National University of Computer and Emerging Sciences

Karachi Campus

Computer Architecture (EE-3009)

Final Exam

Date: May 26th 2025

Course Instructor(s)

Mr. Aashir Mahboob, Dr. Nausheen , Mr. Kashan , Mr. Shoaib Rauf

Total Time (Hrs):	**3**
Total Marks:	**100**
Total Questions:	**7**

_____ _____ _____

Roll No Section Student Signature

Do not write below this line

SOLUTION

Rubrics for Checking: if only logical reason 2 marks for correct response, if example is required, 1 mark for example. Min of 2 differences if asked in question.

Q1: Logical Reasoning **[10x 2=20 Marks]**

i. Differentiate between the compulsory and conflict cache misses with examples.

- A compulsory miss occurs the first time a memory block is accessed and brought into the cache. Since the block has never been loaded into the cache before, the cache has no choice but to fetch it from main memory. E.g. 1,2,3,1,2 → request for 1,2,3
- A conflict miss happens when multiple memory blocks compete for the same cache location, and the cache mapping policy (usually direct-mapped or set-associative) causes one to be evicted and replaced by another. E.g. → 4 mod 2 & 8 mod 2 (Both maps in block 0 of cache)

ii. Briefly explain Pros and Cons of optimizing cache miss rate using higher associativity?

Pros→ Reduce conflict miss, Reduce miss rate and eviction rate.

Cons→ Increase in the hit time, Complex design.

iii. Explain Write Strategies in case of Cache Hit with examples.

- Write through—The information is written to both the block in the cache and to the block in the lower-level memory.
- Write back—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced. (use of dirty bit)

iv. How conflict misses are increased due to a larger block size. Assume cache size remains constant.

When the **cache size remains constant**, but the **block size increases**, the **number of blocks (cache lines)** in the cache **decreases**. This reduction increases the chances of **conflict misses**, especially in **direct-mapped** or **low-associativity** caches.

v. Differentiate between Temporal and Spatial Locality with examples.

- **Temporal locality** refers to the reuse of specific data or resources **within relatively short time intervals.** If a memory location is accessed, it is likely to be accessed **again soon.** E.g. **1, 2, 3,** 5, 6, **2, 3,** 1, 7, **1, 2, 3**
- **Spatial locality** refers to accessing **data elements that are close together in memory** within a short time period. E.g. **MISSISSIPPI**

vi. What is the impact on execution time of a given program using a deeper pipeline?

A **deeper pipeline** means breaking the instruction execution path into **more stages,** allowing **more instructions to be in-flight simultaneously.** This approach is intended to increase **instruction throughput** and improve **CPU clock speed.**

vii. How does superscalar architecture processor exploits Instruction level parallelism?

Multiple Execution Units, Instruction Fetch and Decode in Parallel, Out-of-Order Execution, Register Renaming, Dynamic Scheduling etc.

viii. Why operation decoding is a parallel process in VLIW processors?

The bundles follow a fixed format and so the decode operations are done in parallel. Pre-scheduled by the compiler, Independent of each other and Targeted at different functional units

ix. Give a coding example, how branch penalty is reduced using delayed branch method.

```
 BEQ R1, R2, LABEL      ; Branch if R1 == R2
    ; <--- stall (branch delay slot)
LABEL:
    ADD R3, R4, R5        ; Continue execution
          With delayed Branch Method
BEQ R1, R2, LABEL         ; Branch if R1 == R2
    ADD R6, R7, R8            ; Delay slot (always executed)
LABEL:
    ADD R3, R4, R5            ; Continue execution.
```

x. Distinguish between Dynamic and Static Scheduling.

- **Static scheduling** is determined at **compile time,** while **dynamic scheduling** is determined **at runtime** by the processor.
- **Static scheduling** relies on the **compiler's analysis** of dependencies, while **dynamic scheduling** adapts to **runtime conditions** like hazards and branches.
- **Static scheduling** has **lower hardware complexity,** whereas **dynamic scheduling** requires more complex hardware for dependency tracking and out-of-order execution.

Rubrics for Checking:12 Marks for Instruction Status Table (4+4+4), 3 marks for Functional Unit Status table and 1 mark for Register Status Table. Deduct 0.5 mark for each wrong entry in table 1. If table 2 and 3 entries/table are/is missing, deduct 3 and 1 mark respectively.

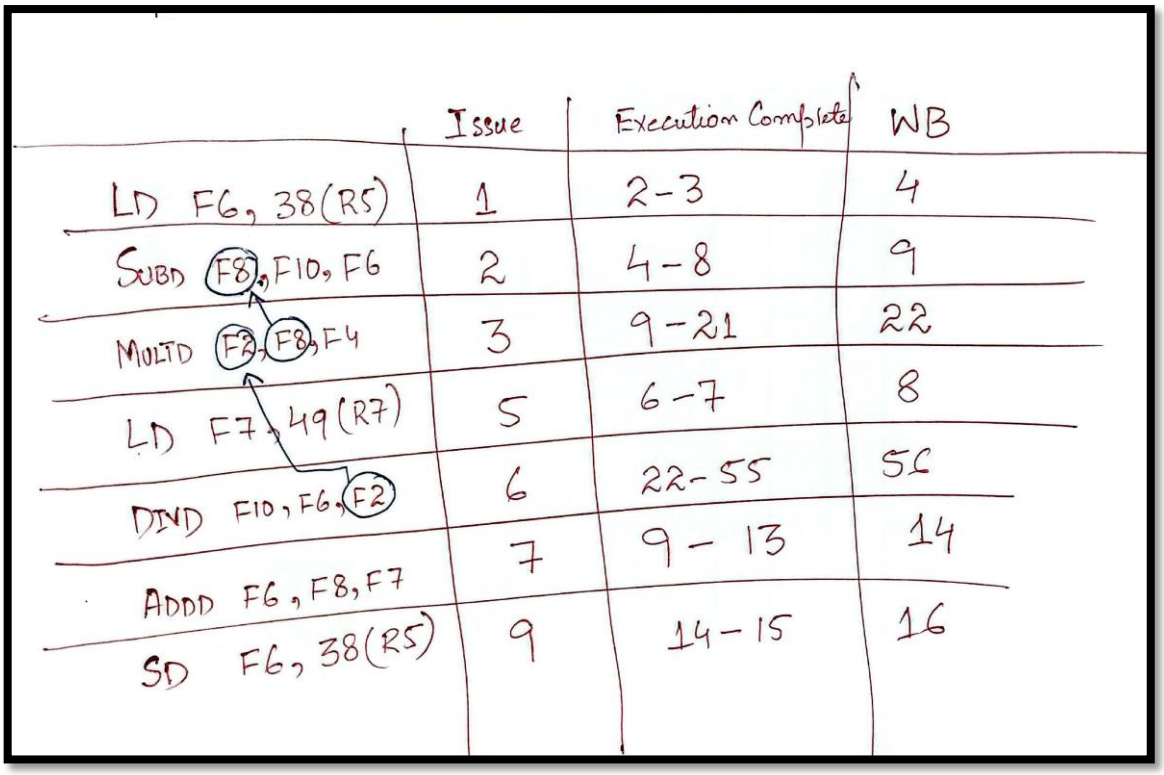
Q2: Determine the scheduling of the instructions based on the ***Tomasulo's technique*** for the following MIPS instructions executed in Pipelined System. **[16 Marks]**

LD F6 ,38(R5)
SUBD F8, F10, F6

MULTD F2, F8, F4
LD F7 ,49(R7)
DIVD F10, F6, F2
ADDD F6, F8, F7
SD F6 ,38(R5)

- Instruction Status, Functional unit Status and Register Result Status tables must be drawn.
- Write down order of issue, execution and competition.:

Operation	Latency	No : of Units Available
LD/SD	2 Cycle	1
ADD/SUB	4 Cycle	2
MULD	12 Cycles	2
DIVD	33 Cycles	2



Rubrics for Checking: 2 Marks for Hazard Identification and Justification, 1 Mark for Optimal Solution and 2 Marks for Hazard free pipeline Diagram.

Q3: Consider the following code sequences, latencies of FADD is 4, FMUL is 7, FSD/FLD is 1.

[5*4=20 Marks]

I. Identify and Justify all type(s) of Dependencies between instructions (Hazards).

II. Suggest optimal solution to resolve the hazard(s).

III. Draw Pipeline diagram (without hazard) for each case to support your answer.

A)
`fld f4,0(x2)`
`fmul.d f0,f4,f6`
`fadd.d f2,f0,f8`
`fsd f2,0(x2)`
	B)
	`fmul.d f0,f4,f6`
	`ADD R1,R2,R3`
	`SUB R4,R5,R6`
	`fadd.d f2,f4,f6`
	`SUB R7,R8,R10`
	`ADD R11,R12,R13`
	`fsd f2,0(x2)`

C)
`add.d f0, f2, f4`
`add.d f6, f8, f10`
`add.d F12, f14, f16`
`add.d f18, F12, f20`
	D)
	`mul.d f0, f1, f2`
	`sub.d f5, f0, f6`
	`add.d f0, f3, f4`

Part A)

i. RAW Hazard Between I1 and I2 (F4), RAW Hazard Between I2 and I3 (F0), RAW Hazard Between I3 and I4 (F2)

ii. Stall +Forwarding is solution

iii.

	Clock cycle number																
Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
`fld f4,0(x2)`	IF	ID	EX	MEM	WB												
`fmul.d f0,f4,f6`		IF	ID	Stall	M1	M2	M3	M4	M5	M6	M7	MEM	WB				
`fadd.d f2,f0,f8`			IF	Stall	ID	Stall	Stall	Stall	Stall	Stall	Stall	A1	A2	A3	A4	MEM	WB
`fsd f2,0(x2)`					IF	Stall	Stall	Stall	Stall	Stall	Stall	ID	EX	Stall	Stall	Stall	MEM

Part B)

i. Structural Hazard between I1, I4, I7 (WB in same Clock)

ii. Stall I4 by 1 cycle and I7 by 2 Cycles after Memory Access Stage.

	Clock Number										
	1	2	3	4	5	6	7	8	9	10	11
MULTD F0, F4, F6	IF	D	M1	M2	M3	M4	M5	M6	M7	M	WB
...		F	D	E	M	W					
...			F	D	E	M	W				
ADDD F2, F4, F6				F	D	A1	A2	A3	A4	M	WB
...					F	D	E	M	W		
...						F	D	E	M	W	
LD F2, 0(R2)							F	D	E	M	WB

Part B Solution

	1	2	3	4	5	6	7	8	9	10	11	12	13
fmul.d f0, f4, f6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB		
ADD R1, R2, R3		IF	ID	EX	ME	WB							
SUB R4, R5, R6			IF	ID	EX	ME	WB						
Fadd.d F2, F4, F6				IF	ID	A1	A2	A3	A4	MEM	Stall	WB	
SUB R7, R8, R10					IF	ID	EX	ME	WB				
ADD R11, R12, R13						IF	ID	EX	ME	WB			
FSD F2, 0(x2)							IF	ID	EX	ME	Stall	Stall	WB

Part C)

iii. RAW Hazard Between Instruction I3 and I4 (F12 Dependency)

iv. Stall I4, until Result of I3 is generated. And forward the result (Stall+Data Forwarding)

PART C

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ADD.D F0, F2, F4	IF	ID	A1	A2	A3	A4	ME	WB						
ADD.D F6, F8, F10		IF	ID	A1	A2	A3	A4	ME	WB					
ADD.D F12, F14, F16			IF	ID	A1	A2	A3	A4	ME	WB				
ADD.D F18, F12, F20				IF	ID	Stall	Stall	Stall	A1	A2	A3	A4	ME	W

Part D)

v. RAW Hazard Between I1 and I2 (F0 Dependency).

vi. Solution → Stall +Forwarding

i.

PART D

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Mul.d	F0, F1, F2	IF	ID	M1	M2	M3	M4	M5	M6	M7	ME	WB				
SUB.D	F5, F0, F6		IF	ID	Stall	Stall	Stall	Stall	Stall	Stall	A1	A2	A3	A4	ME	WB
ADD.D	F0, F3, F4			IF	Stall	Stall	Stall	Stall	Stall	Stall	ID	A1	A2	A3	A4	ME

Rubrics for Checking: Each part is divided as 2+2+1, where 1 mark is for working and 1 for answer. For speedup, 1 mark for answer only. If ratio is taken inverse by student, deduct 0.5 marks, if unit is missing in answer, deduct 0.5 marks.

Q4: Assume that the original processor is a 5-stage pipeline with a 1 ns clock cycle. The second processor is a 12-stage pipeline with a 0.5 ns clock cycle. The 5-stage pipeline experiences one stall cycle due to a data hazard every 5 instructions, whereas the 12-stage pipeline experiences 3 stall cycles every 8 instructions. In addition, branches constitute 20% of the instruction count, and the misprediction rate for both pipelines is 5%. **[5*2=10 Marks]**

a) What is the speedup of the 12-stage pipeline over the 5-stage pipeline, taking into account only data hazards?

Average CPI (5-stage pipeline) = 1 + 1/5 = 6/5 (Data hazard stalls only)
Average CPI (12-stage pipeline) = 1 + 3/8 = 11/8 (Data hazard stalls only)
Speedup = (6/5 × 1 ns) / (11/8 × 0.5 ns) = 1.745 (Data hazards only)

b) If the branch misprediction penalty is 2 cycles for the 5-stage pipeline, but 6 cycles for the 12-stage pipeline, what are the CPIs of each, taking into account the stalls of the data hazards and branch hazards?

Average CPI (5-stage pipeline) = 1 + 1/5 + 0.2 × 0.05 × 2 = 1.22 (Data + Branch Hazards)
Average CPI (12-stage pipeline) = 1 + 3/8 + 0.2 × 0.05 × 6 = 1.435 (Data + Branch Hazards)
Speedup = (1.22 × 1 ns) / (1.435 × 0.5 ns) = 1.70 (Data + Branch Hazards)

Rubrics for Checking: 1 mark for calculation, 1 mark for answer, 1 mark for explanation in each part. If unit is wrong in answer, deduct 0.5 marks.

Q5: A computer has a single cache (off-chip) with a 2 ns hit time and a 98% hit rate. Main memory has a 40 ns access time. ***Briefly explain results calculated in each part below.*** **[3*3=9 Marks]**

a) What is the computer's effective access time?

$$2 \text{ ns} + .02 * 40 \text{ ns} = 2.8 \text{ ns.}$$

b) If we add an on-chip cache with a .5 ns hit time and a 94% hit rate, what is the computer's effective access time?

With the on-chip cache, we have $.5 \text{ ns} + .06 * (2 \text{ ns} + .02 * 40 \text{ ns}) = .668 \text{ ns.}$

c) How much of a speedup does the on-chip cache give the computer?

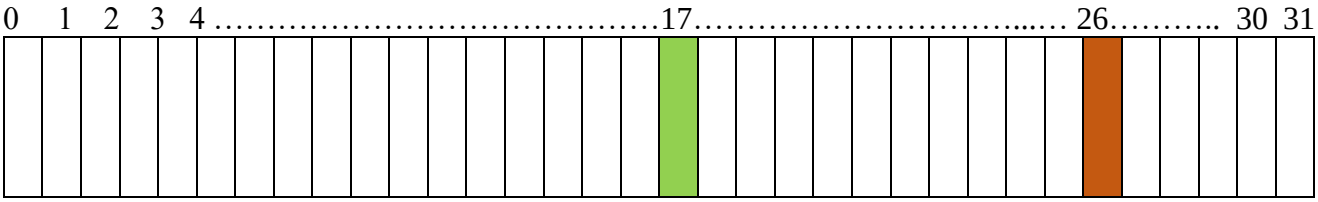
$$\text{The Speedup} = \frac{2.8\ ns}{.668\ ns} = 4.19 \rightarrow 4.2$$

Rubrics for Checking: 1 mark for Block identification, 1 mark for pictorial representation, 1 mark for explanation in each part. If blocks are numbered from 1 instead of 0, deduct 0.5 marks.

Q6: Consider a Main Memory having 32 Blocks each of 8-bits. A Cache memory having a capacity of 96-bits with a block size of 1 Byte. The processor requests for block 17 and 26 from Main memory. Calculate and give pictorial representation of block placement in cache for the following placement policies: **[3*3 = 9 Marks]**

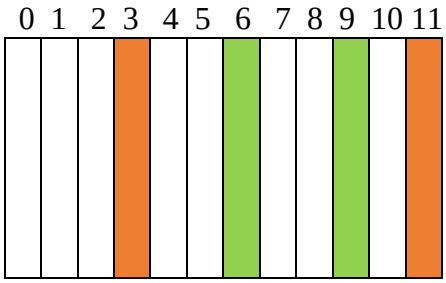
a) Fully Associative
b) Direct Mapped
c) 4-Way Associative

Main Memory



Cache Memory

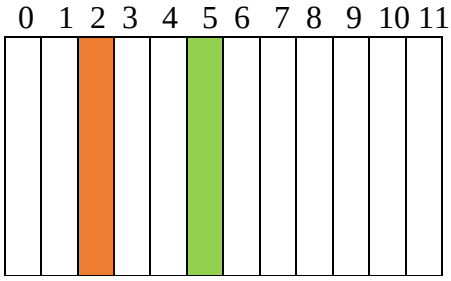
a)



Both can be mapped anywhere in cache.

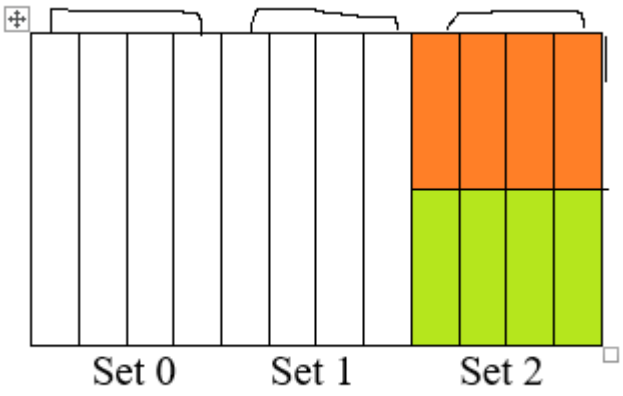
Part B. Direct Mapped.

26 MOD 12 → 2
17 MOD 12 → 5



Part C. 4-Way Associative.

26 MOD 3 → 2
17 MOD 3 → 2



Since it is 4-way associative, both blocks can occupy any place in Set -2

Rubrics for Checking: Part I, 2 marks for re-write code with stalls, 2 marks for cycles calculation, Part ii, 4 marks for unroll loop (specially checking for LD/SD constant value and stalls), 2 marks for calculation. Part iii 4 marks for re-order/re-naming of registers, 2 marks for calculation.

Q7: Consider the given RISC-V code with following operation latencies (Fig 1): **[16 Marks]**

```
For(i=99 ; i>=0; i=i-1)
        X[i] = a*X[i]+S
```

```
FOO: FLD      F0,0(X1)
     MULTD    F4,FO,F3
     FADD.D   F5,F4,F2
     FSD      F5,0(X1)
     ADDI     X1,X1,-8
     BNE      X1,X2,FOO
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Figure 1 Instruction Latency

i. Calculate, how many cycles does the RISC-V code instructions require? Re-write the code and show all necessary steps.

```
FOO: FLD      F0,0(X1)
     Stall
     MULTD    F4,FO,F3
     Stall
     Stall
     Stall
     FADD.D   F5,F4,F2
     Stall
     Stall
     FSD      F5,0(X1)
     ADDI     X1,X1,-8
```

```
BNE     X1,X2,FOO
```

```
Total#of Cycles = 12 (each iteration)
Total#of Cycles = 12x100 = 1200 (Complete Code)
```

ii. Loop unrolling is one standard compiler technique for finding more parallelism in code. Hand-unroll 3 times above code and calculate total number of clock cycles. Re-write the code and show all necessary steps.

```
FLD         F0,0(X1)
Stall
MULTD       F4,FO,F3
Stall
Stall
Stall
FADD.D      F5,F4,F2
Stall
Stall
FSD         F5,0(X1)
```

```
FLD         F0,-8(X1)
Stall
MULTD       F4,FO,F3
Stall
Stall
Stall
FADD.D      F5,F4,F2
Stall
Stall
FSD         F5,-8(X1)
```

```
FLD         F0,-16(X1)
Stall
MULTD       F4,FO,F3
Stall
Stall
Stall
FADD.D      F5,F4,F2
Stall
Stall
FSD         F5,-16(X1)
```

```
Total#of Cycles = 10 (each iteration)
Total#of Cycles = 10x100 = 1000 (Complete Code)
```

iii. After unrolling 3 times, Reorder the instructions to improve performance of the code. How many cycles does your reordered code take? Re-write the code and show all necessary steps.

```
FOO:     FLD       F1,0(X1)
         FLD       F2,-8(X1)
         FLD       F3,-16(X1)
         MULTD     F4,F1,F5
         MULTD     F4,F2,F5
         MULTD     F4,F3,F5
         FADD.D    F5,F4,F2
         FADD.D    F5,F4,F2
         FADD.D    F5,F4,F2
         FSD       F5,0(X1)
         FSD       F5,-8(X1)
         FSD       F5,-16(X1)

         ADDI   X1,X1,-24
         BNE    X1,X2,FOO
```

Total#of Cycles = 14

========== **Good Luck** ==========