University of Chinese Academy of Sciences

B0911006Y-01 2024-2025学年春季学期

计算机组成原理

Principles of Computer Organization

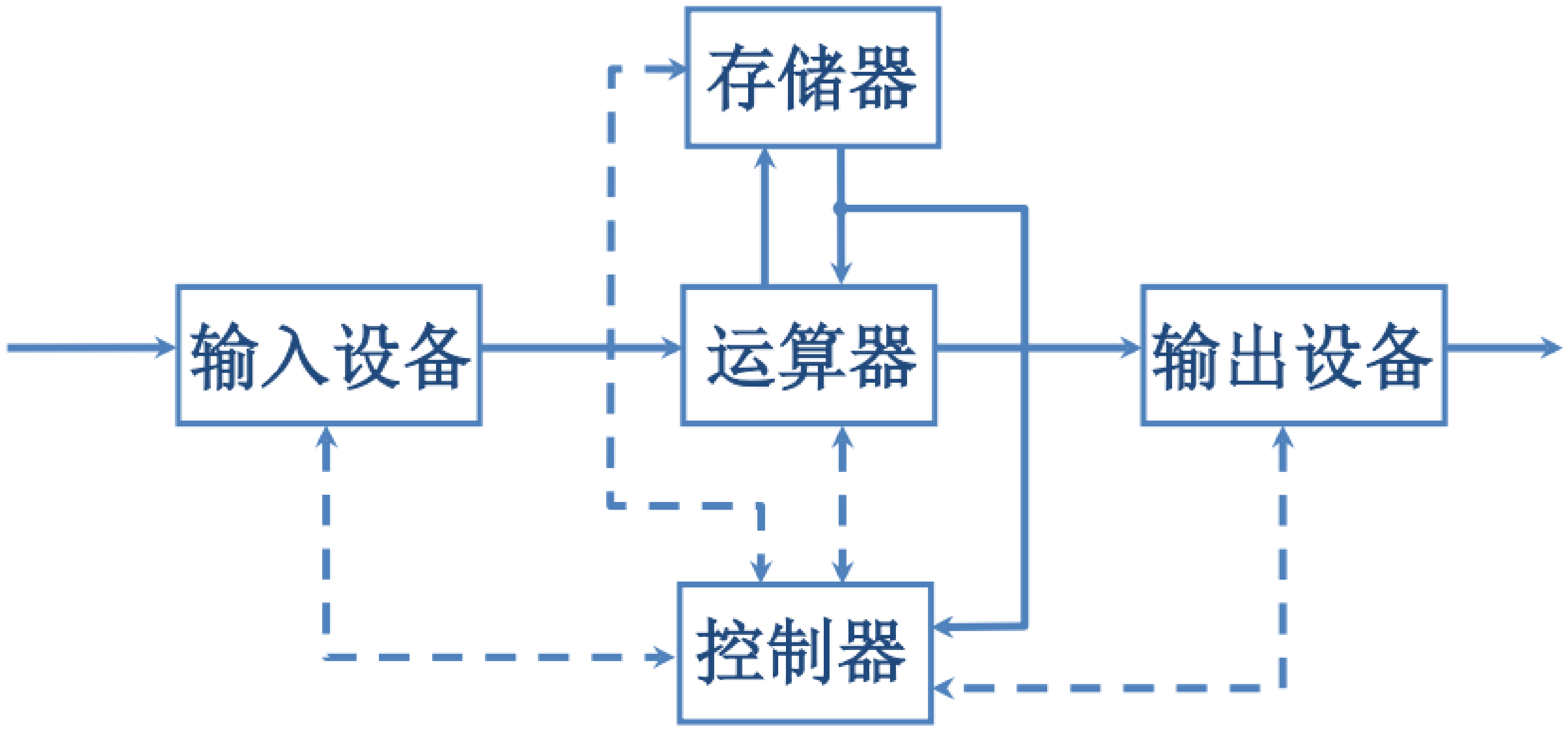
单周期处理器 I

基本组件，如何设计、组装数据通路

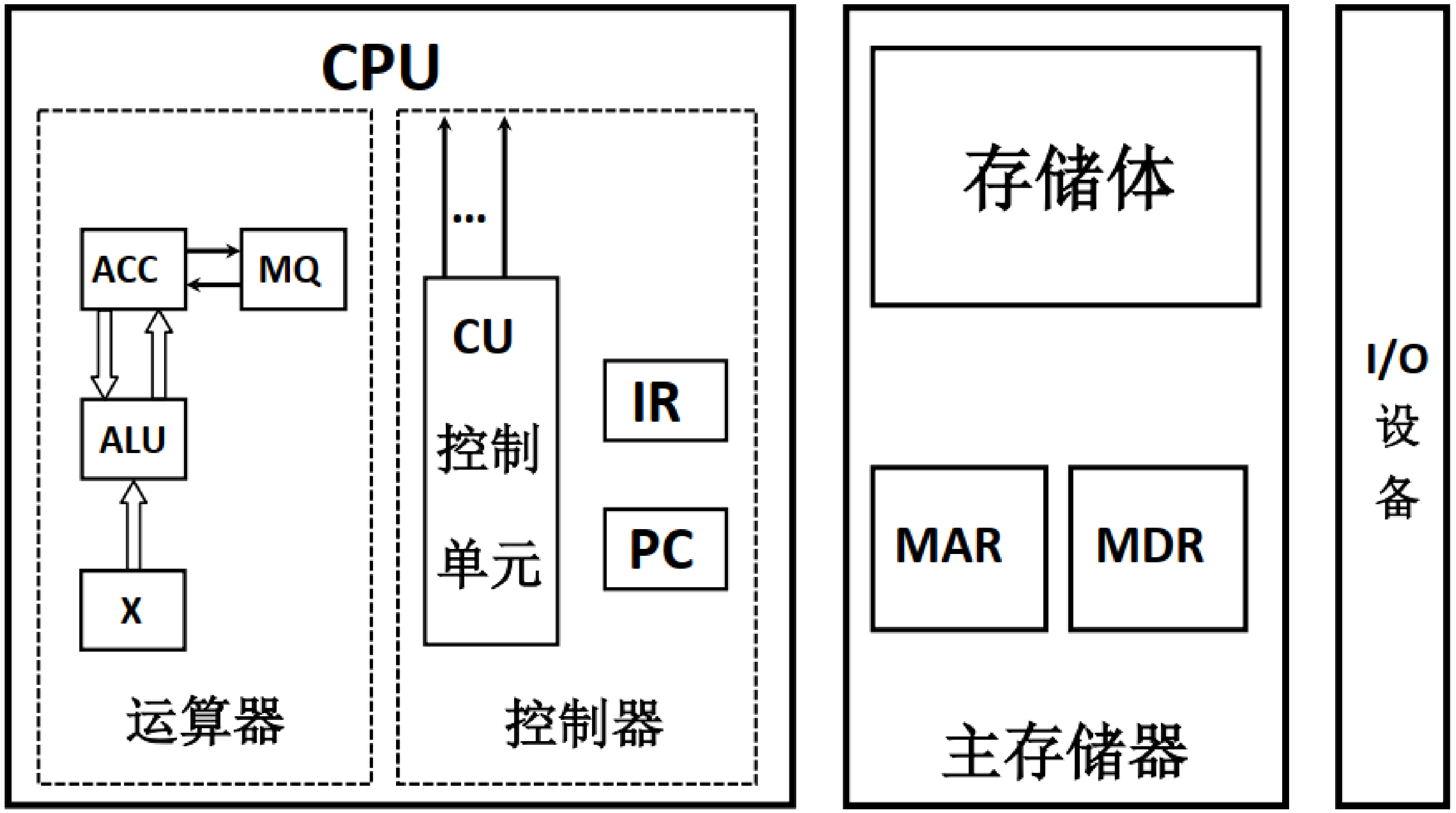
主讲教师：石 侃
shikan@ict.ac.cn

2025年4月9日

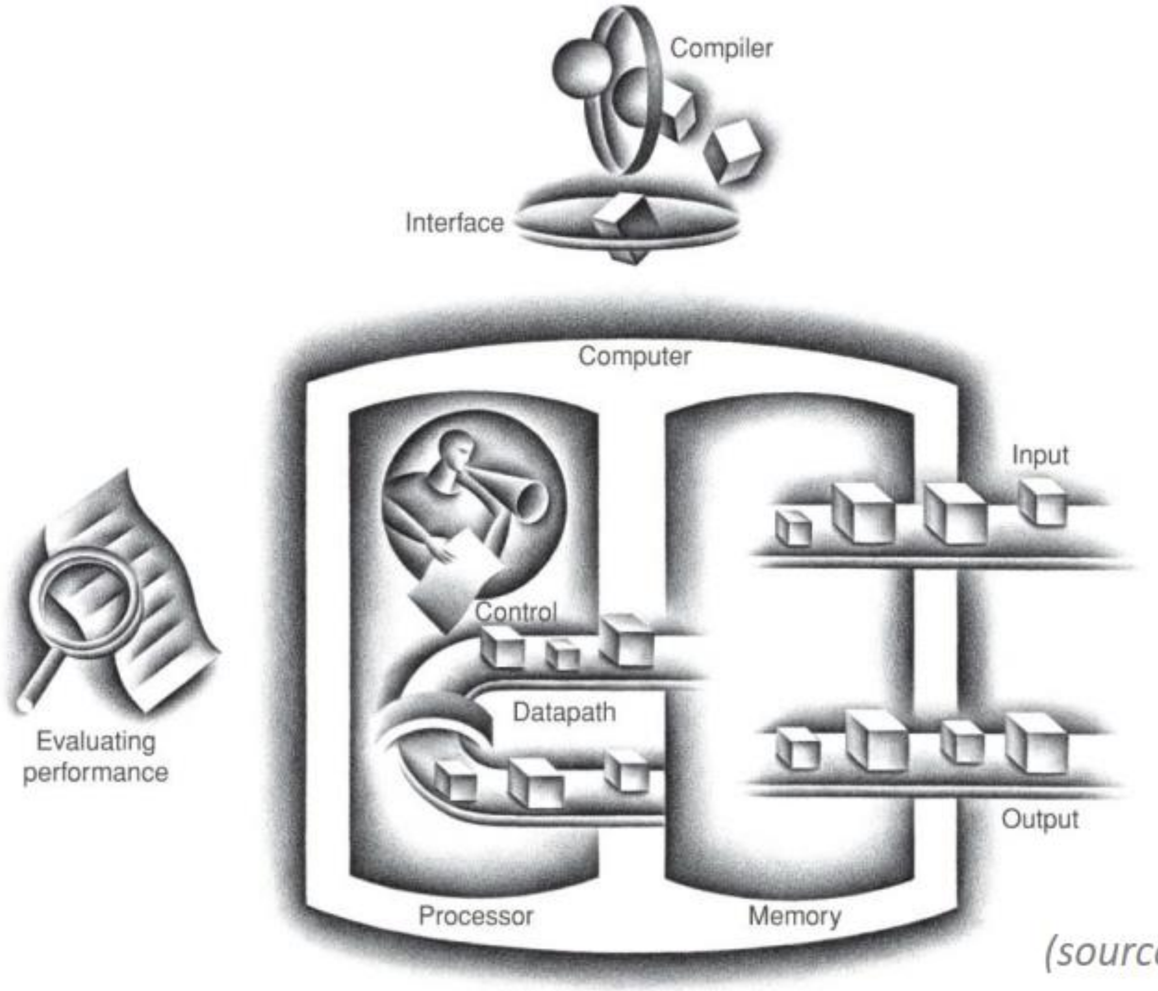
回顾：冯诺依曼计算机五大部件



回顾：冯诺依曼计算机五大部件



回顾：冯诺依曼计算机五大部件



(source: P&H-COD)

The Processor

- **Processor (CPU)**: Implements the instructions of the Instruction Set Architecture (ISA)

The Processor

- **Processor (CPU)**: Implements the instructions of the Instruction Set Architecture (ISA)
    - *Datapath*: part of the processor that contains the hardware necessary to perform operations required by the processor (“the brawn”)
        - 指令执行过程中，数据所经过的路径，包括路径中的部件
        - 是指令执行的部件
        - 组合元件和存储元件通过总线或分散方式连接而成的进行数据存储、处理和传送的路径。

The Processor

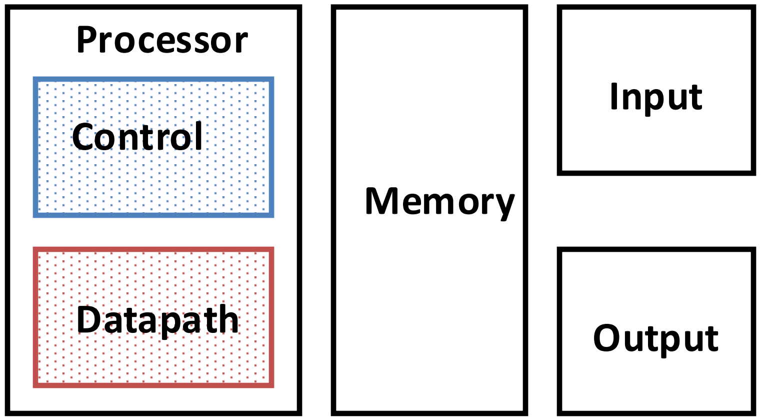
- **Processor (CPU)**: Implements the instructions of the Instruction Set Architecture (ISA)
  - *Datapath*: part of the processor that contains the hardware necessary to perform operations required by the processor (“the brawn”)
    - 指令执行过程中，数据所经过的路径，包括路径中的部件
    - 是指令执行的部件
    - 组合元件和存储元件通过总线或分散方式连接而成的进行数据存储、处理和传送的路径。
  - *Control*: part of the processor (also in hardware) which tells the datapath what needs to be done (“the brain”)
    - 对指令进行译码，生成指令对应的控制信号，控制数据通路的动作
    - 是指令的控制部件，对执行部件发出控制信号

Processor Design Process

- Five 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
    - Formulate Logic Equations
    - Design Circuits

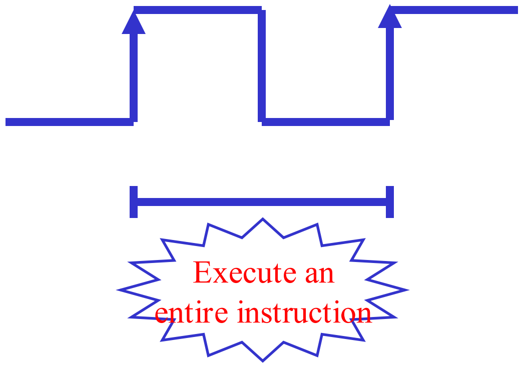
Datapath

Control



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
- CPUTime(ET) = IC× CPI × Cycle Time
  - 指令数目由编译器和ISA决定
  - 时钟周期和CPU由CPU的设计和实现决定



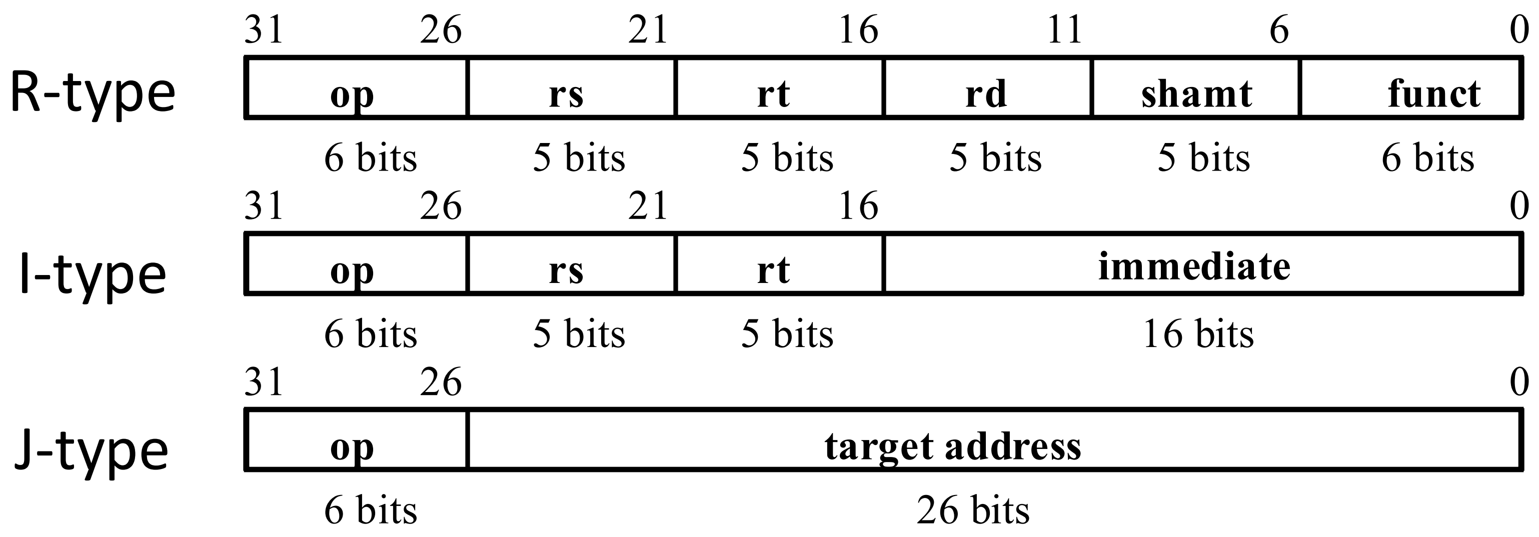
Processor Datapath and Control

- We‘re ready to look at an implementation of a simplified MIPS CPU contains only:
  - memory-reference instructions: lw, sw
  - arithmetic-logical instructions: add, sub, and, or, slt
  - control flow instructions: beq
- Generic Implementation:
  - use the program counter (PC) to supply instruction address
  - get the instruction from memory
  - read registers
  - use the instruction to decide exactly what to do
- Which instructions will use the ALU after register reading?
  - memory-reference? arithmetic? control flow?
  - **ALL of THESE**

MIPS Instruction Formats

- 无内部互锁流水级的微处理器

  （Microprocessor without Interlocked Piped Stages）
- All instructions 32-bits long
- 3 Formats:



The MIPS Subset

- R-Type
  - add rd, rs, rt
  - sub, and, or, slt

31	26	21	16	11	6	0
op (0)	**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 (35/43)	**rs**	**rt**	**immediate**	
6 bits	5 bits	5 bits	16 bits	

- BRANCH:
  - beq rs, rt, imm16

31	26	21	16	0
op (4)	**rs**	**rt**	**offset**	
6 bits	5 bits	5 bits	16 bits	

Basic Steps of Execution

- Instruction Fetch
  - Where is the instruction?

Instruction memory
address: PC

- Decode
  - What's the incoming instruction?
  - Where are the operands in an instruction?

Register file

- Execution: ALU
  - What is the function that ALU should perform?

ALU

- Memory access
  - Where is my data?

Data memory
address: effective address

- Write back results to registers
  - Where to write?

Register file

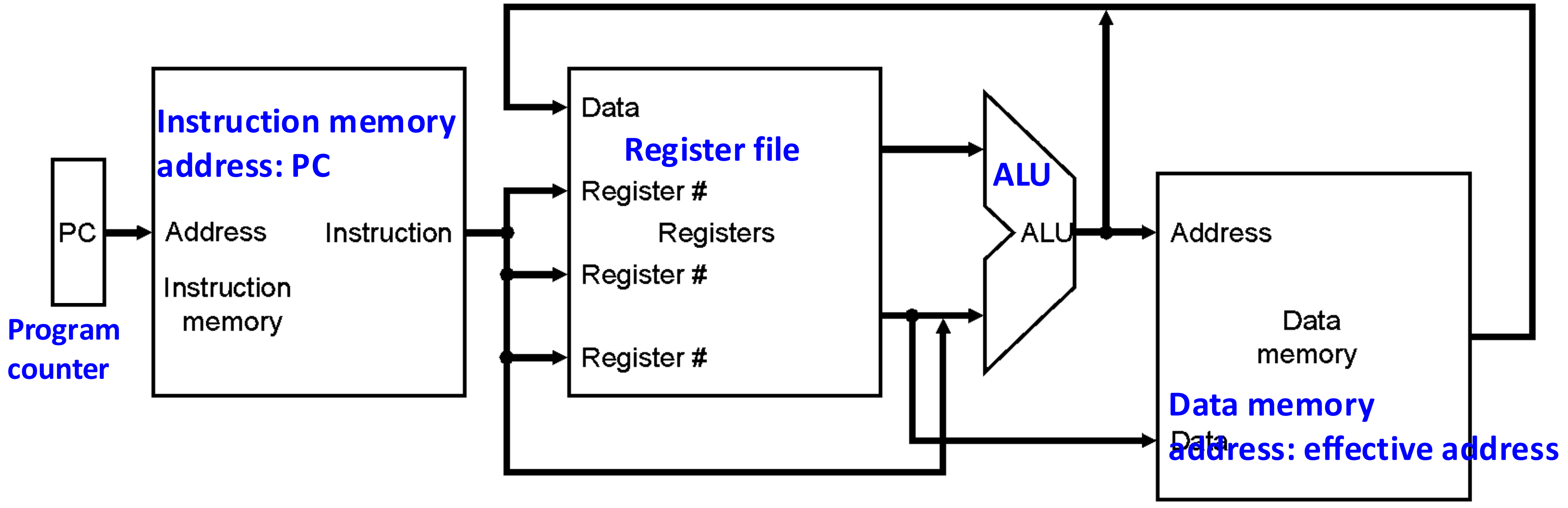
- Determine the next PC

Program counter

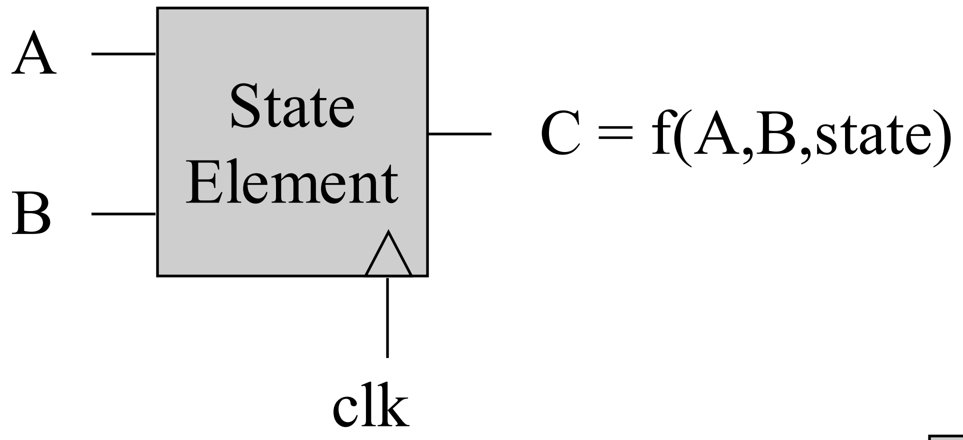
Generic Mechanism of Processor

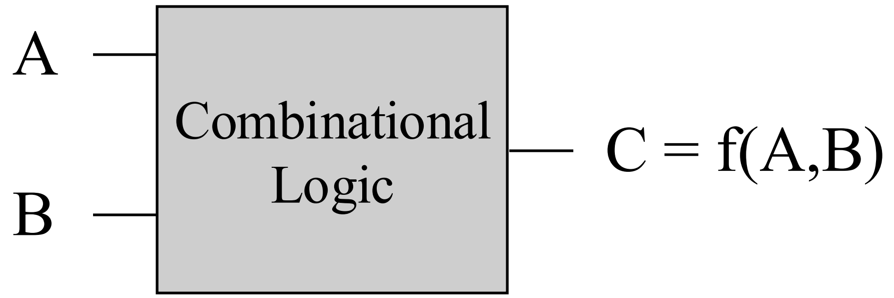
- Use Program Counter (PC) to supply an instruction address
- Get the instruction from memory
- Use the instruction to decide (control) exactly which register(s) to read or write
- Use the instruction to decide (control) exactly what operation(s) to execute

Where We're Going: The High-level View



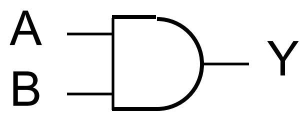
Review: Two Logical Components



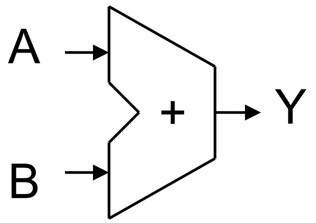


Combinational Elements

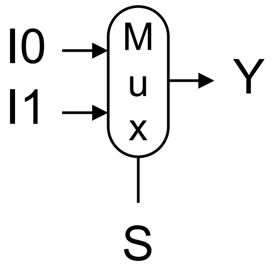
- AND-gate
  - Y = A & B



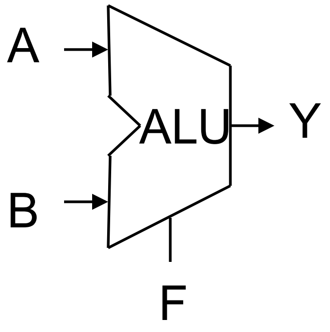
- Adder
  - Y = A + B



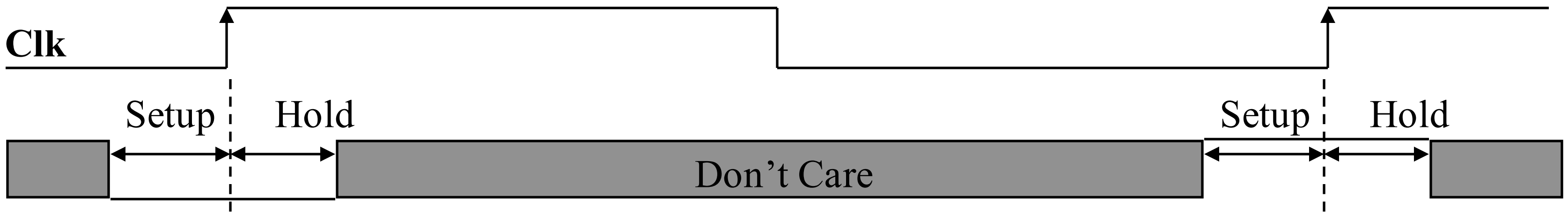
- Multiplexer
  - Y = S ? I1 : I0



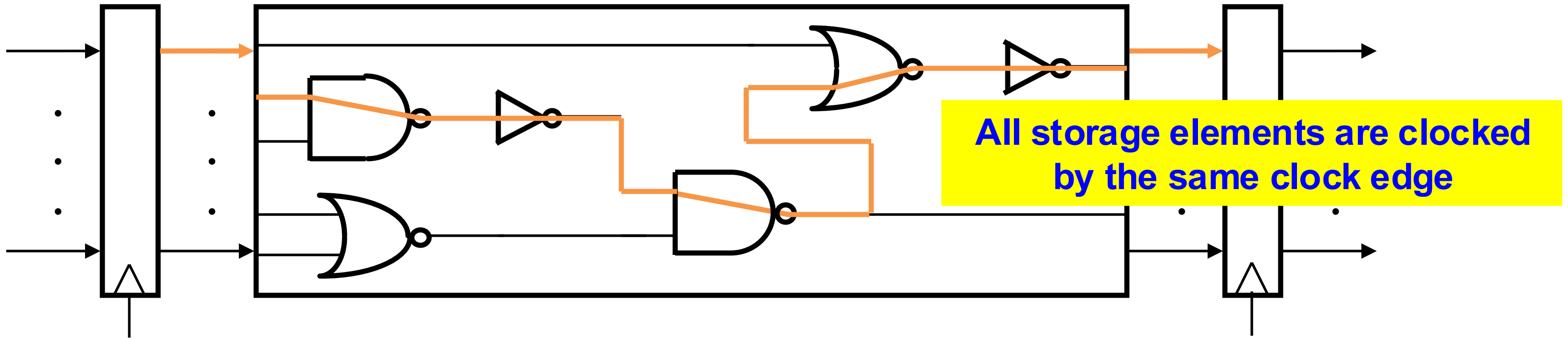
- Arithmetic/Logic Unit
  - Y = F(A, B)



Clocking Methodology (定时方法)



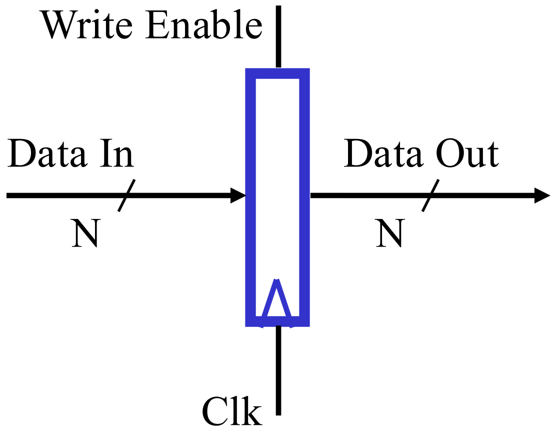
- Setup Time: how long the input must be stable before the CLK trigger for proper input read
- Hold Time: how long the input must be stable after the CLK trigger for proper input read
- CLK-to-Q Delay（锁存延迟）：how long it takes the output to change, measured from the CLK trigger

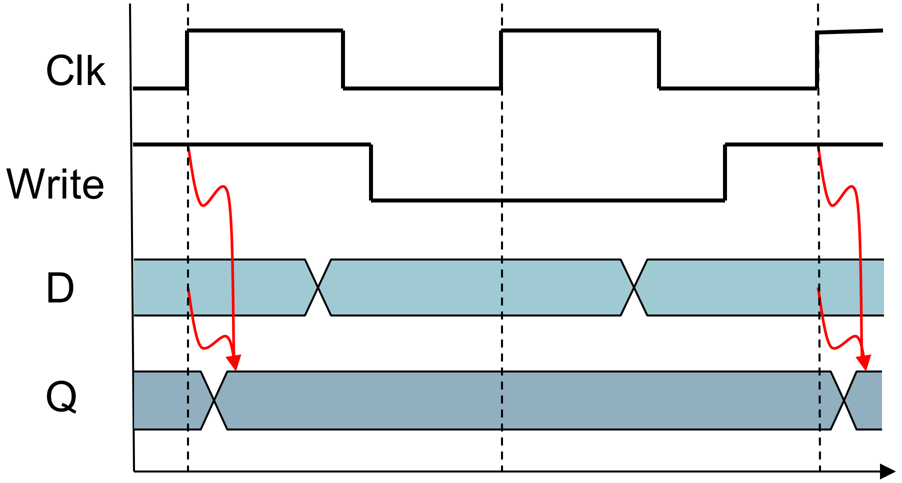


- The critical path is the longest delay between any two registers in a circuit
- Critical path determines length of clock period
  - The clock period must be longer than this critical path, or the signal will not propagate properly to that next register
  - This includes CLK-to-Q delay and setup delay

Storage Element: The Register

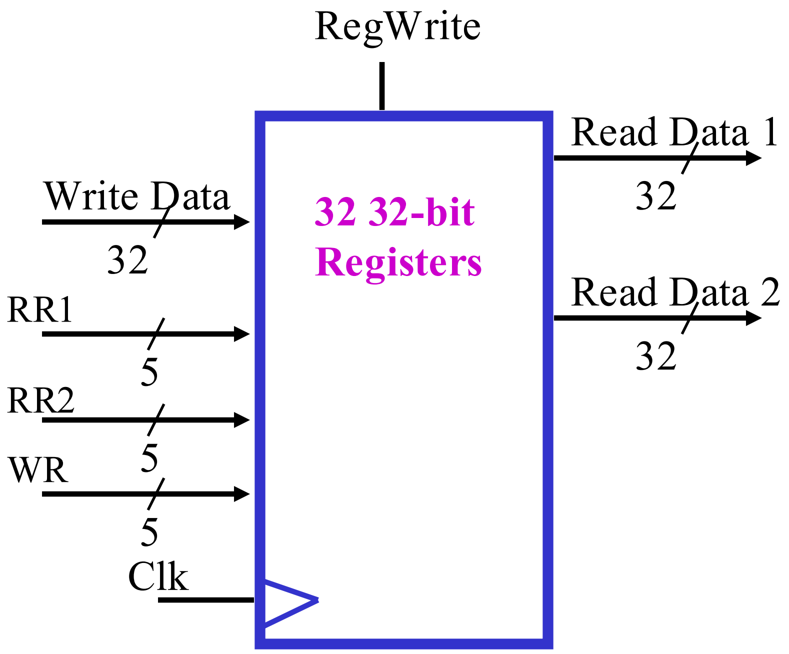
- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
- Write Enable:
  - 0: Data Out will not change
  - 1: Data Out will become Data In (on the clock edge)
    - Only updates on clock edge when write control input is 1
    - Used when stored value is required later





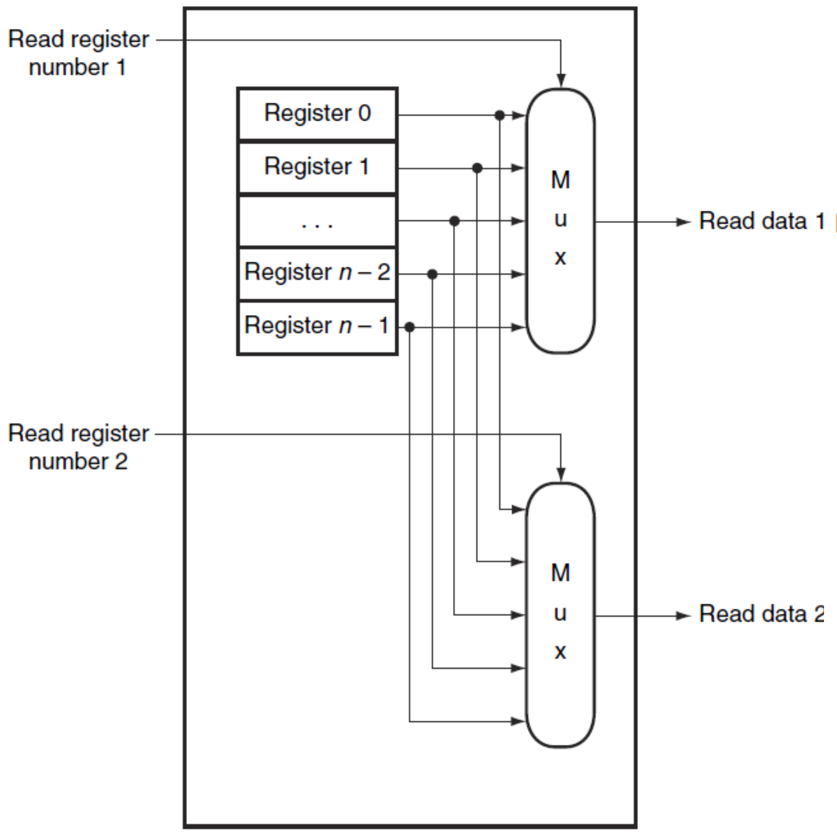
Storage Element: Register File

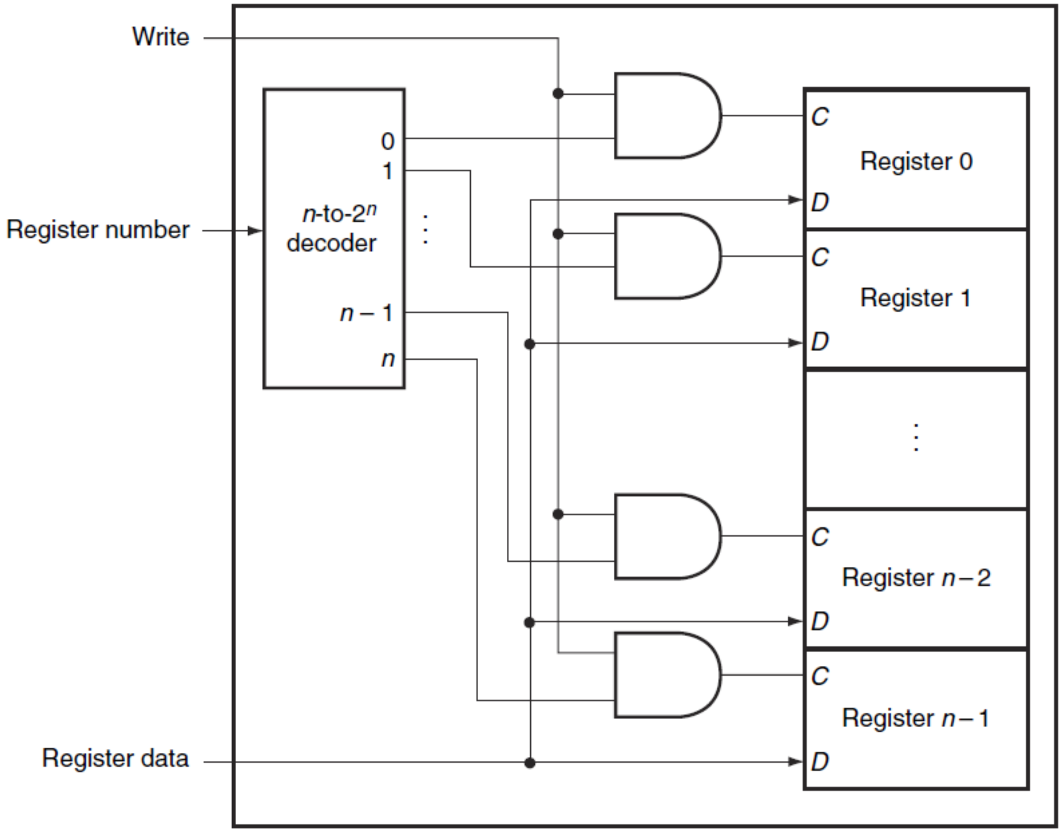
- Register File consists of (32) registers:
  - Two 32-bit output buses
  - One 32-bit input bus
- Register is selected by:
  - RR1 selects the register to put on bus "Read Data 1"
  - RR2 selects the register to put on bus "Read Data 2"
  - WR selects the register to be written
  - via WriteData when RegWrite is 1
- Clock input (CLK)



Inside the Register File

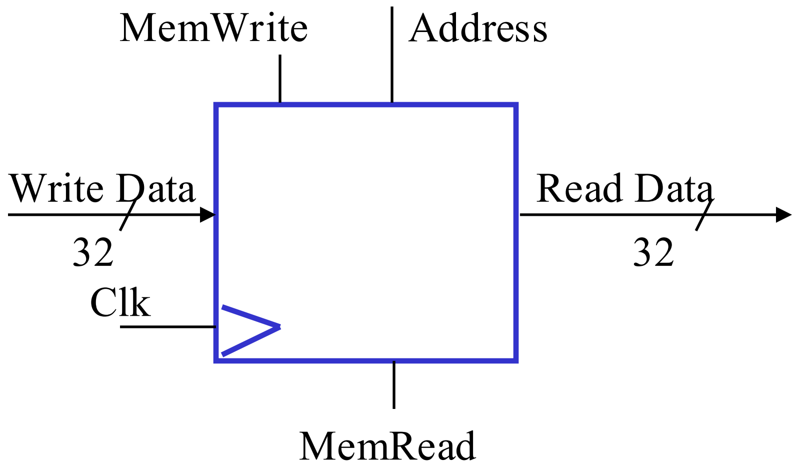
- The implementation of two read ports register file
  - n registers
  - done with a pair of n-to-1 multiplexors, each 32 bits wide.





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
  - If 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."



RTL: Register Transfer Language

- Describes the movement and manipulation of data between storage elements:
  - R[i]表示寄存器堆中寄存器i的内容
  - M[addr]表示存储单元addr的内容
  - 传送方向用<- 表示，传送源在右目的在左
  - 程序计数器PC直接用PC表示其内容

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

中国科学院大学 University of Chinese Academy of Sciences B0911006Y-01 2024-2025学年春季学期

计算机组成原理
Principles of Computer Organization

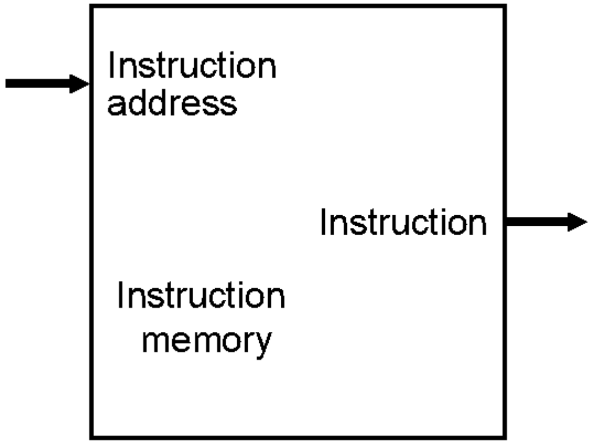
单周期处理器 II

数据通路的设计方法

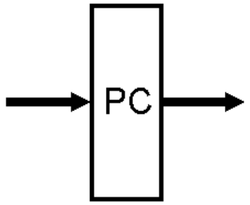
主讲教师：石 侃
shikan@ict.ac.cn

2025年4月14日

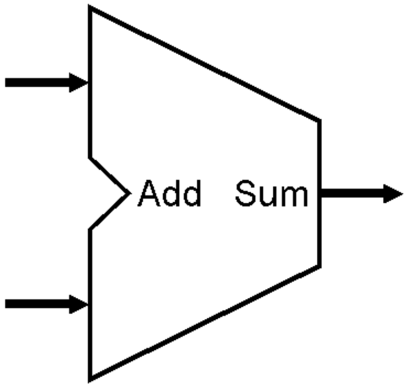
Instruction Fetch and Program Counter Management



a. Instruction memory



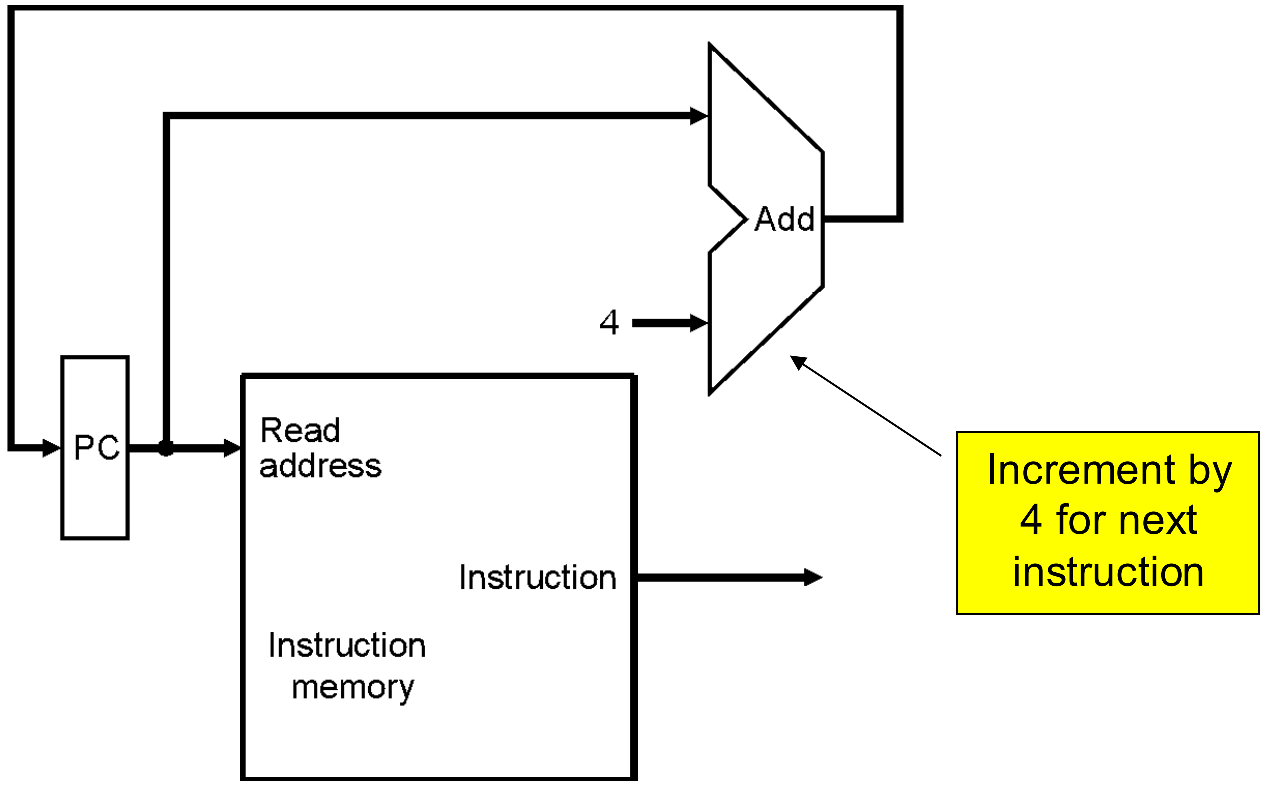
b. Program counter



c. Adder

Overview of the Instruction Fetch Unit

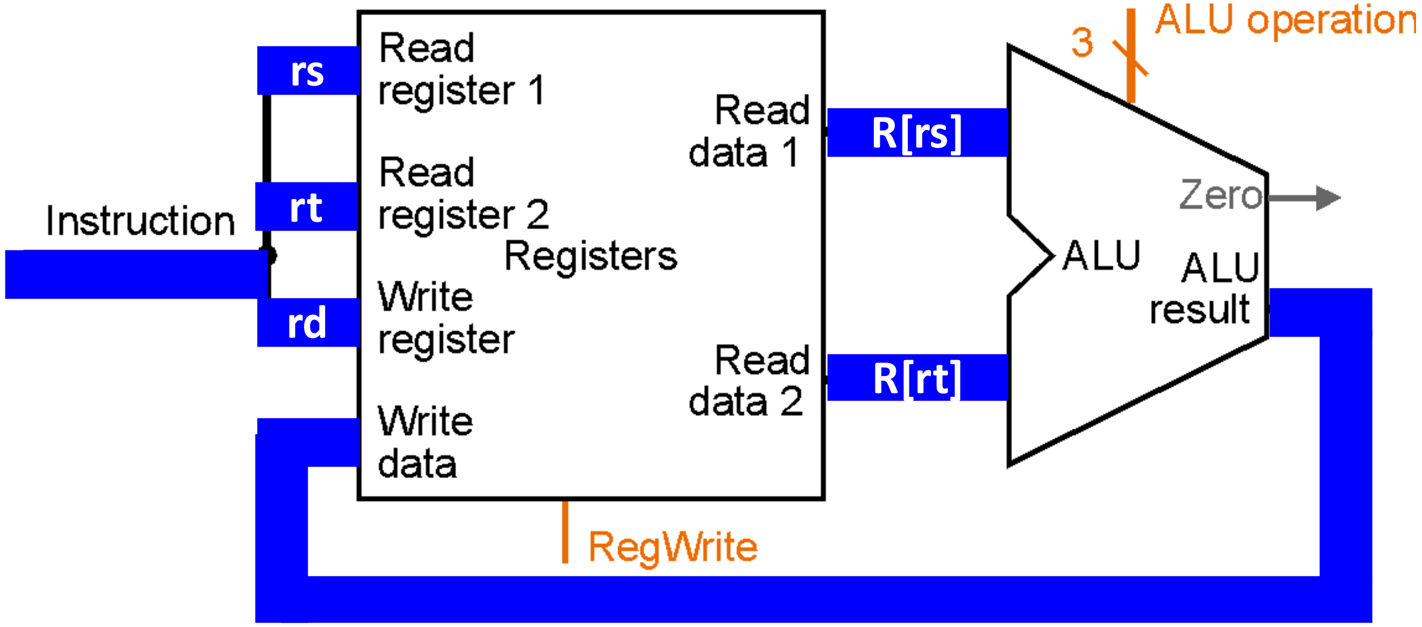
- The common RTL operations
  - Fetch the Instruction: inst <- mem[PC]
  - Update the program counter:
    - Sequential Code: PC <- PC + 4
    - Branch and Jump   PC <- "something else"



Datapath for Register-Register Operations

- R[rd] <- R[rs] op R[rt] Example: *add rd, rs, rt*
  - RR1, RR2, and WR comes from instruction's rs, rt, and rd fields
  - ALUoperation and RegWrite: control logic after decoding instruction

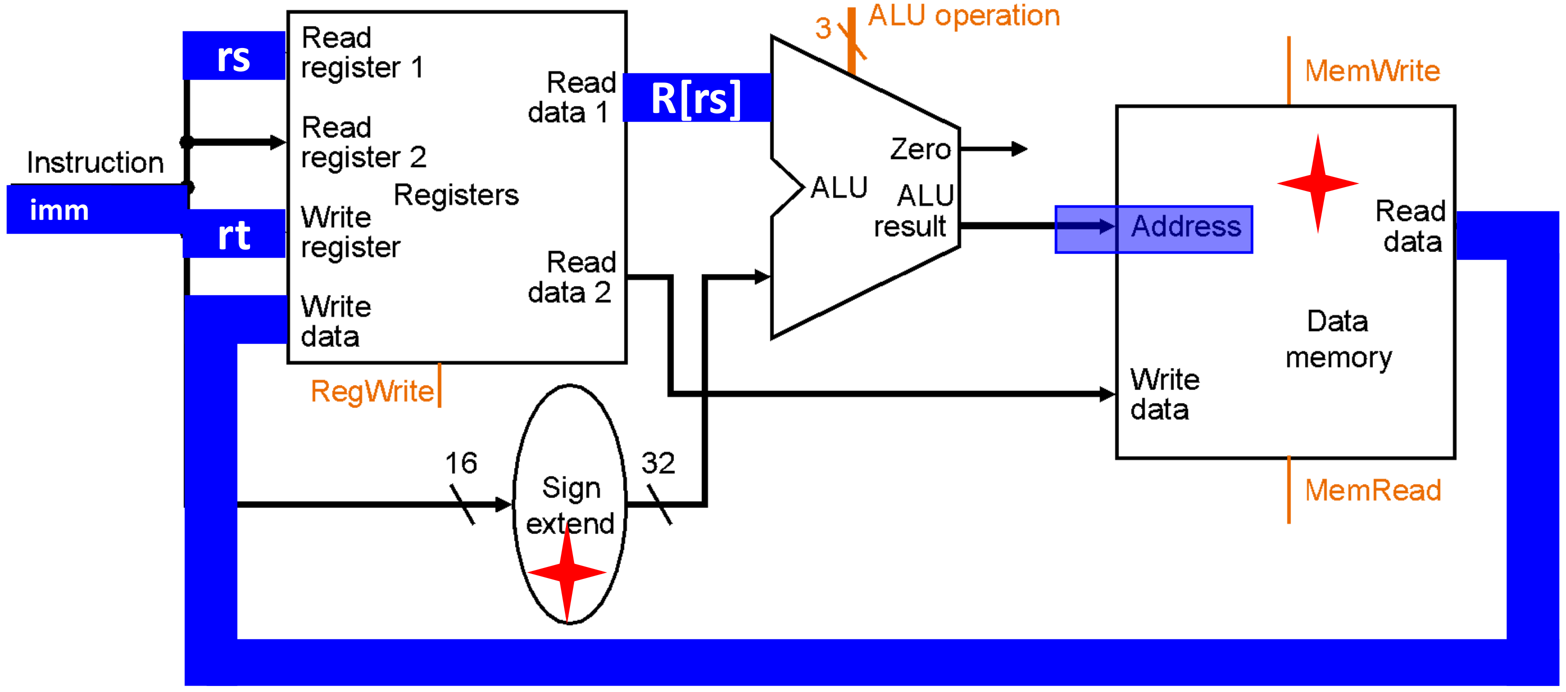
31–26	25–21	20–16	15–11	10–6	5–0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



Datapath for Load Operations

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

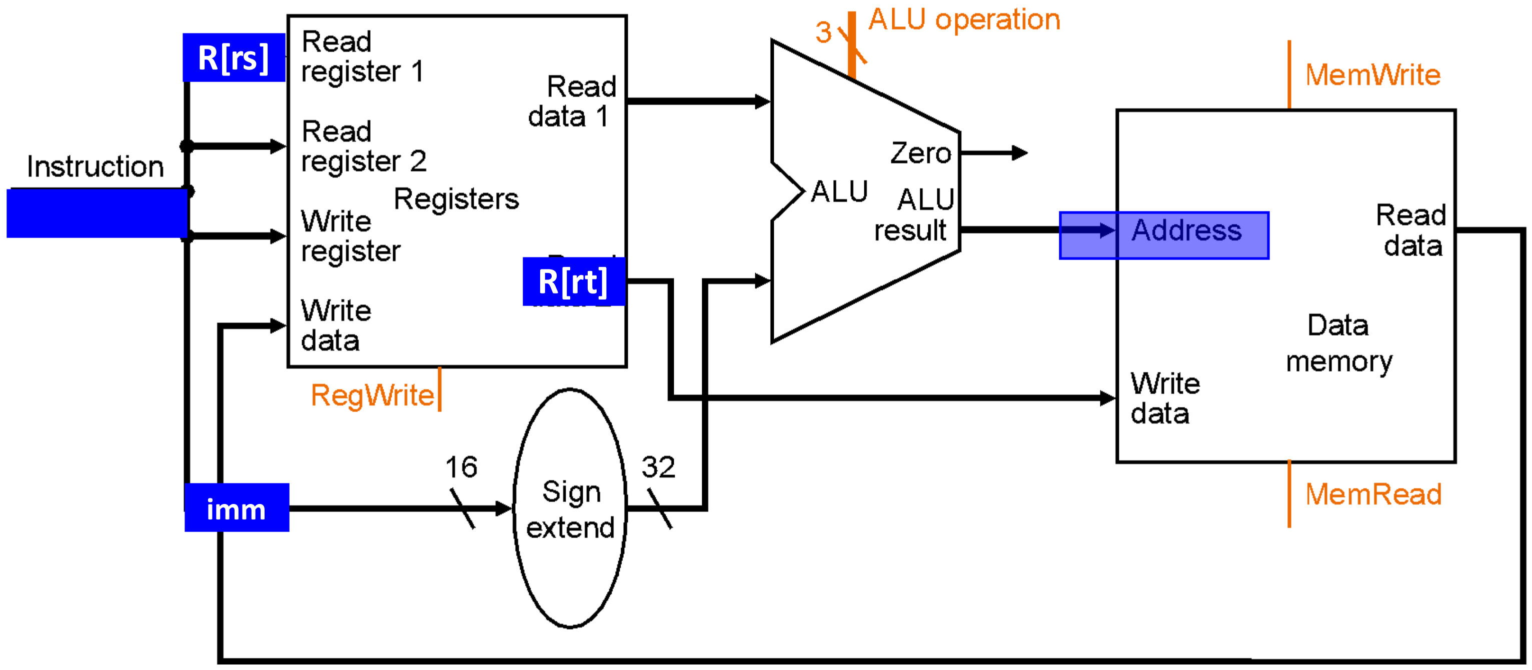
31 26	21	16	0
op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits



Datapath for Store Operations

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

31–26	25–21	20–16	15–0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits



Datapath for Branch Operations

- 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

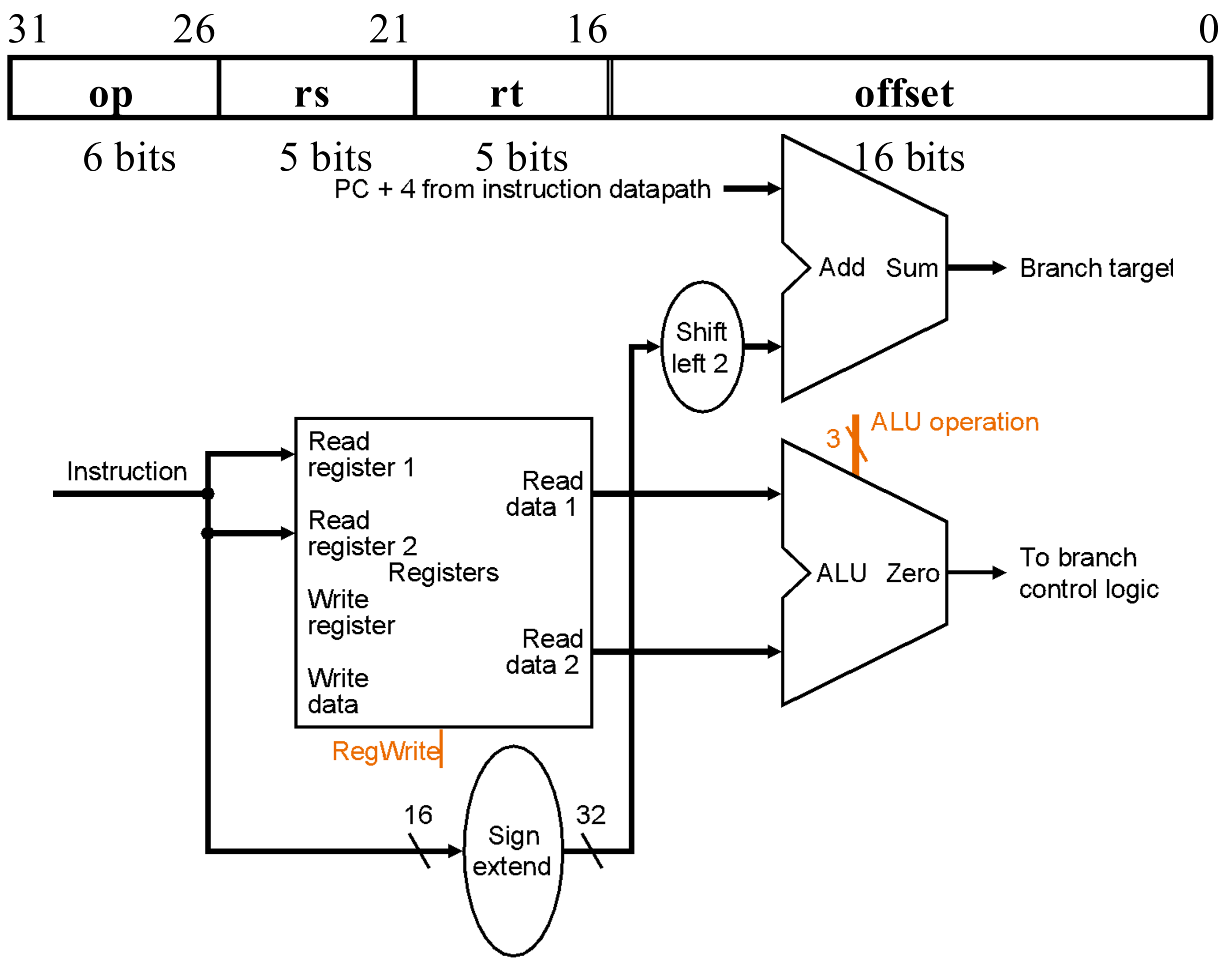
Datapath for Branch Operations

- Z <- (rs == rt); if Z, PC = PC+4+imm16; else PC = PC+4
  - Example: *beq   rs, rt, imm16*

31 ... 26	21	16	0
op	**rs**	**rt**	**offset**
6 bits	5 bits	5 bits	16 bits

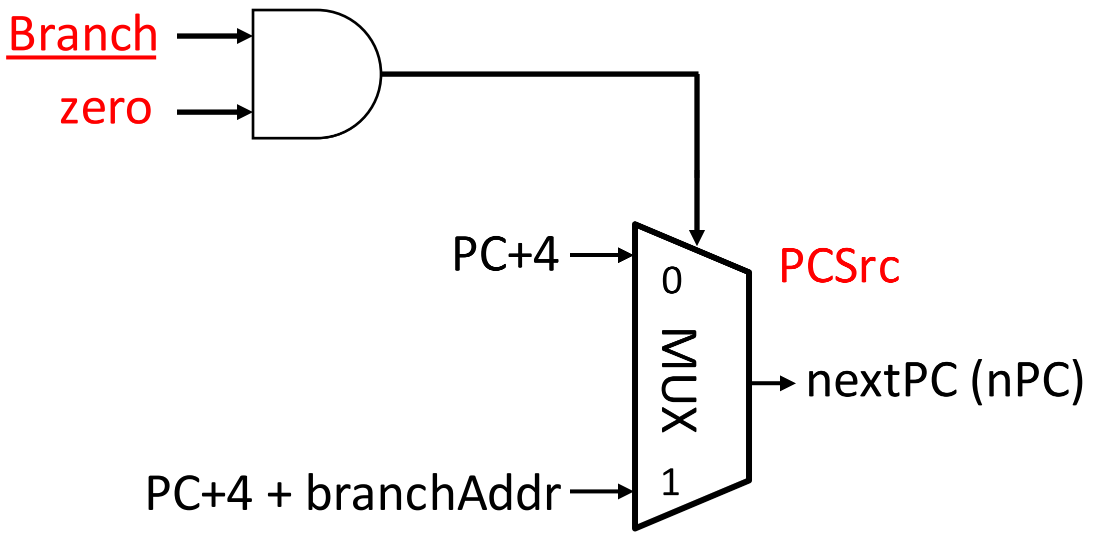
Datapath for Branch Operations

- Z <- (rs == rt); if Z, PC = PC+4+imm16; else PC = PC+4
  - Example: *beq   rs, rt, imm16*



Datapath for Branch Operations

- Revisit “next address logic”:
  - PCSrc should be 1 if branch, 0 otherwise



Branch	zero	PCSrc
0	0	0
0	1	0
1	0	0
1	1	1

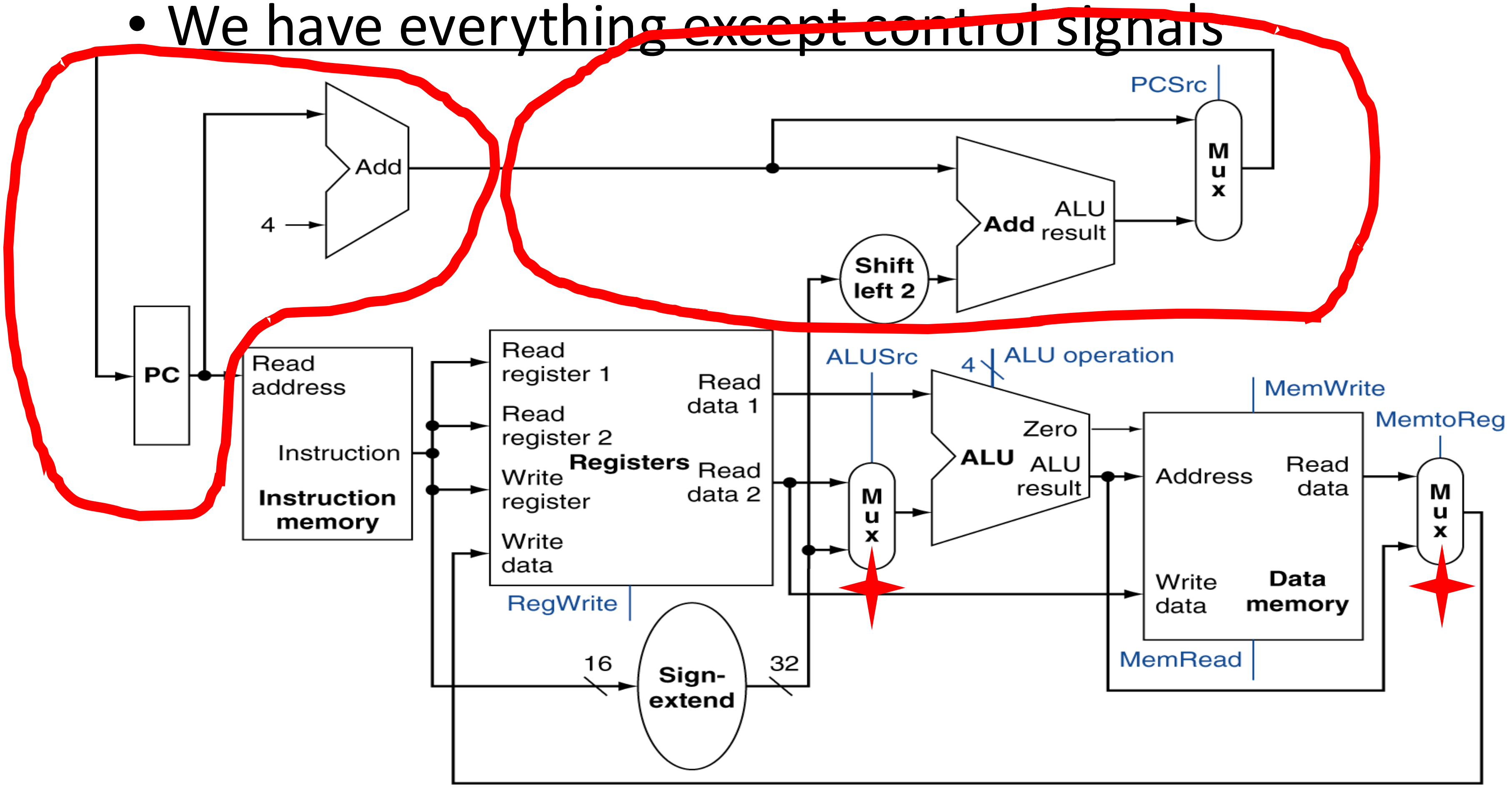
How does this change if we add `bne`*?*

Binary Arithmetic for the Next Address

- In theory, the PC is a 32-bit byte address into the instruction memory:
  - Sequential operation: PC<31:0> = PC<31:0> + 4
  - Branch operation: PC<31:0> = PC<31:0> + 4 + SignExt[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 (Least Significant Bit) 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<31:2>:
  - Sequential operation: PC<31:2> = PC<31:2> + 1
  - Branch operation: PC<31:2> = PC<31:2> + 1 + SignExt[Imm16]
  - In either case: Instruction Memory Address = PC<31:2> concat "00"

Putting it All Together: A Single Cycle Datapath

- We have everything except control signals

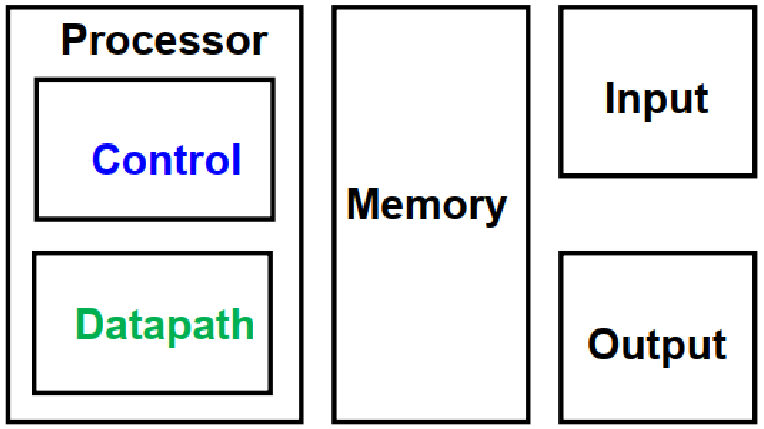


Key Points

- 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?

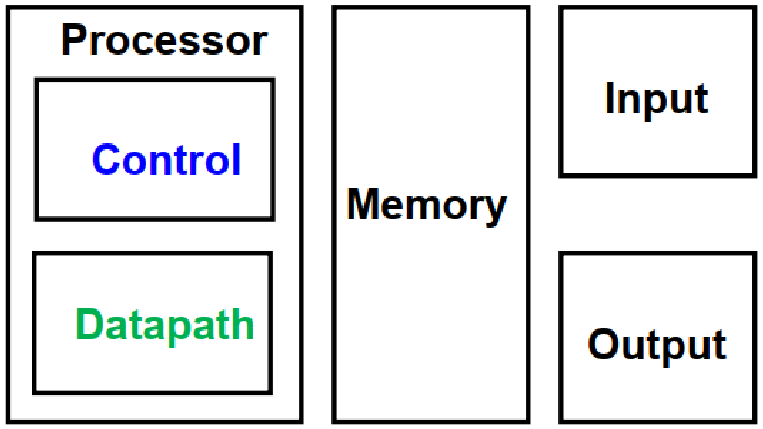
The Big Picture: Where are we now

- Five classic components:



The Big Picture: Where are we now

- Five classic components:



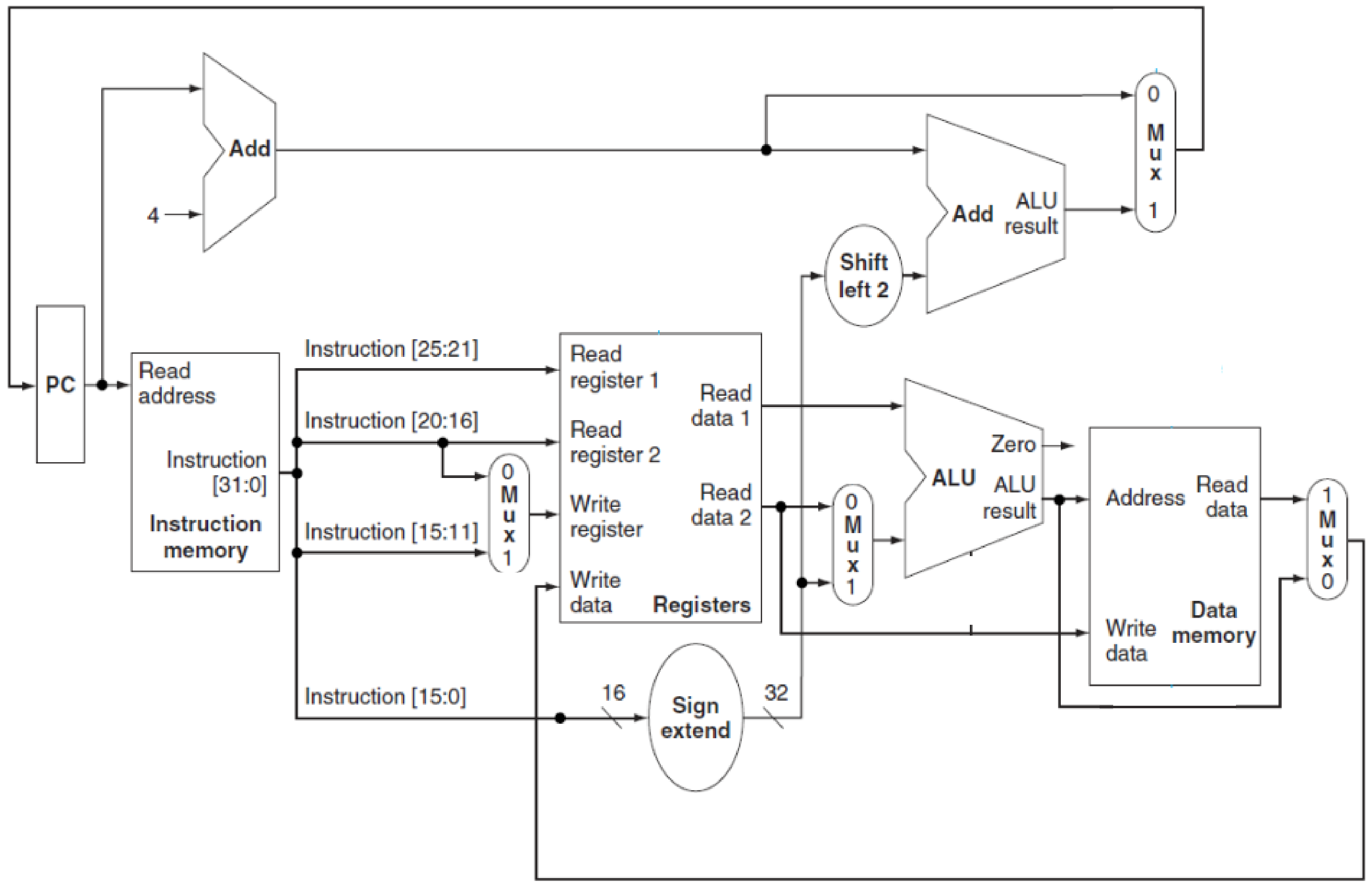
- Our target now: design the control logic

1）根据每条指令的功能，分析控制信号的取值，并在指令译码表中列出

2）根据列出的指令和控制信号的关系，写出每个控制信号的逻辑表达式

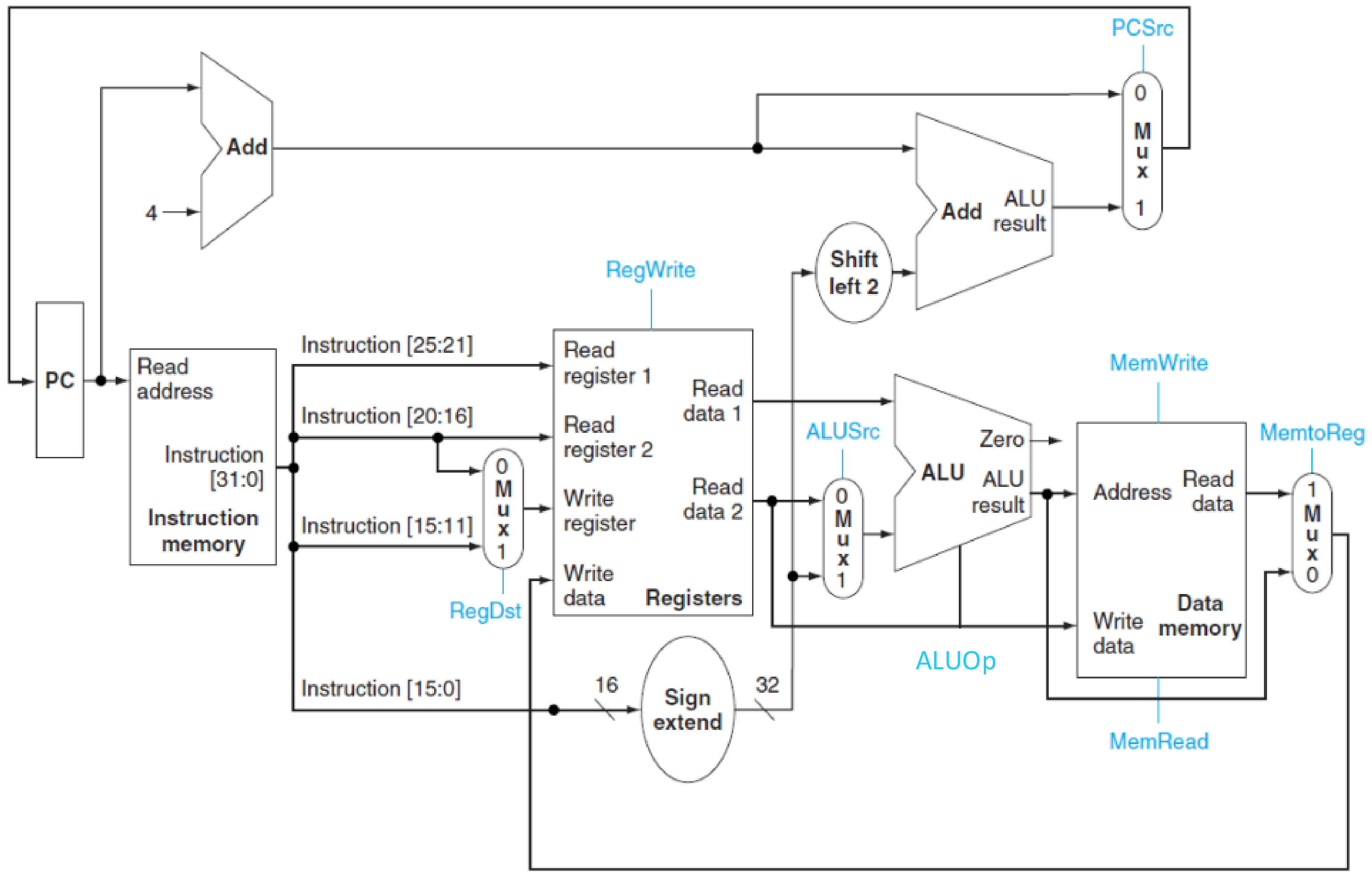
A Single Cycle Datapath

- We have everything except control signals

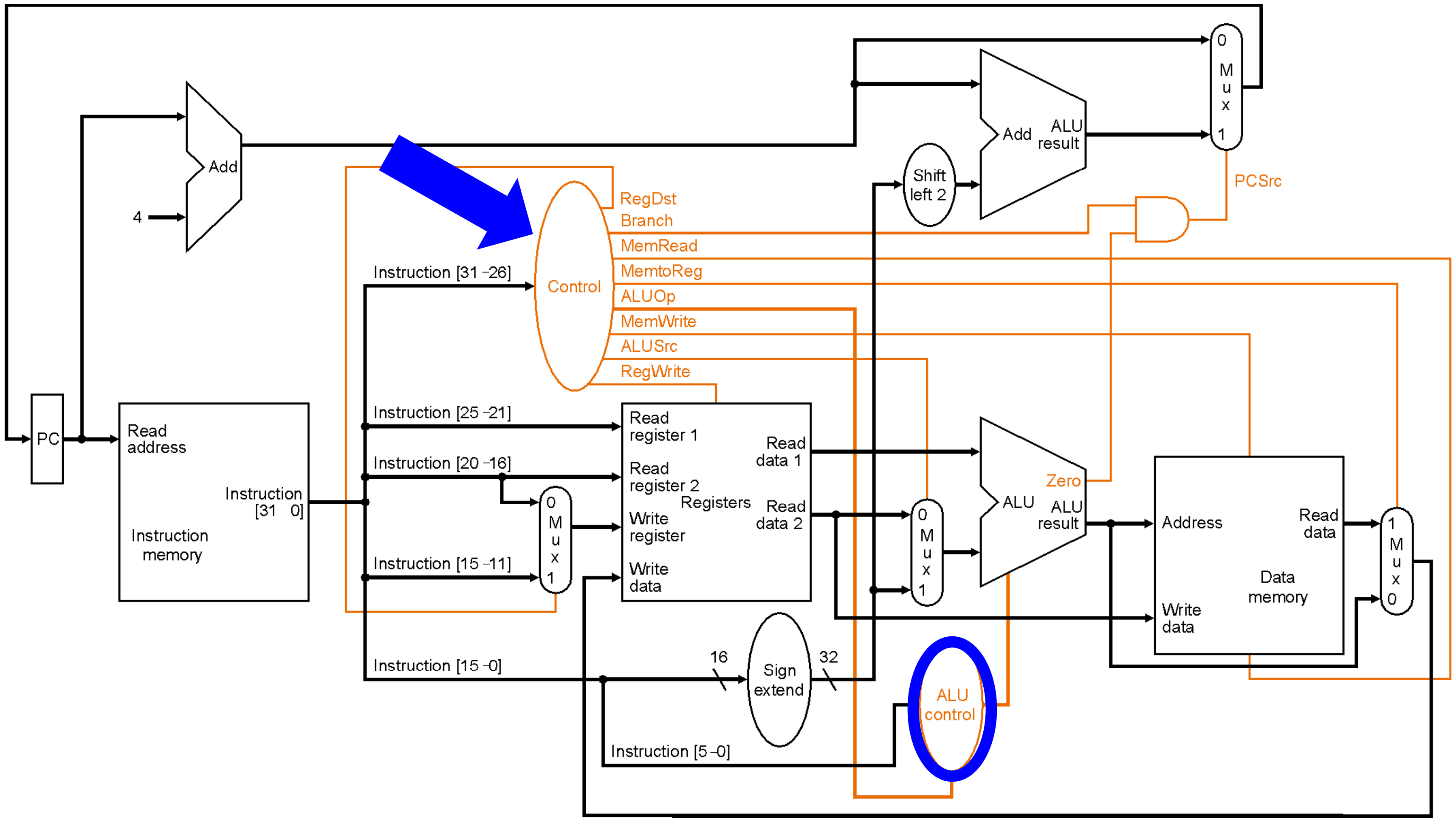


A Single Cycle Datapath

- We have everything except control signals



Okay, then, what about those Control Signals?



PCSrc
Add
4
Add
ALU result
Shift left 2
M u x
0
1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31 –26]
Instruction [25 –21]
Instruction [20 –16]
Instruction [15 –11]
Instruction [15 –0]
Instruction [5 –0]
PC
Read address
Instruction [31 0]
Instruction memory
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign extend
32
ALU
Zero
ALU result
ALU control
Address
Read data
Write data
Data memory