# CARLETON UNIVERSITY

**Department of Systems and Computer Engineering**

**SYSC 5704** **Elements of Computer Systems** **Assignment 4**

Due date: Thursday, November 13th, 18:00.

---

**4.3** When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. In the following three problems, assume that we are starting with a datapath from Figure 4.2, where I-Mem, Add, Mux, ALU, Regs, D-Mem, and Control blocks have latencies of 400 ps, 100 ps, 30 ps, 120 ps, 200 ps, 350 ps, and 100 ps, respectively, and costs of 1000, 30, 10, 100, 200, 2000, and 500, respectively.

Consider the addition of a multiplier to the ALU. This addition will add 300 ps to the latency of the ALU and will add a cost of 600 to the ALU. The result will be 5% fewer instructions executed since we will no longer need to emulate the `MUL` instruction.

1. [10] <§4.1> What is the clock cycle time with and without this improvement?
2. [10] <§4.1> What is the speedup achieved by adding this improvement?
3. [10] <§4.1> Compare the cost/performance ratio with and without this improvement.

**4.8** 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 | 250ps | 150ps | 300ps | 200ps |

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

| alu | beq | lw | sw |
|---|---|---|---|
| 45% | 20% | 20% | 15% |

1. [5] <§4.5> What is the clock cycle time in a pipelined and non-pipelined processor?
2. [10] <§4.5> What is the total latency of an LW instruction in a pipelined and non-pipelined processor?
3. [10] <§4.5> 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. [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the data memory?
5. [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the write-register port of the "Registers" unit?
6. [30] <§4.5> 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 (e.g., ST only takes 4 cycles because it does not need the WB stage). Compare clock cycle times and execution times with single-cycle, multi-cycle, and pipelined organization.

**4.9** In this exercise, we examine how data dependencies affect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise 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:

| Without Forwarding | With Full Forwarding | With ALU-ALU Forwarding |
|---|---|---|
| 250ps | 300ps | 290ps |

1. [10] <§4.5> Indicate dependencies and their type.
2. [10] <§4.5> Assume there is *no forwarding* in this pipelined processor. Indicate hazards and add `NOP` instructions to eliminate them.
3. [10] <§4.5> Assume there is *full forwarding*. Indicate hazards and add `NOP` instructions to eliminate them.
4. [10] <§4.5> 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?
5. [10] <§4.5> Add `NOP` instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).
6. [10] <§4.5> What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

**4.18** In this exercise we compare the performance of 1-issue and 2-issue processors, taking into account program transformations that can be made to optimize for 2-issue execution. Problems in this exercise refer to the following loop (written in C):

```
for ( i = 0; i != j; i += 2 ) {
    b[i] = a[i] - a[i+i];
}
```

When writing MIPS code, assume that variables are kept in registers as follows, and that all registers except those indicated as *Free* are used to keep various variables, so they cannot be used for anything else.

| i | j | a | b | c | Free |
|---|---|---|---|---|---|
| R5 | R6 | R1 | R2 | R3 | R10, R11, R12 |

1. [– skip –] <§4.10> Translate this C code into MIPS instructions. Your translation should be direct, without rearranging instructions to achieve better performance. *Here is the code so that everyone starts from a common base.*

```
        ADD     R5,R0,R0
Again:  BEQ     R5,R6,End
        ADD     R10,R5,R1
        LW      R11,0(R10)
        LW      R10,1(R10)
        SUB     R10,R11,R10
        ADD     R11,R5,R2
        SW      R10,0(R11)
        ADDI    R5,R5,2
        BEQ     R0,R0,Again
End:
```

2. [10] <§4.10> If the loop exits after executing only **one** iteration, draw a pipeline diagram for your MIPS code from 4.18.1 executed on a 2-issue processor shown in Figure 4.69. Assume the processor has perfect branch prediction and can fetch any two instructions (not just consecutive instructions) in the same cycle. *Use the pipeline diagram of first two instructions below as your template for this answer:*

| Instructions | | Pipeline | | | | | |
|---|---|---|---|---|---|---|---|
| ADD | R5,R0,R0 | IF | ID | EX | ME | WB | |
| BEQ | R5,R6,End | IF | ID | ** | EX | ME | WB |

3. [10] <§4.10> Rearrange your code from 4.18.1 to achieve better performance on a 2-issue statically scheduled processor from Figure 4.69.
4. [– skip –]

5. [10] <§4.10> What is the speedup of going from a 1-issue processor to a 2-issue processor from Figure 4.69? Use your code from 4.18.1 for both 1-issue and 2-issue, and assume that 1,000,000 iterations of the loop are executed. As in 4.18.2, assume that the processor has perfect branch predictions, and that a 2-issue processor can fetch any two instructions in the same cycle. *(Hint: two iterations of the loop are necessary to calculate the CPI for the two-issue version of the processor.)*
6. [10] <§4.10> Repeat 4.18.5 but this time assume that in the 2-issue processor one of the instructions to be executed in a cycle can be of any kind, and the the other must be a non-memory instruction. *(Hint: use your code from 4.18.3 – watch out for stalls across pairs of fetches!)*