Surname (COGNOME)	**SOLUTION**
Name	
POLIMI Personal Code	
Signature	

Politecnico di Milano, 11 February, 2025

Course on Advanced Computer Architectures

Prof. C. Silvano

EX1	**(5 points)**	
EX2	**(3 points)**	
EX3	**(2 points)**	
EX4	**(5 points)**	
Q1	**(5 points)**	
Q2	**(5 points)**	
QUIZZES	**(8 points)**	
TOTAL	**(33 points)**	

EXERCISE 1 – DEPENDENCY ANALYSIS + TOMASULO (5 points)

1. *Let's consider the following assembly code containing multiple types of intra-loop dependences. Complete the following table by inserting all types of true-data-dependences, anti-dependences and output dependences for each instruction:*

I#	TYPE OF INSTRUCTION	ANALYSIS OF DEPENDECES: 1. True data dependence with I# for $Fx 2. Anti-dependence with I# for $Fy 3. Output-dependence with I# for $Fz
I0	`LOOP: LD $F0, A($R0)`	None
I1	`LD $F2, B($R0)`	None
I2	`FADD $F4, $F0, $F2`	True data dependence with **I0** for **$F0** True data dependence with **I1** for **$F2**
I3	`FADD $F6, $F4, $F4`	True data dependence with **I2** for **$F4**
I4	`SD $F4, C($R0)`	True data dependence with **I2** for **$F4**
I5	`SD $F6, D($R0)`	True data dependence with **I3** for **$F6**
I6	`ADDI $R0, $R0, 4`	Anti dependence with **I0, I1, I4, I5** for **$R0**
I7	`BNE $R0, $R1, LOOP`	True data dependence with **I6** for **$R0**

2. Let's consider the previous assembly code to be executed on a CPU with dynamic scheduling based on **TOMASULO** algorithm with all cache HITS, a single Common Data Bus and:
   - *2 RESERV. STATIONS **(RS1, RS2)** with 2 LOAD/STORE units **(LDU1, LDU2)** with latency **6***
   - *2 RESERVATION STATION **(RS3, RS4)** with 2 FP unit12 **(FPU1, FPU2)** with latency **2***
   - *1 RESERVATION STATION **(RS5)** with 1 INT_ALU/BR unit **(ALU1)** with latency **1***

   Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`LOOP: LD $F0, A($R0)`	1	2	8	None	RS1	LDU1
`LD $F2, B($R0)`	2	3	9	None	RS2	LDU2
`FADD $F4, $F0, $F2`	3	**10**	12	**RAW $F0 RAW $F2**	RS3	ALU1
`FADD $F6, $F4, $F4`	4	**13**	15	**RAW $F4**	RS4	ALU2
`SD $F4, C($R0)`	**9**	**13**	19	**STRUCT RS1 RAW $F4**	RS1	LDU1
`SD $F6, D($R0)`	10	**16**	22	**RAW $F6**	RS2	LDU2
`ADDI $R0, $R0, 4`	11	12	13		RS5	ALU1
`BNE $R0, $R1, LOOP`	14	15	16	**STRUCT RS4** (RAW $R0)	RS5	ALU2

(*) Only hazards that have caused the introduction of some stalls are reported in the table. Other dependencies are already solved.
(**) In Tomasulo, WAW and WAR hazards are already solved by Register Renaming in ISSUE stage.

*Calculate the **CPI** =* ***CPI = # clock cycles / IC = 22 / 8 = 2,75***

EXERCISE 2 – VLIW SCHEDULING (3 points)

Let's consider the following LOOP code:

```
LOOP:LD F1, A(R1)
     LD F2, A(R2)
     LD F3, A(R3)
     FADD F1, F1, F2
     FADD F2, F2, F3
     FMUL F1, F1, F2
     FADD F3, F3, F3
     ADDUI R2, R1, 8
     ADDUI R3, R1, 8
     SD F1, B(R1)
     ADDUI R1, R1, 4
     BNE R1, R6, LOOP
```

Given a **3-issue VLIW** machine with **fully pipelined functional units:**

- **1 Memory Unit with 3 cycles latency**
- **1 FP ALU with 2 cycles latency**
- **1 Integer ALU with 1 cycle latency to next Int/FP & 2 cycle latency to next Branch**

The branch is completed with 1 cycle delay slot (branch solved in ID stage). **No branch prediction.** In the Register File, it is possible to read and write at the same address at the same clock cycle. *Considering one iteration of the loop, complete the following table by using the* ***list-based scheduling*** (do NOT introduce any software pipelining, loop unrolling and modifications to loop indexes) on the *4-issue VLIW machine including the BRANCH DELAY SLOT.* Please do not write in NOPs.

	Memory Unit 1	Floating Point Unit 1	Integer Unit
C1	LD F1, A(R1)		
C2	LD F2, A(R2)		ADDUI R2,R1,8
C3	LD F3, A(R3)		ADDUI R3,R1,8
C4			
C5		FADD F1,F1,F2	
C6		FADD F2,F2,F3	
C7		FADD F3,F3,F3	
C8		FMUL F1,F1,F2	
C9			
C10	SD F1,B(R1)		ADDUI R1,R1,4
C11			
C12			BNE R1,R2, LOOP
C13			Br. delay slot
C14			
C15			

1. How long is the critical path?

   13 cycles (There is NO branch prediction, so the branch delay slot cannot be used for next iteration)

2. How much is the code efficiency?

   Code_eff = IC/ (# cycles * # issues) = 12 / (13 * 3) = 12 / 39 = 0.3

EXERCISE 2 – MESI PROTOCOL (2 points)

Let's consider the following access patterns on a **4-processor** system with a direct-mapped, write-back cache with one cache block per processor and a two-cache block memory.

Assume the **MESI protocol** is used with **write-back** caches, **write-allocate,** and **write-invalidate** of other caches.

Please COMPLETE the following table:

Cycle	After Operation	P0 cache block state	P1 cache block state	P2 cache block state	P3 cache block state	Mem. at bl. 0 up to date?	Mem. at bl. 1 up to date?
1	P0: Read Bl. 1	Excl (1)	Invalid	Invalid	Invalid	Yes	Yes
2	P2: Read Bl. 0	Excl (1)	Invalid	Excl (0)	Invalid	Yes	Yes
3	P3: Read Bl. 0	Excl (1)	Invalid	Shared (0)	Shared (0)	Yes	Yes
4	P1: Write Bl. 0	Excl (1)	Mod (0)	Invalid	Invalid	No	Yes
5	P0: Write Bl. 1	Mod (1)	Mod (0)	Invalid	Invalid	No	No
6	P3: Read Bl. 1	Shared(1)	Mod (0)	Invalid	Shared (1)	No	Yes

EXERCISE 4 on REORDER BUFFER (5 points)

Let's consider the following assembly loop where registers **$R1** *and* **$R2** *are initialized at 0 and 40 respectively:*

```
L0:    LD $F2, 0 ($R1)
L1:    ADDD $F4, $F2, $F2
L2:    SD $F4, 0 ($R1)
L3:    ADDI $R1, $R1, 4
L4:    BNE $R1, $R2, L0 # branch predicted as taken
```

- *How many loop iterations?* 10 iterations
- *How many instructions per iteration?* 5 instructions
- *How many control instructions vs datapath instructions?* 2 vs 3

1. Write the unrolled version of the loop with unrolling factor 2 by using Register Renaming:

L0:	LD $F2, 0 ($R1)
L1:	ADDD $F4, $F2, $F2
L2:	SD $F4, 0 ($R1)
L3:	LD $F6, 4 ($R1)
L4:	ADDD $F8, $F6, $F6
L5:	SD $F8, 4 ($R1)
L6:	ADDI $R1, $R1, 8
L7:	BNE $R1, $R2, LOOP

- *How many loop iterations?* 5 iterations
- *How many instructions per iteration?* 8 instructions
- *How many control instructions vs datapath instructions?* 2 vs 6

2. *Execute the unrolled version of the loop by the* ***Speculative Tomasulo*** *architecture with a* ***10-entry ROB*** *and:*
   - *4 Load Buffers (Load1, Load2, Load3, Load4);*
   - *4 FP Reservation Stations (FP1, FP2, FP3, FP4)*
   - *2 Integer Reservation Stations (Int1, Int2)*

   Complete the ROB and the Rename Table until the ROB becomes ***full*** *while the first instruction is still in execution due to a cache miss (*):*

ROB Table

ROB#	Instruction	Dest.	Resource Allocation	Ready /Status	Spec.	
ROB0	`L0: LD $F2, 0 ($R1)`	**$F2**	Load1	No, exec.(*)	No	HEAD
ROB1	`L1: ADDD $F4, $F2, $F2`	**$F4**	FP1	No, issued	No	
ROB2	`L2: SD $F4, 0 ($R1)`	Mem	--	No, issued	No	
ROB3	`L3: LD $F6, 4 ($R1)`	**$F6**	Load2	No, issued	No	
ROB4	`L4: ADDD $F8, $F6, $F6`	**$F8**	FP2	No, issued	No	
ROB5	`L5: SD $F8, 4 ($R1)`	Mem	--	No, issued	No	
ROB6	`L6: ADDI $R1, $R1, 8`	$R1	Int1	No, issued	No	
ROB7	`L7: BNE $R1, $R2, LOOP`	--	Int2	No, issued	No	
ROB8	`L0: LD $F2, 0 ($R1)`	**$F2**	Load3	No, issued	Yes	
ROB9	`L1: ADDD $F4, $F2, $F2`	**$F4**	FP3	No, issued	Yes	

Rename Table:

Reg#	ROB#
$F0	--
$F2	~~ROB0,~~ ROB8
$F4	~~ROB1,~~ ROB9
$F6	ROB3
$F8	ROB4

QUESTION 1: Instruction-Level and Thread-Level Parallelism (5 points)

Modern processors exploit Instruction Level Parallelism and Thread Level Parallelism.
Answer to the following questions:

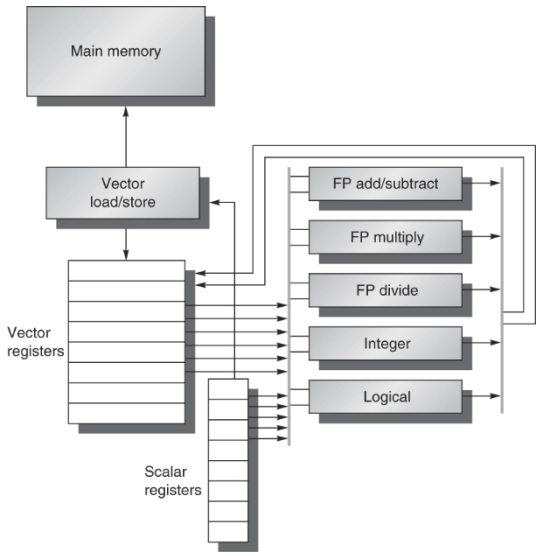
	Instruction Level Parallelism (ILP)	**Thread Level Parallelism (TLP)**
Explain the main concepts for each approach.		
Which type of technique can be applied in superscalar processors to combine both ILP and TLP?		

Explain what type of instruction scheduling is used in superscalar processors to combine both ILP and TLP?	
What are the main hardware modifications required to a generic superscalar processor to support both ILP and TLP?	

QUESTION 2: VECTOR PROCESSORS (5 points)

Consider a vector processor architecture, as VMIPS shown in the figure.

- Present the main concepts;
- Present the main advantages of vector execution with respect to scalar execution, detailing the features of the vector architecture that provide the advantage.



QUIZZES (8 points)

Question 1 (format Multiple Choice – Single answer)

Let's consider the following code executed by a Vector Processor with:

- *Vector Register File composed of 32 vectors of 8 elements per 64 bits/element;*
- *Scalar FP Register File composed of 32 registers of 64 bits;*
- *One Load/Store Vector Unit with operation chaining and memory bandwidth 64 bits;*
- *One ADD/SUB Vector Unit with operation chaining;*
- *One MUL/DIV Vector Unit with operation chaining.*

```
L.V V1, RA          # Load vector from memory address RA into V1
L.V V3, RB          # Load vector from memory address RB into V3
MULVS.D V1, V1, F0  # FP multiply vector V1 to scalar F0
ADDVV.D V2, V1, V1  # FP add vectors V1 and V1
MULVS.D V2, V2, F0  # FP multiply vector V2 to scalar F0
ADDVV.D V3, V2, V3  # FP add vectors V2 and V3
S.V V1, RX          # Store vector V3 into memory address RX
S.V V2, RY          # Store vector V3 into memory address RY
S.V V3, RZ          # Store vector V3 into memory address RZ
```

How many convoys? How many clock cycles to execute the code?
(SINGLE ANSWER)
1 point

Answer 1: 2 convoys; 16 clock cycles
Answer 2: 3 convoys; 24 clock cycles
Answer 3: 4 convoys; 32 clock cycles
Answer 4: 5 convoys; 40 clock cycles **(TRUE)**
Answer 5: 6 convoys; 48 clock cycles

Motivate your answer by completing the following table:
1 point

	Load/Store Vector Unit	Add/Sub Vector Unit	Mul/Div Vector Unit
1^ convoy	L.V V1, RA;	ADDVV.D V2, V1, V1	MULVS.D V1, V1, F0
2^ convoy	L.V V3, RB	ADDVV.D V3, V2, V3	MULVS.D V2, V2, F0
3^ convoy	S.V V1, RX		
4^ convoy	S.V V2, RY		
5^ convoy	S.V V3, RZ		

Another possible solution is:

	Load/Store Vector Unit	Add/Sub Vector Unit	Mul/Div Vector Unit
1^ convoy	L.V V1, RA;	ADDVV.D V2, V1, V1	MULVS.D V1, V1, F0
2^ convoy	S.V V1, RX		
3^ convoy	L.V V3, RY	ADDVV.D V3, V2, V3	MULVS.D V2, V2, F0
4^ convoy	S.V V2, RY		
5^ convoy	S.V V3, RZ		

Question 2 (format True/False)
In VLIW architectures, the compiler can detect parallelism only in basic blocks of the code;
(format True/False)
1 point

Answer 1: True / False **(False)**

Motivate your answer:
1 point

_____

_____

_____

_____

Question 3 (format True/False)
A 4-issue VLIW processor requires 4 Program Counters to load the necessary instructions in the 4 parallel lanes
(format True/False)
1 point

Answer 1: True / False **(False)**

Motivate your answer:
1 point

_____

_____

_____

_____

Question 4 (format Multiple Choice – Single answer)

*In the **Speculative Tomasulo Architecture**, what type of hardware block is used to undo speculative instructions in case of a mispredicted branch?*
(SINGLE ANSWER)
1 point

Answer 1: Reorder Buffer; **(TRUE)**

Answer 2: Instruction Dispatcher;

Answer 3: Store Buffers;

Answer 4: Reservation Stations;

Answer 5: Load Buffers;

Motivate your answer by explaining what happens on a mispredicted branch:
1 point