Advanced Computer Architectures
Exercises

Christian Rossi

Academic Year 2023-2024

Abstract

The course topics are:

- Review of basic computer architecture: the RISC approach and pipelining, the memory hierarchy.
- Basic performance evaluation metrics of computer architectures.
- Techniques for performance optimization: processor and memory.
- Instruction level parallelism: static and dynamic scheduling; superscalar architectures: principles and problems; VLIW (Very Long Instruction Word) architectures, examples of architecture families.
- Thread-level parallelism.
- Multiprocessors and multicore systems: taxonomy, topologies, communication management, memory management, cache coherency protocols, example of architectures.
- Stream processors and vector processors; Graphic Processors, GP-GPUs, heterogeneous architectures.

Contents

CHAPTER 1

Performance evaluation

1.1 Exercise one

Assessing the impact of modifications on performance:

1. Substituting a hardware component with a faster alternative.
2. Incorporating multiple parallel systems for executing independent tasks.

Solution

1. By scaling up: response time decreases, while throughput increases.
2. Through scaling out: throughput experiences an increase. Response time will only escalate if a queue was present, awaiting computing resources.

1.2 Exercise two

Let's consider two CPUs: CPU1 and CPU2. CPU1 operates with a clock cycle of $2\,ns$, while CPU2 has an operating frequency of $700\,MHz$. Given the frequencies of occurrence of instructions for both CPUs:

Operation type	**Frequency**	**CPU1 cycle**	**CPU2 cycle**
A	0.3	2	2
B	0.1	3	3
C	0.2	4	3
D	0.3	2	2
E	0.1	4	3

1. Calculate the average CPI for CPU1 and CPU2.
2. Determine which CPU is the fastest.

Solution

1. The CPI (Cycle Per Instruction) is calculated as:

$$\text{CPI} = \frac{\text{clock cycles}}{\text{instruction}}$$

The average CPI is then obtained by:

$$\sum_{i=1}^{n} \text{CPI}_i \cdot \text{F}_i$$

Where, $\text{F}_i = \frac{\text{I}_i}{\text{instruction count}}$. For CPU1:

$$\text{CPI}_1 = 0.3 \cdot 2 + 0.1 \cdot 3 + 0.2 \cdot 4 + 0.3 \cdot 2 + 0.1 \cdot 4 = 2.7$$

For CPU2:

$$\text{CPI}_2 = 0.3 \cdot 2 + 0.1 \cdot 3 + 0.2 \cdot 3 + 0.3 \cdot 2 + 0.1 \cdot 3 = 2.4$$

2. We have that:

$$\begin{aligned}\frac{\text{EXE}_{\text{CPU1}}}{\text{EXE}_{\text{CPU2}}} &= \left(\frac{\text{IC}_1 \cdot \text{CPI}_1}{F_1}\right)\left(\frac{F_2}{\text{IC}_2 \cdot \text{CPI}_2}\right)\\ &= \frac{\text{IC}_1 \cdot \text{CPI}_1 \cdot F_2}{F_1 \cdot \text{IC}_2 \cdot \text{CPI}_2}\\ &= \frac{\text{CPI}_1 \cdot F_2}{F_1 \cdot \text{CPI}_2}\\ &= \frac{2.7 \cdot 700MHz}{2.4 \cdot 500MHz}\\ &= 1.575\end{aligned}$$

Hence, CPU2 is approximately 1.575 times faster than CPU1.

1.3 Exercise three

The speed of image processing on FPGA is 2.86 times faster than on a CPU. The power consumption of an FPGA is 100 W, while that of the CPU is 30.85 W. We want to achieve a speedup of 2 with the addition of an FPGA.

Solution

To achieve a speedup of 2 with the addition of an FPGA, we use Amdahl's law:

$$S_{overall} = \frac{1}{(1 - F_{enhanced}) + \frac{F_{enhanced}}{S_{enhanced}}}$$

Given $S_{overall} = 2$, we solve for F_{enhanced}:

$$2 = \frac{1}{(1 - F_{enhanced}) + \frac{F_{enhanced}}{2.86}} \rightarrow F_{enhanced} = 0.768$$

Therefore, with 76.8% of the processing offloaded to the FPGA, a speedup of 2 can be achieved.

CHAPTER 2

Branch prediction

2.1 Exercise one

Explain (with effective support) the design of a 1-bit branch history table (1-BHT) and a 2-bit branch history Table (2-BHT) capable of executing the provided assembly code (with R0 set to 2000 and R1 set to 0).

```
LOOP:   LD F1 0 R0
        ADDD F2 F1 F1
        ADDI R1 R1 100
LOOP2:  MULTD F2 F2 F1
        SUBI R1 R1 1
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

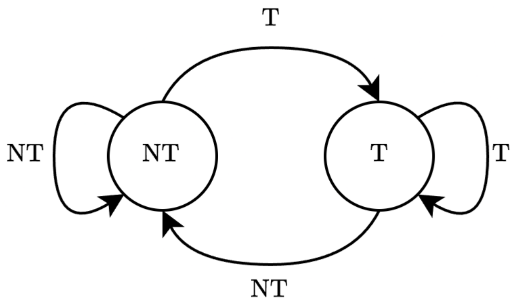
Compute the number of mispredictions in the various cases.

Solution

The outer loop iterates 1000 times because R0 starts at 2000 and decreases by two each iteration.

The inner loop iterates 100 times for each outer loop iteration because R1 starts at 100 and decreases by one each iteration. Consequently, the total iterations of the inner loop amount to $1000 \times 100 = 100000$.

For the one-bit branch table computation, a finite state machine can be employed, depicted below:



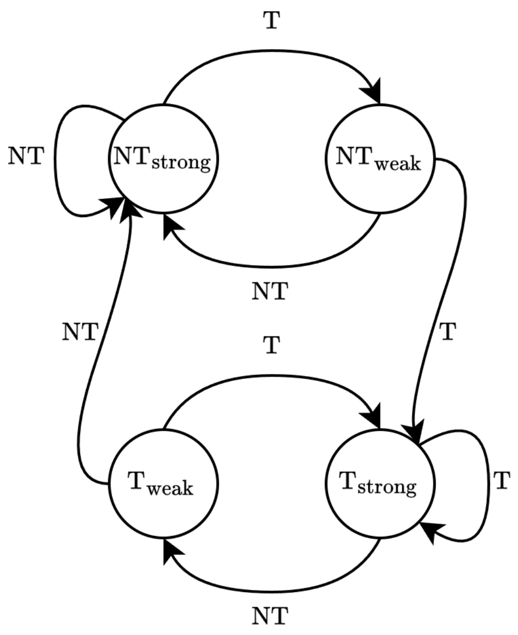
In the absence of collision, one bit can be assigned for each loop. This allows four possible initializations: T-T, T-NT, NT-T, NT-NT.

- For the first case (T-T), there are $1 + 1 + (1000 - 1) \times 2$ iterations.
- For the second case (T-NT), there are $1 + 1000 \times 2$ iterations.
- For the third case (NT-T), there are $2 + 1 + (1000 - 1) \times 2$ iterations.
- For the fourth case (NT-NT), there are $2 + 1000 \times 2$ iterations.

In the presence of collision, one bit is allocated for each loop. This yields two possible initializations: T, NT. The iterations for each case are calculated as follows:

- For the first case (T), there are $(1 + 1) \times (1000 - 1) + 1$ iterations.
- For the second case (NT), there are $1 + (1 + 1) \times (1000 - 1) + 1$ iterations.

The two-bit branch table can be designed using a finite state machine, as illustrated below:



In the absence of collision, where one bit is assigned for each loop, there are eight possible initializations. The worst cases are:

- $\mathrm{NT_{weak}}, \mathrm{NT_{weak}}$: resulting in 1000×2 mispredictions for LOOP2 and 2 mispredictions for LOOP.
- $\mathrm{NT_{strong}}, \mathrm{NT_{strong}}$: resulting in $3 + (1000 - 1) \times 1$ mispredictions for LOOP2 and 2 mispredictions for LOOP.

The best cases are:

- $\mathrm{T_{weak}}, \mathrm{T_{weak}}$: Resulting in $1 + (1000 - 1) \times 2$ mispredictions for LOOP2 and 1 for LOOP.
- $\mathrm{T_{strong}}, \mathrm{T_{strong}}$: Resulting in 1000×1 mispredictions for LOOP2 and 1 for LOOP.

Considering these cases, the iterations for each case are calculated accordingly:

- For the first case, we have $1 + 1 + (1000 - 1) \times 2$.
- For the second case, we have $1 + 1000 \times 2$.
- For the third case, we have $2 + 1 + (1000 - 1) \times 2$.
- For the fourth case, we have $2 + 1000 \times 2$.

In the presence of collision, where one bit is used for each loop, there are four possible initializations ($\mathrm{T}_{\mathrm{strong}}$, $\mathrm{T}_{\mathrm{weak}}$, TNstrong, NT : weak). The iterations for each case are calculated as follows:

- For the first case, we have $1 \times 1000 + 1$.
- For the second case, we have $1 \times 1000 + 1$.
- For the third case, we have $2 + 1 \times 1000 + 1$.
- For the fourth case, we have $1 + 1 \times 1000 + 1$.

Comparing the worst-case scenario of the two-bit branch table with the best-case scenario of the one-bit branch table, it's evident that the worst 2BHT performs better than the best 1BHT.

2.2 Exercise two

Describe (the answer has to be effectively supported) a 1-BHT and a 2-BHT able to execute the following assembly code (R0 is set to 1, R1 is set to 300).

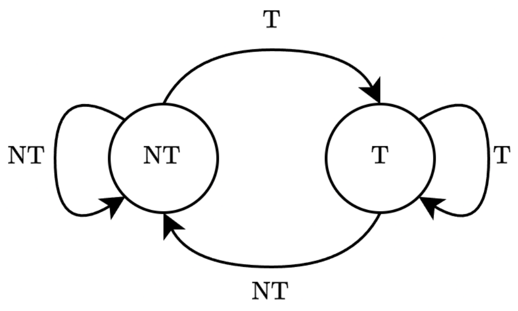
```
LOOP:   LD F3 0 (R0)
        ADDD F1 F3 F3
        ADDI R1 R1 3000
LOOP2:  MULTD F2 F2 F3
        SUBI R1 R1 3
        BNEZ R1 LOOP2
        SUBI R0 R0 2
        BNEZ R0 LOOP
```

The obtained result, in terms of mispredictions, is inline with theoretical characteristics of the two predictors? Please effectively support your answer.

Solution

We have that $R0$ is set to 1, and $R1$ is set to 300. The internal loop is looped $\frac{3300}{3} = 1100$ times. The register for the external loop becomes -1 before the first branch check, and so we have an infinite loop (impossible to have $R0$ set to zero)

For the one-bit branch table computation, a finite state machine can be employed, depicted below:



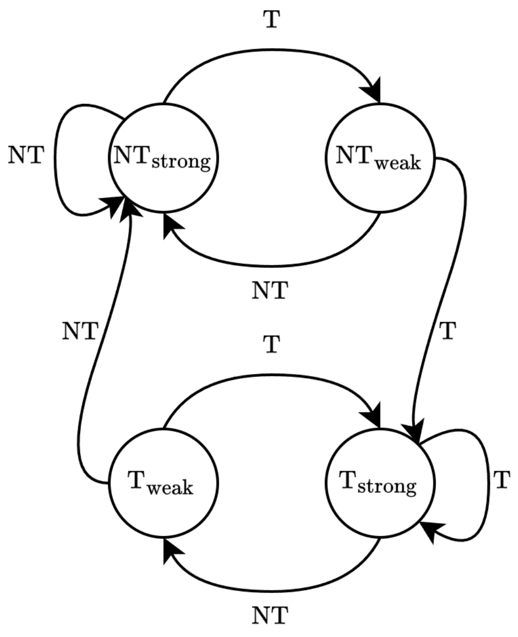
In the absence of collision, one bit can be assigned for each loop. This allows four possible initializations: T-T, T-NT, NT-T, NT-NT.

- For the first case (T-T), there are NO misprediction for loop, and for loop2 a single misprediction at the end of the loop @T0 + 2 misprediction x iteration begin and end of the loop
- For the second case (T-NT), there are NO misprediction for loop, and for loop2 two misprection: beginning and end of the loop
- For the third case (NT-T), there are only initial misprediction for loop, and for loop2 single misprediction at the end of the loop @T0 + 2 misprediction x iteration begin and end of the loop
- For the fourth case (NT-NT), there are only initial misprediction for loop, and for loop2 two misprection: beginning and end of the loop.

In the presence of collision, one bit is allocated for each loop. This yields two possible initializations: T, NT. The iterations for each case are calculated as follows:

- For the first case (T), there are single misprediction at the end of the loop2, and a 100% failure rate for loop.
- For the second case (NT), there are two initial misprection, then end of the loop and a 100% failure rate for loop.

The two-bit branch table can be designed using a finite state machine, as illustrated below:



In the absence of collision, where one bit is assigned for each loop, there are eight possible initializations. The worst cases are:

- NT_{weak}, NT_{weak}: resulting in $2 \times \infty$ mispredictions for LOOP2 and 1 mispredictions for LOOP.
- NT_{strong}, NT_{strong}: resulting in $2+1+1\times\infty$ mispredictions for LOOP2 and 2 mispredictions for LOOP.

The best cases are:

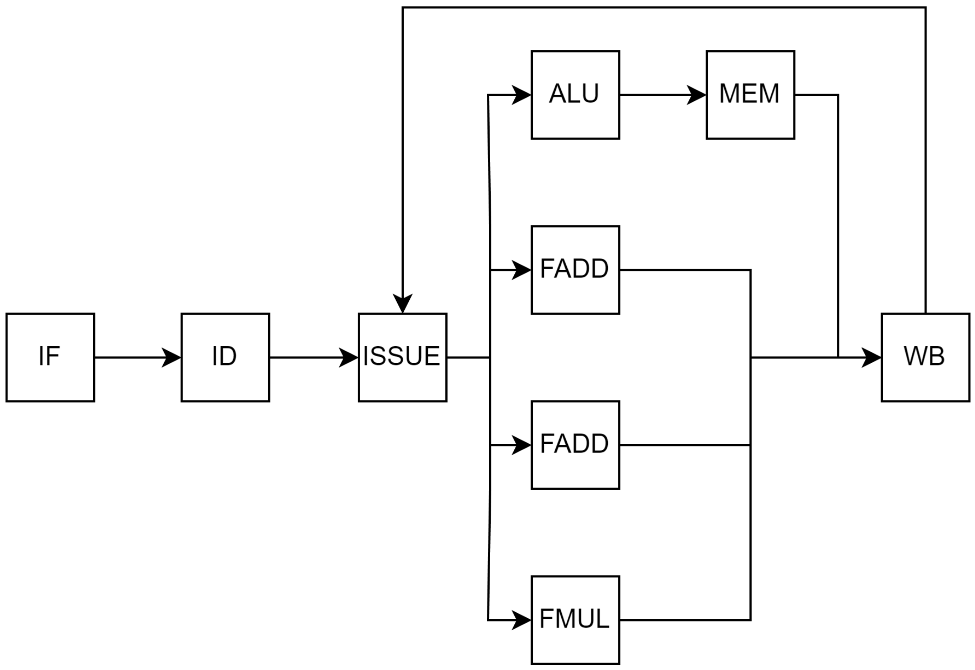
- T_{weak}, T_{weak}: Resulting in $1 + 2 \times \infty$ mispredictions for LOOP2 and 0 for LOOP.
- T_{strong}, T_{strong}: Resulting in $1 \times \infty$ mispredictions for LOOP2 and 0 for LOOP.

CHAPTER 3

Instruction-level parallelism

3.1 Exercise one

In this problem, we'll analyze the execution of a code segment on a single-issue out-of-order processor.



Consider now the code:

```
I1 lw.d $F3,B($R0)
I2 add.d $F2,$F2,$F3
I3 mul.d $F5,$F4,$F4
I4 addi $R0,$R0,8
I5 lw.d $F3,B($R0)
I6 add.d $F2,$F3,$F5
```

Examine each conflict taking into account the following operation durations:

- Arithmetic logic unit operations: one cycle.
- Memory operations: three cycles.
- Floating point addition: three cycles.
- Floating point multiplication: five cycles.

Solution

The conflicts are:

- RAW dependencies between instructions one and two, involving register $F3.
- WAR conflict between instruction one and instructions one and four, concerning register $R0.
- WAW conflict between instruction one and instructions one and five, affecting register $F3.
- WAR conflict between instruction two and instructions two and five, regarding register $F3.
- RAW dependency between instructions four and five, involving register $R0.
- RAW dependency between instructions three and six, involving register $F5.
- WAW conflict between instruction two and instructions two and six, impacting register $F2.
- RAW dependency between instructions five and six, involving register $F3.

A possible execution is:

Instruction	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**	**C16**	**C17**	**C18**	**C19**	**C20**
1	F	D	IS	E1	E2	E3	W													
2		F	D	$\underline{S}$	$\underline{S}$	$\underline{S}$	IS	E1	E2	E3	W									
3			F	$\underline{S}$	$\underline{S}$	$\underline{S}$	D	IS	E1	E2	E3	E4	E5	W						
4				$\underline{S}$	$\underline{S}$	$\underline{S}$	F	D	IS	E	$\underline{S}$	W								
5								F	$\underline{S}$	D	$\underline{S}$	IS	E1	E2	E3	W				
6									$\underline{S}$	F	$\underline{S}$	D	$\underline{S}$	$\underline{S}$	$\underline{S}$	IS	E1	E2	E3	W

If we assume that the issue stage is a buffer with an unlimited capacity to hold instructions awaiting execution:

Instruction	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**	**C16**	**C17**	**C18**	**C19**	**C20**
1	F	D	IS	E1	E2	E3	W													
2		F	D	$\underline{S}$	$\underline{S}$	$\underline{S}$	IS	E1	E2	E3	W									
3			F	D	$\underline{S}$	$\underline{S}$	$\underline{S}$	IS	E1	E2	E3	E4	E5	W						
4				D	$\underline{S}$	D	$\underline{S}$	$\underline{S}$	IS	E	$\underline{S}$	W								
5					$\underline{S}$	F	$\underline{S}$	$\underline{S}$	$\underline{S}$	D	$\underline{S}$	IS	E1	E2	E3	W				
6							$\underline{S}$	$\underline{S}$	$\underline{S}$	F	D	$\underline{S}$	$\underline{S}$	$\underline{S}$	$\underline{S}$	IS	E1	E2	E3	W

3.2 Exercise two

Assuming the following:

- All functional units are pipelined.
- ALU operations take 1 cycle.
- Memory operations take 3 cycles (includes time in ALU).
- Floating-point add instructions take 3 cycles.
- Floating-point multiply instructions take 5 cycles.
- There is no register renaming or forwarding.
- Instructions are fetched, decoded, and issued in order.
- The ISSUE stage is a buffer of unlimited length that holds instructions waiting to start execution.
- An instruction will only enter the issue stage if it does not cause a WAR or WAW hazard.
- Only one instruction can be issued at a time, and in the case of multiple instructions being ready, the oldest one will go first.
- Program Counter calculation for branches and jumps has been anticipated in the ISSUE stage.
- The target address for a branch is available in the FETCH stage.

Consider the code:

```
I1: FOR: ld $f2, VB($r6)
I2: fadd $f3, $f2, $f6
I3: st $f3, VA($r7)
I4: ld $f3, VC($r6)
I5: st $f3, VC($r7)
I6: fadd $f4, $f4, $f3
I7: addi $r6, $r6, 4
I8: addi $r7, $r7, 4
I9: blt $r7, $r8, FOR
```

Examine each conflict considering the operation durations:

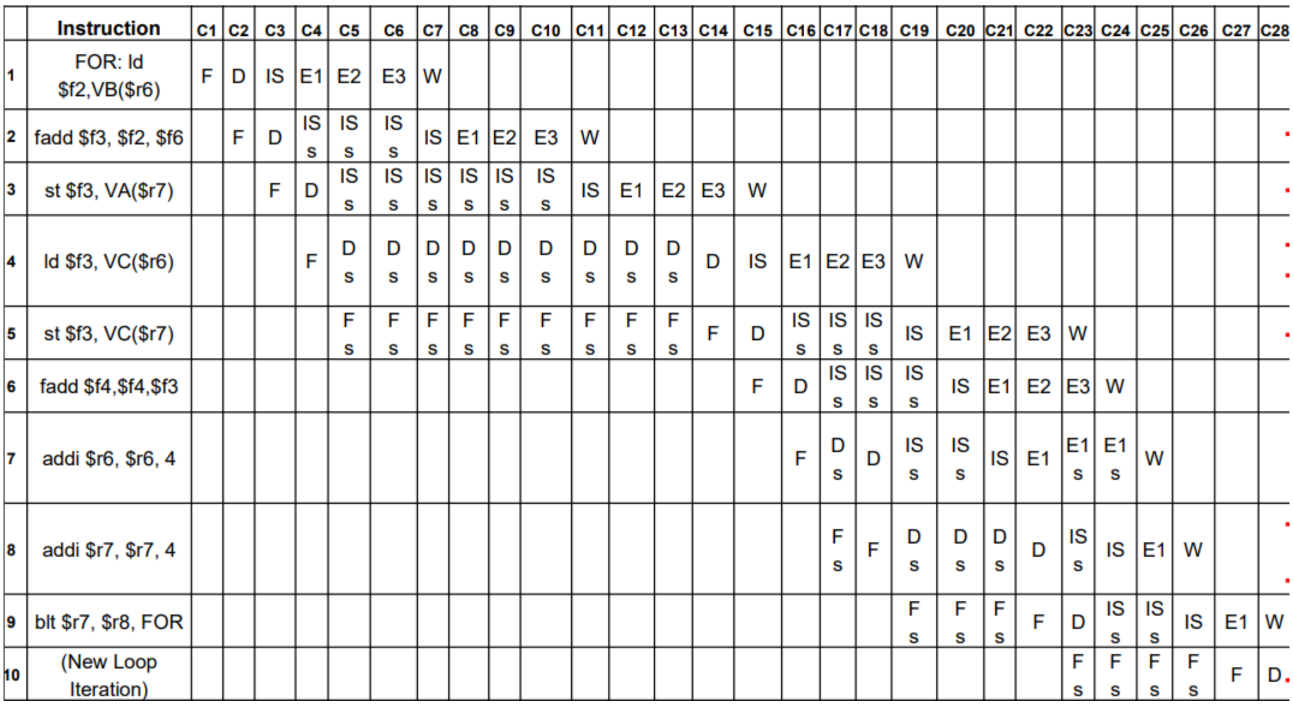
- Arithmetic logic unit operations: 1 cycle.
- Memory operations: 3 cycles.
- Floating-point addition: 3 cycles.
- Floating-point multiplication: 5 cycles.

Solution

The conflicts are:

- RAW: F2 (I1-I2), F3 (I2-I3), F3 (I4-I5), F3 (I4-I6), R7 (I8-I9).
- WAR: R7 (I8-I5, I8-I3), R6 (I7-I1, I7-I4).
- WAW: F3 (I2-I4).
- CNTRL.

The corresponding pipeline schema is:



	Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27	C28
1	FOR: ld $f2,VB($r6)	F	D	IS	E1	E2	E3	W																					
2	fadd $f3, $f2, $f6		F	D	IS s	IS s	IS s	IS	E1	E2	E3	W																	
3	st $f3, VA($r7)			F	D	IS s	IS s	IS s	IS s	IS s	IS s	IS	E1	E2	E3	W													
4	ld $f3, VC($r6)				F	D s	D s	D s	D s	D s	D s	D s	D s	D s	D	IS	E1	E2	E3	W									
5	st $f3, VC($r7)					F s	F s	F s	F s	F s	F s	F s	F s	F s	F	D	IS s	IS s	IS s	IS	E1	E2	E3	W					
6	fadd $f4,$f4,$f3															F	D	IS s	IS s	IS s	IS	E1	E2	E3	W				
7	addi $r6, $r6, 4																F	D s	D	IS s	IS s	IS	E1	E1 s	E1 s	W			
8	addi $r7, $r7, 4																	F s	F	D s	D s	D s	D	IS s	IS	E1	W		
9	blt $r7, $r8, FOR																			F s	F s	F s	F	D	IS s	IS s	IS	E1	W
10	(New Loop Iteration)																							F s	F s	F s	F s	F	D

3.3 Exercise three

Consider the following assembly program:

```
LW      $1, OFF($2)
ADDI    $3, $1, 4
SUB     $4, $1, $2
ADDI    $2, $1, -8
SW      $4, OFF($2)
```

No optimizations are applied in the MIPS pipeline. The processor operates with a clock cycle of 2 ns.

- Draw the pipeline schema and highlight potential hazards.
- Illustrate the actual execution with stall cycles inserted.
- Calculate Instruction Count (IC), CPI, and MIPS.

Solution

1. The pipeline schema is:

Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
LW \$1, OFF (\$2)	F	D	E	M	W				
FDDI \$3, \$1, 4		F	D	E	M	W			
SUB \$4, \$1, \$3			F	D	E	M	W		
ADDI \$2, \$1, -8				F	D	E	M	W	
SW \$5, OFF (\$2)					F	D	E	M	W

   The potential hazards are:

   - Instruction 1 writes to register \$1, and instructions 2, 3, and 4 read from it.
   - Instruction 2 writes to register \$3, and instruction 3 reads from it.
   - Instruction 4 writes to register \$2, and instruction 5 reads from it.

2. The real execution with stall cycles inserted is:

Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
LW \$1, OFF (\$2)	F	D	E	M	W										
ADDI \$3, \$1, 4		F	$\underline{\text{S}}$	$\underline{\text{S}}$	D	E	M	W							
SUB \$4, \$1, \$3					F	$\underline{\text{S}}$	$\underline{\text{S}}$	D	E	M	W				
ADDI \$2, \$1, -8								F	D	E	M	W			
SW \$5, OFF (\$2)									F	$\underline{\text{S}}$	$\underline{\text{S}}$	D	E	M	W

3. The performance metrics are:

$$\text{IC} = 5$$

$$\text{CPI} = \frac{\text{CCs}}{\text{IC}} = \frac{15}{5} = 3$$

$$\text{MIPS} = \frac{\text{clock frequency}}{\text{CPI} \cdot 10^6} = \frac{0.5 \cdot 10^9}{3 \cdot 10^6} = 166$$

3.4 Exercise four

Consider the given program:

```
i1: add $t1, $t0, $t1
i2: add $t2, $t1, $t2
i3: subi $t0, $t2, 1
i4: sw $t0, 0x00BB($t2)
i5: beq $t0, $t2, 0x0089
```

Assuming no forwarding, register file access with read/write optimization, and control hazards solved in the instruction decode stage (ID):

1. Define all conflicts/dependencies and analyze whether they cause hazards and the theoretical amount of stalls.

2. Draw the effective pipeline schema:
3. Draw the effective pipeline schema assuming EX/EX, MEM/EX, and MEM/MEM forwarding paths are available.
4. Draw the effective pipeline schema assuming that the previous forwarding paths with also EX/ID are available.

Solution

1. The potential issues are:

Instruction number	**Instruction dependency**	**Register involved**	**Hazard**	**Stalls**
i2	i1	\$t1	yes	2
i3	i2	\$t2	yes	2
i4	i2	\$t2	yes	1
i4	i3	\$t0	yes	2
i5	i2	\$t2	no	0
i5	i3	\$t0	yes	1

2. The requested pipeline schema with stalls is as follows:

Instruction	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**
add \$t1, \$t0, \$t1	IF	ID	EX	M	WB										
add \$t2, \$t1, \$t2		IF	$\underline{S}$	$\underline{S}$	ID	EX	M	WB							
subi \$t0, \$t2, 1			$\underline{S}$	$\underline{S}$	IF	$\underline{S}$	$\underline{S}$	ID	EX	M	WB				
sw \$t0, 0×00BB (\$t2)						$\underline{S}$	$\underline{S}$	IF	$\underline{S}$	$\underline{S}$	ID	EX	M	WB	
beq \$t0, \$t2, 0×0089									$\underline{S}$	$\underline{S}$	IF	ID	EX	M	WB

3. The requested pipeline schema with forwarding is as follows:

Instruction	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**
add \$t1, \$t0, \$t1	IF	ID	EX	M	WB					
add \$t2, \$t1, \$t2		IF	ID	EX	M	WB				
subi \$t0, \$t2, 1			IF	ID	EX	M	WB			
sw \$t0, 0×00BB (\$t2)				IF	ID	EX	M	WB		
beq \$t0, \$t2, 0×0089					IF	$\underline{S}$	ID	EX	M	WB

4. The requested pipeline schema with forwarding and EX/ID is:

Instruction	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**
add \$t1, \$t0, \$t1	IF	ID	EX	M	WB				
add \$t2, \$t1, \$t2		IF	ID	EX	M	WB			
subi \$t0, \$t2, 1			IF	ID	EX	M	WB		
sw \$t0, 0×00BB (\$t2)				IF	ID	EX	M	WB	
beq \$t0, \$t2, 0×0089					IF	ID	EX	M	WB

3.5 Exercise five

Assuming the following:

- All functional units are pipelined.
- ALU operations take 1 cycle.
- Memory operations take 2 cycles (includes time in ALU).
- Floating-point add instructions take 2 cycles.
- Floating-point multiply instructions take 3 cycles.
- There is no register renaming. No forwarding.
- Instructions are fetched, decoded and issued in order.
- The ISSUE stage is a buffer of unlimited length that holds instructions waiting to start execution.
- An instruction will only enter the issue stage if it does not cause a WAR or WAW hazard.
- Only one instruction can be issued at a time, and in the case multiple instructions are ready, the oldest one will go first.
- Program Counter calculation for branches and jumps has been anticipated in the ISSUE stage.

Consider the code:

```
LOOP:   I1: LD F1, 0 (R2)
        I2: MULTD F2, F1, F1
        I3: ADDD F3, F1, F5
        I4: MULTD F2, F3, F1
        I5: SUBD F5, F1, F5
        I6: SUBI R2, R2, 4
        I7: BNEZ R2, LOOP
```

Find the conflicts.

Solution

The conflicts in the given problem are:

- RAW F1 I1-I2
- RAW F1 I1-I3
- RAW F1 I1-I4
- RAW F1 I1-I5
- RAW F3 I3-I4

- RAW R2 I6-I7
- WAW F2 I2-I4
- WAR F5 I3-I5
- WAR R2 I1-I6
- CNTRL

3.6 Exercise six

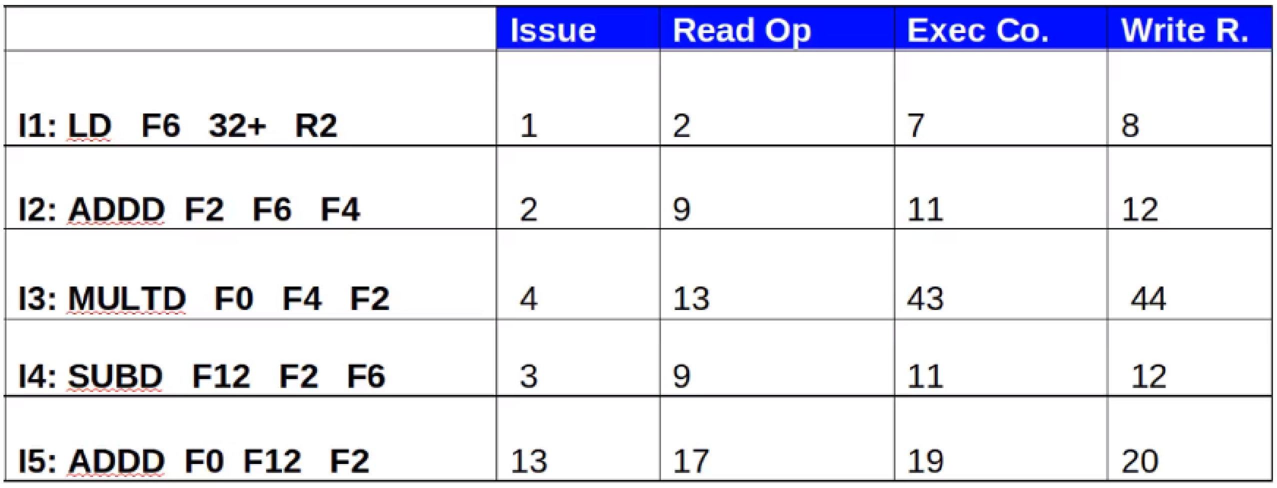
Consider the following code:

```
I1: LD F6 32+ R2
I2: ADDD F2 F6 F4
I3: MULTD F0 F4 F2
I4: SUBD F12 F2 F6
I5: ADDD F0 F12 F2
```

1. Identify all conflicts.
2. Develop a scoreboard for the given code, indicating clock cycles.
3. If the previous table was incorrect, provide the accurate one, specifying the number, type, and latency for each unit.

Solution

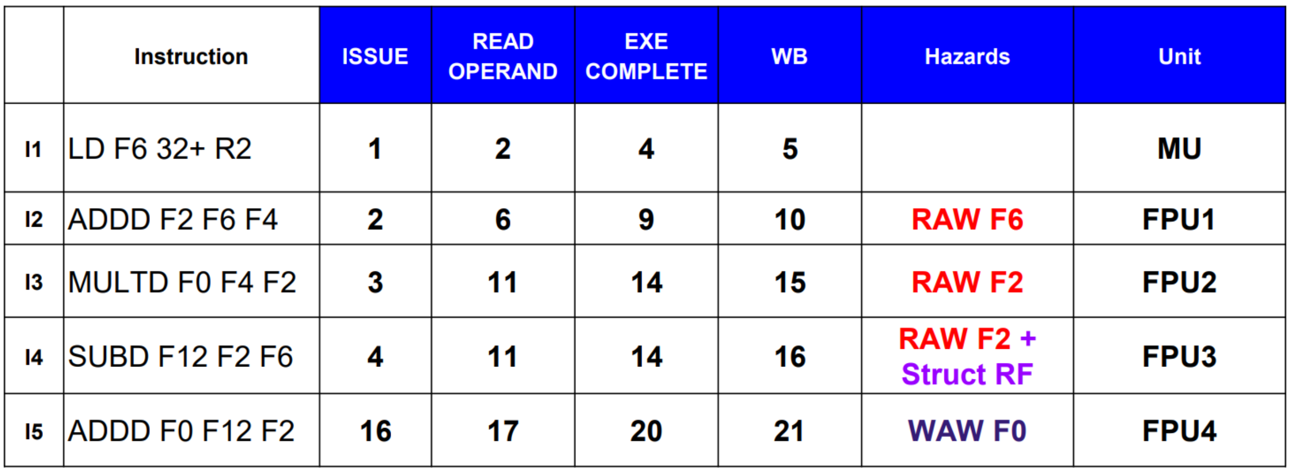
1. Conflicts:
    - RAW F6 between I1 and I2
    - RAW F2 between I2 and I3
    - RAW F2 between I2 and I4
    - RAW F6 between I1 and I4
    - RAW F2 between I2 and I5
    - WAW F0 between I3 and I5
    - RAW F12 between I4 and I5
2. Scoreboard:



	Issue	Read Op	Exec Co.	Write R.
I1: LD F6 32+ R2	1	2	7	8
I2: ADDD F2 F6 F4	2	9	11	12
I3: MULTD F0 F4 F2	4	13	43	44
I4: SUBD F12 F2 F6	3	9	11	12
I5: ADDD F0 F12 F2	13	17	19	20

Although two write results occur simultaneously, this is permissible in a scoreboard-based architecture. However, the issue order is incorrect, rendering the scoreboard configuration inaccurate.

3. Correct Scoreboard:



	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD F6 32+ R2	1	2	4	5		MU
I2	ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
I3	MULTD F0 F4 F2	3	11	14	15	RAW F2	FPU2
I4	SUBD F12 F2 F6	4	11	14	16	RAW F2 + Struct RF	FPU3
I5	ADDD F0 F12 F2	16	17	20	21	WAW F0	FPU4

3.7 Exercise seven

Consider the following code:

```
I1: LD F6 32+ R2
I2: ADDD F2 F6 F4
I3: MULTD F0 F4 F2
I4: SUBD F12 F2 F6
I5: ADDD F0 F12 F2
```

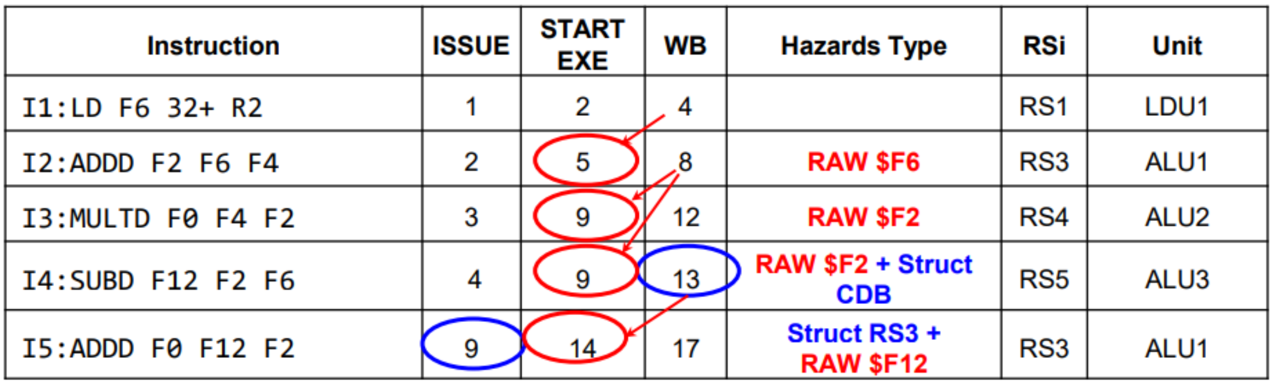
We have:

- 2 RESERVATION STATIONS (RS1, RS2) + 1 LOAD/STORE unit (LDU1) with latency 2
- 3 RESERVATION STATIONS (RS3, RS4, RS5) + 3 ALU/BR FUs (ALU1, ALU2, ALU3) with latency 3

1. Find all the conflicts.
2. Apply the Tomasulo algorithm.

Solution

1. The conflicts are the following:
   - RAW F6 I1-I2
   - RAW F2 I2-I3
   - RAW F2 I2-I4
   - RAW F6 I1-I4
   - RAW F2 I2-I5
   - WAW F0 I3-I5
   - RAW F12 I4-I5
2. The final result of the Tomasulo algorithm is:



Instruction	ISSUE	START EXE	WB	Hazards Type	RSi	Unit
I1:LD F6 32+ R2	1	2	4		RS1	LDU1
I2:ADDD F2 F6 F4	2	5	8	RAW $F6	RS3	ALU1
I3:MULTD F0 F4 F2	3	9	12	RAW $F2	RS4	ALU2
I4:SUBD F12 F2 F6	4	9	13	RAW $F2 + Struct CDB	RS5	ALU3
I5:ADDD F0 F12 F2	9	14	17	Struct RS3 + RAW $F12	RS3	ALU1

3.8 Exercise eight

Consider the following code:

```
I1: LD $F1, 0($R1)
I2: FADD $F2, $F2, $F3
I3: ADDI $R3, $R3, 8
I4: LD $F4, 0(R2)
I5: FADD $F5, $F4, $F2
I6: FMULT $F6, $F1, $F4
I7: ADDI $R5, $R5, 1
I8: LD $R6, 0($R4)
I9: SD $F6, 0($R5)
I10: SD $F5, 0($R6)
```

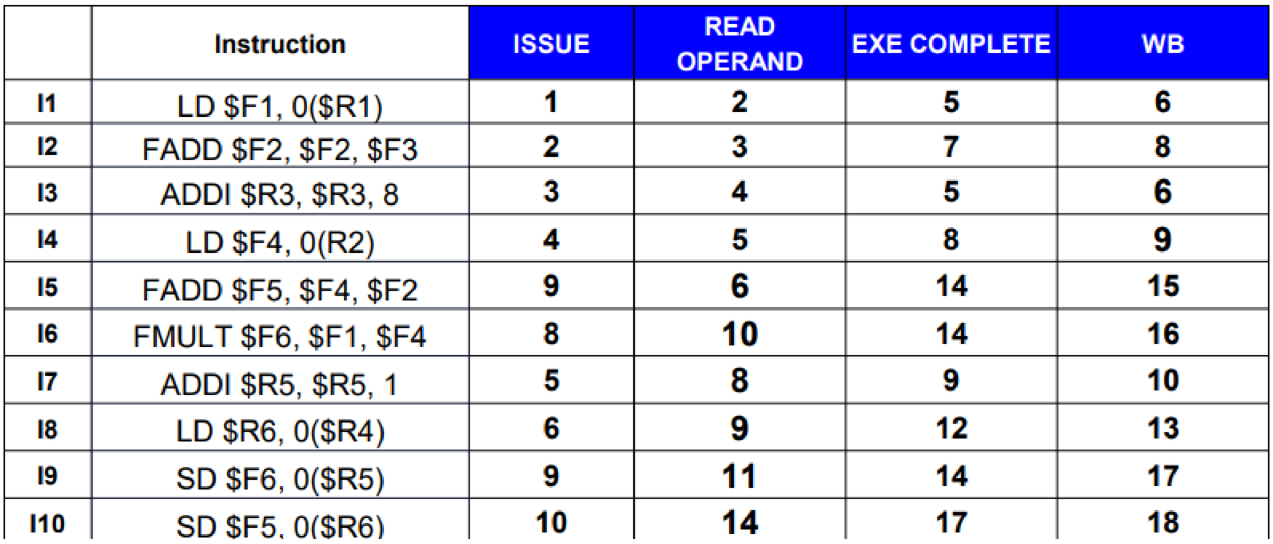
1. Identify all conflicts.
2. Develop a scoreboard for the given code, indicating clock cycles.
3. If the previous table was incorrect, provide the accurate one, specifying the number, type, and latency for each unit.

Solution

1. Conflicts:

    - RAW F6 between I1 and I2
    - RAW F2 between I2 and I3
    - RAW F2 between I2 and I4
    - RAW F6 between I1 and I4
    - RAW F2 between I2 and I5
    - WAW F0 between I3 and I5
    - RAW F12 between I4 and I5

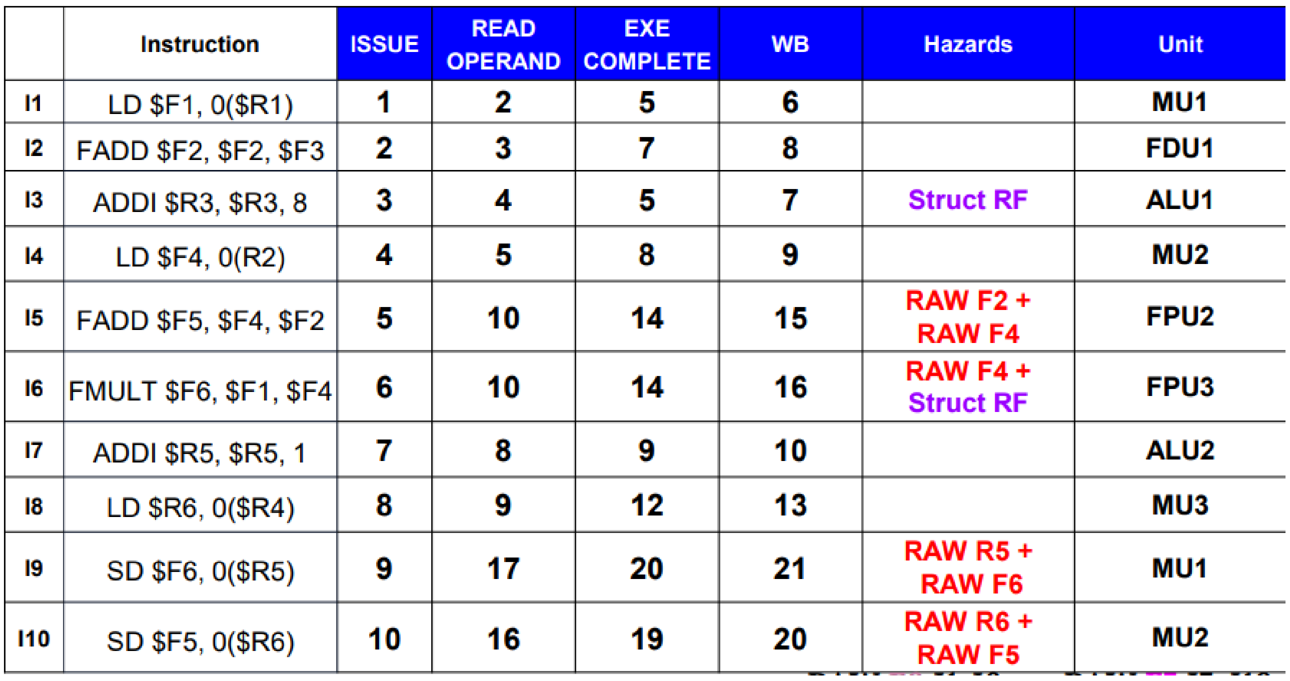
2. Scoreboard:



	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB
I1	LD $F1, 0($R1)	1	2	5	6
I2	FADD $F2, $F2, $F3	2	3	7	8
I3	ADDI $R3, $R3, 8	3	4	5	6
I4	LD $F4, 0(R2)	4	5	8	9
I5	FADD $F5, $F4, $F2	9	6	14	15
I6	FMULT $F6, $F1, $F4	8	10	14	16
I7	ADDI $R5, $R5, 1	5	8	9	10
I8	LD $R6, 0($R4)	6	9	12	13
I9	SD $F6, 0($R5)	9	11	14	17
I10	SD $F5, 0($R6)	10	14	17	18

The issue order is incorrect, rendering the scoreboard configuration inaccurate.

3. Correct Scoreboard:



	Instruction	ISSUE	READ OPERAND	EXE COMPLETE	WB	Hazards	Unit
I1	LD $F1, 0($R1)	1	2	5	6		MU1
I2	FADD $F2, $F2, $F3	2	3	7	8		FDU1
I3	ADDI $R3, $R3, 8	3	4	5	7	Struct RF	ALU1
I4	LD $F4, 0(R2)	4	5	8	9		MU2
I5	FADD $F5, $F4, $F2	5	10	14	15	RAW F2 + RAW F4	FPU2
I6	FMULT $F6, $F1, $F4	6	10	14	16	RAW F4 + Struct RF	FPU3
I7	ADDI $R5, $R5, 1	7	8	9	10		ALU2
I8	LD $R6, 0($R4)	8	9	12	13		MU3
I9	SD $F6, 0($R5)	9	17	20	21	RAW R5 + RAW F6	MU1
I10	SD $F5, 0($R6)	10	16	19	20	RAW R6 + RAW F5	MU2