

INTRODUCTION

Vernon B. Hester distributes all software on an "AS-IS" basis without warranty.

Vernon B. Hester shall not be liable and/or responsible to the purchaser with respect to liability, loss, and/or damage caused and/or alleged to be caused directly or indirectly by the use of this software, that includes but is not limited to any interruption of service, loss of business, and/or anticipatory profits, and/or consequential damage resulting from use of this software.

ZEUS is copyrighted with all rights reserved. Copying, duplicating, selling, or any unauthorized distributing of this product is expressly forbidden. In accepting this product, the purchaser recognizes and accepts this agreement.

ZEUS

Copyright (c) 1983, 1984 by Cosmopolitan Electronics Corporation.
Copyright (c) 1990, 1997, 1998, 2000, and 2004 by Vernon B. Hester

REFERENCE MANUAL

Copyright (c) 1983, 1984 by Cosmopolitan Electronics Corporation.
Copyright (c) 1991, 1997, 1998, 2000, 2002, and 2010 by Vernon B. Hester

First printing August 1983
Second printing December 1983
Third printing April 1984
Fourth printing September 1991
Fifth printing April 1997
Sixth printing January 1998
Seventh printing April 2000
Eight printing April 2002
Ninth printing July 2010

Z80[®] is a Registered Trademark of Zilog, Inc.

INTRODUCTION

ZEUS Editor/Assembler

ZEUS is distributed on a 40-cylinder single-sided 5¼" data diskette, and is available for ESOTERIC; Model 4 MULTIDOS; TRSDOS/LS-DOS 6.x.x, DOSPLUS IV; and all MODEL I and Model III operating systems (four compositions). You must specify the operating system you are planning to use with ZEUS. ZEUS is also available on a double-sided 3½" diskette with ESOTERIC, Model I, Model III, Max-80, and Model 4 MULTIDOS operating systems.

ZEUS

The files on the ZEUS diskette are:

```
ZEUS/CMD -- Editor/Assembler.  
Z80Z/ASM -- Source code of 1136 Z80® instructions in object code order.
```

ZEUS is executed by entering from the DOS command level.

```
ZEUS[ PROG]<ENTER>
```

PROG/ASM is the source filename, and PROG inclusion is optional. If PROG is entered in the command prompt, then PROG/ASM loads after ZEUS initialization. The load modifiers, #, *, and % must precede the filename. e.g., ZEUS #PROG for the file PROG/ASM source file in ASCII format.

The main power of ZEUS is its operating speed. ZEUS is the fastest full-featured editor/assembler in the universe. ZEUS obtains its speed by partially assembling each line as it is entered and keeping the label table intact as long as the text is not changed (if you are familiar with BASIC, the label table is very much like variables assigned during program execution).

ZEUS can create source code in four formats:

1. ZEUS format
2. ASCII format
3. EDTASM format
4. the modified EDTASM format used by NEWDOS/80.

There are no special conversion programs required to accomplish the different formats. You control the conversion when you save the file.

EXAMPLES:

```
S filespec      {ZEUS format}  
S # filespec    {ASCII format}  
S * filespec    {EDTASM format}  
S % filespec    {Modified EDTASM format used by NEWDOS/80}
```

Inversely ZEUS can load files in any of these four formats.

EXAMPLES:

```
L filespec      {ZEUS format}  
L # filespec    {ASCII format}  
L * filespec    {EDTASM format}  
L % filespec    {Modified EDTASM format used by NEWDOS/80}
```

INTRODUCTION

FILENAMES

ZEUS appends **/ASM** to all prompted source filenames, and **/CMD** to all object filenames that do not include an extension. Filenames defined with the **GET** instruction must have the full filename (and must be in the correct case) because the **GET** instruction is not prompted for a source file. e.g., GET BAS8/ASM. To access a prompted file without an extension, append a / immediately behind the filename.

EXAMPLES:

<u>Your input</u>	<u>Filespec given to DOS</u>
DOGGY	DOGGY/ASM
KOOL.MAN	KOOL/ASM.MAN
COLUMNS:2	COLUMNS/ASM:2
TABLES.MONKEY:0	TABLES/ASM.MONKEY:0
MOVIE/INC	MOVIE/INC
VIDEO/SOR.CAR	VIDEO/SOR.CAR
KILLER/GET:2	KILLER/GET:2
OVERMAKE/SRC.FUNNY:1	OVERMAKE/SRC.FUNNY:1
SEXY/	SEXY
MUCHO/.GIRL	MUCHO.GIRL
WOMAN/:3	WOMAN:3
COLDHEAT/.SUN:3	COLDHEAT.SUN:3

Filenames prompted and answered by you are remembered with default/non-default extensions. If you key in "FAST" to a load filename response, then "FAST/ASM" is the default filename for <L>oads, <S>aves, and <K>ills; with "FAST/CMD" as the default object filename.

<u>Your input</u>	<u>Source filename</u>	<u>Object filename</u>
DOG	DOG/ASM	DOG/CMD
DOG/SOR	DOG/SOR	DOG/CMD
DOG/	DOG	DOG/CMD

Your input of an object filename is retained without affecting the source default filename.

EXAMPLE:

```
L SAM {SAM/ASM is the default source filename}
ANO<ENTER>
Current Filespec: SAM/CMD
Filespec: MARY/CMD<ENTER>
```

The default source filename remains as SAM/ASM, whereas MARY/CMD is the default object filename.

INTRODUCTION

ZEUS I/O

Throughout this manual, the words `print` and `LPrint` are used to designate ZEUS screen ASCII and printer ASCII output respectively.

`Print` is used whenever output is directed to the device in the video DCB (the contents of 401DH for the MODEL I/III and all models of MULTIDOS). Unless this device is routed elsewhere, the output goes to the display.

`LPrint` is used whenever output is directed to the device in the printer DCB, (the contents 4025H for the MODEL I/III and all models of MULTIDOS). Unless this device is routed elsewhere, the output goes to the printer.

ZEUS uses 256 as the logical record length for all files.

For MODEL I/III and all models of MULTIDOS, disk I/O is performed by using the calls at 0013H and 001BH for byte I/O, and the DOS vectors at 4436H and 4439H for sector I/O. For MODEL 4 TRSDOS/LS-DOS 6.x.x and DOSPLUS IV, disk I/O is performed by using the appropriate supervisor calls (3, 4, 67, or 75).

ZEUS PROCESS

ZEUS is a two-pass assembler. The first pass creates the label table, and establishes the values for **DEFS**, **DS**, **END**, **EQU**, and **ORG** pseudo-ops. If there are no first pass errors, then the first pass complete status byte is set, and ZEUS continues with the second pass. The second pass outputs in accordance with the selected options: **H**, **N** or **S**, then prints the total errors. When the second pass is complete, the second pass status byte is set. Additional **A** commands with the second pass status byte set is instant assembly of the source code in the text buffer. This feature enables you to assemble with **AN**, to check for assembly errors. If there are no errors, proceed with **ANO** for the creation of an object file. Top speed is obtained if most of the code is in the text buffer. The descriptions of the ZEUS commands start in section 3.

INTRODUCTION

REFERENCE MANUAL NOTATION

< > Carats enclose a key to be pressed. The "<" and ">" are not keyed in. e.g., <ENTER> = Press the "ENTER" key, <BREAK> = Press the "BREAK" key, <N> = Press the "N" key

{ } Braces enclose manual comments. The braces "{and the enclosed comment}" are not keyed in. e.g., D 23/47 {deletes lines 23 to 47 inclusive}

[] Brackets enclose optional parameters. The brackets "[" and "]" are not keyed in. i.e., A[H]<ENTER> can be A<ENTER> or AH<ENTER>

Ellipses represent the repetition of an optional parameter. e.g., [,EXP][,EXP][,...]

RELNUM RELNUM is a parameter that can be either a number that represents a relative line in the text buffer, or a LABEL that indirectly represents a relative line in the text buffer where LABEL is in the label field of the relative line.

The pound sign is a RELNUM parameter for line 1.

* The asterisk is a RELNUM parameter for the highest line.

. The period is a RELNUM parameter for the current line.

< Less than modifier for RELNUM. (One number less). e.g., <354 is a RELNUM parameter of 353

GLOSSARY

ASCII format: Alphanumeric source code stored without line numbers that usually uses the tab character, 09H, instead of spaces, 20H, to separate the fields. Most editors usually can handle this format. You can use a word processor to edit source code stored in ASCII format; however, the word processor probably uses spaces instead of tabs to separate the fields. ZEUS can load, **L#**, and save, **S#**, source code stored in ASCII format. In addition, ZEUS deletes the line-feed character, 0AH, if a line is terminated with a carriage-return line-feed pair, 0DH/0AH.

Current line: The last printed, edited, or entered text buffer line.

EDTASM format: Source code stored with a header byte, D3H, followed by a six character name, alphanumeric text with each line preceded by a five digit line number (the high bit is set on each digit), and a terminator byte, 1AH. This is the format used by the tape version of the Radio Shack's Editor/Assembler, and the first implementation on NEWDOS/21. The modified version of EDTASM, used by NEWDOS/80, does not have the header byte and the six-character filename. ZEUS can load, **L***, and save, **S***, source code stored in EDTASM format as well as load, **L%**, and save, **S%**, source code stored in the modified EDTASM format used by NEWDOS/80.

LABEL: A string used to represent a value symbolically (ZEUS enables labels to be from 1 to 127 characters in length; however, a label of 127 characters cannot be referenced because the maximum line length is also 127 characters).

START XOR A

INTRODUCTION

Mnemonic: An assembly language instruction. OPCODE [OPERAND]

```
CALL    C,INIT
```

Object code: Binary code directly executable by the microprocessor.

Operator: A single symbol describing an operation to be performed.

ZEUS operators are:

```
+  ADDITION
-  SUBTRACTION
*  MULTIPLICATION
/  DIVISION
!  Logical OR
#  Logical XOR
&  Logical AND
%  MODULO
<  SHIFT
```

```
LD      HL,MUCH+MORE    {The + operator}
```

Pseudo-op: Special orders you give to the assembler. Pseudo-ops do not generate object code. However, some pseudo-ops tell ZEUS how to translate the operand field into object code.

```
                COMM    'This is a comment
VIDEO          EQU     33H
                ORG     5200H
```

Relative address: The value for the beginning of the current instruction.

```
78A2  DDCBDD4E  00623      BIT     1, (IX-23H)      {relative address 78A2H}
78A6  CA3B56    00624      JP     Z,GOUSA      {relative address 78A6H}
```

Source code: The symbolic representation of LABELS and mnemonics, that the assembler translates into specific object code.

```
LABEL  OPCODE [OPERAND]
        OPCODE
```

Text buffer: The RAM resident storage area for source code.

INTRODUCTION

A specific format is required to enable ZEUS to translate source code into object code. Fields are used to distinguish the various zones in a source text line for the symbolic representation of your program, and comments. A source text line can be blank, contain only a label, or contain only a semicolon. The maximum source text line length is 127 characters.

The **LABEL** field, optional, contains a label with a value equivalent to the relative address of, or by, the instruction that follows.

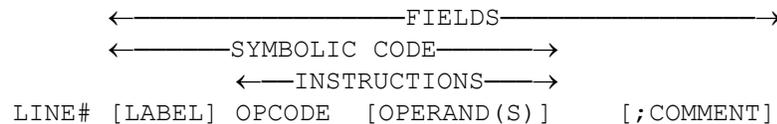
The **OPCODE** field contains a **Z80[®]** opcode, or a ZEUS pseudo-op.

The **OPERAND** field contains operands. An operand is one or two groups of characters separated by a comma.

The **INSTRUCTION** field is the "mnemonic", opcode and operand consecution that is translated into object code by the assembler.

The **COMMENT** field, optional, contains remarks.

SOURCE TEXT LINE FORMAT:



LINE#: The line number indicates the relative line in the text buffer. ZEUS dynamically renumbers the text 1, 2, 3, 4, etc. whenever text is added, deleted, or moved. When ZEUS loads an EDTASM file the line numbers in the EDTASM file are ignored, and the resulting text is numbered 1, 2, 3, 4, etc. Also, ZEUS does not store the line numbers in the text buffer.

LABEL: Labels are a string of characters with every character and character case significant. To distinguish a label from a numeric constant, the first character in a label must be a **@**, **A** through **Z**, **[**, ****, **]**, **^**, **_**, **a** through **z**, **{**, **|**, **}**, or **~**. The balance of the label can also have **0** through **9**, **:**, **=**, **>**, or **?**.

COMMENT: A comment begins with a semicolon, and overrides any field.

```
00348 ;DETERMINE IF ERROR
00349 CALC      CALL      CALC0
00350          JP        C,CALC2          ;NO ERROR IF CARRY
00351          RET
```

Line 348 the comment overrides the label field, opcode field, and operand field.

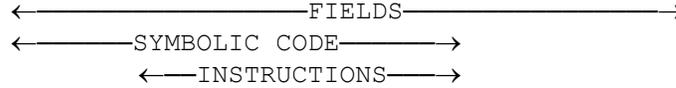
Line 349 has a label: CALC; an opcode: CALL; and an operand: CALC0.

Line 350 has an opcode: JP; operands: C,CALC2; and a comment: NO ERROR IF CARRY.

line 351 only has an opcode: RET

INTRODUCTION

During assembly printing, two additional zones are added.



ADDR **OBJCODE** LINE# [LABEL] OPCODE [OPERAND(S)] [;COMMENT]

ADDR: The ADDR zone contains a hexadecimal number that is either 1) the loading address for the ORG pseudo-op, 2) the relative address of the current instruction, 3) the label value for the EQU and DEFL pseudo-ops, 4) the length of the space for the DEFS pseudo-op, or 5) the transfer address for the END pseudo-op.

OBJCODE: The OBJCODE zone contains the hexadecimal object code translated from the mnemonics.

```
          00001 ;CDIR
          00002      COMM      'CLEARDIR 0985~*      NOTICE  **(c)19
85      ** Cosmopolitan **
          00003      COMM      '      Electronics ** Corporation. **NOTIC
E      *****
0000      00004 @FIG      EQU      0
0001      00005 @TYP      EQU      1
0002      00006 @NOH      EQU      2
0008      00007 @SPH      EQU      8
0009      00008 @DFS      EQU      9
402D      00009 TODOS      EQU      402DH
4409      00010 ERROR      EQU      4409H
.... {more source text}
5200      00020      ORG      5200H
.... {more source text}
52EA 010000      00028 START  LD      BC,0
52ED 2B      00029      DEC      HL
52EE 7E      00030      LD      A,(HL)
52EF FE2C      00031      CP      ', '
52F1 281C      00032      JR      Z,DFLT
52F3 D7      00033      RST      10H
52F4 3803      00034      JR      C,NUM
52F6 2017      00035      JR      NZ,DFLT
.... {more source text}
52EA      00192 X      END      START
```

Line 1 is a comment line: no object code is produced.
Lines 2 and 3 use the COMM pseudo-op that does not create object code.
Lines 4 - 10 use the EQU pseudo-op. The ADDR zone contains the label values. The EQU pseudo-op does not produce object code.
Line 20 uses the ORG pseudo-op. The ADDR zone contains the ORG value.
Lines 28 - 35 have object code. The OBJCODE zone contains the object code equivalent of the OPCODE and OPERAND(S).
Line 192 has the transfer address in the ADDR zone.

INTRODUCTION

CONSTANT: A constant is a numeric representation of a value. To distinguish a numeric constant from a label, the first character must be numeric or the constant must be enclosed in single quotes. Decimal constants use the digits **0** through **9**, and are optionally suffixed with the letter **D**. Binary constants use the digits **0** and **1**, and are suffixed with the letter **B**. Hexadecimal constants use the digits **0** through **9**, the letters **A** through **F**, and are suffixed with the letter **H**. ASCII constants use the entire ASCII set and are enclosed in single quotes. The single character dollar sign, **\$**, is reserved to use as the value of the relative address for the current instruction in which it appears.

EXAMPLES:

45666	decimal constant
56eeh	hexadecimal constant
0101101B	binary constant
FFEDH	label {the first character is not numeric}
0FFEDH	hexadecimal constant
'X'	ASCII constant
23145d	decimal constant
'This is'	ASCII constant
\$	relative address constant

5310	4F	00048	DRVE	LD	C,A	{no constants}
5311	CDDA44	00049		CALL	GTDC	{no constants}
5314	D7	00050		RST	10H	{10H is a constant}
5315	FE27	00051		CP	27H	{27H is a constant}
5317	2008	00052		JR	NZ,ONES	{no constants}
5319	FDCB006E	00053		BIT	5, (IY+@FIG)	{5 is a constant}
531D	28E2	00054		JR	Z, IDN	{no constants}
531F	CBE1	00055		SET	4,C	{4 is a constant}
5321	CDCD53	00056	ONES	CALL	ALIVE	{no constants}
5384	7D	00107	EON	LD	A,L	{no constants}
5385	80	00108		ADD	A,B	{no constants}
5386	6F	00109		LD	L,A	{no constants}
5387	7E	00110		LD	A, (HL)	{no constants}
5388	FE20	00111		CP	' '	{' ' is a constant}
538A	2802	00112		JR	Z,EOX	{no constants}
538C	3E2F	00113		LD	A, '/'	{'/' is a constant}
53DB	B7	00152		OR	A	{no constants}
53DC	F8	00153		RET	M	{no constants}
53DD	CB67	00154		BIT	4,A	{4 is a constant}
53DF	2017	00155		JR	NZ,ALIVD	{no constants}
53E1	3ED0	00156		LD	A,0D0H	{0D0H is a constant}
53E3	CD9744	00157		CALL	LWAIT	{no constants}
53E6	CDC944	00158		CALL	MOTON	{no constants}
53E9	1636	00159		LD	D,36H	{36H is a constant}
53EB	CD0654	00160		CALL	ALIV4	{no constants}
53EE	200D	00161		JR	NZ,ALIV2	{no constants}

INTRODUCTION

EXPRESSION: An expression is a group of characters consisting of two or more constants and/or labels separated by an operator. Expressions are evaluated (up to 16 bits) in a left to right order, maintaining an interim result after processing a constant or label.

The nine ZEUS operators are:

- + ADDITION** Adds the value following the operator to the interim result establishing the sum as the new interim result.
- SUBTRACTION** Subtracts the value following the operator from the interim result establishing the difference as the new interim result.
- * MULTIPLICATION** Multiplies the value following the operator to the interim result establishing the product as the new interim result.
- / DIVISION** Divides the value following the operator into the interim result establishing the integer portion of the quotient as the new interim result.
- ! Logical OR** Logical OR between the value following the operator and the interim result establishing a new interim result.
- # Logical XOR** Logical XOR between the value following the operator and the interim result establishing a new interim result.
- & Logical AND** Logical AND between the value following the operator and the interim result establishing a new interim result.
- % MODULO** Divides the value following the operator into the interim result establishing the integer remainder as the new interim result.
- < SHIFT** Shifts the interim result the number of bits of the value following the operator establishing a new interim result. If the value following the operator is positive, then the shift is left, otherwise the shift is right.

<u>EXPRESSION</u>	<u>OPERATION PERFORMED</u>	<u>RESULT</u>
3EH +23H	addition	97D or 61H
456D -8EH	subtraction	314 or 13AH
1010B*56	multiplication	560D or 230H
134/19	division	7
3eh!4fh	logical or	127D or 7FH
898 #1011B	logical xor	905D or 389H
345 &122	logical and	88D or 58H
67 %17	modulo	16D or 10H
34 <2	left shift	136D or 88H
57 <-3	right shift	7
12 +3 *9+3	addition, multiplication, addition	138D or 8AH

The operator must be adjacent to the following label or constant.

PSEUDO-OPS

PSEUDO-OPS

1. **COMM** Comment.
2. **DEFB** or **DB** Define byte(s).
3. **DEFL** or **DL** Define label.
4. **DEFM** or **DM** Define string(s) and/or byte(s) (define message).
5. **DEFS** or **DS** Define space.
6. **DEFW** or **DW** Define word(s).
7. **END** Terminate assembly.
8. **ENIF** Delimit conditional assembly.
9. **ERR** Abort assembly.
10. **EQU** Set a label to a value (equate a label to a value).
11. **GET** Include source from a disk file.
12. **IF** Begin conditional assembly (see **ENIF**).
13. **LIST** Control printing.
14. **MESP** Print and LPrint message.
15. **MESV** Print message.
16. **ORG** Establish relative address (origin).
17. **PAGE** LPrint to top of form (Linefeeds or CHR\$(12) depending on FORMS).
18. **SBTL** Establish a subtitle for LPrint (see **PAGE**).
19. **SHOW** Override **N** option (See **LIST**).
20. **TITL** Establish the title for LPrint (see **PAGE**).
21. **WAIT** Pause assembly.

Pseudo-ops are entered in the instruction field of a source text line.

PSEUDO-OPS

COMM

COMM — Generate an object file comment block at the beginning of the file. **COMM** is primarily used to put a message on the first sector of an object file.

NOTE: A **COMM** instruction is only recognized if no object code has been generated.

```
Syntax:  00017          COMM  'STRING
```

STRING is a string of characters 1 to 63 characters in length, and must be preceded by a single quote. Subsequent single quotes are considered part of the comment. To create a comment greater than 63 characters, add an additional line with a **COMM** instruction immediately behind the line with the first **COMM** instruction. ZEUS appends these STRINGS into one comment.

EXAMPLES:

```
00003          COMM  'I can't stop loving you.
```

Generates the comment **I can't stop loving you.**

```
00008          COMM  '*****          My          **          best          *
00009          COMM  '*  Program!  *****
```

Generates a 78 byte comment block that, when viewed with a ZAP utility, would appear as:

```
.N*****
*      My      *
*      Best    *
*  Program!   *
*****
```

The comment block on a disk sector is preceded by two bytes. The first byte is a 05H, and the second byte is the length of the comment. In this example the second byte is 4EH, which appears as an 'N'.

The layout of this comment is as follows:

```
1st line: *****          {14 *'s plus the two bytes = 16 characters}
          *bbbbbbMybbbbbb*   {16 characters b = one space}
          *bbbbbbBestbbbbbb* {16 characters}
2nd line: *bbbProgram!bbb*   {16 characters}
          *****          {16 characters}
```

PSEUDO-OPS

DEFB

DEFB or **DB** — Generate a byte(s) at the relative address. **DEFB** is primarily used to create one-byte look-up tables.

Syntax: 01333 DEFB BVAL[,BVAL][,BVAL][,...]

BVAL is a constant, expression, or label with a value of 0 to 255. For each byte generated, the relative address is incremented by one.

EXAMPLES:

```
00422            DEFB    45
00017            DEFB    46H,63H,73H,74H
00324            DEFB    78,CAT,'U'        {CAT is defined elsewhere}
```

During assembly printing, only the first byte value is printed.

```
8923 4E            00324            DEFB    78,CAT,'U'
```

DEFL

DEFL or **DL** — Establish a value for a label that can only be redefined with another **DEFL** pseudo-op. **DEFL** is primarily used to perform a complex assembly function. An example of a complex assembly function is shown at the end of the **ENIF** pseudo-op detail. Also, the file **PASS/ASM** is an example of the **DEFL** pseudo-op usage.

Syntax: 00389 LABEL DEFL WVAL

WVAL is a constant, expression, or label. If WVAL is a label or an expression with a label, and the label has not been established, then LABEL has a value of zero on the first pass. If WVAL is "LABEL" then the value of LABEL is established as the value of the relative address.

EXAMPLES:

```
00120 TOPCO    DEFL    78BAH            {TOPCO = 78BAH}
00121 MAKE    DEFL    MAKE            {MAKE = relative address}
00122 KITTEN   DEFL    TOPCO-80H        {KITTEN = 783AH}
00123           DEFB    KITTEN&0FFH      {byte value = 3AH}
00124 KITTEN   DEFL    TOPCO+80H        {KITTEN = 793AH}
00125           DB     KITTEN<-8        {byte value = 79H}
```

PSEUDO-OPS

DEFM

DEFM or **DM** — Generate a string(s), and/or byte(s) at the relative address. **DEFM** is primarily used to insert messages into the object code.

Syntax: 00457 DEFM 'STRING'[,BVAL][,...]

Although **DEFM** contains **DEFB** in its repertoire, **DEFM** can generate object code from a string of ASCII characters enclosed in single quotes.

EXAMPLES:

```
00317           DEFM    'This is a message.'
00237           DEFM    'This is a message with a terminator.',13
01283           DEFM    28,31,'ERROR','!' +80H,10,13
```

During assembly printing, only the first byte value is printed.

```
AB77 1C           01283           DEFM    28,31,'ERROR','!' +80H,10,13
```

DEFS

DEFS or **DS** — Increment the relative address by a value that you define. **DEFS** is primarily used to retain the contents of a block of memory. The special case, **DEFS 0**, is primarily used to mark the end of a table for reference purposes.

Syntax: 00017 DEFS WVAL

WVAL is a constant; expression with labels, if any, previously defined; or previously defined label with a value between 0 and 65535 inclusive. A **DEFS** pseudo-op generates a new loading address for the object file, that when loaded, skips over WVAL bytes of RAM. Therefore, **DEFS** preserves WVAL bytes of RAM starting at the relative address.

EXAMPLES:

```
00056           DEFS    256        {leave 256 bytes of RAM untouched when
                                  the object program is loaded into RAM.}

00330 PICBEG  DB        'a','b','c','d','e','f'
00331 PICEND  DEFS    0
          ... More source text
00677           LD       B,PICEND-PICBEG {loads B with the length of the
                                          table, 6.}
```

PSEUDO-OPS

DEFW

DEFW or **DW** — Generate a word(s) at the relative address. **DEFW** is primarily used to create an address table.

Syntax: 00017 DEFW WVAL[,WVAL][,WVAL][,...]

WVAL is a constant; expression with labels, if any, previously defined; or previously defined label with a value between 0 and 65535 inclusive. For each word generated, the relative address is incremented by two.

EXAMPLES:

```
00188            DEFW    TEST    {TEST is defined elsewhere}
00376            DEFW    4488H,65123,3,12
```

During assembly printing, only the first word value is printed.

```
1228 8844        00376            DEFW    4488H,65123,3,12
```

END

END — Terminate assembly. The **END** pseudo-op is mandatory in that it directs ZEUS to ignore any source text that follows. The **END** pseudo-op in a GET file terminates assembly of the GET file and resumes assembly from the source text that has the GET instruction.

Syntax: 00017 END [TRANSFER]

TRANSFER is the address that the DOS program execution routine transfers control (hopefully) over to your program. This address has also been designated as the program execution point. TRANSFER can be a constant, expression (unusual), or a previously defined label.

EXAMPLES:

```
03122            END                    {transfer address 0000H}
02883            END    402DH        {transfer address 402DH}

00001            ORG    7000H
00002 START    LD    A, (HL)        {START = 7000H}
... More source text
02893            RET
02894            END    START        {transfer address 7000H}
```

PSEUDO-OPS

ENIF/IF

ENIF and **IF** — Conditional assembly pseudo-ops. Conditional assembly provides you with an efficient tool to create programs for a variety of conditions. Conditional assembly source text is structured as follows:

```
00017          IF      EXP
              ... Conditional source text
00043          ENIF
```

If EXP is evaluated to a non-zero value, then the conditional source text is assembled. If EXP is evaluated to a zero value, then the conditional source text is not assembled,

or

```
00017          IF      NOT,EXP
              ... Conditional source text
00043          ENIF
```

If EXP is evaluated to a zero value, then the conditional source text is assembled. If EXP is evaluated to a non-zero value, then the conditional source text is not assembled.

During an assembly printing the **IF** and **ENIF** text lines, regardless of the condition, are not printed. If the conditional source code is not assembled, then the lines that contain the conditional source text also are not printed.

Typically, you should incorporate the status for conditional assembly at the beginning of the source text.

```
00002 MOD1     DEFL    1
00003 MOD3     DEFL    1-MOD1 {MOD3 = 0}
```

Then have subsequent code test this status

```
00789          IF      MOD1
00790          LD      A, (37ECH)
00791          ENIF
00792          IF      MOD3
00793          IN      A, (0F0H)
00794          ENIF
```

EXAMPLE:

```
00001          ORG      7000H
00002 DOG      DEFL    DOG-$
00003          IF      NOT,DOG
00004          GET      EQU/ASM
00005          ENIF
```

The GET instruction on line 00003 is only executed on the first pass. The file EQU/ASM must only have EQU's!

PSEUDO-OPS

ERR

ERR — Abort assembly and display message. The **ERR** pseudo-op is usually in a conditional block of source text, set to assemble if an undesirable situation exists. **ERR** may also be used to get the relative address in a program.

Syntax: 00017 ERR 'STRING

If the line containing **ERR** is assembled, then the line number and relative address is printed followed by STRING. STRING must be preceded by a single quote.

EXAMPLE:

```
00001           ORG       7000H
          ... More source text
01782           RET
01783           ERR       'is the first free byte.
01784           END       START
```

If the relative address of line 1783 is 8A22H, then (during the first pass) an assembly would produce:

01783 8A22 is the first free byte.

EQU

EQU — Explicitly establish the value for a label. The **EQU** pseudo-op is primarily used to make references to constants using labels.

Syntax: 00017 LABEL EQU WVAL

WVAL is a constant; expression with labels, if any, previously defined; or previously defined label with a value between 0 and 65535 inclusive.

EXAMPLE:

```
00002 KEYBRD EQU       2BH
00003 VIDEO  EQU       33H
00004 PRINT  EQU       3BH
00005 LINEIN EQU       40H
00006 KBWAIT EQU       49H
          ... More source text
00345 VLINE  LD       A, (HL)
00346           CALL   VIDEO
00347           INC     HL
00348           CP      0DH
00349           JR      NZ, VLINE
00350           RET
```

PSEUDO-OPS

GET

GET — Includes source text from disk file. The **GET** pseudo-op is used to assemble large amounts of source text by reading the disk file as source text instead of the resident text buffer.

Syntax: 00017 GET FILESPEC

If FILESPEC is not found, ZEUS prompts you to insert the diskette with FILESPEC by printing the message:

Mount the disk with FILESPEC
Press <ENTER> or <SPACE> to continue.

Press <ENTER> or <SPACE> after the proper diskette is mounted, or press <BREAK> to abort assembly.

An **END** instruction is not necessary in a **GET** file. If an **END** instruction is found in a **GET** file or if the end of file is encountered, then assembly resumes in the source text that has the **GET** instruction.

574 bytes of RAM are dynamically allocated for each concurrent opened **GET** file. The limit of nesting **GET** files is dependent on the amount of RAM space available.

During assembly printing, the text line with the **GET** instruction is not printed, unless the **G** option of the **A** command is used. The **G** option directs ZEUS to bypass the function of a **GET** pseudo-ops and print the line with the **GET** pseudo-op during assembly printing.

LIST/SHOW

LIST and **SHOW** — Control assembly printing. The **LIST** pseudo-op is primarily used during program development to limit assembly printing to pertinent parts of the assembled code.

Syntax: 00017 LIST OFF

LIST OFF disables assembly printing, starting with this line,

or

Syntax: 00033 LIST ON

LIST ON enables assembly printing with the following line, provided the **N** option is not entered under the **A** command.

The pseudo-op **SHOW** overrides the **N** option under the **A** command, and enables the **LIST OFF** and **LIST ON** pseudo-ops.

PSEUDO-OPS

MESP

MESP — This pseudo-op prints a message each pass the **MESP** pseudo-op is encountered. If the **H** option is entered under the **A** command, then the message also is LPrinted.

Syntax: 03466 MESP 'See here!

Each pass line 3466 is assembled, the message **See here!** is printed. The message **See here!** is also LPrinted if the **H** option is entered under the **A** command. The message must be preceded with one single quote. Subsequent single quotes are printed.

During assembly printing, the line with the pseudo-op **MESP** is not printed.

MESV

MESV — This pseudo-op prints a message each pass the **MESV** pseudo-op is encountered. The message is not LPrinted even if the **H** option is entered under the **A** command.

Syntax: 00882 MESV 'See here!

Each pass line 882 is assembled, the message **See here!** is printed. The message must be preceded with one single quote. Subsequent single quotes are printed.

During assembly printing, the line with the pseudo-op **MESV** is not printed.

ORG

ORG — Establish a new value for relative address. The **ORG** pseudo-op instructs ZEUS to start a new load address.

Syntax: 00001 ORG WVAL

WVAL is a constant; expression with labels, if any, previously defined; or previously defined label between the values of 0 to 65535 inclusive.

The source text may have as many ORGs as you require.

PSEUDO-OPS

PAGE/SBTL/TITL

PAGE, **SBTL**, and **TITL** — Assembly LPrinting pseudo-ops. These pseudo-ops dress up the assembly LPrinting by placing an optional title and/or subtitle on each page. The **PAGE** pseudo-op generates the code(s) necessary to get to the top of the next page. Double paging is suppressed.

```
Syntax:  00023          TITL   'STRING TITLE
          00024          SBTL   'STRING SUBTITLE
          00876          PAGE
```

TITL is used to place a title, **STRING TITLE**, on the first line on each page of an assembly LPrinting. Only the first title is LPrinted, subsequent titles are ignored. The subtitle, **STRING SUBTITLE** is LPrinted on the line after title. New subtitles may be inserted in the source text, but only appears at the next page break. You may use **TITL** and **SBTL** independent of each other. That is you may have a **TITL** without **SBTLs**, and you may have **SBTLs** without a **TITL**.

EXAMPLE:

```
00023          SBTL   'Screen formatting routines.
... More source text
00317          SBTL   'The DOG is MAN's best friend.
00318          PAGE
```

The original subtitle, **Screen formatting routines.**, is printed immediately after a title string, if a **TITL** pseudo-op exists. When ZEUS processes lines 00317 and 00318, the pseudo-op **PAGE** instructs FORMS to generate the necessary code to get to the next top of form, then establish **The DOG is MAN's best friend.** as the new subtitle, printing this subtitle on the new page.

If a **TITL** or **SBTL** is encountered in a GET file, ZEUS limits the **TITL** and/or **SBTL** to a maximum of 63 characters.

During assembly printing, text lines with **TITL**, **SBTL** or **PAGE** are not printed.

The assembly LPrinting LPrints the page number followed by the date and time the page starts printing, regardless of the existence of a **TITL** or **SBTL**, and inserts one blank line prior to the source text being LPrinted. A **TITL** and/or **SBTL** LPrints prior to the page number line.

STRING TITLE STRING SUBTITLE

Page 1, 02/06/86 20:07:50

```
00001 ;CDIR
00002 COMM 'CLEARDIR 0985~*NOTICE ** (c) 1 985 **
Cosmopolitan **
00003 COMM ' Electronics ** Corporation. * * NOTI
CE *****
```

PSEUDO-OPS

WAIT

WAIT — Suspends assembly. The **WAIT** pseudo-op suspends assembly and waits for you to press <ENTER> or <SPACE> to continue.

Syntax: 00237 **WAIT**

When a line containing the **WAIT** pseudo-op is assembled, assembly is suspended and the message

"Press <ENTER> or <SPACE> to continue."

is printed. If <BREAK> is pressed, ZEUS aborts assembly and returns you to the command mode. During assembly printing, the text line with the **WAIT** pseudo-op is not printed.

The **WAIT** pseudo-op suspends assembly each pass it is encountered.

EXAMPLE:

```
00345             WAIT
00346             GET     LIB1/ASM
```

You can now mount the diskette with LIB1/ASM at line 345, then press <ENTER> or <SPACE> to continue with line 346. Keep in mind, if you are creating an object file, the destination disk for the object file must remain mounted.

You can use the same drive for an EQUates and the object file diskette by using conditional assembly to read the EQUates diskette on the first pass only.

```
00001 ;SUPERDUPER
00002 MODEL   EQU     2             ;0=MOD1, 1=MOD3, 3=MOD4
00003           ORG     5200H
00004 LEQUS   DL     LEQUS-$
00005           IF     NOT,LEQUS
00006           IF     NOT,MODEL         ;MOD1
00007           MESV   'Mount the MODEL 1 EQU diskette.
00008           WAIT
00009           GET     EQUATES1/ASM
00010           ENIF
00011           IF     NOT,MODEL-1       ;MOD3
00012           MESV   'Mount the MODEL 3 EQU diskette.
00013           WAIT
00014           GET     EQUATES3/ASM
00015           ENIF
00016           IF     NOT,MODEL-3       ;MOD4
00017           MESV   'Mount the MODEL 4 EQU diskette.
00018           WAIT
00019           GET     EQUATES4/ASM
00020           ENIF
00021           ENIF
00022           IF     LEQUS
00023           MESV   'Mount the Object diskette.
00024           WAIT
00025           ENIF
```

PSEUDO-OPS

In several examples presented, the code lines similar to:

```
00002 DOG      DEFL   DOG-$
00003          IF     NOT,DOG
... more source text
```

or

```
00004 LEQUS   DL      LEQUS-$
00005          IF     NOT,LEQUS
... more source text
```

work as intended when the relative address is not zero. e.g.,

```
00001          ORG    0
00002 DOG      DEFL   DOG-$
00003          IF     NOT,DOG
... more source text
```

When the relative address is zero, then these examples do not work. Similarly, when the address is 8000H, every other assembly command causes the conditional code to assemble.

The best way to design your code would be along the lines of the following:

```
00001          ORG    8000H
00002 DOG      DEFL   VIDEO {EQUated in the file EQUATES/ASM}
00003          IF     NOT,DOG
00004          GET   EQUATES/ASM
00005          ENIF
... more source text
01255          END
```

Regardless of the relative address when line 2 is processed, this method always works provided the DEFL label (DOG in this example) gets EQUated in the GET file (see line 3 in the following code sequence).

```
00001;EQUATES/ASM
00002 KEYBRD  EQU     002BH
00003 VIDEO  EQU     003BH
00004 PRINT  EQU     003BH
... more source text
```

In summary, conditional EQUate GETting should use labels EQUated in the GET file as the control for the conditional assembly instead of using the quick and dirty method of:

```
00005 LABEL  EQU     LABEL-$.
00006          IF     NOT,LABEL
00007          GET   EQUATES/ASM
```

COMMANDS

SINGLE KEYSTROKE COMMANDS

PAUSE COMMAND

CURSOR MOVES

SINGLE LETTER COMMANDS:

A	Assemble.
B	Calculator.
C	Global change, and case conversion.
D	Delete line(s).
E	Edit line.
F	Find label.
G	LPrint format. LPrint is used to indicate output to the device in the printer DCB.
H	LPrint line(s). LPrint is used to indicate output to the device in the printer DCB.
I	Insert.
J	LPrint raw data. LPrint is used to indicate output to the device in the printer DCB.
K	Remove file.
L	Load file.
M	Move/Duplicate line(s).
NC	Remove comments.
NE	Reset buffer.
O	Opcode/Operand reference.
P	Print line(s). Print is used to indicate output to the device in the video DCB.
QU	Exit.
R	Reference.
S	Save file.
T	Print label table. Print is used to indicate output to the device in the video DCB.
U	Usage.
V	View directory.
X	Recover text on reentry.

COMMANDS

SINGLE KEYSTROKE COMMANDS

Single keystroke commands are recognized only in the first position.

<↑>	Prints next lower line, if exists, otherwise prints line 1 (first line)
<↓>	Prints next higher line, if exists, otherwise does nothing.
<ENTER>	Prints page from current line.
<SHIFT·ENTER>	Prints page to current line, first press. Subsequent presses, prints page to current line less 14 (16 line video format) or 22 (24 line video format).
<SHIFT·↑>	Prints line 1 (first line).
<SHIFT·↓>	Prints the highest line.
. {period}	Prints current line.
, {comma}	Edits current line.
<CLEAR>	Clears display.

PAUSE COMMAND

<SPACE>	<SPACE> pauses printing, LPrinting or changes under the change command. After a pause, you can single step with <SPACE>, continue without pausing with <ENTER>, or abort with <BREAK>.
---------	--

CURSOR MOVES

<SHIFT·SPACE>	Move the cursor to the next TAB position.
<→>	Move the cursor to the next TAB position.
<←>	Backspace one position and remove character at the new cursor position.
<SHIFT·←>	Erase all characters previously keyed in, and position cursor to the first input position.

INTERPRET/ABORT

<ENTER>	<ENTER> is used to indicate completion of an input command. All commands, except the single keystroke commands, are terminated with <ENTER>.
<BREAK>	<BREAK> aborts any command and returns you to the command prompt ">".

COMMANDS

A — Assemble source code.

Syntax **A**[E] [G] [H] [J] [N] [O] [P] [Q] [S] [T[V] [nn]] [U] <ENTER>

The A command has 11 options:

- E** Wait on error. The **E** option causes ZEUS to wait whenever an error occurs. After the error message and the offending line is printed, press <SPACE> to wait on next error or <ENTER> to continue without waiting.
- G** Ignore GET instructions. The **G** option causes ZEUS to ignore all GET pseudo-ops, and prints the line with the GET pseudo-op. This is helpful while debugging a module for "Out of range!", "Expression!", "Overflow!", and "Redefinition!" errors without assembling the entire source text.
- H** LPrint link to print. The **H** option links LPrint to print.
- J** Top of form after the label table is LPrinted. The **J** option is recognized if the **T** option is specified along with the **H** or **S** option. (Please see the **P** option below.)
- N** Printing off. The **N** option inhibits assembly printing and LPrinting of non-erroneous lines, and overrides the LIST ON pseudo-op. The **N** option can be canceled with the SHOW pseudo-op.
- O** Create an object file. The **O** option directs ZEUS to write object code to a filespec you can select. With the **O** option, the prompt:

Current Filespec: (default) {If a filespec had been used}
Filespec:

is printed. <ENTER> uses the default filespec. ZEUS appends /CMD to a filename entered without an extension, then searches for the filespec. If the filespec is found, then the query

Overwrite file (Y/N)?

is printed. Enter <Y> to overwrite file, <N> to return to the filespec prompt, or <BREAK> to abort assembly. If the filespec is not found, then the query

Create file (C/N)?

is printed. Press <C> to create a new file, <N> to return to the filespec prompt, or <BREAK> to abort assembly.

- P** Top of form after the source code is LPrinted. The **P** option is recognized if the **H** or **S** option is specified. The proper use of the **J** and **P** options are as follows (TOF = top of form):

Source Code, TOF	AHP	or	ASP
Source Code, Label Table, TOF	AHTJ	or	ASTJ
Source Code, TOF, Label Table, TOF	AHTJP	or	ASTJP

COMMANDS

- Q** Quit if an error occurs on the first pass. The **Q** option terminates assembly if an error is found during the first pass. After ZEUS makes one pass (builds label table), ZEUS does not make this first pass again unless **Q** is entered as an assembly option, or a disk I/O error occurs, or a **C, D, E, I, L, M, NE,** or **QU** command is invoked. The pseudo-ops **IF, DEFS, DS, EQU, ORG,** and **END** generate first pass and second pass errors if the corresponding expressions cannot be evaluated - undefined label or expression error. The total errors include the first pass as well as the second pass errors, even if the same error is indicated on both passes. If the **A** command is executed again, then the total errors are the results of the second pass only. A repeated label in the label field (redefinition error) results in first pass errors only.
- S** LPrint assembly listing. The **S** option directs the assembly to LPrint only. (The **H** option without printing.)
- T[nn]** Print label table. The **T** option sorts the labels in ascending alphanumeric order, then prints the labels and their corresponding hexadecimal values using **nn** characters for the width of output (multiple labels wide). If the **H** or **S** option is selected and precedes the **T** option without **nn** entered, then LPrint defaults to the character width set under the **G** command. A **V** immediately after **T** causes the sort to be in label value order.
- U** Ignore "Undefined label!" errors. The **U** option directs ZEUS to ignore all undefined label errors. This option is useful when a module is assembled looking for "Out of range!", "Expression!", "Overflow!", and "Redefinition!" errors. Or if the **G** option is specified and you want to avoid "Undefined label!" errors for references to another file.

Options may be 1) in any order, 2) repeated, 3) separated by one or more spaces, 4) separated by one or more commas, 5) separated by any combination of spaces and commas, or 6) not separated at all.

EXAMPLES:

ANO is equivalent to: AON or A,O,N or A N O or A, ,O,,N,O,O,N NN

A,N,H<ENTER> {Print and LPrint error(s) and error line(s).}

A,N,S<ENTER> {LPrint error(s) and error line(s).}

AT55H<ENTER> {Print and LPrint source code and label table using 55 characters for the width of the label table.}

ATH<ENTER> {Print and LPrint source code and label table using the video width characters for the width of the label table.}

A,H,T<ENTER> {Print and LPrint source code and label table using the value of **w-i** set under the **G** command for the number of characters for the width of the label table.}

A Q E<ENTER> {Pause on each error line and terminate assembly after the first pass if any errors occurred.}

COMMANDS

B — Command mode calculator. (You may also use the question mark, ?)

Syntax **B**exp1[**operator**exp2][**operator**exp3][...]<ENTER>

Operator can be:

+	ADD.
-	SUBTRACT (also unary).
*	MULTIPLY.
/	DIVIDE.
!	OR.
#	XOR.
&	AND.
%	MODULO.
<	SHIFT LEFT.
<-	SHIFT RIGHT.

The **B** command calculates expressions in the range of 0 to 65535 decimal, returning an integer result in the range of 0000000000000000B to 1111111111111111B binary, 00000 to 65535 decimal, and 0000H to FFFFH hexadecimal.

EXAMPLES: {<ENTER> and the binary results are left out for clarity}

B 38H 00056 0038H	B 99 00099 0063H	B 56+5 00061 003DH
B 56H+5 00091 005BH	B 0A0H+23H 00195 00C3H	B 37-3 0034 0022H
B 500-45H 00431 01AFH	B 67D*33 02211 08A3H	B 0AH*0AH 00100 0064H
B 78/9 00008 0008H	B 0FE9H/23H 00116 0074H	B 3C00H/4 03840 0F00H
B 5!25 00029 001DH	B 37H!23 00055 0037H	B 0F132H!89DDH 63999 F9FFH
B 0F23H#1A44H 05479 1567H	B 89#19 00074 004AH	B 99d#78h 00027 001BH
B 5&-5 00001 0001H	B 23&5/3 00001 0001H	B 89!8&444/2 00012 000CH
B 89%7+23*6 00168 00A8H	B 89H%7 00004 0004H	B 12345%2 00001 0001H
B 89<1 00178 00B2H	B 0FFFFH<8 65280 FF00H	B 0FFFFH<16 00000 0000H
B 89<-1 00044 002CH	B 0FFFFH<-8 00255 00FFH	B 566H/23<-3 00007 0007H

COMMANDS

If the label table is set, then the labels can be used in expressions.

EXAMPLE:

```
B OSET*3           {from the file Z80Z/ASM}.
00090 005AH
```

Calculations are performed on a left to right basis. If an expression is missing or an operator is used improperly, then

Expression?

is printed, and the **B** command terminates.

EXAMPLE:

```
B 75-33*
Expression?
```

If a label is used as an expression, and this label is not found in the label table, then

Undefined label!

is printed, and the **B** command terminates.

EXAMPLE:

```
B UNCLE+3
Undefined label!
```

However, if the **U** option was selected, and the **B** command uses an undefined label, then the **B** command simply returns to the command prompt indicating no error.

If the resulting calculation exceeds 65535 decimal (FFFFH hexadecimal), then the results is a modulo 65536 answer.

EXAMPLE:

```
B300*512          {300*512=153600. 153600%65536=22528.}
0101100000000000B 22528 5800H
```

C — Global change

Syntax **C**[OPT]STR1/[STR2] [,RELNUM1[/RELNUM2]]<ENTER>

The **C** command changes STR1 to STR2 from RELNUM1 to RELNUM2 inclusive. Opcodes, and operands are not changed by the **C** command. Single quoted text and comments are untouched, unless OPT is used as specified below. If STR2 is null, then STR1 is deleted.

COMMANDS

OPT can be either a single quote, or a semicolon. If OPT is a single quote, then changes only take place in single quoted text. If OPT is a semicolon, then changes only take place in comment text.

If a requested change to a line would result in the line exceeding 127 characters in length, then the message:

Out of Memory!

is printed, and the **C** command aborts without changing that line.

The **C** command prints the line before the change occurs and again after the change has successfully taken place. The <BREAK> key may be used to terminate the **C** command, and the <SPACE> bar may be used to pause.

If RELNUM1 is not specified, then line one is used. If RELNUM2 is not specified and RELNUM1 is specified, then RELNUM2 defaults to RELNUM1 (single line change). If neither is specified, then the changes take place over the entire resident source text. RELNUMs can have a constant offset applied, e.g., RELNUM +22, DOG -3.

EXAMPLES:

C'aborted,terminated	{change aborted to terminated, in single quoted text, throughout the text buffer}
C DOG/,4/37-3	{delete DOG, from line 4 to 34 inclusive}
C CAT/	{delete CAT throughout the text buffer}
C MAN/BOY,./*	{change MAN to BOY, from the current line to the end of the text buffer}
C ABC/DEF,LOG/<PERK	{change ABC to DEF, from the line which has the label LOG in the label field, to one line less than the line which has PERK in the label field}

NOTE: The **C** command processes each changed line through the syntax check routine. If a requested change would result in an erroneous line, then the **E** command is invoked, permitting corrections to the offending line. Although the **C** command continues after the line is deemed acceptable, the outcome may not be the desired result. The **C** command can be terminated immediately after the edit, by pressing the <ENTER> and <BREAK> keys simultaneously. The **C** command can be directed to bypass the syntax checking by using an exclamation point for OPT, e.g., C!;GET/*[tab]GET

C — Case Conversion

Syntax **C"**<ENTER> convert to upper case.
 C.<ENTER> convert to lower case.

The case conversion affects only labels and constants not enclosed in single quotes. Text in single quotes, comment text and a filespec in a GET instruction are not converted. The case conversion does not switch case, it makes all letters upper case, **C"**, or all letters lower case, **C.**

COMMANDS

D — Delete specific line from the text buffer.

Syntax **D**[RELNUM1[/RELNUM2]]<ENTER>

The **D** command removes lines from RELNUM1 to RELNUM2 inclusive. If RELNUM1 is not specified, then the current line is used. If RELNUM2 is not specified, then RELNUM2 defaults to RELNUM1. If RELNUM2 is specified, then RELNUM2 must be equal to or greater than RELNUM1 else a parameter error occurs. RELNUMs can have a constant offset applied, e.g., RELNUM +22, DOG -3.

EXAMPLES: {current line = 37}

```
D/.           {delete line 37}
D             {delete line 37}
D/           {delete line 37}
D.           {delete line 37}
D./         {delete line 37}
D37         {delete line 37}
D#/.        {delete lines 1 to 37 inclusive}
D./*        {delete lines 37 to the end of the text buffer}
```

```
D43/. {results in}
. <-- parameter error {RELNUM2 cannot be less than RELNUM1}
```

```
D 35+2       {delete line 37}
```

```
D 43/22      {results in}
22 <-- parameter error
```

EXAMPLE:

```
00075        RET
00076 LOOP  LD   A, (HL)
00077      INC   HL
00078      OR   A
00079      RET   Z
00080      DJNZ LOOP
00081 DOG    LD    HL,MOKE
00082        CALL  BOTTLE
```

Lines 76 through 80 can be deleted by one of the following:

- | | |
|------------------|------------------|
| 1) D76/80 | 2) D76/<81 |
| 3) D76/<DOG | 4) D76/DOG-1 |
| 5) D<77/80 | 6) D<77/<81 |
| 7) D<77/<DOG | 8) D<77/DOG-1 |
| 9) DLOOP/80 | 10) DLOOP/<81 |
| 11) DLOOP/<DOG | 12) DLOOP/DOG-1 |
| 13) DLOOP/LOOP+4 | 14) DDOG-5/DOG-1 |

The use of labels to delimit a RELNUM range is very useful.

COMMANDS

E — Edit specific text line.

Syntax **E**[RELNUM]<ENTER>

The **E** command enables you to modify the contents of a text line. If RELNUM is not specified, then the current line is used. When the **E** command is invoked, the line is printed with a small transparent rectangular cursor at the first position, and a block at the end of the physical line. Now you are in the overstrike mode, and any non-control key replaces the character at the cursor position.

The edit control functions are implemented by pressing the <CTRL> key and one of the four keys listed below:

A	ABORT	abort changes and start editing again.
I	INSERT	shift between overstrike and insert.
D	DELETE	delete character at cursor.
H	HACK	delete all characters from cursor to end of line.

(Max-80 MULTIDOS, Model 4 MULTIDOS, or ESOTERIC): When editing a line <F1>, <F2>, and <F3> can be used in place of <CTRL·I>, <CTRL·D>, and <CTRL·H> respectively.

In the insert mode, indicated by the rectangular cursor becoming a larger rectangular cursor, any non-control character is inserted at the cursor position, and the balance of the line moved right one position. The insert mode drops the 128th character if the number of characters in a line would exceed 127.

Cursor moves:

<→>	move right one character position. Use <SHIFT·SPACE> to insert or overwrite a tab.
<←>	move left one character position.
<SHIFT·→>	move to end of the physical line.
<SHIFT·←>	move to beginning of line.

Other control keys:

<BREAK>	abort EDIT and leave the line without any changes.
<ENTER>	process the modified line through the syntax check routine. If no error is found, replace the edited line with the modified line, and exit the E command. If an error is found, the <BREAK> key is disabled, and the overstrike mode is set. The cursor is positioned at the first character in the potentially erroneous symbolic field.

If you find yourself unable to satisfy a forced edit, position the cursor at the first position and key <;> (make the line a comment), then correct the line after you have gathered your thoughts.

COMMANDS

F — Print line with the specified label.

Syntax **F**LABEL<ENTER>

The **F** command searches from line one to the end of the source text for the specified LABEL, and prints the line if the LABEL is found. An initial LABEL is mandatory with the **F** command. If the initial LABEL is null, then

Missing information!

is printed. If the LABEL does not exist in the label field, then

Line not found.

is printed.

Once a LABEL is established, the **F** command finds this LABEL with just F<ENTER>. The **F** command can be used to find a line beginning with a comment. If you know the first word in the comment, enter **F;WORD**. If the reference command, R, is used to reference a label, then @<ENTER> effects a **F label<ENTER>** command.

G — Set printer values.

Syntax **G**[L|N]t,i,w,p<ENTER>

or

Syntax **G**<ENTER> prints the current forms settings.

The **G** command sets up the LPrint characteristics. The parameters are:

L = start adding a line feed after each carriage return.

N = stop adding a line feed after each carriage return.

t = the number of printed text lines per page.

i = the number of spaces indented on each line (left margin).

w = the total line length, including the **i** parameter (width of paper).

p = the number of physical lines per page (length of paper).

NOTE: If **p** is less than **t**, then the value stored for **p** is added to **t**.

The total characters printed before ZEUS sends a carriage return is the value of **w** less the value of **i**, i.e., print (**w-i**) characters.

EXAMPLES:

G 50,0,80,52<ENTER> {LPrint 50 text lines, of 80 characters maximum on a 52 line page.

G 80,10,132,88<ENTER> {LPrint 80 text lines, of 122 characters maximum on a 88 line page. The left margin is set to 10 characters spaces}

G 60,0,80,6<ENTER> {LPrint 60 text lines, of 80 characters maximum on a 66 line page. NOTE: 66 (60+6) is stored for **p**.

COMMANDS

H — LPrint line(s).

Syntax **H**[RELNUM1 [/RELNUM2]]<ENTER>

The **H** command LPrints and prints lines from RELNUM1 to RELNUM2 inclusive. The **H** command links the line printer to the video monitor. i.e., you will see the lines displayed as they are sent to the printer. If RELNUM1 is not specified, then the current line is used. If RELNUM2 is not specified, then RELNUM2 defaults to RELNUM1. If RELNUM2 is specified, then RELNUM2 must be equal to or greater than RELNUM1 else a parameter error occurs. RELNUMs can have a constant offset applied, e.g., RELNUM +22, DOG -3.

EXAMPLES: {current line = 37}

```
H/.          {LPrint and print line 37}
H            {LPrint and print line 37}
H/          {LPrint and print line 37}
H.          {LPrint and print line 37}
H./        {LPrint and print line 37}
H#+36      {LPrint and print line 37}
H33+4      {LPrint and print line 37}
H137-100   {LPrint and print line 37}
H#/.       {LPrint and print lines 1 to 37 inclusive}
H./*      {LPrint and print lines 37 to the end of the text
           buffer}
```

```
H43/.       {results in}
. <-- parameter error {RELNUM2 cannot be less than RELNUM1}

H 43/22     {results in}
22 <-- parameter error
```

EXAMPLE:

```
00075      RET
00076 LOOP  LD    A, (HL)
00077      INC   HL
00078      OR    A
00079      RET   Z
00080      DJNZ  LOOP
00081 DOG   LD     HL,MOKE
00082      CALL   BOTTLE
```

Lines 76 through 80 can be LPrinted and printed by:

```
1) H76/80          2) H76/<81
3) H76/<DOG        4) H76/DOG-1
5) H<77/80        6) H<77/<81
7) H<77/<DOG      8) H<77/DOG-1
9) HLOOP/80       10) HLOOP/<81
11) HLOOP/<DOG    12) HLOOP/DOG-1
13) HLOOP/LOOP+4 14) HDOG-5/DOG-1
```

The use of labels to delimit a RELNUM range is very useful.

COMMANDS

I — Insert line(s).

Syntax **I**[RELNUM]<ENTER>

The **I** command is used to insert or add source code into the text buffer immediately behind RELNUM. If RELNUM is not specified then the current line is used. I0<ENTER> is used to insert a line ahead of line one in the text buffer.

When you want to create source text with an empty text buffer RELNUM is not required, i.e., I<ENTER>.

The **I** command prints a five digit line number one unit higher than RELNUM followed by a space and waits for you to key in text. Use <SHIFT-SPACE> or <-> to insert a tab. After a text line is keyed in and terminated with <ENTER>, the line is processed through the syntax check routine. If no error is found, then the line is inserted into the text buffer. If an error is found, the **I** command is suspended, the <BREAK> key disabled, and the **E** command is invoked, with the edit function in the overstrike mode. The cursor is positioned at the first character in the potentially erroneous symbolic field. After subsequent modifications have corrected the erroneous line, the **I** command resumes.

The **I** command continues to print a line number one unit higher than the previously entered line. To exit the **I** command press <BREAK>.

J— Raw data to Lprint.

Syntax **J**<ENTER> {Software top of form.}
Syntax **J**data<ENTER> {bypasses FORMS}

The use of **J**data completely bypasses ZEUS (or the DOS) FORMS and/or any RAM or DISK spooler. This function is provided to send special codes directly to the printer to control special printer functions.

EXAMPLES:

```
J27,69            J 1BH,45H            J 27,'E'            J 1BH,'E'  
J 'Z','e','u','s',13<ENTER> {LPrints "Zeus".}
```

After you have LPrinted source code, you can key J<ENTER> to position the paper to the next top of form.

During ZEUS initialization, ZEUS sends the bytes located on relative file sector zero, relative byte 80H through 94H (20d bytes) to the printer for any special printer initialization. The code string cannot have an intervening zero byte value, may be as little as one byte or as many as 20 bytes, and is terminated by a 00H byte.

COMMANDS

K — Remove filespec.

Syntax **K**[filespec]<ENTER> {auto /ASM.}

If the filespec was not entered in the command prompt, then

```
Current Filespec: (default)
Filespec:
```

is printed. Press <ENTER> for the default filespec, <BREAK> to abort, or key in another filespec then press <ENTER>.

L — Load source file.

Syntax **L**[OPT1][filespec][OPT2]<ENTER> {auto /ASM.}

```
If OPT1 = #, then load an ASCII format file.
If OPT1 = *, then load an EDTASM format file.
If OPT1 = %, then load an EDTASM format file without header.
                (Source code produced by NEWDOS/80's DISASSEM/CMD)
```

If there is source text in the buffer, then

```
Append to buffer?
```

is printed. Respond <Y>, <N>, or <BREAK>. If the filespec was not entered in the command prompt, then

```
Current Filespec: (default)
Filespec:
```

is printed. Press <ENTER> for the default filespec, <BREAK> to abort, or key in another filespec then press <ENTER>.

OPT2 is valid for loading files using OPT1. OPT2 directs ZEUS to skip over OPT2 number of lines before loading source code into the text buffer. This feature is useful when a disassembler produces source code and the source code is larger than the text buffer. If the source code is very large, then using OPT2 can segment it. If you are loading source code and you receive an "Out of memory!" error message, then use OPT2, with several hundred lines for overhead.

Let's say the "Out of memory!" error occurred after loading 2675 lines of the file ROM/ASM. An easy way to partition this source code is to delete the last 675 lines by D2001/*<ENTER>. Save the source code in the text buffer, using a different filename -- S ROM1<ENTER>. Now load the same file again using OPT2 = 2000 i.e., L%ROM 2000<ENTER>. If you receive an "OUT of memory!" error message again -- line 2736, delete lines 2001 to 2736, and save this as ROM2. You have just saved two files of 2000 lines each. Both files are numbered 1 to 2000. The next load would use OPT2 = 4000 i.e., L%ROM 4000<ENTER>.

COMMANDS

M — Move or duplicate text lines.

Syntax **M**[!][RELNUM1[/RELNUM2]],RELNUM3<ENTER>

The **M** command moves RELNUM1 to RELNUM2 inclusive, behind RELNUM3. If RELNUM1 is not specified, then the current line is used. If RELNUM2 is not specified, then RELNUM2 defaults to RELNUM1 — single line move. If RELNUM2 is specified, then RELNUM2 must be equal to or greater than RELNUM1. RELNUM3 is required and must be greater than RELNUM2 or less than RELNUM1.

RELNUMs can have a constant offset applied, e.g., RELNUM +2, DOG -3.

If ! is specified, then the lines in the range of RELNUM1 to RELNUM2 inclusive are duplicated behind RELNUM3.

EXAMPLES:

M LOOP/<DOG,483	{moves the lines from LOOP to one line less than DOG behind line 483}
M 23,44	{moves RELNUM 23 behind RELNUM 44}
M 12/23,44	{moves RELNUMs 12 through 23 behind RELNUM 44}

If you attempt to move a block of text greater than the available memory, then ZEUS moves the text in multiple moves printing an M for each multiple move (the number of moves is the number of Ms printed plus one).

NC — Remove remarks/blank lines.

Syntax **NC**<ENTER>

Removes all remarks and all blank lines in the text buffer.

NE — New text buffer.

Syntax **NE**<ENTER>

Resets all pointers. The text is recoverable if you exit ZEUS then re-enter ZEUS with <X> as the first key press.

COMMANDS

O — Opcode/operand reference.

Syntax **O**[#]STR1<ENTER>

The **O** command searches from line one for the first occurrence of STR1. STR1 can be any combination of labels, tabs, opcodes, or operands.

If # is specified, then the **O** command prints the line numbers STR1 is found in and the total occurrences.

EXAMPLE:

```
OLD[tab]A, (HL)<ENTER> {locates LD    A, (HL).}
```

If STR1 is found, then subsequent **O**<ENTER>'s find the next occurrence of STR1 until
Text end.

is printed. A subsequent reference restarts at line one.

If STR1 is found, then **O**#<ENTER> prints all remaining line numbers with STR1.

P — Print line(s).

Syntax **P**[RELNUM1 [/RELNUM2]]<ENTER>

The **P** command prints text lines from RELNUM1 to RELNUM2 inclusive to the device in the video DCB. If RELNUM1 is not specified, then the current line is used. If RELNUM2 is not specified, then RELNUM2 defaults to RELNUM1. If RELNUM2 is specified, then RELNUM2 must be equal to or greater than RELNUM1 else a parameter error occurs. The **P** command is similar to the **H** command except the listing is only directed to the device in the video DCB. RELNUMs can have a constant offset applied, e.g., RELNUM +22, DOG -3. If RELNUM1 is a number then the **P** may be omitted. i.e., P23<ENTER> or 23<ENTER> prints line 23; P33+6<ENTER> or 33+6<ENTER> prints line 39.

QU — Exit ZEUS.

Syntax **QU**<ENTER>.

If ZEUS has detected a change in the source text and you haven't saved the changes then:

```
Changes to text not saved.  
Quit (Y/N)?
```

is printed. Press <BREAK> to return to the command mode, enter <Y> to exit, or <N> to save the file and exit.

COMMANDS

R — Reference.

Syntax **R**[#]STR1<ENTER>

The **R** command searches from line one for the first occurrence of STR1. STR1 is ASCII text and not opcodes or operands. The **O** command is used to find opcodes and/or operands.

If # is specified, then the **R** command prints the line numbers STR1 is found in and the total occurrences.

EXAMPLE:

R DOGGY<ENTER> {locates DOGGY.}

If STR1 is found, then subsequent R<ENTER>'s find the next occurrence of STR1 until

Text end.

is printed. A subsequent reference restarts at line one.

If STR1 is found, then R#<ENTER> prints all remaining line numbers with STR1.

If STR1 is found, @<ENTER> effects the command F STR1<ENTER>.

S — Save file in text buffer.

Syntax **S**[OPT1][filespec][OPT2]<ENTER> {auto /ASM}

If OPT1 = #, then save is in ASCII format.

If OPT1 = *, then save is in EDTASM format.

If OPT1 = %, then save is in EDTASM format without header.

OPT2 is used with OPT1 to start the save after OPT2 number of lines. If OPT1 is * or %, and OPT2 is used, then the saved text has the first line number of OPT2 + 1. e.g., S*TTONE,1400<ENTER>, would save text starting with line number 1401 and the first line number in the file TTONE/ASM would also be 1401.

If the filespec was not in the command prompt, then

Current Filespec: (default)

Filespec:

is printed. Press <ENTER> for the default filespec, <BREAK> to abort, or key in another filespec and press <ENTER>.

COMMANDS

T — Sorted label table printing.

Syntax **T**[V][nn]<ENTER> {nn = 1 to 252}

V directs the sort to be in label value order.
nn is the number of columns for the label table.
If nn = 0 or not specified, then the number of columns defaults to the video width or the printer width.

After ZEUS completes pass one, the label table is set (a disk I/O error, or the completion of a **C**, **D**, **E**, **I**, **L**, **M**, or **NE** command resets the label table). The **T** command sorts a set label table in ascending alphanumeric order, printing the label table in the format:

```
LABEL 0000        LABEL 0000        LABEL 0000        LABEL 0000        LABEL 0000
```

where LABEL represents the name of the LABEL and 0000 is the hexadecimal address of the LABEL, the last defined (DEFL, DL) value of the LABEL, or the EQUated value of the LABEL.

U — Memory usage.

Syntax **U**<ENTER>

The **U** command prints the memory usage.

EXAMPLE:

```
ZEUS Z80Z<ENTER>
AN<ENTER>
U<ENTER>
16139 source {source text usage.}
00035 table {memory for labels in label table.}
00000 GET {memory GET files used.}
30416 free {unused.}            NOTE: the amount of free space depends
on the operating system and hi-memory
usage.
```

After the label table is sorted, the U command prints the number of labels after "table"

EXAMPLE:

```
T<ENTER>
BYTE 0048    DISP 8032    OSET 001E    WORD 8BA0    Z80Z 8000
U<ENTER>
16139 source
00035 table 0005
00000 GET
30416 free
```

COMMANDS

V — View directory (Max-80 MULTIDOS, Model 4 MULTIDOS, or ESOTERIC).

Syntax V[[:]d[']]<ENTER>
 d = drive number (0 to 7)

EXAMPLE:

```
V1<ENTER>       {directory of drive 1}
```

If the directory has more files that can be printed on the display, then the V command pauses. Press <SPACE> to print another line of files or <ENTER> to print up to one more screen of files.

X — Recover source text on reentry to ZEUS.

<X> must be the first key pressed upon entry to ZEUS.

EXAMPLE:

```
ZEUS Z80Z<ENTER>  
QU<ENTER>  
ZEUS<ENTER>  
X  
01144 text lines.
```

In this example 1144 lines of source code was recovered, indicated by "01144 text lines". If you did not press <X> as the first key and wanted to recover text, exit ZEUS with QU<ENTER> then re-initialize ZEUS, and press <X>.

ZEUS is a two pass assembler. The first pass creates the label table, and establishes the values for DEFS, DS, END, EQU, and ORG pseudo-ops. If there are no first pass errors, then the first pass complete status byte is set, and ZEUS continues with the second pass. The second pass outputs in accordance with the selected options (**H**, **N** or **S**), then prints the total errors. When the second pass is complete, the second pass status byte is set. Additional **A** commands with the second pass status byte set is instant assembly of the source code in the text buffer. This feature enables you to assemble with **AN**, to check for assembly errors. If there are no errors, proceed with **ANO** for the creation of an object file. Top speed is obtained if most of the code is in the text buffer.

(Max-80 MULTIDOS, Model 4 MULTIDOS, or ESOTERIC): Function key F4 (<RIGHT-SHIFT·F1> for **MULTIDOS** or **ESOTERIC**) dynamically displays the amount of free memory in the upper right hand corner of the display with the current line displayed below the amount of free memory. Function key F5 (<RIGHT-SHIFT·F2> for **MULTIDOS** or **ESOTERIC**; (<SHIFT·F4> **Max-80 MULTIDOS**) disables display. **MULTIDOS** or **ESOTERIC**: Function key F6 (<RIGHT-SHIFT·F3>) displays the cursor position in front of the amount of free memory.

ERRORS and SOURCE TEXT

Errors encountered with ZEUS are separated into two modes: command mode and assembler mode. Assemble mode errors are divided into two categories: terminal and warning. Terminal errors print the error message, terminate assembly, and return you to the command mode. Warning errors print the error message, and continue assembly. If the O option is specified, then the resulting object code probably is not what you want.

COMMAND MODE

1. " " is not recognized!

Printed if:

- a) the first character in a command is not recognized, or
- b) an option for the **A** command is not recognized.

=<ENTER>

"=" is not recognized!

ATNK<ENTER>

"K" is not recognized!

2. <-- Parameter error.

Printed if a command's RELNUM parameter is not within proper limits.

```
00078 DOG      IN      A, (MOUSE)
00079          OR      A
00080          JR      Z, DOG
00081 CAT      LD      (HL), A
```

D CAT/DOG <ENTER>

DOG <-- parameter error.

3. Label or line not found.

Printed if:

- a) the label in the **F** command is not in the text buffer, or
- b) the target for the **R** or **O** command does not exist, or
- c) a RELNUM parameter does not exist for the **D**, **E**, **H**, **I**, **M**, or **P** command.

D DOG/COT<ENTER>

Line not found.

4. No Text

Printed if a command requires source text in the text buffer, and the text buffer is empty.

NE<ENTER>

A<ENTER>

No text

ERRORS and SOURCE TEXT

5. Out of Memory!

Printed if:

- a) the **C** command would change a line to a length greater than 127 characters, or
- b) the **E** command modifies a line to a length that will not fit in the available memory, or
- c) the **I** command is inserting a line that will not fit in the available memory, or
- d) the **M** command attempts to duplicate a block of text larger than the amount of free memory.

```
M!1/2000,2000<ENTER>
```

```
Out of memory!
```

6. Reference what?

Printed if just R or O is entered and the **R** or **O** command has not established a reference target.

```
R<ENTER>
```

```
Reference what?
```

7. Missing information!

Printed if:

- a) there has never been a label behind the F in the **F** command, or
- b) the **C** command did not have a slash indicating the termination of the target, or
- c) if the **C** command has no target.

```
F<ENTER>
```

```
Missing information!
```

```
C TARGET REPLA<ENTER>
```

```
Missing information!
```

```
C<ENTER>
```

```
Missing Information!
```

```
C/REPLA<ENTER>
```

```
Missing information!
```

ERRORS and SOURCE TEXT

8. Text end.

Printed when the **R** or **O** command has searched to the end of text.

```
R<ENTER>  
Text end.
```

9. Expression!

Printed if the **B** command has an expression with a constant greater than 65535, a constant containing a non-constant character, or an expression with a missing or extra operator.

```
B 89UI<ENTER>  
Expression!
```

10. Load errors.

The DOS error 22H, "Load file format error" is invoked if:

- a) an attempt to load a modified EDTASM file and the % option is missing, or
- b) an attempt to load an EDTASM file and the * option is missing, or
- c) an attempt to load a file using the * option and the file is not in EDTASM format.

If you attempt to load an ASCII format file, without the # option, the text buffer is not changed.

If you specify either the % or # option, and the file is not in the correct format, ZEUS invokes the edit command. To exit the forced edit and abort the load, press <SHIFT-←>, then press <;>, then press <ENTER> and <BREAK> at the same time.

ERRORS and SOURCE TEXT

ASSEMBLER

1. Unrecognized character.

TERMINAL/first pass.

Printed if an unrecognized character is encountered in the source code. Usually occurs when a GET file is in EDTASM or ASCII format. And the " " **is not recognized** message is printed after the offending line is printed.

```
*SHOW/ASM
Unrecognized character.
0017 44094958 00013 ,DE
", " is not recognized!
```

2. Expression!

WARNING/first and second pass for ORG, END, IF, DS, DEFS, and EQU.
WARNING/second pass.

Printed if:

- a) a constant is greater than 65535, or
- b) a constant contains a non-constant character, or
- c) an expression with a missing or extra operator.

```
LD      A,66G
XOR     99999
LD      (8400H+),HL
```

Expression is set to ZERO regardless of the interim results before the error.

3. No End!

TERMINAL/first pass.

Printed if the end of the RAM resident source text file is encountered before an END instruction is found. (ZEUS ignores all text in a conditional source block of text that is not assembled. i.e., You might have an END instruction but also have an IF without a corresponding ENIF.)

4. Out of memory!

TERMINAL/first or second pass.

Printed if:

- a) insufficient memory is available for the label table, or
- b) 574 bytes are not available to GET a file.

ERRORS and SOURCE TEXT

5. Out of range!

WARNING/second pass.

Printed if:

- a) the relative addressing displacement requirement is not in the range of -126 to +129 bytes from the relative address.

The relative addressing displacement is a signed two's complement number that is added to the relative address after this instruction, \$+2. The relative addressing displacement can be in the range of \$+2-128 to \$+2+127, i.e., \$-126 to \$+129.

or

- b) the indexed addressing displacement is not in the range of a signed two's complement number (-128 to +127).

If this error is encountered, ZEUS substitutes a displacement of -2, 0FEH.

Out of range!

```
A9DB 28FE      00023      JR      Z,MONKEY {MONKEY is A920H}
```

Out of range!

```
8765 FD36FE10 00987      LD      (IY+0EDH),16
```

6. Overflow!

WARNING/second pass.

Printed if a byte requirement was satisfied with a value greater than 255.

Overflow!

```
9A23 3EFD      00234      LD      A,-3
```

If the word LSB is desired use WORD&0FFH or WORD&255.

```
9A23 3EFD      00234      LD      A,-3&0FFH
```

Although this example would produce the correct object code, the warning error is printed in cases where you wanted to use a register pair and you inadvertently keyed in one-half of the register pair, e.g.,

```
LD      E,-50
or
LD      D,-50
instead of LD      DE,-50
```

ERRORS and SOURCE TEXT

7. Redefinition!

WARNING/first pass.

Printed if a label in the label field is identical to a previously encountered label in the label field. When a label is first encountered in the label field it is defined to a value.

```
00001 ORG      7800H
00002 TEST    LD      A, (HL)
00003 TEST    INC     HL
00004         OR      A
00005         JP      NZ, TEST
00006         END
```

Redefinition!

```
7801 23      00003 TEST    INC     HL
```

{Printed on the first pass only. The value for TEST is 7800H.}

ZEUS does not flag the label that is redefined; and, the **F** command finds the first and only the first occurrence of the label.

8. Undefined label!

WARNING/first pass for ORG, END, IF, DS, DEFS, and EQU.

WARNING/second pass.

Printed if a label in the operand field has not been defined.

Undefined label!

```
A971 C30000 00923      JP      DOGGY
```

Undefined label!

```
7845 210000 00344 MANUAL LD      HL, FORT+CATTLE
```

9. ENIF without IF!

TERMINAL/first pass.

Printed if an ENIF is encountered without a corresponding IF.

e.g., source text: 00478 ENIF

```
00478 73DE ENIF without IF!
```

ERRORS and SOURCE TEXT

GENERAL INFORMATION

1. If a **WARNING** error occurs while ZEUS is processing a GET file, then, before ZEUS prints the WARNING message, the name of the GET file is printed preceded by a quantity of asterisks indicating the nesting level.

**HOWLER/INC

Expression!

```
8923 CD0000    00233          CALL    DOG+
```

The file HOWLER/INC is in a GET instruction of a file that is in a GET instruction of the RAM resident source text.

2. ZEUS has no special requirements for source text lines. A source text line may be blank, contain a semicolon only, or contain a label only.

```
00023
00024
00025 ;
00026 CATER          {CATER is valued at the relative address.}
00027
00028          CALL    MAXI    {a reference to CATER is here.}
00029 ;
00030 MAKER          {same as MAKER EQU    $}
```

3. The conditional assembly implemented in ZEUS does not support alternate conditional assembly. However, certain aspects of alternate conditional assembly can be implemented with nested IF's or the use of NOT. If you want to see if an expression is equal to another expression, then you would use the format:

```
00999          IF      NOT,EXP1-EXP2
01000
```

Assembly would continue with line 1000 if EXP1 equals EXP2.

ERRORS and SOURCE TEXT

The following sequence of code aborts assembly if the program exceeds the value that MAXI is EQUated to. This example shows you how to ensure that your program does not exceed the space allocated. The ORG statement in line 2 is put there for clarity. Lines 3 and 4 are the last two lines of your source code.

```
00001 ;LIMIT/ASM
00002     ORG     07FFEH
00003     XOR     B
00004 PEND   RET           ;End of object code.
00005     LIST   OFF
00006 MAXI   EQU     07FFFH ;Last byte that can be used.
00007 SSUB   EQU     PEND<-15
00008     IF     NOT,MAXI<-15#SSUB
00009     IF     MAXI-PEND<-15
00010     ORG     $-1
00011     ERR    'Code is too long!
00012     ENIF
00013     ENIF
00014     IF     MAXI<-15#SSUB
00015     IF     SSUB
00016     ORG     $-1
00017     ERR    'Code is too long!
00018     ENIF
00019     ENIF
00020     END
```

```
00001 ;This short program is a demonstration of the way
00002 ;ZEUS can detect which PASS is being executed.
00003 ;Lines 6 - 13 is all a source file requires
00004 ;to detect which PASS is being executed.
00005 ;PASS
00006 PAS2   DL      PAS1
00007     IF     NOT,PAS2
00008     MESV   'Executing pass 1.
00009 PAS1   DL      1
00010     ENIF
00011     IF     PAS2
00012     MESV   'Executing pass 2.
00013     ENIF
00014     END
```

How about?

```
00001 PASS   DL      DIDI {something that gets equated - see line 8}
00002     IF     NOT,PASS
00003     MESV   'Executing pass 1.
00004     ENIF
00005     IF     PASS
00006     MESV   'Executing pass 2.
00007     ENIF
00008 DIDI   EQU     23
00009     END
```

ERRORS and SOURCE TEXT

TOKEN VALUES

ZEUS tokenizes opcodes and operands when you enter a source text line. The token values for some opcodes are the same for the operands.

<u>OPERAND</u>	<u>OPCODE</u>	<u>VALUE</u>	<u>OPCODE</u>	<u>VALUE</u>
B	NOP	80H	BIT	AFH
C	RLCA	81H	RES	B0H
D	RRCA	82H	SET	B1H
E	RLA	83H	ADC	B2H
H	RRA	84H	SBC	B3H
L	DAA	85H	ADD	B4H
(HL)	CPL	86H	DEC	B5H
A	SCF	87H	INC	B6H
BC	CCF	88H	CALL	B7H
DE	HALT	89H	RET	B8H
HL	DI	8AH	JP	B9H
SP	EI	8BH	RST	BAH
(BC)	EXX	8CH	IN	BBH
(DE)	NEG	8DH	OUT	BCH
AF	RLD	8EH	EX	BDH
AF'	RRD	8FH	PUSH	BEH
NZ	LDI	90H	POP	BFH
Z	CPI	91H	LD	C0H
NC	INI	92H	JR	C1H
NOT	OUTI	93H	DJNZ	C2H
PO	LDIR	94H	IM	C3H
PE	CPIR	95H	ENIF	C4H
P	INIR	96H	END	C5H
M	OTIR	97H	ORG	C6H
(C)	LDD	98H	EQU	C7H
I	CPD	99H	IF	C8H
R	IND	9AH	COMM	C9H
(SP)	OUTD	9BH	SBTL	CAH
HX	LDDR	9CH	TITL	CBH
LX	CPDR	9DH	LIST	CCH
HY	INDR	9EH	GET	CDH
LY	OTDR	9FH	PAGE	CEH
OFF	RETI	A0H	ERR	CFH
ON	RETN	A1H	DEFS	D0H
IX	AND	A2H	DS	D1H
IY	CP	A3H	DEFL	D2H
(IX)	OR	A4H	DL	D3H
(IY)	SUB	A5H	DEFM	D4H
(IX	XOR	A6H	DM	D5H
(IY	RL	A7H	DEFB	D6H
(n	RLC	A8H	DB	D7H
n	RR	A9H	DEFW	D8H
	RRC	AAH	DW	D9H
	SLA	ABH	WAIT	DAH
	SLL	ACH	SHOW	DBH
	SRA	ADH	MESV	DCH
	SRL	AEH	MESP	DDH

ERRORS and SOURCE TEXT

LINE FORMAT

The first byte is the length of the text line including the terminating carriage return. The second byte bits are defined as:

<u>BIT</u>	<u>If set</u>
7	DB, DEFB, DM, DEFM, DW or DEFW instruction.
6	The line contains a label and opcode.
5,4,3	If these bits are: 0 0 1 RST opcode 0 1 0 has one byte to evaluate 0 1 1 CB opcode 1 0 1 has two bytes to evaluate 1 1 0 has one word to evaluate 1 1 1 BIT, SET, RES opcode
2,1,0	These bits hold the length of the object code, except for DB, DEFB, DM, DEFM, DW or DEFW pseudo-ops.

The third through sixth bytes (if required) are used to hold the object code. The fourth through sixth are used if the object code is more than one byte. The balance of the line depends on the contents, and the last byte is a carriage return, 0DH.

EXAMPLES:

The ??s are initially 00, and change when the source code is assembled.

```
01874      LD      A, (HL)
is stored as: 0A 01 7E 09 C0 09 87 2C 86 0D
```

```
00237      LD      A, FAST
is stored as: 0E 12 3E ?? 09 C0 09 87 2C 46 41 53 54 0D
```

```
00588 LOOP  BIT      7, (IX+FILE)
is stored as: 17 7C DD CB ?? 46 4C 4F 4F 50 09 AF 09 37 2C A6 2B 46 49
              4C 45 29 0D
```

```
00001;COMMENT
is stored as: 0B 00 3B 43 4F 4D 4D 45 4E 54 0D
```

```
00067      CALL    33H
is stored as: 0C 33 CD ?? ?? 09 B7 09 33 33 48 0D
```

```
00892                                     {blank line}
is stored as: 03 00 0D                                     {i.e., a blank line uses 3 bytes}
```

```
00461      LD      HL, (BASBF)
is stored as: 12 33 2A ?? ?? 09 C0 09 8A 2C 28 42 41 53 42 46 29 0D
```

```
01587 POINT DW      RUP, FRM18, BEE0, PUTT
is stored as: 20 F2 ?? ?? 50 4F 49 4E 54 09 D9 09 52 55 50 2C 46 52 4D
              31 38 2C 42 45 45 30 2C 50 55 54 54 0D
```

8-BIT LOAD

LD *r,s*

BINARY:

0 1 p p p q q q

 if *r* is B, C, D, E, H, L, (HL), or A

1 1 j 1 1 1 0 1	0 1 p p p 1 1 0
-----------------	-----------------

 if *s* is HX, LX, HY, or LY

1 1 j 1 1 1 0 1	0 1 1 1 0 q q q
-----------------	-----------------

 if *r* is HX, LX, HY, or LY

HEX:

	s	B	C	D	E	H	L	(HL)	A	HX	LX	HY	LY	
qqq		000	001	010	011	100	101	110	111					
	j									0	0	1	1	
r	ppp	j												
B	000		40	41	42	43	44	45	46	47	DD44	DD45	FD44	FD45
C	001		48	49	4A	4B	4C	4D	4E	4F	DD4C	DD4D	FD4C	FD4D
D	010		50	51	52	53	54	55	56	57	DD54	DD55	FD54	FD55
E	011		58	59	5A	5B	5C	5D	5E	5F	DD5C	DD5D	FD5C	FD5D
H	100		60	61	62	63	64	65	66	67				
L	101		68	69	6A	6B	6C	6D	6E	6F				
(HL)	110		70	71	72	73	74	75		77				
A	111		78	79	7A	7B	7C	7D	7E	7F	DD7C	DD7D	FD7C	FD7D
HX		0	DD60	DD61	DD62	DD63				DD67	DD64	DD65		
LX		0	DD68	DD69	DD6A	DD6B				DD6F	DD6C	DD6D		
HY		1	FD60	FD61	FD62	FD63				FD67			FD64	FD65
LY		1	FD68	FD69	FD6A	FD6B				FD6F			FD6C	FD6D

FLAGS: No change

TIMING: If *r* is B, C, D, E, H, L, or A: 1 M cycle, 4 T-states.
 If *r* is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL) or *s* is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, HX, LX, HY, or LY: Register *r* is loaded with the contents of register *S*.
 • If *r* is (HL): The memory location of the HL register pair is loaded with the contents of register *S*.
 • If *s* is (HL): Register *r* is loaded with contents of the memory location of the HL register pair.

8-BIT LOAD

LD *r*,*n*

BINARY:

0 0 p p p 1 1 0	n n n n n n n n
-----------------	-----------------

 if *r* is B, C, D, E, H, L, (HL), or A

1 1 j 1 1 1 0 1	0 0 1 0 q 1 1 0	n n n n n n n n
-----------------	-----------------	-----------------

 if *r* is HX, LX, HY, or LY

HEX:

<i>r</i>	ppp	j	q	
B	000			06 n
C	001			0E n
D	010			16 n
E	011			1E n
H	100			26 n
L	101			2E n
(HL)	110			36 n
A	111			3E n
HX		0	0	DD26 n
LX		0	1	DD2E n
HY		1	0	FD26 n
LY		1	1	FD2E n

FLAGS: No change

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 7 (4,3) T-states.
 If *r* is HX, LX, HY, or LY: 3 M cycles, 11 (4,4,3) T-states.
 If *r* is (HL): 3 M cycles, 10 (4,3,3) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, HX, LX, HY, or LY: Register *r* is loaded with the one-byte value *n*.
 • If *r* is (HL): The memory location of the HL register pair is loaded with the one-byte value *n*.

8-BIT LOAD

LD *A*,(*rr*)

BINARY:

0	0	0	p	1	0	1	0
---	---	---	---	---	---	---	---

HEX:

<i>rr</i>	p	
BC	0	0A
DE	1	1A

FLAGS: No change

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The *A* register is loaded with the contents of the memory location of register pair *rr*.

8-BIT LOAD

LD (*rr*),A

BINARY:

0	0	0	p	0	0	1	0
---	---	---	---	---	---	---	---

HEX:

<i>rr</i>	p	
BC	0	02
DE	1	12

FLAGS: No change

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The memory location of register pair *rr* is loaded with the contents of the A register.

8-BIT LOAD

LD A,(mn)

BINARY:

0 0 1 1 1 0 1 0	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------

HEX:

3A nm

FLAGS: No change

TIMING: 4 M cycles, 13 (4,3,3,3) T-states.

DESCRIPTION: The A register is loaded with the contents of memory location *mn*.

8-BIT LOAD

LD (*mn*),A

BINARY:

0 0 1 1 0 0 1 0	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------

HEX:

32 nm

FLAGS: No change

TIMING: 4 M cycles, 13 (4,3,3,3) T-states.

DESCRIPTION: The memory location *mn* is loaded with the contents of the A register.

8-BIT LOAD

LD $r, (ir+d)$

BINARY:

1 1 q 1 1 1 0 1	0 1 p p p 1 1 0	d d d d d d d d
-----------------	-----------------	-----------------

HEX:

ir	IX	IY
q	0	1
r	ppp	
B	000	DD46 d FD46 d
C	001	DD4E d FD4E d
D	010	DD56 d FD56 d
E	011	DD5E d FD5E d
H	100	DD66 d FD66 d
L	101	DD6E d FD6E d
A	111	DD7E d FD7E d

FLAGS: No change

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: Register r is loaded with the contents of the memory location determined by the sum of the index register ir plus the displacement d . The displacement is a two's complement byte value in the range of -128 to +127.

8-BIT LOAD

LD $(ir+d),r$

BINARY:

1 1 q 1 1 1 0 1	0 1 1 1 0 p p p	d d d d d d d d
-----------------	-----------------	-----------------

HEX:

	ir	IX	IY
	q	0	1
r	ppp		
B	000	DD70 d	FD70 d
C	001	DD71 d	FD71 d
D	010	DD72 d	FD72 d
E	011	DD73 d	FD73 d
H	100	DD74 d	FD74 d
L	101	DD75 d	FD75 d
A	111	DD77 d	FD77 d

FLAGS: No change

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The memory location determined by the sum of the index register ir plus the displacement d is loaded with the contents of register r . The displacement d is a two's complement byte value in the range -128 to +127.

8-BIT LOAD

LD (*ir+d*),*n*

BINARY:

1 1 <i>q</i> 1 1 1 0 1	0 0 1 1 0 1 1 0	<i>d d d d d d d d</i>	<i>n n n n n n n n</i>
------------------------	-----------------	------------------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DD36 <i>d n</i>
IY	1	FD36 <i>d n</i>

FLAGS: No change

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The memory location determined by the sum of the index register *ir* plus the displacement *d* is loaded with the one-byte value *n*. The displacement *d* is a two's complement byte value in the range -128 to +127.

8-BIT LOAD

EX AF,AF'

BINARY:

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

HEX:

08

FLAGS: The contents of flag register F'.

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: The contents of the main AF register pair is exchanged with the contents of the alternate AF register pair, AF'.

8-BIT LOAD

LD *sr*,A

BINARY:

1 1 1 0 1 1 0 1	0 1 0 0 p 1 1 1
-----------------	-----------------

HEX:

<i>sr</i>	<i>p</i>	
I	0	ED47
R	1	ED4F

FLAGS: No change

TIMING: 2 M cycles, 9 (4,5) T-states.

DESCRIPTION: The special purpose register ***sr*** is loaded with the contents of the **A** register. **I** is the interrupt vector register, and **R** is the memory refresh register.

8-BIT LOAD

LD *A, sr*

BINARY:

1 1 1 0 1 1 0 1	0 1 0 1 p 1 1 1
-----------------	-----------------

HEX:

<i>sr</i>	<i>p</i>	
I	0	ED57
R	1	ED5F

FLAGS:

- S Set if bit 7 of *sr* is set, else reset (Note: bit 7 of the **R** register can only be set with a **LD R,A** instruction.)
- Z Set if *sr* is zero, else reset
- H Reset
- P/V NMOS chip: Flag state is uncertain: If Interrupt Flip-Flop 2 (**IFF₂**) is reset, then the P/V flag is reset. If Interrupt Flip-Flop 2 is set, then the P/V flag may or may not be set.
CMOS chip: State of Interrupt Flip-Flop 2 (**IFF₂**)
- N Reset
- C No change

TIMING: 2 M cycles, 9 (4,5) T-states.

DESCRIPTION: The **A** register is loaded with the contents of the special purpose register *sr*. **I** is the interrupt vector register, and **R** is the memory refresh register.

8-BIT ARITHMETIC and LOGIC

ADC A,r

BINARY:

1 0 0 0 1 p p p

 if r is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1	1 0 0 0 1 p p p
-----------------	-----------------

 if r is HX, LX, HY, or LY

HEX:

r	ppp	q	
B	000		88
C	001		89
D	010		8A
E	011		8B
H	100		8C
L	101		8D
(HL)	110		8E
A	111		8F
HX	100	0	DD8C
LX	101	0	DD8D
HY	100	1	FD8C
LY	101	1	FD8D

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if carry from bit 3, else reset
 P/V Set if overflow, else reset
 N Reset
 C Set if carry from bit 7, else reset

TIMING: If r is B, C, D, E, H, L, (HL), or A: 1 M cycle, 4 T-states.
 If r is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If r is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If r is B, C, D, E, H, L, A, HX, LX, HY, or LY: The contents of register r and the contents of the Carry Bit are added to the contents of the A register, and the result is stored in the A register.
 • If r is (HL): The contents of the memory location of the HL register pair and the contents of the Carry Bit are added to the contents of the A register, and the result is stored in the A register.

8-BIT ARITHMETIC and LOGIC

ADC A,n

BINARY:

1 1 0 0 1 1 1 0	n n n n n n n n
-----------------	-----------------

HEX:

CE n

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if carry from bit 3, else reset
 P/V Set if overflow, else reset
 N Reset
 C Set if carry from bit 7, else reset

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The one-byte value n and the contents of the Carry Bit are added to the contents of the **A** register, and the result is stored in the **A** register.

8-BIT ARITHMETIC and LOGIC

ADC $A, (ir+d)$

BINARY:

1 1 q 1 1 1 0 1	1 0 0 0 1 1 1 0	d d d d d d d d
-----------------	-----------------	-----------------

HEX:

ir	q	
IX	0	DD8E d
IY	1	FD8E d

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if carry from bit 3, else reset
 P/V Set if overflow, else reset
 N Reset
 C Set if carry from bit 7, else reset

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus displacement *d* and the contents of the Carry Bit are added to the contents of the *A* register, and the result is stored in the *A* register. The displacement is a two's complement byte value in the range of -128 to +127.

8-BIT ARITHMETIC and LOGIC

ADD A,r

BINARY: 1 0 0 0 0 p p p if r is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1 1 0 0 0 0 p p p if r is HX, LX, HY, or LY

HEX:

r	ppp	q	
B	000		80
C	001		81
D	010		82
E	011		83
H	100		84
L	101		85
(HL)	110		86
A	111		87
HX	100	0	DD84
LX	101	0	DD85
HY	100	1	FD84
LY	101	1	FD85

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if carry from bit 3, else reset
 P/V Set if overflow, else reset
 N Reset
 C Set if carry from bit 7, else reset

TIMING: If r is B, C, D, E, H, L, or A: 1 M cycle, 4 T-states.
 If r is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If r is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If r is B, C, D, E, H, L, A, HX, LX, HY, or LY: The contents of register r is added to the contents of the A register, and the result is stored in the A register.
 • If r is (HL): The contents of the memory location of the HL register pair is added to the contents of the A register, and the result is stored in the A register.

8-BIT ARITHMETIC and LOGIC

ADD A,n

BINARY:

1 1 0 0 0 1 1 0	n n n n n n n n
-----------------	-----------------

HEX:

C6 n

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if carry from bit 3, else reset
P/V Set if overflow, else reset
N Reset
C Set if carry from bit 7, else reset

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The one-byte value n is added to the contents of the A register, and the result is stored in the A register.

8-BIT ARITHMETIC and LOGIC

ADD $A,(ir+d)$

BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 0 0 0 0 1 1 0	<i>d d d d d d d d</i>
------------------------	-----------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DD86 <i>d</i>
IY	1	FD86 <i>d</i>

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if carry from bit 3, else reset
P/V Set if overflow, else reset
N Reset
C Set if carry from bit 7, else reset

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus displacement *d* is added to the contents of the **A** register, and the result is stored in the **A** register. The displacement *d* is a two's complement byte value in the range of -128 to +127.

8-BIT ARITHMETIC and LOGIC

AND *r*

BINARY:

1 0 1 0 0 p p p

 if *r* is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1	1 0 1 0 0 p p p
-----------------	-----------------

 if *r* is HX, LX, HY, or LY

HEX:

<i>r</i>	ppp	q	
B	000		A0
C	001		A1
D	010		A2
E	011		A3
H	100		A4
L	101		A5
(HL)	110		A6
A	111		A7
HX	100	0	DDA4
LX	101	0	DDA5
HY	100	1	FDA4
LY	101	1	FDA5

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set
 P/V Set if even parity, else reset
 N Reset
 C Reset

TIMING: If *r* is B, C, D, E, H, L, or A: 1 M cycle, 4 T-states.
 If *r* is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, if *r* is HX, LX, HY, or LY: The contents of register *r* is ANDed with the contents of the A register, and the result is stored in the A register.
 • If *r* is (HL): The contents of the memory location of the HL register pair is ANDed with the contents of the A register, and the result is stored in the A register.

8-BIT ARITHMETIC and LOGIC

AND n

BINARY:

1 1 1 0 0 1 1 0	n n n n n n n n
-----------------	-----------------

HEX:

E6 n

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set
 P/V Set if even parity, else reset
 N Reset
 C Reset

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The one-byte value n is ANDed with the contents of the **A** register, and the result is stored in the **A** register.

8-BIT ARITHMETIC and LOGIC

AND (*ir+d*)

BINARY:

1 1 q 1 1 1 0 1	1 0 1 0 0 1 1 0	d d d d d d d d
-----------------	-----------------	-----------------

HEX:

ir	q	
IX	0	DDA6 d
IY	1	FDA6 d

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set
P/V Set if even parity, else reset
N Reset
C Reset

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus displacement *d* is ANDed with the contents of the **A** register, and the result is stored in the **A** register. The displacement *d* is a two's complement byte value in the range of -128 to +127.

8-BIT ARITHMETIC and LOGIC

CP *r*

BINARY: 1 0 1 1 1 p p p if *r* is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1 1 0 1 1 1 p p p if *r* is HX, LX, HY, or LY

HEX:

r	ppp	q	
B	000		B8
C	001		B9
D	010		BA
E	011		BB
H	100		BC
L	101		BD
(HL)	110		BE
A	111		BF
HX	100	0	DDBC
LX	101	0	DDBD
HY	100	1	FDBC
LY	101	1	FDBD

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if borrow from bit 4, else reset
 P/V Set overflow, else reset
 N Set
 C Set if borrow, else reset

TIMING: If *r* is B, C, D, E, H, L, or A: 1 M cycle, 4 T-states.
 If *r* is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, HX, LX, HY, or LY: The contents of register *r* is subtracted from the contents of the A register to affect the flags. The contents of the A register is unchanged.
 • If *r* is (HL): The contents of the memory location of the HL register pair is subtracted from the contents of the A register to affect the flags. The contents of the A register is unchanged.

8-BIT ARITHMETIC and LOGIC

CP *n*

BINARY:

1 1 1 1 1 1 1 0	n n n n n n n n
-----------------	-----------------

HEX:

FE n

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if borrow from bit 4, else reset
P/V Set if overflow, else reset
N Set
C Set if borrow, else reset

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The one-byte value *n* is subtracted from the contents of the A register to affect the flags. The contents of the A register is unchanged.

8-BIT ARITHMETIC and LOGIC

CP (*ir+d*)

BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 0 1 1 1 1 1 0	<i>d d d d d d d d</i>
------------------------	-----------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDBE <i>d</i>
IY	1	FDBE <i>d</i>

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if borrow from bit 4, else reset
P/V Set if overflow, else reset
N Set
C Set if borrow, else reset

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus displacement *d* is subtracted from the contents of the **A** register to affect the flags. The contents of the **A** register is unchanged. The displacement *d* is a two's complement byte value in the range of -128 to +127.

8-BIT ARITHMETIC and LOGIC

DEC *r*

BINARY: 0 0 p p p 1 0 1 if *r* is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1 0 0 p p p 1 0 1 if *r* is HX, LX, HY, or LY

HEX:

<i>r</i>	ppp	q	
B	000		05
C	001		0D
D	010		15
E	011		1D
H	100		25
L	101		2D
(HL)	110		35
A	111		3D
HX	100	0	DD25
LX	101	0	DD2D
HY	100	1	FD25
LY	101	1	FD2D

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if borrow from bit 4, else reset
 P/V Set if *r* is 7FH (was 80H before instruction execution), else reset
 N Set
 C No change

TIMING: If *r* is B, C, D, E, H, L, or A: 1 M cycle, 4 T-states.
 If *r* is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, HX, LX, HY, or LY: The contents of register *r* is decremented by one.
 • If *r* is (HL): The contents of the memory location of the HL register pair is decremented by one.

8-BIT ARITHMETIC and LOGIC

DEC (*ir+d*)

BINARY:

1 1 <i>q</i> 1 1 1 0 1	0 0 1 1 0 1 0 1	<i>d d d d d d d d</i>
------------------------	-----------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DD35 <i>d</i>
IY	1	FD35 <i>d</i>

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if borrow from bit 4, else reset
P/V Set if (*ir+d*) is 7FH (was 80H before instruction execution), else reset
N Set
C No change

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus displacement *d* is decremented by one. The displacement *d* is a two's complement byte value in the range of -128 to +127.

8-BIT ARITHMETIC and LOGIC

INC *r*

BINARY: 0 0 p p p 1 0 0 if *r* is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1 0 0 p p p 1 0 0 if *r* is HX, LX, HY, or LY

HEX:

<i>r</i>	ppp	q	
B	000		04
C	001		0C
D	010		14
E	011		1C
H	100		24
L	101		2C
(HL)	110		34
A	111		3C
HX	100	0	DD24
LX	101	0	DD2C
HY	100	1	FD24
LY	101	1	FD2C

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if carry from bit 3, else reset
 P/V Set if *r* is 80H (was 7FH before instruction execution), else reset
 N Reset
 C No change

TIMING: If *r* is B, C, D, E, H, L, or A: 1 M cycle, 4 T-states.
 If *r* is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, HX, LX, HY, or LY: The contents of register *r* is incremented by one.
 • If *r* is (HL): The contents of the memory location of the HL register pair is incremented by one.

8-BIT ARITHMETIC and LOGIC

INC ($ir+d$)

BINARY:

1 1 <i>q</i> 1 1 1 0 1	0 0 1 1 0 1 0 0	<i>d d d d d d d d</i>
------------------------	-----------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DD34 <i>d</i>
IY	1	FD34 <i>d</i>

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if carry from bit 3, else reset
P/V Set if ($ir+d$) is 80H (was 7FH before instruction execution), else reset
N Reset
C No change

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus displacement d is incremented by one. The displacement d is a two's complement byte value in the range of -128 to +127.

8-BIT ARITHMETIC and LOGIC

OR *r*

BINARY:

1 0 1 1 0 p p p

 if *r* is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1	1 0 1 1 0 p p p
-----------------	-----------------

 if *r* is HX, LX, HY, or LY

HEX:

<i>r</i>	ppp	q	
B	000		B0
C	001		B1
D	010		B2
E	011		B3
H	100		B4
L	101		B5
(HL)	110		B6
A	111		B7
HX	100	0	DDB4
LX	101	0	DDB5
HY	100	1	FDB4
LY	101	1	FDB5

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Reset

TIMING: If *r* is B, C, D, E, H, L, or A: 1 M cycle, 4 T-states.
 If *r* is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, HX, LX, HY, or LY: The contents of register *r* is inclusively ORed with the contents of the A register, and the result is stored in the A register.
 • If *r* is (HL): The contents of the memory location of the HL register pair is inclusively ORed with the contents of the A register, and the result is stored in the A register.

8-BIT ARITHMETIC and LOGIC

OR n

BINARY:

1 1 1 1 0 1 1 0	n n n n n n n n
-----------------	-----------------

HEX:

F6 n

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Reset

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The one-byte value n is inclusively ORed with the contents of the **A** register, and the result is stored in the **A** register.

8-BIT ARITHMETIC and LOGIC

OR (*ir+d*)

BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 0 1 1 0 1 1 0	<i>d d d d d d d d</i>
------------------------	-----------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDB6 <i>d</i>
IY	1	FDB6 <i>d</i>

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Reset

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus displacement *d* is inclusively ORed with the contents of the **A** register, and the result is stored in the **A** register. The displacement *d* is a two's complement byte value in the range of -128 to +127.

8-BIT ARITHMETIC and LOGIC

SBC *A,r*

BINARY:

1 0 0 1 1 p p p

 if *r* is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1	1 0 0 1 1 p p p
-----------------	-----------------

 if *r* is HX, LX, HY, or LY

HEX:

<i>r</i>	ppp	q	
B	000		98
C	001		99
D	010		9A
E	011		9B
H	100		9C
L	101		9D
(HL)	110		9E
A	111		9F
HX	100	0	DD9C
LX	101	0	DD9D
HY	100	1	FD9C
LY	101	1	FD9D

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if borrow from bit 4, else reset
 P/V Set if overflow, else reset
 N Set
 C Set if borrow, else reset

TIMING: If *r* is B, C, D, E, H, L, or A: 1 M cycle, 4 T-states.
 If *r* is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, HX, LX, HY, or LY: The contents of register *r* and the contents of the Carry Bit are subtracted from the contents of the A register, and the result is stored in the A register.
 • If *r* is (HL): The contents of the memory location of the HL register pair and the contents of the Carry Bit are subtracted from the contents of the A register, and the result is stored in the A register.

8-BIT ARITHMETIC and LOGIC

SBC A,n

BINARY:

1 1 0 1 1 1 1 0	n n n n n n n n
-----------------	-----------------

HEX:

DE n

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if borrow from bit 4, else reset
P/V Set if overflow, else reset
N Set
C Set if borrow, else reset

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The one-byte value n and the contents of the Carry Bit are subtracted from the contents of the **A** register, and the result is stored in the **A** register.

8-BIT ARITHMETIC and LOGIC

SBC $A, (ir+d)$

BINARY:

1 1 q 1 1 1 0 1	1 0 0 1 1 1 1 0	d d d d d d d d
-----------------	-----------------	-----------------

HEX:

ir	q	
IX	0	DD9E d
IY	1	FD9E d

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if borrow from bit 4, else reset
P/V Set if overflow, else reset
N Set
C Set if borrow, else reset

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus displacement d and the contents of the Carry Bit are subtracted from the contents of the A register, and the result is stored in the A register. The displacement is a two's complement byte value in the range of -128 to +127.

8-BIT ARITHMETIC and LOGIC

SUB *r*

BINARY:

1 0 0 1 0 p p p

 if *r* is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1

1 0 0 1 0 p p p

 if *r* is HX, LX, HY, or LY

HEX:

<i>r</i>	ppp	q	
B	000		90
C	001		91
D	010		92
E	011		93
H	100		94
L	101		95
(HL)	110		96
A	111		97
HX	100	0	DD94
LX	101	0	DD95
HY	100	1	FD94
LY	101	1	FD95

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if borrow from bit 4, else reset
 P/V Set if overflow, else reset
 N Set
 C Set if borrow, else reset

TIMING: If *r* is B, C, D, E, H, L, or A: 1 M cycle, 4 T-states.
 If *r* is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, HX, LX, HY, or LY: The contents of register *r* is subtracted from the contents of the A register, and the result is stored in the A register.
 • If *r* is (HL): The contents of the memory location of the HL register pair is subtracted from the contents of the A register, and the result is stored in the A register.

8-BIT ARITHMETIC and LOGIC

SUB n

BINARY:

1 1 0 1 0 1 1 0	n n n n n n n n
-----------------	-----------------

HEX:

D6 n

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if borrow from bit 4, else reset
 P/V Set if overflow, else reset
 N Set
 C Set if borrow, else reset

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The one-byte value n is subtracted from the contents of the **A** register, and the result is stored in the **A** register.

8-BIT ARITHMETIC and LOGIC

SUB (*ir+d*)

BINARY:

1 1 q 1 1 1 0 1	1 0 0 1 0 1 1 0	d d d d d d d d
-----------------	-----------------	-----------------

HEX:

<i>ir</i>	q	
IX	0	DD96 d
IY	1	FD96 d

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Set if borrow from bit 4, else reset
 P/V Set if overflow, else reset
 N Set
 C Set if borrow, else reset

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus displacement *d* is subtracted from the contents of the A register, and the result is stored in the A register. The displacement *d* is a two's complement byte value in the range of -128 to +127.

8-BIT ARITHMETIC and LOGIC

XOR *r*

BINARY: 1 0 1 0 1 p p p if *r* is B, C, D, E, H, L, (HL), or A

1 1 q 1 1 1 0 1 1 0 1 0 1 p p p if *r* is HX, LX, HY, or LY

HEX:

r	ppp	q	
B	000		A8
C	001		A9
D	010		AA
E	011		AB
H	100		AC
L	101		AD
(HL)	110		AE
A	111		AF
HX	100	0	DDAC
LX	101	0	DDAD
HY	100	1	FDAC
LY	101	1	FDAD

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Reset

TIMING: If *r* is B, C, D, E, H, L, A: 1 M cycle, 4 T-states.
 If *r* is (HL): 2 M cycles, 7 (4,3) T-states.
 If *r* is HX, LX, HY, or LY: 2 M cycles, 8 (4,4) T-states.

DESCRIPTION: • If *r* is B, C, D, E, H, L, A, HX, LX, HY, or LY: The contents of register *r* is exclusively ORed with the contents of the A register, and the result is stored in the A register.
 • If *r* is (HL): The contents of the memory location of the HL register pair is exclusively ORed with the contents of the A register, and the result is stored in the A register.

8-BIT ARITHMETIC and LOGIC

XOR *n*

BINARY:

1 1 1 0 1 1 1 0	n n n n n n n n
-----------------	-----------------

HEX:

EE n

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Reset
P/V Set if even parity, else reset
N Reset
C Reset

TIMING: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION: The one-byte value *n* is exclusively ORed with the contents of the A register, and the result is stored in the A register.

8-BIT ARITHMETIC and LOGIC

XOR (*ir+d*)

BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 0 1 0 1 1 1 0	<i>d d d d d d d d</i>
------------------------	-----------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDAE <i>d</i>
IY	1	FDAE <i>d</i>

FLAGS: S Set if result is negative, else reset
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Reset

TIMING: 5 M cycles, 19 (4,4,3,5,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus displacement *d* is exclusively ORed with the contents of the **A** register, and the result is stored in the **A** register. The displacement *d* is a two's complement byte value in the range of -128 to +127.

GENERAL PURPOSE and CPU CONTROL

CCF

BINARY:

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

HEX:

3F

FLAGS: S No change
Z No change
H Copy of carry flag before instruction execution
P/V No change
N Reset
C Complement of carry flag before instruction execution

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: Complement carry flag. The carry flag in the F register is reversed.

GENERAL PURPOSE and CPU CONTROL

SCF

BINARY:

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

HEX:

37

FLAGS: S No change
Z No change
H Reset
P/V No change
N Reset
C Set

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: Set carry flag. The carry flag in the F register is set.

GENERAL PURPOSE and CPU CONTROL

CPL

BINARY:

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

HEX:

2F

FLAGS: S No change
 Z No change
 H Set
 P/V No change
 N Set
 C No change

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: Complement. The bits in the A register are reversed (one's complement).

GENERAL PURPOSE and CPU CONTROL

NEG

BINARY:

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

HEX:

ED44

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if borrow from bit 4, else reset
P/V Set if A = 80H (was 80H before instruction execution), else reset
N Set
C Set if A <> 0 before instruction execution, else reset

TIMING: 2 M cycles, 8 (4,4) T-states.

DESCRIPTION: Negate the A register. The contents of the A register is subtracted from zero, and the result is stored in the A register (two's complement).

GENERAL PURPOSE and CPU CONTROL

DAA

BINARY: 0 0 1 0 0 1 1 1

HEX:

27

FLAGS:
 S Set if bit 7 set, else reset
 Z Set if **A** is zero, else reset
 H Set if carry from bit 3, else reset
 P/V Set if even parity, else reset
 N No change
 C Refer to chart below

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: Decimal adjust accumulator (**A** register). The contents of the **A** register is adjusted for BCD addition and BCD subtraction. The following chart describes the effects of a DAA instruction:

	N	H	C			C
Add/Subtract flag before DAA	Half Carry flag before DAA	Carry flag before DAA	Hex value in upper nibble	Hex value in lower nibble	Value added to byte	Carry flag after DAA
0	0	0	0 to 9	0 to 9	00	0
0	0	0	0 to 8	A to F	06	0
0	1	0	0 to 9	0 to 3	06	0
0	0	0	A to F	0 to 9	60	1
0	0	0	9 to F	A to F	66	1
0	1	0	A to F	0 to 3	66	1
0	0	1	0 to 2	0 to 9	60	1
0	0	1	0 to 2	A to F	66	1
0	1	1	0 to 3	0 to 3	66	1
1	0	0	0 to 9	0 to 9	00	0
1	1	0	0 to 8	6 to F	FA	0
1	0	1	7 to F	0 to 9	A0	1
1	1	1	6 to F	6 to F	9A	1

GENERAL PURPOSE and CPU CONTROL

NOP

BINARY:

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

HEX:

00

FLAGS: No change

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: No Operation. Executing a NOP instruction does nothing except consume four T-states.

GENERAL PURPOSE and CPU CONTROL

HALT

BINARY:

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

HEX:

76

FLAGS: No change

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: The HALT instruction directs the Z80[®] to execute NOP's until a non-maskable interrupt or reset is received by the Z80[®].

GENERAL PURPOSE and CPU CONTROL

DI

BINARY:

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

HEX:

F3

FLAGS: No change

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: Disables the maskable interrupt by resetting both Interrupt Flip-Flop 1 (IFF₁) and Interrupt Flip-Flop 2 (IFF₂).

GENERAL PURPOSE and CPU CONTROL

EI

BINARY:

1 1 1 1 1 0 1 1

HEX:

FB

FLAGS: No change

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: Enables the maskable interrupt by setting both interrupt flip-flops (IFF₁ and IFF₂). Maskable interrupts will not be recognized until the completion of the instruction following the EI instruction.

GENERAL PURPOSE and CPU CONTROL

IM 0

BINARY:

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

HEX:

ED46

FLAGS: No change

TIMING: 2 M cycles, 8 (4,4) T-states.

DESCRIPTION: Sets interrupt mode 0. When a maskable interrupt is accepted, the Z80[®] executes the instruction on the data bus. This is similar to an 8080A microprocessor; however, the Z80[®] only generates one interrupt acknowledge pulse whereas the 8080A generates three interrupt acknowledge pulses.

GENERAL PURPOSE and CPU CONTROL

IM 1

BINARY:

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

HEX:

ED56

FLAGS: No change

TIMING: 2 M cycles, 8 (4,4) T-states.

DESCRIPTION: Sets interrupt mode 1. When a maskable interrupt is accepted, the Z80[®] executes a RST 38H instruction.

GENERAL PURPOSE and CPU CONTROL

IM 2

BINARY:

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

HEX:

ED5E

FLAGS: No change

TIMING: 2 M cycles, 8 (4,4) T-states.

DESCRIPTION: Sets interrupt mode 2. When a maskable interrupt is accepted, the Z80[®] jumps to the memory location made up of the data bus and the Interrupt Vector Register, I. The data bus forms the least significant eight bits and the I register the most significant eight bits.

ROTATE and SHIFT

RLA

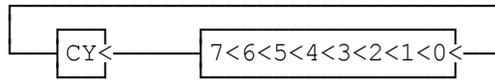
BINARY:

0 0 0 1 0 1 1 1

HEX:

17

SYMBOLIC:



FLAGS: S No change
Z No change
H Reset
P/V No change
N Reset
C Contents of bit 7 before instruction execution

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: The contents of the A register is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, the contents of bit 7 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 0.

ROTATE and SHIFT

RLCA

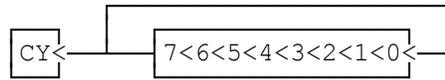
BINARY:

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

HEX:

07

SYMBOLIC:



FLAGS: S No change
 Z No change
 H Reset
 P/V No change
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: The contents of the **A** register is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into both bit 0 and the Carry Bit.

ROTATE and SHIFT

RRA

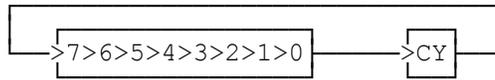
BINARY:

0 0 0 1 1 1 1 1

HEX:

1F

SYMBOLIC:



FLAGS: S No change
Z No change
H Reset
P/V No change
N Reset
C Contents of bit 0 before instruction execution

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: The contents of the A register is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 7.

ROTATE and SHIFT

RRCA

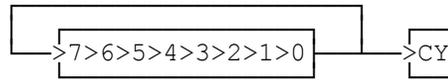
BINARY:

0 0 0 0 1 1 1 1

HEX:

0F

SYMBOLIC:



FLAGS: S No change
 Z No change
 H Reset
 P/V No change
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: The contents of the **A** register is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into both bit 7 and the Carry Bit.

ROTATE and SHIFT

RL *r*

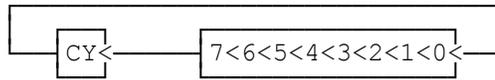
BINARY:

1 1 0 0 1 0 1 1	0 0 0 1 0 p p p
-----------------	-----------------

HEX:

<i>r</i>	ppp	
B	000	CB10
C	001	CB11
D	010	CB12
E	011	CB13
H	100	CB14
L	101	CB15
(HL)	110	CB16
A	111	CB17

SYMBOLIC:



FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 4 M cycles, 15 (4,5,3,3) T-states.

DESCRIPTION:

- If *r* is B, C, D, E, H, L, or A: The contents of register *r* is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, the contents of bit 7 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 0.
- If *r* is (HL): The contents of the memory location of the HL register pair is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, the contents of bit 7 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 0.

ROTATE and SHIFT

RL ($ir+d$)

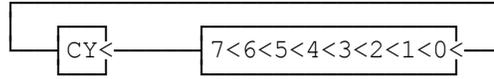
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 0 1 0 1 1 0
-------------------	-----------------	---------------------------------	-----------------

HEX:

	ir	q	
IX	0		DDCB d 16
IY	1		FDCB d 16

SYMBOLIC:



FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus the displacement d is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, the contents of bit 7 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 0. The displacement d is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

RL $r, (ir+d)$

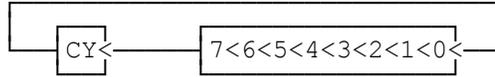
BINARY:

1	1	q	1	1	1	0	1	1	1	0	1	1	d	d	d	d	d	d	d	d	0	0	0	1	0	p	p	p
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

HEX:

ir	q	
IX	0	DDCB d 00010ppp
IY	1	FDCB d 00010ppp

SYMBOLIC:



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

- FLAGS:
- S Contents of bit 6 before instruction execution
 - Z Set if result is zero, else reset
 - H Reset
 - P/V Set if even parity, else reset
 - N Reset
 - C Contents of bit 7 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus the displacement d is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, the contents of bit 7 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 0; and register r is loaded with the contents of the memory location determined by the sum of the index register ir plus the displacement d . The displacement d is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

RLC *r*

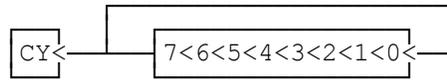
BINARY:

1 1 0 0 1 0 1 1	0 0 0 0 0 p p p
-----------------	-----------------

HEX:

<i>r</i>	ppp	
B	000	CB00
C	001	CB01
D	010	CB02
E	011	CB03
H	100	CB04
L	101	CB05
(HL)	110	CB06
A	111	CB07

SYMBOLIC:



FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 4 M cycles, 15 (4,5,3,3) T-states.

DESCRIPTION:

- If *r* is B, C, D, E, H, L, or A: The contents of register *r* is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into both bit 0 and the Carry Bit.
- If *r* is (HL): The contents of the memory location of the HL register pair is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into both bit 0 and the Carry Bit.

ROTATE and SHIFT

RLC (*ir+d*)

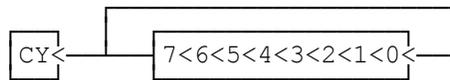
BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 1 0 0 1 0 1 1	<i>d d d d d d d d</i>	0 0 0 0 0 1 1 0
------------------------	-----------------	------------------------	-----------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDCB <i>d</i> 06
IY	1	FDCB <i>d</i> 06

SYMBOLIC:



FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus the displacement *d* is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into both bit 0 and the Carry Bit. The displacement *d* is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

RLC $r,(ir+d)$

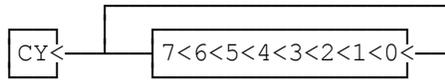
BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 1 0 0 1 0 1 1	<i>d d d d d d d d</i>	0 0 0 0 0 <i>p p p</i>
------------------------	-----------------	------------------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDCB <i>d 00000ppp</i>
IY	1	FDCB <i>d 00000ppp</i>

SYMBOLIC:



<i>ppp</i>	000	001	010	011	100	101	111
<i>r</i>	B	C	D	E	H	L	A

FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus the displacement *d* is rotated one bit to the left: the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into both bit 0 and the Carry Bit; and register *r* is loaded with the contents of the memory location determined by the sum of the index register *ir* plus the displacement *d*. The displacement *d* is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

RR *r*

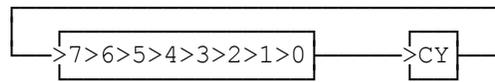
BINARY:

1 1 0 0 1 0 1 1	0 0 0 1 1 p p p
-----------------	-----------------

HEX:

<i>r</i>	ppp	
B	000	CB18
C	001	CB19
D	010	CB1A
E	011	CB1B
H	100	CB1C
L	101	CB1D
(HL)	110	CB1E
A	111	CB1F

SYMBOLIC:



FLAGS: S Contents of the carry flag before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 4 M cycles, 15 (4,5,3,3) T-states.

DESCRIPTION:

- If *r* is B, C, D, E, H, L, or A: The contents of register *r* is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 7.
- If *r* is (HL): The contents of the memory location of the HL register pair is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 7.

ROTATE and SHIFT

RR (*ir+d*)

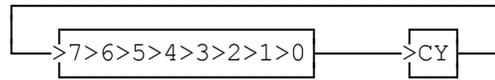
BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 1 0 0 1 0 1 1	<i>d d d d d d d d</i>	0 0 0 1 1 1 1 0
------------------------	-----------------	------------------------	-----------------

HEX:

	<i>ir</i>	<i>q</i>	
IX	0		DDCB <i>d</i> 1E
IY	1		FDCB <i>d</i> 1E

SYMBOLIC:



FLAGS: S Contents of the carry flag before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus the displacement *d* is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 7. The displacement *d* is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

RR $r, (ir+d)$

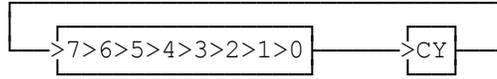
BINARY:

1	1	q	1	1	1	0	1	1	1	0	1	1	d	d	d	d	d	d	d	d	0	0	0	1	1	p	p	p
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

HEX:

ir	q	
IX	0	DDCB d 00011ppp
IY	1	FDCB d 00011ppp

SYMBOLIC:



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

- FLAGS:
- S Contents of the carry flag before instruction execution
 - Z Set if result is zero, else reset
 - H Reset
 - P/V Set if even parity, else reset
 - N Reset
 - C Contents of bit 0 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus the displacement d is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into the Carry Bit, and the contents of the Carry Bit before instruction execution is moved into bit 7; and register r is loaded with the contents of the memory location determined by the sum of the index register ir plus the displacement d . The displacement d is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

RRC *r*

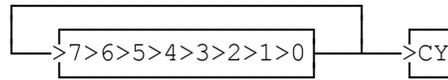
BINARY:

1 1 0 0 1 0 1 1	0 0 0 0 1 p p p
-----------------	-----------------

HEX:

<i>r</i>	ppp	
B	000	CB08
C	001	CB09
D	010	CB0A
E	011	CB0B
H	100	CB0C
L	101	CB0D
(HL)	110	CB0E
A	111	CB0F

SYMBOLIC:



FLAGS: S Contents of bit 0 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 4 M cycles, 15 (4,5,3,3) T-states.

DESCRIPTION:

- If *r* is B, C, D, E, H, L, or A: The contents of register *r* is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into both bit 7 and the Carry Bit.
- If *r* is (HL): The contents of the memory location of the HL register pair is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into both bit 7 and the Carry Bit.

ROTATE and SHIFT

RRC ($ir+d$)

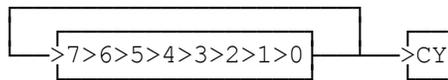
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 0 0 1 1 1 0
-----------------	-----------------	-----------------	-----------------

HEX:

	ir	q	
IX	0		DDCB d 0E
IY	1		FDCB d 0E

SYMBOLIC:



FLAGS: S Contents of bit 0 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus the displacement d is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into both bit 7 and the Carry Bit. The displacement d is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

RRC $r, (ir+d)$

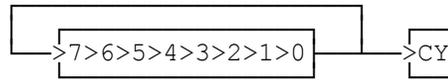
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 0 0 1 p p p
-----------------	-----------------	-----------------	-----------------

HEX:

	<i>ir</i>	<i>q</i>	
IX	0		DDCB d 00001ppp
IY	1		FDCB d 00001ppp

SYMBOLIC:



<i>ppp</i>	000	001	010	011	100	101	111
<i>r</i>	B	C	D	E	H	L	A

FLAGS: S Contents of bit 0 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus the displacement *d* is rotated one bit to the right: the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, the contents of bit 0 before instruction execution is moved into both bit 7 and the Carry Bit; and register *r* is loaded with the contents of the memory location determined by the sum of the index register *ir* plus the displacement *d*. The displacement *d* is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

SLA *r*

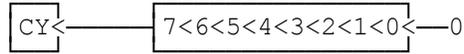
BINARY:

1 1 0 0 1 0 1 1	0 0 1 0 0 p p p
-----------------	-----------------

HEX:

<i>r</i>	ppp	
B	000	CB20
C	001	CB21
D	010	CB22
E	011	CB23
H	100	CB24
L	101	CB25
(HL)	110	CB26
A	111	CB27

SYMBOLIC:



FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 4 M cycles, 15 (4,5,3,3) T-states.

DESCRIPTION:

- If *r* is B, C, D, E, H, L, or A: The contents of register *r* is rotated one bit to the left: bit zero is reset, the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into the Carry Bit.
- If *r* is (HL): The contents of the memory location of the HL register pair is rotated one bit to the left: bit zero is reset, the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into the Carry Bit.

ROTATE and SHIFT

SLA ($ir+d$)

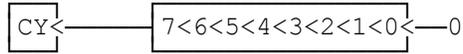
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 1 0 0 1 1 0
-------------------	-----------------	---------------------------------	-----------------

HEX:

	ir	q	
IX	0		DDCB d 26
IY	1		FDCB d 26

SYMBOLIC:



- FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus the displacement d is rotated one bit to the left: bit zero is reset, the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into the Carry Bit. The displacement d is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

SLA $r, (ir+d)$

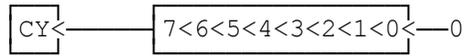
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 1 0 0 p p p
-----------------	-----------------	-----------------	-----------------

HEX:

ir	q	
IX	0	DDCB d 00100ppp
IY	1	FDCB d 00100ppp

SYMBOLIC:



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

- FLAGS:
- S Contents of bit 6 before instruction execution
 - Z Set if result is zero, else reset
 - H Reset
 - P/V Set if even parity, else reset
 - N Reset
 - C Contents of bit 7 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus the displacement d is rotated one bit to the left: bit zero is reset, the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into the Carry Bit; and register r is loaded with the contents of the memory location determined by the sum of the index register ir plus the displacement d . The displacement d is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

SLL *r*

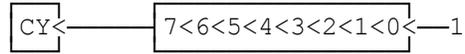
BINARY:

1 1 0 0 1 0 1 1	0 0 1 1 0 p p p
-----------------	-----------------

HEX:

<i>r</i>	ppp	
B	000	CB30
C	001	CB31
D	010	CB32
E	011	CB33
H	100	CB34
L	101	CB35
(HL)	110	CB36
A	111	CB37

SYMBOLIC:



FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 4 M cycles, 15 (4,5,3,3) T-states.

DESCRIPTION:

- If *r* is B, C, D, E, H, L, or A: The contents of register *r* is rotated one bit to the left: bit zero is set, the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into the Carry Bit.
- If *r* is (HL): The contents of the memory location of the HL register pair is rotated one bit to the left: bit zero is set, the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into the Carry Bit.

ROTATE and SHIFT

SLL (*ir+d*)

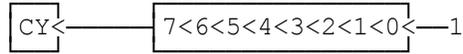
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 1 1 0 1 1 0
-----------------	-----------------	-----------------	-----------------

HEX:

	<i>ir</i>	<i>q</i>	
IX	0		DDCB d 36
IY	1		FDCB d 36

SYMBOLIC:



FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus the displacement *d* is rotated one bit to the left: bit zero is set, the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into the Carry Bit. The displacement *d* is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

SLL $r, (ir+d)$

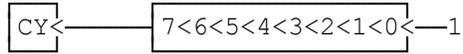
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 1 1 0 p p p
-----------------	-----------------	-----------------	-----------------

HEX:

	<i>ir</i>	<i>q</i>	
IX	0		DDCB d 00110ppp
IY	1		FDCB d 00110ppp

SYMBOLIC:



<i>ppp</i>	000	001	010	011	100	101	111
<i>r</i>	B	C	D	E	H	L	A

FLAGS: S Contents of bit 6 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 7 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus the displacement *d* is rotated one bit to the left: bit zero is set, the contents of bit 0 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 7, and the contents of bit 7 before instruction execution is moved into the Carry Bit; and register *r* is loaded with the contents of the memory location determined by the sum of the index register *ir* plus the displacement *d*. The displacement *d* is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

SRA *r*

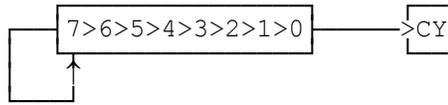
BINARY:

1 1 0 0 1 0 1 1	0 0 1 0 1 p p p
-----------------	-----------------

HEX:

<i>r</i>	ppp	
B	000	CB28
C	001	CB29
D	010	CB2A
E	011	CB2B
H	100	CB2C
L	101	CB2D
(HL)	110	CB2E
A	111	CB2F

SYMBOLIC:



FLAGS: S Contents of bit 7 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 4 M cycles, 15 (4,5,3,3) T-states.

DESCRIPTION:

- If *r* is B, C, D, E, H, L, or A: The contents of register *r* is rotated one bit to the right: the contents of bit 7 is copied into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, and the contents of bit 0 before instruction execution is moved into the Carry Bit.
- If *r* is (HL): The contents of the memory location of the HL register pair is rotated one bit to the right: the contents of bit 7 is copied into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, and the contents of bit 0 before instruction execution is moved into the Carry Bit.

ROTATE and SHIFT

SRA ($ir+d$)

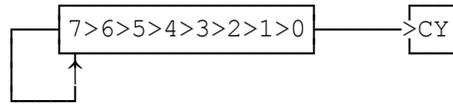
BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 1 0 0 1 0 1 1	<i>d d d d d d d d</i>	0 0 1 0 1 1 1 0
------------------------	-----------------	------------------------	-----------------

HEX:

	<i>ir</i>	<i>q</i>	
IX	0		DDCB <i>d</i> 2E
IY	1		FDCB <i>d</i> 2E

SYMBOLIC:



FLAGS: S Contents of bit 7 before instruction execution
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register *ir* plus the displacement *d* is rotated one bit to the right: the contents of bit 7 is copied into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, and the contents of bit 0 before instruction execution is moved into the Carry Bit. The displacement *d* is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

SRA $r, (ir+d)$

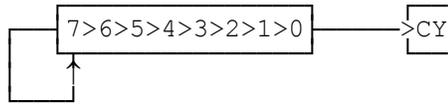
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 1 0 1 p p p
-----------------	-----------------	-----------------	-----------------

HEX:

SYMBOLIC:

ir	q	
IX	0	DDCB d 00101ppp
IY	1	FDCB d 00101ppp



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

- FLAGS:
- S Reset
 - Z Set if result is zero, else reset
 - H Reset
 - P/V Set if even parity, else reset
 - N Reset
 - C Contents of bit 0 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus the displacement d is rotated one bit to the right: the contents of bit 7 is copied into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, and the contents of bit 0 before instruction execution is moved into the Carry Bit; and register r is loaded with the contents of the memory location determined by the sum of the index register ir plus the displacement d .

ROTATE and SHIFT

SRL *r*

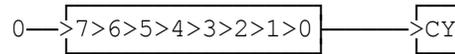
BINARY:

1 1 0 0 1 0 1 1	0 0 1 1 1 p p p
-----------------	-----------------

HEX:

	<i>r</i>	ppp	
B	000		CB38
C	001		CB39
D	010		CB3A
E	011		CB3B
H	100		CB3C
L	101		CB3D
(HL)	110		CB3E
A	111		CB3F

SYMBOLIC:



FLAGS: S Reset
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 4 M cycles, 15 (4,5,3,3) T-states.

DESCRIPTION:

- If *r* is B, C, D, E, H, L, or A: The contents of register *r* is rotated one bit to the right: bit 7 is reset, the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, and the contents of bit 0 before instruction execution is moved into the Carry Bit.
- If *r* is (HL): The contents of the memory location of the HL register pair is rotated one bit to the right: bit 7 is reset, the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, and the contents of bit 0 before instruction execution is moved into the Carry Bit.

ROTATE and SHIFT

SRL ($ir+d$)

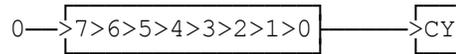
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 1 1 1 1 1 0
-----------------	-----------------	-----------------	-----------------

HEX:

	ir	q	
IX	0		DDCB d 3E
IY	1		FDCB d 3E

SYMBOLIC:



FLAGS: S Reset
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus the displacement d is rotated one bit to the right: bit 7 is reset, the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, and the contents of bit 0 before instruction execution is moved into the Carry Bit. The displacement d is a two's complement byte value in the range of -128 to +127.

ROTATE and SHIFT

SRL $r, (ir+d)$

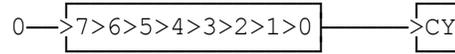
BINARY:

1 1 q 1 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	0 0 1 1 1 p p p
-------------------	-----------------	---------------------------------	-----------------------

HEX:

ir	q	
IX	0	DDCB d 00111ppp
IY	1	FDCB d 00111ppp

SYMBOLIC:



FLAGS: S Reset
 Z Set if result is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C Contents of bit 0 before instruction execution

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: The contents of the memory location determined by the sum of the index register ir plus the displacement d is rotated one bit to the right: bit 7 is reset, the contents of bit 7 before instruction execution is moved into bit 6, the contents of bit 6 before instruction execution is moved into bit 5, the contents of bit 5 before instruction execution is moved into bit 4, the contents of bit 4 before instruction execution is moved into bit 3, the contents of bit 3 before instruction execution is moved into bit 2, the contents of bit 2 before instruction execution is moved into bit 1, the contents of bit 1 before instruction execution is moved into bit 0, and the contents of bit 0 before instruction execution is moved into the Carry Bit; and register r is loaded with the contents of the memory location determined by the sum of the index register ir plus the displacement d .

ROTATE and SHIFT

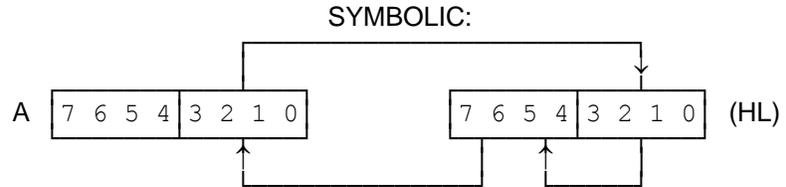
RLD

BINARY:

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

HEX:

ED6F



FLAGS: S Set if bit 7 in A is set, else reset
 Z Set if A is zero, else reset
 H Reset
 P/V Set if even parity, else reset
 N Reset
 C No change

TIMING: 5 M cycles, 18 (4,4,3,4,3) T-states.

DESCRIPTION: The contents of nibbles (four bits) are rotated one nibble to the left: the contents of the lower nibble (bits 0-3) of the HL memory location before instruction execution are copied to the high nibble (bits 4-7) of the same memory location, the contents of the higher nibble (bits 4-7) of the HL memory location before instruction execution are copied to the lower nibble (bits 0-3) of the A register, and the contents of the lower nibble (bits 0-3) of the A register before instruction execution are copied to the lower nibble (bits 0-3) of the HL memory location.

ROTATE and SHIFT

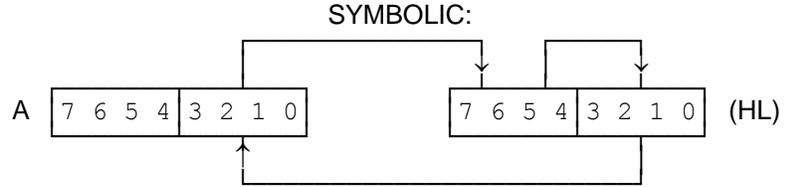
RRD

BINARY:

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

HEX:

ED67



FLAGS: S Set if bit 7 in A is set, else reset
Z Set if A is zero, else reset
H Reset
P/V Set if even parity, else reset
N Reset
C No change

TIMING: 5 M cycles, 18 (4,4,3,4,3) T-states.

DESCRIPTION: The contents of nibbles (four bits) are rotated one nibble to the right: the contents of the lower nibble (bits 0-3) of the A register before instruction execution are copied to the high nibble (bits 4-7) of the HL memory location, the contents of the higher nibble (bits 4-7) of the HL memory location before instruction execution are copied to the lower nibble (bits 0-3) of the same memory location, and the contents of the lower nibble (bits 0-3) of the HL memory location before instruction execution are copied to the lower nibble (bits 0-3) of the A register.

BIT MANIPULATION

BIT b, r

BINARY:

1 1 0 0 1 0 1 1	0 1 p p p q q q
-----------------	-----------------

HEX:

	r	B	C	D	E	H	L	(HL)	A
	qqq	000	001	010	011	100	101	110	111
b	ppp								
0	000	CB40	CB41	CB42	CB43	CB44	CB45	CB46	CB47
1	001	CB48	CB49	CB4A	CB4B	CB4C	CB4D	CB4E	CB4F
2	010	CB50	CB51	CB52	CB53	CB54	CB55	CB56	CB57
3	011	CB58	CB59	CB5A	CB5B	CB5C	CB5D	CB5E	CB5F
4	100	CB60	CB61	CB62	CB63	CB64	CB65	CB66	CB67
5	101	CB68	CB69	CB6A	CB6B	CB6C	CB6D	CB6E	CB6F
6	110	CB70	CB71	CB72	CB73	CB74	CB75	CB76	CB77
7	111	CB78	CB79	CB7A	CB7B	CB7C	CB7D	CB7E	CB7F

FLAGS: S Set if bit 7 is set and instruction is **BIT 7**, else reset
 Z Set if bit b is 0, else reset
 H Set
 P/V Set if bit b is 0, else reset
 N Reset
 C No change

TIMING: If r is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If r is (HL): 3 M cycles, 12 (4,4,4) T-states.

DESCRIPTION: • If r is B, C, D, E, H, L, or A: The complement of bit b of register r is copied to the Z flag.
 • If r is (HL): The complement of bit b of the memory location of the HL register pair is copied to the Z flag.

BIT MANIPULATION

BIT $b, (ir+d)$

BINARY:

1 1 0 <i>q</i> 1 1 0 1	1 1 0 0 1 0 1 1	<i>d</i> <i>d</i> <i>d</i> <i>d</i> <i>d</i> <i>d</i> <i>d</i> <i>d</i>	0 1 <i>p</i> <i>p</i> <i>p</i> 1 1 0
------------------------	-----------------	---	--------------------------------------

HEX:

		ir	IX	IY
		<i>q</i>	0	1
<i>b</i>	<i>ppp</i>			
0	000		DDCB <i>d</i> 46	FDCB <i>d</i> 46
1	001		DDCB <i>d</i> 4E	FDCB <i>d</i> 4E
2	010		DDCB <i>d</i> 56	FDCB <i>d</i> 56
3	011		DDCB <i>d</i> 5E	FDCB <i>d</i> 5E
4	100		DDCB <i>d</i> 66	FDCB <i>d</i> 66
5	101		DDCB <i>d</i> 6E	FDCB <i>d</i> 6E
6	110		DDCB <i>d</i> 76	FDCB <i>d</i> 76
7	111		DDCB <i>d</i> 7E	FDCB <i>d</i> 7E

FLAGS: S Set if bit 7 is set and instruction is BIT 7, else reset
 Z Set if bit b is 0, else reset
 H Set
 P/V Set if bit b is 0, else reset
 N Reset
 C No change

TIMING: 5 M cycles, 20 (4,4,3,5,4) T-states.

DESCRIPTION: The complement of bit b of the memory location determined by the sum of the index register ir plus the displacement d is copied to the Z flag. The displacement d is a two's complement byte value in the range of -128 to +127.

BIT MANIPULATION

RES *b,r*

BINARY:

1 1 0 0 1 0 1 1	1 0 p p p q q q
-----------------	-----------------

HEX:

	<i>r</i>	B	C	D	E	H	L	(HL)	A
	qqq	000	001	010	011	100	101	110	111
<i>b</i>	ppp								
0	000	CB80	CB81	CB82	CB83	CB84	CB85	CB86	CB87
1	001	CB88	CB89	CB8A	CB8B	CB8C	CB8D	CB8E	CB8F
2	010	CB90	CB91	CB92	CB93	CB94	CB95	CB96	CB97
3	011	CB98	CB99	CB9A	CB9B	CB9C	CB9D	CB9E	CB9F
4	100	CBA0	CBA1	CBA2	CBA3	CBA4	CBA5	CBA6	CBA7
5	101	CBA8	CBA9	CBAA	CBAB	CBAC	CBAD	CBAE	CBAF
6	110	CBB0	CBB1	CBB2	CBB3	CBB4	CBB5	CBB6	CBB7
7	111	CBB8	CBB9	CBBA	CBBB	CBBC	CBBD	CBBE	CBBF

FLAGS: No change

TIMING: If *r* is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If *r* is (HL): 3 M cycles, 12 (4,4,4) T-states.

DESCRIPTION:

- If *r* is B, C, D, E, H, L, or A: Bit *b* of register *r* is reset.
- If *r* is (HL): Bit *b* of the memory location of the HL register pair is reset.

BIT MANIPULATION

RES $b,(ir+d)$

BINARY:

1 1 0 q 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	1 0 p p p 1 1 0
-----------------	-----------------	-----------------	-----------------

HEX:

		ir	IX	IY
		q	0	1
b	ppp			
0	000	DDCB d 86	FDCB d 86	
1	001	DDCB d 8E	FDCB d 8E	
2	010	DDCB d 96	FDCB d 96	
3	011	DDCB d 9E	FDCB d 9E	
4	100	DDCB d A6	FDCB d A6	
5	101	DDCB d AE	FDCB d AE	
6	110	DDCB d B6	FDCB d B6	
7	111	DDCB d BE	FDCB d BE	

FLAGS: No change

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: Bit b of the memory location determined by the sum of the index register ir plus the displacement d is reset.

BIT MANIPULATION

RES $b, r, (ir+d)$

BINARY:

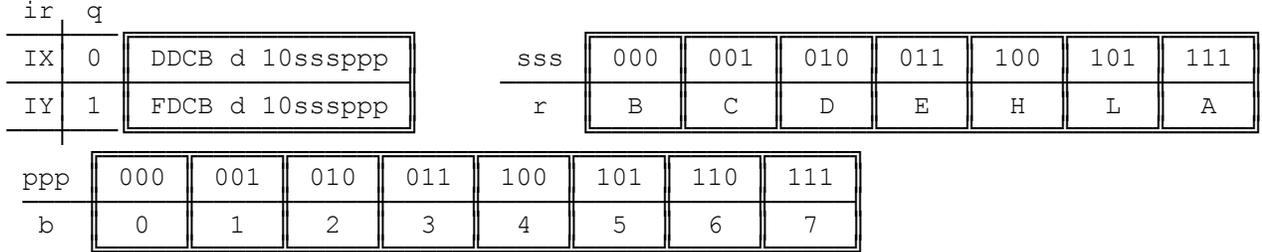
1	1	q	1	1	1	0	1
---	---	---	---	---	---	---	---

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

d	d	d	d	d	d	d	d
---	---	---	---	---	---	---	---

1	0	p	p	p	s	s	s
---	---	---	---	---	---	---	---

HEX:



FLAGS: No change

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: Bit b of the memory location determined by the sum of the index register ir plus the displacement d is reset, and register r is loaded with the contents of the memory location determined by the sum of the index register ir plus the displacement d .

BIT MANIPULATION

SET b,r

BINARY:

1 1 0 0 1 0 1 1	1 1 p p p q q q
-----------------	-----------------

HEX:

r	B	C	D	E	H	L	(HL)	A	
qqq	000	001	010	011	100	101	110	111	
b	ppp								
0	000	CBC0	CBC1	CBC2	CBC3	CBC4	CBC5	CBC6	CBC7
1	001	CBC8	CBC9	CBCA	CBCB	CBCC	CBCD	CBCE	CBCF
2	010	CBD0	CBD1	CBD2	CBD3	CBD4	CBD5	CBD6	CBD7
3	011	CBD8	CBD9	CBDA	CBDB	CBDC	CBDD	CBDE	CBDF
4	100	CBE0	CBE1	CBE2	CBE3	CBE4	CBE5	CBE6	CBE7
5	101	CBE8	CBE9	CBEA	CBEB	CBEC	CBED	CBEE	CBEF
6	110	CBF0	CBF1	CBF2	CBF3	CBF4	CBF5	CBF6	CBF7
7	111	CBF8	CBF9	CBFA	CBFB	CBFC	CBFD	CBFE	CBFF

FLAGS: No change

TIMING: If r is B, C, D, E, H, L, or A: 2 M cycles, 8 (4,4) T-states.
 If r is (HL): 3 M cycles, 12 (4,4,4) T-states.

DESCRIPTION:

- If r is B, C, D, E, H, L, or A: Bit b of register r is set.
- If r is (HL): Bit b of the memory location of the HL register pair is set.

BIT MANIPULATION

SET $b, (ir+d)$

BINARY:

1 1 0 q 1 1 0 1	1 1 0 0 1 0 1 1	d d d d d d d d	1 1 p p p 1 1 0
-----------------	-----------------	-----------------	-----------------

HEX:

		ir	IX	IY
		q	0	1
b	ppp			
0	000	DDCB d C6	FDCB d C6	
1	001	DDCB d CE	FDCB d CE	
2	010	DDCB d D6	FDCB d D6	
3	011	DDCB d DE	FDCB d DE	
4	100	DDCB d E6	FDCB d E6	
5	101	DDCB d EE	FDCB d EE	
6	110	DDCB d F6	FDCB d F6	
7	111	DDCB d FE	FDCB d FE	

FLAGS: No change

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: Bit b of the memory location determined by the sum of the index register ir plus the displacement d is set.

BIT MANIPULATION

SET $b,r,(ir+d)$

BINARY:

1	1	<i>q</i>	1	1	1	0	1
---	---	----------	---	---	---	---	---

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

<i>d</i>							
----------	----------	----------	----------	----------	----------	----------	----------

1	1	<i>p</i>	<i>p</i>	<i>p</i>	<i>s</i>	<i>s</i>	<i>s</i>
---	---	----------	----------	----------	----------	----------	----------

HEX:

<i>ir</i>	<i>q</i>									
IX	0	DDCB <i>d</i> 11sssppp								
IY	1	FDCB <i>d</i> 11sssppp								
		<i>sss</i>	000	001	010	011	100	101	111	
		<i>r</i>	B	C	D	E	H	L	A	
		<i>ppp</i>	000	001	010	011	100	101	110	111
		<i>b</i>	0	1	2	3	4	5	6	7

FLAGS: No change

TIMING: 6 M cycles, 23 (4,4,3,5,4,3) T-states.

DESCRIPTION: Bit *b* of the memory location determined by the sum of the index register *ir* plus the displacement *d* is set, and register *r* is loaded with the contents of the memory location determined by the sum of the index register *ir* plus the displacement *d*.

INPUT and OUTPUT

IN $A,(n)$

BINARY:

1 1 0 1 1 0 1 1	n n n n n n n n
-----------------	-----------------

HEX:

DB n

FLAGS: No change

TIMING: 3 M cycles, 11 (4,3,4) T-states.

DESCRIPTION: The A register is valued with the data in port n .

INPUT and OUTPUT

IN $r, (C)$

BINARY:

1 1 1 0 1 1 0 1	0 1 p p p 0 0 0
-----------------	-----------------

HEX:

r	ppp	
B	000	ED40
C	001	ED48
D	010	ED50
E	011	ED58
H	100	ED60
L	101	ED68
(HL)	110	ED70
A	111	ED78

Sets flags only. No data is transferred.

FLAGS:

- S Set if input data is negative, else reset
- Z Set if input data is zero, else reset
- H Reset
- P/V Set if input data has even parity, else reset
- N Reset
- C No change

TIMING: 3 M cycles, 12 (4,4,4) T-states.

DESCRIPTION: If r is B, C, D, E, H, L, or A, then register r is valued with the data in port C.

INPUT and OUTPUT

IND

BINARY:

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

HEX:

EDAA

FLAGS: S To an indeterminate state
Z Set if **B** = 1 before instruction execution, else reset
H To an indeterminate state
P/V To an indeterminate state
N To an indeterminate state
C To an indeterminate state

TIMING: 4 M cycles, 16 (4,5,3,4) T-states.

DESCRIPTION: The **C** register holds the address of the port that is used for input into the memory location of the **HL** register pair. After the value on the data in port **C** is written to the memory location of the **HL** register pair, the **HL** register pair is decremented by one and the **B** register is decremented by one.

INPUT and OUTPUT

INDR

BINARY:

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

HEX:

E DBA

FLAGS:

S	Reset
Z	Set
H	To an indeterminate state
P/V	To an indeterminate state
N	To an indeterminate state
C	To an indeterminate state

TIMING:

If **B** <> 1 before instruction execution: 5 M cycles, 21 (4,5,3,4,5) T-states.
If **B** = 1 before instruction execution: 4 M cycles, 16 (4,5,3,4) T-states.

DESCRIPTION:

The **C** register holds the address of the port that is used for input into the memory location of the **HL** register pair. After the value on the data in port **C** is written to the memory location of the **HL** register pair, the **HL** register pair is decremented by one and the **B** register is decremented by one.

- If the **B** register is not zero (after being decremented), then the program counter, **PC**, is decremented by two and the instruction is repeated.
- If the **B** register is zero (after being decremented), then the program counter, **PC**, continues.

INPUT and OUTPUT

INI

BINARY:

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

HEX:

EDA2

FLAGS: S To an indeterminate state
Z Set if **B** = 1 before instruction execution, else reset
H To an indeterminate state
P/V To an indeterminate state
N To an indeterminate state
C To an indeterminate state

TIMING: 4 M cycles, 16 (4,5,3,4) T-states.

DESCRIPTION: The **C** register holds the address of the port that is used for input into the memory location of the **HL** register pair. After the value on the data in port **C** is written to the memory location of the **HL** register pair, the **HL** register pair is incremented by one and the **B** register is decremented by one.

INPUT and OUTPUT

INIR

BINARY:

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

HEX:

E DB 2

FLAGS:

S	Reset
Z	Set
H	To an indeterminate state
P/V	To an indeterminate state
N	To an indeterminate state
C	To an indeterminate state

TIMING:

If **B** <> 1 before instruction execution: 5 M cycles, 21 (4,5,3,4,5) T-states.
If **B** = 1 before instruction execution: 4 M cycles, 16 (4,5,3,4) T-states.

DESCRIPTION:

The **C** register holds the address of the port that is used for input into the memory location of the **HL** register pair. After the value on the data in port **C** is written to the memory location of the **HL** register pair, the **HL** register pair is incremented by one and the **B** register is decremented by one.

- If the **B** register is not zero (after being decremented), then the program counter, **PC**, is decremented by two and the instruction is repeated.
- If the **B** register is zero (after being decremented), then the program counter, **PC**, continues.

INPUT and OUTPUT

OUT $(n),A$

BINARY:

1 1 0 1 0 0 1 1	n n n n n n n n
-----------------	-----------------

HEX:

D3 n

FLAGS: No change

TIMING: 3 M cycles, 11 (4,3,4) T-states.

DESCRIPTION: The contents of the A register is written to port n .

INPUT and OUTPUT

OUT (C),*r*

BINARY:

1 1 1 0 1 1 0 1	0 1 p p p 0 0 1
-----------------	-----------------

HEX:

<i>r</i>	ppp	
B	000	ED41
C	001	ED49
D	010	ED51
E	011	ED59
H	100	ED61
L	101	ED69
(HL)	110	ED71
A	111	ED79

Zero is output to port.

FLAGS: No change

TIMING: 3 M cycles, 12 (4,4,4) T-states.

DESCRIPTION: If *r* is B, C, D, E, H, L, or A, then the contents of register *r* is written to port *n*.

INPUT and OUTPUT

OUTD

BINARY:

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

HEX:

EDAB

FLAGS: S To an indeterminate state
Z Set if **B** = 1 before instruction execution, else reset
H Reset
P/V To an indeterminate state
N To an indeterminate state
C To an indeterminate state

TIMING: 4 M cycles, 16 (4,5,3,4) T-states.

DESCRIPTION: The memory location of the **HL** register pair contains the data to be written to the address of the port in the **C** register. After the value from the memory location of the **HL** register pair is written to port **C**, the **HL** register pair is decremented by one and the **B** register is decremented by one.

INPUT and OUTPUT

OTDR

BINARY:

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

HEX:

E DBB

FLAGS:

S	Reset
Z	Set
H	To an indeterminate state
P/V	To an indeterminate state
N	To an indeterminate state
C	To an indeterminate state

TIMING:

If **B** <> 1 before instruction execution: 5 M cycles, 21 (4,5,3,4,5) T-states.
If **B** = 1 before instruction execution: 4 M cycles, 16 (4,5,3,4) T-states.

DESCRIPTION:

The memory location of the **HL** register pair contains the data to be written to the address of the port in the **C** register. After the value from the memory location of the **HL** register pair is written to port **C**, the **HL** register pair is decremented by one and the **B** register is decremented by one.

- If the **B** register is not zero (after being decremented), then the program counter, **PC**, is decremented by two and the instruction is repeated.
- If the **B** register is zero (after being decremented), then the program counter, **PC**, continues.

INPUT and OUTPUT

OUTI

BINARY:

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

HEX:

EDA3

FLAGS: S To an indeterminate state
Z Set if **B** = 1 before instruction execution, else reset
H Reset
P/V To an indeterminate state
N To an indeterminate state
C To an indeterminate state

TIMING: 4 M cycles, 16 (4,5,3,4) T-states.

DESCRIPTION: The memory location of the **HL** register pair contains the data to be written to the address of the port in the **C** register. After the value from the memory location of the **HL** register pair is written to port **C**, the **HL** register pair is incremented by one and the **B** register is decremented by one.

INPUT and OUTPUT

OTIR

BINARY:

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

HEX:

E DB 3

FLAGS:

S	Reset
Z	Set
H	To an indeterminate state
P/V	To an indeterminate state
N	To an indeterminate state
C	To an indeterminate state

TIMING:

If **B** <> 1 before instruction execution: 5 M cycles, 21 (4,5,3,4,5) T-states.
If **B** = 1 before instruction execution: 4 M cycles, 16 (4,5,3,4) T-states.

DESCRIPTION:

The memory location of the **HL** register pair contains the data to be written to the address of the port in the **C** register. After the value from the memory location of the **HL** register pair is written to port **C**, the **HL** register pair is incremented by one and the **B** register is decremented by one.

- If the **B** register is not zero (after being decremented), then the program counter, **PC**, is decremented by two and the instruction is repeated.
- If the **B** register is zero (after being decremented), then the program counter, **PC**, continues.

16-BIT LOAD

LD *rr,mn*

BINARY:

0 0 p p 0 0 0 1	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------

HEX:

<i>rr</i>	<i>pp</i>	
BC	00	01 <i>nm</i>
DE	01	11 <i>nm</i>
HL	10	21 <i>nm</i>
SP	11	31 <i>nm</i>

FLAGS: No change

TIMING: 3 M cycles, 10 (4,3,3) T-states.

DESCRIPTION: Register pair *rr* is loaded with the word value *mn*.

16-BIT LOAD

LD *rr*,(*mn*)

BINARY:

1 1 1 0 1 1 0 1	0 1 p p 1 0 1 1	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------	-----------------

HEX:

<i>rr</i>	<i>pp</i>	
BC	00	ED4B <i>nm</i>
DE	01	ED5B <i>nm</i>
HL	10	ED6E <i>nm</i>
SP	11	ED7B <i>nm</i>

FLAGS: No change

TIMING: 6 M cycles, 20 (4,4,3,3,3,3) T-states.

DESCRIPTION: The low order portion of register pair *rr* is loaded with the contents of memory location *mn*. The high order portion of register pair *rr* is loaded with the contents of memory location *mn*+1.

The low order portion of the BC register pair is the C register, and the high order portion of the BC register pair is the B register. The low order portion of the DE register pair is the E register, and the high order portion of the DE register pair is the D register. The low order portion of the HL register pair is the L register, and the high order portion of the HL register pair is the H register.

16-BIT LOAD

LD (mn),rr

BINARY:

1 1 1 0 1 1 0 1	0 1 p p 0 0 1 1	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------	-----------------

HEX:

rr	pp	
BC	00	ED43 nm
DE	01	ED53 nm
HL	10	ED63 nm
SP	11	ED73 nm

FLAGS: No change

TIMING: 6 M cycles, 20 (4,4,3,3,3,3) T-states.

DESCRIPTION: Memory location *mn* is loaded with the contents of the low order portion of register pair *rr*. Memory location *mn*+1 is loaded with the contents of the high order portion of register pair *rr*.

The low order portion of the **BC** register pair is the **C** register, and the high order portion of the **BC** register pair is the **B** register. The low order portion of the **DE** register pair is the **E** register, and the high order portion of the **DE** register pair is the **D** register. The low order portion of the **HL** register pair is the **L** register, and the high order portion of the **HL** register pair is the **H** register.

16-BIT LOAD

LD *ir,mn*

BINARY:

1 1 <i>q</i> 1 1 1 0 1	0 0 1 0 0 0 0 1	<i>n n n n n n n n</i>	<i>m m m m m m m m</i>
------------------------	-----------------	------------------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DD21 nm
IY	1	FD21 nm

FLAGS: No change

TIMING: 4 M cycles, 14 (4,4,3,3) T-states.

DESCRIPTION: Index register pair *ir* is loaded with the word value *mn*.

16-BIT LOAD

LD *ir*,(*mn*)

BINARY:

1 1 q 1 1 1 0 1	0 0 1 0 1 0 1 0	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------	-----------------

HEX:

<i>ir</i>	q	
IX	0	DD2A nm
IY	1	FD2A nm

FLAGS: No change

TIMING: 6 M cycles, 20 (4,4,3,3,3,3) T-states.

DESCRIPTION: The low order portion of index register pair *ir* is loaded with the contents of memory location *mn*. The high order portion of index register pair *ir* is loaded with the contents of memory location *mn*+1.

The low order portion of the IX register pair is the LX register, and the high order portion of the IX register pair is the HX register. The low order portion of the IY register pair is the LY register, and the high order portion of the IY register pair is the HY register.

16-BIT LOAD

LD (*mn*),*ir*

BINARY:

1 1 <i>q</i> 1 1 1 0 1	0 0 1 0 0 0 1 0	<i>n n n n n n n n</i>	<i>m m m m m m m m</i>
------------------------	-----------------	------------------------	------------------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DD22 <i>nm</i>
IY	1	FD22 <i>nm</i>

FLAGS: No change

TIMING: 6 M cycles, 20 (4,4,3,3,3,3) T-states.

DESCRIPTION: Memory location *mn* is loaded with the contents of the low order portion of index register *ir*. Memory location *mn*+1 is loaded with the contents of the high order portion of index register *ir*.

The low order portion of the IX register pair is the LX register, and the high order portion of the IX register pair is the HX register. The low order portion of the IY register pair is the LY register, and the high order portion of the IY register pair is the HY register.

16-BIT LOAD

LD HL,(*mn*)

BINARY:

0 0 1 0 1 0 1 0	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------

HEX:

2A nm

FLAGS: No change

TIMING: 5 M cycles, 16 (4,3,3,3,3) T-states.

DESCRIPTION: The L register is loaded with the contents of memory location *mn*, and the H register is loaded with the contents of memory location *mn*+1.

16-BIT LOAD

LD (*mn*),HL

BINARY:

0 0 1 0 0 0 1 0	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------

HEX:

22 nm

FLAGS: No change

TIMING: 5 M cycles, 16 (4,3,3,3,3) T-states.

DESCRIPTION: The contents of the **L** register is loaded into memory location *mn*, and the contents of the **H** register is loaded into memory location *mn*+1.

16-BIT LOAD

LD SP,HL

BINARY:

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

HEX:

F9

FLAGS: No change

TIMING: 1 M cycle, 6 T-states.

DESCRIPTION: The stack pointer, **SP**, is loaded with the contents of the **HL** register pair.

16-BIT LOAD

LD SP,*ir*

BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 1 1 1 1 0 0 1
------------------------	-----------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDF9
IY	1	FDF9

FLAGS: No change

TIMING: 2 M cycles, 10 (4,6) T-states.

DESCRIPTION: The stack pointer, **SP**, is loaded with the contents of index register pair *ir*.

16-BIT LOAD

EX DE,HL

BINARY:

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

HEX:

EB

FLAGS: No change

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: The contents of the **DE** register pair is exchanged with the contents of the **HL** register pair (the contents of the **E** register is exchanged with the contents of the **L** register, and the contents of the **D** register is exchanged with the contents of the **H** register).

16-BIT LOAD

EXX

BINARY:

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

HEX:

D9

FLAGS: No change

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: The contents of the main BC, DE, and HL register pairs are exchanged with the contents of the alternate BC, DE, and HL register pairs, BC', DE', and HL'.

16-BIT LOAD

PUSH *rr*

BINARY:

1	1	p	p	0	1	0	1
---	---	---	---	---	---	---	---

HEX:

<i>rr</i>	<i>pp</i>	
BC	00	C5
DE	01	D5
HL	10	E5
AF	11	F5

FLAGS: No change

TIMING: 3 M cycles, 11 (5,3,3) T-states.

DESCRIPTION: The stack pointer, **SP**, is decremented; the memory location of the stack pointer, **SP**, is loaded with the contents of the high order portion of register pair *rr*; then the stack pointer, **SP**, is decremented again; and the memory location of the stack pointer, **SP**, is loaded with the contents of the low order portion of register pair *rr*.

The low order portion of the **BC** register pair is the **C** register, and the high order portion of the **BC** register pair is the **B** register. The low order portion of the **DE** register pair is the **E** register, and the high order portion of the **DE** register pair is the **D** register. The low order portion of the **HL** register pair is the **L** register, and the high order portion of the **HL** register pair is the **H** register. The low order portion of the **AF** register pair is the **F** register, and the high order portion of the **AF** register pair is the **A** register.

16-BIT LOAD

PUSH *ir*

BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 1 1 0 0 1 0 1
------------------------	-----------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDE5
IY	1	FDE5

FLAGS: No change

TIMING: 4 M cycles, 15 (4,5,3,3) T-states.

DESCRIPTION: The stack pointer, **SP**, is decremented; the memory location of the stack pointer, **SP**, is loaded with the contents of the high order portion of index register pair *ir*; then the stack pointer, **SP**, is decremented again; and the memory location of the stack pointer, **SP**, is loaded with the contents of the low order portion of index register pair *ir*.

The low order portion of the **IX** register pair is the **LX** register, and the high order portion of the **IX** register pair is the **HX** register. The low order portion of the **IY** register pair is the **LY** register, and the high order portion of the **IY** register pair is the **HY** register.

16-BIT LOAD

POP *rr*

BINARY:

1	1	p	p	0	0	0	1
---	---	---	---	---	---	---	---

HEX:

<i>rr</i>	<i>pp</i>	
BC	00	C1
DE	01	D1
HL	10	E1
AF	11	F1

FLAGS: No change if *rr* is BC, DE, or HL.
If AF, then the flags depend on the value loaded into flag register F.

TIMING: 3 M cycles, 10 (4,3,3) T-states.

DESCRIPTION: The low order portion of register *rr* is loaded with the contents of the memory location of the stack pointer, SP; the stack pointer, SP, is incremented; then the high order portion of register *rr* is loaded with the contents of the memory location of the stack pointer, SP; and the stack pointer, SP, is incremented again.

The low order portion of the BC register pair is the C register, and the high order portion of the BC register pair is the B register. The low order portion of the DE register pair is the E register, and the high order portion of the DE register pair is the D register. The low order portion of the HL register pair is the L register, and the high order portion of the HL register pair is the H register. The low order portion of the AF register pair is the F register, and the high order portion of the AF register pair is the A register.

16-BIT LOAD

POP *ir*

BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 1 1 0 0 0 0 1
------------------------	-----------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDE1
IY	1	FDE1

FLAGS: No change

TIMING: 4 M cycles, 14 (4,4,3,3) T-states.

DESCRIPTION: The low order portion of index register *ir* is loaded with the contents of the memory location of the stack pointer, *SP*; the stack pointer, *SP*, is incremented; then the high order portion of index register *ir* is loaded with the contents of the memory location of the stack pointer, *SP*; and the stack pointer, *SP*, is incremented again.

The low order portion of the IX register pair is the LX register, and the high order portion of the IX register pair is the HX register. The low order portion of the IY register pair is the LY register, and the high order portion of the IY register pair is the HY register.

16-BIT LOAD

EX (SP),HL

BINARY:

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

HEX:

E3

FLAGS: No change

TIMING: 5 M cycles, 19 (4,3,4,3,5) T-states.

DESCRIPTION: The contents of the **L** register is exchanged with the contents of the memory location of the stack pointer, **SP**. The contents of the **H** register is exchanged with the memory location of the stack pointer plus 1, **SP+1**.

16-BIT LOAD

EX (SP),*ir*

BINARY:

1 1 <i>q</i> 1 1 1 0 1	1 1 1 0 0 0 1 1
------------------------	-----------------

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDE3
IY	1	FDE3

FLAGS: No change

TIMING: 6 M cycles, 23 (4,4,3,4,3,5) T-states.

DESCRIPTION: The low order portion of index register *ir* is exchanged with the contents of the memory location of the stack pointer, **SP**. The high order portion of index register *ir* is exchanged with the contents of the memory location of the stack pointer plus 1, **SP+1**.

The low order portion of the **IX** register pair is the **LX** register, and the high order portion of the **IX** register pair is the **HX** register. The low order portion of the **IY** register pair is the **LY** register, and the high order portion of the **IY** register pair is the **HY** register.

16-BIT ARITHMETIC

ADC HL,rr

BINARY:

1 1 1 0 1 1 0 1	0 1 p p 1 0 1 0
-----------------	-----------------

HEX:

rr	pp	
BC	00	ED4A
DE	01	ED5A
HL	10	ED6A
SP	11	ED7A

FLAGS: S Set if result is negative, else reset
Z Set if result is zero, else reset
H Set if carry from bit 11, else reset
P/V Set if overflow, else reset
N Reset
C Set if carry from bit 15, else reset

TIMING: 4 M cycle, 15 (4,4,4,3) T-states.

DESCRIPTION: The contents of register pair *rr* and the contents of the Carry Bit are added to the contents of the HL register pair, and the result is stored in the HL register pair.

16-BIT ARITHMETIC

ADD HL,*rr*

BINARY:

0	0	<i>p</i>	<i>p</i>	1	0	0	1
---	---	----------	----------	---	---	---	---

HEX:

<i>rr</i>	<i>pp</i>	
BC	00	09
DE	01	19
HL	10	29
SP	11	39

FLAGS: S No change
Z No change
H Set if carry from bit 11, else reset
P/V No change
N Reset
C Set if carry from bit 15, else reset

TIMING: 3 M cycles, 11 (4,4,3) T-states.

DESCRIPTION: The contents of register pair *rr* is added to the contents of the HL register pair, and the result is stored in the HL register pair.

16-BIT ARITHMETIC

ADD *ir,rr*

BINARY:

1 1 q 1 1 1 0 1	0 0 p p 1 0 0 1
-----------------	-----------------

HEX:

<i>ir</i>	IX	IY
<i>q</i>	0	1
<i>rr</i>	<i>pp</i>	
BC	00	DD 09 FD 09
DE	01	DD 19 FD 19
IX	10	DD 29
IY	10	FD 29
SP	11	DD 39 FD 39

FLAGS: S No change
 Z No change
 H Set if carry from bit 11, else reset
 P/V No change
 N Reset
 C Set if carry from bit 15, else reset

TIMING: 4 M cycles, 15 (4,4,4,3) T-states.

DESCRIPTION: The contents of register pair *rr* or index register pair *ir* is added to the contents of index register pair *ir*, and the result is stored in index register pair *ir*.

16-BIT ARITHMETIC

DEC *rr*

BINARY:

0	0	p	p	1	0	1	1
---	---	---	---	---	---	---	---

HEX:

<i>rr</i>	<i>pp</i>	
BC	00	0B
DE	01	1B
HL	10	2B
SP	11	3B

FLAGS: No change

TIMING: 1 M cycle, 6 T-states.

DESCRIPTION: Register pair *rr* is decremented by one.

16-BIT ARITHMETIC

DEC *ir*

BINARY:

1	1	q	1	1	1	0	1	0	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DD2B
IY	1	FD2B

FLAGS: No change

TIMING: 2 M cycles, 10 (4,6) T-states.

DESCRIPTION: Index register pair *ir* is decremented by one.

16-BIT ARITHMETIC

INC *rr*

BINARY:

0	0	p	p	0	0	1	1
---	---	---	---	---	---	---	---

HEX:

<i>rr</i>	<i>pp</i>	
BC	00	03
DE	01	13
HL	10	23
SP	11	33

FLAGS: No change

TIMING: 1 M cycle, 6 T-states.

DESCRIPTION: Register pair *rr* is incremented by one.

16-BIT ARITHMETIC

INC *ir*

BINARY:

1	1	q	1	1	1	0	1	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

HEX:

<i>ir</i>	q	
IX	0	DD23
IY	1	FD23

FLAGS: No change

TIMING: 2 M cycles, 10 (4,6) T-states.

DESCRIPTION: Index register pair *ir* is incremented by one.

16-BIT ARITHMETIC

SBC HL,*rr*

BINARY:

1 1 1 0 1 1 0 1	0 1 p p 0 0 1 0
-----------------	-----------------

HEX:

<i>rr</i>	<i>pp</i>	
BC	00	ED42
DE	01	ED52
HL	10	ED62
SP	11	ED72

FLAGS: S Set if result negative, else reset
Z Set if result zero, else reset
H Set if borrow from bit 12, else reset
P/V Set if overflow, else reset
N Set
C Set if borrow, else reset

TIMING: 4 M cycles, 15 (4,4,4,3) T-states.

DESCRIPTION: The contents of register pair *rr* and the contents of the Carry Bit are subtracted from the HL register pair, and the result is stored in the HL register pair.

BRANCH

JP *mn*

BINARY:

1 1 0 0 0 0 1 1	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------

HEX:

C3 nm

FLAGS: No change

TIMING: 3 M cycles, 10 (4,3,3) T-states.

DESCRIPTION: The program counter, *PC*, is loaded with the value *mn*, and program execution branches to memory location *mn*.

BRANCH

JP *cc,mn*

BINARY:

1 1 p p p 0 1 0	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------

HEX:

Condition Description	cc	flag state	ppp	
non-zero	NZ	0	000	C2 nm
zero	Z	1	001	CA nm
non-carry	NC	0	010	D2 nm
carry	C	1	011	DA nm
parity odd	PO	0	100	E2 nm
parity even	PE	1	101	EA nm
sign positive	P	0	110	F2 nm
sign negative	M	1	111	FA nm

FLAGS: No change

TIMING: 3 M cycles, 10 (4,3,3) T-states.

DESCRIPTION:

- If condition **cc** is true, then the program counter, **PC**, is loaded with the value *mn*, and program execution branches to memory location *mn*.
- If condition **cc** is not true, then the program counter, **PC**, continues.

BRANCH

JP (HL)

BINARY:

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

HEX:

E9

FLAGS: No change

TIMING: 1 M cycle, 4 T-states.

DESCRIPTION: The program counter, **PC**, is loaded with the contents of the **HL** register pair, and program execution branches to the memory location of the **HL** register pair.

BRANCH

JP (*ir*)

BINARY:

1	1	q	1	1	1	0	1	1	1	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

HEX:

<i>ir</i>	<i>q</i>	
IX	0	DDE9
IY	1	FDE9

FLAGS: No change

TIMING: 2 M cycles, 8 (4,4) T-states.

DESCRIPTION: The program counter, *PC*, is loaded with the contents of index register *ir*, and program execution branches to the memory location of index register pair *ir*.

BRANCH

JR *d*

BINARY:

0 0 0 1 1 0 0 0	d d d d d d d d
-----------------	-----------------

HEX:

18 d

FLAGS: No change

TIMING: 4 M cycles, 12 (4,3,5) T-states.

DESCRIPTION: The program counter, **PC**, branches to the memory location determined by the displacement, **d**. The displacement is a two's complement byte value in the range of -128 to +127. The branched to memory location is in the range of -126 (-128 +2) to +129 (+127 +2) from the JR instruction.

BRANCH

JR *cc,d*

BINARY:

0 0 1 p p 0 0 0	d d d d d d d d
-----------------	-----------------

HEX:

Condition Description	cc	flag state	pp	
non-zero	NZ	0	00	20 d
zero	Z	1	01	28 d
non-carry	N	0	10	30 d
carry	C	1	11	38 d

FLAGS: No change

TIMING: If condition **CC** is true: 4 M cycles, 12 (4,3,5) T-states.
If condition **CC** is not true: 2 M cycles, 7 (4,3) T-states.

DESCRIPTION:

- If condition **CC** is true, then the program counter, **PC**, branches to the memory location determined by the displacement, **d**. The displacement is a two's complement byte value in the range of -128 to +127. The branched to memory location is in the range of -126 (-128 +2) to +129 (+127 +2) from the JR instruction.
- If condition **CC** is not true, then the program counter, **PC**, continues.

BRANCH

DJNZ *d*

BINARY:

0 0 0 1 0 0 0 0	d d d d d d d d
-----------------	-----------------

HEX:

10 d

FLAGS: No change

TIMING: If **B** <> 1 before instruction execution: 4 M cycles, 13 (4,3,3,3) T-states.
If **B** = 1 before instruction execution: 2 M cycles, 8 (4,4) T-states.

DESCRIPTION: Decrement (**B**) and jump non-zero.

- If **B** <> 1 before instruction execution, then the program counter, **PC**, branches to the memory location determined by the displacement, *d*. The displacement is a two's complement byte value in the range of -128 to +127. The branched to memory location is in the range of -126 (-128 +2) to +129 (+127 +2) from the DJNZ instruction.
- If **B** = 1 before instruction execution, then the program counter, **PC**, continues.

Although this instruction is named decrement and jump non-zero, the flags are not changed. This is also true if **B** was valued with one before instruction execution. i.e., the "Z" flag does not change with this instruction.

BRANCH

CALL *mn*

BINARY:

1 1 0 0 1 1 0 1	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------

HEX:

CD nm

FLAGS: No change

TIMING: 5 M cycles, 17 (4,3,4,3,3) T-states.

DESCRIPTION: The stack pointer, *SP*, is decremented; the memory location of the stack pointer, *SP*, is loaded with the contents of the high order portion of the program counter, *PC*; then the stack pointer, *SP*, is decremented again; and the memory location of the stack pointer, *SP*, is loaded with the contents of the low order portion of the program counter, *PC*. The program counter, *PC*, is loaded with the value *mn*, and program execution branches to memory location *mn*.

BRANCH

CALL *cc,mn*

BINARY:

1 1 p p p 1 0 0	n n n n n n n n	m m m m m m m m
-----------------	-----------------	-----------------

HEX:

Condition Description	cc	flag state	ppp	
non-zero	NZ	0	000	C4 nm
zero	Z	1	001	CC nm
non-carry	NC	0	010	D4 nm
carry	C	1	011	DC nm
parity odd	PO	0	100	E4 nm
parity even	PE	1	101	EC nm
sign positive	P	0	110	F4 nm
sign negative	M	1	111	FC nm

FLAGS: No change

TIMING: If condition **CC** is true: 5 M cycles, 17 (4,3,4,3,3) T-states.
 If condition **CC** is not true: 3 M cycles, 10 (4,3,3) T-states.

DESCRIPTION:

- If condition **CC** is true, then the stack pointer, **SP**, is decremented; the memory location of the stack pointer, **SP**, is loaded with the contents of the high order portion of the program counter, **PC**; then the stack pointer, **SP**, is decremented again; and the memory location of the stack pointer, **SP**, is loaded with the contents of the low order portion of the program counter, **PC**. The program counter, **PC**, is loaded with the value *mn*, and program execution branches to memory location *mn*.
- If condition **CC** is not true, then the program counter, **PC**, continues.

BRANCH

RET

BINARY:

1 1 0 0 1 0 0 1

HEX:

C9

FLAGS: No change

TIMING: 3 M cycles, 10 (4,3,3) T-states.

DESCRIPTION: The low order portion of the program counter, **PC**, is loaded with the contents of the memory location of the stack pointer, **SP**; the stack pointer, **SP**, is incremented; then the high order portion of the program counter, **PC**, is loaded with the contents of the memory location of the stack pointer, **SP**; and the stack pointer, **SP**, is incremented again. Program execution branches to the address in the program counter, **PC**.

BRANCH

RET *cc*

BINARY:

1	1	<i>p</i>	<i>p</i>	<i>p</i>	0	0	0
---	---	----------	----------	----------	---	---	---

HEX:

Condition Description	<i>cc</i>	flag state	<i>ppp</i>	
non-zero	NZ	0	000	C0
zero	Z	1	001	C8
non-carry	NC	0	010	D0
carry	C	1	011	D8
parity odd	PO	0	100	E0
parity even	PE	1	101	E8
sign positive	P	0	110	F0
sign negative	M	1	111	F8

FLAGS: No change

TIMING: If condition **CC** is true: 3 M cycles, 11 (5,3,3) T-states.
If condition **CC** is not true: 1 M cycles, 5 T-states.

DESCRIPTION:

- If condition **CC** is true, then The low order portion of the program counter, **PC**, is loaded with the contents of the memory location of the stack pointer, **SP**; the stack pointer, **SP**, is incremented; then the high order portion of the program counter, **PC**, is loaded with the contents of the memory location of the stack pointer, **SP**; and the stack pointer, **SP**, is incremented again. Program execution branches to the address in the program counter, **PC**.
- If condition **CC** is not true, then the program counter, **PC**, continues.

BRANCH

RETI

BINARY:

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

HEX:

ED4D

FLAGS: No change

TIMING: 4 M cycles, 14 (4,4,3,3) T-states.

DESCRIPTION: Return from maskable interrupt. The low order portion of the program counter, **PC**, is loaded with the contents of the memory location of the stack pointer, **SP**; the stack pointer, **SP**, is incremented; then the high order portion of the program counter, **PC**, is loaded with the contents of the memory location of the stack pointer, **SP**; and the stack pointer, **SP**, is incremented again. Program execution branches to the address in the program counter, **PC**.

RETI must be used with interrupt mode 2 (IM 2) to enable daisy-chained interrupting devices to have the device's interrupt condition reset.

BRANCH

RETN

BINARY:

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

HEX:

ED45

FLAGS: No change

TIMING: 4 M cycles, 14 (4,4,3,3) T-states.

DESCRIPTION: Return from non-maskable interrupt. The low order portion of the program counter, **PC**, is loaded with the contents of the memory location of the stack pointer, **SP**; the stack pointer, **SP**, is incremented; then the high order portion of the program counter, **PC**, is loaded with the contents of the memory location of the stack pointer, **SP**; and the stack pointer, **SP**, is incremented again. Program execution branches to the address in the program counter, **PC**. The contents of **IFF₂** is copied back into **IFF₁** to restore **IFF₁** to its state prior to the non-maskable interrupt.

BRANCH

RST n

BINARY:

1	1	p	p	p	1	1	1
---	---	---	---	---	---	---	---

HEX:

n	ppp	
00H	000	C7
08H	001	CF
10H	010	D7
18H	011	DF
20H	100	E7
28H	101	EF
30H	110	F7
38H	111	FF

FLAGS: No change

TIMING: 3 M cycles, 11 (5,3,3) T-states.

DESCRIPTION: The stack pointer, **SP**, is decremented; the memory location of the stack pointer, **SP**, is loaded with the contents of the high order portion of the program counter, **PC**; then the stack pointer, **SP**, is decremented again; and the memory location of the stack pointer, **SP**, is loaded with the contents of the low order portion of the program counter, **PC**. The program counter, **PC**, is loaded with the value n , and program execution branches to memory location n .

COPY/MOVE

LDD

BINARY:

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

HEX:

EDA8

FLAGS: S No change
Z No change
H Reset
P/V Set if **BC** <> 1 before instruction execution, else reset
N Reset
C No change

TIMING: 4 M cycles. 16 (4,4,3,5) T-states.

DESCRIPTION: The contents of the memory location of the **HL** register pair is copied to the memory location of the **DE** register pair; the **DE** and **HL** register pairs are decremented by one, and the **BC** register pair is decremented by one.

COPY/MOVE

LDDR

BINARY:

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

HEX:

E DB 8

FLAGS:

S	No change
Z	No change
H	Reset
P/V	Reset
N	Reset
C	No change

TIMING:

If **B** <> 1 before instruction execution: 5 M cycles, 21 (4,4,3,5,5) T-states.
If **B** = 1 before instruction execution: 4 M cycles, 16 (4,4,3,5) T-states.

DESCRIPTION:

The contents of the memory location of the **HL** register pair is copied to the memory location of the **DE** register pair; the **DE** and **HL** register pairs are decremented by one, and the **BC** register pair is decremented by one.

- If the **BC** register pair is not zero (after being decremented), then the program counter, **PC**, is decremented by two and the instruction is repeated.
- If the **BC** register pair is zero (after being decremented), then the program counter, **PC**, continues.

COPY/MOVE

LDI

BINARY:

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

HEX:

E D A 0

FLAGS: S No change
Z No change
H Reset
P/V Set if **BC** <> 1 before instruction execution, else reset
N Reset
C No change

TIMING: 4 M cycles. 16 (4,4,3,5) T-states.

DESCRIPTION: The contents of the memory location of the **HL** register pair is copied to the memory location of the **DE** register pair; the **DE** and **HL** register pairs are incremented by one, and the **BC** register pair is decremented by one.

COPY/MOVE

LDIR

BINARY:

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

HEX:

E DB 0

FLAGS: S No change
Z No change
H Reset
P/V Reset
N Reset
C No change

TIMING: If **B** <> 1 before instruction execution: 5 M cycles, 21 (4,4,3,5,5) T-states.
If **B** = 1 before instruction execution: 4 M cycles, 16 (4,4,3,5) T-states.

DESCRIPTION: The contents of the memory location of the **HL** register pair is copied to the memory location of the **DE** register pair; the **DE** and **HL** register pairs are incremented by one, and the **BC** register pair is decremented by one.

- If the **BC** register pair is not zero (after being decremented), then the program counter, **PC**, is decremented by two and the instruction is repeated.
- If the **BC** register pair is zero (after being decremented), then the program counter, **PC**, continues.

SEARCH

CPD

BINARY:

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

HEX:

EDA9

FLAGS: S Set if result is negative, else reset
Z Set if **A** = (**HL**), before instruction execution, else reset
H Set if borrow from bit 4, else reset
P/V Set if **BC** <> 1 before instruction execution, else reset
N Set
C No change

TIMING: 4 M cycles, 16 (4,4,3,5) T-states.

DESCRIPTION: The contents of the memory location of the **HL** register pair is subtracted from the contents of the **A** register to affect all flags except the C flag. The contents of the **A** register is unchanged. The **HL** register pair is decremented by one, and the **BC** register pair is decremented by one.

SEARCH

CPDR

BINARY:

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

HEX:

EDB9

FLAGS: S Set if result is negative, else reset
Z Set if A = (HL), else reset
H Set if borrow from bit 4, else reset
P/V Set if BC <> 1 before instruction execution, else reset
N Set
C No change

TIMING: If B <> 1 before instruction execution: 5 M cycles, 21 (4,4,3,5,5) T-states.
If B = 1 before instruction execution: 4 M cycles, 16 (4,4,3,5) T-states.

DESCRIPTION: The contents of the memory location of the HL register pair is subtracted from the contents of the A register to affect all flags except the C flag. The contents of the A register is unchanged. The HL register pair is decremented by one, and the BC register pair is decremented by one.

- If the BC register pair is not zero (after being decremented) **and** A <> (HL), then the program counter, PC, is decremented by two and the instruction is repeated.
- If the BC register is zero (after being decremented) **or** A = (HL) then the program counter, PC, continues.

SEARCH

CPI

BINARY:

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

HEX:

EDA1

FLAGS: S Set if result is negative, else reset
Z Set if **A** = (**HL**), before instruction execution, else reset
H Set if borrow from bit 4, else reset
P/V Set if **BC** <> 1 before instruction execution, else reset
N Set
C No change

TIMING: 4 M cycles. 16 (4,4,3,5) T-states.

DESCRIPTION: The contents of the memory location of the **HL** register pair is subtracted from the contents of the **A** register to affect all flags except the C flag. The contents of the **A** register is unchanged. The **HL** register pair is incremented by one, and the **BC** register pair is decremented by one.

SEARCH

CPIR

BINARY:

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

HEX:

EDB1

FLAGS: S Set if result is negative, else reset
Z Set if **A = (HL)**, else reset
H Set if borrow from bit 4, else reset
P/V Set if **BC <> 1** before instruction execution, else reset
N Set
C No change

TIMING: If **B <> 1** before instruction execution: 5 M cycles, 21 (4,4,3,5,5) T-states.
If **B = 1** before instruction execution: 4 M cycles, 16 (4,4,3,5) T-states.

DESCRIPTION: The contents of the memory location of the **HL** register pair is subtracted from the contents of the **A** register to affect all flags except the **C** flag. The contents of the **A** register is unchanged. The **HL** register pair is incremented by one, and the **BC** register pair is decremented by one.

- If the **BC** register pair is not zero (after being decremented) **and** **A <> (HL)**, then the program counter, **PC**, is decremented by two and the instruction is repeated.
- If the **BC** register is zero (after being decremented) **or** **A = (HL)** then the program counter, **PC**, continues.

Z80 Flags

The Z80[®] has two 8-bit flag registers F and F'. The bits are set or reset by various CPU instructions. Zilog, Inc. assigned names to six of the eight bits: bits 3 and 5 are not assigned nor affects documented. Four bits are testable and are used for conditional branching: S, Z, P/V, and C. The flag bit affects shown here are what the flags actually do. All of the other publications I have seen on Z80[®] flags are incorrect.

In addition to CPU instructions, a program can manipulate the flag bits because they are in specific positions in the following format:

bit	7	6	5	4	3	2	1	0
flag	S	Z	*	H	*	P/V	N	C

bit	flag	definition
7	S	SIGN
6	Z	ZERO
5	*	not used
4	H	HALF CARRY
3	*	not used
2	P/V	PARITY/OVERFLOW
1	N	ADD/SUBTRACT
0	C	CARRY

SIGN flag: S

Mnemonic: SIGN flag set = M e.g., branch if sign is negative (S=1): JP M,mn
 SIGN flag reset = P e.g., branch if sign is positive (S=0): JP P,mn

The SIGN flag is set if the result of an arithmetic instruction using signed numbers is negative. Basically, 8-bit signed numbers are a two's complement byte value in the range of -128 to +127, and 16-bit signed numbers are a two's complement word value in the range of -32768 to +32767. The most significant bit is reset for positive signed numbers and set for negative signed numbers; therefore, an 8-bit representation of -16 is 11110000. Arithmetic instructions reflect the state of the most significant bit of the result in the SIGN flag. If the most significant bit of a result is set, then the SIGN flag is set. If the most significant bit of the result is reset, then the SIGN flag is reset.

Instructions that place the SIGN flag into an indeterminate state:

IND INI OUTD OUTI

Instructions that reset the SIGN flag:

SRL INDR INIR OTDR OTIR

Other instructions that affect the SIGN flag:

EX AF,AF'	LD A,sr	AND	OR	XOR	NEG	DAA
RL	RLC	RR	RRC	SLA	SLL	SRA
RLD	RRD	BIT	IN r,(C)	POP AF	CPD	CPDR
CPI	CPIR					

Arithmetic instructions that do not change the SIGN flag:

ADD HL,rr ADD ir,rr DEC rr DEC ir INC rr INC ir

Z80 Flags

ZERO flag: Z

Mnemonic: ZERO flag set = Z e.g., branch if zero (Z=1): JP Z,mn
ZERO flag reset = NZ e.g., branch if not zero (Z=0): JP NZ,mn

The zero flag is set if the result of an arithmetic instruction is zero. Remember the CP instruction performs a subtraction without affecting the contents in the A register. So, if the A register contained 23H and the L register contained 23H, then a CP A,L instruction would result in the ZERO flag set. Also, if the sum of an add instruction is exactly one more than the maximum value for resultant register/register pair, then the ZERO (and CARRY) flag is set.

Other instructions that affect the ZERO flag:

EX AF,AF'	LD A,sr	AND	OR	XOR	NEG	DAA
RL	RLC	RR	RRC	SLA	SLL	SRA
RLD	SRL	RRD	BIT	IN r,(C)	POP AF	CPD
CPDR	CPI	CPIR				

Arithmetic instructions that do not change the ZERO flag:

ADD HL,rr	ADD ir,rr	DEC rr	DEC ir	INC rr	INC ir
-----------	-----------	--------	--------	--------	--------

Instructions that set the ZERO flag:

INDR	INIR	OTDR	OTIR
------	------	------	------

HALF CARRY flag: H

The HALF CARRY flag is set by the execution of: an 8-bit arithmetic instruction that produces a carry from bit 3 into bit 4, or a borrow from bit 4 into bit 3; a 16-bit arithmetic instruction that produces a carry from bit 11 into bit 12, or a borrow from bit 12 into bit 11. The HALF CARRY flag is tested only with the DAA instruction to correct the result after a BCD add or subtract instruction.

The CCF instruction copies the state of the CARRY flag before instruction execution into the HALF CARRY flag.

Instructions that place the HALF CARRY flag into an indeterminate state:

IND	INDR	INI	INIR	OTDR	OTIR	OUTD
OUTI						

Instructions that reset the HALF CARRY flag:

LD A,sr	OR	XOR	SCF	RLA	RLCA	RRA
RRCA	RL	RLC	RR	RRC	SLA	SLL
SRA	SRL	RLD	RRD	IN r,(C)	LDD	LDDR
LDI	LDIR					

Other instructions that affect the HALF CARRY flag:

EX AF,AF'	NEG	DAA	POP AF	CPD	CPDR	CPI
CPIR						

Instructions that set the HALF CARRY flag:

AND	CPL	BIT
-----	-----	-----

Z80 Flags

PARITY/OVERFLOW flag: P/V

Mnemonic: P/V flag set = PE e.g., branch if even parity (P/V=1): JP PE,mn
P/V flag reset = PO e.g., branch if odd parity (P/V=0): JP PO,mn

The PARITY/OVERFLOW flag serves three purposes: parity, other, and overflow.

PARITY

For AND, OR, XOR, and IN r,(C) instructions: the PARITY/OVERFLOW flag is set if the result has an even number of bits set; the PARITY/OVERFLOW flag is reset if the result has an odd number of bits set.

OTHER

After the execution of a CPI, CPIR, CPD, CPDR, LDI, or LDD instruction, the PARITY/OVERFLOW flag reflects the status of the BC register pair: if the BC register pair is not zero, then the PARITY/OVERFLOW flag is set; if the BC register pair is zero, then the PARITY/OVERFLOW flag is reset.

The LD A,I and LD A,R instructions affect the PARITY/OVERFLOW flag as follows:

NMOS chip: If Interrupt Flip-Flop 2 (IFF₂) is set, then the PARITY/OVERFLOW flag may or may not be set. If Interrupt Flip-Flop 2 is reset, then the PARITY/OVERFLOW flag is reset.

CMOS chip: If Interrupt Flip-Flop 2 (IFF₂) is set, then the PARITY/OVERFLOW flag is set. If Interrupt Flip-Flop 2 is reset, then the PARITY/OVERFLOW flag is reset.

OVERFLOW

The PARITY/OVERFLOW flag is set when an overflow occurs during an arithmetic instruction that uses signed numbers. This can only happen when adding operands with like signs or subtracting operands with unlike signs. e.g., if the INC D instruction is encountered when D contains 7FH, then the PARITY/OVERFLOW flag is set

Instructions that place the PARITY/OVERFLOW flag into an indeterminate state:

IND	INDR	INI	INIR	OTD	OTDR	OTI
OTIR						

Instructions that reset the PARITY/OVERFLOW flag:

LDDR	LDIR
------	------

Other instructions that affect the PARITY/OVERFLOW flag:

EX AF,AF'	AND	OR	XOR	NEG	DAA	RL
RLC	RR	RRC	SLA	SLL	SRA	SRL
RLD	RRD	BIT	IN r,(C)	POP AF	LDD	LDI
CPD	CPDR	CPI	CPIR			

Arithmetic instructions that do not change the PARITY/OVERFLOW flag:

ADD HL,rr	ADD ir,rr	ADD ir,ir	DEC rr	DEC ir	INC rr	INC ir
-----------	-----------	-----------	--------	--------	--------	--------

Z80 Flags

ADD/SUBTRACT flag: N

The ADD/SUBTRACT flag is reset for all add instructions and is set for all subtract instructions. The ADD/SUBTRACT flag is tested only with the DAA instruction to correct the result after a BCD add or subtract instruction.

Instructions that place the ADD/SUBTRACT flag into an indeterminate state:

IND	INDR	INI	INIR	OTDR	OTIR	OUTD
OUTI						

Instructions that reset the ADD/SUBTRACT flag:

LD A, <i>sr</i>	ADC	ADD	AND	INC	OR	XOR
CCF	RLA	RLCA	RRA	RRCA	RL	RLC
RR	RRC	SLA	SLL	SRA	SRL	RLD
RRD	BIT	IN <i>r</i> ,(C)	LDD	LDDR	LDI	LDIR

Other instructions that affect the ADD/SUBTRACT flag:

EX AF,AF'	POP AF
-----------	--------

Instructions that set the ADD/SUBTRACT flag:

DEC	CP	SBC	SUB	SCF	CPL	NEG
CPD	CPDR	CPI	CPIR			

CARRY flag: C

Mnemonic: CARRY flag set = C e.g., branch if carry (C=1): JP C,mn
CARRY flag reset = NC e.g., branch if no carry (C=0): JP NC,mn

The carry flag is set if the execution of an add instruction produces a carry from the most significant bit. The CARRY flag also is set if the execution of a subtract instruction produces a borrow into the most significant bit.

Instructions that place the CARRY flag into an indeterminate state:

IND	INDR	INI	INIR	OTDR	OTIR	OUTD
OUTI						

Instructions that reset the CARRY flag:

AND	OR	XOR
-----	----	-----

Other instructions that affect the CARRY flag:

EX AF,AF'	CCF	NEG	DAA	RLA	RLCA	RRA
RRCA	RL	RLC	RR	RRC	SLA	SLL
SRA	SRL	POP AF				

Arithmetic instructions that do not change the CARRY flag:

DEC	INC
-----	-----

Instructions that set the CARRY flag:

SCF

Z80 Flags

Flag summary:

The table below reflects how the flags are affected by various Z80[®] instructions. The following legend is used:

- ? = flag is indeterminate
- 0 = flag is reset
- ↑ = flag is affected in accordance with the previous deliberation
- = flag is not affected
- 1 = flag is set

flag				S	Z	H	P/V	N	C
8-bit:									
ADC	ADD			↑	↑	↑	↑	0	↑
AND				↑	↑	1	↑	0	0
BIT				↑	b'	1	b'	0	•
CCF				•	•	C	•	0	C'
CP	NEG	SBC	SUB	↑	↑	↑	↑	1	↑
CPL				•	•	1	•	1	•
DAA				↑	↑	↑	↑	•	↑
DEC				↑	↑	↑	↑	1	•
INC				↑	↑	↑	↑	0	•
IND	INI	OUTD	OUTI	?	↑	?	?	?	?
INDR	INIR	OTDR	OTIR	0	1	?	?	?	?
IN <i>r</i> , (C)				↑	↑	0	↑	0	•
LD <i>A</i> , <i>sr</i>				↑	↑	0	IFF ₂	0	•
OR	XOR			↑	↑	0	↑	0	0
RLA	RLCA	RRA	RRCA	•	•	0	•	0	↑
RL	RLC	RR	RRC	↑	↑	0	↑	0	↑
SLA	SLL	SRA							
RLD	RRD			↑	↑	0	↑	0	•
SCF				•	•	0	•	0	1
SRL				0	↑	0	↑	0	↑
16-bit:									
ADC				↑	↑	↑	↑	0	↑
ADD				•	•	↑	•	0	↑
CPD	CPDR	CPI	CPIR	↑	↑	↑	↑	1	•
LDD	LDI			•	•	0	↑	0	•
LDDR	LDIR			•	•	0	0	0	•
SBC				↑	↑	↑	↑	1	↑

Z80 Execution Times

The Z80[®] executes each instruction in one to six machine cycles (M cycles). Each Z80[®] machine cycle takes from three to six clock periods (T-states). e.g., LD A,23H requires two machine cycles: the first machine cycle, M1, requires four clock periods (T-states) and the second machine cycle requires three clock periods; therefore, this instruction requires 1.72625 μ S (7 T-states) to execute in a machine with a CPU clock speed of 4.05504 MHz (standard TRS-80[®] Model 4 with no added wait states).

The table below is the execution time for specific T-States for various CPU speeds.

Z80[®] speed in MHz

	6.75840	6.33600	6.08256	5.06880	4.05504	3.54817	2.66113	2.02752	1.77408
	[20.2752/3]	[12.672/2]	[20.2752/3.33]	[20.2752/4]	[20.2752/5]	[10.6445/3]	[10.6445/4]	[20.2752/10]	[10.64456/6]
T-states	Execution time in microseconds								
1	0.14796	0.15783	0.16440	0.19729	0.24661	0.28184	0.37578	0.49321	0.56367
2	0.29593	0.31566	0.32881	0.39457	0.49321	0.56367	0.75156	0.98643	1.12734
3	0.44389	0.47348	0.49321	0.59186	0.73982	0.84551	1.12734	1.47964	1.69101
4	0.59186	0.63131	0.65762	0.78914	0.98643	1.12734	1.50312	1.97285	2.25469
5	0.73982	0.78914	0.82202	0.98643	1.23303	1.40918	1.87890	2.46607	2.81836
6	0.88778	0.94697	0.98643	1.18371	1.47964	1.69101	2.25469	2.95928	3.38203
7	1.03575	1.10480	1.15083	1.38100	1.72625	1.97285	2.63047	3.45249	3.94570
8	1.18371	1.26263	1.31524	1.57828	1.97285	2.25469	3.00625	3.94571	4.50937
9	1.33168	1.42045	1.47964	1.77557	2.21946	2.53652	3.38203	4.43892	5.07304
10	1.47964	1.57828	1.64404	1.97285	2.46607	2.81836	3.75781	4.93213	5.63671
11	1.62760	1.73611	1.80845	2.17014	2.71267	3.10019	4.13359	5.42535	6.20039
12	1.77557	1.89394	1.97285	2.36742	2.95928	3.38203	4.50937	5.91856	6.76406
13	1.92353	2.05177	2.13726	2.56471	3.20589	3.66386	4.88515	6.41177	7.32773
14	2.07150	2.20960	2.30166	2.76199	3.45249	3.94570	5.26093	6.90499	7.89140
15	2.21946	2.36742	2.46607	2.95928	3.69910	4.22754	5.63671	7.39820	8.45507
16	2.36742	2.52525	2.63047	3.15657	3.94571	4.50937	6.01249	7.89141	9.01874
17	2.51539	2.68308	2.79488	3.35385	4.19231	4.79121	6.38828	8.38463	9.58241
18	2.66335	2.84091	2.95928	3.55114	4.43892	5.07304	6.76406	8.87784	10.14608
19	2.81132	2.99874	3.12368	3.74842	4.68553	5.35488	7.13984	9.37105	10.70976
20	2.95928	3.15657	3.28809	3.94571	4.93213	5.63671	7.51562	9.86427	11.27343
21	3.10724	3.31439	3.45249	4.14299	5.17874	5.91855	7.89140	10.35748	11.83710
22	3.25521	3.47222	3.61690	4.34028	5.42535	6.20039	8.26718	10.85069	12.40077
23	3.40317	3.63005	3.78130	4.53756	5.67195	6.48222	8.64296	11.34391	12.96444

Z80 Instructions by Mnemonic

ADC	A, A	8F	AND	C	A1
ADC	A, B	88	AND	D	A2
ADC	A, C	89	AND	E	A3
ADC	A, D	8A	AND	H	A4
ADC	A, E	8B	AND	HX	DDA4
ADC	A, H	8C	AND	HY	FDA4
ADC	A, HX	DD8C	AND	L	A5
ADC	A, HY	FD8C	AND	LX	DDA5
ADC	A, L	8D	AND	LY	FDA5
ADC	A, LX	DD8D	AND	n	E648
ADC	A, LY	FD8D	AND	(HL)	A6
ADC	A, n	CE48	AND	(IX+d)	DDA61E
ADC	A, (HL)	8E	AND	(IY+d)	FDA61E
ADC	A, (IX+d)	DD8E1E	BIT	0, A	CB47
ADC	A, (IY+d)	FD8E1E	BIT	0, B	CB40
ADC	HL, BC	ED4A	BIT	0, C	CB41
ADC	HL, DE	ED5A	BIT	0, D	CB42
ADC	HL, HL	ED6A	BIT	0, E	CB43
ADC	HL, SP	ED7A	BIT	0, H	CB44
ADD	A, A	87	BIT	0, L	CB45
ADD	A, B	80	BIT	0, (HL)	CB46
ADD	A, C	81	BIT	0, (IX+d)	DDCB1E46
ADD	A, D	82	BIT	0, (IY+d)	FDCB1E46
ADD	A, E	83	BIT	1, A	CB4F
ADD	A, H	84	BIT	1, B	CB48
ADD	A, HX	DD84	BIT	1, C	CB49
ADD	A, HY	FD84	BIT	1, D	CB4A
ADD	A, L	85	BIT	1, E	CB4B
ADD	A, LX	DD85	BIT	1, H	CB4C
ADD	A, LY	FD85	BIT	1, L	CB4D
ADD	A, n	C648	BIT	1, (HL)	CB4E
ADD	A, (HL)	86	BIT	1, (IX+d)	DDCB1E4E
ADD	A, (IX+d)	DD861E	BIT	1, (IY+d)	FDCB1E4E
ADD	A, (IY+d)	FD861E	BIT	2, A	CB57
ADD	HL, BC	09	BIT	2, B	CB50
ADD	HL, DE	19	BIT	2, C	CB51
ADD	HL, HL	29	BIT	2, D	CB52
ADD	HL, SP	39	BIT	2, E	CB53
ADD	IX, BC	DD09	BIT	2, H	CB54
ADD	IX, DE	DD19	BIT	2, L	CB55
ADD	IX, IX	DD29	BIT	2, (HL)	CB56
ADD	IX, SP	DD39	BIT	2, (IX+d)	DDCB1E56
ADD	IY, BC	FD09	BIT	2, (IY+d)	FDCB1E56
ADD	IY, DE	FD19	BIT	3, A	CB5F
ADD	IY, IY	FD29	BIT	3, B	CB58
ADD	IY, SP	FD39	BIT	3, C	CB59
AND	A	A7	BIT	3, D	CB5A
AND	B	A0	BIT	3, E	CB5B

Z80 Instructions by Mnemonic

BIT	3,H	CB5C	CALL	NC,mn	D4A00B
BIT	3,L	CB5D	CALL	NZ,mn	C4A00B
BIT	3,(HL)	CB5E	CALL	P,mn	F4A00B
BIT	3,(IX+d)	DDCB1E5E	CALL	PE,mn	ECA00B
BIT	3,(IY+d)	FDCB1E5E	CALL	PO,mn	E4A00B
BIT	4,A	CB67	CALL	Z,mn	CCA00B
BIT	4,B	CB60	CCF		3F
BIT	4,C	CB61	CP	A	BF
BIT	4,D	CB62	CP	B	B8
BIT	4,E	CB63	CP	C	B9
BIT	4,H	CB64	CP	D	BA
BIT	4,L	CB65	CP	E	BB
BIT	4,(HL)	CB66	CP	H	BC
BIT	4,(IX+d)	DDCB1E66	CP	HX	DDBC
BIT	4,(IY+d)	FDCB1E66	CP	HY	FDBC
BIT	5,A	CB6F	CP	L	BD
BIT	5,B	CB68	CP	LX	DDBD
BIT	5,C	CB69	CP	LY	FDBD
BIT	5,D	CB6A	CP	n	FE48
BIT	5,E	CB6B	CP	(HL)	BE
BIT	5,H	CB6C	CP	(IX+d)	DDBE1E
BIT	5,L	CB6D	CP	(IY+d)	FDDE1E
BIT	5,(HL)	CB6E	CPD		EDA9
BIT	5,(IX+d)	DDCB1E6E	CPDR		EDB9
BIT	5,(IY+d)	FDCB1E6E	CPI		EDA1
BIT	6,A	CB77	CPIR		EDB1
BIT	6,B	CB70	CPL		2F
BIT	6,C	CB71	DAA		27
BIT	6,D	CB72	DEC	A	3D
BIT	6,E	CB73	DEC	B	05
BIT	6,H	CB74	DEC	BC	0B
BIT	6,L	CB75	DEC	C	0D
BIT	6,(HL)	CB76	DEC	D	15
BIT	6,(IX+d)	DDCB1E76	DEC	DE	1B
BIT	6,(IY+d)	FDCB1E76	DEC	E	1D
BIT	7,A	CB7F	DEC	H	25
BIT	7,B	CB78	DEC	HL	2B
BIT	7,C	CB79	DEC	HX	DD25
BIT	7,D	CB7A	DEC	HY	FD25
BIT	7,E	CB7B	DEC	IX	DD2B
BIT	7,H	CB7C	DEC	IY	FD2B
BIT	7,L	CB7D	DEC	L	2D
BIT	7,(HL)	CB7E	DEC	LX	DD2D
BIT	7,(IX+d)	DDCB1E7E	DEC	LY	FD2D
BIT	7,(IY+d)	FDCB1E7E	DEC	SP	3B
CALL	C,mn	DCA00B	DEC	(HL)	35
CALL	M,mn	FCA00B	DEC	(IX+d)	DD351E
CALL	mn	CDA00B	DEC	(IY+d)	FD351E

Z80 Instructions by Mnemonic

DI		F3	JP	mn	C3A00B
DJNZ	e	101C	JP	NC, mn	D2A00B
EI		FB	JP	NZ, mn	C2A00B
EX	AF, AF'	08	JP	P, mn	F2A00B
EX	DE, HL	EB	JP	PE, mn	EAA00B
EX	(SP), HL	E3	JP	PO, mn	E2A00B
EX	(SP), IX	DDE3	JP	Z, mn	CAA00B
EX	(SP), IY	FDE3	JP	(HL)	E9
EXX		D9	JP	(IX)	DDE9
HALT		76	JP	(IY)	FDE9
IM	0	ED46	JR	C, e	38DE
IM	1	ED56	JR	e	1810
IM	2	ED5E	JR	NC, e	30EC
IN	A, (C)	ED78	JR	NZ, e	2006
IN	A, (n)	DB48	JR	Z, e	28F8
IN	B, (C)	ED40	LD	A, A	7F
IN	C, (C)	ED48	LD	A, B	78
IN	D, (C)	ED50	LD	A, C	79
IN	E, (C)	ED58	LD	A, D	7A
IN	H, (C)	ED60	LD	A, E	7B
IN	L, (C)	ED68	LD	A, H	7C
IN	(HL), (C)	ED70	LD	A, HX	DD7C
INC	A	3C	LD	A, HY	FD7C
INC	B	04	LD	A, I	ED57
INC	BC	03	LD	A, L	7D
INC	C	0C	LD	A, LX	DD7D
INC	D	14	LD	A, LY	FD7D
INC	DE	13	LD	A, n	3E48
INC	E	1C	LD	A, R	ED5F
INC	H	24	LD	A, (BC)	0A
INC	HL	23	LD	A, (DE)	1A
INC	HX	DD24	LD	A, (HL)	7E
INC	HY	FD24	LD	A, (IX+d)	DD7E1E
INC	IX	DD23	LD	A, (IY+d)	FD7E1E
INC	IY	FD23	LD	A, (mn)	3AA00B
INC	L	2C	LD	B, A	47
INC	LX	DD2C	LD	B, B	40
INC	LY	FD2C	LD	B, C	41
INC	SP	33	LD	B, D	42
INC	(HL)	34	LD	B, E	43
INC	(IX+d)	DD341E	LD	B, H	44
INC	(IY+d)	FD341E	LD	B, HX	DD44
IND		EDAA	LD	B, HY	FD44
INDR		EDBA	LD	B, L	45
INI		EDA2	LD	B, LX	DD45
INIR		EDB2	LD	B, LY	FD45
JP	C, mn	DAA00B	LD	B, n	0648
JP	M, mn	FAA00B	LD	B, (HL)	46

Z80 Instructions by Mnemonic

LD	B, (IX+d)	DD461E	LD	E, (HL)	5E
LD	B, (IY+d)	FD461E	LD	E, (IX+d)	DD5E1E
LD	BC, mn	01A00B	LD	E, (IY+d)	FD5E1E
LD	BC, (mn)	ED4BA00B	LD	H, A	67
LD	C, A	4F	LD	H, B	60
LD	C, B	48	LD	H, C	61
LD	C, C	49	LD	H, D	62
LD	C, D	4A	LD	H, E	63
LD	C, E	4B	LD	H, H	64
LD	C, H	4C	LD	H, L	65
LD	C, HX	DD4C	LD	H, n	2648
LD	C, HY	FD4C	LD	H, (HL)	66
LD	C, L	4D	LD	H, (IX+d)	DD661E
LD	C, LX	DD4D	LD	H, (IY+d)	FD661E
LD	C, LY	FD4D	LD	HL, mn	21A00B
LD	C, n	0E48	LD	HL, (mn)	2AA00B
LD	C, (HL)	4E	LD	HX, A	DD67
LD	C, (IX+d)	DD4E1E	LD	HX, B	DD60
LD	C, (IY+d)	FD4E1E	LD	HX, C	DD61
LD	D, A	57	LD	HX, D	DD62
LD	D, B	50	LD	HX, E	DD63
LD	D, C	51	LD	HX, HX	DD64
LD	D, D	52	LD	HX, LX	DD65
LD	D, E	53	LD	HX, n	DD2648
LD	D, H	54	LD	HY, A	FD67
LD	D, HX	DD54	LD	HY, B	FD60
LD	D, HY	FD54	LD	HY, C	FD61
LD	D, L	55	LD	HY, D	FD62
LD	D, LX	DD55	LD	HY, E	FD63
LD	D, LY	FD55	LD	HY, HY	FD64
LD	D, n	1648	LD	HY, LY	FD65
LD	D, (HL)	56	LD	HY, n	FD2648
LD	D, (IX+d)	DD561E	LD	I, A	ED47
LD	D, (IY+d)	FD561E	LD	IX, mn	DD21A00B
LD	DE, mn	11A00B	LD	IX, (mn)	DD2AA00B
LD	DE, (mn)	ED5BA00B	LD	IY, mn	FD21A00B
LD	E, A	5F	LD	IY, (mn)	FD2AA00B
LD	E, B	58	LD	L, A	6F
LD	E, C	59	LD	L, B	68
LD	E, D	5A	LD	L, C	69
LD	E, E	5B	LD	L, D	6A
LD	E, H	5C	LD	L, E	6B
LD	E, HX	DD5C	LD	L, H	6C
LD	E, HY	FD5C	LD	L, L	6D
LD	E, L	5D	LD	L, n	2E48
LD	E, LX	DD5D	LD	L, (HL)	6E
LD	E, LY	FD5D	LD	L, (IX+d)	DD6E1E
LD	E, n	1E48	LD	L, (IY+d)	FD6E1E

Z80 Instructions by Mnemonic

LD	LX,A	DD6F	LD	(mn),A	32A00B
LD	LX,B	DD68	LD	(mn),BC	ED43A00B
LD	LX,C	DD69	LD	(mn),DE	ED53A00B
LD	LX,D	DD6A	LD	(mn),HL	22A00B
LD	LX,E	DD6B	LD	(mn),IX	DD22A00B
LD	LX,HX	DD6C	LD	(mn),IY	FD22A00B
LD	LX,LX	DD6D	LD	(mn),SP	ED73A00B
LD	LX,n	DD2E48	LDD		EDA8
LD	LY,A	FD6F	LDDR		EDB8
LD	LY,B	FD68	LDI		EDA0
LD	LY,C	FD69	LDIR		EDB0
LD	LY,D	FD6A	NEG		ED44
LD	LY,E	FD6B	NOP		00
LD	LY,HY	FD6C	OR	A	B7
LD	LY,LY	FD6D	OR	B	B0
LD	LY,n	FD2E48	OR	C	B1
LD	R,A	ED4F	OR	D	B2
LD	SP,HL	F9	OR	E	B3
LD	SP,IX	DDF9	OR	H	B4
LD	SP,IY	FDF9	OR	HX	DDB4
LD	SP,mn	31A00B	OR	HY	FDB4
LD	SP,(mn)	ED7BA00B	OR	L	B5
LD	(BC),A	02	OR	LX	DDB5
LD	(DE),A	12	OR	LY	FDB5
LD	(HL),A	77	OR	n	F648
LD	(HL),B	70	OR	(HL)	B6
LD	(HL),C	71	OR	(IX+d)	DDB61E
LD	(HL),D	72	OR	(IY+d)	FDB61E
LD	(HL),E	73	OTDR		EDBB
LD	(HL),H	74	OTIR		EDB3
LD	(HL),L	75	OUT	(C),A	ED79
LD	(HL),n	3648	OUT	(C),B	ED41
LD	(IX+d),A	DD771E	OUT	(C),C	ED49
LD	(IX+d),B	DD701E	OUT	(C),D	ED51
LD	(IX+d),C	DD711E	OUT	(C),E	ED59
LD	(IX+d),D	DD721E	OUT	(C),H	ED61
LD	(IX+d),E	DD731E	OUT	(C),L	ED69
LD	(IX+d),H	DD741E	OUT	(C),(HL)	ED71
LD	(IX+d),L	DD751E	OUT	(n),A	D348
LD	(IX+d),n	DD361E48	OUTD		EDAB
LD	(IY+d),A	FD771E	OUTI		EDA3
LD	(IY+d),B	FD701E	POP	AF	F1
LD	(IY+d),C	FD711E	POP	BC	C1
LD	(IY+d),D	FD721E	POP	DE	D1
LD	(IY+d),E	FD731E	POP	HL	E1
LD	(IY+d),H	FD741E	POP	IX	DDE1
LD	(IY+d),L	FD751E	POP	IY	FDE1
LD	(IY+d),n	FD361E48	PUSH	AF	F5

Z80 Instructions by Mnemonic

PUSH	BC	C5	RES	1, L, (IX+d)	DDCB1E8D
PUSH	DE	D5	RES	1, L, (IY+d)	FDCB1E8D
PUSH	HL	E5	RES	1, (HL)	CB8E
PUSH	IX	DDE5	RES	1, (IX+d)	DDCB1E8E
PUSH	IY	FDE5	RES	1, (IY+d)	FDCB1E8E
RES	0, A	CB87	RES	2, A	CB97
RES	0, A, (IX+d)	DDCB1E87	RES	2, A, (IX+d)	DDCB1E97
RES	0, A, (IY+d)	FDCB1E87	RES	2, A, (IY+d)	FDCB1E97
RES	0, B	CB80	RES	2, B	CB90
RES	0, B, (IX+d)	DDCB1E80	RES	2, B, (IX+d)	DDCB1E90
RES	0, B, (IY+d)	FDCB1E80	RES	2, B, (IY+d)	FDCB1E90
RES	0, C	CB81	RES	2, C	CB91
RES	0, C, (IX+d)	DDCB1E81	RES	2, C, (IX+d)	DDCB1E91
RES	0, C, (IY+d)	FDCB1E81	RES	2, C, (IY+d)	FDCB1E91
RES	0, D	CB82	RES	2, D	CB92
RES	0, D, (IX+d)	DDCB1E82	RES	2, D, (IX+d)	DDCB1E92
RES	0, D, (IY+d)	FDCB1E82	RES	2, D, (IY+d)	FDCB1E92
RES	0, E	CB83	RES	2, E	CB93
RES	0, E, (IX+d)	DDCB1E83	RES	2, E, (IX+d)	DDCB1E93
RES	0, E, (IY+d)	FDCB1E83	RES	2, E, (IY+d)	FDCB1E93
RES	0, H	CB84	RES	2, H	CB94
RES	0, H, (IX+d)	DDCB1E84	RES	2, H, (IX+d)	DDCB1E94
RES	0, H, (IY+d)	FDCB1E84	RES	2, H, (IY+d)	FDCB1E94
RES	0, L	CB85	RES	2, L	CB95
RES	0, L, (IX+d)	DDCB1E85	RES	2, L, (IX+d)	DDCB1E95
RES	0, L, (IY+d)	FDCB1E85	RES	2, L, (IY+d)	FDCB1E95
RES	0, (HL)	CB86	RES	2, (HL)	CB96
RES	0, (IX+d)	DDCB1E86	RES	2, (IX+d)	DDCB1E96
RES	0, (IY+d)	FDCB1E86	RES	2, (IY+d)	FDCB1E96
RES	1, A	CB8F	RES	3, A	CB9F
RES	1, A, (IX+d)	DDCB1E8F	RES	3, A, (IX+d)	DDCB1E9F
RES	1, A, (IY+d)	FDCB1E8F	RES	3, A, (IY+d)	FDCB1E9F
RES	1, B	CB88	RES	3, B	CB98
RES	1, B, (IX+d)	DDCB1E88	RES	3, B, (IX+d)	DDCB1E98
RES	1, B, (IY+d)	FDCB1E88	RES	3, B, (IY+d)	FDCB1E98
RES	1, C	CB89	RES	3, C	CB99
RES	1, C, (IX+d)	DDCB1E89	RES	3, C, (IX+d)	DDCB1E99
RES	1, C, (IY+d)	FDCB1E89	RES	3, C, (IY+d)	FDCB1E99
RES	1, D	CB8A	RES	3, D	CB9A
RES	1, D, (IX+d)	DDCB1E8A	RES	3, D, (IX+d)	DDCB1E9A
RES	1, D, (IY+d)	FDCB1E8A	RES	3, D, (IY+d)	FDCB1E9A
RES	1, E	CB8B	RES	3, E	CB9B
RES	1, E, (IX+d)	DDCB1E8B	RES	3, E, (IX+d)	DDCB1E9B
RES	1, E, (IY+d)	FDCB1E8B	RES	3, E, (IY+d)	FDCB1E9B
RES	1, H	CB8C	RES	3, H	CB9C
RES	1, H, (IX+d)	DDCB1E8C	RES	3, H, (IX+d)	DDCB1E9C
RES	1, H, (IY+d)	FDCB1E8C	RES	3, H, (IY+d)	FDCB1E9C
RES	1, L	CB8D	RES	3, L	CB9D

Z80 Instructions by Mnemonic

RES	3, L, (IX+d)	DDCB1E9D	RES	5, L, (IX+d)	DDCB1EAD
RES	3, L, (IY+d)	FDCB1E9D	RES	5, L, (IY+d)	FDCB1EAD
RES	3, (HL)	CB9E	RES	5, (HL)	CBAE
RES	3, (IX+d)	DDCB1E9E	RES	5, (IX+d)	DDCB1EAE
RES	3, (IY+d)	FDCB1E9E	RES	5, (IY+d)	FDCB1EAE
RES	4, A	CBA7	RES	6, A	CBB7
RES	4, A, (IX+d)	DDCB1EA7	RES	6, A, (IX+d)	DDCB1EB7
RES	4, A, (IY+d)	FDCB1EA7	RES	6, A, (IY+d)	FDCB1EB7
RES	4, B	CBA0	RES	6, B	CBB0
RES	4, B, (IX+d)	DDCB1EA0	RES	6, B, (IX+d)	DDCB1EB0
RES	4, B, (IY+d)	FDCB1EA0	RES	6, B, (IY+d)	FDCB1EB0
RES	4, C	CBA1	RES	6, C	CBB1
RES	4, C, (IX+d)	DDCB1EA1	RES	6, C, (IX+d)	DDCB1EB1
RES	4, C, (IY+d)	FDCB1EA1	RES	6, C, (IY+d)	FDCB1EB1
RES	4, D	CBA2	RES	6, D	CBB2
RES	4, D, (IX+d)	DDCB1EA2	RES	6, D, (IX+d)	DDCB1EB2
RES	4, D, (IY+d)	FDCB1EA2	RES	6, D, (IY+d)	FDCB1EB2
RES	4, E	CBA3	RES	6, E	CBB3
RES	4, E, (IX+d)	DDCB1EA3	RES	6, E, (IX+d)	DDCB1EB3
RES	4, E, (IY+d)	FDCB1EA3	RES	6, E, (IY+d)	FDCB1EB3
RES	4, H	CBA4	RES	6, H	CBB4
RES	4, H, (IX+d)	DDCB1EA4	RES	6, H, (IX+d)	DDCB1EB4
RES	4, H, (IY+d)	FDCB1EA4	RES	6, H, (IY+d)	FDCB1EB4
RES	4, L	CBA5	RES	6, L	CBB5
RES	4, L, (IX+d)	DDCB1EA5	RES	6, L, (IX+d)	DDCB1EB5
RES	4, L, (IY+d)	FDCB1EA5	RES	6, L, (IY+d)	FDCB1EB5
RES	4, (HL)	CBA6	RES	6, (HL)	CBB6
RES	4, (IX+d)	DDCB1EA6	RES	6, (IX+d)	DDCB1EB6
RES	4, (IY+d)	FDCB1EA6	RES	6, (IY+d)	FDCB1EB6
RES	5, A	CBAF	RES	7, A	CBBF
RES	5, A, (IX+d)	DDCB1EAF	RES	7, A, (IX+d)	DDCB1EBF
RES	5, A, (IY+d)	FDCB1EAF	RES	7, A, (IY+d)	FDCB1EBF
RES	5, B	CBA8	RES	7, B	CBB8
RES	5, B, (IX+d)	DDCB1EA8	RES	7, B, (IX+d)	DDCB1EB8
RES	5, B, (IY+d)	FDCB1EA8	RES	7, B, (IY+d)	FDCB1EB8
RES	5, C	CBA9	RES	7, C	CBB9
RES	5, C, (IX+d)	DDCB1EA9	RES	7, C, (IX+d)	DDCB1EB9
RES	5, C, (IY+d)	FDCB1EA9	RES	7, C, (IY+d)	FDCB1EB9
RES	5, D	CBAA	RES	7, D	CBBA
RES	5, D, (IX+d)	DDCB1EAA	RES	7, D, (IX+d)	DDCB1EBA
RES	5, D, (IY+d)	FDCB1EAA	RES	7, D, (IY+d)	FDCB1EBA
RES	5, E	CBAB	RES	7, E	CBBB
RES	5, E, (IX+d)	DDCB1EAB	RES	7, E, (IX+d)	DDCB1EBB
RES	5, E, (IY+d)	FDCB1EAB	RES	7, E, (IY+d)	FDCB1EBB
RES	5, H	CBAC	RES	7, H	CBBC
RES	5, H, (IX+d)	DDCB1EAC	RES	7, H, (IX+d)	DDCB1EBC
RES	5, H, (IY+d)	FDCB1EAC	RES	7, H, (IY+d)	FDCB1EBC
RES	5, L	CBAD	RES	7, L	CBBD

Z80 Instructions by Mnemonic

RES	7, L, (IX+d)	DDCB1EBD	RLC	C, (IX+d)	DDCB1E01
RES	7, L, (IY+d)	FDCB1EBD	RLC	C, (IY+d)	FDCB1E01
RES	7, (HL)	CBBE	RLC	D	CB02
RES	7, (IX+d)	DDCB1EBE	RLC	D, (IX+d)	DDCB1E02
RES	7, (IY+d)	FDCB1EBE	RLC	D, (IY+d)	FDCB1E02
RET	C	D8	RLC	E	CB03
RET	M	F8	RLC	E, (IX+d)	DDCB1E03
RET	NC	D0	RLC	E, (IY+d)	FDCB1E03
RET	NZ	C0	RLC	H	CB04
RET	P	F0	RLC	H, (IX+d)	DDCB1E04
RET	PE	E8	RLC	H, (IY+d)	FDCB1E04
RET	PO	E0	RLC	L	CB05
RET	Z	C8	RLC	L, (IX+d)	DDCB1E05
RET		C9	RLC	L, (IY+d)	FDCB1E05
RETI		ED4D	RLC	(HL)	CB06
RETN		ED45	RLC	(IX+d)	DDCB1E06
RL	A	CB17	RLC	(IY+d)	FDCB1E06
RL	A, (IX+d)	DDCB1E17	RLCA		07
RL	A, (IY+d)	FDCB1E17	RLD		ED6F
RL	B	CB10	RR	A	CB1F
RL	B, (IX+d)	DDCB1E10	RR	A, (IX+d)	DDCB1E1F
RL	B, (IY+d)	FDCB1E10	RR	A, (IY+d)	FDCB1E1F
RL	C	CB11	RR	B	CB18
RL	C, (IX+d)	DDCB1E11	RR	B, (IX+d)	DDCB1E18
RL	C, (IY+d)	FDCB1E11	RR	B, (IY+d)	FDCB1E18
RL	D	CB12	RR	C	CB19
RL	D, (IX+d)	DDCB1E12	RR	C, (IX+d)	DDCB1E19
RL	D, (IY+d)	FDCB1E12	RR	C, (IY+d)	FDCB1E19
RL	E	CB13	RR	D	CB1A
RL	E, (IX+d)	DDCB1E13	RR	D, (IX+d)	DDCB1E1A
RL	E, (IY+d)	FDCB1E13	RR	D, (IY+d)	FDCB1E1A
RL	H	CB14	RR	E	CB1B
RL	H, (IX+d)	DDCB1E14	RR	E, (IX+d)	DDCB1E1B
RL	H, (IY+d)	FDCB1E14	RR	E, (IY+d)	FDCB1E1B
RL	L	CB15	RR	H	CB1C
RL	L, (IX+d)	DDCB1E15	RR	H, (IX+d)	DDCB1E1C
RL	L, (IY+d)	FDCB1E15	RR	H, (IY+d)	FDCB1E1C
RL	(HL)	CB16	RR	L	CB1D
RL	(IX+d)	DDCB1E16	RR	L, (IX+d)	DDCB1E1D
RL	(IY+d)	FDCB1E16	RR	L, (IY+d)	FDCB1E1D
RLA		17	RR	(HL)	CB1E
RLC	A	CB07	RR	(IX+d)	DDCB1E1E
RLC	A, (IX+d)	DDCB1E07	RR	(IY+d)	FDCB1E1E
RLC	A, (IY+d)	FDCB1E07	RRA		1F
RLC	B	CB00	RRC	A	CB0F
RLC	B, (IX+d)	DDCB1E00	RRC	A, (IX+d)	DDCB1E0F
RLC	B, (IY+d)	FDCB1E00	RRC	A, (IY+d)	FDCB1E0F
RLC	C	CB01	RRC	B	CB08

Z80 Instructions by Mnemonic

RRC	B, (IX+d)	DDCB1E08	SBC	HL, SP	ED72
RRC	B, (IY+d)	FDCB1E08	SCF		37
RRC	C	CB09	SET	0, A	CBC7
RRC	C, (IX+d)	DDCB1E09	SET	0, A, (IX+d)	DDCB1EC7
RRC	C, (IY+d)	FDCB1E09	SET	0, A, (IY+d)	FDCB1EC7
RRC	D	CB0A	SET	0, B	CBC0
RRC	D, (IX+d)	DDCB1E0A	SET	0, B, (IX+d)	DDCB1EC0
RRC	D, (IY+d)	FDCB1E0A	SET	0, B, (IY+d)	FDCB1EC0
RRC	E	CB0B	SET	0, C	CBC1
RRC	E, (IX+d)	DDCB1E0B	SET	0, C, (IX+d)	DDCB1EC1
RRC	E, (IY+d)	FDCB1E0B	SET	0, C, (IY+d)	FDCB1EC1
RRC	H	CB0C	SET	0, D	CBC2
RRC	H, (IX+d)	DDCB1E0C	SET	0, D, (IX+d)	DDCB1EC2
RRC	H, (IY+d)	FDCB1E0C	SET	0, D, (IY+d)	FDCB1EC2
RRC	L	CB0D	SET	0, E	CBC3
RRC	L, (IX+d)	DDCB1E0D	SET	0, E, (IX+d)	DDCB1EC3
RRC	L, (IY+d)	FDCB1E0D	SET	0, E, (IY+d)	FDCB1EC3
RRC	(HL)	CB0E	SET	0, H	CBC4
RRC	(IX+d)	DDCB1E0E	SET	0, H, (IX+d)	DDCB1EC4
RRC	(IY+d)	FDCB1E0E	SET	0, H, (IY+d)	FDCB1EC4
RRCA		0F	SET	0, L	CBC5
RRD		ED67	SET	0, L, (IX+d)	DDCB1EC5
RST	0	C7	SET	0, L, (IY+d)	FDCB1EC5
RST	10H	D7	SET	0, (HL)	CBC6
RST	18H	DF	SET	0, (IX+d)	DDCB1EC6
RST	20H	E7	SET	0, (IY+d)	FDCB1EC6
RST	28H	EF	SET	1, A	CBCF
RST	30H	F7	SET	1, A, (IX+d)	DDCB1ECF
RST	38H	FF	SET	1, A, (IY+d)	FDCB1ECF
RST	8	CF	SET	1, B	CBC8
SBC	A, A	9F	SET	1, B, (IX+d)	DDCB1EC8
SBC	A, B	98	SET	1, B, (IY+d)	FDCB1EC8
SBC	A, C	99	SET	1, C	CBC9
SBC	A, D	9A	SET	1, C, (IX+d)	DDCB1EC9
SBC	A, E	9B	SET	1, C, (IY+d)	FDCB1EC9
SBC	A, H	9C	SET	1, D	CBCA
SBC	A, HX	DD9C	SET	1, D, (IX+d)	DDCB1ECA
SBC	A, HY	FD9C	SET	1, D, (IY+d)	FDCB1ECA
SBC	A, L	9D	SET	1, E	CBCB
SBC	A, LX	DD9D	SET	1, E, (IX+d)	DDCB1ECB
SBC	A, LY	FD9D	SET	1, E, (IY+d)	FDCB1ECB
SBC	A, n	DE48	SET	1, H	CBCC
SBC	A, (HL)	9E	SET	1, H, (IX+d)	DDCB1ECC
SBC	A, (IX+d)	DD9E1E	SET	1, H, (IY+d)	FDCB1ECC
SBC	A, (IY+d)	FD9E1E	SET	1, L	CBCD
SBC	HL, BC	ED42	SET	1, L, (IX+d)	DDCB1ECD
SBC	HL, DE	ED52	SET	1, L, (IY+d)	FDCB1ECD
SBC	HL, HL	ED62	SET	1, (HL)	CBCE

Z80 Instructions by Mnemonic

SET	1, (IX+d)	DDCB1ECE	SET	3, (IX+d)	DDCB1EDE
SET	1, (IY+d)	FDCB1ECE	SET	3, (IY+d)	FDCB1EDE
SET	2,A	CBD7	SET	4,A	CBE7
SET	2,A, (IX+d)	DDCB1ED7	SET	4,A, (IX+d)	DDCB1EE7
SET	2,A, (IY+d)	FDCB1ED7	SET	4,A, (IY+d)	FDCB1EE7
SET	2,B	CBD0	SET	4,B	CBE0
SET	2,B, (IX+d)	DDCB1ED0	SET	4,B, (IX+d)	DDCB1EE0
SET	2,B, (IY+d)	FDCB1ED0	SET	4,B, (IY+d)	FDCB1EE0
SET	2,C	CBD1	SET	4,C	CBE1
SET	2,C, (IX+d)	DDCB1ED1	SET	4,C, (IX+d)	DDCB1EE1
SET	2,C, (IY+d)	FDCB1ED1	SET	4,C, (IY+d)	FDCB1EE1
SET	2,D	CBD2	SET	4,D	CBE2
SET	2,D, (IX+d)	DDCB1ED2	SET	4,D, (IX+d)	DDCB1EE2
SET	2,D, (IY+d)	FDCB1ED2	SET	4,D, (IY+d)	FDCB1EE2
SET	2,E	CBD3	SET	4,E	CBE3
SET	2,E, (IX+d)	DDCB1ED3	SET	4,E, (IX+d)	DDCB1EE3
SET	2,E, (IY+d)	FDCB1ED3	SET	4,E, (IY+d)	FDCB1EE3
SET	2,H	CBD4	SET	4,H	CBE4
SET	2,H, (IX+d)	DDCB1ED4	SET	4,H, (IX+d)	DDCB1EE4
SET	2,H, (IY+d)	FDCB1ED4	SET	4,H, (IY+d)	FDCB1EE4
SET	2,L	CBD5	SET	4,L	CBE5
SET	2,L, (IX+d)	DDCB1ED5	SET	4,L, (IX+d)	DDCB1EE5
SET	2,L, (IY+d)	FDCB1ED5	SET	4,L, (IY+d)	FDCB1EE5
SET	2, (HL)	CBD6	SET	4, (HL)	CBE6
SET	2, (IX+d)	DDCB1ED6	SET	4, (IX+d)	DDCB1EE6
SET	2, (IY+d)	FDCB1ED6	SET	4, (IY+d)	FDCB1EE6
SET	3,A	CBDF	SET	5,A	CBEF
SET	3,A, (IX+d)	DDCB1EDF	SET	5,A, (IX+d)	DDCB1EEF
SET	3,A, (IY+d)	FDCB1EDF	SET	5,A, (IY+d)	FDCB1EEF
SET	3,B	CBD8	SET	5,B	CBE8
SET	3,B, (IX+d)	DDCB1ED8	SET	5,B, (IX+d)	DDCB1EE8
SET	3,B, (IY+d)	FDCB1ED8	SET	5,B, (IY+d)	FDCB1EE8
SET	3,C	CBD9	SET	5,C	CBE9
SET	3,C, (IX+d)	DDCB1ED9	SET	5,C, (IX+d)	DDCB1EE9
SET	3,C, (IY+d)	FDCB1ED9	SET	5,C, (IY+d)	FDCB1EE9
SET	3,D	CBDA	SET	5,D	CBEA
SET	3,D, (IX+d)	DDCB1EDA	SET	5,D, (IX+d)	DDCB1EEA
SET	3,D, (IY+d)	FDCB1EDA	SET	5,D, (IY+d)	FDCB1EEA
SET	3,E	CBDB	SET	5,E	CBEB
SET	3,E, (IX+d)	DDCB1EDB	SET	5,E, (IX+d)	DDCB1EEB
SET	3,E, (IY+d)	FDCB1EDB	SET	5,E, (IY+d)	FDCB1EEB
SET	3,H	CBDC	SET	5,H	CBEC
SET	3,H, (IX+d)	DDCB1EDC	SET	5,H, (IX+d)	DDCB1EEC
SET	3,H, (IY+d)	FDCB1EDC	SET	5,H, (IY+d)	FDCB1EEC
SET	3,L	CBDD	SET	5,L	CBED
SET	3,L, (IX+d)	DDCB1EDD	SET	5,L, (IX+d)	DDCB1EED
SET	3,L, (IY+d)	FDCB1EDD	SET	5,L, (IY+d)	FDCB1EED
SET	3, (HL)	CBDE	SET	5, (HL)	CBEE

Z80 Instructions by Mnemonic

SET	5, (IX+d)	DDCB1EEE	SET	7, (IX+d)	DDCB1EFE
SET	5, (IY+d)	FDCB1EEE	SET	7, (IY+d)	FDCB1EFE
SET	6,A	CBF7	SLA	A	CB27
SET	6,A, (IX+d)	DDCB1EF7	SLA	A, (IX+d)	DDCB1E27
SET	6,A, (IY+d)	FDCB1EF7	SLA	A, (IY+d)	FDCB1E27
SET	6,B	CBF0	SLA	B	CB20
SET	6,B, (IX+d)	DDCB1EF0	SLA	B, (IX+d)	DDCB1E20
SET	6,B, (IY+d)	FDCB1EF0	SLA	B, (IY+d)	FDCB1E20
SET	6,C	CBF1	SLA	C	CB21
SET	6,C, (IX+d)	DDCB1EF1	SLA	C, (IX+d)	DDCB1E21
SET	6,C, (IY+d)	FDCB1EF1	SLA	C, (IY+d)	FDCB1E21
SET	6,D	CBF2	SLA	D	CB22
SET	6,D, (IX+d)	DDCB1EF2	SLA	D, (IX+d)	DDCB1E22
SET	6,D, (IY+d)	FDCB1EF2	SLA	D, (IY+d)	FDCB1E22
SET	6,E	CBF3	SLA	E	CB23
SET	6,E, (IX+d)	DDCB1EF3	SLA	E, (IX+d)	DDCB1E23
SET	6,E, (IY+d)	FDCB1EF3	SLA	E, (IY+d)	FDCB1E23
SET	6,H	CBF4	SLA	H	CB24
SET	6,H, (IX+d)	DDCB1EF4	SLA	H, (IX+d)	DDCB1E24
SET	6,H, (IY+d)	FDCB1EF4	SLA	H, (IY+d)	FDCB1E24
SET	6,L	CBF5	SLA	L	CB25
SET	6,L, (IX+d)	DDCB1EF5	SLA	L, (IX+d)	DDCB1E25
SET	6,L, (IY+d)	FDCB1EF5	SLA	L, (IY+d)	FDCB1E25
SET	6, (HL)	CBF6	SLA	(HL)	CB26
SET	6, (IX+d)	DDCB1EF6	SLA	(IX+d)	DDCB1E26
SET	6, (IY+d)	FDCB1EF6	SLA	(IY+d)	FDCB1E26
SET	7,A	CBFF	SLL	A	CB37
SET	7,A, (IX+d)	DDCB1EFF	SLL	A, (IX+d)	DDCB1E37
SET	7,A, (IY+d)	FDCB1EFF	SLL	A, (IY+d)	FDCB1E37
SET	7,B	CBF8	SLL	B	CB30
SET	7,B, (IX+d)	DDCB1EF8	SLL	B, (IX+d)	DDCB1E30
SET	7,B, (IY+d)	FDCB1EF8	SLL	B, (IY+d)	FDCB1E30
SET	7,C	CBF9	SLL	C	CB31
SET	7,C, (IX+d)	DDCB1EF9	SLL	C, (IX+d)	DDCB1E31
SET	7,C, (IY+d)	FDCB1EF9	SLL	C, (IY+d)	FDCB1E31
SET	7,D	CBFA	SLL	D	CB32
SET	7,D, (IX+d)	DDCB1EFA	SLL	D, (IX+d)	DDCB1E32
SET	7,D, (IY+d)	FDCB1EFA	SLL	D, (IY+d)	FDCB1E32
SET	7,E	CBFB	SLL	E	CB33
SET	7,E, (IX+d)	DDCB1EFB	SLL	E, (IX+d)	DDCB1E33
SET	7,E, (IY+d)	FDCB1EFB	SLL	E, (IY+d)	FDCB1E33
SET	7,H	CBFC	SLL	H	CB34
SET	7,H, (IX+d)	DDCB1EFC	SLL	H, (IX+d)	DDCB1E34
SET	7,H, (IY+d)	FDCB1EFC	SLL	H, (IY+d)	FDCB1E34
SET	7,L	CBFD	SLL	L	CB35
SET	7,L, (IX+d)	DDCB1EFD	SLL	L, (IX+d)	DDCB1E35
SET	7,L, (IY+d)	FDCB1EFD	SLL	L, (IY+d)	FDCB1E35
SET	7, (HL)	CBFE	SLL	(HL)	CB36

Z80 Instructions by Mnemonic

SLL	(IX+d)	DDCB1E36	SRL	(IX+d)	DDCB1E3E
SLL	(IY+d)	FDCB1E36	SRL	(IY+d)	FDCB1E3E
SRA	A	CB2F	SUB	A	97
SRA	A, (IX+d)	DDCB1E2F	SUB	B	90
SRA	A, (IY+d)	FDCB1E2F	SUB	C	91
SRA	B	CB28	SUB	D	92
SRA	B, (IX+d)	DDCB1E28	SUB	E	93
SRA	B, (IY+d)	FDCB1E28	SUB	H	94
SRA	C	CB29	SUB	HX	DD94
SRA	C, (IX+d)	DDCB1E29	SUB	HY	FD94
SRA	C, (IY+d)	FDCB1E29	SUB	L	95
SRA	D	CB2A	SUB	LX	DD95
SRA	D, (IX+d)	DDCB1E2A	SUB	LY	FD95
SRA	D, (IY+d)	FDCB1E2A	SUB	n	D648
SRA	E	CB2B	SUB	(HL)	96
SRA	E, (IX+d)	DDCB1E2B	SUB	(IX+d)	DD961E
SRA	E, (IY+d)	FDCB1E2B	SUB	(IY+d)	FD961E
SRA	H	CB2C	XOR	A	AF
SRA	H, (IX+d)	DDCB1E2C	XOR	B	A8
SRA	H, (IY+d)	FDCB1E2C	XOR	C	A9
SRA	L	CB2D	XOR	D	AA
SRA	L, (IX+d)	DDCB1E2D	XOR	E	AB
SRA	L, (IY+d)	FDCB1E2D	XOR	H	AC
SRA	(HL)	CB2E	XOR	HX	DDAC
SRA	(IX+d)	DDCB1E2E	XOR	HY	FDAC
SRA	(IY+d)	FDCB1E2E	XOR	L	AD
SRL	A	CB3F	XOR	LX	DDAD
SRL	A, (IX+d)	DDCB1E3F	XOR	LY	FDAD
SRL	A, (IY+d)	FDCB1E3F	XOR	n	EE48
SRL	B	CB38	XOR	(HL)	AE
SRL	B, (IX+d)	DDCB1E38	XOR	(IX+d)	DDAE1E
SRL	B, (IY+d)	FDCB1E38	XOR	(IY+d)	FDAE1E
SRL	C	CB39	mn	DEFS	12<-2#1
SRL	C, (IX+d)	DDCB1E39	d	EQU	3C00H/200H
SRL	C, (IY+d)	FDCB1E39	n	EQU	40H!8
SRL	D	CB3A	e	EQU	9%7+30H
SRL	D, (IX+d)	DDCB1E3A			
SRL	D, (IY+d)	FDCB1E3A			
SRL	E	CB3B			
SRL	E, (IX+d)	DDCB1E3B			
SRL	E, (IY+d)	FDCB1E3B			
SRL	H	CB3C			
SRL	H, (IX+d)	DDCB1E3C			
SRL	H, (IY+d)	FDCB1E3C			
SRL	L	CB3D			
SRL	L, (IX+d)	DDCB1E3D			
SRL	L, (IY+d)	FDCB1E3D			
SRL	(HL)	CB3E			

Z80 Instructions by Object Code

00	NOP		30EC	JR	NC, e
01A00B	LD	BC, mn	31A00B	LD	SP, mn
02	LD	(BC), A	32A00B	LD	(mn), A
03	INC	BC	33	INC	SP
04	INC	B	34	INC	(HL)
05	DEC	B	35	DEC	(HL)
0648	LD	B, n	3648	LD	(HL), n
07	RLCA		37	SCF	
08	EX	AF, AF'	38DE	JR	C, e
09	ADD	HL, BC	39	ADD	HL, SP
0A	LD	A, (BC)	3AA00B	LD	A, (mn)
0B	DEC	BC	3B	DEC	SP
0C	INC	C	3C	INC	A
0D	DEC	C	3D	DEC	A
0E48	LD	C, n	3E48	LD	A, n
0F	RRCA		3F	CCF	
101C	DJNZ	e	40	LD	B, B
11A00B	LD	DE, mn	41	LD	B, C
12	LD	(DE), A	42	LD	B, D
13	INC	DE	43	LD	B, E
14	INC	D	44	LD	B, H
15	DEC	D	45	LD	B, L
1648	LD	D, n	46	LD	B, (HL)
17	RLA		47	LD	B, A
1810	JR	e	48	LD	C, B
19	ADD	HL, DE	49	LD	C, C
1A	LD	A, (DE)	4A	LD	C, D
1B	DEC	DE	4B	LD	C, E
1C	INC	E	4C	LD	C, H
1D	DEC	E	4D	LD	C, L
1E48	LD	E, n	4E	LD	C, (HL)
1F	RRA		4F	LD	C, A
2006	JR	NZ, e	50	LD	D, B
21A00B	LD	HL, mn	51	LD	D, C
22A00B	LD	(mn), HL	52	LD	D, D
23	INC	HL	53	LD	D, E
24	INC	H	54	LD	D, H
25	DEC	H	55	LD	D, L
2648	LD	H, n	56	LD	D, (HL)
27	DAA		57	LD	D, A
28F8	JR	Z, e	58	LD	E, B
29	ADD	HL, HL	59	LD	E, C
2AA00B	LD	HL, (mn)	5A	LD	E, D
2B	DEC	HL	5B	LD	E, E
2C	INC	L	5C	LD	E, H
2D	DEC	L	5D	LD	E, L
2E48	LD	L, n	5E	LD	E, (HL)
2F	CPL		5F	LD	E, A

Z80 Instructions by Object Code

60	LD	H, B	90	SUB	B
61	LD	H, C	91	SUB	C
62	LD	H, D	92	SUB	D
63	LD	H, E	93	SUB	E
64	LD	H, H	94	SUB	H
65	LD	H, L	95	SUB	L
66	LD	H, (HL)	96	SUB	(HL)
67	LD	H, A	97	SUB	A
68	LD	L, B	98	SBC	A, B
69	LD	L, C	99	SBC	A, C
6A	LD	L, D	9A	SBC	A, D
6B	LD	L, E	9B	SBC	A, E
6C	LD	L, H	9C	SBC	A, H
6D	LD	L, L	9D	SBC	A, L
6E	LD	L, (HL)	9E	SBC	A, (HL)
6F	LD	L, A	9F	SBC	A, A
70	LD	(HL), B	A0	AND	B
71	LD	(HL), C	A1	AND	C
72	LD	(HL), D	A2	AND	D
73	LD	(HL), E	A3	AND	E
74	LD	(HL), H	A4	AND	H
75	LD	(HL), L	A5	AND	L
76	HALT		A6	AND	(HL)
77	LD	(HL), A	A7	AND	A
78	LD	A, B	A8	XOR	B
79	LD	A, C	A9	XOR	C
7A	LD	A, D	AA	XOR	D
7B	LD	A, E	AB	XOR	E
7C	LD	A, H	AC	XOR	H
7D	LD	A, L	AD	XOR	L
7E	LD	A, (HL)	AE	XOR	(HL)
7F	LD	A, A	AF	XOR	A
80	ADD	A, B	B0	OR	B
81	ADD	A, C	B1	OR	C
82	ADD	A, D	B2	OR	D
83	ADD	A, E	B3	OR	E
84	ADD	A, H	B4	OR	H
85	ADD	A, L	B5	OR	L
86	ADD	A, (HL)	B6	OR	(HL)
87	ADD	A, A	B7	OR	A
88	ADC	A, B	B8	CP	B
89	ADC	A, C	B9	CP	C
8A	ADC	A, D	BA	CP	D
8B	ADC	A, E	BB	CP	E
8C	ADC	A, H	BC	CP	H
8D	ADC	A, L	BD	CP	L
8E	ADC	A, (HL)	BE	CP	(HL)
8F	ADC	A, A	BF	CP	A

Z80 Instructions by Object Code

C0	RET	NZ	CB25	SLA	L
C1	POP	BC	CB26	SLA	(HL)
C2A00B	JP	NZ, mn	CB27	SLA	A
C3A00B	JP	mn	CB28	SRA	B
C4A00B	CALL	NZ, mn	CB29	SRA	C
C5	PUSH	BC	CB2A	SRA	D
C648	ADD	A, n	CB2B	SRA	E
C7	RST	0	CB2C	SRA	H
C8	RET	Z	CB2D	SRA	L
C9	RET		CB2E	SRA	(HL)
CAA00B	JP	Z, mn	CB2F	SRA	A
CB00	RLC	B	CB30	SLL	B
CB01	RLC	C	CB31	SLL	C
CB02	RLC	D	CB32	SLL	D
CB03	RLC	E	CB33	SLL	E
CB04	RLC	H	CB34	SLL	H
CB05	RLC	L	CB35	SLL	L
CB06	RLC	(HL)	CB36	SLL	(HL)
CB07	RLC	A	CB37	SLL	A
CB08	RRC	B	CB38	SRL	B
CB09	RRC	C	CB39	SRL	C
CB0A	RRC	D	CB3A	SRL	D
CB0B	RRC	E	CB3B	SRL	E
CB0C	RRC	H	CB3C	SRL	H
CB0D	RRC	L	CB3D	SRL	L
CB0E	RRC	(HL)	CB3E	SRL	(HL)
CB0F	RRC	A	CB3F	SRL	A
CB10	RL	B	CB40	BIT	0, B
CB11	RL	C	CB41	BIT	0, C
CB12	RL	D	CB42	BIT	0, D
CB13	RL	E	CB43	BIT	0, E
CB14	RL	H	CB44	BIT	0, H
CB15	RL	L	CB45	BIT	0, L
CB16	RL	(HL)	CB46	BIT	0, (HL)
CB17	RL	A	CB47	BIT	0, A
CB18	RR	B	CB48	BIT	1, B
CB19	RR	C	CB49	BIT	1, C
CB1A	RR	D	CB4A	BIT	1, D
CB1B	RR	E	CB4B	BIT	1, E
CB1C	RR	H	CB4C	BIT	1, H
CB1D	RR	L	CB4D	BIT	1, L
CB1E	RR	(HL)	CB4E	BIT	1, (HL)
CB1F	RR	A	CB4F	BIT	1, A
CB20	SLA	B	CB50	BIT	2, B
CB21	SLA	C	CB51	BIT	2, C
CB22	SLA	D	CB52	BIT	2, D
CB23	SLA	E	CB53	BIT	2, E
CB24	SLA	H	CB54	BIT	2, H

Z80 Instructions by Object Code

CB55	BIT	2,L	CB85	RES	0,L
CB56	BIT	2,(HL)	CB86	RES	0,(HL)
CB57	BIT	2,A	CB87	RES	0,A
CB58	BIT	3,B	CB88	RES	1,B
CB59	BIT	3,C	CB89	RES	1,C
CB5A	BIT	3,D	CB8A	RES	1,D
CB5B	BIT	3,E	CB8B	RES	1,E
CB5C	BIT	3,H	CB8C	RES	1,H
CB5D	BIT	3,L	CB8D	RES	1,L
CB5E	BIT	3,(HL)	CB8E	RES	1,(HL)
CB5F	BIT	3,A	CB8F	RES	1,A
CB60	BIT	4,B	CB90	RES	2,B
CB61	BIT	4,C	CB91	RES	2,C
CB62	BIT	4,D	CB92	RES	2,D
CB63	BIT	4,E	CB93	RES	2,E
CB64	BIT	4,H	CB94	RES	2,H
CB65	BIT	4,L	CB95	RES	2,L
CB66	BIT	4,(HL)	CB96	RES	2,(HL)
CB67	BIT	4,A	CB97	RES	2,A
CB68	BIT	5,B	CB98	RES	3,B
CB69	BIT	5,C	CB99	RES	3,C
CB6A	BIT	5,D	CB9A	RES	3,D
CB6B	BIT	5,E	CB9B	RES	3,E
CB6C	BIT	5,H	CB9C	RES	3,H
CB6D	BIT	5,L	CB9D	RES	3,L
CB6E	BIT	5,(HL)	CB9E	RES	3,(HL)
CB6F	BIT	5,A	CB9F	RES	3,A
CB70	BIT	6,B	CBA0	RES	4,B
CB71	BIT	6,C	CBA1	RES	4,C
CB72	BIT	6,D	CBA2	RES	4,D
CB73	BIT	6,E	CBA3	RES	4,E
CB74	BIT	6,H	CBA4	RES	4,H
CB75	BIT	6,L	CBA5	RES	4,L
CB76	BIT	6,(HL)	CBA6	RES	4,(HL)
CB77	BIT	6,A	CBA7	RES	4,A
CB78	BIT	7,B	CBA8	RES	5,B
CB79	BIT	7,C	CBA9	RES	5,C
CB7A	BIT	7,D	CBAA	RES	5,D
CB7B	BIT	7,E	CBAB	RES	5,E
CB7C	BIT	7,H	CBAC	RES	5,H
CB7D	BIT	7,L	CBAD	RES	5,L
CB7E	BIT	7,(HL)	CBAE	RES	5,(HL)
CB7F	BIT	7,A	CBAF	RES	5,A
CB80	RES	0,B	CBB0	RES	6,B
CB81	RES	0,C	CBB1	RES	6,C
CB82	RES	0,D	CBB2	RES	6,D
CB83	RES	0,E	CBB3	RES	6,E
CB84	RES	0,H	CBB4	RES	6,H

Z80 Instructions by Object Code

CBB5	RES	6, L	CBE5	SET	4, L
CBB6	RES	6, (HL)	CBE6	SET	4, (HL)
CBB7	RES	6, A	CBE7	SET	4, A
CBB8	RES	7, B	CBE8	SET	5, B
CBB9	RES	7, C	CBE9	SET	5, C
CBBA	RES	7, D	CBEA	SET	5, D
CBBB	RES	7, E	CBEB	SET	5, E
CBBC	RES	7, H	CBEC	SET	5, H
CBBD	RES	7, L	CBED	SET	5, L
CBBE	RES	7, (HL)	CBEE	SET	5, (HL)
CBBF	RES	7, A	CBEF	SET	5, A
CBC0	SET	0, B	CBF0	SET	6, B
CBC1	SET	0, C	CBF1	SET	6, C
CBC2	SET	0, D	CBF2	SET	6, D
CBC3	SET	0, E	CBF3	SET	6, E
CBC4	SET	0, H	CBF4	SET	6, H
CBC5	SET	0, L	CBF5	SET	6, L
CBC6	SET	0, (HL)	CBF6	SET	6, (HL)
CBC7	SET	0, A	CBF7	SET	6, A
CBC8	SET	1, B	CBF8	SET	7, B
CBC9	SET	1, C	CBF9	SET	7, C
CBCA	SET	1, D	CBFA	SET	7, D
CBCB	SET	1, E	CBFB	SET	7, E
CBCC	SET	1, H	CBFC	SET	7, H
CBCD	SET	1, L	CBFD	SET	7, L
CBCE	SET	1, (HL)	CBFE	SET	7, (HL)
CBCF	SET	1, A	CBFF	SET	7, A
CBD0	SET	2, B	CCA00B	CALL	Z, mn
CBD1	SET	2, C	CDA00B	CALL	mn
CBD2	SET	2, D	CE48	ADC	A, n
CBD3	SET	2, E	CF	RST	8
CBD4	SET	2, H	D0	RET	NC
CBD5	SET	2, L	D1	POP	DE
CBD6	SET	2, (HL)	D2A00B	JP	NC, mn
CBD7	SET	2, A	D348	OUT	(n), A
CBD8	SET	3, B	D4A00B	CALL	NC, mn
CBD9	SET	3, C	D5	PUSH	DE
CBDA	SET	3, D	D648	SUB	n
CBDB	SET	3, E	D7	RST	10H
CBDC	SET	3, H	D8	RET	C
CBDD	SET	3, L	D9	EXX	
CBDE	SET	3, (HL)	DAA00B	JP	C, mn
CBDF	SET	3, A	DB48	IN	A, (n)
CBE0	SET	4, B	DCA00B	CALL	C, mn
CBE1	SET	4, C	DD09	ADD	IX, BC
CBE2	SET	4, D	DD19	ADD	IX, DE
CBE3	SET	4, E	DD21A00B	LD	IX, mn
CBE4	SET	4, H	DD22A00B	LD	(mn), IX

Z80 Instructions by Object Code

DD23	INC	IX	DD771E	LD	(IX+d),A
DD24	INC	HX	DD7C	LD	A,HX
DD25	DEC	HX	DD7D	LD	A,LX
DD2648	LD	HX,n	DD7E1E	LD	A,(IX+d)
DD29	ADD	IX,IX	DD84	ADD	A,HX
DD2AA00B	LD	IX,(mn)	DD85	ADD	A,LX
DD2B	DEC	IX	DD861E	ADD	A,(IX+d)
DD2C	INC	LX	DD8C	ADC	A,HX
DD2D	DEC	LX	DD8D	ADC	A,LX
DD2E48	LD	LX,n	DD8E1E	ADC	A,(IX+d)
DD341E	INC	(IX+d)	DD94	SUB	HX
DD351E	DEC	(IX+d)	DD95	SUB	LX
DD361E48	LD	(IX+d),n	DD961E	SUB	(IX+d)
DD39	ADD	IX,SP	DD9C	SBC	A,HX
DD44	LD	B,HX	DD9D	SBC	A,LX
DD45	LD	B,LX	DD9E1E	SBC	A,(IX+d)
DD461E	LD	B,(IX+d)	DDA4	AND	HX
DD4C	LD	C,HX	DDA5	AND	LX
DD4D	LD	C,LX	DDA61E	AND	(IX+d)
DD4E1E	LD	C,(IX+d)	DDAC	XOR	HX
DD54	LD	D,HX	DDAD	XOR	LX
DD55	LD	D,LX	DDAE1E	XOR	(IX+d)
DD561E	LD	D,(IX+d)	DDB4	OR	HX
DD5C	LD	E,HX	DDB5	OR	LX
DD5D	LD	E,LX	DDB61E	OR	(IX+d)
DD5E1E	LD	E,(IX+d)	DDBC	CP	HX
DD60	LD	HX,B	DDBD	CP	LX
DD61	LD	HX,C	DDBE1E	CP	(IX+d)
DD62	LD	HX,D	DDCB1E00	RLC	B,(IX+d)
DD63	LD	HX,E	DDCB1E01	RLC	C,(IX+d)
DD64	LD	HX,HX	DDCB1E02	RLC	D,(IX+d)
DD65	LD	HX,LX	DDCB1E03	RLC	E,(IX+d)
DD661E	LD	H,(IX+d)	DDCB1E04	RLC	H,(IX+d)
DD67	LD	HX,A	DDCB1E05	RLC	L,(IX+d)
DD68	LD	LX,B	DDCB1E06	RLC	(IX+d)
DD69	LD	LX,C	DDCB1E07	RLC	A,(IX+d)
DD6A	LD	LX,D	DDCB1E08	RRC	B,(IX+d)
DD6B	LD	LX,E	DDCB1E09	RRC	C,(IX+d)
DD6C	LD	LX,HX	DDCB1E0A	RRC	D,(IX+d)
DD6D	LD	LX,LX	DDCB1E0B	RRC	E,(IX+d)
DD6E1E	LD	L,(IX+d)	DDCB1E0C	RRC	H,(IX+d)
DD6F	LD	LX,A	DDCB1E0D	RRC	L,(IX+d)
DD701E	LD	(IX+d),B	DDCB1E0E	RRC	(IX+d)
DD711E	LD	(IX+d),C	DDCB1E0F	RRC	A,(IX+d)
DD721E	LD	(IX+d),D	DDCB1E10	RL	B,(IX+d)
DD731E	LD	(IX+d),E	DDCB1E11	RL	C,(IX+d)
DD741E	LD	(IX+d),H	DDCB1E12	RL	D,(IX+d)
DD751E	LD	(IX+d),L	DDCB1E13	RL	E,(IX+d)

Z80 Instructions by Object Code

DDCB1E14	RL	H, (IX+d)	DDCB1E66	BIT	4, (IX+d)
DDCB1E15	RL	L, (IX+d)	DDCB1E6E	BIT	5, (IX+d)
DDCB1E16	RL	(IX+d)	DDCB1E76	BIT	6, (IX+d)
DDCB1E17	RL	A, (IX+d)	DDCB1E7E	BIT	7, (IX+d)
DDCB1E18	RR	B, (IX+d)	DDCB1E80	RES	0, B, (IX+d)
DDCB1E19	RR	C, (IX+d)	DDCB1E81	RES	0, C, (IX+d)
DDCB1E1A	RR	D, (IX+d)	DDCB1E82	RES	0, D, (IX+d)
DDCB1E1B	RR	E, (IX+d)	DDCB1E83	RES	0, E, (IX+d)
DDCB1E1C	RR	H, (IX+d)	DDCB1E84	RES	0, H, (IX+d)
DDCB1E1D	RR	L, (IX+d)	DDCB1E85	RES	0, L, (IX+d)
DDCB1E1E	RR	(IX+d)	DDCB1E86	RES	0, (IX+d)
DDCB1E1F	RR	A, (IX+d)	DDCB1E87	RES	0, A, (IX+d)
DDCB1E20	SLA	B, (IX+d)	DDCB1E88	RES	1, B, (IX+d)
DDCB1E21	SLA	C, (IX+d)	DDCB1E89	RES	1, C, (IX+d)
DDCB1E22	SLA	D, (IX+d)	DDCB1E8A	RES	1, D, (IX+d)
DDCB1E23	SLA	E, (IX+d)	DDCB1E8B	RES	1, E, (IX+d)
DDCB1E24	SLA	H, (IX+d)	DDCB1E8C	RES	1, H, (IX+d)
DDCB1E25	SLA	L, (IX+d)	DDCB1E8D	RES	1, L, (IX+d)
DDCB1E26	SLA	(IX+d)	DDCB1E8E	RES	1, (IX+d)
DDCB1E27	SLA	A, (IX+d)	DDCB1E8F	RES	1, A, (IX+d)
DDCB1E28	SRA	B, (IX+d)	DDCB1E90	RES	2, B, (IX+d)
DDCB1E29	SRA	C, (IX+d)	DDCB1E91	RES	2, C, (IX+d)
DDCB1E2A	SRA	D, (IX+d)	DDCB1E92	RES	2, D, (IX+d)
DDCB1E2B	SRA	E, (IX+d)	DDCB1E93	RES	2, E, (IX+d)
DDCB1E2C	SRA	H, (IX+d)	DDCB1E94	RES	2, H, (IX+d)
DDCB1E2D	SRA	L, (IX+d)	DDCB1E95	RES	2, L, (IX+d)
DDCB1E2E	SRA	(IX+d)	DDCB1E96	RES	2, (IX+d)
DDCB1E2F	SRA	A, (IX+d)	DDCB1E97	RES	2, A, (IX+d)
DDCB1E30	SLL	B, (IX+d)	DDCB1E98	RES	3, B, (IX+d)
DDCB1E31	SLL	C, (IX+d)	DDCB1E99	RES	3, C, (IX+d)
DDCB1E32	SLL	D, (IX+d)	DDCB1E9A	RES	3, D, (IX+d)
DDCB1E33	SLL	E, (IX+d)	DDCB1E9B	RES	3, E, (IX+d)
DDCB1E34	SLL	H, (IX+d)	DDCB1E9C	RES	3, H, (IX+d)
DDCB1E35	SLL	L, (IX+d)	DDCB1E9D	RES	3, L, (IX+d)
DDCB1E36	SLL	(IX+d)	DDCB1E9E	RES	3, (IX+d)
DDCB1E37	SLL	A, (IX+d)	DDCB1E9F	RES	3, A, (IX+d)
DDCB1E38	SRL	B, (IX+d)	DDCB1EA0	RES	4, B, (IX+d)
DDCB1E39	SRL	C, (IX+d)	DDCB1EA1	RES	4, C, (IX+d)
DDCB1E3A	SRL	D, (IX+d)	DDCB1EA2	RES	4, D, (IX+d)
DDCB1E3B	SRL	E, (IX+d)	DDCB1EA3	RES	4, E, (IX+d)
DDCB1E3C	SRL	H, (IX+d)	DDCB1EA4	RES	4, H, (IX+d)
DDCB1E3D	SRL	L, (IX+d)	DDCB1EA5	RES	4, L, (IX+d)
DDCB1E3E	SRL	(IX+d)	DDCB1EA6	RES	4, (IX+d)
DDCB1E3F	SRL	A, (IX+d)	DDCB1EA7	RES	4, A, (IX+d)
DDCB1E46	BIT	0, (IX+d)	DDCB1EA8	RES	5, B, (IX+d)
DDCB1E4E	BIT	1, (IX+d)	DDCB1EA9	RES	5, C, (IX+d)
DDCB1E56	BIT	2, (IX+d)	DDCB1EAA	RES	5, D, (IX+d)
DDCB1E5E	BIT	3, (IX+d)	DDCB1EAB	RES	5, E, (IX+d)

Z80 Instructions by Object Code

DDCB1EAC	RES	5, H, (IX+d)	DDCB1EDC	SET	3, H, (IX+d)
DDCB1EAD	RES	5, L, (IX+d)	DDCB1EDD	SET	3, L, (IX+d)
DDCB1EAE	RES	5, (IX+d)	DDCB1EDE	SET	3, (IX+d)
DDCB1EAF	RES	5, A, (IX+d)	DDCB1EDF	SET	3, A, (IX+d)
DDCB1EB0	RES	6, B, (IX+d)	DDCB1EE0	SET	4, B, (IX+d)
DDCB1EB1	RES	6, C, (IX+d)	DDCB1EE1	SET	4, C, (IX+d)
DDCB1EB2	RES	6, D, (IX+d)	DDCB1EE2	SET	4, D, (IX+d)
DDCB1EB3	RES	6, E, (IX+d)	DDCB1EE3	SET	4, E, (IX+d)
DDCB1EB4	RES	6, H, (IX+d)	DDCB1EE4	SET	4, H, (IX+d)
DDCB1EB5	RES	6, L, (IX+d)	DDCB1EE5	SET	4, L, (IX+d)
DDCB1EB6	RES	6, (IX+d)	DDCB1EE6	SET	4, (IX+d)
DDCB1EB7	RES	6, A, (IX+d)	DDCB1EE7	SET	4, A, (IX+d)
DDCB1EB8	RES	7, B, (IX+d)	DDCB1EE8	SET	5, B, (IX+d)
DDCB1EB9	RES	7, C, (IX+d)	DDCB1EE9	SET	5, C, (IX+d)
DDCB1EBA	RES	7, D, (IX+d)	DDCB1EEA	SET	5, D, (IX+d)
DDCB1EBB	RES	7, E, (IX+d)	DDCB1EEB	SET	5, E, (IX+d)
DDCB1EBC	RES	7, H, (IX+d)	DDCB1EEC	SET	5, H, (IX+d)
DDCB1EBD	RES	7, L, (IX+d)	DDCB1EED	SET	5, L, (IX+d)
DDCB1EBE	RES	7, (IX+d)	DDCB1EEE	SET	5, (IX+d)
DDCB1EBF	RES	7, A, (IX+d)	DDCB1EEF	SET	5, A, (IX+d)
DDCB1EC0	SET	0, B, (IX+d)	DDCB1EF0	SET	6, B, (IX+d)
DDCB1EC1	SET	0, C, (IX+d)	DDCB1EF1	SET	6, C, (IX+d)
DDCB1EC2	SET	0, D, (IX+d)	DDCB1EF2	SET	6, D, (IX+d)
DDCB1EC3	SET	0, E, (IX+d)	DDCB1EF3	SET	6, E, (IX+d)
DDCB1EC4	SET	0, H, (IX+d)	DDCB1EF4	SET	6, H, (IX+d)
DDCB1EC5	SET	0, L, (IX+d)	DDCB1EF5	SET	6, L, (IX+d)
DDCB1EC6	SET	0, (IX+d)	DDCB1EF6	SET	6, (IX+d)
DDCB1EC7	SET	0, A, (IX+d)	DDCB1EF7	SET	6, A, (IX+d)
DDCB1EC8	SET	1, B, (IX+d)	DDCB1EF8	SET	7, B, (IX+d)
DDCB1EC9	SET	1, C, (IX+d)	DDCB1EF9	SET	7, C, (IX+d)
DDCB1ECA	SET	1, D, (IX+d)	DDCB1EFA	SET	7, D, (IX+d)
DDCB1ECB	SET	1, E, (IX+d)	DDCB1EFB	SET	7, E, (IX+d)
DDCB1ECC	SET	1, H, (IX+d)	DDCB1EFC	SET	7, H, (IX+d)
DDCB1ECD	SET	1, L, (IX+d)	DDCB1EFD	SET	7, L, (IX+d)
DDCB1ECE	SET	1, (IX+d)	DDCB1EFE	SET	7, (IX+d)
DDCB1ECF	SET	1, A, (IX+d)	DDCB1EFF	SET	7, A, (IX+d)
DDCB1ED0	SET	2, B, (IX+d)	DDE1	POP	IX
DDCB1ED1	SET	2, C, (IX+d)	DDE3	EX	(SP), IX
DDCB1ED2	SET	2, D, (IX+d)	DDE5	PUSH	IX
DDCB1ED3	SET	2, E, (IX+d)	DDE9	JP	(IX)
DDCB1ED4	SET	2, H, (IX+d)	DDF9	LD	SP, IX
DDCB1ED5	SET	2, L, (IX+d)	DE48	SBC	A, n
DDCB1ED6	SET	2, (IX+d)	DF	RST	18H
DDCB1ED7	SET	2, A, (IX+d)	E0	RET	PO
DDCB1ED8	SET	3, B, (IX+d)	E1	POP	HL
DDCB1ED9	SET	3, C, (IX+d)	E2A00B	JP	PO, mn
DDCB1EDA	SET	3, D, (IX+d)	E3	EX	(SP), HL
DDCB1EDB	SET	3, E, (IX+d)	E4A00B	CALL	PO, mn

Z80 Instructions by Object Code

E5	PUSH	HL	ED7A	ADC	HL, SP
E648	AND	n	ED7BA00B	LD	SP, (mn)
E7	RST	20H	EDA0	LDI	
E8	RET	PE	EDA1	CPI	
E9	JP	(HL)	EDA2	INI	
EAA00B	JP	PE, mn	EDA3	OUTI	
EB	EX	DE, HL	EDA8	LDD	
ECA00B	CALL	PE, mn	EDA9	CPD	
ED40	IN	B, (C)	EDAA	IND	
ED41	OUT	(C), B	EDAB	OUTD	
ED42	SBC	HL, BC	EDB0	LDIR	
ED43A00B	LD	(mn), BC	EDB1	CPIR	
ED44	NEG		EDB2	INIR	
ED45	RETN		EDB3	OTIR	
ED46	IM	0	EDB8	LDDR	
ED47	LD	I, A	EDB9	CPDR	
ED48	IN	C, (C)	EDBA	INDR	
ED49	OUT	(C), C	EDBB	OTDR	
ED4A	ADC	HL, BC	EE48	XOR	n
ED4BA00B	LD	BC, (mn)	EF	RST	28H
ED4D	RETI		F0	RET	P
ED4F	LD	R, A	F1	POP	AF
ED50	IN	D, (C)	F2A00B	JP	P, mn
ED51	OUT	(C), D	F3	DI	
ED52	SBC	HL, DE	F4A00B	CALL	P, mn
ED53A00B	LD	(mn), DE	F5	PUSH	AF
ED56	IM	1	F648	OR	n
ED57	LD	A, I	F7	RST	30H
ED58	IN	E, (C)	F8	RET	M
ED59	OUT	(C), E	F9	LD	SP, HL
ED5A	ADC	HL, DE	FAA00B	JP	M, mn
ED5BA00B	LD	DE, (mn)	FB	EI	
ED5E	IM	2	FCA00B	CALL	M, mn
ED5F	LD	A, R	FD09	ADD	IY, BC
ED60	IN	H, (C)	FD19	ADD	IY, DE
ED61	OUT	(C), H	FD21A00B	LD	IY, mn
ED62	SBC	HL, HL	FD22A00B	LD	(mn), IY
ED67	RRD		FD23	INC	IY
ED68	IN	L, (C)	FD24	INC	HY
ED69	OUT	(C), L	FD25	DEC	HY
ED6A	ADC	HL, HL	FD2648	LD	HY, n
ED6F	RLD		FD29	ADD	IY, IY
ED70	IN	(HL), (C)	FD2AA00B	LD	IY, (mn)
ED71	OUT	(C), (HL)	FD2B	DEC	IY
ED72	SBC	HL, SP	FD2C	INC	LY
ED73A00B	LD	(mn), SP	FD2D	DEC	LY
ED78	IN	A, (C)	FD2E48	LD	LY, n
ED79	OUT	(C), A	FD341E	INC	(IY+d)

Z80 Instructions by Object Code

FD351E	DEC	(IY+d)	FD95	SUB	LY
FD361E48	LD	(IY+d),n	FD961E	SUB	(IY+d)
FD39	ADD	IY,SP	FD9C	SBC	A,HY
FD44	LD	B,HY	FD9D	SBC	A,LY
FD45	LD	B,LY	FD9E1E	SBC	A,(IY+d)
FD461E	LD	B,(IY+d)	FDA4	AND	HY
FD4C	LD	C,HY	FDA5	AND	LY
FD4D	LD	C,LY	FDA61E	AND	(IY+d)
FD4E1E	LD	C,(IY+d)	FDAC	XOR	HY
FD54	LD	D,HY	FDAD	XOR	LY
FD55	LD	D,LY	FDAE1E	XOR	(IY+d)
FD561E	LD	D,(IY+d)	FDB4	OR	HY
FD5C	LD	E,HY	FDB5	OR	LY
FD5D	LD	E,LY	FDB61E	OR	(IY+d)
FD5E1E	LD	E,(IY+d)	FDBC	CP	HY
FD60	LD	HY,B	FDBD	CP	LY
FD61	LD	HY,C	FDBE1E	CP	(IY+d)
FD62	LD	HY,D	FDCB1E00	RLC	B,(IY+d)
FD63	LD	HY,E	FDCB1E01	RLC	C,(IY+d)
FD64	LD	HY,HY	FDCB1E02	RLC	D,(IY+d)
FD65	LD	HY,LY	FDCB1E03	RLC	E,(IY+d)
FD661E	LD	H,(IY+d)	FDCB1E04	RLC	H,(IY+d)
FD67	LD	HY,A	FDCB1E05	RLC	L,(IY+d)
FD68	LD	LY,B	FDCB1E06	RLC	(IY+d)
FD69	LD	LY,C	FDCB1E07	RLC	A,(IY+d)
FD6A	LD	LY,D	FDCB1E08	RRC	B,(IY+d)
FD6B	LD	LY,E	FDCB1E09	RRC	C,(IY+d)
FD6C	LD	LY,HY	FDCB1E0A	RRC	D,(IY+d)
FD6D	LD	LY,LY	FDCB1E0B	RRC	E,(IY+d)
FD6E1E	LD	L,(IY+d)	FDCB1E0C	RRC	H,(IY+d)
FD6F	LD	LY,A	FDCB1E0D	RRC	L,(IY+d)
FD701E	LD	(IY+d),B	FDCB1E0E	RRC	(IY+d)
FD711E	LD	(IY+d),C	FDCB1E0F	RRC	A,(IY+d)
FD721E	LD	(IY+d),D	FDCB1E10	RL	B,(IY+d)
FD731E	LD	(IY+d),E	FDCB1E11	RL	C,(IY+d)
FD741E	LD	(IY+d),H	FDCB1E12	RL	D,(IY+d)
FD751E	LD	(IY+d),L	FDCB1E13	RL	E,(IY+d)
FD771E	LD	(IY+d),A	FDCB1E14	RL	H,(IY+d)
FD7C	LD	A,HY	FDCB1E15	RL	L,(IY+d)
FD7D	LD	A,LY	FDCB1E16	RL	(IY+d)
FD7E1E	LD	A,(IY+d)	FDCB1E17	RL	A,(IY+d)
FD84	ADD	A,HY	FDCB1E18	RR	B,(IY+d)
FD85	ADD	A,LY	FDCB1E19	RR	C,(IY+d)
FD861E	ADD	A,(IY+d)	FDCB1E1A	RR	D,(IY+d)
FD8C	ADC	A,HY	FDCB1E1B	RR	E,(IY+d)
FD8D	ADC	A,LY	FDCB1E1C	RR	H,(IY+d)
FD8E1E	ADC	A,(IY+d)	FDCB1E1D	RR	L,(IY+d)
FD94	SUB	HY	FDCB1E1E	RR	(IY+d)

Z80 Instructions by Object Code

FDCB1E1F	RR	A, (IY+d)	FDCB1E87	RES	0, A, (IY+d)
FDCB1E20	SLA	B, (IY+d)	FDCB1E88	RES	1, B, (IY+d)
FDCB1E21	SLA	C, (IY+d)	FDCB1E89	RES	1, C, (IY+d)
FDCB1E22	SLA	D, (IY+d)	FDCB1E8A	RES	1, D, (IY+d)
FDCB1E23	SLA	E, (IY+d)	FDCB1E8B	RES	1, E, (IY+d)
FDCB1E24	SLA	H, (IY+d)	FDCB1E8C	RES	1, H, (IY+d)
FDCB1E25	SLA	L, (IY+d)	FDCB1E8D	RES	1, L, (IY+d)
FDCB1E26	SLA	(IY+d)	FDCB1E8E	RES	1, (IY+d)
FDCB1E27	SLA	A, (IY+d)	FDCB1E8F	RES	1, A, (IY+d)
FDCB1E28	SRA	B, (IY+d)	FDCB1E90	RES	2, B, (IY+d)
FDCB1E29	SRA	C, (IY+d)	FDCB1E91	RES	2, C, (IY+d)
FDCB1E2A	SRA	D, (IY+d)	FDCB1E92	RES	2, D, (IY+d)
FDCB1E2B	SRA	E, (IY+d)	FDCB1E93	RES	2, E, (IY+d)
FDCB1E2C	SRA	H, (IY+d)	FDCB1E94	RES	2, H, (IY+d)
FDCB1E2D	SRA	L, (IY+d)	FDCB1E95	RES	2, L, (IY+d)
FDCB1E2E	SRA	(IY+d)	FDCB1E96	RES	2, (IY+d)
FDCB1E2F	SRA	A, (IY+d)	FDCB1E97	RES	2, A, (IY+d)
FDCB1E30	SLL	B, (IY+d)	FDCB1E98	RES	3, B, (IY+d)
FDCB1E31	SLL	C, (IY+d)	FDCB1E99	RES	3, C, (IY+d)
FDCB1E32	SLL	D, (IY+d)	FDCB1E9A	RES	3, D, (IY+d)
FDCB1E33	SLL	E, (IY+d)	FDCB1E9B	RES	3, E, (IY+d)
FDCB1E34	SLL	H, (IY+d)	FDCB1E9C	RES	3, H, (IY+d)
FDCB1E35	SLL	L, (IY+d)	FDCB1E9D	RES	3, L, (IY+d)
FDCB1E36	SLL	(IY+d)	FDCB1E9E	RES	3, (IY+d)
FDCB1E37	SLL	A, (IY+d)	FDCB1E9F	RES	3, A, (IY+d)
FDCB1E38	SRL	B, (IY+d)	FDCB1EA0	RES	4, B, (IY+d)
FDCB1E39	SRL	C, (IY+d)	FDCB1EA1	RES	4, C, (IY+d)
FDCB1E3A	SRL	D, (IY+d)	FDCB1EA2	RES	4, D, (IY+d)
FDCB1E3B	SRL	E, (IY+d)	FDCB1EA3	RES	4, E, (IY+d)
FDCB1E3C	SRL	H, (IY+d)	FDCB1EA4	RES	4, H, (IY+d)
FDCB1E3D	SRL	L, (IY+d)	FDCB1EA5	RES	4, L, (IY+d)
FDCB1E3E	SRL	(IY+d)	FDCB1EA6	RES	4, (IY+d)
FDCB1E3F	SRL	A, (IY+d)	FDCB1EA7	RES	4, A, (IY+d)
FDCB1E46	BIT	0, (IY+d)	FDCB1EA8	RES	5, B, (IY+d)
FDCB1E4E	BIT	1, (IY+d)	FDCB1EA9	RES	5, C, (IY+d)
FDCB1E56	BIT	2, (IY+d)	FDCB1EAA	RES	5, D, (IY+d)
FDCB1E5E	BIT	3, (IY+d)	FDCB1EAB	RES	5, E, (IY+d)
FDCB1E66	BIT	4, (IY+d)	FDCB1EAC	RES	5, H, (IY+d)
FDCB1E6E	BIT	5, (IY+d)	FDCB1EAD	RES	5, L, (IY+d)
FDCB1E76	BIT	6, (IY+d)	FDCB1EAE	RES	5, (IY+d)
FDCB1E7E	BIT	7, (IY+d)	FDCB1EAF	RES	5, A, (IY+d)
FDCB1E80	RES	0, B, (IY+d)	FDCB1EB0	RES	6, B, (IY+d)
FDCB1E81	RES	0, C, (IY+d)	FDCB1EB1	RES	6, C, (IY+d)
FDCB1E82	RES	0, D, (IY+d)	FDCB1EB2	RES	6, D, (IY+d)
FDCB1E83	RES	0, E, (IY+d)	FDCB1EB3	RES	6, E, (IY+d)
FDCB1E84	RES	0, H, (IY+d)	FDCB1EB4	RES	6, H, (IY+d)
FDCB1E85	RES	0, L, (IY+d)	FDCB1EB5	RES	6, L, (IY+d)
FDCB1E86	RES	0, (IY+d)	FDCB1EB6	RES	6, (IY+d)

Z80 Instructions by Object Code

FDCB1EB7	RES	6,A, (IY+d)	FDCB1EE7	SET	4,A, (IY+d)
FDCB1EB8	RES	7,B, (IY+d)	FDCB1EE8	SET	5,B, (IY+d)
FDCB1EB9	RES	7,C, (IY+d)	FDCB1EE9	SET	5,C, (IY+d)
FDCB1EBA	RES	7,D, (IY+d)	FDCB1EEA	SET	5,D, (IY+d)
FDCB1EBB	RES	7,E, (IY+d)	FDCB1EEB	SET	5,E, (IY+d)
FDCB1EBC	RES	7,H, (IY+d)	FDCB1EEC	SET	5,H, (IY+d)
FDCB1EBD	RES	7,L, (IY+d)	FDCB1EED	SET	5,L, (IY+d)
FDCB1EBE	RES	7, (IY+d)	FDCB1EEE	SET	5, (IY+d)
FDCB1EBF	RES	7,A, (IY+d)	FDCB1EEF	SET	5,A, (IY+d)
FDCB1EC0	SET	0,B, (IY+d)	FDCB1EF0	SET	6,B, (IY+d)
FDCB1EC1	SET	0,C, (IY+d)	FDCB1EF1	SET	6,C, (IY+d)
FDCB1EC2	SET	0,D, (IY+d)	FDCB1EF2	SET	6,D, (IY+d)
FDCB1EC3	SET	0,E, (IY+d)	FDCB1EF3	SET	6,E, (IY+d)
FDCB1EC4	SET	0,H, (IY+d)	FDCB1EF4	SET	6,H, (IY+d)
FDCB1EC5	SET	0,L, (IY+d)	FDCB1EF5	SET	6,L, (IY+d)
FDCB1EC6	SET	0, (IY+d)	FDCB1EF6	SET	6, (IY+d)
FDCB1EC7	SET	0,A, (IY+d)	FDCB1EF7	SET	6,A, (IY+d)
FDCB1EC8	SET	1,B, (IY+d)	FDCB1EF8	SET	7,B, (IY+d)
FDCB1EC9	SET	1,C, (IY+d)	FDCB1EF9	SET	7,C, (IY+d)
FDCB1ECA	SET	1,D, (IY+d)	FDCB1EFA	SET	7,D, (IY+d)
FDCB1ECB	SET	1,E, (IY+d)	FDCB1EFB	SET	7,E, (IY+d)
FDCB1ECC	SET	1,H, (IY+d)	FDCB1EFC	SET	7,H, (IY+d)
FDCB1ECD	SET	1,L, (IY+d)	FDCB1EFD	SET	7,L, (IY+d)
FDCB1ECE	SET	1, (IY+d)	FDCB1EFE	SET	7, (IY+d)
FDCB1ECF	SET	1,A, (IY+d)	FDCB1EFF	SET	7,A, (IY+d)
FDCB1ED0	SET	2,B, (IY+d)	FDE1	POP	IY
FDCB1ED1	SET	2,C, (IY+d)	FDE3	EX	(SP), IY
FDCB1ED2	SET	2,D, (IY+d)	FDE5	PUSH	IY
FDCB1ED3	SET	2,E, (IY+d)	FDE9	JP	(IY)
FDCB1ED4	SET	2,H, (IY+d)	FDF9	LD	SP, IY
FDCB1ED5	SET	2,L, (IY+d)	FE48	CP	n
FDCB1ED6	SET	2, (IY+d)	FF	RST	38H
FDCB1ED7	SET	2,A, (IY+d)	mn	DEFS	12<-2#1
FDCB1ED8	SET	3,B, (IY+d)	d	EQU	3C00H/200H
FDCB1ED9	SET	3,C, (IY+d)	n	EQU	40H!8
FDCB1EDA	SET	3,D, (IY+d)	e	EQU	9%7+30H
FDCB1EDB	SET	3,E, (IY+d)			
FDCB1EDC	SET	3,H, (IY+d)			
FDCB1EDD	SET	3,L, (IY+d)			
FDCB1EDE	SET	3, (IY+d)			
FDCB1EDF	SET	3,A, (IY+d)			
FDCB1EE0	SET	4,B, (IY+d)			
FDCB1EE1	SET	4,C, (IY+d)			
FDCB1EE2	SET	4,D, (IY+d)			
FDCB1EE3	SET	4,E, (IY+d)			
FDCB1EE4	SET	4,H, (IY+d)			
FDCB1EE5	SET	4,L, (IY+d)			
FDCB1EE6	SET	4, (IY+d)			

Undocumented Instructions

Regardless what you may have heard about the undocumented instructions, all Z80[®]s are the same (except for the timing problem with the P/V flag that was corrected with the CMOS chip). I did not say that a Z80[®] is the same as an HD64180, it is not.

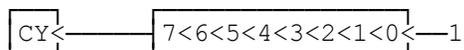
Zilog, the Z80[®] designer, indicates that a Z80[®] will execute 696 instructions. However, the number of unique instructions a Z80[®] will execute is 1138, and the number of different instructions a Z80[®] can encounter is 1786. The table below illustrates how I derived at the 1786 number:

	Zilog	Unique	Total
Opcodes 00 through FF; less CB, DD, ED, and FD	252	252	252
CB Opcodes	248	256	256
DD Opcodes; less DDCB	39	85	255
DDCB Opcodes	31	200	256
ED Opcodes	56	60	256
FD Opcodes; less FDCB	39	85	255
FDCB opcodes	31	200	256
TOTALS	696	1138	1786

Unique summary:

CB Family

The eight additional CB instructions not documented by Zilog are CB30 through CB37. The de facto opcode for CB30 through CB37 is SLL. The SLL instruction is described on page 5-72. The symbolic representation of this instruction is:



The SLL instruction doubles the value in the register and increments by one. The S, Z, P/V, and C flags are affected as expected (for complete detail, please see page 5-72).

The SLL instruction can be simulated with the following instruction consecution:

```
SLA r
INC r           this requires three bytes and affects the flags differently
```

or, if r is the accumulator:

```
SCF
RLA           this requires two bytes and only affects the C flag.
```

The HD64180 does not recognize the SLL instruction. If an HD64180 encounters an SLL instruction, the HD64180 performs a RST 0 on the CB part and executes the code starting with the 30 through 37 as [part of] the next instruction (provided the PC returns from the RST 0).

Undocumented Instructions

DD Family

The 46 additional DD instructions not documented by Zilog are the half-index instructions - the operands with HX and LX. [NOTE: for the Z280™, Zilog refers to the upper-half of the IX register as IXH, and the lower half of the IX register as IXL. I prefer HX over IXH and LX over IXL to keep all register symbols at one or two characters.] The HD64180 does not recognize the half-index instructions.

DDCB Family

The 169 additional DDCB instructions not documented by Zilog are the rotate/shift-load (57), reset-load (56), and set-load (56) instructions. The rotate/shift-load group has one more undocumented instruction than the other two groups because Zilog does not document the SLL (IX+d) mnemonic. The HD64180 does not recognize the 169 additional DDCB instructions.

ED Family

The four additional ED instructions are:

ED70	IN (HL), (C)	nothing is transferred; however, flags are updated
ED71	OUT (C), (HL)	zero is output to port
ED63nm	LD (mn), HL	performs (less efficiently) the same as 22nm
ED6Bnm	LD HL, (mn)	performs (less efficiently) the same as 2Anm

The HD64180 recognizes ED70, ED63nm, and ED6Bnm [the HD64180 processes ED71 as RST 0 followed by LD (HL), C].

FD Family

The 46 additional FD instructions not documented by Zilog are the half-index instructions - the operands with HY and LY. [NOTE: for the Z280™, Zilog refers to the upper-half of the IY register as IYH, and the lower half of the IY register as IYL. I prefer HY over IYH and LY over IYL to keep all register symbols at one or two characters.] The HD64180 does not recognize the half-index instructions.

FDCB Family

The 169 additional FDCB instructions not documented by Zilog are the rotate/shift-load (57), reset-load (56), and set-load (56) instructions. The rotate/shift-load group has one more undocumented instruction than the other two groups because Zilog does not document the SLL (IY+d) mnemonic. The HD64180 does not recognize the 169 additional FDCB instructions.

Of the 1,138 unique instructions, ZEUS can produce 1,136. For the LD (mn), HL instruction, Zeus produces 22nm and does not produce ED63nm. For the LD HL, (mn) instruction, ZEUS produces 2Anm and does not produce ED6Bnm.

How a Z80[®] processes DD[[xx[xx]]xx]

DD 00			NOP : NOP	DD 20	n		NOP : JR NZ,n	DD 40			NOP : LD B,B	DD 60			LD HX,B
DD 01	n	m	NOP : LD BC,mn	DD 21	n	m	LD IX,word	DD 41			NOP : LD B,C	DD 61			LD HX,C
DD 02			NOP : LD (BC),A	DD 22	n	m	LD (word),IX	DD 42			NOP : LD B,D	DD 62			LD HX,D
DD 03			NOP : INC BC	DD 23			INC IX	DD 43			NOP : LD B,E	DD 63			LD HX,E
DD 04			NOP : INC B	DD 24			INC HX	DD 44			LD B,HX	DD 64			LD HX,HX
DD 05			NOP : DEC B	DD 25			DEC HX	DD 45			LD B,LX	DD 65			LD HX,LX
DD 06	n		NOP : LD B,n	DD 26	n		LD HX,byte	DD 46	d		LD B,(IX+d)	DD 66	d		LD H,(IX+d)
DD 07			NOP : RLCA	DD 27			NOP : DAA	DD 47			NOP : LD B,A	DD 67			LD HX,A
DD 08			NOP : EX AF,AF'	DD 28	n		NOP : JR Z,n	DD 48			NOP : LD C,B	DD 68			LD LX,B
DD 09			ADD IX,BC	DD 29			ADD IX,IX	DD 49			NOP : LD C,C	DD 69			LD LX,C
DD 0A			NOP : LD A,(BC)	DD 2A	n	m	LD IX,(word)	DD 4A			NOP : LD C,D	DD 6A			LD LX,D
DD 0B			NOP : DEC BC	DD 2B			DEC IX	DD 4B			NOP : LD C,E	DD 6B			LD LX,E
DD 0C			NOP : INC C	DD 2C			INC LX	DD 4C			LD C,HX	DD 6C			LD LX,HX
DD 0D			NOP : DEC C	DD 2D			DEC LX	DD 4D			LD C,LX	DD 6D			LD LX,LX
DD 0E	n		NOP : LD C,n	DD 2E	n		LD LX,byte	DD 4E	d		LD C,(IX+d)	DD 6E	d		LD L,(IX+d)
DD 0F			NOP : RRCA	DD 2F			NOP : CPL	DD 4F			NOP : LD C,A	DD 6F			LD LX,A
DD 10	n		NOP : DJNZ n	DD 30	n		NOP : JR NC,n	DD 50			NOP : LD D,B	DD 70	d		LD (IX+d),B
DD 11	n	m	NOP : LD DE,mn	DD 31	n	m	NOP : LD SP,mn	DD 51			NOP : LD D,C	DD 71	d		LD (IX+d),C
DD 12			NOP : LD (DE),A	DD 32	n	m	NOP : LD (mn),A	DD 52			NOP : LD D,D	DD 72	d		LD (IX+d),D
DD 13			NOP : INC DE	DD 33			NOP : INC SP	DD 53			NOP : LD D,E	DD 73	d		LD (IX+d),E
DD 14			NOP : INC D	DD 34	d		INC (IX+d)	DD 54			LD D,HX	DD 74	d		LD (IX+d),H
DD 15			NOP : DEC D	DD 35	d		DEC (IX+d)	DD 55			LD D,LX	DD 75	d		LD (IX+d),L
DD 16	n		NOP : LD D,n	DD 36	d	n	LD (IX+d),byte	DD 56	d		LD D,(IX+d)	DD 76			NOP : HALT
DD 17			NOP : RLA	DD 37			NOP : SCF	DD 57			NOP : LD D,A	DD 77	d		LD (IX+d),A
DD 18	n		NOP : JR n	DD 38	n		NOP : JR C,n	DD 58			NOP : LD E,B	DD 78			NOP : LD A,B
DD 19			ADD IX,DE	DD 39			ADD IX,SP	DD 59			NOP : LD E,C	DD 79			NOP : LD A,C
DD 1A			NOP : LD A,(DE)	DD 3A	n	m	NOP : LD A,(mn)	DD 5A			NOP : LD E,D	DD 7A			NOP : LD A,D
DD 1B			NOP : DEC DE	DD 3B			NOP : DEC SP	DD 5B			NOP : LD E,E	DD 7B			NOP : LD A,E
DD 1C			NOP : INC E	DD 3C			NOP : INC A	DD 5C			LD E,HX	DD 7C			LD A,HX
DD 1D			NOP : DEC E	DD 3D			NOP : DEC A	DD 5D			LD E,LX	DD 7D			LD A,LX
DD 1E	n		NOP : LD E,n	DD 3E	n		NOP : LD A,n	DD 5E	d		LD E,(IX+d)	DD 7E	d		LD A,(IX+d)
DD 1F			NOP : RRA	DD 3F			NOP : CCF	DD 5F			NOP : LD E,A	DD 7F			NOP : LD A,A

How a Z80[®] processes DD[[xx[xx]]xx]

DD 80		NOP : ADD A,B	DD A0		NOP : AND B	DD C0		NOP : RET NZ	DD E0		NOP : RET PO
DD 81		NOP : ADD A,C	DD A1		NOP : AND C	DD C1		NOP : POP BC	DD E1		POP IX
DD 82		NOP : ADD A,D	DD A2		NOP : AND D	DD C2	n m	NOP : JP NZ,mn	DD E2	n m	NOP : JP PO,mn
DD 83		NOP : ADD A,E	DD A3		NOP : AND E	DD C3	n m	NOP : JP mn	DD E3		EX (SP),IX
DD 84		ADD A,HX	DD A4		AND HX	DD C4	n m	NOP : CALL NZ,mn	DD E4	n m	NOP : CALL PO,mn
DD 85		ADD A,LX	DD A5		AND LX	DD C5		NOP : PUSH BC	DD E5		PUSH IX
DD 86	d	ADD A,(IX+d)	DD A6	d	AND (IX+d)	DD C6	n	NOP : ADD A,n	DD E6	n	NOP : AND n
DD 87		NOP : ADD A,A	DD A7		NOP : AND A	DD C7		NOP : RST 00H	DD E7		NOP : RST 20H
DD 88		NOP : ADC A,B	DD A8		NOP : XOR B	DD C8		NOP : RET Z	DD E8		NOP : RET PE
DD 89		NOP : ADC A,C	DD A9		NOP : XOR C	DD C9		NOP : RET	DD E9		JP (IX)
DD 8A		NOP : ADC A,D	DD AA		NOP : XOR D	DD CA	n m	NOP : JP Z,mn	DD EA	n m	NOP : JP PE,mn
DD 8B		NOP : ADC A,E	DD AB		NOP : XOR E	DD CB		see Opcode DDCB	DD EB		NOP : EX DE,HL
DD 8C		ADC A,HX	DD AC		XOR HX	DD CC	n m	NOP : CALL Z,mn	DD EC	n m	NOP : CALL PE,mn
DD 8D		ADC A,LX	DD AD		XOR LX	DD CD	n m	NOP : CALL mn	DD ED		NOP : EDxx
DD 8E	d	ADC A,(IX+d)	DD AE	d	XOR (IX+d)	DD CE	n	NOP : ADC A,n	DD EE	n	NOP : XOR n
DD 8F		NOP : ADC A,A	DD AF		NOP : XOR A	DD CF		NOP : RST 08H	DD EF		NOP : RST 28H
DD 90		NOP : SUB B	DD B0		NOP : OR B	DD D0		NOP : RET NC	DD F0		NOP : RET P
DD 91		NOP : SUB C	DD B1		NOP : OR C	DD D1		NOP : POP DE	DD F1		NOP : POP AF
DD 92		NOP : SUB D	DD B2		NOP : OR D	DD D2	n m	NOP : JP NC,mn	DD F2	n m	NOP : JP P,mn
DD 93		NOP : SUB E	DD B3		NOP : OR E	DD D3	n	NOP : OUT (m),A	DD F3		NOP : DI
DD 94		SUB HX	DD B4		OR HX	DD D4	n m	NOP : CALL NC,mn	DD F4	n m	NOP : CALL P,mn
DD 95		SUB LX	DD B5		OR LX	DD D5		NOP : PUSH DE	DD F5		NOP : PUSH AF
DD 96	d	SUB (IX+d)	DD B6	d	OR (IX+d)	DD D6	n	NOP : SUB n	DD F6	n	NOP : OR n
DD 97		NOP : SUB A	DD B7		NOP : OR A	DD D7		NOP : RST 10H	DD F7		NOP : RST 30H
DD 98		NOP : SBC A,B	DD B8		NOP : CP B	DD D8		NOP : RET C	DD F8		NOP : RET M
DD 99		NOP : SBC A,C	DD B9		NOP : CP C	DD D9		NOP : EXX	DD F9		LD SP,IX
DD 9A		NOP : SBC A,D	DD BA		NOP : CP D	DD DA	n m	NOP : JP C,mn	DD FA	n m	NOP : JP M,mn
DD 9B		NOP : SBC A,E	DD BB		NOP : CP E	DD DB	n	NOP : IN A,(m)	DD FB		NOP : EI
DD 9C		SBC A,HX	DD BC		CP HX	DD DC	n m	NOP : CALL C,mn	DD FC	n m	NOP : CALL M,mn
DD 9D		SBC A,LX	DD BD		CP LX	DD DD		NOP : DDxx[xx[xx]]	DD FD		NOP : FDxx[xx[xx]]
DD 9E	d	SBC A,(IX+d)	DD BE	d	CP (IX+d)	DD DE	n	NOP : SBC A,n	DD FE	n	NOP : CP n
DD 9F		NOP : SBC A,A	DD BF		NOP : CP A	DD DF		NOP : RST 18H	DD FF		NOP : RST 38H

How a Z80[®] processes DDCBxxxx

DD	CB	d	00	RLC B,(IX+d)		DD	CB	d	20	SLA B,(IX+d)		DD	CB	d	40	BIT 0,(IX+d)		DD	CB	d	60	BIT 4,(IX+d)
DD	CB	d	01	RLC C,(IX+d)		DD	CB	d	21	SLA C,(IX+d)		DD	CB	d	41	BIT 0,(IX+d)		DD	CB	d	61	BIT 4,(IX+d)
DD	CB	d	02	RLC D,(IX+d)		DD	CB	d	22	SLA D,(IX+d)		DD	CB	d	42	BIT 0,(IX+d)		DD	CB	d	62	BIT 4,(IX+d)
DD	CB	d	03	RLC E,(IX+d)		DD	CB	d	23	SLA E,(IX+d)		DD	CB	d	43	BIT 0,(IX+d)		DD	CB	d	63	BIT 4,(IX+d)
DD	CB	d	04	RLC H,(IX+d)		DD	CB	d	24	SLA H,(IX+d)		DD	CB	d	44	BIT 0,(IX+d)		DD	CB	d	64	BIT 4,(IX+d)
DD	CB	d	05	RLC L,(IX+d)		DD	CB	d	25	SLA L,(IX+d)		DD	CB	d	45	BIT 0,(IX+d)		DD	CB	d	65	BIT 4,(IX+d)
DD	CB	d	06	RLC (IX+d)		DD	CB	d	26	SLA (IX+d)		DD	CB	d	46	BIT 0,(IX+d)		DD	CB	d	66	BIT 4,(IX+d)
DD	CB	d	07	RLC A,(IX+d)		DD	CB	d	27	SLA A,(IX+d)		DD	CB	d	47	BIT 0,(IX+d)		DD	CB	d	67	BIT 4,(IX+d)
DD	CB	d	08	RRC B,(IX+d)		DD	CB	d	28	SRA B,(IX+d)		DD	CB	d	48	BIT 1,(IX+d)		DD	CB	d	68	BIT 5,(IX+d)
DD	CB	d	09	RRC C,(IX+d)		DD	CB	d	29	SRA C,(IX+d)		DD	CB	d	49	BIT 1,(IX+d)		DD	CB	d	69	BIT 5,(IX+d)
DD	CB	d	0A	RRC D,(IX+d)		DD	CB	d	2A	SRA D,(IX+d)		DD	CB	d	4A	BIT 1,(IX+d)		DD	CB	d	6A	BIT 5,(IX+d)
DD	CB	d	0B	RRC E,(IX+d)		DD	CB	d	2B	SRA E,(IX+d)		DD	CB	d	4B	BIT 1,(IX+d)		DD	CB	d	6B	BIT 5,(IX+d)
DD	CB	d	0C	RRC H,(IX+d)		DD	CB	d	2C	SRA H,(IX+d)		DD	CB	d	4C	BIT 1,(IX+d)		DD	CB	d	6C	BIT 5,(IX+d)
DD	CB	d	0D	RRC L,(IX+d)		DD	CB	d	2D	SRA L,(IX+d)		DD	CB	d	4D	BIT 1,(IX+d)		DD	CB	d	6D	BIT 5,(IX+d)
DD	CB	d	0E	RRC (IX+d)		DD	CB	d	2E	SRA (IX+d)		DD	CB	d	4E	BIT 1,(IX+d)		DD	CB	d	6E	BIT 5,(IX+d)
DD	CB	d	0F	RRC A,(IX+d)		DD	CB	d	2F	SRA A,(IX+d)		DD	CB	d	4F	BIT 1,(IX+d)		DD	CB	d	6F	BIT 5,(IX+d)
DD	CB	d	10	RL B,(IX+d)		DD	CB	d	30	SLL B,(IX+d)		DD	CB	d	50	BIT 2,(IX+d)		DD	CB	d	70	BIT 6,(IX+d)
DD	CB	d	11	RL C,(IX+d)		DD	CB	d	31	SLL C,(IX+d)		DD	CB	d	51	BIT 2,(IX+d)		DD	CB	d	71	BIT 6,(IX+d)
DD	CB	d	12	RL D,(IX+d)		DD	CB	d	32	SLL D,(IX+d)		DD	CB	d	52	BIT 2,(IX+d)		DD	CB	d	72	BIT 6,(IX+d)
DD	CB	d	13	RL E,(IX+d)		DD	CB	d	33	SLL E,(IX+d)		DD	CB	d	53	BIT 2,(IX+d)		DD	CB	d	73	BIT 6,(IX+d)
DD	CB	d	14	RL H,(IX+d)		DD	CB	d	34	SLL H,(IX+d)		DD	CB	d	54	BIT 2,(IX+d)		DD	CB	d	74	BIT 6,(IX+d)
DD	CB	d	15	RL L,(IX+d)		DD	CB	d	35	SLL L,(IX+d)		DD	CB	d	55	BIT 2,(IX+d)		DD	CB	d	75	BIT 6,(IX+d)
DD	CB	d	16	RL (IX+d)		DD	CB	d	36	SLL (IX+d)		DD	CB	d	56	BIT 2,(IX+d)		DD	CB	d	76	BIT 6,(IX+d)
DD	CB	d	17	RL A,(IX+d)		DD	CB	d	37	SLL A,(IX+d)		DD	CB	d	57	BIT 2,(IX+d)		DD	CB	d	77	BIT 6,(IX+d)
DD	CB	d	18	RR B,(IX+d)		DD	CB	d	38	SRL B,(IX+d)		DD	CB	d	58	BIT 3,(IX+d)		DD	CB	d	78	BIT 7,(IX+d)
DD	CB	d	19	RR C,(IX+d)		DD	CB	d	39	SRL C,(IX+d)		DD	CB	d	59	BIT 3,(IX+d)		DD	CB	d	79	BIT 7,(IX+d)
DD	CB	d	1A	RR D,(IX+d)		DD	CB	d	3A	SRL D,(IX+d)		DD	CB	d	5A	BIT 3,(IX+d)		DD	CB	d	7A	BIT 7,(IX+d)
DD	CB	d	1B	RR E,(IX+d)		DD	CB	d	3B	SRL E,(IX+d)		DD	CB	d	5B	BIT 3,(IX+d)		DD	CB	d	7B	BIT 7,(IX+d)
DD	CB	d	1C	RR H,(IX+d)		DD	CB	d	3C	SRL H,(IX+d)		DD	CB	d	5C	BIT 3,(IX+d)		DD	CB	d	7C	BIT 7,(IX+d)
DD	CB	d	1D	RR L,(IX+d)		DD	CB	d	3D	SRL L,(IX+d)		DD	CB	d	5D	BIT 3,(IX+d)		DD	CB	d	7D	BIT 7,(IX+d)
DD	CB	d	1E	RR (IX+d)		DD	CB	d	3E	SRL (IX+d)		DD	CB	d	5E	BIT 3,(IX+d)		DD	CB	d	7E	BIT 7,(IX+d)
DD	CB	d	1F	RR A,(IX+d)		DD	CB	d	3F	SRL A,(IX+d)		DD	CB	d	5F	BIT 3,(IX+d)		DD	CB	d	7F	BIT 7,(IX+d)

How a Z80[®] processes DDCBxxxx

DD	CB	d	80	RES 0,B,(IX+d)		DD	CB	d	A0	RES 4,B,(IX+d)		DD	CB	d	C0	SET 0,B,(IX+d)		DD	CB	d	E0	RES 4,B,(IX+d)
DD	CB	d	81	RES 0,C,(IX+d)		DD	CB	d	A1	RES 4,C,(IX+d)		DD	CB	d	C1	SET 0,C,(IX+d)		DD	CB	d	E1	SET 4,C,(IX+d)
DD	CB	d	82	RES 0,D,(IX+d)		DD	CB	d	A2	RES 4,D,(IX+d)		DD	CB	d	C2	SET 0,D,(IX+d)		DD	CB	d	E2	SET 4,D,(IX+d)
DD	CB	d	83	RES 0,E,(IX+d)		DD	CB	d	A3	RES 4,E,(IX+d)		DD	CB	d	C3	SET 0,E,(IX+d)		DD	CB	d	E3	SET 4,E,(IX+d)
DD	CB	d	84	RES 0,H,(IX+d)		DD	CB	d	A4	RES 4,H,(IX+d)		DD	CB	d	C4	SET 0,H,(IX+d)		DD	CB	d	E4	SET 4,H,(IX+d)
DD	CB	d	85	RES 0,L,(IX+d)		DD	CB	d	A5	RES 4,L,(IX+d)		DD	CB	d	C5	SET 0,L,(IX+d)		DD	CB	d	E5	SET 4,L,(IX+d)
DD	CB	d	86	RES 0,(IX+d)		DD	CB	d	A6	RES 4,(IX+4)		DD	CB	d	C6	SET 0,(IX+d)		DD	CB	d	E6	SET 4,(IX+4)
DD	CB	d	87	RES 0,A,(IX+d)		DD	CB	d	A7	RES 4,A,(IX+d)		DD	CB	d	C7	SET 0,A,(IX+d)		DD	CB	d	E7	SET 4,A,(IX+d)
DD	CB	d	88	RES 1,B,(IX+d)		DD	CB	d	A8	RES 5,B,(IX+d)		DD	CB	d	C8	SET 1,B,(IX+d)		DD	CB	d	E8	SET 5,B,(IX+d)
DD	CB	d	89	RES 1,C,(IX+d)		DD	CB	d	A9	RES 5,C,(IX+d)		DD	CB	d	C9	SET 1,C,(IX+d)		DD	CB	d	E9	SET 5,C,(IX+d)
DD	CB	d	8A	RES 1,D,(IX+d)		DD	CB	d	AA	RES 5,D,(IX+d)		DD	CB	d	CA	SET 1,D,(IX+d)		DD	CB	d	EA	SET 5,D,(IX+d)
DD	CB	d	8B	RES 1,E,(IX+d)		DD	CB	d	AB	RES 5,E,(IX+d)		DD	CB	d	CB	SET 1,E,(IX+d)		DD	CB	d	EB	SET 5,E,(IX+d)
DD	CB	d	8C	RES 1,H,(IX+d)		DD	CB	d	AC	RES 5,H,(IX+d)		DD	CB	d	CC	SET 1,H,(IX+d)		DD	CB	d	EC	SET 5,H,(IX+d)
DD	CB	d	8D	RES 1,L,(IX+d)		DD	CB	d	AD	RES 5,L,(IX+d)		DD	CB	d	CD	SET 1,L,(IX+d)		DD	CB	d	ED	SET 5,L,(IX+d)
DD	CB	d	8E	RES 1,(IX+d)		DD	CB	d	AE	RES 5,(IX+5)		DD	CB	d	CE	SET 1,(IX+d)		DD	CB	d	EE	SET 5,(IX+5)
DD	CB	d	8F	RES 1,A,(IX+d)		DD	CB	d	AF	RES 5,A,(IX+d)		DD	CB	d	CF	SET 1,A,(IX+d)		DD	CB	d	EF	SET 5,A,(IX+d)
DD	CB	d	90	RES 2,B,(IX+d)		DD	CB	d	B0	RES 6,B,(IX+d)		DD	CB	d	D0	SET 2,B,(IX+d)		DD	CB	d	F0	SET 6,B,(IX+d)
DD	CB	d	91	RES 2,C,(IX+d)		DD	CB	d	B1	RES 6,C,(IX+d)		DD	CB	d	D1	SET 2,C,(IX+d)		DD	CB	d	F1	SET 6,C,(IX+d)
DD	CB	d	92	RES 2,D,(IX+d)		DD	CB	d	B2	RES 6,D,(IX+d)		DD	CB	d	D2	SET 2,D,(IX+d)		DD	CB	d	F2	SET 6,D,(IX+d)
DD	CB	d	93	RES 2,E,(IX+d)		DD	CB	d	B3	RES 6,E,(IX+d)		DD	CB	d	D3	SET 2,E,(IX+d)		DD	CB	d	F3	SET 6,E,(IX+d)
DD	CB	d	94	RES 2,H,(IX+d)		DD	CB	d	B4	RES 6,H,(IX+d)		DD	CB	d	D4	SET 2,H,(IX+d)		DD	CB	d	F4	SET 6,H,(IX+d)
DD	CB	d	95	RES 2,L,(IX+d)		DD	CB	d	B5	RES 6,L,(IX+d)		DD	CB	d	D5	SET 2,L,(IX+d)		DD	CB	d	F5	SET 6,L,(IX+d)
DD	CB	d	96	RES 2,(IX+d)		DD	CB	d	B6	RES 6,(IX+6)		DD	CB	d	D6	SET 2,(IX+d)		DD	CB	d	F6	SET 6,(IX+6)
DD	CB	d	97	RES 2,A,(IX+d)		DD	CB	d	B7	RES 6,A,(IX+d)		DD	CB	d	D7	SET 2,A,(IX+d)		DD	CB	d	F7	SET 6,A,(IX+d)
DD	CB	d	98	RES 3,B,(IX+d)		DD	CB	d	B8	RES 7,B,(IX+d)		DD	CB	d	D8	SET 3,B,(IX+d)		DD	CB	d	F8	SET 7,B,(IX+d)
DD	CB	d	99	RES 3,C,(IX+d)		DD	CB	d	B9	RES 7,C,(IX+d)		DD	CB	d	D9	SET 3,C,(IX+d)		DD	CB	d	F9	SET 7,C,(IX+d)
DD	CB	d	9A	RES 3,D,(IX+d)		DD	CB	d	BA	RES 7,D,(IX+d)		DD	CB	d	DA	SET 3,D,(IX+d)		DD	CB	d	FA	SET 7,D,(IX+d)
DD	CB	d	9B	RES 3,E,(IX+d)		DD	CB	d	BB	RES 7,E,(IX+d)		DD	CB	d	DB	SET 3,E,(IX+d)		DD	CB	d	FB	SET 7,E,(IX+d)
DD	CB	d	9C	RES 3,H,(IX+d)		DD	CB	d	BC	RES 7,H,(IX+d)		DD	CB	d	DC	SET 3,H,(IX+d)		DD	CB	d	FC	SET 7,H,(IX+d)
DD	CB	d	9D	RES 3,L,(IX+d)		DD	CB	d	BD	RES 7,L,(IX+d)		DD	CB	d	DD	SET 3,L,(IX+d)		DD	CB	d	FD	SET 7,L,(IX+d)
DD	CB	d	9E	RES 3,(IX+d)		DD	CB	d	BE	RES 7,(IX+7)		DD	CB	d	DE	SET 3,(IX+d)		DD	CB	d	FE	SET 7,(IX+7)
DD	CB	d	9F	RES 3,A,(IX+d)		DD	CB	d	BF	RES 7,A,(IX+d)		DD	CB	d	DF	SET 3,A,(IX+d)		DD	CB	d	FF	SET 7,A,(IX+7)

How a Z80[®] processes EDxx[xxxx]

ED 00		NOP	ED 20		NOP	ED 40		IN B,(C)	ED 60		IN H,(C)
ED 01		NOP	ED 21		NOP	ED 41		OUT (C),B	ED 61		OUT (C),H
ED 02		NOP	ED 22		NOP	ED 42		SBC HL,BC	ED 62		SBC HL,HL
ED 03		NOP	ED 23		NOP	ED 43	n m	LD (word),BC	ED 63	n m	LD (word),HL
ED 04		NOP	ED 24		NOP	ED 44		NEG	ED 64		NEG
ED 05		NOP	ED 25		NOP	ED 45		RETN	ED 65		RETN
ED 06		NOP	ED 26		NOP	ED 46		IM 0	ED 66		IM 0
ED 07		NOP	ED 27		NOP	ED 47		LD I,A	ED 67		RRD
ED 08		NOP	ED 28		NOP	ED 48		IN C(C)	ED 68		IN L,(C)
ED 09		NOP	ED 29		NOP	ED 49		OUT (C),C	ED 69		OUT (C),L
ED 0A		NOP	ED 2A		NOP	ED 4A		ADC HL,BC	ED 6A		ADC HL,HL
ED 0B		NOP	ED 2B		NOP	ED 4B	n m	LD BC,(word)	ED 6B	n m	LD HL,(word)
ED 0C		NOP	ED 2C		NOP	ED 4C		NEG	ED 6C		NEG
ED 0D		NOP	ED 2D		NOP	ED 4D		RETI	ED 6D		RETN
ED 0E		NOP	ED 2E		NOP	ED 4E		IM 0	ED 6E		IM 0
ED 0F		NOP	ED 2F		NOP	ED 4F		LD R,A	ED 6F		RLD
ED 10		NOP	ED 30		NOP	ED 50		IN D,(C)	ED 70		IN (HL),(C)
ED 11		NOP	ED 31		NOP	ED 51		OUT (C),D	ED 71		OUT (C),(HL)
ED 12		NOP	ED 32		NOP	ED 52		SBC HL,DE	ED 72		SBC HL,SP
ED 13		NOP	ED 33		NOP	ED 53	n m	LD (word),DE	ED 73	n m	LD (word),SP
ED 14		NOP	ED 34		NOP	ED 54		NEG	ED 74		NEG
ED 15		NOP	ED 35		NOP	ED 55		RETN	ED 75		RETN
ED 16		NOP	ED 36		NOP	ED 56		IM 1	ED 76		IM 1
ED 17		NOP	ED 37		NOP	ED 57		LD A,I	ED 77		NOP
ED 18		NOP	ED 38		NOP	ED 58		IN E,(C)	ED 78		IN A,(C)
ED 19		NOP	ED 39		NOP	ED 59		OUT (C),E	ED 79		OUT (C),A
ED 1A		NOP	ED 3A		NOP	ED 5A		ADC HL,DE	ED 7A		ADC HL,SP
ED 1B		NOP	ED 3B		NOP	ED 5B	n m	LD DE,(word)	ED 7B	n m	LD SP,(word)
ED 1C		NOP	ED 3C		NOP	ED 5C		NEG	ED 7C		NEG
ED 1D		NOP	ED 3D		NOP	ED 5D		RETN	ED 7D		RETN
ED 1E		NOP	ED 3E		NOP	ED 5E		IM 2	ED 7E		IM 2
ED 1F		NOP	ED 3F		NOP	ED 5F		LD A,R	ED 7F		NOP

How a Z80[®] processes EDxx[xxxx]

ED 80		NOP	ED A0	LDI	ED C0	NOP	ED E0	NOP
ED 81		NOP	ED A1	CPI	ED C1	NOP	ED E1	NOP
ED 82		NOP	ED A2	INI	ED C2	NOP	ED E2	NOP
ED 83		NOP	ED A3	OUTI	ED C3	NOP	ED E3	NOP
ED 84		NOP	ED A4	NOP	ED C4	NOP	ED E4	NOP
ED 85		NOP	ED A5	NOP	ED C5	NOP	ED E5	NOP
ED 86		NOP	ED A6	NOP	ED C6	NOP	ED E6	NOP
ED 87		NOP	ED A7	NOP	ED C7	NOP	ED E7	NOP
ED 88		NOP	ED A8	LDD	ED C8	NOP	ED E8	NOP
ED 89		NOP	ED A9	CPD	ED C9	NOP	ED E9	NOP
ED 8A		NOP	ED AA	IND	ED CA	NOP	ED EA	NOP
ED 8B		NOP	ED AB	OUTD	ED CB	NOP	ED EB	NOP
ED 8C		NOP	ED AC	NOP	ED CC	NOP	ED EC	NOP
ED 8D		NOP	ED AD	NOP	ED CD	NOP	ED ED	NOP
ED 8E		NOP	ED AE	NOP	ED CE	NOP	ED EE	NOP
ED 8F		NOP	ED AF	NOP	ED CF	NOP	ED EF	NOP
ED 90		NOP	ED B0	LDIR	ED D0	NOP	ED F0	NOP
ED 91		NOP	ED B1	CPIR	ED D1	NOP	ED F1	NOP
ED 92		NOP	ED B2	INIR	ED D2	NOP	ED F2	NOP
ED 93		NOP	ED B3	OTIR	ED D3	NOP	ED F3	NOP
ED 94		NOP	ED B4	NOP	ED D4	NOP	ED F4	NOP
ED 95		NOP	ED B5	NOP	ED D5	NOP	ED F5	NOP
ED 96		NOP	ED B6	NOP	ED D6	NOP	ED F6	NOP
ED 97		NOP	ED B7	NOP	ED D7	NOP	ED F7	NOP
ED 98		NOP	ED B8	LDDR	ED D8	NOP	ED F8	NOP
ED 99		NOP	ED B9	CPDR	ED D9	NOP	ED F9	NOP
ED 9A		NOP	ED BA	INDR	ED DA	NOP	ED FA	NOP
ED 9B		NOP	ED BB	OTDR	ED DB	NOP	ED FB	NOP
ED 9C		NOP	ED BC	NOP	ED DC	NOP	ED FC	NOP
ED 9D		NOP	ED BD	NOP	ED DD	NOP	ED FD	NOP
ED 9E		NOP	ED BE	NOP	ED DE	NOP	ED FE	NOP
ED 9F		NOP	ED BF	NOP	ED DF	NOP	ED FF	NOP

How a Z80[®] processes FD[[xx[xx]]xx]

FD 00			NOP : NOP	FD 20	n		NOP : JR NZ,n	FD 40			NOP : LD B,B	FD 60			LD HY,B
FD 01	n	m	NOP : LD BC,mn	FD 21	n	M	LD IY,word	FD 41			NOP : LD B,C	FD 61			LD HY,C
FD 02			NOP : LD (BC),A	FD 22	n	M	LD (word),IY	FD 42			NOP : LD B,D	FD 62			LD HY,D
FD 03			NOP : INC BC	FD 23			INC IY	FD 43			NOP : LD B,E	FD 63			LD HY,E
FD 04			NOP : INC B	FD 24			INC HY	FD 44			LD B,HY	FD 64			LD HY,HY
FD 05			NOP : DEC B	FD 25			DEC HY	FD 45			LD B,LY	FD 65			LD HY,LY
FD 06	n		NOP : LD B,n	FD 26	n		LD HY,byte	FD 46	d		LD B,(IY+d)	FD 66	d		LD H,(IY+d)
FD 07			NOP : RLCA	FD 27			NOP : DAA	FD 47			NOP : LD B,A	FD 67			LD HY,A
FD 08			NOP : EX AF,AF'	FD 28	n		NOP : JR Z,n	FD 48			NOP : LD C,B	FD 68			LD LY,B
FD 09			ADD IY,BC	FD 29			ADD IY,IY	FD 49			NOP : LD C,C	FD 69			LD LY,C
FD 0A			NOP : LD A,(BC)	FD 2A	n	M	LD IY,(word)	FD 4A			NOP : LD C,D	FD 6A			LD LY,D
FD 0B			NOP : DEC BC	FD 2B			DEC IY	FD 4B			NOP : LD C,E	FD 6B			LD LY,E
FD 0C			NOP : INC C	FD 2C			INC LY	FD 4C			LD C,HY	FD 6C			LD LY,HY
FD 0D			NOP : DEC C	FD 2D			DEC LY	FD 4D			LD C,LY	FD 6D			LD LY,LY
FD 0E	n		NOP : LD C,n	FD 2E	n		LD LY,byte	FD 4E	d		LD C,(IY+d)	FD 6E	d		LD L,(IY+d)
FD 0F			NOP : RRCA	FD 2F			NOP : CPL	FD 4F			NOP : LD C,A	FD 6F			LD LY,A
FD 10	n		NOP : DJNZ n	FD 30	n		NOP : JR NC,n	FD 50			NOP : LD D,B	FD 70	d		LD (IY+d),B
FD 11	n	m	NOP : LD DE,mn	FD 31	n	M	NOP : LD SP,mn	FD 51			NOP : LD D,C	FD 71	d		LD (IY+d),C
FD 12			NOP : LD (DE),A	FD 32	n	M	NOP : LD (mn),A	FD 52			NOP : LD D,D	FD 72	d		LD (IY+d),D
FD 13			NOP : INC DE	FD 33			NOP : INC SP	FD 53			NOP : LD D,E	FD 73	d		LD (IY+d),E
FD 14			NOP : INC D	FD 34	d		INC (IY+d)	FD 54			LD D,HY	FD 74	d		LD (IY+d),H
FD 15			NOP : DEC D	FD 35	d		DEC (IY+d)	FD 55			LD D,LY	FD 75	d		LD (IY+d),L
FD 16	n		NOP : LD D,n	FD 36	d	N	LD (IY+d),byte	FD 56	d		LD D,(IY+d)	FD 76			NOP : HALT
FD 17			NOP : RLA	FD 37			NOP : SCF	FD 57			NOP : LD D,A	FD 77	d		LD (IY+d),A
FD 18	n		NOP : JR n	FD 38	n		NOP : JR C,n	FD 58			NOP : LD E,B	FD 78			NOP : LD A,B
FD 19			ADD IY,DE	FD 39			ADD IY,SP	FD 59			NOP : LD E,C	FD 79			NOP : LD A,C
FD 1A			NOP : LD A,(DE)	FD 3A	n	M	NOP : LD A,(mn)	FD 5A			NOP : LD E,D	FD 7A			NOP : LD A,D
FD 1B			NOP : DEC DE	FD 3B			NOP : DEC SP	FD 5B			NOP : LD E,E	FD 7B			NOP : LD A,E
FD 1C			NOP : INC E	FD 3C			NOP : INC A	FD 5C			LD E,HY	FD 7C			LD A,HY
FD 1D			NOP : DEC E	FD 3D			NOP : DEC A	FD 5D			LD E,LY	FD 7D			LD A,LY
FD 1E	n		NOP : LD E,n	FD 3E	n		NOP : LD A,n	FD 5E	d		LD E,(IY+d)	FD 7E	d		LD A,(IY+d)
FD 1F			NOP : RRA	FD 3F			NOP : CCF	FD 5F			NOP : LD E,A	FD 7F			NOP : LD A,A

How a Z80[®] processes FD[[xx[xx]]xx]

FD 80		NOP : ADD A,B	FD A0		NOP : AND B	FD C0		NOP : RET NZ	FD E0		NOP : RET PO
FD 81		NOP : ADD A,C	FD A1		NOP : AND C	FD C1		NOP : POP BC	FD E1		POP IY
FD 82		NOP : ADD A,D	FD A2		NOP : AND D	FD C2	n m	NOP : JP NZ,mn	FD E2	n m	NOP : JP PO,mn
FD 83		NOP : ADD A,E	FD A3		NOP : AND E	FD C3	n m	NOP : JP mn	FD E3		EX (SP),IY
FD 84		ADD A,HY	FD A4		AND HY	FD C4	n m	NOP : CALL NZ,mn	FD E4	n m	NOP : CALL PO,mn
FD 85		ADD A,LY	FD A5		AND LY	FD C5		NOP : PUSH BC	FD E5		PUSH IY
FD 86	d	ADD A,(IY+d)	FD A6	d	AND (IY+d)	FD C6	n	NOP : ADD A,n	FD E6	n	NOP : AND n
FD 87		NOP : ADD A,A	FD A7		NOP : AND A	FD C7		NOP : RST 00H	FD E7		NOP : RST 20H
FD 88		NOP : ADC A,B	FD A8		NOP : XOR B	FD C8		NOP : RET Z	FD E8		NOP : RET PE
FD 89		NOP : ADC A,C	FD A9		NOP : XOR C	FD C9		NOP : RET	FD E9		JP (IY)
FD 8A		NOP : ADC A,D	FD AA		NOP : XOR D	FD CA	n m	NOP : JP Z,mn	FD EA	n m	NOP : JP PE,mn
FD 8B		NOP : ADC A,E	FD AB		NOP : XOR E	FD CB		see Opcode FDCB	FD EB		NOP : EX DE,HL
FD 8C		ADC A,HY	FD AC		XOR HY	FD CC	n m	NOP : CALL Z,mn	FD EC	n m	NOP : CALL PE,mn
FD 8D		ADC A,LY	FD AD		XOR LY	FD CD	n m	NOP : CALL mn	FD ED		NOP : EDxx
FD 8E	d	ADC A,(IY+d)	FD AE	d	XOR (IY+d)	FD CE	n	NOP : ADC A,n	FD EE	n	NOP : XOR n
FD 8F		NOP : ADC A,A	FD AF		NOP : XOR A	FD CF		NOP : RST 08H	FD EF		NOP : RST 28H
FD 90		NOP : SUB B	FD B0		NOP : OR B	FD D0		NOP : RET NC	FD F0		NOP : RET P
FD 91		NOP : SUB C	FD B1		NOP : OR C	FD D1		NOP : POP DE	FD F1		NOP : POP AF
FD 92		NOP : SUB D	FD B2		NOP : OR D	FD D2	n m	NOP : JP NC,mn	FD F2	n m	NOP : JP P,mn
FD 93		NOP : SUB E	FD B3		NOP : OR E	FD D3	n	NOP : OUT (m),A	FD F3		NOP : DI
FD 94		SUB HY	FD B4		OR HY	FD D4	n m	NOP : CALL NC,mn	FD F4	n m	NOP : CALL P,mn
FD 95		SUB LY	FD B5		OR LY	FD D5		NOP : PUSH DE	FD F5		NOP : PUSH AF
FD 96	d	SUB (IY+d)	FD B6	d	OR (IY+d)	FD D6	n	NOP : SUB n	FD F6	n	NOP : OR n
FD 97		NOP : SUB A	FD B7		NOP : OR A	FD D7		NOP : RST 10H	FD F7		NOP : RST 30H
FD 98		NOP : SBC A,B	FD B8		NOP : CP B	FD D8		NOP : RET C	FD F8		NOP : RET M
FD 99		NOP : SBC A,C	FD B9		NOP : CP C	FD D9		NOP : EXX	FD F9		LD SP,IY
FD 9A		NOP : SBC A,D	FD BA		NOP : CP D	FD DA	n m	NOP : JP C,mn	FD FA	n m	NOP : JP M,mn
FD 9B		NOP : SBC A,E	FD BB		NOP : CP E	FD DB	n	NOP : IN A,(m)	FD FB		NOP : EI
FD 9C		SBC A,HY	FD BC		CP HY	FD DC	n m	NOP : CALL C,mn	FD FC	n m	NOP : CALL M,mn
FD 9D		SBC A,LY	FD BD		CP LY	FD DD		NOP : DDxx[xx[xx]]	FD FD		NOP : FDxx[xx[xx]]
FD 9E	d	SBC A,(IY+d)	FD BE	d	CP (IY+d)	FD DE	n	NOP : SBC A,n	FD FE	n	NOP : CP n
FD 9F		NOP : SBC A,A	FD BF		NOP : CP A	FD DF		NOP : RST 18H	FD FF		NOP : RST 38H

How a Z80[®] processes FDCBxxxx

FD CB d 00	RLC B,(IY+d)	FD CB d 20	SLA B,(IY+d)	FD CB d 40	BIT 0,(IY+d)	FD CB d 60	BIT 4,(IY+d)
FD CB d 01	RLC C,(IY+d)	FD CB d 21	SLA C,(IY+d)	FD CB d 41	BIT 0,(IY+d)	FD CB d 61	BIT 4,(IY+d)
FD CB d 02	RLC D,(IY+d)	FD CB d 22	SLA D,(IY+d)	FD CB d 42	BIT 0,(IY+d)	FD CB d 62	BIT 4,(IY+d)
FD CB d 03	RLC E,(IY+d)	FD CB d 23	SLA E,(IY+d)	FD CB d 43	BIT 0,(IY+d)	FD CB d 63	BIT 4,(IY+d)
FD CB d 04	RLC H,(IY+d)	FD CB d 24	SLA H,(IY+d)	FD CB d 44	BIT 0,(IY+d)	FD CB d 64	BIT 4,(IY+d)
FD CB d 05	RLC L,(IY+d)	FD CB d 25	SLA L,(IY+d)	FD CB d 45	BIT 0,(IY+d)	FD CB d 65	BIT 4,(IY+d)
FD CB d 06	RLC (IY+d)	FD CB d 26	SLA (IY+d)	FD CB d 46	BIT 0,(IY+d)	FD CB d 66	BIT 4,(IY+d)
FD CB d 07	RLC A,(IY+d)	FD CB d 27	SLA A,(IY+d)	FD CB d 47	BIT 0,(IY+d)	FD CB d 67	BIT 4,(IY+d)
FD CB d 08	RRC B,(IY+d)	FD CB d 28	SRA B,(IY+d)	FD CB d 48	BIT 1,(IY+d)	FD CB d 68	BIT 5,(IY+d)
FD CB d 09	RRC C,(IY+d)	FD CB d 29	SRA C,(IY+d)	FD CB d 49	BIT 1,(IY+d)	FD CB d 69	BIT 5,(IY+d)
FD CB d 0A	RRC D,(IY+d)	FD CB d 2A	SRA D,(IY+d)	FD CB d 4A	BIT 1,(IY+d)	FD CB d 6A	BIT 5,(IY+d)
FD CB d 0B	RRC E,(IY+d)	FD CB d 2B	SRA E,(IY+d)	FD CB d 4B	BIT 1,(IY+d)	FD CB d 6B	BIT 5,(IY+d)
FD CB d 0C	RRC H,(IY+d)	FD CB d 2C	SRA H,(IY+d)	FD CB d 4C	BIT 1,(IY+d)	FD CB d 6C	BIT 5,(IY+d)
FD CB d 0D	RRC L,(IY+d)	FD CB d 2D	SRA L,(IY+d)	FD CB d 4D	BIT 1,(IY+d)	FD CB d 6D	BIT 5,(IY+d)
FD CB d 0E	RRC (IY+d)	FD CB d 2E	SRA (IY+d)	FD CB d 4E	BIT 1,(IY+d)	FD CB d 6E	BIT 5,(IY+d)
FD CB d 0F	RRC A,(IY+d)	FD CB d 2F	SRA A,(IY+d)	FD CB d 4F	BIT 1,(IY+d)	FD CB d 6F	BIT 5,(IY+d)
FD CB d 10	RL B,(IY+d)	FD CB d 30	SLL B,(IY+d)	FD CB d 50	BIT 2,(IY+d)	FD CB d 70	BIT 6,(IY+d)
FD CB d 11	RL C,(IY+d)	FD CB d 31	SLL C,(IY+d)	FD CB d 51	BIT 2,(IY+d)	FD CB d 71	BIT 6,(IY+d)
FD CB d 12	RL D,(IY+d)	FD CB d 32	SLL D,(IY+d)	FD CB d 52	BIT 2,(IY+d)	FD CB d 72	BIT 6,(IY+d)
FD CB d 13	RL E,(IY+d)	FD CB d 33	SLL E,(IY+d)	FD CB d 53	BIT 2,(IY+d)	FD CB d 73	BIT 6,(IY+d)
FD CB d 14	RL H,(IY+d)	FD CB d 34	SLL H,(IY+d)	FD CB d 54	BIT 2,(IY+d)	FD CB d 74	BIT 6,(IY+d)
FD CB d 15	RL L,(IY+d)	FD CB d 35	SLL L,(IY+d)	FD CB d 55	BIT 2,(IY+d)	FD CB d 75	BIT 6,(IY+d)
FD CB d 16	RL (IY+d)	FD CB d 36	SLL (IY+d)	FD CB d 56	BIT 2,(IY+d)	FD CB d 76	BIT 6,(IY+d)
FD CB d 17	RL A,(IY+d)	FD CB d 37	SLL A,(IY+d)	FD CB d 57	BIT 2,(IY+d)	FD CB d 77	BIT 6,(IY+d)
FD CB d 18	RR B,(IY+d)	FD CB d 38	SRL B,(IY+d)	FD CB d 58	BIT 3,(IY+d)	FD CB d 78	BIT 7,(IY+d)
FD CB d 19	RR C,(IY+d)	FD CB d 39	SRL C,(IY+d)	FD CB d 59	BIT 3,(IY+d)	FD CB d 79	BIT 7,(IY+d)
FD CB d 1A	RR D,(IY+d)	FD CB d 3A	SRL D,(IY+d)	FD CB d 5A	BIT 3,(IY+d)	FD CB d 7A	BIT 7,(IY+d)
FD CB d 1B	RR E,(IY+d)	FD CB d 3B	SRL E,(IY+d)	FD CB d 5B	BIT 3,(IY+d)	FD CB d 7B	BIT 7,(IY+d)
FD CB d 1C	RR H,(IY+d)	FD CB d 3C	SRL H,(IY+d)	FD CB d 5C	BIT 3,(IY+d)	FD CB d 7C	BIT 7,(IY+d)
FD CB d 1D	RR L,(IY+d)	FD CB d 3D	SRL L,(IY+d)	FD CB d 5D	BIT 3,(IY+d)	FD CB d 7D	BIT 7,(IY+d)
FD CB d 1E	RR (IY+d)	FD CB d 3E	SRL (IY+d)	FD CB d 5E	BIT 3,(IY+d)	FD CB d 7E	BIT 7,(IY+d)
FD CB d 1F	RR A,(IY+d)	FD CB d 3F	SRL A,(IY+d)	FD CB d 5F	BIT 3,(IY+d)	FD CB d 7F	BIT 7,(IY+d)

How a Z80[®] processes FDCBxxxx

FD CB d 80	RES 0,B,(IY+d)	FD CB d A0	RES 4,B,(IY+d)	FD CB d C0	SET 0,B,(IY+d)	FD CB d E0	RES 4,B,(IY+d)
FD CB d 81	RES 0,C,(IY+d)	FD CB d A1	RES 4,C,(IY+d)	FD CB d C1	SET 0,C,(IY+d)	FD CB d E1	SET 4,C,(IY+d)
FD CB d 82	RES 0,D,(IY+d)	FD CB d A2	RES 4,D,(IY+d)	FD CB d C2	SET 0,D,(IY+d)	FD CB d E2	SET 4,D,(IY+d)
FD CB d 83	RES 0,E,(IY+d)	FD CB d A3	RES 4,E,(IY+d)	FD CB d C3	SET 0,E,(IY+d)	FD CB d E3	SET 4,E,(IY+d)
FD CB d 84	RES 0,H,(IY+d)	FD CB d A4	RES 4,H,(IY+d)	FD CB d C4	SET 0,H,(IY+d)	FD CB d E4	SET 4,H,(IY+d)
FD CB d 85	RES 0,L,(IY+d)	FD CB d A5	RES 4,L,(IY+d)	FD CB d C5	SET 0,L,(IY+d)	FD CB d E5	SET 4,L,(IY+d)
FD CB d 86	RES 0,(IY+d)	FD CB d A6	RES 4,(IX+4)	FD CB d C6	SET 0,(IY+d)	FD CB d E6	SET 4,(IX+4)
FD CB d 87	RES 0,A,(IY+d)	FD CB d A7	RES 4,A,(IY+d)	FD CB d C7	SET 0,A,(IY+d)	FD CB d E7	SET 4,A,(IY+d)
FD CB d 88	RES 1,B,(IY+d)	FD CB d A8	RES 5,B,(IY+d)	FD CB d C8	SET 1,B,(IY+d)	FD CB d E8	SET 5,B,(IY+d)
FD CB d 89	RES 1,C,(IY+d)	FD CB d A9	RES 5,C,(IY+d)	FD CB d C9	SET 1,C,(IY+d)	FD CB d E9	SET 5,C,(IY+d)
FD CB d 8A	RES 1,D,(IY+d)	FD CB d AA	RES 5,D,(IY+d)	FD CB d CA	SET 1,D,(IY+d)	FD CB d EA	SET 5,D,(IY+d)
FD CB d 8B	RES 1,E,(IY+d)	FD CB d AB	RES 5,E,(IY+d)	FD CB d CB	SET 1,E,(IY+d)	FD CB d EB	SET 5,E,(IY+d)
FD CB d 8C	RES 1,H,(IY+d)	FD CB d AC	RES 5,H,(IY+d)	FD CB d CC	SET 1,H,(IY+d)	FD CB d EC	SET 5,H,(IY+d)
FD CB d 8D	RES 1,L,(IY+d)	FD CB d AD	RES 5,L,(IY+d)	FD CB d CD	SET 1,L,(IY+d)	FD CB d ED	SET 5,L,(IY+d)
FD CB d 8E	RES 1,(IY+d)	FD CB d AE	RES 5,(IX+5)	FD CB d CE	SET 1,(IY+d)	FD CB d EE	SET 5,(IX+5)
FD CB d 8F	RES 1,A,(IY+d)	FD CB d AF	RES 5,A,(IY+d)	FD CB d CF	SET 1,A,(IY+d)	FD CB d EF	SET 5,A,(IY+d)
FD CB d 90	RES 2,B,(IY+d)	FD CB d B0	RES 6,B,(IY+d)	FD CB d D0	SET 2,B,(IY+d)	FD CB d F0	SET 6,B,(IY+d)
FD CB d 91	RES 2,C,(IY+d)	FD CB d B1	RES 6,C,(IY+d)	FD CB d D1	SET 2,C,(IY+d)	FD CB d F1	SET 6,C,(IY+d)
FD CB d 92	RES 2,D,(IY+d)	FD CB d B2	RES 6,D,(IY+d)	FD CB d D2	SET 2,D,(IY+d)	FD CB d F2	SET 6,D,(IY+d)
FD CB d 93	RES 2,E,(IY+d)	FD CB d B3	RES 6,E,(IY+d)	FD CB d D3	SET 2,E,(IY+d)	FD CB d F3	SET 6,E,(IY+d)
FD CB d 94	RES 2,H,(IY+d)	FD CB d B4	RES 6,H,(IY+d)	FD CB d D4	SET 2,H,(IY+d)	FD CB d F4	SET 6,H,(IY+d)
FD CB d 95	RES 2,L,(IY+d)	FD CB d B5	RES 6,L,(IY+d)	FD CB d D5	SET 2,L,(IY+d)	FD CB d F5	SET 6,L,(IY+d)
FD CB d 96	RES 2,(IY+d)	FD CB d B6	RES 6,(IX+6)	FD CB d D6	SET 2,(IY+d)	FD CB d F6	SET 6,(IX+6)
FD CB d 97	RES 2,A,(IY+d)	FD CB d B7	RES 6,A,(IY+d)	FD CB d D7	SET 2,A,(IY+d)	FD CB d F7	SET 6,A,(IY+d)
FD CB d 98	RES 3,B,(IY+d)	FD CB d B8	RES 7,B,(IY+d)	FD CB d D8	SET 3,B,(IY+d)	FD CB d F8	SET 7,B,(IY+d)
FD CB d 99	RES 3,C,(IY+d)	FD CB d B9	RES 7,C,(IY+d)	FD CB d D9	SET 3,C,(IY+d)	FD CB d F9	SET 7,C,(IY+d)
FD CB d 9A	RES 3,D,(IY+d)	FD CB d BA	RES 7,D,(IY+d)	FD CB d DA	SET 3,D,(IY+d)	FD CB d FA	SET 7,D,(IY+d)
FD CB d 9B	RES 3,E,(IY+d)	FD CB d BB	RES 7,E,(IY+d)	FD CB d DB	SET 3,E,(IY+d)	FD CB d FB	SET 7,E,(IY+d)
FD CB d 9C	RES 3,H,(IY+d)	FD CB d BC	RES 7,H,(IY+d)	FD CB d DC	SET 3,H,(IY+d)	FD CB d FC	SET 7,H,(IY+d)
FD CB d 9D	RES 3,L,(IY+d)	FD CB d BD	RES 7,L,(IY+d)	FD CB d DD	SET 3,L,(IY+d)	FD CB d FD	SET 7,L,(IY+d)
FD CB d 9E	RES 3,(IY+d)	FD CB d BE	RES 7,(IX+7)	FD CB d DE	SET 3,(IY+d)	FD CB d FE	SET 7,(IX+7)
FD CB d 9F	RES 3,A,(IY+d)	FD CB d BF	RES 7,A,(IY+d)	FD CB d DF	SET 3,A,(IY+d)	FD CB d FF	SET 7,A,(IX+7)

INDEX

- A**
A command 3-3
ADC
 8-bit 5-13
 16-bit 5-121
ADD
 8-bit 5-16
 16-bit 5-122
ADDR. 1-8
addresses 1-8
AND, logical (source code). 1-10, 3-5
AND, mnemonic 5-19
arithmetic
 8-bit 5-13
 16-bit 5-121
assemble. 3-3
- B**
B command 3-5
base (constant) 1-9
bit manipulation. 5-83
BIT 5-83
branch. 5-129
- C**
C command 3-6, 3-7
calculator. 3-5
CALL. 5-136
CCF 5-41
change. 3-6
COMM. 2-2
Commands
 A. 3-3
 B. 3-5
 C. 3-6, 3-7
 D. 3-8
 E. 3-9
 F. 3-10
 G. 3-10
 H. 3-11
 I. 3-12
 J. 3-12
 K. 3-13
 L. 3-13
 M. 3-14
 NC 3-15
 NE 3-15
 O. 3-15
 P. 3-15
 QU 3-16
 R. 3-16
 S. 3-16
 T. 3-17
 U. 3-17
 V. 3-18
 X. 3-18
comment 1-7
Constant. 1-9
copy, source text 3-14
copy/move (instructions). 5-143
CP. 5-22
CPD 5-147
CPDR. 5-148
CPI 5-149
CPIR. 5-150
CPL 5-43
CPU control (instructions). 5-41
Current line. 1-5
- D**
D command 3-8
DAA 5-45
DB. 2-3
DEC
 8-bit 5-25
 16-bit 5-124
DEFB. 2-3
DEFL. 2-3
DEFM. 2-4
DEFS. 2-4
DEFW. 2-5
delete. 3-8
delete, file. 3-13
directory 3-18
DI. 5-48
DJNZ. 5-135
DL. 2-3
DM. 2-4
DOS errors (load errors). 4-3
DS. 2-4
duplicate, source text. 3-14
DW. 2-5

INDEX

- E**
 - E command 3-9
 - E option. 3-3
 - EDTASM format 1-2, 3-13, 3-16
 - EI. 5-49
 - END 2-5
 - ENIF. 2-6
 - erase file. 3-13
 - ERR 2-7
 - EQU 2-7
 - errors. 4-1
 - EX
 - 8-bit 5-10
 - 16-bit 5-113, 5-119
 - execution times 6-6
 - exit (exit ZEUS). 3-16
 - expression. 1-10
 - EXX 5-114

- F**
 - F command 3-10
 - fields. 1-7
 - FILENAMES 1-3
 - find labels 3-10
 - flags 6-1
 - forms, set printer values 3-10
 - forms, top of 3-3, 3-12

- G**
 - G command 3-10
 - G option. 3-3
 - general purpose 5-41
 - GET 2-8
 - global changes. 3-6
 - glossary. 1-5

- H**
 - H command 3-11
 - H option. 3-3
 - HALT. 5-47
 - hack, edit. 3-9

- I**
 - I command 3-12
 - I/O 1-4
 - IF. 2-6
 - IM. 5-50
 - IN. 5-91
 - INC
 - 8-bit 5-27
 - 16-bit 5-126
 - IND 5-93
 - INDR. 5-94
 - INI 5-95
 - INIR. 5-96
 - initialization. 1-2
 - input and output. 5-91
 - insert, edit. 3-9
 - insert, source text 3-12
 - Instruction 1-7
 - instructions by mnemonic. 6-7
 - instructions by object code . . . 6-19

- J**
 - J command 3-12
 - J option. 3-3
 - JP. 5-129
 - JR. 5-133

- K**
 - K command 3-13
 - keystroke, single 3-2
 - KILL. 3-13

- L**
 - L command 3-13
 - Label 1-6, 1-7
 - label, find 3-10
 - label, reference. 3-16
 - LD
 - 8-bit 5-1
 - 16-bit 5-103
 - LDD 5-143
 - LDDR. 5-144
 - LDI 5-145
 - LDIR. 5-146
 - line length 1-7
 - line number 1-7
 - LIST. 2-8
 - list source text. 3-11, 3-15
 - load. 3-13
 - load (instructions)
 - 8-bit 5-1
 - 16-bit 5-103
 - logic 8-bit 5-19
 - logical AND 1-10, 3-5
 - logical OR. 1-10, 3-5
 - logical XOR 1-10, 3-5
 - LPrint. 1-4, 2-9, 3-3, 3-11, 3-12

INDEX

- M**
M command 3-14
manual notation 1-5
memory usage. 3-17
MESP. 2-9
MESV. 2-9
Mnemonic. 1-6
move. 3-14
multiplication. 1-10, 3-5
- N**
N option. 3-3
NC command. 3-14
NE command. 3-14
NEG 5-44
NOP 5-46
notation, manual. 1-5
- O**
O command 3-15
O option. 3-3
Object code 1-6, 3-3
OBJCODE 1-8
opcode. 1-7
opcode reference. 3-15
operand 1-7
operand reference 3-15
Operator. 1-6
operation 1-10
OR, logical (source code) . 1-10, 3-5
OR, mnemonic. 5-29
ORG 2-9
OTDR. 5-100
OTIR. 5-102
OUT 5-97
Out of memory 3-7, 4-2, 4-4
Out of range. 4-5
OUTD. 5-99
OUTI. 5-101
Overflow. 4-5
- P**
P command 3-15
P option. 3-3
PAGE. 2-10
pause 3-2
POP 5-117
print 3-15
PROG. 1-2
pseudo-op 1-6, 1-7, 1-8, 2-1
PUSH. 5-115
- Q**
Q option. 3-4
QU command. 3-15
- R**
R command 3-16
raw data to printer 3-12
recover text. 3-18
Redefinition. 2-3, 4-6
reference 3-15, 3-16, 4-12
reference opcodes, operands . . 3-15
reference error 4-2
Relative address.
. 1-6, 1-9, 2-7, 2-9,
2-12
RELNUM. 1-5, 3-9, 3-12
RELNUM1 3-6, 3-8, 3-11
RELNUM2 3-6, 3-8, 3-11
RELNUM3 3-14
remove files. 3-13
remove remarks. 3-15
RES 5-85
RET 5-138
RETI. 5-140
RETN. 5-141
RL. 5-57
RLA 5-53
RLC 5-60
RLCA. 5-54
RLD 5-81
rotate and shift. 5-53
RR. 5-63
RRA 5-55
RRC 5-66
RRCA. 5-56
RRD 5-82
RST 5-142

INDEX

- S**
S command 3-16
S option. 3-4
save. 3-16
SBC
 8-bit 5-32
 16-bit 5-128
SBTL. 2-10
SCF 5-42
search. 5-147
SET 5-88
shift 1-10, 3-5
SHOW. 2-8
SLA 5-69
SLL 5-72
sort. 3-4, 3-17
Source code 1-6
source text format. 4-10
speed 1-2, 3-18
SRA 5-75
SRL 5-78
SUB 5-35
subtraction 1-10, 3-5
symbolic code 1-7
- T**
T command 3-17
T option. 3-4
table, label. 3-4, 3-17
target. 4-2
terminal errors 4-4
Text buffer 1-6
Text end. 3-15, 3-16, 4-3
TITL. 2-10
token 4-9
- U**
U command 3-17
U option. 3-4
Undefined label 4-6
Unrecognized character. 4-4
undocumented instructions 7-1
usage 3-17
- V**
V command 3-18
- W**
WAIT. 2-11
Warning errors. 4-4, 4-7
- X**
X command 3-18
XOR, logical. 1-10, 3-5
XOR, mnemonic 5-38
- Z**
Z80Z. 1-2
zaps (printer initialization) 3-12