Session 8: Performance and Microarchitecure

1 Introduction

The goal of this exercise session is to introduce you to some microarchitecure concepts and low level optimization. Learning about these optimizations will not only make you a better programmer, but will also give you more insight in to the wonderful low-level world and enhance your reasoning skills about it. You should read the book from section 4.1 to 4.9 to get a better grasp on processor design and architecture.

2 Ripes

To help you solve and reason about the upcoming exercises, it is advised to install the Ripes RISC-V simulator from `https://github.com/mortbopet/Ripes/releases/latest`. There is support for Windows, Mac and Linux. On Ubuntu, make the .AppImage executable using the command `chmod +x <ripes-filename>` to run the simulator.

Using Ripes, you can simulate four different processors: a single cycle RISC-V and three variants of a 5-stage pipeline. It also has a great cache simulator which can help you better understand what was learned in the caching session.

All programs presented in this session can be executed cycle per cycle using Ripes.

3 Microarchitecture and Performance

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 datapaths *a* and *b* have the following latencies:

	IF	ID	EX	MEM	WB
a	300ps	400ps	350ps	500ps	100ps
b	200ps	150ps	120ps	190ps	140ps

Exercise 3.1. *What is the clock cycle time in a pipelined and single-cycle non-pipelined processor?*

Exercise 3.2. *What is the total latency of a* `lw` *instruction in a pipelined and single-cycle non-pipelined processor?*

Exercise 3.3. *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 Microarchitecture and Performance 2

Consider a processor where the individual instruction fetch, decode, execute, memory, and writeback stages in the datapath have the following latencies:

IF	ID	EX	MEM	WB
150ps	50ps	200ps	100ps	100ps

We assume a classic RISC instruction set where all 5 stages are needed for loads (**lw**), but the writeback **WB** stage is not necessarily needed for stores (**sw**) and the memory **MEM** stage is not necessarily needed for register (R-format) instructions.

Instead of a single-cycle organization, we can use a multi-cycle organization where each instruction takes multiple clock cycles but one instruction finishes before another is fetched. In this organization, an instruction only goes through stages it actually needs (e.g. `sw` only takes four clock cycles because it does not need the WB stage). Compare clock cycle times and execution times with single-cycle, multi-cycle, and pipelined organization.

Exercise 4.1. *Calculate the clock cycle time for a single-cycle, multi-cycle, and pipelined implementation of this datapath.*

Exercise 4.2. *What are the best and worst case total latencies for an instruction in each of the 3 designs (single-cycle, multi-cycle, pipelined)? Note: express instruction latency both in number of cycles as well as in total time (ps). Based on these numbers, which design would you prefer? Explain.*

Exercise 4.3. *Consider the following RISC-V program:*

```
1 addi t0, zero, 10
2 sw zero, 4(t1)
3 lw t2, 10(t3)
```

For each of the 3 CPU designs (single-cycle, multi-cycle, pipelined), we provide a grid below where the horizontal axis represents time (every cell is 100 ps), and the vertical axis lists the instruction stream. First draw the clock signal indicating at which time intervals a new CPU cycle starts, and then visualize how the processor executes the instruction stream over time. Clearly indicate the start and end of each of the 5 datapath stages (IF, ID, EX, MEM, WB) for all instructions. Note: you can assume the CPU starts from a clean state (e.g., after a system reset).

single-cycle design (each cell is 100ps)

clock																									
addi																									
sw																									
lw																									

multi-cycle design (each cell is 100ps)

clock																									
addi																									
sw																									
lw																									

pipeline design (each cell is 100ps)

clock																									
addi																									
sw																									
lw																									

Exercise 4.4. *What is the total time and cycles needed to execute the above RISC-V program from question (4.3) for each of the three CPU designs? What if we add 50 no-operation instructions (`add zero, zero, zero`)? Provide a formula to explain your answer.*

Exercise 4.5. *Describe the performance implications for each of the three CPU designs, if in the above RISC-V program the first instruction is changed to `addi t1, zero, 10`. (hint: hazards)*

5 Forwarding

The code below describes a simple function in RISC-V assembly.

```
or t1,t2,t3
or t2,t1,t4
or t1,t1,t2
```

Assume the following cycle times for each of the options related to forwarding;

Without forwarding:250ps

With Full Forwarding:300ps

Exercise 5.1. *Indicate the dependencies and their type: Read After Write (RAW), Write After Read (WAR), Write after Write (WAW).*

Exercise 5.2. *Assume there is no forwarding in the pipelined processor. Indicate hazards and add* `nop` *(no operation) instructions to eliminate them.*

Exercise 5.3. *Assume there is full forwarding in the pipelined processor. Indicate the remaining hazards and add* `nop` *(no operation) instructions to eliminate them. Compared the speedup achieved by adding full forwarding to a pipeline with no forwarding.*

6 Code Optimization

The code below describes a simple function in RISC-V assembly(A = B + E; C = B + F;).

```
lw t1, 0(t0)
lw t2, 4(t0)
add t3, t1, t2
sw t3, 12(t0)
lw t4, 8(t0)
add t5, t1, t4
sw t5, 16(t0)
```

Exercise 6.1. *Assume the above program will be executed on a 5-stage pipelined processor with forwarding and hazard detection. How many clock cycles will it take to correctly run this RISC-V code?*

Exercise 6.2. *Reorganize the code to optimize the performance?(Hint: try to remove the stalls)*

7 Branching

The code below describes a simple function in RISC-V assembly.

```
        add x1, x0, x0
bar:
        bne x1, x0, exit
        bge x1, x0, foo
        addi x1, x1, -100
        add x5, x5, x5
        add x6, x1, x1
        sub x1, x1, x2
foo:
        addi x1, x1, 1
        jal  x10, bar
exit:
        xor x20,x21,x22
```

Exercise 7.1. *Fill out the following instruction/time diagram for the following set of instructions until the instruction on line 13 (`xor`) fully executes and commits. Execution starts from line 1.*

PC	Instruction	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15	T16	T17	T18	T19	T20	T21
0x2000	add	F	D	X	M	W																
0x2004	bne		F	D	X	M	W															