ICS 233 – Computer Architecture & Assembly Language

Assignment 7: Pipelined Processor

Solution

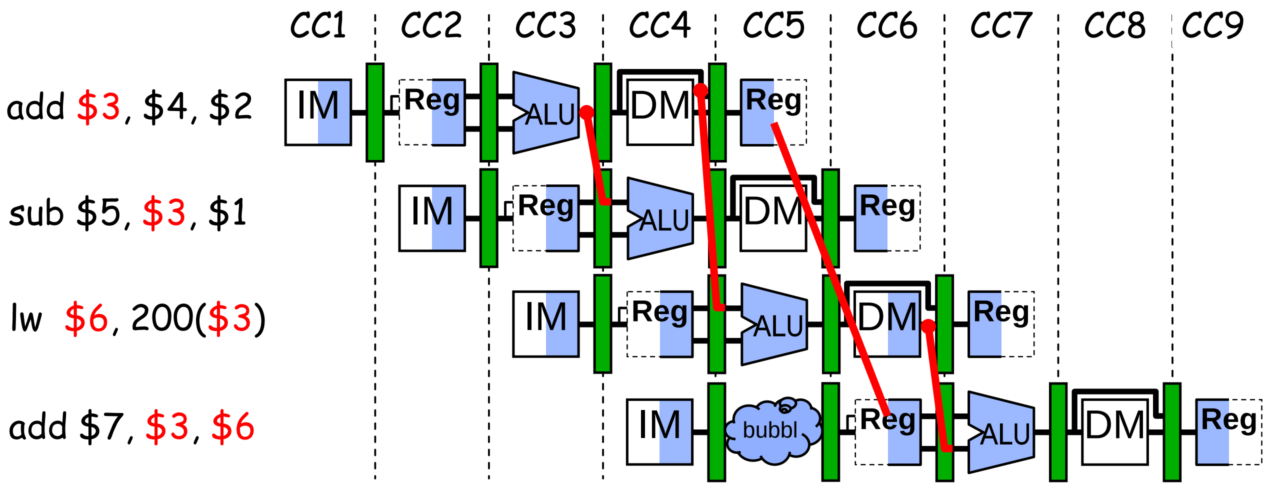
1. **(4 pts)** Identify all the RAW data dependencies in the following code. Which dependencies are data hazards that will be resolved by forwarding? Which dependencies are data hazards that will cause a stall? Using a graphical representation of the pipeline, show the forwarding paths and stalled cycles if any.

```
add $3, $4, $2
sub $5, $3, $1
lw  $6, 200($3)
add $7, $3, $6
```

Solution:

RAW dependencies:

```
add $3, $4, $2   and    sub $5, $3, $1    (forwarding)
add $3, $4, $2   and    lw  $6, 200($3)   (forwarding)
lw  $6, 200($3)  and    add $7, $3, $6    (stall 1, forward)
add $3, $4, $2   and    add $7, $3, $6    (from register)
```



2. **(4 pts)** We have a program of 10^6 instructions in the format of "**lw,add,lw,add,…**". The **add** instruction depends only on the **lw** instruction right before it. The **lw** instruction also depends only on the **add** instruction right before it. If this program is executed on the 5-stage MIPS pipeline:

   a) Without forwarding, what would be the actual CPI?
   b) With forwarding, what would be the actual CPI?

Solution:

a) **Without forwarding, the value being written into a register can only be read in the same cycle. As a result, there will be a bubble of 2 cycles between a LW and the dependent ADD to allow the LW to progress through the MEM and WB stages. Similarly, there will be a bubble of 2 cycles between an ADD and the dependent LW.**

   Therefore, it takes 6 cycles on average to complete one LW and one ADD.
   1 cycle (to complete LW) + 2 cycles (bubbles) + 1 cycle (to complete ADD) + 2 cycles (bubbles) = 6 cycles

   So, it takes 6 cycles to complete 2 instructions
   Average CPI = 6/2 = 3.

b) **With forwarding, there will be a bubble of 1 cycle between a LW and the dependent ADD. However, no bubble exists between an ADD and the dependent LW.**

   Therefore, it takes only 3 cycles on average to to complete one LW and one ADD.
   1 cycle (to complete LW) + 1 cycle (bubble) + 1 cycle (to complete ADD) = 3 cycles

   So, it takes 3 cycles to complete 2 instructions
   Average CPI = 3/2 = 1.5.

3. **(4 pts)** A 10-stage instruction pipeline runs at a clock rate of 1 GHz. The instruction mix is such that 15% of instructions cause one bubble to be inserted into the pipeline, and 10% of instructions cause two bubbles to be inserted. The equivalent single-cycle implementation would lead to a clock rate of 150 MHz.

a) What is the increase in the pipeline CPI over the ideal CPI as a result of bubbles?
b) What is the speedup of pipelined implementation over single-cycle?

Solution:

a) **Ideal pipeline CPI = 1 cycle per instruction (if no bubbles)**

   Increase in CPI due to bubbles = 0.15 * 1 + 0.1 * 2 = 0.35 cycles per instruction

   Pipeline CPI with bubbles = 1 + 0.35 = 1.35 (35% increase over ideal CPI)

b) **Speedup of pipelined implementation =**

   (Pipeline Clock Rate / Single-Cycle Clock) * (Single-Cycle CPI / Pipeline CPI) =

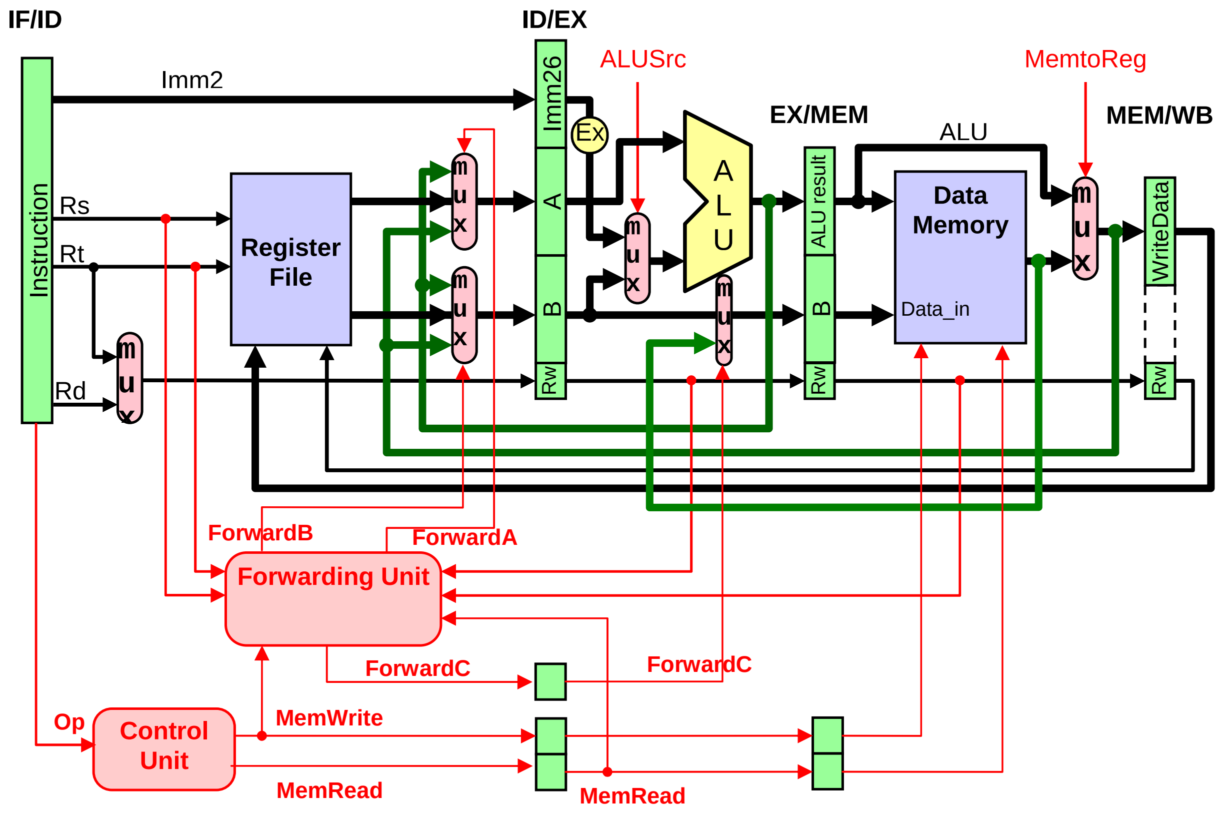
   (1000 MHz / 150 MHz) * (1 / 1.35) = 4.94

4. **(4 pts) Store-after-load data dependence.** Consider copying an array of *n* words from one address in memory to another. This can be accomplished by placing a sequence of **lw** and **sw** instructions in a loop, with each loop iteration copying one word. In the current pipelined implementation shown in the lecture slides, this leads to one bubble (stall cycle) between **lw** and **sw**. Is it possible to avoid this stalling via additional data forwarding hardware? Discuss how this can be done or explain how the bubble is unavoidable.

Solution: Yes, forwarding is possible and we can avoid stalling the pipeline. Consider:

```
LW   $8, ...       # LW instruction writes $8
SW   $8, ...       # SW instruction uses $8
```

We need a multiplexer at the input of EX/MEM.B register as show below. The data read from the data memory in the MEM stage should be fed back at the input of this multiplexer. A control signal "ForwardC" is needed to control the selection of this multiplexer. The Forwarding unit in the DECODE stage will generate the "ForwardC" signal and pipeline it, after detecting the dependency between a SW and a previous LW instruction. The SW instruction is currently in the DECODE stage (MemWrite = 1). The LW instruction is in the EXE stage (ID/EX.MemRead = 1). The ID/EX.RW register for the LW instruction contains the same register number as Rt for the SW instruction. The Forwarding Unit can detect this situation and generate the ForwardC signal.



5. **(4 pts)** We have a program core consisting of five conditional branches. The program core will be executed millions of times. Below are the outcomes of each branch for one execution of the program core (T for taken and N for not taken).

   Branch 1: T-T-T
   Branch 2: N-N-N-N
   Branch 3: T-N-T-N-T-N
   Branch 4: T-T-T-N-T
   Branch 5: T-T-N-T-T-N-T

   Assume that the behavior of each branch remains the same for each program core execution. For dynamic branch prediction schemes, assume that each branch has its own prediction buffer and each buffer is initialized to the same state before each execution. List the predictions and the accuracies for each of the following branch prediction schemes:

   a) Always taken
   b) Always not taken
   c) 1-bit predictor, initialized to predict taken
   d) 2-bit predictor, initialized to weakly predict taken

Solution:

Prediction accuracy = 100% * Correct Predictions / Total Branches

a) **Branch 1: prediction: T-T-T, right = 3, wrong = 0**
   Branch 2: prediction: T-T-T-T, right = 0, wrong = 4
   Branch 3: prediction: T-T-T-T-T-T, right = 3, wrong = 3
   Branch 4: prediction: T-T-T-T-T, right = 4, wrong = 1
   Branch 5: prediction: T-T-T-T-T-T-T, right = 5, wrong = 2
   Total right = 15, Total wrong = 10, Accuracy = 100% * 15/25 = 60%

b) **Branch 1: prediction: N-N-N, right = 0, wrong = 3**
   Branch 2: prediction: N-N-N-N, right = 4, wrong = 0
   Branch 3: prediction: N-N-N-N-N-N, right = 3, wrong = 3
   Branch 4: prediction: N-N-N-N-N, right = 1, wrong = 4
   Branch 5: prediction: N-N-N-N-N-N-N, right = 2, wrong = 5
   Total right = 10, Total wrong = 15, Accuracy = 100% * 10/25 = 40%

c) **Branch 1: prediction: T-T-T, right = 3, wrong = 0**
   Branch 2: prediction: T-N-N-N, right = 3, wrong = 1
   Branch 3: prediction: T-T-N-T-N-T, right = 1, wrong = 5
   Branch 4: prediction: T-T-T-T-N, right = 3, wrong = 2
   Branch 5: prediction: T-T-T-N-T-T-N, right = 3, wrong = 4
   Total right = 13, Total wrong = 12, Accuracy = 100% * 13/25 = 52%

d) **Branch 1: prediction: T-T-T, right = 3, wrong = 0**
   Branch 2: prediction: T-N-N-N, right = 3, wrong = 1
   Branch 3: prediction: T-T-T-T-T-T, right = 3, wrong = 3
   Branch 4: prediction: T-T-T-T-T, right = 4, wrong = 1
   Branch 5: prediction: T-T-T-T-T-T-T, right = 5, wrong = 2
Total right = 18, Total wrong = 7, Accuracy = 100% * 18/25 = 72%