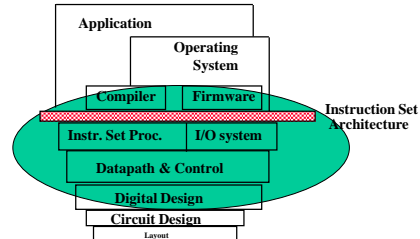
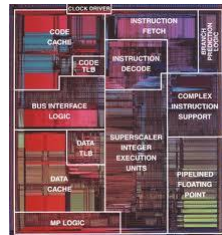


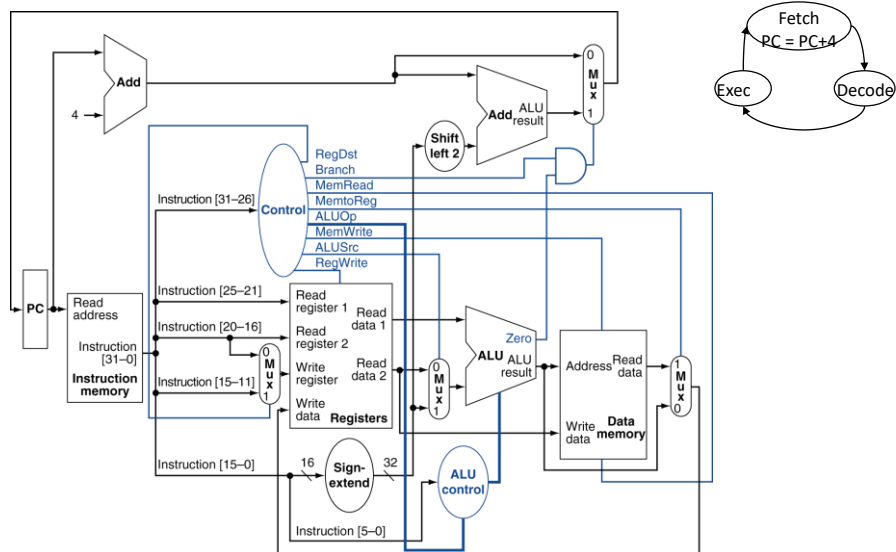
CS/SE 3340

Computer Architecture



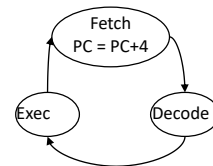
Building the Processor – Data Path & Control

Adapted from slides by Profs. D. Patterson and J. Hennessey



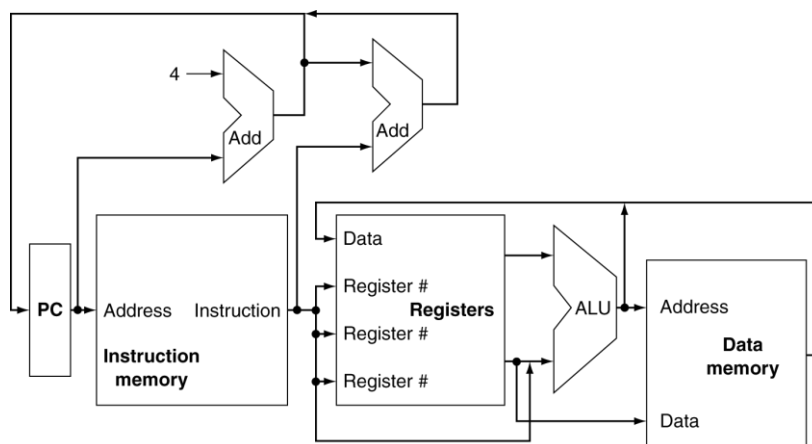
Instruction Execution

- PC → instruction memory, fetch instruction
 - PC is automatically updated to PC + 4 (*why?*)
- Register numbers → register file, read/write registers
- Depending on instruction type
 - Use **ALU** to calculate
 - Arithmetic result
 - Memory address for load/store
 - Branch target address
 - Access data memory for load/store
 - PC ← target address or PC + 4: *next instruction*



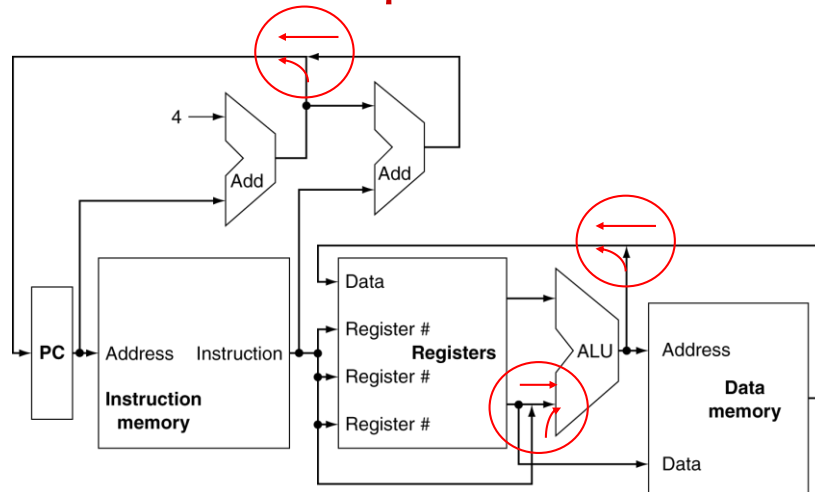
3

CPU Overview



4

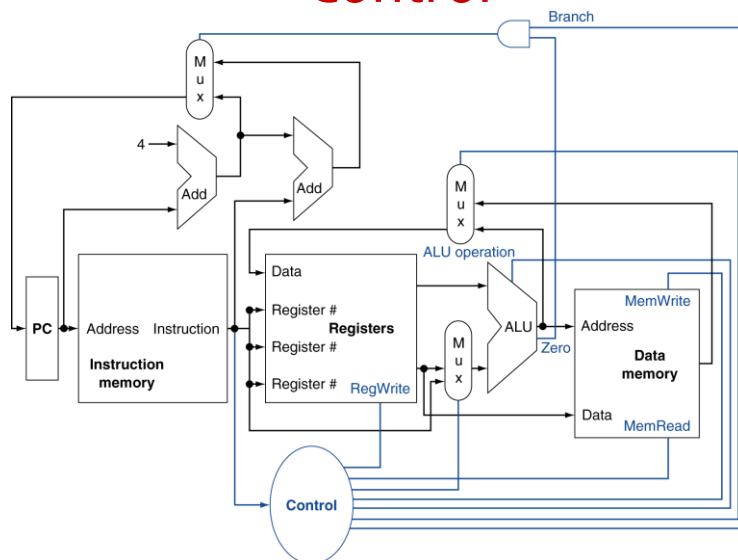
Multiplexers



- Can't just join wires together
- Must use *multiplexers*

5

Control



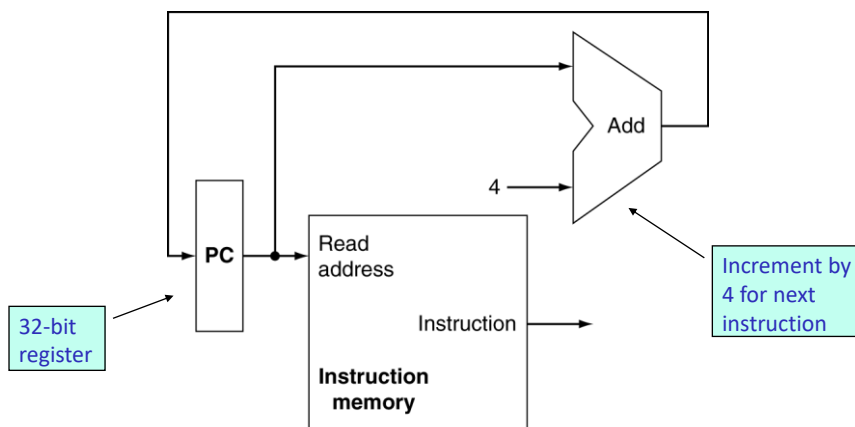
6

Building a Datapath

- Datapath
 - Elements that process data and **addresses** in the CPU
 - Registers, ALUs, mux's, memories, ...
- We will build a MIPS datapath incrementally
 - Refining the overview design

7

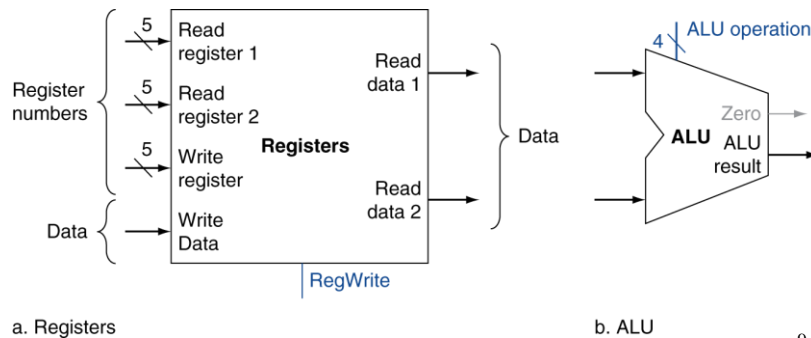
Instruction Fetch



8

R-Format Instructions

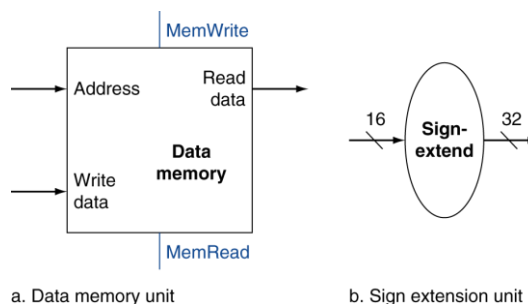
- Read two register operands
- Perform arithmetic/logical operation
- Write register result



9

Load/Store Instructions

- Read register operands
- Calculate address using 16-bit offset
 - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory



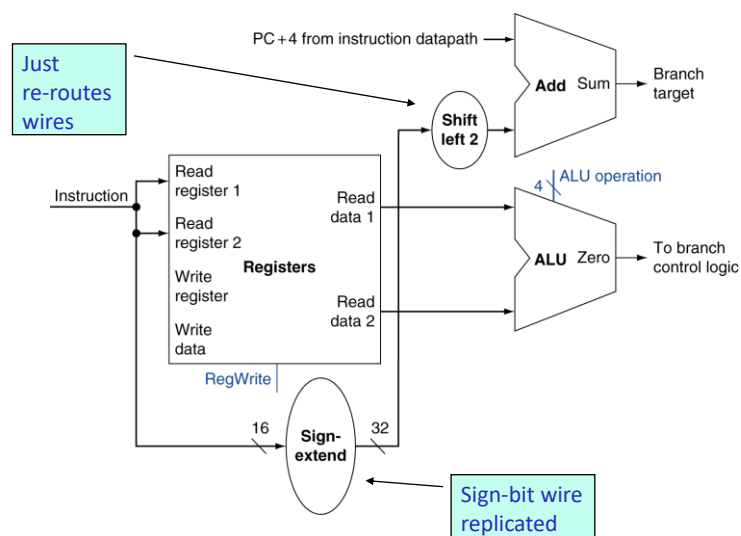
10

Branch Instructions

- Read register operands
- Compare operands
 - Use ALU, subtract and check Zero output
- Calculate target address
 - Sign-extend displacement
 - Shift left 2 places (word displacement)
 - Add to PC + 4
 - Already calculated by instruction fetch

11

Branch Instructions



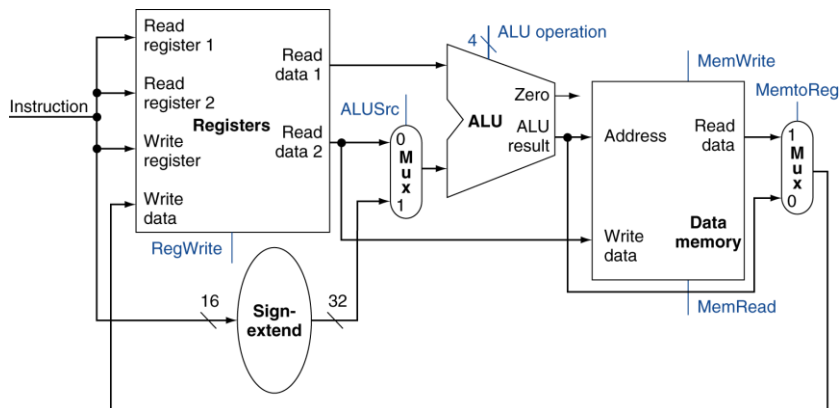
12

Composing the Elements

- First-cut data path does an instruction in one clock cycle
 - Each datapath element can only do one function at a time
 - Hence, we need separate instruction and data memories
- Use multiplexers where alternate data sources are used for different instructions

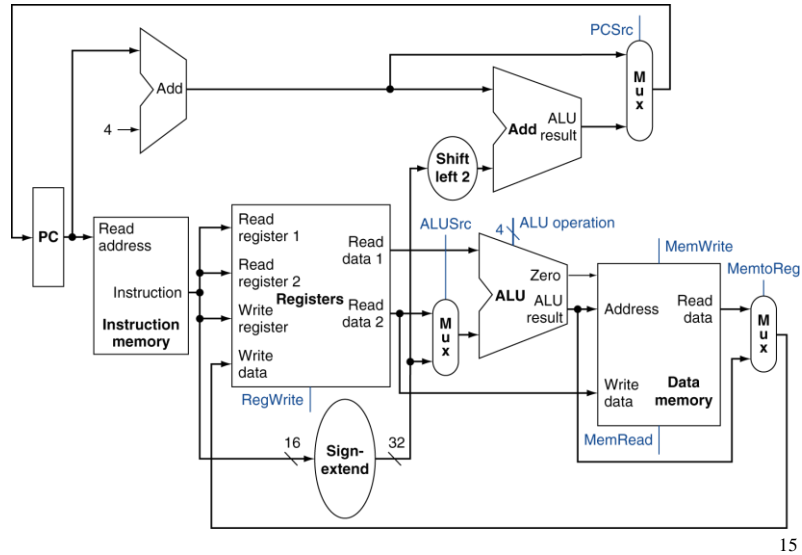
13

R-Type/Load/Store Datapath



14

Full Datapath



ALU Control

- ALU is used for
 - Load/Store: F = add
 - Branch: F = subtract
 - R-type: F depends on funct field

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

16

ALU Control

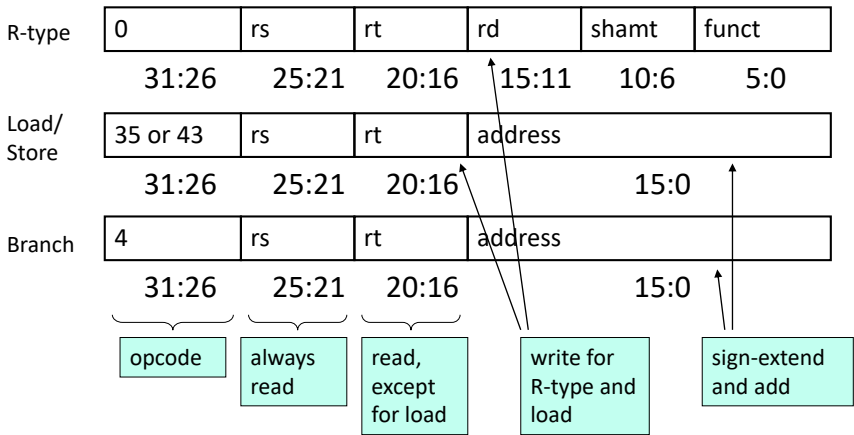
- Assume 2-bit ALUOp derived from opcode
 - Combinational logic derives ALU control

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

17

The Main Control Unit

- Control signals derived from instruction



18

Datapath With Control

The diagram illustrates a CPU datapath with control logic. Key components and their interactions are as follows:

- PC (Program Counter):** Provides the address for the instruction memory.
- Instruction memory:** Provides the instruction based on the PC address. The instruction is split into fields: [31-0], [25-21], [20-16], [15-11], and [5-0].
- Control:** A central unit that receives the instruction fields and generates control signals: RegDst, Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, and RegWrite.
- Registers:**
 - Read registers:** Read register 1 and Read register 2 provide data to the ALU.
 - Write register:** Receives data from the ALU and the data memory.
- ALU (Arithmetic Logic Unit):**
 - ALU control:** Receives the ALUOp signal and controls the ALU operation.
 - Sign-extend:** Takes the 5-bit ALUSrc field and extends it to 32 bits.
 - Mux (Multiplexer):** Selects between the 32-bit ALUSrc field and the 32-bit ALU result.
 - Zero:** A flag that indicates if the ALU result is zero.
- Data memory:**
 - Read data:** Provides data to the ALU based on the ALUSrc field.
 - Write data:** Receives data from the ALU and the data memory.
- Other components:**
 - Add:** Adds the PC and the 4-bit Branch field to calculate the next PC value.
 - Shift left 2:** Shifts the ALU result left by 2 bits to calculate the branch target address.
 - Mux (Multiplexer):** Selects between the 32-bit ALU result and the 32-bit branch target address.

The diagram shows the flow of data and control signals, illustrating how the datapath executes instructions and updates the PC.



R-Type Instruction

The diagram illustrates the execution flow of an R-Type instruction in a processor. The components and their interactions are as follows:

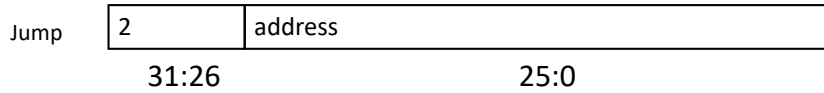
- PC (Program Counter):** Provides the initial address to the **Instruction memory**.
- Instruction memory:** Outputs the instruction based on the **Read address** from the PC.
- Control:** Receives the instruction and outputs control signals: *RegDst*, *Branch*, *MemRead*, *MemtoReg*, *ALUOp*, *MemWrite*, *ALUSrc*, and *RegWrite*.
- Registers:**
 - Read register 1:** Outputs *Read data 1* to the **ALU**.
 - Read register 2:** Outputs *Read data 2* to the **ALU**.
 - Write register:** Receives the *Write data* from the **ALU** and the *RegDst* control signal.
- ALU (Arithmetic Logic Unit):**
 - Inputs: *Read data 1*, *Read data 2*, and the *ALUSrc* control signal.
 - Control: Receives *ALUOp* from the **Control** unit.
 - Output: *ALU result*.
- MUX (Multiplexer):**
 - Write data MUX:** Selects the *Write data* for the **Registers** based on the *RegDst* control signal.
 - Read data MUX:** Selects the *Read data* for the **PC** based on the *MemtoReg* control signal.
- PC Update:** The **PC** is updated with the *Read data* from the **Read data MUX**.
- Branch:** The *Branch* control signal is ANDed with the *Zero* flag from the **ALU** to determine if the **PC** should be updated with the *ALU result*.
- Sign-extend:** The *Instruction [5-0]* is sign-extended to 32 bits and used as the *ALUSrc* input to the **ALU**.
- ALU control:** Receives the *ALUOp* control signal and provides control to the **ALU**.
- Shift left 2:** The *ALU result* is shifted left by 2 bits to be used as the *Write data* for the **Registers**.
- ALU result MUX:** Selects the *ALU result* for the *Write data* MUX based on the *MemWrite* control signal.
- Zero:** The *Zero* flag from the **ALU** is used in the branch logic.



Load Instruction

Branch-on-Equal Instruction

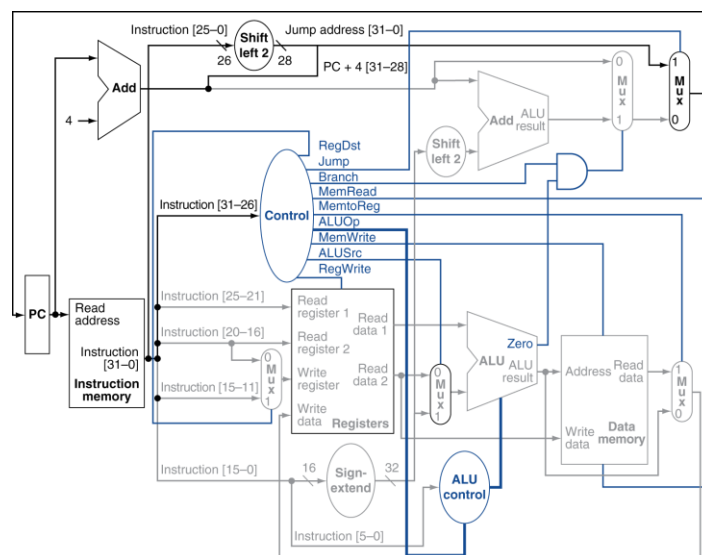
Implementing Jumps



- Jump uses word address
- Update PC with concatenation of
 - Top 4 bits of old PC
 - 26-bit jump address
 - 00
- Need an extra control signal decoded from opcode

23

Datapath With Jumps Added



24

Logic Design Basics

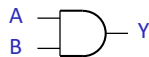
- Information encoded in binary
 - Low voltage = 0, High voltage = 1
 - One wire per bit
 - Multi-bit data encoded on multi-wire buses
- Combinational elements
 - Operate on data
 - Output is a function of input
- State (sequential) elements
 - Store information, behavior depends on state

25

Combinational Elements

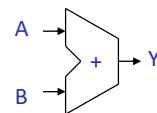
■ AND-gate

- $Y = A \& B$



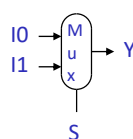
■ Adder

- $Y = A + B$



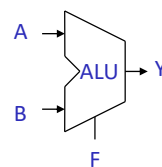
■ Multiplexer

- $Y = S ? I1 : I0$



■ Arithmetic/Logic Unit

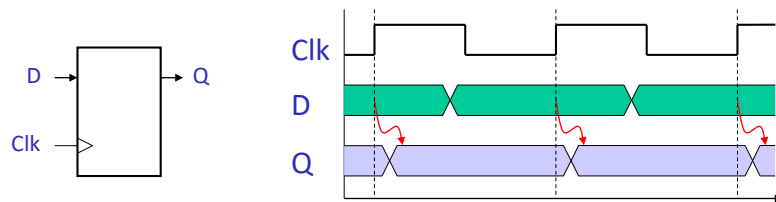
- $Y = F(A, B)$



26

Sequential Elements

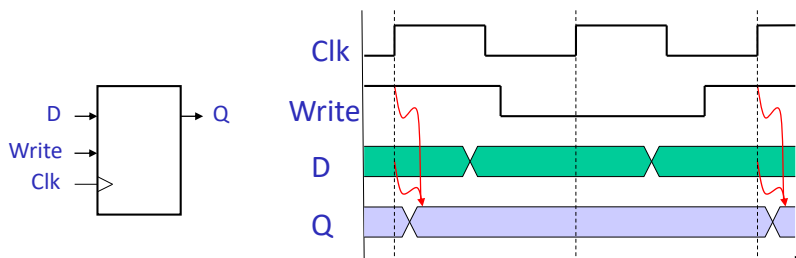
- Register: stores data in a circuit
 - Uses a *clock signal* to determine when to update the stored value
 - Edge-triggered: update when Clk changes from 0 to 1



27

Sequential Elements

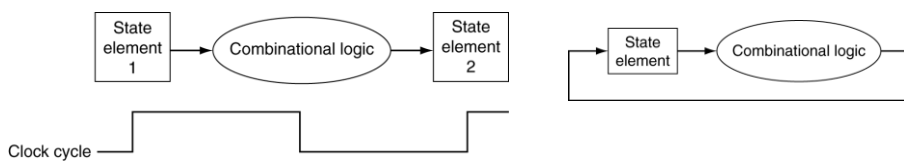
- Register with write control
 - Only updates on clock edge when write control input is 1
 - Used when stored value is required later



28

Clocking Methodology

- Combinational logic transforms data during clock cycles
 - Between clock edges
 - Input from state elements, output to state element
 - Longest delay determines clock period



29

Summary

- To build a processor we need to build a **data path** and a **control** unit
- Multiplexers are needed to select (“merge”) inputs
- A data path and control for a simple (single cycle) processor that supports most instruction types were examined
- At hardware (circuit) level **combinational** and **sequential** logics are needed for this purpose

30