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]**

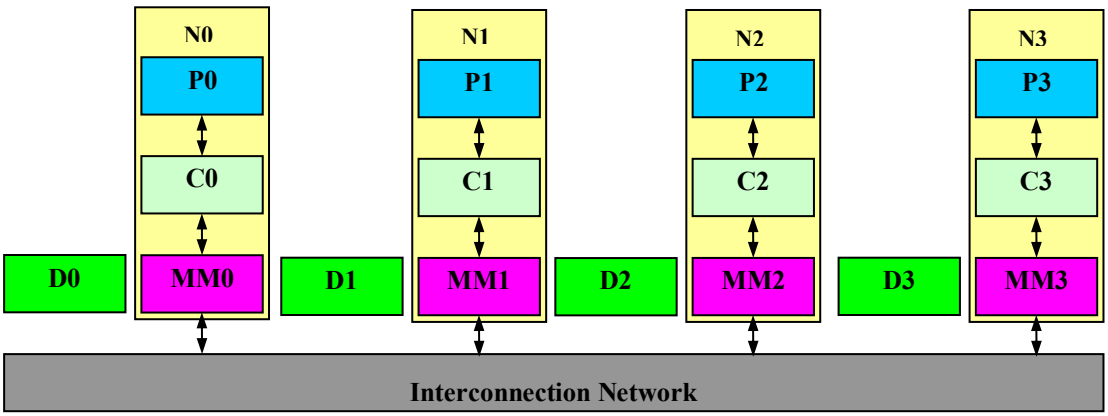
Being a set-associative cache, the block replacement policy in each set uses the **LRU algorithm**.

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*