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

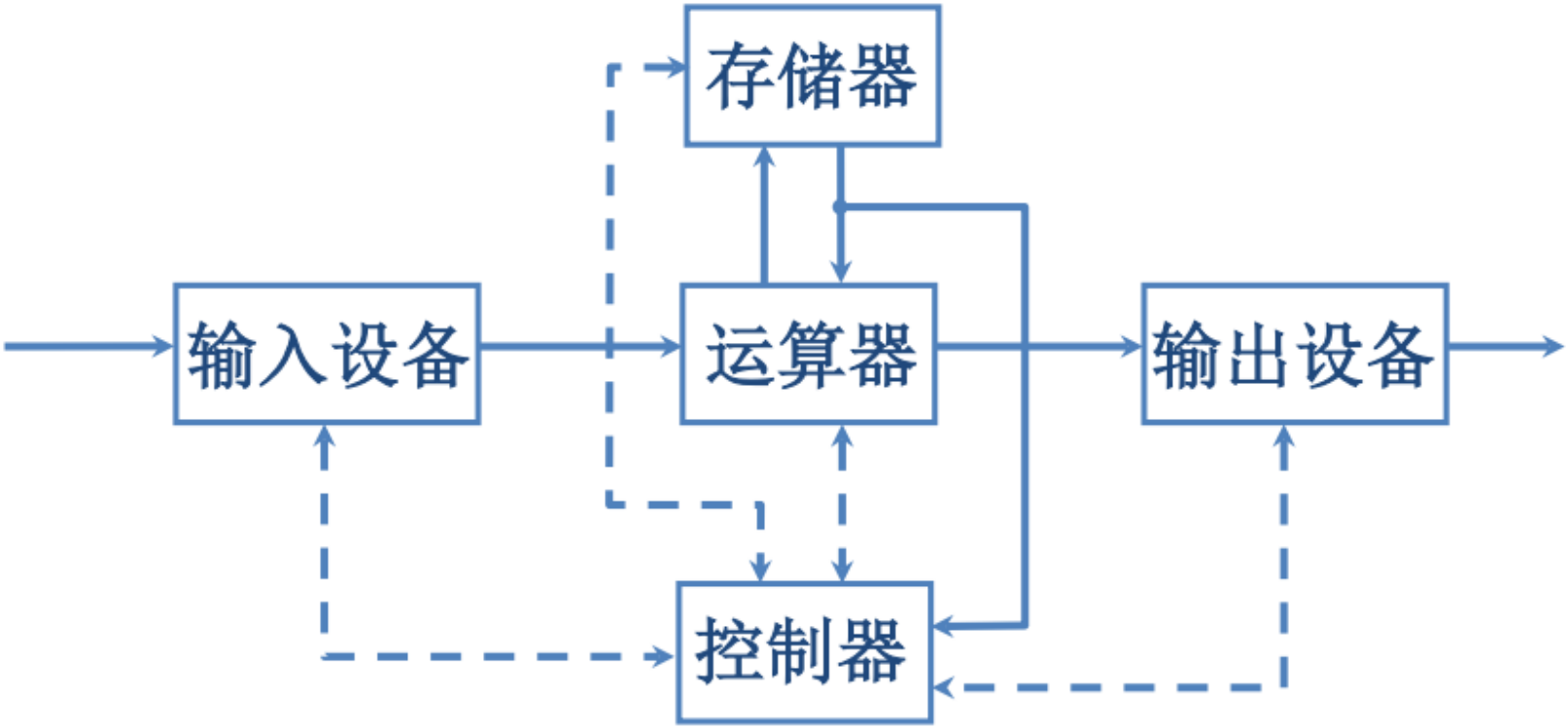
单周期处理器 I

基本组件，如何设计、组装数据通路

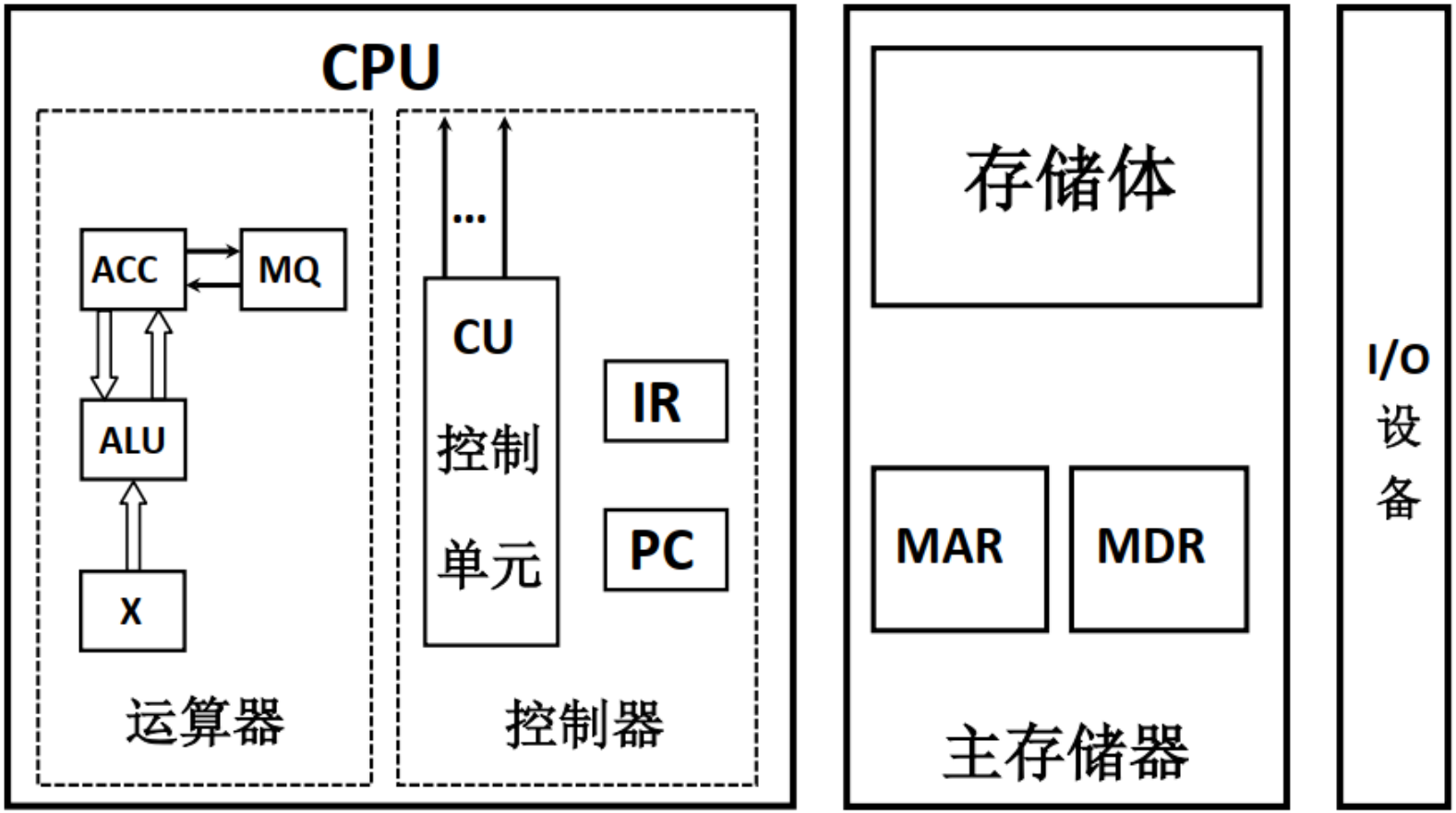
主讲教师：石 侃
shikan@ict.ac.cn

2025年4月9日

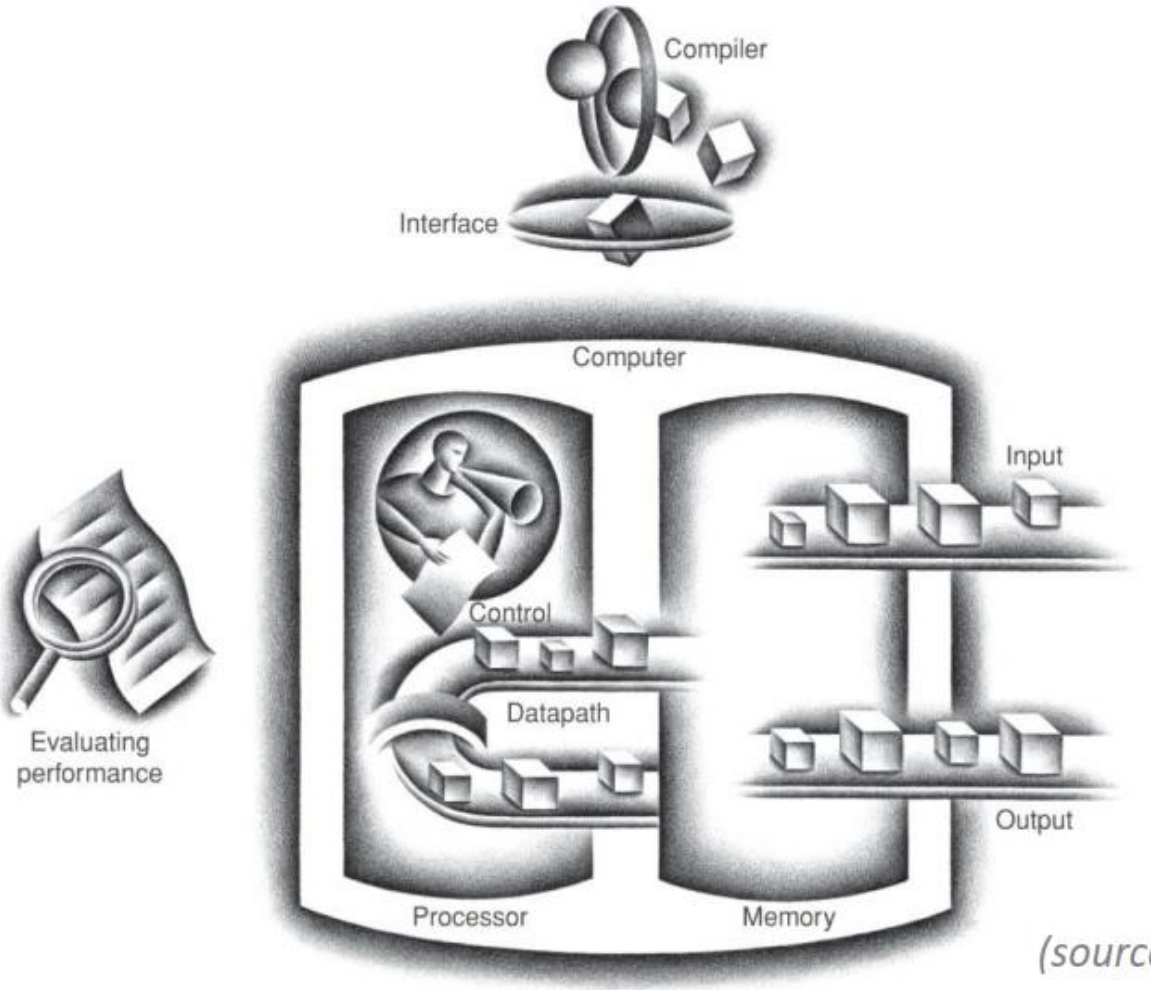
回顾：冯诺依曼计算机五大部件



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

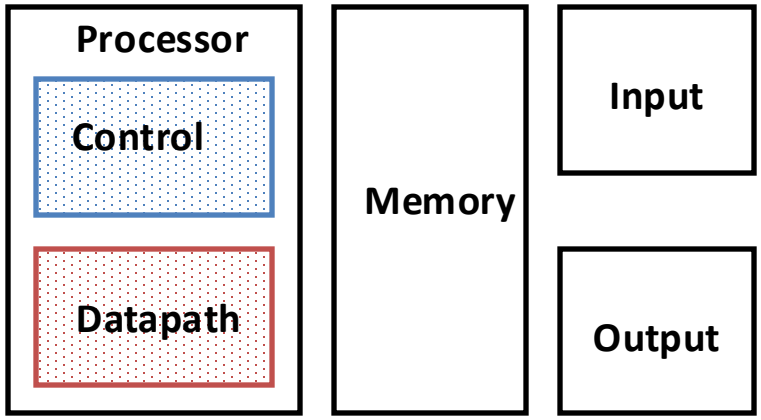
- **Processor (CPU)**: Implements the instructions of the Instruction Set Architecture (ISA)
  - *Datapath*: part of the processor that contains the hardware necessary to perform operations required by the processor ("the brawn")
    - 指令执行过程中，数据所经过的路径，包括路径中的部件
    - 是指令执行的部件
    - 组合元件和存储元件通过总线或分散方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:

Datapath

- 1. Analyze instruction set → datapath requirements
- 2. Select set of datapath components & establish clock methodology
- 3. Assemble datapath meeting the requirements

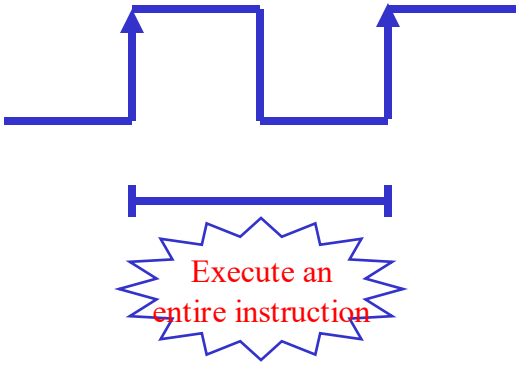


Control

- 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

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



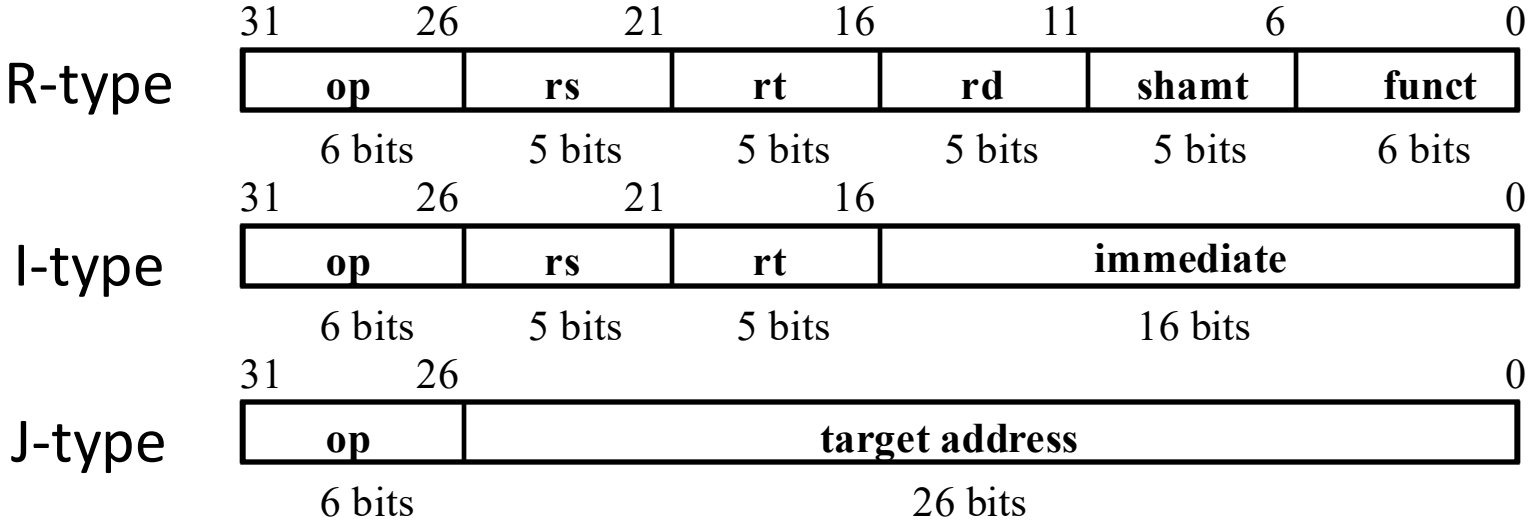
- CPUTime(ET) = IC× CPI × Cycle Time
  - 指令数目由编译器和ISA决定
  - 时钟周期和CPU由CPU的设计和实现决定

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**

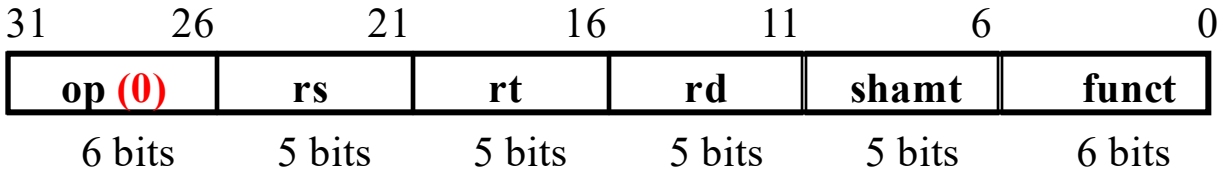
MIPS Instruction Formats

- 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:

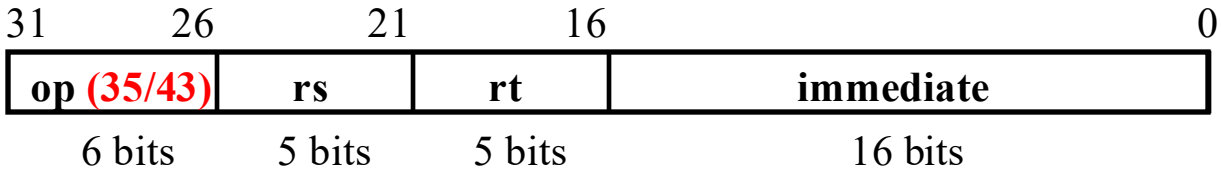


The MIPS Subset

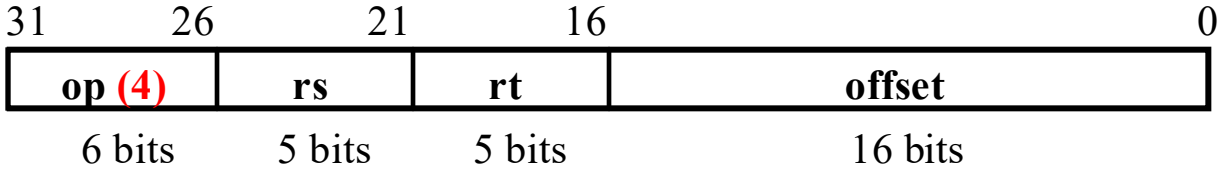
- R-Type
  - add rd, rs, rt
  - sub, and, or, slt



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



- BRANCH:
  - beq rs, rt, imm16



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

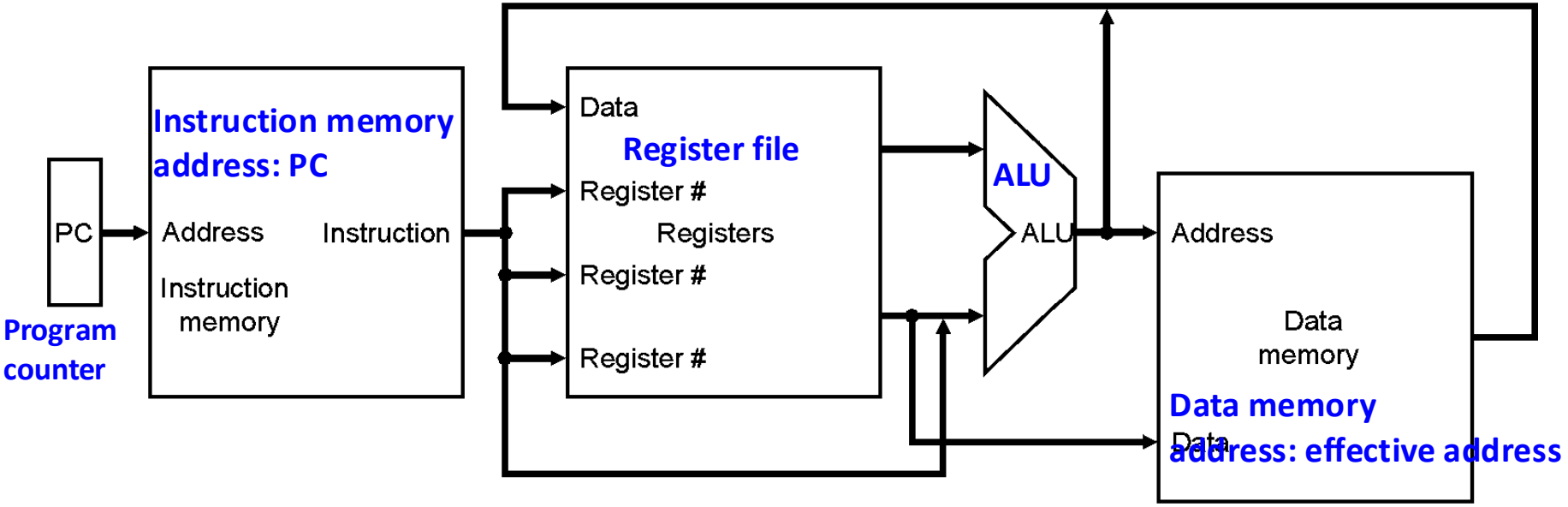
- Determine the next PC

Program counter

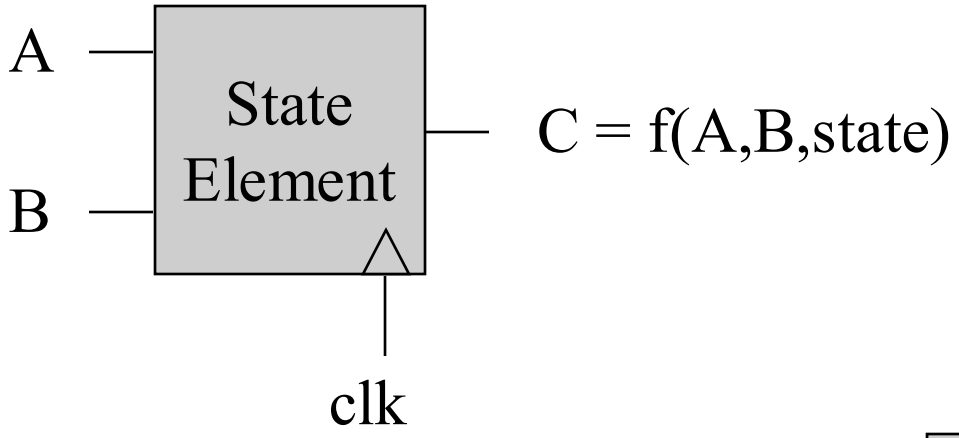
Generic Mechanism of Processor

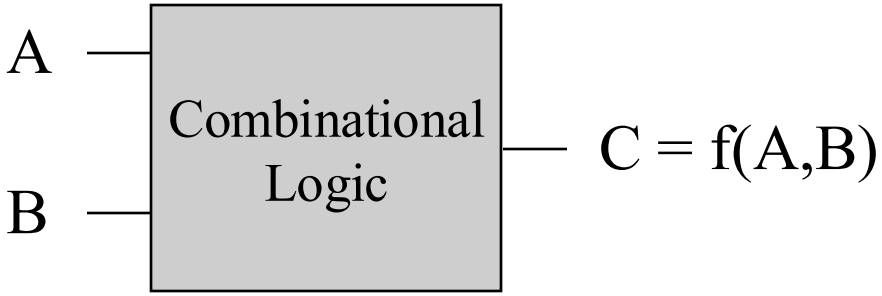
- Use Program Counter (PC) to supply an instruction address
- Get the instruction from memory
- Use the instruction to decide (control) exactly which register(s) to read or write
- Use the instruction to decide (control) exactly what operation(s) to execute

Where We're Going: The High-level View



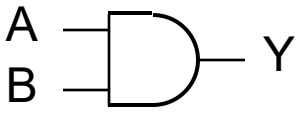
Review: Two Logical Components



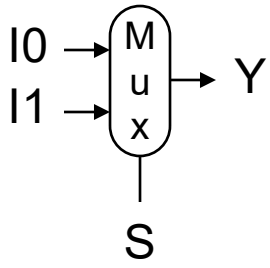


Combinational Elements

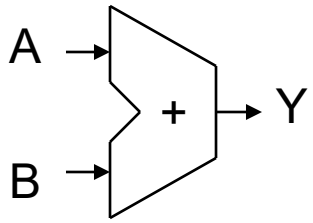
- AND-gate
  - Y = A & B



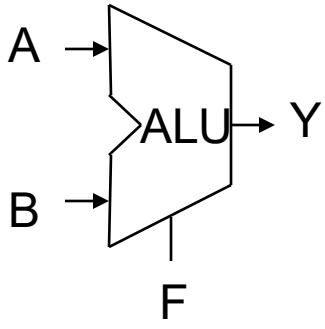
- Multiplexer
  - Y = S ? I1 : I0



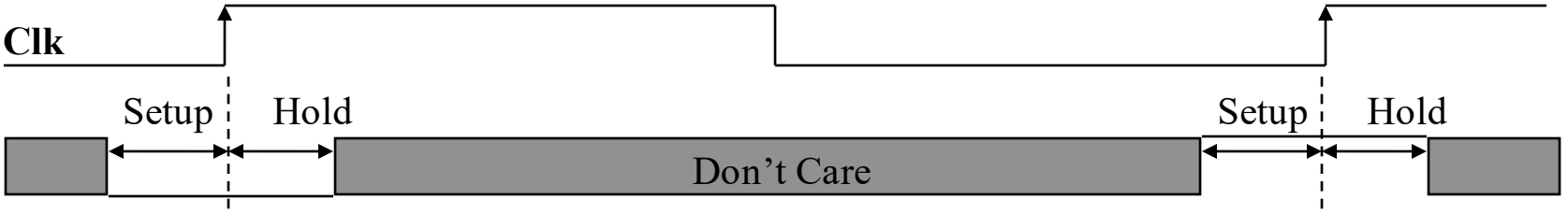
- Adder
  - Y = A + B



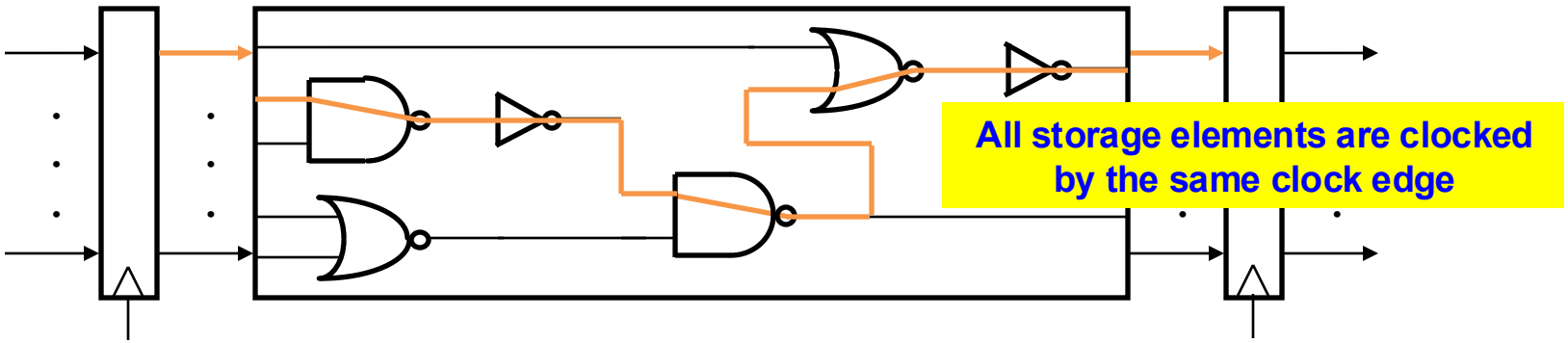
- Arithmetic/Logic Unit
  - Y = F(A, B)



Clocking Methodology (定时方法)



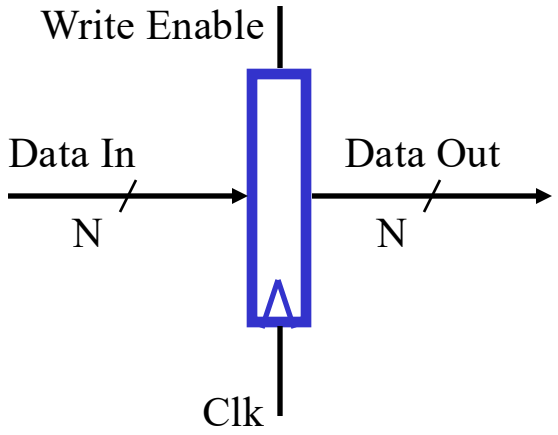
- Setup Time: how long the input must be stable before the CLK trigger for proper input read
- Hold Time: how long the input must be stable after the CLK trigger for proper input read
- CLK-to-Q Delay（锁存延迟）: how long it takes the output to change, measured from the CLK trigger

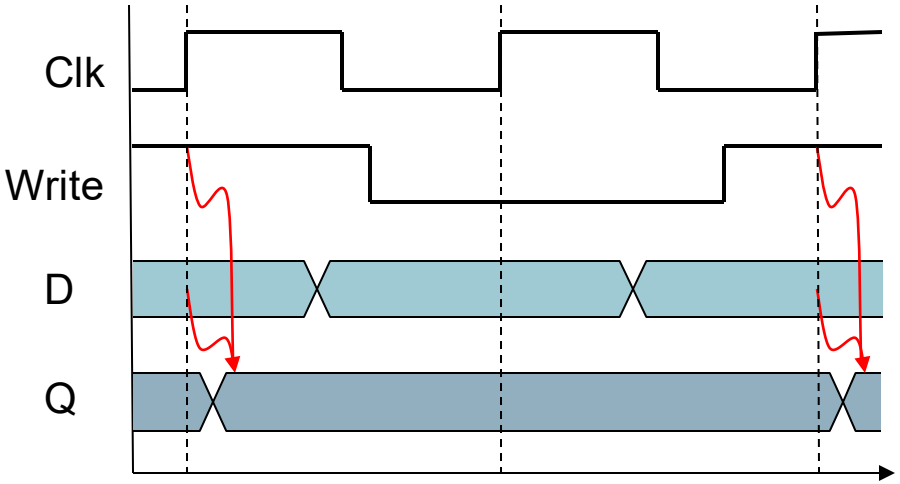


- The critical path is the longest delay between any two registers in a circuit
- Critical path determines length of clock period
  - The clock period must be longer than this critical path, or the signal will not propagate properly to that next register
  - This includes CLK-to-Q delay and setup delay

Storage Element: The Register

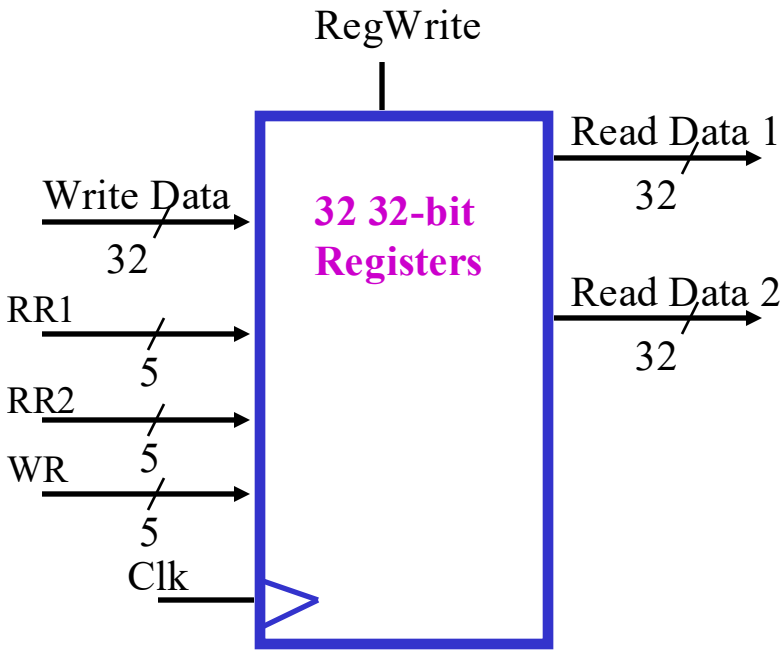
- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
- Write Enable:
  - 0: Data Out will not change
  - 1: Data Out will become Data In (on the clock edge)
    - Only updates on clock edge when write control input is 1
    - Used when stored value is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





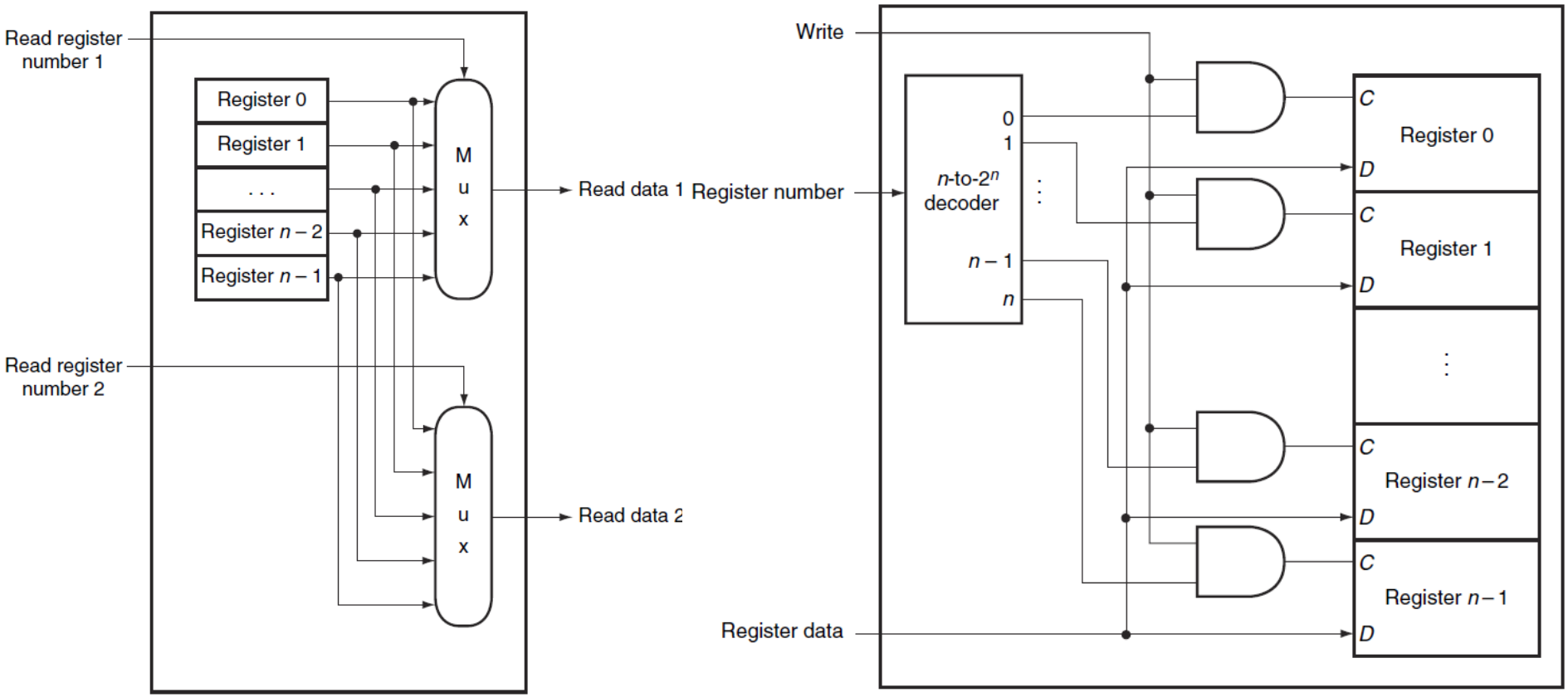
Storage Element: Register File

- 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)



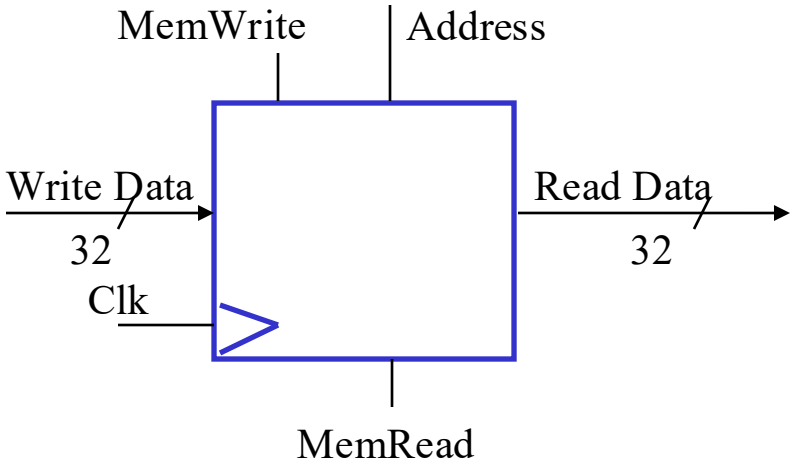
Inside the Register File

- 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

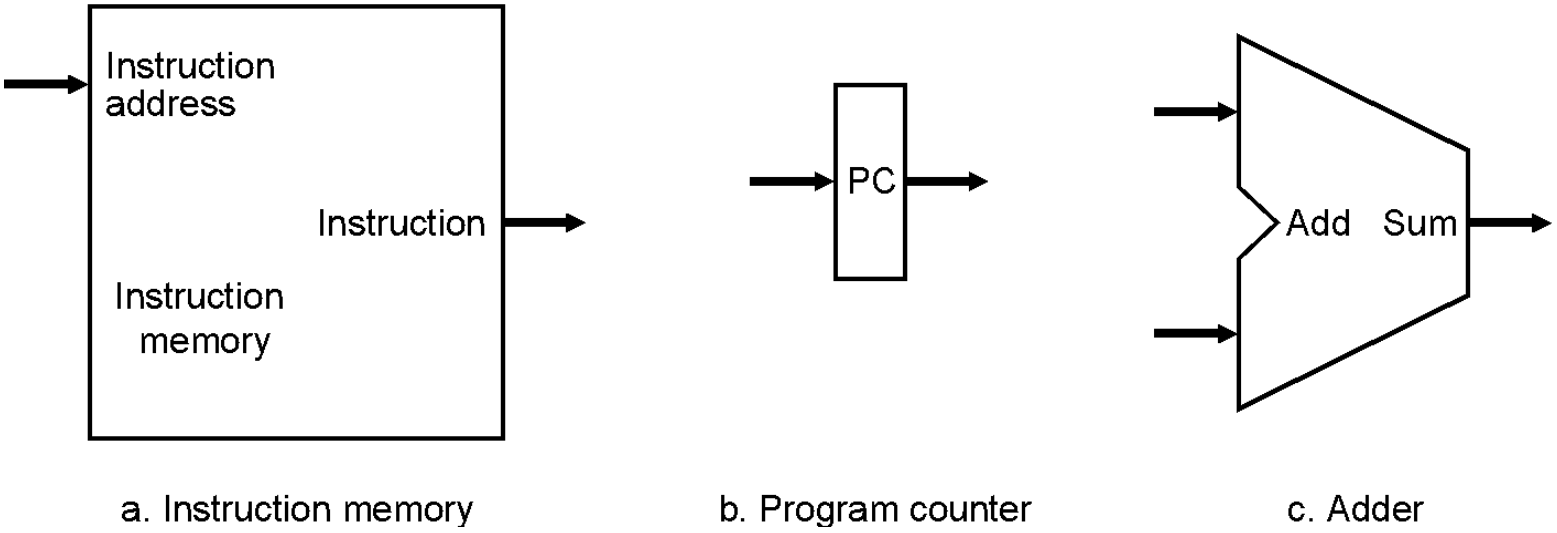
单周期处理器 II

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



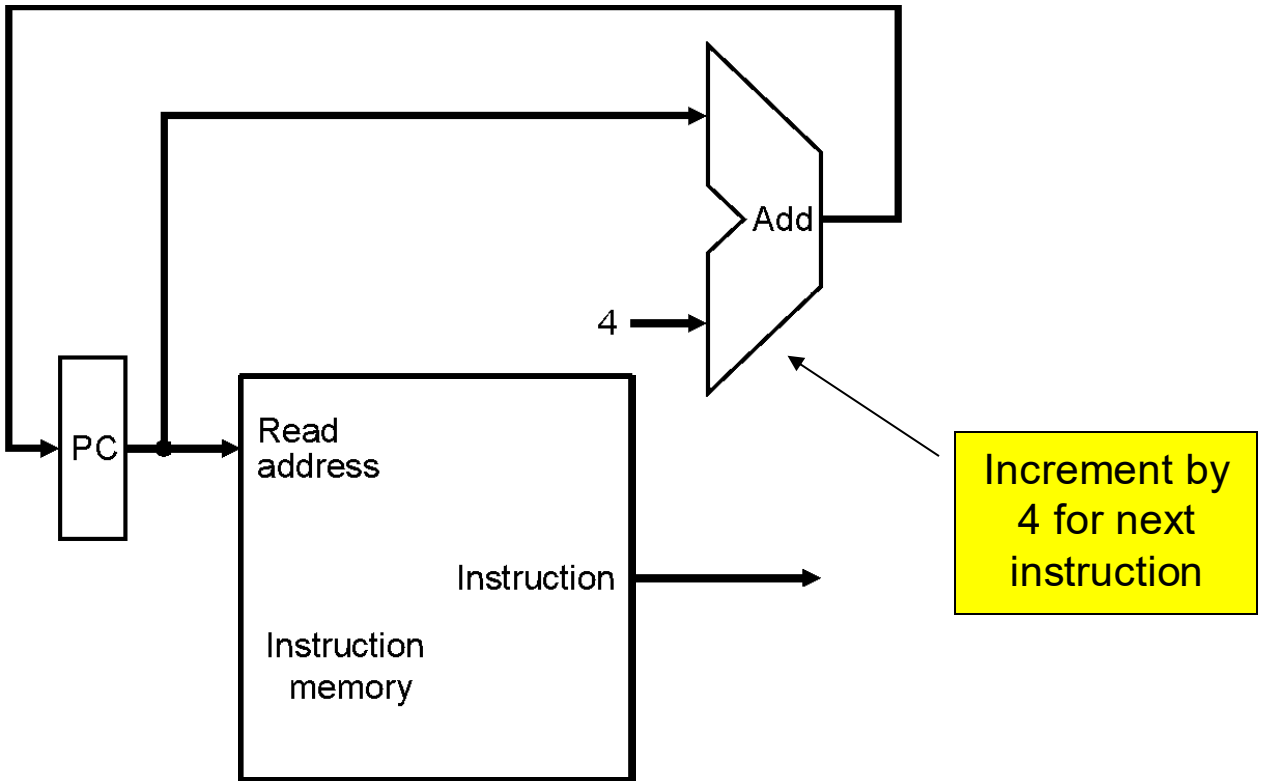
a. Instruction memory

b. Program counter

c. Adder

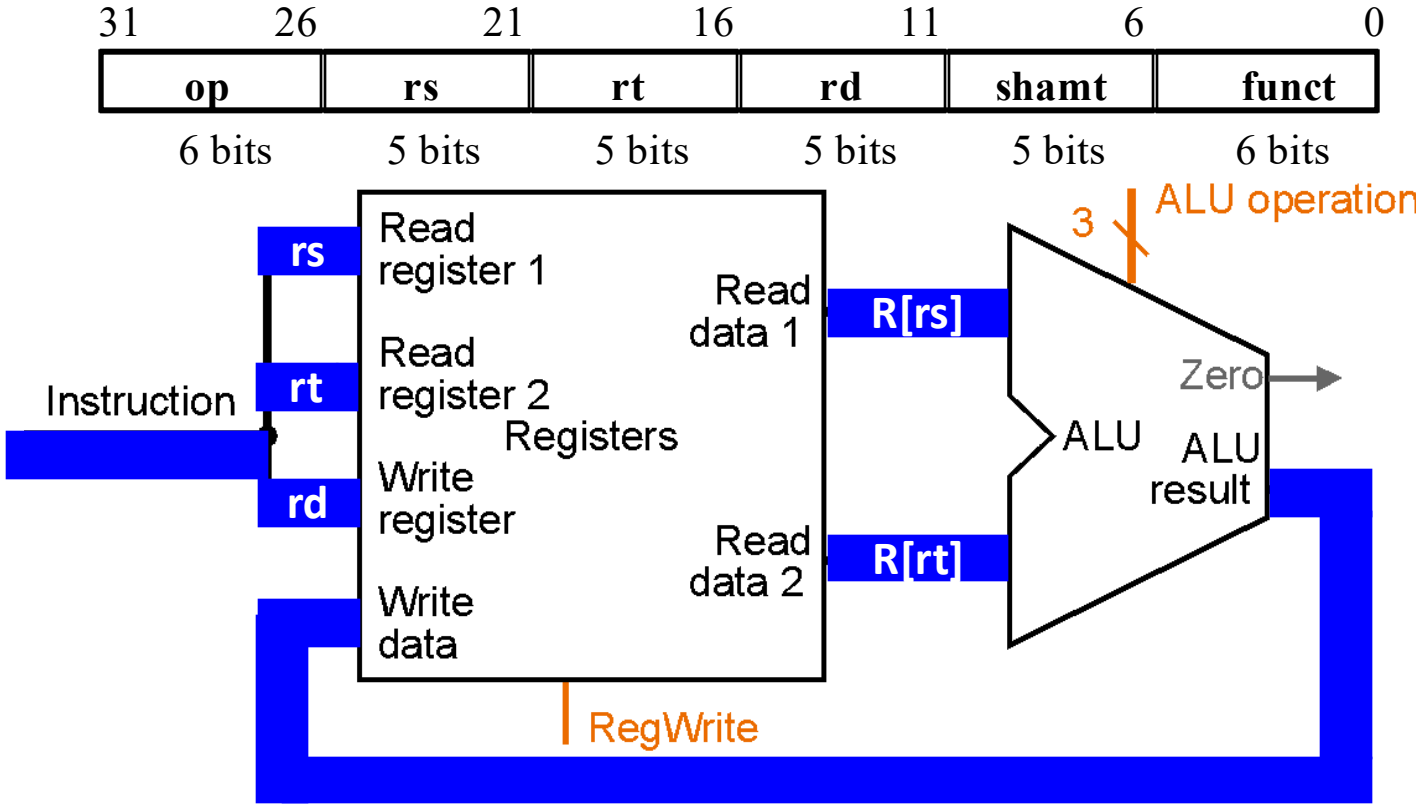
Overview of the Instruction Fetch Unit

- 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

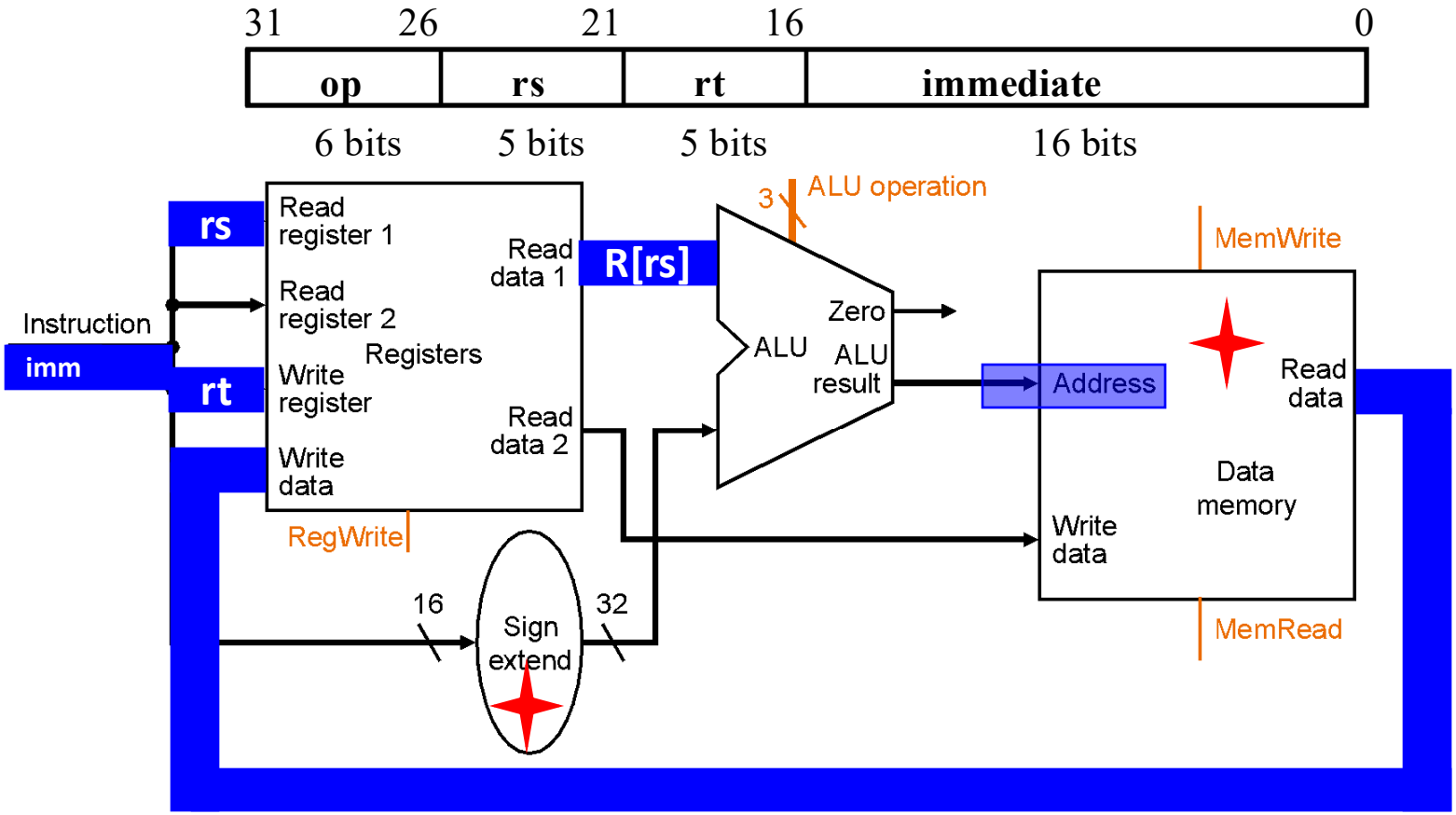
- R[rd] <- R[rs] op R[rt] Example: *add rd, rs, rt*
  - RR1, RR2, and WR comes from instruction's rs, rt, and rd fields
  - ALUoperation and RegWrite: control logic after decoding instruction



Datapath for Load Operations

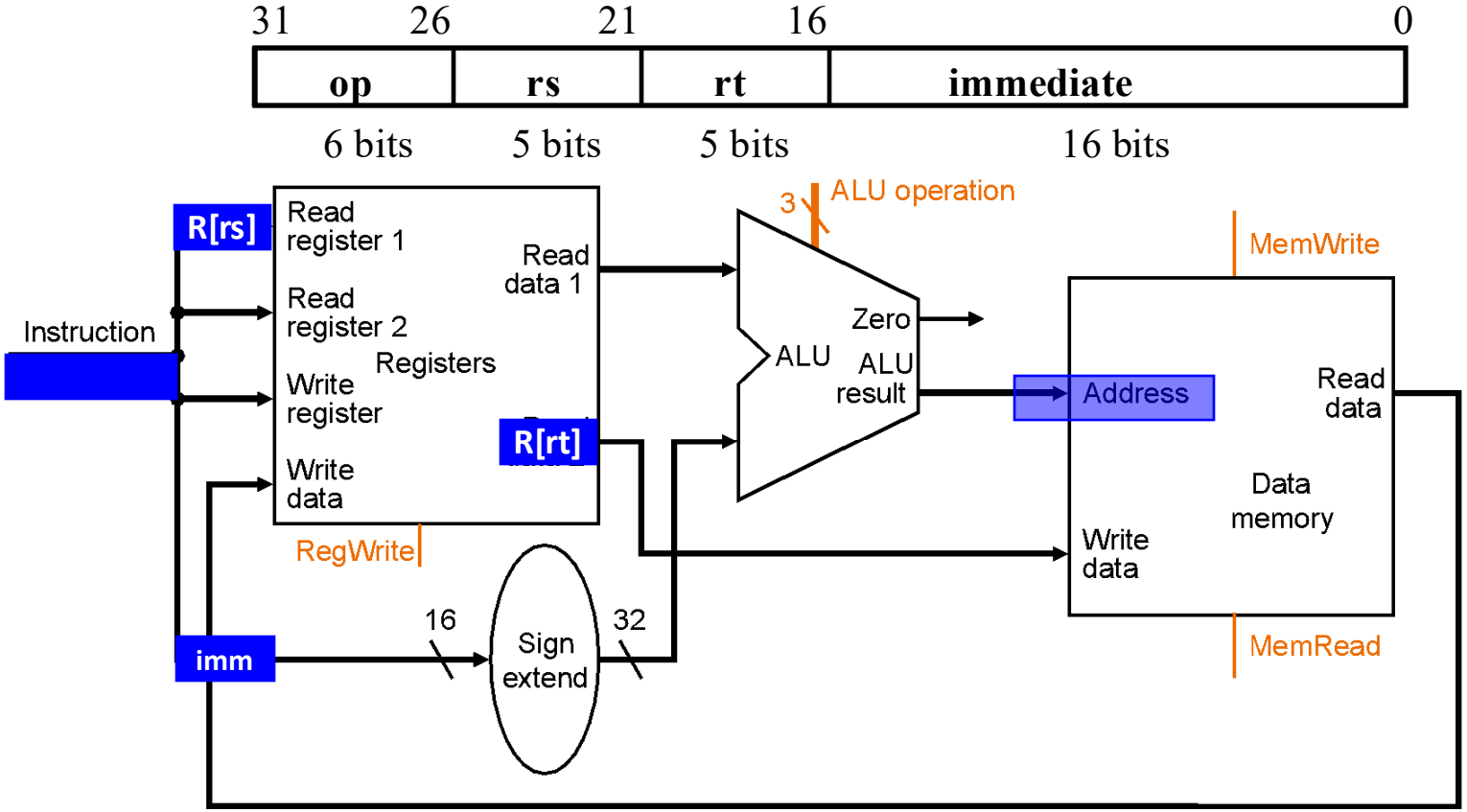
- R[rt] <- Mem[R[rs] + SignExt[imm16]]
  - Example: *lw rt, rs, imm16*

这是一种变址寻址

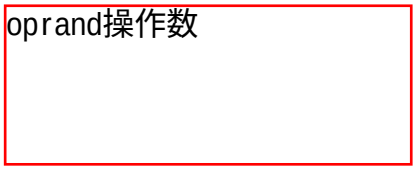


Datapath for Store Operations

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



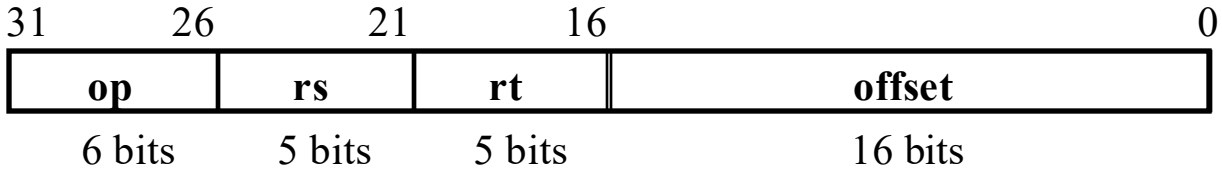
Datapath for Branch Operations



- 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

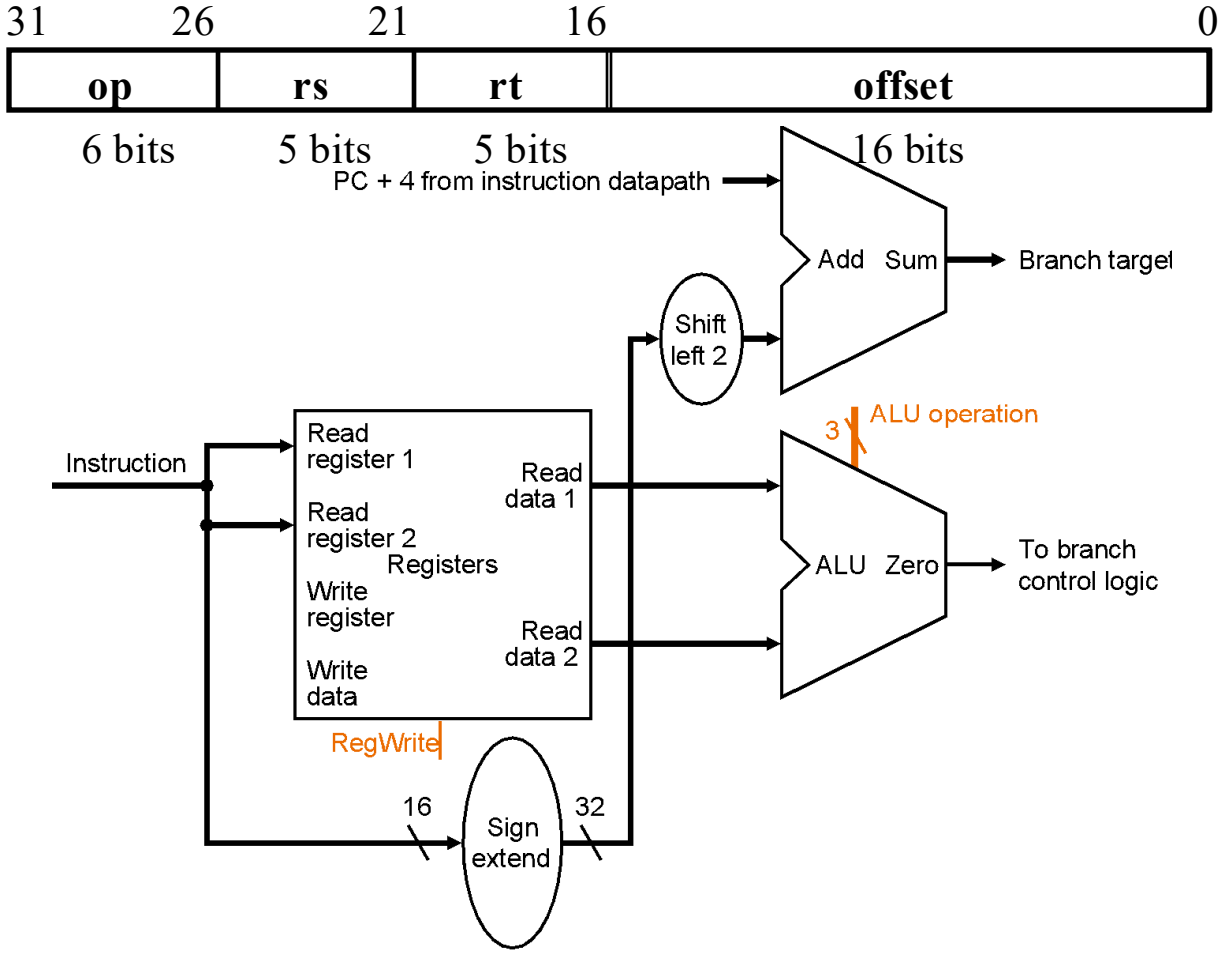
Datapath for Branch Operations

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



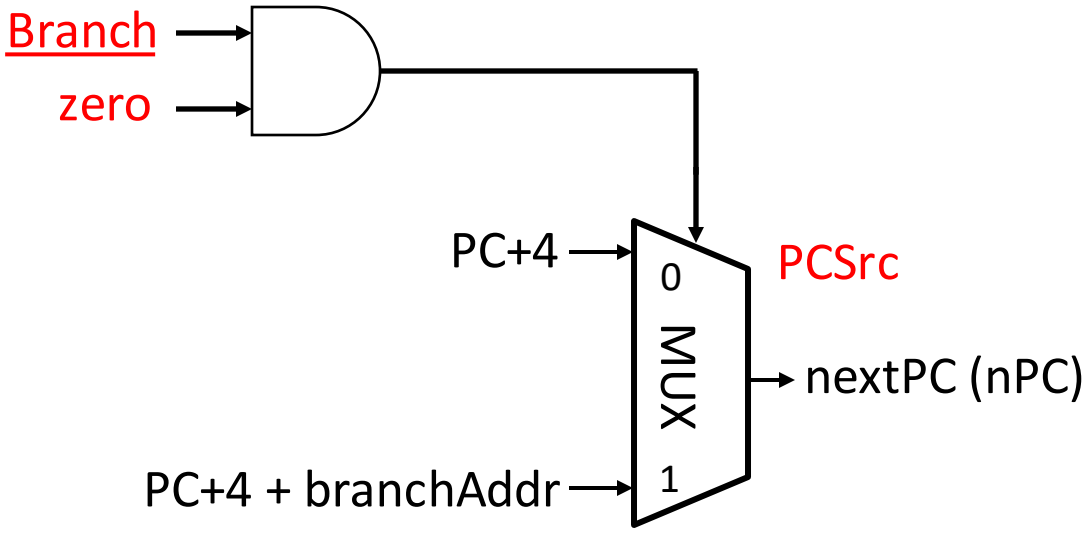
Datapath for Branch Operations

- 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

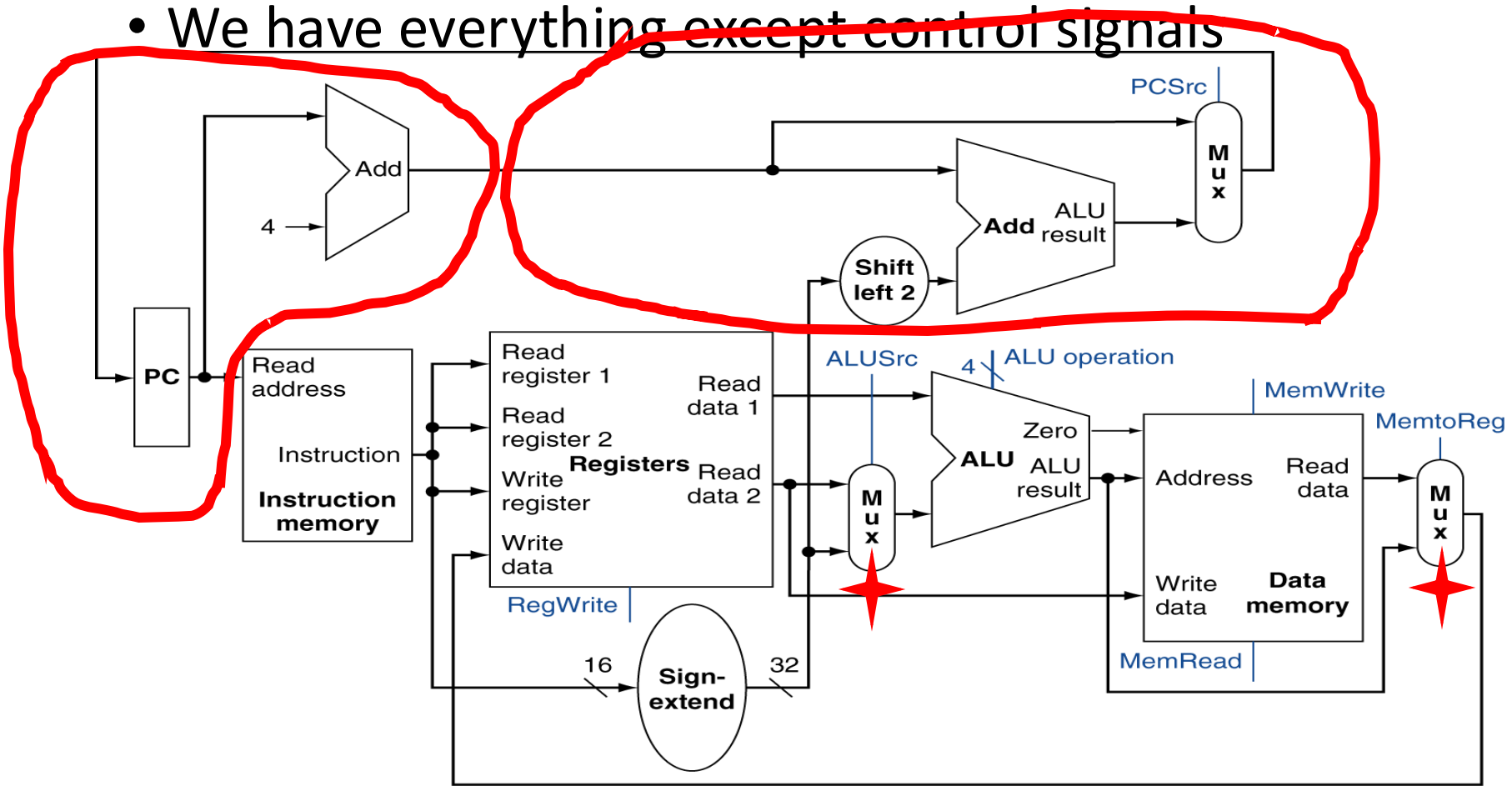
How does this change if we add `bne`*?*

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

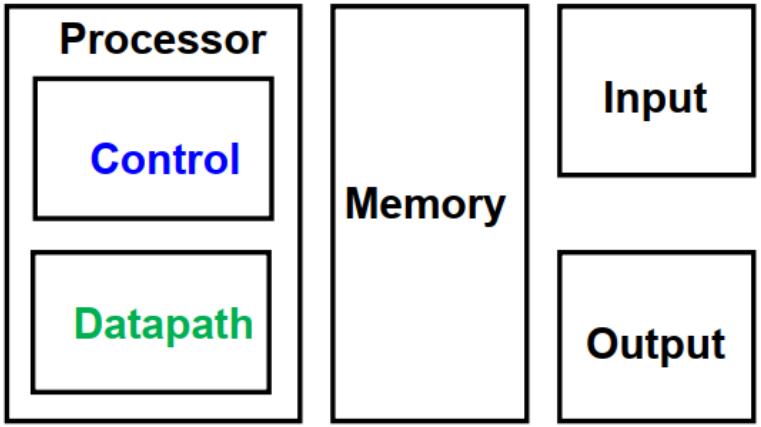


Key Points

- 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?

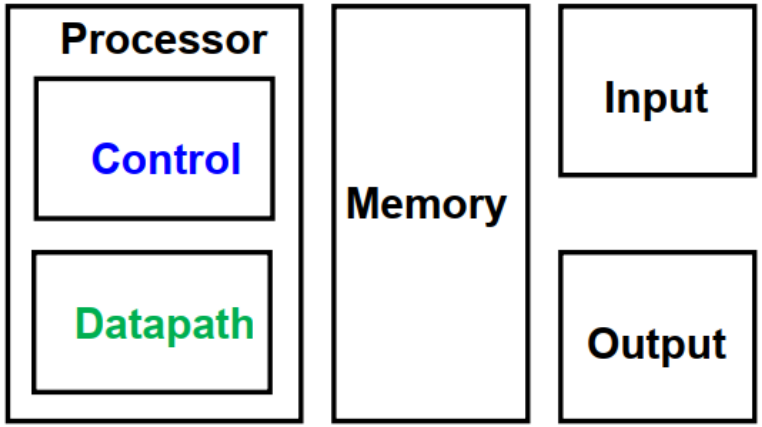
The Big Picture: Where are we now

- Five classic components:



The Big Picture: Where are we now

- Five classic components:



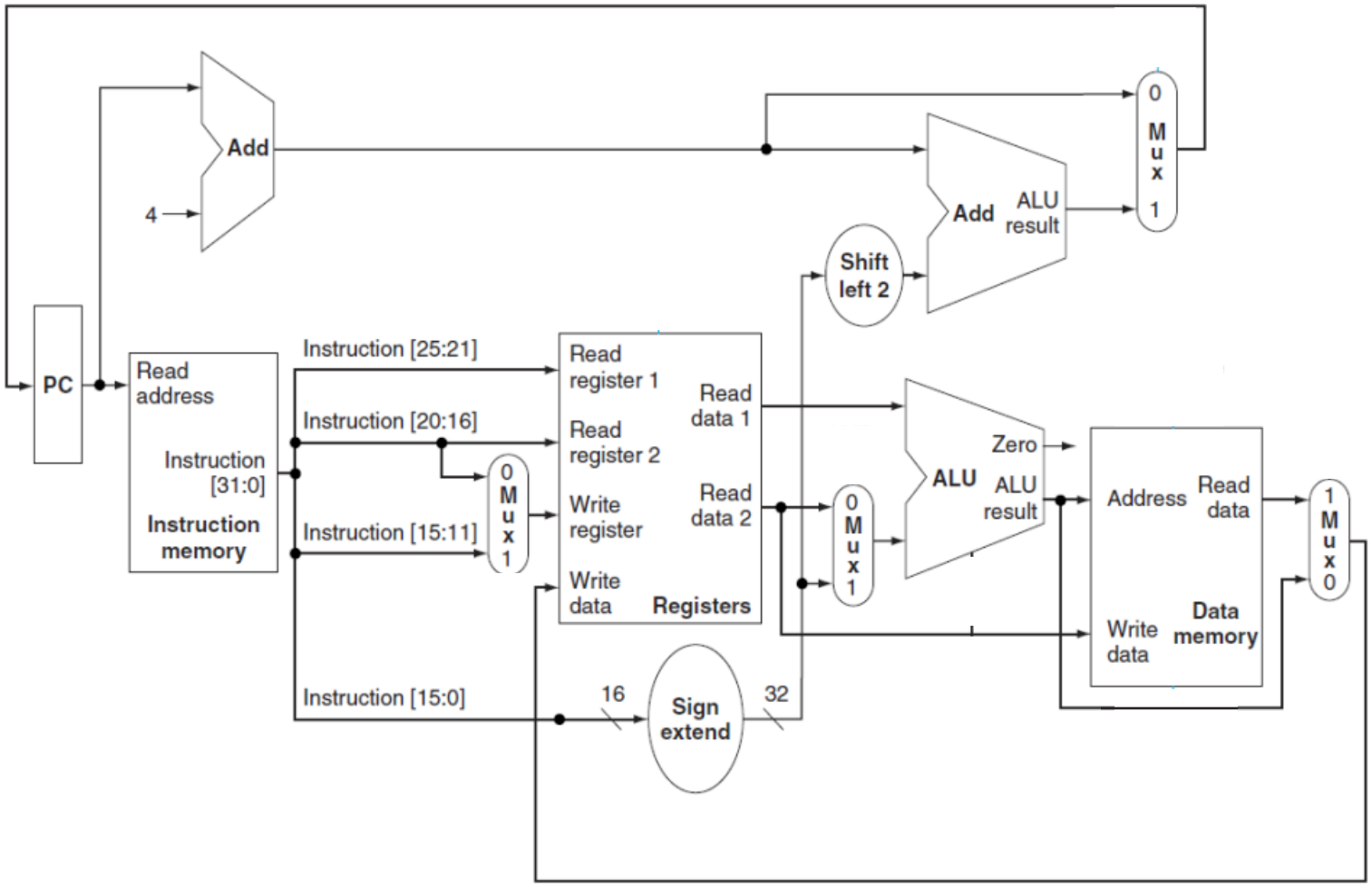
- Our target now: design the control logic

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

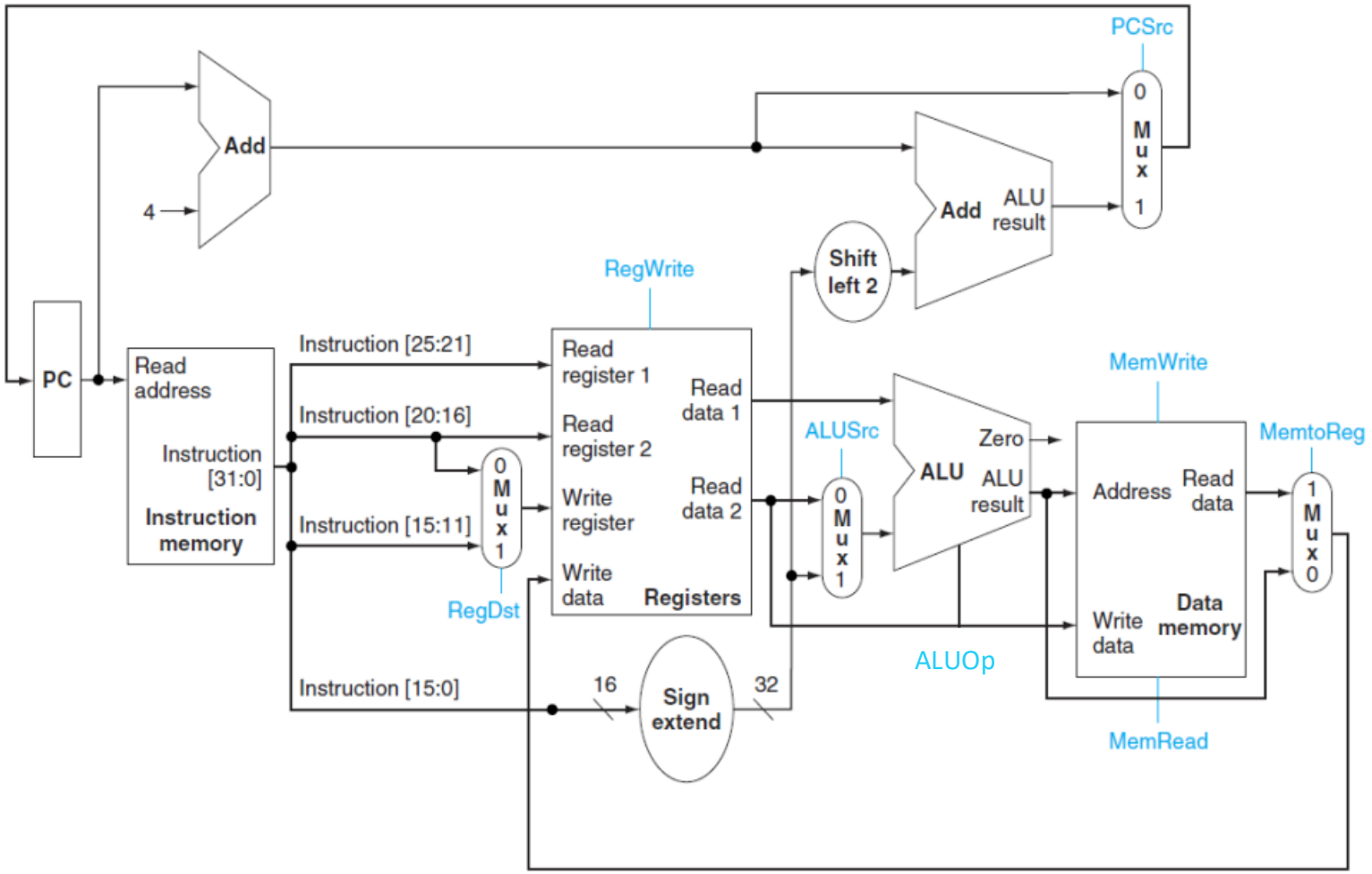
A Single Cycle Datapath

- We have everything except control signals

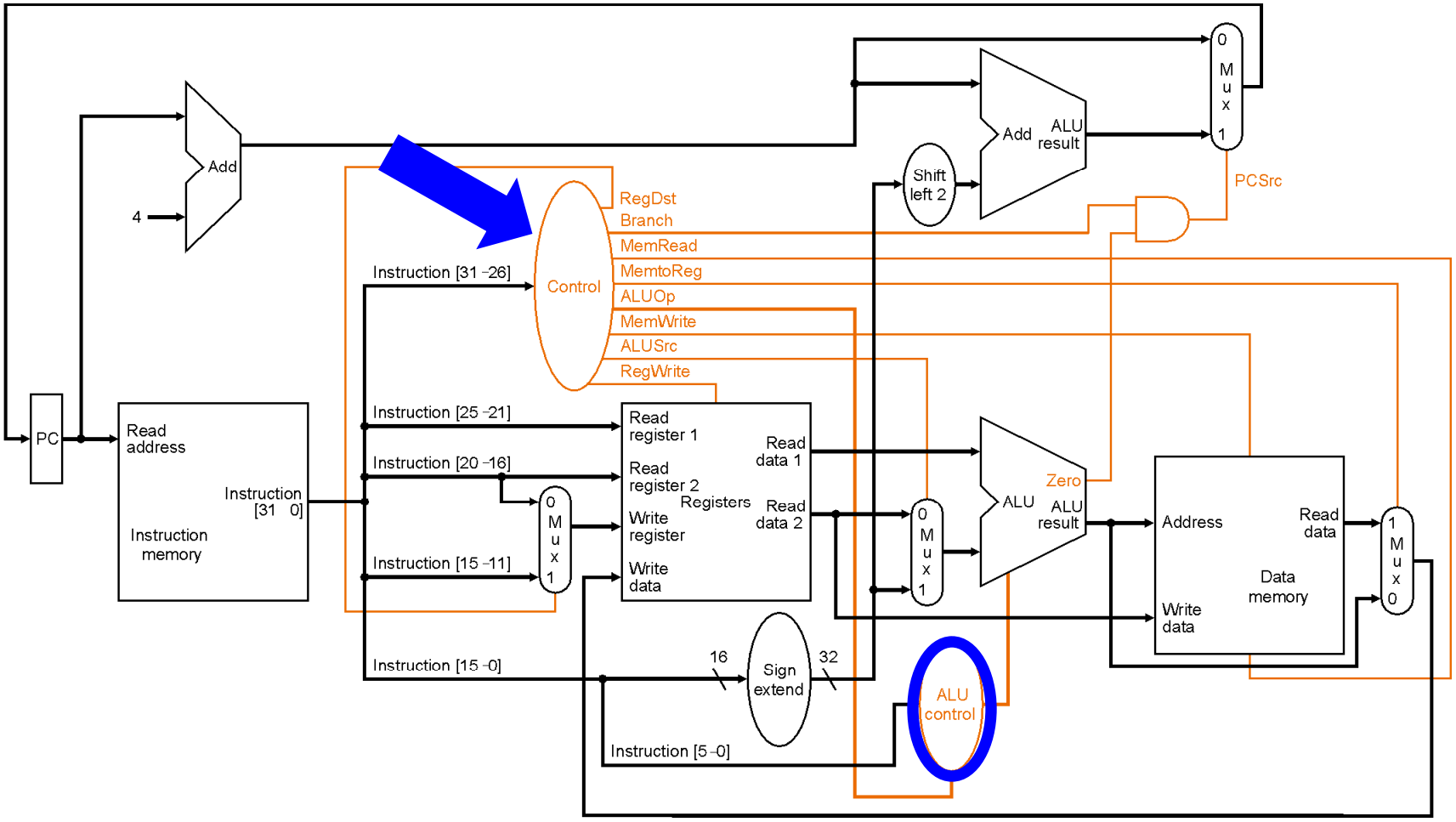


A Single Cycle Datapath

- We have everything except control signals



Okay, then, what about those Control Signals?



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 III

CPU控制部件、时钟及性能分析

主讲教师：石 侃
shikan@ict.ac.cn

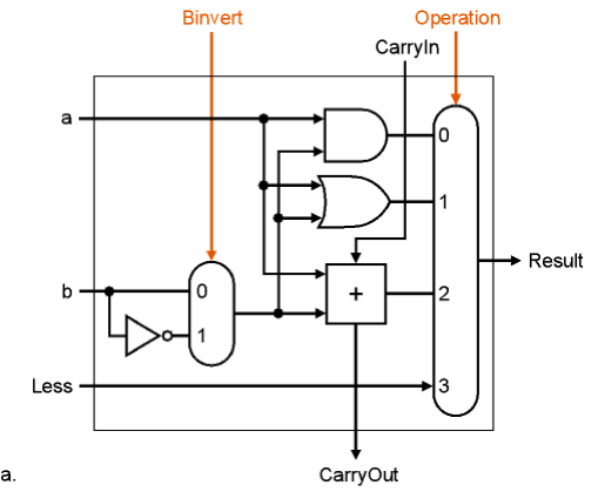
2025年4月18日

ALU control bits

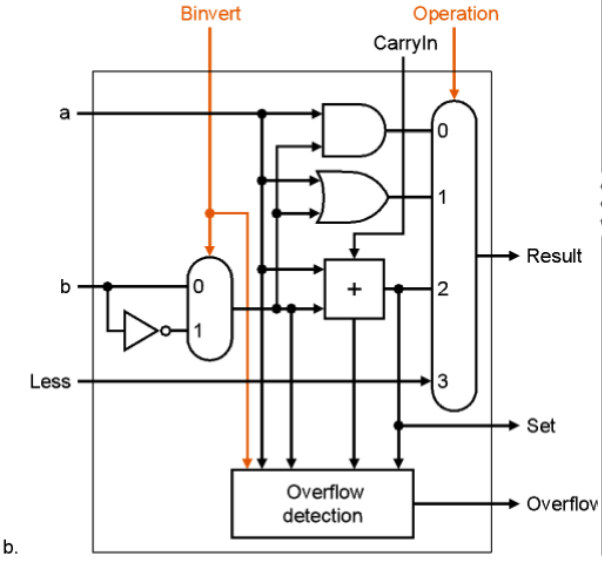
- 5-Function ALU

ALU control input	Function	Operations
000	And	and
001	Or	or
010	Add	add, lw, sw
110	Subtract	sub, beq
111	Slt	slt

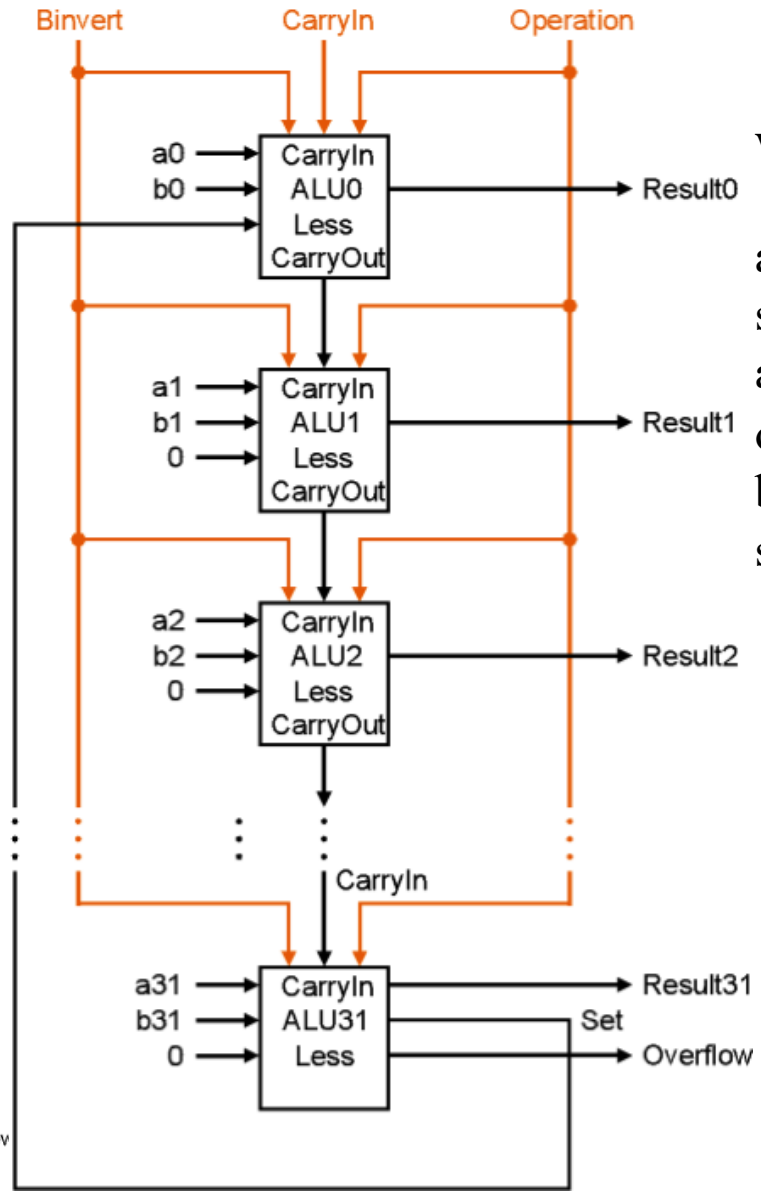
Full ALU



a.

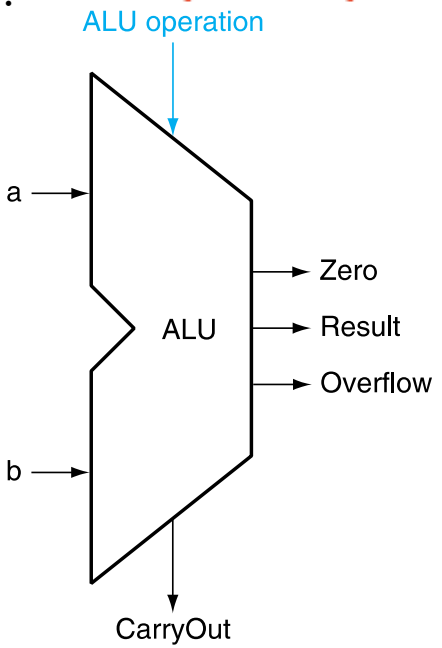


b.



what signals accomplish:

	Binvert	CIn	Oper
add?	0	0	10
sub?	1	1	10
and?	0	0	00
or?	0	0	01
beq?	1	1	10
slt?	1	1	11

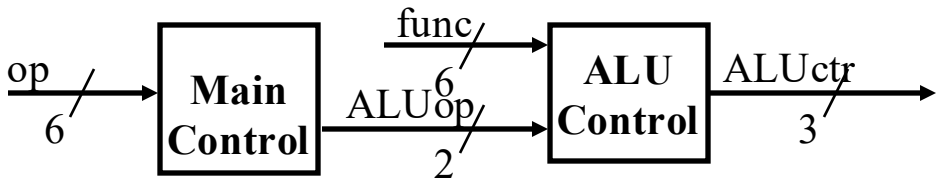


ALU control bits

- 5-Function ALU

ALU control input	Function	Operations
000	And	and
001	Or	or
010	Add	add, lw, sw
110	Subtract	sub, beq
111	Slt	slt

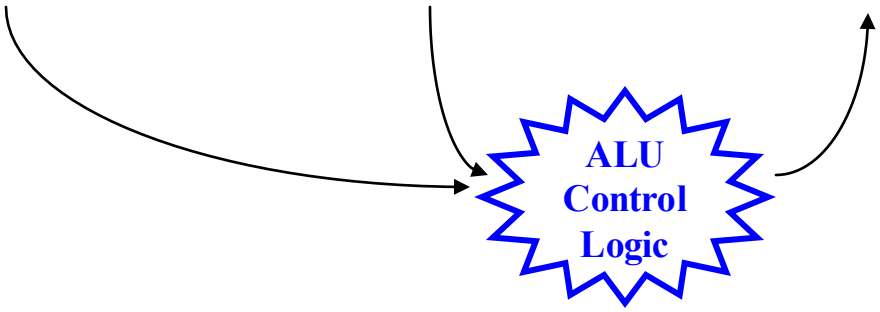
- MIPS: based on opcode (bits 31-26) and function code (bits 5-0) from instruction
- RISC-V: based on func7, func3, and opcode from instruction
- ALU doesn't need to know all opcodes--we will summarize opcode with ALUOp (2 bits):
  - **00 - lw,sw** **01 – beq** **10 - R-format**



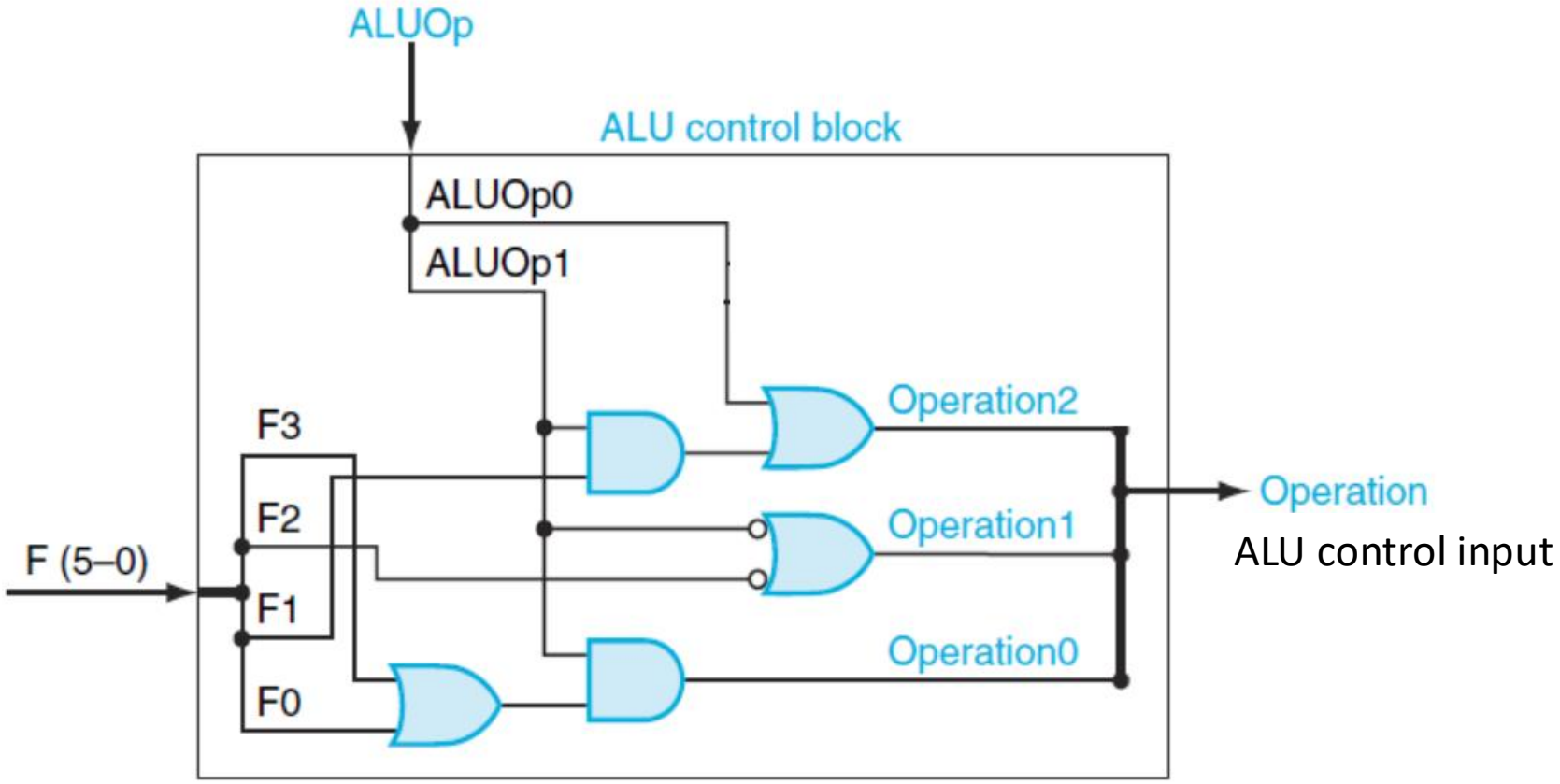
多层解码：减小主控单元规模，提高控制速度

Generating ALU Control

Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control input
lw	00	load word	xxxxxx	add	010
sw	00	store word	xxxxxx	add	010
beq	01	branch eq	xxxxxx	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	slt	101010	slt	111

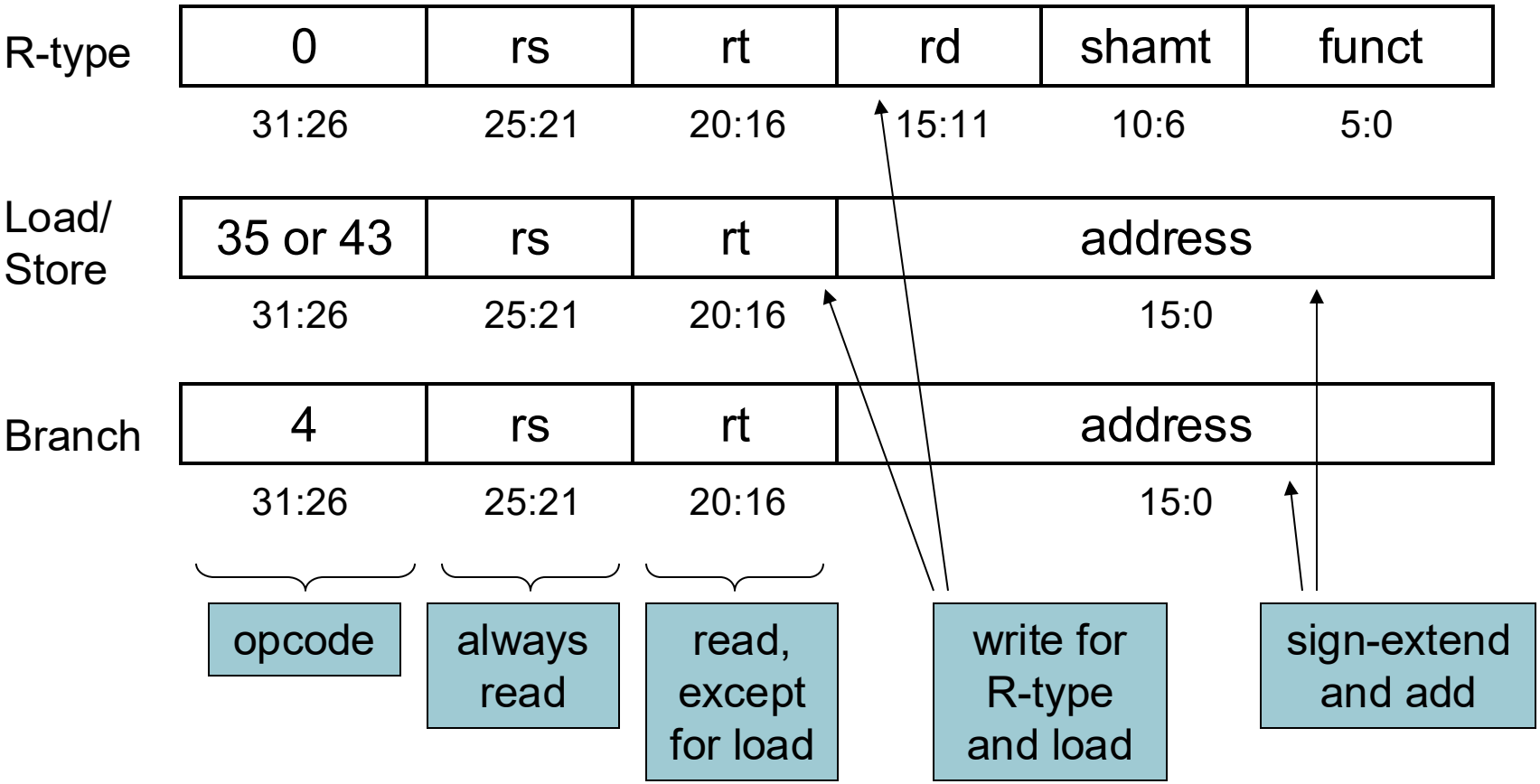


ALU Control Logic

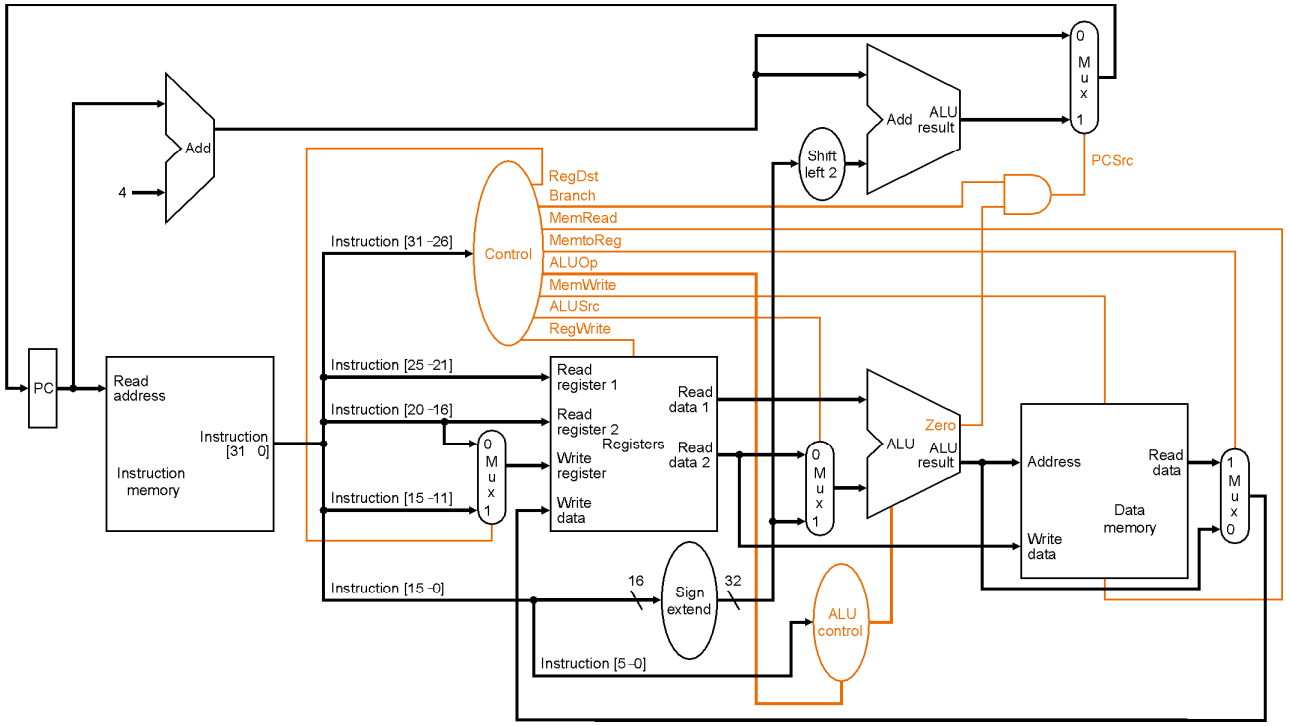


The Main Control Unit

- Control signals derived from instruction



Controlling the CPU



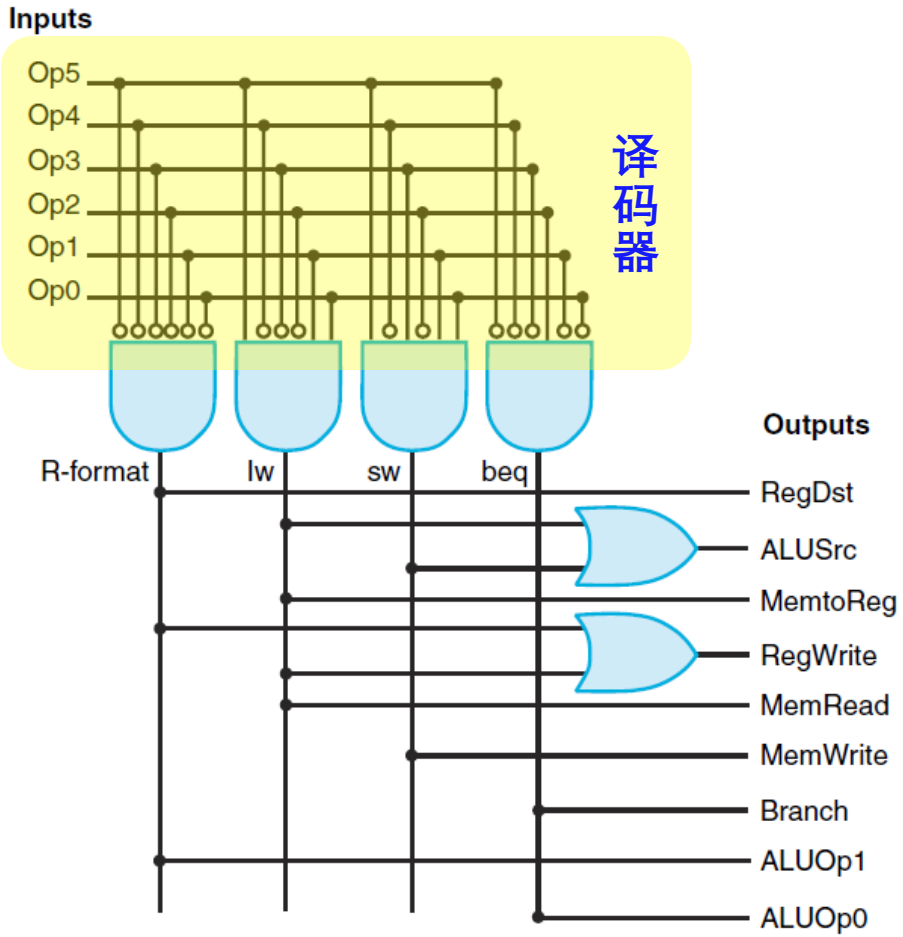
Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Control Truth Table

		R-format	lw	sw	beq
Opcode		000000	100011	101011	000100
Outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

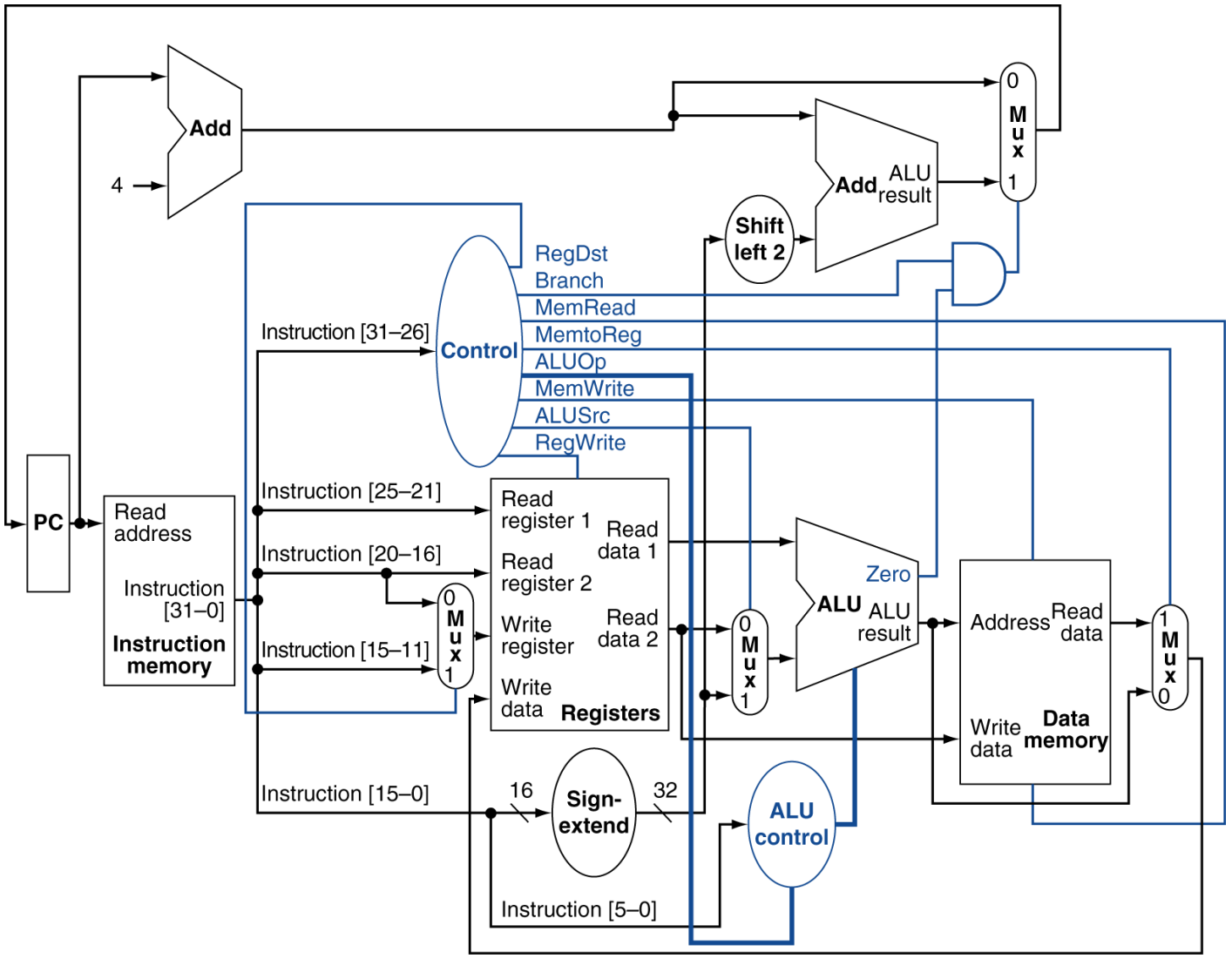
Control

- Simple combinational logic (truth tables)

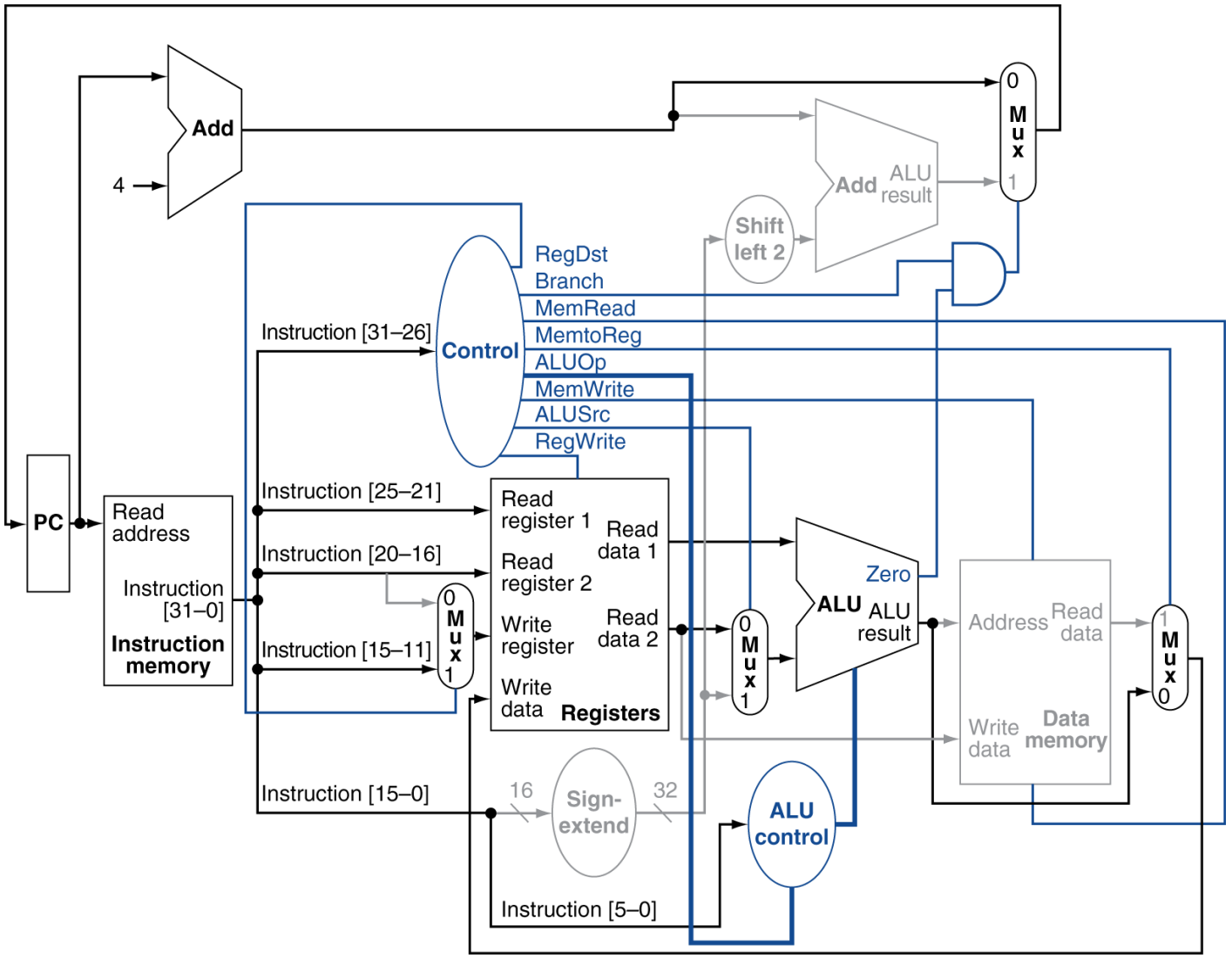


可编程逻辑阵列(Programmable Logic Arrary,PLA)

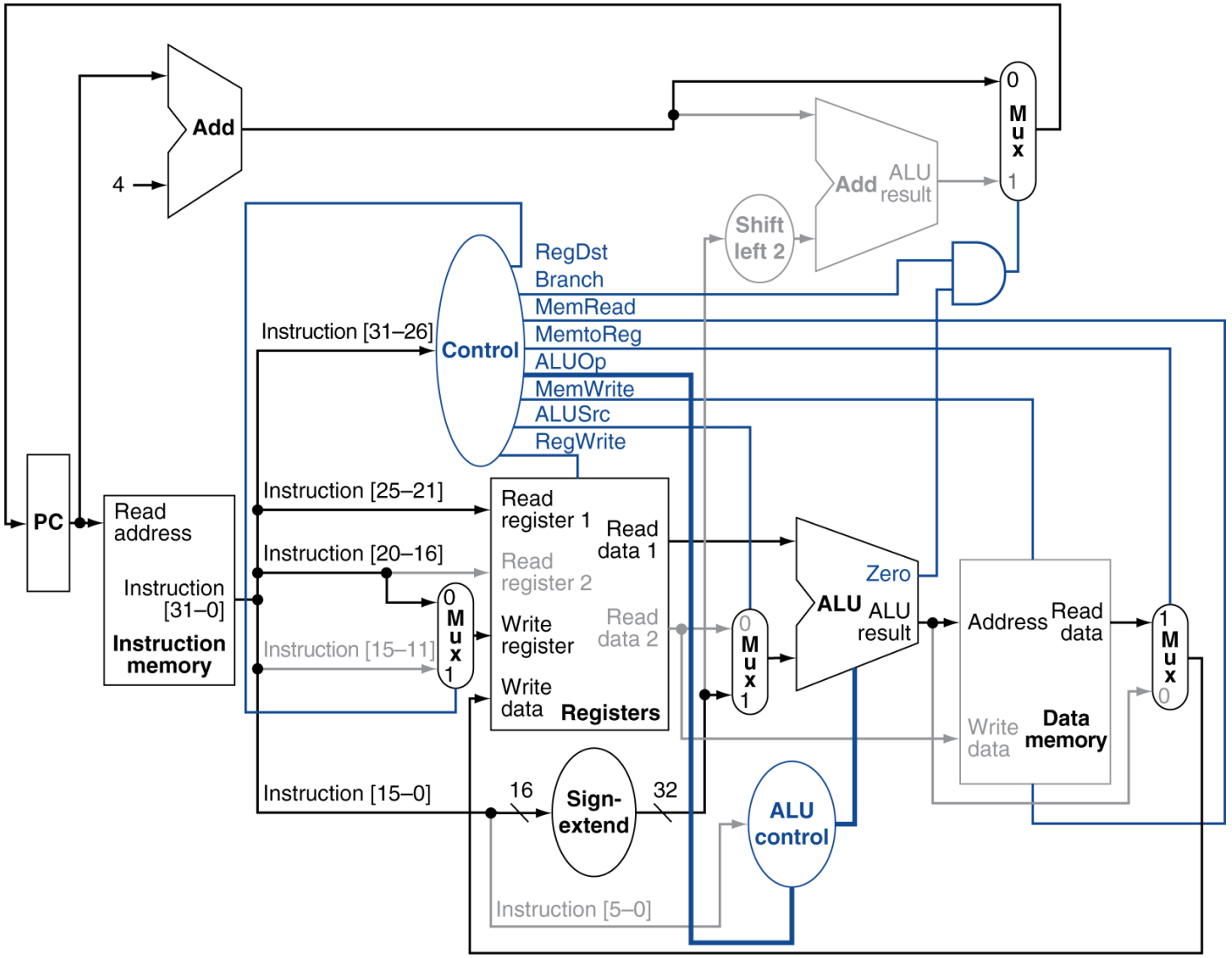
Datapath With Control



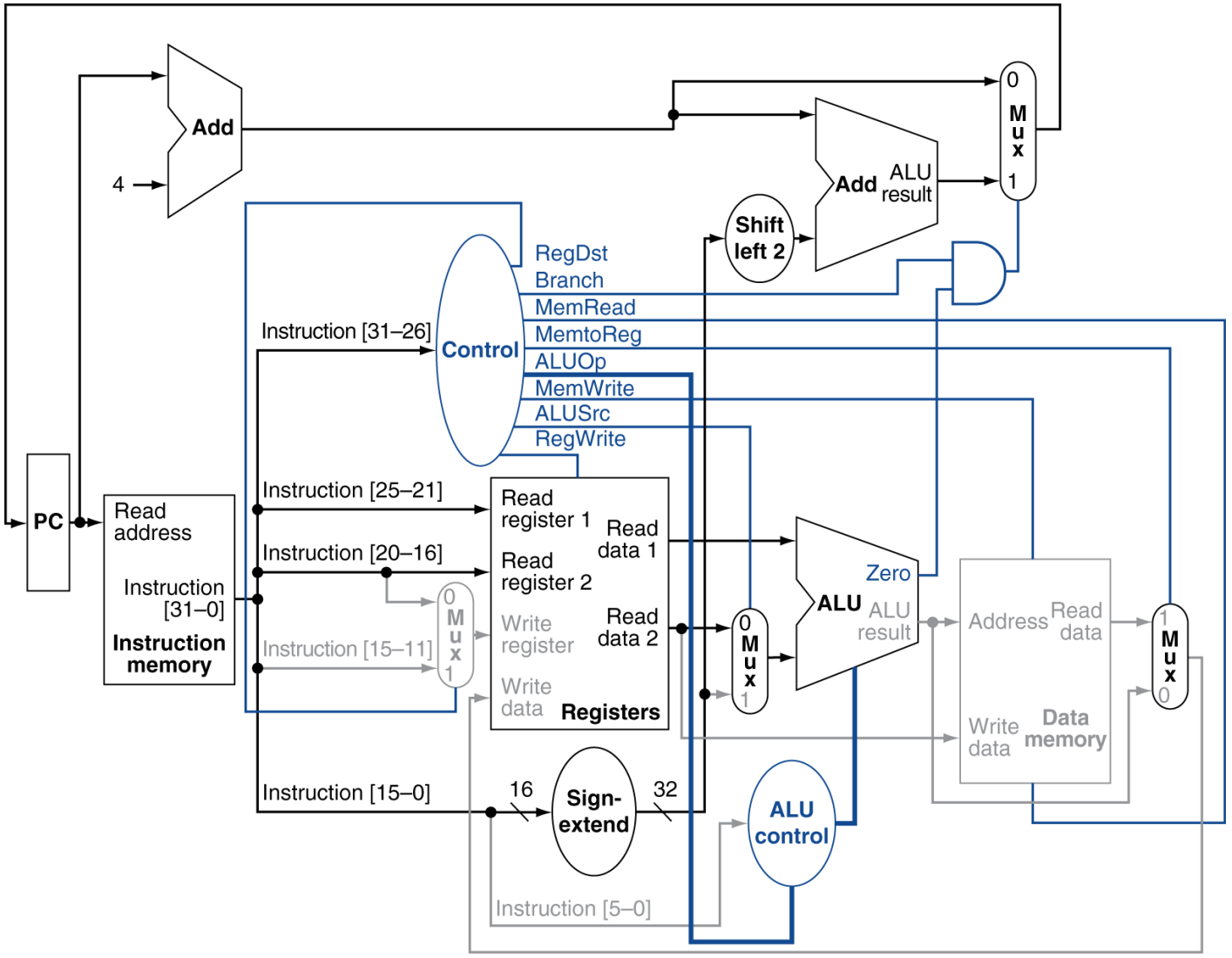
R-Type Instruction



Load Instruction

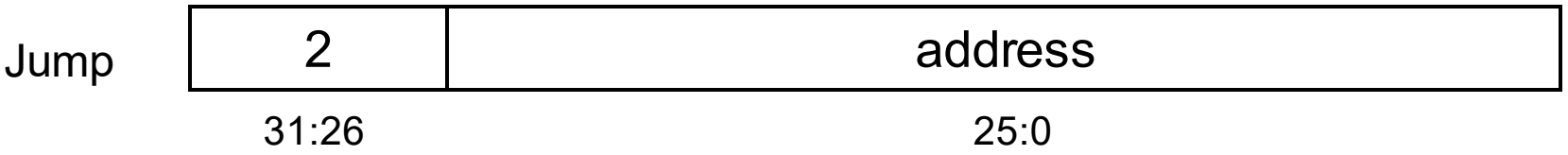


Branch-on-Equal Instruction

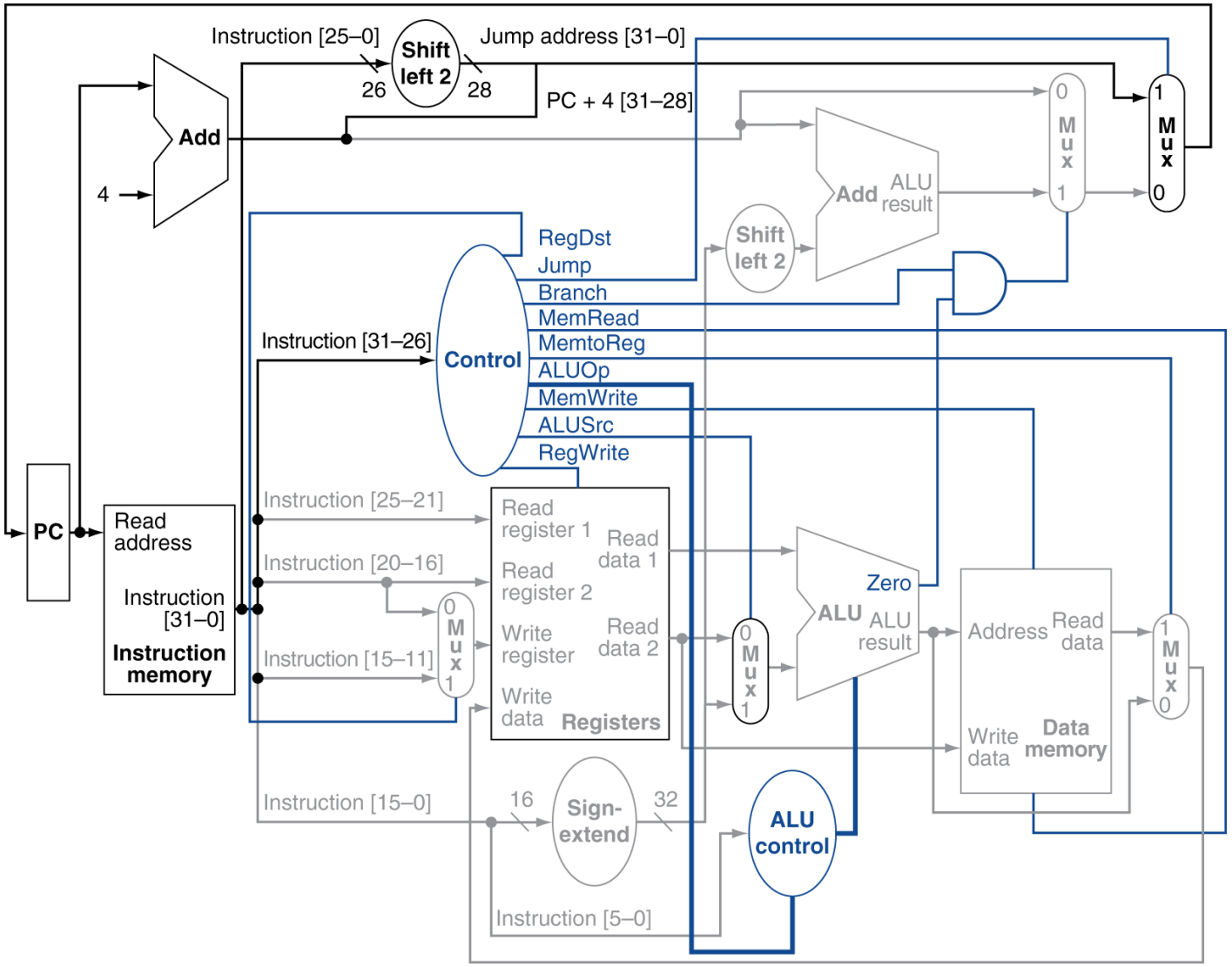


Implementing Jumps

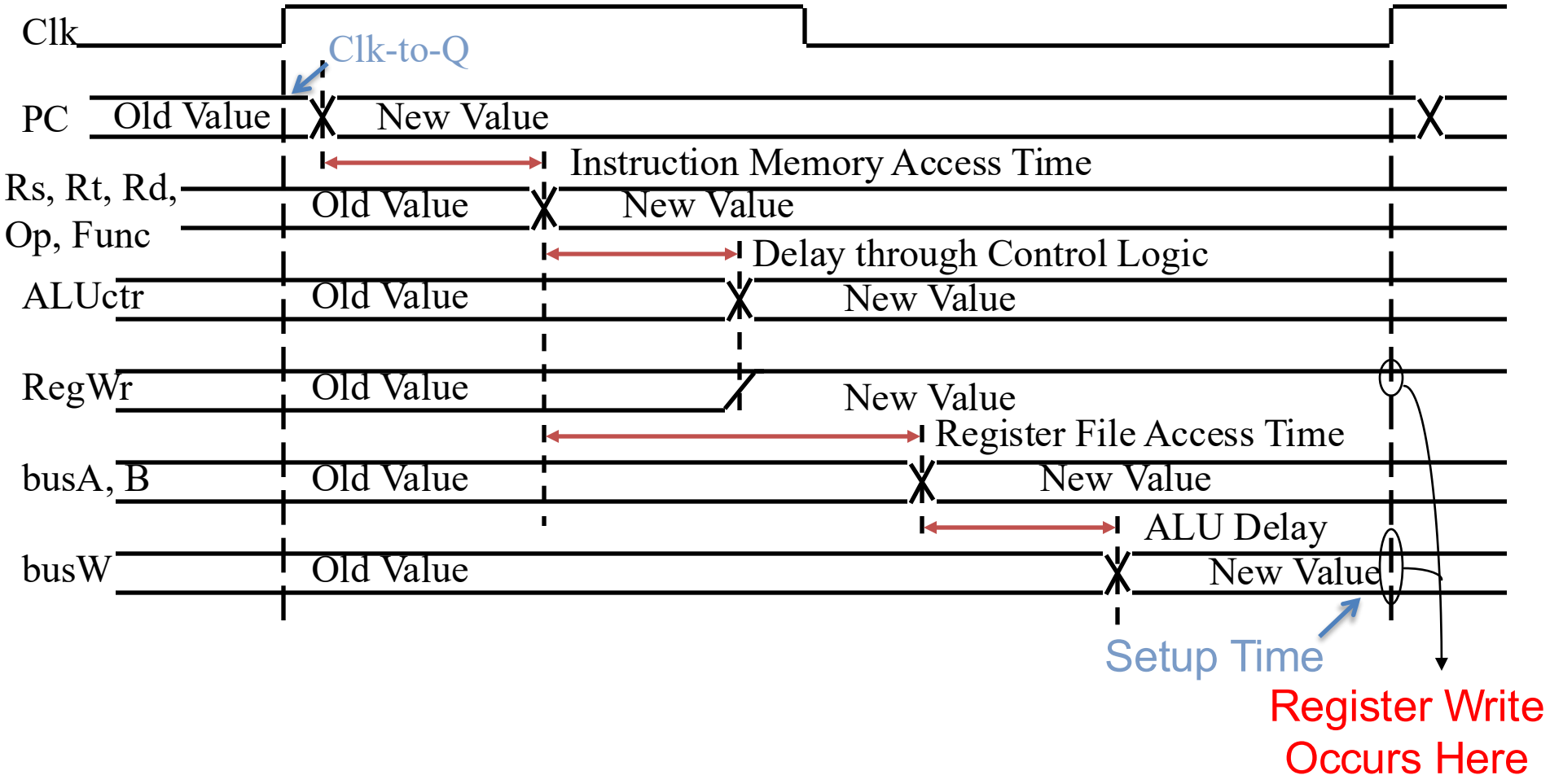
- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of (PC+4)
  - 26-bit jump address “inst_index”
  - 2’b00
- Need an extra control signal decoded from opcode



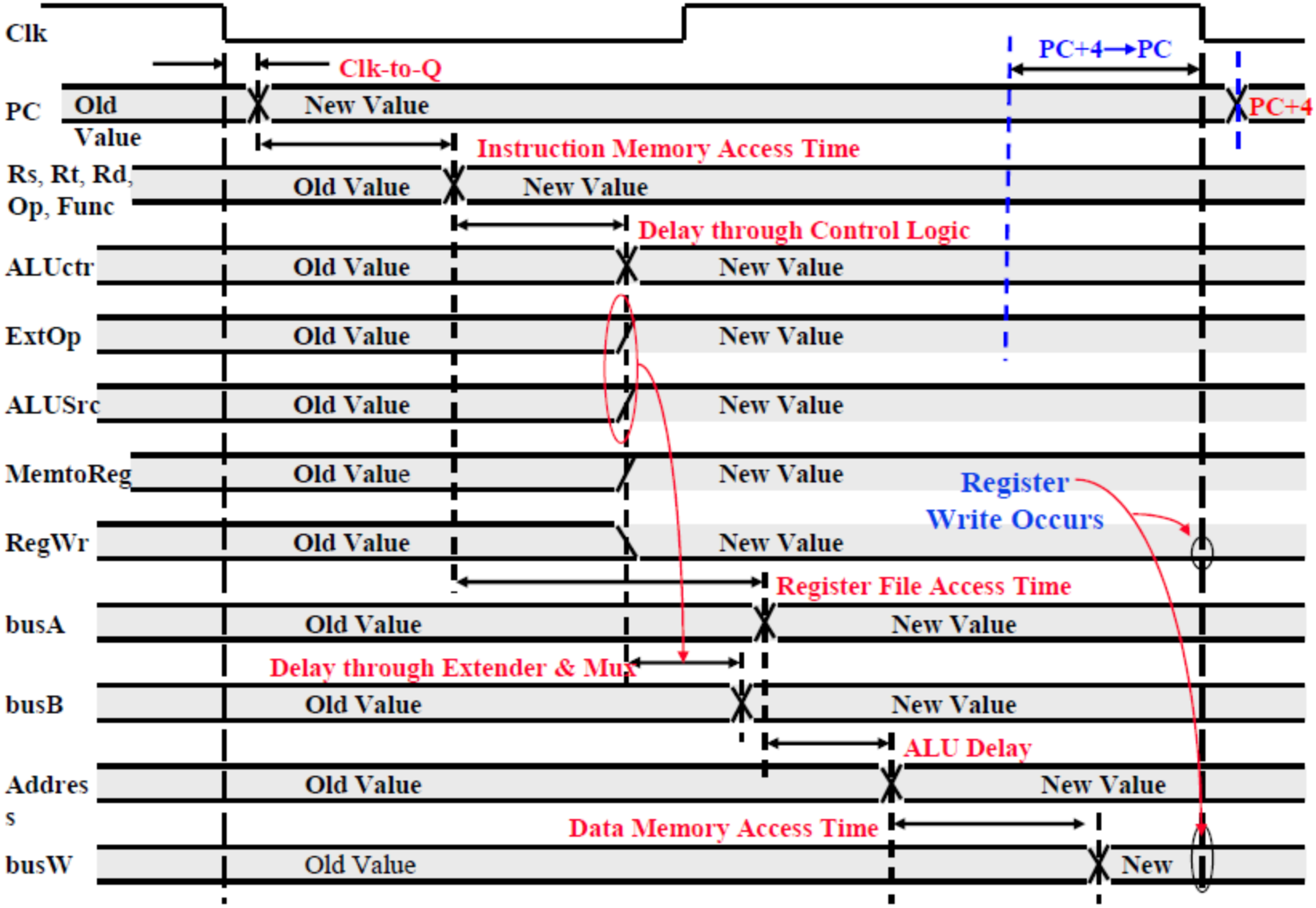
Datapath With Jumps Added



Register-Register Timing: One Complete Cycle for add



lw指令的执行时间最长，它所花时间作为时钟周期



单周期计算机的性能

- 单周期处理器的CPI为多少？ **CPI=1!**
  - 其他条件一定的情况下，CPI越小，则性能越好！
  - CPI=1，不是很好吗？
- 单周期处理器的性能会不会很好？为什么？
  - 计算机的性能除CPI外，还取决于时钟周期的宽度
  - 单周期处理器的时钟宽度为最复杂指令的执行时间
  - 很多指令可以在更短的时间内完成
    - 非load/store指令无需访问数据存储器
    - J-指令无需访问寄存器

单周期计算机的性能

假设在单周期处理器中，各主要功能单元的操作时间为：

- 存储单元：200ps
- ALU和加法器：100ps
- 寄存器堆（读/写）：50ps

假设MUX、控制单元、PC、扩展器和传输线路都没有延迟，则下面实现方式中，哪个更快？快多少？

（1）每条指令在一个固定长度的时钟周期内完成

（2）每条指令在一个时钟周期内完成，但时钟周期仅为指令所需，也即为可变的（实际不可行，只是为了比较）

假设程序中各类指令占比：25%取数、10%存数、45%ALU、15%分支、5%跳转

单周期计算机的性能

解：CPU执行时间=指令条数 x CPI x 时钟周期=指令条数 x 时钟周期

两种方案的指令条数都一样，CPI都为1，所以只要比较时钟周期宽度即可。

Instruction class	Functional units used by the instruction class				
R-type	Instruction fetch	Register access	ALU	Register access	
Load word	Instruction fetch	Register access	ALU	Memory access	Register access
Store word	Instruction fetch	Register access	ALU	Memory access	
Branch	Instruction fetch	Register access	ALU		
Jump	Instruction fetch				

各类指令要求的时间长度为：

Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200		550 ps
Branch	200	50	100	0		350 ps
Jump	200					200 ps

单周期计算机的性能

对于方式（1），时钟周期由最长指令来决定，应该是load指令，为600ps

对于方式（2），时钟周期取各条指令所需时间，时钟周期从600ps至200ps

根据各类指令的频度，计算出平均时钟周期长度为：

CPU时钟周期

=600x25%+550x10%+400x45%+350x15%+200x5%=447.5ps

$$\text{CPU性能比}=\frac{\text{方式(1)的CPU执行时间}}{\text{方式(2)的CPU执行时间}}=\frac{\text{方式(1)的CPU 时钟周期}}{\text{方式(2)的CPU 时钟周期}}=\frac{600}{447.5}=1.34$$

由此可见，可变时钟周期的性能是定长周期的1.34倍！

但是，对每类指令采用可变长时钟周期实现非常困难，而且所带来的额外开销会很大，不合算！

早期的小指令集计算机用过单周期实现技术，但现代计算机都不采用。

下一讲介绍多周期数据通路和控制器，其特点是：

时钟周期固定、时钟周期数可变

单周期CPU小结

- 考察每条指令在单周期数据通路中的执行过程
  - 每条指令在一个时钟周期内完成
  - 每个时钟到来时，都开始进入取指令操作
    - 经过CLK-to-Q，PC得到新值，经过access time后得到当前指令
    - 按三种方式分别计算下条指令地址，在branch / zero / jump的控制下，选择其中之一送到PC输入端，但不会马上写到PC中，一直到下个时钟到达时，才会更新PC。三种下址方式为：
      - **branch=jump=0：PC+4**
      - **branch=zero=1： PC+4+SignEXT(imm16)*4**
      - **jump=1： (PC+4)[31,28]+imm26*4**
  - 指令取出后被译码，产生指令对应的控制信号
    - R-型指令：rd为目的寄存器，无访存操作，……
    - lw指令： rt为目的寄存器，符号扩展，计算地址、读存储器，……
    - sw指令： rt为源寄存器，符号扩展，计算地址、写存储器，……
- 汇总每条指令控制信号的取值，生成真值表，写出逻辑表达式，设计控制器逻辑

处理器的设计步骤

ISA确定后，做处理器设计的大致步骤：

- 分析每条指令的功能，并用RTL表示
- 根据指令的功能选择所需组件，考虑时钟等方案
- 将模块组合成数据通路
- 确定每个组件所需的控制信号及取值，汇总所有指令涉及到的控制信号，做一个体现“指令-控制信号”关系的表
- 根据表得到每个控制信号的逻辑表达式

练习：

对于课上提到的各种指令，在单周期处理器结构图上，画出不同指令在执行过程中依次激发的数据通路及控制信号