

(10 pts) Exercise 5-9

Suppose we want to add a new instruction to MIPS called `l_inc` (load and increment). This instruction loads a word from memory and increments the index register after performing the load. For instance, the single instruction

```
l_inc $t0, 0($t1)
```

would have the same effect as the following:

```
lw    $t0, 0($t1)
```

```
addi  $t1, $t1, 4
```

Explain why it is not possible to modify the **single-cycle implementation** given in Figure 5.17 to implement this new instruction **without modifying the register file** (other changes would also be necessary, but focus here on the register file).

The remaining exercises all refer to the multi-cycle implementation.

(5 pts) Exercise 5-11: How many cycles do we need?

In once cycle can do: Register read or write, memory access, ALU

a.) Fill in the cycle number for each task below

<u>Cycle #</u>	<u>Task (for load instruction)</u>
	IR <= Memory[PC]
	A <= Reg[IR[25:21]]
	ALUOut <= A + sign-extend(IR[15:0])
	MDR = Memory[ALUOut]
	Reg[IR[20-16]] = MDR
	PC <= PC + 4

b.) What is the total number of cycles needed?

(5 pts) Exercise 5-12: How many cycles do we need?

In once cycle can do: Register read or write, memory access, ALU

a.) Fill in the cycle number for each task below

<u>Cycle #</u>	<u>Task (for store instruction)</u>
	IR <= Memory[PC]
	A <= Reg[IR[25-21]]
	B <= Reg[IR[20-16]]
	ALUOut <= A + sign-extend(IR[15:0])
	Memory[ALUOut] = B
	PC <= PC + 4

b.) What is the total number of cycles needed?

(5 pts) Exercise 5-13: How many cycles do we need?

In once cycle can do: Register read or write, memory access, ALU

a.) Fill in the cycle number for each task below

<u>Cycle #</u>	<u>Task (for branch instruction)</u>
	IR <= Memory[PC]
	PC <= PC + 4
	A <= Reg[IR[25-21]]
	B <= Reg[IR[20-16]]
	ALUOut <= PC + (sign-extend(IR[15-0]) << 2)
	if (A ==B) PC = ALUOut

b.) What is the total number of cycles needed?

(5 pts) Exercise 5-21: Specify control signals needed for a **load** instruction

Step 3:

ALUOut <= A + sign-extend(IR[15:0]);

Step 4:

MDR = Memory[ALUOut]

Step 5:

Reg[IR[20-16]] = MDR

(3 pts) Exercise 5-22: Specify control signals needed for a **R-type** instruction

Step 3:

ALUOut <= A op B

Step 4:

Reg[IR[15:11]] <= ALUOut;

(2 pts) Exercise 5-23: Specify control signals needed for a **branch** instruction

Step 3:

if (A==B) PC <= ALUOut;

(10 pts) Exercise 5-24: Write out steps 3-4 for a **store** instruction and show the control signals needed (as done on previous exercise)

(15 pts) Exercise 5-26

This is like the earlier exercise on stuck-at-0 faults, but now do for the multi-cycle implementation. For each of the following faults, describe what instructions will not work and explain why. Consider these instructions: R-type, lw, sw, beq.

IRWrite=0

PCWrite=0

MemRead=0

MemWrite=0

RegWrite=0