Advanced Computer Architectures
Exercises

Christian Rossi

Academic Year 2023-2024

Abstract

The course covers several key topics in computer architecture. It begins with a review of fundamental concepts, such as the RISC approach, pipelining, and the memory hierarchy. Students will learn about performance evaluation metrics for computer architectures and techniques for optimizing performance in both processors and memory. The course delves into instruction-level parallelism, discussing static and dynamic scheduling, superscalar architectures, their principles and challenges, and VLIW architectures, with examples from various architecture families. Additionally, the course covers thread-level parallelism and explores multiprocessor and multicore systems, including their taxonomy, topologies, communication management, memory management, and cache coherency protocols, with examples of different architectures. Finally, the course examines stream processors, vector processors, graphics processors, GP-GPUs, and heterogeneous architectures.

Contents

CHAPTER 1

Performance evaluation

1.1 Exercise 1

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 2

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 3

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:

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

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

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

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 1

Explain the design of a 1-BHT and a 2-BHT capable of executing the provided assembly code (with R0 set to 2000 and R1 set to 0).

```
1 LOOP:   LD F1 0 R0
2         ADDD F2 F1 F1
3         ADDI R1 R1 100
4 LOOP2:  MULTD F2 F2 F1
5         SUBI R1 R1 1
6         BNEZ R1 LOOP2
7         SUBI R0 R0 2
8         BNEZ R0 LOOP
```

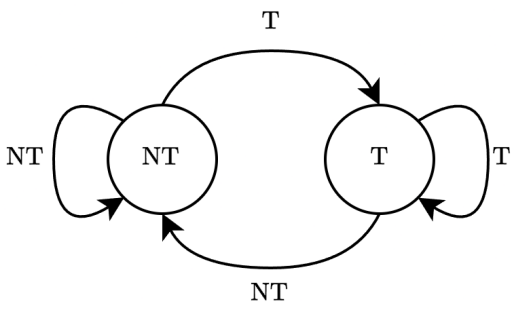
Compute the number of mispredictions in the various cases.

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:



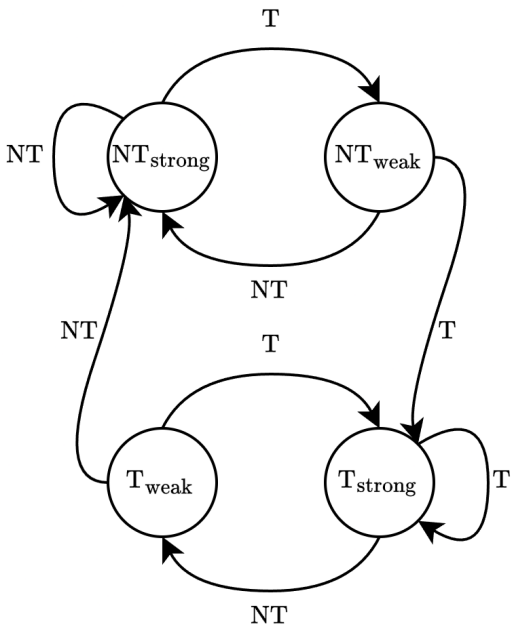
In the absence of collision, one bit can be assigned for each loop. This allows four possible initializations: T-T, T-NT, NT-T, NT-NT.

- 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:

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

The best cases are:

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

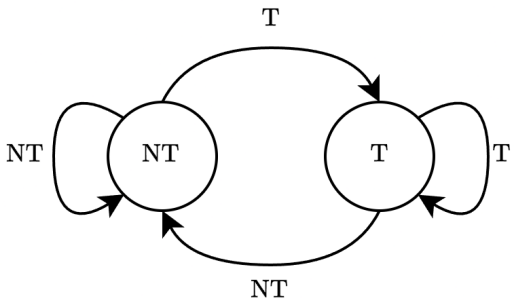
Describe a 1-BHT and a 2-BHT able to execute the following assembly code (R0 is set to 1, R1 is set to 300).

```
1 LOOP:   LD F3 0 (R0)
2         ADDD F1 F3 F3
3         ADDI R1 R1 3000
4 LOOP2:  MULTD F2 F2 F3
5         SUBI R1 R1 3
6         BNEZ R1 LOOP2
7         SUBI R0 R0 2
8         BNEZ R0 LOOP
```

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:



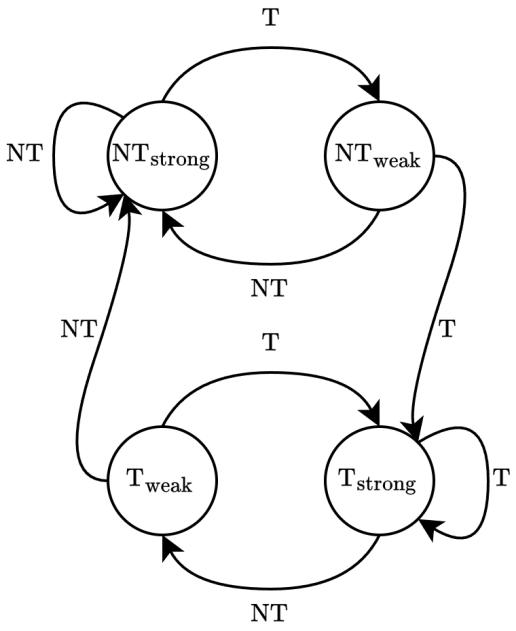
In the absence of collision, one bit can be assigned for each LOOP. This allows four possible initializations: T-T, T-NT, NT-T, NT-NT.

- For the first case (T-T), there are NO misprediction for LOOP, and for LOOP2 a single misprediction at the end of LOOP2 and 2 misprediction per iteration begin and end of LOOP.
- For the second case (T-NT), there are NO misprediction for LOOP, and for LOOP2 two misprection: beginning and end of LOOP.
- For the third case (NT-T), there are only initial misprediction for LOOP, and for LOOP2 single misprediction at the end of LOOP2 and 2 misprediction per iteration begin and end of LOOP.
- For the fourth case (NT-NT), there are only initial misprediction for LOOP, and for LOOP2 two misprection: beginning and end of 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 LOOP2, and a 100% failure rate for LOOP.
- For the second case (NT), there are two initial misprection, then end of 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:

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

The best cases are:

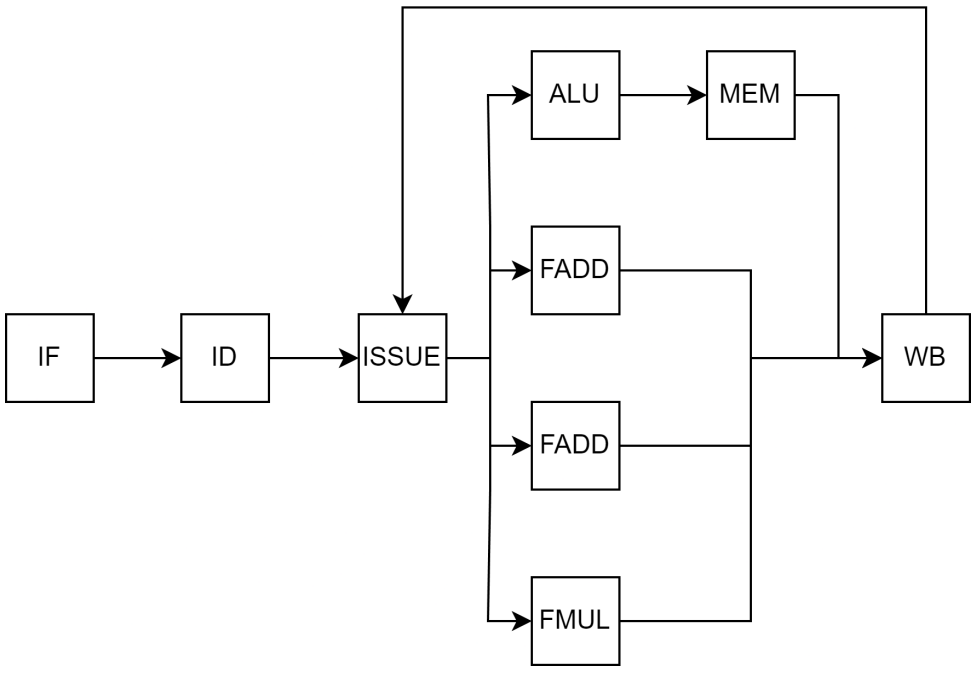
- $\mathrm{T}_{\mathrm{weak}}, \mathrm{T}_{\mathrm{weak}}$: Resulting in $1 + 2 \times \infty$ mispredictions for LOOP2 and 0 for LOOP.
- $\mathrm{T}_{\mathrm{strong}}, \mathrm{T}_{\mathrm{strong}}$: Resulting in $1 \times \infty$ mispredictions for LOOP2 and 0 for LOOP.

CHAPTER 3

Pipelining

3.1 Exercise 1

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



Consider now the code:

```
1 LW F3,B(R0)
2 ADD F2,F2,F3
3 MUL F5,F4,F4
4 ADDI R0,R0,8
5 LW F3,B(R0)
6 ADD 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 R0 I4-I5.
- RAW F3 I5-I6.
- RAW F5 I3-I6.
- RAW F3 I1-I2.
- WAW F2 I2-I6.
- WAW F3 I1-I5.
- WAR F3 I2-I5.
- WAR R0 I1-I4.

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	S	S	S	IS	E1	E2	E3	W									
3			F	S	S	S	D	IS	E1	E2	E3	E4	E5	W						
4				S	S	S	F	D	IS	E	S	W								
5								F	S	D	S	IS	E1	E2	E3	W				
6									S	F	S	D	S	S	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	S	S	S	IS	E1	E2	E3	W									
3			F	D	S	S	S	IS	E1	E2	E3	E4	E5	W						
4				D	S	D	S	S	IS	E	S	W								
5					S	F	S	S	S	D	S	IS	E1	E2	E3	W				
6							S	S	S	F	D	S	S	S	S	IS	E1	E2	E3	W

3.2 Exercise 2

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:

```
1 FOR:    LD F2, VB(R6)
2         FADD F3, F2, F6
3         ST F3, VA(R7)
4         LD F3, VC(R6)
5         ST F3, VC(R7)
6         FADD F4, F4, F3
7         ADDI R6, R6, 4
8         ADDI R7, R7, 4
9         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.
- RAW F3 I2-I3.
- RAW F3 I4-I5.

- RAW F3 I4-I6.
- RAW R7 I8-I9.
- WAW F3 I2-I4.
- WAR R7 I8-I5.
- WAR R7 I8-I3.
- WAR R6 I7-I1.
- WAR R6 I7-I4.
- CNTRL.

The corresponding pipeline schema is:

Instruction	**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**
LD F2, VB(R6)	F	D	IS	E1	E2	E3	W																					
FADD F3, F2, F6		F	D	S	S	S	IS	E1	E2	E3	W																	
ST F3, VA(R7)			F	D	S	S	S	S	S	S	IS	E1	E2	E3	W													
LD F3, VC(R6)				F	S	S	S	S	S	S	S	S	S	D	IS	E1	E2	E3	W									
ST F3, VC(R7)														F	D	S	S	S	IS	E1	E2	E3	W					
FADD F4, F4, F3															F	S	D	S	S	IS	E1	S	S	W				
ADDI R6, R6, 4																S	F	S	S	S	D	S	IS	E1	W			
ADDI R7, R7, 4																	S	F	S	S	S	D	S	IS	E1	W		
BLT R7, R8, FOR																			S	S	S	F	D	S	S	IS	E1	W
CNTRL																							S	S	S	S	IF	ID

3.3 Exercise 3

Consider the following assembly program:

```
1 LW 1, OFF(2)
2 ADDI 3, 1, 4
3 SUB 4, 1, 2
4 ADDI 2, 1, -8
5 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				
FADDI 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	S	S	D	E	M	W							
SUB 4, 1, 3					F	S	S	D	E	M	W				
ADDI 2, 1, -8								F	D	E	M	W			
SW 5, OFF (2)									F	S	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 4

Consider the given program:

```
1 ADD T1, T0, T1
2 ADD T2, T1, T2
3 SUBI T0, T2, 1
4 SW T0, 0x00BB(T2)
5 BEQ T0, T2, 0x0089
```

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

1. Define all 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	S	S	ID	EX	M	WB							
SUBI T0, T2, 1			S	S	IF	S	S	ID	EX	M	WB				
SW T0, 0×00BB (T2)						S	S	IF	S	S	ID	EX	M	WB	
BEQ T0, T2, 0×0089									S	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	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 5

Consider now the code:

```
1 ADDI S3, S2, 2
2 ADD S5, S4, S3
3 SW S5, 4(S3)
4 SUB S7, S5, S6
5 LS S6, 4(S7)
```

1. Find the conflicts.
2. Simple Pipelining.
3. Reschedule the instructions to reduce the stalls; draw the pipeline schema showing all the data dependencies.

Solution

1. The conflicts are:
   - RAW S3 I1-I2.
   - RAW S7 I4-I5.
   - RAW S5 I2-I3.
   - RAW S3 I1-I3.
   - RAW S5 I2-I4.
2. With simple pipelining we have:

Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
ADDI S3, S2, 2	IF	ID	EX	MEM	WB										
ADD S5, S4, S3		IF	S	S	ID	EX	MEM	WB							
SW S5, 4(S3)			S	S	IF	S	S	ID	EX	MEM	WB				
SUB S7, S5, S6						S	S	IF	ID	EX	MEM	WB			
LS S6, 4(S7)									IF	S	S	ID	EX	MEM	WB

3. By using forwarding paths we have:

Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Forwarding path**
ADDI S3, S2, 2	IF	ID	EX	MEM	WB					
ADD S5, S4, S3		IF	ID	EX	MEM	WB				EX-EX
SW S5, 4(S3)			IF	ID	EX	MEM	WB			MEM-EX, MEM-MEM
SUB S7, S5, S6				IF	ID	EX	MEM	WB		MEM-EX
LS S6, 4(S7)					IF	ID	EX	MEM	WB	EX-EX

3.6 Exercise 6

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:

```
1 LOOP:   LD F1, 0 (R2)
2         MULTD F2, F1, F1
3         ADDD F3, F1, F5
4         MULTD F2, F3, F1
5         SUBD F5, F1, F5
6         SUBI R2, R2, 4
7         BNEZ R2, LOOP
```

Examine each conflict considering the operation durations:

- ALU OP: 1 cycle
- MEM OP: 2 cycles
- FP ADD: 2 cycles
- FP MULT: 3 cycles

1. Find the conflicts.
2. Compute the pipeline schema.

Solution

1. The conflicts 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.

2. The pipeline schema is the following:

Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
LD F1, 0 (R2)	F	D	IS	E1	E2	W													
MULTD F2, F1, F1		F	D	S	S	IS	E1	E2	E3	W									
ADDD F3, F1, F5			F	D	S	S	IS	E1	E2	S	W								
MULTD F2, F3, F1				F	S	S	S	S	D	S	IS	E1	E2	E3	W				
SUBD F5, F1, F5					S	S	S	S	F	D	S	IS	E1	E2	S	W			
SUBI R2, R2, 4										F	D	S	IS	E1	S	S	W		
BNEZ R2, LOOP											F	D	S	S	S	S	IS	E1	W
CNTRL												S	S	S	S	S	S	F	D

CHAPTER 4

Scoreboard

4.1 Exercise 1

Consider the following code:

```
1 LD F6 32+ R2
2 ADDD F2 F6 F4
3 MULTD F0 F4 F2
4 SUBD F12 F2 F6
5 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 I1-I2.
   - RAW F2 I2-I3.
   - RAW F2 I2-I4.
   - RAW F6 I1-I4.
   - RAW F2 I2-I5.
   - RAW F12 I4-I5.
   - WAW F0 I3-I5.
2. Scoreboard:

Instruction	Issue	Read	Execute	Write
LD F6 32+ R2	1	2	7	8
ADDD F2 F6 F4	2	9	11	12
MULTD F0 F4 F2	4	13	43	44
SUBD F12 F2 F6	3	9	11	12
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	Execution	Write	Hazards	Unit
LD F6 32+ R2	1	2	4	5		MU
ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
MULTD F0 F4 F2	3	11	14	15	RAW F2	FPU2
SUBD F12 F2 F6	4	11	14	16	RAW F2 Struct RF	FPU3
ADDD F0 F12 F2	16	17	20	21	WAW F0	FPU4

4.2 Exercise 2

Consider the following code:

```
1  LD F1, 0(R1)
2  FADD F2, F2, F3
3  ADDI R3, R3, 8
4  LD F4, 0(R2)
5  FADD F5, F4, F2
6  FMULT F6, F1, F4
7  ADDI R5, R5, 1
8  LD R6, 0(R4)
9  SD F6, 0(R5)
10 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 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. Scoreboard:

Instruction	Issue	Read	Execution	Write
LD F1, 0(R1)	1	2	5	6
FADD F2, F2, F3	2	3	7	8
ADDI R3, R3, 8	3	4	5	6
LD F4, 0(R2)	4	5	8	9
FADD F5, F4, F2	9	6	14	15
FMULT F6, F1, F4	8	10	14	16
ADDI R5, R5, 1	5	8	9	10
LD R6, 0(R4)	6	9	12	13
SD F6, 0(R5)	9	11	14	17
SD F5, 0(R6)	10	14	17	18

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

3. Correct Scoreboard:

Instruction	Issue	Read	Execution	Write	Hazards	Unit
LD F1, 0(R1)	1	2	5	6		MU1
FADD F2, F2, F3	2	3	7	8		FDU1
ADDI R3, R3, 8	3	4	5	7	Struct RF	ALU1
LD F4, 0(R2)	4	5	8	9		MU2
FADD F5, F4, F2	5	10	14	15	RAW F2 RAW F4	FPU2
FMULT F6, F1, F4	6	10	14	16	RAW F4 Struct RF	FPU3
ADDI R5, R5, 1	7	8	9	10		ALU2
LD R6, 0(R4)	8	9	12	13		MU3
SD F6, 0(R5)	9	17	20	21	RAW F5 RAW F6	MU1
SD F5, 0(R6)	10	16	19	20	RAW F6 RAW F5	MU2

4.3 Exercise 3

Consider the following code:

```
1 LD F6 32+ R2
2 ADDD F2 F6 F4
3 MULTD F0 F4 F2
4 SUBD F12 F2 F6
5 ADDD F0 F12 F2
```

We have that 4 Floating Point ALU with 3 clock cycles latency, a single write port for the pool, and 1 Memory unit with 2 clock cycles latency.

1. Identify all conflicts.
2. Develop a scoreboard for the given code, indicating clock cycles and using Register Renaming.

Solution

1. Conflicts:
   - 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. Scoreboard:

Instruction	Issue	Read	Execution	Write	Hazards	Unit
LD F6 32+ R2	1	2	4	5		MU
ADDD F2 F6 F4	2	6	9	10	RAW F6	FPU1
MULTD F0 F4 F2	3	11	14	15	RAW F2	FPU2
SUBD F12 F2 F6	4	11	14	16	RAW F2 RAW F6 Struct RF	FPU3
ADDD F0 F12 F2	5	17	20	21	RAW F2 RAW F12	FPU4

4.4 Exercise 4

Consider the following code:

```
1 LW F1, 0(R0)
2 FADDI F1, F1, C1
3 FADDI F2, F1, C2
4 SW F2, 0(R0)
5 LW F2, 4(R0)
6 FADD F2, F2, F2
7 SW F2, 4(R0)
```

We have that 4 Floating Point ALU with 2 clock cycles latency, single write port for the pool, 4 Load Store unit with 4 clock cycles latency.

1. Identify all conflicts.

2. Develop a scoreboard for the given code, indicating clock cycles.
3. Develop a scoreboard for the given code, indicating clock cycles and with Register Renaming.

COnsider a cache miss for the first instruction with a penalty of 2 clock cycles and a cache miss for the fifth instruction with a penalty of 5 clock cycles.

Solution

1. Conflicts:
   - RAW F1 I1-I2.
   - RAW F1 I2-I3.
   - RAW F2 I3-I4.
   - RAW F2 I6-I7.
   - RAW F2 I5-I6.
   - WAW F2 I5-I6.
   - WAW F2 I5-I3.
   - WAW F1 I2-I1.
   - WAW F2 I6-I3.
   - WAR F2 I4-I5.
   - WAR F2 I4-I6.
2. Scoreboard:

Instruction	**Issue**	**Read**	**Execution**	**Write**	**Unit**
LW F1, 0(R0)	1	2	8	9	LDU1
FADDI F1, F1, C1	10	11	13	14	FPU1
FADDI F2, F1, C2	11	15	17	18	FPU2
SW F2, 0(R0)	12	19	13	24	LDU2
LW F2, 4(R0)	19	20	29	30	LDU3
FADD F2, F2, F2	31	32	34	35	FPU3
SW F2, 4(R0)	32	36	40	41	LDU4

3. Scoreboard with Register Renaming:

Instruction	**Issue**	**Read**	**Execution**	**Write**	**Unit**
LW F1, 0(R0)	1	2	8	9	LDU1
FADDI F1, F1, C1	2	10	12	13	FPU1
FADDI F2, F1, C2	3	14	16	17	FPU2
SW F2, 0(R0)	4	18	22	23	LDU2
LW F2, 4(R0)	5	6	15	16	LDU3
FADD F2, F2, F2	6	20	22	23	FPU3
SW F2, 4(R0)	7	24	28	29	LDU4

CHAPTER 5

Tomasulo

5.1 Exercise 1

Consider the following code:

```
1 LD F6 32+ R2
2 ADDD F2 F6 F4
3 MULTD F0 F4 F2
4 SUBD F12 F2 F6
5 ADDD F0 F12 F2
```

We have:

- 2 Reservation Stations (RS1, RS2) with 1 Load Store unit (LDU1) with 2 clock cycles latency.
- 3 Reservation Stations (RS3, RS4, RS5) with 3 ALU/BR FUs (ALU1, ALU2, ALU3) with latency 3 clock cycles latency.

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.
    - RAW F12 I4-I5.
    - WAW F0 I3-I5.

2. The final result of the Tomasulo algorithm is:

Instruction	Issue	Execution	Write	Hazards	RS	Unit
LD F6 32+ R2	1	2	4		RS1	LDU1
ADDD F2 F6 F4	2	5	8	RAW F6	RS3	ALU1
MULTD F0 F4 F2	3	9	12	RAW F2	RS4	ALU2
SUBD F12 F2 F6	4	9	13	RAW F2 Struct CDB	RS5	ALU3
ADDD F0 F12 F2	9	14	17	Struct RS3 RAW F12	RS3	ALU1

5.2 Exercise 2

Consider the following code:

```
1  LD F1, 0(R1)
2  FADD F2, F2, F3
3  ADDI R3, R3, 8
4  LD F4, 0(R2)
5  FADD F5, F4, F2
6  FMULT F6, F1, F4
7  ADDI R5, R5, 1
8  LD R6, 0(R4)
9  SD F6, 0(R5)
10 SD F5, 0(R6)
```

We have:

- 3 Reservation Stations (RS1, RS2, RS3) with 3 Load Store unit (LDU1, LDU2, LDU3) with 3 clock cycles latency.
- 3 Reservation Stations (RS4, RS5, RS6) with 3 FPUs (FPU1, FPU2, FPU3) with 3 clock cycles latency.
- 2 Reservation Stations (RS7, RS8) with 1 Integer ALU (ALU1) with 1 clock cycles latency.

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

Solution

1. The conflicts are the following:
   - RAW F0 I1-I6.
   - RAW F2 I2-I5.
   - RAW F4 I4-I5.
   - RAW F4 I4-I6.

- RAW F6 I6-I9.
- RAW F5 I5-I10.
- RAW R5 I7-I9.
- RAW R6 I8-I10.

2. The final result of Tomasulo is:

Instruction	Issue	Execution	Write	Hazards	RS	Unit
LD F1, 0(R1)	1	2	5		RS1	LDU1
FADD F2, F2, F3	2	3	6		RS4	FPU1
ADDI R3, R3, 8	3	4	7	struct CDB	RS7	ALU1
LD F4, 0(R2)	4	5	8		RS2	LDU2
FADD F5, F4, F2	5	9	12	RAW F4 RAW F2	RS5	FPU1
FMULT F6, F1, F4	6	9	13	RAW F4 struct CDB	RS6	FPU2
ADDI R5, R5, 1	7	8	9		RS8	LDU1
LD R6, 0(R4)	8	9	14	struct CDB	RS1	LDU1
SD F6, 0(R5)	9	14	17	RAW F6	RS2	LDU2
SD F5, 0(R6)	10	15	18	RAW F5 RAW R6	RS3	LDU1

5.3 Exercise 3

Consider the following code:

```
1 LW F1, 0(R0)
2 FADDI F1, F1, C1
3 FADDI F2, F1, C2
4 SW F2, 0(R0)
5 LW F2, 4(R0)
6 FADD F2, F2, F2
7 SW F2, 4(R0)
```

We have:

- 2 Reservation Stations (RS1, RS2) with 1 Load Store unit (LDU1) with 4 clock cycles latency
- 2 Reservation Stations (RS3, RS4) with 1 Integer ALU (ALU1) with 2 clock cycles latency

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

Solution

1. The conflicts are the following:
   - RAW F1 I1-I2.
   - RAW F1 I2-I3.
   - RAW F2 I3-I4.
   - RAW F2 I6-I7.
   - RAW F2 I5-I6.
   - WAW F2 I5-I6.
   - WAW F2 I5-I3.
   - WAW F1 I2-I1.
   - WAW F2 I6-I3.
   - WAR F2 I4-I5.
   - WAR F2 I4-I6.
2. The final result of Tomasulo is:

Instruction	Issue	Execution	Write	Hazards	RS	Unit
LW F1, 0(R0)	1	2	6		RS1	LDU1
FADDI F1, F1, C1	2	7	9	RAW F1	RS3	ALU1
FADDI F2, F1, C2	3	10	12	RAW F1 struct ALU1	RS4	ALU1
SW F2, 0(R0)	4	14	18	RAW F2 struct LDU1	RS2	LDU1
LW F2, 4(R0)	7	8	13	struct RS1 struct CDB	RS1	LDU1
FADD F2, F2, F2	10	14	16	struct RS3 RAW F2 struct ALU1	RS3	ALU1
SW F2, 4(R0)	14	19	23	struct RS1 RAW F2 struct LDU1	RS1	LDU1

5.4 Exercise 4

Consider the following code:

```
1 LOOP:   MULTD F2, F6, F8
2         ADDD F0, F6, F4
3         MULTD F10, F0, F2
4         ADDD F0, F12, F14
5         SUBI R0, R0, 4
6         BNEZ R0, LOOP
```

We have:

- 3 Reservation Stations (RS1, RS2, RS3) with 3 MULT units (MULT1,MULT2, MULT3) with 4 clock cycles latency.
- 2 Reservation Stations (RS4, RS5, RS6) with 3 ADDD units (ADDD1, ADDD2, ADDD3) with 2 clock cycles latency.
- A reorder buffer with seven slots.

In the case of hazard on CDB, the oldest instruction has priority. Let's assume that we discover the BNEZ misprediction 5 clock cycles after the issue.

1. Find all the conflicts.
2. Apply the Tomasulo algorithm wit Reorder buffer.

Solution

1. The conflicts are the following:
   - RAW F2 I1-I3.
   - RAW F0 I2-I3.
   - RAW R0 I5-I6.
   - WAW F0 I2-I4.
   - WAR F0 I3-I4.
2. The final result of the Tomasulo algorithm is:

Instruction	Issue	Execution	Write	Commit	ROB	Hazards	RS	Unit
MULTD F2, F6, F8	1	2	6	7	0		RS1	MULT1
ADDD F0, F6, F4	2	3	5	8	1		RS4	ADDD1
MULTD F10, F0, F2	3	7	11	12	2	RAW F2 RAW F0	RS2	MULT2
ADDD F0, F12, F14	4	5	7	13	3		RS5	ADDD2
MULTD F2, F6, F8	7	8			6		RS1	MULT1
ADDD F0, F6, F4	8	9			0	struct CDB	RS4	ADDD1
MULTD F10, F0, F2	9				1	RAW F2 RAW F0	RS3	
ADDD F0, F12, F14						struct ROB		

The second iteration of the loop has ben flushed since we hade more that 5 clock cycles of misprediction.

CHAPTER 6

Very Long Instruction Word

6.1 Exercise 1

Consider the program be executed on a three-issue VLIW MIPS architecture with three fully pipelined functional units:

- Integer ALU with 1 cycle latency to next and 2 cycle latency to next branch.
- Memory Unit with 3 cycle latency.
- Floating Point Unit with 3 cycle latency.

Branch completed with one cycle delay slot. Do not use neither software pipelining nor loop unrolling nor modifying loop indexes. Consider the following code:

```
1 FOR:    LD F2, VB(R6)
2         FADD F3, F2, F6
3         ST F3, VA(R7)
4         LD F3, VC(R6)
5         ST F3, VC(R7)
6         FADD F4,F4,F3
7         ADDI R6, R6, 4
8         ADDI R7, R7, 4
9         BLT R7, R8, FOR
```

1. Identify all the conflicts.
2. Considering one iteration of the loop, schedule the assembly code for the VLIW machine in the following table by using the list-based scheduling.

Solution

1. The hazards in the code are:
   - RAW F2 I1-I2.
   - RAW F3 I2-I3.

- RAW F3 I4-I5.
- RAW F3 I4-I6.
- RAW R7 I8-I9.
- WAW F3 I2-I4.
- WAR R7 I8-I5.
- WAR R7 I8-I3.
- WAR R6 I7-I1.
- WAR R6 I7-I4.
- WAR F3 I3-I4.
- CNTRL.

2. The corresponding pipelined VLIW pipeline is the following:

Clock cycle	**Integer ALU**	**Memory Unit**	**FPU**
1		LD F2, VB(R6)	
2			
3			
4			FADD F3, F2, F6
5			
6			
7		ST F3, VA(R7)	
8	ADDI R6, R6, 4	LD F3, VC(R6)	
9			
10			
11	ADDI R7, R7, 4	ST F3, VC(R7)	FADD F4,F4,F3
12			
13	BLT R7, R8, FOR		
14	(branch delay slot)		

6.2 Exercise 2

Consider the program be executed on a three-issue VLIW architecture with three non-pipelined functional units:

- Integer ALU with 1 cycle latency to and 2 cycle latency to next branch.
- Memory Unit with 3 cycle latency.
- Floating Point Unit with 3 cycle latency.

Branch completed with 1 cycle delay slot. Do not use neither software pipelining nor loop unrolling nor modifying loop indexes. Consider the following code:

```
1 LW F2, A(F4)
2 ADDI F2, F2, 4
3 LW F3, B(F4)
4 ADDI F6, F2, -5
```

```
5 SUB F5, F4, F3
6 SW F5, C(F4)
7 ADDI F4, F4, 4
8 BNE F4, F7, L1
```

1. Execute the pipelined VLIW pipeline.
2. Considering one iteration of the loop, schedule the assembly code for the VLIW machine in the following table by using the list-based scheduling.

Solution

1. The corresponding pipelined VLIW pipeline is the following:

Clock cycle	Integer ALU	Memory Unit	FPU
1		LW F2, A(F4)	
2			
3			
4	ADDI F2, F2, 4	LW F3, B(F4)	
5	ADDI F6, F2, -5		
6			
7	SUB F5, F4, F3		
8	ADDI F4, F4, 4	SW F5, C(F4)	
9			
10	BNE F4, F7, L1		
11	(branch delay slot)		

2. The corresponding non-pipelined VLIW pipeline is the following:

Clock cycle	Integer ALU	Memory Unit	FPU
1		LW F2, A(F4)	
2		LW F3, B(F4)	
3			
4	ADDI F2, F2, 4		
5	ADDI F6, F2, -5		
6	SUB F5, F4, F3		
7	ADDI F4, F4, 4	SW F5, C(F4)	
8			
9	BNE F4, F7, L1		
10	(branch delay slot)		

CHAPTER 7

Cache coherency

7.1 Exercise 1

Consider the following access pattern on a two processor system with a direct-mapped, write back cache with one cache block and a two cache block memory. Assume the MESI protocol is used, with write back caches, write-allocate, and invalidation of other caches on write (instead of updating the value in the other caches).

In the initial state the cache of the processor zero is in state Exclusive(1) and the cache of the other one is in Invalid state. Both main memory blocks are up to date.

Solution

The final table, considering the operations given is:

Clock cycle	Operation	P0 cache	P1 cache	Block 0	Block 1
0	P0: read block 1	Exclusive (1)	Invalid	Yes	Yes
1	P1: read block 0	Exclusive (1)	Exclusive (0)	Yes	Yes
2	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
3	P0: read block 1	Modified (1)	Exclusive (0)	Yes	No
4	P1: read block 0	Modified (1)	Exclusive (0)	Yes	No
5	P0: write block 1	Modified (1)	Exclusive (0)	Yes	No
6	P1: read block 1	Shared (1)	Shared (1)	Yes	Yes
7	P1: read block 0	Shared (1)	Exclusive (0)	Yes	Yes
8	P0: read block 1	Shared (1)	Shared (1)	Yes	Yes
9	P0: write block 1	Modified (1)	Invalid	Yes	Yes
10	P1: write block 0	Modified (1)	Modified (0)	Yes	Yes
11	P0: read block 1	Shared (1)	Shared (1)	Yes	Yes