**PREM KRISHNA CHETTRI**

**Computer Architecture Assignment 3 Submission Date: 19th Oct ‘15**

**Solution 1.1**: Serialize the instruction startup means we will issue this **“ready to execute**” instruction to functional unit one at a time serially in a sequential manner, even if we have multiple **“ready to execute**” instructions in our centralized issue queue and multiple functional units are available. This is because of the limitation of the path from the issue queue to functional units.

Serializing the instruction startup may create the bottleneck and decrease the performance of an architecture, as even if we have multiple **“ready to execute”** instructions in our centralized issue queue and that many functional units available, we are basically limited to issue as many instructions to the functional units as the number of available paths between them at any instance. So they have to wait for the earlier **“ready to execute”** instructions to clear up the path as they start to access the functional unit one at a time.

**Solution 1.2**: This again creates the same bottleneck issue as multiple functional units might have completed their instruction execution and is waiting for the writeback to happen. However, as there is only one shared write back, so each result from the functional units has to wait for its time in a queue to gain the access of the bus, so that the write back can happen. The results of the executed instruction are not available to the dependent instructions due to writeback bus congestion, and hence eventually hurting the performance of the system.

**Solution 1.3**:

Lets take a look at the following list of 5 instructions

I1 :- ADD R3 R2 R1

I2 :- ADD R4 R3 R2

I3 :- MUL R5 R3 R2

I4 :- ADD R6 R5 R1

I5 :- MUL R7 R3 R5

With Serialization: - Here even if R3 , R5 were available to both I2, I3 and I4,I5 respectively. As we allow only one startup, we have to serialize R3 and R5 one instruction cycle later causing an addition of 2 cycle in our pipeline.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F | 1 | 2 | 3 | 4 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  | 1 | 2 | 3 | 4 |  | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| A |  |  | 1 | 2 |  | 2 | 4 |  |  |  |  |  | 4 |  |  |  |  |  |  |  |  |
| M |  |  |  |  | 3 |  | 3 | 3 | 3 | 3 | 5 |  |  | 5 | 5 | 5 | 5 |  |  |  |  |
| ME |  |  |  | 1 |  |  | 2 |  |  |  | 3 |  |  | 4 | 5 |  |  | 5 |  |  |  |
| WB |  |  |  |  | 1 |  |  | 2 |  |  |  | 3 |  |  | 4 | 5 |  |  | 5 |  |  |

Without Serialization

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F | 1 | 2 | 3 | 4 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  | 1 | 2 | 3 | 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| A |  |  | 1 | 2 |  | 2 | 4 |  |  |  |  | 4 |  |  |  |  |  |  |  |  |  |
| M |  |  |  |  | 3 | 3 | 3 | 3 | 3 | 5 |  | 5 | 5 | 5 | 5 |  |  |  |  |  |  |
| ME |  |  |  | 1 |  |  | 2 |  |  | 3 |  |  | 4 |  |  | 5 |  |  |  |  |  |
| WB |  |  |  |  | 1 |  |  | 2 |  |  | 3 |  |  | 4 |  |  | 5 |  |  |  |  |

**Solution 1.4**:

Lets take a look at the following list of 5 instructions

I1 :- ADD R3 R2 R1

I2 :- ADD R4 R3 R2

I3 :- MUL R5 R3 R2

I4 :- ADD R6 R5 R1

I5 :- MUL R7 R3 R5

With Serialization Here even though R3 value were evaluated by instruction I1, it is not available to I2 and I3 at the same time. Hence the there is one cycle delay for I3. Same is the case with instruction I4 and I5 as they both were waiting for the value of R5 causing an addition of 2 cycle in our pipeline.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F | 1 | 2 | 3 | 4 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  | 1 | 2 | 3 | 4 |  | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| A |  |  | 1 | 2 |  | 2 | 4 |  |  |  |  |  | 4 |  |  |  |  |  |  |  |  |
| M |  |  |  |  | 3 |  | 3 | 3 | 3 | 3 | 5 |  |  | 5 | 5 | 5 | 5 |  |  |  |  |
| ME |  |  |  | 1 |  |  | 2 |  |  |  | 3 |  |  | 4 | 5 |  |  | 5 |  |  |  |
| WB |  |  |  |  | 1 |  |  | 2 |  |  |  | 3 |  |  | 4 | 5 |  |  | 5 |  |  |

Without Serialization

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F | 1 | 2 | 3 | 4 | 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  | 1 | 2 | 3 | 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| A |  |  | 1 | 2 |  | 2 | 4 |  |  |  |  | 4 |  |  |  |  |  |  |  |  |  |
| M |  |  |  |  | 3 | 3 | 3 | 3 | 3 | 5 |  | 5 | 5 | 5 | 5 |  |  |  |  |  |  |
| ME |  |  |  | 1 |  |  | 2 |  |  | 3 |  |  | 4 |  |  | 5 |  |  |  |  |  |
| WB |  |  |  |  | 1 |  |  | 2 |  |  | 3 |  |  | 4 |  |  | 5 |  |  |  |  |

**Solution 2**: We can demonstrate this by the following sequence of instructions.

I1 :- ADD R3 R2 R1

I2 :- MUL R4 R3 R1

I3 :- ADD R4 R3 R2

With out of order execution, we will have the following timeline

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F | I1 | I2 | I3 |  |  |  |  |  |  |  |  |
| D |  | I1 | I2 | I3 |  |  |  |  |  |  |  |
| ADD |  |  | I1 |  | I3 | I3 |  |  |  |  |  |
| MUL |  |  |  | I2 |  | I2 | I2 | I2 | I2 |  |  |
| MEM |  |  |  | I1 |  |  | I3 |  |  | I2 |  |
| WB |  |  |  |  | I1 |  |  | I3 |  |  | I2 |

As, we can see, that both instruction I2 and I3 will be able to execute as soon as I1 gets evaluated but as I2 is a multiplication operation, I2 requires more cycles to complete the execution, and so, after the execution, architectural register R4 will hold the value of MUL R4 R3 R1 when we were expecting R4 to hold the ADD R4 R3 R2.

**Solution 3.2**:There is a false dependency exit between instructions I7 and I8 as I7’s read from register R4 must return the last value return to register R4 and not the one written after I8 although we are using the same register R4. As this kind of false dependencies, can be overcome by register renaming when we use different physical register while executing the instructions. So we can ignore it.

**Solution 3.3**:

If forwarding happens from execute to execute stage

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ADD R3 R2 R1 | F | D | E | M | WB |  |
| ADD R4 R3 R1 |  | F | D | E | M | WB |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ADD R3 R2 R1 | F | D | E | M | WB |  |
| MUL R4 R5 R1 |  | F | D | E | M | WB |

If forwarding happens from memory to execute stage

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| LOAD R1 R2 #4 | F | D | E | M | WB |  |  |
| ADD R3 R1 R7 |  | F | D | NOP | E | M | WB |

If forwarding happens from writeback to execute stage

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LOAD R1 R2 $4 | F | D | E | M | WB |  |  |  |
| ADD R3 R1 R7 |  | F | D | NOP | NOP | E | M | WB |