Past Ques soln set

Arpan Adhikari
Roll No. 231309

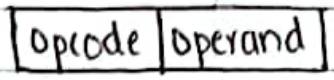
CA

2024 Fall

1.a) The manner in which each address field specifies the memory location is called addressing mode.

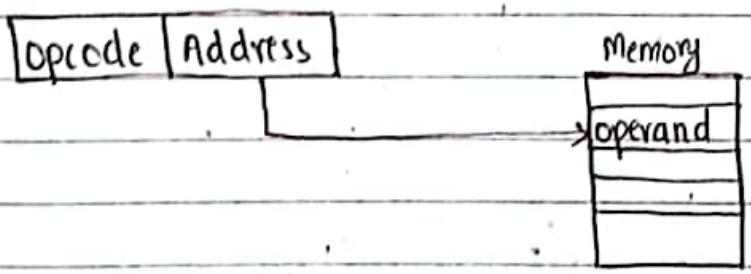
Types:

(i) Immediate Addressing Mode
Operand present on instruction itself.



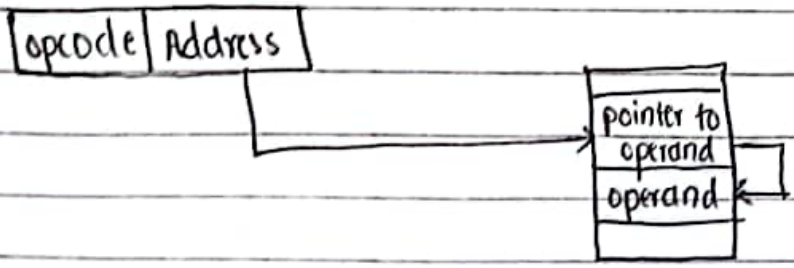
E.A. = -
Operand = A (Address field)

(ii) Direct Addressing mode



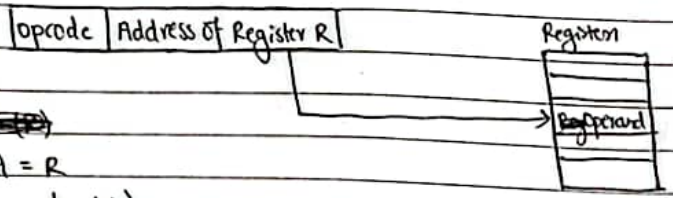
E.A = A
Operand = (A)

(iii) Indirect Addressing mode



E.A = (A)
Operand = ((A))

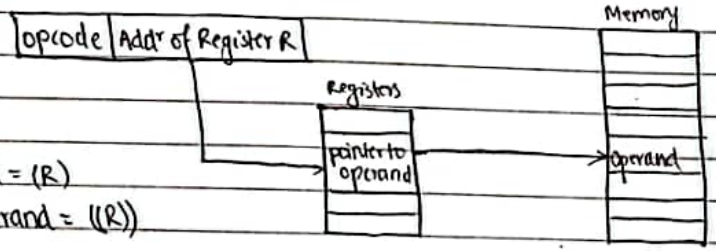
(iv) Register Addressing Mode



EA = R

Operand = (R)

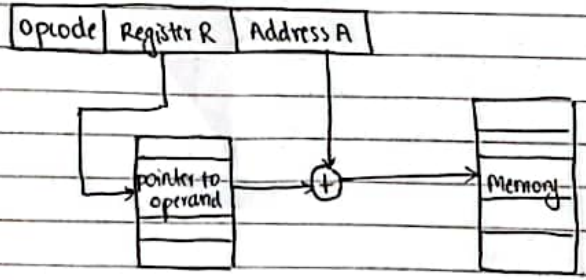
(v) Register Indirect Addressing Mode



EA = (R)

Operand = ((R))

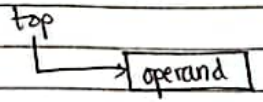
(vi) Displacement Addressing Mode



EA = (R) + A

Operand = ((R) + A)

(vii) Stack Addressing Mode



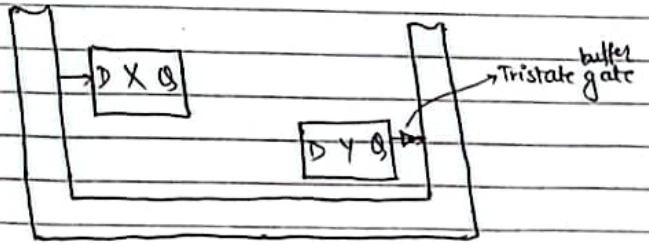
Q.1b)

Data transfer micro-operations in register involves moving binary information from one register to another within CPU. These operations are fundamental for processing instructions and managing data flow inside a computer.

Two techniques of data transfer:

1. Direct (Point-to-point) transfer
   - For n registers, $n(n-1)$ lines.
   - If K-bit transfer needed between two register K lines.
     Altogether, $K \times n(n-1)$

2. Bus Implementation



- All register connect output by using tri-state buffer gate or multiplexer.
- Only one register can transmit data at one time.
  - → Wiring efficiency
  - → Scalability
  - → Bus contentention : only one register use bus at one time.
  - → Flexibility

2a)

Internal structure of processor :

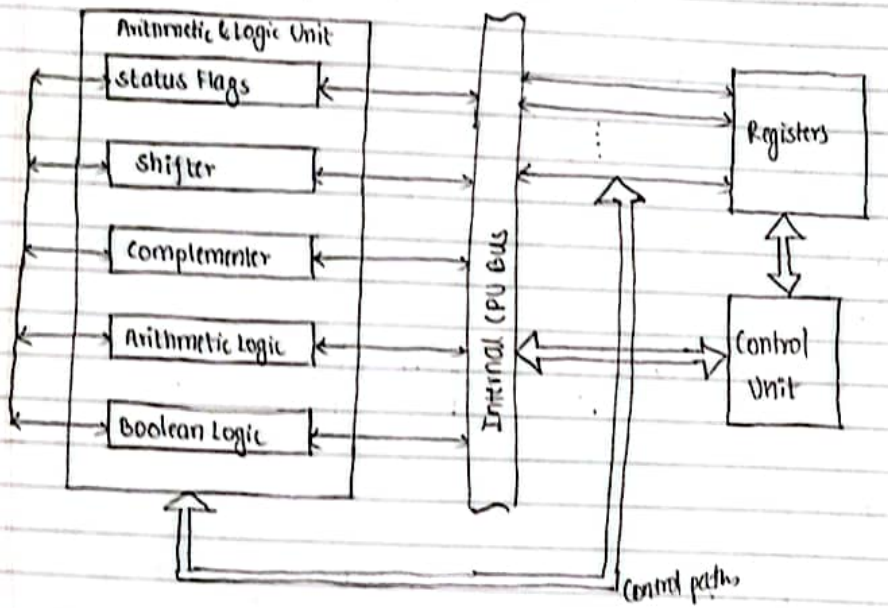


Fig. Internal structure of CPU

1. Arithmetic and Logic Unit (ALU)
   - Performs all arithmetic and logical operations.
   - Key sub-components :
     # Status Flags : Store results of operation, such as zero, overflow flags
     # shifter : Handles bit-wise shifting or rotation operations.
     # complementer : Performs bitwise complement operations.
     # Arithmetic Logic : Executes arithmetic operations
     # Boolean Logic : Executes boolean operations

2. Registers
   - Small, fast storage units within CPU used to temporarily hold data, instructions, address and results.
   - Directly accessible by ALU & Control Unit for fast read/ write operations.

3. Internal CPU Bus
   - Set of parallel lines connecting major components like ALU and registers
   - Facilitates data transfer within CPU

4. Control Unit
   - Generates control signal that coordinate & manage all activities inside CPU.
   - Decides which operation to perform, when to move data & what components to engage.

Design principle of modern system :
1. Efficient memory utilization
2. Pipelining
3. Parallel processing
4. Uses of caches at different levels
5. Multicore system

2024 Spring:

1b) $X = (M * N) + (P * Q)$

(i) Three Address Instructions

Each instruction can specify three addresses : two for source operands and one for destination.

For given qsn,

```
MUL R1,M,N      ; R1 = M * N
MUL R2,P,Q      ; R2 = P * Q
ADD X,R1,R2     ; X = R1 + R2
```

(ii) Two Address Instructions

Each instruction can specify two addresses : typically, the destination is also one of the sources.

(iii) One Address Instruction

A single explicit address is used ; accumulator (AC) is implied as the other operand.

(iv) Zero-Address Instruction

Use stack.

1b) $X = (M * N) + (P * Q)$

(i) 3-Address

```
MUL R1,M,N
MUL R2,P,Q
ADD X,R1,R2
```

(ii) 2-Address :

Instruction	comment
~~MUL M,N~~	
MOV R1,M	; R1 = M
MUL R1,N	; R1 = R1 * N
MOV R2,P	; R2 = P
MUL R2,Q	; R2 = R2 * Q
ADD R1,R2	; R1 = R1 + R2
MOV X,R1	; X = R1

(iii) 1-Address:

Instruction	comment	
LOAD M	; AC = M	
MUL N	; AC = AC * N	
STORE TEMP1	; TEMP1 = AC	
LOAD P	; AC = P	
MUL Q	; AC = AC * Q	
ADD TEMP1	; AC = AC + TEMP1	(Uses a memory location (TEMP1) for storing results)
STORE X	; X = AC	

(iv) 0-Address

Instruction	Stack comment
PUSH M	; ~~stack~~ M
PUSH N	; ~~stack~~ M,N
MUL	(M*N)
PUSH P	(M*N), P
PUSH Q	(M*N), P, Q
MUL	(M*N), (P*Q)

ADD (M×N)x(P×Q)

POP X Empty stack ; X=(result)

1c	2b	3d	4d	5c	6a	7d	8a	9b	10d
11d	12b	13b	14c	15d	16a	17d	18d	19b	20c
21c	22a	23b	24a	25b	26c	27d	28b	29c	30b
31c	32c	33d	34b	35b	36b	37d	38b	39b	40d

Rough

Q2a)

Structure of VHDL Programming :

The structure of VHDL program consists of several fundamental sections that organize how digital circuits are described & implemented

1. Library Declarations
- Libraries make standard types, functions, and components available for use.

For e.g

```
library IEEE;
use IEEE.STD_LOGIC_ARITH.ALL;
```

2. Entity Declaration
- The entity block describes how circuit connects to the outside world.
- Defines circuit interface (input/output)
- It is much like declaring pin diagram of IC.
- Declares name of entity and port, specifying their direction (IN, OUT, INOUT) and type (STD_LOGIC etc)

Syntax e.g.:

```
ENTITY entity_name IS
  PORT (
    port1 : IN STD_LOGIC;
    port2 : OUT STD_LOGIC
    );
END entity_name;
```

3. Architecture Section
- Describes the internal operation or behavior of the circuit defined by the entity.
- May contain signal declarations, concurrent statements, component instantiations and so on.

e.g.

```
architecture Behavioral of entity_name is
  -- optional declarations (signals, constants, components)
BEGIN
  -- concurrent statements (logic, assignments, processes etc.)
END Behavioral;
```

2b)

Register types:

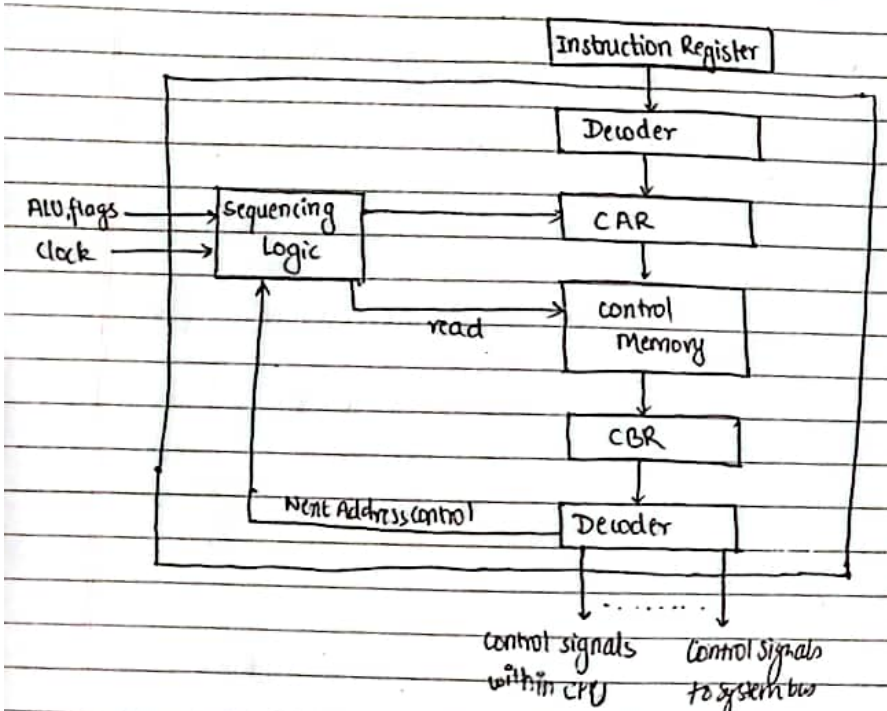
1. User Visible Registers
   (i) Address register
   (ii) Data register
   (iii) General purpose register
   (iv) Convolution code

2. Status & control registers
   (i) MAR
   (ii) MBR
   (iii) IR
   (iv) PC

Q3a)

Control memory is a type of memory located within control unit that stores microprograms composed of microinstructions.

1- In Microprogrammed control unit, control memory is used as Read Only Memory (ROM), where all control information is stored permanently.



Horizontal μ-instruction

- Every bit of microinstruction correspond to control signal.

Difference bet^n Horizontal & vertical microinstruction:

Aspect	Horizontal Control Unit	Vertical control Unit
Control word width	Longer (wide, many bits)	Shorter (narrower, fewer bits)
Control signal specification	Explicit (One bit per control signal)	Encoded (fields decoded to control signals)
Parallelism	High (many operations in parallel)	Low/sequential (few operations at a time)
Hardware Requirment	No extra hardware needed for decoding	Requires decoders to expand control signals.
Flexibility	More flexible, direct hardware control	Less flexible, more like regular instruction
Speed	Faster (direct signal activation)	slower (decoding introduces delay)
Memory Requirment	High (large word, more storage needed)	Low (compact, less storage needed)
Micro-instruction Type	Each bit controls a separate function.	Encoded field represents set of operations

16 3 2 1
0 1 0 0 0 1
1 1 1 1 1 0
0 0 0 1 0 0
0 0 0 0 1 0

1 1 1 0 1 1
+ 1
1 1 1 1 0 0

1 1 1 1 1 0
0 0 0 1 0 0
0 0 0 0 1 0

Q3b) 17 * 4 using Booth's

010001 * 000100
Q M

A	Q	Q-1	M	Operation
000000	010001	0	000100	
111100 / ~~000000~~	010001	0	000100	A←A-M
111110	001000	1	000100	Ashr
000010	001000	1	000100	A←A+M
000001	000100	0	000100	Ashr
000000	100010	0	000100	Ashr
000000	010001	0	000100	Ashr
111100	010001	0	000100	A←A-M
111110	001000	1	000100	Ashr
000010	001000	1	000100	A←A+M
000001	000100	0	000100	Ashr

Result is stored in AQ

000001 000100
64 32 16 8 4 2 1

17 * 4 = (68)

Ans

Q4a)

The pipelining hazards are:

1. Resource conflict
It is caused when two segments access memory at same time. These conflicts can be removed by using separate memories for data and instructions.

2. Data conflict
It arises when an instruction depends on the result of previous one.
Resolving:
(i) Hardware interlock
(ii) Operand forwarding
(iii) Delayed Load

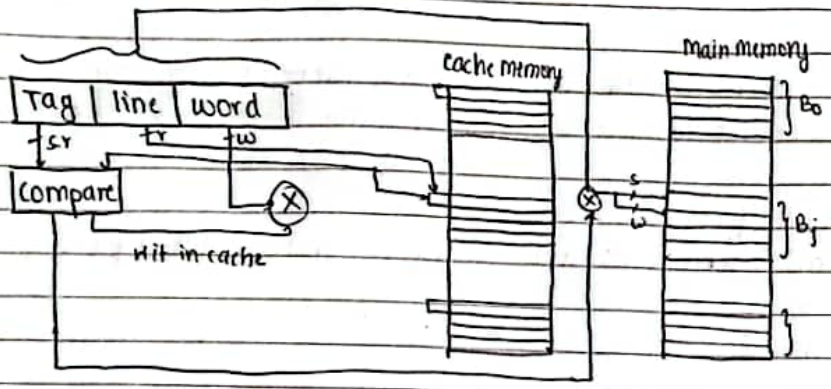
3. Branch conflict
It arises from branch and other instructions that change the value of PC.
Resolutions:
(i) Branch Prediction
(ii) Delayed Branch
(iii) Loop Buffer
(iv) Prefetch ~~Instruct~~ target instruction
(v) Branch target Buffer

Q4b)

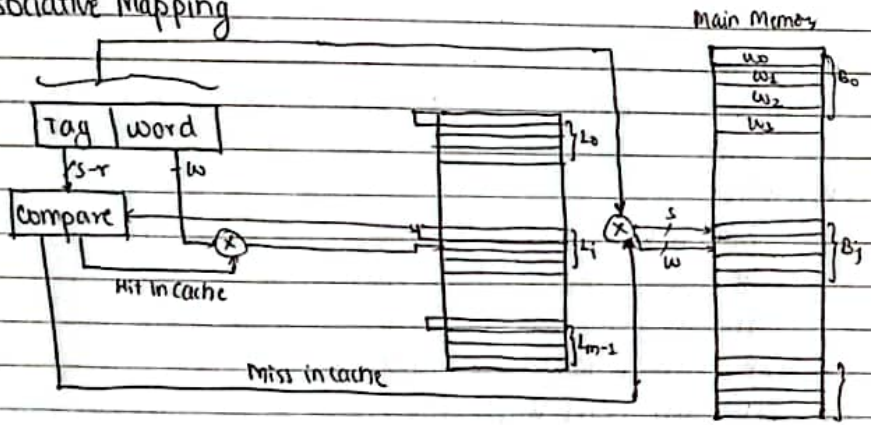
The transformation of data from main memory to cache memory is called mapping process.

(i) Direct Mapping

Maps each block of main memory into only one possible cache line.

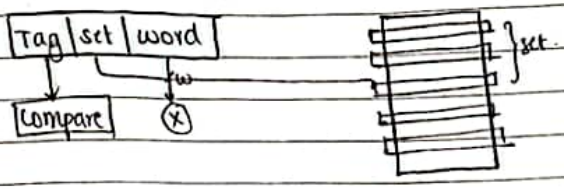


(ii) Associative Mapping



PTO

(iii) Set Associative Mapping



Q5a)

Memory storage devices:

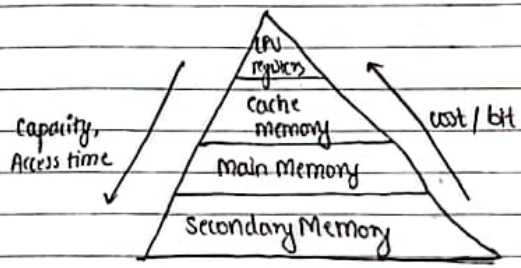


Fig. Memory hierarchy.

1. Registers

These are small, ultra-fast memory locations inside the CPU that hold data and instructions currently needed for processing. They have the quickest access time but minimal storage capacity.

2. Cache Memory

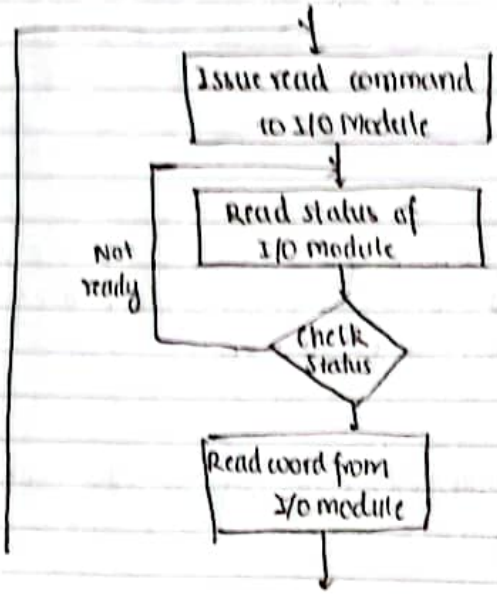
Located close to CPU, cache stores frequently accessed data or instructions to speed up processing.

3. Main Memory
This is the central storage area the CPU can access directly. It stores program data and instructions currently in use and consists of both RAM and ROM.

4. Auxiliary Memory
These storage devices, such as hard disk drives (HDD), solid state drives (SSD) and optical disks, provide large, non-volatile storage for data not currently in use.

5. Associative Memory
Special Memory allowing data retrieval by content rather than address, enabling parallel searches.

(Q5b)



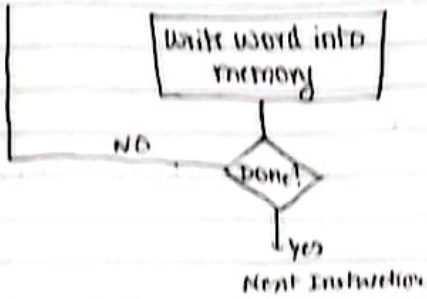
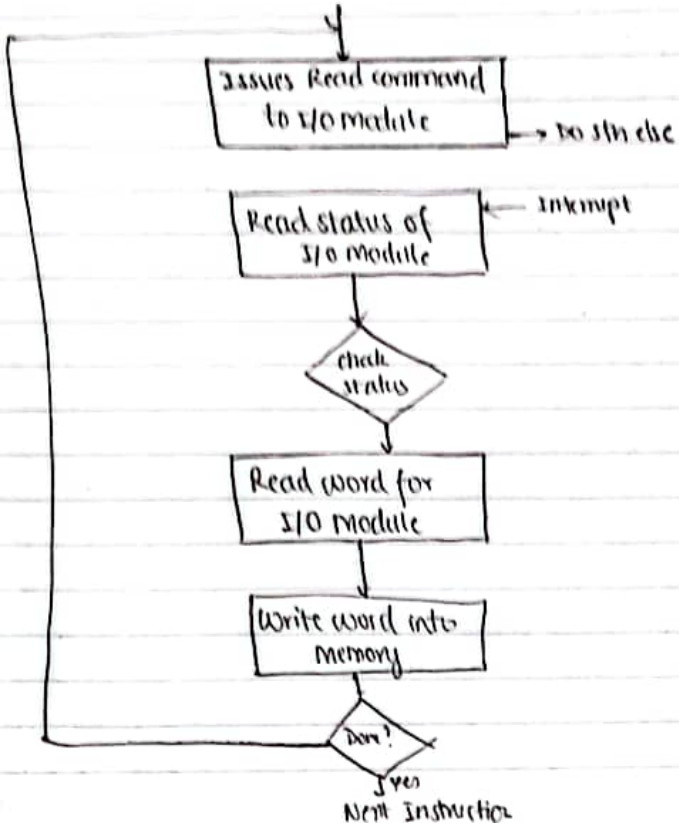
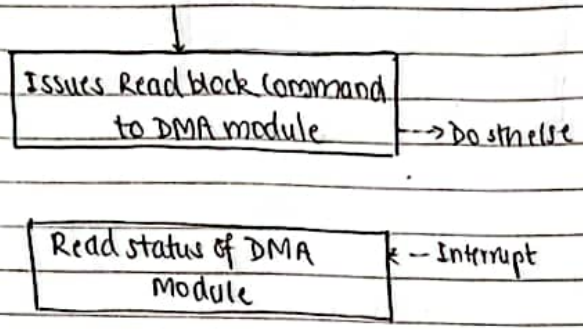


Fig Programmed I/O

Interrupt driven I/O:



DMA:



Q6a)

Flynn's Classification:

(i) Single Instruction Single Data (SISD)
(ii) Multiple Instruction Multiple Data (MIMD)
(iii) Single Instruction Multiple Data (SIMD)
(iv) Multiple Instruction Single Data (MISD)

Q6b)

Hardware related performance issues:

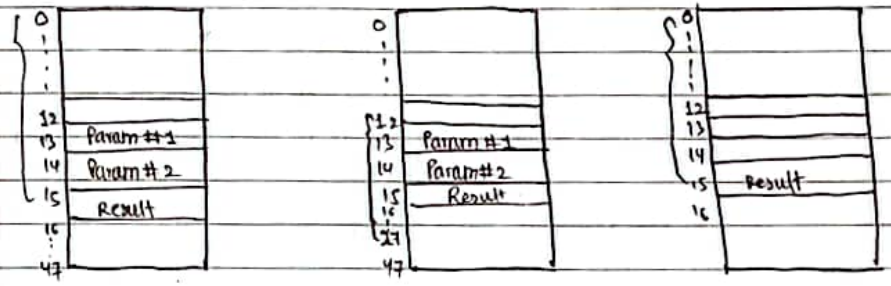
(i) Pipeline limitations and stalls
(ii) Resource contention
(iii) Cache coherency & Latency
(iv) Memory bandwidth & Latency
(v) Power consumption & Thermal Limits
(vi) Scalability Limits

Software Related Performance issues:

1. Workload Parallelization
2. Thread and Process Management Overheads
3. Load Balancing
4. Cache coherency Overhead
5. Scalability of Algorithms
6. Software compatibility & Legacy Applications
7. Operating System support

Q7b)

Register windowing & Renaming



a) First window active only 1st window has valid data

b) Second window is active 1st two windows have valid data

c) 1st window is active Only 1st window has valid data

Q7c) GPU v/s TPU

Feature	GPU	TPU
Developer	NVIDIA, AMD, Others	Google
Purpose	Originally for graphics; now for parallel tasks & AI	Purpose built for AI, especially tensor operations
Architecture	Thousands of cores for parallelism, general purpose use	custom ASIC, optimized for matrix/tensor ops
Flexibility	Highly flexible; supports many frameworks	Limited; mainly optimized for TensorFlow and JAX
Performance	Excellent for a wide range of AI models	Superior speed for large scale tensor computations in AI
Energy Efficiency	Good, but higher power use under load	Higher efficiency for deep learning workloads
Accessibility	Available for desktops, data centers, cloud	Cloud based not sold individually