For More Practice

Single Cycle Datapaths with Floating Point

5.4 [5] <§5.4> Suppose we have a floating-point unit that requires 400 ps for a floating-point add and 600 ps for a floating-point multiply, not including the time to get the instruction or read and write any registers, which take the same as for an integer instruction. Use the functional unit times from the example on page 315. In these exercises, we will find the performance ratio between an implementation in which the clock cycle is different for each instruction class and an implementation in which all instructions have the same clock cycle time. Assume the following:

- All loads take the same time and comprise 30% of the instructions.
- All stores take the same time and comprise 15% of the instructions.
- R-format instructions comprise 25% of the mix.
- Branches comprise 10% of the instructions, while jumps comprise 5%.
- FP add and subract take the same time and together total 5% of the instructions.
- FP multiply and divide take the same time and together total 10% of the instructions.

Find the time for the FP operations

5.5 [5] <§5.4> For the datapath and instruction mix in Exercise 5.4, find the time for the processor with a single clock cycle length equal to the longest instruction.

5.6 [10] <§5.4> For the datapath and instruction mix in Exercise 5.4, find the time for the processor with a varying length clock.

Effects of Faults in Control Multiplexors

5.15 [5] <§5.4> Describe the effect that a single stuck-at-0 fault (i.e., regardless of what it should be, the signal is always 0) would have on the multiplexors in the single-cycle datapath in Figure 5.17 on page 307. Which instructions, if any, would still work? Consider each of the following faults separately: RegDst = 0, ALUSrc = 0, MemtoReg = 0, Zero = 0.

5.16 [5] <§5.4> This exercise is similar to Exercise 5.15, but this time consider stuck-at-1 faults (the signal is always 1).

5.17 [5] <§5.5> This exercise is similar to Exercise 5.15, but this time consider the effect that the stuck-at-0 faults would have on the multiplexors in the multiple-cycle datapath in Figure 5.28 on page 323. Consider each of the following faults: RegDst = 0, MemtoReg = 0, IorD = 0, ALUSrcA = 0.

5.18 [5] <§5.5> This exercise is similar to Exercise 5.17, but this time consider stuck-at-1 faults (the signal is always 1).

Adding Instructions to the Datapath

5.19 [15] <§5.4> We wish to add the instruction `addi` (add immediate) to the single-cycle datapath described in this chapter. Add any necessary datapaths and control signals to the single-cycle datapath of Figure 5.17 on page 307 and show the necessary additions to Figure 5.18 on page 308. You can photocopy these figures to make it faster to show the additions.

5.20 [15] <§5.4> This question is similar to Exercise 5.19 except that we wish to add the instruction `jal` (jump and link), which is described in Chapter 2 on page 79. You may find it easier to modify the datapath in Figure 5.25 on page 318.

5.21 [8] <§5.4> This question is similar to Exercise 5.19 except that we wish to add the instruction `bne` (branch if not equal), which is described in Chapter 3.

5.22 [15] <§5.4> This question is similar to Exercise 5.19 except that we wish to add a variant of the `lw` (load word) instruction, which sums two registers to obtain the address of the data to be loaded and uses the R-format.

5.23 [5] <§5.4> Explain why it is not possible to modify the single-cycle implementation to implement the swap instruction described in In More Depth: Logical Instruction Exercise 2.52 without modifying the register file.

Datapath Control Signals

5.24 [5] <§§5.4, 5.5> A friend is proposing that the control signal MemtoReg be eliminated. The multiplexor that has MemtoReg as an input will instead use the control signal MemRead. Will your friend's modification work? Consider both datapaths.

5.25 [10] <§5.4> This exercise is similar to Exercise 5.24 but more general. Determine whether any of the control signals (other than MemtoReg) in the single-cycle implementation can be eliminated and replaced by another existing control signal. Why or why not?

Modifying the Datapath and Control

5.26 [15] <§5.4> Consider the following idea: Let's modify the instruction set architecture and remove the ability to specify an offset for memory access instructions.

Specifically, all load-store instructions with nonzero offsets would become pseudo-instructions and would be implemented using two instructions. For example:

```
addi  $at, $t1, 104 # add the offset to a temporary
lw    $t0, $at      # new way of doing lw $t0, 104 ($t1)
```

What changes would you make to the single-cycle datapath and control if this simplified architecture were to be used?

Adding Instructions to the Datapath

5.40 [15] <§5.5> We wish to add the instruction `lui` (load upper immediate) to the multicycle datapath described in this chapter. This instruction is described in Chapter 2 on page 95. Add any necessary datapaths and control signals to the multicycle datapath of Figure 5.28 on page 323 and show the necessary modifications to the finite state machine of Figure 5.37 on page 338. You may find it helpful to examine the execution steps shown on pages 325 through 329 and consider the steps that will need to be performed to execute the new instruction. You can photocopy existing figures to make it easier to show your modifications. Try to find a solution that minimizes the number of clock cycles required for the new instruction. Please explicitly state how many cycles it takes to execute the new instruction on your modified datapath and finite state machine.

5.41 [15] <§5.5> This question is similar to Exercise 5.40 except that we wish to add the instruction `jal` (jump and link), which is described in Chapter 2.

5.42 [15] <§5.5> This question is similar to Exercise 5.40 except that we wish to add the `swap` instruction described in In More Depth: Logical Instruction Exercise 2.52. Do not modify the register file. Since the instruction format for `swap` has not yet been defined, you are free to define it however you wish.

5.43 [15] <§5.5> This question is similar to Exercise 5.40 except that we wish to add a new instruction, `wai` (where am I), which puts the instruction's location (the value of the PC when the instruction was fetched) into a register specified by the rt field of the machine language instruction. Assume that the datapath hasn't changed and that, as usual, the clock cycle is too short to allow an ALU operation and a register file access in a single clock cycle if one of them is dependent on the results of the other.

5.44 [15] <§5.5> This question is similar to Exercise 5.40 except that we wish to add a new instruction, `jm` (jump memory). Its instruction format is similar to that of load word except that the rt field is not used because the data loaded from memory is put in the PC instead of the target register.

5.45 [20] <§5.5> This question is similar to Exercise 5.40 except that we wish to add support for four-operand arithmetic instructions such as `add3`, which adds three numbers together instead of two:

```
add3 $t5, $t6, $t7, $t8   # $t5 = $t6 + $t7 + $t8
```

Assume that the instruction set is modified by introducing a new instruction format similar to the R-format except that bits [0–4] are used to specify the additional register (we still use rs, rt, and rd) and of course a new opcode is used. Your solution should not rely on adding additional read ports to the register file, nor should a new ALU be used.

5.46 [10] <§5.5> Show how the jump register instruction (described on pages 76 and A-64) can be implemented simply by making changes to the finite state machine of Figure 5.37 on page 338. (It may help you to remember that $0 = `$zero` = 0.)

Comparing Processor Performance

5.47 [15] <§§5.1–5.5> For this problem, use the SPEC2000 Integer benchmark data in Figure 3.26 on page 228. Assume that there are three machines:

a. M1: The multicycle datapath of Chapter 5 with a 500 MHz clock.

b. M2: A machine like the multicycle datapath of Chapter 5, except that register updates are done in the same clock cycle as a memory read or ALU operation. Thus, in Figure 5.37 on page 338, states 6 and 7 and states 3 and 4 are combined. This machine has a 400 MHz clock, since the register update increases the length of the critical path.

c. M3: A machine like M2, except that effective address calculations are done in the same clock cycle as a memory access. Thus, states 2, 3, and 4 can be combined, as can 2 and 5, as well as 6 and 7. This machine has a 250 MHz clock because of the long cycle created by combining address calculation and memory access.

Find out which machine is fastest. Are there instruction mixes that would make another machine faster, and if so, what are they?

Implementing Instructions in MIPS

5.48 [20] <§5.5> Suppose there were a MIPS instruction, called `bcp`, that copied a block of words from one address to another. Assume that this instruction requires that the starting address of the source block is in register `$t1` and the destination address is in `$t2`, and that the number of words to copy is in `$t3` (which is ≥ 0). Furthermore, assume that the values of these registers as well as register

`$t4` can be destroyed in executing this instruction (so that the registers can be used as temporaries to execute the instruction).

Write the MIPS assembly language program to implement block copy. How many instructions will be executed to perform a 100-word block copy? Using the CPI of the instructions in the multicycle implementation, how many cycles are needed for the 100-word block copy?

Adding Instructions to the Datapath

5.52 [15] <§5.6> We wish to add the instruction `rfe` (return from exception) to the multicycle datapath described in this chapter. A primary task of the `rfe` instruction is to copy the contents of the EPC to the PC (the exception mechanisms require several additional capabilities that we discuss in Chapter 7). Add any necessary datapaths and control signals to the multicycle datapath of Figure 5.39 on page 344 and show the necessary modifications to the finite state machine of Figure 5.40 on page 345. You can photocopy the figures to make it easier to show your modifications.

Microcode

5.54 [30] <§5.7> Using the strategy you developed in Exercise 5.56, modify the MIPS microinstruction format described in Section 5.7, Figure 5.7.4 on page 5.7-11 and provide the complete microprogram for the `bcp` instruction. Describe in detail how you extended the microcode so as to support the creation of more complex control structures (such as a loop) within the microcode. Has support for the `bcp` instruction changed the size of the microcode? Will other instructions besides `bcp` be affected by the change in the microinstruction format?

5.55 [5] <§5.7> Write the microcode sequences for the `addi` instruction. If you need to make any changes to the microinstruction format or field contents, indicate how the new format and fields will set the control outputs.

5.56 [30] <§5.7> Microcode has been used to add more powerful instructions to an instruction set; let's explore the potential benefits of this approach. Devise a strategy for implementing the `bcp` instruction described in Exercise 5.48 using the multicycle datapath and microcode. You will probably need to make some changes to the datapath in order to efficiently implement the `bcp` instruction. Provide a description of your proposed changes and describe how the `bcp` instruction will work. Are there any advantages that can be obtained by adding internal registers to the datapath to help support the `bcp` instruction? Estimate the improvement in performance that you can achieve by implementing the instruction in hardware (as opposed to the software solution you obtained in Exercise 5.48) and explain where the performance increase comes from.