## Lecture 10: Simple Data Path

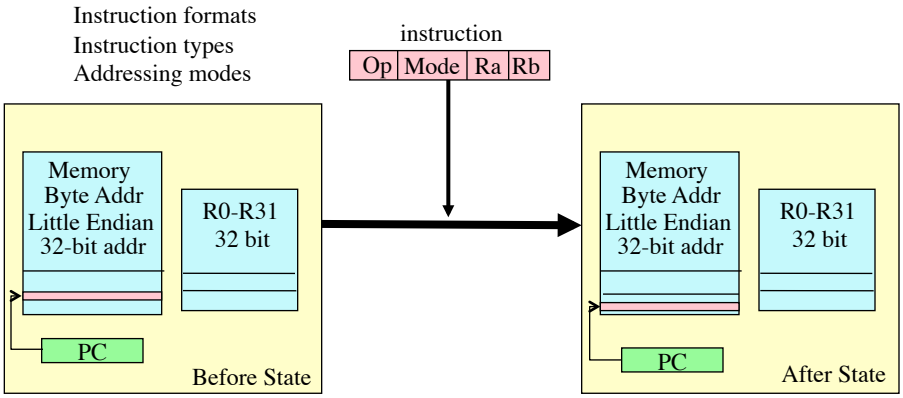
- Course so far
  - Performance comparisons
  - Amdahl's law
  - ISA function & principles
  - What do bits mean?
  - Computer math
- Today
  - Take QUIZ 6 over P&H 4.1-4, before 11:59pm today
  - How do computers execute instructions?
    - From specification to state changes
    - In order simple data path execution



## Exam Return

- Perusal in ACES 3.422 (Gem Naivar's office)
- Questions in office hours
- Grade distribution
  - 90 - 100 2
  - 80 - 89 19
  - 70 - 79 29
  - 60 - 69 16
  - 50 - 59 3 - Go to office hours
  - 40 - 49 1
  - 30 - 39 2
  - 0 - 29 0

# ISA Specifies How the Computer Changes State




Machine state: PC, memory, registers



# Simplify for Understanding

- Let's abstract & simplify: 9 MIPS instructions from the three categories:
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq, j`

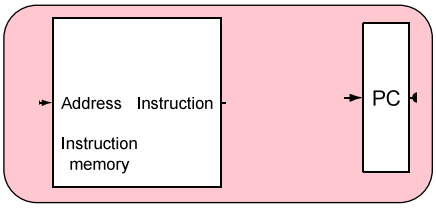
add $8, $17, $18 Instruction Format:

| 000000 | 10001 | 10010 | 01000 | 00000 | 100000 |
|---|---|---|---|---|---|
| op | rs | rt | rd | shamt | funct |

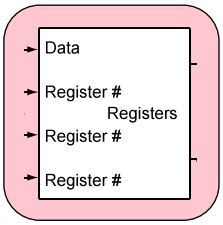
What state & logic do we need to execute instructions?

## What do we need to execute instructions?

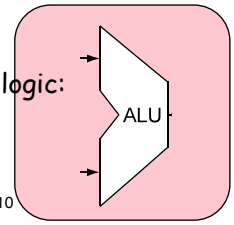
Which instruction?
• instruction memory, PC: `beq, j`

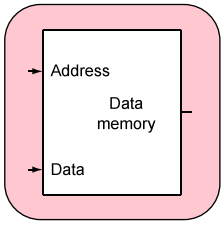



Which registers?
• register storage file




Which Math?
• combinational logic:
`add, sub, and, or, slt`



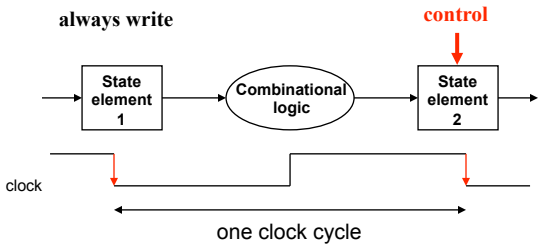




Which data?
• memory:
`lw, sw`



## Remember: State & Clocking Review

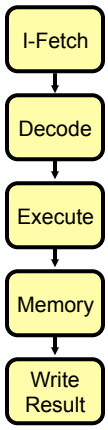
- Clocking methodology defines when to read/write a signal
  - An edge-triggered methodology
- Typical execution
  - read contents of state elements
  - send values through combinational logic
  - write results to one or more state elements
  - No control: always write or
  - Control the Write (write only on "1")



## What is Instruction Execution?

**5 logical steps**

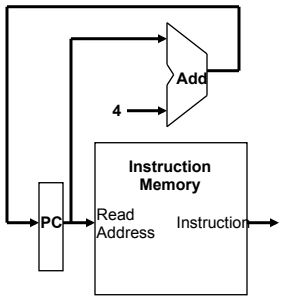
- IF: fetch instruction
- ID/R: decode instruction and read registers
- EX: execute (add, sub, …)
- MEM: access memory
- WB: store result (write back)






## Fetch the Next Instruction

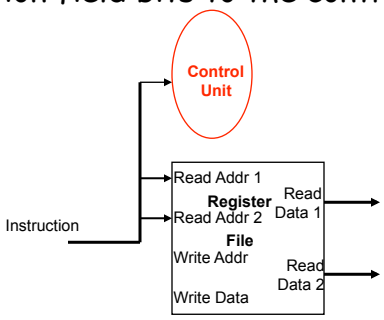
- read the instruction from the Instruction Memory
- update the PC to hold the address of the next instruction




- No write control because
  - The processor updates the PC every cycle
  - The processor reads instruction memory every cycle

# Decode the Instruction

- send the opcode of the fetched instruction and function field bits to the control unit

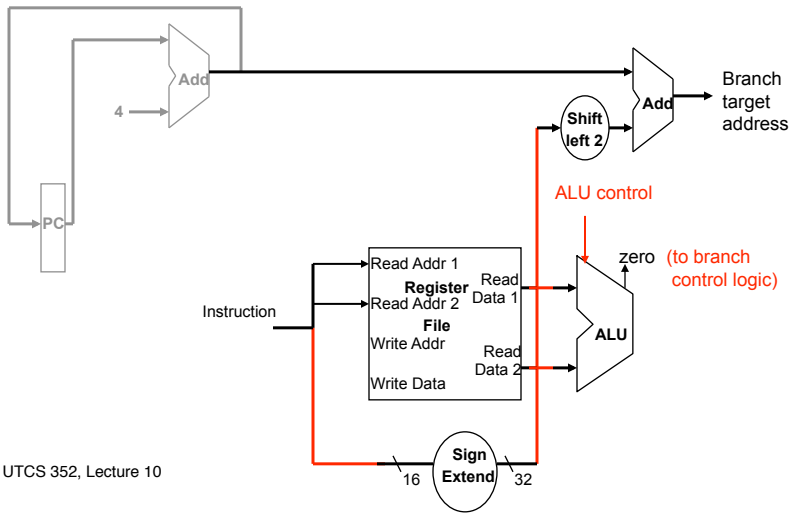



- read two values from the Register File
  - Register File identifiers are in the instruction



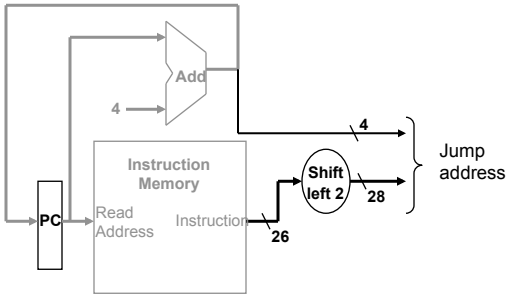
# Executing Branch Operations

- compare registers + compute branch target



# Executing Jump Operations

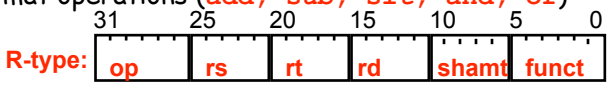
- replace the lower 28 bits of the PC with the lower 26 bits of the fetched instruction shifted left by 2 bits



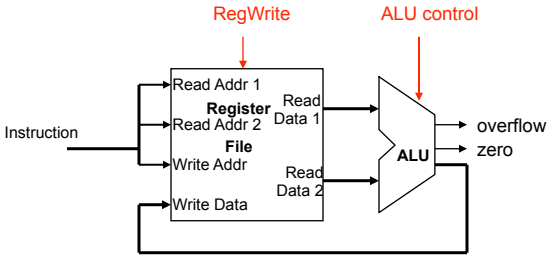



# Executing R Format Operations

- R format operations (`add, sub, slt, and, or`)



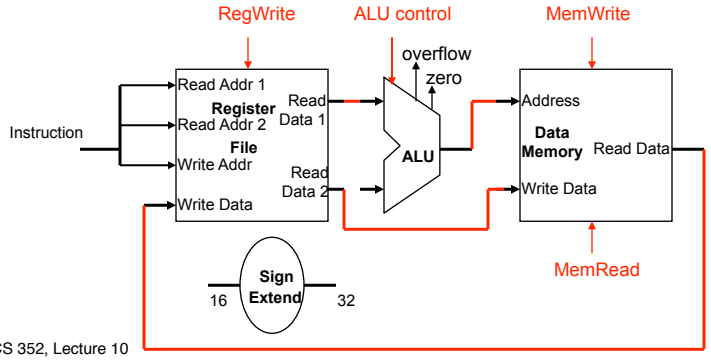

  - perform the (op and funct) operation on values in rs and rt
  - store the result back into the Register File (into location rd)




  - The Register File is not written every cycle (e.g. `sw`), so we need an explicit write control signal for the Register File

## Executing Load and Store Operations

- compute memory address: add base register (read from Register File during decode) to 16-bit signed-extended offset field
- store write to Data Memory value (read from Register File during decode)
- load write to Register File value, read from Data Memory






## Summary

- Break down of instruction execution and logic based on needed functionality
- Next Time
  - Putting it all together and adding control logic
  - Pipelining
  - Homework #4 is due Tuesday March 2, 2010
- Reading: P&H 4.5-6