HW4

Exercise: 4.3, 4.5, 4.7, 4.16, 4.27, 4.28, 4.29

4.3 Consider the following instruction mix:

R-type	I-type (non-lw)	Load	Store	Branch	Jump
24%	28%	25%	10%	11%	2%

4.3.1 [5] <§4.4> What fraction of all instructions use data memory?

4.3.2 [5] <§4.4> What fraction of all instructions use instruction memory?

4.3.3 [5] <§4.4> What fraction of all instructions use the sign extend?

4.3.4 [5] <§4.4> What is the sign extend doing during cycles in which its output is not needed?

4.5 In this exercise, we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word: `0x00c6ba23`.

4.5.1 [10] <§4.4> What are the values of the ALU control unit's inputs for this instruction?

4.5.2 [5] <§4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

4.5.3 [10] <§4.4> For each mux, show the values of its inputs and outputs during the execution of this instruction. List values that are register outputs at `Reg [xn]`.

4.5.4 [10] <§4.4> What are the input values for the ALU and the two add units?

4.5.5 [10] <§4.4> What are the values of all inputs for the registers unit?

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?

4.16 In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

Also, assume that instructions executed by the processor are broken down as follows:

ALU/Logic	Jump/Branch	Load	Store
45%	20%	20%	15%

4.16.1 [5] <§4.6> What is the clock cycle time in a pipelined and non-pipelined processor?

4.16.2 [10] <§4.6> What is the total latency of an `lw` instruction in a pipelined and non-pipelined processor?

4.16.3 [10] <§4.6> If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

4.16.4 [10] <§4.6> Assuming there are no stalls or hazards, what is the utilization of the data memory?

4.16.5 [10] <§4.6> Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?

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.

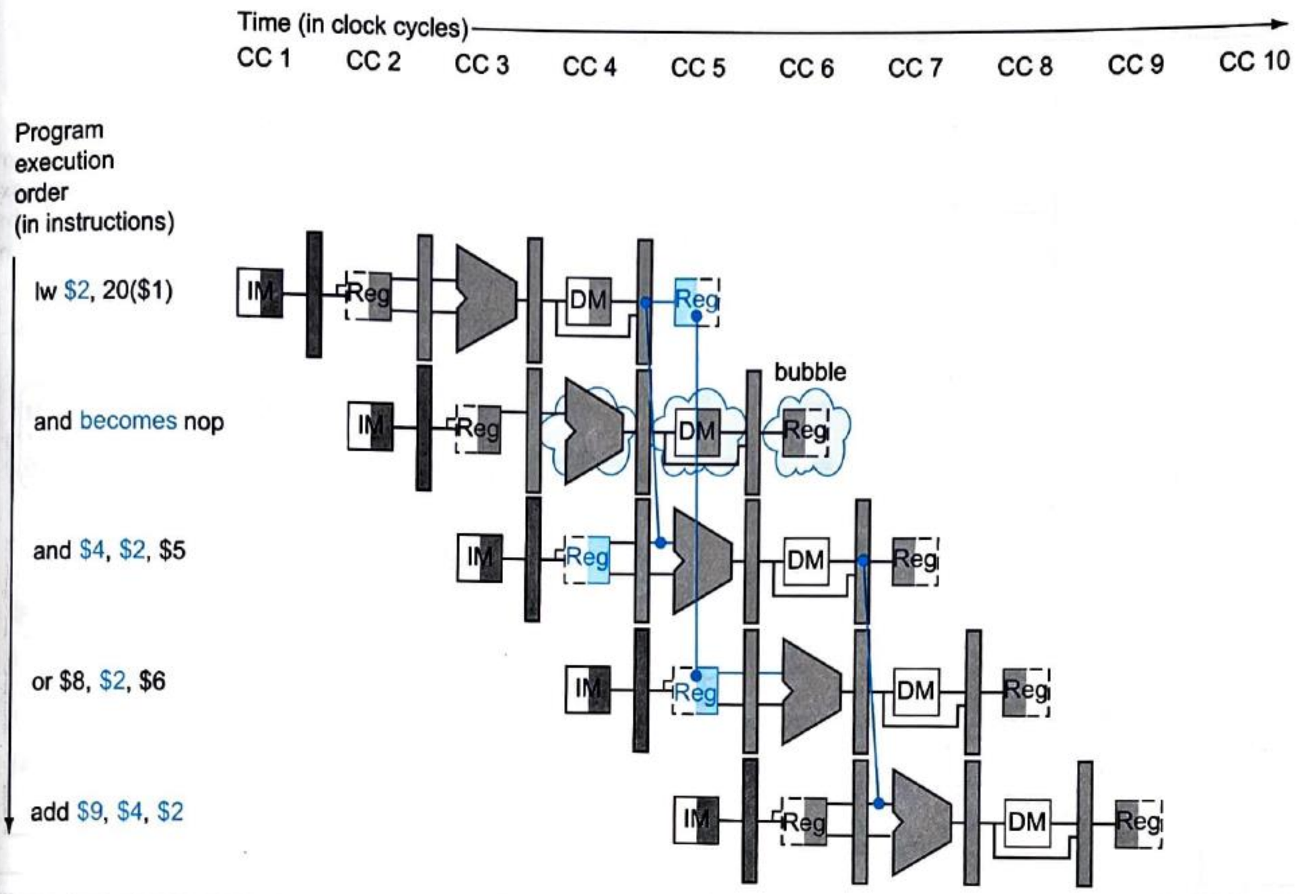

FIGURE 4.59 The way stalls are really inserted into the pipeline. A bubble is inserted beginning in clock cycle 4, by changing the and instruction to a nop. Note that the and instruction is really fetched and decoded in clock cycles 2 and 3, but its EX stage is delayed until lock cycle 5 (versus the 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 urther hazards occur.

4.28 The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

R-type	beqz/bnez	jal	lw	sw
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

4.28.1 [10] <§4.9> Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the ID stage and applied in the EX stage that there are no data hazards, and that no delay slots are used.

4.28.2 [10] <§4.9> Repeat 4.28.1 for the "always-not-taken" predictor.

4.28.3 [10] <§4.9> Repeat 4.28.1 for the 2-bit predictor.

4.28.4 [10] <§4.9> With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions to some ALU instruction? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

4.28.5 [10] <§4.9> With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions in a way that replaced each branch instruction with two ALU instructions? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

4.28.6 [10] <§4.9> Some branch instructions are much more predictable than others. If we knowthat 80% of all executed branch instructions are easy-to-predict loop-back branches that are always predicted correctly, what is the accuracy of the 2-bit predictor on the remaining 20% of the branch instructions?

4.29 This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes: T, NT, T, T, NT.

4.29.1 [5] <§4.9> What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

4.29.2 [5] <§4.9> What is the accuracy of the 2-bit predictor for the first four branches in this pattern, assuming that the predictor starts off in the bottom left state from Figure 4.61 (predict not taken)?

4.29.3 [10] <§4.9> What is the accuracy of the 2-bit predictor if this pattern is repeated forever?

4.29.4 [30] <§4.9> Design a predictor that would achieve a perfect accuracy if this pattern is repeated forever. You predictor should be a sequential circuit with one output that provides a prediction (1 for taken, 0 for not taken) and no inputs other than the clock and the control signal that indicates that the instruction is a conditional branch.

4.29.5 [10] <§4.9> What is the accuracy of your predictor from 4.29.4 if it is given a repeating pattern that is the exact opposite of this one?

4.29.6 [20] <§4.9> Repeat 4.29.4, but now your predictor should be able to eventually (after a warm-up period during which it can make wrong predictions) start perfectly predicting both this pattern and its opposite. Your predictor should have an input that tells it what the real outcome was. Hint: this input lets your predictor determine which of the two repeating patterns it is given.