MICRO PROCESSORS & MICROCONTROLLERS LAB (PC268EE)

LABORATORY MANUAL

VI SEM B.E.(EEE/EIE) AICTE-MC

DEPARTMENT OF ELECTRICAL ENGINEERING

MUFFAKHAM JAH COLLEGE OF ENGINEERING & TECHNOLOGY
Banjara Hills Road No 3, Hyderabad 34
www.mjcollege.ac.in

2020-21

Prepared By: G. RAVI KIRAN, Asst. Professor

MICROPROCESSORS & MICROCONTROLLERS LAB
(EEE & EIE)
LIST OF EXPERIMENTS

Using GNU 8085 Simulator

Demo: (A) Addition of two 8 Bit Numbers.
(B) Subtraction of two 8 Bit Numbers.

1. (a) Programs for Signed/Unsigned Multiplication.
   (b) Program for Signed/Unsigned Division.

2. (a) Program to find Average of 8 Bit Numbers in an Array.
   (b) Program for finding the largest/smallest number in an Array.
   (c) Program for finding the square root a given number.

3. Program for sorting the set of numbers.
   (a) Program for arranging the numbers in ascending order.
   (b) Program for arranging the numbers in descending order.

4. Programs for code conversion like BCD numbers to seven segment.

USING 8085 KIT

5. 8255 – PPI: ALP to generate Triangular wave using DAC

   (a) Program to generate Saw tooth wave form.
   (b) Program to generate Triangular wave form.
   (c) Program to generate Square wave form.

USING 8051 KIT

6. Arithmetic Instructions: Multibyte Operations
   (a) Program for addition/subtraction of two 16 bit numbers.
   (b) Program for multiplication/division of two 16 bit/32 bit numbers.

7. Data Transfer – block move, exchange, sorting, finding largest number in an array.
   (a) Program for exchange of data.
   (b) Program for sorting the set of numbers.
   (c) Program for finding maximum/minimum number in an array.

8. Boolean & Logical Instructions (Bit Manipulations)
   (a) Program for reverse & logical 'OR' of a given number.

9. Program for use of "JUMP" & "CAL" instructions.

USING (KEIL Software) for 8051

Demo: (a) Program to find addition of two numbers.
(b) Program of Multibyte Addition

10. Program for activating ports and generation of square wave.

11. (a) Program for ascending order/descending order of a given numbers
(b) Program for data transfer.

Course Code	Course Title						Core/Elective
PC268EE	**Microprocessor and Microcontrollers Lab**						**Core**
Prerequisite	Contact Hours per Week				CIE	EE	Credits
	L	T		P			
-	-	-		**2**	**25**	**50**	**1**

Course Objectives

- Developing of assembly level programs and providing the basics of the processors
- To provide solid foundation on interfacing the external devices to the processor according to the user requirements to create novel products and solutions for the real time problems.
- To assist the students with an academic environment needed for a successful professional career.

Course Outcomes

At the end of the course students will be able to

- Familiarize with the assembly language programming.
- Write programs for given task using different addressing modes.
- Interface various IO devices using 8255 PPI
- Write programs using various interrupts.
- Interface the microcontroller for some real life applications.

List of Experiments:

8085 based:

1. Signed/unsigned multiplication and division.
2. Finding average, largest, square root, etc.
3. Sorting set of numbers.
4. Code conversion like BCD numbers into binary.
5. 8255 PPI for interfacing LEDs.
6. 8255 PPI for interfacing to generate triangular wave using DAC.
7. Using interrupts.
8. Interfacing seven segment display.
9. Interfacing matrix keyboard.

8051 based:

1. Data transfer block move, exchange, sorting, finding largest element in array.
2. Arithmetic instructions: multi byte operations.
3. Boolean & logical instructions (Bit manipulations).
4. Programs to generate delay, programs using serial port and on chip timer/counter.
5. Use of JUMP and CALL instructions.
6. Square wave generation using timers.
7. Interfacing of keyboard and 7-segment display module.
8. DAC interfacing for generation of sinusoidal wave.

Note: At least five experiments for 8085 and at least five experiments for 8051.

INTRODUCTION TO MASM

GNUSim8085 is a graphical simulator, assembler and debugger for the Intel 8085 microprocessor in Linux and Windows. It is among the 20 winners of the FOSS India Awards announced on February, 2008. GNUSim8085 was originally written by Sridhar Ratna kumar in fall 2003 when he realized that no proper simulators existed for Linux. Several patches, bug fixes and software packaging have been contributed by the GNUSim8085 community. GNUSim8085 users are encouraged to contribute to the simulator through coding, documenting, testing, translating and porting the simulator. GNUSim8085 development is becoming active as of 09/2016.[5]

Editor

1. Program editor with interactive input wizard for all the standard instructions
2. Syntax highlighting in editor to distinguish between instructions, operands, comments etc.
3. A separate opcode view which displays assembled code in hex

Assembler

1. Support for all standard instructions of the 8085
2. Minimalistic support for three assembler directives (.equ, .db, .ds) to control data locations, there exist no directives to directly control code locations
3. Code start is defined outside source code ("load me at" entry) - if not defined (default), code is generated (strangely) from 4200h (instead from the real reset vector 0000h)
4. Assembly results can be stored as listing file only (no binary file output)

Debugger

1. Complete view of registers and flags
2. Support for breakpoints
3. Step by step execution/debugging of program
4. Hex / Decimal Converter
5. Runtime inspection of stack and source code variables defined
6. Runtime inspection and manipulation of memory and I/O ports

Printing

1. Printing of program from editor as well as assembled hex code (known not to work well in Windows)

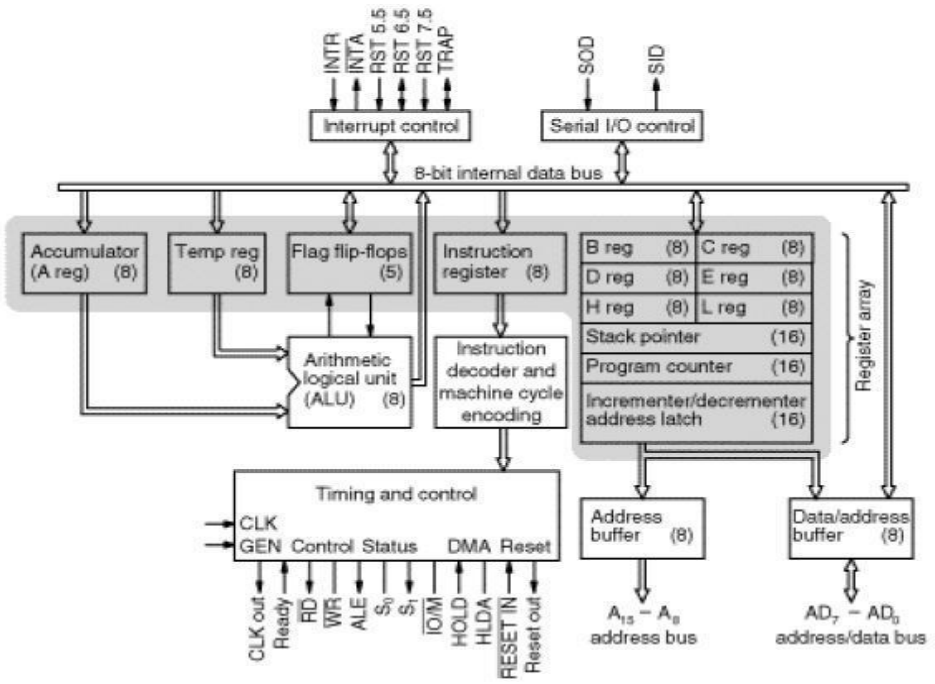
Key Features of GNU 8085 Simulator

- A simple editor component with syntax highlighting.
- A keypad to input assembly language instructions with appropriate arguments.
- Easy view of register contents.
- Easy view of flag contents.
- Hexadecimal <–> Decimal converter.
- View of stack, memory and I/O contents.
- Support for breakpoints for program debugging.
- Stepwise program execution.
- One click conversion of assembly program to opcode listing.
- Printing support.
- UI translated in various languages.

8085 MICROPROCESSOR

Introduction

The 8085 microprocessor was made by Intel in mid 1970s. It was binary compatible with 8080 microprocessor but required less supporting hardware thus leading to less expensive microprocessor systems. It is a general purpose microprocessor capable of addressing 64k of memory. The device has 40 pins, require a +5V power supply and can operate with 3 MHz single phase clock. It has also a separate address space for up to 256 I/O ports. The instruction set is backward compatible with its predecessor 8080 even though they are not pin-compatible.



The 8085 has a 16 bit address bus which enables it to address 64 KB of memory, a data bus 8 bit wide and control buses that carry essential signals for various operations. It also has a built in register array which are usually labeled A(Accumulator), B, C, D, E, H, and L. Further special-purpose registers are the 16-bit Program Counter (PC), Stack Pointer (SP), and 8-bit flag register F. The microprocessor has three maskable interrupts (RST 7.5, RST 6.5 and RST 5.5), one Non-Maskable interrupt (TRAP), and one externally serviced interrupt (INTR). The RST n.5 interrupts refer to actual pins on the processor a feature which permitted simple systems to avoid the cost of a separate interrupt controller chip.

Control Unit

It generates signals within microprocessor to carry out the instruction, which has been decoded. In reality causes certain connections between blocks of the processor be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Arithmetic Logic Unit

The ALU performs the actual numerical and logic operation such as “add”, “subtract”, “AND”, “OR”, etc. Uses data from memory and from Accumulator to perform arithmetic and always stores the result of operation in the Accumulator.

Registers

The 8085 microprocessor includes six registers, one accumulator, and one flag register, as shown in Fig 1. In addition, it has two 16-bit registers: the stack pointer and the program counter. The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H, and L as shown in Fig 1. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

Accumulator

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flag Registers

The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero(Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer (SP)

The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

Instruction Register / Decoder

This is a temporary storage for the current instruction of a program. Latest instruction is sent to here from memory prior to execution. Decoder then takes instruction and “decodes” or interprets the instruction. Decoded instruction is then passed to next stage.

Memory Address Register (MAR)

It holds addresses received from PC for eg: of next program instruction. MAR feeds the address bus with address of the location of the program under execution.

Control Generator

It generates signals within microprocessor to carry out the instruction which has been decoded. In reality it causes certain connections between blocks of the processor to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Register Selector

This block controls the use of the register stack. Just a logic circuit which switches between different registers in the set will receive instructions from Control Unit.

8085 System Bus

The microprocessor performs four operations primarily.

• Memory Read
• Memory Write
• I/O Read
• I/O Write

All these operations are part of the communication processes between microprocessor and peripheral devices. The 8085 performs these operations using three sets of communication lines called buses - the address bus, the data bus and the control bus.

Address Bus

The address bus is a group of 16 lines. The address bus is unidirectional: bits flow only in one direction – from the 8085 to the peripheral devices. The microprocessor uses the address bus to perform the first function: identifying a peripheral or memory location. Each peripheral or memory location is identified by a 16 bit address. The 8085 with its 16 lines is capable of addressing 64 K memory locations.

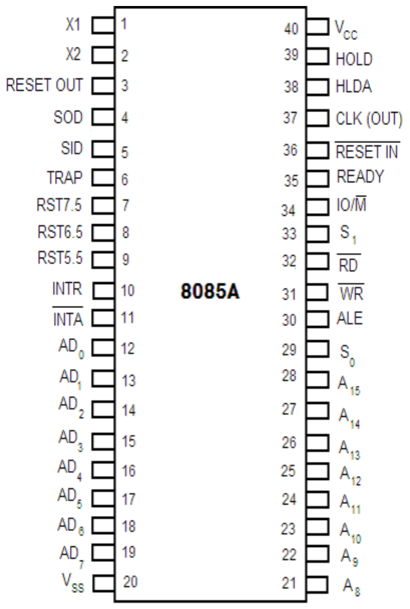
Data Bus

The data bus is a group of eight lines used for dataflow. They are bidirectional: data flows in both direction between the 8085 and memory and peripheral devices. The 8 lines enable the microprocessor to manipulate 8-bit data ranging from 00 to FF.

Control Bus

The control bus consists of various single lines that carry synchronization signals. These are not groups of lines like address of data bus but individual lines that provide a pulse to indicate an operation. The 8085 generates specific control signal for each operation it performs. These signals are used to identify a device type which the processor intends to communicate.

8085 Pin Diagram



8085 Pin Description
Properties

- ƒ Single + 5V Supply
- ƒ 4 Vectored Interrupts (One is Non Maskable)
- ƒ Serial In/Serial Out Port
- ƒ Decimal, Binary, and Double Precision Arithmetic
- ƒ Direct Addressing Capability to 64K bytes of memory

A8-A15 (Output 3 states)
Address Bus carries the most significant 8 bits of the memory address or the 8 bits of the I/0 address; 3 stated during Hold and Halt modes.

AD0 - AD 7 (Input/Output 3state)
Multiplexed Address/Data Bus carries Lower 8 bits of the memory address (or I/O address) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

ALE (Output)
Address Latch Enable occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. ALE can also be used to strobe the status information. ALE is never 3 stated.

SO, S1 (Output)
Data Bus Status: Encoded status of the bus cycle

S1	S0	
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

RD (Output 3state)
READ indicates the selected memory or 1/0 device is to be read and that the Data Bus is available for the data transfer.

WR (Output 3state)
WRITE indicates the data on the Data Bus is to be written into the selected memory or 1/0 location. Data is set up at the trailing edge of WR. 3 stated during Hold and Halt modes.

READY (Input)
If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.

HOLD (Input)
HOLD indicates that another Master is requesting the use of the address and data buses. The CPU, upon receiving the Hold request, will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue. The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

HLDA (Output)
HOLD ACKNOWLEDGE indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

INTR (Input)
INTERRUPT REQUEST is used as a general purpose interrupt. It is sampled only using the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

INTA (Output)
INTERRUPT ACKNOWLEDGE is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

RST 5.5/ RST 6.5/ RST 7.5
RESTART INTERRUPTS have the same timing as I NTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 → Highest Priority
RST 6.5
RST 5.5 → Lowest Priority

The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

TRAP (Input)
Trap interrupt is a non-maskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

RESET IN (Input)
Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip flops. None of the other flags or registers (except the instruction register) are affected The CPU is held in the reset condition as long as Reset is applied.

RESET OUT (Output)
It indicates that CPU is been reset. It used as a system RESET. The signal is synchronized to the processor clock.

X1, X2 (Input)
Crystal or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

CLK (Output)
Clock Output is used as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

IO/M (Output)
IO/M indicates whether the Read/Write is to memory or l/O. It is tri stated during Hold and Halt modes.

SID (Input)
Serial input data line: The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

SOD (output)
Serial output data line: The output SOD is set or reset as specified by the SIM instruction.

Vcc	+5V supply.
Vss	Ground Reference

8085 Addressing modes
They are mainly classified into four:
- Immediate addressing.
- Register addressing.
- Direct addressing.
- Indirect addressing.

Immediate addressing
Data is present in the instruction. Load the immediate data to the destination provided.
Example: MVI R, data

Register addressing
Data is provided through the registers.
Example: MOV Rd, Rs

Direct addressing
It is used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device. Accept the data from the port 00H and store them into the accumulator or Send the data from the accumulator to the port 01H.
Example: IN 00H or OUT 01H

Indirect Addressing
In this mode the Effective Address is calculated by the processor and the contents of the address (and the one following) are used to form a second address. The second address is where the data is stored. Note that this requires several memory accesses; two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded into the register.

Instruction Format of 8085:

Each Instruction Format of 8085 and Data Format of 8085 microprocessor has specific information fields. These information fields of instructions are called **elements of instruction**.

These are:

- **Operation code:** The operation code field in the instruction specifies the operation to be performed. The operation is specified by binary code, hence the name operation code or simply opcode. For example, for 8085 processor operation code for ADD B instruction is 80H.
- **Source / destination operand:** The source/destination operand field directly specifies the source/destination operand for the instruction. In the Instruction Format of 8085, the instruction MOV A,B has B register contents as a source operand and A register contents as a destination operand because this instruction copies the contents of register B to register A.
- **Source operand address:** We know that the operation specified by the instruction may require one or more operands. The source operand may be in the 8085 register or in the memory. Many times the Instruction Format of 8085 specifies the address of the source operand so that operand(s) can be accessed and operated by the 8085 according to the instruction.

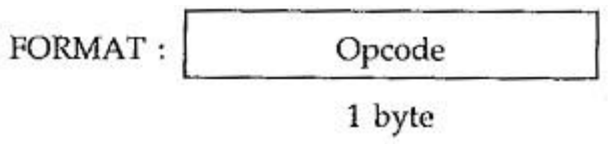
In 8085, the source operand address for instruction ADD M is given by HL register pair.

- **Destination operand address:** The operation executed by the 8085 may produce result. Most of the times the result is stored in one of the operand. Such operand is known as destination operand. The Instruction and Data Format of 8085 which produce result specifies the destination operand address. In 8085, the destination operand address for instruction INR M is given by HL register pair because INR M instruction increments the contents of memory location specified by HL register pair and stores the result in the same memory location.
- **Next instruction address :** The next instruction address tells the 8085 from where to fetch the next instruction after completion of execution of current instruction. For BRANCH instructions the address of the next instruction is specified within the instruction. However, for other instructions, the next instruction to be fetched immediately follows the current instruction. For example, in 8085, instruction after INR B follows it. The instruction JMP 2000H specifies the next instruction address as 2000H.

Instruction Formats:

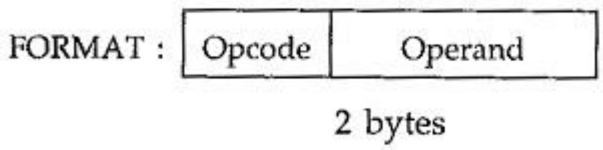
The Instruction Format of 8085 set consists of one, two and three byte instructions. The first byte is always the opcode; in two-byte instructions the second byte is usually data; in three byte instructions the last two bytes present address or 16-bit data.

1. One byte instruction:



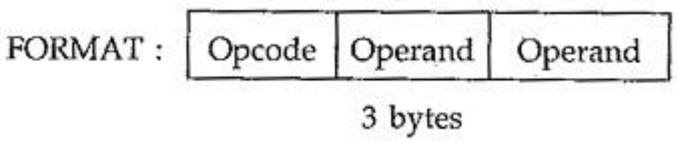
For Example: MOV A, B whose opcode is 78H which is one byte. This Instruction and Data Format of 8085 copies the contents of B register in A register.

2. Two byte instruction :



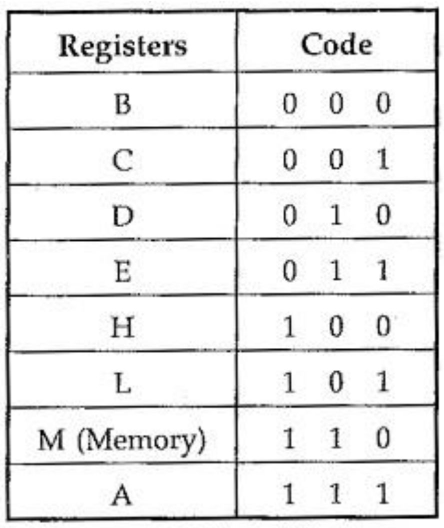
For Example: MVI B, 02H. The opcode for this instruction is 06H and is always followed by a byte data (02H in this case). This instruction is a two byte instruction which copies immediate data into B register.

3. Three byte instruction :



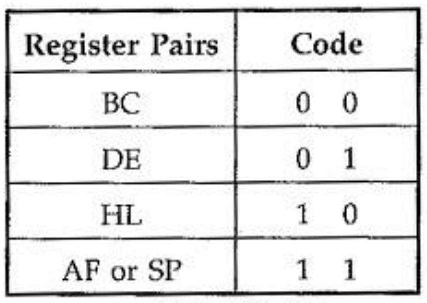
For Example: JMP 6200H. The opcode for this instruction is C3H and is always followed by 16 bit address (6200H in this case). This instruction is a three byte instruction which loads 16 bit address into program counter.

Opcode Format of 8085:
The 8085A microprocessor has 8-bit opcodes. The opcode is unique for each Instruction and Data Format of 8085 and contains the information about operation, register to be used, memory to be used etc. The 8085A identifies all operations, registers and flags with a specific code. For example, all internal registers are identified as shown in the Tables 2.1(a) and 2.2(b).



Registers	Code
B	0 0 0
C	0 0 1
D	0 1 0
E	0 1 1
H	1 0 0
L	1 0 1
M (Memory)	1 1 0
A	1 1 1

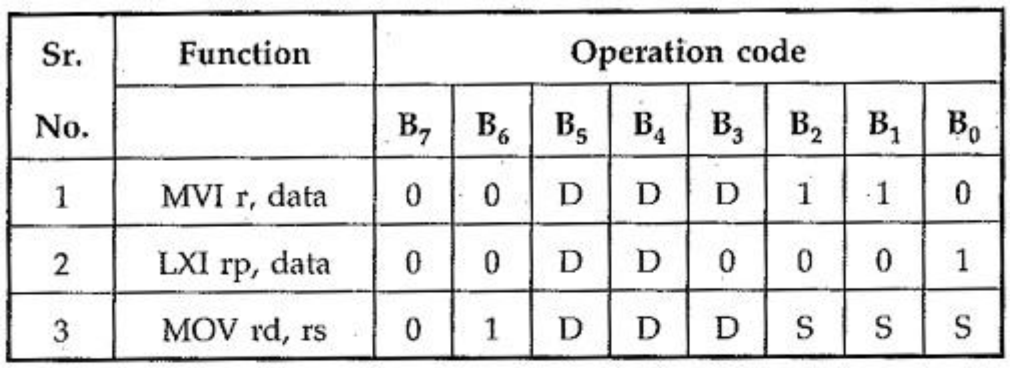
Table 2.1(a)



Register Pairs	Code
BC	0 0
DE	0 1
HL	1 0
AF or SP	1 1

Table 2.1 (b)

Similarly, there are different codes for each opera are identified as follows :



Sr.	Function	Operation code							
No.		B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
1	MVI r, data	0	0	D	D	D	1	1	0
2	LXI rp, data	0	0	D	D	0	0	0	1
3	MOV rd, rs	0	1	D	D	D	S	S	S

Table 2.2

Note: DDD defines the destination register, SSS defines the source register and DD defines the register pair.

Data Format of 8085 Microprocessor:

The operand is another name for data. It may appear in different forms :

- **Addresses**
- **Numbers/Logical data and**
- **Characters**

Addresses: The address is a 16-bit unsigned integer ,number used to refer a memory location.

Numbers/Data: The 8085 supports following numeric data types.

- **Signed Integer:** A signed integer number is either a positive number or a negative number. In 8085, 8-bits are assigned for signed integer, in which most significant bit is used for sign and remaining seven bits are used for Sign bit 0 indicates positive number whereas sign bit 1 indicates negative number.
- **Unsigned Integer:** The 8085 microprocessor supports 8-bit unsigned integer.
- **BCD:** The term BCD number stands for binary coded decimal number. It uses ten digits from 0 through 9. The 8-bit register of 8085 can store two digit BCD

Characters: The 8085 uses ASCII code to represent characters. It is a 7-bit alphanumeric code that represents decimal numbers, English alphabets, and other special characters.

Introduction to Microcontroller 8051

The most universally employed set of microcontrollers come from the 8051 family. 8051 Microcontrollers persist to be an ideal choice for a huge group of hobbyists and experts. In the course of 8051, the humankind became eyewitness to the most ground- breaking set of microcontrollers. The original 8051 microcontroller was initially invented by Intel. The two other members of this 8051 family are-

- 8052-This microcontroller has 3 timers & 256 bytes of RAM. Additionally it has all the features of the traditional 8051 microcontroller. 8051 microcontroller is a subset of 8052 microcontroller.
- 8031 - This microcontroller is ROM less, other than that it has all the features of a traditional 8051 microcontroller. For execution an external ROM of size 64K bytes can be added to its chip.

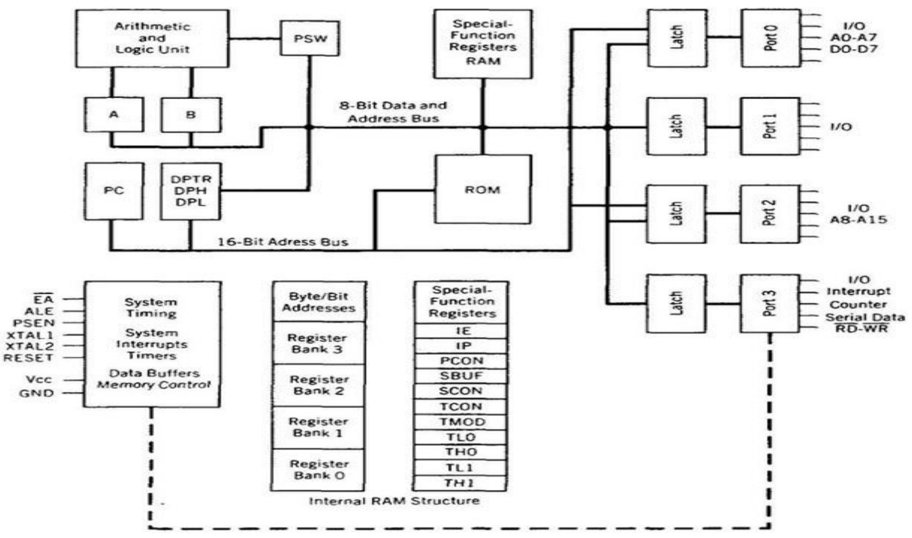
8051 microcontroller brings into play 2 different sorts of memory such as - NV-RAM, UV - EPROM and Flash.
8051 is the basic microcontroller to learn embedded systems projects.

FEATURES OF 8051

8051 microcontroller is an eight bit microcontroller launched. It is available in 40 pin DIP (dual inline package). It has 4kB of ROM (on- chip programmable space) and 128 bytes of RAM space which is inbuilt, if desired 64KB of external memory can be interfaced with the microcontroller. There are four parallel 8 bits ports which are easily programmable as well as addressable.

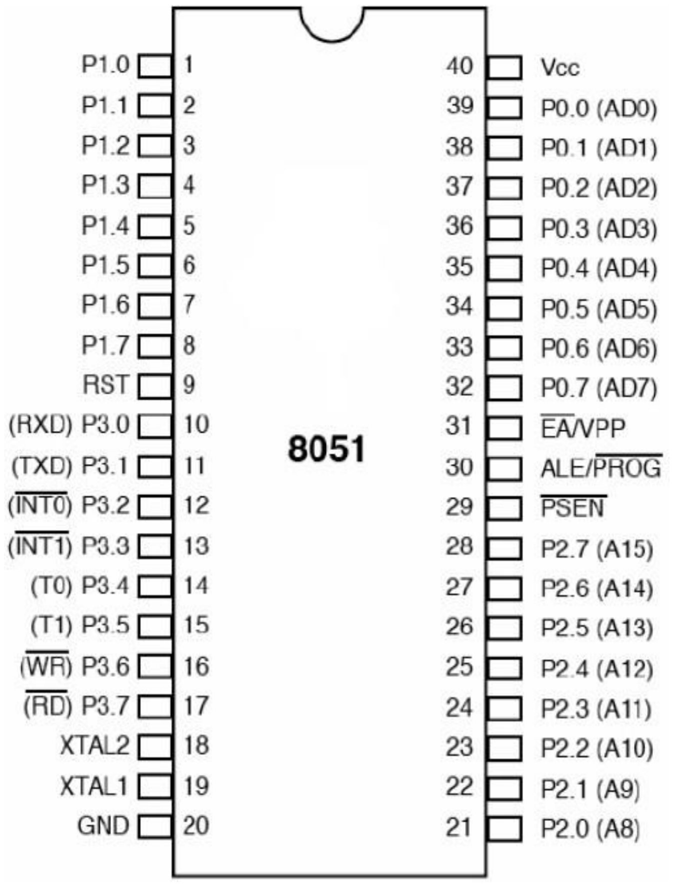
An on- chip crystal oscillator is integrated in the microcontroller which has crystal frequency of 12MHz. In the microcontroller there is a serial input/output port which has 2 pins. Two timers of 16 bits are also incorporated in it; these timers can be employed as timer for internal functioning as well as counter for external functioning.
The microcontroller comprise of 5 interrupt sources namely- Serial Port Interrupt, Timer Interrupt 1, External Interrupt 0, Timer Interrupt 0, External Interrupt 1.
The programming mode of this micro-controller includes GPRs (general purpose registers), SFRs (special function registers) and SPRs (special purpose registers).

INTERNAL ARCHITECHURE OF 8051 MICRO-CONTROLLER


Arithmetic and Logic Unit
PSW
Special-Function Registers RAM
A
B
8-Bit Data and Address Bus
PC
DPTR DPH DPL
ROM
16-Bit Adress Bus
Latch
Port 0
Port 1
Port 2
Port 3
I/O AO-A7 DO-D7
I/O
I/O A8-A15
I/O Interrupt Counter Serial Data RD-WR
EA ALE PSEN XTAL1 XTAL2 RESET Vcc GND
System Timing
System Interrupts Timers
Data Buffers Memory Control
Byte/Bit Addresses
Register Bank 3
Register Bank 2
Register Bank 1
Register Bank 0
Special-Function Registers
IE IP PCON SBUF SCON TCON TMOD TLO THO TL1 TH1
Internal RAM Structure

1. **ALU**
   All arithmetic and logical functions are carried out by the ALU.
   Addition, subtraction with carry, and multiplication come under arithmetic operations.
   Logical AND, OR and exclusive OR (XOR) come under logical operations.
2. **Program Counter (PC)**
   A program counter is a 16-bit register and it has no internal address. The basic function of program counter is to fetch from memory the address of the next instruction to be executed. The PC holds the address of the next instruction residing in memory and when a command is encountered, it produces that instruction. This way the PC increments automatically, holding the address of the next instruction.
3. **Registers**
   Registers are usually known as data storage devices. 8051 microcontroller has 2 registers, namely Register A and Register B. Register A serves as an accumulator while Register B functions as a general purpose register. These registers are used to store the output of mathematical and logical instructions. The operations of addition, subtraction, multiplication and division are carried out by Register A. Register B is usually unused and comes into picture only when multiplication and division functions are carried out by Register A. Register A also involved in data transfers between the microcontroller and external memory.

PIN DIAGRAM OF 8051 MICRO-CONTROLLER



PINOUT DESCRIPTION

Pins 1-8: Port 1 pins can be configured as an input or an output.

Pin 9: RS

A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.

Pins10-17: Port 3 Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions:

Pin 10: RXD Serial asynchronous communication input or Serial synchronous communication output.

Pin 11: TXD Serial asynchronous communication output or Serial synchronous Communication clock output.

Pin 12: INT0 Interrupt 0 input.

Pin 13: INT1 Interrupt 1 input.

Pin 14: T0 Counter 0 clock input.

Pin 15: T1 Counter 1 clock input.

Pin 16: WR Write to external (additional) RAM.

Pin 17: RD Read from external RAM.

Pin 18, 19: X2 X1 Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins. Instead of it, miniature ceramics resonators can also be used for frequency stability. Later versions of microcontrollers operate at a frequency of 0 Hz up to over 50 Hz.

Pin 20: GND Ground.

Pin 21-28: Port 2 If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.

Pin 29: PSEN If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.

Pin 30: ALE Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external register (usually 74HCT373 or 74HCT375 add-on chip) memorizes the state of P0 and uses it as a memory chip address. Immediately after that, the ALU pin is returned its previous logic state and P0 is now used as a Data Bus. As seen, port data multiplexing is performed by means of only one additional (and cheap) integrated circuit. In other words, this port is used for both data and address transmission.

Pin 31: EA By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).

Pin 32-39: Port 0 Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).

Pin 40: VCC +5V power supply.

OPERATION OF 8051 KIT

- Switch on power supply. Message “ANSHUMAN” will be displayed.
- Press “E” &then “ENTER” key.
- Select C=A & then press enter .default 6000 address will be displayed.
    - Note: for changing address select C=A address.
- Now enter the program. At the end press “ENTER” key twice.
- Then C= will be displayed. Press “Q”.
- Press “S” & press enter.
- By pressing any key, select, EXT. memory, register. etc. &press “enter” key.
- For register, select general (AS, DPL, DPR etc),BANK etc. press enter.
- Now enter the inputs &press enter key
- Press “G” press “enter” key.
- BURST will be displayed. Press enter.
- ADDR will be displayed. Esc 6000&press enter.
- Wait, DONE message will be displayed.
- Now to view output, press “S”& press “ENTER”.

PROCEDURE FOR PROGRAMS ON KEIL SOFTWARE

- Click on Keil uvision3.
- Click on **'Project'**, create a new project and save it in a new folder choose target option for **Atmel** and **AT89C51**.
- Go to File, click on new file, and type the program.
- Go to File, click on **'save as'**, save the program with extension **.asm** on your particular folder where you saved your project.
- Add your program to **Source Group 1** which is at Target1 (Project workspace) which is created after selecting the target in step 2.
    - To do this right clicks on Source Group 1 and select 'Add files to Source Group 1'.
    - Search your code with .asm extension.
- Now Click on **Translate current file** tab present file toolbar and check for errors. If error present then rectify.
- Click on **Rebuild all target files** to add our program to the AT89C51 target.
- Go to **Debug**, click on S**tart/Stop debug session**.
- For giving input data: Go to **view**, click on **Memory window**.
    - Enter **inputs** for corresponding memory addresses.
        - For internal memory type: i:0x20 for example
        - For external memory type: x:0x2000 for example
- Now click on **"Run",** check the results.
- While in Debug don't make any changes in the program.
- After running, again click S**tart/Stop debug session** to edit mode for changes in program.

USING 8085 GNU SIMULATOR

Demo Program (A):

ADDITION OF TWO 8 BIT NUMBERS

AIM: To implement assembly language program for addition of two 8-bit numbers.

APPARTUS: GNU Simulator, P.C.

ALGORITHM:

1) Start the program by loading the first data into Accumulator.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Store the value of sum and carry in memory location.
7) Terminate the program.

PROGRAM:

```
         JMP START

        ; DATA

        ; CODE

START:   NOP
         MVI    C, 00     Initialize C register to 00
         LDA    4150      Load the value to Accumulator.
         MOV    B, A      Move the content of Accumulator to B register.
         LDA    4151      Load the value to Accumulator.
         ADD    B         Add the value of register B to A
         JNC    LOOP      Jump on no carry.
         INR    C         Increment value of register C
 LOOP:   STA    4152      Store the value of Accumulator (SUM).
         MOV    A, C      Move content of register C to Acc.
         STA    4153      Store the value of Accumulator (CARRY)
         HLT              Halt the program.
```

OBSERVATION:

```
Input:      80 (4150)
            80 (4151)
Output:     00 (4152)
            01 (4153)
```

RESULT:

Thus the program to add two 8-bit numbers was executed.

Demo Program (B):

SUBTRATION OF TWO 8 BIT NUMBERS

AIM: To implement assembly language program for subtraction of two 8-bit numbers.

APPARTUS: GNU Simulator, P.C.

ALGORITHM:

1. Start the program by loading the first data into Accumulator.
2. Move the data to a register (B register).
3. Get the second data and load into Accumulator.
4. Subtract the two register contents.
5. Check for carry.
6. If carry is present take 2's complement of Accumulator.
7. Store the value of borrow in memory location.
8. Store the difference value (present in Accumulator) to a memory
9. location and terminate the program.

PROGRAM:

```
JMP START
          ; DATA

          ; CODE
START:   NOP

          MVI      C, 00      Initialize C to 00
          LDA      4150       Load the value to Acc.
          MOV      B, A       Move the content of Acc to B register.
          LDA      4151       Load the value to Acc.
          SUB      B
          JNC      LOOP       Jump on no carry.
          CMA                 Complement Accumulator contents.
          INR      A          Increment value in Accumulator.
          INR      C          Increment value in register C
 LOOP:   STA      4152       Store the value of A-reg to memory address.
          MOV      A, C       Move contents of register C to Accumulator.
          STA      4153       Store the value of Accumulator memory address.
          HLT                 Terminate the program.
```

OBSERVATION:

```
Input :  06 (4150)
         02 (4151)
Output:  04 (4152)
         01 (4153)
```

RESULT:

Thus the program to subtract two 8-bit numbers was executed.

Exp.No.01 (a)

ASSEMBLY LANGUAGE PROGRAM TO MULTIPLY TWO 8-BIT SIGNED/UNSIGNED NUMBERS

AIM: To implement assembly language program to multiply two 8-bit **signed numbers**.

APPARTUS: GNU Simulator and PC

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register (B register).
3) Get the second data and load into Accumulator.
4) Add the two register contents.
5) Check for carry.
6) Increment the value of carry.
7) Check whether repeated addition is over and store the value of product and carry in memory location.
8) Terminate the program.

PROGRAM:

```
      JMP START
                ; DATA

                ; CODE
          START:  NOP

                  MVI      D, 00       Initialize register D to 00
                  MVI      A, 00       Initialize Accumulator content to 00
                  LXI      H, 4150     LXI indicates for pair register & H – HL pair
                  MOV      B, M        Get the first number in B – reg   ;  M→ [HLpair]
                  INX      H
                  MOV      C, M        Get the second number in C- reg.
        LOOP:     ADD      B           Add content of A - reg to register B.
                  JNC      NEXT        Jump on no carry to NEXT.
                  INR      D           Increment content of register D
        NEXT:     DCR      C           Decrement content of register C.
                  JNZ      LOOP        Jump on no zero to address
                  STA      4152        Store the result in Memory
                  MOV      A, D
                  STA      4153        Store the MSB of result in Memory
                  HLT                  Terminate the program.
```

OBSERVATION:

Input :	FF (4150)
	FF (4151)
Output:	01 (4152)
	FE (4153)

RESULT:

Thus the program to multiply two 8-bit numbers was executed.

TASK: Complete the unsigned multiplication of unsigned two 8-bit numbers

Exp.No. 01(b)

ASSEMBLY LANGUAGE PROGRAM FOR SIGNED/UNSIGNED DIVISION OF TWO NUMBERS

AIM: To implement assembly language program for division of two8-bit numbers.

APPARTUS: GNU Simulator, P.C.

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.
2) Move the data to a register(B register).
3) Get the second data and load into Accumulator.
4) Compare the two numbers to check for carry.
5) Subtract the two numbers.
6) Increment the value of carry .
7) Check whether repeated subtraction is over and store the value of product and carry in memory location.
8) Terminate the program.

PROGRAM:

```
              JMP START
              ; DATA

              ; CODE
      START:   NOP
               LXI     H, 2050
               MOV     B, M       Get the dividend in B – reg.
               MVI     C, 00      Clear C – reg for qoutient
               INX     H
               MOV     A, M       Get the divisor in A – reg.
    NEXT:      SUB     B          Subtract A – reg from B- reg.
               JC      LOOP       Jump on carry to LOOP
               INR     C          Increment content of register C.
               JMP     NEXT       Jump to NEXT
    LOOP:      MOV     A, C
                STA    2052       Store the quotient in memory
                HLT               Terminate the program.
```

OBSERVATION:

Input :	02 (2050)
	08 (2051)
Output:	04 (2052)

RESULT:

Thus the program to division of two 8-bit numbers was executed.

TASK: Complete the unsigned division of unsigned two 8-bit numbers

Exp.No. 02 (A)

ASSEMBLY LANGUAGE PROGRAM TO FIND AVERAGE OF 8-BIT NUMBERS IN AN ARRAY

AIM: To implement ALP to find average of 8-bit numbers in array.

APPARTUS:

GNU Simulator, P.C.

ALGORITHM:

1. Start the program by loading HL register pair with address of memory location.
2. Move the data to a B register.
3. Get the second data and load into Accumulator.
4. Compare the two numbers to check for carry.
5. Subtract the two numbers.
6. Increment the value of carry .
7. Check whether repeated subtraction is over and store the value of product and carry in memory location.
8. Terminate the program

PROGRAM:

```
JMP START
                ; DATA
                ; CODE
   START:     NOP
              LXI      H, 5000
              MVI     C,10
              MVI     A, 00
              MVI     B,00
              MVI     E,00
    NEXT:     ADD M
              JNC SKIP
              INR B
    SKIP:     INX H
              DCR C
              JNZ      NEXT
              MOV M,B
              MVI D,10
   LOOP1:     SUB D
              JC LOOP2
              INR E
              JMP LOOP1
  LOOP2:      MOV A,E
              STA 5050
              HLT
```

RESULT:

Inputs:

5000	5001	5002	5003	5004	5005	5006	5007	5008	5009
01	02	03	04	05	06	07	08	09	02

Output: 5020(ext. memory location) – 04

Exp.No. 2 (B)

ASSEMBLY LANGUAGE PROGRAM TO FIND LARGEST NUMBER IN AN ARRAY

AIM: To implement ALP to find the largest number in the array.

APPARTUS:

GNU Simulator, P.C.

ALGORITHM:

1) Load the address of the first element of the array in HL pair
2) Move the count to B – reg.
3) Increment the pointer
4) Get the first data in A – reg.
5) Decrement the count.
6) Increment the pointer
7) Compare the content of memory addressed by HL pair with that of A - reg.
8) If Carry = 0, go to step 10 or if Carry = 1 go to step 9
9) Move the content of memory addressed by HL to A – reg.
10) Decrement the count
11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
12) Store the largest data in memory.
13) Terminate the program.

PROGRAM:

```
JMP START
          ; DATA

          ; CODE
START:    NOP
          LXI H, 2000
          MVI C,0AH
          MVI A,00H
LOOP:     CMP M   ; compare M with A
          JNC SKIP
          MOV A,M
SKIP:     INX H
          DCR C
          JNZ   LOOP
          MOV M,A
          HLT
```

RESULT:

Inputs:

2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
01	02	03	04	09	06	07	08	05	02

Output: A(Accumulator) – 09

Thus the program to find the largest number in an array of data was executed

Exp.No. 2(B)(Contd…)

ASSEMBLY LANGUAGE PROGRAM TO FIND SMALLEST IN AN ARRAY

AIM: To implement ALP to find the smallest number in the array.

APPARTUS:

GNU Simulator, P.C.

ALGORITHM:

1) Load the address of the first element of the array in HL pair
2) Move the count to C – reg.
3) Increment the pointer
4) Get the first data in A – reg.
5) Decrement the count.
6) Increment the pointer
7) Compare the content of memory addressed by HL pair with that of A - reg.
8) If carry = 1, go to step 10 or if Carry = 0 go to step 9
9) Move the content of memory addressed by HL to A – reg.
10) Decrement the count
11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
12) Store the smallest data in memory.
13) Terminate the program.

PROGRAM:

```
          JMP START
               ; DATA

               ; CODE
    START:  NOP
              LXI H,2000
              MVI C, 10
              MVI A, 50
  LOOP:      CMP M
             JC SKIP
             MOV A, M
 SKIP:       INX H
             DCR C
             JNZ LOOP
             MOV M,A
             HLT
```

RESULT:

Inputs:

2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
01	02	03	04	09	06	07	08	05	02

Output: A(Accumulator) – 01

Thus the program to find the smallest number in an array of data was executed

Exp.No. 2 (C)

ASSEMBLY LANGUAGE PROGRAM TO FIND SQUARE ROOT OF A GIVEN NUMBER

AIM: To implement ALP to find the square root of a given number.

APPARATUS:

GNU Simulator, P.C.

ALGORITHM:

1. Assign 01 to register D and E
2. Load the value, stored at memory location 2050 in accumulator A
3. Subtract value stored at accumulator A from register D
4. Check if accumulator holds 0, if true then jump to step 8
5. Increment value of register D by 2
6. Increment value of register E by 1
7. Jump to step 3
8. Move value stored at register E in A
9. Store the value of A in memory location 2060

PROGRAM:

```
JMP START
          ; DATA

          ; CODE
START:    NOP
          MVI D, 01        ; initialize register D with 01; MVI → move immediate
                              data
          MVI E, 01        ; initialize register E with 01
          LDA 2050         ; loads the content of memory location 2050 in
                           accumulator A
LOOP1:    SUB D            ; subtract value of D from A
          JZ     LOOP2     ; make jump to memory location LOOP2 if zero flag is
                           set
                           ; Increments value of register D by 1. Since it is used
          INR D            two times,
                             therefore value of D is incremented by 2
          INR D
          INR E            ; increments value of register E by 1
          JMP    LOOP1     ; make jump to memory location LOOP1
LOOP2:    MOV A, E         ; moves the value of register E in accumulator A
          STA 2060         ; stores value of A in 2060
          HLT              ; stops executing the program and halts any further
                           execution
```

OBSERVATION:

Input:	09 (2050)
Output:	03 (2060)

RESULT:

Thus the program to find the square root of a given number was executed.

Exp.No. 3(A)

ASSEMBLY LANGUAGE PROGRAM FOR ASCENDING ORDER OF NUMBERS IN AN ARRAY

AIM: To implement ALP for ascending order of given numbers in an array.

APPARATUS: GNU Simulator, P.C.

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGARM:

```
        JMP START
                ; DATA

                ; CODE
        START:  NOP

        LXI         H,4200
        MOV         C,M
        DCR         C
REPEAT: MOV         D,C
        LXI         H,4201
LOOP:   MOV         A,M
        INX         H
        CMP         M
        JC          SKIP
        MOV         B,M
        MOV         M,A
        DCX         H
        MOV         M,B
        INX         H
SKIP:   DCR         D
        JNZ         LOOP
        DCR         C
        JNZ         REPEAT
        HLT
```

OBSERVATION:

Input:	4200	05 (Array Size)
	4201	05
	4202	04
	4203	03
	4204	02
	4205	01

Output:	4200	05(Array Size)
	4201	01
	4202	02
	4203	03
	4204	04
	4205	05

RESULT:

Thus the given array of data was arranged in ascending order.

Exp.No. 3(B)

ASSEMBLY LANGUAGE PROGRAM FOR DESCENDING ORDER OF NUMBERS IN AN ARRAY

AIM: To implement ALP for descending order of given numbers in an array.

APPARATUS: GNU Simulator, P.C.

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGRAM:

```
        JMP START
                ; DATA

                ; CODE
      START:    NOP
                LXI     H,4200
                MOV     C,M
                DCR     C
REPEAT:         MOV     D,C
                LXI     H,4201
LOOP:           MOV     A,M
                INX     H
                CMP     M
                JNC     SKIP
                MOV     B,M
                MOV     M,A
                DCX     H
                MOV     M,B
                INX     H
SKIP:           DCR     D
                JNZ     LOOP
                DCR     C
                JNZ     REPEAT
                HLT
```

OBSERVATION:

Input:	4200	05 (Array Size)
	4201	01
	4202	02
	4203	03
	4204	04
	4205	05

Output:	4200	05(Array Size)
	4201	05
	4202	04
	4203	03
	4204	02
	4205	01

RESULT:

Thus the given array of data was arranged in descending order.

Exp.No. 04

ASSEMBLY LANGUAGE PROGRAM TO CONVERT BCD NUMBER TO SEVEN SEGMENT

AIM: To implement ALP to convert the BCD number to seven segment number.

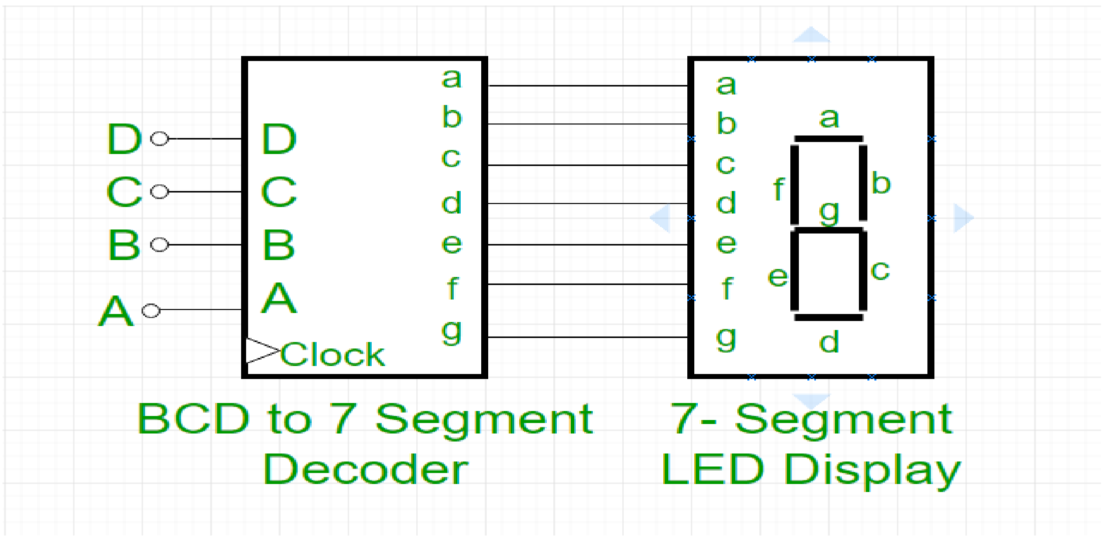
APPARATUS: GNU Simulator, P.C.

ALGORITHM:

1. Start.
2. Initialize the data segment.
3. Clear the base register.
4. Initialize the counter.
5. Rotate the number, check for '1'.
6. Result is displayed.
7. Stop.

LOOKUP TABLE:

	Common Cathode	**Common Anode**
BCD NUMBER	**EQUIVALENT SEVEN SEGMENT NUMBER**	**EQUIVALENT SEVEN SEGMENT NUMBER**
0	3F	40
1	06	79
2	5B	24
3	4F	30
4	66	19
5	6D	12
6	7D	02
7	07	78
8	7F	00
9	6F	10



PROGRAM:

```
        JMP START
                ; DATA

                ; CODE
      START:  NOP
              LXI     H, 6000      ; Initialize lookup table pointer
              LXI     D, 6020      ; Initialize source memory pointer
              LXI     B, 6050       ; Initialize destination memory pointer
      BACK:   LDAX   D             ; Get the number → load accumulator from memory
                                                   pointed by external register
              MOV    L, A             ; A point to the 7-segment code
              MOV    A, M             ; Get the 7-segment code
              STAX   B                ; Store the result at destination memory location
              HLT                     ; End the program
```

RESULT:

After assemble, enter the equivalent seven segment number in the given address

Address	EQUIVALENT SEVEN SEGMENT NUMBER
6000	40
6001	79
6002	24
6003	30
6004	19
6005	12
6006	02
6007	78
6008	00
6009	10

and also enter the source data in the given address i.e. 6020 -- 05

after debugging, check the result in the address 6050

Output: 12 (6050)

USING 8085 KIT

Exp.No. 05

ASSEMBLY LANGUAGE PROGRAM TO GENERATE TRIANGULAR, SQUARE & SAWTOOTH USING DAC

AIM: Write an 8085 program to interface 8255 PPI.

1. Generate saw tooth wave
2. triangular wave
3. Square wave using DAC interfacing

APPARATUS: 1) MP 8085 trainer kit
2) SMPS
3) DAC Interface module
4) Power Supply (5V)
5) 26 pin flat ribbon cable
6) 4/8 wire relimate cable
7) Oscilloscope
8) CRO probes

5(A). GENERATION OF SAW TOOTH WAVE:-

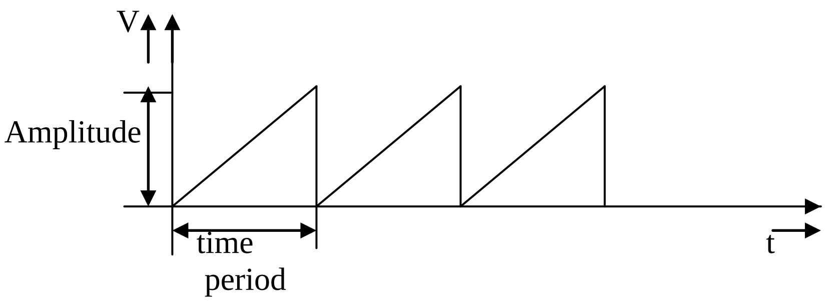
ALGORITHM:

1. Intialization a control word for 8255, for it to operate in I/O mode and for ports A,B and C to operate in output mode.
2. Clear the accumulator content and output it.
3. Increment accumulator content and compare with FFH.
4. Jump if nor zero to step 2.
5. Continue the above steps.

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE/OPERAND
C600		MVI A,80_H	3E 80
C602		OUT CWR	D3 DB
C604	START	MVI A,00_H	3E 00
C606	REPEAT	OUT PORTA	D3 D8
C608		INR A	3C
C609		CPI FF_H	FE FF
C60B		JNZ REPEAT	C2 06 C6
C60E		MVI A,00_H	3E 00
C610		OUT PORTA	D3 D8
C612		JMP START	C3 04 C6

EXPECTED WAVEFORM:



EXPECTED RESULT:

Amplitude =   Frequency =

Time Period =

5(B). TRIANGULAR WAVE GENERATION:

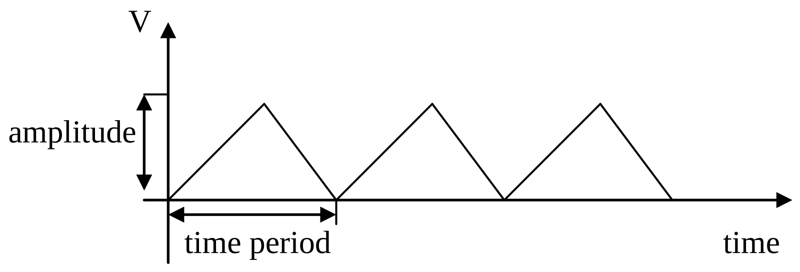
ALGORITHM:

1. Intialization a control word for 8255,for it to operate in I/O mode and for ports A, B and C to operate in output mode.
2. Clear the accumulator content and output it.
3. Increment accumulator content and compare with FFH
4. Jump if nor zero to step 2.
5. Decrement accumulator content.
6. Output it and compare with 00H and go to step5 if not zero.
7. Continue the above steps.

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE/OPERAND
C500		MVI A,80$_H$	3E 80
C502		OUT CWR	D3 DB
C504	START	MVI A,OO$_H$	3E 00
C506	POS	OUT PORTA	D3 D8
C508		INR A	3C
C509		CPI FF$_H$	FE FF
C50B		JNZ POS	C2 06 C5
C50E	NEG	DCR A	3D
C50F		OUT PORTA	D3 D8
C511		CPI 00$_H$	FE 00
C513		JNZ NEG	C2 0E C5
C516		JMP START	C3 04 C5

EXPECTED WAVEFORM:



EXPECTED RESULT:

Amplitude = Frequency =

Time Period =

5(C). SQUARE WAVE FORM GENERATION

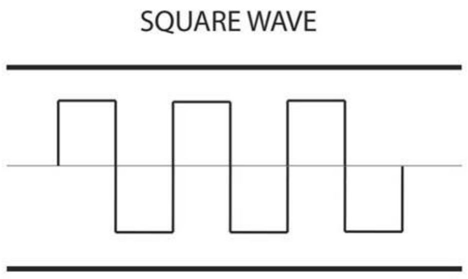
ALGORITHM:

1. Intialization a control word for 8255, for it to operate in I/O mode and for ports A, B and C to operate in output mode.
2. Clear the accumulator content and output it.
3. Call delay subroutine.
4. More immediate accumulator with FFH and output it.
5. Continue the steps 2 to 4.

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE/OPERAND
C800		MVI A,80$_H$	3E 80
C802		OUT CWR	D3 DB
C804	REPEAT	MVI A,OO$_H$	3E 00
C806		OUT PORTA	D3 D8
C808		CALL DELAY	CD 15 C8
C80B		MVI A,FF$_H$	3E FF
C80D		OUT PORTA	D3 D8
C80F		CALL DELAY	CD 15 C8
C812		JMP REPEAT	C3 04 C8
C815	DELAY	MVI C,85$_H$	0E 85
C817	AGAIN	DCR C	0D
C818		JNZ AGAIN	C2 17 C8
C81B		RET	C9

EXPECTED WAVEFORM:



EXPECTED RESULT:

Amplitude =   Frequency =

Time Period =

USING 8051 KIT

Exp. No. 6(A) (i)

ASSEMBLY LANGUAGE PROGRAM FOR MULTIBYTE ADDITION

AIM: Write 8051 program to implement multiple byte addition (addition of two 32-bit no's).

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

THEORY:

Generally 8 bits are called a byte, 16 bits are called as word, 32 bits are called as double word, and the data more than 4 byte is called as Multiple byte.

ALGORITHM:

1. Start.
2. Get the number 100.Get the first number.
3. Add result with second number.
4. Store in R_0 (or) in first number register.
5. Repeat the step for given no. of inputs.
6. Output is displayed in R_0, R_1, R_2, R_3.
7. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERANDS
6000	MOV	A, R_0
6001	ADD	A, R_4
6002	MOV	R_0, A
6003	MOV	A, R_1
6004	ADDC	A, R_5
6005	MOV	R_1, A
6006	MOV	A, R_2
6007	ADDC	A, R_6
6008	MOV	R_2, A
6009	MOV	A, R_3
600A	ADDC	A, R_7
600B	MOV	R_3, A
600C	RET	

EXPECTED RESULTS:

Inputs: R_0 = 11h, R_1 = 11h, R_2 = 11h, R_3 = 11h
R_4 = 22h, R_5 = 22h, R_6 = 22h, R_7 = 22h

Outputs: R_0 = 33h, R_1 = 33h, R_2 = 33h, R_3 = 33h

Exp. No. 6(A)(ii)

ASSEMBLY LANGUAGE PROGRAM FOR MULTIBYTE SUBTRACTION

AIM: Write 8051 program to implement subtraction of two 32 bit numbers.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

THEORY:

Generally 8 bits are called a byte, 16 bits are called as word, 32 bits are called as double word. Here we are subtracting two bytes, which are stored in the register. By using the instruction SUBB we can subtract byte by byte.

ALGORITHM:

1. Start.
2. Get the first number.
3. Subtract with the second number.
4. Store result in R_0.
5. Repeat the above steps for given no. of inputs.
6. Output is displayed in R_0, R_1, R_2, R_3.
7. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERAND
6000	CLR	C
6001	MOV	A, R_0
6002	SUBB	A, R_4
6003	MOV	R_0, A
6004	MOV	A, R_1
6005	SUBB	A, R_5
6006	MOV	R_1, A
6007	MOV	A, R_2
6008	SUBB	A, R_6
6009	MOV	R_2, A
600A	MOV	A, R_3
600B	SUBB	A, R_7
600C	MOV	R_3, A
600D	RET	

EXPECTED RESULT:

Inputs: R_0 = 55h, R_1 = 55h, R_2 = 55h, R_3 = 55h
R_4 = 22h, R_5 = 22h, R_6 = 22h, R_7 = 22h

Outputs: R_0 = 33h, R_1 = 33h, R_2 = 33h, R_3 = 33h

Exp. No. 6(B)(i)

ASSEMBLY LANGUAGE PROGRAM FOR MULTIPLICATION OF 32-BIT NUMBERS

AIM: Write 8051 program to implement multiplication.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

THEORY:

After multiplication, if it is 16 bit multiplication the result will be stored in register A and register B. If it is 8 bit multiplication then the result will be store in register A.

ALGORITHM:

1. Start.
2. Get the first number.
3. Store the number.
4. Get the second number.
5. Multiply A & B.
6. Increment data pointer.
7. Get the higher byte & lower byte of result.
8. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERANDS
6000	MOV	DPTR, #20A1
6003	MOVX	A, @DPTR
6004	MOV	F_0, A
6006	MOV	DPTR, #20A0
6009	MOVX	A, @DPTR
600A	MUL	AB
600B	MOV	DPTR, #20A2
600E	MOVX	@DPTR, A
600F	INC	DPTR
6010	MOV	A, F_0
6012	MOVX	@DPTR, A
6013	RET	

EXPECTED RESULT:

Inputs: 20A0 = 05h & 20A1 = 04h

Output: 20A2 = 14h

Exp. No. 6(B)(ii)

ASSEMBLY LANGUAGE PROGRAM FOR DIVISION OF TWO 8 BIT NUMBERS

AIM: Write 8051 program to implement division operation.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

THEORY:

After division the quotient is stored in register 'A' and the remainder will be stored in register 'B'.

ALGORITHM:

1. Start.
2. Get the first number.
3. Store the number.
4. Get the second number.
5. Divide A & B.
6. Increment data pointer.
7. Get the quotient, reminder & display.
8. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERANDS
6000	MOV	A, #00H
6003	MOV	DPTR, #20A0
6004	MOVX	A, @DPTR
6006	MOV	F_0, A
6009	MOV	A, #00H
600A	INC	DPTR
600B	MOVX	A, @DPTR
600C	DIV	A, B
600D	INC	DPTR
600E	MOVX	@DPTR, A
6011	MOV	A, F_0
6012	INC	DPTR
6013	MOVX	@DPTR, A
6014	RET	

EXPECTED RESULT:

Inputs: 20A0 = 15h & 20A1 = 03h

Output: 20A2 = 07h & 20A3 = 00h

Exp. No. 7(A)

ASSEMBLY LANGUAGE PROGRAM FOR EXCHANGE OF DATA

AIM: Write a program for exchange of data in 8051.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

ALGORITHM:

1. Start.
2. Get the first number in Accumulator
3. Get the second number in R_0
4. Swap A, and exchange with R_0.
5. Display the result.
6. Stop.

PROGRAM:

ADDR	MNEMONICS	OPERANDS
6000	MOV	A, #C5H
6002	MOV	R_0, #C6H
6004	SWAP	A
6005	XCH	A, R_0
6006	RET	

EXPECTED RESULT:

‘A’ becomes 5Ch and moved to **R_0 = 5Ch**

R_0 = C6h is moved to **A = C6h**

Exp. No. 7(C)(i)

ASSEMBLY LANGUAGE PROGRAM FOR FINDING MAXIMUM NUMBER FROM 8-BIT TEN NUMBERS

AIM: Write a program for finding the maximum number from 8-bit ten numbers in 8051 kit.

APPARATUS:

3. MC 8051 trainer kit
4. SMPS

PROGRAM:

```
6000      MOV   DPTR, #7000              ; initialize the pointer to memory where
                                           numbers are stored

6003      MOV   R0, #0A                  ; initialize the counter

6005      MOV   F0, #00                  ; maximum = 0

6008      AGAIN: MOVX A, @DPTR           ; get the number from the memory

6009      CJNE A, F0, 02                 ; NE = 600E – 600C=02, compare number with
                                           maximum

600C      AJMP 6012                      ; address of SKIP = 6012, if equal go to SKIP

600E      NE: JC 02                      ; SKIP = 6012- 6010, if not equal check for
                                           carry, if carry go to SKIP

6010      MOV F0,A                       ; otherwise maximum  = number

6012      SKIP: INC DPTR                 ; increment memory pointer

6013      DJNZ R0,F3                     ; AGAIN = FF – (6013-6007), decrement
                                           count, if count = 0 stop, otherwise go to AGAIN
6015      RET
```

EXPECTED RESULT:

INPUT:

7000	08;	7003	05;	7006	04;	7009	00
7001	02;	7004	06;	7007	07;		
7002	03;	7005	01;	7008	19;		

OUTPUT

B=19h

Forward Jump:

For SKIP and NE label=

Address of location where to jump – address of location of next instruction after jump instruction => 600E-600C=02

Backward Jump:

For AGAIN label=

No. of bytes= (address of location of the count)-(address of location where to jump)

Count=FF- No. of bytes=FF-(6013-6007)=F3

Exp. No. 7(C)(ii)

ASSEMBLY LANGUAGE PROGRAM FOR FINDING MINIMUM NUMBER FROM 8-BIT TEN NUMBERS

AIM: Write a program for finding the minimum number from 8-bit ten numbers in 8051 kit.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

PROGRAM:

```
6000      MOV   DPTR, #7000 ; initialize the pointer to memory where
                                numbers are stored

6003      MOV   R0, #0A        ; initialize the counter

6005      MOV   F0, #FF        ; minimum =FF

6008     AGAIN: MOVX A, @DPTR   ; get the number from the memory

6009     CJNE A, F0, 02         ; NE = 600E – 600C=02, compare number with
                                 minimum

600C     AJMP 6012              ; address of SKIP = 6012, if equal go to SKIP

600E      NE: JNC 02              ; SKIP = 6012- 6010, if not equal check for
                                  carry, if carry go to SKIP

6010     MOV F0,A             ; otherwise minimum  = number

6012     SKIP: INC DPTR           ; increment memory pointer

6013     DJNZ R0,F3           ; AGAIN = FF – (6013-6007), decrement

          count, if count = 0 stop, otherwise go to
                                  AGAIN
6015     RET
```

RESULT:

INPUT:

7000	08;	7003	05;	7006	04;	7009	05
7001	02;	7004	06;	7007	07;		
7002	03;	7005	01;	7008	19;		

OUTPUT:

B=01h

Exp. No. 8

ASSEMBLY LANGUAGE PROGRAM FOR REVERSE AND LOGICAL ‘OR’

AIM: Write a program for reverse the numbers and apply logic instruction OR gate to the given
numbers using 8051kit.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

PROGRAM:

```
MOV DPL, #34      ; instead of dpl, type 82
MOV DPH, #12     ; instead of dph, type 83
MOV A, DPL
RL A
RL A
RL A
RL A
MOV DPL, A
MOV A, DPH
RL A
RL A
RL A
RL A
MOV DPH, A
ORL A, DPL
RET
```

EXPECTED RESULT:

Logical ‘OR’ result for given numbers 43h & 21h is A = 63h

DPL= 43h

DPH =21h

Exp. No. 9

ASSEMBLY LANGUAGE PROGRAM FOR "JUMP" & "CALL" INSTRUCTIONS

AIM: (a) Write a ALP to find the sum of values 79h, F5h and E2h using "JUMP' instruction and load the sum in R_0 & R_6. (in 8051kit)

(b) Write a ALP to find the factorial of a given number using "CALL" & "RETURN" instructions.

APPARATUS:

1. MC 8051 trainer kit
2. SMPS

PROGRAM (A): USING "JUMP" INSTRUCTION

ADDRESS	LABEL	MNEUMONICS	COMMNETS
6000		MOV A, #00	Clear Accumulator
6002		MOV R5, A	Clear R5
6003		ADD A, #79	A = 0 + 79h = 79h
6005		JNC N1	If CY = 0, add next number
6007		INC R5	Else increment R5
6008	N1:	ADD A, #05	A = 79h + F5h = 6Eh and CY = 1
600A		JNC N2	If CY = 0, add next number
600C		INC R5	Cy = 1 , then increment R5
600D	N2:	ADD A, #0E2	A = 6Eh + E2h = 50h and CY = 1
600F		JNC	If CY = 0, copy result
6011		INC R5	If CY = 1, increment R5
6012	OVER:	MOV R0, A	Now, R0 = 50h & R6 = 02h
6013	HERE:	SJMP HERE	Halt the program

RESULT:

R0 = 50h & R6 = 02h

PROGRAM (B): USING "CALL" & "RETURN" INSTRUCTION

ADDRESS	LABEL	MNEUMONICS	COMMNETS
8100		MOV A, #05	Copy 05h to Register A
8102		MOV R0, A	Store 05h to Register R0
8103		CALL 9000	Call subprogram at 9000h
8106	HERE:	SJMP HERE	End main program
9000		CJNE R0, #01,9004	Compare and jump
9003		RET	Return to main program
9004		DEC R0	Decrement R0
9005		MOV F0, R0	Move R0 to register B
9007		MUL AB	Repeat multiplication
9008		JC 9000	
900B	AGAIN:	SJMP AGAIN	End subprogram

RESULT:

A = 78h (factorial of a number 05)

USING (KEIL Software) for 8051

Demo: (A) Program to find addition of two numbers.
(B) Program of Multibyte Addition

Demo (A)

ASSEMBLY LANGUAGE PROGRAM FOR ADDITION OF TWO NUMBERS

AIM: Write an assembly language program for adding two 8-bit numbers using keil Software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM:

```
MOV A, #05H
MOV B,#02H
ADD A,B
END
```

RESULT:

In accumulator, a= 7h

Demo (B):

ASSEMBLY LANGUAGE PROGRAM FOR MULTIBYTE ADDITION

AIM: Write an assembly language program for multibyte addition using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM:

```
MOV R0,#20H
MOV R1,#30H
MOV R3,#04H
CLR C
CLR A
AGAIN:   MOV A,@R0

         ADDC A,@R1

         MOV @R1,A

         INC R0

         INC R1

         DJNZ R3,AGAIN

      END
```

RESULT:

Inputs:

```
i: 0x20  --  01h, 02h, 03h, 04h
i: 0x30  --  05h, 06h, 07h, 08h
```

Output:

```
i: 0x30  --  06h, 08h, 0Ah, 0Ch
```

Exp. No. 10

ASSEMBLY LANGUAGE PROGRAM FOR ACTIVATING PORTS & GENERATION OF SQUARE WAVE

AIM: Write an assembly language program for generating square waveform using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM(1):

```
MOV SP,#7H
BACK: CLR P1.0
ACALL DELAY
SETB P1.0
ACALL DELAY
SJMP BACK
DELAY:MOV R1,#0FFH
AGAIN:DJNZ R1,AGAIN
RET
END
```

PROGRAM(2):

```
MOV SP,#7H                  ; initialize stack pointer
                              ; since we are using subroutine programe
BACK:MOV P1,#00H           ; send 00h on port 1 to generate
                            ; low level of square wave
ACALL DELAY                 ; wait for some time
MOV P1,#0FFH               ; send ffh on port 1 to generate
                            ; high level of square wave
ACALL DELAY                 ; wait for some time
SJMP BACK                   ; repeat the sequence
DELAY:MOV R1,#0FFH         ; load count
AGAIN:DJNZ R1,AGAIN         ; decrement count and repeat the process
                              ; until count is zero
RET                           ; return to main programe
```

EXPECTED RESULTS:

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
							✓
							✓

Program (1) : Activating Individual PORT1 pin 0

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
✓	✓	✓	✓	✓	✓	✓	✓
✓	✓	✓	✓	✓	✓	✓	✓

Program (2) : Activating PORT1

Exp. No. 11(A)(i)

ASSEMBLY LANGUAGE PROGRAM FOR ASCENDING ORDER OF A GIVEN NUMBERS

AIM: Write an assembly language program for arranging in ascending/descending order using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM FOR ASCENDING ORDER:

```
          MOV R0,#5              ; INITIALIZE COUNTER 1
AGAIN:     MOV DPTR,#2000H      ; initialize memory pointer
          MOV R1,#4              ; initialize counter 2
BACK:      MOV R2,DPL            ; save lower byte of memory address
           MOVX A,@DPTR          ; Get the num ber
           MOV B,A               ; Save the number
           INC DPTR              ; Increment the memory pointer
           MOVX A,@DPTR          ; Get the next number
           CJNE A,B,n            ; If not equal check for greater or less
           AJMP SKIP             ; Otherwise go to skip
    n:    JNC SKIP               ;If
           MOV DPL,R2            ;Exchange
           MOVX @DPTR,A
           INC DPTR
           MOV A,B
           MOVX @dptr,A
SKIP:      DJNZ R1,BACK          ;If R1 not equal to 0 go to BACK
          DJNZ R0,AGAIN          ;If R0 not equal to 0 go to AGAIN
```

RESULT:

Inputs:

x: 0x2000 -- 05h, 02h, 01h, 04h

Output:

x: 0x2000 -- 01h, 02h, 04h, 05h

Exp. No. 11(A)(ii)

ASSEMBLY LANGUAGE PROGRAM FOR DESCENDING ORDER OF A GIVEN NUMBERS

AIM: Write an assembly language program for arranging in ascending/descending order using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM FOR DESCENDING ORDER:

```
              MOV R0, #5             ; INITIALIZE COUNTER 1
AGAIN:          MOV DPTR,#2000H    ; initialize memory pointer
              MOV R1,#4                ; initialize counter 2
BACK:          MOV R2,DPL             ; save lower byte of memory address
               MOVX A,@DPTR         ; Get the num ber
               MOV B,A                  ; Save the number
               INC DPTR                 ; Increment the memory pointer
               MOVX A,@DPTR         ; Get the next number
               CJNE A,B,n               ; If not equal check for greater or less
               AJMP SKIP                ; Otherwise go to skip
      n:      JC SKIP                   ;If
               MOV DPL,R2              ;Exchange
              MOVX @DPTR,A
              INC DPTR
              MOV A,B
              MOVX @dptr,A
SKIP:        DJNZ R1,BACK                   ;If R1 not equal to 0 go to BACK
            DJNZ R0,AGAIN                  ;If R0 not equal to 0 go to AGAIN
```

RESULT:

Inputs:

```
x: 0x2000  --  05h, 02h, 01h, 04h
```

Output:

```
x: 0x2000  --  05h, 04h, 02h, 01h
```

Exp. No. 11(B)

ASSEMBLY LANGUAGE PROGRAM FOR DATA TRANSFER

AIM: Write an assembly language program for block move from one address to another address using keil software (AT89C51).

APPARATUS:

1. Keil software
2. P.C.

PROGRAM:

```
MOV R0,#20H
MOV R1,#30H
MOV R3,#10H
CLR A
AGAIN:MOV A,@R0
MOV @R1,A
INC R0
INC R1
DJNZ R3,AGAIN
END
```

RESULT:

Inputs:

i: 0x20 -- 01h, 02h, 03h, 04h,05h,06h,07h,08h,09h,0Ah

Output:

i: 0x30 -- 01h, 02h, 03h, 04h,05h,06h,07h,08h,09h,0Ah

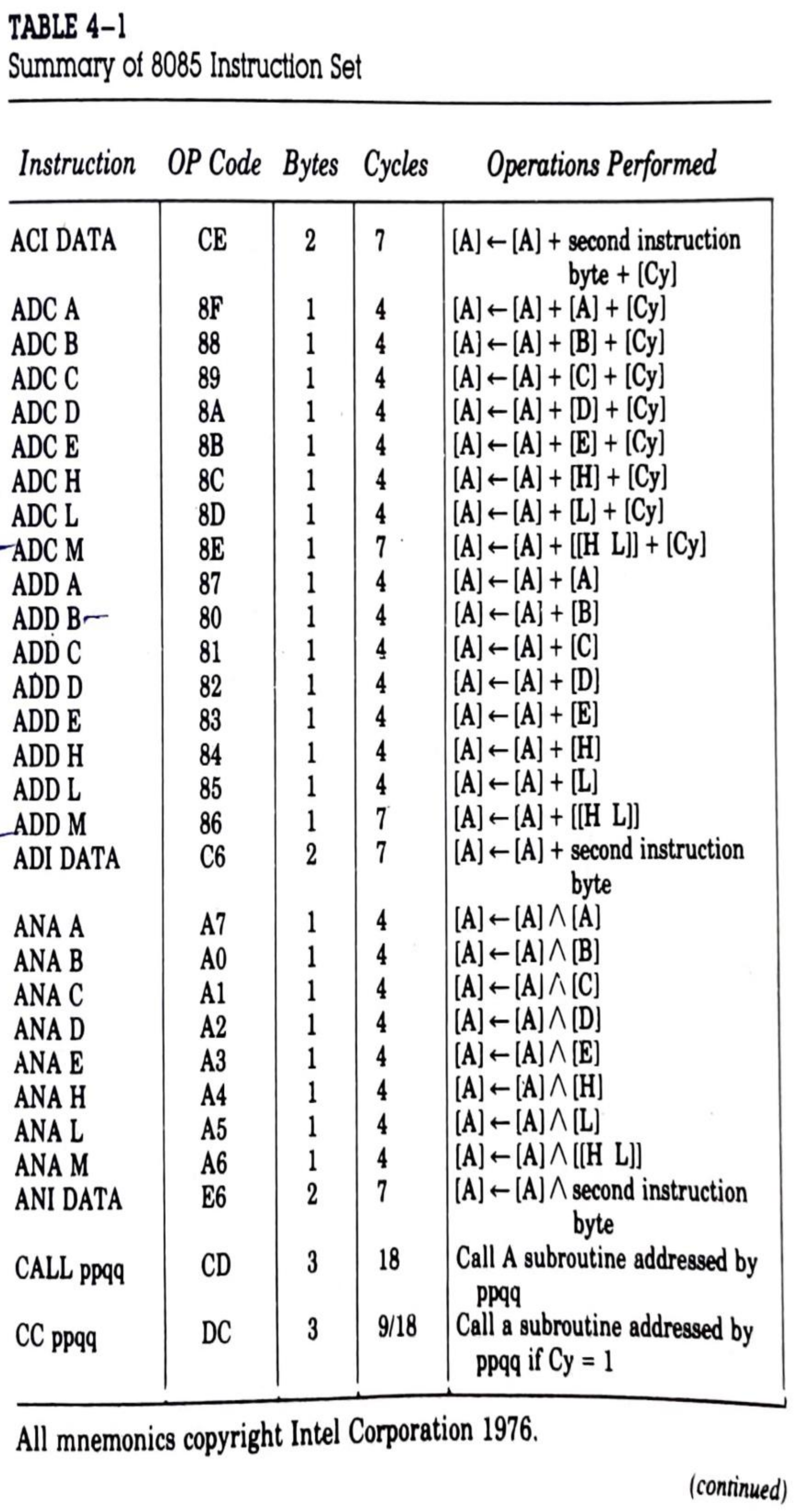


TABLE 4–1
Summary of 8085 Instruction Set

Instruction	*OP Code*	*Bytes*	*Cycles*	*Operations Performed*
ACI DATA	CE	2	7	[A] ← [A] + second instruction byte + [Cy]
ADC A	8F	1	4	[A] ← [A] + [A] + [Cy]
ADC B	88	1	4	[A] ← [A] + [B] + [Cy]
ADC C	89	1	4	[A] ← [A] + [C] + [Cy]
ADC D	8A	1	4	[A] ← [A] + [D] + [Cy]
ADC E	8B	1	4	[A] ← [A] + [E] + [Cy]
ADC H	8C	1	4	[A] ← [A] + [H] + [Cy]
ADC L	8D	1	4	[A] ← [A] + [L] + [Cy]
ADC M	8E	1	7	[A] ← [A] + [[H L]] + [Cy]
ADD A	87	1	4	[A] ← [A] + [A]
ADD B	80	1	4	[A] ← [A] + [B]
ADD C	81	1	4	[A] ← [A] + [C]
ADD D	82	1	4	[A] ← [A] + [D]
ADD E	83	1	4	[A] ← [A] + [E]
ADD H	84	1	4	[A] ← [A] + [H]
ADD L	85	1	4	[A] ← [A] + [L]
ADD M	86	1	7	[A] ← [A] + [[H L]]
ADI DATA	C6	2	7	[A] ← [A] + second instruction byte
ANA A	A7	1	4	[A] ← [A] ∧ [A]
ANA B	A0	1	4	[A] ← [A] ∧ [B]
ANA C	A1	1	4	[A] ← [A] ∧ [C]
ANA D	A2	1	4	[A] ← [A] ∧ [D]
ANA E	A3	1	4	[A] ← [A] ∧ [E]
ANA H	A4	1	4	[A] ← [A] ∧ [H]
ANA L	A5	1	4	[A] ← [A] ∧ [L]
ANA M	A6	1	4	[A] ← [A] ∧ [[H L]]
ANI DATA	E6	2	7	[A] ← [A] ∧ second instruction byte
CALL ppqq	CD	3	18	Call A subroutine addressed by ppqq
CC ppqq	DC	3	9/18	Call a subroutine addressed by ppqq if Cy = 1

All mnemonics copyright Intel Corporation 1976.

(continued)

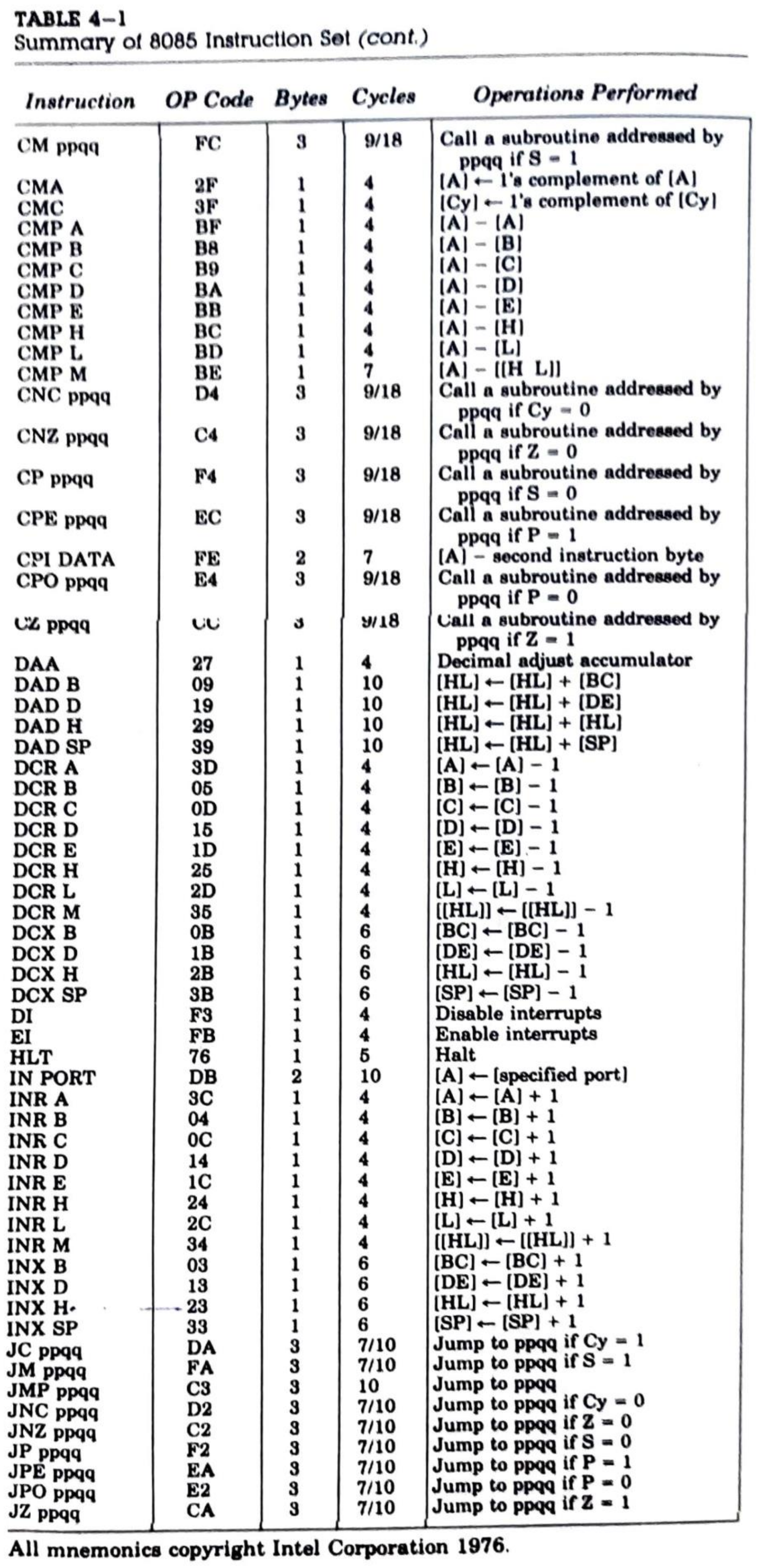


TABLE 4–1
Summary of 8085 Instruction Set *(cont.)*

Instruction	*OP Code*	*Bytes*	*Cycles*	*Operations Performed*
CM ppqq	FC	3	9/18	Call a subroutine addressed by ppqq if S = 1
CMA	2F	1	4	[A] ← 1's complement of [A]
CMC	3F	1	4	[Cy] ← 1's complement of [Cy]
CMP A	BF	1	4	[A] – [A]
CMP B	B8	1	4	[A] – [B]
CMP C	B9	1	4	[A] – [C]
CMP D	BA	1	4	[A] – [D]
CMP E	BB	1	4	[A] – [E]
CMP H	BC	1	4	[A] – [H]
CMP L	BD	1	4	[A] – [L]
CMP M	BE	1	7	[A] – [[H L]]
CNC ppqq	D4	3	9/18	Call a subroutine addressed by ppqq if Cy = 0
CNZ ppqq	C4	3	9/18	Call a subroutine addressed by ppqq if Z = 0
CP ppqq	F4	3	9/18	Call a subroutine addressed by ppqq if S = 0
CPE ppqq	EC	3	9/18	Call a subroutine addressed by ppqq if P = 1
CPI DATA	FE	2	7	[A] – second instruction byte
CPO ppqq	E4	3	9/18	Call a subroutine addressed by ppqq if P = 0
CZ ppqq	CC	3	9/18	Call a subroutine addressed by ppqq if Z = 1
DAA	27	1	4	Decimal adjust accumulator
DAD B	09	1	10	[HL] ← [HL] + [BC]
DAD D	19	1	10	[HL] ← [HL] + [DE]
DAD H	29	1	10	[HL] ← [HL] + [HL]
DAD SP	39	1	10	[HL] ← [HL] + [SP]
DCR A	3D	1	4	[A] ← [A] – 1
DCR B	05	1	4	[B] ← [B] – 1
DCR C	0D	1	4	[C] ← [C] – 1
DCR D	15	1	4	[D] ← [D] – 1
DCR E	1D	1	4	[E] ← [E] – 1
DCR H	25	1	4	[H] ← [H] – 1
DCR L	2D	1	4	[L] ← [L] – 1
DCR M	35	1	4	[[HL]] ← [[HL]] – 1
DCX B	0B	1	6	[BC] ← [BC] – 1
DCX D	1B	1	6	[DE] ← [DE] – 1
DCX H	2B	1	6	[HL] ← [HL] – 1
DCX SP	3B	1	6	[SP] ← [SP] – 1
DI	F3	1	4	Disable interrupts
EI	FB	1	4	Enable interrupts
HLT	76	1	5	Halt
IN PORT	DB	2	10	[A] ← [specified port]
INR A	3C	1	4	[A] ← [A] + 1
INR B	04	1	4	[B] ← [B] + 1
INR C	0C	1	4	[C] ← [C] + 1
INR D	14	1	4	[D] ← [D] + 1
INR E	1C	1	4	[E] ← [E] + 1
INR H	24	1	4	[H] ← [H] + 1
INR L	2C	1	4	[L] ← [L] + 1
INR M	34	1	4	[[HL]] ← [[HL]] + 1
INX B	03	1	6	[BC] ← [BC] + 1
INX D	13	1	6	[DE] ← [DE] + 1
INX H	23	1	6	[HL] ← [HL] + 1
INX SP	33	1	6	[SP] ← [SP] + 1
JC ppqq	DA	3	7/10	Jump to ppqq if Cy = 1
JM ppqq	FA	3	7/10	Jump to ppqq if S = 1
JMP ppqq	C3	3	10	Jump to ppqq
JNC ppqq	D2	3	7/10	Jump to ppqq if Cy = 0
JNZ ppqq	C2	3	7/10	Jump to ppqq if Z = 0
JP ppqq	F2	3	7/10	Jump to ppqq if S = 0
JPE ppqq	EA	3	7/10	Jump to ppqq if P = 1
JPO ppqq	E2	3	7/10	Jump to ppqq if P = 0
JZ ppqq	CA	3	7/10	Jump to ppqq if Z = 1

All mnemonics copyright Intel Corporation 1976.

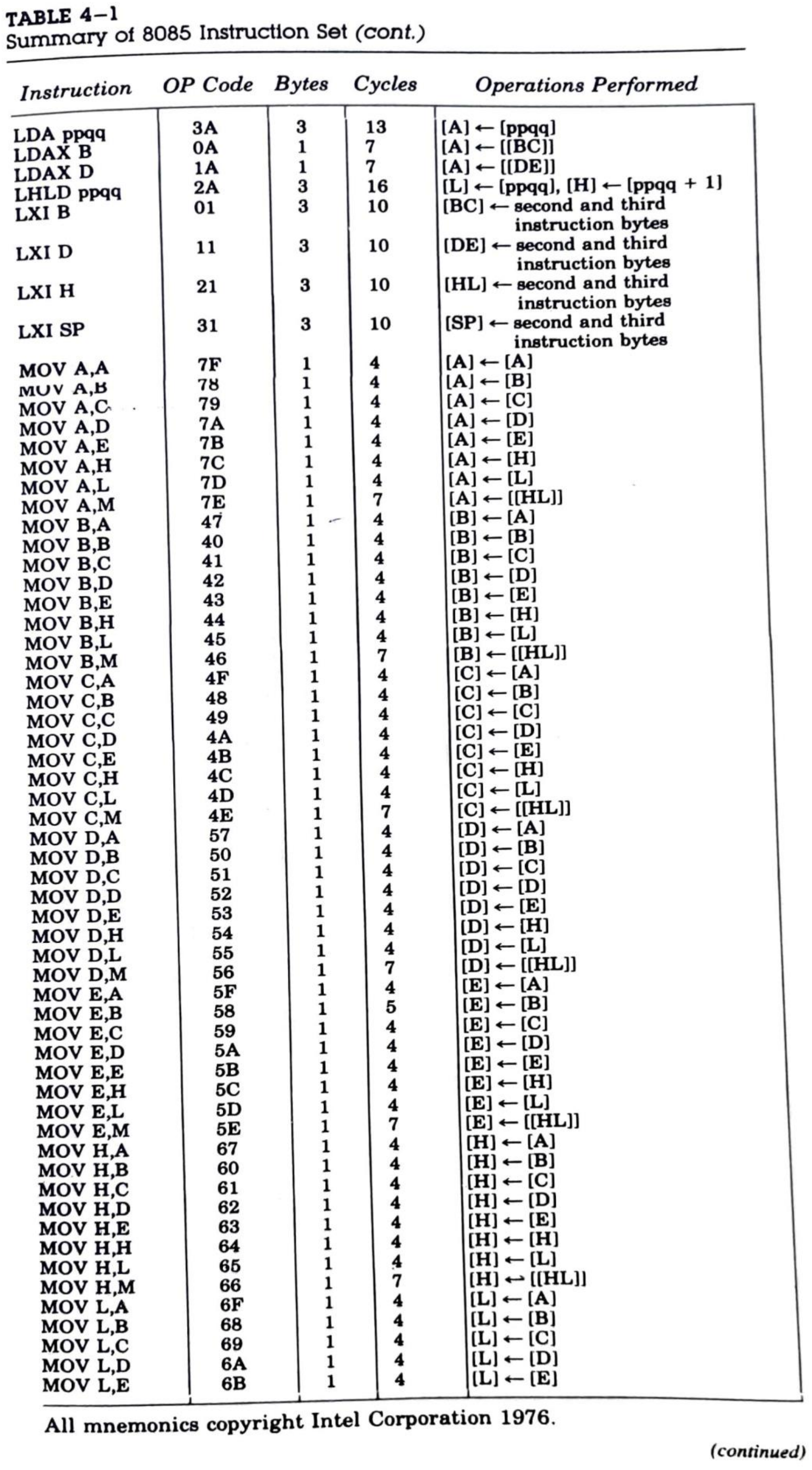


TABLE 4–1
Summary of 8085 Instruction Set *(cont.)*

Instruction	*OP Code*	*Bytes*	*Cycles*	*Operations Performed*
LDA ppqq	3A	3	13	[A] ← [ppqq]
LDAX B	0A	1	7	[A] ← [[BC]]
LDAX D	1A	1	7	[A] ← [[DE]]
LHLD ppqq	2A	3	16	[L] ← [ppqq], [H] ← [ppqq + 1]
LXI B	01	3	10	[BC] ← second and third instruction bytes
LXI D	11	3	10	[DE] ← second and third instruction bytes
LXI H	21	3	10	[HL] ← second and third instruction bytes
LXI SP	31	3	10	[SP] ← second and third instruction bytes
MOV A,A	7F	1	4	[A] ← [A]
MOV A,B	78	1	4	[A] ← [B]
MOV A,C	79	1	4	[A] ← [C]
MOV A,D	7A	1	4	[A] ← [D]
MOV A,E	7B	1	4	[A] ← [E]
MOV A,H	7C	1	4	[A] ← [H]
MOV A,L	7D	1	4	[A] ← [L]
MOV A,M	7E	1	7	[A] ← [[HL]]
MOV B,A	47	1	4	[B] ← [A]
MOV B,B	40	1	4	[B] ← [B]
MOV B,C	41	1	4	[B] ← [C]
MOV B,D	42	1	4	[B] ← [D]
MOV B,E	43	1	4	[B] ← [E]
MOV B,H	44	1	4	[B] ← [H]
MOV B,L	45	1	4	[B] ← [L]
MOV B,M	46	1	7	[B] ← [[HL]]
MOV C,A	4F	1	4	[C] ← [A]
MOV C,B	48	1	4	[C] ← [B]
MOV C,C	49	1	4	[C] ← [C]
MOV C,D	4A	1	4	[C] ← [D]
MOV C,E	4B	1	4	[C] ← [E]
MOV C,H	4C	1	4	[C] ← [H]
MOV C,L	4D	1	4	[C] ← [L]
MOV C,M	4E	1	7	[C] ← [[HL]]
MOV D,A	57	1	4	[D] ← [A]
MOV D,B	50	1	4	[D] ← [B]
MOV D,C	51	1	4	[D] ← [C]
MOV D,D	52	1	4	[D] ← [D]
MOV D,E	53	1	4	[D] ← [E]
MOV D,H	54	1	4	[D] ← [H]
MOV D,L	55	1	4	[D] ← [L]
MOV D,M	56	1	7	[D] ← [[HL]]
MOV E,A	5F	1	4	[E] ← [A]
MOV E,B	58	1	5	[E] ← [B]
MOV E,C	59	1	4	[E] ← [C]
MOV E,D	5A	1	4	[E] ← [D]
MOV E,E	5B	1	4	[E] ← [E]
MOV E,H	5C	1	4	[E] ← [H]
MOV E,L	5D	1	4	[E] ← [L]
MOV E,M	5E	1	7	[E] ← [[HL]]
MOV H,A	67	1	4	[H] ← [A]
MOV H,B	60	1	4	[H] ← [B]
MOV H,C	61	1	4	[H] ← [C]
MOV H,D	62	1	4	[H] ← [D]
MOV H,E	63	1	4	[H] ← [E]
MOV H,H	64	1	4	[H] ← [H]
MOV H,L	65	1	4	[H] ← [L]
MOV H,M	66	1	7	[H] ↔ [[HL]]
MOV L,A	6F	1	4	[L] ← [A]
MOV L,B	68	1	4	[L] ← [B]
MOV L,C	69	1	4	[L] ← [C]
MOV L,D	6A	1	4	[L] ← [D]
MOV L,E	6B	1	4	[L] ← [E]

All mnemonics copyright Intel Corporation 1976.

(continued)

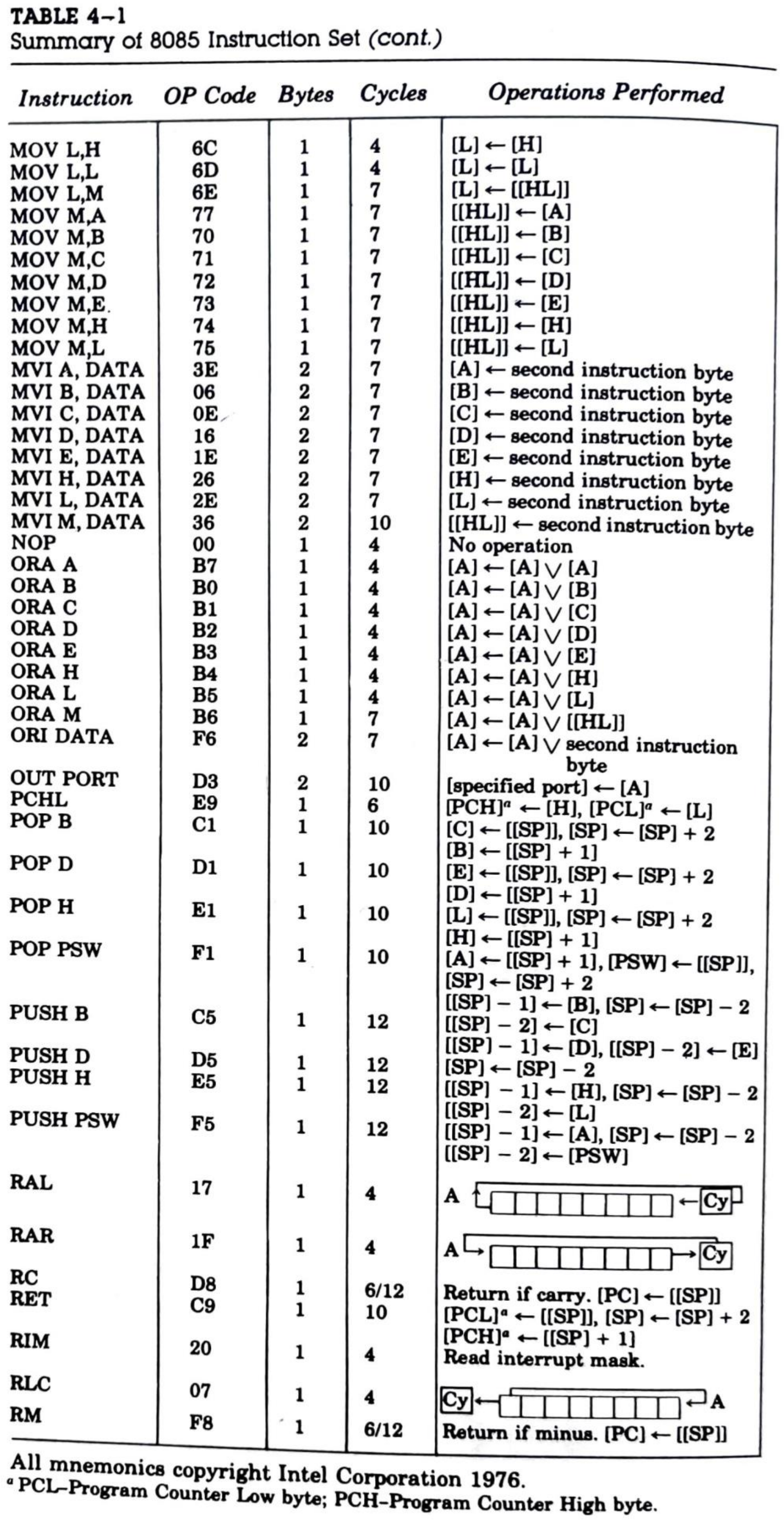
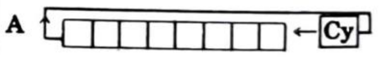
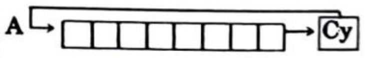
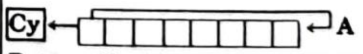
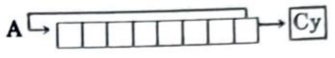


TABLE 4–1
Summary of 8085 Instruction Set *(cont.)*

Instruction	*OP Code*	*Bytes*	*Cycles*	*Operations Performed*
MOV L,H	6C	1	4	[L] ← [H]
MOV L,L	6D	1	4	[L] ← [L]
MOV L,M	6E	1	7	[L] ← [[HL]]
MOV M,A	77	1	7	[[HL]] ← [A]
MOV M,B	70	1	7	[[HL]] ← [B]
MOV M,C	71	1	7	[[HL]] ← [C]
MOV M,D	72	1	7	[[HL]] ← [D]
MOV M,E	73	1	7	[[HL]] ← [E]
MOV M,H	74	1	7	[[HL]] ← [H]
MOV M,L	75	1	7	[[HL]] ← [L]
MVI A, DATA	3E	2	7	[A] ← second instruction byte
MVI B, DATA	06	2	7	[B] ← second instruction byte
MVI C, DATA	0E	2	7	[C] ← second instruction byte
MVI D, DATA	16	2	7	[D] ← second instruction byte
MVI E, DATA	1E	2	7	[E] ← second instruction byte
MVI H, DATA	26	2	7	[H] ← second instruction byte
MVI L, DATA	2E	2	7	[L] ← second instruction byte
MVI M, DATA	36	2	10	[[HL]] ← second instruction byte
NOP	00	1	4	No operation
ORA A	B7	1	4	[A] ← [A] ∨ [A]
ORA B	B0	1	4	[A] ← [A] ∨ [B]
ORA C	B1	1	4	[A] ← [A] ∨ [C]
ORA D	B2	1	4	[A] ← [A] ∨ [D]
ORA E	B3	1	4	[A] ← [A] ∨ [E]
ORA H	B4	1	4	[A] ← [A] ∨ [H]
ORA L	B5	1	4	[A] ← [A] ∨ [L]
ORA M	B6	1	7	[A] ← [A] ∨ [[HL]]
ORI DATA	F6	2	7	[A] ← [A] ∨ second instruction byte
OUT PORT	D3	2	10	[specified port] ← [A]
PCHL	E9	1	6	[PCH][a] ← [H], [PCL][a] ← [L]
POP B	C1	1	10	[C] ← [[SP]], [SP] ← [SP] + 2 [B] ← [[SP] + 1]
POP D	D1	1	10	[E] ← [[SP]], [SP] ← [SP] + 2 [D] ← [[SP] + 1]
POP H	E1	1	10	[L] ← [[SP]], [SP] ← [SP] + 2 [H] ← [[SP] + 1]
POP PSW	F1	1	10	[A] ← [[SP] + 1], [PSW] ← [[SP]], [SP] ← [SP] + 2
PUSH B	C5	1	12	[[SP] − 1] ← [B], [SP] ← [SP] − 2 [[SP] − 2] ← [C]
PUSH D	D5	1	12	[[SP] − 1] ← [D], [[SP] − 2] ← [E] [SP] ← [SP] − 2
PUSH H	E5	1	12	[[SP] − 1] ← [H], [SP] ← [SP] − 2 [[SP] − 2] ← [L]
PUSH PSW	F5	1	12	[[SP] − 1] ← [A], [SP] ← [SP] − 2 [[SP] − 2] ← [PSW]
RAL	17	1	4	
RAR	1F	1	4	
RC	D8	1	6/12	Return if carry. [PC] ← [[SP]]
RET	C9	1	10	[PCL][a] ← [[SP]], [SP] ← [SP] + 2 [PCH][a] ← [[SP] + 1]
RIM	20	1	4	Read interrupt mask.
RLC	07	1	4	
RM	F8	1	6/12	Return if minus. [PC] ← [[SP]]

All mnemonics copyright Intel Corporation 1976.
[a] PCL–Program Counter Low byte; PCH–Program Counter High byte.

TABLE 4–1
Summary of 8085 Instruction Set *(cont.)*

Instruction	*OP Code*	*Bytes*	*Cycles*	*Operations Performed*
RNC	D0	1	6/12	Return if no carry. [PC] ← [[SP]]
RNZ	C0	1	6/12	Return if result not zero. [PC] ← [[SP]]
RP	F0	1	6/12	Return if positive. [PC] ← [[SP]], [SP] ← [SP] + 2
RPE	E8	1	6/12	Return if parity even. [PC] ← [[SP]], [SP] ← [SP] + 2
RPO	E0	1	6/12	Return if parity odd. [PC] ← [[SP]], [SP] ← [SP] + 2
RRC	0F	1	4	
RST0	C7	1	12	Restart
RST1	CF	1	12	Restart
RST2	D7	1	12	Restart
RST3	DF	1	12	Restart
RST4	E7	1	12	Restart
RST5	EF	1	12	Restart
RST6	F7	1	12	Restart
RST7	FF	1	12	Restart
RZ	C8	1	6/12	Return if zero. [PC] ← [[SP]]
SBB A	9F	1	4	[A] ← [A] − [A] − [Cy]
SBB B	98	1	4	[A] ← [A] − [B] − [Cy]
SBB C	99	1	4	[A] ← [A] − [C] − [Cy]
SBB D	9A	1	4	[A] ← [A] − [D] − [Cy]
SBB E	9B	1	4	[A] ← [A] − [E] − [Cy]
SBB H	9C	1	4	[A] ← [A] − [H] − [Cy]
SBB L	9D	1	4	[A] ← [A] − [L] − [Cy]
SBB M	9E	1	7	[A] ← [A] − [[HL]] − [Cy]
SBI DATA	DE	2	7	[A] ← [A] − second instruction byte − [Cy]
SHLD ppqq	22	3	16	[ppqq] ← [L], [ppqq + 1] ← [H]
SIM	30	1	4	Set interrupt mask
SPHL	F9	1	6	[SP] ← [HL]
STA ppqq	32	3	13	[ppqq] ← [A]
STAX B	02	1	7	[[BC]] ← [A]
STAX D	12	1	7	[[DE]] ← [A]
STC	37	1	4	[Cy] ← 1
SUB A	97	1	4	[A] ← [A] − [A]
SUB B	90	1	4	[A] ← [A] − [B]
SUB C	91	1	4	[A] ← [A] − [C]
SUB D	92	1	4	[A] ← [A] − [D]
SUB E	93	1	4	[A] ← [A] − [E]
SUB H	94	1	4	[A] ← [A] − [H]
SUB L	95	1	4	[A] ← [A] − [L]
SUB M	96	1	7	[A] ← [A] − [[HL]]
SUI DATA	D6	2	7	[A] ← [A] − second instruction byte
XCHG	EB	1	4	[D] ↔ [H], [E] ↔ [L]
XRA A	AF	1	4	[A] ← [A] ⊻ [A]
XRA B	A8	1	4	[A] ← [A] ⊻ [B]
XRA C	A9	1	4	[A] ← [A] ⊻ [C]
XRA D	AA	1	4	[A] ← [A] ⊻ [D]
XRA E	AB	1	4	[A] ← [A] ⊻ [E]
XRA H	AC	1	4	[A] ← [A] ⊻ [H]
XRA L	AD	1	4	[A] ← [A] ⊻ [L]
XRA M	AE	1	7	[A] ← [A] ⊻ [[HL]]
XRI DATA	EE	2	7	[A] ← [A] ⊻ second instruction byte
XTHL	E3	1	16	[[SP]] ↔ [L], [[SP] + 1] ↔ [H]

*PCL—program counter low byte; PCH—program counter high byte.
⊻ or ⊕ may be used to represent Exclusive-OR operation.
All mnemonics copyright Intel Corporation 1976.

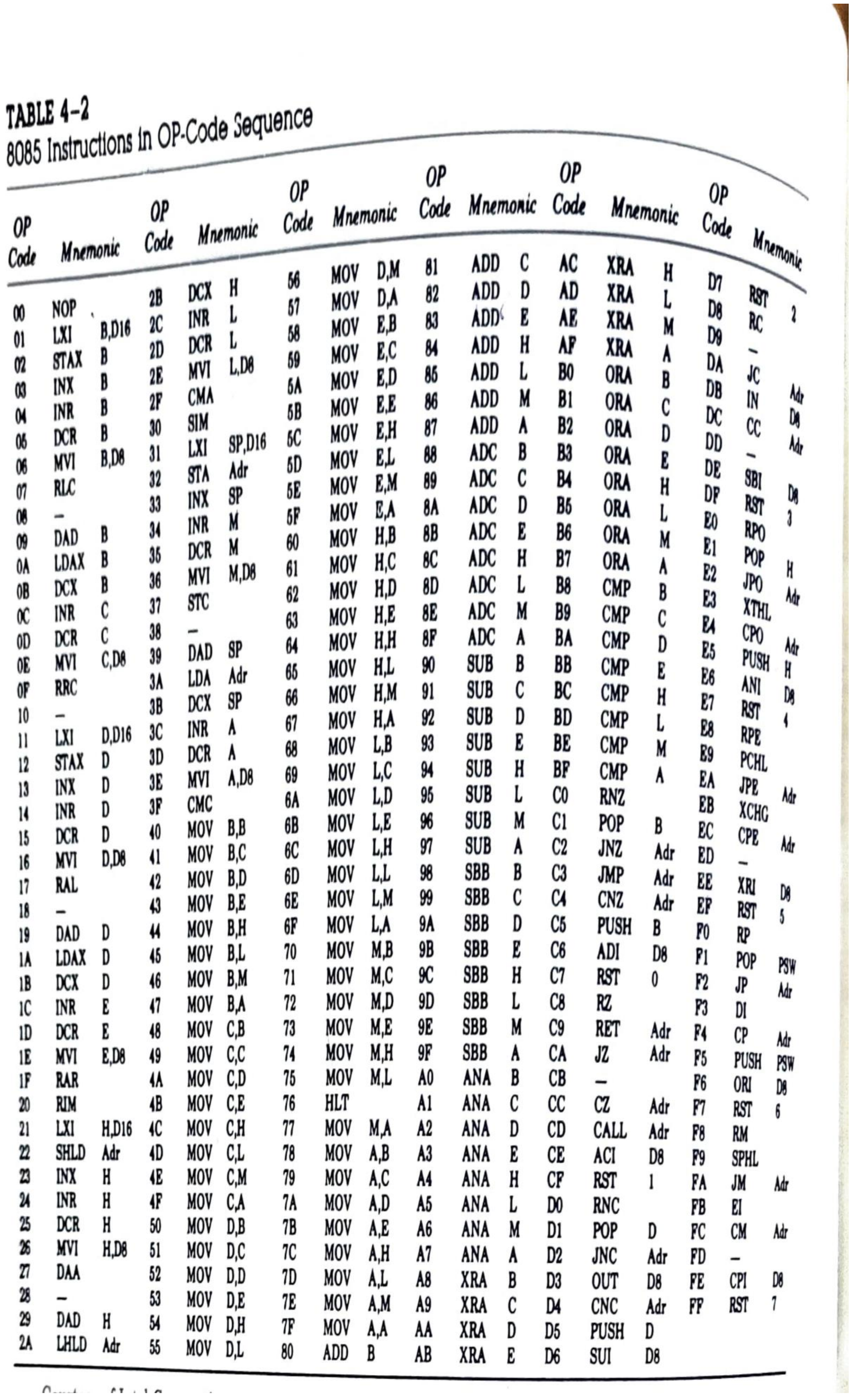


TABLE 4–2
8085 Instructions in OP-Code Sequence

OP Code	Mnemonic		OP Code	Mnemonic		OP Code	Mnemonic		OP Code	Mnemonic		OP Code	Mnemonic		OP Code	Mnemonic	
00	NOP		2B	DCX	H	56	MOV	D,M	81	ADD	C	AC	XRA	H	D7	RST	2
01	LXI	B,D16	2C	INR	L	57	MOV	D,A	82	ADD	D	AD	XRA	L	D8	RC	
02	STAX	B	2D	DCR	L	58	MOV	E,B	83	ADD	E	AE	XRA	M	D9	–	
03	INX	B	2E	MVI	L,D8	59	MOV	E,C	84	ADD	H	AF	XRA	A	DA	JC	Adr
04	INR	B	2F	CMA		5A	MOV	E,D	85	ADD	L	B0	ORA	B	DB	IN	D8
05	DCR	B	30	SIM		5B	MOV	E,E	86	ADD	M	B1	ORA	C	DC	CC	Adr
06	MVI	B,D8	31	LXI	SP,D16	5C	MOV	E,H	87	ADD	A	B2	ORA	D	DD	–	
07	RLC		32	STA	Adr	5D	MOV	E,L	88	ADC	B	B3	ORA	E	DE	SBI	D8
08	–		33	INX	SP	5E	MOV	E,M	89	ADC	C	B4	ORA	H	DF	RST	3
09	DAD	B	34	INR	M	5F	MOV	E,A	8A	ADC	D	B5	ORA	L	E0	RPO	
0A	LDAX	B	35	DCR	M	60	MOV	H,B	8B	ADC	E	B6	ORA	M	E1	POP	H
0B	DCX	B	36	MVI	M,D8	61	MOV	H,C	8C	ADC	H	B7	ORA	A	E2	JPO	Adr
0C	INR	C	37	STC		62	MOV	H,D	8D	ADC	L	B8	CMP	B	E3	XTHL	
0D	DCR	C	38	–		63	MOV	H,E	8E	ADC	M	B9	CMP	C	E4	CPO	Adr
0E	MVI	C,D8	39	DAD	SP	64	MOV	H,H	8F	ADC	A	BA	CMP	D	E5	PUSH	H
0F	RRC		3A	LDA	Adr	65	MOV	H,L	90	SUB	B	BB	CMP	E	E6	ANI	D8
10	–		3B	DCX	SP	66	MOV	H,M	91	SUB	C	BC	CMP	H	E7	RST	4
11	LXI	D,D16	3C	INR	A	67	MOV	H,A	92	SUB	D	BD	CMP	L	E8	RPE	
12	STAX	D	3D	DCR	A	68	MOV	L,B	93	SUB	E	BE	CMP	M	E9	PCHL	
13	INX	D	3E	MVI	A,D8	69	MOV	L,C	94	SUB	H	BF	CMP	A	EA	JPE	Adr
14	INR	D	3F	CMC		6A	MOV	L,D	95	SUB	L	C0	RNZ		EB	XCHG	
15	DCR	D	40	MOV	B,B	6B	MOV	L,E	96	SUB	M	C1	POP	B	EC	CPE	Adr
16	MVI	D,D8	41	MOV	B,C	6C	MOV	L,H	97	SUB	A	C2	JNZ	Adr	ED	–	
17	RAL		42	MOV	B,D	6D	MOV	L,L	98	SBB	B	C3	JMP	Adr	EE	XRI	D8
18	–		43	MOV	B,E	6E	MOV	L,M	99	SBB	C	C4	CNZ	Adr	EF	RST	5
19	DAD	D	44	MOV	B,H	6F	MOV	L,A	9A	SBB	D	C5	PUSH	B	F0	RP	
1A	LDAX	D	45	MOV	B,L	70	MOV	M,B	9B	SBB	E	C6	ADI	D8	F1	POP	PSW
1B	DCX	D	46	MOV	B,M	71	MOV	M,C	9C	SBB	H	C7	RST	0	F2	JP	Adr
1C	INR	E	47	MOV	B,A	72	MOV	M,D	9D	SBB	L	C8	RZ		F3	DI	
1D	DCR	E	48	MOV	C,B	73	MOV	M,E	9E	SBB	M	C9	RET	Adr	F4	CP	Adr
1E	MVI	E,D8	49	MOV	C,C	74	MOV	M,H	9F	SBB	A	CA	JZ	Adr	F5	PUSH	PSW
1F	RAR		4A	MOV	C,D	75	MOV	M,L	A0	ANA	B	CB	–		F6	ORI	D8
20	RIM		4B	MOV	C,E	76	HLT		A1	ANA	C	CC	CZ	Adr	F7	RST	6
21	LXI	H,D16	4C	MOV	C,H	77	MOV	M,A	A2	ANA	D	CD	CALL	Adr	F8	RM	
22	SHLD	Adr	4D	MOV	C,L	78	MOV	A,B	A3	ANA	E	CE	ACI	D8	F9	SPHL	
23	INX	H	4E	MOV	C,M	79	MOV	A,C	A4	ANA	H	CF	RST	1	FA	JM	Adr
24	INR	H	4F	MOV	C,A	7A	MOV	A,D	A5	ANA	L	D0	RNC		FB	EI	
25	DCR	H	50	MOV	D,B	7B	MOV	A,E	A6	ANA	M	D1	POP	D	FC	CM	Adr
26	MVI	H,D8	51	MOV	D,C	7C	MOV	A,H	A7	ANA	A	D2	JNC	Adr	FD	–	
27	DAA		52	MOV	D,D	7D	MOV	A,L	A8	XRA	B	D3	OUT	D8	FE	CPI	D8
28	–		53	MOV	D,E	7E	MOV	A,M	A9	XRA	C	D4	CNC	Adr	FF	RST	7
29	DAD	H	54	MOV	D,H	7F	MOV	A,A	AA	XRA	D	D5	PUSH	D			
2A	LHLD	Adr	55	MOV	D,L	80	ADD	B	AB	XRA	E	D6	SUI	D8			