**PREM KRISHNA CHETTRI**

**Computer Architecture Assignment 4 Submission Date: 30th Oct ‘15**

**Solution 1.1**. Primarily two factors limit the number of instructions that can be dispatched.

1>Number of functional units available for the execution of an instruction.

2> Amount of parallelism in the instructions.

**Solution 1.2.1** Instruction has to be finally available at architectural register for the ISA to determine the instruction value for further execution. Moreover, we have a limited number of physical register and we needed to free these physical register as soon as it finishes the execution for the other instruction to use it.

**Solution 1.2.2.** Register Alias Table, Rename table, Status bit.

**Solution 1.3.** When flag is false, it means, the register K has not been renamed and when the Boolean flag is true means it is renamed and we have to go and check the register alias table for the new reference number of the architectural register corresponding to this physical register mapping.

**Solution 1.4.** Forwarding is basically forwarding data from one functional unit to all other who are waiting for the results within the same clock cycle, where as completion is the basically the forwarded result written to reorder buffer.

When VFU finishes the execution of an instruction, it waits for the write back bus to be available for the write back and this is when the forwarding happens. Now, if the forwarded result satisfies all the dependencies then the value in the re-order buffer is changed as accordingly and leads to the completion.

Direct precondition is all dependencies are resolved and there is no instruction dependent and waiting for the value for the other instruction in the reorder buffer.

**Solution 1.5.** We have to keep some form of thread Id associated with each of the instruction so that when the instruction retires we will able to associate which thread has issued the instruction for process further execution.

**Solution 1.6.** This is anti dependency (write after read). We can ignore this dependency as it is not a true dependency and for static scheduling and software interlocking, we can use the addition NOP operation to remove this. In out of order CPU, using register renaming, we can avoid any false dependency over its registers. We just assign two different physical register for source and destination R3 while register renaming.

**Solution 2.1.**

All Architecture register is being renamed to physical register and hence renamed code fragment will be.

I1:  MOVI P6, #4000

I2:  MOV P7, P6

I3:  LOAD P8, P7, #9

I4:  SUB P9, P7, P8 // P7, P8 can now be freed

I5:  MUL P7, P9, P6 // P6 can be freed now

I6:  STORE P6, P9, #11

I7:  LOAD P6, P9, #33 // P9 Can be freed now

I8:  AND P8, P6, P7

Grantt Chart CYCLES

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 |
| F | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |
| ISQ |  |  | 1 | 2 | 3 | 4 | 5 4 | 6 5 4 | 7 6 5 | 8 7 | 8 | 8 | 8 | 8 | 8 |  |  |  |  |  |  |  |
| A L U |  |  |  | 1 | 2 |  |  |  | 4 |  |  |  |  |  |  | 8 |  |  |  |  |  |  |
| L S 0 |  |  |  |  |  | 3 |  |  |  | 6 | 7 |  |  |  |  |  |  |  |  |  |  |  |
| L S 1 |  |  |  |  |  |  | 3 |  |  |  | 6 | 7 |  |  |  |  |  |  |  |  |  |  |
| L S 2 |  |  |  |  |  |  |  | 3 |  |  |  | 6 |  |  | 7 |  |  |  |  |  |  |  |
| M0 |  |  |  |  |  |  |  |  |  | 5 |  |  |  |  |  |  |  |  |  |  |  |  |
| M1 |  |  |  |  |  |  |  |  |  |  | 5 |  |  |  |  |  |  |  |  |  |  |  |
| M2 |  |  |  |  |  |  |  |  |  |  |  | 5 |  |  |  |  |  |  |  |  |  |  |
| M3 |  |  |  |  |  |  |  |  |  |  |  |  | 5 |  |  |  |  |  |  |  |  |  |
| RET |  |  |  |  | 1 | 2 |  |  | 3 |  |  |  | 4 | 5 | 6 | 7 |  |  |  |  |  |  |

RAT

CYCLES

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 |  | 6 | 6 | 6 | 6 | 6 | **6** |  |  |  |  |  | 6 | 6 |  |  |  |  |  |  |  |  |
| 2 | 2 |  |  | 7 | 7 | 7 | **7** | 9 | 9 | **9** |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | 4 |  |  |  | 8 | 8 | 8 | 8 | 8 | **8** | 7 | 7 | 7 | 7 |  |  |  |  |  |  |  |  |  |
| 5 | 5 |  |  |  |  |  |  | 6 | 6 | 6 | 6 | 6 | 6 |  |  |  |  |  |  |  |  |  |  |

Allocated

P CYCLES

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 |
| 0 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 |  |  |  |  |  |  |  |  |  |  |  | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  | 1 |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  | 1 |  |  |  |  |  |  |  |  |  |  |  | 1 |  |
| 9 |  |  |  |  |  |  |  |  |  | 1 |  |  | 1 |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Valid

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |  |  |
| 7 |  |  |  |  |  |  |  | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  | 1 | 0 | 0 |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**Solution 2.9.** With respect to the following chart of Issue Queue. Each instruction will have the following dispatch -> issue cycle

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 |
| ISQ |  |  | 1 | 2 | 3 | 4 | 5 4 | 6 5 4 | 7 6 5 | 8 7 | 8 | 8 | 8 |

I1 :- 3 (Spends only 1 cycle)

I2 : 4

I3 : 5

I4 :- 6 -> 8

I5 :- 7 -> 9

I6 :- 8 -> 9

I7 :- 9 -> 10

I8 :- 10 -> 13

**Solution 3.1.**

2 Operand ISA

MOV R1, R2

SUB R1, R3

MUL R2, R1

LDI R1, #100

ADD R5, R4

MOV R6, R5

AND R6, R4

XOR R5, R6

LDI R6, #100

**Solution 3.2.**

**Renamed 3-Operand Instructions**

SUB P32, P2, P3

MUL P33, P32, P2

LDI P32, #100 // Release P32 after this

ADD P32, P5, P33

ADD P34, P32, P33 // Release P33 after this

XOR P33, P34, P32

LDI P34, #100

**Solution 3.3.**

**Renamed 2-operand Instructions.**

MOV P32, P2

SUB P32, P3

MUL P2, P32

LDI P32, #100 // Release P32 After this

ADD P5, P4

MOV P32, P5

AND P32, P4

XOR P5, P32

LDI P32, #100

**Solution 3.4.** The one way to implement this is the detection of the register content. We can use the mechanism where if we detect that either of the source register (src1 or src2 ) in 3 operand instruction like ADD dst src1 src2 is zero, we rather than passing through the pipeline to add these value, we will directly copy the non-zero source content to destination register directly. This way we will completely remove the execution step of ADD in a pipeline.

Example, for instruction ADD R1 R2 R3 , if we could detect R2 as zero content register, we will not issue ADD instruction to pipeline, but we will copy content of R3 (non-zero here), to the write back bus of the processor and fill R1.

**Solution 3.5.** As from the paper, we found some benchmark, which explains that, for zeroing a register; add operation is slower then subtracting the register content. However, the best way to zeroing the register is by x-or to itself.

However, the problem with x-or a register to itself is, if there is a data dependency, it has to wait till all the dependencies resolves before it goes for the xor operation. Now, if we can detect that the dependency is actually a false dependency, we can execute this xor operation.

So, if by the usage of the register renaming, we can resolve false dependency and we can get two physical register with same value and we can x-or them to produce the zero valued register. The paper further explains the implementation in Sandy Bridge processor, where once we found that there is no data dependencies among the registers, rather then passing through the execution stage to do the x-or operation, we can integrate a mechanism in renaming register to detect the condition that always zeros the register and rather then processing through the execution steps, it can directly zero the register itself.

So, the way the zero instruction could be implemented with minimal overhead is to detect a condition in which if we know that there is no data dependency and resolves that it is always going to zero the register, set the register value to zero without execution through pipeline.