CprE 381 – Computer Organization and Assembly Level Programming

HW-07

[Note from Joe: This assignment is on the design of processor datapath and control and its impact on performance. You may find the first part relatively easy as you have already had to consider adding features to the single-cycle processor in order to complete your Project Part B. To simplify your effort for the required illustrations, I have included in HW-07.zip three of the figures from the text.]

Reading: Patterson & Hennessy, Sections 4.1-4.4

1) Single-Cycle MIPS Enhancements

(a) We wish to add the instruction `jr` (jump register) to the single-cycle datapath described in this chapter. Add any necessary datapaths and control signals to the single-cycle datapath of Figure 4.21 on page 268 and show the necessary additions to Figure 4.22 on page 269.

(b) This question is similar to part a) except that we wish to add the instruction `lui` (load upper immediate), which is described in Section 2.2.

(c) Consider the single-cycle datapath in Figure 4.21 on page 268. A friend is proposing to modify this single-cycle datapath by eliminating the control signal MemtoReg. The multiplexor that has MemtoReg as an input will instead use either the ALUSrc or the MemRead control signal. Will your friend's modification work? Can one of the signals (MemRead and ALUSrc) substitute for the other? Explain.

(d) 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 pseudoinstructions 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?

2) Architecture and Performance

Assume that the logic blocks needed to implement a single-cycle processor's datapath have the following latencies:

	I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-extend	sll 2
P1	400ps	100ps	30ps	120ps	200ps	350ps	20ps	0ps
P2	500ps	150ps	100ps	180ps	220ps	1000ps	90ps	20ps

(a) Which of the two sets of latencies (P1 or P2) seem more reasonable to you? Why?

(b) What is the clock cycle time for P1 and P2 if the only type of instructions we need to support are ALU instructions (`add`, `and`, etc.)?

(c) What is the clock cycle time for P1 and P2 if we only had to support `lw` instructions?

(d) What is the clock cycle time for P1 and P2 if we must support `add`, `beq`, `lw`, and `sw` instructions?

3) Fault Tolerance

When silicon chips are fabricated, defects in materials and manufacturing errors can result in defective circuits. A very common defect is for one wire to affect the signal in another. This is called a cross-talk fault. A special class of cross-talk faults is when a signal is connected to a wire that has a constant logical value (e.g., a power supply wire). In this case we have a stuck-at-0 or a stuck-at-1 fault, and the affected signal always has a logical value of 0 or 1, respectively. The following problems refer to bit 0 of the Write Register input on the register file in Figure 4.24.

(a) P&H(4.6.1) <§4.3,4.4> Let us assume that the processor testing is done by filling the PC, registers, and data and instruction memories with some values (you can choose which values), letting a single instruction execute, then reading the PC, memories, and registers. These values are then examined to determine if a particular fault is present. Can you design a test (values for PC, memories, and registers) that would determine if there is a stuck-at-0 fault on this signal?

(b) P&H(4.6.2) <§4.3,4.4> Repeat part a) for a stuck-at-1 fault. Can you use a single test for both stuck-at-0 and stuck-at-1? If yes, explain how; if no, explain why not.

(c) P&H(4.6.3) <§4.3,4.4> If we know that the processor has a stuck-at-1 fault on this signal, is the processor still usable? To be usable, we must be able to convert any program that executes on a normal MIPS processor into a program that works on this processor. You can assume that there is enough free instruction and data memory to let you make the program longer and store additional data. Hint: the processor is usable if every instruction "broken" by this fault can be replaced with a sequence of "working" instructions that achieve the same effect.