B.M.S INSTITUTE OF TECHNOLOGY & MANAGEMENT

Department of Computer Science & Engineering

LAB MANUAL

MICROPROCESSOR - SOFTWARE PART (8086)

Sub Code: 15CSL48

4th Semester CSE

Prepared by:

Shankar. R
Assistant Professor
CSE, BMSIT&M

Reviewed By:

Dr. G Thippeswamy
Professor & Head, CSE
BMSIT&M

Programs

1. Design and develop an assembly language program to search a key element “X” in a list of n 16-bit numbers. Adopt **Binary Search** algorithm in your program for searching.

2. Design and develop an assembly program to sort a given set of n 16-bit numbers in ascending order. Adopt **Bubble Sort** algorithm to sort given elements.

3. Develop an assembly language program to reverse a given string and verify whether it is a **Palindrome** or not. Display the appropriate message.

4. Develop an assembly language program to compute **nCr** using recursive procedure. Assume that n and r as non-negative integers.

5. Design and develop an assembly language program to read the current **Time and Date** from the system and display it in the standard format on the screen.

Some Facts

1. Microprocessor???????

A: Microprocessor is the CPU of microcomputer**.**

- It is a 16-bit Microprocessor(μp).It's ALU, internal registers works with 16bit binary word.
- 8086 has a 20 bit address bus can access up to 2^{20}= 1 MB memory locations.
- 8086 has a 16bit data bus. It can read or write data to a memory/port either 16bits or 8 bit at a time.
- It can support up to 64K I/O ports.
- It provides 14, 16 -bit registers.
- Frequency range of 8086 is 6-10 MHz
- It has multiplexed address and data bus AD0**-** AD15 and A16 – A19.
- It requires single phase clock with 33% duty cycle to provide internal timing.
- It can prefetch upto 6 instruction bytes from memory and queues them in order to speed up instruction execution.
- It requires +5V power supply.
- A 40 pin dual in line package.
- 8086 is designed to operate in two modes, Minimum mode and Maximum mode.

  The minimum mode is selected by applying logic 1 to the MN / MX# input pin. This is a single microprocessor configuration.

  The maximum mode is selected by applying logic 0 to the MN / MX# input pin. This is a multi micro processors configuration.

2. What are the components of micro computer?

A: CPU, memory, input and output circuitry.

3. What is IP?

A: It is an instruction pointer which contains the address of next instruction to be executed.

4. What are general purpose registers (GPR)?

A: They are temporary registers used to act upon data…AX,BX,CX,DX

5. What is memory?

A: It is a section usually consists of mixture of RAM and ROM. It may also have magnetic disks, hard disks or optical disks.

6. What is purpose of using memory?

A: a) is used to store binary codes for the sequence of instructions.
b) is to store the binary coded data with which the computer is going to be working.

7. What is bus? What are different types of buses?

A: It is a collection of wires. There are 3 types of buses, they are:
a) Address bus b) Control bus c) Data bus.
In general address bus consists of 16, 20, 24 or 32 parallel signals lines.
Data bus consists of 8, 16 or 32 parallel signal lines.
Control bus consists of 4 to 10 parallel signal lines.

8. What are the different registers present in 8086?

A: Different registers present in 8086 are:-
AX-Accumulator register.
BX- Base register.
CX- Counter register.
DX- Data register.

9. What are the different pointers and index registers in 8086?

A: The different pointers and index registers in 8086 are:-
SI-Source index registers.
DI-Destination index registers.
BP-Base pointer registers.
SP-Stack pointer registers.
IP-Instruction pointer registers.

10. What are the different segment registers?

A: The different segment registers are:-

CS register: - Code segment.
DS register: - Data segment.
ES register: - Extra segment.
SS register: - Stack segment.

11. What is micro controller?
A: It is the collection of microprocessor, RAM and ROM.

12. How many bits does 8086 contain?
A: It is of 16 bits.

13.Why it is only of 16 bits?
A: Because in 8086, ALU is composed of 16 bits.

14. What is Extra segment?
A: It is extended portion of data segment. It is used whenever we use strings.

15. What is syntax of MOV instruction?
A: MOV destination, source

Here, source can be any register, memory or immediate number. But destination can be register or memory but cannot be an immediate number.

16. What are the 2 major parts of 8086 architecture?
A**: BIU** → Bus interface unit
EU → Execution unit

17. What is the application of BIU?
A: **BIU** sends address, fetches instruction from memory,read data from parts & memory & writes data to parts & memory

OR

It handles all transfers of data & addresses on buses for the execution unit.

18. What is application of EU?

The EU of 8086 tells the BIU where to fetch instruction or data from, decodes instruction & executes instruction.

19. What are components of EU?

Control circuitary,instruction decode & ALU

This directs internal operations

Decoder: It translates instruction fetchedfrom memory into a series of actions which the EU Carries out.

ALU: It is 16-bit unit which can add, subtract,AND,OR,XOR,…etc or shifting binary numbers.

20) What is flag?

It is a flip flop that indicates some condition produced by the execution of instruction or controls certain operation of EU.

21) What is a flip flop?

It holds the binary data & holds any single value.

22) How many types of flags we have?

There are 9 types of flags. In which 6 are *conditional flags* & 3 are *control flags.*

CONDITIONAL FLAGS:

CF- Carry flag→ if the carry generates then 1 or else 0.
PF- Parity flag→ set if result has even parity
AF-Auxillary flag→ in BCD system
ZF-Zero flag→ set if result= 0
SF-Sign flag → MSB of result → when the condition produced is negative then it is 1
OF-Overflow flag→ If memory has overflow.

<u>CONTROL FLAGS:</u>

TF- Trap flag → used for single stepping through a program

IF- Interrupt flag → which is used to allow or prohibit the interruption of a program

DF-Direction flag → which is used with string instruction

23) What are different ALP development tool devices we have?

i. Editor
ii. Assembler
iii. Linker
iv. Loader
v. Debugger
vi. Emulator

24) What is emulator?

It is a mixture of software & hardware used to list the program whether program will work properly or not.

25) Brief about various Assembler Directives?

ASSUME: **-** Assume CS: Code, DS: DATA
DATA Segment
DATA Segment Ends

DB: - Define byte
Ex: s db “abc” ; It stores 3 bytes .
a db 6 ; (int a=6 in ‘C’)

END: **-** Logical end of a program.

EQU: Equate
Ex: a equ 6 ;(int a==6 in ‘C’)

EVEN: **-** It directs the assembler to increment the location count (IP) to the next en=ven address if its not already in even address.

EXTR: **-** EXTERNAL: It si used to tell the assembler that the names or labels following the directives are in some other assembly module.

26) Validate the following instructions

a. mov ds,3653h → *INVALID* → Because immediate data can’t be in segment register.

b. mov dx,3653h → VALID

c. mov al,bx → *INVALID* → operands should have same size

d. mov ds,es → *INVALID* → both operands can’t be segment register

e. mov IP,ax → *INVALID* → because destination can’t be IP address

f. mov [bx],[cx] → *INVALID*→ both operands can’t be memory locations

g. mov 0C4034h,bx → *INVALID* → destination can’t be immediate address

h. pop cs → *INVALID*

I. xchg [bx],[si] → *INVALID* → one of operands should be a region

j. xchg cs,bx → *INVALID* → Improper use of register

k. lea ds,56h[si] → *INVALID* → Can’t use ds in source & also destination can’t be a segment register

l. lea bx,si → *INVALID* → Illegal use of register. Always specify source of addressing mode.

m. lea dx,0C4034h → *INVALID* → Immediate data not allowed
lea dx,[0C4034h] → *VALID* → **EA=DS+0C4034h**

n. add ds,0C4034h → *INVALID* → Segment register are not used

o. Inc [si] → *INVALID* → Operands must have size

p. OR ds, 0C4034h → *INVALID* → improper use of segment register

27) Write a code for initializing data segment?

```
mov ax, @data
 mov ds,ax
```

28) What are branching instructions?

The statements that alter sequence of execution of the program are called branching instructions.

29) Write a code for the termination of the program?

```
mov ah,4ch
int 21h
```

30) Write a code for the initialization of es?

```
mov es,ax
```

31) What are the major differences between macro & procedure?

Sl. No	MACRO	PROCEDURE
1	Access using macro name during ALP (.i.e. conversion from low level language to machine level language by assembler).	Access using CALL & RET mechanism during execution.
2	Doesn’t use stack mechanism.	It uses stack mechanism.
3	Takes less time to execute since there is no transfer control.	Takes time to execute since control has to be transferred & from the procedure.
4	Machine code is generated for instruction each time the macro is called or invoked.	The machine code is generated for the procedure it placed in the memory only once.
5	Size of the execution is more.	Due to the reason specified earlier the size of the executed is less.
6	Parameters are passed as part of statement which calls macro.	Parameters can be passed using registers, memory or stack.

32) What is re-entrant procedure?

A portion of the code that can be called by a procedure while another is already executing is called re-entrant.

The procedure that contains executing code is called re-entrant procedure.

33) What is key pad interface?

The interface which has 8 rows and 3 columns. Rows are connected to 26 pin connector through a register to ground. Columns are directly connected to the 26 pin connector using data cable at the intersection of rows and columns of keyboard are provided.

34) What is polling effect?

In microprocessor it will be scanning 8x3 keypad each and every second until the input is given is called polling effect.

35) Key debouncing effect?

When a key is pressed the signals may be generated more than once as a microprocessor is fast processor it takes all the three signals to avoid these to take only one signal at a time we use call delay procedure.

36) Which are arithmetic instructions?

1)AAA 2)AAS 3)AAM 4)AAD 5)ADC

37) What are shift instructions?

SHR:- Shift Logical right operation by one bit (division by 2).

SHL:- Shift logical left operation (multiply by 2).

38) What are rotate instructions?

ROL:- Rotate by left by one bit.

Syntax: ROL destination,count

ROR:- Rotate by right by one bit.

39) Write the Fibonacci series?

0,1,1,2,3,5,8,13………n.

40) Write syntax for MUL?

Syntax: mul src

mul bl: - It multiply with al (by default) result is stored in ax.

mul bx:- Multiply with AX & result is stored in DX,AX. DX is M.S.W and AX is L.S.W.

41) Write a syntax of IN?

Take the input from logic controller.

Syntax: IN Accumulator,port

Copies data from a port to the accumulator register it can be done is 2 ways.

a) Fixed Port:- Here 8 bit address is specified directly in the instruction.
   Ex:- IN al,3536h
b) Variable Port:- Here port address is loaded into dx register before IN & port address ranges from 00-FFFFH
   Flags:- None of the flags.

42) What is CLD?

CLD:- Clear Direction Flag.
Syntax:- CLD
Operation:- It clear the direction flag DF=0 left to right.
SI & DI auto incremented by 1.

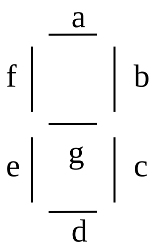
43) What is LED?

LED:- Light Emitting Diode.

44) Write a code for display a string in the program?

```
lea dx,string_name
mov ah,09h
int 21h
```

45) Write a symbolic notation of seven segment display?



46) What is DAC module?

This device is a bit converter that transforms 8 bit binary number into analog voltage.

47) Write the formula to calculate the Vout when angle is known?

Vout = [5v+5sinx]256/10

48) What is STD?

STD:- Set Direction Flag
Syntax:- STD
It sets the direction DF=1….SI & DI auto decremented by 1.

49) Write a code for time storage(access)?

```
mov ah,2ch
int 21h
```

50) What is Stepper Motor?
It is an output device or rotating device.

51) Function of stepper motor?
It rotates the motor both in clock wise & anti clockwise direction.

52) What is the sequence we use in half step mode?
11,22,44,88

53) Sequence of full step mode?
33,66,99

54) Write the interrupt for deleting a file?

```
mov ah,41h
mov dx,offset file name  or lea dx, file name
int 21h
```

55) Write the interrupt for creating a file?

```
mov ah,3ch
mov dx,offset file name or lea dx, file name
int 21h
```

56) Write a code to take a single character?

```
mov ah,01h
int 21h
```

57) Write a code to display a single character?

```
 mov dl,printable_character's_ASCII_value
mov ah,02h
 int 21h
```

58)What are the 8086 interrupt types?

Dedicated interrupts

Type 0: Divide by zero interrupt
Type 1: Single step interrupt or trap interrupt

Type 2:Non maskable interrupt
Type 3: Breakpoint
Type 4: Overflow interrupt

Software interrupts
Type 0-255

59. What is interrupt service routine?

Interrupt means to break the sequence of operation. While the CPU is executing a program an interrupt breaks the normal sequence of execution of instructions & diverts its execution to some other program. This program to which the control is transferred is called the interrupt service routine

60. What is microcontroller?

A device which contains the microprocessor with integrated peripherals like memory, serial ports, parallel ports, timer/counter, interrupt controller, data acquisition interfaces like ADC,DAC is called microcontroller.

ADDRESSING MODES:

- **Immediate addressing mode**: In this mode of addressing the data to be manipulated is part of the instruction.
  Ex: mov dx,4000h
  Immediate data should be in source field.And the destination can be register or memory location .

- **Register addressing mode**: In this mode,the data to be manipulated is present in register.
  Ex: mov bx,ax

- **Memory addressing mode**: In this mode,one operand is a memory location.There are 2 types of memory addressing mode:

- **Direct addressing mode**:In this mode,16-bit offset address of memory location is directly specified in the instruction.
  Ex: mov ax,[5000h]
- **Indirect addressing mode**: In this mode,the offset address of the data to be accessed is not present in the instruction,instead it is present in any register.
  Ex: mov ax,[bx]

This is base register indirect addressing mode,
Similarly,for base register indirect addressing mode with displacement;
Mov ax,23h[bx]
For indexed addressing mode,offset addressing will be present in si or di .
Ex: mov ax,[si]
With displacement,
mov ax,23h[si]
For base indexed addressing mode,the offset of the address will be stored in bx/bp and si/di.
Ex: mov ax,[bx+si]
With displacement,
Mov ax,10000h[bx+si]

- **Implied mode of addressing**: In this mode,the operands are not explicitly specified.
  Ex: DAA
  By default the result sio stored in ax.
- **I/O addressing mode**:
  1. Fixed port addressing → Ex: IN al,70h, OUT 70h,al
  2. Variable port addressing →Ex: IN al,dx, OUT dx,al

  In this mode, microprocessor si connected to I/O device & memory for communication.

Program memory addressing mode: The control jumps are example for this mode

Selected 8086 Instructions

The following is a brief summary of the 8086 instruction set:

Data Transfer Instructions

MOV	Move byte or word to register or memory
IN, OUT	Input byte or word from port, output word to port
LEA	Load effective address
LDS, LES	Load pointer using data segment, extra segment
PUSH, POP	Push word onto stack, pop word off stack
XCHG	Exchange byte or word
XLAT	Translate byte using look-up table

Logical Instructions

NOT	Logical NOT of byte or word (one's complement)
AND	Logical AND of byte or word
OR	Logical OR of byte or word
XOR	Logical exclusive-OR of byte or word
TEST	Test byte or word (AND without storing)

Shift and Rotate Instructions

SHL, SHR	Logical shift left, right byte or word by 1 or CL
SAL, SAR	Arithmetic shift left, right byte or word by 1 or CL
ROL, ROR	Rotate left, right byte or word by 1 or CL
RCL, RCR	Rotate left, right through carry byte or word by 1 or CL

Arithmetic Instructions

ADD, SUB	Add, subtract byte or word
ADC, SBB	Add, subtract byte or word and carry (borrow)
INC, DEC	Increment, decrement byte or word
NEG	Negate byte or word (two's complement)
CMP	Compare byte or word (subtract without storing)
MUL, DIV	Multiply, divide byte or word (unsigned)
IMUL, IDIV	Integer multiply, divide byte or word (signed)
CBW, CWD	Convert byte to word, word to double word (useful before multiply/divide)
AAA, AAS, AAM, AAD	ASCII adjust for addition, subtraction, multiplication, division (ASCII codes 30-39)
DAA, DAS	Decimal adjust for addition, subtraction (binary coded decimal numbers)

Transfer Instructions

JMP	Unconditional jump
JA (JNBE)	Jump if above (not below or equal)
JAE (JNB)	Jump if above or equal (not below)
JB (JNAE)	Jump if below (not above or equal)
JBE (JNA)	Jump if below or equal (not above)
JE (JZ)	Jump if equal (zero)
JG (JNLE)	Jump if greater (not less or equal)
JGE (JNL)	Jump if greater or equal (not less)JL (JNGE) Jump if less (not greater nor equal)
JLE (JNG)	Jump if less or equal (not greater)
JC, JNC	Jump if carry set, carry not set
JO, JNO	Jump if overflow, no overflow
JS, JNS	Jump if sign, no sign
JNP (JPO)	Jump if no parity (parity odd)
JP (JPE)	Jump if parity (parity even)
LOOP	Loop unconditional, count in CX
LOOPE (LOOPZ)	Loop if equal (zero), count in CX
LOOPNE (LOOPNZ)	Loop if not equal (not zero), count in CX
JCXZ	Jump if CX equals zero

Subroutine and Interrupt Instructions

CALL, RET Call	, return from procedure
INT, INTO	Software interrupt, interrupt if overflow
IRET	Return from interrupt

String Instructions

MOVS	Move byte or word string
MOVSB, MOVSW	Move byte, word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS, STOS	Load, store byte or word string
REP	Repeat
REPE, REPZ	Repeat while equal, zero
REPNE, REPNZ	Repeat while not equal (zero)

Processor Control Instructions

STC, CLC, CMC	Set, clear, complement carry flag
STD, CLD	Set, clear direction flag
STI, CLI	Set, clear interrupt enable flag
LAHF, SAHF	Load AH from flags, store AH into flags

PUSHF, POPF	Push flags onto stack, pop flags off stack
ESC	Escape to external processor interface
LOCK	Lock bus during next instruction
NOP	No operation (do nothing)
WAIT	Wait for signal on TEST input
HLT	Halt processor

Important Usage Notes:

1. The first operand of an instruction is also the destination if there is a resulting value. Divide and multiply instructions are common exceptions to this rule.
2. There can be *at most* one memory operand per instruction.
3. There can be *at most* one immediate operand per instruction.
4. Operands generally must be of the same size (i.e., byte or word).
5. Using a label is the same as using an immediate or constant value.
6. When BP is used in a memory reference, SS is assumed as the segment. Otherwise DS is assumed.
7. While an instruction is executing, IP refers to the next instruction.
8. Many instructions are smaller if you use the appropriate registers (usually AX or AL).
9. In NASM, all labels are case sensitive but instruction and register names are not.

Terminology Used:

- **memory** - Refers to an 8 or 16-bit memory location determined by an effective address.
- **register** - AX, BX, CX, DX, SI, DI, BP, or SP as well as the 8-bit derivatives of AX, BX, CX, and DX (other registers or flags are not allowed).
- **immediate** - A numeric constant or label.
- **REG1::REG2** - The concatenation of two registers (e.g., the 32-bit value DX::AX) A single colon is used for memory addresses.
- **XF** or **XF=b** - A flag's value after an instruction can be 0 or 1 and usually depends on the result of the instruction. A flag being set to '?' by an instruction indicates that the flag is undefined after the operation.

Instructions:

adc Add with carry flag

Syntax: adc dest, src
dest: memory or register
src: memory, register, or immediate
Action: dest = dest + src + CF

Flags Affected: OF, SF, ZF, AF, PF, CF
Notes: This instruction is used to perform 32-bit addition.

add Add two numbers
Syntax: add dest, src
dest: register or memory
src: register, memory, or immediate
Action: dest = dest + src
Flags Affected: OF, SF, ZF, AF, PF, CF
Notes: Works for both signed and unsigned numbers.

and Bitwise logical AND
Syntax: and dest, src
dest: register or memory
src: register, memory, or immediate
Action: dest = dest & src
Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0

call Call procedure or function
Syntax: call addr
addr: register, memory, or immediate
Action: Push IP onto stack, set IP to addr.
Flags Affected: None

cbw Convert byte to word (signed)
Syntax: cbw
Action: Sign extend AL to create a word in AX.
Flags Affected: None
Notes: For unsigned numbers use "mov ah, 0".

cli Clear interrupt flag (disable interrupts)
Syntax: cli
Action: Clear IF
Flags Affected: IF=0

cmp Compare two operands
Syntax: cmp op1, op2
op1: register or memory
op2: register, memory, or immediate
Action: Perform op1-op2, discarding the result but setting the flags.
Flags Affected: OF, SF, ZF, AF, PF, CF
Notes: Usually used before a conditional jump instruction.

cwd Convert word to doubleword (signed)
Syntax: cwd
Action: Sign extend AX to fill DX, creating a dword contained in DX::AX.

Flags Affected: None
Notes: For unsigned numbers use "xor dx, dx" to clear DX.

dec Decrement by 1

Syntax: dec op
op: register or memory
Action: op = op - 1
Flags Affected: OF, SF, ZF, AF, PF

div Unsigned divide

Syntax: div op8
div op16
op8: 8-bit register or memory
op16: 16-bit register or memory
Action: If operand is op8, unsigned AL = AX / op8 and AH = AX % op8
If operand is op16, unsigned AX = DX::AX / op16 and DX = DX::AX % op16
Flags Affected: OF=?, SF=?, ZF=?, AF=?, PF=?, CF=?
Notes: Performs both division and modulus operations in one instruction.

imul Signed multiply

Syntax: imul op8
imul op16
op8: 8-bit register or memory
op16: 16-bit register or memory
Action: If operand is op8, signed AX = AL * op8
If operand is op16, signed DX::AX = AX * op16
Flags Affected: OF, SF=?, ZF=?, AF=?, PF=?, CF

in Input (read) from port

Syntax: in AL, op8
in AX, op8
op8: 8-bit immediate or DX
Action: If destination is AL, read byte from 8-bit port op8.
If destination is AX, read word from 16-bit port op8.
Flags Affected: None

inc Increment by 1

Syntax: inc op
op: register or memory
Action: op = op + 1
Flags Affected: OF, SF, ZF, AF, PF

int Call to interrupt procedure

Syntax: int imm8
imm8: 8-bit unsigned immediate
Action: Push flags, CS, and IP; clear IF and TF (disabling interrupts); load

word at address (imm8*4) into IP and word at (imm8*4 + 2) into CS.
Flags Affected: IF=0, TF=0
Notes: This instruction is usually used to call system routines.

iret Interrupt return

Syntax: iret
Action: Pop IP, CS, and flags (in that order).
Flags Affected: All
Notes: This instruction is used at the end of ISRs.

j?? Jump if ?? condition met

Syntax: j?? rel8
rel8: 8-bit signed immediate
Action: If condition ?? met, IP = IP + rel8 (sign extends rel8)
Flags Affected: None
Notes: Use the cmp instruction to compare two operands then j?? to jump conditionally. The ?? of the instruction name represents the jump condition, allowing for following instructions:

ja	jump if above, unsigned >
jae	jump if above or equal, unsigned >=
jb	jump if below, unsigned <
jbe	jump if below or equal, unsigned <=
je	jump if equal, ==
jne	jump if not equal, !=
jg	jump if greater than, signed >
jge	jump if greater than or equal, signed >=
jl	jump if less than, signed <
jle	jump if less than or equal, signed <=

All of the ?? suffixes can also be of the form n?? (e.g., jna for jump if not above). See 8086 documentation for many more ?? conditions.

An assembler label should be used in place of the rel8 operand. The assembler will then calculate the relative distance to jump. Note also that rel8 operand greatly limits conditional jump distance (-127 to +128 bytes from IP). Use the jmp instruction in combination with j?? to overcome this barrier.

jmp Unconditional jump

Syntax: jump rel
jump op16
jump seg:off
rel: 8 or 16-bit signed immediate
op16: 16-bit register or memory
seg:off: Immediate 16-bit segment and 16-bit offset

Action: If operand is rel, IP = IP + rel
If operand is op16, IP = op16
If operand is seg:off, CS = seg, IP = off
Flags Affected: None
Notes: An assembler label should be used in place of the rel8 operand. The assembler will then calculate the relative distance to jump.

lea Load effective address offset

Syntax: lea reg16, memref
reg16: 16-bit register
memref: An effective memory address (e.g., [bx+2])
Action: reg16 = address offset of memref
Flags Affected: None
Notes: This instruction is used to easily calculate the address of data in memory. It does not actually access memory.

mov Move data

Syntax: mov dest, src
dest: register or memory
src: register, memory, or immediate
Action: dest = src
Flags Affected: None

mul Unsigned multiply

Syntax: mul op8
mul op16
op8: 8-bit register or memory
op16: 16-bit register or memory
Action: If operand is op8, unsigned AX = AL * op8
If operand is op16, unsigned DX::AX = AX * op16
Flags Affected: OF, SF=?, ZF=?, AF=?, PF=?, CF

neg Two's complement negate

Syntax: neg op
op: register or memory
Action: op = 0 - op
Flags Affected: OF, SF, ZF, AF, PF, CF

nop No operation

Syntax: nop
Action: None
Flags Affected: None

not One's complement negate

Syntax: not op
op: register or memory
Action: op = ~op
Flags Affected: None

or Bitwise logical OR
Syntax: or dest, src
dest: register or memory
src: register, memory, or immediate
Action: dest = dest | src
Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0

out Output (write) to port
Syntax: out op, AL
out op, AX
op: 8-bit immediate or DX
Action: If source is AL, write byte in AL to 8-bit port op.
If source is AX, write word in AX to 16-bit port op.
Flags Affected: None

pop Pop word from stack
Syntax: pop op16
reg16: 16-bit register or memory
Action: Pop word off the stack and place it in op16 (i.e., op16 = [SS:SP] then SP = SP + 2).
Flags Affected: None
Notes: Pushing and popping of SS and SP are allowed but strongly discouraged.

popf Pop flags from stack
Syntax: popf
Action: Pop word from stack and place it in flags register.
Flags Affected: All

push Push word onto stack
Syntax: push op16
op16: 16-bit register or memory
Action: Push op16 onto the stack (i.e., SP = SP - 2 then [SS:SP] = op16).
Flags Affected: None
Notes: Pushing and popping of SS and SP are allowed but strongly discouraged.

pushf Push flags onto stack
Syntax: pushf
Action: Push flags onto stack as a word.
Flags Affected: None

ret Return from procedure or function
Syntax: ret
Action: Pop word from stack and place it in IP.
Flags Affected: None

sal Bitwise arithmetic left shift (same as shl)

Syntax: sal op, 1
sal op, CL
op: register or memory
Action: If operand is 1, op = op << 1
If operand is CL, op = op << CL
Flags Affected: OF, SF, ZF, AF=?, PF, CF

sar Bitwise arithmetic right shift (signed)

Syntax: sar op, 1
sar op, CL
op: register or memory
Action: If operand is 1, signed op = op >> 1 (sign extends op)
If operand is CL, signed op = op >> CL (sign extends op)
Flags Affected: OF, SF, ZF, AF=?, PF, CF

sbb Subtract with borrow

Syntax: sbb dest, src
dest: register or memory
src: register, memory, or immediate
Action: dest = dest - (src + CF)
Flags Affected: OF, SF, ZF, AF, PF, CF
Notes: This instruction is used to perform 32-bit subtraction.

shl Bitwise left shift (same as sal)

Syntax: shl op, 1
shl op, CL
op: register or memory
Action: If operand is 1, op = op << 1
If operand is CL, op = op << CL
Flags Affected: OF, SF, ZF, AF=?, PF, CF

shr Bitwise right shift (unsigned)

Syntax: shr op, 1
shr op, CL
op: register or memory
Action: If operand is 1, op = (unsigned)op >> 1
If operand is CL, op = (unsigned)op >> CL
Flags Affected: OF, SF, ZF, AF=?, PF, CF

sti Set interrupt flag (enable interrupts)

Syntax: sti
Action: Set IF
Flags Affected: IF=1

sub Subtract two numbers

Syntax: sub dest, src

dest: regsiter or memory
src: register, memory, or immediate
Action: dest = dest - src
Flags Affected: OF, SF, ZF, AF, PF, CF
Notes: Works for both signed and unsigned numbers.

test Bitwise logical compare
Syntax: test op1, op2
op1: register, memory, or immediate
op2: register, memory, or immediate
Action: Perform op1 & op2, discarding the result but setting the flags.
Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0
Notes: This instruction is used to test if bits of a value are set.

xor Bitwise logical XOR
Syntax: xor dest, src
dest: register or memory
src: register, memory, or immediate
Action: dest = dest ^ src
Flags Affected: OF=0, SF, ZF, AF=?, PF, CF=0

The 8086 Microprocessor Pin Diagram

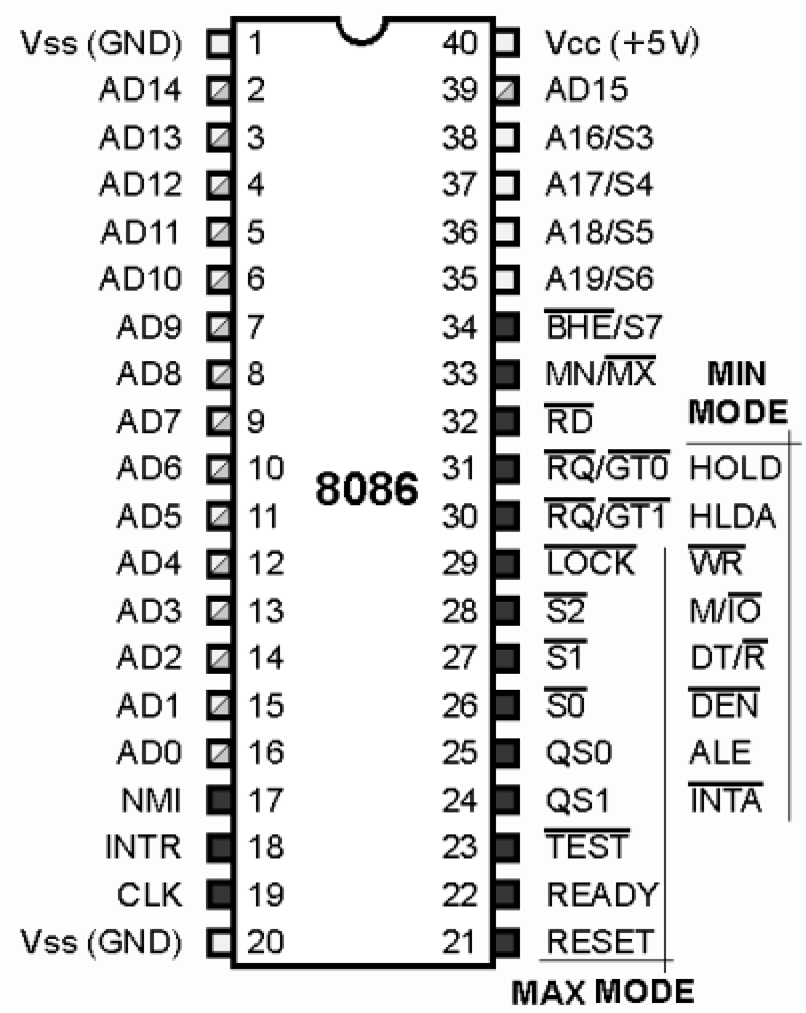


Fig:- The 8086 Microprocessor Pin Diagram

MASM COMMANDS

```
C :/>cd masm32
C:/masm32>edit filename.asm
```

After this command executed in command prompt an editor window will open. Program should be typed in this window and saved. The **program structure** is given below.

```
.model tiny/small/medium/large
.Stack <some number>
.data
        ; Initialize data
        ; which is used in program.
.code
        ; Program logic goes here.
        ;
end
```

To run the program, the following steps have to be followed:

```
C:/masm32>masm filename.asm;
```

After this command is executed in command prompt if there are no errors in program regarding to syntax the assembler will generates an object module as discuss above.

```
C:/ masm32 >link filename.obj;
```

After verifying the program for correct syntax and the generated object files should be linked together. For this the above link command should be executed and it will give an EXE file if the model directive is small as discuss above.

OR

C:/ masm32>ml /Zi filename.asm

This is similar to above commands which will check for sytax error as well as it will link the object file, in the above command Z (capital Z) is case sensitive.

C:/ masm32>debug filename.exe

After generating EXE file by the assembler it's the time to check the output. For this the above command is used and the execution of the program can be done in different ways. It is as shown below:

-g **;** complete execution of program in single step.

-t **;** Stepwise execution**.**

-d ds: starting address or ending address **;** To see data in memory locations

-p **;** Used to execute interrupt or procedure during stepwise execution of program

-q **;** To quit the execution.

C:/ masm32>cv filename.exe

This command is used as code viewer which will help in executing the code line by line parallely we can see the contents of registers

Some of the commands for code viewer are:

F8 :line by line executation

F5 :complete executation

Q :quit

Enough enough enough,

Let's start!

1. Design and develop an assembly language program to search a key element “X” in a list of ‘n’ 16-bit numbers. Adopt *Binary search* algorithm in your program for searching.

```
.model small

initds macro
    mov ax,@data        ; Initializing the Data Segment
    mov ds,ax           ; it is ds, not dx
endm

printf macro msg
    lea dx,msg          ; Load the Effective Address to DX
    mov ah,9            ; Function Number is 9
    int 21h             ; Using DOS interrupt 21h
endm

putchar macro char
    mov dl,char        ; load the printable character's HEX value in DL
    mov ah,2            ; Function Number is 9
    int 21h             ; Using DOS interrupt 21h
endm

exit macro
    mov ah,4ch          ; to terminate
    int 21h
endm

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
.data

    array dw 1122h,2345h,3333h,4455h,6666h       ; 16 bit array
    len dw ($-array)/2          ; len = (last_index - first_index)/2

    search equ 2345h            ; key to Search

    foundmsg db 'Element found at position : $'
    position db  0              ; now it’s 0, later we shall put

    notfoundmsg db 'Element not found $'

.code

    initds              ; Initializing Data Segment (call that macro)

    mov bx,1                 ; low
    mov dx,len               ; high
    mov cx,search            ; key
    again:

        cmp bx,dx           ; while(low<high)
        ja failure          ; if (low>high) then its not found case.

        mov ax,bx
        add ax,dx           ; low+high
        shr ax,1            ; (low+high) /2
        mov si,ax           ; have an index
        dec si              ; adjust the index (pointing to the mid)
        add si,si           ; for 16 bit data
        cmp cx,array[si]    ; if(key==array[mid])
        jae bigger          ; search in the RIGHT part of the array

        dec ax     ; dec high (search in the LEFT part of the array)
        mov dx,ax           ; make this as new high
        jmp again           ; continue searching

    bigger:

        je success          ; found case
        inc ax              ; inc low
        mov bx,ax           ; make this as new low
        jmp again           ; continue searching

    success:

        add al,30h          ; add 30h (or '0') to the position(AL)
                             ; (just to convert to ascii)
        mov position,al     ; move the position to our variable

        printf foundmsg     ; printing found message
        putchar position    ; printing found position
        exit                ; you are done, so bye bye!

    failure:

        printf notfoundmsg  ; printing not found message
        exit                ; bye!

end
```

****

OUTPUT:

```
masm 1.asm;

link 1.obj;

1

Element found at position : 2
```

NOTE:

- it is “mov ah,4ch and int 21h” not “mov al,4ch and int 21h”.
- it is putchar position not printf position

2. Design and develop an assembly program to sort a given set of 'n' 16-bit numbers in ascending order. Adopt *Bubble Sort* algorithm to sort given elements.

```
.model small

initds macro
    mov ax,@data       ; initializing the data segment
    mov ds,ax          ; it is ds, not dx
endm
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.data
    array dw 20h,70h,40h,10h,50h  ; our array which has to be sorted
    count dw ($-array)/2          ; length of our array (5 elements)

.code
    initds                  ; call that macro

    mov dx, count           ; copy count to dx
    dec dx                  ; n-1 iterations

    outerloop:               ; i loop

        mov cx,dx             ; temporary copy to cx
        lea si,array          ; first element's index to SI

        innerloop:                ; j loop

            mov ax,[si]           ; first element to ax
            cmp ax,[si+2]         ; compare 1st and 2nd element
            jl noswap             ; if(1st < 2nd) then don't swap

            xchg [si+2],ax  }     ; else swapping is required
            mov [si],ax     }

          noswap:

                add si,02               ; point to next element
                loop innerloop          ; finish innerloop first (j)
                dec dx                  ; dec i
                jnz outerloop           ; go and finish i loop

    int 3                       ; halt or breakpoint
    align 16                    ; properly align
end                             ; bye bye!
```

Count = ($ – array)/2
= (10 – 0)/2
= 5

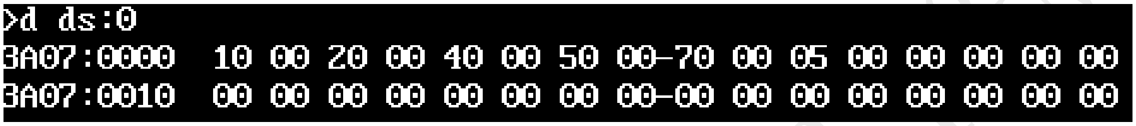
****

OUTPUT: (please follow these steps for this program)

- masm 2.asm;
- link 2.obj;
- cv 2.exe

  press f5 or g → (g means go and execute)

  d ds:0 → (d means dump, ds means data segment)



```
>d ds:0
3A07:0000  10 00 20 00 40 00 50 00-70 00 05 00 00 00 00 00
3A07:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Note:

d ds:0 means dump the data segment from 0th location

Working of Bubble Sort Algorithm

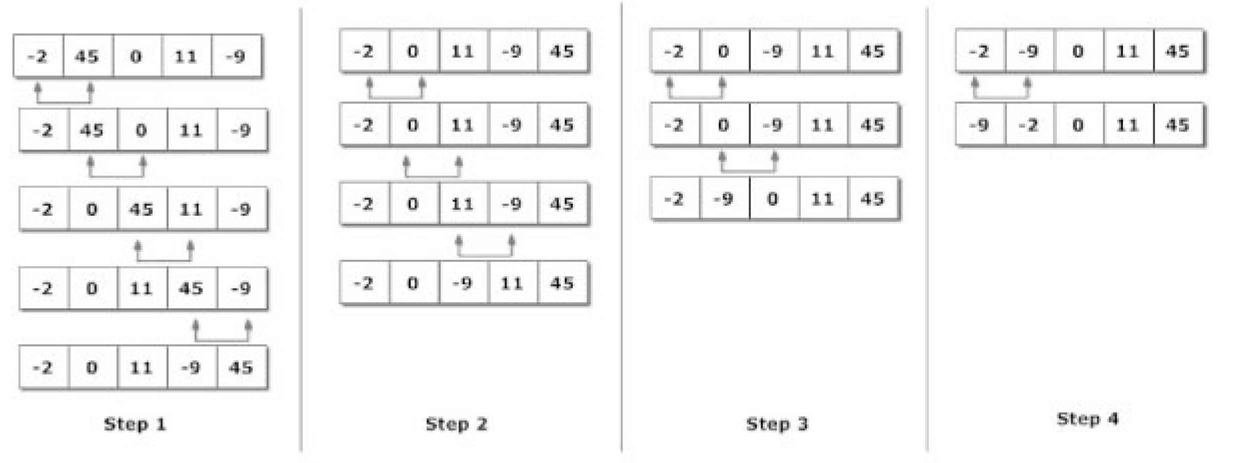


Figure: Working of Bubble sort algorithm

3. Develop an assembly language program to reverse a given string and verify whether it is a *Palindrome* or not. Display the appropriate message

```
.model small

initds macro
    mov ax,@data      ; initializing the data segment
    mov ds,ax         ; it is ds, not dx
endm

inites macro
    mov es,ax         ; initializing the extra segment
endm

printf macro msg
    lea dx,msg        ; load the effective address to dx
    mov ah,9          ; function number is 9
    int 21h           ; using dos interrupt 21h
endm

getchar macro
    mov ah,1          ; this macro takes 1 key input,
    int 21h           ; its ascii value in hex stores in al
endm

exit macro
    mov ah,4ch        ; to terminate
    int 21h
endm

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.data
      original db 30 dup(?)  ; 1st array
      reverse db 30 dup(?)   ; 2nd array to store the reversed array

      ask db 10,13,"String please:$"
      palindromemsg db 10,13,"Palindrome$"
      notpalindromemsg db 10,13,"Not Palindrome$"

.code

      initds

      inites       ; initializing extra segment (why??? b'coz we are
                   ; playing with strings)

      lea si, original ; 1st array starting index to si
      lea di, reverse  ; 2nd array starting index to di
      printf ask
      mov cx,00 ;counter..right now it’s 0 (we haven't taken any i/p)

      takeinput:

             getchar      ; takes single character (pressed key's
                          ; ascii value goes to AL automatically)
             cmp al,13    ; compare with ENTER key
             je done      ; if you press ENTER key, then goto done
             mov [si],al  ; else, store your key in array
             inc cx       ; keeps the no. of elements in array
             inc si       ; move to next position
             jmp takeinput ; repeat till you press ENTER key

      done: dec si     ; point to the last position

      reversingtask:

             mov al,[si] ; last element of si
             mov [di],al ; put that to first element of di
             inc di      ; inc 2nd array position
             dec si      ; dec 1st array position
             jnz reversingtask

      lea si, original ; comparison part
      lea di, reverse
      cld               ; clear direction flag
                       ;  (so that si & di are auto incremented)
      repe cmpsb        ; comparing [si] & [di]
      je palin     ; if all the characters are equal, then goto palin

      printf notpalindromemsg ; else, not palindrome case
      exit                    ; bye bye!


      palin: printf palindromemsg    ; palindrome
      exit                           ; bye bye!

  end
```

****

OUTPUT 1:

```
3.EXE
String please:MADAM

Palindrome
```

OUTPUT 2:

```
3.EXE
String please:COLLEGE

Not Palindrome
```

NOTE:

1. MAKE SURE YOU INITIALIZE EXTRA SEGMENT (mov es,ax)

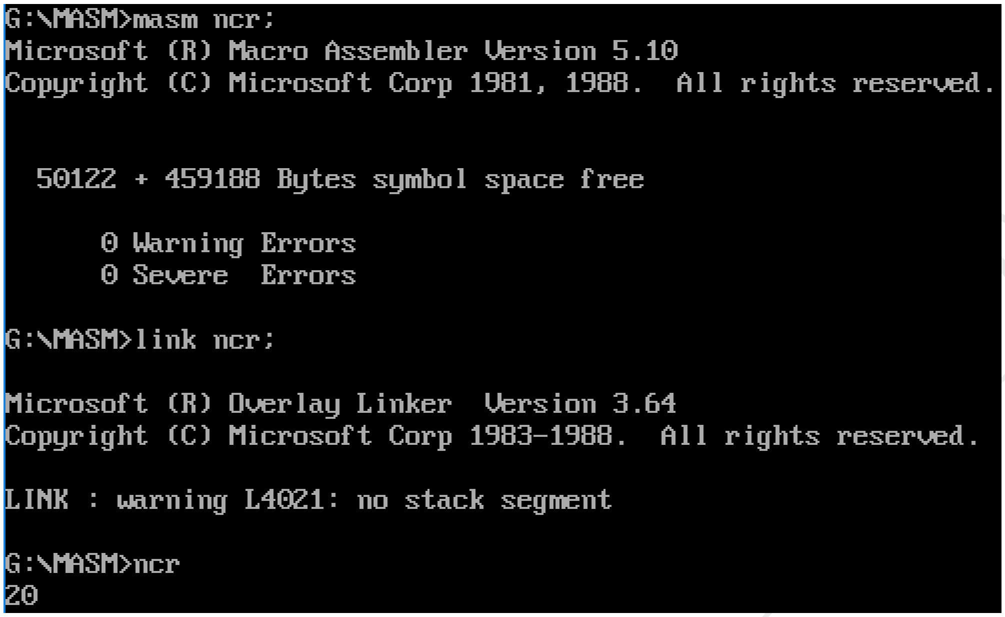
4. Develop an assembly language program to compute *nCr* using recursive procedure. Assume that ‘n’ and ‘r’ are non-negative integers.

```
.model small

initds macro
    mov ax,@data          ; initializing the data segment
    mov ds,ax             ; it is ds, not dx
endm

putchar macro char
    mov dl,char          ; load the printable character's hex value in dl
    mov ah,2              ; function number is 9
    int 21h               ; using dos interrupt 21h
endm

exit macro
    mov ah,4ch            ; to terminate
    int 21h
endm

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

.data
             n db 6                   ; aim is to find -> 6c3
             r db 3
             answer db 0
 .code
             initds

             mov al,n
             mov bl,r

             call ncr                 ; call ncr procedure

              mov al,answer            ; copy that answer to your al
             aam                      ; split al into al & ah
             add ax,3030h             ; convert into ascii
             mov bx,ax                ; take a copy to be safe
             putchar bh               ; display 1st digit
             putchar bl               ; display 2nd digit

             exit
```

```
ncr proc

            cmp bl,0                    ; nC0 = 1
            jne go1
            add answer,1
            ret

      go1:  cmp bl,al                   ; nCn = 1
            jne go2
            add answer,1
            ret

      go2:  cmp bl,1                    ; nC1 = n
            jne go3
            add answer,al
            ret

      go3:  dec al                      ; nCn-1= n
            cmp bl,al
            jne go4
            inc al
            add answer,al
            ret

      go4:  push ax
            push bx        ; n-1
            call ncr            C
            pop bx               r
            pop ax

            dec bx
            push ax       ; n-1
            push bx           C
            call ncr            r-1
            pop bx
            pop ax
            ret
ncr endp
end
```

****

OUTPUT:



```
G:\MASM>masm ncr;
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988.  All rights reserved.


  50122 + 459188 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors

G:\MASM>link ncr;

Microsoft (R) Overlay Linker  Version 3.64
Copyright (C) Microsoft Corp 1983-1988.  All rights reserved.

LINK : warning L4021: no stack segment

G:\MASM>ncr
20
```

NOTE:

FORMULA that we USE: $^{n}C_{r} = {}^{n-1}C_{r} + {}^{n-1}C_{r-1}$

USEFUL VALUES

$^{n}C_{0} = 1$

$^{n}C_{n} = 1$

$^{n}C_{1} = n$

$^{n}C_{n-1} = n$

Another formula $^{n}C_{r} = \frac{n!}{r!\,(n-r)!}$

5. Design and develop an assembly language program to read the current *Time and Date* from the system and display it in the standard format on the screen.

```
.model small

initds macro
    mov ax,@data      ; initializing the data segment
    mov ds,ax         ; it is ds, not dx
endm

printf macro msg
    lea dx,msg        ; load the effective address to dx
    mov ah,9          ; function number is 9
    int 21h           ; using dos interrupt 21h
endm

putchar macro char
    mov dl,char      ; load the printable character's hex value in dl
    mov ah,2          ; function number is 9
    int 21h           ; using dos interrupt 21h
endm

accesstime macro
    mov ah,2ch        ; time interrupt  ch=hours; cl=minutes
    int 21h           ;                 dh=seconds; dl=milliseconds
endm

accessdate macro      ; date interrupt  dl=day; dh=month; cx=year
    mov ah,2ah
    int 21h
endm

display macro value
    mov al,value      ; copy the passed value to AL bcoz next
                        instruction (aam) works only on AL
    aam               ; split al into ah & al
    add ax,3030h      ; convert ah & al to ascii
    mov bx,ax         ; copy ax to bx to be safe
    putchar bh        ; print first digit
    putchar bl        ; print second digit
endm

exit macro
    mov ah,4ch        ; to terminate
    int 21h
endm
```

```
time macro
    printf timemsg    ; print "current time is"
    accesstime        ; call accesstime macro
    display ch        ; display hours
    putchar ':'       ; print ':
    display cl        ; display minutes
endm

date macro
    printf datemsg    ; print "current date is"
    accessdate        ; call accessdate macro
    display dl        ; display day
    putchar ':'       ; print ':'
    display dh        ; display month
endm

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


.data

    timemsg db 10,13,"current time is $"
    datemsg db 10,13,"current date is $"



.code

    initds            ; initialze data segment
    time              ; time task
    date              ; date task
    exit              ; bye bye!


end
```

```
***************************************************************************************
```

OUTPUT:

5.EXE

```
current time is 10:37
current date is 14:03
```

Procedures:

Delay proc

```
DELAY PROC
        MOV AX,0CFFH
OUTER:  MOV CX,0FFFFH
INNER:  LOOP INNER
        DEC AX
        JNZ OUTER
        RET
DELAY ENDP
```

Basically, keep decrementing a huge number till zero huge number of times.

By the time, microprocessor does these huge decrements; you can actually see your front-end output.

Clear screen Proc

```
CLS PROC NEAR
          MOV AH,0FH      }       ; get the current mode
          INT 10H
          MOV AH,00H      }       ; clear that current mode
          INT 10H
          RET
CLS ENDP
```

Important Questions

1.What are the flags in 8086?
- In 8086 Carry flag, Parity flag, Auxiliary carry flag, Zero flag, Overflow flag, Trace flag, Interrupt flag, Direction flag, and Sign flag.

2.What are the various interrupts in 8086?
- Maskable interrupts, Non-Maskable interrupts.

3.What do you mean by Maskable interrupts?
- An interrupt that can be turned off by the programmer is known as Maskable interrupt.

4.What are Non-Maskable interrupts?
An interrupt which can be never be turned off (ie.disabled) is known as Non-Maskable interrupt.

5.Which interrupts are generally used for critical events?
- Non-Maskable interrupts are used in critical events. Such as Power failure, Emergency, Shut off etc.,

6.Give examples for Maskable interrupts?
- RST 7.5, RST6.5, RST5.5 are Maskable interrupts

7.Give example for Non-Maskable interrupts?
- Trap is known as Non-Maskable interrupts, which is used in emergency condition.

8.What is the Maximum clock frequency in 8086?
- 5 Mhz is the Maximum clock frequency in 8086.

9.What are the various segment registers in 8086?
- Code, Data, Stack, Extra Segment registers in 8086.

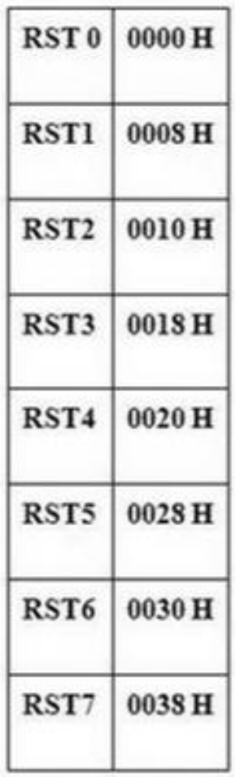
10.Which Stack is used in 8086?
- FIFO (First In First Out) stack is used in 8086.In this type of Stack the first stored information is retrieved first.

11.What is SIM and RIM instructions?
- SIM is Set Interrupt Mask. Used to mask the hardware interrupts. RIM is Read Interrupt Mask. Used to check whether the interrupt is Masked or not.

12.Which is the tool used to connect the user and the computer?
- Interpreter is the tool used to connect the user and the tool.

13.What is the position of the Stack Pointer after the PUSH instruction?
- The address line is 02 less than the earlier value.

14.What are the address lines for the software interrupts? -



RST 0	0000 H
RST1	0008 H
RST2	0010 H
RST3	0018 H
RST4	0020 H
RST5	0028 H
RST6	0030 H
RST7	0038 H

15.What is the position of the Stack Pointer after the POP instruction?
- The address line is 02 greater than the earlier value.

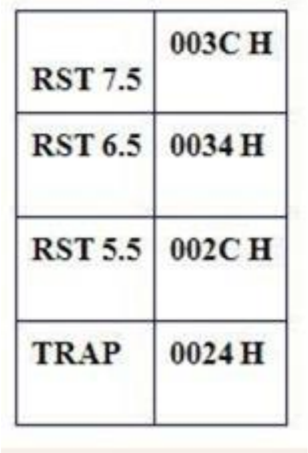
16.Logic calculations are done in which type of registers?
- Accumulator is the register in which Arithmetic and Logic calculations are done.

17.What are the different functional units in 8086?
- Bus Interface Unit and Execution unit, are the two different functional units in 8086.

18.Give examples for Micro controller?
- Z80, Intel MSC51 &96, Motorola are the best examples of Microcontroller.

19.What is meant by cross-compiler?
- A program runs on one machine and executes on another is called as cross-compiler.

20.What are the address lines for the hardware interrupts? –



RST 7.5	003C H
RST 6.5	0034 H
RST 5.5	002C H
TRAP	0024 H

21.Which Segment is used to store interrupt and subroutine return address registers?
- Stack Segment in segment register is used to store interrupt and subroutine return address registers.

22.Which Flags can be set or reset by the programmer and also used to control the operation of the processor?
- Trace Flag, Interrupt Flag, Direction Flag.

23.What does EU do?
- Execution Unit receives program instruction codes and data from BIU, executes these instructions and store the result in general registers.

24.Which microprocessor accepts the program written for 8086 without any changes?
- 8088 is that processor.

25.What is the difference between 8086 and 8088?
- The BIU in 8088 is 8-bit data bus & 16- bit in 8086.Instruction queue is 4 byte long in 8088and 6 byte in 8086.

You try to answer!!!!

1. Name the different flag registers in 8086.

2. What are GPR's and name them.

3. What is the opcode and operand ?

4. What are the different addressing modes? Give examples.

5. What are the categories of instruction set in 8086?

6. Explain AAA and DAA.

7. Name the string instructions.

8. Give the difference between CMPS and SCAS.

9. What are the interrupts?

10. Name different JUMP instructions.

11. Give the difference between MACRO and PROCEDURE.

12. What are the assembler directives? And name them.

13. What is the use of EVEN, EXTERN, GROUP.

14. Why 8086 has 2 "GND" pins.

15. What are stacks?

16. What is NMI?

17. What formulas are used to generate time delay for 8086 system?

18. Give the differences between static and dynamic RAM.

19. What are the methods of interfacing 10 devices?

20. What are the modes of operation of 8255?

******************** All the Best! ************************