

The CP/M® Z-80® Microcomputer

ZSIDTM
SYMBOLIC INSTRUCTION DEBUGGER

COMMAND SUMMARY

Z-80 VERSION

DIGITAL RESEARCH

COPYRIGHT

Copyright © 1979, 1981 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California, 93950.

TRADEMARKS

CP/M is a registered trademark of Digital Research. Z-80 is a registered trademark of Zilog. ZSID is a trademark of Digital Research.

All information contained herein is proprietary to Digital Research.

ABOUT THIS MANUAL

The starting point for this PDF file was the plain text contained in the file zsid.txt (OCR'd from a CP/M manual) zipped in the file zsid-m.zip, that was downloaded from www.cpm.z80.de, The Unofficial CP/M Web Site.

Please, enter in this site to know how you can help to the CP/M community working on CP/M manuals.

Miguel I. García López, 24 May 2007.

ZSID COMMAND SUMMARY

- 1.1 Startup
- 1.2 Response
- 1.3 Letter commands
- 1.4 Command line
- 1.5 Literal numbers
- 1.6 Decimal numbers
- 1.7 Characters
- 1.8 Symbol references
- 1.9 Qualified symbols
- 1.10 Symbolic expressions
- 1.11 Unary plus/minus

- 2.1 Assemble
- 2.2 Call
- 2.3 Display memory
- 2.4 Fill memory
- 2.5 Go to program
- 2.6 Hex values
- 2.7 Input line
- 2.8 List code
- 2.9 Move memory
- 2.10 Pass counter
- 2.11 Read code/symbols
- 2.12 Set memory
- 2.13 Trace mode
- 2.14 Untrace mode
- 2.15 Examine cpu state

- 3.1 ZSID utilities
- 3.2 The hist utility
- 3.3 The trace utility

- 4.1 Implementation notes

- 5.1 Z80 mnemonics
- 5.2 Z80-cpu instruction set

1.1 / STARTUP

- (1) ZSID
- (2) ZSID x.y
- (3) ZSID x.HEX
- (4) ZSID x.UTL
- (5) ZSID x.y u.v

Form (1) starts ZSID without a test program, (2) loads the test program x.y (y is normally COM), (3) loads x.HEX in Intel "hex" format, (4) loads and executes utility x, (5) loads x.y with the symbol table u.v (normally x.SYM).

Example:

```
ZSID SORT.COM SORT.SYM
```

1.2 / RESPONSE

- (1) #
- (2) SYMBOLS
- (3) NEXT PC END
nnnn pppp eeee

Form (1) indicates ZSID is ready to accept commands, (2) indicates machine code loaded, commencing symbol table load, (3) shows successful machine code and/or symbol load where nnnn, pppp, and eeee are hexadecimal values giving the next unfilled machine code location, the initial program counter, and the last free memory location, respectively.

1.3 / LETTER COMMANDS

A	Assemble	M	Move
C	Call	P	Pass Point
D	Display	R	Read
F	Fill Memory	S	Set Memory
G	Go	T	Trace
H	Hex	U	Untrace
I	Input Line	X	Examine
L	List Mnemonics		

1.4 / COMMAND LINE

ZSID reads commands from the system console following the # prompt. Each command line is based upon the command letter and optional symbolic expressions. All CP/M line editing is available on 64 character lines terminated by carriage returns. A space serves as a comma delimiter.

ZSID terminates whenever control-C is typed.

1.5 / LITERAL NUMBERS

ZSID uses the hexadecimal number base, consisting of the decimal digits 0-9 along with the hex digits A-F. Numbers exceeding four digits are truncated to the right.

Examples are:

```
30 3F 3f FF3E F3
```

1.6 / DECIMAL NUMBERS

Decimal numbers are preceded by a #, and consist of decimal digits 0-9. Numbers exceeding 65535 are truncated to the rightmost 16 bits.

Examples are:

```
#48 #9999 #65535 #0
```

1.7 / CHARACTERS

ZSID accepts graphic ASCII characters within paired string apostrophes (''). Strings of length greater than two are truncated to the right. The rightmost character of a two character string becomes the least significant byte. A one character string has a high order 00 byte, zero length strings are disallowed, and a pair of apostrophes within a string reduces to a single apostrophe. Lower case letters are not translated in strings.

Examples are:

```
'a' 'A' 'xy' '#' ''
```

1.8 / SYMBOL REFERENCES

ZSID symbolic expressions may involve symbol references when a symbol table is present:

- (1) .s
- (2) @s
- (3) =s

Form (1) denotes the address of symbol s, (2) denotes the 16-bit value at .s, (3) denotes the 8-bit value at .s, where s is a sequence of characters matching a symbol table element.

1.9 / QUALIFIED SYMBOLS

ZSID searches for a symbol match starting at the first symbol loaded until the first symbol matches. When duplicate symbols exist, a qualified reference of the form

```
s1/s2/.../sn
```

matches symbols from left to right as the search proceeds sequentially through the symbol table.

An example is:

```
ALPHA/GAMMA/I
```

1.10 / SYMBOLIC EXPRESSIONS

Expressions consist of a left to right sequence of literal numbers, decimal numbers, character strings, and symbol references, separated by plus ("+") and minus ("-") operators. Values are added or subtracted, accordingly, with no overflow checks, to produce the final 16-bit result.

A leading minus, as in -x, is computed as 0-x. A leading plus, as in +x, is computed as x'+x, where x' is the value of the last expression typed. A sequence of n ^'s produces the n'th stacked value in the program under test (see the G command). Blanks are not allowed within expressions.

Examples are given with individual commands.

1.11 / UNARY PLUS/MINUS

For convenience, symbolic expressions may be preceded by either a plus or minus sign taking the forms

- (1) +x
- (2) -x

where x is a symbolic expression. Form (1) is computed as x'+x, where x' is the value of the last symbolic expression typed by the operator, or zero if no expression has been entered.

For example

```
D.GAMMA+5,+#10
```

is equivalent to

```
D.GAMMA+5,.GAMMA+5+#10
```

Form (2) is computed as 0-X and thus

```
R-100
```

is equivalent to

```
RFF00
```

2.1 / ASSEMBLE

- (1) As
- (2) A
- (3) -A

Form (1) begins in-line assembly at location *s*, where each successive address is displayed until a null line or "." is entered by the operator. Form (2) is equivalent to (1) with assumed starting address derived from last assembled, listed, or traced address. Form (3) removes the assembler/ disassembler module, discards existing symbol information, and disables subsequent A or L commands. In this case, machine hex code is displayed in subsequent traces.

Examples:

```
A100
A#100
A.CRLF+5
A@GAMMA+@X-==I
A+30
```

2.2 / CALL

- (1) Cs
- (2) Cs,b
- (3) Cs,b,d

Form (1) performs a direct call from ZSID to location *s* in memory, without disturbing the CPU state of the program under test, and is most often used with ZSID Utilities. In this case, registers BC=0000, DE=0000. Form (2) calls *s* with data BC=b, DE=0000, while form (3) also fills DE=d.

Examples:

```
C100
C#4096
C.DISPLAY
C@JMPVEC+=X
C.CRLF,#34
C.CRLF,@X,+=X
```

2.3 / DISPLAY MEMORY

- (1) Ds
- (2) Ds,f
- (3) D
- (4) D,f
- (5) DWS
- (6) DWS,f
- (7) DW
- (8) DW,f

Form (1) types memory contents in 8-bit format starting at location s for 1/2 screen with graphic ASCII to the right of each line, (2) is similar, but ends at location f. Form (3) continues the display from the last displayed location, or the value of the HL register pair following CPU state display, for 1/2 screen, (4) is similar, but terminates at location f. Forms (5) through (8) are equivalent to (1) through (4), but display in word format (16-bits).

Examples:

```
DF3F
D#100,#200
D.gamma,.DELTA+#30
d,.GAMMA
DW@ALPHA,+#100
```

2.4 / FILL MEMORY

Fs,f,d

Fills memory with 8-bit data d starting at location s, continuing through location f.

Examples:

```
F100,3FF,ff
f.gamma,+#100,#23
F@ALPHA,+=I,=X
```

2.5 / GO TO PROGRAM

- (1) G
- (2) Gp
- (3) G,a
- (4) Gp,a
- (5) G,a,b
- (6) Gp,a,b
- (7) -G...

Form (1) starts the program under test from the current PC without breakpoints. Execution is in real time. Form (2) is equivalent, but sets PC=p before execution, (3) starts from the current PC with a breakpoint at location a, (4) is similar to (3) but sets the PC to p. Form (5) is equivalent to (3) but sets breakpoints at a and b, while (6) presets the PC to p before execution. Upon encountering a breakpoint (or an externally provided RST 7), the break address is printed in the form:

*nnnn

and the optional breakpoints are cleared. Forms given by (7) parallel (1) through (6), except "pass points" are not traced until the corresponding pass count becomes zero (see P command). The symbol "^" in an expression produces the topmost stacked value, which is used to set a break following a subroutine call. Given that a breakpoint has occurred at a subroutine, the command

G,^

continues execution with a return breakpoint set.

Examples:

```
G100
G100,103
G.CRLF,.PRINT,#1024
G@JMPVEC+=I,.ENDC,.ERRC
G,.errsub
G,.ERRSUB,+30
-G100,+10,+10
```

2.6 / HEX VALUES

- (1) Ha,b
- (2) Ha
- (3) H

Form (1) produces the hexadecimal sum (a+b) and difference (a-b) of operands. Form (2) performs number conversion by typing the value of a in the format:

```
hhhh #dddd 'c' .ssss
```

where hhhh is a's hex value, dddd is the decimal value, c is the ASCII value, if it exists, and ssss is the symbolic value, if it exists. Form (3) prints the hex values for each symbol table element (abort with rubout).

Examples:

```
H100,200
H#1000,#965
H.GAMMA+=I,@ALPHA-#10
H#53
H@X+=Y-5
```

2.7 / INPUT LINE

```
Ic1c2...cn
```

Initializes default low memory areas for the R command or the program under test, as if the characters c1 through cn had been read and setup at the console command processor level. Default FCB's are initialized, and the default buffer is set to the initial input line.

Examples:

```
I x.dat
ix.inp y.out
I a:x.inp b:y.out $-p
ITEST.COM
I TEST.HEX TEST.SYM
```

2.8 / LIST CODE

- (1) Ls
- (2) Ls,f
- (3) L
- (4) -L...

Form (1) lists disassembled machine code starting at location s for 1/2 screen, (2) lists mnemonics from location s through f (abort typeouts with rubout). Form (3) lists mnemonics from the last listed, assembled, or traced location for 1/2 screen. Form (4) parallels (1) through (3), but labels and symbolic operands are not printed. Labels are printed in the form

```
ssss:
```


ahead of the lines to which they correspond.

Non-Z80 mnemonics are printed as

```
??= hh
```

where hh is the hex value at that location.

Examples:

```
L100
L#1024,#1034
L.CRLF
L@ICALL,+30
-L.PRBUFFER+=I,'A'
```

2.9 / MOVE MEMORY

```
Ms,h,d
```

Move data values from start address s through h address h to destination address d. Data areas may overlap during the move process.

Examples:

```
M100,1FF,300
M.X,.Y,.Z
M.GAMMA,+FF,.DELTA
M@alpha+=x,+50,+100
```

2.10 / PASS COUNTER

```
(1) Pp
(2) Pp,c
(3) P
(4) -Pp
(5) -P
```

A "pass point" is a program counter location to monitor during execution of a test program. A pass point has an associated "pass counter" in the range 1-FF (0-#255) which is decremented each time the test program executes the pass point address. When a pass count reaches 1, the pass point becomes a permanent breakpoint and the pass count remains at 1. Unlike a temporary breakpoint (see G), pass points with pass count 1 stop execution following execution of the instruction at the break address. Form (1) sets a pass point at address p with pass count 1, (2) sets pass point p with pass count c, (3) displays active pass points and counts, (4) clears the pass point at p (equivalent to Pp,0), and (5) clears all pass points. Up to 8 pass points can be active at any time. CPU registers are displayed when executing a pass point, with the header

```
nn PASS hhhh .ssss
```

showing the pass count nn and address hhhh with optional symbol ssss. Registers are not displayed if -G or -U is in effect until the pass count reaches 1. Execution can be aborted during the pass trace with rabout.

Examples:

```
P100,ff
P.BDOS
P@ICALL+30,#20
-P .CRLF
```

2.11 / READ CODE/SYMBOLS

- (1) R
- (2) Rd

The I command sets up code and symbol files for subsequent loading with the R command. Form (1) reads optional code and optional symbols in preparation for program test, (2) is similar, but loads code and/or symbols with the bias valued. The sequence:

```
I x.y  
R
```

Sets up machine code file x.y (y is usually COM), and reads machine code to the transient area. If y is HEX, the file must be in Intel "hex" format. The sequence:

```
I x.y u.v  
R
```

also reads the symbol file u.v (u is usually the same as x, and v is normally SYM). The form:

```
I * u.v  
R
```

skips the machine code load, and reads only the symbol file.

When a symbol file is specified, the response

```
SYMBOLS
```

shows the start of the symbol file read operation. Thus, a "?" error before the SYMBOL message indicates a machine code read error, while "?" following the SYMBOL message shows a symbol file read error.

Examples:

```
I COPY.COM  
R  
I SORT.HEX SORT.SYM  
R  
I merge.com merge.sym  
R1000  
I * test.sym  
R-#256
```

2.12 / SET MEMORY

- (1) Ss
- (2) SWS

Form (1) sets memory locations in 8-bit format, (2) sets memory in 16-bit "word" format. In either case, each address is displayed, along with the current content. If a null line is entered, no change is made, and the next address is prompted. If a value is typed, then the data is changed and the next address is prompted. Input terminates with either invalid input, or a single "." from the console. Long ASCII input is entered with form (1) by typing a leading quote (") followed by graphic characters, terminated by a carriage return.

The examples show underlined console input:

```
S100
0100 C3 34
0101 24 #254
0102 CF
0103 4B "Ascii
0108 6E =X+5
0109 D4 .
SW.X+#30
2300 006D 44F
2302 4F32 @GAMMA
2304 33E2
2306 FF11 0+.X+=I-#20
2308 348F .
```

2.13 / TRACE MODE

- (1) Tn
- (2) T
- (3) Tn,c
- (4) T,c
- (5) -T ...
- (6) TW ...
- (7) -TW ...

Form (1) traces n program steps, showing the CPU state at each step, while (2) traces one step. Form (3) is used with ZSID utilities, and "calls" the utility function c at each trace step. Form (4) is similar to (3), but traces only one step. Form (5) parallels (1) to (4), but disables symbols.

Form (6) parallels (1) to (4), but performs "trace without call" showing only local execution. Form (7) is similar to (6) with symbols disabled.

Examples:

```
T100
T#30,.COLLECT
-TW=I,3E03
```

2.14 / UNTRACE MODE

- (1) U ...
- (2) -U
- (3) UW ...
- (4) -UW ...

U performs the same function as T, except the register state is not displayed. Forms (2) and (4), however, disable intermediate pass point trace (see P). U and T both run fully monitored, with automatic breaks at each instruction.

Execution can be aborted with rubout.

Examples:

```
Ufffff
U#10000,.COLLECT
UW=GAMMA,.COLLECT
```

2.15 / EXAMINE CPU STATE

- (1) X
- (2) Xf
- (3) Xr

Form (1) displays the CPU state in the format:

```
f A=a B=b D=d H=h S=s P=p i s
```

where f is the "flag state," a is the Z80 accumulator content, b is the 16-bit BC register pair value, d is the DE value, h is the HL value, s is the SP value, p is the PC value, i is the decoded instruction at p, and s is symbolic information. The flag are represented by dashes ("-") when false, and their letters when true:

Carry Zero Minus Even parity Interdigit carry

Form (2) allows flag state change, where f is one of C,Z,M,E, or I. The current state is displayed (either "-" or the letter). Enter the value 1 for true, 0 for false, or null for no change. Form (3) allows register state change, where r is one of A, B, D, H, S, or P. Symbol information is given at s when i references an address, including LDAX and STAX. The form "=mm" is printed for memory referencing instructions (e.g., INR M, ADD M), where mm is the memory value before execution.

Examples with operator input underlined:

```
XM
M O
XB
3E04 3EFF
XP
446E .CRLF+10
```

3.1 / ZSID UTILITIES

Utilities execute with ZSID to provide additional debugging facilities.

A utility is loaded initially by typing:

```
ZSID x.UTL
```

where x is the utility name. Upon loading, the utility is setup for execution with ZSID, and responds with:

```
.INITIAL = iiii
.COLLECT = cccc
.DISPLAY = dddd
```

where iiii, cccc, and dddd are three absolute address entries to the utility for (re)initializing, collecting debug data, and displaying collected information, respectively. The ZSID symbol table contains these three entry names. A utility is reinitialized by typing:

```
Ciiii or C.INITIAL
```

The display information is obtained by typing:

```
Cdddd or C.DISPLAY
```

while data collection occurs during monitored execution using the T or U commands, where the second argument gives the collection address.

Examples are:

```
Uffff,.collect
U#1000,3403
TW1000,.COLLECT
UW@GAMMA,.COLLECT
```

Pass points may be set during data collection to stop the monitoring at the end of program areas under test. The actual initialization, collection, and display functions depend upon the particular ZSID utility.

3.2 / THE HIST UTILITY

The HIST utility creates a histogram of program execution between two locations given during initialization. Program addresses are monitored during U or T mode execution, with summary data displayed at any time. Upon startup or reinitialization, HIST prompts with:

```
TYPE HISTOGRAM BOUNDS:
```

Respond with:

```
aaaa,bbbb
```

for a histogram between locations aaaa and bbbb, inclusive. Collect data in U or T mode, then display results. Output is scaled to the maximum collected value, accumulating until reinitialization.

An example:

```
ZSID HIST.UTL
TYPE HISTOGRAM BOUNDS 100,A00
.INITIAL = 3E03
.COLLECT = 3E06
.DISPLAY = 3E09
#I SORT.COM SORT.SYM
#R
SYMBOLS
#UFF,.COLLECT
(register display and break)
#C.DISPLAY
(histogram display)
U1000,.COLLECT
(display and eventual break)
C.DISPLAY
(updated histogram display)
#C.INITIAL
(histogram bounds reset)
```

3.3 / THE TRACE UTILITY

The TRACE utility provides a dynamic backtrace of up to 256 instructions which ended at the current break address. Instruction address collection occurs only in U or T mode. Pass points can be active, however, during the data collection, and will halt execution when the pass count becomes 1. Initialization clears the accumulated instructions, collection records the instruction address in a wraparound buffer, and display prints the backtrace in decoded mnemonic form with symbol references and labels when they occur. If "-A" is in effect, only instruction addresses are given. In this case, TRACE is loaded by typing:

```
ZSID
#-A
#I TRACE.UTL
#R
ADDRESSES ONLY
...
```

An example of normal operation:

```
ZSID TRACE.UTL
READY FOR SYMBOLIC BACKTRACE
#I MERGE.COM MERGE.SYM
#R
#UFFF,.COLLECT
(register display, wait, break)
#C.DISPLAY
(symbolic backtrace appears)
...
```

4.1 / IMPLEMENTATION NOTES

The ZSID program operates in about 10K bytes, and self-relocates directly below the BDOS (overlying the CCP area). The ZSID symbol table fills downward from the base of ZSID. As the table fills, the BDOS jump address is altered to reflect the reduced free space. Programs which "size" memory using the BDOS jump address should not be started until all symbols are loaded.

The "-A" command increases the free space by about 4K bytes. Any existing symbol information must be reloaded after issuing the command.

Programs will trace up to the BDOS where tracing is discontinued until control returns to the calling program.

ROM subroutine tracing is discontinued when ROM is entered through a call or jump and resumed upon return to the calling program in RAM.

Use rubout to abort programs running fully monitored in T or U mode, and an externally provided restart (RST 7) when running unmonitored with G.

5.1 / Z80 MNEMONICS

The Z80 mnemonics which follow (reproduced with permission from Zilog Corporation), can be entered directly in assembly mode (see A), and are produced by ZSID in list mode (see L). Data fields can consist of symbolic expressions. Given that "A100" has been typed, and that the symbols X, Y, and Z exist, the following is valid input:

```
LD  A,B
LD  A,OFF
LD  B,#255
LD  (HL),'x'
LD  HL,'ab'
JP  100
CALL .X
JP  Z,@Y
LD  HL,@X+=Z
JP  .X/Y+5
```

Notable differences between MAC and the ZSID "A" command are that no pseudo operations are allowed, operands are ZSID symbolic expressions*, labels cannot be inserted, and register references must be names, not numbers.

*In particular, note that

```
LD HL,'ab'
```

fills H with 'a' and L with 'b' due to the nature of ZSID expressions, which is counter to the MAC convention.

The Z80 is a Federally registered trademark of Zilog Corporation.

5.2 / Z80-CPU INSTRUCTION SET

OBJ CODE	SOURCE STATEMENT	OPERATION
8E DD8E05 FD8E05 8F 88 89 8A 8B 8C 8D CE20	ADC A,(HL) ADC A,(IX+d) ADC A,(IY+d) ADC A,A ADC A,B ADC A,C ADC A,D ADC A,E ADC A,H ADC A,L ADC A,n	Add with Carry Operand to Acc.
ED4A ED5A ED6A ED7A	ADC HL,BC ADC HL,DE ADC HL,HL ADC HL,SP	Add with Carry Reg Pair to HL
86 DD8605 FD8605 87 80 81 82 83 84 85 C620	ADD A,(HL) ADD A,(IX+d) ADD A,(IY+d) ADD A,A ADD A,B ADD A,C ADD A,D ADD A,E ADD A,H ADD A,L ADD A,n	Add Operand to Acc.
09 19 29 39	ADD HL,BC ADD HL,DE ADD HL,HL ADD HL,SP	Add Reg. Pair to HL
DD09 DD19 DD29 DD39	ADD IX,BC ADD IX,DE ADD IX,IX ADD IX,SP	Add Reg. Pair to IX
FD09 FD19 FD29 FD39	ADD IY,BC ADD IY,DE ADD IY,IY ADD IY,SP	Add Reg. Pair to Iy
A6 DDA605 FDA605 A7 A0 A1 A2 A3 A4 A5 E620	AND (HL) AND (IX+d) AND (IY+d) AND A AND B AND C AND D AND E AND H AND L AND n	Logical 'AND' of Operand and Acc.
CB46 DDCB0546 FDCB0546 CB47 CB40 CB41 CB42 CB43 CB44	BIT 0,(HL) BIT 0,(IX+d) BIT 0,(IY+d) BIT 0,A BIT 0,B BIT 0,C BIT 0,D BIT 0,E BIT 0,H	Test Bit b of Location or Reg.

CB45	BIT 0,L
CB4E	BIT 1,(HL)
DDCB054E	BIT 1,(IX+d)
FDCB054E	BIT 1,(IY+d)
CB4F	BIT 1,A
CB48	BIT 1,B
CB49	BIT 1,C
CB4A	BIT 1,D
CB4B	BIT 1,E
CB4C	BIT 1,H
CB4D	BIT 1,L
CB56	BIT 2,(HL)
DDCB0556	BIT 2,(IX+d)
FDCB0556	BIT 2,(IY+d)
CB57	BIT 2,A
CB50	BIT 2,B
CB51	BIT 2,C
CB52	BIT 2,D
CB53	BIT 2,E
CB54	BIT 2,H
CB55	BIT 2,L
CB5E	BIT 3,(HL)
DDCB055E	BIT 3,(IX+d)
FDCB055E	BIT 3,(IY+d)
CB5F	BIT 3,A
CB58	BIT 3,B
CB59	BIT 3,C
CB5A	BIT 3,D
CB5B	BIT 3,E
CB5C	BIT 3,H
CB5D	BIT 3,L
CB66	BIT 4,(HL)
DDCB0566	BIT 4,(IX+d)
FDCB0566	BIT 4,(IY+d)
CB67	BIT 4,A
CB60	BIT 4,B
CB61	BIT 4,C
CB62	BIT 4,D
CB63	BIT 4,E
CB64	BIT 4,H
CB65	BIT 4,L
CB6E	BIT 5,(HL)
DDCB056E	BIT 5,(IX+d)
FDCB056E	BIT 5,(IY+d)
CB6F	BIT 5,A
CB68	BIT 5,B
CB69	BIT 5,C
CB6A	BIT 5,D
CB6B	BIT 5,E
CB6C	BIT 5,H
CB6D	BIT 5,L
CB76	BIT 6,(HL)
DDCB0576	BIT 6,(IX+d)
FDCB0576	BIT 6,(IY+d)
CB77	BIT 6,A
CB70	BIT 6,B
CB71	BIT 6,C
CB72	BIT 6,D
CB73	BIT 6,E
CB74	BIT 6,H
CB75	BIT 6,L
CB7E	BIT 7,(HL)
DDCB057E	BIT 7,(IX+d)
FDCB057E	BIT 7,(IY+d)
CB7F	BIT 7,A
CB78	BIT 7,B
CB79	BIT 7,C
CB7A	BIT 7,D
CB7B	BIT 7,E
CB7C	BIT 7,H

CB7D	BIT 7,L	
DC8405	CALL C,nn	Call Subroutine at
FC8405	CALL M,nn	Location nn if Condition True
D48405	CALL NC,nn	
C48405	CALL NZ,nn	
F48405	CALL P,nn	
EC8405	CALL PE,nn	
E48405	CALL PO,nn	
CC8405	CALL Z,nn	
CD8405	CALL nn	Unconditional Call to Subroutine at nn
3F	CCF	Complement Carry Flag
BE	CP (HL)	Compare Operand with Acc.
DDBE05	CP (IX+d)	
FDBE05	CP (IY+d)	
BF	CP A	
B8	CP B	
B9	CP C	
BA	CP D	
BB	CP E	
BC	CP H	
BD	CP L	
FE20	CP n	
EDA9	CPD	Compare Location (HL) and Acc. Decrement HL and BC
EDB9	CPDR	Compare Location (HL) and Acc. Decrement HL and BC. Repeat until BC = 0
EDA1	CPI	Compare Location (HL) and Acc. Increment HL and Decrement BC
EDB1	CPIR	Compare Location (HL) and Acc. Increment HL, Decrement BC Repeat until BC = 0
2F	CPL	Complement Acc. (1's Comp).
27	DAA	Decimal Adjust Acc
35	DEC (HL)	Decrement Operand
DD3505	DEC (IX+d)	
FD3505	DEC (IY+d)	
3D	DEC A	
05	DEC B	
0B	DEC BC	
0D	DEC C	
15	DEC D	
1B	DEC DE	
1D	DEC E	
25	DEC H	
2B	DEC HL	
DD2B	DEC IX	
FD2B	DEC IY	
2D	DEC L	
3B	DEC SP	
F3	DI	Disable Interrupts
102E	DJNZ e	Decrement B and Jump Relative if B=0
FB	EI	Enable Interrupts

E3	EX (SP),HL	Exchange Location and (SP)
DDE3	EX (SP),IX	
FDE3	EX (SP),IY	
08	EX AF,AF'	Exchange the Contents of AF and AF'
EB	EX DE,HL	Exchange the Contents of DE and HL
D9	EXX	Exchange the Contents of BC, DE, HL with Contents of BC', DE', HL' Respectively
76	HALT	HALT (Wait for Interrupt or Reset)
ED46	IM 0	Set Interrupt Mode
ED56	IM 1	
ED5E	IM 2	
ED78	IN A,(C)	Load Reg. with Input from Device (C)
ED40	IN B,(C)	
ED48	IN C,(C)	
ED50	IN D,(C)	
ED58	IN E,(C)	
ED60	IN H,(C)	
ED68	IN L,(C)	
34	INC (HL)	Increment Operand
DD3405	INC (IX+d)	
FD3405	INC (IY+d)	
3C	INC A	
04	INC B	
03	INC BC	
0C	INC C	
14	INC D	
13	INC DE	
1C	INC E	
24	INC H	
23	INC HL	
DD23	INC IX	
FD23	INC IY	
2C	INC L	
33	INC SP	
DB20	IN A,(n)	Load Acc. with Input from Device n
EDAA	IND	Load Location (HL) with Input from Port (C), Decrement HL and B
EDBA	INDR	Load Location (HL) with Input from Port (C), Decrement HL and Decrement B, Repeat until B=0
EDA2	INI	Load Location (HL) with Input from Port (C), Increment HL and Decrement B
EDB2	INIR	Load Location (HL) with Input from Port (C), Increment HL and Decrement B, Repeat until B=0
C38405	JP nn	Unconditional Jump to Location
E9	JP (HL)	
DDE9	JP (IX)	
FDE9	JP (IY)	
DA8405	JP C,nn	Jump to Location if Condition True
FA8405	JP M,nn	
D28405	JP NC,nn	
C28405	JP NZ,nn	

F28405	JP P,nn	
EA8405	JP PE,nn	
E28405	JP PO,nn	
CA8405	JP Z,nn	
382E	JR C,e	Jump Relative to PC+e if
302E	JR NC,e	Condition True
202E	JR NZ,e	
282E	JR Z,e	
182E	JR e	Unconditional Jump Relative to PC+e
02	LD (BC),A	Load Source to Destination
12	LD (DE),A	
77	LD (HL),A	
70	LD (HL),B	
71	LD (HL),C	
72	LD (HL),D	
73	LD (HL),E	
74	LD (HL),H	
75	LD (HL),L	
3620	LD (HL),n	
DD7705	LD (IX+d),A	
DD7005	LD (IX+d),B	
DD7105	LD (IX+d),C	
DD7205	LD (IX+d),D	
DD7305	LD (IX+d),E	
DD7405	LD (IX+d),H	
DD7505	LD (IX+d),L	
DD360520	LD (IX+d),n	
FD7705	LD (IY+d),A	
FD7005	LD (IY+d),B	
FD7105	LD (IY+d),C	
FD7205	LD (IY+d),D	
FD7305	LD (IY+d),E	
FD7405	LD (IY+d),H	
FD7505	LD (IY+d),L	
FD360520	LD (IY+d),n	
328405	LD (nn),A	
ED438405	LD (nn),BC	
ED538405	LD (nn),DE	
228405	LD (nn),HL	
DD228405	LD (nn),IX	
FD228405	LD (nn),IY	
ED738405	LD (nn),SP	
0A	LD A,(BC)	
1A	LD A,(DE)	
7E	LD A,(HL)	
DD7E05	LD A,(IX+d)	
FD7E05	LD A,(IY+d)	
3A8405	LD A,(nn)	
7F	LD A,A	
78	LD A,B	
79	LD A,C	
7A	LD A,D	
7B	LD A,E	
7C	LD A,H	
ED57	LD A,I	
7D	LD A,L	
3E20	LD A,n	
ED5F	LD A,R	
46	LD B,(HL)	
DD4605	LD B,(IX+d)	
FD4605	LD B,(IY+d)	
47	LD B,A	
40	LD B,B	
41	LD B,C	
42	LD B,D	
43	LD B,E	
44	LD B,H	

45	LD B,L
0620	LD B,n
ED4B8405	LD BC,(nn)
018405	LD BC,nn
4E	LD C,(HL)
DD4E05	LD C,(IX+d)
FD4E05	LD C,(IY+d)
4F	LD C,A
48	LD C,B
49	LD C,C
4A	LD C,D
4B	LD C,E
4C	LD C,H
4D	LD C,L
0E20	LD C,n
56	LD D,(HL)
DD5605	LD D,(IX+d)
FD5605	LD D,(IY+d)
57	LD D,A
50	LD D,B
51	LD D,C
52	LD D,D
53	LD D,E
54	LD D,H
55	LD D,L
1620	LD D,n
ED5B8405	LD DE,(nn)
118405	LD DE,nn
5E	LD E,(HL)
DD5E05	LD E,(IX+d)
FD5E05	LD E,(IY+d)
5F	LD E,A
58	LD E,B
59	LD E,C
5A	LD E,D
5B	LD E,E
5C	LD E,H
5D	LD E,L
1E20	LD E,n
66	LD H,(HL)
DD6605	LD H,(IX+d)
FD6605	LD H,(IY+d)
67	LD H,A
60	LD H,B
61	LD H,C
62	LD H,D
63	LD H,E
64	LD H,H
65	LD H,L
2620	LD H,n
2A8405	LD HL,(nn)
218405	LD HL,nn
ED47	LD I,A
DD2A8405	LD IX,(nn)
DD218405	LD IX,nn
FD2A8405	LD IY,(nn)
FD218405	LD IY,nn
6E	LD L,(HL)
DD6E05	LD L,(IX+d)
FD6E05	LD L,(IY+d)
6F	LD L,A
68	LD L,B
69	LD L,C
6A	LD L,D
6B	LD L,E
6C	LD L,H
6D	LD L,L
2E20	LD L,n
ED4F	LD R,A
ED7B8405	LD SP,(nn)

F9	LD SP,HL	
DDF9	LD SP,IX	
FDF9	LD SP,IY	
318405	LD SP,nn	
EDA8	LDD	Load Location (DE) with Location (HL) Decrement DE, HL and BC
EDB8	LDDR	Load Location (DE) with Location (HL) Repeat until BC = 0
EDA0	LDI	Load Location (DE) with Location (HL) Increment DE, HL, Decrement BC
EDB0	LDIR	Load Location (DE) with Location (HL) Increment DE, HL, Decrement BC and Repeat until BC = 0
ED44	NEG	Negate Acc. (2's Complement)
00	NOP	No Operation
B6	OR (HL)	Logical "OR" of Operand and Acc.
DDB605	OR (IX+d)	
FDB605	OR (IY+d)	
B7	OR A	
B0	OR B	
B1	OR C	
B2	OR D	
B3	OR E	
B4	OR H	
B5	OR L	
F620	OR n	
EDBB	OTDR	Load Output Port (C) with Location (HL) Decrement HL and B, Repeat until B=0
EDB3	OTIR	Load Output Port (C) with Location (HL), Increment HL, Decrement B, Repeat until B=0
ED79	OUT (C),A	Load Output Port (C) with Reg.
ED41	OUT (C),B	
ED49	OUT (C),C	
ED51	OUT (C),D	
ED59	OUT (C),E	
ED61	OUT (C),H	
ED69	OUT (C),L	
D320	OUT (n),A	Load Output Port (n) with Acc.
EDAB	OUTD	Load Output Port (C) with Location (HL). Decrement HL and B
EDA3	OUTI	Load Output Port (C) with Location (HL). Increment HL and Decrement B
F1	POP AF	Load Destination with Top of Stack
C1	POP BC	
D1	POP DE	
E1	POP HL	
DDE1	POP IX	
FDE1	POP IY	

F5	PUSH AF	Load Source to Stack
C5	PUSH BC	
D5	PUSH DE	
E5	PUSH HL	
DDE5	PUSH IX	
FDE5	PUSH IY	
CB86	RES 0, (HL)	Reset Bit b of Operand
DDCB0586	RES 0, (IX+d)	
FDCB0586	RES 0, (IY+d)	
CB87	RES 0, A	
CB80	RES 0, B	
CB81	RES 0, C	
CB82	RES 0, D	
CB83	RES 0, E	
CB84	RES 0, H	
CB85	RES 0, L	
CB8E	RES 1, (HL)	
DDCB058E	RES 1, (IX+d)	
FDCB058E	RES 1, (IY+d)	
CB8F	RES 1, A	
CB88	RES 1, B	
CB89	RES 1, C	
CB8A	RES 1, D	
CB8B	RES 1, E	
CB8C	RES 1, H	
CB8D	RES 1, L	
CB96	RES 2, (HL)	
DDCB0596	RES 2, (IX+d)	
FDCB0596	RES 2, (IY+d)	
CB97	RES 2, A	
CB90	RES 2, B	
CB91	RES 2, C	
CB92	RES 2, D	
CB93	RES 2, E	
CB94	RES 2, H	
CB95	RES 2, L	
CB9E	RES 3, (HL)	
DDCB059E	RES 3, (IX+d)	
FDCB059E	RES 3, (IY+d)	
CB9F	RES 3, A	
CB98	RES 3, B	
CB99	RES 3, C	
CB9A	RES 3, D	
CB9B	RES 3, E	
CB9C	RES 3, H	
CB9D	RES 3, L	
CBA6	RES 4, (HL)	
DDCB05A6	RES 4, (IX+d)	
FDCB05A6	RES 4, (IY+d)	
CBA7	RES 4, A	
CBA0	RES 4, B	
CBA1	RES 4, C	
CBA2	RES 4, D	
CBA3	RES 4, E	
CBA4	RES 4, H	
CBA5	RES 4, L	
CBAE	RES 5, (HL)	
DDCB05AE	RES 5, (IX+d)	
FDCB05AE	RES 5, (IY+d)	
CBAF	RES 5, A	
CBA8	RES 5, B	
CBA9	RES 5, C	
CBAA	RES 5, D	
CBAB	RES 5, E	
CBAC	RES 5, H	
CBAD	RES 5, L	
CBB6	RES 6, (HL)	
DDCB05B6	RES 6, (IX+d)	
FDCB05B6	RES 6, (IY+d)	

CBB7	RES 6,A	
CBB0	RES 6,B	
CBB1	RES 6,C	
CBB2	RES 6,D	
CBB3	RES 6,E	
CBB4	RES 6,H	
CBB5	RES 6,L	
CBBE	RES 7,(HL)	
DDCB05BE	RES 7,(IX+d)	
FDCB05BE	RES 7,(IY+d)	
CBBF	RES 7,A	
CBB8	RES 7,B	
CBB9	RES 7,C	
CBBA	RES 7,D	
CBBB	RES 7,E	
CBBC	RES 7,H	
CBBD	RES 7,L	
C9	RET	Return from Subroutine
D8	RET C	Return from Subroutine
F8	RET M	if Condition True
D0	RET NC	
C0	RET NZ	
F0	RET P	
E8	RET PE	
E0	RET PO	
C8	RET Z	
ED4D	RETI	Return from Interrupt
ED45	RETN	Return from Non Maskable Interrupt
CB16	RL (HL)	Rotate Left Through Carry
DDCB0516	RL (IX+d)	
FDCB0516	RL (IY+d)	
CB17	RL A	
CB10	RL B	
CB11	RL C	
CB12	RL D	
CB13	RL E	
CB14	RL H	
CB15	RL L	
17	RLA	Rotate Left Acc. Through Carry
CB06	RLC (HL)	Rotate Left Circular
DDCB0506	RLC (IX+d)	
FDCB0506	RLC (IY+d)	
CB07	RLC A	
CB00	RLC B	
CB01	RLC C	
CB02	RLC D	
CB03	RLC E	
CB04	RLC H	
CB05	RLC L	
07	RLCA	Rotate Left Circular Acc.
ED6F	RLD	Rotate Digit Left and Right between Acc. and Location (HL)
CB1E	RR (HL)	Rotate Right Through Carry
DDCB051E	RR (IX+d)	
FDCB051E	RR (IY+d)	
CB1F	RR A	
CB18	RR B	
CB19	RR C	
CB1A	RR D	

CB1B	RR E	
CB1C	RR H	
CB1D	RR L	
1F	RRA	Rotate Right Acc. Through Carry
CB0E	RRC (HL)	Rotate Right Circular
DDCB050E	RRC (IX+d)	
FDCB050E	RRC (IY+d)	
CB0F	RRC A	
CB08	RRC B	
CB09	RRC C	
CB0A	RRC D	
CB0B	RRC E	
CB0C	RRC H	
CB0D	RRC L	
0F	RRCA	Rotate Right Circular Acc.
ED67	RRD	Rotate Digit Right and Left Between Acc. and Location (HL)
C7	RST 00H	Restart to Location
CF	RST 08H	
D7	RST 10H	
DF	RST 18H	
E7	RST 20H	
EF	RST 28H	
F7	RST 30H	
FF	RST 38H	
DE20	SBC A,n	Subtract Operand from Acc. with Carry
9E	SBC A,(HL)	
DD9E05	SBC A,(IX+d)	
FD9E05	SBC A,(IY+d)	
9F	SBC A,A	
98	SBC A,B	
99	SBC A,C	
9A	SBC A,D	
9B	SBC A,E	
9C	SBC A,H	
9D	SBC A,L	
ED42	SBC HL,BC	
ED52	SBC HL,DE	
ED62	SBC HL,HL	
ED72	SBC HL,SP	
37	SCF	Set Carry Flag (C=1)
CBC6	SET 0,(HL)	Set Bit b of Location
DDCB05C6	SET 0,(IX+d)	
FDCB05C6	SET 0,(IY+d)	
CBC7	SET 0,A	
CBC0	SET 0,B	
CBC1	SET 0,C	
CBC2	SET 0,D	
CBC3	SET 0,E	
CBC4	SET 0,H	
CBC5	SET 0,L	
CBCE	SET 1,(HL)	
DDCB05CE	SET 1,(IX+d)	
FDCB05CE	SET 1,(IY+d)	
CBCF	SET 1,A	
CBC8	SET 1,B	
CBC9	SET 1,C	
CBCA	SET 1,D	
CBCB	SET 1,E	
CBCD	SET 1,H	
CBCD	SET 1,L	
CBD6	SET 2,(HL)	

DDCB05D6 SET 2, (IX+d)
 FDCB05D6 SET 2, (IY+d)
 CBD7 SET 2, A
 CBD0 SET 2, B
 CBD1 SET 2, C
 CBD2 SET 2, D
 CBD3 SET 2, E
 CBD4 SET 2, H
 CBD5 SET 2, L
 CBD8 SET 3, B
 CBDE SET 3, (HL)
 DDCB05DE SET 3, (IX+d)
 FDCB05DE SET 3, (IY+d)
 CBDF SET 3, A
 CBD9 SET 3, C
 CBDA SET 3, D
 CBDB SET 3, E
 CBDC SET 3, H
 CBDD SET 3, L
 CBE6 SET 4, (HL)
 DDCB05E6 SET 4, (IX+d)
 FDCB05E6 SET 4, (IY+d)
 CBE7 SET 4, A
 CBE0 SET 4, B
 CBE1 SET 4, C
 CBE2 SET 4, D
 CBE3 SET 4, E
 CBE4 SET 4, H
 CBE5 SET 4, L
 CBEE SET 5, (HL)
 DDCB05EE SET 5, (IX+d)
 FDCB05EE SET 5, (IY+d)
 CBEF SET 5, A
 CBE8 SET 5, B
 CBE9 SET 5, C
 CBEA SET 5, D
 CBEB SET 5, E
 CBEC SET 5, H
 CBED SET 5, L
 CBF6 SET 6, (HL)
 DDCB05F6 SET 6, (IX+d)
 FDCB05F6 SET 6, (IY+d)
 CBF7 SET 6, A
 CBF0 SET 6, B
 CBF1 SET 6, C
 CBF2 SET 6, D
 CBF3 SET 6, E
 CBF4 SET 6, H
 CBF5 SET 6, L
 CBF6 SET 6, (HL)
 DDCB05FE SET 7, (IX+d)
 FDCB05FE SET 7, (IY+d)
 CBFF SET 7, A
 CBF8 SET 7, B
 CBF9 SET 7, C
 CBFA SET 7, D
 CBFB SET 7, E
 CBFC SET 7, H
 Cbfd SET 7, L

 CB26 SLA (HL)
 DDCB0526 SLA (IX+d)
 FDCB0526 SLA (IY+d)
 CB27 SLA A
 CB20 SLA B
 CB21 SLA C
 CB22 SLA D
 CB23 SLA E
 CB24 SLA H
 CB25 SLA L

Shift operand Left Arithmetic

CB2E	SRA	(HL)
DDCB052E	SRA	(IX+d)
FDCB052E	SRA	(IY+d)
CB2F	SRA	A
CB28	SRA	B
CB29	SRA	C
CB2A	SRA	D
CB2B	SRA	E
CB2C	SRA	H
CB2D	SRA	L

CB3E	SRL	(HL)
DDCB053E	SRL	(IX+d)
FDCB053E	SRL	(IY+d)
CB3F	SRL	A
CB38	SRL	B
CB39	SRL	C
CB3A	SRL	D
CB3B	SRL	E
CB3C	SRL	H
CB3D	SRL	L

Shift Operand Right Logical

96	SUB	(HL)
DD9605	SUB	(IX+d)
FD9605	SUB	(IY+d)
97	SUB	A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
D620	SUB	n

Subtract Operand from Acc.

AE	XOR	(HL)
DDAE05	XOR	(IX+d)
FDAE05	XOR	(IY+d)
AF	XOR	A
A8	XOR	B
A9	XOR	C
AA	XOR	D
AB	XOR	E
AC	XOR	H
AD	XOR	L
EE20	XOR	n

Exclusive "OR" Operand and Acc.

Example Values

nn	EQU	584H
d	EQU	5
n	EQU	20H
e		30H

Note that ZSID accepts an address instead of a byte value in the jmp relative commands.