Computer Architecture - CS 301

Rishit Saiya - 180010027, Assignment - 7

October 17, 2020

1

Hazards in general inculcate when there the instruction is about to have a conflict while fetching correct value in the EX stage. Interlocking is the hardware mechanism to enforce correctness in the pipeline. There are two kinds of Interlocks:

- **Data Lock:** Here we don't allow a consumer instruction to move beyond OF stage till it has read the correct values, i.e., stall the IF and OF stages. It is in general used for data hazards.
- **Branch Lock:** Here, we don't execute instructions in the wrong path. It is in general used for control hazards.

When the branch is taken there, we convert the instructions in IF & OF stage to bubbles. We know that an instruction is a branch instruction in the OF stage. This ensures the branch-lock condition.

We can do it by following the below algorithm steps:

```
if (Branch Instruction in EX stage is taken){
    Invalidate the instructions in IF & OF stages;
    Fetch values from Branch Target;
}
else {
    continue;
}
```

As a demonstration, let's consider the example as given in video:

```
[1] beq .foo_buff
[2] add r1, r2, r3
```

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
IF	**1**	**2**	**3**	**4**				
OF		**1**	**2**	**B**	**4**			
EX			**1**	**B**	**B**	**4**		
MA				**1**	**B**	**B**	**2**	
RW					**1**	**B**	**B**	**2**

Table 1: Pipeline View

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
IF	**1**	**2**	**3**	**4**				
OF		**1**	**2**	**3**	**4**			
EX			**1**	**2**	**3**	**4**		
MA				**1**	**2**	**3**	**4**	
RW					**1**	**2**	**3**	**4**

Table 2: Example for Q.3

```
[3] add r4, r5, r6
 .
 .
 .
    .foo_buff
[4] add r7, r8, r9
```

The Figure 1 shows the Pipeline view of the above set of instructions. The compiler interprets uses the above algorithm and checks that if the branch instruction is taken, which is taken in this case. It then makes those Instructions [2], [3] which were in IF & OF stages bubbled and makes sure that they are not executed further.

Minimum number of Bubbles:

If the branch is taken, and all the other instructions in IF and OF stages belonging to the same cycle that doesn't belong to this branch, in that case, the minimum number of bubbles will be **2** as shown in the above example.

But if the branch isn't taken, and all the other instructions in IF and OF stages belonging to the same cycle does not belong to this branch, in that case, the minimum number of bubbles will be **0**.

A case where the branch is taken but all the other instructions in IF and OF stages belonging to the same cycle belong to this branch, in that case the minimum no. of bubbles will be **0**.

So, the minimum no. of bubbles in this case would be 0.

2

Yes, we can reduce the performance impact introduced through interlocking-based handling of control hazards by Predict-Not-Taken Method. This algorithm/method involves Branch-Look in the following steps:

```
if (Branch Instruction in EX stage is taken){
    Invalidate the instructions in IF & OF stages;
    Fetch values from Branch Target;
}
else {
    continue;
}
```

This has proven to be more efficient than the Normal-Branch-Lock Method.

3

In the Figure 2, in the 5th cycle, Instruction [4] (whose source operand is r5) needs the value of register r5 and Instruction [1] & Instruction [2] (which are in MA & RW stages respectively) write the value of r5.

Now, we wait for another cycle to process and then execute the 2 stage forwarding from RW to EX stage because Instruction [4] will need the value of r5 in EX stage.

It is assumed that WAW hazard is already taken care of in the previous cycles.

4

The general expression to calculate Performance is as follows:

$$Performance = \frac{Frequency}{CPI}$$

or

$$Performance = \frac{Frequency}{CPI_{ideal} + (Penalty_{stall} \times Rate_{stall})}$$

where CPI is the Cycle per Instruction.

Let us say that Performance of Processor P1 is P_1 and that of Processor P2 is P_2. Similarly, let their frequencies be f_1 & f_2 respectively.

Conditional Instructions are given as 30% of Total Instructions & Branches not taken are 80% of Conditional Instructions. So,

$$Penalty_{stall} = (Branch\,Instruction\,Fraction) \times (Branches\,Taken\,Fraction)$$

$$Penalty_{stall} = 0.3 \times (1 - 0.8)$$

$$Penalty_{stall} = 0.06$$

Given that CPI_{ideal} for P_1 & P_2 is 1. It is also given that f_1 & f_2 are 1 GHz & 1.2 GHz respectively. $Penalty_{stall}$ for P_1 & P_2 are 2 cycles & 3 cycles respectively.

Since there are no data hazards in the program, we calculate each performance as follows:
For Processor P1:

$$P_1 = \frac{1\,GHz}{1 + 2 \times (0.3 \times 0.2)}$$

$$P_1 \approx 0.892857$$

For Processor P2:

$$P_2 = \frac{1.2\,GHz}{1 + 3 \times (0.3 \times 0.2)}$$

$$P_2 \approx 1.0169491$$

Clearly, Processor 2 performs better than Processor 1.