King
CS 331
21 November 2020

Homework Assignment 5

Objectives

The purpose of this assignment is to:

- Understand the key principles for building a pipelined datapath and control that implements a subset of MIPS instructions,
- Understand how data and control dependencies affect execution of a pipelined design, and how such dependences are handled using forwarding, hazard detection, code rescheduling, and branch prediction,
- Understand how splitting stages and deeper pipelines can affect processor performance.

Guidelines

All question numbers refer to exercises at the end of chapter 4 of the textbook (Computer organization and design: the Hardware/Software interface, 5th edition). Solutions for the following problems are to be done by you and only you.

Questions

Exercise 4.8.1 to 4.8.3

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	350ps	150ps	300ps	200ps

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

alu	beq	lw	sw
45%	20%	20%	15%

1. What is the clock cycle time in a pipelined and non-pipelined processor?
2. What is the total latency of an LW instruction in a pipelined and non-pipelined processor?

3. 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?

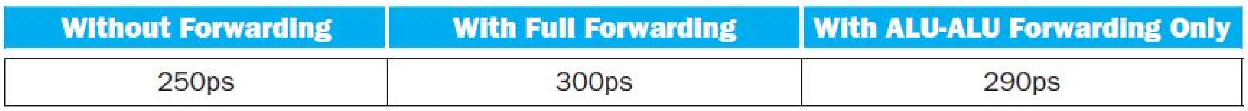
In a non-pipelined processor, the latency is IF + ID + EX + MM + WB = (250 + 350 + 150 + 300 + 200) ps = 1250 ps. Since ID takes the longest when overlapping each stage’s execution time, the latency in the pipelined processor is 350 ps.
The latency of a LW instruction through the 5 stages of this non-pipelined processor is 1250 ps. Since ID creates the longest cycle time throughout any stage, the latency in the pipelined processor is (5)*(350 ps) = 1750 ps.
Since it has the highest latency, the ID can be split into two 175 ps stages. The new clock cycle time for the pipelined processor would then be 300 ps, since the MEM stage would now have the highest latency.

Exercise 4.9.1 to 4.9.4 (Note there are three types of data dependences RAW, WAR, and WAW)

In this exercise, we examine how data dependencies affect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:

```
or r1,r2,r3
or r2,r1,r4
or r1,r1,r2
```

Also, assume the following cycle times for each of the options related to forwarding:



Without Forwarding	With Full Forwarding	With ALU-ALU Forwarding Only
250ps	300ps	290ps

1. Indicate dependences and their type.
2. Assume there is no forwarding in this pipelined processor. Indicate hazards and add nop instructions to eliminate them.
3. Assume there is full forwarding. Indicate hazards and add NOP instructions to eliminate them.
4. What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

```
Given:
I1. or r1,r2,r3
I2. or r2,r1,r4
I3. or r1,r1,r2

The second instruction I2 has a dependency on r1 from the first instruction
(read after write).

The third instruction I3 has a dependency on r1 from the first instruction
(write after write) and r2 from the second instruction (read after write).
```

```
I1. IF ID EX MM WB
I2. -- -- -- IF ID EX MM WB          (read after write)
I3. -- -- -- -- -- -- IF ID EX MM WB (read after write)
```

```
I1. IF ID EX MM WB       (forward from EX/MEM register)
I2. -- IF ID EX MM WB    (forward from EX/MEM register)
I3. -- -- IF ID EX MM WB
```

```
Without forwarding, the instruction sequence requires 11 clock cycles.
```

Exercise 4.13.1

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 r5,r2,r1
lw r3,4(r5)
lw r2,0(r2)
or r3,r5,r3
sw r3,0(r5)
```

If there is no forwarding or hazard detection, insert nops to ensure correct execution.

```
Given:
I1. add r5,r2,r1
I2. lw r3,4(r5)
I3. lw r2,0(r2)
I4. or r3,r5,r3
I5. sw r3,0(r5)

I1. IF ID EX MM WB
I2. -- -- -- IF ID EX MM WB
```

```
I3. -- -- -- -- IF ID EX MM WB
I4. -- -- -- -- -- IF ID EX MM WB
I5. -- -- -- -- -- -- -- -- IF ID EX MM WB
```

Exercise 4.15.1

The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

R-Type	BEQ	JMP	LW	SW
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the EX stage, that there are no data hazards, and that no delay slots are used.

```
Given:
Branch outcomes are determined in the EX stage, no data hazards, no delay slots
used.

In the EX stage, up to 3 cycles may be discarded since EX occurs from the third
clock cycle.
```

Exercise 4.16.1 to 4.16.3

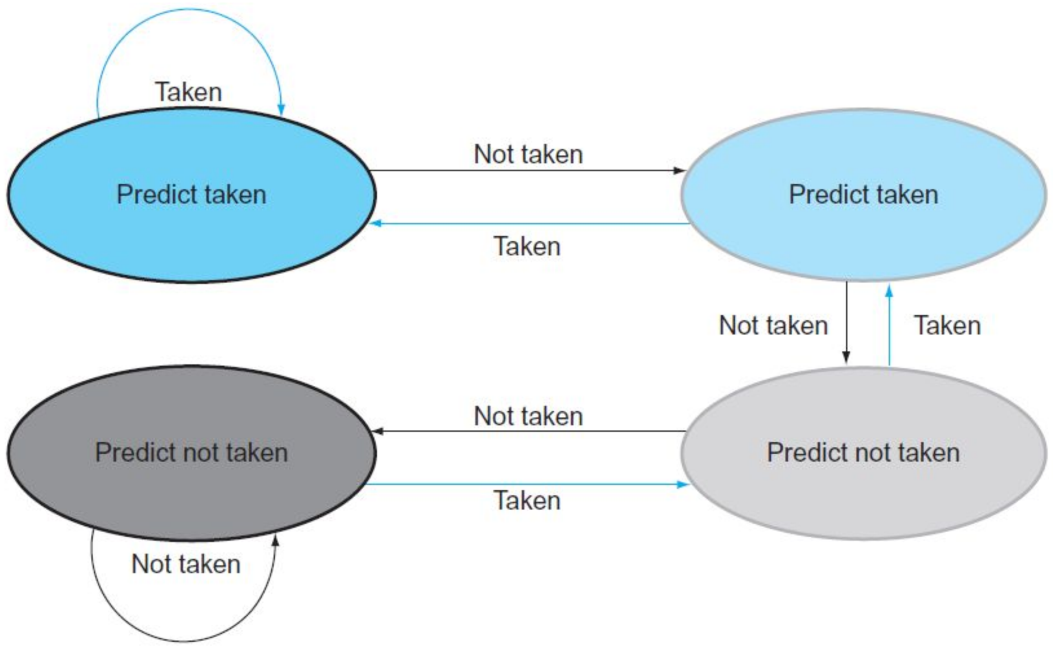


FIGURE 4.63 The states in a 2-bit prediction scheme. By using 2 bits rather than 1, a branch that strongly favors taken or not taken—as many branches do—will be mispredicted only once. The 2 bits are used to encode the four states in the system. The 2-bit scheme is a general instance of a counter-based predictor, which is incremented when the prediction is accurate and decremented otherwise, and uses the mid-point of its range as the division between taken and not taken.

This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes: T, NT, T, T, NT

1. What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?
2. What is the accuracy of the two-bit predictor for the first 4 branches in this pattern, assuming that the predictor starts off in the bottom left state from Figure 4.63 (predict not taken)?
3. What is the accuracy of the two-bit predictor if this pattern is repeated forever?

Given: T NT T T NT, with the always-taken occurring 3 out of 5 times in the pattern and the always-not-taken occurring 2 out of the 5 times. The accuracy of the always-taken predictor is 3/5 = 60%, and the always-not-taken is 2/5 = 40%.
S1. “Predict not taken” (left) > T > “Predict not taken” (right) S2. “Predict not taken” (right) > NT > “Predict not taken” (left) S3. “Predict not taken” (left) > T > “Predict not taken” (right) S4. “Predict not taken” (right) > T > “Predict taken” (right) S5. The last NT branch in the pattern is not counted. The resulting predictors are: NT, NT, NT, and T. Therefore, the accuracy of the

two-bit predictor for the first 4 branches in this pattern is 1/4 = 25%.

Since the pattern of the always-taken predictor is 60%, the accuracy of the two-bit predictor is 60% if repeated infinitely.

Extra Credit

What does each one of the techniques listed below help reduce/improve: data hazard stalls, control stalls, or/and CPI? The techniques are:

Dynamic scheduling, branch prediction, multiple issue, speculation, loop unrolling.

Dynamic scheduling: Reduces data hazard stalls by executing independent instructions in between dependent instructions.

Branch prediction:

Multiple-issue:

Speculation:

Loop unrolling: