

## CSE 30321 – Lecture 17 – In Class Handout

### Example 5:

Assume the following:

- 25% of instructions are loads → 50% of the time, the *next* instruction uses the loaded value
- 13% of instructions are stores
- 19% of instructions are conditional branches
- 2% of instructions are unconditional branches
- 43% of instructions are something else

Also...

- You have a 5 stage pipeline with forwarding
- There is a 1 CC penalty if an instruction immediately needs a loaded value
- We have added extra hardware to resolve a jump/branch instruction in the decode stage
  - o Therefore, there is just a 1 CC penalty
- 75% of conditional branches are predicted correctly

What is the CPI of our pipeline?

- $0.23 \times 0.5 \times 1 = 0.115$ 
  - o 23% of the time we have a lw and 50% of those times, we need the result right away
- $0.02 \times 1 = 0.02$ 
  - o 2% of the time we have a jump and have a 1 CC penalty
- $0.25 \times 0.19 \times 1 = 0.0475$ 
  - o 25% of the time we guess wrong on our branch and have a 1 CC penalty

Therefore  $0.115 + 0.02 + 0.0475 = 0.1825$ , If our ideal CPI is 1, then our new CPI is 1.1825

### Example 6:

- Assume that **forwarding HAS been implemented**
- We will stall if we encounter a branch instruction
- Branches or Jumps are resolved after the EX stage.
- Assume that register \$2 has the value of 0 and \$3 has the value of 0

| Instruction                    | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14  | 15 | 16 | 17 |
|--------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|----|----|----|
| <b>LW</b><br>\$1, 4(\$9)       | IF | ID | EX | M  | W  |    |    |    |    |    |    |    |    |   |    |    |    |
| <b>Add</b><br>\$4, \$1, \$9    |    | IF | ID | ID | EX | M  | W  |    |    |    |    |    |    | <b>Add gets data from lw forwarding</b>                               |    |    |    |
| <b>Sub</b><br>\$7, \$4, \$9    |    |    | IF | IF | ID | EX | M  | WB |    |    |    |    |    | <b>Sub gets data from add forwarding</b>                              |    |    |    |
| <b>BEQ</b><br>\$2, \$3, X      |    |    |    |    | IF | ID | EX |    |    |    |    |    |    | <b>BEQ must still wait to enter the pipeline</b>                      |    |    |    |
| <b>Add</b><br>\$9, \$8, \$7    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |    |    |    |
| <b>And</b><br>\$4, \$5, \$5    |    |    |    |    |    |    |    |    |    |    |    |    |    |   |    |    |    |
| <b>X: Add</b><br>\$4, \$5, \$9 |    |    |    |    |    |    |    | IF | ID | EX | M  | WB |    | <b>Can't start Add until after BEQ finishes executing (comparing)</b> |    |    |    |

Example 7:

- Assume that **forwarding HAS been implemented**
- We will predict that any branch instruction is **NOT TAKEN**
- Branches or Jumps are resolved after the EX stage.
- Assume that register \$2 has the value of 0 and \$3 has the value of 0

| Instruction                    | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12  | 13 | 14 | 15 | 16 | 17 |
|--------------------------------|----|----|----|----|----|----|----|----|----|----|----|---|----|----|----|----|----|
| <b>LW</b><br>\$1, 4(\$9)       | IF | ID | EX | M  | W  |    |    |    |    |    |    |   |    |    |    |    |    |
| <b>Add</b><br>\$4, \$1, \$9    |    | IF | ID | ID | EX | M  | W  |    |    |    |    |   |    |    |    |    |    |
| <b>Sub</b><br>\$7, \$4, \$9    |    |    | IF | IF | ID | EX | M  | WB |    |    |    |   |    |    |    |    |    |
| <b>BEQ</b><br>\$2, \$3, X      |    |    |    |    | IF | ID | EX |    |    |    |    |   |    |    |    |    |    |
| <b>Add</b><br>\$9, \$8, \$7    |    |    |    |    |    | IF | ID |    |    |    |    | <b>Add and And start down pipeline; however, they would not change state until CC 10 and 11. They never get this far so there is no harm done. We can kill them and restart the next add instruction.</b> |    |    |    |    |    |
| <b>And</b><br>\$4, \$5, \$5    |    |    |    |    |    |    | IF |    |    |    |    |   |    |    |    |    |    |
| <b>X: Add</b><br>\$4, \$5, \$9 |    |    |    |    |    |    |    | IF | ID | EX | M  | W   |    |    |    |    |    |

Example 8:

- Assume that **forwarding HAS been implemented**
- We will predict that any branch instruction is **TAKEN**
- Branches or jumps are resolved after the EX stage.
- Assume that register \$2 has the value of 0 and \$3 has the value of 0

| Instruction                    | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|--------------------------------|----|----|----|----|----|----|----|----|---|----|----|----|----|----|----|----|----|
| <b>LW</b><br>\$1, 4(\$9)       | IF | ID | EX | M  | W  |    |    |    |   |    |    |    |    |    |    |    |    |
| <b>Add</b><br>\$4, \$1, \$9    |    | IF | ID | ID | EX | M  | W  |    |   |    |    |    |    |    |    |    |    |
| <b>Sub</b><br>\$7, \$4, \$9    |    |    | IF | IF | ID | EX | M  | WB |   |    |    |    |    |    |    |    |    |
| <b>BEQ</b><br>\$2, \$3, X      |    |    |    |    | IF | ID | EX |    |   |    |    |    |    |    |    |    |    |
| <b>Add</b><br>\$9, \$8, \$7    |    |    |    |    |    |    |    |    |   |    |    |    |    |    |    |    |    |
| <b>And</b><br>\$4, \$5, \$5    |    |    |    |    |    |    |    |    |   |    |    |    |    |    |    |    |    |
| <b>X: Add</b><br>\$4, \$5, \$9 |    |    |    |    |    | IF | ID | EX | M | W  |    |    |    |    |    |    |    |

This is the best situation – the last add instruction finishes 2 CC's earlier.

Example 9:

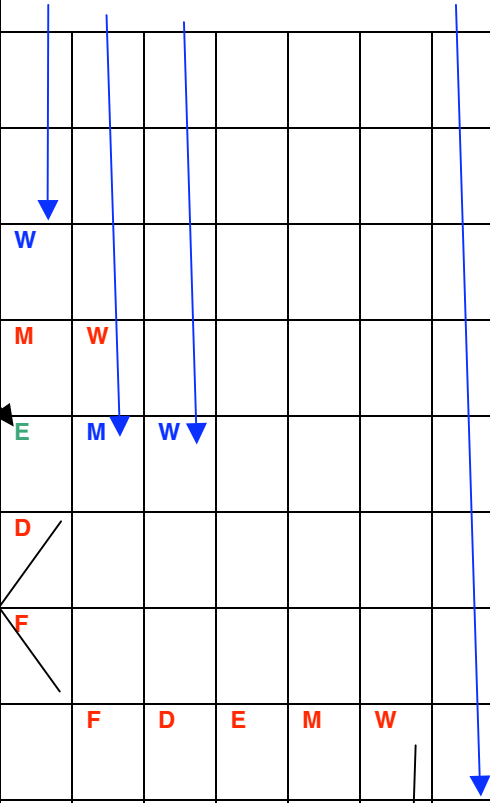
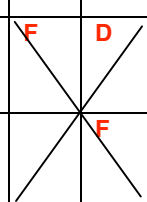
For the sequence of instructions shown below, show how they would progress through the pipeline.

Part 1:

- Assume that **forwarding HAS been implemented**
- We will predict that any branch instruction is **NOT TAKEN**
- Branches or Jumps are resolved after the EX stage.
- Assume that register \$8 does not equal \$1 for the 1<sup>st</sup> Beq instruction
- Assume that register \$17 does equal \$26 for the 2<sup>nd</sup> Beq instruction

| Instruction                     | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17  |
|---------------------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|-----|
| <b>SUB</b><br>\$1, \$2, \$3     | F | D | E | M | W |   |   |   |   |    |    |    |    |    |    |    |     |
| <b>Add</b><br>\$8, \$9, \$10    |   | F | D | E | M | W |   |   |   |    |    |    |    |    |    |    |     |
| <b>Beq</b><br>\$1, \$8, X       |   |   | F | D | E | M | W |   |   |    |    |    |    |    |    |    |     |
| <b>Lw</b><br>\$7, 0(\$20)       |   |   |   | F | D | E | M | W |   |    |    |    |    |    |    |    |     |
| <b>Add</b><br>\$11, \$7, \$12   |   |   |   |   | F | D | D | E | M | W  |    |    |    |    |    |    |     |
| <b>Sw</b><br>\$11, 0(\$24)      |   |   |   |   |   | F | F | D | E | M  | W  |    |    |    |    |    |     |
| <b>X: Addi</b><br>\$17, \$17, 1 |   |   |   |   |   |   | F | D | E | M  | W  |    |    |    |    |    |     |
| <b>Beq</b><br>\$17, \$26, Y     |   |   |   |   |   |   |   | F | D | E  | M  | W  |    |    |    |    |     |
| <b>Sub</b><br>\$5, \$6, \$7     |   |   |   |   |   |   |   |   | F | D  |    |    |    |    |    |    |     |
| <b>Or</b><br>\$8, \$5, \$5      |   |   |   |   |   |   |   |   |   | F  |    |    |    |    |    |    |     |
| <b>Y: Addi</b><br>\$17, \$17, 1 |   |   |   |   |   |   |   |   |   |    | F  | D  | E  | M  | W  |    |     |
| <b>Sw</b><br>\$17, 0(\$10)      |   |   |   |   |   |   |   |   |   |    |    | F  | D  | E  | M  | W  |     |
| <b>SUB</b><br>\$1, \$2, \$3     |   |   |   |   |   |   |   |   |   |    |    |    |    | F  | D  | E  | ... |
| <b>Add</b><br>\$8, \$9, \$10    |   |   |   |   |   |   |   |   |   |    |    |    |    |    | F  | D  | ... |

Technically, nothing done, but can think of instruction as progressing through pipeline



Part 2:

- (i) Assume that this sequence of code is executed 100 times. How many cycles does the pipelined implementation take?
- (ii) How many cycles would this code take in a multi-cycle implementation?
  
- From Part 1, you can see that it takes 17 clock cycles to execute 12 instructions.
- However, we can start the next "iteration" in clock cycle 14. Therefore, it *really* only takes 13 cycles for each iteration and 17 CCs for the last one.
- Therefore, iterations 1 through 99 take 13 CCs each
  - o  $(13 \times 99 = 1287 \text{ CCs})$
- Iteration 100 takes 17 CCs
- Therefore  $1287 \text{ CCs} + 17 \text{ CCs} = 1304 \text{ CCs}$
  
- For the multi-cycle implementation, we have:
  - o 9 instructions that take 4 CCs
  - o 2 instructions that take 3 CCs
  - o 1 instruction that takes 5 CCs
- Therefore, each "iteration" takes:  $(9 \times 4) + (2 \times 3) + (1 \times 5) = 36 + 6 + 5 = 47 \text{ CCs}$
- If there are 100 iterations, then 4700 CCs are required

Pipelining gives us a speed up of  $4700 / 1304 = 3.6$  for this implementation

- Little to no extra HW is needed!