

## SI232 – Homework #5 (Chapter 5)

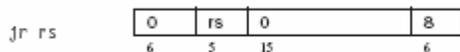
Due: **Friday** March 31, 2006, start of class.

**NOTE:** Last possible time to submit will be Monday April 3 at 0755  
(solutions provided after this point)

Show your work.

- 1) (20 pts) A “stuck-at-0” fault is a defect that can occur during manufacturing, where a particular signal becomes hardwired to zero. Considering the single-cycle implementation shown in Figure 5.17 on page 307, describe the effect that a stuck-at-0 fault would have for each of the following signals. Which instructions, if any, will not work correctly? Explain why. The first is done for you as an example. Consider these instructions: R-type, lw, sw, beq
  - a. RegDst = 0. *lw and sw would not work, because the immediate value from the instruction couldn't be provided to the ALU as needed.*
  - b. MemRead = 0
  - c. MemWrite = 0
  - d. ALUop1 = 0
  - e. ALUop0 = 0
  - f. RegWrite = 0
  
- 2) (20 pts) Consider the jr instruction (jump register), which is described as follows:

**Jump register**

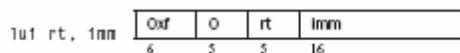


Unconditionally jump to the instruction whose address is in register rs.

We wish to add this instruction to our single-cycle implementation. Add any necessary hardware (gates, adders, wires, etc.) to the single-cycle datapath of page 307 and add an appropriate row to the control chart of page 308 for this new instruction (a copy of these are provided for you on the next page). Note: ‘jr \$s0’ states that the next PC value should come from register \$s0. It is not the same as instructions like ‘j Loop’, whose datapath and control is described on page 315 (though reading about may help with the general idea)

- 3) (20 pts) Do the same as above, but for the ‘lui’ instruction:

**Load upper immediate**



Load the lower halfword of the immediate imm into the upper halfword of register rt. The lower bits of the register are set to 0.

You can find more info on this instruction in Section 2.9.

Again, the following page provides figures for you to use. There are multiple ways to solve this problem; provide some brief text explaining how your solution works. Make sure that the other instructions continue to work.

(more on back)

- 4) (10 pts) 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 $rs, 0($rt)
```

would have the same effect as the following:

```
lw    $rs, 0($rt)
```

```
addi  $rt, $rt, 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, be focus here on the register file).

- 5) (20 pts) This is like the first question on stuck-at-0 faults, but 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.

- a. IRWrite=0
- b. PCWrite=0
- c. MemRead=0
- d. MemWrite=0
- e. RegWrite=0

Problem #2 (adding jump register)

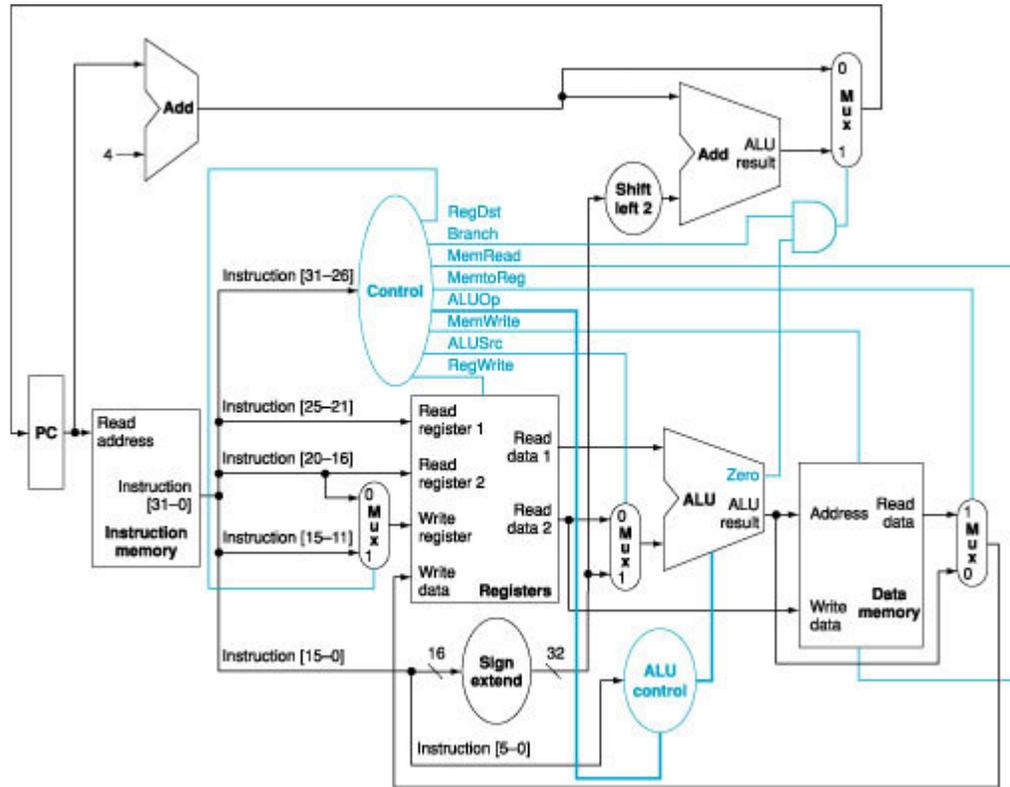


Figure 5.18 – add new row to table for the new instruction

Instr	RegDst	ALUSrc	MemtoReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp2
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Problem #3 (adding load upper immediate)

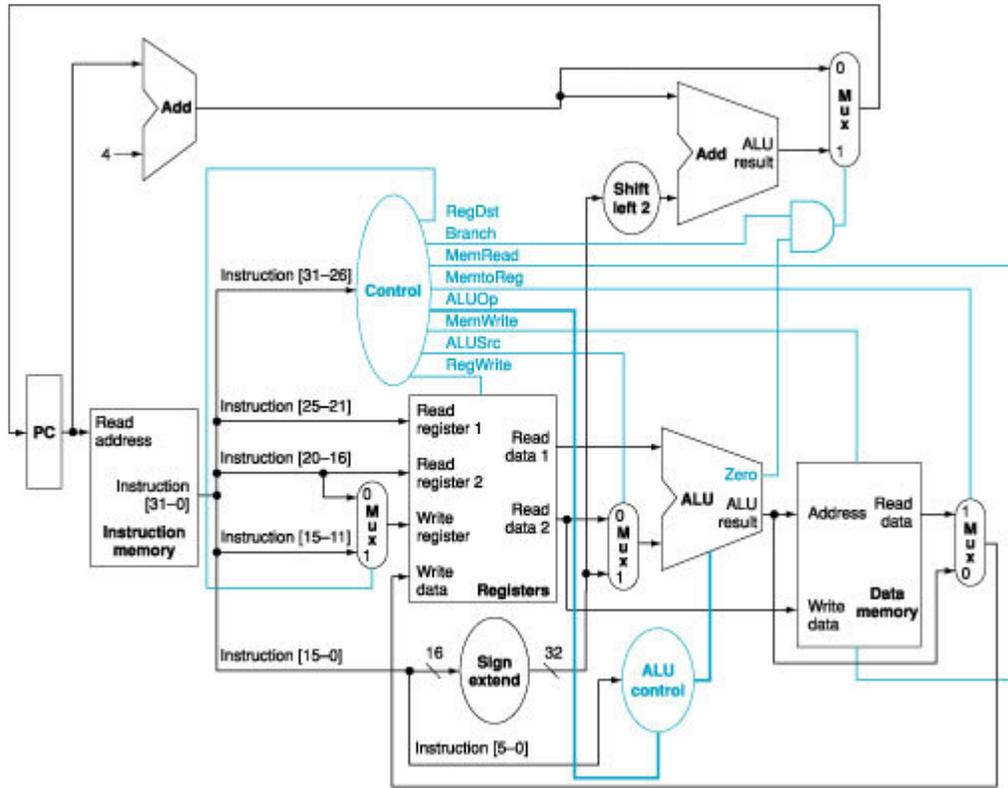


Figure 5.18 – add new row to table for the new instruction

Instr	RegDst	ALUSrc	MemtoReg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp2
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1