Mahatma Gandhi Mission's College of Engineering and Technology
Kamothe,Navi Mumbai

Department of Computer Engineering

Course Code: CSC405 **Course Name: Microprocessor**

Assignment No. 2

Class: SE **AY: 2021-22**

Q. No	Question
Q1.Justify Your Answers	
	(a) The instruction that is used as prefix to an instruction to execute it repeatedly until the CX register becomes zero is REP.
	(b) The expansion of DAS is Decimal Adjust after Subtraction.
	(c) AAD is the instruction in which adjustment is made before performing the operation.
	(d) NOP instruction introduces delay.
	(e) The instruction that unconditionally transfers the control of execution to the specified address is IMP.
Q2. Choose Correct Options	
	(a) Which of the following is not a data copy/transfer instruction? a) MOVb) PUSHc) DASd) POP
	(b) Which of the following instruction is not valid? a) MOV AX, BX b) MOV DS, 5000H c) MOV AX, 5000Hd) PUSH AX
	(c) The instruction that loads effective address formed by destination operand into the specified source register is a) LEA b) LDS c) LES d) LAHF
	(d) The instruction, CMP to compare source and destination operands it performs a) addition b) subtraction c) division d) multiplication
	(e) In the RCL instruction, the contents of the destination operand undergo function as a) carry flag is pushed into LSB & MSB is pushed into the carry flag b) carry flag is pushed into MSB & LSB is pushed into the carry flag c) auxiliary flag is pushed into LSB & MSB is pushed into the carry flag d) parity flag is pushed into MSB & LSB is pushed into the carry flag
Q3. state whether the following statements are true or false (Give Reasons)	
	(a) The LAHF instruction that loads the AH register with the lower byte of the flag register. (True/False)
	(b) The instruction that pushes the flag register on to the stack is PUSHF(True/False)
	(c) ADC instruction that supports addition when carry exists. (True/False)
Q4. Name the following or define or design the following	
	(a) Define instruction SHAL,DAA ,TEST and NOP. Ans: SHAL- The shl or sal instruction is used to shift the bits of the operand destination to the left, by the numbe of bits specified in the current operand. Bits shifted beyond the destination are first shifted into CF flag. Zeros vacated position during the shift operator. DAA-The chemical adjust after addition instruction allows addition of numbers represented in 8 bit packed BCD code. It is used immediately after normal addition instruction operating on BCD codes. TEST- The TEST instruction performs a bitwise AND two operands.

The flag SF, ZF, PF are modified while the result of the AND is discarded.
The OF and CF flags are set to O, while AF flag is undefined.
NOP-NOP stands for NO operation. NOP is an instruction which is falling under machine control instruction category. This instruction does nothing during execution.

(b) Name the instruction set in 8086 .

Ans: Instructions are classified on the basis of functions they perform. They are categorized into the following main types:

Data Transfer instruction

All the instructions which perform data movement come under this category. The source data may be a register, memory location, port etc. the destination may be a register, memory location or port. The following instructions come under this category:

Instruction	Description
MOV	Moves data from register to register, register to memory, memory to register, mer accumulator to memory, etc.
LDS	Loads a word from the specified memory locations into specified register. It also loads two memory locations into DS register.
LES	Loads a word from the specified memory locations into the specified register. It also lo two memory locations into ES register.
LEA	Loads offset address into the specified register.
LAHF	Loads low order 8-bits of the flag register into AH register.
SAHF	Stores the content of AH register into low order bits of the flags register.
XLAT/XLATB	Reads a byte from the lookup table.
XCHG	Exchanges the contents of the 16-bit or 8-bit specified register with the contents of register or memory locations.
PUSH	Pushes (sends, writes or moves) the content of a specified register or memory locati the stack.
POP	Pops (reads) two bytes from the top of the stack and keeps them in a specified location(s).
POPF	Pops (reads) two bytes from the top of the stack and keeps them in the flag register.

IN	Transfers data from a port to the accumulator or AX, DX or AL register.
OUT	Transfers data from accumulator or AL or AX register to an I/O port identified by the instruction.

Arithmetic Instructions

Instructions of this group perform addition, subtraction, multiplication, division, increment, decrement, comparison, ASCII and decimal adjustment etc.

The following instructions come under this category:

Instruction	Description
ADD	Adds data to the accumulator i.e. AL or AX register or memory locations.
ADC	Adds specified operands and the carry status (i.e. carry of the previous stage).
SUB	Subtract immediate data from accumulator, memory or register.
SBB	Subtract immediate data with borrow from accumulator, memory or register.
MUL	Unsigned 8-bit or 16-bit multiplication.
IMUL	Signed 8-bit or 16-bit multiplication.
DIV	Unsigned 8-bit or 16-bit division.
IDIV	Signed 8-bit or 16-bit division.
INC	Increment Register or memory by 1.
DEC	Decrement register or memory by 1.
DAA	**Decimal Adjust after BCD Addition:** When two BCD numbers are added, the DAA ADC instruction to get correct answer in BCD.
DAS	**Decimal Adjust after BCD Subtraction:** When two BCD numbers are added, the DAS SBB instruction to get correct answer in BCD.
AAA	**ASCII Adjust for Addition:** When ASCII codes of two decimal digits are added, th addition to get correct answer in unpacked BCD.

AAD	**Adjust AX Register for Division:** It converts two unpacked BCD digits in AX to th number. This adjustment is done before dividing two unpacked BCD digits in AX by an u
AAM	**Adjust result of BCD Multiplication:** This instruction is used after the multiplication of t
AAS	**ASCII Adjust for Subtraction:** This instruction is used to get the correct result in unpa subtraction of the ASCII code of a number from ASCII code another number.
CBW	Convert signed Byte to signed Word.
CWD	Convert signed Word to signed Doubleword.
NEG	Obtains 2's complement (i.e. negative) of the content of an 8-bit or 16-bit specified location(s).
CMP	Compare Immediate data, register or memory with accumulator, register or memory loca

Logical Instructions

Instruction of this group perform logical AND, OR, XOR, NOT and TEST operations. **The following instructions come under this category:**

Instruction	Description
AND	Performs bit by bit logical AND operation of two operands and places the result in the
OR	Performs bit by bit logical OR operation of two operands and places the result in the sp
XOR	Performs bit by bit logical XOR operation of two operands and places the result in the
NOT	Takes one's complement of the content of a specified register or memory location(s).
TEST	Perform logical AND operation of a specified operand with another specified operand.

Rotate Instructions

The following instructions come under this category:

Instruction	Description
RCL	Rotate all bits of the operand left by specified number of bits through carry flag.
RCR	Rotate all bits of the operand right by specified number of bits through carry flag.

ROL	Rotate all bits of the operand left by specified number of bits.
ROR	Rotate all bits of the operand right by specified number of bits.

Shift Instructions

The following instructions come under this category:

Instruction	Description
SAL or SHL	Shifts each bit of operand left by specified number of bits and put zero in LSB positic
SAR	Shift each bit of any operand right by specified number of bits. Copy old MSB into ne
SHR	Shift each bit of operand right by specified number of bits and put zero in MSB posit

Branch Instructions

It is also called program execution transfer instruction. Instructions of this group transfer program execution from the normal sequence of instructions to the specified destination or target. The following instructions come under this category:

Instruction	Description
JA or JNBE	Jump if above, not below, or equal i.e. when CF and ZF = 0
JAE/JNB/JNC	Jump if above, not below, equal or no carry i.e. when CF = 0
JB/JNAE/JC	Jump if below, not above, equal or carry i.e. when CF = 0
JBE/JNA	Jump if below, not above, or equal i.e. when CF and ZF = 1
JCXZ	Jump if CX register = 0
JE/JZ	Jump if zero or equal i.e. when ZF = 1
JG/JNLE	Jump if greater, not less or equal i.e. when ZF = 0 and CF = OF
JGE/JNL	Jump if greater, not less or equal i.e. when SF = OF
JL/JNGE	Jump if less, not greater than or equal i.e. when SF ≠ OF
JLE/JNG	Jump if less, equal or not greater i.e. when ZF = 1 and SF ≠ OF

JMP	Causes the program execution to jump unconditionally to the memory address o instruction.
CALL	Calls a procedure whose address is given in the instruction and saves their return ad
RET	Returns program execution from a procedure (subroutine) to the next instruction o
IRET	Returns program execution from an interrupt service procedure (subroutine) to the
INT	Used to generate software interrupt at the desired point in a program.
INTO	Software interrupts to indicate overflow after arithmetic operation.
LOOP	Jump to defined label until CX = 0.
LOOPZ/LOOPE	Decrement CX register and jump if CX ≠ 0 and ZF = 1.
LOOPNZ/LOOPNE	Decrement CX register and jump if CX ≠ 0 and ZF = 0.

Here, CF = Carry Flag
ZF = Zero Flag
OF = Overflow Flag
SF = Sign Flag
CX = Register

Flag Manipulation and Processor Control Instructions

Instructions of this instruction set are related to flag manipulation and machine control. The following instructions come under this category:

Instruction	Description
CLC	**Clear Carry Flag:** This instruction resets the carry flag CF to 0.
CLD	**Clear Direction Flag:** This instruction resets the direction flag DF to 0.
CLI	**Clear Interrupt Flag:** This instruction resets the interrupt flag IF to 0.
CMC	This instruction take complement of carry flag CF.
STC	Set carry flag CF to 1.
STD	Set direction flag to 1.

STI	Set interrupt flag IF to 1.
HLT	Halt processing. It stops program execution.
NOP	Performs no operation.
ESC	**Escape:** makes bus free for external master like a coprocessor or peripheral device.
WAIT	When WAIT instruction is executed, the processor enters an idle state in which the processing.
LOCK	It is a prefix instruction. It makes the LOCK pin low till the execution of the next instructio

String Instructions

String is series of bytes or series of words stored in sequential memory locations. The 8086 provides some instructions which handle string operations such as string movement, comparison, scan, load and store.

The following instructions come under this category:

Instruction	Description
MOVS/MOVSB/MOVSW	Moves 8-bit or 16-bit data from the memory location(s) addressed by SI re location addressed by DI register.
CMPS/CMPSB/CMPSW	Compares the content of memory location addressed by DI register with th location addressed by SI register.
SCAS/SCASB/SCASW	Compares the content of accumulator with the content of memory locat register in the extra segment ES.
LODS/LODSB/LODSW	Loads 8-bit or 16-bit data from memory location addressed by SI register in
STOS/STOSB/STOSW	Stores 8-bit or 16-bit data from AL or AX register in the memory locat register.
REP	Repeats the given instruction until CX ≠ 0
REPE/ REPZ	Repeats the given instruction till CX ≠ 0 and ZF = 1
REPNE/REPNZ	Repeats the given instruction till CX ≠ 0 and ZF = 0

(c) How to find whether given word is palindrome or not in assembly language?

Ans: `DATA SEGMENT`

```
BLOCK1 DB 'MALAYALAM'
MSG1 DB "IT IS PALINDROME $"
MSG2 DB "IT IS NOT PALINDROME $"
PAL DB 00H

DATA ENDS

PRINT MACRO MSG

MOV AH,09H
LEA DX,MSG
INT 21H
INT 3H

ENDM

EXTRA SEGMENT

BLOCK2 DB 9 DUP(?)
EXTRA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA,ES:EXTRA
START: MOV AX,DATA
MOV DS,AX
MOV AX,EXTRA
MOV ES,AX
LEA SI,BLOCK1
LEA DI,BLOCK2+8
MOV CX,00009H
BACK: CLD
LODSB
STD
STOSB
LOOP BACK
LEA SI,BLOCK1
LEA DI,BLOCK2
MOV CX,0009H
CLD
REPZ CMPSB
JNZ SKIP
PRINT MSG1
SKIP: PRINT MSG2
CODE ENDS
END START
```

Q5. Answer the following questions in brief (20 to 30 words)

(a) Justify the source and destination in an instruction both cannot be memory location.

Ans:

- Both the source and the destination operands cannot be memory locations except for string instructions.
- The actual current stack top is always occupied by the previously pushed data.

- So, the push operation decrements SP by 2 and then stores the two bytes contents of the operand data onto the stack.

(b) Try to find out the physical address for instruction LDS BX, Count .(Assume count=2045H,THE CONTENT OF DS=2314 H,CS=2105 H, IP=187A H AND ES=2010 H.)

Ans: LDS BX, count
Physical Address?
DS=2314H
CS=2105H
IP=187AH
ES=2010H
Count=2045H
LDS-> Data transfer instructionLDS, Des, Src
It loads 82 bit pointer from memory source to destination register and DS LDS BX, [2045H]
Segment Add->2314->0010 0011 0001 0100
Offset Add->187A->0001 1000 0111 1010
Segment address shifted by 4-bit position:-
0010 0011 0001 0100 0000
\+ 0001 1000 0111 1010
0010 0100 1001 1011 1010
---
2 4 9 B A
Physical Address->(249BA)

(c) Justify with your answer PUSH cannot be used to push immediate data onto the stack.
Ans: In 8086 assembly language programming, we can only loads data into a segment register and then we have to move it from this general register to the segment register.

. Q6. Answer the following questions in brief (50 to 70 words)

(a) Explain in assembler directives of 8086?

Ans: An assembler directive is a **statement to give direction to the assembler to perform task of the assembly process**. It control the organization if the program and provide necessary information to the assembler to understand the assembly language programs to generate necessary machine codes. Assembler directives **supply data to the program and control the assembly process**. Assembler directives enable you to do the following: Assemble code and data into specified sections. Reserve space in memory for uninitialized variables.

(b) Explain the Addressing modes of 8086 with suitable diagram .

Ans: The way of specifying data to be operated by an instruction is known as **addressing modes**. This specifies that the given data is an immediate data or an address. It also specifies whether the given operand is register or register pair.
Types of addressing modes:

1. **Register mode –** In this type of addressing mode both the operands are registers. Example:

```
2. MOV AX, BX

3. XOR AX, DX

   ADD AL, BL
```

4. **Immediate mode –** In this type of addressing mode the source operand is a 8 bit or 16 bit data. Destination operand can never be immediate data. Example:

```
MOV AX, 2000

MOV CL, 0A

ADD AL, 45

AND AX, 0000
```

Note that to initialize the value of segment register an register is required.

```
MOV AX, 2000

MOV CS, AX
```

8. **Displacement or direct mode –** In this type of addressing mode the effective address is directly given in the instruction as displacement.
   Example:

```
MOV AX, [DISP]

MOV AX, [0500]
```

10. **Register indirect mode –** In this addressing mode the effective address is in SI, DI or BX.
    Example: Physical Address = Segment Address + Effective Address

```
    MOV AX, [DI]

    ADD AL, [BX]

MOV AX, [SI]
```

13. **Based indexed mode –** In this the effective address is sum of base register and index register.

```
    Base register: BX, BP

Index register: SI, DI
```

The physical memory address is calculated according to the base register.
Example:

```
MOV AL, [BP+SI]

MOV AX, [BX+DI]
```

15. **Indexed mode –** In this type of addressing mode the effective address is sum of index register and displacement.
    Example:

```
    MOV AX, [SI+2000]

MOV AL, [DI+3000]
```

17. **Based mode –** In this the effective address is the sum of base register and displacement.
    Example:

```
MOV AL, [BP+ 0100]
```

18. **Based indexed displacement mode –** In this type of addressing mode the effective address is the sum of index register, base register and displacement.

Example:

```
MOV AL, [SI+BP+2000]
```

19. **String mode –** This addressing mode is related to string instructions. In this the value of SI and DI are auto incremented and decremented depending upon the value of directional flag.
    Example:

```
20.     MOVS B

        MOVS W
```

21. **Input/Output mode –** This addressing mode is related with input output operations.
    Example:

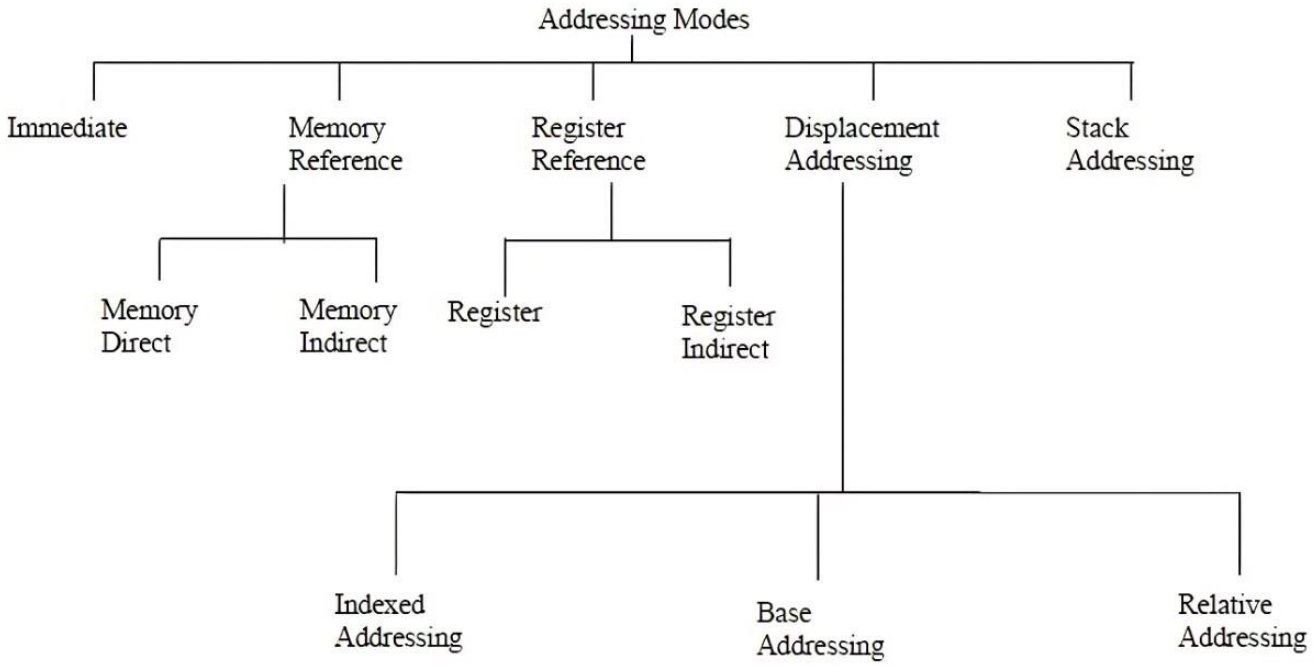
```
22.     IN A, 45

        OUT A, 50
```

23. **Relative mode –**
    In this the effective address is calculated with reference to instruction pointer.
    Example:

```
24.     JNZ 8 bit address

        IP=IP+8 bit address
```



Common Addressing Modes

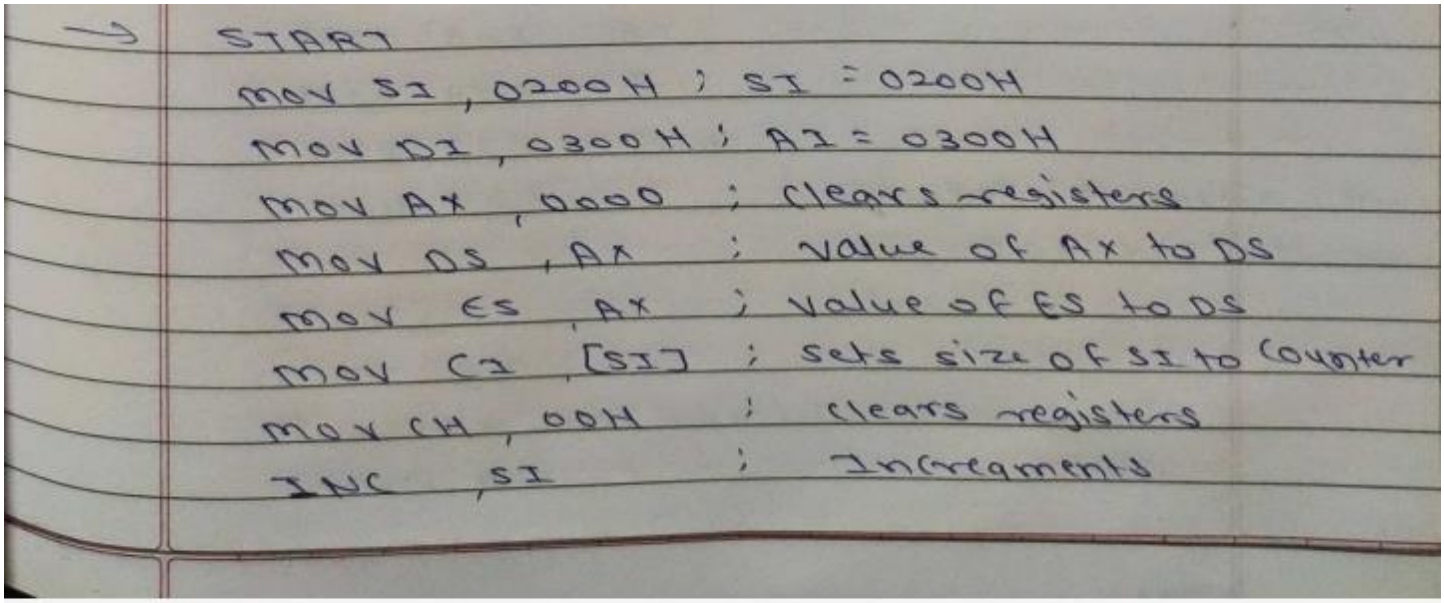
(c) Explain how to compute the address of memory operand for 8086:i) MOV AX,[BX] ii) MOV AL,[BP+SI](assume CS: 0100H,DS:0200H,SS:0400H,ES:0030H,BP:0010H,BX:0020,SI:0030H,SP=0040H). Clearly Show Computations.
Ans:

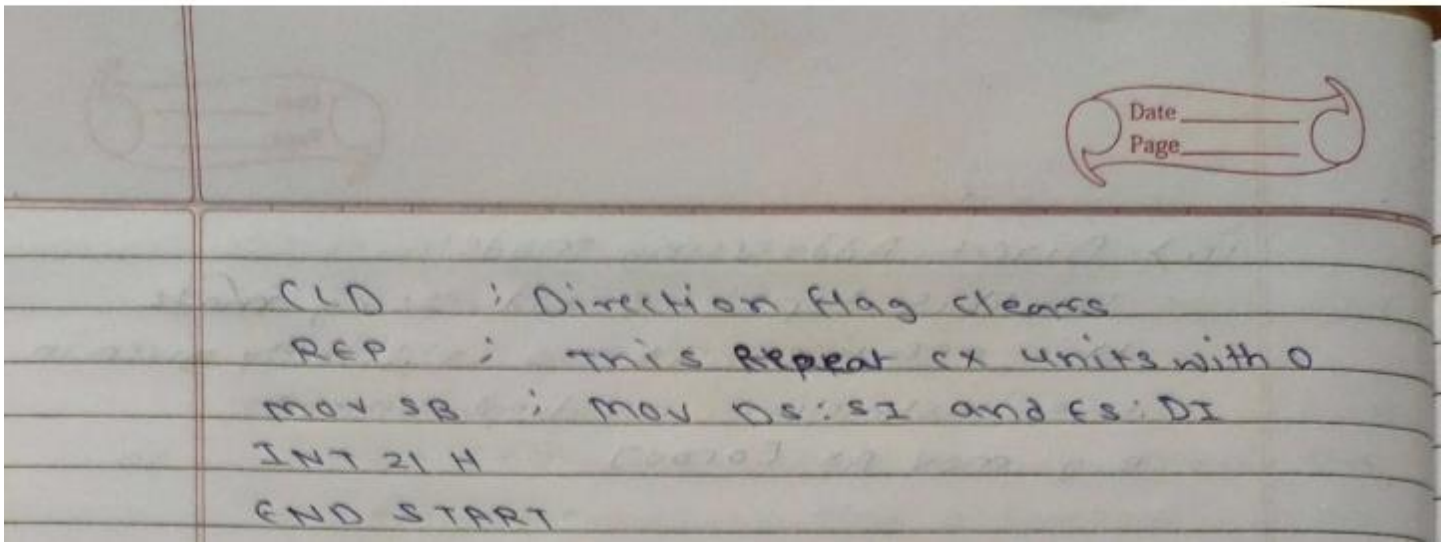
Q7. Think and Answer

(a) Write assembly language program for exchange the blocks of 1KB located at 0200H and 0300H using

string instruction.

Ans:





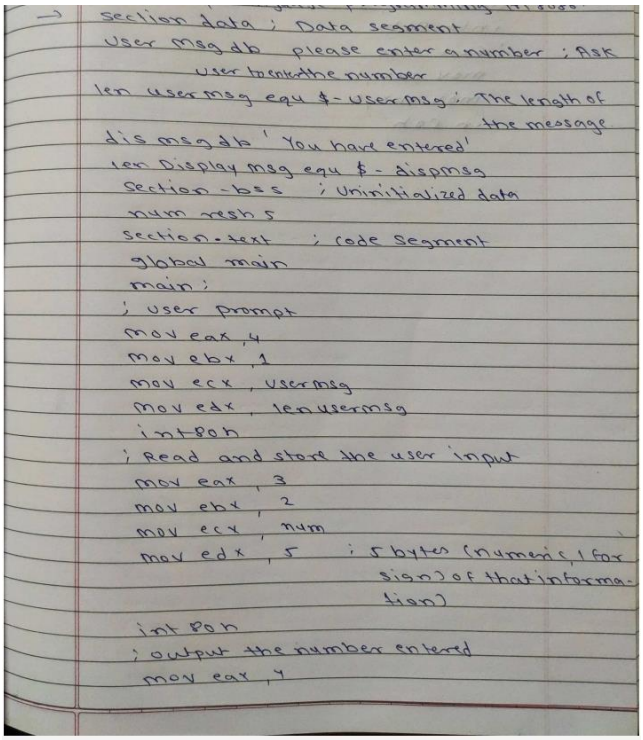
(b) Justify with any real time application of assembly language.

Ans: 1. Assembly Language is used when speed and reliability are the overriding factor like small footprint real-time operating systems. 2. By using assembly language, programmers can maximize on speed to a level. It is easy to write than machine code programs. 3. It allows the programmer access to registers or instructions that are not usually provided by a High-level language. 4. The main Application of Assembly Language is for direct hardware manipulation i.e. device drivers. 5. Assembly language also directly correlates which machine instructions; the only way to get closer to the machine is to write in binary or hex code.

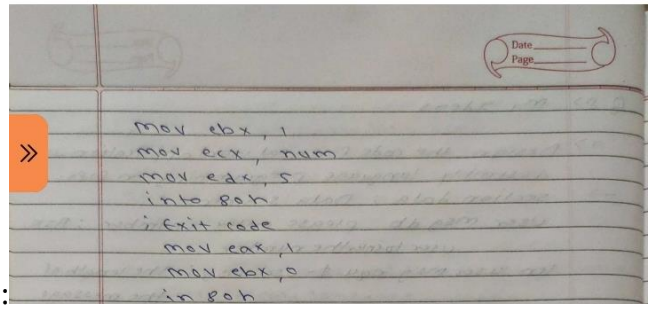
Q8. **My Ideas**

(a) Design the code for real world application using assembly language programming in 8086.

Ans:



```
section .data ; Data segment
User msg db please enter a number ; Ask
        user to enter the number
len usermsg equ $-usermsg ; The length of
                            the message
dis msg db 'You have entered'
len Display msg equ $ - dispmsg
Section .bss ; Uninitialized data
num resb 5
section .text ; code Segment
global main
main:
; user prompt
mov eax,4
mov ebx,1
mov ecx, usermsg
mov edx, len usermsg
int 80h
; Read and store the user input
mov eax, 3
mov ebx, 2
mov ecx, num
mov edx, 5 ; 5 bytes (numeric, 1 for
                sign) of that informa-
                tion)
int 80h
; output the number entered
mov eax, 4
```



```
mov ebx, 1
mov ecx, num
mov edx, 5
int 80h
; Exit code
mov eax, 1
mov ebx, 0
int 80h
```