The Q1 Assembler

Preliminary Edition

Table of Contents

Q1 Processor

Six main components form the general structure of the Q1 Processor. The relationship of the components is illustrated in Figure
The components are as follows:

1. Control and Timing Unit
2. Arithmetic and Logic Unit
3. General Purpose Registers
4. Memory
5. Program Counter and Stack Pointer
6. Input/Output Control Unit

The main functions of these six components are summarized by the following descriptions.

Control and Timing Unit

The Control and Timing Unit accepts an instruction and stores the first byte (8 bits) of the instruction into an 8-bit register, called the Instruction Register (I-register). The operation code of the instruction is decoded, and the proper data paths for the other processor components are selected.

Arithmetic and Logic Unit

The Arithmetic and Logic Unit (ALU) performs all arithmetic and logical functions for the processor. When any of the available arithmetic, logical or shift instructions are used, the ALU is under control of a program.

Five Condition Code Flags are also controlled by the ALU. These are two Carry Flags, a Minus Flag, a Zero Flag, and an Even Parity Flag. The condition of one or more of these flags is tested or set by the ALU during execution of arithmetic, logical and shift instructions. A variety of branching instructions also are available by which a program may test the condition of each flag. With each such instruction, the ALU participates in the operation.

General Purpose Registers

Seven 8-bit registers are available for general programming purposes. The registers are identified by the letters A, B, C, D, E, H, L. Register A serves not only as a general purpose register, but also as the Accumulator for arithmetic, logical and shift operations, and as an input/output unit register.

Memory

Processor semiconductor memory is available in increments to a maximum of 65,536 directly addressable bytes. This memory may be obtained as read/write random access memory (RAM) or as a combination of RAM memory and non-volatile read only memory (ROM).

The first 1K of memory is normally ROM and is used for loading and initializing purposes.

Program Counter and Stack Pointer

The Program Counter (P-register) points to the instruction currently being executed by containing the address of that instruction. During instruction execution, the address in the Program Counter is incremented by 1 as the contents of each instruction byte are sent by the Timing and Control Unit to its appropriate destination. The Program Counter is incremented one, two or three times, depending upon the number of bytes in the instruction.

The Address Stack is located in memory and is used to store return addresses from subroutines. A 16-bit counter, which is called the Stack Pointer, points to the location of the next return address. The stack can also be used to save the registers and the condition codes.

Input/Output Control Unit

The Input/Output Control Unit (ICU) controls the flow of all information between the processor and the input and output units (such as a printer, display, keyboard, disk unit, programmable timer, communications). A maximum of 32 devices may be connected to the processor through the ICU. All devices may be under control of the Interrupt System.

Q1 Assembler

Introduction

The minimum machine configuration for the Q1 Assembler consists of 8K CPU, the keyboard, a Disk unit, the printer, and the display.

A source program can be written by using the Editor. The source program is written on the disk as ASCII file. In order to assemble the program, enter the following command: ASM NAME1 NAME2 and press the RETURN button. NAME1 is the file containing the source code and NAME2 is the file where the object code is to be put.

Since the computer will not operate using mnemonic codes and addresses, these must be translated to absolute codes and absolute addresses. This translation is done by the assembler. The assembler reads the program, translates the mnemonic operation codes to hexadecimal operation codes, and assigns memory locations for storing instructions and data. The assembler then generates a block of absolute code which is written as a binary file. The translation from symbolic code is generally one-for-one: one symbolic instruction results in one instruction in absolute code. The assembler keeps track of all memory locations. Once a memory location is assigned to a particular symbolic address, the same memory location will be assigned to all subsequent uses of the associated symbolic name. The translation process includes error diagnostics to detect certain types of coding errors.

The assembler makes two passes over the source code. The first pass generates a symbol table from the labels in the source code and checks for certain error conditions, primarily in syntax and form. The diagnostics will be a one-letter flag, a space, and the faulty line.

These one-letter flags are:

U	Undefined symbol
M	Multiply defined symbol
O	Op code error

W	Weird error
S	Memory overflow
F	File overflow

The second pass generates the object code.

By making two passes over the source code, the assembler is able to provide "forward referencing," i.e., it can reference a symbol that occurs later in the program. The first pass determines the definitions of all the symbols, and the second pass produces the object code, so that forward referencing can be accommodated.

In order to get a symbol table and program listing printed, enter the command: ASM,L NAME1 NAME2. In order to get the error messages only, enter: ASM,N NAME1 NAME2. In order to get the error messages and symbol table only, enter the command: ASM NAME1 NAME2.

Conventions Used in Instruction Description

Condition Code Flags: C = Carry, M = Minus, E = Even, Z = Zero. There is a second Carry flag, C1, which is used by the Decimal Adjust Accumulator–DAA.

u = Flag unaffected by instruction

a = Flag affected by instruction

0 = Flag value is zero as a result of instruction

An understanding of the general organization of the Q1 processor is assumed on the part of the reader of this section. Additionally, sufficient programming experience with assembly languages to be familiar with the basic terms and concepts is assumed.

Numbers mentioned in this section are hexadecimal numbers unless otherwise subscripted, i.e., FF is a hexadecimal number; 255_{10} is a decimal number. The computer, however, operates using binary numbers. Hexadecimal notation is chosen for facility of reading and writing programs, computer inputs and computer outputs.

Assembly Language Coding Conventions

The following coding conventions should be observed when preparing programs to be assembled by the Q1 assembler.

1. The basic elements of a line of coding are a label, an operation code, an operand, and program comments, not all of which need to be present on each line. The general conventions which apply to these elements are the following:

   1.1 Elements of a line are separated from one another by at least one space.

   1.2 An instruction may not be continued from line to line.

   1.3 Instructions, including comments, should not extend beyond column 63.

   1.4 Intervening spaces may not be present in an expression.

   1.5 Registers may be designated by the letters A, B, C, D, E, H, L. (M may be used to designate the contents of the address in H and L.)

2. Labels, sometimes called "names" or "tags" of instructions (including pseudo instructions) begin in the leftmost position of the coding line. Labels consist of one or more alphanumeric characters, the first of which is alphabetic. Labels may not contain special characters. They may be of indefinite length, so long as conventions 1.2 and 1.3 above are observed.

3. Operation codes consist of the capital letters, commas, and register designations described by the instruction mnemonics in the instruction section. Spaces may be substituted for commas in operation codes unless otherwise noted under the instruction description. Operation codes may begin in any column except column 1 if the instruction has no label.

4. Operands consist of the expressions described in the instruction section as operands, and, where appropriate, may be varied through the use of the following symbols:

4.1 %

The byte displacement symbol signals the assembler to shift right by one byte the designated two-byte value. The effect is that the high-order byte of a two-byte value, such as an address, is accessed at program execution time, i.e.:

> LI,H BUFFER%
> The high-order portion of the two-byte address of a location named BUFFER is loaded into register H.
>
> AI TABLE%
> The high-order portion of the two-byte address of TABLE is added to the accumulator.

4.2 +

The address arithmetic operator signals the assembler to add the hexadecimal value to the right of the plus symbol to the designated address. The address increment may be within the range of –8000 to +7FFF. Examples are as follows:

> J PROCESS+3
> A jump occurs to a location which is 3 bytes greater than the address of PROCESS.
>
> J PROCESS+0FF80
> A jump occurs to a location at PROCESS–10_{16}. (FF80 is the two's complement of -10_{16}.)

4.3 Arithmetic and Logic Operators

+	Add
–	Subtract
*	Multiply
&	And each bit of the two operands
!	Or each bit of the two operands
'	Complement each bit of the preceding expression
>	Result is all ones if greater, zeros otherwise
<	Result is all ones if less, zeros otherwise
=	Result is all ones if equal, zeros otherwise

4.4 0

Hexadecimal values are designated by a leading zero. Values of 9 or less may be coded without a leading zero. For example:

> 0FF
> The decimal value 255 has the hexadecimal equivalent of FF.
>
> 010
> The decimal value 16 has the hexadecimal equivalent of 10.
>
> 9
> The decimal value 9 has the hexadecimal equivalent of 9. No leading zero is required.

4.5 "

ASCII values are designated by enclosing the ASCII characters in double quotation marks. All ASCII characters, including spaces, may be enclosed within the quotation marks with the following exception: the ASCII symbol for quotation marks may not appear among the enclosed characters. Instead, quotation marks may be represented by the equivalent hexadecomal code 22_{16}. Examples are:

```
CON     "FORWARDING ADDRESS IS:"
CON     "100 MAIN STREET"
CON     "The quotation reads," 022, "veni, vidi,
        vici.", 022
LI,1    "K"
```

4.6 *

The asterisk may be used to designate substitution of the contents of the program counter (a 16-bit address) for an address which is used as the operand of an instruction. Examples of this are:

SET *+0100
This pseudo instruction results in an area which is 256 bytes long being reserved by the assembler.

J *
This instruction jumps to itself, resulting in an indefinite loop.

4.7 Multiple operands should be separated by a comma without intervening spaces, i.e.:

```
CON     0F,0E,0D,0C,0B,0A,"151413121110"
```

5. Numbers are 16-bit unsigned integers. A symbol will be interpreted as a hex number if it begins with 0, or as a decimal number if it begins with another digit.

6. A comment may occupy a line by itself if preceded by the number symbol (#) in column 1.

7. 16-bit constants are stored in memory with the least significant byte first, although they are written most significant byte first in an assembly language program.

Assembler Pseudo Operations

Pseudo operations are statements which are recognized by the assembler as valid instructions, but which are not actually machine instructions. When encountered by the assembler, the results of the desired operation or the hexadecimal machine instructions necessary to obtain the desired results are incorporated into the program being assembled at each point where the pseudo op has been included. Pseudo ops are provided to save the programmer from tedious or error-prone tasks and, occasionally, to supply otherwise unavailable information to the assembled program.

DIRECTIVES

Directives are pseudo operations that direct the assembler to produce certain desired results, but which produce no hexadecimal machine instructions.

SET PROGRAM COUNTER – SET

Actual address or symbolic label

The SET PROGRAM COUNTER directive is a pseudo operation that causes the program counter to be set to the address specified by the operand. The address may be specified as an actual hexadecimal location or as the symbolic label if used It may refer only to a label that precedes the SET directive in the program.

The SET directive occupies no space in the assembled objective program and does not increase processing time. It has no effect upon the settings of the conditional flags.

Examples:

Mnemonic	Operand
SET	03E8

Sets program counter to hex location 03E8. Subsequent instructions are assembled at locations 03E9, 03EA, etc.

Mnemonic	Operand
SET	START

Program counter is set to address of previously programmed label, START.

END OF PROGRAM – END

No operand or actual address or symbolic label

The END OF PROGRAM directive signals the assembler that the previous instruction was the last instruction in the program. If no operand is provided, the assembler stops at the end of the assembly process. If an operand is provided, control is transferred to the location specified by the operand following the loading of the assembled program. The directive occupies no space at the end of the assembled object program. Instead, the address of the location, if specified, is stored in memory location 0001 and 0002 by the assembler.

Examples:

Mnemonic	Operand
END	none

Signals assembler that no more program instructions remain to be assembled.

END	START

Jumps to location of START following loading process.

END	03E8

Jumps to location 03E8 following loading process.

EQUATE — EQU
Actual address or symbolic label

The EQUATE directive assigns the address specified by the operand to the label appearing at the left of the EQU directive. The operand may be an actual address or a symbolic label associated elsewhere in the program with a location.

Examples:

Label	Mnemonic	Operand
SSNO	EQU	EMPID

If SSNO is assembled at location 03E8, the name, EMPID, may also be used to refer to location 03E8.

DATA	EQU	03E8

The location 03E8 may be referred to by the symbolic name, DATA.

OTHER PSEUDO OPERATIONS

The remaining pseudo operations result in the generation of program constants or assembled instructions. Since these pseudo operations accomplish more than merely directing the assembler, they are not referred to as directives.

DEFINE CONSTANT – CON

c1,c2,c3,c4,c5,. . .cn

The CON pseudo operation allows one or more bytes of constants to be defined each time a CON pseudo op is used. Each byte may have a value from 00 to FF or, in other words, may contain any possible combination of bit values. The CON pseudo op may be labeled, in which case the label refers to the location of the first constant byte.

The CON pseudo op itself occupies no space in the assembled object program. Each constant, however, occupies one byte. The operation does not increase program execution time after the program is loaded.

Examples:

Mnemonic	Operand
CON	3,0E8

Defines in the program two one-byte constants.

CON	04D,041, 052,059

Defines in the program four one-byte constants which could be used to print or display name "MARY."

CON	"MARY"

Same as above.

DEFINE ADDRESS CONSTANT – ADCON

Symbolic label or actual address

The DEFINE ADDRESS CONSTANT pseudo operation allows one or more two-byte addresses to be defined within the program. The operand of the ADCON may be specified as a symbolic label or, if known, as an actual address. If a symbolic label is the operand, the assembler determines the actual location of the label and inserts in the program the actual two-byte address as a high order byte of the address in the rightmost position. The addresses when so transposed may be used conveniently in the addresses of JUMP and CALL instructions if desired. (See descriptions of these instructions for the transposition that takes place during their assembly.)

An ADCON pseudo op occupies no space for itself, but address occupies two bytes of object program space. The operation does not increase program execution time after the program is loaded. The ADCON pseudo op may be labeled.

Examples:

Mnemonic	Operand
ADCON	TABLE

If TABLE has previously been assembled at location 03E8, this two-byte constant appears in the object program.

Mnemonic	Operand
ADCON	03E8,9

Two two-byte address constants appear in the object program.

DUPLICATE -- DUP n

The DUPLICATE pseudo operation duplicates the line of coding following it as many times as specified by n. The line following the DUP may be any kind of instruction, comment, or pseudo operation except another DUP or an END pseudo operation. The maximum value specified by n should not exceed FF. The DUP instruction can be labeled, in which case the label refers to the first byte in the duplicated series of bytes. A DUP instruction should not have as its operand a forward referencing expression.

The DUP expression itself occupies no space and requires no execution time in the assembled object program. Depending upon what type of instruction is duplicated, the assembled duplicated instructions will require additional time.

Examples:

Label	Mnemonic	Operand
BLANKS	DUP	5
	CON	020

The line following DUP is duplicated 5 times and can be referred to by the symbolic label BLANKS.

Mnemonic	Operand
DUP	3
SLC	

The SHIFT LEFT WITH CARRY instruction is duplicated three times in the assembled object program.

Machine Instructions

LOAD INSTRUCTIONS

LOAD REGISTER — LR,n1,n2

	C	Z	M	E
Length=1 byte	u	u	u	u

The LOAD REGISTER instruction moves data from the register specified by n2 to the register specified by n1. The instruction takes no operand. When the same register is specified as both source and destination (for example, LR,A,A) the instruction functions as NOP (no operation).

LOAD REGISTER FROM MEMORY (H,L) — L,n

	C	Z	M	E
Length=1 byte	u	u	u	u

Loads the register specified by n with the content of memory location addressed by the content of registers H and L. The H-register must contain the high order address byte and the L-register the low-order address byte (see paragraph 7 of "Assembly Language Coding Conventions").

Example:

Mnemonic	Operand
L,L	none

Loads L-register with byte from memory.

LOAD IMMEDIATE – LI,n

Literal or label

	C	Z	M	E
Length=2 bytes	u	u	u	u

The LOAD IMMEDIATE instruction loads the register specified by n with the literal value located in the second byte of this instruction. The operand may be expressed as a literal or as a label. (If a label is the operand, it is assembled to refer to the pseudo instruction LADR for a convenient means of loading the entire two-byte address.)

Example:

Mnemonic	Operand
LI,A	0

Loads the A-register with zero.

LOAD A-REGISTER FROM MEMORY – LDA

Symbolic label or actual address

	C	Z	M	E
Length=3 bytes	u	u	u	u

Loads the A-register with the content of the memory location addressed by byte 2 (the high order byte) and 3 (the low order byte) of the instruction.

Example:

Mnemonic	Operand
LDA	TABLE

Content of location TABLE is loaded into the A-register.

LOAD A-REGISTER FROM MEMORY (B,C) – LDAM,B

	C	Z	M	E
Length=1 byte	u	u	u	u

Loads the A-register with the contents of the memory location addressed by the contents of registers B and C. B-register must contain the high order byte and C-register the low order byte.

LOAD A-REGISTER FROM MEMORY (D,E) – LDAM,D

	C	Z	M	E
Length=1 byte	u	u	u	u

Loads the A-register with the contents of the memory location addressed by the content of registers D and E.

LOAD REGISTERS H,L WITH CONTENT OF ADDRESS – LHL
Symbolic label or actual address

	C	Z	M	E
Length=3 bytes	u	u	u	u

Loads the registers H and L with the contents of the memory location addressed by bytes 2 and 3 of the instruction.

Example:

Mnemonic	Operand
LHL	03E8

Contents of address 03E8 are stored in L, of address 03E9 in H.

LOAD REGISTERS WITH ADDRESS – LADR,n1,n2
Symbolic label or actual address

	C	Z	M	E
Length=4 bytes	u	u	u	u

The LOAD REGISTERS WITH ADDRESS pseudo operation loads two registers with the memory address provided or named by the operand. If the two registers to be loaded are not specified by the pseudo op, the assumption is made that H and L are the intended registers. If a symbolic label is the operand of the pseudo op, the assembler determines the memory location of the label and generates instructions of the address into the register specified at n1 (or the H-register if not specified) and the low-order portion of the address into the register specified at n2 (or the L-register if not specified). If an actual address appears in the operand of the pseudo op, the leftmost byte of the operand is loaded into n1 and the next byte is loaded into n2.

The assembler generates two instructions which are inserted in the program at each occurrence of a LADR pseudo instruction. The length of the LADR pseudo instruction is three or four bytes depending on the registers being loaded.

Examples:

Mnemonic	Operand
LADR	TABLE

The high-order byte of TABLE is loaded into the H-register and the low-order byte into the L-register.

LADR,H,A	TABLE

Same as above, except that the low-order byte is loaded into the A-register, possible for addition or subtraction before it is used as a memory reference.

LADR	03E8

H- and L-registers are loaded with 03 and E8 respectively.

LADR,SP	03E8

E8 is loaded into the lower order 8-bit of the stack pointer and 03 into the higher order 8-bit of the stack pointer.

STORE INSTRUCTIONS

STORE REGISTER – ST,n

	C	Z	M	E
Length=1 byte	u	u	u	u

The STORE REGISTER instruction stores the contents of the register specified by n in memory at the address contained in H and L. (The H- and L-registers must have been previously loaded with a 16-bit memory address.) The STORE REGISTER instruction takes no operand.

Examples:

Mnemonic	Operand
ST,A	none

The contents of the A-register are stored in memory at the memory location specified by registers H and L.

Mnemonic	Operand
ST,L	none

The contents of the A-register are stored in memory.

STORE IMMEDIATE – STI

Literal or label

Length=2 bytes

C	Z	M	E
u	u	u	u

The STORE IMMEDIATE instruction stores the literal value of the second byte of this instruction into memory at the address specified by registers H and L. (H and L must previously have been loaded with a 16-bit address.) The operand may be expressed as a hexadecimal value, an ASCII value, or a symbolic label. If the operand is expressed as a label, the assembler refers to the lower order byte of the two-byte address of the label.

Examples:

Mnemonic	Operand
STI	"$"

An ASCII dollar sign is stored in memory at address specified by H- and L-registers.

STI	060

Hexadecimal value 60 is stored in memory.

STI	PLACE

If PLACE is assembled at memory location 03E8, the low-order byte of the 16-bit address of PLACE is stored in memory.

STORE A-REGISTER IN MEMORY – STA

Symbolic label or actual address

Length=3 bytes

C	Z	M	E
u	u	u	u

Stores the contents of the A-register into the memory location addressed by bytes 2 and 3 of the instruction.

STORE A-REGISTER IN MEMORY (B,C) – STAM,B

	C Z M E
Length=1 byte	u u u u

Stores the contents of the A-register in the memory location addressed by the contents of registers B and C.

STORE A-REGISTER IN MEMORY (D,E) – STAM,D

	C Z M E
Length=1 byte	u u u u

Stores the contents of the A-register in the memory location addressed by the contents of registers D and E.

STORE REGISTERS H,L IN MEMORY – STHL

Symbolic label or actual address	C Z M E
Length=3 bytes	u u u u

Stores the contents of registers H and L into the memory location addressed by bytes 2 and 3 of the instruction.

Example:

Mnemonic	Operand
STHL	03E8

Contents of L are stored in 03E8 and contents of H in 03E9.

STACK INSTRUCTIONS

SAVE A,F IN STACK – PUSH,A

	C Z M E
Length=1 byte	u u u u

Save the contents of register A and of F (5 flags) into the push-down stack addressed by the stack pointer register SP. The content of SP is decremented by 2. (Content of A is stored in memory address SP–1, F in SP–2, and SP=SP–2.)

SAVE B,C IN STACK – PUSH,B
SAVE D,E IN STACK – PUSH,D
SAVE H,L IN STACK – PUSH,H

	C	Z	M	E
Length=1 byte	u	u	u	u

RESTORE A,F FROM STACK – POP,A

	C	Z	M	E
Length=1 byte	a	a	a	a

Restore the last values in the pushdown stack addressed by SP into A and F. The content of SP is incremented by two. (Content of memory address SP stored in F, SP+1 in A. SP=SP+2.)

RESTORE B,C FROM STACK – POP,B
RESTORE D,E FROM STACK – POP,D
RESTORE H,L FROM STACK – POP,H

	C	Z	M	E
Length=1 byte	u	u	u	u

EXCHANGE H,L WITH SP – SWAP

	C	Z	M	E
Length=1 byte	u	u	u	u

Exchange the contents of registers H and L and the last values in the pushdown stack addressed by register SP. (Content of address SP with L, SP+1 with H.)

TRANSFER H,L INTO SP – SPHL

	C	Z	M	E
Length=1 byte	u	u	u	u

Transfer the contents of registers H, L into register SP.

ARITHMETIC INSTRUCTIONS

ADD – A,n
A,M

	C	Z	M	E
Length=1 byte	a	a	a	a

The ADD instruction adds the value from the register specified by n to the accumulator. If the letter "M" is specified instead of a register, a byte from memory is added to the accumulator. The location of the byte in memory is determined by the memory address previously loaded into the H- and L-registers. All condition code flags are affected by this instruction. The flags reflect the status of the A-register at the end of the operation.

Examples:

Mnemonic	Operand
A,L	none

Contents of the register L added to A-register.

A,M	none

Contents of a memory byte, whose address is contained in H and L, added to A-register.

SUBTRACT – S,n
S,M

	C	Z	M	E
Length=1 byte	a	a	a	a

The SUBTRACT instruction subtracts a value in the register specified at n from a value in the accumulator (A-register). If the letter "M" is specified instead of a register, a byte from memory is subtracted from the accumulator. The location of the memory byte is determined by the address contained in registers H and L. All condition code flags are affected by this instruction. The flags reflect the status of the A-register at the end of the operation. If a borrow beyond the high-order bit has occurred, the carry flag is set to 1.

Examples:

Mnemonic	Operand
S,A	none

Subtracts contents of A-register from A-register (result=0).

Mnemonic	Operand
S,M	none

Subtracts contents of memory byte (whose location is specified by registers H and L) from A-register.

ADD IMMEDIATE — AI

	C Z M E
Literal or symbolic label	
Length=2 bytes	a a a a

The ADD IMMEDIATE instruction adds the contents of the second byte of this instruction to the value in the A-register. The operand of the instruction, which is assembled as a literal in the second byte of the instruction, may contain a one-byte literal value or may refer to a symbolic label. If a symbolic label is the operand, only the low-order byte of a two-byte address will be referenced.

The SUBTRACT IMMEDIATE instruction operates in the same manner except that the operand is subtracted from the A-register.

The instruction is two bytes long. All condition code flags are affected by the operation.

Examples:

Mnemonic	Operand
AI	07A

The value 7A is added to the A-register.

Mnemonic	Operand
SI	KON2

If the label KON2 is assembled at 03E8, the low-order byte of the two-byte address is subtracted from the A-register.

ADD WITH CARRY — AC,n
AC,M

	C Z M E
Length=1 byte	a a a a

SUBTRACT WITH BORROW – SB,n
SB,M

The ADD WITH CARRY instruction adds the value of the carry flag and the value of the specified register n to the A-register.

The SUBTRACT WITH BORROW instruction subtracts the value of the carry (borrow) flag and the value of the specified register n from the A-register. If an "M" is specified instead of a register, a memory byte and the carry flag are subtracted or added. The location of the memory byte is determined by the address contained in registers H and L.

All condition code flags are affected by these instructions.

Examples:

Mnemonic	Operand
AC,A	none

Value of carry flag and contents of A-register are added to the A-register (effect is identical to SCL instruction).

Mnemonic	Operand
AC,M	none

Value of carry flag and contents of memory byte at location specified by H and L are added to the A-register.

Mnemonic	Operand
SB,M	none

Value of carry flag and contents of memory byte whose location is specified by H and L are subtracted from A-register.

ADD WITH CARRY IMMEDIATE – ACI
Literal or symbolic label
Length=2 bytes

C	Z	M	E
a	a	a	a

SUBTRACT WITH BORROW IMMEDIATE – SBI
Literal or symbolic label

The ADD WITH CARRY IMMEDIATE adds the value of the carry flag and the value of the second byte of this instruction to the contents of the A-register. The operand of the instruction may be expressed as a literal or as a symbolic label. If the operand is a symbolic label, only the low-order byte of a two-byte address is referenced.

The SUBTRACT WITH BORROW IMMEDIATE subtracts the value of the borrow (carry) flag and the value of the second byte of the instruction from the contents of the A-register.

Examples:

Mnemonic	Operand
ACI	0BF

The value of the carry flag and the value BF are added to the A-register.

ACI	0

Only the value of the carry flag is added to the A-register, because the literal value also added is zero.

SBI	1

The value of the carry (borrow) flag and the literal value 1 are subtracted from the A-register.

SBI	TAG

The value of the carry (borrow) flag and the low-order byte of TAG's address (which could be assembled at 0BB8) are subtracted from the A-register.

ADD SP TO H,L — DAD,SP

	C	Z	M	E
Length=1 byte	a	u	u	u

Adds the content of register SP to the contents of registers H and L. If the overflow is generated, the carry flag is set; otherwise, the carry flag is reset. The other condition flags are not affected. This is useful for addressing data in the stack.

ADD B,C TO H,L — DAD,B
ADD D,E TO H,L — DAD,D
ADD H,L TO H,L — DAD,H

	C	Z	M	E
Length=1 byte	a	u	u	u

DECIMAL ADJUST ACCUMULATOR — DAA

Length=1 byte

C1	C	Z	M	E
a	a	a	a	a

The 8 bit value in the accumulator is adjusted to form two 4-bit binary coded decimal digits using two carry flags. The first carry flag, C1, checks the overflow from the 4th bit. The other is the usual carry flag, C. The following value is added to the A-register under the conditions of carry and previous contents of A-register for the decimal adjust.

Condition A:	A-register bits 3 to 0 10 and bits 7 to 4 = 9 or A-register bits 7 to 4 10 or C = 1
Condition B:	A-register bits 3 to 0 10 or C1 = 1

A	B	Add to A-register
false	false	00
false	true	06
true	false	60
true	true	66

INCREMENT — INC,n
DECREMENT — DEC,n

Length=1 byte

C	Z	M	E
u	a	a	a

The INCREMENT instruction adds the value 1 to the specified register n.

The DECREMENT instruction subtracts the value 1 from the specified register n.

Both of these instructions are one byte long. The carry flag is not affected by the INCREMENT or DECREMENT instructions. All other flags reflect the status of the specified register n at the end of the instruction execution.

Examples:

Mnemonic	Operand
INC,A	none

The contents of the A-register are incremented by 1.

Mnemonic	Operand
DEC,L	none

The contents of the L-register are decremented by 1.

INCREMENT MEMORY – INC,M
DECREMENT MEMORY – DEC,M

	C	Z	M	E
Length=1 byte	u	a	a	a

The contents of memory designated by registers H and L are incremented (decremented) by one. All of the condition flags except the carry flag are affected by the result.

INCREMENT DOUBLE B,C – IND,B

	C	Z	M	E
Length=1 byte	u	u	u	u

The content of register pair B and C is incremented by one.

Example:

Mnemonic	Operand
IND,B	none

Content of B-register = 01, C-register = E9 if the previous content of B-register was 01, C-register E8.

INCREMENT DOUBLE D,E – IND,D
INCREMENT DOUBLE H,L – IND,H
INCREMENT DOUBLE SP – IND,SP

	C	Z	M	E
Length=1 byte	u	u	u	u

DECREMENT DOUBLE B,C – DCD,B
DECREMENT DOUBLE D,E – DCD,D
DECREMENT DOUBLE H,L – DCD,H
DECREMENT DOUBLE SP – DCD,SP

SHIFT INSTRUCTIONS

SHIFT RIGHT – SR
SHIFT LEFT – SL

	C	Z	M	E
Length=1 byte	a	u	u	u

All shifts are circular.

The SHIFT RIGHT instruction shifts the contents of the A-register one bit to the right, placing the underflow bit into the vacated high-order position and copying it into the carry flag as well. The Zero, Minus, and Even Parity flags are unaffected by this instruction.

SHIFT RIGHT	C	7	6	5	4	3	2	1	0	Underflow
(before)	0	1	0	0	0	0	0	1	1	
			1	0	0	0	0	0	1	1
(after)	1	1	1	0	0	0	0	0	1	

The SHIFT LEFT instruction operates in the same manner except that the direction of the shift is to the left and the overflow bit is placed in the low-order position and copied into the carry flag as well.

SHIFT LEFT	Overflow	C	7	6	5	4	3	2	1	0
(before)		1	0	1	0	0	0	1	1	0
	0		1	0	0	0	1	1	0	
(after)		0	1	0	0	0	1	1	0	0

Examples:

Mnemonic	Operand
SR	none

Contents of the A-register shifted one bit right.

SL	none

Contents of the A-register shifted one bit left. In both examples, only the carry flag is affected.

SHIFT RIGHT WITH CARRY – SRC
SHIFT LEFT WITH CARRY – SLC

	C	Z	M	E
Length=1 byte	a	u	u	u

The SHIFT RIGHT WITH CARRY instruction links the A-register to the carry flag position as if they formed a 9-bit register and shifts the contents of the nine bits one bit to the right. The underflow bit from the low-order position is placed in the vacated carry flag position. The Zero, Minus, and Even Parity flags are unaffected by this instruction.

SHIFT RIGHT WITH CARRY

	C	7	6	5	4	3	2	1	0	Underflow
(before)	1	1	0	0	1	0	1	1	0	
		1	1	0	0	1	0	1	1	0
(after)	0	1	1	0	0	1	0	1	1	

The SHIFT LEFT WITH CARRY instruction operates in the same manner except that the direction of the shift is to the left. The overflow carry bit is placed in the low-order position.

UNCONDITIONAL BRANCH INSTRUCTIONS

JUMP – J

	C	Z	M	E
Symbolic label or actual address Length=3 bytes	u	u	u	u

The JUMP instruction discontinues the normal sequential flow of control from instruction to instruction and unconditionally jumps to the instruction whose label is specified in the operand of the instruction. The operand may express the location as an actual address or as a symbolic label of that address.

Note that the JUMP instruction is three bytes in length. The left-most byte contains the operation code of the instruction. The other two bytes contain the 16-bit address of the instruction to which control will be transferred. The assembler transposes the bytes containing the address so that the high-order portion (8 bits) are assembled in the rightmost byte and the low-order portion (8 bits) are assembled in the middle byte. The address contained in the operand is moved to the program counter (P-register) during execution of this instruction.

Examples:

Mnemonic	Operand
J	03E8

Jumps unconditionally to instruction assembled at program location 03E8. Note that high- and low-order portions of address are transposed by the assembler.

Mnemonic	Operand
J	END1

If END1 is assembled at memory location 0FA0, the JUMP instruction is assembled as shown.

CALL – CL

Symbolic label or actual address	C Z M E
Length=3 bytes	u u u u

Transfers contents of program counter to the pushdown stack in memory addressed by the register SP. High-order byte in SP–1 and low-order byte in SP–2. The content of SP is decremented by two (SP=SP–2). The address represented by byte 2 and byte 3 is transferred into the program counter. Jump unconditionally to the instruction located in memory location addressed by bytes 2 and 3 of the instruction.

Examples:

Mnemonic	Operand
CL	DIVIDE

If the subroutine named DIVIDE is located in memory at 0BB8, program control is transferred to that address after the return address is saved.

Mnemonic	Operand
CL	0BB8

Same as above example.

RETURN – R

	C Z M E
Length=1 byte	u u u u

Returns to the instruction in the memory location addressed by the last values shifted into the pushdown stack addressed by SP. (Content of SP and SP+1 are transferred into the program counter.) The content of SP is incremented by two.

CONDITIONAL BRANCH INSTRUCTIONS

Each conditional branch instruction tests the status of one of the four condition control flags during execution. The numbers and names of the flags and their settings are shown in the table below:

Flag Number:	00	01	10	11
Flag Name:	Carry	Zero	Minus	Even Parity
Flag Off Status:	0 NC	0 NZ	0 NM	0 NE
Flag On Status:	1 C	1 Z	1 M	1 E
Flag On Status Meaning:	Overflow past high-order bit of A-register	Register contains all zeros.	Register contains 1 in high-order bit	Sum of 1's in Register is an even number.

Flag Test	Conditional Jumps	Conditional Calls	Conditional Returns
Carry	JC	CLC	RC
Not Carry	JNC	CLNC	RNC
Zero	JZ	CLZ	RZ
Not Zero	JNZ	CLNZ	RNZ
Minus	JM	CLM	RM
Not Minus	JNM	CLNM	RNM
Even Parity	JE	CLE	RE
Not Even Parity	JNE	CLNE	RNE

CONDITIONAL JUMP INSTRUCTIONS

Length=3 bytes	C	Z	M	E
	u	u	u	u

The CONDITIONAL JUMP instruction tests the specified condition codes for the designated condition (on or off), and if the condition is found to be true, transfers control to the instruction address supplied by the operand of the conditional jump instruction. If the designated condition is found to be false, control continues in line to the next instruction following the conditional jump instruction.

The CONDITIONAL JUMP instruction, like the unconditional JUMP, is three bytes long, with the high- and low-order address bytes transposed by the assembler. Although the instruction tests a condition code flag, the flags are unaffected by the test.

CONDITIONAL CALL INSTRUCTIONS

Length=3 bytes	C	Z	M	E
	u	u	u	u

The CONDITIONAL CALL instruction tests the specified condition code for the designated condition and, if the condition is found to be true, transfers control to the subroutine whose label or address is contained in the operand of the instruction. The CONDITIONAL CALL instruction also saves the address of the next sequential instruction as a return address from the subroutine. The return address is saved in the stack as described in the section on the unconditional CALL instruction. If the specified condition is found to be false, program control continues in line to the next sequential instruction following the CONDITIONAL CALL. The instruction address is transposed by the assembler so that the high-order portion occupies the rightmost byte.

CONDITIONAL RETURN – RC

Length=1 byte

C	Z	M	E
u	u	u	u

The CONDITIONAL RETURN instruction tests the specified conditional code flag for the designated condition and, if the condition is found to be true, transfers control to the return address placed in the stack by the last CALL instruction. The CONDITIONAL RETURN instruction takes no operand.

LOGICAL INSTRUCTIONS

The following logical instructions compare the value of each bit position of the accumulator (A-register) to the value in the corresponding bit position of the designated register, memory byte, or instruction byte. The result of each type of logical operation for all possible combinations of bit values is as follows:

(Logical Product) AND	(Logical Sum) OR	(Logical Difference) Exclusive OR
1 0 1 0	1 0 1 0	1 0 1 0
1 1 0 0	1 1 0 0	1 1 0 0
1 0 0 0	1 1 1 0	1 1 0

Results in A-Register

None of the logical operations can generate a carry condition. Consequently, the carry condition code is always set to zero as a result of a logical operation. The zero, minus, and even parity condition code flags reflect the status of the A-register at the end of a logical operation.

AND — N,n
N,M

	C Z M E
Length=1 byte	0 a a a

The AND instruction "ands" the contents of the A-register with the contents of the register specified by n. If "M" is specified instead of a register, the instruction "ands" the contents of a memory byte with the A-register. The location of the memory byte is specified by the address in the H- and L-registers. The result of the AND operation, the logical product, is formed in the A-register.

The carry flag is set to zero as a result of this instruction. All other condition code flags—zero, minus, and even parity— reflect the status of the A-register at the end of the instruction execution.

Example:

Mnemonic	Operand
N,H	none

The contents of the H-register and "anded" with the contents of the A-register.

OR — O,n
O,M

	C Z M E
Length=1 byte	0 a a a

The OR instruction "ors" the contents of the A-register with the contents of the register specified by n. If "M" is specified instead of a register, the instruction "ors" the contents of a memory byte with the A-register. The location of the memory byte is specified by the address contained in the H-and L-registers. The result of the OR operation, the logical sum, is formed in the A-register.

The carry flag is set to zero as a result of this operation. All other condition code flags reflect the status of the A-register at the end of instruction execution.

Example:

Mnemonic	Operand
O,A	none

Contents of the A-register are "ored" with itself, resulting in no change in the A-register, but all condition flags except carry reflect status of A-register.

EXCLUSIVE OR – X,n
X,M

	C	Z	M	E
Length=1 byte	0	a	a	a

The EXCLUSIVE OR instruction "exclusively ors" the contents of the A-register with the contents of the register specified by n. If "M" is specified instead of a register, the instruction "exclusively ors" the contents of a memory byte with the A-register. The location of the memory byte is specified by the address contained in the H- and L-registers. The result of the EXCLUSIVE OR operation, the logical difference, is formed in the A-register.

The carry flag is set to zero as a result of this operation. All other condition code flags reflect the status of the A-register at the end of the instruction execution.

Example:

Mnemonic	Operand
X,A	none

Contents of the A-register are "exclusively ored" with itself. Result in A-register is zero.

AND IMMEDIATE – NI
OR IMMEDIATE – OI
EXCLUSIVE OR IMMEDIATE – XI

Literal or symbolic label	C Z M E
Length=2 bytes	0 a a a

The AND IMMEDIATE, OR IMMEDIATE and EXCLUSIVE OR IMMEDIATE instructions produce the same logical results as the AND, OR and EXCLUSIVE OR instructions, respectively. Instead of combining the contents of the A-register logically with another register or with a byte from memory, these instructions combine the A-register logically with the second byte of the instruction. The AND IMMEDIATE, OR IMMEDIATE and the EXCLUSIVE OR IMMEDIATE instructions are each two bytes long. The value in the second byte, the operand, may be expressed as a literal value or as a symbolic label. If reference is to a symbolic label, it is the low-order byte of the address of the label which is assembled in the second byte of the instruction.

The carry flag is set to zero following execution of the AND IMMEDIATE, OR IMMEDIATE or EXCLUSIVE OR IMMEDIATE instruction. The other flags reflect the status of the A-register following execution of the instruction.

Example:

Mnemonic	Operand
NI	0FF

The contents of the A-register are "anded" with the hex value FF.

COMPARE – C,n
C,M

	C Z M E
Length=1 byte	a a a a

The COMPARE instruction subtracts the value in the specified register n from the contents of the A-register without altering the contents of the A-register from its initial value. The results of the subtraction are not available. The function of the instruction is to set the condition code flags as if a subtract instruction had occurred, thereby comparing two registers without destroying the contents of either register. If an M is specified instead of a register, the A-register is compared with a memory byte whose location is designated by the address in registers H and L.

All condition code flags are affected by this instruction. At the end of the instruction execution, the flags reflect the status of the result that would have appeared in the A-register if a SUBTRACT instruction had been executed.

COMPARE IMMEDIATE — CI

	C	Z	M	E
Literal or symbolic label				
Length=2 bytes	a	a	a	a

The COMPARE IMMEDIATE instruction sets the condition code flags as if a subtraction of the second byte of the instruction from the contents of the A-register had occurred. The initial contents of the A-register are not altered by the instruction. The value in the second byte may be any literal value or may be expressed as a symbolic label. If a symbolic label is the operand, the low-order byte of the address of the label is compared with the A-register. The COMPARE IMMEDIATE instruction is two bytes long.

All condition code flags reflect the status of the A-register as if a SUBTRACT IMMEDIATE instruction had occurred.

Example:

Mnemonic	Operand
CI	0D

The hexadecimal value, 0D, is compared to (as if subtracted from) the A-register.

MISCELLANEOUS INSTRUCTIONS

STOP – STOP

	C	Z	M	E
Length=1 byte	u	u	u	u

The STOP instruction causes the program to halt. If an interrupt subroutine is executed, control is returned to the instruction following the STOP. An impassable STOP routine is shown in the example below.

Example:

	Mnemonic	Operand
LOOP	STOP	none
	J	LOOP

Interrupt system services interrupts (if any) and returns control to this instruction, which jumps back to STOP instruction (infinite loop).

NO OPERATION – NOP

	C	Z	M	E
Length=1 byte	u	u	u	u

The NOP instruction has no discernable effect except that the program counter is advanced to the next sequential instruction following the NOP and time elapses to perform the "no operation."

Example:

Mnemonic	Operand
NOP	none

No operation. The program continues to the next instruction following the NOP.

EXCHANGE H,L and D,E – XCH

	C	Z	M	E
Length=1 byte	u	u	u	u

Exchanges the contents of registers H,L and D,E.

H,L INTO PROGRAM COUNTER — PCHL

	C	Z	M	E
Length=1 byte	u	u	u	u

Transfers the contents of registers H,L into the program counter.

CMA

	C	Z	M	E
Length=1 byte	u	u	u	u

The content of the A-register is complemented.

STC

	C	Z	M	E
Length=1 byte	a	u	u	u

Sets the carry flag to 1.

CMC

	C	Z	M	E
Length=1 byte	a	u	u	u

The content of the carry flag is complemented.

MACROS

A macro call should name the macro and list the parameters, separating them by spaces or commas. Parameter zero is in the label field. Macro calls may be nested (one macro calling another) to a depth of four.

A macro definition begins with a MACRO pseudo op and ends with a MEND pseudo op. The macro definition may reference a parameter in parentheses nested to any depth. When the macro call is expanded, the parameter will be substituted for the parenthesized expression. A single character can be obtained by using two expressions within the parentheses: a parameter number and a character number (starting at 1). The JUMP pseudo op is followed by an expression and a macro label (begins with a period '.'). Assembly is transferred to the label if the expression is not zero.

Examples:

1. A macro to put the contents of two consecutive memory locations into registers A and B.

   The macro definition should appear before the first use of the macro, as follows:

```
MACRO
GET2
LADR        (1)
L,A
LADR        (1)+1
L,B
MEND
```

   Note the use of (1) to reference the first (and only) parameter of the macro. This parameter is used to specify the address of the two memory locations.

```
GET2        LABEL
```

   The above can now be used as shorthand for:

```
LADR        LABEL
L,A
LADR        LABEL+1
L,B
```

2. Suppose we wished to be able to specify the registers which are loaded in the GET2 macro. These can be given as parameters 2 and 3 if the macro definition is:

```
MACRO
GET2
LADR        (1)
L,(2)
LADR        (1)+1
L,(3)
MEND
```

Now the macro call

```
   GET2           STUFF,D,E
```

will be the equivalent of

```
   LADR           STUFF
   L,D
   LADR           STUFF+1
   L,E
```

3. The use of the JUMP pseudo op can be illustrated by creating a macro to shift right or left a specified number of places, with or without the carry bit.

```
                  MACRO
                  SHIFT
   COUNT          EQU
   AGAIN          S(1)(3)
   COUNT          EQU            COUNT–1
                  JUMP           COUNT+0',.AGAIN
                  MEND
```

Note the use of ' to complement the results of +.

```
   SHIFT          R,3,C
```

will generate

```
   SRC
   SRC
   SRC
```

* * * *

```
   SHIFT          L,4
```

will generate

```
   SL
   SL
   SL
   SL
```

INPUT/OUTPUT DEVICE INTERRUPTS

Interrupt capability is provided with the Q1 processor so that slower input/output devices may be serviced while the main program continues execution.

Automatic Interrupt Features

One level of interrupt is provided by system hardware. An input/output device sends an interrupt signal to the processor whenever it needs attention from the processor. Following completion of the program instruction in progress, the interrupt is acknowledged by the processor unless the following conditions exist:

1. Another interrupt is being serviced, in which case all interrupts are disabled until the current I/O interrupt is completed.
2. Interrupts are disabled.

If none of the above conditions exist, the processor will be interrupted. The processor automatically issues a call to location 00.

INTERRUPT INSTRUCTIONS

DISABLE INTERRUPT – DIS

	C	Z	M	E
Length=1 byte	u	u	u	u

The DISABLE INTERRUPT instruction prevents any interrupts from occurring.

ENABLE INTERRUPT – ENB

	C	Z	M	E
Length=1 byte	u	u	u	u

The ENABLE INTERRUPT instruction allows interrupts. The ENB instruction does not affect interrupt control until one instruction cycle has elapsed. This delay permits the execution of an instruction, such as a return from an interrupt subroutine to the main program, before another interrupt can occur.

Input/Output Devices

A maximum of thirty-two input/output device addresses may be connected to the input/output control unit (ICU) of the Q1 processor. (Some devices use more than one address.) The characteristics of the devices available from Q1 Corporation, and the status byte and control instructions applicable to each device, are described below.

Keyboard

Address 01

The keyboard is an input/output device. It is an electronic unit which produces a subtle audible click when a key is depressed. Each key position generates an eight-bit code with the exception of the four shift keys and a repeat key. The repeat key inputs the code of a previously depressed key at an approximate rate of ten characters per second.

MODE SHIFTING

Depressing the alpha key or the unshifted condition produces Mode 1. Keys depressed in Mode 1 will cause the keyboard to output codes corresponding to the lower case, or unshifted, characters depicted on the keytops.

Depressing the shift or shift lock keys produces Mode 2. This keyboard mode is indicated by the illumination of the shift lock keytop. Keys depressed in Mode 2 will cause the keyboard to output codes corresponding to the upper case, or shifted, characters depicted on the keytops.

Depressing the third numeric shift key produces Mode 3. This keyboard mode is indicated by the illumination of the third numeric keytop. Keys depressed in Mode 3 will cause the keyboard to output codes corresponding to the upper legends depicted on the keytops.

Depressing the fourth shift key produces Mode 4. This keyboard mode is indicated by the illumination of the fourth shift keytop. Keys depressed in Mode 4 will cause the keyboard to output codes corresponding to additional special characters. Depressing a shift or control shift key will reset the shift lock. All shift modes can be externally programmed; however, depressing shift keys will over-ride this external control.

IN,01

Loads the A-register with a data byte from the keyboard buffer, and clears the buffer to a hexadecimal 00.

OUT,01

Keyboard Control

In the A-register, the status bit assignments are:

7	6	5	4	3	2	1	0

Bit 0	– Click: keyboard produces an audible click
Bit 1	– Beep: keyboard produces an audible beep
Bit 2) Bit 3)	– In combination, control the shift keys and keyboard mode. See table below.
Bit 4	– Illuminates K1 keytop
Bit 5	– Illuminates K2 keytop
Bit 6	– Illuminates K3 keytop

Bit 3	Bit 2	Alpha	Shift	Third	Fourth	Mode
0	0	X	OFF	OFF	OFF	1
0	1	OFF	X	OFF	OFF	2
1	0	OFF	OFF	X	OFF	3
1	1	OFF	OFF	OFF	X	4
X	X	ON	OFF	OFF	OFF	1
X	X	OFF	ON	OFF	OFF	2
X	X	OFF	OFF	ON	OFF	3
X	X	OFF	OFF	OFF	ON	4

KEYBOARD INTERRUPTS
The keyboard attempts to interrupt the processor whenever a key is depressed.

Display

Address 03,04

The display is an output device. It is a fully buffered electronic unit, capable of displaying eight lines of data under program control. Each line contains a minimum of 37 character positions. Each character code is seven bits long. When a data byte is written from the processor to the display, the high-order bit is ignored. After one line has been completely written, the next character automatically appears in position 1 of the following line. However, when the last line is filled, the next character will automatically appear in position 1 of the first line.

To write a character on the display, address the device and output the first character. Since the display is refreshed by internal electronics asynchronous to processor operations, the display status must be tested prior to writing each subsequent character or reset command. Control instructions may be issued to reset the display to character position 1 of line 1, nondistructively blank or restore the display, or stip the display to the next character position.

OUT,03

Writes a character on the display

IN,03

Status

Display busy if bit 7 = 1.

OUT,04

Display Control

The control bit assignments for the A-register are:

7	6	5	4	3	2	1	0

Bit 0 – Reset: resets display to leftmost position of line 1

Bit 1 – Blank: display is blanked, but display buffer is not erased

Bit 2 – Unblank: buffer contents restored to display

Bit 3 – Step: character position is advanced one space to the right or to position 1 of the next line if prior position was at the end of a line

Printer

Address 05,06,07

The printer is a serial impact printer. Separate addresses are used for printing and paper or carriage motion as follows.

OUT,05

Print Character

IN,05

Status

The printer status byte has the following format:

7	6	5	4	3	2	1	0

Bit 5	– Out of ribbon if bit = 1
Bit 6	– Error occurred if bit = 1
Bit 7	– Printer busy if bit = 1

OUT,06

Moves the carriage in a direction determined by a previous OUT,07. The number of increments moved is normally determined by a 10 bit number. The low-order 8 bits are the contents of the A-register when OUT,06 is given. The highest two bits are the lowest two bits of the A-register when a previous OUT,07 is given. The increments are 1/48 of an inch in the vertical direction and 1/60 of an inch in the horizontal direction.

OUT,07

7	6	5	4	3	2	1	0

Bit 0) Bit 1)	– See OUT,06
Bit 2	– 0 = Forward Motion; 1 = Reverse Motion
Bit 3	– 0 = Carriage Motion; 1 = Paper Motion
Bit 4	– Lower ribbon
Bit 5	– Raise ribbon
Bit 6	– Provides expanded horizontal resolution of 1/120 of an inch
Bit 7	– Resets the printer and the interface electronics and moves the carriage to the extreme left position. Should be used before any other commands are given to the printer. It should also be used when an error condition occurs.

PRINTER INTERRUPTS

The printer is ready to accept data or control instructions when bit 7 of the status byte = 0. The printer interrupts as bit 7 becomes 0.

The Disk Unit

Address 19,1A,1B,1C

The Q1 disk has 77 tracks with from 8 to 511 bytes per record. The rotational speed of the disk is 360 r.p.m. It has a recording density of 3200 bpi and a data transfer rate of 250 kilobits per second.

IN,19

Read Data

This instruction transfers one parallel byte of data from the disk controller to the processor's accumulator. The controller normally supplies a byte of data every 32 microseconds. However, it is capable of stacking up to 3 bytes in a first-in-first-out (FIFO) holding register.

OUT,19

Write Data

This instruction transfers one parallel byte of data from the A-register to the disk controller. Each byte is loaded into the FIFO register. The controller transfers one byte of data from the FIFO to a one byte shift register every 32 micro-seconds.

IN,1A

Status

This instruction makes disk unit status information available for interrogation by the processor. The format of the status byte is:

7	6	5	4	3	2	1	0

Bit 0	—	Byte Ready: The FIFO is ready to either transfer a byte of data to the processor or accept a byte of data from the processor
Bit 1	—	File Unsafe: The selected disk has detected a controller violation or disk unit hardware failure during a write operation
Bit 2	—	AM Detect: The controller has read an address
Bit 4	—	Track 0: The selected drive's head is located at track 0
Bit 6	—	SD Ready: The selected disk is loaded properly and is up to speed
Bit 7	—	Busy: The controller is in the process of performing a write or step operation

OUT,1A

Disk Control 1

This instruction, in conjunction with the contents of the A-register, selects the appropriate disk within the system, or performs a step track or resets the disk system.

A-Register

7	6	5	4	3	2	1	0

Bit 0	—	Step track up
Bit 1	—	Step track down
Bit 2	—	Select disk drive 1 and load head
Bit 3	—	Select disk drive 2 and load head
Bit 4	—	Select disk drive 3 and load head
Bit 5	—	Select disk drive 4 and load head
Bit 6	—	Deselect all disk drives
Bit 7	—	Reset disk system

OUT,1B

Disk Control 2

This instruction, in conjunction with the contents of the A-register, initiates either a read, write sector or write track operation.

A-Register

7	6	5	4	3	2	1	0

Bit 0 – Start Read

Bit 6 – Start Write Track

Bit 7 – Start Write Sector

OUT,1C

Disk Control 3

If the clock pattern to be written is not OFF, this instruction transfers the clock data from the A-register to the disk controller. The clock data must be transferred to the controller before the data.

DISK INTERRUPT

The disk controller does not use the interrupt system. The controller is busy during a step or write operation.

Programmable Timer

Address 00

The programmable timer is an input/output unit. It may be used to issue an interrupt request after a prescribed delay. The programmable timer also puts a value of 1 into the second bit position of the status byte after each programmed interval. To enable the timer, it is necessary to set the timer from the A-register with an OUT instruction. An IN instruction will read its status.

OUT,00

Enable Timer

This instruction, in conjunction with the contents of the A-register, starts the timer. The timer will run for approximately 2.048 milliseconds times the decimal equivalent in the A-register.

IN,00

Status

This instruction reads the status of the programmable timer.

The time status byte has the following format:

7	6	5	4	3	2	1	0

Bit 0 — Restart Interrupt if bit = 1

Bit 1 — Timer running if bit = 0

PROGRAMMABLE TIMER INTERRUPT

The programmable time interval is over when bit 1 of the status byte becomes 1. The time interrupts as bit 1 becomes 1.

Communications Interface Unit
Address 010,011,012,013

The Communications Interface Unit (CIU) contains all the hardware required to communicate asynchronously or synchronously through standard modems such as 103, 201, 94 202 types. Interface signals are in accordance with EIA specification RS–232C. Teletype current loop signals are also available. The CIU also has the capability to control an 801C2 type Automatic Calling Equipment (ACE) as defined by EIA specification RS–366. The CIU operates in simplex, half or full-duplex mode at programmably selected data rates. The CIU is expandable to control up to eight mixed speed communication channels.

ADR	IN	OUT
010	DATA	DATA
011	STATUS 1	CONTROL I
012	STATUS 2	CONTROL 2
013		CONTROL 3

OUT 011
CONTROL 1

A-Register

7	6	5	4	3	2	1	0

Bit 0	– Number Bit 1	
Bit 1	– Number Bit 2	
Bit 2	– Number Bit 3	To ACE
Bit 3	– Number Bit 8	
Bit 4	– Digit Present	
Bit 5	– Channel Select Control	
Bit 6	– Receiver Control 3	
Bit 7	– Transmitter Control 3	

IN 011
STATUS 1

A-Register
7 6 5 4 3 2 1 0

Bit 0	–	Power on	
Bit 1	–	Call originate status	
Bit 2	–	Data line occupied	From ACE
Bit 3	–	Abandon call and retry	
Bit 4	–	Present next digit	
Bit 5	–	Spare	
Bit 6	–	Spare	
Bit 7	–	Ring Indicator	From RS–232

OUT 012
CONTROL 2

A-Register
7 6 5 4 3 2 1 0

Bit 0	–	Master reset
Bit 1	–	DAta terminal ready
Bit 2	–	Request to send
Bit 3	–	Secondary transmitter on
Bit 4	–	Set error flag
Bit 5	–	Call requested
Bit 6	–	Stop elements: 0 = one element, 1 = more than one element (async mode only)
Bit 7	–	Synchronous mode

IN 012
STATUS 2

A-Register
7 6 5 4 3 2 1 0

Bit 0	–	Secondary carrier detected	
Bit 1	–	Data set ready	
Bit 2	–	Clear to send	from RS–232
Bit 3	–	Carrier detect	
Bit 4	–	Framing or overrun error/ data not available	
Bit 5	–	Sync detect	
Bit 6	–	Receiver ready	
Bit 7	–	Transmitter ready	

OUT 013
CONTROL 3

A-Register
7 6 5 4 3 2 1 0

Bit 0	–	Receiver Channel Address 0	/Rate Select 0
Bit 1	–	Receiver Channel Address 1	/Rate Select 1
Bit 2	–	Receiver Channel Address 2	/Rate Select 2
Bit 3	–	Spare	/Rate Select 3
Bit 4	–	Transmitter State Address 0	/Even Parity
Bit 5	–	Transmitter State Address 1	/Inhibit Parity
Bit 6	–	Transmitter State Address 2	/Word Length 0
Bit 7	–	Spare	/Word Length 1

Rate Select Bits				
3	2	1	0	Baud Rate
0	0	0	0	50
0	0	0	1	75
0	0	1	0	110
0	0	1	1	134.5
0	1	0	0	150
0	1	0	1	300
0	1	1	0	600
0	1	1	1	1200
1	0	0	0	1800
1	0	0	1	2000
1	0	1	0	2400
1	0	1	1	3600
1	1	0	0	4800
1	1	0	1	7200
1	1	1	0	9600
1	1	1	1	19,200

Word Length Bits		Code
7	6	Level
0	0	5
0	1	6.
1	0	7
1	1	8

INTERRUPTS

• Any of the following conditions causes an interrupt:

1. Ring indicator
2. Any change on the Data Set Ready of Clear to Send status lines or secondary received data line
3. Sync detect
4. Receiver ready
5. Transmitter ready

CHANNEL SELECTION

To select a communications channel execute an OUT 011 with 20 in the A-register then load the A-register with the appropriate receiver and transmitter channel addresses and execute an OUT 013.

RECEIVER BAUD RATE AND MODE CONTROL

To program the receiver, execute an OUT 011 with 40 in the A-register. Set A-register bits 6 and 7 for desired mode and execute an OUT 012.

Bit 7	Bit 6	Mode
0	0	Asynchronous mode with 1 stop element
0	1	Asynchronous mode with 1.5 stop elements for 5 level code or 2 stops for 6, 7, or 8 level codes
1	0	Synchronous mode
1	1	Forbidden combination

Load the A-register with the appropriate Baud rate, parity, and word length codes and execute an OUT 013.

TRANSMITTER BAUD RATE AND MODE CONTROL

To program the transmitter, execute an OUT 011 with 80 in the A-register. If the transmitter characteristics are to differ from the receiver's, execute OUT 012 and OUT 013 as described in the receiver programming description; otherwise skip the OUT 012 and reload the A-register with the Baud rate, parity, and word length codes and execute an OUT 013.

ORIGINATING A CALL

Prior to originating a call, the channel must be selected, receiver and transmitter programmed, and the ACE turned on. ACE status is tested by executing an IN 011. Bit 0, ACE power on, should be the only bit set. To originate a call execute an OUT 012 with 11 in the A-register. This turns on Data Terminal Ready and Call Request. Call Request must remain set to a one until the call is terminated. Test ACE status until 15 is read, ACE power on, data line occupied and present next digit. At this time, load the A-register bits 0 through 3, number bits, with the BCD equivalent of the first digit to be dialed and bit 4, digit present, set to a one.

Execute an OUT 011 for the first digit to be dialed. ACE status bit 4, present next digit, will go to a zero. When present next digit returns to a one, put up the BCD equivalent of the next digit to be dialed and digit present, then execute an OUT 011.

Repeat this sequence until the complete number has been dialed. At this time, the ACE status will read 17 indicating that the call originate status is set to a one. The data set is now in the data mode and the ACE has relinquished control of the communications channel to the data set. This status is verified by testing bit 1, data set ready, of status byte 2 for a one condition.

If the call is not completed, ACE status bit 3, Abandon call and retry, will become set to a one. In this instance, terminate the call and repeat the call originate procedure.

ANSWERING A CALL

An incoming call is detected by bit 7, ring indicator, of status byte 1 being set to a one. In response to this status, execute an OUT 012 with A-register bit 1, data terminal ready, set to a one.

TERMINATING A CALL

To terminate a call, execute an OUT 012 with 00 in the A-register.

Interface Operation

RECEIVE MODE

When the distant data set is about to send, bit 3, carrier detect, of status byte 2 will be set to a one. Received data will be assembled in a buffer. When a byte is ready to be transferred from this buffer to the processor, bit 6, receiver ready, of status byte 2 will be set to a one. An IN 010 response will transfer the received data byte to the A-register and reset the receiver ready bit to zero. If the above IN 010 is not executed by the time the next received byte is assembled in the buffer, the previous byte will be lost and the receiver will set bit 4, overrun error, of status byte 2. If, in the asynchronous mode, no valid stop bit has been detected when a received byte is assembled in the buffer, the receiver will set bit 4, framing error, of status byte 2. To clear the overrun and/or framing error status, execute an OUT 012 with bit 4, reset error flag, in the A-register set to a one. Each time a received data byte, encoded 23X, is assembled in the buffer, bit 5, sync detect, of status byte 2 will be set to a one.

In the synchronous mode, character synchronization is performed by the receiver. Sync characters to be transferred to the processor will be indicated by bit 4 of status byte 2 for stripping purposes.

TRANSMIT MODE

In order to transmit, bit 3, carrier detect, of status byte 2 must be tested for a zero condition to insure that the distant data set is in the receive mode. To enable transmission from the local data set, execute an OUT 012 with bit 2, request to send, set in the A-register. The first character to be transmitted may be loaded into the transmitter buffer by executing an OUT 010. When the data set is ready to transmit, bit 2, clear to send, of status byte 2 will be set to a one. At this time, the transmitter will send the byte previously loaded. If data had not been loaded into the transmitter, the transmitter will send a rub out character if in the asynchronous mode or a sync character if in the synchronous mode. When the transmitter is ready to accept a byte from the processor, bit 7, transmitter ready, of status byte 2 will be set to a one. If a byte of data is not sent to the transmitter in time, a fill character will be sent. This will be indicated by bit 4, data not available, of status byte 2 set to a one. To clear the data not available status, execute an OUT 012 with bit 4, reset error flag, in the A-register set to a one. The fill character will be a rub out when in the asynchronous mode and a sync character in the synchronous mode.

Instruction Set

(Alphabetic Order)

Code	Mnem	Length	Time
87	A,A	1	2
80	A,B	1	2
81	A,C	1	2
82	A,D	1	2
83	A,E	1	2
84	A,H	1	2
85	A,L	1	2
86	A,M	1	3.5
8F	AC,A	1	2
88	AC,B	1	2
89	AC,C	1	2
8A	AC,D	1	2
8B	AC,E	1	2
8C	AC,H	1	2
8D	AC,L	1	2
8E	AC,M	1	3.5
CE	ACI	1	3.5
C6	AI	2	3.5
BF	C,A	1	2
B8	C,B	1	2
B9	C,C	1	2
BA	C,D	1	2
BB	C,E	1	2
BC	C,H	1	2
BD	C,L	1	2
BE	C,M	1	3.5
FE	CI	2	3.5
CD	CL	3	8.5
DC	CLC	3	5.5/9
ED	CLE	3	5.5/9
FC	CLM	3	5.5/9
D4	CLNC	3	5.5/9

Code	Mnem	Length	Time
E4	CLNE	3	5.5/9
F4	CLNM	3	5.5/9
C4	CLNZ	3	5.5/9
CC	CLZ	3	5.5/9
2F	CMA	1	2
3F	CMC	1	2
27	DAA	1	2
09	DAD,B	1	5
19	DAD,D	1	5
29	DAD,H	1	5
39	DAD,SP	1	5
0B	DCD,B	1	2.5
1B	DCD,D	1	2.5
2B	DCD,H	1	2.5
3B	DCD,SP	1	2.5
3D	DEC,A	1	2.5
05	DEC,B	1	2.5
0D	DEC,C	1	2.5
15	DEC,D	1	2.5
1D	DEC,E	1	2.5
25	DEC,H	1	2.5
2D	DEC,L	1	2.5
35	DEC,M	1	5
F3	DIS	1	2
FB	ENB	1	2
DB	IN	2	5
3C	INC,A	1	2.5
04	INC,B	1	2.5
0C	INC,C	1	2.5
14	INC,D	1	2.5
1C	INC,E	1	2.5
24	INC,H	1	2.5

Note: Timing is in microseconds.

Code	Mnem	Length	Time
2C	INC,L	1	2.5
34	INC,M	1	5
03	IND,B	1	2.5
13	IND,D	1	2.5
23	IND,H	1	2.5
33	IND,SP	1	2.5
C3	J	3	5
DA	JC	3	5
EA	JE	3	5
FA	JM	3	5
D2	JNC	3	5
E2	JNE	3	5
F2	JNM	3	5
C2	JNZ	3	5
CA	JZ	3	5
7E	L,A	1	3.5
46	L,B	1	3.5
4E	L,C	1	3.5
56	L,D	1	3.5
5E	L,E	1	3.5
66	L,H	1	3.5
6E	L,L	1	3.5
01	LADR,B,C	3	8
11	LADR,D,E	3	8
21	LADR,H,L	3	8
31	*LADR,SP	3	8
3A	LDA	3	6.5
0A	LDAM,B	1	3.5
1A	LDAM,D	1	3.5
2A	LHL	3	8.5
3E	LI,A	2	3.5
06	LI,B	2	3.5

Code	Mnem	Length	Time
0E	LI,C	2	3.5
16	LI,D	2	3.5
1E	LI,E	2	3.5
26	LI,H	2	3.5
2E	LI,L	2	3.5
7F	LR,A,A	1	2.5
78	LR,A,B	1	2.5
79	LR,A,C	1	2.5
7A	LR,A,D	1	2.5
7B	LR,A,E	1	2.5
7C	LR,A,H	1	2.5
7D	LR,A,L	1	2.5
47	LR,B,A	1	2.5
40	LR,B,B	1	2.5
41	LR,B,C	1	2.5
42	LR,B,D	1	2.5
43	LR,B,E	1	2.5
44	LR,B,H	1	2.5
45	LR,B,L	1	2.5
4F	LR,C,A	1	2.5
48	LR,C,B	1	2.5
49	LR,C,C	1	2.5
4A	LR,C,D	1	2.5
4B	LR,C,E	1	2.5
4C	LR,C,H	1	2.5
4D	LR,C,L	1	2.5
57	LR,D,A	1	2.5
50	LR,D,B	1	2.5
51	LR,D,C	1	2.5
52	LR,D,D	1	2.5
53	LR,D,E	1	2.5
54	LR,D,H	1	2.5

*All other LADR instructions are the combination of two LI instructions.

Code	Mnem	Length	Time
55	LR,D,L	1	2.5
5F	LR,E,A	1	2.5
58	LR,E,B	1	2.5
59	LR,E,C	1	2.5
5A	LR,E,D	1	2.5
5B	LR,E,E	1	2.5
5C	LR,E,H	1	2.5
5D	LR,E,L	1	2.5
67	LR,H,A	1	2.5
60	LR,H,B	1	2.5
61	LR,H,C	1	2.5
62	LR,H,D	1	2.5
63	LR,H,E	1	2.5
64	LR,H,H	1	2.5
65	LR,H,L	1	2.5
6F	LR,L,A	1	2.5
68	LR,L,B	1	2.5
69	LR,L,C	1	2.5
6A	LR,L,D	1	2.5
6B	LR,L,E	1	2.5
6C	LR,L,H	1	2.5
6D	LR,L,L	1	2.5
A7	N,A	1	2
A0	N,B	1	2
A1	N,C	1	2
A2	N,D	1	2
A3	N,E	1	2
A4	N,H	1	2
A5	N,L	1	2
A6	N,M	1	3.5
E6	NI	2	3.5
00	NOP	1	2

Code	Mnem	Length	Time
B7	O,A	1	2
B0	O,B	1	2
B1	O,C	1	2
B2	O,D	1	2
B3	O,E	1	2
B4	O,H	1	2
B5	O,L	1	2
B6	O,M	1	3.5
F6	OI	2	3.5
D3	OUT	2	5
E9	PCHL	1	2.5
F1	POP,A	1	5
C1	POP,B	1	5
D1	POP,D	1	5
E1	POP,H	1	5
F5	PUSH,A	1	5.5
C5	PUSH,B	1	5.5
D5	PUSH,D	1	5.5
E5	PUSH,H	1	5.5
C9	R	1	5
D8	RC	3	2.5/5.5
E8	RE	3	2.5/5.5
F8	RM	3	2.5/5.5
D0	RNC	3	2.5/5.5
E0	RNE	3	2.5/5.5
F0	RNM	3	2.5/5.5
C0	RNZ	3	2.5/5.5
C8	RZ	3	2.5/5.5
97	S,A	1	2
90	S,B	1	2
91	S,C	1	2
92	S,D	1	2

Code	Mnem	Length	Time
93	S,E	1	2
94	S,H	1	2
95	S,L	1	2
96	S,M	1	3.5
9F	SB,A	1	2
98	SB,B	1	2
99	SB,C	1	2
9A	SB,D	1	2
9B	SB,E	1	2
9C	SB,H	1	2
9D	SB,L	1	2
9E	SB,M	1	3.5
DE	SBI	2	3.5
D6	SI	2	3.5
17	SL	1	2
07	SLC	1	2
F9	SPHL	1	2.5
1F	SR	1	2
0F	SRC	1	2
77	ST,A	1	3.5
70	ST,B	1	3.5
71	ST,C	1	3.5
72	ST,D	1	3.5
73	ST,E	1	3.5
74	ST,H	1	3.5
75	ST,L	1	3.5
32	STA	3	6.5
02	STAM,B	1	3.5
12	STAM,D	1	3.5
37	STC	1	2
22	STHL	3	8.5
36	STI	2	5

Code	Mnem	Length	Time
76	STOP	1	0.5
E3	SWAP	1	9
AF	X,A	1	2
A8	X,B	1	2
A9	X,C	1	2
AA	X,D	1	2
AB	X,E	1	2
AC	X,H	1	2
AD	X,L	1	2
AE	X,M	1	3.5
EB	XCH	1	2
EE	XI	2	3.5

(Numeric Order)

Code	Mnem	Length	Time
00	NOP	1	2
01	LADR,B,C	3	8
02	STAM,B	1	3.5
03	IND,B	1	2.5
04	INC,B	1	2.5
05	DEC,B	1	2.5
06	LI,B	2	3.5
07	SLC	1	2
09	DAD,B	1	5
0A	LDAM,B	1	3.5
0B	DCD,B	1	2.5
0C	INC,C	1	2.5
0D	DEC,C	1	2.5
0E	LI,C	2	3.5
0F	SRC	1	2
11	LADR,D,E	3	8
12	STAM,D	1	3.5
13	IND,D	1	2.5
14	INC,D	1	2.5
15	DEC,D	1	2.5
16	LI,D	2	3.5
17	SL	1	2
19	DAD,D	1	5
1A	LDAM,D	1	3.5
1B	DCD,D	1	2.5
1C	INC,E	1	2.5
1D	DEC,E	1	2.5
1E	LI,E	2	3.5
1F	SR	1	2
21	LADR,H,L	3	8
22	STHL	3	8.5
23	IND,H	1	2.5

Code	Mnem	Length	Time
24	INC,H	1	2.5
25	DEC,H	1	2.5
26	LI,H	2	3.5
27	DAA	1	2
29	DAD,H	1	5
2A	LHL	3	8.5
2B	DCD,H	1	2.5
2C	INC,L	1	2.5
2D	DEC,L	1	2.5
2E	LI,L	2	3.5
2F	CMA	1	2
31	LADR,SP	3	8
32	STA	3	6.5
33	IND,SP	1	2.5
34	INC,M	1	5
35	DEC,M	1	5
36	STI	2	5
37	STC	1	2
39	DAD,SP	1	5
3A	LDA	3	6.5
3B	DCD,SP	1	2.5
3C	INC,A	1	2.5
3D	DEC,A	1	2.5
3E	LI,A	2	3.5
3F	CMC	1	2
40	LR,B,B	1	2.5
41	LR,B,C	1	2.5
42	LR,B,D	1	2.5
43	LR,B,E	1	2.5
44	LR,B,H	1	2.5
45	LR,B,L	1	2.5
46	L,B	1	3.5

Code	Mnem	Length	Time
47	LR,B,A	1	2.5
48	LR,C,B	1	2.5
49	LR,C,C	1	2.5
4A	LR,C,D	1	2.5
4B	LR,C,E	1	2.5
4C	LR,C,H	1	2.5
4D	LR,C,L	1	2.5
4E	L,C	1	3.5
4F	LR,C,A	1	2.5
50	LR,D,B	1	2.5
51	LR,D,C	1	2.5
52	LR,D,D	1	2.5
53	LR,D,C	1	2.5
54	LR,D,H	1	2.5
55	LR,D,L	1	2.5
56	L,D	1	3.5
57	LR,D,A	1	2.5
58	LR,E,B	1	2.5
59	LR,E,C	1	2.5
5A	LR,E,D	1	2.5
5B	LR,E,E	1	2.5
5C	LR,E,H	1	2.5
5D	LR,E,L	1	2.5
5E	L,E	1	3.5
5F	LR,E,A	1	2.5
60	LR,H,B	1	2.5
61	LR,H,C	1	2.5
62	LR,H,D	1	2.5
63	LR,H,E	1	2.5
64	LR,H,H	1	2.5
65	LR,H,L	1	2.5
66	L,H	1	3.5
67	LR,H,A	1	2.5
68	LR,L,B	1	2.5
69	LR,L,C	1	2.5
6A	LR,L,D	1	2.5
6B	LR,L,E	1	2.5
6C	LR,L,H	1	2.5
6D	LR,L,L	1	2.5
6E	L,L	1	3.5
6F	LR,L,A	1	2.5
70	ST,B	1	3.5
71	ST,C	1	3.5
72	ST,D	1	3.5
73	ST,E	1	3.5
74	ST,H	1	3.5
75	ST,L	1	3.5
76	STOP	1	0.5
77	ST,A	1	3.5
78	LR,A,B	1	2.5
79	LR,A,C	1	2.5
7A	LR,A,D	1	2.5
7B	LR,A,E	1	2.5
7C	LR,A,H	1	2.5
7D	LR,A,L	1	2.5
7E	L,A	1	3.5
7F	LR,A,A	1	2.5
80	A,B	1	2
81	A,C	1	2
82	A,D	1	2
83	A,E	1	2
84	A,H	1	2
85	A,L	1	2
86	A,M	1	3.5

Code	Mnem	Length	Time
87	A,A	1	2
88	AC,B	1	2
89	AC,C	1	2
8A	AC,D	1	2
8B	AC,E	1	2
8C	AC,H	1	2
8D	AC,L	1	2
8E	AC,M	1	3.5
8F	AC,A	1	2
90	S,B	1	2
91	S,C	1	2
92	S,D	1	2
93	S,E	1	2
94	S,H	1	2
95	S,L	1	2
96	S,M	1	3.5
97	S,A	1	2
98	SB,B	1	2
99	SB,C	1	2
9A	SB,D	1	2
9B	SB,E	1	2
9C	SB,H	1	2
9D	SB,L	1	2
9E	SB,M	1	3.5
9F	SB,A	1	2
A0	N,B	1	2
A1	N,C	1	2
A2	N,D	1	2
A3	N,E	1	2
A4	N,H	1	2
A5	N,L	1	2
A6	N,M	1	3.5
A7	N,A	1	2
A8	X,B	1	2
A9	X,C	1	2
AA	X,D	1	2
AB	X,E	1	2
AC	X,H	1	2
AD	X,L	1	2
AE	X,M	1	3.5
AF	X,A	1	2
B0	O,B	1	2
B1	O,C	1	2
B2	O,D	1	2
B3	O,E	1	2
B4	O,H	1	2
B5	O,C	1	2
B6	O,M	1	3.5
B7	O,A	1	2
B8	C,B	1	2
B9	C,C	1	2
BA	C,D	1	2
BB	C,E	1	2
BC	C,H	1	2
BD	C,L	1	2
BE	C,M	1	3.5
BF	C,A	1	2
C0	RNZ	3	2.5/5.5
C1	POP,B	1	5
C2	JNZ	3	5
C3	J	3	5
C4	CLNZ	3	5.5/9
C5	PUSH,B	1	5.5
C6	AI	2	3.5

Code	Mnem	Length	Time
C8	RZ	3	2.5/5.5
C9	R	1	5
CA	JZ	3	5
CC	CLZ	3	5.5/9
CD	CL	3	8.5
CE	ACI	1	3.5
D0	RNC	3	2.5/5.5
D1	POP,D	1	5
D2	JNC	3	5
D3	OUT	2	5
D4	CLNC	3	5.5/9
D5	PUSH,D	1	5.5
D6	SI	2	3.5
D8	RC	3	2.5/5.5
DA	JC	3	5
DB	IN	2	5
DC	CLC	3	5.5/9
DE	SBI	2	3.5
E0	RNE	3	2.5/5.5
E1	POP,H	1	5
E2	JNE	3	5
E3	SWAP	1	9
E4	CLNE	3	5.5/9
E5	PUSH,H	1	5.5
E6	NI	2	3.5
E8	RE	3	2.5/5.5
E9	PCHL	1	2.5
EA	JE	3	5
EB	XCH	1	2
ED	CLE	3	5.5/9
EE	XI	2	3.5
F0	RNM	3	2.5/5.5

Code	Mnem	Length	Time
F1	POP,A	1	5
F2	JNM	3	5
F3	DIS	1	2
F4	CLNM	3	5.5/9
F5	PUSH,A	1	5.5
F6	OI	2	3.5
F8	RM	3	2.5/5.5
F9	SPHL	1	2.5
FA	JM	3	5
FB	ENB	1	2
FC	CLM	3	5.5/9
FE	CI	2	3.5