TDV 2100 ASSEMBLER

USER'S MANUAL

Nov. 1976

TEXT HANDLER
FORM HANDLER
EXAMINER/EDITOR

TABLE OF CONTENTS

Table of contents (cont'd)

1. INTRODUCTION

This manual has been written to assist the reader to program the Tandberg TDV 2100 in assembly language. Accordingly, this manual assumes that the reader has a good understanding of logic and is familiar with programming.

2. THE HARDWARE ENVIRONMENT OF THE TOS 21 ASSEMBLER

The purpose of an assembly language is to symbolically represent words in a computer. Therefore, before the specifications of the TOS 21 Assembler is described, it is appropriate to provide the reader with a functional overview of the TDV 2100 CPU module (and the 8080 CPU). This section will give the programmer the neccessary background information in order to write efficient programs.

To the programmer, the computer is represented as consisting of the following parts:

(1) Seven working registers in which all data operations occur, and which may be used for addressing memory.

(2) Memory which may hold instructions and/or data.

(3) The program counter, the contents of which indicate the next program instruction to be executed.

(4) The stack pointer, a register which enables various portions of memory to be used as stacks.

(5) Input/Output, which is the interface between a running program and the outside world.

The minimum hardware configuration required for operation of the assembler is a TDV 2114 with floppy disc or cartridge.

Depending on the individual demands of the customer, one may add additional memory (RAM) and additional input/output devices such as:

1. Additional floppy disc drives (up to 4 total)
2. Additional cartridge drives (up to 4 total)
3. Line printer.

2.1 Working Registers

In the TDV 2100 Programmable Terminal the programmer is provided with an 8-bit accumulator and six additional 8-bit "scratch-pad" registers. These seven working registers are accessed via the letters B, C, D, E, H, L and A (for the accumulator), respectively.

Some operations on the working registers in pairs are referenced by the letters B, D, H and PSW. These correspondences are shown as follows:

REGISTER PAIR REFERENCE

Register Pair B →	B	C
Register Pair D →	D	E
Register Pair H →	H	L
Register Pair PSW →	A	Stat. Δ

special status byte

NOTE: When register pair PSW (Program Status Word) is specified, the last (least significant) 8 bits referenced are a special byte reflecting the current status of the machine.

The different bits of the status byte have the following meaning:

Bit 0 : Carry
1 : Free (1)
2 : Parity bit (even)
3 : Free (0)
4 : Aux.carry
5 : Free (0)
6 : Zero
7 : Sign

The Tandberg TDV 2100 Assembler is written as a one-pass assembler and can be used as an on-line one-pass assembler with console keyboard input, or as a two-pass assembler with diskette input and output files.

The Assembler is loaded and executed under TOS 21[1)] control, and control is transferred to TOS 21 after completion.

When entered in On-Line mode, the assembled program may be entered directly into RAM in executable form. The source program which is typed on the console can be saved for later editing and reassembling. The operator may exit to Monitor and run parts of his program, and later re-enter the Assembler with the symbol table intact.

1) TOS 21 = Tandberg Operating System for TDV 2114

Each source statement may be written in free format.

Symbolic addressing is permitted and both defined and undefined symbols are allowed in constants. Expressions may consist of symbols and octal, decimal and hexadecimal numbers, the binary operators + and - and codes for ASCII-characters.

The symbol tables may be placed anywhere in RAM memory and with variable size.

Data statements like DB (define byte) and DW (define word).

A set of pseudo instructions are available in order to direct the assembly process, like ORG (define origin), EQU (assign symbol value), DS (define storage) and END (end of program).

A set of interactional assembly commands to control the information flow of the source program, listing and object code, and to help with the debugging process.

2.2 Memory

The TDV 2100 can be used with both read only memory and read/write memory. A program can cause data to be read from any type of memory, but can only cause data to be written into read/write memory.

The programmer visualizes memory as a sequence of bytes, each of which may contain 8 bits. The bits stored in a memory byte may represent the encoded form of an instruction or may be data.

A description of the different types of memory used in TDV 2100 folloes:

The CPU Module may contain different types of memory:

- x 8k byte of Programmable Read Only Memory (PROM); range of memory address is 0-1FFFH (H=Hexadecimal).
- x 2k byte of Random Access Memory (RAM); range of memory address is 2000-27FFH.

  The Monitor is contained on the CPU Module and occupies 4k bytes from addresses 0 to 0FFFH.

The RAM Memory Module is a random access memory module containing 16384 8-bit Bytes.

The PROM Memory Module is a programmable read only memory containing 16384 8-bit Bytes. It has sockets for 16 Intel 8708 PROMS which are reprogrammable. The mask programmed 2308 can be used in place of the PROM's if so is desired.

The specific location in the 65kByte memory space of both memory modules is selected in 16kByte steps by setting bits 12-15 to the proper value. For this purpose a jumper must be set in the proper position on the board.

The minimum configuration of the TDV 2114 contains one 16k byte RAM Memory Module at address C000H. 8k byte of this memory is reserved for TOS 21 use (addresses E000H - FFFFH), and the remaining 8k is used by the Assembler for program and symbol table. Of the 2k Byte on the CPU Module, TOS requires 256 bytes from 2700 to 27 FFH. The remaining is at the users disposal.

2.3 Program Counter

The program counter (PC) is a 16-bit register, which is accessible to the programmer and the contents of which indicate the address of the next instruction to be executed.

2.4 Stack Pointer

A stack is an area of memory set aside by the programmer in which data or addresses are stored an retrieved by stack operations. Stack operations are performed by several of the machine instructions and facilitate execution of subroutines and handling of program interrupts. The programmer specifies which addresses the stack instructions will operate upon via a special 16-bit register called the stack pointer (SP).

2.5 Input/Output

To the TDV 2100 CPU the outside world consists of 256 input ports and 256 output ports. Each device connected to an I/O port communicates with the central processing unit via data bytes sent to or received from the accumulator, and each port is assigned a physical device number (from 0 to 255). The instructions which perform these data transmissions are described in Section 5.

Communication with peripheral devices may be programmed directly, or I/O routines located in the Monitor may be used. It is strongly recommended that the user use the Monitor routines, especially for floppy disc and data cartridge I/O. See the TOS 21 User's Manual for details.

2.6 Program Representation in Memory

A computer program consists of a sequence of instructions. Each instruction enables an elementary operation such as moving a data byte, and arithmetic or logical operation on a data byte, or a change in instruction execution sequence. Instructions are described individually in Section 3.

A program will be stored in memory as a sequence of bits which represent the instructions of the program, and which will be represented by hexadecimal digits. The memory address of the next instruction to be executed is held in the program counter. Just before each instruction is executed, the program counter is advanced to the address of the next sequential instruction. Program execution proceeds sequentially unless a transfer-of-control instruction (jump, call or return) is executed, which causes the program counter to be set to a specified address. Execution then continues sequentially from this new address in memory.

Upon examining the contents of a memory byte, there is no way of telling whether the byte contains an encoded instruction or data.

It is up to the logic of a program to insure that data is not misinterpreted as an instruction code. The machine instructions may require 1, 2 or 3 bytes to encode an instruction; in each case the program counter is automatically advanced to the start of the next instruction. In order to avoid errors, the programmer must be sure that a data byte does not follow an instruction when another instruction is expected.

A class of instructions (referred to as transfer-of-control instructions) cause program execution to branch to an instruction that may be anywhere in memory.

3. THE TDV 2100 ASSEMBLY LANGUAGE

A program in the TDV 2100 Assembly language, as in most assembly languages, consits of a series of lines, each of which contains a command to the assembler or an instruction or constant which is to be assembled into a particular memory location. The particular memory location is selected by the value of an internal variable called the location counter. After each instruction or constant is assembled, the location counter is incremented to the next available memory location. Thus, instructions and constants on successive lines are assembled into successive memory locations.

The purpose of assembly language is to symbolically represent words in a computer, in this case the 8-bit bytes of the TDV 2100 CPU. Therefore, before the TDV 2100 Assembly language is described, it is appropriate to introduce the reader to the basic elements of the language.

3.1 Basic elements of the TDV 2100 Assembly Language

3.1.1 Characters

The most basic element is a character from which all more complex elements are formed. The character set used is seven bits ASCII right justified in an eight-bit field with left bit (parity bit) zero.

Some characters have a special meaning, either as commands, arithmetic operators or special symbols. Letters and digits are generally used to construct more complex linguistic elements. An ASCII-constant is a single character enclosed in quotes (').

3.2.1 Numbers

There are three kinds of numbers in the TDV 2100 Assembly language: octal numbers, decimal numbers and hexadecimal numbers.

Octal numbers are formed from the digits 0 through 7 and terminated by the character O or Q.

TEXT HANDLER
FORM HANDLER

Decimal numbers are formed from the digits 0 through 9.

Hexadecimal numbers are formed from the digits 0 through 9 and the letters A through F and terminated by the character H. A hexadecimal number should always start with a digit.

All types of numbers may be preceeded by an unary operator + or -. Negative numbers are represented internally in two's complement notation, and if the arithmetic numbers are considered to be signed integers, the range of possible single byte integer values is from -128_{10} to 127_{10} and

the range of possible double byte integer values is from -32768_{10} to 32767_{10}. Alternatively, integer numbers are often considered to range from 0 to 256_{10} (single byte) or from 0 to 65535_{10} (double byte) when having no sign.

3.1.3 Symbols

Symbols consist of a string of letters and digits starting with a letter. Any number of letters and digits may be used in a symbol, but only the first five characters distinguish symbols. Thus, ABCDE1 and ABCDE2 are treated as the same symbol.

Every symbol either has a numeric value and is said to be defined or does not have a value and is said to be undefined.

3.1.4 Expressions

Expressions consists of numbers, symbols and/or ASCII-constants separated or preceeded by the arithmetic operators + and -.

The value of an expression is the arithmetic sum of the values of the symbols and numbers.

All symbols included in an expression should be defined. Undefined symbols are not permitted.

3.1.5 Examples

Legal Constants	Internal Representation (hexa)
100	0064
-100	FF9C
+100Q	0040
-100Q	FFC0
1234H	1234
-1234H	EDCC
65535	FFFF
-1	FFFF
177777Q	FFFF
32767	7FFF
32768	8000
-32768	8000
'A'	0041

Legal Symbols

A
AB
XYZ
C123
A1X2B
LONGLABEL

Legal Expressions (symbols are assumed to be defined)

```
A+10
1FH+AB
XYZ-A+25
77Q+0FH-19
LABEL-100H
'A'-'Z'+1
```

3.2 Assembly Language Format

The assembly program consists of a sequence of symbolic statements. Each statement belongs to one of three statement cathegories: assembly instructions, pseudo instructions or commands.

Assembly language instructions must adhere to a fixed set of rules as described below. Each instruction is divided into four separate and distinct fields in the following order:

1. Label Field
2. Code Field
3. Operand Field
4. Comment Field

The assembler allows free format input of symbolic source statements, that is, the different fields may be separated by any number of blanks or tabulators. As tabulator can be used the right arrow (→) on the cursor pad on the keyboard.

3.2.1 Label Field

The label field is an optional field, which if present consists of a symbol followed by a colon (:).

When a symbol is defined to be a label in this way, the value of the symbol will be the current value of the location counter.

3.2.2 Code Field

This field contains a code which identifies the machine operation (add, subtract, jump, etc.) to be performed. There are altogether 78 different basic operations and each of these are identified by a mneumonic consisting of a two-to-four-letter symbol.

Spaces and/or tabulators separating the label field and the code field are optional.

Spaces separating the label field and the code field are optional.

When the operation code requires a succeeding operand, then the code field and the operand field must be separated by at least one space or tabulator.

3.2.3 Operand Field

This field contains information used in conjunction with the code field in order to define precisely the operation to be performed by the instruction. Depending upon the code field, the operand field may be absent, or may consist of one item or two items separated by a comma (,).

There are two types of information that may be requested as items of an operand field:

1. Register or memory reference

Several of the basic machine operations is divided into sub-operations by additional information specified in the operand field. This information will be combined with the basic operation code byte, thus generating a complete machine instruction, i.e. the first byte of the object code.

Information that might be required is as follows:

A single register (or code to indicate memory reference) to serve as the source or destination in a data operation. Registers are specified as follows:

Specification	Representing
B	Register B
C	" C
D	" D
E	" E
H	" H
L	" L
M	A memory reference
A	Register A

A double register to serve as the source or destination in a data operation. Register pairs are specified as follows:

Specification	Representing
B	Register pair B and C
D	" " D and E
H	" " H and L
PSW	One byte indicating the state of the condition flip-flops (bits), and Register A
SP	The 16-bit stack pointer register

2. Data

This type of information is requested when the complete machine instruction requires a single or double byte data operand.

The information required is specified either as a number, a symbol, and expression or as an ASCII constant.

A single byte data operand will be decoded and stored into byte No. 2 of the complete object code.

A double byte data operand will be decoded and stored into byte No. 2 and No. 3 of the complete object code with byte No. 2 as the least significant part.

3.2.4 Comment Field

The comment field is optional.

A comment is introduced by the character ; (semicolon) and continues to the end of a line. As already stated, comments are ignored by the assembler. Any characters may occur within a comment, except a carriage return which would end the comment.

A comment can stand alone on a line.

3.3 Data Statements

The different ways in which data can be specified by an assembly program will be described below.

3.3.1 Define Byte of Data (DB)

In order to define a sequence of single byte data elements one must apply a statement in the following format:

Label	Code	Operand	Comment
symbol:	DB	list	; DEFINE BYTE

where the label and the comments are optional and "list" is a list of either:

1. Numbers, symbols and arithmetic expressions which evaluate to eight-bit data elemtns.

2. Strings of ASCII characters enclosed in quotes.

The different elements of the list must be separated by commas (,).

The eight-bit values generated by the data list will be stored sequentially into memory starting with the byte addressed by "symbol".

Thus, the statement:

```
LABEL:    DB    37Q,  'TEXT',  99
```

require six byte of memory storage and the next instruction will be assembled into memory address "LABEL + 6".

3.3.2 Define Word of Data (DW)

In order to define a sequence of double byte (word) data elements one must apply a statement in the following format:

Label	Code	Operand	Comment
symbol:	DW	list	; DEFINE WORD

where the label and the comments are optional and "list" is a list of: .

Numbers, symbols and arithmetic expressions which evaluate to sixteen-bit data elements.

The different elements of the list must be separated by commas.

The double byte values generated by the data list will be stored sequentially into memory, starting with the byte addressed by "symbol" and with the least significant byte preceeding the other.

Thus, the statement:

```
LABEL:  DW  1234H,999
```

require four byte of memory storage and the next instruction will be assembled into memory address "LABEL + 4".

3.4 Pseudo Instructions

The purpose of pseudo instructions is to direct the assembler and to define symbol values required by an assembly program.

Pseudo instructions do not cause any object code to be generated. The following instructions are available:

Mnemonic	Description
ORG	Set origin of program counter
DS	Define storage
EQU	Assign value to symbol
END	End of source record
EOF	End of source file

3.4.1 Set Origin (ORG)

To set the location counter of the assembler, a statement in the following format may be applied:

Label	Code	Operand	Comment
symbol:	ORG	expr	; SET ORIGIN

where the label and the comments are optional and "expr" is a single number, symbol or arithmetic expression. Only defined symbols are allowed.

The next instruction will be assembled at memory location (value of "expr").

The value of "symbol" will be equal to the value that the location counter had before the ORG pseudo-instruction was executed.

3.4.2 Define Storage (DS)

In order to reserve a specific number of memory bytes for data storage, a statement in the following format may be applied:

Label	Code	Operand	Comment
LABEL:	DS	expr	; DEFINE STORAGE

where the label and the comments are optional and "expr" is a single number, symbol or arithmetic expression. Only defined symbols are allowed. No data values are assembled into this data storage; in particular the programmer should not assume that these bytes will be zero, or any other specific value.

The next instruction will be assembled at memory location (value of "symbol" + value of "expr").

The DS pseudo-instruction can perform a function equivalent to the ORG (set origin) pseudo-instruction. One may consider the DS instruction to be a relative modification and the ORG instruction to be an absolute modification of the location counter.

3.4.3 Equate (EQU)

In order to assign a specific value to a not previously defined symbol, the EQU pseudo-instruction should be used:

Label	Code	Operand	Comment
name	EQU	expr	; EQUATE

where the comments are optional.

The symbol "name" is assigned the value of "expr" by the assembler. Whenever the symbol "name" is encountered subsequently in the assembly, this value will be used. If the symbol was previously referenced but not defined, the table of undefined symbols will be updated and the value of "name" will be assembled into every memory address from where the symbol was referenced.

3.4.4 End of Program (END)

The END statement signifies to the assembler that the end of the program has been reached. The statement has the following format:

Label	Code	Comment
symbol:	END	; END OF PROGRAM

where the label and the comments are optional.

A program is defined to be a sequence of lines terminated by the END statement.

4. USE OF THE TDV 2100 ASSEMBLER

The TDV 2100 ASSEMBLER is invoked by the command

```
ASM  si=<input file >,so=<hex file>,sl=<list file>,ao=
     <source output file>,al=<error list file>
```

The command is typed by the user on the console keyboard.

The assignments are specified as follows:

<input file> must be a diskette or cartridge file, or the assignment may be omitted, in which case the assembler will run in on-line mode as a one-pass assembler and take input from the console keyboard.

<hex file> may be a diskette or cartridge file or a physical device, such as line-printer or console output, or the assignment may be omitted.

<list file> may be a diskette or cartridge file or a physical device, or the assignment may be omitted.

<source output file > may be a diskette or cartridge file or a physical device, or the assignment may be omitted. This assignment is intended for use when the programmer enters a program from keyboard in on-line mode, and the assembler can thus save the source program on a diskette file for later editing and reassembling.

< error list file > may be a diskette or cartridge file or a physical device, or the assignment may be omitted. If this assignment is made, all error messages will be output to the error list file and will not appear on the standard list file. If omitted, error messages will appear on the standard list file.

Examples:

```
ASM  si=PROG.SRC,so=PROG.HEX,sl=PROG.LST
```

will cause a program in a diskette file named PROG.SRC to be assembled. The assembler creates a diskette file called PROG.HEX which will contain the hexadecimal output, and another file called PROG.LST which contains the listing of the program and error messages if any.

```
ASM  si=PROG.SRC,al=:LP:
```

will assemble PROG.SRC, produce no hex file and no list file, and list any errors on the line printer.

```
ASM  ao=PROG.SRC
```

will cause the assembler to enter the one-pass on-line mode, assemble the program as it is entered on the keyboard and give error messages, and save the source on the diskette.

If neither input nor output files are specified, the assembler will enter the on-line mode and will store the assembled code in RAM, ready for execution. It is then the programmer's responsibility not to interfere with TOS 21, the assembler or its symbol table.

The assembler program occupies 5k bytes of RAM from CC00H, leaving 3k bytes for symboltables. The operator has the opportunity to release some of the space intended for symbol tables by using the .I command (see next section) when in on-line mode.

4.1 Assembly Commands

When used in on-line mode, the assembler will respond to directives called assembler commands.

Assembler commands have a number of different formats. All commands are directives to the assembler to take some action but never to cause any instructions to be assembled. In this section all commands available will be described in detail. Most commands start with a point (.) followed by a single letter.

The available physical devices are each denoted by a symbolic logical device (logdev). The physical devices and their indentifiers are:

logdev	physical device
NONE	dummy device
LP	line printer
RAM	read/write memory

A detailed description of the different assignment commands is given below.

4.1.1 List Device

Format of command: .A L = logdev

where logdev is either NONE or LP

This command will determine which of the available output devices to be connected to the list data-flow.

Thus, the command

.A L= NONE

will turn off the listing.

If the list stream is connected to the dummy device, error messages are output to the console screen.

4.1.2 Object Code Output

The generated object code may be placed directly into memory.

Format of command: .A M = logdev

This command will determine whether the assembled program is to be stored into memory or not.

Thus, the command:

.A M = NONE

directs the assembler not to store the object code into memory.

4.1.3 Initialization of the assembler

Format of command: .I

This command will clear the symbol tables of the assembler so that another program may be assembled without confusion due to doubly defined symbols, and without restarting.

The assembler will respond by printing :

SYMBOL TABLE AREA : XXXXH - YYYYH :

where XXXX and YYYY are the lower and upper limits of the memory area set aside for symbol tables (hexadecimal addresses).

The assembler now expects the user to specify a new symbol table area to the right of the last colon (:). If the old limits are acceptable the user should type a carriage return (↵).

The limits will be set initially the first time the assembler is entered after power-on.

Thus the following conversation:

.I
SYMBOL TABLE AREA : C000H-CBFFH:2000H,26FFH↵

will initiate the assembler and reserve the memory area from 2000H to 26FFH for the symbol tables. The text underlined is typed by the user.

The initialization command also include the automatic assignment of physical devices to the logical list and object outputs.

This automatic assignment of devices will have the same effect as the following sequence of commands:

.A L = LP
.A M = RAM

4.1.4 List Defined Symbols

Command: .S

This command prints out all of the user defined symbols on the device associated with the list stream.

The corresponding values are printed as hexadecimal numbers.

4.1.5 List Undefined Symbols

Command .U

This command will print out on the assigned list device all symbols that are referenced but not defined.

The memory addresses from where the symbols are referenced are printed as hexadecimal numbers.

4.1.6 Kill Defined Symbols

Command: .K symbol

This command is used to remove a sequence of symbols from the defined symbol table. All symbols which were defined later (in time) than the "symbol" specified with the command, "symbol" included, will be removed from the table.

Symbols deleted in this fashion may be reused as if they had never existed.

4.1.7 Start Execution of Program

Command: .G expr

where "expr" is the start address of the assembled program. "expr" may be any legal number, defined symbol or arithmetic expression.

The command to start execution will act like a subroutine call. The return address to the assembler will be pushed into the stack, so that the program execution may be terminated by transferring control back to the assembler if desired.

4.1.8 Call to the TOS 21 Monitor

Command: .M

This command will transfer control of operation to the system monitor, which is a handy tool for program debugging.

Som of the main features of the monitor are as follows:

- Register examine/change
- Memory examine/change
- Hexa dump of memory areas
- Program execution under break-point control
- Return to the assembler is achieved by the command: G

For detailed description of the monitor see the TOS 21 manual.

4.1.9 Return_to_TOS_21

Command .X

This command will transfer control to the operating system.

4.2 Assembly Listings

The format of the assembly listing is shown in App. G. The meaning of the fields in the listings are as follows:

Field 1. Absolute hexadecimal address of every statement.
" 2. Machine instruction code.
" 3. The least significant byte of the operand.
" 4. The most significant byte of the operand.
" 5. Original symbolic assembly statement.

Any field which is not required to represent a complete statement will be filled with blanks.

Symbols not yet defined when referenced will be printed as zeroes (field 3,4). This applies to on-line operation only.

4.3 On-line Features

When the computer is operating in on-line mode with source program input from Console Keyboard, the assembler will display as a heading on each line the current value of the location counter.

The operator may now enter his program in free format. The assembler will assemble the program as it is entered, displaying the assembled code on the screen as the typing progresses. The assembler will tabulate to the next tab stop when the right arrow key (—») is depressed. Spaces will be converted to tab codes except if spaces are typed beyond column 32. (comment field). Op-codes are moved to the op-code field if they are started at the beginning of the line.

The assembler displays an error message immidiately if an error is detected. The operator has the option of retyping the entire statement, in which case he types a carriage return, or he may delete characters in the statement by using the left arrow key («—).

If a source output file is specified, the source will be written in this file.

A feature for examining the contents of memory is provided by the command: /

When typing the slash (/) immediately after the printed location counter, the assembler will print out the contents of the three following memory locations. The location counter will not be updated.

5. ASSEMBLY INSTRUCTION REPERTOIRE

A computer, no matter how sophisticated, can only do what it is "told" to do. One "tells" the computer what to do via a series of coded instructions referred to as a Program. The realm of the programmer is referred to as Software, in contrast to the Hardware that comprises the actual computer equipment. A computer's software refers to all of the programs that have been written for that computer.

When a computer is designed, the engineers provide the Central Processing Unit (CPU) with the ability to perform a particular set of operations. The CPU is designed such that a specific operation is performed when the CPU control logic decodes a particular instruction. Consequently, the operations that can be performed by a CPU define the computer's Instruction Set.

Each computer instruction allows the programmer to initiate the performance of a specific operation. All computers implement certain arithmetic operations in their instruction set, such as an instruction to add the contents of two registers. Often logical operations (e.g., OR the contents of two registers) and register operate instructions (e.g., increment a register) are included in the instruction set. A computer's instruction set will also have instructions that move data between registers, between a register and memory, and between a register and an I/O device. Most instruction sets also provide Conditional Instructions. A conditional instruction specifies an operation to be performed only if certain conditions have been met; for example, jump to a particular instruction if the result of the last operation was zero. Conditional instructions provide a program with a decision-making capability.

By logically organizing a sequence of instructions into a coherent program, the programmer can "tell" the computer to perform a very specific and useful function.

The computer, however, can only execute programs whose instructions are in a binary coded form (i.e., a series of 1's and 0's), that is called Machine Code. Because it would be extremely cumbersome to program in machine code, programming languages have been developed. There are programs available which convert the programming language instructions into machine code that can be interpreted by the processor.

One type of programming language is Assembly Language. A unique assembly language mnemonic is assigned to each of the computer's instructions. The programmer can write a program (called the Source Program) using these mnemonics and certain operands; the source program is then converted into machine instructions (called the Object Code). Each assembly language instruction is converted into one machine code instruction (1 or more bytes) by an Assembler program. Assembly languages are usually machine dependent (i.e., they are usually able to run on only one type of computer).

THE 8080 INSTRUCTION SET

The 8080 instruction set includes five different types of instructions:

- **Data Transfer Group**—move data between registers or between memory and registers
- **Arithmetic Group** – add, subtract, increment or decrement data in registers or in memory
- **Logical Group** – AND, OR, EXCLUSIVE-OR, compare, rotate or complement data in registers or in memory
- **Branch Group** – conditional and unconditional jump instructions, subroutine call instructions and return instructions
- **Stack, I/O and Machine Control Group** – includes I/O instructions, as well as instructions for maintaining the stack and internal control flags.

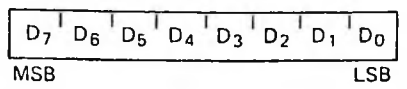
Instruction and Data Formats:

Memory for the 8080 is organized into 8-bit quantities, called Bytes. Each byte has a unique 16-bit binary address corresponding to its sequential position in memory.

Chapter 5 is reprinted by permission fron Intel Corporation

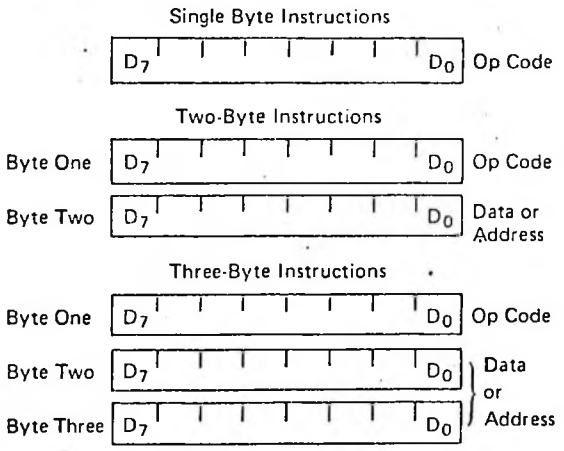
The 8080 can directly address up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access memory (RAM) elements (read/write memory).

Data in the 8080 is stored in the form of 8-bit binary integers:



When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, BIT 0 is referred to as the Least Significant Bit (LSB), and BIT 7 (of an 8 bit number) is referred to as the Most Significant Bit (MSB).

The 8080 program instructions may be one, two or three bytes in length. Multiple byte instructions must be stored in successive memory locations; the address of the first byte is always used as the address of the instructions. The exact instruction format will depend on the particular operation to be executed.



Addressing Modes:

Often the data that is to be operated on is stored in memory. When multi-byte numeric data is used, the data, like instructions, is stored in successive memory locations, with the least significant byte first, followed by increasingly significant bytes. The 8080 has four different modes for addressing data stored in memory or in registers:

- Direct – Bytes 2 and 3 of the instruction contain the exact memory address of the data item (the low-order bits of the address are in byte 2, the high-order bits in byte 3).
- Register – The instruction specifies the register or register-pair in which the data is located.
- Register Indirect – The instruction specifies a register-pair which contains the memory address where the data is located (the high-order bits of the address are in the first register of the pair, the low-order bits in the second).
- Immediate – The instruction contains the data itself. This is either an 8-bit quantity or a 16-bit quantity (least significant byte first, most significant byte second).

Unless directed by an interrupt or branch instruction, the execution of instructions proceeds through consecutively increasing memory locations. A branch instruction can specify the address of the next instruction to be executed in one of two ways:

- Direct – The branch instruction contains the address of the next instruction to be executed. (Except for the 'RST' instruction, byte 2 contains the low-order address and byte 3 the high-order address.)
- Register indirect – The branch instruction indicates a register-pair which contains the address of the next instruction to be executed. (The high-order bits of the address are in the first register of the pair, the low-order bits in the second.)

The RST instruction is a special one-byte call instruction (usually used during interrupt sequences). RST includes a three-bit field; program control is transferred to the instruction whose address is eight times the contents of this three-bit field.

Condition Flags:

There are five condition flags associated with the execution of instructions on the 8080. They are Zero, Sign, Parity, Carry, and Auxiliary Carry, and are each represented by a 1-bit register in the CPU. A flag is "set" by forcing the bit to 1; "reset" by forcing the bit to 0.

Unless indicated otherwise, when an instruction affects a flag, it affects it in the following manner:

Zero: If the result of an instruction has the value 0, this flag is set; otherwise it is reset.

Sign: If the most significant bit of the result of the operation has the value 1, this flag is set; otherwise it is reset.

Parity: If the modulo 2 sum of the bits of the result of the operation is 0, (i.e., if the result has even parity), this flag is set; otherwise it is reset (i.e., if the result has odd parity).

Carry: If the instruction resulted in a carry (from addition), or a borrow (from subtraction or a comparison) out of the high-order bit, this flag is set; otherwise it is reset.

Auxiliary Carry: If the instruction caused a carry out of bit 3 and into bit 4 of the resulting value, the auxiliary carry is set; otherwise it is reset. This flag is affected by single precision additions, subtractions, increments, decrements, comparisons, and logical operations, but is principally used with additions and increments preceding a DAA (Decimal Adjust Accumulator) instruction.

Symbols and Abbreviations:

The following symbols and abbreviations are used in the subsequent description of the 8080 instructions:

SYMBOLS	MEANING
accumulator	Register A
addr	16-bit address quantity
data	8-bit data quantity
data 16	16-bit data quantity
byte 2	The second byte of the instruction
byte 3	The third byte of the instruction
port	8-bit address of an I/O device
r,r1,r2	One of the registers A,B,C,D,E,H,L
DDD,SSS	The bit pattern designating one of the registers A,B,C,D,E,H,L (DDD=destination, SSS=source):

DDD or SSS	REGISTER NAME
111	A
000	B
001	C
010	D
011	E
100	H
101	L

rp — One of the register pairs:

B represents the B,C pair with B as the high-order register and C as the low-order register;

D represents the D,E pair with D as the high-order register and E as the low-order register;

H represents the H,L pair with H as the high-order register and L as the low-order register;

SP represents the 16-bit stack pointer register.

RP — The bit pattern designating one of the register pairs B,D,H,SP:

RP	REGISTER PAIR
00	B-C
01	D-E
10	H-L
11	SP

rh	The first (high-order) register of a designated register pair.
rl	The second (low-order) register of a designated register pair.
PC	16-bit program counter register (PCH and PCL are used to refer to the high-order and low-order 8 bits respectively).
SP	16-bit stack pointer register (SPH and SPL are used to refer to the high-order and low-order 8 bits respectively).
r_m	Bit m of the register r (bits are number 7 through 0 from left to right).
Z,S,P,CY,AC	The condition flags: Zero, Sign, Parity, Carry, and Auxiliary Carry, respectively.
( )	The contents of the memory location or registers enclosed in the parentheses.
←	"Is transferred to"
∧	Logical AND
∀	Exclusive OR
∨	Inclusive OR
+	Addition
–	Two's complement subtraction
*	Multiplication
↔	"Is exchanged with"
—	The one's complement (e.g., $(\overline{A})$)
n	The restart number 0 through 7
NNN	The binary representation 000 through 111 for restart number 0 through 7 respectively.

Description Format:

The following pages provide a detailed description of the instruction set of the 8080. Each instruction is described in the following manner:

1. The MAC 80 assembler format, consisting of the instruction mnemonic and operand fields, is printed in BOLDFACE on the left side of the first line.
2. The name of the instruction is enclosed in parenthesis on the right side of the first line.
3. The next line(s) contain a symbolic description of the operation of the instruction.
4. This is followed by a narative description of the operation of the instruction.
5. The following line(s) contain the binary fields and patterns that comprise the machine instruction.

6. The last four lines contain incidental information about the execution of the instruction. The number of machine cycles and states required to execute the instruction are listed first. If the instruction has two possible execution times, as in a Conditional Jump, both times will be listed, separated by a slash. Next, any significant data addressing modes (see Page 4-2) are listed. The last line lists any of the five Flags that are affected by the execution of the instruction.

Data Transfer Group:

This group of instructions transfers data to and from registers and memory. Condition flags are not affected by any instruction in this group.

MOV r1, r2 (Move Register)

(r1) ← (r2)

The content of register r2 is moved to register r1.

0	1	D	D	D	S	S	S

Cycles: 1
States: 5
Addressing: register
Flags: none

MOV r, M (Move from memory)

(r) ← ((H) (L))

The content of the memory location, whose address is in registers H and L, is moved to register r.

0	1	D	D	D	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

MOV M, r (Move to memory)

((H) (L)) ← (r)

The content of register r is moved to the memory location whose address is in registers H and L.

0	1	1	1	0	S	S	S

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

MVI r, data (Move Immediate)

(r) ← (byte 2)

The content of byte 2 of the instruction is moved to register r.

0	0	D	D	D	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: none

MVI M, data (Move to memory immediate)

((H) (L)) ← (byte 2)

The content of byte 2 of the instruction is moved to the memory location whose address is in registers H and L.

0	0	1	1	0	1	1	0
data							

Cycles: 3
States: 10
Addressing: immed./reg. indirect
Flags: none

LXI rp, data 16 (Load register pair immediate)

(rh) ← (byte 3),
(rl) ← (byte 2)

Byte 3 of the instruction is moved into the high-order register (rh) of the register pair rp. Byte 2 of the instruction is moved into the low-order register (rl) of the register pair rp.

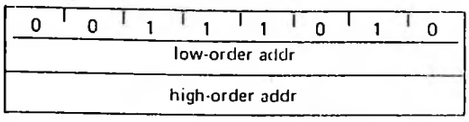
0	0	R	P	0	0	0	1
low-order data							
high-order data							

Cycles: 3
States: 10
Addressing: immediate
Flags: none

LDA addr (Load Accumulator direct)

(A) ← ((byte 3)(byte 2))

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register A.

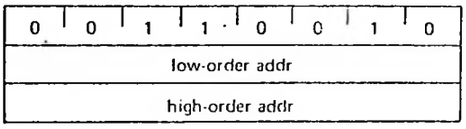


Cycles: 4
States: 13
Addressing: direct
Flags: none

STA addr (Store Accumulator direct)

((byte 3)(byte 2)) ← (A)

The content of the accumulator is moved to the memory location whose address is specified in byte 2 and byte 3 of the instruction.



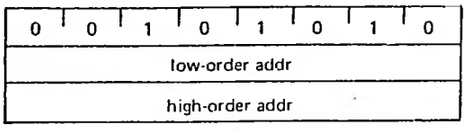
Cycles: 4
States: 13
Addressing: direct
Flags: none

LHLD addr (Load H and L direct)

(L) ← ((byte 3)(byte 2))

(H) ← ((byte 3)(byte 2) + 1)

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register L. The content of the memory location at the succeeding address is moved to register H.



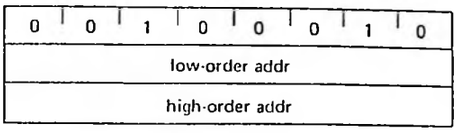
Cycles: 5
States: 16
Addressing: direct
Flags: none

SHLD addr (Store H and L direct)

((byte 3)(byte 2)) ← (L)

((byte 3)(byte 2) + 1) ← (H)

The content of register L is moved to the memory location whose address is specified in byte 2 and byte 3. The content of register H is moved to the succeeding memory location.

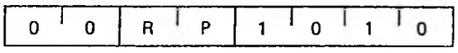


Cycles: 5
States: 16
Addressing: direct
Flags: none

LDAX rp (Load accumulator indirect)

(A) ← ((rp))

The content of the memory location, whose address is in the register pair rp, is moved to register A. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.

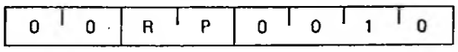


Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

STAX rp (Store accumulator indirect)

((rp)) ← (A)

The content of register A is moved to the memory location whose address is in the register pair rp. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.



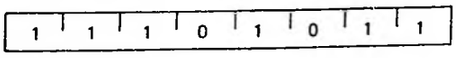
Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

XCHG (Exchange H and L with D and E)

(H) ←→ (D)

(L) ←→ (E)

The contents of registers H and L are exchanged with the contents of registers D and E.



Cycles: 1
States: 4
Addressing: register
Flags: none

Arithmetic Group:

This group of instructions performs arithmetic operations on data in registers and memory.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Carry, and Auxiliary Carry flags according to the standard rules.

All subtraction operations are performed via two's complement arithmetic and set the carry flag to one to indicate a borrow and clear it to indicate no borrow.

ADD r (Add Register)

(A) ← (A) + (r)

The content of register r is added to the content of the accumulator. The result is placed in the accumulator.

1	0	0	0	0	S	S	S

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

ADD M (Add memory)

(A) ← (A) + ((H) (L))

The content of the memory location whose address is contained in the H and L registers is added to the content of the accumulator. The result is placed in the accumulator.

1	0	0	0	0	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

ADI data (Add immediate)

(A) ← (A) + (byte 2)

The content of the second byte of the instruction is added to the content of the accumulator. The result is placed in the accumulator.

1	1	0	0	0	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

ADC r (Add Register with carry)

(A) ← (A) + (r) + (CY)

The content of register r and the content of the carry bit are added to the content of the accumulator. The result is placed in the accumulator.

1	0	0	0	1	S	S	S

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

ADC M (Add memory with carry)

(A) ← (A) + ((H) (L)) + (CY)

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are added to the accumulator. The result is placed in the accumulator.

1	0	0	0	1	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

ACI data (Add immediate with carry)

(A) ← (A) + (byte 2) + (CY)

The content of the second byte of the instruction and the content of the CY flag are added to the contents of the accumulator. The result is placed in the accumulator.

1	1	0	0	1	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

SUB r (Subtract Register)

(A) ← (A) − (r)

The content of register r is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	0	0	1	0	S	S	S

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

SUB M (Subtract memory)

(A) ← (A) – ((H) (L))

The content of the memory location whose address is contained in the H and L registers is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	0	0	1	0	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

SUI data (Subtract immediate)

(A) ← (A) – (byte 2)

The content of the second byte of the instruction is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	1	0	1	0	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

SBB r (Subtract Register with borrow)

(A) ← (A) – (r) – (CY)

The content of register r and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	0	0	1	1	S	S	S

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

SBB M (Subtract memory with borrow)

(A) ← (A) – ((H) (L)) – (CY)

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	0	0	1	1	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

SBI data (Subtract immediate with borrow)

(A) ← (A) – (byte 2) – (CY)

The contents of the second byte of the instruction and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	1	0	1	1	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

INR r (Increment Register)

(r) ← (r) + 1

The content of register r is incremented by one. Note: All condition flags except CY are affected.

0	0	D	D	D	1	0	0

Cycles: 1
States: 5
Addressing: register
Flags: Z,S,P,AC

INR M (Increment memory)

((H) (L)) ← ((H) (L)) + 1

The content of the memory location whose address is contained in the H and L registers is incremented by one. Note: All condition flags except CY are affected.

0	0	1	1	0	1	0	0

Cycles: 3
States: 10
Addressing: reg. indirect
Flags: Z,S,P,AC

DCR r (Decrement Register)

(r) ← (r) – 1

The content of register r is decremented by one Note: All condition flags except CY are affected.

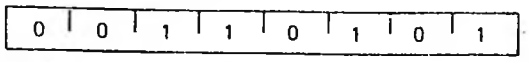
0	0	D	D	D	1	0	1

Cycles: 1
States: 5
Addressing: register
Flags: Z,S,P,AC

DCR M (Decrement memory)

((H) (L)) ← ((H) (L)) – 1

The content of the memory location whose address is contained in the H and L registers is decremented by one. Note: All condition flags except CY are affected.

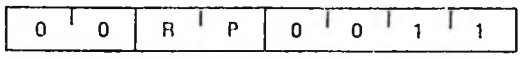


Cycles: 3
States: 10
Addressing: reg. indirect
Flags: Z,S,P,AC

INX rp (Increment register pair)

(rh) (rl) ← (rh) (rl) + 1

The content of the register pair rp is incremented by one. Note: No condition flags are affected.

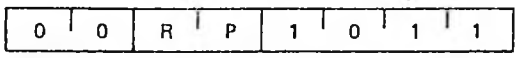


Cycles: 1
States: 5
Addressing: register
Flags: none

DCX rp (Decrement register pair)

(rh) (rl) ← (rh) (rl) – 1

The content of the register pair rp is decremented by one. Note: No condition flags are affected.

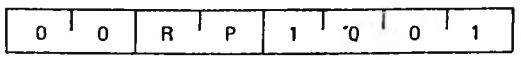


Cycles: 1
States: 5
Addressing: register
Flags: none

DAD rp (Add register pair to H and L)

(H) (L) ← (H) (L) + (rh) (rl)

The content of the register pair rp is added to the content of the register pair H and L. The result is placed in the register pair H and L. Note: Only the CY flag is affected. It is set if there is a carry out of the double precision add; otherwise it is reset.



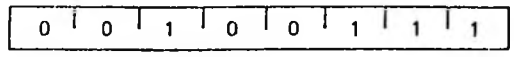
Cycles: 3
States: 10
Addressing: register
Flags: CY

DAA (Decimal Adjust Accumulator)

The eight-bit number in the accumulator is adjusted to form two four-bit Binary-Coded-Decimal digits by the following process:

1. If the value of the least significant 4 bits of the accumulator is greater than 9 or if the AC flag is set, 6 is added to the accumulator.
2. If the value of the most significant 4 bits of the accumulator is now greater than 9, or if the CY flag is set, 6 is added to the most significant 4 bits of the accumulator.

NOTE: All flags are affected.



Cycles: 1
States: 4
Flags: Z,S,P,CY,AC

Logical Group:

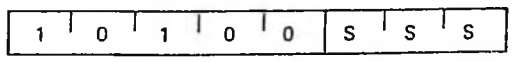
This group of instructions performs logical (Boolean) operations on data in registers and memory and on condition flags.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Auxiliary Carry, and Carry flags according to the standard rules.

ANA r (AND Register)

(A) ← (A) ∧ (r)

The content of register r is logically anded with the content of the accumulator. The result is placed in the accumulator. The CY flag is cleared.

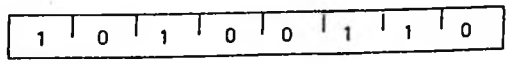


Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

ANA M (AND memory)

(A) ← (A) ∧ ((H) (L))

The contents of the memory location whose address is contained in the H and L registers is logically anded with the content of the accumulator. The result is placed in the accumulator. The CY flag is cleared.

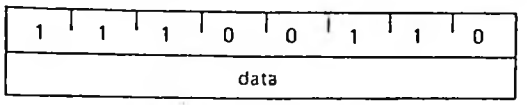


Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

ANI data (AND immediate)

(A) ← (A) ∧ (byte 2)

The content of the second byte of the instruction is logically anded with the contents of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

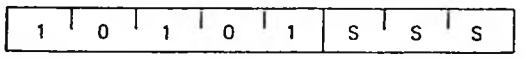


Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

XRA r (Exclusive OR Register)

(A) ← (A) ∀ (r)

The content of register r is exclusive-or'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

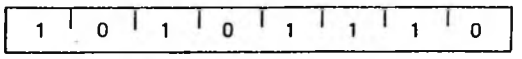


Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

XRA M (Exclusive OR Memory)

(A) ← (A) ∀ ((H) (L))

The content of the memory location whose address is contained in the H and L registers is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

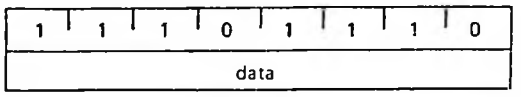


Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

XRI data (Exclusive OR immediate)

(A) ← (A) ∀ (byte 2)

The content of the second byte of the instruction is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

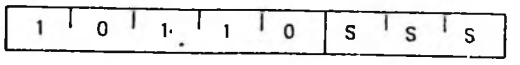


Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

ORA r (OR Register)

(A) ← (A) V (r)

The content of register r is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

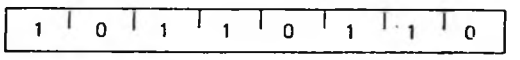


Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

ORA M (OR memory)

(A) ← (A) V ((H) (L))

The content of the memory location whose address is contained in the H and L registers is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

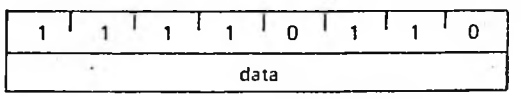


Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

ORI data (OR Immediate)

(A) ← (A) V (byte 2)

The content of the second byte of the instruction is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. The CY and AC flags are cleared.

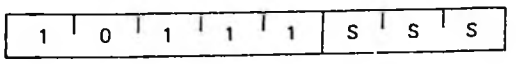


Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

CMP r (Compare Register)

(A) − (r)

The content of register r is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. The Z flag is set to 1 if (A) = (r). The CY flag is set to 1 if (A) < (r).

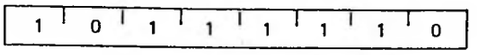


Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

CMP M (Compare memory)

(A) − ((H) (L))

The content of the memory location whose address is contained in the H and L registers is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. The Z flag is set to 1 if (A) = ((H) (L)). The CY flag is set to 1 if (A) < ((H) (L)).

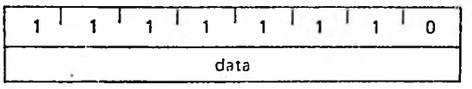


Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

CPI data (Compare immediate)

(A) − (byte 2)

The content of the second byte of the instruction is subtracted from the accumulator. The condition flags are set by the result of the subtraction. The Z flag is set to 1 if (A) = (byte 2). The CY flag is set to 1 if (A) < (byte 2).

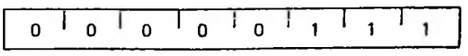


Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

RLC (Rotate left)

$(A_{n+1}) \leftarrow (A_n)$; $(A_0) \leftarrow (A_7)$
$(CY) \leftarrow (A_7)$

The content of the accumulator is rotated left one position. The low order bit and the CY flag are both set to the value shifted out of the high order bit position. Only the CY flag is affected.

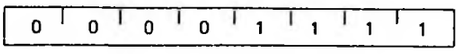


Cycles: 1
States: 4
Flags: CY

RRC (Rotate right)

$(A_n) \leftarrow (A_{n-1})$; $(A_7) \leftarrow (A_0)$
$(CY) \leftarrow (A_0)$

The content of the accumulator is rotated right one position. The high order bit and the CY flag are both set to the value shifted out of the low order bit position. Only the CY flag is affected.

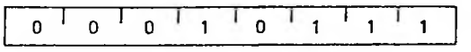


Cycles: 1
States: 4
Flags: CY

RAL (Rotate left through carry)

$(A_{n+1}) \leftarrow (A_n)$; $(CY) \leftarrow (A_7)$
$(A_0) \leftarrow (CY)$

The content of the accumulator is rotated left one position through the CY flag. The low order bit is set equal to the CY flag and the CY flag is set to the value shifted out of the high order bit. Only the CY flag is affected.

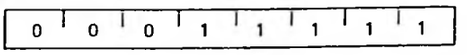


Cycles: 1
States: 4
Flags: CY

RAR (Rotate right through carry)

$(A_n) \leftarrow (A_{n+1})$; $(CY) \leftarrow (A_0)$
$(A_7) \leftarrow (CY)$

The content of the accumulator is rotated right one position through the CY flag. The high order bit is set to the CY flag and the CY flag is set to the value shifted out of the low order bit. Only the CY flag is affected.

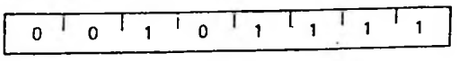


Cycles: 1
States: 4
Flags: CY

CMA (Complement accumulator)

$(A) \leftarrow (\overline{A})$

The contents of the accumulator are complemented (zero bits become 1, one bits become 0). No flags are affected.



Cycles: 1
States: 4
Flags: none

CMC (Complement carry)

(CY) ← $(\overline{CY})$

The CY flag is complemented. No other flags are affected.

0	0	1	1	1	1	1	1

Cycles: 1
States: 4
Flags: CY

STC (Set carry)

(CY) ← 1

The CY flag is set to 1. No other flags are affected.

0	0	1	1	0	1	1	1

Cycles: 1
States: 4
Flags: CY

Branch Group:

This group of instructions alter normal sequential program flow.

Condition flags are not affected by any instruction in this group.

The two types of branch instructions are unconditional and conditional. Unconditional transfers simply perform the specified operation on register PC (the program counter). Conditional transfers examine the status of one of the four processor flags to determine if the specified branch is to be executed. The conditions that may be specified are as follows:

CONDITION	CCC
NZ – not zero (Z = 0)	000
Z – zero (Z = 1)	001
NC – no carry (CY = 0)	010
C – carry (CY = 1)	011
PO – parity odd (P = 0)	100
PE – parity even (P = 1)	101
P – plus (S = 0)	110
M – minus (S = 1)	111

JMP addr (Jump)

(PC) ← (byte 3) (byte 2)

Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.

1	1	0	0	0	0	1	1
low-order addr							
high-order addr							

Cycles: 3
States: 10
Addressing: immediate
Flags: none

Jcondition addr (Conditional jump)

If (CCC),

(PC) ← (byte 3) (byte 2)

If the specified condition is true, control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction; otherwise, control continues sequentially.

1	1	C	C	C	0	1	0
low-order addr							
high-order addr							

Cycles: 3
States: 10
Addressing: immediate
Flags: none

CALL addr (Call)

((SP) – 1) ← (PCH)

((SP) – 2) ← (PCL)

(SP) ← (SP) – 2

(PC) ← (byte 3) (byte 2)

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.

1	1	0	0	1	1	0	1
low-order addr							
high-order addr							

Cycles: 5
States: 17
Addressing: immediate/reg. indirect
Flags: none

XTHL (Exchange stack top with H and L)

(L) ⟷ ((SP))
(H) ⟷ ((SP) + 1)

The content of the L register is exchanged with the content of the memory location whose address is specified by the content of register SP. The content of the H register is exchanged with the content of the memory location whose address is one more than the content of register SP.

1	1	1	0	0	0	1	1

Cycles: 5
States: 18
Addressing: reg. indirect
Flags: none

SPHL (Move HL to SP)

(SP) ← (H) (L)

The contents of registers H and L (16 bits) are moved to register SP.

1	1	1	1	1	0	0	1

Cycles: 1
States: 5
Addressing: register
Flags: none

IN port (Input)

(A) ← (data)

The data placed on the eight bit bi-directional data bus by the specified port is moved to register A.

1	1	0	1	1	0	1	1
port							

Cycles: 3
States: 10
Addressing: direct
Flags: none

OUT port (Output)

(data) ← (A)

The content of register A is placed on the eight bit bi-directional data bus for transmission to the specified port.

1	1	0	1	0	0	1	1
port							

Cycles: 3
States: 10
Addressing: direct
Flags: none

EI (Enable interrupts)

The interrupt system is enabled following the execution of the next instruction.

1	1	1	1	1	0	1	1

Cycles: 1
States: 4
Flags: none

DI (Disable interrupts)

The interrupt system is disabled immediately following the execution of the DI instruction.

1	1	1	1	0	0	1	1

Cycles: 1
States: 4
Flags: none

HLT (Halt)

The processor is stopped. The registers and flags are unaffected.

0	1	1	1	0	1	1	0

Cycles: 1
States: 7
Flags: none

NOP (No op)

No operation is performed. The registers and flags are unaffected.

0	0	0	0	0	0	0	0

Cycles: 1
States: 4
Flags: none

Chapter 5 is printed by permission from Intel Corporation.

INSTRUCTION SET

Summary of Processor Instructions

Mnemonic	Description	Instruction Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock Cycles
MOV r1,r2	Move register to register	0	1	D	D	D	S	S	S	5
MOV M,r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r,M	Move memory to register	0	1	D	D	D	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	7
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	D	D	D	1	0	0	5
DCR r	Decrement register	0	0	D	D	D	1	0	1	5
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10
JZ	Jump on zero	1	1	0	0	1	0	1	0	10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10
JP	Jump on positive	1	1	1	1	0	0	1	0	10
JM	Jump on minus	1	1	1	1	1	0	1	0	10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10
CALL	Call unconditional	1	1	0	0	1	1	0	1	17
CC	Call on carry	1	1	0	1	1	1	0	0	11/17
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
CP	Call on positive	1	1	1	1	0	1	0	0	11/17
CM	Call on minus	1	1	1	1	1	1	0	0	11/17
CPE	Call on parity even	1	1	1	0	1	1	0	0	11/17
CPO	Call on parity odd	1	1	1	0	0	1	0	0	11/17
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	5/11
RNC	Return on no carry	1	1	0	1	0	0	0	0	5/11

Mnemonic	Description	Instruction Code D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock Cycles
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	0	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RST	Restart	1	1	A	A	A	1	1	1	11
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	5
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable interrupts	1	1	1	1	1	0	1	1	4
DI	Disable interrupt	1	1	1	1	0	0	1	1	4
NOP	No operation	0	0	0	0	0	0	0	0	4

NOTES. 1. DDD or SSS – 000 B – 001 C – 010 D – 011 E – 100 H – 101 L – 110 Memory – 111 A.
2. Two possible cycle times, (5/11) indicate instruction cycles dependent on condition flags.

SUMMARY OF INSTRUCTIONS AND SUB-INSTRUCTIONS

MOV r_1, r_2 1 BYTE 6 states

r_1 = DESTINATION	r_2 = SOURCE B	C	D	E	H	L	M	A
B	40	41	42	43	44	45	46	47
C	48	49	4A	4B	4C	4D	4E	4F
D	50	51	52	53	54	55	56	57
E	58	59	5A	5B	5C	5D	5E	5F
H	60	61	62	63	64	65	66	67
L	68	69	6A	6B	6C	6D	6E	6F
M	70	71	72	73	74	75	76	77
A	78	79	7A	7B	7C	7D	7E	7F

INCREMENT/DECRENT INSTRUCTION 1 BYTE 5 states
(x): 11 states

	B	C	D	E	H	L	M(x)	A
INR r	04	0C	14	1C	24	2C	34	3C
DCR r	05	0D	15	1D	25	2D	35	3D

MVI INSTRUCTION 2 BYTES 8 states
(x): 11 states

	B	C	D	E	H	L	M(x)	A
MVI r	06	0E	16	1E	26	2E	36	3E

ACCUMULATOR ARITHMETIC 5 states (x): 8 states

	B	C	D	E	H	L	M(x)	A
ADD r	80	81	82	83	84	85	86	87
ADC r	88	89	8A	8B	8C	8D	8E	8F
SUB r	90	91	92	93	94	95	96	97
SBB r	98	99	9A	9B	9C	9D	9E	9F
ANA r	A0	A1	A2	A3	A4	A5	A6	A7
XRA r	A8	A9	AA	AB	AC	AD	AE	AF
ORA r	B0	B1	B2	B3	B4	B5	B6	B7
CMP r	B8	B9	BA	BB	BC	BD	BE	BF

Mnemonic	Code	Bytes	States
ADI	C6	2	9
ACI	CE	2	9
SUI	D6	2	9
SBI	DE	2	9
ANI	E6	2	9
XRI	EE	2	9
ORI	F6	2	9
CPI	FE	2	9
RLC	07	1	5
RRC	0F	1	5
RAL	17	1	5
RAR	1F	1	5
JMP	C3	3	13
JC	DA	3	13
JNC	D2	3	13
JZ	CA	3	13
JNZ	C2	3	13
JP	F2	3	13
JM	FA	3	13
JPE	E2	3	13
JPO	E2	3	13
CALL	CD	3	22
CC	DC	3	14/17
CNC	D4	3	14/17
CZ	CC	3	14/17
CNZ	C4	3	14/17
CP	F4	3	14/17
CM	FA	3	14/17
CPE	EC	3	14/17
CPO	E4	3	14/17
RET	C9	1	6/12
RC	D8	1	6/12
RNC	D0	1	6/12
RZ	C8	1	6/12
RNZ	C0	1	6/12
RP	F0	1	6/12
RM	F0	1	6/12
RPE	E8	1	6/12
RPO	E0	1	6/12
RST	(ref. tab. 2.4)	1	14
IN	DB	2	12
OUT	D3	2	12
CMA	2F	1	5
STC	37	1	5
CMC	3F	1	5

Mnemonic	Code	Bytes	States
DAA	27	1	5
EI	FB	1	5
DI	F3	1	5
NOP	00	1	5
HLT	76	1	8
LXI B	01	3	13
LXI D	11	3	13
LXI H	21	3	13
LXI SP	31	3	13
PUSH B	C5	1	14
PUSH D	D5	1	14
PUSH H	E5	1	14
PUSH PSW	F5	1	14
POP B	C1	1	13
POP D	D1	1	13
POP H	E1	1	13
POP PSW	F1	1	13
STA	32	3	16
LDA	3A	3	16
XCHG	EB	1	5
XTHL	E3	1	23
PCHL	E9	1	6
DAD B	09	1	13
DAD D	19	1	13
DAD H	29	1	13
DAD SP	39	1	13
STAX B	02	2	9
STAX D	12	2	9
LDAX B	0A	2	9
LDAX D	1A	2	9
INX B	03	1	6
INX D	13	1	6
INX H	23	1	6
INX SP	33	1	6
DXC B	0B	1	6
DXC D	1B	1	6
DCX H	2B	1	6
DCX SP	3B	1	6
SHLD	22	3	21
LHLD	2A	3	21

TDV 2100 character set

Graphic	Hexa Value	Decimal Value	Abbreviation	Comments
	0	0		Not used
	1	1		Not used
	2	2	STX	Video off
	3	3	ETX	Video on
	4	4	ERL	Erace line
	5	5	ENQ	Enquiry
	6	6	ACK	Acknowledge
	7	7	BEL	Bell
	8	8	←	Backspace (cursor left)
	9	9		Not used
	A	10	LF	Line feed
	B	11	↓	Cursor down
	C	12	Rollup	Roll up
	D	13	CR	Carriage return
	E	14	UL	Underline
	F	15	Norm	Normal
	10	16	CL	Cursor load
	11	17		Not used
	12	18		Not used
	13	19		Not used
	14	20		Not used
	15	21	NAK	Negative acknowledge
	16	22	LCLR	Lamp clear
	17	23	Rolldwn	Roll Down
	18	24	→	Cursor right
	19	25	ERP	Erase page
	1A	26		Not used
	1B	27		Not used
	1C	28	↑	Cursor up
	1D	29	↖	Cursor home
	1E	30		Not used
	1F	31		Not used
	20	32	SP	Space
!	21	33	!	Exclamation point
"	22	34	"	Quotation mark
#	23	35	#	Number sign
$	24	36	$	Dollar sign
%	25	37	%	Percent sign
&	26	38	&	Ampersand
'	27	39	'	Apostrophe
(	28	40	(	Opening parenthesis
)	29	41	)	Closing parenthesis
*	2A	42	*	Asterisk
+	2B	43	+	Plus
,	2C	44	,	Comma
-	2D	45	-	Hyphen (Minus)
.	2E	46	.	Period (Decimal)

Graphic	Hexa Value	Decimal Value	Abbreviation	Comments
/	2F	47	/	Slant
0	30	48	0	Zero
1	31	49	1	One
2	32	50	2	Two
3	33	51	3	Three
4	34	52	4	Four
5	35	53	5	Five
6	36	54	6	Six
7	37	55	7	Seven
8	38	56	8	Eight
9	39	57	9	Nine
:	3A	58	:	Colon
;	3B	59	;	Semi-colon
<	3C	60	<	Less than
=	3D	61	=	Equals
>	3E	62	>	Greater than
?	3F	63	?	Question mark
@	40	64	@	Commercial at
A	41	65	A	Uppercase A
B	42	66	B	Uppercase B
C	43	67	C	Uppercase C
D	44	68	D	Uppercase D
E	45	69	E	Uppercase E
F	46	70	F	Uppercase F
G	47	71	G	Uppercase G
H	48	72	H	Uppercase H
I	49	73	I	Uppercase I
J	4A	74	J	Uppercase J
K	4B	75	K	Uppercase K
L	4C	76	L	Uppercase L
M	4D	77	M	Uppercase M
N	4E	78	N	Uppercase N
O	4F	79	O	Uppercase O
P	50	80	P	Uppercase P
Q	51	81	Q	Uppercase Q
R	52	82	R	Uppercase R
S	53	83	S	Uppercase S
T	54	84	T	Uppercase T
U	55	85	U	Uppercase U
V	56	86	V	Uppercase V
W	57	87	W	Uppercase W
X	58	88	X	Uppercase X
Y	59	89	Y	Uppercase Y
Z	5A	90	Z	Uppercase Z

Graphic	Hexa Value	Decimal Value	Abbreviation	Comments
Æ	5B	91	Æ	Uppercase Æ
Ø	5C	92	Ø	Uppercase Ø
Å	5D	93	Å	Uppercase Å
^	5E	94	^	Circumflex, up-arrow
■	5F	95	■	Solid block
`	60	96	`,GRA	Grave accent
a	61	97	a,LCA	Lowercase a
b	62	98	b,LCB	Lowercase b
c	63	99	c,LCC	Lowercase c
d	64	100	d,LCD	Lowercase d
e	65	101	e,LCE	Lowercase e
f	66	102	f,LCF	Lowercase f
g	67	103	g,LCG	Lowercase g
h	68	104	h,LCH	Lowercase h
i	69	105	i,LCI	Lowercase i
j	6A	106	j,LCJ	Lowercase j
k	6B	107	k,LCK	Lowercase k
l	6C	108	l,LCL	Lowercase l
m	6D	109	m,LCM	Lowercase m
n	6E	110	n,LCN	Lowercase n
o	6F	111	o,LCO	Lowercase o
p	70	112	p,LCP	Lowercase p
q	71	113	q,LCQ	Lowercase q
r	72	114	r,LCR	Lowercase r
s	73	115	s,LCS	Lowercase s
t	74	116	t,LCT	Lowercase t
u	75	117	u,LCU	Lowercase u
v	76	118	v,LCV	Lowercase v
w	77	119	w,LCW	Lowercase w
x	78	120	x,LCX	Lowercase x
y	79	121	y,LCY	Lowercase y
z	7A	122	z,LCZ	Lowercase z
æ	7B	123	æ,LCÆ	Lowercase æ
ø	7C	124	ø,LCØ	Lowercase ø
å	7D	125	å,LCÅ	Lowercase å
|	7E	126	|	Left vertical line
	7F	127	DEL	Delete, rubout

SUMMARY OF ASSEMBLY COMMANDS

Command format	Description
.A L = logdev	Assignment of list device
.A M = logdev	Assignment of memory
.I	Initialization of assembler
.S	List defined symbols
.U	List undefined symbols
.K symbol	Kill sequence of defined symbols
.G expr	Start execution of program
.M	Call to monitor
/	Memory examine

ERROR MESSAGES

When an error occurs, a message will be printed in the following format:

ERROR XX LC = YYYY

where XX is error number and YYYY is the current value of the location counter. Error messages will be printed on the assigned list device. If the dummy device is used as list device, error messages will be printed on the system console.

Error no.	Description
1	Illegal instruction (first charcter)
2	Illegal instruction (last item)
3	Illegal instruction (first item)
4	Pseudo instruction syntax error
5	Doubly defined symbols
6	Symbol table full
7	Missing end of string
8	Expression syntax error
9	Illegal terminator of expression
10	Missing operand
11	Missing constant (DB, DW)
12	Undefined symbol in expression
13	Missing END
50	Input record too long
51	Illegal logical source device
52	Illegal logical list device
53	Illegal logical object device
54	Illegal assembly command
55	Insufficient disc or cartridge space
56	Disc or cartridge not ready
57	Cartridge is write-protected