Ira A. Fulton Schools of Engineering
School of Computing, Informatics and Decision Systems Engineering

SER 450 COMPUTER ARCHITECTURE

Your Name: Jacob Hreshchyshyn Jacob Hreshchyshyn

When completing this worksheet, please use a different color text or typeface so that we can easily identify your work. Remember to show your work / give justification for your answers.

PRACTICE PROBLEMS 4B

Problem 1:

In this problem we examine how pipelining affects the clock cycle time of the processor. For this problem, assume that the 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	beq	lw	sw
45%	20%	20%	15%

A. What is the clock cycle time in a pipelined and non-pipelined processor?

We know that in a non-pipelined processor, each stage of the datapath will be executed sequentially. This means that we would add up the latencies of each stage in the datapath to find the total clock cycle time of a non-pipelined processor:

250ps + 350ps + 150ps + 300ps + 200ps = 1250ps

On the other hand, a pipelined processor will execute each stage of the datapath in parallel simultaneously. This means that the clock cycle time will be equivalent to the execution time of the datapath stage with the largest latency, which is 350ps in this case.

B. What is the total latency of an LW instruction in a pipelined/non-pipelined processor?

From the slides, we know that the LW instruction will require the use of all 5 stages in the datapath. For this reason, in a non-pipelined processor, its latency will be the same as an entire clock cycle, which would be 1250ps.

On the other hand, we know that the cycle latency of an LW instruction is 350ps in a pipelined processor. For this LW instruction to be finished, it will need 5 cycles, 1 for each datapath stage the instruction occupies. Thus, its total latency will be 5*350ps = 1750ps.

C. **If instead of a single-cycle organization, we can use a multi-cycle organization where each instruction takes multiple cycles but one instruction finishes before another is fetched. In this organization, an instruction only goes through stages it actually needs. Compare the clock cycle times and execution times with single-cycle and pipelined organizations (assume no hazards).**

First, it helps to know what datapath stages are occupied during each instruction. We know that LW instructions use all 5 stages. SW instructions use only 4 of those stages since it has no WB need. The ALU instructions use only 4 stages since it does not hit the MEM stage. Finally, BEQ instructions use only 4 stages since it does not hit the WB stage. If we summed up the percentages corresponding to each instruction, we would find that 80% of the instructions use 4 datapath stages while the remaining 20% use all 5. We can use this to find the speedup of the multi-cycle execution organization versus the pipelined organization, which would be 0.2 * 5 datapaths + 0.8 * 4 datapaths = 4.2 times faster than the pipelined organization.

We also know that the cycle times of the single-cycle and pipelined organizations. By dividing one from the other, we can determine that pipelined organizations are 1250ps/350ps = 3.571428571 times faster with respect to clock cycles than the single-cycle organization.

Problem 2:

In this problem, we examine how data dependencies affect execution in the basic 5-stage pipeline described in the text. Refer to the following sequence of instructions:

```
or r1, r2, r3
or r2, r1, r4
or r1, r1, r2
```

Also assume the following cycle times for each of the options related to forwarding (longer cycle times are due to higher combinational logic complexity related to forwarding):

Without forwarding	With full forwarding	With ALU-ALU forwarding only
250ps	300ps	290ps

A. **Assume there is no forwarding in this pipelined processor. Indicate hazards and add nop instructions to eliminate them.**

Nop instructions would be added before instruction 2 since it references r1, which is being written to in instruction 1. Additionally, nop instructions are needed before instruction 3 since it references r2, which is being written to in instruction 2. Each needs two nops in order to align the previous register write with the subsequent register read.

or r1, r2, r3

nop

nop

or r2, r1, r4

nop

nop

or r1, r1, r2

B. **Assume there is full forwarding. Indicate hazards and add nop instructions to eliminate them.**

Assuming full forwarding, there would be no need to add nop instructions since this would remove the previously identified hazards.

C. **What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?**

First, we need to know how many cycles are run when executing these three instructions. Since we are running a pipelined processor, we know that each datapath stage will take a single cycle.

This means that the first instruction will take 5 cycles. Assuming there are no hazards, the next instruction will run a cycle after the previous instruction begins. This applies to the instruction after that. This means that there are 7 cycles to run without hazards being factored in. Without forwarding, an additional 4 cycles of nop are needed. This gives us the following:

No forwarding execution time = (7 + 4) * 250ps = 2750ps

With forwarding execution time = 7 * 300ps = 2100ps

Speedup by full forwarding = 2750ps/2100ps = 1.30952381

Problem 3:

Consider the following loop:

```
loop:     lw   r1, 0(r1)
          and  r1, r1, r2
          lw   r1, 0(r1)
          lw   r1, 0(r1)
          beq  r1, r0, loop
```

Assume perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, and that the pipeline has full forwarding support. Also assume that many iterations of this loop are executed before the loop exits.

A. **Show a pipeline execution diagram for the third iteration of this loop from the cycle in which we fetch the first instruction of that iteration up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. Show all instructions that are in the pipeline during these cycles (not just those from the third iteration)**

B. **How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?**