ILP III

October 6, 2004

1 Out of Order Execution

Figure 1 shows the different stages in our pipeline. We would be looking at the *scoreboard, rename* and *reorder* units in this lecture.

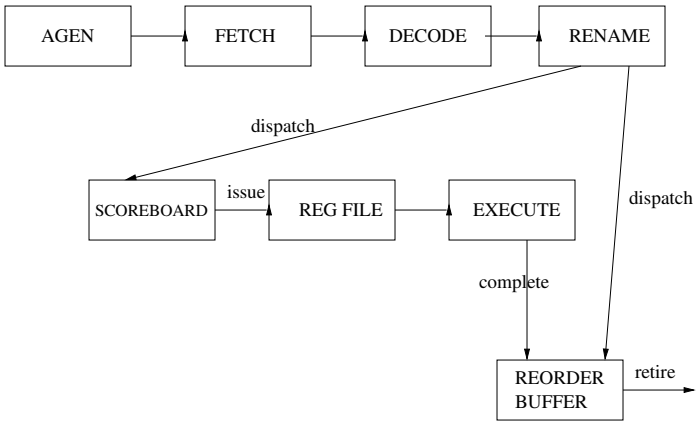

Figure 1: Pipeline Stages

Table 1 shows the instructions in the scoreboard at any given time (assume t=0), and Figure 2 shows the dependency graph of the following instructions set.

Rules for Issue (From Scoreboard to Register File)

1. All previous instructions that write my sources have completed (*True Dependency*)
2. All previous instructions that write my destination have completed (*Output Dependency*)
3. All previous instructions that read my destination have issued (*Anti Dependency*)

Instruction	Number of Cycles to complete
$MUL : R_m < -R_a, R_b$	10
$ADD : R_n < -R_c, R_d$	5
$SUB : R_x < -R_m, R_n$	5
$ADD : R_m < -R_r, R_s$	5
$ADD : R_n < -R_v, R_t$	5
$DIV : R_y < -R_m, R_n$	20

Table 1: Instructions in a Scoreboard at any given time *t=0*

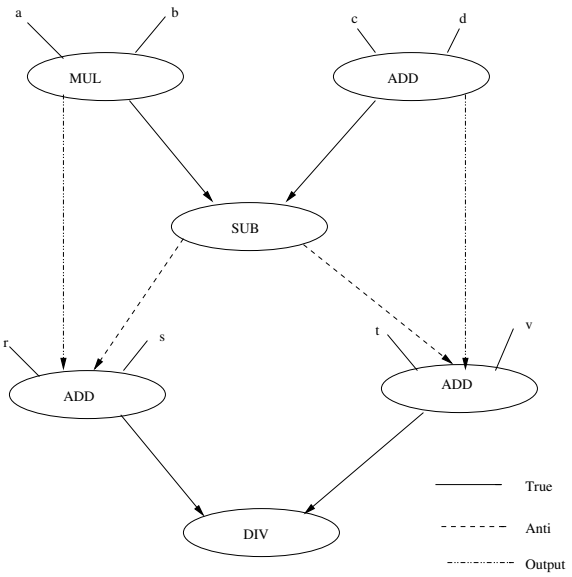

Figure 2: Dependency Graph

Instruction	True	Output	Anti	Issue	Complete
$MUL_{10} : R_m < -R_a, R_b$	1	1	1	1	11
$ADD_5 : R_n < -R_c, R_d$	1	1	1	2	7
$SUB_5 : R_x < -R_m, R_n$	11	1	1	11	16
$ADD_5 : R_m < -R_r, R_s$	1	11	12	12	17
$ADD_5 : R_n < -R_v, R_t$	1	7	12	13	18
$DIV_{20} : R_y < -R_m, R_n$	18	1	1	18	38

Table 2: Analysis of the instructions given the Rules stated above

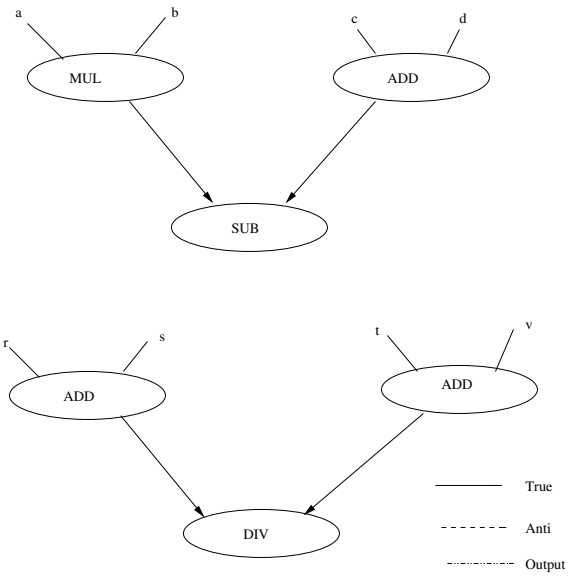

Figure 3: Dependency Graph after Renaming

Instruction	Number of Cycles to complete
$MUL : R_m < -R_a, R_b$	10
$ADD : R_n < -R_c, R_d$	5
$SUB : R_x < -R_m, R_n$	5
$ADD : \mathbf{R_p} < -R_r, R_s$	5
$ADD : \mathbf{R_q} < -R_v, R_t$	5
$DIV : R_y < -\mathbf{R_p}, \mathbf{R_q}$	20

Table 3: Instructions in a Scoreboard with Renaming

We are assuming that, everything happens at clock edges.

The various anaylysis of the dependancies and issuing of the various instructions is shown in Table 2. MUL issues on cycle 1, then the completion for the MUL instruction would happen during cycle 11 and it would be finished at cycle 11. The value of register R_m would be written at cycle 11. If any instruction is issued on cycle 11, then it would be latched onto 11, and if any instruction writing during cycle 11, then it values would be read out on cycle 12. It is okay to issue instruction during the same period as values are being written to the register file.

2 Renaming

In the above example, we are using 12 registers. Let us suppose our machine has more than 12 registers, let's assume 14 registers (R_p, R_q as the additional two registers). Our scoreboard would then look like as shown in Table 3, and our dependency graph would be as shown in Figure 3. We can see that the *anti* and *output* dependencies have been removed.

From Table 4 we can see that the instructions have issued in a completely different order as from Table 2. The order of completion is also completely different. The total cycles taken is 29 as compared to 38 in the previous case. In the above example, we have made the assumption that *prioritize oldest instruction for issue*. This is not a necessary condition, we could have done better with another issue heuristic.

Instruction	True	Output	Anti	Issue	Complete
$MUL_{10} : R_m < -R_a, R_b$	1	1	1	1	11
$ADD_5 : R_n < -R_c, R_d$	1	1	1	2	7
$SUB_5 : R_x < -R_m, R_n$	11	1	1	11	16
$ADD_5 : R_p < -R_r, R_s$	1	1	1	3	8
$ADD_5 : R_q < -R_v, R_t$	1	1	1	4	9
$DIV_{20} : R_y < -R_p, R_q$	9	1	1	9	29

Table 4: Analysis of the instructions after Renaming

Arch Registers R	Physical Registers P
R_a	1
R_b	2
R_c	3
R_d	4
R_m	$5 \rightarrow 13 \rightarrow 16$
R_n	$6 \rightarrow 14 \rightarrow 17$
R_r	7
R_s	8
R_t	9
R_v	10
R_x	$11 \rightarrow 15$
R_y	$12 \rightarrow 18$

Table 5: Rename Alias Table

2.1 Rename Alias Table

So far, we have assumed that only arithmetic instructions are there in our pipeline.

Let us make another *outrageous* assumption: *We have infinite number of registers and every instruction register has a different destination register*. Making this assumption, we would never have any *output* dependency and also no *anti* dependency.

We can classify registers into two types:

1. Architecture Registers (R): The registers names in the ISA
2. Physical Registers (P): The names of the actual registers in the register file. On a typical machine $P \geq R$

During renaming we need to do a little bookkeeping. This book-keeping is done using the *Rename Alias Table* at the rename stage as shown in Table 5. Along with this a *Free list* is also maintained, which has all the physical registers which are not currently assigned. From Table 5 we can see that R_m was earlier assigned P_5, but then when it was used as a destination register ($MUL : R_m \leftarrow R_a(P_1), R_b(P_2)$), then it is assigned a *new* register from the Free list, which happens to be be P_{13}. Again, when the instruction ($ADD : R_m \leftarrow R_r(P_5), R_s(P_6)$) is decoded, R_m is again remapped to another physical register P_{16} from the free list.

If we have enough registers, every instructions would be writing to a different registers and there would be no *output* and *anti* dependency.

3 Reorder Buffer

Instructions come into the Re-order Buffer(ROB) from the Rename (after "dispatch") and instructions at the head of the queue leave when complete. So, instructions leave the Reorder Buffer in same order as they were at the Rename stage.

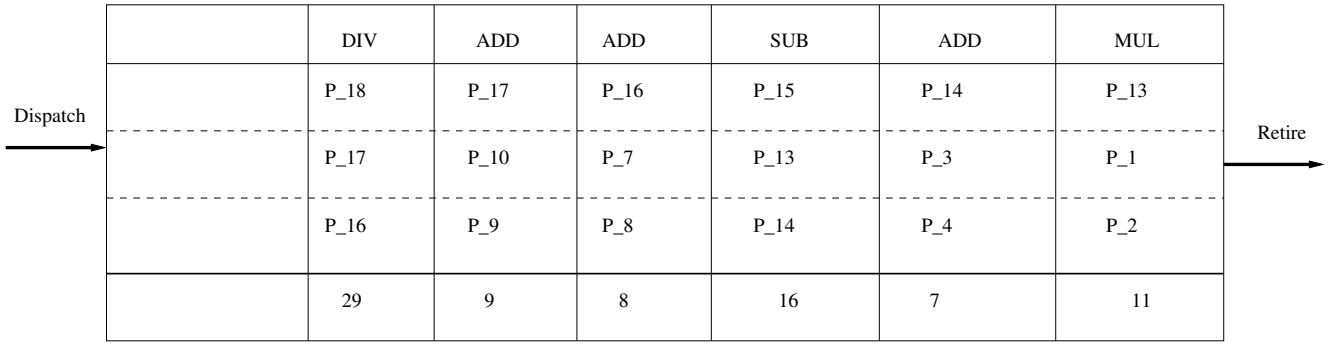

Figure 4: Reorder Buffer

The ROB is a FIFO. It looks at the instruction at the head and if nothing else has gone wrong[1], then it retires the instruction. Figure 4 shows the retiring order of the various instructions used in our example. As the instructions retire, they add the physical registers earlier assigned to them back to the free list. At Cycle 11, the instruction at the head of the ROB would have completed, and so would retire at cycle 12. At this point, P_5 can be safely put onto the free list, as it SUB which was at the head of the queue was using P_{13} and not P_5, and so instructions using P_5 would have already completed. ADD would retire at cycle 13, as it already completed at cycle 7, and P_6 would be put back on the free list.

This is called the **Tomasulo's Algorithm**.

[1] e.g. branch mispredict