Anna Tang
3180300155

Computer Organization: Assignment 4

5.1)

CL = Combinational Logic, SL = Seqeuntial Logic, Both = CL + SL

a. multiplexor - CL

b. comparator - CL

c. incrementer/decrementer - CL

d. barrel shifter - Both

e. multiplier with shifters and adders - Both

f. register - SL

g. memory - SL

h. ALU (the ones in single-cycle and multiple-cycle datapaths) - CL

i. carry look-ahead adder - CL

j. latch - SL

k. general finite state machine (FSM) – Both

5.2)

a) RegWrite = 0

All R-format and lw instructions cannot work if RegWrite was deasserted because the result of the operations would not be written into the destination registers.

b) ALUop0 = 0

This would only affect the beq instruction, because ALUop0 needs to be asserted for when performing a subtraction operation for rs and rt. Deasserting it would cause the ALU to do rs + rt instead.

c) ALUop1 = 0

All R-format instructions (except for subtraction) would be affected. The ALU would do subtraction instead of the intended operation.

d) Branch = 0

This would only affect the beq instruction. The PC would not know to take the branching address when it should have.

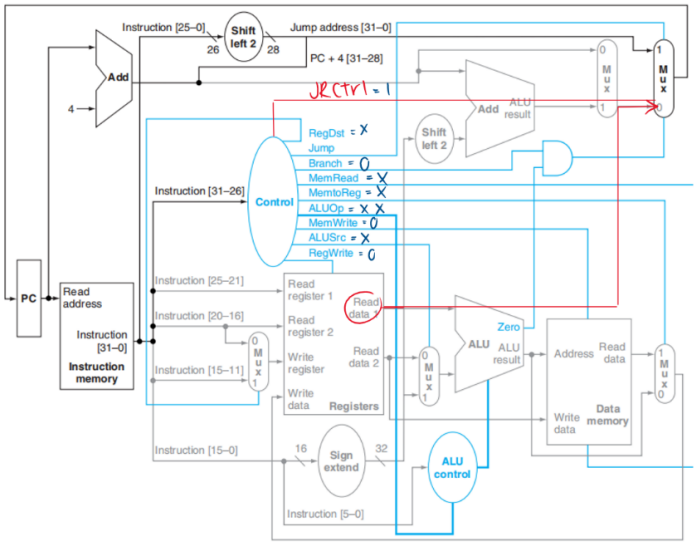
e) MemRead = 0

This would only affect the lw instruction. Deasserting this signal would not allow the data memory to be read from.

f) MemWrite = 0

This would only affect the sw instruction. Deasserting this signal would not allow the data memory to be written to.

5.8) Implementing jr instruction to single-cycle datapath:

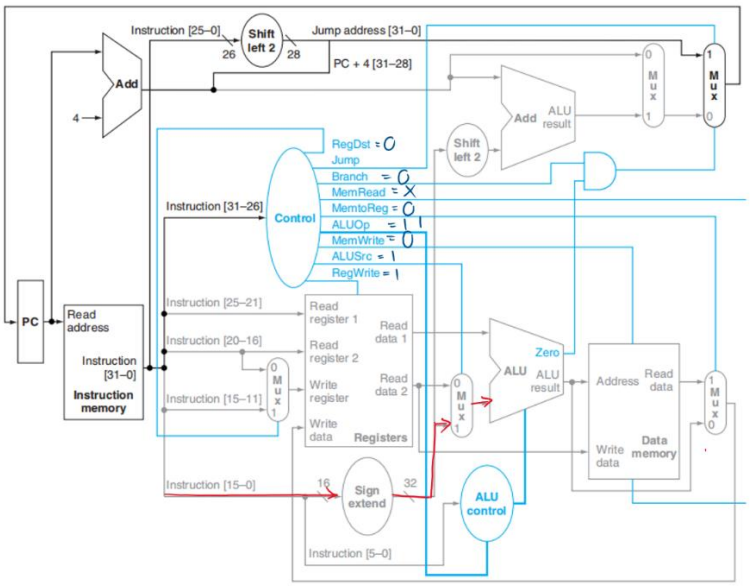


ADDITIONS:

- another ctrl signal in CTRL ("JRCtrl"
  - controls the MUX and tells it to select input from read Data 1
- A wire that connects the output from read Data to the MUX input port

CTRL signals indicated in blue.

5.10) Implementing lui instruction to single-cycle datapath:

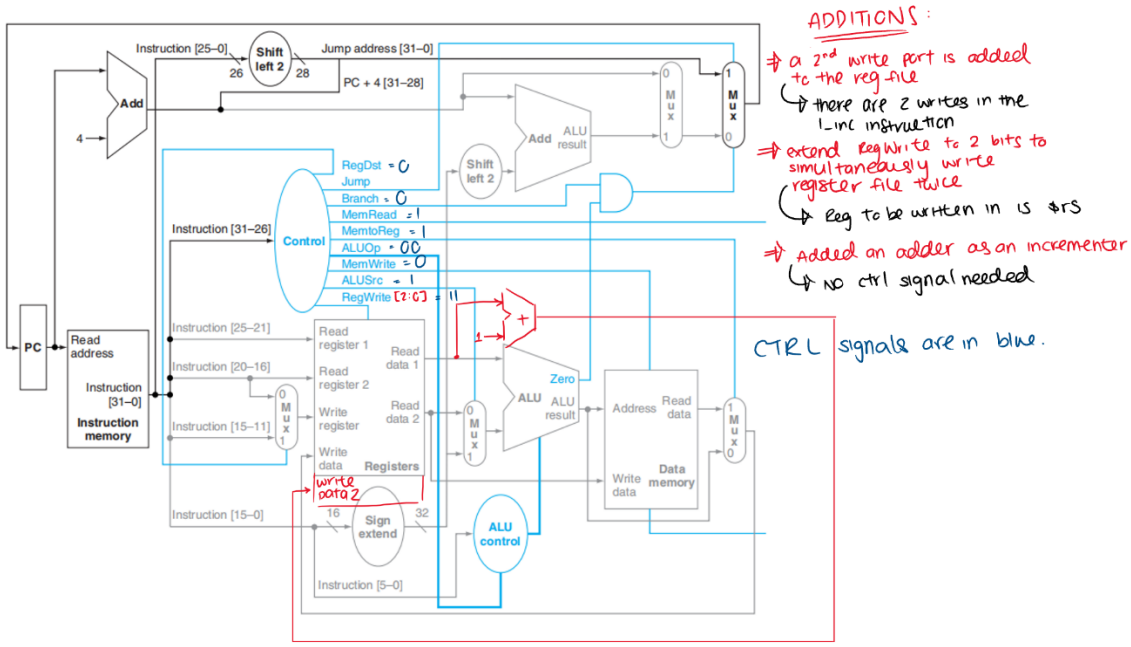


ADDITIONS:

- no additions needed; just feed instruction [15:0] to the ALU input.
- set ALU for a sll 16 operation

controls indicated in blue.

5.11) Implementing l_inc instruction to single-cycle datapath:



5.14)

Case 1: There is an available register that may be destroyed

Let $t0 represent the temporary register used in the swap. We assume that there is nothing in this register and will be overwritten.

```
move $t0, $rs     #Replace $t0 with the contents of $rs

move $rs, $rt     #Replace $rs with the contents of $rt

move $rt, $t0     #Replace $rt with the contents of $t0, which has the
original contents of $rs
```

Case 2: There exists no register

Adapted from https://en.wikipedia.org/wiki/XOR_swap_algorithm.

```
XOR $rs, $rs, $rt       #$rs = $rs ^ $rt, $rt = $rt

XOR $rt, $rs, $rt       #$rs = $rs ^ $rt, $rt = $rs ^ 0 = $rs

XOR $rs, $rs, $rt       #$rs = $rt ^ 0 = $rt, $rt = $rs
```

If hardware implementation needs a 10% increase in the clock cycle period, that implies that the swap instructions takes up a total of 10% of the total instructions. We can also imply that there will be a 1% increase in the clock cycle period. Overall, a hardware implementation is generally preferable for most cases of the swap instruction.

5.29)

a) RegWrite = 0

This would affect the lw instruction and all R-format instructions. Deasserting this control would not let the register file to be written into.

b) MemRead = 0

This would affect the all instructions. Deasserting this signal would not let the datapath retrieve any data from the memory.

c) MemWrite = 0

This would affect the sw instruction. Deasserting this signal would not let the location at the specified memory address be overwritten.

d) IRWrite = 0

All instructions would be affected. This would not let the memory write into the instruction register.

e) PCWrite = 0

This would affect the jump instructions. This would not mark the operation as an unconditional branch, thus disabling the datapath to write the PC with the branching address.

f) PCWriteCond = 0

This would only affect the branching (beq, bne etc.) instructions. This would not let the PC be overwritten with the branching address, if the zero signal from the ALU was asserted.

5.30)

a) RegWrite = 1

The sw instruction will write into the register file instead of the memory address. Branch and jump instructions would end up writing their branch/target addresses into the register file and not into the PC.

b) MemRead = 1

This does not affect any instruction in the datapath. Nothing is done to the memory, other than it being constantly read.

c) MemWrite = 1

This would affect all instructions, because the memory contents at the specified location would be replaced by the contents in a register.

d) IRWrite = 1

The lw instruction would be affected, because the output from the memory data register sent to the instruction register and get translated as an instruction.

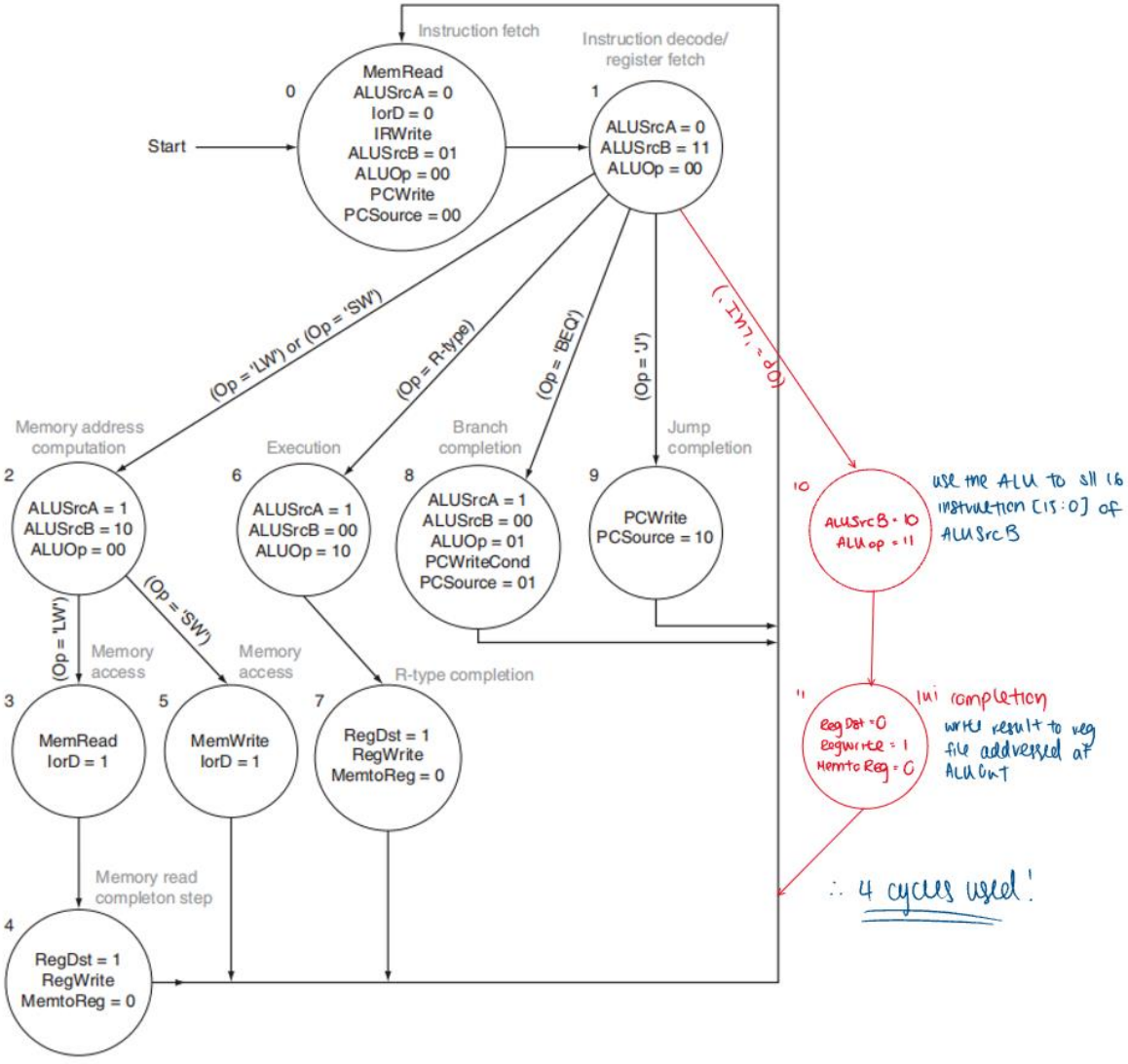
e) PCWrite = 1

Every instruction except for jump instructions would not work correctly, because the ALU would misinterpret the signal and the PC would get written with a non-existing/non-intensional jump address, and attempt to take that as a branch.

f) PCWriteCond = 1

Every instruction except for the branch instructions would not work correctly, because this signal is only supposed to be asserted when the zero port of the ALU is asserted as well. The PC would be written with the output of the ALUout port, which might not necessarily be a valid address and can potentially be misinterpreted.

5.32) Implement lui into multicycle datapath:



5.34) Implement ldi into multicycle datapath: