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