Normal Exam Session Solution

Module: Computer Architecture

Academic year : 2025-2026

Exercise 1: Comprehensive Questions (7 points)

1. Comparison of CISC and RISC architectures (1 point)

CISC (Complex Instruction Set Computer)	RISC (Reduced Instruction Set Computer)
Instruction Complexity: Complex, variable-length instructions (1-15 bytes)	**Instruction Complexity:** Simple, fixed-length instructions (typically 4 bytes)
Instruction Length: Variable (multiple formats)	**Instruction Length:** Fixed (simplifies decoding)
Registers: Fewer general-purpose registers (8-32)	**Registers:** More general-purpose registers (32-256)
Typical CPI: Higher CPI (1.5-2.5) due to complex instructions	**Typical CPI:** Lower CPI (close to 1.0) with pipelining

0,5 0,5

2. Principle of locality (1 point)

The principle of locality states that programs tend to reuse data and instructions that they have used recently. There are two types:

0,5

Temporal Locality: Recently accessed items are likely to be accessed again soon.

- **Example**: A loop variable accessed repeatedly

0,25

Spatial Locality: Items near recently accessed items are likely to be accessed soon. 0,25

- **Example**: Array elements accessed sequentially

3. Compare SRAM and DRAM (1 point)

Aspect	SRAM	DRAM
Structure	6 transistors (flip-flop)	1 transistor + 1 capacitor
Speed	Fast (1-10 ns access time)	Slow (50-100 ns access time)
Cost	Expensive	Cheap
Typical Use	Cache memory	Main memory (RAM)

0,5 0,5

4. Write-through vs Write-back cache policies (1 point)

Write-through:

- Data is written to both cache and main memory simultaneously
- **Advantage**: Simpler implementation, memory always consistent

Write-back:

- Data is written only to cache initially; main memory updated when block is replaced
- **Advantage**: Lower memory traffic, faster writes

5. Indirect addressing (1 point)

Why slower than direct addressing: Indirect addressing requires at least two memory accesses: one to get the address from memory, then another to access the actual data. Direct addressing needs only one memory access.

Useful applications:

- Pointer-based data structures (linked lists, trees)
- Dynamic memory allocation
- Virtual function tables in object-oriented programming
- Callback functions and function pointers

6. Modified vs Pure Harvard architecture (1 point)

Pure Harvard architecture has separate physical memories and buses for instructions and data.

Modified Harvard architecture (used in most modern computers):

- Separate caches for instructions and data (L1 cache)
- Unified main memory
- Allows more flexibility and better resource utilization
- Enables self-modifying code when needed
- More cost-effective implementation

7. Pipeline hazards (1 point)

Three types of pipeline hazards:

- **Structural Hazard:** Occurs when hardware resources are insufficient to support simultaneous execution.
  - **Example:** When memory is used for both instruction fetch and data access in the same cycle

- **Data Hazard:** Occurs when instructions depend on data produced by previous instructions.
  - **Types:** RAW (Read After Write), WAR (Write After Read), WAW (Write After Write)
- **Control Hazard:** Occurs due to branches, jumps, or exceptions that change the instruction flow.
  - **Example:** Branch instructions causing wrong instructions to be fetched

Exercise 2: (7 points)

1. Cache simulation table

Access	Block	Cache State	Frequency Count	Action
1	0	[0]	0:1	Miss
2	1	[0, 1]	0:1, 1:1	Miss
3	4	[0, 1, 4]	0:1, 1:1, 4:1	Miss
4	2	[1, 4, 2]	1:1, 4:1, 2:1	Miss (replace 0 choose oldest)
5	0	[4, 2, 0]	4:1, 2:1, 0:1	Miss (replace 1)
6	1	[2, 0, 1]	2:1, 0:1, 1:1	Miss (replace 4)
7	3	[0, 1, 3]	0:2, 1:2, 3:1	Miss (replace 2 - lowest frequency)
8	0	[0, 1, 3]	0:3, 1:2, 3:1	Hit
9	1	[0, 1, 3]	0:3, 1:3, 3:1	Hit
10	4	[0, 1, 4]	0:3, 1:3, 4:1	Miss (replace 3 - lowest frequency)
11	2	[0, 1, 2]	0:3, 1:3, 2:1	Miss (replace 4 - lowest frequency)
12	3	[0, 1, 3]	0:3, 1:3, 3:1	Miss (replace 2 - lowest frequency)
Total		**Hits: 2**	**Misses: 10**	

2. Hit rate and miss rate calculation

$$\text{Total accesses} = 12$$
$$\text{Hits} = 2$$
$$\text{Misses} = 10$$
$$\text{Hit rate} = \frac{2}{12} = 16,66\%$$
$$\text{Miss rate} = \frac{10}{12} = 83,33\%$$

3. Advantage of set-associative over direct-mapped cache

Set-associative caches reduce conflict misses compared to direct-mapped caches. In a direct-mapped cache, each memory block can only go to one specific cache line, causing thrashing when multiple frequently-used blocks map to the same line. Set-associative caches allow each block to be placed in any of several lines within a set, reducing such conflicts and improving hit rates.

4. Number of sets calculation

$$\text{Number of lines} = \frac{\text{Cache size}}{\text{Line size}} = \frac{65536}{128} = 512 \text{ lines}$$

$$\text{Number of sets} = \frac{\text{Number of lines}}{\text{Ways}} = \frac{512}{2} = 256 \text{ sets}$$

5. Tag bits calculation

$$\text{Offset bits} = \log_2(\text{Line size}) = \log_2(128) = 7 \text{ bits}$$
$$\text{Index bits} = \log_2(\text{Number of sets}) = \log_2(256) = 8 \text{ bits}$$
$$\text{Tag bits} = 32 - (\text{Index bits} + \text{Offset bits}) = 32 - (8 + 7) = 17 \text{ bits}$$

Exercise 3: (6 points)

1. Dependencies Identification

– RAW (Read After Write) :

- `Instr 2` → `Instr 1`: Instr 2 reads $t1, Instr 1 writes $t1
- `Instr 4` → `Instr 1`: Instr 4 reads $t1, Instr 1 writes $t1
- `Instr 4` → `Instr 2`: Instr 4 reads $t2, Instr 2 writes $t2
- `Instr 5` → `Instr 3`: Instr 5 reads $t6, Instr 3 writes $t6

– WAR (Write After Read) :

- `Instr 3` → `Instr 5`: Instr 3 reads $t4 in ID stage (C6), Instr 5 writes $t4 in WB stage (C13)

No WAW (Write After Write) Hazards

2. Pipeline execution diagram with stalls (No Forwarding)

Inst	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
1	IF	ID	EX	MEM	WB						
2		IF	ID	**ST**	**ST**	EX	MEM	WB			
3			IF	ID	EX	MEM	WB				
4				IF	ID	**ST**	**ST**	**ST**	EX	MEM	WB
5					IF	ID	**ST**	EX	MEM	WB	

3. Total cycles and IPC calculation

$$\text{Total cycles} = 11$$
$$\text{Number of instructions} = 5$$
$$\text{IPC (Instructions Per Cycle)} = \frac{5}{11} \approx 0.454$$

4. How forwarding reduces stalls

Forwarding (also called bypassing) is a hardware technique used in pipelined processors to reduce pipeline stalls caused by data hazards especially read-after-write (RAW) hazards. Forwarding avoids waiting for write-back by sending the result directly from where it is produced to where it is needed. With forwarding (data bypassing):

- **Instr 1 → Instr 2**: $t1 available after MEM stage (C4) → forward to Instr 2's EX stage (C4) → **Reduce 2 stalls to 0** 0,25
- **Instr 1 → Instr 4**: $t1 available after MEM stage (C4) → forward to Instr 4's EX stage (C6) → **Already available when needed** 0,25
- **Instr 2 → Instr 4**: $t2 available after MEM stage (C7) → forward to Instr 4's EX stage (C7) → **Reduce 3 stalls to 0** 0,25
- **Instr 3 → Instr 5**: $t6 available after EX stage (C5) → forward to Instr 5's EX stage (C6) → **Reduce 3 stalls to 0** 0,25