Lab 3

SCT212-0051/2021 MAKUMI BEDAN NGUGI

EXAMPLE 1

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

Read After Write (R1): The DADD instruction needs to read the value of R1 after the LD instruction has written to it. If DADD reads R1 too early in the pipeline (before LD completes its WB stage), it will get the old, incorrect value.

b.

MULT R1, R2, R3

DADD R1, R2, R3

Write After Write (R1): Both instructions write to the same register R1. If the instructions could complete out of order, it's possible for DADD to write its result before MULT writes its result, violating the program order.

c.

MULT R1, R2, R3

MULT R4, R5, R6

No Hazard (N/A): The two instructions use completely different registers for their inputs and outputs (R1, R2, R3 vs. R4, R5, R6). There is no dependency between them.

d.

DADD R1, R2, R3

SD 2000(R0), R1

Read After Write (R1): The SD instruction needs to read the value from R1 to store it into memory. It needs this value after the DADD instruction has calculated and written it to R1. Reading R1 too early would store the wrong value.

e.

DADD R1, R2, R3

SD 2000(R1), R4

Read After Write (R1): The SD instruction needs to read the value of R1 to calculate the memory address (Address = Value(R1) + 2000). It needs this value after the DADD instruction has written the potentially new value into R1. Using the old value of R1 would cause the store to go to the wrong memory location.

EXAMPLE 2

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.

A 2-bit saturating counter branch predictor uses a 2-bit value for each branch instruction to track its recent history and predict its future behavior. It has four states, typically interpreted as:

- **00:** Strongly Not Taken (SN)
- **01:** Weakly Not Taken (WN)
- **10:** Weakly Taken (WT)
- **11:** Strongly Taken (ST)

Prediction Rule:

- If the counter is in states 00 or 01, predict **Not Taken**.
- If the counter is in states 10 or 11, predict **Taken**.

State Transitions (Behavior):

If the actual outcome was **Taken**: Increment the counter value (binary addition), but stop (saturate) at 11.

00	->	01
01	->	10
10	->	11
11	->	11

If the actual outcome was **Not Taken**: Decrement the counter value (binary subtraction), but stop (saturate) at 00.

11	->	10
10	->	01
01	->	00
00	->	00

b. Consider the following code:

```
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:

```
loop: L.D F1, 0(R2)
L.D F2, 0(R3)
BNEZ F1, else
ADD.D F2, F0, F0
BEZ R0, fall
else: DIV.D F2, F2, F1
fall: DADDI R2, R2, 8
DADDI R3, R3, 8
DSUBI R1, R1, 1
S.D -8(R3), F2
BNEZ R1, loop
```

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

Prediction Trace Table

Iteration i	Value of x[i]	Actual Branch Outcome (BNEZ F1, else)	Counter State at Start of Iteration	Prediction Made	Prediction Correct?	Counter State at End of Iteration
0	0	NT (Not Taken)	00 (SN)	NT	Yes	00 (SN)
1	!= 0	T (Taken)	00 (SN)	NT	No	01 (WN)
2	0	NT (Not Taken)	01 (WN)	NT	Yes	00 (SN)
3	!= 0	T (Taken)	00 (SN)	NT	No	01 (WN)
4	0	NT (Not Taken)	01 (WN)	NT	Yes	00 (SN)
5	!= 0	T (Taken)	00 (SN)	NT	No	01 (WN)