Pipeline Implementation/Operation

Troy Veldhuizen

6. **a. For each RAW dependency listed above, give a sequence of at least 3 assembly statements that exhibits that dependency.**

   - EX to 1st only

     ```
     add $r3, $r4, $r5
     add $r6, $r2, $r3
     add $r7, $r8, $r9
     ```

   - MEM to 1st only

     ```
     lw $a0, 8($sp)
     add $r3, $r4, $a0
     add $r7, $r8, $r9
     ```

   - EX to 2nd only

     ```
     add $r3, $r4, $r5
     add $r7, $r8, $r9
     add $r6, $r3, $r2
     ```

   - MEM to 2nd only

     ```
     lw $a0, 8($sp)
     add $r7, $r8, $r9
     add $r3, $r4, $a0
     ```

   - EX to 1st and EX to 2nd

     ```
     add $r3, $r4, $r5
     add $r6, $r7, $r3
     add $r8, $r9, $r3
     ```

     b. For each RAW dependency above, how many nops would need to be inserted to allow your code from 6a to run correctly on a pipeline with no forwarding or hazard detection? show where the nops could be inserted.

   - EX to 1st only

     ```
     add $r3, $r4, $r5
     nop
     ```

```
nop
add $r6, $r2, $r3
add $r7, $r8, $r9
```

- MEM to 1st only

```
lw $a0, 8($sp)
nop
nop
add $r3, $r4, $a0
add $r7, $r8, $r9
```

- EX to 2nd only

```
add $r3, $r4, $r5
nop
add $r7, $r8, $r9
add $r6, $r3, $r2
```

- MEM to 2nd only

```
lw $a0, 8($sp)
nop
add $r7, $r8, $r9
add $r3, $r4, $a0
```

- EX to 1st and EX to 2nd

```
add $r3, $r4, $r5
nop
nop
add $r6, $r7, $r3
add $r8, $r9, $r3
```

c. Analyzing each instruction independently will over-count the number of nops needed to run a program on a pipeline with no forwarding or hazard detection. Write a sequence of three assembly instructions so that, when you consider each instruction in the sequence independently, the sum of the stalls is larger than the number of stalls the sequence actually needs to avoid data hazards.

```
add $r3, $r4, $r5
add $r6, $r7, $r3
add $r8, $r9, $r3
```

If nops are added between the first and second instruction, the last instruction no longer needs any error handleing to work correctly.

d. Assuming no other hazards, what is the CPI for the program described by the table above when run on a pipeline

with no forwarding? What percent of cycles are stalls? (For simplicity, assume that all necessary cases are listed above and can be treated independently.) Extra instructions added on average for each error case.

$$\begin{aligned} EX1 = (0.05)(2) &= 0.1 \\ MEM1 = (0.2)(2) &= 0.4 \\ EX2 = (0.05)(1) &= 0.05 \\ MEM2 = (0.1)(1) &= 0.1 \\ EX12 = (0.1)(2) &= 0.2 \end{aligned}$$

Add average error to 1 to get total adjusted CPI

$$CPI = 0.1 + 0.4 + 0.05 + 0.1 + 0.2 + 1 = 1.85$$

e. What is the CPI if we use full forwarding(forward all results that can be forwarded)? What percent of cycles are stalls? The loads can not be completely solved by forwarding. A RAW error can occur if an instruction immediately follows a load. Because of this, the MEM to 1st Only case is unsolved by forwarding. However, we now only need 1 stall to solve the RAW error. 20% of cycles will result in stalls. Using this we can calculate the CPI.

$$CPI = (0.2)(1) + 1 = 1.2$$

f. Let us assume that we cannot afford to have three-input multiplexors that are needed for full forwarding. We have to decide if it is better to forward only from the EX/MEM pipeline register (next-cycle forwarding) or only from the MEM/WB pipeline register (two-cycle forwarding). What is the CPI for each option?

- EX/MEM: This will solve the EX1 and EX2 cases. We are completely out of luck with the MEM1 and MEM2 cases.

$$\begin{aligned} MEM1 = (0.2)(2) &= 0.4 \\ MEM2 = (0.1)(1) &= 0.1 \\ CPI = 0.1 + 0.4 + 1 &= 1.5 \end{aligned}$$

- MEM/WB case: This will solve MEM2 errors and help with MEM1 errors, but will not help EX1 and EX2 errors.

$$\begin{aligned} MEM1 = (0.2)(1) &= 0.2 \\ EX1 = (0.05)(2) &= 0.1 \\ EX2 = (0.05)(1) &= 0.05 \\ CPI = 0.2 + 0.1 + 0.05 + 1 &= 1.35 \end{aligned}$$

g. For the given hazard probabilities and pipeline stage latencies, what is the speedup achieved by adding each type of forwarding (EX/MEM, MEM/WB, or full) to a pipeline that has no forwarding?

- EX/MEM :

$$\begin{aligned} speedup &= \frac{oldtime}{newtime} \\ &= \frac{n * p * cpi}{n * p * cpi} \\ &= \frac{120 * 1.85}{120 * 1.5} \\ &= \frac{1.85}{1.5} \\ speedup &= 1.23 \end{aligned}$$

- MEM/WB :

$$\begin{aligned} speedup &= \frac{oldtime}{newtime} \\ &= \frac{n * p * cpi}{n * p * cpi} \\ &= \frac{120 * 1.85}{120 * 1.35} \\ &= \frac{1.85}{1.35} \\ speedup &= 1.37 \end{aligned}$$

- Full:

$$\begin{aligned} speedup &= \frac{oldtime}{newtime} \\ &= \frac{n * p * cpi}{n * p * cpi} \\ &= \frac{120 * 1.85}{130 * 1.2} \\ &= \frac{222}{156} \\ speedup &= 1.42 \end{aligned}$$

h. What would be the additional speedup (relative to the fastest processor from part 6g) if we added time-travel forwarding that eliminates all data hazards? Assume that the yetto-be-invented time-travel circuitry adds 100 ps to the

latency of the full-forwarding EX stage.

$$\begin{aligned} speedup &= \frac{oldtime}{newtime} \\ &= \frac{n * p * cpi}{n * p * cpi} \\ &= \frac{130 * 1.2}{230 * 1} \\ &= \frac{156}{230} \\ speedup &= 0.68 \end{aligned}$$

i. The table of hazard types has separate entries for "EX to 1st" and "EX to 1st and EX to 2nd". Why is there no entry for "MEM to 1st and MEM to 2nd"? The "MEM to 2nd" would involve using a value that was placed in a register from a load instruction 2 instructions previous. Because the value is stored in a register now, it is no longer a load issue, because we can just use the value that is in the register. The instruction has already done the work of moving it in.