

EDITOR- ASSEMBLER Package



S
O
F
T
W
A
R
E

Catalog Number 26-2202

Contents

1. EDIT-80 User's Manual
 2. MACRO-80 User's Manual
 3. LINK-80 Reference Manual
 4. Z-80 Instruction Set
-
-

O V E R V I E W

- - - - -

This manual describes Radio Shack's Editor-Assembler package, for use with the TRS-80 Disk Operating System (TRSDOS). It does not teach you how to write programs. You will need to consult another source for information on programming.

The Editor-Assembler package includes five modules:

- .The Editor, EDIT/CMD, for writing and editing source programs.
- .The Macro-Assembler, M80/CMD, which reads your assembly-language source program and translates it into relocatable object code.
- .The Cross Reference Facility, CREF80/CMD, which generates a cross reference listing for your assembly-language source program.
- .The Linking Loader, L80/CMD, which lets you load assembled or complied programs, execute them, and save them as TRSDOS command files.
- .The FORTRAN Subroutine Library, FORLIB/REL, for use by the Linking Loader in conjunction with your relocatable FORTRAN files.

In the manual you will find several references to the TRS-80 FORTRAN compiler. This program is not included in the Editor-Assembler package, nor is it necessary for using it. You can ignore all references to it.

Using the Editor-Assembler Package

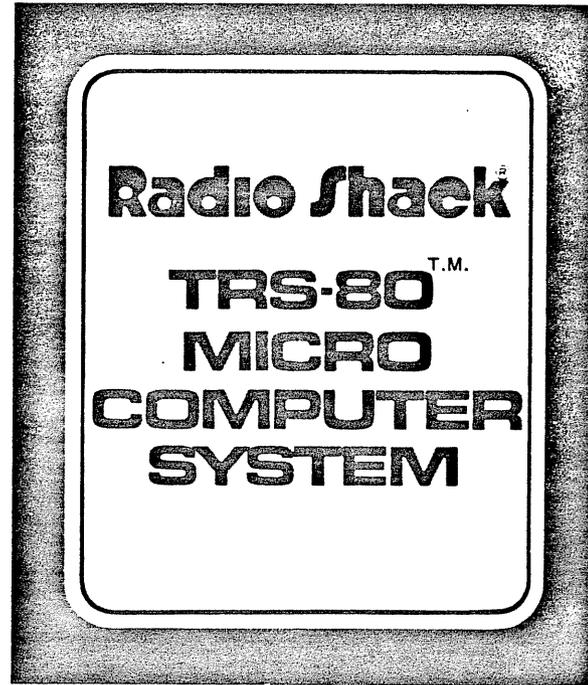
The Editor-Assembler package contains two disks. One disk contains M80/CMD, CREF80/CMD, and EDIT/CMD. The other contains L80/CMD and FORLIB/REL. Both disks contain all TRSDOS files, so either diskette can be placed in Drive 0.

WARNING

Never remove a diskette which contains open files. This means you should never swap diskettes during an edit session, for example. Before changing diskettes, be sure all files are closed.

(c) Copyright 1979 by Microsoft, Licensed to Radio Shack, A Division of Tandy Corporation, Fort Worth, Texas

**EDIT-80
User's
Manual**



Important Note

Be sure to make **BACKUP** copies of both diskettes before you begin using the **EDITOR—ASSEMBLER** package.



Contents

CHAPTER 1	EDIT-80 Operation	5
1.1	Introduction	5
1.2	Running EDIT-80	5
1.3	Ending the Editing Session	6
1.4	Line Numbers and Ranges	7
1.5	Format Notation	8
CHAPTER 2	Beginning Interline Editing	10
2.1	Insert Command	10
2.2	Delete Command	11
2.3	Replace Command	11
2.4	Print Command	12
2.5	List Command	12
2.6	Number Command	13
CHAPTER 3	Intraline Editing - Alter Mode	15
3.1	Alter Command	15
3.2	Alter Mode Subcommands	15
3.3	Cursor Position	16
3.4	Insert Text	16
3.5	Delete Text	17
3.6	Replace Text	18
3.7	Find Text	18
3.8	Ending and Restarting Alter Mode	19
3.9	Extend Command	19
CHAPTER 4	Find and Substitute Commands	20
4.1	Find Command	20
4.2	Substitute Command	22
CHAPTER 5	Pages	23
5.1	Specifying Page Numbers	23
5.2	Inserting Page Marks	24
5.3	Deleting Page Marks	24
5.4	Begin Command	25
5.5	Other Commands and Page Marks	25

CHAPTER 6	Exiting EDIT-80	26
6.1	Exit Command	26
6.2	Quit Command	26
6.3	Write Command	26
6.4	Index Files	27
6.5	Parameters	27
6.5.1	BASIC Switch	28
6.5.2	SEQ and UNSEQ Switches	28
APPENDIX A	- Alphabetic Summary of Commands	30
APPENDIX B	- Alphabetic Summary of Alter Mode Subcommands .	32
APPENDIX C	- Summary of Notation	34
APPENDIX D	- EDIT-80 Special Characters	35
APPENDIX E	- Error Messages	36
APPENDIX F	- Output File Format	38

CHAPTER 1

EDIT-80 Operation

1.1 Introduction

EDIT-80 is a line-oriented and character-oriented text editor. EDIT-80 commands are simple and straightforward, yet powerful enough to accommodate the most demanding user. For the novice or for those requiring only cursory use of EDIT-80, the first four chapters of this document contain all the information necessary to complete a fairly extensive editing session. The remaining chapters describe the enhancements to EDIT-80 that provide the user with more sophisticated techniques.

1.2 Running EDIT-80

To run EDIT-80, type and enter

EDIT

at TRSDOS command level. EDIT-80 will ask for the filename by typing

FILE:

Enter the name of your file. Use TRSDOS filename format for the filename:

filename[/extension][.password][:drive#]

If the filename refers to a file that already exists, type the filename followed by <enter>, and EDIT-80 will read in the file. If the file does not have line numbers, EDIT-80 will append them, beginning with line number 100 and incrementing by 100. After EDIT-80 prints

```
Version x.x
Copyright 1977,78 (c) by Microsoft
Created: xxxx
xxxx Bytes free
*
```

it is at command level, as indicated by the * prompt. All commands to EDIT-80 are entered after the * prompt.

If the filename refers to a new file to be created, type the filename followed by the <break> key.

EDIT-80 will return the message

```

Creating
Version x.x
Copyright 1977,78 (c) by Microsoft
Created: xxxx
xxxx Bytes free
*
-

```

Next enter the command I (see Section 2.1 for a further description of the I command). EDIT-80 will type the first line number, 00100, followed by a tab.

```

*I
00100

```

Now you are ready to enter the first line of your file. A line consists of up to 255 characters and is terminated by <enter>. After every line entered, EDIT-80 will type the next line number, incrementing by 100. This is the "permanent increment." (There are various commands that will change the permanent increment - see Chapter 2.) Line numbers 00000 through 99999 are available for use in your EDIT-80 file.

NOTE

Radio Shack's FORTRAN package and Macro-Assembler both support input files which include EDIT-80 line numbers.

If a typing error is made while entering or editing a line, use the Delete key (←) to delete the incorrect character(s). If, while typing a line, you wish to erase the entire line and start over, type shift ←.

When you wish to stop entering lines and return to command level, type the <break> key after the next available line number.

1.3 Ending the Editing Session

To exit EDIT-80, enter the Exit command:

```

*E
-

```

The Exit command writes the edited file to disk under the filename that was used to create the file. Subsequent editing sessions with that file require that a filename be specified with the Exit

command. See Section 6.1.

To exit EDIT-80 without writing the edited file to disk, enter the Quit command:

*Q

After execution of a Quit command, all the changes entered during the editing session are lost.

1.4 Line Numbers and Ranges

Most commands to EDIT-80 require a reference to a line number or a range of line numbers. A line number is specified by using the number itself (it is not necessary to type the leading zeros), or one of three special characters that EDIT-80 recognizes as line numbers. These special characters are:

.	(period)	refers to the current line
^	(up arrow)	refers to the first line
*	(asterisk)	refers to the last line

Ranges may be specified in one of two ways:

1. With a colon. The designation

200:1000

means all lines from line number 200 to line number 1000, inclusive. If lines 200 and 1000 do not exist, the range will begin with the first line number greater than 200 and end with the last line number less than 1000.

2. With an exclamation point. The designation

200!3

means the range of three lines that starts with line 200. If line 200 does not exist, 200!3 means the range of three lines that starts with the first line after 200.

Here are some examples of line and range specifications (shown here with the Print command):

P.:2000 Prints the range that begins with
the current line and ends with
line 2000.

P500 Prints line 500.

P. Prints the current line.

- P.!15 Prints the range that begins at the current line and ends after the next 15 lines.
- PA:1500 Prints the range that begins with the first line and ends with line 1500.
- PA:* Prints the entire file.

See Appendix C for more examples of range specification.

1.5 Format Notation

Throughout this document, generalized formats of EDIT-80 commands are given to guide the user. These formats employ the following conventions:

1. Items in square brackets are optional.
2. Items in capital letters must be entered as shown.
3. Items in lower case letters enclosed in angle brackets are to be supplied by the user:

<position> supply any line number (up to five digits) or ".", "^" or "*"

<range> supply any <position> or any <range>
 <range> = <position>:<position>
 or
 <position>!<number>

<inc> supply a non-zero integer to be used as an increment between line numbers

<filename> supply any legal TRSDOS filename as described in Section 1.2

4. Punctuation must be included where shown.
5. Items separated by a vertical line are mutually exclusive. Choose one.
6. <break> refers to the break key and is echoed as \$. If you see a \$ in a format notation, it refers to the break key.

7. In any command format, spaces and tabs are insignificant; except within a line number or a filename.
8. Underlined items are typed by EDIT-80.

CHAPTER 2

Beginning Interline Editing

Editing a file by printing, inserting, deleting and replacing entire lines or groups of lines is termed interline editing. This section describes the commands used to perform these functions.

2.1 Insert Command

The Insert command is used to insert lines of text into the file. EDIT-80 types each line number for you during insert mode. The format of the Insert command is:

```
I[<position>[,<inc> | ;<inc>]]
```

Insertion of lines begins at <position> and continues until <break> is typed or until the available space at that point in the file is depleted. (In either case, EDIT-80 returns to command level.)

If no <inc> is included with the command, the default is the permanent increment. ,<inc> specifies a new increment that is then established as the permanent increment. ;<inc> specifies a temporary increment for use with the current command, but does not change the permanent increment.

If no argument is supplied with the Insert command (I<enter>), insertion resumes where the last insert command was terminated, using the last temporary increment. If only <position> is supplied (I<position><enter>), the permanent increment is used.

EDIT-80 will not allow insertion where a line already exists. If <position> is a line number that already exists, the command I<position> will add the permanent increment (or the temporary increment, if one was specified) to <position> and allow insertion at line number <position>+<inc>. If line <position>+<inc> already exists, or if line numbers exist between <position> and <position>+<inc>, an error message will be printed.

The line feed (↓) key may be used to start a new physical line without starting a new logical line, thus providing compatibility with Microsoft BASIC

source files.

Here is an example using the Insert command:

```

*I7740,10
07740
-----
07750
-----
07760 $
*
-

```

K=K+1
GO TO 400

Note that the insertion is terminated with <break>. The <break> key may be typed at the end of the last line inserted (instead of <enter>) or at the beginning of the next line. A line is not saved if <break> is the first key typed on that line.

2.2 Delete Command

The Delete command removes a line or range of lines from the file. The format of the command is:

D<range>

After a Delete command is executed, the current line (".") is set to the first line of the deleted range.

Examples of the Delete Command:

```

D7000      delete line 7000

D.         delete the current line

D200:900  delete lines 200 through 900

D2000:*   delete all lines from line
          2000 through the last line

```

2.3 Replace Command

The Replace command combines the effects of the Delete and Insert commands. The format of the command is:

R<range>[,<inc> | ;<inc>]

The Replace command deletes all of the lines in <range>, then allows the user to enter new text as if an Insert command had been issued. (EDIT-80 types the line numbers.)

The options for selecting the increment between

line numbers are the same as those for the Insert command (see Section 2.1).

Here is an example using the Replace command:

```
*R500:600;50
00500          DO 80 I=1,7
00550          Y(I)=ALOG(Y(I))
00600 80      CONTINUE
*
```

In the above example, the lines in the range 500 to 600 were deleted and replaced by three new lines (500, 550 and 600), using a temporary increment of 50. Insertion terminated automatically because there was not enough room for EDIT-80 to create line 650.

2.4 Print Command

The Print command prints lines at the terminal. The format of the command is:

P<range>

Examples of the Print command:

P.:700 print all lines from the
current line through line 700

P800:* print all lines from line 800
through the end of the file

Typing <line feed> (↓) at command level will cause the line after the current line to be printed. Typing <break> at command level will cause the line before the current line to be printed. Typing P<enter> will cause the next 20 lines to be printed.

2.5 List Command

The List command

L<range>

is the same as the Print command, except the output goes to the line printer.

2.6

Number Command

The Number command renumbers lines of text. You may wish to renumber lines to "make room" for an insertion, or just to organize the line numbers in a file. The format of the Number command is

```
N[<start>][,<inc> | ;<inc>][=<range>]
```

where:

1. <start> is the first number of the new sequence. If <start> is omitted but <range> is included, <start> is set to the first line of <range>. If <start> and <range> are omitted, but <inc> is included, <start> is set to <inc>. If <start> is omitted and <inc> is included and <range> specifies only a page number (e.g., =/2), <start> is also set to <inc> on that page. If <start>, <range> and <inc> are omitted, <start> is set to the permanent increment.
2. <inc> is the increment between line numbers in the new sequence. The options for selection of the increment are the same as those described for the Insert command (see Section 2.1).
3. <range> is the range of line numbers to be renumbered. If <range> is omitted, the entire file is renumbered.

If the current line is renumbered, "." is reset to the same physical line.

If a Number command would result in line numbers being placed out of sequence, or if EDIT-80 cannot fit all the lines using the given increment, an "Out of order" error message is returned.

Due to EDIT-80's internal memory requirements for executing a Number command, an attempt to renumber a very large file may result in an "Insufficient memory" error. If this situation arises, renumber a smaller portion of the file, write it to disk, renumber another portion, and so on. (See Write Command, Section 6.3.)

Examples of the Number command:

```
N7000;100=200:1000
```

Lines 200 through 1000 will be renumbered to begin at line 7000 and increment by 100.

N,10=400:*

Lines 400 through the end will be renumbered to begin with 400 and increment by 10.

N9000=10000:*

Using the permanent increment lines 10000 through the end will be renumbered to begin at 9000.

N,100

Renumber the whole file using increment 100.

N,5=2350!10

This command could be used to make room for an insert by compactifying the ten lines starting with 2350.

CHAPTER 3

Intraline Editing - Alter Mode

The interline editing commands discussed thus far let you edit by inserting, deleting or replacing entire lines. Of course many editing situations require changes to an existing line but not necessarily retyping of the line. Editing a line without retyping it is called intraline editing, and it is done in Alter mode.

3.1 Alter Command

The Alter command is used to enter Alter mode. The format of the command is:

A<range>

In Alter mode, EDIT-80 types the line number of the line to be altered and waits for an Alter mode subcommand.

3.2 Alter Mode Subcommands

Alter mode subcommands are used to move the cursor; search for text; or insert, delete or replace text within a line. The subcommands are not echoed on the terminal.

Many of the Alter mode subcommands may be preceded by an integer, causing the command to be executed that number of times. (When no integer is specified, the default is always 1.) In many cases, the entire command may also be prefaced with a minus sign (-) which changes the normal direction of the command's action. For example:

D	deletes the next character
6D	deletes the next 6 characters
-D	deletes the last character
-12D	deletes the last 12 characters

Each Alter mode subcommand is described below. A summary of the subcommands is given in Appendix B.

NOTE

In the following descriptions, \$ represents <break>, <ch> represents any character, <text> represents a string of characters of arbitrary length and i represents any integer.

3.3 Cursor Position

The following commands or terminal keys are used to change the position of the cursor in the line. The location of the cursor is called the "current position."

- <space> spaces over characters. i<space> moves the cursor i characters to the right. -i<space> moves the cursor i characters to the left. Characters are printed as you space over them.
- moves the cursor to the end of the line. If preceded by a minus sign, moves the cursor to the beginning of the line.
- L prints the remainder of the line and positions the cursor at the beginning of the line. Proceed with the next Alter mode subcommand.
- P prints the remainder of the line and recycles the cursor to the current position. Proceed with the next Alter mode subcommand.
- W moves to the beginning of the next word. A word is defined as a contiguous collection of letters, numbers, ".", "\$", or "%". iW advances the cursor over the next i words. -iW moves the cursor back through i words to the left.

3.4 Insert Text

- I inserts text. I<text>\$ inserts the given text beginning at the current position. Note that the text must be followed by a <break> or by <enter>.

- B inserts spaces (blanks) at the current position. The B command may be preceded by an integer to insert that many spaces. Spaces are inserted to the right of the cursor only.
- G inserts characters. `iG<ch>` inserts `i` copies of `<ch>`.
- X extends a line. The X subcommand types the remainder of the line, goes into insert mode and lets you insert text at the end of the line. The `-X` subcommand moves to the beginning of the line and goes into insert mode. (Don't forget to end your insertion with `<break>` or `<enter>`.)

3.5 Delete Text

- D deletes the character at the current position. `iD` deletes `i` characters beginning at the current position. `-iD` deletes `i` characters to the left of the current position. Deleted characters are surrounded by double exclamation points.
- ← The back-arrow key may also be used to delete characters. The character immediately to the left of the current position is deleted. `i<back-arrow>` is equivalent to `-iD`.
- H deletes (hacks) the remainder of the line to the right of the cursor (or to the left of the cursor if `-H` is typed) and enters the insert mode. Text insertion proceeds as if an I command had been typed.
- K deletes (kills) characters. `K<ch>` deletes all characters up to but not including `<ch>`. `iK<ch>` deletes all characters up to the `i`th occurrence of `<ch>`. `-iK<ch>` deletes all characters up to and including the `i`th previous occurrence of `<ch>`. If `<ch>` is not found, the command is not executed.

- O deletes (obliterates) text. O<text>\$ deletes all text up to but not including the next occurrence of <text>. iO<text>\$ deletes all text up to the ith occurrence of <text>. -iO<text>\$ deletes all characters up to and including the ith previous occurrence of <text>.
- T deletes (truncates) the remainder of the line to the right of the cursor (or to the left of the cursor if -T is typed) and exits Alter mode.
- Z deletes (zaps) words. iZ deletes the next i words. -iZ deletes words to the left of the cursor.

3.6 Replace Text

- R replaces text. iR<text>\$ deletes the next i characters and replaces them with <text>. -iR<text>\$ replaces text to the left of the cursor. The deleted characters are echoed between double exclamation points.
- C changes characters one character at a time. C<ch> changes the next character to <ch>. Only the new character is echoed. iC may be followed by i characters to change that many characters; or it may be followed by fewer than i characters and terminated with <break>, in which case the remaining characters will not be changed. -iC does an i<back arrow> and then an iC. The i<back arrow> is echoed between exclamation points.

3.7 Find Text

- S searches for a character. S<ch> searches for the next occurrence of <ch> after the current position and positions the cursor before the character. iS<ch> searches for the ith occurrence of <ch>. -S<ch> and -iS<ch> search for the (ith) previous occurrence of <ch> and position the cursor immediately before it. The character at the cursor position is not included in the search. If <ch> is not found, the command is ignored.

F finds text. F<text>\$ finds the next occurrence of <text> and positions the cursor at the beginning of the string. iF<text>\$ finds the ith occurrence of <text>. -F<text>\$ and -iF<text>\$ find the (ith) previous occurrence of <text> and position the cursor before it.

3.8 Ending and Restarting Alter Mode

<cr> carriage return. Prints the remainder of the line, enters the changes and concludes altering of that line.

A same as carriage return.

E enters the changes and concludes altering of that line, but does not print the remainder of the line.

N restores the original line (changes are not saved) and either moves to the next line (if an A<range> command is still in progress), or returns to command level.

Q restores the original line (changes are not saved), exits (quits) Alter mode, and returns to command level.

Shift ← Restores the original line, stays in Alter mode and repositions the cursor at the beginning of the line. Echoes as ^Y.

3.9 Extend Command

The Extend command is issued at command level and is used to extend lines. The format of the command is

X<range>

The effect of the X command is equivalent to typing an A command, followed by an X subcommand. After entering an X command, proceed by typing the text to be inserted at the end of the line. Don't forget you are now in Alter mode and may use any of the Alter mode subcommands, once <break> has been typed.

The Extend command is particularly useful for placing comments in assembly language programs.

CHAPTER 4

Find and Substitute Commands

When it is necessary to change a certain portion of text, it is not always immediately known where that text is located in the file. Even with a listing of the file on hand, it is a tiresome task to scan the listing to find the line number of a particular item that must be changed.

The EDIT-80 Find and Substitute commands allow the user to quickly locate text and make necessary changes.

4.1 Find Command

The Find command locates a given string of text in the file and types the line(s) containing that string. The format of the command is:

```
F[<range>][,<limit>] <enter> | $<string>$
```

where \$ represents the escape key and <limit> is the number of lines containing <string> to be found. A limit of zero will find all occurrences of <string>. The following rules apply to the format of the Find command:

1. If \$<string>\$ is omitted, the last string given in a Find command is used.
2. If <limit> is omitted and \$<string>\$ is included, <limit> is assumed to be 1.
3. If <limit> and \$<string>\$ are omitted, the previous limit is assumed.
4. If <range> is omitted and \$<string>\$ is included, the entire range from the previous Find command is used.
5. If <range> and \$<string>\$ are omitted, the search for the previous string continues from the line where the last occurrence was found.

If the search is unsuccessful, an error message is printed.

Here is a sample editing session using Find:

```
*F^:*$WHI(I)$
01100   WHI(I)=0
*F<enter>
01400   IF (P.GT.WHI(I))WHI(I)=P
*A.
01400   .
.
.
*F,2$WLO(I)$
01200   WLO(I)=9999
01500   IF (P.LT.WLO(I))WLO(I)=P
*A.
01500   .
.
.
*F.:*$AVG$
Search fails
*F$MEAN$
03700   MEAN=SUM/40
*F,0
04200   IF (P.GT.MEAN) M=M+1
06700   WRITE (6,170) MEAN, M
*A4200
04200   .
.
.
```

Find the first line that contains WHI(I). Prints line 1100. Find the next one. Prints line 1400. Caught a mistake in this line. Alter it.

Find the first two lines in the file that contain WLO(I) (range is still .*). Prints lines 1200 and 1500. Alter line 1500.

Find the first line in the file that contains AVG. There aren't any. Try finding MEAN instead. Prints line 3700. Find all other lines containing MEAN. (Search begins at the line after line 3700.) Finds two more (4200 and 6700). Alter line 4200, etc.

4.2 Substitute Command

The Substitute command locates a given string, replaces it with a new string and types the new line(s). The format of the command is:

S[<range>][,<limit>] <enter> | \$<old string>\$<new string>\$

where \$ represents <break>, and <limit> is the number of lines in which <old string> is to be replaced by <new string>. A limit of zero will replace all occurrences of <old string> with <new string>. <new string> may be a null string. The following rules apply to the format of the Substitute command:

1. If \$<old string>\$<new string>\$ are omitted, the strings given in the last Substitute command are used.
2. If <limit> is omitted and \$<old string>\$<new string>\$ are included, <limit> is assumed to be zero.
3. If <limit> and \$<old string>\$<new string>\$ are omitted, the previous limit is assumed.
4. If <range> is omitted and \$<old string>\$<new string>\$ are included, the entire range from the previous Substitute command is used.
5. If <range> and \$<old string>\$<new string>\$ are omitted, substitution continues from where the last substitution left off.

If no occurrence of <old string> is found, an error message is printed.

Example:

```
*SA:5000$ALPHA$BETA$
00950  BETA(K)=ABS(1.-LST(K))
01750  WRITE(6,400) BETA(K)
04100  IF (BETA(K).LT.0)GOTO 9000
```

From the first line to line 5000, replace all occurrences of ALPHA with BETA.

CHAPTER 5

Pages

It is possible to divide an EDIT-80 file into sections called pages, which are separated by page marks. The first page of a file is always page 1, and EDIT-80 always enters command level on page 1 of a multiple-page file. Each subsequent page begins with a page mark and is numbered sequentially. On any given page, the complete range of line numbers (00000 to 99999 or any portion thereof) may be used.

If EDIT-80 encounters a form feed while reading in a file, it will enter a page mark at that point in the file. If EDIT-80 encounters a line number that is less than the previous line number, it will automatically insert a page mark so that proper line number sequence may be maintained. When EDIT-80 writes a file out to disk, a form feed is output with each page mark. Then, when the file is listed, each new page of the file starts on a new physical page.

5.1 Specifying Page Numbers

In a single-page file, only a line number is needed to indicate <position>. In a multiple-page file, EDIT-80 must know the page number as well as the line number to determine a <position>. That is, <position> is indicated by

```
<line>[/<page>]
```

where

<line> is ".", "^", "*" or a number of up to five digits.

<page> is ".", "^", "*" or a number of up to five digits. When specifying a page, the characters ".", "^" and "*" refer to the current page, the first page and the last page, respectively. If <page> is omitted, the current page is assumed.

Consequently, in a multiple-page file a <range>, which may be indicated by

```
<position>:<position>
or
<position>!<number>
```

may also contain page numbers. If the page number is omitted from the first line number in the range, it is assumed to be the current page. If the page

number is omitted from the second line number in the range, it is assumed to be on the same page as the first line number in the range.

Here are some examples of line numbers and ranges that include page number specification:

100/2:*/*	Line 100 on page 2 through the last line on the last page
100/2:*	Line 100 on page 2 through the end of that page
100:*/5	Line 100 on the current page through the last line on page 5
100/*	Line 100 on the last page
100/.:*/3	Line 100 on the current page through the last line on page 3

See Appendix C for more examples of range specification.

5.2 Inserting Page Marks

Page marks may be inserted in the file at the discretion of the user. To insert a page mark, use the Mark command. The format is:

M<position>

The page mark is inserted immediately after <position>. <position> must exist or an error message will be printed.

The current line reference (".") is retained after a Mark command is executed. That is, if <position> is before ".", then "." will be moved to the next page and will still point to the same physical line.

5.3 Deleting Page Marks

Page marks are deleted with the K (Kill) command. The format of the command is:

K/<page>

The K command deletes the page mark after <page>. For example, in a four-page file, K/2 would delete

the second page mark (the page mark that started page 3), and the pages would then be numbered 1, 2, and 3. The last line number on <page> must be lower than the first line number on <page>+1 before a K/<page> command can be executed.

5.4 Begin Command

Use the Begin command to return to the beginning of a page. The format of the Begin command is:

B[/<page>]

If <page> is omitted, the B command returns to the beginning of page one.

5.5 Other Commands and Page Marks

1. A Delete command that crosses over a page boundary will delete all lines in the range, but will not delete the page mark.
2. A Print command that moves off the current page will print the new page number prior to printing the first line specified in the command.
3. When output is being done with the List command, a form feed will be printed with each page mark, and the page number will be printed on each page.
4. A range specified with an exclamation point may cross a page boundary.
5. If the range specified in a Number command crosses page boundaries, numbering will start over on each new page; the first line number will equal the increment. Consequently, in the Number command, <start> and the first line of <range> must be on the same page.

CHAPTER 6

Exiting EDIT-80

Section 1.3 introduced the Exit and Quit commands for exiting EDIT-80. These two commands will be described more completely in this chapter. An additional command, the Write command, will also be presented.

6.1 Exit Command

The Exit command is used to write the file to disk and return to TRSDOS. The format of the command is:

```
E[<filename>][-<switch>]
```

The edited file is saved on the disk under <filename>. When exiting a new file for the first time, <filename> may be omitted. (In which case, the opening filename is assigned.) Otherwise, a new filename is required for each Exit. The previous file serves as a back-up.

The optional <switch> controls the format of the output. (See Section 6.5.)

6.2 Quit Command

The Quit command is used to return to TRSDOS without writing the edited file to disk. To Quit editing, simply enter:

```
Q
```

After a Quit command, all changes entered during the editing session are lost.

6.3 Write Command

The Write command writes the edited text to disk and then returns to EDIT-80 command level. It does not exit the editor, and the current position in the file is not changed. The format of the command is:

```
W[<filename>][-<switch>]
```

A filename is not required in the first Write of a new file. A filename is required, however, in all subsequent Write and Exit commands.

The optional <switch> controls the format of the output. (See Section 6.5.)

6.4 Index Files

When reading in a file to be edited, EDIT-80 generates information it needs about each block of the disk file. With a small file, this information is generated in a few seconds, each time the file is read in. However, with larger files (5K or more), the time lag required to read in the file becomes significant. Thus, when EDIT-80 saves a file of 42 or more records on the disk, it also saves a small file, separate from the text file, containing the required information about the text file.

This small file is called the index file, and it can be read faster than the text file. EDIT-80 saves the index file under a filename that is the same as the text filename (passwords not included), with a Z preceding the first two letters of the extension. For example, if the file is called FOO/MAC.SAM, the index file is called FOO/ZMA.

When EDIT-80 is asked to edit a file, it first checks for an index file. If an index file exists, EDIT-80 reads the index file instead of the text file. Care must be taken if the text file is modified by another editor or changed and saved in BASIC. The user must then delete the index file prior to editing the text file again with EDIT-80. If the index file is not deleted, EDIT-80 will have meaningless information about the text file.

6.5 Parameters

When reading in a file, EDIT-80 expects it to be in its own representation. If the file appears to be in another representation, EDIT-80 will add line numbers and try to convert the file to EDIT-80 standard format. There are, however, several other representations that EDIT-80 accepts, if the proper switch is appended to the input filename. Switches are always preceded by a dash (-):

```
filename[/ext][.password][:drive#] [-switch]
```

For example: FOO/BAS.SAM-BASIC

6.5.1 BASIC Switch

If the BASIC switch is appended to the input filename, EDIT-80 will read the file using the following algorithm:

1. All leading spaces and tabs are removed from each line.
2. The first non-blank character must be a digit.
3. From 1 to 5 leading digits are converted to a line number. More than 5 leading digits constitutes a fatal error.
4. A tab is inserted if the first non-digit is not a space or a tab. If the first non-digit is a space, it is replaced by a tab. If the first non-digit is a tab, it is left alone.
5. On output, if UNSEQ (see Section 6.5.2) has been selected, leading zeros in the line number are suppressed and the tab is converted to a space.

Because BASIC uses line numbers to control the sequence of program execution, BASIC users should beware of renumbering with the N command. Microsoft BASIC will ignore page marks from the EDIT-80 file, so a BASIC file may have multiple pages. Insure, however, that no line number appears more than once in the program.

6.5.2 SEQ and UNSEQ Switches

If the SEQ switch is appended to the input filename, EDIT-80 will use the same algorithm to interpret the text file as with the BASIC switch. However, when the file is output, it will be in standard EDIT-80 format, unless the UNSEQ switch is appended to the output filename.

The UNSEQ switch on input tells EDIT-80 to append its own line numbers to the incoming file, regardless of what it looks like. This switch must be used if the incoming file has digits at the beginning of lines with high bits on that are not to be interpreted as line numbers.

On output, the UNSEQ switch must be specified (if it hasn't been already) to output a non-standard file. That is, if BASIC is specified on input and UNSEQ is specified on output, the file will be output in BASIC format. If BASIC was not specified

on input and UNSEQ is specified on output, the file will be output with no line numbers and no trailing tab. If the UNSEQ switch was specified on input and the user wishes to output a standard file, the SEQ switch on output will override the UNSEQ switch.

APPENDIX A

Alphabetic Summary of Commands

<u>Command</u>	<u>Format and Description</u>	<u>Page</u>
Alter	A<range> Enters Alter mode.	15
Begin	B[<page>] Moves to the beginning of <page>. Default is page 1.	25
Delete	D<range> Deletes lines.	11
Exit	E[<filename>][-<switch> Writes the edited text to disk and exits the editor.	6, 26
Find	F[<range>][,<limit>] <enter> \$<string>\$ Finds occurrences of <string>.	20
Insert	I[<position>][,<inc> ;<inc>] Inserts lines beginning at <position> using increment <inc>. With no argument, continues with previous Insert command.	10
Kill	K/<page> Deletes the page mark at the end of <page>.	24
List	L<range> Prints lines at the line printer.	12
Mark	M<position> Inserts a page mark after <position>.	24
Number	N[<start>][,<inc> ;<inc>][=<range>] Renumbers the lines in <range> so they begin at <start> and increment by <inc>.	13
Print	P[<range>] Prints lines at the terminal. With no argument, prints the next 20 lines.	12
Quit	Q Exits the editor without writing the edited text to disk.	6, 26

Replace	R<range>[,<inc> ;<inc>] Replaces line(s) using increment <inc>.	18
Substitute	S[<range>][,<limit>]<enter> \$<old string>\$<new string>\$ Replaces <old string> with <new string>.	22
Write	W[<filename>][-<switch>] Writes the edited text to disk but does not exit the editor.	26
eXtend	X<range> Allows insertion of text at the end of a line.	19

APPENDIX B

Alphabetic Summary of Alter Mode Subcommands

<u>Command</u>	<u>Format</u>	<u>Action</u>
A	A	Prints the remainder of the line, enters the changes and concludes altering of that line
B	[i]B	Inserts spaces
C	[-][i]C<ch>[...<ch>]	Replaces characters
D	[-][i]D	Deletes characters
E	E	Enters the changes and concludes altering of that line
F	[-][i]F\$<text>\$	Finds <text>
G	[i]G<ch>	Inserts i copies of <ch>
H	[-]H<text>\$	Deletes the remainder of the line and enters the insert mode
I	I<text>\$	Inserts <text>
K	[-][i]K<ch>	Deletes all characters up to <ch>
L	L	Positions the cursor at the beginning of the line
N	N	Restores the original line and either moves to the next line (if an A<range> command is still in progress) or returns to command level
O	[-][i]O<text>\$	Deletes all characters up to <text>
P	P	Recycles the cursor to the current position
Q	Q	Exits Alter mode and restores the original line

R	<code>[-][i]R<text>\$</code>	Replaces <i>i</i> characters with <code><text></code>
S	<code>[-][i]S<ch></code>	Finds <code><ch></code>
T	<code>[-]T</code>	Deletes the remainder of the line and concludes altering of the line
W	<code>[-][i]W</code>	Moves the cursor over words
X	<code>[-]X</code>	Extends the line
Z	<code>[-][i]Z</code>	Deletes words
	<code>[-] →</code>	Moves the cursor to the end of the line
	←	Deletes characters
	<code>[-][i]<space></code>	Moves the cursor over characters
	<code><enter></code>	Prints the remainder of the line, enters changes and concludes altering of that line
	Shift ←	Restores the original line, stays in Alter mode and repositions the cursor at the beginning of the line. Echoes as ↑Y.

APPENDIX C
Summary of Notation

The notation used in this document may be defined as follows:

<line> = <number> | . | ^ | *
 <page> = <number> | . | ^ | *
 <position> = <line>[/<page>]
 <range> = <position>[:<position> | !<number>]

where:

<number> = <digit> | <number><digit>
 <digit> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Shorthand Notation for Ranges

The following "shorthand" forms of range specifications may be used with EDIT-80 commands.

<u>Shorthand Notation</u>	<u>Equivalent To</u>	<u>Range Specified</u>
/<page>	^/<page>:*/<page>	All of <page>.
/<page1>:/<page2>	^/<page1>:*/<page2>	The first line on <page1> through the last line on <page2>.
:	^/1:*/*	The entire file.
<position>:	<position>:*/*	<position> through the end of the file. e.g., .: is the same as ./.:*/*
:<position>	^/1:<position>	The first line in the file through <position>. e.g., .: is the same as ^/1:./.

APPENDIX D

EDIT-80 Special Characters

<break>	Aborts the command in progress and returns to EDIT-80 command level.
→	Types a tab.
Shift ←	Erases the line being typed and lets you start over. When used in Alter mode, Shift<-- restores the original line, stays in Alter mode and repositions the cursor at the beginning of the line.

Control characters are typed by holding down the shift key, the down-arrow (↓) key and the correct alpha key at the same time.

Control O	Suspends/resumes output (at the terminal or line printer) from an EDIT-80 command.
Control S	Halts/resumes execution of an EDIT-80 command.

APPENDIX E

Error Messages

Fatal Errors

Disk I/O errors are fatal. The corresponding TRSDOS error message will be printed.

Any TRSDOS system error message is fatal.

Illegal line format

Occurs when EDIT-80 finds a line with strange contents or a strange line number. This should not normally occur when editing a file created by EDIT-80. It is usually caused by reading files not meant for editing, such as binary files.

Edit Error MessagesIllegal command

Tells the user a nonexistent or ill-formed command was typed.

Insufficient memory available

Occurs when the user has made enough changes to the file to have exhausted EDIT-80's memory area. This should only happen when a large file has many changes or when large portions of code are being inserted or renumbered. A W command should be done to compress memory.

No string given

Tells the user the F or S command was given without a search string. This usually happens when using the F or S command with no arguments prior to issuing an F or S command with arguments, or when an <escape> without a search string is typed following the range.

No such line(s)

This message is issued if a command references a line or range which does not exist. Usually occurs when the proper page number is omitted from the line or range.

Line too long

This message is issued when the user attempts to enter a line longer than 255 characters. This may happen when the line is read or as a result of a command which alters the line.

Out of order

Indicates that the line numbers in the file would not be in ascending order if the command were to be executed. This frequently happens when trying to insert where there is not

enough room or trying to delete a page mark.

Search fails

An informative message that tells the user a search was unsuccessful.

Wrap around

This message is printed whenever a line greater than 99999 would be generated.

File Errors

File already exists

Issued if the user tries to give the name of an existing file to a new file, or tries to rename a file using the name of an existing file in an E or W command.

File not found

Issued if the file specified in a command could not be found.

Illegal file specification

Informs the user that the command string contains an illegal character of some kind.

APPENDIX F

Output File Format

Compilers and assemblers should ignore the line numbers and page marks included in EDIT-80 output files (except when included in listing files). Microsoft TRS-80 FORTRAN and MACRO-80 both do so.

A line number consists of five decimal digits followed by a tab character. All six bytes have the high order bit (bit 7) equal to one. It is not recommended that EDIT-80 files be listed with the TRSDOS LIST command. Graphics characters may appear in the line numbers. Use EDIT-80's Print command instead.

When writing a file with -BASIC set, the line numbers have the high order bits equal to zero. Each line number is followed by a space that has the high order bit equal to zero.

A page mark is a form feed character with the high order bit equal to one.

Index

Alter command	15
Alter mode	15
Alter mode subcommands	15-19, 32
BASIC switch	28, 38
Begin command	25
Command level	5
Control-O	35
Control-S	35
Delete command	11, 25
Delete key	6, 33
Error messages	36
Exit command	6, 26
Extend command	19
Find command	20
Form feed	23, 25, 38
Index files	27
Insert command	6, 10
Kill command	24
Line feed	10, 12
Line numbers	5-7, 23, 27, 38
List command	12, 25
Mark command	24
Number command	13, 25, 28
Page mark	23-25, 28
Page numbers	23
Parameters	27
Permanent increment	6, 10, 13
Print command	12, 25, 38
Quit command	7, 26
Replace command	11
SEQUENCE switch	28
Shift<--	6, 19, 33, 35
Space bar	16
Substitute command	22
Switches	27
Tab key	16, 35
TRSDOS	5-6, 8, 26, 36, 38

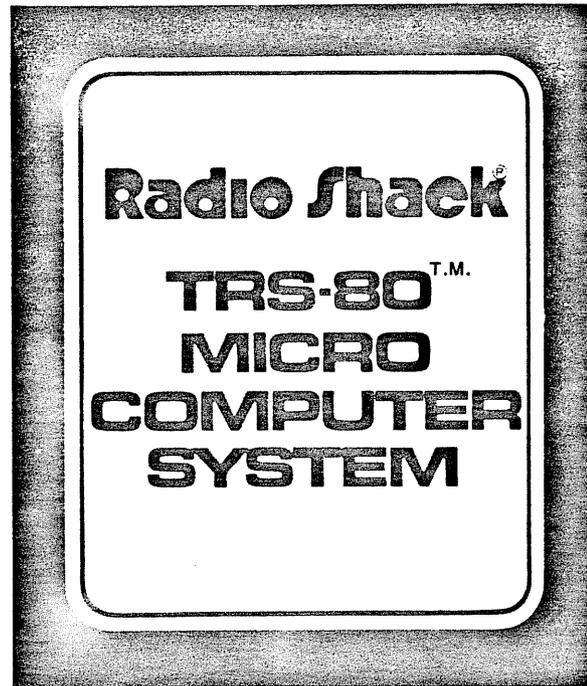
UNSEQUENCE switch 28

Write command 26

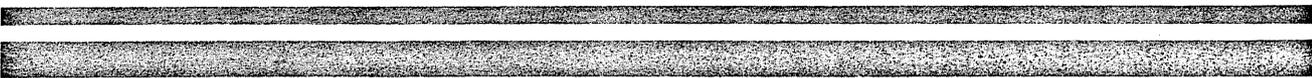




MACRO-80 User's Manual



**For Use with the TRS-80
Disk Operating System (TRSDOS)**





CONTENTS

1	Running MACRO-80	5
	1.1 Command Format	5
2	Format of MACRO-80 Source Files	8
	2.1 Statements	8
	2.2 Symbols	9
	2.3 Numeric Constants	10
	2.4 Strings	11
3	Expression Evaluation	11
	3.1 Arithmetic and Logical Operators	11
	3.2 Modes	12
	3.3 Externals	13
4	Opcodes as Operands	14
5	Pseudo Operations	14
	5.1 ASEG	14
	5.2 COMMON	14
	5.3 CSEG	15
	5.4 Define Byte	15
	5.5 Define Character	15
	5.6 Define Space	16
	5.7 DSEG	16
	5.8 Define Word	16
	5.9 END	16
	5.10 ENTRY/PUBLIC	17
	5.11 EQU	17
	5.12 EXT/EXTRN	17
	5.13 NAME	17
	5.14 Define Origin	18
	5.15 PAGE	18
	5.16 SET	18
	5.17 SUBTTL	18
	5.18 TITLE	18
	5.19 .COMMENT	19
	5.20 .PRINTX	19
	5.21 .RADIX	20
	5.22 .REQUEST	20
	5.23 .Z80	20
	5.24 .8080	20
	5.25 Conditional Pseudo Operations	21
	5.26 Listing Control Pseudo Operations	22
	5.27 Relocation Pseudo Operations	22
	5.28 Relocation Before Loading	24

6	Macros and Block Pseudo Operations	24
6.1	Terms	24
6.2	REPT-ENDM	25
6.3	IRP-ENDM	26
6.4	IRPC-ENDM	26
6.5	MACRO	26
6.6	ENDM	28
6.7	EXITM	28
6.8	LOCAL	29
6.9	Special Macro Operators and Forms .	29
7	Using Z80 Pseudo-ops	30
8	Sample Assembly	31
9	MACRO-80 Errors	32
10	Compatibility with Other Assemblers	33
11	Format of Listings	34
11.1	Symbol Table Listing	35
12	Cross Reference Facility	36

MACRO-80 Assembler

Assembly language programs and subroutines are assembled with MACRO-80. Just as the FORTRAN compiler generates relocatable object code from a FORTRAN program, MACRO-80 generates relocatable object code from an assembly language program. Running MACRO-80 is very similar to running the FORTRAN compiler, and the command format is identical. The default extension for a MACRO-80 source file is /MAC.

1 Running MACRO-80

When you give TRSDOS the command

```
M80
```

(diskette #1 must be in the disk drive), you are running the MACRO-80 assembler. When the assembler is ready to accept commands, it prompts the user with an asterisk. To exit the assembler, use the <break> key.

Command lines are also supported by MACRO-80. After executing a command line, the assembler automatically exits to the operating system.

1.1 Command Format

An assembler command conveys the name of the source file you want to assemble, and what options you want to use. Here is the format for an assembler command (square brackets indicate optional):

```
[object filename][,listing filename]=source filename[-switch...]
```

NOTE

All filenames must be in TRSDOS filename format:

```
filename[/ext][.password][:drive#].
```

If you are using the assembler's default extensions, it is not necessary to specify an extension in an assembler command.

Let's look individually at each part of the assembler command:

1. Object filename
To create a relocatable object file, this part of the command must be included. It is simply the name that you want to call the object file. The default extension for the object filename is /REL.
2. Listing filename
To create a listing file, this part of the command must be included. It is simply the name that you want to call the listing file. The default extension for the listing file is /LST.
3. Source filename
An assembler command must always include a source filename -- that is how the assembler "knows" what to assemble. It is simply the name of a MACRO-80 program you have saved on disk. The default extension for a MACRO-80 source filename is /MAC. The source filename is always preceded by an equal sign in an assembler command.

Examples (asterisk is typed by M80):

- | | |
|-----------------------|---|
| *=TEST | Assemble the program TEST/MAC without creating an object file or listing file. |
| *TEST,TEST=TEST | Assemble the program TEST/MAC. Create a relocatable object file called TEST/REL and a listing file called TEST/LST. |
| *,TEST.PASS=TEST.PASS | Assemble the program TEST/MAC.PASS and create a listing file called TEST/LST.PASS (No object file created.) |
| *TESTOBJ=TEST | Assemble the program TEST/MAC and create an object file called TESTOBJ/REL. (No listing file created.) |

4. Switch
A switch on the end of a command specifies a special parameter to be used during assembly. Switches are always preceded by a dash (-). More than one switch may be used in the same

command. The available switches are:

<u>Switch</u>	<u>Action</u>
O	Print all listing addresses in octal.
H	Print all listing addresses in hexadecimal (default condition).
C	Force generation of a cross reference file.
Z	Assemble Z80 (Zilog format) mnemonics (default condition).
I	Assemble 8080 mnemonics.

Examples:

*CT.ME,CT.ME=CT.ME-O Assemble the program CT/MAC.ME. Create a listing file called CT/LST.ME and an object file called CT/REL.ME. The addresses in the listing file will be in octal.

*LT,LT=LT-C Assemble the program LT/MAC. Create an object file called LT/REL, a listing file called LT/LST, and a cross reference file called LT/CRF. (See Section 12.)

2 Format of MACRO-80 Source Files

In general, MACRO-80 accepts a source file that is almost identical to source files for INTEL compatible assemblers. Input source lines of up to 132 characters in length are acceptable.

MACRO-80 preserves lower case letters in quoted strings and comments. All symbols, opcodes and pseudo-opcodes typed in lower case will be converted to upper case.

NOTE

If the source file includes line numbers from an editor, each byte of the line number must have the high bit on. Line numbers from Microsoft's EDIT-80 Editor are acceptable.

2.1 Statements

Source files input to MACRO-80 consist of statements of the form:

```
[label[:]] [operator] [arguments] [;comment]
```

With the exception of the ISIS assembler \$ controls (see Section 1.10), it is not necessary that

statements begin in column 1. Multiple blanks or tabs may be used to improve readability.

If a label is present, it is the first item in the statement and is immediately followed by a colon. If it is followed by two colons, it is declared as PUBLIC (see ENTRY/PUBLIC, Section 5.10). For example:

```
FOO:: RET
```

i, equivalent to

```
PUBLIC FOO
FOO: RET
```

The next item after the label (or the first item on the line i, no label is present) is an operator. An operator may be an opcode (8080 or Z80 mnemonic), pseudo-op, macro call or expression. The evaluation order is as follows:

1. Macro call
2. Opcode/Pseudo operation
3. Expression

Instead of flagging an expression as an error, the assembler treats it as if it were a DB statement (see Section 5.4).

The arguments following the operator will, of course, vary in form according to the operator.

A comment always begins with a semicolon and ends with a carriage return. A comment may be a line by itself or it may be appended to a line that contains a statement. Extended comments can be entered using the .COMMENT pseudo operation (see Section 5.19).

2.2 Symbols

MACRO-80 symbols may be of any length, however, only the first six characters are significant. The following characters are legal in a symbol:

```
A-Z    0-9    $    .    ?    @
```

The underline character is also legal in a symbol. A symbol may not start with a digit. When a symbol is read, lower case is translated into upper case. If a symbol reference is followed by ## it is

declared external (see also the EXT/EXTRN pseudo-op, Section 5.12).

2.3 Numeric Constants

The default base for numeric constants is decimal. This may be changed by the .RADIX pseudo-op (see Section 5.21). Any base from 2 (binary) to 16 (hexadecimal) may be selected. When the base is greater than 10, A-F are the digits following 9. If the first digit of the number is not numeric (i.e., A-F), the number must be preceded by a zero. This eliminates the use of zero as a leading digit for octal constants, as in previous versions of MACRO-80.

Numbers are 16-bit unsigned quantities. A number is always evaluated in the current radix unless one of the following special notations is used:

nnnnB	Binary
nnnnD	Decimal
nnnnO	Octal
nnnnQ	Octal
nnnnH	Hexadecimal
X'nnnn'	Hexadecimal

Overflow of a number beyond two bytes is ignored and the result is the low order 16-bits.

A character constant is a string comprised of zero, one or two ASCII characters, delimited by quotation marks, and used in a non-simple expression. For example, in the statement

```
DB      'A' + 1
```

'A' is a character constant. But the statement

```
DB      'A'
```

uses 'A' as a string because it is in a simple expression. The rules for character constant delimiters are the same as for strings.

A character constant comprised of one character has as its value the ASCII value of that character. That is, the high order byte of the value is zero, and the low order byte is the ASCII value of the character. For example, the value of the constant 'A' is 41H.

A character constant comprised of two characters has as its value the ASCII value of the first

character in the high order byte and the ASCII value of the second character in the low order byte. For example, the value of the character constant "AB" is $41H * 256 + 42H$.

2.4 Strings

A string is comprised of zero or more characters delimited by quotation marks. Either single or double quotes may be used as string delimiters. The delimiter quotes may be used as characters if they appear twice for every character occurrence desired. For example, the statement

```
DB      "I am ""great"" today"
```

stores the string

```
I am "great" today
```

If there are zero characters between the delimiters, the string is a null string.

3 Expression Evaluation

3.1 Arithmetic and Logical Operators

The following operators are allowed in expressions. The operators are listed in order of precedence.

NUL

LOW, HIGH

*, /, MOD, SHR, SHL

Unary Minus

+, -

EQ, NE, LT, LE, GT, GE

NOT

AND

OR, XOR

Parentheses are used to change the order of precedence. During evaluation of an expression, as soon as a new operator is encountered that has precedence less than or equal to the last operator

encountered, all operations up to the new operator are performed. That is, subexpressions involving operators of higher precedence are computed first.

All operators except +, -, *, / must be separated from their operands by at least one space.

The byte isolation operators (HIGH, LOW) isolate the high or low order 8 bits of an Absolute 16-bit value. If a relocatable value is supplied as an operand, HIGH and LOW will treat it as if it were relative to location zero.

3.2 Modes

All symbols used as operands in expressions are in one of the following modes: Absolute, Data Relative, Program (Code) Relative or COMMON. (See Section 5 for the ASEG, CSEG, DSEG and COMMON pseudo-ops.) Symbols assembled under the ASEG, CSEG (default), or DSEG pseudo-ops are in Absolute, Code Relative or Data Relative mode respectively. The number of COMMON modes in a program is determined by the number of COMMON blocks that have been named with the COMMON pseudo-op. Two COMMON symbols are not in the same mode unless they are in the same COMMON block.

In any operation other than addition or subtraction, the mode of both operands must be Absolute.

If the operation is addition, the following rules apply:

1. At least one of the operands must be Absolute.
2. Absolute + <mode> = <mode>

If the operation is subtraction, the following rules apply:

1. <mode> - Absolute = <mode>
2. <mode> - <mode> = Absolute
where the two <mode>s are the same.

Each intermediate step in the evaluation of an expression must conform to the above rules for modes, or an error will be generated. For example, if FOO, BAZ and ZAZ are three Program Relative symbols, the expression

FOO + BAZ - ZAZ

will generate an R error because the first step (FOO + BAZ) adds two relocatable values. (One of the values must be Absolute.) This problem can always be fixed by inserting parentheses. So that

FOO + (BAZ - ZAZ)

is legal because the first step (BAZ - ZAZ) generates an Absolute value that is then added to the Program Relative value, FOO.

3.3 Externals

Aside from its classification by mode, a symbol is either External or not External. (See EXT/EXTRN, Section 5.12.) An External value must be assembled into a two-byte field. (Single-byte Externals are not supported.) The following rules apply to the use of Externals in expressions:

1. Externals are legal only in addition and subtraction.
2. If an External symbol is used in an expression, the result of the expression is always External.
3. When the operation is addition, either operand (but not both) may be External.
4. When the operation is subtraction, only the first operand may be External.

4 Opcodes as Operands

8080 opcodes are valid one-byte operands. Note that only the first byte is a valid operand. For example:

```
MVI    A,(JMP)
ADI    (CPI)
MVI    B,(RNZ)
CPI    (INX H)
ACI    (LXI B)
MVI    C,MOV A,B
```

Errors will be generated if more than one byte is included in the operand -- such as (CPI 5), LXI B,LABEL1) or (JMP LABEL2).

Opcodes used as one-byte operands need not be enclosed in parentheses.

NOTE

Opcodes are not valid operands in Z80 mode.

5 Pseudo Operations

5.1 ASEG

ASEG

ASEG sets the location counter to an absolute segment of memory. The location of the absolute counter will be that of the last ASEG (default is 0), unless an ORG is done after the ASEG to change the location. The effect of ASEG is also achieved by using the code segment (CSEG) pseudo operation and the -P switch in LINK-80. See also Section 5.27.

5.2 COMMON

```
COMMON /<block name>/
```

COMMON sets the location counter to the selected common block in memory. The location is always the beginning of the area so that compatibility with the FORTRAN COMMON statement is maintained. If <block name> is omitted or consists of spaces, it is considered to be blank common. See also Section 5.27.

5.3 CSEG

CSEG

CSEG sets the location counter to the code relative segment of memory. The location will be that of the last CSEG (default is 0), unless an ORG is done after the CSEG to change the location. CSEG is the default condition of the assembler (the INTEL assembler defaults to ASEG). See also Section 5.27.

5.4 Define Byte

```
DB    <exp>[,<exp>...]
```

```
DB    <string>[<string>...]
```

The arguments to DB are either expressions or strings. DB stores the values of the expressions or the characters of the strings in successive memory locations beginning with the current location counter.

Expressions must evaluate to one byte. (If the high byte of the result is 0 or 255, no error is given; otherwise, an A error results.)

Strings of three or more characters may not be used in expressions (i.e., they must be immediately followed by a comma or the end of the line). The characters in a string are stored in the order of appearance, each as a one-byte value with the high order bit set to zero.

Example:

```
0000'   4142           DB    'AB'
0002'    42           DB    'AB' AND 0FFH
0003'   41 42 43      DB    'ABC'
```

5.5 Define Character

```
DC    <string>
```

DC stores the characters in <string> in successive memory locations beginning with the current location counter. As with DB, characters are stored in order of appearance, each as a one-byte value with the high order bit set to zero. However, DC stores the last character of the string with the high order bit set to one. An error will

result if the argument to DC is a null string.

5.6 Define Space

DS <exp>

DS reserves an area of memory. The value of <exp> gives the number of bytes to be allocated. All names used in <exp> must be previously defined (i.e., all names known at that point on pass 1). Otherwise, a V error is generated during pass 1 and a U error may be generated during pass 2. If a U error is not generated during pass 2, a phase error will probably be generated because the DS generated no code on pass 1.

5.7 DSEG

DSEG

DSEG sets the location counter to the Data Relative segment of memory. The location of the data relative counter will be that of the last DSEG (default is 0), unless an ORG is done after the DSEG to change the location. See also Section 5.27.

5.8 Define Word

DW <exp>[,<exp>...]

DW stores the values of the expressions in successive memory locations beginning with the current location counter. Expressions are evaluated as 2-byte (word) values.

5.9 END

END [<exp>]

The END statement specifies the end of the program. If <exp> is present, it is the start address of the program. If <exp> is not present, then no start address is passed to LINK-80 for that program.

5.10 ENTRY/PUBLIC

```

ENTRY <name>[,<name>...]
or
PUBLIC <name>[,<name>...]

```

ENTRY or PUBLIC declares each name in the list as internal and therefore available for use by this program and other programs to be loaded concurrently. All of the names in the list must be defined in the current program or a U error results. An M error is generated if the name is an external name or common-blockname.

5.11 EQU

```
<name> EQU <exp>
```

EQU assigns the value of <exp> to <name>. If <exp> is external, an error is generated. If <name> already has a value other than <exp>, an M error is generated.

5.12 EXT/EXTRN

```

EXT <name>[,<name>...]
or
EXTRN <name>[,<name>...]

```

EXT or EXTRN declares that the name(s) in the list are external (i.e., defined in a different program). If any item in the list references a name that is defined in the current program, an M error results. A reference to a name where the name is followed immediately by two pound signs (e.g., NAME##) also declares the name as external.

5.13 NAME

```
NAME ('modname')
```

NAME defines a name for the module. Only the first six characters are significant in a module name. A module name may also be defined with the TITLE pseudo-op. In the absence of both the NAME and TITLE pseudo-ops, the module name is created from the source file name.

5.14 Define Origin

```
ORG <exp>
```

The location counter is set to the value of <exp> and the assembler assigns generated code starting with that value. All names used in <exp> must be known on pass 1, and the value must either be absolute or in the same area as the location counter.

5.15 PAGE

```
PAGE [<exp>]
```

PAGE causes the assembler to start a new output page. The value of <exp>, if included, becomes the new page size (measured in lines per page) and must be in the range 10 to 255. The default page size is 50 lines per page. The assembler puts a form feed character in the listing file at the end of a page.

5.16 SET

```
<name> SET <exp>
```

SET is the same as EQU, except no error is generated if <name> is already defined.

5.17 SUBTTL

```
SUBTTL <text>
```

SUBTTL specifies a subtitle to be listed on the line after the title (see TITLE, Section 5.18) on each page heading. <text> is truncated after 60 characters. Any number of SUBTTLS may be given in a program.

5.18 TITLE

```
TITLE <text>
```

TITLE specifies a title to be listed on the first line of each page. If more than one TITLE is given, a Q error results. The first six characters of the title are used as the module name unless a NAME pseudo operation is used. If neither a NAME or TITLE pseudo-op is used, the module name is created from the source filename.

5.19 .COMMENT

```
.COMMENT <delim><text><delim>
```

The first non-blank character encountered after `.COMMENT` is the delimiter. The following `<text>` comprises a comment block which continues until the next occurrence of `<delimiter>` is encountered. For example, using an asterisk as the delimiter, the format of the comment block would be:

```
.COMMENT *
any amount of text entered
here as the comment block
.
.
.      *
;return to normal mode
```

5.20 .PRINTX

```
.PRINTX <delim><text><delim>
```

The first non-blank character encountered after `.PRINTX` is the delimiter. The following text is listed on the terminal during assembly until another occurrence of the delimiter is encountered. `.PRINTX` is useful for displaying progress through a long assembly or for displaying the value of conditional assembly switches. For example:

```
IF      CPM
.PRINTX /CPM version/
ENDIF
```

NOTE

`.PRINTX` will output on both passes. If only one printout is desired, use the `IF1` or `IF2` pseudo-op.

5.21 .RADIX

```
.RADIX <exp>
```

The default base (or radix) for all constants is decimal. The `.RADIX` statement allows the default radix to be changed to any base in the range 2 to 16. For example:

```
LXI    H,OFFH
.RADIX 16
LXI    H,OFF
```

The two LXIs in the example are identical. The `<exp>` in a `.RADIX` statement is always in decimal radix, regardless of the current radix.

5.22 .REQUEST

```
.REQUEST <filename>[,<filename>...]
```

`.REQUEST` sends a request to the LINK-80 loader to search the filenames in the list for undefined globals before searching the FORTRAN library. The filenames in the list should be in the form of legal MACRO-80 symbols. They should not include filename extensions or disk specifications. The LINK-80 loader will supply its default extension and will assume the currently selected disk drive.

5.23 .Z80

`.Z80` enables the assembler to accept Z80 opcodes. This is the default condition. Z80 mode may also be set by appending the Z switch to the MACRO-80 command string -- see Section 1.2.

5.24 .8080

`.8080` enables the assembler to accept 8080 opcodes. 8080 mode may also be set by appending the I switch to the MACRO-80 command string -- see Section 1.2.

5.25 Conditional Pseudo Operations

The conditional pseudo operations are:

IF/IFT <exp>	True if <exp> is not 0.
IFE/IFF <exp>	True if <exp> is 0.
IF1	True if pass 1.
IF2	True if pass 2.
IFDEF <symbol>	True if <symbol> is defined or has been declared External.
IFNDEF <symbol>	True if <symbol> is undefined or not declared External.
IFB <arg>	True if <arg> is blank. The angle brackets around <arg> are required.
IFNB <arg>	True if <arg> is not blank. Used for testing when dummy parameters are supplied. The angle brackets around <arg> are required.

All conditionals use the following format:

```

IFxx  [argument]
.
.
.
[ELSE
.
.
.
]
ENDIF

```

Conditionals may be nested to any level. Any argument to a conditional must be known on pass 1 to avoid V errors and incorrect evaluation. For IF, IFT, IFF, and IFE the expression must involve values which were previously defined and the expression must be absolute. If the name is defined after an IFDEF or IFNDEF, pass 1 considers the name to be undefined, but it will be defined on pass 2.

ELSE

Each conditional pseudo operation may optionally be used with the ELSE pseudo operation which allows alternate code to be generated when the opposite condition exists. Only one ELSE is permitted for a

given IF, and an ELSE is always bound to the most recent, open IF. A conditional with more than one ELSE or an ELSE without a conditional will cause a C error.

ENDIF

Each IF must have a matching ENDIF to terminate the conditional. Otherwise, an 'Unterminated conditional' message is generated at the end of each pass. An ENDIF without a matching IF causes a C error.

5.26 Listing Control Pseudo Operations

Output to the listing file can be controlled by two pseudo-ops:

.LIST and .XLIST

If a listing is not being made, these pseudo-ops have no effect. .LIST is the default condition. When a .XLIST is encountered, source and object code will not be listed until a .LIST is encountered.

The output of cross reference information is controlled by .CREF and .XCREF. If the cross reference facility (see Section 12) has not been invoked, .CREF and .XCREF have no effect. The default condition is .CREF. When a .XCREF is encountered, no cross reference information is output until .CREF is encountered.

The output of MACRO/REPT/IRP/IRPC expansions is controlled by three pseudo-ops: .LALL, .SALL, and .XALL. .LALL lists the complete macro text for all expansions. .SALL lists only the object code produced by a macro and not its text. .XALL is the default condition; it is similar to .SALL, except a source line is listed only if it generates object code.

5.27 Relocation Pseudo Operations

The ability to create relocatable modules is one of the major features of MACRO-80. Relocatable modules offer the advantages of easier coding and faster testing, debugging and modifying. In addition, it is possible to specify segments of assembled code that will later be loaded into RAM (the Data Relative segment) and ROM/PROM (the Code Relative segment). The pseudo operations that

select relocatable areas are CSEG and DSEG. The ASEG pseudo-op is used to generate non-relocatable (absolute) code. The COMMON pseudo-op creates a common data area for every COMMON block that is named in the program.

The default mode for the assembler is Code Relative. That is, assembly begins with a CSEG automatically executed and the location counter in the Code Relative mode, pointing to location 0 in the Code Relative segment of memory. All subsequent instructions will be assembled into the Code Relative segment of memory until an ASEG or DSEG or COMMON pseudo-op is executed. For example, the first DSEG encountered sets the location counter to location zero in the Data Relative segment of memory. The following code is assembled in the Data Relative mode, that is, it is assigned to the Data Relative segment of memory. If a subsequent CSEG is encountered, the location counter will return to the next free location in the Code Relative segment and so on.

The ASEG, DSEG, CSEG pseudo-ops never have operands. If you wish to alter the current value of the location counter, use the ORG pseudo-op.

ORG Pseudo-op

At any time, the value of the location counter may be changed by use of the ORG pseudo-op. The form of the ORG statement is:

```
ORG <exp>
```

where the value of <exp> will be the new value of the location counter in the current mode. All names used in <exp> must be known on pass 1 and the value of <exp> must be either Absolute or in the current mode of the location counter. For example, the statements

```
DSEG
ORG 50
```

set the Data Relative location counter to 50, relative to the start of the Data Relative segment of memory.

LINK-80

The LINK-80 linking loader (see Section 2 of this manual) combines the segments and creates each relocatable module in memory when the program is loaded. The origins of the relocatable segments are not fixed until the program is loaded and the origins are assigned by LINK-80. The command to

LINK-80 may contain user-specified origins through the use of the -P (for Code Relative) and -D (for Data and COMMON segments) switches.

For example, a program that begins with the statements

```

      ASEG
      ORG      800H

```

and is assembled entirely in Absolute mode will always load beginning at 800 unless the ORG statement is changed in the source file. However, the same program, assembled in Code Relative mode with no ORG statement, may be loaded at any specified address by appending the -P:<address> switch to the LINK-80 command string.

5.28 Relocation Before Loading

Two pseudo-ops, .PHASE and .DEPHASE, allow code to be located in one area, but executed only at a different, specified area.

For example:

```

0000'                               .PHASE 100H
0100   CD 0106   FOO:   CALL   BAZ
0103   C3 0007'                               JMP   ZOO
0106   C9                               BAZ:   RET
                               .DEPHASE
0007'   C3 0005   ZOO:   JMP    5

```

All labels within a .PHASE block are defined as the absolute value from the origin of the phase area. The code, however, is loaded in the current area (i.e., from 0' in this example). The code within the block can later be moved to 100H and executed.

6 Macros and Block Pseudo Operations

The macro facilities provided by MACRO-80 include three repeat pseudo operations: repeat (REPT), indefinite repeat (IRP), and indefinite repeat character (IRPC). A macro definition operation (MACRO) is also provided. Each of these four macro operations is terminated by the ENDM pseudo operation.

6.1 Terms

For the purposes of discussion of macros and block

operations, the following terms will be used:

1. <dummy> is used to represent a dummy parameter. All dummy parameters are legal symbols that appear in the body of a macro expansion.
2. <dummylist> is a list of <dummy>s separated by commas.
3. <arglist> is a list of arguments separated by commas. <arglist> must be delimited by angle brackets. Two angle brackets with no intervening characters (<>) or two commas with no intervening characters enter a null argument in the list. Otherwise an argument is a character or series of characters terminated by a comma or >. With angle brackets that are nested inside an <arglist>, one level of brackets is removed each time the bracketed argument is used in an <arglist>. (See example, Section 6.5.) A quoted string is an acceptable argument and is passed as such. Unless enclosed in brackets or a quoted string, leading and trailing spaces are deleted from arguments.
4. <paramlist> is used to represent a list of actual parameters separated by commas. No delimiters are required (the list is terminated by the end of line or a comment), but the rules for entering null parameters and nesting brackets are the same as described for <arglist>. (See example, Section 6.5.)

6.2 REPT-ENDM

```

REPT   <exp>
      .
      .
      .
ENDM

```

The block of statements between REPT and ENDM is repeated <exp> times. <exp> is evaluated as a 16-bit unsigned number. If <exp> contains any external or undefined terms, an error is generated. Example:

```

SET    0
REPT   10      ;generates DB1-DB10
SET    X+1
DB     X
ENDM

```

6.3 IRP-ENDM

```

IRP    <dummy>,<arglist>
      .
      .
      .
ENDM

```

The <arglist> must be enclosed in angle brackets. The number of arguments in the <arglist> determines the number of times the block of statements is repeated. Each repetition substitutes the next item in the <arglist> for every occurrence of <dummy> in the block. If the <arglist> is null (i.e., <>), the block is processed once with each occurrence of <dummy> removed. For example:

```

IRP    X,<1,2,3,4,5,6,7,8,9,10>
DB     X
ENDM

```

generates the same bytes as the REPT example.

6.4 IRPC-ENDM

```

IRPC   <dummy>,string (or <string>)
      .
      .
      .
ENDM

```

IRPC is similar to IRP but the arglist is replaced by a string of text and the angle brackets around the string are optional. The statements in the block are repeated once for each character in the string. Each repetition substitutes the next character in the string for every occurrence of <dummy> in the block. For example:

```

IRPC   X,0123456789
DB     X+1
ENDM

```

generates the same code as the two previous examples.

6.5 MACRO

Often it is convenient to be able to generate a given sequence of statements from various places in a program, even though different parameters may be required each time the sequence is used. This capability is provided by the MACRO statement. The form is

```
<name> MACRO <dummylist>
```

```
.
```

```
.
```

```
.
```

```
ENDM
```

where <name> conforms to the rules for forming symbols. <name> is the name that will be used to invoke the macro. The <dummy>s in <dummylist> are the parameters that will be changed (replaced) each time the MACRO is invoked. The statements before the ENDM comprise the body of the macro. During assembly, the macro is expanded every time it is invoked but, unlike REPT/IRP/IRPC, the macro is not expanded when it is encountered.

The form of a macro call is

```
<name> <paramlist>
```

where <name> is the name supplied in the MACRO definition, and the parameters in <paramlist> will replace the <dummy>s in the MACRO <dummylist> on a one-to-one basis. The number of items in <dummylist> and <paramlist> is limited only by the length of a line. The number of parameters used when the macro is called need not be the same as the number of <dummy>s in <dummylist>. If there are more parameters than <dummy>s, the extras are ignored. If there are fewer, the extra <dummy>s will be made null. The assembled code will contain the macro expansion code after each macro call.

NOTE

A dummy parameter in a MACRO/REPT/IRP/IRPC is always recognized exclusively as a dummy parameter. Register names such as A and B will be changed in the expansion if they were used as dummy parameters.

Here is an example of a MACRO definition that defines a macro called FOO:

```

FOO    MACRO  X
Y      SET    0
        REPT  X
Y      SET    Y+1
        DB    Y
        ENDM
        ENDM

```

This macro generates the same code as the previous three examples when the call

```
FOO    10
```

is executed.

Another example, which generates the same code, illustrates the removal of one level of brackets when an argument is used as an arglist:

```

FOO    MACRO  X
        IRP   Y,<X>
        DB    Y
        ENDM
        ENDM

```

When the call

```
FOO    <1,2,3,4,5,6,7,8,9,10>
```

is made, the macro expansion looks like this:

```

IRP    Y,<1,2,3,4,5,6,7,8,9,10>
DB     Y
ENDM

```

6.6 ENDM

Every REPT, IRP, IRPC and MACRO pseudo-op must be terminated with the ENDM pseudo-op. Otherwise, the 'Unterminated REPT/IRP/IRPC/MACRO' message is generated at the end of each pass. An unmatched ENDM causes an O error.

6.7 EXITM

The EXITM pseudo-op is used to terminate a REPT/IRP/IRPC or MACRO call. When an EXITM is executed, the expansion is exited immediately and any remaining expansion or repetition is not generated. If the block containing the EXITM is nested within another block, the outer level

continues to be expanded.

6.8 LOCAL

LOCAL <dummylist>

The LOCAL pseudo-op is allowed only inside a MACRO definition. When LOCAL is executed, the assembler creates a unique symbol for each <dummy> in <dummylist> and substitutes that symbol for each occurrence of the <dummy> in the expansion. These unique symbols are usually used to define a label within a macro, thus eliminating multiply-defined labels on successive expansions of the macro. The symbols created by the assembler range from ..0001 to ..FFFF. Users will therefore want to avoid the form ..nnnn for their own symbols. If LOCAL statements are used, they must be the first statements in the macro definition.

6.9 Special Macro Operators and Forms

& The ampersand is used in a macro expansion to concatenate text or symbols. A dummy parameter that is in a quoted string will not be substituted in the expansion unless it is immediately preceded by &. To form a symbol from text and a dummy, put & between them. For example:

```
ERRGEN MACRO X
ERROR&X:PUSH B
        MVI B, '&X'
        JMP ERROR
        ENDM
```

In this example, the call ERRGEN A will generate:

```
ERRORA: PUSH B
        MVI B, 'A'
        JMP ERROR
```

;; In a block operation, a comment preceded by two semicolons is not saved as part of the expansion (i.e., it will not appear on the listing even under .LALL). A comment preceded by one semicolon, however, will be preserved and appear in the expansion.

! When an exclamation point is used in an argument, the next character is entered literally (i.e., !; and <;> are equivalent).

NUL NUL is an operator that returns true if its argument (a parameter) is null. The remainder of a line after NUL is considered to be the argument to NUL. The conditional

IF NUL argument

is false if, during the expansion, the first character of the argument is anything other than a semicolon or carriage return. It is recommended that testing for null parameters be done using the IFB and IFNB conditionals.

7 Using Z80 Pseudo-ops

The following Z80 pseudo-ops are valid. The function of each pseudo-op is equivalent to that of its 8080 counterpart.

<u>Z80 pseudo-op</u>	<u>Equivalent 8080 pseudo-op</u>
COND	IFT
ENDC	ENDIF
*EJECT	PAGE
DEFB	DB
DEFS	DS
DEFW	DW
DEFM	DB
DEFL	SET
GLOBAL	PUBLIC
EXTERNAL	EXTRN

The formats, where different, conform to the 8080 format. That is, DEFB and DEFW are permitted a list of arguments (as are DB and DW), and DEFM is permitted a string or numeric argument (as is DB).

8 Sample Assembly

DOS READY

M80

*EXMPL1,TTY:=EXMPL1

MAC80 3.2 PAGE 1

		00100	;CSL3(P1,P2)
		00200	;SHIFT P1 LEFT CIRCULARLY 3 BITS
		00300	;RETURN RESULT IN P2
		00400	ENTRY CSL3
		00450	;GET VALUE OF FIRST PARAMETER
		00500	CSL3:
0000'	7E	00600	MOV A,M
0001'	23	00700	INX H
0002'	66	00800	MOV H,M
0003'	6F	00900	MOV L,A
		01000	;SHIFT COUNT
0004'	06 03	01100	MVI B,3
0006'	AF	01200	LOOP: XRA A
		01300	;SHIFT LEFT
0007'	29	01400	DAD H
		01500	;ROTATE IN CY BIT
0008'	17	01600	RAL
0009'	85	01700	ADD L
000A'	6F	01800	MOV L,A
		01900	;DECREMENT COUNT
000B'	05	02000	DCR B
		02100	;ONE MORE TIME
000C'	C2 0006'	02200	JNZ LOOP
000F'	EB	02300	XCHG
		02400	;SAVE RESULT IN SECOND PARAMETER
0010'	73	02500	MOV M,E
0011'	23	02600	INX H
0012'	72	02700	MOV M,D
0013'	C9	02800	RET
		02900	END

MAC80 3.2 PAGE S

CSL3 0000I' LOOP 0006'

No Fatal error(s)

9 MACRO-80 Errors

MACRO-80 errors are indicated by a one-character flag in column one of the listing file. If a listing file is not being printed on the terminal, each erroneous line is also printed or displayed on the terminal. Below is a list of the MACRO-80 Error Codes:

- A Argument error
Argument to pseudo-op is not in correct format or is out of range (.PAGE 1; .RADIX 1; PUBLIC 1; STAX H; MOV M,M; INX C).
- C Conditional nesting error
ELSE without IF, ENDIF without IF, two ELSEs on one IF.
- D Double Defined symbol
Reference to a symbol which is multiply defined.
- E External error
Use of an external illegal in context (e.g., FOO SET NAME##; MVI A,2-NAME##).
- M Multiply Defined symbol
Definition of a symbol which is multiply defined.
- N Number error
Error in a number, usually a bad digit (e.g., 8Q).
- O Bad opcode or objectionable syntax
ENDM, LOCAL outside a block; SET, EQU or MACRO without a name; bad syntax in an opcode (MOV A:); or bad syntax in an expression (mismatched parenthesis, quotes, consecutive operators, etc.).
- P Phase error
Value of a label or EQU name is different on pass 2.
- Q Questionable
Usually means a line is not terminated properly. This is a warning error (e.g. MOV A,B,).
- R Relocation
Illegal use of relocation in expression, such as abs-rel. Data, code and COMMON areas are relocatable.

- U Undefined symbol
A symbol referenced in an expression is not defined. (For certain pseudo-ops, a V error is printed on pass 1 and a U on pass 2.)
- V Value error
On pass 1 a pseudo-op which must have its value known on pass 1 (e.g., .RADIX, .PAGE, DS, IF, IFE, etc.), has a value which is undefined. If the symbol is defined later in the program, a U error will not appear on the pass 2 listing.

Error Messages:

'No end statement encountered on input file'
No END statement: either it is missing or it is not parsed due to being in a false conditional, unterminated IRP/IRPC/REPT block or terminated macro.

'Unterminated conditional'
At least one conditional is unterminated at the end of the file.

'Unterminated REPT/IRP/IRPC/MACRO'
At least one block is unterminated.

[xx] [No] Fatal error(s) [,xx warnings]
The number of fatal errors and warnings. The message is listed on the CRT and in the list file.

10 Compatibility with Other Assemblers

The \$EJECT and \$TITLE controls are provided for compatibility with INTEL's ISIS assembler. The dollar sign must appear in column 1 only if spaces or tabs separate the dollar sign from the control word. The control

\$EJECT

is the same as the MACRO-80 PAGE pseudo-op.
The control

\$TITLE('text')

is the same as the MACRO-80 SUBTTL <text>
pseudo-op.

The INTEL operands PAGE and INPAGE generate Q errors when used with the MACRO-80 CSEG or DSEG

pseudo-ops. These errors are warnings; the assembler ignores the operands.

When MACRO-80 is entered, the default for the origin is Code Relative 0. With the INTEL ISIS assembler, the default is Absolute 0.

With MACRO-80, the dollar sign (\$) is a defined constant that indicates the value of the location counter at the start of the statement. Other assemblers may use a decimal point or an asterisk. Other constants are defined by MACRO-80 to have the following values:

A=7	B=0	C=1	D=2	E=3
H=4	L=5	M=6	SP=6	PSW=6

11 Format of Listings

On each page of a MACRO-80 listing, the first two lines have the form:

```
[TITLE text]      MAC80 3.2      PAGE x[-y]
[SUBTTL text]
```

where:

1. TITLE text is the text supplied with the TITLE pseudo-op, if one was given in the source program.
2. x is the major page number, which is incremented only when a form feed is encountered in the source file. (When using Microsoft's EDIT-80 text editor, a form feed is inserted whenever a page mark is done.) When the symbol table is being printed, x = 'S'.
3. y is the minor page number, which is incremented whenever the .PAGE pseudo-op is encountered in the source file, or whenever the current page size has been filled.
4. SUBTTL text is the text supplied with the SUBTTL pseudo-op, if one was given in the source program.

Next, a blank line is printed, followed by the first line of output.

A line of output on a MACRO-80 listing has the following form:

```
[crf#]          [error] loc#m      xx      xxxx ...      source
```

If cross reference information is being output, the first item on the line is the cross reference number, followed by a tab.

A one-letter error code followed by a space appears next on the line, if the line contains an error. If there is no error, a space is printed. If there is no cross reference number, the error code column is the first column on the listing.

The value of the location counter appears next on the line. It is a 4-digit hexadecimal number of 6-digit octal number, depending on whether the -O or -H switch was given in the MACRO-80 command string.

The character at the end of the location counter value is the mode indicator. It will be one of the following symbols:

'	Code Relative
"	Data Relative
!	COMMON Relative
<space>	Absolute
*	External

Next, three spaces are printed followed by the assembled code. One-byte values are followed by a space. Two-byte values are followed by a mode indicator. Two-byte values are printed in the opposite order they are stored in, i.e., the high order byte is printed first. Externals are either the offset or the value of the pointer to the next External in the chain.

The remainder of the line contains the line of source code, as it was input.

11.1 Symbol Table Listing

In the symbol table listing, all the macro names in the program are listed alphabetically, followed by all the symbols in the program, listed alphabetically. After each symbol, a tab is printed, followed by the value of the symbol. If the symbol is Public, an I is printed immediately after the value. The next character printed will be one of the following:

U	Undefined symbol.
C	COMMON block name. (The "value" of the COMMON block is its length (number of bytes) in hexadecimal or octal.)
*	External symbol.
<space>	Absolute value.
'	Program Relative value.
"	Data Relative value.
!	COMMON Relative value.

12 Cross Reference Facility

The Cross Reference Facility is invoked by typing CREF80 at TRSDOS command level. In order to generate a cross reference listing, the assembler must output a special listing file with embedded control characters. The MACRO-80 command string tells the assembler to output this special listing file. (See Section 5.26 for the .CREF and .XCREF pseudo-ops.) -C is the cross reference switch. When the -C switch is encountered in a MACRO-80 command string, the assembler opens a /CRF file instead of a /LST file.

Examples:

*=TEST-C	Assemble file TEST/MAC and create object file TEST/REL and cross reference file TEST/CRF.
*T,U=TEST-C	Assemble file TEST/MAC and create object file T/REL and cross reference file U/CRF.

When the assembler is finished, it is necessary to call the cross reference facility by typing CREF80. (CREF80 is on diskette #1) CREF80 command format is:

*listing file=source file

The default extension for the source file is /CRF. the -L switch is ignored, and any other switch will cause an error message to be sent to the terminal. Possible command strings are:

*=TEST Examine file TEST/CRF and
generate a cross reference
listing file TEST/LST.

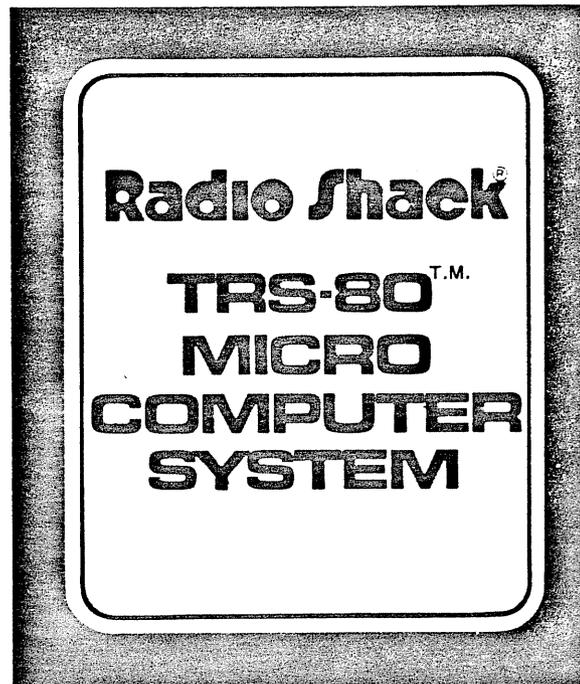
*T=TEST Examine file TEST/CRF and
generate a cross reference
listing file T/LST.

Cross reference listing files differ from ordinary
listing files in that:

1. Each source statement is numbered with a cross
reference number.
2. At the end of the listing, variable names
appear in alphabetic order along with the
numbers of the lines on which they are
referenced or defined. Line numbers on which
the symbol is defined are flagged with '#'.



**LINK-80
Reference
Manual**



**For Use with the TRS-80
Disk Operating System (TRSDOS)**



CONTENTS

1	Running LINK-80	4
	1.1 LINK-80 Commands	4
	1.2 LINK-80 Switches	5
2	Sample Link	7
3	Format of LINK Compatible Object Files.	8
4	LINK-80 Error Messages.	10
5	Program Break Information	11

LINK-80 Linking Loader

The LINK-80 Linking Loader takes the relocatable object files generated by the FORTRAN compiler and MACRO-80 assembler and loads them into memory in a form that can be executed. In addition, LINK-80 automatically searches the system library (FORLIB) and loads the library routines needed to satisfy any undefined global references (i.e., calls generated by the compiled program to subroutines in the system library).

LINK-80 provides the user with several loading options. Programs may be loaded at user-specified locations, and program areas and data areas may be separated in memory. A memory image of the executable file produced by LINK-80 can be written to disk. The default extension for the name of the executable file is /CMD.

1 Running LINK-80

When you give TRSDOS the command

L80

(diskette #2 must be in the disk drive), you are running the LINK-80 linking loader. When the loader is ready to accept commands, it prompts the user with an asterisk. The loader will exit back to TRSDOS after a command containing an E or G switch (see Section 1.1), or after a <break> is done at command level.

Command lines are also supported by LINK-80.

1.1 LINK-80 Commands

A command to LINK-80 consists of a string of filenames and/or switches. The command format is:

```
[filename1][-switch1][,filename2][-switch2]...
```

All filenames must be in TRSDOS filename format.

After LINK-80 receives the command, it will load or search (see the S switch) the specified files. Then it will list all the symbols that remained undefined, with each followed by an asterisk.

Example:

```
*MAIN
DATA      5200      5300
SUBR1*    (SUBR1 is undefined)
DATA      5200      5300
*SUBR1
*-G      (Starts Execution - see below)
```

Typically, to execute a FORTRAN program and subroutines, the user types the list of filenames followed by -G (begin execution). Before execution begins, LINK-80 will always search the system library (FORLIB/REL) to satisfy any unresolved external references. If you wish to first search libraries of your own, append the filenames that are followed by -S to the end of the loader command string.

1.2 LINK-80 Switches

A number of switches may be given in the LINK-80 command string to specify actions affecting the loading process. Each switch must be preceded by a dash (-). These switches are:

<u>Switch</u>	<u>Action</u>
R	Reset. Put loader back in its initial state. Use -R if you loaded the wrong file by mistake and want to restart. -R takes effect as soon as it is encountered in a command string.
E or E:Name	Exit LINK-80 and return to the Operating System. The system library will be searched on the current disk to satisfy any existing undefined globals. The optional form E:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program. Use -E to load a program and exit back to the monitor.
G or G:Name	Start execution of the program as soon as the current command line has been interpreted. The system

library will be searched on the current disk to satisfy any existing undefined globals. Before execution actually begins, LINK-80 prints two numbers and a BEGIN EXECUTION message. The two numbers are the start address and the address of the next available byte. The optional form G:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program.

N If a <filename>-N is specified, the program will be saved on disk under the selected name (with a default extension of CMD) when a -E or -G is done.

P and D -P and -D allow the origin(s) to be set for the next program loaded. -P and -D take effect when seen (not deferred), and they have no effect on programs already loaded. The form is -P:<address> or -D:<address>, where <address> is the desired origin in the current typeout radix. (Default radix is hexadecimal. -O sets radix to octal; -H to hex.) LINK-80 does a default -P:<link origin> (i.e., 5200).

If no -D is given, data areas are loaded before program areas for each module. If a -D is given, all Data and Common areas are loaded starting at the data origin and the program area at the program origin. Example:

```
*-P:200,FOO
Data    200    300
*-R
*-P:200 -D:400,FOO
Data    400    480
Program 200    280
```

U List the origin and end of the program and data area and all undefined globals as soon as the current command line has been interpreted. The program informa-

tion is only printed if a -D has been done. Otherwise, the program is stored in the data area.

M List the origin and end of the program and data area, all defined globals and their values, and all undefined globals followed by an asterisk. The program information is only printed if a -D has been done. Otherwise, the program is stored in the data area.

S Search the filename immediately preceding the -S in the command string to satisfy any undefined globals.

Examples:

*-M List all globals

*MYPROG,SUBROT,MYLIB-S
Load MYPROG.REL and SUBROT.REL and then search MYLIB.REL to satisfy any remaining undefined globals.

*-G Begin execution of main program

2 Sample Link

```
DOS READY
L80
*EXAMPL,EXMPL1-G
DATA 5200 52AC
[5200 52AC]
[BEGIN EXECUTION]

      1792          14336
      14336        -16383
     -16383           14
           14         112
           112        896
DOS READY
```

3 Format of LINK Compatible Object Files

NOTE

Section 3 is reference material for users who wish to know the load format of LINK-80 relocatable object files. Most users will want to skip this section, as it does not contain material necessary to the operation of the package.

LINK-compatible object files consist of a bit stream. Individual fields within the bit stream are not aligned on byte boundaries, except as noted below. Use of a bit stream for relocatable object files keeps the size of object files to a minimum, thereby decreasing the number of disk reads/writes.

There are two basic types of load items: Absolute and Relocatable. The first bit of an item indicates one of these two types. If the first bit is a 0, the following 8 bits are loaded as an absolute byte. If the first bit is a 1, the next 2 bits are used to indicate one of four types of relocatable items:

- 00 Special LINK item (see below).
- 01 Program Relative. Load the following 16 bits after adding the current Program base.
- 10 Data Relative. Load the following 16 bits after adding the current Data base.
- 11 Common Relative. Load the following 16 bits after adding the current Common base.

Special LINK items consist of the bit stream 100 followed by:

a four-bit control field

an optional A field consisting of a two-bit address type that is the same as the two-bit field above except 00 specifies absolute address

an optional B field consisting

of 3 bits that give a symbol length and up to 8 bits for each character of the symbol

A general representation of a special LINK item is:

```

1 00 xxxx yy nn      zzz + characters of symbol name
-----
          A field          B field

```

xxxx Four-bit control field (0-15 below)
yy Two-bit address type field
nn Sixteen-bit value
zzz Three-bit symbol length field

The following special types have a B-field only:

```

0      Entry symbol (name for search)
1      Select COMMON block
2      Program name
3      Request library search
4      Reserved for future expansion

```

The following special LINK items have both an A field and a B field:

```

5      Define COMMON size
6      Chain external (A is head of address chain,
B is name of external symbol)
7      Define entry point (A is address, B is name)
8      Reserved for future expansion

```

The following special LINK items have an A field only:

```

9      External + offset. The A value will
be added to the two bytes starting
at the current location counter
immediately before execution.
10     Define size of Data area (A is size)
11     Set loading location counter to A
12     Chain address. A is head of chain,
replace all entries in chain with current
location counter.
The last entry in the chain has an
address field of absolute zero.
13     Define program size (A is size)
14     End program (forces to byte boundary)

```

The following special Link item has neither an A nor a B field:

```

15     End file

```

4 LINK-80 Error Messages

LINK-80 has the following error messages:

- ?No Start Address A -G switch was issued,
but no main program
had been loaded.
- ?Loading Error The last file given for input
was not a properly formatted
LINK-80 object file.
- ?Out of Memory Not enough memory to load
program.
- ?Command Error Unrecognizable LINK-80
command.
- ?<file> Not Found <file>, as given in the command
string, did not exist.
- %2nd COMMON Larger /XXXXXX/
The first definition of
COMMON block /XXXXXX/ was not
the largest definition. Re-
order module loading sequence
or change COMMON block
definitions.
- %Mult. Def. Global YYYYYY
More than one definition for
the global (internal) symbol
YYYYYY was encountered during
the loading process.
- %Overlying [Program] Area [,Start = xxxx
 [Data] [,Public = <symbol name>(xxxx)
 [,External = <symbol name>(xxxx)]
A -D or -P will cause already
loaded data to be destroyed.
- ?Intersecting [Program] Area
 [Data] The program and data area
intersect and an address or
external chain entry is in
this intersection. The
final value cannot be con-
verted to a current value
since it is in the area
intersection.

?Start Symbol - <name> - Undefined

After a -E: or -G: is given,
the symbol specified was not
defined.

Origin Above] Loader Memory, Move Anyway (Y or N)?
 Below]

After a -E or -G was given,
either the data or program
area has an origin or top
which lies outside loader
memory (i.e., loader origin
to top of memory). If a
Y <cr> is given, LINK-80
will move the area and con-
tinue. If anything else is
given, LINK-80 will exit.
In either case, if a -N was
given, the image will already
have been saved.

?Can't Save Object File

A disk error occurred when
the file was being saved.

5 Program Break Information

LINK-80 stores the address of the first free
location in a global symbol called \$MEMORY if that
symbol has been defined by a program loaded.
\$MEMORY is set to the top of the data area +1.

NOTE

If -D is given and the data origin is less
than the program area, the user must be
sure there is enough room to keep the
program from being destroyed. This is
particularly true with the disk driver for
FORTRAN-80 which uses \$MEMORY to allocate
disk buffers and FCB's.

APPENDIX A

FORTRAN-80 Library Subroutines

The FORTRAN-80 library contains a number of subroutines that may be referenced by the user from FORTRAN or assembly programs. In the following descriptions, \$AC refers to the floating accumulator; \$AC is the address of the low byte of the mantissa. \$AC+3 is the address of the exponent. \$DAC refers to the DOUBLE PRECISION accumulator; \$DAC is the address of the low byte of the mantissa. \$DAC+7 is the address of the DOUBLE PRECISION exponent.

All arithmetic routines (addition, subtraction, multiplication, division, exponentiation) adhere to the following calling conventions.

1. Argument 1 is passed in the registers:
 - Integer in [HL]
 - Real in \$AC
 - Double in \$DAC
2. Argument 2 is passed either in registers, or in memory depending upon the type:
 - a. Integers are passed in [HL], or [DE] if [HL] contains Argument 1.
 - b. Real and Double Precision values are passed in memory pointed to by [HL]. ([HL] points to the low byte of the mantissa.)

The following arithmetic routines are contained in the Library:

<u>Function</u>	<u>Name</u>	<u>Argument 1 Type</u>	<u>Argument 2 Type</u>
Addition	\$AA	Real	Integer
	\$AB	Real	Real
	\$AQ	Double	Integer
	\$AR	Double	Real
	\$AU	Double	Double
Division	\$D9	Integer	Integer
	\$DA	Real	Integer
	\$DB	Real	Real
	\$DQ	Double	Integer
	\$DR	Double	Real
	\$DU	Double	Double
Exponentiation	\$E9	Integer	Integer
	\$EA	Real	Integer
	\$EB	Real	Real
	\$EQ	Double	Integer
	\$ER	Double	Real
	\$EU	Double	Double
Multiplication	\$M9	Integer	Integer
	\$MA	Real	Integer
	\$MB	Real	Real
	\$MQ	Double	Integer
	\$MR	Double	Real
	\$MU	Double	Double
Subtraction	\$SA	Real	Integer
	\$SB	Real	Real
	\$SQ	Double	Integer
	\$SR	Double	Real
	\$SU	Double	Double

Additional Library routines are provided for converting between value types. Arguments are always passed to and returned by these conversion routines in the appropriate registers:

Logical in [A]

Integer in [HL]

Real in \$AC

Double in \$DAC

<u>Name</u>	<u>Function</u>
\$CA	Integer to Real
\$CC	Integer to Double
\$CH	Real to Integer
\$CJ	Real to Logical
\$CK	Real to Double
\$CX	Double to Integer
\$CY	Double to Real
\$CZ	Double to Logical

Z-80 Instruction Set and Appendix



CONTENTS

8-Bit Load Group	1
16-Bit Load Group	12
Exchange, Block Transfer and Search Group	22
8-Bit Arithmetic and Logic Group	31
General Purpose Arithmetic and CPU Control Group	44
16-Bit Arithmetic Group	51
Rotate and Shift Group	57
Bit Set, Reset and Test Group	69
Jump Group	74
Call and Return Group	80
Input and Output Group	86

APPENDIX

Z-80 Hardware Configuration	96
Numeric List of Instruction Set	102
Z-80 CPU Register Configuration	107
Alphabetic List of Instruction Set	108
Level II BASIC ROM Calls	113

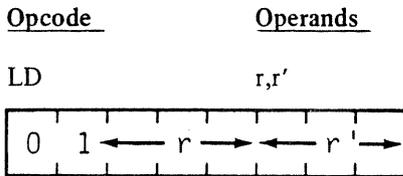


8 BIT LOAD GROUP

LD r, r'

Operation: $r \leftarrow r'$

Format:



Description:

The contents of any register r' are loaded into any other register r . Note: r, r' identifies any of the registers A, B, C, D, E, H, or L, assembled as follows in the object code:

Register	r, r'
A	= 111
B	= 000
C	= 001
D	= 010
E	= 011
H	= 100
L	= 101

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.0

Condition Bits Affected: None

Example:

If the H register contains the number 8AH, and the E register contains 10H, the instruction

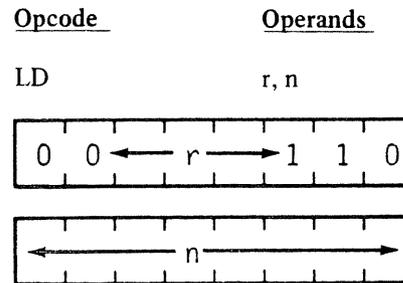
LD H, E

would result in both registers containing 10H.

LD r, n

Operation: $r \leftarrow n$

Format:



Description:

The eight-bit integer n is loaded into any register r , where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	= 111
B	= 000
C	= 001
D	= 010
E	= 011
H	= 100
L	= 101

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

After the execution of

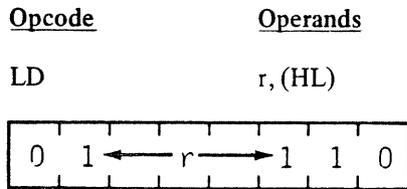
LD E, A5H

the contents of register E will be A5H.

LD r, (HL)

Operation: $r \leftarrow (HL)$

Format:



Description:

The eight-bit contents of memory location (HL) are loaded into register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	= 111
B	= 000
C	= 001
D	= 010
E	= 011
H	= 100
L	= 101

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

If register pair HL contains the number 75A1H, and memory address 75A1H contains the byte 58H, the execution of

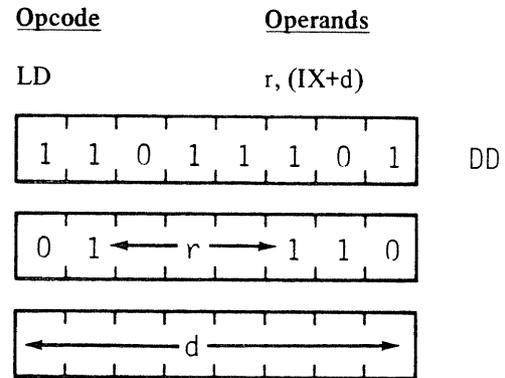
LD C, (HL)

will result in 58H in register C.

LD r, (IX+d)

Operation: $r \leftarrow (IX+d)$

Format:



Description:

The operand (IX+d) (the contents of the Index Register IX summed with a displacement integer d) is loaded into register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	= 111
B	= 000
C	= 001
D	= 010
E	= 011
H	= 100
L	= 101

M CYCLES: 5 T STATES: 19(4,4,3,5,3) 4 MHZ E.T.: 4.75

Condition Bits Affected: None

Example:

If the Index Register IX contains the number 25AFH, the instruction

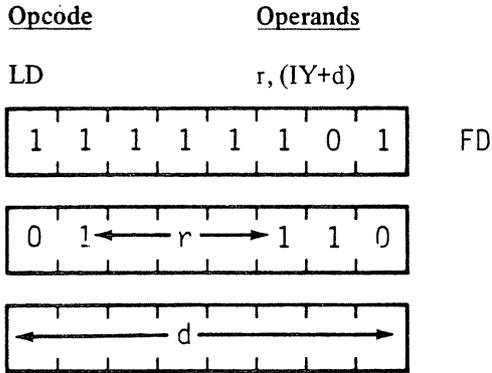
LD B, (IX+19H)

will cause the calculation of the sum 25AFH + 19H, which points to memory location 25C8H. If this address contains byte 39H, the instruction will result in register B also containing 39H.

LD r, (IY+d)

Operation: $r \leftarrow (IY+d)$

Format:



Description:

The operand (IY+d) (the contents of the Index Register IY summed with a displacement integer d) is loaded into register r, where r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	= 111
B	= 000
C	= 001
D	= 010
E	= 011
H	= 100
L	= 101

M CYCLES: 5 T STATES: 19(4,4,3,5,3) 4 MHZ E.T.: 4.75

Condition Bits Affected: None

Example:

If the Index Register IY contains the number 25AFH, the instruction

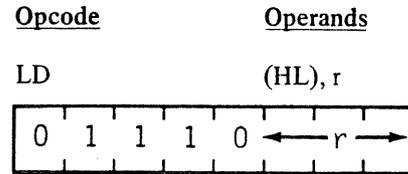
LD B, (IY+19H)

will cause the calculation of the sum 25AFH + 19H, which points to memory location 25C8H. If this address contains byte 39H, the instruction will result in register B also containing 39H.

LD (HL), r

Operation: (HL) ← r

Format:



Description:

The contents of register r are loaded into the memory location specified by the contents of the HL register pair. The symbol r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

Register	r
A	= 111
B	= 000
C	= 001
D	= 010
E	= 011
H	= 100
L	= 101

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

If the contents of register pair HL specifies memory location 2146H, and the B register contains the byte 29H, after the execution of

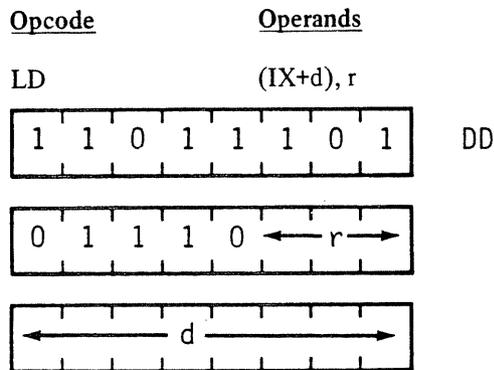
LD (HL), B

memory address 2146H will also contain 29H.

LD (IX+d), r

Operation: (IX+d) ← r

Format:



Description:

The contents of register r are loaded into the memory address specified by the contents of Index Register IX summed with d, a two's complement displacement integer. The symbol r identifies register A, B, C, D, E, H or L, assembled as follows in the object code:

<u>Register</u>	<u>r</u>
A =	111
B =	000
C =	001
D =	010
E =	011
H =	100
L =	101

M CYCLES: 5 T STATES: 19(4,4,3,5,3) 4 MHZ E.T.: 4.75

Condition Bits Affected: None

Example:

If the C register contains the byte 1CH, and the Index Register IX contains 3100H, then the instruction

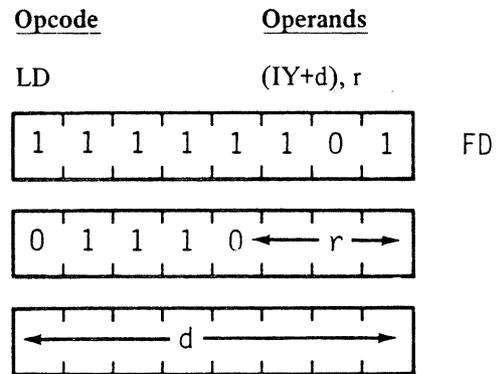
LD (IX+6H), C

will perform the sum 3100H + 6H and will load 1CH into memory location 3106H.

LD (IY+d), r

Operation: (IY+d) ← r

Format:



Description:

The contents of register r are loaded into the memory address specified by the sum of the contents of the Index Register IY and d, a two's complement displacement integer. The symbol r is specified according to the following table.

<u>Register</u>	<u>r</u>
A =	111
B =	000
C =	001
D =	010
E =	011
H =	100
L =	101

M CYCLES: 5 T STATES: 19(4,4,3,5,3) 4 MHZ E.T.: 4.75

Condition Bits Affected: None

Example:

If the C register contains the byte 48H, and the Index Register IY contains 2A11H, then the instruction

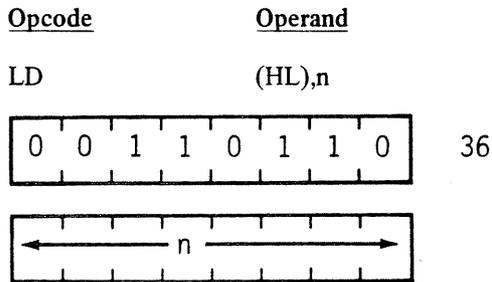
LD (IY+4H), C

will perform the sum 2A11H + 4H, and will load 48H into memory location 2A15.

LD (HL), n

Operation: (HL) ← n

Format:



Description:

Integer n is loaded into the memory address specified by the contents of the HL register pair.

M CYCLES: 3 T STATES: 10(4,3,3) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If the HL register pair contains 4444H, the instruction

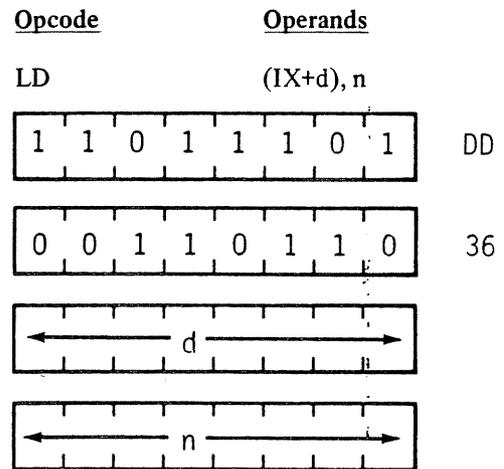
LD (HL), 28H

will result in the memory location 4444H containing the byte 28H.

LD (IX+d), n

Operation: (IX+d) ← n

Format:



Description:

The n operand is loaded into the memory address specified by the sum of the contents of the Index Register IX and the two's complement displacement operand d.

M CYCLES: 5 T STATES: 19(4,4,3,5,3) 4 MHZ E.T.: 4.75

Condition Bits Affected: None

Example:

If the Index Register IX contains the number 219AH the instruction

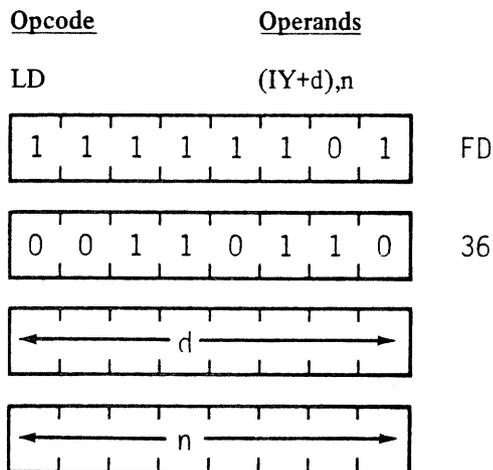
LD (IX+5H), 5AH

would result in the byte 5AH in the memory address 219FH.

LD (IY+d), n

Operation: (IY+d) ← n

Format:



Description:

Integer n is loaded into the memory location specified by the contents of the Index Register summed with a displacement integer d.

M CYCLES: 5 T STATES: 19(4,4,3,5,3) 4 MHZ E.T.: 4.75

Condition Bits Affected: None

Example:

If the Index Register IY contains the number A940H, the instruction

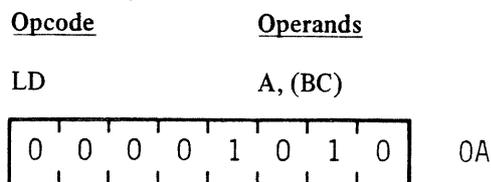
LD (IY+10H), 97H

would result in byte 97 in memory location A950H.

LD A, (BC)

Operation: A ← (BC)

Format:



Description:

The contents of the memory location specified by the contents of the BC register pair are loaded into the Accumulator.

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

If the BC register pair contains the number 4747H, and memory address 4747H contains the byte 12H, then the instruction

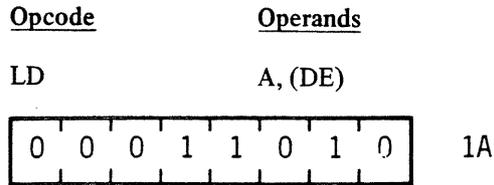
LD A, (BC)

will result in byte 12H in register A.

LD A, (DE)

Operation: $A \leftarrow (DE)$

Format:



Description:

The contents of the memory location specified by the register pair DE are loaded into the Accumulator.

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

If the DE register pair contains the number 30A2H and memory address 30A2H contains the byte 22H, then the instruction

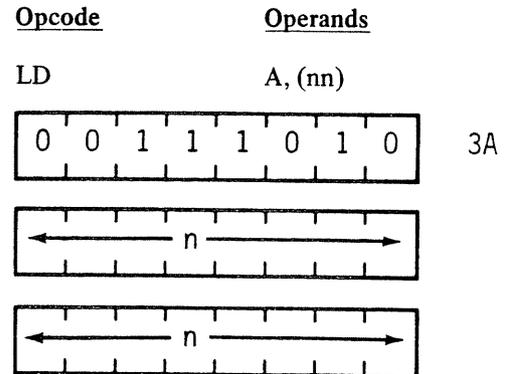
LD A, (DE)

will result in byte 22H in register A.

LD A, (nn)

Operation: $A \leftarrow (nn)$

Format:



Description:

The contents of the memory location specified by the operands nn are loaded into the Accumulator. The first n operand is the low order byte of a two-byte memory address.

M CYCLES: 4 T STATES: 13(4,3,3,3) 4 MHZ E.T.: 3.25

Condition Bits Affected: None

Example:

If the contents of nn is number 8832H, and the content of memory address 8832H is byte 04H, after the instruction

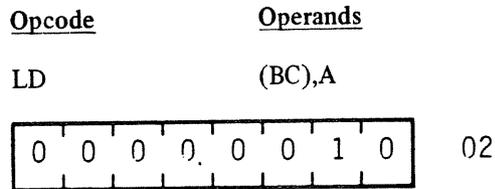
LD A, (nn)

byte 04H will be in the Accumulator.

LD (BC), A

Operations: (BC) ← A

Format:



Description:

The contents of the Accumulator are loaded into the memory location specified by the contents of the register pair BC.

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

If the Accumulator contains 7AH and the BC register pair contains 1212H the instruction

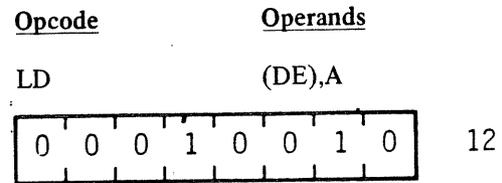
LD (BC), A

will result in 7AH being in memory location 1212H.

LD (DE), A

Operation: (DE) ← A

Format:



Description:

The contents of the Accumulator are loaded into the memory location specified by the DE register pair.

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

If the contents of register pair DE are 1128H, and the Accumulator contains byte A0H, the instruction

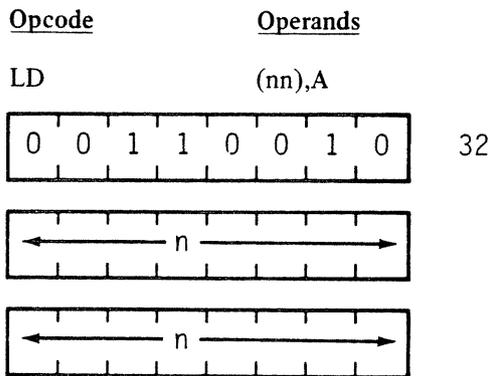
LD (DE), A

will result in A0H being in memory location 1128H.

LD (nn), A

Operation: (nn) ← A

Format:



Description:

The contents of the Accumulator are loaded into the memory address specified by the operands nn. The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 4 T STATES: 13(4,3,3,3) 4 MHZ E.T.: 3.25

Condition Bits Affected: None

Example:

If the contents of the Accumulator are byte D7H, after the execution of

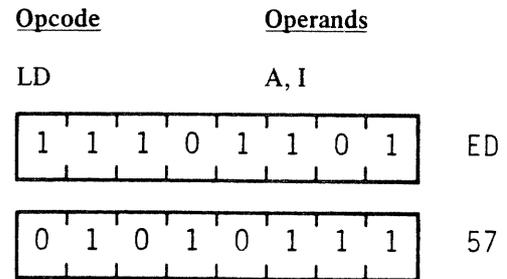
LD (3141H), A

D7H will be in memory location 3141H.

LD A, I

Operation: A ← I

Format:



Description:

The contents of the Interrupt Vector Register I are loaded into the Accumulator.

M CYCLES: 2 T STATES: 9(4,5) 4 MHZ E.T.: 2.25

Condition Bits Affected:

S: Set if I-Reg. is negative; reset otherwise
 Z: Set if I-Reg. is zero; reset otherwise
 H: Reset
 P/V: Contains contents of IFF2
 N: Reset
 C: Not affected

Example:

If the Interrupt Vector Register contains the byte 4AH, after the execution of

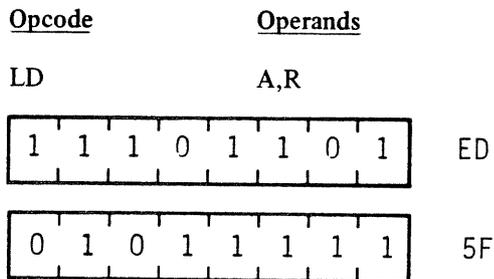
LD A, I

the accumulator will also contain 4AH.

LD A, R

Operation: $A \leftarrow R$

Format:



Description:

The contents of Memory Refresh Register R are loaded into the Accumulator.

M CYCLES: 2 T STATES: 9(4,5) 4 MHZ E.T.: 2.25

Condition Bits Affected:

S: Set if R-Reg. is negative; reset otherwise
Z: Set if R-Reg. is zero; reset otherwise
H: Reset
P/V: Contains contents of IFF2
N: Reset
C: Not affected

Example:

If the Memory Refresh Register contains the byte 4AH, after the execution of

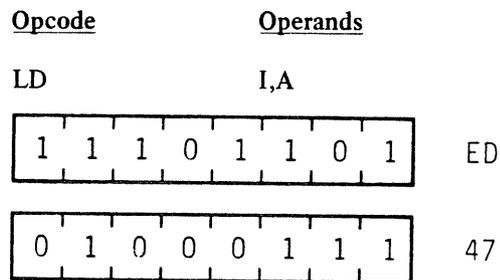
LD A,R

the Accumulator will also contain 4AH.

LD I, A

Operation: $I \leftarrow A$

Format:



Description:

The contents of the Accumulator are loaded into the Interrupt Control Vector Register, I.

M CYCLES: 2 T STATES: 9(4,5) 4 MHZ E.T.: 2.25

Condition Bits Affected: None

Example:

If the Accumulator contains the number 81H, after the instruction

LD I,A

the Interrupt Vector Register will also contain 81H.

LD R, A

Operation: R ← A

Format:

Opcode

Operands

LD

R,A,

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

ED

0	1	0	0	1	1	1	1
---	---	---	---	---	---	---	---

4F

Description:

The contents of the Accumulator are loaded into the Memory Refresh register R.

M CYCLES: 2 T STATES: 9(4,5) 4 MHZ E.T.: 2.25

Condition Bits Affected: None

Example:

If the Accumulator contains the number B4H, after the instruction

LD R, A

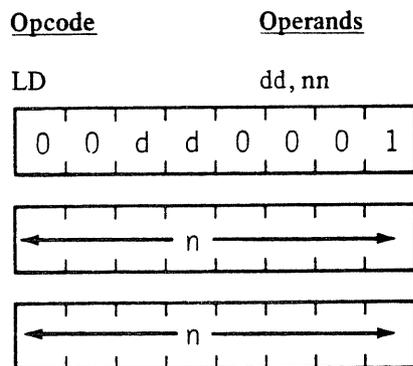
the Memory Refresh Register will also contain B4H.

16 BIT LOAD GROUP

LD dd, nn

Operation: dd ← nn

Format:



Description:

The two-byte integer nn is loaded into the dd register pair, where dd defines the BC, DE, HL, or SP register pairs, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand in the assembled object code is the low order byte.

M CYCLES: 3 T STATES: 10(4,3 3) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

After the execution of

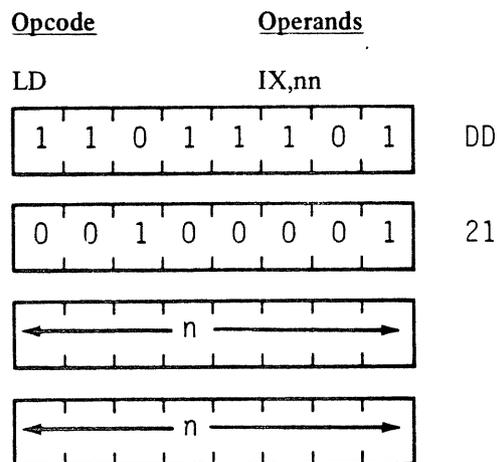
LD HL, 5000H

the contents of the HL register pair will be 5000H.

LD IX, nn

Operation: IX ← nn

Format:



Description:

Integer nn is loaded into the Index Register IX. The first n operand in the assembled object code above is the low order byte.

M CYCLES: 4 T STATES: 14(4,4,3,3) 4 MHZ E.T.: 3.50

Condition Bits Affected: None

Example:

After the instruction

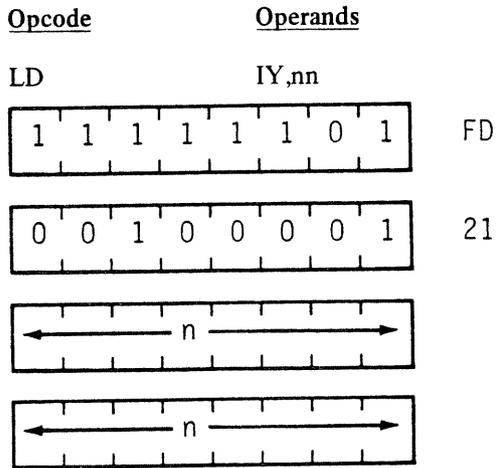
LD IX, 45A2H

the Index Register will contain integer 45A2H.

LD IY, nn

Operation: IY ← nn

Format:



Description:

Integer nn is loaded into the Index Register IY. The first n operand in the assembled object code above is the low order byte.

M CYCLES: 4 T STATES: 14(4,4,3,3) 4 MHZ E.T.: 3.50

Condition Bits Affected: None

Example:

After the instruction:

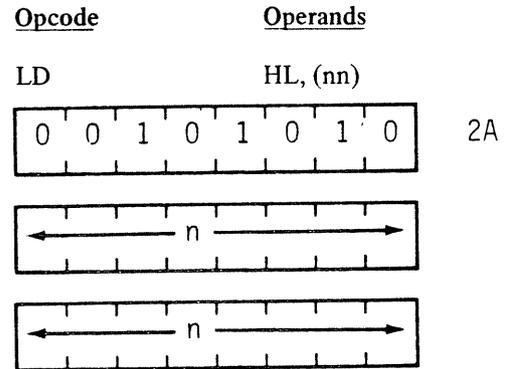
```
LD IY,7733H
```

the Index Register IY will contain the integer 7733H.

LD HL, (nn)

Operation: H ← (nn+1), L ← (nn)

Format:



Description:

The contents of memory address nn are loaded into the low order portion of register pair HL (register L), and the contents of the next highest memory address nn+1 are loaded into the high order portion of HL (register H). The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 5 T STATES: 16(4,3,3,3,3) 4 MHZ E.T.: 4.00

Condition Bits Affected: None

Example:

If address 4545H contains 37H and address 4546H contains A1H after the instruction

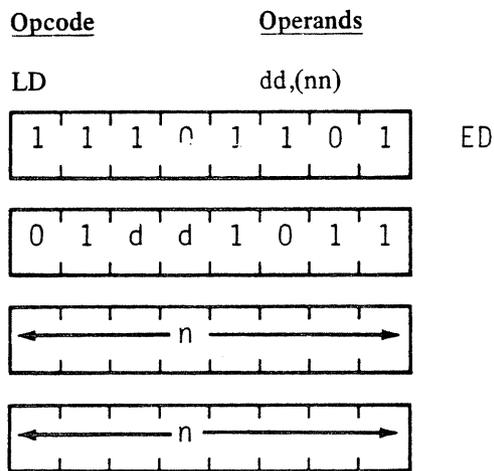
```
LD HL, (4545H)
```

the HL register pair will contain A137H.

LD dd, (nn)

Operation: $dd_H \leftarrow (nn+1), dd_L \leftarrow (nn)$

Format:



Description:

The contents of address nn are loaded into the low order portion of register pair dd, and the contents of the next highest memory address nn+1 are loaded into the high order portion of dd. Register pair dd defines BC, DE, HL, or SP register pairs, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand in the assembled object code above is the low order byte of (nn).

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3) 4 MHZ E.T.: 5.00

Condition Bits Affected: None

Example:

If Address 2130H contains 65H and address 2131M contains 78H after the instruction

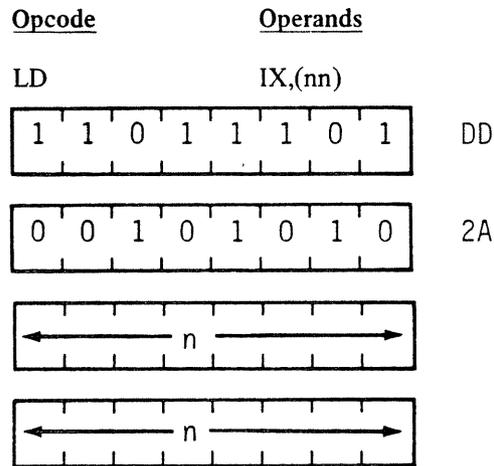
```
LD BC, (2130H)
```

the BC register pair will contain 7865H.

LD IX, (nn)

Operation: $IX_H \leftarrow (nn+1), IX_L \leftarrow (nn)$

Format:



Description:

The contents of the address nn are loaded into the low order portion of Index Register IX, and the contents of the next highest memory address nn+1 are loaded into the high order portion of IX. The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3) 4 MHZ E.T.: 5.00

Condition Bits Affected: None

Example:

If address 6666H contains 92H and address 6667H contains DAH, after the instruction

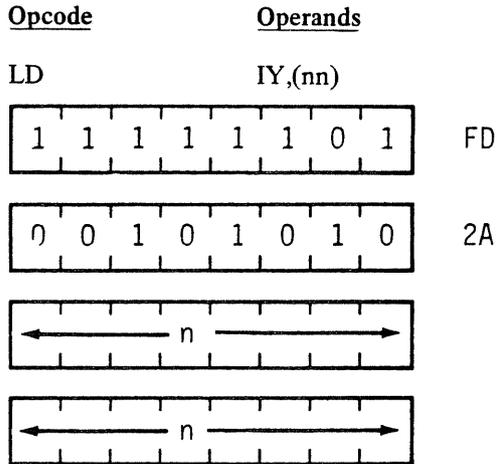
```
LD IX, (6666H)
```

the Index Register IX will contain DA92H.

LD IY, (nn)

Operation: $IY_H \leftarrow (nn+1), IY_L \leftarrow (nn)$

Format:



Description:

The contents of address nn are loaded into the low order portion of Index Register IY, and the contents of the next highest memory address nn+1 are loaded into the high order portion of IY. The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3) 4 MHZ E.T.: 5.00

Condition Bits Affected: None

Example:

If address 6666H contains 92H and address 6667H contains DAH, after the instruction

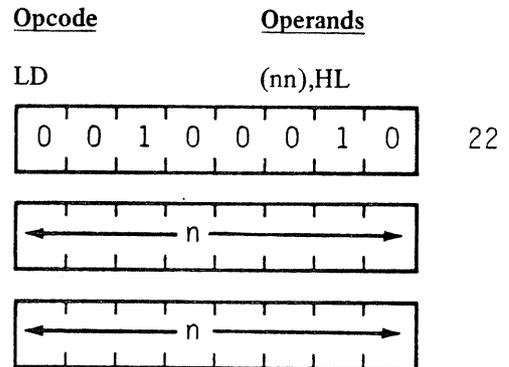
LD IY, (6666H)

the Index Register IY will contain DA92H.

LD (nn), HL

Operation: $(nn+1) \leftarrow H, (nn) \leftarrow L$

Format:



Description:

The contents of the low order portion of register pair HL (register L) are loaded into memory address nn, and the contents of the high order portion of HL (register H) are loaded into the next highest memory address nn+1. The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 5 T STATES: 16(4,3,3,3,3) 4 MHZ E.T.: 4.00

Condition Bits Affected: None

Example:

If the content of register pair HL is 483AH, after the instruction

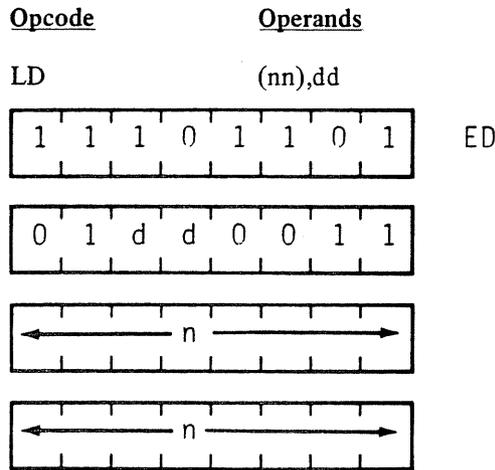
LD (B229H), HL

address B229H) will contain 3AH, and address B22AH will contain 48H.

LD (nn), dd

Operation: $(nn+1) \leftarrow dd_H, (nn) \leftarrow dd_L$

Format:



Description:

The low order byte of register pair dd is loaded into memory address nn ; the upper byte is loaded into memory address nn+1 . Register pair dd defines either BC, DE, HL, or SP, assembled as follows in the object code:

Pair	dd
BC	00
DE	01
HL	10
SP	11

The first n operand in the assembled object code is the low order byte of a two byte memory address.

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3) 4 MHZ E.T.: 5.00

Condition Bits Affected: None

Example:

If register pair BC contains the number 4644H, the instruction

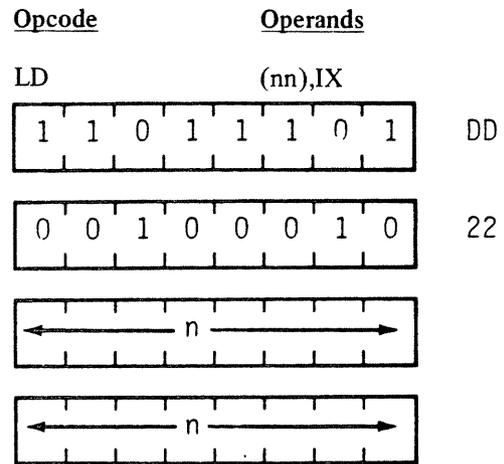
LD (1000H),BC

will result in 44H in memory location 1000H, and 46H in memory location 1001H.

LD (nn), IX

Operation: $(nn+1) \leftarrow IX_H, (nn) \leftarrow IX_L$

Format:



Description:

The low order byte in Index Register IX is loaded into memory address nn ; the upper order byte is loaded into the next highest address nn+1 . The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3) 4 MHZ E.T.: 5.00

Condition Bits Affected: None

Example:

If the Index Register IX contains 5A30H, after the instruction

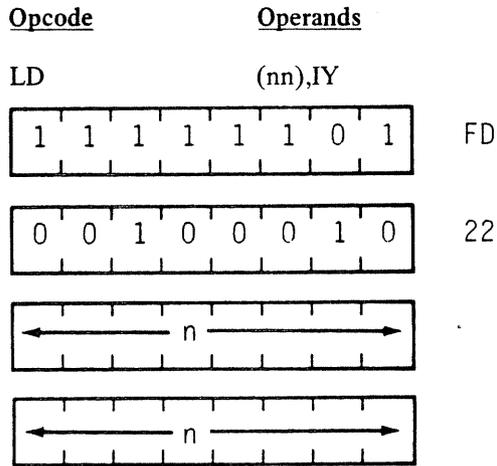
LD (4392H),IX

memory location 4392H will contain number 30H and location 4393H will contain 5AH.

LD (nn), IY

Operation: $(nn+1) \leftarrow IY_H, (nn) \leftarrow IY_L$

Format:



Description:

The low order byte in Index Register IY is loaded into memory address nn ; the upper order byte is loaded into memory location nn+1. The first n operand in the assembled object code above is the low order byte of nn.

M CYCLES: 6 T STATES: 20(4,4,3,3,3,3) 4 MHZ E.T.: 5.00

Condition Bits Affected: None

Example:

If the Index Register IY contains 4174H after the instruction

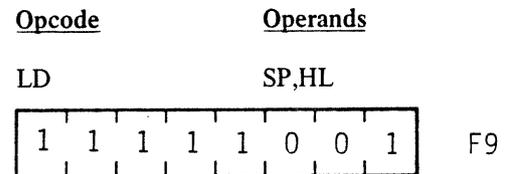
LD 8838H, IY

memory location 8838H will contain number 74H and memory location 8839H will contain 41H.

LD SP, HL

Operation: $SP \leftarrow HL$

Format:



Description:

The contents of the register pair HL are loaded into the Stack Pointer SP.

M CYCLES: 1 T STATES: 6 4 MHZ E.T.: 1.50

Condition Bits Affected: None

Example:

If the register pair HL contains 442EH, after the instruction

LD SP,HL

the Stack Pointer will also contain 442EH.

LD SP, IX

Operation: $SP \leftarrow IX$

Format:

<u>Opcode</u>	<u>Operands</u>									
LD	SP,IX									
<table border="1"><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	1	1	0	1	1	1	0	1		DD
1	1	0	1	1	1	0	1			
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table>	1	1	1	1	1	0	0	1		F9
1	1	1	1	1	0	0	1			

Description:

The two byte contents of Index Register IX are loaded into the Stack Pointer SP.

M CYCLES: 2 T STATES: 10(4,6) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of the Index Register IX are 98DAH, after the instruction

LD SP, IX

the contents of the Stack Pointer will also be 98DAH.

LD SP, IY

Operation: $SP \leftarrow IY$

Format:

<u>Opcode</u>	<u>Operands</u>									
LD	SP,IY									
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	1	1	1	1	1	1	0	1		FD
1	1	1	1	1	1	0	1			
<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table>	1	1	1	1	1	0	0	1		F9
1	1	1	1	1	0	0	1			

Description:

The two byte contents of Index Register IY are loaded into the Stack Pointer SP.

M CYCLES: 2 T STATES: 10(4,6) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If Index Register IY contains the integer A227H, after the instruction

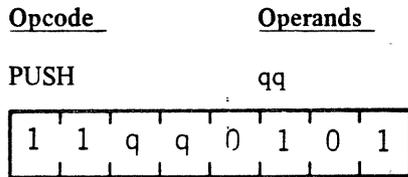
LD SP, IY

the Stack Pointer will also contain A227H.

PUSH qq

Operation: $(SP-2) \leftarrow qq_L, (SP-1) \leftarrow qq_H$

Format:



Description:

The contents of the register pair qq are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of register pair qq into the memory address now specified by the SP; then decrements the SP again and loads the low order byte of qq into the memory location corresponding to this new address in the SP. The operand qq means register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	qq
BC	00
DE	01
HL	10
AF	11

M CYCLES: 3 T STATES: 11(5,3,3) 4 MHZ E.T.: 2.75

Condition Bits Affected: None

Example:

If the AF register pair contains 2233H and the Stack Pointer contains 1007H, after the instruction

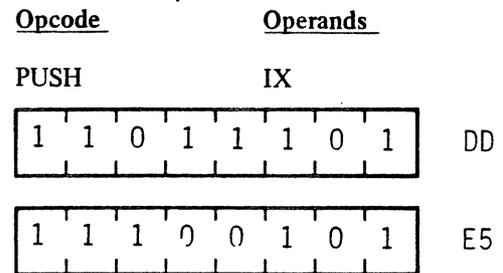
PUSH AF

memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H.

PUSH IX

Operation: $(SP-2) \leftarrow IX_L, (SP-1) \leftarrow IX_H$

Format:



Description:

The contents of the Index Register IX are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of IX into the memory address now specified by the SP; then decrements the SP again and loads the low order byte into the memory location corresponding to this new address in the SP.

M CYCLES: 3 T STATES: 15(4,5,3,3) 4 MHZ E.T.: 3.75

Condition Bits Affected: None

Example:

If the Index Register IX contains 2233H and the Stack Pointer contains 1007H, after the instruction

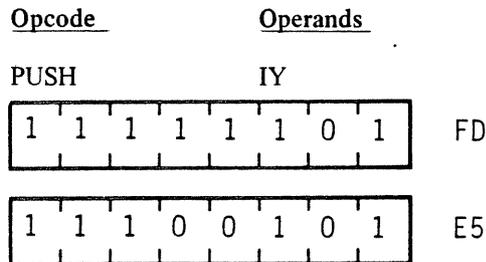
PUSH IX

memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H.

PUSH IY

Operation: $(SP-2) \leftarrow IY_L, (SP-1) \leftarrow IY_H$

Format:



Description:

The contents of the Index Register IY are pushed into the external memory LIFO (last-in, first-out) Stack. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first decrements the SP and loads the high order byte of IY into the memory address now specified by the SP; then decrements the SP again and loads the low order byte into the memory location corresponding to this new address in the SP.

M CYCLES: 4 T STATES: 15(4,5,3,3) 4 MHZ E.T.: 3.75

Condition Bits Affected: None

Example:

If the Index Register IY contains 2233H and the Stack Pointer contains 1007H, after the instruction

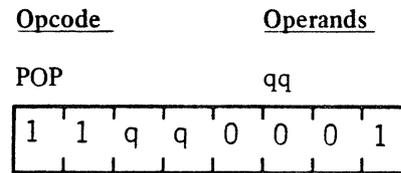
PUSH IY

memory address 1006H will contain 22H, memory address 1005H will contain 33H, and the Stack Pointer will contain 1005H.

POP qq

Operation: $qq_H \leftarrow (SP+1), qq_L \leftarrow (SP)$

Format:



Description:

The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into register pair qq. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of qq, the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of qq and the SP is now incremented again. The operand qq defines register pair BC, DE, HL, or AF, assembled as follows in the object code:

Pair	r
BC	00
DE	01
HL	10
AF	11

M CYCLES: 3 T STATES: 10(4,3,3) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

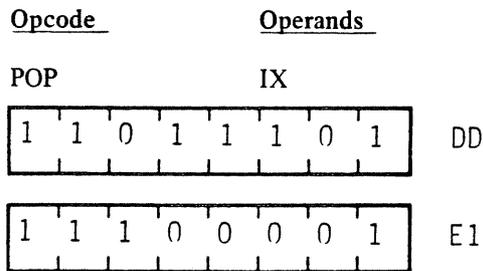
POP HL

will result in register pair HL containing 3355H, and the Stack Pointer containing 1002H.

POP IX

Operation: $IX_H \leftarrow (SP+1), IX_L \leftarrow (SP)$

Format:



Description:

The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into Index Register IX. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of IX the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of IX. The SP is now incremented again.

M CYCLES: 4 T STATES: 14(4,4,3,3) 4 MHZ E.T.: 3.50

Condition Bits Affected: None

Example:

If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

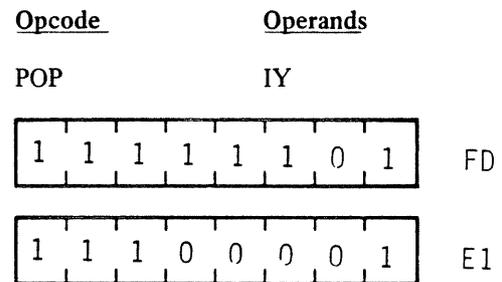
POP IX

will result in the Index Register IX containing 3355H, and the Stack Pointer containing 1002H.

POP IY

Operation: $IY_H \leftarrow (SP+1), IY_L \leftarrow (SP)$

Format:



Description:

The top two bytes of the external memory LIFO (last-in, first-out) Stack are popped into Index Register IY. The Stack Pointer (SP) register pair holds the 16-bit address of the current "top" of the Stack. This instruction first loads into the low order portion of IY the byte at the memory location corresponding to the contents of SP; then SP is incremented and the contents of the corresponding adjacent memory location are loaded into the high order portion of IY. The SP is now incremented again.

M CYCLES: 4 T STATES: 14(4,4,3,3) 4 MHZ E.T.: 3.50

Condition Bits Affected: None

Example:

If the Stack Pointer contains 1000H, memory location 1000H contains 55H, and location 1001H contains 33H, the instruction

POP IY

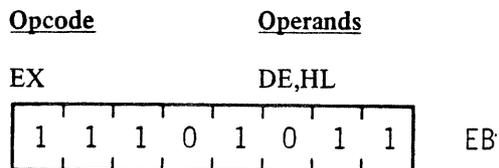
will result in Index Register IY containing 3355H, and the Stack Pointer containing 1002H.

EXCHANGE, BLOCK TRANSFER AND SEARCH GROUP

EX DE, HL

Operation: DE ↔ HL

Format:



Description:

The two-byte contents of register pairs DE and HL are exchanged.

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected: None

Example:

If the content of register pair DE is the number 2822H, and the content of the register pair HL is number 499AH, after the instruction

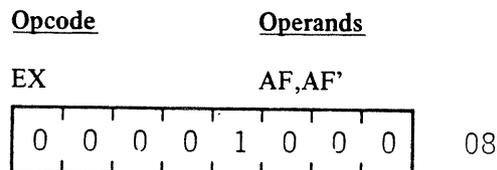
EX DE,HL

the content of register pair DE will be 499AH and the content of register pair HL will be 2822H.

EX AF, AF'

Operation: AF ↔ AF'

Format:



Description:

The two-byte contents of the register pairs AF and AF' are exchanged. (Note: register pair AF' consists of registers A' and F'.)

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected: None

Example:

If the content of register pair AF is number 9900H, and the content of register pair AF' is number 5944H, after the instruction

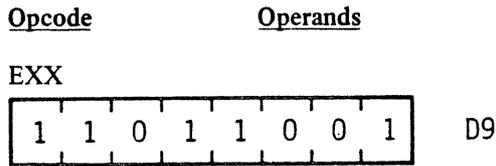
EX AF,AF'

the contents of AF will be 5944H, and the contents of AF' will be 9900H.

EXX

Operation: (BC) ↔ (BC'), (DE) ↔ (DE'), (HL) ↔ (HL')

Format:



Description:

Each two-byte value in register pairs BC, DE, and HL is exchanged with the two-byte value in BC', DE', and HL', respectively.

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected: None

Example:

If the contents of register pairs BC, DE, and HL are the numbers 445AH, 3DA2H, and 8859H, respectively, and the contents of register pairs BC', DE', and HL' are 0988H, 9300H, and 00E7H, respectively, after the instruction

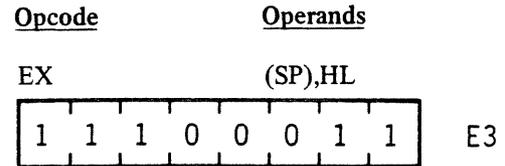
EXX

the contents of the register pairs will be as follows:
BC: 0988H; DE: 9300H; HL: 00E7H; BC': 445AH;
DE': 3DA2H; and HL': 8859H.

EX (SP), HL

Operation: H ↔ (SP+1), L ↔ (SP)

Format:



Description:

The low order byte contained in register pair HL is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of HL is exchanged with the next highest memory address (SP+1).

M CYCLES: 5 T STATES: 19(4,3,4,3,5) 4 MHZ E.T.: 4.75

Condition Bits Affected: None

Example:

If the HL register pair contains 7012H, the SP register pair contains 8856H, the memory location 8856H contains the byte 11H, and the memory location 8857H contains the byte 22H, then the instruction

EX (SP), HL

will result in the HL register pair containing number 2211H, memory location 8856H containing the byte 12H, the memory location 8857H containing the byte 70H and the Stack Pointer containing 8856H.

EX (SP), IX

Operation: $IX_H \leftrightarrow (SP+1), IX_L \leftrightarrow (SP)$

Format:

Opcode	Operands	
EX	(SP),IX	
1 1 0 1 1 1 0 1		DD
1 1 1 0 0 0 1 1		E3

Description:

The low order byte in Index Register IX is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of IX is exchanged with the next highest memory address (SP+1).

M CYCLES: 6 T STATES: 23(4,4,3,4,3,5) 4 MHZ E.T.: 5.75

Condition Bits Affected: None

Example:

If the Index Register IX contains 3988H, the SP register pair contains 0100H, the memory location 0100H contains the byte 90H, and memory location 0101H contains byte 48H, then the instruction

EX (SP), IX

will result in the IX register pair containing number 4890H, memory location 0100H containing 88H, memory location 0101H containing 39H and the Stack Pointer containing 0100H.

EX (SP), IY

Operation: $IY_H \leftrightarrow (SP+1), IY_L \leftrightarrow (SP)$

Format:

Opcode	Operands	
EX	(SP),IY	
1 1 1 1 1 1 0 1		FD
1 1 1 0 0 0 1 1		E3

Description:

The low order byte in Index Register IY is exchanged with the contents of the memory address specified by the contents of register pair SP (Stack Pointer), and the high order byte of IY is exchanged with the next highest memory address (SP+1).

M CYCLES: 6 T STATES: 23(4,4,3,4,3,5) 4 MHZ E.T.: 5.75

Condition Bits Affected: None

Example:

If the Index Register IY contains 3988H, the SP register pair contains 0100H, the memory location 0100H contains the byte 90H, and memory location 0101H contains byte 48H, then the instruction

EX (SP), IY

will result in the IY register pair containing number 4890H, memory location 0100H containing 88H, memory location 0101H containing 39H, and the Stack Pointer containing 0100H.

LDI

Operation:

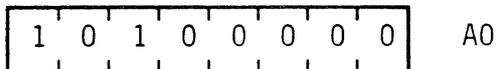
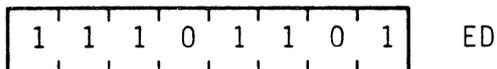
$(DE) \leftarrow (HL), DE \leftarrow DE+1, HL \leftarrow HL+1, BC \leftarrow BC-1$

Format:

Opcode

Operands

LDI



Description:

A byte of data is transferred from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both these register pairs are incremented and the BC (Byte Counter) register pair is decremented.

M CYCLES: 4 T STATES: 16(4,4,3,5) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Not affected
- Z: Not affected
- H: Reset
- P/V: Set if $BC-1 \neq 0$; reset otherwise
- N: Reset
- C: Not affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains the byte 88H, the DE register pair contains 2222H, the memory location 2222H contains byte 66H, and the BC register pair contains 7H, then the instruction

L D I

will result in the following contents in register pairs and memory addresses:

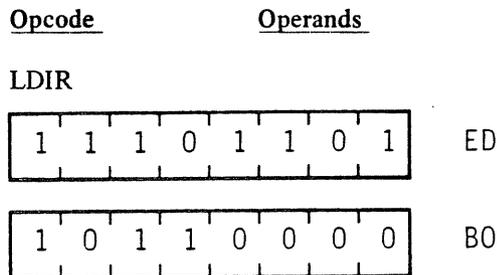
HL	:	1112H
(1111H)	:	88H
DE	:	2223H
(2222H)	:	88H
BC	:	6H

LDIR

Operation:

$(DE) \leftarrow (HL), DE \leftarrow DE+1, HL \leftarrow HL+1, BC \leftarrow BC-1$

Format:



Description:

This two byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the DE register pair. Then both these register pairs are incremented and the BC (Byte Counter) register pair is decremented. If decrementing causes the BC to go to zero, the instruction is terminated. If BC is not zero the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64K bytes. Also, interrupts will be recognized after each data transfer.

For $BC \neq 0$:

M CYCLES: 5 T STATES: 21(4,4,3,5,5) 4 MHZ E.T.: 5.25

For $BC=0$:

M CYCLES: 4 T STATES: 16(4,4,3,5) 4 MHZ E.T.: 4.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Reset
N: Reset
C: Not affected

Example:

If the HL register pair contains 1111H, the DE register pair contains 2222H, the BC register pair contains 0003H, and memory locations have these contents:

(1111H) : 88H	(2222H) : 66H
(1112H) : 36H	(2223H) : 59H
(1113H) : A5H	(2224H) : C5H

then after the execution of

LDIR

the contents of register pairs and memory locations will be:

HL : 1114H
DE : 2225H
BC : 0000H

(1111H) : 88H	(2222H) : 88H
(1112H) : 36H	(2223H) : 36H
(1113H) : A5H	(2224H) : A5H



LDD

Operation:

(DE) ← (HL), DE ← DE-1, HL ← HL-1, BC ← BC-1

Format:

Opcode

Operands

LDD

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

 ED

1	0	1	0	1	0	0	0
---	---	---	---	---	---	---	---

 A8

Description:

This two byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these register pairs including the BC (Byte Counter) register pair are decremented.

M CYCLES: 4 T STATES: 16(4,4,3,5) 4 MHZ E.T.: 4.00

Condition Bits Affected:

S: Not affected
Z: Not affected
H: Reset
P/V: Set if BC-1≠0; reset otherwise
N: Reset
C: Not affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains the byte 88H, the DE register pair contains 2222H, memory location 2222H contains byte 66H, and the BC register pair contains 7H, then the instruction

LDD

will result in the following contents in register pairs and memory addresses:

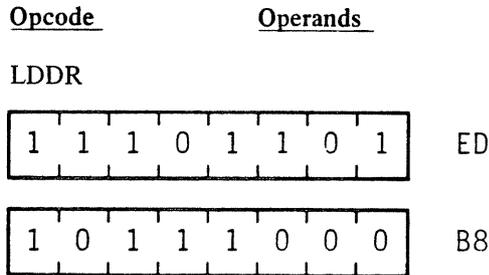
HL	:	1110H
(1111H)	:	88H
DE	:	2221H
(2222H)	:	88H
BC	:	6H

LDDR

Operation:

$(DE) \leftarrow (HL), DE \leftarrow DE-1, HL \leftarrow HL-1, BC \leftarrow BC-1$

Format:



Description:

This two byte instruction transfers a byte of data from the memory location addressed by the contents of the HL register pair to the memory location addressed by the contents of the DE register pair. Then both of these registers as well as the BC (Byte Counter) are decremented. If decrementing causes the BC to go to zero, the instruction is terminated. If BC is not zero, the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64K bytes. Also, interrupts will be recognized after each data transfer.

For $BC \neq 0$:

M CYCLES: 5 T STATES: 21(4,4,3,5,5) 4 MHZ E.T.: 5.25

For $BC = 0$:

M CYCLES: 4 T STATES: 16(4,4,3,5) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Not affected
- Z: Not affected
- H: Reset
- P/V: Reset
- N: Reset
- C: Not affected

Example:

If the HL register pair contains 1114H, the DE register pair contains 2225H, the BC register pair contains 0003H, and memory locations have these contents:

(1114H) : A5H	(2225H) : C5H
(1113H) : 36H	(2224H) : 59H
(1112H) : 88H	(2223H) : 66H

then after the execution of

LDDR

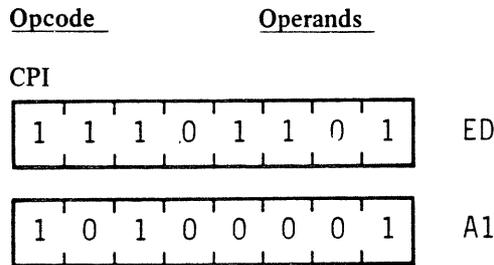
the contents of register pairs and memory locations will be:

HL	:	1111H	
DE	:	2222H	
BC	:	0000H	
(1114H)	:	A5H	(2225H) : A5H
(1113H)	:	36H	(2224H) : 36H
(1112H)	:	88H	(2223H) : 88H

CPI

Operation: A ← (HL), HL ← HL+1, BC ← BC-1

Format:



Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. Then HL is incremented and the Byte Counter (register pair BC) is decremented.

M CYCLES: 4 T STATES: 16(4,4,3,5) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if A=(HL); reset otherwise
- H: Set if no borrow from Bit 4; reset otherwise
- P/V: Set if BC-1≠0; reset otherwise
- N: Set
- C: Not affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains 3BH, the Accumulator contains 3BH, and the Byte Counter contains 0001H, then after the execution of

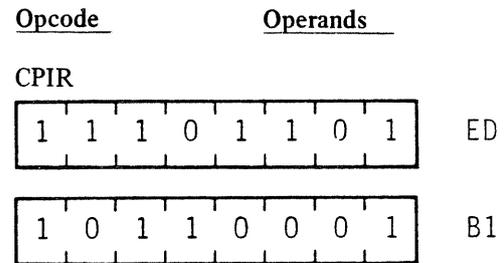
CPI

the Byte Counter will contain 0000H, the HL register pair will contain 1112H, the Z flag in the F register will be set, and the P/V flag in the F register will be reset. There will be no effect on the contents of the Accumulator or address 1111H.

CPIR

Operation: A ← (HL), HL ← HL+1, BC ← BC-1

Format:



Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL is incremented and the Byte Counter (register pair BC) is decremented. If decrementing causes the BC to go to zero or if A=(HL), the instruction is terminated. If BC is not zero and A≠(HL), the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero before the execution, the instruction will loop through 64K bytes, if no match is found. Also, interrupts will be recognized after each data comparison.

For BC≠0 and A≠(HL):

M CYCLES: 5 T STATES: 21(4,4,3,5,5) 4 MHZ E.T.: 5.25

For BC=0 or A=(HL):

M CYCLES: 4 T STATES: 16(4,4,3,5) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if A=(HL); reset otherwise
- H: Set if no borrow from Bit 4; reset otherwise
- P/V: Set if BC-1≠0; reset otherwise
- N: Set
- C: Not affected

Example:

If the HL register pair contains 1111H, the Accumulator contains F3H, the Byte Counter contains 0007H, and memory locations have these contents:

(1111H) : 52H
 (1112H) : 00H
 (1113H) : F3H

then after the execution of

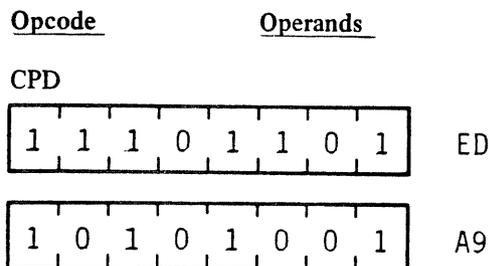
CPIR

the contents of register pair HL will be 1114H, the contents of the Byte Counter will be 0004H, the P/V flag in the F register will be set and the Z flag in the F register will be set.

CPD

Operation: A – (HL), HL ← HL–1, BC ← BC–1

Format:



Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL and the Byte Counter (register pair BC) are decremented.

M CYCLES: 4 T STATES: 16(4,4,3,5) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if A=(HL); reset otherwise
- H: Set if no borrow from Bit 4; reset otherwise
- P/V: Set if BC–1≠0; reset otherwise
- N: Set
- C: Not affected

Example:

If the HL register pair contains 1111H, memory location 1111H contains 3BH, the Accumulator contains 3BH, and the Byte Counter contains 0001H, then after the execution of

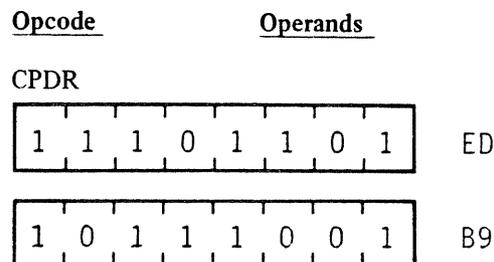
CPD

the Byte Counter will contain 0000H, the HL register pair will contain 1110H, the Z flag in the F register will be set, and the P/V flag in the F register will be reset. There will be no effect on the contents of the Accumulator or address 1111H.

CPDR

Operation: A – (HL), HL ← HL–1, BC ← BC–1

Format:



Description:

The contents of the memory location addressed by the HL register pair is compared with the contents of the Accumulator. In case of a true compare, a condition bit is set. The HL and BC (Byte Counter) register pairs are decremented. If decrementing causes the BC to go to zero or if A=(HL), the instruction is terminated. If BC is not zero and A≠(HL), the program counter is decremented by 2 and the instruction is repeated. Note that if BC is set to zero prior to instruction execution, the instruction will loop through 64K bytes, if no match is found. Also, interrupts will be recognized after each data comparison.

For BC≠0 and A≠(HL):

M CYCLES: 5 T STATES: 21(4,4,3,5,5) 4 MHZ E.T.: 5.25

For BC=0 or A=(HL):

M CYCLES: 4 T STATES: 16(4,4,3,5) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if A=(HL); reset otherwise
- H: Set if no borrow from Bit 4; reset otherwise
- P/V: Set if BC–1≠0; reset otherwise
- N: Set
- C: Not affected

Example:

If the HL register pair contains 1118H, the Accumulator contains F3H, the Byte Counter contains 0007H, and memory locations have these contents:

(1118H) : 52H
 (1117H) : 00H
 (1116H) : F3H

then after the execution of

CPDR

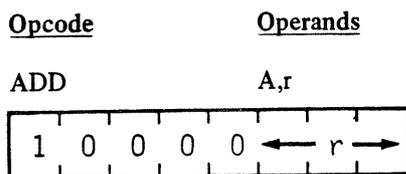
the contents of register pair HL will be 1115H, the contents of the Byte Counter will be 0004H, the P/V flag in the F register will be set, and the Z flag in the F register will be set.

8 BIT ARITHMETIC AND LOGICAL GROUP

ADD A, r

Operation: $A \leftarrow A + r$

Format:



Description:

The contents of register r are added to the contents of the Accumulator, and the result is stored in the Accumulator. The symbol r identifies the registers A,B,C,D,E,H or L assembled as follows in the object code:

Register	r
A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Reset
- C: Set if carry from Bit 7; reset otherwise

Example:

If the contents of the Accumulator are 44H, and the contents of register C are 11H, after the execution of

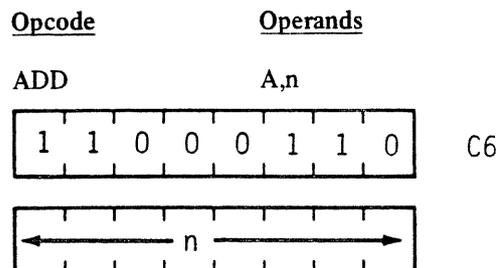
ADD A, C

the contents of the Accumulator will be 55H.

ADD A, n

Operation: $A \leftarrow A + n$

Format:



Description:

The integer n is added to the contents of the Accumulator and the results are stored in the Accumulator.

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Reset
- C: Set if carry from Bit 7; reset otherwise

Example:

If the contents of the Accumulator are 23H, after the execution of

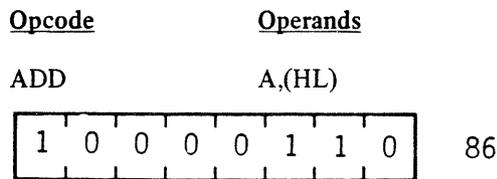
ADD A, 33H

the contents of the Accumulator will be 56H.

ADD A, (HL)

Operation: $A \leftarrow A + (HL)$

Format:



Description:

The byte at the memory address specified by the contents of the HL register pair is added to the contents of the Accumulator and the result is stored in the Accumulator.

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Reset
- C: Set if carry from Bit 7; reset otherwise

Example:

If the contents of the Accumulator are A0H, and the content of the register pair HL is 2323H, and memory location 2323H contains byte 08H, after the execution of

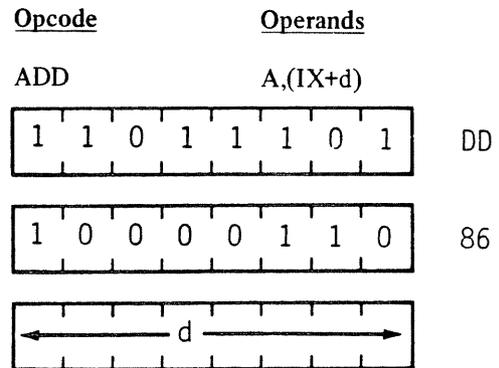
ADD A, (HL)

the Accumulator will contain A8H.

ADD A, (IX+d)

Operation: $A \leftarrow A + (IX+d)$

Format:



Description:

The contents of the Index Register (register pair IX) is added to a displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator and the result is stored in the Accumulator.

M CYCLES: 5 T STATES: 19(4,4,3,5,3) 4 MHZ E.T.: 4.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Reset
- C: Set if carry from Bit 7; reset otherwise

Example:

If the Accumulator contents are 11H, the Index Register IX contains 1000H, and if the content of memory location 1005H is 22H, after the execution of

ADD A, (IX+5H)

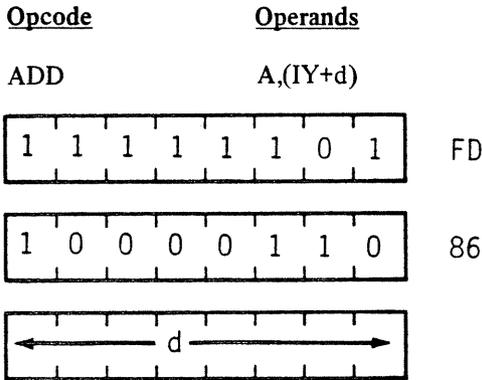
the contents of the Accumulator will be 33H.

ADD A, (IY+d)



Operation: $A \leftarrow A + (IY + d)$

Format:



Description:

The contents of the Index Register (register pair IY) is added to the displacement d to point to an address in memory. The contents of this address is then added to the contents of the Accumulator and the result is stored in the Accumulator.

M CYCLES: 5 T STATES: 19(4,4,3,5,3) 4 MHZ E.T.: 4.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Reset
- C: Set if carry from Bit 7; reset otherwise

Example:

If the Accumulator contents are 11H, the Index Register pair IY contains 1000H, and if the content of memory location 1005H is 22H, after the execution of

ADD A, (IY+5H)

the contents of the Accumulator will be 33H.

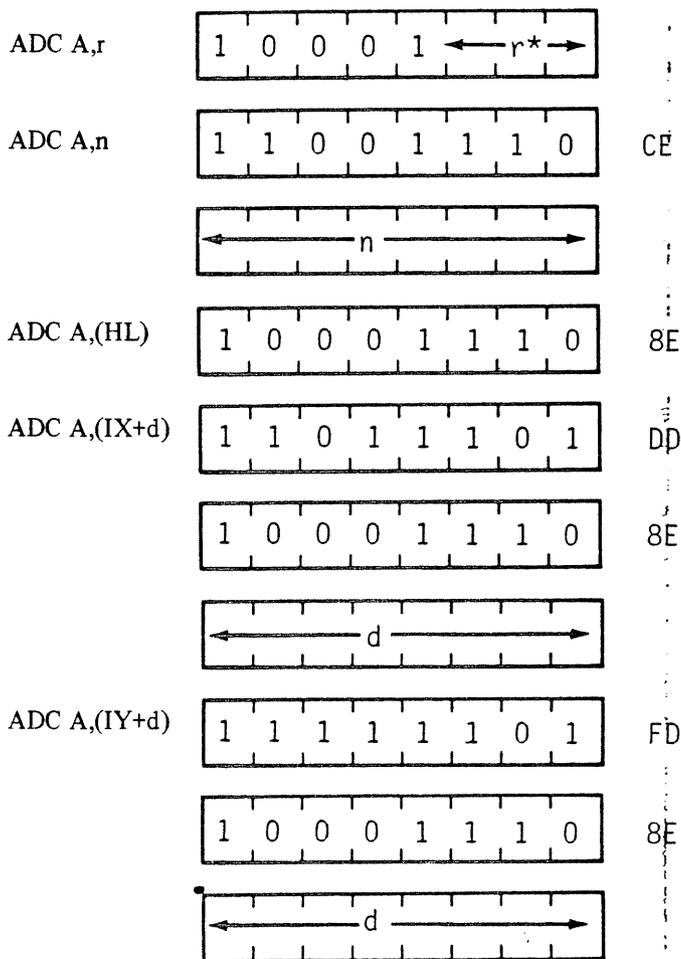
ADC A, s

Operation: $A \leftarrow A + s + CY$

Format:

Opcode	Operands
ADC	A,s

The s operand is any of r,n,(HL),(IX+d) or (IY+d) as defined for the analogous ADD instruction. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The s operand, along with the Carry Flag ("C" in the F register) is added to the contents of the Accumulator, and the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
ADC A,r	1	4	1.00
ADC A,n	2	7(4,3)	1.75
ADC A,(HL)	2	7(4,3)	1.75
ADC A,(IX+d)	5	19(4,4,3,5,3)	4.75
ADC A,(IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Reset
- C: Set if carry from Bit 7; reset otherwise

Example:

If the Accumulator contains 16H, the Carry Flag is set, the HL register pair contains 6666H, and address 6666H contains 10H, after the execution of

ADC A, (HL)

the Accumulator will contain 27H.

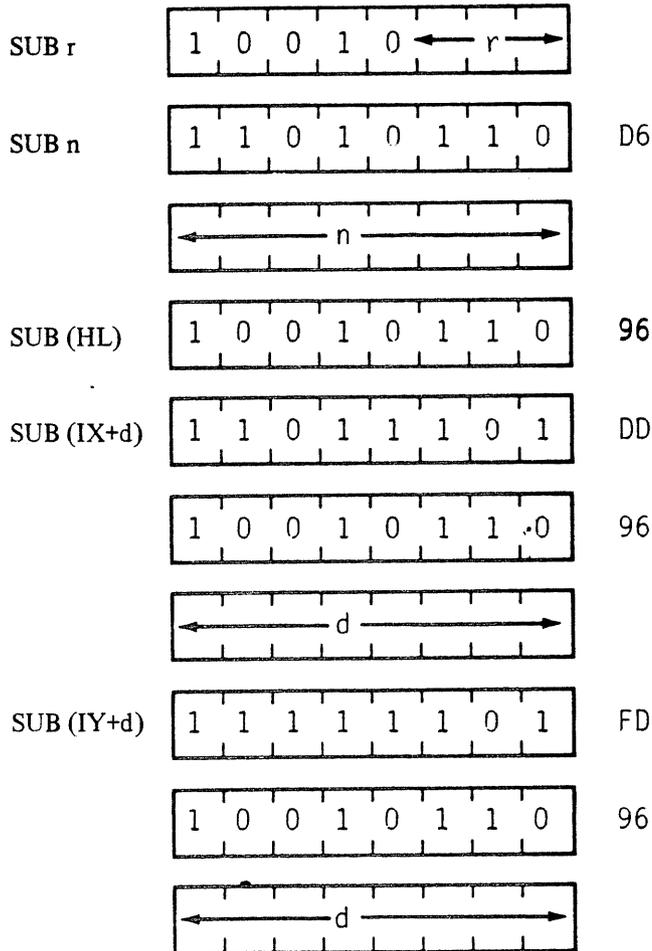
SUB s

Operation: $A \leftarrow A - s$

Format:

Opcode	Operands
SUB	s

The s operand is any of r,n,(HL),(IX+d) or (IY+d) as defined for the analogous ADD instruction. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The s operand is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
SUB r	1	4	1.00
SUB n	2	7(4,3)	1.75
SUB (HL)	2	7(4,3)	1.75
SUB (IX+d)	5	19(4,4,3,5,3)	4.75
SUB (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if no borrow from Bit 4; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Set
- C: Set if borrow; reset otherwise

Example:

If the Accumulator contains 29H and register D contains 11H, after the execution of

SUB D

the Accumulator will contain 18H.

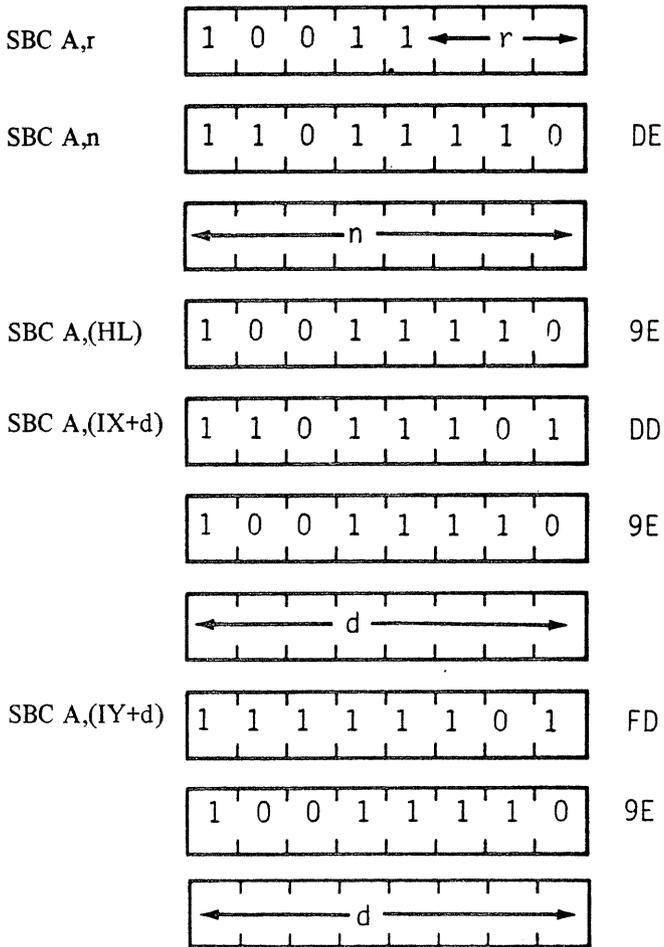
SBC A, s

Operation: $A \leftarrow A - s - CY$

Format:

Opcode	Operands
SBC	A,s

The s operand is any of r,n,(HL),(IX+d) or (IY+d) as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description

The s operand, along with the Carry Flag ("C" in the F register) is subtracted from the contents of the Accumulator, and the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
SBC A,r	1	4	1.00
SBC A,n	2	7(4,3)	1.75
SBC A,(HL)	2	7(4,3)	1.75
SBC A,(IX+d)	5	19(4,4,3,5,3)	4.75
SBC A,(IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if no borrow from Bit 4; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Set
- C: Set if borrow; reset otherwise

Example:

If the Accumulator contains 16H, the Carry Flag is set, the HL register pair contains 3433H, and address 3433H contains 05H, after the execution of

```
SBC A, (HL)
```

the Accumulator will contain 10H.

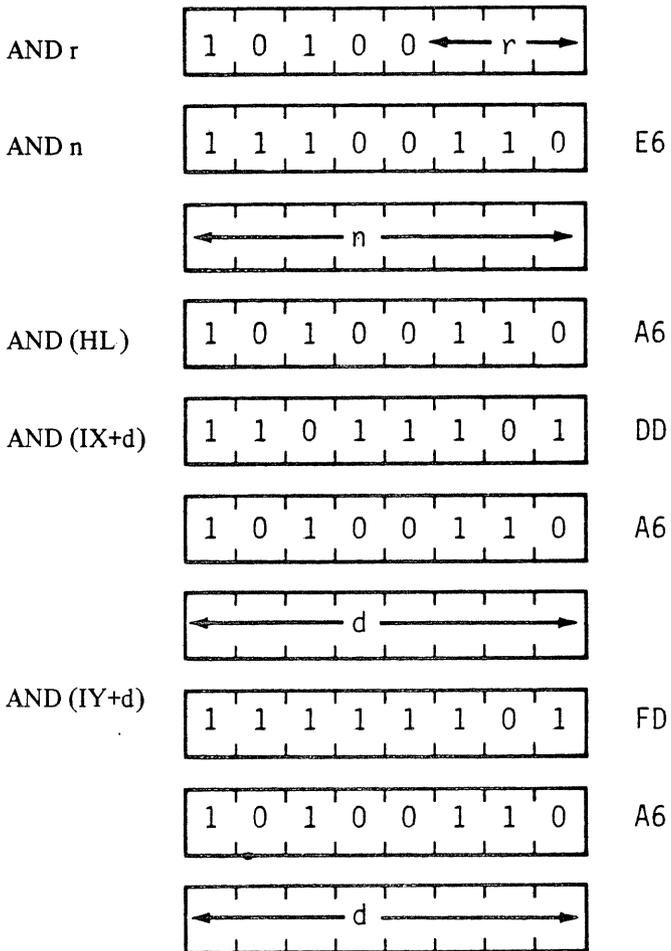
AND s

Operation: $A \leftarrow A \wedge s$

Format:

Opcode	Operands
AND	s

The s operand is any of r,n,(HL),(IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies register B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

A logical AND operation, Bit by Bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
AND r	1	4	1.00
AND n	2	7(4,3)	1.75
AND (HL)	2	7(4,3)	1.75
AND (IX+d)	5	19(4,4,3,5,3)	4.75
AND (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set
- P/V: Set if parity even; reset otherwise
- N: Reset
- C: Reset

Example:

If the B register contains 7BH (01111011) and the Accumulator contains C3H (11000011) after the execution of

AND B

the Accumulator will contain 43H (01000011).

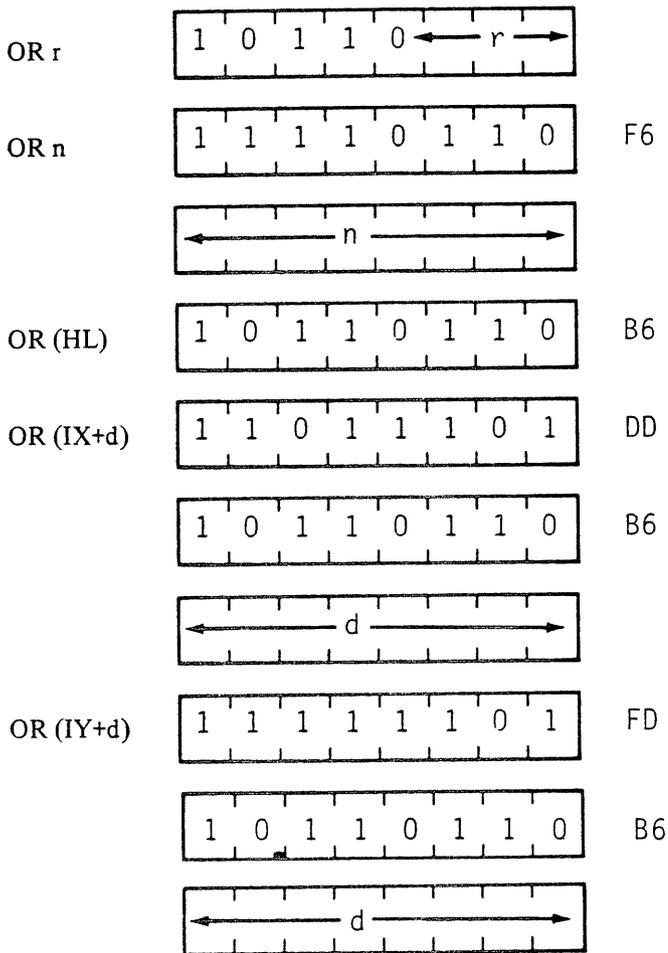
OR s

Operation: $A \leftarrow A \vee s$

Format:

Opcode	Operands
OR	s

The s operand is any of r,n,(HL),(IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies register B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

A logical OR operation, Bit by Bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
OR r	1	4	1.00
OR n	2	7(4,3)	1.75
OR (HL)	2	7(4,3)	1.75
OR (IX+d)	5	19(4,4,3,5,3)	4.75
OR (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set
- P/V: Set if parity even; reset otherwise
- N: Reset
- C: Reset

Example:

If the H register contains 48H (010001000) and the Accumulator contains 12H (00010010) after the execution of

OR H

the Accumulator will contain 5AH (01011010).

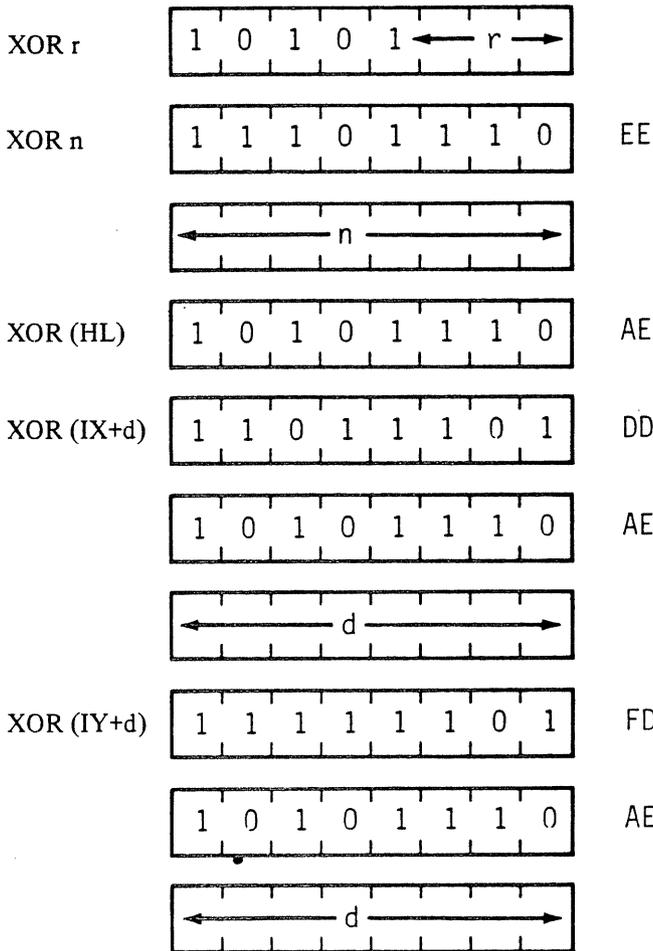
XOR s

Operation: $A \leftarrow A \oplus s$

Format:

Opcode	Operands
XOR	s

The s operand is any of r,n, (HL),(IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register r

B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

A logical exclusive-OR operation, bit by bit, is performed between the byte specified by the s operand and the byte contained in the Accumulator; the result is stored in the Accumulator.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
XOR r	1	4	1.00
XOR n	2	7(4,3)	1.75
XOR (HL)	2	7(4,3)	1.75
XOR (IX+d)	5	19(4,4,3,5,3)	4.75
XOR (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Reset

Example:

If the Accumulator contains 96H (10010110), after the execution of

XOR 5DH (Note: 5DH = 01011101)

the Accumulator will contain CBH (11001011).

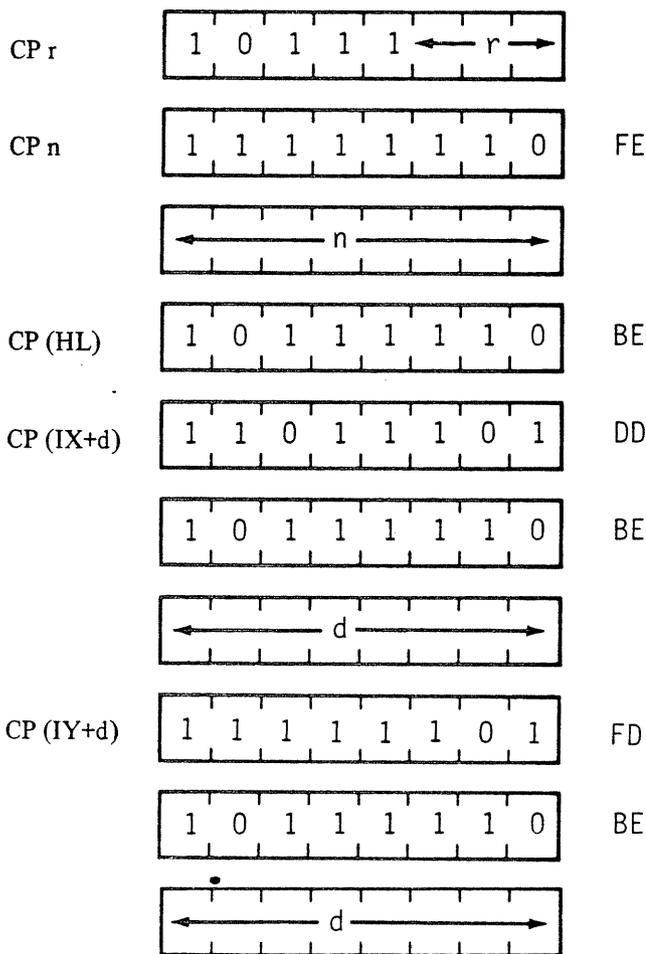
CP s

Operation: A - s

Format:

Opcode	Operands
CP	s

The s operand is any of r,n,(HL),(IX+d) or (IY+d), as defined for the analogous ADD instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies registers B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of the s operand are compared with the contents of the Accumulator. If there is a true compare, a flag is set.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
CP r	1	4	1.00
CP n	2	7(4,3)	1.75
CP (HL)	2	7(4,3)	1.75
CP (IX+d)	5	19(4,4,3,5,3)	4.75
CP (IY+d)	5	19(4,4,3,5,3)	4.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if no borrow from Bit 4; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Set
- C: Set if borrow; reset otherwise

Example:

If the Accumulator contains 63H, the HL register pair contains 6000H and memory location 6000H contains 60H, the instruction

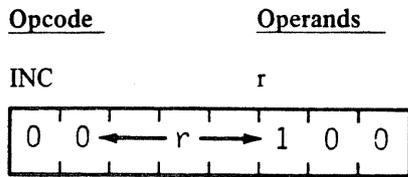
CP (HL)

will result in the P/V flag in the F register being reset.

INC r

Operation: $r \leftarrow r + 1$

Format:



Description:

Register r is incremented. r identifies any of the registers A,B, C,D,E,H or L, assembled as follows in the object code.

Register r

A	111
B	000
C	001
D	010
E	011
H	100
L	101

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if r was 7FH before operation; reset otherwise
- N: Reset
- C: Not affected

Example:

If the contents of register D are 28H, after the execution of

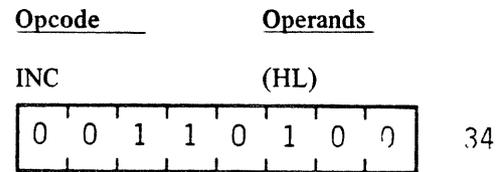
INC D

the contents of register D will be 29H.

INC (HL)

Operation: $(HL) \leftarrow (HL)+1$

Format:



Description:

The byte contained in the address specified by the contents of the HL register pair is incremented.

M CYCLES: 3 T STATES: 11(4,4,3) 4 MHZ E.T.: 2.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if (HL) was 7FH before operation; reset otherwise
- N: Reset
- C: Not Affected

Example:

If the contents of the HL register pair are 3434H, and the contents of address 3434H are 82H, after the execution of

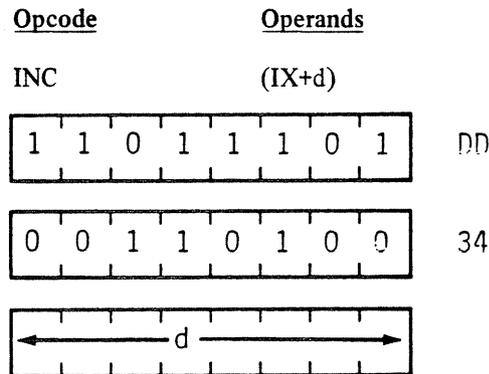
INC (HL)

memory location 3434H will contain 83H.

INC (IX+d)

Operation: $(IX+d) \leftarrow (IX+d)+1$

Format:



Description:

The contents of the Index Register IX (register pair IX) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3) 4 MHZ E.T.: 5.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if (IX+d) was 7FH before operation; reset otherwise
- N: Reset
- C: Not affected

Example:

If the contents of the Index Register pair IX are 2020H, and the memory location 2030H contains byte 34H, after the execution of

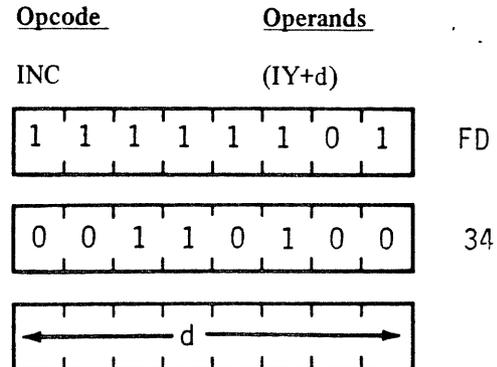
INC (IX+10H)

the contents of memory location 2030H will be 35H.

INC (IY+d)

Operation: $(IY+d) \leftarrow (IY+d)+1$

Format:



Description:

The contents of the Index Register IY (register pair IY) are added to a two's complement displacement integer d to point to an address in memory. The contents of this address are then incremented.

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3) 4 MHZ E.T.: 5.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if carry from Bit 3; reset otherwise
- P/V: Set if (IY+d) was 7FH before operation; reset otherwise
- N: Reset
- C: Not Affected

Example:

If the contents of the Index Register pair IY are 2020H, and the memory location 2030H contain byte 34H, after the execution of

INC (IY+10H)

the contents of memory location 2030H will be 35H.

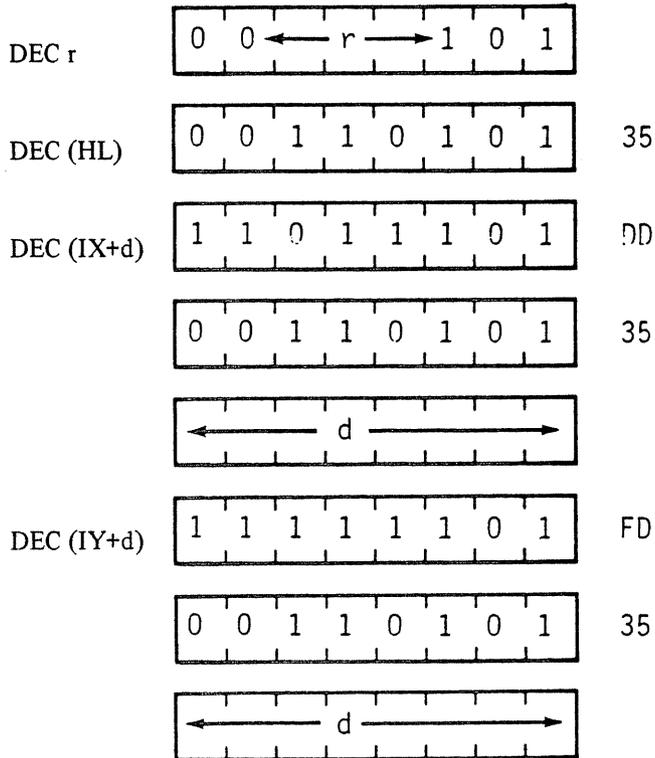
DEC m

Operation: $m \leftarrow m-1$

Format:

Opcode	Operands
DEC	m

The m operand is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous INC instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



*r identifies register B,C,D,E,H,L or A assembled as follows in the object code field above:

Register	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The byte specified by the m operand is decremented.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
DEC r	1	4	1.00
DEC (HL)	3	11(4,4,3)	2.75
DEC (IX+d)	6	23 (4,4,3,5,4,3)	5.75
DEC (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if no borrow from Bit 4; reset otherwise
P/V:	Set if m was 80H before operation; reset otherwise
N:	Set
C:	Not affected

Example:

If the D register contains byte 2AH, after the execution of

DEC D

register D will contain 29H.

GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS

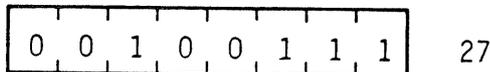
DAA

Operation: —

Format:

Opcode

DAA



Description:

This instruction conditionally adjusts the Accumulator for BCD addition and subtraction operations. For addition (ADD, ADC, INC) or subtraction (SUB, SBC, DEC, NEG), the following table indicates operation performed:

OPERATION	C BE- FORE DAA	HEX	HEX	NUM-	C	
		VALUE IN UPPER DIGIT (bit 7-4)	BE- FORE DAA	VALUE IN LOWER DIGIT (bit 3-0)		BER C ADD- ED TO DAA BYTE
ADD ADC INC	0	0-9	0	0-9	00	0
	0	0-8	0	A-F	06	0
	0	0-9	1	0-3	06	0
	0	A-F	0	0-9	60	1
	0	9-F	0	A-F	66	1
	0	A-F	1	0-3	66	1
	1	0-2	0	0-9	60	1
SUB SBC DEC NEG	1	0-2	0	A-F	66	1
	1	0-3	1	0-3	66	1
	0	0-9	0	0-9	00	0
	0	0-8	1	6-F	FA	0
	1	7-F	0	0-9	A0	1
	1	6-F	1	6-F	9A	1

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

- S: Set if most significant bit of Acc, is 1 after operation; reset otherwise
- Z: Set if Acc. is zero after operation; reset otherwise
- H: See instruction
- P/V: Set if Acc. is even parity after operation; reset otherwise
- N: Not affected
- C: See instruction

Example:

If an addition operation is performed between 15 (BCD) and 27 (BCD), simple decimal arithmetic gives this result:

$$\begin{array}{r} 15 \\ +27 \\ \hline 42 \end{array}$$

But when the binary representations are added in the Accumulator according to standard binary arithmetic,

$$\begin{array}{r} 0001 \quad 0101 \\ +0010 \quad 0111 \\ \hline 0011 \quad 1100 = 3C \end{array}$$

the sum is ambiguous. The DAA instruction adjusts this result so that the correct BCD representation is obtained:

$$\begin{array}{r} 0011 \quad 1100 \\ +0000 \quad 0110 \\ \hline 0100 \quad 0010 = 42 \end{array}$$

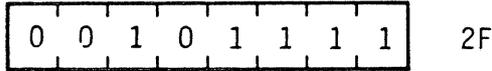
CPL

Operation: $A \leftarrow \bar{A}$

Format:

Opcode

CPL



Description:

Contents of the Accumulator (register A) are inverted (1's complement).

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

S: Not affected
 Z: Not affected
 H: Set
 P/V: Not affected
 N: Set
 C: Not affected

Example:

If the contents of the Accumulator are 1011 0100, after the execution of

CPL

the Accumulator contents will be 0100 1011.

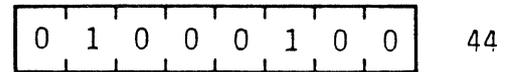
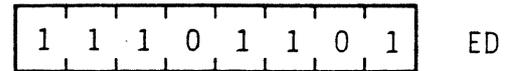
NEG

Operation: $A \leftarrow o-A$

Format:

Opcode

NEG



Description:

Contents of the Accumulator are negated (two's complement). This is the same as subtracting the contents of the Accumulator from zero. Note that 80H is left unchanged.

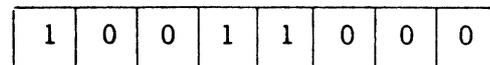
M CYCLES: 2 T STATES: 8(4,4) 4 MHZ E.T.: 2.00

Condition Bits Affected:

S: Set if result is negative; reset otherwise
 Z: Set if result is zero; reset otherwise
 H: Set if no borrow from Bit 4; reset otherwise
 P/V: Set if Acc. was 80H before operation; reset otherwise
 N: Set
 C: Set if Acc. was not 00H before operation; reset otherwise

Example:

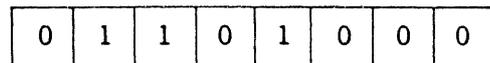
If the contents of the Accumulator are



after the execution of

NEG

the Accumulator contents will be



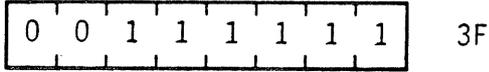
CCF

Operation: $CY \leftarrow \overline{CY}$

Format:

Opcode

CCF



Description:

The C flag in the F register is inverted.

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

- S: Not affected
- Z: Not affected
- H: Previous carry will be copied
- P/V: Not affected
- N: Reset
- C: Set if CY was 0 before operation; reset otherwise

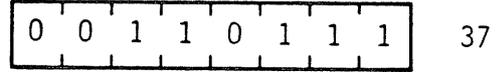
SCF

Operation: $CY \leftarrow 1$

Format:

Opcode

SCF



Description:

The C flag in the F register is set.

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

- S: Not affected
- Z: Not affected
- H: Reset
- P/V: Not affected
- N: Reset
- C: Set

NOP

Operation: _____

Format:

Opcode

NOP



Description:

CPU performs no operation during this machine cycle.

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected: None

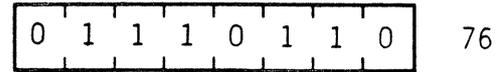
HALT

Operation: _____

Format:

Opcode

HALT



Description:

The HALT instruction suspends CPU operation until a subsequent interrupt or reset is received. While in the halt state, the processor will execute NOP's to maintain memory refresh logic.

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected: None

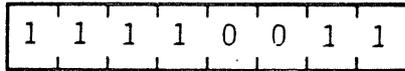
DI

Operation: IFF ← 0

Format:

Opcode

DI



F3

Description:

DI disables the maskable interrupt by resetting the interrupt enable flip-flops(IFF1 and IFF2). Note that this instruction disables the maskable interrupt during its execution.

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected: None

Example:

When the CPU executes the instruction

DI

the maskable interrupt is disabled until it is subsequently re-enabled by an EI instruction. The CPU will not respond to an Interrupt Request (INT) signal.

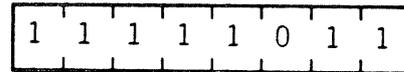
EI

Operation: IFF ← 1

Format:

Opcode

EI



FB

Description:

EI enables the maskable interrupt by setting the interrupt enable flip-flops(IFF1 and IFF2). Note that this instruction disables the maskable interrupt during its execution.

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected: None

Example:

When the CPU executes instruction

EI

the maskable interrupt is enabled. The CPU will now respond to an Interrupt Request (INT) signal.

IM 0

Operation: —

Format:

<u>Opcode</u>	<u>Operands</u>	
IM	\emptyset	
1 1 1 0 1 1 0 1		ED
0 1 0 0 0 1 1 0		46

Description:

The IM \emptyset instruction sets interrupt mode \emptyset . In this mode the interrupting device can insert any instruction on the data bus and allow the CPU to execute it.

M CYCLES: 2 T STATES: 8(4,4) 4 MHZ E.T.: 2.00

Condition Bits Affected: None

IM 1

Operation: —

Format:

<u>Opcode</u>	<u>Operands</u>	
IM	1	
1 1 1 0 1 1 0 1		ED
0 1 0 1 0 1 1 0		56

Description:

The IM instruction sets interrupt mode 1. In this mode the processor will respond to an interrupt by executing a restart to location $\emptyset\emptyset38H$.

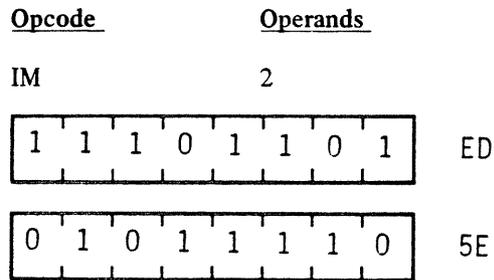
M CYCLES: 2 T STATES: 8(4,4) 4 MHZ E.T.: 2.00

Condition Bits Affected: None

IM 2

Operation: —

Format:



Description:

The IM 2 instruction sets interrupt mode 2. This mode allows an indirect call to any location in memory. With this mode the CPU forms a 16-bit memory address. The upper eight bits are the contents of the Interrupt Vector Register I and the lower eight bits are supplied by the interrupting device.

M CYCLES: 2 T STATES: 8(4,4) 4 MHZ E.T.: 2.00

Condition Bits Affected: None

16 BIT ARITHMETIC GROUP

ADD HL, SS

Operation: HL ← HL+ss

Format:

Opcode	Operands
ADD	HL,ss
0 0 s s 1 0 0 1	

Description:

The contents of register pair ss (any of register pairs BC,DE, HL or SP) are added to the contents of register pair HL and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M CYCLES: 3 T STATES: 11(4,4,3) 4 MHZ E.T.: 2.75

Condition Bits Affected:

S:	Not affected
Z:	Not affected
H:	Set if carry out of Bit 11; reset otherwise
P/V:	Not affected
N:	Reset
C:	Set if carry from Bit 15; reset otherwise

Example:

If register pair HL contains the integer 4242H and register pair DE contains 1111H, after the execution of

ADD HL, DE

the HL register pair will contain 5353H.

ADC HL, SS

Operation: HL ← HL+ss+CY

Format:

Opcode	Operands
ADC	HL,ss
1 1 1 0 1 1 0 1	ED
0 1 s s 1 0 1 0	

Description:

The contents of register pair ss (any of register pairs BC,DE, HL or SP) are added with the Carry Flag (C flag in the F register) to the contents of register pair HL, and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M CYCLES: 4 T STATES: 15(4,4,4,3) 4 MHZ E.T.: 3.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Set if carry out of Bit 11; reset otherwise
P/V:	Set if overflow; reset otherwise
N:	Reset
C:	Set if carry from Bit 15; reset otherwise

Example:

If the register pair BC contains 2222H, register pair HL contains 5437H and the Carry Flag is set, after the execution of

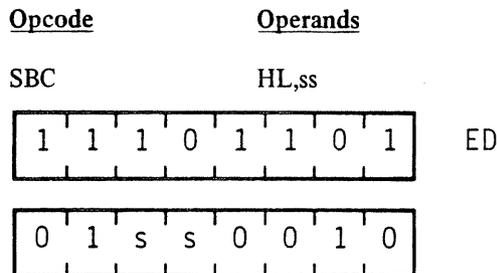
ADC HL, BC

the contents of HL will be 765AH.

SBC HL, ss

Operation: HL ← HL - ss - CY

Format:



Description:

The contents of the register pair ss (any of register pairs BC,DE,HL or SP) and the Carry Flag (C flag in the F register) are subtracted from the contents of register pair HL and the result is stored in HL. Operand ss is specified as follows in the assembled object code.

<u>Register Pair</u>	<u>ss</u>
BC	00
DE	00
HL	10
SP	11

M CYCLES: 4 T STATES: 15(4,4,4,3) 4 MHZ E.T.: 3.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Set if no borrow from Bit 12; reset otherwise
- P/V: Set if overflow; reset otherwise
- N: Set
- C: Set if borrow; reset otherwise

Example:

If the contents of the HL register pair are 9999H, the contents of register pair DE are 1111H, and the Carry Flag is set, after the execution of

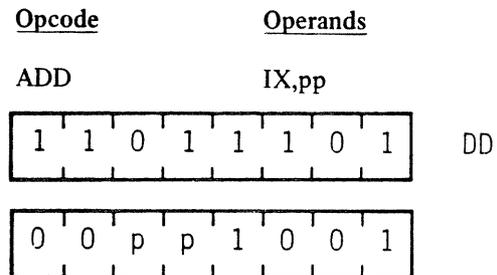
SBC HL, DE

the contents of HL will be 8887H.

ADD IX, pp

Operation: IX ← IX + pp

Format:



Description:

The contents of register pair pp (any of register pairs BC,DE, IX or SP) are added to the contents of the Index Register IX, and the results are stored in IX. Operand pp is specified as follows in the assembled object code.

<u>Register Pair</u>	<u>pp</u>
BC	00
DE	01
IX	10
SP	11

M CYCLES: 4 T STATES: 15(4,4,4,3) 4 MHZ E.T.: 3.75

Condition Bits Affected:

- S: Not affected
- Z: Not affected
- H: Set if carry out of Bit 11; reset otherwise
- P/V: Not affected
- N: Reset
- C: Set if carry from Bit 15; reset otherwise

Example:

If the contents of Index Register IX are 333H and the contents of register pair BC are 5555H, after the execution of

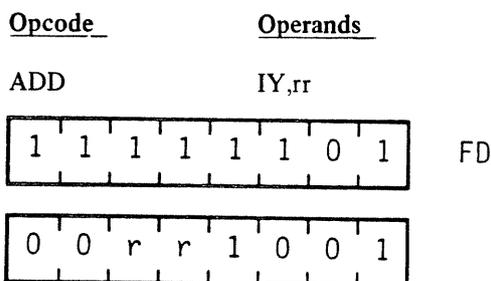
ADD IX, BC

the contents of IX will be 8888H.

ADD IY, rr

Operation: $IY \leftarrow IY + rr$

Format:



Description:

The contents of register pair rr (any of register pairs BC,DE, IY or SP) are added to the contents of Index Register IY, and the result is stored in IY. Operand rr is specified as follows in the assembled object code.

Register Pair	rr
BC	00
DE	01
IY	10
SP	11

M CYCLES: 4 T STATES: 15(4,4,4,3) 4 MHZ E.T.: 3.75

Condition Bits Affected:

S: Not affected
 Z: Not affected
 H: Set if carry out of Bit 11; reset otherwise
 P/V: Not affected
 N: Reset
 C: Set if carry from Bit 15, reset otherwise

Example:

If the contents of Index Register IY are 333H and the contents of register pair BC are 555H, after the execution of

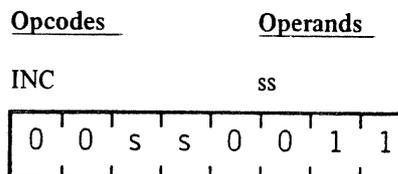
ADD IY,BC

the contents of IY will be 888H.

INC ss

Operation: $ss \leftarrow ss + 1$

Format:



Description:

The contents of register pair ss (any of register pairs BC, DE,HL or SP) are incremented. Operand ss is specified as follows in the assembled object code.

Register Pair	ss
BC	00
DE	01
HL	10
SP	11

M CYCLES: 1 T STATES: 6 4 MHZ E.T. 1.50

Condition Bits Affected: None

Example:

If the register pair contains 1000H, after the execution of

INC HL

HL will contain 1001H.

INC IX

Operation: $IX \leftarrow IX + 1$

Format:

<u>Opcode</u>	<u>Operands</u>	
INC	IX	
1 1 0 1 1 1 0 1		DD
0 0 1 0 0 0 1 1		23

Description:

The contents of the Index Register IX are incremented.

M CYCLES: 2 T STATES: 10(4,6) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If the Index Register IX contains the integer 3300H after the execution of

INC IX

the contents of Index Register IX will be 3301H.

INC IY

Operation: $IY \leftarrow IY + 1$

Format:

<u>Opcode</u>	<u>Operands</u>	
INC	IY	
1 1 1 1 1 1 0 1		FD
0 0 1 0 0 0 1 1		23

Description:

The contents of the Index Register IY are incremented.

M CYCLES: 2 T STATES: 10(4,6) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of the Index Register are 2977H, after the execution of

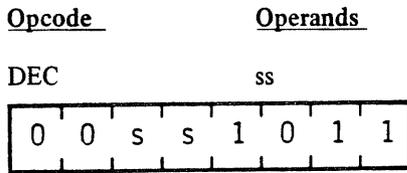
INC IY

the contents of Index Register IY will be 2978H.

DEC ss

Operation: $ss \leftarrow ss - 1$

Format:



Description:

The contents of register pair ss (any of the register pairs BC,DE,HL or SP) are decremented. Operand ss is specified as follows in the assembled object code.

<u>Pair</u>	<u>ss</u>
BC	00
DE	01
HL	10
SP	11

M CYCLES: 1 T STATES: 6 4 MHZ E.T.: 1.50

Condition Bits Affected: None

Example:

If register pair HL contains 1001H, after the execution of

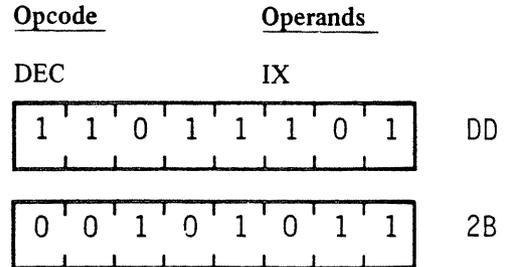
DEC HL

the contents of HL will be 1000H.

DEC IX

Operation: $IX \leftarrow IX - 1$

Format:



Description:

The contents of Index Register IX are decremented.

M CYCLES: 2 T STATES: 10(4,6) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of Index Register IX are 2006H, after the execution of

DEC IX

the contents of Index Register IX will be 2005H.

DEC IY

Operation: $IY \leftarrow IY - 1$

Format:

<u>Opcode</u>	<u>Operands</u>
DEC	IY
1 1 1 1 1 1 0 1	FD
0 0 1 0 1 0 1 1	2B

Description:

The contents of the Index Register IY are decremented.

M CYCLES: 2 T STATES: 10 (4,6) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of the Index Register IY are 7649H, after the execution of

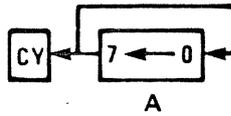
DEC IY

the contents of Index Register IY will be 7648H.

ROTATE AND SHIFT GROUP

RLCA

Operation:

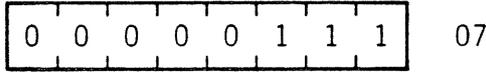


Format:

Opcode

Operands

RLCA



Description:

The contents of the Accumulator (register A) are rotated left: the content of bit 0 is moved to the bit 1; the previous content of bit 1 is moved to bit 2; this pattern is continued throughout the register. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. (Bit 0 is the least significant bit.)

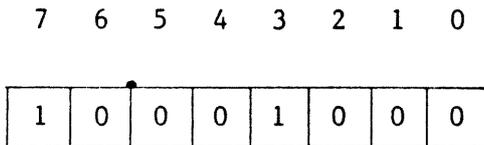
M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

- S: Not affected
- Z: Not affected
- H: Reset
- P/V: Not affected
- N: Reset
- C: Data from Bit 7 of Acc.

Example:

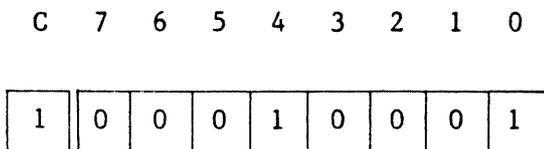
If the contents of the Accumulator are



after the execution of

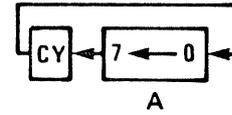
RLCA

the contents of the Accumulator and Carry Flag will be



RLA

Operation:

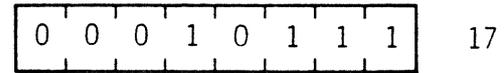


Format:

Opcode

Operands

RLA



Description:

The contents of the Accumulator (register A) are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the register. The content of bit 7 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 0. Bit 0 is the least significant bit.

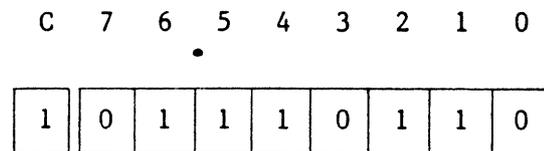
M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

- S: Not affected
- Z: Not affected
- H: Reset
- P/V: Not affected
- N: Reset
- C: Data from Bit 7 of Acc.

Example:

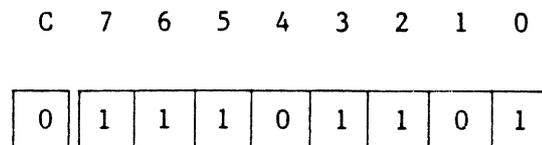
If the contents of the Accumulator and the Carry Flag are



after the execution of

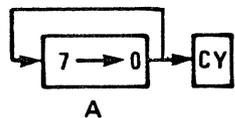
RLA

the contents of the Accumulator and the Carry Flag will be



RRCA

Operation:

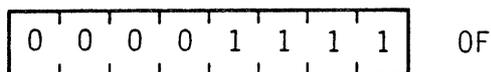


Format:

Opcode

Operands

RRCA



Description:

The contents of the Accumulator (register A) is rotated right: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the register. The content of bit 0 is copied into bit 7 and also into the Carry Flag (C flag in register F.) Bit 0 is the least significant bit.

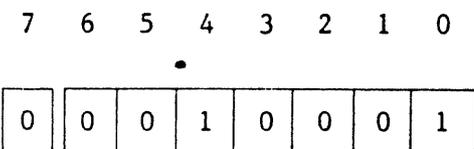
M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

- S: Not affected
- Z: Not affected
- H: Reset
- P/V: Not affected
- N: Reset
- C: Data from Bit 0 of Acc.

Example:

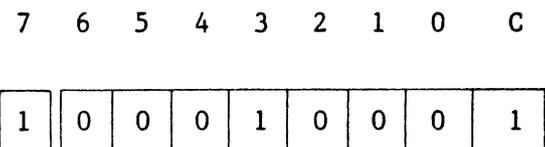
If the contents of the Accumulator are



After the execution of

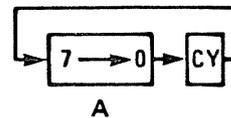
RRCA

the contents of the Accumulator and the Carry Flag will be



RRA

Operation:

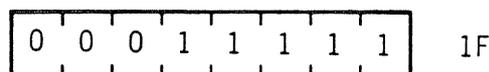


Format:

Opcode

Operands

RRA



Description:

The contents of the Accumulator (register A) are rotated right: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the register. The content of bit 0 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 7. Bit 0 is the least significant bit.

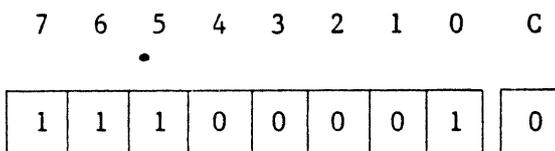
M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected:

- S: Not affected
- Z: Not affected
- H: Reset
- P/V: Not affected
- N: Reset
- C: Data from Bit 0 of Acc.

Example:

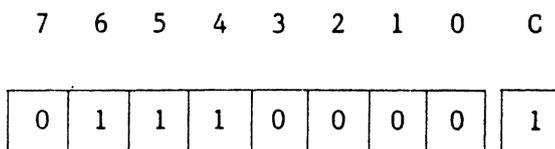
If the contents of the Accumulator and the Carry Flag are



after the execution of

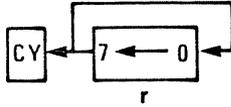
RRA

the contents of the Accumulator and the Carry Flag will be

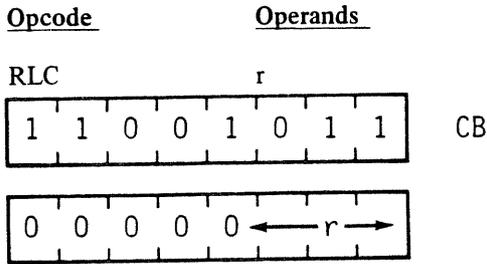


RLC r

Operation:



Format:



Description:

The eight-bit contents of register r are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the register. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Operand r is specified as follows in the assembled object code:

Register r

B	000
C	001
D	010
E	011
H	100
L	101
A	111

Note: Bit 0 is the least significant bit.

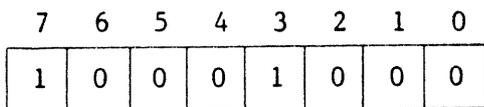
M CYCLES: 2 T STATES: 8(4,4) 4 MHZ E.T.: 2.00

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Reset
- P/V: Set if parity even; reset otherwise
- N: Reset
- C: Data from Bit 7 of source register

Example:

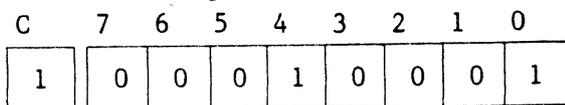
If the contents of register r are



after the execution of

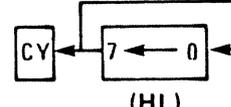
RLC r

the contents of register r and the Carry Flag will be

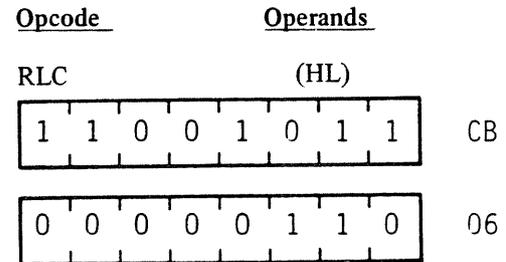


RLC (HL)

Operation:



Format:



Description:

The contents of the memory address specified by the contents of register pair HL are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Bit 0 is the least significant bit.

M CYCLES: 4 T STATES: 15(4,4,4,3) 4 MHZ E.T.: 3.75

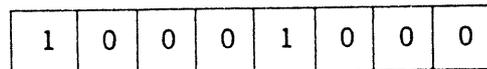
Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Reset
- P/V: Set if parity even; reset otherwise
- N: Reset
- C: Data from Bit 7 of source register

Example:

If the contents of the HL register pair are 2828H, and the contents of memory location 2828H are

7 6 5 4 3 2 1 0

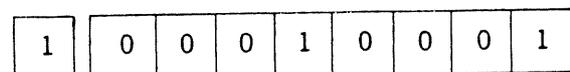


after the execution of

RLC (HL)

the contents of memory locations 2828H and the Carry Flag will be

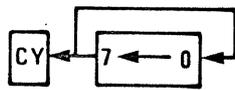
C 7 6 5 4 3 2 1 0



RLC (IX+d)

Operation:

Format:

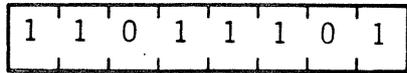


Opcode

(IX+d)
Operands

RLC

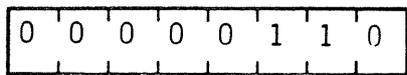
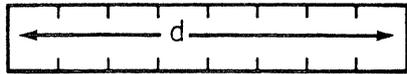
(IX+d)



DD



CB



06

Description:

The contents of the memory address specified by the sum of the contents of the Index Register IX and a two's complement displacement integer d, are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Bit 0 is the least significant bit.

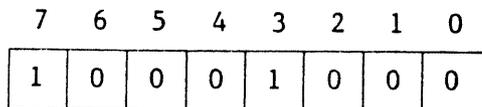
M CYCLES: 6 T STATES: 23(4,4,3,5,4,3) 4 MHZ E.T.: 5.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Reset
- P/V: Set if parity even; reset otherwise
- N: Reset
- C: Data from Bit 7 of source register

Example:

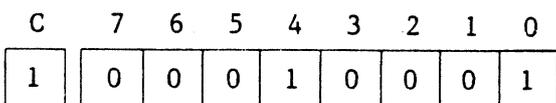
If the contents of the Index Register IX are 1000H, and the contents of memory location 1022H are



after the execution of

RLC (IX+2H)

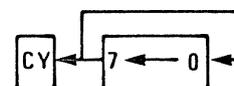
the contents of memory location 1002H and the Carry Flag will be



RLC (IY+d)

Operation:

Format:

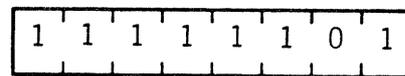


Opcode

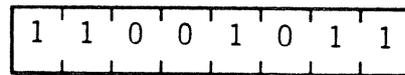
(IY+d)
Operands

RLC

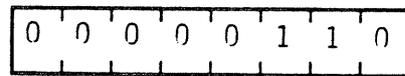
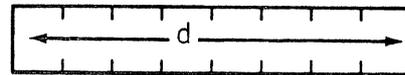
(IY+d)



FD



CB



06

Description:

The contents of the memory address specified by the sum of the contents of the Index Register IY and a two's complement displacement integer d are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this process is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and also into bit 0. Bit 0 is the least significant bit.

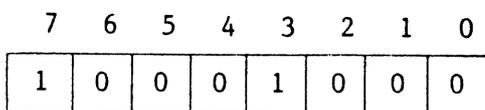
M CYCLES: 6 T STATES: 23(4,4,3,5,4,3) 4 MHZ E.T.: 5.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Reset
- P/V: Set if parity even; reset otherwise
- N: Reset
- C: Data from Bit 7 of source register

Example:

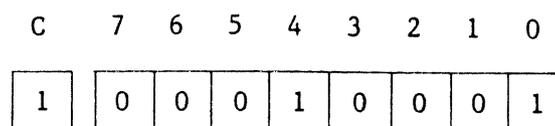
If the contents of the Index Register IY are 1000H, and the contents of memory location 1002H are



after the execution of

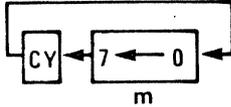
RLC (IY+2H)

the contents of memory location 1002H and the Carry Flag will be



RL m

Operation:



Format:

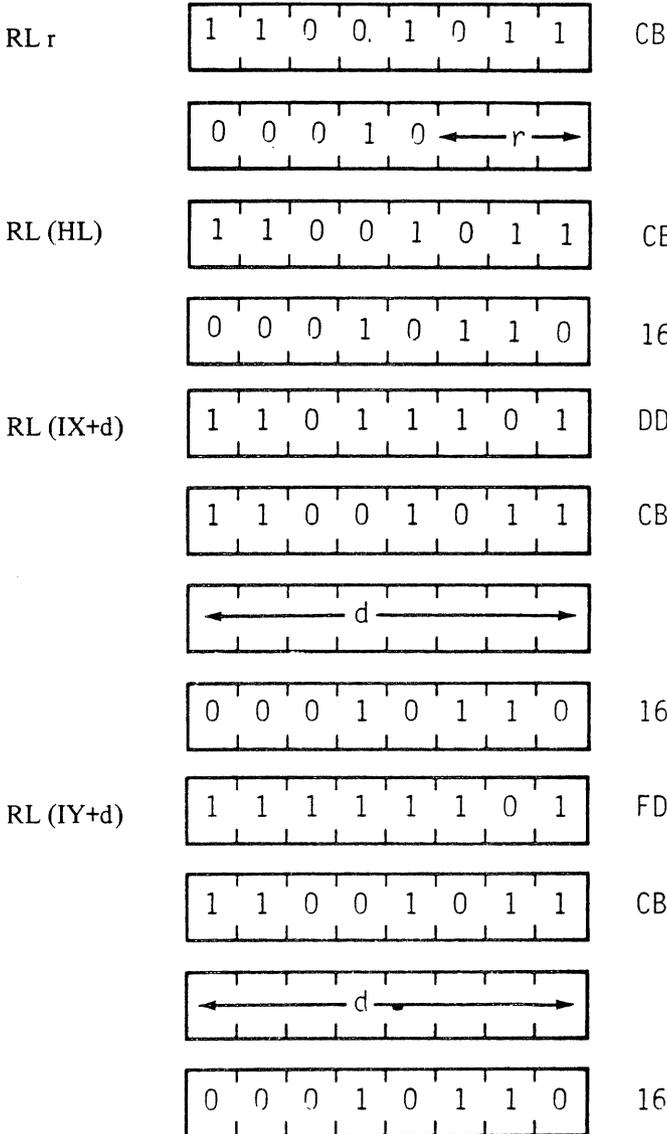
Opcode

Operands

RL

m

The m operand is any of r,(HL),(IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



*r identifies register B,C,D,E,H,L or A specified as follows in the assembled object code above:

Register r

B	000
C	001
D	010

E	011
H	011
L	101
A	111

Description:

The contents of the m operand are rotated left: the content of bit 0 is copied into bit 1; the previous content of bit 1 is copied into bit 2; this pattern is continued throughout the byte. The content of bit 7 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 0 (Bit 0 is the least significant bit.)

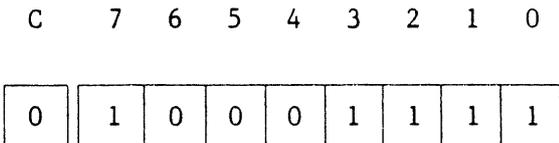
INSTRUCTION	M		4 MHZ E.T.
	CYCLES	T STATES	
RL r	2	8(4,4)	2.00
RL (HL)	4	15(4,4,4,3)	3.75
RL (IX+d)	6	23(4,4,3,5,4,3)	5.75
RL (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Reset
- P/V: Set if parity even; reset otherwise
- N: Reset
- C: Data from Bit 7 of source register

Example:

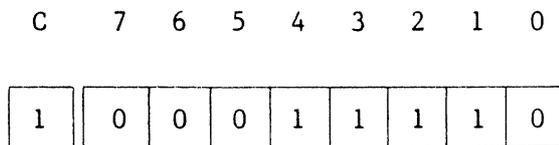
If the contents of register D and the Carry Flag are



after the execution of

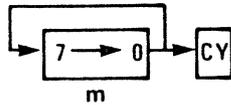
RL D

the contents of register D and the Carry Flag will be



RRC m

Operation:



Format:

Opcode	Operands
RRC	m

The m operand is any of r,(HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

RRC r	1 1 0 0 1 0 1 1	CB
	0 0 0 0 1 ← r →	
RRC (HL)	1 1 0 0 1 0 1 1	CB
	0 0 0 0 1 1 1 0	OE
RRC (IX+d)	1 1 0 1 1 1 0 1	DD
	1 1 0 0 1 0 1 1	CB
	← d →	
	0 0 0 0 1 1 1 0	OE
RRC (IY+d)	1 1 1 1 1 1 0 1	FD
	1 1 0 0 1 0 1 1	CB
	← d →	
	0 0 0 0 1 1 1 0	OE

*r identifies register B,C,D,E,H,L or A specified as follows in the assembled object code above:

Register r

B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of operand m are rotated right: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag (C flag in the F register) and also into bit 7. Bit 0 is the least significant bit.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
RRC r	2	8(4,4)	2.00
RRC (HL)	4	15(4,4,4,3)	3.75
RRC (IX+d)	6	23(4,4,3,5,4,3)	5.75
RRC (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity even; reset otherwise
N:	Reset
C:	Data from Bit 0 of source register

Example:

If the contents of register A are

7 6 5 4 3 2 1 0

0	0	1	1	0	0	0	1
---	---	---	---	---	---	---	---

after the execution of

RRC A

the contents of register A and the Carry Flag will be

7 6 5 4 3 2 1 0 C

1	0	0	1	1	0	0	0	1
---	---	---	---	---	---	---	---	---

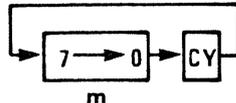
RR m

Operation:

Format:

Opcode

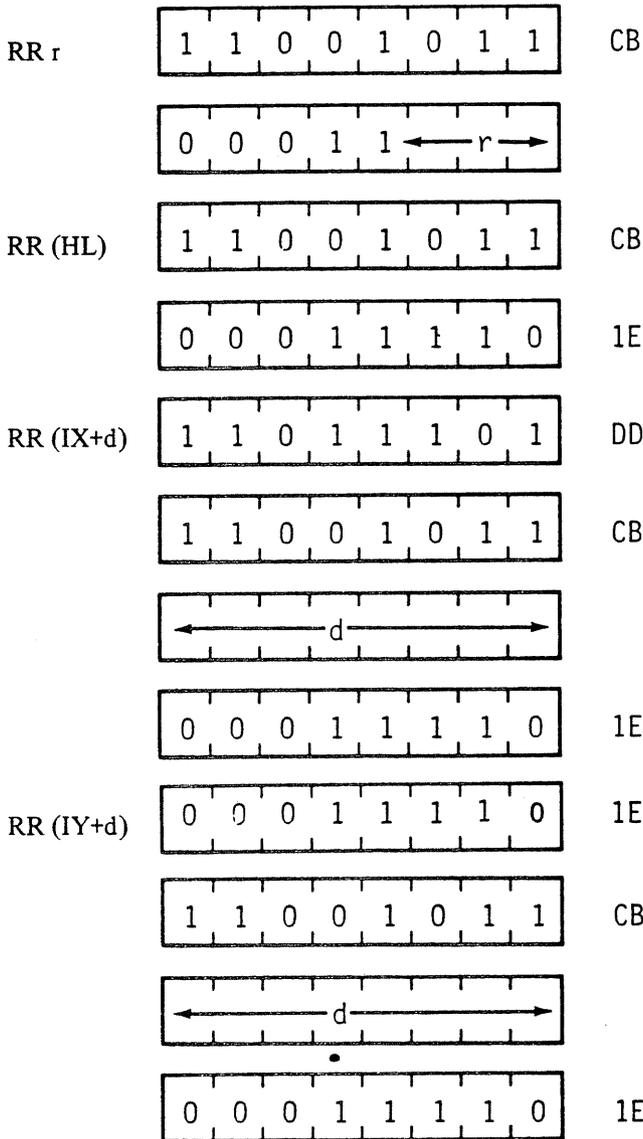
RR



Operand

m

The m operand is any of r, (HL), (IX+d), or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



*r identifies registers B,C,D,E,H,L or A specified as follows in the assembled object code above:

Register r

B	000
C	001
D	010
E	011
H	100
L	101
A	111

Description:

The contents of operand m are rotated right: the contents of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag (C flag in register F) and the previous content of the Carry Flag is copied into bit 7. Bit 0 is the least significant bit.

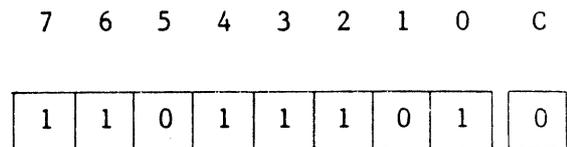
INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
RR r	2	8(4,4)	2.00
RR (HL)	4	15(4,4,4,3)	3.75
RR (IX+d)	6	23(4,4,3,5,4,3)	5.75
RR (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Reset
- P/V: Set if parity is even; reset otherwise
- N: Reset
- C: Data from Bit 0 of source register

Example:

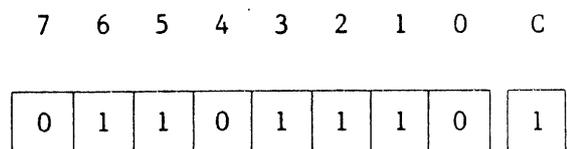
If the contents of the HL register pair are 4343H, and the contents of memory location 4343H and the Carry Flag are



after the execution of

RR (HL)

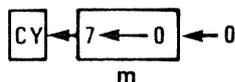
the contents of location 4343H and the Carry Flag will be



SLA m

Operation:

Format:



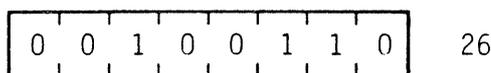
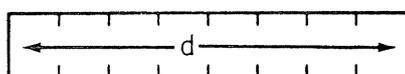
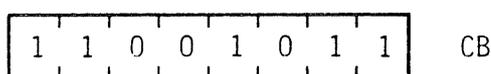
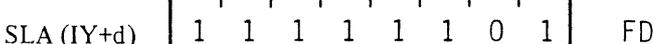
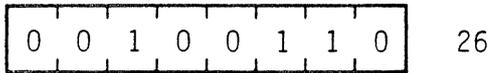
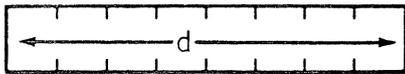
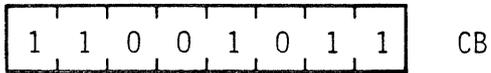
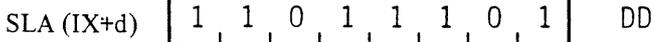
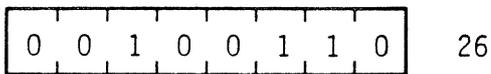
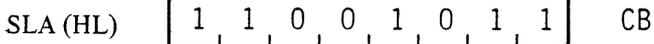
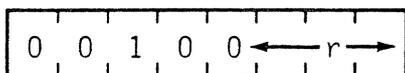
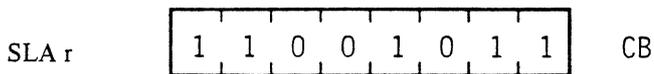
Opcode

Operands

SLA

m

The m operand is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



*r identifies registers B,C,D,E,H,L or A specified as follows in the assembled object code field above:

Register r

B 000
C 001
D 010

E 011
H 100
L 101
A 111

Description:

An arithmetic shift left is performed on the contents of operand m: bit 0 is reset, the previous content of bit 0 is copied into bit 1, the previous content of bit 1 is copied into bit 2; this pattern is continued throughout; the content of bit 7 is copied into the Carry Flag (C flag in register F). Bit 0 is the least significant bit.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
SLA r	2	8(4,4)	2.00
SLA (HL)	4	15(4,4,4,3)	3.75
SLA (IX+d)	6	23(4,4,3,5,4,3)	5.75
SLA (IY+d)	6	23(4,4,3,5,4,3)	5.75

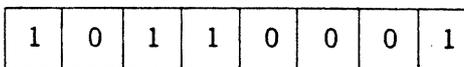
Condition Bits Affected:

S: Set if result is negative; reset otherwise
Z: Set if result is zero; reset otherwise
H: Reset
P/V: Set if parity is even; reset otherwise
N: Reset
C: Data from Bit 7

Example:

If the contents of register L are

7 6 5 4 3 2 1 0

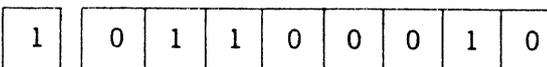


after the execution of

SLA L

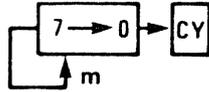
the contents of register L and the Carry Flag will be

C 7 6 5 4 3 2 1 0



SRA m

Operation:



Format:

Opcode	Operands
SRA	m

The m operand is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:

SRA r	1 1 0 0 1 0 1 1	CB
	0 0 1 0 1 ← r →	
SRA(HL)	1 1 0 0 1 0 1 1	CB
	0 0 1 0 1 1 1 0	2E
SRA (IX+d)	1 1 0 1 1 1 0 1	DD
	1 1 0 0 1 0 1 1	CB
	← d →	
	0 0 1 0 1 1 1 0	2E
SRA(IY+d)	1 1 1 1 1 1 0 1	FD
	1 1 0 0 1 0 1 1	CB
	← d →	
	0 0 1 0 1 1 1 0	2E

*r means register B,C,D,E,H,L or A specified as follows in the assembled object code field above:

Register	r
B	000
C	001
D	010

E	011
H	100
L	101
A	111

An arithmetic shift right is performed on the contents of operand m: the content of bit 7 is copied into bit 6; the previous content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag (C flag in register F), and the previous content of bit 7 is unchanged. Bit 0 is the least significant bit.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
SRA r	2	8(4,4)	2.00
SRA (HL)	4	15(4,4,4,3)	3.75
SRA (IX+d)	6	23(4,4,3,5,4,3)	5.75
SRA (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected:

- S: Set if result is negative; reset otherwise
- Z: Set if result is zero; reset otherwise
- H: Reset
- P/V: Set if parity is even; reset otherwise
- N: Reset
- C: Data from Bit 0 of source register

Example:

If the contents of the Index Register IX are 1000H, and the contents of memory location 1003H are

7	6	5	4	3	2	1	0
1	0	1	1	1	0	0	0

after the execution of

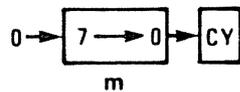
SRA (IX+3H)

the contents of memory location 1003H and the Carry Flag will be

7	6	5	4	3	2	1	0	C
1	1	0	1	1	1	0	0	0

SRL m

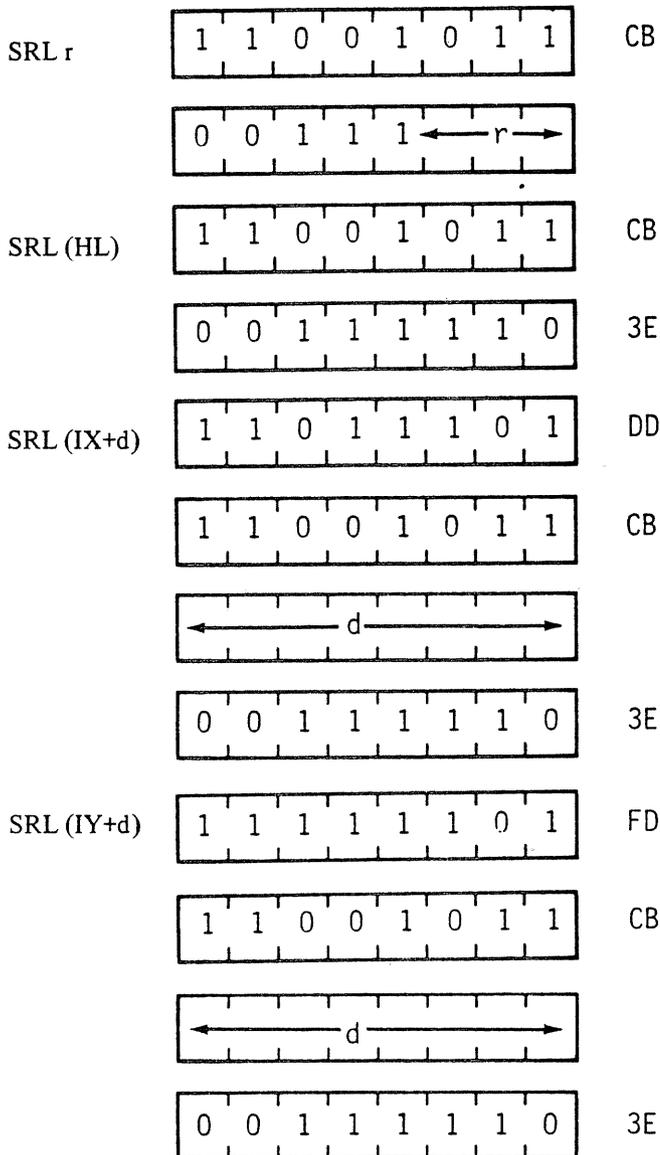
Operation:



Format:

Opcode	Operands
SRL	m

The operand m is any of r, (HL), (IX+d) or (IY+d), as defined for the analogous RLC instructions. These various possible opcode-operand combinations are specified as follows in the assembled object code:



*r identifies registers B,C,D,E,H,L or A specified as follows in the assembled object code fields above:

Register	r
B	000
C	001
D	010

E	011
H	100
L	101
A	111

Description:

The contents of operand m are shifted right: the content of bit 7 is copied into bit 6; the content of bit 6 is copied into bit 5; this pattern is continued throughout the byte. The content of bit 0 is copied into the Carry Flag, and bit 7 is reset. Bit 0 is the least significant bit.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
SRL r	2	8(4,4)	2.00
SRL (HL)	4	15(4,4,4,3)	3.75
SRL (IX+d)	6	23(4,4,3,5,4,3)	5.75
SRL (IY+d)	6	23(4,4,3,5,4,3)	5.75

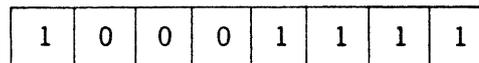
Condition Bits Affected:

S:	Set if result is negative; reset otherwise
Z:	Set if result is zero; reset otherwise
H:	Reset
P/V:	Set if parity is even; reset otherwise
N:	Reset
C:	Data from Bit 0 of source register

Example:

If the contents of register B are

7 6 5 4 3 2 1 0

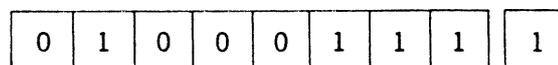


after the execution of

SRL B

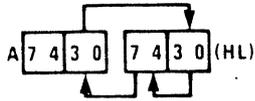
the contents of register B and the Carry Flag will be

7 6 5 4 3 2 1 0 c



RLD

Operation:

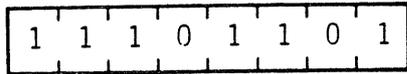


Format:

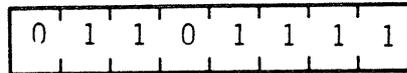
Opcode

Operands

RLD



ED



6F

Description:

The contents of the low order four bits (bits 3,2,1 and 0) of the memory location (HL) are copied into the high order four bits (7,6,5 and 4) of that same memory location; the previous contents of those high order four bits are copied into the low order four bits of the Accumulator (register A), and the previous contents of the low order four bits of the Accumulator are copied into the low order four bits of memory location (HL). The contents of the high order bits of the Accumulator are unaffected. Note: (HL) means the memory location specified by the contents of the HL register pair.

M CYCLES: 5 T STATES: 18(4,4,3,4,3) 4 MHZ E.T.: 4.50

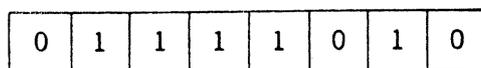
Condition Bits Affected:

- S: Set if Acc. is negative after operation; reset otherwise
- Z: Set if Acc. is zero after operation; reset otherwise
- H: Reset
- P/V: Set if parity of Acc. is even after operation; reset otherwise
- N: Reset
- C: Not affected

Example:

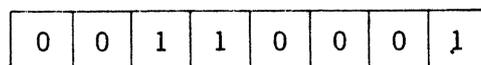
If the contents of the HL register pair are 5000H, and the contents of the Accumulator and memory location 5000H are

7 6 5 4 3 2 1 0



Accumulator

7 6 5 4 3 2 1 0



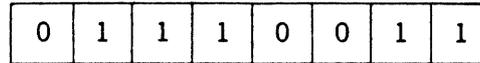
(5000H)

after the execution of

RLD

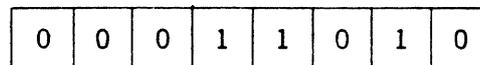
the contents of the Accumulator and memory location 5000H will be

7 6 5 4 3 2 1 0



Accumulator

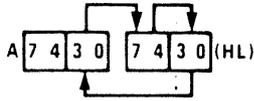
7 6 5 4 3 2 1 0



(5000H)

RRD

Operation:



Format:

Opcode

Operands

RRD

1	1	1	0	1	1	0	1
---	---	---	---	---	---	---	---

ED

0	1	1	0	0	1	1	1
---	---	---	---	---	---	---	---

67

Description:

The contents of the low order four bits (bits 3,2,1 and 0) of memory location (HL) are copied into the low order four bits of the Accumulator (register A); the previous contents of the low order four bits of the Accumulator are copied into the high order four bits (7,6,5 and 4) of location (HL); and the previous contents of the high order four bits of (HL) are copied into the low order four bits of (HL). The contents of the high order bits of the Accumulator are unaffected. Note: (HL) means the memory location specified by the contents of the HL register pair.

M CYCLES: 5 T STATES: 18(4,4,3,4,3) 4 MHZ E.T.: 4.50

Condition Bits Affected:

- S: Set if Acc. is negative after operation; reset otherwise
- Z: Set if Acc, is zero after operation; reset otherwise
- H: Reset
- P/V: Set if parity of Acc. is even after operation; reset otherwise
- N: Reset
- C: Not affected

Example:

If the contents of the HL register pair are 5000H, and the contents of the Accumulator and memory location 5000H are

7 6 5 4 3 2 1 0

1	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

Accumulator

7 6 5 4 3 2 1 0

0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---

(5000H)

after the execution of

RRD

the contents of the Accumulator and memory location 5000H will be

7 6 5 4 3 2 1 0

1	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Accumulator

7 6 5 4 3 2 1 0

0	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---

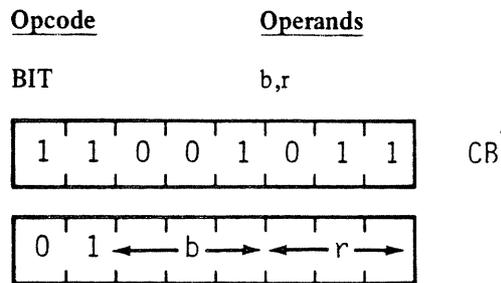
(5000H)

BIT SET, RESET AND TEST GROUP

BIT b, r

Operation: $Z \leftarrow \overline{r}_b$

Format:



Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the indicated register. Operands b and r are specified as follows in the assembled object code:

Bit Tested	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	111
7	111		

M CYCLES: 2 T STATES: 8(4,4) 4 MHZ E.T.: 2.00

Condition Bits Affected:

S: Unknown
 Z: Set if specified Bit is 0; reset otherwise
 H: Set
 P/V: Unknown
 N: Reset
 C: Not affected

Example:

If bit 2 in register B contains 0, after the execution of

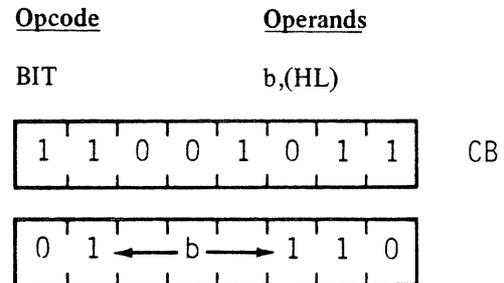
BIT 2, B

the Z flag in the F register will contain 1, and bit 2 in register B will remain 0. Bit 0 in register B is the least significant bit.

BIT b, (HL)

Operation: $Z \leftarrow \overline{(HL)}_b$

Format:



Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the HL register pair. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M CYCLES: 3 T STATES: 12(4,4,4) 4 MHZ E.T.: 3.00

Condition Bits Affected:

S: Unknown
 Z: Set if specified Bit is 0; reset otherwise
 H: Set
 P/V: Unknown
 H: Reset
 C: Not affected

Example:

If the HL register pair contains 4444H, and bit 4 in the memory location 444H contains 1, after the execution of

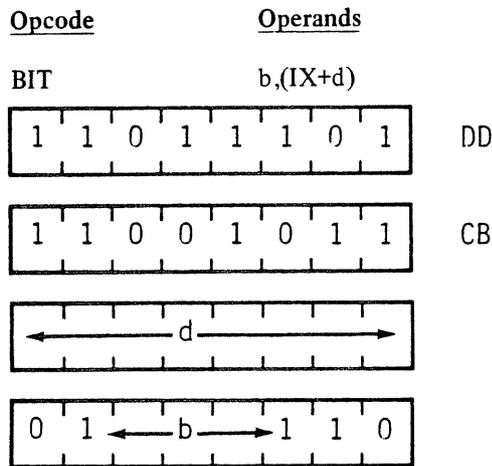
BIT 4, (HL)

the Z flag in the F register will contain 0, and bit 4 in memory location 444H will still contain 1. (Bit 0 in memory location 444H is the least significant bit.)

BIT b, (IX+d)

Operation: $Z \leftarrow \overline{(IX+d)}_b$

Format:



Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the memory location pointed to by the sum of the contents register pair IX (Index Register IX) and the two's complement displacement integer d. Operand b is specified as follows in the assembled object code.

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M CYCLES: 5 T STATES: 20(4,4,3,5,4) 4 MHZ E.T.: 5.00

Condition Bits Affected:

S: Unknown
 Z: Set if specified Bit is 0; reset otherwise
 H: Set
 P/V: Unknown
 N: Reset
 C: Not affected

Example:

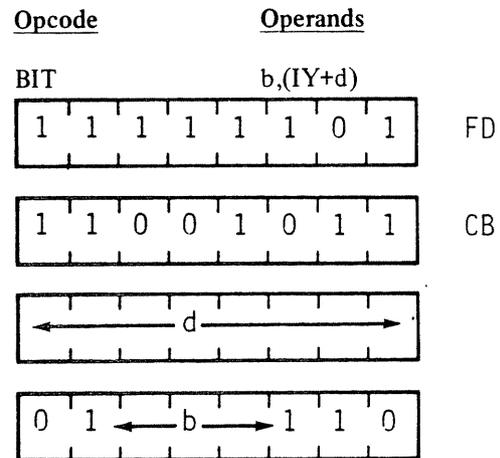
If the contents of Index Register IX are 2000H, and bit 6 in memory location 2004H contains 1, after the execution of **BIT 6, (IX+4H)**

the Z flag in the F register will contain 0, and bit 6 in memory location 2004H will still contain 1. (Bit 0 in memory location 2004H is the least significant bit.)

BIT b, (IY+d)

Operation: $Z \leftarrow \overline{(IY+d)}_b$

Format:



Description:

After the execution of this instruction, the Z flag in the F register will contain the complement of the indicated bit within the contents of the memory location pointed to by the sum of the contents of register pair IY (Index Register IY) and the two's complement displacement integer d. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M CYCLES: 5 T STATES: 20(4,4,3,5,4) 4 MHZ E.T.: 5.00

Condition Bits Affected:

S: Unknown
 Z: Set if specified Bit is 0; reset otherwise
 H: Set
 P/V: Unknown
 N: Reset
 C: Not affected

Example:

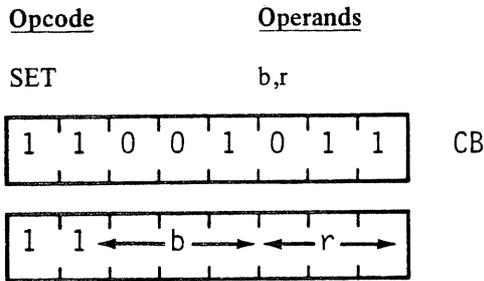
If the contents of Index Register are 2000H, and bit 6 in memory location 2004H contains 1, after the execution of **BIT 6, (IY+4H)**

the Z flag in the F register still contain 0, and bit 6 in memory location 2004H will still contain 1. (Bit 0 in memory location 2004H is the least significant bit.)

SET b, r

Operation: $r_b \leftarrow 1$

Format:



Description:

Bit b (any bit, 7 through 0) in register r (any of register B,C,D,E,H,L or A) is set. Operands b and r are specified as follows in the assembled object code:

Bit	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	110
7	111		

M CYCLES: 2 T STATES: 8(4,4) 4 MHZ E.T.: 2.00

Condition Bits Affected: None

Example:

After the execution of

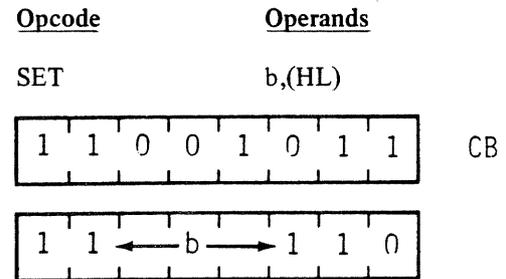
SET 4, A

bit 4 in register A will be set. (Bit 0 is the least significant bit.)

SET b, (HL)

Operation: $(HL)_b \leftarrow 1$

Format:



Description:

Bit b (any bit, 7 through 0) in the memory location addressed by the contents of register pair HL is set. Operand b is specified as follows in the assembled object code:

Bit	b
Tested	
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M CYCLES: 4 T STATES: 15(4,4,4,3) 4 MHZ E.T.: 3.75

Condition Bits Affected: None

Example:

If the contents of the HL register pair are 3000H, after the execution of

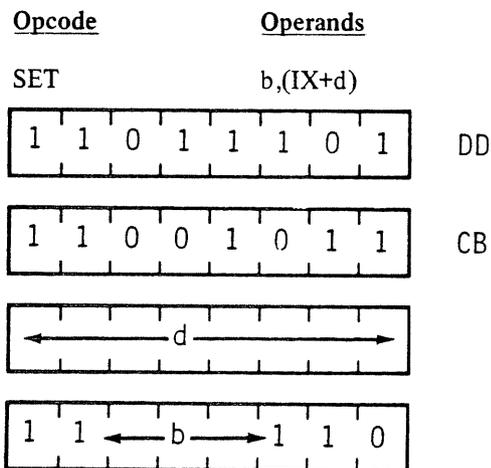
SET 4, (HL)

bit 4 in memory location 3000H will be 1. (Bit 0 in memory location 3000H is the least significant bit.)

SET b, (IX+d)

Operation: $(IX+d)_b \leftarrow 1$

Format:



Description:

Bit b (any bit, 7 through 0) in the memory location addressed by the sum of the contents of the IX register pair (Index Register IX) and the two's complement integer d is set. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3) 4 MHZ E.T.: 5.75

Condition Bits Affected: None

Example:

If the contents of Index Register are 2000H, after the execution of

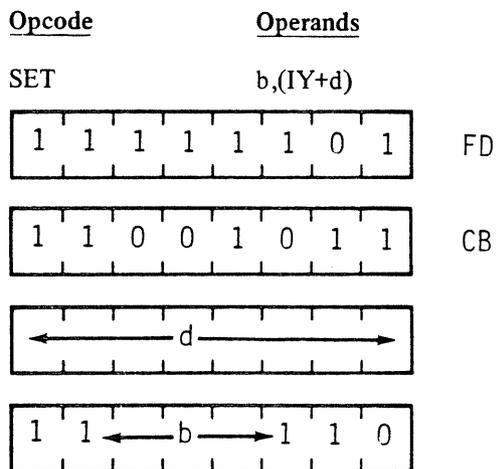
SET 0, (IX+3H)

bit 0 in memory location 2003H will be 1. (Bit 0 in memory location 2003H is the least significant bit.)

SET b, (IY+d)

Operation: $(IY+d)_b \leftarrow 1$

Format:



Description:

Bit b (any bit, 7 through 0) in the memory location addressed by the sum of the contents of the IY register pair (Index Register IY) and the two's complement displacement d is set. Operand b is specified as follows in the assembled object code:

Bit Tested	b
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

M CYCLES: 6 T STATES: 23(4,4,3,5,4,3) 4 MHZ E.T.: 5.75

Condition Bits Affected: None

Example:

If the contents of Index Register IY are 2000H, after the execution of

SET 0, (IY+3H)

bit 0 in memory location 2003H will be 1. (Bit 0 in memory location 2003H is the least significant bit.)

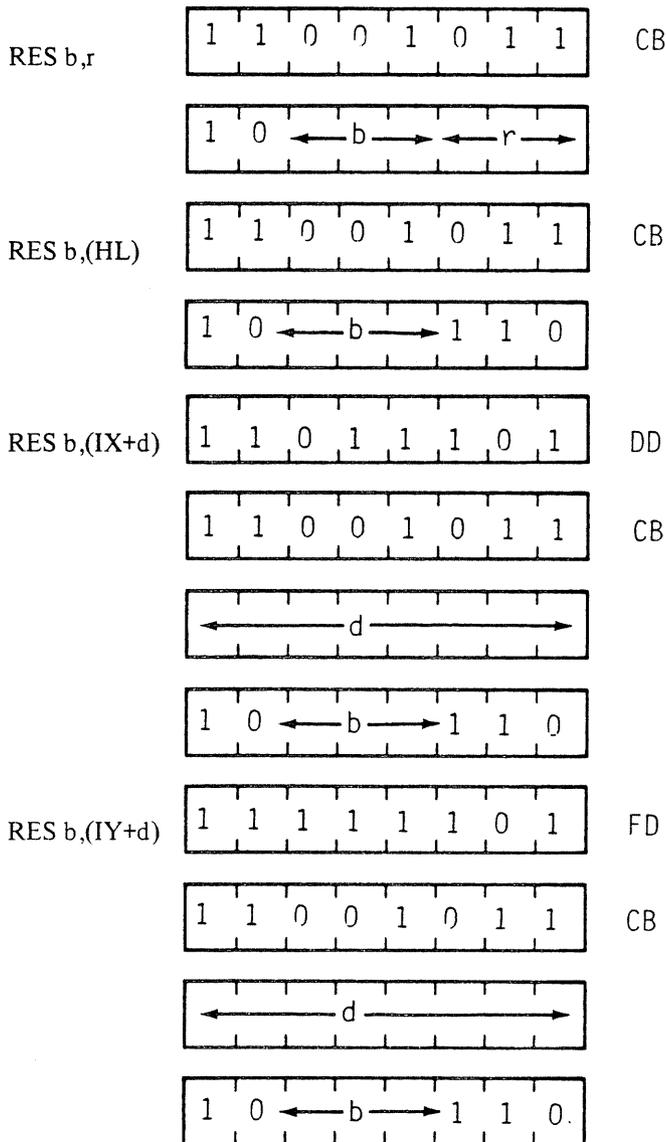
RES b, m

Operation: $s_b \leftarrow 0$

Format:

Opcode	Operands
RES	b,m

Operand b is any bit (7 through 0) of the contents of the m operand, (any of r, (HL), (IX+d) or (IY+d) as defined for the analogous SET instructions. These various possible opcode-operand combinations are assembled as follows in the object code:



Bit Reset	b	Register	r
0	000	B	000
1	001	C	001
2	010	D	010
3	011	E	011
4	100	H	100
5	101	L	101
6	110	A	110
7	111		

Description:

Bit b in operand m is reset.

INSTRUCTION	M CYCLES	T STATES	4 MHZ E.T.
RES r	4	8(4,4)	2.00
RES (HL)	4	15(4,4,4,3)	3.75
RES (IX+d)	6	23(4,4,3,5,4,3)	5.75
RES (IY+d)	6	23(4,4,3,5,4,3)	5.75

Condition Bits Affected: None

Example:

After the execution of

RES 6,D

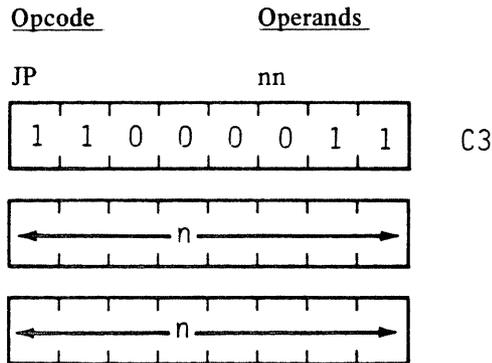
bit 6 in register D will be reset. (Bit 0 in register D is the least significant bit.)

JUMP GROUP

JP nn

Operation: PC ← nn

Format:



Note: The first operand in this assembled object code is the low order byte of a 2-byte address.

Description:

Operand nn is loaded into register pair PC (Program Counter) and points to the address of the next program instruction to be executed.

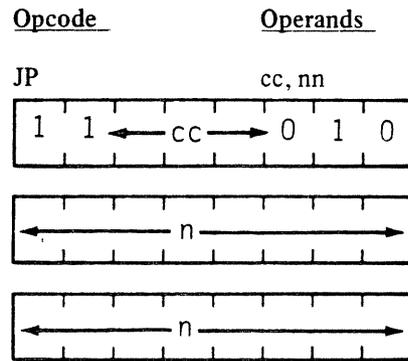
M CYCLES: 3 T STATES: 10(4,3,3) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

JP cc, nn

Operation: IF cc TRUE, PC ← nn

Format:



Note: The first n operand in this assembled object code is the low order byte of a 2-byte memory address.

Description:

If condition cc is true, the instruction loads operand nn into register pair PC (Program Counter), and the program continues with the instruction beginning at address nn. If condition cc is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. Condition cc is programmed as one of eight status which corresponds to condition bits in the Flag Register (register F). These eight status are defined in the table below which also specifies the corresponding cc bit fields in the assembled object code.

cc	CONDITION	RELEVANT FLAG
000	NZ non zero	Z
001	Z zero	Z
010	NC no carry	C
011	C carry	C
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

M CYCLES: 3 T STATES: 10(4,3,3) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

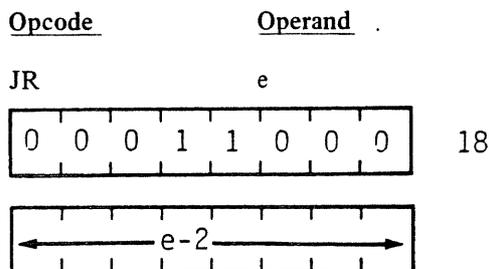
If the Carry Flag (C flag in the F register) is set and the contents of address 1520 are 03H, after the execution of JP C, 1520H

the Program Counter will contain 1520H, and on the next machine cycle the CPU will fetch from address 1520H the byte 03H.

JR e

Operation: $PC \leftarrow PC + e$

Format:



Description:

This instruction provides for unconditional branching to other segments of a program. The value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. This jump is measured from the address of the instruction opcode and has a range of -126 to $+129$ bytes. The assembler automatically adjusts for the twice incremented PC.

M CYCLES: 3 T STATES: 12(4,3,5) 4 MHZ E.T.: 3.00

Condition Bits Affected: None

Example:

To jump forward 5 locations from address 480, the following assembly language statement is used:

JR \$+5

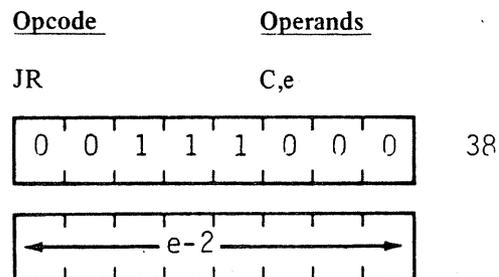
The resulting object code and final PC value is shown below:

Location	Instruction
480	18
481	03
482	—
483	—
484	—
485	\leftarrow PC after jump

JR C, e

Operation: If $C = 0$, continue
If $C = 1$, $PC \leftarrow PC + e$

Format:



Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is equal to a '1', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to $+129$ bytes. The assembler automatically adjusts for the twice incremented PC.

If the flag is equal to a '0', the next instruction to be executed is taken from the location following this instruction.

If condition is met:

M CYCLES: 3 T STATES: 12(4,3,5) 4 MHZ E.T.: 3.00

If condition is not met:

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

The Carry Flag is set and it is required to jump back 4 locations from 480. The assembly language statement is:

JR C, \$-4

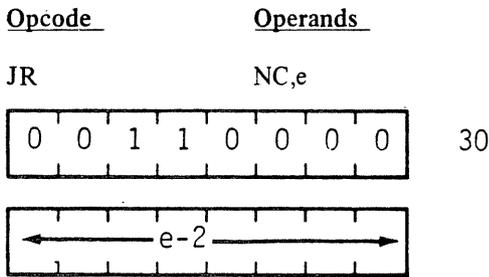
The resulting object code and final PC value is shown below:

Location	Instruction
47C	\leftarrow PC after jump
47D	—
47E	—
47F	—
480	38
481	FA (2's complement -6)

JR NC, e

Operation: If C = 1, continue
If C = 0, PC ← PC + e

Format:



Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Carry Flag. If the flag is equal to '0', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 byte. The assembler automatically adjusts for the twice incremented PC.

If the flag is equal to a '1', the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M CYCLES: 3 T STATES: 12(4,3,5) 4 MHZ E.T.: 3.00

If the condition is not met:

M CYCLES: 7 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

The Carry Flag is reset and it is required to repeat the jump instruction. The assembly language statement is:

JR NC, \$

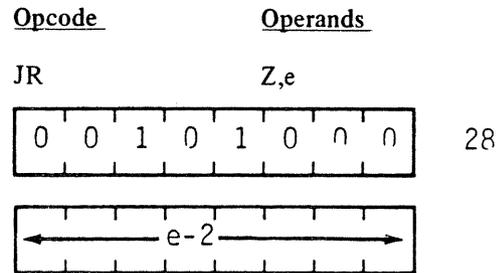
The resulting object code and PC after the jump are shown below:

Location	Instruction
480	30 ← PC after jump
481	00

JR Z, e

Operation: If Z = 0, continue
If Z = 1, PC ← PC + e

Format:



Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag is equal to a '1', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the Zero Flag is equal to a '0', the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M CYCLES: 3 T STATES: 12(4,3,5) 4 MHZ E.T.: 3.00

If the condition is not met:

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

The Zero Flag is set and it is required to jump forward 5 locations from address 300. The following assembly language statement is used:

JR Z, \$ +5

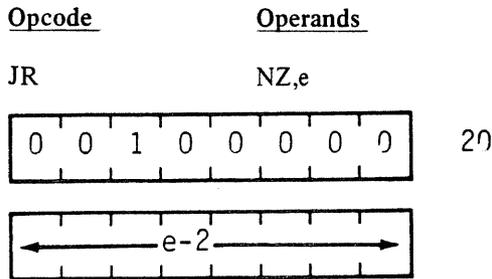
The resulting object code and final PC value is shown below:

Location	Instruction
300	28
301	03
302	—
303	—
304	—
305	← PC after jump

JR NZ, e

Operation: If Z = 1, continue
If Z = 0, PC ← PC + e

Format:



Description:

This instruction provides for conditional branching to other segments of a program depending on the results of a test on the Zero Flag. If the flag is equal to a '0', the value of the displacement e is added to the Program Counter (PC) and the next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the Zero Flag is equal to a '1', the next instruction to be executed is taken from the location following this instruction.

If the condition is met:

M CYCLES: 3 T STATES: 12(4,3,5) 4 MHZ E.T.: 3.00

If the condition is not met:

M CYCLES: 2 T STATES: 7(4,3) 4 MHZ E.T.: 1.75

Condition Bits Affected: None

Example:

The Zero Flag is reset and it is required to jump back 4 locations from 480. The assembly language statement is:

JR NZ, \$-4

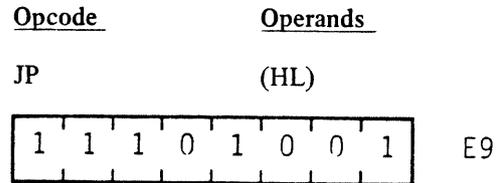
The resulting object code and final PC value is shown below:

Location	Instruction
47C	← PC after jump
47D	—
47E	—
47F	—
480	20
481	FA (2' complement-6)

JP (HL)

Operation: PC ← HL

Format:



Description:

The Program Counter (register pair PC) is loaded with the contents of the HL register pair. The next instruction is fetched from the location designated by the new contents of the PC.

M CYCLES: 1 T STATES: 4 4 MHZ E.T.: 1.00

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1000H and the contents of the HL register pair are 4800H, after the execution of

JP (HL)

the contents of the Program Counter will be 4800H.

JP (IX)

Operation: PC ← IX

Format:

Opcode

Operands

JP

(IX)

1	1	0	1	1	1	0	1
---	---	---	---	---	---	---	---

DD

1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---

E9

Description:

The Program Counter (register pair PC) is loaded with the contents of the IX Register Pair (Index Register IX). The next instruction is fetched from the location designated by the new contents of the PC.

M CYCLES: 2 T STATES: 8(4,4) 4 MHZ E.T.: 2.00

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1000H, and the contents of the IX Register Pair are 4800H, after the execution of

JP (IX)

the contents of the Program Counter will be 4800H.

JP (IY)

Operation: PC ← IY

Format:

Opcode

Operands

JP

(IY)

1	1	1	1	1	1	0	1
---	---	---	---	---	---	---	---

FD

1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---

E9

Description:

The Program Counter (register pair PC) is loaded with the contents of the IY register pair (Index Register IY). The next instruction is fetched from the location designated by the new contents of the PC.

M CYCLES: 2 T STATES: 8(4,4) 4 MHZ E.T.: 2.00

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1000H and the contents of the IY Register Pair are 4800H, after the execution of

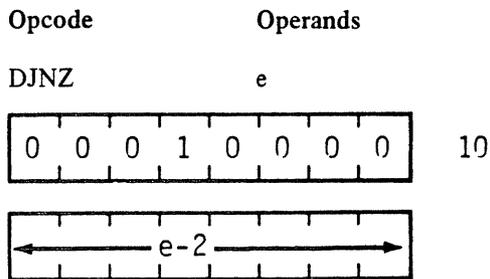
JP (IY)

the contents of the Program Counter will be 4800H.

DJNZ, e

Operation: —

Format:



Description:

The instruction is similar to the conditional jump instructions except that a register value is used to determine branching. The B register is decremented and if a non zero value remains, the value of the displacement e is added to the Program Counter (PC). The next instruction is fetched from the location designated by the new contents of the PC. The jump is measured from the address of the instruction opcode and has a range of -126 to +129 bytes. The assembler automatically adjusts for the twice incremented PC.

If the result of decrementing leaves B with a zero value, the next instruction to be executed is taken from the location following this instruction.

If B≠0:

M CYCLES: 3 T STATES: 13(5,3,5) 4 MHZ E.T.: 3.25

If B=0:

M CYCLES: 2 T STATES: 8(5,3) 4 MHZ E.T.: 2.00

Condition Bits Affected: None

Example:

A typical software routine is used to demonstrate the use of the DJNZ instruction. This routine moves a line from an input buffer (INBUF) to an output buffer (OUTBUF). It moves the bytes until it finds a CR, or until it has moved 80 bytes, whichever occurs first.

```

LD    B,80           ;Set up counter
LD    HL,Inbuf       ;Set up pointers
LD    DE,Outbuf

LOOP: LD    A,(HL)    ;Get next byte from
                    ;input buffer
LD    (DE),A         ;Store in output buffer
CP    00H            ;Is it a CR?
JR    Z,DONE         ;Yes finished

```

```

INC    HL           ;Increment pointers
INC    DE
DJNZ   LOOP         ;Loop back if 80
                    ;bytes have not
                    ;been moved

```

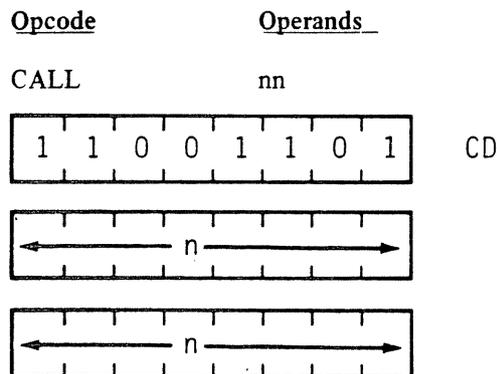
DONE:

CALL AND RETURN GROUP

CALL nn

Operation: $(SP-1) \leftarrow PC_H, (SP-2) \leftarrow PC_L, PC \leftarrow nn$

Format:



Note: The first of the two n operands in the assembled object code above is the least significant byte of a two-byte memory address.

Description:

After pushing the current contents of the Program Counter (PC) onto the top of the external memory stack, the operands nn are loaded into PC to point to the address in memory where the first opcode of a subroutine is to be fetched. (At the end of the subroutine, a RETURN instruction can be used to return to the original program flow by popping the top of the stack back into PC.) The push is accomplished by first decrementing the current contents of the Stack Pointer (register pair SP), loading the high-order byte of the PC contents into the memory address now pointed to by the SP; then decrementing SP again, and loading the low-order byte of the PC contents into the top of stack. Note: Because this is a 3-byte instruction, the Program Counter will have been incremented by 3 before the push is executed.

M CYCLES: 5 T STATES: 17(4,3,4,3,3) 4 MHZ E.T.: 4.25

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 1A47H, the contents of the Stack Pointer are 3002H, and memory locations have the contents:

Location	Contents
1A47H	CDH
1A48H	35H
1A49H	21H

then if an instruction fetch sequence begins, the three-byte instruction CD3521H will be fetched to the CPU for execution. The mnemonic equivalent of this is

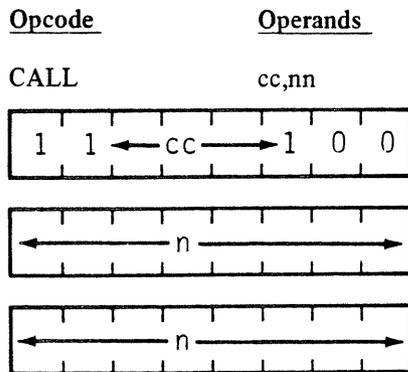
CALL 2135H

After the execution of this instruction, the contents of memory address 3001H will be 1AH, the contents of address 3000H will be 4AH, the contents of the Stack Pointer will be 3000H, and the contents of the Program Counter will be 2135H, pointing to the address of the first opcode of the subroutine now to be executed.

CALL cc, nn

Operation: IF cc TRUE: (SP-1) ← PC_H
(SP-2) ← PC_L, PC ← nn

Format:



Note: The first of the two n operands in the assembled object code above is the least significant byte of the two-byte memory address.

Description:

If condition cc is true, this instruction pushes the current contents of the Program Counter (PC) onto the top of the external memory stack, then loads the operands nn into PC to point to the address in memory where the first opcode of a subroutine is to be fetched. (At the end of the subroutine, a RETurn instruction can be used to return to the original program flow by popping the top of the stack back into PC.) If condition cc is false, the Program Counter is incremented as usual, and the program continues with the next sequential instruction. The stack push is accomplished by first decrementing the current contents of the Stack Pointer (SP), loading the high-order byte of the PC contents into the memory address now pointed to by SP; then decrementing SP again, and loading the low-order byte of the PC contents into the top of the stack. Note: Because this is a 3-byte instruction, the Program Counter will have been incremented by 3 before the push is executed. Condition cc is programmed as one of eight status which corresponds to condition bits in the Flag Register (register F). Those eight status are defined in the table below, which also specifies the corresponding cc bit fields in the assembled object code:

cc	Condition	Relevant Flag
000	NZ non zero	Z
001	Z zero	Z
010	NC non carry	C
011	C carry	C
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

If cc is true:

M CYCLES: 5 T STATES: 17(4,3,4,3,3) 4 MHZ E.T.: 4.25

If cc is false:

M CYCLES: 3 T STATES: 10(4,3,3) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If the C Flag in the F register is reset, the contents of the Program Counter are 1A47H, the contents of the Stack Pointer are 3002H, and memory locations have the contents:

Location	Contents
1A47H	D4H
1A48H	35H
1A49H	21H

then if an instruction fetch sequence begins, the three-byte instruction D43521H will be fetched to the CPU for execution. The mnemonic equivalent of this is

CALL NC, 2135H

After the execution of this instruction, the contents of memory address 3001H will be 1AH, the contents of address 3000H will be 4AH, the contents of the Stack Pointer will be 3000H, and the contents of the Program Counter will be 2135H, pointing to the address of the first opcode of the subroutine now to be executed.

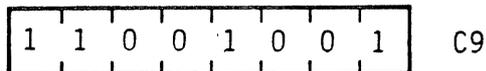
RET

Operation: $PC_L \leftarrow (SP), PC_H \leftarrow (SP+1)$

Format:

Opcode

RET



Description:

Control is returned to the original program flow by popping the previous contents of the Program Counter (PC) off the top of the external memory stack, where they were pushed by the CALL instruction. This is accomplished by first loading the low-order byte of the PC with the contents of the memory address pointed to by the Stack Pointer (SP), then incrementing the SP and loading the high-order byte of the PC with the contents of the memory address now pointed to by the SP. (The SP is now incremented a second time.) On the following machine cycle the CPU will fetch the next program opcode from the location in memory now pointed to by the PC.

M CYCLES: 3 T STATES: 10(4,3,3) 4 MHZ E.T.: 2.50

Condition Bits Affected: None

Example:

If the contents of the Program Counter are 3535H, the contents of the Stack Pointer are 2000H, the contents of memory location 2000H are B5H, and the contents of memory location 2001H are 18H, then after the execution of

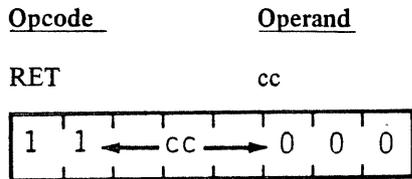
RET

the contents of the Stack Pointer will be 2002H and the contents of the Program Counter will be 18B5H, pointing to the address of the next program opcode to be fetched.

RET CC

Operation: IF cc TRUE: $PC_L \leftarrow (SP), PC_H \leftarrow (SP+1)$

Format:



Description:

If condition cc is true, control is returned to the original program flow by popping the previous contents of the Program Counter (PC) off the top of the external memory stack, where they were pushed by the CALL instruction. This is accomplished by first loading the low-order byte of the PC with the contents of the memory address pointed to by the Stack Pointer (SP), then incrementing the SP, and loading the high-order byte of the PC with the contents of the memory address now pointed to by the SP. (The SP is now incremented a second time.) On the following machine cycle the CPU will fetch the next program opcode from the location in memory now pointed to by the PC. If condition cc is false, the PC is simply incremented as usual, and the program continues with the next sequential instruction. Condition cc is programmed as one of eight status which correspond to condition bits in the Flag Register (register F). These eight status are defined in the table below, which also specifies the corresponding cc bit fields in the assembled object code.

<u>cc</u>	<u>Condition</u>	<u>Relevant Flag</u>
000	NZ non zero	Z
001	Z zero	Z
010	NC non carry	C
011	C carry	C
100	PO parity odd	P/V
101	PE parity even	P/V
110	P sign positive	S
111	M sign negative	S

If cc is true:

M CYCLES: 3 T STATES: 11(5,3,3) 4 MHZ E.T.: 2.75

If cc is false:

M CYCLES: 1 T STATES: 5 4 MHZ E.T.: 1.25

Condition Bits Affected: None

Example:

If the S flag in the F register is set, the contents of the Program Counter are 3535H, the contents of the Stack Pointer are 2000H, the contents of memory location 2000H are B5H, and the contents of memory location 2001H are 18H, then after the execution of

RET M

the contents of the Stack Pointer will be 2002H and the contents of the Program Counter will be 18B5H, pointing to the address of the next program opcode to be fetched.

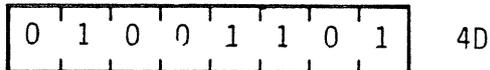
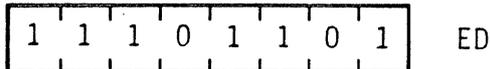
RETI

Operation: Return from interrupt

Format:

Opcode

RETI



Description:

This instruction is used at the end of an interrupt service routine to:

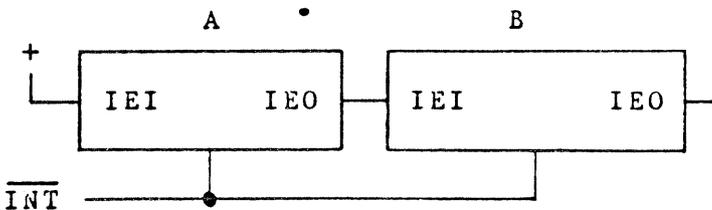
1. Restore the contents of the Program Counter (PC) (analogous to the RET instruction).
2. To signal an I/O device that the interrupt routine has been completed. The RETI instruction facilitates the nesting of interrupts allowing higher priority devices to suspend service of lower priority service routines. This instruction also resets the IFF1 and IFF2 flip flops.

M CYCLES: 4 T STATES: 14(4,4,3,3) 4 MHZ E.T.: 3.5 ϕ

Condition Bits Affected: None

Example:

Given: Two interrupting devices, A and B connected in a daisy chain configuration with A having a higher priority than B.



B generates an interrupt and is acknowledged. (The interrupt enable out, IEO, of B goes low, blocking any lower priority devices from interrupting while B is being serviced). Then A generates an interrupt, suspending service of B. (The IEO of A goes 'low' indicating that a higher priority device is being serviced.) The A routine is completed and a RETI is issued resetting the IEO of A, allowing the B routine to continue. A second RETI is issued on completion of the B routine and the IEO of B is reset (high) allowing lower priority devices interrupt access.

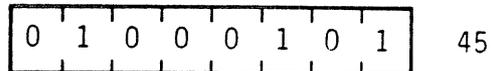
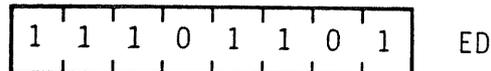
RETN

Operation: Return from non maskable interrupt

Format:

Opcode

RETN



Description:

Used at the end of a service routine for a non maskable interrupt, this instruction executes an unconditional return which functions identical to the RET instruction. That is, the previously stored contents of the Program Counter (PC) are popped off the top of the external memory stack; the low-order byte of PC is loaded with the contents of the memory location pointed to by the Stack Pointer (SP), SP is incremented, the high-order byte of PC is loaded with the contents of the memory location now pointed to by SP, and SP is incremented again. Control is now returned to the original program flow: on the following machine cycle the CPU will fetch the next opcode from the location in memory now pointed to by the PC. Also the state of IFF2 is copied back into IFF1 to the state it had prior to the acceptance of the NMI.

M CYCLES: 4 T STATES: 14(4,4,3,3) 4 MHZ E.T.: 3.5 ϕ

Condition Bits Affected: None

Example:

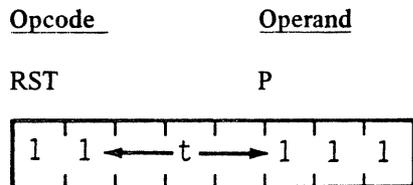
If the contents of the Stack Pointer are 1000H and the contents of the Program Counter are 1A45H when a non maskable interrupt (NMI) signal is received, the CPU will ignore the next instruction and will instead restart to memory address 0066H. That is, the current Program Counter contents of 1A45H will be pushed onto the external stack address of OFFFH and OFFEH, high order-byte first, and 0066H will be loaded onto the Program Counter. That address begins an interrupt service routine which ends with RETN instruction. Upon the execution of RETN, the former Program Counter contents are popped off the external memory stack, low-order first, resulting in a Stack Pointer contents again of 1000H. The program flow continues where it left off with an opcode fetch to address 1A45H.

RST p

Operation:

$(SP-1) \leftarrow PC_H, (SP-2) \leftarrow PC_L, PC_H \leftarrow O, PC_L \leftarrow P$

Format:



Description:

The current Program Counter (PC) contents are pushed onto the external memory stack, and the page zero memory location given by operand p is loaded into the PC. Program execution then begins with the opcode in the address now pointed to by PC. The push is performed by first decrementing the contents of the Stack Pointer (SP), loading the high-order byte of PC into the memory address now pointed to by SP, decrementing SP again, and loading the low-order byte of PC into the address now pointed to by SP. The ReStart instruction allows for a jump to one of eight addresses as shown in the table below. The operand p is assembled into the object code using the corresponding T state. Note: Since all addresses are in page zero of memory, the high order byte of PC is loaded with 00H. The number selected from the “p” column of the table is loaded into the low-order byte of PC.

<u>P</u>	<u>t</u>
00H	000
08H	001
10H	010
18H	011
20H	100
28H	101
30H	110
38H	111

M CYCLES: 3 T STATES: 11(5,3,3) 4 MHZ E.T.: 2.75

Example:

If the contents of the Program Counter are 15B3H, after the execution of

RST 18H (Object code 1101111)

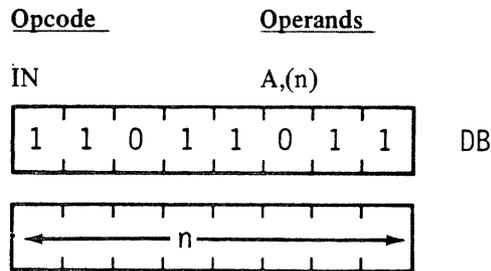
the PC will contain 0018H, as the address of the next opcode to be fetched.

INPUT AND OUTPUT GROUP

IN A, (n)

Operation: $A \leftarrow (n)$

Format:



Description:

The operand n is placed on the bottom half (A_0 through A_7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator also appear on the top half (A_8 through A_{15}) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written into the Accumulator (register A) in the CPU.

M CYCLES: 3 T STATES: 11(4,3,4) 4 MHZ E.T.: 2.75

Condition Bits Affected: None

Example:

If the contents of the Accumulator are 23H and the byte 7BH is available at the peripheral device mapped to I/O port address 01H, then after the execution of

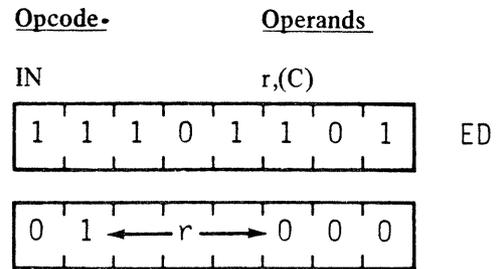
IN A, (01H)

the Accumulator will contain 7BH.

IN r, (C)

Operation: $r \leftarrow (C)$

Format:



Description:

The contents of register C are placed on the bottom half (A_0 through A_7) of the address bus to select the I/O device at one of 256 possible ports. The contents of Register B are placed on the top half (A_8 through A_{15}) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written into register r in the CPU. Register r identifies any of the CPU registers shown in the following table, which also shows the corresponding 3-bit "r" field for each. The flags will be affected, checking the input data.

Reg.	r
B	000
C	001
D	010
E	011
H	100
L	101
A	111

M CYCLES: 3 T STATES: 12(4,4,4) 4 MHZ E.T.: 3.00

Condition Bits Affected:

- S: Set if input data is negative; reset otherwise
- Z: Set if input data is zero; reset otherwise
- H: Reset
- P/V: Set if parity is even; reset otherwise
- N: Reset
- C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

IN D, (C)

register D will contain 7BH

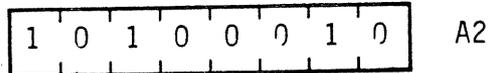
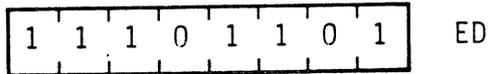
INI

Operation: (HL) ← (C), B ← B-1, HL ← HL + 1

Format:

Opcode

INI



Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are then placed on the address bus and the input byte is written into the corresponding location of memory. Finally the byte counter is decremented and register pair HL is incremented.

M CYCLES: 4 T STATES: 16(4,5,3,4) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Unknown
- Z: Set if B-1=0; reset otherwise
- H: Unknown
- P/V: Unknown
- N: Set
- C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

INI

memory location 1000H will contain 7BH, the HL register pair will contain 1001H, and register B will contain 0FH.

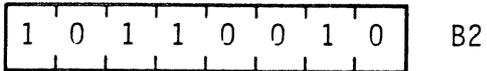
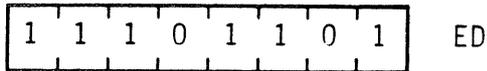
INIR

Operation: (HL) ← (C), B ← B-1, HL ← HL + 1

Format:

Opcode

INIR



Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Then register pair HL is incremented, the byte counter is decremented. If decrementing causes B to go to zero, the instruction is terminated. If B is not zero, the PC is decremented by two and the instruction repeated. Note that if B is set to zero prior to instruction execution, 256 bytes of data will be input. Also interrupts will be recognized after each data transfer.

If B≠0:

M CYCLES: 5 T STATES: 21(4,5,3,4,5) 4 MHZ E.T.: 5.25

If B=0:

M CYCLES: 4 T STATES: 16(4,5,3,4) 4 MHZ E.T.: 4.00

Condition Bits Affected:

S: Unknown
Z: Set
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and the following sequence of bytes are available at the peripheral device mapped to I/O port of address 07H:

51H
A9H
03H

then after the execution of

INIR

the HL register pair will contain 1003H, register B will contain zero, and memory locations will have contents as follows:

<u>Location</u>	<u>Contents</u>
1000H	51H
1001H	A9H
1002H	03H

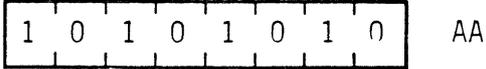
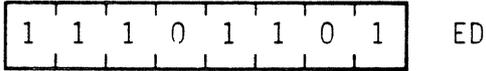
IND

Operation: (HL) ← (C), B ← B-1, HL ← HL-1

Format:

Opcode

IND



Description:-

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Finally the byte counter and register pair HL are decremented.

M CYCLES: 4 T STATES: 16(4,5,3,4) 4 MHZ E.T.: 4.00

Condition Bits Affected:

S: Unknown
Z: Set if B-1=0; reset otherwise
H: Unknown
P/V: Unknown
N: Set
C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the byte 7BH is available at the peripheral device mapped to I/O port address 07H, then after the execution of

IND

memory location 1000H will contain 7BH, the HL register pair will contain 0FFFH, and register B will contain 0FH.

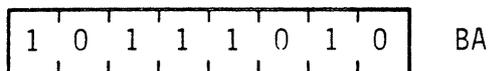
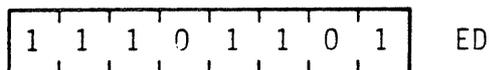
INDR

Operation: (HL) ← (C), B ← B-1, HL ← HL-1

Format:

Opcode

INDR



Description:

The contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B is used as a byte counter, and its contents are placed on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the CPU. The contents of the HL register pair are placed on the address bus and the input byte is written into the corresponding location of memory. Then HL and the byte counter are decremented. If decrementing causes B to go to zero, the instruction is terminated. If B is not zero, the PC is decremented by two and the instruction repeated. Note that if B is set to zero prior to instruction execution, 256 bytes of data will be input. Also interrupts will be recognized after each data transfer.

If B≠0:

M CYCLES: 5 T STATES: 21(4,5,3,4,5) 4 MHZ E.T.: 5.25

If B=0:

M CYCLES: 4 T STATES: 16(4,5,3,4) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Unknown
- Z: Set
- H: Unknown
- P/V: Unknown
- N: Set
- C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and the following sequence of bytes are available at the peripheral device mapped to I/O port address 07H:

51H
A9H
03H

then after the execution of

INDR

the HL register pair will contain 0FFDH, register B will contain zero, and memory locations will have contents as follows:

Location	Contents
0FFEH	03H
0FFFH	A9H
1000H	51H

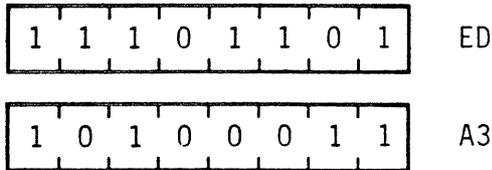
OUTI

Operation: (C) ← (HL), B ← B-1, HL ← HL + 1

Format:

Opcode

OUTI



Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus. The byte to be output is placed on the data bus and written into selected peripheral device. Finally the register pair HL is incremented.

M CYCLES: 4 T STATES: 16(4,5,3,4) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Unknown
- Z: Set if B-1=0; reset otherwise
- H: Unknown
- P/V: Unknown
- N: Set
- C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the contents of memory address 1000H are 59H, then after the execution of

OUTI

register B will contain 0FH, the HL register pair will contain 1001H, and the byte 59H will have been written to the peripheral device mapped to I/O port address 07H.

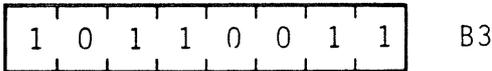
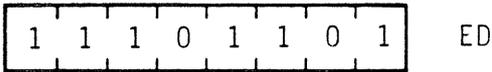
OTIR

Operation: (C) ← (HL), B ← B-1, HL ← HL + 1

Format:

Opcode

OTIR



Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Then register pair HL is incremented. If the decremented B register is not zero, the Program Counter (PC) is decremented by 2 and the instruction is repeated. If B has gone to zero, the instruction is terminated. Note that if B is set to zero prior to instruction execution, the instruction will output 256 bytes of data. Also, interrupts will be recognized after each data transfer.

If B ≠ 0:

M CYCLES: 5 T STATES: 21(4,5,3,4,5) 4 MHZ E.T.: 5.25

If B = 0:

M CYCLES: 4 T STATES: 16(4,5,3,4) 4 MYZ E.T.: 4.00

Condition Bits Affected:

- S: Unknown
- Z: Set
- H: Unknown
- P/V: Unknown
- N: Set
- C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and memory locations have the following contents:

Location	Contents
1000H	51H
1001H	A9H
1002H	03H

then after the execution of

OTIR

the HL register pair will contain 1003H, register B will contain zero, and a group of bytes will have been written to the peripheral device mapped to I/O port address 07H in the following sequence:

- 51H
- A9H
- 03H

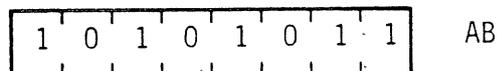
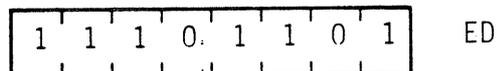
OUTD

Operation: (C) ← (HL), B ← B-1, HL ← HL+1

Format:

Opcode

OUTD



Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Finally the register pair HL is incremented.

M CYCLES: 4 T STATES: 16(4,5,3,4) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Unknown
- Z: Set if B-1=0; reset otherwise
- H: Unknown
- P/V: Unknown
- N: Set
- C: Not affected

Example:

If the contents of register C are 07H, the contents of register B are 10H, the contents of the HL register pair are 1000H, and the contents of memory location 1000H are 59H, after the execution of

OUTD

register B will contain 0FH, the HL register pair will contain 0FFFH, and the byte 59H will have been written to the peripheral device mapped to I/O port address 07H.

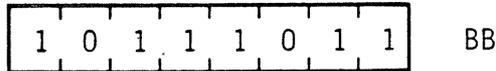
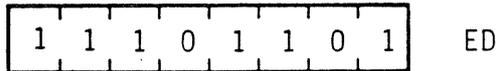
OTDR

Operation: (C) ← (HL), B ← B-1, HL ← HL-1

Format:

Opcode

OTDR



Description:

The contents of the HL register pair are placed on the address bus to select a location in memory. The byte contained in this memory location is temporarily stored in the CPU. Then, after the byte counter (B) is decremented, the contents of register C are placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. Register B may be used as a byte counter, and its decremented value is placed on the top half (A8 through A15) of the address bus at this time. Next the byte to be output is placed on the data bus and written into the selected peripheral device. Then register pair HL is decremented and if the decremented B register is not zero, the Program Counter (PC) is decremented by 2 and the instruction is repeated. If B has gone to zero, the instruction is terminated. Note that if B is set to zero prior to instruction execution, the instruction will output 256 byte of data. Also, interrupts will be recognized after each data transfer.

If B≠0:

M CYCLES: 5 T STATES: 21(4,5,3,4,5) 4 MHZ E.T.: 5.25

If B=0:

M CYCLES: 4 T STATES: 16(4,5,3,4) 4 MHZ E.T.: 4.00

Condition Bits Affected:

- S: Unknown
- Z: Set
- H: Unknown
- P/V: Unknown
- N: Set
- C Not affected

Example:

If the contents of register C are 07H, the contents of register B are 03H, the contents of the HL register pair are 1000H, and memory locations have the following contents:

Location Contents

0FFEH	51H
0FFFH	A9H
1000H	03H

then after the execution of

OTDR

the HL register pair will contain 0FFDH, register B will contain zero, and a group of bytes will have been written to the peripheral device mapped to I/O port address 07H in the following sequence:

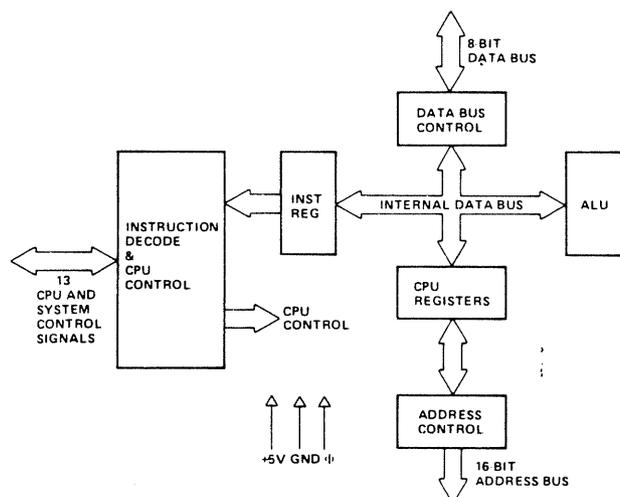
- 03H
- A9H
- 51H

Z-80 Hardware Configuration

This section gives information about the actual Z80 chip.

Z-80 CPU ARCHITECTURE

A block diagram of the internal architecture of the Z-80 CPU is shown in Figure 1. The diagram shows all of the major elements in the CPU and it should be referred to throughout the following description.



Z-80 CPU BLOCK DIAGRAM
FIGURE 1

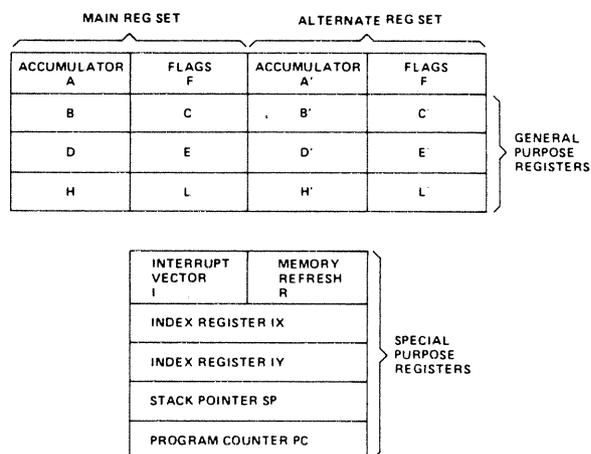
CPU REGISTERS

The Z-80 CPU contains 208 bits of R/W memory that are accessible to the programmer. Figure 2 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 as 16-bit registers. There are also two sets of accumulator and flag registers.

Special Purpose Registers

- 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.



Z-80 CPU REGISTER CONFIGURATION
FIGURE 2

- 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

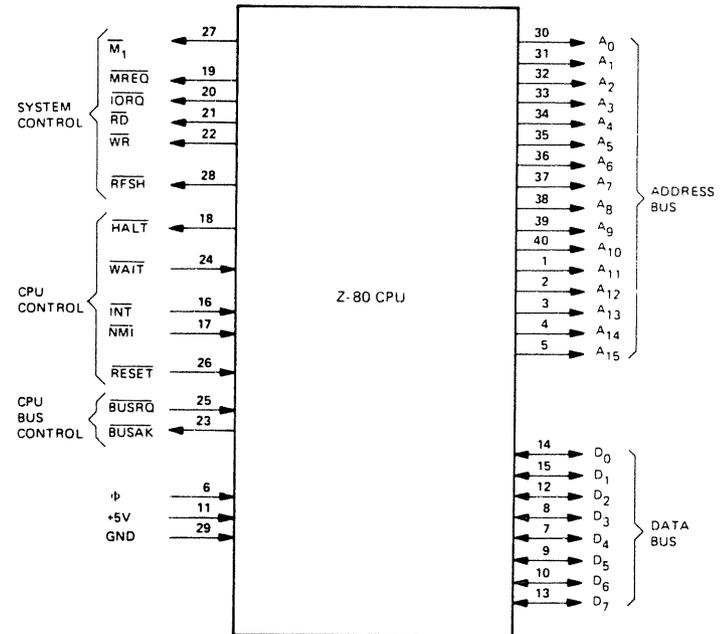
INSTRUCTION REGISTER AND CPU CONTROL

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 3 and the function of each is described below.



Z-80 PIN CONFIGURATION
FIGURE 3

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 IORQ to indicate an interrupt acknowledge cycle.

\overline{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{IORQ} (Input/Output Request)	Tri-state output, active low. The \overline{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{IORQ} signal is also generated with an \overline{MI} 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{RD} (Memory Read)	Tri-state output, active low. \overline{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{WR} (Memory Write)	Tri-state output, active low. \overline{WR} indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.
\overline{RFSH} (Refresh)	Output, active low. \overline{RFSH} indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current \overline{MREQ} signal should be used to do a refresh read to all dynamic memories.
\overline{HALT} (Halt state)	Output, active low. \overline{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{WAIT} (Wait)	Input, active low. \overline{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{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{BUSRQ} signal is not active. When the CPU accepts the interrupt, an acknowledge signal (\overline{IORQ} during M_1 time) is sent out at the beginning of the next instruction cycle.

\overline{NMI}
(Non Maskable Interrupt)

Input, negative edge triggered. The non maskable interrupt request line has a higher priority than \overline{INT} and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. \overline{NMI} automatically forces the Z-80 CPU to restart to location $0066H$. 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{BUSRQ} will override a \overline{NMI} .

\overline{RESET}

Input, active low. \overline{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 = $00H$
- 3) Set Register R = $00H$
- 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{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{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{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 33 Ω 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

INTRODUCTION TO INSTRUCTION TYPES

The load instructions move data internally between CPU registers or between CPU registers and external memory. All of these instructions must specify a source location from which the data is to be moved and a destination location. The source location is not altered by a load instruction. Examples of load group instructions include moves between any of the general purpose registers such as move the data to Register B from Register C. This group also includes load immediate to any CPU register or to any external memory location. Other types of load instructions allow transfer between CPU registers and memory locations. The exchange instructions can trade the contents of two registers.

A unique set of block transfer instructions is provided in the Z-80. With a single instruction a block of memory of any size can be moved to any other location in memory. This set of block moves is extremely valuable when large strings of data must be processed. The Z-80 block search instructions are also valuable for this type of processing. With a single instruction, a block of external memory of any desired length can be searched for any 8-bit character. Once the character is found or the end of the block is reached, the instruction automatically terminates. Both the block transfer and the block search instructions can be interrupted during their execution so as to not occupy the CPU for long periods of time.

The arithmetic and logical instructions operate on data stored in the accumulator and other general purpose CPU registers or external memory locations. The results of the operations are placed in the accumulator and the appropriate flags are set according to the result of the operation. An example of an arithmetic operation is adding the accumulator to the contents of an external memory location. The results of the addition are placed in the accumulator. This group also includes 16-bit addition and subtraction between 16-bit CPU registers.

The rotate and shift group allows any register or any memory location to be rotated right or left with or without carry either arithmetic or logical. Also, a digit in the accumulator can be rotated right or left with two digits in any memory location.

The bit manipulation instructions allow any bit in the accumulator, any general purpose register or any external memory location to be set, reset or tested with a single instruction. For example, the most significant bit of register H can be reset. This group is especially useful in control applications and for controlling software flags in general purpose programming.

The jump, call and return instructions are used to transfer between various locations in the user's program. This group uses several different techniques for obtaining the new program counter address from specific external memory locations. A unique type of call is the restart instruction. This instruction actually contains the new address as a part of the 8-bit OP code. This is possible since only 8 separate addresses located in page zero of the external memory may be specified. Program jumps may also be achieved by loading register HL, IX or IY directly into the PC, thus allowing the jump address to be a complex function of the routine being executed.

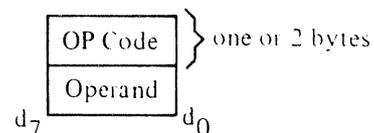
The input/output group of instructions in the Z-80 allow for a wide range of transfers between external memory locations or the general purpose CPU registers, and the external I/O devices. In each case, the port number is provided on the lower 8 bits of the address bus during any I/O transaction. One instruction allows this port number to be specified by the second byte of the instruction while other Z-80 instructions allow it to be specified as the content of the C register. One major advantage of using the C register as a pointer to the I/O device is that it allows different I/O ports to share common software driver routines. This is not possible when the address is part of the OP code if the routines are stored in ROM. Another feature of these input instructions is that they set the flag register automatically so that additional operations are not required to determine the state of the input data (for example its parity). The Z-80 CPU includes single instructions that can move blocks of data (up to 256 bytes) automatically to or from any I/O port directly to any memory location. In conjunction with the dual set of general purpose registers, these instructions provide for fast I/O block transfer rates. The value of this I/O instruction set is demonstrated by the fact that the Z-80 CPU can provide all required floppy disk formatting (i.e., the CPU provides the preamble, address, data and enables the CRC codes) on double density floppy disk drives on an interrupt driven basis.

Finally, the basic CPU control instructions allow various options and modes. This group includes instructions such as setting or resetting the interrupt enable flip flop or setting the mode of interrupt response.

ADDRESSING MODES

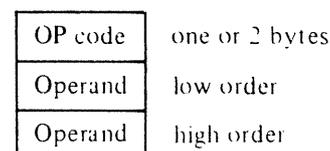
Most of the Z-80 instructions operate on data stored in internal CPU registers, external memory or in the I/O ports. Addressing refers to how the address of this data is generated in each instruction. This section gives a brief summary of the types of addressing used in the Z-80 while subsequent sections detail the type of addressing available for each instruction group.

Immediate. In this mode of addressing the byte following the OP code in memory contains the actual operand.



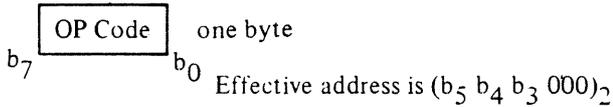
Examples of this type of instruction would be to load the accumulator with a constant, where the constant is the byte immediately following the OP code.

Immediate Extended. This mode is merely an extension of immediate addressing in that the two bytes following the OP codes are the operand.

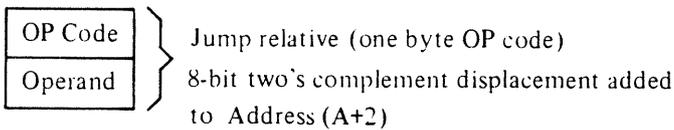


Examples of this type of instruction would be to load the HL register pair (16-bit register) with 16 bits (2 bytes) of data.

Modified Page Zero Addressing. The Z-80 has a special single byte CALL instruction to any of 8 locations in page zero of memory. This instruction (which is referred to as a restart) sets the PC to an effective address in page zero. The value of this instruction is that it allows a single byte to specify a complete 16-bit address where commonly called subroutines are located, thus saving memory space.

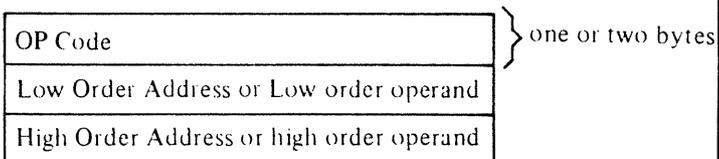


Relative Addressing. Relative addressing uses one byte of data following the OP code to specify a displacement from the existing program to which a program jump can occur. This displacement is a signed two's complement number that is added to the address of the OP code of the following instruction.



The value of relative addressing is that it allows jumps to nearby locations while only requiring two bytes of memory space. For most programs, relative jumps are by far the most prevalent type of jump due to the proximity of related program segments. Thus, these instructions can significantly reduce memory space requirements. The signed displacement can range between +127 and -128 from A + 2. This allows for a total displacement of +129 to -126 from the jump relative OP code address. Another major advantage is that it allows for relocatable code.

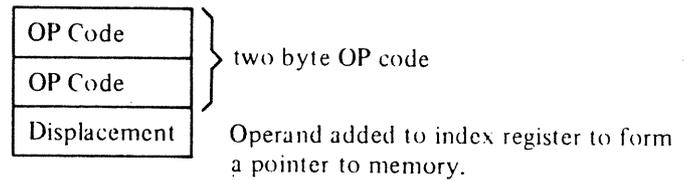
Extended Addressing. Extended Addressing provides for two bytes (16 bits) of address to be included in the instruction. This data can be an address to which a program can jump or it can be an address where an operand is located.



Extended addressing is required for a program to jump from any location in memory to any other location, or load and store data in any memory location.

When extended addressing is used to specify the source or destination address of an operand, the notation (nn) will be used to indicate the content of memory at nn, where nn is the 16-bit address specified in the instruction. This means that the two bytes of address nn are used as a pointer to a memory location. The use of the parentheses always means that the value enclosed within them is used as a pointer to a memory location. For example, (1200) refers to the contents of memory at location 1200.

Indexed Addressing. In this type of addressing, the byte of data following the OP code contains a displacement which is added to one of the two index registers (the OP code specifies which index register is used) to form a pointer to memory. The contents of the index register are not altered by this operation.



An example of an indexed instruction would be to load the contents of the memory location (Index Register + Displacement) into the accumulator. The displacement is a signed two's complement number. Indexed addressing greatly simplifies programs using tables of data since the index register can point to the start of any table. Two index registers are provided since very often operations require two or more tables. Indexed addressing also allows for relocatable code.

The two index registers in the Z-80 are referred to as IX and IY. To indicate indexed addressing the notation:

(IX+d) or (IY+d)

is used. Here d is the displacement specified after the OP code. The parentheses indicate that this value is used as a pointer to external memory.

Register Addressing. Many of the Z-80 OP codes contain bits of information that specify which CPU register is to be used for an operation. An example of register addressing would be to load the data in register B into register C.

Implied Addressing. Implied addressing refers to operations where the OP code automatically implies one or more CPU registers as containing the operands. An example is this set of arithmetic operations where the accumulator is always implied to be the destination of the results.

Register Indirect Addressing. This type of addressing specifies a 16-bit CPU register pair (such as HL) to be used as a pointer to any location in memory. This type of instruction is very powerful and it is used in a wide range of applications.



An example of this type of instruction would be to load the accumulator with the data in the memory location pointed to by the HL register contents. Indexed addressing is actually a form of register indirect addressing except that a displacement is added with indexed addressing. Register indirect addressing allows for very powerful but simple to implement memory accesses. The block move and search commands in the Z-80 are extensions of this type of addressing where automatic register incrementing, decrementing and comparing has been added. The notation for indicating register indirect addressing is to put parentheses around the name of the register that is to be used as the pointer. For example, the symbol

(HL)

specifies that the contents of the HL register are to be used as a pointer to a memory location. Often register indirect addressing is used to specify 16-bit operands. In this case, the register contents point to the low order portion of the operand while the register contents are automatically incremented to obtain the upper portion of the operand.

Bit Addressing. The Z-80 contains a large number of bit set, reset and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation through one of three previous addressing modes (register, register indirect and indexed) while three bits in the OP code specify which of the eight bits is to be manipulated.

ADDRESSING MODE COMBINATIONS

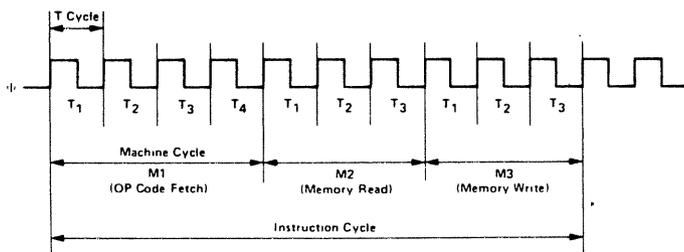
Many instructions include more than one operand (such as arithmetic instructions or loads). In these cases, two types of addressing may be employed. For example, load can use immediate addressing to specify the source and register indirect or indexed addressing to specify the destination.

CPU TIMING

The Z-80 CPU executes instructions by stepping through a very precise set of a few basic operations. These include:

- Memory read or write
- I/O device read or write
- Interrupt acknowledge

All instructions are merely a series of these basic operations. Each of these basic operations can take from three to six clock periods to complete or they can be lengthened to synchronize the CPU to the speed of external devices. The basic clock periods are referred to as T cycles and the basic operations are referred to as M (for machine) cycles. Figure 4 illustrates how a typical instruction will be merely a series of specific M and T cycles. Notice that this instruction consists of three machine cycles (M1, M2 and M3). The first machine cycle of any instruction is a fetch cycle which is four, five or six T cycles long (unless lengthened by the wait signal which will be fully described in the next section). The fetch cycle (M1) is used to fetch the OP code of the next instruction to be executed. Subsequent machine cycles move data between the CPU and memory or I/O devices and they may have anywhere from three to five T cycles (again they may be lengthened by wait states to synchronize the external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles. In section 10, the exact timing for each instruction is specified.



**BASIC CPU TIMING EXAMPLE
FIGURE 4**

NUMERIC LIST OF INSTRUCTION SET

Z-80 CROSS ASSEMBLER VERSION 1.06 OF 06/18/76

07/09/76 10:20:50

OPCODE LISTING

LOC	OBJ CODE	STMT	SOURCE STATEMENT	LOC	OBJ CODE	STMT	SOURCE STATEMENT
0000	00	1	NOP	0065	47	72	LD B,A
0001	018405	2	LD BC,NN	0066	48	73	LD C,B
0004	02	3	LD (BC),A	0067	49	74	LD C,C
0005	03	4	INC BC	0068	4A	75	LD C,D
0006	04	5	INC B	0069	4B	76	LD C,E
0007	05	6	DEC B	006A	4C	77	LD C,H
0008	0620	7	LD B,N	006B	4D	78	LD C,L
000A	07	8	RLCA	006C	4E	79	LDC,(HL)
000B	08	9	EX AF,AF'	006D	4F	80	LD C,A
000C	09	10	ADD HL,BC	006E	50	81	LD D,B
000D	0A	11	LD A,(BC)	006F	51	82	LD D,C
000E	0B	12	DEC BC	0070	52	83	LD D,D
000F	0C	13	INC C	0071	53	84	LD D,E
0010	0D	14	DEC C	0072	54	85	LD D,H
0011	0E20	15	LD C,N	0073	55	86	LD D,L
0013	0F	16	RRCA	0074	56	87	LD D,(HL)
0014	102E	17	DJNZ DIS	0075	57	88	LD D,A
0016	118405	18	LD DE,NN	0076	58	89	LD E,B
0019	12	19	LD (DE),A	0077	59	90	LD E,C
001A	13	20	INC DE	0078	5A	91	LD E,D
001B	14	21	INC D	0079	5B	92	LD E,E
001C	15	22	DEC D	007A	5C	93	LD E,H
001D	1620	23	LD D,N	007B	5D	94	LD E,L
001F	17	24	RLA	007C	5E	95	LD E,(HL)
0020	182E	25	JR DIS	007D	5F	96	LD E,A
0022	19	26	ADD HL,DE	007E	60	97	LD H,B
0023	1A	27	LD A,(DE)	007F	61	98	LD H,C
0024	1B	28	DEC DE	0080	62	99	LD H,D
0025	1C	29	INC E	0081	63	100	LD H,E
0026	1D	30	DEC E	0082	64	101	LD H,H
0027	1E20	31	LD E,N	0083	65	102	LD H,L
0029	1F	32	RRA	0084	66	103	LD H,(HL)
002A	202E	33	JR NZ,DIS	0085	67	104	LD H,A
002C	218405	34	LD HL,NN	0086	68	105	LD L,B
002F	228405	35	LD (NN),HL	0087	69	106	LD L,C
0032	23	36	INC HL	0088	6A	107	LD L,D
0033	24	37	INC H	0089	6B	108	LD L,E
0034	25	38	DEC H	008A	6C	109	LD L,H
0035	2620	39	LD H,N	008B	6D	110	LD L,L
0037	27	40	DAA	008C	6E	111	LD L,(HL)
0038	282E	41	JR Z,DIS	008D	6F	112	LD L,A
003A	29	42	ADD HL,HL	008E	70	113	LD (HL),B
003B	2A8405	43	LD HL,(NN)	008F	71	114	LD (HL),C
003E	2B	44	DEC HL	0090	72	115	LD (HL),D
003F	2C	45	INC L	0091	73	116	LD (HL),E
0040	2D	46	DEC L	0092	74	117	LD (HL),H
0041	2E20	47	LD L,N	0093	75	118	LD (HL),L
0043	2F	48	CPL	0094	76	119	HALT
0044	302E	49	JR NC,DIS	0095	77	120	LD (HL),A
0046	318405	50	LD SP,NN	0096	78	121	LD A,B
0049	328405	51	LD (NN),A	0097	79	122	LD A,C
004C	33	52	INC SP	0098	7A	123	LD A,D
004D	34	53	INC (HL)	0099	7B	124	LD A,E
004E	35	54	DEC (HL)	009A	7C	125	LD A,H
004F	3620	55	LD (HL),N	009B	7D	126	LD A,L
0051	37	56	SCF	009C	7E	127	LD A,(HL)
0052	382E	57	JR C,DIS	009D	7F	128	LD A,A
0054	39	58	ADD HL,SP	009E	80	129	ADD A,B
0055	3A8405	59	LD A,(NN)	009F	81	130	ADD A,C
0058	3B	60	DEC SP	00A0	82	131	ADD A,D
0059	3C	61	INC A	00A1	83	132	ADD A,E
005A	3D	62	DEC A	00A2	84	133	ADD A,H
005B	3E20	63	LD A,N	00A3	85	134	ADD A,L
005D	3F	64	CCF	00A4	86	135	ADD A,(HL)
005E	40	65	LD B,B	00A5	87	136	ADD A,A
005F	41	66	LD B,C	00A6	88	137	ADC A,B
0060	42	67	LD B,D	00A7	89	138	ADC A,C
0061	43	68	LD B,E	00A8	8A	139	ADC A,D
0062	44	69	LD B,H,NN	00A9	8B	140	ADC A,E
0063	45	70	LD B,L	00AA	8C	141	ADC A,H
0064	46	71	LD B,(HL)	00AB	8D	142	ADC A,L

Z-80 CROSS ASSEMBLER VERSION 1.06 OF 06/18/76

07/09/76 10:20:50

OPCODE LISTING

LOC	OBJ CODE	STMT	SOURCE STATEMENT	LOC	OBJ CODE	STMT	SOURCE STATEMENT
00AC	8E	143	ADC A,(HL)	010B	DA8405	218	JP C,NN
00AD	8F	144	ADC A,A	010E	DB20	219	IN A,N
00AE	90	145	SUB B	0110	DC8405	220	CALL C,NN
00AF	91	146	SUB C	0113	DE20	221	SBC A,N
00B0	92	147	SUB D	0115	DF	222	RST 18H
00B1	93	148	SUB E	0116	E0	223	RET PO
00B2	94	149	SUB H	0117	E1	224	POP HL
00B3	95	150	SUB L	0118	E28405	225	JP PO,NN
00B4	96	151	SUB (HL)	011B	E3	226	EX (SP),HL
00B5	97	152	SUB A	011C	E48405	227	CALL PO,NN
00B6	98	153	SBC A,B	011F	E5	228	PUSH HL
00B7	99	154	SBC A,C	0120	E620	229	AND N
00B8	9A	155	SBC A,D	0122	E7	230	RST 20H
00B9	9B	156	SBC A,E	0123	E8	231	RET PE
00BA	9C	157	SBC A,H	0124	E9	232	JP (HL)
00BB	9D	158	SBC A,L	0125	EA8405	233	JP PE,NN
00BC	9E	159	SBC A,(HL)	0128	EB	234	EX DE,HL
00BD	9F	160	SBC A,A	0129	EC8405	235	CALL PE,NN
00BE	A0	161	AND B	012C	EE20	236	XOR N
00BF	A1	162	AND C	012E	EF	237	RST 28H
00C0	A2	163	AND D	012F	F0	238	RET P
00C1	A3	164	AND E	0130	F1	239	POP AF
00C2	A4	165	AND H	0131	F28405	240	JP P,NN
00C3	A5	166	AND L	0134	F3	241	DI
00C4	A6	167	AND (HL)	0135	F48405	242	CALL P,NN
00C5	A7	168	AND A	0138	F5	243	PUSH AF
00C6	A8	169	XOR B	0139	F620	244	OR N
00C7	A9	170	XOR C	013B	F7	245	RST 30H
00C8	AA	171	XOR D	013C	F8	246	RET M
00C9	AB	172	XOR E	013D	F9	247	LD SP,HL
00CA	AC	173	XOR H	013E	FA8405	248	JP M,NN
00CB	AD	174	XOR L	0141	FB	249	EI
00CC	AE	175	XOR (HL)	0142	FC8405	250	CALL M,NN
00CD	AF	176	XOR A	0145	FE20	251	CP N
00CE	B0	177	OR B	0147	FF	252	RST 38H
00CF	B1	178	OR C	0148	CB00	253	RLC B
00D0	B2	179	OR D	014A	CB01	254	RLC C
00D1	B3	180	OR E	014C	CB02	255	RLC D
00D2	B4	181	OR H	014E	CB03	256	RLC E
00D3	B5	182	OR L	0150	CB04	257	RLC H
00D4	B6	183	OR (HL)	0152	CB05	258	RLC L
00D5	B7	184	OR A	0154	CB06	259	RLC (HL)
00D6	B8	185	CP B	0156	CB07	260	RLC A
00D7	B9	186	CP C	0158	CB08	261	RRC B
00D8	BA	187	CP D	015A	CB09	262	RRC C
00D9	BB	188	CP E	015C	CB0A	263	RRC D
00DA	BC	189	CP H	015E	CB0B	264	RRC E
00DB	BD	190	CP L	0160	CB0C	265	RRC H
00DC	BE	191	CP (HL)	0162	CB0D	266	RRC L
00DD	BF	192	CP A	0164	CB0E	267	RRC (HL)
00DE	C0	193	RET NZ	0166	CB0F	268	RRC A
00DF	C1	194	POP BC	0168	CB10	269	RL B
00E0	C28405	195	JP NZ, NN	016A	CB11	270	RL C
00E3	C38405	196	JP NN	016C	CB12	271	RL D
00E6	C48405	197	CALL NZ,NN	016E	CB13	272	RL E
00E9	C5	198	PUSH BC	0170	CB14	273	RL H
00EA	C620	199	ADD A,N	0172	CB15	274	RL L
00EC	C7	200	RST 0	0174	CB16	275	RL (HL)
00ED	C8	201	RET Z	0176	CB17	276	RL A
00EE	C9	202	RET	0178	CB18	277	RR B
00EF	CA8405	203	JP Z,NN	017A	CB19	278	RR C
00F2	CC8405	204	CALL Z,NN	017C	CB1A	279	RR D
00F5	CD8405	205	CALL NN	017E	CB1B	280	RR E
00F8	CE20	206	ADC A,N	0180	CB1C	281	RR H
00FA	CF	207	RST 8	0182	CB1D	282	RR L
00FB	D0	208	RET NC	0184	CB1E	283	RR (HL)
00FC	D1	209	POP DE	0186	CB1F	284	RR A
00FD	D28405	210	JP NC,NN	0188	CB20	285	SLA B
0100	D320	211	OUT N,A	018A	CB21	286	SLA C
0102	D48405	212	CALL NC,NN	018C	CB22	287	SLA D
0105	D5	213	PUSH DE	018E	CB23	288	SLA E
0106	D620	214	SUB N	0190	CB24	289	SLA H
0108	D7	215	RST 10H	0192	CB25	290	SLA L
0109	D8	216	RET C	0194	CB26	291	SLA (HL)
010A	D9	217	EXX	0196	CB27	292	SLA A

07/09/76 10:20:50

OPCODE LISTING

LOC	OBJ CODE	STMT SOURCE STATEMENT	LOC	OBJ CODE	STMT SOURCE STATEMENT
0198	CB28	293 SRA B	0230	CB7C	369 BIT 7,H
019A	CB29	294 SRA C	0232	CB7D	370 BIT 7,L
019C	CB2A	295 SRA D	0234	CB7E	371 BIT 7,(HL)
019E	CB2B	296 SRA E	0236	CB7F	372 BIT 7,A
01A0	CB2C	297 SRA H	0238	CB80	373 RES 0,B
01A2	CB2D	298 SRA L	023A	CB81	374 RES 0,C
01A4	CB2E	299 SRA (HL)	023C	CB82	375 RES 0,D
01A6	CB2F	300 SRA A	023E	CB83	376 RES 0,E
01A8	CB38	301 SRL B	0240	CB84	377 RES 0,H
01AA	CB39	302 SRL C	0242	CB85	378 RES 0,L
01AC	CB3A	303 SRL D	0244	CB86	379 RES 0,(HL)
01AE	CB3B	304 SRL E	0246	CB87	380 RES 0,A
01B0	CB3C	305 SRL H	0248	CB88	381 RES 1,B
01B2	CB3D	306 SRL L	024A	CB89	382 RES 1,C
01B4	CB3E	307 SRL (HL)	024C	CB8A	383 RES 1,D
01B6	CB3F	308 SRL A	024E	CB8B	384 RES 1,E
01B8	CB40	309 BIT 0,B	0250	CB8C	385 RES 1,H
01BA	CB41	310 BIT 0,C	0252	CB8D	386 RES 1,L
01BC	CB42	311 BIT 0,D	0254	CB8E	387 RES 1,(HL)
01BE	CB43	312 BIT 0,E	0256	CB8F	388 RES 1,A
01C0	CB44	313 BIT 0,H	0258	CB90	389 RES 2,B
01C2	CB45	314 BIT 0,L	025A	CB91	390 RES 2,C
01C4	CB46	315 BIT 0,(HL)	025C	CB92	391 RES 2,D
01C6	CB47	316 BIT 0,A	025E	CB93	392 RES 2,E
01C8	CB48	317 BIT 1,B	0260	CB94	393 RES 2,H
01CA	CB49	318 BIT 1,C	0262	CB95	394 RES 2,L
01CC	CB4A	319 BIT 1,D	0264	CB96	395 RES 2,(HL)
01CE	CB4B	320 BIT 1,E	0266	CB97	396 RES 2,A
01D0	CB4C	321 BIT 1,H	0268	CB98	397 RES 3,B
01D2	CB4D	322 BIT 1,L	026A	CB99	398 RES 3,C
01D4	CB4E	323 BIT 1,(HL)	026C	CB9A	399 RES 3,D
01D6	CB4F	324 BIT 1,A	026E	CB9B	400 RES 3,E
01D8	CB50	325 BIT 2,B	0270	CB9C	401 RES 3,H
01DA	CB51	326 BIT 2,C	0272	CB9D	402 RES 3,L
01DC	CB52	327 BIT 2,D	0274	CB9E	403 RES 3,(HL)
01DE	CB53	328 BIT 2,E	0276	CB9F	404 RES 3,A
01E0	CB54	329 BIT 2,H	0278	CBA0	405 RES 4,B
01E2	CB55	330 BIT 2,L	027A	CBA1	406 RES 4,C
01E4	CB56	331 BIT 2,(HL)	027C	CBA2	407 RES 4,D
01E6	CB57	332 BIT 2,A	027E	CBA3	408 RES 4,E
01E8	CB58	333 BIT 3,B	0280	CBA4	409 RES 4,H
01EA	CB59	334 BIT 3,C	0282	CBA5	410 RES 4,L
01EC	CB5A	335 BIT 3,D	0284	CBA6	411 RES 4,(HL)
01EE	CB5B	336 BIT 3,E	0286	CBA7	412 RES 4,A
01F0	CB5C	337 BIT 3,H	0288	CBA8	413 RES 5,B
01F2	CB5D	338 BIT 3,L	028A	CBA9	414 RES 5,C
01F4	CB5E	339 BIT 3,(HL)	028C	CBAA	415 RES 5,D
01F6	CB5F	340 BIT 3,A	028E	CBAB	416 RES 5,E
01F8	CB60	341 BIT 4,B	0290	CBAC	417 RES 5,H
01FA	CB61	342 BIT 4,C	0292	CBAD	418 RES 5,L
01FC	CB62	343 BIT 4,D	0294	CBAE	419 RES 5,(HL)
01FE	CB63	344 BIT 4,E	0296	CBAF	420 RES 5,A
0200	CB64	345 BIT 4,H	0298	CBB0	421 RES 6,B
0202	CB65	346 BIT 4,L	029A	CBB1	422 RES 6,C
0204	CB66	347 BIT 4,(HL)	029C	CBB2	423 RES 6,D
0206	CB67	348 BIT 4,A	029E	CBB3	424 RES 6,E
0208	CB68	349 BIT 5,B	02A0	CBB4	425 RES 6,H
020A	CB69	350 BIT 5,C	02A2	CBB5	426 RES 6,L
020C	CB6A	351 BIT 5,D	02A4	CBB6	427 RES 6,(HL)
020E	CB6B	352 BIT 5,E	02A6	CBB7	428 RES 6,A
0210	CB6C	353 BIT 5,H	02A8	CBB8	429 RES 7,B
0212	CB6D	354 BIT 5,L	02AA	CBB9	430 RES 7,C
0214	CB6E	355 BIT 5,(HL)	02AC	CBBA	431 RES 7,D
0216	CB6F	356 BIT 5,A	02AE	CBBB	432 RES 7,E
0218	CB70	357 BIT 6,B	0280	CBBC	433 RES 7,H
021A	CB71	358 BIT 6,C	0282	CBBD	434 RES 7,L
021C	CB72	359 BIT 6,D	0284	CBBE	435 RES 7,(HL)
021E	CB73	360 BIT 6,E	0286	CBBF	436 RES 7,A
0220	CB74	361 BIT 6,H	0288	CBC0	437 SET 0,B
0222	CB75	362 BIT 6,L	02BA	CBC1	438 SET 0,C
0224	CB76	363 BIT 6,(HL)	02BC	CBC2	439 SET 0,D
0226	CB77	364 BIT 6,A	02BE	CBC3	440 SET 0,E
0228	CB78	365 BIT 7,B	02C0	CBC4	441 SET 0,H
022A	CB79	366 BIT 7,C	02C2	CBC5	442 SET 0,L
022C	CB7A	367 BIT 7,D	02C4	CBC6	443 SET 0,(HL)
022E	CB7B	368 BIT 7,E	02C6	CBC7	444 SET 0,A

Z-80 CROSS ASSEMBLER VERSION 1.06 OF 06/18/76

07/09/76 10:20:50

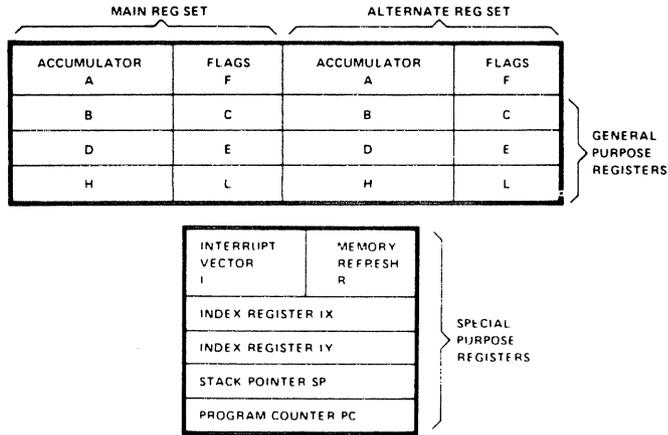
OPCODE LISTING

LOC	OBJ CODE	STMT	SOURCE STATEMENT	LOC	OBJ CODE	STMT	SOURCE STATEMENT
02C8	CBC8	445	SET 1,B	036F	DD7105	520	LD (IX+IND),C
02CA	CBC9	446	SET 1,C	0372	DD7205	521	LD (IX+IND),D
02CC	CBCA	447	SET 1,D	0375	DD7305	522	LD (IX+IND),E
02CE	CBCB	448	SET 1,E	0378	DD7405	523	LD (IX+IND),H
02D0	CBCD	449	SET 1,H	037B	DD7505	524	LD (IX+IND),L
02D2	CBCD	450	SET 1,L	037E	DD7705	525	LD (IX+IND),A
02D4	CBCD	451	SET 1,(HL)	0381	DD7E05	526	LD A,(IX+IND)
02D6	CBCF	452	SET 1,A	0384	DD8605	527	ADD A,(IX+IND)
02D8	CBD0	453	SET 2,B	0387	DD8E05	528	ADC A,(IX+IND)
02DA	CBD1	454	SET 2,C	038A	DD9605	529	SUB (IX+IND)
02DC	CBD2	455	SET 2,D	038D	DD9E05	530	SBC A,(IX+IND)
02DE	CBD3	456	SET 2,E	0390	DDA605	531	AND (IX+IND)
02E0	CBD4	457	SET 2,H	0393	DDAE05	532	XOR (IX+IND)
02E2	CBD5	458	SET 2,L	0396	DDB605	533	OR (IX+IND)
02E4	CBD6	459	SET 2,(HL)	0399	DDBE05	534	CP (IX+IND)
02E6	CBD7	460	SET 2,A	039C	DDE1	535	POP IX
02E8	CBD8	461	SET 3,B	039E	DDE3	536	EX (SP),IX
02EA	CBD9	462	SET 3,C	03A0	DDE5	537	PUSH IX
02EC	CBDA	463	SET 3,D	03A2	DDE9	538	JP (IX)
02EE	CBDB	464	SET 3,E	03A4	DDF9	539	LD SP,IX
02F0	CBDC	465	SET 3,H	03A6	DDCB0506	540	RLC (IX+IND)
02F2	CBDD	466	SET 3,L	03AA	DDCB050E	541	RRC (IX+IND)
02F4	CBDE	467	SET 3,(HL)	03AE	DDCB0516	542	RL (IX+IND)
02F6	CBDF	468	SET 3,A	03B2	DDCB051E	543	RR (IX+IND)
02F8	CBE0	469	SET 4,B	03B6	DDCB0526	544	SLA (IX+IND)
02FA	CBE1	470	SET 4,C	03BA	DDCB052E	545	SRA (IX+IND)
02FC	CBE2	471	SET 4,D	03BE	DDCB053E	546	SRL (IX+IND)
02FE	CBE3	472	SET 4,E	03C2	DDCB0546	547	BIT 0,(IX+IND)
0300	CBE4	473	SET 4,H	03C6	DDCB054E	548	BIT 1,(IX+IND)
0302	CBE5	474	SET 4,L	03CA	DDCB0556	549	BIT 2,(IX+IND)
0304	CBE6	475	SET 4,(HL)	03CE	DDCB055E	550	BIT 3,(IX+IND)
0306	CBE7	476	SET 4,A	03D2	DDCB0566	551	BIT 4,(IX+IND)
0308	CBE8	477	SET 5,B	03D6	DDCB056E	552	BIT 5,(IX+IND)
030A	CBE9	478	SET 5,C	03DA	DDCB0576	553	BIT 6,(IX+IND)
030C	CBEA	479	SET 5,D	03DE	DDCB057E	554	BIT 7,(IX+IND)
030E	CBEB	480	SET 5,E	03E2	DDCB0586	555	RES 0,(IX+IND)
0310	CBEC	481	SET 5,H	03E6	DDCB058E	556	RES 1,(IX+IND)
0312	CBED	482	SET 5,L	03EA	DDCB0596	557	RES 2,(IX+IND)
0314	CBEE	483	SET 5,(HL)	03EE	DDCB059E	558	RES 3,(IX+IND)
0316	CBEF	484	SET 5,A	03F2	DDCB05A6	559	RES 4,(IX+IND)
0318	CBF0	485	SET 6,B	03F6	DDCB05AE	560	RES 5,(IX+IND)
031A	CBF1	486	SET 6,C	03FA	DDCB05B6	561	RES 6,(IX+IND)
031C	CBF2	487	SET 6,D	03FE	DDCB05BE	562	RES 7,(IX+IND)
031E	CBF3	488	SET 6,E	0402	DDCB05C6	563	SET 0,(IX+IND)
0320	CBF4	489	SET 6,H	0406	DDCB05CE	564	SET 1,(IX+IND)
0322	CBF5	490	SET 6,L	040A	DDCB05D6	565	SET 2,(IX+IND)
0324	CBF6	491	SET 6,(HL)	040E	DDCB05DE	566	SET 3,(IX+IND)
0326	CBF7	492	SET 6,A	0412	DDCB05E6	567	SET 4,(IX+IND)
0328	CBF8	493	SET 7,B	0416	DDCB05EE	568	SET 5,(IX+IND)
032A	CBF9	494	SET 7,C	041A	DDCB05F6	569	SET 6,(IX+IND)
032C	CBFA	495	SET 7,D	041E	DDCB05FE	570	SET 7,(IX+IND)
032E	CBFB	496	SET 7,E	0422	ED40	571	IN B.(C)
0330	CBFC	497	SET 7,H	0424	ED41	572	OUT (C),B
0332	CBFD	498	SET 7,L	0426	ED42	573	SBC HL,BC
0334	CBFE	499	SET 7,(HL)	0428	ED438405	574	LD (NN),BC
0336	CBFF	500	SET 7,A	042C	ED44	575	NEG
0338	DD09	501	ADD IX,BC	042E	ED45	576	RETN
033A	DD19	502	ADD IX,DE	0430	ED46	577	IM 0
033C	DD218405	503	LD IX,NN	0432	ED47	578	LD I,A
0340	DD228405	504	LD (NN),IX	0434	ED48	579	IN C.(C)
0344	DD23	505	INC IX	0436	ED49	580	OUT (C),C
0346	DD29	506	ADD IX,IX	0438	ED4A	581	ADC HL,BC
0348	DD2A8405	507	LD IX,(NN)	043A	ED4B8405	582	LD BC,(NN)
034C	DD2B	508	DEC IX	043E	ED4D	583	RETI
034E	DD3405	509	INC (IX+IND)	0440	ED50	584	IN D.(C)
0351	DD3505	510	DEC (IX+IND)	0442	ED51	585	OUT (C),D
0354	DD360520	511	LD (IX+IND),N	0444	ED52	586	SBC HL,DE
0358	DD39	512	ADD IX,SP	0446	ED538405	587	LD (NN),DE
035A	DD4605	513	LD B,(IX+IND)	044A	ED56	588	IM 1
035D	DD4E05	514	LD C,(IX+IND)	044C	ED57	589	LD A,I
0360	DD5605	515	LD D,(IX+IND)	044E	ED58	590	IN E.(C)
0363	DD5E05	516	LD E,(IX+IND)	0450	ED59	591	OUT (C),E
0366	DD6605	517	LD H,(IX+IND)	0452	ED5A	592	ADC HL,DE
0369	DD6E05	518	LD L,(IX+IND)	0454	ED5B8405	593	LD DE,(NN)
036C	DD7005	519	LD (IX+IND),B	0458	ED5E	594	IM 2

07/09/76 10:20:50

OPCODE LISTING

LOC	OBJ CODE	STMT SOURCE STATEMENT	LOC	OBJ CODE	STMT SOURCE STATEMENT
045A	ED60	595 IN H.(C)	0520	FDCB053E	670 SRL (IY+IND)
045C	ED61	596 OUT (C).H	0524	FDCB0546	671 BIT 0.(IY+IND)
045E	ED62	597 SBC HL,HL	0528	FDCB054E	672 BIT 1.(IY+IND)
0460	ED67	598 RRD	052C	FDCB0556	673 BIT 2.(IY+IND)
0462	ED68	599 IN L.(C)	0530	FDCB055E	674 BIT 3.(IY+IND)
0464	ED69	600 OUT (C).L	0534	FDCB0566	675 BIT 4.(IY+IND)
0466	ED6A	601 ADC HL,HL	0538	FDCB056E	676 BIT 5.(IY+IND)
0468	ED6F	602 RLD	053C	FDCB0576	677 BIT 6.(IY+IND)
046A	ED72	603 SBC HL,SP	0540	FDCB057E	678 BIT 7.(IY+IND)
046C	ED738405	604 LD (NN),SP	0544	FDCB0586	679 RES 0.(IY+IND)
0470	ED78	605 IN A.(C)	0548	FDCB058E	680 RES 1.(IY+IND)
0472	ED79	606 OUT (C).A	054C	FDCB0596	681 RES 2.(IY+IND)
0474	ED7A	607 ADC HL,SP	0550	FDCB059E	682 RES 3.(IY+IND)
0476	ED7B8405	608 LD SP, (NN)	0554	FDCB05A6	683 RES 4.(IY+IND)
047A	EDA0	609 LDI	0558	FDCB05AE	684 RES 5.(IY+IND)
047C	EDA1	610 CPI	055C	FDCB05B6	685 RES 6.(IY+IND)
047E	EDA2	611 INI	0560	FDCB05BE	686 RES 7.(IY+IND)
0480	EDA3	612 OUTI	0564	FDCB05C6	687 SET 0.(IY+IND)
0482	EDA8	613 LDD	0568	FDCB05CE	688 SET 1.(IY+IND)
0484	EDA9	614 CPD	056C	FDCB05D6	689 SET 2.(IY+IND)
0486	EDAA	615 IND	0570	FDCB05DE	690 SET 3.(IY+IND)
0488	EDAB	616 OUTD	0574	FDCB05E6	691 SET 4.(IY+IND)
048A	EDB0	617 LDIR	0578	FDCB05EE	692 SET 5.(IY+IND)
048C	EDB1	618 CPIR	057C	FDCB05F6	693 SET 6.(IY+IND)
048E	EDB2	619 INIR	0580	FDCB05FE	694 SET 7.(IY+IND)
0490	EDB3	620 OTIR	0584		695 NN DEFS 2
0492	EDB8	621 LDDR			696 IND EQU 5
0494	EDB9	622 CPDR			697 M EQU 10H
0496	EDBA	623 INDR			698 N EQU 20H
0498	EDBB	624 OTDR			699 DIS EQU 30H
049A	FD09	625 ADD IY,BC			700 END
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	FD4E05	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	FD9	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)			



Z80-CPU REGISTER CONFIGURATION

HEXADECIMAL COLUMNS					
6	5	4	3	2	1
HEX = DEC	HEX = DEC	HEX = DEC	HEX = DEC	HEX = DEC	HEX = DEC
0	0	0	0	0	0
1	1 048 576	1 65 536	1 4 096	1 256	1 16 1 1
2	2 097 152	2 131 072	2 8 192	2 512	2 32 2 2
3	3 145 728	3 196 608	3 12 288	3 768	3 48 3 3
4	4 194 304	4 262 144	4 16 384	4 1 024	4 64 4 4
5	5 242 880	5 327 680	5 20 480	5 1 280	5 80 5 5
6	6 291 456	6 393 216	6 24 576	6 1 536	6 96 6 6
7	7 340 032	7 458 752	7 28 672	7 1 792	7 112 7 7
8	8 388 608	8 524 288	8 32 768	8 2 048	8 128 8 8
9	9 437 184	9 589 824	9 36 864	9 2 304	9 144 9 9
A	10 485 760	A 655 360	A 40 960	A 2 560	A 160 A 10
B	11 534 336	B 720 896	B 45 056	B 2 816	B 176 B 11
C	12 582 912	C 786 432	C 49 152	C 3 072	C 192 C 12
D	13 631 488	D 851 968	D 53 248	D 3 328	D 208 D 13
E	14 680 064	E 917 504	E 57 344	E 3 584	E 224 E 14
F	15 728 640	F 983 040	F 61 440	F 3 840	F 240 F 15
0 1 2 3	4 5 6 7	0 1 2 3	4 5 6 7	0 1 2 3	4 5 6 7
BYTE		BYTE		BYTE	

ASCII CHARACTER SET (7-BIT CODE)

MSD \ LSD	0	1	2	3	4	5	6	7
	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	p
1	0001	SOH	DC1	!	1	A	Q	a
2	0010	STX	DC2	"	2	B	R	b
3	0011	ETX	DC3	#	3	C	S	c
4	0100	EOT	DC4	\$	4	D	T	d
5	0101	ENG	NAK	%	5	E	U	e
6	0110	ACK	SYN	&	6	F	V	f
7	0111	BEL	ETB	'	7	G	W	g
8	1000	BS	CAN	(8	H	X	h
9	1001	HT	EM)	9	I	Y	i
A	1010	LF	SUB	*		J	Z	j
B	1011	VT	ESC	+		K	[k
C	1100	FF	FS	,		L	\	l
D	1101	CR	GS	-		M]	m
E	1110	SO	RS	.		N	^	n
F	1111	SI	VS	/			_	o
				:			DEL	DEL

POWERS OF 2

2 ⁿ	n
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24

2 ⁰ = 16 ⁰
2 ⁴ = 16 ¹
2 ⁸ = 16 ²
2 ¹² = 16 ³
2 ¹⁶ = 16 ⁴
2 ²⁰ = 16 ⁵
2 ²⁴ = 16 ⁶
2 ²⁸ = 16 ⁷
2 ³² = 16 ⁸
2 ³⁶ = 16 ⁹
2 ⁴⁰ = 16 ¹⁰
2 ⁴⁴ = 16 ¹¹
2 ⁴⁸ = 16 ¹²
2 ⁵² = 16 ¹³
2 ⁵⁶ = 16 ¹⁴
2 ⁶⁰ = 16 ¹⁵

POWERS OF 16

16 ⁿ	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15

ALPHABETIC LIST OF INSTRUCTION SET

Z-80 CROSS ASSEMBLER VERSION 1.06 OF 06/18/76

07/09/76 10:22:47

OPCODE LISTING

LOC	OBJ CODE	STMT	SOURCE STATEMENT	LOC	OBJ CODE	STMT	SOURCE STATEMENT
0000	8E	1	ADC A, (HL)	0088	CB50	74	BIT 2, B
0001	DD8E05	2	ADC A, (IX+IND)	008A	CB51	75	BIT 2, C
0004	FD8E05	3	ADC A, (IY+IND)	008C	CB52	76	BIT 2, D
0007	8F	4	ADC A, A	008E	CB53	77	BIT 2, E
0008	88	5	ADC A, B	0090	CB54	78	BIT 2, H
0009	89	6	ADC A, C	0092	CB55	79	BIT 2, L
000A	8A	7	ADC A, D	0094	CB5E	80	BIT 3, (HL)
000B	8B	8	ADC A, E	0096	DDCB055E	81	BIT 3, (IX+IND)
000C	8C	9	ADC A, H	009A	FDCB055E	82	BIT 3, (IY+IND)
000D	8D	10	ADC A, L	009E	CB5F	83	BIT 3, A
000E	CE20	11	ADC A, N	00A0	CB58	84	BIT 3, B
0010	ED4A	12	ADC HL, BC	00A2	CB59	85	BIT 3, C
0012	ED5A	13	ADC HL, DE	00A4	CB5A	86	BIT 3, D
0014	ED6A	14	ADC HL, HL	00A6	CB5B	87	BIT 3, E
0016	ED7A	15	ADC HL, SP	00A8	CB5C	88	BIT 3, H
0018	86	16	ADD A, (HL)	00AA	CB5D	89	BIT 3, L
0019	DD8605	17	ADD A, (IX+IND)	00AC	CB66	90	BIT 4, (HL)
001C	FD8605	18	ADD A, (IY+IND)	00AE	DDCB0566	91	BIT 4, (IX+IND)
001F	87	19	ADD A, A	00B2	FDCB0566	92	BIT 4, (IY+IND)
0020	80	20	ADD A, B	00B6	CB67	93	BIT 4, A
0021	81	21	ADD A, C	00B8	CB60	94	BIT 4, B
0022	82	22	ADD A, D	00BA	CB61	95	BIT 4, C
0023	83	23	ADD A, E	00BC	CB62	96	BIT 4, D
0024	84	24	ADD A, H	00BE	CB63	97	BIT 4, E
0025	85	25	ADD A, L	00C0	CB64	98	BIT 4, H
0026	C620	26	ADD A, N	00C2	CB65	99	BIT 4, L
0028	09	27	ADD HL, BC	00C4	CB6E	100	BIT 5, (HL)
0029	19	28	ADD HL, DE	00C6	DDCB056E	101	BIT 5, (IX+IND)
002A	29	29	ADD HL, HL	00CA	FDCB056E	102	BIT 5, (IY+IND)
002B	39	30	ADD HL, SP	00CE	CB6F	103	BIT 5, A
002C	DD09	31	ADD IX, BC	00D0	CB68	104	BIT 5, B
002E	DD19	32	ADD IX, DE	00D2	CB69	105	BIT 5, C
0030	DD29	33	ADD IX, IX	00D4	CB6A	106	BIT 5, D
0032	DD39	34	ADD IX, SP	00D6	CB6B	107	BIT 5, E
0034	FD09	35	ADD IY, BC	00D8	CB6C	108	BIT 5, H
0036	FD19	36	ADD IY, DE	00DA	CB6D	109	BIT 5, L
0038	FD29	37	ADD IY, IY	00DC	CB76	110	BIT 6, (HL)
003A	FD39	38	ADD IY, SP	00DE	DDCB0576	111	BIT 6, (IX+IND)
003C	A6	39	AND (HL)	00E2	FDCB0576	112	BIT 6, (IY+IND)
003D	DDA605	40	AND (IX+IND)	00E6	CB77	113	BIT 6, A
0040	FDA605	41	AND (IY+IND)	00E8	CB70	114	BIT 6, B
0043	A7	42	AND A	00EA	CB71	115	BIT 6, C
0044	A0	43	AND B	00EC	CB72	116	BIT 6, D
0045	A1	44	AND C	00EE	CB73	117	BIT 6, E
0046	A2	45	AND D	00F0	CB74	118	BIT 6, H
0047	A3	46	AND E	00F2	CB75	119	BIT 6, L
0048	A4	47	AND H	00F4	CB7E	120	BIT 7, (HL)
0049	A5	48	AND L	00F6	DDCB057E	121	BIT 7, (IX+IND)
004A	E620	49	AND N	00FA	FDCB057E	122	BIT 7, (IY+IND)
004C	CB46	50	BIT O, (HL)	00FE	CB7F	123	BIT 7, A
004E	DDCB0546	51	BIT O, (IX+IND)	0100	CB78	124	BIT 7, B
0052	FDBC0546	52	BIT O, (IY+IND)	0102	CB79	125	BIT 7, C
0056	CB47	53	BIT O, A	0104	CB7A	126	BIT 7, D
0058	CB40	54	BIT O, B	0106	CB7B	127	BIT 7, E
005A	CB41	55	BIT O, C	0108	CB7C	128	BIT 7, H
005C	CB42	56	BIT O, D	010A	CB7D	129	BIT 7, L
005E	CB43	57	BIT O, E	010C	DC8405	130	CALL C, NN
0060	CB44	58	BIT O, H	010F	FC8405	131	CALL M, NN
0062	CB45	59	BIT O, L	0112	D48405	132	CALL NC, NN
0064	CB4E	60	BIT 1, (HL)	0115	CD8405	133	CALL NN
0066	DDCB054E	61	BIT 1, (IX+IND)	0118	C48405	134	CALL NZ, NN
006A	FDCB054E	62	BIT 1, (IY+IND)	011B	F48405	135	CALL P, NN
006E	CB4F	63	BIT 1, A	011E	EC8405	136	CALL PE, NN
0070	CB48	64	BIT 1, B	0121	E48405	137	CALL PO, NN
0072	CB49	65	BIT 1, C	0124	CC8405	138	CALL Z, NN
0074	CB4A	66	BIT 1, D	0127	3F	139	CCF
0076	CB4B	67	BIT 1, E	0128	BE	140	CP (HL)
0078	CB4C	68	BIT 1, H	0129	DDBE05	141	CP (IX+IND)
007A	CB4D	69	BIT 1, L	012C	FDBE05	142	CP (IY+IND)
007C	CB56	70	BIT 2, (HL)	012F	BF	143	CP A
007E	DDCB0556	71	BIT 2, (IX+IND)	0130	B8	144	CP B
0082	FDCB0556	72	BIT 2, (IY+IND)	0131	B9	145	CP C
0086	CB57	73	BIT 2, A	0132	BA	146	CP D

07/09/76 10:22:47		OPCODE LISTING				OPCODE LISTING			
LOC	OBJ CODE	STMT	SOURCE STATEMENT	LOC	OBJ CODE	STMT	SOURCE STATEMENT	LOC	SOURCE STATEMENT
0133	BB	147	CP E	01AD	F28405	222	JP P, NN		
0134	BC	148	CP H	01B0	EA8405	223	JP PE, NN		
0135	BD	149	CP L	01B3	E28405	224	JP PO, NN		
0136	FE20	150	CP N	01B6	CA8405	225	JP Z, NN		
0138	EDA9	151	CPD	01B9	382E	226	JR C, DIS		
013A	EDB9	152	CPDR	01BB	182E	227	JR DIS		
013C	EDA1	153	CPI	01BD	302E	228	JR NC, DIS		
013E	EDB1	154	CPIR	01BF	202E	229	JR NZ, DIS		
0140	2F	155	CPL	01C1	282E	230	JR Z, DIS		
0141	27	156	DAA	01C3	02	231	LD (BC), A		
0142	35	157	DEC (HL)	01C4	12	232	LD (DE), A		
0143	DD3505	158	DEC (IX+IND)	01C5	77	233	LD (HL), A		
0146	FD3505	159	DEC (IY+IND)	01C6	70	234	LD (HL), B		
0149	3D	160	DEC A	01C7	71	235	LD (HL), C		
014A	05	161	DEC B	01C8	72	236	LD (HL), D		
014B	0B	162	DEC BC	01C9	73	237	LD (HL), E		
014C	0D	163	DEC C	01CA	74	238	LD (HL), H		
014D	15	164	DEC D	01CB	75	239	LD (HL), L		
014E	1B	165	DEC DE	01CC	3620	240	LD (HL), N		
014F	1D	166	DEC E	01CE	DD7705	241	LD (IX+IND), A		
0150	25	167	DEC H	01D1	DD7005	242	LD (IX+IND), B		
0151	2B	168	DEC HL	01D4	DD7105	243	LD (IX+IND), C		
0152	DD2B	169	DEC IX	01D7	DD7205	244	LD (IX+IND), D		
0154	FD2B	170	DEC IY	01DA	DD7305	245	LD (IX+IND), E		
0156	2D	171	DEC L	01DD	DD7405	246	LD (IX+IND), H		
0157	3B	172	DEC SP	01E0	DD7505	247	LD (IX+IND), L		
0158	F3	173	DI	01E3	DD360520	248	LD (IX+IND), N		
0159	102E	174	DJNZ DIS	01E7	FD7705	249	LD (IY+IND), A		
015B	FB	175	EI	01EA	FD7005	250	LD (IY+IND), B		
015C	E3	176	EX (SP), HL	01ED	FD7105	251	LD (IY+IND), C		
015D	DDE3	177	EX (SP), IX	01F0	FD7205	252	LD (IY+IND), D		
015F	FDE3	178	EX (SP), IY	01F3	FD7305	253	LD (IY+IND), E		
0161	08	179	EX AF, AF'	01F6	FD7405	254	LD (IY+IND), H		
0162	EB	180	EX DE, HL	01F9	FD7505	255	LD (IY+IND), L		
0163	D9	181	EXX	01FC	FD360520	256	LD (IY+IND), N		
0164	76	182	HALT	0200	328405	257	LD (NN), A		
0165	ED46	183	IM 0	0203	ED438405	258	LD (NN), BC		
0167	ED56	184	IM 1	0207	ED538405	259	LD (NN), DE		
0169	ED5E	185	IM 2	020B	228405	260	LD (NN), HL		
016B	ED78	186	IN A, (C)	020E	DD228405	261	LD (NN), IX		
016D	DB20	187	IN A, N	0212	FD228405	262	LD (NN), IY		
016F	ED40	188	IN B, (C)	0216	ED738405	263	LD (NN), SP		
0171	ED48	189	IN C, (C)	021A	0A	264	LD A, (BC)		
0173	ED50	190	IN D, (C)	021B	1A	265	LD A, (DE)		
0175	ED58	191	IN E, (C)	021C	7E	266	LD A, (HL)		
0177	ED60	192	IN H, (C)	021D	DD7E05	267	LD A, (IX+IND)		
0179	ED68	193	IN L, (C)	0220	FD7E05	268	LD A, (IY+IND)		
017B	34	194	INC (HL)	0223	3A8405	269	LD A, (NN)		
017C	DD3405	195	INC (IX+IND)	0226	7F	270	LD A, A		
017F	FD3405	196	INC (IY+IND)	0227	78	271	LD A, B		
0182	3C	197	INC A	0228	79	272	LD A, C		
0183	04	198	INC B	0229	7A	273	LD A, D		
0184	03	199	INC BC	022A	7B	274	LD A, E		
0185	0C	200	INC C	022B	7C	275	LD A, H		
0186	14	201	INC D	022C	ED57	276	LD A, I		
0187	13	202	INC DE	022E	7D	277	LD A, L		
0188	1C	203	INC E	022F	3E20	278	LD A, N		
0189	24	204	INC H	0231	46	279	LD B, (HL)		
018A	23	205	INC HL	0232	DD4605	280	LD B, (IX+IND)		
018B	DD23	206	INC IX	0235	FD4605	281	LD B, (IY+IND)		
018D	FD23	207	INC IY	0238	47	282	LD B, A		
018F	2C	208	INC L	0239	40	283	LD B, B		
0190	33	209	INC SP	023A	41	284	LD B, C		
0191	EDAA	210	IND	023B	42	285	LD B, D		
0193	EDBA	211	INDR	023C	43	286	LD B, E		
0195	EDA2	212	INI	023D	44	287	LD B, H, NN		
0197	EDB2	213	INIR	023E	45	288	LD B, L		
0199	E9	214	JP (HL)	023F	0620	289	LD B, N		
019A	DDE9	215	JP (IX)	0241	ED4B8405	290	LD BC, (NN)		
019C	FDE9	216	JP (IY)	0245	018405	291	LD BC, NN		
019E	DA8405	217	JP C, NN	0248	4E	292	LD C, (HL)		
01A1	FA8405	218	JP M, NN	0249	DD4E05	293	LD C, (IX+IND)		
01A4	D28405	219	JP NC, NN	024C	FD4E05	294	LD C, (IY+IND)		
01A7	C38405	220	JP NN	024F	4F	295	LD C, A		
01AA	C28405	221	JP NZ, NN	0250	48	296	LD C, B		

07/09/76 10:22:47

OPCODE LISTING

LOC	OBJ CODE	STMT	SOURCE STATEMENT	LOC	OBJ CODE	STMT	SOURCE STATEMENT
0251	49	297	LD C, C	02D8	B2	373	OR D
0252	4A	298	LD C, D	02D9	B3	374	OR E
0253	4B	299	LD C, E	02DA	B4	375	OR H
0254	4C	300	LD C, H	02DB	B5	376	OR L
0255	4D	301	LD C, L	02DC	F620	377	OR N
0256	0E20	302	LD C, N	02DE	EDBB	378	OTDR
0258	56	303	LD D, (HL)	02E0	EDB3	379	OTIR
0259	DD5605	304	LD D, (IX+IND)	02E2	ED79	380	OUT (C),A
025C	FD5605	305	LD D, (IY+IND)	02E4	ED41	381	OUT (C),B
025F	57	306	LD D, A	02E6	ED49	382	OUT (C),C
0260	50	307	LD D, B	02E8	ED51	383	OUT (C),D
0261	51	308	LD D, C	02EA	ED59	384	OUT (C),E
0262	52	309	LD D, D	02EC	ED61	385	OUT (C),H
0263	53	310	LD D, E	02EE	ED69	386	OUT (C),L
0264	54	311	LD D, H	02F0	D320	387	OUT N,A
0265	55	312	LD D, L	02F2	EDAB	388	OUTD
0266	1620	313	LD D, N	02F4	EDA3	389	OUTI
0268	ED5B8405	314	LD DE, (NN)	02F6	F1	390	POP AF
026C	118405	315	LD DE, NN	02F7	C1	391	POP BC
026F	5E	316	LD E, (HL)	02F8	D1	392	POP DE
0270	DD5E05	317	LD E, (IX+IND)	02F9	E1	393	POP HL
0273	FD5E05	318	LD E, (IY+IND)	02FA	DDE1	394	POP IX
0276	5F	319	LD E, A	02FC	FDE1	395	POP IY
0277	58	320	LD E, B	02FE	F5	396	PUSH AF
0278	59	321	LD E, C	02FF	C5	397	PUSH BC
0279	5A	322	LD E, D	0300	D5	398	PUSH DE
027A	5B	323	LD E, E	0301	E5	399	PUSH HL
027B	5C	324	LD E, H	0302	DDE5	400	PUSH IX
027C	5D	325	LD E, L	0304	FDE5	401	PUSH IY
027D	1E20	326	LD E, N	0306	CB86	402	RES 0,(HL)
027F	66	327	LD H, (HL)	0308	DDCB0586	403	RES 0,(IX+IND)
0280	DD6605	328	LD H, (IX+IND)	030C	FDCB0586	404	RES 0,(IY+IND)
0283	FD6605	329	LD H, (IY+IND)	0310	CB87	405	RES 0,A
0286	67	330	LD H, A	0312	CB80	406	RES 0,B
0287	60	331	LD H, B	0314	CB81	407	RES 0,C
0288	61	332	LD H, C	0316	CB82	408	RES 0,D
0289	62	333	LD H, D	0318	CB83	409	RES 0,E
028A	63	334	LD H, E	031A	CB84	410	RES 0,H
028B	64	335	LD H, H	031C	CB85	411	RES 0,L
028C	65	336	LD H, L	031E	CB8E	412	RES 1,(HL)
028D	2620	337	LD H, N	0320	DDCB058E	413	RES 1,(IX+IND)
028F	2A8405	338	LD HL, (NN)	0324	FDCB058E	414	RES 1,(IY+IND)
0292	218405	339	LD HL, NN	0328	CB8F	415	RES 1,A
0295	ED47	340	LD I, A	032A	CB88	416	RES 1,B
0297	DD2A8405	341	LD IX, (NN)	032C	CB89	417	RES 1,C
029B	DD218405	342	LD IX, NN	032E	CB8A	418	RES 1,D
029F	FD2A8405	343	LD IY, (NN)	0330	CB8B	419	RES 1,E
02A3	FD218405	344	LD IY, NN	0332	CB8C	420	RES 1,H
02A7	6E	345	LD L, (HL)	0334	CB8D	421	RES 1,L
02A8	DD6E05	346	LD L, (IX+IND)	0336	CB96	422	RES 2,(HL)
02AB	FD6E05	347	LD L, (IY+IND)	0338	DDCB0596	423	RES 2,(IX+IND)
02AE	6F	348	LD L, A	033C	FDCB0596	424	RES 2,(IY+IND)
02AF	68	349	LD L, B	0340	CB97	425	RES 2,A
02B0	69	350	LD L, C	0342	CB90	426	RES 2,B
02B1	6A	351	LD L, D	0344	CB91	427	RES 2,C
02B2	6B	352	LD L, E	0346	CB92	428	RES 2,D
02B3	6C	353	LD L, H	0348	CB93	429	RES 2,E
02B4	6D	354	LD L, L	034A	CB94	430	RES 2,H
02B5	2E20	355	LD L, N	034C	CB95	431	RES 2,L
02B7	ED7B8405	356	LD LN, (NN)	034E	CB9E	432	RES 3,(HL)
02BB	F9	357	LD SP, HL	0350	DDCB059E	433	RES 3,(IX+IND)
02BC	DDF9	358	LD SP, IX	0354	FDCB059E	434	RES 3,(IY+IND)
02BE	FD9F	359	LD SP, IY	0358	CB9F	435	RES 3,A
02C0	318405	360	LD SP, NN	035A	CB98	436	RES 3,B
02C3	EDA8	361	LDD	035C	CB99	437	RES 3,C
02C5	EDB8	362	LDDR	035E	CB9A	438	RES 3,D
02C7	EDA0	363	LDI	0360	CB9B	439	RES 3,E
02C9	EDB0	364	LDIR	0362	CB9C	440	RES 3,H
02CB	ED44	365	NEG	0364	CB9D	441	RES 3,L
02CD	00	366	NOP	0366	CBA6	442	RES 4,(HL)
02CE	B6	367	OR (HL)	0368	DDCB05A6	443	RES 4,(IX+IND)
02CF	DDB605	368	OR (IX+IND)	036C	FDCB05A6	444	RES 4,(IY+IND)
02D2	FDB605	369	OR (IY+IND)	0370	CBA7	445	RES 4,A
02D5	B7	370	OR A	0372	CBA0	446	RES 4,B
02D6	B0	371	OR B	0374	CBA1	447	RES 4,C
02D7	B1	372	OR C	0376	CBA2	448	RES 4,D

Z-80 CROSS ASSEMBLER VERSION 1.06 OF 06/18/76

07/09/76 10:22:47

OPCODE LISTING

LOC	OBJ CODE	STMT	SOURCE STATEMENT	LOC	OBJ CODE	STMT	SOURCE STATEMENT
0378	CBA3	449	RES 4,E	041B	CB1C	524	RR H
037A	CBA4	450	RES 4,H	041D	CB1D	525	RR L
037C	CBA5	451	RES 4,L	041F	1F	526	RRA
037E	CBAE	452	RES 5,(HL)	0420	CB0E	527	RRC (HL)
0380	DDCB05AE	453	RES 5,(IX+IND)	0422	DDCB050E	528	RRC (IX+IND)
0384	FDCB05AE	454	RES 5,(IY+IND)	0426	FDCB050E	529	RRC (IY+IND)
0388	CBAF	455	RES 5,A	042A	CB0F	530	RRC A
038A	CBA8	456	RES 5,B	042C	CB08	531	RRC B
038C	CBA9	457	RES 5,C	042E	CB09	532	RRC C
038E	CBAA	458	RES 5,D	0430	CB0A	533	RRC D
0390	CBAB	459	RES 5,E	0432	CB0B	534	RRC E
0392	CBAC	460	RES 5,H	0434	CB0C	535	RRC H
0394	CBAD	461	RES 5,L	0436	CB0D	536	RRC L
0396	CBB6	462	RES 6,(HL)	0438	0F	537	RRCA
0398	DDCB05B6	463	RES 6,(IX+IND)	0439	ED67	538	RRD
039C	FDCB05B6	464	RES 6,(IY+IND)	043B	C7	539	RST 0
03A0	CBB7	465	RES 6,A	043C	D7	540	RST 10H
03A2	CBB0	466	RES 6,B	043D	DF	541	RST 18H
03A4	CBB1	467	RES 6,C	043E	E7	542	RST 20H
03A6	CBB2	468	RES 6,D	043F	EF	543	RST 28H
03A8	CBB3	469	RES 6,E	0440	F7	544	RST 30H
03AA	CBB4	470	RES 6,H	0441	FF	545	RST 38H
03AC	CBB5	471	RES 6,L	0442	CF	546	RST 8
03AE	CBBE	472	RES 7,(HL)	0443	9E	547	SBC A,(HL)
03B0	DDCB05BE	473	RES 7,(IX+IND)	0444	DD9E05	548	SBC A,(IX+IND)
03B4	FDCB05BE	474	RES 7,(IY+IND)	0447	FD9E05	549	SBC A,(IY+IND)
03B8	CBBF	475	RES 7,A	044A	9F	550	SBC A.A
03BA	CBB8	476	RES 7,B	044B	98	551	SBC A.B
03BC	CBB9	477	RES 7,C	044C	99	552	SBC A.C
03BE	CBBA	478	RES 7,D	044D	9A	553	SBC A.D
03C0	CBBB	479	RES 7,E	044E	9B	554	SBC A.E
03C2	CBBC	480	RES 7,H	044F	9C	555	SBC A.H
03C4	CBBD	481	RES 7,L	0450	9D	556	SBC A.L
03C6	C9	482	RET	0451	DE20	557	SBC A.N
03C7	D8	483	RET C	0453	ED42	558	SBC HL,BC
03C8	F8	484	RET M	0455	ED52	559	SBC HL,DE
03C9	D0	485	RET NC	0457	ED62	560	SBC HL,HL
03CA	C0	486	RET NZ	0459	ED72	561	SBC HL,SP
03CB	F0	487	RET P	045B	37	562	SCF
03CC	E8	488	RET PE	045C	CBC6	563	SET 0,(HL)
03CD	E0	489	RET PO	045E	DDCB05C6	564	SET 0,(IX+IND)
03CE	C8	490	RET Z	0462	FDCB05C6	565	SET 0,(IY+IND)
03CF	ED4D	491	RETI	0466	CBC7	566	SET 0,A
03D1	ED45	492	RETN	0468	CBC0	567	SET 0,B
03D3	CB16	493	RL (HL)	046A	CBC1	568	SET 0,C
03D5	DDCB0516	494	RL (IX+IND)	046C	CBC2	569	SET 0,D
03D9	FDCB0516	495	RL (IY+IND)	046E	CBC3	570	SET 0,E
03DD	CB17	496	RL A	0470	CBC4	571	SET 0,H
03DF	CB10	497	RL B	0472	CBC5	572	SET 0,L
03E1	CB11	498	RL C	0474	CBCE	573	SET 1,(HL)
03E3	CB12	499	RL D	0476	DDCB05CE	574	SET 1,(IX+IND)
03E5	C813	500	RL E	047A	FDCB05CE	575	SET 1,(IY+IND)
03E7	CB14	501	RL H	047E	CBCF	576	SET 1,A
03E9	CB15	502	RL L	0480	CBC8	577	SET 1,B
03EB	17	503	RLA	0482	CBC9	578	SET 1,C
03EC	CB06	504	RLC (HL)	0484	CBCA	579	SET 1,D
03EE	DDCB0506	505	RLC (IX+IND)	0486	CBCB	580	SET 1,E
03F2	FDCB0506	506	RLC (IY+IND)	0488	CBCC	581	SET 1,H
03F6	CB07	507	RLC A	048A	CB0D	582	SET 1,L
03F8	CB00	508	RLC B	048C	CBD6	583	SET 2,(HL)
03FA	CB01	509	RLC C	048E	DDCB05D6	584	SET 2,(IX+IND)
03FC	CB02	510	RLC D	0492	FDCB05D6	585	SET 2,(IY+IND)
03FE	CB03	511	RLC E	0496	CBD7	586	SET 2,A
0400	CB04	512	RLC H	0498	CBD0	587	SET 2,B
0402	CB05	513	RLC L	049A	CBD1	588	SET 2,C
0404	07	514	RLCA	049C	CBD2	589	SET 2,D
0405	ED6F	515	RLD	049E	CBD3	590	SET 2,E
0407	CB1E	516	RR (HL)	04A0	CBD4	591	SET 2,H
0409	DDCB051E	517	RR (IX+IND)	04A2	CBD5	592	SET 2,L
040D	FDCB051E	518	RR (IY+IND)	04A4	CBD8	593	SET 3,B
0411	CB1F	519	RR A	04A6	CBDE	594	SET 3,(HL)
0413	CB18	520	RR B	04A8	DDCB05DE	595	SET 3,(IX+IND)
0415	CB19	521	RR C	04AC	FDCB05DE	596	SET 3,(IY+IND)
0417	CB1A	522	RR D	04B0	CBDF	597	SET 3,A
0419	CB1B	523	RR E	04B2	CBD9	598	SET 3,C

07/09/76 10:22:47

OPCODE LISTING

LOC	OBJ CODE	STMT	SOURCE STATEMENT	LOC	OBJ CODE	STMT	SOURCE STATEMENT
04B4	CBDA	599	SET 3,D	0568	FD9605	675	SUB (IY+IND)
04B6	CBDB	600	SET 3,E	056B	97	676	SUB A
04B8	CBDC	601	SET 3,H	056C	90	677	SUB B
04BA	CBDD	602	SET 3,L	056D	91	678	SUB C
04BC	CBE6	603	SET 4,(HL)	056E	92	679	SUB D
04BE	DDCB05E6	604	SET 4,(IX+IND)	056F	93	680	SUB E
04C2	FDCB05E6	605	SET 4,(IY+IND)	0570	94	681	SUB H
04C6	CBE7	606	SET 4,A	0571	95	682	SUB L
04C8	CBE0	607	SET 4,B	0572	D620	683	SUB N
04CA	CBE1	608	SET 4,C	0574	AE	684	XOR (HL)
04CC	CBE2	609	SET 4,D	0575	DDAE05	685	XOR (IX+IND)
04CE	CBE3	610	SET 4,E	0578	FDAE05	686	XOR (IY+IND)
04D0	CBE4	611	SET 4,H	057B	AF	687	XOR A
04D2	CBE5	612	SET 4,L	057C	A8	688	XOR B
04D4	CBEE	613	SET 5,(HL)	057D	A9	689	XOR C
04D6	DDCB05EE	614	SET 5,(IX+IND)	057E	AA	690	XOR D
04DA	FDCB05EE	615	SET 5,(IY+IND)	057F	AB	691	XOR E
04DE	CBEF	616	SET 5,A	0580	AC	692	XOR H
04E0	CBE8	617	SET 5,B	0581	AD	693	XOR L
04E2	CBE9	618	SET 5,C	0582	EE20	694	XOR N
04E4	CBEA	619	SET 5,D	0584		695 NN	DEFS 2
04E6	CBEB	620	SET 5,E			696 IND	EQU 5
04E8	CBEC	621	SET 5,H			697 M	EQU 10H
04EA	CBED	622	SET 5,L			698 N	EQU 20H
04EC	CBF6	623	SET 6,(HL)			699 DIS	EQU 30H
04EE	DDCB05F6	624	SET 6,(IX+IND)			700	END
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	CBF9	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)				

Level II Basic Addresses

TURN ON CURSOR
CHARACTER

```

PUSH      DE      ;MUST SAVE
PUSH      IY      ;  DE & IY
LD        A,0EH   ;0EH IS CURSOR BYTE
CALL      33H     ;DISPLAY ROUTINE
POP       IY      ;RESTORE
POP       DE      ;  DE & IY
    
```

KEYBOARD SCAN
A-register contains byte when loop
falls through.
Byte is not displayed on Screen!

AGN

```

PUSH      DE      ;MUST SAVE
PUSH      IY      ;  DE & IY
CALL      2BH     ;SCAN ROUTINE
OR        A       ;A=0 IF KB CLEAR
JR        Z,AGN   ;BRANCH IF NO BYTE
POP       IY      ;RESTORE
POP       DE      ;  DE & IY
    
```

DISPLAY BYTE
AT CURSOR

```

PUSH      DE      ;MUST SAVE
PUSH      IY      ;  DE & IY
LD        A,20H   ;BYTE TO DISPLAY
CALL      33H     ;DISPLAY
POP       IY      ;RESTORE
POP       DE      ;  DE & IY
    
```

DEFINE DRIVE

;A-REGISTER SPECIFIES CASSETTE (0 OR 1)

```

LD        A,0     ;ON BOARD CASSETTE
CALL      0212H   ;DEFINE DRIVE
    
```

WRITE LEADER
AND SYNC BYTE

```
CALL 0278H
```

TURN OFF
CASSETTE

```
CALL 01F8H
```

SAVE MEMORY
TO CASSETTE
User must CALL 264H often enough
to keep up with 500 baud. Timing is
automatic.

```

LD        A,0     ;ON BOARD CASSETTE
CALL      0212H   ;DEFINE DRIVE
CALL      0287H   ;WRITE LEADER
LD        A,20H   ;BYTE TO RECORD
CALL      0264H   ;OUTPUT BYTE

CALL      01F8H   ;CASSETTE OFF
    
```

LOOK FOR LEADER
AND SYNC BYTE

```
CALL 0296H
```

LOAD MEMORY
FROM CASSETTE

Your program must CALL 0235H often
enough to keep up with 500 baud, and
must do its own checksum if desired.
A-register contains byte read. The user
must turn off the Cassette (CALL 01F8H)
when all bytes have been read.

```

LD        A,0     ;DEFINE DRIVE
CALL      0212H   ;DEFINE DRIVE
CALL      0296H   ;FIND SYNC BYTE
CALL      0235H   ;READ ONE BYTE
    
```

RETURN TO
LEVEL II BASIC

Press	RESET	
JP	0	;LIKE POWER UP
JP	1A19H	;RE-ENTRY

RETURN TO TBUG
UNDER LEVEL II BASIC

Set	Set	Set a Breakpoint to next opcode address.	
	JP	43A0H	;RE-ENTER TBUG

OUTPUT TO LINE PRINTER
(LEVEL II ONLY)

			;PUT ASCII BYTE IN
			;A-REGISTER AND CALL PRTOUT
			;BUSY CONDITION TESTED FOR
			;SAVE REGS.
PRTOUT	EXX		
	LD	HL,37E8H	;LOAD LP POINTER IN HL
PRTL8	LD	D,(HL)	;LOAD LP STATUS BYTE
	BIT	7,D	;IS THE PRINTER BUSY?
	JP	NZ,PRTL8	
	LD	(HL),A	;OUTPUT BYTE TO PRINTER
	EXX		
	RET		