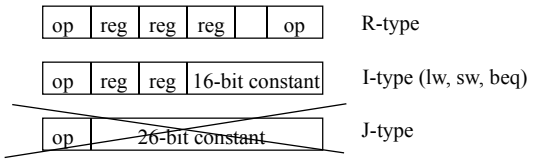
# Designing the Processor

## Datapath & Control (review of Sec. 4.1 – 4.4)



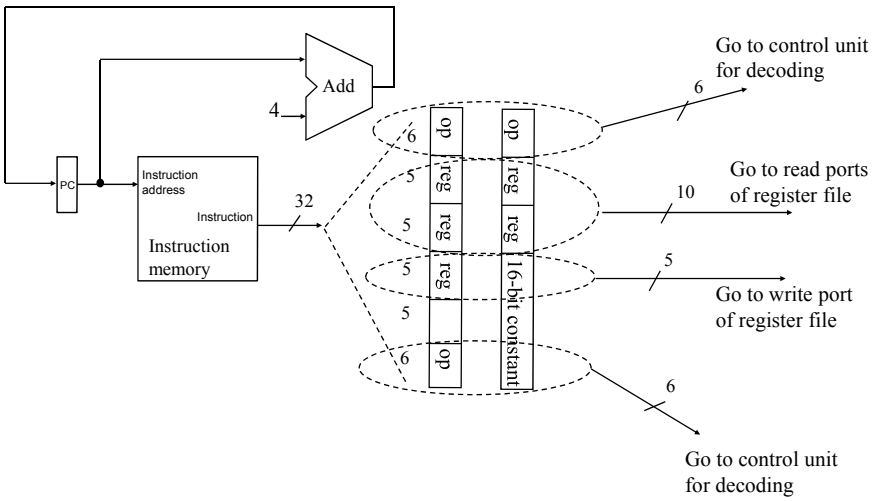
## Review of Single cycle implementation

- **Simplified to contain only:**
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq`
- **Generic Implementation:**
  - The program counter (PC) supplies instruction address
  - get the instruction from the "instruction memory"
  - the op-code determines exactly what to do.



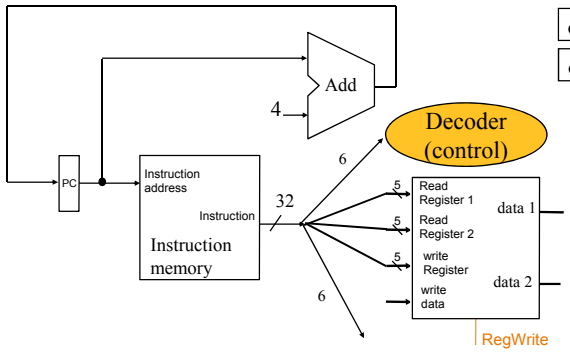

MIPS instruction format and architecture

# Fetching instruction and incrementing PC






# The first two steps of executing instructions




| op | reg | reg | reg | | op | R-type |
|---|---|---|---|---|---|---|
| op | reg | reg | 16-bit constant | | | I-type |

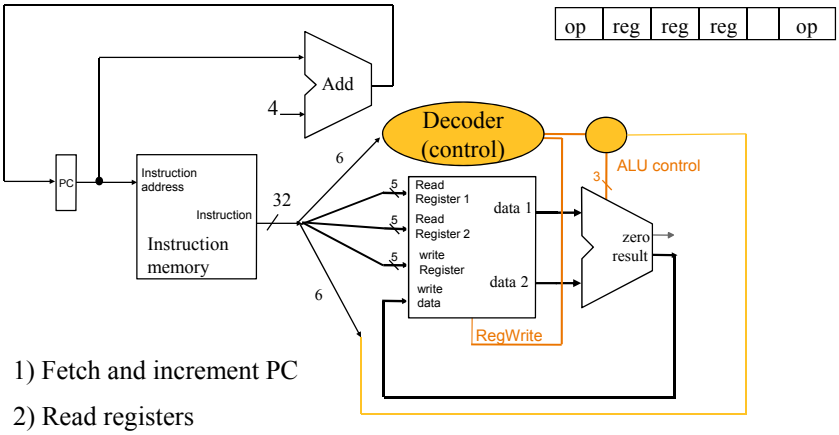
1) Fetch and increment PC

2) Read registers

The control unit should decode the op-code to determine what to do next

- depends on instruction type

# Executing *R-type* instructions (in one cycle)



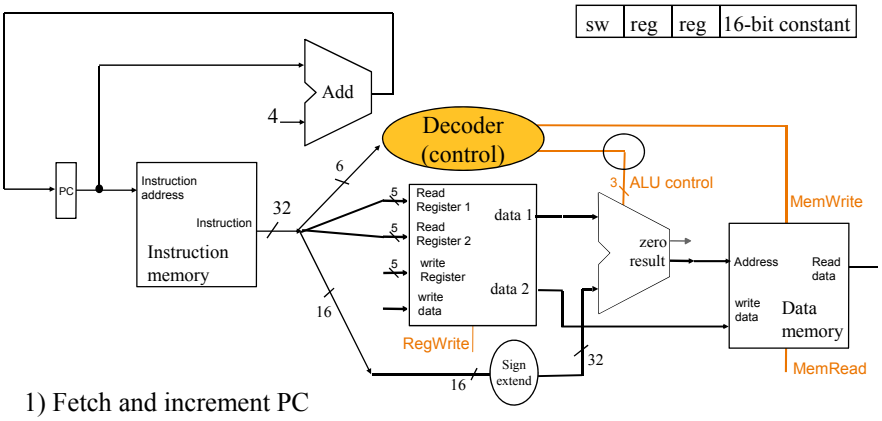

1) Fetch and increment PC
2) Read registers
3) Execute operation
4) Store result

The control unit issues:

- the corresponding ALU control
- RegWrite signal

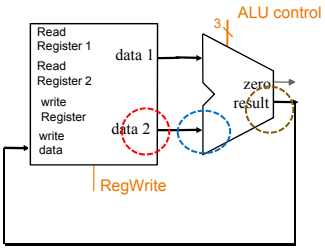


# Executing *sw* instructions

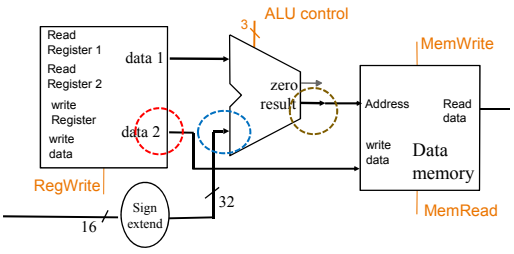



1) Fetch and increment PC
2) Read registers
3) Calculate memory address (issue ALU control signals – add)
4) Write data to memory (issue MemWrite signal)

# Data path conflicts




Path for R-type




Path for sw

1) The second input to the ALU comes from two different places

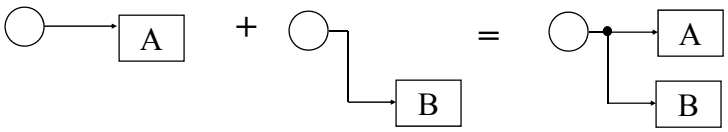
2) The data2 output goes to two different places

3) The output of the ALU goes to two different places

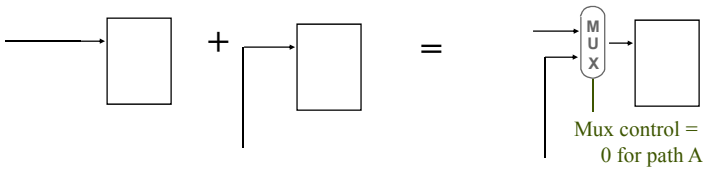


# Reconciling data paths

1) No problems in fanning out one signal to two points

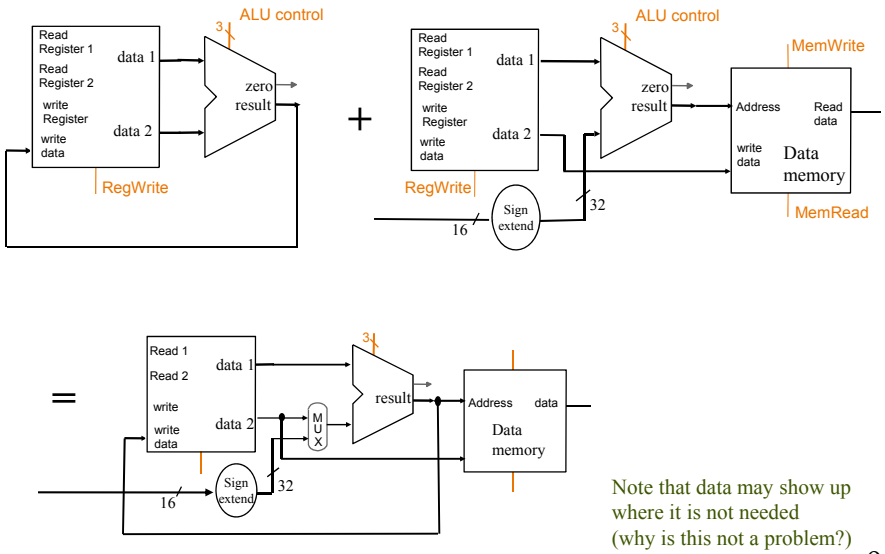



2) Cannot connect two signals to one point (should use multiplexers )




Path A Path B

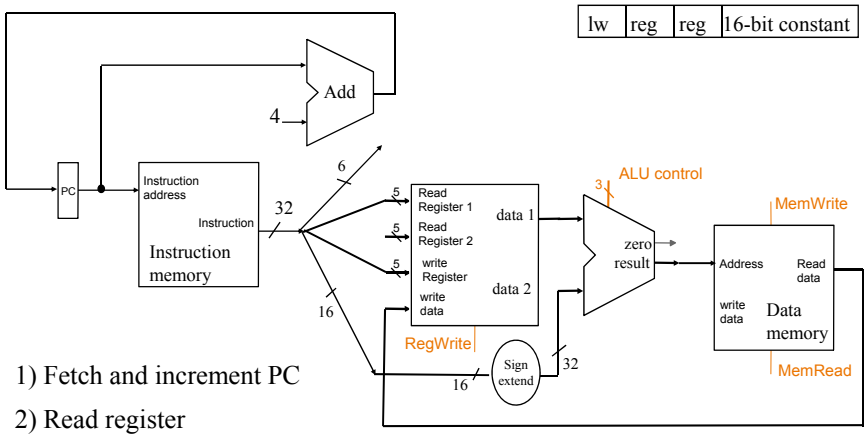
# Resolving data path conflicts




Note that data may show up where it is not needed (why is this not a problem?)

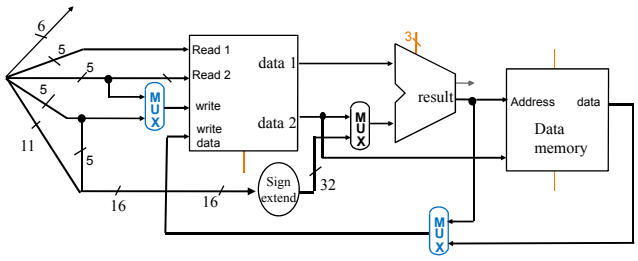


# Executing *lw* instructions (in one cycle)



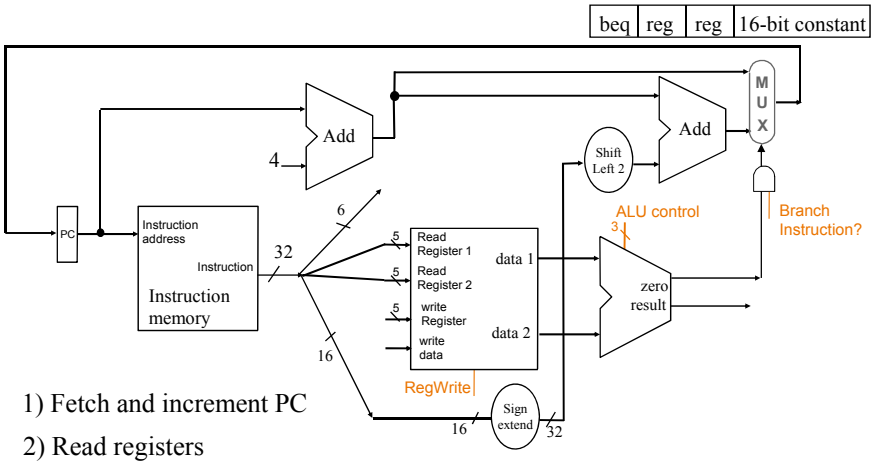

1) Fetch and increment PC
2) Read register
3) Calculate memory address (issue ALU control signals – add)
4) Read data from memory (issue MemRead signal)
5) Store data in register (issue Regwrite signal)

# Reconciling *R-type*, *sw* and *lw* data paths





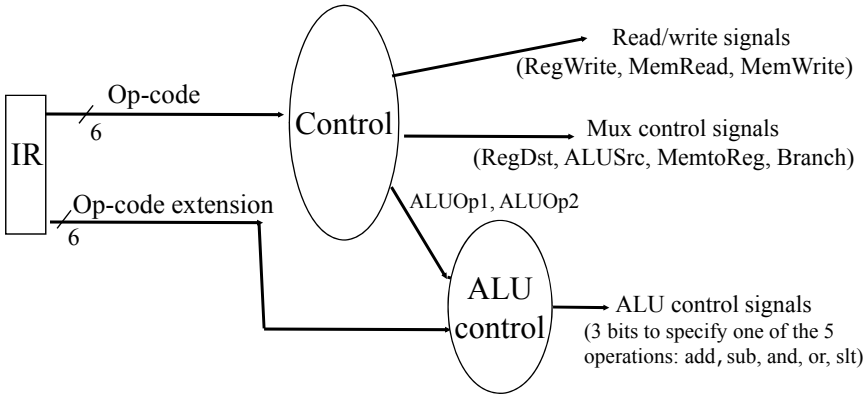

# Executing *beq* instructions



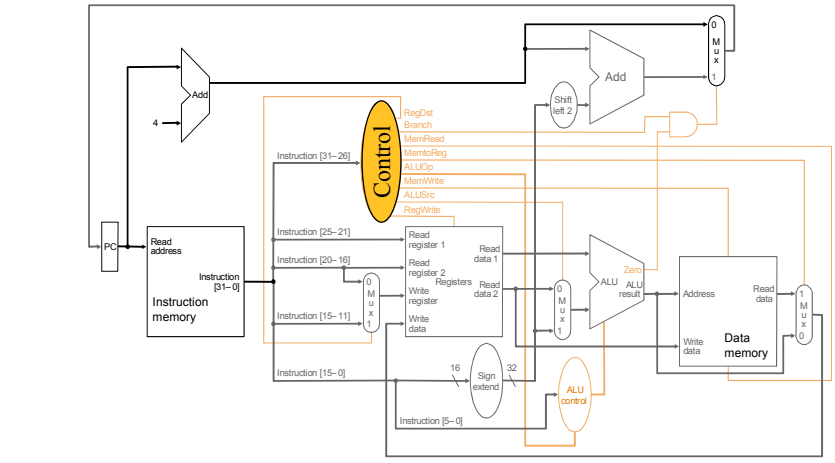

1) Fetch and increment PC
2) Read registers
3) Calculate branch condition (issue ALU control signals – subtract)
4) Calculate target address
5) Select next PC depending on the result in 3

# Control

- **Using the op-code from the instruction, the control issues signals to:**
  - select the operations to perform (ALU, read/write to registers and memory)
  - control the flow of data (multiplexor inputs).



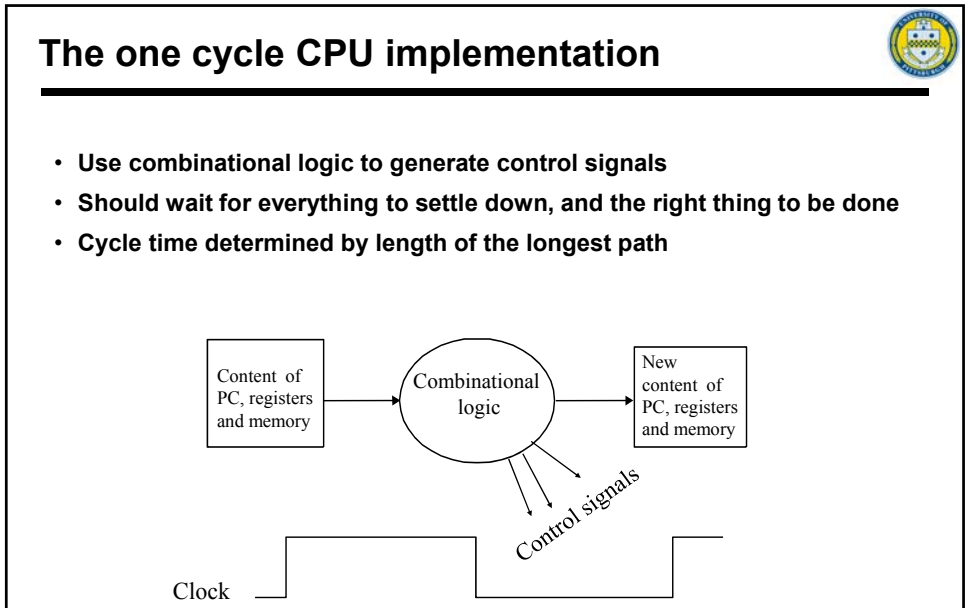



Control inputs ⟷ Control outputs

| Instruction | RegDst | ALUSrc | Memto-Reg | Reg Write | Mem Read | Mem Write | Branch | ALUOp1 | ALUp0 | |
|---|---|---|---|---|---|---|---|---|---|---|
| R-format | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | X | → Consult op-code extension |
| lw | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | → add |
| sw | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 | |
| beq | X | 0 | X | 0 | 0 | 0 | 1 | 0 | 1 | → subtract |

# The one cycle CPU implementation



- **Use combinational logic to generate control signals**
- **Should wait for everything to settle down, and the right thing to be done**
- **Cycle time determined by length of the longest path**

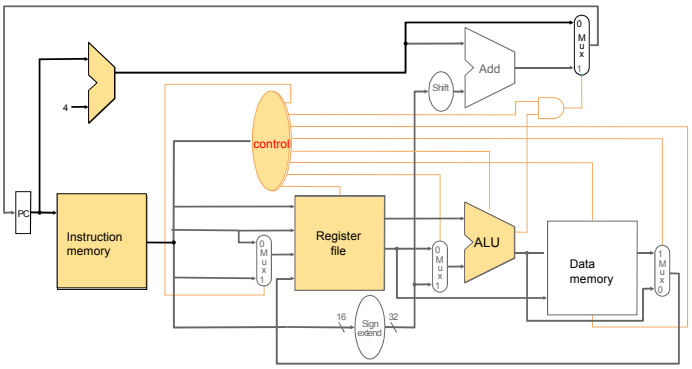





# Single Cycle Implementation

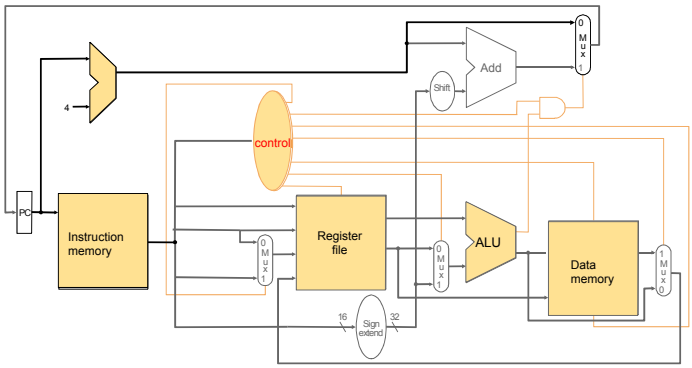
Assume negligible delays for all components except:
memory (0.2ns), ALU and adders (0.2ns), register file access (0.1ns)

- **Question:** calculate the time to execute an ***R-type*** instruction
- **Answer:** 0.2ns to read instruction + 0.1ns to read registers + 0.2ns for ALU operation + 0.1 n.sec to write into register = 0.6ns



# Single Cycle Implementation (lw)

- **Similarly, the following is needed to execute the *lw* instructions:**
- 0.2ns to read instruction + 0.1ns to read registers + 0.2ns for ALU operation + 0.2ns to read from data memory + 0.1 n.sec to write into register = 0.8ns.




- Can calculate that *sw* takes 0.7ns, and *branch* take 0.5ns
- **Hence**, the cycle time of the machine should be large enough to execute every instruction. That is 0.8ns.