REVL WALTER MBOGO NJUKI – SCT212-0053/2020

Computer Architecture - tutorial 3 [TUTOR COPY]

Context, Objectives and Organization

Supplements material from Lecture 5 (Handling hazards).

The goals of the quantitative exercises in this tutorial are: to obtain familiarity with the process of identifying hazards (E1), and to review and apply the operation of different types of branch predictors (E2).

E1: 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
```

b.

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

c.

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

R6

d.

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

e.

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

Solution

A. **Hazard Type:** RAW (Read After Write)
Registers Involved: R1 **Explanation:**

- The LD instruction loads a value from memory into register R1.
- The DADD instruction immediately after uses R1 as a source operand.
- Since DADD tries to read R1 before LD completes writing to it, a RAW hazard occurs.

B. **Hazard Type:** WAW (Write After Write) **Registers Involved:** R1
Explanation:
    - The MULT instruction writes to R1.
    - The following DADD also writes to R1.
    - If both instructions complete in overlapping cycles (e.g., in a pipelined architecture), the final value in R1 may depend on the instruction completion order, which causes a WAW hazard.

C. **Hazard Type:** Structural Hazard
Registers Involved: None (applies to functional units) **Explanation:**

- Both instructions are MULT, and if the processor only has one multiplier unit, they cannot execute simultaneously.
- This results in a structural hazard due to resource contention for the multiplier unit.

D. **Hazard Type:** RAW (Read After Write)
Registers Involved: R1 **Explanation:**
- DADD computes a result and writes it to R1.
- The SD instruction stores the value from R1 into memory.
- Since SD needs to read the value from R1 (during the MEM stage) after DADD writes it (in WB), a RAW hazard is present.

E. **Hazard Type:** RAW (Read After Write)
Registers Involved: R1 **Explanation:**

- The DADD writes the result to R1.
- The SD instruction uses R1 to compute the memory address (base register in 2000(R1)), which happens in the ALU stage.
- If SD starts using R1 before DADD completes writing to it, this leads to a RAW hazard.

E2: 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.

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.

Solution

A. 2-bit saturating counter branch predictor

current counter value	prediction	actual outcome	new counter value
00	NT	NT	00
00	NT	T	01
01	NT	NT	00
01	NT	T	10
10	T	NT	01
10	T	T	11
11	T	NT	10
11	T	T	11

State Transitions:

- **From 00 (Strongly Not Taken):**
  - If branch **Not Taken** → Stay in 00
  - If branch **Taken** → Move to 01
- **From 01 (Weakly Not Taken):**
  - If branch **Not Taken** → Move to 00
  - If branch **Taken** → Move to 10
- **From 10 (Weakly Taken):**
  - If branch **Not Taken** → Move to 01
  - If branch **Taken** → Move to 11
- **From 11 (Strongly Taken):**
  - If branch **Not Taken** → Move to 10
  - If branch **Taken** → Stay in 11

B. 2-bit counter prediction rate

Iteration	current counter value	prediction	actual outcome	new counter value
1	00	NT	NT	00 (hit)
2	00	NT	T	01 (miss)
3	01	NT	NT	00 (hit)
4	00	NT	T	01 (miss)
...	...	...	...	...