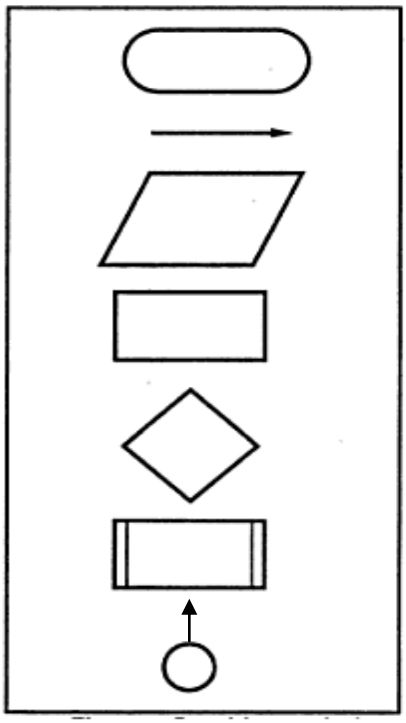
Unit II Assembly Language Programs (8085 only)

Flow chart

- The thinking process described here and the steps necessary to write the program can be represented in a pictorial format, called a flowchart.
- Generally, a flowchart is used for two purposes: to assist and clarify the thinking process and to communicate the programmer's thoughts or logic to others
- Symbols commonly used in flowcharting are shown in Figure



Oval : It indicates start or stop operation.

Arrow : It indicates flow with direction.

Parallelogram : It indicates input/output operation.

Rectangle : It indicates process operation.

Diamond : It indicates decision making operation.

Double sided Rectangle : It indicates execution of pre-defined process (subroutine).

Circle with an arrow: Represents continuation (an entry or exit) to a different page

DATA TRANSFER (COPY) OPERATIONS

- One of the primary functions of the microprocessor is copying data, from a register(or I/O or memory) called the source, to another register (or I/O or memory) called the destination
- The contents of the source are not transferred, but are copied into the destination register without modifying the contents of the source.
-

Several instructions are used to copy data. This section is concerned with the following operations.

MOV: Move -Copy a data byte.

MVI: Move Immediate -Load a data byte directly.

OUT: Output to Port -Send a data byte to an output device.

IN: Input from Port -Read a data byte from an input device.

- The term *copy* is equally valid for input/output functions because the contents of the source are not altered.
- However, the term *data transfer* is used so commonly to indicate the data copy function that, these terms are used interchangeably when the meaning is not ambiguous.
- In addition to data copy instructions, it is necessary to introduce two machinecontrol operations to execute programs.

HLT: Halt Stop processing and wait.

NOP: No Operation Do not perform any operation.

DATA MANIPULATIONOPERATIONS:

1. ARITHMETIC OPERATIONS

ADD: Add Add the contents of a register.*

ADI: Add Immediate Add 8-bit data.

SUB: Subtract Subtract the contents of a register.

SUI: Subtract Immediate Subtract 8-bit data.

INR: Increment Increase the contents of a register by 1.

DCR: Decrement Decrease the contents of a register by 1.

Addition

- The 8085 performs addition with 8-bit binary numbers and stores the sum in the accumulator.
- If the sum is larger than eight bits (FFH), it sets the Carry flag.
- Addition can be performed either by adding the contents of a source register (B, C, D, E, **H,** L, or memory) to the contents of the accumulator (ADD) or by adding the second byte directly to the contents of the accumulator (ADI).

Subtraction

- The 8085 performs subtraction by using the method of 2's complement.
- Subtraction can be performed by using either the instruction SUB to subtract the contents of a source register or the instruction SUI to subtract an 8-bitnumber from the contents of the accumulator. In either case, the accumulator contents are regarded as the minuend (the number from which to subtract).

The8085performs the following steps internally to execute the instruction SUB (or *SUI).*

Step1: Convertssubtrahend (the number to be subtracted) into its l's complement.

Step2: Adds 1 to l's complement to obtain 2's complement of the subtrahend.

Step3: Add 2's complement to the minuend (the contents of the accumulator).

Step4: Complementsthe Carry flag.

2. LOGIC OPERATIONS

- A microprocessor is basically a programmable logic chip.
- It can perform all the logic functions of the hard-wired logic through its instruction set.
- The 8085 instruction set includes such logic functions as AND, OR, Ex OR, and NOT (complement). The opcodes of these operations are as follows:*

ANA: AND Logically AND the contents of A- register

ANI: AND Immediate Logically AND 8-bit data.

ORA: OR Logically OR the contents of A- register.

ORI: OR Immediate Logically OR 8-bit data.

XRA: X-OR Exclusive-OR the contents of A- register.

XRI : X-OR Immediate Exclusive-OR 8-bit data.

All logic operations are performed in relation to the contents of the accumulator.

OR, Exclusive-OR, and NOT

The instruction ORA (and ORI) simulates logic ORing with eight 2-input OR gates; this process is similar to that of AND ing. The instruction XRA (and XRI)performsExc1usive-ORingof eight bits and the instruction CMA invert the bits of the accumulator.

BRANCH OPERATIONS

- The branch instructions are the most powerful instructions because they allow the microprocessor to change the sequence of a program, either unconditionally or under certain test conditions.
- These instructions are the key to the flexibility and versatility of a computer.
- The microprocessor is a sequential machine; it executes machine codes from one memory location to the next.
- Branch instructions instruct the microprocessor to go to a different memory location, and the microprocessor continues executing machine codes from that new location.
- The address of the new memory location is either specified explicitly or supplied by the microprocessor or by extra hardware.
- The branch instructions are classified in three categories:
    1. Jump instructions
    2. Call and Return instructions
    3. Restart instructions

- The Jump instructions specify the memory location explicitly.

- They are 3-byte instructions: one byte for the operation code, followed by a 16-bit memory address.

- Jump instructions are classified into two categories: **Unconditional Jump and Conditional Jump.**

Unconditional Jump

The 8085 instruction set includes one unconditional Jump instruction. The unconditional Jump instruction enables the programmer to set up continuous loops.

Example:
JMP 8500

Description
- This is a 3-byte instruction
- The second and third bytes specify the 16 bit memory address. However, the second byte specifies the low-order and the third byte specifies the high-order memory address

Conditional Jumps

- Conditional Jump instructions allow the microprocessor to make decisions based on certain conditions indicated by the flags.
- After logic and arithmetic operations, flip-flops (flags) are set or reset to reflect data conditions.
- The conditional Jump instructions check the flag conditions and make decisions to change or not to change the sequence of a program.
- Four flags used by the Jump instructions are
  1. Carry flag2. Zero flag3. Sign flag4. Parity flag

Instruction:
All conditional Jump instructions in the 8085 are 3-byte instructions; the second byte specifies the low-order (line number) memory address, and the third byte specifies the high-order (page number) memory address.

Example:

JC 8500 - Jump on Carry (if result generates carry and CY=1)

JNC 8500 - Jump on No Carry (CY =0)

JZ 8500 - Jump on Zero (if result is zero and Z = 1)

JNZ 8500 - Jump on No Zero (Z =0)

Assembly language program for 8085 microprocessor:

1. **8 bit – Addition**

LABLE	MNEMONICS	COMMENT
	LDA 8200_H	Get first data in A register
	MOV B,A	Move A to B
	LDA 8201_H	Get second data in A register
	MVI C,00_H	Clear C register
	ADD B	Add B and A and store in A register
	JNC AHEAD	If carry is 0 go to AHEAD
	INR C	If carry is 1 increment C register
AHEAD	STA 8202_H	Store the sum on memory
	MOV A,C	Move the content C to A register
	STA 8203_H	Store the carry in memory
	HLT	halt program execution

2. 8 bit – Subtraction

LABLE	MNEMONICS	COMMENT
	MVI C,00_H	Move the immediate data 00_h into the C register
	LDA 9000_H	Load the content of 9000_h into A register
	MOV B,A	Copy the content of A to B
	LDA 9001_H	Load the content of 9001_H into A register
	SUB B	Subtract the content of B from the accumulator content
	JNC L1 ($800E_H$)	Jump on to L1 , if there is no carry
	INR C	Increment the content of C reg by 1
L1	STA 8500_H	Store accumulator content in the memory
	MOV A,C	Copy the content of C to A register
	STA 8501_H	Store the accumulator content in the memory
	HLT	halt program execution

3. 8 bit – Multiplication

LABLE	MNEMONICS	COMMENT
	MVI D, 00_H	Move the immediate data 00_h into the D register
	LDA 8500_H	Load the content of 8500_h into A register
	MOV B,A	Copy the content of A to B
	LDA 8501_H	Load the content of 8501_H into A register
	MOV C,A	Copy the content of A to C
	XRA A	Clear the accumulator
L2	ADD B	Add the content of A with B
	JNC L1(8010_H)	Jump on to L1 , if there is no carry
	INR D	Increment the content of D register by 1
L1	DCR C	Decrement the content of C register by 1
	JNZ L2($800B_H$)	Jump on to L2 , if there is no 0
	STA 9000_H	Store the accumulator content in the memory
	MOV A,D	Copy the content of D to A register
	STA 9001_H	Store the accumulator content in the memory
	HLT	halt program execution

4. 8bit - Division

LABLE	MNEMONICS	COMMENT
	MVI C, 00_H	Move the immediate data 00_h into the C register
	LDA 8500_H	Load the content of 8500_h into A register
	MOV B,A	Copy the content of A to B
	LDA 8501_H	Load the content of 8501_H into A register
L2	CMP B	Compare accumulator value with B value
	JC L1(8012_H)	Jump on to L1 , if there is no carry
	SUB B	Subtract the content of B from the accumulator content
	INR C	Increment the content of C register by 1
	JMP L2	Jump on to L2, without any condition
L1	STA 9000_H	Store the accumulator content in the memory
	MOV A,C	Move C to A register
	STA 9001_H	Store the accumulator content in the memory
	HLT	halt program execution

5. Ascending order

(Write an program to sort on array of data in the Ascending order. The array is stored in the memory starting from 4200H the first element of the array gives the count value for the number of elements in the array)

LABLE	MNEMONICS	COMMENT
	LDA 4200H	Load the count value in A-register.
	MOV B, A	Set count for N-1 repetition
	DCR B	of N-1comparison
LOOP 2	LXI H,4200H	Set pointer for array
	MOV C, M	Setv count for N-1 comparisons
	DCR C	
	INX H	Increment the pointer
LOOP 1	MOV A, M	get one data of array in A-register.
	INX H	
	CMP M	Compare the next data of array with content of A-register.
	JC AHEAD	If content of A is Less than memory, then go to

		AHEAD
	MOV D, M	If the content of A is greater than content of memory, then exchange the content of memory pointed by HL and previous memory location
	MOV M,A	
	DCX H	
	MOV M , D	
	INX H	
AHEAD	DCR C	
	JNZ LOOP 1	Repeat comparison until C- count is zero
	DCR B	
	JNZ LOOP 2	Repeat N-1 comparison until B- count is zero
	HLT	halt program execution

6. Descending order

(Write an program to sort on array of data in the Descending order. The array is stored in the memory starting from 4200H the first element of the array gives the count value for the number of elements in the array)

LABLE	MNEMONICS	COMMENT
	LDA 4200H	Load the count value in A-register.
	MOV B, A	Set count for N-1 repetition
	DCR B	of N-1comparison
LOOP 2	LXI H,4200H	Set pointer for array
	MOV C, M	Setv count for N-1 comparisons
	DCR C	
	INX H	Increment the pointer
LOOP 1	MOV A, M	get one data of array in A-register.
	INX H	
	CMP M	Compare the next data of array with the content of A-register.
	JNC AHEAD	If content of A is greater than content of memory addressed by HL pair, then go to AHEAD
	MOV D, M	If the content of A is less than content of memory addressed by HL pair, then exchange content of memory pointed by HL and previous memory location
	MOV M,A	
	DCX H	
	MOV M , D	
	INX H	
AHEAD	DCR C	
	JNZ LOOP 1	Repeat comparison until C- count is zero
	DCR B	
	JNZ LOOP 2	Repeat N-1 comparison until B- count is zero
	HLT	halt program execution

7. Search for Smallest data in an array

(Write an assembly language program to search the smallest data in an array of N data stored in memory from 4200H to (4200H+N). The first element of the array gives the number of data in the array)

LABLE	MNEMONICS	COMMENT
	LXIH, 4200H	set pointer for array
	MOV B, M	set count for no. of elements in array
	INX H	
	MOV A , M	Set first element of array as smallest data
	DCR B	Decrement the count
LOOP	INX H	
	CMP M	Compare an element of array with current smallest data
	JC AHEAD	If CF =1, go to AHEAD
	MOV A, M	If CF =0, Then content of memory is smaller than A-register. Hence, if CF = 0, make memory as smallest by moving to A - register
AHEAD	DCR B	
	JNZ LOOP	Repeat comparison until count is zero
	STA 4300H	Store the smallest data in memory
	HLT	halt program execution

8. Search for Largest data in an array

(write an assembly language program to search the largest data in an arrayof N data stored in memory from 4200H to (4200H+N). The first element of the array gives the number of data in the array)

LABLE	MNEMONICS	COMMENT
	LXIH, 4200H	set pointer for array
	MOV B, M	set count for no. of elements in array
	INX H	
	MOV A , M	Set first element of array as smallest data
	DCR B	Decrement the count
LOOP	INX H	
	CMP M	Compare an element of array with current smallest data
	JC AHEAD	If CF =0, go to AHEAD
	MOV A, M	If CF =1, Then content of memory is larger than A-register. Hence, if CF = 1, make memory content as current largest by moving it to A - register

AHEAD	DCR B	
	JNZ LOOP	Repeat comparison until count is zero
	STA 4300H	Store the largest data in memory
	HLT	halt program execution

9. Square root of 8-bit binary number

(Write an assembly language program to find the Square root of an 8-bit binary number.The binary number is stored in memory location 4200H and store the square root in 4201H.)

LABLE	MNEMONICS	COMMENT
	LDA 4200H	Get the given data(Y) in A-register
	MOV B, A	Save the data in B-register
	MVI C, 02H	Get the deviser (02H) in C-register
	CALL DIV	Call division subroutine to get initial value (X) in the D- register.
REP	MOV E, D	Save the initial value in E- register
	MOV A, B	Get the dividend (Y) in A-register
	MOV C, D	Get the divisor (X) in C-register
	CALL DIV	Call division subroutine to get (Y/X) in D- register.
	MOV A, D	Move (Y/X) in A- register.
	ADD E	Get ((Y/X) / X) in A- register.
	MVI C, 02H	Get the deviser (02H) in C-register
	CALL DIV	Call division subroutine to get XNEW in the D-register.
	MOV A, E	Get X in A- register.
	CMP D	Compare X and XNEW
	JNZ REP	If XNEW is not equal to X, then repeat.
	STA 4201H	save the square root in memory
	HLT	halt program execution
;division subroutine		
DIV	MVI D, 00H	Clear the D- register for quotient
NEXT	SUB C	Subtract the divisor from dividend
	INR D	Increment the quotient
	CMP C	Repeat subtraction until the divisor is less than dividend.
	JNC NEXT	
	RET	Return to main program

10. Find the square of given number

(Find the square of the given numbers from memory location 6100H and store the result from memory location 7000H)

LABLE	MNEMONICS	COMMENT
	LXI H,6200H	Initialize lookup table pointer
	LXI D, 6100H	Initialize source memory pointer
	LXI B, 7000H	Initialize Destination memory pointer
BACK	LDAX D	Get the number
	MOV L,A	A point to the square
	MOV A, M	Get he square
	STAX B	Store the result at destination memory location
	INX D	Increment source memory pointer
	INX B	Increment destination memory pointer
	MOV A, C	
	CPI 05H	Check for last number
	JNZ BACK	If not repeat
	HLT	halt program execution

Write short notes on look up table and its usage. (6)

- Lookup table is an array that replaces runtime computation with a simpler array indexing operation.
- The savings in terms of processing time can be significant, since retrieving a value from memory is often faster than undergoing an 'expensive' computation or input/output operation.
- The tables may be pre-calculated and stored in static program storage, calculated (or "pre-fetched") as part of a program's initialization phase (memorization), or even stored in hardware in application-specific platforms.
- Lookup tables are also used extensively to validate input values by matching against a list of valid (or invalid) items in an array.