CS151B
Computer Systems Architecture

Week 5 Discussion

2/9/2018

Logistics

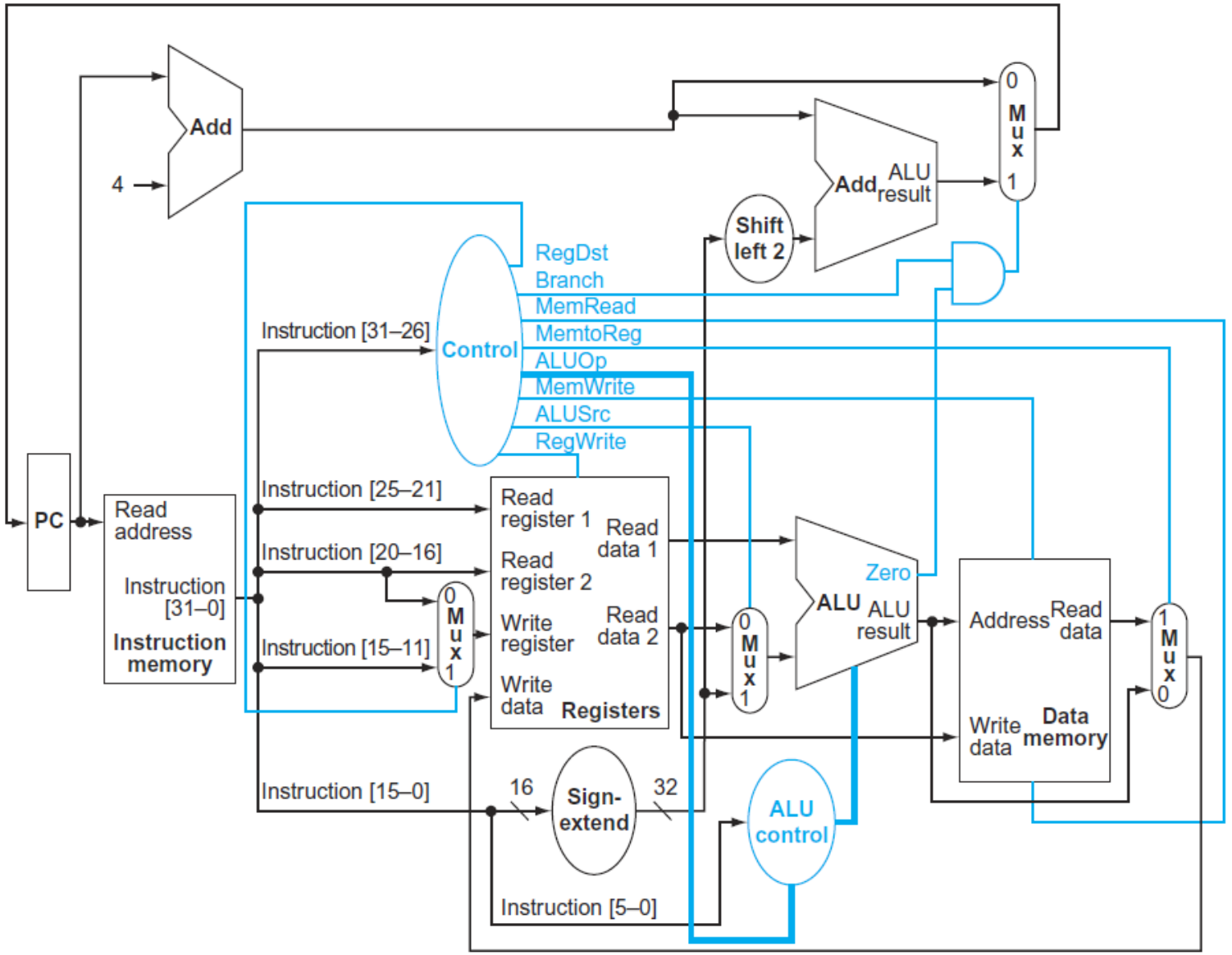
- HW4 due today
  - Implement blt, jal and jr

Agenda

- Review

A Short Topic List

- Performance
  - ET, CPI
- ISA
- ALU
  - Adders (RCA, CLA, HCLA, CSA)
  - Multipliers (Booth's Algorithm)
- Single cycle datapath
  - Implement new instructions


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

Main Controller

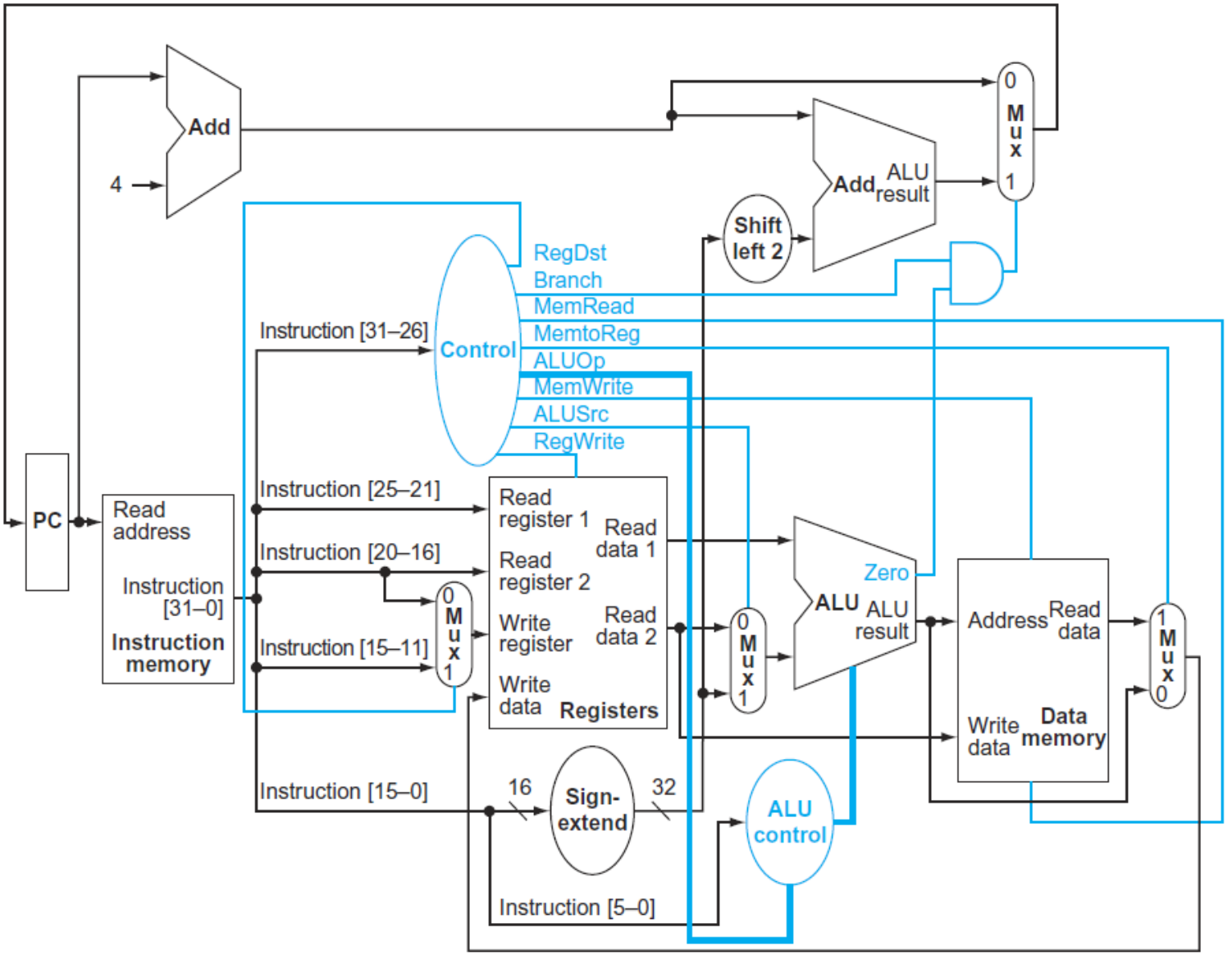
Input or Output	Signal Name	R-format	lw	sw	Beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

ALU Controller

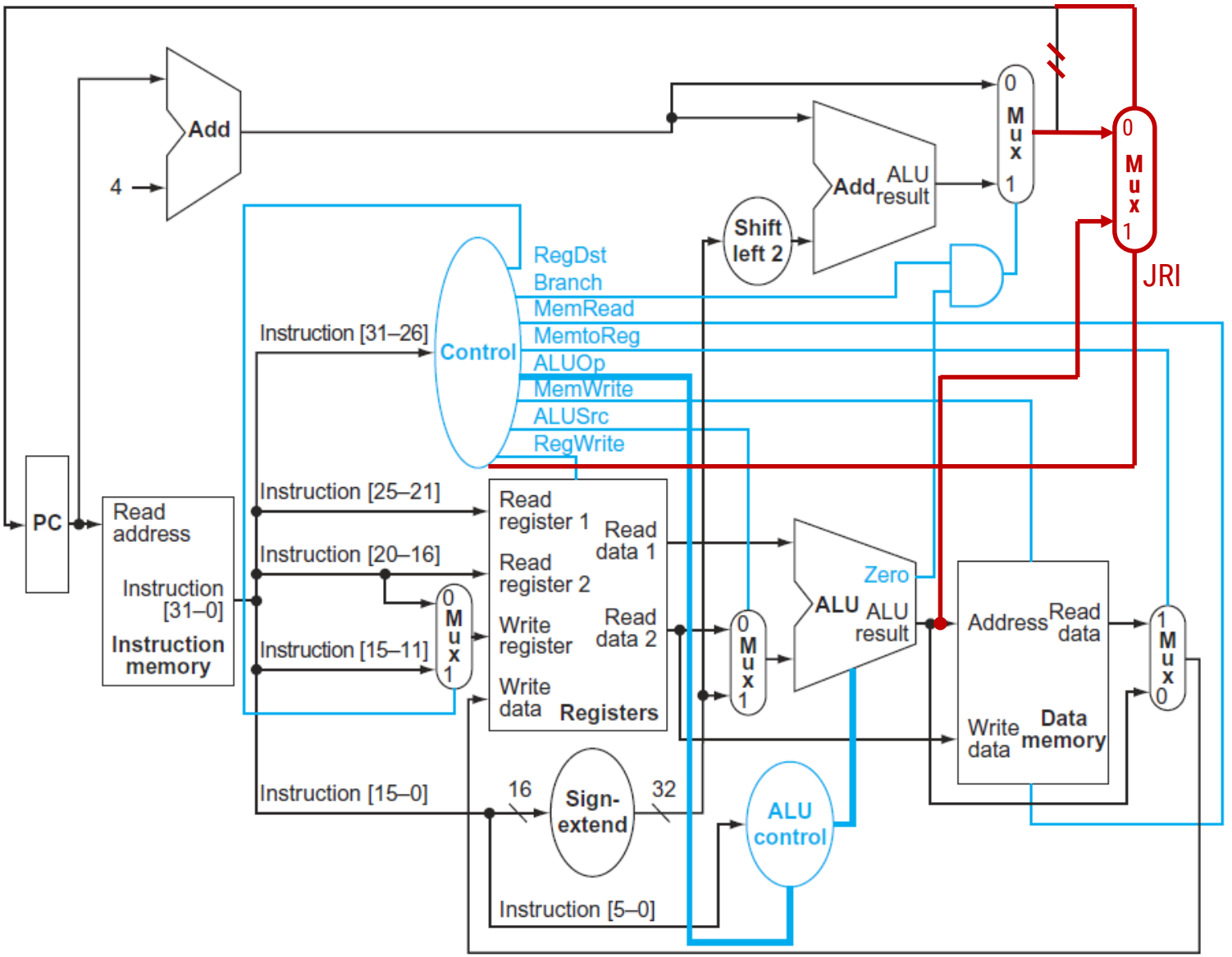
Opcode	**ALUOp**	**instruction**	**function**	**ALU Action**	**ALUCtrl**
Lw	00	load word	XXXXXX	add	010
Sw	00	store word	XXXXXX	add	010
Beq	01	branch equal	XXXXXX	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	AND	000
R-type	10	OR	100101	OR	001
R-type	10	SLT	101010	SLT	111

Sample Question

- jri: jump register + immediate
  - jri $s0 IMM
  - PC = R[rs] + sign-extend(IMM)



jri: PC = R[rs] + sign-extend(IMM)



jri: PC = R[rs] + sign-extend(IMM)

	jri
RegDst	
ALUSrc	
MemtoReg	
RegWrite	
MemRead	
MemWrite	
Branch	
ALUOp1	
ALUOp0	
JRI	

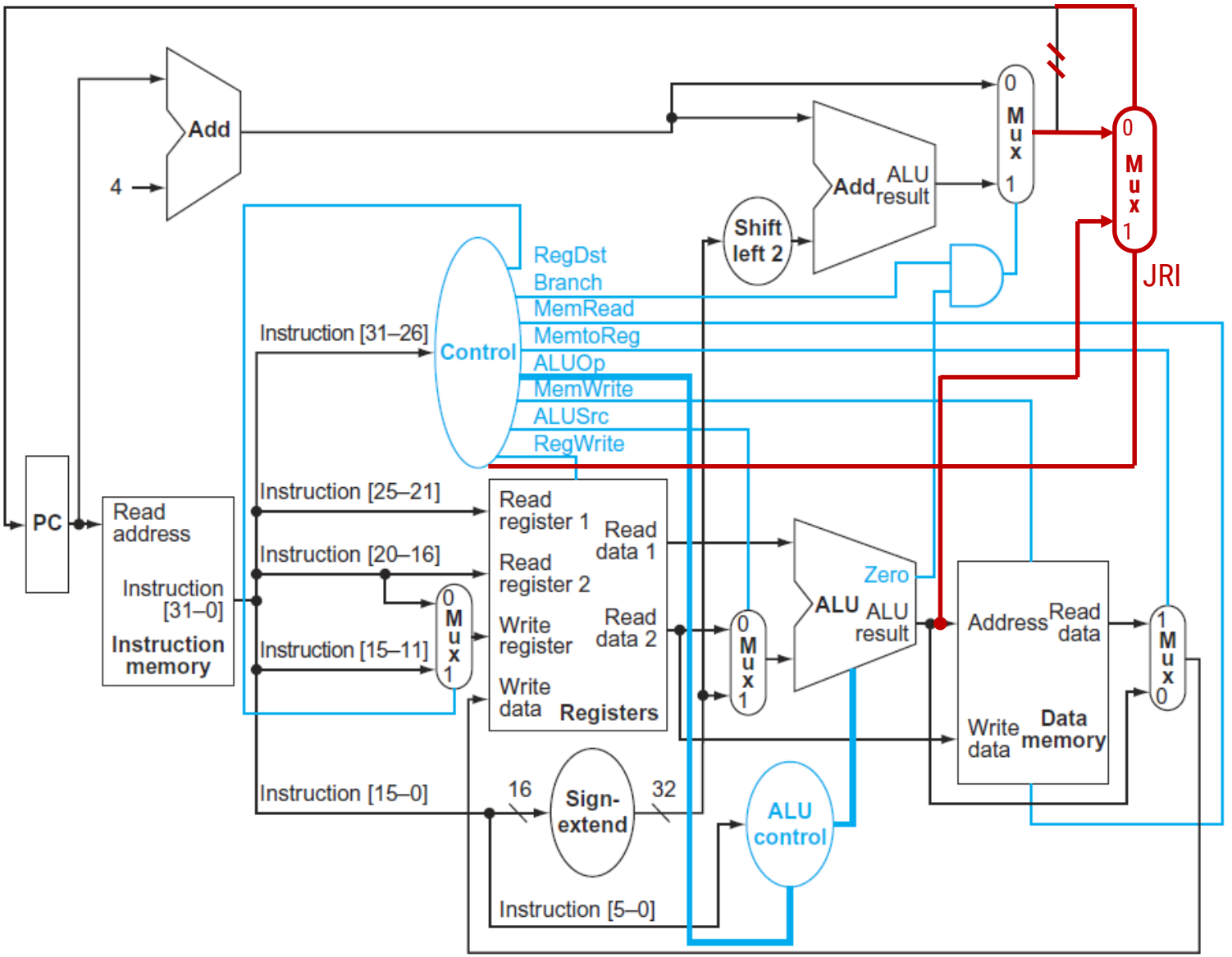
jri: PC = R[rs] + sign-extend(IMM)

	jri
RegDst	**x**
ALUSrc	**1**
MemtoReg	**x**
RegWrite	**0**
MemRead	**0**
MemWrite	**0**
Branch	**x**
ALUOp1	**0**
ALUOp0	**0**
JRI	**1**

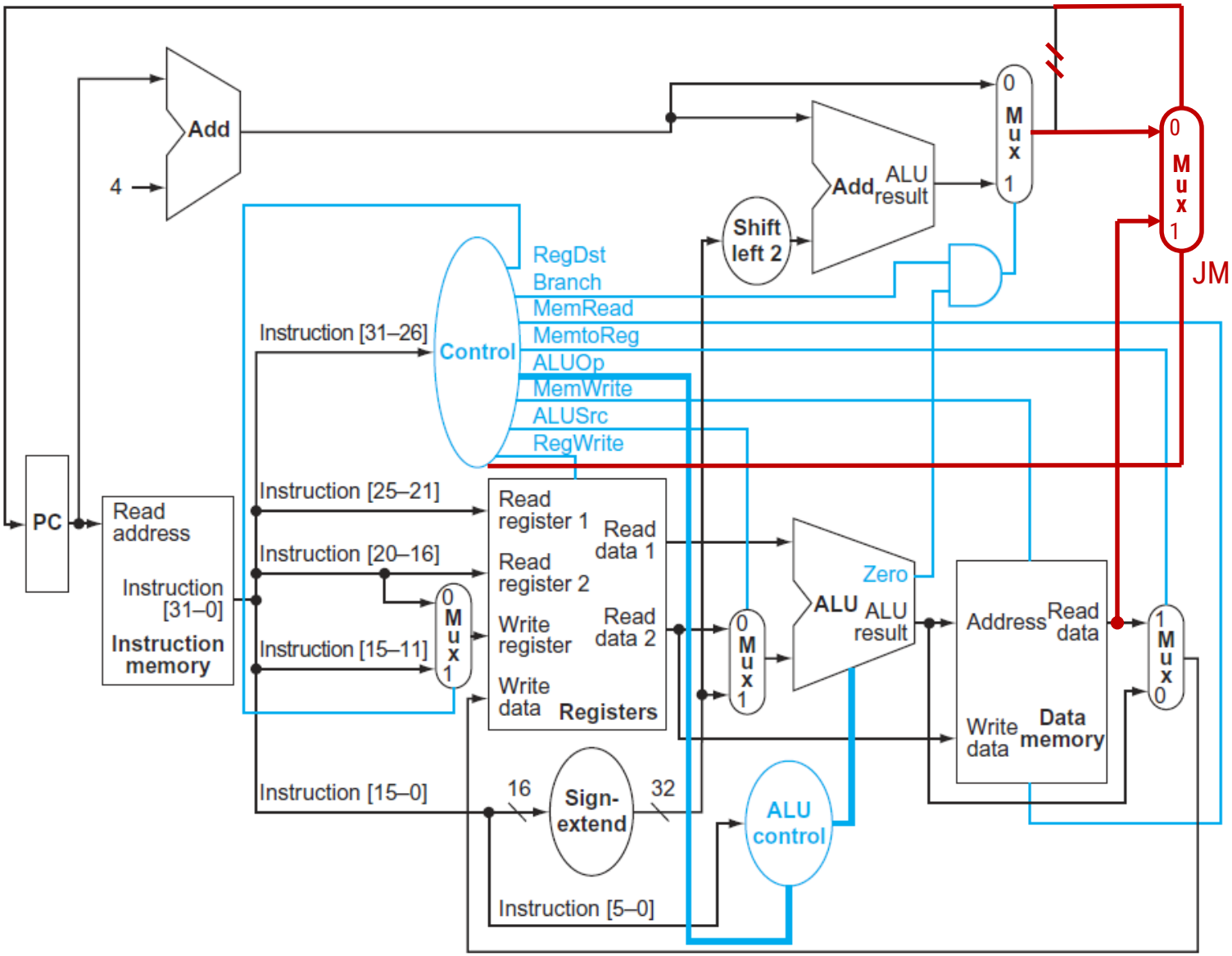
jri: PC = R[rs] + sign-extend(IMM)

Sample Question

- jm: jump to M[register + immediate]
  - jm IMM($rs)
  - PC = M[R[rs] + sign-extend(IMM)]



jri: PC = R[rs] + sign-extend(IMM)



jm: PC = M[R[rs] + sign-extend(IMM)]

	jm
RegDst	
ALUSrc	
MemtoReg	
RegWrite	
MemRead	
MemWrite	
Branch	
ALUOp1	
ALUOp0	
JM	

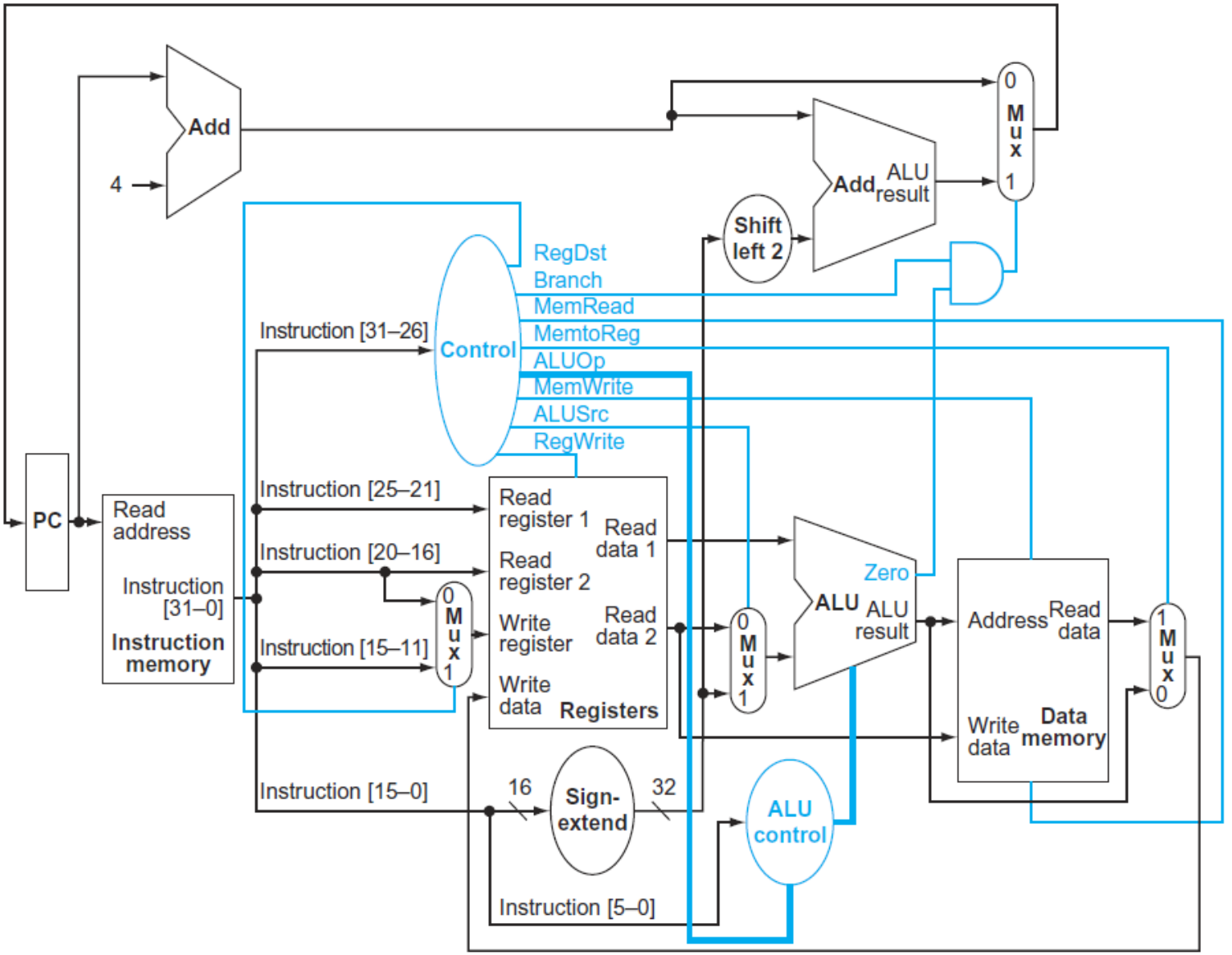
jm: PC = M[R[rs] + sign-extend(IMM)]

	jm
RegDst	**x**
ALUSrc	**1**
MemtoReg	**x**
RegWrite	**0**
MemRead	**1**
MemWrite	**0**
Branch	**x**
ALUOp1	**0**
ALUOp0	**0**
JM	**1**

jm: PC = M[R[rs] + sign-extend(IMM)]

Sample Question

- lwr: sums two registers to obtain the effective load address
  - lwr $rd $rs $rt
  - R[rd] = M[R[rs] + R[rt]]



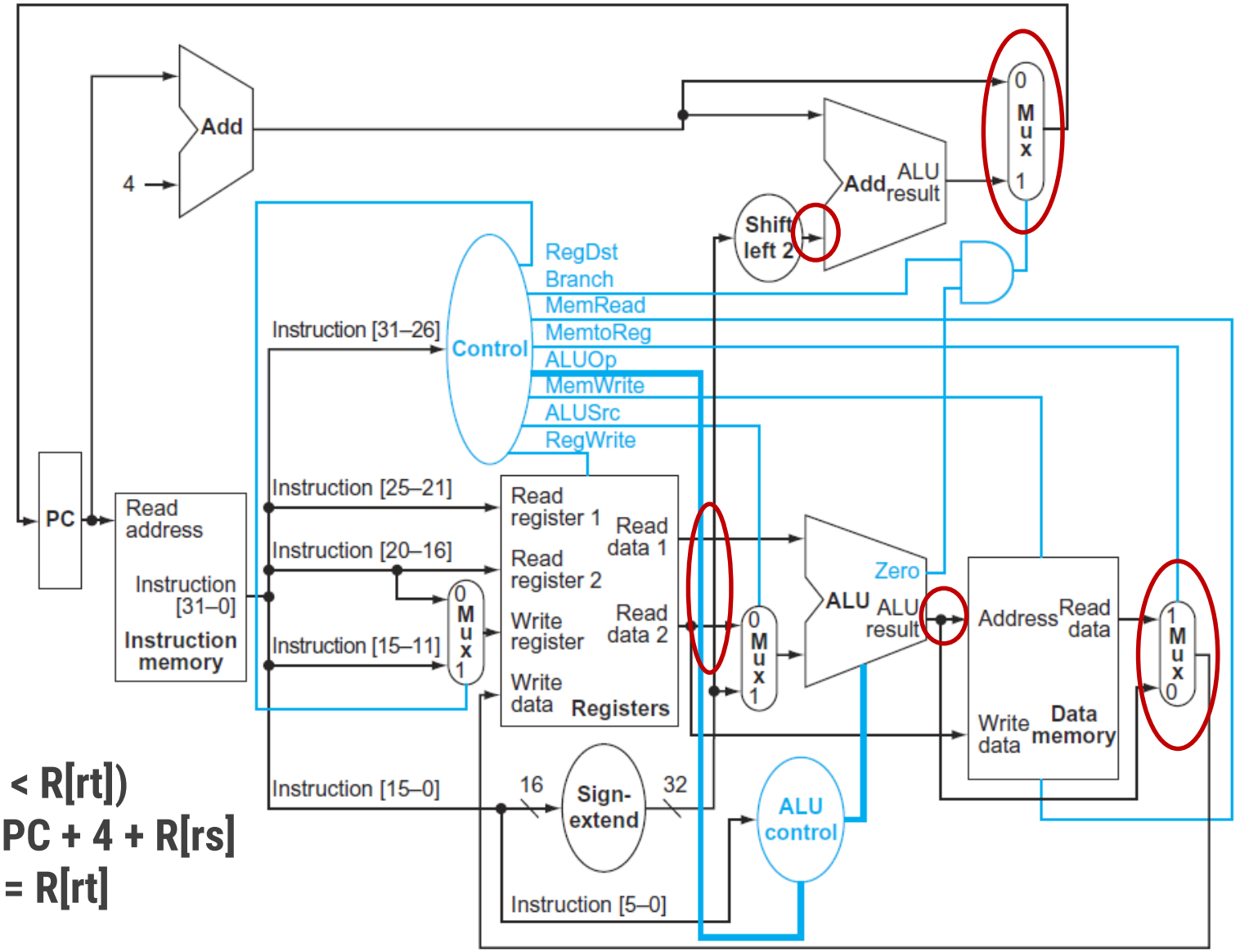
lwr: R[rd] = M[R[rs] + R[rt]]

- **Control table?**
  - We want a different set of control signals than that of the default R-types (lwr accesses memory while other R-types don't)
  - Since lwr is an R-type instruction, the main controller cannot generate a different set of control signals for it
  - We will have to resort to the ALU controller
  - The ALU controller need to generate control signals that used to be generated by the main controller when we want a different value, plus a dedicated control signal for lwr
  - We need muxes to select between the two versions of control signals, toggled by lwr

lwr: R[rd] = M[R[rs] + R[rt]]

Sample Question

- funkyb
  - funkyb $rd $rs $rt
  - if (R[rs] < R[rt])
    PC = PC + 4 + R[rs]
    R[rd] = R[rt]
    else
    PC = PC + 4 + R[rt]
    R[rd] = R[rs]




```
if (R[rs] < R[rt])
    PC = PC + 4 + R[rs]
    R[rd] = R[rt]
else
    PC = PC + 4 + R[rt]
    R[rd] = R[rs]
```

ALU Controller

Opcode	ALUOp	instruction	function	ALU Action	ALUCtrl
Lw	00	load word	XXXXXX	add	010
Sw	00	store word	XXXXXX	add	010
Beq	01	branch equal	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

Opcode	ALUOp	instruction	function	ALU Action	ALU Ctrl	Funkyb
	10	funkyb		subtract	110	1

```
if (R[rs] < R[rt])
    PC = PC + 4 + R[rs]
    R[rd] = R[rt]
else
    PC = PC + 4 + R[rt]
    R[rd] = R[rs]
```