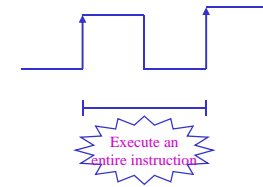


# Designing a Single Cycle Datapath

or  
*The Do-It-Yourself CPU Kit*

# The Big Picture: The Performance Perspective

- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Starting today:
  - Single cycle processor:
    - Advantage: One clock cycle per instruction
    - Disadvantage: long cycle time



• ET = ?

# The Processor: Datapath & Control

- We're ready to look at an implementation of the MIPS simplified to contain only:
  - memory-reference instructions:
  - arithmetic-logical instructions: `add`, `sub`, `and`, `or`, `slt`
  - control flow instructions:
- Generic Implementation:
  - use the `PC` to supply instruction address
  - get the `PC` from memory
  - read registers
  - use the instruction to decide exactly what to do
- All instructions use the ALU after reading the registers
  - memory-reference? arithmetic? control flow?

# The MIPS Subset

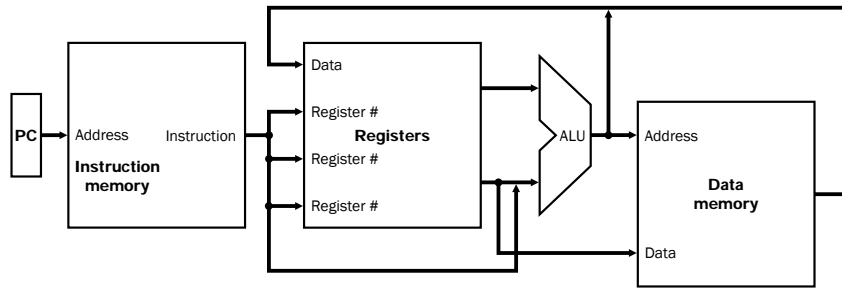
- R-type
  - `add rd, rs, rt`
  - `sub, and, or, slt`

31	26	21	16	11	6	0
op	rs	rt	rd	shamt	funct	
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
- LOAD and STORE
  - `lw rt, rs, imm16`
  - `sw rt, rs, imm16`

31	26	21	16	0
op	rs	rt	immediate	
6 bits	5 bits	5 bits	16 bits	
- BRANCH:
  - `beq rs, rt, imm16`

31	26	21	16	0
op	rs	rt	displacement	
6 bits	5 bits	5 bits	16 bits	

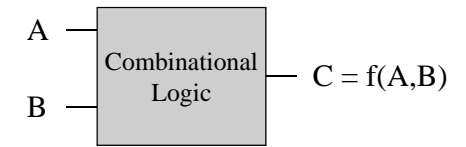
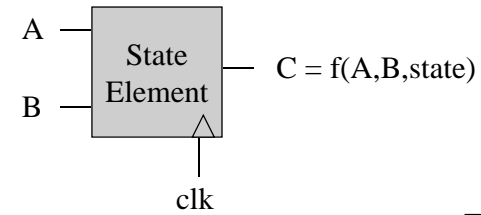
## Where We're Going - The High-level View



CSE 141

Dean Tullsen

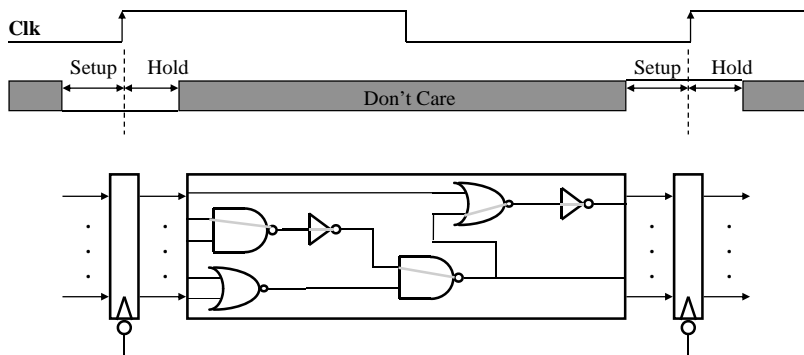
## Review: Two Types of Logic Components



CSE 141

Dean Tullsen

## Clocking Methodology



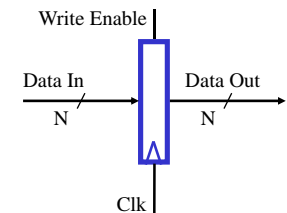
- All storage elements are clocked by the same clock edge

CSE 141

Dean Tullsen

## Storage Element: Register

- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable:
    - 0: Data Out will not change
    - 1: Data Out will become Data In (on the clock edge)

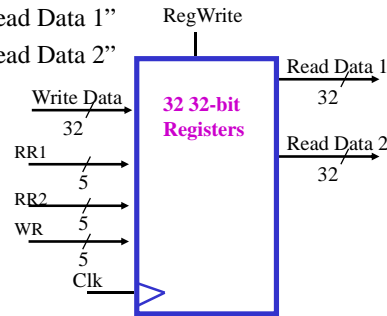


CSE 141

Dean Tullsen

## Storage Element: Register File

- Register File consists of (32) registers:
  - Two 32-bit output buses:
  - One 32-bit input bus: busW
- Register is selected by:
  - selects the register to put on bus “Read Data 1”
  - selects the register to put on bus “Read Data 2”
  - selects the register to be written via WriteData when RegWrite is 1
- Clock input (CLK)

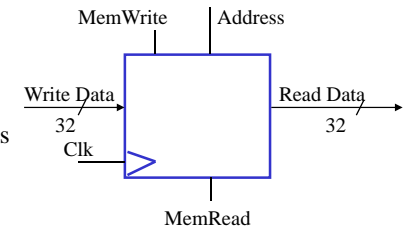


CSE 141

Dean Tullsen

## Storage Element: Memory

- Memory
  - Two input buses: WriteData, Address
  - One output bus: ReadData
- Memory word is selected by:
  - Address selects the word to put on ReadData bus
  - MemWrite = 1: address selects the memory word to be written via the WriteData bus
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - Address valid => ReadData valid after “access time.”



CSE 141

Dean Tullsen

## Register Transfer Language (RTL)

- is a mechanism for describing the movement and manipulation of data between storage elements:

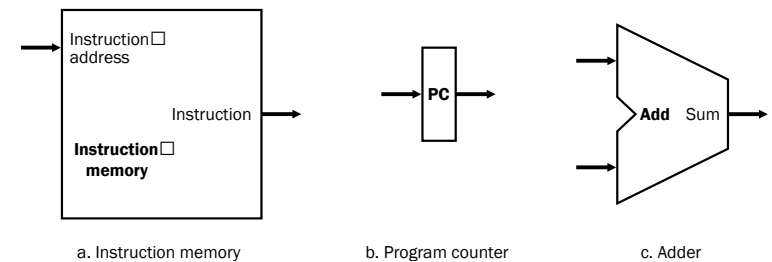
```

R[3] <- R[5] + R[7]
PC <- PC + 4 + R[5]
R[rd] <- R[rs] + R[rt]
R[rt] <- Mem[R[rs] + immed]
    
```

CSE 141

Dean Tullsen

## Instruction Fetch and Program Counter Management

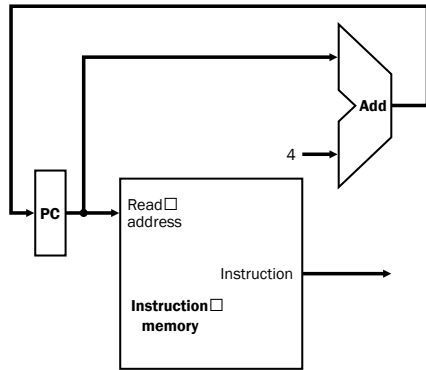


CSE 141

Dean Tullsen

# Overview of the Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction:  $inst \leftarrow mem[PC]$
  - Update the program counter:
    - Sequential Code:  $PC \leftarrow PC + 4$
    - Branch and Jump  $PC \leftarrow \text{“something else”}$

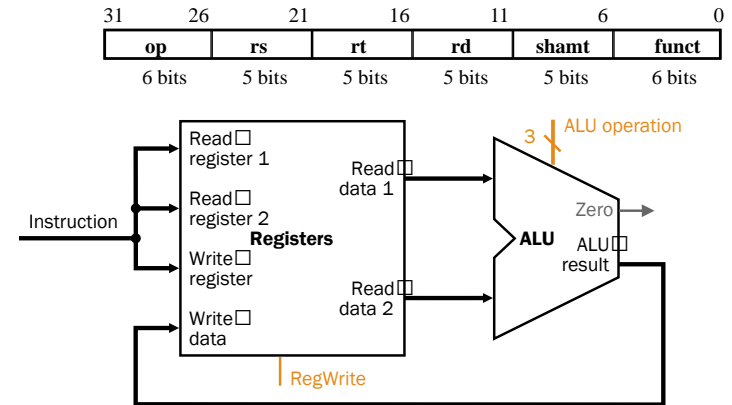


CSE 141

Dean Tullsen

# Datapath for Register-Register Operations

- $R[rd] \leftarrow R[rs] \text{ op } R[rt]$  Example: *add rd, rs, rt*
  - RR1, RR2, and WR comes from instruction's rs, rt, and rd fields
  - and : control logic after decoding instruction

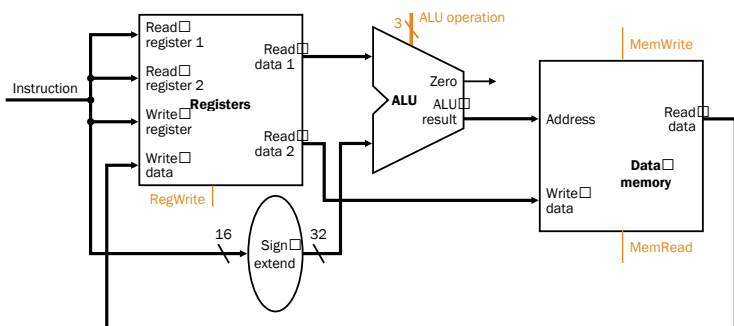
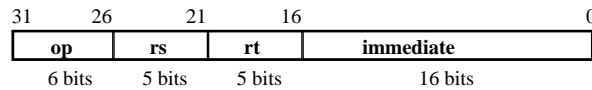


CSE 141

Dean Tullsen

# Datapath for Load Operations

$R[rt] \leftarrow Mem[R[rs] + SignExt[imm16]]$  Example: *lw rt, rs, imm16*

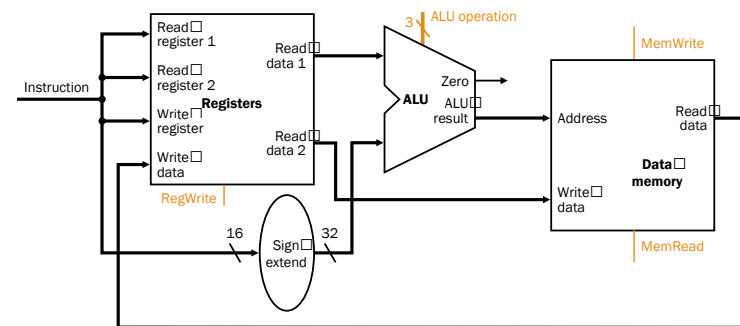
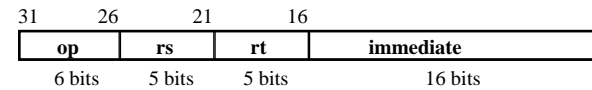


CSE 141

Dean Tullsen

# Datapath for Store Operations

$Mem[R[rs] + SignExt[imm16]] \leftarrow R[rt]$  Example: *sw rt, rs, imm16*

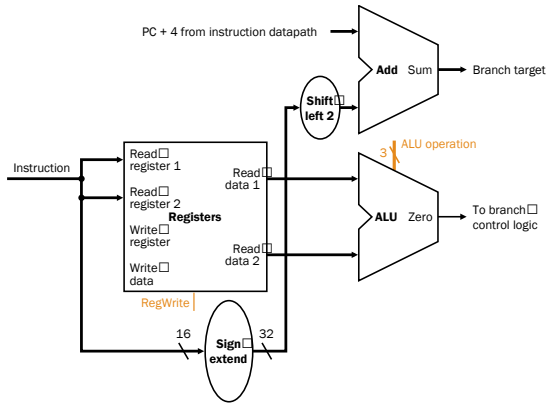
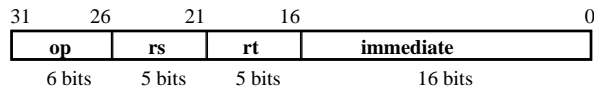


CSE 141

Dean Tullsen

# Datapath for Branch Operations

$Z \leftarrow (rs == rt); \text{ if } Z, PC = PC + 4 + \text{imm16}; \text{ else } PC = PC + 4$   
*beq rs, rt, imm16*



CSE 141

Dean Tullsen

# Binary Arithmetic for the Next Address

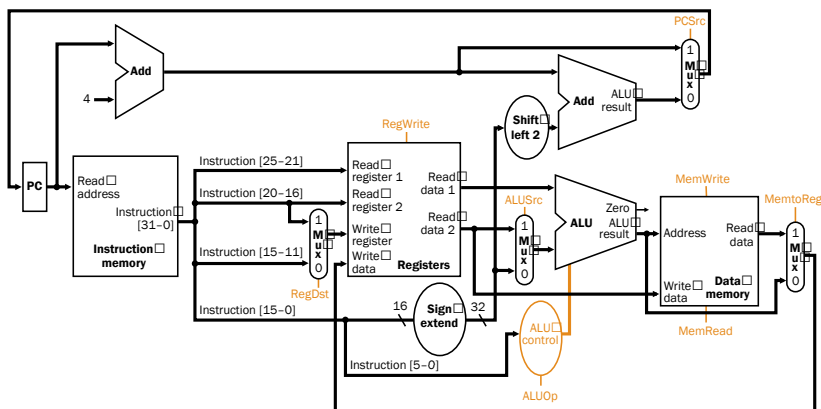
- In theory, the PC is a 32-bit byte address into the instruction memory:
  - Sequential operation:  $PC\langle 31:0 \rangle = PC\langle 31:0 \rangle + 4$
  - Branch operation:  $PC\langle 31:0 \rangle = PC\langle 31:0 \rangle + 4 + \text{SignExt}[\text{Imm16}] * 4$
- The magic number “4” always comes up because:
  - The 32-bit PC is a byte address
  - And all our instructions are 4 bytes (32 bits) long
  - The 2 LSBs of the 32-bit PC are always zeros
  - There is no reason to have hardware to keep the 2 LSBs
- In practice, we can simplify the hardware by using a 30-bit  $PC\langle 31:2 \rangle$ :
  - Sequential operation:  $PC\langle 31:2 \rangle = PC\langle 31:2 \rangle + 1$
  - Branch operation:  $PC\langle 31:2 \rangle = PC\langle 31:2 \rangle + 1 + \text{SignExt}[\text{Imm16}]$
  - In either case: Instruction Memory Address =  $PC\langle 31:2 \rangle$  concat “00”

CSE 141

Dean Tullsen

# Putting it All Together: A Single Cycle Datapath

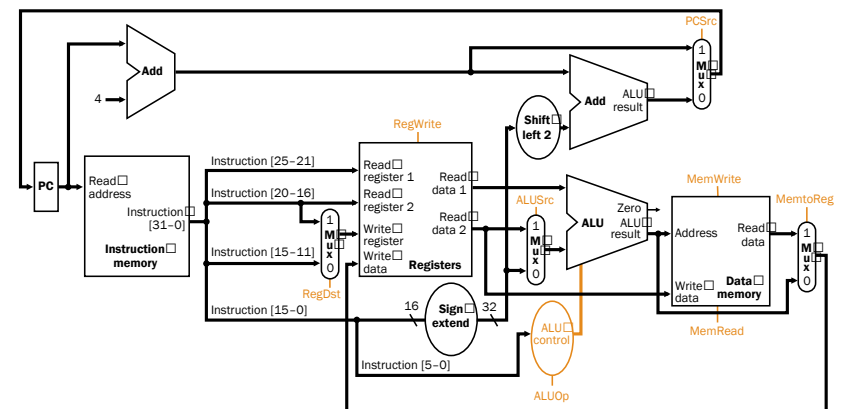
- We have everything except control signals



CSE 141

Dean Tullsen

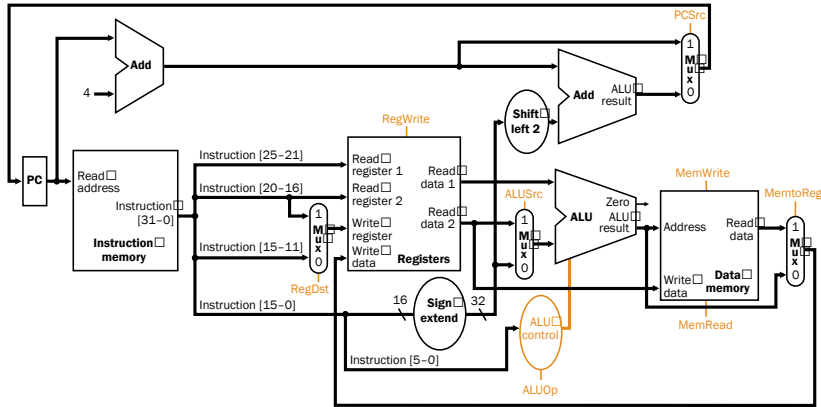
# The R-Format (e.g. add) Datapath



CSE 141

Dean Tullsen

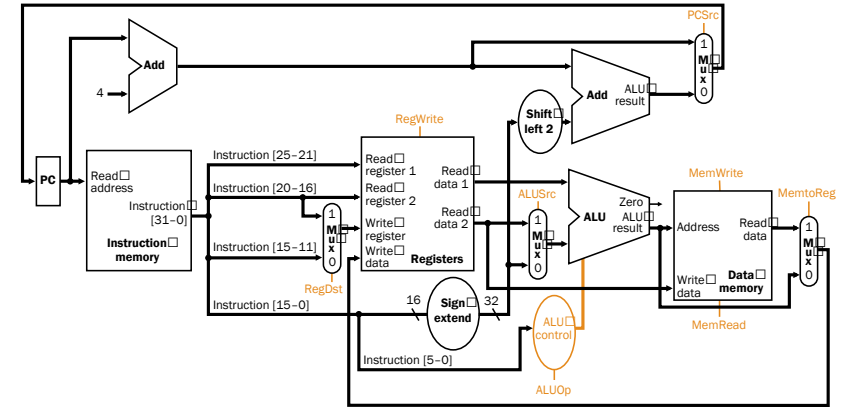
## The Load Datapath



CSE 141

Dean Tullsen

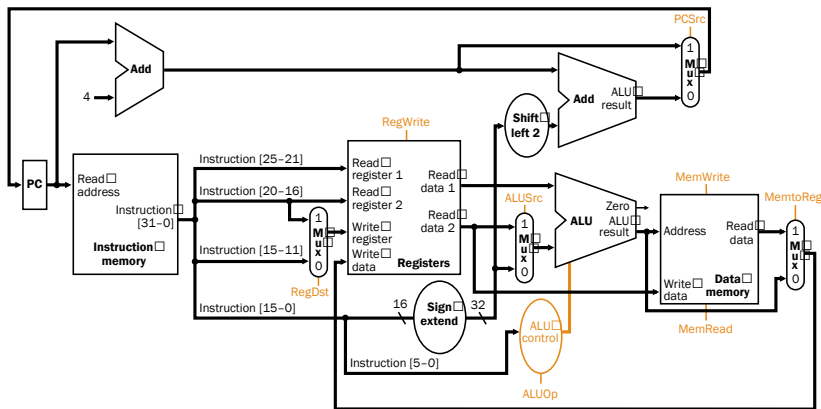
## The store Datapath



CSE 141

Dean Tullsen

## The beq Datapath



CSE 141

Dean Tullsen

## Key Points

- CPU is just a collection of state and combinational logic
- We just designed a very rich processor, at least in terms of functionality
- $ET = IC * CPI * Cycle Time$ 
  - where does the single-cycle machine fit in?

CSE 141

Dean Tullsen