Z-80 ASSEMBLY LANGUAGE UNDER TurboDOS

Fourth Edition

R. Roger Breton

Z-80 Assembly Language Programming under TurboDOS
Fourth Edition

Table of Chapters

Table of Contents

Abbreviations, Terms, and Symbols

(BC)	A memory indirect address: the byte or word in memory whose address is in the BC register pair.
(C)	A port-indirect address: the I/O port whose address is in the C register.
(DE)	A memory-indirect address: the byte or word in memory whose address is in the DE register pair.
(HL)	A memory-indirect address: the byte or word in memory whose address is in the HL register pair.
(IX)	A memory-indirect address: the byte or word in memory whose address is in the IX index register.
(IX+d)	An indexed memory-indirect address: the byte in memory whose address is the sum of in index register IX plus the value of the sign-extended displacement byte.
(IY)	A memory-indirect address: the byte or word in memory whose address is in the IY index register.
(IY+d)	An indexed memory-indirect address: the byte in memory whose address is the sum of in index register IY plus the value of the sign-extended displacement byte.
(Ii+d)	A non-specific indexed memory-indirect operand. The byte in memory whose address is the sum of a non-specific index register (either IX or IY) plus the value of the sign-extended displacement byte.
(SP)	A memory-indirect address: the byte or word in memory whose address is in the stack pointer register.
(port)	An 8-bit immediate port-direct address value. The address of the I/O port from or to which data is to be transferred.
(reg16)	A non-specific 16-bit memory-indirect operand. The word in memory whose low-order byte is “reg16” and whose high-order byte is “reg16 + 1”, where “reg16” is a register pair or 16-bit register, usually HL, IX or IY.
(regpr)	A non-specific memory-indirect operand. The byte in memory whose address is in a register pair, usually HL, DE or BC.
*	A symbol representing multiplication.
+	A symbol representing addition.
–	A symbol representing subtraction.
-high	A suffix to a 16-bit operand indicating that only the high-order byte of the operand is affected or significant.
-low	A suffix to a 16-bit operand indicating that only the low-order byte of the operand is affected or significant.
<	A symbol representing the expression: “is less than.”
←	A symbol representing the expression: “is made equal to.”
↔	A symbol representing the expression: “is exchanged with.”
#	A symbol representing the expression: “is not equal to.”
=	A symbol representing the expression: “is equal to.”
A	The accumulator (8-bit A register).
AF	The 16-bit processor status word, consisting of the 8-bit A and F registers taken together.
AF'	The alternative 16-bit processor status word.
AND	A symbol representing a bitwise logical AND.
B	The 8-bit B register.
BC	The 16-bit BC register pair, consisting of the 8-bit B and C registers taken together.
BC'	The alternative 16-bit BC register pair.
BITSUM	A symbol representing the one-bit result of adding the bits of a bitstream together, ignoring any carry, the starting and stopping bits being those before and after the BITSUM. The expression “A[7 BITSUM 0]” is identical to the expression “A[7] + A[6] + A[5] + A[4] + A[3] + A[2] + A[1] + A[0]”, and is 0 if an even number of bits are set (zero itself is even), or 1 if an odd number are set.
C	The 8-bit C register.

C	A condition statement meaning “execute on carry: execute if the carry flag is set.”
CF	The carry flag, bit 0 of the flag register.
D	The 8-bit D register.
DE	The 16-bit DE register pair, consisting of the 8-bit D and E registers taken together.
DE'	The alternative 16-bit DE register pair.
E	The 8-bit E register.
F	The 8-bit flag or status register.
H	The 8-bit H register.
HF	The half-carry flag, bit 4 of the flag register.
HL	The 16-bit HL register pair, consisting of the 8-bit H and L registers taken together.
HL'	The alternative 16-bit HL register pair.
I	The 8-bit interrupt vector register.
IFF1	The interrupt status flip-flop “1”.
IFF2	The interrupt status flip-flop “2”.
IMFa	The interrupt mode flip-flop “a”.
IMFb	The interrupt mode flip-flop “b”.
IX	The 16-bit IX index register.
IY	The 16-bit IY index register.
Ii	A non-specific index register: either IX or IY.
L	The 8-bit L register.
M	A condition statement meaning “execute on minus (negative): execute if the sign flag is set.”
NC	A condition statement meaning “execute on not carry: execute if the carry flag is not set.”
NF	The subtraction flag, bit 1 of the flag register.
NOT	A symbol representing a bitwise logical NOT (one's compliment).
NZ	A condition statement meaning “execute on not zero: execute if the zero flag is not set.”
OR	A symbol representing a bitwise logical inclusive-OR.
ORSUM	A symbol representing the 1-bit result of executing an inclusive-OR on all bits of a bitstream, the starting and stopping bits being those before and after the ORSUM. The expression “A[7 ORSUM 0]” is identical to the expression “A[7] OR A[6] OR A[5] OR A[4] OR A[3] OR A[2] OR A[1] OR A[0]”, and is if none of the bits are set, or 1 if any of the bits are set.
P	A condition statement meaning “execute on plus (positive): execute if the sign flag is not set.”
PC	The 16-bit program counter register.
PE	A condition statement meaning “execute on parity even: execute if the parity/overflow flag is set.”
PF	The parity/overflow flag, bit 2 of the status register.
PO	A condition statement meaning “execute on parity odd: execute if the parity/overflow flag is not set.”
R	The 8-bit memory refresh register.
SF	The sign flag, bit 7 of the flag register.
SP	The 16-bit stack pointer register.
Z	A condition statement meaning “execute on zero: execute if the zero flag is set.”
ZF	The zero flag, bit 6 of the flag register.
ab	An address-displacement op-code byte.

addr	An immediate 16-bit address value.
ah	The high-order op-code byte of an address word.
al	The low-order op-code byte of an address word.
cond	A conditional program transfer condition statement, usually one of the following (only the first four are used in relative instructions): NZ, Z, NC, C, PE, PO, P and M.
d	An immediate-byte displacement: used in the expressions (Ii+d), (IX+d) and (IY+d).
db	A indexed-memory displacement op-code byte.
disp	An immediate 8-bit address displacement value, used in relative program transfer instructions. This value is sign-extended and added to the then-current contents of the program counter to determine the 16-bit address value. Because the byte containing the displacement value is the second byte of the instruction, an offset occurs, causing the displacement to be valid for +126 to –129 bytes from the beginning of the instruction. Furthermore, the byte stored in the instruction op-code is the displacement value less 2.
dest	A non-specific destination operand. An operand in which one of the parameters of an arithmetic or logical operation resides, and in which the result resides. This operand is usually altered by the operation.
ib	An 8-bit immediate value op-code byte.
ih	The high-order op-code byte of a 16-bit immediate value.
il	The low-order op-code byte of a 16-bit immediate value.
immed	An 8-bit immediate value.
imwrd	A 16-bit immediate value.
left	A non-specific operand, used where both the left and right operands are of equal status, as in the EX (exchange) instruction.
o[b]	A symbol representing the expression: “bit 'b' of operand 'o'.”
op	An op-code byte containing some variable portion.
pb	An 8-bit port address op-code byte.
reg	An 8-bit register operand, usually one of the following registers: A, B, C, D, E, H or L.
reg'	An 8-bit register operand, identical to “reg”. Used in the LD instruction as the second register operand.
reg16	A 16-bit register or register pair operand.
regpr	A 16-bit register or register pair operand, usually (but not always) one of the following: AF, BC, DE, HL or SP.
right	A non-specific operand, used where both the left and right operands are of equal status, as in the EX (exchange) instruction.
source	A non-specific source operand. An operand in which one of the parameters of an arithmetic or logical operation resides. This operand is usually left unchanged by the operation.
temp	An internal 16-bit/8-bit pseudo-register for the processor's arithmetic/logic unit. This pseudo-register is used by virtually all operations to store transient values during multi-step operations, and contains the result of virtually all arithmetic and logical operations. It is not available to the user, and is only indicated where knowledge of its contents is fundamental to the understanding of a particular operation.
v	A 1-, 2-, or 3-bit variable portion of an op-code byte, usually representing the register coding bits.
v'	A 3-bit variable portion of an op-code byte. Identical to v in structure, and used in the LD instruction to represent the second register value.
vector	A non-specific operand containing a restart vector value of 00, 08, 10, 18, 20, 28, 30 or 38, which represents the low-order byte of the vector address (the high-order byte being 00).
w	A 1-bit variable portion of an op-code byte, representing the index register determination bit. If w is 0, then the op-code byte is DD and the index register is IX. If w is 1 then the op-code byte is FD and the index register is IY.
x	A 3-bit variable portion of an op-code byte, representing a specific bit operand.

y	A 2- or 3-bit variable portion of an op-code byte, usually representing the particular condition of conditional program transfer instructions.
z	A 3-bit portion of an op-code byte representing a reset vector.
~	A symbol used to indicate an instruction that is a Z-180 instruction, rather than a Z-80 instruction.

Microsoft Macro-80 Assembler

Description

Microsoft's Macro-80 assembler, also called M80, is a fully relocatable macro assembler written in 8080 mnemonics and intended for use with the CP/M operating system. Even with this limitation, it is still one of the best assemblers for use under TurboDOS.

The object files produced by Macro-80 are in Microsoft REL file format, and must be linked via a suitable linker, such as Microsoft's Link-80 linker or Software 2000's GEN linker, in order to produce an executable COM or SYS file. The structure of a REL file is such that a series of routines or subroutines containing global (public) entrypoints and external references may be linked into a homogenous whole at any given address, thus making them truly relocatable.

Version 3.** of Macro-80 is about 20K in size and assembles at the rate of about 1000 lines per minute.

Commands

The basic command structure of Macro-80 is as follows:

```
M80 objfile,prnfile=srcfile/s1/s2/s3.../sn

M80 objfile=srcfile/s1/s2/s3.../sn

M80 ,=srcfile/s1/s2/s3.../sn

M80 =srcfile/s1/s2/s3.../sn

M80
* command
* command
    * attention abort
```

Where:

“srcfile” = the source code file, usually of type MAC.

“prnfile” = the print/listing file of type PRN.

“objfile” = the relocatable object file of type REL.

“s1”, etc = optional switches

There are several interesting things to note about the command line. In the full command, “objfile,prnfile=srcfile/s1/s2/s3.../sn”, the object file is separated from the print file by a comma, and the print file from the source file by an equals. It stands to reason then that neither a comma nor an equals may be a part of any filename. This is in fact the case and TurboDOS, like CP/M, defines the comma and equals as reserved characters, like the period and colon.

What is not so obvious is that because the switches are separated by slashes, the slash may not be used in any filename, even though so allowed by TurboDOS! To do so produces an error.

Interpretation of the commands is as follows:

M80 objfile,prnfile=srcfile

Assemble “srcfile.MAC” into “objfile.REL”, producing “prnfile.PRN”. This is the standard command structure. A drive designation may be used with any of the filenames to allow flexibility, but all files must be on the current user area.

M80 objfile=srcfile

Assemble “srcfile.MAC” into “objfile.REL”. This is the command structure to use when the source and object files have different names or drives.

M80 ,=srcfile

Assemble nothing, but do a syntax check of “srcfile.MAC”. This command structure allows the checking of a source file for errors even though it is not yet ready for assembly. Typically, certain known errors (those caused by the file's state of unreadiness) would be ignored while other errors would be noted and corrected.

M80 =srcfile

Assemble “srcfile.MAC” into “srcfile.REL”. This is the standard shorthand command when the source and object files have the same names and drives.

It is interesting to note that because the comma and the equals are reserved characters, the last two command forms may be entered without whitespace:

```
M80,=srcfile
```

```
M80=srcfile
```

If the command “M80” is entered with no arguments or with illegal arguments, an interactive mode is entered and an asterisk prompt appears. Command lines may then be entered one after the other for as long as desired.

Under CP/M, the proper method of exiting the interactive mode was via a warmboot (control-C). As TurboDOS does not have a warmboot in the same manner, it is necessary to enter an attention-abort sequence (typically break/control-C or control-@/control-C). If the console in use disallows attention, then there is no exit from the interactive mode short of resetting the processor in use.

Switches

Switches are command parameters that allow certain optional operations to take place:

/C

The /C switch forces generation of a cross reference file of type CRF. This file is used by the Microsoft C-REF utility to produce a cross reference listing of all symbols.

See the .CREF assembler directive.

/H

The /H switch prints all PRN file listing addresses in hexadecimal (default). This switch is for the most part superfluous, as hexadecimal is the normal number base used.

See the /O switch.

/I

The /I switch forces assembly using the Intel 8080 mnemonics (default). This switch is for the most part superfluous as 8080 mnemonics are the norm for Macro-80.

Under TurboDOS, only the Z-80 processor and its clones may be used, the 8080 processor is not allowed, and the Z-80 mnemonic set is thus the instruction set of choice.

See the /Z switch and the .8080 and .Z80 assembler directives.

/L

The /L switch forces generation of a print file of type PRN.

The following two commands are identical in function:

```
M80 =srcfile/L
```

```
M80 srcfile,srcfile=srcfile
```

/M

The /M switch initializes all DS or DSEG bytes to 00h. If this switch is used, then any bytes allocated by the DS or DSEG pseudo-operation are initialized to 00h at assembly time. Otherwise, those bytes contain whatever was in memory at the time.

See the DS and DSEG pseudo-operations.

/O

The /O switch prints all listing addresses in octal.

See the /H switch.

/P

The /P switch allows an extra 256 (100h) bytes of stack space during assembly. This switch is used if a stack overflow error occurs during assembly.

Multiply switches may be used: /P/P allocates 512 bytes.

/R

The /R switch forces generation of an object file of type REL. This switch cancels the effect of a leading comma in the command line.

The following two commands are identical in operation:

```
M80 ,=srcfile/R

M80 =srcfile
```

/X

The /X switch suppresses listing of false conditionals.

See the .LFCOND, .SFCOND, and .TFCOND assembler directives.

/Z

The /Z switch causes assembly using Zilog Z-80 op-codes.

See the /I switch and the .8080 and .Z80 assembler directives.

Statement Format

The Macro-80 assembler is a somewhat loose column oriented assembler. Considerable tolerance is given in the size and positions of the columns; however, historical and standardization practices dictate two columns of 8, one column of either 16 or 24, and one column of undefined spaces or equivalent whitespace be used:

```
        8       8               16  or  24 spaces
0       1       2               4   or  5 tabs
label:  operand arguments       ;comments
```

This convention arises from the traditional tabbing of terminals at every 8 spaces, thus allowing columns at the 0, 1, 2, and 4 or 5 tab position. It is considered very poor programming form to modify this traditional structure.

The four columns are dedicated to the following structures:

Column 1: labels

Column 2: operands, pseudo-operands, and assembler directives

Column 3: arguments

Column 4 comments, preceded by a semicolon

In any given statement any given column or combination of columns may be omitted, with the exception that the arguments column is never used when the operands column is not (arguments of what?).

Whitespace

Whitespace, the space separating the columns, may be either spaces or tabs in any combination. All whitespace is ignored during operation, but is required to separate the columns unless another delimiter is used, such as a colon or semicolon.

Labels

Column 1 should be limited to labels or whitespace. A label is defined by being in column 1.

A label may be from 1 to 6 characters and must begin with a letter, period, or underline (never a number). A label may not be a reserved word: any of the assembler directives, pseudo-operations, Z-80 operands, or register names. When in column 1, a label always ends with a colon unless it is followed by the EQU pseudo-operation.

If a label in column 1 is followed by a double colon "::", it is a global or public entrypoint; otherwise, it is a local label, defined within the current module. See the ENTRY, GLOBAL, and PUBLIC pseudo-operations.

A label is usually used in column 3 as an argument of a JUMP or CALL operand. In this case, the JUMP or CALL is to the line where the label is defined (in column 1), or to the address whose value is the same as the label.

If the label in column 3 is followed by a double number sign "##", it is a reference to an external label; that is a label that is global in another module, and is not found locally.

Examples of various labels are:

```
FOO     EQU     12              ;Define FOO as the value 12
OPSYS   EQU     0000H           ;Define OPSYS as the address 0000H
START:                          ;Define local label START
        LD      A,FOO           ;Load accumulator with 12
        JR      DONE            ;Jump to label DONE
;
OUTSID::NOP                     ;Define global label OUTSID
        CALL    OTHER##         ;Call external label OTHER
;
DONE:                           ;Define local label DONE
        JP      OPSYS           ;Jump to address 0000H
        END
```

Operands

An operand in column 2 may be any of the following: an assembler directive, a pseudo-operation, or a Z-80 operand. Most but not all operands require one or more arguments in column 3.

Arguments

An argument or arguments in column 3 are often required to fulfill or complete the operand in column 2. An argument may be of the following types: a register, a memory location (either direct or indirect), an immediate numerical value, a character, a displacement, or a label. Not all operands accept all types of arguments.

Comments

A comment may be placed in column 4, preceded by a semicolon. Any such comment has no effect upon the final object code.

If fact, a semicolon by definition causes itself and anything following it on the same line to be ignored. A semicolon is often used at the very beginning of a line to allow a "comment line" to be inserted into the code. When used alone in the first position, it produces a "blank" line, useful for code legibility.

It is considered bad programming form to have truly blank lines in the code.

Comments may also be entered via the .COMMENT assembler directive.

Number Base Reference

The default number base is decimal: that is, all numerical values are assumed to be decimal unless special care is taken to indicate otherwise.

The .RADIX assembler directive allows the number base to be altered:

```
        .RADIX  2               ;Number base is binary
        .RADIX  8               ;Number base is octal
        .RADIX  10              ;Number base is decimal
        .RADIX  16              ;Number base is hexadecimal
```

Once the base has been changed via the .RADIX assembler directive, it remains at the new base unless changed again. Any number of base changes may be made in a given module.

In order to enter a number not in the current base, a special form of the number may be used:

0000B for Binary

```
        LD      A,10111010B     ;Load A with binary 10111010 (decimal 186)
```

0000D for Decimal

```
        LD      A,0123D         ;Load A with decimal 123
```

000O for Octal (last character is letter “O”, not number “0”)

```
        LD      A,377O          ;Load A with octal 377 (decimal 255)
```

000Q for Octal

```
        LD      A,376Q          ;Load A with octal 376 (decimal 254)
```

0000H for Hexadecimal

```
        LD      A,0ABH          ;Load A with hexadecimal AB (decimal 171)
```

X'0000' for Hexadecimal

```
        LD      A,X'0D2'        ;Load A with hexadecimal D2 (decimal 210)
```

Note that a number must always begin with a digit, never a letter: this injunction affect hexadecimal values:

```
        .RADIX  16              ;All values in hexadecimal
        LD      A,D             ;Load A with register D
        LD      A,0D            ;Load A with the value D (decimal 13)
        LD      A,0003D         ;Load A with decimal 3
```

Notice that the “0D” in the above example did not require an “H” suffix since the base radix was hexadecimal already.

Notice also that the leading zeroes in “0003D” were NOT ignored, but caused the value to be treated as decimal 3 rather than hexadecimal 3D. Extreme care must be taken when dealing with decimal numbers in this manner. Earlier versions of Macro-80 had subtle bugs in this area.

When dealing with binary values in any base all 8 digits of a byte (16 digits for a binary word) must be used to avoid problems.

Arithmetic and Logical Operators

There is a generous collection of arithmetic and logical operators allowed in an argument:

`$`	current assembly address	Return with the current assembly address
`n + m`	addition	Return the sum of n plus m.
`n – m`	subtraction	Return the difference of n minus m.
`n * m`	multiplication	Return the unsigned product of n times m.
`n / m`	division	Return the unsigned integer quotient of n divided by m.
`n MOD m`	modulus	Return the unsigned remainder of n divided by m.
`n AND m`	bitwise AND	Return the bitwise logical AND of n and m.
`n OR m`	bitwise inclusive OR	Return the bitwise logical inclusive OR of n and m.
`n XOR m`	bitwise exclusive OR	Return the bitwise logical exclusive OR of n and m.
`NOT n`	bitwise inversion (one's compliment)	Return the bitwise logical inversion of n.
`n SHL m`	shift left	Return n shifted left by m with zero fill.
`n SHR m`	shift right	Return n shifted right by m with zero fill.
`HIGH n`	high byte of word	Return the high-order byte of word n.
`LOW n`	low byte of word	Return the low-order byte of word n.
`n EQ m`	true if equal	Return true (–1) if n is equal to m, else return false (0).
`n GE m`	true if greater than or equal	Return true (–1) if n is greater than or equal to m, else return false (0).
`n GT m`	true if greater than	Return true (–1) if n is greater than m, else return false (0).

`n LE m`	true if less than or equal	Return true (–1) if n is less than or equal to m, else return false (0).
`n LT m`	true if less than	Return true (–1) if n is less than to m, else return false (0).
`n NE m`	true if not equal	Return true (–1) if n is not equal to m, else return false (0).
`NUL n`	true if null	Return true (–1) if n is null, else return false (0).

Symbol Table

During assembly a symbol table is generated: if a PRN file is created, the symbol table is printed as a part of this file. Following each symbol in the table is its value, and following the value is a designator indicating the location/status of the symbol:

I	global label
space	absolute value
*	external label
'	code relative label
"	data relative label
!	common relative label
C	common block name
U	undefined symbol

Assembler Directives

An assembler directive is an operand (column 2) that is used to convey an instruction to the assembler itself, rather than the final code. All assembler directives begin with a period.

.8080

The .8080 directive causes the assembler to accept Intel 8080 op-codes instead of Zilog Z-80 opcodes. This is the default condition and is equivalent to the /I command switch. See the .Z80 assembler directive and the /I and /Z switches.

```
;
        .8080                   ;Use Intel mnemonics
;
```

.COMMENT delim text delim

The .COMMENT directive causes the first non-whitespace character after .COMMENT to be treated as a delimiter character, then everything between that delimiter and the next delimiter to be treated as a comment text. Comment texts have no affect on assembly. The comment text need not be limited to a single line.

```
;
        .COMMENT *This is a comment*
;
        .COMMENT *
This is also a comment,
but is stretched over several lines.
Note that no semicolon is required for each line.*
;
```

.CREF

The .CREF directive causes a cross reference file of type CRF to be created during assembly. This is equivalent to the /C command switch. The .XCREF directive cancels the effect of a .CREF directive. Any number of .CREF/.XCREF pairs may be encountered in the same module. See the .XCREF assembler directive and the /C switch.

```
;
        .CREF                   ;Begin cross-reference
;
        ...                     ;Body of code
```

```
;
        .XCREF  ;End cross-reference
;
        ...                     ;Body of code
;
        .CREF                   ;Begin cross-reference again
;
```

.DEPHASE

The .DEPHASE directive cancels the effect of a .PHASE directive, defining the end of the relocated block of code. See the .PHASE assembler directive.

```
;
        CSEG                    ;Locate in code segment
;
        ...                     ;Body of code
;
        .PHASE  0100H           ;Place code at 0100H
;
        ...                     ;Out-of-phase code
;
        .DEPHASE                ;Return to code segment
;
        ...                     ;Body of code
;
```

.LALL

The .LALL directive causes the entire text of every macro to be listed. The output of the MACRO, REPT, IRP, and IRPC pseudo-ops are controlled by the three directives .LALL, .SALL, and .XALL.

See the .SALL and .XALL assembler directives and the MACRO, REPT, IRP, and IRPC pseudo-operations.

```
;
        .LALL                   ;List all macro text
;
```

.LFCOND

The .LFCOND directive lists conditionals that evaluate as false.

See the .SFCOND and .TFCOND assembler directives.

```
;
        .LFCOND                 ;List false conditionals
;
```

.LIST

The .LIST directive lists output to PRN file (default).

See the .XLIST assembler directive.

```
;
        .LIST                   ;List output
;
```

.PHASE arg

The .PHASE directive allows code to be assembled in one area but executed in another.

See the .DEPHASE assembler directive.

```
;
        CSEG                    ;Locate in code segment
;
        ...                     ;Body of code
;
        .PHASE  0100H           ;Place code at 0100H
;
        ...                     ;Out-of-phase code
```

```
;
        .DEPHASE                ;Return to code segment
;
        ...                     ;Body of code
;
```

.PRINTX delim text delim

The .PRINTX directive causes the first non-whitespace character after .PRINTX to be treated as a delimiter character, then everything between that delimiter and the next delimiter to be treated as a comment-to-screen text. Comment-to-screen texts have no affect on assembly. The comment-to-screen text need not be limited to a single line.

Comment-to-screen texts are displayed during assembly. Unless the IF1 and IF2 pseudo-operations are use, display occurs during both passes.

```
;
        IF1
        .PRINTX +Performing pass 1+
        ELSE
        .PRINTX +Performing pass 2+
        ENDIF
;
        IF      MEMSIZE LT 32
        .PRINTX *File is less than 32K*
        ENDIF
;
```

.RADIX arg

The .RADIX directive sets the default radix to the base specified: 2, 8, 10, or 16. The default if .RADIX is not used is 10 (decimal).

```
;
        LD      A,10            ;Load A with decimal 10
        LD      B,0F1H          ;Load B with hexadecimal F1
;
        .RADIX  16              ;All values in hexadecimal
;
        LD      A,10            ;Load A with decimal 16
        LD      B,0F1           ;"H" not required
;
```

All the following are the same:

```
;
        .RADIX  2               ;All values in binary
        LD      A,10111010      ;Load A with decimal 186
;
        .RADIX  8               ;All values in octal
        LD      A,272           ;Load A with decimal 186
;
        .RADIX  10              ;All values in decimal
        LD      A,186           ;Load A with decimal 186
;
        .RADIX  16              ;All values in hexadecimal
        LD      A,0BA           ;Load A with decimal 186
;
```

.REQUEST filename,filename,...

The .REQUEST directive sends a request to the Link-80 linker to search the listed files for undefined external global labels. The filenames specified must not include filetypes or drive designations.

```
;
        .REQUEST INIT,SUBMAR,MYFILE
;
```

.SALL

The .SALL directive suppresses the listing of all macro text. The output of the MACRO, REPT, IRP, and IRPC pseudo-ops are controlled by the three directives .LALL, .SALL, and .XALL.

See the .LALL and .XALL assembler directives and the MACRO, REPT, IRP, and IRPC pseudo-operations.

```
;
        .SALL                   ;Suppress all macro text listing
;
```

.SFCOND

The SFCOND directive suppresses listing of conditionals that evaluate as false.

See the .LFCOND and TFCOND assembler directives.

```
;
        .SFCOND                 ;Do not list false conditionals
;
```

.TFCOND

The .TFCOND directive toggles the listing of conditionals that evaluate as false: if listing is active it is suppressed; if suppressed, it is made active.

If a /X command switch is in effect, the effect of .TFCOND is reversed.

See the .LFCOND and .SFCOND assembler directives and the /X switch.

```
;
        .LFCOND                 ;List false conditionals
;
        ...
;
        .TFCOND                 ;Do not list false conditionals
;
        ...
;
        .SFCOND                 ;Do not list false conditionals
;
        ...
;
        .TFCOND                 ;List false conditionals
;
        ...
;
        .TFCOND                 ;Do not list false conditionals
;
```

.XALL

The .XALL directive lists of all macro text that is expanded (default). The output of the MACRO, REPT, IRP, and IRPC pseudo-ops are controlled by the three directives .LALL, .SALL, and .XALL.

See the .LALL and .SALL assembler directives and the MACRO, REPT, IRP, and IRPC pseudo-operations.

```
;
        .XALL                   ;List all expanded macro text
;
```

.XCREF

The .XCREF directive suppresses the creation of a cross reference file of type CRF. The .CREF directive cancels the effect of the .XCREF directive. Any number of .CREF/.XCREF directive pairs may be encountered in the same module.

See the .CREF assembler directive and the /C switch.

```
;
        .CREF                   ;Begin cross-reference
;
        ...                     ;Body of code
```

```
;
        .XCREF                  ;End cross-reference
;
        ...                     ;Body of code
;
        .CREF                   ;Begin cross-reference again
;
```

.XLIST

The .XLIST directive suppresses list output to PRN file.

See the .LIST assembler directive.

```
;
        .XLIST                  ;Suppress list output
;
```

.Z80

The .Z80 directive causes the assembler to accept Zilog Z-80 op-codes instead of Intel 8080 opcodes. This is equivalent to the /Z command switch.

See the .8080 assembler directive and the /I and /Z switches.

```
;
        .Z80                    ;Use Zilog mnemonics
;
```

Pseudo Operations

A pseudo-operation is an “op-code” that is a property of the Macro-80 assembler rather than the processor. Except for a couple of minor exceptions, all pseudo-operations are treated as and act like normal op-codes.

Just as the Macro-80 assembler accepts and recognizes both Intel 8080 and Zilog Z-80 op-codes, there are “8080” and “Z-80” pseudo-operations. For the most part, all pseudo-operations work with the Zilog Z-80 mnemonic set, while those indicated “[Z-80]” not work with the Intel 8080 mnemonic set.

$EJECT argument

The $EJECT pseudo-op causes the assembler to start a new output page in the listing file. The value of the argument, if present, becomes the new page size. The value of the argument is measured in lines per page, with a minimum of 10, a maximum of 255, and a default of 50.

Be aware that the assembler outputs a formfeed character (control-L) at the end of each page rather than counting lines. TurboDOS's TYPE utility properly outputs the PRN file to a printer, other utilities may not.

The $EJECT pseudo-op is identical with the PAGE pseudo-op and is provided for historical reasons: the PAGE pseudo-op is preferred.

See the PAGE pseudo-operation.

```
;
        $EJECT  40              ;Eject, set page to 40 lines
;
```

$INCLUDE filename

The $INCLUDE pseudo-op assembles the specified source file into the current module at the location of the $INCLUDE pseudo-op. This eliminates the need to repeat an often used sequence of statements in every module.

Identical with the INCLUDE and MACLIB pseudo-ops, the $INCLUDE pseudo-op is provided for historical reasons: the INCLUDE pseudo-op is preferred.

See the INCLUDE and MACLIB pseudo-operations.

```
;
        $INCLUDE EQUATES        ;Include equates file
;
```

$TITLE text

The $TITLE pseudo-op specifies a title of “text” to be listed on the first line of each page of the PRN listing file. If the NAME pseudo-op is missing, the current module is defined (named) as the first six characters of “text”.

The $TITLE pseudo-op is provided for historical reasons and is identical with the TITLE pseudo-op, which is preferred.

```
;
        $TITLE MYPROG routine main module
;
```

ASEG

The ASEG pseudo-op sets the current location counter to an absolute segment of memory. The location of the absolute segment counter is that of the last value of ASEG (default is 0000H) unless the ASEG pseudo-op is immediately followed by the ORG pseudo-op to change the counter.

See the ORG, CSEG, DSEG, and COMMON pseudo-operations.

```
;
        START:  ASEG            ;Locate in absolute segment
        ORG     2000H           ;Absolute segment is 2000H
;
```

COMMON /name/

The COMMON pseudo-op sets the current location counter to a common segment of memory designated “name”. Unlike the code, data, and absolute segments, controlled by the CSEG, DSEG, and ASEG pseudo-ops, a segment designated by COMMON must be in a single contiguous block of code per module: .PHASE and .DEPHASE assembler directives are not allowed. Multiple modules may use the same common block designations, however, and like blocks are concatenated during linking.

The ORG pseudo-op may be used to place the common segment at any desired location.

See the ORG, ASEG, CSEG, and DSEG pseudo-operations.

```
;
        COMMON  /?INIT?/        ;Locate in ?INIT? segment
        ORG     0000H           ;Init at 0000H
;
```

Macro-80 COMMON memory blocks are compatible with the FORTRAN COMMON statement.

COND argument [Z-80]

The COND pseudo-op causes the code between itself and a following ENDC pseudo-op to be assembled if and only if “argument” evaluates to true (not 0).

If an ELSE pseudo-op is included between the COND and ENDC, then the COND pseudo-op causes the code between itself and ELSE to be assembled if “argument” is true and between ELSE and ENDC if “argument” is false (0).

The COND pseudo-op is identical with the IF and IFT pseudo-ops. The IFT pseudo-op is preferred.

See the IF and IFT pseudo operations.

```
;
        COND    USART               ;Do if USART <> 0
        LD      A,0FFH          ;Initialize the USART
        OUT     (PORT1),A
        LD      A,0BEH
        OUT     (PORT2),A
        ENDC
;

;
        COND    argument
        ...                     ;Do if argument <> 0
        ELSE
        ...                     ;Do if argument = 0
        ENDC
;
```

CSEG

The CSEG pseudo-op sets the current location counter to the code relative or program relative segment of memory. The location of the code segment counter is that of the last value of CSEG (default is 0000H) unless the CSEG pseudo-op is immediately followed by the ORG pseudo-op to change the counter.

See the ORG, ASEG, DSEG, and COMMON pseudo-operations.

```
;
        CSEG                    ;Locate in code segment
;
```

DB argument,argument,...

The DB pseudo-op defines a series of bytes in memory whose values are the values of the arguments. If the arguments are strings, the values of the defined bytes are the ASCII values of the string characters.

The DB pseudo-op is identical with the DEFB and DEFM pseudo-ops, but is preferred.

See the DEFB and DEFM pseudo-operations.

```
;
        DB      6,42,129        ;Defines 06H,2AH,81H
        DB      'TEST'          ;Defines 54H,45H,53H,54H
;
```

DC string

The DC pseudo-op defines a series of bytes in memory whose values are the ASCII values of the string characters of the arguments, with the most significant (sign or parity) bit of the last character in the string being set.

```
;
        DC      'TEST'          ;Defines 54H,45H,53H,0D4H
;
```

DEFB argument,argument,... [Z-80]

The DEFB pseudo-op defines a series of bytes in memory whose values are the values of the arguments. If the arguments are strings, the values of the defined bytes are the ASCII values of the string characters.

The DEFB pseudo-op is identical with the DB and DEFM pseudo-ops. The DB pseudo-op is preferred.

See the DB and DEFM pseudo-operations.

```
;
        DEFB    6,42,129        ;Defines 06H,2AH,81H
        DEFB    'TEST'          ;Defines 54H,45H,53H,54H
;
```

name DEFL argument [Z-80]

The DEFL pseudo-op assigns the value of the argument to the associated name. If the argument is external, an error occurs.

The DEFL pseudo-op is identical to the SET pseudo-op, which is preferred. DEFL is also very similar to EQU, save that no error occurs if an attempt is made to redefine an already existing name.

See the EQU and SET pseudo-operations.

```
;
CR      DEFL    13              ;ASCII carriage return
LF      DEFL    10              ;ASCII linefeed
;
```

DEFM string [Z-80]

The DEFM pseudo-op defines a series of bytes in memory whose values are the ASCII values of the string characters of the arguments.

The DEFM pseudo-op is identical with but a subset of the DB and DEFB pseudo-ops. The DB pseudo-op is preferred.

See the DB and DEFB pseudo-operations.

```
;
        DEFM    'TEST'          ;Defines 54H,45H,53H,54H
```

```
;
```

DEFS argument [Z-80]

The DEFS pseudo-op reserves an area of memory "argument" bytes in size. This area of memory is not initialized unless the /M command switch is used.

The DEFS pseudo-op is identical to the DS pseudo-op, which is preferred.

See the DS pseudo-op and the /M switch.

```
;
        DEFS    128             ;Reserve 128 bytes of memory
;
```

DEFW argument,argument,... [Z-80]

The DEFW pseudo-op defines a series of words in memory whose values are the values of the arguments.

The DEFW pseudo-op is identical with the DW pseudo-op, which is preferred.

See the DW pseudo-operation.

```
;
        DEFW    6,42,6129       ;Defines 0006H,002AH,17F1H
;
```

DS argument

The DS pseudo-op reserves an area of memory "argument" bytes in size. This area of memory is not initialized unless the /M command switch is used.

The DS pseudo-op is identical to the DEFS pseudo-op and is preferred.

See the DEFS pseudo-op and the /M switch.

```
;
        DS      128             ;Reserve 128 bytes of memory
;
```

DSEG

The DSEG pseudo-op sets the current location counter to the data relative segment of memory. The location of the data segment counter is that of the last value of DSEG (default is 0000H) unless the DSEG pseudo-op is immediately followed by the ORG pseudo-op to change the counter.

See the ORG, ASEG, CSEG, and COMMON pseudo-operations.

```
;
        DSEG                    ;Locate in data segment
;
```

DW argument,argument,...

The DW pseudo-op defines a series of words in memory whose values are the values of the arguments.

The DW pseudo-op is identical to the DEFW pseudo-op and is preferred.

See the DEFW pseudo-operation.

```
;
        DW      6,42,6129       ;Defines 0006H,002AH,17F1H
;
```

ELSE

Each conditional pseudo-operation may include an ELSE pseudo-op between itself and its ENDIF or ENDC pseudo-op. The statements between the conditional pseudo-op and the ELSE are assembled if the condition is met, and those between the ELSE and the ENDIF or ENDC if the condition is not met.

```
;
        IF      argument
        ...                     ;Do if argument <> 0
        ELSE
        ...                     ;Do if argument = 0
        ENDIF
```

```
;
```

END argument

The END pseudo-op specifies the end of the program or module and must be the last statement in the module.

If present, the value of “argument” is retained by Macro-80 and passed to Link-80 as the start address of the code (program) segment of the module. If argument is missing, the module retains its relocatability.

It is important that the END pseudo-op not only be included in a module, but that it be on a terminated line. Many programmers have developed the habit of failing to terminate the last line of a program with a CR-LF pair. This is tolerable in most BASIC's and many other languages, but intolerable in Macro-80 (it is also very sloppy programming practice).

```
;
        ...                     ;Code body
;
        END
```

ENDC [Z-80]

The ENDC pseudo-op is used to terminate a conditional assembly started by the COND pseudo-op.

See the COND pseudo operation.

```
;
        COND    USART           ;Do if USART <> 0
        LD      A,0FFH          ;Initialize the USART
        OUT     (PORT1),A
        LD      A,0BEH
        OUT     (PORT2),A
        ENDC
;

;
        COND    argument
        ...                     ;Do if argument <> 0
        ELSE
        ...                     ;Do if argument = 0
        ENDC
;
```

ENDIF

The ENDIF pseudo-op is used to terminate a conditional assembly started by any IFxx pseudo-op.

```
;
        IFT     USART           ;Do if USART <> 0
        LD      A,0FFH          ;Initialize the USART
        OUT     (PORT1),A
        LD      A,0BEH
        OUT     (PORT2),A
        ENDIF
;

;
        IFT     argument
        ...                     ;Do if argument <> 0
        ELSE
        ...                     ;Do if argument = 0
        ENDIF
;
```

ENDM

The ENDM pseudo-op is used to terminate all macros. Macros are those statement groups started by the MACRO, REPT, IRP, and IRPC pseudo-ops.

See the MACRO, REPT, IRP, and IRPC pseudo-operations.

```
;
DIV     MACRO   X
```

```
        ...                     ;Body of macro
        ENDM
;
```

ENTRY name,name,...

The ENTRY pseudo-op declares that every name in the list is internal and global, and therefore available for use by the current and all other modules linked with the current module. All the listed names must be defined within the current module.

The ENTRY pseudo-op is identical with the GLOBAL and PUBLIC pseudo-ops, but the GLOBAL pseudo-op is preferred.

See the GLOBAL and PUBLIC pseudo-operations.

Where defined elsewhere in the module, the names must not have a double colon "::". They are defined as global either by the ENTRY pseudo-op or the double colon, but not both. The following are identical:

```
;
        ENTRY   GLBL1,GLBL2
GLBL1:  ...
GLBL2:  ...
;

;
GLBL1:: ...
GLBL2:: ...
;
```

name EQU argument

The EQU pseudo-op assigns the value of the argument to the associated name. If the argument is external or the name already was defined, an error occurs.

See the DEFL and SET pseudo-operations.

```
;
CR      EQU     13              ;ASCII carriage return
LF      EQU     10              ;ASCII linefeed
;
```

EXITM

The EXITM pseudo-op is intended for use inside the body of a macro, and causes an early termination of the assembly of the macro.

If the block containing the EXITM pseudo-op is nested within another block, the outer block continues to be assembled.

```
;
SUB     MACRO   X
        IFT     X LT 0
        EXITM                   ;Terminate assembly if X<0
        ENDIF
        ...
        ENDM
;
```

EXT name,name,...

The EXT pseudo-op declares that every name in the list is external, that is, defined as global in a different module. The same name may not appear in the current module.

The EXT pseudo-op is identical with the EXTRN and EXTERNAL pseudo-ops. The EXTERN pseudo-op is preferred.

See the EXTRN and EXTERNAL pseudo-operations.

Elsewhere in the module, the names must not have a double number sign "##". They are defined as external either by the EXT pseudo-op or the double number sign but not both. The following are identical:

```
;
        EXT     ELBL1,ELBL2
        CALL    ELBL1
        JP      ELBL2
;
```

```
;
        CALL    ELBL1##
        JP      ELBL2##
;
```

EXTERNAL name,name,... [Z-80]

The EXTERNAL pseudo-op declares that every name in the list is external, that is, defined as global in a different module. The same name may not appear in the current module.

The EXTERNAL pseudo-op is identical with the EXT and EXTRN pseudo-ops. The EXTERN pseudo-op is preferred.

See the EXT and EXTRN pseudo-operations.

Elsewhere in the module, the names must not have a double number sign "##". They are defined as external either by the EXTERNAL pseudo-op or the double number sign but not both. The following are identical:

```
;
        EXTERNAL ELBL1,ELBL2
        CALL    ELBL1
        JP      ELBL2
;

;
        CALL    ELBL1##
        JP      ELBL2##
;
```

EXTRN name,name,...

The EXTRN pseudo-op declares that every name in the list is external, that is, defined as global in a different module. The same name may not appear in the current module.

The EXTRN pseudo-op is identical with the EXT and EXTERNAL pseudo-ops and is preferred.

See the EXT and EXTERNAL pseudo-operations.

Elsewhere in the module, the names must not have a double number sign "##". They are defined as external either by the EXTRN pseudo-op or the double number sign but not both. The following are identical:

```
;
        EXTRN   ELBL1,ELBL2
        CALL    ELBL1
        JP      ELBL2
;

;
        CALL    ELBL1##
        JP      ELBL2##
;
```

GLOBAL name,name,...

The GLOBAL pseudo-op declares that every name in the list is internal and global, and therefore available for use by the current and all other modules linked with the current module. All the listed names must be defined within the current module.

The GLOBAL pseudo-op is identical with the ENTRY and PUBLIC pseudo-ops and is preferred.

See the ENTRY and PUBLIC pseudo-operations.

When defined elsewhere in the module the names must not have a double colon "::". The names are defined as global either by the GLOBAL pseudo-op or the double colon, but no both. The following are identical:

```
;
        GLOBAL  GLBL1,GLBL2
;
GLBL1:  ...
;
GLBL2:  ...
;

;
```

```
GLBL1:: ...
;
GLBL2:: ...
;
```

IF argument

The IF pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if “argument” evaluates to true (not 0).

If an ELSE pseudo-op is included between the IF and ENDIF, then the IF pseudo-op causes the code between itself and ELSE to be assembled if “argument” is true and between ELSE and ENDIF if “argument” is false (0).

The IF pseudo-op is identical with the COND and IFT pseudo-ops. The IFT pseudo-op is preferred.

See the COND and IFT pseudo-operations.

```
;
        IF      USART           ;Do if USART <> 0
        LD      A,0FFH          ;Initialize the USART
        OUT     (PORT1),A
        LD      A,0BEH
        OUT     (PORT2),A
        ENDIF
;

;
        IF      argument
        ...                     ;Do if argument <> 0
        ELSE
        ...                     ;Do if argument = 0
        ENDIF
;
```

IF1

The IF1 pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if the assembler is currently executing pass 1 of the assembly.

If an ELSE pseudo-op is included between the IF1 and ENDIF, then the IF1 pseudo-op causes the code between itself and ELSE to be assembled during pass 1 and between ELSE and ENDIF during pass 2.

See the IF2 pseudo-operation.

```
;
        IF1
        .PRINTX $Hi there!  I'm halfway through pass 1!$
        ENDIF
;

;
        IF1
        ...                     ;Do during pass 1
        ELSE
        ...                     ;Do during pass 2
        ENDIF
;
```

IF2

The IF2 pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if the assembler is currently executing pass 2 of the assembly.

If an ELSE pseudo-op is included between the IF2 and ENDIF, then the IF2 pseudo-op causes the code between itself and ELSE to be assembled during pass 2 and between ELSE and ENDIF during pass 1.

See the IF1 pseudo-operation.

```
;
        IF2
```

```
        .PRINTX +Whew!  I am now halfway through pass 2!+
        ENDIF
;
;
        IF2
        ...                     ;Do during pass 2
        ELSE
        ...                     ;Do during pass 1
        ENDIF
;
```

IFB <argument>

The IFB pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if “argument” is blank. The angle brackets around “argument” are required.

If an ELSE pseudo-op is included between the IFB and ENDIF, then the IFB pseudo-op causes the code between itself and ELSE to be assembled if “argument” is blank and between ELSE and ENDIF if “argument” is not blank.

The IFB pseudo-op is intended for use where parameters have been passed.

See the IFNB pseudo-operation.

```
;
MULTI   MACRO   X,Y,Z
        IFB     <X>
        EXITM
        ENDIF
        ...                     ;Body of macro
        ENDM
;
;
        IFB     <argument>
        ...                     ;Do if blank
        ELSE
        ...                     ;Do if not blank
        ENDIF
;
```

IFDEF name

The IFDEF pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if “name” has been defined or declared external.

If an ELSE pseudo-op is included between the IFDEF and ENDIF, then the IFDEF pseudo-op causes the code between itself and ELSE to be assembled if “name” has been defined or declared external and between ELSE and ENDIF if “name” has neither been defined nor declared external.

See the IFNDEF pseudo-operation.

```
;
        IFDEF   START
        LD      HL,START
        ENDIF
;
;
        IFDEF   name
        ...                     ;Do if defined
        ELSE
        ...                     ;Do if not defined
        ENDIF
;
```

IFDIF <string1>,<string2>

The IFDIF pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if “string1” is different from “string2”. The angle brackets around “string1” and “string2” are required.

If an ELSE pseudo-op is included between the IFDIF and ENDIF, then the IFDIF pseudo-op causes the code between itself and ELSE to be assembled if “string1” is different from “string2” and between ELSE and ENDIF if “string1” if identical to “string2”.

See the IFIDN pseudo-operation.

```
;
        IFDIF   <X>,<Y>
        .PRINTX +The strings are different!+
        ENDIF
;

;
        IFDIF   <string1>,<string2>
        ...                     ;Do if different
        ELSE
        ...                     ;Do if identical
        ENDIF
;
```

IFE argument

The IFE pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if “argument” evaluates to false (0).

If an ELSE pseudo-op is included between the IFE and ENDIF, then the IFE pseudo-op causes the code between itself and ELSE to be assembled if “argument” is false and between ELSE and ENDIF if “argument” is true (not 0).

The IFE pseudo-op is identical with the IFF pseudo-op, which is preferred.

See the IFF pseudo-operation.

```
;
        IFE     USART           ;Do if USART = 0
        LD      A,0FFH          ;Initialize the USART
        OUT     (PORT1),A
        LD      A,0BEH
        OUT     (PORT2),A
        ENDIF
;

;
        IFE     argument
        ...                     ;Do if argument = 0
        ELSE
        ...                     ;Do if argument <> 0
        ENDIF
;
```

IFF argument

The IFF pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if “argument” evaluates to false (0).

If an ELSE pseudo-op is included between the IFF and ENDIF, then the IFF pseudo-op causes the code between itself and ELSE to be assembled if “argument” is false and between ELSE and ENDIF if “argument” is true (not 0).

The IFF pseudo-op is identical with the IFE pseudo-op and is preferred.

See the IFE pseudo-operation.

```
;
        IFF     USART           ;Do if USART = 0
        LD      A,0FFH          ;Initialize the USART
        OUT     (PORT1),A
        LD      A,0BEH
        OUT     (PORT2),A
        ENDIF
;

;
```

```
        IFF     argument
        ...                     ;Do if argument = 0
        ELSE
        ...                     ;Do if argument <> 0
        ENDIF
;
```

IFIDN <string1>,<string2>

The IFIDN pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if “string1” is identical to “string2”. The angle brackets around “string1” and “string2” are required.

If an ELSE pseudo-op is included between the IFIDN and ENDIF, then the IFIDN pseudo-op causes the code between itself and ELSE to be assembled if “string1” is identical to “string2” and between ELSE and ENDIF if “string1” if different from “string2”.

See the IFDIF pseudo-operation.

```
;
        IFIDN   <X>,<Y>
        .PRINTX +The strings are identical!+
        ENDIF
;

;
        IFIDN   <string1>,<string2>
        ...                     ;Do if identical
        ELSE
        ...                     ;Do if different
        ENDIF
;
```

IFNB <argument>

The IFNB pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if “argument” is not blank. The angle brackets around “argument” are required.

If an ELSE pseudo-op is included between the IFNB and ENDIF, then the IFNB pseudo-op causes the code between itself and ELSE to be assembled if “argument” is not blank and between ELSE and ENDIF if “argument” is blank.

The IFNB pseudo-op is intended for use where parameters have been passed.

See the IFB pseudo-operation.

```
;
MULTI   MACRO   X,Y,Z
        IFNB    <X>
        ...                     ;Body of macro
        ENDIF
        ENDM
;

;
        IFNB    <argument>
        ...                     ;Do if not blank
        ELSE
        ...                     ;Do if blank
        ENDIF
;
```

IFNDEF name

The IFNDEF pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if “name” has neither been defined nor declared external.

If an ELSE pseudo-op is included between the IFNDEF and ENDIF, then the IFNDEF pseudo-op causes the code between itself and ELSE to be assembled if “name” has neither been defined nor declared external and between ELSE and ENDIF if “name” has been defined or declared external.

See the IFDEF pseudo-operation.

```
;
        IFNDEF  START
        LD      HL,OTHER
        ENDIF
;

;
        IFNDEF  name
        ...                     ;Do if not defined
        ELSE
        ...                     ;Do if defined
        ENDIF
;
```

IFT argument

The IFT pseudo-op causes the code between itself and a following ENDIF pseudo-op to be assembled if and only if "argument" evaluates to true (not 0).

If an ELSE pseudo-op is included between the IFT and ENDIF, then the IFT pseudo-op causes the code between itself and ELSE to be assembled if "argument" is true and between ELSE and ENDIF if "argument" is false (0).

The IFT pseudo-op is identical with the COND and IF pseudo-ops and is preferred.

See the COND and IF pseudo operations.

```
;
        IFT     USART           ;Do if USART <> 0
        LD      A,0FFH          ;Initialize the USART
        OUT     (PORT1),A
        LD      A,0BEH
        OUT     (PORT2),A
        ENDIF
;

;
        IFT     argument
        ...                     ;Do if argument <> 0
        ELSE
        ...                     ;Do if argument = 0
        ENDIF
;
```

INCLUDE filename

The INCLUDE pseudo-op assembles the specified source file into the current module at the location of the INCLUDE pseudo-op. This eliminates the need to repeat an often used sequence of statements in every module.

Identical with the $INCLUDE and MACLIB pseudo-ops, the INCLUDE pseudo-op is preferred.

See the $INCLUDE and MACLIB pseudo-operations.

```
;
        INCLUDE EQUATES         ;Include equates file
;
```

IRP dummy,<arg-list>

The IRP pseudo-op repeats the block of statements as many times as there are arguments in "arg-list", substituting the argument-at-hand for every occurrence of dummy. The angle brackets around "arg-list" are required.

The following code:

```
;
        IRP     X,<0,2,4,6,8>
        DB      X,X+1
        ENDM
;
```

assembles as:

```
;
        DB      0,1
        DB      2,3
        DB      4,5
        DB      6,7
        DB      8,9
;
```

IRPC dummy,<string>

The IRPC pseudo-op repeats the block of statements as many times as there are characters in “string”, substituting the character-at-hand for every occurrence of dummy. The angle brackets around “string” are optional.

The following code:

```
;
SIZE    SET     10
        IRPC    X,<01234>
        DB      X+1
        DS      X+SIZE
        ENDM
;
```

assembles as:

```
;
        DB      1
        DS      10
        DB      2
        DS      11
        DB      3
        DS      12
        DB      4
        DS      13
        DB      5
        DS      14
;
```

LOCAL <dummy-list>

The LOCAL pseudo-op creates a unique symbol for each “dummy” in the “dummy-list” and substitutes that symbol for each occurrence of the “dummy” in the statement block.

The LOCAL pseudo-op is allowed only inside a macro and must be the first statement in the macro.

The following macro:

```
;
        MACRO   X,Y
        LOCAL   A,B,C
A:      JR      B
        NOP
B:      JP      C
        NOP
C:      JR      A
        ENDM
;
```

assembles properly no matter how many time it is expanded within the body of the module (no duplicate labels occur).

MACLIB filename

The MACLIB pseudo-op assembles the specified source file into the current module at the location of the MACLIB pseudo-op. This eliminates the need to repeat an often used sequence of statements in every module.

Identical with the $INCLUDE and INCLUDE pseudo-ops, the MACLIB pseudo-op is provided for historical reasons: the INCLUDE pseudo-op is preferred.

See the $INCLUDE and INCLUDE pseudo-operations.

```
;
        MACLIB  EQUATES         ;Include equates file
;
```

name MACRO dummy-list

The MACRO pseudo-op generates a sequence of macro statements from differing places in the module while allowing different parameters to be passed each time the macro is used. The “name” must conform to the label conventions and is used to invoke the macro itself. The “dummy-list” provides the parameters that are passed into the macro and are different each time the macro is expanded.

```
;
ADDVAR  MACRO   T,U
        LD      A,T
        ADD     A,U
        ENDM
;
```

NAME ('name')

The NAME pseudo-op defines the module name as “name”. The “name” argument must conform to the rules of labels.

If the NAME pseudo-op is omitted, the module is defined by the $TITLE or TITLE pseudo-op. If all are omitted, the module is defined by the first six character of the source file name.

If some versions of Macro-80, the parentheses are optional, in others they are required.

See the $TITLE and TITLE pseudo-operations.

```
;
        NAME     ('MYPROG')     ;Module ID
;
```

ORG argument

The ORG pseudo-op set the current location counter to the value of “argument”, and the assembler then assigns code to addresses beginning at that value.

The value of “argument” may be absolute or in the same area as the current location counter (ASEG, DSEG, CSEG, or COMMON). All expressions used in “argument” must be recognized on pass 1 of the assembler.

See the ASEG, CSEG, DSEG, and COMMON pseudo-operations.

```
;
        CSEG                    ;Locate in code segment
BEGIN:  ORG     0100H           ;Set code segment at 0100H
;
        ...                     ;Body of code
;
        END     BEGIN           ;Chain code segment to linker
;
```

PAGE argument

The PAGE pseudo causes the assembler to start a new output page in the listing file. The value of the argument, if present, becomes the new page size. The value of the argument is measured in lines per page, with a minimum of 10, a maximum of 255, and a default of 50. Be aware that the assembler output a formfeed character (control-L) at the end of each page rather than counting lines. TurboDOS's TYPE utility properly outputs the PRN file to a printer, other utilities may not.

The PAGE pseudo-op is identical with the $EJECT pseudo-op and is preferred.

See the $EJECT pseudo-operation.

```
;
        PAGE    40              ;Eject, set page to 40 lines
;
```

PUBLIC name,name,...

The PUBLIC pseudo-op declares that every name in the list is internal and global, and therefore available for use by the current and all other modules linked with the current module. All the listed names must be defined within the current module.

The PUBLIC pseudo-op is identical with the ENTRY and GLOBAL pseudo-ops, but the GLOBAL pseudo-op is preferred.

See the ENTRY and GLOBAL pseudo-operations.

When defined elsewhere in the module the names must not have a double colon "::". The names are defined as global either by the PUBLIC pseudo-op or the double colon, but no both. The following are identical:

```
;
        PUBLIC  GLBL1,GLBL2
;
GLBL1:  ...
;
GLBL2:  ...
;

;
GLBL1:: ...
;
GLBL2:: ...
;
```

REPT argument

The REPT pseudo-op repeats the statement block between itself and ENDM "argument" times, where "argument" must resolve to a 16-bit unsigned integer.

The following code:

```
;
        SET     0
        REPT    6
        SET     X X+1
        DB      X
        ENDM
;
```

assembles as:

```
;
        DB      1
        DB      2
        DB      3
        DB      4
        DB      5
        DB      6
;
```

name SET argument

The SET pseudo-op assigns the value of the argument to the associated name. If the argument is external, an error occurs.

The SET pseudo-op is identical to the DEFL pseudo-op and is preferred. SET is also very similar to EQU, save that no error occurs if an attempt is made to redefine an already existing name.

See the DEFL and EQU pseudo-operations.

```
;
CR      SET     13              ;ASCII carriage return
LF      SET     10              ;ASCII linefeed
;
```

SUBTTL text

The SUBTTL pseudo-op specifies a subtitle of "text" to be listed on the second line of each page of the PRN listing file.

```
;
        TITLE   MYPROG routine main module
        SUBTTL  Copyright (C) 1990, R. Roger Breton
;
```

TITLE text

The TITLE pseudo-op specifies a title of "text" to be listed on the first line of each page of the PRN listing file.

If the NAME pseudo-op is missing, the current module is defined named) as the first six characters of “text”.

The TITLE pseudo-op is identical with the $TITLE pseudo-op and is preferred.

```
;
        TITLE   MYPROG routine main module
;
```

TYPE argument

The TYPE pseudo-op returns a single byte that describes the mode of the assembler and indicates whether “argument” is locally or externally defined. If “argument” is invalid, TYPE returns a zero.

The returned byte is evaluated on a bit basis as follows:

xxxxxx00	absolute mode
xxxxxx01	code relative mode
xxxxxx10	data relative mode
xxxxxx11	common relative mode
xx0xxxxx	undefined or external
xx1xxxxx	locally defined
0xxxxxxx	“argument” is local
1xxxxxxx	“argument” contains an external

```
;
TEST    MACRO   X
        LOCAL   Z
Z       SET     TYPE X
        IFT     Z
        ...
;
```

Error Codes

Macro-80 responds to an assembly error via an error code.

A Argument Error

The argument of a pseudo-op or directive is out of range or not in correct form.

C Conditional Nesting Error

A conditional is improperly nested, such as ELSE without IFxx, ENDIF without IFxx, or two ELSE's on one IFxx.

D Double Definition Error

A symbol has been defined more than once.

E External Error

There has been illegal use of an external label.

M Multiple Definition Error

A label has been defined more than once.

N Number Error

An illegal digit was used, such as 8Q (octal uses the digits 0–7).

O Opcode or Syntax Error

An illegal use of a pseudo-op has occurred, such as ENDM or LOCAL outside of a macro, or SET, EQU, or MACRO without an attendant name.

A syntax error in an op-code has occurred, such as “LD A,DE”.

A syntax error in an expression has occurred, such as mismatched parentheses, quotes, concatenated operators, etc.

P Phase Error
A label or symbol has different values during each assembler pass.

Q Questionable Error or Warning
A line is usually improperly terminated. This is a warning error.

R Relocation Error
An illegal use of relocation in an expression has occurred, such as an attempt to relocate an absolute label. Only code, data, and common areas are relocatable.

U Undefined Symbol Error
An attempt was made to use a symbol that has not been defined.

For certain pseudo-ops, a V error occurs on pass 1 and a U error on pass 2.

V Value Error
An attempt was made to perform a pass-1 assembly of a pseudo-op whose value is unknown at the time of assembly. If the value remains unknown, a U error occurs on pass 2. If the value is later defined, there is no pass-2 error.

REL File Format

Microsoft REL files consist of a bit stream, as contrasted to the byte stream of normal object (COM) files. Individual field within the bit stream are NOT aligned on byte boundaries, except as indicated below. The use of a bit steam was chosen to keep file size to a minimum.

There are two basic types of bit stream fields: absolute and relocatable.

Absolute fields consist of a 0 bit, followed by the 8-bits of the absolute byte.

```
0 11000011              ;absolute C3h  JP 0050h
0 01010000              ;absolute 50h
0 00000000              ;absolute 00h
```

Relocatable field consist of a 1 bit, followed by 2 bits indicating the type of relocatable item, followed by the relocatable item. The two bits indicating the type of relocatable item are:

```
1 00 ...                ;special item
1 01 nnnnnnnn nnnnnnnn  ;code-relative item
1 10 nnnnnnnn nnnnnnnn  ;data-relative item
1 11 nnnnnnnn nnnnnnnn  ;common-relative item
```

With the code, data, and common relative items, the relocation address is determined by adding the following 16 bits to the code, data, or common base address.

With the special items, the format is as follows:

```
1                       ;relocatable field
00                      ;special item
yy nnnnnnnn nnnnnnnn    ;Optional "A" subfield
zzz cccccccc cccccccc cccccccc ...      ;Optional "B" subfield
```

The “A” subfield consists of a 2-bit address field type (yy) followed by a 16-bit value (nnnnnnnn nnnnnnnn):

```
00 nnnnnnnn nnnnnnnn    ;absolute address
01 nnnnnnnn nnnnnnnn    ;code-relative address
10 nnnnnnnn nnnnnnnn    ;data-relative address
11 nnnnnnnn nnnnnnnn    ;common-relative address
```

The “B” subfield consists of a 3-bit symbol length (1–6 bytes) followed by the symbol:

```
001 cccccccc            ;1-byte symbol
010 cccccccc cccccccc   ;2-byte symbol
...
```

The sixteen special items are as follows:

```
1 00 0000 "B"           ;Entry symbol (name for search).
1 00 0001 "B"           ;Select common block.
```

```
        1 00 0010 "B"           ;Program/module name.
        1 00 0011 "B"           ;Request library search.
        1 00 0100 special       ;Extension item.
        1 00 0101 "A" "B"       ;Define common size.
        1 00 0110 "A" "B"       ;Chain external:  "A" is head of address chain, "B" is
                                ;    name of external symbol.
        1 00 0111 "A" "B"       ;Define entry point:  "A" is address, "B" is name.
        1 00 1000 "A"           ;External - offset:  used for JP and CALL.
        1 00 1001 "A"           ;External + offset:  "A" will be added to the two
                                ;    bytes starting at the current location counter
                                ;    immediately before execution.
        1 00 1010 "A"           ;Define data size.
        1 00 1011 "A"           ;Set loading location counter.
        1 00 1100 "A"           ;Chain address:  "A" is head of chain.  Replace all
                                ;    entries in chain with current location counter.
                                ;    Last entry in chain has address field of absolute
                                ;    zero.
        1 00 1101 "A"           ;Define code size.
        1 00 1110 "A"           ;End program/module.
        1 00 1111               ;End file.
```

An extension item has the format:

```
        1 00 0100 zzz ssssssss bbbbbbbb bbbbbbbb ...
```

where “zzz” is a 3-bit item length, followed by an 8-bit subtype identifier, followed by 1–7 bytes of special information or symbol (symbols limited to 6 bytes):

```
        1 00 0100 001 ssssssss                  ;no info/symbol
        1 00 0100 010 ssssssss bbbbbbbb         ;1-byte info/symbol
        ...
```

At present, only the following extension item subtypes are allowed:

```
        00110101                ;COBOL overlay segment sentinel
        01000001                ;Arithmetic fixup (arithmetic operator)
        01000010                ;Arithmetic fixup (external reference)
        01000011                ;Arithmetic fixup (area base + offset)
```

Sample Source Code

A sample source code follows (some comments have been eliminated to shorten code:

```
;CONDRV.MAC
;
;Authors:       R. Roger Breton
;Version:       3.00
;
        NAME    ('CONDRV')      ;Module ID
        .Z80                        ;Zilog mnemonic set
;
STRING  MACRO   A,B,C,D
        IFNB    <D>
        DB      A,B,C,D+128
        ELSE
        IFNB    <C>
        DB      A,B,C+128,128
        ELSE
        IFNB    <B>
        DB      A,B+128,128,128
        ELSE
        IFNB    <A>
        DB      A+128,128,128,128
        ELSE
```

```
        DB      128,128,128,128
        ENDIF
        ENDIF
        ENDIF
        ENDIF
        DB      128,128,128,128
        DB      128,128,128,128
        DB      128,128,128,128
        ENDM
;
        DSEG                        ;Locate in data segment
;
CONBR:: DB      0CFH                ;Baud-rate code
FFCHR::                             ;Old-style formfeed label
CONFF:: DB      12                  ;Formfeed character
CONCLS::STRING  26                  ;Clear-screen string
CONSOS::STRING  13,10               ;Shift-out string
CONSIS::STRING  13,10               ;Shift-in string
CONOFF::DB      0                   ;Output-disable byte
CONON:: DB      0                   ;Output-enable byte
;
        CSEG                        ;Locate in prog area
;
;       Note:  The "JP IN.0" is replaced by "LD A,E" and "SUB 2"
;              by initialization.
;
CONDR@::
DR.0:   JP      IN.0                ;Skip to initialization
;
;       LD      A,E                 ;Get function code
;       SUB     2                   ;Function 0 or 1:  get status or character
        JP      C,SERIAL##
        JP      Z,F2.0              ;Function 2:  output character
        SUB     6                   ;Function 8:  shift out for error message
        JP      Z,F8.0
        DEC     A                   ;Function 9:  shift in from error message
        JP      Z,F9.0
        DEC     A                   ;Function 10:  conditional output character
        JP      Z,F2.0
        RET                         ;Illegal function, done
;
IN.0:   PUSH    DE                  ;Save parameters
        PUSH    BC
        LD      A,7BH               ;Make 1st byte of router "LD A,E"
        LD      (DR.0),A
        LD      A,0D6H              ;Make 2nd & 3rd bytes of router "SUB 2"
        LD      (DR.0+1),A
        LD      A,2
        LD      (DR.0+2),A
        XOR     A                   ;Preclear output-disable key byte
        LD      (F2.1),A
        LD      A,(CONBR)       ;Set the Baud rate
        LD      C,A
        LD      E,03
        CALL    SERIAL##
        CALL    F2.3                ;Clear the screen
        POP     BC                  ;Restore parameters
        POP     DE
        JR      DR.0                ;Go to router
;
F2.0:   LD      A,(CONOFF)      ;Output-disable enabled?
```

```
        OR      A
        JR      Z,F2.2              ;If no, skip
        CP      C                   ;Disable output?
        JR      Z,F2.4              ;If yes, skip
        LD      A,(CONON)         ;Enable output?
        CP      C
        JR      Z,F2.5              ;If yes, skip
F2.1:   NOP                         ;Output-disable key byte
F2.2:   LD      A,(CONFF)         ;Formfeed disabled?
        OR      A
        JP      Z,SERIAL##        ;If yes, output the character
        CP      C                   ;Formfeed?
        JP      NZ,SERIAL##       ;If no, output the character
F2.3:   LD      HL,CONCLS         ;Output clear-screen string
        JP      DMSHL##
;
F2.4:   LD      A,0C9H              ;Set key byte to "RET"
        LD      (F2.1),A
        RET                         ;Done
;
F2.5:   XOR     A                   ;Clear key byte
        LD      (F2.1),A
        RET                         ;Done
;
F8.0:   LD      HL,CONSOS         ;Output error shift-out string
        JP      DMSHL##
;
F9.0:   LD      HL,CONSIS         ;Output error shift-in string
        JP      DMSHL##
;
        END
```

Sample PRN File

The preceding source code assembles into the following PRN file (all comments have been removed to shorten code):

```
                                        NAME    ('CONDRV')
                                        .Z80
  0000'                                 DSEG
  0000"   CF                    CONBR:: DB      0CFH
  0001"                         FFCHR::
  0001"   0C                    CONFF:: DB      12
  0002"                         CONCLS::STRING  26
  0002"   9A 80 80 80     +             DB      26+128,128,128,1
  0006"   80 80 80 80     +             DB      128,128,128,128
  000A"   80 80 80 80     +             DB      128,128,128,128
  000E"   80 80 80 80     +             DB      128,128,128,128
  0012"                         CONSOS::STRING  13,10
  0012"   0D 8A 80 80     +             DB      13,10+128,128,12
  0016"   80 80 80 80     +             DB      128,128,128,128
  001A"   80 80 80 80     +             DB      128,128,128,128
  001E"   80 80 80 80     +             DB      128,128,128,128
  0022"                         CONSIS::STRING  13,10
  0022"   0D 8A 80 80     +             DB      13,10+128,128,12
  0026"   80 80 80 80     +             DB      128,128,128,128
  002A"   80 80 80 80     +             DB      128,128,128,128
  002E"   80 80 80 80     +             DB      128,128,128,128
  0032"   00                    CONOFF::DB      0
  0033"   00                    CONON:: DB      0
  0034"                                 CSEG
  0000'                         CONDR@::
```

```
0000'   C3 0017'             DR.0:   JP      IN.0
0003'   DA 0000*                     JP      C,SERIAL##
0006'   CA 003C'                     JP      Z,F2.0
0009'   D6 06                        SUB     6
000B'   CA 0068'                     JP      Z,F8.0
000E'   3D                          DEC     A
000F'   CA 006E'                     JP      Z,F9.0
0012'   3D                          DEC     A
0013'   CA 003C'                     JP      Z,F2.0
0016'   C9                          RET
0017'   D5                  IN.0:   PUSH    DE
0018'   C5                          PUSH    BC
0019'   3E 7B                       LD      A,7BH
001B'   32 0000'                    LD      (DR.0),A
001E'   3E D6                       LD      A,0D6H
0020'   32 0001'                    LD      (DR.0+1),A
0023'   3E 02                       LD      A,2
0025'   32 0002'                    LD      (DR.0+2),A
0028'   AF                          XOR     A
0029'   32 004B'                    LD      (F2.1),A
002C'   3A 0000"                    LD      A,(CONBR)
002F'   4F                          LD      C,A
0030'   1E 03                       LD      E,03
0032'   CD 0000*                    CALL    SERIAL##
0035'   CD 0057'                    CALL    F2.3
0038'   C1                          POP     BC
0039'   D1                          POP     DE
003A'   18 C4                       JR      DR.0
003C'   3A 0032"             F2.0:   LD      A,(CONOFF)
003F'   B7                          OR      A
0040'   28 0A                       JR      Z,F2.2
0042'   B9                          CP      C
0043'   28 18                       JR      Z,F2.4
0045'   3A 0033"                    LD      A,(CONON)
0048'   B9                          CP      C
0049'   28 18                       JR      Z,F2.5
004B'   00                  F2.1:   NOP
004C'   3A 0001"             F2.2:   LD      A,(CONFF)
004F'   B7                          OR      A
0050'   CA 0000*                    JP      Z,SERIAL##
0053'   B9                          CP      C
0054'   C2 0000*                    JP      NZ,SERIAL##
0057'   21 0002"             F2.3:   LD      HL,CONCLS
005A'   C3 0000*                    JP      DMSHL##
005D'   3E C9                F2.4:   LD      A,0C9H
005F'   32 004B'                    LD      (F2.1),A
0062'   C9                          RET
0063'   AF                  F2.5:   XOR     A
0064'   32 004B'                    LD      (F2.1),A
0067'   C9                          RET
0068'   21 0012"             F8.0:   LD      HL,CONSOS
006B'   C3 0000*                    JP      DMSHL##
006E'   21 0022"             F9.0:   LD      HL,CONSIS
0071'   C3 0000*             JP      DMSHL##
                                    END
```

Sample REL File

The preceding source code also assembles into the following REL file. The source code has been laid aside the REL code to show the relationship. Special attention should be paid to the method used to control externals.

```
85  1.......            / 0000'  Program name ---------       NAME ('CONDRV')
    .00.....            |
    ...0010.            |
90  .......1 10......  |            6 chrs
D3  ..010000 11......  |               "C"
D3  ..010011 11......  |               "O"
91  ..010011 10......  |               "N"
14  ..010001 00......  |               "D"
95  ..010100 10......  |               "R"
A0  ..010101 10......   \               "V"
    ..1.....            / 0000'  Entry Symbol name
    ...00...            |
54  .....000 0.......  |
    .101....            |            5 chrs
34  ....0100 0011....  |               "C"
F4  ....0100 1111....  |               "O"
E4  ....0100 1110....  |               "N"
25  ....0100 0010....  |               "B"
28  ....0101 0010....   \               "R"
    ....1...            / 0000'  Entry Symbol name
    .....00.            |
15  .......0 000.....  |
    ...101..            |            5 chrs
19  ......01 000110..  |               "F"
19  ......01 000110..  |               "F"
0D  ......01 000011..  |               "C"
21  ......01 001000..  |               "H"
4A  ......01 010010..   \               "R"
    ......1.            / 0000'  Entry Symbol name
05  .......0 0.......  |
    .0000...            |
    .....101            |            5 chrs
43  01000011            |               "C"
4F  01001111            |               "O"
4E  01001110            |               "N"
46  01000110            |               "F"
46  01000110            \               "F"
81  1.......            / 0000'  Entry Symbol name
    .00.....            |
    ...0000.            |
90  .......1 10......  |            6 chrs
D3  ..010000 11......  |               "C"
D3  ..010011 11......  |               "O"
90  ..010011 10......  |               "N"
D3  ..010000 11......  |               "C"
14  ..010011 00......  |               "L"
E0  ..010100 11......   \               "S"
    ..1.....            / 0000'  Entry Symbol name
    ...00...            |
64  .....000 0.......  |
    .110....            |            6 chrs
34  ....0100 0011....  |               "C"
F4  ....0100 1111....  |               "O"
E5  ....0100 1110....  |               "N"
34  ....0101 0011....  |               "S"
F5  ....0100 1111....  |               "O"
38  ....0101 0011....   \               "S"
    ....1...            / 0000'  Entry Symbol name
    .....00.            |
19  .......0 000.....  |
    ...110..            |            6 chrs
```

```
0D  ......01 000011..  |              "C"
3D  ......01 001111..  |              "O"
39  ......01 001110..  |              "N"
4D  ......01 010011..  |              "S"
25  ......01 001001..  |              "I"
4E  ......01 010011..   \              "S"
    ......1.            / 0000'  Entry Symbol name
06  .......0 0.......  |
    .0000...            |
    .....110            |           6 chrs
43  01000011            |              "C"
4F  01001111            |              "O"
4E  01001110            |              "N"
4F  01001111            |              "O"
46  01000110            |              "F"
46  01000110            \              "F"
81  1.......            / 0000'  Entry Symbol name
    .00.....            |
    ...0000.            |
50  .......1 01......  |           5 chrs
D3  ..010000 11......  |              "C"
D3  ..010011 11......  |              "O"
93  ..010011 10......  |              "N"
D3  ..010011 11......  |              "O"
A0  ..010011 10......   \              "N"
    ..1.....            / 0000'  Entry Symbol name
    ...00...            |
64  .....000 0.......  |
    .110....            |           6 chrs
34  ....0100 0011....  |              "C"
F4  ....0100 1111....  |              "O"
E4  ....0100 1110....  |              "N"
45  ....0100 0100....  |              "D"
24  ....0101 0010....  |              "R"
09  ....0100 0000....   \              "@"
    ....1...            / 0000'  Define data size
    .....00.            |
41  .......1 010.....  |
    ...00...            |           absolute
A0  .....001 10100...  |              0034
04  .....000 00000...   \
    .....1..            / 0000'  Define code size
    ......00            |
D5  1101....            |
    ....01..            |           code-relative
D0  ......01 110100..  |              0074
02  ......00 000000..   \
    ......1.            / 0000'  Set location counter -       DSEG
5C  .......0 0.......  |
    .1011...            |
    .....10.            |           data-relative
00  .......0 0000000.  |              0000
00  .......0 0000000.   \
    .......0            / 0000"  absolute -------------       DB   0CF
CF  11001111            \           CF
06  0.......            / 0001"  absolute -------------       DB   0C
26  .0000110 0.......   \           OC
    .0......            / 0002"  absolute -------------       DB   9A,80,80,80
90  ..100110 10......   \           9A
    ..0.....            / 0003"  absolute
08  ...10000 000.....   \           80
```

```
    ...0....            / 0004"  absolute
04  ....1000 0000....   \           80
    ....0...            / 0005"  absolute
02  .....100 00000...   \           80
    .....0..            / 0006"  absolute ------------       DB   80,80,80,80
01  ......10 000000..   \           80
    ......0.            / 0007"  absolute
00  .......1 0000000.   \           80
    .......0            / 0008"  absolute
80  10000000            \           80
40  0.......            / 0009"  absolute
20  .1000000 0.......   \           80
    .0......            / 000A"  absolute ------------       DB   80,80,80,80
10  ..100000 00......   \           80
    ..0.....            / 000B"  absolute
08  ...10000 000.....   \           80
    ...0....            / 000C"  absolute
04  ....1000 0000....   \           80
    ....0...            / 000D"  absolute
02  .....100 00000...   \           80
    .....0..            / 000E"  absolute ------------       DB   80,80,80,80
01  ......10 000000..   \           80
    ......0.            / 000F"  absolute
80  .......1 0000000.   \           80
        .......0        / 0010"  absolute
80      10000000        \           80
40      0.......        / 0011"  absolute
03      .1000000 0..... \           80
    .0......            / 0012"  absolute ------------       DB   0D,8A,80,80
51  ..000011 01......   \           0D
    ..0.....            / 0013"  absolute
48  ...10001 010.....   \           8A
    ...0....            / 0014"  absolute
04  ....1000 0000....   \           80
    ....0...            / 0015"  absolute
02  .....100 00000...   \           80
    .....0..            / 0016"  absolute ------------       DB   80,80,80,80
01  ......10 000000..   \           80
    ......0.            / 0017"  absolute
00  .......1 0000000.   \           80
    .......0            / 0018"  absolute
80  10000000            \           80
40  0.......            / 0019"  absolute
20  .1000000 0.......   \           80
    .0......            / 001A"  absolute ------------       DB   80,80,80,80
10  ..100000 00......   \           80
    ..0.....            / 001B"  absolute
08  ...10000 000.....   \           80
    ...0....            / 001C"  absolute
04  ....1000 0000....   \           80
    ....0...            / 001D"  absolute
02  .....100 00000...   \           80
    .....0..            / 001E"  absolute ------------       DB   80,80,80,80
01  ......10 000000..   \           80
    ......0.            / 001F"  absolute
00  .......1 0000000.   \           80
    .......0            / 0020"  absolute
80  10000000            \           80
40  0.......            / 0021"  absolute
03  .1000000 0.......   \           80
    .0......            / 0022"  absolute ------------       DB   0D,8A,80,80
```

```
51  ..000011 01......   \           0D
    ..0.....            / 0023"  absolute
48  ...10001 010.....   \           8A
    ...0....            / 0024"  absolute
04  ....1000 0000....   \           80
    ....0...            / 0025"  absolute
02  .....100 00000...   \           80
    .....0..            / 0026"  absolute ------------       DB   80,80,80,80
01  ......10 000000..   \           80
    ......0.            / 0027"  absolute
00  .......1 0000000.   \           80
    .......0            / 0028"  absolute
80  10000000            \           80
40  0.......            / 0029"  absolute
20  .1000000 0.......   \           80
    .0......            / 002A"  absolute ------------       DB   80,80,80,80
10  ..100000 00......   \           80
    ..0.....            / 002B"  absolute
08  ...10000 000.....   \           80
    ...0....            / 002C"  absolute
04  ....1000 0000....   \           80
    ....0...            / 002D"  absolute
02  .....100 00000...   \           80
    .....0..            / 002E"  absolute ------------       DB   80,80,80,80
01  ......10 000000..   \           80
    ......0.            / 002F"  absolute
00  .......1 0000000.   \           80
    .......0            / 0030"  absolute
80  10000000            \           80
40  0.......            / 0031"  absolute
00  .1000000 0.......   \           80
    .0......            / 0032"  absolute ------------       DB   00
00  ..000000 00......   \           00
    ..0.....            / 0033"  absolute ------------       DB   00
12  ...00000 000.....   \           00
    ...1....            / 0033   Set location counter -      CSEG
    ....00..           |
D0  ......10 11......  |
    ..01....           |       code-relative
00  ....0000 0000....  |           0000'
06  ....0000 0000....   \
    ....0...            / 0000'  absolute ------------ DR.0: JP   IN.0
1D  .....110 00011...   \           C3
    .....1..            / 0001'  code-relative offset
    ......01           |
17  00010111           |           0017'
00  00000000            \
    0.......            / 0003'  absolute ------------       JP   C,SERIAL##
00  .1101101 0.......   \           DA
    .0......            / 0004'  absolute
00  ..000000 00......   \           00
    ..0.....            / 0005'  absolute
0C  ...00000 000.....   \           00
    ...0....            / 0006'  absolute ------------       JP   Z,F2.0
AA  ....1100 1010....   \           CA
    ....1...            / 0007'  code-relative offset
    .....01.           |
78  .......0 0111100.  |           003C'
00  .......0 0000000.   \
    .......0            / 0009'  absolute ------------       SUB  6
D6  11010110            \           D6
```

```
03  0.......            / 000A'  absolute
32  .0000011 0.......   \           06
    .0......            / 000B'  absolute ------------       JP   Z,F8.0
AB  ..110010 10......   \           CA
    ..1.....            / 000C'  code-relative offset
    ...01...           |
40  .....011 01000...  |           0068'
00  .....000 00000...   \
    .....0..            / 000E'  absolute ------------       DEC  A
F5  ......00 111101..   \           3D
    ......0.            / 000F'  absolute ------------       JP   Z,F9.0
95  .......1 1001010.  \           CA
    .......1            / 0010'  code-relative offset
5B  01......           |
80  ..011011 10......  |           006E'
07  000000 00......     \
    ..0.....            / 0012'  absolute ------------       DEC  A
AC  ...00111 101.....  \           3D
    ...0....            / 0013'  absolute ------------       JP   Z,F2.0
AA  ....1100 1010....  \           CA
    ....1...            / 0014'  code-relative offset
    .....01.           |
78  .......0 0111100.  |           003C'
00  .......0 0000000.   \
    .......0            / 0016'  absolute ------------       RET
C9  11001001            \           C9
6A  0.......            / 0017'  absolute ------------ IN.0: PUSH DE
B1  .1101010 1.......  \           D5
    .0......            / 0018'  absolute ------------       PUSH BC
47  ..110001 01......  \           C5
    ..0.....            / 0019'  absolute ------------       LD   A,7B
C7  ...00111 110.....  \           3E
    ...0....            / 001A'  absolute
B1  ....0111 1011....  \           7B
    ....0...            / 001B'  absolute ------------       LD   (DR.0),A
95  .....001 10010...  \           32
    .....1..            / 001C'  code-relative offset
    ......01           |
00  00000000           |           0000'
00  00000000            \
1F  0.......            / 001E'  absolute ------------       LD   A,0D6
35  .0011111 0.......  \           3E
    .0......            / 001F'  absolute
86  ..110101 10......  \           D6
    ..0.....            / 0020'  absolute ------------       LD   (DR.0+1),A
54  ...00110 010.....  \           32
    ...1....            / 0021'  code-relative offset
    ....01..           |
04  ......00 000001..  |           0001'
00  ......00 000000..   \
    ......0.            / 0023'  absolute ------------       LD   A,2
7C  .......0 0111110.  \           3E
    .......0            / 0024'  absolute
02  00000010            \           02
19  0.......            / 0025'  absolute ------------       LD   (DR.0+2),A
50  .0011001 0.......  \           32
    .1......            / 0026'  code-relative offset
    ..01....           |
20  ....0000 0010....  |           0002'
05  ....0000 0000....   \
    ....0...            / 0028'  absolute ------------       XOR  A
```

```
78  .....101 01111...   \           AF
    .....0..             / 0029'  absolute ------------       LD   (F2.1),A
CA  ......00 110010..    \           32
    ......1.             / 002A'  code-relative offset
A5  .......0 1.......    |
80  .0100101 1.......    |           004B'
0E  .0000000 0.......    \
    .0......             / 002C'  absolute ------------       LD   A,(CONBR)
B0  ..001110 10......    \           3A
    ..1.....             / 002D'  data-relative offset
    ...10...             |
00  .....000 00000...    |           0000"
01  .....000 00000...    \
    .....0..             / 002F'  absolute ------------       LD   C,A
3C  ......01 001111..    \           4F
    ......0.             / 0030'  absolute ------------       LD   E,03
3C  .......0 0011110.    \           1E
    .......0             / 0031'  absolute
03  00000011             \           03
66  0.......             / 0032'  absolute ------------       CALL SERIAL##
D0  .1100110 1.......    \           CD
    .1......             / 0033'  code-relative offset
    ..01....             |
40  ....0000 0100....    |           0004'
06  ....0000 0000....    \
    ....0...             / 0035'  absolute ------------       CALL F2.3
6D  .....110 01101...    \           CD
    .....1..             / 0036'  code-relative offset
    ......01             |
57  01010111             |           0057'
00  00000000             \
60  0.......             / 0038'  absolute ------------       POP  BC
B4  .1100000 1.......    \           C1
    .0......             / 0039'  absolute ------------       POP  DE
43  ..110100 01......    \           D1
    ..0.....             / 003A'  absolute ------------       JR   DR.O
0C  ...00011 000.....    \           18
    ...0....             / 003B'  absolute
41  ....1100 0100....    \           C4
    ....0...             / 003C'  absolute ------------ F2.0: LD   A,(CONOFF)
D6  .....001 11010...    \           3A
    .....1..             / 003D'  data-relative offset
    ......10             |
32  00110010             |           0032"
00  00000000             \
5B  0.......             / 003F'  absolute ------------       OR   A
8A  .1011011 1.......    \           B7
    .0......             / 0040'  absolute ------------       JR   Z,F2.2
01  ..001010 00......    \           28
    ..0.....             / 0041'  absolute
4B  ...00001 010.....    \           0A
    ...0....             / 0042'  absolute ------------       CP   C
91  ....1011 1001....    \           B9
    ....0...             / 0043'  absolute ------------       JR   Z,F2.4
40  .....001 01000...    \           28
    .....0..             / 0044'  absolute
60  ......00 011000..    \           18
    ......0.             / 0045'  absolute ------------       LD   A,(CONON)
75  .......0 0111010.    \           3A
    .......1             / 0046'  data-relative offset
8C  10......             |
```

```
C0  ..001100 11......  |           0033"
17  ..000000 00......   \
    ..0.....            / 0048'  absolute ------------       CP   C
22  ...10111 001.....   \           B9
    ...0....            / 0049'  absolute ------------       JR   Z,F2.5
80  ....0010 1000....   \           28
    ....0...            / 004A'  absolute
C0  .....000 11000...   \           18
    .....0..            / 004B'  absolute ------------ F2.1: NOP
00  ......00 000000..   \           00
    ......0.            / 004C'  absolute ------------ F2.2  LD   A,(CONFF)
75  .......0 0111010.   \           3A
    .......1            / 004D'  data-relative offset
80  10......            |
40  ..000000 01......   |        0001"
16  ..000000 00......   \
    ..0.....            / 004F'  absolute ------------       OR   A
EC  ...10110 111.....   \           B7
    ...0....            / 0050'  absolute ------------       JP   Z,SERIAL##
AA  ....1100 1010....   \           CA
    ....1...            / 0051'  code-relative offset
    .....01.            |
66  .......0 0110011.   |           0033'
00  .......0 0000000.    \
    .......0            / 0053'  absolute ------------       CP   C
B9  10111001            \           B9
61  0.......            / 0054'  absolute ------------       JP   NZ,SERIAL##
55  .1100001 0.......   \           C2
    .1......            / 0055'  code-relative offset
    ..01....            |
10  ....0101 0001....   |           0051'
01  ....0000 0000....   \
    ....0...            / 0057'  absolute ------------ F2.3: LD   HL,CONCLS
0E  .....001 00001...   \           21
    .....1..            / 0058'  data-relative offset
    ......10            |
02  00000010            |           0002"
00  00000000            \
61  0.......            / 005A'  absolute ------------       JP   DMSHL##
80  .1100001 1.......   \           C3
    .0......            / 005B'  absolute
00  ..000000 00......   \           00
    ..0.....            / 005C'  absolute
03  ...00000 000.....   \           00
    ...0....            / 005D'  absolute ------------ F2.4: LD   A,0C9
E6  ....0011 1110....   \           3E
    ....0...            / 005E'  absolute
48  .....110 01001...   \           C9
    .....0..            / 005F'  absolute ------------       LD   (F2.1),A
CA  ......00 110010..   \           32
    ......1.            / 0060'  code-relative offset
A5  .......0 1.......   |
80  .0100101 1.......   |           004B'
32  .0000000 0.......    \
    .0......            / 0062'  absolute ------------       RET
55  ..110010 01......   \           C9
    ..0.....            / 0063'  absolute ------------ F2.5: XOR  A
E3  ...10101 111.....   \           AF
    ...0....            / 0064'  absolute ------------       LD   (F2.1),A
2A  ....0011 0010....   \           32
    ....1...            / 0065'  code-relative offset
```

```
    .....01.           |
96  .......0 1001011. |           004B'
00  .......0 0000000.  \
    .......0           / 0067'  absolute -------------       RET
C9  11001001           \           C9
10  0.......           / 0068'  absolute ------------- F8.0: LD   HL,CONSOS
E1  .0010000 1.......  \           21
    .1......           / 0069'  data-relative offset
    ..10....           |
20  ....0001 0010....  |           0012"
06  ....0000 0000....  \
    ....0...           / 006B'  absolute -------------       JP   DMSHL##
1D  .....110 00011...  \           C3
    .....1..           / 006C'  code-relative offset
    ......01           |
5B  01011011           |           005B'
00  00000000           \
10  0.......           / 006E'  absolute ------------- F9.0: LD   HL,CONSIS
E2  .0010000 1.......  \           21
    .1......           / 006F'  data-relative offset
    ..10....           |
20  ....0010 0010....  |           0022"
06  ....0000 0000....  \
    ....0...           / 0071'  absolute -------------       JP   DMSHL##
1D  .....110 00011...  \           C3
    .....1..           / 0072'  code-relative offset
    ......01           |
6C  01101100           |           006C'
00  00000000           \
8F  1.......           / 0072'  Define Entry Point
    .00.....           |
    ...0111.           |
00  .......1 0.......  |           data-relative
00  .0000000 0.......  |              0000"
54  .0000000 0.......  |
    .101....           |           5 chrs
34  ....0100 0011....  |              "C"
F4  ....0100 1111....  |              "O"
E4  ....0100 1110....  |              "N"
25  ....0100 0010....  |              "B"
28  ....0101 0010....  \              "R"
    ....1...           / 0072'  Define Entry Point
    .....00.           |
F0  .......0 111.....  |
    ...10...           |           data-relative
10  .....000 00010...  |              0002"
06  .....000 00000...  |
    .....110           |           6 chrs
43  01000011           |              "C"
4F  01001111           |              "O"
4E  01001110           |              "N"
43  01000011           |              "C"
4C  01001100           |              "L"
53  01010011           \              "S"
8E  1.......           / 0072'  Define Entry Point
    .00.....           |
    ...0111.           |
80  .......0 1.......  |           code-relative
00  .0000000 0.......  |              0000'
64  .0000000 0.......  |
    .110....           |           6 chrs
```

```
34  ....0100 0011....  |               "C"
F4  ....0100 1111....  |               "O"
E4  ....0100 1110....  |               "N"
45  ....0100 0100....  |               "D"
24  ....0101 0010....  |               "R"
08  ....0100 0000....   \              "@"
    ....1...            / 0072'  Define Entry Point
    .....00.           |
F0  .......0 111.....  |
    ...10...           |           data-relative
08  .....000 00001...  |              0001
05  .....000 00000...  |
    .....101           |           5 chrs
43  01000011           |              "C"
4F  01001111           |              "O"
4E  01001110           |              "N"
46  01000110           |              "F"
46  01000110            \             "F"
8F  1.......            / 0072'  Define Entry Point
    .00.....           |
    ...0111.           |
19  .......1 0.......  |           data-relative
00  .0011001 0.......  |              0032"
64  .0000000 0.......  |
    .110....           |           6 chrs
34  ....0100 0011....  |              "C"
F4  ....0100 1111....  |              "O"
E4  ....0100 1110....  |              "N"
F4  ....0100 1111....  |              "O"
64  ....0100 0110....  |              "F"
68  ....0100 0110....   \             "F"
    ....1...            / 0072'  Define Entry Point
    .....00.           |
F1  .......0 111.....  |
    ...10...           |           data-relative
98  .....001 10011...  |              0033"
05  .....000 00000...  |
    .....101           |           5 chrs
43  01000011           |              "C"
4F  01001111           |              "O"
4E  01001110           |              "N"
4F  01001111           |              "O"
4E  01001110            \             "N"
8F  1.......            / 0072'  Define Entry Point
    .00.....           |
    ...0111.           |
11  .......1 0.......  |           data-relative
00  .0010001 0.......  |              0022"
64  .0000000 0.......  |
    .110....           |           6 chrs
34  ....0100 0011....  |              "C"
F4  ....0100 1111....  |              "O"
E5  ....0100 1110....  |              "N"
34  ....0101 0011....  |              "S"
95  ....0100 1001....  |              "I"
38  ....0101 0011....   \             "S"
    ....1...            / 0072'  Define Entry Point
    .....00.           |
F0  .......0 111.....  |
    ...10...           |           data-relative
90  .....000 10010...  |              0012"
```

```
06  .....000 00000...  |
    .....110            |            6 chrs
43  01000011            |               "C"
4F  01001111            |               "O"
4E  01001110            |               "N"
53  01010011            |               "S"
4F  01001111            |               "O"
53  01010011             \              "S"
8C  1.......             / 0072'  Chain External
    .00.....            |
    ...0110.            |
B9  .......0 1.......   |            code-relative
00  .0111001 0.......   |               0072'
54  .0000000 0.......   |
    .101....            |            5 chrs
44  ....0100 0100....   |               "D"
D5  ....0100 1101....   |               "M"
34  ....0101 0011....   |               "S"
84  ....0100 1000....   |               "H"
C8  ....0100 1100....    \              "L"
    ....1...             / 0072'  Define Entry Point
    .....00.            |
F0  .......0 111.....   |
    ...10...            |            data-relative
08  .....000 00001...   |               0001"
05  .....000 00000...   |
    .....101            |            5 chrs
46  01000110            |               "F"
46  01000110            |               "F"
43  01000011            |               "C"
48  01001000            |               "H"
52  01010010             \              "R"
8C  1.......             / 0072'  Chain External
    .00.....            |
    ...0110.            |
AA  .......0 1.......   |            code-relative
80  .0101010 1.......   |               0055'
65  .0000000 0.......   |
    .110....            |            6 chrs
34  ....0101 0011....   |               "S"
55  ....0100 0101....   |               "E"
24  ....0101 0010....   |               "R"
94  ....0100 1001....   |               "I"
14  ....0100 0001....   |               "A"
C9  ....0100 1100....    \              "L"
    ....1...             / 0072'  End module -----------       END
    .....00.            |
C0  .......1 110.....   |
    ...00...            |            absolute
00  .....000 00000...   |               0000
00  .....000 00000...    \
    000.....            -
9E  1.......             / 0072'  End of file
    .00.....            |
    ...1111.             \
    .......0            -
```

Microsoft Link-80 Linker

Description

Microsoft Link-80 (L80) is a comprehensive linking loader designed especially for use with Microsoft's family of relocatable language compilers/assemblers: FORTRAN-80, COBOL-80, BASCOM, and Macro-80. The purpose of Link-80 is to resolve all addresses and data references in a standard Microsoft REL file into executable code, usually a COM (command) or SYS (system) file. All commands and utilities used under the TurboDOS operating system are COM files, while the operating system itself is a SYS file. SYS files are difficult to create using Link-80, though it is possible, and should not be attempted under normal circumstances: TurboDOS' GEN linking loader should be used instead.

"LOADING" is the process of placing into memory a REL file and rationalizing every internal relocatable address and data reference in the file into an absolute address relative to the beginning of the file.

"LINKING" is the process of rationalizing all address and data references that are external to a given REL file with the corresponding global addresses and data references in another REL file being simultaneously loaded.

After loading and linking, the resultant object file may be saved and/or executed.

Commands

The basic command structure for Link-80 is as follows:

```
L80 commandlist
```

or

```
L80
* command
* command
...
* command
* /E or attention-abort
```

The individual commands in a Link-80 operation consist of either a file reference (filename) or a switch.

Usage

In normal operation, Link-80, unless specifically instructed otherwise, loads a file's data segment first at the default address of 0103h, followed by any COMMON segments, and then followed by the code segment. Proper operation is maintained by placing a "routing jump" at 0100h, which cause execution to jump around the data and COMMON segments to the code segment. Several potential problems arise concerning this routing jump. TurboDOS's GEN linker eliminates these problems by always placing the code segment first at 0100h, followed by the data and COMMON segments.

The following sample source code is typical of TurboDOS (but not CP/M) code:

```
;HELLO.MAC
;
        NAME    ('HELLO')
;
        .Z80                    ;Use Zilog mnemonics
;
CR      EQU     13              ;ASCII carriage return
LF      EQU     10              ;ASCII linefeed
;
        DSEG                    ;Locate in data segment
;
MSG:    DB      CR,LF,LF
        DB      'Hello, this is a test message.'
        DB      CR,LF,LF,'$'
;
        CSEG                    ;Locate in code segment
```

```
;
        LD      DE,MSG          ;Output the test message to the console
        LD      C,9
        JP      0005H
;
        END
```

Ultimately, the above source code would be used to create a command file of type COM which, when executed, would cause the message “Hello, this is a test file.” to appear on the console. The heart of this operation is the three lines of code between the CSEG and the END pseudo-operations. These three lines should become the C-function (BDOS) operation that causes a string to be passed to the console, exactly as desired.

When TurboDOS' GEN command is used, the resultant COM file code (as seen through a debugger) would be:

```
0100    LD      DE,0108
0103    LD      C,09
0105    JP      0005
0108    message begins here
...
012C    message ends here
```

This code, like all COM file code, would begin execution at 0100h and would perform exactly as desired.

Unfortunately, if the exact same source code is used with Link-80 in the standard manner for CP/M:

```
L80 HELLO,HELLO/N/E
```

the following code would result:

```
0100    NOP
0101    NOP
0102    NOP
0103    message begins here
...
0127    message ends here
0128    LD      DE,0103
012B    LD      C,09
012D    JP      0005
```

This code would also begin execution at 0100h, but would not produce the desired results: the bytes of the message would attempt to execute as though they were code! Chaos would result!

The solution to this problem is simple. If the source code is available, a slight modification would correct the situation:

```
;HELLO.MAC
;
        NAME ('HELLO')
;
        .Z80                    ;Use Zilog mnemonics
;
CR      EQU     13              ;ASCII carriage return
LF      EQU     10              ;ASCII linefeed
;
        DSEG                    ;Locate in data segment
;
MSG:    DB      CR,LF,LF
        DB      'Hello, this is a test message.'
        DB      CR,LF,LF,'$'
;
        CSEG                    ;Locate in code segment
;
BEGIN:  LD      DE,MSG          ;Output the test message to the console
        LD      C,9
        JP      0005H
;
        END     BEGIN
```

The addition of the label “BEGIN” and the reference to the label in the END pseudo-operation causes the generation of a proper routing jump:

```
0100    JP      0128
0103    message begins here
...
0127    message ends here
0128    LD      DE,0103
012B    LD      C,09
012D    JP      0005
```

In a similar manner, a slightly different modification:

```
;HELLO.MAC
;
        NAME ('HELLO')
;
        .Z80                    ;Use Zilog mnemonics
;
CR      EQU     13              ;ASCII carriage return
LF      EQU     10              ;ASCII linefeed
;
        DSEG                    ;Locate in data segment
;
MSG:    DB      CR,LF,LF
        DB      'Hello, this is a test message.'
        DB      CR,LF,LF,'$'
;
        CSEG                    ;Locate in code segment
;
BEGIN:: LD      DE,MSG          ;Output the test message to the console
        LD      C,9
        JP      0005H
;
        END
```

coupled with a slight command variation:

```
L80 HELLO,HELLO/N/E:BEGIN
```

has the exact same effect and produces a proper routing jump.

A third solution involves the somewhat more drastic modification of making the source code “segmentless,” with neither CSEG nor DSEG, nor COMMON specified, and with the code already in the exact order required by the COM file:

```
;HELLO.MAC
;
        NAME ('HELLO')
;
        .Z80                    ;Use Zilog mnemonics
;
CR      EQU     13              ;ASCII carriage return
LF      EQU     10              ;ASCII linefeed
;
        LD      DE,MSG          ;Output the test message to the console
        LD      C,9
        JP      0005H
;
MSG:    DB      CR,LF,LF
        DB      'Hello, this is a test message.'
        DB      CR,LF,LF,'$'
;
        END
```

When used with the normal Link-80 command:

```
L80 HELLO,HELLO/N/E
```

this approach produces the following code:

```
0100    NOP
0101    NOP
0103    NOP
0104    LD      DE,0108
0107    LD      C,09
0109    JP      0005
010C    message begins here
...
012F    message ends here
```

In this manner, the entire file is placed in the default segment (data) and loaded at 0103h, without a routing jump. The three NOP's at the beginning do nothing, and the remainder is exactly as desired.

If the source code is unavailable (only the REL file is present), the routing jump problem can still be conquered via some creative loading. First, load the file so the code segment is at 0100h and the data segment is obviously well beyond the end of the code segment:

```
L80 /P:0100/D:1000,HELLO,HELLO/N/E
```

```
Data      1000
Program   0100          ←look here!
```

Link-80 faithfully reports that the first byte following the code (program) segment is 0108. Reload the file with the code segment at 0100h and the data segment immediately following:

```
L80 /P:0100/D:0108,HELLO,HELLO/N/E
```

This method produces the following code:

```
0100    LD      DE,0108
0103    LD      C,09
0105    JP      0005
0108    message begins here
...
012C    message ends here
```

which is identical with that produced by TurboDOS' GEN linker.

Obviously, creating source code with proper labels and END pseudo-ops is the optimum way to proceed when Link-80 is to be used.

Filenames

The term “filename” and its variations used with regard to Link-80 may be taken to mean a full file reference:

```
{d:}filename{.typ}
```

where the braces indicate optional. The filename may have a drive designation but may not have a user designation. Link-80 is a CP/M program and cannot cross user boundaries in its searches.

The minimal (and normal) Link-80 command list is:

```
L80 relfil,outfil/N/E
```

where “relfile” is the name of the REL file to be loaded and “outfile” is the name of the COM file to be created. The two filenames must be separated by a comma.

If more than one REL file is to be linked, the minimal command list becomes:

```
L80 relfile1,relfile2,relfil3,...,relfilen,outfile/N/E
```

The filenames must be separated by commas, and the first relfile encountered should be the “main module.”

All relfiles are assumed to be of type REL unless another filetype is specified. Even then, the file must be a type-REL file in format and structure though it has a different filetype. Only REL-format files are acceptable for linking and loading.

The output file “outfile” is assumed to be of type COM unless another filetype is designated.

There are exceptions to this: it is the /N switch that causes a COM or COM-like file to be created. If the /X switch is used instead, the output file is presumed to be an Intel HEX file; if the /Y switch is used, a ZSID SYM file.

The most often-committed error when using Link-80 is fail to specify at least two filenames, the relfile and the outfile, even though the two names are identical. Attempting to execute Link-80 with only one filename is at best a waste of time: either there is no relfile loaded or no output file created.

Switches

Link-80 uses a series of switches to perform operation upon the filename specified in the command list. Do not confuse Link-80 switches with Macro-80 switches: they are completely unrelated.

/D:addr Set Data Segment Address

The /D:addr switch sets the origin for the data segment (actually the segment defined by DSEG, followed by any segments defined by COMMON), as distinct from the code (program) segment (the segment defined by CSEG). If the /D:addr switch is not entered, Link-80 automatically assigns both the data and code segments to the address defined by the /P:addr switch or, if there is no /P:addr switch, to 0103h. “addr” must be in the current radix: the default radix is hexadecimal.

The /D:addr switch takes effect as soon as it is encountered and has no effect on file already loaded. It is important that the /D:addr switch be placed before the file(s) it is to affect.

The /P:addr and /D:addr switches may be concatenated, but must be separated from a following filename by a comma:

```
L80 /P:0100/D:0108,HELLO,HELLO/N/E
```

You may enter more than one /D:addr switch in a given command list. This allows the placement of data and code segments at addresses that are not end-to-end. This multiple use is restricted as follows:

1. One data or code segment may not overlay another.
2. Data segments may not be separated by a code segment.
3. Data or code segments that are not contiguous may have garbage data present in the gaps between segments. This garbage data is whatever happens to be in memory at link time, and may or may not cause operational problems, depending upon code structure and logic.

It may be possible to load and link a file or series of files that are technically too big for memory. By using both the /P:addr and /D:addr switches the need is eliminated for Link-80 to build a relocation address table in memory which, with large numbers of relocation address, can be a significant memory savings.

To determine the proper load addresses for this memory-saving possibility, determine the size of the data segment (the combined sizes of segments designated by DSEG and COMMON) from a PRN file, add 0204h (0103h for the data segment base and 0101h for the code segment offset), and the result is the address for the /P:addr switch: the /D:addr switch should be /D:0103.

See the /P:addr switch.

/E Exit

The /E switch causes Link-80 to exit to the operating system when the linking/loading session is finished.

When linking and loading is finished, Link-80 displays certain information:

```
Data      ssss    nnnn    < llll>
Program   ssss    nnnn    < llll>

fffff Bytes Free
[bbbb   eeee   rrrr]
```

where:

ssss = segment absolute start address
nnnn = segment absolute end address + 1 (next byte)
llll = segment length
fffff = number of byte of TPA above program (decimal)
bbbb = program relative start address (usually 0000h)
eeee = program absolute end address + 1
rrrr = number of 128-byte records used to save the program

See the /E:name, /G, and /G:name switches.

/E:name Set Start Address and Exit

The /E:name switch sets the routing jump to the address of the global label “name” and then causes Link-80 to exit to the operating system when the linking/loading session is finished. “name” must be defined in one of the REL files loaded.

See the /E, /G, and /G:name switches.

/G Execute and Exit

The /G switch causes Link-80 to execute the loaded and linked program from memory and then exit to the operating system when the linking/loading session is finished.

See the /E, /E:name, and /G:name switches.

/G:name Set Start Address, Execute and Exit

The /G:name switch sets the routing jump to the address of the global label “name”, then causes Link-80 to execute the loaded and linked program from memory, and then exit to the operating system when the linking/loading session is finished. “name” must be defined in one of the REL files loaded.

See the /E, /E:name, and /G switches.

/H Set Radix to Hexadecimal

The /H switch sets or resets the current radix to hexadecimal. The default radix is hexadecimal.

See the /O switch.

/M Display All Globals

The /M switch directs Link-80 to display all globals, both defined and undefined, on the console screen. Defined globals are followed by their addresses in the current radix. Undefined globals are followed by an asterisk.

See the /U switch.

/N Save Output File as COM File

The /N switch causes the filename immediately prior to it to be the name of the output file to be created. The output file is assumed to be of type COM, or COM-like in structure with a different (specified) filetype, unless either the /X or /Y switch is also present.

The /N switch has no effect until a /E, /E:name, /G, or /G:name switch is encountered.

The single most common error in using Link-80 is a failure to specify at least two filenames, even if they are alike. In the following command:

```
L80 TEST,TEST/N/E
```

the input is the file TEST.REL and the output is the file TEST.COM.

It is the switches and not the order that determine which is the input and which is the output file. The following two commands are identical:

```
L80 TSTIN,TSTOUT/N/E

L80 TSTOUT/N,TSTIN/E
```

Both have the input file TSTIN.REL and the output file TSTOUT.COM.

See the /E, /E:name, /G, /G:name, /N:P, /X, and /Y switches.

/N:P Save Code Segment of COM File

The /N: switch causes Link-80 to save the code (program) segment only into the file whose name is immediately prior to the switch. By saving the code segment only, a small COM file can be created for a program with a large but undefined data area, such as a big buffer defined and initialized on-line during execution.

The output file is assumed to be of type COM, or COM-like in structure with a different (specified) filetype, unless either the /X or /Y switch is also present.

The /N:P switch has no effect until a /E, /E:name, /G, or /G:name switch is encountered.

See the /E, /E:name, /G, /G:name, /N, /X, and /Y switches.

/O Set Radix to Octal

The /O switch sets or resets the current radix to octal. The default radix is hexadecimal.

See the /H switch.

/P:addr Set Code Segment Address

The /P:addr switch sets the origin for both the data segment (actually the segment defined by DSEG, followed by any segments defined by COMMON) and the code (program) segment (the segment defined by CSEG). If the /P:addr switch is not entered, Link-80 automatically assigns both the data and code segments to 0103h. “addr” must be in the current radix: the default radix is hexadecimal.

The /P:addr switch takes effect as soon as it is encountered and has no effect on file already loaded. It is important that the /P:addr switch be placed before the file(s) it is to affect.

The /P:addr and /D:addr switches may be concatenated, but must be separated from a following filename by a comma:

```
L80 /P:0100/D:0108,HELLO,HELLO/N/E
```

You may enter more than one /P:addr switch in a given command list. This allows the placement of data and code segments at addresses that are not end-to-end. This multiple use is restricted as follows:

1. One data or code segment may not overlay another.
2. Code segments may not be separated by a data segment.
3. Data or code segments that are not contiguous may have garbage data present in the gaps between segments. This garbage data is whatever happens to be in memory at link time, and may or may not cause operational problems, depending upon code structure and logic.

See the /D:addr switch.

/R Reset Link-80

The /R switch causes Link-80 to cancel all previous loading and linking and restore it to initial condition.

/S Perform Library Search

The /S switch causes Link-80 to search the library file immediately preceding it for routines, subroutines, global definitions, etc.

In a command line, the filename with the /S switch must be separated from the rest of the command list by commas:

```
L80 TEST/N,TSTLIB/S,TEST/G
```

The /S switch suppresses the display of undefined globals. The display may be restored via the /U switch.

See the /U switch.

/U Display Undefined Globals

The /U switch directs Link-80 to display all undefined globals on the console screen. Undefined globals are followed by an asterisk.

The /U switch operates only in command lists that contain neither a /E, /E:name, /G, nor /G:name switch: this switch is primarily for use in conjunction with the /S switch, which normally suppresses the display of undefined globals (the display of undefined globals is normally automatic).

See the /E, /E:name, /G, /G:name, /M, and /S switches.

/X Save HEX file

The /X switch, when used in conjunction with the /N switch, causes the filename immediately prior to it to be the name of the output file to be created as an Intel HEX file. Intel HEX files are primarily used as code for EPROM programmers and similar devices.

The /N/X switch pair has no effect until a /E or /E:name switch is encountered.

See the /E, /E:name, /N, and /N:P switches.

/Y Save SYM file

The /Y switch, when used in conjunction with the /N switch, causes the filename immediately prior to it to be the name of the output file to be created as a special SYM symbol-table file for use with Digital Research's SID or ZSID debuggers.

The /N/Y switch pair has no effect until a /E or /E:name switch is encountered.

See the /E, /E:name, /N, and /N:P switches.

Error Messages

%2nd COMMON Larger /XXXXXX/
The first definition of a specific COMMON segment encountered was not the largest definition of that COMMON. Reorder the modules so the largest definition is first or redefine the COMMON segments involved.

%Intersecting Data Area
The data areas intersect and an external chain entry is in this intersection, preventing the final value from being converted to a current value.

%Intersecting Program Area
The code (program) areas intersect and an external chain entry is in this intersection, preventing the final value from being converted to a current value.

%Mult. Def. Global YYYYYY
The global symbol “YYYYYY” has been defined in more than one linked module.

%Overlaying Data Area
The /P:addr and /D:addr switches were set with addresses too close together, causing the data segment to be overlain.

%Overlaying Program Area
The /P:addr switch is set to an area already filled or an area reserved by Link-80, causing the code (program) segment to be overlain.

?<filename> Not Found
The REL format file <filename> does not exist, is not on the current user area, is locked by another process, or otherwise cannot be found.

?Can't Save Object File
The directory is full, the disk is full, or the object file already exists as read only or is locked by another process.

?Command Error
An illegal command was issued.

?Loading Error
The last file loaded was not a standard Microsoft REL format file.

?No Start Address
The /G or /G:name switch was issued but no main program has been loaded.

?Nothing Loaded
The /E, /E:name, /G, /G:name, or /S switch was encountered but nothing has been loaded.

?Out of Memory
Not enough memory available to load the current module/file.

?Start Symbol—<name>—Undefined
An /E:name or /G:name switch was encountered but the global symbol “name” has not been defined.

Origin Above Loader Memory
An /E, /E:name, /G, or /G:name was encountered and the program requirements or a /D:addr or /P:addr switch exceed the upper limit of Link-80's memory space.

This error prompts the operator with a “Move Anyway (Y or N)?” message, if “Y” is entered Link-80 attempts to move the program and proceed. If the /N switch is present, the output file is saved regardless of operator response.

Origin Below Loader Memory
An /E, /E:name, /G, or /G:name was encountered and a /D:addr or /P:addr switch is set below the minimum address of 0100h.

This error prompts the operator with a “Move Anyway (Y or N)?”

message, if “Y” is entered Link-80 attempts to move the program and proceed. If the /N switch is present, the output file is saved regardless of operator response.

Microsoft Cross-Reference Utility

Description

The Microsoft cross reference utility, Cref-80 (CREF80), is designed to process a special assembly file into a cross reference listing file. The special assembly file is created by Macro-80 when its /C switch is used.

The cross reference listing file has the type PRN, and is similar to the PRN listing file created by Macro-80 with two additional features:

1. Each source statement is numbered with a cross reference line number.
2. At the end of the listing, variable names appear in alphanumeric order, with each name followed by the numbers of every line where it is referenced or defined (the line where it is defined is flagged by a "#").

Creation

A cross reference listing file is created in two operations. The first operation is the creation of the CRF file via Macro-80:

```
M80 =CONDRV/C
```

The second is the creation of the cross reference listing PRN file from the CRF file via Cref-80:

```
CREF80 =CONDRV
```

The exact format of a Cref-80 command is:

```
CREF80 {o:}={d:}filename{.typ}
```

where:

{o:}	= the drive of the PRN file if not the current drive
{d:}	= the drive of the CRF file if not the current drive
filename	= the name of the CRF file
{.typ}	= the type of the CRF file if not CRF

Confusion often arises between the PRN files created directly with Macro-80 and the PRN files created via Cref-80. They are indeed very similar, but those created with Cref-80 contain the additional cross reference information and provide a superior troubleshooting and debugging tool.

Control Directives

Occasionally, it may be desirable to create a cross reference listing for all or part of a file on an ongoing basis. This may be done via Macro-80's .CREF and .XCREF assembler directives. these directives are covered in detail in the Macro-80 chapter.

Sample Cross Reference Listing File

A sample file based upon the code sampled in the Macro-80 chapter follows (again, comments have been removed to shorten code), compare this file with the PRN listing file:

```
 1                                      NAME    ('CONDRV')
 2                                      .Z80
 3                              STRING  MACRO   A,B,C,D
 4                                      IFNB    <D>
 5                                      DB      A,B,C,D+128
 6                                      ELSE
 7                                      IFNB    <C>
 8                                      DB      A,B,C+128,128
 9                                      ELSE
10                                      IFNB    <B>
11                                      DB      A,B+128,128,128
```

```
12                                              ELSE
13                                              IFNB    <A>
14                                              DB      A+128,128,128,128
15                                              ELSE
16                                              DB      128,128,128,128
17                                              ENDIF
18                                              ENDIF
19                                              ENDIF
20                                              ENDIF
21                                              DB      128,128,128,128
22                                              DB      128,128,128,128
23                                              DB      128,128,128,128
24                                              ENDM
25  0000'                                       DSEG
26  0000"   CF                  CONBR:: DB      0CFH
27  0001"                       FFCHR::
28  0001"   0C                  CONFF:: DB      12
29  0002"                       CONCLS::STRING  26
30  0002"   9A 80 80 80    +          DB      26+128,128,128,128
31  0006"   80 80 80 80    +          DB      128,128,128,128
32  000A"   80 80 80 80    +          DB      128,128,128,128
33  000E"   80 80 80 80    +          DB      128,128,128,128
34  0012"                       CONSOS::STRING  13,10
35  0012"   0D 8A 80 80    +          DB      13,10+128,128,128
36  0016"   80 80 80 80    +          DB      128,128,128,128
37  001A"   80 80 80 80    +          DB      128,128,128,128
38  001E"   80 80 80 80    +          DB      128,128,128,128
39  0022"                       CONSIS::STRING  13,10
40  0022"   0D 8A 80 80    +          DB      13,10+128,128,128
41  0026"   80 80 80 80    +          DB      128,128,128,128
42  002A"   80 80 80 80    +          DB      128,128,128,128
43  002E"   80 80 80 80    +          DB      128,128,128,128
44  0032"   00                  CONOFF::DB      0
45  0033"   00                  CONON:: DB      0
46  0034"                               CSEG
47  0000'                       CONDR@::
48  0000'   C3 0017'            DR.0:   JP      IN.0
49  0003'   DA 0000*                    JP      C,SERIAL##
50  0006'   CA 003C'                    JP      Z,F2.0
51  0009'   D6 06                       SUB     6
52  000B'   CA 0068'                    JP      Z,F8.0
53  000E'   3D                          DEC     A
54  000F'   CA 006E'                    JP      Z,F9.0
55  0012'   3D                          DEC     A
56  0013'   CA 003C'                    JP      Z,F2.0
57  0016'   C9                          RET
58  0017'   D5                  IN.0:   PUSH    DE
59  0018'   C5                          PUSH    BC
60  0019'   3E 7B                       LD      A,7BH
61  001B'   32 0000'                    LD      (DR.0),A
62  001E'   3E D6                       LD      A,0D6H
63  0020'   32 0001'                    LD      (DR.0+1),A
64  0023'   3E 02                       LD      A,2
65  0025'   32 0002'                    LD      (DR.0+2),A
66  0028'   AF                          XOR     A
67  0029'   32 004B'                    LD      (F2.1),A
68  002C'   3A 0000"                    LD      A,(CONBR)
69  002F'   4F                          LD      C,A
70  0030'   1E 03                       LD      E,03
71  0032'   CD 0000*                    CALL    SERIAL##
72  0035'   CD 0057'                    CALL    F2.3
```

```
   73  0038'   C1                          POP     BC
   74  0039'   D1                          POP     DE
   75  003A'   18 C4                       JR      DR.0
   76  003C'   3A 0032"           F2.0:    LD      A,(CONOFF)
   77  003F'   B7                          OR      A
   78  0040'   28 0A                       JR      Z,F2.2
   79  0042'   B9                          CP      C
   80  0043'   28 18                       JR      Z,F2.4
   81  0045'   3A 0033"                    LD      A,(CONON)
   82  0048'   B9                          CP      C
   83  0049'   28 18                       JR      Z,F2.5
   84  004B'   00                  F2.1:   NOP
   85  004C'   3A 0001"           F2.2:    LD      A,(CONFF)
   86  004F'   B7                          OR      A
   87  0050'   CA 0000*                    JP      Z,SERIAL##
   88  0053'   B9                          CP      C
   89  0054'   C2 0000*                    JP      NZ,SERIAL##
   90  0057'   21 0002"           F2.3:    LD      HL,CONCLS
   91  005A'   C3 0000*                    JP      DMSHL##
   92  005D'   3E C9               F2.4:    LD      A,0C9H
   93  005F'   32 004B'                    LD      (F2.1),A
   94  0062'   C9                          RET
   95  0063'   AF                  F2.5:    XOR     A
   96  0064'   32 004B'                    LD      (F2.1),A
   97  0067'   C9                          RET
   98  0068'   21 0012"           F8.0:    LD      HL,CONSOS
   99  006B'   C3 0000*                    JP      DMSHL##
  100  006E'   21 0022"           F9.0:    LD      HL,CONSIS
  101  0071'   C3 0000*                    JP      DMSHL##
  102                                      END

Macros:
STRING

Symbols:
0000I"   CONBR       0002I"   CONCLS      0000I'   CONDR@
0001I"   CONFF       0032I"   CONOFF      0033I"   CONON
0022I"   CONSIS      0012I"   CONSOS      0072*    DMSHL
0000'    DR.0        003C'    F2.0        004B'    F2.1
004C'    F2.2        0057'    F2.3        005D'    F2.4
0063'    F2.5        0068'    F8.0        006E'    F9.0
0001I"   FFCHR       0017'    IN.0        0055*    SERIAL

CONBR    26#      68
CONCLS   29#      90
CONDR@   47#
CONFF    28#      85
CONOFF   44#      76
CONON    45#      81
CONSIS   39#      100
CONSOS   34#      98
DMSHL    91       99       101
DR.0     48#      61       63       65       75
F2.0     50       56       76#
F2.1     67       84#      93       96
F2.2     78       85#
F2.3     72       90#
F2.4     80       92#
F2.5     83       95#
F8.0     52       98#
F9.0     54       100#
```

```
FFCHR    27#
IN.0     48       58#
SERIAL   49       71       87       89
STRING   3#       29       34       39
.
```

Microsoft Lib-80 Library Utility

Description

Microsoft's LIB-80 (LIB80 or LIB) library utility is a powerful but simple library manager that allows the programmer to perform several distinct but related tasks:

Concatenate multiple Microsoft REL format files into a single REL format file.

List the contents, global entrypoints, and external references of a library file.

Extract one or more modules from a library file.

Replace a module in a library file with another.

IMPORTANT: LIB-80 is potentially destructive, so NEVER OPERATE ON THE ONLY COPY OF A LIBRARY FILE. It is surprisingly easy to overwrite or otherwise destroy a file with LIB-80.

Commands

The basic command structures of LIB-80 are:

LIB80 libfile=srcfile,srcfile,.../s1/s2/s3.../sn

LIB80 libfile=srclbry<modules>,.../s1/s2/s3.../sn

LIB80 srclbry/s1/s2/s3.../s4

LIB80
* command
* command
* ...
* attention abort

where:

“libfile”	= the output library file, either type LIB or REL
“srcfile”	= an individual REL file
“scrlbry”	= a REL library file
“<modules>“	= one or more modules within a REL library file
“/s1”, etc.	= optional switches

Output Field

The output field “libfile=“ is optional, but the equal sign is required if the field is present.

If a command structure or switch is used that produces an output library, but the output library field is not present, LIB-80 produces an output library of the default filename “FORLIB.LIB” or, if the /E switch is used, “FORLIB.REL”. “FORLIB.REL is also the name of the standard library file provided with FORTRAN-80, so be especially careful if you have this language.

When an output library file is created, any existing file of the same name is destroyed.

Source Field

The source field is that portion of the command line between the optional output field and the optional switches: an entry of some type is required in the source field (the source field may be the only field).

The source field may consist of:

One or more filenames.

One or more modules within one or more filenames.

A combination of the above.

If the source field consists of one or more simple (one module) REL files, or one or more library REL files taken in entirety, separate the filenames by commas:

```
file1,file2,file3,...
```

If the source field consists of one or more modules within one or more library REL files, follow each filename immediately (no space) with its module designation in angle brackets and separate the filename/module groups with commas:

```
file1<modules>,file2<modules>,file3<modules>,...
```

Of course, the two forms may be combined:

```
file1,file2<modules>,file3<modules>,file4,...
```

If the direct entry (asterisk prompt) form of command entry is chosen, the last example would become:

```
* file1
* file2<modules>
* file3<modules>
* file4
* ...
```

Module Designations

The "<modules>" designator can be in any of several distinct syntaxes.

1. To extract a single module from a file, the syntax is:

   ```
   file<mod1>
   ```

2. To extract more than one distinct modules:

   ```
   file<mod1,mod2,mod3,mod4>
   ```

   The named modules need not be contiguous.

3. To extract all modules from the first module through a specified module:

   ```
   file<..mod1>
   ```

   Notice that there are TWO periods.

4. To extract all modules from a specific module through the last module:

   ```
   file<mod1..>
   ```

5. To extract all modules from one specific module through another specific module:

   ```
   file<mod1..mod2>
   ```

6. To extract n modules following a given module, but not the given module itself:

   ```
   file<mod1+4>
   ```

   The four modules following "mod1" are extracted, "mod1" is not.

7. To extract n modules before a given module, but not the given module itself:

   ```
   file<mod1–1>
   ```

   The module ahead of "mod1" is extracted, "mod1" is not.

8. To extract all modules between to given modules:

   ```
   file<mod1+1..mod2–1>
   ```

   All modules between "mod1" and "mod2" are extracted, but neither "mod1" nor "mod2".

9. To extract all modules EXCEPT a given module:

   ```
   file<..mod1–1,mod1+1..>
   ```

   All modules except "mod1" are extracted.

10. To extract all modules:

```
file
```

Other combinations are of course possible.

Switch Field

An entry in the switch field instructs LIB-80 to perform a specific task. Each switch is a letter preceded by a slash.

/C Cancel

The /C switch cancels all LIB-80 actions taken so far, without exiting to the operating system. Use /C if you are operating interactively and have specified the wrong modules or the wrong order.

/E Exit

The /E switch exits to the operating system, finishing whatever work is in process. The current output library is renamed from type LIB to type REL, and any previous copy of the library is deleted in the process.

If it is desired that the library retain the LIB filetype, exit via an attention abort rather than /E.

WARNING: THE /E SWITCH CAN EASILY DESTROY YOUR LIBRARY. Because the /E switch is so dangerous, several recommendations are made here:

ALWAYS work from a copy of a library, never the only copy.

ALWAYS use a name in the output field, even if one is not needed: “GARBAGE” is a good name. This covers you in case of error.

ALWAYS exit LIB-80 via an attention abort, rather than /E, unless you specifically want the /E rename.

In addition to the above recommendations, it is further recommended that master copies of a library be renamed to a type other than LIB or REL.

/H Hexadecimal

The /H switch sets the listing mode to hexadecimal (default) and cancel the effects of a previous /O switch. This switch is normally used in conjunction with the /L or /U switch.

/L List

The /L switch lists all the modules in a library, and all the global entrypoints and external references in each module.

All listing is to console: attention echo or some other means must be used to route the listing to a printer or file.

/O Octal

The /O switch sets the listing mode to octal and cancels the effects of a previous /H switch. This switch is normally used in conjunction with the /L or /U switch.

/R Rename

The /R switch renames the current output library from type LIB to type REL. This switch performs exactly like the /E switch save that it does not exit to the operating system.

WARNING: ALL THE WARNINGS APPLIED TO THE /E SWITCH APPLY TO THE /R SWITCH AS WELL.

/U List Undefined

The /U switch lists all those external references which could be left undefined by a single pass through a file or library. Since this is a single pass operation, those external references that refer “backwards” to a module already passed are listed.

Software 2000 GEN Utility

Introduction

GEN is a specialized two-pass linker utility for the Z-80 processor provided by Software 2000, Inc., for use with the TurboDOS operating system. GEN links together relocatable Microsoft format object code modules of type ".REL" to produce a command file of type ".COM", a TurboDOS system file of type ".SYS", or an absolute code file of any type.

Basic Operation

The GEN linker may be operated in either a direct or interactive mode. GEN is invoked via the command line:

```
GEN srcfile {destfile} {;options}
```

The "srcfile" argument identifies the two input files used by the linker: a configuration or generation file "srcfile.GEN" and a parameter file "srcfile.PAR". Either "srcfile.GEN" and/or "srcfile.PAR" may be missing. If "srcfile.GEN" is missing, then GEN enters an interactive mode. If "srcfile.PAR" is missing, no parameters are modified from their default values.

If the file "srcfile.GEN" is found, it must contain a list of the relocatable object code modules to be linked. Each line of the "srcfile.GEN" must contain one of the following two forms:

```
objfile {,objfile} ... {;comments}

; {comments}
```

Whitespace may be liberally used, but blank lines are NOT permitted and may cause premature "srcfile.GEN" termination.

If the file "srcfile.GEN" is not found, then GEN enters an interactive mode, presenting an asterisk "*" prompt. The syntax of each directive in the interactive mode is:

```
objfile {,objfile} ...
```

A null entry (carriage return only) terminates input and cause linking to commence.

All "objfile" entries, whether from "srcfile.GEN" or from interactive mode, are presumed to have type ".REL" unless an explicit filetype is specified.

After obtaining the list of "objfile" modules, whether from "srcfile.GEN" or interactively from the console, the GEN linker links the modules into an executable "destfile" output file, using a two-pass linking process that displays the encoded name of each module (as defined by the "MODULE" pseudo-instruction in the TASM assembly language source file) as it is linked.

During the linking process, each module is checked for a magnetically encoded serial number consisting of two parts: an "origin" number, indicating the TurboDOS OEM from whom the module was obtained, and a "unit" number indicating the specific TurboDOS from that OEM. GEN does not link modules with differing serial numbers or from OEM's other than GEN's own. No restrictions are place on non-serialized modules, which may be freely linked with each other or with serialized modules.

After linking, GEN looks for a "srcfile.PAR" parameter file and, if found, patches the global parameters of "destfile" according to the instructions contained in "srcfile.PAR".

Finally, after linking and patching, "destfile" is written to disk.

The "destfile" argument specifies the name of the executable output file to be created. If "destfile" is omitted, then "srcfile" is taken as "destfile".

The "destfile" argument should have an explicit filetype. The filetypes recognized are ".COM" and ".SYS". If the filetype is missing, type ".COM" is assumed. If the filetype is not ".SYS" a file identical with a type ".COM"

file is created, regardless of filetype, unless options are used to force a non-".COM" file.

Options

The options for GEN are always preceded by a semicolon prefix ";", and must be the last entries on the command line. Any number of options may be concatenated after a single semicolon. The options recognized by GEN are:

;Knnnn
Defines the lower limit of the common area in a bankswitched environment as “nnnn” hex.

;Lnnnn
Defines the lower limit of a “.COM” or other non-”.SYS” output file as “nnnn” hex. The default value is “L0100”, which is standard for “.COM” files.

;M
Lists a load map.

;S
Lists a sorted symbol table.

;Unnnn
Defines the upper limit of a “.SYS” output file as “nnnn” hex. The default is “UFFFF”, which is the 64KB upper memory boundary.

;X
Diagnose undefined references.

Generation File

The GEN generation file “srcfile.GEN” consists of a list of the relocatable object code modules, produced by the TASM assembler, to be linked together. This file may be created using any ordinary text editor or word processor, as long as the text editor or word processor output is a true ASCII file. ASCII is a 7-bit code, and GEN may fail if the generation file contains any character with bit 8 set.

A sample generation file follows:

```
;OSSINGLE.GEN
;
;Generic single-user system without spooling
;
;February 15, 1987
;
STDSINGLE   ;Standard non-spooled single-user package
PATCH       ;Patch area
;
CPMSUP      ;Support for C-fnc 7,8,24,28,29,31,37,107
;MPMSUP     ;Support for C-fnc 141-143,153,159-161
;QUEMGR     ;Support for C-fnc 134-140
;SUBMIT     ;Support for CP/M $$$.SUB files
;USRSUP     ;Support for user-defined functions
;
HDWNIT      ;Hardware initialization
SPDCPU      ;Serial/parallel driver
RTCCPU      ;Real-time clock driver
INTCPU      ;Interrupt-controller driver
;
CON192      ;Console driver
LSTCTS      ;Serial list driver with CTS handshaking
;LSTXON     ;Serial list driver with Xon/Xoff handshaking
;LSTETX     ;Serial list diver with ETX/ACK handshaking
LSTPAR      ;Parallel list driver
;
DSKFLP      ;Floppy disk driver
DSKHRD      ;Hard disk driver
;
;End of GENeration file
```

Please note the liberal use of whitespace and comments to increase legibility, and that there are NO blank lines (there are, instead, many blank comment lines containing only a semicolon). GEN would interpret this file as:

```
STDSINGLE,PATCH,CPMSUP,HDWNIT,SPDCPU,RTCCPU,INTCPU,CON192,LSTCTS,LSTPAR,DSKFLP,DSKHRD
```

Parameter File

GEN includes a very powerful symbolic patch facility that may be used to alter global parameters (those with "::" labels in the M80 assembly language source file) in the linked system or command file from their default value to whatever value is appropriate for the output file at hand. This patching is performed via a parameter file "srcfile.PAR" as the final step in the linking process. The syntax of each parameter file must be in one of two forms:

location = value {,value} ... {;comment}

; {comment}

where the "value" assignments are to be found in consecutive locations in memory, starting at the address specified by "location".

The "location" argument may be the name of a public or global symbol, an integer constant, or an expression composed of a public symbol name plus or minus an integer constant, and must be followed by an equal sign "=".

The "value" arguments may be a public symbol name, an integer constant, an expression composed of a public symbol name plus or minus an integer constant, or an ASCII string.

The "value" expression is stored as a word value (two bytes, least significant byte first) if its value exceeds 255 or if it is enclosed in parentheses (...), otherwise it is stored as a byte. Public symbol names are always treated as words.

All integer constants must be in hexadecimal and must start with a digit to distinguish them from names.

ASCII strings must be enclosed in double quotes "..." and are stored as a series of bytes. Control characters may be entered as "^c" (circumflex character) in a one character string, but not in a multi-character string. The circumflex character may not be entered in a string, and must be entered as the hex value 5E.

It may be mentioned at this time that either a comma "," or a carriage return/linefeed pair may be used as a value separator. Strings and discrete hex values may be entered on the same line or not, as desired.

A sample parameter file follows:

```
;OSSINGLE.PAR
;
;Generic single-user system without spooling
;
;15 February 1987
;
BUFSIZ = 03                         ;Buffer size, 1024 bytes
NMBUFS = 02                         ;Number of disk buffers
;
MEMRES = (0800)                     ;Reserved operating system scratch area
;
SRHDRV = 0FF                        ;Search drive
AUTUSR = 80                         ;Automatic log-on
ATNCHR = 00                         ;Attention character
;
WARMFN = 00,"WARMSNGL","SYS"        ;Filename to warmstart
;
CONAST = 00,CONDRA                  ;Console = serial port 0
;
CONBR  = 8E                         ;Console Baud rate
FFCHR  = 1A                         ;Console formfeed chr
;
PTRAST = 01,LSTDRA                  ;Printer A = CTS serial port 1
00,LSTDRB                           ;Printer B = parallel port 0
;
CTSBR  = 67                         ;CTS printer Baud rate
CTSFF  = 0C                         ;CTS printer auto formfeed
;
LSTPFF = 0C                         ;Parallel printer auto formfeed
;
DSKAST = 00,DSKDRA                  ;Drive A = floppy
01,DSKDRA                           ;Drive B = floppy
00,DSKDRB                           ;Drive C = hard
```

```
;
;End of PARameter file
```

Again, as in the generation file, note the liberal use of whitespace and comments to increase legibility. GEN would treat this parameter file as:

```
BUFSIZ = 03
NMBUFS = 02
MEMRES = (0800)
SRHDRV = 0FF
AUTUSR = 80
ATNCHR = 00
WARMFN = 00,"WARMSNGL","SYS"
CONAST = 00,CONDRA
CONBR  = 8E
FFCHR  = 1A
PTRAST = 01,LSTDRA,00,LSTDRB
CTSBR  = 67
CTSFF  = 0C
LSTPFF = 0C
DSKAST = 00,DSKDRA,01,DSKDRA,00,DSKDRB
```

Serialization

The Software 2000 GEN linker is a serialization sensitive linker. That is, it may link either serialized or unserialized modules together or to each other indiscriminately, providing the serialized modules have identical serial numbers and match the partial serial number of GEN itself.

In the TurboDOS environment, a module may be serialized by means of an origin and unit number defined in the source code as:

```
GLOBAL  ORIGIN,UNIT

;

ORIGIN  EQU     nnnnH           ;Origin number

UNIT    EQU     nnnnH           ;Unit number
```

Note that the definition is that of a GLOBAL EQUATE, which produces a word value for each variable that is readable by GEN during the linking process. GEN itself is partially serialized with the ORIGIN value.

If an attempt is made to execute GEN with serialized modules with ORIGIN values different from GEN's own, a serialization error occurs and linking is denied. Likewise, if an attempt is made to link serialized modules with either differing ORIGIN or UNIT values, a similar error occurs. The solution is to use properly serialized modules, or either remove or change the serial numbers causing conflict.

WARNING: It is unethical (as well as illegal) to reserialize any module unless you are legally authorized to have that module. Remember always the Golden Rule: Semper Non Rippus Offus.

Individual REL modules may be reserialized via the RELSER public domain utility provided as part of the Z80 manual library.

The serial number for GEN.COM is located in a word at address 01BBH (assuming GEN.COM starts at 0100H). Since this word is stored, like all words, least significant byte first, and since the maximum ORIGIN value was under 64 or 0040H, only the single byte located at 01BBH need be changed. This may be accomplished via ZSID or any other debugger, via the TurboDOS MONITOR utility, of via the GENSER utility provided as part of the Z80 manual library. Source code for the GENSER utility follows:

```
        NAME    ('GENSER')
;
        .Z80                            ;Use Zilog mnemonics
;
;*****  Equates  ************************************************************
;
BEL     EQU     7                       ;Bell
FF      EQU     10                      ;Formfeed
CR      EQU     13                      ;Carriage Return
;
```

```
BDOS    EQU     0005H                   ;C-function entry address
TDOS    EQU     0050H                   ;T-function entry address
;
SERBYT  EQU     00BBH                   ;GEN.COM serial number address
;
WRCNB   EQU     9                       ;C-9:   write console buffer
RDCNB   EQU     10                      ;C-10:  read console buffer
OPFIL   EQU     15                      ;C-15:  open file
CLFIL   EQU     16                      ;C-16:  close file
WRSEQ   EQU     21                      ;C-21:  write sequential
WRATT   EQU     30                      ;C-30:  write file attributes
RDRAN   EQU     33                      ;C-33:  read random
;
WRCPT   EQU     13                      ;T-13:  write COMPAT byte
;
;*****  Data  ************************************************************
;
        DSEG                            ;Locate in data segment
;
TFCB:   DB      0                       ;File control block
        DB      'GEN     ','COM'
        DB      0,0,0,0
        DB      0,0,0,0,0,0,0,0
        DB      0,0,0,0,0,0,0,0
        DB      0,0,0,0
;
INPBUF: DB      2,0,0,0,0           ;Input Buffer
;
IINMSG: DB      BEL,CR,FF,FF
        DB      'Illegal entry, try again!'
INPMSG: DB      CR,FF,FF
        DB      '   The HEXADECIMAL value of the old serial number is:  '
MSDSER: DB      '0'
LSDSER: DB      '0'
        DB      CR,FF,FF
        DB      'Enter the HEXADECIMAL value of the new serial number:  $'
;
OUTMSG: DB      CR,FF,FF
        DB      'The serial number has been changed.'
        DB      CR,FF,'$'
;
FNFMSG: DB      BEL,CR,FF,FF
        DB      'GEN.COM has not been found.'
        DB      CR,FF,'$'
;
;*****  Code  ************************************************************
;
        CSEG                            ;Locate in code segment
;
        LD      E,04H                   ;Inhibit global access
        LD      C,WRCPT
        CALL    TDOS
;
        LD      DE,TFCB                 ;Clear all GEN.COM attributes
        LD      C,WRATT
        CALL    BDOS
;
        OR      A                       ;Was file found?
        JR      Z,_S1                   ;If yes, skip
;
        LD      DE,FNFMSG           ;Display file not found and exit
```

```
        LD      C,WRCNB
        JP      BDOS
;
_S1:    LD      DE,TFCB                 ;Open GEN.COM
        LD      C,OPFIL
        CALL    BDOS
;
        LD      DE,TFCB                 ;Read 1st record
        LD      C,RDRAN
        CALL    BDOS
;
        LD      A,(SERBYT)          ;Get current serial number
        LD      B,A                     ;Save ls digit
        RRCA                            ;Shift ms digit
        RRCA
        RRCA
        RRCA
        AND     0FH                     ;Mask it
        OR      '0'                     ;Make it ASCII
        CP      ':'                     ;0-9?
        JR      C,_S2                   ;If yes, skip
;
        ADD     A,7                     ;Convert to A-F
_S2:    LD      (MSDSER),A          ;Stuff ms digit
        LD      A,B                     ;Get ls digit
        AND     0FH                     ;Mask it
        OR      '0'                     ;Make it ASCII
        CP      ':'                     ;0-9?
        JR      C,_S3                   ;If yes, skip
;
        ADD     A,7                     ;Convert to A-F
_S3:    LD      (LSDSER),A          ;Stuff ls digit
        LD      DE,INPMSG           ;Display input message
        LD      C,WRCNB
        CALL    BDOS
;
_S4:    LD      DE,INPBUF           ;Get new serial number
        LD      C,RDCNB
        CALL    BDOS
;
        LD      A,(INPBUF+1)        ;Were 2 digits entered?
        CP      2
        JR      Z,_S6                   ;If yes, skip
;
_S5:    LD      DE,IINMSG           ;Display illegal input message
        LD      C,WRCNB
        CALL    BDOS
;
        JR      _S4                     ;Try again
;
_S6:    LD      A,(INPBUF+2)        ;Get ms digit
        CP      '0'                     ;Below 0?
        JR      C,_S5                   ;If yes, error
;
        CP      ':'                     ;0-9?
        JR      C,_S7                   ;If yes, skip
;
        SUB     7                       ;Convert A-F
        CP      ':'                     ;Below A?
        JR      C,_S5                   ;If yes, error
;
```

```
        CP      '@'                     ;A-F?
        JR      NC,_S5                  ;If no, error
;
_S7:    AND     0FH                     ;Mask ms digit
        RLCA                            ;Shift to upper nybble
        RLCA
        RLCA
        RLCA
        LD      B,A                     ;Save it
        LD      A,(INPBUF+3)        ;Get ls digit
        CP      '0'                     ;Below 0?
        JR      C,_S5                   ;If yes, error
;
        CP      ':'                     ;0-9?
        JR      C,_S8                   ;If yes, skip
;
        SUB     7                       ;Convert A-F
        CP      ':'                     ;Below A?
        JR      C,_S5                   ;If yes, error
;
        CP      '@'                     ;A-F?
        JR      NC,_S5                  ;If no, error
;
_S8:    AND     0FH                     ;Mask ls digit
        OR      B                       ;Bring in ms digit
        LD      (SERBYT),A          ;Stuff new serial number
        LD      DE,TFCB                 ;Write it to disk
        LD      C,WRSEQ
        CALL    BDOS
;
        LD      DE,TFCB                 ;Close GEN.COM
        LD      C,16
        CALL    BDOS
;
        LD      DE,OUTMSG           ;Display exit message
        LD      C,WRCNB
        JP      BDOS
;
;*******************************************************************************
;
        END
```

Software 2000 Package Utility

The Software 2000 PACKAGE utility provided with TurboDOS allows the programmer to concatenate a collection of related Microsoft format REL files into a single REL file. This has two advantages:

Modules critical to each other cannot inadvertently be omitted from the linking procedure.

Modules can still be distributed in REL file format, offering full protection while still allowing the flexibility of the GEN linker's PARameter file feature.

The PACKAGE utility has the following command formats:

PACKAGE {srcefile} destfile

PACKAGE destfile
* relfile{,relfile}{,relfile}{,...}
* relfile{,...}
* ...
* relfile{,...}
*

In the first command format, the PACKAGE utility searches for a file with the name “srcefile.PKG” or, if “srcefile” was not specified, “destfile.PKG”, and processes this file, concatenating the REL file modules listed, and produce an output file of name “destfile.REL”.

The PKG file is merely an ASCII file containing a list of the REL file modules to be concatenated (similar in structure to the GEN file used by the GEN linker).

In the second command format, there is no PKG file to be found, so PACKAGE enters an interactive mode (again, similar to the GEN linker), allowing the REL file modules to be entered one at a time, in groups, or however. Entering a carriage return at the prompt terminates input and causes the entered modules to be concatenated into “destfile.REL”.

The final, concatenated REL file is indistinguishable from a library module created by the Microsoft LIB-80 utility, and can be decatenated and manipulated by that utility

Error messages displayed by the PACKAGE utility are:

File name missing from command

Invalid input file name

Non-privileged user

Unexpected EOF in input file

Disk is full

Can't make output file

Can't open input file

No input files

Digital Research ZSID Debugger

Description

A debugger is a program that allows another program to be inspected and/or modified. Its primary task is for use in debugging a new program, but it can also be used to analyze and understand the operation of a program, or as an aid in disassembly.

The Digital Research ZSID debugger allows the debugging and analyzing of programs written in Zilog Z-80 code. The code set of ZSID is a true representation of the Z-80 instruction set with only a couple of minor glitches (which are covered later), thus is an ideal debugger for use with the Z-80 processor.

Command Line Format

There are several means of entering ZSID from the operating system:

0A>ZSID

This method enters ZSID without loading any file.

0A>ZSID filename.typ

This method enters ZSID, loading the file “filename.typ” at 0100h. Notice the “filename.typ” is loaded as though it were a COM file, regardless of the filetype (with the exceptions of HEX, UTL, and SYM files covered below), but the file, once loaded, can be moved in memory very easily.

0A>ZSID filename.HEX

This method enters ZSID, loading the Intel HEX file “filename.HEX” at the address specified within the file itself.

0A>ZSID filename.UTL

This method enters ZSID, loading the UTiLity file “filename.UTL” created by either TRACE or UTIL, the ZSID utilities.

0A>ZSID filename.typ filename.SYM

This method enters ZSID, loading the file “filename.typ” at 0100h, and its symbol table file “filename.SYM” at the top of TPA.

Once within ZSID, with or without a file loaded, analysis and debugging may proceed with ZSID's internal command structure.

Special Characters

There are several characters that have a special meaning in a ZSID command:

Command Level Prompt:

Starts a command.

Decimal Value:

When placed in front of a number or expression, “#” means decimal. The following two commands are identical:

#D0103

Display starting at 0103h

#D#259

Display starting at 0103h

. Symbol:

When symbols have been installed, “.” references the 16-bit value of the symbol’s address.

#D.START

Display the value of START

@ 16-Bit Value:
16-bit values a symbol location.

#D@START

Display 16-bit value (word) at START

= 8-bit value:
8-bit value at a symbol location.

#D=START

Display 8-bit value (byte) at START

+ Addition:
Adds the value that follows to the previous value. The result can be no larger than 16 bits, so wraparound may take place (FFFFh + 0001h = 0000h, not 10000h).

#D.START+5

Displays the value of (START+0005)

+ Incremental offset:
Adds the value that follows to the previous value for that parameter.

#D+20

Displays from current display counter plus 20h.

– Subtraction:
Subtracts the value that follows from the previous value (actually, adds the complement of the value that follows to the previous value). The result can be no larger than 16 bits, so wraparound may take place.

#D@START–10

Displays word at START–10

– Decremental Offset:
Subtracts the value that follows from the previous value for that parameter.

#D–20

Displays from current display counter less 20h.

" String:
Accepts graphic ASCII characters after a double quote.

#S.MESSAGE 0234 23 “This string is going into memory 0255 D1 . #

' Short String:
Accepts one or two graphic characters within paired single quotes. An attempt to insert more than two characters results in truncation. An apostrophe must be paired for storage.

#S0080 0080 12 'ab' 0082 34 'C'" 0084 56 'defg' 0086 78 'h' 0087 90 . #

Stores “a” at 0080, “b” at 0081
Stores “C” at 0082, “ ' “ at 0083 ('C'" is ' + C + ' + ' + ')
Stores “d” at 0084, “e” at 0085
Stores “h” at 0086

Note that values within single quotes are TWO BYTES, not one word, and do not reverse during storage. “ LD HL 'mn' “ loads H with “m” and L with “n”.

Commands

A Assemble
Begins in line assembly at the current pointer position, as was left from the last assembled, listed or traced address: the Z-80 mnemonics may be entered directly in assembly language, and data fields may consist of symbolic expressions.

As Assemble from “s”
Begins in line assembly at location “s”: the Z-80 mnemonics may be entered directly in assembly language and data fields may consist of symbolic expressions.

“s” may be either an address or a symbol.

-A Disable Assembly
Removes the assembler and disassembler from ZSID, gaining about 4K of debugging room, and disabling subsequent “A” and “L” commands.

This command rarely requires execution under a bank-switched TurboDOS system, due to the large TPA involved.

Cs Call Subroutine “s”
Performs a direct subroutine call to location “s” without disturbing the CPU state of the program under test.

“s” may be either an address or a symbol.

Cs,b Load BC, Call Subroutine “s”
Loads register pair BC with “b”, then performs a direct subroutine call to location “s” without disturbing the CPU state of the program under test.

“s” may be either an address or a symbol, and “b” may be either a value or a symbol.

Cs,b,d Load BC, Load DE, Call Subroutine “s”
Loads register pair DE with “d”, register pair BC with “b”, then performs a direct subroutine call to location “s” without disturbing the CPU state of the program under test.

“s” may be either an address or a symbol, and “b” and “d” may be either values or symbols.

D Display Memory
Displays 11 lines of memory as bytes from the last address displayed.

D,f Display Memory to “f”
Displays memory as bytes from the last address displayed through location “d”.

“f” may be either an address or a symbol.

Ds Display Memory from “s”
Displays 11 lines of memory as bytes from location “s”.

“s” may be either an address or a symbol.

Ds,f Display Memory from “s” to “f”
Displays memory as bytes from location “s” through location “d”.

“s” and “f” may be either addresses or symbols.

DW Display Word Memory
Displays 11 lines of memory as words from the last address displayed.

DW,f Display Word Memory to “f”
Displays memory as words from the last address displayed through location “d”.

“f” may be either an address or a symbol.

DWs Display Word Memory from “s”
Displays 11 lines of memory as words from location “s”.

“s” may be either an address or a symbol.

DWs,f Display Word Memory from “s” to “f”
Displays memory as words from location “s” through location “d”.

“s” and “f” may be either addresses or symbols.

Fs,f,d Fill Memory from “s” to “f” with “d”
Fill memory from location “s” through location “f” with the byte “d”

“s” and “f” may be either addresses or symbols, and “d” may be either a value or an expression.

G Go to (Run Program)
Runs the program under test from the current register PC location.

G,a Go to Breakpoint “a”
Runs the program under test from the current register PC location until breakpoint “a” is reached.

“a” may be either an address or a symbol.

G,a,b Go to Breakpoint “a” or “b”
Runs the program under test from the current register PC location until either breakpoint “a” or breakpoint “b” is reached.

“a” and “b” either addresses or symbols.

Gp Go from “p”
Runs the program under test from location “p”

“p” may be either an address or a symbol.

Gp,a Go from “p” to Breakpoint “a”
Runs the program under test from location “p” until breakpoint “a” is reached.

“p” and “a” may be either addresses or symbols.

Gp,a,b Go from “p” to Breakpoint “a” or “b”
Runs the program under test from location “p” until either breakpoint “a” or breakpoint “b” is reached.

“p”, “a” and “b” may be either addresses or symbols.

-G Go to until passpoint=01
Runs the program under test from the current register PC location, or until a passpoint decrements to 01.

-G,a Go to Breakpoint “a” or until passpoint=01
Runs the program under test from the current register PC location until breakpoint “a” is reached, or until a passpoint decrements to 01.

“a” may be either an address or a symbol.

-G,a,b Go to Breakpoint “a” or “b” or until passpoint=01
Runs the program under test from the current register PC location until either breakpoint “a” or breakpoint “b” is reached, or until a passpoint decrements to 01.

“a” and “b” may be either addresses or symbols.

-Gp Go from “p” or until passpoint=01
Runs the program under test from location “p”, or until a passpoint decrements to 01.

“p” may be either an address or a symbol.

-Gp,a Go from “p” to Breakpoint “a” or until passpoint=01
Runs the program under test from location “p” until breakpoint “a” is reached, or until a passpoint decrements to 01.

“p” and “a” may be either addresses or symbols.

-Gp,a,b Go from “p” to Breakpoint “a” or “b” or until passpoint=01
Runs the program under test from location “p” until either breakpoint “a” or breakpoint “b” is reached, or until a passpoint decrements to 01.

“p”, “a” and “b” may be either addresses or symbols.

H Dump Symbol Table
Dumps the values for all symbols currently contained in the symbol table: use DELETE to abort.

Ha Displays “a” in Hexadecimal, Decimal, ASCII, and Symbol
Performs number conversion in the format:

```
HHHH #DDDDD 'C' .SSSSSS
```

where:

HHHH = hexadecimal value

#DDDDD = decimal value
'C' = ASCII character value
.SSSSSS = symbol (if one exists)

“a” may be either a hexadecimal or decimal value, an ASCII character, or a symbol.

Ha,b Displays “a + b” and “a – b”

Produces the hexadecimal sum and difference in the format:

SSSS DDDD

“a” and “b” must be either hexadecimal or decimal values.

Ifilename.typ Initializes Default FCB to “filename.typ”

Parses “filename.typ” into the default FCB at address 005C. Does NOT load the file: a subsequent “R” command does that.

I*filename.SYM Initializes Default FCB for “filename.SYM” Symbols

Parses “filename.SYM” into the default FCB at address 005C and sets a flag so that a subsequent “R” command loads the symbols only and not the file.

L Lists Disassembly

Diassembles and lists 11 lines of machine code from the last listed, traced, or assembled location.

Ls Lists Disassembly from “s”

Diassembles and lists 11 lines of machine code from location “s”.

“s” may be either an address or a symbol.

Ls,f Lists Disassembly from “s” to “f”

Diassembles and lists machine code from location “s” through location “f”.

“s” and “f” may be either addresses or symbols.

-L Lists Absolute Disassembly

Diassembles and lists 11 lines of machine code from the last listed, traced, or assembled location. Labels and symbolic operands are not displayed.

-Ls Lists Absolute Disassembly from “s”

Diassembles and lists 11 lines of machine code from location “s”. Labels and symbolic operands are not displayed.

“s” may be either an address or a symbol.

-Ls,f Lists Absolute Disassembly from “s” to “f”

Diassembles and lists machine code from location “s” through location 'f'. Labels and symbolic operands are not displayed.

“s” and “f” may be either addresses or symbols.

Ms,f,d Move Memory from “s” through “f” to “d”

Moves an area of memory from location “s” through location “f” to an area of memory beginning at location “d”.

““s”, “f”, and “d” may be either addresses or symbols.

P Display All Passpoints

Displays all active passpoints and passcounters. A passpoint is a program counter location which is monitored during execution of a test program. A passpoint has an associated passcounter in the range of 01h through FFh, which is decremented each time the test executes the passpoint address. When the passcount equals 01, the passpoint address becomes a permanent breakpoint.

The maximum number of active passpoints that can be set is eight.

Pp Set Passpoint to “p” and passcount to 01

Sets a passpoint at location “p” with a passcount of 01.

“p” may be either an address or a symbol.

Pp,c Set Passpoint to “p” and passcount to “c”

Sets a passpoint at location “p” with a passcount of “c”.

“p” may be either an address or a symbol, and “c” may be either a hexadecimal or decimal value.

-P Clear All Passpoints
Clears all passpoints currently active.

-Pp Clear Passpoint at “p”
Clears the passpoint at location “p”.

“p” may be either an address or a symbol.

R Read File into Memory at 0100h
Reads the file currently indexed by the default FCB at address 005Ch into memory starting at address 0100h.

Rb Read File into Memory at 0100h+”b”
Reads the file currently indexed by the default FCB at address 005Ch into memory starting at address 0100+”b”, where “b” is an offset bias.

Ss Store Bytes into Memory Starting at “s”
Stores subsequent bytes or byte-strings into memory starting at location “s”. Enter a period to terminate the command.

“s” may be either an address or a symbol.

SWs Store Words into Memory Starting at “s”
Stores subsequent words into memory starting at location “s”. Enter a period to terminate the command.

“s” may be either an address or a symbol.

T Trace Next Step
Executes a single instruction at the current register PC location, showing the CPU status in full.

T,c Trace Next Step and Call “c”
Executes a single instruction at the current register PC location, showing the CPU status in full, then calls location “c” without disturbing the CPU status.

“c” may be either an address or a symbol.

Tn Trace Next “n” Steps
Executes “n” instructions at the current register PC location, showing the CPU status in full at each step.

“n” may be either a hexadecimal or a decimal value.

Tn,c Trace Next “n” Steps and Call “c”
Executes “n” instructions at the current register PC location, showing the CPU status in full at each step, calling location “c” at each step without disturbing the CPU status.

“n” may be either a hexadecimal or a decimal value, and “c” may be either an address or a symbol.

-T Trace Next Step w/o Symbols
Executes a single instruction at the current register PC location, disabling the symbols, showing the CPU status in full.

-T,c Trace Next Step and Call “c” w/o Symbols
Executes a single instruction at the current register PC location, disabling the symbols, showing the CPU status in full, then calls location “c” without disturbing the CPU status.

“c” may be either an address or a symbol.

-Tn Trace Next “n” Steps w/o Symbols
Executes “n” instructions at the current register PC location, disabling the symbols, showing the CPU status in full at each step.

“n” may be either a hexadecimal or a decimal value.

-Tn,c Trace Next “n” Steps and Call “c” w/o Symbols
Executes “n” instructions at the current register PC location, disabling the symbols, showing the CPU status in full at each step, calling location “c” at each step without disturbing the CPU status.

“n” may be either a hexadecimal or a decimal value, and “c” may be either an address or a symbol.

TW Trace Next Step Locally
Executes a single instruction at the current register PC location, showing the CPU status in full. Only local execution is displayed.

TW,c Trace Next Step Locally and Call “c”
Executes a single instruction at the current register PC location, showing the CPU status in full, then calls location “c” without disturbing the CPU status. Only local execution is displayed.

“c” may be either an address or a symbol.

TWn Trace Next “n” Steps Locally
Executes “n” instructions at the current register PC location, showing the CPU status in full at each step. Only local execution is displayed.

“n” may be either a hexadecimal or a decimal value.

TWn,c Trace Next “n” Steps Locally and Call “c”
Executes “n” instructions at the current register PC location, showing the CPU status in full at each step, calling location “c” at each step without disturbing the CPU status. Only local execution is displayed.

“n” may be either a hexadecimal or a decimal value, and “c” may be either an address or a symbol.

-TW Trace Next Step Locally w/o Symbols
Executes a single instruction at the current register PC location, disabling the symbols, showing the CPU status in full. Only local execution is displayed.

-TW,c Trace Next Step Locally and Call “c” w/o Symbols
Executes a single instruction at the current register PC location, disabling the symbols, showing the CPU status in full, then calls location “c” without disturbing the CPU status. Only local execution is displayed.

“c” may be either an address or a symbol.

-TWn Trace Next “n” Steps Locally w/o Symbols
Executes “n” instructions at the current register PC location, disabling the symbols, showing the CPU status in full at each step. Only local execution is displayed.

“n” may be either a hexadecimal or a decimal value.

-TWn,c Trace Next “n” Steps Locally and Call “c” w/o Symbols
Executes “n” instructions at the current register PC location, disabling the symbols, showing the CPU status in full at each step, calling location “c” at each step without disturbing the CPU status. Only local execution is displayed.

“n” may be either a hexadecimal or a decimal value, and “c” may be either an address or a symbol.

U Untrace Next Step
Executes a single instruction at the current register PC location.

U,c Untrace Next Step and Call “c”
Executes a single instruction at the current register PC location, then calls location “c” without disturbing the CPU status.

“c” may be either an address or a symbol.

Un Untrace Next “n” Steps
Executes “n” instructions at the current register PC location.

“n” may be either a hexadecimal or a decimal value.

Un,c Untrace Next “n” Steps and Call “c”
Executes “n” instructions at the current register PC location, calling location “c” at each step without disturbing the CPU status.

“n” may be either a hexadecimal or a decimal value, and “c” may be either an address or a symbol.

-U Untrace Next Step w/o Intermediate Passpoints
Executes a single instruction at the current register PC location, disabling the intermediate passpoints.

-U,c Untrace Next Step and Call “c” w/o Intermediate Passpoints
Executes a single instruction at the current register PC location, disabling the intermediate passpoints, then calls location “c” without disturbing the CPU status.

“c” may be either an address or a symbol.

-Un Untrace Next “n” Steps w/o Intermediate Passpoints
Executes “n” instructions at the current register PC location, disabling the intermediate passpoints.

“n” may be either a hexadecimal or a decimal value.

-Un,c Untrace Next “n” Steps and Call “c” w/o Intermediate Passpoints

Executes “n” instructions at the current register PC location, disabling the intermediate passpoints, calling location “c” at each step without disturbing the CPU status.

“n” may be either a hexadecimal or a decimal value, and “c” may be either an address or a symbol.

UW Untrace Next Step Locally

Executes a single instruction at the current register PC location. This is a “trace without call” operation.

UW,c Untrace Next Step Locally and Call “c”

Executes a single instruction at the current register PC location, then calls location “c” without disturbing the CPU status. This is a “trace without call” operation.

“c” may be either an address or a symbol.

UWn Untrace Next “n” Steps Locally

Executes “n” instructions at the current register PC location. This is a “trace without call” operation.

“n” may be either a hexadecimal or a decimal value.

UWn,c Untrace Next “n” Steps Locally and Call “c”

Executes “n” instructions at the current register PC location, calling location “c” at each step without disturbing the CPU status. This is a “trace without call” operation.

“n” may be either a hexadecimal or a decimal value, and “c” may be either an address or a symbol.

-UW Untrace Next Step Locally w/o Intermediate Passpoints

Executes a single instruction at the current register PC location, disabling the intermediate passpoints. This is a “trace without call” operation.

-UW,c Untrace Next Step Locally and Call “c” w/o Intermediate Passpoints

Executes a single instruction at the current register PC location, disabling the intermediate passpoints, then calls location “c” without disturbing the CPU status. This is a “trace without call” operation.

“c” may be either an address or a symbol.

-UWn Untrace Next “n” Steps Locally w/o Intermediate Passpoints

Executes “n” instructions at the current register PC location, disabling the intermediate passpoints. This is a “trace without call” operation.

“n” may be either a hexadecimal or a decimal value.

-UWn,c Untrace Next “n” Steps Locally and Call “c” w/o Intermediate Passpoints

Executes “n” instructions at the current register PC location, disabling the intermediate passpoints, calling location “c” at each step without disturbing the CPU status. This is a “trace without call” operation.

“n” may be either a hexadecimal or a decimal value, and “c” may be either an address or a symbol.

X Examine CPU State

Displays all registers and flags.

Xr Examine/Change Register “r”

Displays register “r”: at display, enter a new value to change or <return> to leave as is.

r = A = register A
r = A' = register A'
r = B = register pair BC
r = B' = register pair BC'
r = D = register pair DE
r = D' = register pair DE'
r = H = register pair HL
r = H' = register pair HL'
r = P = register PC
r = S = register SP
r = X = index register IX

r = Y = index register IY

Xf Examine/Change Flag “f”

Displays flag “f”, as a dash if clear or a letter if set: at display, enter a 0 to clear the flag, a 1 to set the flag, or <return> to leave as is.

f = C = carry
f = C' = alternative carry
f = Z = zero
f = Z' = alternative zero
f = M = sign (minus)
f = M' = alternative sign (alternative minus)
f = E = parity/overflow (parity even)
f = E' = alternative parity/overflow (alternative parity even)
f = I = half-carry (intercarry)
f = I' = alternative half-carry (alternative intercarry)

ZSID UTILITIES

ZSID has a pair of utilities to allow more powerful debugging. These utilities are HIST.UTL and TRACE.UTIL.

When either utility is loaded, ZSID responds with the following information:

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

where “iiii”, “cccc” and “dddd” are three absolute address entries to the utility for (re)initialization, collecting debug data, and displaying collected information, respectively. These labels are entered into a symbol table and may be referenced whenever desired. The normal method for running a utility is to call it by name or address.

To initialize the system and get addresses, execute:

```
#C.INITIAL
```

To collect data, use either the trace or untrace command, as:

```
#T100,.COLLECT
#U,.COLLECT
```

To increase user control, use pass points while monitoring:

```
#P120,15
#U1000,.COLLECT
```

To display collected data, execute:

```
#C,.DISPLAY
```

HIST.UTL

This utility creates a histogram of program execution between two locations given during initialization. Program addresses are monitored during Trace and Untrace commands. Summary data is displayed by using the display command. For this utility, the operator performs the following three steps:

1. INITIALize (get address bounds)
2. COLLECT the data
3. DISPLAY the data

Example:

```
0A>ZSID HIST.UTL
TYPE HISTOGRAM BOUNDS 100,115
.INITIAL = 3E03
.COLLECT = 3E06
.DISPLAY = 3E09
```

#ISAMPLE.COM SAMPLE.SYM
#R
SYMBOLS
#U32,.COLLECT
(The registers are displayed; wait a second, and prompt returns.)
#C.DISPLAY
(The histogram is displayed.)

TRACE.UTL

This utility provides a dynamic backtrace of up to 256 instructions, which end at the current breakpoint address. Instruction address collection occurs only in the Trace and Untrace commands. Passpoints can be used to increase operator control.

Example:

0A>ZSID
#ITRACE.UTL
#R
READY FOR SYMBOLIC BACKTRACE
#ISAMPLE.COM SAMPLE.SYM
#R
NEXT PC
0200 0100
#U100,.COLLECT
(The registers are displayed; wait a second, and prompt returns.)
#C.DISPLAY
(Symbolic backtrace appears.)

ZSAVE.MAC

One major problem with ZSID (or any other CP/M debugger) is the fact that there is no practical way to save the debugged file back onto disk from memory: under CP/M, the built in SAVE command is used, but TurboDOS has NO built in commands.

The author has corrected this problem by the creation of ZSAVE.MAC, which, when assembled and linked, provides a simple and elegant solution to the save problem.

Instructions for ZSAVE are contained in the source code as comments:

```
;ZSAVE.MAC
;
;Routine to save file from debugger memory
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       1.00
;
;Edit History:
;       03/13/90  rrb  1.00  Created
;
;*****************************************************************************
;
        .COMMENT ~

This routine is meant to be used with Digital Research's ZSID debugger when
operating under TurboDOS, and takes the place of CP/M's SAVE command.

Normally, use this routine in the following manner:

  1.  Enter ZSID bringing this routine along with you:
```

```
ZSID ZSAVE.RTN                    (or whatever name you chose)
#
```

2. Run this routine within ZSID:

```
#G
```

3. Note the "SAVE" value:

```
SAVE = D700
#
```

4. Load the file to be debugged:

```
#IMYFILE.COM
#R
```

5. Note its "NEXT" value:

```
NEXT
0680
#
```

6. Debug your file:

7. Run the saver (use "SAVE" address from above):

```
#GD700
```

8. Enter your program's "NEXT" value:

```
Enter NEXT: 0680

File "ZSAVE.TMP" saved.

(E)xit to operating system or return to debugger?
```

9. Enter "E" or "e" to exit to TurboDOS, or anything else to stay in ZSID.

If you have debugged a program and THEN wish to save it:

1. Load ZSAVE at the program's NEXT address

```
#IZSAVE.RTN
#R680
#
```

2. Add 100 hex (to get past COM-file offset) and set the program counter:

```
#XP

PC = nnnn  780

#
```

3. Execute "2T" to get past 0100 hex initialization:

4. Stuff NEXT+0100 into 0080 and 0081 in proper word order:

```
#S80
```

```
        0080 = nn  80
        0081 = nn  07
        0082 = nn  .

        #
```

5. Execute "G":

```
        #G
```

6. Note the "SAVE" value:

```
        SAVE = D700
        #
```

7. Run the saver:

```
        #GD700
```

8. Enter your program's "NEXT" value:

```
        Enter NEXT: 0680

        File "ZSAVE.TMP" saved.

        (E)xit to operating system or return to debugger?
```

9. Enter "E" or "e" to exit to TurboDOS, or anything else to stay in ZSID.

Conversely, special load-later versions of this routine may be generated by setting ORIGIN to the desired load address and re-assembling and linking.

After assembling with Macro-80, either link via TurboDOS's GEN linker:

```
        0A>GEN ZSAVE
        *ZSAVE
        *
        0A>
```

or Link-80 with the special command line:

```
        0A>L80 /P:0100,ZSAVE,ZSAVE/N/E
```

Either of these methods will produce a COM file without the 3-byte routing jump, which is absolutely essential in this case.

Since this is NOT a standard COM file and should NEVER be run from a system prompt, I suggest renaming the filetype (I use RTN, for "RouTiNe").

```
~
;*****************************************************************************
;
        NAME    ('ZSAVE')
;
        .Z80                    ;Use Zilog mnemonics
        .RADIX  16              ;All values in hex
;
;*****  Equates
;
ORIGIN  EQU     0100            ;Load address inside debugger
```

```
;
BEL     EQU     07              ;Bell
LF      EQU     0A              ;Linefeed
CR      EQU     0D              ;Carriage return
;
OPSYS   EQU     0000            ;Warm-start entrypoint address
BDOS    EQU     0005            ;C-function entrypoint address
;
ORGADR  EQU     0080            ;Origin address
FILLEN  EQU     0082            ;File length in records
DMAADR  EQU     0084            ;Saver DMA address
INPBUF  EQU     0086            ;Input buffer
;
;*****  Transfer Routine  ************************************************
;
        CSEG                    ;Locate in code segment
;
START:  LD      HL,ORIGIN       ;Stuff origin
        LD      (ORGADR),HL
        LD      HL,(BDOS+01)    ;Get bottom of debugger
        DEC     HL              ;Convert to top of TPA
        DEC     HL              ;Insurance
        DEC     HL              ;Room for save-module page
        PUSH    HL              ;Save SM-page address
        LD      DE,SMLEN        ;Room for save module
        OR      A
        SBC     HL,DE
        LD      L,00            ;Set to page boundary
        EX      (SP),HL         ;Save SM page
        POP     DE
        LD      (HL),D
        LD      A,D             ;Get MS digit of page
        RRCA
        RRCA
        RRCA
        RRCA
        AND     0F              ;Make it ASCII
        CP      0A
        JR      C,TR1
;
        ADD     A,07
TR1:    ADD     A,'0'
        LD      HL,(ORGADR)     ;Stuff it
        LD      BC,TRMD
        ADD     HL,BC
        LD      (HL),A
        LD      A,D             ;Get LS digit of page
        AND     0F              ;Make it ASCII
        CP      0A
        JR      C,TR2
;
        ADD     A,07
TR2:    ADD     A,'0'
        INC     HL              ;Stuff it
        LD      (HL),A
        LD      HL,(ORGADR)     ;Transfer save module
        LD      BC,SMBEG
        ADD     HL,BC
        LD      BC,SMLEN
        LDIR
;
```

```
        LD      HL,(ORGADR)     ;Display transfer message
        LD      DE,TRM1
        ADD     HL,DE
        EX      DE,HL
        LD      C,09
        CALL    BDOS
;
        RST     38              ;Return control to debugger
;
;*****  Transfer message
;
TRM1    EQU     $-START
        DB      CR,LF
        DB      'SAVE = '
TRMD    EQU     $-START
        DB      'XX00'
        DB      CR,LF,LF
        DB      'After loading main program, note '
        DB      'its "NEXT" address.  Save the debugged'
        DB      CR,LF
        DB      'program by the command "Gssss" where '
        DB      'ssss is the "SAVE" above.  Answer the'
        DB      CR,LF
        DB      "save modules question with your program's "
        DB      '"NEXT".'
        DB      CR,LF,'$'
;
;*****  Save Module  ************************************************
;
SMBEG   EQU     $-START
        LD      HL,(BDOS+01)    ;Get bottom of debugger
        DEC     HL              ;Get save-module page
        DEC     HL
        DEC     HL
        LD      D,(HL)
        LD      E,00            ;Make it an address
        LD      (ORGADR),DE     ;Save it in default DMA
        LD      DE,SMM2         ;Get entry message offset
SM1:    LD      HL,(ORGADR)     ;Display message
        ADD     HL,DE
        EX      DE,HL
        LD      C,09
        CALL    BDOS
;
        LD      HL,INPBUF       ;Get "NEXT"
        LD      (HL),04
        EX      DE,HL
        LD      C,0A
        CALL    BDOS
;
        LD      HL,INPBUF+01    ;Check on actual entry
        LD      A,(HL)          ;Get number of characters
        CP      03              ;At least 3?
SM2:    LD      DE,SMM1         ;(Get invalid entry message offset)
        JR      C,SM1           ;If no, try again
;
        LD      B,A             ;Set count to number of characters
        LD      DE,0000         ;Preset savefile length to zero
SM3:    INC     HL              ;Get entry character
        LD      A,(HL)
        CP      '0'             ;Before "0"?
```

```
        JR      C,SM2           ;If yes, try again
;
        CP      ":"             ;"0-9"?
        JR      C,SM4           ;If yes, skip
;
        AND     5F              ;Convert any "a-z" to "A-Z"
        CP      'A'             ;Before "A"?
        JR      C,SM2           ;If yes, try again
;
        CP      'G'             ;After "F"?
        JR      NC,SM2
;
        SUB     07              ;Adjust for "A-F"
SM4:    SUB     30              ;De-ASCII the character
        RLA                     ;Shift it into savefile length
        RLA
        RLA
        RLA
        LD      C,04
SM5:    RLA
        RL      E
        RL      D
        DEC     C
        JR      NZ,SM5
;
        DJNZ    SM3             ;Do next character
;
        DEC     DE              ;Convert "NEXT" to "LAST"
        LD      A,D             ;Enough to save?
        OR      A
        JR      Z,SM2           ;If no, try again
;
        LD      HL,(ORGADR)     ;Too much to save?
        DEC     HL
        OR      A
        SBC     HL,DE
        JR      C,SM2           ;If yes, try again
;
        DEC     D               ;Adjust for start at 0100
        RL      E               ;Convert to number of records
        RL      D
        LD      E,D
        LD      D,00
        RL      D
        INC     DE
        PUSH    DE              ;Save savefile length in records
        LD      HL,005C         ;Initialize FCB
        LD      DE,005D
        LD      BC,23
        LD      (HL),00
        LDIR
        LD      HL,(ORGADR)     ;Transfer savefile name
        PUSH    HL
        LD      DE,SMFN
        ADD     HL,DE
        LD      DE,005D
        LD      BC,000B
        LDIR
        LD      C,0D            ;Set DMA to default
        CALL    BDOS
;
```

```
        LD      DE,005C         ;Delete any existing old savefile
        LD      C,13
        CALL    BDOS
;
        LD      DE,005C         ;Check for one still out there
        LD      C,11
        CALL    BDOS
;
        POP     HL              ;Restore beginning address
        LD      (ORGADR),HL
        POP     HL              ;Restore savefile length
        LD      (FILLEN),HL
        LD      DE,SMM5         ;Was savefile found?
        CP      0FF
        JR      NZ,SM7          ;If yes, error
;
        LD      DE,005C         ;Create new savefile
        LD      C,16
        CALL    BDOS
;
        LD      DE,SMM6         ;Good create?
        OR      A
        JR      NZ,SM7          ;If no, error
;
        LD      HL,0100         ;Set DMA to file to be saved
        LD      (DMAADR),HL     ;Save DMA address
SM6:    LD      DE,(DMAADR)     ;Set DMA
        LD      C,1A
        CALL    BDOS
;
        LD      DE,005C         ;Write a record
        LD      C,15
        CALL    BDOS
;
        LD      HL,(DMAADR)     ;Increment DMA pointer
        LD      DE,ORGADR
        ADD     HL,DE
        LD      (DMAADR),HL
        LD      HL,(FILLEN)     ;Decrement record counter
        DEC     HL
        LD      (FILLEN),HL
        LD      A,H             ;Last record written?
        OR      L
        JR      NZ,SM6          ;If no, do next record
;
        LD      DE,005C         ;Close savefile
        LD      C,10
        CALL    BDOS
;
        LD      HL,(ORGADR)     ;Display saved message
        LD      DE,SMM3
        ADD     HL,DE
        EX      DE,HL
        LD      C,09
        CALL    BDOS
;
        LD      C,01            ;Get exit character
        CALL    BDOS
;
        PUSH    AF              ;Save it
        LD      HL,(ORGADR)     ;Display cr-lf
```

```
        LD      DE,SMM4
        ADD     HL,DE
        EX      DE,HL
        LD      C,09
        CALL    BDOS
;
        POP     AF              ;Restore exit character
        CP      'E'             ;Exit?
        JP      Z,OPSYS         ;If yes, go to operating system
;
        CP      'e'             ;Still exit?
        JP      Z,OPSYS         ;If yes, go to operating system
;
        RST     38              ;Return control to debugger
;
SM7:    LD      HL,(ORGADR)     ;Display error message
        ADD     HL,DE
        EX      DE,HL
        LD      C,09
        CALL    BDOS
;
        RST     38              ;Return control to debugger
;
;*****  Save Module Data
;
SMFN    EQU     $-START-SMBEG
        DB      'ZSAVE   ','TMP'
;
SMM1    EQU     $-START-SMBEG
        DB      BEL,CR,LF
        DB      'Invalid "NEXT" hex address.'
        DB      CR,LF
SMM2    EQU     $-START-SMBEG
        DB      CR,LF
        DB      'Enter value of "NEXT" byte:  $'
;
SMM3    EQU     $-START-SMBEG
        DB      CR,LF,LF
        DB      'File "ZSAVE.TMP" saved.'
        DB      BEL,CR,LF
        DB      '(E)xit to operating system or return to debugger? $'
;
SMM4    EQU     $-START-SMBEG
        DB      CR,LF,'$'
;
SMM5    EQU     $-START-SMBEG
        DB      BEL,CR,LF
        DB      'Unable to delete existing "ZSAVE.TMP" File.'
        DB      CR,LF,'$'
;
SMM6    EQU     $-START-SMBEG
        DB      BEL,CR,LF
        DB      'Unable to create "ZSAVE.TMP" File.'
        DB      CR,LF,'$'
;
SMLEN   EQU     $-START-SMBEG
;
;*************************************************************************
;
        END     START
```

Assembly under TurboDOS

General

Assembly language under the TurboDOS operating system is unique only in the facilities provided the programmer by the operating system. These facilities fall into two broad areas: memory usage and function calls.

Memory usage deals with the way TurboDOS utilizes the system memory, most notably the base page and pseudo-BIOS table.

Function calls deal with the CP/M BDOS functions supported by TurboDOS (C-functions), as well as those functions unique to TurboDOS (T-functions).

It must be emphasized that the object here is to cover general programming under TurboDOS, and not the programming of modules or drivers within TurboDOS itself. The "hooks" necessary for module and driver programming are covered in the TurboDOS Z-80 Implementer's Guide from Software 2000.

Base Page

Like CP/M, TurboDOS reserves the first 0100 hex (256 decimal) bytes of memory for operating system use. This base page (256 bytes = 1 page) contains many special entrypoints and values:

0000h–0002h Warmstart Jump

A 3-byte jump instruction to the pseudo-BIOS table warmstart vector is located at address 0000h. Because the pseudo-BIOS table always starts on a page boundary and the warmstart vector is the second 3-byte entry in the table, the byte at 0001h is always 03, and the byte at 0002h is always the page address of the pseudo-BIOS table (if this byte is FCh, the table resides at FC00h, and the warmstart vector is at address FC03h).

An application program usually terminates by executing a jump to 0000h.

0003h I/O Byte

The CP/M I/O byte is located at address 0003h. This byte is seldom used by TurboDOS systems, but can be used by an application program according to CP/M I/O Byte rules.

The I/O byte is used by a CP/M system to control the routing of its input/output devices. Each 2-bits of the I/O byte determine the routing of a different device:

Bits 1,0 — CON: Console

00 TTY:=CON: console is teletype
01 CRT:=CON: console is terminal
10 BAT:=CON: console input is reader console output is list
11 UC1:=CON: console is user-defined console 1

Bits 3,2 — RDR: Reader

00 TTY:=RDR: reader is teletype
01 RDR:=RDR: reader is high speed reader
10 UR1:=RDR: reader is user-defined reader 1
11 UR2:=RDR: reader is user-defined reader 2

Bits 5,4 — PUN: Punch

00 TTY:=PUN: punch is teletype
01 PUN:=PUN: punch is high speed punch
10 UP1:=PUN: punch is user-defined punch 1
11 UP2:=PUN: punch is user-defined punch 2

Bits 7,6 — LST: List Device

00 TTY:=LST: list is teletype
01 CRT:=LST: list is terminal
10 LPT:=LST: list is line printer
11 UL1:=LST: list is user-defined list device

Many if not most of the I/O devices mentioned above do not exist any longer and are not supported by TurboDOS (or later versions of CP/M).

0004h Drive and User

The CP/M drive and user byte is located at address 0004h. Like the I/O byte, the drive and user byte is really a CP/M phenomenon of limited use under TurboDOS. The lower nybble (bits 3–0) of the drive and user byte contains the current drive assignment:

0000	Drive A:	1000	Drive I:
0001	Drive B:	1001	Drive J:
0010	Drive C:	1010	Drive K:
0011	Drive D:	1011	Drive L:
0100	Drive E:	1100	Drive M:
0101	Drive F:	1101	Drive N:
0110	Drive G:	1110	Drive O:
0111	Drive H:	1111	Drive P:

The upper nybble (bits 7–4) contain the current user area assignment as a modulo-16 value. Under CP/M, this was sufficient as there were only 16 user areas, but under TurboDOS with its 32 user areas, this value is only semi-useful:

0000	User 0: or 16:	1000	User 8: or 24:
0001	User 1: or 17:	1001	User 9: or 25:
0010	User 2: or 18:	1010	User 10: or 26:
0011	User 3: or 19:	1011	User 11: or 27:
0100	User 4: or 20:	1100	User 12: or 28:
0101	User 5: or 21:	1101	User 13: or 29:
0110	User 6: or 22:	1110	User 14: or 30:
0111	User 7: or 23:	1111	User 15: or 31:

0005h–0007h C-Function (BDOS) Jump

A 3-byte jump instruction to the C-function entrypoint is located at address 0005h. An application program invokes a C-function by placing the function number in the C register and executing a “CALL 0005” instruction.

The value contained in addresses 0006h and 0007h is one greater than the top of the transient program area: inspection of this value allows an application program to determine the amount of memory available for its own use.

0008h–000Fh Restart 08

Addresses 0008h–000Fh are reserved for use by the “RST 08” instruction.

0010h–0017h Restart 10

Addresses 0010h–0017h are reserved for use by the “RST 10” instruction.

0018h–001Fh Restart 18

Addresses 0018h–001Fh are reserved for use by the “RST 18” instruction.

0020h–0027h Restart 20

Addresses 0020h–0027h are reserved for use by the “RST 20” instruction.

0028h–002Fh Restart 28

Addresses 0028h–002Fh are reserved for use by the “RST 28” instruction.

0030h–0037h Restart 30

Addresses 0030h–0037h are reserved for use by the “RST 30” instruction.

0038h–003Fh Restart 38

Addresses 0038h–003Fh are reserved for use by the “RST 38” instruction. This instruction is often used by a debugger to trap operation and allow single stepping through a program or routine.

0040h–004Fh Reserved

The addresses 0040h–004Fh are unused by TurboDOS, but re-initialized by certain internal operations. Hence, these addresses may not be reliably used for other purposes.

0050h–0052h T-Function Jump

A 3-byte jump instruction to the T-function entrypoint is located at address 0050h. An application program invokes a T-function by placing the function number in the C register and executing a “CALL 0050” instruction.

0053h–005Bh Reserved

The addresses 0053h–005Bh are unused by TurboDOS, but re-initialized by certain internal operations. Hence, these addresses may not be reliably used for other purposes.

005Ch–007Fh Default File Control Block (FCB)

The 36-byte system default file control block is located at address 005Ch. This is a full FCB and may be used for any file operation or function call.

When the command tail contains a file specification as its first entry, that file specification is automatically parsed into the default FCB.

005Ch–006Bh Primary Partial Default File Control Block

The 16-byte primary partial default file control block is located at addresses 005Ch–006Bh. This partial FCB may be easily and automatically converted into a full FCB through the use of any of several file operations and function calls.

When the command tail contains two file specifications as its first two entries, the first of those file specifications is automatically parsed into the primary partial default FCB.

006Ch–007Bh Secondary Partial Default File Control Block

The 16-byte secondary partial default file control block is located at addresses 005Ch–006Bh. This partial FCB is overwritten by an expansion of the primary partial FCB into a full FCB, and it is the responsibility of the application programmer to move the data in this area to an alternative FCB prior to implementing any file operations or function calls.

When the command tail contains two file specifications as its first two entries, the second of those file specifications is automatically parsed into the secondary partial default FCB.

0080h–00FFh Command Tail Buffer

The 128-byte command tail buffer is located at address 0080h. The command tail is fully parsed into this buffer up to a maximum length of 126 bytes. If the command tail exceeds 126 bytes, it is truncated.

This command tail buffer is overwritten by the first file read that takes place at the default DMA address, which is also 0080h, and it is the responsibility of the application programmer to either set the DMA to another buffer address or to either fully process or move the parsed command tail data into another area.

0080h–00FFh Default Direct Memory Access (DMA) Buffer

The 128-byte default direct memory access buffer is located at address 0080h. This buffer is used by the system for disk reads and writes and other DMA functions unless the DMA address is intentionally changed to another location via C-function 26.

The default DMA buffer occupies the same memory as and overwrites the command tail buffer. It is the responsibility of the application programmer to take steps to prevent this from happening.

Pseudo-BIOS Branch Table

For compatibility with CP/M, TurboDOS provides a full simulated “BIOS Branch Table” to support applications that make direct BIOS calls. The branch table is compatible with CP/M 2.2, and always begins on a page boundary (address is always nn00h) as in CP/M. BIOS calls are for the most part equivalent to C-function calls and unless there is some compelling reason they should not be used under TurboDOS.

nn00h Coldstart

Executes C-function 255, System Cold Reset.

nn03h Warmstart

Executes C-function 0, System Reset.

nn06h Get Console Status in A

Executes C-function 11, Get Console Status.

nn09h Get Console Input in A

Executes C-function 3, Raw Console Input.

nn0Ch Put Console Output from C

Moves C to E and executes C-function 4, Raw Console Output.

nn0Fh Put List Output from C

Moves C to E and executes C-function 5, List Output.

nn12h Put Raw Console Output from C
Moves C to E and executes C-function 4, Raw Console Output.

nn15h Get Raw Console Input in A
Executes C-function 3, Raw Console Input.

nn18h Home Drive to Track 0
Indexes current drive track pointer to 0000.

nn1Bh Select Drive from C
Moves C to E and executes C-function 14, Select Drive.

nn1Eh Set Drive Track from BC
Indexes current drive track pointer to BC.

nn21h Set Drive Sector from BC
Indexes current drive sector pointer to BC – 1.

nn24h Set DMA Address from BC
Moves BC to DE and executes C-function 26, Set DMA Address.

nn27h Read Drive Sector
Executes several functions to read current sector.

nn2Ah Write Drive Sector
Executes several functions to write current sector.

nn2Dh Get List Status in A
Not supported under TurboDOS, always returns list ready.

nn30h Translate Drive Sector from BC to HL
Moves BC+1 to HL.

Base page addresses 0001h and 0002h contain the address of the pseudo-BIOS table warmstart entrypoint. Therefore, the table address can always be obtained by taking the word at 0001h and subtracting 0003h.

Command Tail Parsing

The command tail is that part of the command line following the command itself until either a carriage return or a command separator is reached. In the command:

```
M80 MYPROG,MYPROG=MYPROG/Z
```

The command is “M80” and the command tail is “ MYPROG,MYPROG=MYPROG/Z”. Note that the space following the command is a part of the command tail.

The command tail is parsed into the command tail buffer, located at base page address 0080h, in the following manner:

0080h	The length of the command tail is placed here, and is a value between 0 and 126.
0081h–00nnh	The command tail is parsed into the buffer exactly as it was entered, starting at address 0081h.
00nnh+1	The command tail is terminated with a null byte (0).

The remainder of the command tail buffer is unused and is not initialized: it contains whatever was in it prior to command execution.

If the first entry in the command tail (ignoring leading whitespace) meets the requirements of a file specification, it is also parsed into the default file control block at base page address 005Ch.

If the first two entries in the command tail meet the requirements of a file specification, they are parsed into the primary and secondary partial file control blocks at base page addresses 005Ch and 006Ch.

If there are more than two file specifications in the command tail, or if a file specification is not the first or second entry, it is the sole responsibility of the programmer to parse the file specification into the desired file control block. File specification parsing may be automatically accomplished via C-function 152.

Program Termination

An application program may terminate in any of several different ways, all of which are equivalent:

By jumping to location 0000h, which causes a system warmstart.

By executing a C-function 0, which causes a system warmstart.

By executing a C-function 143, which causes a system warmstart.

By executing a T-function 0, which causes a system warmstart.

By executing a "RET" when the stack is balanced (in the same condition as on entry), causing a system warmstart.

By executing a "JP 0005" rather than a "CALL 0005" when the stack is balanced, which causes a system warmstart after executing the C-function invoked.

By executing a "JP 0050" rather than a "CALL 0050" when the stack is balanced, which causes a system warmstart after executing the T-function invoked.

By executing a C-function 47, which causes a system warmstart then executes the passed command line.

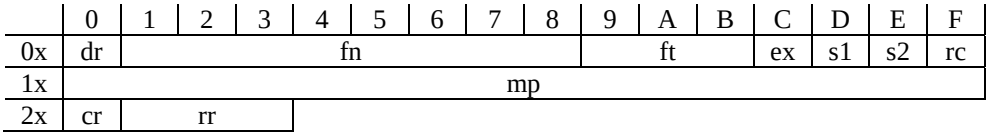
File Control Block

A file consists of a sequence of up to 1,048,576 128-byte physical records, and may be accessed only in integral increments of these physical records: up to 128 contiguous physical records may be read or written at one time, either sequentially or randomly. The maximum size of a file is thus defined as 134,217,728 bytes.

These physical records must not be confused with a logical or data record as specified by some programs and which may be of any length.

Files may be altered in size at any time, and TurboDOS automatically allocates and deallocates disk space to fit, but only in allocation block sized pieces. An allocation block may be 2048, 4096, 8192, or 16384 bytes long (1024 bytes on single sided, single density floppy disks only). Therefore, storing a new file containing one byte of data requires a 128-byte (1-record) write and occupies at least 2048 bytes on the disk!

All C-function and T-functions that operate upon a file require the existence of a file control block (FCB) whose address is passed in the DE register pair. This FCB consists of 36 bytes in the following format:



As can be seen, the FCB contains 10 fields:

fld	bytes	description
dr	00h	drive code
fn	01h–08h	filename and attributes f1-f8
ft	09h–0Bh	filetype and attributes t1-t3
ex	0Ch	extent counter
s1	0Dh	system byte 1 — flag byte
s2	0Eh	system byte 2 — extended extent counter
rc	0Fh	record count
mp	10h–1Fh	allocation map
cr	20h	current record number
rr	21h–23h	random record number

Each of these ten fields serves a specific purpose:

dr Drive Code

The drive code (dr) field contains a value indicative of the drive upon which the file is (to be) located:

00h = current drive
01h = drive A: 09h = drive I:
02h = drive B: 0Ah = drive J:

03h = drive C:	0Bh = drive K:
04h = drive D:	0Ch = drive L:
05h = drive E:	0Dh = drive M:
06h = drive F:	0Eh = drive N:
07h = drive G:	0Fh = drive O:
08h = drive H:	10h = drive P:

fn Filename and Attributes f1-f8

The filename (fn) field contains the ASCII filename of the file. If the filename is less than 8 characters, the remaining bytes of the field are set to ASCII space (20h).

Since an ASCII character requires only 7 bits, the remaining (most significant) bits of each byte of the filename field contain file attributes f1 through f8:

attr	byte	description
f1	01h	FIFO attribute
f2	02h	MS-DOS directory attribute
f3	03h	Unassigned attribute
f4	04h	Unassigned attribute
f5	05h	File locking attribute
f6	06h	File locking attribute
f7	07h	System attribute
f8	08h	System attribute

ft Filetype and Attributes t1-t3

The filetype (ft) field contains the ASCII filetype of the file. If the filetype is less than 3 characters, the remaining bytes of the field are set to ASCII space (20h).

As in the filename field, the remaining bits of each byte in the filetype field contain attributes t1 through t3:

attr	byte	description
t1	09h	Read only attribute
t2	0Ah	Global attribute
t3	0Bh	Archived attribute

ex Extent Counter

The extent counter (ex) field contains a modulo-32 number indicating the current extent of the file.

A file may contain 134,217,728 bytes (1,048,576 128-byte records). As an extent is usually 16K or 16,384 bytes (larger extent sizes are possible under some circumstances), there may be a maximum of 8192 extents in a file. A modulo-32 (5-bit) number is insufficient for this value: a 13–bit number is required. The problem is solved by using the system byte 2 (s2) field as an extended extent counter in the following manner:

s2	7	6	5	4	3	2	1	0				
ex									4	3	2	1	0
s2+ex	C	B	A	9	8	7	6	5	4	3	2	1	0

s1 System Byte 1 — Flag Byte

The system byte 1 (s1) field contains the system flag byte. Do not alter this byte under any circumstances or unpredictable errors may occur.

s2 System Byte 2 — Extended Extent Counter

The system byte 2 (s2) field contains the extended extent counter, and is used with the extent counter field as described above.

rc Record Count

The record count (rc) field contains the number of records in the current extent of the file. If the extent is full, this byte contains 80h.

mp Allocation Map

The allocation map (mp) field contains the allocation map as loaded from the directory. Do not alter this field in any way or reading and/or writing a record may access the wrong spot on the disk. An extreme danger exists in that if this field or any portion of it is set to zeros, the directory may be overwritten.

cr Current Record Number

The current record number (cr) field contains the number of the record in the current extent to which the file is currently positioned. This is the next record accessed by a sequential read or write.

rr Random Record Number

The random record number (rr) field contains the number of the absolute record (1 out of 1,048,510) which the file accesses at the next random read or write. The file may or may not be positioned to this record before, by, or after the access.

The random record number is a 20-bit number (occasionally interpreted as a 24-bit number with 0000 in the most significant four bits) arranged as follows:

23h	3	2	1	0															
22h					7	6	5	4	3	2	1	0							
21h													7	6	5	4	3	2	1	0
rr	13	12	11	10	0F	0E	0D	0C	0B	0A	09	08	07	06	05	04	03	02	01	00

In general, the application program must initialize bytes 0–12 of the FCB before opening, creating, or searching for a file. It must also zero FCB byte 32 before reading or writing a file sequentially from the beginning. In practice, the entire FCB is usually initialized.

When a file is opened or created, TurboDOS fills bytes 0–31 with information from the directory. When the file is closed, TurboDOS updates the directory with information from the FCB. It is critical, therefore, that the application program does not alter bytes 0–31 on an open file or the directory may be scrambled or damaged by file closure. Unlike CP/M, TurboDOS automatically closes any files a processor has open at warmstart.

A directory entry has the same basic structure as the first 32 bytes of an FCB, with the exception of byte 0: in a directory byte 0 contains the user area code, and in an FCB it contains the drive code.

When a file is deleted, byte 0 of its directory entry is made equal to E5h and its disk space is marked for reallocation: no other action takes place at that time. This means that changing byte 0 back to its original user area code and reallocating the disk space allows a deleted file to be restored, PROVIDED THAT NO OTHER DISK WRITES HAVE OCCURRED WHICH MAY ALTER OR OVERWRITE THE DIRECTORY OR THE DATA ON THE DISK. In a hashed directory, only the original user code may be used or the hashing algorithm will fail to find the file properly: in a non-hashed directory, any user file may be used.

File Specification Parsing

At the command line, a file specification must be in the following format:

uud:filename.typ

“uud:”

This is an optional user area/drive designation in one of the following six forms:

uu: d: uud: duu: uu:d: d:uu:

If either the user area or the drive is missing, the current user area or drive is assumed.

“filename”

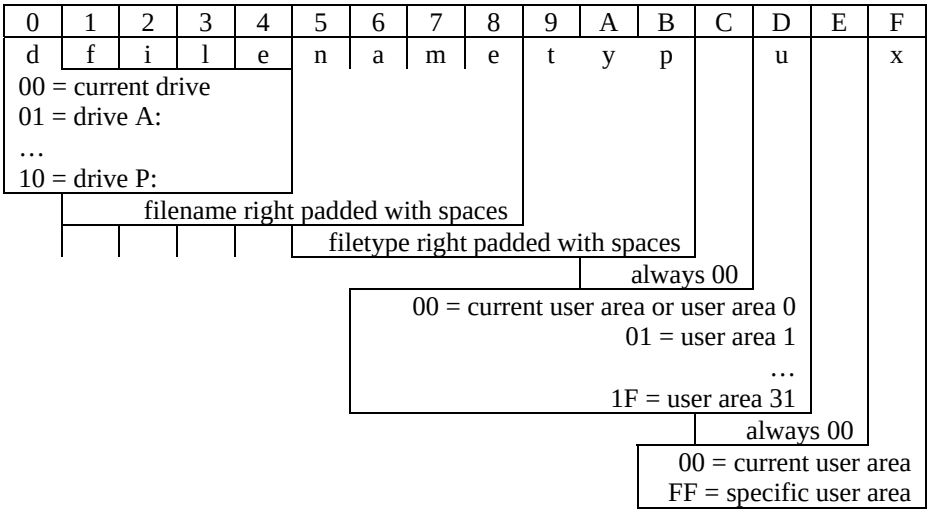
This is the filename itself, which may be from one to eight characters in length, and may consist of any characters except control codes, space, comma, period, semicolon, colon, equals, or the TurboDOS command separator (usually a backslash).

“.typ”

This is an option filetype designation, which may be from one to three characters in length and is set off from the filename by a period. The characters of the filetype are subject to the same rules as the filename.

At the command level, all lower case letters are converted to upper case.

The file specification is parsed into the first 16 bytes of the FCB as follows:



It should be noted that even though a file specification parsed from a command line disallows all lower case letters and control-codes, as well as certain special characters, any character at all may be placed into an FCB when programming, and that character becomes a part of the file name. Special care must be taken if control-codes are used, however, as they at minimum distort the screen display of the DIR command, and at worst may send the terminal south for the winter.

File Attributes

File attributes are stored in the high-order bits of the FCB name field (fn) and type field (ft), and are used to control how a file may be accessed:

Byte	Attr	aka	Description
1	f1	F	FIFO attribute
2	f2	D	MS-DOS directory attribute
3	f3	—	Not assigned, available for private use
4	f4	—	Not assigned, available for private use (used by COPY)
5	f5	—	File locking mode attribute
6	f6	—	File locking mode attribute
7	f7	—	Not assigned, reserved
8	f8	—	Not assigned, reserved
9	t1	R	Read only attribute
10	t2	G	Global attribute
11	t3	A	Archived attribute

Normally, only attributes f1-f4 and t1-t3 are recorded in the directory. A newly created file has all attributes cleared to 0, and no attribute is set unless intentionally so by C-function 30, Set File Attributes.

Under normal circumstances, the application program does not set the attributes in the FCB, the exceptions being attributes f5 and f6 under special circumstances, prior to opening or crating a file. Once opened, the attributes are automatically copied into the FCB from the directory, and should not be altered unless specifically for C-function 30.

f1 FIFO

Attribute f1, the FIFO attribute, denotes a special FIFO (first-in, first-out) file structure, not unlike that of an MP/M queue or a Unix pipe. FIFO files not only send and receive data in a special manner, they also have a unique header record that indicates the content and structure of the FIFO. Do not set the FIFO attribute on a normal file, but create the file as a FIFO in the first place by creating the file, creating and writing the special header record, then setting the FIFO attribute.

FIFO files may not be deleted, renamed, or copied unless the FIFO attribute is first cleared.

f2 MS-DOS Directory

Attribute f2, the MS-DOS directory attribute, marks a file as a special MS-DOS directory file, used to allow recognition of the drive and user area as a directory when operating under MS-DOS on a system networked in via TurboDOS/PC. There are usually two of these directory files per user per drive, with the special names "." and "..".

MS-DOS directory files cannot be deleted or renamed unless the MS-DOS directory attribute is first cleared.

f3 Not Assigned

Attribute f3 is not assigned and is free for use by an application program.

f4 Not Assigned

Attribute f4 is not assigned and is free for use by an application program.

There is an exception to this: when a very large file is fractionated and copied to a series of floppy disks using the “B” option of the COPY command, the file segments have attribute f4 set on the floppies in order to allow reconcatenation at a later time.

f5-f6 File Locking Mode

Attributes f5 and f6 work together to determine the file locking mode. Setting and/or clearing these attributes in the FCB prior to opening or creating a file causes that file to be opened or created in one of the four file locking modes:

f5	f6	Mode
0	0	Exclusive or Permissive (default)
1	0	Shared
0	1	Read only
1	1	Permissive or Exclusive

The operation of file and record locking is covered later.

Attributes f5 and f6 are not recorded in the directory.

f7-f8 Not Assigned, Reserved

Attributes f7 and f8 are not assigned, but are reserved for system use and are not recorded in the directory.

t1 Read only

Attribute t1, the read only attribute, marks a file as being read only. Read only files may not be written to, deleted, or renamed.

t2 Global

Attribute t2, the global attribute, marks a file as a system global file. If a global file is located on user area 0 of a given drive, then any user area on that drive may access the file. If a global file is located on user area 0 of the system (boot) drive, then any user area of any drive may access the file.

t3 Archived

Attribute t3, the archived attribute, marks a file as having been archived (copied via the “A” option of the COPY command). TurboDOS automatically clears the archived attribute during any write or rename operation.

User Areas

TurboDOS provides 32 distinct user areas: numbered 0–31. When programming, only the current user area of each drive is accessible. In order to do something on a different user area, the programmer must explicitly move to the desired user area, then explicitly return, via C-function 32.

Even though the current version of TurboDOS returns to the “current” or “original” user area when a program terminates, some earlier versions did not, and it is considered bad practice not to do your own housekeeping when programming.

When a file specification is parsed into an FCB, if the specification contains a user designation, that designation is parsed into byte 13 and an FFh is placed in byte 15 as a flag. It is the responsibility of the programmer to inspect byte 15 for an FFh and, if it is found, to extract the user area from byte 13 prior to any file process. Opening, creating, or searching for a file causes bytes 13 and 15 to be overwritten.

File Locking and Sharing

In a TurboDOS system, it is possible for more than one operator to access the same file at the same time. It is the responsibility of the programmer to prevent conflict when this multiple access occurs. This is accomplished in one of two ways: creating files that inherently do not cause a conflict; and implementing file and/or record locking.

The file locking and sharing facilities of TurboDOS are compatible with those used in MP/M-II, but offer considerable improvement over the basic MP/M-II scheme.

File level interlocks are provided through the use of four distinct file opening modes: exclusive, shared, read only, and permissive. These modes are dually controlled through the use of file attributes f5 and f6 and the system COMPAT (compatibility) flag byte.

Exclusive Mode

A file opened in exclusive mode by one process cannot be opened by any other process in any mode whatever until it is closed by the original process. The opening process is allowed to read, write, or extend the file.

A file cannot be opened in exclusive mode if it is already opened by another process in any mode whatever.

Shared Mode

A file opened in shared mode by one process may be opened in shared mode by any number of other processes simultaneously. All processes may read, write, or extend the file.

If a process extends the file in shared mode, the new records are immediately available to all other processes that have the file open.

Shared mode is the only mode in which record locking may occur.

Read Only Mode

A file opened in read only mode by one process may be opened in read only mode by any number of other processes simultaneously. All processes may read the file, but no process may write or extend the file.

It is important that the read only file opening mode not be confused with the read only attribute of a file.

Permissive Mode

A file opened in permissive mode by one process may be opened in permissive mode by any number of other processes simultaneously. All processes may read the file at any time, but if any process writes or extends the file then that process gains an exclusive write lock on the file and only that process and no other may write or extend the file until that process closes the file, releasing the exclusive write lock.

If a process extends the file in permissive mode, the new records are immediately available to all other processes that have the file open.

Record locking is controlled by the use of explicit record locking requests in the application program. These requests are honored only in shared file opening mode. Each process must explicitly lock a record or records via C-function 42, then explicitly unlock the record or records via C-function 43 after updating.

If an attempt is made to lock a record already locked by another process, the calling process either receives an error code, which it must trap and process, or is suspended until the lock can take place, depending upon the status of the suspend bit of the COMPAT byte.

Extension of a file in shared mode should be accomplished by locking record n+1, where n is the last record in the file. The new record may then be appended safely, then unlocked.

COMPAT

TurboDOS' file and record locking is designed to be compatible with MP/M-II, yet still offer a considerable degree of flexibility. This is accomplished though the use of the COMPAT (compatibility) byte, which is a series of six flags, one for each of the bits in the byte (bits 1 and 0 are not used).

bit 7 Permissive Flag

If the permissive flag is not set, then MP/M-II rules are followed and the file opening mode attributes f5 and f6 act as follows:

f5	f6	Mode
0	0	Exclusive (default)
1	0	Shared
0	1	Read Only
1	1	Permissive

If the permissive flag is set, then the file opening mode attributes act as follows:

f5	f6	Mode
0	0	Permissive (default)
1	0	Shared
0	1	Read Only
1	1	Exclusive

In this manner, the permissive flag controls the default opening mode of a file, the mode in which neither f5 nor f6 is set, and provides basic file locking automatically.

bit 6 Suspend Flag

If the suspend flag is not set, then MP/M-II rules are followed and an attempt to lock a record already locked by another process results in an error. The application program must trap and process this error, or the process is dropped to the operating system error prompt with a possibility of warmstart.

If the suspend flag is set, then an attempt to lock a record already locked by another process results in a suspension of the calling process until the other process has released the lock, at which time the calling process may proceed. Multiple process queuing can occur as required.

The suspend flag is limited to record locking errors in files opened in shared mode, and has no effect on file lock conflicts, which always return an error.

bit 5 Global Write Flag

If the global write flag is not set, then MP/M-II rules apply and a process not in user area 0 may read but not write or extend global files in user area 0.

If the global write flag is set, then a process not in user area 0 may, read, write, or extend global files in user area 0.

bit 4 Mixed Mode Flag

If the mixed mode flag is not set, then MP/M-II rules apply and a process may not open a file in read only mode that another process has open in shared mode or vice versa. Read only and shared modes are mutually exclusive.

If the mixed mode flag is set, then any number of processes may have files open in either shared or read only mode simultaneously.

bit 3 Logical Flag

If the logical flag is not set, MP/M-II rules apply and the FCB random record number (bytes 33–35) is interpreted by C-functions 42 and 43 (lock and unlock record) as the relative number of a 128-byte record, and causes the file to be positioned to that record.

If the logical flag is set, then the FCB random record number is interpreted by C-functions 42 and 43 as an arbitrary 24-bit logical record number which is not validated and does not cause file positioning.

bit 2 Global Inhibit Flag

If the global inhibit flag is not set, then MP/M-II rules apply and a process not in user area 0 may access global files in user area 0.

If the global inhibit flag is set, then a process not in user area 0 may not access global files in user area 0.

The default settings of the COMPAT byte may be altered for any given program through the use of T-function 13, but return to the default settings upon program termination or any warmstart.

FIFO Files

To facilitate communications between processes, TurboDOS supports a special kind of file called a FIFO (first-in, first-out), similar in concept to a Unix pipe. FIFO files are opened, closed, read and written exactly like ordinary sequential files. However, a record written to a FIFO file is always appended to the end, and a record read from a FIFO file is always taken from the beginning and removed.

A FIFO file is differentiated from other files by the presence of the FIFO attribute. The first record of a FIFO file contains a special header:

byte	function
0	type of FIFO: 0 = RAM; –1 = disk
1	mode: 0 = return error; –1 = suspend
2–3	maximum size (records)
4–5	current size (records)
6–7	number of last record read
8–9	number of last record written

The rest of the header record is unused but reserved.

RAM-resident FIFO files provide high speed but limited size, no more than 127 data records, usually much less, and occupy memory space in the disk buffer area of the system file server. A RAM-resident FIFO file may be lost in the event of system failure, as only the header resides on the disk.

A disk-resident FIFO file is slower, but may contain up to 65535 data records, and is flushed from the buffers in a normal manner.

Unlike other files, FIFO files are of fixed length (even when empty) and cannot be extended. Normally, reading an empty FIFO file returns an end of file error, while writing to a full FIFO file returns a disk full error. However, if the FIFO file is set to suspend, the reading an empty FIFO file or writing to a full FIFO file causes the calling process to be suspended until the FIFO file is not empty or not full.

A FIFO file may be accessed directly, bypassing normal FIFO operation, by the use of C-functions 33 and 34 (read and write random). Only the disk-resident header of a RAM FIFO file may be accessed in this manner.

An attempt to create a FIFO file via C-function 22 is treated as an open, C-function 15. An attempt to delete a FIFO file via C-function 19 is ignored. A FIFO file may only be deleted by first clearing the FIFO attribute.

TurboDOS supports programs that use MP/M-II queue files by simulating those files through the use of RAM FIFO files. The simulated queue files are created via C-function 134 (create queue), giving the FIFO file the designated name and the filetype ".QUE".

C-Functions

The C-functions supported by TurboDOS are analogous to the corresponding CP/M and MP/M BDOS functions with minor exceptions.

To invoke a C-function, a program executes a "CALL 0005" instruction with the function number in register C.

Byte length arguments are normally passed in register E. Word length arguments are normally passed in register pair DE.

Byte length arguments are normally returned in register A and duplicated in register L. Word length arguments are normally returned in register pair HL, with register H duplicated in register B and register L duplicated in register A.

All C-functions must be assumed to destroy the current contents of all normal registers except IX, IY, and the alternative register set.

If a C-function call is made to an unsupported function, a 00h is returned in register A.

A listing of supported C-functions in numerical order is given separately.

T-Functions

The T-functions, TurboDOS unique functions, supplement the C-functions.

To invoke a T-function, a program executes a "CALL 0050" instruction the function number in register C. Arguments are passed and values returned in the other registers, as described below for each T-function.

All T-functions must be assumed to destroy the current contents of all normal registers except IX, IY, and the alternative register set.

If a T-function call is made to an unsupported function, a 00h is returned in register A.

A listing of T-functions in numerical order is given separately.

C-Function Listing

A listing of all supported C-functions in numerical order, one function per page.

C-Function 0 System Reset

Call: Return:

C = 00h

The System Reset function terminates the calling program and executes a warmstart. Program termination is more commonly performed by executing a “JP 0000”, which has exactly the same effect.

Program termination ends any active print job, close any open files, release any locked records, and restore the user number and drive that were in effect at the time the program was originally loaded into TPA.

C-Function 1 Console Input

Call:	Return:
C = 01h	A = input character

The Console Input function fetches the character available from the console keyboard, waiting if necessary, and returns it in register A. All printable (graphic) characters, carriage return, linefeed, and backspace are echoed to the console screen. Horizontal tabs are expanded into multiple spaces based upon tab stops at every eighth column.

C-Function 2 Console Output

Call: Return:

E = output character
C = 02h

The Console Output function sends the character passed in register E to the console screen. Horizontal tabs are expanded into multiple spaces based upon tab stops at every eighth column.

C-Function 3 Raw Console Input

Call:	Return:
C = 03h	A = input character

The Raw Console Input function fetches the next available character from the console keyboard, waiting if necessary, and returns it in register A. Input characters are not echoed to the console screen.

This function is compatible with MP/M: in CP/M 2.2, it is Input from Reader Device; in CP/M 3.x, it is Auxiliary Input.

C-Function 4 Raw Console Output

Call: Return:

E = output character
C = 04h

The Raw Console Output function sends the character passed in register E to the console screen. Horizontal tabs are not expanded.

This function is compatible with MP/M: in CP/M 2.2, it is Output to Punch Device; in CP/M 3.x, it is Auxiliary Output.

C-Function 5 List Output

Call: Return:

E = output character
C = 05h

The List Output function sends the character passed in register E to the list device. Horizontal tabs are not expanded.

C-Function 6 Direct Console I/O

Call: Status/Input	Return:
E = FFh C = 06h	A = input chr; 00h if no chr available A = 00h if no chr available
Call:	**Return:**
E = FEh (status) C = 06h	A = 00h if no chr available A = FFh if chr available
Call:	**Return:**
E = FDh (input) C = 06h	A = input character
Call:	**Return:**
E = character (output) C = 06h	

The Direct Console I/O function performs one of four possible sub-functions, depending upon the argument passed in register E.

If E = FFh (–1), then the function fetches any currently available character from the console keyboard, without waiting, and return it in register A. If no character is currently available, a 00h is returned in register A. Input characters are not echoed to the console screen.

If E = FEh (–2), then the function returns the console status in register A: 00h if a character is available; FFh (–1) if it is not. This is equivalent to C-function 11 (Get Console Status).

If E = FDh (–3), then the function fetches the next available character from the console keyboard, waiting if necessary, and return it in register A. Input characters are not echoed to the console screen. This is equivalent to C-function 3 (Raw Console Input).

For other values of E, the function sends the character passed in register E to the console screen. Horizontal tabs are not expanded. This is equivalent to C-function 4 (Raw Console Output).

C-Function 7 Get I/O Byte

Call:	Return:
C = 07h	A = current I/O byte

The Get I/O Byte function returns the current value of the I/O byte, located at address 0003h, in register A.

This function is supported only if the module CPMSUP has been incorporated into the operating system.

C-Function 8 Set I/O Byte

Call: Return:

E = new I/O byte
C = 08h

The Set I/O Byte function sets the I/O byte, located at address 0003h, to the value passed in register E.

This function is supported only if the module CPMSUP has been incorporated into the operating system.

C-Function 9 Print String

Call: Return:

DE = string address
C = 09h

The Print String function sends a string of characters to the console screen. The string may be of any length, and must be terminated by a reserved delimiter, which is normally the dollar sign "$", but which may be changed via C-function 110 (Get/Set Output Delimiter). Horizontal tabs are expanded into multiple spaces, based upon tab stops at every eighth column.

C-Function 10 Read Console Buffer

Call:
DE = buffer address
C = 0Ah

Return:

The Read Console Buffer function fetches an entire line of edited input from the console keyboard. The input buffer whose address must be passed in register DE has the following structure:

Bytes	Direction	Description
00h	passed	max input size (n)
01h	returned	actual input (0 – n)
02h to n+1	returned	input characters

The first byte of the buffer must be preset to the maximum number of characters allowed in the input line. Input is accepted until either a carriage return is entered. A full buffer is indicated by a console “beep,” and excess characters are ignored. Input characters are echoed to the console screen, but tabs are not expanded.

The input line may be edited via the input line editor incorporated into the operating system. TurboDOS' default input line editor allows the erasure of a character from the end of the line via the backspace (control-H) or delete keys, or the erasure of the entire line via the control-U or control-X keys.

Upon return, the second byte of the buffer contains the actual number of input characters entered.

The input characters begin at the third byte of the buffer. The terminating carriage return is neither stored in the buffer nor included in the count.

C-Function 11 Get Console Status

Call:	Return:
C = 0Bh	A = 00h if no chr available A = FFh if chr available

The Get Console Status function checks the console to see whether or not a character is available for input, and return an FFh (–1) in register A if one is, or a 00h if one is not. The character is neither fetched nor consumed.

C-Function 12 Return CP/M Version

Call:	Return:
C = 0Ch	H = 00h: CP/M (not MP/M) L = 31h: CP/M version DE = network address

The Return Version function returns a CP/M compatibility code, the CP/M version number and the network address of the calling process.

A 00h is always returned in register H, indicating compatibility with CP/M but not MP/M.

The compatible CP/M version, usually 31h for version 3.1, is returned in register L. The value returned in register L is dependent upon the value of the CPMVER patch point set during system generation.

The network address of the calling process is returned in register pair DE (circuit in D and node in E). This address is identical to the first address in the circuit assignment table (CKTAST) established during system generation, and is unique in a properly configured network.

C-Function 13 Reset Disk System

Call: C = 0Dh

Return:

In TurboDOS, the only effect of the Reset Disk System function is to reset the current DMA address to its default of 0080h.

C-Function 14 Select Disk

Call: Return:

E = disk drive code
C = 0Eh

The Select Disk function causes the disk drive whose code is passed in register E to become the current (default) drive. This drive is the drive used in subsequent file operation where the FCB drive code is 00h.

The disk drive code to be passed in register E must be:

00h = A:	04h = E:	08h = I:	0Ch = M:
01h = B:	05h = F:	09h = J:	0Dh = N:
02h = C:	06h = G:	0Ah = K:	0Eh = O:
03h = D:	07h = H:	0Bh = L:	0Fh = P:

C-Function 15 Open File

Call:	Return:
DE = FCB address	A = 00h if opened
C = 0Fh	A = FFh if file not found

The Open File function opens the file specified by the FCB whose address is passed in register pair DE. The file must exist on the specified drive, and must exist either on the current user area, or as a global file on user area 0.

Only bytes 0 through 12 of the FCB (drive, filename, filetype, and extent) need be initialized, with byte 12 (extent) normally set to 00h. After opening, the remainder of the FCB contains information derived from the directory.

If a file is to be opened in other than the default file locking mode, then attributes f5 and f6 must be set in the FCB prior to calling this function. See the section on file and record locking to determine attribute settings.

If the FCB current record field (byte 32) is set to FFh (–1), this function returns the byte count of the last record of the file in the current record field. The calling program should zero the current record field before executing sequential reads or writes.

C-Function 16 Close File

Call:	Return:
DE = FCB address	A = 00h if closed
C = 10h	A = FFh if file not found

The Close File function closes the previously opened file specified by the FCB whose address is passed in register pair DE. The directory is updated as required, and any locks on the file or its records are released.

If the FCB attribute f5 is set, this function performs a partial close, updating the directory but leaving the file open.

C-Function 17 Search for First

Call:	Return:
DE = FCB address	A = entry number if found
C = 11h	A = FFh if file not found

The Search for First function scans the directory for the first entry which matches the file specified by the FCB whose address is passed in register pair DE. The file is searched for only on the specified drive, and on the current user area or as a global file on user area 0.

Only bytes 0 through 12 of the FCB (drive, filename, filetype, and extent) need be initialized, with byte 12 (extent) normally set to 00h. If any character of the filename or filetype is an ASCII question mark (3Fh), it is treated as a wildcard and provides a match to any character in that position.

If the search is successful, this function returns a directory record, containing four 32-byte directory entries, at the current DMA address, and also returns a value of 00h through 03h in register A, indicating which of the four directory entries provided the match. If the search is not successful, FFh (–1) is returned in register A.

If this function is successful, then C-function 18 (Search for Next) may be called repeatedly to locate all remaining matches in the directory.

A special case exists if an ASCII question mark (3Fh) is placed in byte 0 of the FCB: the remainder of the FCB is ignored and the first directory entry of the current drive (usually the drive label) is returned. C-function 18 (Search for Next) then finds each subsequent directory entry, regardless of user area. Even deleted entries are returned in this case.

C-Function 18 Search for Next

Call:	Return:
DE = FCB address	A = entry number if found
C = 12h	A = FFh if file not found

The Search for Next function scans the directory, starting at the current location, for the next entry which matches the file specified by the FCB whose address is passed in register pair DE. The file is searched for only on the specified drive, and on the current user area or as a global file on user area 0.

Only bytes 0 through 12 of the FCB (drive, filename, filetype, and extent) need be initialized, with byte 12 (extent) normally set to 00h. If any character of the filename or filetype is an ASCII question mark (3Fh), it is treated as a wildcard and provides a match to any character in that position.

If the search is successful, this function returns a directory record, containing four 32-byte directory entries, at the current DMA address, and also returns a value of 00h through 03h in register A, indicating which of the four directory entries provided the match. If the search is not successful, FFh (–1) is returned in register A.

C-Function 19 Delete File

Call:	Return:
DE = FCB address	A = 00h if deleted
C = 13h	A = FFh if not deleted

The Delete File function deletes the file specified by the FCB whose address is passed in register pair DE. The file must exist on the specified drive, and must exist either on the current user area, or as a global file on user area 0.

Only bytes 0 through 11 of the FCB (drive, filename, and filetype) need be initialized. If any character of the filename or filetype is an ASCII question mark (3Fh), it is treated as a wildcard and provides a match to any character in that position.

A process may delete a file that it has open, in which case an implicit close is executed before the file is deleted. It is not, however, permitted to delete a file that another process has open, nor a file that has the read only, FIFO, or MS-DOS directory attributes set.

If attribute f5 is set, this function performs no operation, yet returns 00h in register A: this provides compatibility with M/PM, where the f5 attribute causes XFCBs to be deleted.

C-Function 20 Read Sequential

Call:	Return:
DE = FCB address	A = 00h if good read
C = 14h	A = 01h if at end of file
	A = 80h if current record invalid

The Read Sequential function reads into memory at the current DMA address the next 1 to 128 sequential records from the previously opened file specified by the FCB whose address is passed in register pair DE.

The FCB extent and current record fields are used to determine the record to be read. After a record is read, the current record field is incremented and the file is positioned to that record. If the increment causes a current record field overflow, the next extent is opened and the current record field is reset prior to positioning.

The number of records to be read at one time is previously set via C-function 44 (Set Multi-Record Count): the default is one record.

If the file to be read is a FIFO file, a record is read from the beginning and removed from the FIFO file. Reading from an empty FIFO file either suspends or returns an end of file code (01h) in register A, depending upon the mode byte in the FIFO file header. However, if FCB attribute f5 is set, reading from an empty FIFO file always returns an end of file code.

C-Function 21 Write Sequential

Call:
DE = FCB address
C = 15h

Return:
A = 00h if good write
A = 01h if file too large
A = 02h if disk full or
A = file read only
A = 08h if record locked
A = 80h if current record invalid
A = FFh if directory full

The Write Sequential function writes from memory at the current DMA address the next 1 to 128 sequential records to the previously opened file specified by the FCB whose address is passed in register pair DE.

The FCB extent and current record fields are used to determine the record to be read. After a record is read, the current record field is incremented and the file is positioned to that record. If the increment causes a current record field overflow, the next extent is opened or created and the current record field is reset prior to positioning.

The number of records to be read at one time is previously set via C-function 44 (Set Multi-Record Count): the default is one record.

If the file to be written is a FIFO file, a record is appended to the end of the FIFO file. Writing to a full FIFO file either suspends or returns a disk full code (02h) in register A, depending upon the mode byte in the FIFO file header. However, if FCB attribute f5 is set, writing to a full FIFO file always return a disk full code.

C-Function 22 Create File

Call:	Return:
DE = FCB address	A = 00h if created
C = 16h	A = FFh if directory full, file already exists, or FCB invalid

The Create File function creates a new (empty) file specified by the FCB whose address is passed in register pair DE. The file must not exist on the specified drive and current user area.

Only bytes 0 through 12 of the FCB (drive, filename, filetype, and extent) need be initialized, with byte 12 (extent) set to 00h. After creation, the file is left open.

If a file is to be created in other than the default file locking mode, then attributes f5 and f6 must be set in the FCB prior to calling this function: a file may not be created in read only mode. See the section on file and record locking to determine attribute settings.

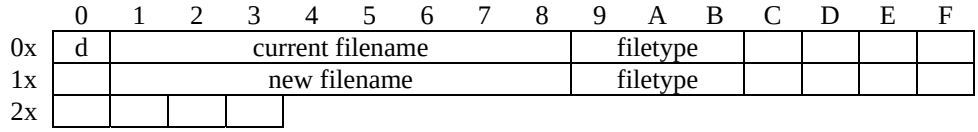
The calling program should zero the current record field before executing sequential reads or writes.

C-Function 23 Rename File

Call:	Return:
DE = FCB address	A = 00h if renamed
C = 17h	A = FFh if file not found, in use or invalid name

A program may rename a file that it has open, in which case a close is performed implicitly before the file is renamed. However, a program is not permitted to rename a file that another process has open, nor a file that has the read only attribute.

Bytes 00h through 0Bh (0 through 11) of the FCB are used to specify the current drive, name and type of the file, while bytes 11h through 1Bh (17 through 27) are used to specify the new name and type: all remaining bytes of the FCB are ignored. Both specifications must be explicit: wildcard characters are not permitted. The special FCB format is as follows:



A process may rename a file that it has open, in which case an implicit close is executed before the file is renamed. It is not, however, permitted to rename a file that another process has open, nor a file that has the read only, FIFO, or MS-DOS directory attributes set.

If attribute f5 is set, this function performs no operation, yet returns 00h in register A: this provides compatibility with M/PM, where the f5 attribute causes XFCBs to be deleted.

C-Function 24 Return Login Vector

Call:	Return:
C = 18h	HL = login vector

The Return Login Vector function tests the status of all sixteen possible logical drives and return a 16-bit vector in register pair HL denoting the results.

Each bit of the returned vector designates the status of a different drive, with bit 0 indicating drive A: status and bit 15 indicating drive P: status. A 1 in the appropriate bit indicates the drive is on line and ready for access: a 0 indicates the drive is not ready or not assigned.

This function is supported only if the module CPMSUP has been incorporated into the operating system.

C-Function 25 Return Current Disk

Call:	Return:
C = 19h	A = disk drive code

The Return Current Disk function returns the identity of the current (default) disk drive in register A.

The disk drive code to be returned in register A is:

00h = A:	04h = E:	08h = I:	0Ch = M:
01h = B:	05h = F:	09h = J:	0Dh = N:
02h = C:	06h = G:	0Ah = K:	0Eh = O:
03h = D:	07h = H:	0Bh = L:	0Fh = P:

C-Function 26 Set DMA Address

Call: Return:

DE = DMA address
C = 1Ah

The Set DMA Address function causes the memory address passed in register pair DE to become the start of the DMA record buffer to be used for subsequent file read and write operations. In a banked memory system, the DMA address is always interpreted as an address in the same memory bank as the TPA.

Whenever a program is loaded into the TPA, the DMA address is initialized to 0080h, the address of the command tail and default DMA record buffer. The DMA address is returned to this default via C-function 13 (Reset Disk System).

C-Function 27 Get ALV Address

Call:	Return:
C = 1Bh	HL = 0000h

The Get ALV Address function performs no operation under TurboDOS: under CP/M, it returns the address of the memory resident allocation vector for the current disk.

C-Function 28 Write Protect Disk

Call: C = 1Ch

Return:

The Write Protect Disk function sets the current (default) disk drive as read only, preventing any program from writing to the disk. C-function 37 (Reset Drive) must be used before the disk again allows writes.

Unlike CP/M, TurboDOS does not re-enable writing after a warmstart or C-functions 0 (System Reset) or 13 (Reset Disk System). Consequently, write protection of a disk drive is not nearly as transient as it is under CP/M.

This function is supported only if the module CPMSUP has been incorporated into the operating system.

C-Function 29 Get Read-Only Vector

Call:	Return:
C = 1Dh	HL = read only vector

The Get Read only Vector function tests the read only status of all sixteen possible logical drives and return a 16-bit vector in register pair HL denoting the results.

Each bit of the returned vector designates the read only status of a different drive, with bit 0 indicating drive A: status and bit 15 indicating drive P: status. A 1 in the appropriate bit indicates the drive is read only: a 0 indicates the drive is read/write.

This function is supported only if the module CPMSUP has been incorporated into the operating system.

C-Function 30 Set File Attributes

Call:	Return:
DE = FCB address	A = 00h if set
C = 1Eh	A = FFh if file not found or in use

The Set File Attributes function searches the directory for the file specified by the FCB whose address is passed in register pair DE, and updates the directory attributes for that file to those found in the FCB.

Only bytes 0 through 11 of the FCB (drive, filename, and filetype) need be initialized. Wildcard characters are not permitted.

A process may set attributes on a file that it has open, in which case an implicit close is executed before the attributes are updated. It is not, however, permitted to delete a file that another process has open.

If attribute f6 is set, this function updates the last record byte count of the file. The count is obtained from the current record field (byte 32) of the FCB, and stored in the flag field (byte 13) of the directory entry.

C-Function 31 Get DPB Address

Call:	Return:
C = 1Fh	HL = DPB address

The Get DPB Address function causes TurboDOS to construct a CP/M-style Disk Parameter Block (DPB) for the current drive and return its address in register pair HL.

This function is supported only if the module CPMSUP has been incorporated into the operating system.

C-Function 32 Get/Set User Number

Call:	Return:
E = FFh (Get) C = 20h	A = user area number

Call:	Return:
E = user number (Set) C = 20h	

The Get/Set User Number function returns the current user area number in register A if FFh (–1) is passed in register E.

This function sets the current user area to the number passed in register E if the modulo-32 number passed in register E is not FFh and the calling process is privileged: a request by a non-privileged process is ignored.

C-Function 33 Read Random

Call:	Return:
DE = FCB address	A = 00h if good read
C = 21h	A = 01h if unwritten data
	A = 03h if extent change error
	A = 04h if unwritten extent
	A = 80h if random record invalid

The Read Random function reads into memory at the current DMA address the next 1 to 128 consecutive records from the previously opened file specified by the FCB whose address is passed in register pair DE.

The 20-bit FCB random record number (bytes 33 through 35) is used to determine the record to be read. After a record is read, the random record field is incremented and the file is positioned to that record. In addition, the FCB extent and current record field are set to correspond with the random record that was read. Unlike C-function 20 (Read Sequential), this function does not increment the current record field after reading. Thus, if the Read Random function is followed by a Read Sequential or Write Sequential, the same record is re-accessed.

The number of records to be read at one time is previously set via C-function 44 (Set Multi-Record Count): the default is one record.

C-Function 34 Write Random

Call:	Return:
DE = FCB address	A = 00h if good read
C = 22h	A = 02h if disk full or write protected
	A = 03h if extent change error
	A = 05h if directory full
	A = 06h if random record invalid
	A = 08h if record locked

The Write Random function writes from memory at the current DMA address the next 1 to 128 consecutive records to the previously opened file specified by the FCB whose address is passed in register pair DE.

The 20-bit FCB random record number (bytes 33 through 35) is used to determine the record to be written. After a record is written, the random record field is incremented and the file is positioned to that record. In addition, the FCB extent and current record field are set to correspond with the random record that was written. Unlike C-function 20 (Read Sequential), this function does not increment the current record field after writing. Thus, if the Write Random function is followed by a Read Sequential or Write Sequential, the same record is re-accessed.

The number of records to be written at one time is previously set via C-function 44 (Set Multi-Record Count): the default is one record.

C-Function 35 Compute File Size

Call:	Return:
DE = FCB address	A = 00h if size computed A = FFh if file not found

The Compute File Size function searches the directory for the file specified by the FCB whose address is passed in register pair DE. If the file is found, this function sets the FCB random record field (bytes 33 through 35) to a value one greater than the record number of the last record in the file. Thus, a succeeding Write Random function (34) appends an additional record at the end of the file.

This function produces correct results whether or not the file is open. If the file is closed, only bytes 0 through 11 of the FCB (drive, filename, and filetype) need be initialized.

C-Function 36 Set Random Record

Call:
DE = FCB address
C = 24h

Return:

The Set Random Record function returns the current file position of a previously opened file in the random record field (bytes 33–35) of the FCB. The file position is determined from the values of the FCB extent, spec2, and current record fields (bytes 12, 14, and 32). Since the Read Sequential (20) and Write Sequential (21) functions do not update the random record field of the FCB, this function is useful when switching from sequential to random access.

C-Function 37 Reset Drive

Call:
DE = reset vector
C = 25h

Return:

The Reset Drive function write enables any of the sixteen possible logical drives according to a 16-bit reset vector passed in register pair DE.

Each bit of the passed vector designates a different drive, with bit 0 indicating drive A: status and bit 15 indicating drive P: status. A 1 in the appropriate bit indicates the drive is to be reset: a 0 indicates it is not.

This function is supported only if the module CPMSUP has been incorporated into the operating system.

C-Function 40 Write Random with Zero Fill

Call:	Return:
DE = FCB address	A = 00h if good read
C = 28h	A = 02h if disk full or write protected
	A = 03h if extent change error
	A = 05h if directory full
	A = 06h if random record invalid
	A = 08h if record locked

Under TurboDOS, the Write Random with Zero Fill function is identical to C-function 34 (Write Random) and writes from memory at the current DMA address the next 1 to 128 consecutive records to the previously opened file specified by the FCB whose address is passed in register pair DE.

The 20-bit FCB random record number (bytes 33 through 35) is used to determine the record to be written. After a record is written, the random record field is incremented and the file is positioned to that record. In addition, the FCB extent and current record field are set to correspond with the random record that was written. Unlike C-function 20 (Read Sequential), this function does not increment the current record field after writing. Thus, if the Write Random function is followed by a Read Sequential or Write Sequential, the same record is re-accessed.

The number of records to be written at one time is previously set via C-function 44 (Set Multi-Record Count): the default is one record.

C-Function 42 Lock Record

Call:	Return:
DE = FCB address	A = 00h if record locked
C = 2Ah	A = 01h if unwritten data
	A = 03h if extent change error
	A = 04h if unwritten extent
	A = 06h if random record invalid
	A = 08h if already locked

The Lock Record function attempts to obtain a lock on 1 to 128 consecutive records of a previously opened shared mode file specified by the FCB whose address is passed in register pair DE, starting with the record designated by the 20-bit FCB random record number (bytes 33 through 35). If the file is not opened in shared mode, no operation takes place but a 00h is returned in register A.

The file is positioned to the specified record, unless the COMPAT byte logical flag (bit 3) is set or the file is a FIFO file. If the record is already locked by another process, this function either suspends or returns an error depending on the setting of the COMPAT byte suspend flag (bit 6).

The number of records to be locked at one time is previously set via C-function 44 (Set Multi-Record Count): the default is one record.

If the FCB random record field is set to the 24-bit value 0FFFFFFH, then this function attempts to obtain an all inclusive lock on all records of the file at once. In this case, no positioning is performed.

C-Function 43 Unlock Record

Call:	Return:
DE = FCB address	A = 00h if record unlocked
C = 2Bh	A = 01h if unwritten data
	A = 03h if extent change error
	A = 04h if unwritten extent
	A = 06h if random record invalid

The Unlock Record function unlocks 1 to 128 consecutive records of a previously opened shared mode file specified by the FCB whose address is passed in register pair DE, starting with the record designated by the 20-bit FCB random record number (bytes 33 through 35). Attempting to unlock a record not previously locked does not produce an error. If the file is not opened in shared mode no operation takes place but a 00h is returned in register A.

The file is positioned to the specified record, unless the COMPAT byte logical flag (bit 3) is set or the file is a FIFO file. If the record is already locked by another process, this function either suspends or returns an error depending on the setting of the COMPAT byte suspend flag (bit 6).

The number of records to be unlocked at one time is previously set via C-function 44 (Set Multi-Record Count): the default is one record.

If the FCB random record field is set to the 24-bit value 0FFFFFFH, then this function releases an all inclusive lock, but does not affect any records individually locked. In this case, no positioning is performed.

C-Function 44 Set Multi-Record Count

Call:	Return:
E = number of records	A = 00h
C = 2Ch	

The Set Multi-Record Count function enables a process to manipulate from 1 to 128 records at one time during subsequent file operations. This function affects the following C-functions:

20	Read Sequential
21	Write Sequential
33	Read Random
34	Write Random
40	Write Random with Zero Fill
42	Lock Record
43	Unlock Record

and determines the number of consecutive 128-byte records to be read, written, locked or unlocked.

Once set, the specified record count remains in effect until changed by another Set Multi-Record Count function. The initial default value is one record.

C-Function 46 Get Disk Free Space

Call:	Return:
E = disk drive code	A = 00h
C = 2Eh	

The Get Disk Free Space function determines the amount of free space on the specified disk drive, and returns a 24-bit binary value (the number of free 128-byte records) as a three byte quantity stored at the current DMA address, least significant byte first.

The disk drive code is as follows:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

C-Function 47 Chain to Program

Call:
E = 00h (reversion)
C = 2Fh

Return:

Call:
E = FFh (retention)
C = 2Fh

Return:

The Chain to Program function allows chaining from one program to another. The calling program must pass a valid TurboDOS command line, terminated by a null byte, in the current DMA buffer. This function then terminates the calling program, revert to the original drive and user area if 00h is passed in register E or retain the current drive and user area if FFh (–1) is passed in register E, and then execute the passed command line. The commands are echoed to the console as they are executed, unless the command line starts with a command separator character.

C-Function 104 Set Date and Time

Call: Return:

DE = date/time packet address
C = 68h

The Set Date and Time function sets the system date and time to that contained in a 4-byte date/time packet whose address is passed in register pair DE. The date/time packet has the following structure:

byte	Description
0–1	Julian date: 0000h = 31 December 1977
2	Hours: 2 BCD digits
3	Minutes: 2 BCD digits

Seconds are set to zero.

C-Function 105 Get Date and Time

Call:	Return:
DE = date/time packet address	A = seconds: 2 BCD digits
C = 69h	

The Get Date and Time function returns the system date and time and in a 4-byte date/time packet whose address is passed in register pair DE. The date/time packet has the following structure:

byte	Description
0–1	Julian date: 0000h = 31 December 1977
2	Hours: 2 BCD digits
3	Minutes: 2 BCD digits

Seconds are returned in register A. C-function 155 (Get Date and Time) is identical save that seconds are returned in the date/time packet.

C-Function 107 Return CP/M Serial Number

Call:
DE = s/n field address
C = 6Bh

Return:

Under TurboDOS, the Return Serial Number function returns six zeros to the 6-byte CP/M serial number field whose address is passed in register pair DE.

This function is supported only if the module CPMSUP has been incorporated into the operating system.

C-Function 108 Get/Set Program Return Code

Call:	Return:
DE = FFFFh (Get) C = 6Ch	HL = program return code

Call:	Return:
DE = program return code (Set) C = 6Ch	

The Get/Set Program Return Code function allows one program to pass a value to another program via the 16-bit program return code.

If FFFFh is passed in register pair DE, this function gets the current program return code and returns it in register pair HL.

If any value except FFFFh is passed in register pair DE, this function sets the current program return code to that value.

C-Function 110 Get/Set Program Output Delimiter

Call:	Return:
DE = FFFFh (Get) C = 6Eh	A = output delimiter

Call:	Return:
E = output delimiter (Set) C = 6Eh	

The Get/Set Output Delimiter function allows a process to inspect or alter the output string delimiter used by C-function 9 (Print String). The default output string delimiter is the ASCII dollar sign "$" (24h).

If FFFFh is passed in register pair DE, this function gets the current output delimiter and returns it in register A.

If any value other than FFFFh is passed in register pair DE, this function sets the current output string delimiter to the value passed in register E.

C-Function 111 Print Block

Call: Return:

DE = CCB address
C = 6Fh

The Print Block function sends a string of characters to the console screen. The string may be of any length, and is defined by a 4-byte Character Control Block (CCB) whose address is passed in register DE. The CCB has the following structure:

bytes	description
0–1	starting address of string
2–3	byte length of string

Horizontal tabs are expanded into multiple spaces, based upon tab stops at every eighth column.

C-Function 112 List Block

Call:
DE = CCB address
C = 70h

Return:

The List Block function sends a string of characters to the list device. The string may be of any length, and is defined by a 4-byte Character Control Block (CCB) whose address is passed in register DE. The CCB has the following structure:

bytes	description
0–1	starting address of string
2–3	byte length of string

Horizontal tabs are not expanded.

C-Function 134 Create Queue

Call:	Return:
DE = QD address	A = 00h if queue created
C = 86h	A = FFh if unable to create

The Make Queue function emulates the creation of an MP/M queue defined by a 14-byte Queue Descriptor (QD) whose address is passed in register pair DE. The QD has the following structure:

bytes	description
0–1	zeroes (internal use)
2–9	queue name in ASCII (8 chars)
10–11	message length
12–13	maximum number of messages

This function causes the creation of a global RAM FIFO file having the given queue name and type ".QUE" on user area 0 of the disk determined by the system patch QUEDRV. Any pre-existing file with the same name and type is deleted.

Within the FIFO file , the message length is rounded up to the next multiple of 128 bytes, not to exceed 256 bytes if accessed from a banked TPA. The maximum number of messages must not exceed the capacity of a RAM FIFO (16,256 bytes) and is rounded down as necessary.

This function is supported only if the modules MPMSUP and QUEMGR have been incorporated into the operating system.

WARNING: Do not confuse emulated MP/M queue FIFO files with TurboDOS print queue files.

C-Function 135 Open Queue

Call:	Return:
DE = QPB address	A = 00h if opened
C = 87h	A = FFh if unable to open

The Open Queue function opens an emulated MP/M queue defined by a 12-byte Queue Parameter Block (QPB) whose address is passed in register pair DE. The QPB has the following structure:

bytes	description
0–1	queue pointer
2–3	buffer address
4–11	queue name in ASCII (8 chars)

This function causes the opening of a RAM FIFO file having the given queue name and type ".QUE" as previously created by C-function 134 (Make Queue). The Open Queue function fills in the "queue pointer" field of the QPB, and ignores the "buffer address" field.

This function is supported only if the modules MPMSUP and QUEMGR have been incorporated into the operating system.

WARNING: Do not confuse emulated MP/M queue FIFO files with TurboDOS print queue files.

C-Function 136 Delete Queue

Call:	Return:
DE = QPB address	A = 00h if deleted
C = 88h	A = FFh if unable to delete

The Delete Queue function deletes an emulated MP/M queue defined by a 12-byte Queue Parameter Block (QPB) whose address is passed in register pair DE. The QPB has the following structure:

bytes	description
0–1	queue pointer
2–3	buffer address
4–11	queue name in ASCII (8 chars)

The queue must have been previously opened via C-function 135 (Open Queue) so that the QPB "queue pointer" field is valid. The Delete Queue function closes the queue and deletes the associated RAM FIFO file . The "buffer address" and "queue name" fields of the QPB are ignored.

This function is supported only if the modules MPMSUP and QUEMGR have been incorporated into the operating system.

WARNING: Do not confuse emulated MP/M queue FIFO files with TurboDOS print queue files.

C-Function 137 Read Queue

Call:	Return:
DE = QPB address	A = 00h if read
C = 89h	A = FFh if queue not open

The Read Queue function reads a message from an emulated MP/M queue defined by a 12-byte Queue Parameter Block (QPB) whose address is passed in register pair DE. The QPB has the following structure:

bytes	description
0–1	queue pointer
2–3	buffer address
4–11	queue name in ASCII (8 chars)

The queue must have been previously opened via C-function 135 (Open Queue) so that the QPB "queue pointer" field is valid. The Read Queue function reads a message from the queue into memory starting at the buffer address specified by the QPB. If the queue is empty, the calling process is suspended until a message becomes available. The "queue name" field of the QPB is ignored.

This function is supported only if the modules MPMSUP and QUEMGR have been incorporated into the operating system.

WARNING: Do not confuse emulated MP/M queue FIFO files with TurboDOS print queue files.

C-Function 138 Conditional Read Queue

Call:	Return:
DE = QPB address	A = 00h if read
C = 8Ah	A = FFh if queue not open or empty

The Conditional Read Queue function reads a message from an emulated MP/M queue defined by a 12-byte Queue Parameter Block (QPB) whose address is passed in register pair DE. The QPB has the following structure:

bytes	description
0–1	queue pointer
2–3	buffer address
4–11	queue name in ASCII (8 chars)

The queue must have been previously opened via C-function 135 (Open Queue) so that the QPB "queue pointer" field is valid. The Read Queue function reads a message from the queue into memory starting at the buffer address specified by the QPB. If the queue is empty, an FFh (–1) is returned in register A. The "queue name" field of the QPB is ignored.

This function is supported only if the modules MPMSUP and QUEMGR have been incorporated into the operating system.

WARNING: Do not confuse emulated MP/M queue FIFO files with TurboDOS print queue files.

C-Function 139 Write Queue

Call:	Return:
DE = QPB address	A = 00h if written
C = 8Bh	A = FFh if queue not open

The Write Queue function writes a message to an emulated MP/M queue defined by a 12-byte Queue Parameter Block (QPB) whose address is passed in register pair DE. The QPB has the following structure:

bytes	description
0–1	queue pointer
2–3	buffer address
4–11	queue name in ASCII (8 chars)

The queue must have been previously opened via C-function 135 (Open Queue) so that the QPB "queue pointer" field is valid. The Write Queue function writes a message to the queue from memory starting at the buffer address specified by the QPB. If the queue is full, the calling process is suspended until a message becomes available. The "queue name" field of the QPB is ignored.

This function is supported only if the modules MPMSUP and QUEMGR have been incorporated into the operating system.

WARNING: Do not confuse emulated MP/M queue FIFO files with TurboDOS print queue files.

C-Function 140 Conditional Write Queue

Call:	Return:
DE = QPB address	A = 00h if written
C = 8Ch	A = FFh if queue not open or full

The Conditional Write Queue function writes a message from an emulated MP/M queue defined by a 12-byte Queue Parameter Block (QPB) whose address is passed in register pair DE. The QPB has the following structure:

bytes	description
0–1	queue pointer
2–3	buffer address
4–11	queue name in ASCII (8 chars)

The queue must have been previously opened via C-function 135 (Open Queue) so that the QPB "queue pointer" field is valid. The Conditional Write Queue function writes a message to the queue from memory starting at the buffer address specified by the QPB. If the queue is full, an FFh (–1) is returned in register A. The "queue name" field of the QPB is ignored.

This function is supported only if the modules MPMSUP and QUEMGR have been incorporated into the operating system.

WARNING: Do not confuse emulated MP/M queue FIFO files with TurboDOS print queue files.

C-Function 141 Delay

Call: Return:
DE = tick count
C = 8Dh

The Delay function is identical to T-function 2 (Delay Process) and causes the calling process to be suspended for the period of time specified by the tick count passed in register pair DE. A system “tick” is an implementation dependent time interval, usually 1/50 or 1/60 of a second. The actual delay may vary from the requested tick count by plus or minus one tick.

If the specified tick count is zero, then the calling program is suspended only long enough to allow any other ready processes to run (a so called “courtesy” dispatch).

This function is supported only if the module MPMSUP has been incorporated into the operating system.

C-Function 142 Dispatch

Call: Return:

C = 8Eh

The Dispatch function causes the calling process to be suspended only long enough to allow any other ready processes to run (a so called “courtesy” dispatch). This is identical to executing a C-function 141 (Delay) or a T-function 2 (Delay Process) with 0000h passed in register pair DE.

This function is supported only if the module MPMSUP has been incorporated into the operating system.

C-Function 143 Terminate

Call: Return:

C = 8Fh

The Terminate function terminates the calling program and executes a warmstart. It is honored for transient programs only, and is equivalent to C-function 0 (System Reset).

This function is supported only if the module MPMSUP has been incorporated into the operating system.

C-Function 152 Parse Filename

Call:	Return:
DE = PFCB address	HL = 0000h if parse & line end
C = 98h	HL = FFFFh if parse error
	HL = delimiter address otherwise
	DE = delimiter address

The Parse Filename function parses an ASCII file specification into an FCB: the addresses of the ASCII file specification string and the FCB are contained in a 4-byte Parse Filename Control Block (PFCB) whose address is passed in register pair DE. The PFCB has the following structure:

bytes	description
0–1	address of ASCII input string
2–3	address of destination FCB

This function parses the first file specification it finds in the input string, ignoring leading whitespace. Parsing stops upon encountering a space, comma, semicolon, equals sign, or any ASCII control character. This function parses an ASCII file specification of the form:

{uud:}filename{.typ}

where “filename” is a name up to 8 characters long, “.typ” is an optional type up to 3 characters long, and “uud:” is an optional user/drive prefix taking one of the following six forms:

uu: d: uud: duu: uu:d: d:uu:

where “uu” is a decimal user number (0–31) and “d” is a drive letter (A to P).

The FCB drive, name, and type fields (bytes 0 through 11) are initialized according to the parsed file specification. FCB bytes 12 and 14 are set to 00h. In the absence of a user number prefix, FCB bytes 13 and 15 are also set to 00h. If the file specification includes a user number prefix, however, FCB byte 13 is set to the given user number, and FCB byte 15 is set to FFh (–1).

C-Function 153 Get Console Number

Call:	Return:
C = 99h	A = console number

The Get Console Number function returns a console number in register A. This console number is constructed by taking the network node number from the first entry of the circuit assignment table (CKTAST), adding a constant (RCNOFF), default 00h, and masking with another constant (RCNMSK), default FFh. The resulting console number should be unique in most simple (one circuit) networks.

This function is supported only if the module MPMSUP has been incorporated into the operating system.

C-Function 155 Get Date and Time

Call: Return:

DE = date/time packet address

C = 9Bh

The Get Date and Time function returns the system date and time and in a 5-byte date/time packet whose address is passed in register pair DE. The date/time packet has the following structure:

byte	Description
0–1	Julian date: 0000h = 31 December 1977
2	Hours: 2 BCD digits
3	Minutes: 2 BCD digits
4	Seconds: 2 DCB digits

This is a variation of C-function 105 (Get Date and Time) where the seconds are returned in the date/time packet instead of register A.

C-Function 159 Detach List Device

Call: Return:
C = 9Fh

The Detach List Device function is identical to T-function 28 (Signal End of Print) and causes an end of print condition. If spooling is in effect, the current print file is closed and (if appropriate) enqueued for background printing.

An end of print condition may also occur as the result of a warmstart, attention request, or end of print character.

This function is supported only if the module MPMSUP has been incorporated into the operating system.

C-Function 160 Set List

Call: Return:

E = list device number
C = A0h

The Set List function saves the list device number passed in register E for use by a subsequent C-function 161 (Conditional Attach List). The list device may be either a printer or a queue, but not a file or the console. The list device number is:

00h = A	4h = E	8h = I	Ch = M
01h = B	5h = F	9h = J	Dh = N
02h = C	6h = G	Ah = K	Eh = O
03h = D	7h = H	Bh = L	Fh = P

This function is supported only if the module MPMSUP has been incorporated into the operating system.

C-Function 161 Conditional Attach List

Call:	Return:
C = A1h	A = 00h

The Conditional Attach List function sets the current list device to the device specified by a prior C-function 160 (Set List). The list device may be a printer or a queue, but not a file or the console. The print mode and spool drive are not affected.

This function is supported only if the module MPMSUP has been incorporated into the operating system.

T-Function Listing

A listing of all T-functions in numerical order, one function per page.

T-Function 0 Reset Operating System

Call: Return:
C = 00h

The Reset Operating System function unlocks all locked records, close all open files, unlock all locked drives, and terminate any network sessions involving the calling process.

TurboDOS automatically performs this function at each program termination (warmstart), so it is almost never necessary for a program to call this function explicitly.

T-Function 1 Create Process

Call:	Return:
HL = workspace address	A = 00h if process created
DE = entrypoint address	A = FFh if insufficient memory
C = 01h	

The Create Process function creates a new process which starts execution at the entry point address passed in register pair DE. The new process is assigned a TurboDOS work area whose address appears to the new process in index register IX, and a 64-word stack area whose address appears in the register SP. If the process requires a re-entrant work area (usually allocated dynamically using T-function 3), its address should be passed in register pair HL and appears to the new process in index register IY.

If this function is called with register pair DE set to 0000h, it causes the calling process to terminate.

NOTE: This function is not intended for use by transient programs, and must be used with great care.

T-Function 2 Delay Process

Call: Return:
DE = tick count
C = 02h

The Delay Process function causes the calling process to be suspended for the period of time specified by the tick count passed in register pair DE. A system “tick” is an implementation dependent time interval, usually 1/50 or 1/60 of a second. The actual delay may vary from the requested tick count by plus or minus one tick.

If the specified tick count is zero, then the calling program is suspended only long enough to allow any other ready processes to run (a so called “courtesy” dispatch).

T-Function 3 Allocate Memory

Call:	Return:
DE = segment length	HL = segment address
C = 03h	A = 00h if allocated
	A = FFh if insufficient memory

The Allocate Memory function allocates a contiguous memory segment of the byte length passed in register pair DE. If successful, the starting address of the allocated segment is returned in register pair HL.

NOTE: This function is not intended for use by transient programs, and must be used with great care. If a memory segment is allocated by a transient program and not deallocated before the program terminates, then the space is lost permanently.

T-Function 4 Deallocate Memory

Call: Return:
DE = segment address
C = 04h

The Deallocate Memory function restores a previously allocated memory segment to the pool of available memory space.

NOTE: This function is not intended for use by transient programs, and must be used with great care. The address passed in DE must be a segment starting address returned via a prior C-function 3 (Allocate Memory); otherwise, a system crash may occur.

T-Function 5 Send Interprocess Message

Call: Return:

HL = message address
DE = message node address
C = 05h

The Send Interprocess Message function provides a means to send messages from one process to another. Register pair DE must specify the address of a 10-byte message node which must be initialized as follows:

```
MSGNOD: DW      0000h           ;semaphore count
        DW      MSGNOD+2        ;semaphore head
        DW      MSGNOD+2        ;    "      "
        DW      MSGNOD+4        ;msg chain head
        DW      MSGNOD+4        ; "    "    "
```

Register pair HL must specify the address of the message to be sent, which must be prefixed by a 4-byte linkage as follows:

```
MESSAG: DW      0000h           ;message linkage
        DW      0000h           ;   "       "
        DB      ...             ;message text (any length)
```

NOTE: This function is not intended for use by transient programs, and must be used with great care.

T-Function 6 Receive Interprocess Message

Call:
DE = message node address
C = 06h

Return:
HL = message address

The Receive Interprocess Message function provides a means to receive messages sent by another process using C-function 5 (Send Interprocess Message). Register pair DE must specify the address of a 10-byte message node which must be initialized as follows:

```
MSGNOD: DW      0000h           ;semaphore count
        DW      MSGNOD+2        ;semaphore head
        DW      MSGNOD+2        ;    "      "
        DW      MSGNOD+4        ;msg chain head
        DW      MSGNOD+4        ; "    "    "
```

If no message is available from the specified message node, the calling process is suspended until a message arrives. This function returns in HL the address of the received message prefixed by a 4-byte linkage as follows:

```
MESSAG: DW      0000h           ;message linkage
        DW      0000h           ;   "       "
        DB      ...             ;message text (any length)
```

NOTE: This function is not intended for use by transient programs, and must be used with great care.

T-Function 7 Set Error Address

Call: Return:
DE = error address (Set)
C = 07h

Call: Return:
DE = 0000h (Reset)
C = 07h

The Set Error Address function enables a program to establish its own error intercept routine to intercept and process unrecoverable disk errors. The address of the intercept routine is passed in register pair DE. Normal TurboDOS error diagnosis is suppressed.

The error intercept routine must not call any TurboDOS functions, and must return via a RET instruction with register A set to the desired error recovery alternative:

A = 00h retry operation
A = 01h ignore error
A = FFh abort program

If the Set Error Address function is called with register pair DE set to 0000h, normal TurboDOS error diagnosis is restored. This also happens automatically when the program terminates.

T-Function 8 Set Abort Address

Call: Return:
DE = abort address (Set)
C = 08h

Call: Return:
DE = 0000h (Reset)
C = 08h

The Set Abort Address function enables a program to establish its own abort intercept routine to intercept and process user requested aborts (in response to attention requests or disk errors). The address of the intercept routine is passed in register pair DE.

The abort intercept routine may exit via a RET instruction to resume execution of the program at the point of interruption. Alternatively, it may proceed with any desired wrap up processing and then terminate the program.

If the Set Abort Address function is called with register pair DE set to zero, normal TurboDOS abort handling is restored. This also happens automatically when the program terminates.

T-Function 9 Set Date and Time

Call:
HL = Julian date: 0000h = 31 Dec 1947
D = hours: binary
E = minutes: binary
B = seconds: binary
C = 09h

Return:

The Set Date and Time function sets the system date and time. The Julian date passed in register pair HL is the number of days since the base date of 31 December 1947. Dates prior to the base date are represented by negative values.

The system date and time may also be set by means of C-function 104 (Set Date and Time), but the format of arguments is considerably different.

T-Function 10 Get Date and Time

Call:
C = 0Ah

Return:
HL = Julian date: 0000h = 31 Dec 1947
D = Hours: binary
E = Minutes: binary
B = Seconds: binary
C = system tick count

The Get Date and Time function returns the system date and time. The Julian date returned in register pair HL is the number of days since the base date of 31 December 1947. Dates prior to the base date are represented by negative values.

If the system is freshly booted and the date and time has not been set, 8001h is returned in register HL, and hours, minutes, seconds and system ticks count elapsed time from the boot.

The system tick count returned in register C is incremented every system tick. It counts from zero to 255, then wraps around to zero. A system tick is an implementation dependent time interval, usually 1/50 or 1/60 of a second.

The system date and time may also be interrogated by means of C-functions 105 and 155, but the format of returned values is considerably different.

T-Function 11 Rebuild Disk Map

Call:	Return:
E = disk drive code	A = 00h if map regenerated
C = 0Bh	A = FFh if files open or disk read only

The Rebuild Disk Map function regenerates the allocation map on the disk drive Whose code is passed in register E. The disk drive codes for this function are:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

The principal purpose of this function is to support the FIXMAP command.

T-Function 12 Return TurboDOS Serial Number

Call:	Return:
C = 0Ch	HL = origin number DE = unit number B = 00h if non-privileged B = 80h if privileged C = 14h

The Return Serial Number function returns the origin and unit numbers with which this particular copy of TurboDOS was serialized, and may be used in application programs to help prevent unauthorized use.

This function also returns the TurboDOS version number, and a flag which indicates whether or not the current log-on is privileged.

T-Function 13 Set Compatibility Flags

Call:
E = COMPAT byte
C = 0Dh

Return:

The Set Compatibility Flags function allows a program to modify the rules by which file locking is done. The meaning of each compatibility flag within the COMPAT byte is explained in the sections on COMPAT and file locking previously described.

When the program terminates, the COMPAT byte automatically reverts to its default value assigned at system generation.

T-Function 14 Log-On/Log-Off

Call:	Return:
D = disk drive code (Log-on) E = user area + status C = 0Eh	A = 00h if logged-on A = FFh if invalid request

Call:	Return:
DE = FFFFh (Log-off) C = 0Eh	

The Log-On/Log-Off function is provided to support log-on security via the LOGON and LOGOFF commands.

If a value other than FFFFh is passed in register pair DE, this function causes a log-on to the user area and status passed in register E, and with a change in current drive to that passed in register D. The user and status codes for register E are:

user	non-	priv	user	non-	priv	user	non-	priv	user	non-	priv
0	00h	80h	8	08h	88h	16	10h	90h	24	18h	98h
1	01h	81h	9	09h	89h	17	11h	91h	25	19h	99h
2	02h	82h	10	0Ah	8Ah	18	12h	92h	26	1Ah	9Ah
3	03h	83h	11	0Bh	8Bh	19	13h	93h	27	1Bh	9Bh
4	04h	84h	12	0Ch	8Ch	20	14h	94h	28	1Ch	9Ch
5	05h	85h	13	0Dh	8Dh	21	15h	95h	29	1Dh	9Dh
6	06h	86h	14	0Eh	8Eh	22	16h	96h	30	1Eh	9Eh
7	07h	87h	15	0Fh	8Fh	23	17h	97h	31	1Fh	9Fh

The drive codes for register D are.

00h = drive A:	08h = drive I:
01h = drive B:	09h = drive J:
02h = drive C:	0Ah = drive K:
03h = drive D:	0Bh = drive L:
04h = drive E:	0Ch = drive M:
05h = drive F:	0Dh = drive N:
06h = drive G:	0Eh = drive O:
07h = drive H:	0Fh = drive P:
	FFh = no change in current drive

If FFFFh is passed in register pair DE, this function causes a log-off.

After a log-off, another log-on request is not honored until a warmstart has occurred.

NOTE: When this function is called from a resident system process, the D register argument is ignored.

T-Function 15 Load File

Call:	Return:
DE = FCB address	A = 00h if file loaded
C = 0Fh	A = 01h if insufficient memory
	A = FFh if file not found

The Load File function loads the file specified by the FCB drive, name, and type fields (bytes 0 through 11) into memory starting at the current DMA address. The file need not have been opened. If the top of the TPA is reached before the end of file is encountered, the loading stops and an error is returned.

If the specified drive is local to the processor in which the call is made and the TPA is non-banked, the program load optimizer is used. If the drive is not local or the TPA is banked, multi-sector operations are used to optimize performance.

This function is used by the TurboDOS command interpreter to load .COM files into the TPA, and may also be used by application programs to fetch overlays.

T-Function 16 Activate/Deactivate DO-File

Call:	Return:
DE = FCB address (Activate) C = 10h	A = 00h if DO file activated A = FFh if file not found
Call:	Return:
DE = 0000h (Deactivate) C=10h	

The Activate/Deactivate DO File function causes the file specified by the FCB drive, name, and type fields (bytes 0 through 11) to be activated as a DO file. The file need not have been opened. Any currently active DO file and/or command line is stacked (to be reactivated when the new DO file has been processed to completion). The principal purpose of this function is to support the DO command.

This function may also be called with 0000h passed in register pair DE to cancel all active and stacked DO files.

T-Function 17 Disable/Enable Autoload

Call:
E = 00h (Disable)
C = 11h

Return:

Call:
E = FFh (Enable)
C =

Return:

The Disable/Enable Autoload function may be used to disable the warmstart autoload feature of TurboDOS, or to re-enable the feature after it has been disabled.

TurboDOS automatically disables the warmstart autoload feature whenever it fails to find the file WARMSTRT.AUT on the current disk during a warmstart. Creating such a file on disk (or changing the current disk to one that contains such a file) does not result in autoloading unless the autoload feature is explicitly re-enabled by means of this function.

T-Function 18 Send Command Line

Call:
DE = buffer address (Send)
C = 12h:

Return:

Call:
DE = 0000h (Cancel)
C = 12h

Return:

The Send Command Line function allows a program to specify the next command line to be processed by TurboDOS after the program terminates. The buffer address is passed in register pair DE. The first byte of the buffer must contain the command line byte length, and the command line text must occupy the second and succeeding bytes of the buffer. Any currently active command line is stacked, and the new command line is activated.

The commands are echoed to the console, unless the command line starts with a leading command separator (backslash) character.

If 0000h is passed in register pair DE this function cancels all active and stacked command lines.

T-Function 19 Return Disk Allocation Information

Call:	Return:
E = disk drive code	HL = number of blocks
C = 13h	DE = number of unused blocks
	C = number of directory blocks
	A = block size code
	A = FFh if network error

The Return Disk Allocation Information function returns various parameters concerning the logical organization of the drive whose code is passed in register E. The register E drive code is:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

The total number of allocation blocks on the drive is returned in register pair HL.

The number of unused (free) allocation blocks on the disk is returned in register pair DE.

The number of allocation blocks dedicated to the directory is returned in register C.

A code is returned in the least significant nybble of register A indicating the size of an allocation block:

3 = 1024 bytes
4 = 2048 bytes
5 = 4096 bytes
6 = 8192 bytes
7 = 16384 bytes

Bit 7 of register A is used to indicate the type of media:

0 = removable 1 = fixed

Bit 6 of register A is used to indicate EXM status:

0 = EXM=0 not forced 1 = EXM=0 forced

T-Function 20 Return Physical Disk Information

Call:	Return:
E = disk drive code	HL = number of sector/track
C = 14h	DE = number of tracks
	BC = number of reserved tracks
	A = sector size
	A = FFh if network error

The Return Physical Disk Information function returns various parameters concerning the format and physical organization of the disk drive whose code is passed in register E. The register E drive code is:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

The number of sectors per track is returned in register pair HL.

The total number of tracks on the drive is returned in register pair DE.

The number of reserved (boot) tracks is returned in register pair BC.

A code designating the physical sector size is returned in register A:

00h =	128 bytes	04h =	2048 bytes
01h =	256 bytes	05h =	4096 bytes
02h =	512 bytes	06h =	8192 bytes
03h =	1024 bytes	07h =	16384 bytes

T-Function 21 Get/Set Drive Status

Call:	Return:
D = 00h (Set read/write) E = drive code C = 15h	A = 00h if set read/write A = FFh if files open

Call:	Return:
D = 01h (Set read only) E = drive code C = 15h	A = 00h if set read only A = FFh if files open

Call:	Return:
D = FFh (Get status) E = drive code C = 15h	H = 00h if drive read/write H = FFh if drive read only L = 00h if drive not ready L = FFh if drive ready

The Get/Set Drive Status function may be used to interrogate the ready and write protect status of the drive whose code is passed in register E. This function may also be used to change the write protect status of the drive. The code passed in register D controls which of these operations is performed. The register E drive code is:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

Warning: This function may produce a false ready on a remote drive. C-function 24 (Return Login Vector) produces correct cross-network results.

T-Function 22 Physical Disk Access

Call:	Return:
DE = PDR packet address	A = 00h if accessed or drive not ready
C = 16h	A = FFh if access denied or drive ready

The Physical Disk Access function provides direct access to the physical disk drivers. The principal purpose of this function is to support the BOOT, BACKUP, FORMAT, and VERIFY commands. It is honored for privileged log-ons only, and may be used only for local disk drives and non-banked TPA.

Register DE must pass the address of a 14-byte physical disk request (PDR) packet with the following structure:

Bytes	Description
00h	disk operation code (00h–04h)
01h	disk drive code
02h–03h	physical track number (base 0)
04h–05h	physical sector number (base 0)
06h–07h	number of sectors to read or write
08h–09h	number of bytes to read or write
0Ah–0Bh	DMA address for read or write
0Ch–0Dh	disk specification table address

The drive code specified by PDR byte 01h is as follows:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

The physical operation to be performed depends upon the disk operation code in PDR packet byte 00h.

If the PDR opcode is 00h, the specified number of physical sectors (or bytes) is read from the specified drive, track, and sector into the specified DMA address.

If the PDR opcode is 01h, the specified number of physical sectors (or bytes) is written to the specified drive, track, and sector from the specified DMA address.

If the PDR opcode is 02h, the type of the specified disk is determined, and an 11-byte disk specification table (DST) is returned at the specified DMA address, structured as follows:

Bytes	Description
00h	block size code
01h–02h	total number of blocks on disk
03h	number of directory blocks
04h	sector size code
05h–06h	number of sectors per track
07h–08h	number of tracks on the disk
09h–0Ah	number of reserved (boot) tracks

The block size code and sector size code in bytes 00h and 04h of the DST are as follows:

Blocks:	Size	Removable	Fixed	Sectors	Size	Removable	Fixed	Sectors
	128	—	—	00h	2048	04h	84h	04h
	256	—	—	01h	4096	05h	85h	05h
	512	—	—	02h	8192	06h	86h	06h
	1024	03h	83h	03h	16384	07h	87h	07h

If the PDR opcode is 03h, the ready status of the specified drive is returned in register A:

A = 00h if not ready; FFh if ready

If the PDR opcode is 04h, the specified track of the specified drive is formatted, using hardware dependent formatting information provided at the specified DMA address.

NOTE: Opcodes 00h (read) and 01h (write) require that the PDR packet contain the address of a valid DST for the specified disk. Therefore, opcode 02h (return DST) should be invoked first to obtain the DST.

T-Function 23 Set Buffer Parameters

Call: Return:

D = number of buffers
E = buffer size code
C = 17h

The Set Buffer Parameters function enables the number and size of disk buffers to be changed. The principal purpose of this function is to support the BUFFERS command.

The specified number of buffers must be at least 2. If the specified number of buffers cannot be allocated due to insufficient memory, then TurboDOS allocates as many as it can.

The specified buffer size must be as least as large as the largest physical disk sector size being used. The buffer size code passed in register E is:

00h =	128 bytes	04h =	2048 bytes
01h =	256 bytes	05h =	4096 bytes
02h =	512 bytes	06h =	8192 bytes
03h =	1024 bytes	07h =	16384 bytes

If this function is called from a slave processor without local disk storage, then the function is passed over the network to be processed in the master.

T-Function 24 Get Buffer Parameters

Call:	Return:
C = 18h	H = number of buffers L = buffer size code A = number of memory pages

The Get Buffer Parameters function enables the number and size of disk buffers to be interrogated. The principal purpose of this function is to support the BUFFERS command.

The number of buffers is returned in register H, and a code indicating their size is returned in register L:

00h =	128 bytes	04h =	2048 bytes
01h =	256 bytes	05h =	4096 bytes
02h =	512 bytes	06h =	8192 bytes
03h =	1024 bytes	07h =	16384 bytes

The size of memory in pages is returned in register A.

If this function is called from a slave processor without local disk storage, then the function is passed over the network to be processed in the master.

T-Function 25 Lock/Unlock Drive

Call:	Return:
D = 00h (Unlock) E = disk drive code C = 19h	

Call:	Return:
D = FFh (Lock) E = disk drive code C = 19h	A = 00h if drive locked A = FFh if drive in use or already locked

The Lock/Unlock Drive function enables a program to secure a lock on a drive whose code is passed in register E. The register E drive code is:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

If this function is called with FFh in register D, then the specified drive is unlocked if previously locked by the calling process.

This function is used by many TurboDOS commands such as BACKUP, CHANGE, FIXDIR, FIXMAP, FORMAT, and VERIFY to ensure that they cannot compromise the processing of other users.

T-Function 26 Flush/Free Buffers

Call: Return:

D = subfunction code
E = disk drive code
C = 1Ah

The Flush/Free Buffers function causes all written-to disk buffers for the drive whose code is passed in register E to be written out (flushed) to the disk. This function may cause disk buffers for the specified drive to be freed, conditionally or unconditionally, according to the subfunction flags passed in register D:

Bit	Subfunction
7 = 1	flush buffers unconditionally
6 = 1	flush buffers after disk error abort
5 = 1	continue after disk error abort
4 = 1	return after disk error abort

The drive code passed in register E is:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

It is suggested that this function be used prior to media changes and physical disk access (T-function 22).

T-Function 27 Get/Set Print Mode

Call:	Return:
E = print mode (Set)	
D = printer/queue assignment	
B = spool drive code	
C = 1Bh	

Call:	Return:
E = FFh (Get)	H = printer/queue assignment
D = FFh	L = print mode
B = FFh	A = spool drive
C = 1Bh	

The Get/Set Print Mode function allows a program to set or interrogate the list device, and is provided to support the PRINT command.

The desired print mode may be passed in register E:

00h = direct printing, list device is a printer
01h = spooled printing, list device is a queue
02h = console printing, list device is the console
FFh = leave print mode unchanged

If the print mode is direct or spooled, the printer/queue assignment may be passed in register D:

00h = list output is discarded if printing is direct; list device is unqueued file if printing is spooled
01h = list device is printer/queue A
02h = list device is printer/queue B
03h = list device is printer/queue C
04h = list device is printer/queue D
05h = list device is printer/queue E
06h = list device is printer/queue F
07h = list device is printer/queue G
08h = list device is printer/queue H
09h = list device is printer/queue I
0Ah = list device is printer/queue J
0Bh = list device is printer/queue K
0Ch = list device is printer/queue L
0Dh = list device is printer/queue M
0Eh = list device is printer/queue N
0Fh = list device is printer/queue O
10h = list device is printer/queue P
FFh = leave list device unchanged

If print mode is spooled, the spooling drive code may be passed in register B:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

If FFh is passed in registers E, D and B, this function returns the current print mode in register L, the current printer/queue assignment in register H, and the current spooling drive code in register A.

T-Function 28 Signal End-of-Print

Call: Return:
C = 1Ch

The Signal End of Print function causes an end of print condition. If spooling is in effect, the current print file is closed and (if appropriate) enqueued for background printing.

An end of print condition may also occur as the result of a warmstart, attention request, or end of print character.

T-Function 29 Get/Set De-Spool Mode

Call:	Return:
E = despool mode (Set) D = despool queue assignment B = printer C = 1Dh	A = 00h if mode set A = FFh if invalid request

Call:	Return:
E = FFh (Get) D = FFh B = printer C = 1Dh	L = despool mode H = despool queue assignment

The Get/Set Despool Mode function controls background printing, and is provided to support the PRINTER command. The printer to be set or queried is passed in register B:

00h = printer A	04h = printer E	08h = printer I	0Ch = printer M
01h = printer B	05h = printer F	09h = printer J	0Dh = printer N
02h = printer C	06h = printer G	0Ah = printer K	0Eh = printer O
03h = printer D	07h = printer H	0Bh = printer L	0Fh = printer P

Its despool mode is passed in register E:

00h =	process print job
01h =	suspend print job
02h =	begin print job
03h =	terminate print job
FFh =	leave despool mode unchanged

Its despool queue assignment is passed in register D:

00h =	printer is set to offline.	09h =	printer is set to queue I
01h =	printer is set to queue A	0Ah =	printer is set to queue J
02h =	printer is set to queue B	0Bh =	printer is set to queue K
03h =	printer is set to queue C	0Ch =	printer is set to queue L
04h =	printer is set to queue D	0Dh =	printer is set to queue M
05h =	printer is set to queue E	0Eh =	printer is set to queue N
06h =	printer is set to queue F	0Fh =	printer is set to queue O
07h =	printer is set to queue G	10h =	printer is set to queue P
08h =	printer is set to queue H	FFh =	leave assignment unchanged

If FFh is passed in both registers D and E, this function returns the current despool mode in register L and the current despool assignment in register H.

T-Function 30 Queue a Print File

Call:	Return:
DE = FCB address (Enqueue) H = print queue L = user area + delete flag C = 1Eh	A = 00h if file queued A = FFh if invalid request

Call:	Return:
L = FFh (Verify validity) DE = FCB address H = print queue C = 1Eh	A = 00h if drive and queue valid A = FFh if invalid request

The Queue a Print File function enqueues a text file on a specified print queue for background printing. The file to be enqueued is identified by the drive, name and type fields (bytes 0 through 11) of the FCB whose address is passed in register pair DE, together with the user number passed in register L.

The print queue passed in register H is as follows:

00h = queue A	04h = queue E	08h = queue I	0Ch = queue M
01h = queue B	05h = queue F	09h = queue J	0Dh = queue N
02h = queue C	06h = queue G	0Ah = queue K	0Eh = queue O
03h = queue D	07h = queue H	0Bh = queue L	0Fh = queue P

Bit 7 of the user area number passed in register L acts as a flag determining deletion status of the file(s) after enqueuing:

bit 7	description
0	file is not deleted
1	file is deleted

The drive specified by the FCB must be accessible by the processor in which the specified queue resides; otherwise, the request is invalid. To check this, the function may be called with register L set to –1, in which case the FCB drive and requested queue are checked for validity but no file is queued.

T-Function 31 Flush List Buffer

Call: Return:
C = 1Fh

The Flush List Buffer function is used by TurboDOS during direct printing over the network to force any remaining buffered characters to be printed. There should be no need for an application program to call this function.

T-Function 32 Network List Out

Call: Return:

E = output character
C = 20h

The Network List Out function is used by TurboDOS during direct printing over the network. There should be no need for an application program to call this function.

T-Function 33 Remote Console I/O

Call:	Return:
D = FFh (Attach) E = 00h if no input character E = console input character C = 21h	A = 00h if CONREM not present A = 01h if attached A = FFh executing in master

Call:	Return:
D = 00h (Detach) C = 21h	

The Remote Console I/O function works in conjunction with the CONREM console driver to support the MASTER command.

If FFh is passed in register D, it attaches to the master, passes one byte of console input in register E (if available), and returns a count byte and up to 127 bytes of console output at the current DMA address.

If 00h is passed in register D, it detaches from the master.

There should be no need for an application program to call this function.

T-Function 34 Get Comm Channel Status

Call:	Return:
D = channel + remote flag	A = 00h if byte not available
C = 22h	A = FFh if byte available

The Get Comm Channel Status function checks to see whether or not an input byte is available on the comm channel whose number is passed in register D. If a character is available, FFh (–1) is returned in register A; otherwise, 00h is returned.

Bit 7 of the channel number acts as a remote flag:

bit 7	description
0	channel is local
1	channel is remote

T-Function 35 Comm Channel Input

Call:	Return:
D = channel + remote flag	A = input byte
C = 23h	

The Comm Channel Input function obtains the next input byte from the comm channel whose number is passed in register D, waiting if necessary, and returns it in register A.

Bit 7 of the channel number acts as a remote flag:

bit 7	description
0	channel is local
1	channel is remote

T-Function 36 Comm Channel Output

Call: Return:

E = output byte
D = channel + remote flag
C = 24h

The Comm Channel Output function outputs the byte passed in register E on the comm channel whose number is passed in register D..

Bit 7 of the channel number acts as a remote flag:

bit 7	description
0	channel is local
1	channel is remote

T-Function 37 Set Comm Channel Baud Rate

Call: Return:

E = Baud rate/control code
D = channel + remote flag
C = 25h

The Set Comm Channel Baud Rate function sets the Baud rate and control options whose code is passed in register E for the comm channel whose number is passed in register D.

Bit 7 of the channel number acts as a remote flag:

bit 7	description
0	channel is local
1	channel is remote

Each nybble of the Baud rate/control code acts independently. The least significant nybble (bits 3–0) determine the Baud rate:

0h = 50 Baud*	4h = 150 Baud	8h = 1800 Baud	Ch = 4800 Baud
1h = 75 Baud	5h = 300 Baud	9h = 2000 Baud	Dh = 7200 Baud
2h = 110 Baud	6h = 600 Baud	Ah = 2400 Baud	Eh = 9600 Baud
3h = 134.5 Baud	7h = 1200 Baud	Bh = 3600 Baud*	Fh = 19200 Baud

* may be 38400 Baud in some configurations

Bits 7–4 for the Baud rate/control code act independently to control various parameters:

Bit 7 0 = attention detection disabled
1 = attention detection enabled
Bit 6 0 = CTS handshaking disabled
1 = CTS handshaking enabled
Bit 5 0 = data input suppression disabled
1 = data input suppression enabled
Bit 4 0 = Xon/Xoff handshaking disabled
1 = Xon/Xoff handshaking enabled

T-Function 38 Get Comm Channel Baud Rate

Call:	Return:
D = channel + remote flag	A = Baud rate/control code
C = 26h	

The Get Comm Channel Baud Rate function returns the Baud rate and control options code in register A for the comm channel whose number is passed in register D.

Bit 7 of the channel number acts as a remote flag:

bit 7	description
0	channel is local
1	channel is remote

Each nybble of the Baud rate/control code acts independently. The least significant nybble (bits 3–0) determine the Baud rate:

0h = 50 Baud*	4h = 150 Baud	8h = 1800 Baud	Ch = 4800 Baud
1h = 75 Baud	5h = 300 Baud	9h = 2000 Baud	Dh = 7200 Baud
2h = 110 Baud	6h = 600 Baud	Ah = 2400 Baud	Eh = 9600 Baud
3h = 134.5 Baud	7h = 1200 Baud	Bh = 3600 Baud*	Fh = 19200 Baud

* may be 38400 Baud in some configurations

Bits 7–4 for the Baud rate/control code act independently to control various parameters:

Bit 7	0 = attention detection disabled
	1 = attention detection enabled
Bit 6	0 = CTS handshaking disabled
	1 = CTS handshaking enabled
Bit 5	0 = data input suppression disabled
	1 = data input suppression enabled
Bit 4	0 = Xon/Xoff handshaking disabled
	1 = Xon/Xoff handshaking enabled

T-Function 39 Set Comm Channel Modem Controls

Call: Return:

E = modem control vector
D = channel + remote flag
C = 27h

The Set Comm Channel Modem Controls function sets the modem control signals according to the vector passed in register E for the comm channel whose number is passed in register D.

Bit 7 of the channel number acts as a remote flag:

The modem control vector uses bit 7 and 6 as follows (other bits are unassigned):

bit	description
7	set for RTS
6	set for DTR

T-Function 40 Get Comm Channel Modem Status

Call:	Return:
D = channel + remote flag	A = modem status vector
C = 28h	

The Get Comm Channel Modem Status function returns the modem status vector in register A for the comm channel whose number is passed in register D.

Bit 7 of the channel number acts as a remote flag:

bit 7	description
0	channel is local
1	channel is remote

Bits 7–4 of the modem status vector are as follows (the remaining bits are unassigned):

bit	description
7	set for CTS
6	set for DSR
5	set for DCD
4	set for RI

T-Function 41 User-Defined Function

Call:	Return:
HL = user-defined argument	HL = user-defined value
DE = user-defined argument	DE = user-defined value
DE = network address if B=FEh	BC = user-defined value
B = network routing	A = user-defined value
C = 29h	

The User-Defined Function provides a means for adding user-defined extensions to the operating system, taking full advantage of the TurboDOS networking facilities. On entry, register B defines how the request is to be routed over the network. Registers DE and HL plus the 128-byte record at the current DMA address are all passed (over the network if necessary) to a user-defined module with the public entrypoint symbol USRFCN. Upon entry to the USRFCN routine, register BC contains the address of the 128-byte record that was passed. The USRFCN routine may return information to the caller in any of the seven registers A, B, C, D, E, H, and L, and in the 128-byte record.

The network routing byte passed in register B is as follows:

B	description
00h	function processed locally
1xh	function routed per drive “x”
2xh	function routed per printer “x”
3xh	function routed per queue “x”
FEh	function routed to network address passed in DE
FFh	function routed to default network address

T-Function 42 Reorganize Disk Directory

Call:	Return:
E = disk drive code	A = 00h if directory reorganized
C = 2Ah	A = FFh if files open or drive read only

The Reorganize Disk Directory function reorganizes the directory on the disk drive whose code is passed in register E. If the hashed directory flag bit in the volume label has been changed, this function converts a hashed directory into linear format (or vice versa). The principal purpose of this function is to support the FIXDIR command.

The drive code passed in register E is:

00h = drive A:	04h = drive E:	08h = drive I:	0Ch = drive M:
01h = drive B:	05h = drive F:	09h = drive J:	0Dh = drive N:
02h = drive C:	06h = drive G:	0Ah = drive K:	0Eh = drive O:
03h = drive D:	07h = drive H:	0Bh = drive L:	0Fh = drive P:

NOTE: In certain cases, this function may take a very long time to complete (possibly hours), and cannot be interrupted once invoked.

T-Function 43 Select Memory Bank

Call:	Return:
E = 00h (Select Bank 0) C = 2Bh	A = 00h
Call:	Return:
E = 01h (Select Bank 1) C = 2Bh	A = 01h
Call:	Return:
E = FFh (Return bank) C = 2Bh	A = current bank

The Select Memory Bank function selects the memory bank in which the TPA resides, according to the bank number passed in register E. If FFh is passed in register E, this function simply interrogates the current bank mode and returns the current bank number in register A. This function is honored for privileged log-ons only, and is provided primarily to support the BANK command.

A request to select bank 0 is ignored if there is not enough free memory in bank 0 for a minimum size TPA, as defined by the patchable symbol MEMBLL. (More specifically, bank 0 cannot be selected unless MEMBAS – MEMRES–3 is greater than or equal to MEMBLL.)

This function has no effect in a non-banked system, and always shows that bank 0 is selected.

T-Function 44 Open MS-DOS File

Call:	Return:
DE = FCB address	A = 00h if file opened
B = sharing/access vector	A = FFh if unable to open
L = 3-way flag	
C = 2Ch	

The Open MS-DOS File function opens an MS-DOS file specified by the FCB whose address is passed in register pair DE.

If 00h is passed in register L, the file is opened in a manner similar to C-function 15, save that FCB field s2 (byte 14) is not zeroed, attributes f5 and f6 are ignored, and file locking/sharing is as defined by the sharing/access vector passed in register B.

If 01h is passed in register L, the file is opened as above plus the directory field s1 (byte 13) is returned in FCB field cr (byte 32).

If FFh is passed in register L, the file is opened as above plus the exact byte length of the file is returned as a 32-bit value in FCB fields cr and rr (bytes 32–35).

File locking is in accordance with MS-DOS rules as defined in the sharing/access vector passed in register B. The sharing/access vector is defined as follows:

group	bit	name	description
sharing	7	DR	deny read
	6	DW	deny write
	5	DC	deny compatibility
	4	DP	deny permissive
access	3	AR	access read
	2	AW	access write
	1	AC	access compatibility
	0	AP	access permissive

TurboDOS/PC maps the MS-DOS sharing and access modes as follows:

group	mode	map	description
sharing	0	AC	compatibility
	1	DR+DW+DC	deny read/write
	2	DW+DC	deny write
	3	DR+DC	deny read
	4	DC	deny none
access	0	AR	read access
	1	AW	write access
	2	AR+AW	read/write access

TurboDOS maps the MP/M-II-style file open modes into the sharing vector as follows:

mode	map
exclusive	DR+DW+DC+AR+AW
shared	DC+AR+AW
read only (MP/M)	DW+DC+AR
read only (mixed)	DC+AR
permissive	AR+AP

It is recommended that this function not be used in general application programs, as it is intended for the use of TurboDOS/PC across a mixed network

This function is not supported from bank 1 of a bankswitched processor.

T-Function 45 Create MS-DOS File

Call:	Return:
DE = FCB address	A = 00h if file created
B = sharing/access vector	A = FFh if unable to create
C = 2Dh	

The Create MS-DOS File function creates an MS-DOS file specified by the FCB whose address is passed in register pair DE.

The file is created in a manner similar to C-function 22, save that FCB field s2 (byte 14) is not zeroed, attributes f5 and f6 are ignored, and file locking/sharing is as defined by the sharing/access vector passed in register B.

File locking is in accordance with MS-DOS rules as defined in the sharing/access vector passed in register B. The sharing/access vector is defined as follows:

group	bit	name	description
sharing	7	DR	deny read
	6	DW	deny write
	5	DC	deny compatibility
	4	DP	deny permissive
access	3	AR	access read
	2	AW	access write
	1	AC	access compatibility
	0	AP	access permissive

TurboDOS/PC maps the MS-DOS sharing and access modes as follows:

group	mode	map	description
sharing	0	AC	compatibility
	1	DR+DW+DC	deny read/write
	2	DW+DC	deny write
	3	DR+DC	deny read
	4	DC	deny none
access	0	AR	read access
	1	AW	write access
	2	AR+AW	read/write access

TurboDOS maps the MP/M-II-style file open modes into the sharing vector as follows:

mode	map
exclusive	DR+DW+DC+AR+AW
shared	DC+AR+AW
read only (MP/M)	DW+DC+AR
read only (mixed)	DC+AR
permissive	AR+AP

It is recommended that this function not be used in general application programs, as it is intended for the use of TurboDOS/PC across a mixed network

This function is not supported from bank 1 of a bankswitched processor.

T-Function 46 Lock MS-DOS File Region

Call:	Return:
DE = FCB address	A = 00h if region locked
C = 2Eh	A = FFh if lock denied

The Lock MS-DOS File Region function locks that region specified in an 8-byte Region Lock Block (RLB) passed at the current DMA address, of an MS-DOS file specified by the FCB whose address is passed in register pair DE.

The RLB consists of two 32-bit integers and has the following structure:

bytes	description
00h–03h	region byte offset
04h–07h	region byte length

It is recommended that this function not be used in general application programs, as it is intended for the use of TurboDOS/PC across a mixed network

This function is not supported from bank 1 of a bankswitched processor.

T-Function 47 Unlock MS-DOS File Region

Call:	Return:
DE = FCB address	A = 00h if region unlocked
C = 2Fh	A = FFh if unlock denied

The Unlock MS-DOS File Region function unlocks that region, previously locked by T-function 46 (Lock MS-DOS File Region), specified in an 8-byte Region Lock Block (RLB) passed at the current DMA address, of an MS-DOS file specified by the FCB whose address is passed in register pair DE.

The RLB consists of two 32-bit integers and has the following structure:

bytes	description
00h–03h	region byte offset
04h–07h	region byte length

It is recommended that this function not be used in general application programs, as it is intended for the use of TurboDOS/PC across a mixed network

This function is not supported from bank 1 of a bankswitched processor.

T-Function 48 Set MS-DOS File Length

Call:	Return:
DE = FCB address	A = 00h if length set
C = 30h	A = FFh if not set

The Set MS-DOS File Length function sets the exact byte length of an MS-DOS file specified by the FCB whose address is passed in register pair DE. The exact byte length is passed as a 32-bit integer at the current DMA address.

It is recommended that this function not be used in general application programs, as it is intended for the use of TurboDOS/PC across a mixed network

This function is not supported from bank 1 of a bankswitched processor.

T-Function 49 Get MS-DOS File Length

Call:	Return:
DE = FCB address	A = 00h if length fetched
C = 31h	A = FFh if not fetched

The Get MS-DOS File Length function gets the exact byte length of an MS-DOS file specified by the FCB whose address is passed in register pair DE. The exact byte length is returned as a 32-bit integer at the current DMA address.

It is recommended that this function not be used in general application programs, as it is intended for the use of TurboDOS/PC across a mixed network

This function is not supported from bank 1 of a bankswitched processor.

Zilog Z-80 Microprocessor Overview

Z-80 versus Z-180

This chapter is specific to the Zilog Z-80 microprocessor. However, some systems use the Zilog Z-180 microprocessor or variants (such as the Hitachi HD 64180 microprocessor). The Z-180 microprocessor is (or may be considered for our purposes here) a superset of the Z-80. That is, it is fully compatible with the Z-80 and its instruction set, but does have a few extra instructions of its own. These extra instructions are:

~ IN0 reg,(port)	input port-direct
~ MLT regpr	multiply
~ OTDM	output memory and decrement
~ OTDMR	output memory, decrement, and repeat
~ OTIM	output memory and increment
~ OTIMR	output memory, increment, and repeat
~ OUT0	output port-direct
~ SLP	sleep
~ TST source	test
~ TSTIO	test I/O port

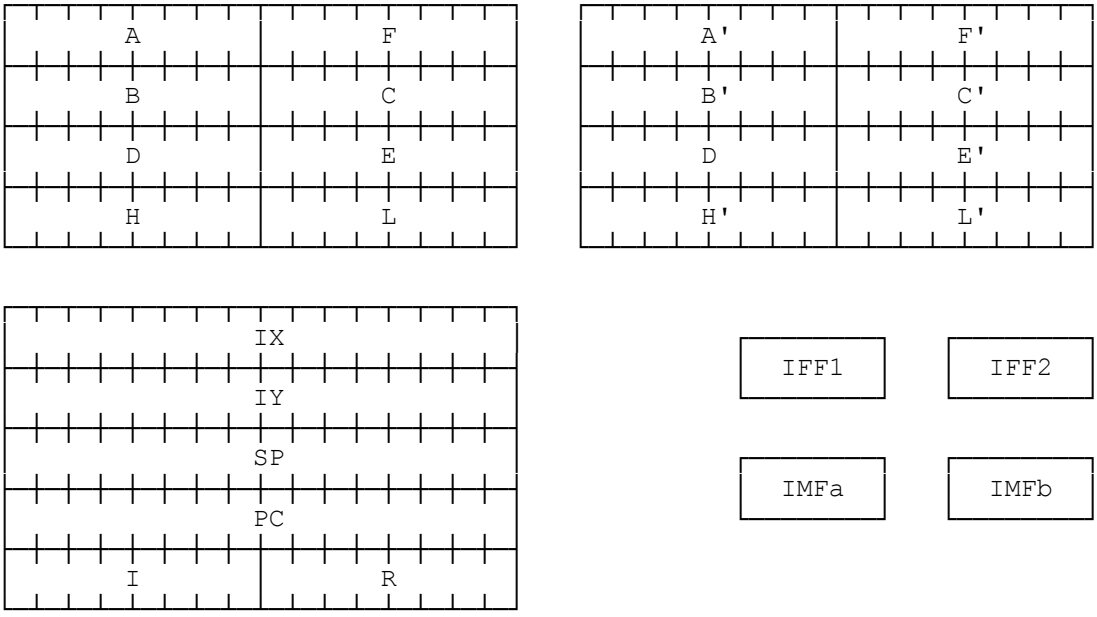
These extra instructions have been included here, and are indicated by a tilde "~" (which is NOT a part of the instruction).

Care should be taken in the use of these extra instructions. Any code using one or more of these instructions will NOT run properly on a Z-80 based system! It is suggested, therefore, that these instructions not be used except for machine-specific utilities.

Registers

The Zilog Z-80 microprocessor and its clones contain 208 bits of random accessed memory arranged as two groups of eight 8-bit registers and one group of two 8-bit registers and four 16-bit registers, all of which are directly accessible to the programmer. In addition, there are also four flip-flops used to control interrupt functions which are also, to a limited degree, controllable by the programmer.

This register set provides a significant improvement over that of its predecessor, the Intel 8080 microprocessor. The register set may be depicted as being arranged in the following manner.



The two groups of eight 8-bit registers are known as the main and alternative register sets, and serve absolutely identical functions; in fact, only one group may be used at one time, and that group, whichever it may be, becomes the “main” register set, while the other group is the “alternative” register set. Each group consists of an accumulator, “A,” a status or flag register “F” (the accumulator and status register are collectively known as the processor status word), and three pairs of general purpose registers “B” and “C,” “D” and “E,” and “H” and “L,” which may be used as six 8-bit registers or three 16-bit registers, depending upon the operations being performed. The “HL” register pair also may serve as a 16-bit accumulator for certain special functions.

In addition to the above groups, there is single group consisting of two 8-bit registers and four 16-bit registers. The 8-bit registers are the interrupt vector register “I” and the memory refresh register “R”. The 16-bit registers are the index registers “IX” and “IY,” the stack pointer “SP,” and the program counter “PC.”

Standard usages for the registers are:

A Accumulator

The 8-bit A register is the accumulator, is used to store a byte value prior to an operation, and normally contains the results of the operation when done (the compare instructions are an exception).

This is the primary register for all 8-bit arithmetic and logical operations.

F Status or Flag Register

The 8-bit F register contains the six operation status flags. The operation and functions of these flags are discussed in detail in a following section.

AF Processor Status Word

The 16-bit AF processor status word consists of the accumulator and the status register taken as a register pair, and is used primarily by the PUSH, POP, and EX instructions.

It is important to remember that if the AF processor status word is taken as a true 16-bit register pair, the A register is the upper byte and the F register is the lower byte. This distinction becomes critical if a set of false flags are generated in another register then moved to the F register via a PUSH and POP.

B General Purpose Register/Counter

The 8-bit B register may be used as a general purpose 8-bit register.

The B register is also used as an 8-bit unsigned loop counter for the DJNZ instruction or 8-bit unsigned repetition counter for the INIR, INDR, OTIR and OTDR instructions.

C General Purpose/Port-Indirect Register

The 8-bit C register may be used as a general purpose 8-bit register.

The C register is also the register that must contain the 8-bit I/O port address for port-indirect instructions. No other register may be used for port-indirect addressing.

BC General Purpose Register Pair/Counter

The 16-bit BC register pair consists of the B and C registers taken together and may be used as a general purpose 16-bit register.

The BC register pair may also be used as an unsigned 16-bit repetition counter for the LDIR, LDDR, CPIR and CPDR instructions.

D General Purpose Register

The 8-bit D register may be used as a general purpose 8-bit register.

E General Purpose Register

The 8-bit E register may be used as a general purpose 8-bit register.

DE General Purpose Register Pair

The 16-bit DE register pair consists of the D and E registers taken together, and may be used as a general purpose 16-bit register.

H General Purpose Register

The 8-bit H register may be used as a general purpose 8-bit register.

L General Purpose Register

The 8-bit L register may be used as a general purpose 8-bit register.

HL General Purpose/Memory-Indirect Register/16-Bit Accumulator

The 16-bit HL register pair consists of the H and L registers taken together, and may be used as a general purpose 16-bit register.

The HL register pair also serves as the primary memory-indirect addressing register (other register pairs may only be used with the accumulator and the LD instruction).

With 16-bit arithmetic instructions the HL register pair serves the function of a limited accumulator.

IX Index/Memory-Indirect Register/16-Bit Accumulator

The 16-bit IX register serves as the indexed memory base address pointer for indexed memory operations. Traditionally, the IX register serves as the "source" index in multi-indexed operations.

For a few limited instructions, the IX register may also serve as a memory-indirect addressing register or a limited 16-bit accumulator.

IY Index/Memory-Indirect Register/16-Bit Accumulator

The 16-bit IY register serves as the indexed memory base address pointer for indexed memory operations. Traditionally, the IY register serves as the "destination" index in multi-indexed operations.

For a few limited instructions, the IY register may also serve as a memory-indirect addressing register or a limited 16-bit accumulator.

SP Stack Pointer

The 16-bit SP register is used as the stack pointer, maintaining at all times the current bottom of the stack address. Note that the stack is inverted (bottom to top).

PC Program Counter

The 16-bit PC register is used as the program counter. After each instruction not explicitly manipulating the PC register, the PC register is incremented the appropriate number of times so that it points to the next instruction.

I Interrupt Vector Register

The 8-bit I register serves as the interrupt vector register for mode 2 interrupts. A detailed explanation of the use of interrupts in the three modes is given later.

R Memory Refresh Register

The 8-bit R register serves as a memory refresh register for user transparent dynamic random access memory refresh. This register is initialized to 00 at reset, then increments once and is placed on the address bus for refresh purposes during each machine fetch cycle. It is the responsibility of the memory arbitration circuitry or of the memory controller to initiate a refresh cycle at this time.

Since, in a normal application program environment, it is virtually impossible to predict the value of this register, fetching that value with the LD A,R instruction provides a reasonably reliable "random" number.

Interrupt Operation

The Z-80 processor has an elegant yet simple interrupt mechanism, controlled by the four interrupt flip-flops. The four flip-flops are divided into two pairs: the interrupt status flip-flops "IFF1" and "IFF2," used to control and/or indicate the current interrupt status, and the interrupt mode flip-flops "IMFa" and "IMFb," used to control the current interrupt mode.

IFF1 Primary Interrupt Status Flip-Flop

The IFF1 flip-flop status determines whether or not the maskable interrupts are enabled. If IFF1 is set to 0, then interrupts are disabled. If IFF1 is set to 1, then interrupts are enabled. The programmer has full control over this flip-flop through the use of the EI and DI instructions. Current status may be obtained by executing the LD A,I or LD A,R instruction, then querying the parity flag, which is set to the status of IFF1.

The non-maskable interrupt, by definition, cannot be disabled, and ignores the status of IFF1.

IFF2 Secondary Interrupt Status Flip-Flop

The IFF2 flip-flop serves as a backup to IFF1 in order to preserve interrupt status through a non-maskable interrupt. Assertion of a non-maskable interrupt causes the transfer of the state of IFF1 into IFF2, then processes according to the interrupt instructions, which should end with a RETN (return from non-maskable interrupt) instruction. The RETN instruction restores the original interrupt status by transferring the state of IFF2 into IFF1.

Execution of the EI or DI instructions sets IFF2 to the same state as IFF1: 0 for the DI instruction and 1 for the EI instruction.

IMFa and IMFb Interrupt Mode Flip-Flops

The IMFa and IMFb flip-flops operate together to determine the interrupt mode:

IMFa	IMFb	Interrupt Mode

0	0	Mode 0
0	1	Not used
1	0	Mode 1
1	1	Mode 2

In interrupt mode 0, assertion of an interrupt causes the processor to fetch the instruction code placed on the data bus by the interrupting peripheral and execute that instruction six times. This instruction would normally be one of the RST instructions, causing the processor to restart operation at the vector address passed in the instruction.

In interrupt mode 1, assertion of an interrupt causes the program counter to be set to 0038, causing the processor to commence executing the code found at that address. This is similar to execution of an RST 38 instruction, save that pointers are NOT saved. Interrupt mode 1 is often used by debuggers and other diagnostic programs to allow simple single step trapping.

In interrupt mode 2, assertion of an interrupt causes the interrupting peripheral to select the address for the interrupt vector. This is done by having the processor take the byte placed on the data bus by the interrupting peripheral as the lower 8-bits of the address and the I (interrupt vector) register as the upper 8-bits of the address. In this manner, the interrupt routine may be located anywhere in memory, indeed, there may be multiple interrupt routines for differing circumstances within the same body of code, selection being made via the LD I,A instruction at varying places in the parent program.

Non-Maskable Interrupts

All discussions of enabling or disabling interrupts, interrupt modes, and the like presume maskable interrupts. If a non-maskable interrupt is received, the processor sets the state of IFF2 to that of IFF1, clears IFF1 (disabling maskable interrupts for the duration of the non-maskable interrupt routine), sets the program counter to 0066, and executes the code at that location.

Under TurboDOS, having an interrupt service routine at that location would clash with the “base page” (discussed under programming). Therefore, in a TurboDOS environment either the non-maskable interrupt is not used (usual case) or special hardware considerations must take place, such as replacing the non-maskable interrupt with a reset.

Status Byte— Instruction Status Flags

Each 8-bit register, except the status register, may be further depicted as having eight bits arranged as follows:

7	6	5	4	3	2	1	0

In like manner, a 16-bit register may be depicted as:

F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

The status register is somewhat different, as rather than standard data bits its contents are the operation status flags. These flags are each one bit in the status register, and program operation directly affects and depends upon the status of the flags at any given time.

SF	ZF	--	HF	--	PF	NF	CF
7	6	5	4	3	2	1	0

The six flags (bits 5 and 3 are not used) represented in the flag register fulfill the following functions:

SF Sign Flag (bit 7)

The sign flag represents the sign of the result of an arithmetic or logical operation. Since all operations are presumed to be on signed integers, the sign flag is identical to the most significant bit of the operation result, and is set (1) for a negative value and cleared (0) for a positive value.

ZF Zero Flag (bit 6)

The zero flag is set or cleared based upon the results of certain arithmetic and logical operations. If the result of one of these operations is zero, then the flag is set, else the flag is cleared.

NOTE: The zero flag is always opposite the status of the result. If the result is 0, the flag is 1: if the result is not 0, the flag is 0.

HF Half-Carry Flag (bit 4)

The half-carry flag is set or reset according to the results of bits 3 or 4 (or 11 or 12) after certain operations. The half-carry flag is, unlike the other flags, not directly available to the programmer for conditional operations. It is used, however, by the DAA instruction to allow correction of BCD addition or subtraction results.

PF Parity/Overflow (P/V) Flag (bit 2)

The parity/overflow flag is set or reset depending upon the operation being performed.

For arithmetical operations, this flag is set (1) to indicate an overflow condition, that is, when the result is greater than the maximum possible integer (+127) or less than the minimum possible integer (–128).

For logical operations, this flag indicates the parity of the result; if the number of 1s in the result is odd (1,3,5,7), this flag is reset (0), if the number of 1s is even (0,2,4,6), this flag is set (1).

NF Subtraction Flag (bit 1)

The subtraction flag is set (1) if a subtraction or like operation is performed, and reset (0) is an addition or like operation is performed. Like the half-carry flag, this flag is used by the DAA operation, but is also available to the programmer for certain conditional functions.

CF Carry Flag (bit 0)

The carry flag is set (1) for any addition function that causes a “carry” beyond the accumulator, or for any subtraction function that does not cause a “borrow.” This flag is reset if the reverse is true.

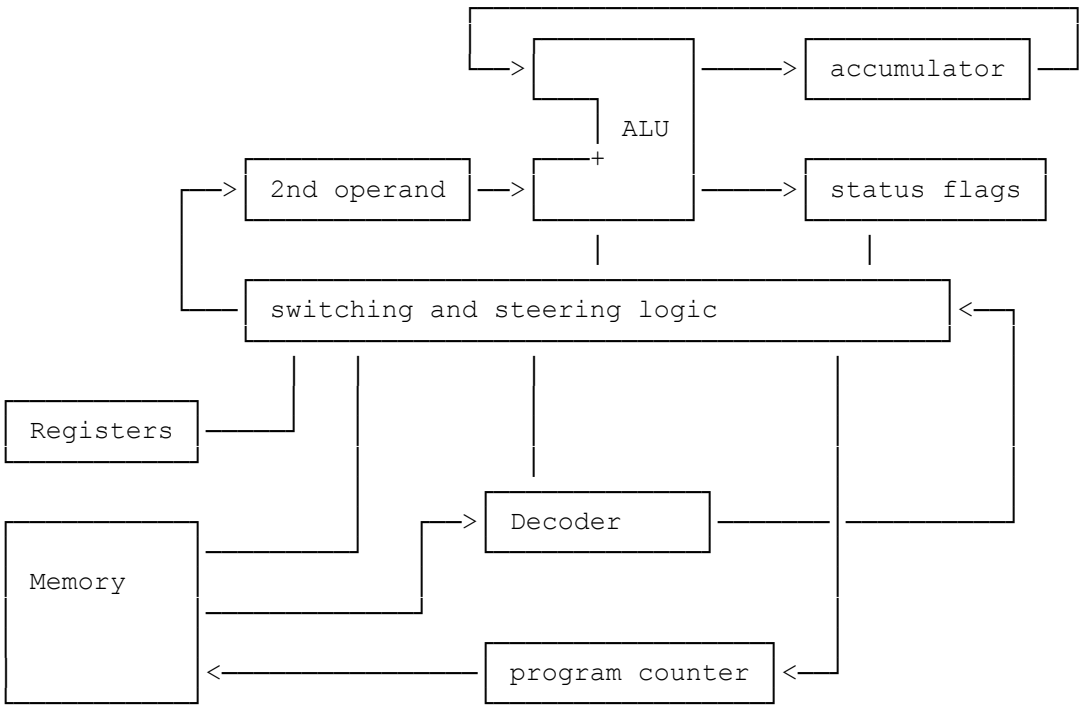
Often it is desirous to set the status of one or more flags prior to executing a particular instruction. The SCF instruction sets the carry flag to 1, and the CCF instruction compliments the carry flag. Various “null” instructions sets or clear particular flags without changing the values of the operands. The AND A instruction, for example, sets the sign, zero and parity/overflow flags to the conditions of the accumulator value, sets the half-carry flag, and clears the subtraction and carry flags. Similarly, the OR A instruction does exactly the same thing except that the half-carry flag is cleared.

General Operation

The Z-80 instruction set consists of operation upon the above registers and flags. These instructions are stored in memory as op-codes of one, two, three, or four bytes. The program counter always points to the next op-code to be executed.

For some operations more than one set of op-codes is possible. In these cases simple and virtually self explanatory tables may be found in the instruction code chapter.

A highly simplified diagram indicating the method in which the Z-80 operates is:



In a super-simplified fashion, to perform a given function, say execute the AND D instruction, the logic unit looks to the program counter for the current memory address, fetches a byte from that address and passes it to the instruction decoder. The instruction decoder sees that it is the instruction AND D, and tells the steering and switching logic to place the "D" register into the 2nd operand buffer (this could just as easily be an indirect or indexed memory byte or immediate byte, according to the requirements of the instruction) and to set the ALU (Arithmetic and Logic Unit) to convert to an AND operation logic unit. The ALU then ANDs the accumulator, acting as the 1st operand, and the 2nd operand, placing the result in the accumulator and setting the appropriate status flags. The steering and switching logic then bumps the program counter so that it points to the next memory address, ready for the next instruction.

This type of operation continues, with variations, throughout the length of the code. In fact, unless some looping provision is made, or a HALT instruction is executed, this process is infinite, with the program counter constantly cycling through memory. The processor must read and act upon the next byte in memory, whatever it is.

In essence, a microprocessor, such as the Z-80, can be defined as a logic circuit that can read an instruction, logically alter its internal structure in a manner conditional upon what it read, perform the logical or arithmetic function defined by that structure, save the result, and read its next instruction.

Conditional Branching

By checking the status of the various flags to determine current conditions, the conditional program transfer instructions allow for conditional branching. These conditional program transfer instructions are:

JP	cond,addr	Jump absolute on condition
JR	cond,disp	Jump relative on condition
CALL	cond,addr	Call on condition
RET	cond	Return on condition

The conditions that may be checked are:

cond	branch if:	description	comments
NZ	ZF = 0	branch on not zero	
Z	ZF = 1	branch on zero	
NC	CF = 0	branch on not carry	
C	CF = 1	branch on carry	
PO	PF = 0	branch on parity odd	no relative branching
PE	PF = 1	branch on parity even	no relative branching
P	SF = 0	branch on plus/positive	no relative branching
M	SF = 1	branch on minus/negative	no relative branching

Since there are only a limited number of flags, the various mathematical and logical conditions required to meet all normal cases sometimes require more than one instruction. The following code segments may be used to provide the required functionality, with the required values in the accumulator and the B register (obviously, the second value may be in any register, in memory, or immediate), and are non-destructive (A and B are left unchanged):

```
;Branch if A < 0:
        CP      00              ;Is A negative?
        JP      C,addr          ;If yes, branch
;
;Branch if A <= 0:
        OR      A               ;Is A negative or zero?
        JP      Z,addr          ;If A is zero, branch
        JP      M,addr          ;If A is negative, branch
;
;Branch if A = 0:
        OR      A               ;Is A zero?
        JP      Z,addr          ;If yes, branch
;
```

```
;Branch if A >= 0:
        OR      A               ;Is A positive or zero?
        JP      P,addr          ;If yes, branch
;
;Branch if A > 0:
        OR      A               ;Is A positive, but not zero?
        JR      Z,$+05          ;If no, continue
        JP      P,addr          ;If yes, branch
;
;Branch if A <> 0:
        OR      A               ;Is a zero?
        JP      NZ,addr         ;If no, branch
;
;Branch if A < B:
        CP      B               ;Is A less than B?
        JP      C,addr          ;If yes, branch
;
;Branch if A <= B:
        CP      B               ;Is A less than or equal to B?
        JP      Z,addr          ;If A equals B, branch
        JP      C,addr          ;If A is less than B, branch
;
;Branch if A = B:
        CP      B               ;Is A equal to B?
        JP      Z,addr          ;If yes, branch
;
;Branch if A >= B:
        CP      B               ;Is A greater than or equal to B?
        JP      NC,addr         ;If yes, branch
;
;Branch if A > B:
        CP      B               ;Is A greater than B?
        JR      Z,$+05          ;If A is equal to B, continue
        JP      NC,addr         ;If A is greater than B, branch
;
;Branch if A <> B:
        CP      B               ;Is A equal to B?
        JP      NZ,addr         ;If no, branch
;
;Branch if A[0] = 0:
```

```
        RRCA                    ;Fetch A[0]
        RLCA                    ;Restore A
        JP      NC,addr         ;If A[0]=0, branch
;
;Branch if A[0] = 1:
        RRCA                    ;Fetch A[0]
        RLCA                    ;Restore A
        JP      C,addr          ;If A[0]=1, branch
;
;Branch if A[7] = 0:
        OR      A               ;Fetch A[7]
        JP      P,addr          ;If A[7]=0, branch
;
;Branch if A[7] = 1:
        OR      A               ;Fetch A[7]
        JP      M,addr          ;If A[7]=1, branch
;
;Branch if r[b] = 0:
        BIT     b,r             ;Fetch r[b]
        JP      NZ,addr         ;If r[b]=0, branch
;
;Branch if r[b] = 1:
        BIT     b,r             ;Fetch r[b]
        JP      Z,addr          ;If r[b]=1, branch
```

There are many other possibilities, especially if the test does not have to be non-destructive, allowing branching for almost any combination of conditions.

Flag Operation

Basic Flag Operation

The Z-80 flag register is an 8-bit register (actually the low-order byte of the processor status word register pair, with the accumulator as the high-order byte), with each of the six flags represented by a single bit:

SF	ZF	--	HF	--	PF	NF	CF
7	6	5	4	3	2	1	0

Notice that bits 3 and 5 are not used.

Any given instruction may affect a given flag in one of six ways:

The instruction may manipulate the flag in a normal manner, in keeping with the flag definition and according to the general rules.

The instruction may manipulate the flag in a non-standard manner, not in keeping with the flag definition and/or according to rules peculiar to that instruction.

The instruction may manipulate the flag in an undefined manner, where the condition of the flag is unpredictable.

The instruction may set the flag (make the bit a 1).

The instruction may clear the flag (make the bit a 0).

The instruction may ignore the flag and leave it unchanged.

Normal operation of each of the six flags is as follows:

CF Carry Flag, bit 0:

When functioning in a normal carry manner, the carry flag is set whenever an operation causes a carry from the most significant bit of the result; otherwise, it is cleared.

When functioning in a normal borrow manner, the carry flag is set whenever an operation causes a borrow into the most significant bit of the result; otherwise, it is cleared.

When functioning in a normal shifting manner, the carry flag is set whenever the bit shifted out of the result is a 1; otherwise, it is cleared.

NF Subtraction Flag, bit 1:

When functioning in a normal manner, the subtraction flag is set whenever a subtraction or like operation takes place; otherwise, it is cleared.

PF Parity/Overflow Flag, bit 2:

When functioning in a normal parity manner, the parity/overflow flag is set whenever a logical operation causes an even number of 1 bits to appear in the result; otherwise, it is cleared. This is seen as:

PF ← NOT result[7 BITSUM 0]

Since BITSUM is a 1-bit addition, even numbers of 1's produce a 0 and odd numbers of 1's produce a 1. Then, since the parity flag is NOT the BITSUM function, the parity/overflow flag is set whenever the BITSUM function yields a 0, and cleared whenever the BITSUM function yields a 1.

When functioning in a normal overflow manner, the parity/overflow flag is set whenever an addition or like operation causes an overflow to occur; otherwise, it is cleared. An overflow occurs whenever there is a transition from a value of 00–7F to a value of 80–FF (0000–7FFF to 8000–FFFF for 16-bit operations) during an addition or like operation. This condition is flagged as an error because while 7F = +127 decimal, 80 = –128, not +128 decimal.

When functioning in a normal underflow manner, the parity/overflow flag is set whenever a subtraction or like operation causes an underflow to occur; otherwise, is cleared. An underflow occurs whenever there is a transition from a value of FF–80 to a value of 7F–00 (FFFF–8000 to 7FFF–0000 for 16-bit operations) during a subtraction or like operation. This condition is flagged as an error because while 80 = –128 decimal, 7F = +127, not –129 decimal.

HF Half-carry Flag, bit 4:

When functioning in a normal half-carry manner, the half-carry flag is set whenever an addition or like operation causes a bit-3 carry; otherwise, it is cleared. A bit-3 carry indicates a carry from bit 3 into bit 4 of the result.

When functioning in a normal half-borrow manner, the half-carry flag is set whenever a subtraction or like operation causes a bit-4 borrow; otherwise, it is cleared. A bit-4 borrow is a borrow from bit 4 into bit 3 of the result.

The half-carry flag is not normally used directly by the programmer, but is used to provide a “look ahead” operation when cascading instructions, most notably if the second instruction is the DAA instruction.

ZF Zero Flag, bit 6:

When functioning in a normal zero manner, the zero flag is set whenever an operation sets the result to zero; otherwise, it is cleared. This is seen as:

ZF ← NOT result[7 ORSUM 0] for 8-bit operations

ZF ← NOT result[15 ORSUM 0] for 16-bit operations

Since, when ORing if ANY bit is a 1 the result is a 1, the ORSUM expression yields a 0 if and only if ALL the bits are 0, else it yields a 1. Then, since the zero flag is NOT the ORSUM function, the flag is set if and only if all bits of the result are 0, else the flag is cleared.

SF Sign Flag, bit 7:

When functioning in a normal sign manner, the sign flag is set if the value of the result of the operation is negative; otherwise, it is cleared. Since a negative value has its most significant bit set, this means:

SF ← result[7] for 8-bit operations

SF ← result[15] for 16-bit operations

Flag Operations Table

The following table contains the flag operation parameters for every instruction in alphanumeric order. Those instructions with a "~" are Z-180 instructions, not part of the Z-80 instruction set.

The following special codes are used:

#n	Flag is manipulated in a non-standard manner, see note "#n".
—	Flag is ignored.
0	Flag is cleared.
1	Flag is set.
B	Carry flag is manipulated in a normal borrow manner.
C	Carry flag is manipulated in a normal carry manner.
P	Parity/overflow flag is manipulated in a normal parity manner.
S	Sign flag is manipulated in a normal sign manner.
U	Parity/overflow flag is manipulated in a normal underflow manner.
V	Parity/overflow flag is manipulated in a normal overflow manner.
X	Flag is manipulated in an undefined manner.
Z	Zero flag is manipulated in a normal zero manner.
b	Half-carry flag is manipulated in a normal borrow manner.
c	Half-carry flag is manipulated in a normal carry manner.

	Instruction	SF	ZF	HF	PF	NF	CF
	ADC ,source	S	Z	c	V	0	C
	ADC HL,source	S	Z	#1	V	0	C
	ADD A,source	S	Z	c	V	0	C
	ADD reg16,source	—	—	#1	—	0	C
	AND source	S	Z	1	P	0	0
	BIT bit,source	X	#2	1	X	0	—
	CALL addr	—	—	—	—	—	—
	CALL cond,addr	—	—	—	—	—	—
	CCF	—	—	#3	—	0	#3
	CP source	S	Z	b	U	1	B
	CPD	S	Z	b	#4	1	—
	CPDR	S	Z	b	#4	1	—
	CPI	S	Z	b	#4	1	—
	CPIR	S	Z	b	#4	1	—
	CPL	—	—	1	—	1	—
	DAA	S	Z	c	P	—	C
	DEC (mem)	S	Z	b	U	1	—
	DEC reg	S	Z	b	U	1	—
	DEC reg16	—	—	—	—	—	—
	DI	—	—	—	—	—	—
	DJNZ disp	—	—	—	—	—	—
	EI	—	—	—	—	—	—
	EX AF,AF'	#5	#5	#5	#5	#5	#5
	EX left,right	—	—	—	—	—	—
	EXX	—	—	—	—	—	—
	HALT	—	—	—	—	—	—
	IM mode	—	—	—	—	—	—
	IN A,(port)	—	—	—	—	—	—
	IN reg,(C)	S	Z	0	P	0	—
~	IN0 reg,(port)	S	Z	0	P	0	—
	INC (mem)	S	Z	c	V	0	—
	INC reg	S	Z	c	V	0	—

	INC reg16	—	—	—	—	—	—
	IND	X	#6	X	X	1	—
	INDR	X	1	X	X	1	—
	INI	X	#6	X	X	1	—
	INIR	X	1	X	X	1	—
	JP (reg16)	—	—	—	—	—	—
	JP addr	—	—	—	—	—	—
	JP cond,addr	—	—	—	—	—	—
	JR cond,disp	—	—	—	—	—	—
	JR disp	—	—	—	—	—	—
	LD dest,source	—	—	—	—	—	—
	LD A,I	#7	#7	0	#7	0	—
	LD A,R	#7	#7	0	#7	0	—
	LDD	—	—	0	#4	0	—
	LDDR	—	—	0	0	0	—
	LDI	—	—	0	#4	0	—
	LDIR	—	—	0	0	0	—
~	MLT regpr	—	—	—	—	—	—
	NEG	S	Z	b	U	1	#8
	NOP	—	—	—	—	—	—
	OR source	S	Z	1	P	0	0
~	OTDM	#9	#6	#9	#9	#10	#9
~	OTDMR	0	1	0	1	#10	0
	OTDR	X	1	X	X	1	X
~	OTIM	#9	#6	#9	#9	#10	#9
~	OTIMR	0	1	1	1	#10	0
	OTIR	X	1	X	X	1	X
	OUT (port),A	—	—	—	—	X	—
	OUT (C),reg	—	—	—	—	X	—
~	OUT0 (port),reg	—	—	—	—	—	—
	OUTD	X	#6	X	X	1	—
	OUTI	X	#6	X	X	1	—
	POP AF	#11	#11	#11	#11	#11	#11
	POP reg16	—	—	—	—	—	—
	PUSH reg16	—	—	—	—	—	—
	RES bit,source	—	—	—	—	—	—
	RET	—	—	—	—	—	—
	RET cond	—	—	—	—	—	—
	RETI	—	—	—	—	—	—
	RETN	—	—	—	—	—	—
	RL dest	S	Z	0	P	0	#12
	RLA	—	—	0	—	0	#12
	RLC dest	S	Z	0	P	0	#12
	RLCA	—	—	0	—	0	#12
	RLD	S	Z	0	P	0	—
	RR dest	S	Z	0	P	0	#13
	RRA	—	—	0	—	0	#13
	RRC dest	S	Z	0	P	0	#13
	RRCA	—	—	0	—	0	#13
	RRD	S	Z	0	P	0	—
	RST vector	—	—	—	—	—	—
	SBC A,source	S	Z	b	U	1	B
	SBC HL,source	S	Z	#14	U	1	C
	SCF	—	—	0	—	0	1

	SET bit,source	—	—	—	—	—	—
	SLA dest	S	Z	0	P	0	#12
~	SLP	—	—	—	—	—	—
	SRA dest	S	Z	0	P	0	#13
	SRL dest	S	Z	0	P	0	#13
	SUB source	S	Z	b	U	1	B
~	TST source	S	Z	1	P	0	0
~	TSTIO immed	S	Z	1	P	0	0
	XOR source	S	Z	1	P	0	0

Notes:

#1 HF = 1 if bit-11 carry, else 0
CF = 1 if bit-15 carry, else 0
#2 ZF = NOT source[bit]
#3 HF = CF before instruction
CF = NOT CF before instruction
#4 PF = 1 if BC <> 0, else 0.
#5 All flags assume the values previously held by the F' register.
#6 ZF = 1 if B = 0, else 0.
#7 SF = 1 if I or R < 0, else 0.
ZF = 1 if I or R = 0, else 0.
PF = status of IFF2.
#8 CF = 1 if A <> 0 before operation, else 0.
#9 SF = 1 if B < 0, else 0.
HF = 1 if B bit-4 borrow, else 0.
PF = 1 if B parity even, else 0.
CF = 1 if B bit 8 borrow, else 0.
#10 NF = 1 if (HL) < 0, else 0.
#11 SF = (SP – 2)[7]
ZF = (SP – 2)[6]
HF = (SP – 2)[4]
PF = (SP – 2)[2]
NF = (SP – 2)[1]
CF = (SP – 2)[0]
#12 CF = dest[7] before operation.
#13 CF = dest[0] before operation.
#14 HF = 1 if bit-12 borrow, else 0.
CF = 1 if bit-16 borrow, else 0.

Instruction Index in Functional Order

Data Transfer Instructions

Data Exchange Instructions

Block Transfer Instructions

Block Search Instructions

CPI	Compare accumulator with (HL) and increment	13–20
CPIR	Compare accumulator with (HL), increment and repeat	13–21
CPD	Compare accumulator with (HL) and decrement	13–18
CPDR	Compare accumulator with (HL), decrement and repeat	13–19

Arithmetic Instructions

ADD dest,source	Add source operand to destination operand	13–6
ADC dest,source	Add source operand plus carry to destination operand	13–4
INC dest	Increment destination operand	13–38
SUB source	Subtract source operand from accumulator	13–112
CP source	Compare source operand with accumulator	13–16
SBC dest,source	Subtract source operand less carry from destination	13–99
DEC dest	Decrement destination operand	13–25
NEG	Negate (two's compliment) accumulator	13–60
~ MLT regpr	Multiply bytes of 16-bit register pair	13–59
DAA	Decimal adjust accumulator	13–23

Logical Instructions

AND source	AND source operand with accumulator	13–8
~ TST source	Test source operand against accumulator	13–114
OR source	Inclusive-OR source operand with accumulator	13–62
XOR source	Exclusive-OR source operand with accumulator	13–117
CPL	One's compliment accumulator	13–22

CPU Control Instructions

NOP	No operation	13–61
HALT	Halt operation	13–32
~ SLP	Enter sleep or system stop mode	13–107
SCF	Set carry flag	13–101
CCF	Compliment carry flag	13–15
EI	Enable interrupts	13–29
DI	Disable interrupts	13–27
IM mode	Set interrupt mode	13–33

Rotate and Shift Instructions

RLA	Rotate accumulator left	13–86
RLCA	Rotate accumulator left circular	13–89
RRA	Rotate accumulator right	13–93
RRCA	Rotate accumulator right circular	13–96
RL dest	Rotate destination operand left	13–84
RLC dest	Rotate destination operand left circular	13–87
RR dest	Rotate destination operand right	13–91
RRC dest	Rotate destination operand right circular	13–94
RLD	Rotate left digit	13–90
RRD	Rotate right digit	13–97
SLA dest	Shift destination operand left arithmetic	13–105
SRA dest	Shift destination operand right arithmetic	13–108
SRL dest	Shift destination operand right logical	13–110

Bit Manipulation Instructions

SET bit,source	Set source operand bit	13–102
RES bit,source	Reset source operand bit	13–77
BIT bit,source	Test source operand bit	13–10

Program Transfer Instructions

JP addr	Unconditional direct absolute jump	13–44
JP (reg16)	Unconditional indirect absolute jump	13–44
JP cond,addr	Conditional absolute jump	13–44
JR disp	Unconditional relative jump	13–46
JR cond,disp	Conditional relative jump	13–46
DJNZ disp	Decrement and jump on not zero	13–28
CALL addr	Unconditional call	13–13
CALL cond,addr	Conditional call	13–13
RET	Unconditional return	13–80
RET cond	Conditional return	13–80
RETI	Return from interrupt	13–82
RETN	Return from non-maskable interrupt	13–83
RST vector	Restart	13–98

Input and Output Instructions

IN A,(port)	Input port-direct byte	13–35
~ IN0 reg,(port)	Input port-direct byte	13–37
IN reg,(C)	Input port-indirect byte	13–35
INI	Input port-indirect byte and increment	13–42
INIR	Input port-indirect byte, increment and repeat	13–43
IND	Input port-indirect byte and decrement	13–40
INDR	Input port-indirect byte, decrement and repeat	13–41
OUT (port),A	Output port-direct byte	13–70
~ OUT0 (port),reg	Output port-direct byte	13–72
OUT (C),reg	Output port-indirect byte	13–70
OUTI	Output port-indirect byte and increment	13–74
OTIR	Output port-indirect byte, increment and repeat	13–69
OUTD	Output port-indirect byte and decrement	13–73
OTDR	Output port-indirect byte, decrement and repeat	13–66
~ OTIM	Output port-indirect memory and increment	13–67
~ OTIMR	Output port-indirect memory, inc and repeat	13–68
~ OTDM	Output port-indirect memory and decrement	13–64
~ OTDMR	Output port-indirect memory, decrement, and repeat	13–65
~ TSTIO immed	Test immediate byte against I/O byte	13–116

Zilog Z-80 Instruction Set

Each instruction entry in this chapter consists of seven different sections of information: Header, Instruction, Operation, Description, Flags, Clocking, and Encoding.

Header

The Header is located at the top of each page and consists of one or two lines of information and a doubled separator line.

The first line contains the instruction base mnemonic at both the left and right margins, and a brief description of the instruction in the center. This line provides a fast look up when fanning pages either to the right or the left, much as the guide words in a dictionary.

The second line, if used, contains the inscription "~ Z-180 ~", indicating an instruction that is unique to the Z-180 processor, and not available with the Z-80 processor. These instructions are included here to provide for Z-180 programming over and above Z-80 programming.

The Header is set off from the rest of the entry by a doubled separator line. If a given instruction should require more than one page, an identical header is placed on each page of the instruction.

Instruction

The Instruction section consists of the mnemonics for the instruction, in generic form, along with a brief description.

For example, the Instruction section for the "AND" instruction is:

AND source AND source operand with accumulator

It can be seen here that the instruction operand is "source." with no attempt made at a finer differentiation. This differentiation is clearly expressed in other areas, the final and exact operands always being given in the Encoding section.

The Instruction section is set off from the rest of the instruction data by a single separator line.

Operation

The Operation section consists of a detailed in-order functional and/or mathematical listing.

The Operation section for the "AND" instruction is:

A[0] ← A[0] AND source[0]
A[1] ← A[1] AND source[1]
A[2] ← A[2] AND source[2]
A[3] ← A[3] AND source[3]
A[4] ← A[4] AND source[4]
A[5] ← A[5] AND source[5]
A[6] ← A[6] AND source[6]
A[7] ← A[7] AND source[7]

All flag operations are performed AFTER the arithmetic or logical operations, unless specifically indicated otherwise, and are not listed unless the flag operation is fundamental to the understanding of the operation of the basic instruction (such as in reiterative operations), or if the flag operation is special (not a normal use of the flag).

There are some special terms used that might need some explanation:

←	This symbol should be read as "is made equal to."
↔	This symbol should be read as "is exchanged with."
(...)	Parentheses indicate the expression "the memory byte whose address is" or "the memory byte whose address is in." The expression "A ← (HL)" means that register A is made equal to the byte whose address is in the HL register pair.
NOT	This reverses the condition: a 0 becomes a 1 and a 1 becomes a 0.
op[bit]	This expression resolves to a logical 0 or 1 equal to the value of bit "[bit]" of operand "op".

These and all other special terms and abbreviations are discussed in detail in the "Abbreviations, Terms, and Symbols" chapter of this document.

Description

The Description section provides a detailed, in-order, formal description of the operation of the instruction, affirming in words what was given logically/mathematically in the operation section.

The description for the “AND” instruction is:

> The accumulator is bitwise ANDed with the source operand.
> The source operand is left unchanged.

As in the Operation section, the description of flag operation is not given unless germane to the understanding of the overall operation of the instruction.

Flags

The Flags section provides a flag by flag indication of the post operation status of the various flags.

The Flags section of the “AND” instruction is:

SF: 1 if A < 0, else 0
ZF: 1 if A = 0, else 0
HF: 1
PF: 1 if parity even, else 0
NF: 0
CF: 0

The Flags section represents a clean, concise and orderly depiction of the flag status AFTER the operation is performed. In order to improve clarity and understanding, the Flags section gives the post operation status of all six flags, not just those manipulated by the instruction.

Detailed descriptions of the operations of each of the six flags are to be found in the “Flag Operation” portion of the Zilog Z-80 Processor Overview chapter of this document.

Clocking

The Clocking section contains two parameters, the number of machine cycles taken by the instruction and the number of tymes (timing states or processor clock cycles, also known as T-states) taken. The tymes for complex instructions are further divided into their interrupt sequence values as specified by Zilog (Z-180 instructions are not subdivided).

The Clocking section for the “AND” instruction is:

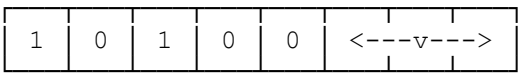
	M-cycles	Tymes
AND reg	1	4
AND (HL)	2	7 (4,3)
AND (Ii+d)	5	19 (4,4,3,5,3)
AND immed	2	7 (4,3)

The actual time an instruction takes is a function of the number of tymes for that instruction divided by the clock speed of the processor.

Encoding

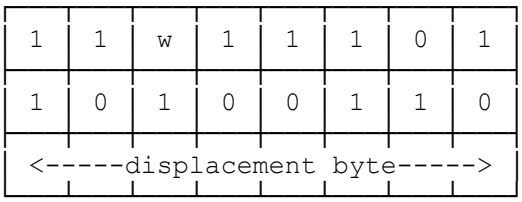
The Encoding section contains the actual bytes and op-codes for the instruction in a self-evident manner.

The Encoding section for the “AND” instruction is:

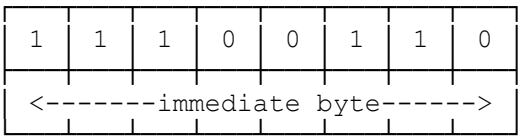


op

	v	op-code
AND B	000	A0
AND C	001	A1
AND D	010	A2
AND E	011	A3
AND H	100	A4
AND L	101	A5
AND (HL)	110	A6
AND A	111	A7



	w	op-code
AND (IX+d)	0	DD A6 db
AND (IY+d)	1	FD A6 db



	op-code
AND immed	E6 ib

It is in the Encoding section that all forms of the operand are explicitly shown, and are in boldface to emphasize this fact. These forms are also shown in the several supplementary tables elsewhere in this manual.

One note of comment. Inspection of the Encoding sections would lead one to believe that memory-indirect operands via the HL register pair are treated somewhat like a special register. This is indeed true. In the TDL instruction set, (HL) is “register” M.

Nowhere does more than one instruction appear on the same page. Whenever possible, only whole sections for a given instruction are presented on a given page. When this is impossible, section division is at logical boundaries. This practice improves clarity, but causes varying amounts of data to be given on each page. Clarity is more important than paper.

All instructions are given in alphanumeric order.

ADC dest,source add source operand plus carry to destination operand

Operation
dest ← dest + source + CF

Description
The destination operand is made equal to the sum of the destination operand plus the source operand plus the status of the carry flag.

The source operand is left unchanged.

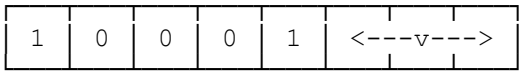
Flags

	If 8-bit Instruction:	If 16-bit Instruction:
SF:	1 if result < 0, else 0	1 if result < 0, else 0
ZF:	1 if result = 0, else 0	1 if result = 0, else 0
HF:	1 if bit-3 carry, else 0	1 if bit-11 carry, else 0
PF:	1 if overflow, else 0	1 if overflow, else 0
NF:	0	0
CF:	1 if bit-7 carry, else 0	1 if bit-15 carry, else 0

Clocking

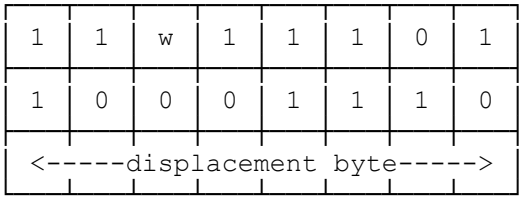
	M-cycles	Tymes
ADC A,reg	1	4
ADC A,(HL)	2	7 (4,3)
ADC A,(Ii+d)	5	19 (4,4,3,5,3)
ADC A,immed	2	7 (4,3)
ADC HL,regpr	4	15 (4,4,4,3)

Encoding



op

	v	op-code
ADC A,B	000	88
ADC A,C	001	89
ADC A,D	010	8A
ADC A,E	011	8B
ADC A,H	100	8C
ADC A,L	101	8D
ADC A,(HL)	110	8E
ADC A,A	111	8F

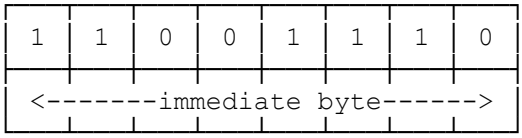


DD or FD

8E

db

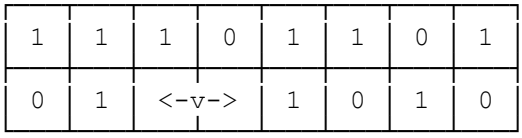
	w	op-code
ADC A,(IX+d)	0	DD 8E db
ADC A,(IY+d)	1	FD 8E db



CE

ib

	op-code
ADC A,immed	CE ib



ED

op

	v	op-code
ADC HL,BC	00	ED 4A
ADC HL,DE	01	ED 5A
ADC HL,HL	10	ED 6A
ADC HL,SP	11	ED 7A

ADD dest,source add source operand to destination operand

Operation
dest ← dest + source

Description
The destination operand is made equal to the sum of the destination operand plus the source operand. The source operand is left unchanged.

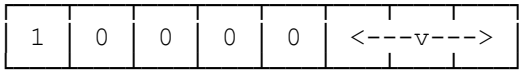
Flags

	If 8-bit Instruction:	If 16-bit Instruction:
SF:	1 if result < 0, else 0	Not Affected
ZF:	1 if result = 0, else 0	Not Affected
HF:	1 if bit-3 carry, else 0	1 if bit-11 carry, else 0
PF:	1 if overflow, else 0	Not Affected
NF:	0	0
CF:	1 if bit-7 carry, else 0	1 if bit-15 carry, else 0

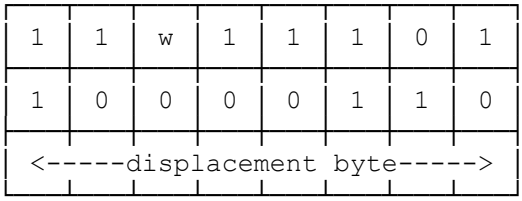
Clocking

	M-cycles	Tymes
ADD A,reg	1	4
ADD A,(HL)	2	7 (4,3)
ADD A,(Ii+d)	5	19 (4,4,3,5,3)
ADD A,immed	2	7 (4,3)
ADD HL,regpr	3	11 (4,4,3)
ADD Ii,regpr	3	11 (4,4,3)

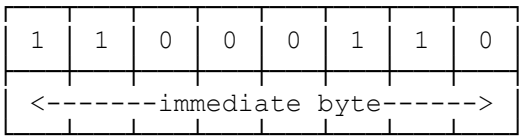
Encoding



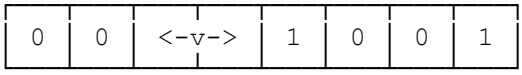
	v	op-code
ADD A,B	000	80
ADD A,C	001	81
ADD A,D	010	82
ADD A,E	011	83
ADD A,H	100	84
ADD A,L	101	85
ADD A,(HL)	110	86
ADD A,A	111	87



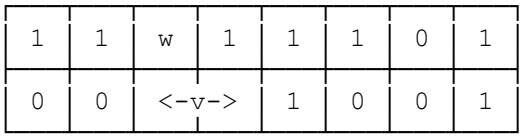
	w	op-code
ADD A,(IX+d)	0	DD 86 db
ADD A,(IY+d)	1	FD 86 db



	op-code
ADD A,immed	C6 ib



	v	op-code
ADD HL,BC	00	09
ADD HL,DE	01	19
ADD HL,HL	10	29
ADD HL,SP	11	39



	w	v	op-code
ADD IX,BC	0	00	DD 09
ADD IX,DE	0	01	DD 19
ADD IX,HL	0	10	DD 29
ADD IX,SP	0	11	DD 39
ADD IY,BC	1	00	FD 09
ADD IY,DE	1	01	FD 19
ADD IY,HL	1	10	FD 29
ADD IY,SP	1	11	FD 39

AND source AND source operand with accumulator

Operation

A[0] ← A[0] AND source[0]
A[1] ← A[1] AND source[1]
A[2] ← A[2] AND source[2]
A[3] ← A[3] AND source[3]
A[4] ← A[4] AND source[4]
A[5] ← A[5] AND source[5]
A[6] ← A[6] AND source[6]
A[7] ← A[7] AND source[7]

Description

The accumulator is bitwise ANDed with the source operand.
The source operand is left unchanged.

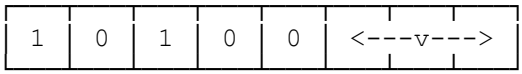
Flags

SF: 1 if A < 0, else 0
ZF: 1 if A = 0, else 0
HF: 1
PF: 1 if parity even, else 0
NF: 0
CF: 0

Clocking

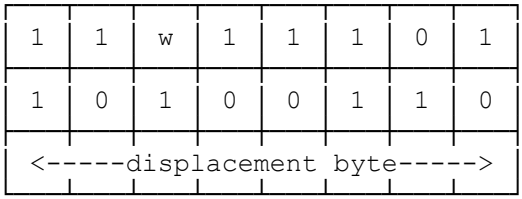
	M-cycles	Tymes
AND reg	1	4
AND (HL)	2	7 (4,3)
AND (Ii+d)	5	19 (4,4,3,5,3)
AND immed	2	7 (4,3)

Encoding



op

	v	op-code
AND B	000	A0
AND C	001	A1
AND D	010	A2
AND E	011	A3
AND H	100	A4
AND L	101	A5
AND (HL)	110	A6
AND A	111	A7

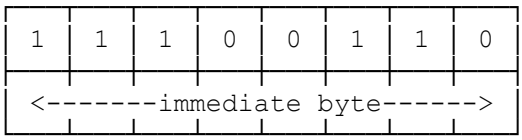


DD or FD

A6

db

	w	op-code
AND (IX+d)	0	DD A6 db
AND (IY+d)	1	FD A6 db



E6

ib

	op-code
AND immed	E6 ib

BIT bit,source test source operand bit

Operation
ZF ← NOT source[bit]

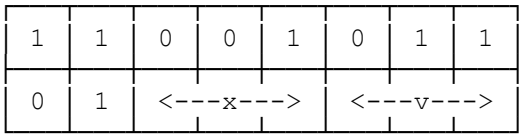
Description
The zero flag is set if the specified bit of the source operand is 0, otherwise, the zero flag is cleared.
The source operand is left unchanged.

Flags
SF: Undefined
ZF: 1 if reg[bit] = 0, else 0
HF: 1
PF: Undefined
NF: 0
CF: Not Affected

Clocking

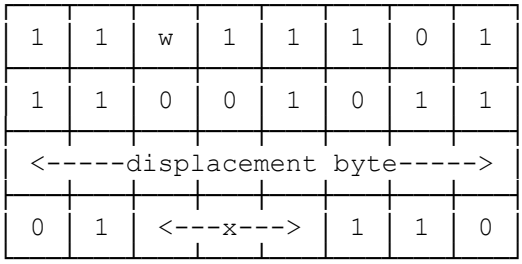
	M-cycles	Tymes
BIT bit,reg	2	8 (4,4)
BIT bit,(HL)	3	12 (4,4,4)
BIT bit,(Ii+d)	5	20 (4,4,3,5,4)

Encoding



	x	v	op-code
BIT 0,B	000	000	CB 40
BIT 0,C	000	001	CB 41
BIT 0,D	000	010	CB 42
BIT 0,E	000	011	CB 43
BIT 0,H	000	100	CB 44
BIT 0,L	000	101	CB 45
BIT 0,(HL)	000	110	CB 46
BIT 0,A	000	111	CB 47
BIT 1,B	001	000	CB 48
BIT 1,C	001	001	CB 49
BIT 1,D	001	010	CB 4A
BIT 1,E	001	011	CB 4B
BIT 1,H	001	100	CB 4C
BIT 1,L	001	101	CB 4D
BIT 1,(HL)	001	110	CB 4E
BIT 1,A	001	111	CB 4F
BIT 2,B	010	000	CB 50
BIT 2,C	010	001	CB 51
BIT 2,D	010	010	CB 52
BIT 2,E	010	011	CB 53
BIT 2,H	010	100	CB 54
BIT 2,L	010	101	CB 55
BIT 2,(HL)	010	110	CB 56
BIT 2,A	010	111	CB 57

BIT 3,B	011	000	CB 58
BIT 3,C	011	001	CB 59
BIT 3,D	011	010	CB 5A
BIT 3,E	011	011	CB 5B
BIT 3,H	011	100	CB 5C
BIT 3,L	011	101	CB 5D
BIT 3,(HL)	011	110	CB 5E
BIT 3,A	011	111	CB 5F
BIT 4,B	100	000	CB 60
BIT 4,C	100	001	CB 61
BIT 4,D	100	010	CB 62
BIT 4,E	100	011	CB 63
BIT 4,H	100	100	CB 64
BIT 4,L	100	101	CB 65
BIT 4,(HL)	100	101	CB 66
BIT 4,A	100	111	CB 67
BIT 5,B	101	000	CB 68
BIT 5,C	101	001	CB 69
BIT 5,D	101	010	CB 6A
BIT 5,E	101	011	CB 6B
BIT 5,H	101	100	CB 6C
BIT 5,L	101	101	CB 6D
BIT 5,(HL)	101	110	CB 6E
BIT 5,A	101	111	CB 6F
BIT 6,B	110	000	CB 70
BIT 6,C	110	001	CB 71
BIT 6,D	110	010	CB 72
BIT 6,E	110	011	CB 73
BIT 6,H	110	100	CB 74
BIT 6,L	110	101	CB 75
BIT 6,(HL)	110	110	CB 76
BIT 6,A	110	111	CB 77
BIT 7,B	111	000	CB 78
BIT 7,C	111	001	CB 79
BIT 7,D	111	010	CB 7A
BIT 7,E	111	011	CB 7B
BIT 7,H	111	100	CB 7C
BIT 7,L	111	101	CB 7D
BIT 7,(HL)	111	110	CB 7E
BIT 7,A	111	111	CB 7F



	w	x	op-code
BIT 0,(IX+d)	0	000	DD CB db 46
BIT 1,(IX+d)	0	001	DD CB db 4E
BIT 2,(IX+d)	0	010	DD CB db 56
BIT 3,(IX+d)	0	011	DD CB db 5E
BIT 4,(IX+d)	0	100	DD CB db 66
BIT 5,(IX+d)	0	101	DD CB db 6E
BIT 6,(IX+d)	0	110	DD CB db 76
BIT 7,(IX+d)	0	111	DD CB db 7E
BIT 0,(IY+d)	1	000	FD CB db 46
BIT 1,(IY+d)	1	001	FD CB db 4E
BIT 2,(IY+d)	1	010	FD CB db 56
BIT 3,(IY+d)	1	011	FD CB db 5E
BIT 4,(IY+d)	1	100	FD CB db 66
BIT 5,(IY+d)	1	101	FD CB db 6E
BIT 6,(IY+d)	1	110	FD CB db 76
BIT 7,(IY+d)	1	111	FD CB db 7E

CALL addr unconditional call

CALL cond,addr conditional call

Operation

If unconditional:
do call

If conditional:
NZ: if ZF = 0 then do call.
Z: if ZF = 1 then do call.
NC: if CF = 0 then do call.
C: if CF = 1 then do call.
PO: if PF = 0 then do call.
PE: if PF = 1 then do call.
P: if SF = 0 then do call.
M: if SF = 1 then do call.

If do call:
temp ← PC + 2
SP ← SP – 1
(SP) ← temp-high
SP ← SP – 1
(SP) ← temp-low
PC ← addr

Description

If unconditional:
The call is done.

If conditional:
A test is made of the appropriate flag.
If the condition is met, the call is done.
If the condition is not met, execution continues with the next instruction.

If the call is done:
The stack pointer is decremented.
The memory byte whose address is in the stack pointer is made equal to the high-order byte of the sum of the program counter plus 2.
The stack pointer is again decremented.
The memory byte whose address is in the stack pointer is made equal to the low-order byte of the sum of the program counter plus 2.
The program counter is made equal to the value of the immediate memory operand.
Execution continues at the address that is in the program counter.

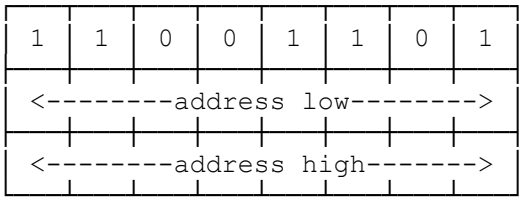
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

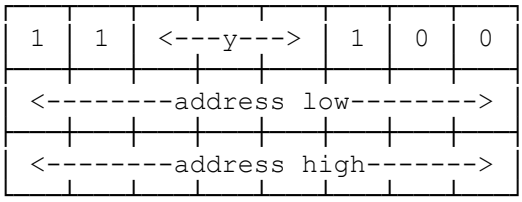
Clocking

	If condition is met:		If condition is not met:	
	M-cycles	Tymes	M-cycles	Tymes
CALL addr	5	17 (4,3,4,3,3)		
CALL cond,addr	5	17 (4,3,4,3,3)	3	10 (4,3,3)

Encoding



	op-code
CALL addr	CD al ah



	y	op-code
CALL NZ,addr	000	C4 al ah
CALL Z,addr	001	CC al ah
CALL NC,addr	010	D4 al ah
CALL C,addr	011	DC al ah
CALL PO,addr	100	E4 al ah
CALL PE,addr	101	EC al ah
CALL P,addr	110	F4 al ah
CALL M,addr	111	FC al ah

CCF compliment carry flag

Operation

HF ← CF
CF ← NOT CF

Description

The half-carry flag is set to the status of the carry flag.
The carry flag is complimented.

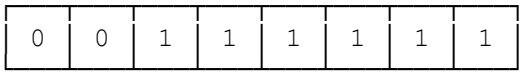
Flags

SF: Not Affected
ZF: Not Affected
HF: 1 if CF = 1 before operation, else 0
PF: Not Affected
NF: 0
CF: 1 if CF = 0 before operation, else 0

Clocking

	M-cycles	Tymes
CCF	1	4

Encoding



	op-code
CCF	3F

CP source compare source operand with accumulator

Operation
temp ← A – source

Description
The source operand is subtracted from the accumulator.

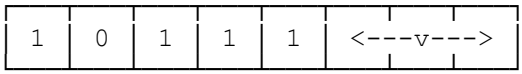
The accumulator is left unchanged.
The source operand is left unchanged.

Flags
SF: 1 if result < 0, else 0
ZF: 1 if result = 0, else 0
HF: 1 if bit-4 borrow, else 0
PF: 1 if underflow, else 0
NF: 1
CF: 1 if bit-8 borrow, else 0

Clocking

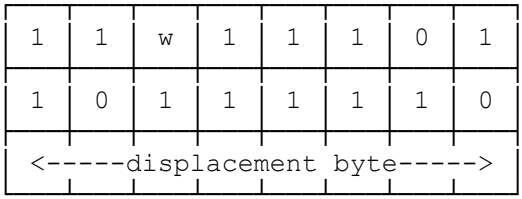
	M-cycles	Tymes
CP reg	1	4
CP (HL)	2	7 (4,3)
CP (Ii+d)	5	19 (4,4,3,5,3)
CP immed	2	7 (4,3)

Encoding



op

	v	op-code
CP B	000	B8
CP C	001	B9
CP D	010	BA
CP E	011	BB
CP H	100	BC
CP L	101	BD
CP (HL)	110	BE
CP A	111	BF

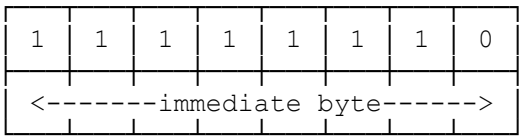


DD or FD

BE

db

	w	op-code
CP (IX+d)	0	DD BE db
CP (IY+d)	1	FD BE db



FE

ib

	op-code
CP immed	FE ib

CPD compare accumulator with (HL) and decrement

Operation

temp ← A – (HL)

HL ← HL – 1
BC ← BC – 1

PF ← BC[15 ORSUM 0]

Description

The memory byte whose address is in the HL register pair is subtracted from the accumulator.

The HL register pair is decremented.
The BC register pair is decremented.

If the BC register pair is zero, the parity/overflow flag is cleared; otherwise, the parity/overflow flag is set.

The accumulator is left unchanged.

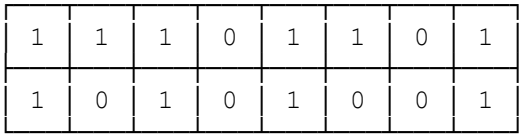
Flags

SF: 1 if result < 0, else 0
ZF: 1 if result = 0, else 0
HF: 1 if bit-4 borrow, else 0
PF: 1 if BC <> 0, else 0
NF: 1
CF: Not Affected

Clocking

	M-cycles	Tymes
CPD	4	16 (4,4,3,5)

Encoding



	op-code
CPD	ED A9

CPDR compare accumulator with (HL), decrement, and repeat

Operation

temp ← A – (HL)

HL ← HL – 1
BC ← BC – 1

PF ← BC[15 ORSUM 0]

If PF = 1, repeat operation

Description

The memory byte whose address is in the HL register pair is subtracted from the accumulator.

The HL register pair is decremented.
The BC register pair is decremented.

If the BC register pair is zero, the parity/overflow flag is cleared; otherwise, the parity/overflow flag is set.

If the parity/overflow flag is set, the operation is repeated.

The accumulator is left unchanged.

Flags

SF: 1 if result < 0, else 0
ZF: 1 if result = 0, else 0
HF: 1 if bit-4 borrow, else 0
PF: 0 after final iteration
NF: 1
CF: Not Affected

Clocking

	While PF = 1:		When PF = 0:	
	M-cycles	Tymes	M-cycles	Tymes
CPDR	5	21 (4,4,3,5,5)	4	16 (4,4,3,5)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	0	1	B9

	op-code
CPDR	ED B9

CPI compare accumulator with (HL) and increment

Operation

temp ← A – (HL)

HL ← HL + 1
BC ← BC – 1

PF ← BC[15 ORSUM 0]

Description

The memory byte whose address is in the HL register pair is subtracted from the accumulator.

The HL register pair is incremented.
The BC register pair is decremented.

If the BC register pair is zero, the parity/overflow flag is cleared; otherwise, the parity/overflow flag is set.

The accumulator is left unchanged.

Flags

SF: 1 if result < 0, else 0
ZF: 1 if result = 0, else 0
HF: 1 if bit-4 borrow, else 0
PF: 1 if BC <> 0, else 0
NF: 1
CF: Not Affected

Clocking

	M-cycles	Tymes
CPI	4	16 (4,4,3,5)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	0	1	A1

	op-code
CPI	ED A1

CPIR compare accumulator with (HL), increment, and repeat

Operation

temp ← A – (HL)

HL ← HL + 1
BC ← BC – 1

PF ← BC[15 ORSUM 0]

If PF = 1, repeat operation

Description

The memory byte whose address is in the HL register pair is subtracted from the accumulator.

The HL register pair is incremented.
The BC register pair is decremented.

If the BC register pair is zero, the parity/overflow flag is cleared; otherwise, the parity/overflow flag is set.

If the parity/overflow flag is set, the operation is repeated.

The accumulator is left unchanged.

Flags

SF: 1 if result < 0, else 0
ZF: 1 if result = 0, else 0
HF: 1 if bit-4 borrow, else 0
PF: 0 after final iteration
NF: 1
CF: Not Affected

Clocking

	While PF = 1:		When PF = 0:	
	M-cycles	Tymes	M-cycles	Tymes
CPIR	5	21 (4,4,3,5,5)	4	16 (4,4,3,5)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	0	1	B1

	op-code
CPIR	ED B1

CPL one's compliment accumulator

Operation

A[0] ← NOT A[0]
A[1] ← NOT A[1]
A[2] ← NOT A[2]
A[3] ← NOT A[3]
A[4] ← NOT A[4]
A[5] ← NOT A[5]
A[6] ← NOT A[6]
A[7] ← NOT A[7]

Description

The accumulator is bitwise complimented (one's complement).

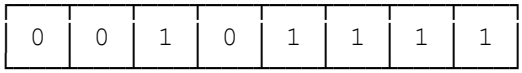
Flags

SF: Not Affected
ZF: Not Affected
HF: 1
PF: Not Affected
NF: 1
CF: Not Affected

Clocking

	M-cycles	Tymes
CPL	1	4

Encoding



2F

	op-code
CPL	2F

DAA decimal adjust accumulator

Operation

xx = 00 if:
(NF = 0 and CF = 0 and HF = 0 and A[hi] = 0–9 and A[lo] = 0–9) or (NF = 1 and CF = 0 and HF = 0)
xx = 06 if:
(NF = 0 and CF = 0 and HF = 0 and A[hi] = 0–8 and A[lo] = A–F) or (NF = 0 and CF = 0 and HF = 1 and A[hi] = 0–9)
xx = 60 if:
(NF = 0 and CF = 0 and HF = 0 and A[hi] = A–F and A[lo] = 0–9) or (NF = 0 and CF = 1 and HF = 0 and A [lo] = 0–9)
xx = 66 if:
(NF = 0 and CF = 0 and HF = 0 and A[hi] = 9–F and A[lo] = A–F) or (NF = 0 and CF = 0 and HF = 1 and A[hi] = A–F) or
(NF = 0 and CF = 1 and HF = 0 and A[lo] = A–F) or (NF = 0 and CF = 1 and HF = 1)
xx = 9A if:
(NF = 1 and CF = 1 and HF = 1)
xx = A0 if:
(NF = 1 and CF = 1 and HF = 0)
xx = FA if:
(NF = 1 and CF = 0 and HF = 1)

A ← A + xx

CF = 0 if:
x = 00 or xx = 06 or xx = FA
CF = 1 if:
= 60 or xx = 66 or xx = 9A or xx = A0
HF = 0 if:
xx = 00 or xx = 60 or xx = A0 or
(xx = 06 or xx = 66 or xx = 9A or xx FA) and there is no carry from bit 3
HF = 1 if:
(xx = 06 or xx = 66 or xx = 9A or xx FA) and there is a carry from bit 3

Description

This operation is executed only after an addition or subtraction with valid packed-BCD operands, and is used to adjust the accumulator to produce a valid packed-BCD result.

A specific hex value “xx” is added to the accumulator, and the accumulator is set to the result.

The half-carry flag is cleared if “xx” is 00, 60 or A0, otherwise, the half-carry flag is set if there is a carry from bit-3 and cleared if there is no carry from bit 3.

The carry flag is cleared if “xx” is 00, 06 or FA, otherwise, the carry flag is set.

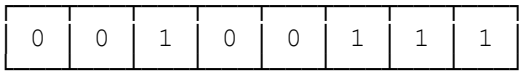
Flags

SF: 1 if A < 0, else 0
ZF: 1 if A = 0, else 0
HF: See operation/description
PF: 1 if parity even, else 0
NF: Not Affected
CF: See operation/description

Clocking

	M-cycles	Tymes
DAA	1	4

Encoding



	op-code
DAA	27

DEC dest decrement destination operand

Operation
dest ← dest – 1

Description
The destination operand is decremented by one.

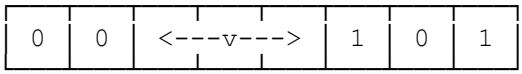
Flags

	If 8-bit Instruction:	If 16-bit Instruction:
SF:	1 if result < 0, else 0	Not Affected
ZF:	1 if result = 0, else 0	Not Affected
HF:	1 if bit-4 borrow, else 0	Not Affected
PF:	1 if underflow, else 0	Not Affected
NF:	1	Not Affected
CF:	Not Affected	Not Affected

Clocking

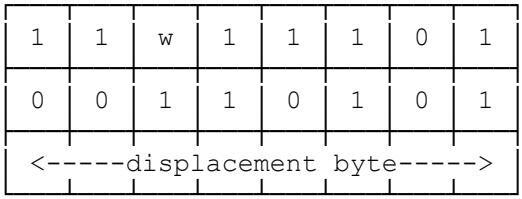
	M-cycles	Tymes
DEC reg	1	4
DEC (HL)	3	11 (4,4,3)
DEC (Ii+d)	6	23 (4,4,3,5,4,3)
DEC regpr	1	6
DEC Ii	2	10 (4,6)

Encoding



op

	v	op-code
DEC B	000	05
DEC C	001	0D
DEC D	010	15
DEC E	011	1D
DEC H	100	25
DEC L	101	2D
DEC (HL)	110	35
DEC A	111	3D

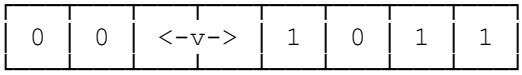


DD or FD

35

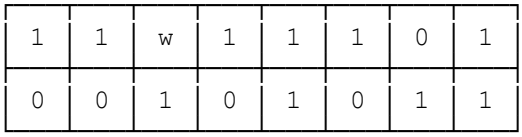
db

	w	op-code
DEC (IX+d)	0	DD 35 db
DEC (IY+d)	1	FD 35 db



op

	v	op-code
DEC BC	00	0B
DEC DE	01	1B
DEC HL	10	2B
DEC SP	11	3B



DD or FD

2B

	w	op-code
DEC IX	0	DD 2B
DEC IY	1	FD 2B

DI disable maskable interrupts

Operation
IFF1 ← 0
IFF2 ← 0

Description
Both interrupt flip-flops are reset.

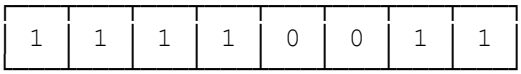
Maskable interrupts occurring during or at the end of this instruction are ignored.

Flags
SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: 1 if result = 0, else 0

Clocking

	M-cycles	Tymes
DI	1	4

Encoding



	op-code
DI	F3

DJNZ disp — decrement and jump on not zero

Note: in most assemblers, this instruction takes the form “DJNZ addr”. The assembler then computes the displacement dynamically as though it were the expression “addr – PC”.

Operation

B ← B – 1

If B[7 ORSUM 0] = 1 then PC ← PC + disp

Description

The B register is decremented.

If the B register does not equal zero:

The program counter is made equal to the sum of the program counter plus the sign-extended address-displacement byte. Execution continues at the address in the program counter.

If the B register equal zero:

Execution continues with the next instruction.

The displacement byte is referenced to the address of the instruction, and has a range of –126 to +129 bytes. Compensation is automatically made for the double program counter increment.

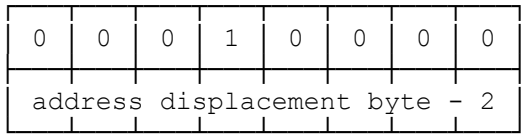
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	If B <> 0:		If B = 0:	
	M-cycles	Tymes	M-cycles	Tymes
DJNZ disp	3	13 (5,3,5)	2	8 (5,3)

Encoding



	op-code
DJNZ disp	10 ab

EI enable maskable interrupts

Operation

IFF1 ← 1
IFF2 ← 1

Description

Both interrupt flip-flops are set.

Maskable interrupts occurring during or at the end of this instruction are ignored.

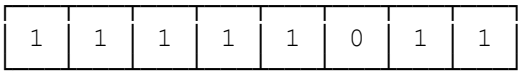
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
EI	1	4

Encoding



	op-code
EI	FB

EX left,right exchange left and right operands

Operation
left ↔ right

Description
The left operand is exchanged with the right operand.

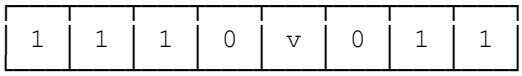
Flags

	If not EX AF,AF'	If EX AF,AF'
SF:	Not Affected	All flags assume the values previously
ZF:	Not Affected	held by the alternative flag register
HF:	Not Affected	
PF:	Not Affected	
NF:	Not Affected	
CF:	Not Affected	

Clocking

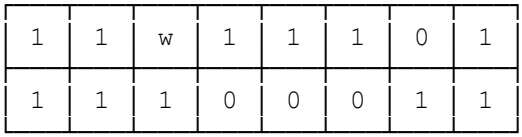
	M-cycles	Tymes
EX DE,HL	1	4
EX (SP),HL	5	19 (4,3,4,3,5)
EX (SP),Ii	6	23 (4,4,3,4,3,5)
EX AF,AF'	1	4

Encoding



op

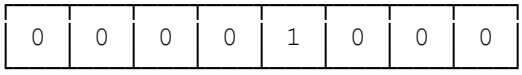
	v	op-code
EX (SP),HL	0	E3
EX DE,HL	1	EB



DD or FD

E3

	w	op-code
EX (SP),IX	0	DD E3
EX (SP),IY	1	FD E3



08

	op-code
EX AF,AF'	08

EXX exchange general register sets

Operation

BC ↔ BC'
DE ↔ DE'
HL ↔ HL'

Description

The primary general registers are exchanged with the alternative general registers.

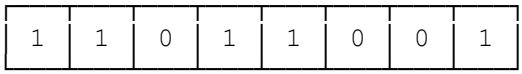
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
EXX	1	4

Encoding



	op-code
EXX	D9

HALT halt operation

Operation
CPU halted.

Description
The CPU halts operation until reset or until an interrupt is received.

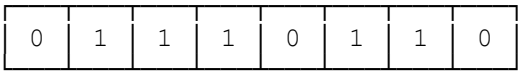
The CPU automatically performs NOP instructions while halted in order to maintain memory refresh logic.

Flags
SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
HALT	1	4

Encoding



	op-code
HALT	76

IM mode set interrupt mode

Operation

If mode = 0:
IMFa ← 0
IMFb ← 0

Upon interrupt:
CPU executes instruction on data bus six times.

If mode = 1:
IMFa ← 1
IMFb ← 0

Upon interrupt:
PC ← 0038

If mode = 2:
IMFa ← 1
IMFb ← 1

Upon interrupt:
PC-low ← data bus
PC-high ← I

Description

The interrupt mode flip-flops “a” and “b” are cleared or set as indicated.

In interrupt mode 0:
The interrupting peripheral places an 8-bit instruction the data bus.
The CPU implements that instruction six times.

The instruction is usually one of the RST instructions, which would then initiate an unconditional jump to the appropriate vector address.

In interrupt mode 1:
When an interrupt is received, the program counter is made equal to 0038.
Execution continues at address 0038.

This operation is identical to executing an RST 38 instruction.

This mode is most often used by a debugger or similar program.

In interrupt mode 2:
The interrupting peripheral places an 8-bit address vector on the data bus.
The low-order byte of the program counter is made equal to the data bus.
The high-order byte of the program counter is made equal to the interrupt vector register (I).
Execution continues at the address in the program counter.

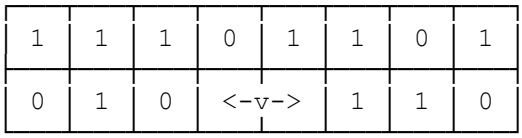
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
IM mode	2	8 (4,4)

Encoding



	v	op-code
IM 0	00	ED 46
IM 1	10	ED 56
IM 2	11	ED 5E

IN A,(port) input port-direct byte

IN reg,(C) input port-indirect byte

Operation

If direct:

address-bus-low ← port
address-bus-high ← A
A ← (address-bus-low)

If indirect:

address-bus-low ← C
address-bus-high ← B
reg ← (address-bus-low)

Description

If direct:

The low-order byte of the address bus is made equal to the immediate port operand value.
The high-order byte of the address bus is made equal to the accumulator
The accumulator is made equal to the byte present at the input port whose address is the low-order byte of the address bus.

If indirect:

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to the B register.
The register operand is made equal to the byte present at the input port whose address is the low-order byte of the address bus.
The B register is left unchanged unless the register operand is the B register.
The C register is left unchanged unless the register operand is the C register.

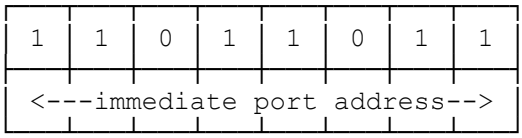
Flags

	If IN A,(port):	If IN reg,(C):
SF:	Not Affected	1 if reg < 0, else 0
ZF:	Not Affected	1 if reg = 0, else 0
HF:	Not Affected	0
PF:	Not Affected	1 if parity even, else 0
NF:	Not Affected	0
CF:	Not Affected	Not Affected

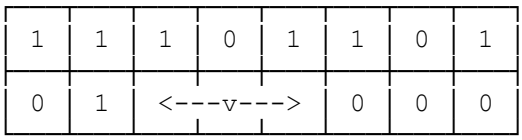
Clocking

	M-cycles	Tymes
IN A,(port)	3	11 (4,3,4)
IN reg,(C)	3	12 (4,4,4)

Encoding



	op-code
IN A,(port)	DB pb



	v	op-code
IN B,(C)	000	ED 40
IN C,(C)	001	ED 48
IN D,(C)	010	ED 50
IN E,(C)	011	ED 58
IN H,(C)	100	ED 60
IN L,(C)	101	ED 68
	110	ED 70
IN A,(C)	111	ED 78

NOTE: The instruction whose op-code is “ED 70” has no mnemonic, and sets the flags in a normal manner without affecting register contents. This instruction may be used to determine the status of an input port byte without actually fetching the byte.

IN0 reg,(port) input port-direct byte

Operation
address-bus-low ← port
address-bus-high ← 00
reg ← (address-bus)

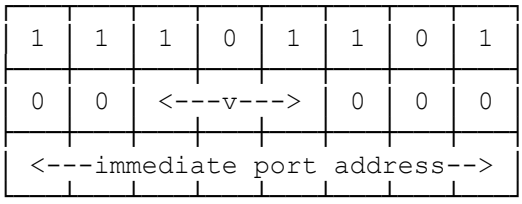
Description
The low-order byte of the address bus is made equal to the immediate port operand value.
The high-order byte of the address bus is made equal to 00.
The register operand is made equal to the byte present at the input port whose address is the low-order byte of the address bus.

Flags
SF: 1 if reg < 0, else 0
ZF: 1 if reg = 0, else 0
HF: 0
PF: 1 if parity even, else 0
NF: 0
CF: Not Affected

Clocking

	M-cycles	Tymes
IN0 reg,(port)	4	12

Encoding



	v	op-code
IN0 B,(port)	000	ED 00 pb
IN0 C,(port)	001	ED 08 pb
IN0 D,(port)	010	ED 10 pb
IN0 E,(port)	011	ED 18 pb
IN0 H,(port)	100	ED 20 pb
IN0 L,(port)	101	ED 28 pb
	110	ED 30 pb
IN0 A,(port)	111	ED 38 pb

NOTE: The instruction whose op-code is “ED 30 pb” has no mnemonic, and sets the flags in a normal manner without affecting register contents. This instruction may be used to determine the status of an input port byte without actually fetching the byte.

INC dest increment destination operand

Operation
dest ← dest + 1

Description
The destination operand is incremented by one.

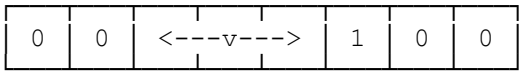
Flags

	If 8-bit Instruction:	If 16-bit Instruction:
SF:	1 if result < 0, else 0	Not Affected
ZF:	1 if result = 0, else 0	Not Affected
HF:	1 if bit-3 carry, else 0	Not Affected
PF:	1 if overflow, else 0	Not Affected
NF:	0	Not Affected
CF:	Not Affected	Not Affected

Clocking

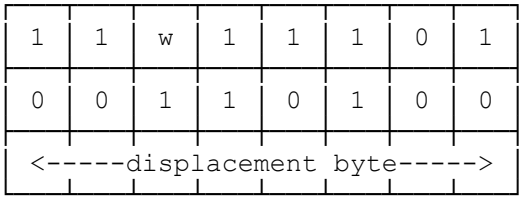
	M-cycles	Tymes
INC reg	1	4
INC (HL)	3	11 (4,4,3)
INC (Ii+d)	6	23 (4,4,3,5,4,3)
INC regpr	1	6
INC Ii	2	10 (4,6)

Encoding



op

	v	op-code
INC B	000	04
INC C	001	0C
INC D	010	14
INC E	011	1C
INC H	100	24
INC L	101	2C
INC (HL)	110	34
INC A	111	3C

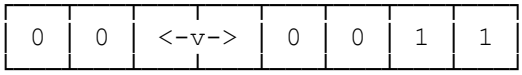


DD or FD

34

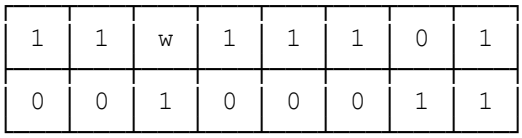
db

	w	op-code
INC (IX+d)	0	DD 34 db
INC (IY+d)	1	FD 34 db



op

	v	op-code
INC BC	00	03
INC DE	01	13
INC HL	10	23
INC SP	11	33



DD or FD

23

	w	op-code
INC IX	0	DD 23
INC IY	1	FD 23

IND input port-indirect byte and decrement

Operation

address-bus-low ← C
address-bus-high ← B
(HL) ← (address-bus-low)

HL ← HL – 1
B ← B – 1

ZF ← B[7 ORSUM 0]

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to the B register.
The memory byte whose address is in the HL register pair is made equal to the byte present at the input port whose address is the low-order byte of the address bus.

The HL register pair is decremented.
The B register is decremented.

If the B register is zero, then the zero flag is set; otherwise the zero flag is cleared.

The C register is left unchanged.

Flags

SF: Undefined
ZF: 1 if B = 0, else 0
HF: Undefined
PF: Undefined
NF: 1
CF: Not Affected

Clocking

	M-cycles	Tymes
IND	4	16 (4,5,3,4)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	0	1	0	1	0	AA

	op-code
IND	ED AA

INDR input port-indirect byte, decrement, and repeat

Operation

address-bus-low ← C
address-bus-high ← B
(HL) ← (address-bus-low)

HL ← HL – 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

If ZF = 0, repeat operation.

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to the B register.
The memory byte whose address is in the HL register pair is made equal to the byte present at the input port whose address is the low-order byte of the address bus.

The HL register pair is decremented.
The B register is decremented.

If the B register is zero, then the zero flag is set; otherwise, the zero flag is cleared.

If the zero flag is not set, the operation is repeated.

The C register is left unchanged.

Flags

SF: Undefined
ZF: 1 after final iteration
HF: Undefined
PF: Undefined
NF: 1
CF: Not Affected

Clocking

	If B <> 0:		If B = 0:	
	M-cycles	Tymes	M-cycles	Tymes
INDR	5	21 (4,5,3,4,5)	4	16 (4,5,3,4)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	1	0	BA

	op-code
INDR	ED BA

INI input port-indirect byte and increment

Operation

address-bus-low ← C
address-bus-high ← B
(HL) ← (address-bus-low)

HL ← HL + 1
B ← B – 1

ZF ← B[7 ORSUM 0]

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to the B register.
The memory byte whose address is in the HL register pair is made equal to the byte present at the input port whose address is the low-order byte of the address bus.

The HL register pair is incremented.
The B register is decremented.

If the B register is zero, then the zero flag is set; otherwise, the zero flag is cleared.

The C register is left unchanged.

Flags

SF: Undefined
ZF: 1 if B = 0, else 0
HF: Undefined
PF: Undefined
NF: 1
CF: Not Affected

Clocking

	M-cycles	Tymes
INI	4	16 (4,5,3,4)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	1	0	A2

	op-code
INI	ED A2

INIR input port-indirect byte, increment, and repeat

Operation

address-bus-low ← C
address-bus-high ← B
(HL) ← (address-bus-low)

HL ← HL + 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

If ZF = 0, repeat operation.

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to the B register.
The memory byte whose address is in the HL register pair is made equal to the byte present at the input port whose address is the low-order byte of the address bus.

The C register is left unchanged.

The HL register pair is incremented.
The B register is decremented.

If the B register is zero, the zero flag is set; otherwise, the zero flag is cleared.

If the zero flag is not set, the operation is repeated.

The C register is left unchanged.

Flags

SF: Undefined
ZF: 1 after final iteration
HF: Undefined
PF: Undefined
NF: 1
CF: Not Affected

Clocking

	If B <> 0:		If B = 0:	
	M-cycles	Tymes	M-cycles	Tymes
INIR	5	21 (4,5,3,4,5)	4	16 (4,5,3,4)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	1	0	B2

	op-code
INIR	ED B2

JP addr	unconditional direct absolute jump
JP (reg16)	unconditional indirect absolute jump
JP cond,addr	conditional absolute jump

Operation

If unconditional and direct:

PC ← addr

If unconditional and indirect:

PC ← (reg16)

If conditional:

NZ: if ZF = 0 then PC ← addr
Z: if ZF = 1 then PC ← addr
NC: if CF = 0 then PC ← addr
C: if CF = 1 then PC ← addr
PO: if PF = 0 then PC ← addr
PE: if PF = 1 then PC ← addr
P: if SF = 0 then PC ← addr
M: if SF = 1 then PC ← addr

Description

If unconditional and direct:

The program counter is made equal to the value of the immediate address operand.
Execution continues at the address in the program counter.

If unconditional and indirect:

The program counter is made equal to the 16-bit register/register pair operand.
Execution continues at the address in the program counter.

If conditional:

A test is made of the indicated flag.
If the condition is met:

The program counter is made equal to the value of the immediate address operand.
Execution continues at the address in the program counter.

If the condition is not met:

Execution continues with the next instruction.

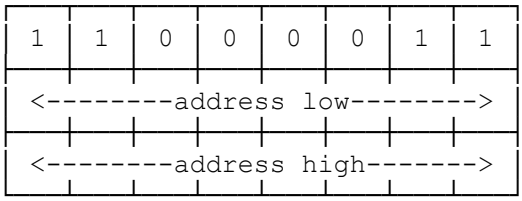
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

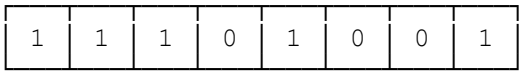
Clocking

	M-cycles	Tymes
JP addr	3	10 (4,3,3)
JP (HL)	1	4
JP (Ii)	2	8 (4,4)
JP cond,addr	3	10 (4,3,3)

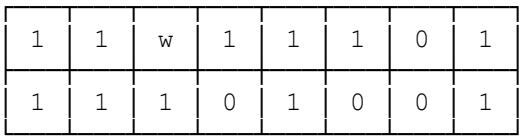
Encoding



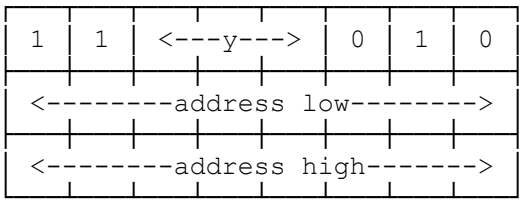
	op-code
JP addr	C3 al ah



	op-code
JP (HL)	E9



	w	op-code
JP (IX)	0	DD E9
JP (IY)	1	FD E9



	y	op-code
JP NZ,addr	000	C2 al ah
JP Z,addr	001	CA al ah
JP NC,addr	010	D2 al ah
JP C,addr	011	DA al ah
JP PO,addr	100	E2 al ah
JP PE,addr	101	EA al ah
JP P,addr	110	F2 al ah
JP M,addr	111	FA al ah

JR disp unconditional relative jump

JR cond,disp conditional relative jump

Note: in most assemblers, this instruction takes the form “JR addr” or “JR cond,addr”. The assembler then computes the displacement dynamically as though it were the expression “addr – PC”.

Operation

If unconditional:

PC ← PC + disp

If conditional:

NZ: if ZF = 0 then PC ← PC + disp
Z: if ZF = 1 then PC ← PC + disp
NC: if CF = 0 then PC ← PC + disp
C: if CF = 1 then PC ← PC + disp

Description

If unconditional:

The program counter is made equal to the sum of the program counter plus the sign-extended address-displacement byte. Execution continues at the address in the program counter.

If conditional:

A test is made of the specified flag.
If the condition is met:

The program counter is made equal to the sum of the program counter plus the sign-extended address-displacement byte. Execution continues at the address in the program counter.

If the condition is not met:

Execution continues with the next instruction.

The address-displacement byte is referenced to the address of the instruction, and has a range of –126 to +129 bytes. Compensation is automatically made for the double program counter increment.

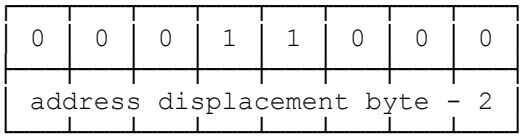
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

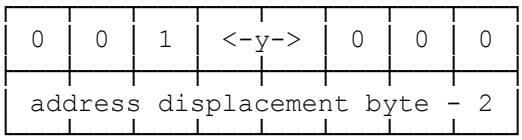
Clocking

	If condition is met:		If condition is not met:	
	M-cycles	Tymes	M-cycles	Tymes
JR disp	3	12 (4,3,5)		
JR cond,disp	3	12 (4,3,5)	2	7 (4,3)

Encoding



	op-code
JR disp	18 ab



	y	op-code
JR NZ,disp	00	20 ab
JR Z,disp	01	28 ab
JR NC,disp	10	30 ab
JR C,disp	11	38 ab

LD dest,source — load destination operand with source operand

Operation
dest ← source

Description
The destination operand is made equal to the source operand.

The source operand is left unchanged.

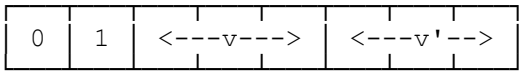
Flags

	If source <> I or R:	If source = I or R:
SF:	Not Affected	1 if source < 0, else 0
ZF:	Not Affected	1 if source = 0, else 0
HF:	Not Affected	0
PF:	Not Affected	Status of IFF2
NF:	Not Affected	0
CF:	Not Affected	Not Affected

Clocking

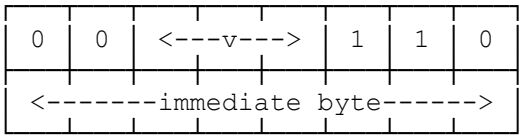
	M-cycles	Tymes
LD reg,reg	1	4
LD reg,(HL)	2	7 (4,3)
LD (HL),reg	2	7 (4,3)
LD reg,immed	2	7 (4,3)
LD (HL),immed	3	10 (4,3,3)
LD reg,(Ii+d)	5	19 (4,4,3,5,3)
LD (Ii+d),reg	5	19 (4,4,3,5,3)
LD (Ii+d),immed	5	19 (4,4,3,5,3)
LD (regpr),A	2	7 (4,3)
LD A,(regpr)	2	7 (4,3)
LD (addr),A	4	13 (4,3,3,3)
LD A,(addr)	4	13 (4,3,3,3)
LD I,A	2	9 (4,5)
LD R,A	2	9 (4,5)
LD A,I	2	9 (4,5)
LD A,R	2	9 (4,5)
LD regpr,imwrd	3	10 (4,3,3)
LD Ii,imwrd	4	14 (4,4,3,3)
LD (addr),HL	5	16 (4,3,3,3,3)
LD HL,(addr)	5	16 (4,3,3,3,3)
LD (addr),regpr	6	20 (4,4,3,3,3,3)
LD regpr,(addr)	6	20 (4,4,3,3,3,3)
LD (addr),Ii	6	20 (4,4,3,3,3,3)
LD Ii,(addr)	6	20 (4,4,3,3,3,3)
LD SP,HL	1	6
LD SP,Ii	2	10 (4,6)

Encoding

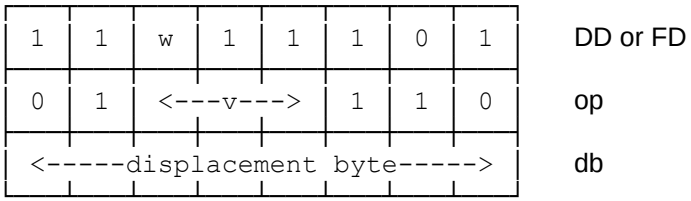


	v	v'	op-code
LD B,B	000	000	40
LD B,C	000	001	41
LD B,D	000	010	42
LD B,E	000	011	43
LD B,H	000	100	44
LD B,L	000	101	45
LD B,(HL)	000	110	46
LD B,A	000	111	47
LD C,B	001	000	48
LD C,C	001	001	49
LD C,D	001	010	4A
LD C,E	001	011	4B
LD C,H	001	100	4C
LD C,L	001	101	4D
LD C,(HL)	001	110	4E
LD C,A	001	111	4F
LD D,B	010	000	50
LD D,C	010	001	51
LD D,D	010	010	52
LD D,E	010	011	53
LD D,H	010	100	54
LD D,L	010	101	55
LD D,(HL)	010	110	56
LD D,A	010	111	57
LD E,B	011	000	58
LD E,C	011	001	59
LD E,D	011	010	5A
LD E,E	011	011	5B
LD E,H	011	100	5C
LD E,L	011	101	5D
LD E,(HL)	011	110	5E
LD E,A	011	111	5F
LD H,B	100	000	60
LD H,C	100	001	61
LD H,D	100	010	62
LD H,E	100	011	63
LD H,H	100	100	64
LD H,L	100	101	65
LD H,(HL)	100	110	66
LD H,A	100	111	67
LD L,B	101	000	68
LD L,C	101	001	69
LD L,D	101	010	6A
LD L,E	101	011	6B
LD L,H	101	100	6C
LD L,L	101	101	6D
LD L,(HL)	101	110	6E
LD L,A	101	111	6F

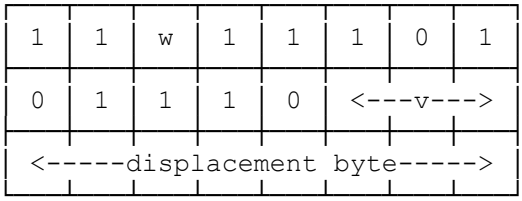
LD (HL),B	110	000	70
LD (HL),C	110	001	71
LD (HL),D	110	010	72
LD (HL),E	110	011	73
LD (HL),H	110	100	74
LD (HL),L	110	101	75
LD (HL),A	110	111	77
LD A,B	111	000	78
LD A,C	111	001	79
LD A,D	111	010	7A
LD A,E	111	011	7B
LD A,H	111	100	7C
LD A,L	111	101	7D
LD A,(HL)	111	110	7E
LD A,A	111	111	7F



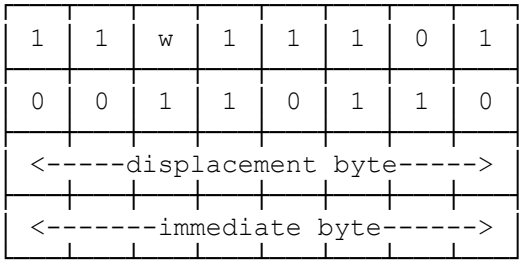
	v	op-code
LD B,immed	000	06 ib
LD C,immed	001	0E ib
LD D,immed	010	16 ib
LD E,immed	011	1E ib
LD H,immed	100	26 ib
LD L,immed	101	2E ib
LD (HL),immed	110	36 ib
LD A,immed	111	3E ib



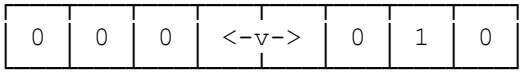
	w	v	op-code
LD B,(IX+d)	0	000	DD 46 db
LD C,(IX+d)	0	001	DD 4E db
LD D,(IX+d)	0	010	DD 56 db
LD E,(IX+d)	0	011	DD 5E db
LD H,(IX+d)	0	100	DD 66 db
LD L,(IX+d)	0	101	DD 6E db
LD A,(IX+d)	0	111	DD 7E db
LD B,(IY+d)	1	000	FD 46 db
LD C,(IY+d)	1	001	FD 4E db
LD D,(IY+d)	1	010	FD 56 db
LD E,(IY+d)	1	011	FD 5E db
LD H,(IY+d)	1	100	FD 66 db
LD L,(IY+d)	1	101	FD 6E db
LD A,(IY+d)	1	111	FD 7E db



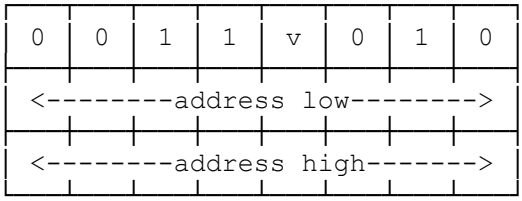
	w	v	op-code
LD (IX+d),B	0	000	DD 70 db
LD (IX+d),C	0	001	DD 71 db
LD (IX+d),D	0	010	DD 72 db
LD (IX+d),E	0	011	DD 73 db
LD (IX+d),H	0	100	DD 74 db
LD (IX+d),L	0	101	DD 75 db
LD (IX+d),A	0	111	DD 77 db
LD (IY+d),B	1	000	FD 70 db
LD (IY+d),C	1	001	FD 71 db
LD (IY+d),D	1	010	FD 72 db
LD (IY+d),E	1	011	FD 73 db
LD (IY+d),H	1	100	FD 74 db
LD (IY+d),L	1	101	FD 75 db
LD (IY+d),A	1	111	FD 77 db



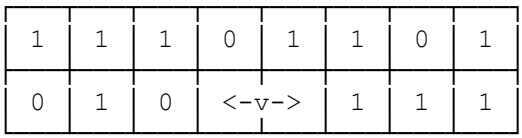
	w	op-code
LD (IX+d),immed	0	DD 36 db ib
LD (IY+d),immed	1	FD 36 db ib



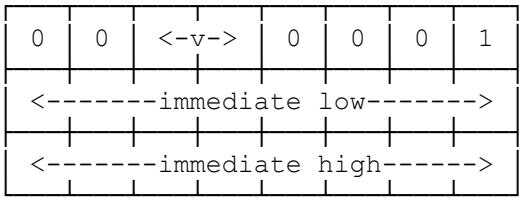
	v	op-code
LD (BC),A	00	02
LD A,(BC)	01	0A
LD (DE),A	10	12
LD A,(DE)	11	1A



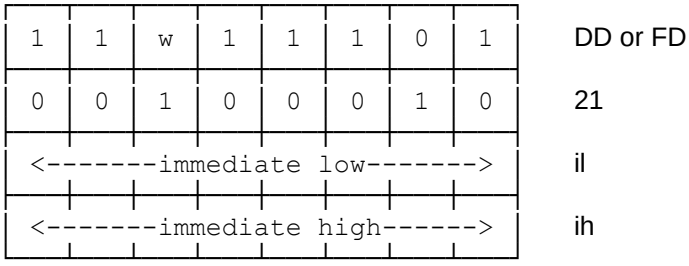
	v	op-code
LD (addr),A	0	32 al ah
LD A,(addr)	1	3A al ah



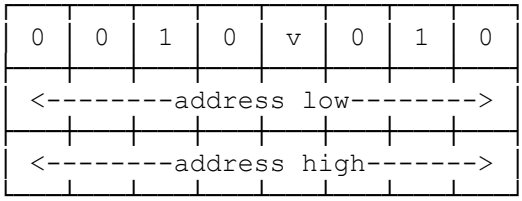
	v	op-code
LD I,A	00	ED 47
LD R,A	01	ED 4F
LD A,I	10	ED 57
LD A,R	11	ED 5F



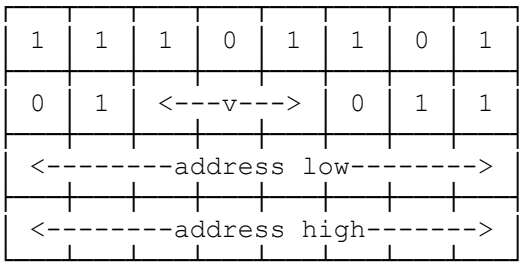
	v	op-code
LD BC,imwrd	00	01 il ih
LD DE,imwrd	01	11 il ih
LD HL,imwrd	10	21 il ih
LD SP,imwrd	11	31 il ih



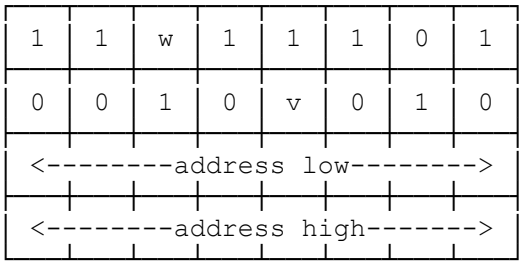
	w	op-code
LD IX,imwrd	0	DD 21 il ih
LD IY,imwrd	1	FD 21 il ih



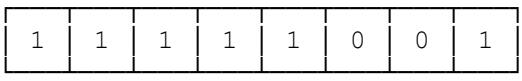
	v	op-code
LD (addr),HL	0	22 al ah
LD HL,(addr)	1	2A al ah



	v	op-code
LD (addr),BC	000	ED 43 al ah
LD BC,(addr)	001	ED 4B al ah
LD (addr),DE	010	ED 53 al ah
LD DE,(addr)	011	ED 5B al ah
LD (addr),HL	100	ED 63 al ah
LD HL,(addr)	101	ED 6B al ah
LD (addr),SP	110	ED 73 al ah
LD SP,(addr)	111	ED 7B al ah



	w	v	op-code
LD (addr),IX	0	0	DD 22 al ah
LD IX,(addr)	0	1	DD 2A al ah
LD (addr),IY	1	0	FD 22 al ah
LD IY,(addr)	1	1	FD 2A al ah



	op-code
LD SP,HL	F9

1	1	w	1	1	1	0	1	DD or FD
1	1	1	1	1	0	0	1	F9

	w	op-code
LD SP,IX	0	DD F9
LD SP,IY	1	FD F9

LDD load (DE) with (HL) and decrement

Operation

(DE) ← (HL)

HL ← HL – 1
DE ← DE – 1
BC ← BC – 1

PF ← BC[15 ORSUM 0]

Description

The memory byte whose address is in the DE register pair is made equal to the memory byte whose address is in the HL register pair.

The HL register pair is decremented.
The DE register pair is decremented.
The BC register pair is decremented.

If the BC register pair is zero, the parity/overflow flag is cleared; otherwise, the parity/overflow flag is set.

Flags

SF: Not Affected
ZF: Not Affected
HF: 0
PF: 1 if BC <> 0, else 0
NF: 0
CF: Not Affected

Clocking

	M-cycles	Tymes
LDD	4	16 (4,4,3,5)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	0	1	0	0	0	A8

	op-code
LDD	ED A8

LDDR load (DE) with (HL), decrement, and repeat until (BC) is 0

Operation

(DE) ← (HL)

HL ← HL – 1
DE ← DE – 1
BC ← BC – 1

PF ← BC[15 ORSUM 0]

If PF = 1, repeat operation.

Description

The memory byte whose address is in the DE register pair is made equal to the memory byte whose address is in the HL register pair.

The HL register pair is decremented.
The DE register pair is decremented.
The BC register pair is decremented.

If the BC register pair is zero, the parity/overflow flag is cleared; otherwise, the parity/overflow flag is set.

If the parity/overflow flag is set, the operation is repeated.

Flags

SF: Not Affected
ZF: Not Affected
HF: 0
PF: 0 after final iteration
NF: 0
CF: Not Affected

Clocking

	While PF = 1:		When PF=0:	
	M-cycles	Tymes	M-cycles	Tymes
LDDR	5	21 (4,4,3,5,5)	4	16 (4,4,3,5)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	1	1	0	0	0	B8

	op-code
LDDR	ED B8

LDI load (DE) with (HL) and increment

Operation

(DE) ← (HL)

HL ← HL + 1
DE ← DE + 1
BC ← BC – 1

PF ← BC[15 ORSUM 0]

Description

The memory byte whose address is in the DE register pair is made equal to the memory byte whose address is in the HL register pair.

The HL register pair is incremented.
The DE register pair is incremented.
The BC register pair is decremented.

If the BC register pair is zero, the parity/overflow flag is cleared; otherwise, the parity/overflow flag is set.

Flags

SF: Not Affected
ZF: Not Affected
HF: 0
PF: 1 if BC <> 0, else 0
NF: 0
CF: Not Affected

Clocking

	M-cycles	Tymes
LDI	4	16 (4,4,3,5)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	0	0	A0

	op-code
LDI	ED A0

LDIR load (DE) with (HL), increment, and repeat until (BC) is 0

Operation

(DE) ← (HL)

HL ← HL + 1
DE ← DE + 1
BC ← BC – 1

PF ← BC[15 ORSUM 0]

If PF = 1, repeat operation.

Description

The memory byte whose address is in the DE register pair is made equal to the memory byte whose address is in the HL register pair.

The HL register pair is incremented.
The DE register pair is incremented.
The BC register pair is decremented.

If the BC register pair is zero, the parity/overflow flag is cleared; , the parity/overflow flag is set.

If the parity/overflow flag is set, the operation is repeated.

Flags

SF: Not Affected
ZF: Not Affected
HF: 0
PF: 0 after final iteration
NF: 0
CF: Not Affected

Clocking

	While PF = 1:		When PF = 0:	
	M-cycles	Tymes	M-cycles	Tymes
LDIR	5	21 (4,4,3,5,5)	4	16 (4,4,3,5)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	0	0	B0

	op-code
LDIR	ED B0

MLT regpr multiply bytes of 16-bit register pair

Operation
regpr ← regpr-low * regpr-high

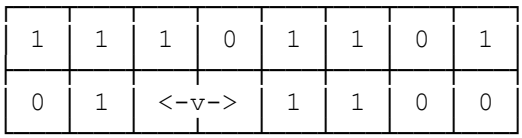
Description
The 16-bit register pair operand is made equal to the product of the low-order byte of the register pair operand times the high-order byte of the register pair operand.

Flags
SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
MLT regpr	13	17

Encoding



	v	op-code
MLT BC	00	ED 4C
MLT DE	01	ED 5C
MLT HL	10	ED 6C
MLT SP	11	ED 7C

NEG negate (two's compliment) accumulator

Operation

temp ← A
A ← 00
A ← A – temp

Description

The accumulator is made equal to the difference of zero minus the accumulator.

Flags

SF: 1 if A < 0, else 0
ZF: 1 if A = 0, else 1
HF: 1 if bit-4 borrow, else 0
PF: 1 if underflow, else 0
NF: 1
CF: 1 if A <> 0 before operation, else 0

Clocking

	M-cycles	Tymes
NEG	2	8 (4,4)

Encoding

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	0	0	44

	op-code
NEG	ED 44

NOP no operation

Operation
None

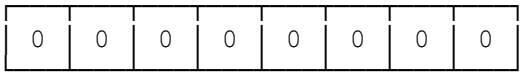
Description
The CPU performs no operation during this machine cycle.

Flags
SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
NOP	1	4

Encoding



	op-code
NOP	00

OR source inclusive-OR source operand with accumulator

Operation

A[0] ← A[0] OR source[0]
A[1] ← A[1] OR source[1]
A[2] ← A[2] OR source[2]
A[3] ← A[3] OR source[3]
A[4] ← A[4] OR source[4]
A[5] ← A[5] OR source[5]
A[6] ← A[6] OR source[6]
A[7] ← A[7] OR source[7]

Description

The accumulator is made equal to the accumulator bitwise inclusive-ORed with the source operand.

The source operand is left unchanged.

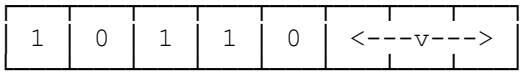
Flags

SF: 1 if A < 0, else 0
ZF: 1 if A = 0, else 0
HF: 1
PF: 1 if parity even, else 0
NF: 0
CF: 0

Clocking

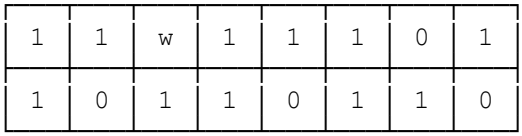
	M-cycles	Tymes
OR reg	1	4
OR (HL)	2	7 (4,3)
OR (Ii+d)	5	19 (4,4,3,5,3)
OR immed	2	7 (4,3)

Encoding



op

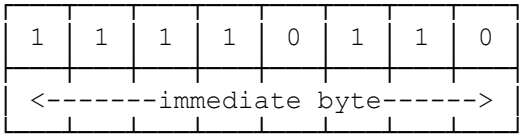
	v	op-code
OR B	000	B0
OR C	001	B1
OR D	010	B2
OR E	011	B3
OR H	100	B4
OR L	101	B5
OR (HL)	110	B6
OR A	111	B7



DD or FD

B6

	w	op-code
OR (IX+d)	0	DD B6 db
OR (IY+d)	1	FD B6 db



F6

ib

	op-code
OR immed	F6 ib

OTDM output port-indirect memory and decrement

Operation

address-bus-low ← C
address-bus-high ← 00
(address-bus) ← (HL)

HL ← HL – 1
C ← C – 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to 00.
The memory byte whose address is in the HL register pair is sent to the output port whose address is the low-order byte of the address bus.

The HL register pair is decremented.
The C register is decremented.
The B register is decremented.

If the B register is zero, the zero flag is set; otherwise, the zero flag is cleared.

Flags

SF: 1 if B < 0, else 0
ZF: 1 if B = 0, else 0
HF: 1 if B bit-4 borrow, else 0
PF: 1 if B parity even, else 0
NF: 1 if (HL) < 0, else 0
CF: 1 if B bit-8 borrow, else 0

Clocking

	M-cycles	Tymes
OTDM	6	14

Encoding

1	1	1	0	1	1	0	1	ED
1	0	0	0	1	0	1	1	8B

	op-code
OTDM	ED 8B

OTDMR output port-indirect memory, decrement, and repeat

Operation

address-bus-low ← C
address-bus-high ← 00
(address-bus) ← (HL)

HL ← HL – 1
C ← C – 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

If ZF = 0, repeat operation.

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to 00.
The memory byte whose address is in the HL register pair is sent to the output port whose address is the low-order byte of the address bus.

The HL register pair is decremented.
The C register is decremented.
The B register is decremented.

If the B register is zero, the zero flag is set; otherwise, the zero flag is cleared.

If the zero flag is not set, the operation is repeated.

Flags

SF: 0 after final iteration
ZF: 1 after final iteration
HF: 0 after final iteration
PF: 1 after final iteration
NF: 1 if (HL) < 0, else 0
CF: 0 after final iteration

Clocking

	If B <> 00:		If B = 00:	
	M-cycles	Tymes	M-cycles	Tymes
OTDMR	8	16	6	14

Encoding

1	1	1	0	1	1	0	1	ED
1	0	0	1	1	0	1	1	9B

	op-code
OTDMR	ED 9B

OTDR output port-indirect byte, decrement, and repeat

Operation

address-bus-low ← C
address-bus-high ← B
(address-bus-low) ← (HL)

HL ← HL – 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

If ZF = 0, repeat operation.

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to the B register.
The memory byte whose address is in the HL register pair is sent to the output port whose address is the low-order byte of the address bus.

The HL register pair is decremented.
The B register is decremented.

If the B register is zero, the zero flag is set; otherwise, the zero flag is cleared.

If the zero flag is set, the operation is repeated.

The C register is left unchanged.

Flags

SF: Undefined
ZF: 1 after final iteration
HF: Undefined
PF: Undefined
NF: 1
CF: Undefined

Clocking

	If B <> 0:		If B = 0:	
	M-cycles	Tymes	M-cycles	Tymes
OTDR	5	21 (4,5,3,4,5)	4	16 (4,5,3,4)

Encoding

	1	1	1	0	1	1	0	1	
	1	0	1	1	1	0	1	1	

ED
BB

	op-code
OTDR	ED BB

OTIM output port-indirect memory and increment

Operation

address-bus-low ← C
address-bus-high ← 00
(address-bus) ← (HL)

HL ← HL + 1
C ← C + 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to 00.
The memory byte whose address is in the HL register pair is sent to the output port whose address is the low-order byte of the address bus.

The HL register pair is incremented.
The C register is incremented.
The B register is decremented.

If the B register is zero, the zero flag is set; otherwise, the zero flag is cleared.

Flags

SF: 1 if B < 0, else 0
ZF: 1 if B = 0, else 0
HF: 1 if B bit-4 borrow, else 0
PF: 1 if B parity even, else 0
NF: 1 if (HL) < 0, else 0
CF: 1 if B bit-8 borrow, else 0

Clocking

	M-cycles	Tymes
OTIM	6	14

Encoding

1	1	1	0	1	1	0	1	ED
1	0	0	0	0	0	1	1	83

	op-code
OTIM	ED 83

OTIMR output port-indirect memory, increment, and repeat

Operation

address-bus-low ← C
address-bus-high ← 00
(address-bus) ← (HL)

HL ← HL + 1
C ← C + 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

If ZF = 0, repeat operation.

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to 00.
The memory byte whose address is in the HL register pair is sent to the output port whose address is the low-order byte of the address bus.

The HL register pair is incremented.
The C register is incremented.
The B register is decremented.

If the B register is zero, the zero flag is set; otherwise, the zero flag is cleared.

If the zero flag is not set, the operation is repeated.

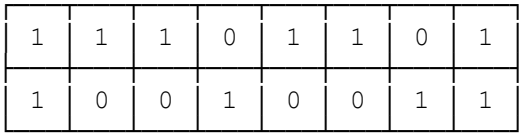
Flags

SF: 0 after final iteration
ZF: 1 after final iteration
HF: 0 after final iteration
PF: 1 after final iteration
NF: 1 if (HL) < 0, else 0
CF: 0 after final iteration

Clocking

	If B <> 00:		If B = 00:	
	M-cycles	Tymes	M-cycles	Tymes
OTIMR	8	16	6	14

Encoding



	op-code
OTIMR	ED 93

OTIR output port-indirect byte, increment, and repeat

Operation

address-bus-low ← C
address-bus-high ← B
(address-bus-low) ← (HL)

HL ← HL + 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

If ZF = 0, repeat instruction.

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to the B register.
The memory byte whose address is in the HL register pair is sent to the output port whose address is the low-order byte of the address bus.

The HL register pair is incremented.
The B register is decremented.

If the B register is zero, the zero flag is set, otherwise, the zero flag is cleared.

If the zero flag is set, the operation is repeated.

The C register is left unchanged.

Flags

SF: Undefined
ZF: 1 after final iteration
HF: Undefined
PF: Undefined
NF: 1
CF: Undefined

Clocking

	If B <> 0:		If B = 0:	
	M-cycles	Tymes	M-cycles	Tymes
OTIR	5	21 (4,5,3,4,5)	4	16 (4,5,3,4)

Encoding

	7	6	5	4	3	2	1	0	
	1	1	1	0	1	1	0	1	ED
	1	0	1	1	0	0	1	1	B3

	op-code
OTIR	ED B3

OUT (port),A output port-direct byte

OUT (C),reg output port-indirect byte

Operation

If direct:

address-bus-low ← port
address-bus-high ← A
(address-bus-low) ← A

If indirect:

address-bus-low ← C
address-bus-high ← B
(address-bus-low) ← reg

Description

If direct:

The low-order byte of the address bus is made equal to the immediate port operand value.
The high-order byte of the address bus is made equal to the accumulator.
The output port whose address is the low-order byte of the address bus is made equal to the accumulator.

The accumulator is left unchanged

If indirect:

The low-order byte of the address bus is made equal to the C register:
The high-order byte of the address bus is made equal to the B register.
The output port whose address is the low-order byte of the address bus is made equal to the register operand.

The register operand is left unchanged.
The B register is left unchanged.
The C register is left unchanged.

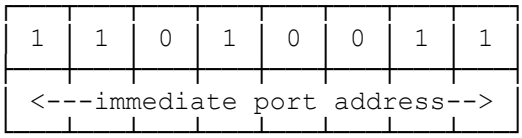
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Undefined
CF: Not Affected

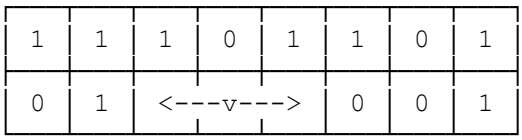
Clocking

	M-cycles	Tymes
OUT (port),A	3	11 (4,3,4)
OUT (C),reg	3	12 (4,4,4)

Encoding



	op-code
OUT (port),A	D3 pb



	v	op-code
OUT (C),B	000	ED 41
OUT (C),C	001	ED 49
OUT (C),D	010	ED 51
OUT (C),E	011	ED 59
OUT (C),H	100	ED 61
OUT (C),L	101	ED 69
OUT (C),A	111	ED 79

OUT0 (port),reg output port-direct byte

Operation

address-bus-low ← port

address-bus-high ← 00

(address-bus) ← reg

Description

The low-order byte of the address bus is made equal to the immediate port operand.
The high-order byte of the address bus is made equal to 00.
The output port whose address is the low-order byte of the address bus is made equal to the register operand.

The register operand is left unchanged.

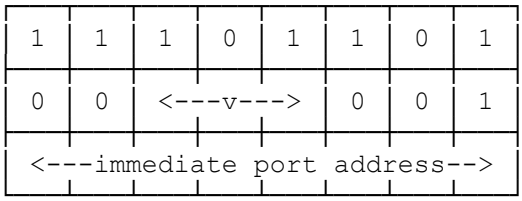
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
OUT0 (port),reg	5	13

Encoding



	v	op-code
OUT0 (port),B	000	ED 01 pb
OUT0 (port),C	001	ED 09 pb
OUT0 (port),D	010	ED 11 pb
OUT0 (port),E	011	ED 19 pb
OUT0 (port),H	100	ED 21 pb
OUT0 (port),L	101	ED 29 pb
OUT0 (port),A	111	ED 39 pb

OUTD output port-indirect byte and decrement

Operation

address-bus-low ← C
address-bus-high ← B
(address-bus-low) ← (HL)

HL ← HL – 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to the B register.
The memory byte whose address is in the HL register pair is sent to the output port whose address is the low-order byte of the address bus.

The HL register pair is decremented.
The B register is decremented.

If the B register is zero, the zero flag is set; otherwise, the zero flag is cleared.

The C register is left unchanged.

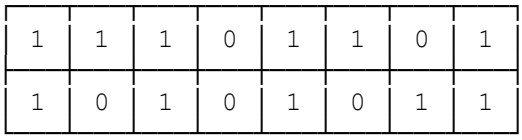
Flags

SF: Undefined
ZF: 1 if B = 0, else 0
HF: Undefined
PF: Undefined
NF: 1
CF: Not Affected

Clocking

	M-cycles	Tymes
OUTD	4	16 (4,5,3,4)

Encoding



	op-code
OUTD	ED AB

OUTI output port-indirect byte and increment

Operation

address-bus-low ← C
address-bus-high ← B
(address-bus-low) ← (HL)

HL ← HL + 1
B ← B – 1

ZF ← NOT B[7 ORSUM 0]

Description

The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to the B register.
The memory byte whose address is in the HL register pair is sent to the output port whose address is the low-order byte of the address bus.

The HL register pair is incremented.
The B register is decremented.

If the B register is zero, the zero flag is set; otherwise, the zero flag is cleared.

The C register is left unchanged.

Flags

SF: Undefined
ZF: 1 if B = 0, else 0
HF: Undefined
PF: Undefined
NF: 1
CF: Not Affected

Clocking

	M-cycles	Tymes
OUTI	4	16 (4,5,3,4)

Encoding

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	1	1	A3

	op-code
OUTI	ED A3

POP reg16 pop 16-bit register/register pair from stack

Operation

reg16-low ← (SP)
SP ← SP + 1
reg16-high ← (SP)
SP ← SP + 1

Description

The low-order byte of the 16-bit register/register pair operand is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is incremented.
The high-order byte of the 16-bit register/register pair operand is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is again incremented.

The memory bytes are left unchanged.

Flags

	If wrdreg <> AF:	If wrdreg = AF:
SF:	Not Affected	Bit 7 of (SP – 2)
ZF:	Not Affected	Bit 6 of (SP – 2)
HF:	Not Affected	Bit 4 of (SP – 2)
PF:	Not Affected	Bit 2 of (SP – 2)
NF:	Not Affected	Bit 1 of (SP – 2)
CF:	Not Affected	Bit 0 of (SP – 2)

Clocking

	M-cycles	Tymes
POP regpr	3	10 (4,3,3)
POP Ii	4	14 (4,4,3,3)

Encoding

1	1	<-v->	0	0	0	1	op

	v	op-code
POP BC	00	C1
POP DE	01	D1
POP HL	10	E1
POP AF	11	F1

1	1	w	1	1	1	0	1	DD or FD
1	1	1	0	0	0	0	1	E1

	w	op-code
POP IX	0	DD E1
POP IY	1	FD E1

PUSH reg16 push 16-bit register/register pair onto stack

Operation

SP ← SP – 1
(SP) ← reg16-high
SP ← SP – 1
(SP) ← reg16-low

Description

The stack pointer is decremented.
The memory byte whose address is in the stack pointer is made equal to the high-order byte of the 16-bit register/register pair operand.
The stack pointer is again decremented.
The memory byte whose address is in the stack pointer is made equal to the value of the low-order byte of the 16-bit register/register pair operand.

The 16-bit register/register pair operand is left unchanged.

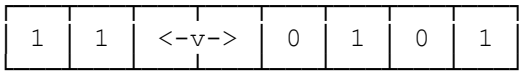
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

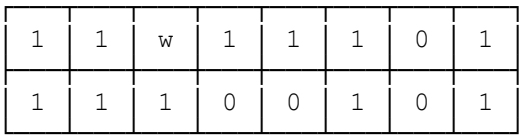
	M-cycles	Tymes
PUSH regpr	3	11 (5,3,3)
PUSH Ii	4	15 (4,5,3,3)

Encoding



op

	v	op-code
PUSH BC	00	C5
PUSH DE	01	D5
PUSH HL	10	E5
PUSH AF	11	F5



DD or FD

E5

	w	op-code
PUSH IX	0	DD E5
PUSH IY	1	FD E5

RES bit,source reset source operand bit

Operation
source[bit] ← 0

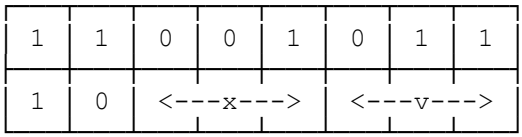
Description
The specified bit of the source operand is cleared.

Flags
SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

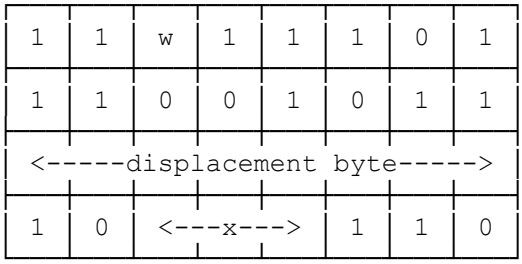
	M-cycles	Tymes
RES bit,reg	2	8 (4,4)
RES bit,(HL)	3	12 (4,4,4)
RES bit,(Ii+d)	5	20 (4,4,3,5,4)

Encoding



	x	v	op-code
RES 0,B	000	000	CB 80
RES 0,C	000	001	CB 81
RES 0,D	000	010	CB 82
RES 0,E	000	011	CB 83
RES 0,H	000	100	CB 84
RES 0,L	000	101	CB 85
RES 0,(HL)	000	110	CB 86
RES 0,A	000	111	CB 87
RES 1,B	001	000	CB 88
RES 1,C	001	001	CB 89
RES 1,D	001	010	CB 8A
RES 1,E	001	011	CB 8B
RES 1,H	001	100	CB 8C
RES 1,L	001	101	CB 8D
RES 1,(HL)	001	110	CB 8E
RES 1,A	001	111	CB 8F
RES 2,B	010	000	CB 90
RES 2,C	010	001	CB 91
RES 2,D	010	010	CB 92
RES 2,E	010	011	CB 93
RES 2,H	010	100	CB 94
RES 2,L	010	101	CB 95
RES 2,(HL)	010	110	CB 96
RES 2,A	010	111	CB 97

RES 3,B	011	000	CB 98
RES 3,C	011	001	CB 99
RES 3,D	011	010	CB 9A
RES 3,E	011	011	CB 9B
RES 3,H	011	100	CB 9C
RES 3,L	011	101	CB 9D
RES 3,(HL)	011	110	CB 9E
RES 3,A	011	111	CB 8F
RES 4,B	100	000	CB A0
RES 4,C	100	001	CB A1
RES 4,D	100	010	CB A2
RES 4,E	100	011	CB A3
RES 4,H	100	100	CB A4
RES 4,L	100	101	CB A5
RES 4,(HL)	100	101	CB A6
RES 4,A	100	111	CB A7
RES 5,B	101	000	CB A8
RES 5,C	101	001	CB A9
RES 5,D	101	010	CB AA
RES 5,E	101	011	CB AB
RES 5,H	101	100	CB AC
RES 5,L	101	101	CB AD
RES 5,(HL)	101	110	CB AE
RES 5,A	101	111	CB AF
RES 6,B	110	000	CB B0
RES 6,C	110	001	CB B1
RES 6,D	110	010	CB B2
RES 6,E	110	011	CB B3
RES 6,H	110	100	CB B4
RES 6,L	110	101	CB B5
RES 6,(HL)	110	110	CB B6
RES 6,A	110	111	CB B7
RES 7,B	111	000	CB B8
RES 7,C	111	001	CB B9
RES 7,D	111	010	CB BA
RES 7,E	111	011	CB BB
RES 7,H	111	100	CB BC
RES 7,L	111	101	CB BD
RES 7,(HL)	111	110	CB BE
RES 7,A	111	111	CB BF



	w	x	op-code
RES 0,(IX+d)	0	000	DD CB db 86
RES 1,(IX+d)	0	001	DD CB db 8E
RES 2,(IX+d)	0	010	DD CB db 96
RES 3,(IX+d)	0	011	DD CB db 9E
RES 4,(IX+d)	0	100	DD CB db A6
RES 5,(IX+d)	0	101	DD CB db AE
RES 6,(IX+d)	0	110	DD CB db B6
RES 7,(IX+d)	0	111	DD CB db BE
RES 0,(IY+d)	1	000	FD CB db 86
RES 1,(IY+d)	1	001	FD CB db 8E
RES 2,(IY+d)	1	010	FD CB db 96
RES 3,(IY+d)	1	011	FD CB db 9E
RES 4,(IY+d)	1	100	FD CB db A6
RES 5,(IY+d)	1	101	FD CB db AE
RES 6,(IY+d)	1	110	FD CB db B6
RES 7,(IY+d)	1	111	FD CB db BE

RET	unconditional return
RET cond	conditional return.

Operation

If unconditional:

do return

If conditional:

NZ: if ZF = 0 then do return.
Z: if ZF = 1 then do return.
NC: if CF = 0 then do return.
C: if CF = 1 then do return.
PO: if PF = 0 then do return.
PE: if PF = 1 then do return.
P: if SF = 0 then do return.
M: if SF = 1 then do return.

If do return:

PC-low ← (SP)
SP ← SP + 1
PC-high ← (SP)
SP ← SP + 1

Description

If unconditional:

The low-order byte of the program counter is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is incremented.
The high-order byte of the program counter is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is again incremented.
Execution continues at the address that is in the program counter.

If conditional:

The appropriate flag is tested.
If the condition is met:

The low-order byte of the program counter is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is incremented.
The high-order byte of the program counter is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is again incremented.
Execution continues at the address that is in the program counter.

If the condition is not met:

Execution continues with the next instruction.

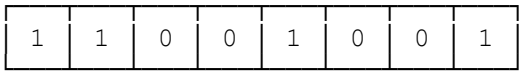
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

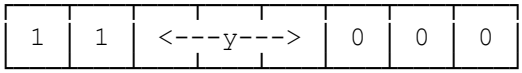
	If condition is met:		If condition is not met:	
	M-cycles	Tymes	M-cycles	Tymes
RET	3	10 (4,3,3)		
RET cond	3	11 (5,3,3)	1	5

Encoding



C9

	op-code
RET	C9



op

	y	op-code
RET NZ	000	C0
RET Z	001	C8
RET NC	010	D0
RET C	011	D8
RET PO	100	E0
RET PE	101	E8
RET P	110	F0
RET M	111	F8

RETI return from maskable interrupt

Operation

PC-low ← (SP)
SP ← SP + 1
PC-high ← (SP)
SP ← SP + 1

interrupting device ← end of interrupt signal

IFF1 ← 0
IFF2 ← 0

Description

The low-order byte of the program counter is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is incremented.
The high-order byte of the program counter is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is again incremented.

The interrupting device is signaled with an end of interrupt signal.

Both interrupt flip-flops are cleared.

Execution continues at the address that is in the program counter.

Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
RETI	4	14 (4,4,3,3)

Encoding

1	1	1	0	1	1	0	1	ED
0	1	0	0	1	1	0	1	4D

	op-code
RETI	ED 4D

RETN return from non-maskable interrupt

Operation

PC-low ← (SP)
SP ← SP + 1
PC-high ← (SP)
SP ← SP + 1

IFF1 ← IFF2

Description

The low-order byte of the program counter is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is incremented.
The high-order byte of the program counter is made equal to the memory byte whose address is in the stack pointer.
The stack pointer is again incremented.

Interrupt flip-flop #1 is made equal to interrupt flip-flop #2.

Execution continues at the address that is in the program counter.

Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
RETN	4	14 (4,4,3,3)

Encoding

1	1	1	0	1	1	0	1	ED
0	1	0	0	0	1	0	1	45

	op-code
RETN	ED 45

RL dest rotate destination operand left

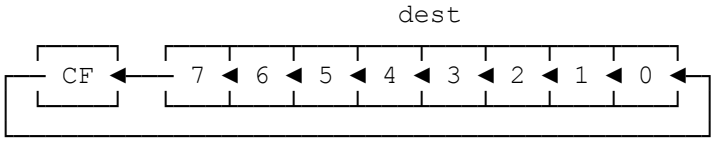
Operation

temp ← dest
dest[0] ← CF
dest[1] ← temp[0]
dest[2] ← temp[1]
dest[3] ← temp[2]
dest[4] ← temp[3]
dest[5] ← temp[4]
dest[6] ← temp[5]
dest[7] ← temp[6]
CF ← temp[7]

Description

The bits of the destination operand plus carry are rotated left.

Graphically, this is:



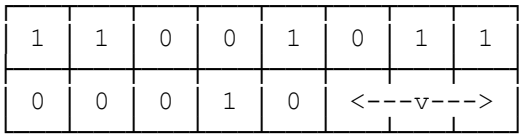
Flags

SF: 1 if reg < 0, else 0
ZF: 1 if reg = 0, else 0
HF: 0
PF: 1 if parity even, else 0
NF: 0
CF: Content of bit-7 before operation

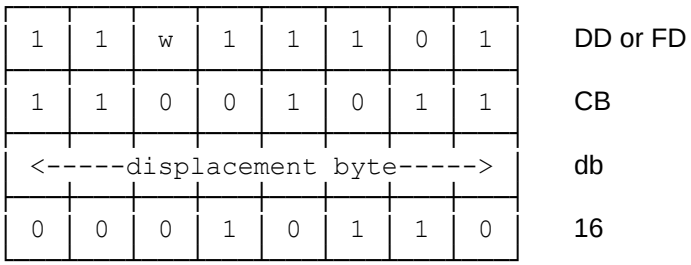
Clocking

	M-cycles	Tymes
RL reg	2	8 (4,4)
RL (HL)	4	15 (4,4,4,3)
RL (Ii+d)	6	23 (4,4,3,5,4,3)

Encoding



	v	op-code
RL B	000	CB 10
RL C	001	CB 11
RL D	010	CB 12
RL E	011	CB 13
RL H	100	CB 14
RL L	101	CB 15
RL (HL)	110	CB 16
RL A	111	CB 17



	w	op-code
RL (IX+d)	0	DD CB db 16
RL (IY+d)	1	FD CB db 16

RLA rotate accumulator left

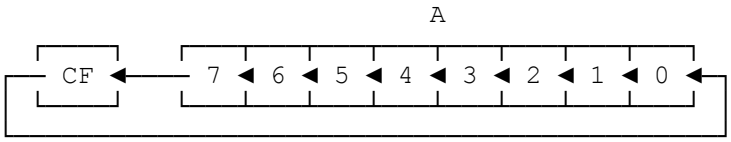
Operation

temp ← A
A[0] ← CF
A[1] ← temp[0]
A[2] ← temp[1]
A[3] ← temp[2]
A[4] ← temp[3]
A[5] ← temp[4]
A[6] ← temp[5]
A[7] ← temp[6]
CF ← temp[7]

Description

The bits of the accumulator plus carry are rotated left.

Graphically, this is:



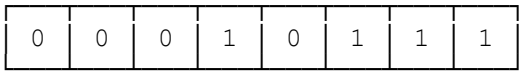
Flags

SF: Not Affected
ZF: Not Affected
HF: 0
PF: Not Affected
NF: 0
CF: Content of bit-7 before operation

Clocking

	M-cycles	Tymes
RLA	1	4

Encoding



17

	op-code
RLA	17

RLC dest rotate destination operand left circular

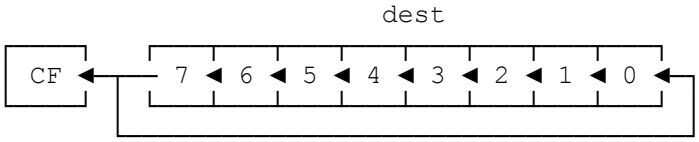
Operation

temp ← dest
dest[0] ← temp[7]
dest[1] ← temp[0]
dest[2] ← temp[1]
dest[3] ← temp[2]
dest[4] ← temp[3]
dest[5] ← temp[4]
dest[6] ← temp[5]
dest[7] ← temp[6]
CF ← temp[7]

Description

The bits of the destination operand are rotated left circular.

Graphically, this is:



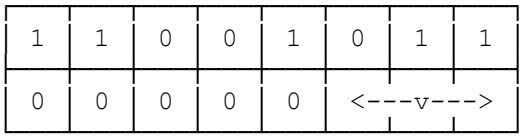
Flags

SF: 1 if reg < 0, else 0
ZF: 1 if reg = 0, else 0
HF: 0
PF: 1 if parity even, else 0
NF: 0
CF: Content of bit-7 before operation

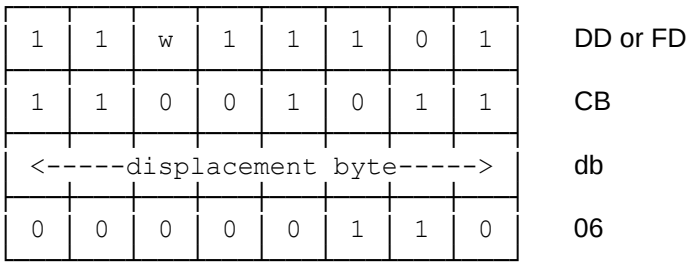
Clocking

	M-cycles	Tymes
RLC reg	2	8 (4,4)
RLC (HL)	4	15 (4,4,4,3)
RLC (Ii+d)	6	23 (4,4,3,5,4,3)

Encoding



	v	op-code
RLC B	000	CB 00
RLC C	001	CB 01
RLC D	010	CB 02
RLC E	011	CB 03
RLC H	100	CB 04
RLC L	101	CB 05
RLC (HL)	110	CB 06
RLC A	111	CB 07



	w	op-code
RLC (IX+d)	0	DD CB db 06
RLC (IY+d)	1	FD CB db 06

RLCA rotate accumulator left circular

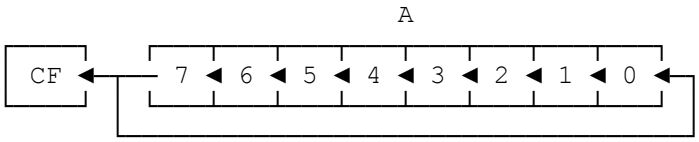
Operation

temp ← A
A[0] ← temp[7]
A[1] ← temp[0]
A[2] ← temp[1]
A[3] ← temp[2]
A[4] ← temp[3]
A[5] ← temp[4]
A[6] ← temp[5]
A[7] ← temp[6]
CF ← temp[7]

Description

The bits of the accumulator are rotated left circular.

Graphically, this is:



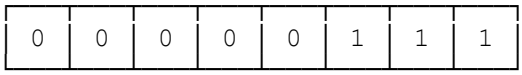
Flags

SF: Not Affected
ZF: Not Affected
HF: 0
PF: Not Affected
NF: 0
CF: Content of bit-7 before operation

Clocking

	M-cycles	Tymes
RLCA	1	4

Encoding



07

	op-code
RLCA	07

RLD rotate left digit

Operation

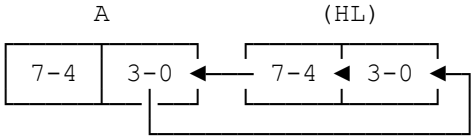
temp ← A
A[3] ← (HL)[7]
A[2] ← (HL)[6]
A[1] ← (HL)[5]
A[0] ← (HL)[4]
(HL)[7] ← (HL)[3]
(HL)[6] ← (HL)[2]
(HL)[5] ← (HL)[1]
(HL)[4] ← (HL)[0]
(HL)[3] ← temp[3]
(HL)[2] ← temp[2]
(HL)[1] ← temp[1]
(HL)[0] ← temp[0]

Description

The accumulator is temporarily saved.
The low-order nybble of the accumulator is made equal to the high-order nybble (bits 7–4) of the memory byte whose address is in the HL register pair.
The high-order nybble of the memory byte whose address is in the HL register pair is made equal to the low-order nybble of the same memory byte.
The low-order nybble of the memory byte whose address is in the HL register pair is made equal to the low-order nybble of the temporarily saved accumulator.

The high-order nybble of the accumulator is left unchanged.
The HL register pair is left unchanged.

Graphically, this is:



Flags

SF: 1 if A < 0, else 0
ZF: 1 if A = 0, else 0
HF: 0
PF: 1 if parity even, else 0
NF: 0
CF: Not Affected

Clocking

	M-cycles	Tymes
RLD	5	18 (4,4,3,4,3)

Encoding

1	1	1	0	1	1	0	1	ED
0	1	1	0	1	1	1	1	6F

	op-code
RLD	ED 6F

RR dest rotate destination operand right

Operation

temp ← dest
dest[7] ← CF
dest[6] ← temp[7]
dest[5] ← temp[6]
dest[4] ← temp[5]
dest[3] ← temp[4]
dest[2] ← temp[3]
dest[1] ← temp[2]
dest[0] ← temp[1]
CF ← temp[0]

Description

The bits of the destination operand plus carry are rotated right.

Graphically, this is:

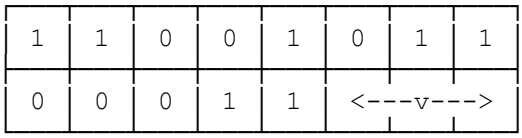
Flags

SF: 1 if reg < 0, else 0
ZF: 1 if reg = 0, else 0
HF: 0
PF: 1 if parity even, else 0
NF: 0
CF: Content of bit-0 before operation

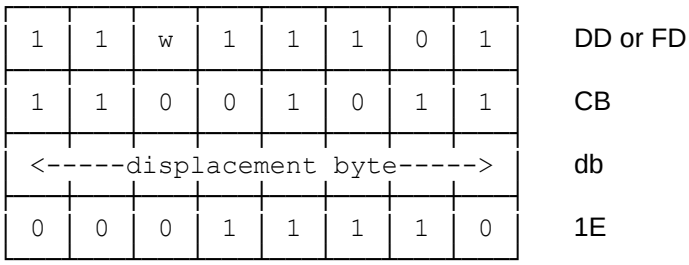
Clocking

	M-cycles	Tymes
RR reg	2	8 (4,4)
RR (HL)	4	15 (4,4,4,3)
RR (Ii+d)	6	23 (4,4,3,5,4,3)

Encoding



	v	op-code
RR B	000	CB 18
RR C	001	CB 19
RR D	010	CB 1A
RR E	011	CB 1B
RR H	100	CB 1C
RR L	101	CB 1D
RR (HL)	110	CB 1E
RR A	111	CB 1F



	w	op-code
RR (IX+d)	0	DD CB db 1E
RR (IY+d)	1	FD CB db 1E

RRA rotate accumulator right

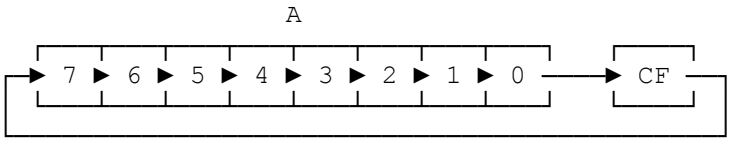
Operation

temp ← A
A[7] ← CF
A[6] ← temp[7]
A[5] ← temp[6]
A[4] ← temp[5]
A[3] ← temp[4]
A[2] ← temp[3]
A[1] ← temp[2]
A[0] ← temp[1]
CF ← temp[0]

Description

The bits of the accumulator plus carry are rotated right.

Graphically, this is:



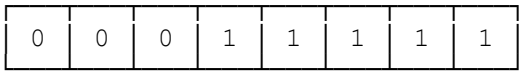
Flags

SF: Not Affected
ZF: Not Affected
HF: 0
PF: Not Affected
NF: 0
CF: Content of bit-0 before operation

Clocking

	M-cycles	Tymes
RRA	1	4

Encoding



1F

	op-code
RRA	1F

RRC dest rotate destination operand right circular

Operation

temp ← dest
dest[7] ← temp[0]
dest[6] ← temp[7]
dest[5] ← temp[6]
dest[4] ← temp[5]
dest[3] ← temp[4]
dest[2] ← temp[3]
dest[1] ← temp[2]
dest[0] ← temp[1]
CF ← temp[0]

Description

The bits of the destination operand are rotated right circular.

Graphically, this is:

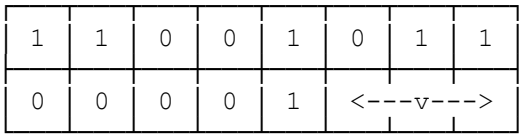
Flags

SF: 1 if reg < 0, else 0
ZF: 1 if reg = 0, else 0
HF: 0
PF: 1 if parity even, else 0
NF: 0
CF: Content of bit-0 before operation

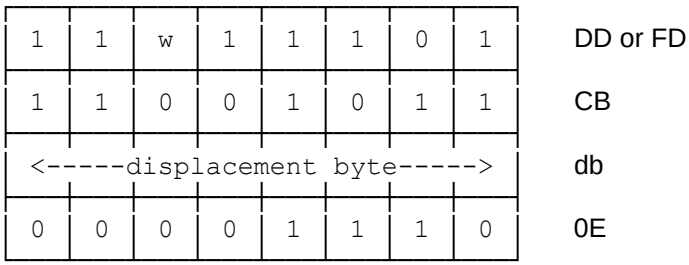
Clocking

	M-cycles	Tymes
RRC reg	2	8 (4,4)
RRC (HL)	4	15 (4,4,4,3)
RRC (Ii+d)	6	23 (4,4,3,5,4,3)

Encoding



	v	op-code
RRC B	000	CB 08
RRC C	001	CB 09
RRC D	010	CB 0A
RRC E	011	CB 0B
RRC H	100	CB 0C
RRC L	101	CB 0D
RRC (HL)	110	CB 0E
RRC A	111	CB 0F



	w	op-code
RRC (IX+d)	0	DD CB db 0E
RRC (IY+d)	1	FD CB db 0E

RRCA rotate accumulator right circular

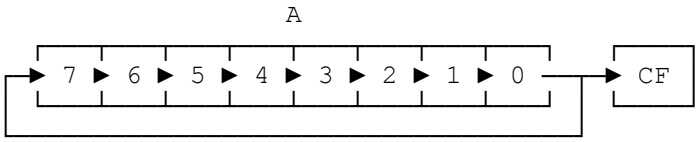
Operation

temp ← A
A[7] ← temp[0]
A[6] ← temp[7]
A[5] ← temp[6]
A[4] ← temp[5]
A[3] ← temp[4]
A[2] ← temp[3]
A[1] ← temp[2]
A[0] ← temp[1]
CF ← temp[0]

Description

The bits of the accumulator are rotated right circular.

Graphically, this is:



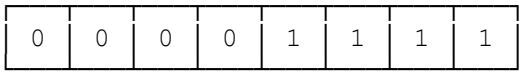
Flags

SF: Not Affected
ZF: Not Affected
HF: 0
PF: Not Affected
NF: Not Affected
CF: Content of bit-0 before operation

Clocking

	M-cycles	Tymes
RRCA	1	4

Encoding



0F

	op-code
RRCA	0F

RRD rotate right digit

Operation

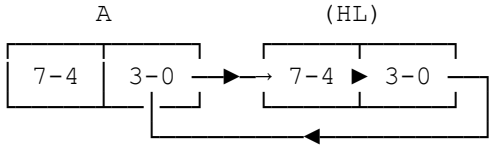
temp ← A
A[0] ← (HL)[0]
A[1] ← (HL)[1]
A[2] ← (HL)[2]
A[3] ← (HL)[3]
(HL)[0] ← (HL)[4]
(HL)[1] ← (HL)[5]
(HL)[2] ← (HL)[6]
(HL)[3] ← (HL)[7]
(HL)[4] ← temp[0]
(HL)[5] ← temp[1]
(HL)[6] ← temp[2]
(HL)[7] ← temp[3]

Description

The accumulator is temporarily saved.
The low-order nybble of the accumulator is made equal to the low-order nybble (bits 7–4) of the memory byte whose address is in the HL register pair.
The low-order nybble of the memory byte whose address is in the HL register pair is made equal to the high-order nybble of the same memory byte.
The high-order nybble of the memory byte whose address is in the HL register pair is made equal to the low-order nybble of the temporarily saved accumulator.

The high-order nybble of the accumulator is left unchanged.
The HL register pair is left unchanged.

Graphically, this is:



Flags

SF: 1 if A < 0, else 0
ZF: 1 if A = 0, else 0
HF: 0
PF: 1 if parity even, else 0
NF: 0
CF: Not Affected

Clocking

	M-cycles	Tymes
RRD	5	18 (4,4,3,4,3)

Encoding

1	1	1	0	1	1	0	1	ED
0	1	1	0	0	1	1	1	67

	op-code
RRD	ED 67

RST vector restart

Operation

SP ← SP – 1
(SP) ← PC-high
SP ← SP – 1
(SP) ← PC-low

PC-low ← vector
PC-high ← 00

Description

The stack pointer is decremented.
The memory byte whose address is in stack pointer is made equal to the high-order byte of the program counter.
The stack pointer is again decremented.
The memory byte whose address is in the stack pointer is made equal to the low-order byte of the program counter.

The low-order byte of the program counter is made equal to the immediate restart vector operand.
The high-order byte of the program counter is made equal to zero.

Execution continues at the address which is in the program counter.

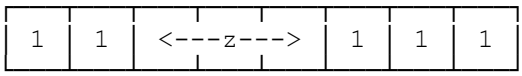
Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
RST vector	3	11 (5,3,3)

Encoding



	z	op-code
RST 00	000	C7
RST 08	001	CF
RST 10	010	D7
RST 18	011	DF
RST 20	100	E7
RST 28	101	EF
RST 30	110	F7
RST 38	111	FF

SBC dest,source subtract source operand less carry from destination operand

Operation
dest ← dest – source – CF

Description
The destination operand is made equal to the difference of the destination operand minus the source operand minus the status of the carry flag.

The source operand is left unchanged.

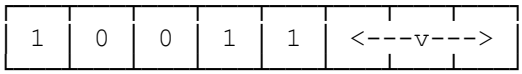
Flags

	If 8-bit Instruction:	If 16-bit Instruction:
SF:	1 if result < 0, else 0	1 if result < 0, else 0
ZF:	1 if result = 0, else 0	1 if result = 0, else 0
HF:	1 if bit-4 borrow, else 0	1 if bit 12 borrow, else 0
PF:	1 if underflow, else 0	1 if underflow, else 0
NF:	1	1
CF:	1 if bit-8 borrow, else 0	1 if bit 16 borrow, else 0

Clocking

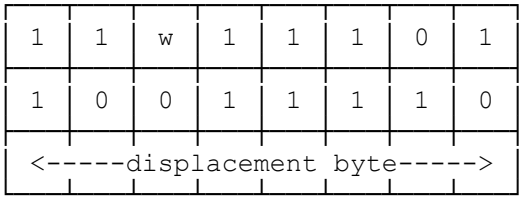
	M-cycles	Tymes
SBC A,reg	1	4
SBC A,(HL)	2	7 (4,3)
SBC A,(Ii+d)	5	19 (4,4,3,5,3)
SBC A,immed	2	7 (4,3)
SBC HL,regpr	4	15 (4,4,4,3)

Encoding



op

	v	op-code
SBC A,B	000	98
SBC A,C	001	99
SBC A,D	010	9A
SBC A,E	011	9B
SBC A,H	100	9C
SBC A,L	101	9D
SBC A,(HL)	110	9E
SBC A,A	111	9F

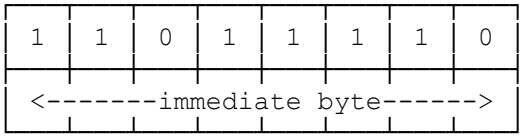


DD or FD

9E

db

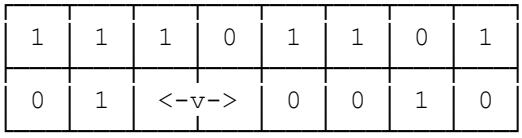
	w	op-code
SBC A,(IX+d)	0	DD 9E
SBC A,(IY+d)	1	FD 9E



DE

ib

	op-code
SBC A,immed	DE ib



ED

op

	v	op-code
SBC HL,BC	00	ED 42
SBC HL,DE	01	ED 52
SBC HL,HL	10	ED 62
SBC HL,SP	11	ED 72

SCF set carry flag

Operation
CF ← 1

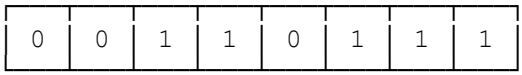
Description
The carry flag is set.

Flags
SF: Not Affected
ZF: Not Affected
HF: 0
PF: Not Affected
NF: 0
CF: 1

Clocking

	M-cycles	Tymes
SCF	1	4

Encoding



	op-code
SCF	37

SET bit,source set source operand bit

Operation
source[bit] ← 1

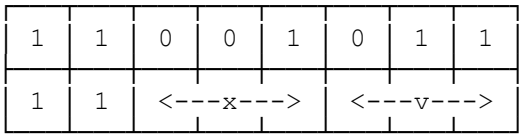
Description:
The specified bit of the source operand is set.

Flags
SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

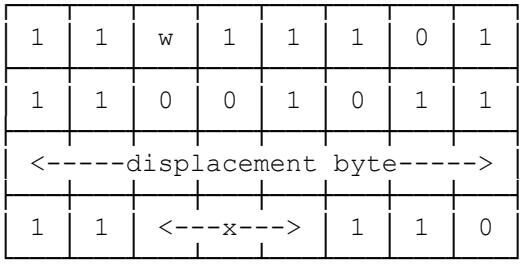
	M-cycles	Tymes
SET bit,reg	2	8 (4,4)
SET bit,(HL)	3	12 (4,4,4)
SET bit,(Ii+d)	5	20 (4,4,3,5,4)

Encoding



	x	v	op-code
SET 0,B	000	000	CB C0
SET 0,C	000	001	CB C1
SET 0,D	000	010	CB C2
SET 0,E	000	011	CB C3
SET 0,H	000	100	CB C4
SET 0,L	000	101	CB C5
SET 0,(HL)	000	110	CB C6
SET 0,A	000	111	CB C7
SET 1,B	001	000	CB C8
SET 1,C	001	001	CB C9
SET 1,D	001	010	CB CA
SET 1,E	001	011	CB CB
SET 1,H	001	100	CB CC
SET 1,L	001	101	CB CD
SET 1,(HL)	001	110	CB CE
SET 1,A	001	111	CB CF
SET 2,B	010	000	CB D0
SET 2,C	010	001	CB D1
SET 2,D	010	010	CB D2
SET 2,E	010	011	CB D3
SET 2,H	010	100	CB D4
SET 2,L	010	101	CB D5
SET 2,(HL)	010	110	CB D6
SET 2,A	010	111	CB D7

SET 3,B	011	000	CB D8
SET 3,C	011	001	CB D9
SET 3,D	011	010	CB DA
SET 3,E	011	011	CB DB
SET 3,H	011	100	CB DC
SET 3,L	011	101	CB DD
SET 3,(HL)	011	110	CB DE
SET 3,A	011	111	CB DF
SET 4,B	100	000	CB E0
SET 4,C	100	001	CB E1
SET 4,D	100	010	CB E2
SET 4,E	100	011	CB E3
SET 4,H	100	100	CB E4
SET 4,L	100	101	CB E5
SET 4,(HL)	100	101	CB E6
SET 4,A	100	111	CB E7
SET 5,B	101	000	CB E8
SET 5,C	101	001	CB E9
SET 5,D	101	010	CB EA
SET 5,E	101	011	CB EB
SET 5,H	101	100	CB EC
SET 5,L	101	101	CB ED
SET 5,(HL)	101	110	CB EE
SET 5,A	101	111	CB EF
SET 6,B	110	000	CB F0
SET 6,C	110	001	CB F1
SET 6,D	110	010	CB F2
SET 6,E	110	011	CB F3
SET 6,H	110	100	CB F4
SET 6,L	110	101	CB F5
SET 6,(HL)	110	110	CB F6
SET 6,A	110	111	CB F7
SET 7,B	111	000	CB F8
SET 7,C	111	001	CB F9
SET 7,D	111	010	CB FA
SET 7,E	111	011	CB FB
SET 7,H	111	100	CB FC
SET 7,L	111	101	CB FD
SET 7,(HL)	111	110	CB FE
SET 7,A	111	111	CB FF



	w	x	op-code
SET 0,(IX+d)	0	000	DD CB db C6
SET 1,(IX+d)	0	001	DD CB db CE
SET 2,(IX+d)	0	010	DD CB db D6
SET 3,(IX+d)	0	011	DD CB db DE
SET 4,(IX+d)	0	100	DD CB db E6
SET 5,(IX+d)	0	101	DD CB db EE
SET 6,(IX+d)	0	110	DD CB db F6
SET 7,(IX+d)	0	111	DD CB db FE
SET 0,(IY+d)	1	000	FD CB db C6
SET 1,(IY+d)	1	001	FD CB db CE
SET 2,(IY+d)	1	010	FD CB db D6
SET 3,(IY+d)	1	011	FD CB db DE
SET 4,(IY+d)	1	100	FD CB db E6
SET 5,(IY+d)	1	101	FD CB db EE
SET 6,(IY+d)	1	110	FD CB db F6
SET 7,(IY+d)	1	111	FD CB db FE

SLA dest shift destination operand left arithmetic

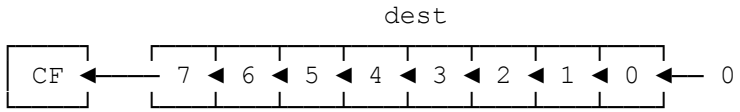
Operation

temp ← dest
dest[0] ← 0
dest[1] ← temp[0]
dest[2] ← temp[1]
dest[3] ← temp[2]
dest[4] ← temp[3]
dest[5] ← temp[4]
dest[6] ← temp[5]
dest[7] ← temp[6]
CF ← temp[7]

Description

The bits of the destination operand are shifted left arithmetic.
The carry flag is made equal to the most significant bit of the destination operand.

Graphically, this is:



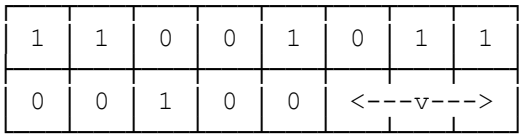
Flags

SF: 1 if reg < 0, else 0
ZF: 1 if reg = 0, else 0
HF: 0
PF: 1 if even parity, else 0
NF: 0
CF: Content of bit-7 before operation

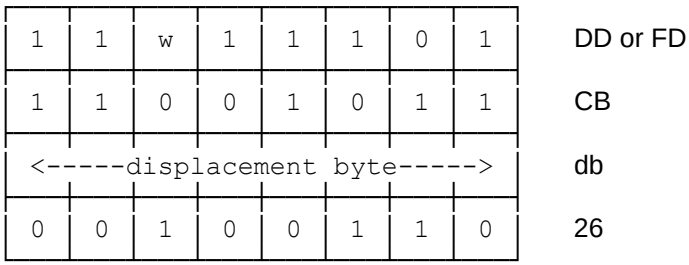
Clocking

	M-cycles	Tymes
SLA reg	2	8 (4,4)
SLA (HL)	4	15 (4,4,4,3)
SLA (Ii+d)	6	26 (4,4,3,5,4,3)

Encoding



	v	op-code
SLA B	000	CB 20
SLA C	001	CB 21
SLA D	010	CB 22
SLA E	011	CB 23
SLA H	100	CB 24
SLA L	101	CB 25
SLA (HL)	110	CB 26
SLA A	111	CB 27



	w	op-code
SLA (IX+d)	0	DD CB db 26
SLA (IY+d)	1	FD CB db 26

SLP enter sleep or system stop mode

Operation

If IOSTP = 0: (IOSTP is IOreg[5])
Enter SLEEP mode.

If IOSTP = 1:
Enter SYSTEM STOP mode.

Description

If the IOSTP bit is cleared:
The CPU enters the sleep mode and halt operation until reset or until an interrupt is received.
The sleep mode differs from the halt mode in that memory refresh is not maintained and power consumption is minimized.

If the IOSTP bit is set:
The CPU enters the system stop mode.
The system stop mode is identical to the sleep mode, with the addition that I/O access is also halted, prohibiting an I/O device interrupt from terminating system stop.

Flags

SF: Not Affected
ZF: Not Affected
HF: Not Affected
PF: Not Affected
NF: Not Affected
CF: Not Affected

Clocking

	M-cycles	Tymes
SLP	2	8

Encoding

1	1	1	0	1	1	0	1	ED
0	1	1	1	0	1	1	0	76

	op-code
SLP	ED 76

SRA dest shift destination operand right arithmetic

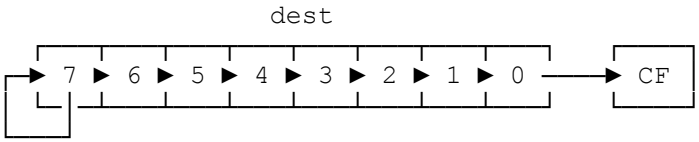
Operation

temp ← dest
dest[7] ← temp[7]
dest[6] ← temp[7]
dest[5] ← temp[6]
dest[4] ← temp[5]
dest[3] ← temp[4]
dest[2] ← temp[3]
dest[1] ← temp[2]
dest[0] ← temp[1]
CF ← temp[0]

Description

The bits of the destination operand are shifted right arithmetic.
The carry flag is made equal to the least significant bit of the destination operand.

Graphically, this is:



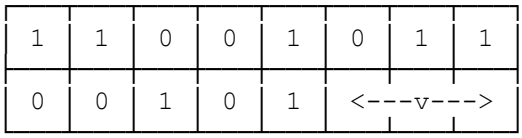
Flags

SF: 1 if reg < 0, else 0
ZF: 1 if reg = 0, else 0
HF: 0
PF: 1 if even parity, else 0
NF: 0
CF: Content of bit-0 before operation

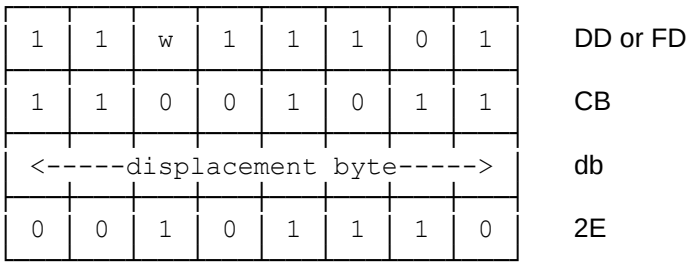
Clocking

	M-cycles	Tymes
SRA reg	2	8 (4,4)
SRA (HL)	4	15 (4,4,4,3)
SRA (Ii+d)	6	26 (4,4,3,5,4,3)

Encoding



	v	op-code
SRA B	000	CB 28
SRA C	001	CB 29
SRA D	010	CB 2A
SRA E	011	CB 2B
SRA H	100	CB 2C
SRA L	101	CB 2D
SRA (HL)	110	CB 2E
SRA A	111	CB 2F



	w	op-code
SRA (IX+d)	0	DD CB db 2E
SRA (IY+d)	1	FD CB db 2E

SRL dest shift destination operand right logical

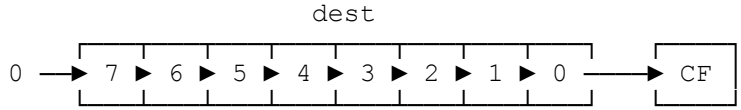
Operation

temp ← dest
dest[7] ← 0
dest[6] ← temp[7]
dest[5] ← temp[6]
dest[4] ← temp[5]
dest[3] ← temp[4]
dest[2] ← temp[3]
dest[1] ← temp[2]
dest[0] ← temp[1]
CF ← temp[0]

Description

The bits of the destination operand are shifted right logical.
The carry flag is made equal to the least significant bit of the destination operand.

Graphically, this is:



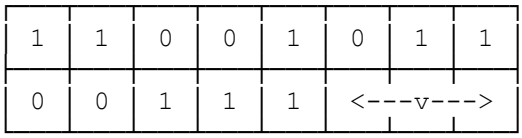
Flags

	If 8-bit Instruction:	If 16-bit Instruction:
SF:	1 if reg < 0, else 0	
ZF:	1 if reg = 0, else 0	
HF:	0	
PF:	1 if even parity, else 0	
NF:	0	
CF:	Content of bit-0 before operation	

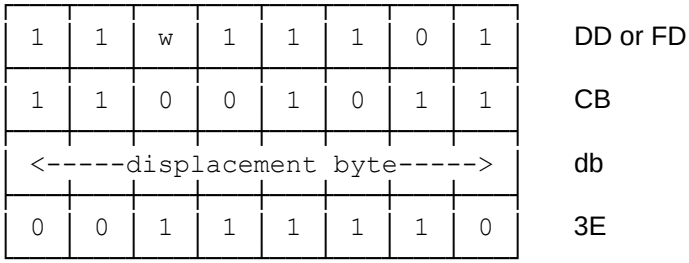
Clocking

	M-cycles	Tymes
SRL reg	2	8 (4,4)
SRL (HL)	4	15 (4,4,4,3)
SRL (Ii+d)	6	26 (4,4,3,5,4,3)

Encoding



	v	op-code
SRL B	000	CB 38
SRL C	001	CB 39
SRL D	010	CB 3A
SRL E	011	CB 3B
SRL H	100	CB 3C
SRL L	101	CB 3D
SRL (HL)	110	CB 3E
SRL A	111	CB 3F



	w	op-code
SRL (IX+d)	0	DD CB db 3E
SRL (IY+d)	1	FD CB db 3E

SUB source subtract source operand from accumulator

Operation
A ← A – source

Description
The accumulator is made equal to the difference of the accumulator minus the source operand.

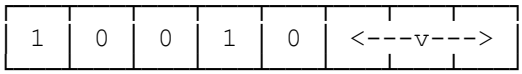
The source operand is left unchanged.

Flags
SF: 1 if A < 0, else 0
ZF: 1 if A = 0, else 0
HF: 1 if bit-4 borrow, else 0
PF: 1 if underflow, else 0
NF: 1
CF: 1 if bit-8 borrow, else 0

Clocking

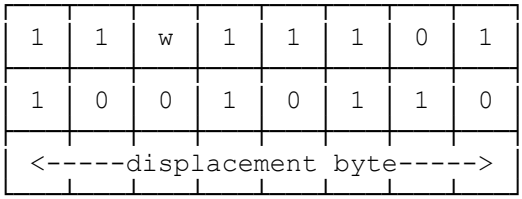
	M-cycles	Tymes
SUB reg	1	4
SUB (HL)	2	7 (4,3)
SUB (Ii+d)	5	19 (4,4,3,5,3)
SUB immed	2	7 (4,3)

Encoding



op

	v	op-code
SUB B	000	90
SUB C	001	91
SUB D	010	92
SUB E	011	93
SUB H	100	94
SUB L	101	95
SUB (HL)	110	96
SUB A	111	97

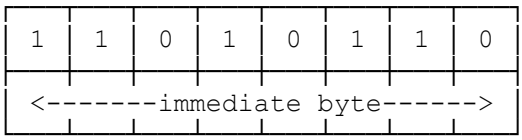


DD or FD

96

db

	w	op-code
SUB (IX+d)	0	DD 96
SUB (IY+d)	1	FD 96



D6

ib

	op-code
SUB immed	D6 ib

TST source test source operand against accumulator

Operation

temp[0] ← A[0] AND source[0]
temp[1] ← A[1] AND source[1]
temp[2] ← A[2] AND source[2]
temp[3] ← A[3] AND source[3]
temp[4] ← A[4] AND source[4]
temp[5] ← A[5] AND source[5]
temp[6] ← A[6] AND source[6]
temp[7] ← A[7] AND source[7]

Description

The source operand is bitwise ANDed with the accumulator.

The accumulator is left unchanged
The source operand is left unchanged.

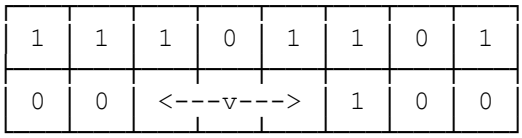
Flags

SF: 1 if result < 0, else 0
ZF: 1 if result = 0, else 0
HF: 1
PF: 1 if parity even, else 0
NF: 0
CF: 0

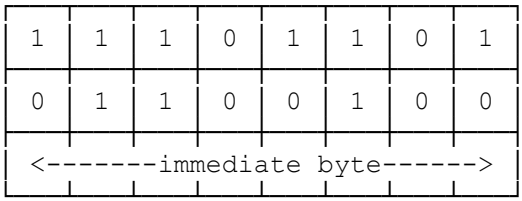
Clocking

	M-cycles	Tymes
TST reg	3	7
TST (HL)	4	10
TST immed	3	9

Encoding



	v	op-code
TST B	000	ED 04
TST C	001	ED 0C
TST D	010	ED 14
TST E	011	ED 1C
TST H	100	ED 24
TST L	101	ED 2C
TST (HL)	110	ED 34
TST A	111	ED 3C



	op-code
TST immed	ED 64 ib

TSTIO immed — test immediate byte against I/O byte

Operation
address-bus-low ← (C)
address-bus-high ← 00

temp[0] ← (address-bus)[0] AND immed[0]
temp[1] ← (address-bus)[1] AND immed[1]
temp[2] ← (address-bus)[2] AND immed[2]
temp[3] ← (address-bus)[3] AND immed[3]
temp[4] ← (address-bus)[4] AND immed[4]
temp[5] ← (address-bus)[5] AND immed[5]
temp[6] ← (address-bus)[6] AND immed[6]
temp[7] ← (address-bus)[7] AND immed[7]

Description
The low-order byte of the address bus is made equal to the C register.
The high-order byte of the address bus is made equal to 00.
The I/O byte whose address is the low-order byte of the address bus is bitwise ANDed with the immediate byte operand.

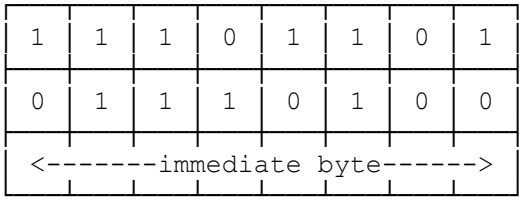
The C register is left unchanged.

Flags
SF: 1 if result < 0, else 0
ZF: 1 if result = 0, else 0
HF: 1
PF: 1 if parity even, else 0
NF: 0
CF: 0

Clocking

	M-cycles	Tymes
TSTIO immed	4	12

Encoding



	op-code
TSTIO immed	ED 74 ib

XOR source exclusive-OR source operand with accumulator

Operation

A[0] ← A[0] XOR source[0]
A[1] ← A[1] XOR source[1]
A[2] ← A[2] XOR source[2]
A[3] ← A[3] XOR source[3]
A[4] ← A[4] XOR source[4]
A[5] ← A[5] XOR source[5]
A[6] ← A[6] XOR source[6]
A[7] ← A[7] XOR source[7]

Description

The accumulator is made equal to the source operand bitwise exclusive-ORed (XORed) with the accumulator.

The source operand is left unchanged.

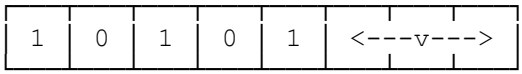
Flags

SF: 1 if A < 0, else 0
ZF: 1 if A = 0, else 0
HF: 1
PF: 1 if parity even, else 0
NF: 0
CF: 0

Clocking

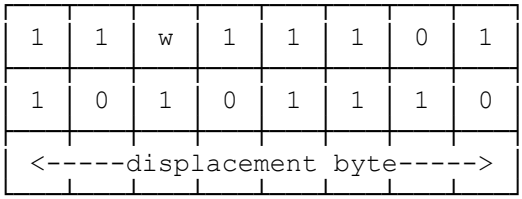
	M-cycles	Tymes
XOR reg	1	4
XOR (HL)	2	7 (4,3)
XOR (Ii+d)	5	19 (4,4,3,5,3)
XOR immed	2	7 (4,3)

Encoding



op

	v	op-code
XOR B	000	A8
XOR C	001	A9
XOR D	010	AA
XOR E	011	AB
XOR H	100	AC
XOR L	101	AD
XOR (HL)	110	AE
XOR A	111	AF

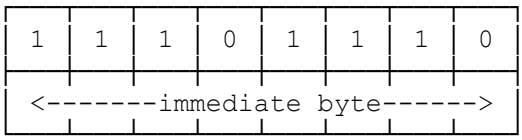


DD or FD

AE

db

	w	op-code
XOR (IX+d)	0	DD AE db
XOR (IY+d)	1	FD AE db



EE

ib

	op-code
XOR immed	EE ib

Machine Code Disassembly

Following is a disassembly from machine code to Zilog, TDL and MAC mnemonics. This table can be of great assistance in translation. The pages given contain the detailed instructions for the op-codes

Those Zilog mnemonics indicated by a "~" and missing in TDL and MAC mnemonics are unique to the Z-180 processor.

Those TDL mnemonics that are indicated by an "*" are the Intel 8080 processor subset, and usually use the TDL or MAC mnemonics (identical for all 8080 instructions).

Disassembly	Zilog	TDL	MAC	Page
00	NOP	* NOP	NOP	13–61
01 il ih	LD BC,imwrd	* LXI B,imwrd	LXI B,imwrd	13–48
02	LD (BC),A	* STAX B	STAX B	13–48
03	INC BC	* INX B	INX B	13–38
04	INC B	* INR B	INR B	13–38
05	DEC B	* DCR B	DCR B	13–25
06 ib	LD B,immed	* MVI B,immed	MVI B,immed	13–48
07	RLCA	* RLC	RLC	13–89
08	EX AF,AF'	EXAF	EXAF	13–30
09	ADD HL,BC	* DAD B	DAD B	13–6
0A	LD A,(BC)	* LDAX B	LDAX B	13–48
0B	DEC BC	* DCX B	DCX B	13–25
0C	INC C	* INR C	INR C	13–38
0D	DEC C	* DCR C	DCR C	13–25
0E ib	LD C,immed	* MVI C,immed	MVI C,immed	13–48
0F	RRCA	* RRC	RRC	13–96
10 ab	DJNZ disp	DJNZ disp	DJNZ disp	13–28
11 il ih	LD DE,imwrd	* LXI D,imwrd	LXI D,imwrd	13–48
12	LD (DE),A	* STAX D	STAX D	13–48
13	INC DE	* INX D	INX D	13–38
14	INC D	* INR D	INR D	13–38
15	DEC D	* DCR D	DCR D	13–25
16 ib	LD D,immed	* MVI D,immed	MVI D,immed	13–48
17	RLA	* RAL	RAL	13–86
18 ab	JR disp	JMPR disp	JR disp	13–46
19	ADD HL,DE	* DAD D	DAD D	13–6
1A	LD A,(DE)	* LDAX D	LDAX D	13–48
1B	DEC DE	* DCX D	DCX D	13–25
1C	INC E	* INR E	INR E	13–38
1D	DEC E	* DCR E	DCR E	13–25
1E ib	LD E,immed	* MVI E,immed	MVI E,immed	13–48
1F	RRA	* RAR	RAR	13–93
20 ab	JR NZ,disp	JRNZ disp	JRNZ disp	13–46
21 il ih	LD HL,imwrd	* LXI H,imwrd	LXI H,imwrd	13–48
22 al ah	LD (addr),HL	* SHLD addr	SHLD addr	13–48
23	INC HL	* INX H	INX H	13–38
24	INC H	* INR H	INR H	13–38
25	DEC H	* DCR H	DCR H	13–25
26 ib	LD H,immed	* MVI H,immed	MVI H,immed	13–48
27	DAA	* DAA	DAA	13–23

Disassembly	Zilog		TDL	MAC	Page
28 ab	JR Z,disp		JRZ disp	JRZ disp	13–46
29	ADD HL,HL	*	DAD H	DAD H	13–6
2A al ah	LD HL,(addr)	*	LHLD addr	LHLD addr	13–48
2B	DEC HL	*	DCX H	DCX H	13–25
2C	INC L	*	INR L	INR L	13–38
2D	DEC L	*	DCR L	DCR L	13–25
2E ib	LD L,immed	*	MVI L,immed	MVI L,immed	13–48
2F	CPL	*	CMA	CMA	13–22
30 ab	JR NC,disp		JRNC disp	JRNC disp	13–46
31 il ih	LD SP,imwrd	*	LXI SP,imwrd	LXI SP,imwrd	13–48
32 al ah	LD (addr),A	*	STA addr	STA addr	13–48
33	INC SP	*	INX SP	INX SP	13–38
34	INC (HL)	*	INR M	INR M	13–38
35	DEC (HL)	*	DCR M	DCR M	13–25
36 ib	LD (HL),immed	*	MVI M,immed	MVI M,immed	13–48
37	SCF	*	STC	STC	13–101
38 ab	JR C,disp		JRC disp	JRC disp	13–46
39	ADD HL,SP	*	DAD SP	DAD SP	13–6
3A al ah	LD A,(addr)	*	LDA addr	LDA addr	13–48
3B	DEC SP	*	DCX SP	DCX SP	13–25
3C	INC A	*	INR A	INR A	13–38
3D	DEC A	*	DCR A	DCR A	13–25
3E ib	LD A,immed	*	MVI A,immed	MVI A,immed	13–48
3F	CCF	*	CMC	CMC	13–15
40	LD B,B	*	MOV B,B	MOV B,B	13–48
41	LD B,C	*	MOV B,C	MOV B,C	13–48
42	LD B,D	*	MOV B,D	MOV B,D	13–48
43	LD B,E	*	MOV B,E	MOV B,E	13–48
44	LD B,H	*	MOV B,H	MOV B,H	13–48
45	LD B,L	*	MOV B,L	MOV B,L	13–48
46	LD B,(HL)	*	MOV B,M	MOV B,M	13–48
47	LD B,A	*	MOV B,A	MOV B,A	13–48
48	LD C,B	*	MOV C,B	MOV C,B	13–48
49	LD C,C	*	MOV C,C	MOV C,C	13–48
4A	LD C,D	*	MOV C,D	MOV C,D	13–48
4B	LD C,E	*	MOV C,E	MOV C,E	13–48
4C	LD C,H	*	MOV C,H	MOV C,H	13–48
4D	LD C,L	*	MOV C,L	MOV C,L	13–48
4E	LD C,(HL)	*	MOV C,M	MOV C,M	13–48
4F	LD C,A	*	MOV C,A	MOV C,A	13–48
50	LD D,B	*	MOV D,B	MOV D,B	13–48
51	LD D,C	*	MOV D,C	MOV D,C	13–48
52	LD D,D	*	MOV D,D	MOV D,D	13–48
53	LD D,E	*	MOV D,E	MOV D,E	13–48
54	LD D,H	*	MOV D,H	MOV D,H	13–48
55	LD D,L	*	MOV D,L	MOV D,L	13–48
56	LD D,(HL)	*	MOV D,M	MOV D,M	13–48
57	LD D,A	*	MOV D,A	MOV D,A	13–48

Disassembly	Zilog	TDL	MAC	Page
58	LD E,B	* MOV E,B	MOV E,B	13–48
59	LD E,C	* MOV E,C	MOV E,C	13–48
5A	LD E,D	* MOV E,D	MOV E,D	13–48
5B	LD E,E	* MOV E,E	MOV E,E	13–48
5C	LD E,H	* MOV E,H	MOV E,H	13–48
5D	LD E,L	* MOV E,L	MOV E,L	13–48
5E	LD E,(HL)	* MOV E,M	MOV E,M	13–48
5F	LD E,A	* MOV E,A	MOV E,A	13–48
60	LD H,B	* MOV H,B	MOV H,B	13–48
61	LD H,C	* MOV H,C	MOV H,C	13–48
62	LD H,D	* MOV H,D	MOV H,D	13–48
63	LD H,E	* MOV H,E	MOV H,E	13–48
64	LD H,H	* MOV H,H	MOV H,H	13–48
65	LD H,L	* MOV H,L	MOV H,L	13–48
66	LD H,(HL)	* MOV H,M	MOV H,M	13–48
67	LD H,A	* MOV H,A	MOV H,A	13–48
68	LD L,B	* MOV L,B	MOV L,B	13–48
69	LD L,C	* MOV L,C	MOV L,C	13–48
6A	LD L,D	* MOV L,D	MOV L,D	13–48
6B	LD L,E	* MOV L,E	MOV L,E	13–48
6C	LD L,H	* MOV L,H	MOV L,H	13–48
6D	LD L,L	* MOV L,L	MOV L,L	13–48
6E	LD L,(HL)	* MOV L,M	MOV L,M	13–48
6F	LD L,A	* MOV L,A	MOV L,A	13–48
70	LD (HL),B	* MOV M,B	MOV M,B	13–48
71	LD (HL),C	* MOV M,C	MOV M,C	13–48
72	LD (HL),D	* MOV M,D	MOV M,D	13–48
73	LD (HL),E	* MOV M,E	MOV M,E	13–48
74	LD (HL),H	* MOV M,H	MOV M,H	13–48
75	LD (HL),L	* MOV M,L	MOV M,L	13–48
76	HALT	* HLT	HLT	13–32
77	LD (HL),A	* MOV M,A	MOV M,A	13–48
78	LD A,B	* MOV A,B	MOV A,B	13–48
79	LD A,C	* MOV A,C	MOV A,C	13–48
7A	LD A,D	* MOV A,D	MOV A,D	13–48
7B	LD A,E	* MOV A,E	MOV A,E	13–48
7C	LD A,H	* MOV A,H	MOV A,H	13–48
7D	LD A,L	* MOV A,L	MOV A,L	13–48
7E	LD A,(HL)	* MOV A,M	MOV A,M	13–48
7F	LD A,A	* MOV A,A	MOV A,A	13–48
80	ADD A,B	* ADD B	ADD B	13–6
81	ADD A,C	* ADD C	ADD C	13–6
82	ADD A,D	* ADD D	ADD D	13–6
83	ADD A,E	* ADD E	ADD E	13–6
84	ADD A,H	* ADD H	ADD H	13–6
85	ADD A,L	* ADD L	ADD L	13–6
86	ADD A,(HL)	* ADD M	ADD M	13–6
87	ADD A,A	* ADD A	ADD A	13–6

Disassembly	Zilog	TDL	MAC	Page
88	ADC A,B	* ADC B	ADC B	13–4
89	ADC A,C	* ADC C	ADC C	13–4
8A	ADC A,D	* ADC D	ADC D	13–4
8B	ADC A,E	* ADC E	ADC E	13–4
8C	ADC A,H	* ADC H	ADC H	13–4
8D	ADC A,L	* ADC L	ADC L	13–4
8E	ADC A,(HL)	* ADC M	ADC M	13–4
8F	ADC A,A	* ADC A	ADC A	13–4
90	SUB B	* SUB B	SUB B	13–112
91	SUB C	* SUB C	SUB C	13–112
92	SUB D	* SUB D	SUB D	13–112
93	SUB E	* SUB E	SUB E	13–112
94	SUB H	* SUB H	SUB H	13–112
95	SUB L	* SUB L	SUB L	13–112
96	SUB (HL)	* SUB M	SUB M	13–112
97	SUB A	* SUB A	SUB A	13–112
98	SBC A,B	* SBB B	SBB B	13–99
99	SBC A,C	* SBB C	SBB C	13–99
9A	SBC A,D	* SBB D	SBB D	13–99
9B	SBC A,E	* SBB E	SBB E	13–99
9C	SBC A,H	* SBB H	SBB H	13–99
9D	SBC A,L	* SBB L	SBB L	13–99
9E	SBC A,(HL)	* SBB M	SBB M	13–99
9F	SBC A,A	* SBB A	SBB A	13–99
A0	AND B	* ANA B	ANA B	13–8
A1	AND C	* ANA C	ANA C	13–8
A2	AND D	* ANA D	ANA D	13–8
A3	AND E	* ANA E	ANA E	13–8
A4	AND H	* ANA H	ANA H	13–8
A5	AND L	* ANA L	ANA L	13–8
A6	AND (HL)	* ANA M	ANA M	13–8
A7	AND A	* ANA A	ANA A	13–8
A8	XOR B	* XRA B	XRA B	13–117
A9	XOR C	* XRA C	XRA C	13–117
AA	XOR D	* XRA D	XRA D	13–117
AB	XOR E	* XRA E	XRA E	13–117
AC	XOR H	* XRA H	XRA H	13–117
AD	XOR L	* XRA L	XRA L	13–117
AE	XOR (HL)	* XRA M	XRA M	13–117
AF	XOR A	* XRA A	XRA A	13–117
B0	OR B	* ORA B	ORA B	13–62
B1	OR C	* ORA C	ORA C	13–62
B2	OR D	* ORA D	ORA D	13–62
B3	OR E	* ORA E	ORA E	13–62
B4	OR H	* ORA H	ORA H	13–62
B5	OR L	* ORA L	ORA L	13–62
B6	OR (HL)	* ORA M	ORA M	13–62
B7	OR A	* ORA A	ORA A	13–62

Disassembly	Zilog	TDL	MAC	Page
B8	CP B	* CMP B	CMP B	13–16
B9	CP C	* CMP C	CMP C	13–16
BA	CP D	* CMP D	CMP D	13–16
BB	CP E	* CMP E	CMP E	13–16
BC	CP H	* CMP H	CMP H	13–16
BD	CP L	* CMP L	CMP L	13–16
BE	CP (HL)	* CMP M	CMP M	13–16
BF	CP A	* CMP A	CMP A	13–16
C0	RET NZ	* RNZ	RNZ	13–80
C1	POP BC	* POP B	POP B	13–75
C2 al ah	JP NZ,addr	* JNZ addr	JNZ addr	13–44
C3 al ah	JP addr	* JMP addr	JMP addr	13–44
C4 al ah	CALL NZ,addr	* CNZ addr	CNZ addr	13–13
C5	PUSH BC	* PUSH B	PUSH B	13–76
C6 ib	ADD A,immed	* ADI immed	ADI immed	13–6
C7	RST 00	* RST 0	RST 0	13–98
C8	RET Z	* RZ	RZ	13–80
C9	RET	* RET	RET	13–80
CA al ah	JP Z,addr	* JZ addr	JZ addr	13–44
CB 00	RLC B	RLCR B	RLCR B	13–87
CB 01	RLC C	RLCR C	RLCR C	13–87
CB 02	RLC D	RLCR D	RLCR D	13–87
CB 03	RLC E	RLCR E	RLCR E	13–87
CB 04	RLC H	RLCR H	RLCR H	13–87
CB 05	RLC L	RLCR L	RLCR L	13–87
CB 06	RLC (HL)	RLCR M	RLCR M	13–87
CB 07	RLC A	RLCR A	RLCR A	13–87
CB 08	RRC B	RRCR B	RRCR B	13–94
CB 09	RRC C	RRCR C	RRCR C	13–94
CB 0A	RRC D	RRCR D	RRCR D	13–94
CB 0B	RRC E	RRCR E	RRCR E	13–94
CB 0C	RRC H	RRCR H	RRCR H	13–94
CB 0D	RRC L	RRCR L	RRCR L	13–94
CB 0E	RRC (HL)	RRCR M	RRCR M	13–94
CB 0F	RRC A	RRCR A	RRCR A	13–94
CB 10	RL B	RALR B	RALR B	13–84
CB 11	RL C	RALR C	RALR C	13–84
CB 12	RL D	RALR D	RALR D	13–84
CB 13	RL E	RALR E	RALR E	13–84
CB 14	RL H	RALR H	RALR H	13–84
CB 15	RL L	RALR L	RALR L	13–84
CB 16	RL (HL)	RALR M	RALR M	13–84
CB 17	RL A	RALR A	RALR A	13–84
CB 18	RR B	RARR B	RARR B	13–91
CB 19	RR C	RARR C	RARR C	13–91
CB 1A	RR D	RARR D	RARR D	13–91
CB 1B	RR E	RARR E	RARR E	13–91
CB 1C	RR H	RARR H	RARR H	13–91
CB 1D	RR L	RARR L	RARR L	13–91
CB 1E	RR (HL)	RARR M	RARR M	13–91
CB 1F	RR A	RARR A	RARR A	13–91

Disassembly	Zilog	TDL	MAC	Page
CB 20	SLA B	SLAR B	SLAR B	13–105
CB 21	SLA C	SLAR C	SLAR C	13–105
CB 22	SLA D	SLAR D	SLAR D	13–105
CB 23	SLA E	SLAR E	SLAR E	13–105
CB 24	SLA H	SLAR H	SLAR H	13–105
CB 25	SLA L	SLAR L	SLAR L	13–105
CB 26	SLA (HL)	SLAR M	SLAR M	13–105
CB 27	SLA A	SLAR A	SLAR A	13–105
CB 28	SRA B	SRAR B	SRAR B	13–108
CB 29	SRA C	SRAR C	SRAR C	13–108
CB 2A	SRA D	SRAR D	SRAR D	13–108
CB 2B	SRA E	SRAR E	SRAR E	13–108
CB 2C	SRA H	SRAR H	SRAR H	13–108
CB 2D	SRA L	SRAR L	SRAR L	13–108
CB 2E	SRA (HL)	SRAR M	SRAR M	13–108
CB 2F	SRA A	SRAR A	SRAR A	13–108
CB 38	SRL B	SRLR B	SRLR B	13–110
CB 39	SRL C	SRLR C	SRLR C	13–110
CB 3A	SRL D	SRLR D	SRLR D	13–110
CB 3B	SRL E	SRLR E	SRLR E	13–110
CB 3C	SRL H	SRLR H	SRLR H	13–110
CB 3D	SRL L	SRLR L	SRLR L	13–110
CB 3E	SRL (HL)	SRLR M	SRLR M	13–110
CB 3F	SRL A	SRLR A	SRLR A	13–110
CB 40	BIT 0,B	BIT 0,B	BIT 0,B	13–10
CB 41	BIT 0,C	BIT 0,C	BIT 0,C	13–10
CB 42	BIT 0,D	BIT 0,D	BIT 0,D	13–10
CB 43	BIT 0,E	BIT 0,E	BIT 0,E	13–10
CB 44	BIT 0,H	BIT 0,H	BIT 0,H	13–10
CB 45	BIT 0,L	BIT 0,L	BIT 0,L	13–10
CB 46	BIT 0,(HL)	BIT 0,M	BIT 0,M	13–10
CB 47	BIT 0,A	BIT 0,A	BIT 0,A	13–10
CB 48	BIT 1,B	BIT 1,B	BIT 1,B	13–10
CB 49	BIT 1,C	BIT 1,C	BIT 1,C	13–10
CB 4A	BIT 1,D	BIT 1,D	BIT 1,D	13–10
CB 4B	BIT 1,E	BIT 1,E	BIT 1,E	13–10
CB 4C	BIT 1,H	BIT 1,H	BIT 1,H	13–10
CB 4D	BIT 1,L	BIT 1,L	BIT 1,L	13–10
CB 4E	BIT 1,(HL)	BIT 1,M	BIT 1,M	13–10
CB 4F	BIT 1,A	BIT 1,A	BIT 1,A	13–10
CB 50	BIT 2,B	BIT 2,B	BIT 2,B	13–10
CB 51	BIT 2,C	BIT 2,C	BIT 2,C	13–10
CB 52	BIT 2,D	BIT 2,D	BIT 2,D	13–10
CB 53	BIT 2,E	BIT 2,E	BIT 2,E	13–10
CB 54	BIT 2,H	BIT 2,H	BIT 2,H	13–10
CB 55	BIT 2,L	BIT 2,L	BIT 2,L	13–10
CB 56	BIT 2,(HL)	BIT 2,M	BIT 2,M	13–10
CB 57	BIT 2,A	BIT 2,A	BIT 2,A	13–10

Disassembly	Zilog	TDL	MAC	Page
CB 58	BIT 3,B	BIT 3,B	BIT 3,B	13–10
CB 59	BIT 3,C	BIT 3,C	BIT 3,C	13–10
CB 5A	BIT 3,D	BIT 3,D	BIT 3,D	13–10
CB 5B	BIT 3,E	BIT 3,E	BIT 3,E	13–10
CB 5C	BIT 3,H	BIT 3,H	BIT 3,H	13–10
CB 5D	BIT 3,L	BIT 3,L	BIT 3,L	13–10
CB 5E	BIT 3,(HL)	BIT 3,M	BIT 3,M	13–10
CB 5F	BIT 3,A	BIT 3,A	BIT 3,A	13–10
CB 60	BIT 4,B	BIT 4,B	BIT 4,B	13–10
CB 61	BIT 4,C	BIT 4,C	BIT 4,C	13–10
CB 62	BIT 4,D	BIT 4,D	BIT 4,D	13–10
CB 63	BIT 4,E	BIT 4,E	BIT 4,E	13–10
CB 64	BIT 4,H	BIT 4,H	BIT 4,H	13–10
CB 65	BIT 4,L	BIT 4,L	BIT 4,L	13–10
CB 66	BIT 4,(HL)	BIT 4,M	BIT 4,M	13–10
CB 67	BIT 4,A	BIT 4,A	BIT 4,A	13–10
CB 68	BIT 5,B	BIT 5,B	BIT 5,B	13–10
CB 69	BIT 5,C	BIT 5,C	BIT 5,C	13–10
CB 6A	BIT 5,D	BIT 5,D	BIT 5,D	13–10
CB 6B	BIT 5,E	BIT 5,E	BIT 5,E	13–10
CB 6C	BIT 5,H	BIT 5,H	BIT 5,H	13–10
CB 6D	BIT 5,L	BIT 5,L	BIT 5,L	13–10
CB 6E	BIT 5,(HL)	BIT 5,M	BIT 5,M	13–10
CB 6F	BIT 5,A	BIT 5,A	BIT 5,A	13–10
CB 70	BIT 6,B	BIT 6,B	BIT 6,B	13–10
CB 71	BIT 6,C	BIT 6,C	BIT 6,C	13–10
CB 72	BIT 6,D	BIT 6,D	BIT 6,D	13–10
CB 73	BIT 6,E	BIT 6,E	BIT 6,E	13–10
CB 74	BIT 6,H	BIT 6,H	BIT 6,H	13–10
CB 75	BIT 6,L	BIT 6,L	BIT 6,L	13–10
CB 76	BIT 6,(HL)	BIT 6,M	BIT 6,M	13–10
CB 77	BIT 6,A	BIT 6,A	BIT 6,A	13–10
CB 78	BIT 7,B	BIT 7,B	BIT 7,B	13–10
CB 79	BIT 7,C	BIT 7,C	BIT 7,C	13–10
CB 7A	BIT 7,D	BIT 7,D	BIT 7,D	13–10
CB 7B	BIT 7,E	BIT 7,E	BIT 7,E	13–10
CB 7C	BIT 7,H	BIT 7,H	BIT 7,H	13–10
CB 7D	BIT 7,L	BIT 7,L	BIT 7,L	13–10
CB 7E	BIT 7,(HL)	BIT 7,M	BIT 7,M	13–10
CB 7F	BIT 7,A	BIT 7,A	BIT 7,A	13–10
CB 80	RES 0,B	RES 0,B	RES 0,B	13–77
CB 81	RES 0,C	RES 0,C	RES 0,C	13–77
CB 82	RES 0,D	RES 0,D	RES 0,D	13–77
CB 83	RES 0,E	RES 0,E	RES 0,E	13–77
CB 84	RES 0,H	RES 0,H	RES 0,H	13–77
CB 85	RES 0,L	RES 0,L	RES 0,L	13–77
CB 86	RES 0,(HL)	RES 0,M	RES 0,M	13–77
CB 87	RES 0,A	RES 0,A	RES 0,A	13–77

Disassembly	Zilog	TDL	MAC	Page
CB 88	RES 1,B	RES 1,B	RES 1,B	13–77
CB 89	RES 1,C	RES 1,C	RES 1,C	13–77
CB 8A	RES 1,D	RES 1,D	RES 1,D	13–77
CB 8B	RES 1,E	RES 1,E	RES 1,E	13–77
CB 8C	RES 1,H	RES 1,H	RES 1,H	13–77
CB 8D	RES 1,L	RES 1,L	RES 1,L	13–77
CB 8E	RES 1,(HL)	RES 1,M	RES 1,M	13–77
CB 8F	RES 1,A	RES 1,A	RES 1,A	13–77
CB 90	RES 2,B	RES 2,B	RES 2,B	13–77
CB 91	RES 2,C	RES 2,C	RES 2,C	13–77
CB 92	RES 2,D	RES 2,D	RES 2,D	13–77
CB 93	RES 2,E	RES 2,E	RES 2,E	13–77
CB 94	RES 2,H	RES 2,H	RES 2,H	13–77
CB 95	RES 2,L	RES 2,L	RES 2,L	13–77
CB 96	RES 2,(HL)	RES 2,M	RES 2,M	13–77
CB 97	RES 2,A	RES 2,A	RES 2,A	13–77
CB 98	RES 3,B	RES 3,B	RES 3,B	13–77
CB 99	RES 3,C	RES 3,C	RES 3,C	13–77
CB 9A	RES 3,D	RES 3,D	RES 3,D	13–77
CB 9B	RES 3,E	RES 3,E	RES 3,E	13–77
CB 9C	RES 3,H	RES 3,H	RES 3,H	13–77
CB 9D	RES 3,L	RES 3,L	RES 3,L	13–77
CB 9E	RES 3,(HL)	RES 3,M	RES 3,M	13–77
CB 9F	RES 3,A	RES 3,A	RES 3,A	13–77
CB A0	RES 4,B	RES 4,B	RES 4,B	13–77
CB A1	RES 4,C	RES 4,C	RES 4,C	13–77
CB A2	RES 4,D	RES 4,D	RES 4,D	13–77
CB A3	RES 4,E	RES 4,E	RES 4,E	13–77
CB A4	RES 4,H	RES 4,H	RES 4,H	13–77
CB A5	RES 4,L	RES 4,L	RES 4,L	13–77
CB A6	RES 4,(HL)	RES 4,M	RES 4,M	13–77
CB A7	RES 4,A	RES 4,A	RES 4,A	13–77
CB A8	RES 5,B	RES 5,B	RES 5,B	13–77
CB A9	RES 5,C	RES 5,C	RES 5,C	13–77
CB AA	RES 5,D	RES 5,D	RES 5,D	13–77
CB AB	RES 5,E	RES 5,E	RES 5,E	13–77
CB AC	RES 5,H	RES 5,H	RES 5,H	13–77
CB AD	RES 5,L	RES 5,L	RES 5,L	13–77
CB AE	RES 5,(HL)	RES 5,M	RES 5,M	13–77
CB AF	RES 5,A	RES 5,A	RES 5,A	13–77
CB B0	RES 6,B	RES 6,B	RES 6,B	13–77
CB B1	RES 6,C	RES 6,C	RES 6,C	13–77
CB B2	RES 6,D	RES 6,D	RES 6,D	13–77
CB B3	RES 6,E	RES 6,E	RES 6,E	13–77
CB B4	RES 6,H	RES 6,H	RES 6,H	13–77
CB B5	RES 6,L	RES 6,L	RES 6,L	13–77
CB B6	RES 6,(HL)	RES 6,M	RES 6,M	13–77
CB B7	RES 6,A	RES 6,A	RES 6,A	13–77

Disassembly	Zilog	TDL	MAC	Page
CB B8	RES 7,B	RES 7,B	RES 7,B	13–77
CB B9	RES 7,C	RES 7,C	RES 7,C	13–77
CB BA	RES 7,D	RES 7,D	RES 7,D	13–77
CB BB	RES 7,E	RES 7,E	RES 7,E	13–77
CB BC	RES 7,H	RES 7,H	RES 7,H	13–77
CB BD	RES 7,L	RES 7,L	RES 7,L	13–77
CB BE	RES 7,(HL)	RES 7,M	RES 7,M	13–77
CB BF	RES 7,A	RES 7,A	RES 7,A	13–77
CB C0	SET 0,B	SET 0,B	SET 0,B	13–102
CB C1	SET 0,C	SET 0,C	SET 0,C	13–102
CB C2	SET 0,D	SET 0,D	SET 0,D	13–102
CB C3	SET 0,E	SET 0,E	SET 0,E	13–102
CB C4	SET 0,H	SET 0,H	SET 0,H	13–102
CB C5	SET 0,L	SET 0,L	SET 0,L	13–102
CB C6	SET 0,(HL)	SET 0,M	SET 0,M	13–102
CB C7	SET 0,A	SET 0,A	SET 0,A	13–102
CB C8	SET 1,B	SET 1,B	SET 1,B	13–102
CB C9	SET 1,C	SET 1,C	SET 1,C	13–102
CB CA	SET 1,D	SET 1,D	SET 1,D	13–102
CB CB	SET 1,E	SET 1,E	SET 1,E	13–102
CB CC	SET 1,H	SET 1,H	SET 1,H	13–102
CB CD	SET 1,L	SET 1,L	SET 1,L	13–102
CB CE	SET 1,(HL)	SET 1,M	SET 1,M	13–102
CB CF	SET 1,A	SET 1,A	SET 1,A	13–102
CB D0	SET 2,B	SET 2,B	SET 2,B	13–102
CB D1	SET 2,C	SET 2,C	SET 2,C	13–102
CB D2	SET 2,D	SET 2,D	SET 2,D	13–102
CB D3	SET 2,E	SET 2,E	SET 2,E	13–102
CB D4	SET 2,H	SET 2,H	SET 2,H	13–102
CB D5	SET 2,L	SET 2,L	SET 2,L	13–102
CB D6	SET 2,(HL)	SET 2,M	SET 2,M	13–102
CB D7	SET 2,A	SET 2,A	SET 2,A	13–102
CB D8	SET 3,B	SET 3,B	SET 3,B	13–102
CB D9	SET 3,C	SET 3,C	SET 3,C	13–102
CB DA	SET 3,D	SET 3,D	SET 3,D	13–102
CB DB	SET 3,E	SET 3,E	SET 3,E	13–102
CB DC	SET 3,H	SET 3,H	SET 3,H	13–102
CB DD	SET 3,L	SET 3,L	SET 3,L	13–102
CB DE	SET 3,(HL)	SET 3,M	SET 3,M	13–102
CB DF	SET 3,A	SET 3,A	SET 3,A	13–102
CB E0	SET 4,B	SET 4,B	SET 4,B	13–102
CB E1	SET 4,C	SET 4,C	SET 4,C	13–102
CB E2	SET 4,D	SET 4,D	SET 4,D	13–102
CB E3	SET 4,E	SET 4,E	SET 4,E	13–102
CB E4	SET 4,H	SET 4,H	SET 4,H	13–102
CB E5	SET 4,L	SET 4,L	SET 4,L	13–102
CB E6	SET 4,(HL)	SET 4,M	SET 4,M	13–102
CB E7	SET 4,A	SET 4,A	SET 4,A	13–102

Disassembly	Zilog	TDL	MAC	Page
CB E8	SET 5,B	SET 5,B	SET 5,B	13–102
CB E9	SET 5,C	SET 5,C	SET 5,C	13–102
CB EA	SET 5,D	SET 5,D	SET 5,D	13–102
CB EB	SET 5,E	SET 5,E	SET 5,E	13–102
CB EC	SET 5,H	SET 5,H	SET 5,H	13–102
CB ED	SET 5,L	SET 5,L	SET 5,L	13–102
CB EE	SET 5,(HL)	SET 5,M	SET 5,M	13–102
CB EF	SET 5,A	SET 5,A	SET 5,A	13–102
CB F0	SET 6,B	SET 6,B	SET 6,B	13–102
CB F1	SET 6,C	SET 6,C	SET 6,C	13–102
CB F2	SET 6,D	SET 6,D	SET 6,D	13–102
CB F3	SET 6,E	SET 6,E	SET 6,E	13–102
CB F4	SET 6,H	SET 6,H	SET 6,H	13–102
CB F5	SET 6,L	SET 6,L	SET 6,L	13–102
CB F6	SET 6,(HL)	SET 6,M	SET 6,M	13–102
CB F7	SET 6,A	SET 6,A	SET 6,A	13–102
CB F8	SET 7,B	SET 7,B	SET 7,B	13–102
CB F9	SET 7,C	SET 7,C	SET 7,C	13–102
CB FA	SET 7,D	SET 7,D	SET 7,D	13–102
CB FB	SET 7,E	SET 7,E	SET 7,E	13–102
CB FC	SET 7,H	SET 7,H	SET 7,H	13–102
CB FD	SET 7,L	SET 7,L	SET 7,L	13–102
CB FE	SET 7,(HL)	SET 7,M	SET 7,M	13–102
CB FF	SET 7,A	SET 7,A	SET 7,A	13–102
CC al ah	CALL Z,addr	* CZ addr	CZ addr	13–13
CD al ah	CALL addr	* CALL addr	CALL addr	13–13
CE ib	ADC A,immed	* ACI immed	ACI immed	13–4
CF	RST 08	* RST 1	RST 1	13–98
D0	RET NC	* RNC	RNC	13–80
D1	POP DE	* POP D	POP D	13–75
D2 al ah	JP NC,addr	* JNC addr	JNC addr	13–44
D3 pb	OUT (port),A	* OUT port	OUT port	13–70
D4 al ah	CALL NC,addr	* CNC addr	CNC addr	13–13
D5	PUSH DE	* PUSH D	PUSH D	13–76
D6 ib	SUB immed	* SUI immed	SUI immed	13–112
D7	RST 10	* RST 2	RST 2	13–98
D8	RET C	* RC	RC	13–80
D9	EXX	EXX	EXX	13–31
DA al ah	JP C,addr	* JC addr	JC addr	13–44
DB pb	IN A,(port)	* IN port	IN port	13–35
DC al ah	CALL C,addr	* CC addr	CC addr	13–13
DD 09	ADD IX,BC	DADX B	DADX B	13–6
DD 19	ADD IX,DE	DADX D	DADX D	13–6
DD 21 il ih	LD IX,imwrd	LXI X,imwrd	LXIX imwrd	13–48
DD 22 al ah	LD (addr),IX	SIXD addr	SIXD addr	13–48
DD 23	INC IX	INX X	INXX	13–38
DD 29	ADD IX,HL	DADX H	DADX H	13–6
DD 2A al ah	LD IX,(addr)	LIXD addr	LIXD addr	13–48
DD 2B	DEC IX	DCX X	DCXX	13–25

Disassembly	Zilog	TDL	MAC	Page
DD 34 db	INC (IX+d)	INR d(X)	INRX d	13–38
DD 35 db	DEC (IX+d)	DCR d(X)	DCRX d	13–25
DD 36 db ib	LD (IX+d),immed	MVI d(X),immed	MVIX d,immed	13–48
DD 39	ADD IX,SP	DADX SP	DADX SP	13–6
DD 46 db	LD B,(IX+d)	MOV B,d(X)	LDX B,d	13–48
DD 4E db	LD C,(IX+d)	MOV C,d(X)	LDX C,d	13–48
DD 56 db	LD D,(IX+d)	MOV D,d(X)	LDX D,d	13–48
DD 5E db	LD E,(IX+d)	MOV E,d(X)	LDX E,d	13–48
DD 66 db	LD H,(IX+d)	MOV H,d(X)	LDX H,d	13–48
DD 6E db	LD L,(IX+d)	MOV L,d(X)	LDX L,d	13–48
DD 70 db	LD (IX+d),B	MOV d(X),B	STX B,d	13–48
DD 71 db	LD (IX+d),C	MOV d(X),C	STX C,d	13–48
DD 72 db	LD (IX+d),D	MOV d(X),D	STX D,d	13–48
DD 73 db	LD (IX+d),E	MOV d(X),E	STX E,d	13–48
DD 74 db	LD (IX+d),H	MOV d(X),H	STX H,d	13–48
DD 75 db	LD (IX+d),L	MOV d(X),L	STX L,d	13–48
DD 77 db	LD (IX+d),A	MOV d(X),A	STX A,d	13–48
DD 7E db	LD A,(IX+d)	MOV A,d(X)	LDX A,d	13–48
DD 86 db	ADD A,(IX+d)	ADD d(X)	ADDX d	13–6
DD 8E db	ADC A,(IX+d)	ADC d(X)	ADCX d	13–4
DD 96	SUB (IX+d)	SUB d(X)	SUBX d	13–112
DD 9E	SBC A,(IX+d)	SBB d(X)	SBBX d	13–99
DD A6 db	AND (IX+d)	ANA d(X)	ANAX d	13–8
DD AE db	XOR (IX+d)	XRA d(X)	XRAX d	13–117
DD B6 db	OR (IX+d)	ORA d(X)	ORAX d	13–62
DD BE db	CP (IX+d)	CMP d(X)	CMPX d	13–16
DD CB db 06	RLC (IX+d)	RLCR d(X)	RLCX d	13–87
DD CB db 0E	RRC (IX+d)	RRCR d(X)	RRCX d	13–94
DD CB db 16	RL (IX+d)	RALR d(X)	RALX d	13–84
DD CB db 1E	RR (IX+d)	RARR d(X)	RARX d	13–91
DD CB db 26	SLA (IX+d)	SLAR d(X)	SLAX d	13–105
DD CB db 2E	SRA (IX+d)	SRAR d(X)	SRAX d	13–108
DD CB db 3E	SRL (IX+d)	SRLR d(X)	SRLX d	13–110
DD CB db 46	BIT 0,(IX+d)	BIT 0,d(X)	BITX 0,d	13–10
DD CB db 4E	BIT 1,(IX+d)	BIT 1,d(X)	BITX 1,d	13–10
DD CB db 56	BIT 2,(IX+d)	BIT 2,d(X)	BITX 2,d	13–10
DD CB db 5E	BIT 3,(IX+d)	BIT 3,d(X)	BITX 3,d	13–10
DD CB db 66	BIT 4,(IX+d)	BIT 4,d(X)	BITX 4,d	13–10
DD CB db 6E	BIT 5,(IX+d)	BIT 5,d(X)	BITX 5,d	13–10
DD CB db 76	BIT 6,(IX+d)	BIT 6,d(X)	BITX 6,d	13–10
DD CB db 7E	BIT 7,(IX+d)	BIT 7,d(X)	BITX 7,d	13–10
DD CB db 86	RES 0,(IX+d)	RES 0,d(X)	RESX 0,d	13–77
DD CB db 8E	RES 1,(IX+d)	RES 1,d(X)	RESX 1,d	13–77
DD CB db 96	RES 2,(IX+d)	RES 2,d(X)	RESX 2,d	13–77
DD CB db 9E	RES 3,(IX+d)	RES 3,d(X)	RESX 3,d	13–77
DD CB db A6	RES 4,(IX+d)	RES 4,d(X)	RESX 4,d	13–77
DD CB db AE	RES 5,(IX+d)	RES 5,d(X)	RESX 5,d	13–77
DD CB db B6	RES 6,(IX+d)	RES 6,d(X)	RESX 6,d	13–77
DD CB db BE	RES 7,(IX+d)	RES 7,d(X)	RESX 7,d	13–77

Disassembly	Zilog	TDL	MAC	Page
DD CB db C6	SET 0,(IX+d)	SET 0,d(X)	SETX 0,d	13–102
DD CB db CE	SET 1,(IX+d)	SET 1,d(X)	SETX 1,d	13–102
DD CB db D6	SET 2,(IX+d)	SET 2,d(X)	SETX 2,d	13–102
DD CB db DE	SET 3,(IX+d)	SET 3,d(X)	SETX 3,d	13–102
DD CB db E6	SET 4,(IX+d)	SET 4,d(X)	SETX 4,d	13–102
DD CB db EE	SET 5,(IX+d)	SET 5,d(X)	SETX 5,d	13–102
DD CB db F6	SET 6,(IX+d)	SET 6,d(X)	SETX 6,d	13–102
DD CB db FE	SET 7,(IX+d)	SET 7,d(X)	SETX 7,d	13–102
DD E1	POP IX	POP X	POPX	13–75
DD E3	EX (SP),IX	XTIX	XTIX	13–30
DD E5	PUSH IX	PUSH X	PUSHX	13–76
DD E9	JP (IX)	PCIX	PCIX	13–44
DD F9	LD SP,IX	SPIX	SPIX	13–48
DE ib	SBC A,immed	* SBI immed	SBI immed	13–99
DF	RST 18	* RST 3	RST 3	13–98
E0	RET PO	* RPO	RPO	13–80
E1	POP HL	* POP H	POP H	13–75
E2 al ah	JP PO,addr	* JPO addr	JPO addr	13–44
E3	EX (SP),HL	* XTHL	XTHL	13–30
E4 al ah	CALL PO,addr	* CPO addr	CPO addr	13–13
E5	PUSH HL	* PUSH H	PUSH H	13–76
E6 ib	AND immed	* ANI immed	ANI immed	13–8
E7	RST 20	* RST 4	RST 4	13–98
E8	RET PE	* RPE	RPE	13–80
E9	JP (HL)	* PCHL	PCHL	13–44
EA al ah	JP PE,addr	* JPE addr	JPE addr	13–44
EB	EX DE,HL	* XCHG	XCHG	13–30
EC al ah	CALL PE,addr	* CPE addr	CPE addr	13–13
ED 00 pb	~ IN0 B,(port)	—	—	13–37
ED 01	~ OUT0 (port),B	—	—	13–72
ED 04	~ TST B	—	—	13–114
ED 08 pb	~ IN0 C,(port)	—	—	13–37
ED 09	~ OUT0 (port),C	—	—	13–72
ED 0C	~ TST C	—	—	13–114
ED 10 pb	~ IN0 D,(port)	—	—	13–37
ED 11	~ OUT0 (port),D	—	—	13–72
ED 14	~ TST D	—	—	13–114
ED 18 pb	~ IN0 E,(port)	—	—	13–37
ED 19	~ OUT0 (port),E	—	—	13–72
ED 1C	~ TST E	—	—	13–114
ED 20 pb	~ IN0 H,(port)	—	—	13–37
ED 21	~ OUT0 (port),H	—	—	13–72
ED 24	~ TST H	—	—	13–114
ED 28 pb	~ IN0 L,(port)	—	—	13–37
ED 29	~ OUT0 (port),L	—	—	13–72
ED 2C	~ TST L	—	—	13–114
ED 30 pb	~ —	—	—	13–37
ED 34	~ TST (HL)	—	—	13–114
ED 38 pb	~ IN0 A,(port)	—	—	13–37
ED 39	~ OUT0 (port),A	—	—	13–72
ED 3C	~ TST A	—	—	13–114

Disassembly	Zilog	TDL	MAC	Page
ED 40	IN B,(C)	INP B	INP B	13–35
ED 41	OUT (C),B	OUTP B	OUTP B	13–70
ED 42	SBC HL,BC	DSBC B	DSBC B	13–99
ED 43 al ah	LD (addr),BC	SBCD addr	SBCD addr	13–48
ED 44	NEG	NEG	NEG	13–60
ED 45	RETN	RETN	RETN	13–83
ED 46	IM 0	IM0	IM0	13–33
ED 47	LD I,A	STAI	STAI	13–48
ED 48	IN C,(C)	INP C	INP C	13–35
ED 49	OUT (C),C	OUTP C	OUTP C	13–70
ED 4A	ADC HL,BC	DADC B	DADC B	13–4
ED 4B al ah	LD BC,(addr)	LBCD addr	LBCD addr	13–48
ED 4C	~ MLT BC	—	—	13–59
ED 4D	RETI	RETI	RETI	13–82
ED 4F	LD R,A	STAR	STAR	13–48
ED 50	IN D,(C)	INP D	INP D	13–35
ED 51	OUT (C),D	OUTP D	OUTP D	13–70
ED 52	SBC HL,DE	DSBC D	DSBC D	13–99
ED 53 al ah	LD (addr),DE	SDED addr	SDED addr	13–48
ED 56	IM 1	IM1	IM1	13–33
ED 57	LD A,I	LDAI	LDAI	13–48
ED 58	IN E,(C)	INP E	INP E	13–35
ED 59	OUT (C),E	OUTP E	OUTP E	13–70
ED 5A	ADC HL,DE	DADC D	DADC D	13–4
ED 5B al ah	LD DE,(addr)	LDED addr	LDED addr	13–48
ED 5C	~ MLT DE	—	—	13–59
ED 5E	IM 2	IM2	IM2	13–33
ED 5F	LD A,R	LDAR	LDAR	13–48
ED 60	IN H,(C)	INP H	INP H	13–35
ED 61	OUT (C),H	OUTP H	OUTP H	13–70
ED 62	SBC HL,HL	DSBC H	DSBC H	13–99
ED 63 al ah	LD (addr),HL	SHLD addr	SHLD addr	13–48
ED 64 ib	~ TST immed	—	—	13–114
ED 67	RRD	RRD	RRD	13–97
ED 68	IN L,(C)	INP L	INP L	13–35
ED 69	OUT (C),L	OUTP L	OUTP L	13–70
ED 6A	ADC HL,HL	DADC H	DADC H	13–4
ED 6B al ah	LD HL,(addr)	LHLD addr	LHLD addr	13–48
ED 6C	~ MLT HL	—	—	13–59
ED 6F	RLD	RLD	RLD	13–90
ED 70	—	—	—	13–35
ED 72	SBC HL,SP	DSBC SP	DSBC SP	13–99
ED 73 al ah	LD (addr),SP	SSPD addr	SSPD addr	13–48
ED 74 ib	~ TSTIO immed	—	—	13–116
ED 76	~ SLP	—	—	13–107
ED 78	IN A,(C)	INP A	INP A	13–35
ED 79	OUT (C),A	OUTP A	OUTP A	13–70
ED 7A	ADC HL,SP	DADC SP	DADC SP	13–4
ED 7B al ah	LD SP,(addr)	LSPD addr	LSPD addr	13–48
ED 7C	~ MLT SP	—	—	13–59

Disassembly	Zilog	TDL	MAC	Page
ED 83	~ OTIM	—	—	13–67
ED 8B	~ OTDM	—	—	13–64
ED 93	~ OTIMR	—	—	13–68
ED 9B	~ OTDMR	—	—	13–65
ED A0	LDI	LDI	LDI	13–57
ED A1	CPI	CCI	CCI	13–20
ED A2	INI	INI	INI	13–42
ED A3	OUTI	OUTI	OUTI	13–74
ED A8	LDD	LDD	LDD	13–55
ED A9	CPD	CCD	CCD	13–18
ED AA	IND	IND	IND	13–40
ED AB	OUTD	OUTD	OUTD	13–73
ED B0	LDIR	LDIR	LDIR	13–58
ED B1	CPIR	CCIR	CCIR	13–21
ED B2	INIR	INIR	INIR	13–43
ED B3	OTIR	OTIR	OTIR	13–69
ED B8	LDDR	LDDR	LDDR	13–56
ED B9	CPDR	CCDR	CCDR	13–19
ED BA	INDR	INDR	INDR	13–41
ED BB	OTDR	OTDR	OTDR	13–66
EE ib	XOR immed	* XRI immed	XRI immed	13–117
EF	RST 28	* RST 5	RST 5	13–98
F0	RET P	* RP	RP	13–80
F1	POP AF	* POP PSW	POP PSW	13–75
F2 al ah	JP P,addr	* JP addr	JP addr	13–44
F3	DI	* DI	DI	13–27
F4 al ah	CALL P,addr	* CP addr	CP addr	13–13
F5	PUSH AF	* PUSH PSW	PUSH PSW	13–76
F6 ib	OR immed	* ORI immed	ORI immed	13–62
F7	RST 30	* RST 6	RST 6	13–98
F8	RET M	* RM	RM	13–80
F9	LD SP,HL	* SPHL	SPHL	13–48
FA al ah	JP M,addr	* JM addr	JM addr	13–44
FB	EI	* EI	EI	13–29
FC al ah	CALL M,addr	* CM addr	CM addr	13–13
FD 09	ADD IY,BC	DADY B	DADY B	13–6
FD 19	ADD IY,DE	DADY D	DADY D	13–6
FD 21 il ih	LD IY,imwrd	LXI Y,imwrd	LXIY imwrd	13–48
FD 22 al ah	LD (addr),IY	SIYD addr	SIYD addr	13–48
FD 23	INC IY	INX Y	INXY	13–38
FD 29	ADD IY,HL	DADY H	DADY H	13–6
FD 2A al ah	LD IY,(addr)	LIYD addr	LIYD addr	13–48
FD 2B	DEC IY	DCX Y	DCXY	13–25
FD 34 db	INC (IY+d)	INR d(Y)	INRY d	13–38
FD 35 db	DEC (IY+d)	DCR d(Y)	DCRY d	13–25
FD 36 db ib	LD (IY+d),immed	MVI d(Y),immed	MVIY d,immed	13–48
FD 39	ADD IY,SP	DADY SP	DADY SP	13–6
FD 46 db	LD B,(IY+d)	MOV B,d(Y)	LDY B,d	13–48
FD 4E db	LD C,(IY+d)	MOV C,d(Y)	LDY C,d	13–48

Disassembly	Zilog	TDL	MAC	Page
FD 56 db	LD D,(IY+d)	MOV D,d(Y)	LDY D,d	13–48
FD 5E db	LD E,(IY+d)	MOV E,d(Y)	LDY E,d	13–48
FD 66 db	LD H,(IY+d)	MOV H,d(Y)	LDY H,d	13–48
FD 6E db	LD L,(IY+d)	MOV L,d(Y)	LDY L,d	13–48
FD 70 db	LD (IY+d),B	MOV d(Y),B	STY B,d	13–48
FD 71 db	LD (IY+d),C	MOV d(Y),C	STY C,d	13–48
FD 72 db	LD (IY+d),D	MOV d(Y),D	STY D,d	13–48
FD 73 db	LD (IY+d),E	MOV d(Y),E	STY E,d	13–48
FD 74 db	LD (IY+d),H	MOV d(Y),H	STY H,d	13–48
FD 75 db	LD (IY+d),L	MOV d(Y),L	STY L,d	13–48
FD 77 db	LD (IY+d),A	MOV d(Y),A	STY A,d	13–48
FD 7E db	LD A,(IY+d)	MOV A,d(Y)	LDY A,d	13–48
FD 86 db	ADD A,(IY+d)	ADD d(Y)	ADDY d	13–6
FD 8E db	ADC A,(IY+d)	ADC d(Y)	ADCY d	13–4
FD 96	SUB (IY+d)	SUB d(Y)	SUBY d	13–112
FD 9E	SBC A,(IY+d)	SBB d(Y)	SBBY d	13–99
FD A6 db	AND (IY+d)	ANA d(Y)	ANAY d	13–8
FD AE db	XOR (IY+d)	XRA d(Y)	XRAY d	13–117
FD B6 db	OR (IY+d)	ORA d(Y)	ORAY d	13–62
FD BE db	CP (IY+d)	CMP d(Y)	CMPY d	13–16
FD CB db 06	RLC (IY+d)	RLCR d(Y)	RLCY d	13–87
FD CB db 0E	RRC (IY+d)	RRCR d(Y)	RRCY d	13–94
FD CB db 16	RL (IY+d)	RALR d(Y)	RALY d	13–84
FD CB db 1E	RR (IY+d)	RARR d(Y)	RARY d	13–91
FD CB db 26	SLA (IY+d)	SLAR d(Y)	SLAY d	13–105
FD CB db 2E	SRA (IY+d)	SRAR d(Y)	SRAY d	13–108
FD CB db 3E	SRL (IY+d)	SRLR d(Y)	SRLY d	13–110
FD CB db 46	BIT 0,(IY+d)	BIT 0,d(Y)	BITY 0,d	13–10
FD CB db 4E	BIT 1,(IY+d)	BIT 1,d(Y)	BITY 1,d	13–10
FD CB db 56	BIT 2,(IY+d)	BIT 2,d(Y)	BITY 2,d	13–10
FD CB db 5E	BIT 3,(IY+d)	BIT 3,d(Y)	BITY 3,d	13–10
FD CB db 66	BIT 4,(IY+d)	BIT 4,d(Y)	BITY 4,d	13–10
FD CB db 6E	BIT 5,(IY+d)	BIT 5,d(Y)	BITY 5,d	13–10
FD CB db 76	BIT 6,(IY+d)	BIT 6,d(Y)	BITY 6,d	13–10
FD CB db 7E	BIT 7,(IY+d)	BIT 7,d(Y)	BITY 7,d	13–10
FD CB db 86	RES 0,(IY+d)	RES 0,d(Y)	RESY 0,d	13–77
FD CB db 8E	RES 1,(IY+d)	RES 1,d(Y)	RESY 1,d	13–77
FD CB db 96	RES 2,(IY+d)	RES 2,d(Y)	RESY 2,d	13–77
FD CB db 9E	RES 3,(IY+d)	RES 3,d(Y)	RESY 3,d	13–77
FD CB db A6	RES 4,(IY+d)	RES 4,d(Y)	RESY 4,d	13–77
FD CB db AE	RES 5,(IY+d)	RES 5,d(Y)	RESY 5,d	13–77
FD CB db B6	RES 6,(IY+d)	RES 6,d(Y)	RESY 6,d	13–77
FD CB db BE	RES 7,(IY+d)	RES 7,d(Y)	RESY 7,d	13–77
FD CB db C6	SET 0,(IY+d)	SET 0,d(Y)	SETY 0,d	13–102
FD CB db CE	SET 1,(IY+d)	SET 1,d(Y)	SETY 1,d	13–102
FD CB db D6	SET 2,(IY+d)	SET 2,d(Y)	SETY 2,d	13–102
FD CB db DE	SET 3,(IY+d)	SET 3,d(Y)	SETY 3,d	13–102
FD CB db E6	SET 4,(IY+d)	SET 4,d(Y)	SETY 4,d	13–102
FD CB db EE	SET 5,(IY+d)	SET 5,d(Y)	SETY 5,d	13–102
FD CB db F6	SET 6,(IY+d)	SET 6,d(Y)	SETY 6,d	13–102
FD CB db FE	SET 7,(IY+d)	SET 7,d(Y)	SETY 7,d	13–102

Disassembly	Zilog	TDL	MAC	Page
FD E1	POP IY	POP Y	POPY	13–75
FD E3	EX (SP),IY	XTIY	XTIY	13–30
FD E5	PUSH IY	PUSH Y	PUSHY	13–76
FD E9	JP (IY)	PCIY	PCIY	13–44
FD F9	LD SP,IY	SPIY	SPIY	13–48
FE ib	CP immed	* CPI immed	CPI immed	13–16
FF	RST 38	* RST 7	RST 7	13–98

Cross Reference by Zilog Mnemonic

Following is an op-code cross reference from Zilog to TDL and MAC mnemonics and machine codes. This table can be of great assistance in translation. The page given contains the detailed instructions for the op-codes.

Those Zilog mnemonics indicated by a "~" and missing in TDL and MAC mnemonics are unique to the Z-180 processor.

Those TDL mnemonics that are indicated by an "*" are the Intel 8080 processor subset, and usually use the TDL or MAC mnemonics (identical for all 8080 instructions).

Zilog	TDL	MAC	Disassembly	Page
ADC A,(HL)	* ADC M	ADC M	8E	13–4
ADC A,(IX+d)	ADC d(X)	ADCX d	DD 8E db	13–4
ADC A,(IY+d)	ADC d(Y)	ADCY d	FD 8E db	13–4
ADC A,A	* ADC A	ADC A	8F	13–4
ADC A,B	* ADC B	ADC B	88	13–4
ADC A,C	* ADC C	ADC C	89	13–4
ADC A,D	* ADC D	ADC D	8A	13–4
ADC A,E	* ADC E	ADC E	8B	13–4
ADC A,H	* ADC H	ADC H	8C	13–4
ADC A,L	* ADC L	ADC L	8D	13–4
ADC A,immed	* ACI immed	ACI immed	CE ib	13–4
ADC HL,BC	DADC B	DADC B	ED 4A	13–4
ADC HL,DE	DADC D	DADC D	ED 5A	13–4
ADC HL,HL	DADC H	DADC H	ED 6A	13–4
ADC HL,SP	DADC SP	DADC SP	ED 7A	13–4
ADD A,(HL)	* ADD M	ADD M	86	13–6
ADD A,(IX+d)	ADD d(X)	ADDX d	DD 86 db	13–6
ADD A,(IY+d)	ADD d(Y)	ADDY d	FD 86 db	13–6
ADD A,A	* ADD A	ADD A	87	13–6
ADD A,B	* ADD B	ADD B	80	13–6
ADD A,C	* ADD C	ADD C	81	13–6
ADD A,D	* ADD D	ADD D	82	13–6
ADD A,E	* ADD E	ADD E	83	13–6
ADD A,H	* ADD H	ADD H	84	13–6
ADD A,L	* ADD L	ADD L	85	13–6
ADD A,immed	* ADI immed	ADI immed	C6 ib	13–6
ADD HL,BC	* DAD B	DAD B	09	13–6
ADD HL,DE	* DAD D	DAD D	19	13–6
ADD HL,HL	* DAD H	DAD H	29	13–6
ADD HL,SP	* DAD SP	DAD SP	39	13–6
ADD IX,BC	DADX B	DADX B	DD 09	13–6
ADD IX,DE	DADX D	DADX D	DD 19	13–6
ADD IX,HL	DADX H	DADX H	DD 29	13–6
ADD IX,SP	DADX SP	DADX SP	DD 39	13–6
ADD IY,BC	DADY B	DADY B	FD 09	13–6
ADD IY,DE	DADY D	DADY D	FD 19	13–6
ADD IY,HL	DADY H	DADY H	FD 29	13–6
ADD IY,SP	DADY SP	DADY SP	FD 39	13–6
AND (HL)	* ANA M	ANA M	A6	13–8
AND (IX+d)	ANA d(X)	ANAX d	DD A6 db	13–8
AND (IY+d)	ANA d(Y)	ANAY d	FD A6 db	13–8

Zilog	TDL	MAC	Disassembly	Page
AND A	* ANA A	ANA A	A7	13–8
AND B	* ANA B	ANA B	A0	13–8
AND C	* ANA C	ANA C	A1	13–8
AND D	* ANA D	ANA D	A2	13–8
AND E	* ANA E	ANA E	A3	13–8
AND H	* ANA H	ANA H	A4	13–8
AND L	* ANA L	ANA L	A5	13–8
AND immed	* ANI immed	ANI immed	E6 ib	13–8
BIT 0,(HL)	BIT 0,M	BIT 0,M	CB 46	13–10
BIT 0,(IX+d)	BIT 0,d(X)	BITX 0,d	DD CB db 46	13–10
BIT 0,(IY+d)	BIT 0,d(Y)	BITY 0,d	FD CB db 46	13–10
BIT 0,A	BIT 0,A	BIT 0,A	CB 47	13–10
BIT 0,B	BIT 0,B	BIT 0,B	CB 40	13–10
BIT 0,C	BIT 0,C	BIT 0,C	CB 41	13–10
BIT 0,D	BIT 0,D	BIT 0,D	CB 42	13–10
BIT 0,E	BIT 0,E	BIT 0,E	CB 43	13–10
BIT 0,H	BIT 0,H	BIT 0,H	CB 44	13–10
BIT 0,L	BIT 0,L	BIT 0,L	CB 45	13–10
BIT 1,(HL)	BIT 1,M	BIT 1,M	CB 4E	13–10
BIT 1,(IX+d)	BIT 1,d(X)	BITX 1,d	DD CB db 4E	13–10
BIT 1,(IY+d)	BIT 1,d(Y)	BITY 1,d	FD CB db 4E	13–10
BIT 1,A	BIT 1,A	BIT 1,A	CB 4F	13–10
BIT 1,B	BIT 1,B	BIT 1,B	CB 48	13–10
BIT 1,C	BIT 1,C	BIT 1,C	CB 49	13–10
BIT 1,D	BIT 1,D	BIT 1,D	CB 4A	13–10
BIT 1,E	BIT 1,E	BIT 1,E	CB 4B	13–10
BIT 1,H	BIT 1,H	BIT 1,H	CB 4C	13–10
BIT 1,L	BIT 1,L	BIT 1,L	CB 4D	13–10
BIT 2,(HL)	BIT 2,M	BIT 2,M	CB 56	13–10
BIT 2,(IX+d)	BIT 2,d(X)	BITX 2,d	DD CB db 56	13–10
BIT 2,(IY+d)	BIT 2,d(Y)	BITY 2,d	FD CB db 56	13–10
BIT 2,A	BIT 2,A	BIT 2,A	CB 57	13–10
BIT 2,B	BIT 2,B	BIT 2,B	CB 50	13–10
BIT 2,C	BIT 2,C	BIT 2,C	CB 51	13–10
BIT 2,D	BIT 2,D	BIT 2,D	CB 52	13–10
BIT 2,E	BIT 2,E	BIT 2,E	CB 53	13–10
BIT 2,H	BIT 2,H	BIT 2,H	CB 54	13–10
BIT 2,L	BIT 2,L	BIT 2,L	CB 55	13–10
BIT 3,(HL)	BIT 3,M	BIT 3,M	CB 5E	13–10
BIT 3,(IX+d)	BIT 3,d(X)	BITX 3,d	DD CB db 5E	13–10
BIT 3,(IY+d)	BIT 3,d(Y)	BITY 3,d	FD CB db 5E	13–10
BIT 3,A	BIT 3,A	BIT 3,A	CB 5F	13–10
BIT 3,B	BIT 3,B	BIT 3,B	CB 58	13–10
BIT 3,C	BIT 3,C	BIT 3,C	CB 59	13–10
BIT 3,D	BIT 3,D	BIT 3,D	CB 5A	13–10
BIT 3,E	BIT 3,E	BIT 3,E	CB 5B	13–10
BIT 3,H	BIT 3,H	BIT 3,H	CB 5C	13–10
BIT 3,L	BIT 3,L	BIT 3,L	CB 5D	13–10
BIT 4,(HL)	BIT 4,M	BIT 4,M	CB 66	13–10
BIT 4,(IX+d)	BIT 4,d(X)	BITX 4,d	DD CB db 66	13–10
BIT 4,(IY+d)	BIT 4,d(Y)	BITY 4,d	FD CB db 66	13–10

Zilog	TDL	MAC	Disassembly	Page
BIT 4,A	BIT 4,A	BIT 4,A	CB 67	13–10
BIT 4,B	BIT 4,B	BIT 4,B	CB 60	13–10
BIT 4,C	BIT 4,C	BIT 4,C	CB 61	13–10
BIT 4,D	BIT 4,D	BIT 4,D	CB 62	13–10
BIT 4,E	BIT 4,E	BIT 4,E	CB 63	13–10
BIT 4,H	BIT 4,H	BIT 4,H	CB 64	13–10
BIT 4,L	BIT 4,L	BIT 4,L	CB 65	13–10
BIT 5,(HL)	BIT 5,M	BIT 5,M	CB 6E	13–10
BIT 5,(IX+d)	BIT 5,d(X)	BITX 5,d	DD CB db 6E	13–10
BIT 5,(IY+d)	BIT 5,d(Y)	BITY 5,d	FD CB db 6E	13–10
BIT 5,A	BIT 5,A	BIT 5,A	CB 6F	13–10
BIT 5,B	BIT 5,B	BIT 5,B	CB 68	13–10
BIT 5,C	BIT 5,C	BIT 5,C	CB 69	13–10
BIT 5,D	BIT 5,D	BIT 5,D	CB 6A	13–10
BIT 5,E	BIT 5,E	BIT 5,E	CB 6B	13–10
BIT 5,H	BIT 5,H	BIT 5,H	CB 6C	13–10
BIT 5,L	BIT 5,L	BIT 5,L	CB 6D	13–10
BIT 6,(HL)	BIT 6,M	BIT 6,M	CB 76	13–10
BIT 6,(IX+d)	BIT 6,d(X)	BITX 6,d	DD CB db 76	13–10
BIT 6,(IY+d)	BIT 6,d(Y)	BITY 6,d	FD CB db 76	13–10
BIT 6,A	BIT 6,A	BIT 6,A	CB 77	13–10
BIT 6,B	BIT 6,B	BIT 6,B	CB 70	13–10
BIT 6,C	BIT 6,C	BIT 6,C	CB 71	13–10
BIT 6,D	BIT 6,D	BIT 6,D	CB 72	13–10
BIT 6,E	BIT 6,E	BIT 6,E	CB 73	13–10
BIT 6,H	BIT 6,H	BIT 6,H	CB 74	13–10
BIT 6,L	BIT 6,L	BIT 6,L	CB 75	13–10
BIT 7,(HL)	BIT 7,M	BIT 7,M	CB 7E	13–10
BIT 7,(IX+d)	BIT 7,d(X)	BITX 7,d	DD CB db 7E	13–10
BIT 7,(IY+d)	BIT 7,d(Y)	BITY 7,d	FD CB db 7E	13–10
BIT 7,A	BIT 7,A	BIT 7,A	CB 7F	13–10
BIT 7,B	BIT 7,B	BIT 7,B	CB 78	13–10
BIT 7,C	BIT 7,C	BIT 7,C	CB 79	13–10
BIT 7,D	BIT 7,D	BIT 7,D	CB 7A	13–10
BIT 7,E	BIT 7,E	BIT 7,E	CB 7B	13–10
BIT 7,H	BIT 7,H	BIT 7,H	CB 7C	13–10
BIT 7,L	BIT 7,L	BIT 7,L	CB 7D	13–10
CALL addr	* CALL addr	CALL addr	CD al ah	13–13
CALL C,addr	* CC addr	CC addr	DC al ah	13–13
CALL M,addr	* CM addr	CM addr	FC al ah	13–13
CALL NC,addr	* CNC addr	CNC addr	D4 al ah	13–13
CALL NZ,addr	* CNZ addr	CNZ addr	C4 al ah	13–13
CALL P,addr	* CP addr	CP addr	F4 al ah	13–13
CALL PE,addr	* CPE addr	CPE addr	EC al ah	13–13
CALL PO,addr	* CPO addr	CPO addr	E4 al ah	13–13
CALL Z,addr	* CZ addr	CZ addr	CC al ah	13–13
CCF	* CMC	CMC	3F	13–15
CP (HL)	* CMP M	CMP M	BE	13–16
CP (IX+d)	CMP d(X)	CMPX d	DD BE db	13–16
CP (IY+d)	CMP d(Y)	CMPY d	FD BE db	13–16

Zilog		TDL	MAC	Disassembly	Page
CP A	*	CMP A	CMP A	BF	13–16
CP B	*	CMP B	CMP B	B8	13–16
CP C	*	CMP C	CMP C	B9	13–16
CP D	*	CMP D	CMP D	BA	13–16
CP E	*	CMP E	CMP E	BB	13–16
CP H	*	CMP H	CMP H	BC	13–16
CP L	*	CMP L	CMP L	BD	13–16
CP immed	*	CPI immed	CPI immed	FE ib	13–16
CPD		CCD	CCD	ED A9	13–18
CPDR		CCDR	CCDR	ED B9	13–19
CPI		CCI	CCI	ED A1	13–20
CPIR		CCIR	CCIR	ED B1	13–21
CPL	*	CMA	CMA	2F	13–22
DAA	*	DAA	DAA	27	13–23
DEC (HL)	*	DCR M	DCR M	35	13–25
DEC (IX+d)		DCR d(X)	DCRX d	DD 35 db	13–25
DEC (IY+d)		DCR d(Y)	DCRY d	FD 35 db	13–25
DEC A	*	DCR A	DCR A	3D	13–25
DEC B	*	DCR B	DCR B	05	13–25
DEC C	*	DCR C	DCR C	0D	13–25
DEC D	*	DCR D	DCR D	15	13–25
DEC E	*	DCR E	DCR E	1D	13–25
DEC H	*	DCR H	DCR H	25	13–25
DEC L	*	DCR L	DCR L	2D	13–25
DEC BC	*	DCX B	DCX B	0B	13–25
DEC DE	*	DCX D	DCX D	1B	13–25
DEC HL	*	DCX H	DCX H	2B	13–25
DEC IX		DCX X	DCXX	DD 2B	13–25
DEC IY		DCX Y	DCXY	FD 2B	13–25
DEC SP	*	DCX SP	DCX SP	3B	13–25
DI	*	DI	DI	F3	13–27
DJNZ disp		DJNZ disp	DJNZ disp	10 ab	13–28
EI	*	EI	EI	FB	13–29
EX (SP),HL	*	XTHL	XTHL	E3	13–30
EX (SP),IX		XTIX	XTIX	DD E3	13–30
EX (SP),IY		XTIY	XTIY	FD E3	13–30
EX AF,AF'		EXAF	EXAF	08	13–30
EX DE,HL	*	XCHG	XCHG	EB	13–30
EXX		EXX	EXX	D9	13–31
HALT	*	HLT	HLT	76	13–32
IM 0		IM0	IM0	ED 46	13–33
IM 1		IM1	IM1	ED 56	13–33
IM 2		IM2	IM2	ED 5E	13–33

Zilog	TDL	MAC	Disassembly	Page
IN A,(C)	INP A	INP A	ED 78	13–35
IN B,(C)	INP B	INP B	ED 40	13–35
IN C,(C)	INP C	INP C	ED 48	13–35
IN D,(C)	INP D	INP D	ED 50	13–35
IN E,(C)	INP E	INP E	ED 58	13–35
IN H,(C)	INP H	INP H	ED 60	13–35
IN L,(C)	INP L	INP L	ED 68	13–35
IN A,(port)	* IN port	IN port	DB pb	13–35
~ IN0 A,(port)	—	—	ED 38 pb	13–37
~ IN0 B,(port)	—	—	ED 00 pb	13–37
~ IN0 C,(port)	—	—	ED 08 pb	13–37
~ IN0 D,(port)	—	—	ED 10 pb	13–37
~ IN0 E,(port)	—	—	ED 18 pb	13–37
~ IN0 H,(port)	—	—	ED 20 pb	13–37
~ IN0 L,(port)	—	—	ED 28 pb	13–37
INC (HL)	* INR M	INR M	34	13–38
INC (IX+d)	INR d(X)	INRX d	DD 34 db	13–38
INC (IY+d)	INR d(Y)	INRY d	FD 34 db	13–38
INC A	* INR A	INR A	3C	13–38
INC B	* INR B	INR B	04	13–38
INC C	* INR C	INR C	0C	13–38
INC D	* INR D	INR D	14	13–38
INC E	* INR E	INR E	1C	13–38
INC H	* INR H	INR H	24	13–38
INC L	* INR L	INR L	2C	13–38
INC BC	* INX B	INX B	03	13–38
INC DE	* INX D	INX D	13	13–38
INC HL	* INX H	INX H	23	13–38
INC IX	INX X	INXX	DD 23	13–38
INC IY	INX Y	INXY	FD 23	13–38
INC SP	* INX SP	INX SP	33	13–38
IND	IND	IND	ED AA	13–40
INDR	INDR	INDR	ED BA	13–41
INI	INI	INI	ED A2	13–42
INIR	INIR	INIR	ED B2	13–43
JP addr	* JMP addr	JMP addr	C3 al ah	13–44
JP (HL)	* PCHL	PCHL	E9	13–44
JP (IX)	PCIX	PCIX	DD E9	13–44
JP (IY)	PCIY	PCIY	FD E9	13–44
JP C,addr	* JC addr	JC addr	DA al ah	13–44
JP M,addr	* JM addr	JM addr	FA al ah	13–44
JP NC,addr	* JNC addr	JNC addr	D2 al ah	13–44
JP NZ,addr	* JNZ addr	JNZ addr	C2 al ah	13–44
JP P,addr	* JP addr	JP addr	F2 al ah	13–44
JP PE,addr	* JPE addr	JPE addr	EA al ah	13–44
JP PO,addr	* JPO addr	JPO addr	E2 al ah	13–44
JP Z,addr	* JZ addr	JZ addr	CA al ah	13–44

Zilog	TDL	MAC	Disassembly	Page
JR disp	JMPR disp	JR disp	18 ab	13–46
JR C,disp	JRC disp	JRC disp	38 ab	13–46
JR NC,disp	JRNC disp	JRNC disp	30 ab	13–46
JR NZ,disp	JRNZ disp	JRNZ disp	20 ab	13–46
JR Z,disp	JRZ disp	JRZ disp	28 ab	13–46
LD (BC),A	* STAX B	STAX B	02	13–48
LD (DE),A	* STAX D	STAX D	12	13–48
LD (HL),A	* MOV M,A	MOV M,A	77	13–48
LD (HL),B	* MOV M,B	MOV M,B	70	13–48
LD (HL),C	* MOV M,C	MOV M,C	71	13–48
LD (HL),D	* MOV M,D	MOV M,D	72	13–48
LD (HL),E	* MOV M,E	MOV M,E	73	13–48
LD (HL),H	* MOV M,H	MOV M,H	74	13–48
LD (HL),L	* MOV M,L	MOV M,L	75	13–48
LD (HL),immed	* MVI M,immed	MVI M,immed	36 ib	13–48
LD (IX+d),A	MOV d(X),A	STX A,d	DD 77 db	13–48
LD (IX+d),B	MOV d(X),B	STX B,d	DD 70 db	13–48
LD (IX+d),C	MOV d(X),C	STX C,d	DD 71 db	13–48
LD (IX+d),D	MOV d(X),D	STX D,d	DD 72 db	13–48
LD (IX+d),E	MOV d(X),E	STX E,d	DD 73 db	13–48
LD (IX+d),H	MOV d(X),H	STX H,d	DD 74 db	13–48
LD (IX+d),L	MOV d(X),L	STX L,d	DD 75 db	13–48
LD (IX+d),immed	MVI d(X),immed	MVIX d,immed	DD 36 db ib	13–48
LD (IY+d),A	MOV d(Y),A	STY A,d	FD 77 db	13–48
LD (IY+d),B	MOV d(Y),B	STY B,d	FD 70 db	13–48
LD (IY+d),C	MOV d(Y),C	STY C,d	FD 71 db	13–48
LD (IY+d),D	MOV d(Y),D	STY D,d	FD 72 db	13–48
LD (IY+d),E	MOV d(Y),E	STY E,d	FD 73 db	13–48
LD (IY+d),H	MOV d(Y),H	STY H,d	FD 74 db	13–48
LD (IY+d),L	MOV d(Y),L	STY L,d	FD 75 db	13–48
LD (IY+d),immed	MVI d(Y),immed	MVIY d,immed	FD 36 db ib	13–48
LD (addr),A	* STA addr	STA addr	32 al ah	13–48
LD (addr),BC	SBCD addr	SBCD addr	ED 43 al ah	13–48
LD (addr),DE	SDED addr	SDED addr	ED 53 al ah	13–48
LD (addr),HL	* SHLD addr	SHLD addr	22 al ah	13–48
LD (addr),IX	SIXD addr	SIXD addr	DD 22 al ah	13–48
LD (addr),IY	SIYD addr	SIYD addr	FD 22 al ah	13–48
LD (addr),SP	SSPD addr	SSPD addr	ED 73 al ah	13–48
LD A,(BC)	* LDAX B	LDAX B	0A	13–48
LD A,(DE)	* LDAX D	LDAX D	1A	13–48
LD A,(HL)	* MOV A,M	MOV A,M	7E	13–48
LD A,(IX+d)	MOV A,d(X)	LDX A,d	DD 7E db	13–48
LD A,(IY+d)	MOV A,d(Y)	LDY A,d	FD 7E db	13–48
LD A,(addr)	* LDA addr	LDA addr	3A al ah	13–48

Zilog	TDL	MAC	Disassembly	Page
LD A,A	* MOV A,A	MOV A,A	7F	13–48
LD A,B	* MOV A,B	MOV A,B	78	13–48
LD A,C	* MOV A,C	MOV A,C	79	13–48
LD A,D	* MOV A,D	MOV A,D	7A	13–48
LD A,E	* MOV A,E	MOV A,E	7B	13–48
LD A,H	* MOV A,H	MOV A,H	7C	13–48
LD A,I	LDAI	LDAI	ED 57	13–48
LD A,L	* MOV A,L	MOV A,L	7D	13–48
LD A,R	LDAR	LDAR	ED 5F	13–48
LD A,immed	* MVI A,immed	MVI A,immed	3E ib	13–48
LD B,(HL)	* MOV B,M	MOV B,M	46	13–48
LD B,(IX+d)	MOV B,d(X)	LDX B,d	DD 46 db	13–48
LD B,(IY+d)	MOV B,d(Y)	LDY B,d	FD 46 db	13–48
LD B,A	* MOV B,A	MOV B,A	47	13–48
LD B,B	* MOV B,B	MOV B,B	40	13–48
LD B,C	* MOV B,C	MOV B,C	41	13–48
LD B,D	* MOV B,D	MOV B,D	42	13–48
LD B,E	* MOV B,E	MOV B,E	43	13–48
LD B,H	* MOV B,H	MOV B,H	44	13–48
LD B,L	* MOV B,L	MOV B,L	45	13–48
LD B,immed	* MVI B,immed	MVI B,immed	06 ib	13–48
LD C,(HL)	* MOV C,M	MOV C,M	4E	13–48
LD C,(IX+d)	MOV C,d(X)	LDX C,d	DD 4E db	13–48
LD C,(IY+d)	MOV C,d(Y)	LDY C,d	FD 4E db	13–48
LD C,A	* MOV C,A	MOV C,A	4F	13–48
LD C,B	* MOV C,B	MOV C,B	48	13–48
LD C,C	* MOV C,C	MOV C,C	49	13–48
LD C,D	* MOV C,D	MOV C,D	4A	13–48
LD C,E	* MOV C,E	MOV C,E	4B	13–48
LD C,H	* MOV C,H	MOV C,H	4C	13–48
LD C,L	* MOV C,L	MOV C,L	4D	13–48
LD C,immed	* MVI C,immed	MVI C,immed	0E ib	13–48
LD D,(HL)	* MOV D,M	MOV D,M	56	13–48
LD D,(IX+d)	MOV D,d(X)	LDX D,d	DD 56 db	13–48
LD D,(IY+d)	MOV D,d(Y)	LDY D,d	FD 56 db	13–48
LD D,A	* MOV D,A	MOV D,A	57	13–48
LD D,B	* MOV D,B	MOV D,B	50	13–48
LD D,C	* MOV D,C	MOV D,C	51	13–48
LD D,D	* MOV D,D	MOV D,D	52	13–48
LD D,E	* MOV D,E	MOV D,E	53	13–48
LD D,H	* MOV D,H	MOV D,H	54	13–48
LD D,L	* MOV D,L	MOV D,L	55	13–48
LD D,immed	* MVI D,immed	MVI D,immed	16 ib	13–48
LD E,(HL)	* MOV E,M	MOV E,M	5E	13–48
LD E,(IX+d)	MOV E,d(X)	LDX E,d	DD 5E db	13–48
LD E,(IY+d)	MOV E,d(Y)	LDY E,d	FD 5E db	13–48

Zilog		TDL	MAC	Disassembly	Page
LD E,A	*	MOV E,A	MOV E,A	5F	13–48
LD E,B	*	MOV E,B	MOV E,B	58	13–48
LD E,C	*	MOV E,C	MOV E,C	59	13–48
LD E,D	*	MOV E,D	MOV E,D	5A	13–48
LD E,E	*	MOV E,E	MOV E,E	5B	13–48
LD E,H	*	MOV E,H	MOV E,H	5C	13–48
LD E,L	*	MOV E,L	MOV E,L	5D	13–48
LD E,immed	*	MVI E,immed	MVI E,immed	1E ib	13–48
LD H,(HL)	*	MOV H,M	MOV H,M	66	13–48
LD H,(IX+d)		MOV H,d(X)	LDX H,d	DD 66 db	13–48
LD H,(IY+d)		MOV H,d(Y)	LDY H,d	FD 66 db	13–48
LD H,A	*	MOV H,A	MOV H,A	67	13–48
LD H,B	*	MOV H,B	MOV H,B	60	13–48
LD H,C	*	MOV H,C	MOV H,C	61	13–48
LD H,D	*	MOV H,D	MOV H,D	62	13–48
LD H,E	*	MOV H,E	MOV H,E	63	13–48
LD H,H	*	MOV H,H	MOV H,H	64	13–48
LD H,L	*	MOV H,L	MOV H,L	65	13–48
LD H,immed	*	MVI H,immed	MVI H,immed	26 ib	13–48
LD I,A		STAI	STAI	ED 47	13–48
LD L,(HL)	*	MOV L,M	MOV L,M	6E	13–48
LD L,(IX+d)		MOV L,d(X)	LDX L,d	DD 6E db	13–48
LD L,(IY+d)		MOV L,d(Y)	LDY L,d	FD 6E db	13–48
LD L,A	*	MOV L,A	MOV L,A	6F	13–48
LD L,B	*	MOV L,B	MOV L,B	68	13–48
LD L,C	*	MOV L,C	MOV L,C	69	13–48
LD L,D	*	MOV L,D	MOV L,D	6A	13–48
LD L,E	*	MOV L,E	MOV L,E	6B	13–48
LD L,H	*	MOV L,H	MOV L,H	6C	13–48
LD L,L	*	MOV L,L	MOV L,L	6D	13–48
LD L,immed	*	MVI L,immed	MVI L,immed	2E ib	13–48
LD R,A		STAR	STAR	ED 4F	13–48
LD BC,(addr)		LBCD addr	LBCD addr	ED 4B al ah	13–48
LD BC,imwrd	*	LXI B,imwrd	LXI B,imwrd	01 il ih	13–48
LD DE,(addr)		LDED addr	LDED addr	ED 5B al ah	13–48
LD DE,imwrd	*	LXI D,imwrd	LXI D,imwrd	11 il ih	13–48
LD HL,(addr)	*	LHLD addr	LHLD addr	2A al ah	13–48
LD HL,imwrd	*	LXI H,imwrd	LXI H,imwrd	21 il ih	13–48
LD IX,(addr)		LIXD addr	LIXD addr	DD 2A al ah	13–48
LD IX,imwrd		LXI X,imwrd	LXIX imwrd	DD 21 il ih	13–48
LD IY,(addr)		LIYD addr	LIYD addr	FD 2A al ah	13–48
LD IY,imwrd		LXI Y,imwrd	LXIY imwrd	FD 21 il ih	13–48
LD SP,(addr)		LSPD addr	LSPD addr	ED 7B al ah	13–48
LD SP,HL	*	SPHL	SPHL	F9	13–48
LD SP,IX		SPIX	SPIX	DD F9	13–48
LD SP,IY		SPIY	SPIY	FD F9	13–48
LD SP,imwrd	*	LXI SP,imwrd	LXI SP,imwrd	31 il ih	13–48

Zilog	TDL	MAC	Disassembly	Page
LDD	LDD	LDD	ED A8	13–55
LDDR	LDDR	LDDR	ED B8	13–56
LDI	LDI	LDI	ED A0	13–57
LDIR	LDIR	LDIR	ED B0	13–58
~ MLT BC	—	—	ED 4C	13–59
~ MLT DE	—	—	ED 5C	13–59
~ MLT HL	—	—	ED 6C	13–59
~ MLT SP	—	—	ED 7C	13–59
NEG	NEG	NEG	ED 44	13–60
NOP	* NOP	NOP	00	13–61
OR (HL)	* ORA M	ORA M	B6	13–62
OR (IX+d)	ORA d(X)	ORAX d	DD B6 db	13–62
OR (IY+d)	ORA d(Y)	ORAY d	FD B6 db	13–62
OR A	* ORA A	ORA A	B7	13–62
OR B	* ORA B	ORA B	B0	13–62
OR C	* ORA C	ORA C	B1	13–62
OR D	* ORA D	ORA D	B2	13–62
OR E	* ORA E	ORA E	B3	13–62
OR H	* ORA H	ORA H	B4	13–62
OR L	* ORA L	ORA L	B5	13–62
OR immed	* ORI immed	ORI immed	F6 ib	13–62
~ OTDM	—	—	ED 8B	13–64
~ OTDMR	—	—	ED 9B	13–65
OTDR	OTDR	OTDR	ED BB	13–66
~ OTIM	—	—	ED 83	13–67
~ OTIMR	—	—	ED 93	13–68
OTIR	OTIR	OTIR	ED B3	13–69
OUT (C),A	OUTP A	OUTP A	ED 79	13–70
OUT (C),B	OUTP B	OUTP B	ED 41	13–70
OUT (C),C	OUTP C	OUTP C	ED 49	13–70
OUT (C),D	OUTP D	OUTP D	ED 51	13–70
OUT (C),E	OUTP E	OUTP E	ED 59	13–70
OUT (C),H	OUTP H	OUTP H	ED 61	13–70
OUT (C),L	OUTP L	OUTP L	ED 69	13–70
OUT (port),A	* OUT port	OUT port	D3 pb	13–70
~ OUT0 (port),A	—	—	ED 39	13–72
~ OUT0 (port),B	—	—	ED 01	13–72
~ OUT0 (port),C	—	—	ED 09	13–72
~ OUT0 (port),D	—	—	ED 11	13–72
~ OUT0 (port),E	—	—	ED 19	13–72
~ OUT0 (port),H	—	—	ED 21	13–72
~ OUT0 (port),L	—	—	ED 29	13–72
OUTD	OUTD	OUTD	ED AB	13–73
OUTI	OUTI	OUTI	ED A3	13–74

Zilog	TDL	MAC	Disassembly	Page
POP AF	* POP PSW	POP PSW	F1	13–75
POP BC	* POP B	POP B	C1	13–75
POP DE	* POP D	POP D	D1	13–75
POP HL	* POP H	POP H	E1	13–75
POP IX	POP X	POPX	DD E1	13–75
POP IY	POP Y	POPY	FD E1	13–75
PUSH AF	* PUSH PSW	PUSH PSW	F5	13–76
PUSH BC	* PUSH B	PUSH B	C5	13–76
PUSH DE	* PUSH D	PUSH D	D5	13–76
PUSH HL	* PUSH H	PUSH H	E5	13–76
PUSH IX	PUSH X	PUSHX	DD E5	13–76
PUSH IY	PUSH Y	PUSHY	FD E5	13–76
RES 0,(HL)	RES 0,M	RES 0,M	CB 86	13–77
RES 0,(IX+d)	RES 0,d(X)	RESX 0,d	DD CB db 86	13–77
RES 0,(IY+d)	RES 0,d(Y)	RESY 0,d	FD CB db 86	13–77
RES 0,A	RES 0,A	RES 0,A	CB 87	13–77
RES 0,B	RES 0,B	RES 0,B	CB 80	13–77
RES 0,C	RES 0,C	RES 0,C	CB 81	13–77
RES 0,D	RES 0,D	RES 0,D	CB 82	13–77
RES 0,E	RES 0,E	RES 0,E	CB 83	13–77
RES 0,H	RES 0,H	RES 0,H	CB 84	13–77
RES 0,L	RES 0,L	RES 0,L	CB 85	13–77
RES 1,(HL)	RES 1,M	RES 1,M	CB 8E	13–77
RES 1,(IX+d)	RES 1,d(X)	RESX 1,d	DD CB db 8E	13–77
RES 1,(IY+d)	RES 1,d(Y)	RESY 1,d	FD CB db 8E	13–77
RES 1,A	RES 1,A	RES 1,A	CB 8F	13–77
RES 1,B	RES 1,B	RES 1,B	CB 88	13–77
RES 1,C	RES 1,C	RES 1,C	CB 89	13–77
RES 1,D	RES 1,D	RES 1,D	CB 8A	13–77
RES 1,E	RES 1,E	RES 1,E	CB 8B	13–77
RES 1,H	RES 1,H	RES 1,H	CB 8C	13–77
RES 1,L	RES 1,L	RES 1,L	CB 8D	13–77
RES 2,(HL)	RES 2,M	RES 2,M	CB 96	13–77
RES 2,(IX+d)	RES 2,d(X)	RESX 2,d	DD CB db 96	13–77
RES 2,(IY+d)	RES 2,d(Y)	RESY 2,d	FD CB db 96	13–77
RES 2,A	RES 2,A	RES 2,A	CB 97	13–77
RES 2,B	RES 2,B	RES 2,B	CB 90	13–77
RES 2,C	RES 2,C	RES 2,C	CB 91	13–77
RES 2,D	RES 2,D	RES 2,D	CB 92	13–77
RES 2,E	RES 2,E	RES 2,E	CB 93	13–77
RES 2,H	RES 2,H	RES 2,H	CB 94	13–77
RES 2,L	RES 2,L	RES 2,L	CB 95	13–77
RES 3,(HL)	RES 3,M	RES 3,M	CB 9E	13–77
RES 3,(IX+d)	RES 3,d(X)	RESX 3,d	DD CB db 9E	13–77
RES 3,(IY+d)	RES 3,d(Y)	RESY 3,d	FD CB db 9E	13–77
RES 3,A	RES 3,A	RES 3,A	CB 9F	13–77
RES 3,B	RES 3,B	RES 3,B	CB 98	13–77
RES 3,C	RES 3,C	RES 3,C	CB 99	13–77
RES 3,D	RES 3,D	RES 3,D	CB 9A	13–77
RES 3,E	RES 3,E	RES 3,E	CB 9B	13–77
RES 3,H	RES 3,H	RES 3,H	CB 9C	13–77
RES 3,L	RES 3,L	RES 3,L	CB 9D	13–77

Zilog	TDL	MAC	Disassembly	Page
RES 4,(HL)	RES 4,M	RES 4,M	CB A6	13–77
RES 4,(IX+d)	RES 4,d(X)	RESX 4,d	DD CB db A6	13–77
RES 4,(IY+d)	RES 4,d(Y)	RESY 4,d	FD CB db A6	13–77
RES 4,A	RES 4,A	RES 4,A	CB A7	13–77
RES 4,B	RES 4,B	RES 4,B	CB A0	13–77
RES 4,C	RES 4,C	RES 4,C	CB A1	13–77
RES 4,D	RES 4,D	RES 4,D	CB A2	13–77
RES 4,E	RES 4,E	RES 4,E	CB A3	13–77
RES 4,H	RES 4,H	RES 4,H	CB A4	13–77
RES 4,L	RES 4,L	RES 4,L	CB A5	13–77
RES 5,(HL)	RES 5,M	RES 5,M	CB AE	13–77
RES 5,(IX+d)	RES 5,d(X)	RESX 5,d	DD CB db AE	13–77
RES 5,(IY+d)	RES 5,d(Y)	RESY 5,d	FD CB db AE	13–77
RES 5,A	RES 5,A	RES 5,A	CB AF	13–77
RES 5,B	RES 5,B	RES 5,B	CB A8	13–77
RES 5,C	RES 5,C	RES 5,C	CB A9	13–77
RES 5,D	RES 5,D	RES 5,D	CB AA	13–77
RES 5,E	RES 5,E	RES 5,E	CB AB	13–77
RES 5,H	RES 5,H	RES 5,H	CB AC	13–77
RES 5,L	RES 5,L	RES 5,L	CB AD	13–77
RES 6,(HL)	RES 6,M	RES 6,M	CB B6	13–77
RES 6,(IX+d)	RES 6,d(X)	RESX 6,d	DD CB db B6	13–77
RES 6,(IY+d)	RES 6,d(Y)	RESY 6,d	FD CB db B6	13–77
RES 6,A	RES 6,A	RES 6,A	CB B7	13–77
RES 6,B	RES 6,B	RES 6,B	CB B0	13–77
RES 6,C	RES 6,C	RES 6,C	CB B1	13–77
RES 6,D	RES 6,D	RES 6,D	CB B2	13–77
RES 6,E	RES 6,E	RES 6,E	CB B3	13–77
RES 6,H	RES 6,H	RES 6,H	CB B4	13–77
RES 6,L	RES 6,L	RES 6,L	CB B5	13–77
RES 7,(HL)	RES 7,M	RES 7,M	CB BE	13–77
RES 7,(IX+d)	RES 7,d(X)	RESX 7,d	DD CB db BE	13–77
RES 7,(IY+d)	RES 7,d(Y)	RESY 7,d	FD CB db BE	13–77
RES 7,A	RES 7,A	RES 7,A	CB BF	13–77
RES 7,B	RES 7,B	RES 7,B	CB B8	13–77
RES 7,C	RES 7,C	RES 7,C	CB B9	13–77
RES 7,D	RES 7,D	RES 7,D	CB BA	13–77
RES 7,E	RES 7,E	RES 7,E	CB BB	13–77
RES 7,H	RES 7,H	RES 7,H	CB BC	13–77
RES 7,L	RES 7,L	RES 7,L	CB BD	13–77
RET	* RET	RET	C9	13–80
RET C	* RC	RC	D8	13–80
RET M	* RM	RM	F8	13–80
RET NC	* RNC	RNC	D0	13–80
RET NZ	* RNZ	RNZ	C0	13–80
RET P	* RP	RP	F0	13–80
RET PE	* RPE	RPE	E8	13–80
RET PO	* RPO	RPO	E0	13–80
RET Z	* RZ	RZ	C8	13–80
RETI	RETI	RETI	ED 4D	13–82
RETN	RETN	RETN	ED 45	13–83

Zilog		TDL	MAC	Disassembly	Page
RL (HL)		RALR M	RALR M	CB 16	13–84
RL (IX+d)		RALR d(X)	RALX d	DD CB db 16	13–84
RL (IY+d)		RALR d(Y)	RALY d	FD CB db 16	13–84
RL A		RALR A	RALR A	CB 17	13–84
RL B		RALR B	RALR B	CB 10	13–84
RL C		RALR C	RALR C	CB 11	13–84
RL D		RALR D	RALR D	CB 12	13–84
RL E		RALR E	RALR E	CB 13	13–84
RL H		RALR H	RALR H	CB 14	13–84
RL L		RALR L	RALR L	CB 15	13–84
RLA	*	RAL	RAL	17	13–86
RLC (HL)		RLCR M	RLCR M	CB 06	13–87
RLC (IX+d)		RLCR d(X)	RLCX d	DD CB db 06	13–87
RLC (IY+d)		RLCR d(Y)	RLCY d	FD CB db 06	13–87
RLC A		RLCR A	RLCR A	CB 07	13–87
RLC B		RLCR B	RLCR B	CB 00	13–87
RLC C		RLCR C	RLCR C	CB 01	13–87
RLC D		RLCR D	RLCR D	CB 02	13–87
RLC E		RLCR E	RLCR E	CB 03	13–87
RLC H		RLCR H	RLCR H	CB 04	13–87
RLC L		RLCR L	RLCR L	CB 05	13–87
RLCA	*	RLC	RLC	07	13–89
RLD		RLD	RLD	ED 6F	13–90
RR (HL)		RARR M	RARR M	CB 1E	13–91
RR (IX+d)		RARR d(X)	RARX d	DD CB db 1E	13–91
RR (IY+d)		RARR d(Y)	RARY d	FD CB db 1E	13–91
RR A		RARR A	RARR A	CB 1F	13–91
RR B		RARR B	RARR B	CB 18	13–91
RR C		RARR C	RARR C	CB 19	13–91
RR D		RARR D	RARR D	CB 1A	13–91
RR E		RARR E	RARR E	CB 1B	13–91
RR H		RARR H	RARR H	CB 1C	13–91
RR L		RARR L	RARR L	CB 1D	13–91
RRA	*	RAR	RAR	1F	13–93
RRC (HL)		RRCR M	RRCR M	CB 0E	13–94
RRC (IX+d)		RRCR d(X)	RRCX d	DD CB db 0E	13–94
RRC (IY+d)		RRCR d(Y)	RRCY d	FD CB db 0E	13–94
RRC A		RRCR A	RRCR A	CB 0F	13–94
RRC B		RRCR B	RRCR B	CB 08	13–94
RRC C		RRCR C	RRCR C	CB 09	13–94
RRC D		RRCR D	RRCR D	CB 0A	13–94
RRC E		RRCR E	RRCR E	CB 0B	13–94
RRC H		RRCR H	RRCR H	CB 0C	13–94
RRC L		RRCR L	RRCR L	CB 0D	13–94
RRCA	*	RRC	RRC	0F	13–96
RRD		RRD	RRD	ED 67	13–97

Zilog		TDL	MAC	Disassembly	Page
RST 00	*	RST 0	RST 0	C7	13–98
RST 08	*	RST 1	RST 1	CF	13–98
RST 10	*	RST 2	RST 2	D7	13–98
RST 18	*	RST 3	RST 3	DF	13–98
RST 20	*	RST 4	RST 4	E7	13–98
RST 28	*	RST 5	RST 5	EF	13–98
RST 30	*	RST 6	RST 6	F7	13–98
RST 38	*	RST 7	RST 7	FF	13–98
SBC A,(HL)	*	SBB M	SBB M	9E	13–99
SBC A,(IX+d)		SBB d(X)	SBBX d	DD 9E	13–99
SBC A,(IY+d)		SBB d(Y)	SBBY d	FD 9E	13–99
SBC A,A	*	SBB A	SBB A	9F	13–99
SBC A,B	*	SBB B	SBB B	98	13–99
SBC A,C	*	SBB C	SBB C	99	13–99
SBC A,D	*	SBB D	SBB D	9A	13–99
SBC A,E	*	SBB E	SBB E	9B	13–99
SBC A,H	*	SBB H	SBB H	9C	13–99
SBC A,L	*	SBB L	SBB L	9D	13–99
SBC A,immed	*	SBI immed	SBI immed	DE ib	13–99
SBC HL,BC		DSBC B	DSBC B	ED 42	13–99
SBC HL,DE		DSBC D	DSBC D	ED 52	13–99
SBC HL,HL		DSBC H	DSBC H	ED 62	13–99
SBC HL,SP		DSBC SP	DSBC SP	ED 72	13–99
SCF	*	STC	STC	37	13–101
SET 0,(HL)		SET 0,M	SET 0,M	CB C6	13–102
SET 0,(IX+d)		SET 0,d(X)	SETX 0,d	DD CB db C6	13–102
SET 0,(IY+d)		SET 0,d(Y)	SETY 0,d	FD CB db C6	13–102
SET 0,A		SET 0,A	SET 0,A	CB C7	13–102
SET 0,B		SET 0,B	SET 0,B	CB C0	13–102
SET 0,C		SET 0,C	SET 0,C	CB C1	13–102
SET 0,D		SET 0,D	SET 0,D	CB C2	13–102
SET 0,E		SET 0,E	SET 0,E	CB C3	13–102
SET 0,H		SET 0,H	SET 0,H	CB C4	13–102
SET 0,L		SET 0,L	SET 0,L	CB C5	13–102
SET 1,(HL)		SET 1,M	SET 1,M	CB CE	13–102
SET 1,(IX+d)		SET 1,d(X)	SETX 1,d	DD CB db CE	13–102
SET 1,(IY+d)		SET 1,d(Y)	SETY 1,d	FD CB db CE	13–102
SET 1,A		SET 1,A	SET 1,A	CB CF	13–102
SET 1,B		SET 1,B	SET 1,B	CB C8	13–102
SET 1,C		SET 1,C	SET 1,C	CB C9	13–102
SET 1,D		SET 1,D	SET 1,D	CB CA	13–102
SET 1,E		SET 1,E	SET 1,E	CB CB	13–102
SET 1,H		SET 1,H	SET 1,H	CB CC	13–102
SET 1,L		SET 1,L	SET 1,L	CB CD	13–102
SET 2,(HL)		SET 2,M	SET 2,M	CB D6	13–102
SET 2,(IX+d)		SET 2,d(X)	SETX 2,d	DD CB db D6	13–102
SET 2,(IY+d)		SET 2,d(Y)	SETY 2,d	FD CB db D6	13–102

Zilog	TDL	MAC	Disassembly	Page
SET 2,A	SET 2,A	SET 2,A	CB D7	13–102
SET 2,B	SET 2,B	SET 2,B	CB D0	13–102
SET 2,C	SET 2,C	SET 2,C	CB D1	13–102
SET 2,D	SET 2,D	SET 2,D	CB D2	13–102
SET 2,E	SET 2,E	SET 2,E	CB D3	13–102
SET 2,H	SET 2,H	SET 2,H	CB D4	13–102
SET 2,L	SET 2,L	SET 2,L	CB D5	13–102
SET 3,(HL)	SET 3,M	SET 3,M	CB DE	13–102
SET 3,(IX+d)	SET 3,d(X)	SETX 3,d	DD CB db DE	13–102
SET 3,(IY+d)	SET 3,d(Y)	SETY 3,d	FD CB db DE	13–102
SET 3,A	SET 3,A	SET 3,A	CB DF	13–102
SET 3,B	SET 3,B	SET 3,B	CB D8	13–102
SET 3,C	SET 3,C	SET 3,C	CB D9	13–102
SET 3,D	SET 3,D	SET 3,D	CB DA	13–102
SET 3,E	SET 3,E	SET 3,E	CB DB	13–102
SET 3,H	SET 3,H	SET 3,H	CB DC	13–102
SET 3,L	SET 3,L	SET 3,L	CB DD	13–102
SET 4,(HL)	SET 4,M	SET 4,M	CB E6	13–102
SET 4,(IX+d)	SET 4,d(X)	SETX 4,d	DD CB db E6	13–102
SET 4,(IY+d)	SET 4,d(Y)	SETY 4,d	FD CB db E6	13–102
SET 4,A	SET 4,A	SET 4,A	CB E7	13–102
SET 4,B	SET 4,B	SET 4,B	CB E0	13–102
SET 4,C	SET 4,C	SET 4,C	CB E1	13–102
SET 4,D	SET 4,D	SET 4,D	CB E2	13–102
SET 4,E	SET 4,E	SET 4,E	CB E3	13–102
SET 4,H	SET 4,H	SET 4,H	CB E4	13–102
SET 4,L	SET 4,L	SET 4,L	CB E5	13–102
SET 5,(HL)	SET 5,M	SET 5,M	CB EE	13–102
SET 5,(IX+d)	SET 5,d(X)	SETX 5,d	DD CB db EE	13–102
SET 5,(IY+d)	SET 5,d(Y)	SETY 5,d	FD CB db EE	13–102
SET 5,A	SET 5,A	SET 5,A	CB EF	13–102
SET 5,B	SET 5,B	SET 5,B	CB E8	13–102
SET 5,C	SET 5,C	SET 5,C	CB E9	13–102
SET 5,D	SET 5,D	SET 5,D	CB EA	13–102
SET 5,E	SET 5,E	SET 5,E	CB EB	13–102
SET 5,H	SET 5,H	SET 5,H	CB EC	13–102
SET 5,L	SET 5,L	SET 5,L	CB ED	13–102
SET 6,(HL)	SET 6,M	SET 6,M	CB F6	13–102
SET 6,(IX+d)	SET 6,d(X)	SETX 6,d	DD CB db F6	13–102
SET 6,(IY+d)	SET 6,d(Y)	SETY 6,d	FD CB db F6	13–102
SET 6,A	SET 6,A	SET 6,A	CB F7	13–102
SET 6,B	SET 6,B	SET 6,B	CB F0	13–102
SET 6,C	SET 6,C	SET 6,C	CB F1	13–102
SET 6,D	SET 6,D	SET 6,D	CB F2	13–102
SET 6,E	SET 6,E	SET 6,E	CB F3	13–102
SET 6,H	SET 6,H	SET 6,H	CB F4	13–102
SET 6,L	SET 6,L	SET 6,L	CB F5	13–102
SET 7,(HL)	SET 7,M	SET 7,M	CB FE	13–102
SET 7,(IX+d)	SET 7,d(X)	SETX 7,d	DD CB db FE	13–102
SET 7,(IY+d)	SET 7,d(Y)	SETY 7,d	FD CB db FE	13–102

Zilog	TDL	MAC	Disassembly	Page
SET 7,A	SET 7,A	SET 7,A	CB FF	13–102
SET 7,B	SET 7,B	SET 7,B	CB F8	13–102
SET 7,C	SET 7,C	SET 7,C	CB F9	13–102
SET 7,D	SET 7,D	SET 7,D	CB FA	13–102
SET 7,E	SET 7,E	SET 7,E	CB FB	13–102
SET 7,H	SET 7,H	SET 7,H	CB FC	13–102
SET 7,L	SET 7,L	SET 7,L	CB FD	13–102
SLA (HL)	SLAR M	SLAR M	CB 26	13–105
SLA (IX+d)	SLAR d(X)	SLAX d	DD CB db 26	13–105
SLA (IY+d)	SLAR d(Y)	SLAY d	FD CB db 26	13–105
SLA A	SLAR A	SLAR A	CB 27	13–105
SLA B	SLAR B	SLAR B	CB 20	13–105
SLA C	SLAR C	SLAR C	CB 21	13–105
SLA D	SLAR D	SLAR D	CB 22	13–105
SLA E	SLAR E	SLAR E	CB 23	13–105
SLA H	SLAR H	SLAR H	CB 24	13–105
SLA L	SLAR L	SLAR L	CB 25	13–105
~ SLP	—	—	ED 76	13–107
SRA (HL)	SRAR M	SRAR M	CB 2E	13–108
SRA (IX+d)	SRAR d(X)	SRAX d	DD CB db 2E	13–108
SRA (IY+d)	SRAR d(Y)	SRAY d	FD CB db 2E	13–108
SRA A	SRAR A	SRAR A	CB 2F	13–108
SRA B	SRAR B	SRAR B	CB 28	13–108
SRA C	SRAR C	SRAR C	CB 29	13–108
SRA D	SRAR D	SRAR D	CB 2A	13–108
SRA E	SRAR E	SRAR E	CB 2B	13–108
SRA H	SRAR H	SRAR H	CB 2C	13–108
SRA L	SRAR L	SRAR L	CB 2D	13–108
SRL (HL)	SRLR M	SRLR M	CB 3E	13–110
SRL (IX+d)	SRLR d(X)	SRLX d	DD CB db 3E	13–110
SRL (IY+d)	SRLR d(Y)	SRLY d	FD CB db 3E	13–110
SRL A	SRLR A	SRLR A	CB 3F	13–110
SRL B	SRLR B	SRLR B	CB 38	13–110
SRL C	SRLR C	SRLR C	CB 39	13–110
SRL D	SRLR D	SRLR D	CB 3A	13–110
SRL E	SRLR E	SRLR E	CB 3B	13–110
SRL H	SRLR H	SRLR H	CB 3C	13–110
SRL L	SRLR L	SRLR L	CB 3D	13–110
SUB (HL)	* SUB M	SUB M	96	13–112
SUB (IX+d)	SUB d(X)	SUBX d	DD 96	13–112
SUB (IY+d)	SUB d(Y)	SUBY d	FD 96	13–112
SUB A	* SUB A	SUB A	97	13–112
SUB B	* SUB B	SUB B	90	13–112
SUB C	* SUB C	SUB C	91	13–112
SUB D	* SUB D	SUB D	92	13–112
SUB E	* SUB E	SUB E	93	13–112
SUB H	* SUB H	SUB H	94	13–112
SUB L	* SUB L	SUB L	95	13–112
SUB immed	* SUI immed	SUI immed	D6 ib	13–112

Zilog	TDL	MAC	Disassembly	Page
~ TST (HL)	—	—	ED 34	13–114
~ TST A	—	—	ED 3C	13–114
~ TST B	—	—	ED 04	13–114
~ TST C	—	—	ED 0C	13–114
~ TST D	—	—	ED 14	13–114
~ TST E	—	—	ED 1C	13–114
~ TST H	—	—	ED 24	13–114
~ TST L	—	—	ED 2C	13–114
~ TST immed	—	—	ED 64 ib	13–114
~ TSTIO immed	—	—	ED 74 ib	13–116
XOR (HL)	* XRA M	XRA M	AE	13–117
XOR (IX+d)	XRA d(X)	XRAX d	DD AE db	13–117
XOR (IY+d)	XRA d(Y)	XRAY d	FD AE db	13–117
XOR A	* XRA A	XRA A	AF	13–117
XOR B	* XRA B	XRA B	A8	13–117
XOR C	* XRA C	XRA C	A9	13–117
XOR D	* XRA D	XRA D	AA	13–117
XOR E	* XRA E	XRA E	AB	13–117
XOR H	* XRA H	XRA H	AC	13–117
XOR L	* XRA L	XRA L	AD	13–117
XOR immed	* XRI immed	XRI immed	EE ib	13–117

Cross Reference by TDL Mnemonic

Following is an op-code cross reference from TDL to Zilog and MAC mnemonics and machine codes. This table can be of great assistance in translation. The page given contains the detailed instructions for the op-codes.

Those TDL mnemonics that are indicated by an "*" are the Intel 8080 processor subset, and usually use the TDL or MAC mnemonics (identical for all 8080 instructions).

	TDL	Zilog	MAC	Disassembly	Page
*	ACI immed	ADC A,immed	ACI immed	CE ib	13–4
*	ADC A	ADC A,A	ADC A	8F	13–4
*	ADC B	ADC A,B	ADC B	88	13–4
*	ADC C	ADC A,C	ADC C	89	13–4
*	ADC D	ADC A,D	ADC D	8A	13–4
*	ADC E	ADC A,E	ADC E	8B	13–4
*	ADC H	ADC A,H	ADC H	8C	13–4
*	ADC L	ADC A,L	ADC L	8D	13–4
*	ADC M	ADC A,(HL)	ADC M	8E	13–4
	ADC d(X)	ADC A,(IX+d)	ADCX d	DD 8E db	13–4
	ADC d(Y)	ADC A,(IY+d)	ADCY d	FD 8E db	13–4
*	ADD A	ADD A,A	ADD A	87	13–6
*	ADD B	ADD A,B	ADD B	80	13–6
*	ADD C	ADD A,C	ADD C	81	13–6
*	ADD D	ADD A,D	ADD D	82	13–6
*	ADD E	ADD A,E	ADD E	83	13–6
*	ADD H	ADD A,H	ADD H	84	13–6
*	ADD L	ADD A,L	ADD L	85	13–6
*	ADD M	ADD A,(HL)	ADD M	86	13–6
	ADD d(X)	ADD A,(IX+d)	ADDX d	DD 86 db	13–6
	ADD d(Y)	ADD A,(IY+d)	ADDY d	FD 86 db	13–6
*	ADI immed	ADD A,immed	ADI immed	C6 ib	13–6
*	ANA A	AND A	ANA A	A7	13–8
*	ANA B	AND B	ANA B	A0	13–8
*	ANA C	AND C	ANA C	A1	13–8
*	ANA D	AND D	ANA D	A2	13–8
*	ANA E	AND E	ANA E	A3	13–8
*	ANA H	AND H	ANA H	A4	13–8
*	ANA L	AND L	ANA L	A5	13–8
*	ANA M	AND (HL)	ANA M	A6	13–8
	ANA d(X)	AND (IX+d)	ANAX d	DD A6 db	13–8
	ANA d(Y)	AND (IY+d)	ANAY d	FD A6 db	13–8
*	ANI immed	AND immed	ANI immed	E6 ib	13–8
	BIT 0,A	BIT 0,A	BIT 0,A	CB 47	13–10
	BIT 0,B	BIT 0,B	BIT 0,B	CB 40	13–10
	BIT 0,C	BIT 0,C	BIT 0,C	CB 41	13–10
	BIT 0,D	BIT 0,D	BIT 0,D	CB 42	13–10
	BIT 0,E	BIT 0,E	BIT 0,E	CB 43	13–10
	BIT 0,H	BIT 0,H	BIT 0,H	CB 44	13–10
	BIT 0,L	BIT 0,L	BIT 0,L	CB 45	13–10
	BIT 0,M	BIT 0,(HL)	BIT 0,M	CB 46	13–10
	BIT 0,d(X)	BIT 0,(IX+d)	BITX 0,d	DD CB db 46	13–10
	BIT 0,d(Y)	BIT 0,(IY+d)	BITY 0,d	FD CB db 46	13–10

TDL	Zilog	MAC	Disassembly	Page
BIT 1,A	BIT 1,A	BIT 1,A	CB 4F	13–10
BIT 1,B	BIT 1,B	BIT 1,B	CB 48	13–10
BIT 1,C	BIT 1,C	BIT 1,C	CB 49	13–10
BIT 1,D	BIT 1,D	BIT 1,D	CB 4A	13–10
BIT 1,E	BIT 1,E	BIT 1,E	CB 4B	13–10
BIT 1,H	BIT 1,H	BIT 1,H	CB 4C	13–10
BIT 1,L	BIT 1,L	BIT 1,L	CB 4D	13–10
BIT 1,M	BIT 1,(HL)	BIT 1,M	CB 4E	13–10
BIT 1,d(X)	BIT 1,(IX+d)	BITX 1,d	DD CB db 4E	13–10
BIT 1,d(Y)	BIT 1,(IY+d)	BITY 1,d	FD CB db 4E	13–10
BIT 2,A	BIT 2,A	BIT 2,A	CB 57	13–10
BIT 2,B	BIT 2,B	BIT 2,B	CB 50	13–10
BIT 2,C	BIT 2,C	BIT 2,C	CB 51	13–10
BIT 2,D	BIT 2,D	BIT 2,D	CB 52	13–10
BIT 2,E	BIT 2,E	BIT 2,E	CB 53	13–10
BIT 2,H	BIT 2,H	BIT 2,H	CB 54	13–10
BIT 2,L	BIT 2,L	BIT 2,L	CB 55	13–10
BIT 2,M	BIT 2,(HL)	BIT 2,M	CB 56	13–10
BIT 2,d(X)	BIT 2,(IX+d)	BITX 2,d	DD CB db 56	13–10
BIT 2,d(Y)	BIT 2,(IY+d)	BITY 2,d	FD CB db 56	13–10
BIT 3,A	BIT 3,A	BIT 3,A	CB 5F	13–10
BIT 3,B	BIT 3,B	BIT 3,B	CB 58	13–10
BIT 3,C	BIT 3,C	BIT 3,C	CB 59	13–10
BIT 3,D	BIT 3,D	BIT 3,D	CB 5A	13–10
BIT 3,E	BIT 3,E	BIT 3,E	CB 5B	13–10
BIT 3,H	BIT 3,H	BIT 3,H	CB 5C	13–10
BIT 3,L	BIT 3,L	BIT 3,L	CB 5D	13–10
BIT 3,M	BIT 3,(HL)	BIT 3,M	CB 5E	13–10
BIT 3,d(X)	BIT 3,(IX+d)	BITX 3,d	DD CB db 5E	13–10
BIT 3,d(Y)	BIT 3,(IY+d)	BITY 3,d	FD CB db 5E	13–10
BIT 4,A	BIT 4,A	BIT 4,A	CB 67	13–10
BIT 4,B	BIT 4,B	BIT 4,B	CB 60	13–10
BIT 4,C	BIT 4,C	BIT 4,C	CB 61	13–10
BIT 4,D	BIT 4,D	BIT 4,D	CB 62	13–10
BIT 4,E	BIT 4,E	BIT 4,E	CB 63	13–10
BIT 4,H	BIT 4,H	BIT 4,H	CB 64	13–10
BIT 4,L	BIT 4,L	BIT 4,L	CB 65	13–10
BIT 4,M	BIT 4,(HL)	BIT 4,M	CB 66	13–10
BIT 4,d(X)	BIT 4,(IX+d)	BITX 4,d	DD CB db 66	13–10
BIT 4,d(Y)	BIT 4,(IY+d)	BITY 4,d	FD CB db 66	13–10
BIT 5,A	BIT 5,A	BIT 5,A	CB 6F	13–10
BIT 5,B	BIT 5,B	BIT 5,B	CB 68	13–10
BIT 5,C	BIT 5,C	BIT 5,C	CB 69	13–10
BIT 5,D	BIT 5,D	BIT 5,D	CB 6A	13–10
BIT 5,E	BIT 5,E	BIT 5,E	CB 6B	13–10
BIT 5,H	BIT 5,H	BIT 5,H	CB 6C	13–10
BIT 5,L	BIT 5,L	BIT 5,L	CB 6D	13–10
BIT 5,M	BIT 5,(HL)	BIT 5,M	CB 6E	13–10
BIT 5,d(X)	BIT 5,(IX+d)	BITX 5,d	DD CB db 6E	13–10
BIT 5,d(Y)	BIT 5,(IY+d)	BITY 5,d	FD CB db 6E	13–10

TDL	Zilog	MAC	Disassembly	Page
BIT 6,A	BIT 6,A	BIT 6,A	CB 77	13–10
BIT 6,B	BIT 6,B	BIT 6,B	CB 70	13–10
BIT 6,C	BIT 6,C	BIT 6,C	CB 71	13–10
BIT 6,D	BIT 6,D	BIT 6,D	CB 72	13–10
BIT 6,E	BIT 6,E	BIT 6,E	CB 73	13–10
BIT 6,H	BIT 6,H	BIT 6,H	CB 74	13–10
BIT 6,L	BIT 6,L	BIT 6,L	CB 75	13–10
BIT 6,M	BIT 6,(HL)	BIT 6,M	CB 76	13–10
BIT 6,d(X)	BIT 6,(IX+d)	BITX 6,d	DD CB db 76	13–10
BIT 6,d(Y)	BIT 6,(IY+d)	BITY 6,d	FD CB db 76	13–10
BIT 7,A	BIT 7,A	BIT 7,A	CB 7F	13–10
BIT 7,B	BIT 7,B	BIT 7,B	CB 78	13–10
BIT 7,C	BIT 7,C	BIT 7,C	CB 79	13–10
BIT 7,D	BIT 7,D	BIT 7,D	CB 7A	13–10
BIT 7,E	BIT 7,E	BIT 7,E	CB 7B	13–10
BIT 7,H	BIT 7,H	BIT 7,H	CB 7C	13–10
BIT 7,L	BIT 7,L	BIT 7,L	CB 7D	13–10
BIT 7,M	BIT 7,(HL)	BIT 7,M	CB 7E	13–10
BIT 7,d(X)	BIT 7,(IX+d)	BITX 7,d	DD CB db 7E	13–10
BIT 7,d(Y)	BIT 7,(IY+d)	BITY 7,d	FD CB db 7E	13–10
* CALL addr	CALL addr	CALL addr	CD al ah	13–13
* CC addr	CALL C,addr	CC addr	DC al ah	13–13
CCD	CPD	CCD	ED A9	13–18
CCDR	CPDR	CCDR	ED B9	13–19
CCI	CPI	CCI	ED A1	13–20
CCIR	CPIR	CCIR	ED B1	13–21
* CM addr	CALL M,addr	CM addr	FC al ah	13–13
* CMA	CPL	CMA	2F	13–22
* CMC	CCF	CMC	3F	13–15
* CMP A	CP A	CMP A	BF	13–16
* CMP B	CP B	CMP B	B8	13–16
* CMP C	CP C	CMP C	B9	13–16
* CMP D	CP D	CMP D	BA	13–16
* CMP E	CP E	CMP E	BB	13–16
* CMP H	CP H	CMP H	BC	13–16
* CMP L	CP L	CMP L	BD	13–16
* CMP M	CP (HL)	CMP M	BE	13–16
CMP d(X)	CP (IX+d)	CMPX d	DD BE db	13–16
CMP d(Y)	CP (IY+d)	CMPY d	FD BE db	13–16
* CNC addr	CALL NC,addr	CNC addr	D4 al ah	13–13
* CNZ addr	CALL NZ,addr	CNZ addr	C4 al ah	13–13
* CP addr	CALL P,addr	CP addr	F4 al ah	13–13
* CPE addr	CALL PE,addr	CPE addr	EC al ah	13–13
* CPI immed	CP immed	CPI immed	FE ib	13–16
* CPO addr	CALL PO,addr	CPO addr	E4 al ah	13–13
* CZ addr	CALL Z,addr	CZ addr	CC al ah	13–13
* DAA	DAA	DAA	27	13–23

	TDL	Zilog	MAC	Disassembly	Page
*	DAD B	ADD HL,BC	DAD B	09	13–6
*	DAD D	ADD HL,DE	DAD D	19	13–6
*	DAD H	ADD HL,HL	DAD H	29	13–6
*	DAD SP	ADD HL,SP	DAD SP	39	13–6
	DADC B	ADC HL,BC	DADC B	ED 4A	13–4
	DADC D	ADC HL,DE	DADC D	ED 5A	13–4
	DADC H	ADC HL,HL	DADC H	ED 6A	13–4
	DADC SP	ADC HL,SP	DADC SP	ED 7A	13–4
	DADX B	ADD IX,BC	DADX B	DD 09	13–6
	DADX D	ADD IX,DE	DADX D	DD 19	13–6
	DADX H	ADD IX,HL	DADX H	DD 29	13–6
	DADX SP	ADD IX,SP	DADX SP	DD 39	13–6
	DADY B	ADD IY,BC	DADY B	FD 09	13–6
	DADY D	ADD IY,DE	DADY D	FD 19	13–6
	DADY H	ADD IY,HL	DADY H	FD 29	13–6
	DADY SP	ADD IY,SP	DADY SP	FD 39	13–6
*	DCR A	DEC A	DCR A	3D	13–25
*	DCR B	DEC B	DCR B	05	13–25
*	DCR C	DEC C	DCR C	0D	13–25
*	DCR D	DEC D	DCR D	15	13–25
*	DCR E	DEC E	DCR E	1D	13–25
*	DCR H	DEC H	DCR H	25	13–25
*	DCR L	DEC L	DCR L	2D	13–25
*	DCR M	DEC (HL)	DCR M	35	13–25
	DCR d(X)	DEC (IX+d)	DCRX d	DD 35 db	13–25
	DCR d(Y)	DEC (IY+d)	DCRY d	FD 35 db	13–25
*	DCX B	DEC BC	DCX B	0B	13–25
*	DCX D	DEC DE	DCX D	1B	13–25
*	DCX H	DEC HL	DCX H	2B	13–25
*	DCX SP	DEC SP	DCX SP	3B	13–25
	DCX X	DEC IX	DCXX	DD 2B	13–25
	DCX Y	DEC IY	DCXY	FD 2B	13–25
*	DI	DI	DI	F3	13–27
	DJNZ disp	DJNZ disp	DJNZ disp	10 ab	13–28
	DSBC B	SBC HL,BC	DSBC B	ED 42	13–99
	DSBC D	SBC HL,DE	DSBC D	ED 52	13–99
	DSBC H	SBC HL,HL	DSBC H	ED 62	13–99
	DSBC SP	SBC HL,SP	DSBC SP	ED 72	13–99
*	EI	EI	EI	FB	13–29
	EXAF	EX AF,AF'	EXAF	08	13–30
	EXX	EXX	EXX	D9	13–31
*	HLT	HALT	HLT	76	13–32
	IM0	IM 0	IM0	ED 46	13–33
	IM1	IM 1	IM1	ED 56	13–33
	IM2	IM 2	IM2	ED 5E	13–33
*	IN port	IN A,(port)	IN port	DB pb	13–35
	IND	IND	IND	ED AA	13–40

TDL	Zilog	MAC	Disassembly	Page
INDR	INDR	INDR	ED BA	13–41
INI	INI	INI	ED A2	13–42
INIR	INIR	INIR	ED B2	13–43
INP A	IN A,(C)	INP A	ED 78	13–35
INP B	IN B,(C)	INP B	ED 40	13–35
INP C	IN C,(C)	INP C	ED 48	13–35
INP D	IN D,(C)	INP D	ED 50	13–35
INP E	IN E,(C)	INP E	ED 58	13–35
INP H	IN H,(C)	INP H	ED 60	13–35
INP L	IN L,(C)	INP L	ED 68	13–35
* INR A	INC A	INR A	3C	13–38
* INR B	INC B	INR B	04	13–38
* INR C	INC C	INR C	0C	13–38
* INR D	INC D	INR D	14	13–38
* INR E	INC E	INR E	1C	13–38
* INR H	INC H	INR H	24	13–38
* INR L	INC L	INR L	2C	13–38
* INR M	INC (HL)	INR M	34	13–38
INR d(X)	INC (IX+d)	INRX d	DD 34 db	13–38
INR d(Y)	INC (IY+d)	INRY d	FD 34 db	13–38
* INX B	INC BC	INX B	03	13–38
* INX D	INC DE	INX D	13	13–38
* INX H	INC HL	INX H	23	13–38
* INX SP	INC SP	INX SP	33	13–38
INX X	INC IX	INXX	DD 23	13–38
INX Y	INC IY	INXY	FD 23	13–38
* JC addr	JP C,addr	JC addr	DA al ah	13–44
* JM addr	JP M,addr	JM addr	FA al ah	13–44
* JMP addr	JP addr	JMP addr	C3 al ah	13–44
JMPR disp	JR disp	JR disp	18 ab	13–46
* JNC addr	JP NC,addr	JNC addr	D2 al ah	13–44
* JNZ addr	JP NZ,addr	JNZ addr	C2 al ah	13–44
* JP addr	JP P,addr	JP addr	F2 al ah	13–44
* JPE addr	JP PE,addr	JPE addr	EA al ah	13–44
* JPO addr	JP PO,addr	JPO addr	E2 al ah	13–44
JRC disp	JR C,disp	JRC disp	38 ab	13–46
JRNC disp	JR NC,disp	JRNC disp	30 ab	13–46
JRNZ disp	JR NZ,disp	JRNZ disp	20 ab	13–46
JRZ disp	JR Z,disp	JRZ disp	28 ab	13–46
* JZ addr	JP Z,addr	JZ addr	CA al ah	13–44
LBCD addr	LD BC,(addr)	LBCD addr	ED 4B al ah	13–48
* LDA addr	LD A,(addr)	LDA addr	3A al ah	13–48
LDAI	LD A,I	LDAI	ED 57	13–48
LDAR	LD A,R	LDAR	ED 5F	13–48
* LDAX B	LD A,(BC)	LDAX B	0A	13–48
* LDAX D	LD A,(DE)	LDAX D	1A	13–48
LDD	LDD	LDD	ED A8	13–55

TDL	Zilog	MAC	Disassembly	Page
LDDR	LDDR	LDDR	ED B8	13–56
LDED addr	LD DE,(addr)	LDED addr	ED 5B al ah	13–48
LDI	LDI	LDI	ED A0	13–57
LDIR	LDIR	LDIR	ED B0	13–58
* LHLD addr	LD HL,(addr)	LHLD addr	2A al ah	13–48
LIXD addr	LD IX,(addr)	LIXD addr	DD 2A al ah	13–48
LIYD addr	LD IY,(addr)	LIYD addr	FD 2A al ah	13–48
LSPD addr	LD SP,(addr)	LSPD addr	ED 7B al ah	13–48
* LXI B,imwrd	LD BC,imwrd	LXI B,imwrd	01 il ih	13–48
* LXI D,imwrd	LD DE,imwrd	LXI D,imwrd	11 il ih	13–48
* LXI H,imwrd	LD HL,imwrd	LXI H,imwrd	21 il ih	13–48
* LXI SP,imwrd	LD SP,imwrd	LXI SP,imwrd	31 il ih	13–48
LXI X,imwrd	LD IX,imwrd	LXIX imwrd	DD 21 il ih	13–48
LXI Y,imwrd	LD IY,imwrd	LXIY imwrd	FD 21 il ih	13–48
* MOV A,A	LD A,A	MOV A,A	7F	13–48
* MOV A,B	LD A,B	MOV A,B	78	13–48
* MOV A,C	LD A,C	MOV A,C	79	13–48
* MOV A,D	LD A,D	MOV A,D	7A	13–48
* MOV A,E	LD A,E	MOV A,E	7B	13–48
* MOV A,H	LD A,H	MOV A,H	7C	13–48
* MOV A,L	LD A,L	MOV A,L	7D	13–48
* MOV A,M	LD A,(HL)	MOV A,M	7E	13–48
MOV A,d(X)	LD A,(IX+d)	LDX A,d	DD 7E db	13–48
MOV A,d(Y)	LD A,(IY+d)	LDY A,d	FD 7E db	13–48
* MOV B,A	LD B,A	MOV B,A	47	13–48
* MOV B,B	LD B,B	MOV B,B	40	13–48
* MOV B,C	LD B,C	MOV B,C	41	13–48
* MOV B,D	LD B,D	MOV B,D	42	13–48
* MOV B,E	LD B,E	MOV B,E	43	13–48
* MOV B,H	LD B,H	MOV B,H	44	13–48
* MOV B,L	LD B,L	MOV B,L	45	13–48
* MOV B,M	LD B,(HL)	MOV B,M	46	13–48
MOV B,d(X)	LD B,(IX+d)	LDX B,d	DD 46 db	13–48
MOV B,d(Y)	LD B,(IY+d)	LDY B,d	FD 46 db	13–48
* MOV C,A	LD C,A	MOV C,A	4F	13–48
* MOV C,B	LD C,B	MOV C,B	48	13–48
* MOV C,C	LD C,C	MOV C,C	49	13–48
* MOV C,D	LD C,D	MOV C,D	4A	13–48
* MOV C,E	LD C,E	MOV C,E	4B	13–48
* MOV C,H	LD C,H	MOV C,H	4C	13–48
* MOV C,L	LD C,L	MOV C,L	4D	13–48
* MOV C,M	LD C,(HL)	MOV C,M	4E	13–48
MOV C,d(X)	LD C,(IX+d)	LDX C,d	DD 4E db	13–48
MOV C,d(Y)	LD C,(IY+d)	LDY C,d	FD 4E db	13–48

TDL	Zilog	MAC	Disassembly	Page
* MOV D,A	LD D,A	MOV D,A	57	13–48
* MOV D,B	LD D,B	MOV D,B	50	13–48
* MOV D,C	LD D,C	MOV D,C	51	13–48
* MOV D,D	LD D,D	MOV D,D	52	13–48
* MOV D,E	LD D,E	MOV D,E	53	13–48
* MOV D,H	LD D,H	MOV D,H	54	13–48
* MOV D,L	LD D,L	MOV D,L	55	13–48
* MOV D,M	LD D,(HL)	MOV D,M	56	13–48
MOV D,d(X)	LD D,(IX+d)	LDX D,d	DD 56 db	13–48
MOV D,d(Y)	LD D,(IY+d)	LDY D,d	FD 56 db	13–48
* MOV E,A	LD E,A	MOV E,A	5F	13–48
* MOV E,B	LD E,B	MOV E,B	58	13–48
* MOV E,C	LD E,C	MOV E,C	59	13–48
* MOV E,D	LD E,D	MOV E,D	5A	13–48
* MOV E,E	LD E,E	MOV E,E	5B	13–48
* MOV E,H	LD E,H	MOV E,H	5C	13–48
* MOV E,L	LD E,L	MOV E,L	5D	13–48
* MOV E,M	LD E,(HL)	MOV E,M	5E	13–48
MOV E,d(X)	LD E,(IX+d)	LDX E,d	DD 5E db	13–48
MOV E,d(Y)	LD E,(IY+d)	LDY E,d	FD 5E db	13–48
* MOV H,A	LD H,A	MOV H,A	67	13–48
* MOV H,B	LD H,B	MOV H,B	60	13–48
* MOV H,C	LD H,C	MOV H,C	61	13–48
* MOV H,D	LD H,D	MOV H,D	62	13–48
* MOV H,E	LD H,E	MOV H,E	63	13–48
* MOV H,H	LD H,H	MOV H,H	64	13–48
* MOV H,L	LD H,L	MOV H,L	65	13–48
* MOV H,M	LD H,(HL)	MOV H,M	66	13–48
MOV H,d(X)	LD H,(IX+d)	LDX H,d	DD 66 db	13–48
MOV H,d(Y)	LD H,(IY+d)	LDY H,d	FD 66 db	13–48
* MOV L,A	LD L,A	MOV L,A	6F	13–48
* MOV L,B	LD L,B	MOV L,B	68	13–48
* MOV L,C	LD L,C	MOV L,C	69	13–48
* MOV L,D	LD L,D	MOV L,D	6A	13–48
* MOV L,E	LD L,E	MOV L,E	6B	13–48
* MOV L,H	LD L,H	MOV L,H	6C	13–48
* MOV L,L	LD L,L	MOV L,L	6D	13–48
* MOV L,M	LD L,(HL)	MOV L,M	6E	13–48
MOV L,d(X)	LD L,(IX+d)	LDX L,d	DD 6E db	13–48
MOV L,d(Y)	LD L,(IY+d)	LDY L,d	FD 6E db	13–48
* MOV M,A	LD (HL),A	MOV M,A	77	13–48
* MOV M,B	LD (HL),B	MOV M,B	70	13–48
* MOV M,C	LD (HL),C	MOV M,C	71	13–48
* MOV M,D	LD (HL),D	MOV M,D	72	13–48
* MOV M,E	LD (HL),E	MOV M,E	73	13–48
* MOV M,H	LD (HL),H	MOV M,H	74	13–48
* MOV M,L	LD (HL),L	MOV M,L	75	13–48

	TDL	Zilog	MAC	Disassembly	Page
	MOV d(X),A	LD (IX+d),A	STX A,d	DD 77 db	13–48
	MOV d(X),B	LD (IX+d),B	STX B,d	DD 70 db	13–48
	MOV d(X),C	LD (IX+d),C	STX C,d	DD 71 db	13–48
	MOV d(X),D	LD (IX+d),D	STX D,d	DD 72 db	13–48
	MOV d(X),E	LD (IX+d),E	STX E,d	DD 73 db	13–48
	MOV d(X),H	LD (IX+d),H	STX H,d	DD 74 db	13–48
	MOV d(X),L	LD (IX+d),L	STX L,d	DD 75 db	13–48
	MOV d(Y),A	LD (IY+d),A	STY A,d	FD 77 db	13–48
	MOV d(Y),B	LD (IY+d),B	STY B,d	FD 70 db	13–48
	MOV d(Y),C	LD (IY+d),C	STY C,d	FD 71 db	13–48
	MOV d(Y),D	LD (IY+d),D	STY D,d	FD 72 db	13–48
	MOV d(Y),E	LD (IY+d),E	STY E,d	FD 73 db	13–48
	MOV d(Y),H	LD (IY+d),H	STY H,d	FD 74 db	13–48
	MOV d(Y),L	LD (IY+d),L	STY L,d	FD 75 db	13–48
*	MVI A,immed	LD A,immed	MVI A,immed	3E ib	13–48
*	MVI B,immed	LD B,immed	MVI B,immed	06 ib	13–48
*	MVI C,immed	LD C,immed	MVI C,immed	0E ib	13–48
*	MVI D,immed	LD D,immed	MVI D,immed	16 ib	13–48
*	MVI E,immed	LD E,immed	MVI E,immed	1E ib	13–48
*	MVI H,immed	LD H,immed	MVI H,immed	26 ib	13–48
*	MVI L,immed	LD L,immed	MVI L,immed	2E ib	13–48
*	MVI M,immed	LD (HL),immed	MVI M,immed	36 ib	13–48
	MVI d(X),immed	LD (IX+d),immed	MVIX d,immed	DD 36 db ib	13–48
	MVI d(Y),immed	LD (IY+d),immed	MVIY d,immed	FD 36 db ib	13–48
	NEG	NEG	NEG	ED 44	13–60
*	NOP	NOP	NOP	00	13–61
*	ORA A	OR A	ORA A	B7	13–62
*	ORA B	OR B	ORA B	B0	13–62
*	ORA C	OR C	ORA C	B1	13–62
*	ORA D	OR D	ORA D	B2	13–62
*	ORA E	OR E	ORA E	B3	13–62
*	ORA H	OR H	ORA H	B4	13–62
*	ORA L	OR L	ORA L	B5	13–62
*	ORA M	OR (HL)	ORA M	B6	13–62
	ORA d(X)	OR (IX+d)	ORAX d	DD B6 db	13–62
	ORA d(Y)	OR (IY+d)	ORAY d	FD B6 db	13–62
*	ORI immed	OR immed	ORI immed	F6 ib	13–62
	OTDR	OTDR	OTDR	ED BB	13–66
	OTIR	OTIR	OTIR	ED B3	13–69
*	OUT port	OUT (port),A	OUT port	D3 pb	13–70
	OUTD	OUTD	OUTD	ED AB	13–73
	OUTI	OUTI	OUTI	ED A3	13–74
	OUTP A	OUT (C),A	OUTP A	ED 79	13–70
	OUTP B	OUT (C),B	OUTP B	ED 41	13–70
	OUTP C	OUT (C),C	OUTP C	ED 49	13–70
	OUTP D	OUT (C),D	OUTP D	ED 51	13–70
	OUTP E	OUT (C),E	OUTP E	ED 59	13–70
	OUTP H	OUT (C),H	OUTP H	ED 61	13–70
	OUTP L	OUT (C),L	OUTP L	ED 69	13–70

	TDL	Zilog	MAC	Disassembly	Page
*	PCHL	JP (HL)	PCHL	E9	13–44
	PCIX	JP (IX)	PCIX	DD E9	13–44
	PCIY	JP (IY)	PCIY	FD E9	13–44
*	POP B	POP BC	POP B	C1	13–75
*	POP D	POP DE	POP D	D1	13–75
*	POP H	POP HL	POP H	E1	13–75
*	POP PSW	POP AF	POP PSW	F1	13–75
	POP X	POP IX	POPX	DD E1	13–75
	POP Y	POP IY	POPY	FD E1	13–75
*	PUSH B	PUSH BC	PUSH B	C5	13–76
*	PUSH D	PUSH DE	PUSH D	D5	13–76
*	PUSH H	PUSH HL	PUSH H	E5	13–76
*	PUSH PSW	PUSH AF	PUSH PSW	F5	13–76
	PUSH X	PUSH IX	PUSHX	DD E5	13–76
	PUSH Y	PUSH IY	PUSHY	FD E5	13–76
*	RAL	RLA	RAL	17	13–86
	RALR A	RL A	RALR A	CB 17	13–84
	RALR B	RL B	RALR B	CB 10	13–84
	RALR C	RL C	RALR C	CB 11	13–84
	RALR D	RL D	RALR D	CB 12	13–84
	RALR E	RL E	RALR E	CB 13	13–84
	RALR H	RL H	RALR H	CB 14	13–84
	RALR L	RL L	RALR L	CB 15	13–84
	RALR M	RL (HL)	RALR M	CB 16	13–84
	RALR d(X)	RL (IX+d)	RALX d	DD CB db 16	13–84
	RALR d(Y)	RL (IY+d)	RALY d	FD CB db 16	13–84
*	RAR	RRA	RAR	1F	13–93
	RARR A	RR A	RARR A	CB 1F	13–91
	RARR B	RR B	RARR B	CB 18	13–91
	RARR C	RR C	RARR C	CB 19	13–91
	RARR D	RR D	RARR D	CB 1A	13–91
	RARR E	RR E	RARR E	CB 1B	13–91
	RARR H	RR H	RARR H	CB 1C	13–91
	RARR L	RR L	RARR L	CB 1D	13–91
	RARR M	RR (HL)	RARR M	CB 1E	13–91
	RARR d(X)	RR (IX+d)	RARX d	DD CB db 1E	13–91
	RARR d(Y)	RR (IY+d)	RARY d	FD CB db 1E	13–91
*	RC	RET C	RC	D8	13–80
	RES 0,A	RES 0,A	RES 0,A	CB 87	13–77
	RES 0,B	RES 0,B	RES 0,B	CB 80	13–77
	RES 0,C	RES 0,C	RES 0,C	CB 81	13–77
	RES 0,D	RES 0,D	RES 0,D	CB 82	13–77
	RES 0,E	RES 0,E	RES 0,E	CB 83	13–77
	RES 0,H	RES 0,H	RES 0,H	CB 84	13–77
	RES 0,L	RES 0,L	RES 0,L	CB 85	13–77
	RES 0,M	RES 0,(HL)	RES 0,M	CB 86	13–77
	RES 0,d(X)	RES 0,(IX+d)	RESX 0,d	DD CB db 86	13–77
	RES 0,d(Y)	RES 0,(IY+d)	RESY 0,d	FD CB db 86	13–77

TDL	Zilog	MAC	Disassembly	Page
RES 1,A	RES 1,A	RES 1,A	CB 8F	13–77
RES 1,B	RES 1,B	RES 1,B	CB 88	13–77
RES 1,C	RES 1,C	RES 1,C	CB 89	13–77
RES 1,D	RES 1,D	RES 1,D	CB 8A	13–77
RES 1,E	RES 1,E	RES 1,E	CB 8B	13–77
RES 1,H	RES 1,H	RES 1,H	CB 8C	13–77
RES 1,L	RES 1,L	RES 1,L	CB 8D	13–77
RES 1,M	RES 1,(HL)	RES 1,M	CB 8E	13–77
RES 1,d(X)	RES 1,(IX+d)	RESX 1,d	DD CB db 8E	13–77
RES 1,d(Y)	RES 1,(IY+d)	RESY 1,d	FD CB db 8E	13–77
RES 2,A	RES 2,A	RES 2,A	CB 97	13–77
RES 2,B	RES 2,B	RES 2,B	CB 90	13–77
RES 2,C	RES 2,C	RES 2,C	CB 91	13–77
RES 2,D	RES 2,D	RES 2,D	CB 92	13–77
RES 2,E	RES 2,E	RES 2,E	CB 93	13–77
RES 2,H	RES 2,H	RES 2,H	CB 94	13–77
RES 2,L	RES 2,L	RES 2,L	CB 95	13–77
RES 2,M	RES 2,(HL)	RES 2,M	CB 96	13–77
RES 2,d(X)	RES 2,(IX+d)	RESX 2,d	DD CB db 96	13–77
RES 2,d(Y)	RES 2,(IY+d)	RESY 2,d	FD CB db 96	13–77
RES 3,A	RES 3,A	RES 3,A	CB 9F	13–77
RES 3,B	RES 3,B	RES 3,B	CB 98	13–77
RES 3,C	RES 3,C	RES 3,C	CB 99	13–77
RES 3,D	RES 3,D	RES 3,D	CB 9A	13–77
RES 3,E	RES 3,E	RES 3,E	CB 9B	13–77
RES 3,H	RES 3,H	RES 3,H	CB 9C	13–77
RES 3,L	RES 3,L	RES 3,L	CB 9D	13–77
RES 3,M	RES 3,(HL)	RES 3,M	CB 9E	13–77
RES 3,d(X)	RES 3,(IX+d)	RESX 3,d	DD CB db 9E	13–77
RES 3,d(Y)	RES 3,(IY+d)	RESY 3,d	FD CB db 9E	13–77
RES 4,A	RES 4,A	RES 4,A	CB A7	13–77
RES 4,B	RES 4,B	RES 4,B	CB A0	13–77
RES 4,C	RES 4,C	RES 4,C	CB A1	13–77
RES 4,D	RES 4,D	RES 4,D	CB A2	13–77
RES 4,E	RES 4,E	RES 4,E	CB A3	13–77
RES 4,H	RES 4,H	RES 4,H	CB A4	13–77
RES 4,L	RES 4,L	RES 4,L	CB A5	13–77
RES 4,M	RES 4,(HL)	RES 4,M	CB A6	13–77
RES 4,d(X)	RES 4,(IX+d)	RESX 4,d	DD CB db A6	13–77
RES 4,d(Y)	RES 4,(IY+d)	RESY 4,d	FD CB db A6	13–77
RES 5,A	RES 5,A	RES 5,A	CB AF	13–77
RES 5,B	RES 5,B	RES 5,B	CB A8	13–77
RES 5,C	RES 5,C	RES 5,C	CB A9	13–77
RES 5,D	RES 5,D	RES 5,D	CB AA	13–77
RES 5,E	RES 5,E	RES 5,E	CB AB	13–77
RES 5,H	RES 5,H	RES 5,H	CB AC	13–77
RES 5,L	RES 5,L	RES 5,L	CB AD	13–77
RES 5,M	RES 5,(HL)	RES 5,M	CB AE	13–77
RES 5,d(X)	RES 5,(IX+d)	RESX 5,d	DD CB db AE	13–77
RES 5,d(Y)	RES 5,(IY+d)	RESY 5,d	FD CB db AE	13–77

	TDL	Zilog	MAC	Disassembly	Page
	RES 6,A	RES 6,A	RES 6,A	CB B7	13–77
	RES 6,B	RES 6,B	RES 6,B	CB B0	13–77
	RES 6,C	RES 6,C	RES 6,C	CB B1	13–77
	RES 6,D	RES 6,D	RES 6,D	CB B2	13–77
	RES 6,E	RES 6,E	RES 6,E	CB B3	13–77
	RES 6,H	RES 6,H	RES 6,H	CB B4	13–77
	RES 6,L	RES 6,L	RES 6,L	CB B5	13–77
	RES 6,M	RES 6,(HL)	RES 6,M	CB B6	13–77
	RES 6,d(X)	RES 6,(IX+d)	RESX 6,d	DD CB db B6	13–77
	RES 6,d(Y)	RES 6,(IY+d)	RESY 6,d	FD CB db B6	13–77
	RES 7,A	RES 7,A	RES 7,A	CB BF	13–77
	RES 7,B	RES 7,B	RES 7,B	CB B8	13–77
	RES 7,C	RES 7,C	RES 7,C	CB B9	13–77
	RES 7,D	RES 7,D	RES 7,D	CB BA	13–77
	RES 7,E	RES 7,E	RES 7,E	CB BB	13–77
	RES 7,H	RES 7,H	RES 7,H	CB BC	13–77
	RES 7,L	RES 7,L	RES 7,L	CB BD	13–77
	RES 7,M	RES 7,(HL)	RES 7,M	CB BE	13–77
	RES 7,d(X)	RES 7,(IX+d)	RESX 7,d	DD CB db BE	13–77
	RES 7,d(Y)	RES 7,(IY+d)	RESY 7,d	FD CB db BE	13–77
*	RET	RET	RET	C9	13–80
	RETI	RETI	RETI	ED 4D	13–82
	RETN	RETN	RETN	ED 45	13–83
*	RLC	RLCA	RLC	07	13–89
	RLCR A	RLC A	RLCR A	CB 07	13–87
	RLCR B	RLC B	RLCR B	CB 00	13–87
	RLCR C	RLC C	RLCR C	CB 01	13–87
	RLCR D	RLC D	RLCR D	CB 02	13–87
	RLCR E	RLC E	RLCR E	CB 03	13–87
	RLCR H	RLC H	RLCR H	CB 04	13–87
	RLCR L	RLC L	RLCR L	CB 05	13–87
	RLCR M	RLC (HL)	RLCR M	CB 06	13–87
	RLCR d(X)	RLC (IX+d)	RLCX d	DD CB db 06	13–87
	RLCR d(Y)	RLC (IY+d)	RLCY d	FD CB db 06	13–87
	RLD	RLD	RLD	ED 6F	13–90
*	RM	RET M	RM	F8	13–80
*	RNC	RET NC	RNC	D0	13–80
*	RNZ	RET NZ	RNZ	C0	13–80
*	RP	RET P	RP	F0	13–80
*	RPE	RET PE	RPE	E8	13–80
*	RPO	RET PO	RPO	E0	13–80
*	RRC	RRCA	RRC	0F	13–96
	RRCR A	RRC A	RRCR A	CB 0F	13–94
	RRCR B	RRC B	RRCR B	CB 08	13–94
	RRCR C	RRC C	RRCR C	CB 09	13–94
	RRCR D	RRC D	RRCR D	CB 0A	13–94
	RRCR E	RRC E	RRCR E	CB 0B	13–94
	RRCR H	RRC H	RRCR H	CB 0C	13–94
	RRCR L	RRC L	RRCR L	CB 0D	13–94
	RRCR M	RRC (HL)	RRCR M	CB 0E	13–94

	TDL	Zilog	MAC	Disassembly	Page
	RRCR d(X)	RRC (IX+d)	RRCX d	DD CB db 0E	13–94
	RRCR d(Y)	RRC (IY+d)	RRCY d	FD CB db 0E	13–94
	RRD	RRD	RRD	ED 67	13–97
*	RST 0	RST 00	RST 0	C7	13–98
*	RST 1	RST 08	RST 1	CF	13–98
*	RST 2	RST 10	RST 2	D7	13–98
*	RST 3	RST 18	RST 3	DF	13–98
*	RST 4	RST 20	RST 4	E7	13–98
*	RST 5	RST 28	RST 5	EF	13–98
*	RST 6	RST 30	RST 6	F7	13–98
*	RST 7	RST 38	RST 7	FF	13–98
*	RZ	RET Z	RZ	C8	13–80
*	SBB A	SBC A,A	SBB A	9F	13–99
*	SBB B	SBC A,B	SBB B	98	13–99
*	SBB C	SBC A,C	SBB C	99	13–99
*	SBB D	SBC A,D	SBB D	9A	13–99
*	SBB E	SBC A,E	SBB E	9B	13–99
*	SBB H	SBC A,H	SBB H	9C	13–99
*	SBB L	SBC A,L	SBB L	9D	13–99
*	SBB M	SBC A,(HL)	SBB M	9E	13–99
	SBB d(X)	SBC A,(IX+d)	SBBX d	DD 9E	13–99
	SBB d(Y)	SBC A,(IY+d)	SBBY d	FD 9E	13–99
	SBCD addr	LD (addr),BC	SBCD addr	ED 43 al ah	13–48
*	SBI immed	SBC A,immed	SBI immed	DE ib	13–99
	SDED addr	LD (addr),DE	SDED addr	ED 53 al ah	13–48
	SET 0,A	SET 0,A	SET 0,A	CB C7	13–102
	SET 0,B	SET 0,B	SET 0,B	CB C0	13–102
	SET 0,C	SET 0,C	SET 0,C	CB C1	13–102
	SET 0,D	SET 0,D	SET 0,D	CB C2	13–102
	SET 0,E	SET 0,E	SET 0,E	CB C3	13–102
	SET 0,H	SET 0,H	SET 0,H	CB C4	13–102
	SET 0,L	SET 0,L	SET 0,L	CB C5	13–102
	SET 0,M	SET 0,(HL)	SET 0,M	CB C6	13–102
	SET 0,d(X)	SET 0,(IX+d)	SETX 0,d	DD CB db C6	13–102
	SET 0,d(Y)	SET 0,(IY+d)	SETY 0,d	FD CB db C6	13–102
	SET 1,A	SET 1,A	SET 1,A	CB CF	13–102
	SET 1,B	SET 1,B	SET 1,B	CB C8	13–102
	SET 1,C	SET 1,C	SET 1,C	CB C9	13–102
	SET 1,D	SET 1,D	SET 1,D	CB CA	13–102
	SET 1,E	SET 1,E	SET 1,E	CB CB	13–102
	SET 1,H	SET 1,H	SET 1,H	CB CC	13–102
	SET 1,L	SET 1,L	SET 1,L	CB CD	13–102
	SET 1,M	SET 1,(HL)	SET 1,M	CB CE	13–102
	SET 1,d(X)	SET 1,(IX+d)	SETX 1,d	DD CB db CE	13–102
	SET 1,d(Y)	SET 1,(IY+d)	SETY 1,d	FD CB db CE	13–102

TDL	Zilog	MAC	Disassembly	Page
SET 2,A	SET 2,A	SET 2,A	CB D7	13–102
SET 2,B	SET 2,B	SET 2,B	CB D0	13–102
SET 2,C	SET 2,C	SET 2,C	CB D1	13–102
SET 2,D	SET 2,D	SET 2,D	CB D2	13–102
SET 2,E	SET 2,E	SET 2,E	CB D3	13–102
SET 2,H	SET 2,H	SET 2,H	CB D4	13–102
SET 2,L	SET 2,L	SET 2,L	CB D5	13–102
SET 2,M	SET 2,(HL)	SET 2,M	CB D6	13–102
SET 2,d(X)	SET 2,(IX+d)	SETX 2,d	DD CB db D6	13–102
SET 2,d(Y)	SET 2,(IY+d)	SETY 2,d	FD CB db D6	13–102
SET 3,A	SET 3,A	SET 3,A	CB DF	13–102
SET 3,B	SET 3,B	SET 3,B	CB D8	13–102
SET 3,C	SET 3,C	SET 3,C	CB D9	13–102
SET 3,D	SET 3,D	SET 3,D	CB DA	13–102
SET 3,E	SET 3,E	SET 3,E	CB DB	13–102
SET 3,H	SET 3,H	SET 3,H	CB DC	13–102
SET 3,L	SET 3,L	SET 3,L	CB DD	13–102
SET 3,M	SET 3,(HL)	SET 3,M	CB DE	13–102
SET 3,d(X)	SET 3,(IX+d)	SETX 3,d	DD CB db DE	13–102
SET 3,d(Y)	SET 3,(IY+d)	SETY 3,d	FD CB db DE	13–102
SET 4,A	SET 4,A	SET 4,A	CB E7	13–102
SET 4,B	SET 4,B	SET 4,B	CB E0	13–102
SET 4,C	SET 4,C	SET 4,C	CB E1	13–102
SET 4,D	SET 4,D	SET 4,D	CB E2	13–102
SET 4,E	SET 4,E	SET 4,E	CB E3	13–102
SET 4,H	SET 4,H	SET 4,H	CB E4	13–102
SET 4,L	SET 4,L	SET 4,L	CB E5	13–102
SET 4,M	SET 4,(HL)	SET 4,M	CB E6	13–102
SET 4,d(X)	SET 4,(IX+d)	SETX 4,d	DD CB db E6	13–102
SET 4,d(Y)	SET 4,(IY+d)	SETY 4,d	FD CB db E6	13–102
SET 5,A	SET 5,A	SET 5,A	CB EF	13–102
SET 5,B	SET 5,B	SET 5,B	CB E8	13–102
SET 5,C	SET 5,C	SET 5,C	CB E9	13–102
SET 5,D	SET 5,D	SET 5,D	CB EA	13–102
SET 5,E	SET 5,E	SET 5,E	CB EB	13–102
SET 5,H	SET 5,H	SET 5,H	CB EC	13–102
SET 5,L	SET 5,L	SET 5,L	CB ED	13–102
SET 5,M	SET 5,(HL)	SET 5,M	CB EE	13–102
SET 5,d(X)	SET 5,(IX+d)	SETX 5,d	DD CB db EE	13–102
SET 5,d(Y)	SET 5,(IY+d)	SETY 5,d	FD CB db EE	13–102
SET 6,A	SET 6,A	SET 6,A	CB F7	13–102
SET 6,B	SET 6,B	SET 6,B	CB F0	13–102
SET 6,C	SET 6,C	SET 6,C	CB F1	13–102
SET 6,D	SET 6,D	SET 6,D	CB F2	13–102
SET 6,E	SET 6,E	SET 6,E	CB F3	13–102
SET 6,H	SET 6,H	SET 6,H	CB F4	13–102
SET 6,L	SET 6,L	SET 6,L	CB F5	13–102
SET 6,M	SET 6,(HL)	SET 6,M	CB F6	13–102
SET 6,d(X)	SET 6,(IX+d)	SETX 6,d	DD CB db F6	13–102
SET 6,d(Y)	SET 6,(IY+d)	SETY 6,d	FD CB db F6	13–102

	TDL	Zilog	MAC	Disassembly	Page
	SET 7,A	SET 7,A	SET 7,A	CB FF	13–102
	SET 7,B	SET 7,B	SET 7,B	CB F8	13–102
	SET 7,C	SET 7,C	SET 7,C	CB F9	13–102
	SET 7,D	SET 7,D	SET 7,D	CB FA	13–102
	SET 7,E	SET 7,E	SET 7,E	CB FB	13–102
	SET 7,H	SET 7,H	SET 7,H	CB FC	13–102
	SET 7,L	SET 7,L	SET 7,L	CB FD	13–102
	SET 7,M	SET 7,(HL)	SET 7,M	CB FE	13–102
	SET 7,d(X)	SET 7,(IX+d)	SETX 7,d	DD CB db FE	13–102
	SET 7,d(Y)	SET 7,(IY+d)	SETY 7,d	FD CB db FE	13–102
*	SHLD addr	LD (addr),HL	SHLD addr	22 al ah	13–48
	SIXD addr	LD (addr),IX	SIXD addr	DD 22 al ah	13–48
	SIYD addr	LD (addr),IY	SIYD addr	FD 22 al ah	13–48
	SLAR A	SLA A	SLAR A	CB 27	13–105
	SLAR B	SLA B	SLAR B	CB 20	13–105
	SLAR C	SLA C	SLAR C	CB 21	13–105
	SLAR D	SLA D	SLAR D	CB 22	13–105
	SLAR E	SLA E	SLAR E	CB 23	13–105
	SLAR H	SLA H	SLAR H	CB 24	13–105
	SLAR L	SLA L	SLAR L	CB 25	13–105
	SLAR M	SLA (HL)	SLAR M	CB 26	13–105
	SLAR d(X)	SLA (IX+d)	SLAX d	DD CB db 26	13–105
	SLAR d(Y)	SLA (IY+d)	SLAY d	FD CB db 26	13–105
*	SPHL	LD SP,HL	SPHL	F9	13–48
	SPIX	LD SP,IX	SPIX	DD F9	13–48
	SPIY	LD SP,IY	SPIY	FD F9	13–48
	SRAR A	SRA A	SRAR A	CB 2F	13–108
	SRAR B	SRA B	SRAR B	CB 28	13–108
	SRAR C	SRA C	SRAR C	CB 29	13–108
	SRAR D	SRA D	SRAR D	CB 2A	13–108
	SRAR E	SRA E	SRAR E	CB 2B	13–108
	SRAR H	SRA H	SRAR H	CB 2C	13–108
	SRAR L	SRA L	SRAR L	CB 2D	13–108
	SRAR M	SRA (HL)	SRAR M	CB 2E	13–108
	SRAR d(X)	SRA (IX+d)	SRAX d	DD CB db 2E	13–108
	SRAR d(Y)	SRA (IY+d)	SRAY d	FD CB db 2E	13–108
	SRLR A	SRL A	SRLR A	CB 3F	13–110
	SRLR B	SRL B	SRLR B	CB 38	13–110
	SRLR C	SRL C	SRLR C	CB 39	13–110
	SRLR D	SRL D	SRLR D	CB 3A	13–110
	SRLR E	SRL E	SRLR E	CB 3B	13–110
	SRLR H	SRL H	SRLR H	CB 3C	13–110
	SRLR L	SRL L	SRLR L	CB 3D	13–110
	SRLR M	SRL (HL)	SRLR M	CB 3E	13–110
	SRLR d(X)	SRL (IX+d)	SRLX d	DD CB db 3E	13–110
	SRLR d(Y)	SRL (IY+d)	SRLY d	FD CB db 3E	13–110
	SSPD addr	LD (addr),SP	SSPD addr	ED 73 al ah	13–48
*	STA addr	LD (addr),A	STA addr	32 al ah	13–48
	STAI	LD I,A	STAI	ED 47	13–48
	STAR	LD R,A	STAR	ED 4F	13–48

	TDL	Zilog	MAC	Disassembly	Page
*	STAX B	LD (BC),A	STAX B	02	13–48
*	STAX D	LD (DE),A	STAX D	12	13–48
*	STC	SCF	STC	37	13–101
*	SUB A	SUB A	SUB A	97	13–112
*	SUB B	SUB B	SUB B	90	13–112
*	SUB C	SUB C	SUB C	91	13–112
*	SUB D	SUB D	SUB D	92	13–112
*	SUB E	SUB E	SUB E	93	13–112
*	SUB H	SUB H	SUB H	94	13–112
*	SUB L	SUB L	SUB L	95	13–112
*	SUB M	SUB (HL)	SUB M	96	13–112
	SUB d(X)	SUB (IX+d)	SUBX d	DD 96	13–112
	SUB d(Y)	SUB (IY+d)	SUBY d	FD 96	13–112
*	SUI immed	SUB immed	SUI immed	D6 ib	13–112
*	XCHG	EX DE,HL	XCHG	EB	13–30
*	XRA A	XOR A	XRA A	AF	13–117
*	XRA B	XOR B	XRA B	A8	13–117
*	XRA C	XOR C	XRA C	A9	13–117
*	XRA D	XOR D	XRA D	AA	13–117
*	XRA E	XOR E	XRA E	AB	13–117
*	XRA H	XOR H	XRA H	AC	13–117
*	XRA L	XOR L	XRA L	AD	13–117
*	XRA M	XOR (HL)	XRA M	AE	13–117
	XRA d(X)	XOR (IX+d)	XRAX d	DD AE db	13–117
	XRA d(Y)	XOR (IY+d)	XRAY d	FD AE db	13–117
*	XRI immed	XOR immed	XRI immed	EE ib	13–117
*	XTHL	EX (SP),HL	XTHL	E3	13–30
	XTIX	EX (SP),IX	XTIX	DD E3	13–30
	XTIY	EX (SP),IY	XTIY	FD E3	13–30

Cross Reference by MAC Mnemonic

Following is an op-code cross reference from MAC to Zilog and TDL mnemonics and machine codes. This table can be of great assistance in translation. The page given contains the detailed instructions for the op-codes.

Those MAC mnemonics that are indicated by an "*" are the Intel 8080 processor subset, and usually use the TDL or MAC mnemonics (identical for all 8080 instructions).

MAC	Zilog		TDL	Disassembly	Page
ACI immed	ADC A,immed	*	ACI immed	CE ib	13–4
ADC A	ADC A,A	*	ADC A	8F	13–4
ADC B	ADC A,B	*	ADC B	88	13–4
ADC C	ADC A,C	*	ADC C	89	13–4
ADC D	ADC A,D	*	ADC D	8A	13–4
ADC E	ADC A,E	*	ADC E	8B	13–4
ADC H	ADC A,H	*	ADC H	8C	13–4
ADC L	ADC A,L	*	ADC L	8D	13–4
ADC M	ADC A,(HL)	*	ADC M	8E	13–4
ADCX d	ADC A,(IX+d)		ADC d(X)	DD 8E db	13–4
ADCY d	ADC A,(IY+d)		ADC d(Y)	FD 8E db	13–4
ADD A	ADD A,A	*	ADD A	87	13–6
ADD B	ADD A,B	*	ADD B	80	13–6
ADD C	ADD A,C	*	ADD C	81	13–6
ADD D	ADD A,D	*	ADD D	82	13–6
ADD E	ADD A,E	*	ADD E	83	13–6
ADD H	ADD A,H	*	ADD H	84	13–6
ADD L	ADD A,L	*	ADD L	85	13–6
ADD M	ADD A,(HL)	*	ADD M	86	13–6
ADDX d	ADD A,(IX+d)		ADD d(X)	DD 86 db	13–6
ADDY d	ADD A,(IY+d)		ADD d(Y)	FD 86 db	13–6
ADI immed	ADD A,immed	*	ADI immed	C6 ib	13–6
ANA A	AND A	*	ANA A	A7	13–8
ANA B	AND B	*	ANA B	A0	13–8
ANA C	AND C	*	ANA C	A1	13–8
ANA D	AND D	*	ANA D	A2	13–8
ANA E	AND E	*	ANA E	A3	13–8
ANA H	AND H	*	ANA H	A4	13–8
ANA L	AND L	*	ANA L	A5	13–8
ANA M	AND (HL)	*	ANA M	A6	13–8
ANAX d	AND (IX+d)		ANA d(X)	DD A6 db	13–8
ANAY d	AND (IY+d)		ANA d(Y)	FD A6 db	13–8
ANI immed	AND immed	*	ANI immed	E6 ib	13–8
BIT 0,A	BIT 0,A		BIT 0,A	CB 47	13–10
BIT 0,B	BIT 0,B		BIT 0,B	CB 40	13–10
BIT 0,C	BIT 0,C		BIT 0,C	CB 41	13–10
BIT 0,D	BIT 0,D		BIT 0,D	CB 42	13–10
BIT 0,E	BIT 0,E		BIT 0,E	CB 43	13–10
BIT 0,H	BIT 0,H		BIT 0,H	CB 44	13–10
BIT 0,L	BIT 0,L		BIT 0,L	CB 45	13–10
BIT 0,M	BIT 0,(HL)		BIT 0,M	CB 46	13–10

MAC	Zilog	TDL	Disassembly	Page
BIT 1,A	BIT 1,A	BIT 1,A	CB 4F	13–10
BIT 1,B	BIT 1,B	BIT 1,B	CB 48	13–10
BIT 1,C	BIT 1,C	BIT 1,C	CB 49	13–10
BIT 1,D	BIT 1,D	BIT 1,D	CB 4A	13–10
BIT 1,E	BIT 1,E	BIT 1,E	CB 4B	13–10
BIT 1,H	BIT 1,H	BIT 1,H	CB 4C	13–10
BIT 1,L	BIT 1,L	BIT 1,L	CB 4D	13–10
BIT 1,M	BIT 1,(HL)	BIT 1,M	CB 4E	13–10
BIT 2,A	BIT 2,A	BIT 2,A	CB 57	13–10
BIT 2,B	BIT 2,B	BIT 2,B	CB 50	13–10
BIT 2,C	BIT 2,C	BIT 2,C	CB 51	13–10
BIT 2,D	BIT 2,D	BIT 2,D	CB 52	13–10
BIT 2,E	BIT 2,E	BIT 2,E	CB 53	13–10
BIT 2,H	BIT 2,H	BIT 2,H	CB 54	13–10
BIT 2,L	BIT 2,L	BIT 2,L	CB 55	13–10
BIT 2,M	BIT 2,(HL)	BIT 2,M	CB 56	13–10
BIT 3,A	BIT 3,A	BIT 3,A	CB 5F	13–10
BIT 3,B	BIT 3,B	BIT 3,B	CB 58	13–10
BIT 3,C	BIT 3,C	BIT 3,C	CB 59	13–10
BIT 3,D	BIT 3,D	BIT 3,D	CB 5A	13–10
BIT 3,E	BIT 3,E	BIT 3,E	CB 5B	13–10
BIT 3,H	BIT 3,H	BIT 3,H	CB 5C	13–10
BIT 3,L	BIT 3,L	BIT 3,L	CB 5D	13–10
BIT 3,M	BIT 3,(HL)	BIT 3,M	CB 5E	13–10
BIT 4,A	BIT 4,A	BIT 4,A	CB 67	13–10
BIT 4,B	BIT 4,B	BIT 4,B	CB 60	13–10
BIT 4,C	BIT 4,C	BIT 4,C	CB 61	13–10
BIT 4,D	BIT 4,D	BIT 4,D	CB 62	13–10
BIT 4,E	BIT 4,E	BIT 4,E	CB 63	13–10
BIT 4,H	BIT 4,H	BIT 4,H	CB 64	13–10
BIT 4,L	BIT 4,L	BIT 4,L	CB 65	13–10
BIT 4,M	BIT 4,(HL)	BIT 4,M	CB 66	13–10
BIT 5,A	BIT 5,A	BIT 5,A	CB 6F	13–10
BIT 5,B	BIT 5,B	BIT 5,B	CB 68	13–10
BIT 5,C	BIT 5,C	BIT 5,C	CB 69	13–10
BIT 5,D	BIT 5,D	BIT 5,D	CB 6A	13–10
BIT 5,E	BIT 5,E	BIT 5,E	CB 6B	13–10
BIT 5,H	BIT 5,H	BIT 5,H	CB 6C	13–10
BIT 5,L	BIT 5,L	BIT 5,L	CB 6D	13–10
BIT 5,M	BIT 5,(HL)	BIT 5,M	CB 6E	13–10
BIT 6,A	BIT 6,A	BIT 6,A	CB 77	13–10
BIT 6,B	BIT 6,B	BIT 6,B	CB 70	13–10
BIT 6,C	BIT 6,C	BIT 6,C	CB 71	13–10
BIT 6,D	BIT 6,D	BIT 6,D	CB 72	13–10
BIT 6,E	BIT 6,E	BIT 6,E	CB 73	13–10
BIT 6,H	BIT 6,H	BIT 6,H	CB 74	13–10
BIT 6,L	BIT 6,L	BIT 6,L	CB 75	13–10
BIT 6,M	BIT 6,(HL)	BIT 6,M	CB 76	13–10

MAC	Zilog	TDL	Disassembly	Page
BIT 7,A	BIT 7,A	BIT 7,A	CB 7F	13–10
BIT 7,B	BIT 7,B	BIT 7,B	CB 78	13–10
BIT 7,C	BIT 7,C	BIT 7,C	CB 79	13–10
BIT 7,D	BIT 7,D	BIT 7,D	CB 7A	13–10
BIT 7,E	BIT 7,E	BIT 7,E	CB 7B	13–10
BIT 7,H	BIT 7,H	BIT 7,H	CB 7C	13–10
BIT 7,L	BIT 7,L	BIT 7,L	CB 7D	13–10
BIT 7,M	BIT 7,(HL)	BIT 7,M	CB 7E	13–10
BITX 0,d	BIT 0,(IX+d)	BIT 0,d(X)	DD CB db 46	13–10
BITX 1,d	BIT 1,(IX+d)	BIT 1,d(X)	DD CB db 4E	13–10
BITX 2,d	BIT 2,(IX+d)	BIT 2,d(X)	DD CB db 56	13–10
BITX 3,d	BIT 3,(IX+d)	BIT 3,d(X)	DD CB db 5E	13–10
BITX 4,d	BIT 4,(IX+d)	BIT 4,d(X)	DD CB db 66	13–10
BITX 5,d	BIT 5,(IX+d)	BIT 5,d(X)	DD CB db 6E	13–10
BITX 6,d	BIT 6,(IX+d)	BIT 6,d(X)	DD CB db 76	13–10
BITX 7,d	BIT 7,(IX+d)	BIT 7,d(X)	DD CB db 7E	13–10
BITY 0,d	BIT 0,(IY+d)	BIT 0,d(Y)	FD CB db 46	13–10
BITY 1,d	BIT 1,(IY+d)	BIT 1,d(Y)	FD CB db 4E	13–10
BITY 2,d	BIT 2,(IY+d)	BIT 2,d(Y)	FD CB db 56	13–10
BITY 3,d	BIT 3,(IY+d)	BIT 3,d(Y)	FD CB db 5E	13–10
BITY 4,d	BIT 4,(IY+d)	BIT 4,d(Y)	FD CB db 66	13–10
BITY 5,d	BIT 5,(IY+d)	BIT 5,d(Y)	FD CB db 6E	13–10
BITY 6,d	BIT 6,(IY+d)	BIT 6,d(Y)	FD CB db 76	13–10
BITY 7,d	BIT 7,(IY+d)	BIT 7,d(Y)	FD CB db 7E	13–10
CALL addr	CALL addr	* CALL addr	CD al ah	13–13
CC addr	CALL C,addr	* CC addr	DC al ah	13–13
CCD	CPD	CCD	ED A9	13–18
CCDR	CPDR	CCDR	ED B9	13–19
CCI	CPI	CCI	ED A1	13–20
CCIR	CPIR	CCIR	ED B1	13–21
CM addr	CALL M,addr	* CM addr	FC al ah	13–13
CMA	CPL	* CMA	2F	13–22
CMC	CCF	* CMC	3F	13–15
CMP A	CP A	* CMP A	BF	13–16
CMP B	CP B	* CMP B	B8	13–16
CMP C	CP C	* CMP C	B9	13–16
CMP D	CP D	* CMP D	BA	13–16
CMP E	CP E	* CMP E	BB	13–16
CMP H	CP H	* CMP H	BC	13–16
CMP L	CP L	* CMP L	BD	13–16
CMP M	CP (HL)	* CMP M	BE	13–16
CMPX d	CP (IX+d)	CMP d(X)	DD BE db	13–16
CMPY d	CP (IY+d)	CMP d(Y)	FD BE db	13–16
CNC addr	CALL NC,addr	* CNC addr	D4 al ah	13–13
CNZ addr	CALL NZ,addr	* CNZ addr	C4 al ah	13–13
CP addr	CALL P,addr	* CP addr	F4 al ah	13–13
CPE addr	CALL PE,addr	* CPE addr	EC al ah	13–13
CPI immed	CP immed	* CPI immed	FE ib	13–16

MAC	Zilog	TDL	Disassembly	Page
CPO addr	CALL PO,addr	* CPO addr	E4 al ah	13–13
CZ addr	CALL Z,addr	* CZ addr	CC al ah	13–13
DAA	DAA	* DAA	27	13–23
DAD B	ADD HL,BC	* DAD B	09	13–6
DAD D	ADD HL,DE	* DAD D	19	13–6
DAD H	ADD HL,HL	* DAD H	29	13–6
DAD SP	ADD HL,SP	* DAD SP	39	13–6
DADC B	ADC HL,BC	DADC B	ED 4A	13–4
DADC D	ADC HL,DE	DADC D	ED 5A	13–4
DADC H	ADC HL,HL	DADC H	ED 6A	13–4
DADC SP	ADC HL,SP	DADC SP	ED 7A	13–4
DADX B	ADD IX,BC	DADX B	DD 09	13–6
DADX D	ADD IX,DE	DADX D	DD 19	13–6
DADX H	ADD IX,HL	DADX H	DD 29	13–6
DADX SP	ADD IX,SP	DADX SP	DD 39	13–6
DADY B	ADD IY,BC	DADY B	FD 09	13–6
DADY D	ADD IY,DE	DADY D	FD 19	13–6
DADY H	ADD IY,HL	DADY H	FD 29	13–6
DADY SP	ADD IY,SP	DADY SP	FD 39	13–6
DCR A	DEC A	* DCR A	3D	13–25
DCR B	DEC B	* DCR B	05	13–25
DCR C	DEC C	* DCR C	0D	13–25
DCR D	DEC D	* DCR D	15	13–25
DCR E	DEC E	* DCR E	1D	13–25
DCR H	DEC H	* DCR H	25	13–25
DCR L	DEC L	* DCR L	2D	13–25
DCR M	DEC (HL)	* DCR M	35	13–25
DCRX d	DEC (IX+d)	DCR d(X)	DD 35 db	13–25
DCRY d	DEC (IY+d)	DCR d(Y)	FD 35 db	13–25
DCX B	DEC BC	* DCX B	0B	13–25
DCX D	DEC DE	* DCX D	1B	13–25
DCX H	DEC HL	* DCX H	2B	13–25
DCX SP	DEC SP	* DCX SP	3B	13–25
DCXX	DEC IX	DCX X	DD 2B	13–25
DCXY	DEC IY	DCX Y	FD 2B	13–25
DI	DI	* DI	F3	13–27
DJNZ disp	DJNZ disp	DJNZ disp	10 ab	13–28
DSBC B	SBC HL,BC	DSBC B	ED 42	13–99
DSBC D	SBC HL,DE	DSBC D	ED 52	13–99
DSBC H	SBC HL,HL	DSBC H	ED 62	13–99
DSBC SP	SBC HL,SP	DSBC SP	ED 72	13–99
EI	EI	* EI	FB	13–29
EXAF	EX AF,AF'	EXAF	08	13–30
EXX	EXX	EXX	D9	13–31
HLT	HALT	* HLT	76	13–32

MAC	Zilog	TDL	Disassembly	Page
IM0	IM 0	IM0	ED 46	13–33
IM1	IM 1	IM1	ED 56	13–33
IM2	IM 2	IM2	ED 5E	13–33
IN port	IN A,(port)	* IN port	DB pb	13–35
IND	IND	IND	ED AA	13–40
INDR	INDR	INDR	ED BA	13–41
INI	INI	INI	ED A2	13–42
INIR	INIR	INIR	ED B2	13–43
INP A	IN A,(C)	INP A	ED 78	13–35
INP B	IN B,(C)	INP B	ED 40	13–35
INP C	IN C,(C)	INP C	ED 48	13–35
INP D	IN D,(C)	INP D	ED 50	13–35
INP E	IN E,(C)	INP E	ED 58	13–35
INP H	IN H,(C)	INP H	ED 60	13–35
INP L	IN L,(C)	INP L	ED 68	13–35
INR A	INC A	* INR A	3C	13–38
INR B	INC B	* INR B	04	13–38
INR C	INC C	* INR C	0C	13–38
INR D	INC D	* INR D	14	13–38
INR E	INC E	* INR E	1C	13–38
INR H	INC H	* INR H	24	13–38
INR L	INC L	* INR L	2C	13–38
INR M	INC (HL)	* INR M	34	13–38
INRX d	INC (IX+d)	INR d(X)	DD 34 db	13–38
INRY d	INC (IY+d)	INR d(Y)	FD 34 db	13–38
INX B	INC BC	* INX B	03	13–38
INX D	INC DE	* INX D	13	13–38
INX H	INC HL	* INX H	23	13–38
INX SP	INC SP	* INX SP	33	13–38
INXX	INC IX	INX X	DD 23	13–38
INXY	INC IY	INX Y	FD 23	13–38
JC addr	JP C,addr	* JC addr	DA al ah	13–44
JM addr	JP M,addr	* JM addr	FA al ah	13–44
JMP addr	JP addr	* JMP addr	C3 al ah	13–44
JNC addr	JP NC,addr	* JNC addr	D2 al ah	13–44
JNZ addr	JP NZ,addr	* JNZ addr	C2 al ah	13–44
JP addr	JP P,addr	* JP addr	F2 al ah	13–44
JPE addr	JP PE,addr	* JPE addr	EA al ah	13–44
JPO addr	JP PO,addr	* JPO addr	E2 al ah	13–44
JR disp	JR disp	JMPR disp	18 ab	13–46
JRC disp	JR C,disp	JRC disp	38 ab	13–46
JRNC disp	JR NC,disp	JRNC disp	30 ab	13–46
JRNZ disp	JR NZ,disp	JRNZ disp	20 ab	13–46
JRZ disp	JR Z,disp	JRZ disp	28 ab	13–46
JZ addr	JP Z,addr	* JZ addr	CA al ah	13–44
LBCD addr	LD BC,(addr)	LBCD addr	ED 4B al ah	13–48
LDA addr	LD A,(addr)	* LDA addr	3A al ah	13–48

MAC	Zilog	TDL	Disassembly	Page
LDAI	LD A,I	LDAI	ED 57	13–48
LDAR	LD A,R	LDAR	ED 5F	13–48
LDAX B	LD A,(BC)	* LDAX B	0A	13–48
LDAX D	LD A,(DE)	* LDAX D	1A	13–48
LDD	LDD	LDD	ED A8	13–55
LDDR	LDDR	LDDR	ED B8	13–56
LDED addr	LD DE,(addr)	LDED addr	ED 5B al ah	13–48
LDI	LDI	LDI	ED A0	13–57
LDIR	LDIR	LDIR	ED B0	13–58
LDX A,d	LD A,(IX+d)	MOV A,d(X)	DD 7E db	13–48
LDX B,d	LD B,(IX+d)	MOV B,d(X)	DD 46 db	13–48
LDX C,d	LD C,(IX+d)	MOV C,d(X)	DD 4E db	13–48
LDX D,d	LD D,(IX+d)	MOV D,d(X)	DD 56 db	13–48
LDX E,d	LD E,(IX+d)	MOV E,d(X)	DD 5E db	13–48
LDX H,d	LD H,(IX+d)	MOV H,d(X)	DD 66 db	13–48
LDX L,d	LD L,(IX+d)	MOV L,d(X)	DD 6E db	13–48
LDY A,d	LD A,(IY+d)	MOV A,d(Y)	FD 7E db	13–48
LDY B,d	LD B,(IY+d)	MOV B,d(Y)	FD 46 db	13–48
LDY C,d	LD C,(IY+d)	MOV C,d(Y)	FD 4E db	13–48
LDY D,d	LD D,(IY+d)	MOV D,d(Y)	FD 56 db	13–48
LDY E,d	LD E,(IY+d)	MOV E,d(Y)	FD 5E db	13–48
LDY H,d	LD H,(IY+d)	MOV H,d(Y)	FD 66 db	13–48
LDY L,d	LD L,(IY+d)	MOV L,d(Y)	FD 6E db	13–48
LHLD addr	LD HL,(addr)	* LHLD addr	2A al ah	13–48
LIXD addr	LD IX,(addr)	LIXD addr	DD 2A al ah	13–48
LIYD addr	LD IY,(addr)	LIYD addr	FD 2A al ah	13–48
LSPD addr	LD SP,(addr)	LSPD addr	ED 7B al ah	13–48
LXI B,imwrd	LD BC,imwrd	* LXI B,imwrd	01 il ih	13–48
LXI D,imwrd	LD DE,imwrd	* LXI D,imwrd	11 il ih	13–48
LXI H,imwrd	LD HL,imwrd	* LXI H,imwrd	21 il ih	13–48
LXI SP,imwrd	LD SP,imwrd	* LXI SP,imwrd	31 il ih	13–48
LXIX imwrd	LD IX,imwrd	LXI X,imwrd	DD 21 il ih	13–48
LXIY imwrd	LD IY,imwrd	LXI Y,imwrd	FD 21 il ih	13–48
MOV A,A	LD A,A	* MOV A,A	7F	13–48
MOV A,B	LD A,B	* MOV A,B	78	13–48
MOV A,C	LD A,C	* MOV A,C	79	13–48
MOV A,D	LD A,D	* MOV A,D	7A	13–48
MOV A,E	LD A,E	* MOV A,E	7B	13–48
MOV A,H	LD A,H	* MOV A,H	7C	13–48
MOV A,L	LD A,L	* MOV A,L	7D	13–48
MOV A,M	LD A,(HL)	* MOV A,M	7E	13–48
MOV B,A	LD B,A	* MOV B,A	47	13–48
MOV B,B	LD B,B	* MOV B,B	40	13–48
MOV B,C	LD B,C	* MOV B,C	41	13–48
MOV B,D	LD B,D	* MOV B,D	42	13–48
MOV B,E	LD B,E	* MOV B,E	43	13–48
MOV B,H	LD B,H	* MOV B,H	44	13–48
MOV B,L	LD B,L	* MOV B,L	45	13–48
MOV B,M	LD B,(HL)	* MOV B,M	46	13–48

MAC	Zilog	TDL	Disassembly	Page
MOV C,A	LD C,A	* MOV C,A	4F	13–48
MOV C,B	LD C,B	* MOV C,B	48	13–48
MOV C,C	LD C,C	* MOV C,C	49	13–48
MOV C,D	LD C,D	* MOV C,D	4A	13–48
MOV C,E	LD C,E	* MOV C,E	4B	13–48
MOV C,H	LD C,H	* MOV C,H	4C	13–48
MOV C,L	LD C,L	* MOV C,L	4D	13–48
MOV C,M	LD C,(HL)	* MOV C,M	4E	13–48
MOV D,A	LD D,A	* MOV D,A	57	13–48
MOV D,B	LD D,B	* MOV D,B	50	13–48
MOV D,C	LD D,C	* MOV D,C	51	13–48
MOV D,D	LD D,D	* MOV D,D	52	13–48
MOV D,E	LD D,E	* MOV D,E	53	13–48
MOV D,H	LD D,H	* MOV D,H	54	13–48
MOV D,L	LD D,L	* MOV D,L	55	13–48
MOV D,M	LD D,(HL)	* MOV D,M	56	13–48
MOV E,A	LD E,A	* MOV E,A	5F	13–48
MOV E,B	LD E,B	* MOV E,B	58	13–48
MOV E,C	LD E,C	* MOV E,C	59	13–48
MOV E,D	LD E,D	* MOV E,D	5A	13–48
MOV E,E	LD E,E	* MOV E,E	5B	13–48
MOV E,H	LD E,H	* MOV E,H	5C	13–48
MOV E,L	LD E,L	* MOV E,L	5D	13–48
MOV E,M	LD E,(HL)	* MOV E,M	5E	13–48
MOV H,A	LD H,A	* MOV H,A	67	13–48
MOV H,B	LD H,B	* MOV H,B	60	13–48
MOV H,C	LD H,C	* MOV H,C	61	13–48
MOV H,D	LD H,D	* MOV H,D	62	13–48
MOV H,E	LD H,E	* MOV H,E	63	13–48
MOV H,H	LD H,H	* MOV H,H	64	13–48
MOV H,L	LD H,L	* MOV H,L	65	13–48
MOV H,M	LD H,(HL)	* MOV H,M	66	13–48
MOV L,A	LD L,A	* MOV L,A	6F	13–48
MOV L,B	LD L,B	* MOV L,B	68	13–48
MOV L,C	LD L,C	* MOV L,C	69	13–48
MOV L,D	LD L,D	* MOV L,D	6A	13–48
MOV L,E	LD L,E	* MOV L,E	6B	13–48
MOV L,H	LD L,H	* MOV L,H	6C	13–48
MOV L,L	LD L,L	* MOV L,L	6D	13–48
MOV L,M	LD L,(HL)	* MOV L,M	6E	13–48
MOV M,A	LD (HL),A	* MOV M,A	77	13–48
MOV M,B	LD (HL),B	* MOV M,B	70	13–48
MOV M,C	LD (HL),C	* MOV M,C	71	13–48
MOV M,D	LD (HL),D	* MOV M,D	72	13–48
MOV M,E	LD (HL),E	* MOV M,E	73	13–48
MOV M,H	LD (HL),H	* MOV M,H	74	13–48
MOV M,L	LD (HL),L	* MOV M,L	75	13–48

MAC	Zilog		TDL	Disassembly	Page
MVI A,immed	LD A,immed	*	MVI A,immed	3E ib	13–48
MVI B,immed	LD B,immed	*	MVI B,immed	06 ib	13–48
MVI C,immed	LD C,immed	*	MVI C,immed	0E ib	13–48
MVI D,immed	LD D,immed	*	MVI D,immed	16 ib	13–48
MVI E,immed	LD E,immed	*	MVI E,immed	1E ib	13–48
MVI H,immed	LD H,immed	*	MVI H,immed	26 ib	13–48
MVI L,immed	LD L,immed	*	MVI L,immed	2E ib	13–48
MVI M,immed	LD (HL),immed	*	MVI M,immed	36 ib	13–48
MVIX d,immed	LD (IX+d),immed		MVI d(X),immed	DD 36 db ib	13–48
MVIY d,immed	LD (IY+d),immed		MVI d(Y),immed	FD 36 db ib	13–48
NEG	NEG		NEG	ED 44	13–60
NOP	NOP	*	NOP	00	13–61
ORA A	OR A	*	ORA A	B7	13–62
ORA B	OR B	*	ORA B	B0	13–62
ORA C	OR C	*	ORA C	B1	13–62
ORA D	OR D	*	ORA D	B2	13–62
ORA E	OR E	*	ORA E	B3	13–62
ORA H	OR H	*	ORA H	B4	13–62
ORA L	OR L	*	ORA L	B5	13–62
ORA M	OR (HL)	*	ORA M	B6	13–62
ORAX d	OR (IX+d)		ORA d(X)	DD B6 db	13–62
ORAY d	OR (IY+d)		ORA d(Y)	FD B6 db	13–62
ORI immed	OR immed	*	ORI immed	F6 ib	13–62
OTDR	OTDR		OTDR	ED BB	13–66
OTIR	OTIR		OTIR	ED B3	13–69
OUT port	OUT (port),A	*	OUT port	D3 pb	13–70
OUTD	OUTD		OUTD	ED AB	13–73
OUTI	OUTI		OUTI	ED A3	13–74
OUTP A	OUT (C),A		OUTP A	ED 79	13–70
OUTP B	OUT (C),B		OUTP B	ED 41	13–70
OUTP C	OUT (C),C		OUTP C	ED 49	13–70
OUTP D	OUT (C),D		OUTP D	ED 51	13–70
OUTP E	OUT (C),E		OUTP E	ED 59	13–70
OUTP H	OUT (C),H		OUTP H	ED 61	13–70
OUTP L	OUT (C),L		OUTP L	ED 69	13–70
PCHL	JP (HL)	*	PCHL	E9	13–44
PCIX	JP (IX)		PCIX	DD E9	13–44
PCIY	JP (IY)		PCIY	FD E9	13–44
POP B	POP BC	*	POP B	C1	13–75
POP D	POP DE	*	POP D	D1	13–75
POP H	POP HL	*	POP H	E1	13–75
POP PSW	POP AF	*	POP PSW	F1	13–75
POPX	POP IX		POP X	DD E1	13–75
POPY	POP IY		POP Y	FD E1	13–75

MAC	Zilog		TDL	Disassembly	Page
PUSH B	PUSH BC	*	PUSH B	C5	13–76
PUSH D	PUSH DE	*	PUSH D	D5	13–76
PUSH H	PUSH HL	*	PUSH H	E5	13–76
PUSH PSW	PUSH AF	*	PUSH PSW	F5	13–76
PUSHX	PUSH IX		PUSH X	DD E5	13–76
PUSHY	PUSH IY		PUSH Y	FD E5	13–76
RAL	RLA	*	RAL	17	13–86
RALR A	RL A		RALR A	CB 17	13–84
RALR B	RL B		RALR B	CB 10	13–84
RALR C	RL C		RALR C	CB 11	13–84
RALR D	RL D		RALR D	CB 12	13–84
RALR E	RL E		RALR E	CB 13	13–84
RALR H	RL H		RALR H	CB 14	13–84
RALR L	RL L		RALR L	CB 15	13–84
RALR M	RL (HL)		RALR M	CB 16	13–84
RALX d	RL (IX+d)		RALR d(X)	DD CB db 16	13–84
RALY d	RL (IY+d)		RALR d(Y)	FD CB db 16	13–84
RAR	RRA	*	RAR	1F	13–93
RARR A	RR A		RARR A	CB 1F	13–91
RARR B	RR B		RARR B	CB 18	13–91
RARR C	RR C		RARR C	CB 19	13–91
RARR D	RR D		RARR D	CB 1A	13–91
RARR E	RR E		RARR E	CB 1B	13–91
RARR H	RR H		RARR H	CB 1C	13–91
RARR L	RR L		RARR L	CB 1D	13–91
RARR M	RR (HL)		RARR M	CB 1E	13–91
RARX d	RR (IX+d)		RARR d(X)	DD CB db 1E	13–91
RARY d	RR (IY+d)		RARR d(Y)	FD CB db 1E	13–91
RC	RET C	*	RC	D8	13–80
RES 0,A	RES 0,A		RES 0,A	CB 87	13–77
RES 0,B	RES 0,B		RES 0,B	CB 80	13–77
RES 0,C	RES 0,C		RES 0,C	CB 81	13–77
RES 0,D	RES 0,D		RES 0,D	CB 82	13–77
RES 0,E	RES 0,E		RES 0,E	CB 83	13–77
RES 0,H	RES 0,H		RES 0,H	CB 84	13–77
RES 0,L	RES 0,L		RES 0,L	CB 85	13–77
RES 0,M	RES 0,(HL)		RES 0,M	CB 86	13–77
RES 1,A	RES 1,A		RES 1,A	CB 8F	13–77
RES 1,B	RES 1,B		RES 1,B	CB 88	13–77
RES 1,C	RES 1,C		RES 1,C	CB 89	13–77
RES 1,D	RES 1,D		RES 1,D	CB 8A	13–77
RES 1,E	RES 1,E		RES 1,E	CB 8B	13–77
RES 1,H	RES 1,H		RES 1,H	CB 8C	13–77
RES 1,L	RES 1,L		RES 1,L	CB 8D	13–77
RES 1,M	RES 1,(HL)		RES 1,M	CB 8E	13–77

MAC	Zilog	TDL	Disassembly	Page
RES 2,A	RES 2,A	RES 2,A	CB 97	13–77
RES 2,B	RES 2,B	RES 2,B	CB 90	13–77
RES 2,C	RES 2,C	RES 2,C	CB 91	13–77
RES 2,D	RES 2,D	RES 2,D	CB 92	13–77
RES 2,E	RES 2,E	RES 2,E	CB 93	13–77
RES 2,H	RES 2,H	RES 2,H	CB 94	13–77
RES 2,L	RES 2,L	RES 2,L	CB 95	13–77
RES 2,M	RES 2,(HL)	RES 2,M	CB 96	13–77
RES 3,A	RES 3,A	RES 3,A	CB 9F	13–77
RES 3,B	RES 3,B	RES 3,B	CB 98	13–77
RES 3,C	RES 3,C	RES 3,C	CB 99	13–77
RES 3,D	RES 3,D	RES 3,D	CB 9A	13–77
RES 3,E	RES 3,E	RES 3,E	CB 9B	13–77
RES 3,H	RES 3,H	RES 3,H	CB 9C	13–77
RES 3,L	RES 3,L	RES 3,L	CB 9D	13–77
RES 3,M	RES 3,(HL)	RES 3,M	CB 9E	13–77
RES 4,A	RES 4,A	RES 4,A	CB A7	13–77
RES 4,B	RES 4,B	RES 4,B	CB A0	13–77
RES 4,C	RES 4,C	RES 4,C	CB A1	13–77
RES 4,D	RES 4,D	RES 4,D	CB A2	13–77
RES 4,E	RES 4,E	RES 4,E	CB A3	13–77
RES 4,H	RES 4,H	RES 4,H	CB A4	13–77
RES 4,L	RES 4,L	RES 4,L	CB A5	13–77
RES 4,M	RES 4,(HL)	RES 4,M	CB A6	13–77
RES 5,A	RES 5,A	RES 5,A	CB AF	13–77
RES 5,B	RES 5,B	RES 5,B	CB A8	13–77
RES 5,C	RES 5,C	RES 5,C	CB A9	13–77
RES 5,D	RES 5,D	RES 5,D	CB AA	13–77
RES 5,E	RES 5,E	RES 5,E	CB AB	13–77
RES 5,H	RES 5,H	RES 5,H	CB AC	13–77
RES 5,L	RES 5,L	RES 5,L	CB AD	13–77
RES 5,M	RES 5,(HL)	RES 5,M	CB AE	13–77
RES 6,A	RES 6,A	RES 6,A	CB B7	13–77
RES 6,B	RES 6,B	RES 6,B	CB B0	13–77
RES 6,C	RES 6,C	RES 6,C	CB B1	13–77
RES 6,D	RES 6,D	RES 6,D	CB B2	13–77
RES 6,E	RES 6,E	RES 6,E	CB B3	13–77
RES 6,H	RES 6,H	RES 6,H	CB B4	13–77
RES 6,L	RES 6,L	RES 6,L	CB B5	13–77
RES 6,M	RES 6,(HL)	RES 6,M	CB B6	13–77
RES 7,A	RES 7,A	RES 7,A	CB BF	13–77
RES 7,B	RES 7,B	RES 7,B	CB B8	13–77
RES 7,C	RES 7,C	RES 7,C	CB B9	13–77
RES 7,D	RES 7,D	RES 7,D	CB BA	13–77
RES 7,E	RES 7,E	RES 7,E	CB BB	13–77
RES 7,H	RES 7,H	RES 7,H	CB BC	13–77
RES 7,L	RES 7,L	RES 7,L	CB BD	13–77
RES 7,M	RES 7,(HL)	RES 7,M	CB BE	13–77

MAC	Zilog		TDL	Disassembly	Page
RESX 0,d	RES 0,(IX+d)		RES 0,d(X)	DD CB db 86	13–77
RESX 1,d	RES 1,(IX+d)		RES 1,d(X)	DD CB db 8E	13–77
RESX 2,d	RES 2,(IX+d)		RES 2,d(X)	DD CB db 96	13–77
RESX 3,d	RES 3,(IX+d)		RES 3,d(X)	DD CB db 9E	13–77
RESX 4,d	RES 4,(IX+d)		RES 4,d(X)	DD CB db A6	13–77
RESX 5,d	RES 5,(IX+d)		RES 5,d(X)	DD CB db AE	13–77
RESX 6,d	RES 6,(IX+d)		RES 6,d(X)	DD CB db B6	13–77
RESX 7,d	RES 7,(IX+d)		RES 7,d(X)	DD CB db BE	13–77
RESY 0,d	RES 0,(IY+d)		RES 0,d(Y)	FD CB db 86	13–77
RESY 1,d	RES 1,(IY+d)		RES 1,d(Y)	FD CB db 8E	13–77
RESY 2,d	RES 2,(IY+d)		RES 2,d(Y)	FD CB db 96	13–77
RESY 3,d	RES 3,(IY+d)		RES 3,d(Y)	FD CB db 9E	13–77
RESY 4,d	RES 4,(IY+d)		RES 4,d(Y)	FD CB db A6	13–77
RESY 5,d	RES 5,(IY+d)		RES 5,d(Y)	FD CB db AE	13–77
RESY 6,d	RES 6,(IY+d)		RES 6,d(Y)	FD CB db B6	13–77
RESY 7,d	RES 7,(IY+d)		RES 7,d(Y)	FD CB db BE	13–77
RET	RET	*	RET	C9	13–80
RETI	RETI		RETI	ED 4D	13–82
RETN	RETN		RETN	ED 45	13–83
RLC	RLCA	*	RLC	07	13–89
RLCR A	RLC A		RLCR A	CB 07	13–87
RLCR B	RLC B		RLCR B	CB 00	13–87
RLCR C	RLC C		RLCR C	CB 01	13–87
RLCR D	RLC D		RLCR D	CB 02	13–87
RLCR E	RLC E		RLCR E	CB 03	13–87
RLCR H	RLC H		RLCR H	CB 04	13–87
RLCR L	RLC L		RLCR L	CB 05	13–87
RLCR M	RLC (HL)		RLCR M	CB 06	13–87
RLCX d	RLC (IX+d)		RLCR d(X)	DD CB db 06	13–87
RLCY d	RLC (IY+d)		RLCR d(Y)	FD CB db 06	13–87
RLD	RLD		RLD	ED 6F	13–90
RM	RET M	*	RM	F8	13–80
RNC	RET NC	*	RNC	D0	13–80
RNZ	RET NZ	*	RNZ	C0	13–80
RP	RET P	*	RP	F0	13–80
RPE	RET PE	*	RPE	E8	13–80
RPO	RET PO	*	RPO	E0	13–80
RRC	RRCA	*	RRC	0F	13–96
RRCR A	RRC A		RRCR A	CB 0F	13–94
RRCR B	RRC B		RRCR B	CB 08	13–94
RRCR C	RRC C		RRCR C	CB 09	13–94
RRCR D	RRC D		RRCR D	CB 0A	13–94
RRCR E	RRC E		RRCR E	CB 0B	13–94
RRCR H	RRC H		RRCR H	CB 0C	13–94
RRCR L	RRC L		RRCR L	CB 0D	13–94
RRCR M	RRC (HL)		RRCR M	CB 0E	13–94
RRCX d	RRC (IX+d)		RRCR d(X)	DD CB db 0E	13–94
RRCY d	RRC (IY+d)		RRCR d(Y)	FD CB db 0E	13–94
RRD	RRD		RRD	ED 67	13–97

MAC	Zilog	TDL	Disassembly	Page
RST 0	RST 00	* RST 0	C7	13–98
RST 1	RST 08	* RST 1	CF	13–98
RST 2	RST 10	* RST 2	D7	13–98
RST 3	RST 18	* RST 3	DF	13–98
RST 4	RST 20	* RST 4	E7	13–98
RST 5	RST 28	* RST 5	EF	13–98
RST 6	RST 30	* RST 6	F7	13–98
RST 7	RST 38	* RST 7	FF	13–98
RZ	RET Z	* RZ	C8	13–80
SBB A	SBC A,A	* SBB A	9F	13–99
SBB B	SBC A,B	* SBB B	98	13–99
SBB C	SBC A,C	* SBB C	99	13–99
SBB D	SBC A,D	* SBB D	9A	13–99
SBB E	SBC A,E	* SBB E	9B	13–99
SBB H	SBC A,H	* SBB H	9C	13–99
SBB L	SBC A,L	* SBB L	9D	13–99
SBB M	SBC A,(HL)	* SBB M	9E	13–99
SBBX d	SBC A,(IX+d)	SBB d(X)	DD 9E	13–99
SBBY d	SBC A,(IY+d)	SBB d(Y)	FD 9E	13–99
SBCD addr	LD (addr),BC	SBCD addr	ED 43 al ah	13–48
SBI immed	SBC A,immed	* SBI immed	DE ib	13–99
SDED addr	LD (addr),DE	SDED addr	ED 53 al ah	13–48
SET 0,A	SET 0,A	SET 0,A	CB C7	13–102
SET 0,B	SET 0,B	SET 0,B	CB C0	13–102
SET 0,C	SET 0,C	SET 0,C	CB C1	13–102
SET 0,D	SET 0,D	SET 0,D	CB C2	13–102
SET 0,E	SET 0,E	SET 0,E	CB C3	13–102
SET 0,H	SET 0,H	SET 0,H	CB C4	13–102
SET 0,L	SET 0,L	SET 0,L	CB C5	13–102
SET 0,M	SET 0,(HL)	SET 0,M	CB C6	13–102
SET 1,A	SET 1,A	SET 1,A	CB CF	13–102
SET 1,B	SET 1,B	SET 1,B	CB C8	13–102
SET 1,C	SET 1,C	SET 1,C	CB C9	13–102
SET 1,D	SET 1,D	SET 1,D	CB CA	13–102
SET 1,E	SET 1,E	SET 1,E	CB CB	13–102
SET 1,H	SET 1,H	SET 1,H	CB CC	13–102
SET 1,L	SET 1,L	SET 1,L	CB CD	13–102
SET 1,M	SET 1,(HL)	SET 1,M	CB CE	13–102
SET 2,A	SET 2,A	SET 2,A	CB D7	13–102
SET 2,B	SET 2,B	SET 2,B	CB D0	13–102
SET 2,C	SET 2,C	SET 2,C	CB D1	13–102
SET 2,D	SET 2,D	SET 2,D	CB D2	13–102
SET 2,E	SET 2,E	SET 2,E	CB D3	13–102
SET 2,H	SET 2,H	SET 2,H	CB D4	13–102
SET 2,L	SET 2,L	SET 2,L	CB D5	13–102
SET 2,M	SET 2,(HL)	SET 2,M	CB D6	13–102

MAC	Zilog	TDL	Disassembly	Page
SET 3,A	SET 3,A	SET 3,A	CB DF	13–102
SET 3,B	SET 3,B	SET 3,B	CB D8	13–102
SET 3,C	SET 3,C	SET 3,C	CB D9	13–102
SET 3,D	SET 3,D	SET 3,D	CB DA	13–102
SET 3,E	SET 3,E	SET 3,E	CB DB	13–102
SET 3,H	SET 3,H	SET 3,H	CB DC	13–102
SET 3,L	SET 3,L	SET 3,L	CB DD	13–102
SET 3,M	SET 3,(HL)	SET 3,M	CB DE	13–102
SET 4,A	SET 4,A	SET 4,A	CB E7	13–102
SET 4,B	SET 4,B	SET 4,B	CB E0	13–102
SET 4,C	SET 4,C	SET 4,C	CB E1	13–102
SET 4,D	SET 4,D	SET 4,D	CB E2	13–102
SET 4,E	SET 4,E	SET 4,E	CB E3	13–102
SET 4,H	SET 4,H	SET 4,H	CB E4	13–102
SET 4,L	SET 4,L	SET 4,L	CB E5	13–102
SET 4,M	SET 4,(HL)	SET 4,M	CB E6	13–102
SET 5,A	SET 5,A	SET 5,A	CB EF	13–102
SET 5,B	SET 5,B	SET 5,B	CB E8	13–102
SET 5,C	SET 5,C	SET 5,C	CB E9	13–102
SET 5,D	SET 5,D	SET 5,D	CB EA	13–102
SET 5,E	SET 5,E	SET 5,E	CB EB	13–102
SET 5,H	SET 5,H	SET 5,H	CB EC	13–102
SET 5,L	SET 5,L	SET 5,L	CB ED	13–102
SET 5,M	SET 5,(HL)	SET 5,M	CB EE	13–102
SET 6,A	SET 6,A	SET 6,A	CB F7	13–102
SET 6,B	SET 6,B	SET 6,B	CB F0	13–102
SET 6,C	SET 6,C	SET 6,C	CB F1	13–102
SET 6,D	SET 6,D	SET 6,D	CB F2	13–102
SET 6,E	SET 6,E	SET 6,E	CB F3	13–102
SET 6,H	SET 6,H	SET 6,H	CB F4	13–102
SET 6,L	SET 6,L	SET 6,L	CB F5	13–102
SET 6,M	SET 6,(HL)	SET 6,M	CB F6	13–102
SET 7,A	SET 7,A	SET 7,A	CB FF	13–102
SET 7,B	SET 7,B	SET 7,B	CB F8	13–102
SET 7,C	SET 7,C	SET 7,C	CB F9	13–102
SET 7,D	SET 7,D	SET 7,D	CB FA	13–102
SET 7,E	SET 7,E	SET 7,E	CB FB	13–102
SET 7,H	SET 7,H	SET 7,H	CB FC	13–102
SET 7,L	SET 7,L	SET 7,L	CB FD	13–102
SET 7,M	SET 7,(HL)	SET 7,M	CB FE	13–102
SETX 0,d	SET 0,(IX+d)	SET 0,d(X)	DD CB db C6	13–102
SETX 1,d	SET 1,(IX+d)	SET 1,d(X)	DD CB db CE	13–102
SETX 2,d	SET 2,(IX+d)	SET 2,d(X)	DD CB db D6	13–102
SETX 3,d	SET 3,(IX+d)	SET 3,d(X)	DD CB db DE	13–102
SETX 4,d	SET 4,(IX+d)	SET 4,d(X)	DD CB db E6	13–102
SETX 5,d	SET 5,(IX+d)	SET 5,d(X)	DD CB db EE	13–102
SETX 6,d	SET 6,(IX+d)	SET 6,d(X)	DD CB db F6	13–102
SETX 7,d	SET 7,(IX+d)	SET 7,d(X)	DD CB db FE	13–102

MAC	Zilog		TDL	Disassembly	Page
SETY 0,d	SET 0,(IY+d)		SET 0,d(Y)	FD CB db C6	13–102
SETY 1,d	SET 1,(IY+d)		SET 1,d(Y)	FD CB db CE	13–102
SETY 2,d	SET 2,(IY+d)		SET 2,d(Y)	FD CB db D6	13–102
SETY 3,d	SET 3,(IY+d)		SET 3,d(Y)	FD CB db DE	13–102
SETY 4,d	SET 4,(IY+d)		SET 4,d(Y)	FD CB db E6	13–102
SETY 5,d	SET 5,(IY+d)		SET 5,d(Y)	FD CB db EE	13–102
SETY 6,d	SET 6,(IY+d)		SET 6,d(Y)	FD CB db F6	13–102
SETY 7,d	SET 7,(IY+d)		SET 7,d(Y)	FD CB db FE	13–102
SHLD addr	LD (addr),HL	*	SHLD addr	22 al ah	13–48
SIXD addr	LD (addr),IX		SIXD addr	DD 22 al ah	13–48
SIYD addr	LD (addr),IY		SIYD addr	FD 22 al ah	13–48
SLAR A	SLA A		SLAR A	CB 27	13–105
SLAR B	SLA B		SLAR B	CB 20	13–105
SLAR C	SLA C		SLAR C	CB 21	13–105
SLAR D	SLA D		SLAR D	CB 22	13–105
SLAR E	SLA E		SLAR E	CB 23	13–105
SLAR H	SLA H		SLAR H	CB 24	13–105
SLAR L	SLA L		SLAR L	CB 25	13–105
SLAR M	SLA (HL)		SLAR M	CB 26	13–105
SLAX d	SLA (IX+d)		SLAR d(X)	DD CB db 26	13–105
SLAY d	SLA (IY+d)		SLAR d(Y)	FD CB db 26	13–105
SPHL	LD SP,HL	*	SPHL	F9	13–48
SPIX	LD SP,IX		SPIX	DD F9	13–48
SPIY	LD SP,IY		SPIY	FD F9	13–48
SRAR A	SRA A		SRAR A	CB 2F	13–108
SRAR B	SRA B		SRAR B	CB 28	13–108
SRAR C	SRA C		SRAR C	CB 29	13–108
SRAR D	SRA D		SRAR D	CB 2A	13–108
SRAR E	SRA E		SRAR E	CB 2B	13–108
SRAR H	SRA H		SRAR H	CB 2C	13–108
SRAR L	SRA L		SRAR L	CB 2D	13–108
SRAR M	SRA (HL)		SRAR M	CB 2E	13–108
SRAX d	SRA (IX+d)		SRAR d(X)	DD CB db 2E	13–108
SRAY d	SRA (IY+d)		SRAR d(Y)	FD CB db 2E	13–108
SRLR A	SRL A		SRLR A	CB 3F	13–110
SRLR B	SRL B		SRLR B	CB 38	13–110
SRLR C	SRL C		SRLR C	CB 39	13–110
SRLR D	SRL D		SRLR D	CB 3A	13–110
SRLR E	SRL E		SRLR E	CB 3B	13–110
SRLR H	SRL H		SRLR H	CB 3C	13–110
SRLR L	SRL L		SRLR L	CB 3D	13–110
SRLR M	SRL (HL)		SRLR M	CB 3E	13–110
SRLX d	SRL (IX+d)		SRLR d(X)	DD CB db 3E	13–110
SRLY d	SRL (IY+d)		SRLR d(Y)	FD CB db 3E	13–110
SSPD addr	LD (addr),SP		SSPD addr	ED 73 al ah	13–48
STA addr	LD (addr),A	*	STA addr	32 al ah	13–48
STAI	LD I,A		STAI	ED 47	13–48
STAR	LD R,A		STAR	ED 4F	13–48
STAX B	LD (BC),A	*	STAX B	02	13–48
STAX D	LD (DE),A	*	STAX D	12	13–48

MAC	Zilog		TDL	Disassembly	Page
STC	SCF	*	STC	37	13–101
STX A,d	LD (IX+d),A		MOV d(X),A	DD 77 db	13–48
STX B,d	LD (IX+d),B		MOV d(X),B	DD 70 db	13–48
STX C,d	LD (IX+d),C		MOV d(X),C	DD 71 db	13–48
STX D,d	LD (IX+d),D		MOV d(X),D	DD 72 db	13–48
STX E,d	LD (IX+d),E		MOV d(X),E	DD 73 db	13–48
STX H,d	LD (IX+d),H		MOV d(X),H	DD 74 db	13–48
STX L,d	LD (IX+d),L		MOV d(X),L	DD 75 db	13–48
STY A,d	LD (IY+d),A		MOV d(Y),A	FD 77 db	13–48
STY B,d	LD (IY+d),B		MOV d(Y),B	FD 70 db	13–48
STY C,d	LD (IY+d),C		MOV d(Y),C	FD 71 db	13–48
STY D,d	LD (IY+d),D		MOV d(Y),D	FD 72 db	13–48
STY E,d	LD (IY+d),E		MOV d(Y),E	FD 73 db	13–48
STY H,d	LD (IY+d),H		MOV d(Y),H	FD 74 db	13–48
STY L,d	LD (IY+d),L		MOV d(Y),L	FD 75 db	13–48
SUB A	SUB A	*	SUB A	97	13–112
SUB B	SUB B	*	SUB B	90	13–112
SUB C	SUB C	*	SUB C	91	13–112
SUB D	SUB D	*	SUB D	92	13–112
SUB E	SUB E	*	SUB E	93	13–112
SUB H	SUB H	*	SUB H	94	13–112
SUB L	SUB L	*	SUB L	95	13–112
SUB M	SUB (HL)	*	SUB M	96	13–112
SUBX d	SUB (IX+d)		SUB d(X)	DD 96	13–112
SUBY d	SUB (IY+d)		SUB d(Y)	FD 96	13–112
SUI immed	SUB immed	*	SUI immed	D6 ib	13–112
XCHG	EX DE,HL	*	XCHG	EB	13–30
XRA A	XOR A	*	XRA A	AF	13–117
XRA B	XOR B	*	XRA B	A8	13–117
XRA C	XOR C	*	XRA C	A9	13–117
XRA D	XOR D	*	XRA D	AA	13–117
XRA E	XOR E	*	XRA E	AB	13–117
XRA H	XOR H	*	XRA H	AC	13–117
XRA L	XOR L	*	XRA L	AD	13–117
XRA M	XOR (HL)	*	XRA M	AE	13–117
XRAX d	XOR (IX+d)		XRA d(X)	DD AE db	13–117
XRAY d	XOR (IY+d)		XRA d(Y)	FD AE db	13–117
XRI immed	XOR immed	*	XRI immed	EE ib	13–117
XTHL	EX (SP),HL	*	XTHL	E3	13–30
XTIX	EX (SP),IX		XTIX	DD E3	13–30
XTIY	EX (SP),IY		XTIY	FD E3	13–30

Cross Reference by Intel 8080 Mnemonic

Following is an op-code cross reference from Intel 8080 to Zilog mnemonics and machine codes. This table can be of great assistance in translation. The page given contains the detailed instructions for the op-codes.

It should be borne in mind that the Intel 8080 mnemonic set is a subset of the TDL or MAC mnemonic sets (actually, the reverse is true, the Z-80 op-codes, regardless of mnemonics, are a superset of the 8080 op-codes).

8080DL	Zilog	Disassembly	Page
ACI immed	ADC A,immed	CE ib	13–4
ADC A	ADC A,A	8F	13–4
ADC B	ADC A,B	88	13–4
ADC C	ADC A,C	89	13–4
ADC D	ADC A,D	8A	13–4
ADC E	ADC A,E	8B	13–4
ADC H	ADC A,H	8C	13–4
ADC L	ADC A,L	8D	13–4
ADC M	ADC A,(HL)	8E	13–4
ADD A	ADD A,A	87	13–6
ADD B	ADD A,B	80	13–6
ADD C	ADD A,C	81	13–6
ADD D	ADD A,D	82	13–6
ADD E	ADD A,E	83	13–6
ADD H	ADD A,H	84	13–6
ADD L	ADD A,L	85	13–6
ADD M	ADD A,(HL)	86	13–6
ADI immed	ADD A,immed	C6 ib	13–6
ANA A	AND A	A7	13–8
ANA B	AND B	A0	13–8
ANA C	AND C	A1	13–8
ANA D	AND D	A2	13–8
ANA E	AND E	A3	13–8
ANA H	AND H	A4	13–8
ANA L	AND L	A5	13–8
ANA M	AND (HL)	A6	13–8
ANI immed	AND immed	E6 ib	13–8
CALL addr	CALL addr	CD al ah	13–13
CC addr	CALL C,addr	DC al ah	13–13
CM addr	CALL M,addr	FC al ah	13–13
CMA	CPL	2F	13–22
CMC	CCF	3F	13–15
CMP A	CP A	BF	13–16
CMP B	CP B	B8	13–16
CMP C	CP C	B9	13–16
CMP D	CP D	BA	13–16
CMP E	CP E	BB	13–16
CMP H	CP H	BC	13–16
CMP L	CP L	BD	13–16
CMP M	CP (HL)	BE	13–16

8080DL	Zilog	Disassembly	Page
CNC addr	CALL NC,addr	D4 al ah	13–13
CNZ addr	CALL NZ,addr	C4 al ah	13–13
CP addr	CALL P,addr	F4 al ah	13–13
CPE addr	CALL PE,addr	EC al ah	13–13
CPI immed	CP immed	FE ib	13–16
CPO addr	CALL PO,addr	E4 al ah	13–13
CZ addr	CALL Z,addr	CC al ah	13–13
DAA	DAA	27	13–23
DAD B	ADD HL,BC	09	13–6
DAD D	ADD HL,DE	19	13–6
DAD H	ADD HL,HL	29	13–6
DAD SP	ADD HL,SP	39	13–6
DCR A	DEC A	3D	13–25
DCR B	DEC B	05	13–25
DCR C	DEC C	0D	13–25
DCR D	DEC D	15	13–25
DCR E	DEC E	1D	13–25
DCR H	DEC H	25	13–25
DCR L	DEC L	2D	13–25
DCR M	DEC (HL)	35	13–25
DCX B	DEC BC	0B	13–25
DCX D	DEC DE	1B	13–25
DCX H	DEC HL	2B	13–25
DCX SP	DEC SP	3B	13–25
DI	DI	F3	13–27
EI	EI	FB	13–29
HLT	HALT	76	13–32
IN port	IN A,(port)	DB pb	13–35
INR A	INC A	3C	13–38
INR B	INC B	04	13–38
INR C	INC C	0C	13–38
INR D	INC D	14	13–38
INR E	INC E	1C	13–38
INR H	INC H	24	13–38
INR L	INC L	2C	13–38
INR M	INC (HL)	34	13–38
INX B	INC BC	03	13–38
INX D	INC DE	13	13–38
INX H	INC HL	23	13–38
INX SP	INC SP	33	13–38
JC addr	JP C,addr	DA al ah	13–44
JM addr	JP M,addr	FA al ah	13–44
JMP addr	JP addr	C3 al ah	13–44
JNC addr	JP NC,addr	D2 al ah	13–44
JNZ addr	JP NZ,addr	C2 al ah	13–44
JP addr	JP P,addr	F2 al ah	13–44
JPE addr	JP PE,addr	EA al ah	13–44
JPO addr	JP PO,addr	E2 al ah	13–44

8080DL	Zilog	Disassembly	Page
JZ addr	JP Z,addr	CA al ah	13–44
LDA addr	LD A,(addr)	3A al ah	13–48
LDAX B	LD A,(BC)	0A	13–48
LDAX D	LD A,(DE)	1A	13–48
LHLD addr	LD HL,(addr)	2A al ah	13–48
LXI B,imwrd	LD BC,imwrd	01 il ih	13–48
LXI D,imwrd	LD DE,imwrd	11 il ih	13–48
LXI H,imwrd	LD HL,imwrd	21 il ih	13–48
LXI SP,imwrd	LD SP,imwrd	31 il ih	13–48
MOV A,A	LD A,A	7F	13–48
MOV A,B	LD A,B	78	13–48
MOV A,C	LD A,C	79	13–48
MOV A,D	LD A,D	7A	13–48
MOV A,E	LD A,E	7B	13–48
MOV A,H	LD A,H	7C	13–48
MOV A,L	LD A,L	7D	13–48
MOV A,M	LD A,(HL)	7E	13–48
MOV B,A	LD B,A	47	13–48
MOV B,B	LD B,B	40	13–48
MOV B,C	LD B,C	41	13–48
MOV B,D	LD B,D	42	13–48
MOV B,E	LD B,E	43	13–48
MOV B,H	LD B,H	44	13–48
MOV B,L	LD B,L	45	13–48
MOV B,M	LD B,(HL)	46	13–48
MOV C,A	LD C,A	4F	13–48
MOV C,B	LD C,B	48	13–48
MOV C,C	LD C,C	49	13–48
MOV C,D	LD C,D	4A	13–48
MOV C,E	LD C,E	4B	13–48
MOV C,H	LD C,H	4C	13–48
MOV C,L	LD C,L	4D	13–48
MOV C,M	LD C,(HL)	4E	13–48
MOV D,A	LD D,A	57	13–48
MOV D,B	LD D,B	50	13–48
MOV D,C	LD D,C	51	13–48
MOV D,D	LD D,D	52	13–48
MOV D,E	LD D,E	53	13–48
MOV D,H	LD D,H	54	13–48
MOV D,L	LD D,L	55	13–48
MOV D,M	LD D,(HL)	56	13–48
MOV E,A	LD E,A	5F	13–48
MOV E,B	LD E,B	58	13–48
MOV E,C	LD E,C	59	13–48
MOV E,D	LD E,D	5A	13–48
MOV E,E	LD E,E	5B	13–48
MOV E,H	LD E,H	5C	13–48
MOV E,L	LD E,L	5D	13–48
MOV E,M	LD E,(HL)	5E	13–48
MOV H,A	LD H,A	67	13–48

8080DL	Zilog	Disassembly	Page
MOV H,B	LD H,B	60	13–48
MOV H,C	LD H,C	61	13–48
MOV H,D	LD H,D	62	13–48
MOV H,E	LD H,E	63	13–48
MOV H,H	LD H,H	64	13–48
MOV H,L	LD H,L	65	13–48
MOV H,M	LD H,(HL)	66	13–48
MOV L,A	LD L,A	6F	13–48
MOV L,B	LD L,B	68	13–48
MOV L,C	LD L,C	69	13–48
MOV L,D	LD L,D	6A	13–48
MOV L,E	LD L,E	6B	13–48
MOV L,H	LD L,H	6C	13–48
MOV L,L	LD L,L	6D	13–48
MOV L,M	LD L,(HL)	6E	13–48
MOV M,A	LD (HL),A	77	13–48
MOV M,B	LD (HL),B	70	13–48
MOV M,C	LD (HL),C	71	13–48
MOV M,D	LD (HL),D	72	13–48
MOV M,E	LD (HL),E	73	13–48
MOV M,H	LD (HL),H	74	13–48
MOV M,L	LD (HL),L	75	13–48
MVI A,immed	LD A,immed	3E ib	13–48
MVI B,immed	LD B,immed	06 ib	13–48
MVI C,immed	LD C,immed	0E ib	13–48
MVI D,immed	LD D,immed	16 ib	13–48
MVI E,immed	LD E,immed	1E ib	13–48
MVI H,immed	LD H,immed	26 ib	13–48
MVI L,immed	LD L,immed	2E ib	13–48
MVI M,immed	LD (HL),immed	36 ib	13–48
NOP	NOP	00	13–61
ORA A	OR A	B7	13–62
ORA B	OR B	B0	13–62
ORA C	OR C	B1	13–62
ORA D	OR D	B2	13–62
ORA E	OR E	B3	13–62
ORA H	OR H	B4	13–62
ORA L	OR L	B5	13–62
ORA M	OR (HL)	B6	13–62
ORI immed	OR immed	F6 ib	13–62
OUT port	OUT (port),A	D3 pb	13–70
PCHL	JP (HL)	E9	13–44
POP B	POP BC	C1	13–75
POP D	POP DE	D1	13–75
POP H	POP HL	E1	13–75
POP PSW	POP AF	F1	13–75
PUSH B	PUSH BC	C5	13–76
PUSH D	PUSH DE	D5	13–76
PUSH H	PUSH HL	E5	13–76
PUSH PSW	PUSH AF	F5	13–76
RAL	RLA	17	13–86

8080DL	Zilog	Disassembly	Page
RAR	RRA	1F	13–93
RC	RET C	D8	13–80
RET	RET	C9	13–80
RLC	RLCA	07	13–89
RM	RET M	F8	13–80
RNC	RET NC	D0	13–80
RNZ	RET NZ	C0	13–80
RP	RET P	F0	13–80
RPE	RET PE	E8	13–80
RPO	RET PO	E0	13–80
RRC	RRCA	0F	13–96
RST 0	RST 00	C7	13–98
RST 1	RST 08	CF	13–98
RST 2	RST 10	D7	13–98
RST 3	RST 18	DF	13–98
RST 4	RST 20	E7	13–98
RST 5	RST 28	EF	13–98
RST 6	RST 30	F7	13–98
RST 7	RST 38	FF	13–98
RZ	RET Z	C8	13–80
SBB A	SBC A,A	9F	13–99
SBB B	SBC A,B	98	13–99
SBB C	SBC A,C	99	13–99
SBB D	SBC A,D	9A	13–99
SBB E	SBC A,E	9B	13–99
SBB H	SBC A,H	9C	13–99
SBB L	SBC A,L	9D	13–99
SBB M	SBC A,(HL)	9E	13–99
SBI immed	SBC A,immed	DE ib	13–99
SHLD addr	LD (addr),HL	22 al ah	13–48
SPHL	LD SP,HL	F9	13–48
STA addr	LD (addr),A	32 al ah	13–48
STAX B	LD (BC),A	02	13–48
STAX D	LD (DE),A	12	13–48
STC	SCF	37	13–101
SUB A	SUB A	97	13–112
SUB B	SUB B	90	13–112
SUB C	SUB C	91	13–112
SUB D	SUB D	92	13–112
SUB E	SUB E	93	13–112
SUB H	SUB H	94	13–112
SUB L	SUB L	95	13–112
SUB M	SUB (HL)	96	13–112
SUI immed	SUB immed	D6 ib	13–112
XCHG	EX DE,HL	EB	13–30

8080DL	Zilog	Disassembly	Page
XRA A	XOR A	AF	13–117
XRA B	XOR B	A8	13–117
XRA C	XOR C	A9	13–117
XRA D	XOR D	AA	13–117
XRA E	XOR E	AB	13–117
XRA H	XOR H	AC	13–117
XRA L	XOR L	AD	13–117
XRA M	XOR (HL)	AE	13–117
XRI immed	XOR immed	EE ib	13–117
XTHL	EX (SP),HL	E3	13–30

Byte Tyme Flag Table

	Size	Bytes	Tymes	S	Z	H	P	N	C	Condition
ADC A,reg	8	1	4	+	+	+	+	0	+	
ADC A,(HL)	8	1	7 (4,3)	+	+	+	+	0	+	
ADC A,(Ii+d)	8	3	19 (4,4,3,5,3)	+	+	+	+	0	+	
ADC A,immed	8	2	7 (4,3)	+	+	+	+	0	+	
ADC HL,regpr	16	2	15 (4,4,4,3)	+	+	x	+	0	+	
ADD A,reg	8	1	4	+	+	+	+	0	+	
ADD A,(HL)	8	1	7 (4,3)	+	+	+	+	0	+	
ADD A,(Ii+d)	8	3	19 (4,4,3,5,3)	+	+	+	+	0	+	
ADD A,immed	8	2	7 (4,3)	+	+	+	+	0	+	
ADD HL,regpr	16	1	11 (4,4,3)	-	-	x	-	0	+	
ADD Ii,regpr	16	2	11 (4,4,3)	-	-	x	-	0	+	
AND reg	8	1	4	+	+	1	+	0	0	
AND (HL)	8	1	7 (4,3)	+	+	1	+	0	0	
AND (Ii+d)	8	3	19 (4,4,3,5,3)	+	+	1	+	0	0	
AND immed	8	2	7 (4,3)	+	+	1	+	0	0	
BIT bit,reg	1	2	8 (4,4)	u	x	1	u	0	-	
BIT bit,(HL)	1	2	12 (4,4,4)	u	x	1	u	0	-	
BIT bit,(Ii+d)	1	4	20 (4,4,3,5,4)	u	x	1	u	0	-	
CALL addr	abs	3	17 (4,3,4,3,3)	-	-	-	-	-	-	
CALL cond,addr	abs	3	17 (4,3,4,3,3)	-	-	-	-	-	-	True
			10 (4,3,3)	-	-	-	-	-	-	False
CCF	1	1	4	-	-	x	-	0	x	
CP reg	8	1	4	+	+	+	+	1	+	
CP (HL)	8	1	7 (4,3)	+	+	+	+	1	+	
CP (Ii+d)	8	3	19 (4,4,3,5,3)	+	+	+	+	1	+	
CP immed	8	2	7 (4,3)	+	+	+	+	1	+	
CPD	8	2	16 (4,4,3,5)	+	+	+	x	1	-	
CPDR	8	2	21 (4,4,3,5,5)	+	+	+	1	1	-	BC <> 0
			16 (4,4,3,5)	+	+	+	0	1	-	
CPI	8	2	16 (4,4,3,5)	+	+	+	x	1	-	
CPIR	8	2	21 (4,4,3,5,5)	+	+	+	1	1	-	BC <> 0
			16 (4,4,3,5)	+	+	+	0	1	-	BC = 0
CPL	8	1	4	-	-	1	-	1	-	
DAA	8	1	4	+	+	+	+	-	+	
DEC reg	8	1	4	+	+	+	+	1	-	
DEC (HL)	8	1	11 (4,4,3)	+	+	+	+	1	-	
DEC (Ii+d)	8	3	23 (4,4,3,5,4,3)	+	+	+	+	1	-	
DEC regpr	16	1	6	-	-	-	-	-	-	
DEC Ii	16	2	10 (4,6)	-	-	-	-	-	-	
DI	ctl	1	4	-	-	-	-	-	-	
DJNZ disp	rel	2	13 (5,3,5)	-	-	-	-	-	-	B <> 0
			8 (5,3)	-	-	-	-	-	-	B = 0
EI	ctl	1	4	-	-	-	-	-	-	

	Size	Bytes	Tymes	S	Z	H	P	N	C	Condition
EX DE,HL	16	1	4	-	-	-	-	-	-	
EX (SP),HL	16	1	19 (4,3,4,3,5)	-	-	-	-	-	-	
EX (SP),Ii	16	2	23 (4,4,3,4,3,5)	-	-	-	-	-	-	
EX AF,AF'	16	1	4	x	x	x	x	x	x	
EXX	16	1	4	-	-	-	-	-	-	
HALT	ctl	1	4	-	-	-	-	-	-	
IM mode	ctl	2	8 (4,4)	-	-	-	-	-	-	
IN A,(port)	8	2	11 (4,3,4)	-	-	-	-	-	-	
IN reg,(C)	8	2	12 (4,4,4)	+	+	0	+	0	-	
~ IN0 reg,(port)	8	3	1	+	+	0	+	0	-	
INC reg	8	1	4	+	+	+	+	0	-	
INC (HL)	8	1	11 (4,4,3)	+	+	+	+	0	-	
INC (Ii+d)	8	3	23 (4,4,3,5,4,3)	+	+	+	+	0	-	
INC regpr	16	1	6	-	-	-	-	-	-	
INC Ii	16	2	10 (4,6)	-	-	-	-	-	-	
IND	8	2	16 (4,5,3,4)	u	x	u	u	1	-	
INDR	8	2	21 (4,5,3,4,5)	u	0	u	u	1	-	B <> 0
			16 (4,5,3,4)	u	1	u	u	1	-	B = 0
INI	8	2	16 (4,5,3,4)	u	x	u	u	1	-	
INIR	8	2	21 (4,5,3,4,5)	u	0	u	u	1	-	B <> 0
			16 (4,5,3,4)	u	1	u	u	1	-	B = 0
JP addr	abs	3	10 (4,3,3)	-	-	-	-	-	-	
JP (HL)	abs	1	4	-	-	-	-	-	-	
JP (Ii)	abs	2	8 (4,4)	-	-	-	-	-	-	
JP cond,addr	abs	3	10 (4,3,3)	-	-	-	-	-	-	True
			10 (4,3,3)	-	-	-	-	-	-	False
JR disp	rel	2	12 (4,3,5)	-	-	-	-	-	-	
JR cond,disp	rel	2	12 (4,3,5)	-	-	-	-	-	-	True
			7 (4,3)	-	-	-	-	-	-	False
LD reg,reg	8	1	4	-	-	-	-	-	-	
LD reg,(HL)	8	1	7 (4,3)	-	-	-	-	-	-	
LD (HL),reg	8	1	7 (4,3)	-	-	-	-	-	-	
LD reg,immed	8	2	7 (4,3)	-	-	-	-	-	-	
LD (HL),immed	8	2	10 (4,3,3)	-	-	-	-	-	-	
LD reg,(Ii+d)	8	3	19 (4,4,3,5,3)	-	-	-	-	-	-	
LD (Ii+d),reg	8	3	19 (4,4,3,5,3)	-	-	-	-	-	-	
LD (Ii+d),immed	8	4	19 (4,4,3,5,3)	-	-	-	-	-	-	
LD (regpr),A	8	1	7 (4,3)	-	-	-	-	-	-	
LD A,(regpr)	8	1	7 (4,3)	-	-	-	-	-	-	
LD (addr),A	8	3	13 (4,3,3,3)	-	-	-	-	-	-	
LD A,(addr)	8	3	13 (4,3,3,3)	-	-	-	-	-	-	
LD I,A	8	2	9 (4,5)	-	-	-	-	-	-	
LD R,A	8	2	9 (4,5)	-	-	-	-	-	-	
LD A,I	8	2	9 (4,5)	x	x	0	x	0	-	
LD A,R	8	2	9 (4,5)	x	x	0	x	0	-	
LD regpr,imwrd	16	3	10 (4,3,3)	-	-	-	-	-	-	
LD Ii,imwrd	16	4	14 (4,4,3,3)	-	-	-	-	-	-	
LD (addr),HL	16	3	16 (4,3,3,3,3)	-	-	-	-	-	-	
LD HL,(addr)	16	3	16 (4,3,3,3,3)	-	-	-	-	-	-	
LD (addr),regpr	16	4	20 (4,4,3,3,3,3)	-	-	-	-	-	-	
LD regpr,(addr)	16	4	20 (4,4,3,3,3,3)	-	-	-	-	-	-	
LD (addr),Ii	16	4	20 (4,4,3,3,3,3)	-	-	-	-	-	-	

	Size	Bytes	Tymes	S	Z	H	P	N	C	Condition
LD Ii,(addr)	16	4	20 (4,4,3,3,3,3)	-	-	-	-	-	-	
LD SP,HL	16	1	6	-	-	-	-	-	-	
LD SP,Ii	16	2	10 (4,6)	-	-	-	-	-	-	
LDD	8	2	16 (4,4,3,5)	-	-	0	x	0	-	
LDDR	8	2	21 (4,4,3,5,5)	-	-	0	1	0	-	BC <> 0
			16 (4,4,3,5)	-	-	0	0	0	-	BC = 0
LDI	8	2	16 (4,4,3,5)	-	-	0	x	0	-	
LDIR	8	2	21 (4,4,3,5,5)	-	-	0	1	0	-	BC <> 0
			16 (4,4,3,5)	-	-	0	0	0	-	BC = 0
~ MLT regpr	8	2	17	-	-	-	-	-	-	
NEG	8	2	8 (4,4)	+	+	+	+	1	x	
NOP	ctl	1	4	-	-	-	-	-	-	
OR reg	8	1	4	+	+	1	+	0	0	
OR (HL)	8	1	7 (4,3)	+	+	1	+	0	0	
OR (Ii+d)	8	3	19 (4,4,3,5,3)	+	+	1	+	0	0	
OR immed	8	2	7 (4,3)	+	+	1	+	0	0	
~ OTDM	8	2	14	x	x	x	x	x	x	
~ OTDMR	8	2	16	x	x	x	x	x	x	B <> 0
			14	0	1	0	1	1	0	B = 0
OTDR	8	2	21 (4,5,3,4,5)	u	x	u	u	1	u	B <> 0
			16 (4,5,3,4)	u	1	u	u	1	u	B = 0
OTIM	8	2	14	x	x	x	x	x	x	
~ OTIMR	8	2	16	x	x	x	x	x	x	B <> 0
			14	0	1	0	1	0	0	B = 0
OTIR	8	2	21 (4,5,3,4,5)	u	x	u	u	0	u	B <> 0
			16 (4,5,3,4)	u	1	u	u	0	u	B = 0
OUT (port),A	8	2	11 (4,3,4)	-	-	-	-	u	-	
OUT (C),reg	8	2	12 (4,4,4)	-	-	-	-	u	-	
~ OUT0 (port),reg	8	3	13	-	-	-	-	-	-	
OUTD	8	2	16 (4,5,3,4)	u	x	u	u	1	-	
OUTI	8	2	16 (4,5,3,4)	u	x	u	u	1	-	
POP regpr	16	1	10 (4,3,3)	-	-	-	-	-	-	
POP AF	16	1	10 (4,3,3)	x	x	x	x	x	x	
POP Ii	16	2	14 (4,4,3,3)	-	-	-	-	-	-	
PUSH regpr	16	1	11 (5,3,3)	-	-	-	-	-	-	
PUSH Ii	16	2	15 (4,5,3,3)	-	-	-	-	-	-	
RES bit,reg	1	2	8 (4,4)	-	-	-	-	-	-	
RES bit,(HL)	1	2	12 (4,4,4)	-	-	-	-	-	-	
RES bit,(Ii+d)	1	4	20 (4,4,3,5,4)	-	-	-	-	-	-	
RET	abs	1	10 (4,3,3)	-	-	-	-	-	-	
RET cond	abs	1	11 (5,3,3)	-	-	-	-	-	-	True
			5	-	-	-	-	-	-	False
RETI	abs	2	14 (4,4,3,3)	-	-	-	-	-	-	
RETN	abs	2	14 (4,4,3,3)	-	-	-	-	-	-	

	Size	Bytes	Tymes	S	Z	H	P	N	C	Condition
RL reg	1	2	8 (4,4)	+	+	0	+	0	x	
RL (HL)	1	2	15 (4,4,4,3)	+	+	0	+	0	x	
RL (Ii+d)	1	4	23 (4,4,3,5,4,3)	+	+	0	+	0	x	
RLA	1	1	4	+	+	0	+	0	x	
RLC reg	1	2	8 (4,4)	+	+	0	+	0	x	
RLC (HL)	1	2	15 (4,4,4,3)	+	+	0	+	0	x	
RLC (Ii+d)	1	4	23 (4,4,3,5,4,3)	+	+	0	+	0	x	
RLCA	1	1	4	-	-	0	+	0	x	
RLD	4	2	18 (4,4,3,4,3)	+	+	0	+	0	-	
RR reg	1	2	8 (4,4)	+	+	0	+	0	x	
RR (HL)	1	2	15 (4,4,4,3)	+	+	0	+	0	x	
RR (Ii+d)	1	4	23 (4,4,3,5,4,3)	+	+	0	+	0	x	
RRA	1	1	4	+	+	0	+	0	x	
RRC reg	1	2	8 (4,4)	+	+	0	+	0	x	
RRC (HL)	1	2	15 (4,4,4,3)	+	+	0	+	0	x	
RRC (Ii+d)	1	4	23 (4,4,3,5,4,3)	+	+	0	+	0	x	
RRCA	1	1	4	+	+	0	+	0	x	
RRD	4	2	18 (4,4,3,4,3)	+	+	0	+	0	-	
RST vector	abs	1	11 (5,3,3)	-	-	-	-	-	-	
SBC A,reg	8	1	4	+	+	+	+	1	+	
SBC A,(HL)	8	1	7 (4,3)	+	+	+	+	1	+	
SBC A,(Ii+d)	8	3	19 (4,4,3,5,3)	+	+	+	+	1	+	
SBC A,immed	8	2	7 (4,3)	+	+	+	+	1	+	
SBC HL,regpr	16	2	15 (4,4,4,3)	+	+	x	+	1	+	
SCF	1	1	4	-	-	0	-	0	1	
SET bit,reg	1	2	8 (4,4)	-	-	-	-	-	-	
SET bit,(HL)	1	2	12 (4,4,4)	-	-	-	-	-	-	
SET bit,(Ii+d)	1	4	20 (4,4,3,5,4)	-	-	-	-	-	-	
SLA reg	1	2	8 (4,4)	+	+	0	+	0	x	
SLA (HL)	1	2	15 (4,4,4,3)	+	+	0	+	0	x	
SLA (Ii+d)	1	4	26 (4,4,3,5,4,3)	+	+	0	+	0	x	
~ SLP	ctl	2	8	-	-	-	-	-	-	
SRA reg	1	2	8 (4,4)	+	+	0	+	0	x	
SRA (HL)	1	2	15 (4,4,4,3)	+	+	0	+	0	x	
SRA (Ii+d)	1	4	26 (4,4,3,5,4,3)	+	+	0	+	0	x	
SRL reg	1	2	8 (4,4)	+	+	0	+	0	x	
SRL (HL)	1	2	15 (4,4,4,3)	+	+	0	+	0	x	
SRL (Ii+d)	1	4	26 (4,4,3,5,4,3)	+	+	0	+	0	x	
SUB reg	8	1	4	+	+	+	+	1	+	
SUB (HL)	8	1	7 (4,3)	+	+	+	+	1	+	
SUB (Ii+d)	8	3	19 (4,4,3,5,3)	+	+	+	+	1	+	
SUB immed	8	2	7 (4,3)	+	+	+	+	1	+	
~ TST reg	8	2	7	+	+	1	+	0	0	
~ TST (HL)	8	2	10	+	+	1	+	0	0	
~ TST immed	8	3	9	+	+	1	+	0	0	
~ TSTIO immed	8	3	12	+	+	1	+	0	0	

	Size	Bytes	Tymes	S	Z	H	P	N	C	Condition
XOR reg	8	1	4	+	+	1	+	0	0	
XOR (HL)	8	1	7 (4,3)	+	+	1	+	0	0	
XOR (Ii+d)	8	3	19 (4,4,3,5,3)	+	+	1	+	0	0	
XOR immed	8	2	7 (4,3)	+	+	1	+	0	0	

Size: The operation size in bits (1, 8, or 16):

1	A bit operand.
4	A nybble operand.
8	A byte operand.
16	A word operand.
abs	An absolute relocation operand.
ctl	A cpu control operand.
rel	A relative relocation operand.

Bytes: The operand size in bytes.

Tymes: The number of tymes (T-states) required for the operation:

Flags:

S	Sign flag
Z	Zero flag
H	Half-carry flag
P	Parity/overflow flag
N	Subtraction flag
C	Carry flag

–	Not affected.
+	Altered in a normal manner.
u	Altered in an undefined manner.
x	Altered in a special manner.
0	Cleared.
1	Set.

Condition: The condition of any conditional or looping operand.

Base Conversion

Signed and Unsigned Values

A value may be signed or unsigned.

Unsigned values are used for addressing, counting, and general data. Unsigned values have the following ranges:

8-Bit Values:

$1111,1111_B$	377_O	255_D	FF_H
$1111,1110_B$	376_O	254_D	FE_H
…	…	…	…
$0000,0001_B$	001_O	1_D	01_H
$0000,0000_B$	000_O	0_D	00_H

16-Bit Values:

$1111,1111,1111,1111_B$	$177,777_O$	$65,535_D$	$FFFF_H$
$1111,1111,1111,1110_B$	$177,776_O$	$65,534_D$	$FFFE_H$
…	…	…	…
$0000,0000,0000,0001_B$	$000,001_O$	1_D	0001_H
$0000,0000,0000,0000_B$	$000,000_O$	0_D	0000_H

Signed values are used for mathematics and other computational functions. Signed values have the following ranges:

8-Bit Values:

$0111,1111_B$	177_O	$+127_D$	$7F_H$
$0111,1110_B$	176_O	$+126_D$	$7E_H$
…	…	…	…
$0000,0001_B$	001_O	$+1_D$	01_H
$0000,0000_B$	000_O	0_D	00_H
$1111,1111_B$	377_O	-1_D	FF_H
$1111,1110_B$	378_O	-2_D	FE_H
…	…	…	…
$1000,0001_B$	201_O	-127_D	81_H
$1000,0000_B$	200_O	-128_D	80_H

16-Bit Values:

$0111,1111,1111,1111_B$	$077,777_O$	$+32,767_D$	$FFFF_H$
$0111,1111,1111,1110_B$	$077,777_O$	$+32,766_D$	$7FFE_H$
…	…	…	…
$0000,0000,0000,0001_B$	$000,001_O$	$+1_D$	0001_H
$0000,0000,0000,0000_B$	$000,000_O$	0_D	0000_H
$1111,1111,1111,1111_B$	$177,777_O$	-1_D	$FFFF_H$
$1111,1111,1111,1111_B$	$177,776_O$	-2_D	$FFFE_H$
…	…	…	…
$1000,0000,0000,0001_B$	$100,001_O$	$-32,767_D$	8001_H
$1000,0000,0000,0000_B$	$100,000_O$	$-32,768_D$	8000_H

The most significant bit of signed values is the sign bit. If the sign bit is 0, the value is positive, if the sign bit is 1, the value is negative.

At first glance, there appears to be an inconsistency in the binary, octal, and hexadecimal signed numbers as they shift from a value of 0 to a value of –1. This inconsistency is not real. If you think of each number as a value X, then the next smaller number has a value of X – 1. In binary, $0000_B - 0001_B = 1111_B$. It helps if you imagine that there is a 1 before the minuend: the equation then becomes: $10000_B - 0001_B = 1111_B$.

When converting between binary and octal, binary and hexadecimal, or octal and hexadecimal, the signs automatically fall into place. No special consideration need be given as to whether a value is unsigned or signed.

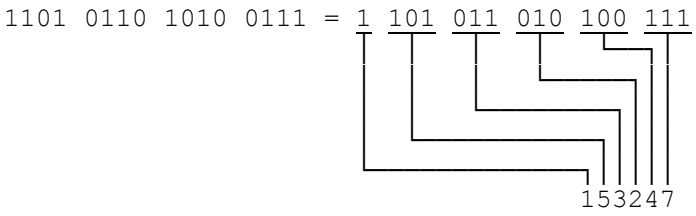
In assembly, decimal values are the odd ones. Therefore, when converting between decimal and binary, decimal and octal, or decimal and hexadecimal, it becomes critical that you know if the value is unsigned or signed. More specifically, whether or not the value is an unsigned negative value.

Binary to Octal Conversion

To convert a value from binary to octal, break the binary value into groups of three bits from right to left and convert each group of three bits into the corresponding octal digit.

000 = 0	010 = 2	100 = 4	110 = 6
001 = 1	011 = 3	101 = 5	111 = 7

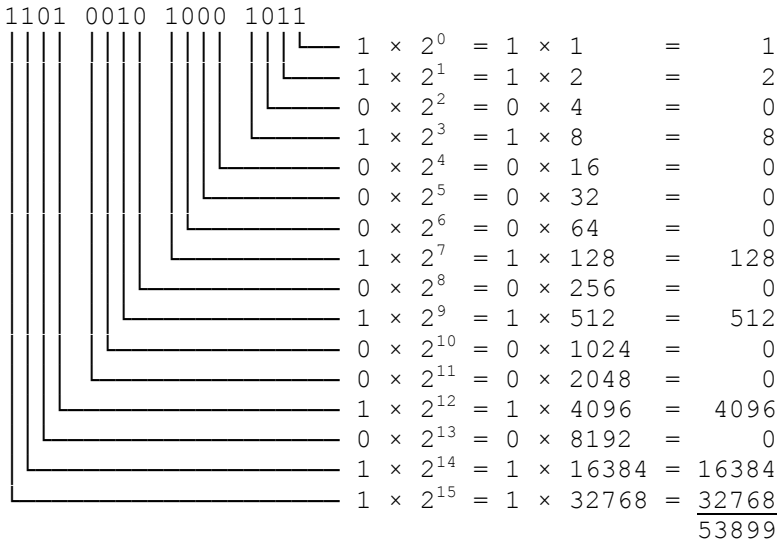
For example, to convert $1101,0110,1010,0111_B$ to $153,247_O$:



Unsigned Binary to Decimal Conversion

To convert an unsigned value from binary to decimal, multiply each bit by its appropriate power of 2 and add the results.

For example, to convert $1101,0010,1000,1011_B$ to $53,899_D$:



Signed Binary to Decimal Conversion

To convert a signed value from binary to decimal, determine first if the value is positive or negative. The most significant bit of an 8- or 16-bit positive value is 0, while that of a negative value is 1. Be certain the value is either 8 or 16 bits: $1001,1100,1111,0100_B$ is a negative value, while $1001,1100,1111_B$ is not ($1001,1100,1111_B$ is actually $0000,1001,1100,1111_B$).

If a positive value, multiply each bit by its appropriate power of 2 and add the results.

For example, to convert $0110,0100,1011,0011_B$ to $25,779_D$:

```
0110 0100 1011 0011
|||| |||| |||| ||||
|||| |||| |||| |||└── 1 × 2^0  = 1 × 1     =     1
|||| |||| |||| ||└─── 1 × 2^1  = 1 × 2     =     2
|||| |||| |||| |└──── 0 × 2^2  = 0 × 4     =     0
|||| |||| |||| └───── 0 × 2^3  = 0 × 8     =     0
|||| |||| |||└─────── 1 × 2^4  = 1 × 16    =    16
|||| |||| ||└──────── 1 × 2^5  = 1 × 32    =    32
|||| |||| |└───────── 0 × 2^6  = 0 × 64    =     0
|||| |||| └────────── 1 × 2^7  = 1 × 128   =   128
|||| |||└──────────── 0 × 2^8  = 0 × 256   =     0
|||| ||└───────────── 0 × 2^9  = 0 × 512   =     0
|||| |└────────────── 1 × 2^10 = 1 × 1024  =  1024
|||| └─────────────── 0 × 2^11 = 0 × 2048  =     0
|||└───────────────── 0 × 2^12 = 0 × 4096  =     0
||└────────────────── 1 × 2^13 = 1 × 8192  =  8192
|└─────────────────── 1 × 2^14 = 1 × 16384 = 16384
└──────────────────── 0 × 2^15 = 0 × 32768 =     0
                                            -------
                                            25779
```

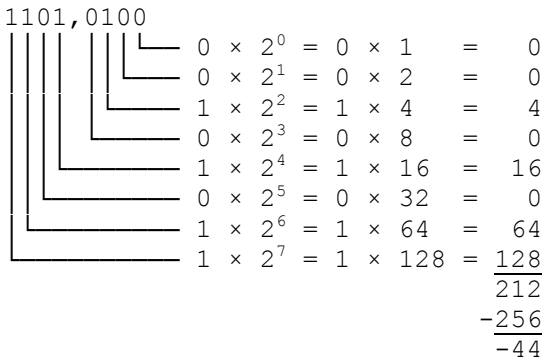
If a 16-bit negative value, multiply each bit by its appropriate power of 2 and add the results, then subtract 65,536.

For example, to convert $1010,0011,1110,0001_B$ to $-23,583_D$:

```
1010 0011 1110 0001
|||| |||| |||| ||||
|||| |||| |||| |||└── 1 × 2^0  = 1 × 1     =     1
|||| |||| |||| ||└─── 0 × 2^1  = 0 × 2     =     0
|||| |||| |||| |└──── 0 × 2^2  = 0 × 4     =     0
|||| |||| |||| └───── 0 × 2^3  = 0 × 8     =     0
|||| |||| |||└─────── 0 × 2^4  = 0 × 16    =     0
|||| |||| ||└──────── 1 × 2^5  = 1 × 32    =    32
|||| |||| |└───────── 1 × 2^6  = 1 × 64    =    64
|||| |||| └────────── 1 × 2^7  = 1 × 128   =   128
|||| |||└──────────── 1 × 2^8  = 1 × 256   =   256
|||| ||└───────────── 1 × 2^9  = 1 × 512   =   512
|||| |└────────────── 0 × 2^10 = 0 × 1024  =     0
|||| └─────────────── 0 × 2^11 = 0 × 2048  =     0
|||└───────────────── 0 × 2^12 = 0 × 4096  =     0
||└────────────────── 1 × 2^13 = 1 × 8192  =  8192
|└─────────────────── 0 × 2^14 = 0 × 16384 =     0
└──────────────────── 1 × 2^15 = 1 × 32768 = 32768
                                            -------
                                            41953
                                           -65536
                                           -------
                                           -23583
```

If an 8-bit negative value, multiply each bit by its appropriate power of 2 and add the results, then subtract 256.

For example, to convert $1101,0100_B$ to -44_D:

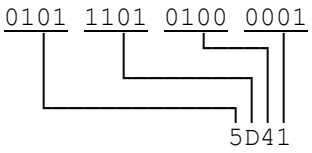


Binary to Hexadecimal Conversion

To convert a value from binary to hexadecimal, break the binary value into groups of four bits from right to left and convert each group of four bits into the corresponding hexadecimal digit.

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

For example, to convert $101,1101,0100,0001_B$ to $5D41_H$:

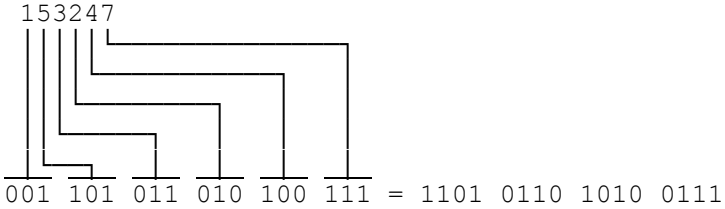


Octal to Binary Conversion

To convert a value from octal to binary, convert each octal digit into the corresponding group of three bits.

0 = 000	2 = 010	4 = 100	6 = 110
1 = 001	3 = 011	5 = 101	7 = 111

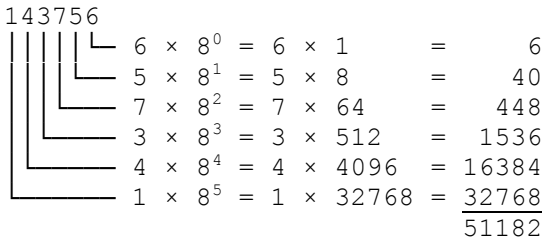
For example, to convert $153,247_O$: to $1101,0110,1010,0111_B$:



Unsigned Octal to Decimal Conversion

To convert an unsigned value from octal to decimal, multiply each octal digit by its appropriate power of 8 and add the results.

For example, to convert $143{,}756_O$ to $51{,}182_D$:



Signed Octal to Decimal Conversion

To convert a signed value from octal to decimal, determine first if the value is positive or negative.

8-bit octal values have 3 digits. If the most significant digit is 0 or 1, the value is positive. If the most significant digit is 2 or 3, the value is negative. If the most significant digit is more than 3, the value is 16-bit.

16-bit octal values have 6 digits. If the most significant digit is 0, the value is positive. If the most significant digit is 1, the value is negative. If the most significant digit is more than 1, the value is more than 16 bits.

If a positive 8- or 16-bit value, multiply each bit by its appropriate power of 8 and add the results.

For example, to convert $026{,}103_O$ to $+11{,}331_D$:

```
026103
|||||└── 3 × 8^0 = 3 × 1     =     31
||||└─── 0 × 8^1 = 0 × 8     =      0
|||└──── 1 × 8^2 = 1 × 64    =     64
||└───── 6 × 8^3 = 6 × 512   =   3072
|└────── 2 × 8^4 = 2 × 4096  =   8192
└─────── 0 × 8^5 = 0 × 32768 =      0
                              +11331
```

If a 16-bit negative value, multiply each bit by its appropriate power of 8 and add the results, then subtract 65,536.

For example, to convert $103{,}521_O$ to $–31{,}075_D$:

```
103521
|||||└── 1 × 8^0 = 1 × 1     =     1
||||└─── 2 × 8^1 = 2 × 8     =    16
|||└──── 5 × 8^2 = 5 × 64    =   320
||└───── 3 × 8^3 = 3 × 512   =  1356
|└────── 0 × 8^4 = 0 × 4096  =     0
└─────── 1 × 8^5 = 1 × 32768 = 32768
                               34461
                              -65536
                              -31075
```

If an 8-bit negative value, multiply each bit by its appropriate power of 2 and add the results, then subtract 256.

For example, to convert 246_O to $–90_D$:

```
246
|||└── 6 × 8^0 = 6 × 1  =   6
||└─── 4 × 8^1 = 4 × 8  =  32
|└──── 2 × 8^2 = 2 × 64 = 128
                         166
                        -256
                         -90
```

Octal to Hexadecimal Conversion

To convert a value from octal to hexadecimal is a two step process. First convert the octal value to binary, then convert the binary value to hexadecimal.

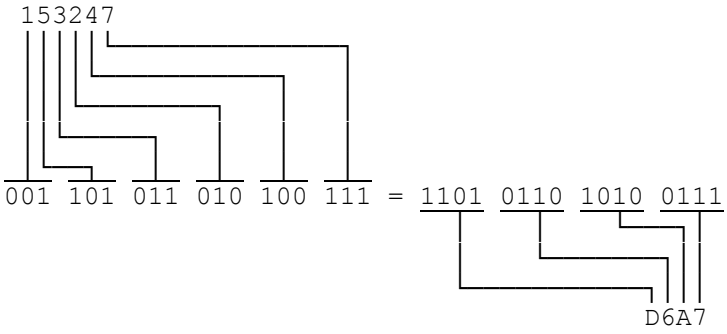
To convert a value from octal to binary, convert each octal digit into the corresponding group of three bits.

0 = 000	2 = 010	4 = 100	6 = 110
1 = 001	3 = 011	5 = 101	7 = 111

Then, to convert the binary value into hexadecimal, break the binary value into groups of four bits from right to left and convert each group of four bits into the corresponding hexadecimal digit.

0000 = 0	0100 = 4	1000 = 8	1100 = C
0001 = 1	0101 = 5	1001 = 9	1101 = D
0010 = 2	0110 = 6	1010 = A	1110 = E
0011 = 3	0111 = 7	1011 = B	1111 = F

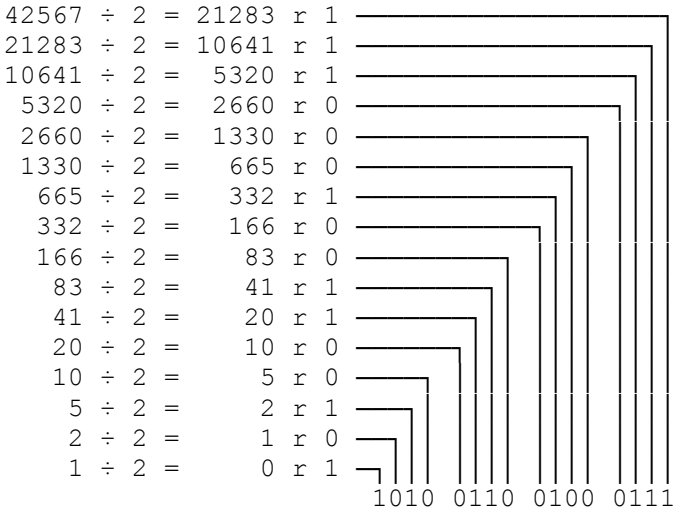
For example, to convert $153{,}247_O$: to $D6A7_H$:



Unsigned Decimal to Binary Conversion

To convert an unsigned value from decimal to binary, continuously integer divides by 2: the remainders of each division are the result, least significant bit first. Leading 0s may be added to achieve a 16- or 8-bit result.

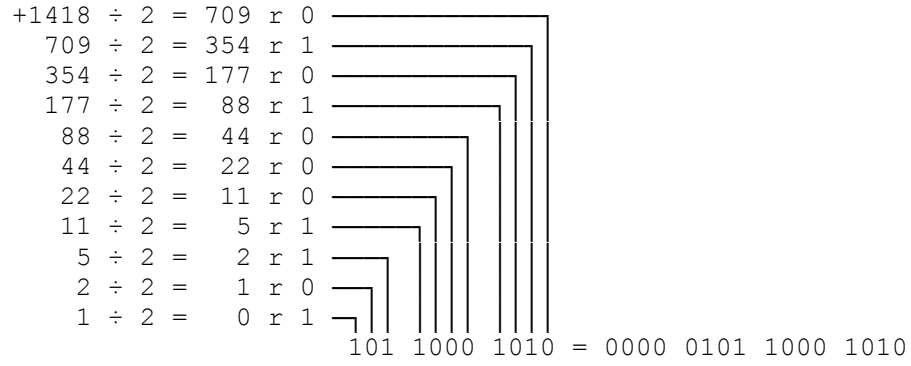
For example, to convert $42{,}567_D$ to $1010{,}0110{,}0100{,}0111_B$:



Signed Decimal to Binary Conversion

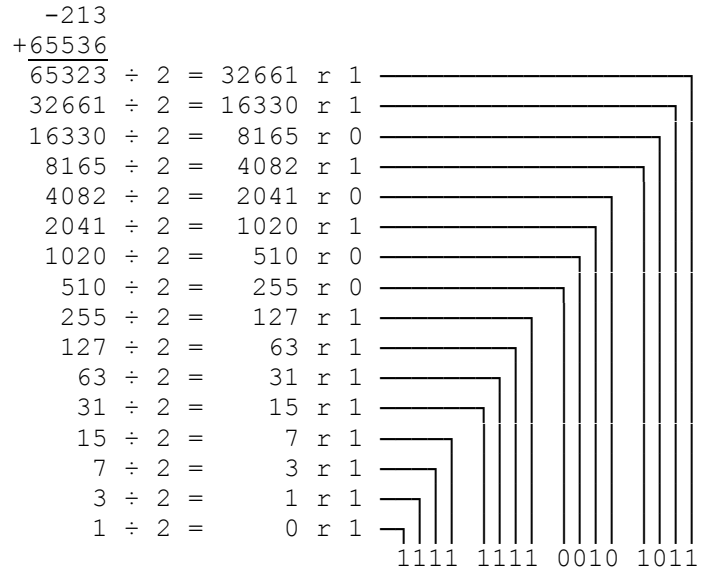
To convert a positive signed value from decimal to binary, continuously integer divides by 2: the remainders of each division are the result, least significant bit first. Leading 0s may be added to achieve a 16- or 8-bit value.

For example, to convert $+1418_D$ to $101,1000,1010_B$ ($0000,0101,1000,1010_B$):

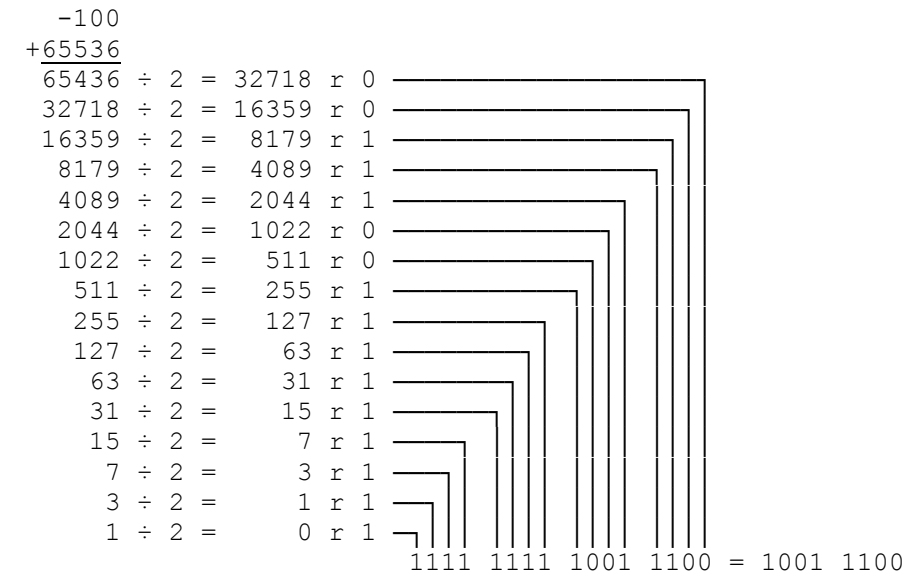


To convert a negative signed value from decimal to binary, first add 65536, then continuously integer divide by 2: the remainders of each division are the result, least significant bit first. If required, leading 1s must then be added to achieve a 16- or 8-bit value. If the most significant 8 bits are all 1s and the 8 least significant bits begin with a 1, then the most significant 8 bits may be dropped to produce an 8-bit value.

For example, to convert $–213_D$ to $1111,1111,0010,1011_B$:



Similarly, to convert -100_D to $1111,1111,1001,1100_B$ or $1001,1100_B$:

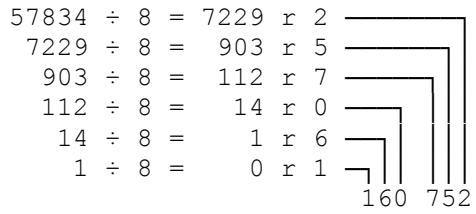


Notice that the last example produces a proper byte result: the eight most significant digits are 1111,1111 and the eight least significant digits begin with a 1. The 1111,1111 of the most significant digits are merely carrying the sign forward: there is actually an infinite string of “1”s proceeding a negative value, just as there are an infinite string of “0”s preceding a positive value.

Unsigned Decimal to Octal Conversion

To convert an unsigned value from decimal to octal, continuously integer divides by 8. The remainders of each division are the result, least significant digit first.

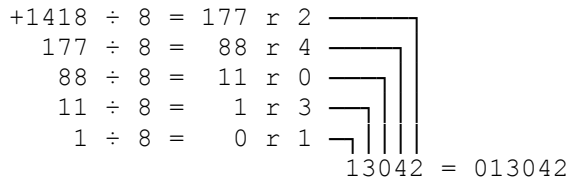
For Example, to convert $57{,}834_D$ to $160{,}752_O$:



Signed Decimal to Octal Conversion

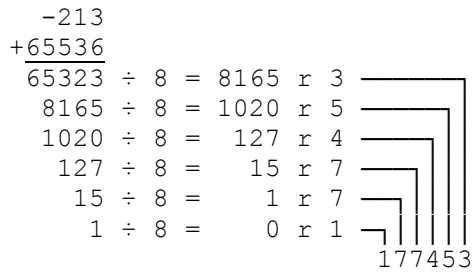
To convert a positive signed value from decimal to octal, continuously integer divides by 8: the remainders of each division are the result, least significant bit first.

For example, to convert $+1418_D$ to $013{,}042_O$:



To convert a negative signed value from decimal to binary, first add 65536, then continuously integer divide by 8: the remainders of each division are the result, least significant bit first.

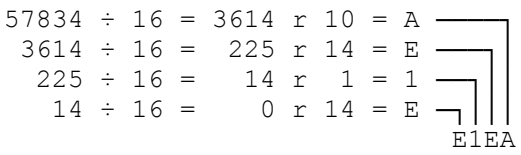
For example, to convert -213_D to $177{,}453_O$:



Unsigned Decimal to Hexadecimal Conversion

To convert an unsigned value from decimal to hexadecimal, continuously integer divide by 16. The remainders of each division are the result, least significant digit first (remember that 10 is A, 11 is B, etc.). Leading 0s may be added to achieve a 16- or 8-bit result.

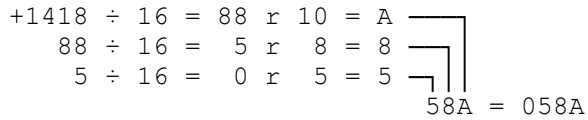
For Example, to convert 57834_D to $E1EA_H$:



Signed Decimal to Hexadecimal Conversion

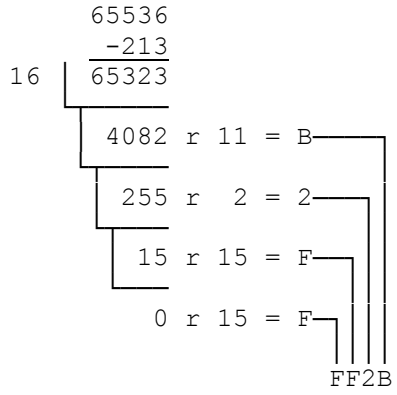
To convert a positive signed value from decimal to hexadecimal, continuously integer divide by 16. The remainders of each division are the result, least significant digit first (remember that 10 is A, 11 is B, etc.). Leading 0s may be added to produce a 16 or 8-bit value.

For example, to convert $+1418_D$ to $058A_H$:

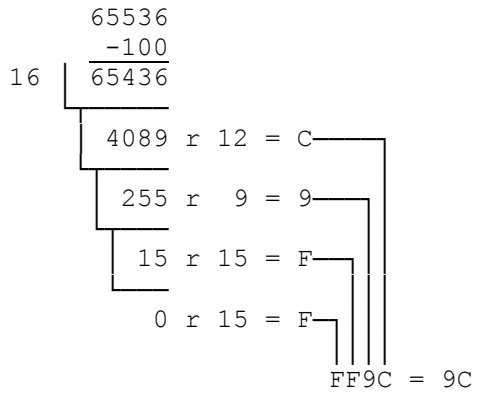


To convert a negative signed value from decimal to hexadecimal, first add the value to 65536, then proceed as though it were unsigned.

For example, to convert -213_D to $FF2B_H$:



Similarly, to convert -100_D to $9C_H$:



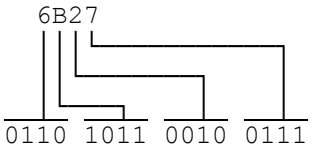
Notice that the last example produces a proper byte result: the two most significant digits are FF and the two least significant digits have a value greater than 7F (80 through FF). The FF of the most significant digits are merely carrying the sign forward: there is actually an infinite string of “F”s preceding a negative value, just as there are an infinite string of “0”s preceding a positive value.

Hexadecimal to Binary Conversion

To convert a value from hexadecimal to binary, simply convert each digit into its appropriate four bits. All digits convert identically.

0 = 0000	4 = 0100	8 = 1000	C = 1100
1 = 0001	5 = 0101	9 = 1001	D = 1101
2 = 0010	6 = 0110	A = 1010	E = 1110
3 = 0011	7 = 0111	B = 1011	F = 1111

For example, to convert $6B27_H$ to $0110,1011,0010,0111_B$:



Hexadecimal to Octal Conversion

To convert a value from hexadecimal to octal is a two step process. First convert the Hexadecimal value to binary, then convert the binary value to octal.

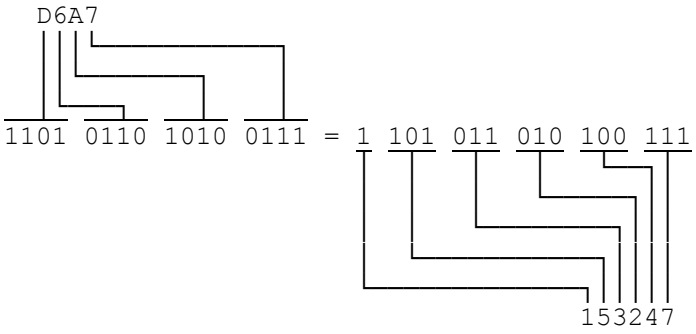
To convert a value from hexadecimal to binary, convert each hexadecimal digit into the corresponding group of four bits.

0 = 0000	4 = 0100	8 = 1000	C = 1100
1 = 0001	5 = 0101	9 = 1001	D = 1101
2 = 0010	6 = 0110	A = 1010	E = 1110
3 = 0011	7 = 0111	B = 1011	F = 1111

Then, to convert the binary value into octal, break the binary value into groups of three bits from right to left and convert each group of three bits into the corresponding octal digit.

000 = 0	010 = 2	100 = 4	110 = 6
001 = 1	011 = 3	101 = 5	111 = 7

For example, to convert $D6A7_H$ to $153,247_O$



Unsigned Hexadecimal to Decimal Conversion

To convert an unsigned value from hexadecimal to decimal, convert each digit to decimal, multiply by its appropriate power of 16, and add the results.

0 = 0	2 = 2	4 = 4	6 = 6	8 = 8	A = 10	C = 12	E = 14
1 = 1	2 = 2	5 = 5	7 = 7	9 = 9	B = 11	D = 13	F = 15

For example, to convert $E6DB_H$ to $59{,}099_D$:

```
E6DB
|||└─ B = 11 × 16^0 = 11 × 1    =     11
||└── D = 13 × 16^1 = 13 × 16   =    208
|└─── 6 =  6 × 16^2 =  6 × 256  =   1536
└──── E = 14 × 16^3 = 14 × 4096 =  57344
                                   59099
```

Signed Hexadecimal to Decimal Conversion

To convert a signed value from hexadecimal to decimal, determine first if the value is positive or negative. The most significant digit of a positive value is less than 8, while that of a negative value is 8 or more. Be certain there is an even number of digits (either four or two) present: 9CF4 is a negative value, while 9CF is not (9CF is actually 09CF).

If a positive value, convert each digit to decimal, multiply by its appropriate power of 16, and add the results.

0 = 0	2 = 2	4 = 4	6 = 6	8 = 8	A = 10	C = 12	E = 14
1 = 1	2 = 2	5 = 5	7 = 7	9 = 9	B = 11	D = 13	F = 15

For example, to convert $64B2_H$ to $+9394_D$:

```
64B2
|||└ 2 =  2 × 16^0 =  2 × 1    =     2
||└─ B = 11 × 16^1 = 11 × 16   =   176
|└── 4 =  4 × 16^2 =  4 × 256  =  1024
└─── 6 =  2 × 16^3 =  2 × 4096 =  8192
                                +9394
```


0 = 0	2 = 2	4 = 4	6 = 6	8 = 8	A = 10	C = 12	E = 14
1 = 1	2 = 2	5 = 5	7 = 7	9 = 9	B = 11	D = 13	F = 15

If a 16-bit negative value, convert each digit to decimal, multiply by its appropriate power of 16, add the results, and then subtract 65536.

For example, to convert $A3E1_H$ to $–23{,}583_D$:

```
A3E1
|||└ 1 =  1 × 16^0 =  1 × 1    =     1
||└─ E = 14 × 16^1 = 14 × 16   =   224
|└── 3 =  3 × 16^2 =  3 × 256  =   768
└─── A = 10 × 16^3 = 10 × 4096 = 40960
                               +41953
                               -65536
                               -23583
```

If an 8-bit negative value, convert each digit to decimal, multiply by its appropriate power of 16, add the results, and then subtract 256.

For example, to convert $D4_H$ to $–44_D$:

```
D4
|└──4 = 4  × 16^0 =  4 × 1  =   4
└───D = 13 × 16^1 = 13 × 16 = 208
                             +212
                             -256
                              -44
```

Source Codes for DSTAT Utility

Obtain Directory Status Utility

```
;DSTAT.MAC
;
;Utility to obtain directory status.
;
;Copyright (C) 1987, 1989, 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       3.10
;
;Edit History:
;       10/16/87  rrb  1.00  Created from scratch
;       02/17/89  rrb  2.00  Corrected allocation block bug
;       04/20/90  rrb  3.00  Rewrote from scratch to increase speed
;       05/12/90  rrb  3.10  Modularized to use generic code
;
        NAME    ('DSTAT')           ;Program ID
;
        .Z80                        ;Zilog mnemonics
;
;*****  SPECIAL NOTE  ************************************************
;
;  Those lines indicated by ";<<<" are to be used if linking is done via
;  TurboDOS's GEN linker as follows:
;
;    0A}GEN DSTAT.COM
;    * DSTAT,DSTNIT,DSTSCN,DSTOUT,DSTERR,DSTMSG,DSTCNT
;    * BSYWRK,CVTOUT,OUTDSP
;    *
;
;  Those lines indicated by ";>>>" are to be used if linking is done via
;  Microsoft's Link-80 (L80) linker as follows:
;
;    0A}L80
;    * DSTAT,DSTNIT,DSTSCN,DSTOUT,DSTERR,DSTMSG,DSTCNT
;    * BSYWRK,CVTOUT,OUTDSP
;    * DSTAT/N/E
;
;  Do not include both sets of lines.
;
;*****  Equates  *****************************************************
;
AFF     EQU     12                  ;^C -- ASCII Formfeed
;
OPSYS   EQU     0000H               ;Warm-start entry point
;
;*****  Main Routine  ************************************************
;
        CSEG                        ;Locate in code segment
;
        JP      BEGIN               ;<<<  ;Skip patch points
;
CLSCHR::DB      AFF                 ;<<<  ;Clear-screen character
;
```

```
BEGIN:  LD      SP,STACK            ;Establish local stack
        CALL    DSTNIT##            ;Initialize parameters
;
        CALL    BSYNIT##            ;Initialize busywork routine
;
        CALL    CVTNIT##            ;Initialize conversion routine
;
        CALL    OUTNIT##            ;Initialize output routine
;
        CALL    DSTSCN##            ;Scan directory and process data
;
        CALL    DSTOUT##            ;Output results
;
        JP      OPSYS               ;Exit to operating system
;
;*****  Data  ************************************************************
;
        DSEG                        ;Locate in data segment
;
;CLSCHR::DB     AFF                 ;>>>Clear screen character
;
;*****  Output user offset table
;
OUOTBL::DW      0,8,16,24,1,9,17,25
        DW      2,10,18,26,3,11,19,27
        DW      4,12,20,28,5,13,21,29
        DW      6,14,22,30,7,15,23,31
;
;*****  User extent counter table
;
EXTTBL::DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
;
;*****  User file counter table
;
FILTBL::DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DW      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
;
;*****  General data
;
FCBFIL::DB      '$       DIR'       ;Directory file spec
;
DRVLBL::DS      11                  ;Drive label
HSHFLG::DB      0                   ;Hashed flag
;
TOTDAT::DW      0,0                 ;Total data capacity
USDDAT::DW      0,0                 ;Used data capacity
REMDAT::DW      0,0                 ;Remaining data capacity
;
TOTEXT::DW      0,0                 ;Total extents
SYSEXT::DW      0,0                 ;System extents
USDEXT::DW      0,0                 ;Used extents
REMEXT::DW      0,0                 ;Remaining extents
;
TOTFIL::DW      0,0                 ;Total Files
;
SIZCOD::DB      0                   ;Allocation block size code
```

```
BLKSIZ::DB      0                   ;Block size in KB
BLKEXT::DW      0                   ;Extents per block
;
DMAADR::DW      0                   ;DMA buffer address
;
        DS      64
STACK:
;
;*****************************************************************************
;
        END                     ;<<<
;       END     BEGIN           ;>>>
```

Initialization Routine

```
;DSTNIT.MAC
;
;Initialization routine for DSTAT.MAC
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       3.10
;
        NAME    ('DSTNIT')          ;Program ID
;
        .Z80                        ;Zilog mnemonics
;
;*****  Equates  ***************************************************************
;
BDOS    EQU     0005H               ;C-function (BDOS) entry point
TDOS    EQU     0050H               ;T-function entry point
DFCB    EQU     005CH               ;Default file control block
;
RDLIV   EQU     24                  ;C24 -- Get login vector
RDCDR   EQU     25                  ;C25 -- Get current drive
WRDMA   EQU     26                  ;C26 -- Set DMA address
WRMSC   EQU     44                  ;C44 -- Set multi-sector count
;
RDTSN   EQU     12                  ;T12 -- Return TurboDOS serial number
RDDAI   EQU     19                  ;T19 -- Return drive allocation info
;
;*****  Initialization and Display Title  **********************************
;
        CSEG                        ;Locate in code segment
;
;*****  Display operating title
;
DSTNIT::CALL    OPTMS##             ;Display operating title
;
;*****  Check for privileged operator
;
        LD      C,RDTSN             ;Get operator status
        CALL    TDOS
;
        BIT     7,B                 ;Privileged?
        JP      Z,NPUERR##          ;If no, error
;
;*****  Set FCB drive
;
```

```
        LD      A,(DFCB)            ;Get selected drive
        OR      A                   ;Current?
        JR      NZ,_1               ;If yes, skip
;
        LD      C,RDCDR             ;Get current drive
        CALL    BDOS
;
        INC     A
_1:     LD      (DFCB),A            ;Set FCB for proper drive
;
;*****  Check for drive ready
;
        LD      C,RDLIV             ;Get log-in vector
        CALL    BDOS
;
        LD      A,(DFCB)            ;Get drive code
        LD      B,A
_2:     SRL     H                   ;Is drive ready?
        RR      L
        DJNZ    _2
;
        JP      NC,DNRERR##         ;If no, error
;
;*****  Initialize drive parameters
;
        DEC     A                   ;Get disk allocation information
        LD      E,A
        LD      C,RDDAI
        CALL    TDOS
;
        CP      -1                  ;Network error?
        JP      Z,NWEERR##          ;If yes, error
;
        LD      B,0                 ;Compute and save total data blocks
        OR      A
        SBC     HL,BC
        LD      (TOTDAT##),HL
        OR      A                   ;Compute and save used data blocks
        SBC     HL,DE
        LD      (USDDAT##),HL
        LD      (REMDAT##),DE       ;Save remaining data blocks
        AND     0FH                 ;Compute and save block size code
        LD      B,A
        LD      C,A
        LD      (SIZCOD##),A
        LD      A,20H               ;Compute and save block size in KB
_3:     RLCA
        DJNZ    _3
;
        LD      (BLKSIZ##),A
        LD      A,C                 ;Get size code
        SUB     3                   ;1KB blocks?
        JR      Z,_7                ;If yes, skip
;
        LD      HL,(TOTDAT##)       ;Convert total data blocks to KB
        LD      DE,0
        LD      B,A
_4:     SLA     L
        RL      H
        RL      E
        DJNZ    _4
```

```
;
        LD      (TOTDAT##),HL       ;Save result
        LD      (TOTDAT##+2),DE
        LD      HL,(USDDAT##)       ;Convert used data blocks to KB
        LD      DE,0
        LD      B,A
_5:     SLA     L
        RL      H
        RL      E
        DJNZ    _5
;
        LD      (USDDAT##),HL       ;Save result
        LD      (USDDAT##+2),DE
        LD      HL,(REMDAT##)       ;Convert remaining data blocks to KB
        LD      DE,0
        LD      B,A
_6:     SLA     L
        RL      H
        RL      E
        DJNZ    _6
;
        LD      (REMDAT##),HL       ;Save result
        LD      (REMDAT##+2),DE
;
;*****  Initialize DMA buffer
;
_7:     LD      A,(SIZCOD##)        ;Convert block size to bytes
        LD      DE,128
        LD      B,A
_8:     SLA     E
        RL      D
        DJNZ    _8
;
        LD      HL,(BDOS+1)         ;Get bottom of o/s
        DEC     HL                  ;Make it top of TPA
        LD      L,0                 ;Put it on page boundary
        OR      A                   ;Compute DMA buffer address
        SBC     HL,DE
        LD      (DMAADR##),HL       ;Save it
        LD      A,H                 ;At least 2K of TPA below DMA buffer?
        CP      9
        JP      C,OOMERR##          ;If no, error
;
        PUSH    HL                  ;Save DMA address
        LD      B,D                 ;Purge DMA buffer
        LD      C,E
        LD      D,H
        LD      E,L
        INC     DE
        DEC     BC
        LD      (HL),0
        LDIR
        POP     DE                  ;Set system DMA address
        LD      C,WRDMA
        CALL    BDOS
;
;*****  Initialize multi-sector count
;
        LD      A,(BLKSIZ##)        ;Convert block size to records
        ADD     A,A
        ADD     A,A
```

```
        ADD     A,A
        LD      E,A
        LD      D,0
        PUSH    DE                  ;Save result
        LD      C,WRMSC             ;Set multi-sector count
        CALL    BDOS
;
;*****  Initialize extents per block
;
        POP     HL                  ;Convert records to extents
        SLA     L
        RL      H
        SLA     L
        RL      H
        LD      (BLKEXT##),HL       ;Save extents per block
;
;*****  Initialize FCB
;
        LD      HL,FCBFIL##         ;Initialize filename and type
        LD      DE,DFCB+1
        LD      BC,11
        LDIR
        LD      H,D                 ;Purge rest of FCB
        LD      L,E
        INC     DE
        LD      BC,23
        LD      (HL),0
        LDIR
        RET                         ;Done
;
;*******************************************************************************
;
        END
```

Scan Directory Routine

```
;DSTSCN.MAC
;
;Directory scanning routine for DSTAT.MAC
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       3.10
;
        NAME    ('DSTSCN')          ;Program ID
;
        .Z80                        ;Use Zilog mnemonics
;
;*****  Equates  ***************************************************
;
BDOS    EQU     0005H               ;C-function (BDOS) entry point
DFCB    EQU     005CH               ;Default file control block
;
OPFIL   EQU     15                  ;C15 -- Open file
CLFIL   EQU     16                  ;C16 -- Close file
RDSEQ   EQU     20                  ;C20 -- Read sequential
;
;*****  Scan Directory and Process Extents  ************************************
;
```

```
        CSEG                    ;Locate in code segment
;
;*****  Display scanning message
;
DSTSCN::CALL    SCNMS##         ;Display scanning message
;
        LD      A,(DFCB)        ;Display drive code
        CALL    DSPALF##
;
        LD      A,':'           ;Display a colon
        CALL    DSPCHR##
;
        LD      A,' '           ;Display two spaces
        CALL    DSPCHR##
        CALL    DSPCHR##
;
;*****  Open directory as a file
;
        LD      DE,DFCB         ;Open directory
        LD      C,OPFIL
        CALL    BDOS
;
        OR      A               ;Good open?
        JP      NZ,UODERR##     ;If no, error
;
;*****  Read an allocation block
;
_1:     LD      DE,DFCB         ;Read a block
        LD      C,RDSEQ
        CALL    BDOS
;
        OR      A               ;Good read?
        JR      NZ,_8           ;If no, skip
;
        CALL    BSYWRK##        ;Do busywork
;
;*****  Check for unused extent
;
        LD      IX,(DMAADR##)   ;Point to DMA buffer
        LD      BC,(BLKEXT##)   ;Get number of extents/block
_2:     PUSH    BC              ;Save extent count
        LD      A,(IX+0)        ;Is extent used?
        CP      0E5H
        JR      NZ,_6           ;If yes, skip
;
;*****  Process label extent
;
        LD      A,(IX+15)       ;Label extent?
        CP      0FFH
        JR      NZ,_3           ;If no, skip
;
        PUSH    BC              ;Save counter
        PUSH    IX              ;Transfer label and hash flag
        POP     HL
        INC     HL
        LD      DE,DRVLBL##
        LD      BC,12
        LDIR
        POP     BC              ;Restore count
        JR      _4              ;Update system extent counter
;
```

```
;*****  Process allocation map extents
;
_3:     CP      0FEH                ;Allocation map extent?
        JR      NZ,_5               ;If no, skip
;
_4:     CALL    SYSCNT##            ;Bump system extent counter
;
        JR      _7                  ;Check for another extent
;
;*****  Process unused extents
;
_5:     CALL    UNUCNT##            ;Update unused extent counters
;
        JR      _7                  ;Check for another extent
;
;*****  Process extents in use
;
_6:     CALL    USRCNT##            ;Bump count for appropriate user
;
;*****  Check for next extent
;
_7:     LD      DE,32               ;Point to next extent
        ADD     IX,DE
        POP     BC                  ;Restore count
        DEC     BC                  ;If more to do, do next extent
        LD      A,B
        OR      C
        JR      NZ,_2
;
        JR      _1                  ;Else, go read next block
;
;*****  Close directory
;
_8:     LD      DE,DFCB             ;Close directory
        LD      C,CLFIL
        JP      BDOS                ;... and done
;
;************************************************************************
;
        END
```

Output Module

```
;DSTOUT.MAC
;
;Output module for DSTAT.MAC
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       3.10
;
        NAME    ('DSTOUT')          ;Program ID
;
        .Z80                        ;Zilog mnemonics
;
;*****  Equates  ********************************************************
;
DFCB    EQU     005CH               ;Default file control block
;
```

```
;*****  Output Results  ********************************************************
;
        CSEG                    ;Locate in code segment
;
;*****  Display list message
;
DSTOUT::LD      A,(LSTFLG##)    ;Is list flag set?
        OR      A
        JR      Z,_1            ;If no, skip
;
        CALL    LSTMS##         ;Display list message
;
;*****  Output heading
;
_1:     CALL    TTLMS##         ;Output title
;
        LD      A,(DFCB)        ;Output drive letter
        CALL    OUTALF##
;
        LD      HL,DRVLBL##     ;Transfer label name
        LD      DE,LBLMS##+4
        LD      BC,8
        LDIR
        INC     DE              ;Skip the period
        LD      BC,3            ;Transfer label type
        LDIR
        CALL    LBLMS##         ;Output drive label
;
        LD      A,(HSHFLG##)    ;Hashed?
        AND     80H
        CALL    Z,NHSMS##       ;If no, output non-hashed message
;
        CALL    NZ,HSHMS##      ;If yes, output hashed message
;
        LD      A,(BLKSIZ##)    ;Get block size
        CP      16              ;16 KB?
        JR      NZ,_2           ;If no, skip
;
        LD      A,'1'           ;Output a "1"
        CALL    OUTCHR##
;
        LD      A,6             ;Set block size to 6
        JR      _3              ;Skip
;
_2:     LD      B,A             ;Save block size
        LD      A,' '           ;Output a space
        CALL    OUTCHR##
;
        LD      A,B             ;Restore block size
_3:     CALL    OUTDGT##        ;Output block size
;
        CALL    BSZMS##         ;Output block size message
;
;*****  Output totalized data
;
        LD      IX,TOTDAT##     ;Convert/output total data capacity
        CALL    CVTOUT##
;
        CALL    NDEMS##         ;Output directory extents message
;
        LD      IX,TOTEXT##     ;Convert/output total extents
```

```
        CALL    CVTOUT##
;
        CALL    DCUMS##             ;Output data capacity in use message
;
        LD      IX,USDDAT##         ;Convert/output data capacity in use
        CALL    CVTOUT##
;
        CALL    NSEMS##             ;Output system extents message
;
        LD      IX,SYSEXT##         ;Convert/output system extents
        CALL    CVTOUT##
;
        CALL    DCRMS##             ;Output capacity remaining message
;
        LD      IX,REMDAT##         ;Convert/output capacity remaining
        CALL    CVTOUT##
;
        CALL    NEUMS##             ;Output extents in use message
;
        LD      IX,USDEXT##         ;Convert/output extents in use
        CALL    CVTOUT##
;
        CALL    NFLMS##             ;Output number of files message
;
        LD      IX,TOTFIL##         ;Convert/output number of files
        CALL    CVTOUT##
;
        CALL    NERMS##             ;Output extents remaining message
;
        LD      IX,REMEXT##         ;Convert/output extents remaining
        CALL    CVTOUT##
;
;*****  Output user data
;
        CALL    HDRMS##             ;Output header message
;
        LD      A,'-'               ;Set substitute character to "-"
        CALL    CVTSUB##
;
        LD      HL,OUOTBL##         ;Point to output user offset table
        LD      B,32                ;Set count to 32 users
_4:     LD      E,(HL)              ;Get current output user offset
        INC     HL
        LD      D,(HL)
        INC     HL
        PUSH    DE                  ;Save user number
        LD      D,' '               ;Preset for space
        LD      A,E                 ;User 0-9?
        CP      10
        JR      C,_5                ;If yes, skip
;
        LD      D,'1'               ;Preset for msd=1
        SUB     10                  ;User 10-19?
        CP      10
        JR      C,_5                ;If yes, skip
;
        INC     D                   ;Preset for msd=2
        SUB     10                  ;User 20-29?
        CP      10
        JR      C,_5                ;If yes, skip
;
```

```
        INC     D                   ;Preset for msd=3
        SUB     10                  ;Set for user 30-31
_5:     LD      E,A                 ;Save lsd
        LD      A,D                 ;Output msd
        CALL    OUTCHR##
;
        LD      A,E                 ;Output lsd
        CALL    OUTDGT##
;
        POP     DE                  ;Restore user number
        LD      A,E                 ;Get it
        PUSH    AF                  ;Save it again
        SLA     E                   ;Convert it to offset
        RL      D
        SLA     E
        RL      D
        LD      IX,EXTTBL##         ;Index to extent count
        ADD     IX,DE
        CALL    CVTOUT##            ;Convert/output it
;
        LD      IX,FILTBL##         ;Index to file count
        ADD     IX,DE
        CALL    CVTOUT##            ;Convert/output it
;
        POP     AF                  ;Restore user number
        CP      24                  ;User 24 or above?
        CALL    C,SEPMS##           ;If no, output separator
;
        CALL    NC,NLNMS##          ;If yes, output new line
;
        DJNZ    _4                  ;Repeat for all 32 users
;
        RET                         ;Done
;
;*****************************************************************************
;
        END
```

Error Routines

```
;DSTERR.MAC
;
;Error routines for DSTAT.MAC
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       3.10
;
        NAME    ('DSTERR')          ;Program ID
;
        .Z80                        ;Zilog mnemonics
;
;*****  Equates  ***********************************************************
;
OPSYS   EQU     0000H               ;Warm-start entry point
;
;*****  Error Routines  ****************************************************
;
        CSEG                        ;Locate in code segment
```

```
;
;*****  Non-privileged user
;
NPUERR::CALL    NPUMS##             ;Display primary message
;
        JR      _1
;
;*****  Drive not ready
;
DNRERR::CALL    DNRMS##             ;Display primary message
;
        JR      _1
;
;*****  Network error
;
NWEERR::CALL    NWEMS##             ;Display primary message
;
        JR      _1
;
;*****  Out of memory
;
OOMERR::CALL    OOMMS##             ;Display primary message
;
        JR      _1
;
;*****  Unable to open directory
;
UODERR::CALL    UODMS##             ;Display primary message
;
_1:     CALL    ERRMS##             ;Display secondary message
;
        POP     HL                  ;Balance the stack
        JP      OPSYS               ;Exit to operating system
;
;*************************************************************************
;
        END
```

Message Routines

```
;DSTMSG.MAC
;
;Messages for DSTAT.MAC
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       3.10
;
        NAME    ('DSTMSG')          ;Program ID
;
        .Z80                        ;Zilog mnemonics
;
;*****  Equates  ***************************************************
;
ABEL    EQU     7                   ;^G  -- ASCII Bell
ALF     EQU     10                  ;^J  -- ASCII Linefeed
ACR     EQU     13                  ;^M  -- ASCII Carriage return
;
PLF     EQU     ALF+80H             ;^J+ -- ASCII Linefeed + parity
```

```
;
;*****  Operational Message Routines  ************************************
;
        CSEG                        ;Locate in code segment
;
;*****  Operational title
;
OPTMS:: PUSH    AF
        LD      A,(CLSCHR##)        ;Clear the screen
        CALL    DSPBYT##
;
        POP     AF
        CALL    DSPMSG##
;
        DB      'Directory Scan Utility'
        DB      ACR,ALF
        DB      'Version 3.10'
        DB      ACR,ALF
        DB      'Copyright (C) 1987,1989,1990, R. Roger Breton'
        DB      ACR,ALF,PLF
;
        RET
;
;*****  Scanning directory message
;
SCNMS:: CALL    DSPMSG##
;
        DC      'Scanning directory of drive '
;
        RET
;
;*****  List output message
;
LSTMS:: CALL    DSPMSG##
;
        DB      ACR
        DC      'Outputting to list device.                                  '
;
        RET
;
;*****  Error Message Routines  ******************************************
;
;*****  Non-privileged message
;
NPUMS:: CALL    DSPMSG##
;
        DC      'Non-privileged user'
;
        RET
;
;*****  Drive not ready message
;
DNRMS:: CALL    DSPMSG##
;
        DC      'Drive not ready'
;
        RET
;
;*****  Out of memory message
;
OOMMS:: CALL    DSPMSG##
```

```
;
        DC      'Out of memory'
;
        RET
;
;*****  Network error message
;
NWEMS:: CALL    DSPMSG##
;
        DC      'Network error'
;
        RET
;
;*****  Unable to open directory
;
UODMS:: CALL    DSPMSG##
;
        DC      'Unable to open directory'
;
        RET
;
;*****  Error message
;
ERRMS:: CALL    DSPMSG##
;
        DB      ', program aborted.'
        DB      ABEL,ACR,PLF
;
        RET
;
;*****  Output Message Routines  ****************************************
;
;*****  Title
;
TTLMS:: PUSH    AF                  ;Save accumulator
        LD      A,(LSTFLG##)        ;Output to list device?
        OR      A
        JR      NZ,_S1              ;If yes, skip
;
        LD      A,(CLSCHR##)        ;Clear console screen
        CALL    OUTBYT##
;
_S1:    POP     AF                  ;Restore accumulator
        CALL    OUTMSG##
;
        DB      '                         Breton Directory Status Utility'
        DB      ACR,ALF
        DB      '                                 Version 3.10'
        DB      ACR,ALF,PLF
;
        RET
;
;*****  Label
;
LBLMS:: CALL    OUTMSG##
;
        DB      ':        .                    '
;
        RET
;
;*****  Non-hashed message
```

```
;
NHSMS:: CALL    OUTMSG##
;
        DC      'Non-Hashed                     '
;
        RET
;
;*****  Hashed message
;
HSHMS:: CALL    OUTMSG##
;
        DC      '  Hashed                       '
;
        RET
;
;*****  Block-size message
;
BSZMS:: CALL    OUTMSG##
;
        DB      ' KB Block Size'
        DB      ACR,ALF,ALF
        DC      ' Data Capacity of Drive:   '
;
        RET
;
;*****  Directory extents message
;
NDEMS:: CALL    OUTMSG##
;
        DC      ' KB  |  Number of Directory Extents: '
;
        RET
;
;*****  Data in-use message
;
DCUMS:: CALL    OUTMSG##
;
        DB      ACR,ALF
        DC      ' Data Capacity in Use:     '
;
        RET
;
;*****  System extents message
;
NSEMS:: CALL    OUTMSG##
;
        DC      ' KB  |  Number of System Extents:    '
;
        RET
;
;*****  Data remaining message
;
DCRMS:: CALL    OUTMSG##
;
        DB      ACR,ALF
        DC      ' Data Capacity Remaining:  '
;
        RET
;
;*****  In-Use extents message
;
```

```
NEUMS:: CALL    OUTMSG##
;
        DC      ' KB  |  Number of Extents in Use:    '
;
        RET
;
;*****  Number of files message
;
NFLMS:: CALL    OUTMSG##
;
        DB      ACR,ALF
        DC      ' Number of Files:          '
;
        RET
;
;*****  Remaining extents message
;
NERMS:: CALL    OUTMSG##
;
        DC      '     |  Number of Extents Remaining: '
;
        RET
;
;*****  Header message
;
HDRMS:: CALL    OUTMSG##
;
        DB      ACR,ALF,ALF
        DB      'Usr   Ext    Fil  |  Usr   Ext    Fil  |  '
        DB      'Usr   Ext    Fil  |  Usr   Ext    Fil'
        DB      ACR,ALF
        DB      '——————————————-  |  ——————————————-  |  '
        DB      '——————————————-  |  ——————————————-'
        DB      ACR,PLF
;
        RET
;
;*****  Field separator
;
SEPMS:: CALL    OUTMSG##
;
        DC      '  |  '
;
        RET
;
;*****  New line
;
NLNMS:: CALL    OUTMSG##
;
        DB      ACR,PLF
;
        RET
;
;*************************************************************************
;
        END
```

Extent Counting Routines

```
;DSTCNT.MAC
;
```

```
;Extent counting subroutines for DSTAT.MAC
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       3.10
;
        NAME    ('DSTCNT')          ;Program ID
;
        .Z80                        ;Zilog mnemonics
;
;*****  Extent Counter Incrementing Subroutines  ****************************
;
        CSEG                        ;Locate in code segment
;
;*****  Increment system extent counter
;
SYSCNT::LD      HL,SYSEXT##         ;Bump system extents
        CALL    _4
;
        JR      _3                  ;Bump total extents
;
;*****  Increment unused extent counter
;
UNUCNT::LD      HL,REMEXT##         ;Bump remaining extents
        CALL    _4
;
        JR      _3                  ;Bump total extents and done
;
;*****  Increment user-area extent counter
;
USRCNT::LD      HL,EXTTBL##         ;Point to extent counter table
        ADD     A,A                 ;Index to proper counter
        ADD     A,A
        LD      E,A
        LD      D,0
        ADD     HL,DE
        PUSH    DE                  ;Save offset
        CALL    _4                  ;Bump user extents
;
        POP     DE                  ;Restore offset
;
;*****  Increment user-area file counter
;
        LD      A,(IX+14)           ;Get 2nd extent counter
        OR      A                   ;Extent 0-31?
        JR      NZ,_2               ;If no, skip
;
        LD      A,(BLKSIZ##)        ;Get block size IN KB
        SRL     A                   ;1 KB blocks?
        OR      A
        JR      NZ,_1               ;If no, skip
;
        INC     A                   ;Set for 1 KB blocks
_1:     LD      L,A                 ;Save block size code
        LD      A,(IX+12)           ;Get 1st extent counter
        CP      L                   ;"1st extent?"
        JR      NC,_2               ;If no, skip
;
        LD      HL,FILTBL##         ;Point to file counter table
```

```
        ADD     HL,DE               ;Index to proper counter
        CALL    _4                  ;Bump user files
;
;*****  Increment total file counter
;
        LD      HL,TOTFIL##         ;Bump total files
        CALL    _4
;
;*****  Increment used extent counter
;
_2:     LD      HL,USDEXT##         ;Go bump used extents
        CALL    _4
;
;*****  Increment total extent counter
;
_3:     LD      HL,TOTEXT##         ;Point to total extent counter
_4:     LD      A,(HL)              ;Bump the counter
        ADD     A,1
        LD      (HL),A
        INC     HL
        LD      A,(HL)
        ADC     A,0
        LD      (HL),A
        INC     HL
        LD      A,(HL)
        ADC     A,0
        LD      (HL),A
        RET                         ;Done
;
;*************************************************************************
;
        END
```

Display Busywork Routines

```
;BSYWRK.MAC
;
;Subroutine to display 32-dot busywork
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       1.00
;
;Edit History:
;       05/12/90  rrb  1.00  Created
;
        NAME    ('BSYWRK')          ;Program ID
;
        .Z80                        ;Zilog mnemonics
;
;*****  Equates  ********************************************************
;
ABS     EQU     8                   ;^H -- ASCII backspace
;
;*****  Busywork Initialization Routine  ********************************
;
        CSEG                        ;Locate in code segment
;
_D1:    DB      0                   ;Busywork counter
```

```
;
BSYNIT::PUSH    AF                  ;Save registers
        XOR     A                   ;Clear busywork counter
        LD      (_D1),A
        POP     AF                  ;Restore registers
        RET                         ;Done
;
;*****  Display Busywork Routine  ****************************************
;
BSYWRK::PUSH    BC                  ;Save registers
        PUSH    AF
        LD      A,'.'               ;Display a period
        CALL    DSPBYT##
;
        LD      A,(_D1)             ;Last one?
        INC     A
        AND     1FH
        LD      (_D1),A
        JR      NZ,_3               ;If no, skip
;
        CALL    _1                  ;Back up to busywork beginning
;
        LD      A,' '               ;Clear busywork
        CALL    _2
;
        LD      BC,_3               ;Set return address
        PUSH    BC
_1:     LD      A,ABS               ;Get backspace
_2:     LD      B,32                ;Display byte 32 times
        JP      DSPSER##
;
_3:     POP     AF                  ;Restore registers
        POP     BC
        RET                         ;Done
;
;*******************************************************************************
;
        END
```

20-Bit Conversion and Output Routines

```
;CVTOUT.MAC
;
;Subroutine to convert and output a 20-bit hex number
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       1.00
;
;Edit History:
;       05/12/90  rrb  1.00  Created
;
        NAME    ('CVTOUT')          ;Program ID
;
        .Z80                        ;Zilog mnemonics
;
;*****  Initialize Conversion Routine  ***************************************
;
        CSEG                        ;Locate in code segment
```

```
;
_D1:    DB      0                   ;First character flag
_D2:    DB      0                   ;Leading zeros flag
_D3:    DB      '0'                 ;Substitute character
;
CVTNIT::PUSH    AF                  ;Save registers
        XOR     A                   ;Clear leading zeros flag
        LD      (_D2),A
        OR      '0'                 ;Initialize substitute character
        LD      (_D3),A
        POP     AF                  ;Restore registers
        RET                         ;Done
;
;*****  Set Substitute Character Routine  **********************************
;
CVTSUB::LD      (_D3),A             ;Set substitute character
        RET                         ;Done
;
;*****  Set/Clear Leading Zeros Routine  ***********************************
;
CVTLZR::LD      (_D2),A             ;Set/clear leading zeros flag
        RET
;
;*****  Data Conversion and Output Routine  ********************************
;
;*****  Initialize
;
CVTOUT::PUSH    HL                  ;Save registers
        PUSH    DE
        PUSH    BC
        PUSH    AF
        LD      A,-1                ;Set first-character flag
        LD      (_D1),A
        LD      L,(IX+0)            ;Get value to be converted
        LD      H,(IX+1)
        LD      A,(IX+2)
;
;*****  Process 1,000,000's
;
        LD      DE,16960            ;15 * 65536 + 16960 = 1000000
        LD      C,15
        CALL    _3
;
;*****  Process 100,000's
;
        LD      DE,34464            ;65536 + 34464 = 100000
        LD      C,1
        CALL    _3
;
;*****  Process 10,000's
;
        LD      DE,10000
        CALL    _3
;
;*****  Process 1000's
;
        LD      DE,1000             ;Do 1000's
        CALL    _3
;
;*****  Process 100's
;
```

```
        LD      DE,100              ;Do 100's
        CALL    _3
;
;*****  Process 10's
;
        LD      DE,10               ;Do 10's
        CALL    _3
;
;*****  Process 1's
;
        LD      A,L                 ;Get 1's
        OR      A                   ;0?
        JR      NZ,_1               ;If no, output it
;
        LD      A,(_D1)             ;First character?
        OR      A
        JR      Z,_1                ;If no, output 0
;
        LD      A,(_D3)             ;Output substitute character
        CALL    OUTCHR##
;
        JR      _2                  ;Skip
;
_1:     CALL    OUTDGT##            ;Output digit
;
;*****  Exit routine
;
_2:     POP     AF                  ;Restore registers
        POP     BC
        POP     DE
        POP     HL
        RET                         ;Done
;
;*****************************************************************************
;
        END
```

Console/List Output Routines

```
;OUTDSP.MAC
;
;Subroutine to support console or list output
;
;Copyright (C) 1990, R. Roger Breton
;
;Author:R. Roger Breton
;
;Version:       1.00
;
;Edit History:
;       05/12/90  rrb  1.00  Created
;
        NAME    ('OUTDSP')          ;Program ID
;
        .Z80                        ;Zilog mnemonics
;
BDOS    EQU     0005H               ;C-function (BDOS) entry point
CTAIL   EQU     0080H               ;Command tail
;
WRRCN   EQU     4                   ;C4 -- Raw console output
WRLST   EQU     5                   ;C5 -- List output
```

```
;
;*****  List Device Flag Initialization Routine  ****************************
;
        CSEG                    ;Locate in code segment
;
;*****  Initialize list device flag
;
LSTFLG::DB      0               ;List device flag
;
OUTNIT::EXX                     ;Save registers
        EX      AF,AF'
        LD      HL,CTAIL        ;Point to command tail
        LD      A,(HL)          ;Is there one?
        INC     HL
        OR      A
        JR      Z,_1            ;If no, skip
;
        LD      C,A             ;Set tail counter
        LD      B,0
        LD      A,';'           ;Look for a semicolon
        CPIR
        JR      NZ,_1           ;If not found, skip
;
        LD      A,'L'           ;Look for list option
        CPIR
        JR      NZ,_1           ;If not found, skip
;
        LD      A,-1            ;Set for output to list device
        JR      _2              ;Skip
;
_1:     XOR     A               ;Set for output to console
_2:     LD      (LSTFLG),A      ;Set list device flag accordingly
        EX      AF,AF'          ;Restore registers
        EXX
        RET                     ;Done
;
;*****  Output to Console or List Device Routines  **************************
;
;*****  Output a message (parity-terminated character string)
;
OUTMSG::EX      (SP),HL         ;Get message address
        PUSH    AF              ;Save registers
_3:     LD      A,(HL)          ;Get a character
        INC     HL
        CALL    OUTCHR          ;Output it
;
        AND     80H             ;Last character?
        JR      Z,_3            ;If no, do next character
;
        POP     AF              ;Restore registers
        EX      (SP),HL         ;Put return address
        RET                     ;Done
;
;*****  Output a series of identical bytes
;
OUTSER::PUSH    BC              ;Save count
_4:     CALL    OUTBYT          ;Output byte
;
        DJNZ    _4              ;Do it required number of times
;
        POP     BC              ;Restore count
```

```
        RET                     ;Done
;
;*****  Output a de-ASCIIed digit
;
OUTDGT::EXX                     ;Save registers
        LD      D,A             ;Save original byte
        AND     0FH             ;ASCII it
        OR      '0'
        LD      E,A             ;Save output digit
        LD      A,D             ;Restore original byte
        JR      _5              ;Output digit
;
;*****  Output a de-ASCIIed alpha character from A to P
;
OUTALF::EXX                     ;Save registers
        LD      D,A             ;Save original byte
        DEC     A               ;ASCII it
        AND     0FH
        INC     A
        OR      '@'
        LD      E,A             ;Save output alpha character
        LD      A,D             ;Restore original byte
        JR      _5              ;Output alpha character
;
;*****  Output an ASCII character (strip parity bit)
;
OUTCHR::EXX                     ;Save registers
        LD      E,A             ;Get output character
        RES     7,E             ;Strip parity bit
        JR      _5              ;Output character
;
;*****  Output a byte (preserve parity bit)
;
OUTBYT::EXX                     ;Save registers
        LD      E,A             ;Get output byte
_5:     EX      AF,AF'          ;Save accumulator
        LD      A,(LSTFLG)      ;Is list flag set
        OR      A
        JR      Z,_9            ;If no, output to console
;
        LD      C,WRLST         ;Else, output byte to list device
        JR      _10
;
;*****  Display (Output to Console Only) Routines  **************************
;
;*****  Display a message (parity-terminated character string)
;
DSPMSG::EX      (SP),HL         ;Get message address
        PUSH    AF              ;Save registers
_6:     LD      A,(HL)          ;Get a character
        INC     HL
        CALL    DSPCHR          ;Display it
;
        AND     80H             ;Last character?
        JR      Z,_6            ;If no, do next character
;
        POP     AF              ;Restore registers
        EX      (SP),HL         ;Put return address
        RET                     ;Done
;
;*****  Display a series of identical bytes
```

```
;
DSPSER::PUSH    BC                  ;Save count
_7:     CALL    DSPBYT              ;Display byte
;
        DJNZ    _7                  ;Do it required number of times
;
        POP     BC                  ;Restore count
        RET                         ;Done
;
;*****  Display a de-ASCIIed digit
;
DSPDGT::EXX                         ;Save registers
        LD      D,A                 ;Save original byte
        AND     0FH                 ;ASCII it
        OR      '0'
        LD      E,A                 ;Save output digit
        LD      A,D                 ;Restore original byte
        JR      _8                  ;Display digit
;
;*****  Display a de-ASCIIed alpha character from A to P
;
DSPALF::EXX                         ;Save registers
        LD      D,A                 ;Save original byte
        DEC     A                   ;ASCII it
        AND     0FH
        INC     A
        OR      '@'
        LD      E,A                 ;Save output alpha character
        LD      A,D                 ;Restore original byte
        JR      _8                  ;Display alpha character
;
;*****  Display an ASCII character (strip parity bit)
;
DSPCHR::EXX                         ;Save registers
        LD      E,A                 ;Get output character
        RES     7,E                 ;Strip parity bit
        JR      _8                  ;Display character
;
;*****  Display a byte (preserve parity bit)
;
DSPBYT::EXX                         ;Save registers
        LD      E,A                 ;Get output character
_8:     EX      AF,AF'              ;Save accumulator
_9:     LD      C,WRRCN             ;Set for console
_10:    CALL    BDOS                ;Display byte
;
        EX      AF,AF'              ;Restore accumulator
        EXX                         ;Restore registers
        RET                         ;Done
;
;*******************************************************************************
;
        END
```

Character Tables

Definitions

Latin character tables are provided. Each character is made up of one byte (8 bits), giving a total of 256 characters. The numerical value of each character is given in binary, octal, decimal, and hexadecimal values, followed by the control-code (for ASCII control characters), the symbol, and a description of the character.

US-ASCII Character Set

The basic Latin character set is the 7-bit ASCII (American Standard Code for Information Interchange) character set. This is the de-facto standard for all characters that may be directly typed upon a standard U.S. keyboard.

Please note that the ASCII character set is "standard" only for the U.S., and is often referred to as US-ASCII to differentiate it from other ASCII-like character sets used in other countries

In the US-ASCII character set, characters 00–1F and 7F (hex) are control characters and characters 20–7E (hex) are printable characters. Control characters represent instructions used to control the flow of data, rather than printable characters.

Please note that the term "ASCII" has become a de-facto word, pronounced "ass´key," meaning "pure text," without regard as to whether the text involved uses the ASCII character set or not. This practice has led to confusion, as an "ASCII file" denotes a "pure text file" that may or may not be an ASCII file.

ISO 646 Character Set

The ISO 646 standard defines an international character set similar to the US-ASCII character set, except that the characters 40, 5B, 5C, 5D, 7B, 7C, and 7D (hex) are specified as "national use" characters, and characters 23, 24, 5E, 5F, 60, and 7E (hex) have some latitude in their use. These "variable" characters allow individual nations to redefine the characters to suit their individual requirements. The most common (but not all) of these redefined characters are shown in an eighth "National Variants" column.

The widespread use of the ISO 646 character set, while of great benefit within national borders, has led to considerable confusion in the creation of "generic" text files. The redefinition of characters results in text that fails to read properly when using a national character set other than the one in which it was created. Of the printable characters, only 20–22, 25–3F, 41–5A, and 61–7A (hex) are safe to use (the space, the punctuation characters ! " % & ' ( ) * + , – . / : ; < = > ?, the digits 0–9, the uppercase letters A–Z, and the lowercase letters a–z).

US-ASCII and ISO 646 character sets are 7-bit character sets, with the most significant bit of each character byte being ignored. The rationale behind this was to allow the most significant bit to be used as a parity bit during data transfer. This practice has long been abandoned, so for all practical purposes (such as these tables) the most significant bit of a US-ASCII or ISO 646 character is considered to be "0".

The 7-bit nature of the US-ASCII and ISO 646 character sets has led to some problems, especially when text is transferred between programs.

Some ASCII-only (7-bit text) programs use the most significant character bit for their own purposes, thereby creating false 8-bit characters where none existed. Wordstar is famous for this. Open any Wordstar file with a plain text editor to see the resulting chaos.

Some 7-bit text programs arbitrarily clear the eighth bit. This completely destroys the integrity of any file with 8-bit characters.

Worst of all, some 7-bit programs arbitrarily set the eighth bit. This not only destroys the integrity of any file with either 7-bit or 8-bit characters, it renders that file usable only by the creating program or a like bit setting program.

It can be see from these problems that, ideally, text files should only be used with the programs that created them. When files must be transferred between programs whose actions are unknown, only a copy of a text file should be opened in case the opening program corrupts the file. The original text file should be preserved unopened.

The first table given here depicts characters 00–7F (hex), the US-ASCII and ISO 646 character set, a subset of the Latin 1 character set.

Latin 1 (ISO 8859-1) Character Set

ISO 8859-1 defines an 8-bit 256 character set, known as the Latin 1 character set. In this definition, characters 00–7F (hex) are identical to those of the ISO 646 7-bit character set, characters 80–9F (hex) are reserved for use as control characters, and characters A0–FF (hex) are defined as various currency signs, punctuation marks, symbols, and Latin letters with accents and other pronunciation marks used internationally.

Even though the Latin 1 character set includes virtually all the “national use” characters substituted for US-ASCII characters under the ISO 646 standard, many nations continue to use these substitutions.

One important feature of the Latin 1 character set is the character A0 (hex), which is a nonbreaking space. This character has all the characteristics of the space except that it does not automatically break over lines. This allows special designations, such as model or part numbers, to be treated as a single word, even though they may contain spaces.

The Latin 1 character set is often referred to as the “extended ASCII” or “8-bit ASCII” character set. This is a complete misnomer. The US-ASCII and ISO 646 characters sets are, by definition, 7-bit character sets. There is no such thing as an “extended ASCII” or “8-bit ASCII” character set.

The third table given here depicts the characters A0–FF (hex) of the Latin 1 (ISO 8859-1) character set, considered as extensions on the ISO 646 character set.

Microsoft WinLatin1 Character Set

Microsoft developed the WinLatin1 character set (Windows code page 1252) for use with Windows as a replacement of the Latin 1 character set. Microsoft later back fitted the WinLatin1 character set into some versions of DOS.

The WinLatin1 character set is essentially identical to the Latin 1 character set except for characters 80–9F (hex). The Latin 1 character set specifically designates characters 80–9F (hex) as control characters, rather than as printable characters. The WinLatin1 characters set defines most (but not all) of characters 80–9F (hex) as printable characters. Characters 81, 8D, 8F, 90, and 9D remain undefined. Microsoft has handily filled this space with various punctuation marks, symbols, and accented letters missing from the Latin 1 character set

The second table given here depicts characters 80–9F of the WinLatin1 character set as variations of the Latin 1 character set.

US-ASCII (ISO 646) Character Set

Binary	Oct	Dec	Hex	Ctrl	Chr/Code	Description	National Variants
00000000	000	0	00	^@	NUL	Null	
00000001	001	1	01	^A	SOH	Start of Heading	
00000010	002	2	02	^B	STX	Start of Text	
00000011	003	3	03	^C	ETX	End of Text	
00000100	004	4	04	^D	EOT	End of Transmission	
00000101	005	5	05	^E	ENQ	Enquiry	
00000110	006	6	06	^F	ACK	Acknowledgement	
00000111	007	7	07	^G	BEL	Bell	
00001000	010	8	08	^H	BS	Backspace	
00001001	011	9	09	^I	HT	Horizontal Tab	
00001010	012	10	0A	^J	LF	Linefeed	
00001011	013	11	0B	^K	VT	Vertical Tab	
00001100	014	12	0C	^L	FF	Formfeed	
00001101	015	13	0D	^M	CR	Carriage Return	
00001110	016	14	0E	^N	SO	Shift Out	
00001111	017	15	0F	^O	SI	Shift In	
00010000	020	16	10	^P	DLE	Data Link Escape	
00010001	021	17	11	^Q	DC1	Device Control 1 (XON)	
00010010	022	18	12	^R	DC2	Device Control 2	
00010011	023	19	13	^S	DC3	Device Control 3 (XOFF)	
00010100	024	20	14	^T	DC4	Device Control 4	
00010101	025	21	15	^U	NAK	Negative Acknowledgement	
00010110	026	22	16	^V	SYN	Synchronous Idle	
00010111	027	23	17	^W	ETB	End of Transmission Block	
00011000	030	24	18	^X	CAN	Cancel Line	
00011001	031	25	19	^Y	EM	End of Medium	
00011010	032	26	1A	^Z	SUB	Substitute	
00011011	033	27	1B	^[	ESC	Escape	
00011100	034	28	1C	^\	FS	File Separator	
00011101	035	29	1D	^]	GS	Group Separator	
00011110	036	30	1E	^^	RS	Record Separator	
00011111	037	31	1F	^_	US	Unit Separator	
00100000	040	32	20			Space	
00100001	041	33	21		!	Exclamation Point	
00100010	042	34	22		"	Double Quotation Mark	
00100011	043	35	23		#	Number Sign	£ Ù
00100100	044	36	24		$	Dollar Sign	¤
00100101	045	37	25		%	Percent Sign	
00100110	046	38	26		&	Ampersand	
00100111	047	39	27		'	Apostrophe or Single Quotation mark	
00101000	050	40	28		(	Left Parenthesis or Left Curved Bracket	
00101001	051	41	29		)	Right Parenthesis or Right Curved Bracket	
00101010	052	42	2A		*	Asterisk	
00101011	053	43	2B		+	Plus Sign	
00101100	054	44	2C		,	Comma	
00101101	055	45	2D		-	Hyphen or Minus Sign	
00101110	056	46	2E		.	Period or Full Stop or Decimal Point	
00101111	057	47	2F		/	Virgule or Slash or Solidus	

Binary	Oct	Dec	Hex	Chr	Description	National Variants
00110000	060	48	30	0	Digit “0”	
00100001	061	49	31	1	Digit “1”	
00110010	062	50	32	2	Digit “2”	
00110011	063	51	33	3	Digit “3”	
00110100	064	52	34	4	Digit “4”	
00110101	065	53	35	5	Digit “5”	
00110110	066	54	36	6	Digit “6”	
00110111	067	55	37	7	Digit “7”	
00111000	070	56	38	8	Digit “8”	
00111001	071	57	39	9	Digit “9”	
00111010	072	58	3A	:	Colon	
00111011	073	59	3B	;	Semicolon	
00111100	074	60	3C	<	Left Angle or Left Angle Bracket or Less Than Sign	
00111101	075	61	3D	=	Equals Sign	
00111110	076	62	3E	>	Right Angle or Right Angle Bracket or Greater Than Sign	
00111111	077	63	3F	?	Question Mark	
01000000	100	64	40	@	Commercial At Sign	É § Ä à³
01000001	101	65	41	A	Uppercase Latin “A”	
01000010	102	66	42	B	Uppercase Latin “B”	
01000011	103	67	43	C	Uppercase Latin “C”	
01000100	104	68	44	D	Uppercase Latin “D”	
01000101	105	69	45	E	Uppercase Latin “E”	
01000110	106	70	46	F	Uppercase Latin “F”	
01000111	107	71	47	G	Uppercase Latin “G”	
01001000	110	72	48	H	Uppercase Latin “H”	
01001001	111	73	49	I	Uppercase Latin “I”	
01001010	112	74	4A	J	Uppercase Latin “J”	
01001011	113	75	4B	K	Uppercase Latin “K”	
01001100	114	76	4C	L	Uppercase Latin “L”	
01001101	115	77	4D	M	Uppercase Latin “M”	
01001110	116	78	4E	N	Uppercase Latin “N”	
01001111	117	79	4F	O	Uppercase Latin “O”	
01010000	120	80	50	P	Uppercase Latin “P”	
01010001	121	81	51	Q	Uppercase Latin “Q”	
01010010	122	82	52	R	Uppercase Latin “R”	
01010011	123	83	53	S	Uppercase Latin “S”	
01010100	124	84	54	T	Uppercase Latin “T”	
01010101	125	85	55	U	Uppercase Latin “U”	
01010110	126	86	56	V	Uppercase Latin “V”	
01010111	127	87	57	W	Uppercase Latin “W”	
01011000	130	88	58	X	Uppercase Latin “X”	
01011001	131	89	59	Y	Uppercase Latin “Y”	
01011010	132	90	5A	Z	Uppercase Latin “Z”	
01011011	133	91	5B	[	Left Bracket or Left Square Bracket	Ä Æ ° â ¡ ÿ é
01011100	134	92	5C	\	Backslash or Reverse Solidus	Ö Ø ç Ñ ½ ¥
01011101	135	93	5D	]	Right Bracket or Right Square Bracket	Å Ü § ê é ¿ |
01011110	136	94	5E	^	Circumflex or Carat	Ü î
01011111	137	95	5F	_	Underscore	è

Binary	Oct	Dec	Hex	Chr	Description	National Variants
01100000	140	96	60	`	Grave Accent	é ä µ ô ù
01100001	141	97	61	a	Lowercase Latin “a”	
01100010	142	98	62	b	Lowercase Latin “b”	
01100011	143	99	63	c	Lowercase Latin “c”	
01100100	144	100	64	d	Lowercase Latin “d”	
01100101	145	101	65	e	Lowercase Latin “e”	
01100110	146	102	66	f	Lowercase Latin “f”	
01100111	147	103	67	g	Lowercase Latin “g”	
01101000	150	104	68	h	Lowercase Latin “h”	
01101001	151	105	69	i	Lowercase Latin “i”	
01101010	152	106	6A	j	Lowercase Latin “j”	
01101011	153	107	6B	k	Lowercase Latin “k”	
01101100	154	108	6C	l	Lowercase Latin “l”	
01101101	155	109	6D	m	Lowercase Latin “m”	
01101110	156	110	6E	n	Lowercase Latin “n”	
01101111	157	111	6F	o	Lowercase Latin “o”	
01110000	160	112	70	p	Lowercase Latin “p”	
01110001	161	113	71	q	Lowercase Latin “q”	
01110010	162	114	72	r	Lowercase Latin “r”	
01110011	163	115	73	s	Lowercase Latin “s”	
01110100	164	116	74	t	Lowercase Latin “t”	
01110101	165	117	75	u	Lowercase Latin “u”	
01110110	166	118	76	v	Lowercase Latin “v”	
01110111	167	119	77	w	Lowercase Latin “w”	
01111000	170	120	78	x	Lowercase Latin “x”	
01111001	171	121	79	y	Lowercase Latin “y”	
01111010	172	122	7A	z	Lowercase Latin “z”	
01111011	173	123	7B	{	Left Brace or Left Curly Bracket	ä æ é à ° ¨
01111100	174	124	7C	|	Vertical Line	ö ø ù ò ñ *f*
01111101	175	125	7D	}	Right Brace or Right Curly Bracket	å ü è ç ¼
01111110	176	126	7E	~	Tilde	ü ¯ ß ¨ û ì ´ _
01111111	177	127	7F	DEL	Delete	

WinLatin1 Character Set Variations

Binary	Oct	Dec	Hex	Chr	Description
10000000	200	128	80	€	Euro Sign
10000001	201	129	81		Undefined
10000010	202	130	82	‘	Left Single Low Quotation Mark
10000011	203	131	83	*f*	Lowercase Latin “f” with hook
10000100	204	132	84	„	Right Double Low Quotation Mark
10000101	205	133	85	…	Ellipsis
10000110	206	134	86	†	Obelisk or Dagger
10000111	207	135	87	‡	Diesis or Double Dagger
10001000	210	136	88	ˆ	Circumflex
10001001	211	137	89	‰	Per Mille Sign
10001010	212	138	8A	Š	Uppercase Latin “S” with Caron
10001011	213	139	8B	‹	Left Single Angle Quotation Mark
10001100	214	140	8C	Œ	Uppercase Ligature “OE”
10001101	215	141	8D		Undefined
10001110	216	142	8E	Ž	Uppercase Latin “Z” with Caron
10001111	217	143	8F		Undefined
10010000	220	144	90		Undefined
10010001	221	145	91	‘	Left Single Quotation Mark
10010010	222	146	92	’	Right Single Quotation Mark
10010011	223	147	93	“	Left Double Quotation Mark
10010100	224	148	94	”	Right Double Quotation Mark
10010101	225	149	95	•	Bullet
10010110	226	150	96	–	En Dash
10010111	227	151	97	—	Em Dash
10011000	230	152	98	˜	Small Tilde
10011001	231	153	99	™	Trademark Sign
10011010	232	154	9A	š	Lowercase Latin “s” with Caron
10011011	233	155	9B	›	Right Single Angle Quotation Mark
10011100	234	156	9C	œ	Lowercase Ligature “oe”
10011101	235	157	9D		Undefined
10011110	236	158	9E	ž	Lowercase Latin “z” with Caron
10011111	237	159	9F	Ÿ	Uppercase Latin “Y” with Dieresis

Latin 1 (ISO 8859-1) Character Set Extensions

Binary	Oct	Dec	Hex	Chr	Description
10100000	240	160	A0		Non-Breaking Space
10100001	241	161	A1	¡	Inverted Exclamation Point
10100010	242	162	A2	¢	Cent Sign
10100011	243	163	A3	£	Pound Sign
10100100	244	164	A4	¤	Currency Sign
10100101	245	165	A5	¥	Yen Sign
10100110	246	166	A6	¦	Broken Vertical Bar
10100111	247	167	A7	§	Section Sign
10101000	250	168	A8	¨	Dieresis
10101001	251	169	A9	©	Copyright Sign
10101010	252	170	AA	ª	Feminine Ordinal Indicator
10101011	253	171	AB	«	Left Double Angle Quotation Mark
10101100	254	172	AC	¬	NOT Sign
10101101	255	173	AD	-	Soft Hyphen
10101110	256	174	AE	®	Registered Sign
10101111	257	175	AF	¯	Macron
10110000	260	176	B0	°	Degree Sign or Ring
10110001	261	177	B1	±	Plus-Minus Sign
10110010	262	178	B2	²	Superscript Digit “2”
10110011	263	179	B3	³	Superscript Digit “3”
10110100	264	180	B4	´	Acute Accent
10110101	265	181	B5	µ	Micro (Mu)
10110110	266	182	B6	¶	Pilcrow
10110111	267	183	B7	·	Middle Dot
10111000	270	184	B8	¸	Cedilla
10111001	271	185	B9	¹	Superscript Digit “1”
10111010	272	186	BA	º	Masculine Ordinal Indicator
10111011	273	187	BB	»	Right Double Angle Quotation Mark
10111100	274	188	BC	¼	Vulgar Fraction One-Quarter
10111101	275	189	BD	½	Vulgar Fraction One-Half
10111110	276	190	BE	¾	Vulgar Fraction Three-Quarters
10111111	277	191	BF	¿	Inverted Question Mark
11000000	300	192	C0	À	Uppercase Latin “A” with Grave Accent
11000001	301	193	C1	Á	Uppercase Latin “A” with Acute Accent
11000010	302	194	C2	Â	Uppercase Latin “A” with Circumflex
11000011	303	195	C3	Ã	Uppercase Latin “A” with Tilde
11000100	304	196	C4	Ä	Uppercase Latin “A” with Dieresis
11000101	305	197	C5	Å	Uppercase Latin “A” with Ring or Angstrom Symbol
11000110	306	198	C6	Æ	Uppercase Latin Ligature “AE”
11000111	307	199	C7	Ç	Uppercase Latin “C” with Cedilla
11001000	310	200	C8	È	Uppercase Latin “E” with Grave Accent
11001001	311	201	C9	É	Uppercase Latin “E” with Acute Accent
11001010	312	202	CA	Ê	Uppercase Latin “E” with Circumflex
11001011	313	203	CB	Ë	Uppercase Latin “E” with Dieresis
11001100	314	204	CC	Ì	Uppercase Latin “I” with Grave Accent
11001101	315	205	CD	Í	Uppercase Latin “I” with Acute Accent
11001110	316	206	CE	Î	Uppercase Latin “I” with Circumflex
11001111	317	207	CF	Ï	Uppercase Latin “I” with Dieresis

Binary	Oct	Dec	Hex	Chr	Description
11010000	320	208	D0	Ð	Uppercase Latin Eth
11010001	321	209	D1	Ñ	Uppercase Latin “N” with Tilde
11010010	322	210	D2	Ò	Uppercase Latin “O” with Grave Accent
11010011	323	211	D3	Ó	Uppercase Latin “O” with Acute Accent
11010100	324	212	D4	Ô	Uppercase Latin “O” with Circumflex
11010101	325	213	D5	Õ	Uppercase Latin “O” with Tilde
11010110	326	214	D6	Ö	Uppercase Latin “O” with Dieresis
11010111	327	215	D7	×	Multiplication Sign
11011000	330	216	D8	Ø	Uppercase Latin “O” with Stroke
11011001	331	217	D9	Ù	Uppercase Latin “U” with Grave Accent
11011010	332	218	DA	Ú	Uppercase Latin “U” with Acute Accent
11011011	333	219	DB	Û	Uppercase Latin “U” with Circumflex
11011100	334	220	DC	Ü	Uppercase Latin “U” with Dieresis
11011101	335	221	DD	Ý	Uppercase Latin “Y” with Acute Accent
11011110	336	222	DE	Þ	Uppercase Latin Thorn
11011111	337	223	DF	ß	Lowercase Latin Sharp “s” (German “ss”)
11100000	340	224	E0	à	Lowercase Latin “a” with Grave Accent
11100001	341	225	E1	á	Lowercase Latin “a” with Acute Accent
11100010	342	226	E2	â	Lowercase Latin “a” with Circumflex
11100011	343	227	E3	ã	Lowercase Latin “a” with Tilde
11100100	344	228	E4	ä	Lowercase Latin “a” with Dieresis
11100101	345	229	E5	å	Lowercase Latin “a” with Ring
11100110	346	230	E6	æ	Lowercase Latin Ligature “ae”
11100111	347	231	E7	ç	Lowercase Latin “c” with Cedilla
11101000	350	232	E8	è	Lowercase Latin “e” with Grave Accent
11101001	351	233	E9	é	Lowercase Latin “e” with Acute Accent
11101010	352	234	EA	ê	Lowercase Latin “e” with Circumflex
11101011	353	235	EB	ë	Lowercase Latin “e” with Dieresis
11101100	354	236	EC	ì	Lowercase Latin “i” with Grave Accent
11101101	355	237	ED	í	Lowercase Latin “i” with Acute Accent
11101110	356	238	EE	î	Lowercase Latin “i” with Circumflex
11101111	357	239	EF	ï	Lowercase Latin “i” with Dieresis
11110000	360	240	F0	ð	Lowercase Latin Eth
11110001	361	241	F1	ñ	Lowercase Latin “n” with Tilde
11110010	362	242	F2	ò	Lowercase Latin “o” with Grave Accent
11110011	363	243	F3	ó	Lowercase Latin “o” with Acute Accent
11110100	364	244	F4	ô	Lowercase Latin “o” with Circumflex
11110101	365	245	F5	õ	Lowercase Latin “o” with Tilde
11110110	366	246	F6	ö	Lowercase Latin “o” with Dieresis
11110111	367	247	F7	÷	Division Sign
11111000	370	248	F8	ø	Lowercase Latin “o” with Stroke
11111001	371	249	F9	ù	Lowercase Latin “u” with Grave Accent
11111010	372	250	FA	ú	Lowercase Latin “u” with Acute Accent
11111011	373	251	FB	û	Lowercase Latin “u” with Circumflex
11111100	374	252	FC	ü	Lowercase Latin “u” with Dieresis
11111101	375	253	FD	ý	Lowercase Latin “y” with Acute Accent
11111110	376	254	FE	þ	Lowercase Latin Thorn
11111111	377	255	FF	ÿ	Lowercase Latin “y” with Dieresis