Noida Institute of Engineering and Technology, Greater Noida

Microprocessor

(ACSE0405)

Unit: 3

Code Conversion and BCD Arithmetic

B.Tech 4th Semester

Manish Kumar
Assistant Professor
ECE

Faculty Introduction

I received my Master of Technology from DBIT Banglore and Bachelor of Technology from Bharath Institute of Science and Technology, Bharath University, Chennai. Presently I am working as an Assistant Professor in Electronics & Communication Engineering department of Noida Institute of Engineering & Technology, Greater Noida, India. My research interest includes Antennas, VLSI and Embedded System. I published National & International Journals in repute. I attended various training programmes in the area of interest. I am a Life member of International Association of Engineers (**IAENG**). I attended various training programs in the area of Mechatronics (Pneumatics, Electro pneumatics & PLC) & Embedded System.

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA

Evaluation Scheme

Sl. No.	Subject Codes	Subject Name	Periods			Evaluation Scheme				End Semester		Total	Credit
			L	T	P	CT	TA	TOTAL	PS	TE	PE		
1	AMIAS0402	Engineering Mathematics IV	3	1	0	30	20	50		100		150	4
2	AMIASL0401	Technical Communication	2	1	0	30	20	50		100		150	3
3	AMICSE0405	Microprocessor	3	0	0	30	20	50		100		150	3
4	AMICSE0403A	Operating Systems	3	0	0	30	20	50		100		150	3
5	AMICSE0404	Theory of Automata and Formal Languages	3	0	0	30	20	50		100		150	3
6	AMICSE0401	Design and Analysis of Algorithm	3	1	0	30	20	50		100		150	4
7	AMICSE0455	Microprocessor Lab	0	0	2				25		25	50	1
8	AMICSE0453A	Operating Systems Lab	0	0	2				25		25	50	1
9	AMICSE0451	Design and Analysis of Algorithm Lab	0	0	2				25		25	50	1
10	AMICSE0459	Mini Project using Open Technology	0	0	2				50			50	1
11	ANC0402 / ANC0401	Environmental Science*/ Cyber Security*(Non Credit)	2	0	0	30	20	50		50		100	0
		GRAND TOTAL										**1100**	**24**

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA

Course Contents / Syllabus

UNIT-I: 8085 Microprocessor: Introduction to Microprocessor, Microprocessor evolution and types, Microprocessor architecture and its operation, Logic devices for interfacing, Pin diagram and internal architecture of 8085 Microprocessor, Example of an 8085 based computer, Instruction and data flow, timer and timing diagram, interrupt and machine cycle, Addressing modes.

UNIT II : 8085 Instructions and Programming Techniques: Instruction sets, Instruction Classification: data transfer operations, arithmetic operations, logical operations, branching operations, machine control and assembler directives, writing assembly language programs, Programming techniques: looping, counting and indexing.

UNIT III: Code Conversion and BCD Arithmetic: Counter and time delays, Illustrative program: Hexadecimal counter, zero-to-nine, (module ten) counter, generating pulse waveforms, Stack, Subroutine, Restart, Conditional call and return instructions, Advance subroutine concepts, Program: BCD-to-Binary conversion, Binary-to-BCD conversion, BCD-to-Seven segment code converter, Binary-to-ASCII and ASCII-to-Binary code conversion, BCD Addition, BCD Subtraction, Introduction to Advance instructions and Application, Multiplication

UNIT IV: Interfacing of I/O devices: Basic interfacing concepts, Memory interfacing, Interfacing output displays, Interfacing input devices, Memory mapped I/O, Interfacing keyboard and seven segment displays, The 8085 Interrupts, 8085 vector interrupts,8259 programmable interrupt controller.

UNIT-V: Programmable Peripheral IC's and 8086 Microprocessor: Peripheral Devices: 8255 programmable peripheral interface,8253/8254 programmable timer/counter, 8237 DMA Controller, 8251 USART and RS232C.Introduction to 8086 microprocessors. Architecture of 8086 (Pin diagram, Functional block diagram, register organization), Addressing Modes

Branch Wise Application

- **Microprocessor**, any of a type of miniature electronic device that contains the arithmetic, logic, and control circuitry necessary to perform the functions of a digital computer's central processing unit.
- This kind of integrated circuit can interpret and execute program instructions as well as handle arithmetic operations.
- Microcontroller and Microprocessors are the basic components of Embedded Systems.
- Without learning Microprocessors and Microcontrollers, you can not work on Embedded Systems. But also, you should be good in C programming.
- You should be good in understanding the documents and write code based on that.

CONTENT

- Introduction to Microprocessor & its application
- Microprocessor evolution and types of microprocessor
- Microprocessor Architecture & operation of its component

Course Objective

The student should be able to

- Understand the functional block diagram and fundamental operation of 8-bit microprocessor(INTEL 8085)
- Learn programming skill in assembly language of 8085/86
- Understand the knowledge of basic functional operation of 16-bit microprocessor(INTEL 8086)
- Acquire basic knowledge of programming and interfacing with peripheral IC's

Course Outcome

The student will be able to

- Apply a basic concept of digital fundamentals to Microprocessor based personal computer system
- Analyze a detailed s/w & h/w structure of the Microprocessor
- Write the programme of 8085 Microprocessor.
- Analyze the properties of Microprocessor (8085/8086)
- Evaluate the data transfer information through serial and parallel ports

CO-PO Mapping

Course Outcome	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
ACSE0405.1	3	1	2		1	-	-	-	-	-	-	3
ACSE0405.2	3	1	-		1	-	-	-	-	-	-	2
ACSE0405.3	3	1	3		1	-	-	-	-	-	-	2
ACSE0405.4	3	1	-		1	-	-	-	-	-	-	2
ACSE0405.5	3	1	2		1	-	-	-	-	-	-	2
Average	3	1	2.3		1	-	-	-	-	-	-	2.1

CO-PSO Mapping

Sr. No.	Course Outcome	PSO1	PSO2	PSO3
1	**ACSE0405.1**	3	2	1
2	**ACSE0405.2**	3	2	1
3	**ACSE0405.3**	3	2	1
4	**ACSE0405.4**	3	2	1
5	**ACSE0405.5**	3	2	2
	Average	**3**	**2**	**1.2**

Unit Objective

1. To study the Counter and time delays.
2. To acquire the concept of programming for counter.
3. To understand the concept of BCD Addition & Subtraction.
4. To learn conversion Binary-to-BCD conversion etc.
5. To discuss concept of Stack, Subroutine, Restart, Conditional call and return instruction.
6. To acquire the knowledge of Advance Instruction and Application, Multiplication.

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA

End Semester Question Paper Template

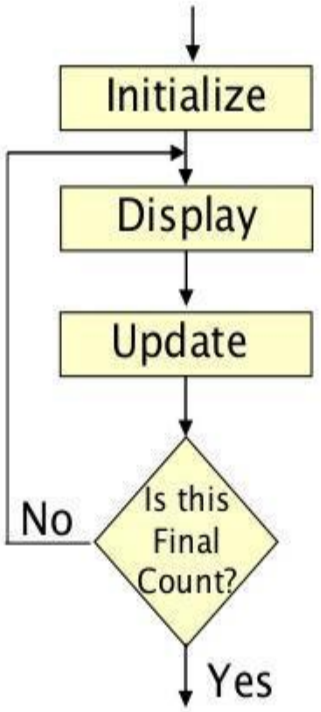
B.Tech (Semester IV Theory Examination 2021-22)		**Total Marks : 100**	
Note: Attempt all sections. If require any missing data, then choose suitably. **Time: 3 hours**			
Section A			
1. Attempt all questions in brief.		2 X 10 = 20	
Q. No.	Question	Marks	CO
a. to j		2	
Section B			
2. Attempt any three of the following		3 X 10 = 30	
Q. No.	Question	Marks	CO
a to e		10	
Section C			
Question no. 3,4,5,6,7. Attempt any one of the following		1 X 10 = 10	
a		10	
b		10	

Topic-1 Objective

Name of the Topic	Objective of the topic	Mapping with CO
Counter and Time Delays	To understand the concept of microprocessor.	CO3

Counter and Time Delays

- A loop counter is set up by loading a register with a certain value.
- Then using the DCR (to decrement) and INR (to increment) the contents of the register are updated.
- A loop is set up with a conditional jump instruction that loops back or not depending on whether the count has reached the termination count.
- The operation of a loop counter can be described using the following flowchart.

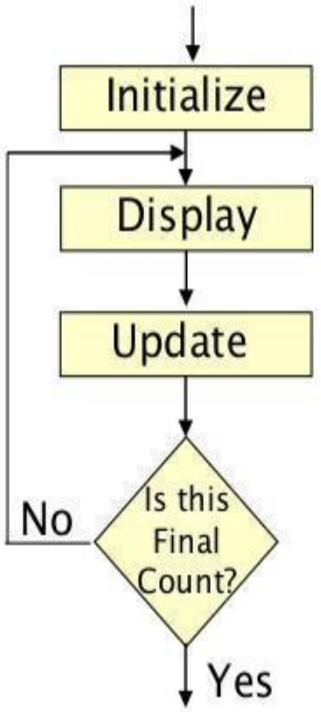


Flowchart of a Counter

Counter and Time Delays

- Using DCR instruction

```
        MVI C, 15H
LOOP:   DCR C
        JNZ LOOP
```



Flowchart of a Counter

Time Delay

Why Time delays?

- In the real time applications, such as **traffic light control, digital clock**, **process control**, **serial communication**, it is important to keep a track with time.
- For example in traffic light control application, it is necessary to give time delays between two transitions.
- These time delays are in few seconds and can be generated with the help of executing group of instructions number of times.
- These **software timers** are also called **time delays or software delays**.
- As you know microprocessor system consists of two basic components, Hardware and software.

Time Delay

- The software component controls and operates the hardware to get the desired output with the help of instructions.
- To execute these instructions, microprocessor takes fix time as per the instruction, since it is driven by constant frequency clock.
- This makes it possible to introduce delay for specific time between two events.
- The procedure used to design a specific delay is similar to that used to set up a counter.
- A register is loaded with a number, depending on the time delay required, and then the register is decremented until it reaches zero by setting up a loop with conditional jump instruction.
- The loop causes a delay, depending upon the clock period of the system.

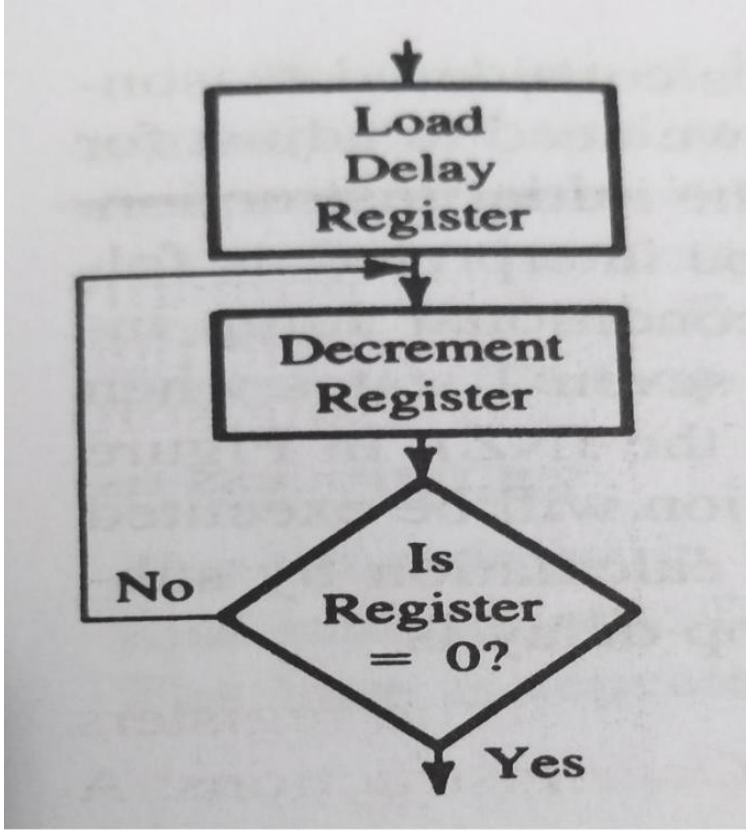
Time Delay

We can design time delay using following **three techniques**:

- Time delay using one register
- Time delay using register pair
- Time delay using a loop within a loop

Time delay using one register

- A count is loaded in a register and the loop is executed until the count reaches to zero. The flowchart has been shown below:



Time delay Formulas

- **Time required to execute an instruction**

  = number of T-states* clock period

- Clock frequency of the system f, then **Clock period T**= 1/f

Time delay Formulas

Example:

Label	opcode	operand	Description	T-state
LOOP:	MVI	C,FFH	;Load register C	7
	DCR	C	;Decrement C	4
	JNZ	LOOP	;Jump back to decrement C	10/7

Clock period= 1/f= 1/2 = 0.5 μs

Find no of T states

T-states required to execute MVI instruction= 7
T-states required to execute DCR instruction= 4
T-states required to execute JNZ instruction= 10/7

Total T-states required to execute one loop = 10+4=14T

Total T-states in Loop= Loop T states * N_{10} =(10+4)* 254=3556T
Total T-states= 3556T+18T = 3574T

Total Time=3574*0.5=1787 μs =1.8 ms

Time delay Formulas

Note:

- 255 is equivalent decimal number of hexadecimal count FF loaded in the register C.

- T-states for JNZ instruction are shown as 10/7. The 8085 microprocessor requires 10 T-states to execute a conditional jump when it jumps and seven T-states when in the last iteration, JNZ will fail (C reaches to zero).

Time Delay Using A Register Pair

- The time delay can be increased by setting a loop using register pair with a 16-bit number (maximum FFFF H).
- The 16-bit number is decremented using DCX instruction. However the, instruction DCX does not set the zero flag. So, additional techniques must be used to set the zero flag.

Example: Find total execution time taken by the below program.

Label	Opcode	Operand	Comments	T states
	LXI	B,2384H	Load BC with 16-bit count	10
LOOP:	DCX	B	Decrement BC by 1	**6**
	MOV	A,C	Place contents of C in A	**4**
	ORA	B	OR B with C to set Zero flag	**4**
	JNZ	LOOP	if result not equal to 0 , jump back to loop	**10**/7

Time Delay Using A Register Pair

Example:

- Clock frequency of the system f = 2 MHz
- Clock period= 1/f= 1/2 = 0.5 μs
- Decimal equivalent of 2384 H = 9092_{10} ⟹ 9091 times loop is executed
- **Total T-states required to execute one loop** = 6+4+4+10=24T
- Total T-states in Loop = 24T *9091=2,16,456
- Total T-states= 2,16,456T+31T = 2,16,487T
- Total Time=2,16,487*0.5=1,08,243 μs =108.2 ms

Time Delay Using A Loop Within A Loop

Example: Time Delay using a LOOP within a LOOP

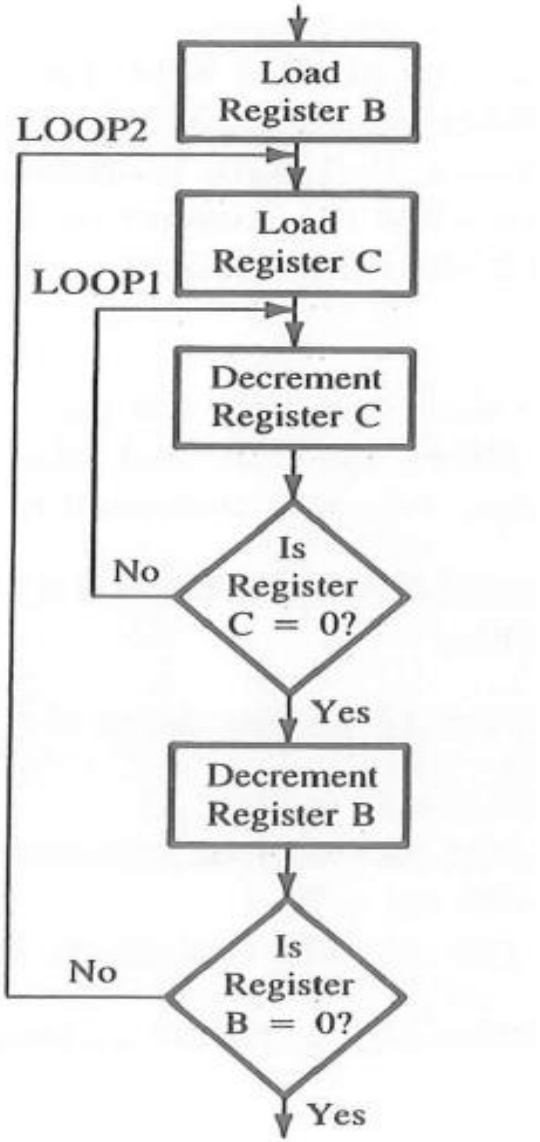
```
        MVI  B,38H      7T
LOOP2:  MVI C,FFH       7T
LOOP1:  DCR C           4T
        JNZ LOOP1       10/7 T
        DCR B           4T
        JNZ LOOP 2      10/7T
```

Timer Delay Using NOP Instruction:

- NOP instruction does nothing but takes 4T states of processor time to execute. So by executing NOP instruction in between two instructions we can get delay of 4 T-state.

Time Delay Using A Loop Within A Loop

FIGURE 8.3
Flowchart for Time Delay with Two Loops



Counter design with time delay

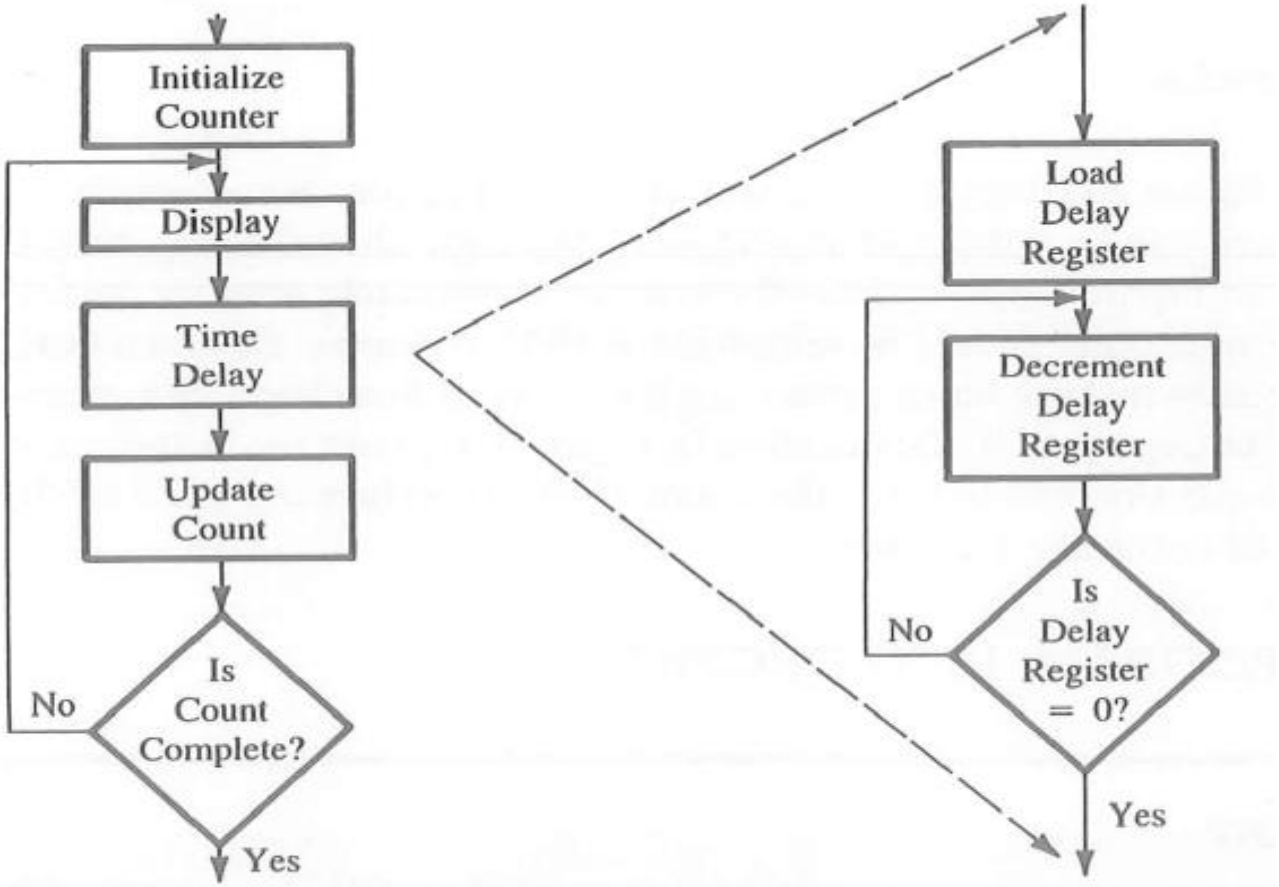


FIGURE 8.4
Flowchart of a Counter with a Time Delay

Topic-2 Objective

Name of the Topic	Objective of the topic	Mapping with CO
Illustrative program: Hexadecimal counter	To understand the concept of Illustrative program: Hexadecimal counter	CO2

Hexadecimal Counter

Program Statement:

Write a program to count continuously in hexadecimal from FFH to 00H in a system with clock period 0.5 microseconds. Use register C to set up a delay of 1ms between each count and display output at one of the output ports.

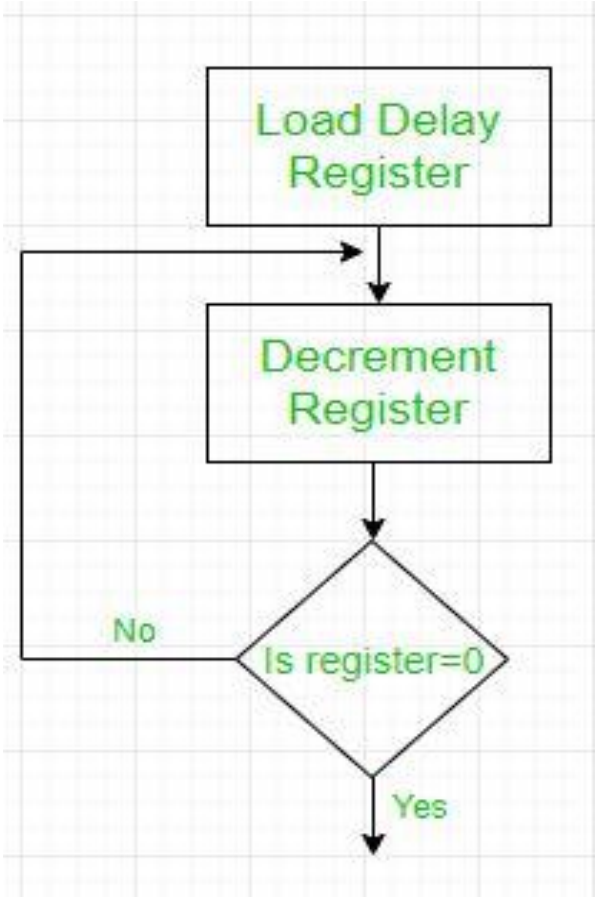
Hexadecimal Counter

Problem Analysis:

1. The hexadecimal counter is set by loading a register with starting number and decrementing it till zero is reached and then again decrementing it to will produce -1, which is two's complement of FFH. Hence, the register again reaches FFH.

2. The 1ms time delay is set up by the procedure shown in flowchart- The register is loaded with appropriate number such that the execution of above loop produces a time delay of 1ms.

Hexadecimal Counter



Program Statement:

Address	Label	Mnemonics
2000H		MVI B, 00H
2002H	NEXT	DCR B
2003H		MVI C, COUNT
2005H	DELAY	DCR C
2006H		JNZ DELAY
2009H		MOV A, B
200AH		OUTPORT#
200CH		JMP NEXT

Hexadecimal Counter

- The C register is the time delay register which is loaded by a value COUNT to produce a time delay of 1ms.
- To find the value of COUNT we do- $T_D = T_L + T_O$

  Where- TD = Time Delay, TL = Time delay inside loop,

  TO = Time delay outside loop
- The delay loop includes two instructions-

  DCR C (4 T-states) and JNZ (10 T-states)

  So, TL = 14*Clock period*COUNT

  => 14*(0.5*10^-6)*COUNT

  => (7*10^-6)*COUNT

Hexadecimal Counter

Delay outside the loop includes-

DCR B : 4T

MVI C, COUNT : 7T

MOV A, B : 4T

OUTPORT : 10T

JMP : 10T

Total : 35T

TO = 35*Clock period => 17.5 microseconds

So, 1ms= (17.5+ 7*COUNT) microsecond

Therefore, COUNT=$(140)_{10}$

Illustrative program: Zero-to-nine, (module ten)

Program Statement:

Write a program to count from 0 to 9 with a one second delay between each count. At the count of 9, the counter should reset itself to 0 and repeat the sequence continuously. Use register pair HL to set up the delay and display each count at one of the output ports. Assume the clock frequency of the microcomputer is 1 MHz.

Illustrative program: Zero-to-nine, (module ten)

Instructions: Review the following Instructions:

- LXI: Load Register Pair Immediate
- DCX: Decrement Register Pair
- INX: Increment Register Pair

These instructions manipulate 16-bit data by using registers in pairs (BC, DE and HL). However, the instructions DCX and INX do not set flags to reflect the outcome of the operation. Therefore, additional instructions must be used to set the flags.

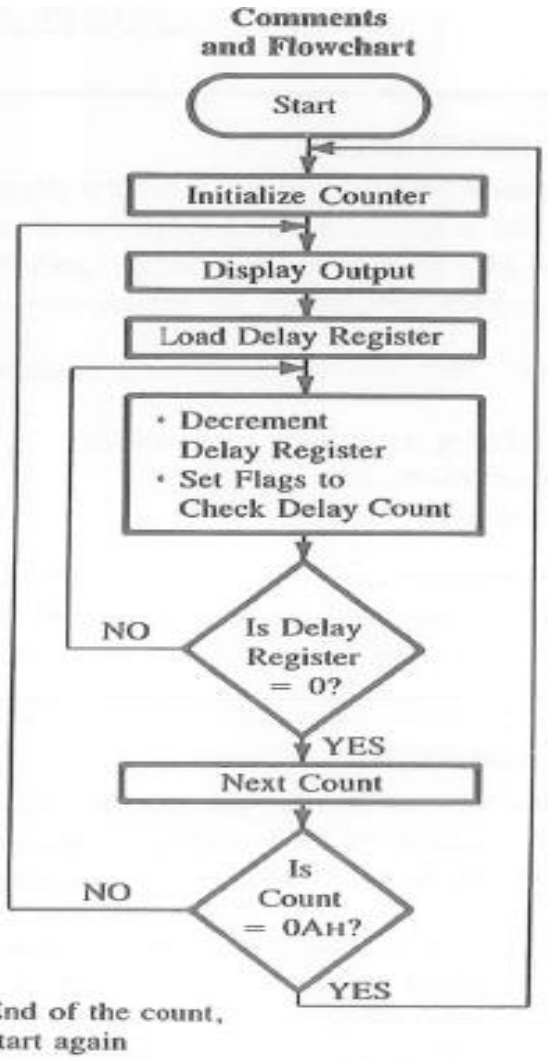
Illustrative program: Zero-to-nine, (module ten)

Problem Analysis: The problem is similar to that in the Illustrative Program for a hexadecimal counter except in two respects: the counter is an up-counter (counts up to the digit 9), and the delay is too long to use just one register.

1. The counter is set up by loading a register with the appropriate number and incrementing it until reaches the digit 9. When the counter register reaches the final count, it is reset to zero.
2. The 1-second delay between each count is set up by using a register pair.

Illustrative program: Zero-to-nine, (module ten)

Label	Mnemonics	T	
START:	MVI B,00H		
DSPLAY:	OUT PORT#	10	T_0
	LXI H,16-Bit	10	
LOOP:	DCX H	6	T_L: 24 T-states
	MOV A,L	4	
	ORA H	4	
	JNZ LOOP	10/7	
	INR B	4	T_0
	MOV A,B	4	
	CPI 0AH	7	
	JNZ DSPLAY	10/7	
	JZ START		



Illustrative program: Zero-to-nine, (module ten)

Delay Calculations:

- The major delay between two counts is provided by the 16-bit number in the delay register HL. (the inner loop in the flowchart)
- This delay is set up by using a register pair.
- **Loop Delay TL** = 24 T-states x Tx Count

  1 second = 24 x 10 microsecond x Count

  Count = $1/(24 \times 10^{-6}) = (41666)_{10} = A2C2H$
- The delay count A2C2H in register HL would provide approximately a 1-second delay between two counts.

Illustrative program: Generating Pulse Waveform

PROBLEM STATEMENT

Write a program to generate a continuous square wave with the period of 500 µs. Assume the system clock period is 325 ns, and use bit D_0 to output the square wave.

Label	Mnemonics	Comments
	MVI D,AA	;Load bit pattern AAH
ROTATE:	MOV A,D	;Load bit pattern in A
	RLC	;Change data from AAH to
		; 55H and vice versa
	MOV D,A	;Save (A)
	ANI 01H	;Mask bits D_7–D_1
	OUT PORT1	;Turn on or off the lights
	MVI B,COUNT (7T)	;Load delay count for 250 µs
DELAY:	DCR B (4T)	;Next count
	JNZ DELAY (10/7T)	;Repeat until (B) = 0
	JMP ROTATE (10T)	;Go back to change logic level

Illustrative program: Generating Pulse Waveform

Delay Calculations In this problem, the pulse width is relatively small (250 µs); therefore, to obtain a reasonably accurate output pulse width, we should account for all the T-states. The total delay should include the delay in the loop and the execution time of the instructions outside the loop.

1. The number of instructions outside the loop is seven; it includes six instructions before the loop beginning at the symbol ROTATE and the last instruction JMP.

$$\text{Delay outside the Loop: } T_O = 46 \text{ T-states} \times 325 \text{ ns} = 14.95 \text{ µs}$$

2. The delay loop includes two instructions (DCR and JNZ) with 14 T-states except for the last cycle, which has 11 T-states.

$$\text{Loop Delay: } T_L = 14 \text{ T-states} \times 325 \text{ ns} \times (\text{Count} - 1) + 11 \text{ T-states} \times 325 \text{ ns}$$
$$= 4.5 \text{ µs } (\text{Count} - 1) + 3.575 \text{ µs}$$

3. The total delay required is 250 µs. Therefore, the count can be calculated as follows:

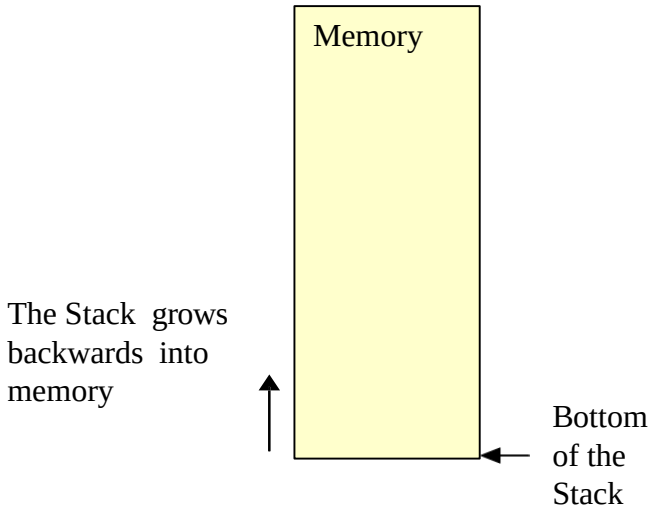
$$T_D = T_O + T_L$$
$$250 \text{ µs} = 14.95 \text{ µs} + 4.5 \text{ µs } (\text{Count} - 1) + 3.575 \text{ µs}$$
$$\text{Count} = 52.4_{10} = 34\text{H}$$

Topic-3 Objective

Name of the Topic	Objective of the topic	Mapping with CO
Stack	To understand the concept of Stack	CO3

Stack

- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO structure- Last In First Out.
- The stack normally grows backwards into memory.
- In other words, the programmer defines the bottom of the stack, and the stack grows up into reducing address range.



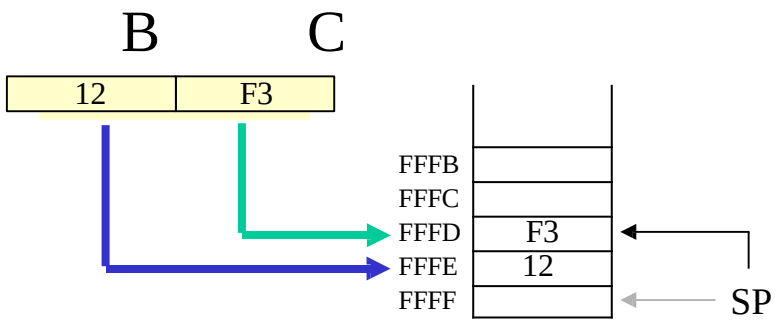
Stack

- In the 8085, the stack is defined by setting the SP (Stack Pointer) register.
    - LXI SP, FFFFH
- This sets the Stack Pointer to location FFFFH (end of memory for the 8085).
- The Size of the stack is limited only by the available memory
  - Information is saved on the stack by PUSHing it on.
  - It is retrieved from the stack by POPing it off.
- The 8085 provides two instructions: PUSH and POP for storing information on the stack and retrieving it back.
- Both PUSH and POP work with register pairs ONLY.

Stack

PUSH Instruction

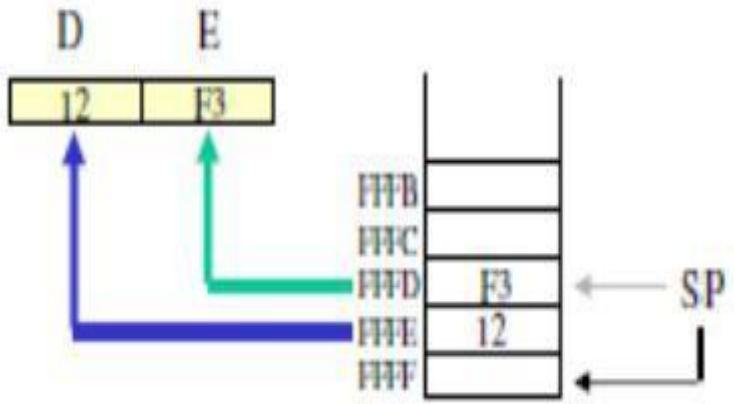
- PUSH B (1 Byte Instruction)
- Decrement SP
- Copy the contents of register B to the memory location pointed by SP
- Decrement SP
- Copy the contents of register C to the memory location pointed by SP



Stack

POP Instruction

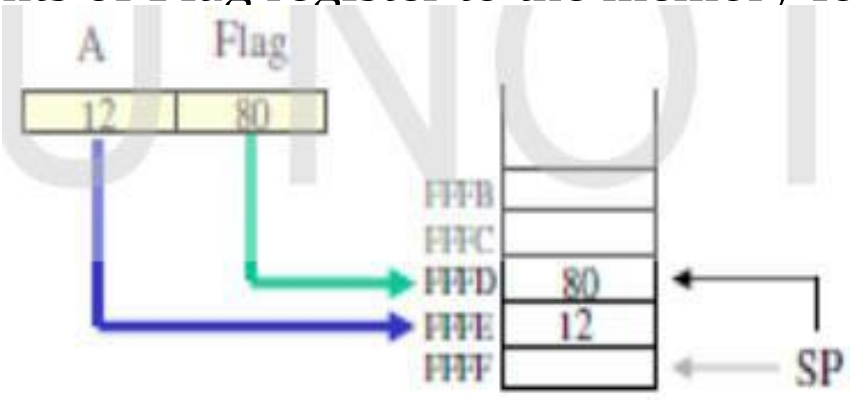
- POP D (1 Byte Instruction)
- Copy the contents of the memory location pointed by the SP to register E
- Increment SP
- Copy the contents of the memory location pointed by the SP to register D
- Increment SP



Stack

PUSH PSW Register Pair

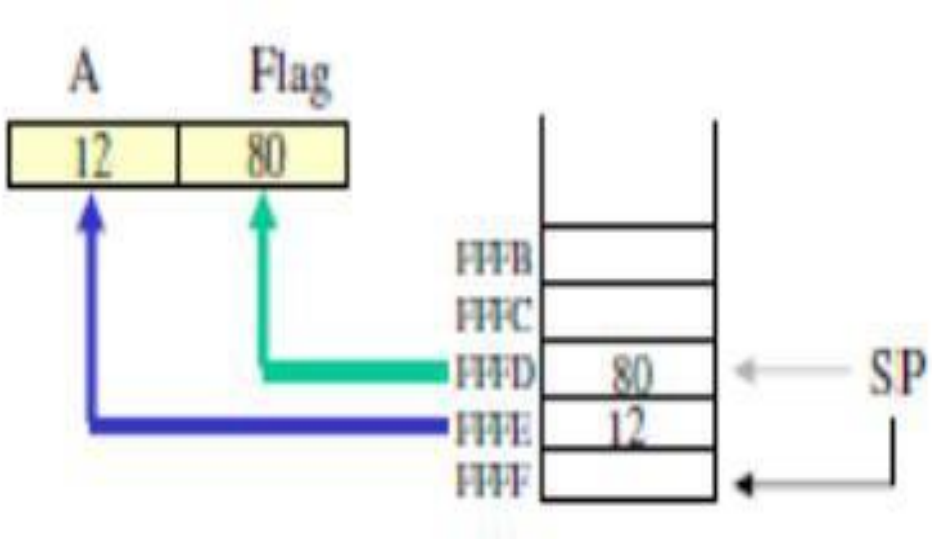
- This register pair is made up of the Accumulator and the Flags registers.
- PUSH PSW (1 Byte Instruction)
- Decrement SP
- Copy the contents of register A to the memory location pointed by SP
- Decrement SP
- Copy the contents of Flag register to the memory location pointed by SP



Stack

POP PSW (1 Byte Instruction)

- Copy the contents of the memory location pointed by the SP to Flag register
- Increment SP
- Copy the contents of the memory location pointed by the SP to register A
- Increment SP



Stack

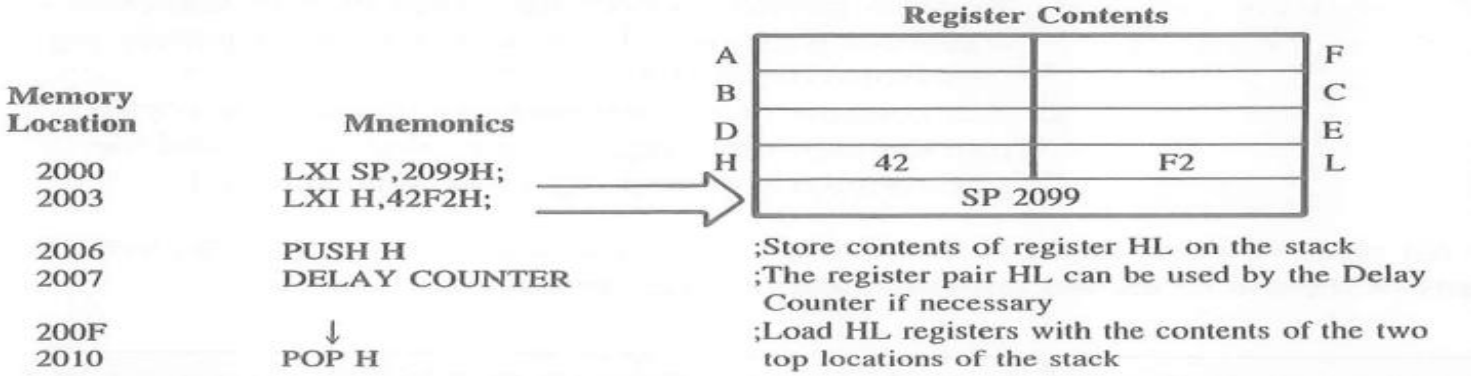


FIGURE 9.1
Instructions and Register Contents in Example 9.1

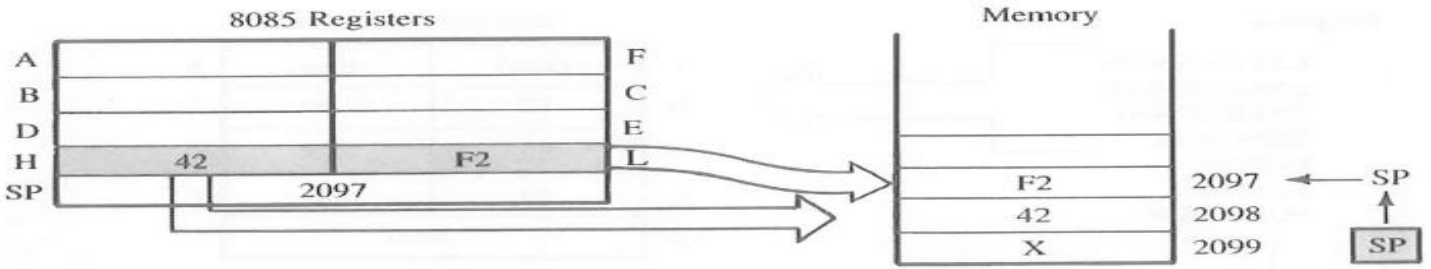


FIGURE 9.2
Contents on the Stack and in the Registers After the PUSH Instruction

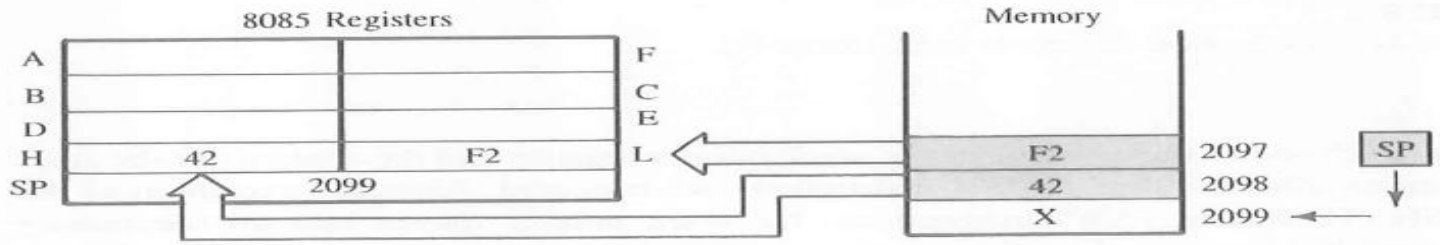


FIGURE 9.3
Contents on the Stack and in the Registers After the POP Instruction

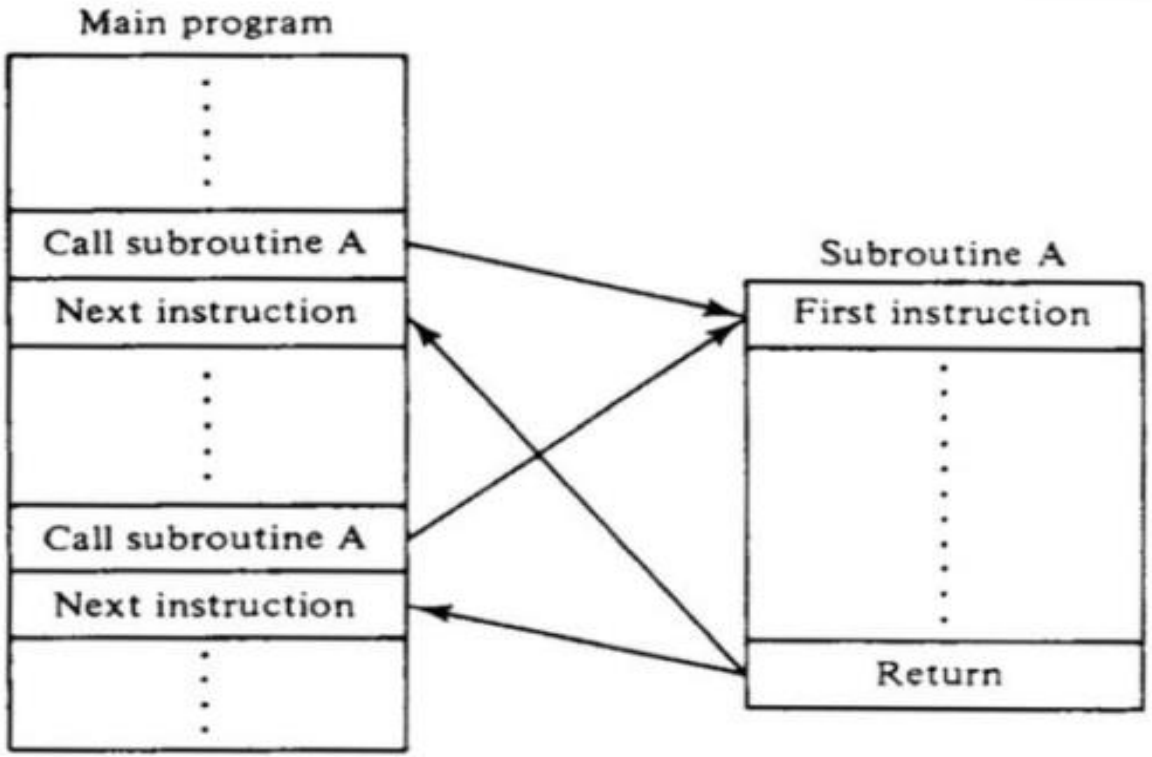
Topic-4 Objective

Name of the Topic	Objective of the topic	Mapping with CO
Subroutine	To understand the concept of Subroutine	CO3

Subroutine

- A subroutine is a group of instructions (subprogram) that will be used repeatedly in different locations of the program.
- Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.
- In Assembly language, a subroutine can exist anywhere in the code.
- However, it is customary to place subroutines separately from the main program.
- The 8085 has **two instructions** for dealing with subroutines.
  - The CALL instruction is used to redirect program execution to the subroutine.
  - The RET instruction is used to return the execution to the calling routine.

Subroutine

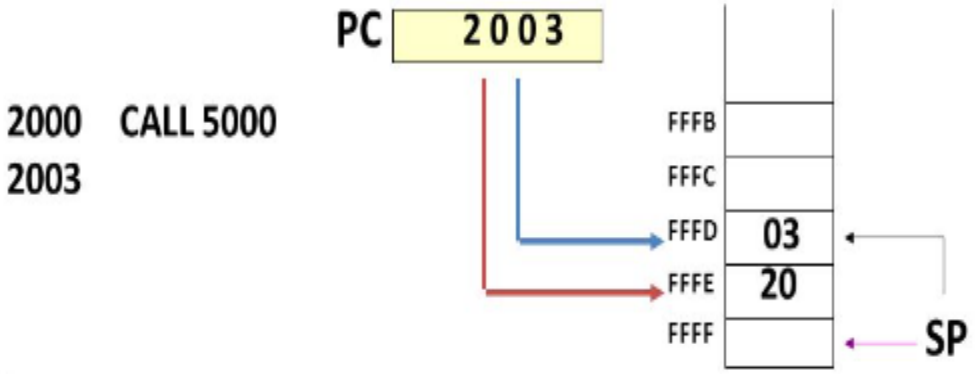

Main program
Call subroutine A
Next instruction
Call subroutine A
Next instruction
Subroutine A
First instruction
Return

Subroutine

The CALL Instruction:

- CALL 5000H (3-byte instruction)

When CALL instruction is fetched, the Microprocessor knows that the next two Memory locations contain 16-bit subroutine address in the memory.

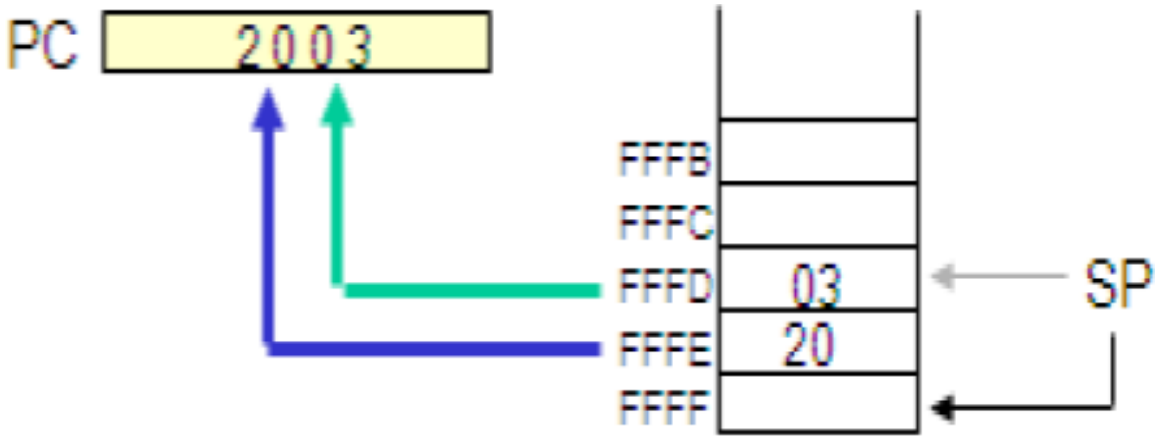


Subroutine

RET (1 byte instruction)

–Retrieve the return address from the top of the stack

–Load the program counter with the return address.



Subroutine

Conditional CALL and RET Instructions:

- The conditional call and the return instructions are based on four data conditions (flags)-carry, zero, sign and parity.
- The conditions are tested by checking the respective flags.
- In case of a conditional call instruction, the program is transferred to the subroutine if condition is met; otherwise, the main program is continued.
- In case of a conditional return instruction, the sequence returns to the main program if the condition is met; otherwise, the sequence in the subroutine is continued.

Subroutine

The 8085 supports conditional CALL and conditional RET instructions.

Conditional CALL:

- CC, calls subroutine if Carry flag is set.
- CNC, call subroutine if Carry flag is not set
- CZ, calls subroutine if zero flag is set.
- CNZ, call subroutine if zero flag is not set
- CM, calls subroutine if sign flag is set. (sign=1 negative number)
- CP, call subroutine if sign flag is not set (sign =0 positive number)
- CPE, call subroutine if Parity flag is set (P=1, even parity)
- CPO, call subroutine if Parity flag is reset (P=0, odd parity)

Subroutine

Conditional Return:

- RC, returns from subroutine if Carry flag is set
- RNC, returns from subroutine if Carry flag is not set etc
- RZ, returns subroutine if zero flag is set.
- RNZ, returns subroutine if zero flag is not set
- RM, returns subroutine if sign flag is set. (Sign=1 negative number)
- RP, returns subroutine if sign flag is not set. (sign =0 positive number)
- –RPE, returns subroutine if parity flag is set. (P=1 even parity)
- –RPO, returns subroutine if parity flag is not set. (P=0 odd parity)

Advanced Subroutine Concepts

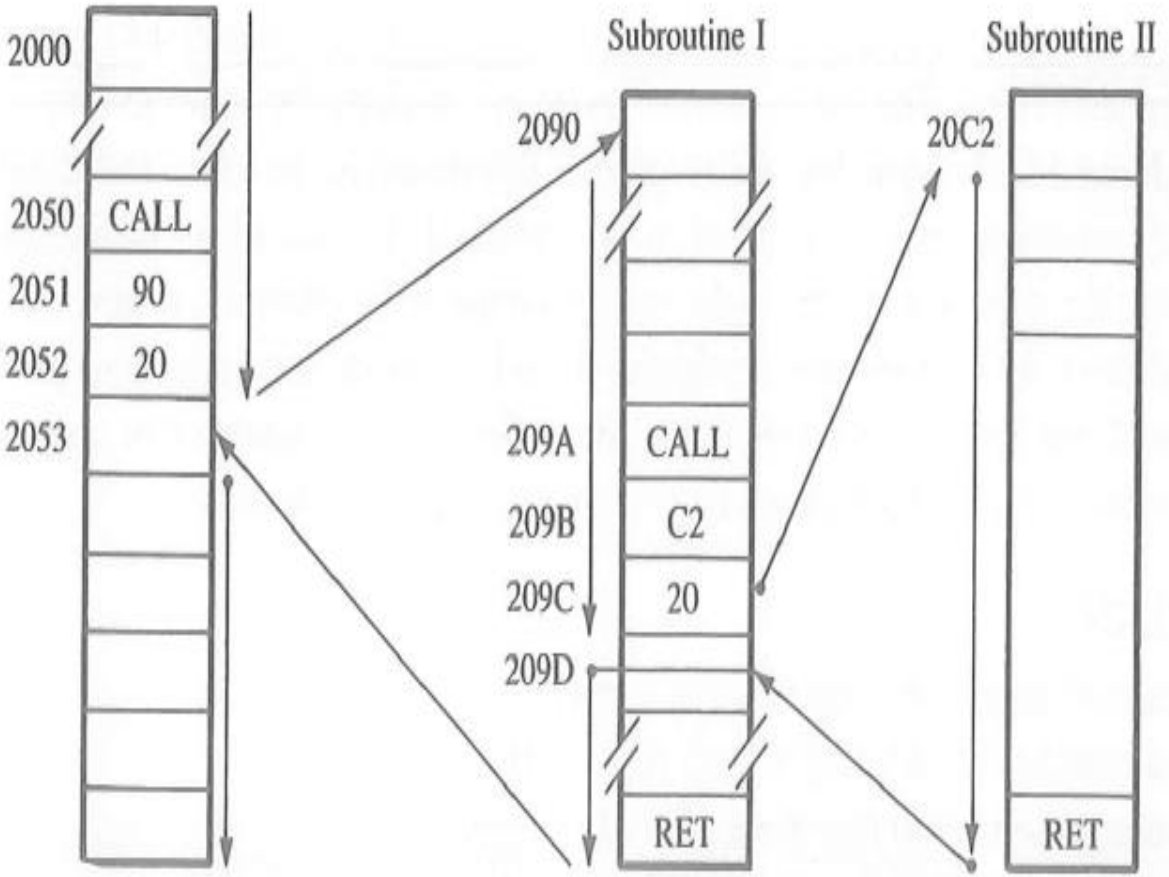
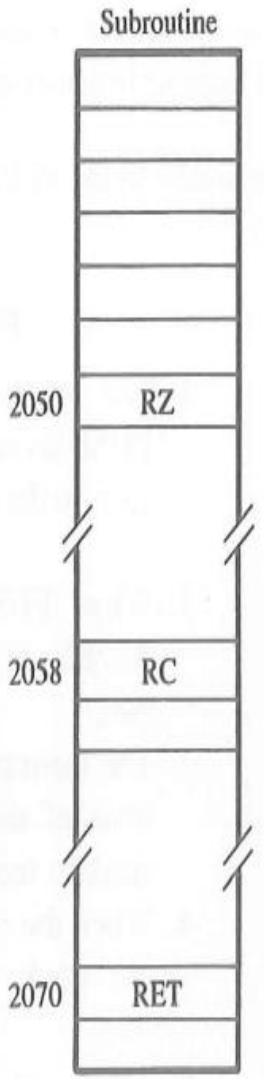


FIGURE 9.15
Nesting of Subroutines

FIGURE 9.16
Multiple-Ending Subroutine



Advanced Subroutine Concepts

The instructions CALL and RET are similar to the instructions PUSH and POP. The similarities and differences are as follows:

CALL and RET	PUSH and POP
1. When CALL is executed, the microprocessor automatically stores the 16-bit address of the instruction next to CALL on the stack.	1. The programmer uses the instruction PUSH to save the contents of a register pair on the stack.
2. When CALL is executed, the stack pointer register is decremented by two.	2. When PUSH is executed, the stack pointer register is decremented by two.
3. The instruction RET transfers the contents of the top two locations of the stack to the program counter.	3. The instruction POP transfers the contents of the top two locations of the stack to the specified register pair.
4. When the instruction RET is executed, the stack pointer is incremented by two.	4. When the instruction POP is executed, the stack pointer is incremented by two.
5. In addition to the unconditional CALL and RET instructions, there are eight conditional CALL and RETURN instructions.	5. There are no conditional PUSH and POP instructions.

Topic-5 Objective

Name of the Topic	Objective of the topic	Mapping with CO
Code Conversion	To understand the concept of Code Conversion	CO3

Code Conversion

- The ASCII (American Standard Code For Information Interchange) keyboard is a commonly used input device.

- Alphanumeric characters (letters and numbers) are displayed on a CRT (cathode ray tube) terminal using the ASCII code. However, inside the microprocessor data processing is usually performed in binary.

- In some instances, arithmetic operations are performed in BCD numbers. Therefore, data must be converted from one code to another code.

Code Conversion

- The programming techniques used for code conversion fall into four general categories:
  1. Conversion based on the position of a digit in a number (BCD to binary and vice versa).
  2. Conversion based on hardware consideration (binary to seven-segment code using table look- up procedure).
  3. Conversion based on sequential order of digits (binary to ASCII and vice versa).
  4. Decimal adjustment in BCD arithmetic operations. (This is an adjustment rather than a code conversion.)

BCD To Binary Conversion

- In most microprocessor-based products, data are entered and displayed in decimal numbers.
- The system-monitor program of the instrument converts each key into an equivalent 4-bit binary number and stores two BCD numbers in an 8-bit register memory location. These numbers are called **packed BCD**.
- Even if data are entered in decimal digits, it is inefficient to process data in BCD numbers because, in each 4-bit combination, digits A through F are unused. Therefore, BCD numbers are converted into binary numbers for data processing.

BCD To Binary Conversion

- The conversion of BCD number into its binary equivalent employs the principle **positional weighting** in a given number.

For example:

$$72_{10} = 7 \times 10 + 2$$

- The digit 7 represents 70, based on its second position the right. Therefore, converting 72_{BCD} into its binary equivalent requires multiplying second digit 10, and adding digit.

BCD To Binary Conversion

Converting 2-digit BCD number into binary equivalent requires the following steps:

1. Separate an 8-bit packed BCD number into 4-bit unpacked digits: BCD_1, and BCD_2.
2. Convert each digit into its binary value according to its position.
3. Add both binary numbers to obtain the binary equivalent of the BCD numbers.

BCD To Binary Conversion

Convert 72_{BCD} into its binary equivalent.

$$72_{10} = 0111\ 0010_{BCD}$$

Step 1: 0111 0010 → 0000 0010 Unpacked BCD_1
→ 0000 0111 Unpacked BCD_2

Step 2: Multiply BCD_2 by 10 (7×10)

Step 3: Add BCD_1 to the answer in Step 2

BCD To Binary Conversion

Problem Statement :-

Convert a 2-digit BCD number stored at memory location 2200H into its binary equivalent number and store the result in memory location 2300H

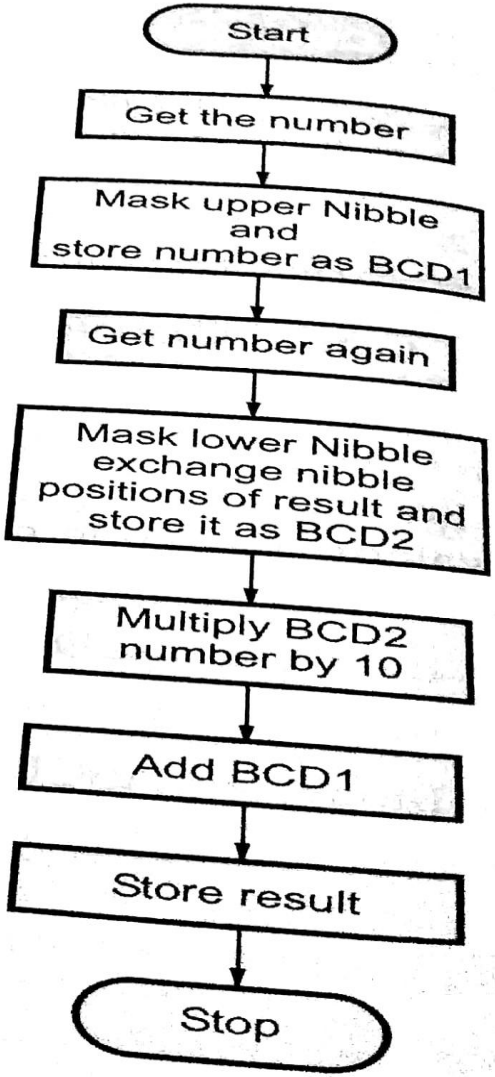
Assume, 2200H → 67 BCD

↓

6 X 0A + 7 = 43H

(Add 0A, 6 times) (0A+0A+0A+0A+0A+0A)+7

Flowchart :



BCD To Binary Conversion

```
        LDA   2200H   ; Load the BCD no. into Accumulator
        MOV   B, A    ; Save BCD number.
        ANI   0FH     ; Mask most significant four bits to
                        unpacked BCD1
        MOV   C, A    ; Save unpacked BCD1 in C
        MOV   A, B    ; Get BCD again
        ANI   F0H     ; Mask least significant four bits to
                        unpacked BCD2
        RRC           ; Convert most significant four bits
        RRC           ; into unpacked BCD2
        RRC
        RRC
        MOV   D, A    ; Save BCD2 in D
        XRA   A       ; Clear accumulator
        MVI   E, 0A   ; set E as multiplier of 10
SUM:    ADD   E       ; Add to until (D) = 0
        DCR   D       ; Reduce BCD2 by one
        JNZ   SUM     ; Is multiplication complete ?
                        If not, go back and add again
        ADD   C       ; Add BCD1
        STA   2300H   ; Stores the result.
        HLT           ; End the program.
```

Binary to BCD Conversion

- Microprocessor-based products, numbers are displayed in decimal. However, if data processing inside the microprocessor is performed in binary, it is necessary to convert the binary results into their equivalent BCD numbers just before they are displayed.

- The conversion of binary to BCD is performed by dividing the number by the powers of ten and the division is performed by the subtraction method.

For example

$$1111\ 1111_2 \text{ (FFH)} = 255_{10}$$

- To represent this number in BCD requires twelve bits or three BCD digits, labelled here as BCD_3 (MSB), BCD_2, and BCD_1 (LSB),

$$= 0010 \quad 0101 \quad 0101$$

$$BCD_3 \quad BCD_2 \quad BCD_1$$

Binary to BCD Conversion

The conversion can be performed as follows:

- **Step 1:** If the number is less than 100, go to Step 2; otherwise, divide by 100 or subtract 100 repeatedly until the remainder is less than 100. The quotient is the most significant BCD digit BCD_3.
- **Step 2:** If the number is less than 10, go to Step 3, otherwise divide by 10 repeatedly until the remainder is less than 10. The quotient is BCD_2.
- **Step 3:** The remainder from step 2 is BCD_1.

Binary to BCD Conversion

The conversion can be performed as follows :-

Step 1 :- If the number is equal to or greater than 100 (64H) divide number by 100 [i.e. subtract 100 (64H)] repeatedly until the remainder is less than 100; otherwise, go to step 2.
The quotient is the most significant BCD digit, BCD_3

Example :-

		Quotient	
255			
−100	= 155	1	step 1
−100	= 55	1	step 1
	BCD_3 =	2	step 1
55			
−10	= 45	1	step 2
−10	= 35	1	step 2
−10	= 25	1	step 2
−10	= 15	1	step 2
−10	= 5	1	step 2
	BCD_2 =	5	step 2
	BCD_1 =	5	step 3

Step 2 :- If the number is less than 10, go to step 3; otherwise, divide by 10 repeatedly until the remainder is less than 10. The quotient is BCD_2.

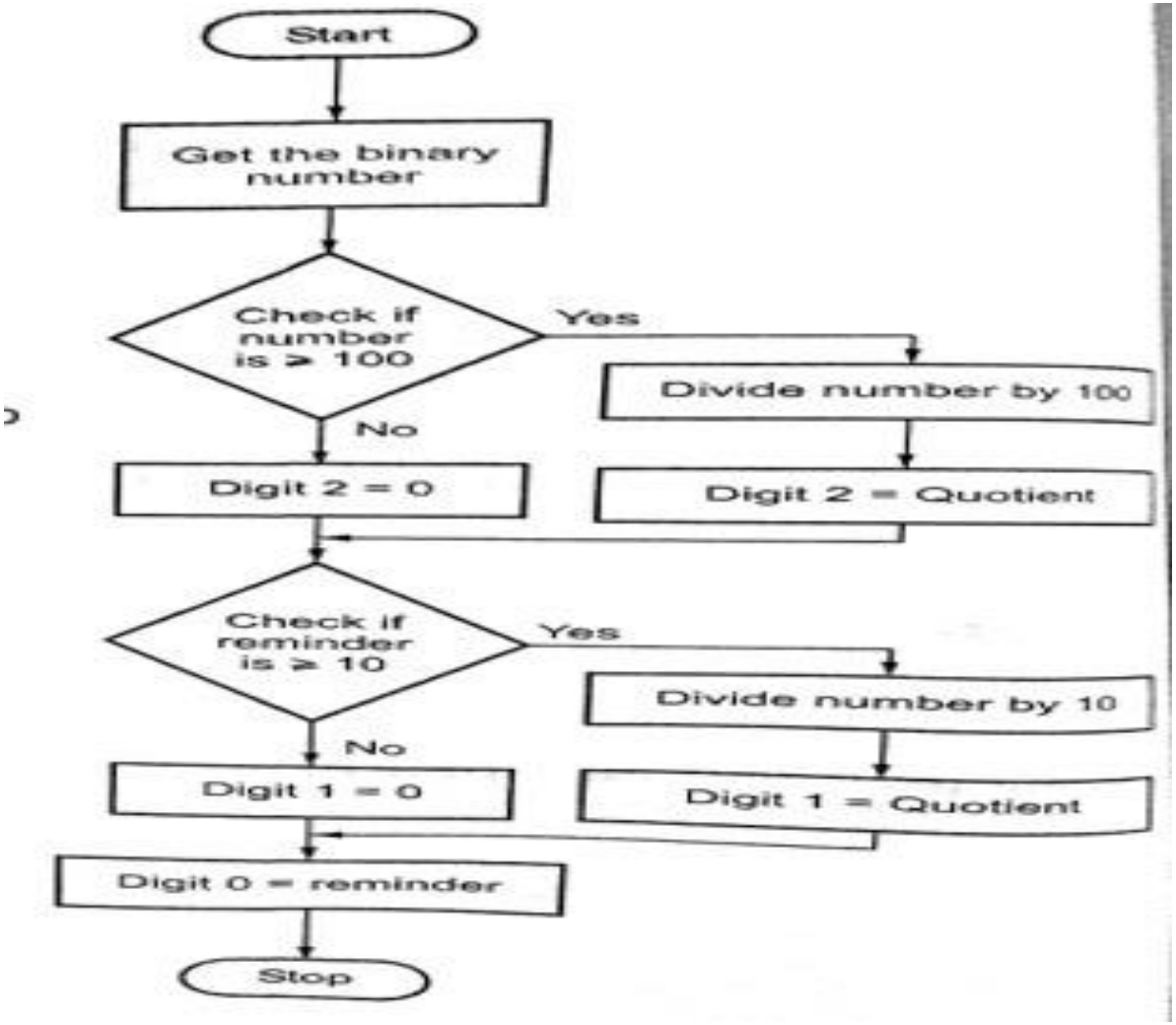
Step 3 :- The remainder from step 2 is BCD_1.

Binary to BCD Conversion

Problem Statement :

Write a main program and conversion subroutine to convert 8 bit number into a BCD number. Thus it requires 12 bits to represent 3 BCD digits. The result is stored as 3 unpacked BCD digits in memory locations start from 6100H.

Binary to BCD Conversion



Binary to BCD Conversion

Main Program-

Address	Instructions	Comment
	LXI SP, 2500H	Initialize Stack pointer
	LDA 6000H	Load Binary number into accumulator
	CALL BINBCD	Call subroutine BINBCD
	HLT	

Binary to BCD Conversion

Subroutine -

Address	Instructions	Comment
BINBCD:	MVI B,64H	Load divisor decimal 100 in B register
	MVI C,0AH	Load divisor decimal 10 in C register
	MVI D,00H	Initialize digit 1
	MVI E,00H	Initialize digit 2
STEP1:	CMP B	Compare number with decimal 100
	JC STEP2	Jump to step 2 if CY =1
	SUB B	Subtract
	INR E	Update Quotient
	JMP STEP1	Jump to step 1
STEP2:	CMP C	Compare number with decimal 10
	JC STEP3	Jump to step 3 if CY =1

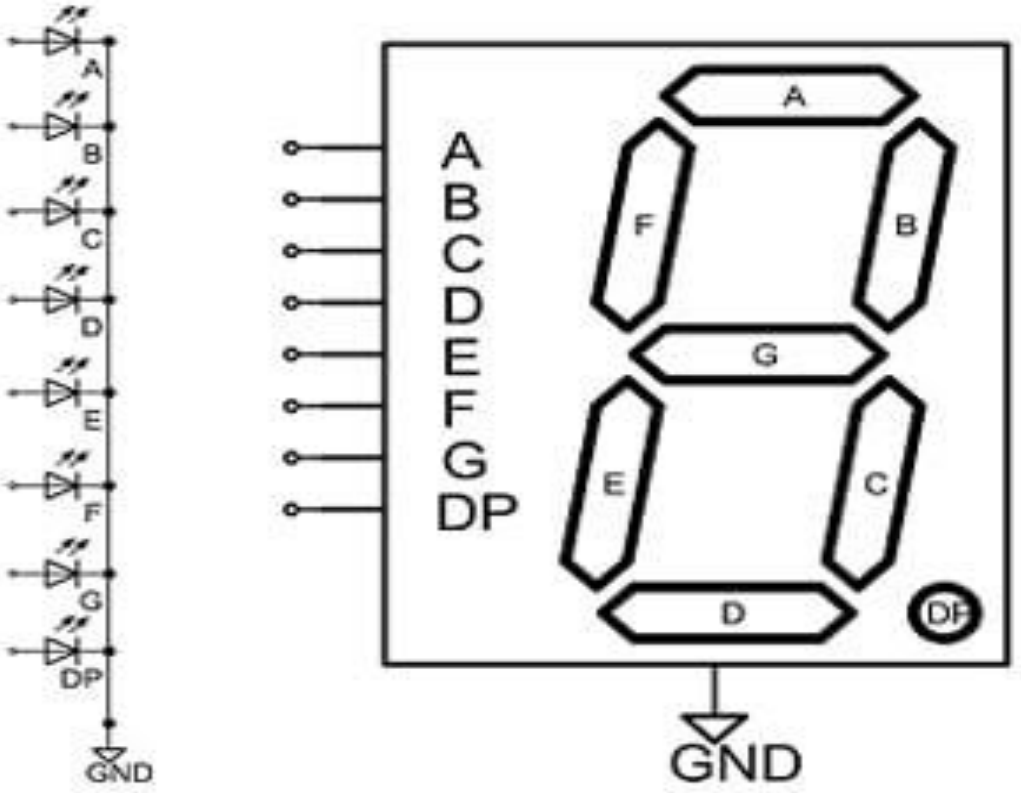
Binary to BCD Conversion

Address	Instructions	Comment
	SUB C	Subtract
	INR D	Update Quotient
	JMP STEP2	Jump to step 2
STEP3:	STA 6100H	Store digit 0
	MOV A,D	Get digit 1
	STA 6101H	Store digit 1
	MOV A,E	Get digit 2
	STA 6102H	Store digit 2
	RET	Return to main program

BCD to Seven Segment Conversion

- Many times 7-segment LED display is used to display the results or parameters in the microprocessor system.
- In such cases we have to convert the result or parameter in 7-segment code.
- This conversion can be done using look-up technique.
- In the look-up table the codes of the digits (0-9) to be displayed are stored sequentially in the memory.
- The conversion program locates the code of a digit based on its BCD digit.
- Let us see the Program for BCD to common cathode 7-segment code conversion.

BCD to Seven Segment Conversion



Seven segment common cathode display

BCD to Seven Segment Conversion

Number	g f e d c b a	Hexadecimal
0	0 1 1 1 1 1 1	3F
1	0 0 0 0 1 1 0	06
2	1 0 1 1 0 1 1	5B
3	1 0 0 1 1 1 1	4F
4	1 1 0 0 1 1 0	66
5	1 1 0 1 1 0 1	6D
6	1 1 1 1 1 0 1	7D
7	0 0 0 0 1 1 1	07
8	1 1 1 1 1 1 1	7F
9	1 1 0 1 1 1 1	6F

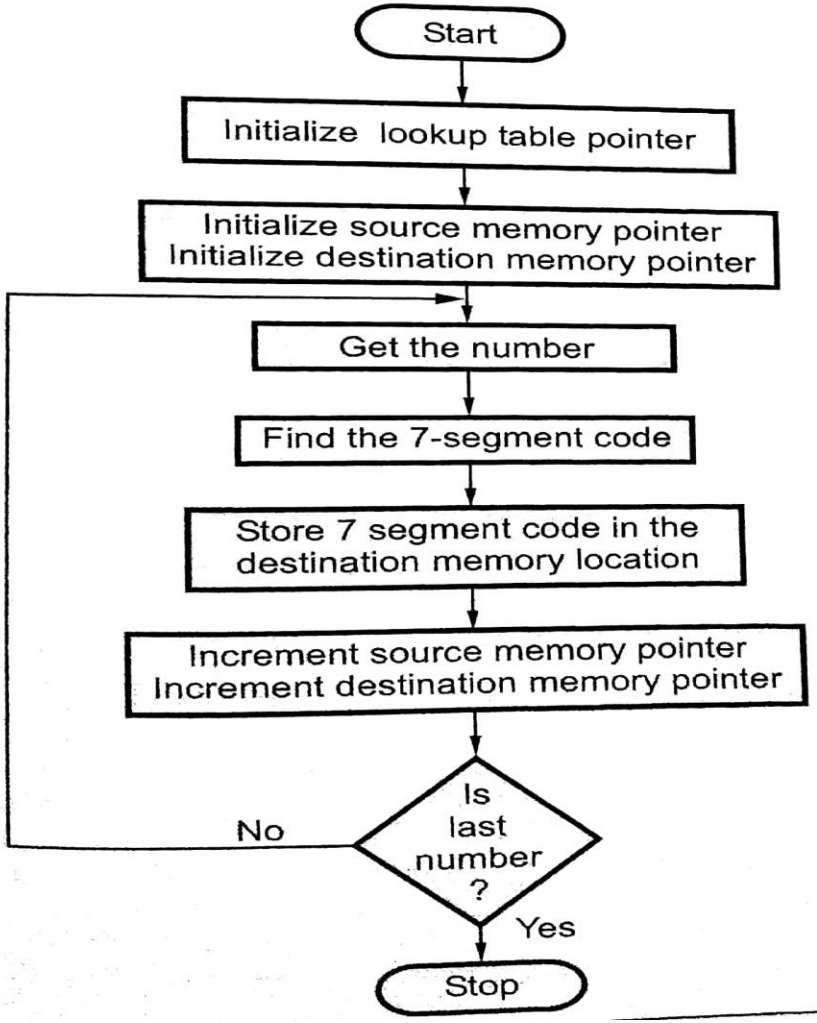
Look Up Table common cathode LED display

BCD to Seven Segment Conversion

Problem Statement –

Using look up table for LED code conversion stored at memory location 3000H find the LED code of BCD number stored at 2000H and store MSB & LSB codes at 4000H & 4001H respectively.

BCD to Seven Segment Conversion



Lookup table

Digit	Code
0	3F
1	06
2	5B
3	4F
4	66
5	6D
6	7D
7	07
8	7F
9	6F

BCD to Seven Segment Conversion

Address	Instructions	Comment
	LXI H,3000H	Initialize lookup table pointer
	LXI D,2000H	Initialize source memory pointer
	LXI B,4000H	Initialize destination memory pointer
	LDAX D	Get the number
	RRC	
	RRC	
	RRC	
	RRC	
	CALL LEDCODE	Call subroutine to identify 7 segment LED code (upper nibble)
	INX B	Increment destination pointer
	LDAX D	

BCD to Seven Segment Conversion

Address	Instructions	Comment
	CALL LEDCODE	Call subroutine to identify 7 segment LED code (lower nibble)
	HLT	Terminate the program

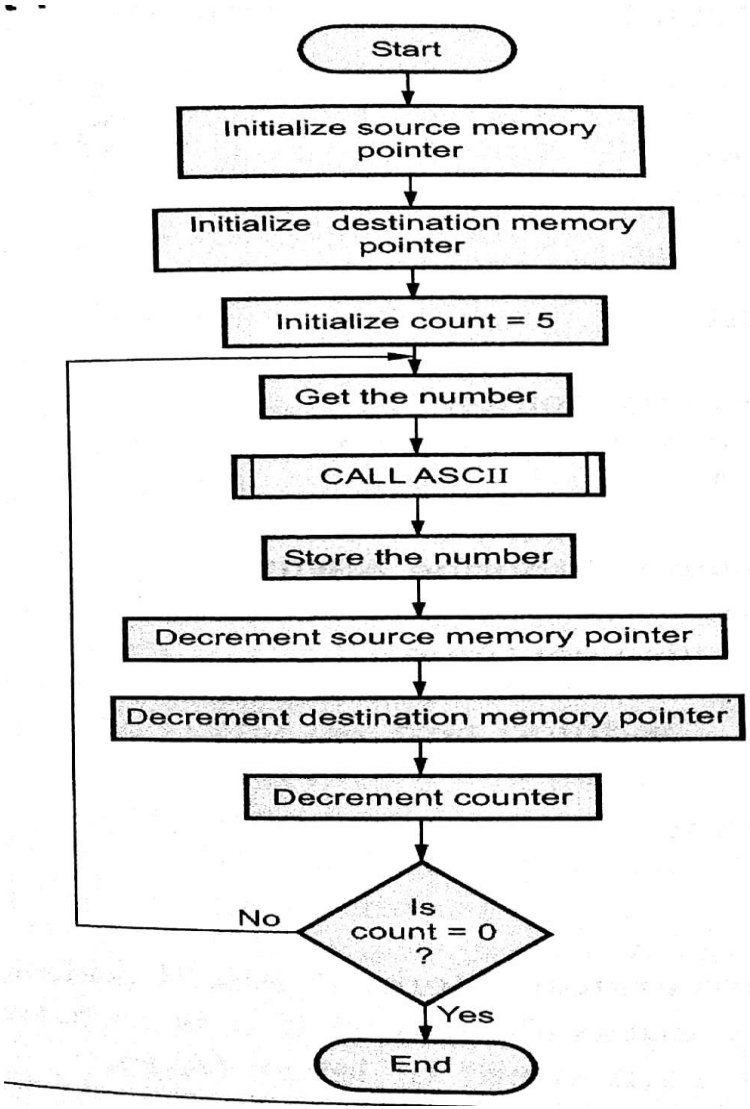
SUBROUTINE

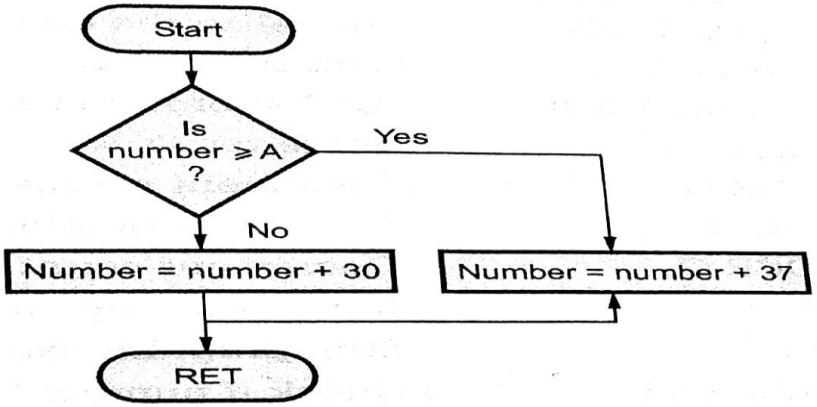
Address	Instructions	Comment
LEDCODE	ANI 0FH	Masking upper nibble
	MOV L,A	A point to seven segment code
	MOV A,M	Get the 7 segment code
	STAX B	Store result at destination memory location
	RET	Return to main program

Binary to ASCII Code Conversion

- The ASCII Code (American Standard Code for Information Interchange) is commonly used for communication.
- In such cases we need to convert binary number to its ASCII equivalent.
- It is a seven bit code.
- In this code number 0 to 9 are represented as 30H to 39H respectively and letters A to Z are represented as 41H to 5AH.
- Therefore, by adding 30H we can convert number into its ASCII equivalent and by adding 37H we can convert letter to its ASCII equivalent.
- Let us see the Program for binary to ASCII code conversion.

Binary to ASCII Code Conversion





Binary to ASCII Code Conversion

PROBLEM STATEMENT

An 8-bit binary number (e.g., 9FH) is stored in memory location XX50H.

1. Write a program to
   a. Transfer the byte to the accumulator.
   b. Separate the two nibbles (as 09 and 0F).
   c. Call the subroutine to convert each nibble into ASCII Hex code.
   d. Store the codes in memory locations XX60H and XX61H.
2. Write a subroutine to convert a binary digit (0 to F) into ASCII Hex code.

Binary to ASCII Code Conversion

MAIN PROGRAM

```
LXI SP,STACK    ;Initialize stack pointer
LXI H,XX50H     ;Point index where binary number is stored
LXI D,XX60H     ;Point index where ASCII code is to be stored
MOV A,M         ;Transfer byte
MOV B,A         ;Save byte
RRC             ;Shift high-order nibble to the position of low-order
RRC             ;  nibble
```

Binary to ASCII Code Conversion

```
        RRC
        RRC
        CALL ASCII      ;Call conversion routine
        STAX D          ;Store first ASCII Hex in XX60H
        INX D           ;Point to next memory location, get ready to store
                        ;   next byte
        MOV A,B         ;Get number again for second digit
        CALL ASCII
        STAX D
        HLT

ASCII:  ;This subroutine converts a binary digit between 0 and F to ASCII Hex
        ;   code
        ;Input: Single binary number 0 to F in the accumulator
        ;Output: ASCII Hex code in the accumulator
        ANI 0FH         ;Mask high-order nibble
        CPI 0AH         ;Is digit less than 10₁₀?
        JC CODE         ;If digit is less than 10₁₀, go to CODE to add 30H
        ADI 07H         ;Add 7H to obtain code for digits from A to F
CODE:   ADI 30H         ;Add base number 30H
        RET
```

ASCII Code to Binary Conversion

PROBLEM STATEMENT

Write a subroutine to convert an ASCII Hex number into its binary equivalent. A calling program places the ASCII number in the accumulator, and the subroutine should pass the conversion back to the accumulator.

SUBROUTINE

```
ASCBIN:   ;This subroutine converts an ASCII Hex number into its binary
          :   equivalent
          ;Input: ASCII Hex number in the accumulator
          ;Output: Binary equivalent in the accumulator

          SUI 30H     ;Subtract 0 bias from the number
          CPI 0AH     ;Check whether number is between 0 and 9
          RC          ;If yes, return to main program
          SUI 07H     ;If not, subtract 7 to find number between A and F
          RET
```

BCD Addition

Memory Address	Instruction	
	Mnemonics	Operands
	LXI	H,3501 H
	MOV	A, M
	INX	H
	ADD	M
	DAA	
	INX	H
	MOV	A, M
	STA	3505
	HLT	

BCD Addition

INPUTS

Memory Location	Numbers
3501	
3502	

OUTPUT

Memory location	Result
3505	

BCD Subtraction

Memory Address	Instruction	
	Mnemonics	Operands
	LDA	2051H
	MOV	C, A
	MVI	A, 99H
	SUB	C
	INR	A
	MOV	B,A
	LDA	2050H
	ADD	B
	DAA	
	STA	3050H
	HLT	

BCD Subtraction

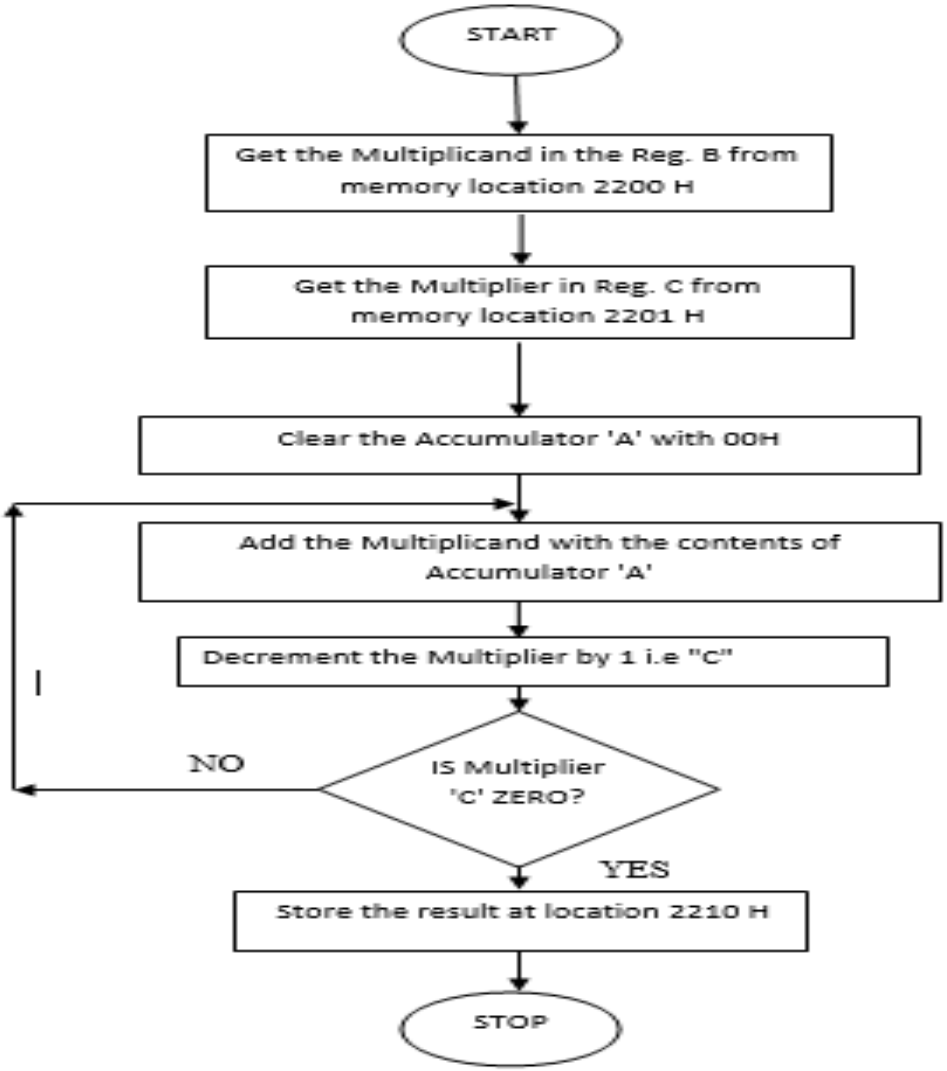
INPUTS

Memory Location	Numbers
2050	72
2051	35

OUTPUT

Memory location	Result
3050	

Multiplication



Multiplication

Memory Address	Label	Instruction	
		Mnemonics	Operands
		LDA	2200 H
		MOV	B,A
		LDA	2201 H
		MOV	C, A
		MVI	A, 00H
	go:	ADD	B
		DCR	C
		JNZ	go
		STA	2210 H
		HLT	

Multiplication

INPUTS:

Memory Location	Numbers
2200	(Multiplicand)
2201	(Multiplier)

OUTPUTS:

Memory location	Result
2210	

Daily Quiz

- PROM stands for:
  - **A. Programmable read-only memory**
  - B. Programmable read write memory
  - C. Programmer read and write memory
  - D. None of these

- The method of connecting a driving device to a loading device is known as _____
  a) Compatibility
  b) **Interface**
  c) Sourcing
  d) Sinking

Topic Links

https://www.youtube.com/watch?v=_EJDoLnJ5BM

https://www.youtube.com/watch?v=bXOpC0z-8o8

Faculty Video Links, Youtube & NPTEL Video Links and Online Courses Details

- Youtube/other Video Links
- https://www.youtube.com/watch?v=xBYhHC8_A6o
- https://www.youtube.com/watch?v=Xi_jRzyq42g
- https://www.youtube.com/watch?v=Xi_jRzyq42g

MCQ s

1)Single Flip-Flop can store ____ bits of information.

2)Which of the following components present in a computer?

A. CPU, Memory

B. Peripherals

C. Both a & b

D. None of the above

3)Program counter(PC) contains?

A. Address of the current instruction

B. Address of the next instruction to fetch

C. Value of the operand

D. starting address of the program

MCQ s

4)Instruction decoder

A. Holds the address of the current instruction

B. Contains next instruction to be fetch

C. Decodes the OPCODE and generates control signals

D. Fetches operands from memory

5)Decimal equivalent of(F4C) 16 is_____

6)ALU stands for

A. Arithmetic Logic Unit

B. Arithmetic Lower unit

C. Addition Logic Unit

D. AND Logic Unit

MCQ s

7)Which of the following sequence of operation conforms to an instruction cycle?

A. Fetch, Execute, Decode

B. Execute, Fetch, Decode

C. Fetch, Decode, Execute

D. Decode, Fetch, Execute

8)What is the high speed memory between the main memory and CPU called?

A. Magnetic Disk

B. Cache memory

C. DRAM

D. Secondary memory

Old Question Papers

Printed Pages: 2 **Sub Code: REC-405**

Paper Id: 131291 **Roll No.** | | | | | | | | | | |

B.TECH

(SEM IV) THEORY EXAMINATION 2018-19

INTRODUCTION TO MICROPROCESSOR

Time: 3 Hours ***Total Marks: 70***

Note: 1. Attempt all Sections. If require any missing data; then choose suitably.

SECTION A

1. Attempt *all* questions in brief. **2 x 7 = 14**

a. What are the functions of an accumulator?

b. Calculate the number of memory chips needed to design 128K-Byte memory if the Memory chip size is 2048 x 1.

c. Explain CALL & RET instructions used in 8085.

d. Explain the functions of the pins HOLD and HLDA in 8085 microprocessor.

e. What do you understand by logical address and physical address?

f. Draw the Flag register of 8086.

g. What do you mean by pipelining?

Old Question Papers

SECTION B

2. Attempt any *three* of the following: 7 x 3 = 21

a. Explain the evolution of microprocessor with its different generations in detail.

b. Write instructions to load two unsigned numbers in register B and register C, respectively. Subtract (C) from (B). If the result is in 2's complement, convert the result in absolute magnitude and display it at PORT1; otherwise, display the positive result. (Assume (B) =42H and (C) =69H).

c. What is the need of counters and time delays? Calculate the maximum time delay that can be produced by using a register pair.

d. Write an assembly language program for the addition of two BCD numbers stored at memory location starting from XX40H, store the result at memory locations starting from XX90H.

e. Explain the various modes of operation of 8254/53 with examples.

SECTION C

3. Attempt any *one* part of the following: 7 x 1 = 7

(a) Connect 8k byte EPROM with microprocessor 8085. The IC available is 2k × 8 EPROM, also draw its address decoding table.

(b) Draw and explain the architecture of 8085 microprocessor, also explain the programmer's model of 8085.

4. Attempt any *one* part of the following: 7 x 1 = 7

(a) Draw and explain the timing diagram of opcode fetch machine cycle.

(b) Define instruction and instruction cycle. Classify the instruction set of 8085 in different groups. Explain each group with two examples; also explain the functions of the examples.

5. Attempt any *one* part of the following: 7 x 1 = 7

(a) What are interrupts? Give the classification of interrupts. Explain the hardware and software interrupts used in 8085.

Old Question Papers

(b) Calculate the 16-bit count to be loaded in register DE to obtain the loop delay of two seconds in LOOP2 (assume the clock frequency of the system to be 2MHz)

```
        MVI B, 14H
LOOP2:  LXI D, COUNT
LOOP1:  DCX D
        MOV A, D
        ORA E
        JNZ   LOOP1
        DCR B
        JNZ   LOOP2
```

6. Attempt any *one* part of the following: 7 x 1 = 7

(a) Write an assembly language program to convert a binary number stored at memory location XX20H, into its equivalent ASCII-Hex code, store the codes at memory locations XX30H and XX31H.

(b) Write an assembly language program to convert $(56)_{BCD}$ to its equivalent binary number. The BCD number is stored at memory location XX50H, store the result in memory location XX60H.

7. Attempt any *one* part of the following: 7 x 1 = 7

(a) Draw and explain the functional block diagram of 8259 PIC. Also explain its Initialization Command Words.

(b) Draw and explain the architecture of 8086 microprocessor. Calculate the physical address if CS=9105H and IP=1724H.

Expected Questions for University Exam

- A processor has the size of its Read Only Memory (ROM) to be 8KB. If ROM chips of size 1Kb is only available, how many such chips are required to realize the ROM of the processor? Also, what is the minimum number of address lines required to realize the ROM? (Notation: b – bit, B – Byte)
- If the crystal frequency is 10 MHz, how much time is required to execute an instruction of 18-T states?
- If the 8085 adds 87H and 79H, specify the contents of the accumulator and status flags.
- Discuss the various logic devices used in interfacing circuits.
- Differentiate between microprocessor, micro-computer and microcontroller. What are the various operations performed by microprocessor?

Summary

- Differences between:
  - Microcomputer – a computer with a microprocessor as its CPU. Includes memory, I/O etc.
  - Microprocessor – silicon chip which includes ALU, register circuits & control circuits
  - Microcontroller – silicon chip which includes microprocessor, memory & I/O in a single package.

References

- Text Book:
  - Ramesh S. Goankar, “Microprocessor Architecture, Programming and Applications with 8085”, 5th Edition, Prentice Hall

ANY QUERIES?

Thank You