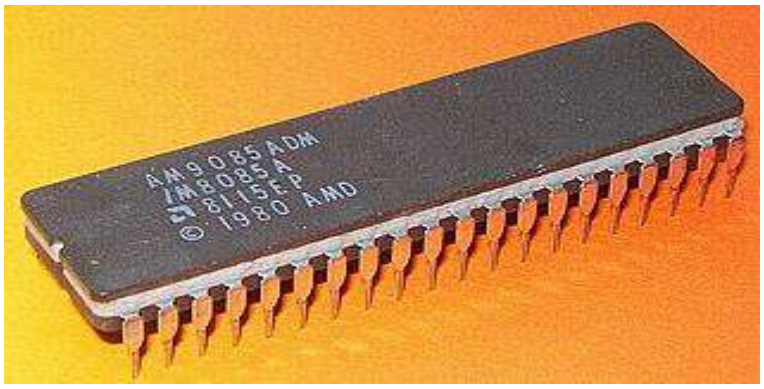
SUB. CODE: 18KP3PELP4

SUB. TITLE: MICROPROCESSOR AND MICROCONTROLLER



UNIT – I

Microprocessor architecture and Instruction set

MICROPROCESSOR ARCHITECTURE

INTRODUCTION

The microprocessor(s) is the central processing unit (CPU) of a computer. It is the heart of the computer. This chapter describes Intel 8085 as it is one of the most popular 8-bit microprocessors. The Intel Corporation has also developed a large number of general purpose and special purpose peripheral devices. These devices are very useful for the development of microprocessor-based system. The availability of a variety of support devices is also one of the causes of popularity of Intel microprocessors.

3.1 INTEL 8085

Intel 8085 is an 8-bit, NMOS microprocessor. It is a 40 pin I.C. package fabricated on a single LSI chip. The Intel 8085 uses a single $+5V_{d.c}$ supply for its operation. Its clock speed is about 3 MHz. The clock cycle is of 320 ns. The time for the clock cycle of the Intel 8085AH-2, version is 200 ns. It has 80 basic instructions and 246 opcodes. Fig. 3.1 shows the block diagram of Intel 8085. It consists of three main sections: an arithmetic and logic unit, a timing and control unit and a set of registers. These important sections are described in the subsequent sections.

3.1.1 ALU

The arithmetic and logic unit, ALU, performs the following arithmetic and logical operations:

(i) Addition
(ii) Subtraction
(iii) Logical AND
(iv) Logical OR
(v) Logical EXCLUSIVE OR
(vi) Complement (logical NOT)
(vii) Increment (add 1)
(viii) Decrement (subtract 1)
(ix) Left shift, Rotate left, Rotate right
(x) Clear etc.

3.1.2 Timing and Control Unit

The timing and control unit is a section of the CPU. It generates timing and control signals which are necessary for the execution of instructions. It controls data flow between CPU and peripherals (including memory). It provides status, control and timing signals which are required for the operation of memory and I/O devices. It controls the entire operations of the microprocessor and peripherals connected to it. Thus, it is seen that the control unit of the CPU acts as the brain of the computer system.

3.1.3 Registers

Figure 3.1 shows the various registers of Intel 8085. Registers are used by the micro-processor for temporary storage and manipulation of data and instructions. Data remain in the registers till they are sent to the memory or I/O devices. In a large computer the number of registers is more and hence the program requires less transfer of data to/from the memory. In a small computer the number of registers is small due to limited size of the chip. Intel 8085 microprocessor has the following registers:

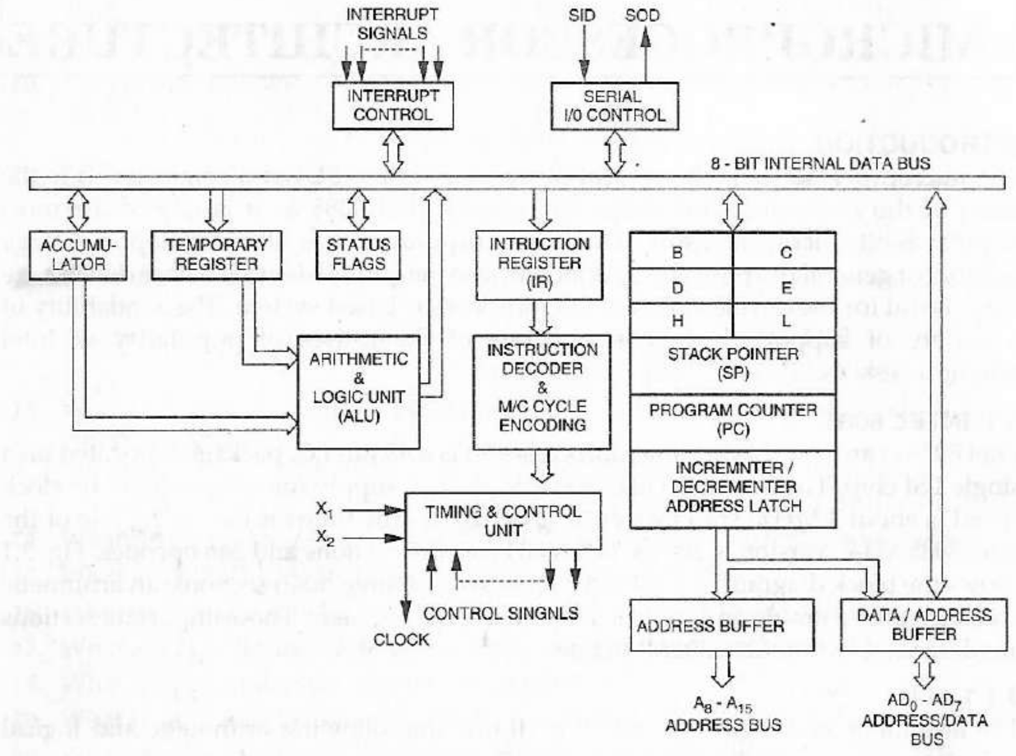


Fig. 3.1 Block diagram of Intel 8085

(i) One 8-bit accumulator (ACC) *i.e.* register *A*
(ii) Six 8-bit general purpose registers. These are *B*, *C*, *D*, *E*, *H* and *L*
(iii) One 16-bit stack pointer, *SP*
(iv) One 16-bit program counter, *PC*
(v) Instruction register
(vi) Temporary register

In addition to the above mentioned registers the 8085 microprocessor contains a set of five flip-flops which serve as flags (or status flags). A flag (or status flag) is a flip-flop which indicates some condition which arises after the execution of an arithmetic or logical instruction.

Accumulator (ACC). The accumulator is an 8-bit register associated with the ALU. The register 'A' in the 8085 is an accumulator. It is used to hold one of the operands of an arithmetic or logical operation. It serves as one input to the ALU. The other operand for an arithmetic or logical operation may be stored either in the memory or in one of the general-purpose registers. The final result of an arithmetic or logical operation is placed in the accumulator.

The above descriptions are true for general cases, not for some typical or exceptional cases. For example, there are some logical instructions which need only one operand. It is held in the accumulator. The result is placed in the accumulator. Such instructions do not require any other register or memory location because there is no other operand. There is one typical instruction DAD rp, for 16-bit addition for which one of the 16-bit operands is kept in H-L pair and the other in the B-C or D-E pair. The result is placed in the H-L pair. See details for such cases in Chapter 4.

General-Purpose Registers. The 8085 microprocessor contains six 8-bit general-purpose registers. They are: B, C, D, E, H and L register. To hold 16-bit data a combination of two 8-bit registers can be employed. The combination of two 8-bit registers is known as a **register-pair.** The valid register pairs in the 8085 are: B-C, D-E and H-L. The programmer can not form a register-pair by selecting any two registers of his choice. The H-L pair is used to act as memory pointer and for this purpose it holds the 16-bit address of a memory location. The general-purpose registers and the accumulator are accessible to programmer. He can store data in these registers during writing his program.

Program Counter (PC). It is a 16-bit special-purpose register. It is used to hold the memory address of the next instruction to be executed. It keeps the track of memory addresses of the instructions in a program while they are being executed. The microprocessor increments the content of the program counter during the execution of an instruction so that it points to the address of the next instruction in the program at the end of the execution of an instruction.

Stack Pointer (SP). It is a 16-bit special function register. The **stack** is a sequence of memory locations set aside by a programmer to store/retrieve the contents of accumulator, flags, program counter and general-purpose registers during the execution of a program. Any portion of the memory can be used as stack. Since, the stack works on LIFO (last-in-first-out) principle, its operation is faster compared normal store/retrieve of memory locations. During the execution of a program sometimes it becomes necessary to save the contents of some registers which are needed for some other operations in the subsequent steps of the program. The contents of such registers are saved in the stack. Then the registers are used for some other operations. After completing the needed operations the contents which were saved in the stack are brought back to the registers. The contents of only those registers are saved, which are needed in the later part of the program. The stack pointer (SP) controls addressing of the stack. The SP holds the address of the top element of data stored in the stack.

The stack is defined and the stack pointer is initialized by the programmer at the beginning of a program which needs stack operation. Stack is also used by the microprocessor. For example, it stores the contents of program counter when it jumps to a subroutine using CALL instruction. Stack has been described in details in Chapter 5.

Instruction Register. The instruction register holds the opcode (operation code or instruction code) of the instruction which is being decoded and executed.

Temporary Register. It is an 8-bit register associated with the ALU. It holds data during an arithmetic/logical operation. It is used by the microprocessor. It is not accessible to programmer.

Flags. The Intel 8085 microprocessor contains five flip-flops to serve as status flags. The flip-flops are set or reset according to the conditions which arise during an arithmetic or logical operation.

The five status flags of Intel 8085 are:

(i) Carry Flag (CS)

(ii) Parity Flag (P)
(iii) Auxiliary Carry Flag (AC)
(iv) Zero Flag (Z)
(v) Sign Flag (S)

If a flip-flop for a particular flag is set, it indicates 1. When it is reset, it indicates 0.

Carry Flag (CS). After the execution of an arithmetic instruction if a carry is produced, the carry flag CS is set to 1, otherwise it is 0. The carry flag is set or reset in case of addition as well as subtraction. After the addition of two 8-bit numbers, if the sum is larger than 8 bits, a carry is produced; and the carry flag is set to 1. In case of subtraction, if borrow occurs, the carry flag is set to 1. The carry flag holds carry out of the most significant bit resulting from the execution of an arithmetic operation.

Parity Flag (P). The parity status flag P is set to 1, if the result of an arithmetic or logical operation contains even number of 1s. It is reset i.e. it is 0, if the result contains odd number of 1s.

Auxiliary Carry Flag (AC). The auxiliary carry flag AC holds carry out of the bit number 3 to the bit number 4 resulting from the execution of an arithmetic operation. The counting of bits starts from 0, and hence, the Bit No. 3 is actually the fourth bit from the least significant bit [See Fig. 3.2(b)].

Zero Flag (Z). The zero status flag Z is set to 1, if the result of an arithmetic or logical operation is 0. If the result is not zero, the flag is set to 0.

Sign Flag (S). The sign flag S is set to 1, if the result of an arithmetic or logical operation is negative. If the result is positive, the sign flag is set to 0.

The sign flag has its significance only when signed arithmetic operation is performed. To represent a signed number the most significant bit is reserved by the programmer to represent the sign of a number. In other words the MSB is used as a sign bit. It represents the sign of the number. If a number is negative, the sign bit is 1. For a positive number, the sign bit is 0. In case of 8-bit signed operation, the remaining 7 bits are used to represent the magnitude of a number. After execution of signed arithmetic operation, the MSB of the result represents its sign. The sign flag acquires the value of the MSB of the result following the execution of signed arithmetic operation. Hence, it represents the sign of the result.

For unsigned arithmetic operation, all the 8 bits are used to represent the magnitude of the number. After the execution of an arithmetic operation, all the 8 bits of the result represent its magnitude. Therefore, the sign flag has no significance in unsigned arithmetic operation. Also, for logical operation sign bit has no significance. Since, the sign flag is set or reset according to the MSB of the result, it is set or reset on the value of MSB of the result of logical operation also.

The above discussion also holds good for signed arithmetic operation of 16-bit, 32-bit or more. In case of 16-bit operation 15 bits are used to represent the magnitude of a number and 1 bit to represent its sign. In case of 32-bit operation 31 bits are used to represent the magnitude and 1 bit to represent its sign.

Figure 3.2 (b) shows the status flags of ADD operation. Take an example of the instruction ADD B. The execution of the instruction ADD B will add the content of the register B to the content of the accumulator. Suppose, the contents of the accumulator and register B are CB and E9 respectively. Now, CB and E9 are added and the result is 01, B4. As the accumulator is an 8-bit register B4 remains in the accumulator and there is a carry. The various status flags are shown in Fig. 3.2 (b).

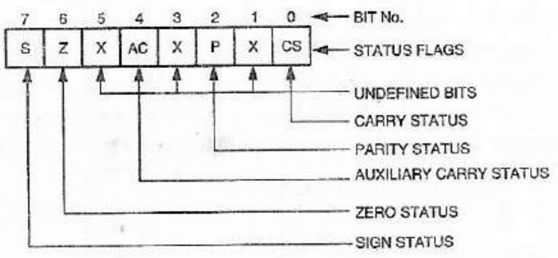


Fig. 3.2 (a) Status Flags of Intel 8085

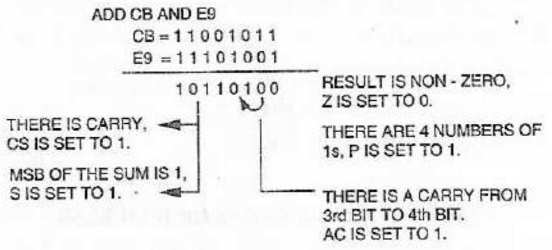

ADD CB AND E9
CB = 1 1 0 0 1 0 1 1
E9 = 1 1 1 0 1 0 0 1
1 0 1 1 0 1 0 0
RESULT IS NON - ZERO, Z IS SET TO 0.
THERE IS CARRY, CS IS SET TO 1.
THERE ARE 4 NUMBERS OF 1s, P IS SET TO 1.
MSB OF THE SUM IS 1, S IS SET TO 1.
THERE IS A CARRY FROM 3rd BIT TO 4th BIT. AC IS SET TO 1.

Fig. 3.2 (b) Status Flags for ADD operation

PSW. In Fig. 3.2 (a), five bits indicate the five status flags and three bits are undefined. The combination of these 8 bits is called **Program Status Word (PSW)**. PSW and the accumulator are treated as a 16-bit unit for stack operation.

3.1.4 Data and Address Bus

The Intel 8085 is an 8-bit microprocessor. Its data bus is 8-bit wide and hence, 8 bits of data can be transmitted in parallel from or to the microprocessor. The Intel 8085 requires a 16-bit wide address bus as the memory addresses are of 16 bits. The 8 most significant bits of the address are transmitted by the address bus, A-bus (pins A_8 to A_{15}). The 8 least significant bits of the address are transmitted by address/data bus, AD-bus (pins AD_0 - AD_7). The address/data bus transmits data and address at different moments. At a particular moment it transmits either data or address. Thus, the AD-bus operates in time shared mode. This technique is known as multiplexing. First of all 16-bit memory address is transmitted by the microprocessor; the 8 MSBs of the address on the A-bus and the 8 LSB_S of the address on AD-bus. Thus, the effective width of the address bus becomes 16-bit wide. Then the 8 LSBs of the address is latched either into the memory or external latch so that the complete 16-bit address remains available for further operation. The 8-bit AD-bus now becomes free, and it is available for data transmission. 2^{16} (= 65536 = 64 K, where 1 K = 1024) memory locations can be addressed directly by Intel 8085. Each memory location contains 1 byte of data.

3.1.5 Pin Configuration

Figure 3.3 shows the schematic diagram of Intel 8085. The description of various pins are as follows:

A_8 - A_{15} (output). These are address bus and are used for the most significant bits of the memory address or 8 bits of I/O address.

AD_0 - AD_7 (input/output). These are time multiplexed address/data bus *i.e.* they serve dual purpose. They are used for the least significant 8 bits of the memory address or I/O address during the first clock cycle of a machine cycle. Again they are used for data during second and third clock cycles.

ALE (output). It is an address latch enable signal. It goes high during first clock cycle of a machine cycle and enables the lower 8 bits of the address to be latched either into the memory or external latch.

IO/$\overline{\text{M}}$ (output). It is a status signal which distinguishes whether the address is for memory or I/O. When it goes high, the address on the address bus is for an I/O device. When it goes low, the address on the address bus is for a memory location.

S0, S_1 (output). These are status signals sent by the microproessor to distinguish the various types of operation given in Table 3.1.

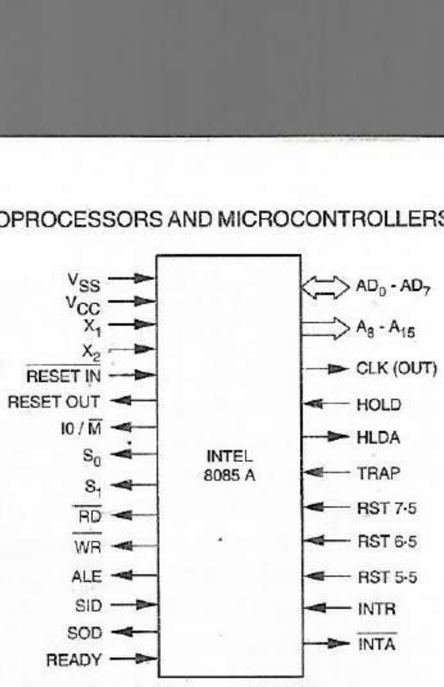


Fig. 3.3 Schematic Diagram of Intel 8085

Table 3.1 Status Codes for Intel 8085

S_1	S_0	*Operations*
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

$\overline{\text{RD}}$ (output). When microprocessor reads data or codes from a memory location or an input device, it is called READ operation. $\overline{\text{RD}}$ is a signal sent by the microprocessor to the memory/input device to control READ operation. When it goes low, the selected memory or input device is read.

$\overline{\text{WR}}$ (output). When microprocessor sends data to a memory location or an output device, it is called WRITE operation. $\overline{\text{WR}}$ is a signal sent by the microprocessor to the memory/output device to control WRITE operation. When it goes low, the data which is on the data bus, is written into the selected memory or sent to the output device.

READY (input). It is a signal sent by an input or output device to the microprocessor. This signal indicates that the input or output device is ready to send or receive data. The microprocessor examines READY signal before it performs data transfer operation. A slow input or output device is connected to the microprocessor through READY line. When READY is high, it indicates that the input or output device is ready to send or receive data. When READY is low, the microprocessor waits till READY becomes high. The microprocessor examines the status of READY signal in the second clock cycle of the machine cycle.

HOLD (input). When another device of the computer system, requires address and data buses for data transfer, it sends HOLD signal to the microprocessor. After receiving the HOLD request, the microprocessor sends out a HLDA (HOLD Acknowledge) signal to the device. Then the microprocessor leaves the control over the buses as soon as the current machine cycle is completed. Internal processing may continue. The microprocessor regains the control over the buses after the HOLD signal is removed.

HLDA (output). It is a HOLD acknowledge signal sent out by the microprocessor after receiving the HOLD signal. It is sent to the device which has issued the HOLD signal. After the removal of the HOLD signal, the HLDA goes low, and thereafter the microprocessor takes over the buses.

INTR (input). It is an interrupt signal sent by an external device to the microporcessor. Through this line an external device informs microprocessor that it is ready to transfer data or to initiate certain operation. The 8085 microprocessor has 5 interrupt lines. The INTR is one of them. When it goes high, the microprocessor suspends the execution of its normal sequence of instructions. After completing the current instruction at hand, it attends the interrupting device. The microprocessor issues an interrupt acknowledge signal $\overline{\text{INTA}}$. Then it transfers data or takes any other action as required.

$\overline{\text{INTA}}$ (output). It is an interrupt acknowledge signal issued by the microprocessor after receiving an interrupt request from an external device. It is a low active signal.

RST 5.5, 6.5, 7.5 and TRAP (inputs). These are interrupts. When an interrupt is recognised the next instruction is executed from a fixed location in the memory as given below:

Line	*Location from which next instruction is picked up*
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C

RST 7.5, RST 6.5 and RST 5.5 are the restart interrupts. Each of them has a programmable mask. The TRAP has the highest priority among interrupts. It is a nonmaskable interrupt. It is unaffected by any mask or interrupt enable. The order of priority of interrupts is as follows:

TRAP (highest priority)
RST 7.5
RST 6.5
RST 5.5
INTR (lowest priority).

RESET IN (input). It resets the program counter to zero. It also resets interrupt enable and HLDA flip-flops. It does not affect any other flag or register except the instruction register. The CPU is held in reset condition as long as RESET is applied.

RESET OUT (output). It indicates that the CPU is being reset.

X_1, X_2 (input). These are terminals to be connected to an external crystal oscillator which drives an internal circuitry of the microprocessor to produce a suitable clock for the operation of microprocessor.

CLK (output). It is a clock output for user, which can be used for other digital ICs. Its frequency is same at which processor operates.

SID (input). It is data line for serial input. The data on this line is loaded into the 7th bit of the accumulator when RIM instruction is executed.

SOD (output). It is a data line for serial output. The 7th bit of the accumulator is output on SOD line when SIM instruction is executed.

V_{CC}	+ 5 Volts supply
V_{SS}	ground reference

3.1.6 Intel 8085 Instructions

A computer receives data from the user, processes data and sends the result back to the user. The computer simply performs a given task on specified data in response to certain instructions. An instruction is a command given to the computer to perform a specified operation on given data.

PERIPHERAL DEVICES AND INTERFACING

7.1 INTRODUCTION

A microprocessor combined with memory and input/output devices, forms a microcomputer. The microprocessor is the heart of a microcomputer. Memories and input/output devices are interfaced to microprocessor to form a microcomputer. In case of large and minicomputers the memories and input/output devices are interfaced to CPU by the manufacturer. In a microprocessor-based system the designer has to select suitable memories and input/output devices for his task and interface them to the microprocessor. The selected memories and input/output devices should be compatible with microprocessor. If a particular device is not compatible, an additional electronic circuit has to be designed through which the device may be interfaced to the CPU.

7.2 ADDRESS SPACE PARTITIONING

The Intel 8085 uses a 16-bit wide address bus for addressing memories and I/O devices. Using 16-bit wide address bus it can access 2^{16} = 64 K bytes of memory and I/O devices. The 64 K addresses are to be assigned to memories and I/O devices for their addressing. There are two schemes for the allocation of addresses to memories and input/output devices:

1. Memory mapped I/O scheme
2. I/O Mapped I/O scheme

7.2.1 Memory Mapped I/O Scheme

In memory mapped I/O scheme there is only one address space. Address space is defined as the set of all possible addresses that a microprocessor can generate. Some addresses are assigned to memories and some addresses to I/O devices. An I/O device is also treated as a memory location and one address is assigned to it. Suppose that memory locations are assigned the addresses 2000 to 24FF. One address is assigned to each memory location. Any one of these addresses cannot be assigned to an I/O device. The addresses for I/O devices are different from the addresses which have been assigned to memories. The addresses which have not been assigned to memories can be assigned to I/O devices. For example, 2500, 2501, 2502 etc. may be assigned to I/O devices. One address is assigned to each I/O device.

In this scheme all the data transfer instructions of the microprocessor can be used for both memory as well as I/O devices. For example, MOV A, M will be valid for data transfer from the memory location or I/O device whose address is in H-L pair. If the H-L pair contains the address of a memory location, data will be transferred from the memory location to the accumulator. If the H-L pair contains the address of an I/O device, data will be moved from the I/O device to the accumulator. The memory mapped I/O scheme is suitable for a small system.

7.2.2 I/O Mapped I/O Scheme

In this scheme the addresses assigned to memory locations can also be assigned to I/O devices. Since, the same address may be assigned to a memory location or an I/O device, the microprocessor must issue a signal to distinguish whether the address on the address bus is for a memory location or an I/O device. The Intel 8085 issues an IO/$\overline{\text{M}}$ signal for this purpose. When this signal is high the address on the address bus is for an I/O device. When this signal is low, the address on the address bus is for a memory location. Two extra instructions IN and OUT are used to address I/O devices. The IN instruction is used to read to data of an input device. The OUT instruction is used to send data to an output device. This scheme is suitable for a large system.

7.3 MEMORY AND I/O INTERFACING

Several memory chips and I/O devices are connected to a microprocessor. Figure 7.1 shows a schematic diagram to interface memory chips or I/O devices to a micro-

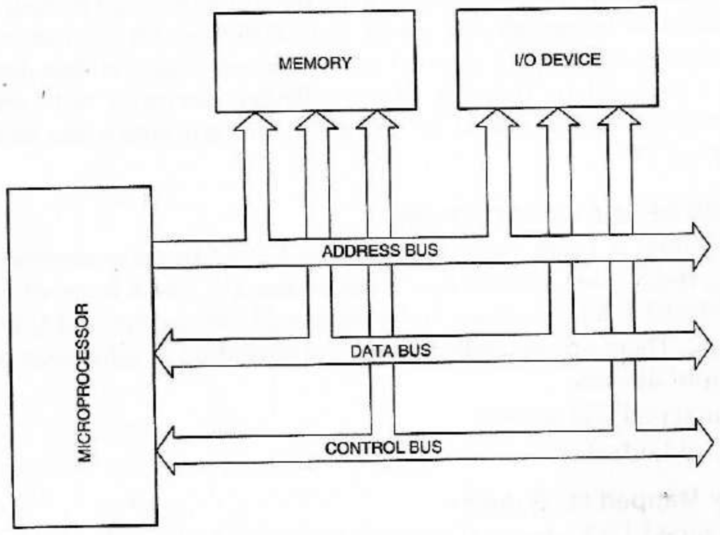


Fig. 7.1 Schematic Diagram for Memory and I/O Interfacing

processor. An address decoding circuit is employed to select the required I/O device or a memory chip. Figure 7.2 shows a schematic diagram of a decoding circuit. If IO/$\overline{\text{M}}$ is high the decoder 2 is activated and the required I/O device is selected. If IO/$\overline{\text{M}}$ is low, the decoder 1 is activated and the required memory chip is selected. A few MSBs of the address lines are applied to the decoder to select a memory chip or an I/O device.

7.3.1 Memory Interfacing

The address of a memory location or an I/O device is sent out by the microprocessor. The corresponding memory chip or I/O device is selected by a decoding circuit. The decoding task can be performed by a decoder, a comparator, a bipolar PROM or PLA (Programmed logic array). In this section the application of 74LS138, a 1 to 8 lines decoder will be illustrated. Figure 7.3 shows the interface of memory chips through 74LS138. G1, G2A and G2B are enable signals. To enable 74LS138, G1 should be high, and G2A and G2B should be low. A, B and C are select lines. By applying proper logic to select lines any one of the outputs can be selected. Y_0, Y_1...... Y_7 are 8 output lines. An output lines goes low when it is selected. Other output lines remain high. Table 7.1 shows the truth table for 74LS138. When G1 is low or G2A is high or G2B is high, all

$\forall$ or $\oplus$	EXCLUSIVE-OR
$\leftarrow$	Move data in the direction of arrow.
$\Leftrightarrow$	Exchange contents.

4.6 INTEL 8085 INSTRUCTIONS

Some of Intel 8085 instructions are frequently, some occasionally and some seldom used by the programmer. It is not necessary that one should learn all the instructions to understand simple programs. The beginner can learn about 15 to 20 important instructions such as MOV, MVI, LXI, LDA, LHLD, STA, SHLD, ADD, ADC, SUB, JMP JC, JNC, JZ, JNZ, INX, DCR, CMP etc., and start to understand simple programs given in Chapter 6. While learning programs he can understand new instructions which he has not learnt earlier.

The operation codes (opcodes) are given in Appendix II. The explanations of the most instructions are given in the subsequent subsections.

4.6.1 Data Transfer Group MOV r_1, r_2

(Move data; Move the content of the one register to another)

$[r_1] \leftarrow [r_2]$. States: 4. Flags: none. Addressing: register.Machine cycle: 1.

The content of register r_2 is moved to register r_1. For example, the instruction MOV A, B moves the content of register B to register A. The instruction MOV B, A moves the content of register A to register B. The time for the execution of this instruction is 4 clock period. One clock period is called *State*. No flag is affected.

MOV r, M. (Move the content of memory to register).

$[r] \leftarrow [[H - L]]$. States: 7. Flag none. Addressing: register indirect. Machine cycles: 2.

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

Example

LXI H, 2000 H	Load H-L pair by 2000H.
MOV B, M	Move the content of the memory location 2000H to register B.
HLT	Halt.

In this example the instruction LXI H, 2000 H loads H-L pair with 2000 H which is the address of a memory location. Then the instruction MOV B, M will move the content of the memory location 2000H to register B.

MOV M, r. (Move the content of register to memory).

$[[H\text{-}L]] \leftarrow [r]$. States: 7. Flags: none. Addressing: reg. indirect. Machine cycles: 2.

The content of register *r* is moved to the memory location addressed by H-L pair. For example, MOV M, C moves the content of register C to the memory location whose address is in H-L pair.

MVI r, data. (Move immediate data to register).

$[r] \leftarrow$ data. States: 7. Flags: none. Addressing: immediate. Machine cycle: 2.

The 1st byte of the instruction is its opcode. The 2nd byte of the instruction is the data which is moved to register *r*. For example, the instruction MVI A, 05 moves 05 to register A. In the code form it is written as 3E, 05. The opcode for MVI A is 3E and 05 is the data which is to be moved to regiser A.

MVI M, data. (Move immediate data to memory).

[[H - L]] ← data. States: 10. Flags: none. Addressing: immediate/reg. indirect. Machine cycle: 3.

The data is moved to memory location whose address is in H-L pair.

Example

LXI H, 2400H	Load H-L pair with 2400H.
MVI M, 08	Move 08 to the memory location 2400H.
HLT	Halt.

In the above example the instruction LXI H, 2400 H loads H-L pair with 2400 H which is the address of a memory location. Then the instruction MVI M, 08 will move 08 to memory location 2400H. In the code form it is written as 36, 08. The opcode for MVI M is 36 and 08 is the data which is to be moved to the memory location 2400H.

LXI rp, data 16. (Load register pair immediate).

[*rp*] ← data 16 bits, [*rh*] ← MSBs, [*rl*] ← 8 LSBs of data.

States: 10. Flags: none. Addressing: immediate. Machine cycles: 3.

This instruction loads 16-bit immediate data into register pair *rp*. This instruction is for register pair; only high order register is mentioned after the instruction. For example, H in the instruction LXI H stands for H-L pair. Similarly, LXI B is for B-C pair. LXI H, 2500H loads 2500H into H-L pair. H with 2500H denotes that the data 2500 is in hexadecimal. In the code form it is written as 21, 00, 25. The 1st byte of the instruction 21 is the opcode for LXI H. The 2nd byte 00 is 8 LSBs of the data and it is loaded into register L. The 3rd byte 25 is 8 MSBs of the data and it is loaded into register H.

LDA addr. (Load Accumulator direct).

[A] ← [*addr*]. States: 13. Flags: none. Addressing: direct. Machine cycles: 4.

The content of the memory location, whose address is specified by the 2nd and 3rd bytes of the instruction; is loaded into the accumulator. The instruction LDA 2400 H will load the content of the memory location 2400 H into the accumulator. In the code form it is written as 3A, 00, 24. The 1st byte 3A is the opcode of the instruction. The 2nd byte 00 is of 8 LSBs of the memory address. The 3rd byte 24 is 8 MSBs of the memory address.

STA Addr. (Store accumulator direct).

[*addr*] ← [A]. States: 13. Flags: none. Addressing: direct. Machine cycles: 4.

The content of the accumulator is stored in the memory location whose address is specified by the 2nd and 3rd byte of the instruction. STA 2000H will store the content of the accumulator in the memory location 2000H.

LHLD addr. (Load H-L pair direct).

[L] ← [*addr*], [H] ← [*addr* + 1]. States: 16. Flags: none. Addressing: direct. Machine cycles: 5.

The content of the memory location, whose address is specified by the 2nd and 3rd bytes of the instruction, is loaded into register L. The content of the next memory location is loaded into register H. For example, LHLD 2500H will load the content of the memory location 2500 H into register L. The content of the memory location 2501H is loaded into register H.

SHLD addr. (Store H-L pair direct)

[*addr*] ← [L], [*addr* + 1] ← [H]. States : 16. Flags: none. Addressing: direct. Machine cycles: 5.

The content of register L is stored in the memory location whose address is specified by the 2nd and 3rd bytes of the instruction. The content of register H is stored in the next memory location. For example, SHLD 2500H will store the content of register L in the memory location 2500H. The content of register H is stored in the memory location 2501H.

LDAX rp. (LOAD accumulator indirect)

[A] ← [[*rp*]]. States; 7. Flags; none. Addressing: register indirect. Machine cycles: 2.

The content of the memory location, whose address is in the register pair *rp*, is loaded into the accumulator. For example, LDAX B will load the content of the memory location, whose address is in the B-C pair, into the accumulator. This instruction is used only for B-C and D-E register pairs.

STAX rp. (Store accumulator indirect)

[[*rp*]] ← [A]. States: 7. Flags: none. Addressing: register indirect. Machine cycles: 2.

The content of the accumulator is stored in the memory location whose address is in the register pair *rp*. For example, STAX D will store the content of the accumulator in the memory location whose address is in D-E pair. This instruction is true only for register pairs B-C and D-E.

XCHG. (Exchange the contents of H-L with D-E pair)

[H-L]↔ [D-E]. States: 4. Flags: none. Addressing: register. Machine cycles: 1.

The contents of H-L pair are exchanged with contents of D-E pair.

4.6.2 Arithmetic Group

ADD r. (Add register to accumulator)

[A] ← [A] + [*r*]. States: 4. Flags: all. Addressing: register. Machine cycle: 1.

The content of register *r* is added to the content of the accumulator, and the sum is placed in the accumulator.

ADD M. (Add memory to accumulator)

[A] ← [A] + [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect, Machine cycles: 2.

The content of the memory location addressed by H-L pair is added to the content of the accumulator. The sum is placed in the accumulator.

ADC r. (Add register with carry to accumulator.)

[A] ← [A] + [*r*] + [CS]. States: 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* and carry status are added to the content of the accumulator. The sum is placed in the accumulator.

ADC M. (Add memory with carry to accumulator)

[A] ← [A] + [[H-L]] + [CS]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair and carry status are added to the content of the accumulator. The sum is placed in the accumulator.

ADI data. (Add immediate data to accumulator)

[A] ← [A] + data. States: 7. Flags: all. Addressing : immediate. Machine cycles: 2.

The immediate data is added to the content of the accumulator. The 1st byte of the instruction is its opcode. The 2nd byte of the instruction is data, and it is added to content of the accumulator. The sum is placed in the accumulator. For example, the instruction ADI 08 will add 08 to the content of the accumulator and place the result in the accumulator. In code form the instruction is written as C6, 08.

ACI data. (Add with carry immediate data to accumulator)

[A] ← [A] + data + [CS]. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2

The 2nd byte of the instruction (which is data) and the carry status are added to the content of the accumulator. The sum is placed in the accumulator.

DAD rp. (Add register paid to H-L pair)

[H-L] ← [H-L] + [*rp*]. States: 10. Flags: CS. Addressing: register. Machine cycles: 3.

The contents of register pair *rp* are added to the contents of H-L pair and the result is placed in H-L pair. Only carry flag is affected.

SUB r. (Subtract register from accumulator)

[A] ← [A] – [*r*]. States: 4 Flags: all. Addressing: register. Machine cycles: 1.

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

SUB M. (Subtract memory from accumulator).

[A] ← [A] – [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2

The content of the memory location addressed by H-L pair is subtracted from the content of the accumulator. The result is placed in the accumulator.

SBB r. (Subtract register from accumulator with borrow).

[A] ← [A] – [*r*] – [CS]. States: 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* and carry status are subtracted from the content of the accumulator. The result is placed in the accumulator.

SBB M. (Subtract memory from accumulator with borrow).

[A] ← [A] – [[H-L]] – [CS]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair and carry status are subtracted from the content of the accumulator. The result is placed in the accumulator.

SUI data. (Subtract immediate data from accumulator)

[A] ← [A] – data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data. It is subtracted from the content of the *accumulator. The result is placed in the accumulator. For example, the instruction SUI 05 will subtract* 05 from the content of the accumulator and place the result in the accumulator. In the code form the above instruction is written as D6, 05.

SBI data. (Subtract immediate data from accumulator with borrow).

[A] ← [A] – data – [CS]. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2

The data and carry status are subtracted from the content of the accumulator. The result is placed in the accumulator

INR r. (Increment register content)

[r] ← [r] + 1. States: 4. Flags: all except carry flag. Addressing: register. Machine cycle: 1.

The content of register *r* is incremented by one. All flags except CS are affected.

INR M. (Increment memory content)

[[H-L]] ← [[H-L]] + 1. States: 10. Flags: all except carry flag. Addressing: reg. indirect.

Machine cycles: 3.

The content of the memory location addressed by H-L pair is incremented by one. All flags except CS are affected.

DCR r. (Decrement register content) [*r*] ←[*r*] − 1. States: 4. Flags: all except carry flag, Addressing: register. Machine cycles: 1.

The content of register *r* is decremented by one. All flags except CS are affected.

DCR M. (Decrement memory content)

[[H-L]] ← [[H-L]] − 1. States: 10. Flags: all except carry flag. Addressing: reg. indirect. Machine cycles: 3.

The content of the memory location addressed by H-L pair is decremented by one. All flags except CS are affected.

INX rp. (increment register pair)

[*rp*] ←[*rp*] + 1. States: 6. Flags: none. Addressing: register. Machine cycles : 1.

The content of the register pair *rp* is incremented by one. No flag is affected.

DCX rp (Decrement register pair)

[*rp*] ←[*rp*] − 1. States: 6. Flags: none. Addressing: register. Machine cycles: 1.

The content of the register pair *rp* is decremented by one. No flag is affected.

DAA. (Decimal adjust accumulator)

States: 4. Flags: all. Machine cycle : 1.

The instruction DAA is used in the program after ADD, ADI, ACI, ADC, etc instructions. After the execution of ADD, ADC, etc instructions the result is in hexadecimal and it is placed in the accumulator. The DAA instruction operates on this result and gives the final result in decimal system. It uses carry and auxiliary carry for decimal adjustment. 6 is added to 4 LSBs of the content of the accumulator if their value lies in between A and F or the AC flag is set to 1. Similarly, 6 is also added to 4 MSBs of the content of the accumulator if their value lies in between A and F or the CS flag is set to 1. All status flags are affected. When DAA is used data should be in decimal numbers.

4.6.3 Logical Group

The instructions of this group perform AND, OR, EXCLUSIVE-OR operations; compare, rotate or take complement of data in register or memory.

ANA r. (AND register with accumulator)

[A] ←[A] ∧ [*r*]. States: 4. Flags : all. Addressing: register. Machine cycles: 1.

The content of register *r* is ANDed with the content of the accumulator, and the result is placed in the accumulator. All status flags are affected. The flag CS is cleared, *i.e.* it is set to 0. Auxiliary carry flag AC is set to 1.

ANA M. (AND memory with accumulator)

[A] ←[A] ∧ [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair is ANDed with the accumulator. The result is placed in the accumulator. All flags are affected. The CS flag is set to 0 and AC to 1.

ANI data. (AND immediate data with accumulator)

[A] ← [A] ∧ data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data, and it is ANDed with the content of the accumulator. The result is placed in the accumulator. The CS flag is set to 0 and AC to 1.

ORA r. (OR register with accumulator)

[A] ← [A] ∨ [*r*] States: 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* is ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. Carry and auxilary carry are cleared *i.e.* the CS and AC flags are set to 0.

ORA M. (OR memory with accumulator)

[A] ← [A] ∨ [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair is ORed with the content of the accumulator. The result is placed in the accumulator. The CS and AC flags are set to 0.

ORI data. (OR immediate data with accumulator)

[A] ← [A] ∨ data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data, and it is ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. The CS and AC flags are set 0.

XRA r. *(EXCLUSIVE - OR register with accumulator)*

[A] ← [A] ∀□ [*r*] States: 4. Flags: all. Addressing: register. *Machine cycles: 1.*

The content of register *r* is EXCLUSIVE - ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. The CS and AC flags are set to 0.

XRA M. (EXCLUSIVE - OR memory with accumulator)

[A] ← [A] ∀ [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair is EXCLUSIVE-ORed with the content of the accumulator. The result is placed in the accumulator. All status flags are affected. The CS and AC flags are set to 0.

XRI data. (EXCLUSIVE - OR immediate data with accumulator)

[A] ← [A] ∀ data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data, and it is EXCLUSIVE-ORed with the content of the accumulator. The result is placed in the accumulator. All flags are affected. The CS and AC flags are set to 0.

CMA. (Complement the accumulator)

[A] ← $[\bar{A}]$. States: 4. Flags: none. Machine cycles: 1. Addressing: implicit.

1's complement of the content of the accumulator is obtained, and the result is placed in the accumulator. To obtain the 1's complement of a binary number 0 is replaced by 1, and 1 by 0. For example, one's complement of 1100 is 0011.

CMC. (Complement the carry status)

[CS] ← $[\overline{CS}]$. States: 4. Flags: CS, Machine cycle: 1.

The CS flag is complemented. Other flags are not affected.

STC. (Set carry status)

[CS] ← 1. States: 4. Flags: CS. Machine cycles: 1.

The status flag CS is set to 1. Other flags are not affected.

CMP r. (Compare register with accumulator)

[A] – [*r*]. States 4. Flags: all. Addressing: register. Machine cycles: 1.

The content of register *r* is subtracted from the content of the accumulator and status flags are set according to the result of the subtraction. But the result is discarded. The content of the accumulator remains unchanged.

CMP M. (Compare memory with accumulator)

[A] – [[H-L]]. States: 7. Flags: all. Addressing: reg. indirect. Machine cycles: 2.

The content of the memory location addressed by H-L pair is subtracted from the content of the accumulator, and status flags are set according to the result of the subtraction. But the result is discarded. The content of the accumulator remains uncharged.

CPI data. (Compare immediate data with accumulator)

[A] – data. States: 7. Flags: all. Addressing: immediate. Machine cycles: 2.

The 2nd byte of the instruction is data, and it is subtracted from the content of the accumulator. The status flags are set according to the result of subtraction. But the result is discarded. The content of the accumulator remains unchanged.

RLC. (Rotate accumulator left)

$[A_{n+1}] \leftarrow [A_n]$, $[A_0] \leftarrow [A_7]$, $[CS] \leftarrow [A_7]$.

States: 4. Flags: CS. Machine cycles : 1. Addressing: implicit.

The content of the accumulator is rotated left by one bit. The seventh bit of the accumulator is moved to carry bit as well as to the zero bit of the accumulator. Only CS flag is affected. See Fig. 4.1

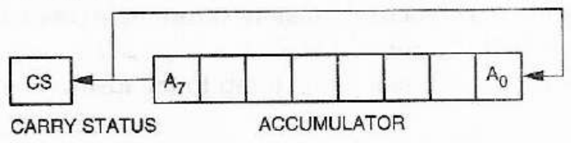


Fig. 4.1 Schematic Diagram for RLC

RRC. (Rotate accumulator right)

$[A_7] \leftarrow [A_0]$, $[CS] \leftarrow [A_0]$, $[A_n] \leftarrow [A_{n+1}]$.

States: 4. Flags: CS. Machine cycles : 1. Addressing: implicit.

The content of the accumulator is rotated right by one bit. The zero bit of the accumulator is moved to the seventh bit as well as to carry bit. Only CS flag is affected. See Fig. 4.2.

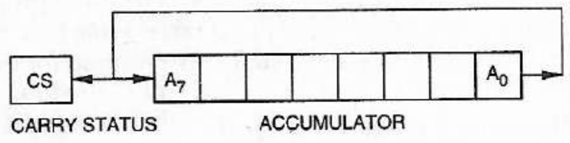


Fig. 4.2 Schematic Diagram for RRC

RAL. (Rotate accumulator left through carry)

$[A_{n+1}] \leftarrow [A_n]$, $[CS] \leftarrow [A_7]$, $[A_0] \leftarrow [CS]$.

States: 4. Flags: CS. Machine cycles: 1. Addressing: implicit.

The content of the accumulator is rotated left one bit through carry. The seventh bit of the accumulator is moved to carry, and the carry bit is moved to the zero bit of the accumulator. Only carry flag is affected. See Fig. 4.3.

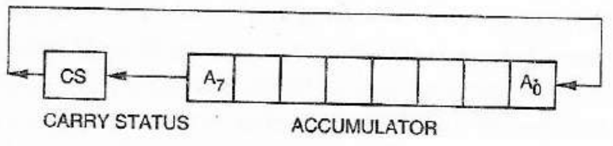


Fig. 4.3 Schematic Diagram for RAL

RAR. (Rotate accumulator right through carry)

$[A_n] \leftarrow [A_{n+1}]$, $[CS] \leftarrow [A_0]$, $[A_7] \leftarrow [CS]$

States: 4 Flags : CS. Machine cycle: 1. Addressing implicit.

The content of the accumulator is rotated right one bit through carry. The zero bit of the accumulator is moved to carry, and the carry bit to the seventh bit of the accumulator. Only CS flag is affected. See Fig. 4.4.

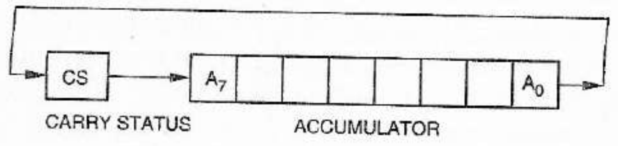


Fig. 4.4 Schematic Diagram for RAR

4.6.4 Branch Group

The instructions of this group change the normal sequence of the program. There are two types of branch instructions: conditional and unconditional. The conditional branch instructions transfer the program to the specified label when certain condition is satisfied. The unconditional branch instructions transfer the program to the specified label unconditionally.

JMP addr (label). (Unconditional jump: jump to the instruction specified by the address).

[PC] ← Label. States: 10. Flags: none. Addressing: immediate. Machine cycles: 3.

Byte 2nd and byte 3rd of the instruction give the address of the label where the program jumps. The address of the label is the address of the memory location for next instruction to be executed. The program jumps to the instruction specified by the address (label) unconditionally.

Conditional Jump addr (label). After the execution of the conditional jump instruction the program jumps to the instruction specified by the address (label) if the specified condition is fulfilled. The program proceeds further in the normal sequence if the specified condition is not fulfilled. If the condition is true and program jumps to the specified label, the execution of a conditional jump takes 3 machine cycles: 10 states. If condition is not true, only 2 machine cycles; 7 states are required for the execution of the instruction.

(i) **JZ addr (label).** (Jump if the result is zero)

[PC] ← address (label), jump if $Z = 1$. States: 7/10. Flags: none. Addressing: immediate.

Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if the result is zero (*i.e.* the zero status Z = 1). Here the result after the execution of the preceding instruction is under consideration.

(ii) **JNZ addr (label).** (Jump if the result is not zero)

[PC] ← address (label), jump if Z = 0. States: 7/10. Flags: none. Addressing : immediate.

Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if the result is non-zero (*i.e* the zero status Z = 0).

(iii) **JC addr (label).** (Jump if there is a carry)

[PC] ← address (label), jump if CS = 1. States: 7/10. Flags: none. Addressing : immediate. Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if there is a carry (*i.e.* the carry status: CS = 1). Here the carry after the execution of the preceding instruction is under consideration.

(iv) **JNC addr (label).** (Jump if there is no carry)

[PC] ← address (label), jump if CS = 0. States: 7/10. Flags: none. Addressing : immediate. Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if there is no carry (*i.e.* the carry states CS = 0).

(v) **JP addr (label).** (Jump if the result is plus)

[PC] ← address (label), jump if S = 0. States: 7/10. Flags: none. Addressing: immediate.

Machine cycles: 2/3.

The program jumps to the instruction specified by the address (label) if the result is plus.

(vi) **JM addr (label).** (Jump if the result is minus)

[PC] ← address (label), jump if S = 1. States: 7/10. Flags: none. Addressing: immediate.

Machine cycles: 2/3.

If the result is minus the program jumps to the instruction specified by the address (label).

(vii) **JPE addr (label).** (Jump if even parity)

[PC] ← address (label), jump if even parity: the parity status P = 1, States: 7/10. Flags: none. Addressing: immediate. Machine cycles: 2/3.

If the result contains even number of 1s, the program jumps to the instruction specified by the address (label).

(viii) **JPO addr (label).** (Jump if odd parity)

[PC] ← address (label), jump if odd parity ; the parity status P = 0, States: 7/10, Flags: none. Addressing: immediate, Machine cycles: 2/3.

If the result contains odd number of 1s, the program jumps to the instruction specified by the address (label).

CALL addr (label). (Unconditional CALL: call the subroutine identified by the address)

[[SP] − 1] ← [PCH], Save the address of the next instruction of the program in the stack.

[[SP] – 2] ← [PCL],

[SP] ← ([SP] – 2)

[PC] ← addr (label)

States: 18. Flags: none, Addressing: immediate/reg. indirect. Machine cycles: 5.

CALL instruction is used to call a subroutine. Before the control is transferred to the subroutine, the address of the next instruction of the main program is saved in the stack. The content of the stack pointer is decremented by two to indicate the new stacktop. Then the program jumps to subroutine starting at address specified by the label.

Conditional CALL addr (label)

[[SP] – 1] ← [PCH], [[SP] – 2] ← [PCL],

[PC] ← addr (label), [SP] ← ([SP– 2]).

States: 9/18. Flags: none. Addressing: immediate/reg. indirect. Machine cycles: 2/5. If the condition is true and program calls the specified subroutine, the execution of a conditional call instruction takes 5 machine cycles; 18 states. If condition is not true, only 2 machine cycles; 9 states are required for the execution of the instruction.

(i)	CC	addr (label)	Call subroutine if carry status CS = 1.
(ii)	CNC	addr (label)	Call subroutine if carry status CS = 0.
(iii)	CZ	addr (label)	Call subroutine if the result is zero; the zero status Z = 1.
(iv)	CNZ	addr (label)	Call subroutine if the result is not zero; the zero status Z = 0.
(v)	CP	addr (label)	Call subroutine if the result is plus; the sign status S = 0.
(vi)	CM	addr (label)	Call subroutine if the result is minus, the sign status S = 1.
(vii)	CPE	addr (label)	Call subroutine if even parity; the parity status P = 1.
(viii)	CPO	addr (label)	Call subroutine if odd parity; the parity status P = 0.

RET. (Return from subroutine)

[PCL] ← [[SP]],

[PCH] ← [[SP] + 1],

[SP] ← ([SP] + 2).

States: 10. Flags: none. Addressing: reg. indirect. Machine cycles: 3.

RET instruction is used at the end of a subroutine. Before the execution of a subroutine the address of the next instruction of the main program is saved in the stack. The execution of RET instruction brings back the saved address from the stack to the program counter. The content of the stack pointer is incremented by 2 to indicate the new stack top. Then the program jumps to the instruction of the main program next to CALL instruction which called the subroutine.

Conditional Return

[PCL] ← [[SP]], [PCH] ← [[SP] + 1],

[SP] ← ([SP] + 2).

States: 6/12. Flags: none. Addressing: reg. indirect. Machine cycles: 1/3. If the condition is true and the program returns from the subroutine, the execution of a conditional return instruction takes 3 machine cycles, 12 states. If condition is not true only one machine cycle, 6 states are required.

(i)	RC	Return from subroutine if carry status CS = 1.
(ii)	RNC	Return from subroutine if carry status CS = 0.
(iii)	RZ	Return from subroutine if the result is zero; the zero status Z = 1.
(iv)	RNZ	Return from subroutine if the result is not zero; the zero status Z = 0.
(v)	RP	Return from subroutine if the result is plus; the sign status S = 0.
(vi)	RM	Return from subroutine if the result is minus, the sign status S = 1.
(vii)	RPE	Return from subroutine if even parity, the parity status P = 1.
(viii)	RPO	Return from subroutine if odd parity, the parity status P = 0.

RST n (Restart).

[[SP] − 1] ← [PCH], [[SP] − 2] ← [PCL],

[SP] ← ([SP] − 2), [PC] ← 8 times *n*.

States: 12. Flags: none, Addressing: reg. indirect. Machine cycles : 3.

Restart is a one-word CALL instruction. The content of the program counter is saved in the stack. The program jumps to the instruction starting at restart location. The address of the restart location is 8 times *n*. The restart instruction and locations are as follows:

Instruction	*Opcode*	*Restart Locations*
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

PCHL. (Jump to address specified by H-L pair)

[PC] ← [H-L], [PCH] ← [H], [PCL] ← [L]

States: 6. Flags: none. Addressing: register. Machine cycle: 1.

The contents of H-L pair are transferred to program counter. The contents of register H are moved to high order 8 bits of register PC. The contents of register L are transferred to low order 8 bits of register PC.

4.6.5 Stack, I/O and Machine Control Group

IN port-address. (Input to accumulator from I/O port)

[A] ← [Port]. States: 10. Flags: none. Addressing: direct. Machine cycles: 3.

The data available on the port is moved to the accumulator. After instruction IN, the address of the port is specified. The 2nd byte of the instruction contains the address of the port. The address of a port is an 8-bit address. For example, IN 01. The address of the port B of an I/O port 8255.1 of a microprocessor kit is 01.

OUT port-address. (Output from accumulator to I/O port)

[Port] ← [A]. States: 10. Flags: none. Addressing: direct. Machine cycles: 3.

The content of the accumulator is moved to the port specified by its address. After the OUT instruction, the port address is specified. The 2nd byte of the instruction contains the address of the port. For example, OUT 00. The address of the port A of an I/O port 8255.1 of a microprocessor kit is 00.

PUSH rp. (Push the content of register pair to stack)

[[SP] − 1] ← [*rh*],

[[SP] − 2] ← [*rl*],

[SP] ← ([SP] − 2).

States: 12. Flags: none. Addressing: register(source)/reg. indirect(destination), Machine cycles: 3.

The content of the register pair *rp* is pushed into the stack.

PUSH PSW. (PUSH program status word to the stack)

[[SP] − 1] ← [A]

[[SP] − 2] ← PSW (Program Status Word)

[SP] ← ([SP] − 2).

States: 12, Flags: none. Addressing: register(source)/reg.indirect(destination), Machine cycles: 3.

The content of the accumulator is pushed into the stack. The contents of status flags are also pushed into the stack. The content of the register SP is decremented by 2 to indicate new stacktop.

POP rp. (Copy two bytes from the top of the stack into the specified register)

[*rl*] ← [[SP]]

[*rh*] ← [[SP] + 1]

[SP] ← ([SP] + 2).

States: 10. Flags: none. Addressing: register(destination)/reg.indirect (source), Machine cycles: 3.

The content of the register pair, which was saved earlier is moved from the stack to the register pair.

POP PSW. (Copy two bytes from the top of the stack into PSW and Accumulator)

PSW ← [[SP]]

[A] ← [[SP] + 1]

[SP] ← ([SP] + 2).

States: 10. Flags: all. Addressing: reg. indirect. Machine cycles: 3.

The processor status word which was saved earlier during the execution of the program is moved from the stack to PSW. The content of the accumulator which was also saved is moved from the stack to the accumulator.

HLT (Halt)

States: 5. Flags: none. Machine cycle: 1.

When this instruction is executed, any further program execution is stopped. The microprocessor remains in Halt state. An interrupt or reset is required to exit from Halt state. Registers and status flags remain unaffected.

XTHL. (Exchange stack-top with H-L)

UNIT – II

Microprocessor Programming

ASSEMBLY LANGUAGE PROGRAMS

6.1 INTRODUCTION

To learn assembly language programming the beginner should write simple programs given in this chapter and try to execute them on Intel 8085 based microprocessor kit. The memory addresses given in the program are for a particular microprocessor kit. These addresses can be changed to suit the microprocessor kit available in the laboratory. Before writing assembly language program, one should learn some important Intel 8085 instructions such as MOV, MVI, ADD, SUB, LXI, LDA, INX, INR, HLT etc. described in chapter 4. While learning programs, one should gradually pick up new instructions which have not been used earlier.

6.2 SIMPLE EXAMPLES

The use of some important instructions are described below with very simple examples.

The memory addresses given in these examples are for Vinytics' kit.

Example 1. Object: Place 05 in register B.

PROGRAM

Memory address	*Machine codes*	*Mnemonics*	*Operands*	*Comments*
FC00	06,05	MVI	B, 05	Get 05 in register B.
FC02	76	HLT		Stop.

The instruction MVI B, 05 moves 05 to register B. HLT halts the program. A program is fed to the microprocessor kit in machine codes. The machine code for the instruction MVI B, 05 is 06, 05. The 1st byte of the instruction is 06 which is the machine code for the instruction MVI B. The second byte of the instruction, 05 is the data which is to be moved to register B. The code for HLT is 76. The machine codes for a program are entered in the memory. In the above program the memory addresses from FC00 to FC02 have been used. The machine code 06 is entered in the memory location FC00 H; 05 in FC01 H and 76 in FC02 H. This program can be executed on a microprocessor kit and the register B can be examined. After the execution of the program the register B will contain 05. The memory address can be changed to suit the microprocessor kit available in the laboratory. The address and data used for a microprocessor based system are in hexadecimal system. The symbol H after a digit denotes that it is in hexadecimal system.

Example 2 Object: Get 05 in register A; then move it to register B.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3E, 05	MVI	A, 05	Get 05 in register A.
FC02	47	MOV	B, A	Transfer 05 from register A to B.
FC03	76	HLT		Stop.

Note: Microprocessors are designed to process hexadecimal numbers. Hence, data and address given in an assembly language program are hexadecimal numbers.

The instruction MVI A, 05 will move 05 to register A. In the code from it is written as 3E, 05. The 1st byte of the instruction is 3E. This code is for MVI A. The second byte 05 is the data which is to be placed in A. The instruction MOV B, A transfers the content of register A to register B. After the execution of the above program register B will contain 05. The program can be executed on a microprocessor kit and the register B can be examined.

Example 3

Object: Load the content of the memory location FC50 H directly to the accumulator, then transfer it to register B. The content of the memory location FC50 H is 05.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3A, 50, FC	LDA	FC50	Get the content of the memory location FC50 H into accumulator.
FC03	47	MOV	B,A	Move the content of register A to B.
FC04	76	HLT		Halt.

DATA

FC50 — 05

The instruction LDA loads the accumulator directly with the content of the memory location specified in the instruction. Thus the instruction LDA FC50 H will load the accumulator with the content of the memory location FC50 H. The instruction MOV B, A will move the content of the accumulator to B. The content of the memory location FC50 H is 05. It is fed to the microprocessor kit as data. After the execution of the above program the register B will contain 05.

Example 4

Object: Move the content of the memory location FC50 H to register C. The content of the memory location FC50 H is 08.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	21, 50, FC	LXI	H,FC50	Get the memory address FC50 H in H-L pair.
FC03	4E	MOV	C,M	Move the content of the memory location, whose address is in the H-L pair, to register C.
FC04	76	HLT		Halt.

DATA

FC50 — 08

The instruction LXI H, FC50 H will place FC50 H in register pair H-L. FC50 H is the address of the memory location from where the data is to be transferred to register C. In the code form the instruction is written as 21, 50, FC. The 1st byte of the instruction is 21. It is the machine code for the instruction LXI H. The operand is FC50 which is to be placed in H-L pair. The 2nd byte of the instruction is 50. It is 8 LSBs of the operand. The 3rd byte is FC which is 8 MSBs of the operand. In the code form the LSB is written first then the MSB. Due to this reason the operand FC50 has been written in the code form as 50, FC. The instruction MOV C, M transfers the content of the memory location, whose address is in H-L pair, to register C. The execution of the previous instruction has placed FC50 H in H-L pair. Therefore, MOV C, M will move the content of FC50 H to register C. After the execution of the above program the register C will contain 08.

Example 5

Object : Place the content of the memory location FC50 H in register B and that of FC51 H in register C. The contents of FC50 and FC51 H are 11 H and 12 H respectively.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	21, 50, FC	LXI	H, FC50 H	Get FC50 H in H-L pair.
FC03	46	MOV	B, M	Move the content of FC50 to B.
FC04	23	INX	H	Increment H-L pair by one.
FC05	4E	MOV	C, M	Move the content of FC51 to C.
FC06	76	HLT		Halt.

DATA

FC50 — 11 H

FC51 — 12 H.

The instruction LXI H, FC50 H will place FC50 H in H-L pair. FC50 H is the address of the memory location which contains 11 H. MOV B, M will move the content of FC50 H to register B. The instruction INX H increases the content of H-L pair by 1. The execution of the instruction INX H will increase the content of H-L pair from FC50 H to FC51 H. MOV C, M will move the content of FC51 H to register C. Thus, after the execution of the above program, the register B and C will contain 11 H and 12 H respectively.

Example 6

Object: Place 05 in the accumulator. Increment it by one and store the result in the memory location FC50 H.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
FC00	3E,05	MVI	A,05	Get 05 in the accumulator.
FC02	3C	INR	A	Increment the content of accumulator by one.
FC03	32,50,FC	STA	FC50 H	Store result in FC50 H.
FC06	76	HLT		Halt

The instruction MVI A, 05 moves 05 to the accumulator. INR A increases the content of the accumulator from 05 to 06. STA FC50 stores the content of the accumulator in the memory location FC50 H. After the execution of the above program the memory location FC50 H will contain 06.

A number of important and useful assembly language programs are given in the subsequent sections. Memory addresses used for them are for Professional's kits/Vinytics' kits.

6.3 ADDITION OF TWO 8-BIT NUMBERS; SUM 8-BIT

Problem:

Add 49 H and 56 H.

The 1st number 49 H is in the memory location 2501 H.

The 2nd number 56 H is in the memory location 2502 H.

The result is to be stored in the memory location 2503 H.

Numbers are represented in hexadecimal system.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
2000	21, 01, 25	LXI	H, 2501 H	Get address of 1st number in H-L pair.
2003	7E	MOV	A,M	1st number in accumulator.
2004	23	INX	H	Increment content of H-L pair.
2005	86	ADD	M	Add 1st and 2nd numbers.
2006	32, 03, 25	STA	2503 H	Store sum in 2503 H.
2009	76	HLT		Stop

DATA

2501 — 49 H

2502 — 56 H

The sum is stored in the memory location 2503 H.

Result

2503 — 9F H.

2501 H is the address of memory location for the 1st number. 2501 is placed in H-L pair by the instruction LXI H, 2501 H. The next instruction is MOV A, M which moves the content of the memory location addressed by H-L pair to the accumulator. In this case H-L pair contains 2501 H and, therefore, the content of the memory location 2501 H is moved to the accumulator. Thus, the 1st number 49 H has been moved to the accumulator. The instruction INX H increases the content of H-L pair by one. Previously, the content of H-L pair was 2501 H. After the execution of INX H it becomes 2502 H. ADD M adds the contents of the accumulator and the content of the memory location addressed by H-L pair. The content of 2502 H is the 2nd number 56 H. So, 56 H is added to 49 H. Sum resides in the accumulator. The instruction STA 2503 H stores the sum in the memory location 2503 H. The instruction HLT ends the program.

6.4 8-BIT SUBTRACTION*

Example 1

	49 H	1st number
	– 32 H	2nd number
Result	17 H	

The 1st number 49 H is in the memory location 2501 H.

The 2nd number 32 H is in the memory location 2502 H.

The result is to be stored in the memory location 2503 H.

***Note:** If the second number is greater than the first number, the processor will give result in 2's complement i.e. negative result. See Section 6.39.

PROGRAM

Memory address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
2000	21, 01, 25	LXI	H, 2501 H	Get address of 1st number in H-L pair.
2003	7E	MOV	A, M	1st number in accumulator.
2004	23	INX	H	Content of H-L pair increases from 2501 to 2502 H.
2005	96	SUB	M	1st number - 2nd number.
2006	23	INX	H	Content of H-L pair becomes 2503 H.
2007	77	MOV	M, A	Store result in 2503 H.
2008	76	HLT		Halt

Example 1	Example 2
DATA	**DATA**
2501 — 49 H	2501 — F8 H
2502 — 32 H	2502 — 9B H
Result is stored in the Memory location 2503 H	Result
2503 — 17 H	2503 — 5D H

The 1st number is in the memory location 2501 H. 2501 is placed in H-L pair by the execution of the instruction LXI H, 2501 H. The instruction MOV A, M moves the content of the memory location addressed by H-L pair to the accumulator. Thus, the 1st number 49 H (Example 1) which is in 2501 H is placed in the accumulator. INX H increases the content of H-L pair from 2501 to 2502 H. The instruction SUB M subtracts the content of the memory location addressed by H-L pair from the accumulator. The 2nd number which is in the memory location 2502 H is subtracted from the 1st number which is in the accumulator. The result resides in the accumulator. The instruction INX H increases the content of H-L pair from 2502 to 2503 H. The instruction MOV M, A transfers the content of the accumulator to the memory location addressed by H-L pair. So the result which is in the accumulator is sotred in the memory location 2503 H. The instruction HLT ends the program.

6.5 ADDITION OF TWO 8-BIT NUMBERS; SUM: 16-BITS

Example 1

Add 98 H and 9A H.

SUM = 01, 32 H.

The 1st number 98 H is in the memory location 2501 H.

The 2nd number 9A H is in the memory location 2502 H.

The results are to be stored in 2503 and 2504 H.

Numbers are represented in hexadecimal.

In this case the sum is to be stored in two consecutive memory locations. The LSBs of the sum is 32 H and it will be stored in the memory location 2503 H. The MSB of the sum is 01 which will be stored in 2504 H.

Note. The program given below is also applicable for addition of 8-bit numbers; sum: 8-bit. The MSBs of the sum for the such a case will be 00.

PROGRAM

Memory address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
2000	21, 01, 25		LXI	H, 2501 H	Address of 1st number in H-L pair.
2003	0E, 00		MVI	C,00	MSBs of sum in register C. Initial value = 00.
2005	7E		MOV	A, M	1st number in accumulator.
2006	23		INX	H	Address of 2nd number 2502 in H-L pair.
2007	86		ADD	M	1st number + 2nd number.
2008	D2, 0C, 20		JNC	AHEAD	Is carry? No, go to the label AHEAD.
200B	0C		INR	C	Yes, increment C.
200C	32, 03, 25	AHEAD	STA	2503 H	LSBs of sum in 2503 H.
200F	79		MOV	A, C	MSBs of sum in accumulator.
2010	32, 04, 25		STA	2504 H	MSBs of sum in 2504 H.
2013	76		HLT		Halt

Example 1	Example 2
DATA	**DATA**
2501 — 98 H	2501 — F5 H
2502 — 9A H	2502 — 8A H
Result	**Result**
2503 — 32 H, LSBs of sum.	2503 — 7F H, LSBs of sum.
2504 — 01 H, LSBs of sum.	2504 — 01 H, MSBs of sum.

The 1st instruction of the program LXI H, 2501 H gets the address of the 1st number in H-L pair. The MSBs of the sum is placed in register C. The initial value of the MSBs of the sum is kept 00. MOV A, M transfers 1st number from the memory location 2501 H to the accumulator. INX H increases the content of H-L pair from 2501 to 2502 H. The 2nd number is in 2502 H. ADD M adds 1st and 2nd numbers. After the execution of the instruction JNC the instruction INR C is executed if the addition of two numbers produces a carry. In binary system carry is equal to one and, therefore, the content of the register C is increased from 00 to 01. The value 01 is nothing but the MSBs of the sum. Now the register C contains the MSBs of the sum. The accumulator contains the LSBs of the sum. The instruction STA 2503 H places the LSBs of the sum in memory location 2503 H. The instruction MOV A, C moves the MSBs of the sum from register C to the accumulator. The instruction STA 2504 H stores the MSBs of the sum in 2504 H.

After the execution of the instruction JNC the program jumps to label AHEAD and executes STA 2503 H, if the addition of two numbers does not produce a carry. The LSBs of the sum is placed in the memory location 2503 H. Register C contains 00, so the execution of the instruction MOV A, C transfers 00 in the accumulator. The instruction STA 2504 H stores 00 in the memory location 2504 H. This is the MSB of the sum. Such a situation will arise when the sum of two numbers is of 8-bits, and hence there is no carry.

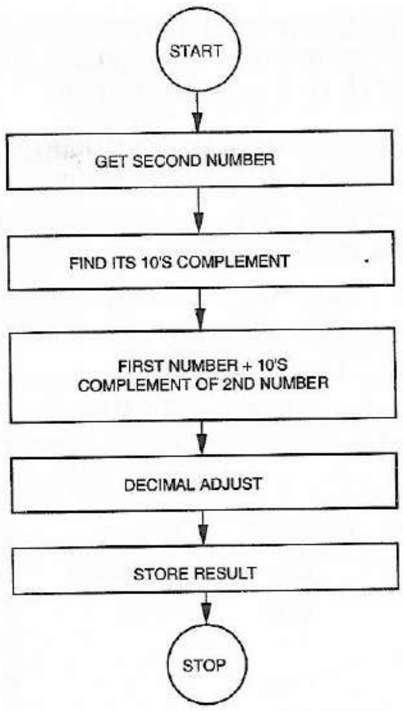


Fig. 6.1 Program Flow Chart for Decimal Subtraction

6.9 FIND ONE'S COMPLEMENT OF AN 8-BIT NUMBER

Example 1. Find one's complement of 96 H. The number in the binary form is represented as follows:

96 H = 1001 0110
(9) (6)

One's complement = 0110 1001 = 69 H.
(6) (9)

To obtain one's complement of a number its 0 bits are replaced by 1 and 1 by 0.

The number is placed in the memory location 2501 H.

The result is stored in the memory location 2502 H.

PROGRAM

Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
2000	3A, 01, 25	LDA	2501 H	Get data in accumulator.
2003	2F	CMA		Take its complement.
2004	32, 02, 25	STA	2502 H	Store result in 2502 H.
2007	76	HLT		Halt.

Example 1	**Example 2**
DATA	**DATA**
2501 — 96 H	2501 — E4 H
Result	Result
2502 — 69 H	2502 — 1B H

6.16 SHIFTING OF A 16-BIT NUMBER LEFT BY 2 BITS

Example. Shift 1596 H left by 2 bits.

1596	= 0001	0101	1001	0110	
	(1)	(5)	(9)	(6)	
Result of shifting left by one bit	= 0010	1011	0010	1100	= 2B2C
	(2)	(B)	(2)	(C)	
Result of shifting 2 bits left	= 0101	0110	0101	1000	= 5658
	(5)	(6)	(5)	(8)	

The number is stored in the memory locations 2501 and 2502 H.

The result is to be stored in the memory locations 2503 and 2504 H.

PROGRAM

Memory Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
2000	2A, 01, 25	LHLD	2501 H	Get data in H-L pair.
2003	29	DAD	H	Shift left by one bit.
2004	29	DAD	H	Again shift left by one bit.
2005	22, 03, 25	SHLD	2503 H	Store result in 2503 and 2504 H.
2008	76	HLT		Stop

DATA

2501 — 96 H, LSBs of the number.
2502 — 15 H, MSBs of the number.

Result

2503 — 58 H, LSBs of the result.
2504 — 56 H, MSBs of the result.

6.17 MASK OFF LEAST SIGNIFICANT 4 BITS OF AN 8-BIT NUMBER

Example.

Number = A6		
=	1010	0110
	(A)	(6)
Result = 06 =	0000	0110
	(A)	(0)

We want to mask off the least significant 4 bits of a given number. The LSD of the given number A6 is 6. It is to be cleared (masked off) *i.e.* it is to be made equal to zero. The MSD of the number A6 is A. In the binary form it is 1010. It is not to be affected. If this number is ANDed with 1111 *i.e.* F, it will not be affected. Similarly, the LSD of the number is 6. In the binary form it is represented by 0110. If it is ANDed with 0000, it becomes 0000 *i.e.* it is cleared. Thus, if the number A6 is ANDed with F0, the LSD of the number is masked off.

PROGRAM

Address	Machine Codes	Mnemonics	Operands	Comments
2000	3A, 01, 25	LDA	2501 H	Get data in accumulator.
2003	E6, F0	ANI	F0	Mask off the least significant 4 bits.
2005	32, 02, 25	STA	2502 H	Store result in 2502 H.
2008	76	HLT		Stop

DATA
2501 — A6
Result
2502 — A0

The instruction LDA 2501 H transfers the content of memory location 2501 H *i.e.* the given number to the accumulator. ANI F0 logically ANDs the content of the accumulator with F0 to clear the least significant 4 bits of the number. STA 2502 H stores the result in memory location 2502 H. HLT stops the program.

6.18 MASK OFF MOST SIGNIFICANT 4 BITS OF AN 8-BIT NUMBER

Example.

```
Number = A6
       = 1010      0110
          (A)       (6)
Result = 06 = 0000      0110
               (0)       (6)
```

To mask off 4 most significant bits of a number, 4 MSBs are ANDed with 0000. The least significant bits are not to be affected and, therefore, they are ANDed with 1111 *i.e.* F. Thus, if an 8-bit number is ANDed with 0F, the 4 most significant bits are cleared.

Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
2000	3A, 01, 25	LDA	2501 H	Get data in accumulator.
2003	E6, 0F	ANI	0F	Mask off the most significant 4 bits.
2005	32, 02, 25	STA	2502 H	Store result in 2502 H.
2008	76	HLT		Stop.

DATA
2501 — A6
Result
2502 — 06

The instruction LDA 2501 H transfers the content of memory location 2501 H to the accumulator. ANI 0F logically ANDs the content of the accumulator with 0F to clear the most significant 4 bits of the number. STA 2502 H stores the result in 2502 H. HLT stops the program.

6.19 TO FIND SQUARE FROM LOOK-UP TABLE

Example. Find square of 07 (decimal) using look-up table technique.

The number 07 D is in the memory location 2500 H.

The result is to be stored in the memory location 2501 H.

The table for square is stored from 2600 to 2609 H.

The address of the memory locations is in hexadecimal.

The number and squares are to be entered in decimal.

The squares of data are stored in certain memory locations in the tabular form. This is called look-up table. For this example the squares of numbers from 00 to 09 are stored in memory locations 2600 to 2609 H. The values of squares are in decimal. The data form the index and it is transferred from memory to the accumulator and then to register L. It forms the LSBs of the memory location where square of the data is placed. The MSBs of the address is moved to register H. Now, the address of the desired memory location where the square of the data resides is in H-L pair. The square of the data is now moved to the accumulator and then it is stored.

In the program LDA 2500 H moves data (07) to the accumulator. MOV L, A moves 07 to register L. MVI H, 26 H moves 26 H in register H. Now, in H-L pair 2607 H is residing which is the address of the memory location where the square of 07 D is placed. MOV A, M transfers 49 D from memory location 2607 to the accumulator. This is stored in 2501 H.

PROGRAM

Address	*Machine Codes*	*Mnemonics*	*Operands*	*Comments*
2000	3A, 00, 25	LDA	2500 H	Get data in accumulator.
2003	6F	MOV	L, A	Get data in register L.
2004	26, 26	MVI	H, 26 H	Get 26 in register H.
2006	7E	MOV	A, M	Square of data in accumulator.
2007	32, 01, 25	STA	2501 H	Store square in 2501 H.
200A	76	HLT		Stop

DATA

2500 — 07 D

Result

2501 — 49 D

Look-up Table

Address (Hex)	2600	2601	2602	2603	2604	2605	2606	2607	2608	2609
Square (Decimal)	00	01	02	09	16	25	36	49	64	81

6.20 TO FIND LARGER OF TWO NUMBERS

Example 1. Find the larger of 98 H and 87 H.

The first number 98 H is placed in the memory location 2501 H.

The 2nd number 87 H is placed in the memory location 2502 H.

The result is stored in the memory location 2503 H.

The numbers are represented in hexadecimal system. The 1st number is moved from its memory location to the accumulator. It is compared with 2nd number. The larger of the two is then placed in the accumulator. From the accumulator the larger number is moved to the desired memory location.

PROGRAM

Memory address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
2000	21, 01, 25		LXI	H, 2501 H	Address of 1st number in H-L pair.
2003	7E		MOV	A, M	1st number in accumulator.
2004	23		INX	H	Address of 2nd number in H-L pair.
2005	BE		CMP	M	Compare 2nd number with 1st number. Is the 2nd number > 1st ?
2006	D2, 0A, 20		JNC	AHEAD	No, larger number is in accumulator. Go to AHEAD.
2009	7E		MOV	A, M	Yes, get 2nd number in accumulator.
200A	32, 03, 25	AHEAD	STA	2503 H	Store larger number in 2503 H.
200D	76		HLT		Stop

Example 1	Example 2
DATA	**DATA**
2501 — 98 H	2501 — A9 H
2502 — 87 H	2502 — EB H

6.24 TO FIND THE SMALLEST NUMBER IN A DATA ARRAY

Example 1. The numbers of a series are: 86, 58 and 75.

As there are three numbers in the series, count = 03.

The count is placed in the memory location 2500 H.

The numbers are placed in the memory location 2501 to 2503 H.

The result is to be stored in the memory location 2450 H.

The 1st number of the series is placed in the accumulator and it is compared with the 2nd number which is in the memory. The smaller of the two is placed in the accumulator. Again this number which is in the accumulator is compared with the 3rd number of the series and smaller number is placed in the accumulator. This process of comparison is repeated till all the numbers of the series are compared and the smallest number is stored in the desired memory location.

The memory location 2500 H contains the count of the series. In Example 1 count is 03. 2500 is placed in H-L pair. The instruction MOV C, M places the count in register C. INX H increases the content of H-L pair from 2500 to 2501 H. The 1st number of the series resides in the memory location 2501 H. MOV A, M moves the 1st number (86) in the accumulator. DCR C decreases the count from 03 to 02, which means that now there are 2 more numbers in the series. INX H increases the content of H-L pair from 2501 to 2502 H. The 2nd number of the series is in 2502 H. CMP M compares the content of the accumulator (1st number) with the content of memory location 2502 H (2nd number). Since the 1st number 86 is greater than the 2nd number 58, there will be no carry. As there is no carry, the instruction MOV A, M will be executed. The smaller number 58 will be placed in the accumulator. DCR C will decrease count from 02 to 01. It indicates that there is one more number left in the series. As the content of register C is not zero, the program will jump to the label LOOP. The INX H will change the content of H-L pair from 2502 to 2503 H. The 3rd number of the series is in 2503 H. Again CMP M will compare the content of the accumulator (58) with the content of 2503 H. The content of 2503 H is 75 which is greater than the content of the accumulator. In this case there will be a carry. After the execution of the instruction JC AHEAD the program will jump to the label AHEAD. The content of the accumulator will remain as it is. DCR C will reduce the count from 01 to 00. As the content of register C is zero, the program will not jump. It will proceed further to execute STA 2450 H. The smallest number (58) which is in the accumulator will be stored in the memory location 2450 H. The instruction HLT will end the program.

PROGRAM

Memory Address	*Machine Codes*	*Labels*	*Mnemonics*	*Operands*	*Comments*
2000	21, 00, 25		LXI	H, 2500 H	Get address for count in H-L pair.
2003	4E		MOV	C, M	Count in register C.
2004	23		INX	H	Get address of 1st number in H-L pair.
2005	7E		MOV	A, M	1st number in accumulator.
2006	0D		DCR	C	Decrement count.
2007	23	LOOP	INX	H	Address of next number in H-L pair.
2008	BE		CMP	M	Compare next number with previous smallest. Is previous smallest less than next number?

2009	DA, 0D, 20		JC	AHEAD	Yes, smaller number in accumulator. Go to AHEAD.
200C	7E		MOV	A, M	No, get next number in accumulator.
200D	0D	AHEAD	DCR	C	Decrement count.
200E	C2, 07, 20		JNZ	LOOP	
2011	32, 50, 24		STA	2450 H	Store smallest number in 2450 H.
2014	76		HLT		Stop.

Example 1
DATA
2500 — 03 H
2501 — 86 H
2502 — 58 H
2503 — 75 H
Result
2450 — 58 H

Example 2
DATA
2500 — 05 H
2501 — EB H
2502 — D4 H
2503 — 3C H
2504 — 0F H
2505 — A8 H
Result
2450 — 0F H

6.24.1 An Alternative Program to Find the Smallest Number from a Series of Numbers

Example 1. The numbers of a series are: 86, 58 and 75. As there are three numbers in the series, count = 03. The count is stored in the memory location 2500 H. The numbers are stored in the memory location 2501 to 2503 H.

The result is to be stored in the memory location 2450 H.

PROGRAM

Memory Address	*Machine Codes*	*Label*	*Mnemonics*	*Operands*	*Comments*
2000	21, 00, 25		LXI	H, 2500 H	Get address for count in H-L pair.
2003	4E		MOV	C, M	Count in register C.
2004	3E, FF		MVI	A, FF	Get FF in accumulator.
2006	23	LOOP	INX	H	Address of the next number of the series.
2007	BE		CMP	M	Compare next number with previous smallest. Is next number < previous smallest?
200A	DA, 0C, 20		JC	AHEAD	No, smallest number is in accumulator. Go to AHEAD.
200B	7E		MOV	A, M	Yes, get smaller number in accumulator.
200C	0D	AHEAD	DCR	C	Decrement count.
200D	C2, 06, 20		JNZ	LOOP	
2010	32, 50, 24		STA	2450 H	Store result in 2450 H.
2013	76		HLT		Stop.

Example 1
DATA
2500 — 03 H
2501 — 86 H

Example 2
DATA
2500 — 05 H
2501 — EB

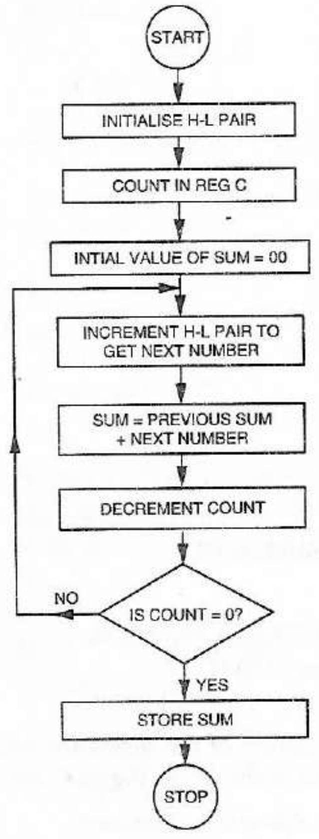


Fig. 6.4 Flow Chart to find sum of a series of 8-Bit numbers; sum: 8-bit

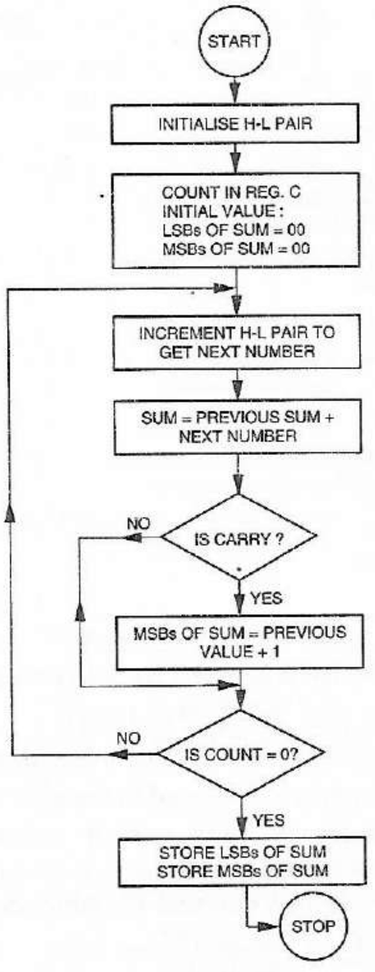


Fig. 6.5 Flow Chart to find sum of a series of 8-bit numbers; sum: 16-bit

6.27 SUM OF A SERIES OF 8-BIT NUMBERS, SUM; 16-BIT

Example 1. Add 45, 98, 8A and E2 H; these are hexadecimal numbers.

The numbers are placed in the memory locations 2501 to 2504 H, and the count in 2500 H.

The sum is to be stored in the memory locations 2450 and 2451 H.

As there are 4 numbers in the series, count = 04. The initial value of the sum is made 00.

The number of the series are taken one by one and added to the sum. The program flow chart is shown in Fig. 6.5.

PROGRAM

Memory Address	*Machine Codes*	*Label*	*Mnemonics*	*Operands*	*Comments*
2400	21, 00, 25		LXI	H, 2500 H	Address of count in H-L pair.
2403	4E		MOV	C, M	Count in register C.
2404	3E, 00		MVI	A, 00	LSBs of sum = 00 (initial value).
2406	47		MOV	B, A	MSBs of sum = 00 (initial value).

2407	23	LOOP	INX	H	Address of next data in H-L pair.
2408	86		ADD	M	Previous sum + next number.
2409	D2, 0D, 24		JNC	AHEAD	Is carry ? No, go to AHEAD.
240C	04		INR	B	Yes, add carry to MSBs of sum.
240D	0D	AHEAD	DCR	C	Decrement count.
240E	C2, 07, 24		JNZ	LOOP	Is count = 0 ? No, jump to LOOP.
2411	32, 50, 24		STA	2450 H	Store LSBs of the sum in 2450 H.
2414	78		MOV	A, B	Get MSBs of sum in accumulator.
2415	32, 51, 24		STA	2451 H	Store MSBs of sum in 2451 H.
2418	76		HLT		Stop.

Example 1

DATA

2500 — 04 H
2501 — 45 H
2502 — 98 H
2503 — 8A H
2504 — E2 H

Result

2450 — 49 H, LSBs of sum.
2451 — 02 H, MSBs of sum.

Example 2

DATA

2500 — 05 H
2501 — 98 H
2502 — 24 H
2503 — 35 H
2504 — 46 H
2505 — 39 H

Result

2450 — 70 H, LSBs of sum.
2451 — 01 H, MSBs of sum.

The count is placed in register C. The LSBs of the sum reside in the accumulator, and MSBs of the sum in register B. Initial values of the LSBs and MSBs of the sum are made 00. The 1st number is added to the initial sum. Count is decreased by one and the program jumps to LOOP. The address of the 2nd number is placed in H-L pair. ADD M adds 2nd number to the previous sum which is in the accumulator. If there is any carry after addition, that is stored in register B. This is nothing but the MSB of the sum. The process is repeated and the remaining numbers of the series are added. The LSBs of the sum is stored in memory location 2450 H and MSBs of the sum in 2451 H.

6.28 SUM OF A SERIES OF 8-BIT DECIMAL NUMBERS, SUM: 16-BIT

Example 1. Add 65, 46, 35 and 98 D, these are decimal numbers. D stands for decimal. Sum = 0244 D

The numbers are placed in the memory locations 2501 to 2504 H.

The sum is to be stored in the memory locations 2450 and 2451 H.

The instruction DAA is used in the program for decimal adjustment. It is used after ADD instruction. Details for DAA instruction have already been given in Section 6.6 under the heading of "Decimal Addition of Two 8-Bit Number."

PROGRAM

Memory Address	*Machine Codes*	*Label*	*Mnemonics*	*Operands*	*Comments*
2400	21, 00, 25		LXI	H, 2500 H	Address of count in H-L pair.
2403	4E		MOV	C, M	Count in register C.
2404	3E, 00		MVI	A, 00	LSBs of the sum = 00 (initial value).
2406	47		MOV	B, A	MSBs of sum = 00 (initial value).
2407	23	LOOP	INX	H	Address of next data in H-L pair.
2408	86		ADD	M	Sum = Sum + data.

```
2409    27                 DAA              Decimal adjust.
240A    D2, 0E, 24         JNC    AHEAD     Is carry? No, go to AHEAD.
240D    04                 INR    B         Yes, add carry to MSBs of sum.
240E    0D         AHEAD   DCR    C         Decrement count.
240F    C2, 07, 24         JNZ    LOOP      Is count = 0? No, jump to LOOP.
2412    32, 50, 24         STA    2450 H    Store LSBs of sum in 2450 H.
2415    78                 MOV    A, B      Get MSBs of sum in accumulator.
2416    32, 51, 24         STA    2451 H    Store MSBs of sum in 2451 H.
2419    76                 HLT              Stop.
```

Example 1

DATA

2500 — 04 H
2501 — 65 D
2502 — 46 D
2503 — 35 D
2504 — 98 D

Result

2450 — 44 D, LSDs of the sum.
2451 — 02 D, MSDs of the sum.

Example 2

DATA

2500 — 05 H
2501 — 96 D
2502 — 98 D
2503 — 85 D
2504 — 89 D
2505 — 93 D

Sum = 0461
2450 — 61 D, LSDs of the sum.
2451 — 04 D, MSDs of the sum.

The program is exactly same as that for "sum of a series of 8-Bit Numbers, sum 16-bit," Section 6.27; except that the instruction DAA has been incorporated after the instruction ADD M. The instruction ADD M gives the sum in hexadecimal system. The instruction DAA makes correction and gives the result in decimal system. Similarly, the program for "sum of a series of 8-bit decimal numbers, sum 8-bit" can be obtained incorporting the instruction DAA in the program for "sum of a series of 8-bit Numbers, sum 8-bit" given in Section 6.26.

6.29 8-BIT MULTIPLICATION : PRODUCT 16-BIT

Decimal Multiplication. For decimal multiplication the following procedure is followed:

Example

```
      36 D, Multiplicand
    × 29 D, Multiplier
    ------------------
     324
     72
    ------------------
    1044 D
```

Product: 1044 D

D stands for decimal number

$$36 \times 29 = 9 \times 36 + 20 \times 36$$

$$324 + 720 = 1044\text{ D}.$$

First of all the multiplicand 36 is multiplied by 9 and the result 324 is written on the paper. Again 36 is multipled by 2 and the result 72 is shifted left to make it 720. 324 and 720 are added. Either 72 is shifted left or 324 right with respect to 72 the final result will be same.

Binary Multiplication

Example. Multiply 7 by 5

First of all 7 and 5 are represented in binary form;

```
            0111 = 7, Multiplicand
       ×   0101 = 5, Multiplier
      ---------------------------
              0111
             0000
            0111
           0000
      ---------------------------
      Product =   010, 0011 = 35
```

It should be noted that the above product is in the binary form. It is not in BCD.

In binary multiplication we see that when a multiplicand is multipled by 1 the product is equal to the multiplicand. When a multiplicand is multiplied by zero, the product is zero. The procedure for multiplication is that first the multiplicand is multipled by the LSB of the multiplier and the partial product is stored and shifted right. Again, the multiplicand is multipled by the 2nd bit and the result is added to the previous shifted partial product. The procedure is repeated. If the bit of the multiplier is 1 the multiplicand is added to the previous partial product. In case of 0 bit there is nothing to be added to the partial product but it will be simply shifted right by one bit. In case of binary multiplication by computer if the partial product is shifted left instead of right, and we take bits of multiplier from MSB side instead of LSB side the final product will remain same.

Conclusion. The conclusion is that each bit of multiplier is taken one by one and it is checked whether it is 1 or 0. If the bit of the multipler is one the multiplicand is added to the product and the product is shifted left. If the bit of multipler is zero, the product is simply shifted left by one bit.

Example 1. Multiply 84 H by 56 H.

In this example multiplicand is 84 H. It is extended to 16-bits and stored in the two consecutive memory locations 2501 and 2502 H. The multiplier is 56 H and it is stored in 2503 H. The product is a 16-bit number and it is stored in 2504 H and in 2505 H. Data and results are in hexadecimal. The program flow chart is shown in Fig. 6.6.

PROGRAM

Memory Address	*Machine Codes*	*Label*	*Mnemonics*	*Operands*	*Comments*
2000	2A, 01, 25		LHLD	2501 H	Get multiplicand in H-L pair.
2003	EB		XCHG		Multiplicand in D-E pair.
2004	3A, 03, 25		LDA	2503 H	Multiplier in accumulator.
2007	21, 00, 00		LXI	H, 0000	Initial value of product = 00 in H-L pair.
200A	0E, 08		MVI	C, 08	Count = 8 in register C.
200C	29	LOOP	DAD	H	Shift partial product left by 1 bit.
200D	17		RAL		Rotate multiplier left one bit. Is multipler's bit = 1?
200E	D2, 12, 20		JNC	AHEAD	No, go to AHEAD.
2011	19		DAD	D	Product = Product + Multiplicand.
2312	0D	AHEAD	DCR	C	Decrement count.
2013	C2, 0C, 20		JNZ	LOOP	
2016	22, 04, 25		SHLD	2504 H	Store result.
2019	76		HLT		Stop.

Example 1
DATA
2501 — 84 H, LSBs of multiplicand.
2502 — 00, MSBs of multiplicand.
2503 — 56 H, Multiplier.
Result
2504 — 58 H, LSBs of product.
2505 — 2C H, MSBs of product.

Example 2
DATA
2501 — 8A H, LSBs of multiplicand.
2502 — 00, MSBs of multiplicand.
2503 — 52 H, Multiplier.
Result
2504 — 34 H, LSBs of product.
2505 — 2C H, MSBs of product.

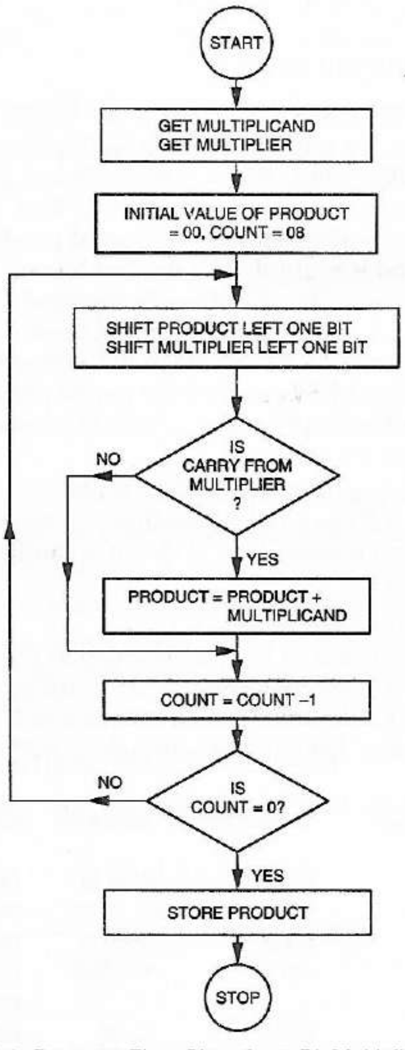

START
GET MULTIPLICAND
GET MULTIPLIER
INITIAL VALUE OF PRODUCT = 00, COUNT = 08
SHIFT PRODUCT LEFT ONE BIT
SHIFT MULTIPLIER LEFT ONE BIT
IS CARRY FROM MULTIPLIER ?
NO
YES
PRODUCT = PRODUCT + MULTIPLICAND
COUNT = COUNT –1
IS COUNT = 0?
NO
YES
STORE PRODUCT
STOP

Fig. 6.6 Program Flow Chart for 8-Bit Multiplication

The instruction LHLD 2501 H transfers 16-bit multiplicand from the memory locations 2501 and 2502 H to H-L pair. By the execution of the instruction XCHG the contents of H-L pair are exchanged with the contents of D-E pair. Thus the multiplicand is placed in D-E pair. The instruction LDA 2503 H transfers multiplier from the memory location 2503 H to the accumulator. LXI H, 0000 makes the initial value of the product equal to zero and it is placed in H-L pair. The count is equal to the number of bits of the multiplier. In this case it is 08 and it is placed in register C. DAD H is an instruction for 16-bit addition. It adds the contents of H-L pair to itself. Thus, the partial product which is in H-L pair is shifted left by one bit. RAL rotates the content of the accumulator left by one bit. The accumulator contains multiplier and hence it is rotated left by one bit. The instruction DAD D adds the content of D-E pair and H-L pair and places the result in H-L pair. D-E pair and H-L pair contain multiplicand and partial product respectively. Thus, the execution of the instruction DAD D adds the multiplicand to the partial product and places the sum which is the new partial product in H-L pair. The instruction DAD D is executed only when the bit of the multiplicand under consideration is one, otherwise it is not executed. To get the result the program moves in the LOOP 8 times as there are 8-bit in the multiplier.

6.30 8-BIT DIVISION

The computer performs division by trial subtractions. The divisor is subtracted from the 8 most significant bits of the dividend. If there is no borrow, the bit of the quotient is set to 1; otherwise 0. To line up the dividend and quotient properly the dividend is shifted left by one bit before each trial of subtraction. The dividend and quotient share a 16-bit register. Due to shift of dividend one bit of the register falls vacant in each step. The quotient is stored in vacant bit positions. The program flow chart is shown in Fig. 6.7.

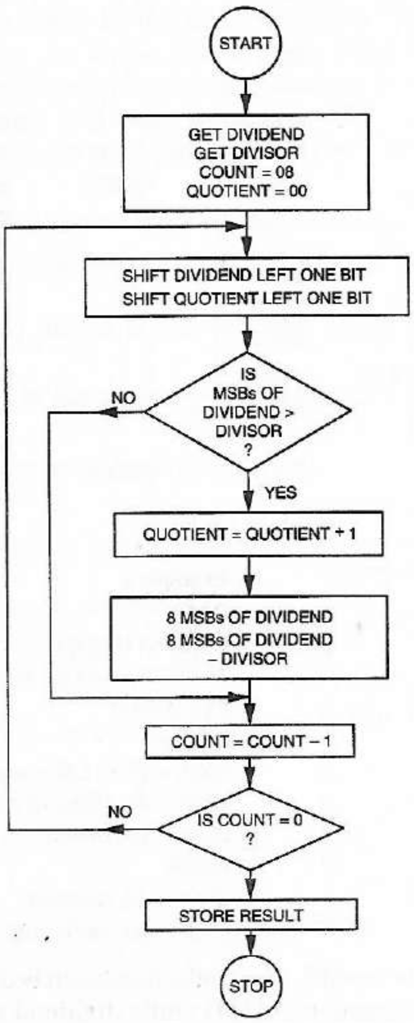


Fig. 6.7 Program Flow Chart for 8-Bit Division

Example 1. Divide 489B by 1A.

These are hexadecimal numbers. The dividend is a 16-bit number and divisor 8-bi number. If the dividend of a problem is an 8-bit number, it is extended to a 16-bi number by placing zeros in MSBs positions.

The dividend is placed in the memory locations 2501 and 2502 H.

The divisor is placed in the memory location 2503 H.

The results are stored in the memory locations 2504 and 2505 H.

The quotient is stored in the memory locations 2504 H.

The remainder is stored in the memory location 2505 H.

PROGRAM

Memory Address	*Machine Codes*	*Label*	*Mnemonics*	*Operands*	*Comments*
2400	2A, 01, 25		LHLD	2501 H	Get dividend in H-L pair.
2403	3A, 03, 25		LDA	2503 H	Get divisor from 2503 H.
2406	47		MOV	B, A	Divisor in register B.
2407	0E, 08		MVI	C, 08	Count = 08 in register C.
2409	29	LOOP	DAD	H	Shift dividend and quotient le by one bit.
240A	7C		MOV	A, H	Most significant bits of dividen in accumulator.
240B	90		SUB	B	Subtract divisor from mos significant bits of dividend.
240C	DA, 11, 24		JC	AHEAD	Is most significant part o dividend > divisor? No, go t AHEAD.
240F	67		MOV	H, A	Most significant bits of dividen in register H.
2410	2C		INR	L	Yes, add 1 to quotient.
2411	0D	AHEAD	DCR	C	Decrement count.
2412	C2, 09, 24		JNZ	LOOP	Is count = 0? No, jump t LOOP.
2415	22, 04, 25		SHLD	2504 H	Store quotient in 2504 an remainder in 2505 H.
2418	76		HLT		Stop.

Example 1

DATA

2501 — 9B H, LSBs of dividend.

2502 — 48 H, MSBs of dividend.

2503 — 1A H, Divisor.

Result

2504 — F2, Quotient.

2505 — 07, Remainder.

Example 2

DATA

Divide 54 H by 09.

The dividend 54 is extended to 16 bits.

54 = 0054 H

DATA

2501 — 54 H, LSBs of dividend.

2502 — 00, MSBs of dividend.

2503 — 09, Divisor.

Result

2504 — 09, Quotient

2505 — 03, Remainder

The count in register C is kept 08. The trial subtraction is done 8 times and an 8-bi quotient is obtained. The instruction DAD H shifts dividend and quotient left by one bit. Due to shift of dividend the bit positions in register L fall vacant. In the vacant bit positions quotient is stored. Note that the dividend is shifted prior to trial subtraction. The MSB of the dividend should be zero, otherwise it will be shifted to carry bit. If a problem contains MSB not equal to zero, it will be solved by splitting it in two parts. Shifting of dividend before subtraction is not done in ordinary division by pen and paper, but the computer method gives correct result as the numbers are represented in binary coded hexadecimal system. You can check this by taking simple numerical examples.

6.31 MULTIBYTE ADDITION

Example 1. 3 A 9 C 8 A 6 7 H, 1st number
+ 9 B 4 7 6 C 8 B H, 2nd number
D 5 E 3 F 6 F 2 H, Sum

A byte consists of 8-bits. In the above example two multibyte hex numbers are to be added. Each number consists of 4 bytes. An 8-bit microcomputer takes one byte of the numbers at a time and adds them with carry. A counter is initiated to count the byte. In Example 1, the count = 4.

The count is placed in the memory location 2500 H.

The 1st number is placed in the memory locations 2501 to 2504 H.

The 2nd number is placed in the memory locations 2601 to 2604 H.

The sum is placed in the memory locations 2501 to 2504 H.

The program flow chart is shown in Fig. 6.8.

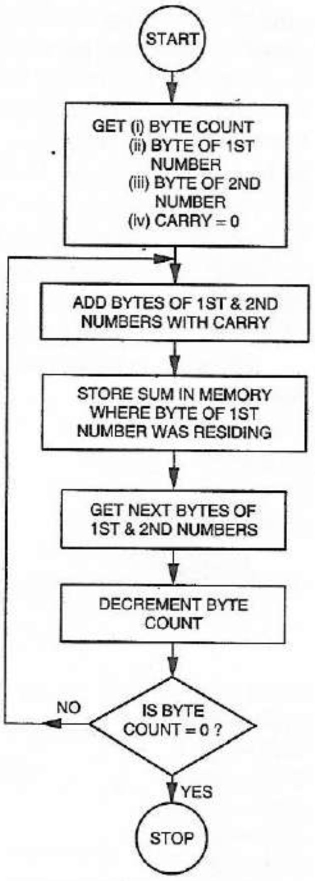

START
GET (i) BYTE COUNT
(ii) BYTE OF 1ST NUMBER
(iii) BYTE OF 2ND NUMBER
(iv) CARRY = 0
ADD BYTES OF 1ST & 2ND NUMBERS WITH CARRY
STORE SUM IN MEMORY WHERE BYTE OF 1ST NUMBER WAS RESIDING
GET NEXT BYTES OF 1ST & 2ND NUMBERS
DECREMENT BYTE COUNT
IS BYTE COUNT = 0 ?
NO
YES
STOP

Fig. 6.8 Program/Flow Chart for Multibyte Addition

PROGRAM

Memory Address	*Machine Codes*	*Label*	*Mnemonics*	*Operands*	*Comments*
2400	21, 00, 25		LXI	H, 2500 H	Address of byte count in H-L pair.
2403	4E		MOV	C, M	Byte count in register C.
2404	23		INX	H	Address of 1st byte of 1st number.
2405	11, 01, 26		LXI	D, 2601 H	Address of 1 st byte of 2nd number.
2408	B7		ORA	A	Clear carry.
2409	1A	LOOP	LDAX	D	Get byte of 2nd number in accumulator.
240A	8E		ADC	M	Byte of 2nd number + byte of 1st number + carry.
240B	77		MOV	M, A	Store sum in memory addressed by IH-L pair.
240C	23		INX	H	Increment the content of H-L pair.

240D	13	INX	D	Increment the content of D-E pair
240E	0D	DCR	C	Decrement count.
240F	C2, 09, 24	JNZ	LOOP	Is count = 0? No, jump to LOOP.
2412	76	HLT		Stop.

Example 1

DATA

```
2500 — 04
2501 — 67          2601 — 8B
2502 — 8A          2602 — 6C
2503 — 9C          2603 — 47
2504 — 3A          2604 — 9B
```

The sum is stored in the memory locations 2501 to 2504 H.

Result

```
2501 — F2
2502 — F6
2503 — E3
2504 — D5
```

Example 2

```
  00B8968B7E
  009C8A5C46
  ----------
  015520E7C4
```

DATA

```
2500 — 05
2501 — 7E          2601 — 46
2502 — 8B          2502 — 5C
2503 — 96          2603 — 8A
2504 — B8          2604 — 9C
2505 — 00          2605 — 00
```

Result

```
2501 — C4
2502 — E7
2503 — 20
2504 — 55
2505 — 01
```

Example 3

```
  00156987 H
  00982142 H
  ----------
  00AD8AC9 H
```

DATA

```
2500 — 04
2501 — 87          2601 — 42
2502 — 69          2602 — 21
2503 — 15          2603 — 98
2504 — 00          2604 — 00
```

Result

```
2501 — C9
2502 — 8A
2503 — AD
2504 — 00
```

Books for Study & Reference:

1. Fundamentals of Microprocessors and Microcontrollers by B. RAM

   DHANPAT RAI Publications (P) Ltd., New Delhi

2. Microprocessor Architecture, programming and application with 8085 by R. Gaonkar

   Penram International Publishing, Mumbai