Processor Design

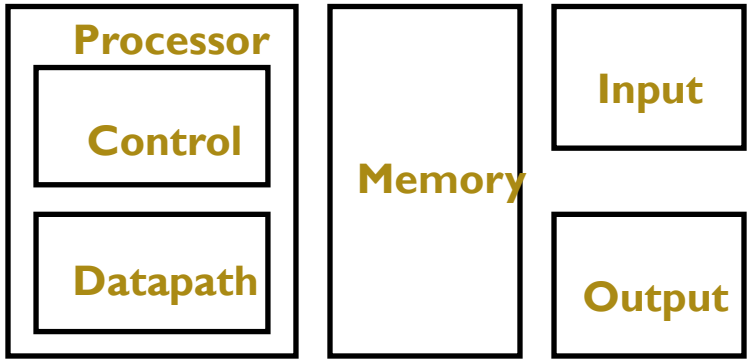
Fall 2012

CS 465

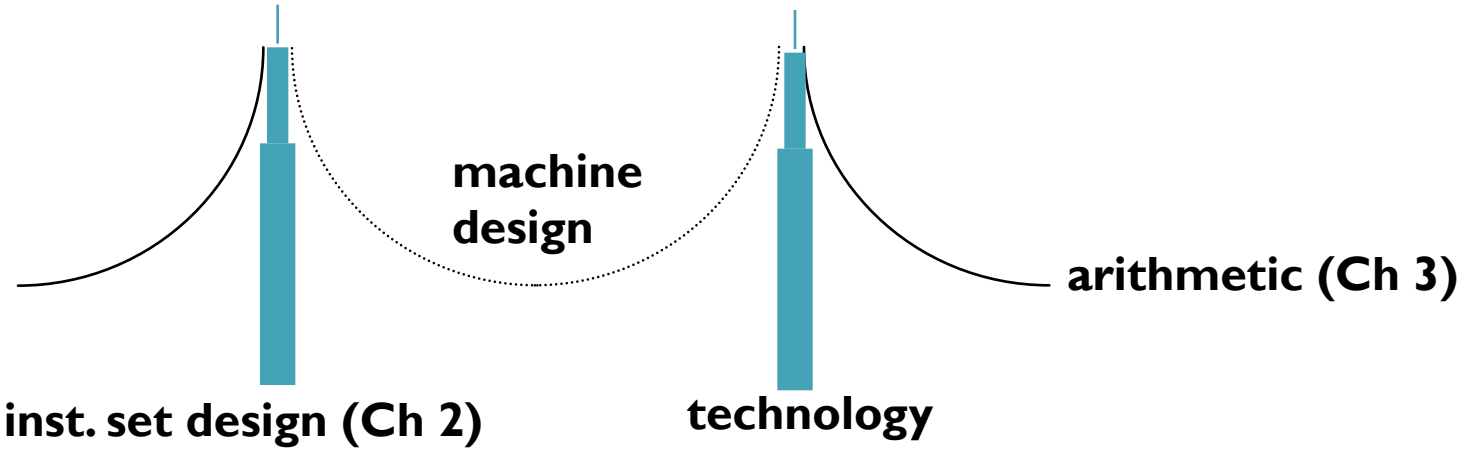
Instructor: Huzefa Rangwala, PhD

Big Picture: Where are We Now?

- Five classic components of a computer



- Today's topic: design a single cycle processor

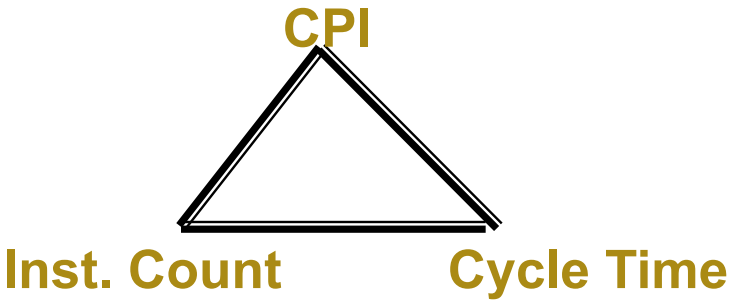


Recall:

- What does performance of a CPU depend on ?
  - Three terms, please!

The Performance Perspective

- Performance of a machine is determined by:
  - Instruction count
  - Clock cycle time
  - Clock cycles per instruction



- Processor design (datapath and control) will determine:
  - Clock cycle time
  - Clock cycles per instruction
- Today: single cycle processor
  - Advantage: one clock cycle per instruction
  - Disadvantage: long cycle time

Focus on a subset of instructions

- Simple subset, shows most aspects
  - Memory reference: `lw`, `sw`
  - Arithmetic/logical: `add`, `sub`, `and`, `or`, `slt`
  - Control transfer: `beq`, `j`

How to Design a Processor

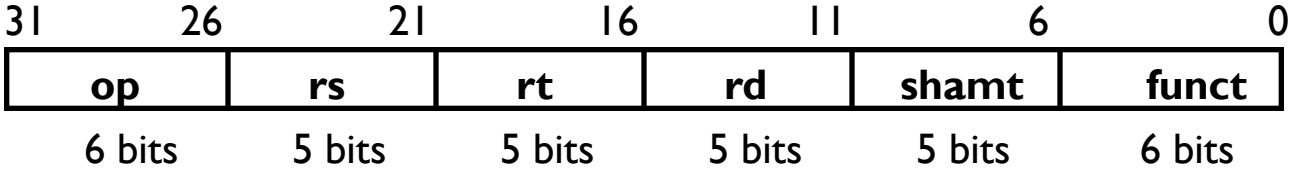
- Analyze instruction set ⇒ datapath requirements
  - Meaning of each instruction given by register transfers
  - Datapath must include storage element for ISA registers (possibly more)
  - Datapath must support each register transfer
- Select set of datapath components and establish clocking methodology
- Assemble datapath meeting the requirements
- Analyze implementation of each instruction to determine setting of control points that effects the register transfer
- Assemble the control logic

Instruction Execution

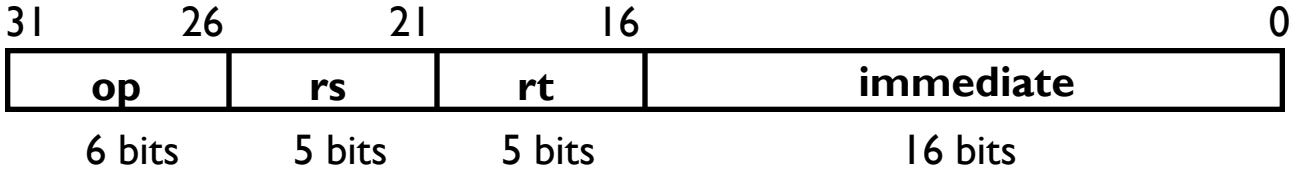
- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC ← target address or PC + 4

MIPS Instruction Formats

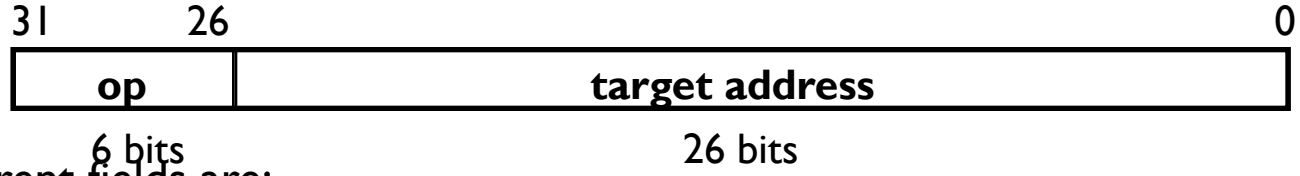
- All MIPS instructions are 32 bits long; 3 instruct. formats:
  - R-type



  - I-type



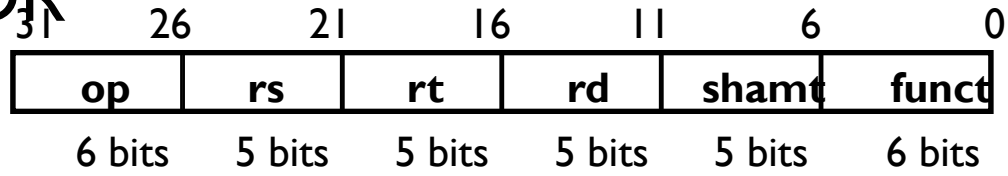
  - J-type



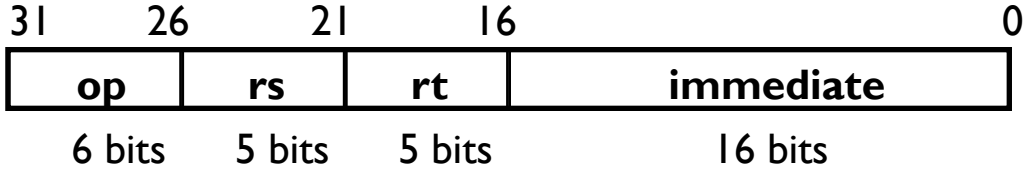
- The different fields are:
  - op: operation of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: target address of the jump instruction

Step 1a: The MIPS-lite Subset

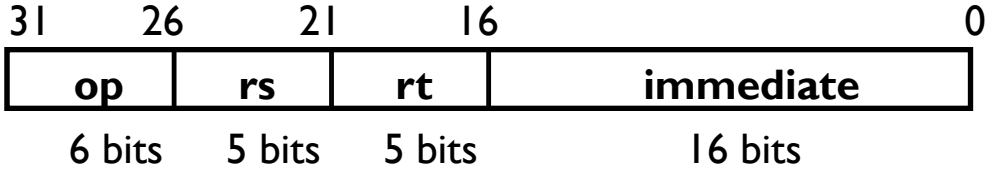
- ADD, SUB, AND, OR
  - add rd, rs, rt
  - sub rd, rs, rt
  - and rd, rs,rt
  - or rd,rs,rt



- LOAD and STORE Word
  - lw rt, rs, imm16
  - sw rt, rs, imm16



- BRANCH:
  - beq rs, rt, imm16



Register Transfer Language

- RTL gives the meaning of the instructions
- First step is to fetch the instruction from memory

op | rs | rt | rd | shamt | funct = MEM[PC]

op | rs | rt | Imm16 = MEM[PC]

inst	Register Transfers	
ADD	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
SUB	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
OR	R[rd] <– R[rs] | R[rt];	PC <– PC + 4
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
STORE	MEM[R[rs] + sign_ext(Imm16)] <– R[rt];	PC <– PC + 4
BEQ	if (R[rs] == R[rt]) then PC <– PC + sign_ext(Imm16)] || 00	
	else PC <– PC + 4	

Step 1b: Requirements of Instr. Set

- Memory
  - Instruction & data
- Registers (32 x 32)
  - Read RS
  - Read RT
  - Write RT or RD
- Extender
- Add and Sub register or extended immediate
- PC
  - Add 4 or extended immediate to PC

Step 2: Components of Datapath

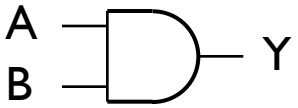
- Combinational elements
- Storage elements
  - Clocking methodology

Logic Design Basics

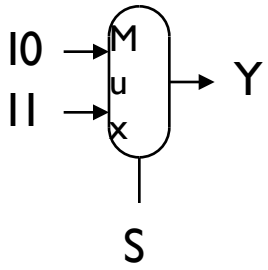
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  - Store information

Combinational Elements

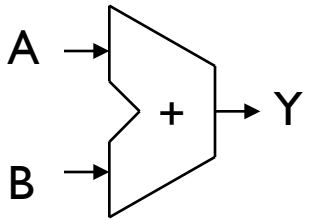
- AND-gate
  - Y = A & B



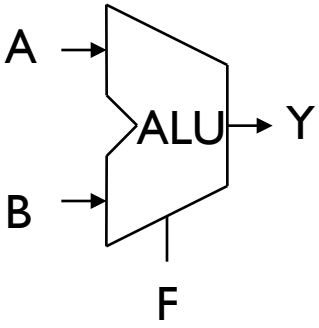
- Multiplexer
  - Y = S ? I1 : I0



- Adder
  - Y = A + B

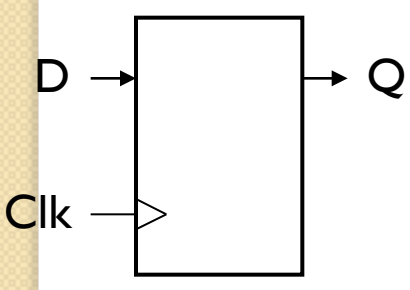


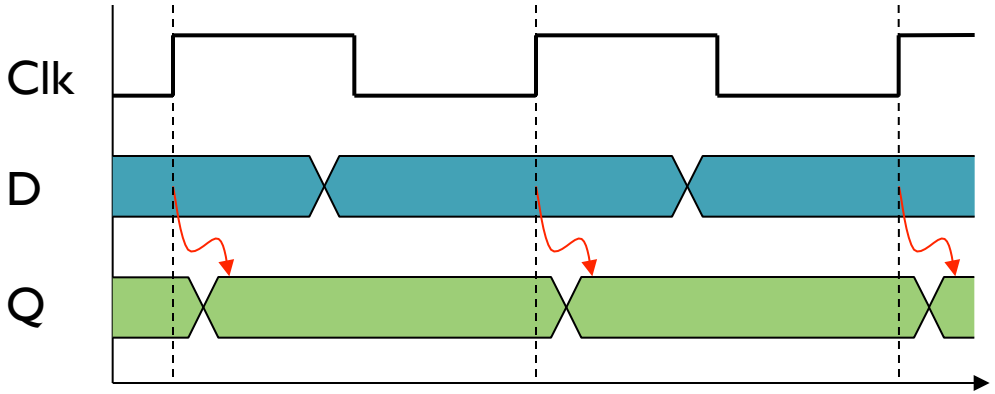
- Arithmetic/Logic Unit
  - Y = F(A, B)



Sequential Elements

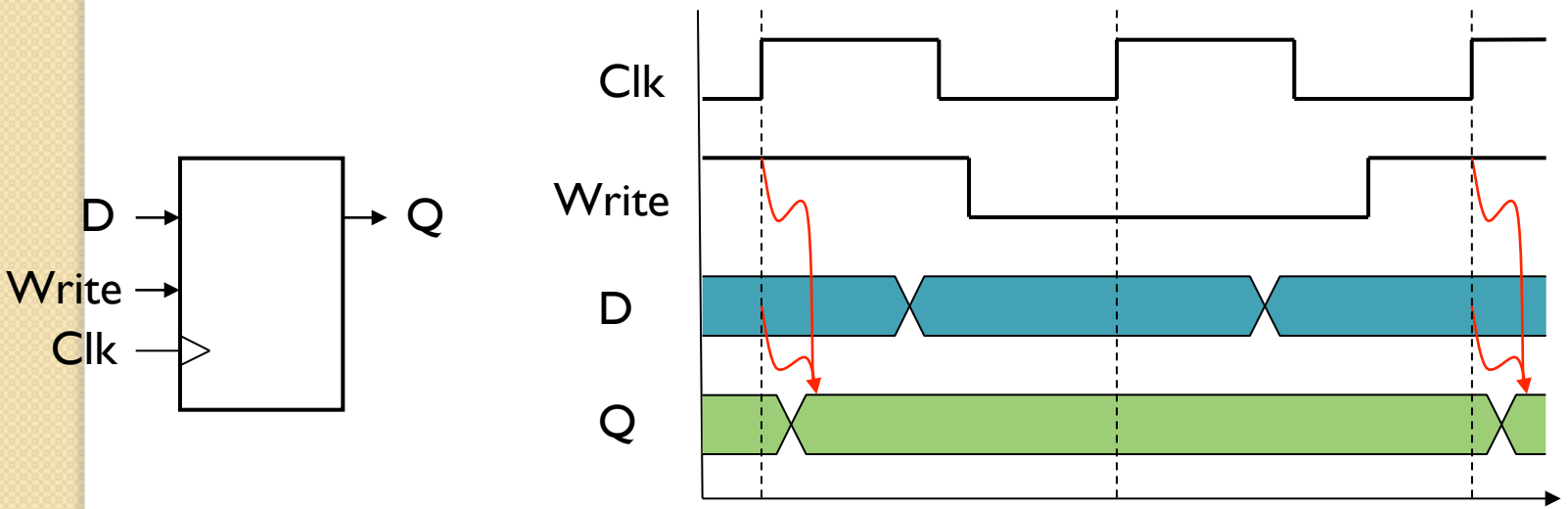
- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1





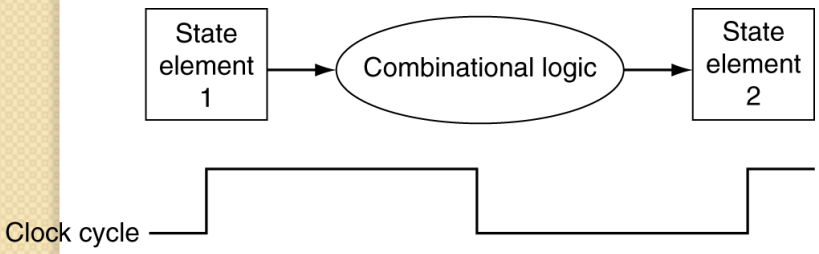
Sequential Elements

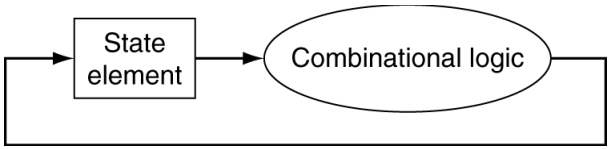
- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later



Clocking Methodology

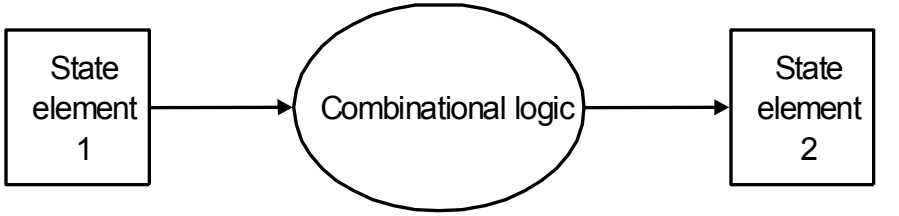
- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  - Longest delay determines clock period

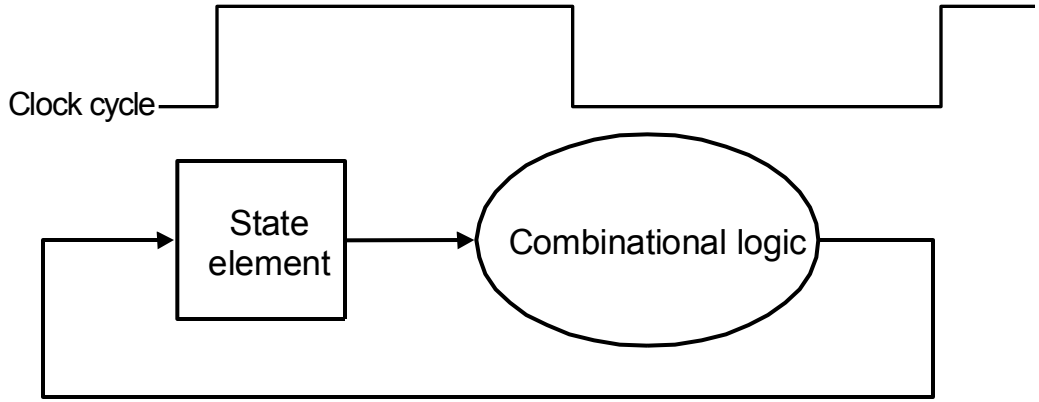




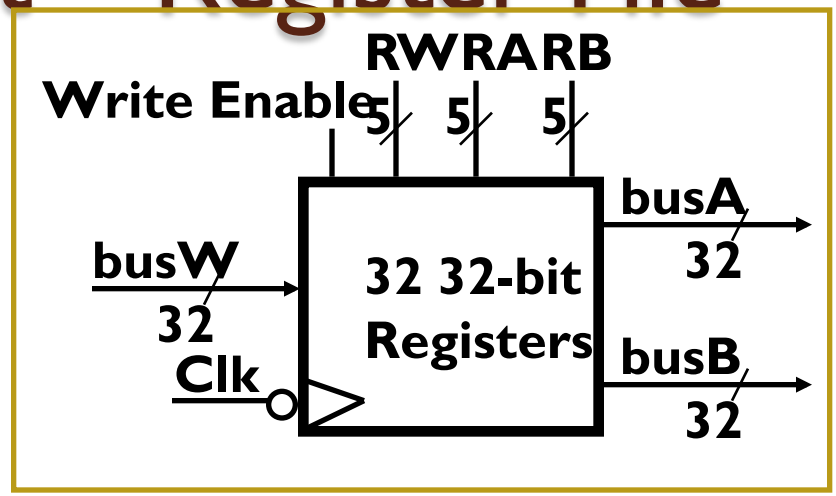
Our Implementation

- An edge triggered methodology
- Typical execution:
  - Read contents of some state elements
  - Send values through some combinational logic
  - Write results to one or more state elements





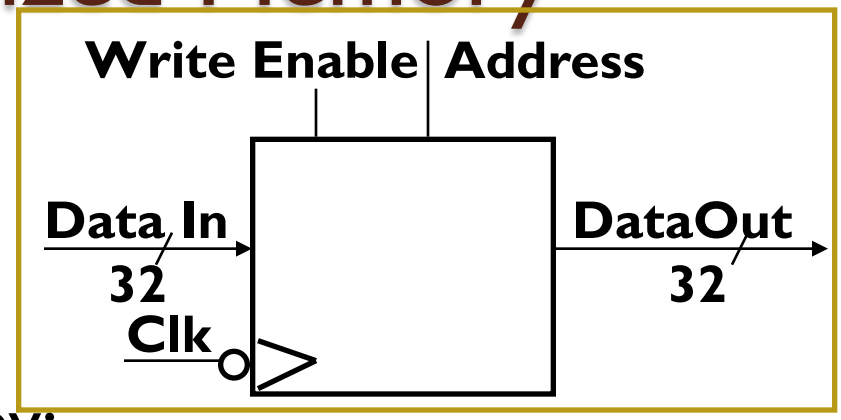
Storage Element – Register File



- Register file consists of 32 registers:
  - Two 32-bit output busses:
    - busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (CLK)
  - The CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid => busA or busB valid after "access time"

Storage Element: Idealized Memory

- Memory (idealized)
  - One input bus: Data In
  - One output bus: Data Out



- Memory word is selected by:
  - Address selects the word to put on Data Out
  - Write Enable = 1: address selects the memory word to be written via the Data In 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 => Data Out valid after "access time"

How to Design a Processor

- 1. Analyze instruction set ⇒ datapath requirements √
- 2. Select set of datapath components and establish clocking methodology √
- 3. Assemble datapath meeting the requirements
- 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer
- 5. Assemble the control logic

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
  - Fetch Instructions
  - Read operands and execute instructions.

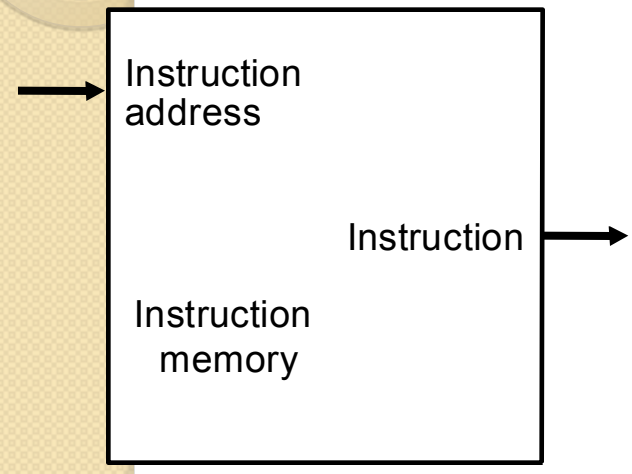
3a: Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump: PC <- "something else"
    - We don't know if instruction is a Branch/Jump or one of the other instructions until we have fetched and interpreted the instruction from memory
    - So all instructions initially increment the PC

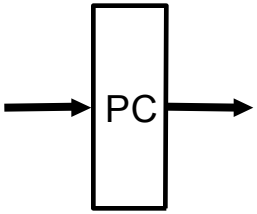
3a: Instruction Fetch Unit

- The common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump: PC <- "something else"
    - We don't know if instruction is a Branch/Jump or one of the other instructions until we have fetched and interpreted the instruction from memory
    - So all instructions initially increment the PC

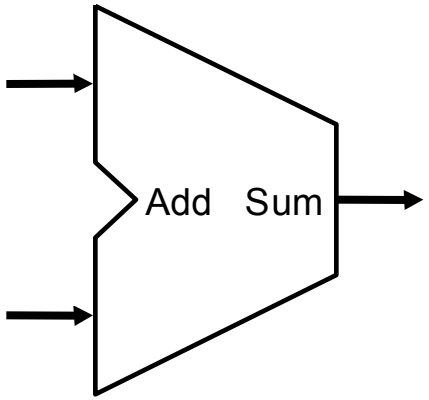
Components to Assemble



a. Instruction memory

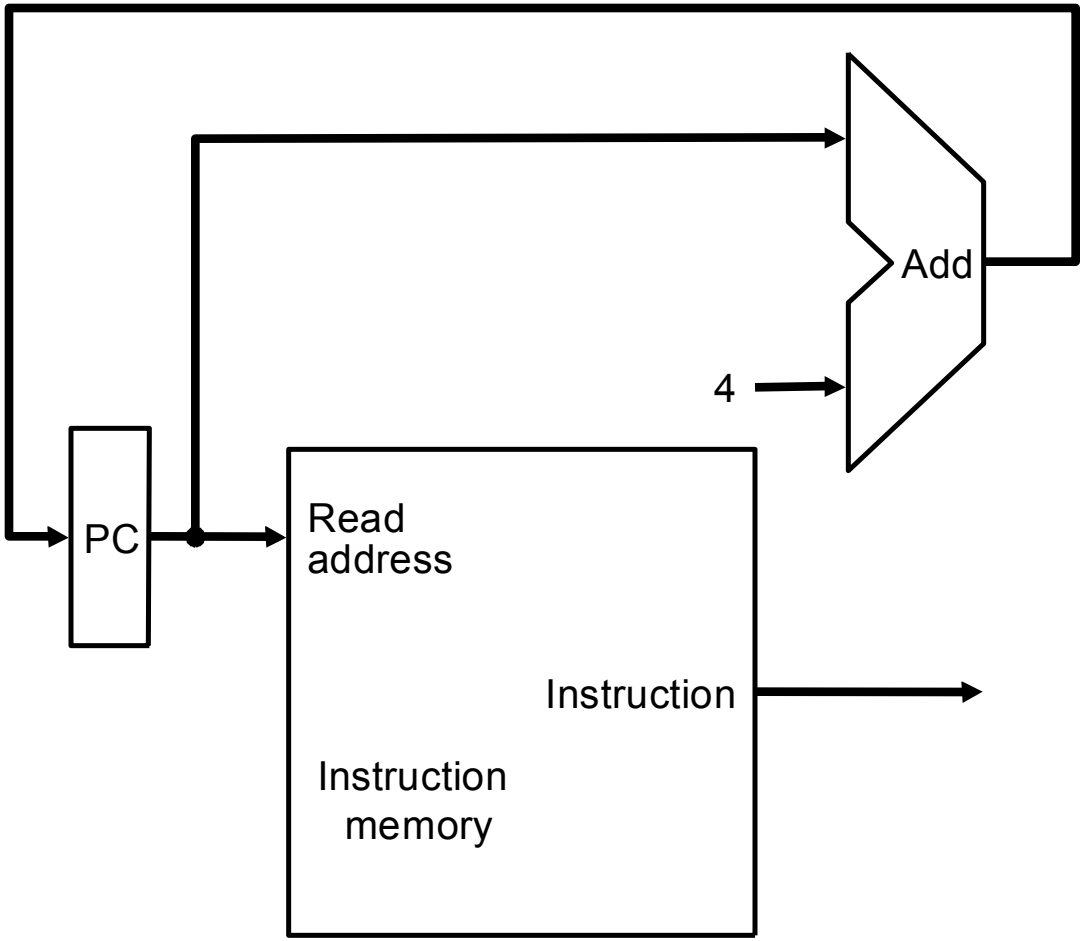


b. Program counter



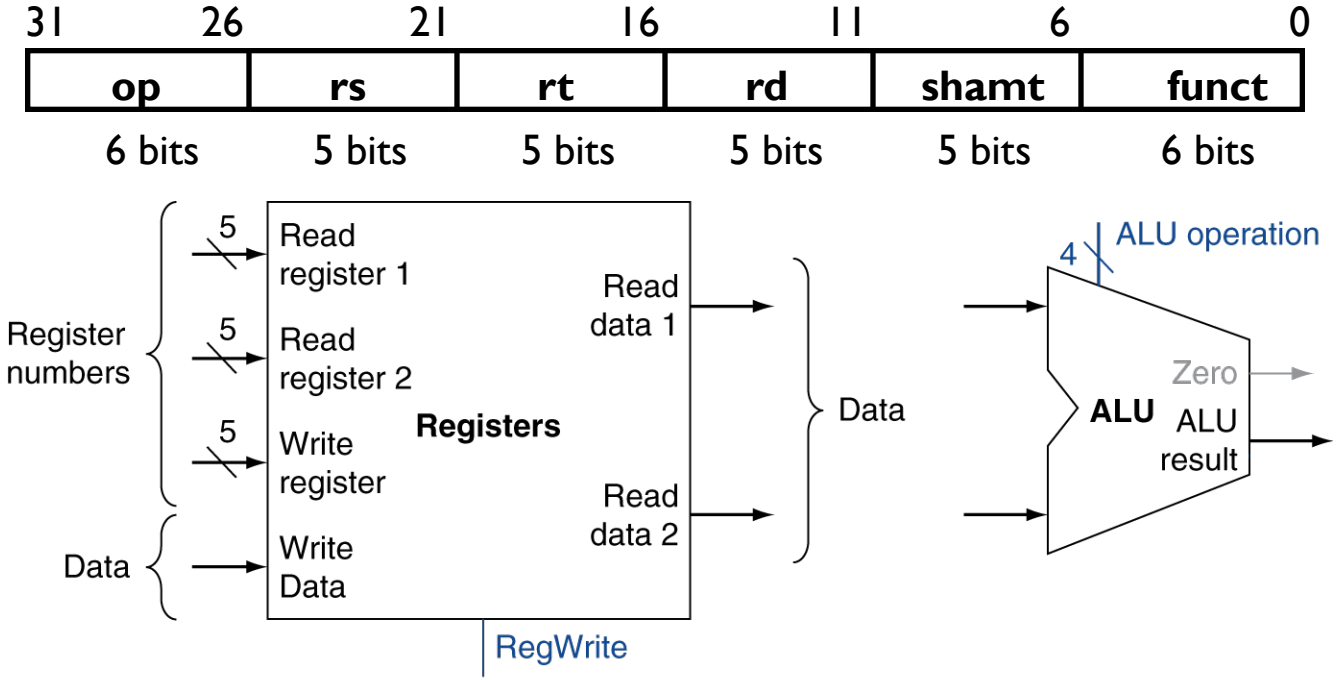
c. Adder

Datapath for Instruction Fetch

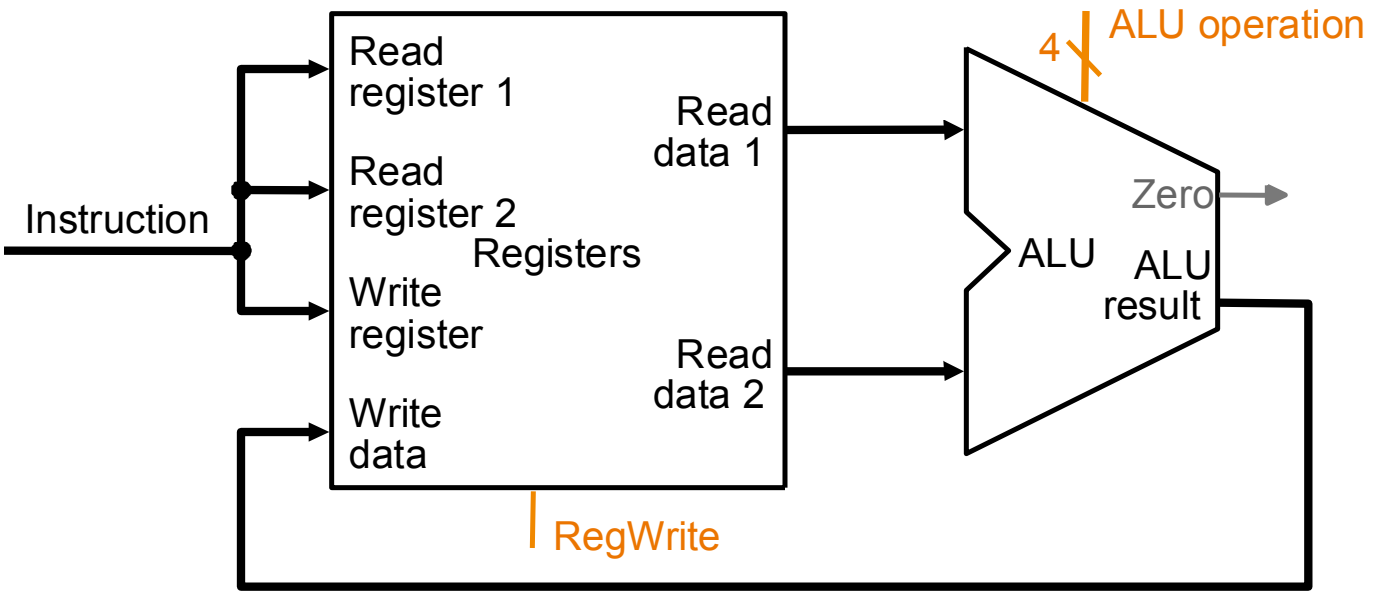


Step 3b: R-format Instructions

- Example instructions: add, sub, and, or, slt
- R[rd] <- R[rs] op R[rt]
  - Read register 1, Read register 2, and Write register come from instruction's rs, rt, and rd fields
  - ALU control and RegWrite: control logic after decoding the instruction

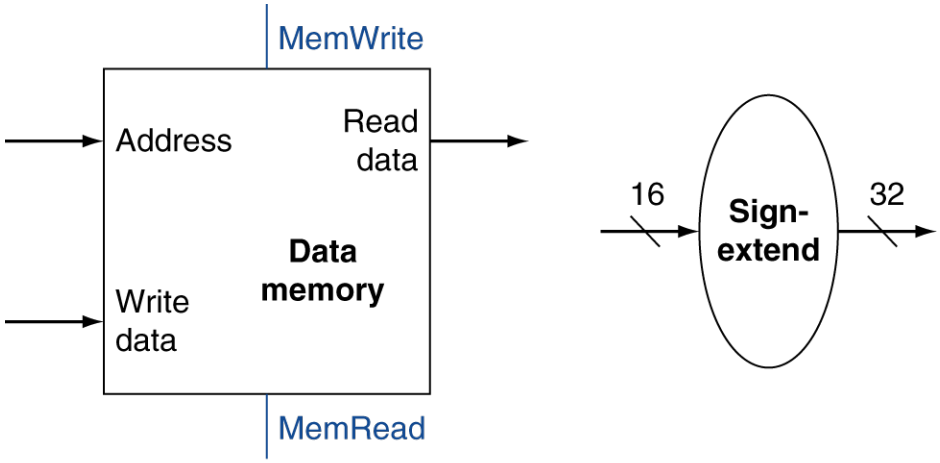


Datapath for R-format Instructions



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

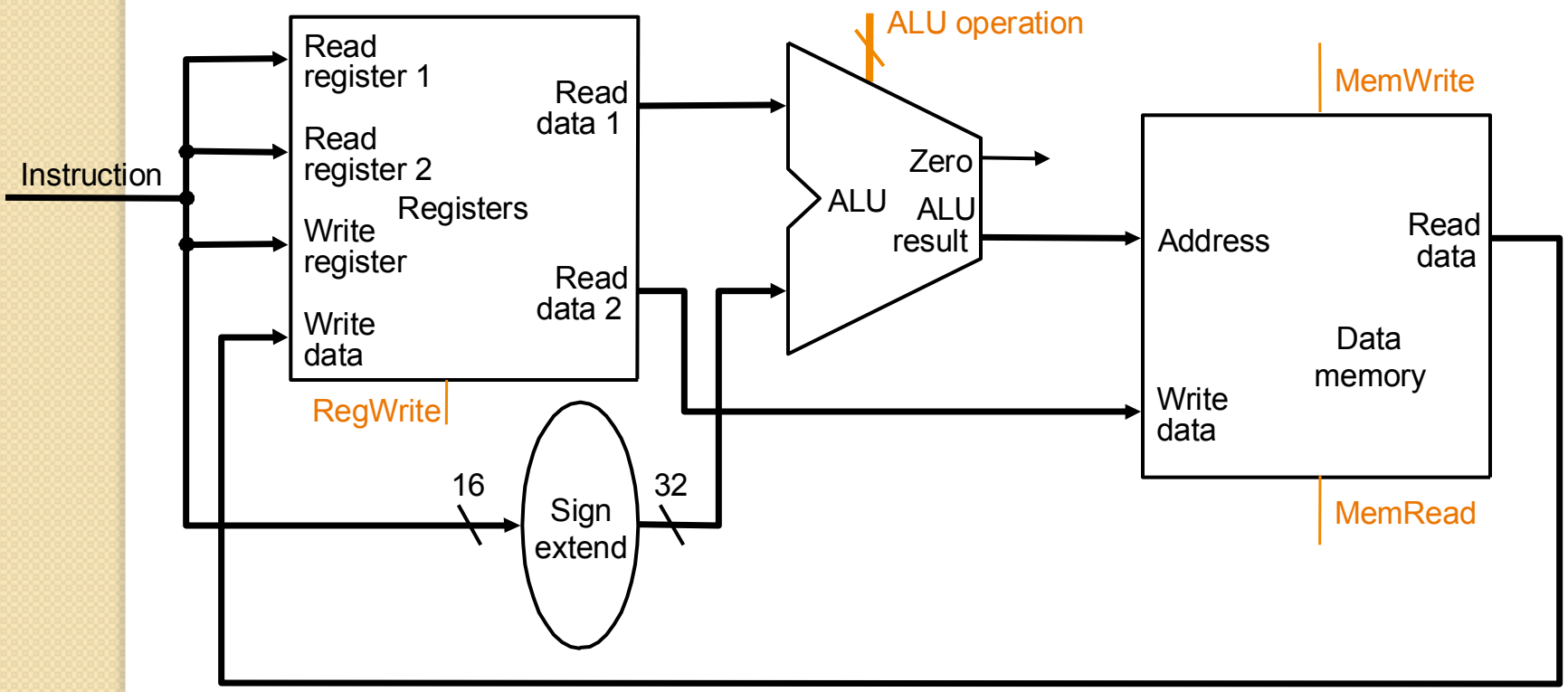


a. Data memory unit

b. Sign extension unit

Datapath for LW & SW

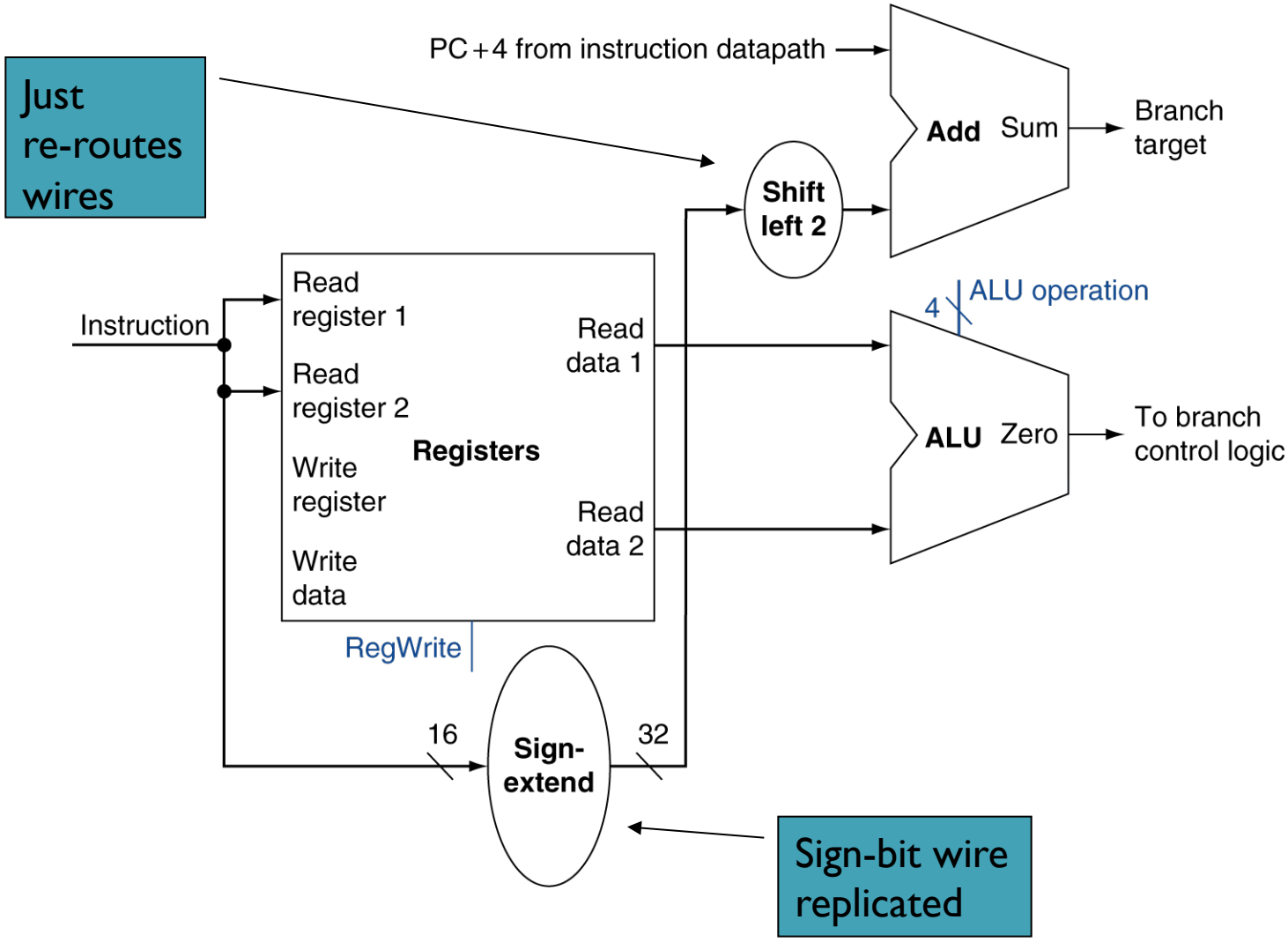
- R[rt] <- Mem[R[rs] + SignExt[imm16]] Example: lw rt, rs, imm16
- Mem[R[rs] + SignExt[imm16] <- R[rt]] Example: sw rt, rs, imm16



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

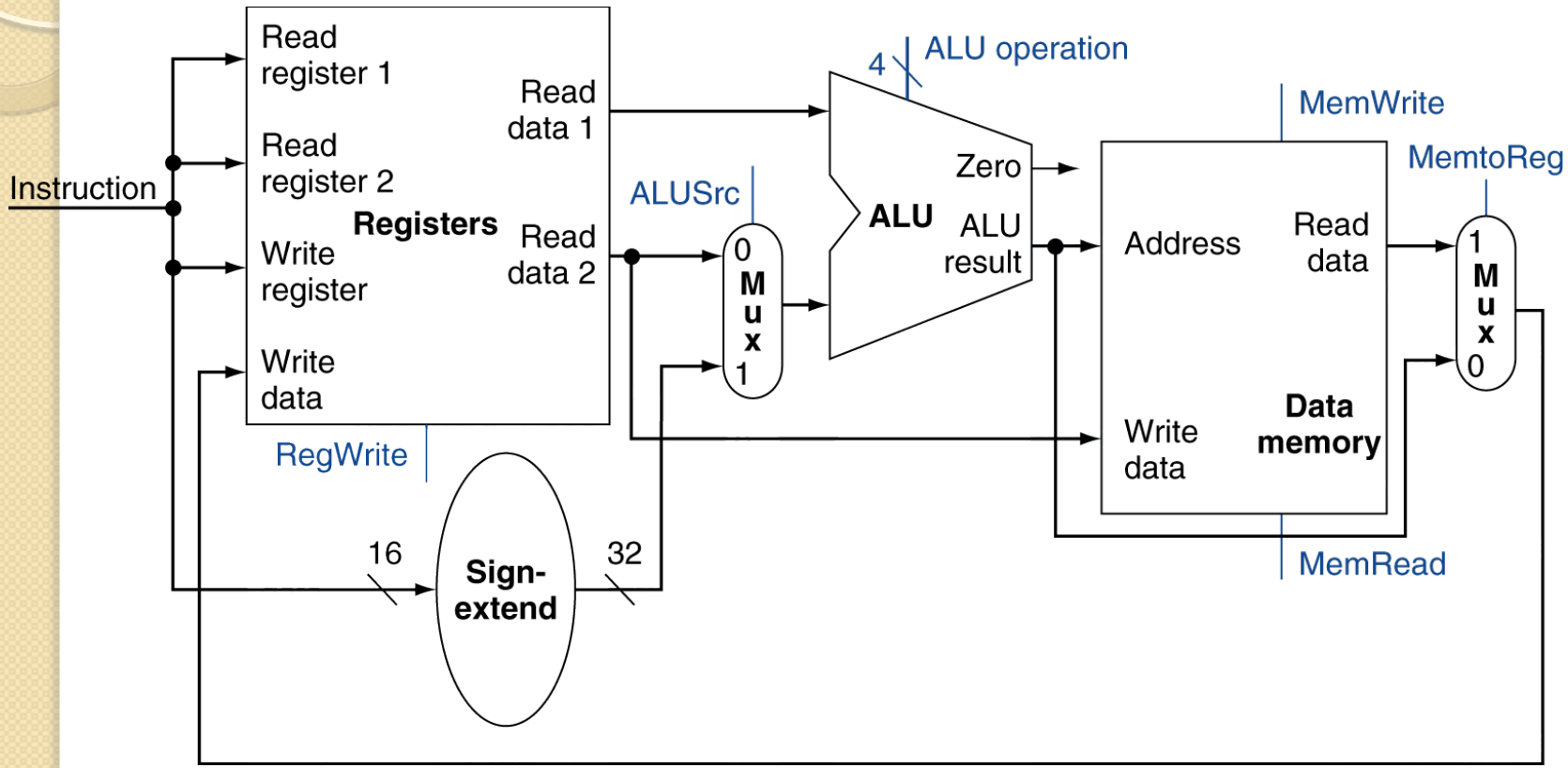
Branch Instructions



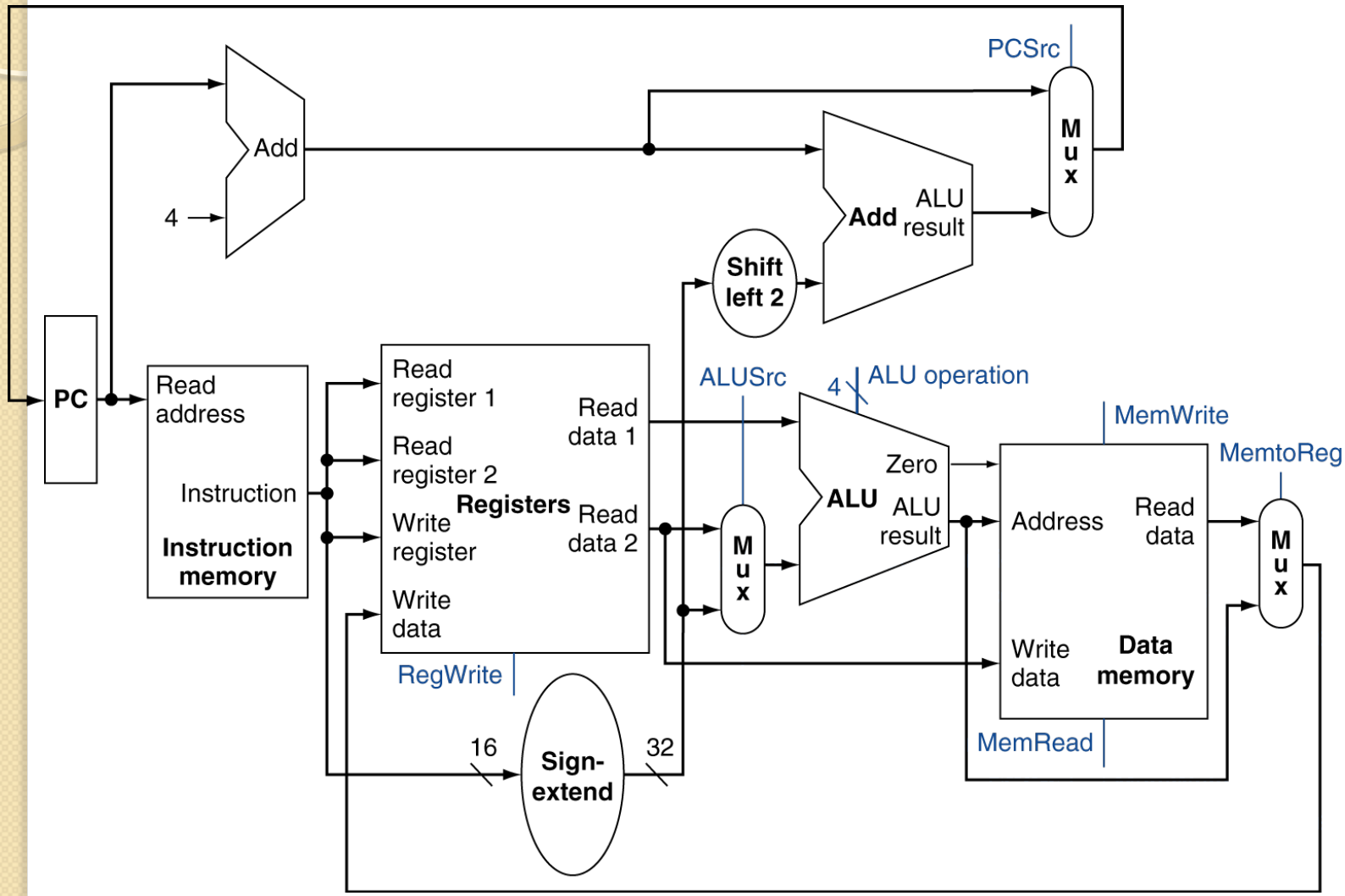
Composing the Elements

- 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

R-Type/Load/Store Datapath



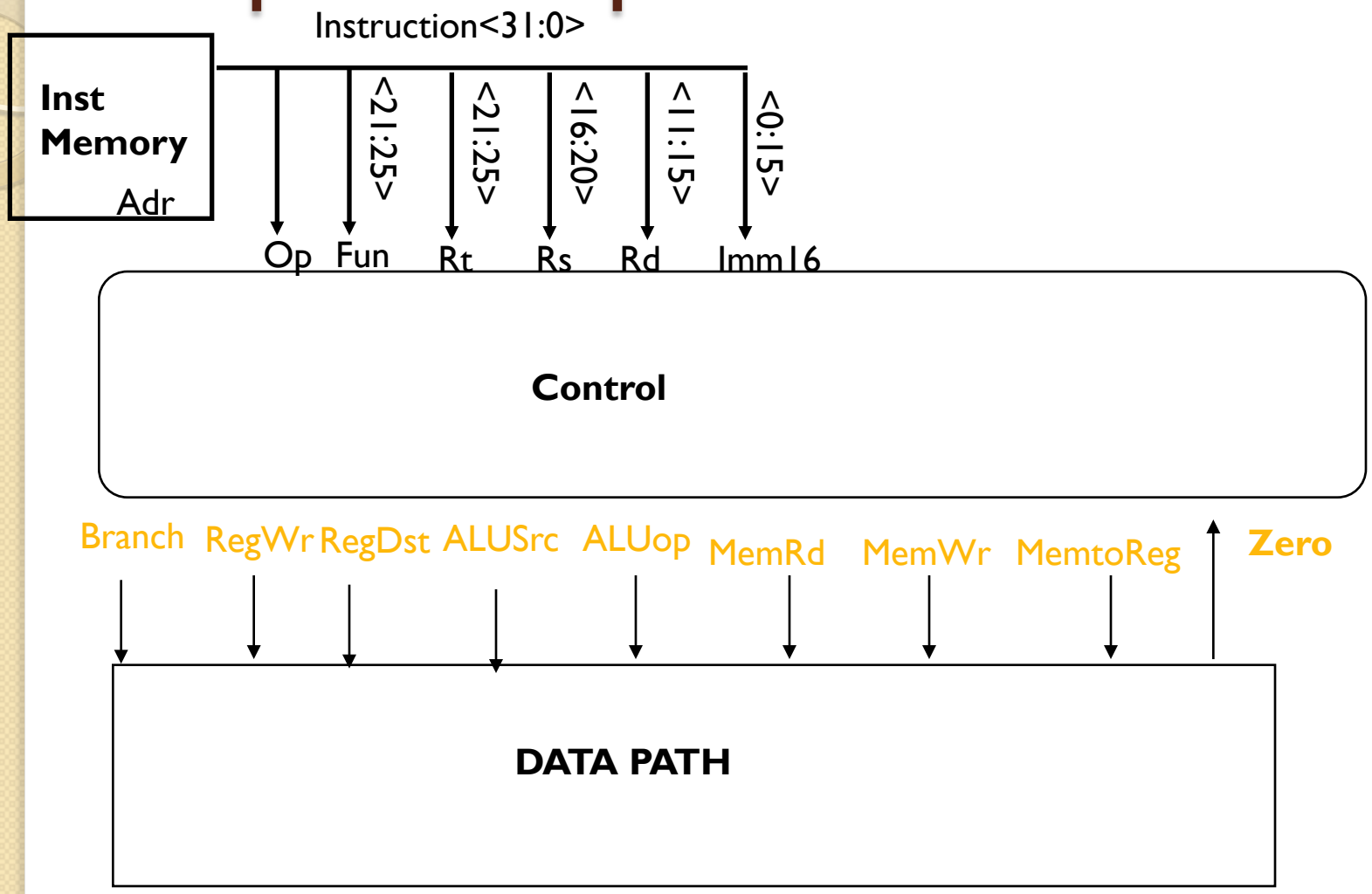
Full Datapath



How to Design a Processor

- 1. Analyze instruction set ⇒ datapath requirements √
- 2. Select set of datapath components and establish clocking methodology √
- 3. Assemble datapath meeting the requirements √
- 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer
- 5. Assemble the control logic

Step 4: Datapath: RTL -> Control



Control

- Selecting the operations to perform (ALU, read/write, etc.)
  - Design the ALU Control Unit
- Controlling the flow of data (multiplexor inputs)
  - Design the Main Control Unit
- Information comes from the 32 bits of the instruction

ALU Control

- ALU 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

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

Control Implementation

- Must describe hardware to compute 4-bit ALU control input
  - Given instruction type
    - 00 = lw, sw
    - 01 = beq
    - 10 = arithmetic

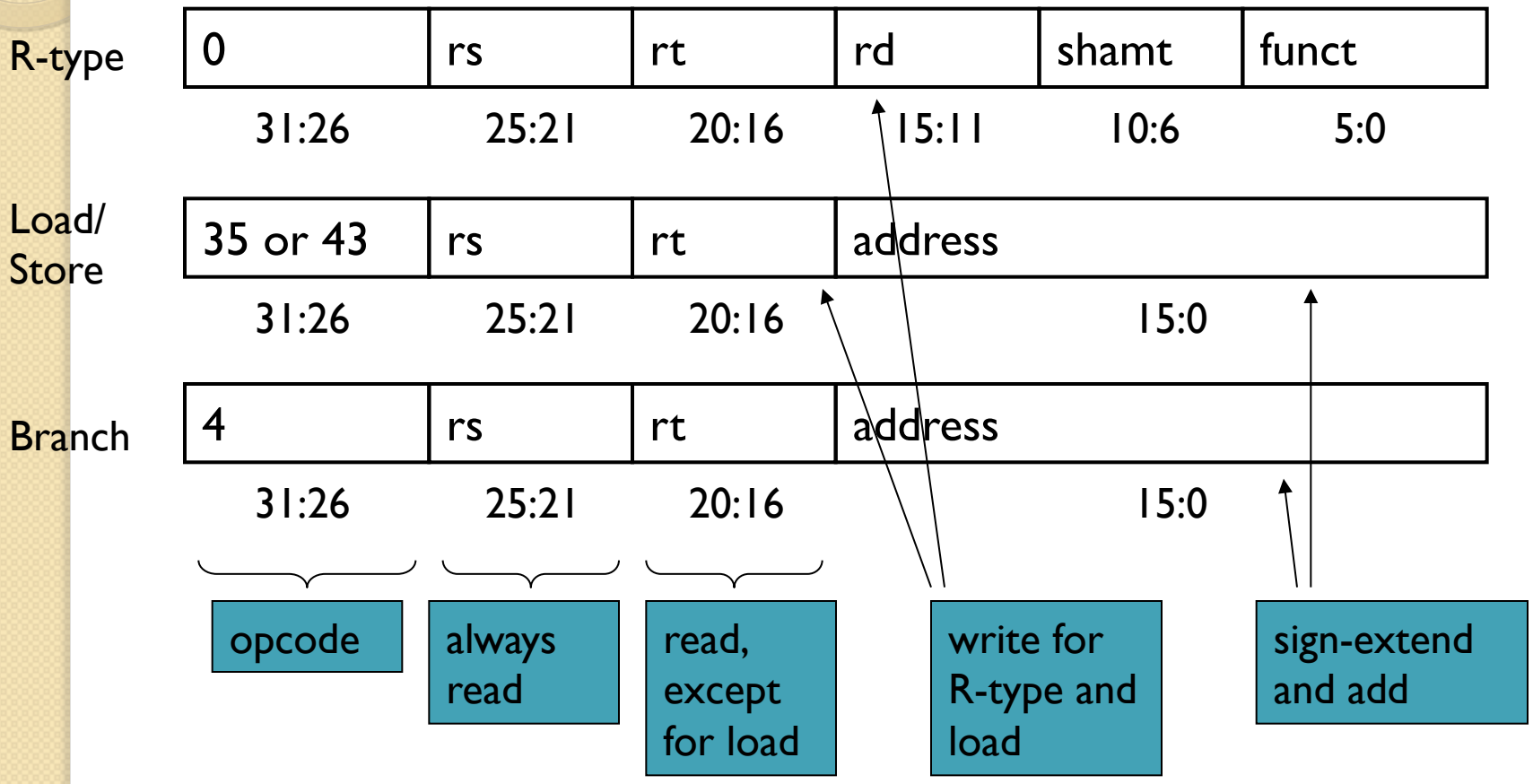
    → **ALUOp computed from instruction type**
  - Function code for arithmetic

- Describe it using a truth table (can turn into gates):

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

The Main Control Unit

- Control signals derived from instruction



Design the Main Control Unit

- Seven control signals
  - RegDst
  - RegWrite
  - ALUSrc
  - PCSrc
  - MemRead
  - MemWrite
  - MemtoReg

Control Signals

1. RegDst = 0 => Register destination number for the Write register comes from the rt field (bits 20-16)

   RegDst = 1 => Register destination number for the Write register comes from the rd field (bits 15-11)
2. RegWrite = 1 => The register on the Write register input is written with the data on the Write data input (at the next clock edge)
3. ALUSrc = 0 => The second ALU operand comes from Read data 2

   ALUSrc = 1 => The second ALU operand comes from the sign-extension unit
4. PCSrc = 0 => The PC is replaced with PC+4

   PCSrc = 1 => The PC is replaced with the branch target address
5. MemtoReg = 0 => The value fed to the register write data input comes from the ALU

   MemtoReg = 1 => The value fed to the register write data input comes from the data memory
6. MemRead = 1 => Read data memory
7. MemWrite = 1 => Write data memory

R-format Instructions

RegDst = 1
RegWrite = 1
ALUSrc = 0
Branch = 0
MemtoReg = 0
MemRead = 0
MemWrite = 0
ALUOp = 10

Memory Access Instructions

Load word

RegDst = 0

RegWrite = 1

ALUSrc = 1

Branch = 0

MemtoReg = 1

MemRead = 1

MemWrite = 0

ALUOp = 00

Store Word

RegDst = X

RegWrite = 0

ALUSrc = 1

Branch = 0

MemtoReg = X

MemRead = 0

MemWrite = 1

ALUOp = 00

Branch on Equal

RegDst = X

RegWrite = 0

ALUSrc = 0

Branch = 1

MemtoReg = X

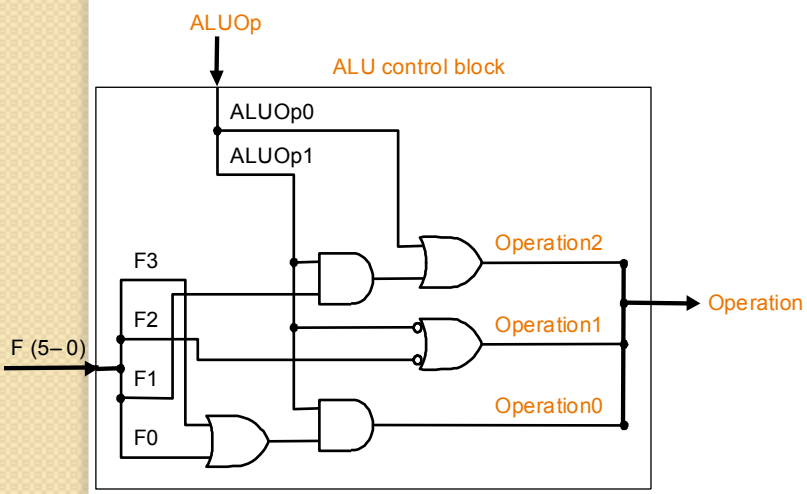
MemRead = 0

MemWrite = 0

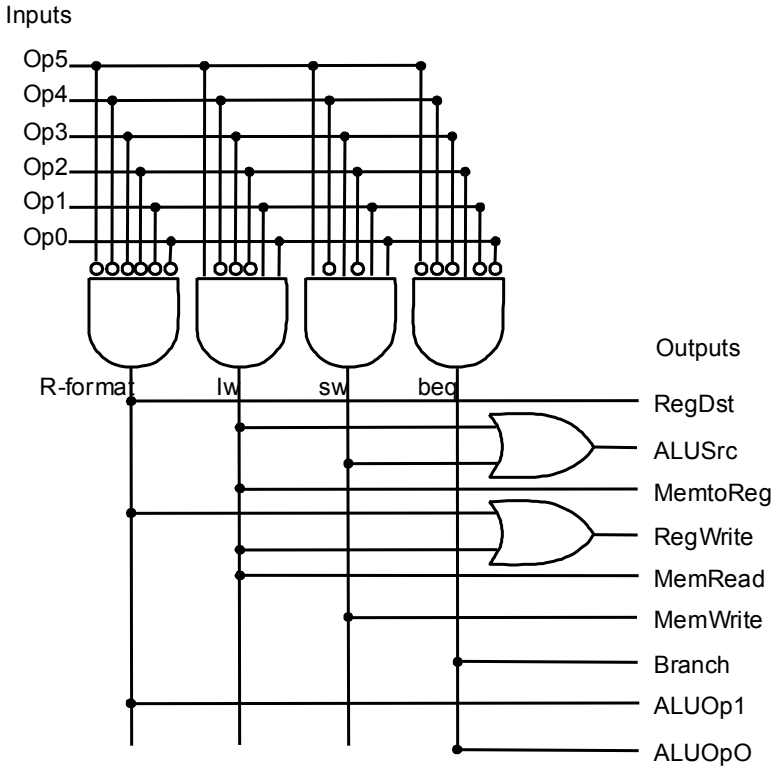
ALUOp = 01

Step 5: Implementing Control

- Simple combinational logic (truth tables)

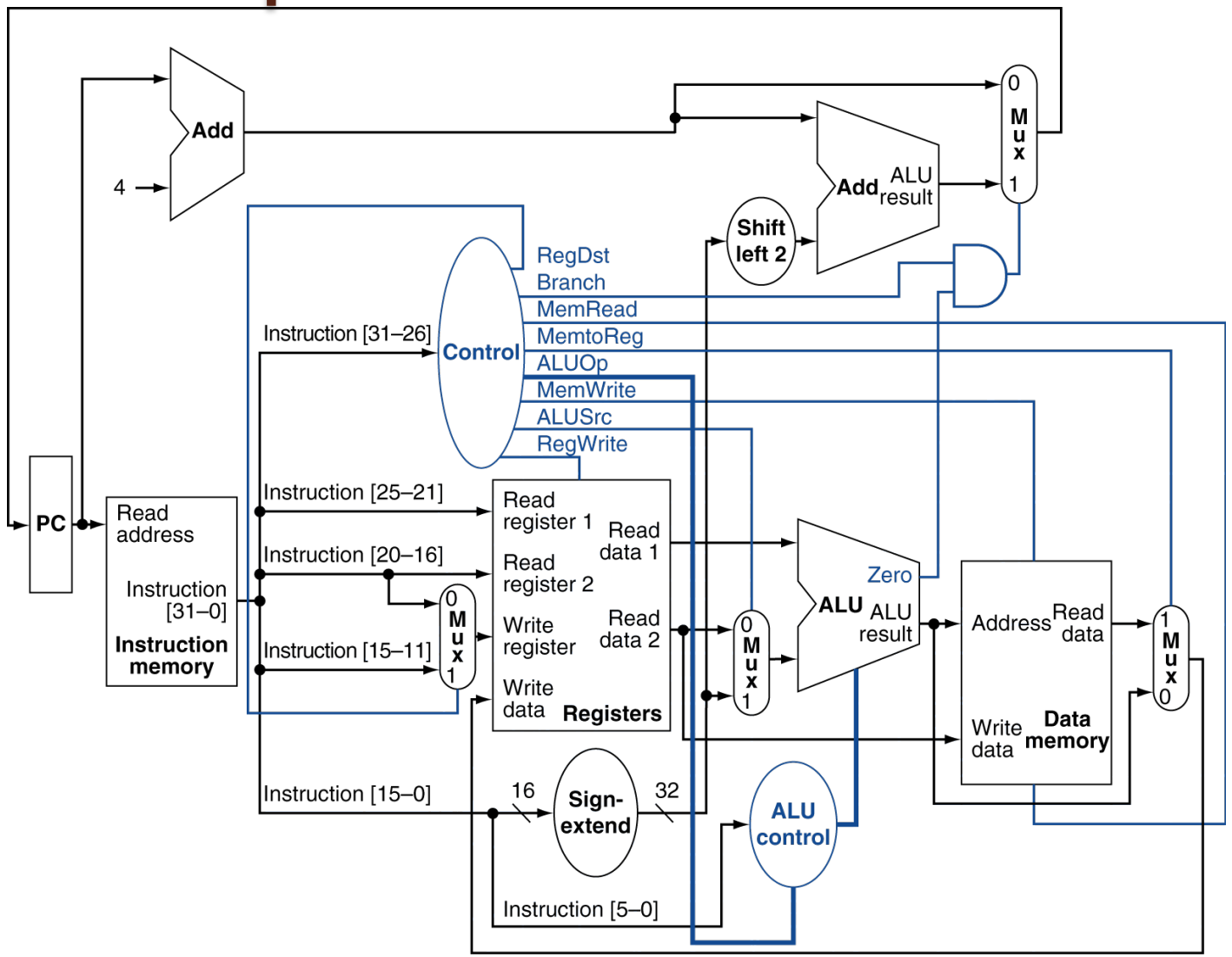


ALU Control Unit

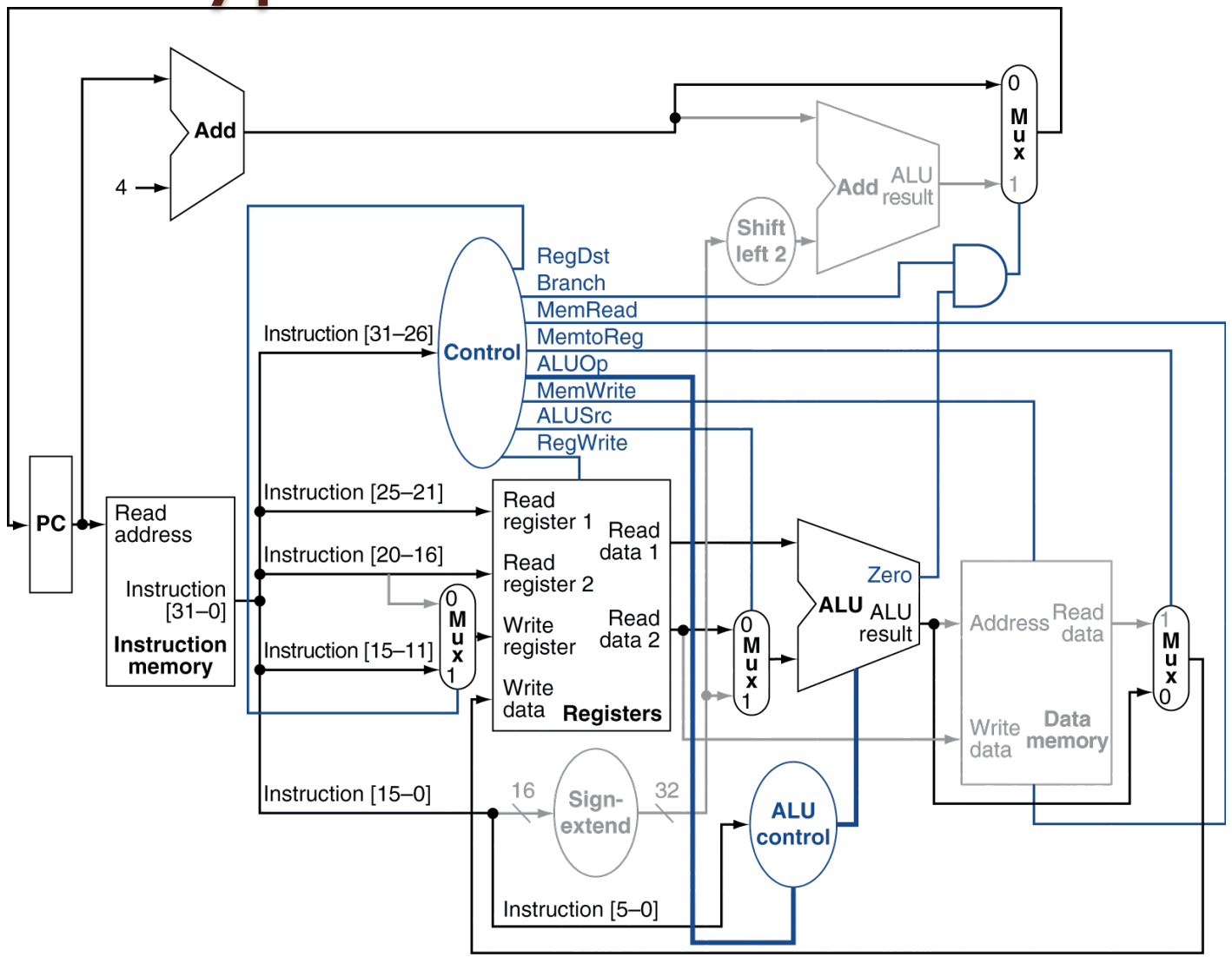


Main Control Unit

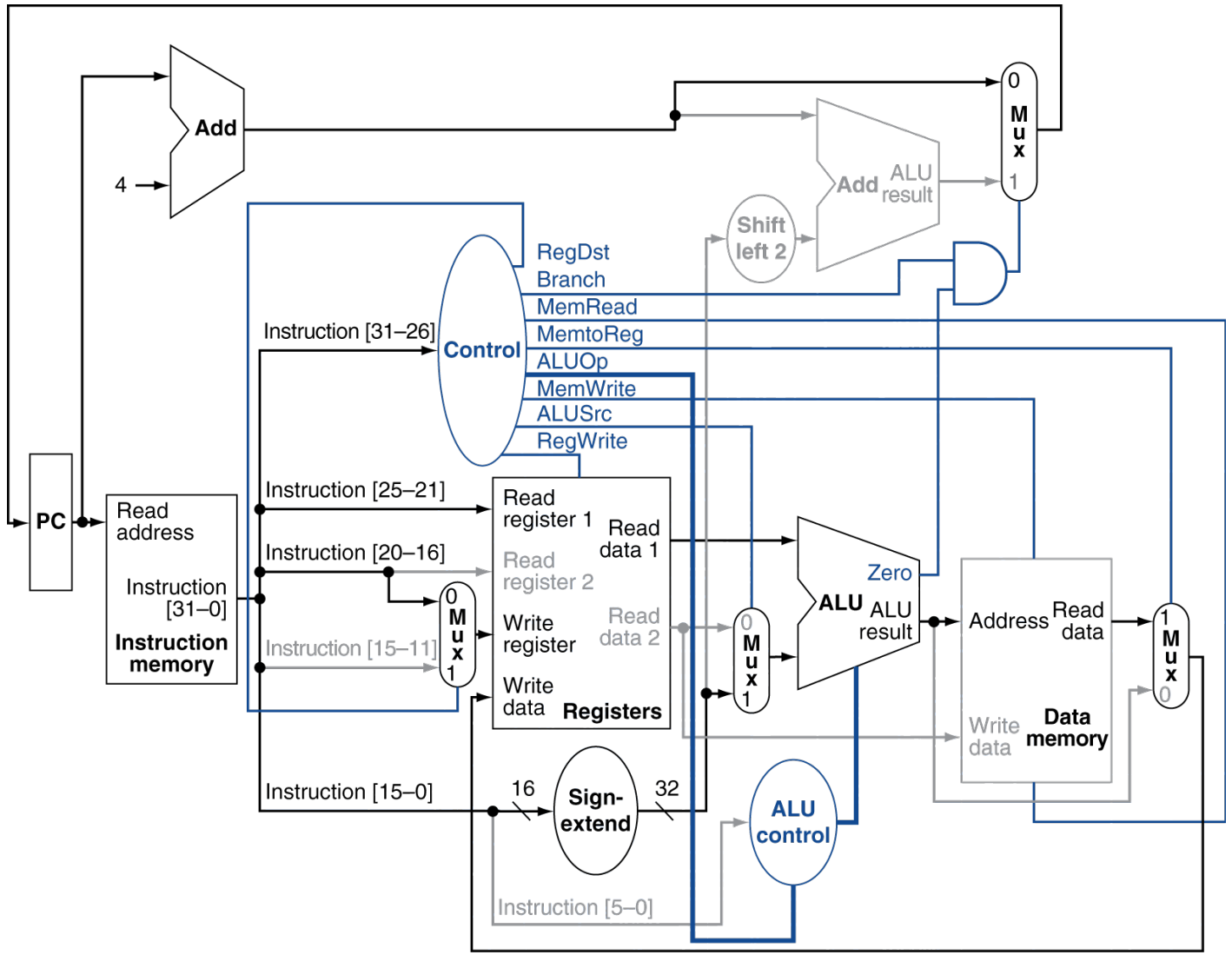
Datapath With Control



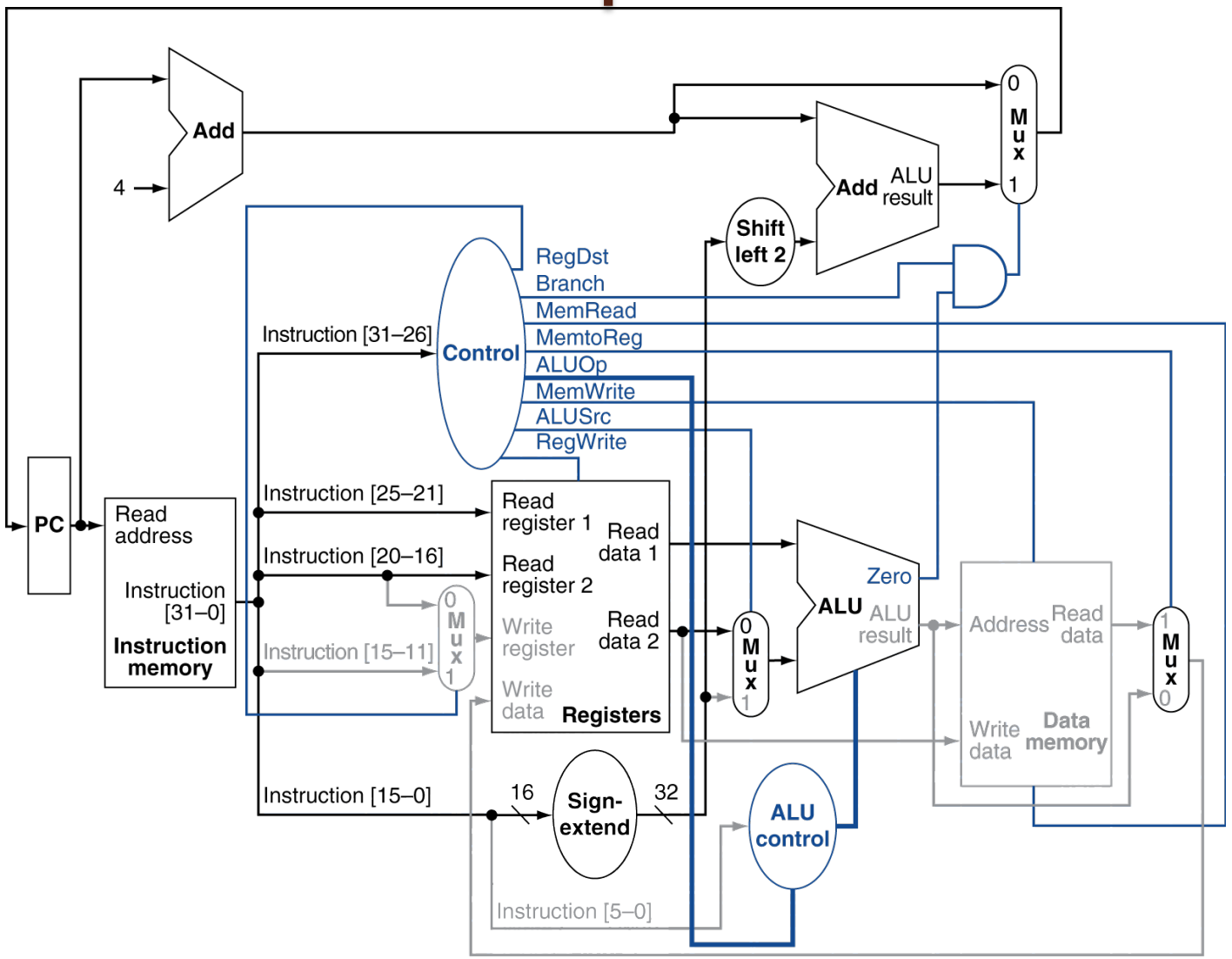
R-Type Instruction



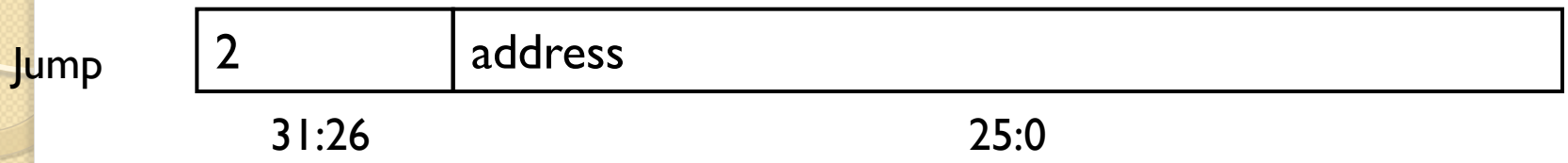
Load Instruction



Branch-on-Equal Instruction

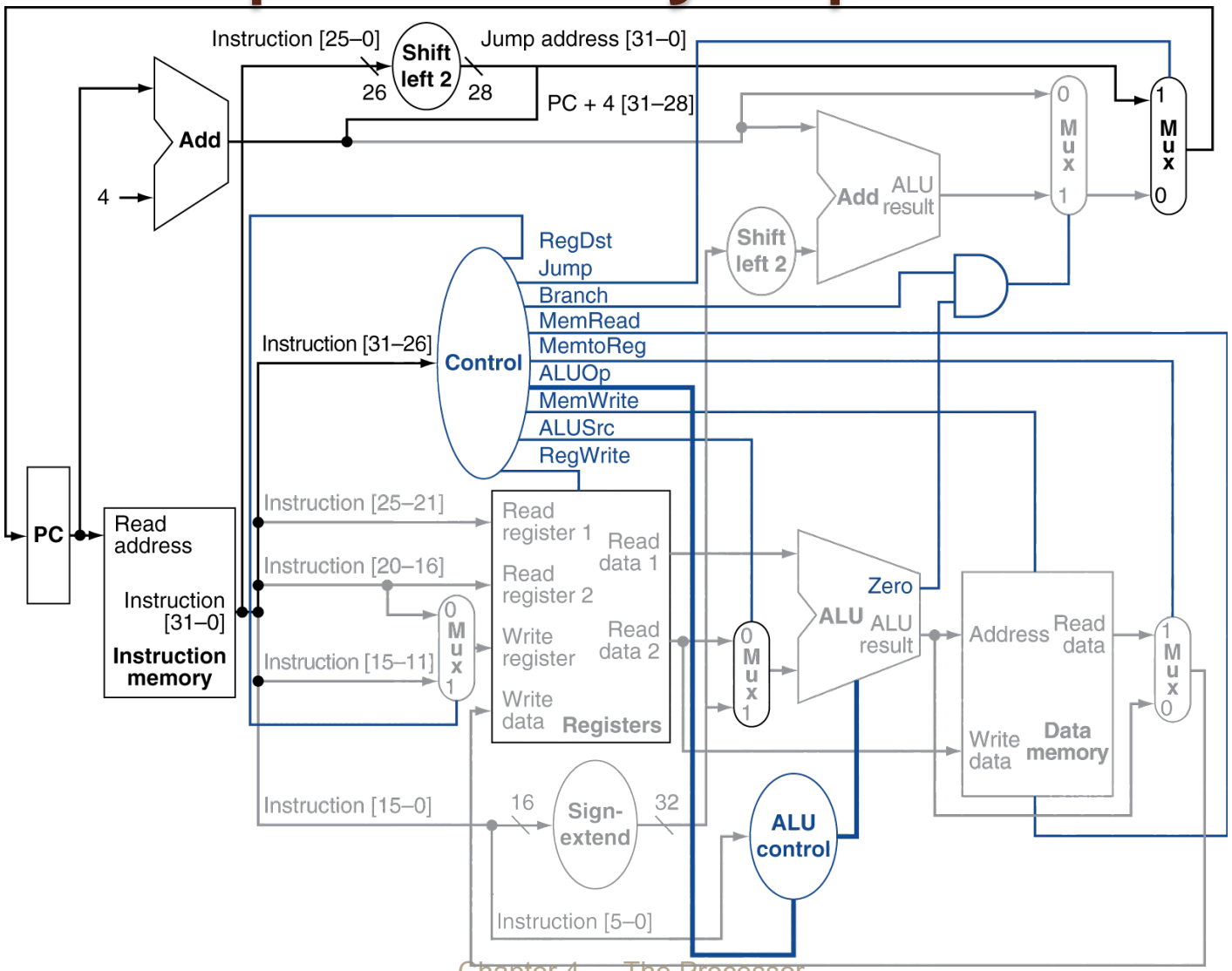


Implementing Jumps



- 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

Datapath With Jumps Added



Performance Issues

- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory → register file → ALU → data memory → register file
- Not feasible to vary period for different instructions
- Violates design principle
  - Making the common case fast
- We will improve performance by pipelining

Summary

- 5 steps to design a processor
  - 1. Analyze instruction set => datapath requirements
  - 2. Select set of datapath components & establish clock methodology
  - 3. Assemble datapath meeting the requirements
  - 4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer
  - 5. Assemble the control logic
- MIPS makes it easier
  - Instructions same size
  - Source registers always in same place
  - Immediates same size, location
  - Operations always on registers/immediates
- Single cycle datapath => CPI=1, Clock Cycle Time => long