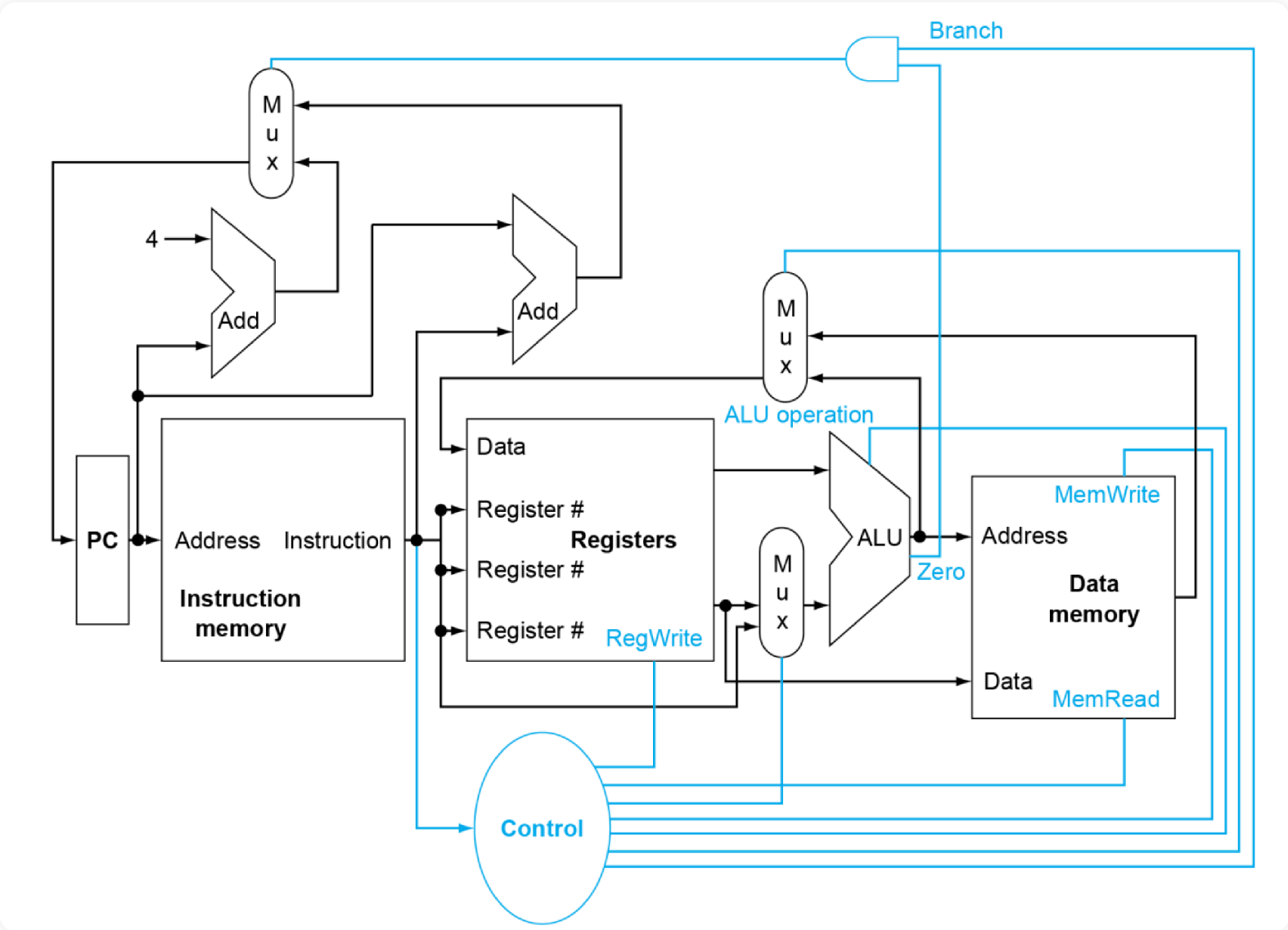
CO 04 The Processor Pt1

1 Introduction



- 在指令进入寄存器模块前，不需要进行解码，因为 RISC-V 中不同指令的 `rd, rs1, rs2` 的相对位置完全一致，*除了立即数，本图缺少了 `ImmGen` 模块*。
- `Contorl` 信号控制所有的 MUX，以及模块的使能信号
- CPUTime 取决于频率和 CPI，Cycle 小可能导致 CPI 增大

2 Datapath

Instruction execution in RISC-V

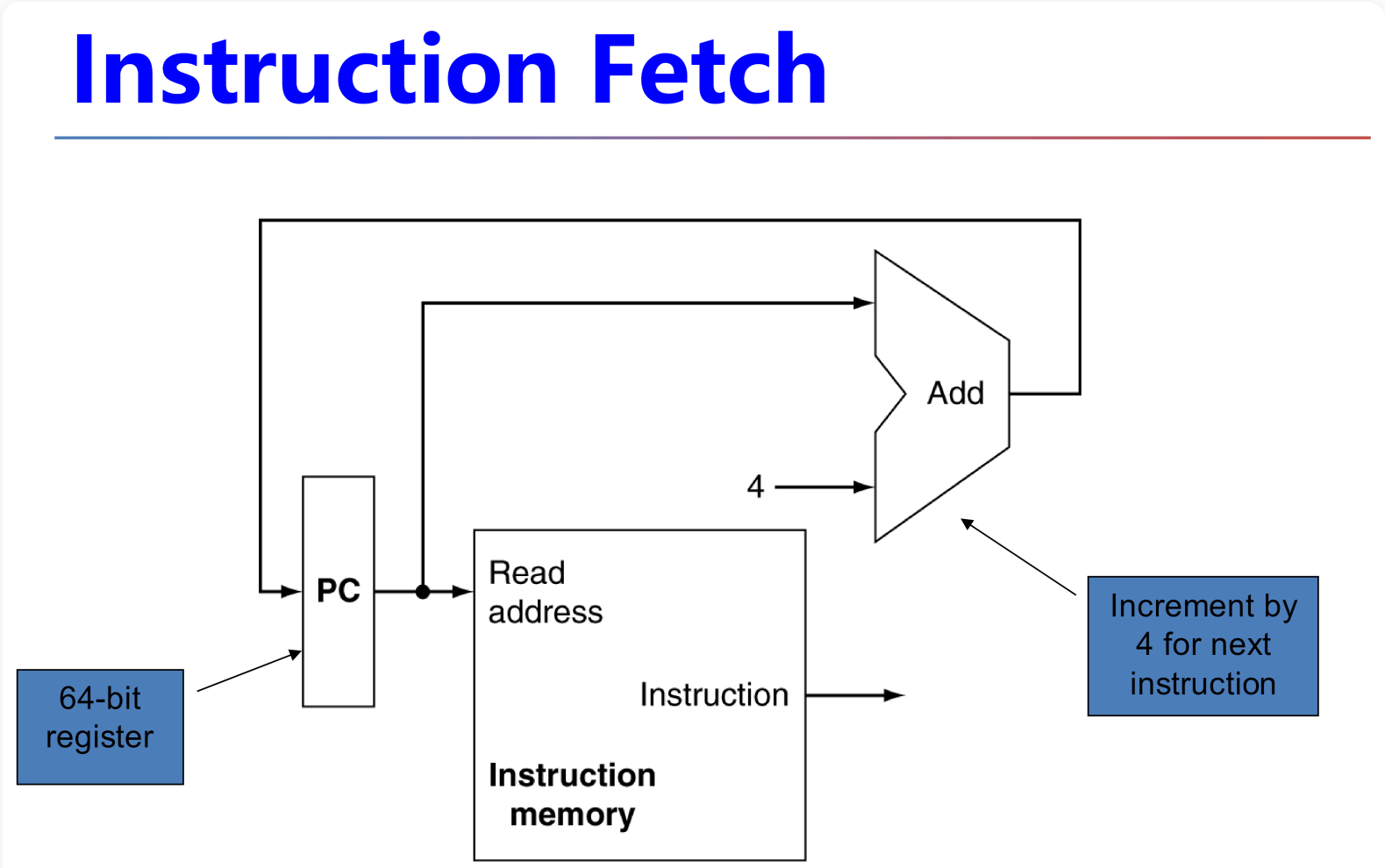
- **Fetch :**
  - Take instructions from the instruction memory
  - Modify PC to point to the next instruction
- **Instruction decoding & Read Operand:**
  - Will be translated into machine control command
  - Reading Register Operands, whether or not to use
  - Reading Register Operands, whether or not to use
- **Executive Control:**
  - Control the implementation of the corresponding ALU operation
- **Memory access:**
  - Write or Read data from memory
  - Only ld/sd
- **Write results to register:**
  - If it is R-type instructions, ALU results are written to rd
  - If it is I-type instructions, memory data are written to rd
- **Modify PC** for branch instructions

- Modify PC 是旁路，剩下五个是标准步骤，**明确某种指令对应的最小需要单元**

- Fetch inst mem
- Inst decoding & read operand，执行速度快，同步进行
- Executive Control, ALU operations
- Mem access，不是所有指令都需要访存，`ld/sd`
- Write to reg, R/I

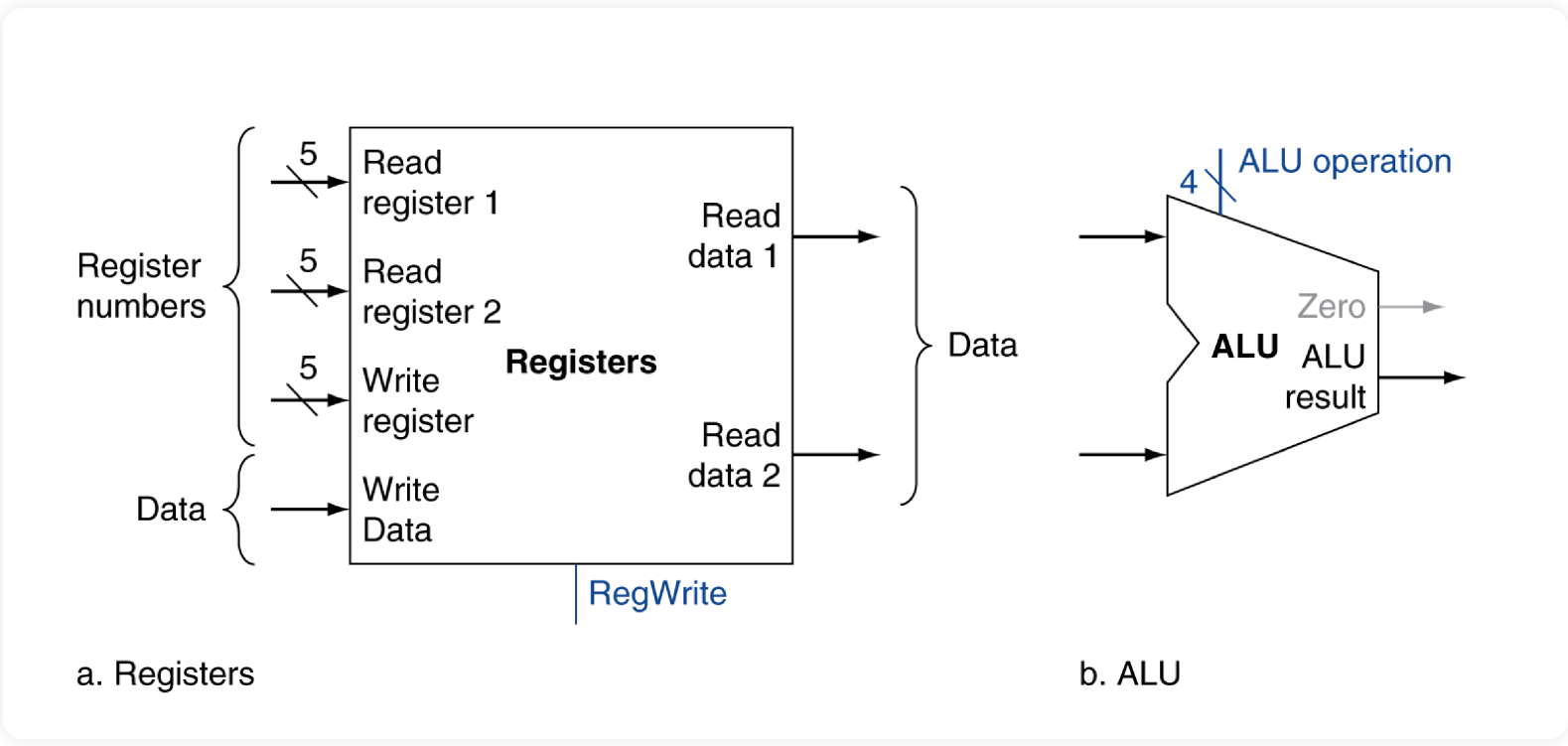
2.1 不同指令使用的模块

2.1.1 Fetch



2.1.2 R-Type

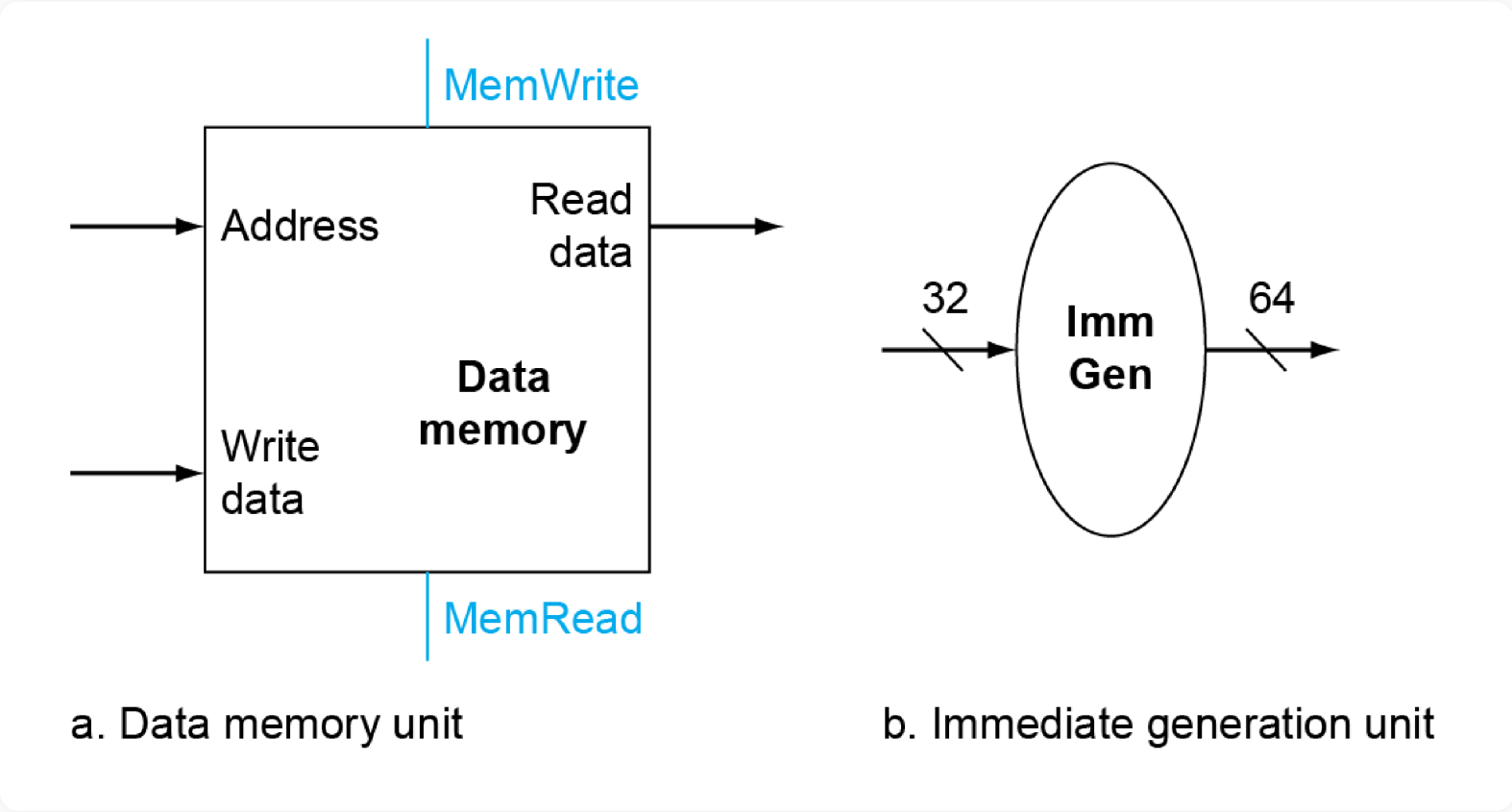
- Read two reg operands
- Perform ALU operation, *arithmetic/logical*
- Write result to reg



- 注意 RISC-V 中 `ALUop` 是 4 bit 的，因为实现了更多功能
- `reg` 的信号要求
  - `rs1, rs2`
  - `rd` 信号保持
  - `RegWrite = 1'b1`

2.1.3 Load/Store

- Read reg operands
- Calculate address, *use ALU, 12-bit offset, sign extened*
- Read/Write reg/mem



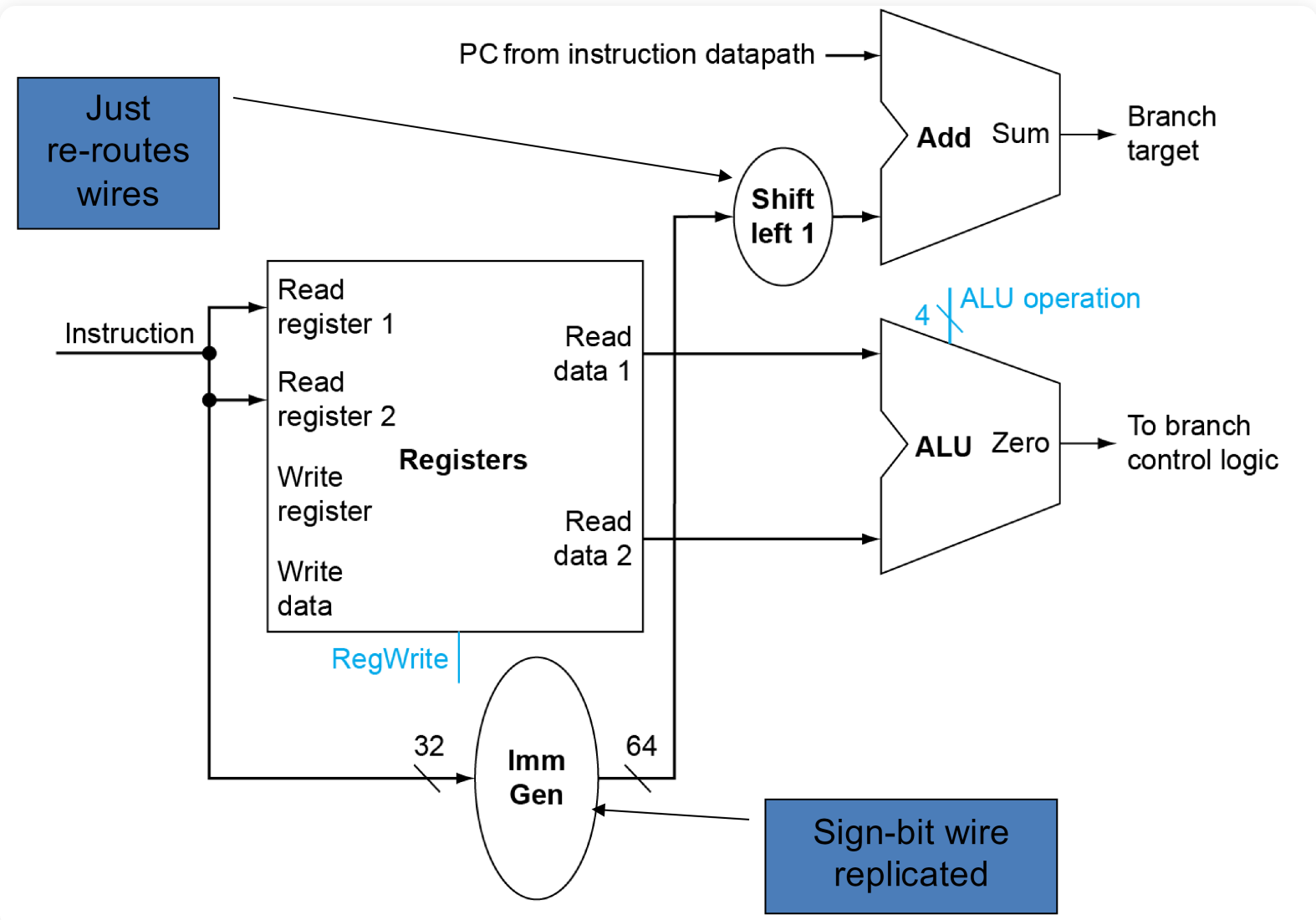
a. Data memory unit

b. Immediate generation unit

- 由于内存访问**开销较大**，设计 `MemRead` 来避免不可控的访存
- `ImmGen` 的输入就是 `inst`，要针对不同类型的指令提取/重组立即数，并进行 sign extend

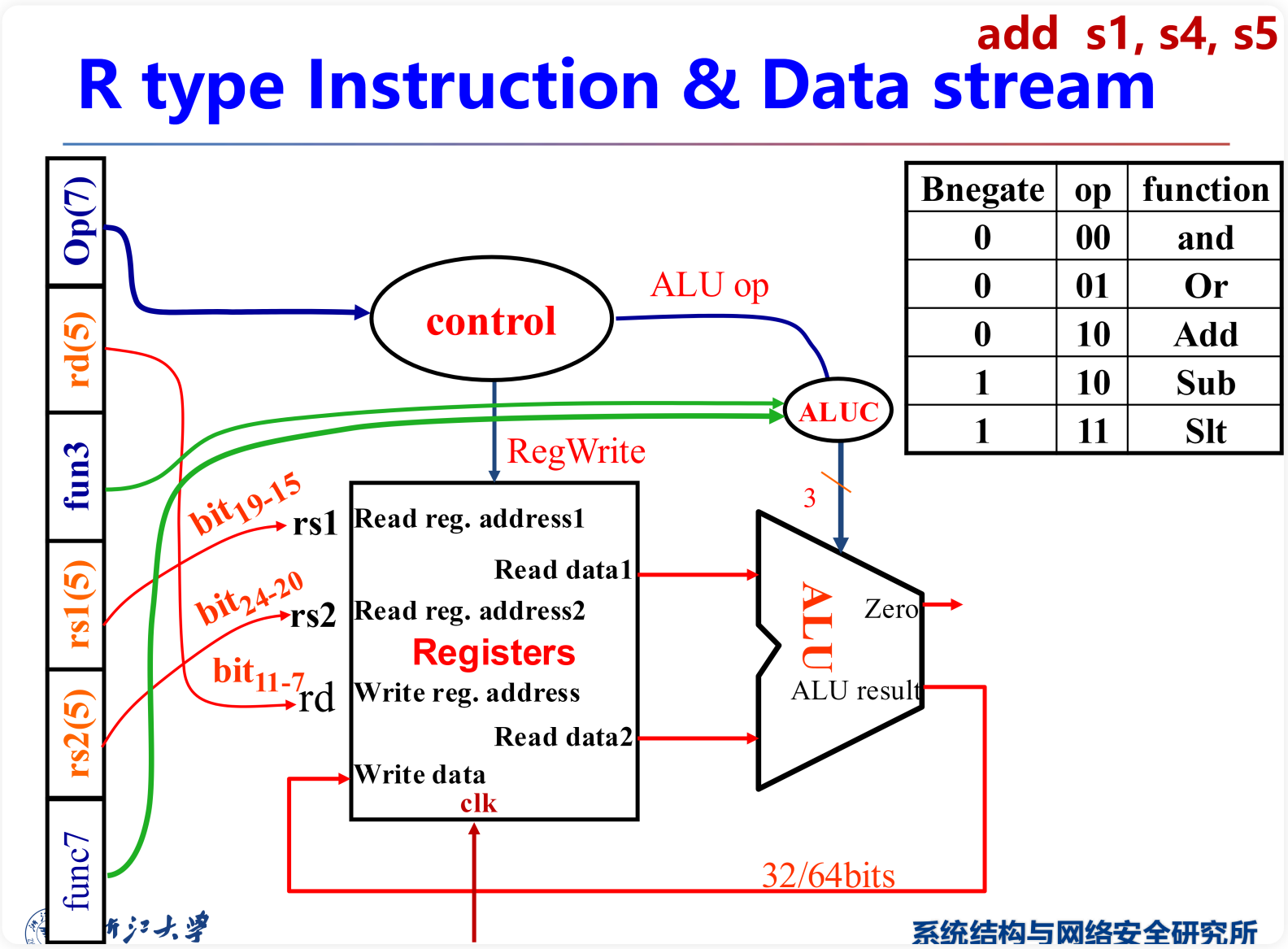
2.1.4 Branch

- Read reg operands
- Compare operands
  - ALU zero output
- Calculate target address
  - sign-extend displacement
  - shift left 1
  - add to PC value



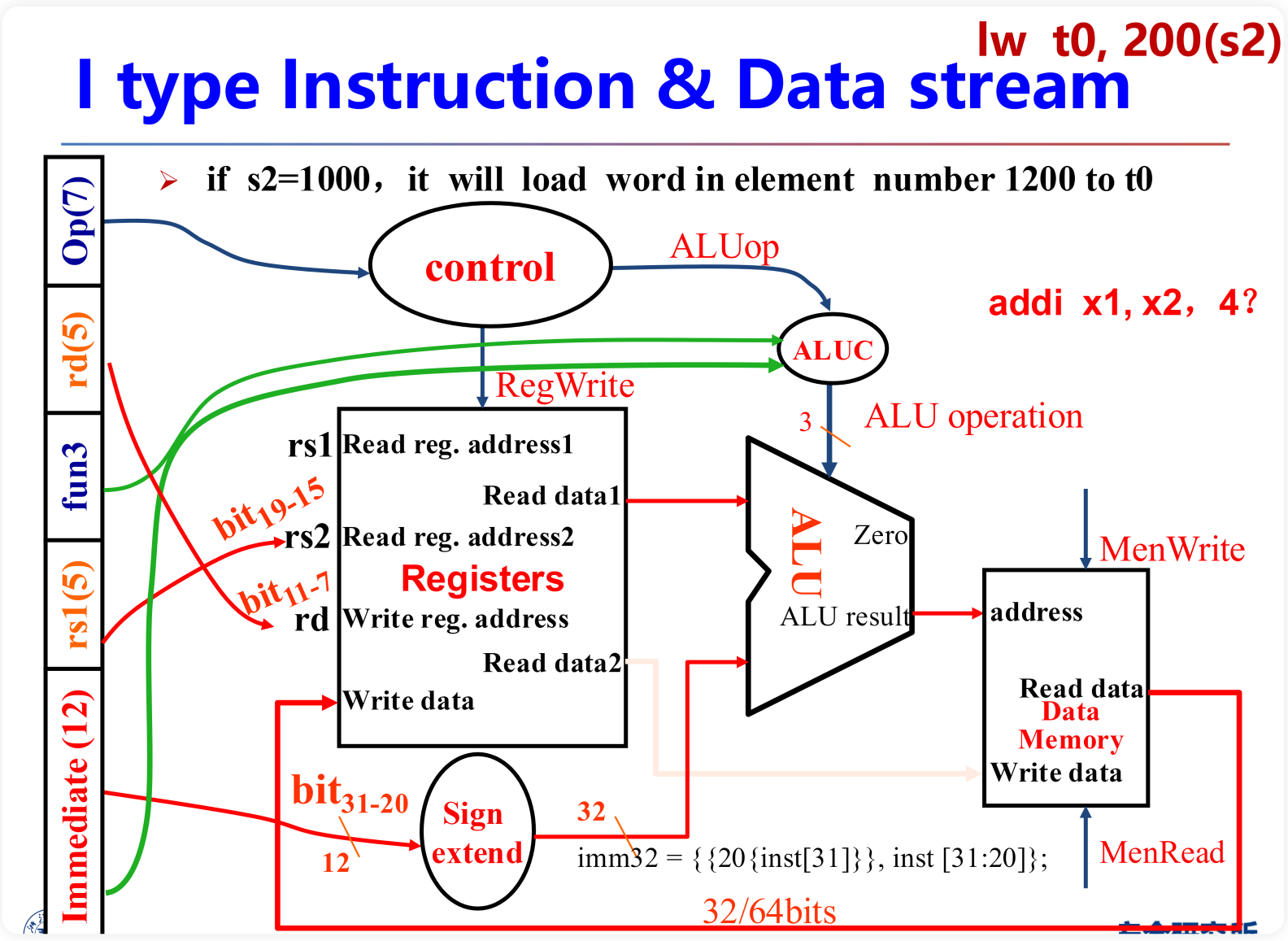
- offset 左移是在电路里完成的，产生了 offset * 2 的效果，但是在写代码的时候不需要考虑，编译器会处理好
- `jal` 没有用到 ALU，因为不需要跳转条件，但是也用到了寄存器，因为保存了 `return address`

2.2 连线题



Bnegate	op	function
0	00	and
0	01	Or
0	10	Add
1	10	Sub
1	11	Slt

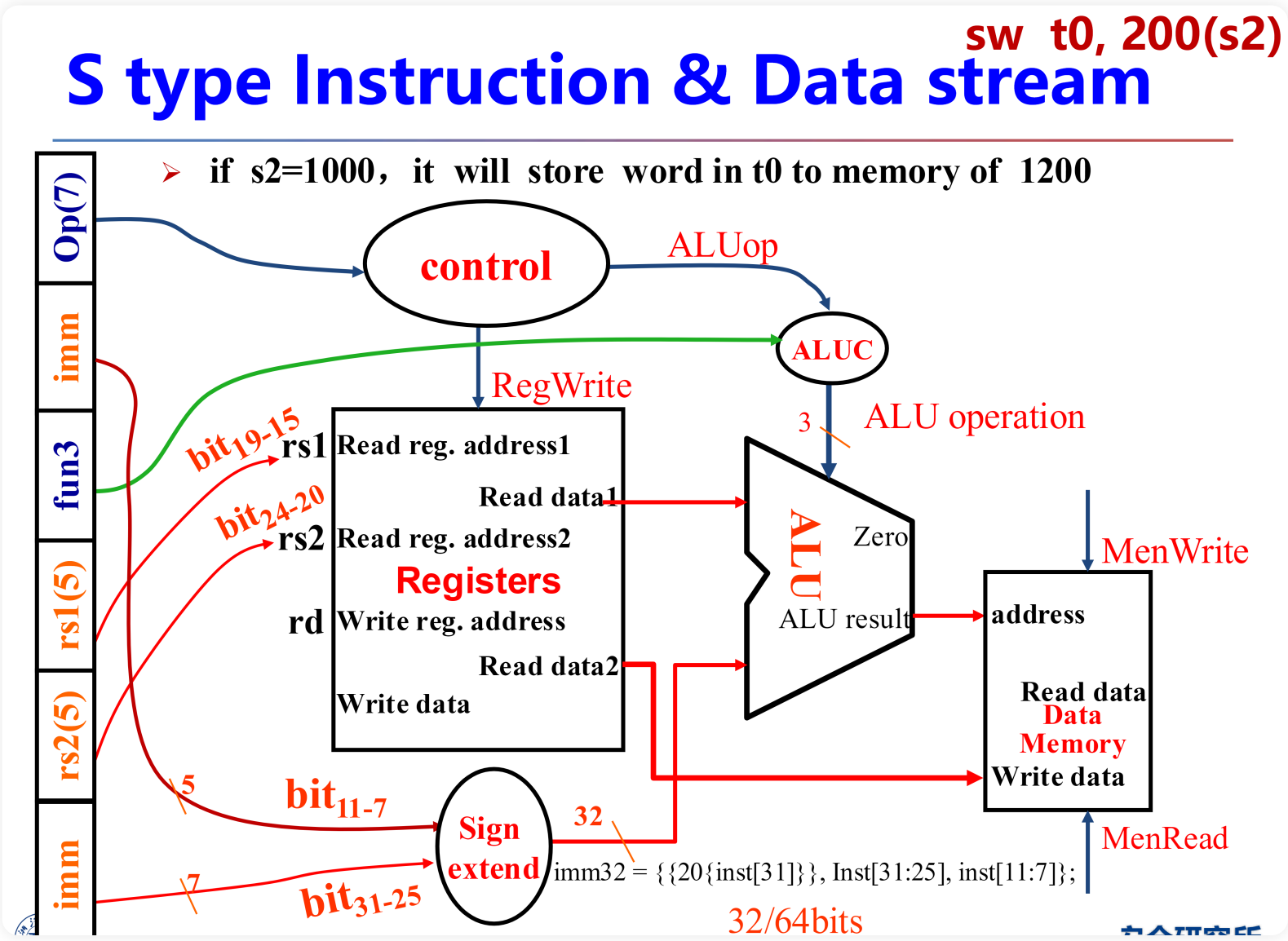
- ALUC 就是 `ALUControl`，将 `fun7, fun3` 单独解码有利于加快速度
- `ALUop[1:0]`
- ! 图中的 `ALUC` 输出应该是 4-bit



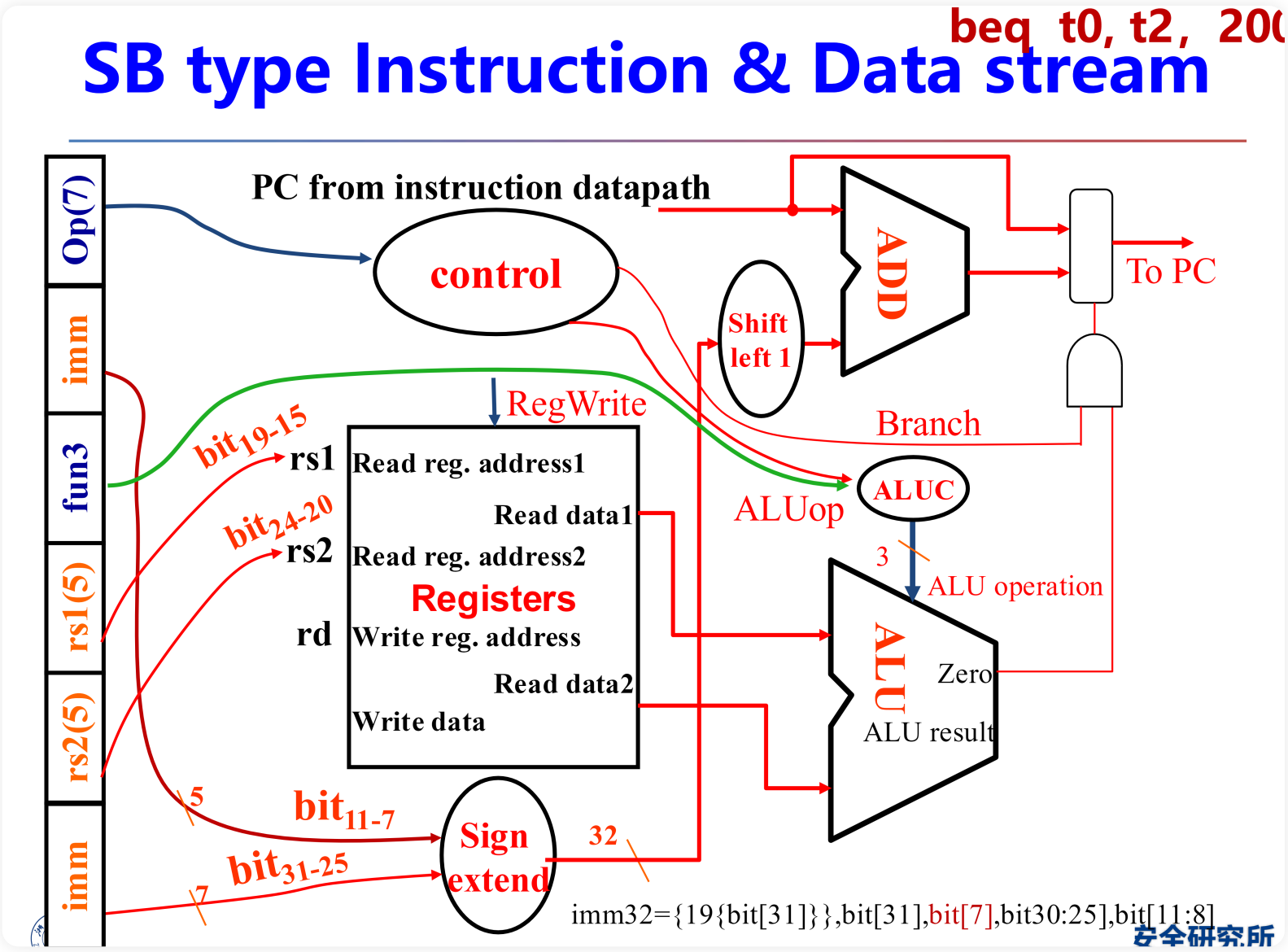
- ! 图中 `rs1` 连错了
- ! 图中 Sign extend 应该是 `ImmGen` 模块
- ! 这张图错误太多了

> ⑦ 改成 `addi`

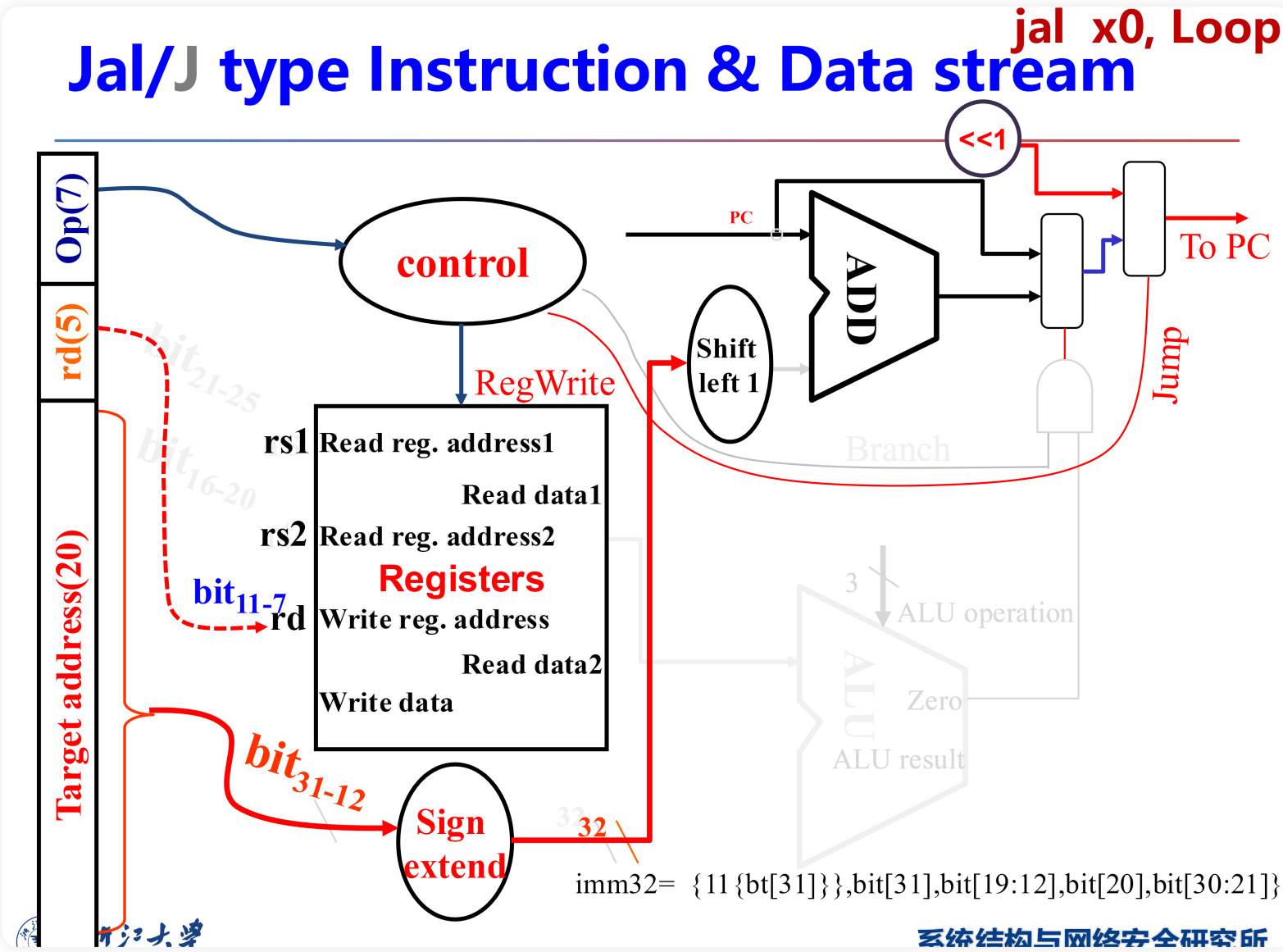
- 将 ALU result 接到 reg write data
- 让 MemWrite MemRead 都为 0



- `RegWrite = MemRead = 0, MemWrite = 1`

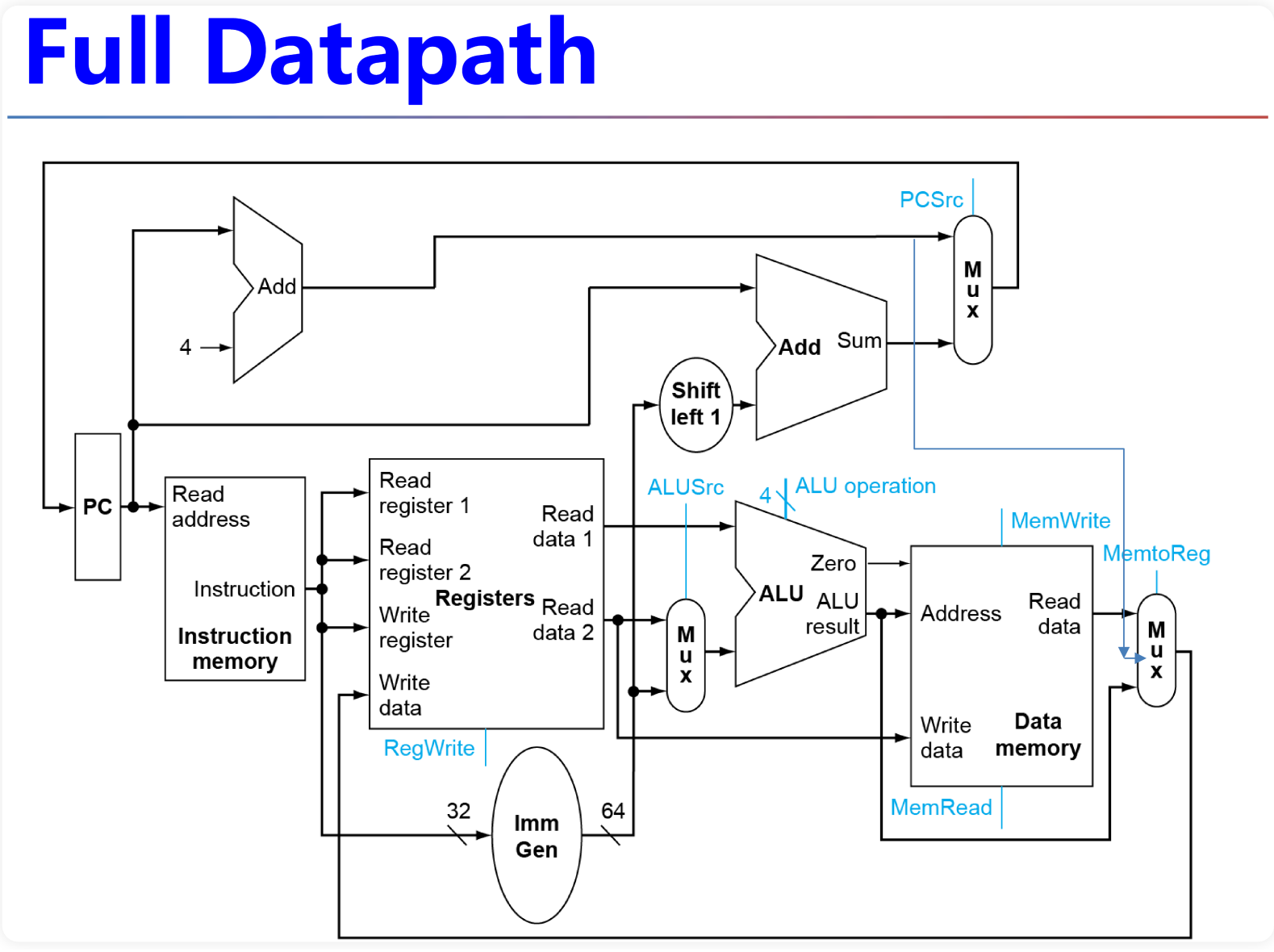


- ! 图中 Sign extend 应该为 `ImmGen`
- ! 图中右上角的 MUX 第一个输入应该为 `PC+4`



- ！这张图不完整，直接 loop 了，没有保存 `return addres`
- ！图中的 Sign extend 应该为 `ImmGen`
- ！应该为 `PC+4`
- 无条件跳转不需要 ALU

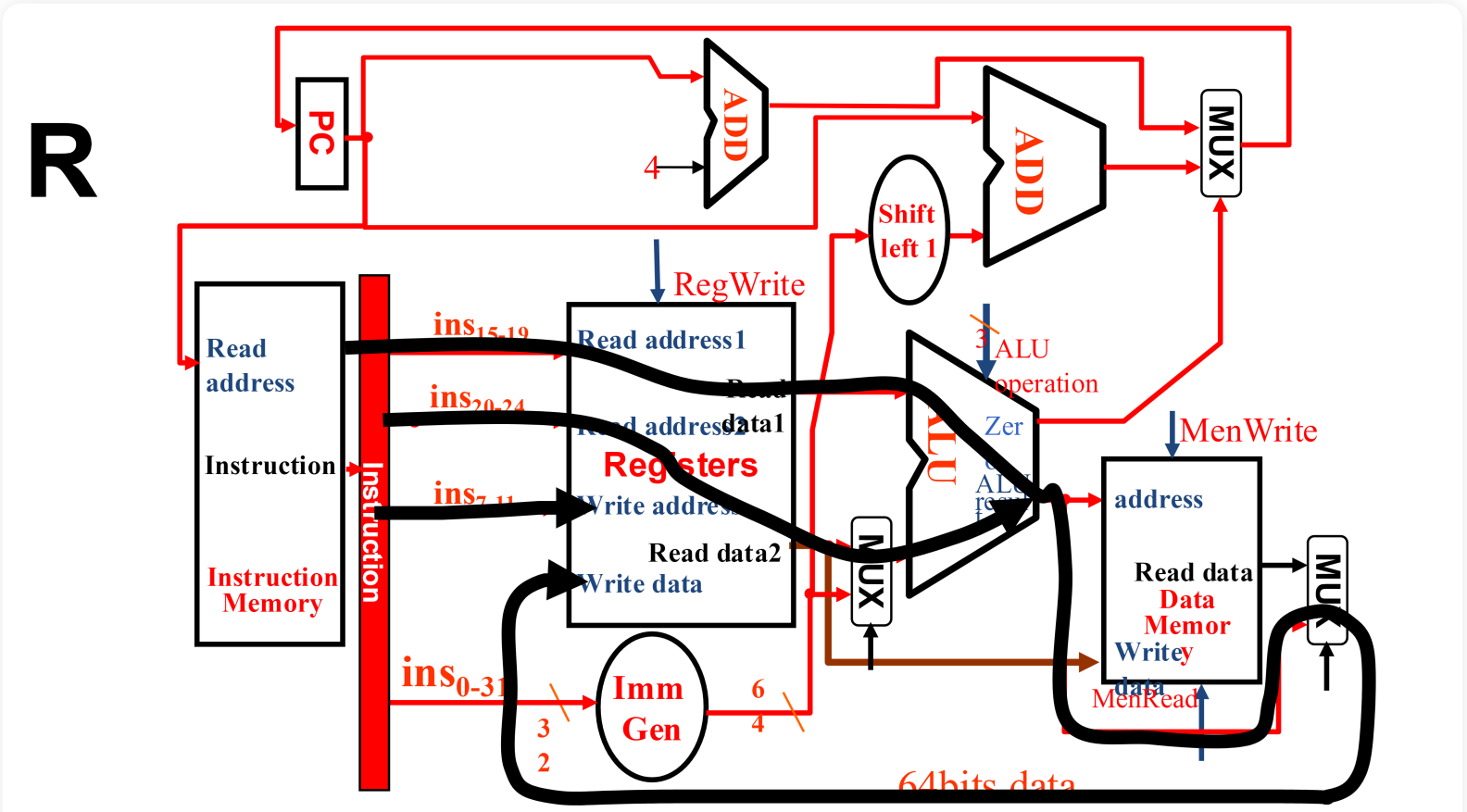
2.3 Full Datapath

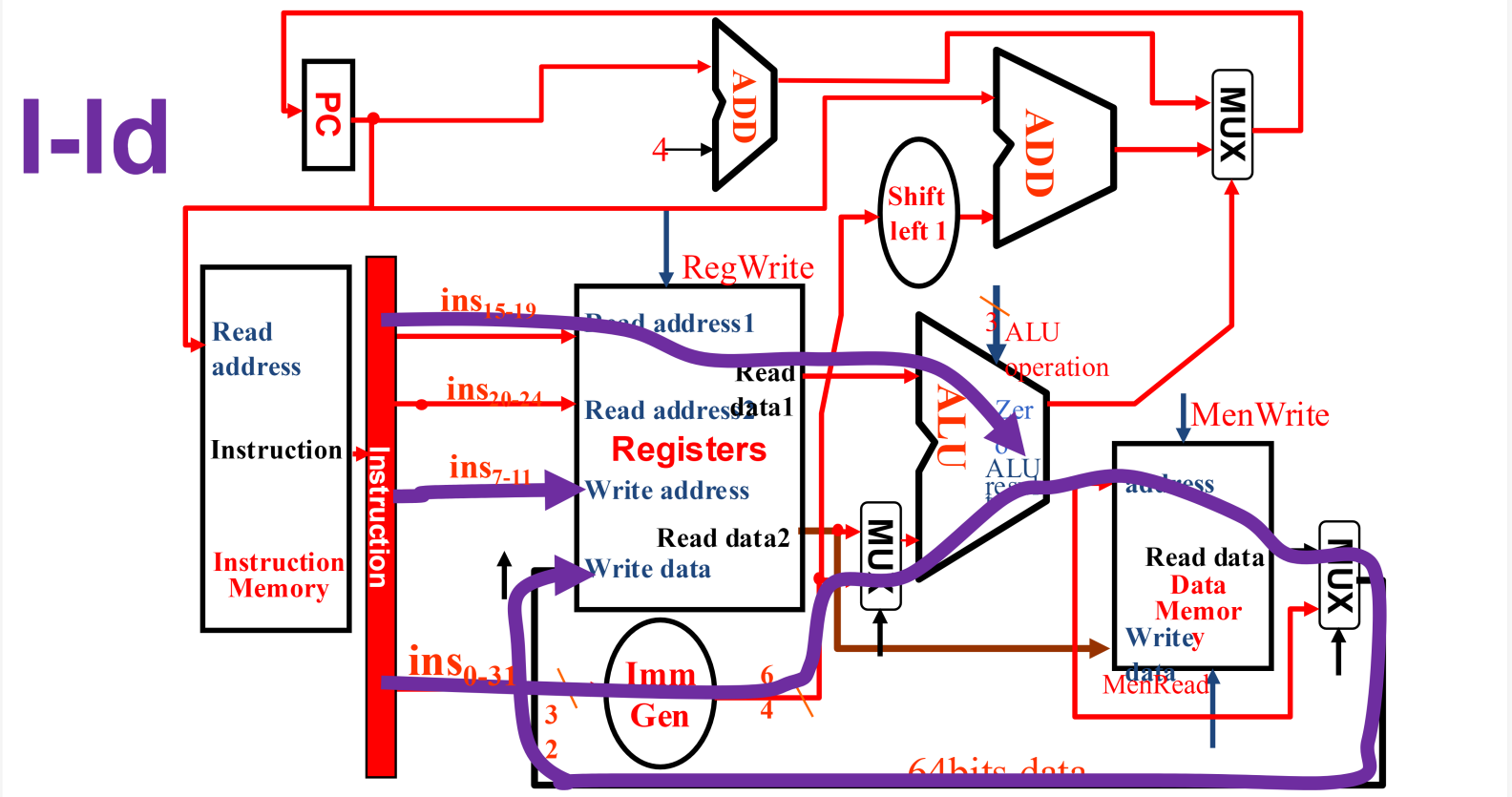


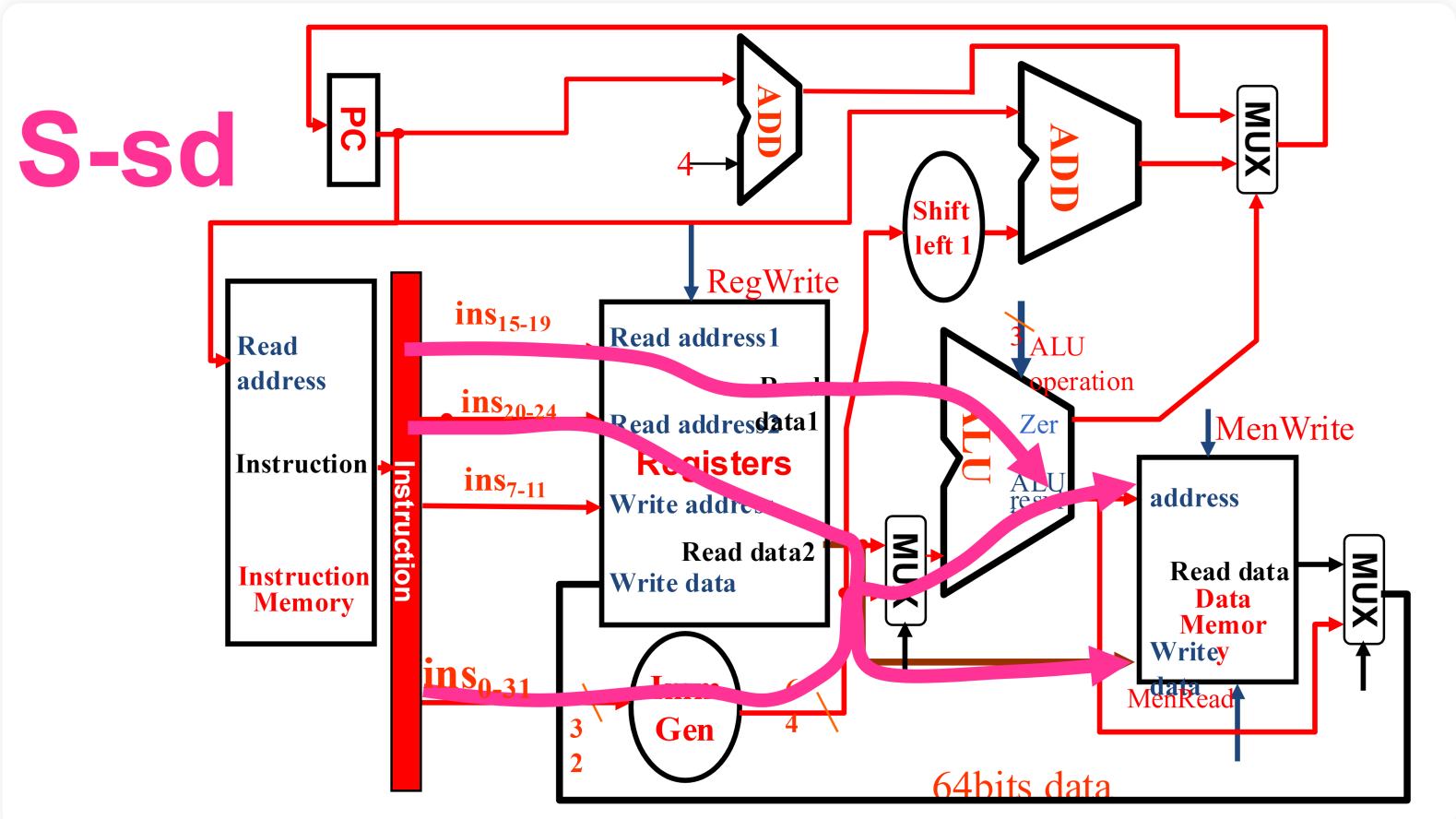
- 三个控制信号
  - `ALUSrc` 决定 operand2 是 `rs2` 还是 `imm`
  - `PCSrc` 决定 `PC` 是用 `PC+4` 还是跳转地址
  - `MemtoReg` 决定 reg write 用 mem 还是 `ALUresult`
- 五个步骤

- inst fetch
- decode & reg read
- execute
- mem
- write to reg

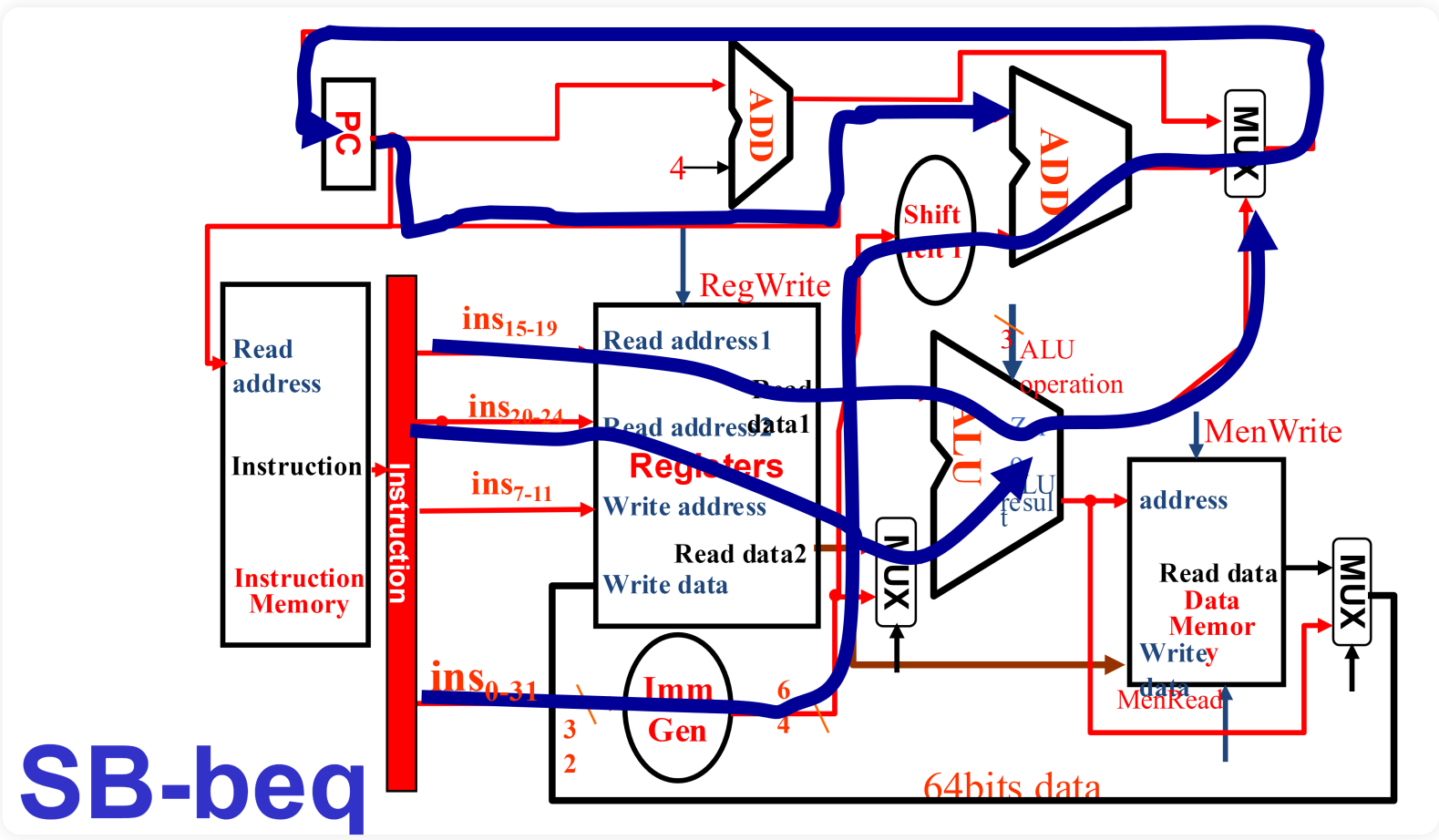
2.4 不同指令的数据通路



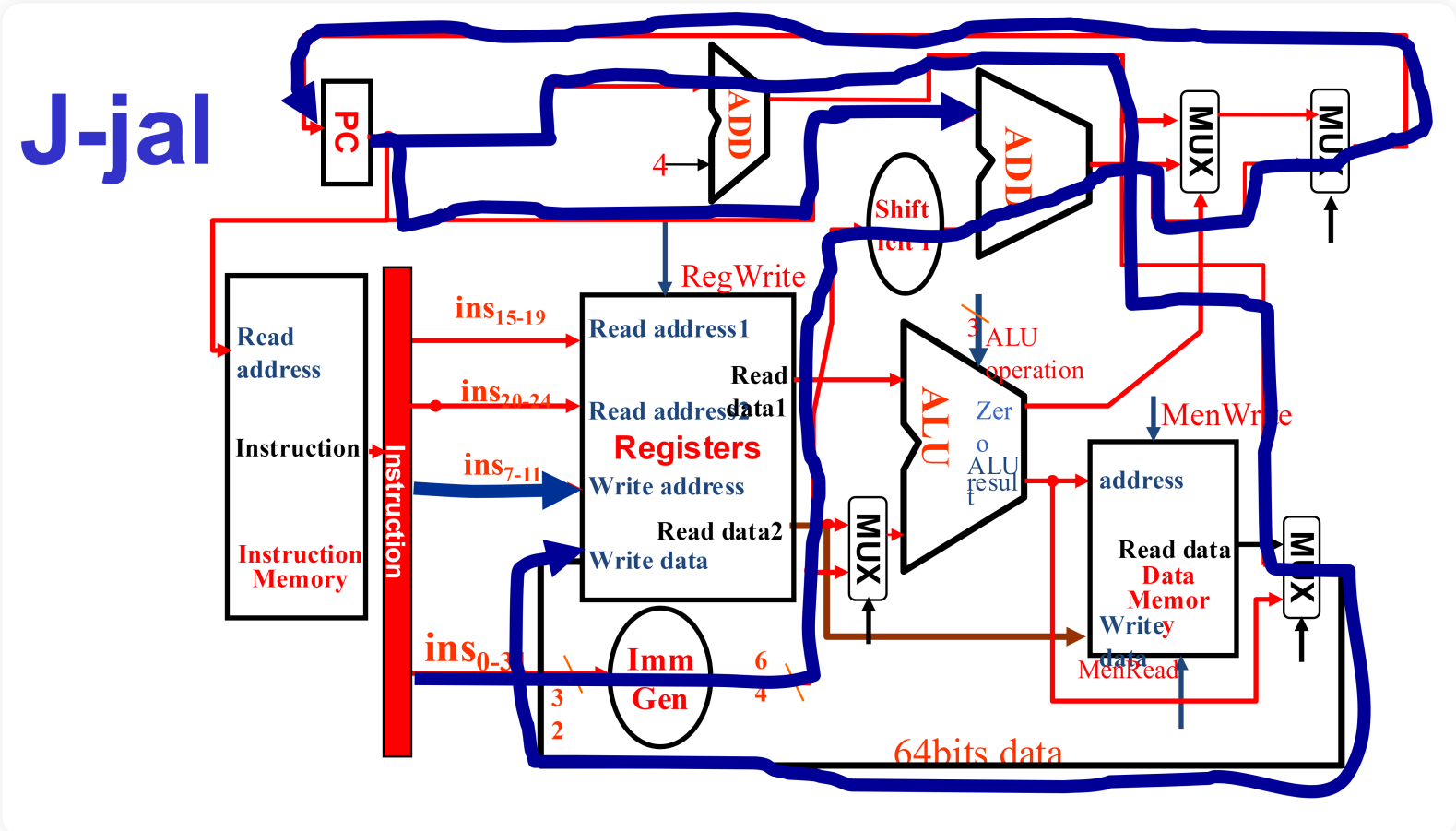




- `ALUSrc` 只能取 `Imm`
- `MemtoReg` 这个 MUX 取任何值都可以，反正 `RegWrite = 0`



- `PCSrc = Branch & ALU_zero`



- `jal` 做了两件事情
  1. 将 `PC+4` 存入 `rd`，需要经过 `MemtoReg`，于是需要换成 2-bit 控制信号
  2. 跳到 `PC+offset`，不用经过 ALU
- `ALUSrc, ALUOperation` 可以随便选，ALU 输入输出没有任何影响

3 Control

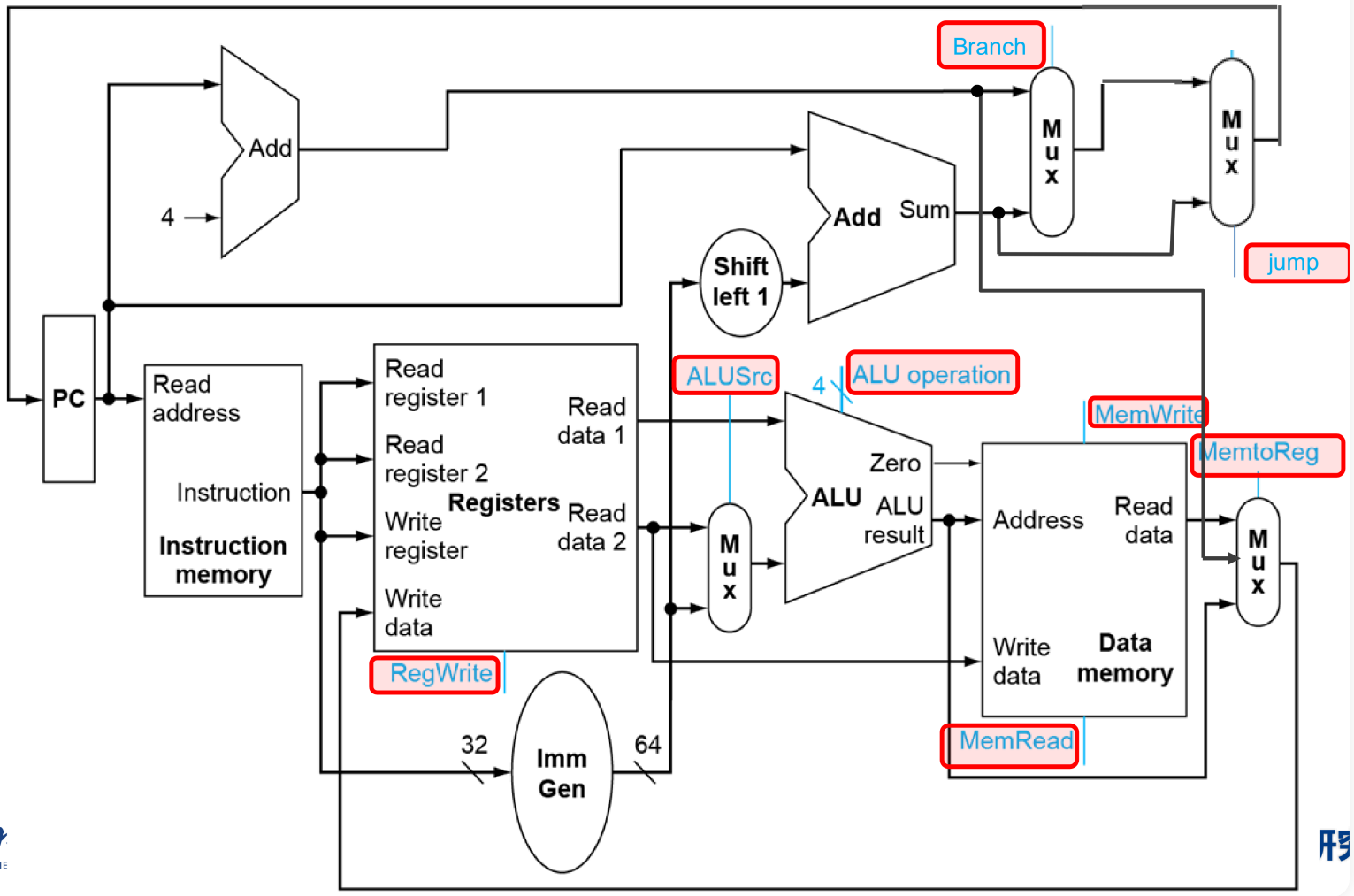
Analyze for cause and effect

- Information comes from the 32 bits of the instruction
- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
- ALU's operation based on instruction type and function code

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

3.1 控制信号 7+4

There are 7+4 signals



- & 7+4 控制信号
  - 7 个单独的信号
  - 4 位 ALU 控制信号
- ! 图中 `ALU_zero` 没有连接 `PCSrc`，当然 `ALU_zero` 也不属于控制信号

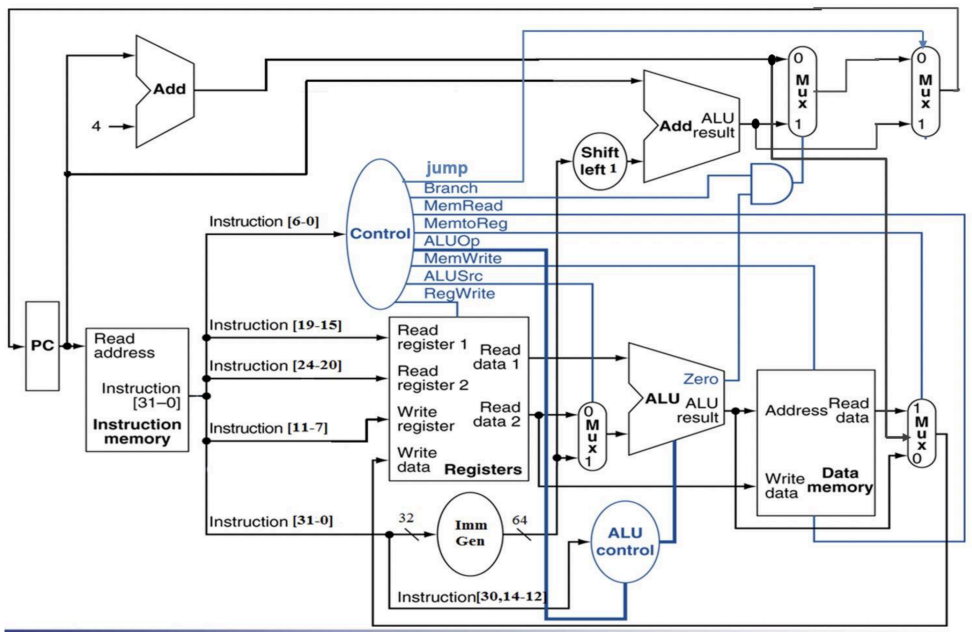
Signals for datapath Defined 7 control signals

Signal name	Effect when deasserted(=0)	Effect when asserted(=1)
RegWrite	None	Register destination input is written with the value on the Write data input
ALUScr	The second ALU operand come from the second register file output (Read data 2)	The second ALU operand is the sign-extended lower 16 bits of the instruction..
Branch（PCSrc）	The PC is replaced by the output of the adder that computers the value PC+4	The PC is replaced by the output of the adder that computers the branch target.
Jump	The PC is replaced by PC+4 or branch target	The PC is updated by jump address computed by adder
MemRead	None	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None	Data memory contents designated by the address input are replaced by value on the Write data input.
MemtoReg (2位)	00：The value fed to register Write data input comes from the Alu	01：The value fed to the register Write data input comes from the data memory.
		10：The value fed to the register Write data input comes from PC+4

- ! 和书上不同，`MemtoReg` 应该是两位的
- & 注意二级译码，`ALUop` 信号是 `ALUController` 二级译码产生的信号

3.2 完整的 control

Truth tables & Circuitry of main Controller



输入		输出								
Instruction	OPCode	ALUSrcB	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
R-format	0110011	0	00	1	0	0	0	0	1	0
Ld(I-Type)	0000011	1	01	1	1	0	0	0	0	0
sd(S-Type)	0100011	1	X	0	0	1	0	0	0	0
beq(SB-Type)	1100111	0	X	0	0	0	1	0	0	1
Jal(UJ-Type)	1101111	X	10	1	0	0	0	1	X	X

- & 一定要记住的一张图
- 这里使用了二级译码，`control` 输出的 `ALUOp` 是 2 位的
- 关于 `X` 的情况
  - `sd, bep` 都不写入寄存器，`RegWrite = 0`，所以 `MemtoReg = don't care`
  - `jal` 不需要主 ALU 的任何操作，所以 `ALUSrcB = ALUOp = don't care`
- Instruction memory & Data memory
  - 从处理器的视角，有利于提高吞吐量
  - 但是在实际内存中是共享的，只是缓存映射不一样

3.2.1 关于二级译码

Design the ALU Decoder second level

- ALU operation is decided by 2-bit ALUOp derived from opcode, and funct7 & funct3 fields of the instruction
  - Combinational logic derives ALU control

opcode	ALUOp	Operation	Funct7	funct3	ALU function	ALU control
ld	00	load register	XXXXXXX	xxx	add	0010
sd	00	store register	XXXXXXX	xxx	add	0010
beq	01	branch on equal	XXXXXXX	xxx	subtract	0110
R-type	10	add	0000000	000	add	0010
		subtract	0100000	000	subtract	0110
		AND	0000000	111	AND	0000
		OR	0000000	110	OR	0001
		SLT	0000000	010	Slt	0111

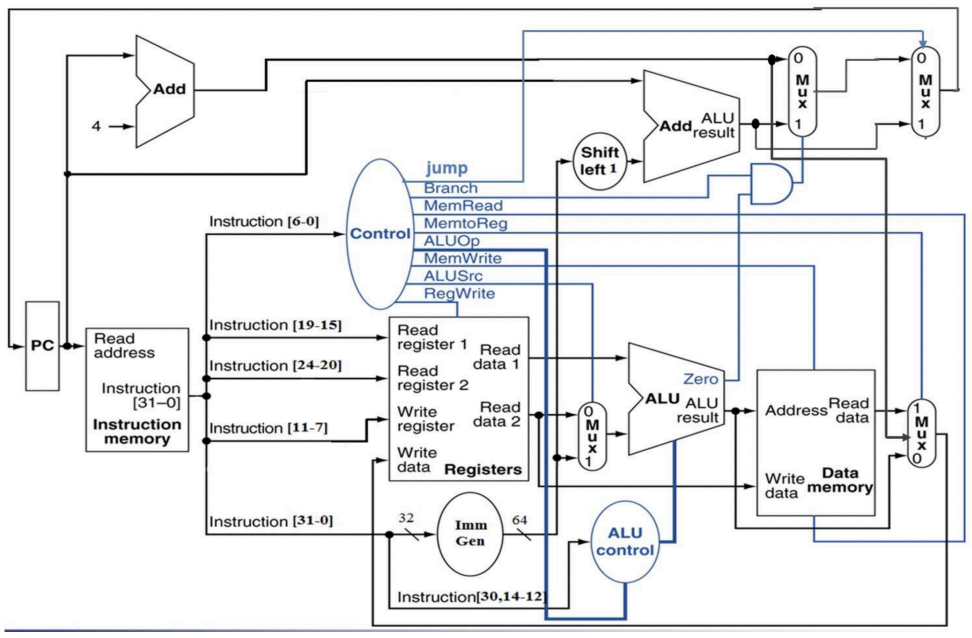
3.3 更多分析题

CO_Ch4_Pt1, p.54

4 Conclusion

- & 记住下面两张图

Truth tables & Circuitry of main Controller



输入		输出								
Instruction	OPCode	ALUSrcB	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU Op1	ALU Op0
R-format	0110011	0	00	1	0	0	0	0	1	0
Ld(I-Type)	0000011	1	01	1	1	0	0	0	0	0
sd(S-Type)	0100011	1	X	0	0	1	0	0	0	0
beq(SB-Type)	1100111	0	X	0	0	0	1	0	0	1
Jal(UJ-Type)	1101111	X	10	1	0	0	0	1	X	X

Design the ALU Decoder second level

- **ALU operation is decided by 2-bit ALUOp derived from opcode, and funct7 & funct3 fields of the instruction**
  - Combinational logic derives ALU control

opcode	ALUOp	Operation	Funct7	funct3	ALU function	ALU control
ld	00	load register	XXXXXXX	xxx	add	0010
sd	00	store register	XXXXXXX	xxx	add	0010
beq	01	branch on equal	XXXXXXX	xxx	subtract	0110
R-type	10	add	0000000	000	add	0010
		subtract	0100000	000	subtract	0110
		AND	0000000	111	AND	0000
		OR	0000000	110	OR	0001
		SLT	0000000	010	Slt	0111