CS321 – Computer Architecture

Name: M. Maheeth Reddy	Roll No.: 1801CS31	Date: 4 October 2020

Autumn 2020 – Mid Semester Examination

Ans 1:

Various Models in Computer Architecture

Von Neumann Architecture:

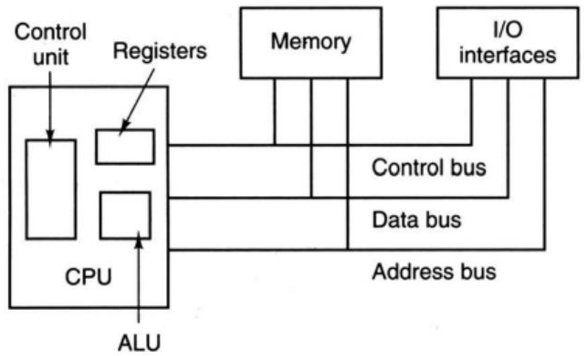
Von Neumann architecture, also called as Princeton architecture was described by John von Neumann, a Hungarian-American scientist in 1945. The architecture is based on the concept of stored program computer. In a stored program computer, instruction data and program data are stored in the same memory. The tasks we want to perform in the computer must be decided upfront, then the corresponding instructions should be stored in main memory. The instructions are then processed one-by-one sequentially unless a control flow instruction is specified.

Von Neumann architecture comprises of the following components:

1. Central Processing Unit (CPU)
2. Memory
3. Input/Output (I/O) Interfaces
4. Clock

The CPU consists of the control unit (CU), arithmetic logic unit (ALU), and various registers. The control unit determines the order in which instructions should be executed and controls the retrieval of the proper operands. It interprets the instructions of the machine. The execution of each instruction is determined by a sequence of control signals produced by the control unit.

As the name suggests, Arithmetic Logic Unit performs Mathematical and Logical operations. The registers are temporary storage locations for faster storing and transferring of data and instructions. Since registers can be accessed faster than memory, to improve performance of the computer registers are used for retrieving operands and storing results.



Computer's memory is used to store program instructions and data. Two of the commonly used type of memories are random-access memory, RAM, which is temporary and read-only memory, ROM, which is permanent. RAM stores the data and general-purpose programs that the machine executes. ROM is used to store the initial boot up instructions of the machine.

A computer must interact with the user for information and communicate with output devices to display the desired information. Input Output interfaces facilitate this function. Also, they allow the computer to communicate to the user and to secondary storage devices like disk and tape drives.

Clock is a crucial component in Von Neumann Architecture. Execution of Instructions is done in reference to the clock cycles. It indicates proper timing for retrieval of data. It keeps all components in sync.

The execution of a program in a von Neumann machine requires the use of all the above components. These components are connected to each other through buses. Buses are used to transfer multiple bits in parallel, between various hardware components. Buses are of three types:

- Data bus allows bidirectional transfer of data from a component.
- Address bus used to identify either a memory location or an I/O device.
- Control bus facilitates communication of CPU with memory and I/O devices.

A program is usually stored on an external mass storage device, like a hard disk. Prior to its execution, a program must be loaded into the memory. The I/O interfaces retrieve the program from secondary storage and load it into the memory. Once the program is in memory, the CPU is then scheduled to begin executing the program instructions. The first step of execution is called instruction fetch. Each instruction to be executed is retrieved from memory. After an instruction is fetched, it is put into the instruction register (IR), a special register in the CPU. While the instruction is in the instruction register, it is decoded to determine what type of operation should be performed. Necessary operands are fetched from memory or from other registers and the instruction is then carried out. The results are stored back into registers or memory. An instruction pointer specifies the address of the next instruction in line for execution. After executing an instruction, the next instruction is loaded to the instruction register. This process is repeated for each instruction of the program until the end of the program is reached.

<u>Advantages of Von Neumann Architecture</u>:

- Instructions and Data occupy same segment of memory.
- Since, execution is sequential, order of executions is known beforehand.
- Debugging is easy in Von Neumann Architecture due to above reason. Hence, easier to write programs.
- Design and development of Control Unit is simpler, faster and cheaper.
- Memory organization is in the hands of the programmers.

<u>Disadvantages of Von Neumann Architecture</u>:

- Parallel execution of programs cannot be done. Because instructions are executed sequentially.
- Operations that require less calculations like addition are processed at the same time as those that require more calculations like multiplication, due to dependency on clock.

- Program and data have shared memory. The data transfer rate between CPU and memory is limited while accessing large amounts of memory because processor becomes idle for certain amount of time. This is called Von Neumann Bottleneck.
- Since program and data have shared memory, there is a risk of a defective program overwriting another program in memory, causing it to crash.

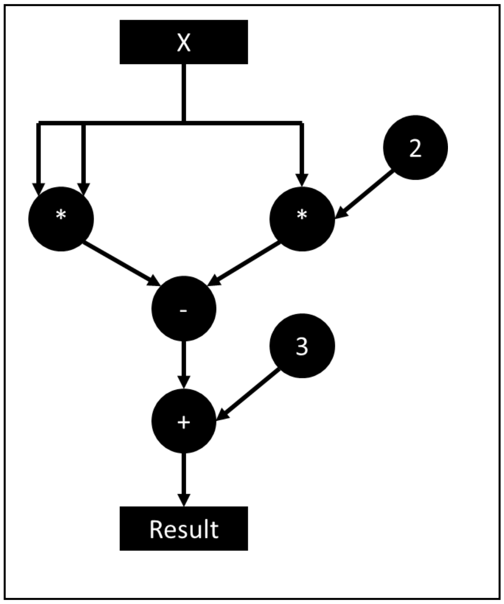
<u>Dataflow Architecture</u>:

Data flow architecture is based on the concept of data-driven computation i.e., an instruction is executed once its operands are ready or become available for computation. Therefore, there is no instruction pointer in this architecture to determine control flow. Instruction execution order is determined by data flow dependence. This characteristic contrasts this architecture from the operation of Von Neumann architecture.

The data-driven concept in this architecture allows many instructions to be executed simultaneously. Therefore, a higher degree of parallelism is inherent in a dataflow computer. Because there is no use of shared memory, dataflow programs are free from side effects mentioned above in Von Neumann Architecture.

Dataflow architecture has been successfully implemented in specialized hardware such as in digital signal processing, network routing, graphics processing, telemetry, data warehousing. They are more prominent in database engine design and parallel computing frameworks.

The order of execution of instructions depends on data flow dependence. Intermediate results are passed directly as data token between instructions. There is no concept of shared data storage unlike Von Neumann Architecture. Program sequencing is constrained only by data dependency among instructions.



Dataflow graphs can be viewed as the machine language for dataflow computers. A data flow graph is a directed graph whose nodes correspond to operators and arcs are pointers for forwarding data tokens. A producing node is connected to a consuming node by an arc, and the “point” where an arc enters a node is called an input port. The execution of an instruction is called the firing of a node. Data is sent along the arcs of the dataflow graph in the form of tokens, which are created by computational nodes and placed on output arcs.

This is a dataflow graph for calculating the expression x^2 – 2x + 3. For subtraction, x^2 and 2x are the required operands. Until both of these operands become available subtraction doesn’t occur. Among, x^2 and 2*x, the time taken for execution depends on varying x values.
After subtraction is done, the result is provided as operand for addition with 3.

Advantages of Dataflow Architecture:

- Since there is no requirement of clock, power consumption is lower than that of Von Neumann Architecture.
- Operations that require less calculations such as addition are processed faster than those that require more calculations like multiplication unlike Von Neumann Architecture.
- There is greater scope of Parallel Processing
- Design of dataflow architecture is problem specific. Therefore, it is used into devices which do not require to be programmed, such as hearing aids, pacemakers etc.

Disadvantages of Dataflow Architecture:

- Extra circuitry is required to check whether the operands are ready for an instruction to be executed.
- Programs on dataflow architecture are difficult to debug because order of execution of instructions depends on inputs. So, it becomes difficult to locate the instruction causing error.
- Design of dataflow architecture is problem specific. Therefore, this architecture can't produce a comprehensive solution for a programmable system as of date.

Stack Machine:

A "stack machine" is a computer that is based on the stack data structure. It uses a last-in, first-out stack to hold short-lived temporary values. Instructions set is designed such that operands are retrieved from the stack, and results are stored in the stack.

For a typical instruction, the operands are "popped" off the stack, the result is calculated, and its results are then "pushed" back onto the stack, ready for the next instruction. Majority of the instructions of stack machines consists of only an opcode commanding an operation, with no additional fields to identify a constant, register or memory cell. The stack easily holds more than two inputs or more than one result, so a richer set of operations can be computed. Integer constant operands are often pushed by separate Load Immediate instructions. Memory is often accessed by separate Load or Store instructions containing a memory address or calculating the address from values in the stack.

For speed, a stack machine often implements some part of its stack with registers. To execute quickly, operands of the arithmetic logic unit (ALU) may be the top two registers of the stack and the result from the ALU is stored in the top register of the stack. Some stack machines have a stack of limited size, implemented as a register file. The ALU will access this with an index. Some machines have a stack of unlimited size, implemented as an array in RAM accessed by a "top of stack" address register. This is slower, but the number of flip-flops is less, making a less-expensive, more compact CPU. Its topmost N values may be cached for speed. A few machines have both an expression stack in memory and a separate register stack. In this case, software, or an interrupt may move data between them.

The instruction set carries out most ALU actions with postfix operations that work only on the expression stack, not on data registers or main memory cells. This is very convenient for executing high-level languages because most arithmetic expressions can be easily translated into postfix notation.

Stack machines may have their expression stack and their call-return stack separated or as one integrated structure. If they are separated, the instructions of the stack machine can be pipelined with fewer interactions and less design complexity hence, making it faster.

Advantages of Stack Machines:
- Compact code because each instruction consists of only an opcode
- Low hardware requirements
- Easy to write a simpler compiler for stack architectures

Disadvantages of Stack Machines:
- There is less scope for parallelism or pipelining
- Data is not always at the top of stack when needed, so additional instructions to retrieve required data are required.
- Difficult to write an optimizing compiler for stack architectures

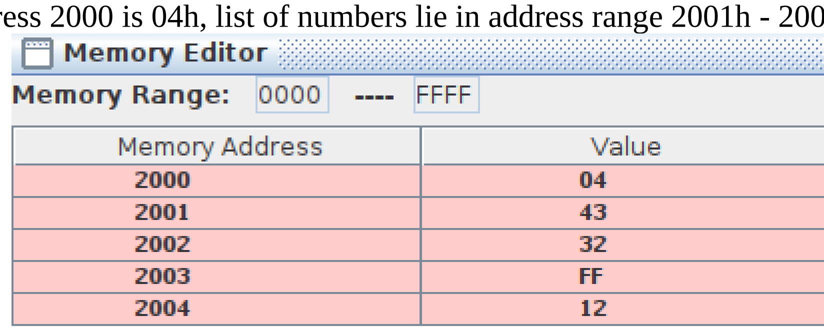
Ans 2:

Program to sort numbers in ascending order using Selection Sort

Specification: Sort given set of numbers in ascending order using Selection Sort. The count of numbers to be sorted is present at memory address 2000. The list of numbers begins from address 2001 onwards.

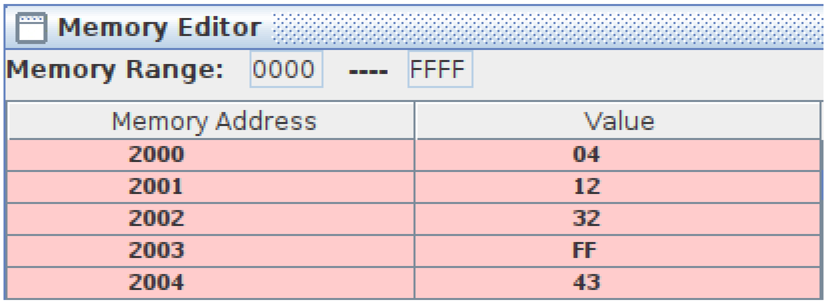
Method:
1. Start with the first number of the array
2. Calculate the minimum number upto the end of the array
3. If the minimum number is lesser than the first number, swap them.
4. Repeat steps 1-3 with second number of the array, followed by third, fourth and so on

Screenshots:
Input: Input is specified in the memory editor as shown here, before program is executed. Since, contents of address 2000 is 04h, list of numbers lie in address range 2001h - 2004h.



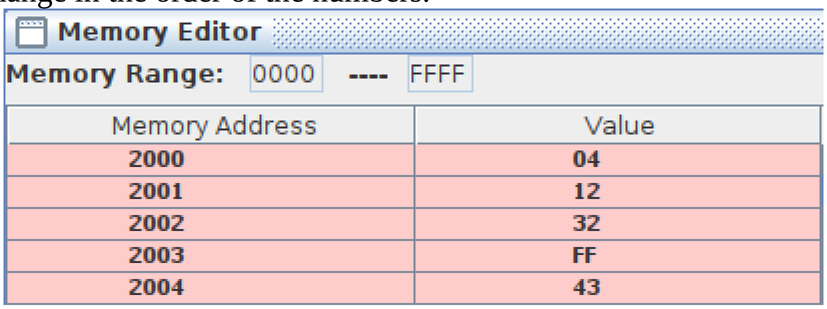
Memory Address	Value
2000	04
2001	43
2002	32
2003	FF
2004	12

Simulation:
In the screenshot above, the first number is 43h and the minimum number is 12h (starting from first number). After the first iteration, these two values are swapped.


Memory Editor
Memory Range: 0000 ---- FFFF

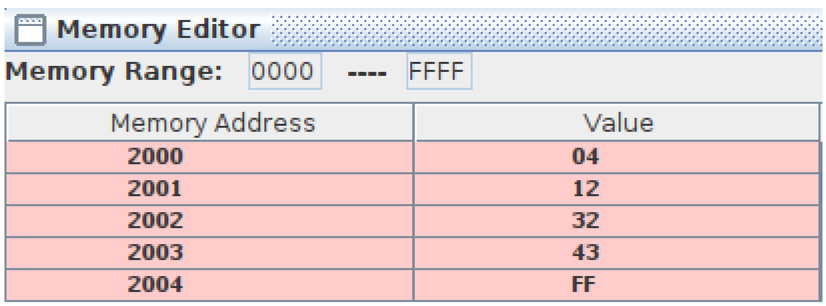
Memory Address	Value
2000	04
2001	12
2002	32
2003	FF
2004	43

The second number is 32h and the minimum number is 32h itself (starting from second number). So, there is no change in the order of the numbers.


Memory Editor
Memory Range: 0000 ---- FFFF

Memory Address	Value
2000	04
2001	12
2002	32
2003	FF
2004	43

Now, the third number FFh is greater than 43h, therefore they will be swapped in the next iteration.


Memory Editor
Memory Range: 0000 ---- FFFF

Memory Address	Value
2000	04
2001	12
2002	32
2003	43
2004	FF

In this way numbers are sorted using Selection Sort.

Code:

```
; clear registers
MVI A,00H
LXI B,0000H
LXI D,0000H

; count of numbers in input is stored at location 2000
; load address 2000 in HL pair
LXI H,2000H

; Store value at address 2000 in C
MOV C,M
CMP C
; if C is 0, quit program
```

```
; because no numbers to sort
JZ QUIT

; address of stack pointer is set to 4000h
LXI SP,4000H

; Loop till the end of given numbers,
; starting from i-th number (0 <= i <= C-1).
; Find the minimum and swap
; with i-th number if lesser
LOOP:
    ; increment HL pair to next location
    INX H
    ; copy value in M to A
    MOV A,M
    CALL FIND_MIN_NUM
    ; compare min number with A
    CMP M
    ; if both are equal dont swap
    JZ DONT_SWAP
    ; otherwise swap
    CALL SWAP
DONT_SWAP:
    DCR C
    JNZ LOOP
QUIT:   HLT

; function to find the minimum
; number in the array by iterating
; through each number until the end
FIND_MIN_NUM:
    ; store current values of
    ; H and B on stack
    ; since we use H,B in this function
    PUSH H
    PUSH B
    ; C stores count of remaining numbers
    DCR C
```

```
ITER:
    ; increment HL pair
    INX H
    ; compare A and M
    CMP M
    ; if equal don't update A
    JC SKIP
    ; update A
    MOV A,M
    ; store location of this new minimum in DE
    MOV D,H
    MOV E,L
SKIP:
    DCR C
    JNZ ITER

    ; restore H,B
    POP B
    POP H
    RET

; function to swap 2 numbers
; in memory
SWAP:
    ; push program status to stack
    PUSH PSW
    ; push value in BC to stack
    PUSH B
    ; load data into A from
    ; location pointed by DE
    LDAX D
    ; copy value in A to B
    MOV B,A
    ; copy value at location HL to A
    MOV A,M
    ; store value in A at location DE
    STAX D
```

```
; store value in B at location HL
MOV M,B
; pop top of stack to BC
POP B
; program status is restored
; in calling function
POP PSW
RET
```

Ans 3:

Program to evaluate a quadratic expression x^2 + 3*x + 2

Procedure:

1. Take a number (0-9) as input from user using INT 21H/AH = 01H interrupt
2. Convert the input from ASCII to integer value
3. Call function to evaluate the expression
    a. Calculate X^2
    b. Calculate 3*X
    c. Add X^2, 3*X, 2
4. Print result on screen using a custom Print function and INT 21H/AH = 02H interrupt
5. Return control to DOS prompt using INT 21H/AH = 4CH

Code:

```
; Program to evaluate a quadratic expression x^2 + 3*x + 2
.model small
.stack 64
.data
   X DB 5      ; X is variable
   co1 DB 3    ; x-coefficient
   const DB 2  ; constant term
   poly DW ?   ; store end result

   prompt db 'Enter a number from (0-9): $'
   newline db ' ',13,10,'$'    ; used to print newline
   finish db 'For x = $'
   result db 'x.x + 3.x + 2 is $'
.code
main proc far
   ; initialize data segment register
   mov ax,@data
   mov ds,ax
```

```
    ; show prompt message
    ; ask user to enter a number
    mov ah,09h
    lea dx,prompt
    int 21h

    ; keyboard input interrupt
    mov ah, 1h
    ; read character into al
    int 21h
    sub al,'0'
    mov X,al ;copy number to cl

    ; print newline
    mov ah, 09h
    lea dx, newline
    int 21h

    ; end of prompt
    mov ah,09
    lea dx,finish
    int 21h

    ; character output interrupt
    mov dl,X
    add dl,'0'
    mov ah, 2h
    int 21h

    ; print newline
    mov ah, 09h
    lea dx, newline
    int 21h

    ; function to evaluate expression
    call eqn

    ; show result string
    mov ah,09
    lea dx,result
    int 21h

    ; print result on screen
    mov ax,poly
    call print

    ; return to DOS
    mov ah,4ch
    int 21h
main endp

eqn proc near
    ; calculate X^2
    mov al,X
    mul al
    mov bx,ax

    ; calculate 3*X
    mov al,X
    mul co1

    ; calculate X^2 + 3*X + 2
    add ax,bx
    mov dl,const
    mov dh,00
    add ax,dx

    ; store result and return
    mov poly,ax
    ret
eqn endp
```

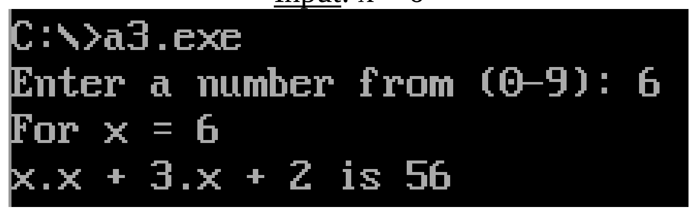
```
; print function
print proc
    ; cx stores count of digits in number
    mov cx,0
    mov dx,0
cmd:
    cmp ax,0
    je printN
    mov bx,10 ; initialize bx to 10
    div bx
    ; extract the last digit
    push dx
    ; push it to stack
    inc cx
    ; increment the count
    mov dx,0
    jmp cmd
printN:
    ; check if count is greater than zero
    cmp cx,0
    je exit
    ; pop the top of stack
    pop dx
    ; convert to ASCII
    add dx,'0'
    ; print character interrupt
    mov ah,02h
    int 21h
    ; decrease the count
    dec cx
    jmp printN
exit: ret
print endp

     end main
```

<u>Screenshots</u>:

Test Case 1:

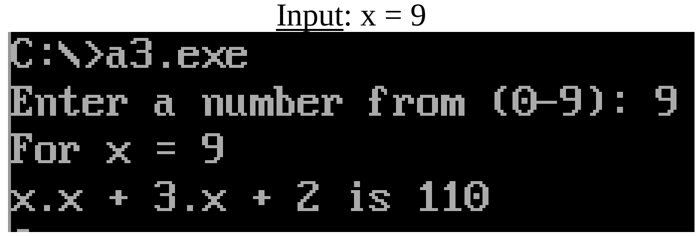
<u>Input</u>: x = 6



<u>Output</u>: X^2 + 3*X + 2 = 56

Test Case 2:

<u>Input</u>: x = 9



<u>Output</u>: X^2 + 3*X + 2 = 110

<u>Ans 4</u>:

<u>Different Addressing Modes in 8086 microprocessor</u>

Every instruction of a program has to operate on a data. The different ways in which a source operand is denoted in an instruction are known as addressing modes.

In 8086 microprocessor, there are 12 addressing modes:

<u>Register Addressing</u>:
The instruction will specify the name of the register which holds the data to be operated by the instruction.
Example:
MOV DS, AX
Here, the content of 16-bit register AX is moved to another 16-bit register DS

```
AX=076D  BX=0000  CX=7040  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0E6E  CS=076A  IP=0003   NV UP EI PL NZ NA PO NC
076A:0003 8ED8          MOV     DS,AX
-t

AX=076D  BX=0000  CX=7040  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076D  ES=075A  SS=0E6E  CS=076A  IP=0005   NV UP EI PL NZ NA PO NC
```

Notice the change in content of DS register from 075A to 076D which is the content of AX.

<u>Immediate Addressing</u>:
In immediate addressing mode, an 8-bit or 16-bit data is specified as part of the instruction.
Example:
MOV AX, 2024H
The 16-bit data (2024H) given in the instruction is moved to AX register

```
AX=076D  BX=0000  CX=7040  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=0007   NV UP EI PL NZ NA PO NC
076A:0007 B82420        MOV     AX,2024
-t

AX=2024  BX=0000  CX=7040  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=000A   NV UP EI PL NZ NA PO NC
```

Notice the change in content of AX register from 076D to 2024

<u>Direct Addressing</u>:
The effective address (16-bit number) of the memory location at which the data operand is stored is directly given in the instruction as shown.
Example:
MOV CH, DS:[7009H]
Data at location 7009H is moved to CH register

```
AX=2024  BX=0000  CX=7040  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=000A   NV UP EI PL NZ NA PO NC
076A:000A 8A2E0970      MOV     CH,[7009]                          DS:7009=74
-t

AX=2024  BX=0000  CX=7440  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=000E   NV UP EI PL NZ NA PO NC
```
Notice the change in content of CH register from 70 to 74, which is the content in DS:7009

Register Indirect Addressing:
Name of the register (any of BX, BP, DI and SI) which holds the effective address (EA) will be specified in the instruction. Content of DS register is used for base address calculation. The data at the calculated address is moved to destination register.
Example:
MOV CL, [BX]
Data at address DS:BX (DS*16 + BX) is moved to CL register
```
AX=2024  BX=7004  CX=7440  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=0012   NV UP EI PL NZ NA PO NC
076A:0012 8A0F          MOV     CL,[BX]                            DS:7004=52
-t

AX=2024  BX=7004  CX=7452  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=0014   NV UP EI PL NZ NA PO NC
```
Notice change in content of CL register from 40 to 52, which is the content at DS:7004. 7004 is the content of BX

Based Addressing:
In Based Addressing, BX or BP is used to hold the base value for effective address and an offset of 8-bit(signed) or 16-bit(unsigned) will be specified in the instruction.
In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.
When BX holds the base value of EA, 20-bit physical address is calculated from BX and DS.
When BP holds the base value of EA, BP and SS is used.
Example:
MOV CH, [BX + 01H]
Data at address DS:(BX+**00**01H) i.e., DS*16 + BX + 0001H is moved to CH register
```
AX=2024  BX=7004  CX=7452  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=0014   NV UP EI PL NZ NA PO NC
076A:0014 8A6F01        MOV     CH,[BX+01]                         DS:7005=65
-t

AX=2024  BX=7004  CX=6552  DX=0000  SP=0040  BP=0000  SI=0000  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=0017   NV UP EI PL NZ NA PO NC
```
Notice change in content of CH register from 74 to 65, which is the content in DS:7005. BX contains 7004 and BX+01 = 7005

Indexed Addressing:
SI or DI register is used to hold an index value for memory data and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction.
Displacement is added to the index value in SI or DI register to obtain the EA.

In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.
Example:
MOV CX, [SI + 0A2H]
Data at address DS:(SI+**FF**A2H) i.e., DS*16 + SI + FFA2H is moved to register CX

MOV CL, [SI+01]
Data at address DS:(SI**+00**01H**)** i.e., DS*16 + SI + 0001H is moved to register CL

```
AX=2024  BX=7004  CX=6552  DX=0000  SP=0040  BP=0000  SI=7004  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=001B   NV UP EI PL NZ NA PO NC
076A:001B 8A4C02        MOV     CL,[SI+02]                    DS:7006=FF
-t

AX=2024  BX=7004  CX=65FF  DX=0000  SP=0040  BP=0000  SI=7004  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=001E   NV UP EI PL NZ NA PO NC
```

Notice the change in contents of CL from 52 to FF, which is present at DS:7006. SI contains 7004 and SI+02is 7006

Based Index Addressing:
In this Addressing, the effective address is computed from the sum of a base register (BX or BP), an index register (SI or DI) and a displacement.
Example:
MOV DL, [BX + SI + 02H]
Data at address DS:(BX + SI + 0002H) i.e., DS*16 + BX + SI + 0002 is moved to register DL

```
AX=2024  BX=7004  CX=65FF  DX=0000  SP=0040  BP=0000  SI=0001  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=0021   NV UP EI PL NZ NA PO NC
076A:0021 8A5002        MOV     DL,[BX+SI+02]                 DS:7007=54
-t

AX=2024  BX=7004  CX=65FF  DX=0054  SP=0040  BP=0000  SI=0001  DI=0000
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=0024   NV UP EI PL NZ NA PO NC
```

Notice change in contents of DL register from 00 to 54, which is present at DS:7007. BX contains 7004, SI contains 0001 and BX+SI+02 is 7007
String Addressing:
Employed in string operations to operate on string data.
The effective address (EA) of source data is stored in SI register and the EA of destination is stored in DI register.
Segment register for calculating base address of source data is DS and that of the destination data is ES.
Example:
MOVSB
Moves a string of width one byte from Source operand to Destination operand
Source operand address is DS:SI
Destination Operand Address is ES:DI

```
AX=2024  BX=700A  CX=65FF  DX=0054  SP=0040  BP=0000  SI=7010  DI=7013
DS=076D  ES=076D  SS=0E6F  CS=076A  IP=002C   NV UP EI PL NZ NA PO NC
076A:002C A5            MOVSW
```

DS:SI contains string NO$ and ES:DI contains PA$

```
076D:7000                                 4E 4F 24 50 41 24         NO$PA$
076D:7010  08 9A 39 CD A2 01 72 F2-8B 07 89 46 FE 8B 47 02   ..9...r....F..G.
076D:7020  89 46 FC 8B 46 06 33 D2-89 07 89 57 02 8B 4E 04   .F..F.3....W..N.
076D:7030  83 F9 02 75 28 0B C0 74-24 83 3E 20 0E 00 75 1D   ...u(..t$.> ..u.
076D:7040  53 B0 23 B4 35 CD 24 20-00 00 2C 00 6A 07 A3 01   S.#.5.$ ..,.j...
076D:7050  BA 61 CD 1E 0E 1F B0 23-B4 25 CD 21 1F 83 F9 08   .a.....#.%.!....
076D:7060  75 1D B8 DE CD 8B 0F 83-F9 02 73 0A B8 D9 CD 0B   u.........s.....
076D:7070  C9 74 03 B8 D8 CD 8C CA-BB 03 00 FF 1E 78 0F 8B   .t...........x..
076D:7080  46 FE 8B 56 FC 8B E5 5D-C3 55                     F..V...].U
```

After executing MOVSB, ES:DI also contains NO$

```
-d ds:700A
076D:7000                                 4E 4F 24 4E 4F 24         NO$NO$
076D:7010  08 9A 39 CD A2 01 72 F2-8B 07 89 46 FE 8B 47 02   ..9...r....F..G.
076D:7020  89 46 FC 8B 46 06 33 D2-89 07 89 57 02 8B 4E 04   .F..F.3....W..N.
076D:7030  83 F9 02 75 28 0B C0 74-24 83 3E 20 0E 00 75 1D   ...u(..t$.> ..u.
076D:7040  53 B0 23 B4 35 CD 24 20-00 00 2D 00 6A 07 A3 01   S.#.5.$ ..-.j...
076D:7050  BA 61 CD 1E 0E 1F B0 23-B4 25 CD 21 1F 83 F9 08   .a.....#.%.!....
076D:7060  75 1D B8 DE CD 8B 0F 83-F9 02 73 0A B8 D9 CD 0B   u.........s.....
076D:7070  C9 74 03 B8 D8 CD 8C CA-BB 03 00 FF 1E 78 0F 8B   .t...........x..
076D:7080  46 FE 8B 56 FC 8B E5 5D-C3 55                     F..V...].U
```

<u>Direct I/O port Addressing</u>:

This addressing mode is used to input data from standard I/O mapped devices or ports.

Example: IN AL, [0009H]

Content of port at address is moved to AL register

```
AX=2024  BX=7004  CX=65FF  DX=0054  SP=0040  BP=0000  SI=700C  DI=700F
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=002D   NV UP EI PL NZ NA PO NC
076A:002D E409          IN      AL,09
-t

AX=20FF  BX=7004  CX=65FF  DX=0054  SP=0040  BP=0000  SI=700C  DI=700F
DS=076D  ES=076D  SS=0E6E  CS=076A  IP=002F   NV UP EI PL NZ NA PO NC
```

Port at address 0009 contains FF and is moved to AL register

<u>Indirect I/O port Addressing</u>:

In indirect port addressing mode, the instruction will specify the name of the register which holds the port address. In 8086, the 16-bit port address is stored in the DX register.

Example: OUT [DX], AX

Content of AX is moved to port whose address is specified by DX register.

<u>Relative Addressing</u>:

In this addressing mode, the effective address of a program instruction is specified relative to Instruction Pointer (IP) by an 8-bit signed displacement.

Example:

JZ 0031H

Here, if ZF = 1, the control flow jumps to instruction at address CS:(IP + 0031H) i.e., CS*16 + IP + 0031H.

If ZF = 0, control flow goes to next instruction

Notice the change in value of Instruction Pointer, IP

```
AX=2001  BX=700A  CX=65FF  DX=0054  SP=0040  BP=0000  SI=7012  DI=7015
DS=076D  ES=076D  SS=0E6F  CS=076A  IP=0032   NV UP EI PL NZ NA PO NC
076A:0032 2C01          SUB     AL,01
-t

AX=2000  BX=700A  CX=65FF  DX=0054  SP=0040  BP=0000  SI=7012  DI=7015
DS=076D  ES=076D  SS=0E6F  CS=076A  IP=0034   NV UP EI PL ZR NA PE NC
076A:0034 74FB          JZ      0031
-t

AX=2000  BX=700A  CX=65FF  DX=0054  SP=0040  BP=0000  SI=7012  DI=7015
DS=076D  ES=076D  SS=0E6F  CS=076A  IP=0031   NV UP EI PL ZR NA PE NC
```

Implied Addressing:
Instructions using this mode have no operands. The instruction itself will specify the data to be operated by the instruction.
Example: CLC
This clears the carry flag to zero.

```
076A:002F F8            CLC
-t

AX=20FF  BX=7006  CX=65FF  DX=0054  SP=0040  BP=0000  SI=700E  DI=7011
DS=076D  ES=076D  SS=0E6F  CS=076A  IP=0030   NV UP EI PL NZ NA PO NC
```

Ans 5:

Program to count number of vowels in an alphabet string

Procedure:

1. Compute length of input string
2. Copy input string to buffer
3. Convert buffer to lowercase
4. Search for vowels in the buffer
5. Increment count whenever a vowel is encountered

Code:

```
; Program to count number of vowels in an alphabet string
.model small
.stack 64
.data
   input DB 'MaheEth$'
   len DW 0
   vowCount DB 0
   vowels DB 'aeiou$'
   numV EQU 05h
   buffer DB ?
.code
main proc far
```

```
; initialize data segment register
    mov ax,@data
    mov ds,ax
    ; initialize extra segment register
    mov es,ax
    ; load address of string to DI register
    lea di,input

    ; count length of string
    mov al,'$'  ; end of string character
    lea si,input
count:
    ; compare [di] with end of string character
    scasb
    ; if equal, stop counting
    je copy_str
    ; otherwise increment length of string
    inc len
    ; keep counting
    jmp count

copy_str:
    ; copy string from input to buffer using m
ovsb
    mov cx, len
    lea si, input
    lea di, buffer
    rep movsb

    ; convert buffer to lowercase
    mov cx, len
    lea si, buffer
    lea di, buffer
to_lower:
    ; retrieve character from buffer
    lodsb
    or al,20h
```

```
    ; change case and store in buffer
    stosb
    dec cx
    jnz to_lower

    ; search for vowels in buffer
    lea si,buffer
    mov bx,len
search:
    mov al,[si]
    lea di,vowels
    mov cx,numV
    cld
    repne scasb
    jne not_found

    ; if a character is vowel increment vowCount
found:
    inc vowCount
    mov dl,vowCount

    ; otherwise, read next character
not_found:
    inc si
    dec bx
    jnz search

    ; copy vowCount to dl register to see in debu
gger
    mov dl,vowCount

    ; exit program
    mov ah,4ch
    int 21h
main endp
     end main
```

Screenshots:

```
AX=0724  BX=0000  CX=0078  DX=0000  SP=0040  BP=0000  SI=0006  DI=000D
DS=0770  ES=0770  SS=0772  CS=076A  IP=0014   NV UP EI NG NZ AC PO CY
076A:0014 FF060E00      INC     WORD PTR [000E]                    DS:000E=0006
-t

AX=0724  BX=0000  CX=0078  DX=0000  SP=0040  BP=0000  SI=0006  DI=000D
DS=0770  ES=0770  SS=0772  CS=076A  IP=0018   NV UP EI PL NZ NA PO CY
076A:0018 EBF7          JMP     0011
-t

AX=0724  BX=0000  CX=0078  DX=0000  SP=0040  BP=0000  SI=0006  DI=000D
DS=0770  ES=0770  SS=0772  CS=076A  IP=0011   NV UP EI PL NZ NA PO CY
076A:0011 AE            SCASB
-t

AX=0724  BX=0000  CX=0078  DX=0000  SP=0040  BP=0000  SI=0006  DI=000E
DS=0770  ES=0770  SS=0772  CS=076A  IP=0012   NV UP EI PL ZR NA PE NC
076A:0012 7406          JZ      001A
-t

AX=0724  BX=0000  CX=0078  DX=0000  SP=0040  BP=0000  SI=0006  DI=000E
DS=0770  ES=0770  SS=0772  CS=076A  IP=001A   NV UP EI PL ZR NA PE NC
076A:001A 8B0E0E00      MOV     CX,[000E]                          DS:000E=0007
```

SCASB compares each character of the input string with '$' to check for end of string. Until then length of input string is counted. Notice length is 7, shown at bottom right.

```
-d ds:0006
0770:0000                    4D 61-68 65 45 74 68 24 07 00         MaheEth$..
0770:0010  00 61 65 69 6F 75 24 4D-61 68 65 45 74 68 8B B6   .aeiou$MaheEth..
0770:0020  FA FE 81 E6 FF 00 C6 82-FB FE 00 2B C0 50 8D 86   ...........+.P..
0770:0030  FB FE 50 E8 08 6A 83 C4-04 0B C0 75 03 E9 A5 00   ..P..j.....u....
0770:0040  C7 86 7A FF 00 00 EB 04-FF 86 7A FF A1 70 08 39   ..z.......z..p.9
0770:0050  86 7A FF 72 03 E9 24 07-00 00 28 00 6A 07 A3 01   .z.r..$...(.j...
0770:0060  8D 86 FA FE 50 8D 86 7C-FF 50 E8 C5 72 83 C4 06   ....P..¦.P..r...
0770:0070  8B 9E 7A FF D1 E3 D1 E3-8B 87 CC 17 8B 97 CE 17   ..z.............
0770:0080  89 46 FC 89 56 FE                                 .F..V.
```

Now, input string is copied to buffer. As seen here, "MaheEth" appears twice. Vowels are stored as "aeiou" for comparision.

```
-d ds:0006
0770:0000                    4D 61-68 65 45 74 68 24 07 00         MaheEth$..
0770:0010  00 61 65 69 6F 75 24 6D-61 68 65 65 74 68 8B B6   .aeiou$maheeth..
0770:0020  FA FE 81 E6 FF 00 C6 82-FB FE 00 2B C0 50 8D 86   ...........+.P..
0770:0030  FB FE 50 E8 08 6A 83 C4-04 0B C0 75 03 E9 A5 00   ..P..j.....u....
0770:0040  C7 86 7A FF 00 00 EB 04-FF 86 7A FF A1 70 08 39   ..z.......z..p.9
0770:0050  86 7A FF 72 03 E9 68 07-00 00 3B 00 6A 07 A3 01   .z.r..h...;.j...
0770:0060  8D 86 FA FE 50 8D 86 7C-FF 50 E8 C5 72 83 C4 06   ....P..¦.P..r...
0770:0070  8B 9E 7A FF D1 E3 D1 E3-8B 87 CC 17 8B 97 CE 17   ..z.............
0770:0080  89 46 FC 89 56 FE                                 .F..V.
```

Buffer is converted to lowercase. As shown here, "MaheEth" has become "maheeth"

```
AX=0761  BX=0006  CX=0005  DX=0000  SP=0040  BP=0000  SI=0018  DI=0011
DS=0770  ES=0770  SS=0772  CS=076A  IP=004C   NV UP EI PL NZ NA PE CY
076A:004C FC            CLD
-t

AX=0761  BX=0006  CX=0005  DX=0000  SP=0040  BP=0000  SI=0018  DI=0011
DS=0770  ES=0770  SS=0772  CS=076A  IP=004D   NV UP EI PL NZ NA PE CY
076A:004D F2            REPNZ
076A:004E AE            SCASB
-t

AX=0761  BX=0006  CX=0004  DX=0000  SP=0040  BP=0000  SI=0018  DI=0012
DS=0770  ES=0770  SS=0772  CS=076A  IP=004F   NV UP EI PL ZR NA PE NC
076A:004F 7508          JNZ     0059
-t

AX=0761  BX=0006  CX=0004  DX=0000  SP=0040  BP=0000  SI=0018  DI=0012
DS=0770  ES=0770  SS=0772  CS=076A  IP=0051   NV UP EI PL ZR NA PE NC
076A:0051 FE061000      INC     BYTE PTR [0010]                    DS:0010=00
-t

AX=0761  BX=0006  CX=0004  DX=0000  SP=0040  BP=0000  SI=0018  DI=0012
DS=0770  ES=0770  SS=0772  CS=076A  IP=0055   NV UP EI PL NZ NA PO NC
076A:0055 8A161000      MOV     DL,[0010]                          DS:0010=01
```

Whenever a vowel is encountered while searching buffer for vowels, vowel count is incremented. In this case the character being read is 'a'. Therefore count increases. Notice the last two lines at the right edge of this screenshot.

```
AX=0768  BX=0001  CX=0000  DX=0003  SP=0040  BP=0000  SI=001D  DI=0016
DS=0770  ES=0770  SS=0772  CS=076A  IP=0059   NV UP EI NG NZ NA PE CY
076A:0059 46            INC     SI
-t

AX=0768  BX=0001  CX=0000  DX=0003  SP=0040  BP=0000  SI=001E  DI=0016
DS=0770  ES=0770  SS=0772  CS=076A  IP=005A   NV UP EI PL NZ NA PE CY
076A:005A 4B            DEC     BX
-t

AX=0768  BX=0000  CX=0000  DX=0003  SP=0040  BP=0000  SI=001E  DI=0016
DS=0770  ES=0770  SS=0772  CS=076A  IP=005B   NV UP EI PL ZR NA PE CY
076A:005B 75E6          JNZ     0043
-t

AX=0768  BX=0000  CX=0000  DX=0003  SP=0040  BP=0000  SI=001E  DI=0016
DS=0770  ES=0770  SS=0772  CS=076A  IP=005D   NV UP EI PL ZR NA PE CY
076A:005D 8A161000      MOV     DL,[0010]                          DS:0010=03
-t
```

After vowels are counted, the count is displayed here in debugger as DS:0010=03 at bottom right.

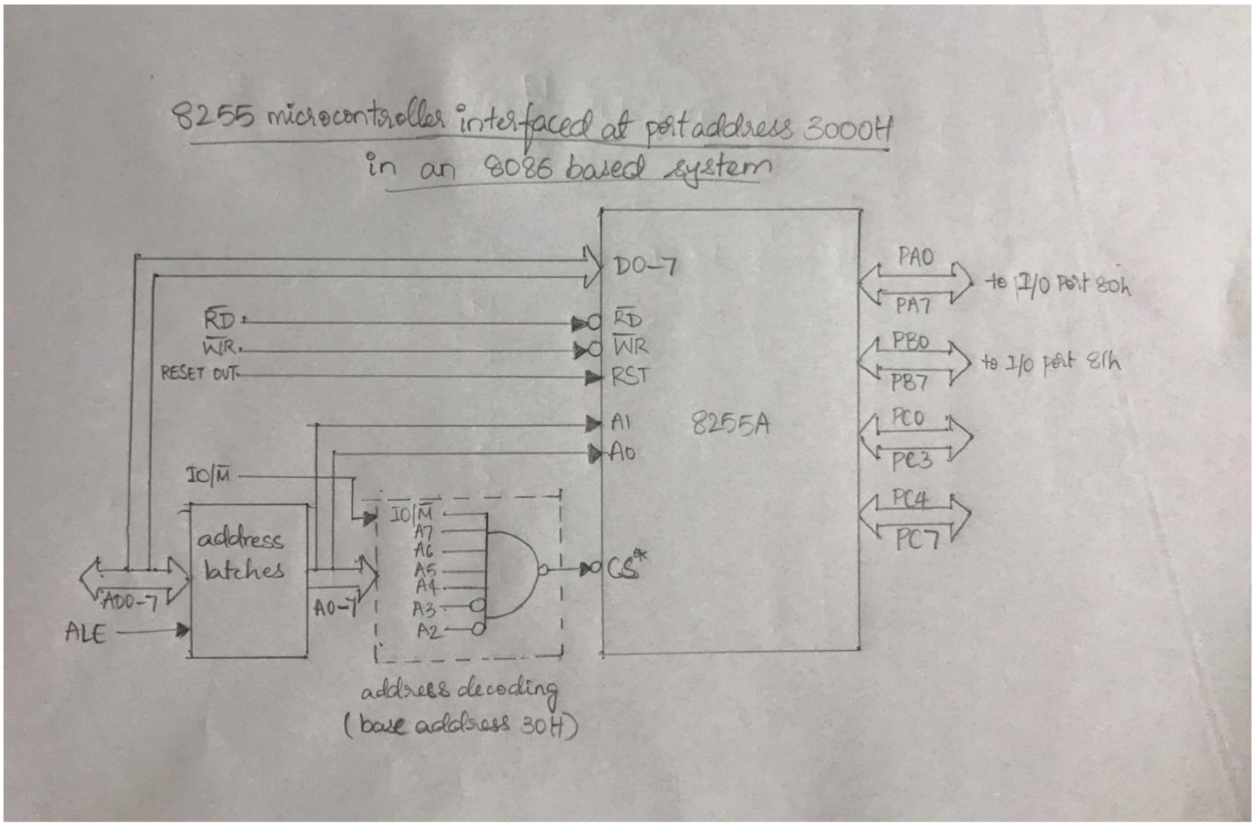
<u>Ans 6</u>:

<u>Program for monitoring Port A temperature in 8255</u>

<u>Procedure</u>:

1. Get temperature of Port A
2. Check whether the temperature has reached to 64H
3. If yes, send to port B and exit
4. Otherwise, keep checking temperature until it reaches 64H

<u>Interfacing Diagram</u>:



Code:

```
.model small
.stack 64
.data
    ; required temp is 64h
    req_temp equ 64h
    ; port A is at 80h
    portA equ 80h
    ; port B is at 81h
    portB equ 81h

.code
main proc far
begin:
    mov ax,@data
    mov ds,ax

    ; obtain temperature of port A into al register
    in al,portA

    ; compare temperature and 64h
    cmp al,req_temp

    ; if it is lesser keep comparing
    jle begin

    ; otherwise send it to port B
    out portB,al

    ; exit program
    mov ah,4ch
    int 21h

main endp
    end main
```

Ans 7:

Simulation of software interrupt using MASM

In this simulation, I tried to simulate INT 21H/AH=09h interrupt for printing a string. After invoking INT 21h, the code segment is stored into the stack. This is represented by STI in the screenshot below. The ???'s represents the requirements of any peripheral devices by the interrupt. After, the ???'s observe that Hello World! String is printed. After which the interrupt execution returns control to code segment through IRET instruction. The next instruction ready for execution in code segment is MOV AH,4CH which is displayed in screenshot below.

```
AX=096B  BX=0000  CX=0022  DX=0005  SP=0040  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=076A  IP=000B   NV UP EI PL NZ NA PO NC
076A:000B CD21          INT     21
-t

AX=096B  BX=0000  CX=0022  DX=0005  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A0   NV UP DI PL NZ NA PO NC
F000:14A0 FB            STI
-t

AX=096B  BX=0000  CX=0022  DX=0005  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A1   NV UP EI PL NZ NA PO NC
F000:14A1 FE38          ???     [BX+SI]                            DS:0000=21
-t
Hello World!
AX=096B  BX=0000  CX=0022  DX=0005  SP=003A  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=F000  IP=14A5   NV UP EI PL NZ NA PO NC
F000:14A5 CF            IRET
-t

AX=096B  BX=0000  CX=0022  DX=0005  SP=0040  BP=0000  SI=0000  DI=0000
DS=076B  ES=075A  SS=076D  CS=076A  IP=000D   NV UP EI PL NZ NA PO NC
076A:000D B44C          MOV     AH,4C
```

Code:

```
.model small

.stack 64

.data

hello db 'Hello World!$'

.code

main proc far

mov ax,@data

mov ds,ax

mov ah,09h

lea dx,hello

int 21h

mov ah,4ch

int 21h

main endp

end main
```