PERIYAR UNIVERSITY

NAAC 'A++' Grade – State University – NIRF Rank 56- State Public University Rank 25
SALEM - 636 011, Tamil Nadu, India

CENTRE FOR DISTANCE AND ONLINE EDUCATION

(CDOE)

B.SC COMPUTER SCIENCE

SEMESTER - II

CORE COURSE: MICROPROCESSOR AND MICROCONTROLLER – LAB

(Candidates admitted from 2024 onwards)

PERIYAR UNIVERSITY

CENTRE FOR DISTANCE AND ONLINE EDUCATION (CDOE)

B.Sc COMPUTER SCIENCE 2024 admission onwards

CORE COURSE – III
Microprocessor and Microcontroller-Lab

Prepared by:

Centre for Distance and Online Education (CDOE)
Periyar University, Salem – 11.

LIST OF CONTENTS

ADDITION AND SUBTRACTION

8 - BIT ADDITION

AIM:

To perform addition of two 8 bit numbers using 8085.

ALGORITHM:

1) Start the program by loading the first data into Accumulator.

2) Move the data to a register (B register).

3) Get the second data and load into Accumulator.

4) Add the two register contents.

5) Check for carry.

6) Store the value of sum and carry in memory location.

7) Terminate the program.

SOURCE CODE

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

SAMPLE INPUT & OUTPUT

Input:	80 (4150)
	80 (4251)
Output:	00 (4152)
	01 (4153)

RESULT

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

16 - BIT ADDITION

AIM:

To write an Assembly Language Program (ALP) for performing 16 bit addition.

ALGORITHM:

1. Initialize the MSBs of sum to 0
2. Get the first number.
3. Add the second number to the first number.
4. If there is any carry, increment MSBs of sum by 1.
5. Store LSBs of sum.
6. Store MSBs of sum

SOURCE CODE

LHLD	7601H	Get 1st no. in HL pair from memory 7601
XCHG		Exchange cont. of DE HL
LHLD	7603H	Get 2st no. in HL pair from location 7603
MVI	C, 00H	Clear reg. C.
DAD	D	Get HL+DE & store result in HL
JNC	LOOP	If no carry move to loop/if carry then move to next step.

```
        INR     C       Increment reg C
LOOP:   MOV     A, C    Move carry from reg. C to reg.A
        STA     7502H   Store carry at 7502H
        SHLD    7500H       Store result in 7500H.
        HLT
```

SAMPLE INPUT & OUTPUT

Input: 7601 77

7602 66

7603 44

7604 22

Output: 7502 BB

7503 88

7500 00

RESULT

Thus the program to add two 16-bit numbers was executed.

8 - BIT SUBTRACTION

AIM:

To perform subtraction of two 8 bit numbers using 8085.

ALGORITHM:

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

SOURCE CODE

MVI	C, 00	Initialize C register to 00
LDA	4150	Load the value to Accumulator.
MOV	B, A	Move the content of Accumulator to B register.
LDA	4151	Load the value to Accumulator.
SUB	B	Subtract the value of register B to A

	JNC	LOOP	Jump on no carry.
	CMA		Complement Accumulator contents
	INR	A	Increment value of register A
	INR	C	Increment value of register C
LOOP:	STA	4152	Store the value of Accumulator (SUM).
	MOV	A, C	Move content of register C to Acc.
	STA	4153	Store the value of Accumulator (CARRY)
	HLT		Halt the program.

SAMPLE INPUT & OUTPUT

Input:	06 (4150)
	02 (4251)
Output:	04 (4152)
	01 (4153)

RESULT

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

BCD SUBTRACTION

AIM:

To perform BCD subtraction of two 8 bit numbers using 8085.

ALGORITHM:

1. Load the data from address 2051 in A
2. Move the data from A to C
3. Move the data 99 in A
4. Subtract the contents of registers A and C
5. Increment the content of A by 1
6. Move the data from A to B
7. Load the data from address 2050 in A
8. Add the contents of A and C and adjust it in BCD format by using DAA instruction
9. Store the result at memory address 3050
10. Stop

SOURCE CODE

```
LDA     2051    A <- 2051
MOV     C, A    C <- A
MVI     A 99    A <- 99
SUB     C       A = A – C
INR     A       A = A + 1
MOV     B, A    B <- A
```

LDA	2050	A <- 2050
ADD	B	A = A + B
DAA		Convert the hexadecimal value to BCD value
STA	3050	3050 <- A
HLT		Stop

SAMPLE INPUT & OUTPUT

Input:	72 (2050)
	35 (2051)
Output:	37 (3050)

RESULT

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

MULTIPLICATION AND DIVISION

Section No	Topic	Page No
2	Multiplication and Division 1. 8 - bit multiplication 2. BCD multiplication 3. 8 - bit division	

8 - BIT MULTIPLICATION

AIM:

To perform the multiplication of two 8 bit numbers using 8085

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.

2) Move the data to a register (B register).

3) Get the second data and load into Accumulator.

4) Add the two register contents.

5) Check for carry.

6) Increment the value of carry.

7) Check whether repeated addition is over and store the value of product and carry in memory location.

8) Terminate the program.

SOURCE CODE

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

SAMPLE INPUT & OUTPUT

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

RESULT

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

BCD MULTIPLICATION

AIM:

To perform the BCD Multiplication of two 8 bit numbers using 8085

ALGORITHM:

1) places H'04 in R6H as a counter that indicates the number of BCD digits in the data; b. clears R2 and R3, where the result of multiplication is to be stored;
2) shifts R2 and R3 four bits (one BCD digit) to left;
3) loads one BCD digit from the higher-order end of the multiplier to R5L, and branches to step g. when R5L is "0";
4) BCD-adds the multiplicand to R2, R3 the same number of times as the value in R5L;
5) Decrements R6H; and g. repeats steps c to f above until R6H has become "0".
6) Terminate the program.
7)

SOURCE CODE

```
        MOV.    B  #H'04,R6H        ;Set bit counter1
        MOV.    W  #H'0000,R2       ;Clear R2
        MOV.    W  R2,R3            ;Clear R3
LBL1
        MOV.B   #H'04,R6L           ;Set bit counter2
        MOV.    B  #H'00,R5L        ;Clear R5L
```

```
LOOP1
        SHLL.    B  R0L          ;Shift multiplier 1 bit left
        ROTXL.   B R0H
        ROTXL.   B R5L
        SHLL.    B R3L          ;Shift result 1 bit left
        ROTXL.   B R3H
        ROTXL.   B R2L
        ROTXL.   B R2H
        DEC.     B R6L          ;Decrement bit counter 2
        BNE      LOOP1          ;Branch if Z=0
        CMP.B    #H'00,R5L
        BEQ      LBL2           ;Branch if Z=1
LOOP2
        ADD.     B  R1L,R3L     ;R1L + R3L -> R1L
        DAA.     B  R3L         ;Decimal adjust R3L
        ADDX.    B  R1H,R3H     ;R1H + R3H + C -> R1H
        DAA.     B  R3H         ;Decimal adjust R3H
        ADDX.    B  #H'00,R2L   ;R2L + #H'00 + C -> R2L
        DAA.B    R2L            ;Decimal adjust R2L
        ADDX.    B  #H'00,R2H   ;R2H + #H'00 + C -> R2H
```

```
        DAA.    B  R2H          ;Decimal adjust R2H
        DEC.    B  R5L          ;Clear bit 0 of R5L
        BNE     LOOP2           ;Branch if Z=0
LBL2 ;
        DEC.B   R6H             ;Decrement bit counter1
        BNE     LBL1            ;Branch if Z=0
        RTS
        END
```

SAMPLE INPUT & OUTPUT

Input: R 1 – 0 0 8 6

R 0 – 0 0 2 5

Output: R 2 – 0 0 0 0

R 3 – 2 1 5 0

RESULT

Thus the program to BCD Multiplication was executed

8 - BIT DIVISION

AIM:

To perform the division of two 8 bit numbers using 8085

ALGORITHM:

1) Start the program by loading HL register pair with address of memory location.

2) Move the data to a register (B register).

3) Get the second data and load into Accumulator.

4) Compare the two numbers to check for carry.

5) Subtract the two numbers.

6) Increment the value of carry.

7) Check whether repeated subtraction is over and store the value of product and carry in memory location.

8) Terminate the program.

SOURCE CODE

```
LXI     H, 4150
MOV     B, M        Get the dividend in B – reg.
MVI     C, 00       Clear C – reg for quotient
INX     H
MOV     A, M        Get the divisor in A – reg.
```

NEXT:	CMP	B	Compare A - reg with register B.
	JC	LOOP	Jump on carry to LOOP
	SUB	B	Subtract A – reg from B- reg.
	INR	C	Increment content of register C.
	JMP	NEXT	Jump to NEXT
LOOP:	STA	4152	Store the remainder in Memory
	MOV	A, C	Move the Content of C register to Accumulator
	STA	4153	Store the quotient in memory
	HLT		Terminate the program.

SAMPLE INPUT & OUTPUT

Input:	FF (4150)
	FF (4251)
Output:	01 (4152) ---- Remainder
	FE (4153) ---- Quotient

RESULT

Thus the program to divide two 8-bit numbers was executed

SORTING AND SEARCHING

Section No	Topic	Page No
3	Sorting and Searching 1. Searching for an element in an array. 2. Sorting in Ascending and Descending order. 3. Finding the largest and smallest elements in an array. 4. Reversing array elements. 5. Block move.	

Searching for an Element in an Array

AIM:

To program in 8085 to search a given number in an array of n numbers.

ALGORITHM:

1) Make the memory pointer points to memory location 2050 by help of LXI H 2050 instruction
2) Store value of array size in register C
3) Store number to be search in register B
4) Increment memory pointer by 1 so that it points to next array index
5) Store element of array in accumulator A and compare it with value of B
6) If both are same i.e. if ZF = 1 then store F0 in A and store the result in memory location 3051 and go to step 9
7) Otherwise store 0F in A and store it in memory location 3051
8) Decrement C by 01 and check if C is not equal to zero i.e. ZF = 0, if true go to step 3 otherwise go to step 9
9) End of program

SOURCE CODE

LXI H 2050	H <- 20, L <- 50	initialize register H and L
MOV C, M	C <- M	H and L → Memory M → Register C
LDA 3050	A <- M[3050]	3050 → A
MOV B, A	B <- A	A → B
INX H	HL <- HL + 0001	increment HL by 1
MOV A, M	A <- M	M → A
CMP B	A – B	subtract B from A
JNZ 2014	Jump if ZF = 0	jump to memory location 2014
		if zero flag is reset i.e. ZF = 0
MVI A F0	A <- F0	F0 → A
STA 3051	M[3051] <- A	stores value of A in 3051
HLT		END
MVI A 0F	A <- 0F	0F → A
STA 3051	M[3051] <- A	stores value of A in 3051
DCR C	C <- C – 01	decrement C by 01
JNZ 2008	Jump if ZF = 0	jump to memory location 2008
		if zero flag is reset
HLT		END

SAMPLE INPUT & OUTPUT

Input: 04 (2050) 49(2051) F2(2052) 14(2053) 39(2054)

Output: F2(3050) F0(3051)

If number is found, then store F0 in memory location 3051 otherwise store 0F in 3051.

RESULT

Thus the program was executed and verified.

Sorting in Ascending and Descending Order

AIM:

To write a program to find the Ascending and Descending order of n numbers in a given array.

ALGORITHM:

1) Initialize Order Flag and Array Address: Load order flag (0 for ascending, 1 for descending) into A and store in ORDER_FLAG; load array start address into HL (LXI H, 2000H).
2) Initialize Loop Counter: Load array size - 1 (15) into register C (MVI C, 0FH).
3) Outer Loop Start: Copy C to D for inner loop (MOV D, C).
4) Inner Loop Start: Reload array address into HL (LXI H, 2000H), load current element into A (MOV A, M), increment HL to next element (INX H), load next element into E (MOV E, M), and store it in B (MOV B, M).
5) Compare Elements: Compare current element (A) with next element (E) using CMP E.
6) Check Order Flag: Load order flag from ORDER_FLAG into A (LDA ORDER_FLAG), move to E (MOV E, A), check flag (JZ ASC_ORDER), if not zero, jump to descending order comparison (JMP DESC_ORDER).
7) Ascending Order Comparison: If flag is 0 and current element is less than next element, skip swap (ASC_ORDER: JC SKIP), else jump to swap (JMP SWAP).
8) Descending Order Comparison: If flag is 1 and current element is greater than or equal to next element, skip swap (DESC_ORDER: JNC SKIP).
9) Swap Elements if Needed: Swap elements: move next element to A (SWAP: MOV A, B), store in memory (MOV M, A), decrement HL (DCX H), store current element in memory (MOV M, E).
10) Update Memory Pointer: Increment HL to next element (SKIP: INX H).

11) Decrement Inner Loop Counter: Decrement D and repeat inner loop if D > 0 (DCR D, JNZ INNER).
12) Decrement Outer Loop Counter: Decrement C and repeat outer loop if C > 0 (DCR C, JNZ OUTER).
13) End of Program: Halt program (HLT).

SOURCE CODE

```
        ORG     0000H           ; Start address
        MVI     A, 00H          ; Set A to 0 for ascending order,
                                  Set A to 1 for descending order
        STA     ORDER_FLAG      ; Store the order flag
START:  LXI     H, 2000H        ; Load the starting address into HL pair
        MVI     C, 0FH          ; Load the count of elements into reg C
OUTER:  MOV     D, C            ; Copy the count to register D
        LXI     H, 2000H        ; Reload the starting address into HL pair
INNER:  MOV     A, M            ; Move the current element into register A
        INX     H               ; Increment HL
        MOV     E, M            ; Move the next element into register E
        MOV     B, M            ; Move the next element into register B
        CMP     E               ; Compare the element
```

```
        LDA     ORDER_FLAG      ; Load the order flag
        MOV     E, A            ; Move the flag into E
        JZ      ASC_ORDER       ; If flag is 0, jump to ascending order
        JMP     DESC_ORDER      ; Else, jump to descending order
ASC_ORDER:
        JC      SKIP            ; If A < M, skip the swap for ascending order
        JMP     SWAP            ; Else, swap the elements
DESC_ORDER:
        JNC     SKIP            ; If A >= M, skip the swap for descending order
SWAP:   MOV     A, B            ; Move the next element into A
        MOV     M, A            ; Swap the elements
        DCX     H               ; Decrement HL back to the current element
        MOV     M, E            ; Move the current element to the next position
SKIP:   INX     H               ; Increment HL to point to the next element
        DCR     D               ; Decrement the inner loop count
        JNZ     INNER           ; If inner loop count is not zero, repeat
        DCR     C               ; Decrement the outer loop count
        JNZ     OUTER           ; If outer loop count is not zero, repeat
        HLT                     ; Halt the program
```

INPUT: Data segment (array of 16 elements)

```
        ORG 2000H

        DB 0AH, 3CH, 2FH, 19H, 21H, 45H, 39H, 50H

        DB 27H, 1FH, 5AH, 36H, 78H, 4BH, 60H, 2DH


ORDER_FLAG DB 00H       ; Flag for order: 0 for ascending, 1 for descending

        END             ; End of the program
```

OUTPUT

Memory Location	Data	Memory Location	Data
2000H	10H	2000H	92H
2001H	12H	2001H	90H
2002H	21H	2002H	89H
2003H	23H	2003H	88H
2004H	33H	2004H	78H
2005H	35H	2005H	67H
2006H	44H	2006H	67H
2007H	54H	2007H	56H
2008H	56H	2008H	54H
2009H	67H	2009H	44H
200AH	67H	200AH	35H
200BH	78H	200BH	33H
200CH	88H	200CH	23H

200DH	89H	200DH	21H
200EH	90H	200EH	12H
200FH	92H	200FH	10H

RESULT

Thus the program was executed and verified.

Finding the largest and smallest elements in an array.

AIM:

To write a program to find the largest and smallest number in a given array elements.

Algorithm –

1) Maximum number is stored in B register and minimum in C register
2) Load counter in D register
3) Load starting element in Accumulator, B and C register
4) Compare Accumulator and B register
5) If carry flag is not set then transfer contents of Accumulator to B. Else, compare Accumulator with C register, if carry flag is set transfer contents of Accumulator to C
6) Decrement D register
7) If D>0 take next element in Accumulator and go to point 4
8) If D=0, store B and C register in memory
9) End of program

Source Code

```
        LXI   H, 2050H    Load starting address of list
        MOV   B, M        Store maximum
        MOV   C, M        Store minimum
        MVI   D, 0AH      Counter for 10 elements
LOOP    MOV   A, M        Retrieve list element in Accumulator
        CMP   B           Compare element with maximum number
        JC    MIN         Jump to MIN if not maximum
```

```
        MOV   B, A        Transfer contents of A to B as A > B
MIN     CMP   C           Compare element with minimum number
        JNC   SKIP        Jump to SKIP if not minimum
        MOV   C, A        Transfer contents of A to C if A < minimum
SKIP    INX   H           Increment memory
        DCR   D           Decrement counter
JNZ     LOOP              Jump to LOOP if D > 0
        LXI   H, 2060H    Load address to store maximum
        MOV   M, B        Move maximum to 2060H
        INX   H           Increment memory
        MOV   M, C        Move minimum to 2061H
        HLT               Halt
```

OUTPUT

LIST

42H 21H 01H 1FH FFH 25H 32H 34H 0AH ABH

Minimum : 01H Maximum: FFH

RESULT

Thus the program was executed and verified.

Reversing Array Elements

AIM:

To write a program to reverse a given array elements.

Algorithm –

1. Load content of memory location 2050 in accumulator A
2. Use RLC instruction to shift the content of A by 1 bit without carry. Use this instruction 4 times to reverse the content of A
3. Store content of A in memory location 3050

Source Code

```
LDA     2050    A <- M[2050]
RLC             Rotate content of accumulator left by 1 bit
RLC             Rotate content of accumulator left by 1 bit
RLC             Rotate content of accumulator left by 1 bit
RLC             Rotate content of accumulator left by 1 bit
STA     3050    M[2050] <- A
HLT             END
```

OUTPUT

98H in Binary Written as :

1001 1000

RLC 1st Time : 0011 0001 {Carry Flag = 1}

RLC 2nd Time : 0110 0010 {Carry Flag = 0}

RLC 3rd Time : 1100 0100 {Carry Flag = 0}

RLC 4th Time : 1000 1001 { Carry Flag = 1}

Converted Number after 4th RLC : 1000 1001 [89H]

Hence our number is reversed from 98H to 89H.

RESULT

Thus the program was executed and verified.

Block Move.

AIM:

To write a program to move the Blocks.

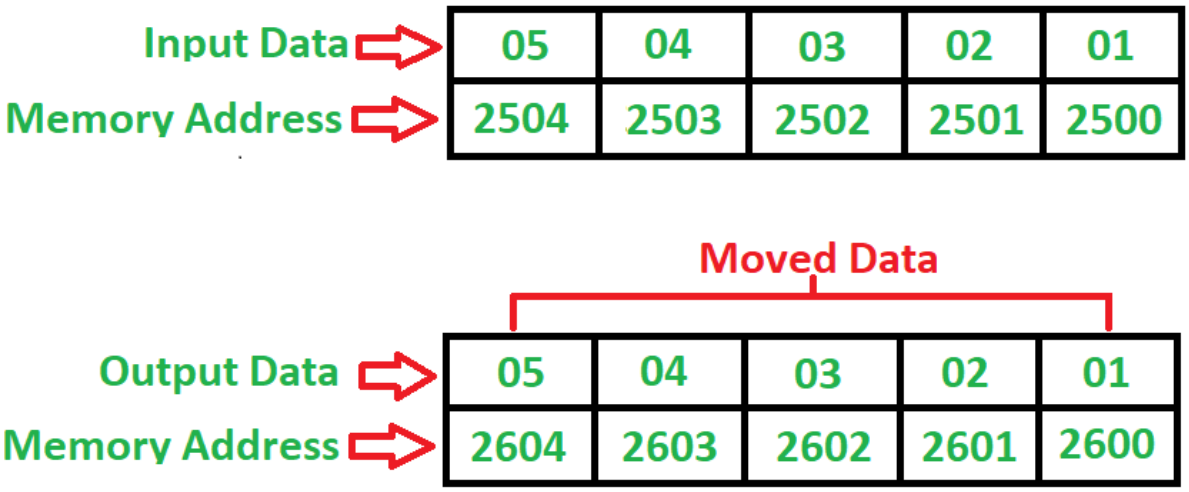
Algorithm –

1. Load register pair H-L with the address 2500H
2. Load register pair D-E with the address 2600H
3. Move the content at memory location into accumulator
4. Store the content of accumulator into memory pointed by D-E
5. Increment value of register pair H-L and D-E by 1
6. Decrements value of register C by 1
7. If zero flag not equal to 1, go to step 3
8. Stop

Source Code

MVI	C, 05	[C] <- 05
LXI	H, 2500	[H-L] <- 2500
LXI	D, 2600	[D-E] <- 2600
MOV	A, M	[A] <- [[H-L]]
STAX	D	[A] -> [[D-E]]
INX	H	[H-L] <- [H-L] + 1
INX	D	[D-E] <- [D-E] + 1
DCR	C	[C] <- [C] – 1
JNZ	2008	Jump if not zero to 2008
HLT		Stop

Input & Output



RESULT

Thus the program was executed and verified.

CODE CONVERSION

Section No	Topic	Page No
IV	Code Conversion 1. BCD to Hex and Hex to BCD 2. Binary to ASCII and ASCII to binary 3. ASCII to BCD and BCD to ASCII	

BCD to Hex and Hex to BCD

1 A) BCD to HEX

Aim

To develop a program that converts a Binary-Coded Decimal (BCD) number to its hexadecimal (Hex) equivalent using a microprocessor.

Algorithm

1. **Initialize the BCD input**: Load the BCD number into a register.
2. **Extract the digits**: Separate the BCD number into its individual digits.
3. **Convert digits**: Convert each BCD digit into its hexadecimal form.
4. **Combine digits**: Combine the converted hexadecimal digits into a single hexadecimal number.
5. **Store/Display the result**: Store or display the hexadecimal result.

Source Code

```
ORG  0000H
MVI  D, 00H     ; Clear D register for storing hex result
MVI  C, 00H     ; Clear C register for storing intermediate result
                ; Load BCD number into register A
```

```
MVI    A, 27H      ; Let's assume BCD number is 27H
                   ; Extract lower BCD digit (units place)
ANI    0FH         ; Mask upper nibble to get lower BCD digit
MOV    B, A        ; Store lower digit in B register
                   ; Extract upper BCD digit (tens place)
MOV    A, 27H      ; Reload original BCD number
ANI    F0H         ; Mask lower nibble to get upper BCD digit
RRC                ; Rotate right to bring upper digit to lower nibble
RRC
RRC
RRC
MOV    C, A        ; Store upper digit in C register
                   ; Convert BCD digits to Hex
MOV    A, C        ; Load upper digit into A
ADD    A, A        ; Multiply by 10 (shift left by 4)
ADD    A, A        ; Continue multiplication (shift left by 1)
ADD    A, A        ; Continue multiplication (shift left by 1)
ADD    A, C        ; Add original upper digit to get tens place in hex
MOV    C, A        ; Store in C register
MOV    A, C        ; Load result back into A
ADD    A, B        ; Add lower digit
MOV    D, A        ; Store final result in D register
HLT                ; End of program
```

Sample Output

Assuming the input BCD number is 27H, the sample output after execution will be stored in the D register.

Result

For the given input 27H (BCD), the hexadecimal equivalent is 1BH. After executing the program, the D register will contain the value 1BH.

1 B) HEX to BCD

Aim

To develop a program that converts a Hexadecimal (Hex) number to its Binary-Coded Decimal (BCD) equivalent using a microprocessor.

Algorithm

1. **Initialize the Hex input**: Load the Hex number into a register.
2. **Extract digits**: Separate the Hex number into its individual digits.
3. **Convert digits**: Convert each Hex digit into its BCD form.
4. **Combine digits**: Combine the converted BCD digits into a single BCD number.
5. **Store/Display the result**: Store or display the BCD result.

Source Code

```
ORG   0000H
MVI   D, 00H      ; Clear D register for storing BCD result
MVI   E, 00H      ; Clear E register for storing intermediate result
                  ; Load Hex number into register A
 MVI  A, 1BH      ; Let's assume Hex number is 1BH
                  ; Extract lower Hex digit (units place)
ANI   0FH         ; Mask upper nibble to get lower Hex digit
MOV   B, A        ; Store lower digit in B register
                  ; Extract upper Hex digit (tens place)
MOV   A, 1BH      ; Reload original Hex number
 ANI  F0H         ; Mask lower nibble to get upper Hex digit
```

```
RRC                    ; Rotate right to bring upper digit to lower nibble
RRC
RRC
RRC
MOV  C, A      ; Store upper digit in C register
                ; Convert Hex digits to BCD
MOV  A, C      ; Load upper digit into A
ADI  03H       ; Add 3 times 10 to convert upper hex digit to BCD
MOV  C, A      ; Store in C register
MOV  A, B      ; Load lower digit into A
ADD  A, C      ; Add lower digit
MOV  D, A      ; Store final result in D register
HLT          ; End of program
```

Sample Output

Assuming the input Hex number is `1BH`, the sample output after execution will be stored in the D register.

Result

For the given input `1BH` (Hex), the BCD equivalent is `27H`. After executing the program, the D register will contain the value `27H`.

Binary to ASCII and ASCII to binary

1 A) Binary to ASCII

Aim

To develop a program that converts a binary number to its ASCII equivalent using a microprocessor.

Algorithm

1. **Initialize the binary input**: Load the binary number into a register.
2. **Convert to Decimal**: Convert the binary number to its decimal equivalent.
3. **Convert to ASCII**: Convert each decimal digit to its ASCII equivalent.
4. **Store/Display the result**: Store or display the ASCII result.

Source Code

```
ORG   0000H
                ; Load binary number into register A (let's assume the
      binary number is 1101, which is 13 in decimal)
MVI   A, 0DH      ; Binary 1101 in decimal is 13 (0DH in hexadecimal)
                ; Convert binary to decimal (since the number is
already in decimal form in A, we can use it directly)
                ; The number 13 will be split into two digits: '1' and '3'
                ; Extract tens place (1 in this case)
MVI   B, 0AH      ; Load 10 into B for division
MOV   C, A        ; Copy A to C for division
MVI   D, 00H      ; Clear D for storing the quotient
CALL  DIVISION    ; Perform division (C / B)
MOV   E, A        ; Store quotient (tens place) in E
                  ; Multiply quotient by 10 to get tens value (10 in this case)
MOV   A, E        ; Move tens place to A
```

```
        MOV   B, 0AH      ; Load 10 into B
        CALL MULTIPLY   ; Perform multiplication
        MOV   E, A        ; Store result in E (E now contains 10)

                          ; Subtract the tens value from the original number to get the
units place
        MOV   A, C        ; Load original number (13) into A
        SUB   E           ; Subtract tens value (10) from A
        MOV   D, A        ; Store result (units place) in D
                          ; Convert tens place to ASCII
        MOV   A, E        ; Load tens place into A
        ADI   30H         ; Add 30H to convert to ASCII
        STA   8000H       ; Store the ASCII character at memory location 8000H
                          ; Convert units place to ASCII
        MOV   A, D        ; Load units place into A
        ADI   30H         ; Add 30H to convert to ASCII
        STA   8001H       ; Store the ASCII character at memory location 8001H
        HLT               ; End of program
                        ; Subroutine for division
    DIVISION:
        MVI   D, 00H      ; Clear D for quotient
      DIV_LOOP:
        CMP   B       ; Compare A with B
        JC    DIV_END  ; If A < B, end division
        SUB   B       ; A = A - B
        INR   D       ; Increment D (quotient)
        JMP   DIV_LOOP; Repeat loop
      DIV_END:
        MOV   A, D    ; Move quotient to A
        RET
                        ; Subroutine for multiplication
```

```
MULTIPLY:
        MVI    D, 00H      ; Clear D for product
   MUL_LOOP:
        CMP   C                ; Compare C with 0
        JZ      MUL_END  ; If C == 0, end multiplication
         ADD   A       ; A = A + A
        DCR   C       ; Decrement C
        JMP   MUL_LOOP; Repeat loop
         MUL_END:
       RET
```

Sample Output

Assuming the input binary number is 1101 (which is 13 in decimal), the sample output after execution will store the ASCII characters for '1' and '3' in memory locations 8000H and 8001H respectively.

Result

For the given input 1101 (binary), the ASCII equivalent is '13'. After executing the program, the memory locations 8000H and 8001H will contain the values 31H (ASCII for '1') and 33H (ASCII for '3') respectively.

1 B) ASCII to BINARY

Aim

To develop a program that converts an ASCII character representing a decimal digit to its binary equivalent using a microprocessor.

Algorithm

1. **Initialize the ASCII input**: Load the ASCII character into a register.
2. **Convert to Decimal**: Subtract 30H from the ASCII value to get the decimal digit.
3. **Convert to Binary**: The result is already in binary format as a single decimal digit.
4. **Store/Display the result**: Store or display the binary result.

Source Code

```
        ORG  0000H      ; Load ASCII character into register A (let's assume the ASCII character is '5', which is 35H in ASCII)
        MVI  A, 35H     ; ASCII for '5'
                        ; Convert ASCII to Decimal (subtract 30H from ASCII value)
        SUI  30H        ; Subtract 30H to get decimal digit (5 in this case) ; The result in A is now the binary equivalent of the ASCII character
        STA  8000H      ; Store the binary result at memory location 8000H
        HLT             ; End of program
```

Sample Output

Assuming the input ASCII character is 35H ('5'), the sample output after execution will store the binary value 05H in memory location 8000H.

Result

For the given input 35H (ASCII for '5'), the binary equivalent is 00000101 (5 in decimal).

ASCII to BCD and BCD to ASCII

ASCII to BCD

Aim

To develop a program that converts an ASCII character representing a decimal digit to its Binary-Coded Decimal (BCD) equivalent using a microprocessor.

Algorithm

1. **Initialize the ASCII input**: Load the ASCII character into a register.
2. **Convert to Decimal**: Subtract 30H from the ASCII value to get the decimal digit.
3. **Convert to BCD**: The result is already in BCD format as it is a single decimal digit.
4. **Store/Display the result**: Store or display the BCD result.

Source Code

```
ORG  0000H
        ; Load ASCII character into register A (let's assume the ASCII character is '5', which is 35H in ASCII)
MVI  A, 35H     ; ASCII for '5'
        ; Convert ASCII to Decimal (subtract 30H from ASCII value)
SUI  30H        ; Subtract 30H to get decimal digit (5 in this case)
                ; The result in A is now the BCD equivalent of the ASCII character
 STA  8000H     ; Store the BCD result at memory location 8000H
HLT             ; End of program
```

Sample Output

Assuming the input ASCII character is 35H ('5'), the sample output after execution will store the BCD value 05H in memory location 8000H.

Result

For the given input 35H (ASCII for '5'), the BCD equivalent is 05H. After executing the program, the memory location 8000H will contain the value 05H.

.

BCD to ASCII

Aim

To develop a program that converts a Binary-Coded Decimal (BCD) number to its ASCII equivalent using a microprocessor.

Algorithm

1. **Initialize the BCD input**: Load the BCD number into a register.
2. **Convert to Decimal**: The BCD number is already in decimal form, so no conversion is needed.
3. **Convert to ASCII**: Add 30H to the decimal digit to get its ASCII equivalent.
4. **Store/Display the result**: Store or display the ASCII result.

Source Code

```
ORG 0000H     ; Load BCD number into register A
MVI     A, 05H      ; BCD for 5
                    ; Convert BCD to ASCII (add 30H to the BCD value)
 ADI    30H         ; Add 30H to get ASCII value for '5'
; The result in A is now the ASCII equivalent of the BCD number
 STA    8000H       ; Store the ASCII result at memory location 8000H
HLT                 ; End of program
```

Sample Output

Assuming the input BCD number is 05H, the sample output after execution will store the ASCII value 35H in memory location 8000H.

Result

For the given input 05H (BCD for 5), the ASCII equivalent is 35H (ASCII for '5').

SIMPLE PROGRAMS ON 8051 MICROCONTROLLER

Section No	Topic	Page No
V	Simple programs on 8051 Microcontroller 1. Addition 2. Subtraction 3. Multiplication 4. Division 5. Interfacing Experiments using 8051 Realisation of Boolean Expression through ports. Time delay generation using subroutines. Display LEDs through ports	

Addition

Aim

To develop a program that performs the addition of two 8-bit numbers using the 8051 microcontroller.

Algorithm

1. **Initialize the first number**: Load the first number into register A.
2. **Initialize the second number**: Load the second number into another register (e.g., R0).
3. **Perform Addition**: Add the two numbers and store the result in register A.
4. **Store/Display the result**: Store the result in a memory location or output port.

Source Code

```
ORG   0000H          ; Load the first number into register A
```

```
MOV   A, #25H      ; Let's assume the first number is 25H
                   ; Load the second number into register R0
 MOV R0, #17H      ; Let's assume the second number is 17H
                   ; Perform the addition
 ADD   A, R0       ; Add the contents of R0 to A
                   ; Store the result in memory location 30H
MOV   30H, A       ; Store the result at memory location 30H
                   ; End of program
SJMP $
```

Sample Output

Assuming the input numbers are 25H and 17H, the sample output after execution will store the result 3CH in memory location 30H.

Result

For the given inputs 25H and 17H, the addition result is 3CH. After executing the program, the memory location 30H will contain the value 3CH.

Subtraction

Aim

To develop a program that performs the subtraction of two 8-bit numbers using the 8051 microcontroller.

Algorithm

1. **Initialize the first number**: Load the first number into register A.
2. **Initialize the second number**: Load the second number into another register (e.g., R0).
3. **Perform Subtraction**: Subtract the second number from the first number and store the result in register A.
4. **Store/Display the result**: Store the result in a memory location or output port.

Source Code

```
ORG   0000H
                    ; Load the first number into register A
MOV   A, #25H     ; Let's assume the first number is 25H
                    ; Load the second number into register R0
MOV   R0, #17H    ; Let's assume the second number is 17H
                    ; Perform the subtraction
SUBB A, R0       ; Subtract the contents of R0 from A
                  ; Store the result in memory location 30H
MOV   30H, A      ; Store the result at memory location 30H
                  ; End of program
SJMP $
```

Sample Output

Assuming the input numbers are 25H and 17H, the sample output after execution will store the result 0EH in memory location 30H.

Result

For the given inputs 25H and 17H, the subtraction result is 0EH. After executing the program, the memory location 30H will contain the value 0EH.

Multiplication

Aim

To develop a program that performs the multiplication of two 8-bit numbers using the 8051 microcontroller.

Algorithm

1. **Initialize the first number**: Load the first number into register A.
2. **Initialize the second number**: Load the second number into register B.
3. **Perform Multiplication**: Use the MUL AB instruction to multiply the numbers in registers A and B.
4. **Store/Display the result**: Store the result in designated registers or memory locations.

Source Code

```
ORG   0000H    ; Load the first number into register A
MOV   A, #12H     ; Let's assume the first number is 12H
                        ; Load the second number into register B
MOV B, #0AH     ; Let's assume the second number is 0AH
                ; Perform the multiplication
MUL   AB          ; Multiply A and B, result is in A (lower byte) B (upper byte)
                ; Store the result in memory locations 30H (low byte) and 31H (high byte)
MOV   30H, A      ; Store the low byte of the result at memory location 30H
MOV   31H, B      ; Store the high byte of the result at memory location 31H
                ; End of program
SJMP $
```

Sample Output

Assuming the input numbers are 12H and 0AH, the sample output after execution will store the result 00H (high byte) and C0H (low byte) in memory locations 31H and 30H respectively.

Result

For the given inputs 12H and 0AH, the multiplication result is 00C0H (192 in decimal).

Division

Aim

To develop a program that performs integer division of two 8-bit numbers using the 8051 microcontroller.

Algorithm

1. **Initialize the dividend**: Load the dividend into register A.
2. **Initialize the divisor**: Load the divisor into register B.
3. **Perform Division**:
    - Subtract the divisor from the dividend until the dividend is less than the divisor.
    - Count the number of subtractions to get the quotient.
    - The remainder is left in register A after the division process.
4. **Store/Display the result**: Store the quotient and remainder in designated registers or memory locations.

Source Code

```
        ORG   0000H       ; Load the dividend into register A
        MOV   A, #75      ; Let's assume the dividend is 75
                          ; Load the divisor into register B
        MOV   B, #10      ; Let's assume the divisor is 10
                          ; Initialize the quotient and remainder
        MOV R2, #00H      ; Quotient (initialize to 0)
        MOV R3, A         ; Remainder (initialize to the dividend)
DIV_LOOP:
                          ; Subtract the divisor from the remainder
        SUBB A, B
```

```
                    ; If A is still positive, increment the quotient and
                repeat
        JNC   DIV_LOOP_END
        INC   R2
        SJMP  DIV_LOOP
DIV_LOOP_END:
                    ; The remainder is in A, quotient is in R2
                    ; Store the quotient and remainder in memory
        MOV   30H, R2     ; Store quotient at memory location 30H
        MOV   31H, A      ; Store remainder at memory location 31H
                    ; End of program
        SJMP $
```

Sample Output

Assuming the input numbers are 75 (dividend) and 10 (divisor), the sample output after execution will store the quotient 7 and remainder 5 in memory locations 30H and 31H respectively.

Result

For the given inputs 75 (dividend) and 10 (divisor):

- The quotient is 7.
- The remainder is 5. After executing the program, the memory location 30H will contain the quotient 07H and 31H will contain the remainder 05H.

Interfacing Experiments using 8051

1. Realisation of Boolean Expression through ports.

Aim

To develop an experiment using the 8051 microcontroller to realize a Boolean expression through its ports.

Algorithm

1. **Initialize the ports**: Configure the ports of the 8051 microcontroller for input and output.
2. **Implement the Boolean expression**: Write the code to evaluate a Boolean expression using the inputs from the ports.
3. **Display the result**: Output the result of the Boolean expression using the LEDs connected to the ports.

Source Code:

```
        ORG   0000H         ; Configure Port 0 (P0) for input (switches)
        MOV   P0, #0FFH     ; Configure P0.0-P0.7 as input
                            ; Configure Port 1 (P1) for output (LEDs)
        MOV P1, #00H        ; Initialize P1.0-P1.7 as output
                            ; Read inputs (switches)
        MOV   A, P0         ; Read the state of Port 0 (switches)
                            ; Extract variables from inputs
        MOV B, A            ; Store inputs in register B
                            ; Implement Boolean expression: Y = A * B' + C
                    ; A is connected to P0.0, B is connected to P0.1, and C is
connected to P0.2
        MOV C, A                    ; Move inputs to C
        ANL C, #01H     ; A AND B
```

ANL A, #02H

Result:

Thus the program was executed successfully

2. Time delay generation using subroutines.

Aim

To develop an experiment using the 8051 microcontroller to generate time delays using subroutines.

Algorithm

1. **Initialize the ports**: Configure the ports of the 8051 microcontroller for output (LEDs).
2. **Implement time delay subroutine**: Write a subroutine to generate a specific time delay.
3. **Call the subroutine**: Call the subroutine from the main program to generate the desired time delay.
4. **Display the time delay**: Use LEDs to indicate the duration of the time delay.

Source Code

```
        ORG   0000H         ; Configure Port 1 (P1) for output (LEDs)
        MOV   P1, #00H      ; Initialize P1.0-P1.7 as output
                            ; Main program loop
MAIN:
                            ; Generate a delay of 1 second (assuming a clock frequency
of 11.0592 MHz)
        CALL  DELAY_1S      ; Toggle LEDs on Port 1
        CPL   P1
        SJMP  MAIN          ; Loop indefinitely
                            ; Subroutine to generate a delay of approximately 1 second
        DELAY_1S:
        MOV R2, #250        ; Initialize outer loop counter
OUTER_LOOP:
        MOV   R1, #250      ; Initialize inner loop counter
```

```
INNER_LOOP:
    DJNZ R1, INNER_LOOP ; Decrement R1, jump if not zero
    DJNZ R2, OUTER_LOOP ; Decrement R2, jump if not zero
    RET                 ; Return from subroutine
```

Sample Input and Output

- **Input**: None (switches or external input are not used in this example)
- **Output**: LEDs on Port 1 will toggle every second

Result

- LEDs connected to Port 1 will toggle every second, indicating the successful generation of the time delay.

3. Display LEDs through ports

Aim

To develop an experiment using the 8051 microcontroller to display LEDs through its ports.

Algorithm

1. **Initialize the ports**: Configure the ports of the 8051 microcontroller for output (LEDs).
2. **Toggle LEDs**: Write a program to toggle LEDs connected to the ports.
3. **Display the output**: Observe the LEDs to verify they are toggling correctly.

Experiment Setup

- **Outputs**: LEDs connected to Port 1 (P1).

Source Code

```
        ORG   0000H         ; Configure Port 1 (P1) for output (LEDs)
        MOV   P1, #00H      ; Initialize P1.0-P1.7 as output
                            ; Main program loop
    MAIN:
                              ; Toggle LEDs on Port 1
        CPL   P1            ; Complement all bits of P1
                            ; Generate a short delay to see LED blinking (assuming a
                clock frequency of 11.0592 MHz)
      CALL DELAY_SHORT
```

```
    SJMP MAIN       ; Loop indefinitely

; Subroutine to generate a short delay
DELAY_SHORT:
    MOV R2, #20     ; Outer loop counter
OUTER_LOOP:
    MOV R1, #30     ; Inner loop counter
INNER_LOOP:
    DJNZ R1, INNER_LOOP ; Decrement R1, jump if not zero
    DJNZ R2, OUTER_LOOP ; Decrement R2, jump if not zero
    RET             ; Return from subroutine
```

Sample Input and Output

- **Input**: None (switches or external input are not used in this example)
- **Output**: LEDs on Port 1 will toggle on and off, creating a blinking effect.

Result

- LEDs connected to Port 1 will blink on and off, indicating successful interfacing and program execution.