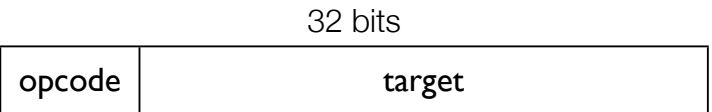
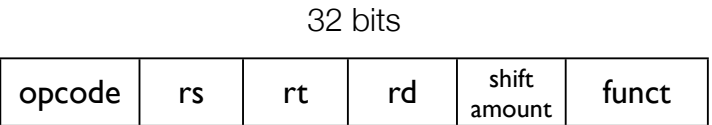
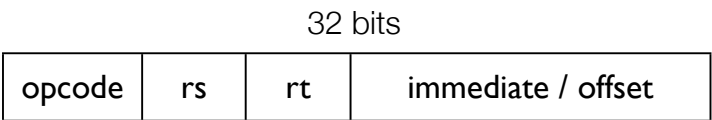
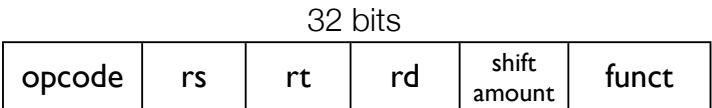


Midterm Review

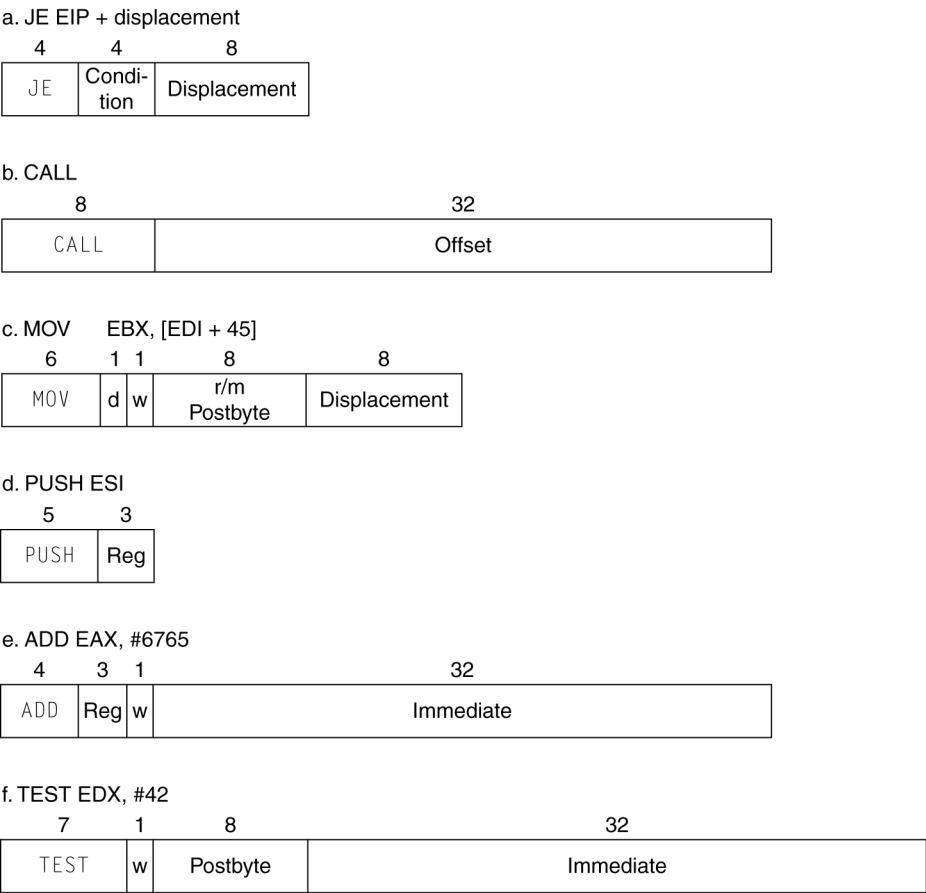
Jason Mars

ISA - Instruction Length

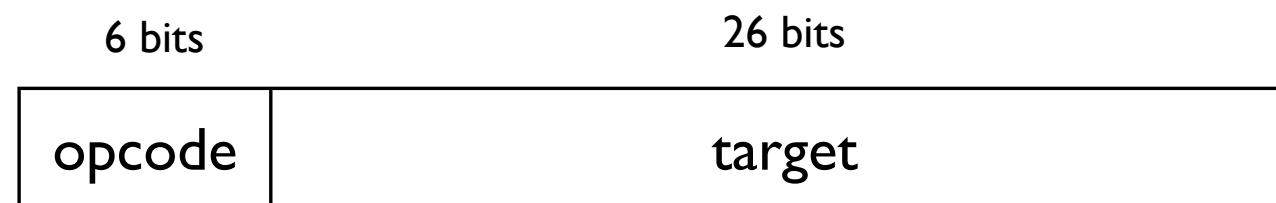
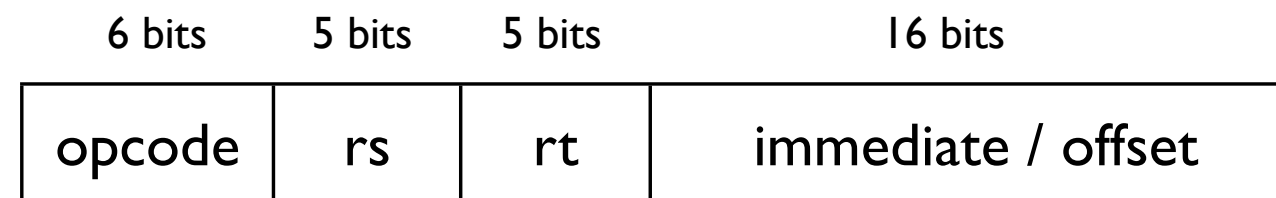
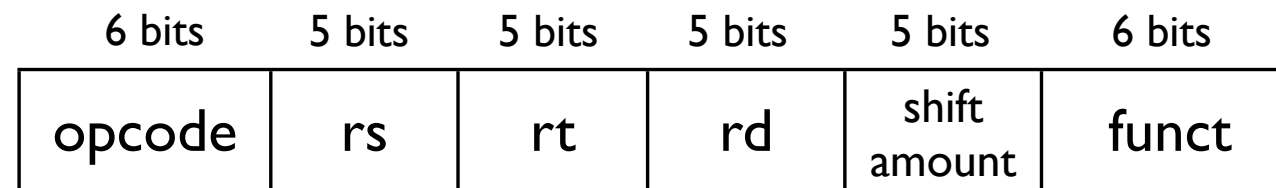
Fixed Length



Variable Length



ISA



- Why 5 bits for rs, rt, rd?
- What impacts # bits in opcode
- Why is it ok to have 16 bits in immediate
- What is the difference between immediate and register source
- What are the trade-offs between fixed length and variable length?
- What is a load-store ISA?

Addressing Modes

- Register direct $R3$
- Immediate (literal) $\#25$
- Direct (absolute) $M[10000]$
- Register indirect $M[R3]$
- Base+Displacement $M[R3 + 10000]$
- Base+Index $M[R3 + R4]$
- Scaled Index $M[R3 + R4 * d + 10000]$
- Autoincrement $M[R3++]$
- Autodecrement $M[R3--]$
- Memory Indirect $M[M[R3]]$

ISA - Addressing Modes in MIPS

register direct

OP	rs	rt	rd	sa	funct
----	----	----	----	----	-------

add \$1, \$2, \$3

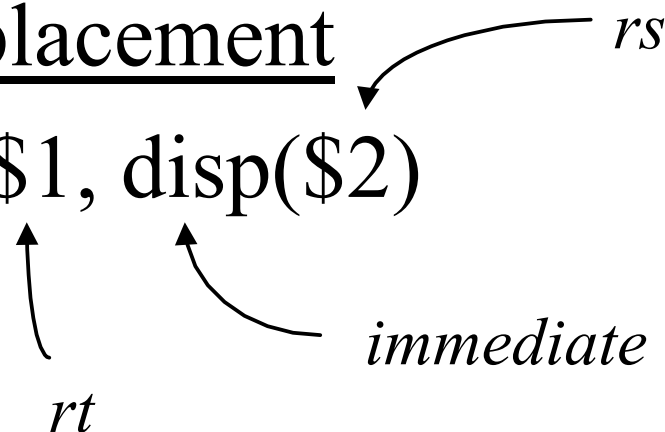
immediate

OP	rs	rt	immediate
----	----	----	-----------

add \$1, \$2, #35

base + displacement

lw \$1, disp(\$2)



$(R1 = M[R2 + disp])$

We get register indirect and absolute for free, but how?

register indirect

$\Rightarrow disp = 0$

absolute

$\Rightarrow (rs) = 0$

Memory Organization

0	32 bits of data
4	32 bits of data
8	32 bits of data
12	32 bits of data

Registers hold 32 bits of data

- What is a word?
- What does it mean to be word aligned?
- What are the values of the last two bits of a word aligned address?

Performance

$$\text{Speedup (X/Y)} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n$$

Performance

- Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?
- Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

Performance

- Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

Speedup (X/Y) =

- Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

Performance

- Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

X = machine B

Speedup (X/Y) =

Y = machine A

- Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

Performance

- Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

$$\begin{array}{l} \text{X = machine B} \\ \text{Speedup (X/Y) = } \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n \end{array} \quad \begin{array}{l} 9 \\ 6 \end{array} \quad \text{1.5x}$$

- Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

Performance

- Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

$$\begin{array}{l} \text{X = machine B} \\ \text{Speedup (X/Y) = } \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n \end{array} \quad \begin{array}{l} 9 \\ 6 \end{array} \quad \text{1.5x}$$

- Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

$$\begin{array}{l} \text{X = machine B} \\ \text{Speedup (X/Y) = } \\ \text{Y = machine A} \end{array}$$

Performance

- Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

$$\begin{array}{l} \text{X = machine B} \\ \text{Speedup (X/Y) = } \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n \end{array} \quad \begin{array}{c} 9 \\ 6 \end{array} \quad \text{1.5x}$$

- Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

$$\begin{array}{l} \text{X = machine B} \\ \text{Speedup (X/Y) = } \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n \end{array} \quad \begin{array}{c} 9 \\ 3 \end{array} \quad \text{3x}$$

Performance - Execution Time

- Assume that we have an application composed with a total of **500,000** instructions, in which **20%** of them are the load/store instructions with an average CPI of **6** cycles, and the rest of the instructions are integer instructions with average CPI of **1** cycle.

$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

Performance - Execution Time

- Assume that we have an application composed with a total of **500,000** instructions, in which **20%** of them are the load/store instructions with an average CPI of **6** cycles, and the rest of the instructions are integer instructions with average CPI of **1** cycle.

$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

500,000 *

Performance - Execution Time

- Assume that we have an application composed with a total of **500,000** instructions, in which **20%** of them are the load/store instructions with an average CPI of **6** cycles, and the rest of the instructions are integer instructions with average CPI of **1** cycle.

$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$500,000 * (.2 * 6 + .8 * 1) *$$

Performance - Execution Time

- Assume that we have an application composed with a total of **500,000** instructions, in which **20%** of them are the load/store instructions with an average CPI of **6** cycles, and the rest of the instructions are integer instructions with average CPI of **1** cycle.

$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$\begin{aligned} & 500,000 * \\ & (.2 * 6 + .8 * 1) * \\ & 1 \text{ ns} = \end{aligned}$$

Performance - Execution Time

- Assume that we have an application composed with a total of **500,000** instructions, in which **20%** of them are the load/store instructions with an average CPI of **6** cycles, and the rest of the instructions are integer instructions with average CPI of **1** cycle.

$$\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$\begin{aligned} & 500,000 * \\ & (.2 * 6 + .8 * 1) * \\ & 1 \text{ ns} = \\ & 1,000,000 \text{ ns} \end{aligned}$$

Amdahl's Law

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$



Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of it's time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of it's time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\text{Speedup} =$$

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\text{Speedup} = \frac{1}{1 - 0.25 + \frac{0.25}{10}}$$

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\text{Speedup} = \frac{1}{\quad}$$

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\text{Speedup} = \frac{1}{(1 - 0.25)}$$

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}$$

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}$$

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of it's time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}$$

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}$$

Amdahl's Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

$$\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

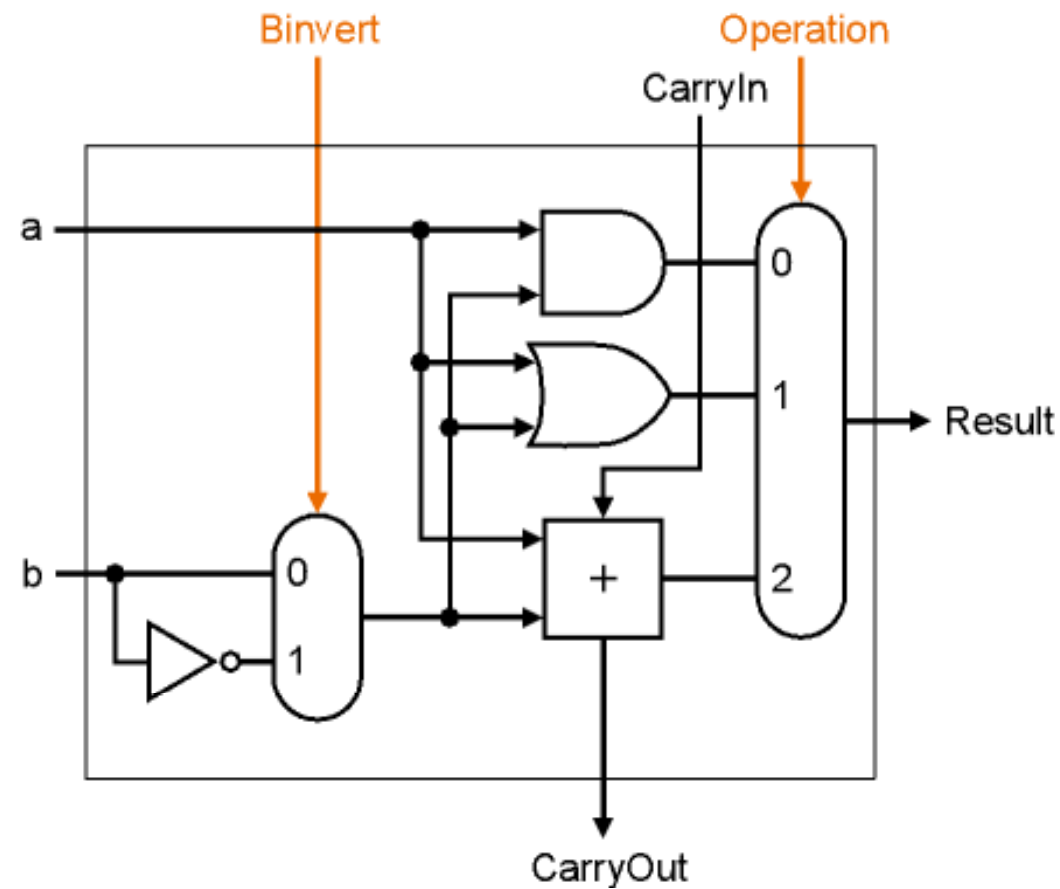
$$\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}} = 1.29$$

ALU - Two's Complement Representation

- 2's complement representation of negative numbers
- Take the bitwise inverse and add 1
- Biggest 4-bit Binary Number: 7 Smallest 4-bit Binary Number: -8

<u>Decimal</u>	<u>Two's Complement Binary</u>
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

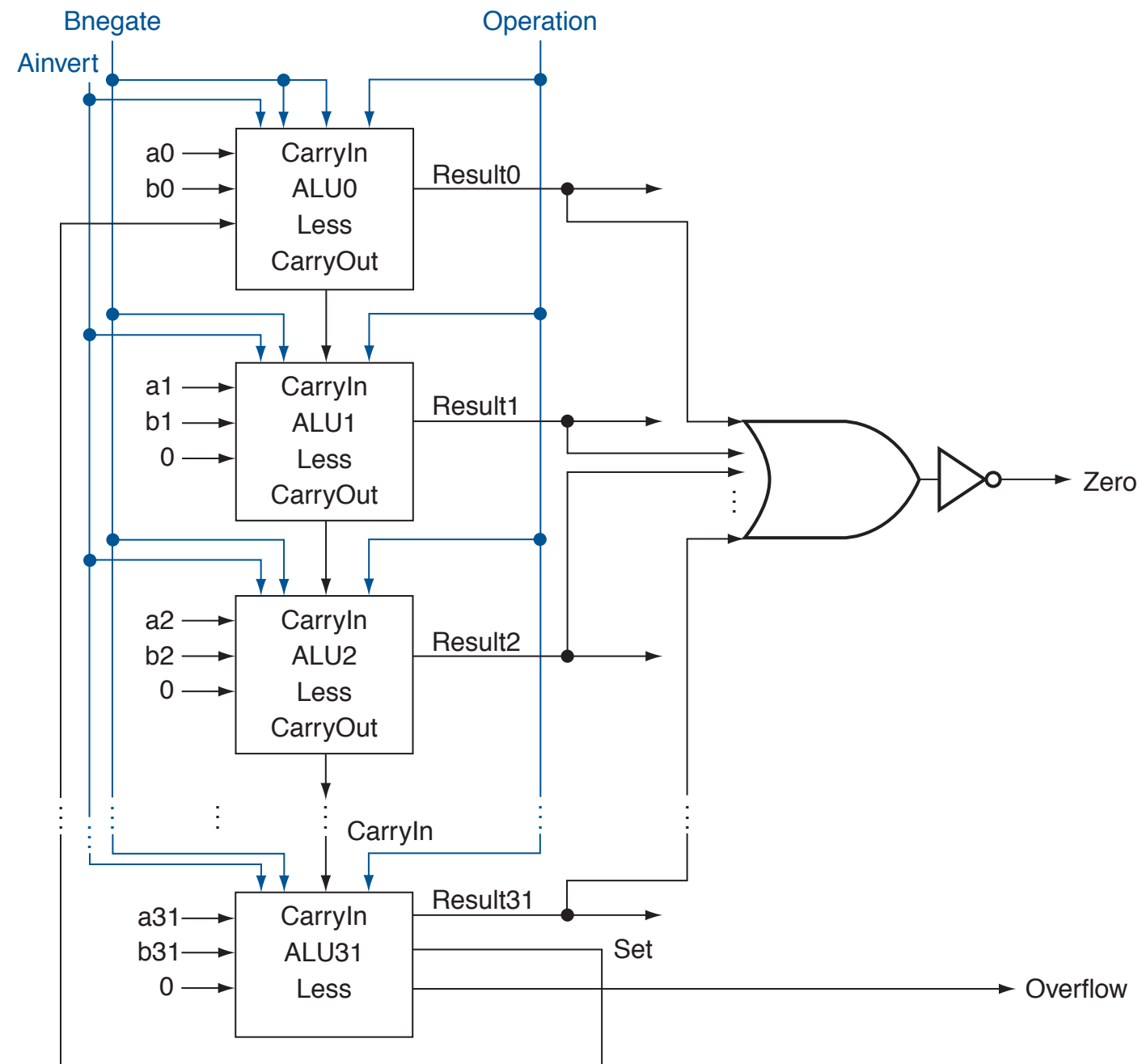
ALU



- How many bits does this ALU support?
- What does Binvert do?
- What about CarryIn, CarryOut?
- How do we make an ALU that supports 32 bits?

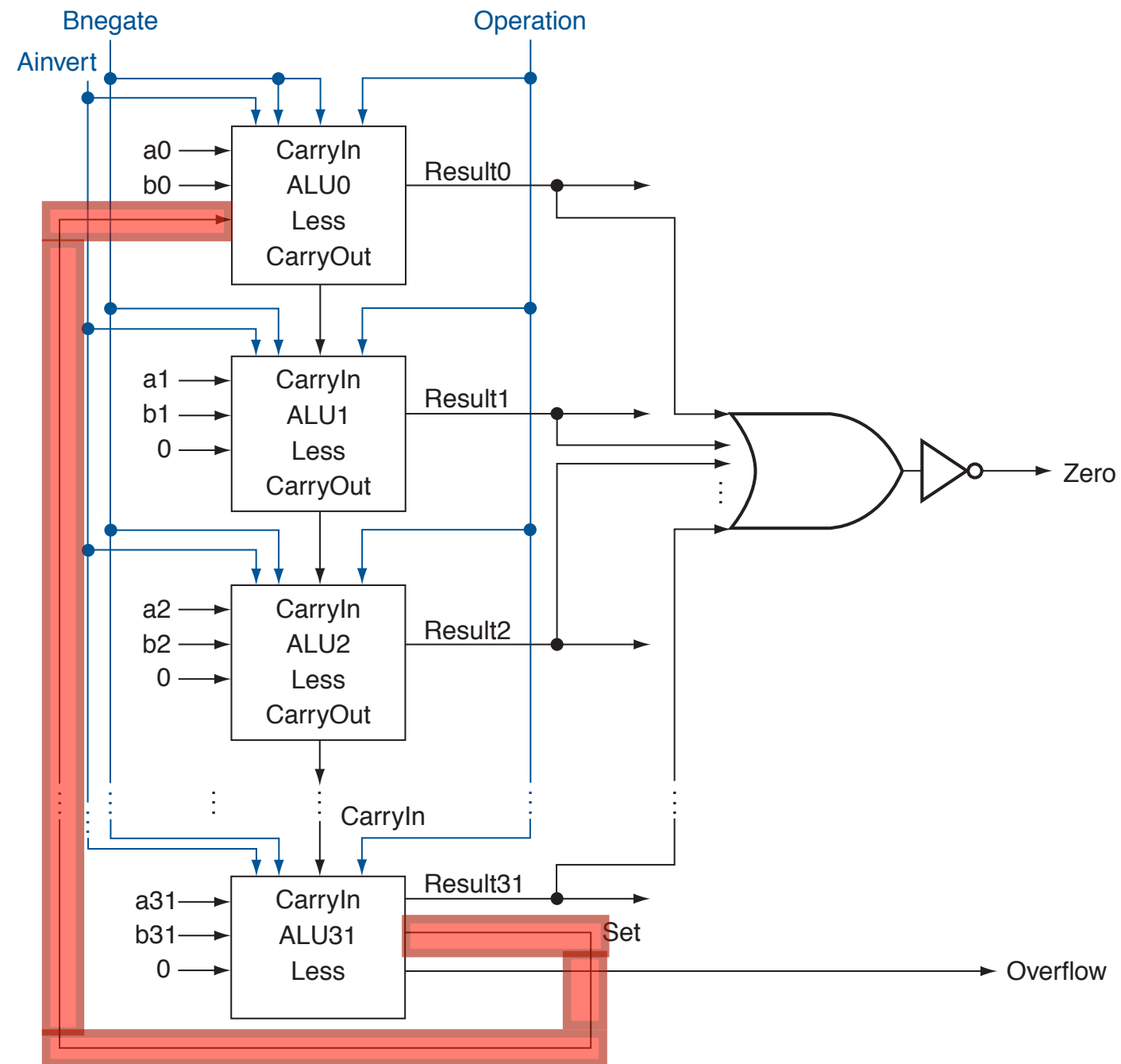
ALU

What is...



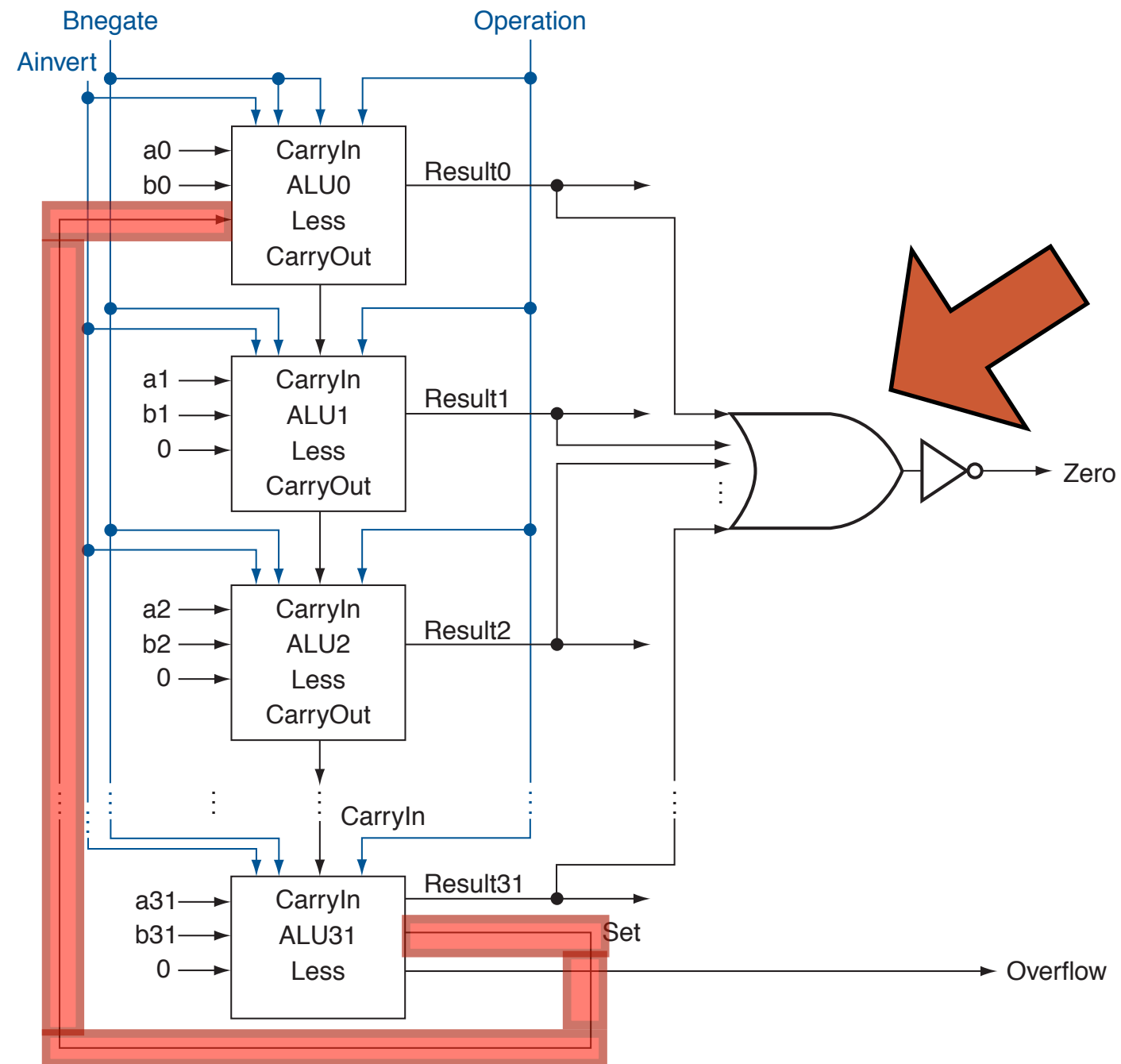
ALU

What is...

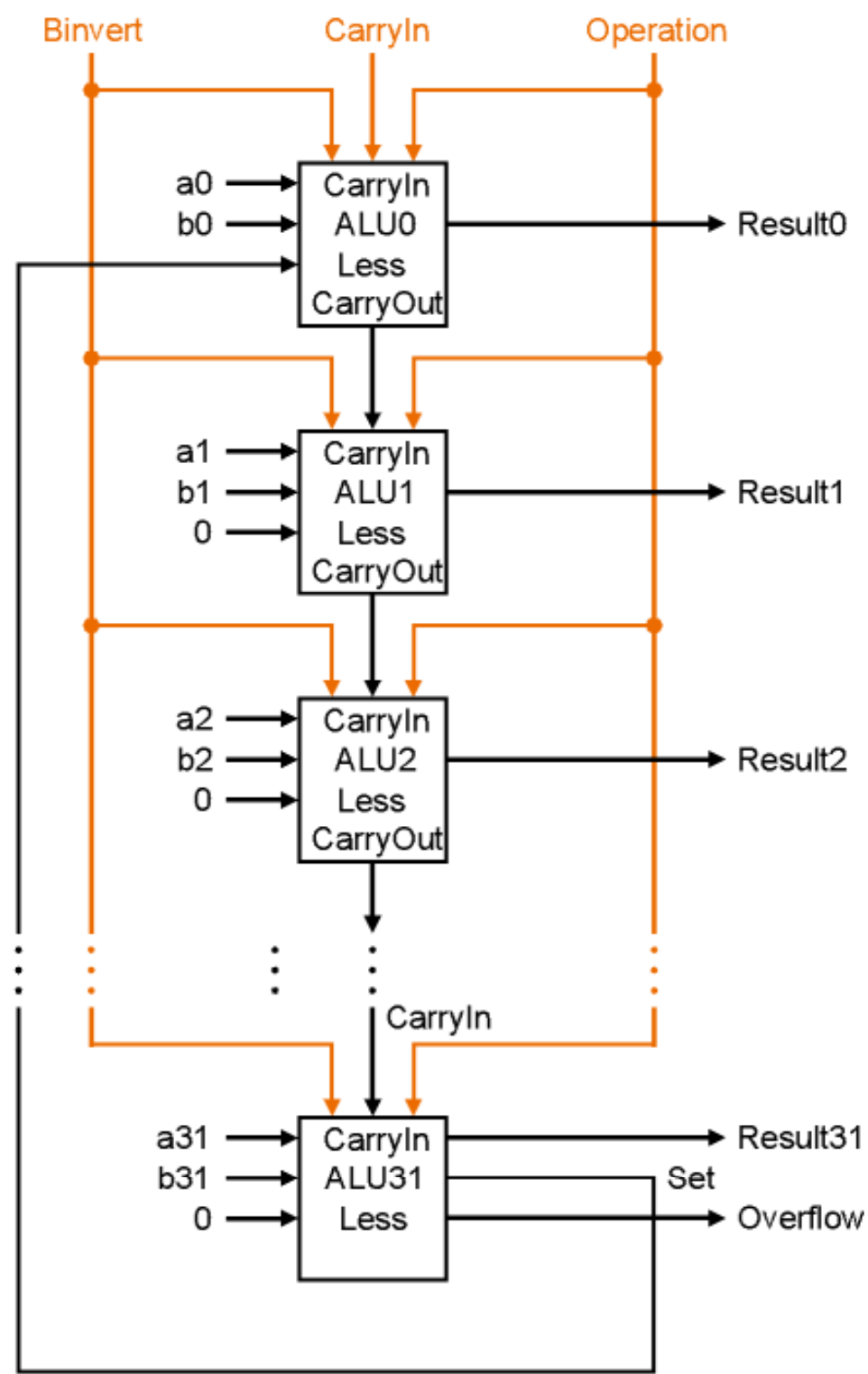
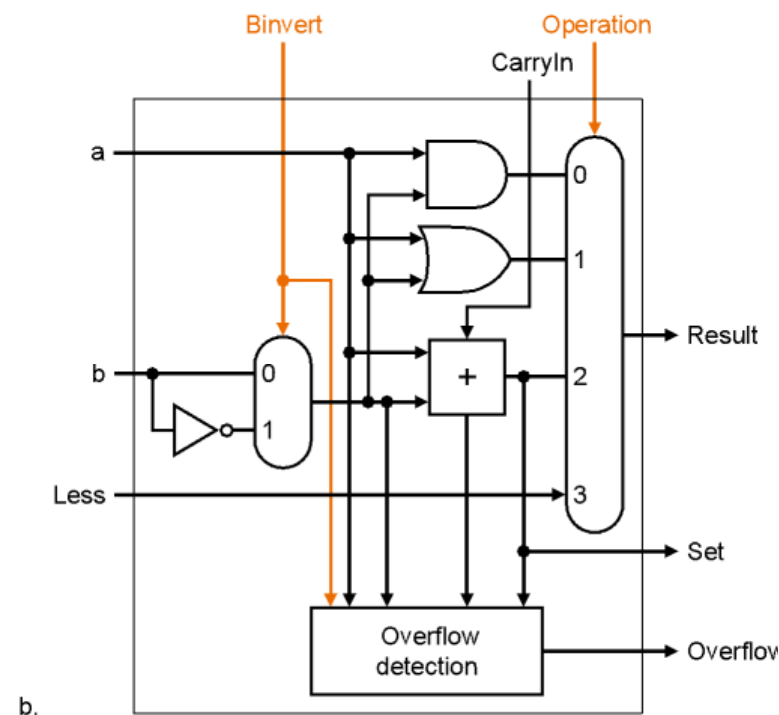
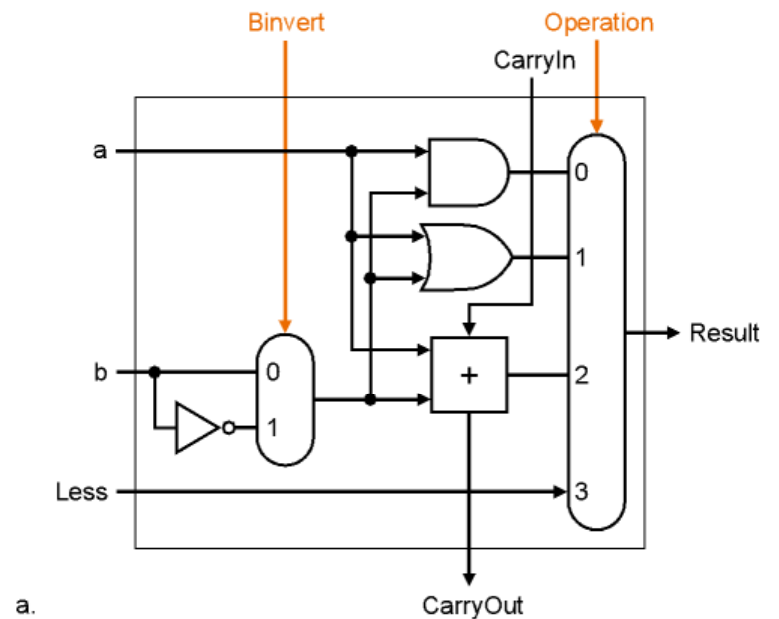


ALU

What is...

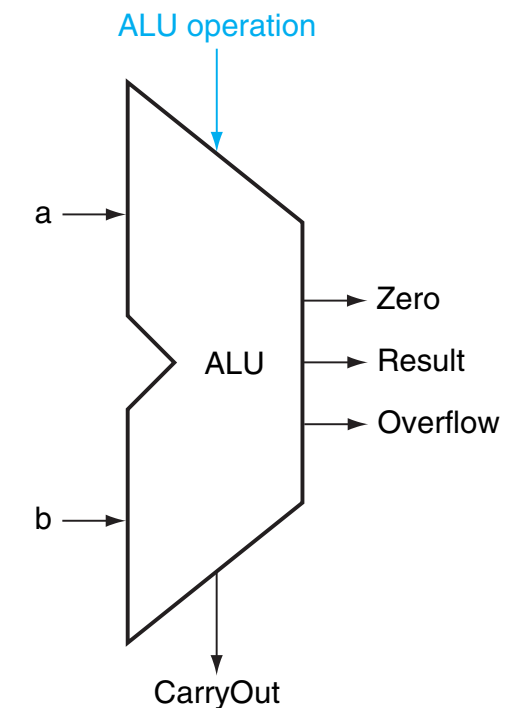


Full ALU

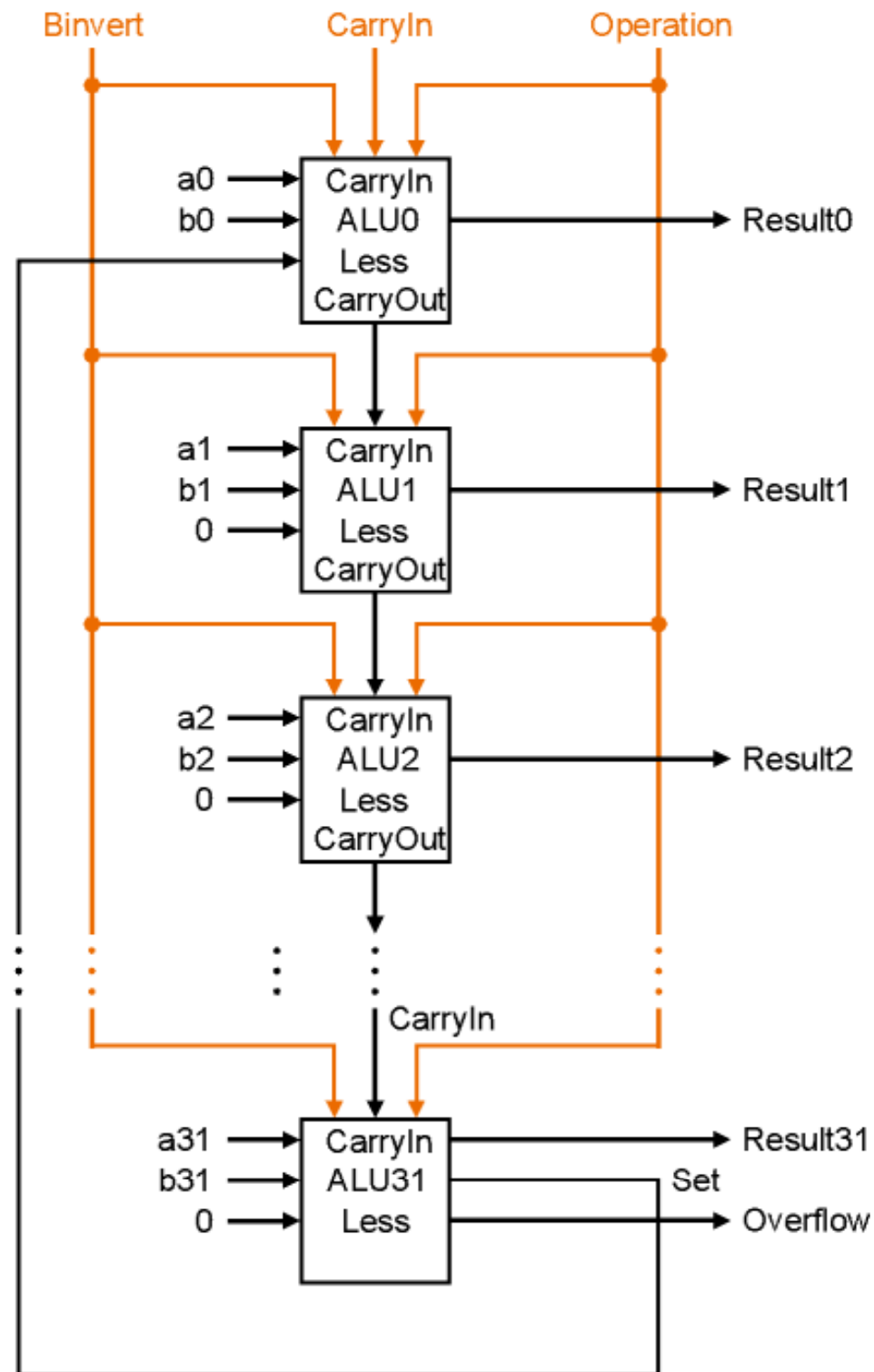
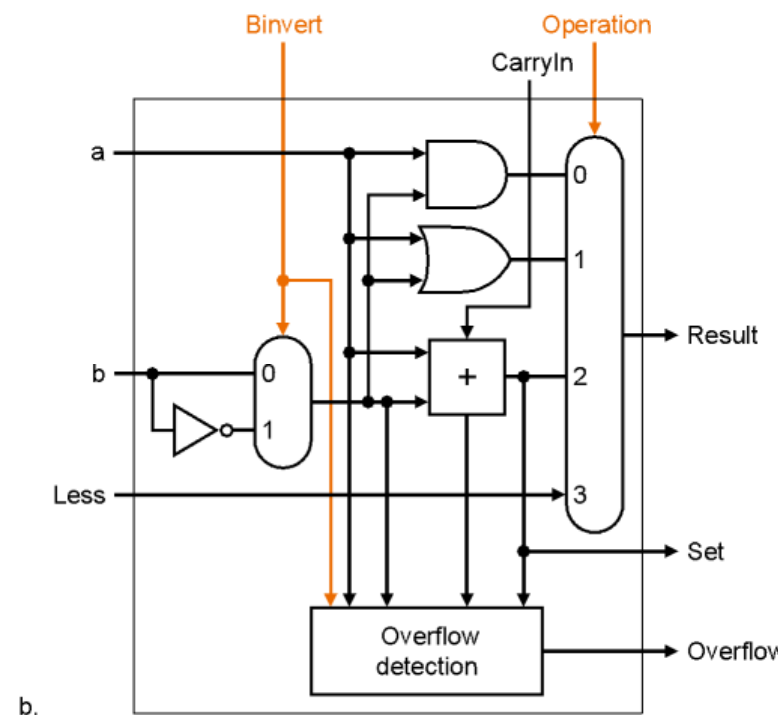
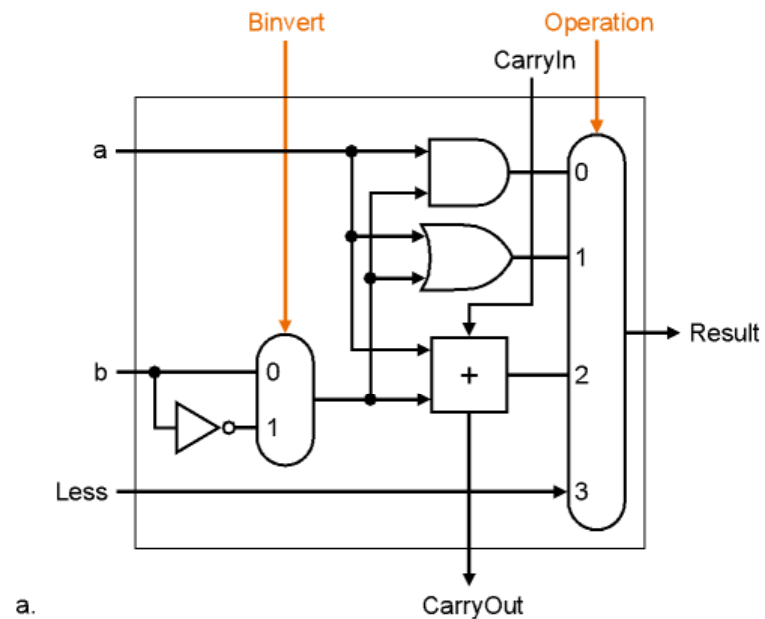


what signals accomplish:
Binvert CIn Oper

add?
 sub?
 and?
 or?
 beq?
 slt?



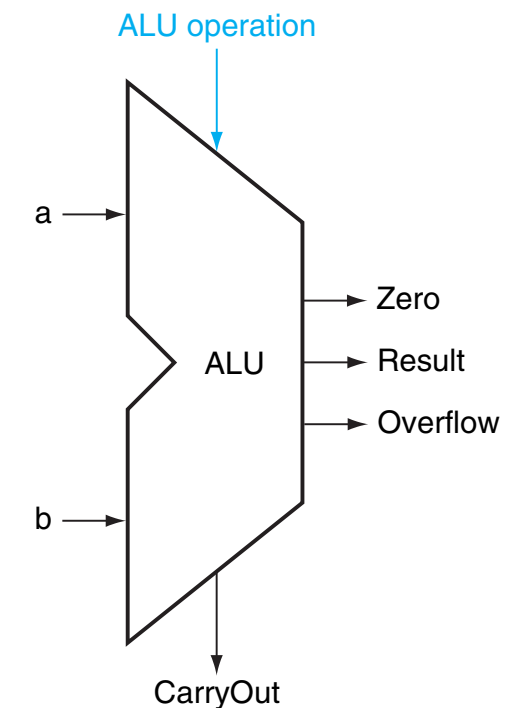
Full ALU



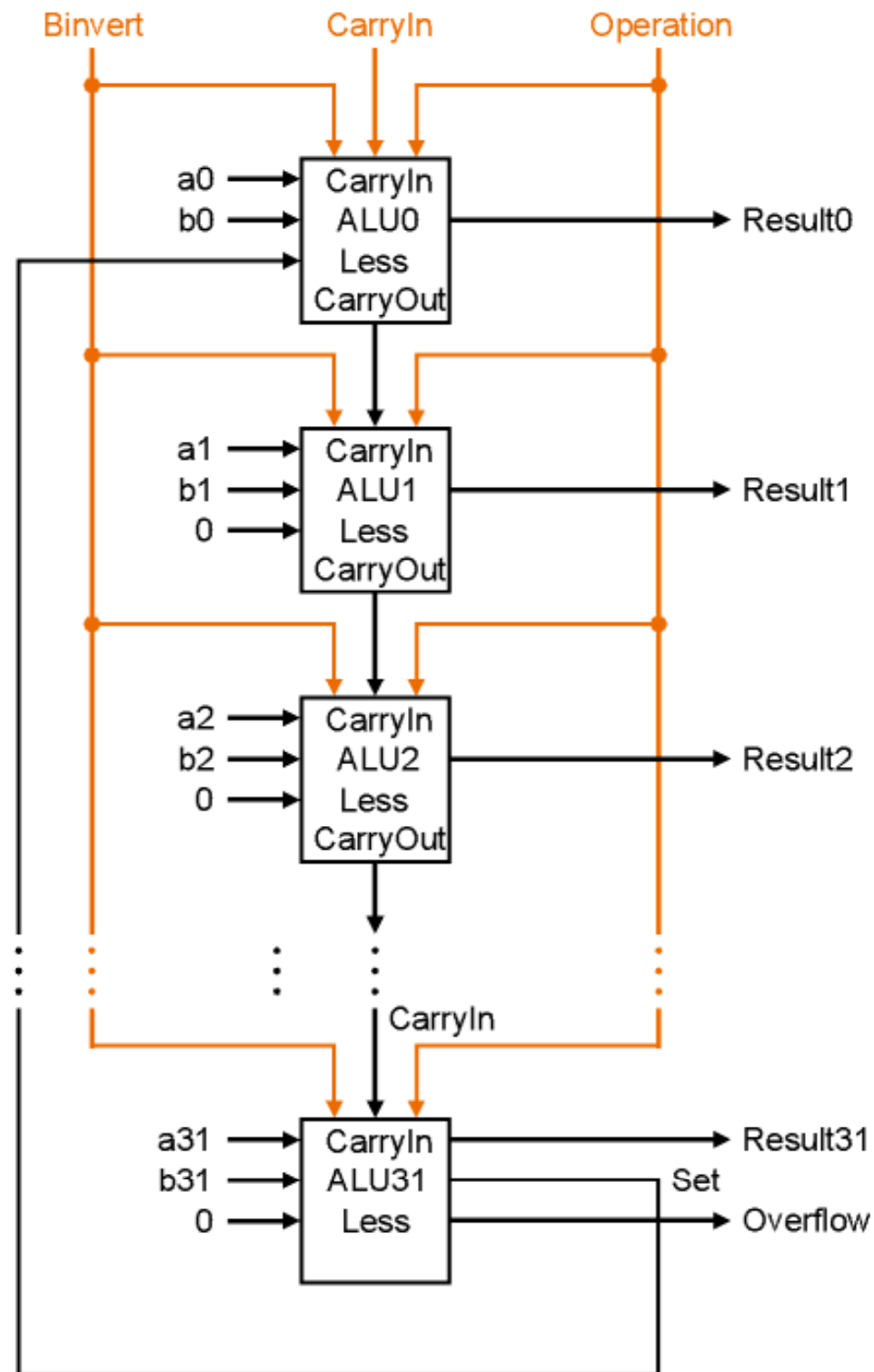
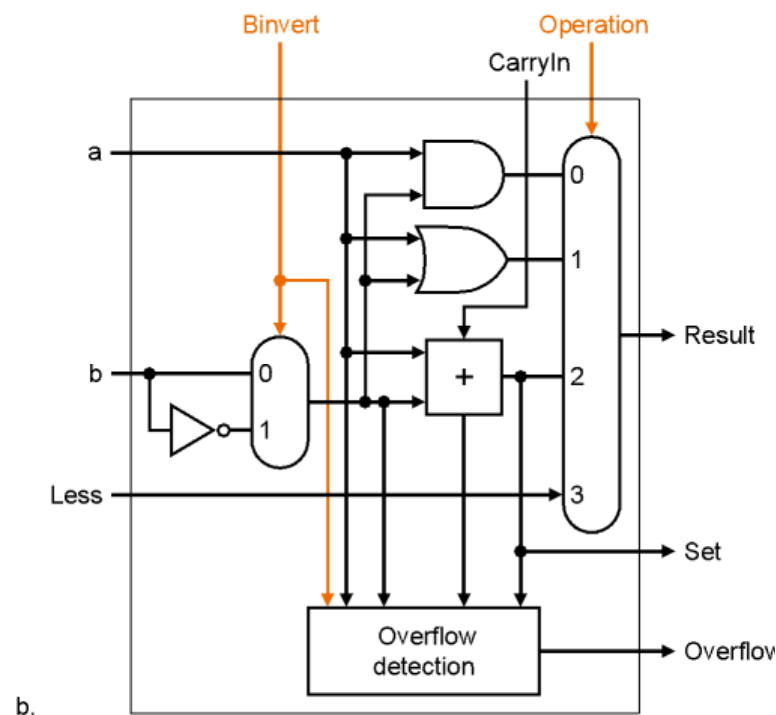
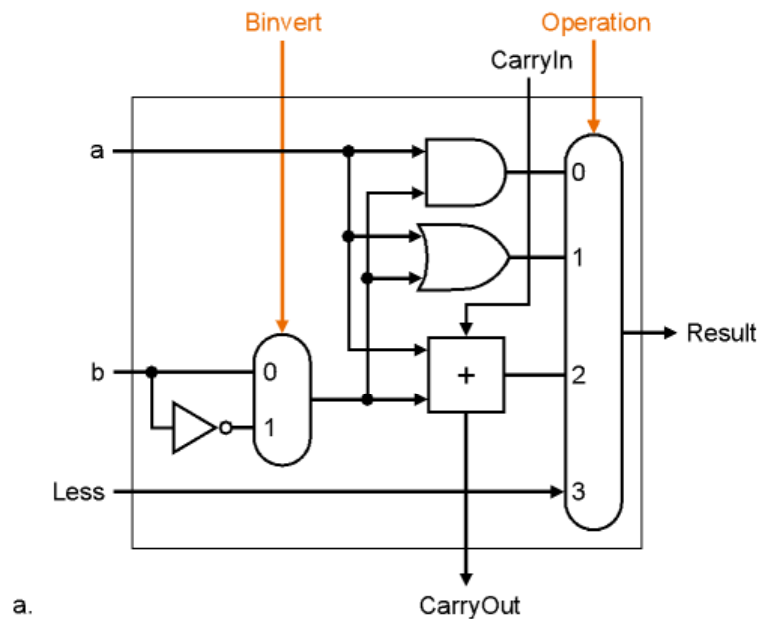
what signals accomplish:

Binvert	CIn	Oper
0	0	10

add?
sub?
and?
or?
beq?
slt?

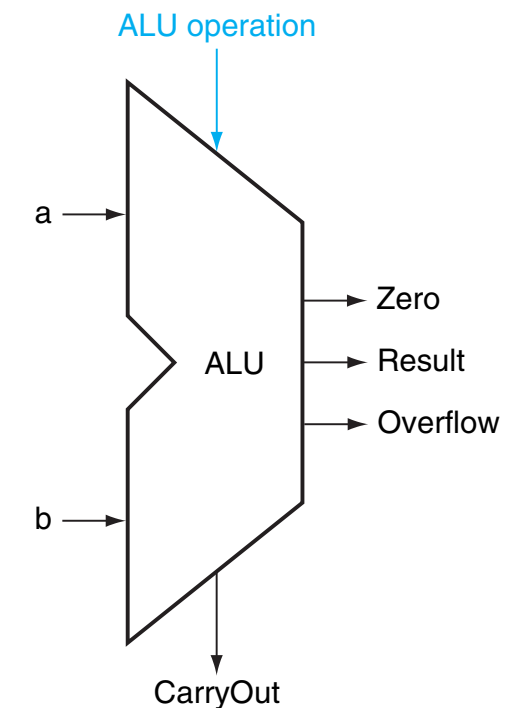


Full ALU

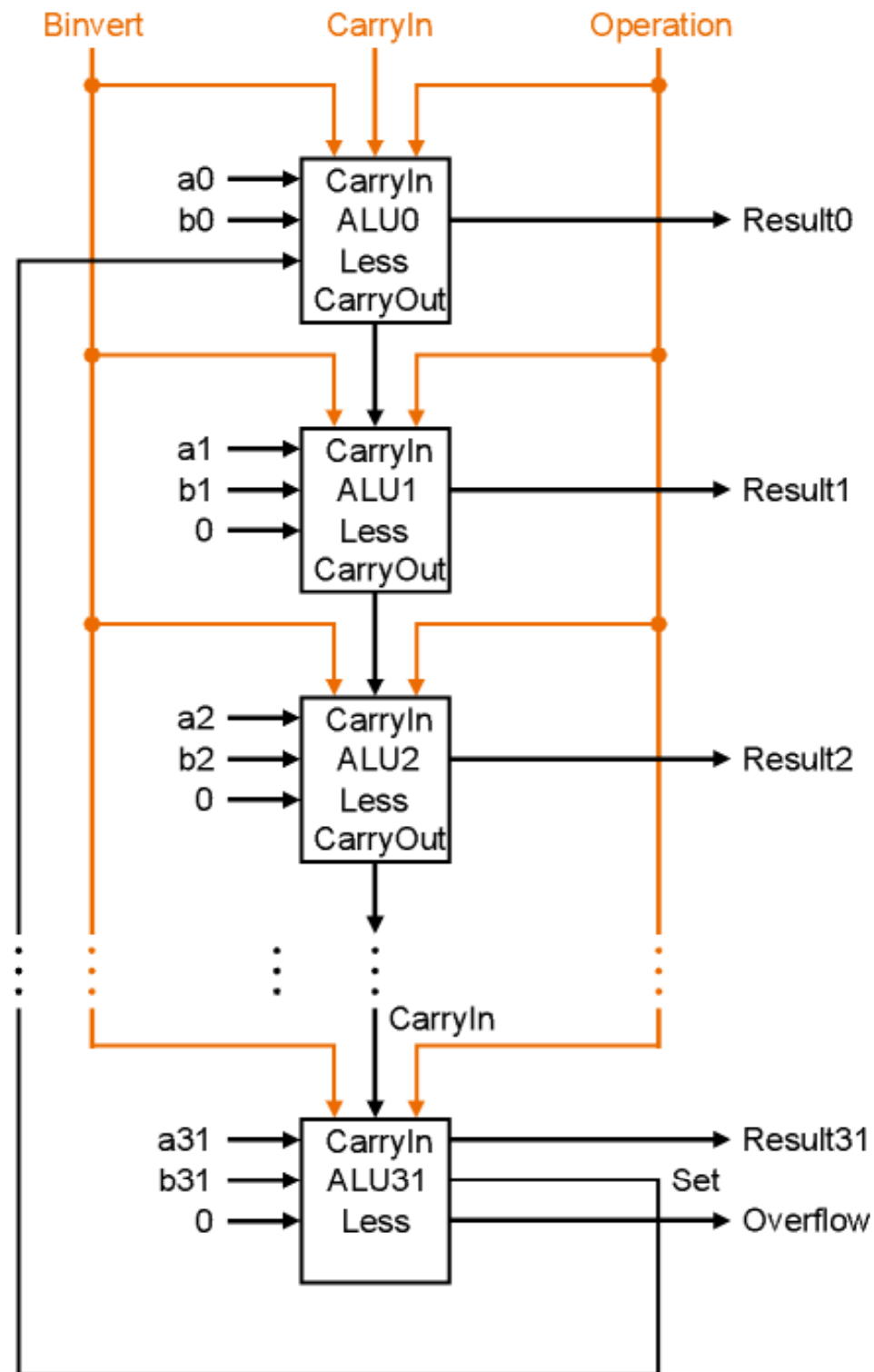
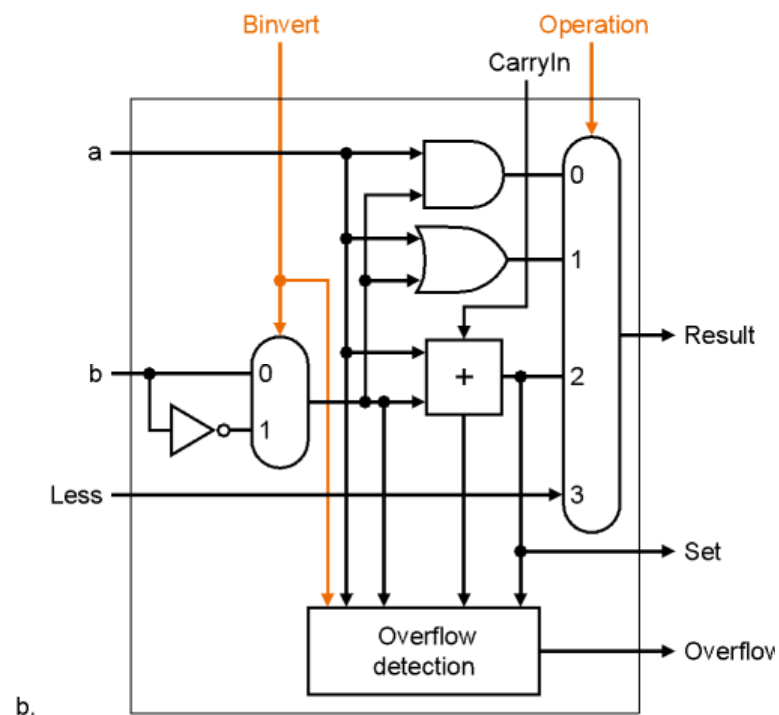
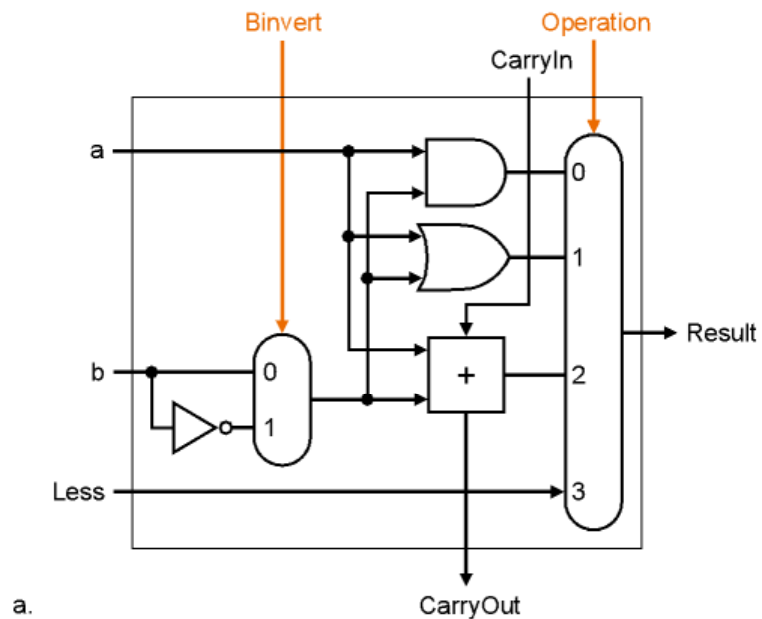


what signals accomplish:

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

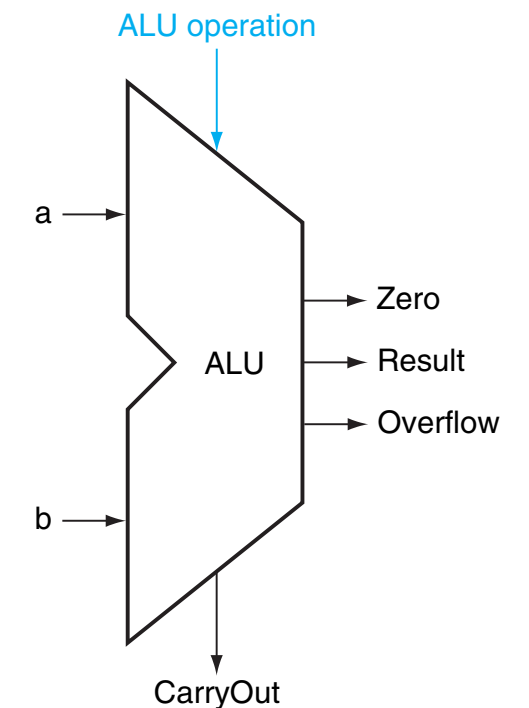


Full ALU

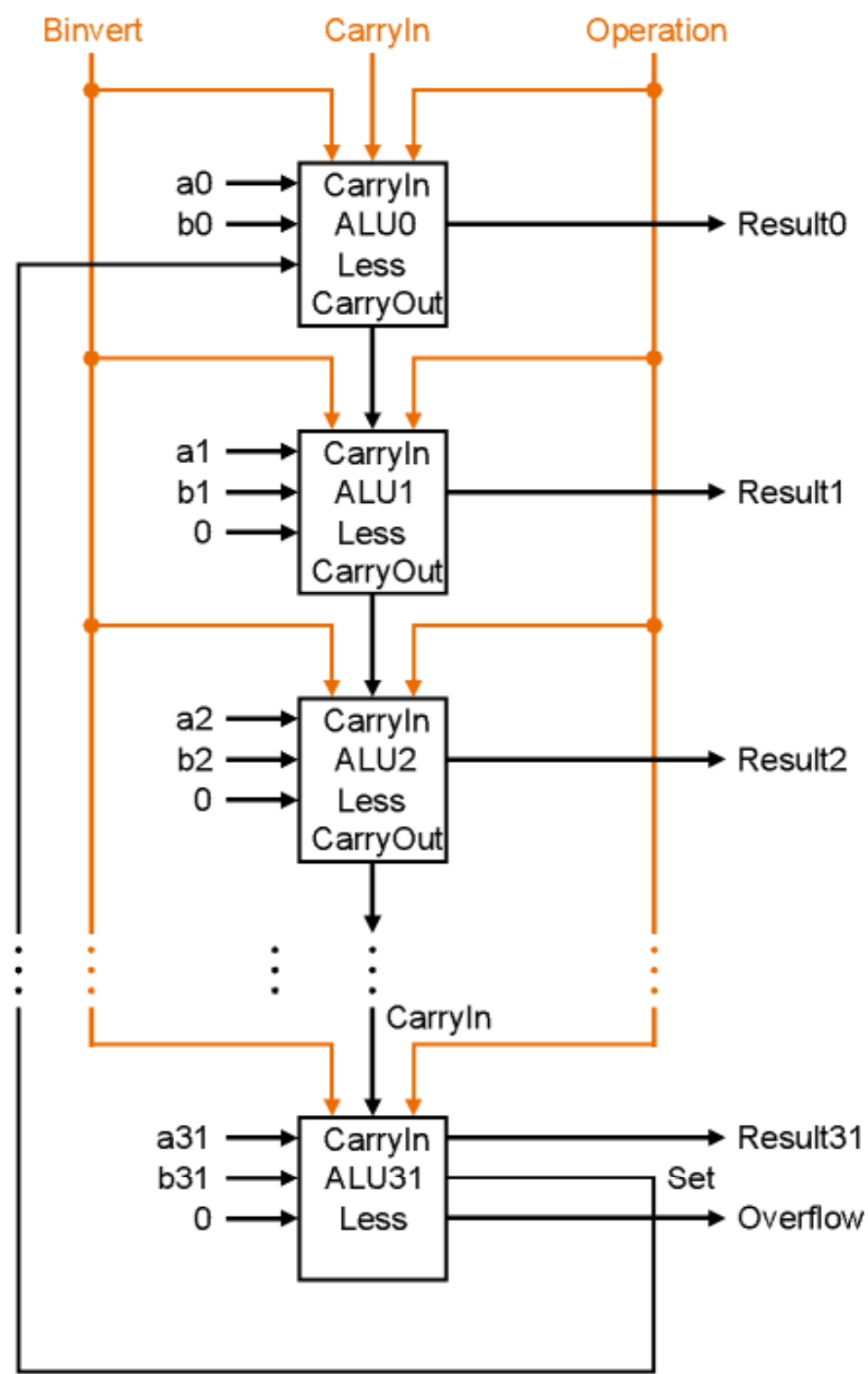
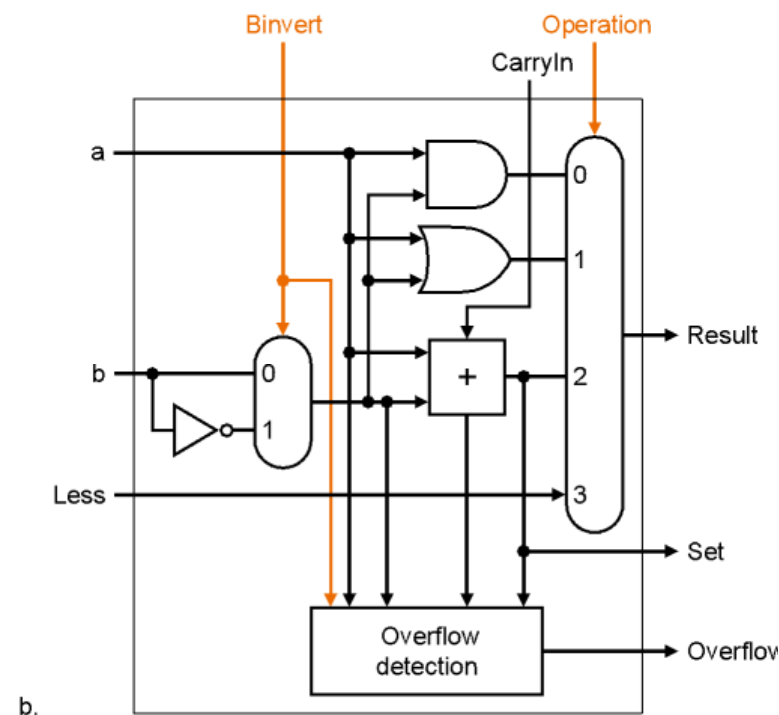
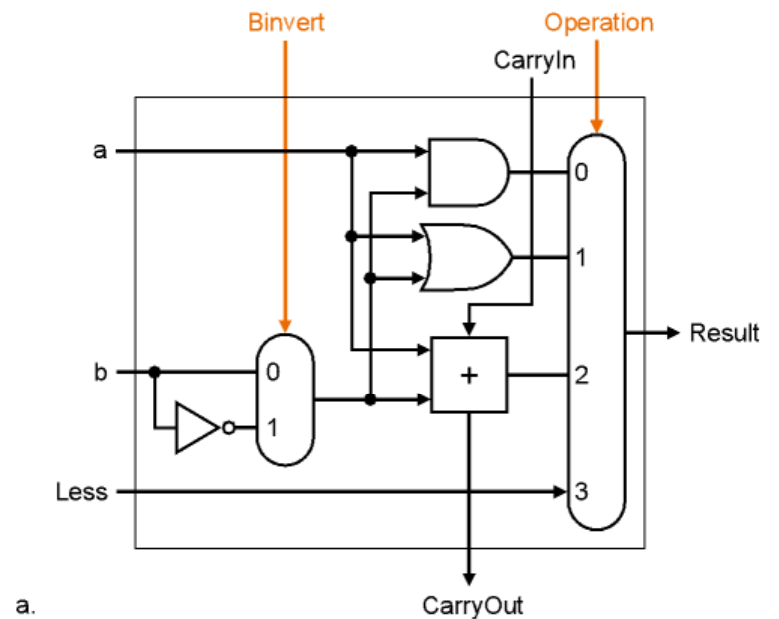


what signals accomplish:

	<u>Binvert</u>	<u>CIn</u>	<u>Oper</u>
add?	0	0	10
sub?	1	1	10
and?	0	0	00
or?			
beq?			
slt?			

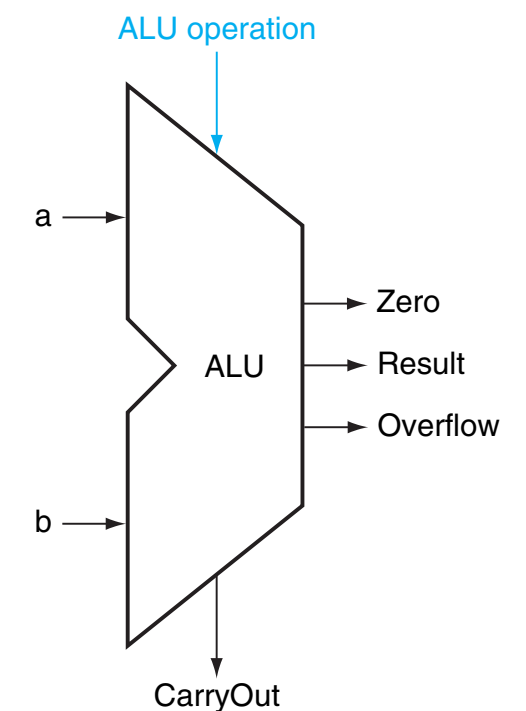


Full ALU

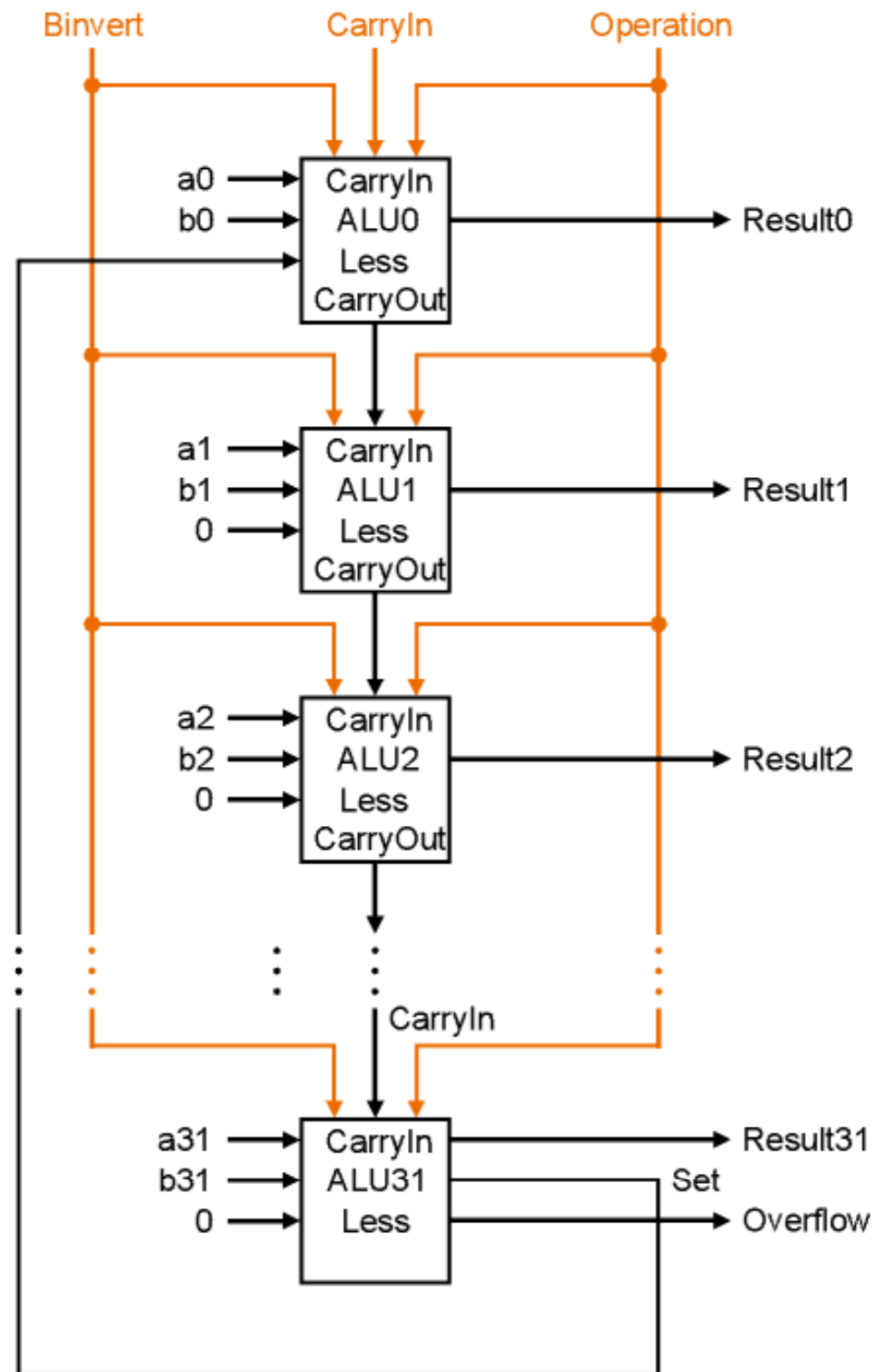
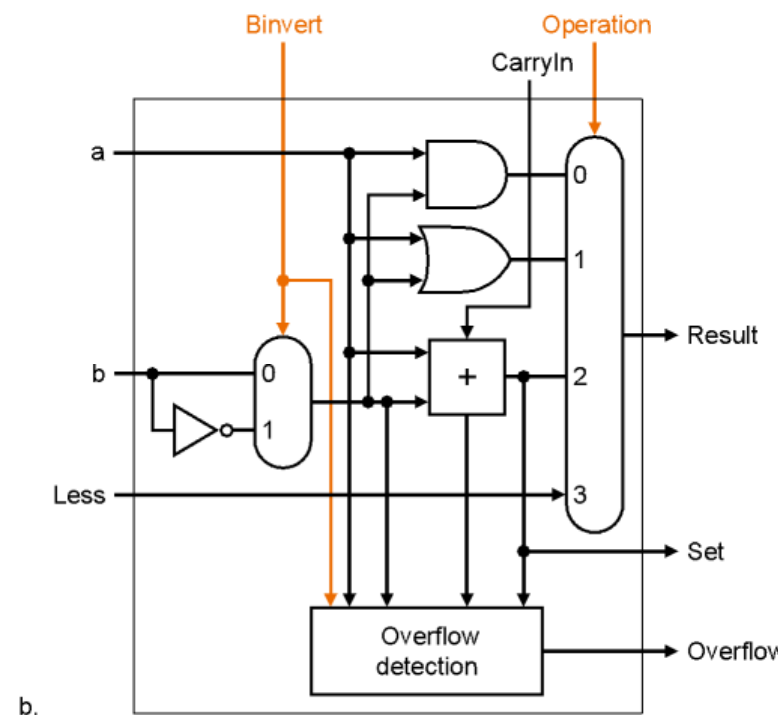
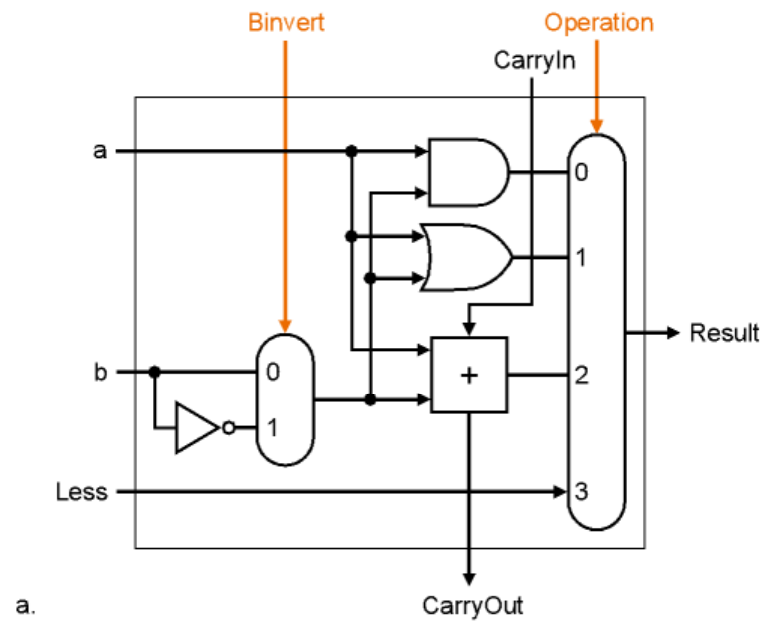


what signals accomplish:

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

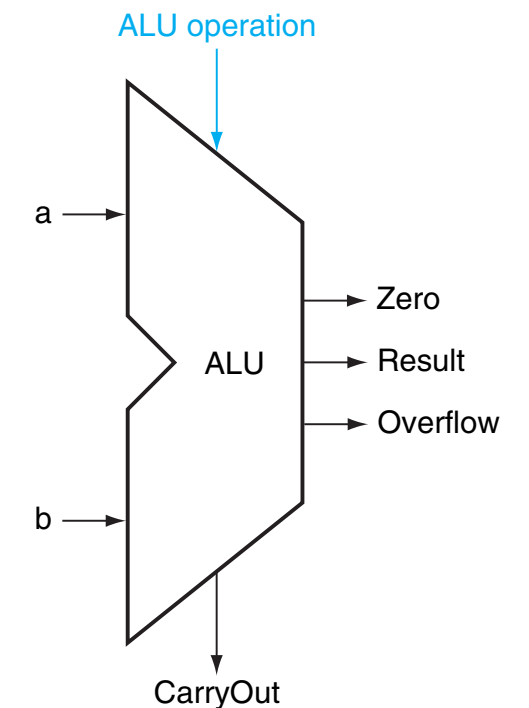


Full ALU

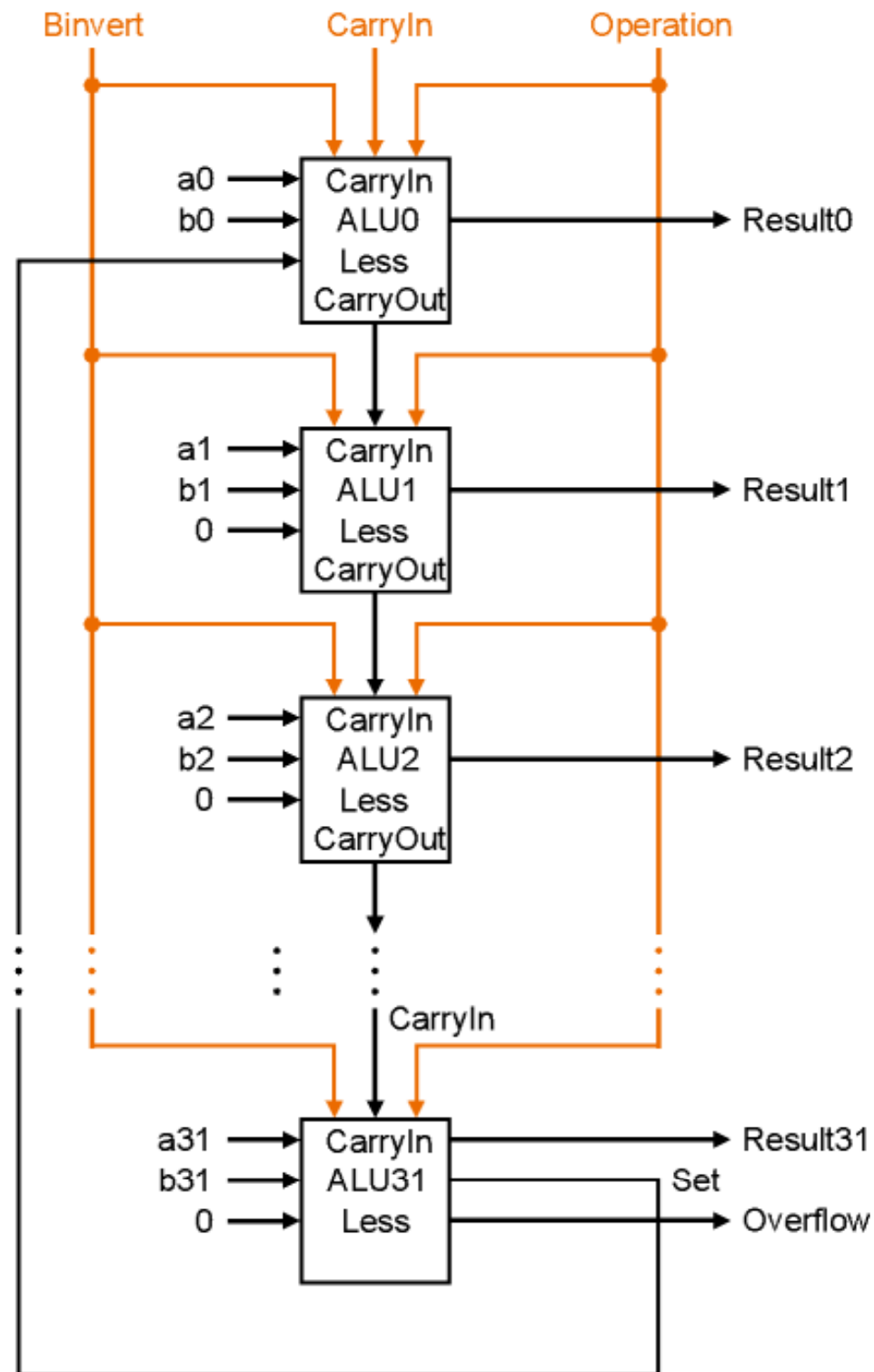
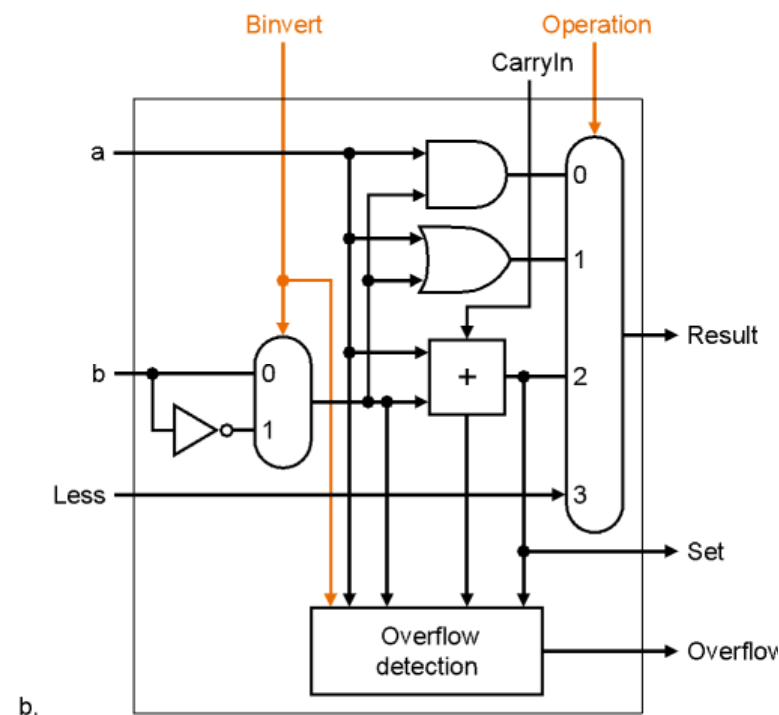
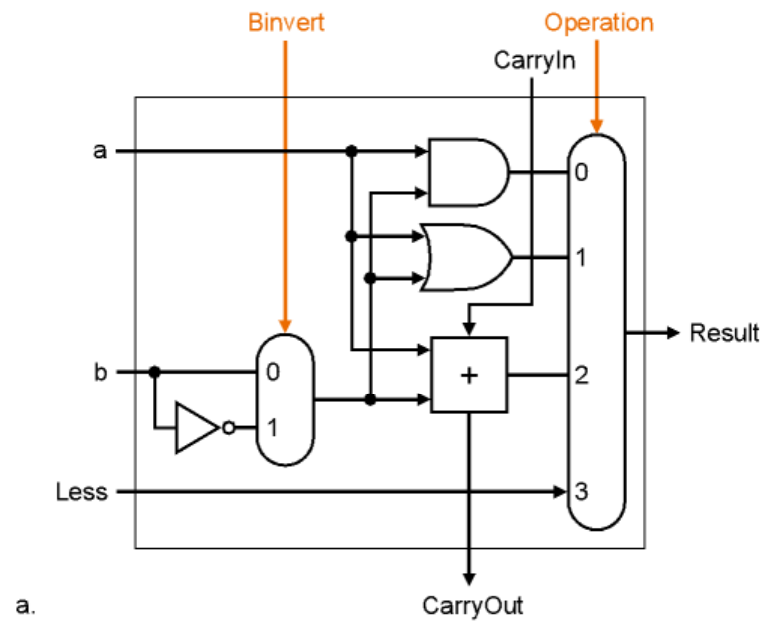


what signals accomplish:

	<u>Binvert</u>	<u>CIn</u>	<u>Oper</u>
add?	0	0	10
sub?	1	1	10
and?	0	0	00
or?	0	0	01
beq?	1	1	10
slt?			

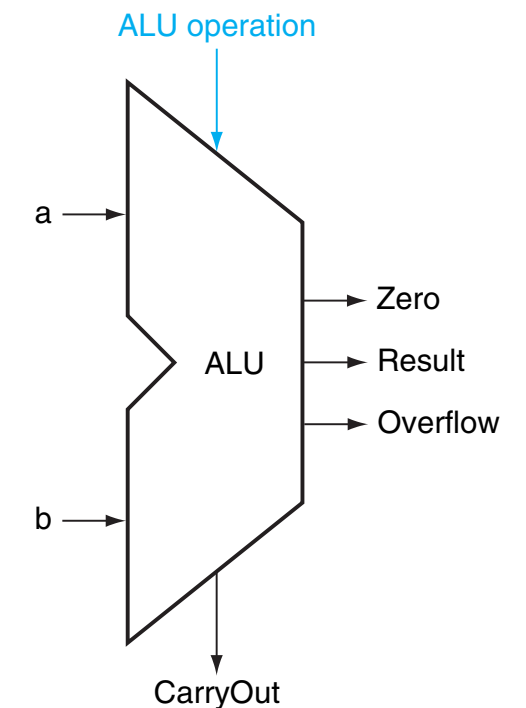


Full ALU



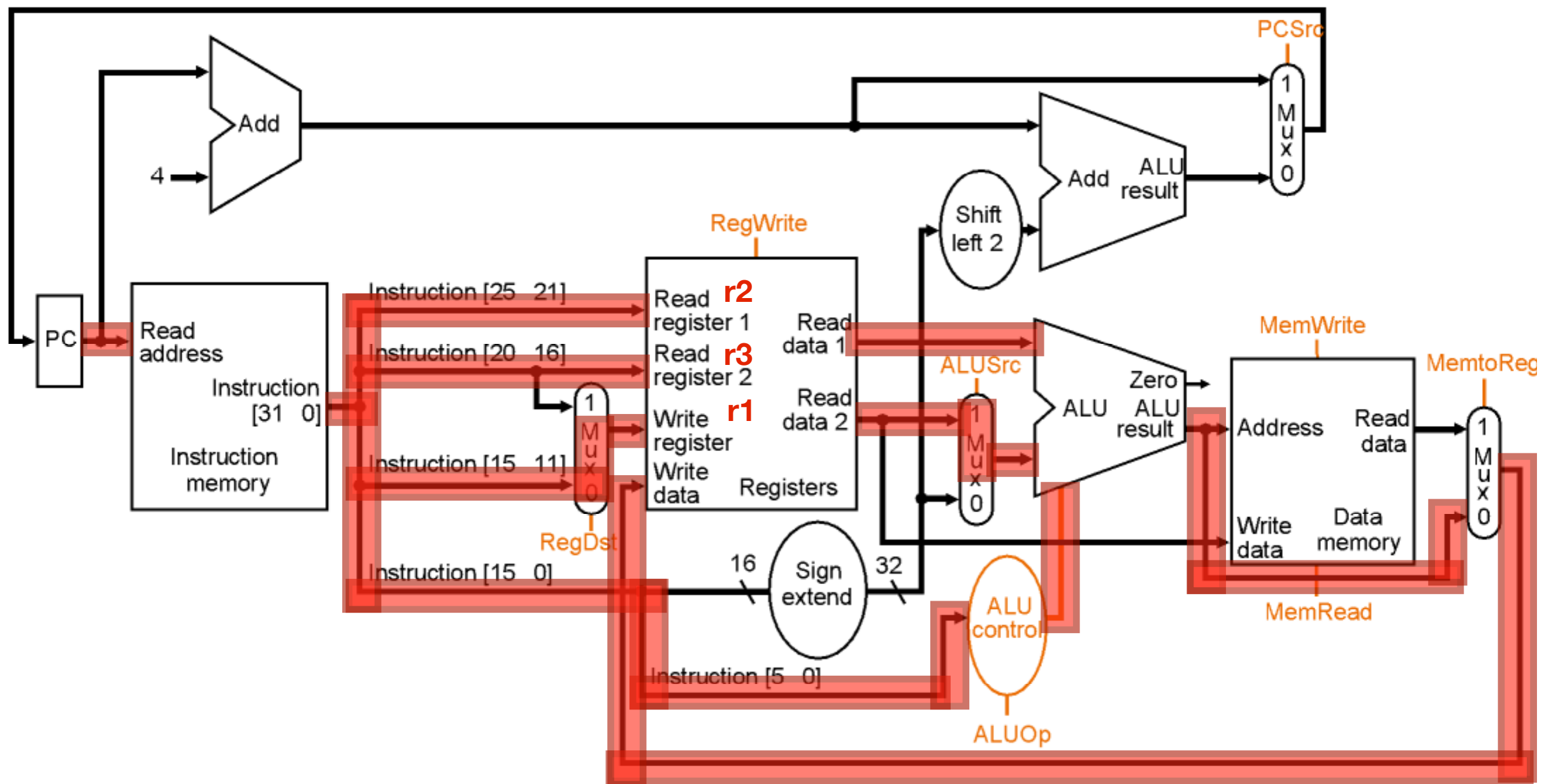
what signals accomplish:

	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



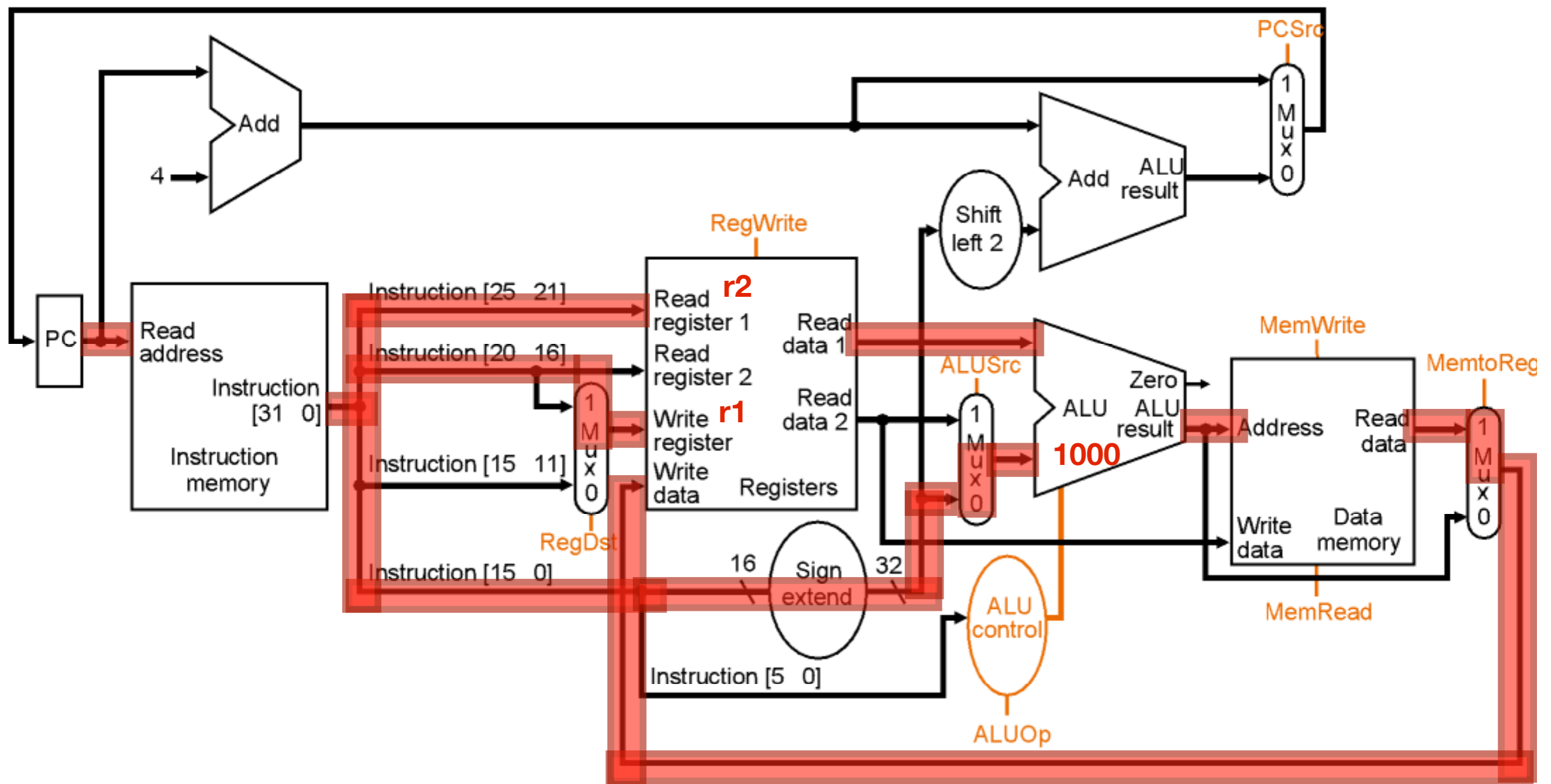
The R-Format (e.g. add) Datapath

add r1, r2, r3



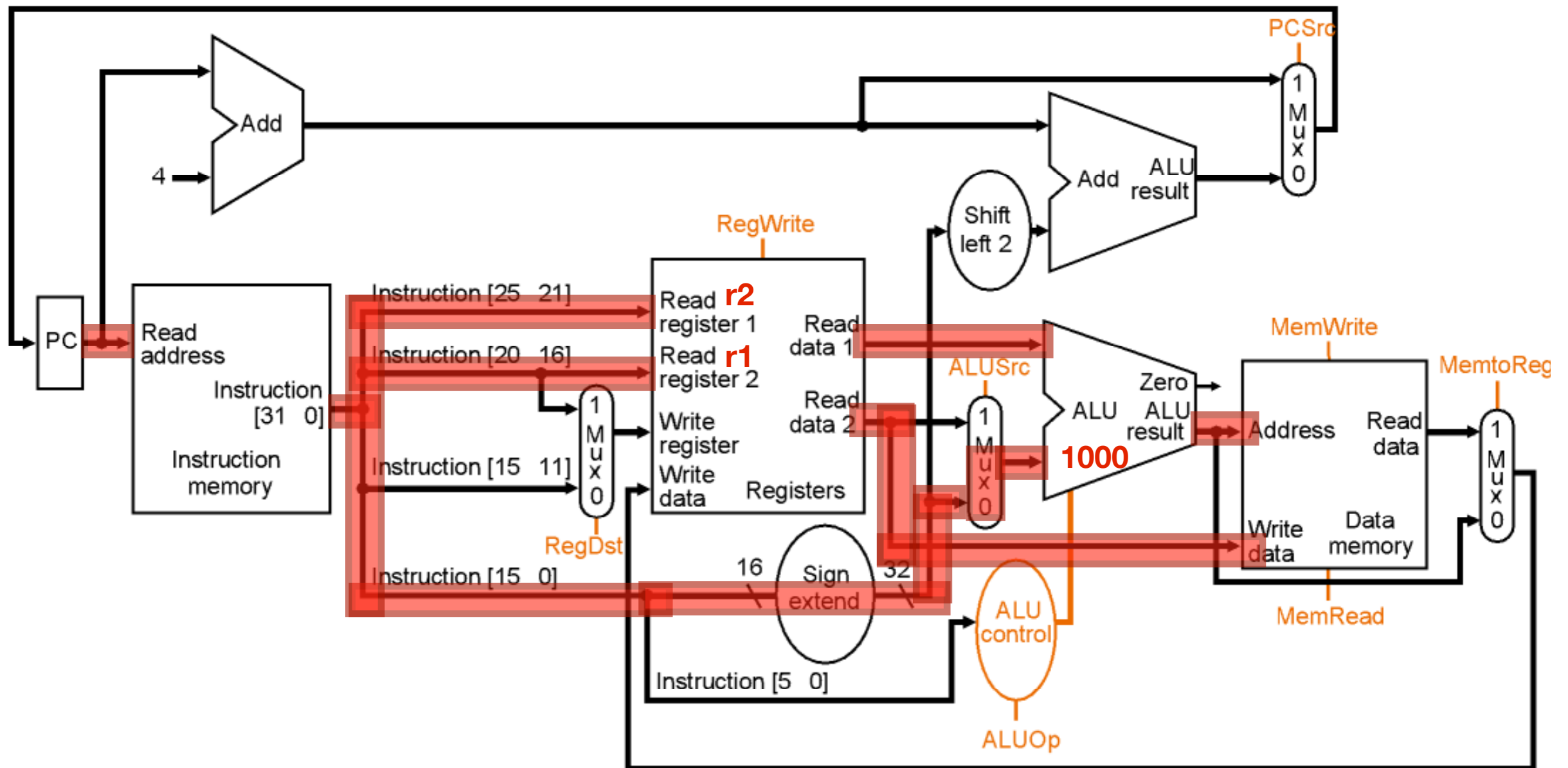
The Load Datapath

lw r1, 1000(r2)



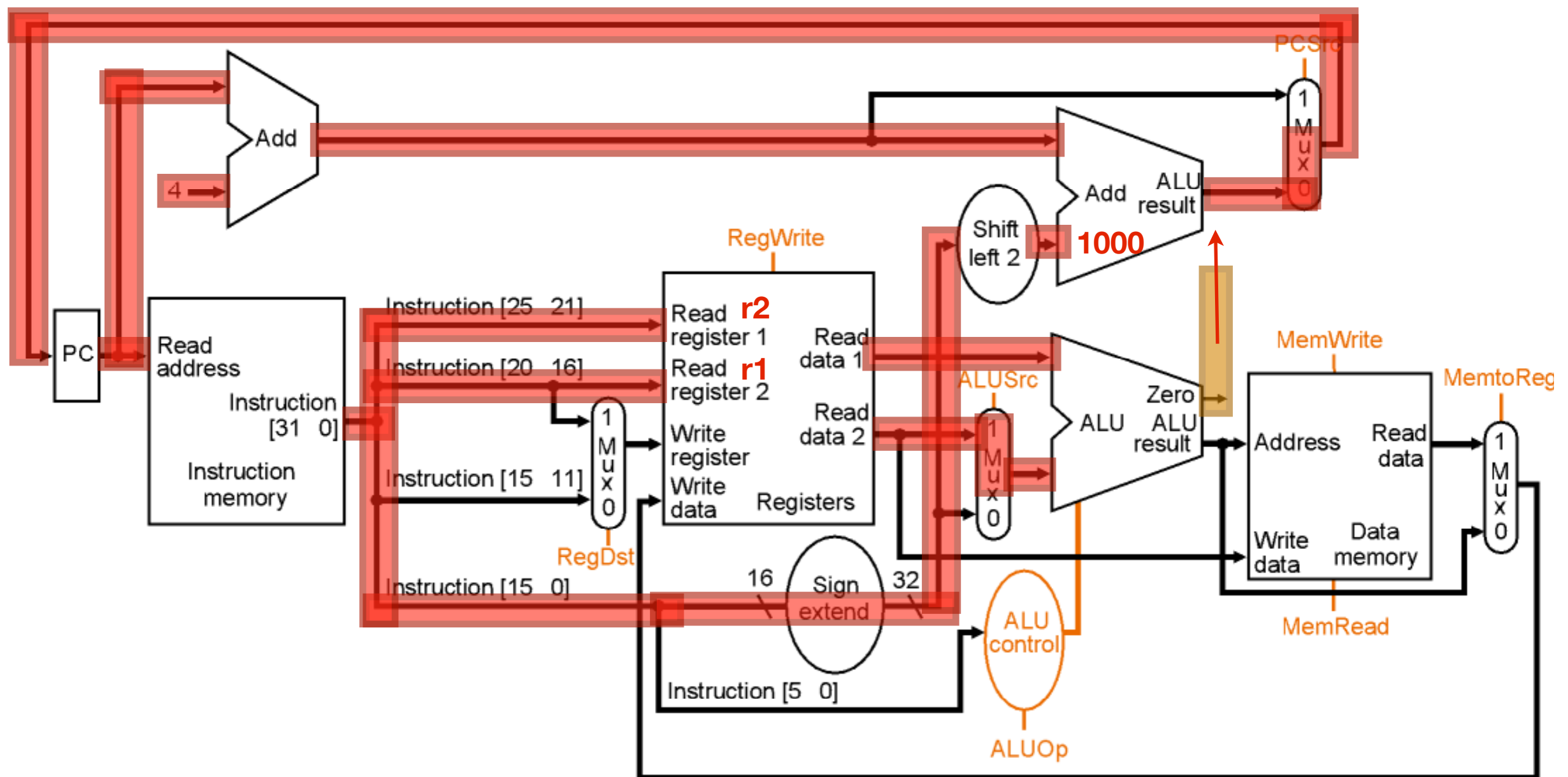
The Store Datapath

sw r1, 1000(r2)

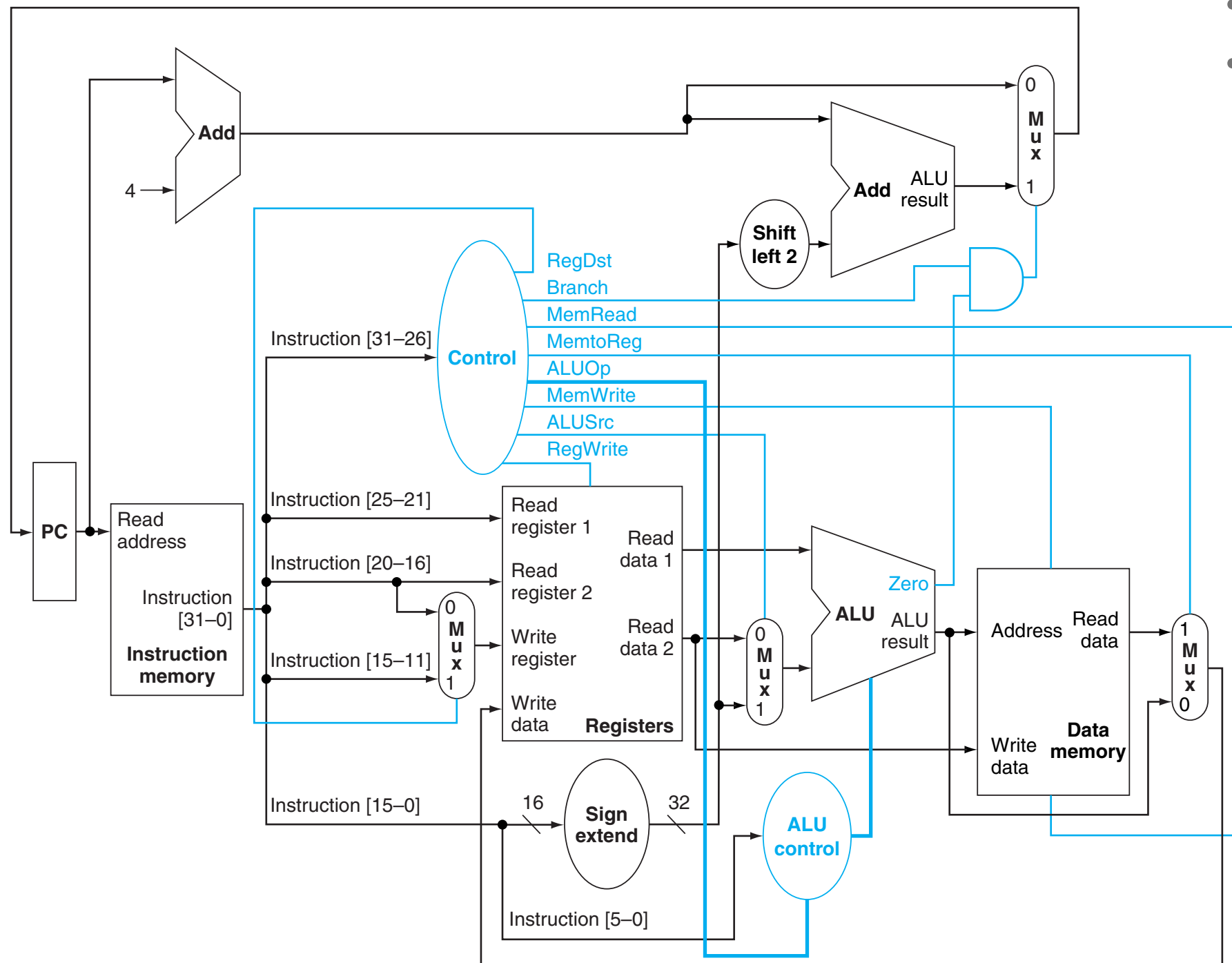


The Branch (beq) Datapath

beq r1, r2, 1000



Single Cycle CPU

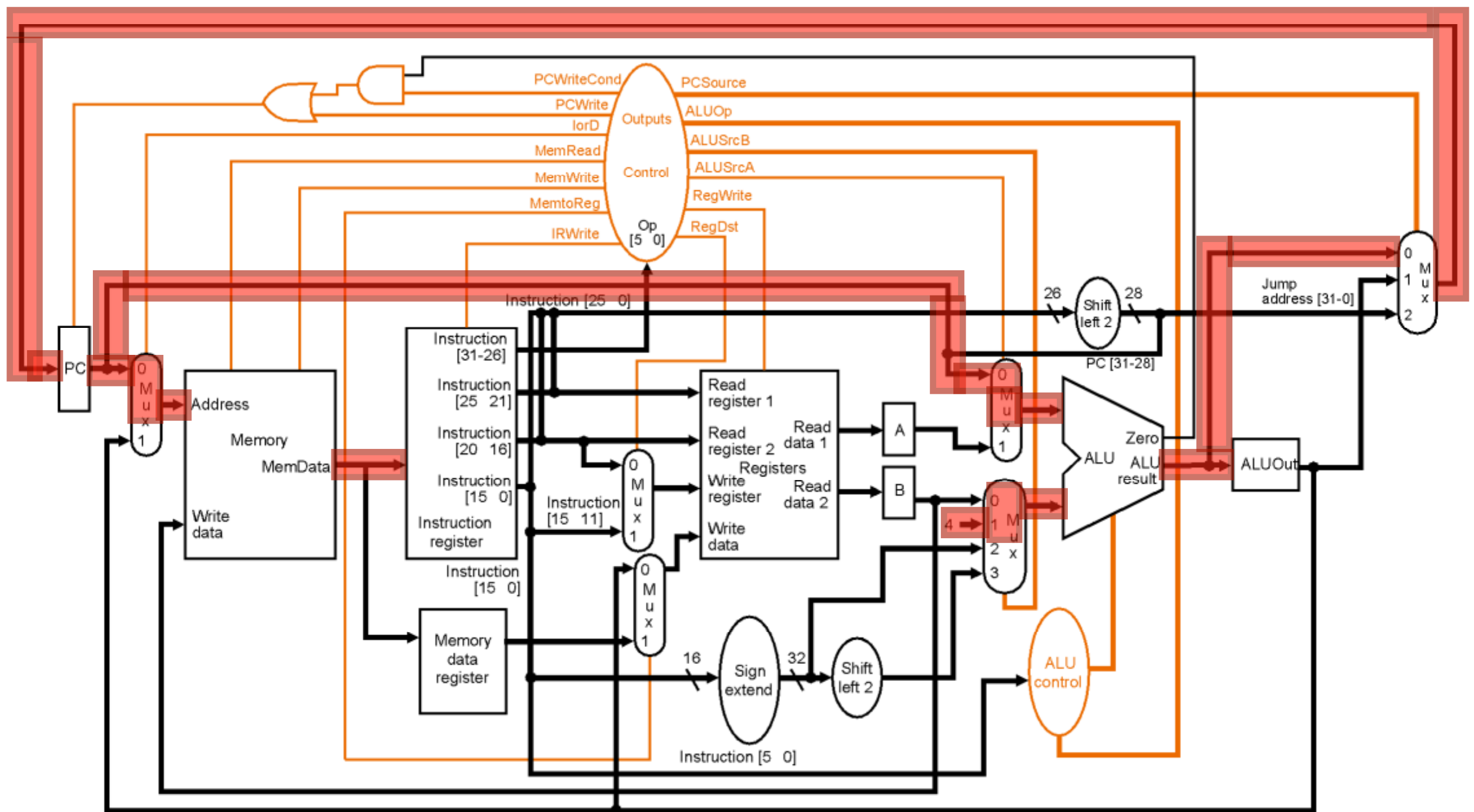


- What are the signals?
- What are the signals for...
 - add r1, r2, r3
 - lw r1, 1000(r3)
 - sw r2, 45(r1)
 - beq r1, r2, Loop

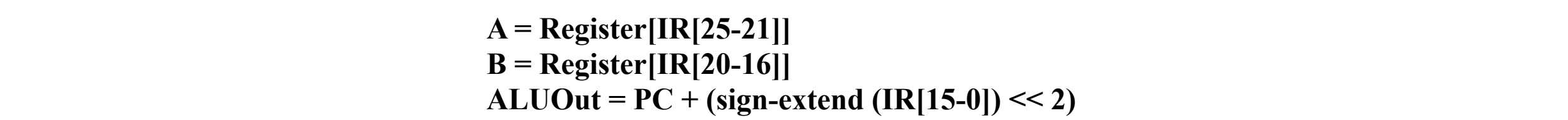
The diagram illustrates the internal architecture of a processor, showing the flow of data and control signals between various components:

- PC (Program Counter):** Receives the next instruction address from the ALU output or the jump address. It outputs the current instruction address to the Memory and the Instruction Register.
- Memory:** Receives the address from the PC and outputs data to the Memory data register. It also receives write data from the ALU output.
- Instruction Register:** Receives the instruction from the PC. It outputs the instruction to the ALU control, the ALU, and the Register File.
- Register File:** Contains registers for reading and writing. It receives the register number from the ALU control and outputs the register data to the ALU.
- ALU (Arithmetic Logic Unit):** Performs operations on data from the registers. It receives the ALU control signal and outputs the ALU result to the ALUOut and the Zero flag.
- Control Unit:** Receives the instruction from the Instruction Register and outputs control signals to the ALU control, the Register File, and the Memory.
- Multiplexers (Mux):** Select between different data sources. For example, the ALU control selects between the ALU result and the jump address. The Register File selects between the register data and the ALU result.
- Shifters:** Perform shift operations on data. The ALU control selects between the ALU result and the jump address. The Register File selects between the register data and the ALU result.
- Sign extend:** Extends the sign of the register data to 32 bits.
- Zero flag:** Indicates the result of the ALU operation is zero.
- Jump address:** The address to jump to, selected by the ALU control.
- ALUOut:** The output of the ALU operation.

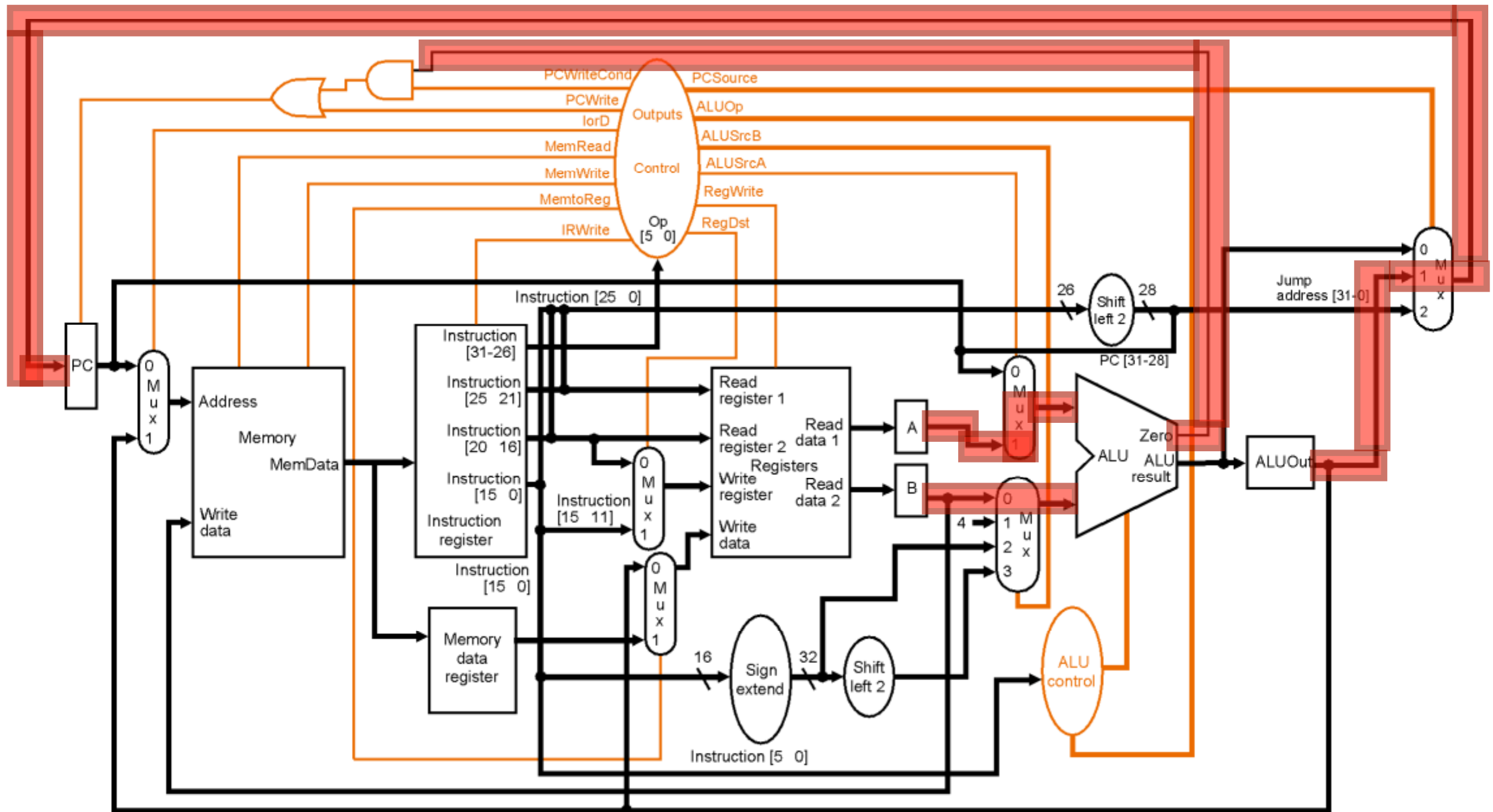
1. Instruction Fetch



IR = Memory[PC]
PC = PC + 4

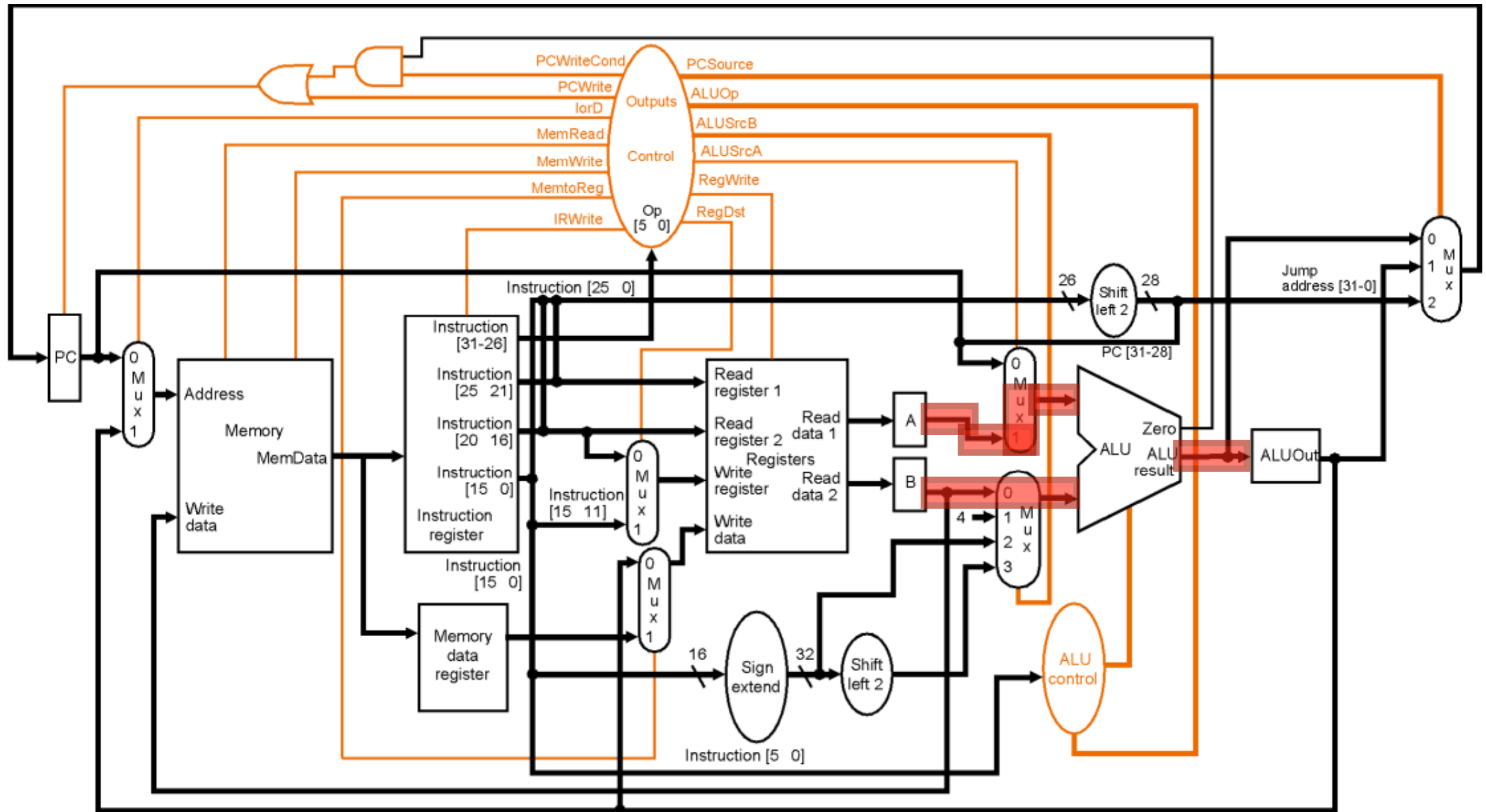

$$\text{ALUOut} = \text{PC} + (\text{sign-extend}(\text{IR}[15-0]) \ll 2)$$

3. Branch Completion



if (A == B) PC = ALUOut

3. Execution (R-Type)



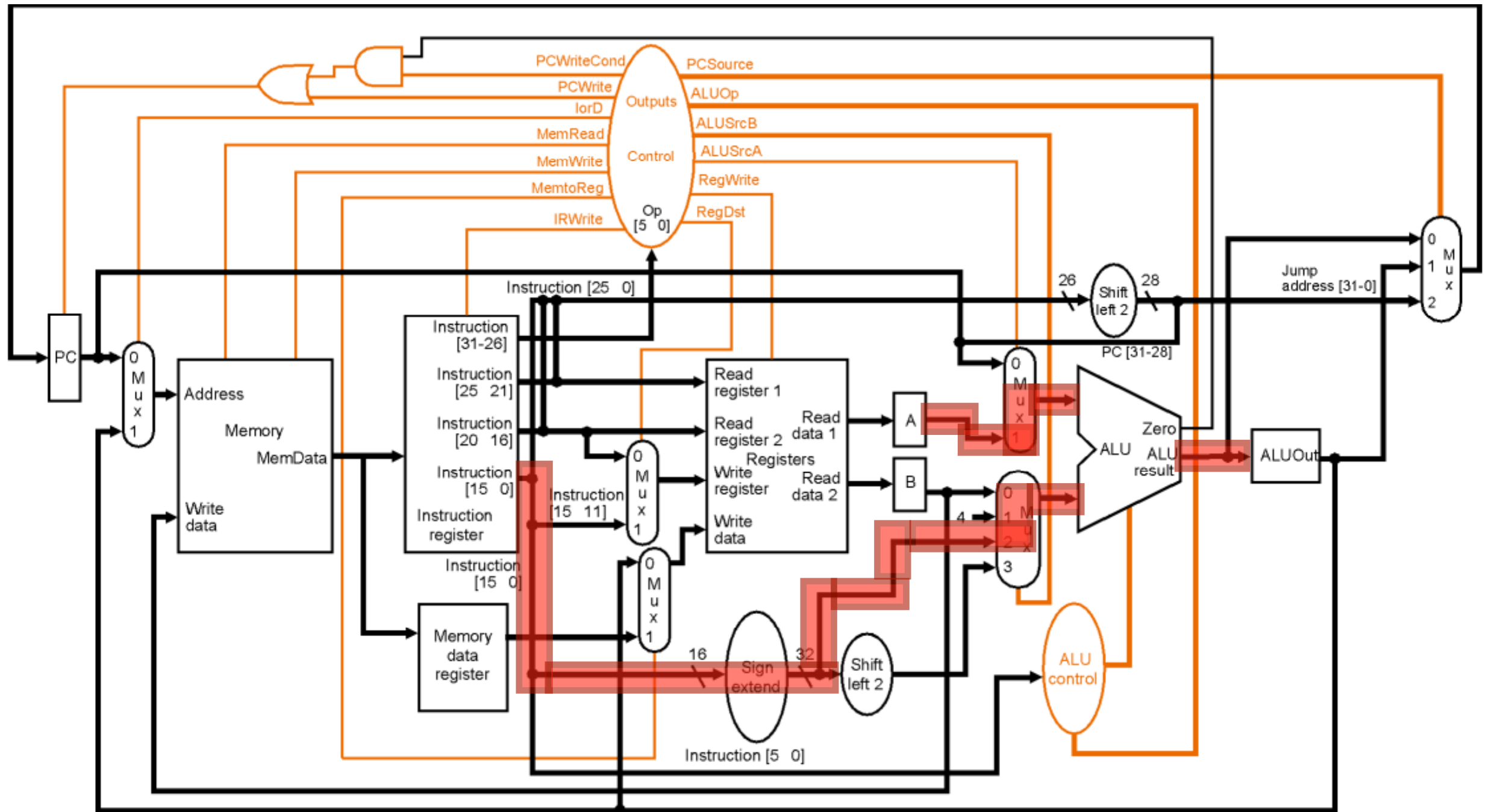
$$ALUout = A \text{ op } B$$

The diagram illustrates the internal components and data flow of a processor. Key elements include:

- PC (Program Counter):** Receives the next instruction address from the ALU output or a jump address. It feeds into a 2-to-1 multiplexer.
- Memory:** Receives an address from the PC multiplexer and outputs data to the Memory data register. It also receives write data from the ALU output.
- Instruction Register:** Receives the instruction from the PC. It is divided into fields: Instruction [31-26], [25-21], [20-16], [15-0], and [15-11].
- Registers:** Includes Read register 1, Read register 2, and a Write register. They receive data from the instruction fields and the Memory data register.
- ALU (Arithmetic Logic Unit):** Takes two operands (A and B) from registers and performs operations based on the ALUOp control signal. It also receives a 4-bit control signal from the ALU control unit.
- Control Unit:** Receives the instruction [5-0] and generates control signals: PCWriteCond, PCWrite, IorD, MemRead, MemWrite, MemtoReg, IRWrite, ALUOp, ALUSrcB, ALUSrcA, RegWrite, and RegDst.
- Multiplexers:** Several 2-to-1 and 4-to-1 multiplexers are used to select between different data sources (e.g., PC, registers, ALU result, jump address).
- Shifters:** Two shift-left-by-2 units are used to shift the PC [31-28] and the sign-extended instruction [15-0] to the right.
- Sign Extender:** Takes the 16-bit instruction [15-0] and extends it to 32 bits.

Reg[IR[15-11]] = ALUout

4. Memory Address Computation



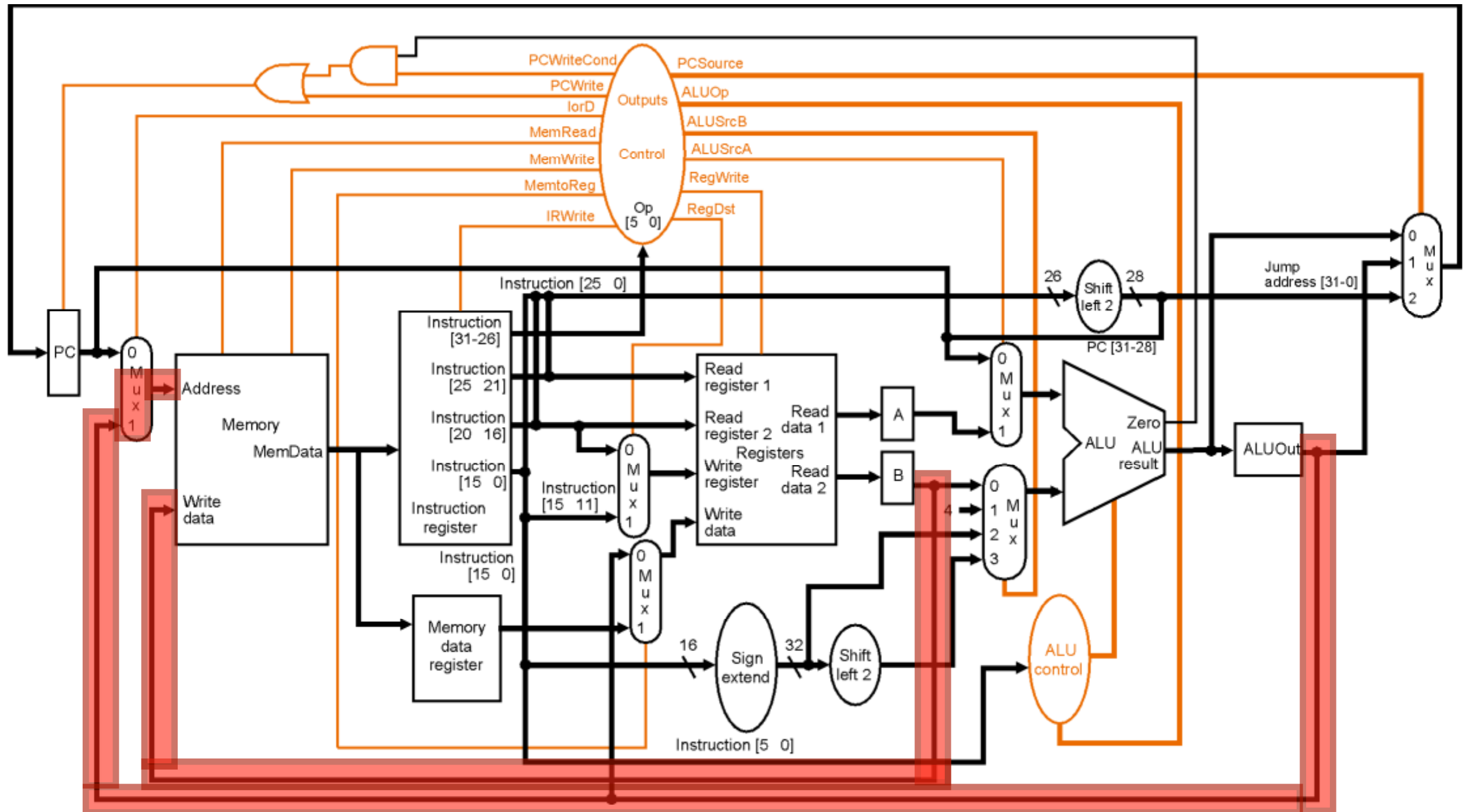
$$\text{ALUout} = A + \text{sign-extend}(\text{IR}[15-0])$$

The diagram illustrates the internal components and data/control flow of a processor. Key elements include:

- Control Unit:** A central block with 'Control' and 'Outputs' sections. It receives the instruction opcode (Op [5:0]) and generates control signals: PCWriteCond, PCWrite, IorD, MemRead, MemWrite, MemtoReg, IRWrite, ALUOp, ALUSrcB, ALUSrcA, RegWrite, and RegDst.
- Instruction Register:** Receives the 32-bit instruction from the PC. It is divided into fields: Instruction [31:26], [25:21], [20:16], [15:0], and [15:11].
- Memory:** Consists of 'Address' and 'MemData' ports. A 2-to-1 multiplexer selects between the PC and the 'Jump address [31:0]' to provide the 'Address' to memory.
- Registers:** Includes a 'Memory data register' and a 'Registers' block with 'Read register 1', 'Read register 2', 'Write register', and 'Write data' ports. A 2-to-1 multiplexer selects between the 'Memory data register' and the 'Registers' block to provide the 'Write data' to the 'Memory data register'.
- ALU and ALU Control:** The ALU takes two 4-bit operands (A and B) and performs operations based on the 'ALU control' signal. It produces a 4-bit 'ALU result' and a 'Zero' flag. A 3-to-1 multiplexer selects between the 'ALU result', the 'Zero' flag, and the 'Jump address [31:0]' to provide the 'Jump address [31:0]' to the 'Memory'.
- PC (Program Counter):** Receives the 'Jump address [31:0]' and outputs the 'PC [31:28]' to the 'Memory' and the 'Instruction register'.
- Sign Extension and Shifts:** The 16-bit instruction field [15:0] is sign-extended to 32 bits and then shifted left by 2 bits to provide the 'ALU control' signal. The 26-bit instruction field [25:0] is shifted left by 2 bits to provide the 'PC [31:28]'.

```
memory-data = Memory[ALUout]
```

4. Memory Access Store



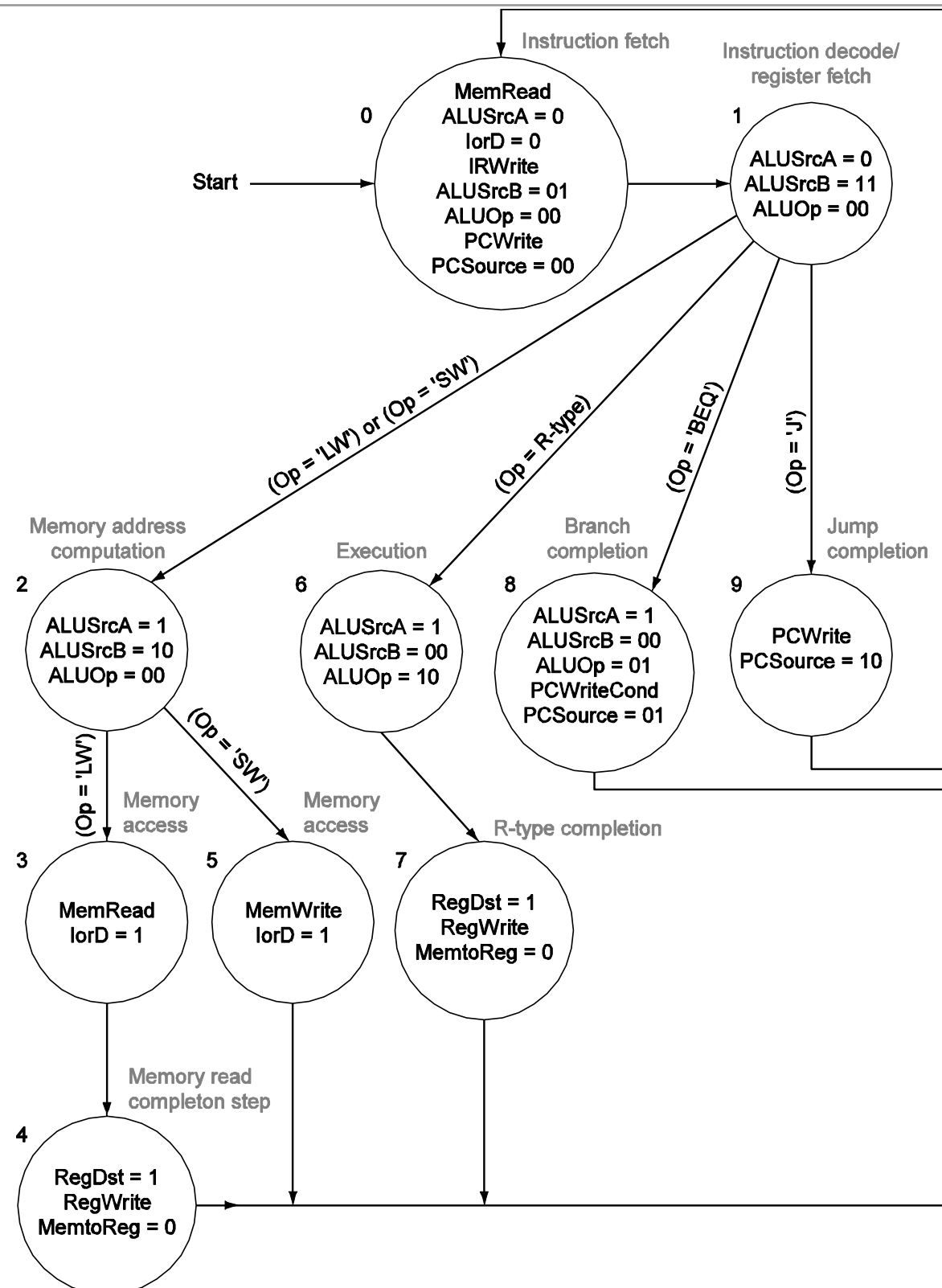
Memory[ALUout] = B

The diagram illustrates the internal components and data/control flow of a processor. Key elements include:

- PC (Program Counter):** Receives the next instruction address from the ALU output or a jump address. It feeds into a 1-to-2 multiplexer.
- Memory:** Receives an address from the PC and provides MemData to the instruction register and memory data register. It also receives write data.
- Instruction Register:** Holds the current instruction, providing fields [31-26], [25-21], [20-16], [15-0], and [15-11] to other components.
- Registers:** A set of registers that can be read (providing Read data 1 and 2) or written to (receiving Write data).
- ALU (Arithmetic Logic Unit):** Performs operations on data from registers or memory. It receives ALUOp, ALUSrcA, ALUSrcB, and ALU control signals. It outputs ALUOut and a Zero flag.
- Control Logic:** Includes a central 'Outputs' block that generates control signals (PCWriteCond, PCSource, ALUOp, ALUSrcB, ALUSrcA, RegWrite, RegDst, Op [5-0], IRWrite, MemRead, MemWrite, MemtoReg, PCWrite, IorD) based on instruction fields and other signals.
- Shifters:** Perform logical shifts (e.g., Shift left 2) on instruction fields or ALU results.
- Multiplexers:** Select between different data sources, such as the PC, ALU result, or jump address.

Reg[IR[20-16]] = memory-data

Multi-Cycle Control - The Full FSM

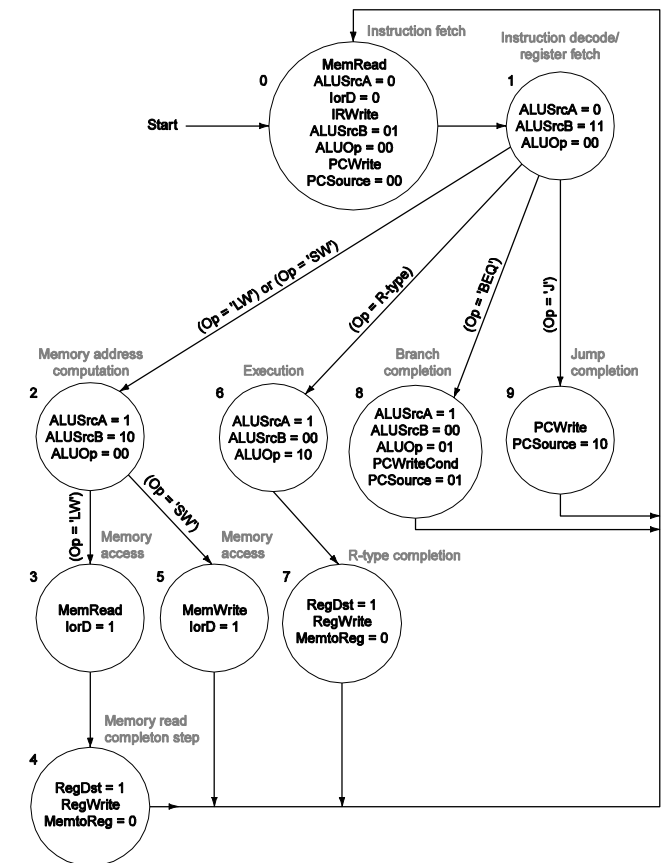


Some Juicy Questions

- How many cycles will it take to execute this code?

```
lw $t2, 0($t3)
lw $t3, 4($t3)
beq $t2, $t3, Label #assume not taken
add $t5, $t2, $t3
sw $t5, 8($t3)
Label: ...
```

- Whats going on during the 8th cycle of execution?
- In what cycle does the actual addition of \$t2 and \$t3 take place?
- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?



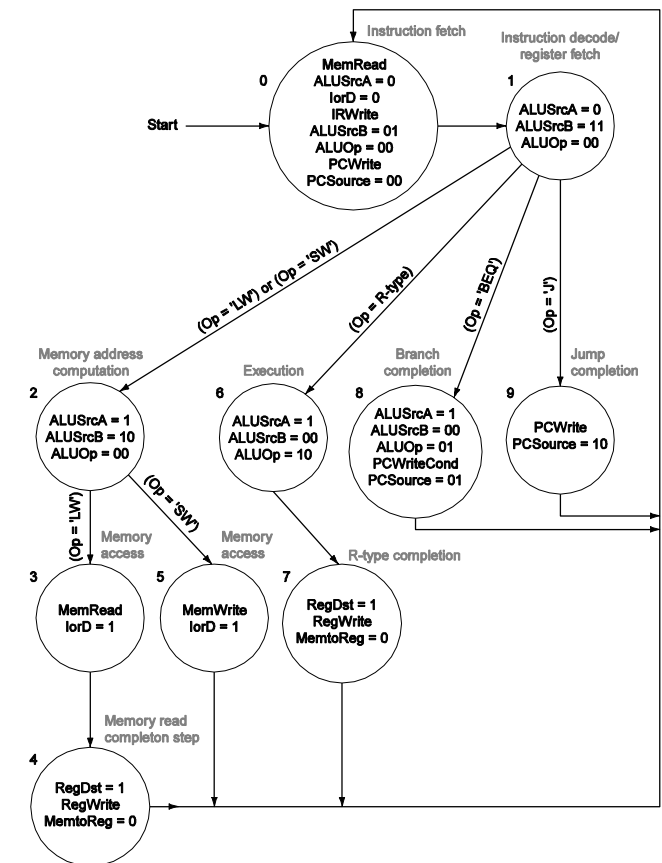
Some Juicy Questions

- How many cycles will it take to execute this code?

```

5  lw $t2, 0($t3)
   lw $t3, 4($t3)
   beq $t2, $t3, Label #assume not taken
   add $t5, $t2, $t3
   sw $t5, 8($t3)
   Label: ...
    
```

- Whats going on during the 8th cycle of execution?
- In what cycle does the actual addition of \$t2 and \$t3 take place?
- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?



Some Juicy Questions

- How many cycles will it take to execute this code?

5 lw \$t2, 0(\$t3)

5 lw \$t3, 4(\$t3)

beq \$t2, \$t3, Label #assume not taken

add \$t5, \$t2, \$t3

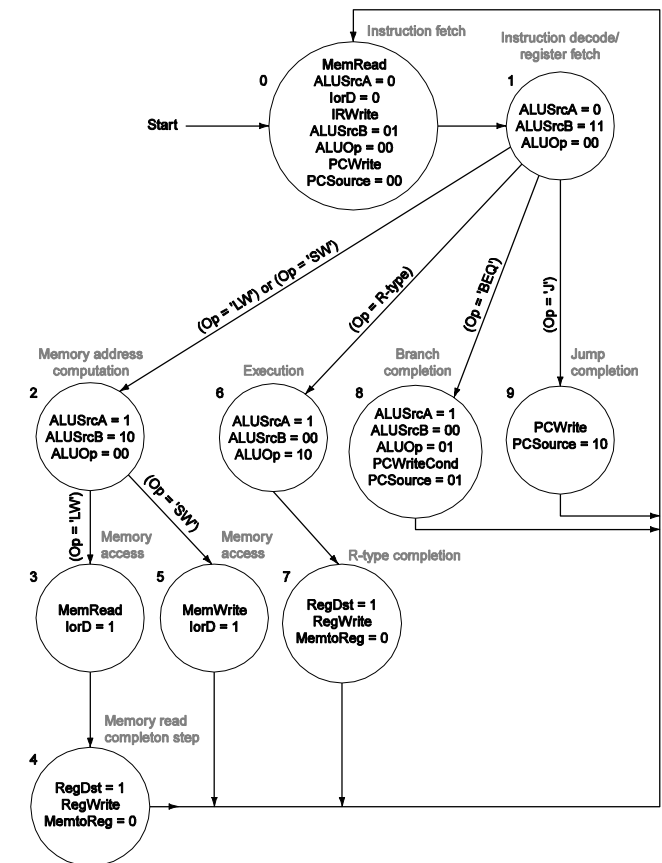
sw \$t5, 8(\$t3)

Label: ...

- Whats going on during the 8th cycle of execution?

- In what cycle does the actual addition of \$t2 and \$t3 take place?

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?



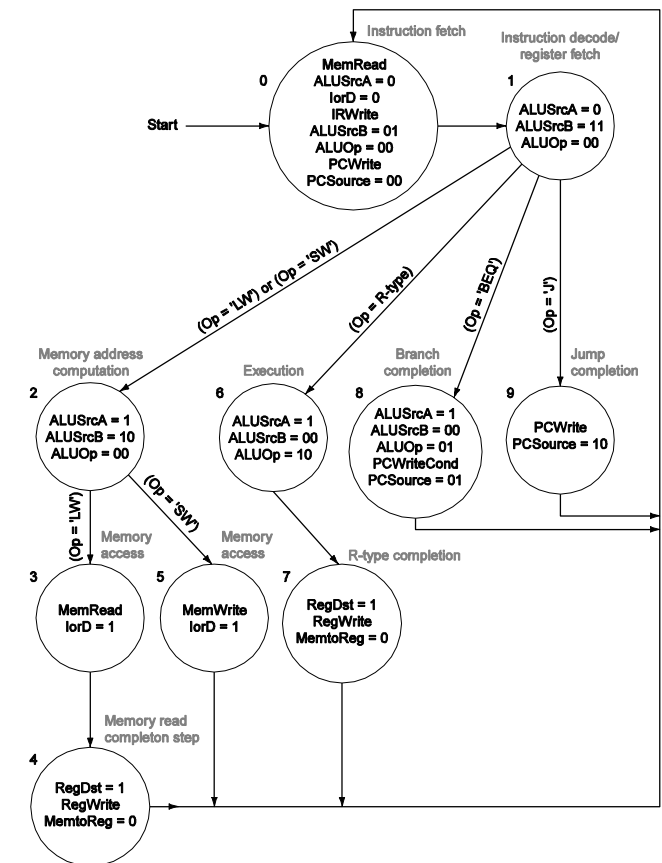
Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
  add $t5, $t2, $t3
  sw $t5, 8($t3)
  Label: ...
    
```

- Whats going on during the 8th cycle of execution?
- In what cycle does the actual addition of \$t2 and \$t3 take place?
- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?



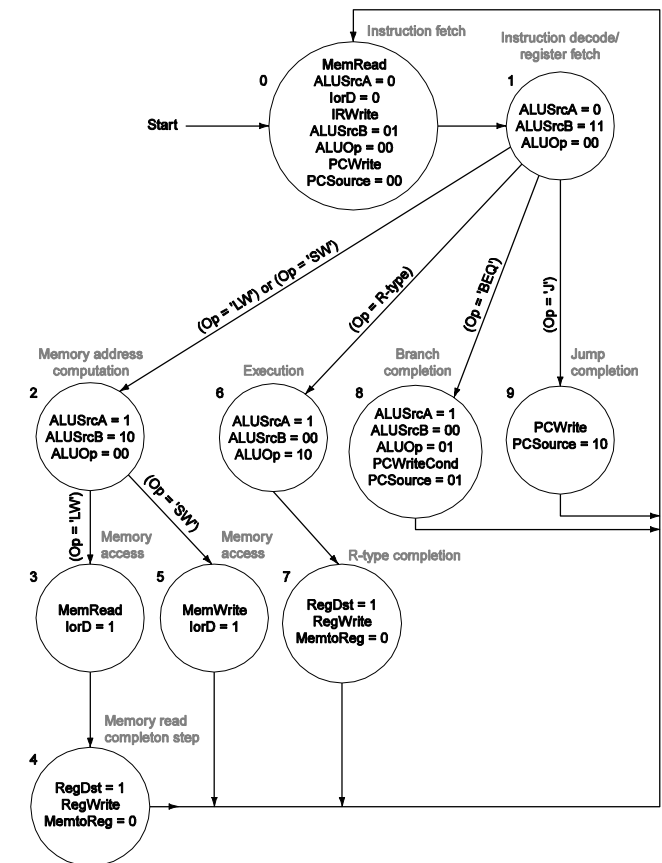
Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
  sw $t5, 8($t3)
Label: ...
    
```

- Whats going on during the 8th cycle of execution?
- In what cycle does the actual addition of \$t2 and \$t3 take place?
- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?



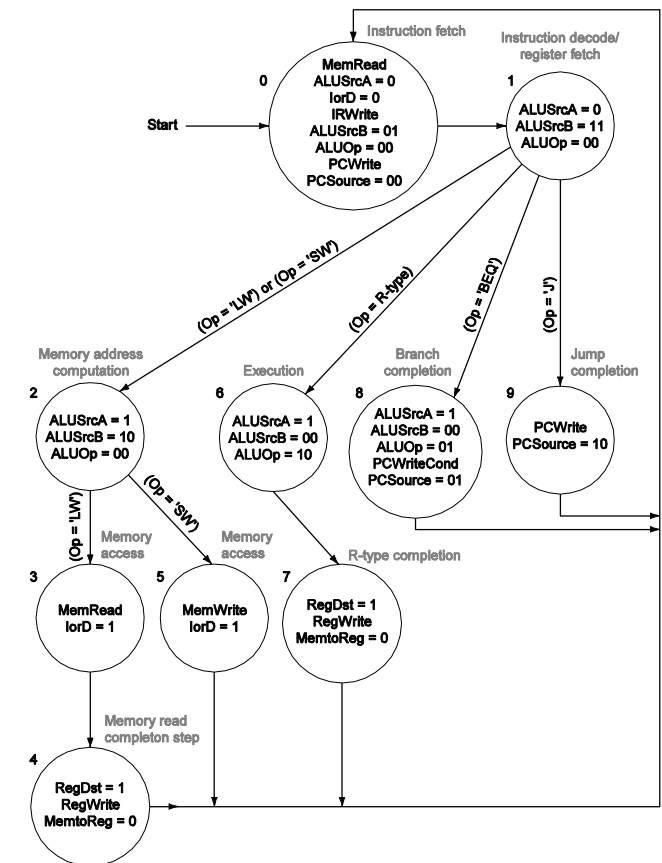
Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
    
```

- Whats going on during the 8th cycle of execution?
- In what cycle does the actual addition of \$t2 and \$t3 take place?
- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?



Some Juicy Questions

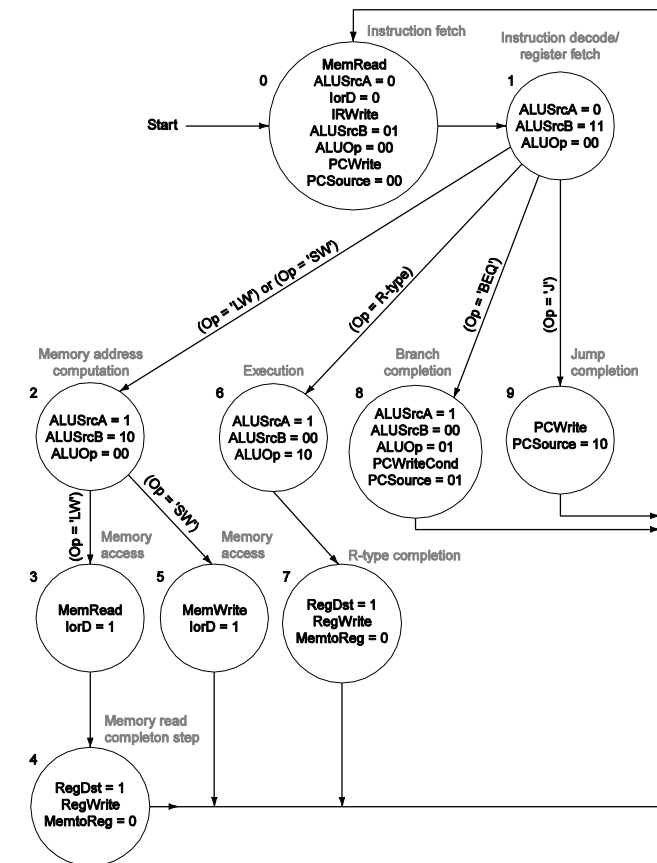
- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
    
```

21

- Whats going on during the 8th cycle of execution?
- In what cycle does the actual addition of \$t2 and \$t3 take place?
- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?



Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
    
```

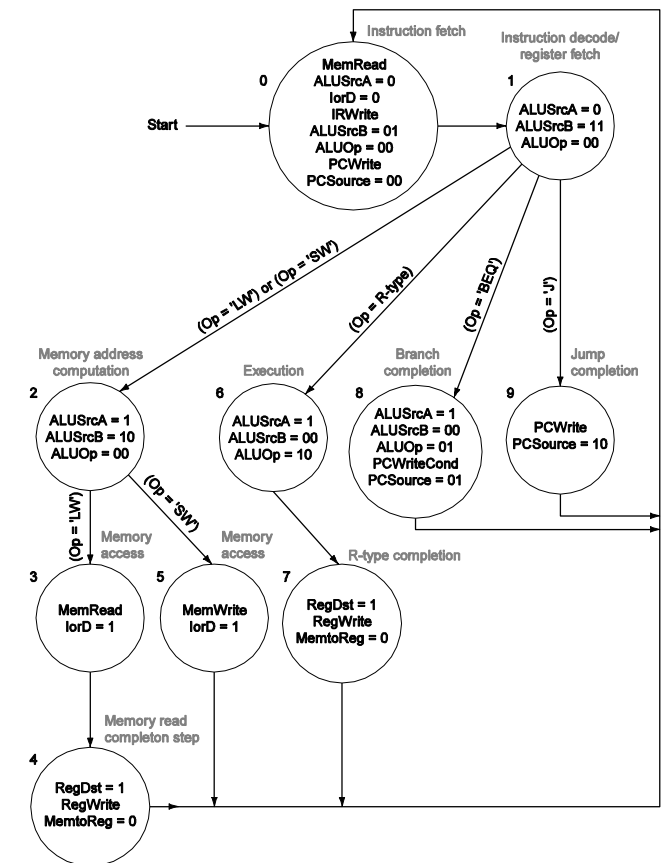
21

- Whats going on during the 8th cycle of execution?

lw

- In what cycle does the actual addition of \$t2 and \$t3 take place?

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?



Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
    
```

21

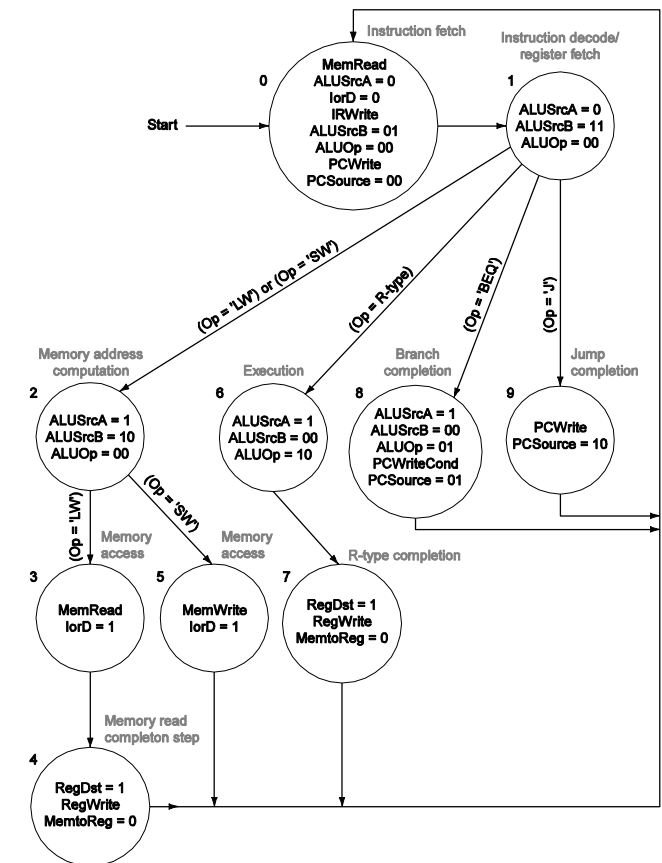
- Whats going on during the 8th cycle of execution?

lw

- In what cycle does the actual addition of \$t2 and \$t3 take place?

16th

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?



Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
    
```

21

- Whats going on during the 8th cycle of execution?

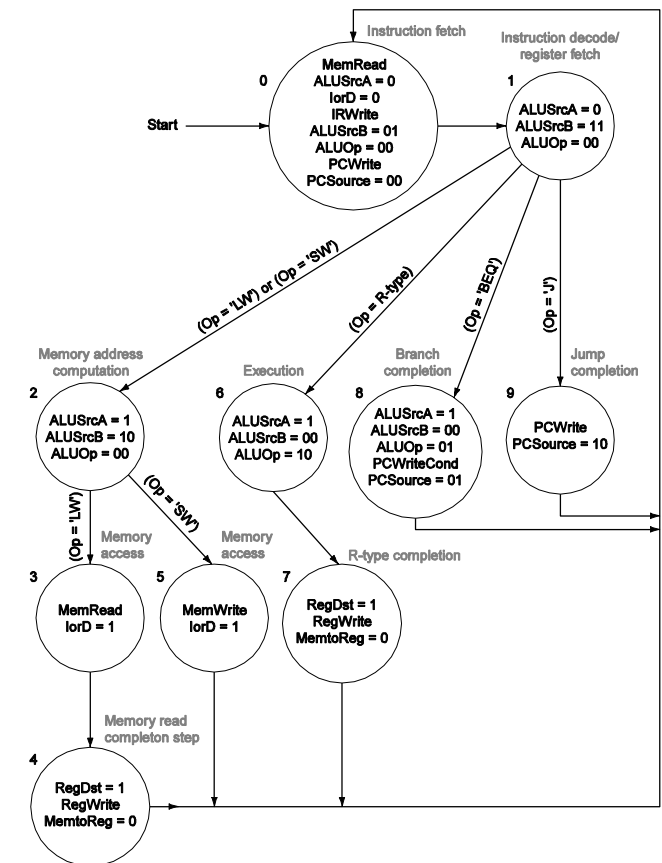
lw

- In what cycle does the actual addition of \$t2 and \$t3 take place?

16th

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

.2*(5) +



Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
    
```

21

- Whats going on during the 8th cycle of execution?

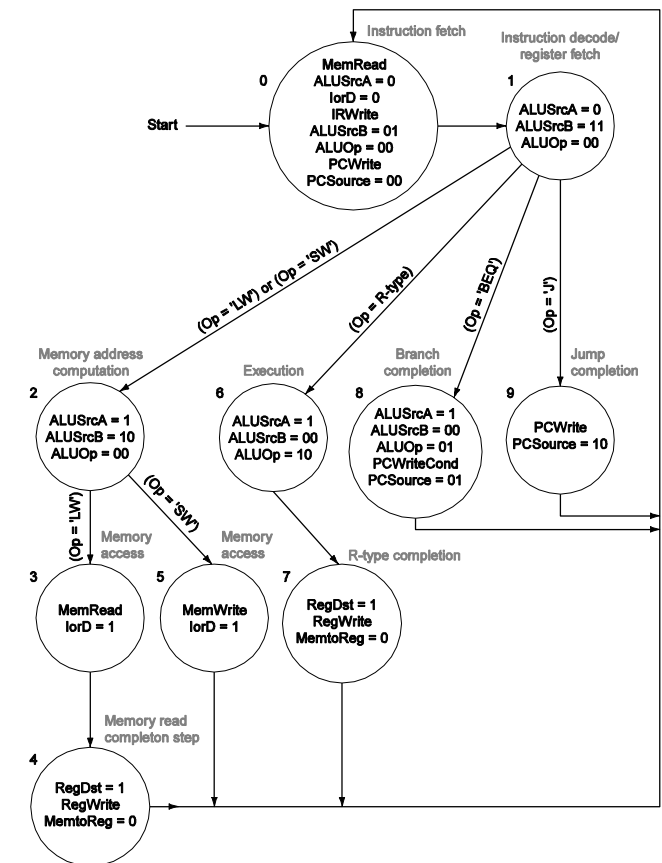
lw

- In what cycle does the actual addition of \$t2 and \$t3 take place?

16th

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

$.2*(5) + .1*(4) +$



Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
    
```

21

- Whats going on during the 8th cycle of execution?

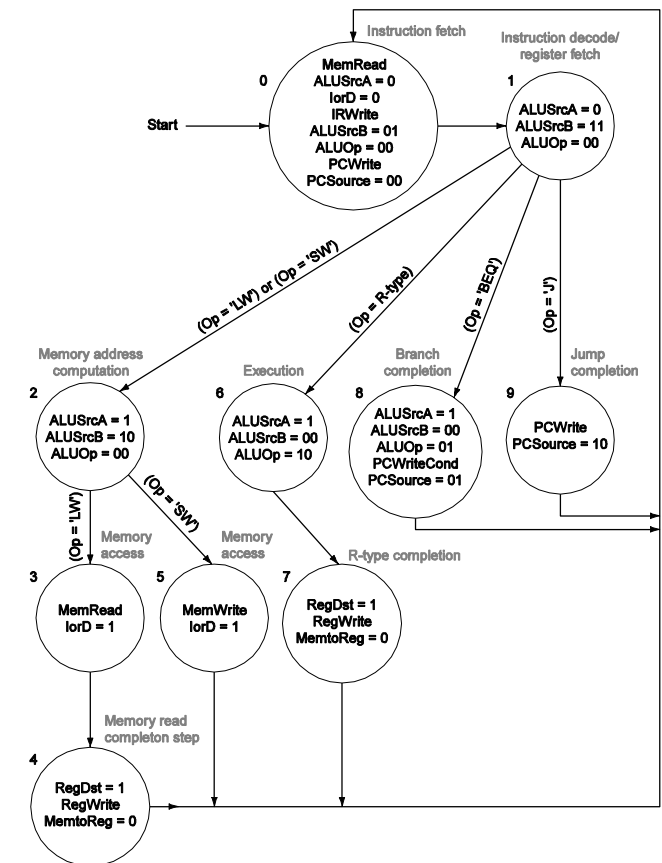
lw

- In what cycle does the actual addition of \$t2 and \$t3 take place?

16th

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

$.2*(5) + .1*(4) + .5*(4) +$



Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
    
```

21

- Whats going on during the 8th cycle of execution?

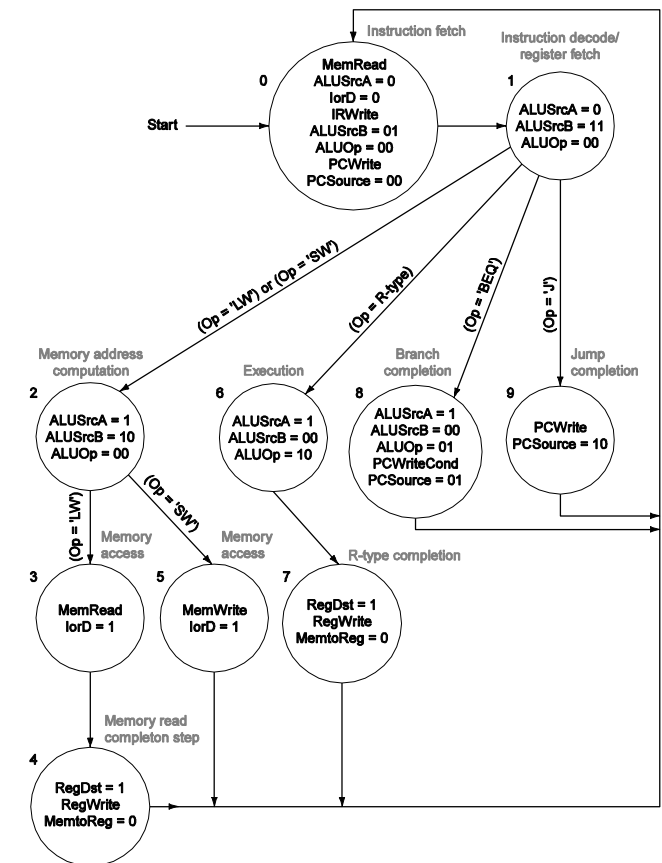
lw

- In what cycle does the actual addition of \$t2 and \$t3 take place?

16th

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

$$.2*(5) + .1*(4) + .5*(4) + .2*(3) =$$



Some Juicy Questions

- How many cycles will it take to execute this code?

```

5 lw $t2, 0($t3)
5 lw $t3, 4($t3)
3 beq $t2, $t3, Label #assume not taken
4 add $t5, $t2, $t3
4 sw $t5, 8($t3)
Label: ...
    
```

21

- Whats going on during the 8th cycle of execution?

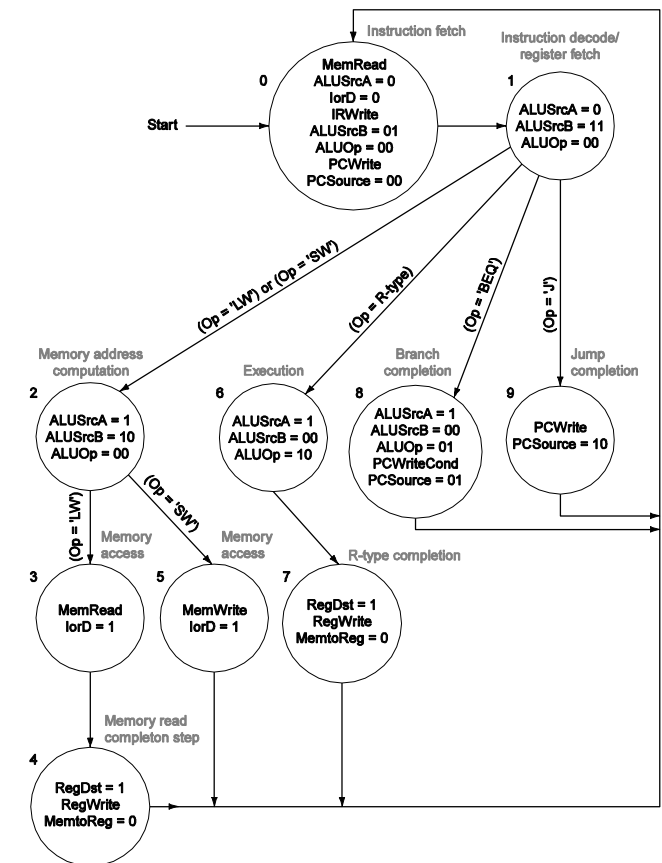
lw

- In what cycle does the actual addition of \$t2 and \$t3 take place?

16th

- Assume 20% loads, 10% stores, 50% R-type, 20% branches, what is the CPI?

$$.2*(5) + .1*(4) + .5*(4) + .2*(3) = 4$$



Good Luck!
