b,m 3Ø7
RESLD r,n,m 388
RESLOC 155
RES n,m 388
RET 324-325
RET cc 325-327
RETI 327-328
RETN 328-329
RL m 281-283
RL m 386
RLA 273
RLCLD r,m 386
RLC m 386

RLC r 276-277
RLC (HL) 277-278
RLC (IX+d) 278-279
RLC (IY+d) 279-281
RLD 294-296
RLLD r,m 386
Rotate and shift group 272
RL m 281-283
RLA 273
RLC r 276-277
RLC (HL) 277-278
RLC (IX+d) 278-279
RLC (IY+d) 279-281
RLD 294-296
RR m 285-287
RRA 275
RRC m 283-285
RRCA 274
RRD 296-297
SLA m 287-289
SRA m 29Ø-292
SRL m 292-294
RR m 285-287
RRA 275
RRC m 283-285
RRCA 274
RRCLD r,m 386
RRD 296-297
RRLLD r,m 386
RST p 329-33Ø
RSTR operand 37Ø-371
Sample Session 11-17
SAVE operand 372-373
SBC A,s 235-237
SBC HL,ss 264
SBC A,XH 39Ø
SCF 257
Search Arg Too Long 377
Series I Editor Assembler Compatibility 35
Converting 393
SET b,r 3Ø2-3Ø3
SET b,(HL) 3Ø3-3Ø4
SET b,(IX+d) 3Ø4-3Ø5
SET b,(IY+d) 3Ø6-3Ø7
SETLOC 155
SET n,m 388
SETLD r,n,m 388
SLA m 287-289
SLOLD r,m 386
SRA m 29Ø-292
SRA m 386
SRALD r,m 386
SRL m 292-294
SRLLD r,m 386
SUB s 233-235
TZ operand 355
XOR s 241-243
SLA m 386
SLALD r,m 386
SLO m 386
SRL m 386
STOP 157
Storage Defining 1Ø4-1Ø5
DEFS 13Ø
FILL 135
NOFILL 147
SUB XH 39Ø
Symbols Defining 1Ø3
External 98, 11Ø-118*
Syntax 91-92
Symbol Table Overflow 381
Syntax Error 377
SVC 374
TITLE 158
TIME 157
Total Line Length Too Long 377
TRSDOS, Model I 398-4Ø5
Undefined External Symbol 381
Undocumented Z8Ø Instructions 385-39Ø
UPGRADE 398
USING 158
VERSION 159
WORD (see DEFW)
XOR XH 39Ø
Z8Ø alphabetic 412-417
extended 347-374
hardware 418-424
mnemonics 161-345*
notations 163-164
numeric List 4Ø6-411
undocumented 385-39Ø