Unit-4

Note: Mainly programs are imp from this unit theory is for understanding

Assembly Language Programming

A. Programming with Intel 8085 microprocessor:

(a) Instruction and Data Format: (less imp)

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

i) One byte instruction → The one byte instruction is always the opcode. For example: MOV A, B whose opcode is 78H which is one byte. The instruction and data format of 8085 copies the contents of B register in A register.

ii) Two byte instruction → The two byte instruction is first byte Opcode and second byte operand. For example: MVI B, 02H. The opcode for this instruction is 06H and is always followed by a byte data (02H in this case). This instruction is a two byte instruction which copies immediate data into B register.

iii) Three byte Instruction → In three byte instruction first is opcode, second byte is Operand and third byte is also operand. For example: JMP 6200H. The opcode for this instruction is C3H and is always followed by 16-bit address (6200H in this case). The instruction is a three byte instruction which loads 16-bit address into program counter.

Date.
Page No.

Data Format:

The operand in an another name is data. It may appear in different forms as follows:-

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

ii) Numbers / Data → The 8085 supports following numeric data types.

(a) Signed integer → A signed integer number is either a positive number or a negative number. In 8085, 8-bits are assigned for signed integer, in which most significant bit is used for sign ~~bit~~ 0 and remaning seven bits are used for sign bit 0 indicates positive number whereas sign bit 1 indicates negative number.

(b). Unsigned integer → The 8085 microprocessor supports 8-bit unsigned integer.

(c). BCD → The term BCD number stands for binary coded decimal number. It uses ten digits 0 through 9. The 8-bit register of 8085 can store two digit BCD.

iii) Characters → The 8085 uses ASCII code to represent characters. It is a 7-bit alphanumeric code that represents decimal number's, English alphabets, and other special characters.

Date.
Page No.

Mnemonic and Opcode: (less imp).

Mnemonic: A mnemonic is a term or symbol or name used to define or specify a computing function. Mnemonics are used in computing to provide users with a means to quickly access a function, serivce or process, by passing the actual more lengthy method used to perform or achiere it. Assembly language also uses a mnemonic to represent machine operation, or opcode.

In computer programming, routine programming functions are assigned a mnemonic that is shorter in length but provides the same functionality as the original function.

For example, the mnemonic MOV is used in assembly language for copying and moving data between registers and memory locations.

Opcode: An opcode identifies which basic computer operation in the instruction set is to be performed. It is used when writing machine code. It tells the computer to do something. Each machine language instruction typically has both an opcode and operands. The opcode is like a verb in a sentence, and the operands are like the subject in a sentence. Operands are typically memory or registry addresses.

Opcodes are seldom used by programmers. In assembly language, a translator program converts program statements, one-for-one, into machine language commands.

refer pdf of Instruction set of Intel 8085 microprocessor for shorter and better.

Date.
Page No.

(*) Instruction Sets: (less imp)

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions that a microprocessor supports is called instruction set. 8085 has 246 instructions. Each instruction is represented by an 8-bit binary value. These 8-bits of value is called Op-code or Instruction Byte.

Following are the instruction set in microprocessor.

1) Data transfer instructions → These instructions move data between registers or between memory and registers. These instructions copy data from source to destination. While copying, the source of contents are not modified.

Following are some data transfer instructions.

(a) MOV → This instruction copies the contents of the source register into the destination register. The contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers.

Example: MOV B,C , MOV B,M , MOV M,C etc.

(b) MVI → The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the H-L registers.

Example: MVI A, 57H , MVI M, 57H.

(c) LXI → This instruction load register pair immediate. It loads 16-bit data in the register pair. Example: LXI H, 2034H.

Date.
Page No.

(d) LDA → This instruction copies the contents of a memory location, specified by a 16-bit address in the operand into accumulator. The contents of the source are not altered.
Example: LDA 2034H.

(e) LDAX → The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator.
Example LDAX B.

(f) STA → The contents of accumulator are copied into the memory location specified by the operand. It stores directly in the accumulator. Example: STA 2500H.

(g) STAX → The contents of accumulator are copied into the memory location specified by the contents of the register pair. Example: STAX B.
It stores indirectly in the accumulator.

(h) XCHG → The contents of register H (H-L pair) are exchanged with the contents of register D (D-E pair).
OR the contents of register L are exchanged with the contents of register E. Example: XCHG.

2) Arithmetic and Logic instructions:

Arithmetic Group

(a) ADD → ~~ADD~~ This instruction Add memory to accumulator or register to accumulator or add memory with carry to accumulator.
Example ADD r, ADD M, ADC M etc.

Date.
Page No.

(b) SUB → This instruction subtract register from accumulator or subtract memory from accumulator or subtract memory from accumulator with borrow. Example: SUB r, SUB M, SBB r etc.

(c) DAD → This instruction add register pair to H-L pair. The result is stored in H-L pair. No other flags are changed but CY is set.

(d) INR → This instruction increments register or memory by 1. Example: INR B or INR M.

(e) INX → This instruction increments register pair by 1. Example: INX H

(f) DCR → This instruction decreaments register or memory by 1. Example: DCR B, or DCR M.

(g) DCX → This instruction decrements register pair by 1. Example. DCX H.

Logical group

(a) AND, OR, XOR (XRA, XRI) operation
(ANA, ANI) (XRA, ORA, ORI)

(b) Compare → CMP. i.e, compare ~~immediate with~~ register or memory accumulator.
CPI i.e, compare immediate with accumulator.

(c) Rotate → RLC → rotate accumulator left
RRC → rotate accumulator right
RAL → rotate accumulator left through carry
RAR → rotate accumulator right through carry.

Date.
Page No.

(d) Complement →

CMA → Complement accumulator
CMC → Complement carry
STC → Set carry.

3) Branching instructions:

The branching instruction alter the normal sequential flow. These instructions alter either unconditionally or conditionally.

JMP → Jump unconditionally
Jx → Jump conditionally
JC → Jump if carry
JNC → Jump if No carry.
JM → Jump if Minus
JZ → Jump if zero.
JNZ → Jump if No zero.
JPE → Jump if Parity Even.
JPO → Jump if Parity Odd.
CALL → Call unconditionally.
Cx → Call conditionally.

4) Control Instructions:

NOP → No operation.
HLT → Halt
DI → Disable interrupt
EI → Enable interrupt
RIM → Read Interrupt mask
SIM → Set Interrupt mask.

5) Stack Instructions:

PUSH and POP.

From This is the imp section

Date.
Page No.

8085 microprocessor programs:

1. Write Assembly Language Program (ALP) to add two 8-bit numbers.

Soln.

```
        MVI C,00H ; Initilize carry as zero.
        LDA 2000H ; load the first 8-bit data.
        MOV B,A ; Move the content of A into register B.
        LDA 2001H ; Load the second 8-bit data.
        ADD B ; Add both the values.
        JNC skip ; Jump to skip if carry does not occur.
        INR C ; Increament C register by 1.
skip:   STA 2002H ; Store the result in accumulator.
        MOV A,C ; Move carry from register C to accumulator
        STA 2003H ; Store the result in accumulator.
        HLT ; Stop the program.
```

2. Write ALP to subtract two 8-bit numbers.

Soln

(assumed memory location)

```
        MVI C,00H ; Initilize carry as zero.
        LDA 4300H ; Load the first 8-bit data.
        MOV B,A ; Copy the value into register B.
        LDA 4301H ; Load the second 8-bit data.
        SUB B ; Subtract both the values.
        JNC loop ; Jump on loop if no borrow.
        INR C ; If borrow is there, increament by one
loop:   CMA ; Complement of 2nd data.
        ADI 01 ; Add 1 to 1's complement of 2nd data.
        STA 4302H ; Store the result in accumulator.
        MOV A,C ; Move the value of borrow from reg.C to acc.
        STA 4303H ; Store the result in accumulator.
        HLT ; Stop the program.
```

Note: ANI 01 is used to test even/odd
ANI 80 is used to test negative/positive.

If INR comes it is increment for 8-bit data
& if INX comes it is increment for 16-bit data.

3. Write ALP to find sum of series.

Soln

```
LDA 4200H; Length of series is loaded into Acc.
MOV C,A; move length of series into counter register C.
SUB A; Set sum=0 (i.e, Accumulator holds value of sum).
LXI H, 4201; Points program counter to 4201.
Back: ADD M; Add the content of memory pointed by PC with acc.
      INX H; Increment pointer to next memory location.
      DCR C; Decrement counter register.
      JNZ Back; Repeat addition if counter not equal to zero.
      STA 4300H; Store the result into memory.
      HLT.; Stop the program.
```

4. Find the number of negative elements in a block of data. The length of block is in memory location 2200H and the block itself begins in memory location 2201. Store the number of negative elements in memory location 2300H.

Soln

```
LDA 2200H; length of block is loaded into acc.
MOV C,A; move length of block into counter register C.
MVI B,00; Initilize B register to zero.
LXI H,2201; Points program counter to 4201.
Loop: MOV A,M; Copy/Move the content of memory into acc.
      ANI 80H; Test the number positive or negative.
      JZ SKIP; Jump to skip if zero occurs.
      INR B; Increament B register by one.
SKIP: DCR C; Decrement register C by one.
      INX H; Increment pointer to next memory location.
      JNZ Loop; Jump to Loop if carry does not occur.
      MOV A,B; Move the content of register B into acc.
      STA 2300H; Store the result into memory.
      HLT.; Stop the program.
```

If we use JNZ inplace of JZ in this program then it is for checking positive

M denotes, memory pointed by H-L pair.

Date.
Page No.

5. Write ALP to count number of even numbers in given array.

Soln

```
LDA 2200H
MOV C,A
MVI B,00H
LXI H,2201
MOV A,M
ANI 01H
JNZ skip
MOV B,A
INR
MOV B,A
skip: DCR C
INX H
JNZ loop
MOV A,B
STA 2300H
HLT
```

6. Write ALP to sum of the series of only even numbers.

Soln

```
LDA 4500H
MOV C,A
MVI B,00H
LXI H 4501
Loop: MOV A,M
ANI 01
JNZ skip
MOV A,B
ADD M
MOV B,A
```

If we use JZ skip instead of JNZ skip here then the program will be for odd numbers

For 16-bit numbers
LHLD → For loading data
SHLD → For storing data
DADD → For adding 16-bit numbers

Date.
Page No.

8. Write ALP to add two 16-bit numbers.

Soln

```
LHLD 4000H; Get first number
XCHG; The contents of register H (H-L pair) are
      exchanged with contents of register D (D-E pair).
LHLD 4002H; Get second 16-bit number in HL
DADD; Add DE and HL.
SHLD 4004H; Store the result.
HLT
```

9. Write ALP to multiply two 8-bit numbers.

Soln

```
LDA 2200H
MOV E,A
MVI D,00H
LDA 2201
MOV C,A
LXI H 0000H
back: DAD D;
      DCR C;
      JNZ back;
      SHLD 2202
      HLT
```

10. Divide a 16-bit number by 8-bit number.

Sample 2200H: 60H
2201H: A0H
2202H: 12H

Store quotient in memory location 2300H & 2301H & remainder in memory location 2302H & 2303H.

A060/12 = Quotient 08E8H, Remainder: 10H.

Date.
Page No.

```
skip: INX H
      DCR C
      JNZ Loop
      MOV A,B
      STA 4600
      HLT
```

7. Write ALP to find out largest number in a given set of data.

Soln

~~LXI H, 2050H~~
~~LXI D, 3396H~~
~~MOV A,M~~
~~CMP M~~

```
      LXI H, 8000H
      MOV C,M
LOOP: INX H
      MOV B,M
      DCR C
      INX H
      MOV A,M
      CMP B
      JC SKIP
      MOV B,A
SKIP: DCR C
      JNZ LOOP
      LXI H, 9000H
      MOV M,B
      HLT
```

Date.
Page No.

Soln

2300H	E8
2301H	08
2302H	10
2303H	00

```
      LHLD 2200H
      LDA 2202H
      MOV C,A
      LXI D,0000H
Back: MOV A,L
      SUB C
      MOV L,A
      JNC SKIP
      DCR H
SKIP: INX D
      MOV A,H
      CPI 00
      JNZ Back
      MOV A,L
      CMP C
      JNC Back
      SHLD 2302H
      XCHG
      SHLD 2300H
      HLT
```

Date.
Page No.

11. Write ALP for division of 16-bit numbers.

Soln

```
      LHLD 2200H
      LDA 2202H
      MOV C,A
      LXI D,0000H
Back: MOV A,L
      SUB C
      MOV L,A
      JNC SKIP
      DCR H
SKIP: INX D
      MOV A,H
      CPI,00
      JNZ Back
      MOV A,L
      CMP C
      JNC Back
      SHLD 2302H
      XCHG
      SHLD 2300H
      HLT
```

12. Find the largest number in a block of data. The length of block is in memory location 2200H and block itself starts from 2201H. Store the maximum number in memory location 2300H.

Soln

```
      LDA 2200H
      MOV C,A
      SUB A
```

Date.
Page No.

```
      LXI 2201H
      CMP M
      JNC SKIP   ← If we use JC SKIP instead the program will be for smallest number.
      MOV A,M
SKIP: INX H
      DCR
      JNZ Back
      STA 2300H
      HLT
```

13. Write ALP to sort given 10 numbers from memory location 2200H in ascending order.

Soln

```
      MVI B,09
START: LXI H,2200H
      MVI C,09
Back: MOV A,M
      INX H
      CMP M
      JC SKIP   } ← For descending we replace these
      JZ SKIP   }   two by JNC SKIP.
      MOV D,M
      MOV M,A
      DCX H
      MOV M,D
      INX H
SKIP: DCR C
      JNZ Back
      DCR B
      JNZ START
      HLT
```

RAL → Left rotation
RAR → Right rotation
RRC → Rotate right with carry flag.
RLC → Rotate left with carry flag.

14. Write ALP for Fibonacci series.

Soln

```
        MVI D, COUNT
        MVI B, 00H
        MVI C, 01H
        MOV A, B
Back:   ADD C
        MOV B, C
        MOV C, A
        DCR D
        JNZ Back
        HLT
```

15. Two digits BCD number is stored in memory location 4200H. Unpack the number and store the digits in memory location 4300H and 4301H.

Soln

```
        LDA 4200H
        ANI F0H
        RRC
        RRC
        RRC
        RRC
        STA 4301H
        LDA 4200H
        ANI 0FH
        STA 4300H
```

Date.
Page No.

Same program for packing is as follows:

```
LDA 4201H
RRC
RRC
RRC
RRC
MOV C,A
LDA 4200H
ADD C
STA 4300H
```

16. Write ALP to seperate even numbers from the given list of 50 numbers and store them in another list starting from 2300H. Assume starting address of 50 number list is 2200H.

Soln

```
      LXI H, 2200H
      LXI D, 2300H
      MVI C, 32H
START: MOV, A, M
       ANI 01
       JNZ SKIP
       MOV A,M   ← Since ANI 01 changes content of A so we repat MOV A,M
       STAX D
       INX D
SKIP:  INX H
       DCR C
       JNZ START
       HLT
```

Date.
Page No.

17. Find number of negative, zero and positive numbers from the list of 100 numbers.

Solⁿ

```
        LXI H, 2200H
        LXI D, 2300H
        MVI E, 64H      Since 100 in Hexadecimal is 64.
        MVI B, 00
        MVI C, 00
        MVI D, 00
START:  MOV A,M
        CPI 00          ; Compare that number with 00.
        JZ zero
        MOV A,M
        ANI 80H
        JZ positive
        INR D
~~zero: INR B~~
        JMP continue    ← unconditional jump with label continue.
zero:   INR B
        JMP continue
positive: INR C
continue: INX H
        DCR E
        JNZ START
        HLT
```

Date.
Page No.

Assembler, Linker & Preprocessor directives:

Assembler: An assembler is a program that converts assembly language into machine code. It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor. Assemblers are similar to compilers in that sense that they produce executable code. However, assemblers are more simplistic since they only convert low-level code to machine code.

Linker: A linker or link editor is a computer program that takes one or more object files generated by a compiler and combines them into one executable program. Computer programs are usually made up of multiple modules that span seperate object files, each being a compiled computer program. The linker combines these seperate files into a single resolving the symbolic references as it goes along.

Preprocessor directives: Preprocessor directives are lines included in a program that begin with the character #, which make them different from a typical source code text. They are invoked by the compiler to process some programs before compilation. Preprocessor directives change the text of the source code and the result is a new source code without these directives.

B. Programming with Intel 8086 microprocessor:

(*) Macro Assemblers (Microcontrollers): (Not more imp) only def^n

A macro assembler is able to generate a program segment, which is defined by a macro, when the name of the macro appears as an opcode in a program. It is an assembler that can perform macro substitution and expansion. The programmer can define a macro name later in the program, thus avoiding having to rewrite the statements. The macro assembler is still capable of regular assembler functioning, generating a machine instruction for each line of assembly language code; but like a compiler it can generate many machine instructions from one line of source code.

(*) Assembling and Linking:

An assembler program is used to translate the assembly language mnemonics for instruction to the corresponding binary codes. Assembling means bringing together or gather into one place.

Linking is done with the help of Linker. A linker is a program used to join several files into one large .obj file. It produces .exe file so that the program becomes executable.

Date.
Page No.

Assembler Directives, Comments:

Assembler directive is a statement to give direction to the assembler to perform task of the assembly process. It does following things:-

- Control the organization of program.
- Provide necessary information to the assembler to understand ALP's and generate necessary machine codes.
- Indicates how an operand or a section of the program is to be processed by the assembler.

Data declaration directives (Data Types)

Definition	Directive	Character string
Byte	DB	e.g. DB "Hello World"
Word	DW	
Double word	DD	Numeric constant
Far word	DF	eg. int a = 'a';
Quad word	DQ	#BINARY Val1 DB 10101B
Ten Bytes	DT	

Assume directive → The assume directive is used to tell the assembler that the name of the logical segment should be used for a specified segment. The 8086 works directly with only 4 physical segments: a code segment, a data segment, a stack segment and an extra segment.

Comments → A comment is a section of regular text that the assembler ignores when turning the assembly code into the machine code. It is very helpful to use some comments explaining what is going on when writing code. In assembly comments are usually denoted or written after semicolon ";".

Date.
Page No.

⊛ INT 21H Functions: (DOS services)

i) 01H → This function number is used for keyboard input with echo. This operation accepts a character from the keyboard buffer. If none is present, waits for keyboard entry. It returns the character in AL.

e.g MOV AH, 01H
int 21H

ii) 02H → This function number is used to display character. It send the character in DL to the standard output device console.

e.g. MOV AH, 02H
int 21H.

iii) 09H → This function number is used for string output. It send a string of characters to the standard output. DX ~~the~~ contains the offset address of string. The string must be terminated with a '$' sign.

e.g. MOV AH, 09H
int 21H.

iv) 0AH → This function number is used for string input.

v) 4CH → This function number is used to terminate the current program.

e.g. MOV AX, 4C00H
int 21H.

—— denotes instruction

Date.
Page No.

INT 10H Functions:

- Video display services (BIOS services)

Function number		Description
00H	→	Set video mode.
01H	→	Set cursor size.
02H	→	Set cursor position.
06H	→	Scroll window up.
07H	→	Scroll window down
08H	→	Read character and attribute at cursor.
09H	→	Display character and attribute at cursor.
0AH	→	Display character at cursor.

INSTRUCTIONS:-

1) Data transfer instructions:

i) MOV destination, source → Used to copy the byte or word from the provided source to the provided destination.

ii) PUSH source → Used to put a word at the top of the stack.

iii) XCHG destination, source → Used to exchange data from two locations.

iv) IN AX, I/O Port address → Used to read a byte or word from the provided port to the accumulator.

v) OUT Port address, AX → Used to send out a byte or word from the accumulator to the provided port.

vi) LEA Register, source → Used to load effective address of operand into specified register.

Date.
Page No.

2) Arithmetic instructions:

i) ADD destination, source → Used to add the byte/word provided.

ii) ADC destination, source → Used to add with carry.

iii) INC register → Used to increament the provided byte/word by 1.

iv) AAA → Used to adjust ASCII after addition.

v) DAA → Used to adjust the decimal after the addition / subtraction operation.

vi) SUB destination, source → Used to subtract the byte/word provided.

vii) SBB destination, source → Used to perform subtraction with borrow.

viii) DEC Register → Used to decreament provided byte/word by 1.

ix) CMP destination, source → Used to compare two provided byte/word.

x) MUL source → Source is multiplied by content of AL and result is stored in AX.

xi) DIV source → Divide the content of AX by source & store the result in AL.

3) Bit Manupulation Instructions:

i) NOT destination

ii) AND destination, source

iii) OR destination, source

iv) XOR destination, source.

4) String Instructions:

i) REP → Repeat instruction until CX=0.

ii) REPE/REPZ → Repeat instruction if Equal/zero.

(Upto 5 are main instructions 6, 7 & 8 are additional

Date.
Page No.

5) Program Execution Transfer Instructions:
(Branch and Loop Instructions)

Sequential instructions
- i) CALL → Call a sub program / procedure.
- ii) RET → Returns to the calling program.
- iii) JMP → Jump to particular label.

Branching instructions
- iv) JC → Jump if carry flag = 1.
- v) JE/JZ → Jump if Equal / Jump if ZF = 1.
- vi) JNC → Jump if CF = 0.
- vii) JNE/JNZ → Jump if Not equal / Jump if ZF = 0.
- viii) JP/JPE → Jump if parity / Jump if parity even (PF = 1).
- ix) JS → Jump if SF = 1.

(Iteration control state)
looping instructions
- x) LOOP → Loop through sequence of instruction until CX = 0.
- xi) LOOPE/LOOPZ → Loop through sequence of instruction until ZF = 1 and CX != 0.
- xii) LOOPNE/LOOPNZ → Loop through sequence of instruction until ZF = 0 and CX != 0.

Interrupt instructions
- xiii) INT → Interrupt generated by various I/O operations.
- xiv) INTO → Interrupt only if OF = 1.
- xv) IRET → Return from interrupt procedure to main calling program.

6) Rotate instructions:

- i) ROL → Rotate bits of byte MSB to LSB and C.F.
- ii) ROR → Rotate bits of byte LSB to MSB and C.F
- iii) RCL → Rotate bits of byte MSB to CF and CF to LSB.
- iv) RCR → Rotate bits of byte C.F to MSB and LSB to CF.

Date.
Page No.

7) Shift instructions:

i) SHL/SAL destination, count → Shift bits of word to left, put zero (0) in LSB.

ii) SHR/SAR destination, count → Shift bits of word to right put zero(0) in MSB.

8) Processor Control instructions:

i) STC → Set carry flag (CF) to 1.

ii) CLC → Clear CF i.e, CF=0.

iii) NOP → Not action except fetch & decode

iv) HLT → Halt (do nothing) until interrupt or reset.

v) WAIT → Wait until signal on the TESTP is in low.

⊛ 8086 microprocessor important (Exam point of view) programs:

1. Sample program to display "Hello World" in 8086.

Sol^n

```
.MODEL SMALL
.STACK 100
.DATA
    STRING DB 'Hello World'$
.CODE
    MAIN PROC
        MOV AX,@DATA
        MOV DX, OFFSET STRING
        MOV AH,09H
        INT21H
        MOV AX, 4C 00H
        INT 21H
    MAIN ENDP
    END MAIN
```

denote size of stack

terminates the string

Address of data moved to accumulator

AH 09 H will display string until $

DOS interrupt function

End procedure

End main programe

Date.
Page No.

2. Write ALP to reverse the input string for 8086.

Soln

```
.MODEL SMALL
.STACK 100
.DATA
.CODE
    MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV CX, 0
Read_character: MOV AH, 01H
    INT 21H
    CMP AL, 0DH
    JE END_OF_LINE
    PUSH AX
    INC CX
    JMP Read_character
END_OF_LINE: POP DX
    MOV AH, 02H
    INT 21H
    DEC CX
    LOOP END_OF_LINE (until CX=0)
    MOV AX, 4C 00H
    INT 21H
    MAIN ENDP
    END MAIN
```

Date.
Page No.

3. Write ALP to find factorial of number for 8086.

Sol^n

```
        MOV AX,05H
        MOV CX, AX
Back:   DEC CX
        MUL CX
        LOOP Back
        ; result stored in AX
        ; to store the result at D000H.
        MOV [D000], AX
        HLT
```

4. Write ALP to print sum of of two 8-bit number.

Sol^n

```
.MODEL SMALL
.STACK 100
.DATA
    Num1 DB EF
    Num2 DB AD
    MSG DB "Sum of two 8-bit numbers"$
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, 0
    MOV AL, Num1.
    ADD AL, Num2
    AAM; convert binary number to unpacked BCD
    ADD AX, 3030 ; to obtain ASCII code of
                   number.
    PUSH AX
    LEA DX, MSG
    MOV AH, 09H
    INT 21H
```

AX for 16-bit AL for 8-bit

Date.
Page No.

```
POP AX
MOV DL, AH
MOV DH, AL
MOV AH, 02H
INT 21H
MOV DL, DH
MOV AH, 02H
INT 21H
MOV AX, 4C00H
INT 21H
        MAIN ENDP
        END MAIN.
```

5. ALP to find length of string on 8086.

Soln

```
CODE SEGMENT
MOV AX, DATA
MOV DS, AX
MOV AL, '$'
MOV CX, 00H
MOV SI, OFFSET STR1
BACK: CMP AL, [SI]
JE GO
INC CL
INC SI
JMP BACK
GO: MOV LENGTH, CL
HLT
CODE ENDS
DATA SEGMENT
STR1 DB 'STUDENT BOX OFFICE $'
LENGTH DB?
DATA ENDS
END
```

Date.
Page No.

6. Find number of times letter 'e' exist in the string 'exercise', store the count at memory.

Solⁿ

```
.MODEL SMALL
.STACK 100H
.DATA
        STRING DB 'exercise', $
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV AL, 00H
    MOV SI, OFFSET STRING
    MOV CX, LENGTH
Back: MOV BH, [SI]
    CMP BH, 'e'
    JNZ Label
    INC AL
Label: INC SI
    LOOP Back
    MOV ANS, AL
    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
```

Date.
Page No.

7. Write an ALP to generate square wave with period of 200 μs and address output device 55H for 8086 microprocessor.

Soln

```
START: MOV AX,01H
       OUT 30H,AX
; to generate loop for 200μs using system frequency 5MHz.
       MOV BX,Count ; 7T
Label: DEC BX       ; 4T
       JNZ Label    ; 10T/7T
       MOV AX,00H
~~Label1: DEC BX~~ OUT 30H, AX
       MOV BX, Count
Label1: DEC BX
       JNZ Label1
       JMP START
```

8. Write an assembly language program to count number of vowels in a given string.

Soln

```
.MODEL SMALL
.STACK 100H
.CODE
  MAIN PROC
      MOV AX, @DATA
      MOV DS, AX
      MOV SI, OFFSET STRING ; initialize P
      MOV CX, LEN           ; length in CX register
      MOV BL,00             ; vowel count=0
Back: MOV AL[SI]
      CMP. AL, 'a'.
      JB VOW
```

Date.
Page No.

```
    CMP AL,'z'  ; Convert the character to upper case.
    JA VOW
    SUB AL,20H
VOW: CMP AL,'A'
     JNZ a3
     INC BL
     JMP a2
a3: CMP AL,'E'
     JNZ a4
     INC BL
     JMP a2
a4: CMP AL,'I'
     JNZ a5
     INC BL
     JMP a2
a5: CMP AL,'O'
     JNZ a6
     INC BL
     JMP a2
a6: CMP AL,'U'
     JNZ a2
     INC BL
a2: INC SI
     LOOP Back
     MOV Vowel,BL
     MOV AX,4C00H
     INT 21H
     MAIN ENDP
     .DATA
     String db 'The quick brown fox jumped over Lazy sleeping dog',$
     len dw $-string.
     Vowel db?
     END MAIN.
```