Surname	
Name	
POLIMI Personal code	
Signature	

SOLUTION

Politecnico di Milano, 5 May 2025

Course on Advanced Computer Architectures

Prof. C. Silvano

EXERCISE 1	**(5 points)**	
EXERCISE 2	**(5 points)**	
EXERCISE 3	**(3 points)**	
QUIZ 4	**(1 point)**	
QUIZ 5	**(1 point)**	
TOTAL	**(15 points)**	
QUIZ 6 OPTIONAL	**+ 3 extra points**	

EXERCISE 1.A - SOFTWARE PIPELINING (2 points)

Consider the following software pipelined loop **SP_LOOP** *and the corresponding start-up and finish-up code:*

```
START_UP:   LD F0, 0 (R1)
            LD F2, 0 (R2)
            FADD F4, F0, F0
            FADD F6, F2, F2
            LD F0, 8 (R1)
            LD F2, 8 (R2)

SP_LOOP:    SD F4, 0 (R1)
            SD F6, 0 (R2)
            FADD F4, F0, F0
            FADD F6, F2, F2
            LD F0, 16 (R1)
            LD F2, 16 (R2)
            ADDUI R1, R1, 8
            ADDUI R2, R2, 8
            ADD R3, R2, R1
            BNE R3, R4, SP_LOOP

FINISH-UP: SD F4, 8 (R1)
            SD F6, 8 (R2)
            FADD F4, F0, F0
            FADD F6, F2, F2
            SD F4, 16 (R1)
            SD F6, 16 (R2)
```

1. *Reconstruct the original (non-pipelined) version of the code:*

Feedback:

```
LOOP: LD F0, 0 (R1)
      LD F2, 0 (R2)
      FADD F4, F0, F0
      FADD F6, F2, F2
      SD F4, 0 (R1)
      SD F6, 0 (R2)
      ADDUI  R1, R1, 8
      ADDUI R2, R2, 8
      ADD R3, R2, R1
      BNE R3, R4, LOOP
```

EXERCISE 1.B - SOFTWARE PIPELINING (3 points)

Given the same software pipelined loop of ***Exercise 1.A:***

```
SP_LOOP: SD F4, 0 (R1)
         SD F6, 0 (R2)
         FADD F4, F0, F0
         FADD $F6, F2, F2
         LD F0, 16 (R1)
         LD F2, 16 (R2)
         ADDUI R1, R1, 8
         ADDUI R2, R2, 8
         ADD R3, R2, R1
         BNE R3, R4, SP_LOOP
```

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

- **1 Memory Units with 3 cycles latency**
- **1 FP ALUs with 3 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.

1) *Considering one iteration of the* `SP_LOOP`, *complete the following table by using the* ***list-based scheduling*** (do NOT introduce any loop unrolling and modifications to loop indexes) on the *3-issue VLIW machine including the BRANCH DELAY SLOT.* Please do not write in NOPs.

	Memory Unit	Floating Point Unit	Integer Unit
C1	`SD F4, 0 (R1)`	`FADD F4, F0, F0`	
C2	`SD F6, 0 (R2)`	`FADD $F6, F2, F2`	
C3	`LD F0, 16 (R1)`		`ADDUI R1, R1, 8`
C4	`LD F2, 16 (R2)`		`ADDUI R2, R2, 8`
C5			`ADD R3, R2, R1`
C6			
C7			`BNE R3, R4, SP_LOOP`
C8			`(br. delay slot)`
C9			
C10			
C11			
C12			
C13			
C14			
C15			

2. *How long is the critical path for a single iteration?*

   _____

3. *What performance did you achieve in CPI?*

   _____

4. *What performance did you achieve in FP ops per cycles?*

   _____

5. *How much is the code efficiency?*

   _____

Feedback

	Memory Unit	Floating Point Unit	Integer Unit
C1	`SD F4, 0(R1)`	`FADD F4, F0, F0`	
C2	`SD F6, 0(R2)`	`FADD F6, F2, F2`	
C3	`LD F0, 16(R1)`		`ADDUI R1, R1, 8`
C4	`LD F2, 16(R2)`		`ADDUI R2, R2, 8`
C5			`ADD R3, R2, R1`
C6			
C7			`BNE R3, R4, SP_LOOP`
C8			`(br. delay slot)`

1. *How long is the critical path for a single iteration? (*)*

   8 cycles

   (*) When considering many iterations, the next iteration could start at cycle C8, therefore the critical path would be 7 cycles and the CPI_{AS} = 7 / 10 = 0.7

2. *What performance did you achieve in CPI?*

   CPI = # cycles / IC = 8 / 10 = 0.8

3. *What performance did you achieve in FP ops per cycles?*

   FP ops per cycles = # FP ops / # cycles = 2 / 8 = 0.25

4. *How much is the code efficiency?*

   Code efficiency = IC / (# cycles x # issues) = 10 / (8 x 3) = 10 / 24 = 0.42

EXERCISE 2.A – DEPENDENCY ANALYSIS (2 points)

1. *Consider the same software pipelined loop of **Exercise 1.A** 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
I1	`SP_LOOP: SD F4, 0 (R1)`	None
I2	`SD F6, 0 (R2)`	None
I3	`FADD F4, F0, F0`	Anti dependence with **I1** for **F4**
I4	`FADD F6, F2, F2`	Anti dependence with **I2** for **F6**
I5	`LD F0, 16 (R1)`	Anti dependence with **I3** for **F0**
I6	`LD F2, 16 (R2)`	Anti dependence with **I4** for **F2**
I7	`ADDUI R1, R1, 8`	Anti dependence with **I1** and **I5** for **R1**
I8	`ADDUI R2, R2, 8`	Anti dependence with **I2** and **I6** for **R2**
I9	`ADD R3, R2, R1`	True data dependence with **I7** for **R1** True data dependence with **I8** for **R2**
I10	`BNE R3, R4, SP_LOOP`	True data dependence with **I9** for **R3**

EXERCISE 2.B – TOMASULO (3 points)

*Consider the same 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 **4***
- *2 RESERVATION STATION **(RS3, RS4)** with 1 FP unit1 **(FPU1)** with latency **4***
- *2 RESERVATION STATION **(RS5, RS6)** with 2 INT_ALU/BR units **(ALU1, ALU2)** with latency **2***

Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`I1: SD F4, 0 (R1)`	1	2	6	None	RS1	LDU1
`I2: SD F6, 0 (R2)`	2	3	7	None	RS2	LDU2
`I3: FADD F4, F0, F0`	3	4	8	(WAR with I1 for F4 solved*)	RS3	FPU1
`I4: FADD F6, F2, F2`	4	**9**	13	**STR. FPU1** (WAR with I2 for F6 solved*)	RS4	FPU1
`I5: LD F0, 16 (R1)`	**7**	8	12	**STR. RS1** (WAR with I3 for F0 solved*)	RS1	LDU1
`I6: LD F2, 16 (R2)`	8	9	**14**	**STR. WR. CDB** (WAR with I4 for F2 solved*)	RS2	LDU2
`I7: ADDUI R1, R1, 8`	9	10	**15**	**STR. WR. CDB** (WAR with I5 for R1 solved*)	RS5	ALU1
`I8: ADDUI R2, R2, 8`	10	11	**16**	**STR. WR. CDB** (WAR with I6 for R2 solved*)	RS6	ALU2
`I9: ADD R3, R2, R1`	**16**	17	19	**STR. RS5** (RAW R1 & R2 solved)	RS5	ALU1
`I10: BNE R3,R4, SPLOOP`	17	**20**	22	**RAW with I9 for R3**	RS6	ALU2

(*) In Tomasulo, WAW and WAR hazards are already solved by Register Renaming in ISSUE stage.

(**) Only hazards that have caused the introduction of some stalls are reported in the table. Other dependencies are already solved.

*Calculate the **CPI** =* _____

CPI = # clock cycles / IC = 22 / 10 = 2.2

QUESTION 3: LOOP UNROLLING (3 points)

Let's consider the following loop code where registers **R1** and **R2** are initialized to **0** and **R3** is initialized to **400:**

```
LOOP:  LD F2, 0 (R1)
       LD F4, 0 (R2)
       FADD F6, F2, F4
       SD F6, 0 (R1)
       ADDUI R1, R1, 4
       ADDUI R2, R2, 4
       BNE R1, R3, LOOP
```

Answer to the following questions:

How many iterations of the loop?	100 iterations
How many instructions per iteration?	7 instructions
How many instructions due to the loop overhead per iteration?	3 out of 7 instructions
How many instructions are executed globally?	700 instructions
How many branch instructions are executed globally?	100 branches
Let's consider a loop unrolling version of the code with unrolling factor 2.	
How many iterations of the unrolled loop?	50 iterations
How many instructions per iteration?	(4 x 2) + 3 = 11 instructions
How many instructions due to the loop overhead per iteration?	3 out of 11 instructions
How many instructions are executed globally?	550 instructions
How many branch instructions are executed globally?	50 branches
How many more registers are needed due to register renaming?	3 more FP registers
What are the registers that need to be renamed?	F2, F4 and F6

How much is the global instruction count decrease?	(700 – 550) / 700 = 21%
Let's consider a loop unrolling version of the code with unrolling factor 4.	
How many iterations of the unrolled loop?	25 iterations
How many instructions per iteration?	(4 x 4) + 3 = 19 instructions
How many instructions due to the loop overhead per iteration?	3 out of 19 instructions
How many instructions are executed globally?	475 instructions
How many branches are executed?	25 branches
How many more registers are needed due to register renaming?	9 more FP registers
What are the registers that need to be renamed?	F2, F4 and F6 need to be renamed three times
How much is the global instruction count decrease?	(700 – 475) / 700 = 32%
What is the best unrolling factor between 2 and 4? Explain why.	

QUIZ 4 – BRANCH PREDICTION (1 point)

During the execution of a program, the number of mispredictions depends on the initialization and the size of the Branch History Table.

Answer: **TRUE** ***TRUE:*** **FALSE**

Motivate your answer:

_____

_____

_____

_____

_____

_____

QUIZ 5 – BRANCH PREDICTION (1 point)

Consider the following assembly code executed by a processor with 1-entry 1-bit Branch History Table initialized as Taken:

```
            ADDI R1, R0, 0
            ADDI R2, R0, 100

LOOP:       BEQ R1, R2, DONE
            ADD R5, R5, R4
            ADDI R1, R1, 1
            J   LOOP
DONE:
```

*Assuming **only** the branch instruction accesses the BHT:*

- *How many accesses to the Branch History Table?*
- *How many mispredictions?*

(SINGLE ANSWER)

Answer 1: 101 accesses | 2 mispredictions **(TRUE)**

Answer 2: 100 accesses | 1 misprediction

Answer 3: 99 accesses | 1 misprediction

Answer 4: 201 accesses | 2 mispredictions

Motivate your answer:

_____

_____

_____

_____

_____

_____

QUIZ 6: SUPERSCALAR PROCESSORS (3 points) OPTIONAL

Considering superscalar processors, answer **TRUE** or **FALSE** to the following questions, ***motivating your answers***.

- Doubling the number of issues reduces the CPI metric.

  Answer: **TRUE (TRUE)** **FALSE**

- Out-of-order execution can be generated by data cache misses **TRUE (TRUE)** **FALSE**

  When a data cache misses occurs, the processor should stall because of memory access, but in out of order execution, processor can execute the independent instructions instead of stall.

- The introduction of dynamic branch predictors improves the CPI metric **TRUE (TRUE)** **FALSE**

Surname	
Name	
POLIMI Personal code	
Signature	

SOLUTION

Politecnico di Milano, 6 May 2024

Course on Advanced Computer Architectures

Prof. C. Silvano

EX 1	**(4 points)**	
EX 2	**(5 points)**	
EX 3	**(4 points)**	
QUIZ 4	**(1 point)**	
QUIZ 5	**(1 point)**	
TOTAL	**(15 points)**	
QUIZ 6 OPTIONAL	**+ 3 extra points**	

EXERCISE 1: VLIW (4 points)

Let's consider the following assembly code:

```
INIT:  ADDUI $R1, $R0, 0
       ADDUI $R2, $R0, 40
       ADDUI $R4, $R0, 8

LOOP1: LD $F0, 0 ($R1)
       FADD $F4, $F0, $F2
       SD $F4, 0 ($R1)
       ADDUI $R3, $R0, 0

LOOP2: LD $F6, 0($R3)
       FADD $F8, $F6, $F2
       SD $F8, 0($R3)
       ADDUI $R3, $R3, 4
       BNE $R3, $R4, LOOP2

       ADDUI $R1, $R1, 4
       BNE $R1, $R2, LOOP1
```

1. *Complete the code of the first iteration of the outer loop* ***LOOP1:***

```
LOOP1: LD $F0, 0 ($R1)
       FADD $F4, $F0, $F2
       SD $F4, 0 ($R1)
       ADDUI $R3, $R0, 0
        LD $F6, 0($R3)
       FADD $F8, $F6, $F2
        SD $F8, 0($R3)
        ADDUI $R3, $R3, 4
        BNE $R3, $R4, LOOP2
        ADDUI $R1, $R1, 4
        BNE $R1, $R2, LOOP1
```

2. *Now schedule **the first iteration of the outer loop LOOP1** by using **the list-based scheduling** (do NOT introduce any software pipelining, loop unrolling and modifications to loop indexes) on a **3-issue VLIW** machine with **fully pipelined functional units:***

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

   ***There is no branch prediction.** The branch is completed with **1 cycle delay slot** (branch solved in ID stage).*
   In the Register File, it is possible to read and write at the same address at the same clock cycle.
   Please do not write in NOPs.

	Memory Unit	Integer Unit	Floating Point Unit
C1	LD $F0, 0 ($R1)	ADDUI $R3, $R0, 0	
C2	LD $F6. 0($R3)		
C3			FADD $F4, $F0, $F2
C4			FADD $F8, $F6, $F2
C5	SD $F4, 0($R1)		
C6	SD $F8, 0($R3)	ADDUI $R3, $R3, 4	
C7			
C8		BNE $E3, $R4, LOOP2	
C9		Br. delay slot	
C10	LD $F6, 0($R3)		
C11			
C12			FADD $F8, $F6, $F2
C13			
C14	SD $F8, 0($R3)	ADDUI $R3, $R3, 4	
C15			
C16		BNE $R3, $R4, LOOP2	
C17		Br. delay slot	
C18		ADDUI $R1, $R1, 4	
C19			
C20		BNE $R1, $R2, LOOP1	
C21		Br. delay slot	
C22			
C23			
C24			
C25			

3. How long is the critical path? _____

21

4. What performance did you achieve in CPIas?

_____

21/16

5. What performance did you achieve in FP ops per cycles?

_____

3 / 21

6. How much is the code efficiency?

_____

16 / (21 * 3)

Feedback

1. *Complete the code of the first iteration of the outer loop* ***LOOP1:***

```
LOOP1: LD $F0, 0 ($R1)
       FADD $F4, $F0, $F2
       SD $F4, 0 ($R1)
       ADDUI $R3, $R0, 0
LOOP2: LD $F6, 0($R3)
       FADD $F8, $F6, $F2
       SD $F8, 0($R3)
       ADDUI $R3, $R3, 4
       BNE $R3, $R4, LOOP2
LOOP2: LD $F6, 0($R3)
       FADD $F8, $F6, $F2
       SD $F8, 0($R3)
       ADDUI $R3, $R3, 4
       BNE $R3, $R4, LOOP2
       ADDUI $R1, $R1, 4
       BNE $R1, $R2, LOOP1
```

2. *Now schedule* ***the first iteration of the outer loop LOOP1***

	Memory Unit 1	Integer Unit	Floating Point Unit
C1	LD $F0,0($R1)	ADDUI $R3,$R0,0	
C2	LD $F6,0($R3)		
C3			FADD $F4, $F0, $F2
C4			FADD $F8, $F6, $F2
C5	SD $F4,0(R1)		
C6	SD $F8,0($R3)	ADDUI R3,R3,4	
C7			
C8		BNE $R3,$R4, LOOP2	
C9		Br. delay slot	
C10	LD $F6,0($R3)		
C11			
C12			FADD $F8, $F6, $F2
C13			
C14	SD $F8,0($R3)	ADDUI R3,R3,4	
C15			
C16		BNE $R3,$R4, LOOP2	
C17		Br. delay slot	
C18		ADDUI $R1,$R1,4	
C19			
C20		BNE $R1,$R2, LOOP1	
C21		Br. delay slot	

3. How long is the critical path?

   21 cycles

4. What performance did you achieve in CPIas?

   CPIas = (# cycles) / IC = 21 / 16 = 1,31

5. What performance did you achieve in FP ops per cycles?

   (# FP ops) / cycles = 3 / 21 = 0,14

6. How much is the code efficiency?

   Code_eff = IC/ (# cycles * # issues) = 16 / (21 * 3) = 0,25

EXERCISE 2: DYNAMIC BRANCH PREDICTION (5 points)

Let's consider the same assembly code used for **EXERCISE 1:**

```
INIT:  ADDUI $R1, $R0, 0
       ADDUI $R2, $R0, 40
       ADDUI $R4, $R0, 8

LOOP1: LD $F0, 0 ($R1)
       FADD $F4, $F0, $F2
       SD $F4, 0 ($R1)
       ADDUI $R3, $R0, 0

LOOP2: LD $F6, 0($R3)
       FADD $F8, $F6, $F2
       SD $F8, 0($R3)
       ADDUI $R3, $R3, 4
       BNE $R3, $R4, LOOP2

       ADDUI $R1, $R1, 4
       BNE $R1, $R2, LOOP1
```

1. *How many iterations for the outer loop **LOOP1?***

   _____

2. *How many iterations for the inner loop **LOOP2?***

   _____

3. *How many branch instructions are executed in the code?*

   _____

4. Assuming there is **no branch prediction** and each branch costs **2 cycle penalty** to fetch the correct instruction, *how many branch penalty cycles are needed to execute both loops?*

   _____

5. Assuming to execute the code on a pipelined processor with a dynamic **Branch Prediction Unit (BPU)** in the **IF-stage** composed of:
   - **2-entry 2-bit Branch History Table**
   - **2-entry Branch Target Buffer**

Let's assume the 2 branch instructions **do not collide** so they are allocated to the 2 entries of the BPU where the **BTB hit**, there are 4 cases for each conditional branch with the related **branch penalty cycles:**

	Branch Outcome	
	Taken	**Not Taken**
Strongly Taken	**1 cycle**	**2 cycles**
Weakly Taken	**1 cycle**	**2 cycles**
Strongly Not Taken	**2 cycles**	**0**
Weakly Not Taken	**2 cycles**	**0**

Let's assume the 2-entries do not collide with BTB hit and are initialized as **Strongly Taken**, *please complete the following table:*

Explain the branch behavior considering the inner LOOP2 in isolation.	***How many branch penalty cycles to execute the LOOP2 in isolation?***	***Calculate the branch misprediction rate to execute the LOOP2 in isolation.***
Explain the branch behavior considering both loops.	***How many branch penalty cycles to execute both loops?***	***Calculate the global branch misprediction rate to execute both loops.***

Feedback

1. *How many iterations for the outer loop **LOOP1?***
   The outer loop LOOP1 is executed **10** times.

2. *How many iterations for the inner loop **LOOP2?***
   The inner loop LOOP2 is executed 2 times for each iteration of LOOP1 => Globally LOOP2 is executed **20** times.

3. *How many branch instructions are executed in the code?*
   There are **20** branches for LOOP2 and **10** branches for BNE-LOOP1=> Globally **30** branches executed.

4. Assuming there is **no branch prediction and** each branch costs **2 cycle penalty** to fetch the correct instruction, *how many branch penalty cycles are needed to execute both loops?*
   Globally **30** branch instructions are executed introducing 2 cycles penalty each => Globally there are **60** branch penalty cycles to execute the code.

Let's assume the 2-entries do not collide with BTB hit and are initialized as **Strongly Taken**, *please complete the following table:*

Explain the branch behavior considering the inner LOOP2 in isolation.	***How many branch penalty cycles to execute the LOOP2 in isolation?***	***Calculate the branch misprediction rate to execute the LOOP 2 in isolation.***
Being the predictor initialized as **Strongly Taken,** the first iteration of LOOP2 is correctly predicted as taken, while there is a misprediction at the second iteration (exit) of the inner LOOP2 and the prediction is turned to **Weakly Taken.**	There are: (1 + 2) = 3 branch penalty cycles.	There is 1 misprediction out of 2 branch predictions => misprediction rate 50%.
Explain the branch behavior considering both loops.	***How many branch penalty cycles to execute both loops?***	***Calculate the global branch misprediction rate to execute both loops.***
As explained above, being the predictor initialized as **ST**, the first iteration of LOOP2 is correctly predicted as taken, while there is a misprediction at the second iteration (exit) of the inner LOOP2 and the prediction is turned to **WT.** Exiting from the inner LOOP2 with the prediction as **WT,** this does not influence the LOOP1 because the 2 branch instructions **do not collide.** The BHT entry used for LOOP1 is initialized as **ST,** so there are 9 iterations correctly predicted as taken, while we have a misprediction at the last iteration of the outer LOOP1 and its prediction is turned to WT. Exiting from the outer LOOP1 with the prediction as **WT,** this does not influence the LOOP2 because the 2 branch instructions **do not collide.** When re-entering in the inner LOOP2 with the its prediction as **WT,** the first iteration is taken, the prediction is turned to ST and we have a misprediction at the second iteration (exit) of the inner LOOP2 and the prediction bit is turned to **WT.**	There are: (1 + 2) = 3 BP cycles to execute LOOP2 for 10 iterations of the outer LOOP1 => **30** BP cycles. For the outer LOOP1, there are: (9 + 2) = **11** BP cycles. Globally there are **41** BP cycles.	We have 1 misprediction for the BNE-LOOP2 only at the exit of LOOP2 times 10 iterations of the outer LOOP1 => globally 10 mispredictions. For the outer LOOP1, we have only 1 mispredictions for BNE-LOOP1 at the last iteration of LOOP1 Globally (10 + 1) = 11 mispredictions, while there are 30 predictions (20 for BNE-LOOP2 and 10 for BNE-LOOP1) => 11 mispredictions out of 30 predictions => 36.67% misprediction rate.

EXERCISE 3 – SCOREBOARD (4 points)

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

INSTRUCTION	ANALYSIS OF DEPENDECES
`I0: LD $F2,A($R6)`	--
`I1: FADD $F3,$F2,$F6`	True data dependence with I0 for $F2
`I2: SD $F3,A($R7)`	True data dependence with I1 for $F3
`I3: LD $F3,B($R6)`	Anti data dependece with I2 for $F3 output data dependece with I1 for $F3
`I4: SD $F3,C($R7)`	True data dependence with I3 for $F3
`I5: ADDUI $R6,$R6,4`	Anti data dependence with I0 for $R6 Anti data dependence with I3 for $R6
`I6: ADDUI $R7,$R7,4`	Anti data dependence with I2 for $R7 Anti data dependence with I4 for $R7

2. *Schedule the code on a CPU with dynamic scheduling based on **OPTIMIZED SCOREBOARD** with the following assumptions:*
   - 2 LOAD/STORE Units (LDU1, LDU2) with latency 3 cycles
   - 1 FP Unit (FPU1) with latency 3 cycles
   - 1 ALU/BR Unit ALU1 with latency 1 cycle
   - Register Filw with 2 read ports and 1 write port
   - Check for WAR and WAW hazards postponed to the WRITE BACK phase
   - Forwarding

INSTRUCTION	ISSUE	READ OPs	EXEC COMPL.	WRITE BACK	Hazards Type Forwarding	UNIT
`I0: LD $F2,A($R6)`	1	2	5	6		LDU1
`I1: FADD $F3,$F2,$F6`	2	6	9	10	RAW $f2 by forw.	FPU1
`I2: SD $F3,A($R7)`	3	10	13	14	RAW $F3 by forw	LDU2
`I3: LD $F3,B($R6)`	7	8	11	12	Structure hazard LDU1 WAW $F3(ok), WAA $F3(ok)	LDU1
`I4: SD $F3,C($R7)`	**13**	14	17	18	structure hazard LDU2	LDU1
`I5: ADDUI $R6,$R6,4`	14	15	16	17	WAR ok	ALU
`I6: ADDUI $R7,$R7,4`	18	19	20	21	Structure hazard ALU	ALU

3. *Express the formula and calculate the CPI:*
   CPI = _____

   21 / 7 = 3

Feedback

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

INSTRUCTION	ANALYSIS OF DEPENDECES
`I0: LD $F2,A($R6)`	--
`I1: FADD $F3,$F2,$F6`	True data dependence with I0 for $f2
`I2: SD $F3,A($R7)`	True data dependence with I1 for $f3
`I3: LD $F3,B($R6)`	Output dependence with I1 for $f3 Anti dependence with I2 for $f3
`I4: SD $F3,C($R7)`	True data dependence with I3 for $f3
`I5: ADDUI $R6,$R6,4`	Anti dependence with I0, I3 for $r6
`I6: ADDUI $R7,$R7,4`	Anti dependence with I2, I4 for $r7

2. *Schedule the code on a CPU with dynamic scheduling based on OPTIMIZED SCOREBOARD with the following assumptions:*
   - 2 LOAD/STORE Units (LDU1, LDU2) with latency 3 cycles
   - 1 FP Unit (FPU1) with latency 3 cycles
   - 1 ALU/BR Unit ALU1 with latency 1 cycle
   - Register File with 2 read ports and 1 write port
   - Check for WAR and WAW hazards postponed to the WRITE BACK phase
   - Forwarding

INSTRUCTION	ISSUE	READ OPs	EXEC COMPL.	WRITE BACK	Hazards Type	UNIT
`I0: LD $F2,A($R6)`	1	2	5	6		LDU1
`I1: FADD $F3,$F2,$F6`	2	6	9	10	RAW $f2 I0 by forw	FPU1
`I2: SD $F3,A($R7)`	3	10	13	14	RAW $f3 I1 by forw	LDU2
`I3: LD $F3,B($R6)`	7	8	11	12	Check STRUCT LDU1 in ISSUE (Check WAW $f3 I1 in WB ok) (Check WAR Sf3 I2 in WB ok)	LDU1
`I4: SD $F3,C($R7)`	13	14	17	18	Check STRUCT LDU1 is ISSUE (Check RAW $f3 I3 ok)	LDU1
`I5: ADDUI $R6,$R6,4`	14	15	16	17	(Check WAR $r6 I3 in WB ok)	ALU1
`I6: ADDUI $R7,$R7,4`	18	19	20	21	Check STRUCT ALU1 in ISSUE (Check WAR $r7 I4 ok)	ALU1

3. *Calculate the CPI: CPI = (#clock cyles / IC) = 21/7 = 3*

QUIZ 4 – SPECULATIVE TOMASULO (1 point)

In the speculative Tomasulo architecture, exceptions are taken when the instruction that generated them reaches the head of the ROB.
(TRUE/FALSE ANSWER)
1 point

Answer: **TRUE** **FALSE**

Motivate your answer:

_____

_____

_____

Feedback:
*The answer is **TRUE**: When the instruction that has generated the exceptions has reached the head of the ROB, it is ready to commit. This means that the instruction is no longer speculative and all its previous instructions have already been committed. This mechanism represents a precise interrupt/exception model.*

QUIZ 5 – CACHE PERFORMANCE (1 point)

Let us consider a computer architecture with L1 and L2 caches with the following parameters:

- Processor Clock Frequency = 1 GHz
- Hit Time L1 = 1 clock cycle
- Hit Rate L1 = 95%
- Hit Time L2 = 5 clock cycles
- Hit Rate L2 = 90%
- Miss Penalty L2 = 15 clock cycles

How much is the Global Miss Rate for Last Level Cache?
(SINGLE ANSWER)
1 point

Answer 1: 5% **T**

Answer 2: 0.5% **T (TRUE)**

Answer 3: 10% **T**

Answer 4: 7.5% **T**

Answer 5: 25% **T**

Motivate your answer:

_____

_____

_____

Feedback:

Global Miss Rate = Miss Rate $_{L1\ L2}$ = Miss Rate$_{L1}$ x Miss Rate$_{L2}$= 0.05 x 0.1 = 0.005 **=** 0.5%

QUIZ 6: CACHE MEMORIES (3 points) OPTIONAL

Given a cache of a given capacity, associativity and block size, answer **TRUE** or **FALSE** to the following questions, ***motivating your answers***.

- Doubling the cache capacity of a direct mapped cache usually reduces conflict misses

  Answer: **TRUE (TRUE)** **FALSE**

_____

_____

_____

_____

- Doubling the block size reduces compulsory misses **TRUE (TRUE)** **FALSE**

_____

_____

_____

_____

- Change the nesting of loops in the code to access data in order stored in memory will increase spatial locality, possibly reducing the miss rate **TRUE (TRUE)** **FALSE**

_____

_____

_____

_____

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

Politecnico di Milano, 19 June, 2024

Course on Advanced Computer Architectures

Prof. C. Silvano

EX1	**(5 points)**	
EX2	**(5 points)**	
EX3	**(5 points)**	
Q1	**(4 points)**	
Q2	**(6 points)**	
QUIZZES	**(8 points)**	
TOTAL	**(33 points)**	

EXERCISE 1 – SCOREBOARD (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 $F2, 0($R1)`	None
I1	`LD $F4, 0($R2)`	None
I2	`FADD $F4,$F2,$F4`	True data dependence with I0 for $F2 True data dependence with I1 for $F4 Output dependence with I1 for $F4
I3	`SD $F4,0($R1)`	True data dependence with I2 for $F4
I4	`ADDUI $R1,$R1,4`	Anti dependence with I0 for $R1 Anti dependence with I3 for $R1
I5	`ADDUI $R2,$R2,4`	Anti dependence with I1 for $R2
I6	`BNE $R1,$R3,LOOP`	True data dependence with I4 for $R1

2. *Schedule the code on a CPU with dynamic scheduling based on **OPTIMIZED SCOREBOARD** with the following assumptions:*
   - 2 LOAD/STORE Units (LDU1, LDU2) with latency 3 cycles
   - 1 FP Unit (FPU1) with latency 4 cycles
   - 2 ALU/BR Units (ALU1, ALU2) with latency 1 cycle
   - Register File with 2 read ports and 1 write port
   - **Check for WAR and WAW hazards postponed to the WRITE BACK phase**
   - **Forwarding**

Feedback

INSTRUCTION	ISSUE	READ OPs	EXEC COMPL.	WRITE BACK	Hazards Type	UNIT
`LOOP:LD $F2, 0($R1)`	1	2	5	6	--	LDU1
`LD $F4, 0($R2)`	2	3	6	7	--	LDU2
`FADD $F4,$F2,$F4`	3	7	11	12	**RAW $F4 I1 solved by forw.** (Check RAW $F2 I0 in WB ok) (Check WAW $F4 I1 in WB ok)	FPU1
`SD $F4,0($R1)`	7	12	15	16	**Check STRUCT LDU1 in ISSUE** **RAW $F4 I2 solved by forw.**	LDU1
`ADDUI $R1,$R1,4`	8	9	10	13	**WAR $R1 with I3** (Check WAR $R1 I0 in WB ok)	ALU1
`ADDUI $R2,$R2,4`	9	10	11	14	**Check STRUCT RF write** (Check WAR $R2 I1 in WB ok)	ALU2
`BNE $R1,$R3,LOOP`	14	15	16	17	**Check STRUCT ALU1 in ISSUE** (Check RAW $R1 I5 ok)	ALU1

Calculate the CPI: CPI = (#clock cyles / IC) = 17/7 = 2.43

EXERCISE 2 – VLIW SCHEDULING (5 points)

Let's consider the following LOOP code, where $**Ri** are integer registers and **Fi** are floating-point registers.

```
L1: ADDI R1, R2, R3
    SUBI R4, R1, R5
    MULI R6, R7, R8
    FADD F1, F2, F3
    FSUB F4, F1, F5
    LD R9, MEM[R4]
    SD R6, MEM[R10]
    SD F4, MEM[R11]
    BNEZ R9, L1
```

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

- **1 Memory Units with 3 cycles latency**
- **1 FP ALUs with 3 cycles latency**
- **1 Integer ALU with 1 cycle latency to next Int/FP/L/S & 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	Floating Point Unit	Integer Unit
C1		FADD F1, F2, F3	ADDI R1, R2, R3
C2			SUBI R4, R1, R5
C3	LD R9, MEM[R4]*		MULI R6, R7, R8
C4	SD R6, MEM[R10]*	FSUB F4, F1, F5	
C5			
C6			BNEZ R9, L1
C7	SD F4, MEM[R11]		(br. delay slot)
C8			
C9			
C10			
C11			
C12			
C13			
C14			
C15			

(*) The pair of LD and SD instructions can be anticipated by one cycle without impacting the critical path.

1. How long is the critical path? _____

   9 cycles (to complete the SD)

2. What performance did you achieve in CPIas?

   _____

   CPIas = #cycles / IC = 7 / 9 = 0.78

   Note that the CPI of one single iteration is CPI = #cycles / IC = 9 / 9 = 1

3. What performance did you achieve in FP ops per cycles?

   _____

   FPops / cycles = 2 / 9

4. How much is the code efficiency?

   _____

   Code_eff = IC / (#cycles * #issues) = 9 / (9 x 3) = 1 / 3 = 0.3 (for one single iteration)

5. Assuming to have **2 INTEGER UNITS** how much is the impact on CPI and code efficiency?

The critical path is still the same, therefore the CPI is not impacted.

The code efficiency becomes worst:

Code_eff = IC / (#cycles * #issues) = 9 / (9 x 4) = 1 / 4 = 0.25

EXERCISE 3 – MESI PROTOCOL (5 points)

Let's consider the following access patterns on a **2-processor** system with a direct-mapped, write-back cache with one cache block per processor and a two-cache block memory.

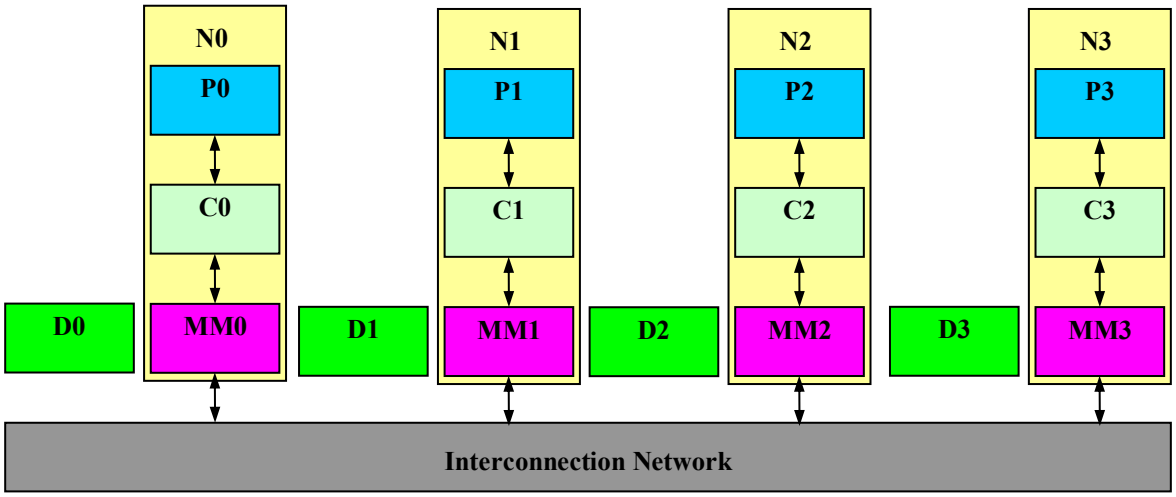
Assume the **MESI protocol** is used, with **write-back** caches, **write-allocate,** and **write-invalidate** of other caches.

Please complete the following table:

Cycle	After Operation	Hit/ Miss	P0 cache block state	P1 cache block state	Memory at bl. 0 up to date?	Memory at bl. 1 up to date?	Write Invalidation sent?
0	**P0: Read Bl. 1**	Miss	Exclusive(1)	Invalid	Yes	Yes	No
1	**P1: Read Bl. 0**	Miss	Exclusive(1)	Exclusive(0)	Yes	Yes	No
2	**P0: Write Bl. 1**	Hit	Modified (1)	Exclusive(0)	Yes	No	No
3	**P0: Write Bl. 0**	Miss	Modified (0)	Invalid	No	Yes	Yes
4	**P1: Read Bl. 1**	Miss	Modified (0)	Exclusive(1)	No	Yes	No
5	**P1: Write Bl. 1**	Hit	Modified (0)	Modified (1)	No	No	No
6	**P0: Read Bl. 1**	Miss	Shared (1)	Shared (1)	Yes	Yes	No
7	**P0: Write Bl. 1**	Hit	Modified (1)	Invalid	Yes	No	Yes
8	**P1: Write Bl. 1**	Miss	Invalid	Modified (1)	Yes	No	Yes

QUESTION 1: DIRECTORY-BASED PROTOCOL (*4 points*)

Let's consider a directory-based protocol for a distributed shared memory system with 4 nodes (N0, N1, N2, N3) where: **Directory N1 Block B1 | State: Modified | Sharer Bits: 0001**



1. **After the message Write Miss on B1 sent by node N2,** please answer to the following questions:

Which is the home node?	**N1**
Which is the local node?	**N2** is the local node sending the request
Which is the remote node?	**N3** is the remote node where there is the most updated copy of the block B1
What is the message sent between the home node and the remote note?	**Fetch/Invalidate** message is sent from home N1 to remote N3 (past owner) to ask to fetch the most updated copy of the block B1 in the home directory of N1 and to invalidate the block B1 in the past owner's cache C3 in N3.
What is are the other messages sent between the nodes?	**1)** **Data Write Back** message reply from remote cache N3 to the home node N1 to write back the most updated copy of the block B1 in the home directory of N1.

	2) **Data Value Reply** message sent from the home node N1 to the local node N2 to get the most updated copy of the block B1 in the cache C2 to make the write. Then the node N2 becomes the new owner of block B1.
At the end of the above sequence of messages:	
Which is the coherence state and sharer bits of the block B1 in the home directory?	Directory N1 Block B1 | State: **Modified** | Sharer Bits: **0010**
Which node is the new owner of the block B1?	**N2**
Which is the coherence state of the block B1 in the Local Cache?	**Modified** in the Local Cache C2 of node N2.

QUESTION 2: LOOP UNROLLING (6 points)

Let's consider the following loop code where registers **$R1** and **$R2** are initialized to **0** and **$R3** is initialized to **400:**

```
LOOP:  LD $F2, 0 ($R1)
       LD $F4, 0 ($R2)
       FADD $F6, $F2, $F4
       SD $F6, 0 ($R1)
       ADDDUI $R1, $R1, 4
       ADDDUI $R2, $R2, 4
       BNE $R1, $R3, LOOP
```

Answer to the following questions:

How many iterations of the loop?	100 iterations
How many instructions per iteration?	7 instructions per iteration
How many instructions dedicated to the loop overhead per iteration?	3 out of 7 instructions per iteration
How many instructions are executed globally?	700 instructions
How many branch instructions are executed globally?	100 branches (one branch per iteration)
Let's consider a loop unrolling version of the code with unrolling factor 2.	
How many iterations of the unrolled loop?	50 iterations
How many instructions per iteration?	(4 x 2) + 3 = 11 instructions per iteration
How many instructions dedicated to the loop overhead per iteration?	3 out of 11 instructions per iteration
How many instructions are executed globally?	550 instructions

How many branch instructions are executed globally?	50 branches (one branch per iteration)
How many more registers per iteration are needed due to register renaming?	3 more FP registers are needed to avoid the WAW hazards on $F2, $F4 and $F6
What are the registers that need to be renamed?	Registers $F2, $F4 and $F6 must be first used and then renamed in the second replication of the original iteration in the unrolled loop.
How much is the global instruction count decrease?	(700 – 550) / 700 = 21.43%
Let's consider a loop unrolling version of the code with unrolling factor 4.	
How many iterations of the unrolled loop?	25 iterations
How many instructions per iteration?	(4 x 4) + 3 = 19 instructions per iteration
How many instructions due to the loop overhead per iteration?	3 out of 19 instructions per iteration
How many instructions are executed globally?	475 instructions
How many branches are executed?	25 branches (1 branch per iteration)
How many more registers per iteration are needed due to register renaming?	(3 x 3) = 9 more FP registers are needed to avoid the WAW hazards on $F2, $F4 and $F6
What are the registers that need to be renamed?	Registers $F2, $F4 and $F6 must be first used and then renamed in the next three replications of the original iteration in the unrolled loop.
How much is the global instruction count decrease?	(700 – 475) / 700 = 32%
What is the best unrolling factor between 2 and 4? Explain why	The best is the unrolling factor 4 if we consider the global instruction count decrease, the reduced number of branches, the increase of the length of the basic block that can be used to better exploit the ILP. The drawback is the higher register pressure due to the higher register renaming.

MULTIPLE-CHOICE QUESTIONS: (8 points)

Question 1 (format Multiple Choice – Multiple answer)

Let's consider the following loop:

```
for (i=1; i<=100, i++) {
     A[i] = A[i] + A[i-1];     /*S1*/
     C[i] = A[i] + B[i-1]      /*S2*/
     D[i] = A[i] + D[i-1]      /*S3*/
}
```

How many loop-carried dependencies are in the code?

(MULTIPLE ANSWERS)
1 point

Answer 1: One in S2 because C[i] depends on B[i-1];

Answer 2: One in S3 because D[i] depends on D[i-1]; (**TRUE**)

Answer 3: One in S1 because A[i] depends on A[i-1]; (**TRUE**)

Answer 4: One in S2 because C[i] depends on A[i];

Motivate your answers:
1 point

_____

_____

_____

_____

Feedback:
The dependence of C[i] on B[i-1] in S2 is not a loop-carried dependence because the vector B[ ] is never modified in the loop.
The dependence of D[i] on D[i-1] in S3 is a loop-carried dependence
The dependence of A[i] on A[i-1] in S1 is a loop-carried dependence
Dependences of C[i] on A[i] in S2 and D[i] on A[i] in S3 are not loop-carried dependences.

Question 2 (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, R1              # Load vector from mem. address R1 into V1
MULVS.D V1, V1, S1      # FP multiply vector V1 to scalar F1
ADDVV.D V2, V1, V1      # FP add vectors V1 and V1
MULVS.D V2, V2, S2      # FP multiply vector V2 to scalar F2
L.V V3, R2              # Load vector from mem. address R2 into V3
ADDVV.D V3, V2, V3      # FP add vectors V2 and V3
S.V V3, R1              # Store vector V3 into memory address R1
```

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

Answer 1: 3 convoys; 24 clock cycles **(TRUE)**

Answer 2: 2 convoys; 16 clock cycles

Answer 3: 4 convoys; 32 clock cycles

Answer 4: 5 convoys; 40 clock cycles

Motivate your answer by completing the following figure:
1 point

0	*1*	*… L.V V1, RX …*	*7*

Question 3 (format Multiple Choice – Single answer)

*Let's consider a **Speculative Tomasulo** architecture with:*

- *2 load buffers (Load1, Load2),*
- *2 FP reservation stations (FPS1 and FPS2)*
- *2 Integer reservation stations (RS1 and RS2)*
- ***8-entry ROB** (ROB0, ROB1, …, ROB7).*

Let's consider the following LOOP after the issue of the instructions of the first iteration (while the first load is executing a cache miss).

```
LOOP: LD $F2, 0 ($R1)
      LD $F4, 0 ($R2)
      FADD $F4, $F2, $F4
      SD $F4, 0 ($R1)
      SUBI $R1, $R1, 8
      BNEZ $R1, LOOP   // branch prediction taken
```

In the Rename Table, what are the pointers used for **$F2** *and* **$F4?**

(SINGLE ANSWER)
1 point

Answer 1: ROB0 for $F2 and ROB1 for $F4

Answer 2: ROB0 for $F2 and ROB2 for $F4**(TRUE)**

Answer 3: ROB0 for $F2 and ROB3 for $F4

Answer 4: ROB0 for $F2 and ROB4 for $F4

Is the ROB full? If not, please explain whether or not the second iteration of the loop can start to issue speculatively. Motivate your answers:
1 point

Feedback:
The 8-entry ROB is not full because there are two free entries. However, the second iteration cannot be issued speculatively (assuming the branch is predicted as taken) because the structural hazard related to the two load buffers that are already busy.

ROB#	Instruction	Dest.	Ready	
ROB0	LD $F2, 0 ($R1) (1^ iteration exec. cache miss)	$F2	No	HEAD
ROB1	LD $F4, 0 ($R2) (1^ iteration issued)	$F4	No	
ROB2	FADD $F4, $F2, $F4 (1^ iteration issued)	$F4	No	
ROB3	SD $F4, 0 ($R1) (1^ iteration issued)	MEM	No	
ROB4	SUBI $R1, $R1, 8 (1^ iteration issued)	$R1	No	
ROB5	BNEZ $R1, LOOP (1^ branch predicted as taken)		No	
ROB6				
ROB7				

Rename Table:

$F0	
$F2	ROB0
$F4	~~ROB1~~ ROB2

Question 4 (format Multiple Choice – Single answer)

Let's consider a loop code iterated 50 times executed by a processor with a 1-entry 1-bit Branch History Table initialized as not-taken.
*How much is the **branch misprediction rate** of the loop code with one conditional branch that takes the execution back to the beginning of the loop (loop-backward branch)?*

(SINGLE ANSWER)
1 point

Answer 1: 2%

Answer 2: 4% (TRUE)

Answer 3: 96 %

Answer 4: 5%

Answer 5: 95 %

Feedback
Being the predictor initialized as NT, we have 2 mispredictions out of 50 branch predictions: one at the first execution of the loop-back branch and one at the loop exit (4% mispredictions).

Question 5 (format Multiple Choice – Single answer)

Let's consider a quad-issue SMT processor that can manage up to 8 simultaneous threads.
What are the values of the ideal CPI and the ideal per-thread CPI?

(SINGLE ANSWER)
1 point

Answer 1: Ideal CPI = 1 & Ideal per-thread CPI = 0.125
Answer 2: Ideal CPI = 0.25 & Ideal per-thread CPI =4
Answer 3: Ideal CPI = 0.5 & Ideal per-thread CPI = 2
Answer 4: Ideal CPI = 0.25 & Ideal per-thread CPI = 2 (TRUE)
Answer 5: Ideal CPI = 0.25 & Ideal per-thread CPI = 8

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

Politecnico di Milano, 15 July, 2024

Course on Advanced Computer Architectures

Prof. C. Silvano

EX1	**(5 points)**	
EX2	**(5 points)**	
EX3	**(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	`FOR:LD $F2, A($R1)`	None
I1	`FADD $F2, $F2, $F2`	True data dependence with **I0** for $**F2** Output data dependence with **I0** for **$F2**
I2	`FADD $F2, $F2, $F4`	True data dependence with **I1** for **$F2** Output data dependence with **I1** for **$F2** Anti-dependence with **I1** for **$F2**
I3	`SD $F2, A($R1)`	True data dependence with **I2** for **$F2**
I4	`ADDUI $R1, $R1, 8`	Anti dependence with **I0** for **$R1** Anti dependence with **I3** for **$R1**
I5	`BNE $R1, $R2, FOR`	True data dependence with I4 for **$R1**

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 **4***
- *2 RESERVATION STATION **(RS3, RS4)** with 1 FP unit1 **(FPU1)** with latency **3***
- *2 RESERVATION STATION **(RS5, RS6)** with 1 INT_ALU/BR unit **(ALU1)** with latency **2***

Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`FOR:LD $F2, A($R1)`	1	2	6	None	RS1	LDU1
`I1: FADD $F2, $F2, $F2`	2	**7**	10	RAW I0 for $F2	RS3	FPU1
`I2: FADD $F2, $F2, $F4`	3	**11**	14	RAW with I1 for $F2 / STR. FPU1	RS4	FPU1
`I3: SD $F2, A($R1)`	4	**15**	19	RAW with I2 for $F2	RS2	LDU2
`I4: ADDUI $R1, $R1, 8`	5	6	8	None (WAR solved by Reg. Renaming**)	RS5	ALU1
`I5: BNE $R1, $R2, FOR`	6	**9**	11	RAW with I4 for $R1/STR. ALU1	RS6	ALU1

(*) 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 = 19 / 6 = 3,17***

EXERCISE 2 – VLIW SCHEDULING (5 points)

Let's consider the following LOOP code, where $**Ri** are integer registers and $**Fi** are floating-point registers.

```
LOOP:  LD $F2, 0 ($R1)
       LD $F4, 0 ($R2)
       FADD $F4, $F4, $F4
       FADD $F6, $F2, $F2
       FADD $F8, $F0, $F0
       SD $F6, 0 ($R1)
       SD $F8, 0 ($R2)
       ADDDUI $R1, $R1, 4
       ADDDUI $R2, $R2, 4
       BNE $R1, $R3, LOOP
```

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

- **1 Memory Units with 3 cycles latency**
- **1 FP ALUs with 3 cycles latency**
- **1 Integer ALU with 1 cycle latency to next Int/FP/L/S & 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	Floating Point Unit	Integer Unit
C1	LD $F2, 0 ($R1)	FADD $F8, $F0, $F0	
C2	LD $F4, 0 ($R2)		
C3			
C4	SD $F8, 0 ($R2)	FADD $F6, $F2, $F2	ADDDUI $R2, $R2, 4
C5		FADD $F4, $F4, $F4	
C6			
C7	SD $F6, 0 ($R1)		ADDDUI $R1, $R1, 4
C8			
C9			BNE $R1, $R3, LOOP
C10			(br. delay slot)
C11			
C12			

How long is the critical path for a single iteration? _____ 10 cycles

How much is the code efficiency for a single iteration? _____

Code_eff = IC / (#cycles * #issues) = 10 / (10 x 3) = 1 / 3 = 0.3

EXERCISE 4 – CACHE MEMORIES (5 points)

Let's consider a 32-block main memory with a **4-way set associative** 8-block cache based on a **write allocate** with **write-back** protocol with **LRU.**

The addresses are expressed as: Memory Addresses: [0, 1, 2, ……. 31] $_{10}$
Cache Addresses: [a, b, c, d, e, f, g, h]
and Cache Tags are expressed as binary numbers.

How many sets?

Answer 1: 2 **(TRUE)**

Answer 2: 4

Answer 3: 8

Answer 3: 16

At cold start the cache is empty, then there is the following sequence of memory accesses.
Please complete the following table:

	Type of memory access	Memory Address	Hit/ Miss Type	Cache Tag	Set	Cache Address	Dirty bit	Write in memory
1	Write	[16] $_{10}$	Cold-start Miss	[1000] $_2$	[0] $_{10}$	a	1	No
2	Write	[14] $_{10}$	Cold-start Miss	[0111] $_2$	[0] $_{10}$	b	1	No
3	Write	[07] $_{10}$	Cold-start Miss	[0011] $_2$	[1] $_{10}$	e	1	No
4	Read	[14] $_{10}$	Hit	[0111] $_2$	[0] $_{10}$	b	1	No
5	Write	[16] $_{10}$	Hit	[1000] $_2$	[0] $_{10}$	a	1	No
6	Read	[12] $_{10}$	Cold-start Miss	[0110] $_2$	[0] $_{10}$	c	0	No
7	Write	[18] $_{10}$	Cold-start Miss	[1001] $_2$	[0] $_{10}$	d	1	No
8	Read	[06] $_{10}$	Conflict Miss	[0011] $_2$	[0] $_{10}$	b	0	Yes Wr. in M[14] $_{10}$
9	Write	[04] $_{10}$	Conflict Miss	[0010] $_2$	[0] $_{10}$	a	1	Yes Wr. in M[16] $_{10}$
10	Write	[30] $_{10}$	Conflict Miss	[1111] $_2$	[0] $_{10}$	c	1	No

How much is the miss rate?

Number of misses / Number of memory accesses = 8 / 10 = 0.8

Feedback

The 2-way set-associative cache has 8-blocks [a, b, c, d, e, f, g, h] organized in **2 sets,** where each set contains 4 blocks as follows: **Set_0: [a , b, c, d]; Set_1: [e, f, g, h]**

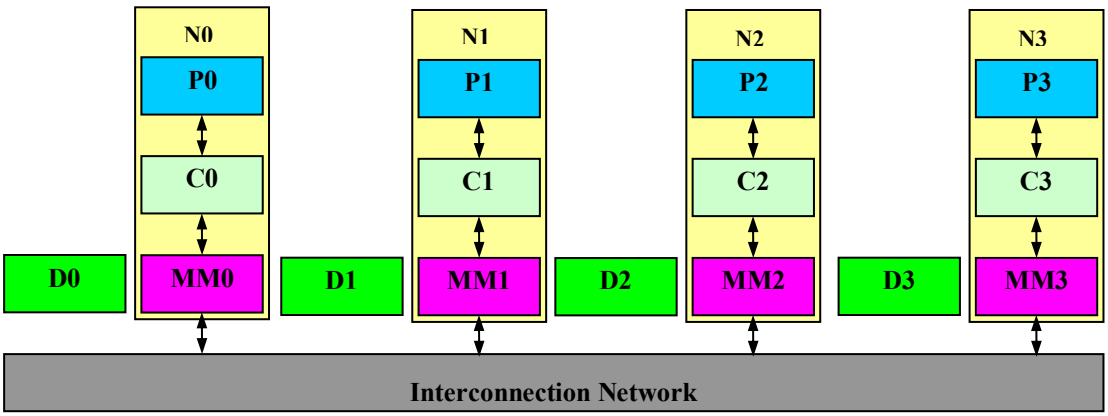
Being a set-associative cache, the block replacement policy in each set uses the **LRU algorithm**.

Memory Address Mapping:

Set_0 [a , b, c, d]	Set_1 [e, f, g, h]
0	1
2	3
4	5
…	…
30	31

QUESTION 1: DIRECTORY-BASED PROTOCOL (*5 points)*

Consider a directory-based protocol for a distributed shared memory system with **4 nodes (N0, N1, N2, N3):**



Please answer to the following questions:

What is the definition of **home node**?	The home node is the node where the memory block and the related directory entry reside.
What is the definition of **local node**?	The local node is the node sending the read/write request.
What is the definition of **remote node**?	The remote node is where there is a copy of the memory block, whether shared or modified. If modified, this is the most updated copy of the block.
What are the possible messages sent from the **local node** to the **home node**?	**Read Miss** when the local node tries to read a block that is not present in the local cache. **Write Miss** when the local node tries to write a block that is not present in the local cache. **Request to Invalidate** sent from local to home to send and invalidate the copies in the remote caches (due to a **Write Hit).**
What are the possible messages sent from the **home node** to the **local node**?	**Data Value Reply** message sent from the home node to the local node to return data value to the requesting local node (This is a reply to a **Read/Write Miss**).

What are the possible messages sent from the **home node** to the **remote node**?	**Fetch** message is sent from home to remote (past owner) to ask to fetch the most updated copy of the block in the home directory (This is a reply to a **Read Miss**) **Invalidate** message is sent to invalidate the shared copy(ies) of the block remote cache(s) (This is a reply to a **Write Hit**) **Fetch/Invalidate** message is sent from home to remote (past owner) to ask to fetch the most updated copy of the block B1 in the home directory of N1 and to invalidate the block B1 in the past owner's cache C3 in N3. (This is a reply to a **Write Miss**)
What are the possible messages sent from the remote node to the home node?	**Data Write Back** message reply from remote cache (owner) to the home node to write back the most updated copy of the data in the home directory. (This is a reply either to a **Fetch** or to a **Fetch/Invalidate**).
What are the possible **coherence states** of a block in the local cache?	1) Invalid 2) Shared 3) Modified
What are the possible **coherence states** of a block in the home directory?	1) Uncached 2) Shared 3) Modified
What is the meaning of the **sharer bits** of a block in the home directory?	If the block is in Shared state, the sharer bit(s) are set to 1 to indicate which processor(s) has(have) a copy of the block. If the block is in Modified state, a single sharer bit is set to 1 to indicate which processor is the owner of the block.

QUESTION 2: MULTITHREADING (5 points)

Let's consider the fine-grained and simultaneous multithreading techniques used to manage thread-level parallelism by hardware processors. *Answer to the following questions:*

<table>
<tr><th></th><th>Fine-grained Multithreading</th><th>Simultaneous Multithreading</th></tr>
<tr><td>Explain the main concepts for each technique.</td><td>In fine-grained MT, the processor switches between threads on each instruction by hardware context switch.

Execution of multiple threads is interleaved in a sort of round-robin fashion, skipping any thread that is stalled at that time.

Suitable for single- and multiple-issue dynamic scheduled processors.</td><td>Suitable for multiple-issue dynamic scheduled processors (such as 4-issues superscalar processors).

As in fine-grained MT, the processor switches between threads on each instruction by hardware context switch and simultaneously schedule instructions execution from several threads in the same cycle.

This is useful to maximize the usage of parallel functional units in multiple-issues processors.

SMT exploits both ILP and TLP in dynamic scheduled processors.</td></tr>
<tr><td>For each technique, make an example with up to 4 threads (T1, T2, T3, T4) on a quad-issue processor</td><td>
<table>
<tr><td>C0</td><td>T1</td><td>T1</td><td>T1</td><td></td></tr>
<tr><td>C1</td><td>T2</td><td></td><td></td><td></td></tr>
<tr><td>C2</td><td>T3</td><td>T3</td><td></td><td></td></tr>
<tr><td>C3</td><td>T4</td><td>T4</td><td>T4</td><td>T4</td></tr>
<tr><td>C4</td><td>T1</td><td>T1</td><td></td><td></td></tr>
<tr><td>C5</td><td>T3</td><td>T3</td><td>T3</td><td>T3</td></tr>
<tr><td>C6</td><td>T4</td><td>T4</td><td></td><td></td></tr>
<tr><td>C7</td><td>T1</td><td>T1</td><td></td><td></td></tr>
<tr><td>C8</td><td>T2</td><td>T2</td><td>T2</td><td></td></tr>
<tr><td>C9</td><td>T3</td><td>T3</td><td></td><td></td></tr>
</table>
</td><td>
<table>
<tr><td>C0</td><td>T1</td><td>T2</td><td>T2</td><td>T3</td></tr>
<tr><td>C1</td><td>T3</td><td>T4</td><td></td><td></td></tr>
<tr><td>C2</td><td>T1</td><td>T2</td><td>T2</td><td></td></tr>
<tr><td>C3</td><td>T3</td><td>T3</td><td>T3</td><td>T3</td></tr>
<tr><td>C4</td><td>T1</td><td>T2</td><td></td><td></td></tr>
<tr><td>C5</td><td>T1</td><td></td><td></td><td></td></tr>
<tr><td>C6</td><td>T1</td><td>T2</td><td>T2</td><td></td></tr>
<tr><td>C7</td><td>T3</td><td>T4</td><td></td><td></td></tr>
<tr><td>C8</td><td>T1</td><td>T2</td><td>T3</td><td>T4</td></tr>
<tr><td>C9</td><td>T3</td><td>T3</td><td>T4</td><td></td></tr>
</table>
</td></tr>
<tr><td>Let's consider a quad-issue SMT processor that can manage up to 4 threads</td><td></td><td>How much is the ideal CPI?

Ideal CPI = 0.25

How much is the ideal per-thread CPI?

Ideal per-thread CPI = 1</td></tr>
</table>

What are the main benefits for each technique?	It can hide both short and long stalls because instructions from other threads are executed when one thread stalls. The frequent switching among threads helps to reduce uniformly the penalty due to stalls because a control/data dependency can be solved during the cycles used by another thread. More fairness: All threads can begin/continue their execution quite uniformly, so there is no individual thread "left behind". Fine-grain (but also coarse-grain) can be applied even to a simple single issue processor core that can be combined in a multicore.	Suitable to maximize the use of parallel functional units available in the multiple-issues by different threads. Exploit more parallelism than coarse- and fine-grained MT. Minimize the number of issue slots that are left unused at the same clock cycle by coarse- and fine-grain MT.
What are the main drawbacks for each technique?	Compared to coarse-grain MT, it slows down the execution of individual threads, since a thread will be delayed by another thread even if it is ready to execute. Higher overhead due to more frequent HW context switch.	In large multiple-issue processors (such as 4-issue) requires complex dynamic control logic to keep busy the multiple-slots. Higher overhead and power consumption. Obviously, it is limited by the number of issues available in the processor.

QUIZZES

Question 1 (format Multiple Choice – Single answer)

Let's consider the following loop:

```
for (i=1; i<=100, i++) {
      X[i] = X[i] + X[i-1];     /*S1*/
      Z[i] = X[i] + Y[i-1]      /*S2*/
}
```

How many loop-carried dependencies are in the code?
(SINGLE ANSWER)
1 point

Answer 1: Only one in S2 because Z[i] depends on Y[i-1];
Answer 2: Only one in S1 because X[i] depends on X[i-1]; (**TRUE**)

Answer 3: Both of them

Feedback:
The dependence of Z[i] on Y[i-1] in S2 is not a loop-carried dependence because the vector Y[ ] is never modified in the loop.

Question 2 (format Multiple Choice – Single answer)

Let's consider the following memory hierarchy addressed at word-level (32-bit):
- main memory size = 4 Giga word
- 4-way set-associative cache with cache size = 1 Mega word and block size = 256 word

Please indicate the structure of the 32-bit memory address:

(SINGLE ANSWER)
1 point

Answer 1: |12-bit tag | 12-bit index | 8-bit offset |

Answer 2: | 24-bit tag | 8-bit offset |

Answer 3: |14-bit tag | 10-bit index | 8-bit offset | (**TRUE**)

Answer 4: |13-bit tag | 11-bit index | 8-bit offset |

Question 3 (format Multiple Choice – Single answer)

Let's consider a dual-issue SMT processor that can manage up to 4 simultaneous threads.
What are the values of the ideal CPI and the ideal per-thread CPI?

(SINGLE ANSWER)
1 point

Answer 1: Ideal CPI = 1 & Ideal per-thread CPI = 0.25

Answer 2: Ideal CPI = 0.25 & Ideal per-thread CPI =0.25

Answer 3: Ideal CPI = 0.5 & Ideal per-thread CPI = 2 (**TRUE**)

Answer 4: Ideal CPI = 0.5 & Ideal per-thread CPI = 1

Answer 5: Ideal CPI = 0.25 & Ideal per-thread CPI = 4

Question 4 (format Multiple Choice – Single answer)

Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where: **Directory N1 Block B1 | State: Shared | Sharer Bits: 1001**
Which is the state of the block B1 in N1 after this sequence:
Read Miss B1 from local cache N1;
Read Miss B1 from local cache N2;
Write Hit B1 from local cache N1 ;

(SINGLE ANSWER)
1 point

Answer 1: Directory N1 Block B1 | State: Shared | Sharer Bits: 1111

Answer 2: Directory N1 Block B1 | State: Modified | Sharer Bits: 0100 (**TRUE**)

Answer 3: Directory N1 Block B1 | State: Shared | Sharer Bits: 0110

Answer 4: Directory N1 Block B1 | State: Modified | Sharer Bits: 0110

Question 5 (format Multiple Choice – Single answer)

Let's consider the following code:

```
for (i=0; i<260; i++)
Y[i] = X[i] + Y[i];
```

Which code transformation can be applied to be executed by the VMIPS processor with a Vector Register File composed of 8 registers of 64 elements and 64 bits/element?

(SINGLE ANSWER)
1 point

Answer 1: Trace scheduling

Answer 2: Software pipeling

Answer 3: Strip mining (**TRUE**)

Answer 4: Apply a mask register

Motivate your answers:
1 point

Feedback:
Because the number N of loop iterations is greater than the physical vector length (64 elements), the Strip Mining technique can be applied to vectorize a "for loop" of N iterations.
Strip Mining is a sort of loop unrolling technique where the length of the first segment (header) is the remainder (N mod MVL) and all the subsequent segments correspond to MVL (Max Vector Length).
In this code, N=200 and MVL= 64, therefore the header is managing the first 4 iterations by scalar instructions, then there are 4 vector instructions of 64 iterations each for a total number of:
4 + (4 x 64) = 260 iterations.

We need to use 2 vector registers: X and Y

Question 6 (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 16 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, R1              # Load vector from mem. address R1 into V1
L.V V3, R2              # Load vector from mem. address R2 into V3
ADDVS.D V2, V1, S1      # FP add vector V1 to scalar S1
MULVS.D V2, V2, S2      # FP multiply vector V2 to scalar S2
ADDVS.D V3, V3, S3      # FP add vector V3 to scalar S3
S.V V2, R1              # Store vector V2 into memory address R1
S.V V3, R2              # Store vector V3 into memory address R2
```

How many convoys? How many clock cycles to execute the code?

(SINGLE ANSWER)
1 point

Answer 1: 3 convoys; 48 clock cycles

Answer 2: 2 convoys; 32 clock cycles

Answer 3: 4 convoys; 64 clock cycles **(TRUE)**

Motivate your answer by completing the following figure:
1 point

0	*1*	*… L.V V1, R1 …*	*15*

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

Politecnico di Milano, 11 September, 2024

Course on Advanced Computer Architectures

Prof. C. Silvano

EX1	**(5 points)**	
EX2	**(5 points)**	
EX3	**(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	`FOR:LD $F2, A($R1)`	None
I1	`FADD $F2, $F2, $F2`	True data dependence with **I0** for $**F2** Output data dependence with **I0** for **$F2**
I2	`FADD $F4, $F4, $F0`	None
I3	`SD $F2, A($R1)`	True data dependence with **I1** for **$F2**
I4	`SD $F4, B($R1)`	True data dependence with **I2** for **$F4**
I5	`ADDUI $R1, $R1, 8`	Anti dependence with **I0** for **$R1** Anti dependence with **I3** for **$R1** Anti dependence with **I4** for **$R1**
I6	`BNE $R1, $R2, FOR`	True data dependence with **I5** for **$R1**

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 **4***
- *2 RESERVATION STATION **(RS3, RS4)** with 2 FP unit12 **(FPU1, FPU2)** with latency **3***
- *2 RESERVATION STATION **(RS5, RS6)** with 2 INT_ALU/BR units **(ALU1, ALU2)** with latency **2***

Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`FOR:LD $F2, A($R1)`	1	2	6	None	RS1	LDU1
`I1: FADD $F2, $F2, $F2`	2	**7**	10	RAW I0 for $F2	RS3	FPU1
`I2: FADD $F4, $F4, $F0`	3	4	7	None	RS4	FPU2
`I3: SD $F2, A($R1)`	4	**11**	15	RAW with I1 for $F2	RS2	LDU2
`I4: SD $F4, B($R1)`	**7**	8	12	**STR RS1** (RAW with I2 for $F4)	RS1	LDU1
`I5: ADDUI $R1, $R1, 8`	8	9	11	None (WAR solved **)	RS5	ALU1
`I6: BNE $R1, $R2, FOR`	9	**12**	14	RAW with I4 for $R1	RS6	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 = 15 / 7 = 2.14***

EXERCISE 2 – VLIW SCHEDULING (5 points)

Let's consider the following LOOP code, where $**Ri** are integer registers and $**Fi** are floating-point registers.

```
LOOP:  LD $F2, 0 ($R1)
       LD $F4, 0 ($R2)
       FADD $F6, $F2, $F2
       FADD $F8, $F0, $F0
       FADD $F4, $F4, $F6
       FADD $F10, $F0, $F0
       SD $F6, 0 ($R1)
       SD $F4, 0 ($R2)
       SD $F10, 0 ($R3)
       ADDDUI $R1, $R1, 4
       ADDDUI $R2, $R2, 4
       ADDDUI $R3, $R3, 4
       BNE $R1, $R5, LOOP
```

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

- **1 Memory Units with 3 cycles latency**
- **1 FP ALUs with 3 cycles latency**
- **1 Integer ALU with 1 cycle latency to next Int/FP/L/S & 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	Floating Point Unit	Integer Unit
C1	LD $F2, 0 ($R1)	FADD $F8, $F0, $F0	
C2	LD $F4, 0 ($R2)	FADD $F10, $F0, $F0	
C3			
C4		FADD $F6, $F2, $F2	
C5	SD $F10, 0 ($R3)		ADDDUI $R3, $R3, 4
C6			
C7	SD $F6, 0 ($R1)	FADD $F4, $F4, $F6	ADDDUI $R1, $R1, 4
C8			
C9			BNE $R1, $R5, LOOP
C10	SD $F4, 0 ($R2)	(br. delay slot)	ADDDUI $R2, $R2, 4
C11	(busy)		
C12	(busy)		

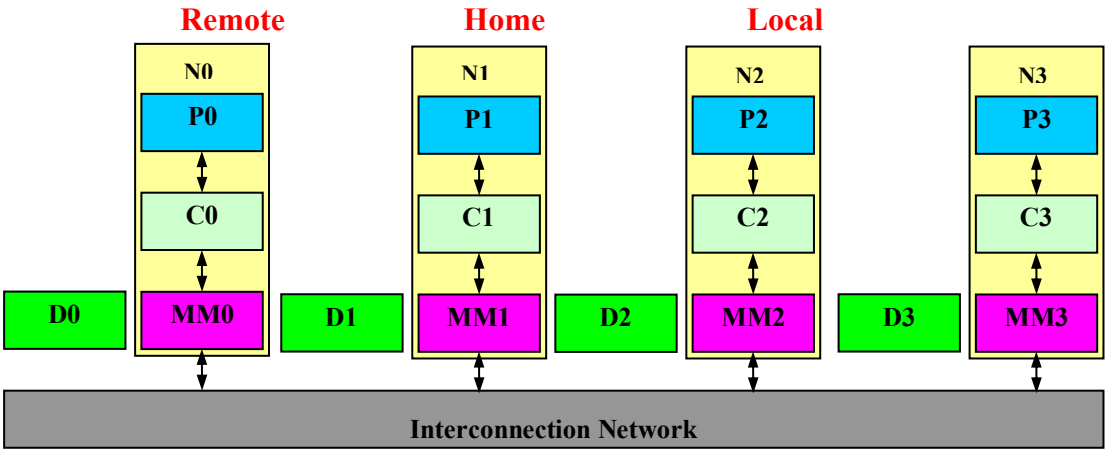
How long is the critical path for a single iteration? _____ **12 cycles**

How much is the code efficiency for a single iteration? _____

Code_eff = IC / (#cycles * #issues) = 13 / (12 x 3) = 13 / 36 = 0.36

EXERCISE 3: DIRECTORY-BASED PROTOCOL (*5 points)*

Consider a directory-based protocol for a distributed shared memory system with **4 nodes (N0, N1, N2, N3) where Directory N1 Block B1 | State: Modified | Sharer Bits 1000 |**



*After a **Write Miss on B1 from Node 2,** please answer to the following questions:*

Which is the **home node**? What is the definition of **home node**?	**Node N1** The home node is the node where the memory block and the related directory entry reside.
Which is the **local node**? What is the definition of **local node**?	**Node N2** The local node is the node sending the Write Miss request.
Which is the **remote node**? What is the definition of **remote node**?	**Node N0** The remote node is where there is a copy of the memory block, in this case modified. Being modified, this is the most updated copy of the block.
What is the message sent from the **home node** in reply to the Write Miss?	**Fetch/Invalidate** message is sent from home node N1 to remote node N0 (past owner) to ask to fetch the most updated copy of the block B1 in the home directory of N1 and to invalidate the block B1 in the past owner's cache C0 in N0. (This is the reply to the**Write Miss** request from N2.

What is the **coherence state** of the block B1 in the remote cache?	The **Cache N0 Block B1** was Modified, but after the Fetch/Invalidate message it becomes: **Cache N0 Block B1 | Invalid**
What is the next message sent from the remote node to the home node?	**Data Write Back** message reply from remote cache N0 (owner) to the home node N1 to write back the most updated copy of the data in the home directory. (This is the reply to the **Fetch/Invalidate** message).
What is the next message sent from the home node to the local node?	**Data Value Reply** message is sent from home node N1 to the local node N2 to write a copy of the block in the cache C2. Then the Local Node is the **new owner** of the block and it will write the new value in block B1.
What is the **coherence state** of the block B1 in the Directory of the home node?	**Directory N1 Block B1 | State: Modified | Sharer Bits 0010 |**
What is the **coherence state** of the block B1in the local cache?	**Cache N2 Block B1 | Modified**
Which is the **new owner** of the block B1?	**Node 2** is the new owner of the block B1.

QUESTION 1: BRANCH PREDICTION (5 points)

Let's consider the STATIC and DYNAMIC branch prediction techniques used in modern microprocessors.
Answer to the following questions:

	STATIC BRANCH PREDICTION	DYNAMIC BRANCH PREDICTION
Explain the main concepts for these two branch prediction techniques..		
What are the main benefits for each branch prediction technique?		

What are the main drawbacks for each branch prediction technique?		
Explain the ways how to schedule an instruction in the branch delay slot		
Explain the main concepts of the correlating branch prediction technique		

QUESTION 2: REORDER BUFFER (5 points)

Let's consider the ReOrder Buffer used in modern microprocessors. *Answer to the following questions:*

Explain the main purpose to introduce the ReOrder buffer in a dynamically scheduled processor.	In traditional tomasulo algorithm, there are three main problems: Speculative Execution, Precise Exception and In order commit. Speculative Execution: ROB can temporarily store the result of speculative instruction instead of write to memory or regiter directly, which reduces branch latency. Precise Exceptiom: Intructions can only update the memory and register in commit stage, if an exception occurs, processor can flush all the instruction after speculative instruction, and restore the precise exception. In order commit: ROB can ensure the commit order same to the program, even through instruction may be executed out of order.
How the Reorder Buffer can support the speculation in the Tomasulo architecture?	ROB can execute the branch instructions based on the speculative result of branch predictor in advance. ROB can temporarily store the result of speculative instruction instead of write to memory and register directly. Only when the instruction is on the head of ROB and the speculative is correct, its result can commit. If the speculative result is false, the procesor will flush all instructions after the misprediction branh in ROB

List the fields of each row entry in a RoB	1.Busy field: indicates the ROB entry is busy or not, if there is both RS and ROB entry, the instruction can be issued 2. Instruction type: records the type of the instruction 3. Ready: The instruction has been completed or not 4. Destination: Target register or memory 5. Value: The result of execution 6. Speculative: indicates the instruction is speculative or not.
Explain what are the four stages of the Speculative Tomasulo pipelined architecture.	1. Issue: check the ROB and RS, if there is a free ROB entry and RS, the instruction can be issued, otherwise instruction stalls. Mark the RS is occupied, and load the instruction oprands information. Allocate a ROB entry, record the instruction infomation. 2. Execution Started: When oprands are ready, start to execute. 3. Write result in ROB: After execution completed, if CDB is available, update the ROB entry and other RS. 4. Commit: update Register file or memory with ROB result. When instruction is on the head of ROB, write back the RF or memory. If the instruction is speculative and the branch predictior mispredict, flush the ROB entries after the instruction.
Explain what happens in a Speculative Tomasulo architecture in the case of a ***branch misprediction***	Flush all the intructions after mispredicted branch. Restore the renaming table, and clear RS

QUIZZES

Question 1 (format Multiple Choice – Single answer)
Let's consider a fully associative write-back cache with many cache entries that at cold start is empty and receives the following sequence of 5 memory accesses:

Write Mem[AAAA]
Write Mem[AAAA]
Read Mem[BBBB]
Write Mem[BBBB]
Write Mem[AAAA]

What are the numbers of cache hits and misses when using a "write allocate" versus a "nowrite allocate" policy?

(SINGLE ANSWER)
1 point
Answer 1: Write allocate has 2 hits & 3 misses | No-write allocate has 1 hit & 4 misses
Answer 2: Write allocate has 3 hits & 2 misses | No-write allocate has 1 hit & 4 misses (**TRUE**)
Answer 3: Write allocate has 1 hit & 4 misses | No-write allocate has 3 hits & 2 misses
Answer 4: Write allocate has 4 hits & 1 miss | No-write allocate has 1 hit & 4 misses
Answer 5: Write allocate has 1 hit & 4 misses | No-write allocate has 2 hits & 3 misses

Motivate your answer:
1 point

Feedback
*For the **write allocate policy,** the first Write to Mem[AAAA] is a miss, but the block is allocated in cache, so the next Write to AAAA is a hit. The next Read to [BBBB] is a miss, but the block is allocated in cache, so the next Write to [BBBB] is a hit. The last Write to AAAA is also a hit. Globally, the write allocate has **3 hits & 2 misses.***

*For the **no-write allocate policy** there is no cache block allocation on writes, therefore the first 2 writes are misses. The next Read Mem[BBBB] is also a miss, but the block corresponding to [BBBB] is allocated in cache. So the next Write Mem[BBBB] is a hit. The last write to Mem[AAAA] is also a miss. Therefore, the no-write allocate has 1 hit & 4 misses.*

Question 2 (format Multiple Choice – Single answer)
Let's consider the following code:

```
for (i=0; i<255; i++)
if (X[i] != 0)
Y[i] = X[i] + Y[i];
```

Which code transformation can be applied to be executed by the VMIPS Vector Processor with a Vector Register File composed of 8 registers of 32 elements and 64 bits/element?

(SINGLE ANSWER)
1 point

Answer 1: Trace scheduling

Answer 2: Software pipeling

Answer 3: Vector strip mining

Answer 4: Vector mask registers (**TRUE**)

Answer 5: Memory striding

Motivate your answers:
1 point

_____

_____

_____

_____

Feedback:
Use a vector mask register to "disable" some elements: The vector mask uses a Boolean vector of length MVL to control the execution of a vector instruction.
When vector mask registers are enabled, any vector instruction operates ONLY on the vector elements whose corresponding masks bits are set to 1.

Question 3 (format Multiple Choice – Multiple answer)
*How does a MESI write-invalidate write-back protocol manage a **Write Hit** on an Exclusive cache block?*

(MULTIPLE ANSWERS)
1 point
Answer 1: The status of the cache block becomes Modified (TRUE)

Answer 2: The cache block is retrieved from memory

Answer 3: An invalidate is broadcasted on the bus to the other copies of the block

Answer 4: The cache block is retrieved from another cache

Answer 5: The cache block is overwritten in the processor's cache (TRUE)

Question 4 (format Multiple Choice – Multiple answer)
*How does a MESI write-invalidate write-back protocol manage a **Write Miss** on a cache block which is Exclusive in another cache?*
(MULTIPLE ANSWERS)
1 point

Answer 1: The status of the cache block becomes Modified (TRUE)

Answer 2: The cache block is retrieved from memory (TRUE)

Answer 3: An invalidate is broadcasted on the bus to the other copies of the block (TRUE)

Answer 4: The cache block is retrieved from another cache

Answer 5: The cache block is overwritten in the processor's cache (TRUE)

Question 5 (format True/False)
To obtain a loop unrolling version of a code, we need to use register renaming if there are some true data dependences in the original code.
(format True/False)
1 point

Answer 1: True / False (False)

Motivate your answer:
1 point

Feedback:
True data dependencies in the code cannot be eliminated by using register renaming because this solves WAW and WAR hazards.
Register renaming can be useful to eliminate anti and output data dependencies.

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

Politecnico di Milano, 20 January, 2025

Course on Advanced Computer Architectures

Prof. C. Silvano

EX1	**(5 points)**	
EX2	**(5 points)**	
EX3	**(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	`LD $F2, A($R3)`	None
I1	`FADD $F2, $F2, $F4`	True data dependence with **I0** for $**F2** Output data dependence with **I0** for **$F2**
I2	`ADD $R1, $R2, $R2`	None
I3	`SD $F2, A($R1)`	True data dependence with **I1** for **$F2** True data dependence with **I2** for **$R1**
I4	`ADD $R2, $R1, $R1`	True data dependence with **I2** for **$R1** Anti dependence with **I2** for **$R2**
I5	`FADD $F4, $F2, $F4`	True data dependence with **I1** for **$F2** Anti dependence with **I1** for **$F4**
I6	`SD $F4, A($R2)`	True data dependence with **I5** for **$F4** True data dependence with **I4** for **$R2**

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 **4***
- *2 RESERVATION STATION **(RS3, RS4)** with 2 FP unit12 **(FPU1, FPU2)** with latency **3***
- *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
`I0: LD $F2, A($R3)`	1	2	6	None	RS1	LDU1
`I1 FADD $F2, $F2, $F4`	2	**7**	10	RAW with **I0** for $**F2** (**)	RS3	FPU1
`I2: ADD $R1, $R2, $R2`	3	4	5	None	RS5	ALU1
`I3: SD $F2, A($R1)`	4	**11**	15	RAW with **I1** for **$F2 (*)**	RS2	LDU2
`I4: ADD $R2, $R1, $R1`	**6**	7	8	**STRUCT $R5**	RS5	ALU1
`I5: FADD $F4, $F2, $F4`	7	**11**	14	RAW with **I1** for **$F2**	RS4	FPU2
`I6: SD $F4, A($R2)`	8	**15**	19	RAW with **I5** for **$F4**	RS1	LDU1

(*) 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 = 19 / 7 = 2,71***

EXERCISE 2 – VLIW SCHEDULING (5 points)

Let's consider the following LOOP code, where $**Ri** are integer registers and $**Fi** are floating-point registers.

```
L1:     LD $F2, 0 ($R1)
        LD $F4, 0 ($R2)
        FADD $F6, $F2, $F2
        FADD $F8, $F0, $F0
        FMUL $F4, $F4, $F6
        FSUB $F10, $F0, $F0
        SD $F6, 0 ($R1)
        SD $F4, 0 ($R2)
        SD $F10, 0 ($R3)
        ADDUI $R1, $R1, 4
        ADDUI $R2, $R2, 4
        ADDUI $R3, $R3, 4
        BNE $R1, $R5, L1
```

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

- **2 Memory Units with 2 cycles latency**
- **1 FP ALUs with 3 cycles latency**
- **1 Integer ALU with 1 cycle latency to next Int/FP/L/S & 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	Memory Unit 2	Floating Point Unit	Integer Unit
C1	LD $F2, 0 ($R1)	LD $F4, 0 ($R2)	FADD $F8, $F0, $F0	
C2			FSUB $F10, $F0,$F0	
C3			FADD $F6, $F2, $F2	
C4				
C5	SD $F10, 0($R3)			ADDUI $R3, $R3,4
C6	SD $F6, 0($R1)		FMUL $F4, $F4, $F6	ADDUI $R1, $R1,4
C7				
C8				BNE $R1, $R5, L1
C9	SD $F4, 0 ($R2)		(br. delay slot)	ADDUI $R2, $R2,4
C10	(busy)			
C11	(busy)			
C12				
C13				
C14				

How long is the critical path for a single iteration? _____ **11 cycles**

How much is the code efficiency for a single iteration? _____

Code_eff = IC / (#cycles * #issues) = 13 / (11 x 4) = 13 / 44 = 0.3

EXERCISE 3: DYNAMIC BRANCH PREDICTION (5 points)

Assume a pipelined processor with a dynamic branch prediction unit composed of a **1-entry 1-bit Branch History Table** in the **IF-stage.**

- Disabling the branch prediction unit, each branch costs **3 cycles penalty** to fetch the correct instruction.
- Enabling the branch prediction unit, there are 4 cases for each conditional branch with the related **branch penalty cycles:**

Branch Outcome Prediction	Branch Outcome	Branch Penalty Cycles
Predicted Not Taken	Not Taken	0
Predicted Not Taken	Taken	3 cycles (misprediction)
Predicted Taken	Not Taken	3 cycles (misprediction)
Predicted Taken	Taken	1 cycle

Let's consider the following assembly loop:

```
INIT:  ADDUI $R1, $R0, 0
       ADDUI $R2, $R0, 80
LOOP:  LD $F0, 0 ($R1)
       FADD $F4, $F0, $F2
       SD $F4, 0 ($R1)
       ADDUI $R1, $R1, 4
       BNE $R1, $R2, LOOP
```

1. *How many loop iterations?* ***20 iterations*** _____

2. *Please complete the following table:*

	Explain the branch behavior in the loop.	***How many branch penalty cycles are needed to execute the loop?***	***Calculate the branch misprediction rate to execute the loop***
Assume the BHT predictor is enabled and initialized as **Not Taken.**	In this case, we have a first misprediction with PNT/T with 3 cycles penalty. Then there are 18 iterations predicted correctly as PT/T with 1 cycle penalty. The last iteration is mispredicted as PT/NT with 3 cycles penalty.	There are: (3 + 18 + 3) = 24 branch penalty cycles.	There are 2 mispredictions out of 20 predictions => **10%** misprediction rate;
Assume the BHT predictor is enabled and initialized as **Taken.**	In this case, we have 19 iterations correctly predicted as PT/T with 1 cycle penalty We have 1 misprediction at the last iteration as PT/NT with 3 cycles penalty.	There are: (19 + 3) = 22 branch penalty cycles.	There is 1 misprediction out of 20 predictions => **5%** misprediction rate.
Assume the BHT predictor is disabled.	At each iteration, each branch costs 3 cycle penalty to fetch the correct instruction.	There are (20 x3) = 60 branch penalty cycles to execute the loop composed of 20 iterations.	

QUESTION 1: CACHE COHERENCE PROTOCOLS (5 points)

Let's consider the SNOOPING and the DIRECTORY-BASED protocols used to maintain the cache coherence in modern multiprocessors. *Answer to the following questions:*

	SNOOPING PROTOCOL	**DIRECTORY-BASED PROTOCOL**
Explain the main characteristics of each protocol.		
For which type of multiprocessor architecture is each protocol suitable for?		

What are the main benefits of each protocol?		
*Explain the **two types** of Snooping protocol depending on what happens on a write operation.*		
*Explain the **three** possible coherence states of a block in the home directory and the meaning of the sharer bits.*		

QUESTION 2: DYNAMIC BRANCH PREDICTION (5 points)

Let's consider the Dynamic Branch Prediction techniques used in modern microprocessors. *Answer to the following questions:*

Explain the main benefits to introduce the dynamic branch prediction in a modern microprocessor.	
*Explain how works a **Branch History Table***	

Explain the main purpose to introduce the ***Branch Target Buffer.***	
Explain the main concepts of the ***correlating branch prediction technique***	
Explain what happens at runtime in a Speculative Tomasulo Architecture in the case of a ***branch misprediction***	

QUIZZES

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

How many convoys? How many clock cycles to execute the code?

(SINGLE ANSWER)
1 point

Answer 1: 3 convoys; 24 clock cycles
Answer 2: 2 convoys; 16 clock cycles
Answer 3: 4 convoys; 32 clock cycles **(TRUE)**
Answer 4: 5 convoys; 40 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, RX;	ADDVV.D V1, V1, F0	MULVS.D V2, V1, V1
2^ convoy	L.V V3, RY	ADDVV.D V3, V2, V3	MULVS.D V2, V2, F0
3^ convoy	S.V V2, RZ		
4^ convoy	S.V V3, RW		
5^ convoy			

Another possible solution is:

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

Question 2 (format True/False)
To obtain a loop unrolling version of a code, we can use register renaming to eliminate name dependences in the original code.
(format True/False)
1 point

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

Question 3 (format Multiple Choice – Single answer)
Let's consider a fully associative write-back cache with many cache entries that at cold start is empty and receives the following sequence of 5 memory accesses:

```
Write Mem[AAAA]
Read  Mem[AAAA]
Read  Mem[BBBB]
Write Mem[CCCC]
Write Mem[BBBB]
```

How much is the Miss Rate when using a "Write Allocate" versus a "No-write Allocate" policy?

(SINGLE ANSWER)
1 point
Answer 1: Miss Rate 40% with Write Allocate | Miss Rate 60% with No-write Allocate

Answer 2: Miss Rate 60% with Write Allocate | Miss Rate 80% with No-write Allocate **(TRUE)**

Answer 3: Miss Rate 60% with Write Allocate | Miss Rate 40% with No-write Allocate

Answer 4: Miss Rate 20% with Write Allocate | Miss Rate 80% with No-write Allocate

Answer 5: Miss Rate 80% with Write Allocate | Miss Rate 20% with No-write Allocate

Motivate your answer by completing the following table:
1 point

	Write Allocate	No-write Allocate
`Write Mem[AAAA]`	*Write Miss*	*Write Miss*
`Read  Mem[AAAA]`	*Read Hit*	*Read Miss*
`Read  Mem[BBBB]`	*Read Miss*	*Read Miss*
`Write Mem[CCCC]`	*Write Miss*	*Write Miss*
`Write Mem[BBBB]`	*Write Hit*	*Write Hit*

Feedback
*For the **write allocate policy,** the first Write to Mem[AAAA] is a miss, but the block is allocated in cache, so the next Read to Mem[AAAA] is a hit. The next Read to [BBBB] is a miss, but the block is allocated in cache, so the next Write to [BBBB] is a hit. The Write to Mem[CCCC] is a miss. Globally, the write allocate policy shows **2 hits & 3 misses.** Therefore, the miss rate is 3 / 5 = 60%.*

*For the **no-write allocate policy** there is no cache block allocation on writes, therefore the first write to Mem[AAAA] is a miss, but also the next Read Mem[AAAA] is a miss. The Read [BBBB] is a miss and the block is allocated in cache. So the next Write to Mem[BBBB] is a hit. The Write to Mem[CCCC] is also a miss. Therefore, the no-write allocate has 1 hit & 4 misses. Therefore, the miss rate is 4 / 5 = 80%.*

Question 4 (format Multiple Choice – Multiple answer)

Consider a dual-node shared-memory multiprocessor. How does a MESI write-invalidate write-back protocol manage a ***Read Miss*** *in node N0 on a cache block BO which is* ***Exclusive*** *in the other cache of node N1?*
(MULTIPLE ANSWERS)
1 point

Answer 1: The status of the cache block B0 becomes Shared in both caches; **(TRUE)**

Answer 2: The cache block is copied in node N0 from the corresponding memory block;

Answer 3: An invalidate is broadcast on the bus to the Exclusive cache block B0 in node N1;

Answer 4: The cache block B0 is copied in N0 from the other cache block in node N1 **(TRUE)**

Answer 5: The status of the cache block B0 becomes Exclusive in the cache of node N0;

Question 5 (format Multiple Choice – Single answer)
Assume to apply a processor optimization resulting ten time faster on computation than the original mode of execution. What is the fraction of computation needed to double the overall speedup?
(SINGLE ANSWER)
1 point

Answer 1: 50%

Answer 2: 55% **(TRUE)**

Answer 3: 25%

Answer 4: 75%

Answer 5: 20%

Motivate your answer:
1 point

_____

_____

_____

_____

Feedback:
To get an overall speedup of 2, we need to apply the Amdahl's law as follows:
$1 / [(1-F_V) + (F_V / 10)] = 2$
$=> 10 / (10 – 9 F_V) = 2$
$=> 18 F_V = 10$
$=> F_V = 10/ 18 = 0.55$
$=> F_V = 55\%$

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:

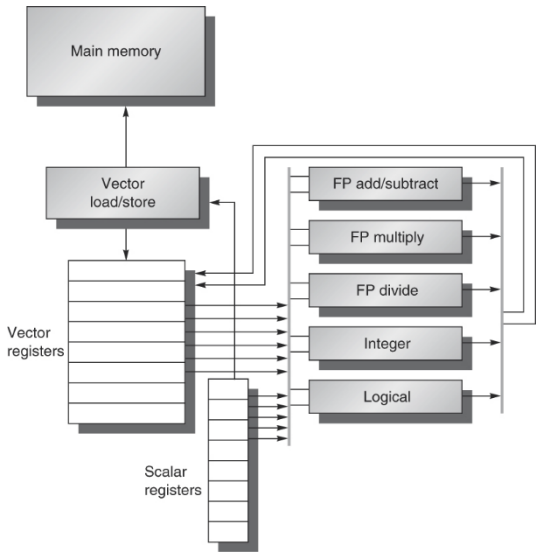
	Instruction Level Parallelism (ILP)	**Thread Level Parallelism (TLP)**
Explain the main concepts for each approach.		
Which type of technique can be applied in superscalar processors to combine both ILP and TLP?		

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

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

Politecnico di Milano, 10 June, 2025

Course on Advanced Computer Architectures

Prof. C. Silvano

EX1	**(5 points)**	
EX2	**(5 points)**	
EX3	**(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	`FOR:LD $F2, A($R1)`	None
I1	`LD $F4, B($R1)`	None
I2	`LD $F6, C($R1)`	None
I3	`FADD $F4, $F2, $F4`	True data dependence with **I0** for $**F2** True data dependence with **I1** for **$F4** Output data dependence with **I1** for **$F4**
I4	`FADD $F6, $F4, $F6`	True data dependence with **I2** for **$F6** True data dependence with **I3** for **$F4** Output data dependence with **I2** for **$F6**
I5	`SD $F6, A($R1)`	True data dependence with **I4** for **$F6**
I6	`ADDUI $R1, $R1, 8`	Anti dependence with **I0** for **$R1** Anti dependence with **I1** for **$R1** Anti dependence with **I2** for **$R1** Anti dependence with **I5** for **$R1**
I7	`BNE $R1, $R2, FOR`	True data dependence with **I6** for **$R1**

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 **4***
- *2 RESERVATION STATION **(RS3, RS4)** with 2 FP units **(FPU1, FPU2)** with latency **3***
- *2 RESERVATION STATION **(RS5, RS6)** with 2 INT_ALU/BR units **(ALU1, ALU2)** with latency **2***

Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`FOR:LD $F2, A($R1)`	1	2	6	None	RS1	LDU1
`I1: LD $F4, B($R1)`	2	3	7	None	RS2	LDU2
`I2: LD $F6, C($R1)`	**7**	8	12	**STR. RS1**	RS1	LDU1
`I3: FADD $F4, $F2, $F4`	8	9	**13**	**STR. WR on CDB** (RAW I0 for $F2 RAW I1 for $F4)	RS3	FPU1
`I4: FADD $F6, $F4, $F6`	9	**14**	17	**RAW I3 for $F4** (RAW I2 for $F6)	RS4	FPU2
`I5: SD $F6, A($R1)`	10	**18**	22	**RAW with I4 for $F6**	RS2	LDU2
`I6: ADDUI $R1, $R1, 8`	11	12	14	None (WAR solved by Reg. Ren.**)	RS5	ALU1
`I7: BNE $R1, $R2, FOR`	12	**15**	17	**RAW with I6 for $R1**	RS6	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 (5 points)

Given the following software pipelined loop:

```
SP_LOOP: SD F6, 0 (R1)
I1:      SD F8, 0 (R2)
I2:      SD F10, 0 (R3)
I3:      FADD F6, F0, F0
I4:      FADD $F8, F2, F2
I5:      FADD $F10, F4, F4
I6:      LD F0, 16 (R1)
I7:      LD F2, 16 (R2)
I8:      LD F4, 16 (R3)
I9:      ADDUI R1, R1, 8
I10:     ADDUI R2, R2, 8
I11:     ADDUI R3, R3, 8
I12:     BNE R1, R4, SP_LOOP
```

Consider a **4-issue VLIW** machine with **fully pipelined functional units:**

- **2 Memory Units with 3 cycles latency**
- **1 FP ALUs with 3 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 `SP_LOOP`, *complete the following table by using the* ***list-based scheduling*** (do NOT introduce any loop transformation) on the ***4-issue VLIW*** *machine including the BRANCH DELAY SLOT.* Please do not write in NOPs.

	Memory Unit 1	Memory Unit 2	Floating Point Unit	Integer Unit
C1	`SD F6, 0(R1)`			
C2				
C3				
C4				
C5				
C6				
C7				
C8				
C9				
C10				
C11				
C12				
C13				
C14				
C15				

Answer to the following questions:

How long is the critical path for a single iteration?	
What performance did you achieve in CPI?	
What performance did you achieve in FP ops per cycles?	
How much is the code efficiency?	

Feedback

From the dependency analysis of the given software pipelined loop, there are several anti-dependencies (WAR hazards) and only one RAW:

```
SP_LOOP: SD F6, 0 (R1)
I1:      SD F8, 0 (R2)
I2:      SD F10, 0 (R3)
I3:      FADD F6, F0, F0   # WAR with I0 for F6
I4:      FADD $F8, F2, F2  # WAR with I1 for F8
I5:      FADD $F10, F4, F4 # WAR with I2 for F10
I6:      LD F0, 16 (R1)          # WAR with I3 for F0
I7:      LD F2, 16 (R2)          # WAR with I4 for F2
I8:      LD F4, 16 (R3)          # WAR with I5 for F4
I9:      ADDUI R1, R1, 8         # WAR with I0, I6 for R1
I10:     ADDUI R2, R2, 8         # WAR with I1, I7 for R2
I11:     ADDUI R3, R3, 8         # WAR with I2, I8 for R3
I12:     BNE R1, R4, SP_LOOP     # RAW with I9 for R1
```

There are two possible scheduling solutions, both with ***5 cycles critical path:***

	Memory Unit 1	Memory Unit 2	Floating Point Unit	Integer Unit
C1	`SD F6, 0(R1)`	`LD F0, 16(R1)`	`FADD F6, F0, F0`	`ADDUI R1, R1, 8`
C2	`SD F8, 0(R2)`	`LD F2, 16(R2)`	`FADD F8, F2, F2`	`ADDUI R2, R2, 8`
C3	`SD F10, 0(R3)`	`LD F4, 16(R3)`	`FADD F10, F4, F4`	`BNE R1, R4, SP_LOOP`
C4	`(busy)`	`(busy)`	`(busy)`	`ADDUI R3, R3, 8` `(branch delay slot)`
C5	`(busy)`	`(busy)`	`(busy)`	
C6				
C7				
C8				

	Memory Unit 1	Memory Unit 2	Floating Point Unit	Integer Unit
C1	`SD F6, 0(R1)`	`SD F8, 0(R2)`	`FADD F6, F0, F0`	
C2	`SD F10, 0(R3)`	`LD F0, 16(R1)`	`FADD F8, F2, F2`	`ADDUI R1, R1, 8`
C3	`LD F2, 16(R2)`	`LD F4, 16(R3)`	`FADD F10, F4, F4`	`ADDUI R2, R2, 8`
C4	`(busy)`	`(busy)`	`(busy)`	`BNE R1, R4, SP_LOOP`
C5	`(busy)`	`(busy)`	`(busy)`	`ADDUI R3, R3, 8` `(branch delay slot)`
C6				
C7				
C8				

How long is the critical path for a single iteration?	***5 cycles***
What performance did you achieve in CPI?	***CPI = # cycles / IC = 5 / 13 = 0.38***
What performance did you achieve in FP ops per cycles?	***FP ops per cycles = # FP ops / # cycles = 3 / 5 = 0.6***
How much is the code efficiency?	***Code efficiency = IC / (# cycles x # issues) = 13 / (5 x 4) = 13 / 20 = 0.65***

EXERCISE 3 – PERFORMANCE EVALUATION (5 points)

1. *Given **CPU$_1$** with clock frequency **500 MHz** and **CPU$_2$** with clock frequency **200 MHz,** let's consider the two CPUs execute a kernel of **100** instructions with the following frequencies of occurrence and clock cycles:*

Instruction Type	Instruction Frequency	CPU$_1$ Clock Cycles	CPU$_2$ Clock Cycles
ALU	30%	2	1
LOAD	10%	8	6
STORE	20%	2	3
JUMP	30%	2	2
BRANCH	10%	3	3

	CPU$_1$ **f_1 = 500 MHz; T_{clk} = 2 ns**	**CPU$_2$** **f_2 = 200 MHz; T_{clk} = 5 ns**
How much is the average CPI for each CPU?	**CPI1 =** **CPI$_1$** = 0.3*2 + 0.1*8 + 0.2*2 + 0.3*2 + 0.1*3 = 0.6 + 0.8 + 0.4 + 0.6 + 0.3= 2.7	**CPI2 =** **CPI$_2$** = 0.3*1 + 0.1*6 + 0.2*3 + 0.3*2 + 0.1*3 = 0.3 + 0.6 + 0.6 + 0.6 + 0.3 = 2.4
How much is the CPU time for each CPU?	**CPUtime1 =** **CPUtime$_1$** = IC * CPI$_1$ * T_{clk1} = = 100 * 2.7 *2 ns = 540 ns	**CPUtime2 =** **CPUtime$_2$** = IC * CPI$_2$* T_{clk2} = = 100 * 2.4 *5 ns = = 1200 ns = 1.2 µs
How much CPU$_1$ is faster than CPU$_2$?	*It is not possible to calculate the speedup by comparing the CPIs because the clock frequencies are different => we need to compare either the MIPS or the CPU times:* **MIPS$_1$** = f_{clk1} / CPI$_1$ *10^6 = 500 / 2.7 **MIPS$_2$** = f_{clk2} / CPI$_2$ *10^6 = 200 / 2.4 **SpeedUp =** MIPS_accelerated / MIPS_unaccelerated MIPS$_1$/MIPS$_2$ = 1200 / 540 = 2.22 ⇨ MIPS$_1$ = 2.22 * MIPS$_2$ *CPU$_1$ is 2.22 times faster than CPU$_2$* By considering the CPU times: **Slowdown =** CPUtime_unaccelerated / CPUtime_accelerated CPUtime$_2$/CPUtime$_1$ = = 1200 ns / 540 ns = 2.22 CPUtime$_2$ is 2.22 times longer than CPUtime$_1$	

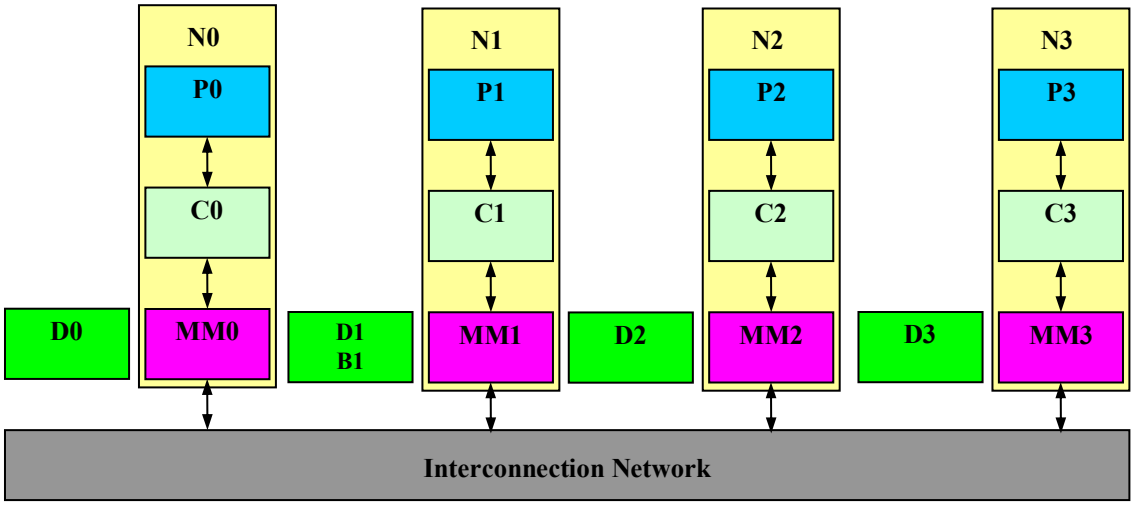
2. *Evaluate the impact of the memory hierarchy on another CPU_3 for which all instructions require **8 clock cycles** in the following cases:*

How much is the CPI_3 when considering an ideal cache (100% hit)?	**CPI_3_ideal_cache =** CPI_3_ideal_cache = CPI_3_exe = 8
How much is the CPI_3 when considering a real cache with a miss rate of 50% and assuming 3 memory accesses on average for each instruction and miss penalty of 3 clock cycles? *How much is the impact on performance of the real cache with respect to the ideal cache (100% hit)?*	CPI_3_exe = 8; Miss Rate = 0.5; Memory Accesses Per Instruction (MAPI) = 3, Miss Penalty = 3; **CPI_3_real_cache =** CPI_3_real_cache = = CPI_3_exe + (MAPI * Miss Rate * Miss_Penalty) = = 8 + (3*0.5*3) = 12.5 Impact on performance (slowdown): CPI_3_real_cache / CPI_3_ideal_cache = 12.5 / 8 = 1.56
How much is the CPI_3 in the case of no cache (100% miss)? *How much is the impact on performance with respect to the real cache (50% hit)?*	**CPI_3_no_cache =** CPI_3_no_cache = = CPI_3_exe + (MAPI * 1 *Miss_Penalty) = = 8 + (3*3) = 17 Impact on performance (slowdown): CPI_3_no_cache / CPI_3_real_cache = 17 / 12.5 = 1.36

QUESTION 1: DIRECTORY-BASED PROTOCOL (*5 points)*

Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where we consider the block **B1** in the directory of **N1:**

Directory N1 Block B1 | State: Shared | Sharer Bits: 0 1 0 0 |



*Given the following **sequence of accesses**, please answer to the following questions:*

Read Miss on B1 from node N2				
Home node?	*Local node?*	*Remote node(s)?*	*What is the sequence of messages sent among the nodes?*	*Which are the final coherence state and sharer bits of the block B1 in the home directory?*
N1				**B1:** | State: _____ | | Sharer Bits: _ _ _ _ |

Write Hit on B1 from node N2				
Home node?	*Local node?*	*Remote node(s)?*	*What is the sequence of messages sent among the nodes?*	*Which are the final coherence state and sharer bits of the block B1 in the home directory?*
				B1: | State: _____ | | Sharer Bits: _ _ _ _ |
Write Miss on B1 from node N0				
Home node?	*Local node?*	*Remote node(s)?*	*What is the sequence of messages sent among the nodes?*	*Which are the final coherence state and sharer bits of the block B1 in the home directory?*
				B1: | State: _____ | | Sharer Bits: _ _ _ _ |

QUESTION 2: MULTICORE PROCESSORS (5 points)

High-performance multicore processors today use a combination of several computer architecture techniques. ***Let's consider a single-chip dual-core shared-memory architecture where each core is a dual-issue superscalar processor with multi-threading up to 2 threads per core.***

Explain the preferred technique used to manage the multiple threads for each core.

*Make **an example** of scheduling up to 4 threads per chip where T1 and T2 are executed on Core1 and T3 and T4 on Core2.*

Cycle	Core 1		Core 2	
	Issue 1	Issue 2	Issue 1	Issue 2
C0				
C1				
C2				
C3				
C4				
C5				
C6				
C7				
C8				
C9				

Let's consider our dual-issue SMT processor that can manage up to 4 concurrent threads (2-thread per core)	*How much is the **ideal per core CPI?*** *1 / 2 = 0.5* *How much is the **ideal dual-core CPI?*** *1 / 4 = 0.25* *How much is the **ideal per-thread CPI?*** *2 / 2 = 1*
Explain the preferred technique used in each superscalar processor to manage the instruction scheduling in each thread.	
Explain the preferred technique used to manage the cache coherency problem in the dual core chip.	

QUIZZES

Question 1

Let's consider the directory-based protocol for multiprocessors.
Which is the message type that can be sent form the remote node to the home node of a block?
(SINGLE ANSWER)
1 point

Answer 1: Data value reply

Answer 2: Data write back **(TRUE)**

Answer 3: Invalidate

Answer 4: Fetch/Invalidate

Question 2

Let's consider a quad-issue SMT processor that can manage up to 8 simultaneous threads.
What are the values of the ideal CPI and the ideal per-thread CPI?
(SINGLE ANSWER)
1 point

Answer 1: Ideal CPI = 1 & Ideal per-thread CPI = 0.125

Answer 2: Ideal CPI = 0.5 & Ideal per-thread CPI =2

Answer 3: Ideal CPI = 0.25 & Ideal per-thread CPI = 2 **(TRUE)**

Answer 4: Ideal CPI = 0.5 & Ideal per-thread CPI = 4

Answer 5: Ideal CPI = 0.25 & Ideal per-thread CPI = 4

Question 3

In the GP-GPU programming model, the CPU (host) and GPU (device) have separate memory address spaces.
(True/False)
1 point

Answer: True False **True**

Question 4

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 V2, RB          # Load vector from memory address RA into V1
L.V V3, RC          # Load vector from memory address RB into V3
MULVS.D V1, V1, F1  # FP multiply vector V1 to scalar F1
MULVS.D V2, V2, F2  # FP multiply vector V2 to scalar F2
MULVS.D V3, V3, F3  # FP multiply vector V3 to scalar F3
ADDVV.D V2, V2, V1  # FP add vectors V2 and V1
ADDVV.D V3, V2, V3  # FP add vectors V2 and V3
S.V V3, RD          # Store vector V3 into memory address RD
```

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 **(TRUE)**

Answer 4: 5 convoys; 40 clock cycles

Answer 5: 6 convoys; 48 clock cycles

Motivate your answer by completing the following table:
1 point

	Load/Store Vector Unit	Mul/Div Vector Unit	Add/Sub Vector Unit
1^ convoy	L.V V1, RA	MULVS.D V1, V1, F1	
2^ convoy	L.V V2, RB	MULVS.D V2, V2, F2	ADDVV.D V2, V2, V1
3^ convoy	L.V V3, RC	MULVS.D V3, V3, F3	ADDVV.D V3, V2, V3
4^ convoy	S.V V3, RZ		
5^ convoy			
6^ convoy			

Question 5

Let's consider the following LOOP to be executed on a processor with a single-entry 2-bit Branch History Table initialized in the state Strongly Not-Taken:

```
INIT: ADDI.U $R1, $R0, 0
      ADDI.U $R2, $R0, 20
LOOP: LD $F0, 0 ($R1)
      ADDI.D $F2, $F0, $F0
      SD $F2, 0 ($R1)
      ADDI.U $R1, $R1, 4
      BNE $R1, $R2, LOOP
```

After the last iteration, how much is the misprediction rate on the BNE instruction?
(SINGLE ANSWER)
1 point

Answer 1: 60% **(TRUE)**
Answer 2: 90%
Answer 3: 99 %
Answer 4: 100%
Answer 5: 80 %

After the last iteration, what is the state of the prediction left in the Branch History Table?
(SINGLE ANSWER)
1 point

Answer 1: Strongly Taken
Answer 2: Weakly Taken **(TRUE)**
Answer 3: Strongly Not Taken
Answer 4: Weakly Not Taken

Motivate your answer by completing the following table:
1 point

Iteration	*1st iteration*	*2^ iteration*	*3^ iteration*	*4^ iteration*	*5^ iteration*
Prediction State	Strongly Not Taken	Weakly Not Taken	Strongly Taken	Strongly Taken	Strongly Taken
Prediction	Not Taken	Not Taken	Taken	Taken	Taken
Branch Outcome	Taken	Taken	Taken	Taken	Not Taken

Surname	
Name	
POLIMI Personal code	
Signature	

SOLUTION

Politecnico di Milano, 8 May 2023

Course on Advanced Computer Architectures

Prof. C. Silvano

EX 1	**(4 points)**	
EX 2	**(4 points)**	
EX 3	**(3 points)**	
EX 4	**(2 points)**	
QUIZ 5	**(1 point)**	
QUIZ 6	**(1 point)**	
TOTAL	**(15 points)**	
EX 7	**+ 3 extra points**	

EXERCISE 1: VLIW (4 points)

Let's consider the following FOR loop:

```
FOR:LD $F2, A($R1)
    LD $F4, B($R1)
    LD $F6, C($R1)
    FADD $F2,$F2,$F2
    SD $F2,B($R1)
    FADD $F4,$F4,$F4
    SD $F4,C($R1)
    FADD $F6,$F6,$F6
    SD $F6,D($R1)
    ADDUI $R1,$R1,4
    BNE $R1,$R2, FOR
```

Given a **4-issue VLIW** machine with **fully pipelined functional units:**

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

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	Memory Unit 2	Integer Unit	Floating Point Unit
C1	`LD F2, A(R1)`	`LD F4, B(R1)`		
C2	`LD $F6, C($R1)`			
C3				`FADD $F2,$F2,$F2`
C4				`FADD $F4,$F4,$F4`
C5				`FADD $F6,$F6,$F6`
C6	`SD $F2,B($R1)`			
C7	`SD $F4,C($R1)`			
C8	`SD $F6,D($R1)`		`ADDUI $R1,$R1,4`	
C9				
C10			`BNE $R1,$R2, FOR`	
C11			`Br. delay slot`	
C12				
C13				
C14				
C15				

1. How long is the critical path? _____

   11 cycles

2. What performance did you achieve in CPIas?

   _____

   1

3. What performance did you achieve in FP ops per cycles?

   _____

   3 / 11

4. How much is the code efficiency?

   _____

   Code_eff = IC/ (# cycles * # issues) = 11 / (11 * 4) = 0.25

Feedback

	Memory Unit 1	Memory Unit 2	Integer Unit	Floating Point Unit
C1	`LD F2,A(R1)`	`LD F4,B(R1)`		
C2	`LD F6,B(R1)`			
C3				`FADD F2, F2, F2`
C4				`FADD F4, F4, F4`
C5				`FADD F6, F6, F6`
C6	`SD $F2,B(R1)`			
C7	`SD $F4,C($R1)`			
C8	`SD $F6,D($R1)`		`ADD R1,R1,4`	
C9				
C10			`BNE $R1,$R2, FOR`	
C11			`Br. delay slot`	
C12				
C13				
C14				
C15				

1. How long is the critical path?

   11 cycles

2. What performance did you achieve in CPIas?

   CPIas = (# cycles) / IC = 11 / 11 = 1

3. What performance did you achieve in FP ops per cycles?

   (# FP ops) / cycles = 3 / 11 = 0.27

4. How much is the code efficiency?

   Code_eff = IC/ (# cycles * # issues) = 11 / (11 * 4) = 0.25

EXERCISE 2 – TOMASULO (4 points)

Let's consider the following loop 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 RESERVATION STATIONS* ***(RS1, RS2)*** *for 2 LOAD/STORE UNITS* ***(LDU1, LDU2)*** *with latency* ***4***
- *2 RESERVATION STATION* ***(RS3, RS4)*** *for 2 ALU/BR FUs* ***(ALU1, ALU2)*** *with latency* ***2***
- Static Branch Prediction **BTFNT (BACKWARD TAKEN FORWARD NOT TAKEN)** with Branch Target Buffer

Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`FOR1:beq $t6,$t7, END`	**1**	**2**	**4**	None	RS3	ALU1
`lw $t2,VECTA($t6)`						
`lw $t3,VECTB($t6)`						
`addi $t2,$t2,k`						
`sw $t2,VECTA($t6)`						
`add $t4,$t2,$t3`						
`sw $t4,VECTC($t6)`						
`addi $t6,$t6,4`						
`j FOR1`						

Calculate the ***CPI*** *and the* ***IPC:***

CPI = _____

IPC= _____

Feedback:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`FOR1:beq $t6,$t7, END`	**1**	**2**	**4**	None	RS3	ALU1
`lw $t2,VECTA($t6)`	2	3	7	(Control solved by BP-NT)	RS1	LDU1
`lw $t3,VECTB($t6)`	3	4	8	None	RS2	LDU2
`addi $t2,$t2,k`	4	8	10*	**RAW $t2**	RS4	ALU2
`sw $t2,VECTA($t6)`	8	11	15	**RAW $t2 + STRUCT RS1**	RS1	LDU1
`add $t4,$t2,$t3`	9	11	13	**RAW $t2** (RAW $t3 ok)	RS3	ALU1
`sw $t4,VECTC($t6)`	10	14	18	**RAW $t4**	RS2	LDU2
`addi $t6,$t6,4`	11	12	14	(WAR $t6 OK)	RS4	ALU2
`j FOR1`	14	15	17	**STRUCT RS3**	RS3	ALU1

(*) The result of ALU2 is forwarded through the CDB to RS1, RS3 and the RF.

***CPI** = # clock cycles / IC = 18 / 9 = 2*

***IPC** = 1/CPI = 1/ 2 = 0.5*

EXERCISE 3 – CACHE MEMORIES (3 points)

Let's consider 32-block main memory with a **2-way set associative** 8-block cache based on a ***write allocate*** with ***write-back*** protocol.

The addresses are expressed as:
Memory Address: $[0, 1, 2, \ldots\ldots\ldots 31]_{10}$
Cache Address: [a, b, c, d, e, f, g, h]
and Cache Tags are expressed as binary numbers.

At cold start the cache is empty, then there is the following sequence of memory accesses.
Please complete the following table:

	Type of memory access	Memory Address	HIT/MISS Type	Cache Tag	Cache Address	Set	Write in memory
1	Read	$[24]_{10}$	Cold-start Miss	$[110]_2$	a	$[0]_{10}$	No
2	Write	$[24]_{10}$	Hit	$[110]_2$	a	$[0]_{10}$	No
3	Read	$[14]_{10}$					
4	Read	$[04]_{10}$					
5	Write	$[14]_{10}$					
6	Write	$[24]_{10}$					
7	Read	$[12]_{10}$					
8	Read	$[21]_{10}$					
9	Write	$[18]_{10}$					
10	Read	$[02]_{10}$					

Feedback

The 2-way set-associative cache has 8-blocks [a, b, c, d, e, f, g, h] organized in 4 sets $[0, 1, 2, 3]_{10}$ where each set contains 2 blocks as follows:

Set_0: [a ,b]; Set_1: [c, d]; Set_2: [e, f]; Set_3: [g, h]

Being a 2-way set associative cache, the block replacement policy in each set uses the LRU algorithm.

Memory Address Mapping

S0	S1	S2	S3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

	Type of memory access	Memory Address	HIT/MISS Type	Cache Tag	Cache Address	Set	Write in memory
1	Read	$[24]_{10}$	Cold-start Miss	$[110]_2$	a	$[0]_{10}$	No
2	Write	$[24]_{10}$	Hit	$[110]_2$	a	$[0]_{10}$	No
3	Read	$[14]_{10}$	Cold-start Miss	$[011]_2$	e	$[2]_{10}$	No
4	Read	$[04]_{10}$	Cold-start Miss	$[001]_2$	b	$[0]_{10}$	No
5	Write	$[14]_{10}$	Hit	$[011]_2$	e	$[2]_{10}$	No
6	Write	$[24]_{10}$	Hit	$[110]_2$	a	$[0]_{10}$	No
7	Read	$[12]_{10}$	Conflict Miss	$[011]_2$	b	$[0]_{10}$	No
8	Read	$[21]_{10}$	Cold-start Miss	$[101]_2$	c	$[1]_{10}$	No
9	Write	$[18]_{10}$	Cold-start Miss	$[100]_2$	f	$[2]_{10}$	No
10	Read	$[02]_{10}$	Conflict Miss	$[000]_2$	e	$[2]_{10}$	Yes Wr. in $M[14]_{10}$

EXERCISE 4 – SPECULATIVE TOMASULO WITH ROB (2 points)

Let's consider the following LOOP to be executed by a **Speculative Tomasulo** architecture with:

- 2 Load buffers (Load1, Load2);
- 2 FP ALU reservation stations (FP1 and FP2);
- 2 Integer reservation stations (INT1, INT2);
- 8-entry ROB (ROB0, ROB1, ….. , ROB7).

```
LOOP: LD $F0, 0 ($R1)
      FADD $F4, $F0, $F2
      FADD $F4, $F4, $F2
      SD $F4, AA ($R1)
      SUBI $R1, $R1, 8
      BNEZ $R1, LOOP   // branch prediction taken
EXIT:
```

Let's consider the ROB table by assuming that load of the first iteration is executing a cache miss. Let's continue to issue the next instructions until there are available resources or the ROB becomes full.
Please complete the ROB and the Rename Table

ROB

ROB#	Instruction	Dest.	Ready	Spec.	
ROB0	LD $F0, 0 ($R1) (1^ iteration exec. cache miss)	$F0	No	No	**HEAD**
ROB1					
ROB2					
ROB3					
ROB4					
ROB5					
ROB6					
ROB7					

Rename Table:

$F0	
$F2	
$F4	

Feedback

ROB

ROB#	Instruction	Dest.	Ready	Spec.	
ROB0	LD $F0, 0 ($R1) (1^ iteration exec. cache miss)	$F0	No	No	**HEAD**
ROB1	FADD $F4, $F0, $F2 (1^ iteration issued)	$F4	No	No	
ROB2	FADD $F4, $F4, $F2 (1^ iteration issued)	$F4	No	No	
ROB3	SD $F4, AA ($R1). (1^ iteration issued)	MEM	No	No	
ROB4	SUBI $R1, $R1, 8 (1^ iteration issued)	$R1	No	No	
ROB5	BNEZ $R1, LOOP (1^ branch predicted as taken)	--	No	No	
ROB6	LD $F0, 0 ($R1) (2^ iteration issued speculatively)	$F0	No	Yes	
ROB7					**TAIL**

Rename Table:

$F0	~~ROB0~~ ROB6
$F2	
$F4	~~ROB1~~ ROB2

We cannot issue instructions until the ROB becomes full because the 2 FP ALU reservation stations are busy. Therefore, the ROB7 is still empty because we cannot issue the FADD (speculatively). Notice that NO Store Buffers are needed in the Speculative Tomasulo architecture with ROB.

In the Rename Table, $F0 points to ROB6 because the WAW $F0 is solved, while $F4 points to ROB2 because the WAW $F4 is solved.

See also L07: Reorder Buffer & Speculation

EXERCISE 5 – DYNAMIC BRANCH PREDICTION (1 point)

Let's consider the following loop where $R1 is set to 0 and $R2 is set to 40:

```
LOOP:  LD $F0, 0($R1)
       FADD $F4, $F0, $F2
       SD $F4, 0($R1)
       ADDI $R1, $R1, 8
       BNE $R1, $R2, LOOP
```

Let's assume to have a 1-bit BHT as dynamic branch predictor initialized as TAKEN.
How much is the **misprediction rate** of this loop?
(SINGLE ANSWER)
1 point

Answer 1: 99%
Answer 2: 20%
Answer 3: 100%
Answer 4: 2 %
Answer 5: 25 %
Feedback
Answer 2: 20% (TRUE)
The LOOP is executed 5 times. Being the predictor initialized as TAKEN, there is 1 misprediction at the last iteration of the loop: 1 misprediction out of 5 predictions (20% misprediction rate and 80% success rate).

EXERCISE 6 – STATIC BRANCH PREDICTION (1 point)

Let's consider the following loop where $R1 is set to 0 and $R2 is set to 80:

```
LOOP:  LD $F0, 0($R1)
       FADD $F4, $F0, $F2
       SD $F4, 0($R1)
       ADDI $R1, $R1, 8
       BNE $R1, $R2, LOOP
```

Let's consider the BACKWARD TAKEN FORWARD NOT TAKEN static branch prediction.
How much is the **misprediction rate** of this loop?
(SINGLE ANSWER)
1 point

Answer 1: 99%
Answer 2: 10%
Answer 3: 100%
Answer 4: 1 %
Answer 5: 20 %

Feedback
Answer 2: 10% (TRUE)
The LOOP is executed 10 times. For this backward branch, the static prediction is TAKEN. Therefore, we have 1 misprediction only at the last iteration of the loop: 1 misprediction out of 10 predictions (10% misprediction rate and 90% success rate).

EXERCISE 7 – BRANCH PREDICTION (3 EXTRA points) -- OPTIONAL

Let's consider the following code where $R0 has been initialized to 0:

```
INIT:   ADDI $R1, $R0, 0
        ADDI $R2, $R0, 80
        ADDI $R3, $R0, 0
        ADDI $R4, $R0, 40

LOOP1: LD $F0, 0($R1)
        FADD $F4, $F0, $F2
        SD $F4, 0($R1)
        ADDI $R3, $R0, 0
LOOP2: LD $F6, 0($R3)
        FADD $F8, $F6, $F2
        SD $F8, 0($R3)
        ADDI $R3, $R3, 8
        BNE $R3, $R4, LOOP2

        ADDI $R1, $R1, 8
        BNE $R1, $R2, LOOP1
```

Answer to the following questions:

Question 1: How many iterations of LOOP1 and LOOP2 are executed?
(1 point)

Question 2: Let's assume to have a 1 entry 1-bit BHT as dynamic branch predictor (where the two branch instructions collide) initialized as Taken.
How many mispredictions are we going to observe? Please explain.
(1 point)

Question 3: Let's assume to have a 1 entry 2-bit BHT as dynamic branch predictor (where the two branch instructions collide) initialized Strongly Taken.
How many mispredictions are we going to observe? Please explain.
(1 point)

Question 1: How many iterations of LOOP1 and LOOP2 are executed?
(1 point)

Feedback
The outer loop LOOP1 is executed 10 times.
The inner loop LOOP2 is executed 5 times for each iteration of LOOP1
=> Globally LOOP2 is executed 50 times.

Question 2: Let's assume to have a 1 entry 1-bit BHT as dynamic branch predictor (where the two branch instructions collide) initialized as Taken.
How many mispredictions are we going to observe? Please explain
(1 point)

Feedback
Being the predictor initialized as TAKEN, we have a misprediction only at the last iteration (exit) of the inner LOOP2 and the prediction bit is turned to NT. So, for LOOP 2, we have 1 misprediction out of 5 predictions (misprediction rate 20% for LOOP2). Exiting from the inner LOOP2 with the NT prediction generates a misprediction on the BNE-LOOP1 for 9 iterations (except for the last iteration).

When re-entering in LOOP1, the prediction bit was turned to TAKEN when re-entering in the inner LOOP2 as before.

Counting the number of mispredictions, we have 1 misprediction for the BNE-LOOP2 only when exiting from LOOP2 for 10 iterations of the outer LOOP1 therefore 10 mispredictions. For the outer LOOP1, we have 9 mispredictions for BNE-LOOP1 (except for the last iteration). **Globally there are (10 + 9) = 19 mispredictions.**

Given **19 mispredictions** while the branch predictor has been used 60 times (50 times for BNE-LOOP2 and 10 times for BNE-LOOP1) => there are **19 mispredictions out of 60 predictions => 31.67% misprediction rate.**

Question 3: Let's assume to have a 1 entry 2-bit BHT as dynamic branch predictor (where the two branch instructions collide) initialized as Strongly Taken.
How many mispredictions are we going to observe? Please explain.
(1 point)
Feedback
Being the predictor initialized as Strongly Taken, the predictor fails and changes to the Weakly Taken state only at the last iteration (exit) of the inner LOOP2, but the prediction bit is still Taken So, for LOOP 2, we have 1 misprediction out of 5 predictions (misprediction rate 20% for LOOP2). Exiting from the inner LOOP2 as Weakly Taken, the prediction as TAKEN is correct for the BNE-LOOP1 for 9 iterations (except for the last iteration).

When re-entering in LOOP 1, the prediction state was turned to Strongly Taken when re-entering in the inner LOOP2 as before.

Counting the number of mispredictions, we have 1 misprediction for the BNE-LOOP2 only when exiting from LOOP2 for 10 iterations of the outer LOOP1 therefore 10 mispredictions. For the outer LOOP1, we have only 1 misprediction for BNE-LOOP1 at the last iteration. **Globally, there are (10 + 1) = 11 mispredictions.**

Given **11 mispredictions** while the branch predictor has been used 60 times (50 times for BNE-LOOP2 and 10 times for BNE-LOOP1) => there are **11 mispredictions out of 60 predictions => 18.33% misprediction rate.** As expected, the behavior of the 2-bit BHT is better than the previous 1-bit BHT.

Nome file: ACA_midterm_exam_2023_05_08_with_sol.docx
Directory: /Users/Cristina/Library/Containers/com.microsoft.Word/Data/Library/Preferences/AutoRecovery
Modello: /Users/Cristina/Library/Group Containers/UBF8T346G9.Office/User Content.localized/Templates.localized/Normal.dotm
Titolo:
Oggetto: Prof. C. Silvano
Autore: silvano
Parole chiave:
Commenti:
Data creazione: 19/09/16 18:51:00
Numero revisione: 128
Data ultimo salvataggio: 01/06/23 22:37:00
Autore ultimo salvataggio: Cristina Silvano
Tempo totale modifica 1.671 minuti
Data ultima stampa: 01/06/23 22:38:00
Come da ultima stampa completa
Numero pagine: 14
Numero parole: 2.333
Numero caratteri: 12.918 (circa)

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

Politecnico di Milano, 15 February, 2024

Course on Advanced Computer Architectures

Prof. C. Silvano

EX1	**(6 points)**	
EX2	**(4 points)**	
EX3	**(3 points)**	
EX4	**(2 points)**	
Q1	**(5 points)**	
Q2	**(5 points)**	
QUIZZES	**(8 points)**	
TOTAL	**(33 points)**	

EXERCISE 1 – Tomasulo (6 points)

Please consider one iteration of the following assembly loop code:

```
LOOP:  lw $t1,VECTA($t0)
I2:    lw $t2,VECTB($t0)
I3:    add $t3,$t1,$t2
I4:    add $t4,$t3,$t3
I5:    sw $t3,VECTC($t0)
I6:    sw $t4,VECTD($t0)
I7:    addi $t0,$t0,4
I8:    bne $t0,$t5,LOOP
```

to be executed on a ***CPU1*** with dynamic scheduling based on **TOMASULO algorithm** with all cache HITS, a single Common Data Bus and:

- *2 RESERVATION STATIONS **(RS1, RS2)** for 2 LOAD/STORE units **(LDU1, LDU2)** with latency **6***
- *2 RESERVATION STATIONS **(RS3, RS4)** for 2 ALU/BR FUs **(ALU1, ALU2)** with latency **2***

1. *Please complete the following table* for ***CPU1***:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`LOOP:lw $t1,VECTA($t0)`	1	2	8	None	RS1	LDU1
`I2: lw $t2,VECTB($t0)`	2	3	9	None	RS2	LDU2
`I3: add $t3,$t1,$t2`						
`I4: add $t4,$t3,$t3`						
`I5: sw $t3,VECTC($t0)`						
`I6: sw $t4,VECTD($t0)`						
`I7: addi $t0,$t0,4`						
`I8: bne $t0,$t5,LOOP`						

2. *Calculate the **CPI1** :*

CPI1 **= _____**

Let's assume to execute the same loop iteration on a ***CPU$_2$*** with dynamic scheduling based on **TOMASULO algorithm** with all cache HITS, a single Common Data Bus and **more reservation stations** as follows:

- *4 RESERVATION STATIONS **(RS1, RS2, RS3, RS4)** for 2 LOAD/STORE units **(LDU1, LDU2)** with latency **6***
- *4 RESERVATION STATIONS **(RS5, RS6, RS7, RS8)** for 2 ALU/BR FUs **(ALU1, ALU2)** with latency **2***

3. *Please complete the following table* for ***CPU$_2$***:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`LOOP:lw $t1,VECTA($t0)`	1	2	8	None	RS1	LDU1
`I2: lw $t2,VECTB($t0)`	2	3	9	None	RS2	LDU2
`I3: add $t3,$t1,$t2`						
`I4: add $t4,$t3,$t3`						
`I5: sw $t3,VECTC($t0)`						
`I6: sw $t4,VECTD($t0)`						
`I7: addi $t0,$t0,4`						
`I8: bne $t0,$t5,LOOP`						

4. *Calculate the **CPI$_2$** :*

CPI$_2$ = _____

5. *Calculate the **Speedup** obtained by adding more reservation stations:*

Speedup = _____

Feedback:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`LOOP:lw $t1,VECTA($t0)`	1	2	8	None	RS1	LDU1
`I2: lw $t2,VECTB($t0)`	2	3	9	None	RS2	LDU2
`I3: add $t3,$t1,$t2`	3	10	12	(RAW $t1) **RAW $t2**	RS3	ALU1
`I4: add $t4,$t3,$t3`	4	13	15	**RAW $t3**	RS4	ALU2
`I5: sw $t3,VECTC($t0)`	9	13	19	**STRUCT RS1 RAW $t3**	RS1	LDU1
`I6: sw $t4,VECTD($t0)`	10	16	22	**RAW $t4**	RS2	LDU2
`I7: addi $t0,$t0,4`	13	14	16	**STRUCT RS3**	RS3	ALU1
`I8: bne $t0,$t5,LOOP`	16	17	19	**STRUCT RS4** (RAW $t0)	RS4	ALU2

CPI_1 = # clock cycles / IC = 22 / 8 = 2,75

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`LOOP:lw $t1,VECTA($t0)`	1	2	8	None	RS1	LDU1
`I2: lw $t2,VECTB($t0)`	2	3	9	None	RS2	LDU2
`I3: add $t3,$t1,$t2`	3	10	12	(RAW $t1) **RAW $t2**	RS5	ALU1
`I4: add $t4,$t3,$t3`	4	13	15	**RAW $t3**	RS6	ALU2
`I5: sw $t3,VECTC($t0)`	5	13	19	**RAW $t3**	RS3	LDU1
`I6: sw $t4,VECTD($t0)`	6	16	22	**RAW $t4**	RS4	LDU2
`I7: addi $t0,$t0,4`	7	13	16	**STRUCT ALU1 + CDB**	RS7	ALU1
`I8: bne $t0,$t5,LOOP`	8	17	19	**RAW $t0**	RS8	ALU2

CPI_2 = # clock cycles / IC = 22 / 8 = 2,75

*There is **no speedup** in adding more reservation stations.*

In Tomasulo, the potential WAW and WAR hazards are already solved by Register Renaming

Only the hazards that have caused the introduction of some stalls are reported in the table, while other potential hazards are either omitted or put into brackets.

EXERCISE 2 on REORDER BUFFER (4 points)

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

```
LOOP:  LD $F2, 0 ($R1)
I2:    ADDD $F4, $F2, $F2
I3:    SD $F4, 0 ($R1)
I4:    ADDI $R1, $R1, 4
I5:    BNE $R1, $R2, LOOP # branch predicted ad taken
```

- *How many loop iterations? 10 iterations*

Then execute the loop by the ***Speculative Tomasulo*** *architecture with an* ***8-entry ROB*** *and:*

- *4 Load Buffers (Load1, Load2, Load3, Load4);*
- *2 FP Reservation Stations (FP1, FP2)*
- *2 Integer Reservation Stations (Int1, Int2)*

- *Please update the following ROB and Rename Tables 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	LOOP: LD $F2, 0 ($R1)	$F2	Load1	No, exec. (*)	No	HEAD
ROB1	I2: ADDD $F4, $F2, $F2	$F4	FP1	No, issued	No	
ROB2	I3: SD $F4, 0 ($R1)	Mem	--	No, issued	No	
ROB3	I4: ADDI $R1, $R1, 8	$R1	Int1	No, issued	No	
ROB4	I5: BNE $R1, $R2, LOOP	--	Int2	No, issued	No	
ROB5	LOOP: LD $F2, 0 ($R1)	$F2	Load 2	No, issued	Yes	
ROB6	I2: ADDD $F4, $F2, $F2	$F4	FP2	No, issued	Yes	
ROB7	I3: SD $F4, 0 ($R1)	Mem	--	No, issued	Yes	

Rename Table:

Reg#	ROB#
$F0	--
$F2	~~ROB0,~~ ROB5
$F4	~~ROB1,~~ ROB6
$F6	--
$F8	--

Please complete the unrolled version of the loop with unrolling factor 2:

```
LOOP: LD $F2, 0 ($R1)
I2:   ADDD $F4, $F2, $F2
I3:   SD $F4, 0 ($R1)
I4:   LD $F6, 4 ($R1)
I5:   ADDD $F8, $F6, $F6
I6:   SD $F8, 4 ($R1)
I7:   ADDI $R1, $R1, 8
I8:   BNE $R1, $R2, LOOP
```

- *How many loop iterations? 5 iterations*

---

*Then execute the unrolled version of the loop by the **Speculative Tomasulo** architecture with an **8-entry ROB** and:*

- *4 Load Buffers (Load1, Load2, Load3, Load4);*
- *2 FP Reservation Stations (FP1, FP2)*
- *2 Integer Reservation Stations (Int1, Int2)*

- *Please update the following ROB and Rename Tables 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	LOOP:LD $F2, 0 ($R1)	$F2	Load1	No, exec. (*)	No	HEAD
ROB1	I2: ADDD $F4, $F2, $F2	$F4	FP1	No, issued	No	
ROB2	I3: SD $F4, 0 ($R1)	Mem	--	No, issued	No	
ROB3	I4: LD $F6, 4 ($R1)	$F6	Load2	No, issued	No	
ROB4	I5: ADDD $F8, $F6, $F6	$F8	FP2	No, issued	No	
ROB5	I6: SD $F8, 4 ($R1)	Mem	--	No, issued	No	
ROB6	I7: ADDI $R1, $R1, 8	$R1	Int1	No, issued	No	
ROB7	I8: BNE $R1, $R2, LOOP	--	Int2	No, issued	No	

Rename Table:

Reg#	ROB#
$F0	--
$F2	ROB0
$F4	ROB1
$F6	ROB3
$F8	ROB4

EXERCISE 3 – CACHE MEMORIES (3 points)

Let's consider 32-block main memory with a **fully-associative** 4-block cache and **LRU** cache replacement policy. The addresses are expressed as:

Memory Block Addresses: $[0, 1, 2, \ldots\ldots\ldots\ldots 31]_{10}$

Cache Block Addresses: [a, b, c, d]

At cold start the cache is empty, then there is the following sequence of memory accesses.

- *Assuming the **Write Allocate** and **Write Back** cache policies, complete the following table:*

	Type of memory access	Memory Address	HIT / MISS Type	Cache Tag	Cache Block Address	Dirty bit	Write-back in memory
1	read	$[12]_{10}$	Cold start MISS	$[01100]_2$	a	0	no
2	write	$[12]_{10}$	HIT	$[01100]_2$	a	1	no
3	write	$[10]_{10}$	*Cold start MISS*	$[01010]_2$	*b*	*1*	*no*
4	read	$[10]_{10}$	*HIT*	$[01010]_2$	*b*	*1*	*no*
5	read	$[26]_{10}$	*Cold start MISS*	$[11010]_2$	*c*	*0*	*no*
6	write	$[7]_{10}$	*Cold start MISS*	$[00111]_2$	*d*	*1*	*no*
7	read	$[12]_{10}$	*HIT*	$[01100]_2$	*a*	*1*	*no*
8	read	$[4]_{10}$	*Conflict MISS*	$[00100]_2$	*b*	*0*	*Yes: WB in* $[10]_{10}$
9	write	$[26]_{10}$	*HIT*	$[11010]_2$	*c*	*1*	*no*
10	write	$[18]_{10}$	*Conflict MISS*	$[10010]_2$	*d*	*1*	*Yes: WB in* $[7]_{10}$
11	read	$[5]_{10}$	*Conflict MISS*	$[00101]_2$	*a*	*0*	*Yes: WB in* $[12]_{10}$
12	write	$[24]_{10}$	*Conflict MISS*	$[11000]_2$	*b*	*1*	*No*

Calculate the Miss Rate:

Miss Rate = Number of misses / Number of memory access = 8 / 12= 0,67

EXERCISE 4 – VECTOR PROCESSORS (2 points)

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, RX          # Load vector from memory address RX into V1
L.V V3, RY          # Load vector from memory address RY 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 V3, RZ          # Store vector V3 into memory address RZ
```

Please complete the following vector instruction scheduling:

0	**1**		**L.V V1, RX**	**7**

How many convoys? 3 convoys

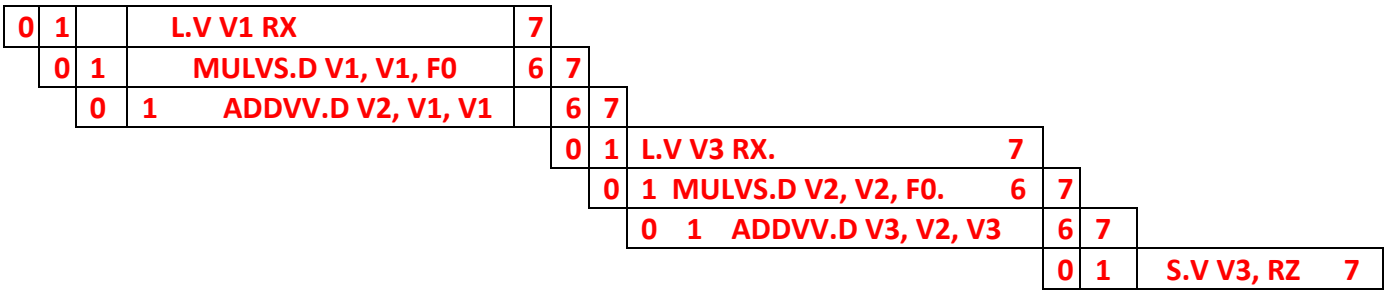
_____

How many clock cycles to execute the code? 24 clock cycles

_____

Feedback:

Vector instruction scheduling:



There are 3 convoys as follows:

1) L.V V1, RX; MULVS.D V1, V1, F0 ADDVV.D V2, V1, V1
2) L.V V3, RY MULVS.D V2, V2, F0 ADDVV.D V3, V2, V3
3) S.V V3, RZ

QUESTION 1: VLIW ARCHITECTURES (5 points)

1. Describe the main concepts of Very Long Instruction Word processor architectures.

2. What are the main issues associated with VLIW architectures?
For each answer mark if it is either **TRUE** or **FALSE** and **motivate your answer**)

Answer 1: Compilers can detect parallelism only in basic blocks of the code **T** **F**

Answer 2: Larger code size **T** **F**

Answer 3: Increased HW complexity **T** **F**

Answer 4: Less code portability **T** **F**

Answer 5: Large number of registers required for register renaming **T** **F**

QUESTION 2: MEMORY HIERARCHY (5 points)

Let's consider the following cache optimization techniques. *Answer to the following questions:*

	Pseudo-associativity and Way Prediction	**Early Restart and Critical Word First**	**Introducing a Victim Cache**
*Explain the **main concepts** for each technique.*			
*What are the main effects of each technique on the **miss rate?***			
*What are the main effects of each technique on the **miss penalty?***			

MULTIPLE-CHOICE QUESTIONS: (8 points)

Question 1 (format Multiple Choice – Single answer)

*In the **speculative Tomasulo architecture**, what are the hardware blocks needed to undo speculative instructions execution on mispredicted branches?*
(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;

Feedback:
The speculative Tomasulo architecture supports speculative instructions execution and branch mispredictions by using ROB entries.

Question 2 (format TRUE/FALSE)

*A **fine-grain multi-threading** processor* exploits more parallelism than a simultaneous multi-threading processor
(TRUE/FALSE)
1 point

Answer: TRUE FALSE **(X)**

Question 3 (format TRUE/FALSE)

*A **VLIW** processor needs multiple Program Counters to load the necessary multiple instructions.*
(TRUE/FALSE)
1 point

Answer: TRUE FALSE **(X)**

Question 4 (format Multiple Choice – Single answer)

Let's consider a 5-issue processor that can manage up to 5 simultaneous threads.
What are the values of the ideal CPI and the ideal per-thread CPI?
(SINGLE ANSWER)
1 point

Answer 1: Ideal CPI = 5 & Ideal per-thread CPI = 1
Answer 2: Ideal CPI = 1 & Ideal per-thread CPI =0.2
Answer 3: Ideal CPI = 0.20 & Ideal per-thread CPI = 1 **(TRUE)**
Answer 4: Ideal CPI = 0.25 & Ideal per-thread CPI = 5
Answer 5: Ideal CPI = 5 & Ideal per-thread CPI = 0.20

Question 5 (format Multiple Choice – Single answer)

Let's consider a (2,1) Correlating Branch Predictor with 4K total entries.
How many Branch History Tables?
How many entries per BHT?
How many bits per entry?
(SINGLE ANSWER)
1 point

Answer 1: 4 BHTs | 1K entries | 1-bit per entry **(TRUE)**
Answer 2: 2 BHTs | 2K entries | 1-bit per entry
Answer 3: 2 BHTs | 2K entries | 2-bit per entry
Answer 4: 1 BHT | 4K entries | 1-bit per entry

Question 6 (format Multiple Choice – Single answer)

*Using a 5-stage **optimized pipeline with the "early evaluation of PC"** for branch instructions, how many stalls we need to introduce to execute the following assembly code?*

```
ld $t1, AA($t6)
beq $t1, $t0, LABEL
```

Answer 1: 1
Answer 2: 2 **(TRUE)**
Answer 3: 3
Answer 4: none

(SINGLE ANSWER)
1 point

Complete the following pipeline scheme to motivate your answer:

`ld $t1, AA($t6)`	IF	ID	EX	ME	WB				
`beq $t1, $t0, LABEL`		IF	ID stall	ID stall	ID	EX	ME	WB	

Question 7 (complete table format)
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

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

Politecnico di Milano, 22 January, 2024

Course on Advanced Computer Architectures

Prof. C. Silvano

EX1	**(5 points)**	
EX2	**(4 points)**	
EX3	**(3 points)**	
EX4	**(3 points)**	
Q1	**(5 points)**	
Q2	**(5 points)**	
QUIZZES	**(8 points)**	
TOTAL	**(33 points)**	

EXERCISE 1 – Tomasulo (5 points)

Please consider the following assembly code:

```
I1:  lw $t2,VECTA($t6)
I2:  lw $t3,VECTB($t6)
I3:  lw $t4,VECTC($t6)
I4:  addi $t2,$t2,k
I5:  sw $t2,VECTA($t6)
I6:  add $t3,$t2,$t3
I7:  add $t4,$t3,$t4
I8:  sw $t4,VECTC($t6)
```

to be executed on a CPU with dynamic scheduling based on **TOMASULO algorithm** with all cache HITS, a single Common Data Bus and:

- *4 RESERVATION STATIONS **(RS1, RS2, RS3, RS4)** for 2 LOAD/STORE unit **(LDU1, LDU2)** with latency **6***
- *2 RESERVATION STATIONS **(RS5, RS6)** for 2 ALU/BR FUs **(ALU1, ALU2)** with latency **2***

1. *Please complete the following table:*

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`I1: lw $t2,VECTA($t6)`	1	2	8	None	RS1	LDU1
`I2: lw $t3,VECTB($t6)`	2	3	9	None	RS2	LDU2
`I3: lw $t4,VECTC($t6)`						
`I4: addi $t2,$t2,k`						
`I5: sw $t2,VECTA($t6)`						
`I6: add $t3,$t2,$t3`						
`I7: add $t4,$t3,$t4`						
`I8: sw $t4,VECTC($t6)`						

2. *Calculate the CPI:*

CPI = _____

Feedback:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
I1: lw $t2,VECTA($t6)	1	2	8	None	RS1	LDU1
I2: lw $t3,VECTB($t6)	2	3	9	None	RS2	LDU2
I3: lw $t4,VECTC($t6)	3	9	15	**STRUCT LDU1**	RS3	LDU1
I4: addi $t2,$t2,k	4	9	11	**RAW $t2**	RS5	ALU1
I5: sw $t2,VECTA($t6)	5	12	18 *	(STRUCT LDU2) **RAW $t2**	RS4	LDU2
I6: add $t3,$t2,$t3	6	12	14	**RAW $t2** (RAW $t3)	RS6	ALU2
I7: add $t4,$t3,$t4	12	16	18	**STRUCT RS5 + RAW $t4** (RAW $t3)	RS5	ALU1
I8: sw $t4,VECTC($t6)	13	19	25	(STRUCT LDU1) **RAW $t4**	RS1	LDU1

(*) Being a store instruction, it does not use the CDB at cycle 18

In Tomasulo, the potential WAW and WAR hazards are already solved by Register Renaming

Only the hazards that have caused the introduction of some stalls are reported in the table, while other potential hazards are put into brackets.

CPI = # clock cycles / IC = 25 / 8 = 3.125

EXERCISE 2 on REORDER BUFFER (4 points)

Let's consider the following assembly code:

```
I0:  LD $F6, VECTA ($R1)
I1:  LD $F4, VECTB ($R1)
I2:  ADDD $F6, $F6, $F8
I3:  SUBD $F4, $F4, $F8
I4:  SD $F6, VECTA ($R1)
I5:  SD $F4, VECTB ($R1)
```

*executed by the **Speculative Tomasulo** architecture with an **8-entry ROB** and:*

- *4 load buffers (Load1, Load2, Load3, Load4);*
- *2 FP RS (FP1, FP2)*
- *2 Integer RS (Int1, Int2)*

- *Please update the following ROB and Rename Tables until the ROB becomes full and I0 is still in execution due to a cache miss:*

ROB Table

ROB#	Instruction	Dest.	Ready	SPECUL.	
ROB0	`I0:  LD $F6, VECTA ($R1)`	$F6	No (miss in exec.)	No	HEAD
ROB1	`I1:  LD $F4, VECTB ($R1)`	$F4	No (issued)	No	
ROB2					
ROB3					
ROB4					
ROB5					
ROB6					
ROB7					

Rename Table:

Reg#	ROB#
$F0	--
$F2	--
$F4	ROB1
$F6	ROB0
$F8	--

Feedback:

ROB#	Instruction	Dest.	Ready	SPECUL.	
ROB0	I0: LD $F6, VECTA ($R1)	$F6	No (miss in exec.)	No	HEAD
ROB1	I1: LD $F4, VECTB ($R1)	$F4	No (issued)	No	
ROB2	**I2: ADDD $F6, $F6, $F8**	**$F6**	**No (issued)**	**No**	
ROB3	**I3: SUBD $F4, $F4, $F8**	**$F4**	**No (issued)**	**No**	
ROB4	**I4: SD $F6, VECTA ($R1)**	**MEM**	**No (issued)**	**No**	
ROB5	**I5: SD $F4, VECTB ($R1)**	**MEM**	**No (issued)**	**No**	
ROB6					**TAIL**
ROB7					

Rename Table:

$F0	--
$F2	--
$F4	~~**ROB1**~~ **ROB3**
$F6	~~**ROB0**~~ **ROB2**
$F8	

EXERCISE 3 – CACHE MEMORIES (3 points)

Let's consider a fully associative cache with 4 blocks [a, b, c, d] that at cold start is empty and receives the following sequence of memory accesses:

```
Read  Mem[AAAA]
Write Mem[AAAA]
Read  Mem[BBBB]
Write Mem[BBBB]
Read  Mem[AAAA]
Write Mem[CCCC]
Read  Mem[CCCC]
Read  Mem[BBBB]
```

- *Assuming the **Write Allocate** and **Write Back** cache policies, complete the following table:*

	Type of memory access	Memory Address	HIT / MISS Type	Cache Tag	Cache Block	Dirty bit	Write-back in memory
1	read	[AAAA] ex	Cold start MISS	[AAAA] ex	a	0	no
2	write	[AAAA] ex	HIT	[AAAA] ex	a	1	no
3	read	[BBBB] ex	*Cold start MISS*	[BBBB] ex	*b*	*0*	*no*
4	write	[BBBB] ex	*HIT*	[BBBB] ex	*b*	*1*	*no*
5	read	[AAAA] ex	*HIT*	[AAAA] ex	*a*	*1*	*no*
6	write	[CCCC] ex	*Cold start MISS*	[CCCC] ex	*c*	*1*	*no*
7	read	[CCCC] ex	*HIT*	[CCCC] ex	*c*	*1*	*no*
8	write	[BBBB] ex	*HIT*	[BBBB] ex	*b*	*1*	*no*

How many cache misses?

3

How many cache hits?

5

Calculate the Miss Rate:

Miss Rate = Number of misses / Number of memory access = 3 / 8= 0.375

- *Assuming the **No-write Allocate** and **Write Through** cache policies, complete the following table:*

	Type of memory access	Memory Address	HIT / MISS Type	Cache Tag	Cache Block	Write in memory
1	read	[AAAA] ex	Cold start MISS	[AAAA] ex	a	no
2	write	[AAAA] ex	HIT	[AAAA] ex	a	yes
3	read	[BBBB] ex	*Cold start MISS*	[BBBB] ex	*b*	*no*
4	write	[BBBB] ex	*HIT*	[BBBB] ex	*b*	*yes*
5	read	[AAAA] ex	*HIT*	[AAAA] ex	*a*	*no*
6	write	[CCCC] ex	**Miss with no-write allocation in cache**	--	--	*Yes (*)*
7	read	[CCCC] ex	*Cold start MISS*	[CCCC] ex	*c*	*no*
8	write	[BBBB] ex	*HIT*	[BBBB] ex	*b*	*yes*

(*) Write done directly in memory with no-write allocation in cache

How many cache misses?

4

How many cache hits?

4

Calculate the Miss Rate:

Miss Rate = Number of misses / Number of memory access = 4 / 8 = 0.5

EXERCISE 4 – CACHE MEMORIES (3 points)

Let's consider 32-block main memory with a **direct-mapped** 8-block cache based on a ***write allocate*** with ***write-back*** protocol.

The addresses are expressed as decimal numbers:

Memory Block Addresses: [0, 1, 2, …… 31] $_{10}$

Cache Block Addresses: [0, 1, 2, …7] $_{10}$

At cold start the cache is empty, then there is the following sequence of memory accesses.

1. *Please complete the following table:*

	Type of memory access	Memory Address	HIT / MISS Type	Cache Tag	Cache Index	Dirty bit	Write-back in memory
1	write	[12] $_{10}$	Cold start MISS	[01] $_2$	[100] $_2$	1	no
2	read	[12] $_{10}$	HIT	[01] $_2$	[100] $_2$	1	no
3	read	[10] $_{10}$	*Cold start MISS*	*[01] $_2$*	*[010] $_2$*	*0*	*no*
4	write	[10] $_{10}$	*HIT*	*[01] $_2$*	*[010] $_2$*	*1*	*no*
5	write	[26] $_{10}$	*Conflict MISS*	*[11] $_2$*	*[010] $_2$*	*1*	*WB in M[10] $_{10}$*
6	read	[20] $_{10}$	*Conflict MISS*	*[10] $_2$*	*[100] $_2$*	*0*	*WB in M[12] $_{10}$*
7	read	[30] $_{10}$	*Cold start MISS*	*[11] $_2$*	*[110] $_2$*	*0*	*no*
8	write	[16] $_{10}$	*Cold start MISS*	*[10] $_2$*	*[000] $_2$*	*1*	*no*
9	write	[6] $_{10}$	*Conflict MISS*	*[00] $_2$*	*[110] $_2$*	*1*	*no (block 30 not modified)*
10	read	[18] $_{10}$	*Conflict MISS*	*[10] $_2$*	*[010] $_2$*	*0*	*WB in M[26] $_{10}$*

2. *Calculate the Miss Rate:*

Miss Rate = Number of misses / Number of memory access = 8/10 = 0.8

QUESTION 1: DYNAMIC BRANCH PREDICTION (5 points)

Assume a pipelined processor with a dynamic branch prediction unit composed of a **1-entry 1-bit Branch History Table** in the **IF-stage.**

Disabling the branch prediction unit, each branch costs **2 cycles penalty** to fetch the correct instruction.

Enabling the branch prediction unit, there are 4 cases for each conditional branch with the related **branch penalty cycles:**

Branch Outcome Prediction	Branch Outcome	Branch Penalty Cycles
Predicted Not Taken	Not Taken	0
Predicted Not Taken	Taken	2 (misprediction)
Predicted Taken	Not Taken	2 (misprediction)
Predicted Taken	Taken	1

Let's consider the following assembly loop:

```
INIT:  ADDUI $R1, $R0, 0
       ADDUI $R2, $R0, 40

LOOP:  LD $F0, 0 ($R1)
       FADD $F4, $F0, $F2
       SD $F4, 0 ($R1)
       ADDUI $R1, $R1, 4
       BNE $R1, $R2, LOOP
```

1. *How many loop iterations?*

   10 iterations _____

2. *Please complete the following table:*

	Explain the branch behavior in the loop.	***How many branch penalty cycles are needed to execute the loop?***	***Calculate the branch misprediction rate to execute the loop***
Assuming the BHT predictor is enabled and initialized as **Not Taken**	In this case, we have a first misprediction with PNT/T with 2 cycles penalty. Then there are 8 iterations predicted correctly as PT/T with 1 cycle penalty. The last iteration is mispredicted as PT/NT with 2 cycles penalty.	There are: (2 + 8 + 2) = 12 branch penalty cycles.	There are 2 mispredictions out of 10 predictions => 20% misprediction rate;
Assuming the BHT predictor is enabled and initialized as **Taken**	In this case, we have 9 iterations correctly predicted as PT/T with 1 cycle penalty The last iteration is mispredicted as PT/NT with 2 cycles penalty.	There are: (9 + 2) = 11 branch penalty cycles.	There is 1 misprediction out of 10 predictions => 10% misprediction rate.
Assuming the BHT predictor is disabled.	At each iteration, each branch costs 2 cycle penalty to fetch the correct instruction.	There are 20 branch penalty cycles to execute the loop.	

QUESTION 2: STATIC and DYNAMIC SCHEDULING (5 points)

Let's consider the two types of instruction scheduling techniques: static and dynamic.

Answer to the following questions:

	Static Instruction Scheduling	**Dynamic Instruction Scheduling**
Explain the main concepts for each technique.		
What are the main benefits for each technique?		

What are the main drawbacks for each technique?		
What are the main hardware resources needed to implement each technique in a pipelined processor?		

MULTIPLE-CHOICE QUESTIONS: (8 points)

Question 1 (format Multiple Choice – Multiple answers)

*How does a MESI write-invalidate write-back protocol manage a **Write Hit** on an **Exclusive** cache block?*
(MULTIPLE ANSWERS)
1 point

Answer 1: An invalidate is broadcasted on the bus to the other copies of the block;
Answer 2: The cache block is updated from another cache owner of the block;
Answer 3: The cache block is updated with the new value; **(TRUE)**
Answer 4: The status of the cache block becomes Modified; **(TRUE)**
Answer 5: The cache block is updated from the memory;

Feedback:
On a Write Hit on an Exclusive cache block means that the block is clean and there are no other copies of the block in other caches. Therefore, there is no need to send an invalidate signal on the snooping bus to other caches. We have that:

- *The processor's cache is updated with the new data values.*
- *The status of the cache block becomes Modified.*

Question 2 (format Multiple Choice – Single answer)

Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where we consider the block B0 in the directory of N0:

Directory N0 Block B0 | State: Shared | Sharer Bits: 0011 |

*Which are the State and the Sharer Bits of the block **B0** in **N0** after this sequence:*
Read Miss B0 from local cache N0;
Read Miss B0 from local cache N1;
(SINGLE ANSWER)
1 point

Answer 1: Directory N0 Block B0 | State: Shared | Sharer Bits: 1100 |
Answer 2: Directory N0 Block B0 | State: Modified | Sharer Bits: 0100 |
Answer 3: Directory N0 Block B0 | State: Modified | Sharer Bits: 1000 |
Answer 4: Directory N0 Block B0 | State: Shared | Sharer Bits: 1111 | **(TRUE)**

Question 3 (format Multiple Choice – Single answer)

Let's consider a 4-issue VLIW processor with 2 Load/Store Units, 2 FP Units and 2 Integer/Branch Units. What is needed to redirect operations and operands to the corresponding functional unit?
(SINGLE ANSWER)
1 point
Answer 1: Dynamic Instruction Scheduler;

Answer 2: Dispatch Network; **(TRUE)**

Answer 3: Common data bus;

Answer 4: Thread Scheduler;

Question 4 (format Multiple Choice – Multiple answers)

To make dynamic loop unrolling in the speculative Tomasulo architecture, what are the hardware blocks needed to eliminate WAR and WAW hazards?
(MULTIPLE ANSWERS)
1 point

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

Answer 2: Rename Table; **(TRUE)**

Answer 3: Store Buffer;

Answer 4: Thread Scheduler;

Answer 5: Vector Processing Unit;

Feedback:
The speculative Tomasulo architecture supports Register Renaming by using ROB entries and the Rename Table to eliminate WAR and WAW hazards and make dynamic loop unrolling.

Question 5 (format Multiple Answers)

*For a **fine-grain multi-threading** processor, which of the following statements are true?*
(MULTIPLE ANSWERS)
2 points

Answer 1: The processor switches between threads at each instruction by skipping any thread that is stalled at that time. **(TRUE)**

Answer 2: The processor exploits more parallelism than simultaneous *multi-threading.*

Answer 3: The processor requires to use a multiple issue processor to exploit both ILP and TLP.

Answer 4: Compared to coarse-grain MT, the processor slows down the execution of individual threads, since a thread will be delayed by another thread even if it is ready to execute.
(TRUE)

Answer 5: The processor has a higher overhead with respect to coarse-grained MT due to more frequent hardware context switches. **(TRUE)**

Question 6 (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, RX          # Load vector from memory address RX into V1
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
L.V V3, RY          # Load vector from memory address RY into V2
ADDVV.D V3, V2, V3  # FP add vectors V2 and V3
S.V V3, RZ          # Store vector V3 into memory address RZ
```

How many convoys? How many clock cycles to execute the code?

(SINGLE ANSWER)
2 points

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

Feedback: There are 3 convoys:

1) L.V V1, RX; MULVS.D V1, V1, F0 ADDVV.D V2, V1, V1
2) L.V V3, RY MULVS.D V2, V2, F0 ADDVV.D V3, V2, V3
3) S.V V3, RZ

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

Politecnico di Milano, 31 August, 2023

Course on Advanced Computer Architectures

Prof. C. Silvano

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

EXERCISE 1 – VLIW (5 points)

Let's consider the following LOOP code:

```
Loop: LD F0,0(R2)
      LD F4,0(R3)
      FMUL F0,F0,F4
      FADD F2,F2,F4
      ADDUI R2,R2,#8
      ADDUI R3,R3,#8
      SUBUI R5,R4,R2
      BNEZ R5,Loop
```

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

- **1 Memory Units with 3 cycles latency**
- **1 FP ALUs with 3 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.

1) *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 *3-issue VLIW machine including the BRANCH DELAY SLOT.* Please do not write in NOPs.

	Memory Unit	Floating Point Unit	Integer Unit
C1	`LD F0,0(R2)`		
C2			
C3			
C4			
C5			
C6			
C7			
C8			
C9			
C10			
C11			
C12			
C13			
C14			
C15			

2. *How long is the critical path for a single iteration?*

   _____

3. *What performance did you achieve in CPI?*

   _____

4. *What performance did you achieve in FP ops per cycles?*

   _____

5. *How much is the code efficiency?*

   _____

6. *Assuming to have **2 FLOATING POINT UNITS** instead of one, how much is the impact on CPI and code efficiency?*

_____

_____

_____

_____

_____

_____

Feedback

	Memory Unit	Floating Point Unit	Integer Unit
C1	`LD F0,0(R2)`		`ADDUI R2,R2,#8`
C2	`LD F4,0(R3)`		`ADDUI R3,R3,#8`
C3			`SUBUI R5,R4,R2`
C4			
C5		`FMUL F0,F0,F4`	`BNEZ R5,Loop`
C6		`FADD F2,F2,F4`	`(br. delay slot)`
C7			
C8			

1. *How long is the critical path for a single iteration? (*)*

   8 cycles (needed to conclude the instruction FADD F2, F2,F4)

   (*) When considering many iterations, the next LF F0 could start at cycle C8, therefore the critical path would be 7 cycles and the $CPI_{AS} = 7 / 8$

2. *What performance did you achieve in CPI?* CPI = # cycles / IC = 8 / 8 = 1

3. *What performance did you achieve in FP ops per cycles?*

   FP ops per cycles = # FP ops / # cycles = 2 / 8 = 0.25

4. *How much is the code efficiency?*

   Code efficiency = IC / (# cycles x # issues) = 8 / (8 x 3) = 8 / 24 = 1 / 3 = 0.33

5. *Assuming to have **2 FLOATING POINT UNITS** instead of one, how much is the impact on CPI and code efficiency?* FMUL and FADD can be scheduled in parallel, therefore we have a critical path for a single iteration of 7 cycles with a slightly improved CPI = 7 / 8 = 0.875 and a slightly worst code efficiency = 8 / (7 x 4) = 2 / 7 = 0.29

EXERCISE 2 – CACHE MEMORIES (5 points)

Let's consider 32-block main memory with a **2-way set associative** 8-block cache based on a ***write allocate*** with ***write-back*** protocol.

The addresses are expressed as:
Memory Address: [0, 1, 2, ……. 31] $_{10}$
Cache Address: [a, b, c, d, e, f, g, h]
and Cache Tags are expressed as binary numbers.

1) *How many sets are in the cache?*

_____

2) *Being set-associative, what is the most used block replacement policy?*

_____

3) At cold start the cache is empty, then there is the following sequence of memory accesses.

Please complete the following table:

	Read / Write	Memory Address	HIT/MISS Type	Cache Tag	Cache Address	Set	Write in memory
1	Write	[12] $_{10}$	Cold-start Miss				
2	Read	[12] $_{10}$					
3	Read	[10] $_{10}$					
4	Write	[10] $_{10}$					
5	Write	[26] $_{10}$					
6	Read	[14] $_{10}$					
7	Write	[30] $_{10}$					
8	Write	[16] $_{10}$					
9	Read	[18] $_{10}$					
10	Read	[24] $_{10}$					

4) *How much is the Miss Rate?*

_____

Feedback

1) *How many sets are in the cache?*

The 2-way set-associative cache has 8-blocks [a, b, c, d, e, f, g, h] organized in **4 sets,** where each set contains **2 blocks** as follows: **Set_0: [a , b] Set_1 [c, d]; Set_2: [e, f] Set_ 3 [g, h]**

Memory Address Mapping:

Tags	Set_0 [a , b]	Set_1 [c, d]	Set_2 [e, f]	Set_1 [g, h]
000	0	1	2	3
001	4	5	6	7
010	8	9	10	11
011	12	13	14	15
	…	…	…	…
111	28	29	30	31

2) *Being set-associative, what is the most used block replacement policy?*

The block replacement policy in each set uses the **Least Recently Used (LRU) algorithm**.

3) *Please complete the following table:*

	Read / Write	Memory Address	HIT / MISS Type	Cache Tag	Cache Address	Set	Write in memory
1	Write	$[12]_{10}$	Cold-start Miss	$[011]_2$	a	$[0]_{10}$	No
2	Read	$[12]_{10}$	Hit	$[011]_2$	a	$[0]_{10}$	No
3	Read	$[10]_{10}$	Cold-start Miss	$[010]_2$	e	$[2]_{10}$	No
4	Write	$[10]_{10}$	Hit	$[010]_2$	e	$[2]_{10}$	No
5	Write	$[26]_{10}$	Cold-start Miss	$[110]_2$	f	$[2]_{10}$	No
6	Read	$[14]_{10}$	Conflict Miss	$[011]_2$	e	$[2]_{10}$	Yes Wr. in $M[10]_{10}$
7	Write	$[30]_{10}$	Conflict Miss	$[111]_2$	f	$[2]_{10}$	Yes Wr. in $M[26]_{10}$
8	Write	$[16]_{10}$	Cold-start Miss	$[100]_2$	b	$[0]_{10}$	No
9	Read	$[18]_{10}$	Conflict Miss	$[100]_2$	e	$[2]_{10}$	No
10	Read	$[24]_{10}$	Conflict Miss	$[110]_2$	a	$[0]_{10}$	Yes Wr. in $M[12]_{10}$

4) *How much is the Miss Rate?*

Miss Rate = Number of misses / Number of memory access = 8/10 = 0.8

EXERCISE 3 - SOFTWARE PIPELINING (2 points)

Given the following software pipelined loop, reconstruct the original (non-pipelined) code:

```
START_UP: LD $F0, 0 ($R1)
          LD $F2, 0 ($R2)
          FADD $F4, $F0, $F0
          FADD $F6, $F2, $F2
          LD $F0, 8 ($R1)
          LD $F2, 8 ($R2)

LOOP: SD $F4, 0 ($R1)
      SD $F6, 0 ($R2)
      FADD $F4, $F0, $F0
      FADD $F6, $F2, $F2
      LD $F0, 16 ($R1)
      LD $F2, 16 ($R2)
      ADDUI $R1, $R1, 8
      ADDUI $R2, $R2, 8
      BNE $R1, $R3, LOOP

FINISH-UP: SD $F4, 8 ($R1)
           SD $F6, 8 ($R2)
           FADD $F4, $F0, $F0
           FADD $F6, $F2, $F2
           SD $F4, 16 ($R1)
           SD $F6, 16 ($R2)
```

Feedback:

```
LOOP: LD $F0, 0 ($R1)
      LD $F2, 0 ($R2)
      FADD $F4, $F0, $F0
      FADD $F6, $F2, $F2
      SD $F4, 0 ($R1)
      SD $F6, 0 ($R2)
      ADDUI $R1, $R1, 8
      ADDUI $R2, $R2, 8
      BNE $R1, $R3, LOOP
```

EXERCISE 4 – VECTOR PROCESSORS (3 points)

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

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

```
L.V V1, RX          # Load vector from memory address RX into V1
L.V V2, RY          # Load vector from memory address RY into V2
MULVS.D V1, V1, F0  # FP multiply vector V1 to scalar F0
ADDVV.D V2, V2, V1  # FP add vectors V1 and V2
MULVS.D V3, V2, F1  # FP multiply vector V2 to scalar F1
S.V V3, RZ          # Store vector V3 into memory address RZ
```

1) *How many convoys?* _____

2) *Please complete the following scheme:*

0	LV V1	63

3) *How many clock cycles to execute the code?*

_____

4) *In case of NO OPERATION CHAINING, how many convoys and clock cycles would be needed to execute the code?*

_____

Feedback:

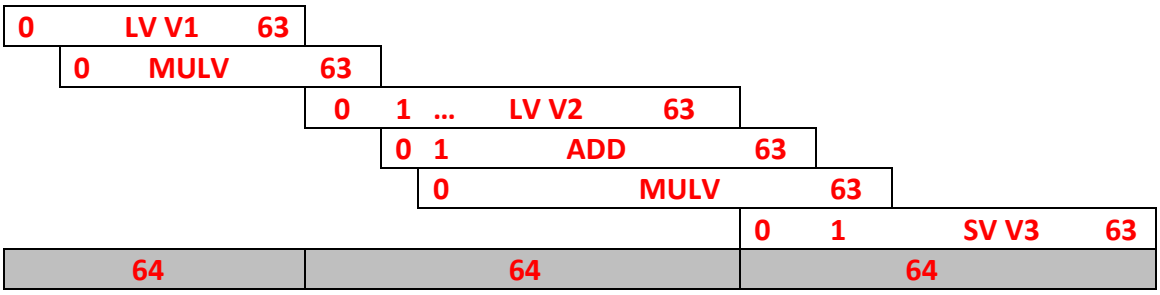
1) *How many convoys?*

   There are 3 convoys as follows:

1) L.V V1, RX;
2) L.V V2, RY
3) S.V V3, RZ

MULVS.D V1, V1, F0
ADDVV.D V2, V2, V1
MULVS.D V3, V2, F1

2) *Please complete the following scheme:*



3) *How many clock cycles to execute the code?*

   3 X 64 = 192 clock cycles

4) *In case of NO OPERATION CHAINING, how many convoys and clock cycles would be needed to execute the code?*

   There are 5 convoys and 5 X 64 = 330 clock cycles as follows:

1) L.V V1, RX;
2) L.V V2, RY
3) ADDVV.D V2, V2, V1
4) MULVS.D V3, V2, F1
5) S.V V3, RZ

MULVS.D V1, V1, F0

QUESTION 1: PROCESSOR PIPELINING (5 points)

1) *What is the difference between a dependency and a hazard?*	
2) *What is the difference between a data dependency and a control dependency?*	
3) *What is the difference between a true data dependency and a name dependency?*	
4) *Which of the WAW, RAW and WAW hazards are generated by true data dependencies and by name dependencies?*	

5) *For each statement of the assembly code, please complete the following table by reporting data dependencies (corresponding to RAW hazards) and name dependencies (WAR/WAW hazards):*

Assembly Code:	Data dependencies (RAW Hazards)	Name dependencies (WAR / WAW hazards)
`Loop: LD F0, 0(R1) # S1`	None	None
`LD F2, 0(R3) # S2`	None	None
`FMUL F0, F0, F2 # S3`	RAW F0 w.S1; RAW F2 w.S2	. . . . .
`FADD F2, F0, F2 # S4`	. . . . .	. . . . .
`SD F2, 0(R3) # S5`	. . . . .	. . . . .
`ADDUI R1, R1, 8 # S6`	. . . . .	. . . . .
`ADDUI R3, R3, 8 # S7`	. . . . .	. . . . .
`BNE R1, R6, Loop # S8`	. . . . .	. . . . .

6) *Use the above Loop code to make an practical example on how* ***Register Renaming*** *can be applied to avoid name dependences:*

```
loop:LD   F0, 0(R1)      # S1
     LD   F2, 0(R3)      # S2
     . . . . . .         # S3
     . . . . . .         # S4
     . . . . . .        .# S5
     . . . . . .       ..# S6
     . . . . . .       ..# S7
     . . . . . .        .# S8
```

Feedback:

Assembly Code:	Data dependencies (RAW Hazards)	Name dependencies (WAR / WAW hazards)
`Loop: LD F0, 0(R1) # S1`	None	None
`LD F2, 0(R3) # S2`	None	None
`FMUL F0, F0, F2 # S3`	RAW F0 w.S1; RAW F2 w.S2	WAW F0 w. S1
`FADD F2, F0, F2 # S4`	RAW F0 w.S3; RAW F2 w.S2	WAW F2 w. S2 WAR F2 w.S3
`SD F2, 0(R3) # S5`	RAW F2 w.S4	None
`ADDUI R1, R1, 8 # S6`	None	WAR R1 w. S1
`ADDUI R3, R3, 8 # S7`	NONE	WAR R3 w. S2,S5
`BNE R1, R6, Loop # S8`	RAW R1 w.S6;	None

Example on how register renaming can be applied to avoid name dependences:

```
loop:LD   F0, 0(R1)      # S1
     LD   F2, 0(R3)      # S2
     FMUL F4, F0, F2     # S3 WAW F0 solved
     FADD F6, F4, F2     # S4 WAW/WAR F2 solved
     SD   F6, 0(R3)      # S5
     ADDUI R1, R1, 8     # S6
     ADDUI R3, R3, 8     # S7
     BNE  R1, R6, loop   # S8
```

QUESTION 2: BRANCH PREDICTION (5 points)

Let's consider the two types of branch prediction techniques: *static and dynamic.*

Answer to the following questions:

	Static Branch Prediction	**Dynamic Branch Prediction**
1) Explain the main concepts for each technique.		
2) What are the main benefits for each technique?		

3) What are the main drawbacks for each technique?		
4) What are the hardware resources needed to implement each technique in a pipelined processor?		

MULTIPLE-CHOICE QUESTIONS: (8 points)

Question 3 (format Multiple Choice – Single answer)

Please consider the following time schedule to be executed on a CPU with dynamic scheduling based on **TOMASULO algorithm** with all cache HITS, a single Common Data Bus and:

- *4 RESERVATION STATIONS **(RS1, RS2, RS3, RS4)** for 2 LOAD/STORE unit **(LDU1, LDU2)** with latency **6***
- *2 RESERVATION STATIONS **(RS5, RS6)** for 2 ALU/BR FUs **(ALU1, ALU2)** with latency **2***

(SINGLE ANSWER)
2 points

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT
I1: lw $t2,VECTA($t6)	1	2	8
I2: lw $t3,VECTB($t6)	2	3	9
I3: lw $t4,VECTC($t6)	3	9	15
I4: addi $t2,$t2,k	4	10	12
I5: sw $t2,VECTA($t6)	5	13	19
I6: add $t4,$t3,$t4	6	16	18
I7: sw $t4,VECTC($t6)	7	19	25

Where is the error in the above schedule?

Answer 1: Line I3 must be: | 3 | 4 | 10 |
Answer 3: Line I4 must be: | 4 | 9 | 11 |
Answer 3: Line I5 must be: | 5 | 6 | 12 |
Answer 4: Line I6 must be: | 6 | 14 | 16 |
Answer 5: Line I7 must be: | 9 | 19 | 25 | **(TRUE)**

Feedback:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
I1: lw $t2,VECTA($t6)	1	2	8	None	RS1	LDU1
I2: lw $t3,VECTB($t6)	2	3	9	None	RS2	LDU2
I3: lw $t4,VECTC($t6)	3	9	15	**STRUCT LDU1**	RS3	LDU1
I4: addi $t2,$t2,k	4	10	12	**RAW $t2 + RF read**	RS5	ALU1
I5: sw $t2,VECTA($t6)	5	13	19	**RAW $t2**	RS4	LDU2
I6: add $t4,$t3,$t4	6	16	18	**RAW $t4**	RS6	ALU2
I7: sw $t4,VECTC($t6)	9	19	25	**STRUCT RS1 + RAW $t4**	RS1	LDU1

Question 4 (format Multiple Choice – Multiple answers)

*How does a MESI write-invalidate write-back protocol manage a **Write Hit** on an Exclusive cache block?*
(MULTIPLE ANSWERS)
2 points

Answer 1: The status of the cache block becomes Modified; **(TRUE)**
Answer 2: The cache block is updated from the memory;
Answer 3: An invalidate is broadcasted on the bus to the other copies of the block;
Answer 4: The cache block is updated from another cache;
Answer 5: The cache block is update with the new value **(TRUE)**

Feedback:
On a Write Hit on an Exclusive cache block means that the block is clean and there are no other copies of the block in other caches. So there is no need to send an invalidate signal on the snooping bus to other caches. We have that:

- *The processor's cache is updated with the new data values.*
- *The status of the cache block becomes Modified*

Question 5 (format Multiple Choice – Single answer)

Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where we consider the block B1 in the directory of N1:

Directory N1 Block B1 | State: Shared | Sharer Bits: 0011 |

*Which are the State and the Sharer Bits of the block **B1** in **N1** after this sequence:*
Read Miss B1 from local cache N1;
Read Miss B1 from local cache N0;
(SINGLE ANSWER)
1 point

Answer 1: Directory N1 Block B1 | State: Shared | Sharer Bits: 1100 |
Answer 2: Directory N1 Block B1 | State: Modified | Sharer Bits: 0100 |
Answer 3: Directory N1 Block B1 | State: Modified | Sharer Bits: 1000 |
Answer 4: Directory N1 Block B1 | State: Shared | Sharer Bits: 1111 | **(TRUE)**

Question 6 (format Multiple Choice – Multiple answers)

What characteristics of the GPU architecture determine its high throughput?
(MULTIPLE ANSWERS)
1 point
Answer 1: The latency of GPU arithmetic units is lower than CPU ALUs;
Answer 2: A GPU has a massive number of parallel processing elements; **(TRUE)**
Answer 3: A GPU does not have sophisticated control units; **(TRUE)**
Answer 4: Switching threads on a GPU is less expensive than on a CPU; **(TRUE**

Question 7 (format True/False)

To make dynamic loop unrolling in the speculative Tomasulo architecture, the register renaming can be done by using the Reorder Buffer entries without the help of the compiler.
(True/False)
1 point

Answer 1: True (**TRUE**)

Answer 2: False

Feedback:
The speculative Tomasulo architecture supports Register Renaming by using ROB entries to eliminate WAR and WAW hazards and enable dynamic loop unrolling.

Question 8 (format Multiple Choice – Single answer)

*In the speculative Tomasulo architecture, what is done at the **Commit stage** when a not-speculative instruction is ready at the head of ROB?*

(SINGLE ANSWER)
1 point

Answer 1: Update destination into RF (or memory in case of store) with ROB result and free ROB entry; **(TRUE)**

Answer 2: Send the instruction result on CDB to update the ROB entry and awaiting Reservation Stations;

Answer 3: Wait for source operand values from CDB to compute the destination result;

Answer 4: Write the instruction result into the inter-stage forwarding registers;

Answer 5: Flush all ROB entries because of the branch misprediction;

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

Politecnico di Milano, 11 July, 2023

Course on Advanced Computer Architectures

Prof. C. Silvano

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

EXERCISE 1 – TOMASULO (5 points)

Let's consider the following 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 RESERVATION STATIONS **(RS1, RS2)** with 2 LOAD/STORE units **(LDU1, LDU2)** with latency **4***
- *2 RESERVATION STATIONS **(RS3, RS4)** with 2 INTEGER ALU/BR units **(ALU1, ALU2)** with latency **2***
- *2 RESERVATION STATIONS **(RS5, RS6)** with 2 FP ALU units **(FPU, FPU2)** with latency **4***

Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`I1: lw $f1,A($r1)`	1	2	6	--	RS1	LDU1
`I2: addi $r2,$r1,4`						
`I3: ld $f2,A($r2)`						
`I4: addi $r3,$r1,8`						
`I5: ld $f3,A($r3)`						
`I6: fadd $f1,$f1,$f2`						
`I7: fadd $f2,$f2,$f3`						
`I8: fmul $f1,$f1,$f2`						

Calculate the CPI:

CPI = _____

Feedback:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type *(*)*	RSi	UNIT
`I1: lw $f1,A($r1)`	1	2	6	--	RS1	LDU1
`I2: addi $r2,$r1,4`	2	3	5	--	RS3	ALU1
`I3: ld $f2,A($r2)`	3	**6**	10	**RAW $r2**	RS2	LDU2
`I4: addi $r3,$r1,8`	4	5	7	--	RS4	ALU2
`I5: ld $f3,A($r3)`	**7**	8	12	**STRUCT RS1** (RAW $r3)	RS1	LDU1
`I6: fadd $f1,$f1,$f2`	8	**11**	15	(RAW $f1), **RAW $f2**	RS5	FPU1
`I7: fadd $f2,$f2,$f3`	9	**13**	17	(RAW $f2) **RAW $f3**	RS6	FPU2
`I8: fmul $f1,$f1,$f2`	**16**	**18**	22	**STRUCT RS5** (RAW $f1) **RAW $f2**	RS5	FPU1

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

(*) Only the hazards that have caused the introduction of some stalls are reported in the table, while other potential hazards are put into brackets.

In Tomasulo, the potential WAW and WAR hazards are already solved by Register Renaming

EXERCISE 2 – VLIW (5 points)

Let's consider the following LOOP code:

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

Given a **4-issue VLIW** machine with **fully pipelined functional units:**

- **1 Memory Units with 3 cycles latency**
- **2 FP ALUs 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	Floating Point Unit 2	Integer Unit
C1	`LD F1, A(R1)`			
C2				
C3				
C4				
C5				
C6				
C7				
C8				
C9				
C10				
C11				
C12				
C13				
C14				
C15				

1. How long is the critical path? _____

2. What performance did you achieve in CPIas?

   _____

3. What performance did you achieve in FP ops per cycles?

   _____

4. How much is the code efficiency?

   _____

5. Assuming to have **1 FP ALUs and 2 INTEGER UNITS** how much is the impact on CPI and code efficiency?

Feedback

	Memory Unit 1	Floating Point Unit 1	Floating Point Unit 2	Integer Unit
C1	`LD F1, A(R1)`			`ADDUI R2,R1,4`
C2	`LD F2, A(R2)`			`ADDUI R3,R1,8`
C3	`LD F3, A(R3)`			
C4				
C5		`FADD F1,F1,F2`		
C6		`FADD F2,F2,F3`		
C7				
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. What performance did you achieve in CPIas?

   CPIas = (# cycles) / IC = 13 / 11 = 1.18

3. What performance did you achieve in FP ops per cycles?

   (# FP ops) / cycles = 3 / 13 = 0.23

4. How much is the code efficiency?

   Code_eff = IC/ (# cycles * # issues) = 11 / (13 * 4) = 11 / 52 = 0.21

5. Assuming to have **1 FP ALUs and 2 INTEGER UNITS** how much is the impact on CPI and code efficiency?

   The impact is null because the 3 FP instructions would have been scheduled in the same way; we could parallelize the 2 ADDUI instructions but the critical path would have been the same.

EXERCISE 3 – MESI PROTOCOL (5 points)

Let's consider the following access patterns on a dual processor system with a direct-mapped, write-back cache with one cache block per processor and a 2 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	Memory at block 0 up to date?	Memory at block 1 up to date?
1	**P1: read block 0**	Exclusive (1)	Exclusive (0)	Yes	Yes
2		Modified (1)	Exclusive (0)	Yes	No
3		Modified (1)	Exclusive (0)	Yes	No
4		Modified (1)	Exclusive (0)	Yes	No
5		Modified (0)	Invalid	No	Yes
6		Modified (0)	Modified (1)	No	No
7		Shared (0)	Shared (0)	Yes	Yes
8		Exclusive (1)	Exclusive (0)	Yes	Yes

Feedback

Cycle	After Operation	P0 cache block state	P1 cache block state	Memory at block 0 up to date?	Memory at block 1 up to date?
1	**P1: read block 0**	Exclusive (1)	Exclusive (0)	Yes	Yes
2	**P0: write block 1**	Modified (1)	Exclusive (0)	Yes	No
3	**P1: read block 0 OR P0 read/write bl.1**	Modified (1)	Exclusive (0)	Yes	No
4	**P1: read block 0 OR P0 read/write bl. 1**	Modified (1)	Exclusive (0)	Yes	No
5	**P0: write block 0**	Modified (0)	Invalid	No	Yes
6	**P1: write block 1**	Modified (0)	Modified (1)	No	No
7	**P1: read block 0**	Shared (0)	Shared (0)	Yes	Yes
8	**P0: read block 1**	Exclusive (1)	Exclusive (0)	Yes	Yes

QUESTION 1: CACHE MEMORIES (5 points)

Describe the main impacts of the 3 different techniques used to design cache memories on the 3 different types of cache misses:

	Compulsory misses	***Capacity misses***	***Conflict misses***
Direct Mapped Cache			
Set Associative Cache			
Fully Associative Cache			

QUESTION 2: DIRECTORY-BASED PROTOCOL (5 points)

Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where: **Directory N0 Block B0 | State: Uncached | Sharer Bits: - - - - |**

Given the following sequence:
Read Miss on B0 from node N1;
Read Miss on B0 from node N2;
Write Hit on B0 from node N1;

Please answer to the following questions:

After the two Read Misses, what is the status of the block B0 in the directory N0?	**Directory N0 Block B0 | State: _____ | Sharer Bits: _____ |**
What are the home node, the local node and the remote node(s) during the Write Hit?	
What is the sequence of messages sent between the nodes during the Write Hit?	
Which is the state of the block B0 in the local cache and in the remote cache at the end of the sequence?	
Which is the state of the block B0 in the home directory at the end of the sequence?	**Directory N0 Block B0 | State: _____ | Sharer Bits: _____ |**

Feedback:

After the two Read Misses, what is the status of the block B0 in the directory N0?	**Directory N0 Block B0 | State: Shared | Sharer Bits: 0110**
What are the home node, the local node and the remote node(s) during the Write Hit?	During the Write Hit we have: **N0** is the home node of B0, **N1** is the local node (requestor and sharer), **N2** is the remote cache (sharer C2);
What are the sequence of messages sent between the nodes during the Write Hit?	The Write Hit on B0 from local node N1 to the home node N0 requires to send an **Invalidate** to the remote node. Then, an **Invalidate** message is sent from the home node N0 to the remote cache (sharer C2); After, the processor P1 (owner) will write in Block B0 of cache C1 becoming **Modified.**
Which is the state of the block B0 in the local cache and in the remote cache at the end of the sequence?	The state of the block B0 in the local cache C1 becomes **Modified.** The state of the block B0 in the remote cache C2 become **Invalid.**
Which is the state of the block B0 in the home directory at the end of the sequence?	The state of the block B0 in the **home directory N0** changes to **Modified** with the requesting node (N1) becoming the **owner**: **Directory N0 Block B0 | State: Modified | Sharer Bits: 0100**

MULTIPLE-CHOICE QUESTIONS: (8 points)

Question 1 (format Multiple Choice – Single answer)

Let's consider a fully associative write-back cache with many cache entries that at cold start is empty and receives the following sequence of memory accesses:

Read Mem[AAAA]
Write Mem[AAAA]
Read Mem[BBBB]
Write Mem[BBBB]
Read Mem[AAAA]
Read Mem[BBBB]

When using a ***"write allocate"*** versus a ***"no-write allocate"*** policy, how many cache hits and misses are there?
(SINGLE ANSWER)
2 points

Answer 1: Write allocate has 4 hits & 2 misses | No-write allocate has 2 hits & 4 misses
Answer 2: Write allocate has 4 hits & 2 misses | No-write allocate has 4 hits & 2 misses **(TRUE)**
Answer 3: Write allocate has 5 hits & 1 miss | No-write allocate has 5 hits & 1 miss
Answer 4: Write allocate has 3 hits & 3 misses | No-write allocate has 2 hits & 4 misses
Answer 5: Write allocate has 2 hits & 4 misses | No-write allocate has 4 hits & 2 misses

Feedback
For the write allocate policy, the first read accesses to Mem[AAAA] and Mem[BBBB] are misses, and the 2 blocks corresponding to [AAAA] and [BBBB] are allocated in cache. The next four accesses are all hits, since the blocks corresponding to [AAAA] and [BBBB] are found in cache. Globally, the write allocate has 2 misses & 4 hits.

For the no-write allocate policy there is the same behavior as in the write allocate policy because the first accesses to Mem[AAAA] and Mem[BBBB] are done on read and the corresponding blocks are allocated in cache. Therefore, also the no-write allocate policy has 2 misses & 4 hits.

Question 2 (format Multiple Choice – Single answer)
Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where we consider the block B1 in the directory of N1:
Directory N1 Block B1 | State: Shared | Sharer Bits: 0011 |

*During a **Write Miss** on B1 from N0, please indicate which are the home node, the local node and the remote node(s):*
(SINGLE ANSWER)
1 point

Answer 1: N1 home node, N2 local node; N3 remote node.
Answer 2: N1 home node; N0 local node; N2 remote node.
Answer 3: N0 home node; N1 local node; N2 and N3 remote nodes.
Answer 4: N1 home node; N0 local node; N2 and N3 remote nodes. **(TRUE)**
Answer 5: N1 home node; N1 local node; N3 remote node.

Question 3 (format Multiple Choice – Multiple answers)

What characteristics make a program suitable to be accelerated on a GPU?
(MULTIPLE ANSWERS)
1 point

Answer 1: Extensive data parallelism; **(TRUE)**
Answer 2: Many if-else constructs;
Answer 3: Frequent communication between parallel tasks;
Answer 4: Few synchronization points; **(TRUE)**
Answer 5: Many mathematical operations on each data; **(TRUE)**

Question 4 (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, RX          # Load vector from memory address RX into V1
L.V V3, RY          # Load vector from memory address RY into V2
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 V3, RZ          # Store vector V3 into memory address RZ
```

How many convoys? How many clock cycles to execute the code?

(SINGLE ANSWER)
2 points

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

Feedback: There are 3 convoys as follows:

1) L.V V1, RX; MULVS.D V1, V1, F0 ADDVV.D V2, V1, V1
2) L.V V3, RY MULVS.D V2, V2, F0 ADDVV.D V3, V2, V3
3) S.V V3, RZ

Question 5 (format Multiple Choice – Multiple answers)

What are the main **disadvantages** of the static scheduling used to support ILP?

(MULTIPLE ANSWER)
1 point

Answer 1: Variable memory latency due to unpredictable cache misses **(TRUE)**
Answer 2: Large amount of parallelism available within a basic block
Answer 3: The need of a cache coherency protocol
Answer 4: Code size increase due to the insertion of NOPs **(TRUE)**
Answer 5: Runtime detection and resolution of data dependencies
Answer 6: Increment of hardware complexity and power dissipation

Question 6 (format Multiple Choice – Single answer)

The reservation stations provided by Tomasulo are effective to:

(SINGLE ANSWER)
1 point

Answer1: *Avoid WAR and WAW hazards by implicit register renaming*

Answer2: *Shorten RAW hazards by providing the results directly from them*

Answer3: *Both of them* **True**

Feedback:
In Tomasulo architecture, registers in instructions are replaced by values or pointers to reservation stations (RS): (1) to enable implicit Register Renaming thus avoiding WAR and WAW hazards by renaming results by using RS numbers instead of RF numbers; (2) to shorten RAW hazards because the operands are given directly by RS without waiting for the write back in the RF (sort of forwarding).

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

Politecnico di Milano, 13 June, 2023

Course on Advanced Computer Architectures

Prof. C. Silvano

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

EXERCISE 1 – TOMASULO (5 points)

Let's consider the following 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 RESERVATION STATIONS **(RS1, RS2)** with 1 LOAD/STORE unit **(LDU1)** with latency **4***
- *2 RESERVATION STATION **(RS3, RS4)** with 1 ALU/BR unit **(ALU1)** with latency **2***

Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`I1: lw $f0,0($r1)`	1	2	6		RS1	LDU1
`I2: lw $f2,4($r1)`	2				RS2	LDU1
`I3: lw $f4,8($r1)`						
`I4: lw $f6,C($r1)`						
`I5:fadd $f8,$f0,$f2`						
`I6:fadd $f10,$f4,$f6`						
`I7:fadd $f10,$f8,$f10`						
`I8: sw $f10,4($r1)`						

Calculate the CPI:

CPI = _____

Feedback:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type *(*)*	RSi	UNIT
`I1: lw $f0,0($r1)`	1	2	6		RS1	LDU1
`I2: lw $f2,4($r1)`	2	**7**	11	**STR. LDU1**	RS2	LDU1
`I3: lw $f4,8($r1)`	**7**	**12**	16	**STR. RS1 + STR. LDU1**	RS1	LDU1
`I4: lw $f6,C($r1)`	**12**	**17**	21	**STR. RS2 + STR.LDU1**	RS2	LDU1
`I5:fadd $f8,$f0,$f2`	13	14	**17**	**STR. CDB/RF WRITE**	RS3	ALU1
`I6:fadd $f10,$f4,$f6`	14	**22**	24	**RAW $f6**	RS4	ALU1
`I7:fadd $f10,$f8,$f10`	**18**	**25**	27	**STRUCT RS3 + RAW Sf10**	RS3	ALU1
`I8: sw $f10,4($r1)`	19	**28**	32	**RAW $f10**	RS1	LDU1

CPI = # clock cycles / IC = 32 / 8 = 4

(*) Only the hazards that have caused the introduction of some stalls are reported in the table. Other potential hazards are already solved.

In Tomasulo, the WAW and WAR hazards are already solved by Register Renaming

EXERCISE 2 – MESI PROTOCOL (5 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	Memory at bl. 0 up to date?	Memory at bl. 1 up to date?
0	**P0: write Bl. 1**	Mod (1)	Invalid	Invalid	Invalid	Yes	No
1	**P2: Read Bl. 0**						
2	**P3: Read Bl. 0**						
3	**P1: Write Bl. 0**						
4	**P2: Write Bl. 1**						
5	**P3: Read Bl. 0**						
6	**P0: Read Bl. 1**						
7	**P2: Write Bl. 0**						

Feedback:

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

EXERCISE 3 – CACHE MEMORIES (5 points)

Let's consider 32-block main memory with a **4-way set associative** 8-block cache based on a ***write allocate*** with ***write-back*** protocol.

The addresses are expressed as:
Memory Address: [0, 1, 2, ……. 31] $_{10}$
Cache Address: [a, b, c, d, e, f, g, h]
and Cache Tags are expressed as binary numbers.

At cold start the cache is empty, then there is the following sequence of memory accesses.
Please complete the following table:

	Type of memory access	Memory Address	HIT/MISS Type	Cache Tag	Cache Address	Set	Write in memory
1	Read	[16] $_{10}$	Cold-start Miss				
2	Write	[16] $_{10}$	Hit				
3	Read	[14] $_{10}$					
4	Read	[07] $_{10}$					
5	Write	[14] $_{10}$					
6	Write	[16] $_{10}$					
7	Read	[12] $_{10}$					
8	Read	[25] $_{10}$					
9	Write	[10] $_{10}$					
10	Read	[02] $_{10}$					
11	Write	[06] $_{10}$					
12	Read	[30] $_{10}$					

Feedback

The 4-way set-associative cache has 8-blocks [a, b, c, d, e, f, g, h] organized in 2 sets, where each set contains 4 blocks as follows: **Set_0: [a , b, c, d]; Set_1: [e, f, g, h]**

Being a set-associative cache, the block replacement policy in each set uses the **LRU algorithm**.

Memory Address Mapping:

Set_0 [a , b, c, d]	Set_1 [e, f, g, h]
0	1
2	3
4	5
…	…
30	31

	Type of memory access	Memory Address	HIT/MISS Type	Cache Tag	Cache Address	Set	Write in memory
1	Read	$[16]_{10}$	Cold-start Miss	$[1000]_2$	a	$[0]_{10}$	No
2	Write	$[16]_{10}$	Hit	$[1000]_2$	a	$[0]_{10}$	No
3	Read	$[14]_{10}$	Cold-start Miss	$[0111]_2$	b	$[0]_{10}$	No
4	Read	$[07]_{10}$	Cold-start Miss	$[0011]_2$	e	$[1]_{10}$	No
5	Write	$[14]_{10}$	Hit	$[0111]_2$	b	$[0]_{10}$	No
6	Write	$[16]_{10}$	Hit	$[1000]_2$	a	$[0]_{10}$	No
7	Read	$[12]_{10}$	Cold-start Miss	$[0110]_2$	c	$[0]_{10}$	No
8	Read	$[25]_{10}$	Cold-start Miss	$[1100]_2$	f	$[1]_{10}$	No
9	Write	$[10]_{10}$	Cold-start Miss	$[0101]_2$	d	$[0]_{10}$	No
10	Read	$[02]_{10}$	Conflict Miss	$[0001]_2$	b	$[0]_{10}$	Yes Wr. in $M[14]_{10}$
11	Write	$[06]_{10}$	Conflict Miss	$[0011]_2$	a	$[0]_{10}$	Yes Wr. in $M[16]_{10}$
12	Read	$[30]_{10}$	Conflict Miss	$[1111]_2$	c	$[0]_{10}$	No

QUESTION 1: MULTITHREADING (5 points)

Let's consider the different multithreading techniques used to manage thread-level parallelism by hardware processors. *Answer to the following questions:*

<table>
<tr><th></th><th>Coarse-grained Multithreading</th><th>Fine-grained Multithreading</th><th>Simultaneous Multithreading</th></tr>
<tr><td>Explain the main concepts for each technique.</td><td>Whenever a thread is stalled (for any reason, such as a dependence, a cache miss, etc.), the processor switches to execute another thread, that uses the processor resources.

The processor must be able to switch threads by hardware context switch.</td><td>In fine-grained MT, the processor switches between threads on each instruction by hardware context switch.

Execution of multiple threads is interleaved in a sort of round-robin fashion, skipping any thread that is stalled at that time.</td><td>Suitable for multiple-issue dynamic scheduled processors (such as 4-issues superscalar processors).

Simultaneously schedule instructions execution from several threads in the same cycle in order to maximize the use of parallel functional units in multiple-issues processors.

SMT exploits both ILP and TLP.</td></tr>
<tr><td>For each technique, make an example with up to 4 threads (T1, T2, T3, T4) on a dual-issue processor</td><td>
<table>
<tr><td>C0</td><td>T1</td><td>T1</td></tr>
<tr><td>C1</td><td>T1</td><td></td></tr>
<tr><td>C2</td><td>T1</td><td>T1</td></tr>
<tr><td>C3</td><td>T1</td><td>T1</td></tr>
<tr><td>C4</td><td>T1</td><td></td></tr>
<tr><td>C5</td><td>T2</td><td>T2</td></tr>
<tr><td>C6</td><td>T2</td><td>T2</td></tr>
<tr><td>C7</td><td>T2</td><td></td></tr>
<tr><td>C8</td><td>T3</td><td></td></tr>
<tr><td>C9</td><td>T3</td><td>T3</td></tr>
</table>
</td><td>
<table>
<tr><td>C0</td><td>T1</td><td>T1</td></tr>
<tr><td>C1</td><td>T2</td><td></td></tr>
<tr><td>C2</td><td>T3</td><td>T3</td></tr>
<tr><td>C3</td><td>T4</td><td>T4</td></tr>
<tr><td>C4</td><td>T1</td><td></td></tr>
<tr><td>C5</td><td>T3</td><td>T3</td></tr>
<tr><td>C6</td><td>T4</td><td>T4</td></tr>
<tr><td>C7</td><td>T1</td><td>T1</td></tr>
<tr><td>C8</td><td>T2</td><td>T2</td></tr>
<tr><td>C9</td><td>T3</td><td></td></tr>
</table>
</td><td>
<table>
<tr><td>C0</td><td>T1</td><td>T2</td></tr>
<tr><td>C1</td><td>T3</td><td>T4</td></tr>
<tr><td>C2</td><td>T1</td><td>T2</td></tr>
<tr><td>C3</td><td>T3</td><td>T3</td></tr>
<tr><td>C4</td><td>T1</td><td>T2</td></tr>
<tr><td>C5</td><td>T1</td><td></td></tr>
<tr><td>C6</td><td>T1</td><td>T2</td></tr>
<tr><td>C7</td><td>T3</td><td>T4</td></tr>
<tr><td>C8</td><td>T1</td><td>T2</td></tr>
<tr><td>C9</td><td>T3</td><td>T4</td></tr>
</table>
</td></tr>
<tr><td>Let's consider a dual-issue SMT processor that can manage up to 4 threads</td><td></td><td></td><td>How much is the ideal CPI?

Ideal CPI = 0.5

How much is the ideal per-thread CPI?

Ideal per-thread CPI = 2</td></tr>
</table>

What are the main benefits for each technique?	Go beyond the limits of ILP to get better performance (this benefit is also valid for fine-grained and SMT) It can hide the penalty of long stalls because, as a example, during an L2 cache miss, it can use the stall cycles to switch to execute another thread. Requires a simpler switching logic with lower overhead than fine-grained and SMT. Requires a less frequent HW context switch than fine-grained and SMT. Less impact on single thread execution time: an individual thread might complete execution earlier than in fine-grained and SMT.	It can hide both short and long stalls because instructions from other threads are executed when one thread stalls. The frequent switching among threads helps to reduce uniformly the penalty due to stalls because a control/data dependency can be solved during the cycles used by another thread. More fairness: All threads can begin/continue their execution quite uniformly, so there is no individual thread "left behind". Fine-grain (but also coarse-grain) can be applied even to a simple single issue processor core that can be combined in a multicore.	Suitable to maximize the use of parallel functional units available in the multiple-issues by different threads. Exploit more parallelism than coarse- and fine-grained MT.
What are the main drawbacks for each technique?	The new thread has a pipeline startup overhead period to empty the pipeline before starting the new thread. This causes a loss of throughput. Less fairness than fine-grain MT: Some threads might not be able to begin/continue execution for some time. If applied to multiple-issue processors, ILP limitations could lead to empty issue slots at each clock cycle (same for fine-grained MT)	Compared to coarse-grain MT, it slows down the execution of individual threads, since a thread will be delayed by another thread even if it is ready to execute. Higher overhead due to more frequent HW context switch.	In large multiple-issue processors (such as 4-issue) requires complex dynamic control logic to keep busy the multiple-slots. Higher overhead and power consumption. Obviously, it is limited by the number of issues available in the processor.

QUESTION 2: BRANCH PREDICTION (5 points)

Let's consider the following code where $R0 has been initialized to 0:

```
INIT:  ADDI $R1, $R0, 0
       ADDI $R2, $R0, 80
       ADDI $R3, $R0, 0
       ADDI $R4, $R0, 40

LOOP1: LD $F0, 0($R1)
       FADD $F2, $F0, $F0
       SD $F2, 0($R1)

       ADDI $R3, $R0, 0
LOOP2: LD $F4, 0($R3)
       FADD $F6, $F4, $F4
       SD $F6, 0($R3)
       ADDI $R3, $R3, 4
BR2:   BNE $R3, $R4, LOOP2

       ADDI $R1, $R1, 4
BR1:   BNE $R1, $R2, LOOP1
```

Let's assume to have a **1 entry 1-bit BHT** as dynamic branch predictor (where the two branch instructions **collide**) initialized as ***Taken.*** *Answer to the following questions:*

	LOOP 1	**LOOP2**	**GLOBALLY**
How many iterations are executed for each loop?	The outer loop LOOP1 is executed **20 times**.	The inner loop LOOP2 is executed **10 times for each iteration of LOOP1.** Globally LOOP2 is executed (10 x 20) = **200 times.**	
How many times the BHT has been used?	The BHT has been used **20 times for BR1** (once per iteration).	The BHT has been used **200 times for BR2** (once per iteration).	Globally, the BHT has been used 220 times (20 + 200). ***=>220 predictions done***

*Assuming the BHT initialized as **Taken,** when there is the **first misprediction?***		At the first iteration of LOOP1, being the predictor initialized as Taken, we have the first misprediction at the last iteration (exit) of the BR2.	
At the first iteration of the outer LOOP1, is there a correct prediction or a misprection?	Exiting from the inner LOOP2 with the NT prediction, this generates a **misprediction** at the first iteration of LOOP1.		
When re-entering in each loop, the value of the prediction bit is Taken or Not Taken?	When re-entering in LOOP1, the predictor is **Taken**.	When re-entering in LOOP2, the predictor is **Taken**.	
How many mispredictions are we going to observe locally in the inner LOOP2?		Considering LOOP2 in isolation, we have **1 misprediction out of 10 predictions** (local misprediction rate 10% for the inner LOOP2).	
How many mispredictions are we going to observe?	Exiting from the inner LOOP2 as Not Taken, this generates a misprediction on the BR1 for 19 iterations (except for the last iteration). ⇨ for the outer LOOP1, we have **19 mispredictions.**	We have 1 misprediction for the BR2 only when exiting from LOOP2, this is for the 20 iterations of the outer LOOP1 => for the inner LOOP2, we have **20 mispredictions.**	Globally there are (19 + 20) = **39 mispredictions.**
How much is the global misprediction rate?			There are 39 mispredictions out of 220 predictions => **39/220 % = 17.72%** global misprediction rate.

MULTIPLE-CHOICE QUESTIONS: (8 points)

Question 3 (format True – False)
A multiple-issue processor has more functional units in parallel than a single thread can use.
(SINGLE ANSWER)
1 point

Answer 1: TRUE **(TRUE)**
Answer 2: FALSE

Question 4 (format Multiple Choice – Single answer)
Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where we consider the block B0 in the directory of N0:
Directory N0 Block B0 | State: Shared | Sharer Bits: 1001 |

*During a **Write Miss** on B0 from N1, please indicate which are the home node, the local node and the remote node(s):*
(SINGLE ANSWER)
1 point

Answer 1: N0 home node, N1 local node; N3 remote node.
Answer 2: N0 home node; N3 local node; N1 remote node.
Answer 3: N0 home node; N1 local node; N0 and N3 remote nodes. **(TRUE)**
Answer 4: N1 home node; N0 local node; N3 remote node.
Answer 5: N1 home node; N1 local node; N3 remote node.

Question 5 (format Multiple Choice – Multiple answer)

Let's consider the following loop:

```
for (i=1; i<=100, i++) {
      X[i] = X[i] + X[i-1];     /*S1*/
      Z[i] = X[i] + Y[i-1]      /*S2*/
      Q[i] = X[i] + Q[i-1]      /*S3*/
}
```

How many loop-carried dependencies are in the code?
(MULTIPLE ANSWERS)
1 point

Answer 1: One in S1 because X[i] depends on X[i-1]; **(TRUE)**
Answer 2: One in S2 because Z[i] depends on Y[i-1];
Answer 3: One in S2 because Z[i] depends on X[i];
Answer 4: One in S3 because Q[i] depends on Q[i-1]; **(TRUE)**

Feedback:
The dependence of Z[i] on Y[i-1] in S2 is not a loop-carried dependence because the vector Y[ ] is never modified in the loop. Also the dependences of Z[i] on X[i] in S2 and Q[i] on X[i] in S3 are not loop-carried dependences.

Question 6 (format Multiple Choice – Multiple answers)

Let's consider the following code sequence executed by a dynamic scheduled processor:

```
I1:  LD.D $FP1, 4($R2)
I2:  ADDI.D $FP0, $FP1, 4
I3:  SD.D $FP0, 0($R1)
I4:  LD.D $FP0, 4($R1)
I5:  ADDI $R1, $R1, 4
```

*Where are the **WAR hazards** in the code?*

(MULTIPLE ANSWERS)
1 point

Answer 1: 1 WAR on $FP1 between I2 and I1;
Answer 2: 1 WAR on $FP0 between I3 and I2;
Answer 3: 1 WAR on $FP0 between I4 and I3; **(TRUE)**
Answer 4: 1 WAR on $R1 between I5 and I3; **(TRUE)**
Answer 5: 1 WAR on $R1 between I5 and I4; **(TRUE)**

Question 7 (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, RX          # Load vector from memory address RX into V1
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
L.V V3, RY          # Load vector from memory address RY into V2
ADDVV.D V3, V2, V3  # FP add vectors V2 and V3
S.V V3, RZ          # Store vector V3 into memory address RZ
```

How many convoys? How many clock cycles to execute the code?

(SINGLE ANSWER)
2 points

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

Feedback: There are 3 convoys:

1) L.V V1, RX; MULVS.D V1, V1, F0 ADDVV.D V2, V1, V1
2) L.V V3, RY MULVS.D V2, V2, F0 ADDVV.D V3, V2, V3
3) S.V V3, RZ

Question 8 (format Multiple Choice – Single answer)

*Let's consider a **Speculative Tomasulo** architecture with 2 load buffers (Load1, Load2), 2 multiply reservation stations (Mult1 and Mult2) and **6-entry ROB** (ROB0, ROB1, …, ROB5).*
*Let's consider the following LOOP when the ROB is **full** after the issue of the instructions of the first iteration (while the first load is executing a cache miss) and the start of the speculative issue of the second iteration.*

```
LOOP: LD $F0, 0 ($R1)
      MULTD $F4, $F0, $F2
      SD $F4, 0 ($R1)
      SUBI $R1, $R1, 8
      BNEZ $R1, LOOP   // branch prediction taken
```

In the Rename Table, what are the pointers used for $F0 and $F4?

(SINGLE ANSWER)
2 points

Answer 1: ROB5 for $F0 and ROB1 for $F4 **(TRUE)**
Answer 2: ROB5 for $F0 and ROB2 for $F4
Answer 3: ROB0 for $F0 and ROB1 for $F4
Answer 4: ROB0 for $F0 and ROB2 for $F4

Feedback:
The 6-entry ROB is FULL:

ROB#	Instruction	Dest.	Ready	
ROB0	LD $F0, 0 ($R1) (1^ iteration exec. cache miss)	$F0	No	HEAD
ROB1	MULTD $F4, $F0, $F2 (1^ iteration issued)	$F4	No	
ROB2	SD $F4, 0 ($R1) (1^ iteration issued)	MEM	No	
ROB3	SUBI $R1, $R1, 8 (1^ iteration issued)	$R1	No	
ROB4	BNEZ $R1, LOOP (1^ branch predicted as taken)		No	
ROB5	LD $F0, 0 ($R1) (2^ iteration issued speculatively)	$F0	No	

Rename Table:

$F0	ROB5
$F2	
$F4	ROB1

Therefore, in the Rename Table, $F0 points to ROB5 because the WAW $F0 is solved while $F4 points to ROB1 because the second MULTD has not yet been issued because the ROB is full. See also L07: Reorder Buffer & Speculation

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

ACA2022_FORM2_29June2022_SILVANO

ACA Course -- Prof. SILVANO

FORM2 is composed of 6 QUESTIONS to get UP TO 12 POINTS

Duration is 24 minutes!

Question 1 (complete table format)
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	Memory at bl. 0 up to date?	Memory at bl. 1 up to date?
0	Initial state	Invalid	Invalid	Invalid	Invalid	Yes	Yes
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	P1: Read Bl. 0						
4	P3: Write Bl. 0						
5	P0: Write Bl. 1						
6	P2: Write Bl. 1						
7	P1: Read Bl. 0						

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

Feedback

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

See MESI protocols on the slides on L13: Multiprocessors.

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

Question 2 (complete table format)
2 points

Given the following assembly code:

```
FOR:LD $F2, A($R1)
    LD $F4, B($R1)
    LD $F6, C($R1)
    FADD $F2,$F2,$F2
    SD $F2,B($R1)
    FADD $F4,$F2,$F4
    SD $F4,C($R1)
    FADD $F6,$F4,$F6
    SD $F6,D($R1)
    ADDUI $R1,$R1,4
    BNE $R1,$R2, FOR
```

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

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

The branch is completed with 1 cycle delay slot (branch solved in ID stage).
In the Register File, it is possible to read and write at the same address at the same clock cycle.
Complete the following table with the list-based scheduling on the 3-issue VLIW machine including the BRANCH DELAY SLOT (NOPs are not written).

	Integer ALU	Mem Unit	FP ADD
C1		`LD F2, A(R1)`	
C2			
C3			
C4			
C5			
C6			
C7			
C8			
C9			
C10			
C11			
C12			
C13			
C14			
C15			

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

Feedback

	Integer ALU	Mem Unit	FP ADDER
C1		LD F2, A(R1)	
C2		LD F4, B(R1)	
C3		LD F6, B(R1)	FADD F2, F0, F2
C4			
C5			
C6		SD F2, B(R1)	FADD F4, F2, F4
C7			
C8			
C9		SD $F4,C($R1)	FADD F6, F4, F6
C10			
C11			
C12	ADD R1,R1,4	SD $F6,D($R1)	
C13			
C14	BNE $R1,$R2, FOR		
C15	Br. delay slot		

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

Question 3 (format open text)
2 points

1. How long is the critical path? _____
2. What performance did you achieve in CPIas?

   _____

3. How much is the code efficiency?

   _____

4. Let's assume the same code to be rescheduled on a **4-issue VLIW** with the same characteristics as before but with **2 Memory Units** instead of 1 Memory Unit:
   - How much has the critical path been improved?

     _____

   - Has the code efficiency been improved?

     _____

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

Feedback

1. How long is the critical path? **15 cycles**
2. What performance did you achieve in CPIas?

   CPIas = (# cycles) / IC = 15 / 11 = 1,14
3. How much is the code efficiency?

   Code_eff = IC/ (# cycles * # issues) = 11 / (15 * 3) = 0,24

Given a **4-issue VLIW** machine with with the same characteristics as before but with **2 Memory Units** instead of 1 Memory Unit we have had the following schedule:

	Integer ALU	Mem Unit	Mem Unit	FP ADDER
C1		`LD F2, A(R1)`	`LD F4, B(R1)`	
C2		`LD F6, B(R1)`		
C3				`FADD F2, F0, F2`
C4				
C5				
C6		`SD F2, B(R1)`		`FADD F4, F2, F4`
C7				
C8				
C9		`SD $F4,C($R1)`		`FADD F6, F4, F6`
C10				
C11				
C12	`ADD R1,R1,4`	`SD $F6,D($R1)`		
C13				
C14	`BNE $R1,$R2, FOR`			
C15	`Br. delay slot`			

- How much has the critical path been improved?

  The critical path is the same as before
- Has the code efficiency been improved?

  No, it hasn't: the Code efficient is worse than before:

  Code_eff = IC/ (# cycles * # issues) = 11 / (15 * 4) = 0,18

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

Question 4 (format complete table and open text)
2 points

Let's consider the following assembly code:

```
I1: lw $f0,0($r1)
I2: fadd $f0,$f0,$f0
I3: sw $f0,0($r1)
I4: lw $f1,4($r1)
I5: lw $f2,8($r1)
I6: fadd $f1,$f1,$f2
I7: sw $f1,4($r1)
I8: sw $f2,C($r1)
```

to be executed on a CPU with dynamic scheduling based on **TOMASULO algorithm** with all cache HITS, a single Common Data Bus and:

- *2 RESERVATION STATIONS **(RS1, RS2)** for the LOAD/STORE unit **(LDU1)** with latency 4*
- *2 RESERVATION STATION **(RS3, RS4)** for the ALU/BR FUs **(ALU1)** with latency 2*

Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`I1: lw $f0,0($r1)`	1	2	6	None	RS1	LDU1
`I2: fadd $f0,$f0,$f0`	2	7	9	RAW $f0	RS3	ALU1
`I3: sw $f0,0($r1)`						
`I4: lw $f1,4($r1)`						
`I5: lw $f2,8($r1)`						
`I6: fadd $f1,$f1,$f2`						
`I7: sw $f1,4($r1)`						
`I8: sw $f2,C($r1)`						

Calculate the CPI:

CPI = _____

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

Feedback:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`I1: lw $f0,0($r1)`	1	2	6		RS1	LDU1
`I2: fadd $f0,$f0,$f0`	2	**7**	9	**RAW $f0**	RS3	ALU1
`I3: sw $f0,0($r1)`	3	**10**	14	**RAW $f0** (STRUCT LDU1)	RS2	LDU1
`I4: lw $f1,4($r1)`	**7**	**15**	19	**STRUCT RS1 + STRUCT LDU1**	RS1	LDU1
`I5: lw $f2,8($r1)`	**15**	**20**	24	**STRUCT RS2 + STRUCT LDU1**	RS2	LDU1
`I6: fadd $f1,$f1,$f2`	16	**25**	27	**RAW $f2** (RAW $f1)	RS3	ALU1
`I7: sw $f1,4($r1)`	**20**	**28**	32	**STRUCT RS1** + (STRUCT LDU1) + **RAW $f1**	RS1	LDU1
`I8: sw $f2,C($r1)`	**25**	**33**	37	**STRUCT RS2 + STRUCT LDU1** + (RAW $f2)	RS2	LDU1

CPI = # clock cycles / IC = 37 / 8 = 4.625

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

Question 5 (format open text)
2 points

Let's consider the following assembly code:

```
LOOP: LD $F0, 0 ($R1)
      LD $F2, 0 ($R2)
      ADDD $F4, $F0, $F0
      ADDD $F6, $F2, $F2
      SD $F4, 0 ($R1)
      SD $F6, 0 ($R2)
      ADDDUI $R1, $R1, 8
      ADDDUI $R2, $R2, 8
      BNE $R1, $R3, LOOP
```

Write a software-pipelined version of this loop by omitting the start-up and finish up code

Open text answer (max 15 rows)

Feedback:

```
LOOP: SD $F4, 0 ($R1)        /* store from iteration [i] corr. to index 0 */
       SD $F6, 0 ($R2)       /* store from iteration [i] corr. to index 0 */
      ADDD $F4, $F0, $F0     /* add from iteration [i+1] corr. to index 8 */
      ADDD $F6, $F2, $F2     /* add from iteration [i+1] corr. to index 8 */
      LD $F0, 16 ($R1)       /* load from iteration [i+2] corr. to index 16 */
      LD $F2, 16 ($R2)       /* load from iteration [i+2] corr. to index 16 */
      ADDDUI $R1, $R1, 8
      ADDDUI $R2, $R2, 8
      BNE $R1, $R3, LOOP
```

See slides on L09 VLIW Code Scheduling

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 29/06/2022

Question 6 (complete table format)
2 points

Let's consider the following assembly code:

Assembly Code:

```
LOOP: LD $F0, 0 ($R1)
      LD $F2, 0 ($R2)
      ADDD $F4, $F0, $F0
      ADDD $F6, $F2, $F2
      SD $F4, 0 ($R1)
      SD $F6, 0 ($R2)
      ADDDUI $R1, $R1, 8
      ADDDUI $R2, $R2, 8
      BNE $R1, $R3, LOOP
```

For each statement of the assembly code, the following table reports data dependencies (corresponding to RAW hazards) and name dependencies (corresponding to WAR/WAW hazards):

Please complete the following table with dependencies (hazards):

Assembly Code:		Data dependencies (RAW Hazards)	Name depend. (WAR / WAW hazards)
LOOP: LD $F0, 0 ($R1)	#S1	None	None
LD $F2, 0 ($R2)	#S2	None	None
ADDD $F4, $F0, $F0	#S3	RAW $F0 w. S1	None
ADDD $F6, $F2, $F2	#S4	RAW $F2 w. S2	None
SD $F4, 0 ($R1)	#S5		
SD $F6, 0 ($R2)	#S6		
ADDDUI $R1, $R1, 8	#S7		
ADDDUI $R2, $R2, 8	#S8		
BNE $R1, $R3, LOOP	#S9		

Feedback

Assembly Code:		Data dependencies (RAW Hazards)	Name dependencies (WAR / WAW hazards)
LOOP: LD $F0, 0 ($R1)	#S1	None	None
LD $F2, 0 ($R2)	#S2	None	None
ADDD $F4, $F0, $F0	#S3	RAW $F0 w. S1	None
ADDD $F6, $F2, $F2	#S4	RAW $F2 w. S2	None
SD $F4, 0 ($R1)	#S5	RAW $F4 w. S3	None
SD $F6, 0 ($R2)	#S6	RAW $F6 w. S4	None
ADDDUI $R1, $R1, 8	#S7	None	WAR $R1 w.S5, S1
ADDDUI $R2, $R2, 8	#S8	None	WAR $R2 w.S6, S2
BNE $R1, $R3, LOOP	#S9	RAW $R1 w. S7	None

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

ACA2022_FORM2_21July2022_SILVANO

ACA Course -- Prof. SILVANO

FORM2 is composed of 6 QUESTIONS to get UP TO 12 POINTS

Duration is 24 minutes!

Question 1 (complete table format)
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	Memory at bl. 0 up to date?	Memory at bl. 1 up to date?
0	Initial state	Invalid	Invalid	Invalid	Invalid	Yes	Yes
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						
4	P1: Write Bl. 0						
5	P0: Write Bl. 1						
6	P3: Read Bl. 1						
7	P2: Read Bl. 0						

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Feedback

Cycle	After Operation	P0 cache block state	P1 cache block state	P2 cache block state	P3 cache block state	Memory at bl. 0 up to date?	Memory at bl. 1 up to date?
0	Initial state	Invalid	Invalid	Invalid	Invalid	Yes	Yes
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
7	P2: Read Bl. 0	Shared (1)	Shared (0)	Shared (0)	Shared (1)	Yes	Yes

See MESI protocols on the slides on L13: Multiprocessors.

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Question 2 (complete table format)
2 points

Let's consider the following assembly code:

```
FOR:LD $F2, A($R1)
    LD $F4, B($R1)
    FADD $F2, $F2, $F2
    FADD $F4, $F4, $F4
    LD $F6, C($R1)
    LD $F8, D($R1)
    FADD $F6, $F2, $F6
    FADD $F8, $F4, $F8
    SD $F2, A($R1)
    SD $F4, B($R1)
    SD $F6, C($R1)
    SD $F8, D($R1)
    ADDUI $R1, $R1, 4
    BNE $R1, $R2, FOR
```

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

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

The branch is completed with 1 cycle delay slot (branch solved in ID stage).
In the Register File, it is possible to read and write at the same address at the same clock cycle.
Complete the following table with the list-based scheduling on the 3-issue VLIW machine including the BRANCH DELAY SLOT (NOPs are not written).

	Integer ALU	Mem Unit	FP ADD
C1		`LD F2, A(R1)`	
C2			
C3			
C4			
C5			
C6			
C7			
C8			
C9			
C10			
C11			
C12			
C13			
C14			
C15			

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Feedback

	Integer ALU	Mem Unit	FP ADDER
C1		LD F2, A(R1)	
C2		LD F4, B(R1)	
C3		LD F6, C(R1)	FADD F2, F2, F2
C4		LD F8, D(R1)	FADD F4, F4, F4
C5			
C6		SD F2, A(R1)	FADD F6, F2, F6
C7		SD $F4, B($R1)	FADD F8, F4, F8
C8			
C9		SD $F6, C($R1)	
C10	ADD R1, R1, 4	SD $F8, D($R1)	
C11			
C12	BNE $R1, $R2, FOR		
C13	Br. delay slot		

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Question 3 (format open text)
2 points

1. How long is the critical path? _____
2. What performance did you achieve in CPIas?

   _____

3. How much is the code efficiency?

   _____

4. Let's assume the same code to be rescheduled on a **4-issue VLIW** with the same characteristics as before but with **2 Memory Units** instead of 1 Memory Unit:
   - How much has the critical path been improved?

     _____

   - Has the code efficiency been improved?

     _____

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Feedback

1. How long is the critical path?

   13 cycles

2. What performance did you achieve in CPIas?

   CPIas = (# cycles) / IC = 13 / 14 = 0.93

3. How much is the code efficiency?

   Code_eff = IC/ (# cycles * # issues) = 14 / (13 * 3) = 0.36

Given a **4-issue VLIW** machine with with the same characteristics as before but with **2 Memory Units** instead of 1 Memory Unit we have had the following schedule:

	Integer ALU	Mem Unit	Mem Unit	FP ADDER
C1		`LD F2, A(R1)`	`LD F4, B(R1)`	
C2		`LD F6, C(R1)`	`LD F8, D(R1)`	
C3				`FADD F2, F2, F2`
C4				`FADD F4, F4, F4`
C5				
C6		`SD F2, A(R1)`		`FADD F6, F2, F6`
C7		`SD $F4,B($R1)`		`FADD F8, F4, F8`
C8				
C9		`SD $F6,C($R1)`		
C10	`ADD R1,R1,4`	`SD $F8,D($R1)`		
C11				
C12	`BNE $R1,$R2, FOR`			
C13	`Br. delay slot`			

- How much has the critical path been improved?

  The critical path is the same as before (13 cycles)

- Has the code efficiency been improved?

  No, it hasn't: the code efficient is worse than before:

  Code_eff = IC/ (# cycles * # issues) = 14 / (13 * 4) = 0.27

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Question 4 (format complete table)
2 points

Let's consider 32-block main memory with a direct-mapped 8-block cache based on a ***write allocate*** with ***write-through*** protocol.

The addresses are expressed as decimal numbers:
Memory Address: $[0, 1, 2, \ldots\ldots\ldots 31]_{10}$
Cache Address: $[0, 1, 2, \ldots 7]_{10}$
and Cache Tags are expressed as binary numbers.

At cold start the cache is empty, then there is the following sequence of memory accesses.
Please complete the following table:

	Type of memory access	Memory Address	HIT/MISS Type	Cache Tag	Cache Address	Write in memory
1	Read	$[24]_{10}$	Cold-start Miss	$[11]_2$	$[0]_{10}$	No
2	Write	$[24]_{10}$	Hit	$[11]_2$	$[0]_{10}$	Yes, Wr. in M$[24]_{10}$
3	Read	$[24]_{10}$	Hit	$[11]_2$	$[0]_{10}$	No
4	Read	$[10]_{10}$				
5	Read	$[16]_{10}$				
6	Write	$[10]_{10}$				
7	Write	$[24]_{10}$				
8	Read	$[12]_{10}$				
9	Read	$[21]_{10}$				
10	Write	$[18]_{10}$				

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Feedback

	Type of memory access	Memory Address	HIT/MISS Type	Cache Tag	Cache Address	Write in memory
1	Read	$[24]_{10}$	Cold-start Miss	$[11]_{2}$	$[0]_{10}$	No
2	Write	$[24]_{10}$	Hit	$[11]_{2}$	$[0]_{10}$	Yes, Wr. in M$[24]_{10}$
3	Read	$[24]_{10}$	Hit	$[11]_{2}$	$[0]_{10}$	No
4	Read	$[10]_{10}$	Cold-start Miss	$[01]_{2}$	$[2]_{10}$	No
5	Read	$[16]_{10}$	Conflict Miss	$[10]_{2}$	$[0]_{10}$	No
6	Write	$[10]_{10}$	Hit	$[01]_{2}$	$[2]_{10}$	Yes, Wr. in M$[10]_{10}$
7	Write	$[24]_{10}$	Conflict Miss	$[11]_{2}$	$[0]_{10}$	Yes, Wr. in M$[24]_{10}$
8	Read	$[12]_{10}$	Cold-start Miss	$[01]_{2}$	$[4]_{10}$	No
9	Read	$[21]_{10}$	Cold-start Miss	$[10]_{2}$	$[5]_{10}$	No
10	Write	$[18]_{10}$	Conflict Miss	$[10]_{2}$	$[2]_{10}$	Yes, W in M$[18]_{10}$

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Question 5 (format open text)
2 points

Let's consider the following assembly code:

```
FOR:LD $F2, A($R1)
    LD $F4, B($R1)
    FADD $F2, $F2, $F2
    FADD $F4, $F4, $F4
    LD $F6, C($R1)
    LD $F8, D($R1)
    FADD $F6, $F2, $F6
    FADD $F8, $F4, $F8
    SD $F2, A($R1)
    SD $F4, B($R1)
    SD $F6, C($R1)
    SD $F8, D($R1)
    ADDUI $R1, $R1, 4
    BNE $R1, $R2, FOR
```

Write a software-pipelined version of this loop without the startup and finish up code.

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Feedback:

```
FOR:SD $F2, A($R1)          /* store from iteration [i] corr. to index 0 */
    SD $F4, B($R1)          /* store from iteration [i] corr. to index 0 */
    SD $F6, C($R1)          /* store from iteration [i] corr. to index 0 */
    SD $F8, D($R1)          /* store from iteration [i] corr. to index 0 */
    FADD $F2, $F2, $F2      /* add from iteration [i+1] corr. to index 4 */
    FADD $F4, $F4, $F4      /* add from iteration [i+1] corr. to index 4 */
    FADD $F6, $F2, $F6      /* add from iteration [i+1] corr. to index 4 */
    FADD $F8, $F4, $F8      /* add from iteration [i+1] corr. to index 4 */
    LD $F2, A+8($R1)          /* load from iteration [i+2] corr. to index 8 */
    LD $F4, B+8($R1)          /* load from iteration [i+2] corr. to index 8 */
    LD $F6, C+8($R1)          /* load from iteration [i+2] corr. to index 8 */
    LD $F8, D+8($R1)          /* load from iteration [i+2] corr. to index 8 */
    ADDUI $R1, $R1, 4
    BNE $R1, $R2, FOR
```

See slides on L09 VLIW Code Scheduling

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

Question 6 (complete table format)
2 points

Let's consider to accelerate a processor architecture by adding a vector mode to it: When executing in vector mode, the processor is 20 times faster than the normal execution mode of.
We indicate as *percentage of vectorization* the percentage of computation time that could be spent using the vector mode.

1. What *percentage of vectorization* F_v is needed to achieve an overall speedup of 2?

2. Given the following percentages of vectorization for a target program, what are the corresponding overall speedups that could be achieved?

F_v	$Speedup_{overall}$
0%	
10%	
25%	
50%	
75%	
90%	
100%	

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 21/07/2022

1. What *percentage of vectorization* F_v is needed to achieve an overall speedup of 2?

We need to apply the Amdahl's law:

$$Speedup_{overall} = 1 \ / \ [(1-F_v) + F_v / S_v]$$

Therefore in our case we have:

$2 = 1 / [(1-F_v) + F_v / 20]$

$\Rightarrow 2 = 20 / [20(1-F_v) + F_v]$

$\Rightarrow 40 (1-F_v) + 2F_v = 20$

$\Rightarrow F_v = 20 / 38 \Rightarrow F_v = 52.6\%$

3. Given the following percentages of vectorization for a target program, what are the corresponding overall speedups that could be achieved?

For each value of F_v, we need to apply the Amdahl's law where $S_v = 20$:

$$Speedup_{overall} = 1 / [(1-F_v) + F_v / 20]$$

F_v	$Speedup_{overall}$
0%	1
10%	1.10
25%	1.31
50%	1.90
75%	3.48
90%	6.90
100%	20

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

ACA2022_FORM2_8Sept2022_SILVANO

ACA Course -- Prof. SILVANO

FORM2 is composed of 5 QUESTIONS to get UP TO 12 POINTS

Duration is 24 minutes!

Question 1 (complete table format)
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	Memory at bl. 0 up to date?	Memory at bl. 1 up to date?
0	Initial state	Invalid	Invalid	Invalid	Invalid	Yes	Yes
1	P0: Read Bl. 1	Excl (1)	Invalid	Invalid	Invalid	Yes	Yes
2	P1: Read Bl. 0	Excl (1)	Excl (0)	Invalid	Invalid	Yes	Yes
3	P2: Read Bl. 0						
4	P0: Write Bl. 1						
5	P3: Write Bl. 0						
6	P2: Write Bl. 1						
7	P1: Read Bl. 1						
8	P0: Read Bl. 0						

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

Feedback

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

See MESI protocols on the slides on L13: Multiprocessors.

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

Question 2 (complete table and open text format)
4 points

Let's consider the following assembly code:

```
FOR1:beq $t6,$t7, END
     lw $t2,VECTA($t6)
     lw $t3,VECTB($t6)
     lw $t4,VECTC($t6)
     addi $t2,$t2,k
     sw $t2,VECTA($t6)
     add $t4,$t3,$t4
     sw $t4,VECTC($t6)
     addi $t6,$t6,4
     j FOR1
```

to be executed on a CPU with dynamic scheduling based on **TOMASULO algorithm** with all cache HITS, a single Common Data Bus and:

- *4 RESERVATION STATIONS **(RS1, RS2, RS3, RS4)** for two LOAD/STORE unit **(LDU1, LDU2)** with latency **6***
- *2 RESERVATION STATION **(RS5, RS6)** for two ALU/BR FUs **(ALU1, ALU2)** with latency **2***
- Static Branch Prediction **BTFNT (BACKWARD TAKEN FORWARD NOT TAKEN)** with Branch Target Buffer

Please complete the following table:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`FOR1:beq $t6,$t7, END`	**1**	**2**	**4**	None	RS5	ALU1
`lw $t2,VECTA($t6)`	**2**	**3**	**9**	None (Control solved by BP)	RS1	LDU1
`lw $t3,VECTB($t6)`	**3**	**4**	**10**	None	RS2	LDU2
`lw $t4,VECTC($t6)`						
`addi $t2,$t2,k`						
`sw $t2,VECTA($t6)`						
`add $t4,$t3,$t4`						
`sw $t4,VECTC($t6)`						
`addi $t6,$t6,4`						
`j FOR1`						

*Calculate the **CPI** and the **IPC:***

CPI = _____

IPC= _____

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

Feedback:

INSTRUCTION	ISSUE	START EXEC	WRITE RESULT	Hazards Type	RSi	UNIT
`FOR1:beq $t6,$t7, END`	**1**	**2**	**4**	None	RS5	ALU1
`lw $t2,VECTA($t6)`	**2**	**3**	**9**	None(Control solved by BP)	RS1	LDU1
`lw $t3,VECTB($t6)`	**3**	**4**	**10**	None	RS2	LDU2
`lw $t4,VECTC($t6)`	4	10	16	**STRUCT LDU1**	RS3	LDU1
`addi $t2,$t2,k`	5	11	13	**RAW $t2 + RF read**	RS5	ALU1
`sw $t2,VECTA($t6)`	6	14	20	**RAW $t2**	RS4	LDU2
`add $t4,$t3,$t4`	7	17	19	**RAW $t4**	RS6	ALU2
`sw $t4,VECTC($t6)`	10	20	26	**STRUCT RS1 + RAW $t4**	RS1	LDU1
`addi $t6,$t6,4`	14	15	17	**STRUCT RS6**	RS5	ALU1
`j FOR1`	18	19	21	**STRUCT RS6**	RS5	ALU2

CPI = # clock cycles / IC = 26 / 10 = 2.6

IPC = 1/CPI = 1/ 2.6 = 0.38

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

Question 3 (open text format)
2 points

Let's consider the following *software-pipelined version* of a given loop including start-up and finish-up codes, assuming registers $R1 and $R2 have been initialized to 0 and 40 respectively:

```
START-UP:  LD $F0, 0 ($R1)
           ADDI.D $F0, $F0, 8
           ADD.D $F4, $F0, $F2
           LD $F0, 4 ($R1)

SW-LOOP:   SD $F4, 0 ($R1)
           ADDI.D $F0, $F0, 8
           ADD.D $F4, $F0, $F2
           LD $F0, 8 ($R1)
           ADDI.U $R1, $R1, 4
           BNE $R1, $R2, SW-LOOP

FINISH-UP: SD $F4, 0 ($R1)
           ADDI.D $F0, $F0, 8
           ADD.D $F4, $F0, $F2
           SD $F4, 4 ($R1)
```

1) How many iterations of the SW-LOOP are executed?

_____

2) Write the original de-pipelined version of the loop:

_____

_____

_____

_____

_____

_____

_____

_____

3) How many iterations of the original de-pipelined loop are executed?

_____

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

Feedback

*1) **10 iterations** of the SW-LOOP are executed (40/4)*

*2) The original de-pipelined version of the loop is the following where the registers **$R1** and **$R2** have been respectively initialized to **0** and **48:***

```
LOOP: LD $F0, 0 ($R1)
      ADDI.D $F0, $F0, 8
      ADD.D $F4, $F0, $F2
      SD $F4, 0 ($R1)
      ADDI.U $R1, $R1, 4
      BNE $R1, $R2, LOOP
```

*3) **12 iterations** of the original de-pipelined LOOP are executed.*

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

Question 4 (open text format)
2 points

Let us consider a 1 GHz processor architecture with 3-level caches and the main memory:

Hit Time $_{L1}$ = 1 ns; Hit Rate $_{L1}$ = 96%
Hit Time $_{L2}$ = 5 ns; Hit Rate $_{L2}$ = 92%
Hit time $_{L3}$ = 10 ns; Hit Rate $_{L3}$ = 90%, Miss Penalty $_{L3}$ = 20 ns.

1. *Write the Global Miss Rate for the last level cache:*

2. *Write the Average Memory Access Time:*

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

Feedback

1. *How much is the Global Miss Rate for Last Level Cache?*

Miss Rate $_{L1\ L2\ L3}$ = Miss Rate$_{L1}$ x Miss Rate$_{L2}$ x Miss Rate$_{L3}$ = 0.04 x 0.08 x 0.1 → 0.032%

2. *How much is the AMAT?*

AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$ =

Hit Time$_{L1}$ + Miss Rate$_{L1}$ x (Hit Time$_{L2}$ + Miss Rate$_{L2}$ x (Hit Time$_{L3}$ + Miss Rate$_{L3}$ x Miss Penalty$_{L3}$)) =

Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Hit Time$_{L2}$ + Miss Rate $_{L1\ L2}$ x Hit Time$_{L3}$ + Miss Rate $_{L1\ L2\ L3}$x Miss Penalty$_{L3}$ =

1 ns + 0.04 x 5ns + 0.0032 x 10ns + 0.00032 x 20 ns = 1.2384 ns

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

Question 5 (complete table format)
2 points

Let's consider 32-block main memory with a direct-mapped 8-block cache based on a ***write allocate*** with ***write-through*** protocol.

The addresses are expressed as decimal numbers:
Memory Address: $[0, 1, 2, \ldots\ldots\ldots 31]_{10}$
Cache Address: $[0, 1, 2, \ldots 7]_{10}$
and Cache Tags are expressed as binary numbers.

At cold start the cache is empty, then there is the following sequence of memory accesses.

Please complete the following table:

	Type of memory access	Memory Address	HIT/MISS Type	Cache Tag	Cache Address	Write in memory
1	Read	$[16]_{10}$	Cold-start Miss	$[10]_{2}$	$[0]_{10}$	No
2	Write	$[16]_{10}$				
3	Read	$[16]_{10}$				
4	Read	$[10]_{10}$				
5	Write	$[8]_{10}$				
6	Write	$[10]_{10}$				
7	Write	$[16]_{10}$				
8	Read	$[26]_{10}$				
9	Read	$[23]_{10}$				
10	Write	$[24]_{10}$				

NAME SURNAME PERSONAL CODE	ACA2022_EXAM_FORM2 Data: 08/09/2022

Feedback

	Type of memory access	Memory Address	HIT/MISS Type	Cache Tag	Cache Address	Write in memory
1	Read	$[16]_{10}$	Cold-start Miss	$[10]_{2}$	$[0]_{10}$	No
2	Write	$[16]_{10}$	Hit	$[10]_{2}$	$[0]_{10}$	Yes, Wr. in M$[16]_{10}$
3	Read	$[16]_{10}$	Hit	$[10]_{2}$	$[0]_{10}$	No
4	Read	$[10]_{10}$	Cold-start Miss	$[01]_{2}$	$[2]_{10}$	No
5	Write	$[8]_{10}$	Conflict Miss	$[01]_{2}$	$[0]_{10}$	Yes, Wr. in M$[8]_{10}$
6	Write	$[10]_{10}$	Hit	$[01]_{2}$	$[2]_{10}$	Yes, Wr. in M$[10]_{10}$
7	Write	$[16]_{10}$	Conflict Miss	$[10]_{2}$	$[0]_{10}$	Yes, Wr. in M$[16]_{10}$
8	Read	$[26]_{10}$	Conflict Miss	$[11]_{2}$	$[2]_{10}$	No
9	Read	$[23]_{10}$	Cold-start Miss	$[10]_{2}$	$[7]_{10}$	No
10	Write	$[24]_{10}$	Conflict Miss	$[11]_{2}$	$[0]_{10}$	Yes, W in M$[24]_{10}$

Prof. Cristina Silvano	ACA2022_EXAM_FORM1 Data: 29/06/2022

ACA2022_FORM1_29June2022_SILVANO

ACA Course -- Prof. SILVANO

FORM1 is composed of 15 QUESTIONS to get UP TO 21 POINTS

Duration is 35 minutes!

IMPORTANT: What to do before leaving MS FORM1:

1. **Please click on:** "Inviami una conferma tramite posta elettronica delle risposte" (Send me a confirmation by email of the answers). This is **fundamental** to get an email with subject: **"My responses".** Open the email and click on the link to **VIEW YOUR RESPONSES.**

2. **Please click on SUBMIT** (Click on **INVIA**) and you will get the message: **"Your answer has been sent".** Points will be assigned **manually** to **FORM1**. Therefore, the view of your results is **not** immediate: it will be enabled by prof. Silvano **after** results publication phase.

Prof. Cristina Silvano	ACA2022_EXAM_FORM1 Data: 29/06/2022

Question 1 (format Multiple Choice – Single answer)

Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where we consider the block B1 in the directory of N1:
Directory N1 Block B1 | State: Shared | Sharer Bits: 1001
Which one was the request message sent for the block B1 to become:
Directory N1 Block B1 | State: Modified | Sharer Bits: 1000

(SINGLE ANSWER)
1 point

Answer 1: Write Hit B1 sent from N0 to N1 **(TRUE)**
Answer 2: Write Miss B1 sent from N0 to N1
Answer 3: Read Hit B1 sent from N0 to N1
Answer 4: Read Miss B1 sent from N1 to N0
Answer 5: Write Miss B1 sent from N1 to N0

Question 2 (format Multiple Choice – Single answer)
Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where we consider the block B0 in the directory of N0:
Directory N0 Block B0 | State: Shared | Sharer Bits: 0100 |

During a Read Miss on B0 from N0, please indicate which are the home node, the local node and the remote node(s):

(SINGLE ANSWER)
1 point

Answer 1: N0 home node, N1 local node; N3 remote node.
Answer 2: N0 home node; N0 local node; N1 remote node. **(TRUE)**
Answer 3: N1 home node; N1 local node; N1 remote node.
Answer 4: N1 home node; N0 local node; N1 remote node.

Question 3 (format Multiple Choice – Single answer)

Let's consider the following code sequence executed by the 5 stage MIPS optimized pipeline with all forwarding paths:

```
I1:         LW $s1, 0($r1)
I2:         ADD $s1, $s1, $S1
I3:         ADDI $s2, $s1, C1
I4:         SW $s2, 8($r1)
```

Which one of the following answers is true?
(SINGLE ANSWER)
1 point

Answer 1: No stalls needed because all hazards are solved by forwarding paths.
Answer 2: One stall in ID stage of I2 & MEM/EX forwarding path for RAW $s1. **(TRUE)**
Answer 3: One stall in ID stage of I4 & EX/MEM forwarding path for $s2.

Question 4 (format Multiple Choice – Single answer)

Let's consider a 5-issue processor that can manage up to 5 simultaneous threads.

What are the values of the ideal CPI and the ideal per-thread CPI?

(SINGLE ANSWER)
1 point

Answer 1: Ideal CPI = 5 & Ideal per-thread CPI = 1
Answer 2: Ideal CPI = 1 & Ideal per-thread CPI =0.2
Answer 3: Ideal CPI = 0.20 & Ideal per-thread CPI = 1 **(TRUE)**
Answer 4: Ideal CPI = 0.25 & Ideal per-thread CPI = 5
Answer 5: Ideal CPI = 5 & Ideal per-thread CPI = 0.20

Question 5 (format Single Answers)

In which type of cache coherence protocol we can say that:
In a shared memory multiprocessor, the delay between writing a cache block by one processor and reading the new value by another processor requires less time.

(SINGLE ANSWER)
1 point

Answer 1: Write update protocol **TRUE**
Answer 2: Write invalidate protocol

Question 6 (format Single Answers)

In a shared memory multiprocessor, multiple writes to the same data with no intervening reads require multiple data broadcast in which type of cache coherence protocol?

(SINGLE ANSWER)
1 point

Answer 1: Write update protocol **TRUE**
Answer 2: Write invalidate protocol

Question 7 (format Multiple Choice – Single answer)

Let's consider the following 5 stage MIPS optimized pipeline execution:

```
I1:| IF | ID | EX  | MEM | WB  |
I2:     | IF | ID  | EX  | MEM | WB|   MEM >> MEM forwarding path used
```

Which pair of instructions are executed?

(SINGLE ANSWER)
1 point

Answer 1: `I1: LW $t0, 0($t2): I2: SW $t1, 0 ($t0)`
Answer 2: `I1: LW $t0, 0($t2): I2: SW $t0, 0 ($t2)` **(TRUE)**
Answer 3: `I1: LW $t0, 0($t2): I2: SW $t2, 0 ($t2)`

Question 8 (format true/false)

A (2,2) Correlating Branch Predictor has a double number of prediction bits with respect to a (2,1) Correlating Branch Predictor with the same number of entries in each BHT.

(TRUE/FALSE)
1 point

Answer 1: **T (TRUE)**

Question 9 (format Multiple Choice – Single answer)

Let's consider a (2,1) Correlating Branch Predictor with 4K total entries.

How many Branch History Tables?
How many entries per BHT?
How many bits per entry?

(SINGLE ANSWER)
1 point

Answer 1: 4 BHTs | 1K entries | 1-bit per entry **(TRUE)**
Answer 2: 2 BHTs | 2K entries | 1-bit per entry
Answer 3: 2 BHTs | 2K entries | 2-bit per entry
Answer 4: 1 BHT | 4K entries | 1-bit per entry

Question 10 (format Multiple Choice – Multiple answers)

Which of the following cache improvements are effective in reducing the ***miss penalty?***
(Multiple answers)
2 points

Answer 1: Pseudo-associativity and Way prediction.
Answer 2: Early restart and critical word first. **(TRUE)**
Answer 3: Giving higher priority to read misses over write misses. **(TRUE)**
Answer 4: Introducing a second level cache. **(TRUE)**

Feedback:
Adopting pseudo-associativity and way prediction are optimization techniques to reduce the miss rate. The other two answers instead are reducing the miss penalty by (1) the introduction of early restart and critical word first techniques are used to do not wait for the full block and (2) by giving priority to read misses over writes, they are used to serve the read misses before the write misses. See slides on L11B: Memory_Hierarchy_Advanced Concepts.

Question 11 (format Multiple Answers)

For multithreading processors, which of the following statements are true?

(MULTIPLE ANSWERS)
2 points

Answer 1: Multiple processors communicate through message passing protocols.
Answer 2: Multiple threads can share the functional units of a single processor. **TRUE**
Answer 3: The processor must duplicate the independent state of each thread by a separate copy of the register set. **TRUE**
Answer 4: The processor must keep independent threads by a separate Program Counter for each thread. **TRUE**

Question 12 (format Multiple answers)

In a shared memory multiprocessor, which of the following statements are true?

(Multiple answers)
2 points

Answer 1: Multiple processors communicate through send/receive message passing protocols.

Answer 2: Multiple processors communicate through shared memory variables and data writes generate coherence problems among multiple caches. **TRUE**

Answer 3: Coherent caches provide multiple copies of shared data to reduce both access latency and read contention. **TRUE**

Answer 4: The shared memory can be either centralized on a single node or distributed over the nodes. **TRUE**

Question 13 (format Multiple Choice – Single answer)

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

- *Vector Register File composed of 8 vectors of 32 elements per 64 bits/element;*
- *Scalar FP Register File composed of 32 registers of 64 bits;*
- *One Load/Store Vector Unit **without operation chaining** and memory bandwidth 64 bits;*
- *One ALU Vector Unit **without operation chaining**.*

```
L.V V1, RX              # load vector from memory address RX into V1
ADDVS.D V2, V1, F0      # FP add vector V1 to scalar F0
MULVV.D V3, V1, V2      # FP Multiply vectors V1 and V2
```

How many convoys? How many clock cycles to execute the code?
(SINGLE ANSWER)
2 points

Answer 1: 3 convoys; 96 clock cycles **(TRUE)**
Answer 2: 2 convoys; 64 clock cycles
Answer 3: 8 convoys; 32 clock cycles
Answer 4: 1 convoy; 32 clock cycles

Question 14 (format Multiple Choice – Single answer)

Let us consider to apply a processor optimization resulting ten time faster on computation than the original mode of execution.
What is the fraction of computation needed to achieve an overall speedup of 2?

(SINGLE ANSWER)
2 points

Answer 1: 50%
Answer 2: 55% (TRUE)
Answer 3: 25%
Answer 4: 75%
Answer 5: 20%

Feedback:

1. *To get an overall speedup of 2, we need to apply the Amdahl's law as follows:*
   $1 / [(1-F_V) + (F_V / 10)] = 2 => 10 / (10 – 9 F_V) = 2 => 18 F_V = 10$
   $=> F_V = 10/ 18 = 0.55 =>$ ***$F_V = 55\%$***

Question 15 (format Multiple Choice – Multiple answers)

Let's consider the following loop code executed by a dynamic scheduled processor:

```
S1: LOOP: LD.D $FP1, A($R1)
S2:       LD.D $FP2, B($R1)
S3:       MUL.D $FP1, $FP1, $FP1
S4:       ADD.D $FP2, $FP1, $FP2
S5:       SD.D $FP1, C($R1)
S6:       SD.D $FP2, D($R1)
S7:       ADDI $R1, $R1, 4
S8:       BNE $R1, $R2, LOOP
```

Assuming to consider only the intra-iteration dependencies: Where are the ***OUTPUT DEPENDENCIES*** *that can generate the WAW hazards?*

(MULTIPLE ANSWERS)
2 points

Answer 1: Between S4 and S2 on $FP2; **(TRUE)**
Answer 2: Between S7 and S6 on $R1;
Answer 3: Between S3 and S1 on $FP1; **(TRUE)**
Answer 4: Between S5 and S3 on $FP1;
Answer 5: Between S6 and S4 on $FP2;

Prof. Cristina Silvano	ACA2022_EXAM_FORM1 Data: 21/07/2022

ACA2022_FORM1_21July2022_SILVANO

ACA Course -- Prof. SILVANO

FORM1 is composed of 15 QUESTIONS to get UP TO 21 POINTS

Duration is 35 minutes!

IMPORTANT: What to do before leaving MS FORM1:

1. **Please click on:** "Inviami una conferma tramite posta elettronica delle risposte" (Send me a confirmation by email of the answers). This is **fundamental** to get an email with subject: **"My responses".** Open the email and click on the link to **VIEW YOUR RESPONSES.**

2. **Please click on SUBMIT** (Click on **INVIA**) and you will get the message: **"Your answer has been sent".** Points will be assigned **manually** to **FORM1**. Therefore, the view of your results is **not** immediate: it will be enabled by prof. Silvano **after** results publication phase.

Question 1 (format Multiple Choice – Single answer)

Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where we consider the block B1 in the directory of N1:

Directory N1 Block B1 | State: Shared | Sharer Bits: 1001

Which is the request sent for the block B1 to become:

Directory N1 Block B1 | State: Modified | Sharer Bits: 0100

(SINGLE ANSWER)
1 point

Answer 1: Write Hit B1 sent from N1 to N1
Answer 2: Write Miss B1 from N1 to N1 **(TRUE)**
Answer 3: Read Hit B1 from N1 to N1
Answer 4: Read Miss B1 from N1 to N1

Question 2 (format Multiple Choice – Single answer)

Let's consider a quad-issue SMT processor that can manage up to 8 simultaneous threads. *What are the values of the ideal CPI and the ideal per-thread CPI?*

(SINGLE ANSWER)
1 point

Answer 1: Ideal CPI = 1 & Ideal per-thread CPI = 0.125
Answer 2: Ideal CPI = 0.25 & Ideal per-thread CPI =4
Answer 3: Ideal CPI = 0.5 & Ideal per-thread CPI = 2
Answer 4: Ideal CPI = 0.5 & Ideal per-thread CPI = 4
Answer 5: Ideal CPI = 0.25 & Ideal per-thread CPI = 2 **(TRUE)**

Question 3 (format Multiple Choice – Single answer)

Let's consider a 1-bit 1-entry Branch History Table initialized as not-taken and a loop iterated 100 times. How many branch mispredictions will be incurred by the conditional branch that takes the execution back to the beginning of the loop (loop-backward branch)?
(SINGLE ANSWER)
1 point

Answer 1: 1%
Answer 2: 2% **(TRUE)**
Answer 3: 99 %
Answer 4: 100%
Answer 5: 98 %

Feedback
Assume initial state NT, we have 2 mispredictions: once when the loop-back branch is first executed and once when the loop exits (98% success and 2% misprediction).

Question 4 (format Multiple answers)

In a distributed shared memory multiprocessor, which of the following statements are true?

(Multiple answers)
2 points

Answer 1: The access time to a shared memory location is uniform for all the processors (UMA)

Answer 2: The access time to a shared memory location is non uniform for all the processors (NUMA) **TRUE**

Answer 3: Multiple processors communicate through load/store operations on shared memory variables. **TRUE**

Answer 4: The shared memory model generates the cache coherence problem among multiple processors. **TRUE**

Answer 5: The shared memory can be either centralized on a single node or distributed over the nodes.

Question 5 (format Multiple answers)

The prediction bit in a Branch History Table could belong to another branch with the same low-order address bits than the current branch, therefore this can generate a misprediction.
To reduce this collision problem, what are the solutions that can be proposed?

(Multiple answers)
2 points

Answer 1: Increase the number of rows in the BHT, even if this is costly in terms of HW. **True**

Answer 2: Introduce a hashing function that maps the branch address on k-bits, even if this can increase the access time to the BHT. **True**

Answer 3: Introduce more prediction bits in each entry of the Branch History Table, even if this is costly in terms of HW.

Answer 4: Combine the BHT to the Branch Target Buffer, that uses tags. **True**

Question 6 (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, RX          # load vector from memory address RX into V1
MULVS.D V1, V1, F0  # FP multiply vector V1 to scalar F0
L.V V2, RY          # load vector from memory address RY into V2
MULVS.D V2, V2, F0  # FP multiply vector V2 to scalar F0
ADDVV.D V3, V1, V2  # FP add vectors V1 and V2
S.V V3, RZ          # store vector V3 into memory address RZ
```

How many convoys? How many clock cycles to execute the code?

(SINGLE ANSWER)
2 points

Answer 1: 3 convoys; 24 clock cycles **(TRUE)**
Answer 2: 2 convoys; 16 clock cycles
Answer 3: 4 convoys; 32 clock cycles
Answer 4: 5 convoys; 40 clock cycles
Answer 5: 3 convoys; 25 clock cycles

Question 7 (format Multiple Choice – Multiple answers)

Let's consider the following loop code:

```
for (i=1; i<=100, i++) {
      X[i] = X[i-1] + Y[i-1] + Z[i-1];     /*S1*/
      Z[i] = Y[i-1] + W[i-1]               /*S2*/
}
```

What are the loop-carried dependences in the code?

(MULTIPLE ANSWERS)

2 points

Answer 1: One in S1 because X[i] depends on X[i-1]; **(TRUE)**

Answer 2: One in S1 because X[i] depends on Y[i-1];

Answer 2: One in S1 because X[i] depends on Z[i-1]; (**TRUE**)

Answer 4: One in S2 because Z[i] depends on Y[i-1];

Answer 5: One in S2 because Z[i] depends on W[i-1];

Feedback

Please note that the dependences of Z[i] on Y[i-1] and on W[i-1] are not loop-carried dependences because the vectors Y[ ] and W[] are never modified in the loop.

Question 8 (format Multiple Choice – Single answer)

Let's consider the following loop kernel to be repeated for 100 iterations:

```
LOOP:  ld.d $FP2,VECTA($s1)
IF:    beq $s1, $s2, ELSE
THEN:  ld.d $FP6, 8($s1)          // more probable path
       sd.d $FP6, 8($s6)
       j INC
ELSE:  ld.d $FP2, 8($s1)         // less probable path
       add.d $FP2, $FP2, $FP0
       mul.d $FP4, $FP2, $FP2
       sd.d $FP2, 12($s1)
INC:   addi $s1,$s1,8
       bne $s0,$s7, LOOP
```

What is the best static branch prediction technique to be applied?

(SINGLE ANSWER)
1 point

Answer 1: Backward Taken Forward Not Taken (**TRUE**)
Answer 2: Always Taken
Answer 3: Always Not Taken
Answer 4: Delayed Branch

Question 9 (Format Multiple Choice – Multiple answers)

A 4-issue VLIW processor exploits:

(MULTIPLE ANSWERS)
2 points
Answer 1: Data-level parallelism by applying a single instruction to multiple data.
Answer 2: Insertion of NOPs to solve true data dependencies. **(TRUE)**
Answer 3: Dynamic scheduling techniques
Answer 4: Multiple-issue instruction level parallelism. **(TRUE)**
Answer 5: Static scheduling techniques **(TRUE)**

Question 10 (format True/False)
To speed up the access time, the level-1 cache can be virtually indexed and physically tagged in order to overlap the cache tag access with virtual address translation.

(True/False)
1 point

Answer: **TRUE**

Question 11 (format True/False)
In the Speculative Tomasulo architecture, the Store Path is designed to connect the store buffers to the memory unit.

(format True/False)
1 point

Answer 1: False

Feedback
In the Speculative Tomasulo architecture, there are no Store Buffers because they are substituted by ReOrder Buffers with the Store Path directly connecting to the memory unit.

Question 12 (format Multiple Choice – Reorder options)

Let's consider a directory-based protocol for a distributed shared memory system with 4 nodes (N0, N1, N2, N3) where:

Directory N1 Block B1 | State: Modified | Sharer Bits: 0010

After a **Write Miss on B1 from node N0**, reorder the sequence of messages sent among the nodes to get:

Directory N1 Block B1 | State: Modified | Sharer Bits: 1000

(REORDER OPTIONS)
1 point

1. **Fetch/Invalidate** message sent from home node N1 to remote node N2
2. **Data Write Back** message from past owner N2 to home node N1
3. **Data Value Reply** from home node N1 to local cache N0.

Feedback:

Due to the Write Miss on B1 from the Local Node N0 to home node N1, first there is **a Fetch/Invalidate** message sent from the home node N1 to the remote node N2 **(past owner)** to get the most recent copy of the block in the home node N1 through a **Data Write Back** message from past owner N2 to home node N1and invalidating the state of the block B1 in the past owner's cache C2.
Then there is a **Data Value Reply** from home node N1 to local cache N0.
The state of the block B1 in the directory N1 stays **Modified** but the **owner is changed** from N2 to N0 as follows: Directory N1 Block B1 | State: **Modified** | Sharer Bits: **1000.**

Question 13 (format Multiple Choice – Single answer)

Let's consider a **Speculative Tomasulo** architecture with 2 load buffers (Load1, Load2), 2 multiply reservation stations (Mult1 and Mult2) and 6-entry ROB (ROB0, ROB1, ROB2, ROB3, ROB4, ROB5).
Let's consider the following LOOP when ROB is full after the issue of the first loop iteration (while the first load is executing a cache miss) and the speculative issue of the LD of the second iteration.

```
LOOP: LD $F0, 0 ($R1)
      MULTD $F4, $F0, $F2
      SD $F4, 0 ($R1)
      SUBI $R1, $R1, 8
      BNEZ $R1, LOOP    // branch prediction taken
```

In the Rename Table, what are the pointer values used for $F0 and $F4?

(SINGLE ANSWER)
2 points

Answer 1: ROB5 for $F0 and ROB1 for $F4 **(TRUE)**
Answer 2: ROB5 for $F0 and ROB2 for $F4
Answer 3: ROB0 for $F0 and ROB1 for $F4
Answer 4: ROB0 for $F0 and ROB2 for $F4

Feedback:
The 6-entry ROB is FULL:

ROB#	Instruction	Dest.	Ready	
ROB0	LD $F0, 0 ($R1) (1^ iteration exec. cache miss)	$F0	No	HEAD
ROB1	MULTD $F4, $F0, $F2 (1^ iteration issued)	$F4	No	
ROB2	SD $F4, 0 ($R1) (1^ iteration issued)	MEM	No	
ROB3	SUBI $R1, $R1, 8 (1^ iteration issued)	$R1	No	
ROB4	BNEZ $R1, LOOP (1^ branch predicted as taken)		No	
ROB5	LD $F0, 0 ($R1) (2^ iteration issued speculatively)	$F0	No	

Rename Table:

$F0	ROB5
$F2	

$F4	ROB1

Therefore, in the Rename Table, $F0 points to ROB5 because the WAW $F0 is solved while $F4 points to ROB1 because the second MULTD has not yet been issued because the ROB is full. See also L07: Reorder Buffer & Speculation

Question 14 (format Multiple Choice – Single answer)

In a directory-based protocol for cache coherency, the message called "Data Value Reply" is sent:
(SINGLE ANSWER)
2 points

Answer 1: From the local node cache to return data to the home directory
Answer 2: From the home directory to return data to the local node cache **(TRUE)**
Answer 3: From the remote node cache to return data to the home directory

Feedback:
See Directory-based protocols messages on the slides on L13: Multiprocessors.

Question 15 (format True/False)
To obtain a loop unrolling version of a code, we need to use register renaming if there are some true data dependences in the original code.

(format True/False)
1 point

Answer 1: False

Prof. Cristina Silvano	ACA2022_EXAM_FORM1 Data: 08/09/2022

ACA2022_FORM1_8Sept2022_SILVANO

ACA Course -- Prof. SILVANO

FORM1 is composed of 14 QUESTIONS to get UP TO 21 POINTS

Duration is 35 minutes!

IMPORTANT: What to do before leaving MS FORM1:

1. **Please click on:** "Inviami una conferma tramite posta elettronica delle risposte" (Send me a confirmation by email of the answers). This is **fundamental** to get an email with subject: **"My responses".** Open the email and click on the link to **VIEW YOUR RESPONSES.**

2. **Please click on SUBMIT** (Click on **INVIA**) and you will get the message: **"Your answer has been sent".** Points will be assigned **manually** to **FORM1**. Therefore, the view of your results is **not** immediate: it will be enabled by prof. Silvano **after** results publication phase.

Question 1 (format Multiple Choice – Single answer)

Let's consider the following LOOP to be executed for 100 iterations on a processor with 1-entry 2-bit Branch History Table initialized at the state Weakly Taken:

```
LOOP: LD $F0, 0 ($R1)
      ADDI.D $F2, $F0, $F0
      SD $F2, 0 ($R1)
      ADDI.U $R1, $R1, 4
      BNE $R1, $R2, LOOP
```

How many branch mispredictions will occur by the BNE instruction?

(SINGLE ANSWER)
2 points

Answer 1: 1% **(TRUE)**
Answer 2: 2%
Answer 3: 99 %
Answer 4: 100 %
Answer 5: 98 %

Feedback
Given the initial state Weakly Taken, the 2-bit predictor fails only on the last branch therefore there are 99% correct predictions and 1% mispredictions.

Question 2 (format Multiple Choice – Single answer)

Let's consider the following LOOP to be executed for 100 iterations on a processor with 1-entry 2-bit Branch History Table initialized in the state Strongly Not-Taken:

```
LOOP: LD $F0, 0 ($R1)
      ADDI.D $F2, $F0, $F0
      SD $F2, 0 ($R1)
      ADDI.U $R1, $R1, 4
      BNE $R1, $R2, LOOP
```

How many branch mispredictions will occur by the BNE instruction?

(SINGLE ANSWER)
2 points

Answer 1: 3% **(TRUE)**
Answer 2: 2%
Answer 3: 99 %
Answer 4: 100%
Answer 5: 98 %

Feedback

Given the initial state Strongly Not-Taken, at the beginning the 2-bit predictor fails twice before changing the prediction to Taken and it also fails on the last branch. Therefore, there are 97% correct predictions and 3% mispredictions.

Question 3 (format True/False)

Write-through policy for caches uses a ReOrder Buffer to continue to service read requests while the writes are served towards the main memory.

(True/False)
1 point

Answer 1: False

Feedback
*Write through policy for caches always uses a **Write Buffer** to continue to service read requests while the writes are served towards the main memory.*

Question 4 (format Multiple Choice – Single answer)

Let's consider an application kernel characterized by multiple consecutive writes to the same variables. Which *write protocol* should be better to use for the management of the memory hierarchy?

(SINGLE ANSWER)
1 point

Answer 1: Write-through
Answer 2: Write-back **(TRUE)**
Answer 3: Indifferent

Feedback:
With the write-though protocol, every write is done to memory to update with all the consecutive writes done to the same variables. The write-back protocol will write to memory

the value of the last write only when there is a cache miss: This is the best choice for the given kernel to avoid useless bus traffic due to multiple memory accesses.

Question 5 (Format Multiple Choice – Single answer)

What is the main purpose of a Branch History Table?

(SINGLE ANSWER)
1 point

Answer 1: To buffer the predicted Branch Target Address.
Answer 2: To buffer the predicted Branch Outcome before it is definitively known based on the previous branch behavior. **(TRUE)**
Answer 3: To buffer the correlation of the past behavior of another branch with respect to the given branch.

Question 6 (format Multiple Choice – Single answer)

Let us assume that, given an optimization technique, a fraction of instructions of an application code is parallelized to obtain, a CPI = 2.5 while the remaining fraction of instructions is executed sequentially with the original CPI = 5.
How much is the fraction of parallelized instructions to get an overall Speedup equal to 10?

(SINGLE ANSWER)
2 points

Answer 1: 2
Answer 2: 1.8 **(TRUE)**
Answer 3: 0.5
Answer 4: 0.2

Feedback:
To calculate the overall speedup we need to apply the Amdahl's law as follows:
$Fraction_E = x$;
$Speedup_E = CPI_o / CPI_E = 5 / 2.5 = 2$;
$Speedup = 1 / [(1-x) + (x / Speedup_E)] \Rightarrow 10 = 1 / [(1-x) + (x / 2)] \Rightarrow x = 9 / 5 = 1.8$

Question 7 (format Multiple Choice – Single answer)

Let's consider a directory-based protocol for a distributed shared memory system with 4 Nodes (N0, N1, N2, N3) where we consider the block B2 in the directory of N2:

Directory N2 Block B2 | State: Shared | Sharer Bits: 1001

Which is the request sent for the block B2 to become:

Directory N2 Block B2 | State: Modified | Sharer Bits: 0100

(SINGLE ANSWER)
1 point

Answer 1: Write Hit B2 sent from N1 to N1
Answer 2: Write Miss B2 from N1 to N2 **(TRUE)**
Answer 3: Read Hit B2 from N1 to N2
Answer 4: Read Miss B2 from N1 to N2

Question 8 (format Multiple answers)
What are the true statements in a shared memory multiprocessor?

(MULTIPLE ANSWERS)
2 points

Answer 1: The access time to a shared memory variable could be either uniform or not uniform for all the processors. **TRUE**

Answer 2: Multiple processors communicate by explicit send/receive messages because they do not have direct access to the shared memory variables.

Answer 3: Multiple copies of shared memory variables generate cache coherence problems among the multiple processors. **TRUE**

Answer 4: The physical memory can be either centralized on a single node or distributed over the nodes. **TRUE**

Question 9 (format Multiple Choice – Multiple answers)

Let's consider the following loop code:

```
for (i=1; i<=100, i++) {
      X[i] = X[i-1] + Z[i-1];          /*S1*/
      Y[i] = Y[i-1] + W[i-1];          /*S2*/
      W[i] = W[i-1];                   /*S3*/
}
```

What are the loop-carried dependences in the code?

(MULTIPLE ANSWERS)

2 points

Answer 1: One in S1 because X[i] depends on X[i-1]; (**TRUE**)

Answer 2: One in S1 because X[i] depends on Z[i-1];

Answer 3: One in S2 because Y[i] depends on Y[i-1]; (**TRUE**)

Answer 4: One in S2 because Y[i] depends on W[i-1]; (**TRUE**)

Answer 5: One in S3 because W[i] depends on W[i-1]; (**TRUE**)

Feedback

Please note that the dependences of X[i] on Z[i-1] is not loop-carried dependences because the vector Z[] are never modified in the loop code.

Answer 1: True

Question 10 (Format Multiple Choice – Single answer)

Consider the following assembly code:

```
            ADDI $R1, $R0, 0
            ADDI $R2, $R0, 100

LOOP:       BEQ $R1, $R2, DONE
            ADD $R5, $R5, $R4
            ADDI $R1, $R1, 4
            J   LOOP
```

Let's assume *that **only** the BEQ instruction accesses the 1 entry 1-bit Branch History Table. How many loop iterations? How many accesses to the BHT are done?*

(SINGLE ANSWER)

2 points

Answer 1: 25 iterations, 26 BHT accesses **(TRUE)**
Answer 2: 100 iterations, 101 BHT accesses
Answer 3: 100 iterations, 100 BHT accesses
Answer 4: 25 iterations, 25 BHT accesses

Feedback:
There are 25 loop iterations.
For each iteration, the BEQ instruction accesses the BHT at each iteration plus one additional time at the end of the loop for a total of 26 BHT accesses.

Question 11 (format True/False)
In the Speculative Tomasulo architecture, the Reorder Buffer is combined to a Rename Table to solve WAW/WAR hazards

(True/False)
1 point

Answer 1: True

Question 12 (format Multiple Choice – Single answer)
*To obtain **a loop unrolling** version of a code, which technique we need to apply to solve output dependences (WAW) and anti-dependences (WAR)?*

(SINGLE ANSWER)
1 point

Answer 1: Register Renaming **True**
Answer 2: List-based Scheduling
Answer 3: Memory Aliasing
Answer 4: Trace Scheduling

Question 13 (format True/False)
Multithreading could be obtained even on a single-core processor architecture making the processor able to manage multiple threads concurrently: if a thread gets stalled, the processor can execute another thread by keeping the functional units busy. **True**

(True/False)
1 point

Answer 1: True

Question 14 (format True/False)
What are the true sentences related to ***multithreading:***

Answer 1: Each thread must preserve its computational state by a private PC and a set of private registers that are separated from the other threads. **True**

Answer 2: Thread switch must be managed by hardware support to get more efficient than process switch supported by the OS. **True**

Answer 3: Simultaneous multithreading is convenient for multiple-issue CPUs because there is a number of functional units that cannot be kept busy with instructions from a single thread. **True**

Answer 4: Multithreading requires a data-parallel programming model

(MULTIPLE ANSWERS)

2 points