JUNAID · GIRKAR
B00041900057
TE COMPS A4
SEMESTER 5

18/02/22

END SEM EXAM

POA

Q1 a

ANS Von Neumann architecture was first published by John von Neumann in 1945.

His computer architecture design consists of a control unit, Arithmatic and Logic Unit (ALU), Memory Unit, Registers and Inputs / outputs.

Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory. This design is still used in most computers produced today

- CENTRAL PROCESSING UNIT (CPU)

The central processing unit (CPU) is the electronic circuit responsible for executing the instructions of a computer program.

It is sometimes referred to as the microprocessor or processor. The CPU contains the ALU, CU and a variety of registers.

- REGISTERS

Registers are high speed storage areas in the CPU. All data must be stored in a register

before it can be processed.

MAR : Memory Address Register holds the memory location of data that needs to be accessed.

MDR : Memory Data Register holds data that is being transferred to or from memory.

AC : Accumulator where intermediate arithmatic and logic results are stored

PC : Program counter which contains the address of the next instruction to be executed.

CIR : current Instruction Register contains the current instruction during processing.

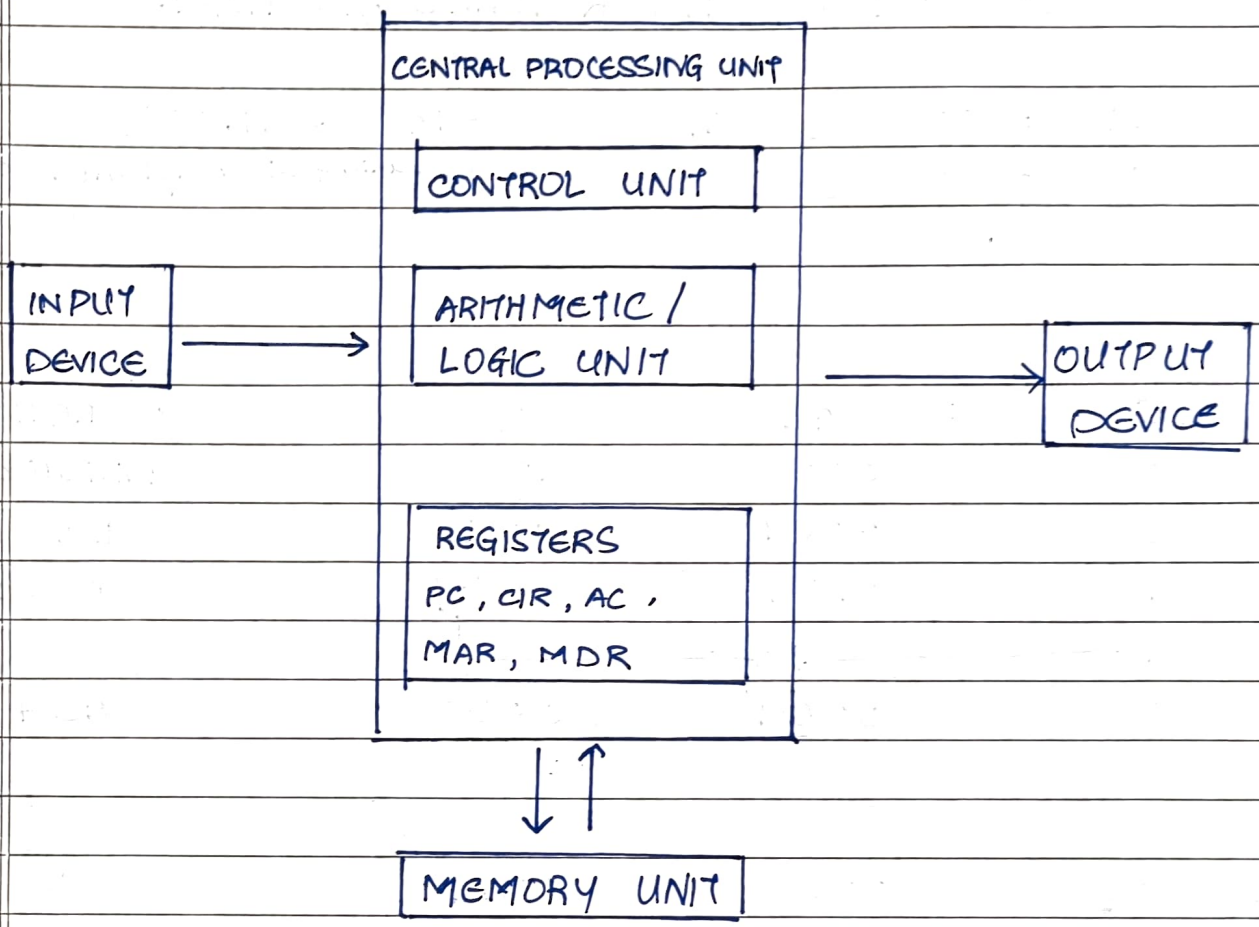
- ARITHMETIC AND LOGIC UNIT (ALU)

The ALU allows arithmetic and logic (AND, OR, etc) operations to be carried out.

- CONTROL UNIT (CU)

The control unit controls the operation of the computer's ALU, memory and input/output devices, telling them how to respond to the program's instructions it has just read and interpreted from the memory unit.

The control unit also provides the timing and control signals required by other computer components.



VON NEUMANN ARCHITECTURE

6000419057

Q1b Evaluate 8 ÷ 4 using Restoring Algorithm.

ANS $M \rightarrow (00100)_2 = (4)_0$ $\quad -M \rightarrow (11100)_2$ by taking 2's complement of M

$Q \rightarrow (1000)_2 = (8)_{10}$

count = 4

	C	AC	Q	OPERATION
	0	0000	1000	Initialization.
1)	0	0001	000□	Left shift (LS)
	1	1101	000[0]	AC - M
	1	→	→	
	0	0001	0000	AC + M.
2)	0	0010	000□	LS
	1	1110	000[0]	AC - M
	1	→	→	
	0	0010	0000	AC + M
3)	0	0100	000□	LS
	0	0000	000[1]	AC - M
	0	→	→	
4)	0	0000	0001	LS
	0	0000	001□	LS
	1	1100	001[0]	AC - M.
	1	→	→	
	0	0000	0010	AC + M.
		↓	↓	
		Remainder	Quotient	

$\therefore \quad (8)_{10} \div (4)_{10} = \text{Quotient } (2)_{10}$
Remainder $(0)_{10}$

$\therefore$ Remainder $= (0)_{10} = (0000)_2$
$\therefore$ Quotient $= (2)_{10} = (0010)_2$

6000419005J.

Q2 b

Page String : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0
Page frame size = 3
Find page fault and page hit for FIFO

7	0	1	2	0	3	0	4	2	3	0	3	1	2	0
7x	7	7	2	2	2	2x	4	4	4x	0	0	0	0x	0
	0	0	0x	0	3	3	3x	2	2	2x	2	1	1	1
		1	1	1	1x	0	0	0x	3	3	3	3x	2	2
F	F	F	F	H	F	F	F	F	F	F	H	F	F	H

Number of pages accessed = 15
Number of page hits = 3
Number of page faults = 12

$\therefore$ Hit ratio $= \frac{3}{15} = \frac{1}{5}$

Fault/Miss ratio $= \frac{12}{15} = \frac{4}{5}$

EXPLAINATION :

1) FIFO stands for First In First Out page / block replacement algorithm.

2) In this algorithm, the page that comes first in the page frames is also taken out first from the page frames.

3) So initially the memory is empty and has 3 page frames.

4) When 7 page was accessed, the page fault occured since there was no page no 7 in the memory. Hence it is then brought from the secondary memory.

5) Now similarly the pages no 0 and 1 are also brought from the secondary memory as the corresponding pages were absent.

6) Then as soon as the page 2 came, the memory became full so we need to replace it with page 7 that had came first into the page frames.

7) After this, page 0 is accessed and it is found in the memory itself. Hence page hit occured and it is given to the processor directly.

8) And so on this can be implemented for all the remaining pages in a similar way.

9) PAGE HIT – The required page is found in the main memory
PAGE FAULT – The required page is not found in the main memory and hence has to be brought from secondary memor

$$\text{HIT RATIO} = \frac{\text{Pages hit}}{\text{Total pages}} \; ; \; \text{Miss RATIO} = \frac{\text{Page faults}}{\text{Total pages}}$$

6000419005F

Q3 a

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 and data is an immediate data or an address

Types of addressing modes are :-

1. IMMEDIATE ADDRESSING MODE

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode

EXAMPLE : MOV BX, 1492H, ADD CX, 1122H, MOV AL, FFH

2. REGISTER ADDRESSING MODE

It means that the register is the source of an operand for an instruction.

```
EXAMPLE :- MOV CX,AX ; copies contents of 16-bit register AX into
                     ; 16 bit register CX
           ADD BX,AX
```

3. DIRECT ADDRESSING MODE :

The addressing mode in which the effective address of the memory location is directly written

in the instruction.

EXAMPLE : MOV AX, [1592] H, MOV AL, [0300] H

4. REGISTER INDIRECT ADDRESSING MODE :

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers : BP, BX, DI & SI

EXAMPLE : MOV AX, [BX] ; suppose the register BX contains 4895H
; then the contents of 4895 H are moved to AX
ADD CX, [BX]

5. BASED ADDRESSING MODE

In this addressing mode, the offset address of the operand is given by the sum of contents of BX / BP registers and 8-bit / 16-bit displacement.

EXAMPLE : MOV DX, [BX+04], ADD CL, [BX+08]

6. INDEXED ADDRESSING MODE

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8 bit / 16-bit displacements

EXAMPLE : MOV BX, [SI+16], ADD AL, [DI+16]

6000419005?

7. BASED-INDEX ADDRESSING MODE

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an index register.

EXAMPLE : ADD CX, [AX+SI], MOV AX, [AX+DI]

8. BASED-INDEX WITH DISPLACEMENT MODE

In this addressing mode, the operands offset is computed by adding the base register contents. An index register contents and 8-bit or 16-bit displacement.

EXAMPLE : MOV AX, [BX+DI+08], ADD CX, [BX+SI+16].

6000419005 9

Q4 a Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an ISR (Interrupt Service Routine) which is a short program to instruct the microprocessor on how to handle the interrupt.

Types of interrupts in a 8086 microprocessor :-

- Hardware interrupt → Maskable interrupt
  → Non-Maskable interrupt
- Software Interrupt.

SOFTWARE INTERRUPT :

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes:-

- INT - Interrupt instruction with type number.

  It is a 2-byte instruction. First byte provides the op-code and second byte provides the interrupt type numer. There are 256 interrupt types under this group.

  The first pointers are dedicated interrupt pointers. i.e :

  - TYPE 0 interrupt represents division by zero.

- TYPE 1 interrupt represents single-step execution during the debugging of a program.
- TYPE 2 interrupt represents non-maskable interrupt
- TYPE 3 interrupt represents break-point interrupt
- TYPE 4 interrupt represents overflow interrupt

Type 5 - 31 : Reserved for other advanced microprocessors
Type 32 - 255 : Available for hardware & software interrupts

- INT 3 - Break Point interrupt instruction.
  It is a 1-byte instruction having op-code is CCH. These instructions are inserted into the program so that when processor reaches there, it stops and follows break-point procedure

- INTO - Interrupt on overflow instruction.
  It is a 1-byte instruction and their mnemonic INTO. The op-code is CEH. It is active only when the overflow flag is set to 1 and branches to the interrupt handler whose type is 4.

EXAMPLES

- ~~Request to~~
- A cout of cin statement would generate a ~~syst~~ software interrupt because it would make a system call to print something.
- A fork() statement in linux would create a software interrupt because it would create a new process

8

8/4 6

• Push instruction

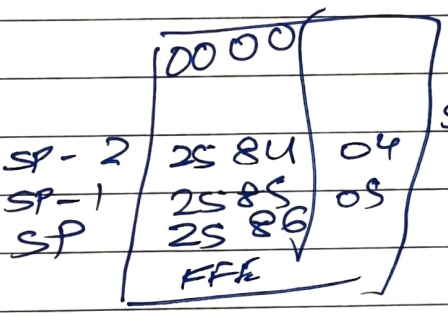
This instruction is used to transfer the contents of the specified register are copied onto the stack by decreasing stack pointer by 1 and higher order register is copied on that location

Mneumonic: Push source
operation: SP = SP − 2
SS : [SR]

e.g: Push AX

	AH	AL
AX	05	04

SP [2586]



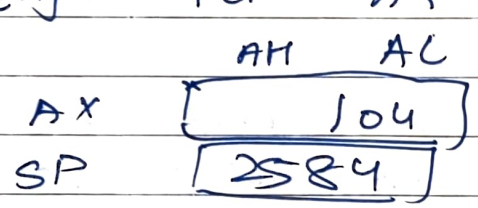
After push SP value is 2584.

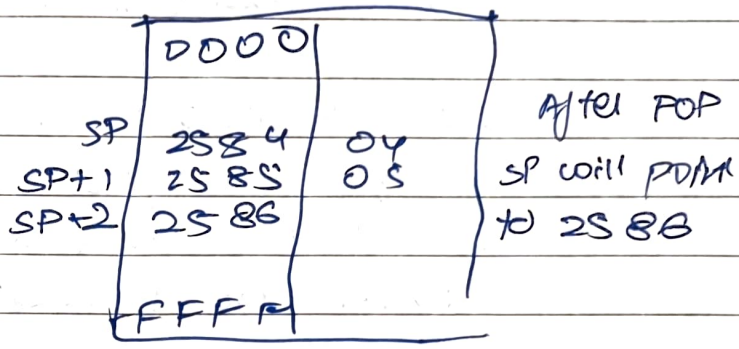
• POP operation.

This instruction is used to transfer the contents of the stack register by increasing stack pointer by 1 and contents of memory location are copied in higher order register.

Mneumonic: POP ~~operation~~ Destination
operation: SP = SP + 2
SS = [SI].

E.g: POP AX





9

GOOG190057

Q6 a

ANS

Pipelining is a process of arrangement of hardware elements of the CPU such that its overall performance is increased. Simultaneous execution of more than one instruction takes place in a pipelined processor.

DESIGN OF A BASIC PIPELINE

- In a pipelined processor, a pipeline has two ends, the input end and the output end. Between these ends, there are multiple stages/segments such that the output of one stage is connected to input of next stage and each stage performs a specific operation.
- Interface registers are used to hold the intermediate output between two stages. These interface registers are also called latch or buffer.
- All the stages in the pipeline along with the interface registers are controlled by a common clock.

PIPELINE STAGES

RISC processor has 5 stages instruction pipeline to execute all the instructions in the RISC instruction set. Following are the 5 stages of RISC pipeline with their respective operations :-

- STAGE 1 (Instruction fetch)
  In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.

- STAGE 2 (Instruction Decode)
  In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.

- STAGE 3 (Instruction execute)
  In this stage, ALU operations are performed.

- STAGE 4 (Memory Access)
  In this stage, memory operands are read and written to/from the memory that is present in the instruction.

- STAGE 5 (Write Back)
  In this stage, computed / fetched value is written back to the register present in the instructions.

EXAMPLE :-

Suppose we have the following 3 lines of code :

R1 <— [1]
R2 <— [2]
R3 <— [3]

G0004190057.

In the code above, we are performing three load types. In one line, we are storing the address 1 to R1, line 2, we are storing address of 2 to R2 and finally in line 3, we are storing the address. 3 to R3.

The RISC pipeline will look like this :-

CODE	INSTRUCTION LINE	STEP 1	STEP 2	STEP 3	STEP 4	STEP 5	STEP 6	STEP 7
R1 <- [1]	1	Fetch	Decode	Execute	Memory	write		
R2 <- [2]	2		Fetch	Decode	Execute	Memory	Write	
R3 <- [3]	3			Fetch	Decode	Execute	Memory	Write.

The above diagram shows how the three line ~~load~~ code of all load types will execute. In step 1, the first line will execute the first step, which fetches. Then in step 2, while line 1 is in the decode phase, line 2 will start fetching, and so on. The 3 lines of code will go through seven steps in order to complete all RISC pipeline for all three lines.