ZHEJIANG UNIVERSITY COLLEGE OF INFORMATION SCIENCE AND ELECTRONICS ENGINEERING

HOMEWORK #5

Computer Organization and Design

Name: Student ID:

Major: Electronic Science and Technology

Date: 2024 年 12 月 6 日

Problem 1.

In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250ps	400ps	150ps	350ps	200ps

Also, assume that instructions executed by the processor are broken down as follows:

alu	beq	lw	sw
40%	20%	25%	15%

a) What is the clock cycle time in a pipelined and non-pipelined processor?
b) What is the total latency of an LW instruction in a pipelined and non-pipelined processor?
c) If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

Answer: a) Clock cycle time in a pipelined and non-pipelined processor

Non-pipelined processor: The clock cycle time of a non-pipelined processor is the sum of the latencies of all stages:

$$T_{\text{non-pipelined}} = \text{IF} + \text{ID} + \text{EX} + \text{MEM} + \text{WB}$$

Substitute the values:

$$T_{\text{non-pipelined}} = 250\,\text{ps} + 400\,\text{ps} + 150\,\text{ps} + 350\,\text{ps} + 200\,\text{ps} = 1350\,\text{ps}$$

Pipelined processor: The clock cycle time of a pipelined processor is determined by the stage with the maximum latency:

$$T_{\text{pipelined}} = \max(\text{IF}, \text{ID}, \text{EX}, \text{MEM}, \text{WB})$$

Substitute the values:

$$T_{\text{pipelined}} = \max(250\,\text{ps}, 400\,\text{ps}, 150\,\text{ps}, 350\,\text{ps}, 200\,\text{ps}) = 400\,\text{ps}$$

b) Total latency of an LW instruction

Non-pipelined processor: no matter pipelined or non-pipelined, the total latency of an LW instruction is the sum of the latencies of all the stages that the instruction passes through:

because the LW instruction passes through all stages, the total latency is the same as the clock cycle time:

$$\text{Latency}_{\text{non-pipelined, LW}} = T_{\text{non-pipelined}} = 1350\,\text{ps}$$

Pipelined processor:

$$\text{Latency}_{\text{pipelined, LW}} = T_{\text{non-pipelined}} = 1350\,\text{ps}$$

c) Splitting one stage to reduce clock cycle time

Selecting the stage to split: we choose the stage with the maximum latency, which is the ID stage (400 ps).

If we split the ID stage (400 ps) into two stages with half the latency each (200 ps), the new maximum stage latency becomes:

$$T_{\text{new-pipelined}} = \max(\text{IF}, \text{ID}/2, \text{EX}, \text{MEM}, \text{WB}) = 400\,\text{ps}$$

New clock cycle time:

$$T_{\text{new-pipelined}} = 350\,\text{ps}$$

Problem 2.

Look for data hazards in the code below, and figure out how forwarding could be used to solve them.

them.

Instruction	C1	C2	C3	C4	C5	C6	C7
1. addi t0, a0, −1	IF	ID	EX	MEM	WB		
2. and s2, t0, a0		IF	ID	EX	MEM	WB	
3. sltiu a0, t0, 5			IF	ID	EX	MEM	WB

Answer : Identification of Hazards

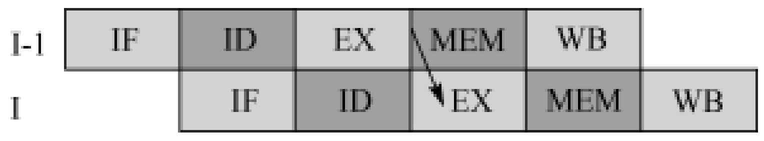
1. Between Instructions 1 and 2: - Instruction 2 requires the value of `t0`, which is produced in the EX stage of Instruction 1 . But in the ID stage of Instruction 2, the value of `t0` is not yet write back to the register file.

2. Between Instructions 1 and 3: - Instruction 3 also requires the value of `t0`, which is produced in the EX stage of Instruction 1 . but in the ID stage of Instruction 3, the value of `t0` is not yet write back to the register file.

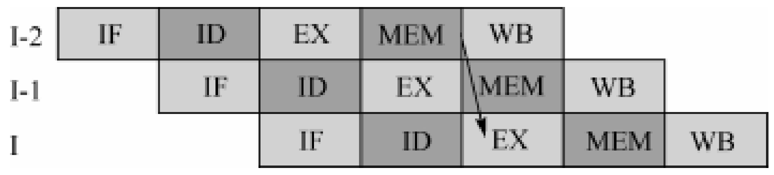
Resolving Hazards Using Forwarding

Forwarding allows the processor to bypass the pipeline register and directly use the result from an earlier stage. Here's how forwarding resolves these hazards:

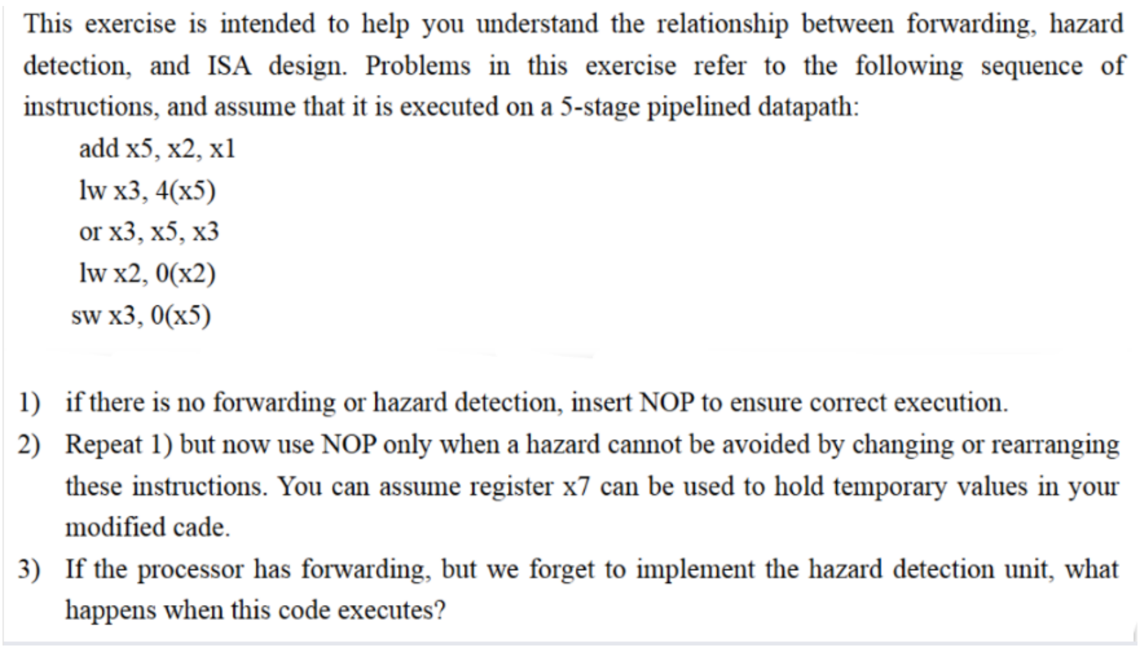
1. Instruction 1 → Instruction 2: - Forward the result of `t0` from the end of EX stage of Instruction 1 to the EX stage of Instruction 2.



2. Instruction 1 → Instruction 3: - Forward the result of `t0` from the end of MEM stage of Instruction 1 to the ID stage of Instruction 3.



Problem 3.



This exercise is intended to help you understand the relationship between forwarding, hazard detection, and ISA design. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a 5-stage pipelined datapath:

```
add x5, x2, x1
lw x3, 4(x5)
or x3, x5, x3
lw x2, 0(x2)
sw x3, 0(x5)
```

1) if there is no forwarding or hazard detection, insert NOP to ensure correct execution.
2) Repeat 1) but now use NOP only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register x7 can be used to hold temporary values in your modified cade.
3) If the processor has forwarding, but we forget to implement the hazard detection unit, what happens when this code executes?

Answer： Assume the register files are read after write in the same clock cycle, so we don't need to consider the 3rd data hazard between Instruction 1 and Instruction 4. Oherwise, we need to insert one more NOP to prevent data hazards.

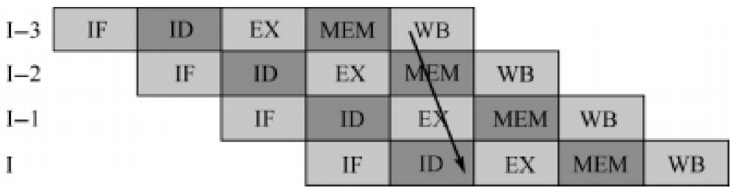


图 1 the 3rd data hazard between Instruction 1 and Instruction 4

Code Sequence

```
1. add x5, x2, x1
2. lw x3, 4(x5)
3. or x3, x5, x3
4. lw x2, 0(x2)
5. sw x3, 0(x5)
```

(1) No Forwarding or Hazard Detection

Hazard Analysis:

- Instruction 1 → Instruction 2: data hazard on `x5`.
- Instruction 2 → Instruction 3: data hazard on `x3`.
- Instruction 4 → Instruction 5: data hazard on `x3`.

Resulting Code with NOPs:

```
1. add x5, x2, x1
2. NOP
3. NOP
4. lw x3, 4(x5)
5. NOP
6. NOP
7. or x3, x5, x3
8. lw x2, 0(x2)
9. NOP
10. sw x3, 0(x5)
```

(2) Reordering Instructions with Temporary Register

the instruction 1,2,3,5 can't be reordered because of the data dependencies. we can only reorder the instruction 4.

But no matter how we reorder the instruction 4, we can't eliminate the NOPs. so we can't eliminate the NOPs by reordering instructions.

but we can avoid the data hazards between Instruction 2 and Instruction 3 by reordering the instruction 3 and 4.

```
1. add x5, x2, x1
2. lw x3, 4(x5)
3. lw x2, 0(x2)
4. or x3, x5, x3
5. sw x3, 0(x5)
```

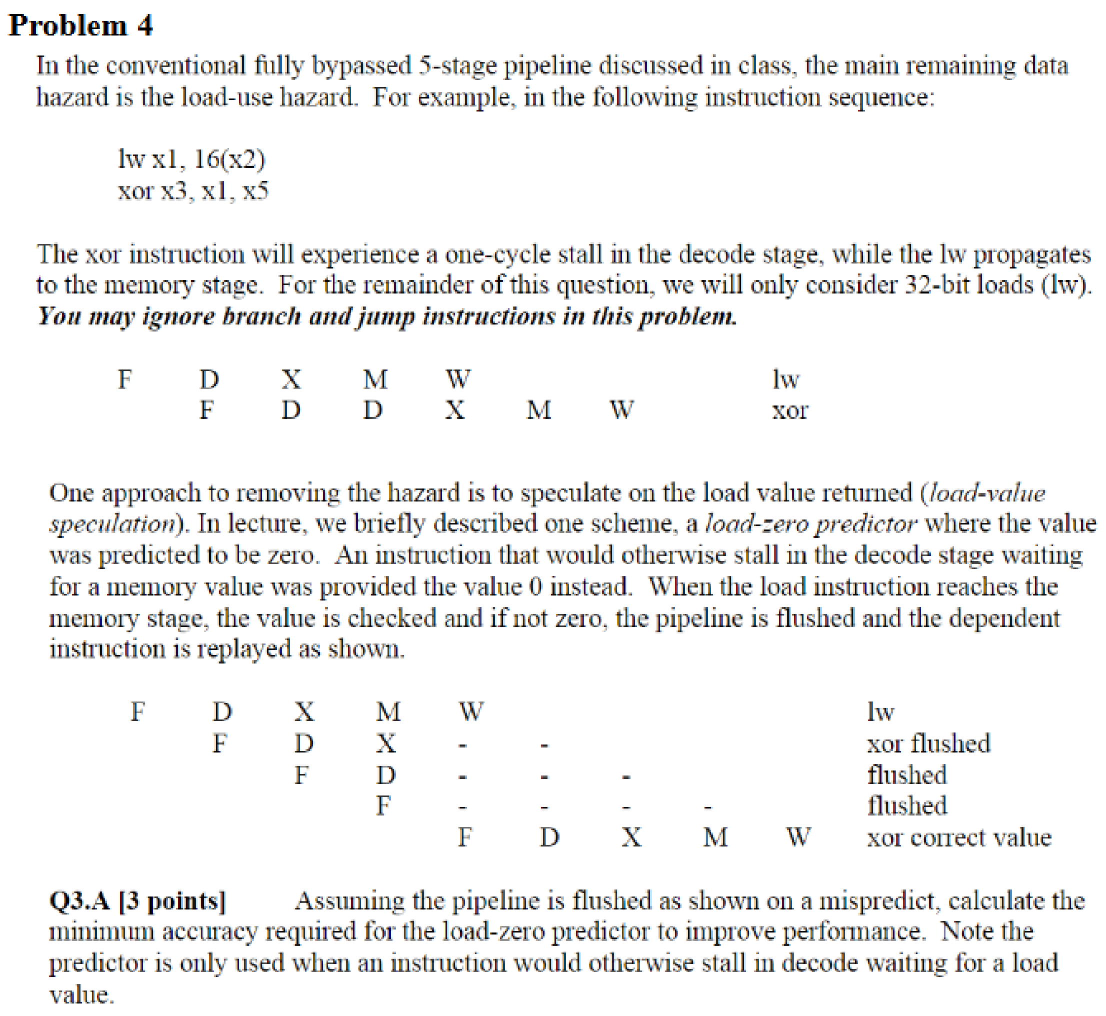
(3) Forwarding Without Hazard Detection

When the instruction following a load uses the result of the load, the hazard is happening.

the original code has a hazard between Instruction lw x3, 4(x5) and or x3, x5, x3.

so the code execution will be wrong.

Problem 4.



Problem 4

In the conventional fully bypassed 5-stage pipeline discussed in class, the main remaining data hazard is the load-use hazard. For example, in the following instruction sequence:

```
lw x1, 16(x2)
xor x3, x1, x5
```

The xor instruction will experience a one-cycle stall in the decode stage, while the lw propagates to the memory stage. For the remainder of this question, we will only consider 32-bit loads (lw). ***You may ignore branch and jump instructions in this problem.***

F	D	X	M	W			lw
	F	D	D	X	M	W	xor

One approach to removing the hazard is to speculate on the load value returned (*load-value speculation*). In lecture, we briefly described one scheme, a *load-zero predictor* where the value was predicted to be zero. An instruction that would otherwise stall in the decode stage waiting for a memory value was provided the value 0 instead. When the load instruction reaches the memory stage, the value is checked and if not zero, the pipeline is flushed and the dependent instruction is replayed as shown.

F	D	X	M	W					lw
	F	D	X	-	-				xor flushed
		F	D	-	-	-			flushed
			F	-	-	-	-		flushed
				F	D	X	M	W	xor correct value

Q3.A [3 points] Assuming the pipeline is flushed as shown on a mispredict, calculate the minimum accuracy required for the load-zero predictor to improve performance. Note the predictor is only used when an instruction would otherwise stall in decode waiting for a load value.

Answer : 1. Pipeline Flush Cost: - A misprediction causes the pipeline to be flushed, with the dependent instruction replayed. - The flush penalty is 3 cycles.

2. Predictor Accuracy Threshold: - Let p be the probability of a correct prediction (predictor accuracy). - On a correct prediction, a 1-cycle stall is avoided. - On a misprediction, a 3-cycle flush penalty occurs.

Performance Analysis

To calculate the required accuracy p, we compare the average cycle costs with and without speculation.

Cost Without Speculation

$$\text{Cost}_{\text{no speculation}} = 1 \quad \text{(1-cycle stall)}$$

Cost With Speculation

$$\text{Cost}_{\text{speculation}} = p \cdot 0 + (1-p) \cdot 3 = 3(1-p)$$

The predictor improves performance if:

$$\text{Cost}_{\text{speculation}} < \text{Cost}_{\text{no speculation}}$$

Solving the Inequality

$$3(1-p) < 1$$
$$3 - 3p < 1$$
$$3p > 2$$
$$p > \frac{2}{3}$$

Conclusion

The load-zero predictor must have an accuracy of at least:

$$\boxed{\frac{2}{3}\,(66.67\%)}$$

to improve pipeline performance.