25 – 2 Computer Architecture Homework #3

Due: 6/09 17:00 p.m

1.

4.13 Examine the difficulty of adding a proposed `ss rt, rs, imm` (Store Sum) instruction to MIPS.

Interpretation: `Mem[Reg[rt]]=Reg[rs]+immediate`

4.13.1 [10] <§4.4> Which new functional blocks (if any) do we need for this instruction?

4.13.2 [10] <§4.4> Which existing functional blocks (ifany) require modification?

4.13.3 [5] <§4.4> What new data paths do we need (if any) to support this instruction?

4.13.4 [5] <§4.4> What new signals do we need (if any) from the control unit to support this instruction?

4.13.5 [5] <§4.4> Modify Figure 4.21 to demonstrate an implementation of this new instruction.

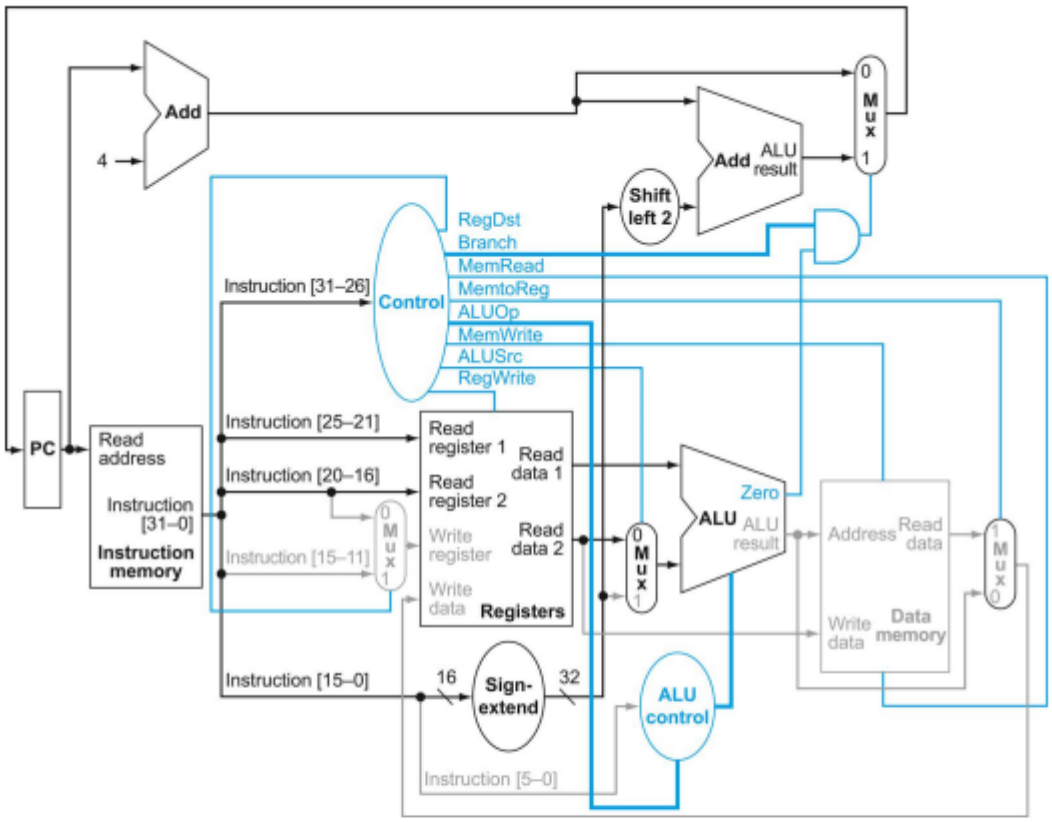

FIGURE 4.21 The datapath in operation for a branch-on-equal instruction.
The control lines, datapath units, and connections that are active are highlighted. After using the register file and ALU to perform the compare, the Zero output is used to select the next program counter from between the two candidates.

Figure 4.21 shows the four steps in execution:

1. An instruction is fetched from the instruction memory, and the PC is incremented.

2. Two registers, $t1 and $t2, are read from the register file.

3. The ALU performs a subtract on the data values read from the register file. The value of PC + 4 is added to the signextended, lower 16 bits of the instruction (offset) shifted left by two; the result is the branch target address.

4. The Zero result from the ALU is used to decide which adder result to store into the PC.

2.

4.7 Problems in this exercise assume that the logic blocks used to implement a processor's datapath have the following latencies:

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250ps	150ps	25ps	200ps	150ps	5ps	30ps	20ps	50ps	50ps

"Register read" is the time needed after the rising clock edge for the new register value to appear on the output. This value applies to the PC only. "Register setup" is the amount of time a register's data input must be stable before the rising edge of the clock. This value applies to both the PC and Register File.

4.7.1 [5] <§4.4> What is the latency of an R-type instruction (i.e., how long must the clock period be to ensure that this instruction works correctly)?

4.7.2 [10] <§4.4> What is the latency of `lw`? (Check your answer carefully. Many students place extra muxes on the critical path.)

4.7.3 [10] <§4.4> What is the latency of `sw`? (Check your answer carefully. Many students place extra muxes on the critical path.)

4.7.4 [5] <§4.4> What is the latency of `beq`?

4.7.5 [5] <§4.4> What is the latency of an arithmetic, logical, or shift I-type (non-load) instruction?

4.7.6 [5] <§4.4> What is the minimum clock period for this CPU?

3.

4.27 Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add  $s3, $s1, $s0
lw   $s2, 4($s3)
lw   $s1, 0($s4)
or   $s2, $s3, $s2
sw   $s2, 0($s3)
```

4.27.1 [5] <§4.8> If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.

4.27.2 [10] <§4.8> Now, change and/or rearrange the code to minimize the number of NOPs needed. You can assume register `$t0` can be used to hold temporary values in your modified code.

4.27.3 [10] <§4> If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?

4.27.4 [20] <§4.8> If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in Figure 4.59.

4.27.5 [10] <§4.8> If there is no forwarding, what new input and output signals do we need for the hazard detection unit in Figure 4.59? Using this instruction sequence as an example, explain why each signal is needed.

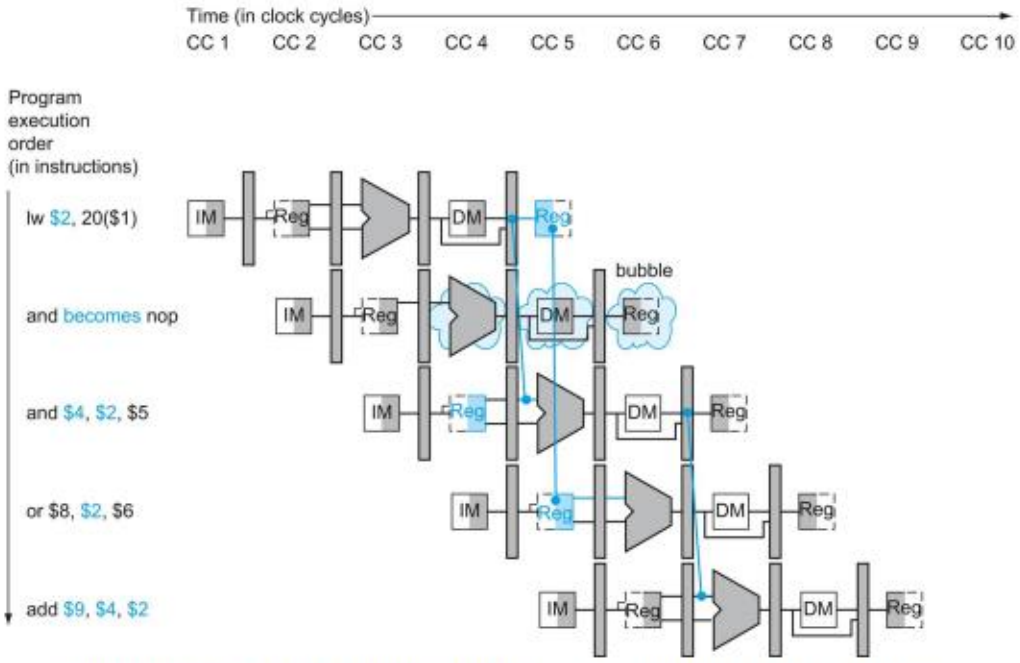

FIGURE 4.59 The way stalls are really inserted into the pipeline.

A bubble is inserted beginning in clock cycle 4, by changing the `and` instruction to a nop. Note that the `and` instruction is really fetched and decoded in clock cycles 2 and 3, but its EX stage is delayed until clock cycle 5 (versus the unstalled position in clock cycle 4). Likewise the `OR` instruction is fetched in clock cycle 3, but its ID stage is delayed until clock cycle 5 (versus the unstalled clock cycle 4 position). After insertion of the bubble, all the dependences go forward in time and no further hazards occur.

4.

4.25 Consider the following loop.

```
LOOP: ld   $s0, 0($s3)
      ld   $s1, 8($s3)
      add  $s2, $s0, $s1
      addi $s3, $s3, -16
      bnez $s2, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the EX (as opposed to the ID) stage.

4.25.1 [10] <§4.8> Show a pipeline execution diagram for the first two iterations of this loop.

4.25.2 [10] <§4.8> Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the `addi` is in the IF stage. End with the cycle during which the `bnez` is in the IF stage.)