## Announcements

The midterm is Friday

Bring a colored pencil or highlighter

1

## Overview of Content

Basics: Representations -- binary, hex, 2's complement, ASCII, strings, floating point single/double precision

Ex: What binary number is the hex BEEF 2ABE?

Machine Language: data types, instruction types, MIPS instruction repertoire, alignment

Ex: Load FAB4 into the first argument register

Ex: Let $s0 contain a negative number; write code to shift it right 3 bits, keeping it negative

Assembly Language: register designations, stack, calling conventions, pseudo instructions

Ex: How is bge $4, $5, Address implemented?

2

## Assembly Language Programming

Write short code segments in MIPS Assembly

Hints:

- Plan out 2 basic structure (if-thens, basic loops)
- Write intended code in C before writing Assem
- Comment profusely

```
                                        int fact(int n) {
fact:
        li    $v0, 1      #set f to 1       int i, f = 1;
        move  $t1,$a0     #set i to param n for (i = n; i > 1; i--)
loop:                                           f = f * i;
        blei  $t1,1,exit  #exit if done     return f;
        mul   $v0,$v0,$t1 #build factorial  }
        subi  $t1, $t1,1  #i--
        j     loop        # iterate
exit:   $ra                #return val in v0
```

3

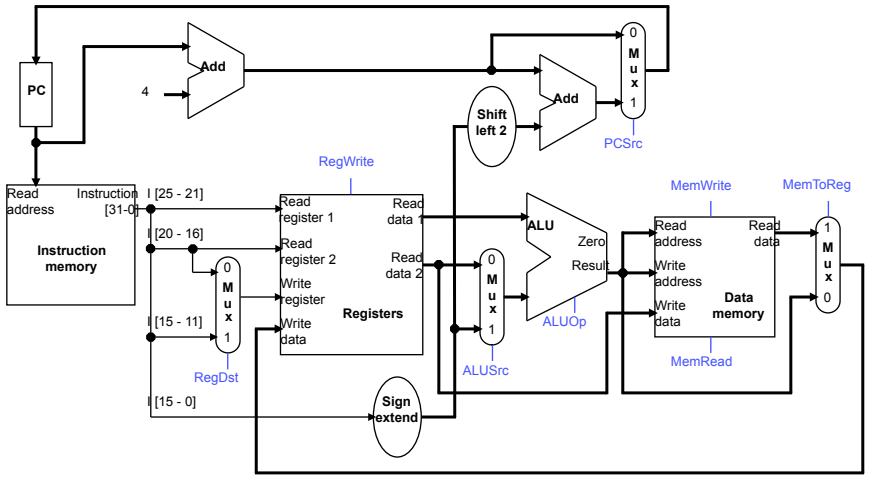
## Design of 1-cycle MIPS

MIPS design:

- Know all components and their operation
- Know flow of logic -- which components are active when implementing a given instruction
- Be able to specify control signals needed to accomplish specific instructions
- Know why the 1-cycle design underperforms

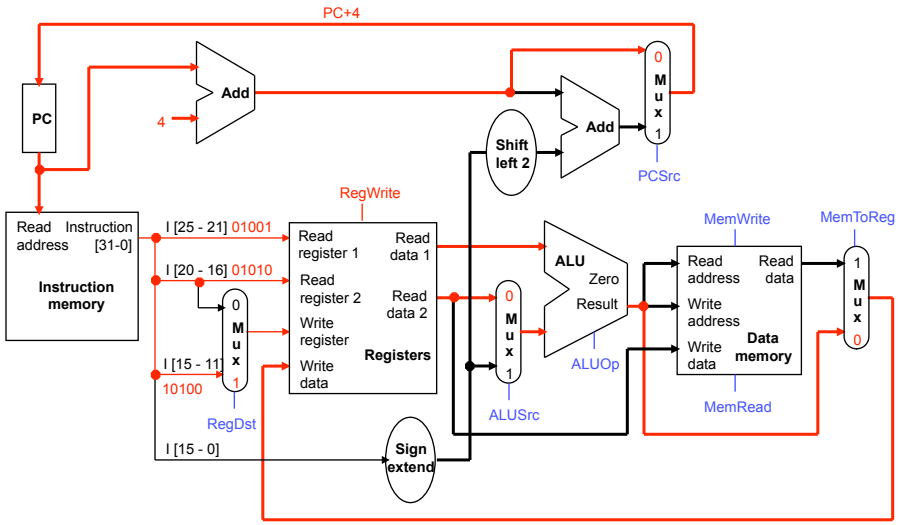
4

# Final 1-cycle design




5

# Instruction Interpretation




6

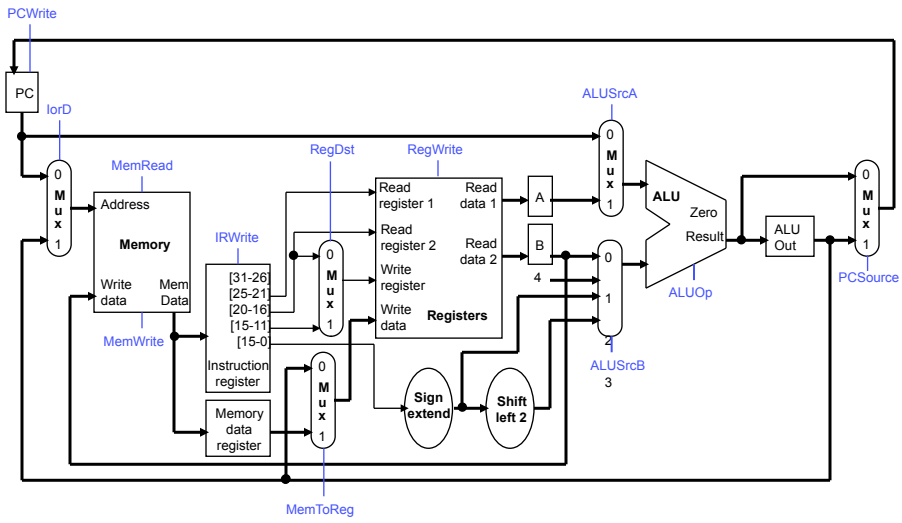
# Multicycle design

Regarding the multicycle design

- How did we segue from 1-cycle design to this?
- Added/changed components from 1-cycle design
- Know multicycle design operation for each instruction, and give control settings to make it happen

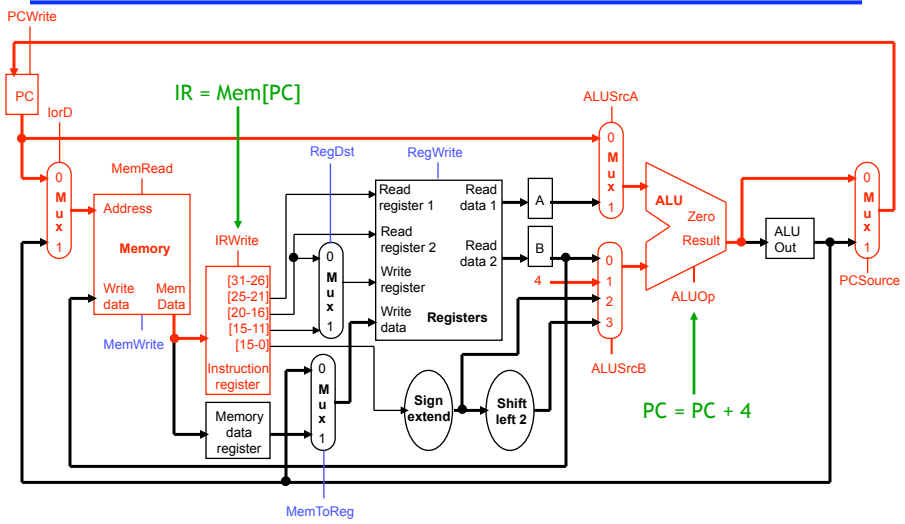
7

# Final Multicycle



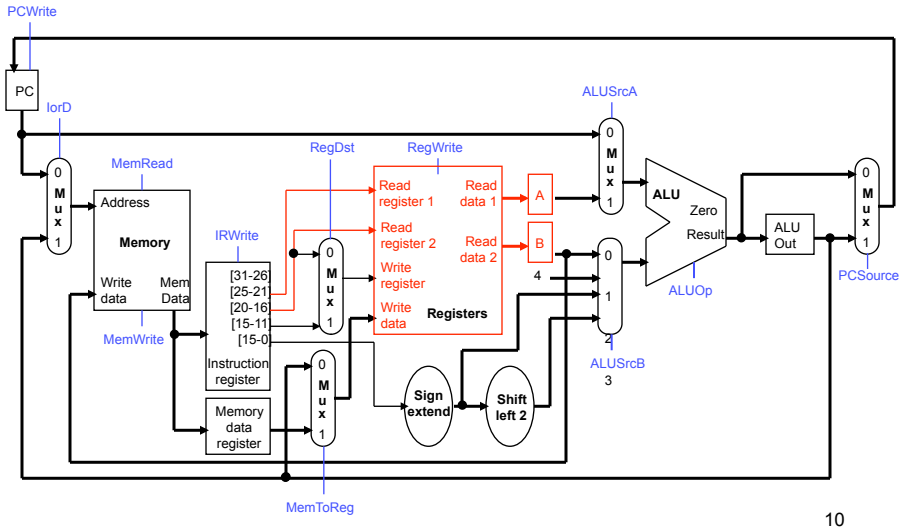

8

# Stage 1: Instruction fetch & PC increment



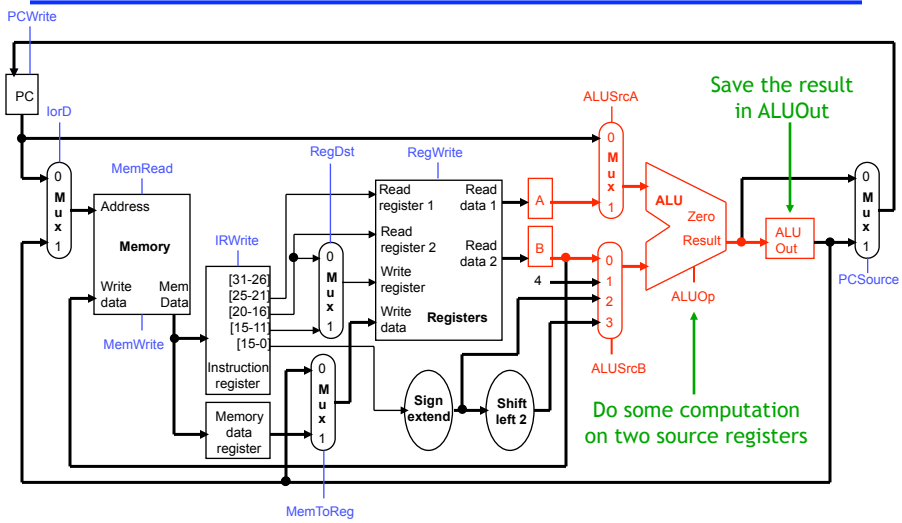

9

# Register File Read



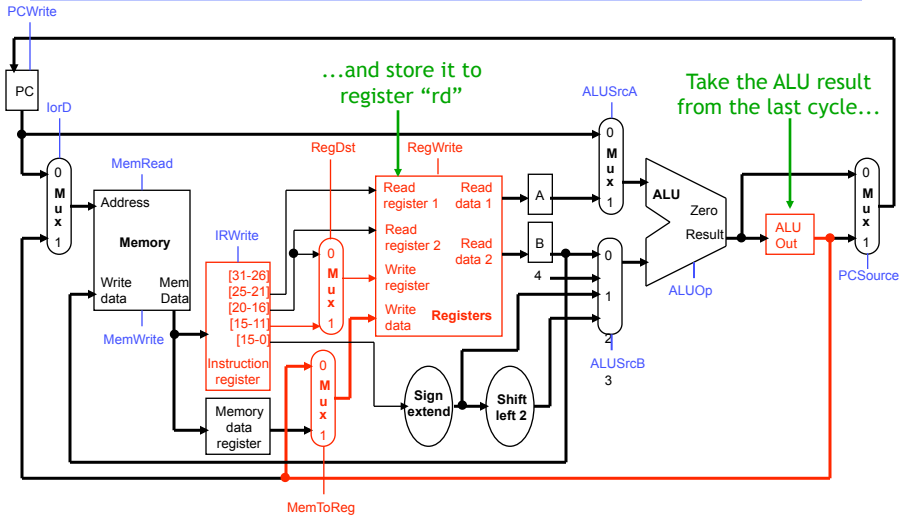

10

# Stage 3 (R-type): instruction execution

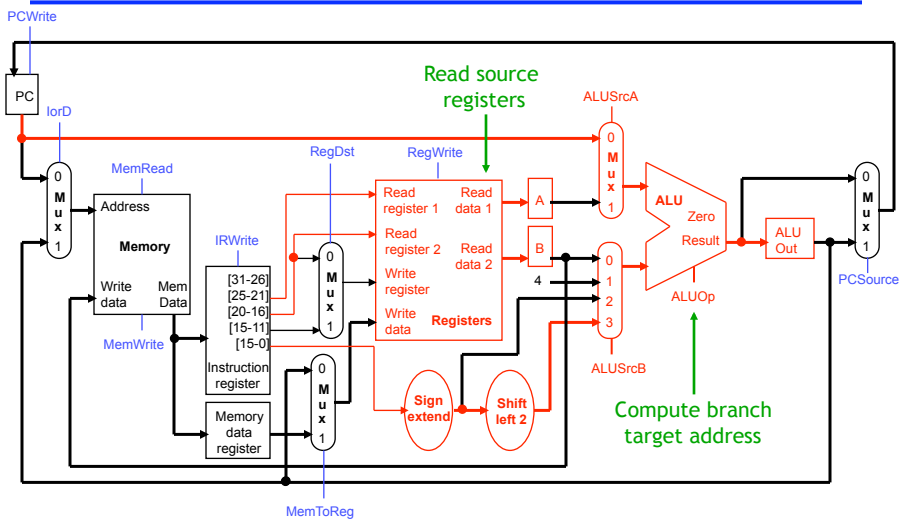





# Stage 4 (R-type): write back

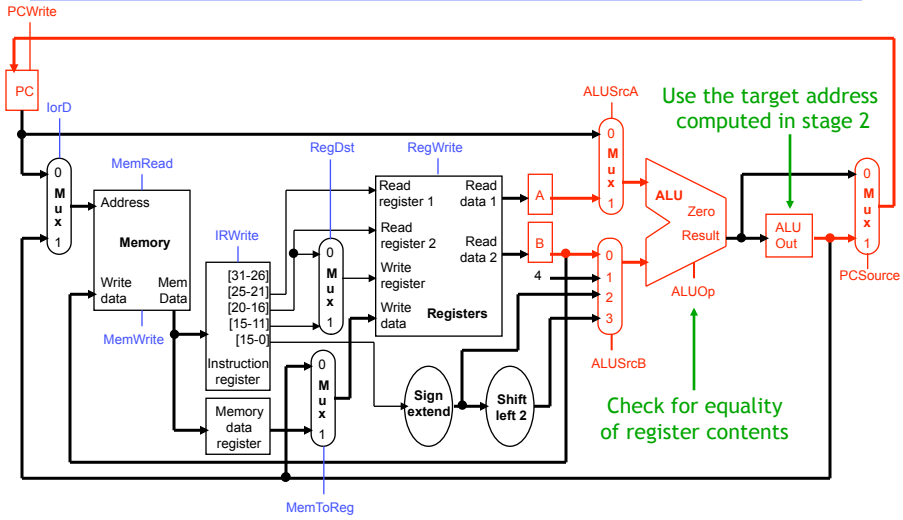


# Stage 2: Reg fetch & branch target



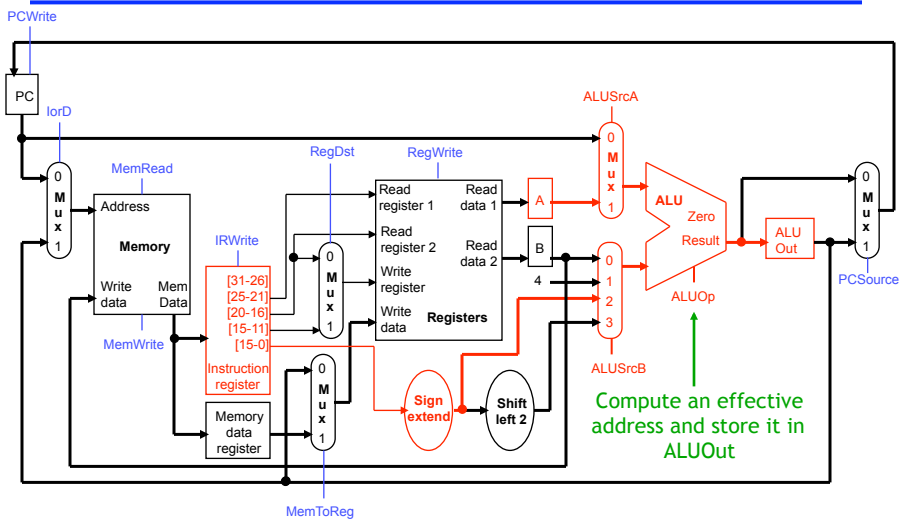

13

# Stage 3 (beq): Branch completion



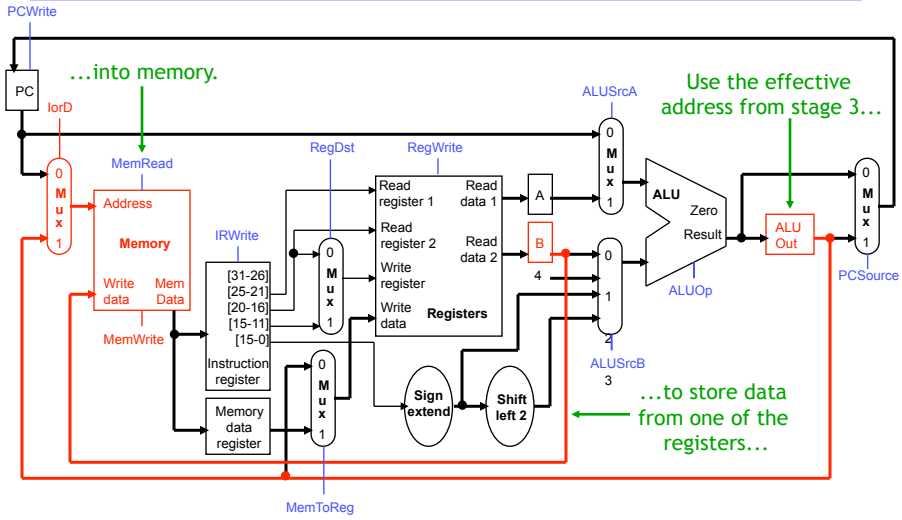

14

# Stage 3 (sw): compute effective address

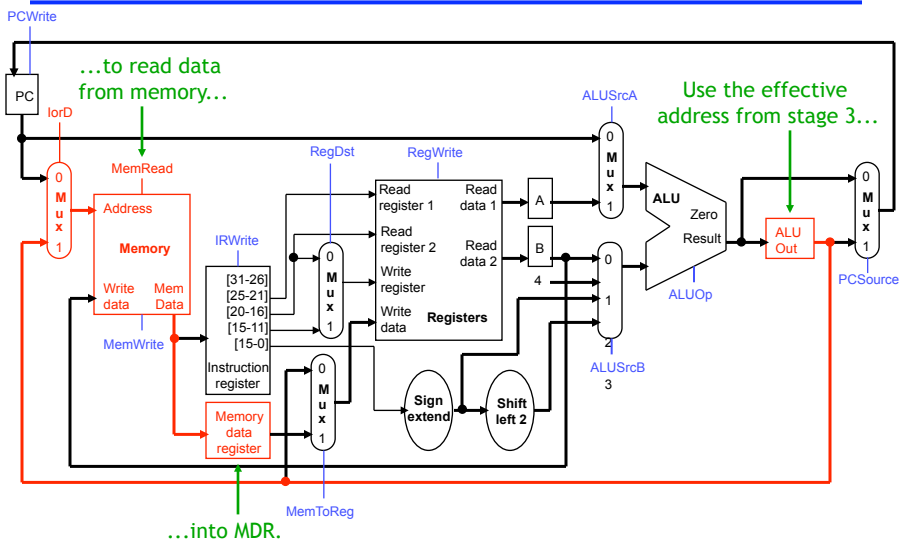





# Stage 4 (sw): memory write

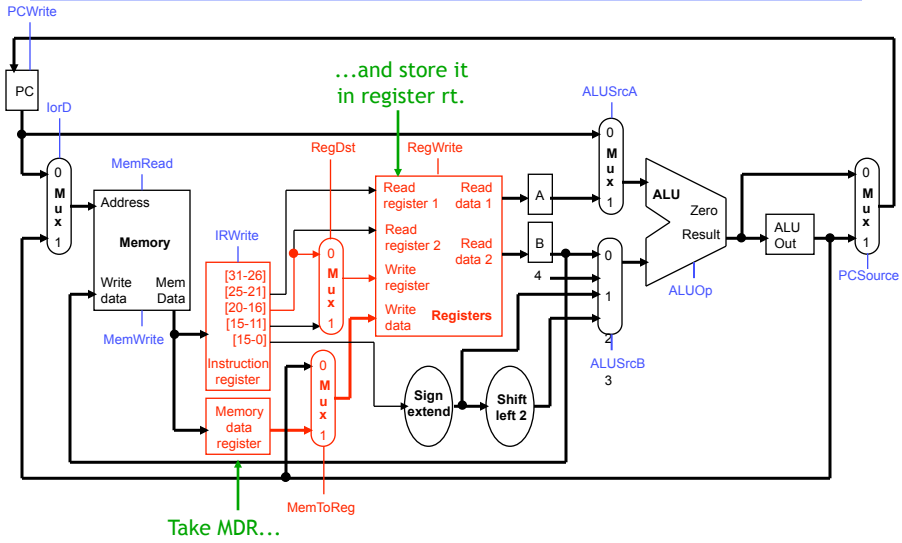


# Stage 4 (lw): memory read




17

# Stage 5 (lw): register write




18