Practice Set- 2

Question 1: Differentiate between the control path and the data path in terms of their role in a pipelined processor. Explain how both interact during the execution of a load instruction.

Solution 1 :

- **Data Path:** Handles data transfer and computation. It includes components like ALU, registers, buses, and memory.
- **Control Path:** Manages the coordination and signaling required to execute instructions properly. It includes the control unit (FSM or microprogrammed logic).

During a load instruction:

- In the **F stage**, the instruction is fetched using the PC.

- The **control path** sets up signals to interpret it as a load.
- In the **D stage**, source registers are read (address computation).
- In the **EX stage**, the address is computed by the ALU.
- The **MA stage** accesses memory using the computed address (data path).
- The **RW stage** writes data into the destination register (data path), as directed by control signals.

Question 2: Consider the following instruction sequence. Identify the type of hazard present and explain a solution.

1. add r1, r2, r3
2. sub r4, r1, r5
3. and r6, r1, r7

Solution 2: Hazard: Read-after-write (RAW) data hazard – Instructions 2 and 3 need the result of instruction 1. These hazards can be mitigated using:

- **Forwarding (bypassing):** Use the output of EX/MEM stage directly instead of waiting for WB.
- **Stalling:** Introduce NOPs (do nothing) to delay instructions 2 and 3 until r1 is updated.
- **Reordering:** If possible, insert independent instructions between them.

Question 3: Given a 5-stage pipeline (F, D, EX, MA, WB), and the following code:Draw the pipeline diagram assuming no forwarding and compute total cycles.

1. lw r1, 0(r2)
2. add r3, r1, r4
3. sub r5, r3, r6

Solution 3:

Cycle	1	2	3	4	5	6	7	8	9
lw	F	D	EX	MA	WB				
add			F	D*	-	EX	MA	WB	
sub					F	D	EX	MA	WB

Total cycles: 9

Question 4: What is the pipeline execution order of the following RISC-V instructions assuming there are no hazards and the pipeline is ideal (no stalls, no forwarding delays).Each instruction goes through the 5 pipeline stages: **F (Fetch), D (Decode), E (Execute), M (Memory), W (Writeback)**

1. add x1, x2, x3
2. sub x4, x5, x6
3. and x7, x8, x9
4. or x10, x11, x12
5. xor x13, x14, x15
6. slt x16, x17, x18

Solution 4:

Cycle	1	2	3	4	5	6	7	8	9	10
add	F	D	E	M	W					
sub		F	D	E	M	W				
and			F	D	E	M	W			
or				F	D	E	M	W		
xor					F	D	E	M	W	
slt						F	D	E	M	W

Total Number of Cycles : 10

Question 5: You have a single-cycle processor with cycle time 20ns, and a pipelined processor with 5 stages, each of 4ns delay.

- Compute speedup if there are no stalls.
- Now assume 30% instructions stall for 1 cycle. What is the new speedup?
- What is the impact if you reduce the pipeline stages to 4 stages, each taking 5.2ns?

Solution 5:

- Single-cycle time per instruction: 20ns
  Pipelined time per instruction (ideal): 1 instruction per 4ns
  Speedup= 20\4 = 5

- Effective CPI in pipeline = 1 + 0.3 × 1 = 1.3
  Effective time = 1.3 × 4 = 5.2ns
  Speedup = 20/5.2≈3.85

- New clock = 5.2ns, CPI = 1 (ideal)
  Speedup (vs single-cycle)=20/5.2≈3.85

Question6 : You have a processor with **clock = 2 MHz**, non-pipelined. A program takes 300 instructions. Now, it is upgraded to a 5-stage pipeline with stage delays: 180ns, 200ns, 150ns, 170ns, 160ns. Register delay = 10ns.

(a) Calculate original execution time.
(b) Compute new clock cycle.
(c) Compute execution time for 300 instructions (no hazards).
(d) Compute speedup.

Solution 6 :

(a) Clock Period = 1/ 2×10^6=500 ns
Time = 300 × 500 = 150,000 ns

(b) Pipeline clock = max(stage delay) + register delay = 200 + 10 = 210 ns

(c) Time = (number of instructions + 4) × clock = 304 × 210 = 63,840 ns

(d) Speedup = 150000/ 63840≈2.35

Question 7 : A pipelined processor has 5 stages. The clock cycle time is 2 ns. A benchmark executes 100 instructions. During execution:

- 10% of instructions cause a **1-cycle stall** (due to data hazards).
- 5% of instructions are **branches**, each incurring **2-cycle penalty** on misprediction.
- Misprediction rate is 40%.

Compute the effective CPI.

Solution 7:

- **Ideal CPI** = 1 (in perfect pipeline)
- **Stall penalty (data hazard)** = 10% × 1 = 0.10
- **Branch penalty** = 5% × 40% × 2 = 0.05 × 0.4 × 2 = 0.04

Effective CPI = 1+0.10+0.04 = 1.14

Question 8 : Two processors, **Alpha** and **Beta**, implement the same ISA. Alpha has a **clock rate of 3 GHz**, and Beta has a **clock rate of 2 GHz**. A benchmark program executes:

Optimization	Instructions on Alpha	CPI (Alpha)	Instructions on Beta	CPI (Beta)
Opt-1	10000	1.2	12000	1.0
Opt-2	8000	1.5	9000	1.1

a) Compute the execution time for each optimization on both processors.

(b) Which processor is faster for Opt-1 and Opt-2?

Solution 8:

Execution Time= Instruction Count×CPI / Clock Rate

Opt-1:

- Alpha: 10000×1.2 / 3×10^9 = 4.00 µs
- Beta: 12000×1.0 / 2×10^9 = 6.00 µs

Opt-2:

- Alpha: 8000×1.5 / 3×10^9 = 4.00 µs
- Beta: 9000×1.1 / 2×10^9 = 4.95 µs

Alpha is faster in both cases.

Question 9 : Define **CPI**, **Instruction Count**, and **Clock Rate**. How are they used to compute processor performance?

Solution9:

- **CPI (Cycles Per Instruction):** Average number of clock cycles needed per instruction.
- **Instruction Count:** Total number of instructions executed.
- **Clock Rate:** Number of cycles per second (Hz).

CPU Time = Instruction Count × CPI / Clock Rate

Processor performance is better if:

- Instruction count is low,
- CPI is low,
- Clock rate is high.

Question 10: Classify the following hazards in pipeline architectures: RAW, WAR, WAW.

Solution 10:

Hazard Type	Description
RAW	Reading a register before it's written by an earlier instruction
WAR	Writing a register before it's read by an earlier instruction
WAW	Writing a register before another write completes

Question 11 : Define temporal locality and spatial locality. Give one example of each from a real program.

Solution 11:

- **Temporal locality**: If a memory location is accessed, it's likely to be accessed again soon (e.g., loop counter variable).
- **Spatial locality**: If a memory location is accessed, nearby addresses are likely to be accessed soon (e.g., array traversal).

Question 12: A non-pipelined processor takes 500 ns per instruction. A pipelined version splits it into 5 stages, each of 120 ns (including register delay).

(a) What is the ideal speedup with pipelining?

(b) If the pipeline has a 20% stall rate, what is actual speedup?

Solution 12 :

(a) Ideal:

- Non-pipelined time: 500 ns/instruction
- Pipelined time: 120 ns/instruction (1 instruction per cycle after fill)

Speedup = 500 / 120 ≈ 4.17

(b) With stalls:

- Effective CPI = 1 + 0.20 = 1.2
- New time per instruction = 1.2 × 120 = 144 ns

Speedup = 500 / 144 ≈ 3.47

Question 13: Explain the need for callee-saved and caller-saved registers. What would go wrong if this convention isn't followed?

Solution 13: callee-saved and caller-saved registers are needed because **:**

- Functions often use registers for local computations.
- If multiple functions call each other, values in registers must be preserved where required.

Without the convention:

- Registers used by a callee could overwrite values expected by the caller.
- Unpredictable behavior, especially when functions are nested.

Caller-saved registers must be saved by the caller before the call if their values are needed later. Callee-saved registers must be saved by the callee only if they are modified.

Question 14 : A function foo( ) calls another function bar( ) inside it. You are using a processor that follows the convention:

- Registers s0-s1 are callee-saved
- Registers s2-s11 are caller-saved

(a) Explain what happens if foo( ) uses s5 and wants to preserve its value across the call to bar().

(b) What happens if bar ( ) uses s1 ?

Solution 14 :

(a) s5 is caller-saved →foo( ) must save s5 to the stack before calling bar( ) and restore it afterward.

(b) s1 is callee-saved →bar ( ) must save s1 on entry and restore it before returning if it modifies

Question 15: True or False

For a 5-stage pipeline:

1. One instruction completes every cycle after the pipeline is full.
2. A data hazard always requires a stall.

3. NOP is a special instruction that affects pipeline control signals.

4. Branch misprediction penalty is zero if the prediction is correct.

Solution 15:

1**.** In a 5-stage pipeline, one instruction completes every cycle after the pipeline is full - **True**

Explanation: Once the pipeline is fully filled (after the first 5 cycles), each cycle completes one instruction due to overlapping execution.

2. A data hazard always requires a stall - **False**

Explanation: If data forwarding is supported, many data hazards (like read-after-write) can be resolved without stalling the pipeline.

3. NOP is a special instruction that affects pipeline control signals - **False**

Explanation: NOP (No Operation) is a dummy instruction; it **does not alter control logic or register values**. It is often used to **occupy a cycle** during stalls or flushes.

4. Branch misprediction penalty is zero if the prediction is correct - **True**

Explanation: If a branch is **predicted correctly**, there is **no need to flush** instructions or insert bubbles. Hence, the **penalty = 0 cycles**.

Question 16

You're designing a smart thermostat with a temperature sensor that logs data every minute. You use a memory chip with **512 memory locations**, each storing **16 bits**.

- What size of data bus and address bus do you need?
- Can this memory store an entire day's worth of data?

Solution 16:

- **Address bus:** $\log_2(512)$ = **9 bits**
- **Data bus:** 16 bits (that's the word size)
- One day = 24 × 60 = **1440 readings**
- Available = 512 readings → **Not enough**

You'd need **three times more memory**, or compress/store fewer bits per reading.

Question 17:

You are analyzing the memory performance of a processor that includes a cache system and the system has the following memory access characteristics:

- **Cache hit time**: 2 clock cycles
- **Cache miss penalty**: 80 clock cycles
- **Miss rate**: 8% = 0.08.

Calculate the **average memory access time (AMAT)** in clock cycles.

Solution 17:

Using the formula:

AMAT=Phit*Thit + Pmiss*Tmiss

AMAT=(0.92×2)+(0.08×80)=1.84+6.4=8.24 clock cycles

Question 18:

You're optimizing the performance of a processor used in an embedded system for a smart camera. The processor is connected to a byte-addressable memory with a total size of **128 KB**. To enhance performance, a **2-way set-associative cache** is implemented with the following configuration:

- **Total cache size**: 16 KB
- **Block size**: 256 bytes
- **Cache associativity**: 2-way set associative
- **Main memory is byte-addressable**

For this memory architecture, determine how many bits from a memory address are used for the following:

1. **Block Offset** (within a cache block)
2. **Set Index** (to identify the cache set)
3. **Tag** (to verify if a block is in the cache)

Solution 18:

1. Total addressable memory = 128 KB
   Since memory is byte-addressable:
   128 KB = 2^17

So, memory addresses are 17 bits.

2. Cache block size = 256 bytes
   Block Offset = log2(256) = 8 bits

3. Cache total size = 16 KB
   Block size = 256 bytes
   Number of blocks in cache = 16 KB / 256 bytes = 64 blocks

This is a 2-way set associative cache:
Number of sets = 64 blocks / 2 = 32 sets
Set Index = log2(32) = 5 bits

4. Tag bits = Total address bits - (Set Index + Block Offset)
   Tag = 17 - (5 + 8) = 4 bits

Question 19:

You are designing a cache system for a processor with the following specifications:

- The processor uses a **byte-addressable** main memory of **16 GB**.
- The cache uses a **block size** of **16 KB**.
- Each memory address is split into **tag, index, and block offset**, and you are given that the **tag uses 10 bits**.

Calculate the **total size of the cache** in bytes.

Solution 19:

1. Main memory size = 16 GB = 2^{34} → **34 address bits**
2. Block size = 16 KB = 2^{14} → **Offset bits = 14**
3. Tag = 10 → Remaining for index = 34 − 10 − 14 = **10 bits**

So, number of cache blocks = 2^{10} = **1024 blocks**
Each block = 16 KB → Cache size = 1024 × 16 KB = 1024 × 2^{14} = 2^{10} × 2^{14} = 2^{24} = **16 MB**

Final Answer: 16 MB

Question 20: What is the difference between throughput and latency in pipelining?

Solution 20:

Latency

- Definition: The time it takes for a single instruction to go through the entire pipeline from start to finish.

- Example: If a pipeline has 5 stages and each stage takes 1 cycle, then the latency is 5 cycles per instruction.

Throughput

- Definition: The number of instructions completed per unit time (e.g., per clock cycle).

- Example: After the pipeline is filled, it can ideally complete 1 instruction per cycle, so the throughput is 1 instruction/cycle.

Question 21:

Explain different types of Cache Associativity

Solution 21

Cache associativity refers to how memory blocks are mapped to cache lines.

- **Direct-Mapped Cache**: Each memory block maps to exactly one cache line. Simple, but prone to conflicts (cache misses).
- **Set-Associative Cache**: Memory blocks map to a set of cache lines. More flexible than direct-mapped and reduces conflicts, but more complex.
- **Fully Associative Cache**: Any memory block can go into any cache line, minimizing conflicts, but it is the most complex and slowest to search.

Question 22:

You have an empty 2-way set-associative cache with eight 2-byte locations with LRU policy (Least Recently Used), given that you have the following (address) access pattern: 100, 104, 103, 108, 109. Determine the data in the cache with each memory access.

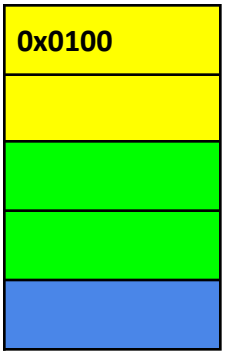
Please note that memory addresses are in decimal format. Data is in hex format.

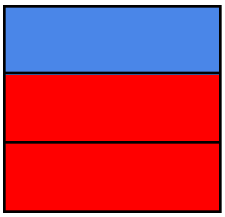
Address	Data
100	0x0100
101	0x0101
102	0x0102
103	0x0109
104	0x0104
105	0x0105
106	0x0106
107	0x0107
108	0x0108
109	0x0109

Solution 22:

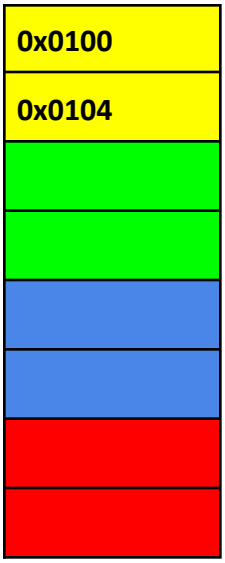
Given the cache with 8 locations and 2 way set associative, the number of sets are 4. Hence, the set is determined by (address mod 4) scheme. Given, memory access pattern as: (Yellow: Set 0, Green:Set 1, Blue: Set 2, Red: Set 3)

100:

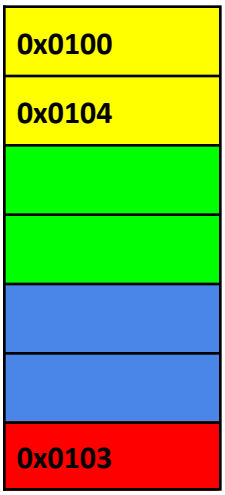




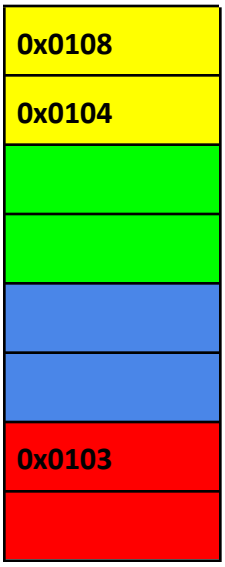
104:



103:



108: (0x0100 is replaced by 0x0108 because 0x0104 is just used recently)



109: