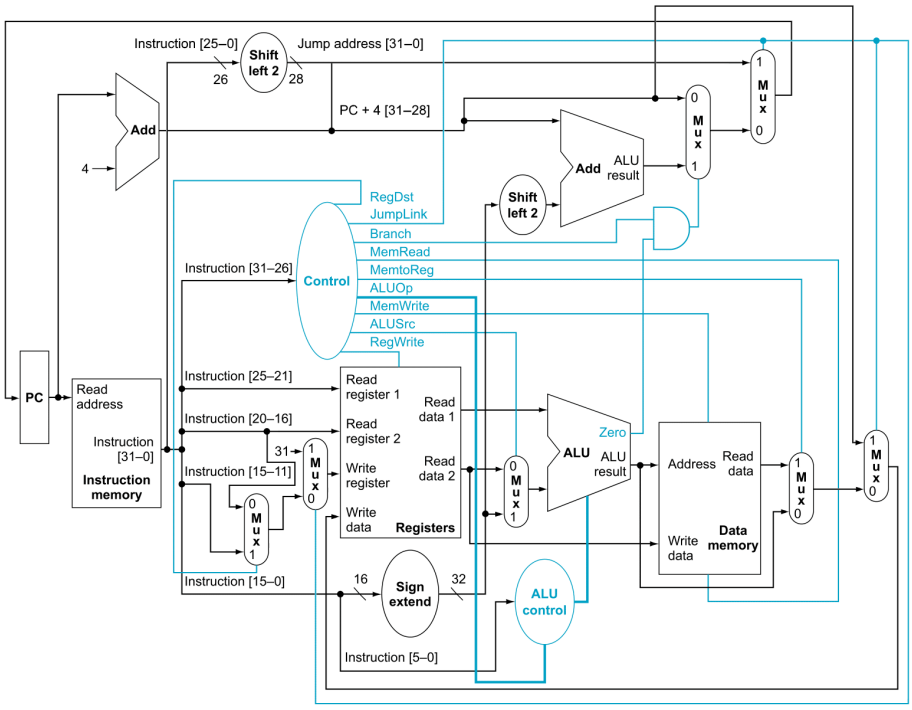
# Computer Science M151B, Homework 3

Michael Wu
UID: 404751542

April 23rd, 2018

## Problem 1




**a)** I have added three multiplexers, one to update the program counter to the target value, one to input the next instruction as a write value, and one

to select register 31, which corresponds to `$ra`. On the upper left the target value is shifted left by 2 bits and the upper 4 bits of the program counter are concatenated on the left of the resulting value, giving a single 32 bit address for the program counter to jump to. A single new control signal `JumpLink`, controls these multiplexers.

**b)** `JumpLink` is the new control signal required. This should be 1 during the `jal` instruction, and controls multiplexers that cause the program counter to jump to the target address and saves the next instruction in register `$ra`.

**c)**

| Input or Output | Signal | R-Format | lw | sw | beq | jal |
|---|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 | 0 |
| | Op4 | 0 | 0 | 0 | 0 | 0 |
| | Op3 | 0 | 0 | 1 | 0 | 0 |
| | Op2 | 0 | 0 | 0 | 1 | 0 |
| | Op1 | 0 | 1 | 1 | 0 | 1 |
| | Op0 | 0 | 1 | 1 | 0 | 1 |
| Outputs | RegDst | 1 | 0 | x | x | x |
| | ALUSrc | 0 | 1 | 1 | 0 | x |
| | MemtoReg | 0 | 1 | x | x | x |
| | RegWrite | 1 | 1 | 0 | 0 | 1 |
| | MemRead | 0 | 1 | 0 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 | 0 |
| | Branch | 0 | 0 | 0 | 1 | 0 |
| | ALUOp1 | 1 | 0 | 0 | 0 | x |
| | ALUOp2 | 0 | 0 | 0 | 1 | x |
| | JumpLink | 0 | 0 | 0 | 0 | 1 |

# Problem 2

```
recf:
  addi $sp, $sp,   -12
  sw   $ra, 0($sp)
  sw   $s0, 4($sp)
  sw   $s1, 8($sp)
```

```
  addi $t0, $zero, 1
  beq  $a0, $zero, baseCase0
  beq  $a0, $t0,   baseCase1
  add  $s0, $a0,   $zero
  srl  $a0, $s0,   2
  addi $a0, $a0,   1
  jal  recf
  add  $s1, $v0,   $zero
  srl  $a0, $s0,   1
  addi $a0, $a0,   -1
  jal  recf
  add  $v0, $s1,   $v0

finish:
  lw   $ra, 0($sp)
  lw   $s0, 4($sp)
  lw   $s1, 8($sp)
  addi $sp, $sp,   12
  jr   $ra

baseCase0:
  addi $v0, $zero, 3
  ja   finish

baseCase1:
  addi $v0, $zero, 2
  ja   finish
```

# Problem 3

**a)** Yes.

**b)** In a single cycle implementation, the number of cycles that a program takes to execute will be fixed. This is because each instruction takes one cycle, and the program will run a fixed number of instructions. So the only way that a program can change speed is if the time for a cycle changes. The time of a cycle is determined by the longest execution path in our implementation.

So increasing the length of time of an `or` operation could increase the cycle time, if the `or` operation is on the longest execution path.

**c)** No.

**d)** Within the ALU is a ripple carry adder, which typically takes much more time to finish execution than the `or` gate. Electrical signals must propagate all the way through the adder, so this execution path should be much longer than the execution path for the `or` operation. Thus the `or` operation is most likely not on the longest execution path, and doubling its time would not increase the cycle time. Thus it is most likely that the execution time of the program would not change.

# Problem 4

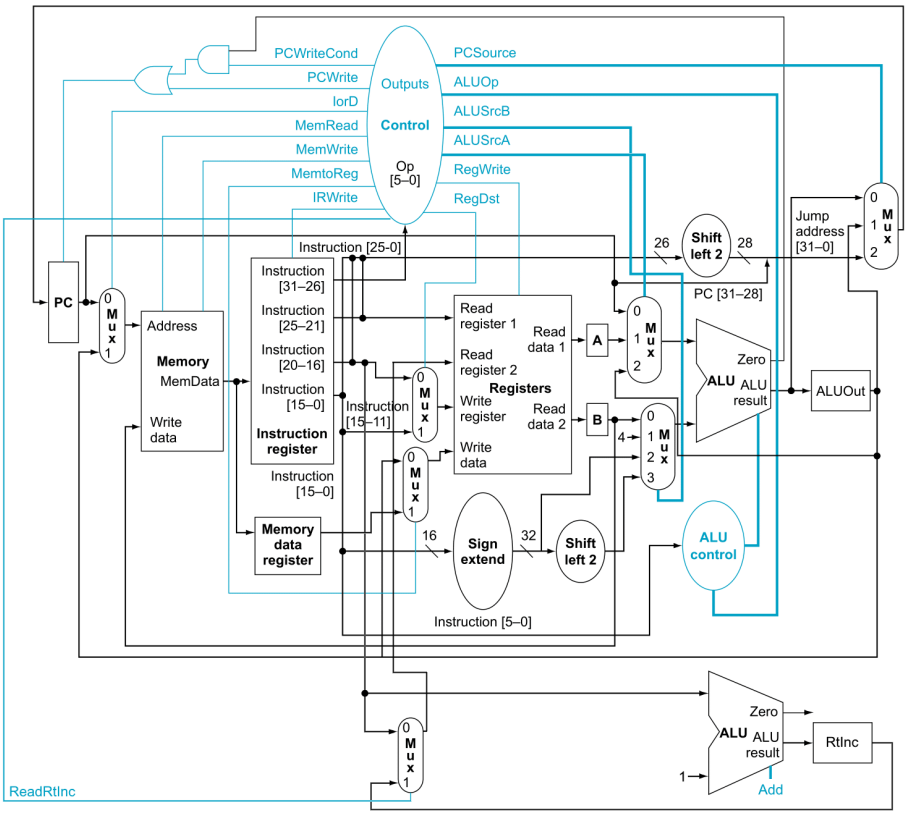
In the multicycle implementation, asserting `ALUSrcA` incorrectly has no visible effect when `RegWrite` is asserted, as the result of the ALU is not used in the next cycle. Thus the only situation where this hardware fault would have a visible effect is during states 2, 6, and 8 in the finite state machine. This is because `ALUSrcA` is asserted while `RegWrite` should be unasserted.

During state 8, the hardware fault causes `beq` to store the branch target address into the register `rt`. In state 6, the lower half of the instruction is used to calculate a branch target address, which is stored in `rt`. So R-format instructions will have the effect of storing some garbage value in `rt`. However, if `rd` is the same as `rt`, this will have no visible effect as the write to `rd` in state 7 will overwrite the write to `rt`. During state 2, `lw` instructions will have no visible change, as state 4 overwrites the garbage value written to `rt`. However, `sw` instructions will set `rt` to some garbage value.

# Problem 5

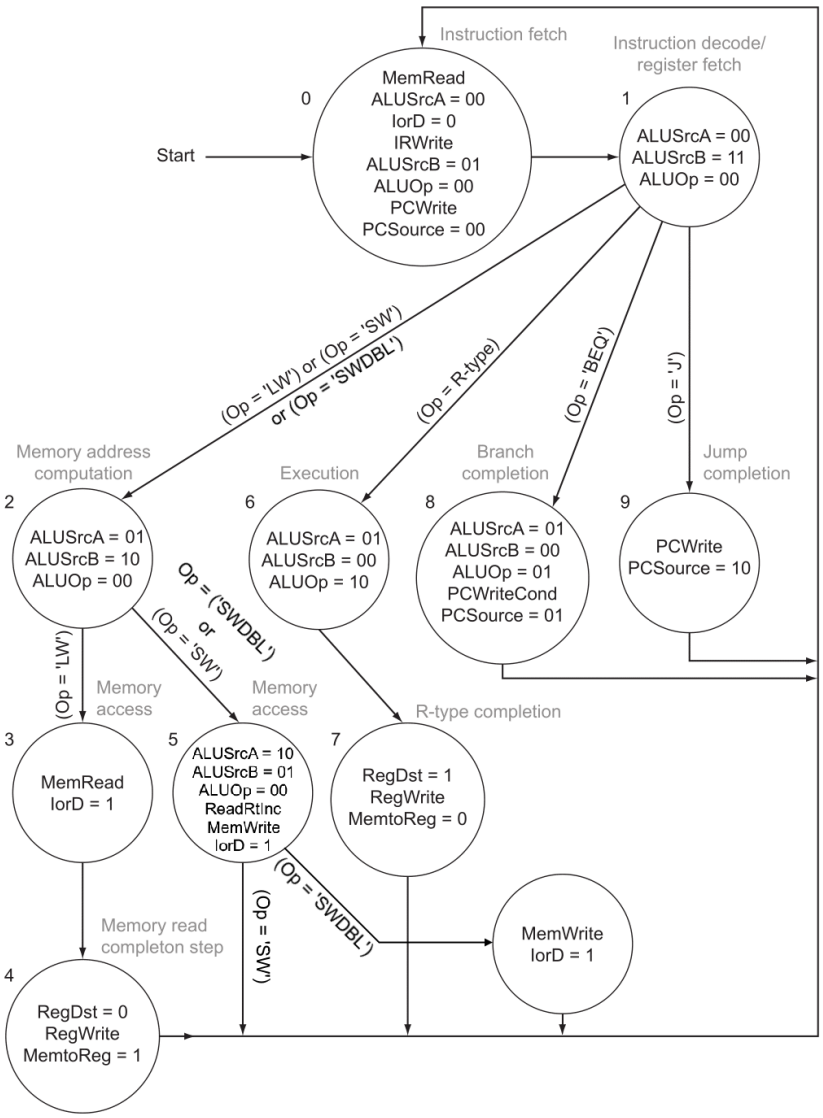
**a)** I will modify the datapath to add another ALU that is only used to increment the register number given by `rt`. This will be fed to a multiplexer to select whether we read the register the `rt` or `rt` + 1. Then I will add an input to the multiplexer controlled by `ALUSrcA` which is `ALUOut`, so that I can increment `ALUOut` by 4. Then `swdbl` requires one additional state after state

5, as the register `rt` + 1 should be read in state 5 and the memory address calculation for the next memory location should happen in one additional state.




**b)** The modified datapath is shown above. There is a dedicated circuit with an ALU and a register that calculates the register number following `rt`, and a multiplexer to choose which to read. Additionally `ALUSrcA` is expanded by one bit in order to add a multiplexer input from `ALUOut` that allows for `ALUOut` to be incremented by 4.

**c)** Yes, a signal into the additional ALU should always send an add signal, and `ReadRtInc` controls whether we should read `rt` or `rt` + 1.




**d)** The modified state diagram is shown above. The control signal `ALUSrcA` has been extended by one bit, and an additional state has been added. Additionally, in state 5 we perform the memory read from the next register and increment the memory address to store into.

**e)** It takes 6 cycles to execute `swdbl`.