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:
$1 / [(1-F_V) + (F_V / 10)] = 2$
$=> 10 / (10 – 9 F_V) = 2$
$=> 18 F_V = 10$
$=> F_V = 10/ 18 = 0.55$
$=> F_V = 55\%$