BCT 2408 Computer Architecture-II: Lab 3 Solution

Name: Jeff Gicharu
Reg No: SCT212-0053/2021

This document provides the solutions to the problems presented in Lab 3, focusing on pipeline hazards and branch prediction.

E1: Hazard Identification (individual 10 mins)

Problem

Consider the following MIPS code fragments, each containing two instructions. For each code fragment identify the type of hazard that exists between the two instructions and the registers involved.

(a)

```
LD R1, 0(R2)
DADD R3, R1, R2
```

- **Answer:** This is a **RAW (Read After Write)** data hazard involving register **R1**. The DADD instruction needs to read R1, but the LD instruction only makes the value available after the MEM stage. This typically requires forwarding and may still incur a one-cycle stall (load-use hazard).

(b)

```
MULT R1, R2, R3
DADD R1, R2, R3
```

- **Answer:** This is a **WAW (Write After Write)** output dependency involving register **R1**. Both instructions write to R1. In simple pipelines, this is often handled by in-order write-back. In more complex pipelines (or with multi-cycle writes), this can cause issues if DADD completes before MULT, requiring techniques like register renaming.

(c)

```
MULT R1, R2, R3
MULT R4, R5, R6
```

- **Answer:** Assuming only one multiplier unit, this is a **Structural Hazard**. Both MULT instructions need the multiplier hardware in the EX stage simultaneously. The second MULT must stall. If multiple multipliers exist, this hazard is removed.

(d)

```
DADD R1, R2, R3
SD   2000(R0), R1
```

- **Answer:** This is a **RAW (Read After Write)** data hazard involving register **R1**. The SD instruction needs to read R1 to store it (in the MEM stage). DADD produces R1 in the EX stage. Forwarding from the EX/MEM pipeline register to the MEM stage input is needed to avoid a stall.

(e)

```
DADD R1, R2, R3
SD   2000(R1), R4
```

- **Answer:** This is a **RAW (Read After Write)** data hazard involving register **R1**. The SD instruction needs R1 to calculate the memory address 2000 + R1 (in the EX stage). DADD produces R1 in its EX stage. Forwarding from the DADD's EX stage output to the SD's EX stage input is needed to avoid a stall.

E2: Branch Prediction (groups of 2, 15 minutes)

Problem

(a) Explain the behaviour of a 2-bit saturating counter branch predictor. Show the state of the predictor and the transition for each outcome of the branch.

- Answer:
  A 2-bit saturating counter uses a 4-state finite state machine for prediction:
  - 00: **Strongly Not Taken** (Predict Not Taken)
  - 01: **Weakly Not Taken** (Predict Not Taken)
  - 10: **Weakly Taken** (Predict Taken)
  - 11: **Strongly Taken** (Predict Taken)

  Transitions:

- *If Branch is Taken:*
  - 00 -> 01
  - 01 -> 10
  - 10 -> 11
  - 11 -> 11 (Saturates)
- *If Branch is Not Taken:*
  - 00 -> 00 (Saturates)
  - 01 -> 00
  - 10 -> 01
  - 11 -> 10

Prediction Logic:

- Predict **Not Taken** if state is 00 or 01.
- Predict **Taken** if state is 10 or 11.

(b) Consider the following code:
c for (i=0; i<N; i++) if (x[i]==0) y[i]=0.0; else y[i]=y[i]/x[i];
Assume that the assembly code generated is then:
mips loop: L.D F1, 0(R2) # Load x[i] into F1 L.D F2, 0(R3) # Load y[i] into F2 BNEZ F1, else # Branch if F1 (x[i]) != 0 ADD.D F2, F0, F0 # y[i] = 0.0 (F0 = 0.0) BEZ R0, fall # Unconditional branch to fall else: DIV.D F2, F2, F1 # y[i] = y[i] / x[i] fall: DADDI R2, R2, 8 # Increment x pointer DADDI R3, R3, 8 # Increment y pointer DSUBI R1, R1, 1 # Decrement loop counter N S.D -8(R3), F2 # Store y[i] result BNEZ R1, loop # Loop if N != 0
where:
* the value of N is already stored in R1
* the base addresses for x and y are stored in R2 and R3, respectively
* register F0 contains the value 0
* register R0 (always) contains the value 0
Assuming that every other element of x has the value 0, starting with the first one, show the outcomes of predictions when a 2-bit saturating counter is used to predict the inner branch `BNEZ F1, else`. Assume that the initial value of the counter is 00.

- Answer:
  The branch BNEZ F1, else is Taken if x[i] != 0 and Not Taken if x[i] == 0. The actual outcome sequence is: Not Taken, Taken, Not Taken, Taken...
  Initial State: 00 (Strongly Not Taken)

Iteration	x[i] value	Actual Outcome	Predictor State	Predictio	Correct?	Predictor State

(i)		(BNEZ F1)	(Start)	n		(End)
0	0	**Not Taken**	00 (SN)	Not Taken	Yes	00 (SN)
1	!= 0	**Taken**	00 (SN)	Not Taken	No	01 (WN)
2	0	**Not Taken**	01 (WN)	Not Taken	Yes	00 (SN)
3	!= 0	**Taken**	00 (SN)	Not Taken	No	01 (WN)
4	0	**Not Taken**	01 (WN)	Not Taken	Yes	00 (SN)
5	!= 0	**Taken**	00 (SN)	Not Taken	No	01 (WN)
...	...	...	...	...	...	...

- **Outcome Summary:** The predictor state alternates between 00 and 01, always predicting **Not Taken**. This prediction is correct 50% of the time for this alternating pattern.