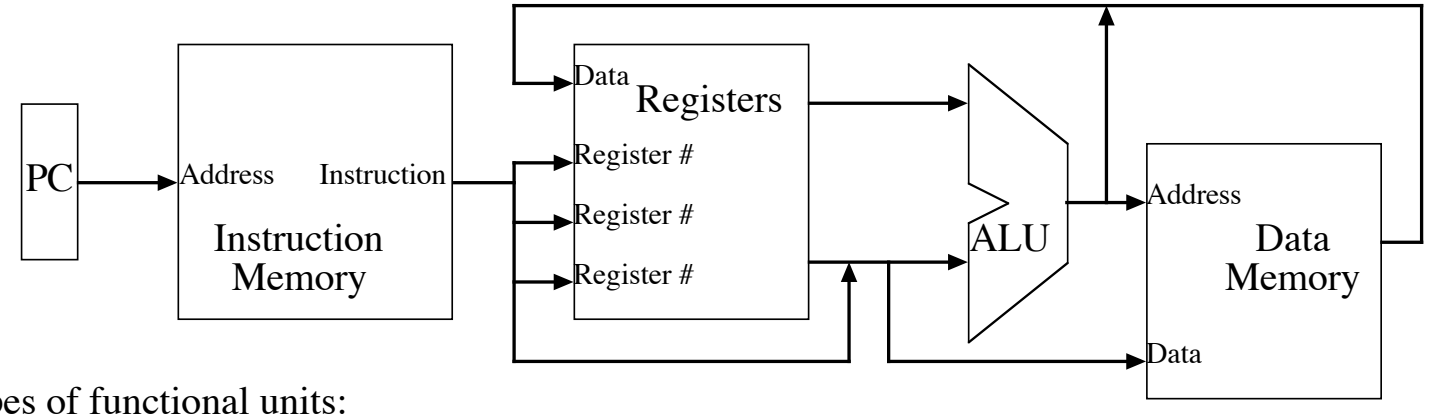
# The Processor

Read: Section 4.1, pages 300-303 (4th edition)

- We are ready to look at an implementation of the MIPS CPU.
- Simplified to contain only:
  - Memory-reference instructions: **`lw`**, **`sw`**.
  - Arithmetic-logical instructions: **`add`**, **`sub`**, **`and`**, **`or`**, **`slt`**.
  - Control flow instructions: **`beq`**, **`j`**.
- Generic implementation:
  - Use the Program Counter (PC) to supply the instruction address.
  - Get the instruction from memory.
  - Read values from registers.
  - Use the instruction to decide exactly what to do.
- All instructions (except **`j`**) use the ALU after reading the registers.
  - Why? Memory-reference? Arithmetic? Control flow?

**Implementation Overview**:

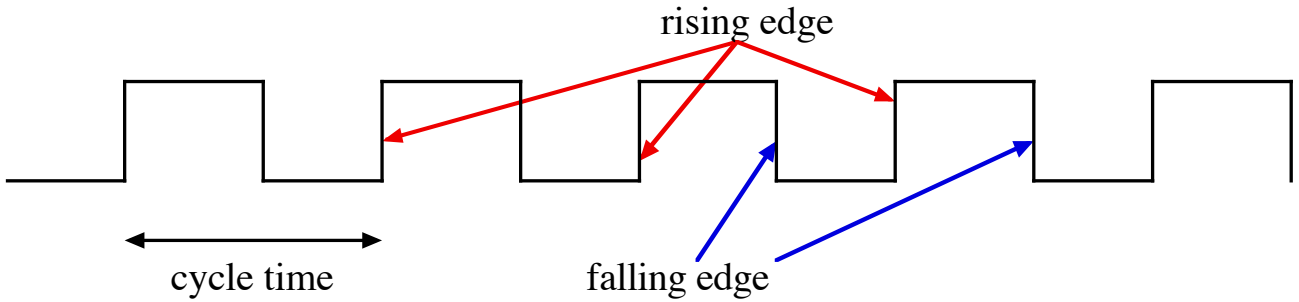
- Abstract/Simplified View:



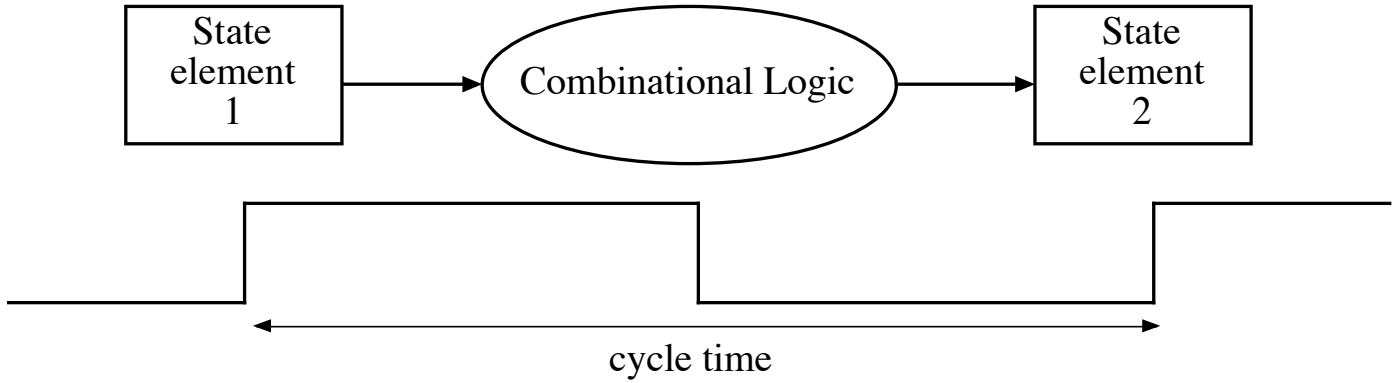

- Two types of functional units:
  - Elements that operate on data values (*combinational* units).
    - Outputs depend only on the current inputs.
    - Example: ALU
  - Elements that contain state (*sequential* units).
    - Has some internal storage (*state*).
    - Example: instruction memory and data memory.
    - Example: registers.

Implementation Overview (continued):

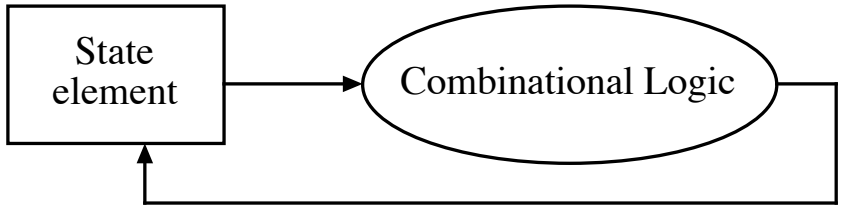
- State Elements:
  - Reading: Section 4.2, pages 303-307 (4th edition)
- Unclocked vs. Clocked: when do we read values, when do we write them?
- Clocks used in synchronous logic:
  - When should an element that contains state be updated?




  - Do not want to update and read at the same time, since state may be inconsistent.



Implementation Overview (continued):

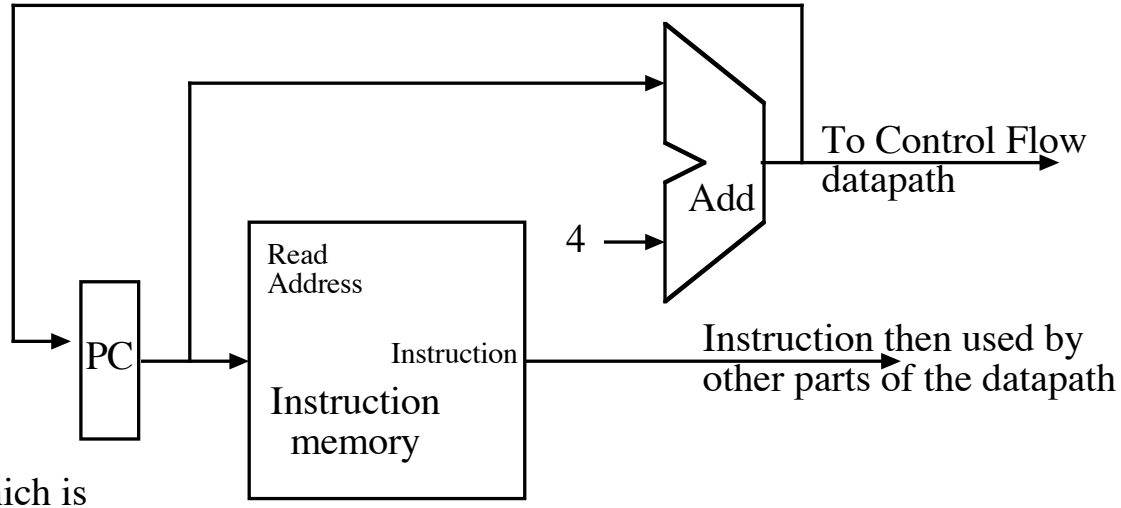



- Clocking Methodology:
  - Will assume "edge-triggered"
    - Still completes in one clock cycle.
    - No feedback can occur in this design — chapter 4 designs fit this requirement.
  - Assume a "simple" implementation:
    - A single long clock cycle is used for every instruction.
    - Easier to understand (yea!).
  - Not practical:
    - Want different instruction classes to take different numbers of clock cycles
      - Improves overall performance.
      - Each clock cycle is then much shorter.
      - More realistic.
      - Requires more complex control.

**Building a datapath**:

- Used by all three classes of instructions.
  - J-format, R-format, I-format.
- Instruction memory:
  - State unit, holds the instructions that make up the program.
  - Given an address, produces an instruction (four bytes).
- Program Counter, PC:
  - State unit.
  - Holds address of the *next* instruction.
- Add ALU:
  - Combinational unit.
  - Hard-wired to perform only addition.
  - Always uses 4 as the second operand, which is the size (in bytes) of one instruction.

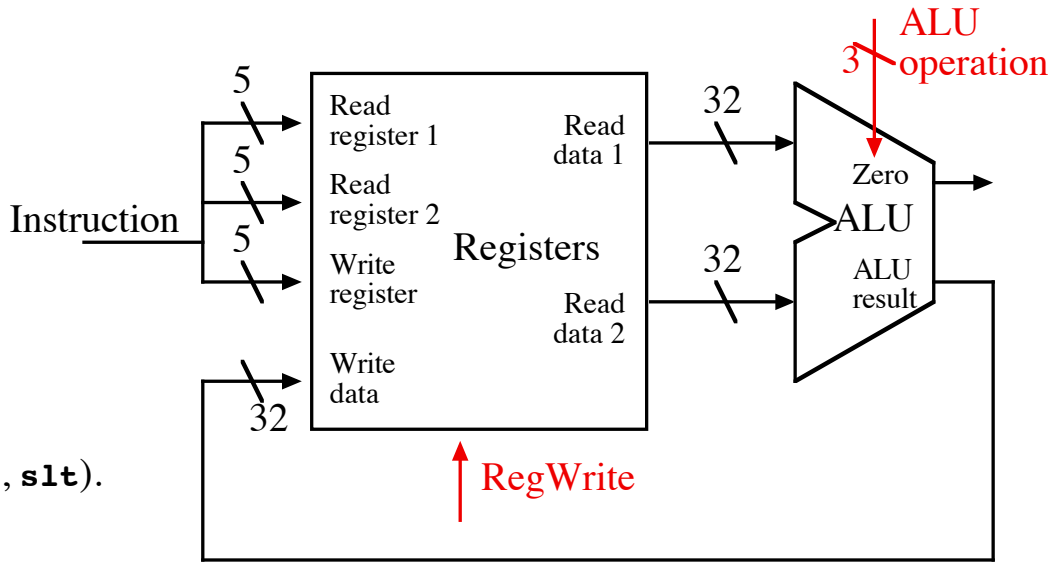
- Reading: Section 4.3, pages 307- (4th edition)



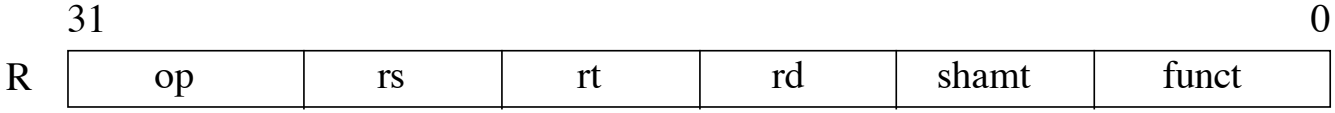
# sub $t7, $s2, $s3

Building a Datapath (continued):

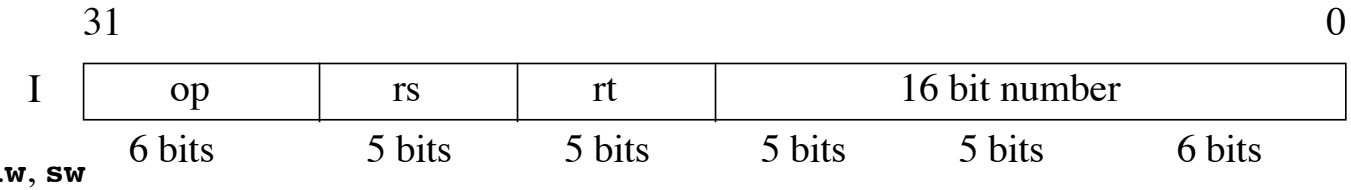
- R-format Instructions:
  - **`add`**, **`sub`**, **`and`**, **`or`**, **`slt`**.
  - Three register operands needed.
  - Read two data words from the register file.
  - Write one data word into the register file.
- **`ALU`** is the full-featured ALU (see Appendix B):
  - 3 control bits for ALU operations:
    - 2 to specify the Operations (**`add`**, **`and`**, **`or`**, **`slt`**).
    - 1 for Bnegate (used for **`sub`**, **`slt`**).



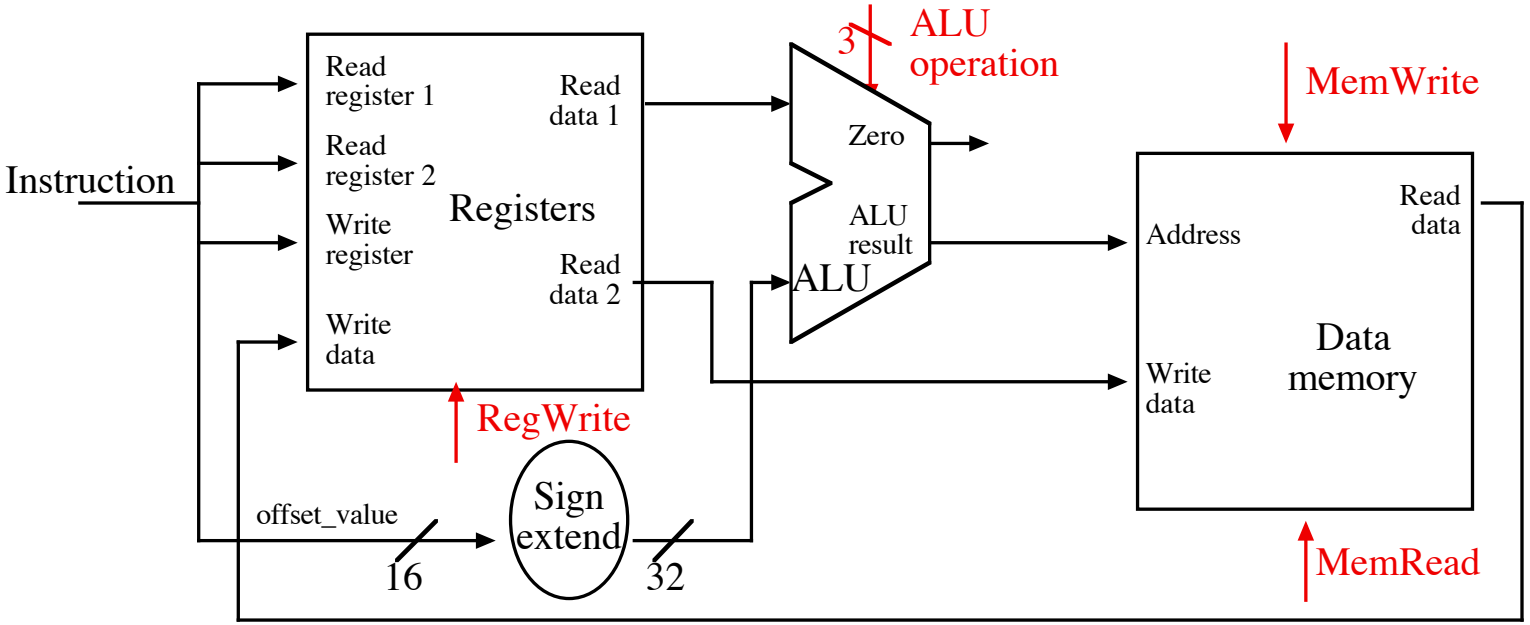

- **`Registers`** inputs and outputs:
  - Four inputs:
    - 3 register numbers (5 bits each).
    - 1 register value for writing the result (32 bits).
  - Two outputs:
    - 2 register values for ALU (32 bits each).
  - Control
    - 1 control value *asserted* to write a value to a register.
    - (Register file always outputs the contents of register numbers indicated on the Read register inputs.)

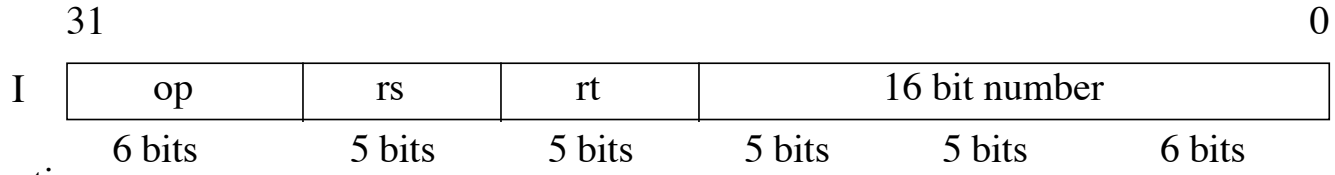


Building a Datapath (continued):




- Memory-reference instructions: **`lw`**, **`sw`**
  - Example: **`lw $t1, offset_value($t2)`**
  - Get contents of data memory at **`$t2 + offset_value`** and place in register **`$t1`**.
- Sign extend unit: the **`offset_value`** is a 16-bit signed value. Must convert ("extend") to a 32-bit signed value.
- Memory unit: Address gives location. Can read value (output) or write value (input).
  - Two inputs: address and write data (both 32 bits).
  - One output: read Data (32 bits).
  - Two control wires: **`MemWrite`**, **`MemRead`**.

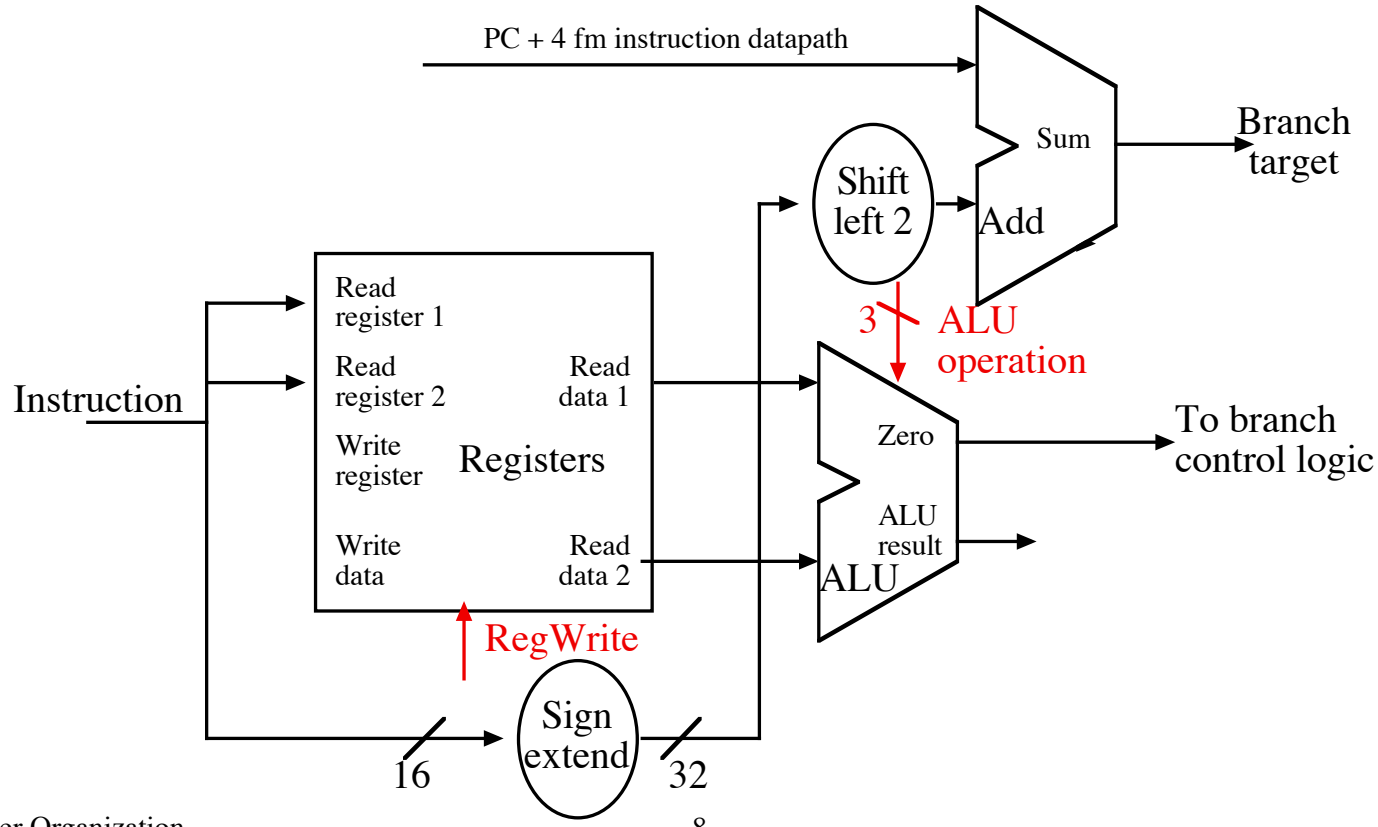





Building a Datapath (continued):

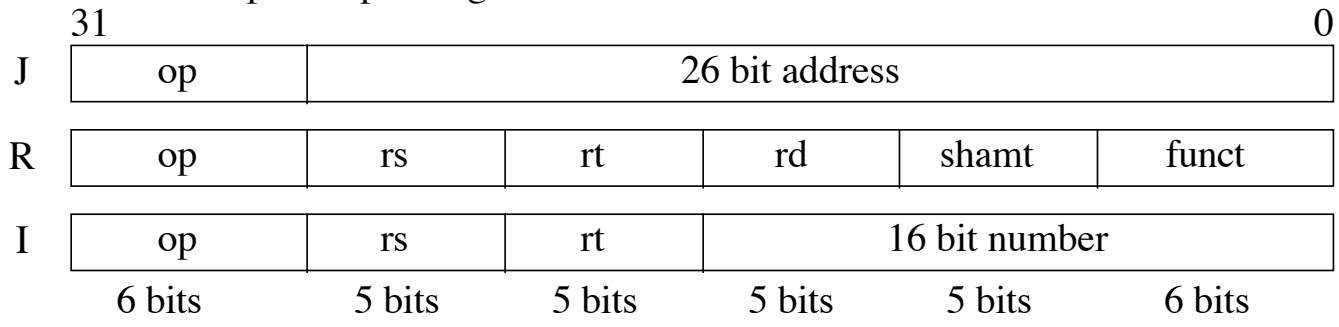
- Implementing Control Flow Instructions:
  - **`beq`**: 3 operands; 2 registers to compare, and a 16-bit signed offset from the PC.
  - Base address is the address of the instruction *following* the current instructions.
  - Offset is shifted two places (all instruction addresses are *word-aligned*).
  - Two outputs: Result of comparison, and Branch target.

```
beq  $t2, $s7, atTheEnd
```



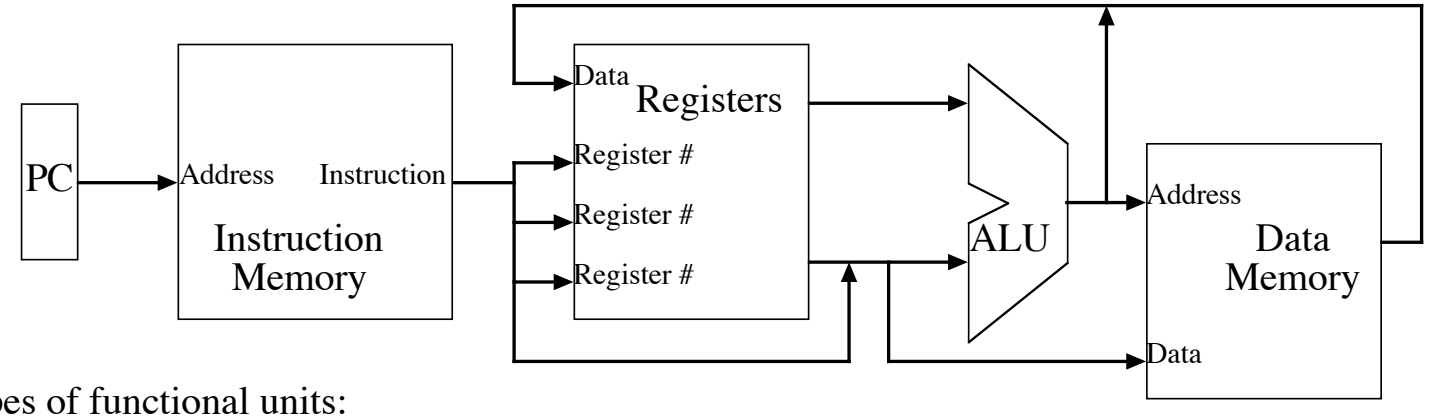
**Single-cycle Implementation**:

- Reading: Section 4.4, pages 316-330 (4th edition)
- Single Datapath from the four pieces shown earlier (and described in Section 4.2).
  - Memory-reference instructions: **`lw`**, **`sw`**.
  - Arithmetic-logical instructions: **`add`**, **`sub`**, **`and`**, **`or`**, **`slt`**.
  - Control flow instructions: **`beq`**, **`j`**.
- Simplest datapath: execute each instruction in a single clock cycle.
- Requires some duplication of elements (multiple ALU's, for example).
- Requires separate instruction memory and data memory.
- Can share some elements from different datapaths.
  - Multiplexors are necessary several places.
  - Choose between two inputs depending on the instruction.



**Implementation Overview**:

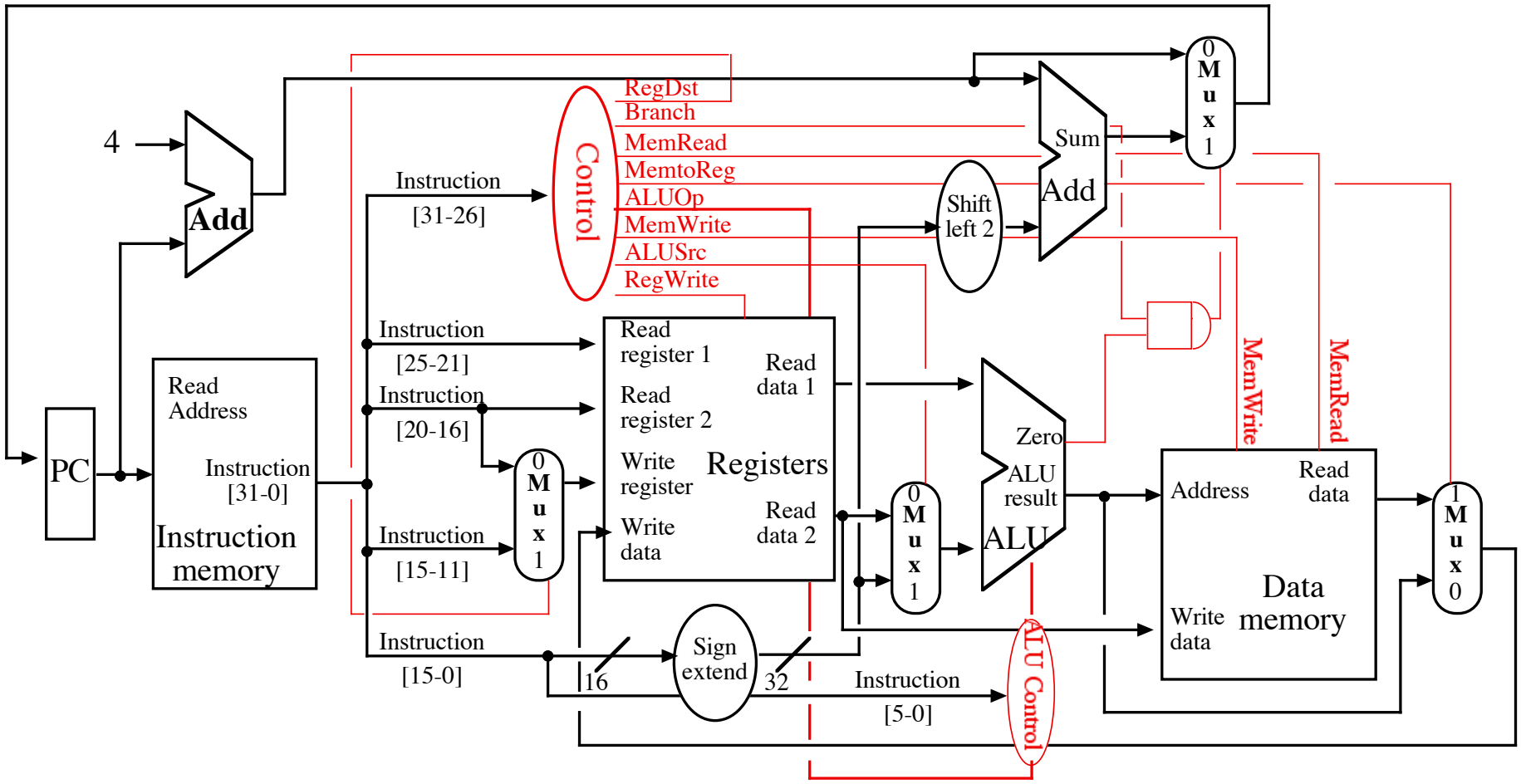
- Abstract/Simplified View:




- Two types of functional units:
  - Elements that operate on data values (*combinational* units).
    - Outputs depend only on the current inputs.
    - Example: ALU
  - Elements that contain state (*sequential* units).
    - Has some internal storage (*state*).
    - Example: instruction memory and data memory.
    - Example: registers.

Single-cycle Implementation (continued):

- Overview:

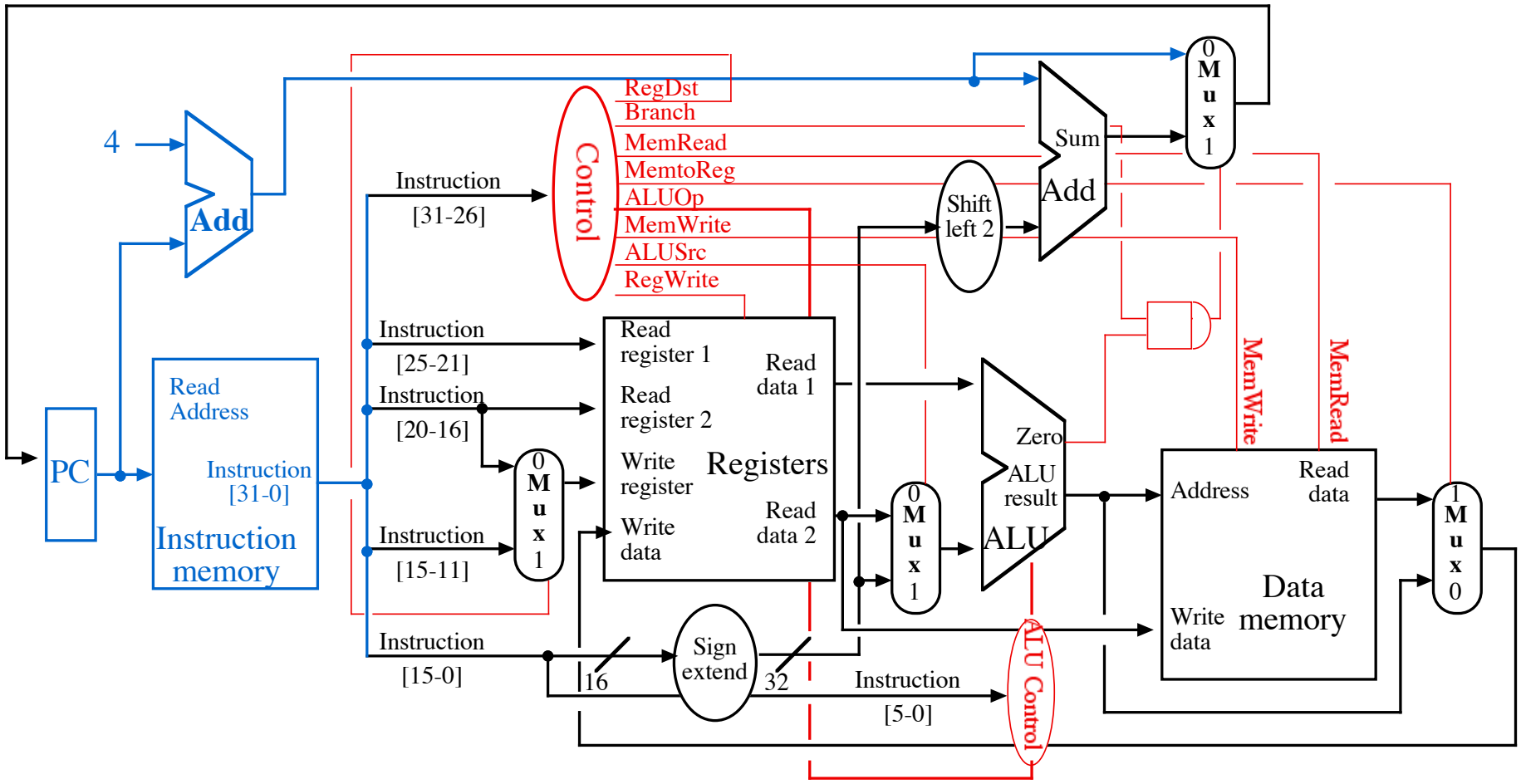


| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| R | op | rs | rt | rd | shamt | funct |

Single-cycle Implementation (continued):

- R-Format instruction: **`add`**, **`sub`**, **`slt`**, **`and`**, **`or`**.
- Fetch: Get instruction from memory and increment PC.

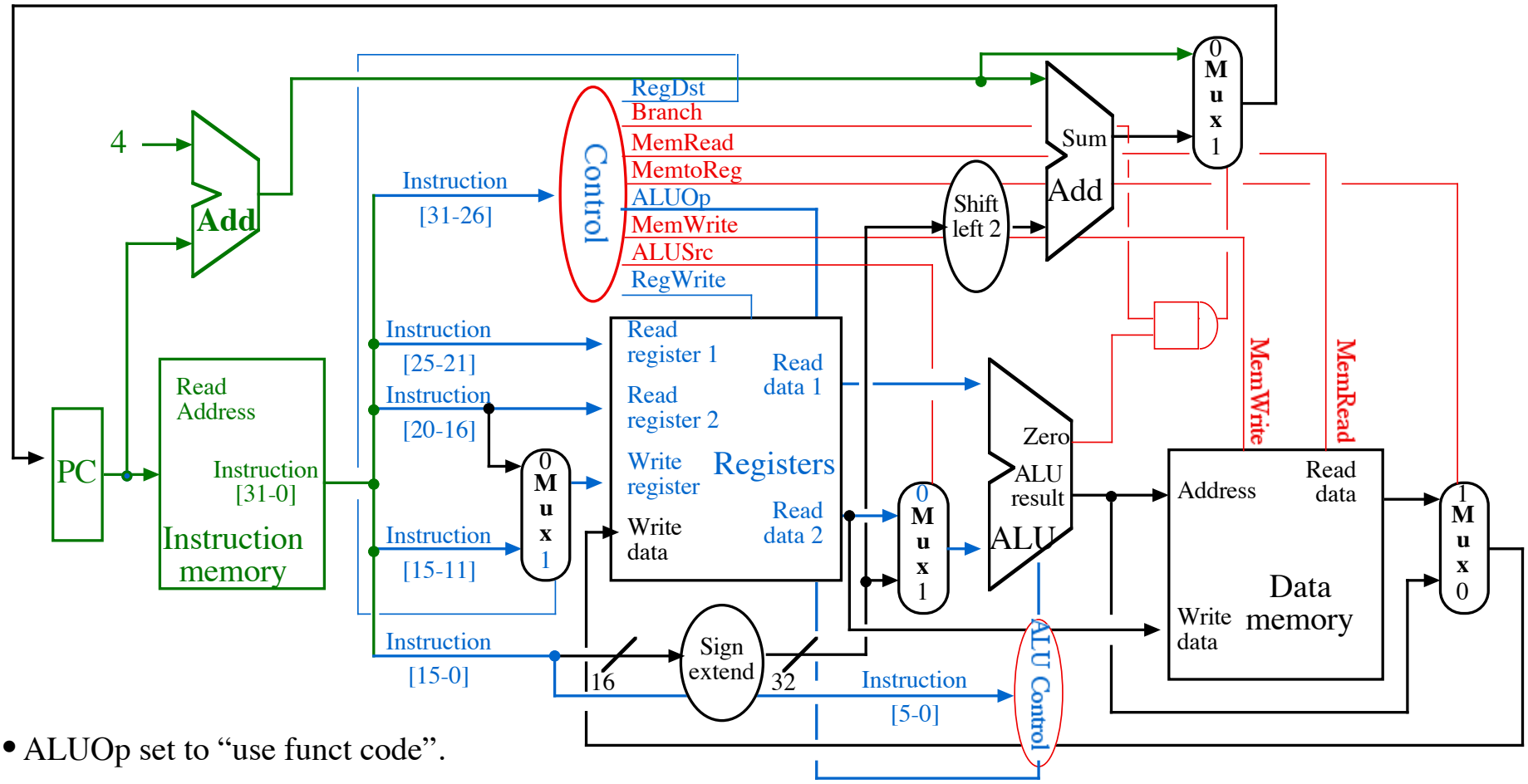
Blue lines/text indicate the current action.



| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| R | op | rs | rt | rd | shamt | funct |

Single-cycle Implementation (continued):

- R-Format instruction: Get the two source register values from Register file.
- Indicate register to write (will be written later).
- RegDst and RegWrite turned on.

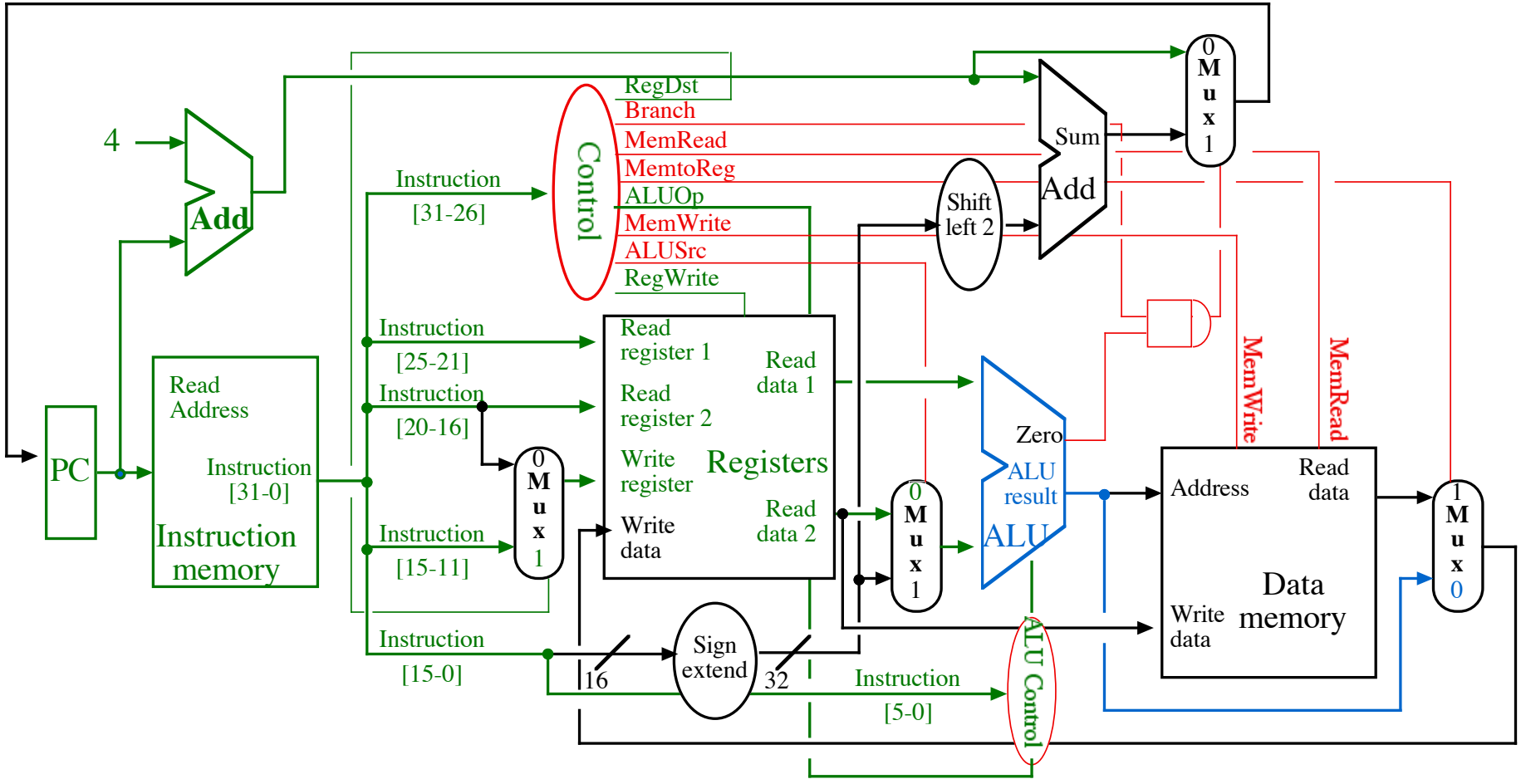



- ALUOp set to “use funct code”.

| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| R | op | rs | rt | rd | shamt | funct |

Single-cycle Implementation (continued):

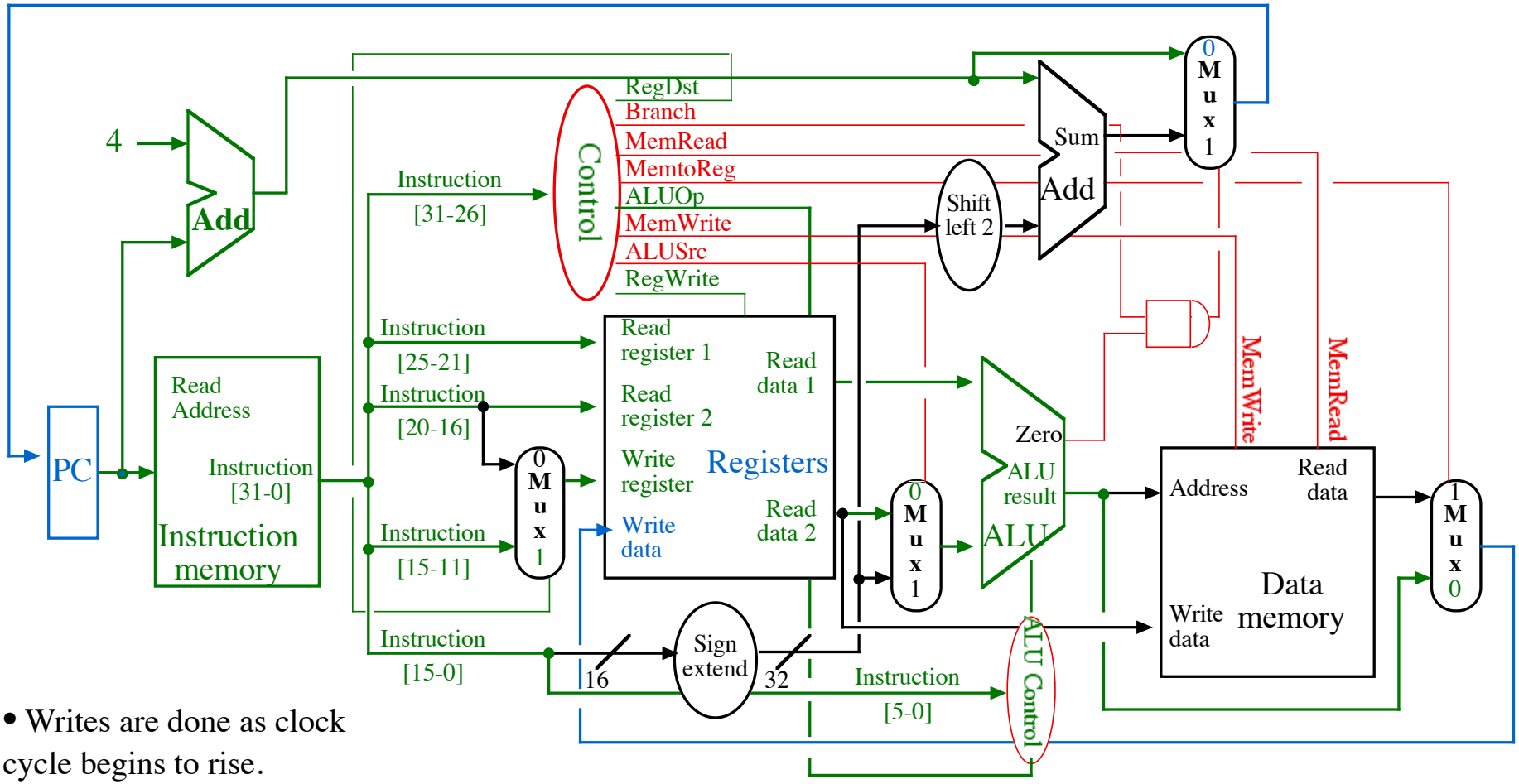
- R-Format instruction: Compute the answer.
  - MemtoReg is 0; we want the ALU result to go to the Registers.



| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| R | op | rs | rt | rd | shamt | funct |

Single-cycle Implementation (continued):

- R-Format instruction: Compute the answer.
  - Branch is 0; PC+4 written to PC.
  - ALU Result is written to Registers.

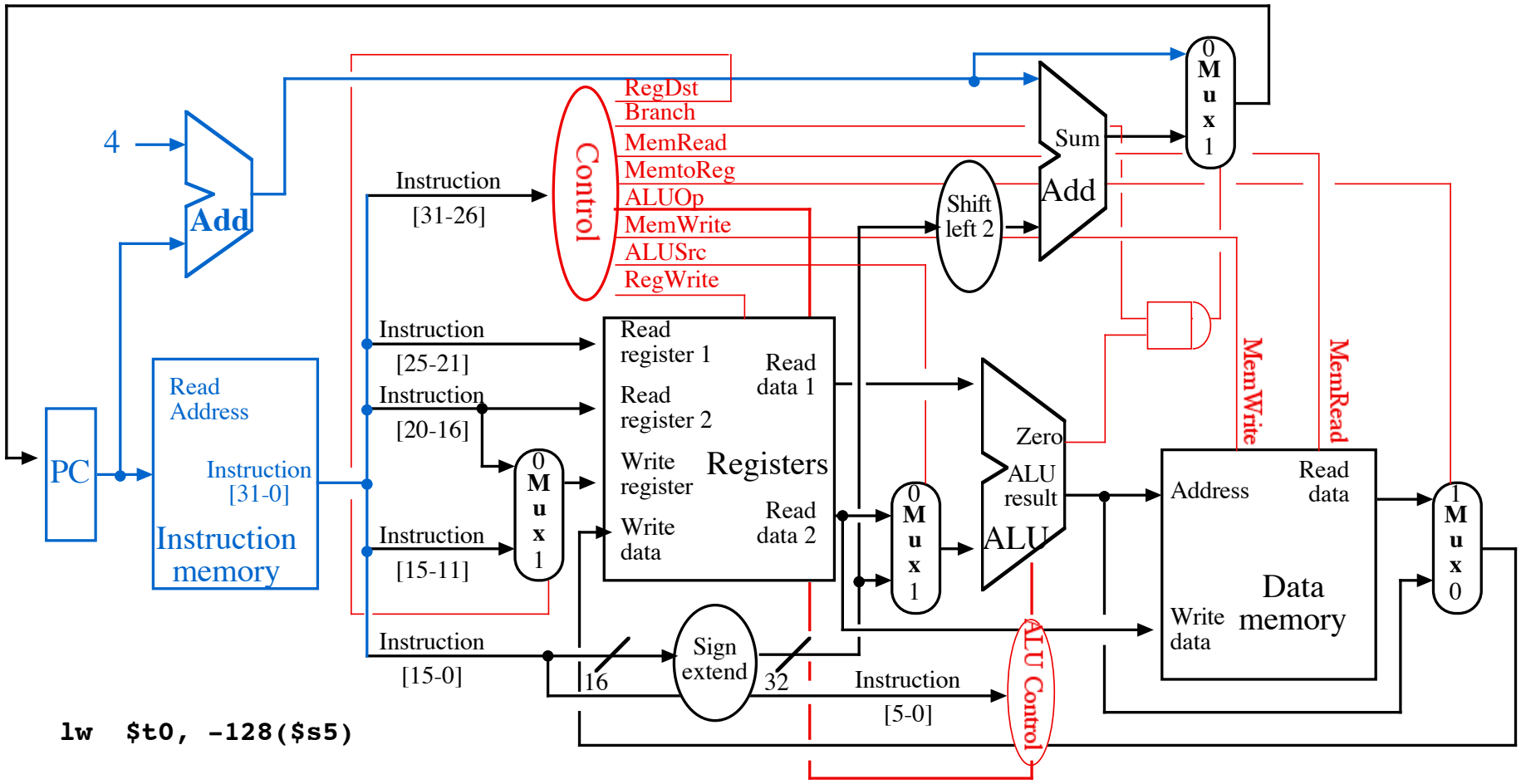



- Writes are done as clock cycle begins to rise.

| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| I | op | rs | rt | 16 bit number | | |
| | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

Single-cycle Implementation (continued):

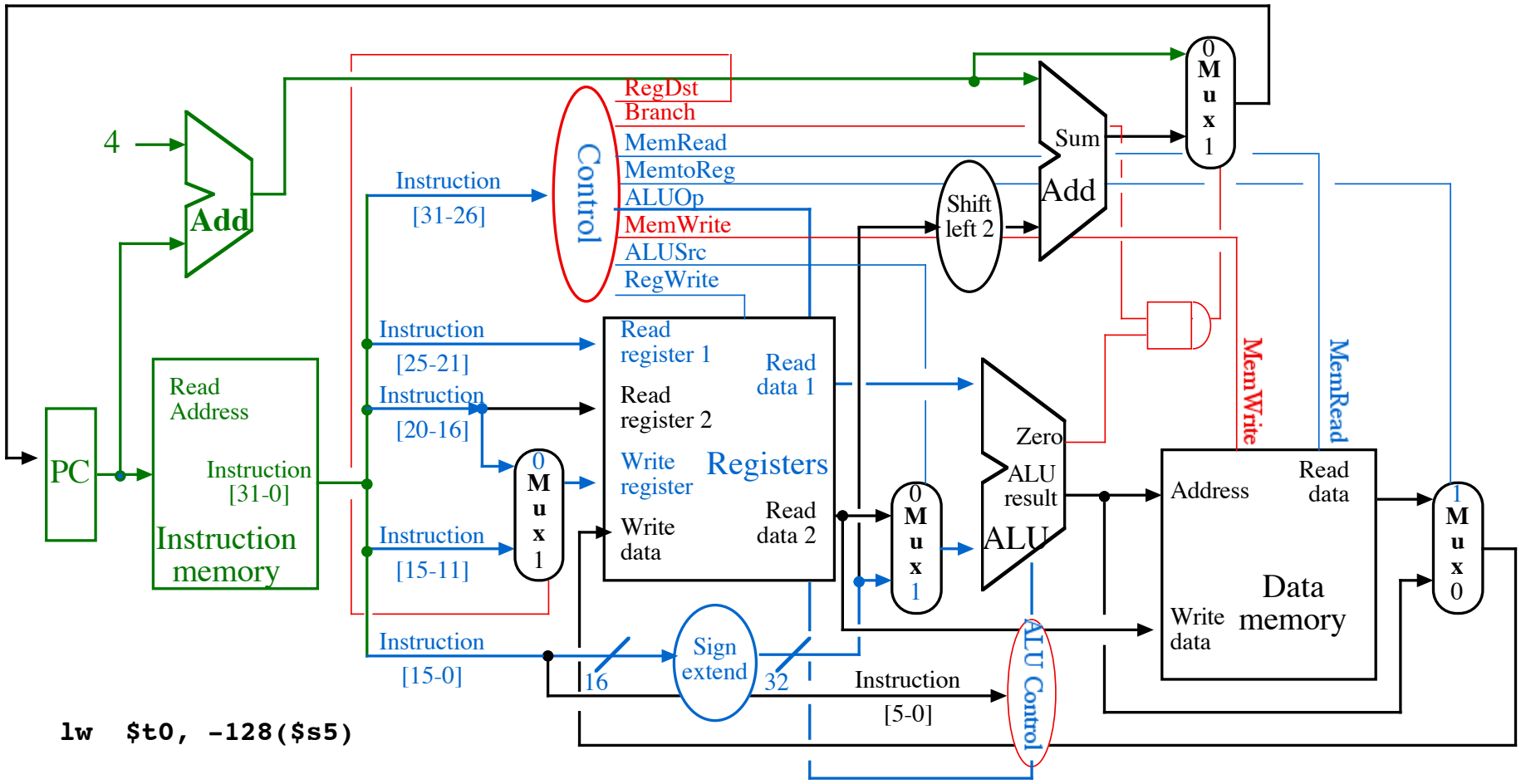
- I-Format instruction: **`lw`**, **`sw`**.
- Fetch: Get instruction from memory and increment PC.
- Fetch is exactly the same as for R-Format.
- This is the same figure as Slide 11.



| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| I | op | rs | rt | 16 bit number | | |
| | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

Single-cycle Implementation (continued):

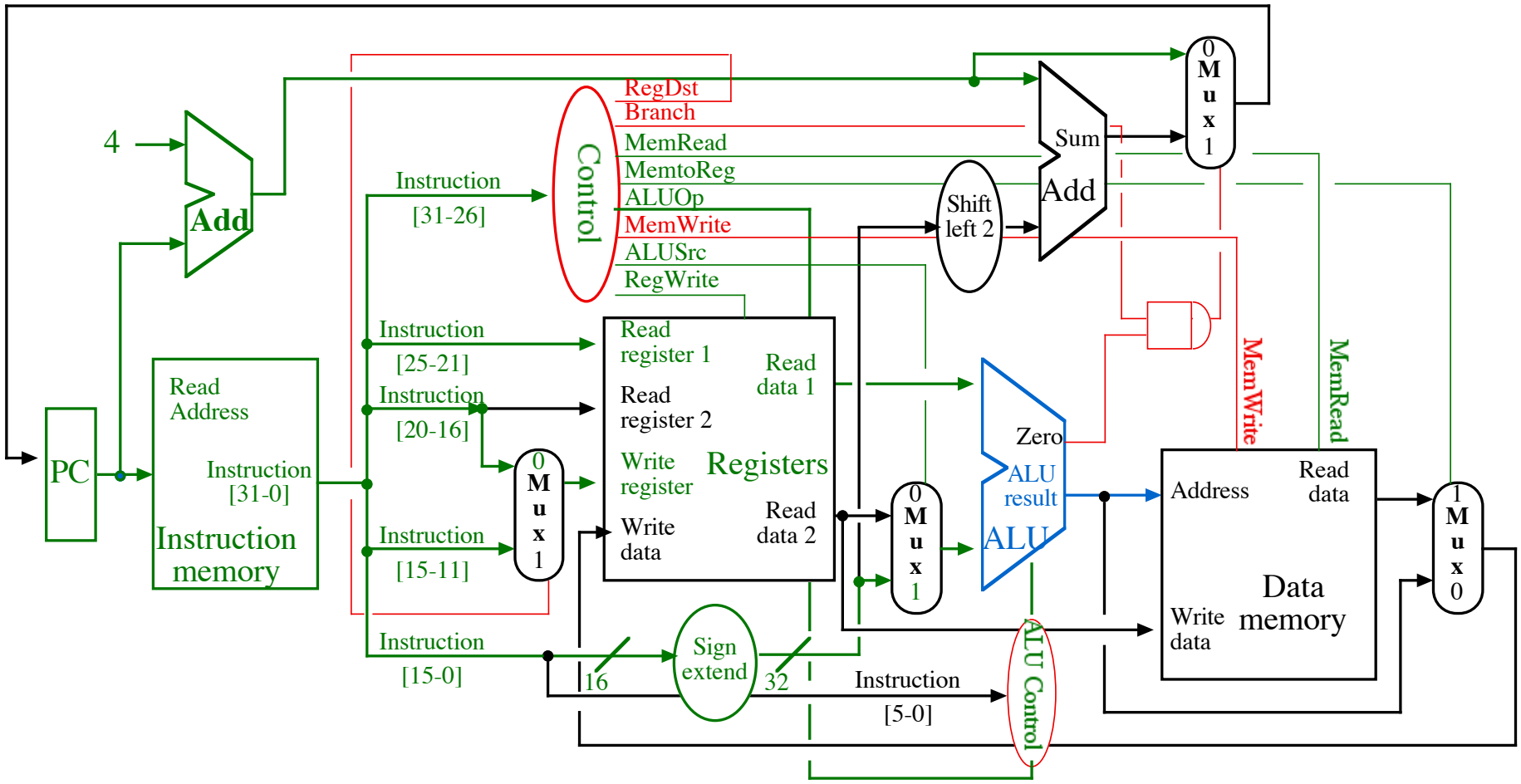
- I-Format instruction: Load word.
- One register is read and sent to ALU.
- Second register specifies the register to write result.
- Offset is sign-extended and sent to ALU.



| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| I | op | rs | rt | 16 bit number | | |
| | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

Single-cycle Implementation (continued):

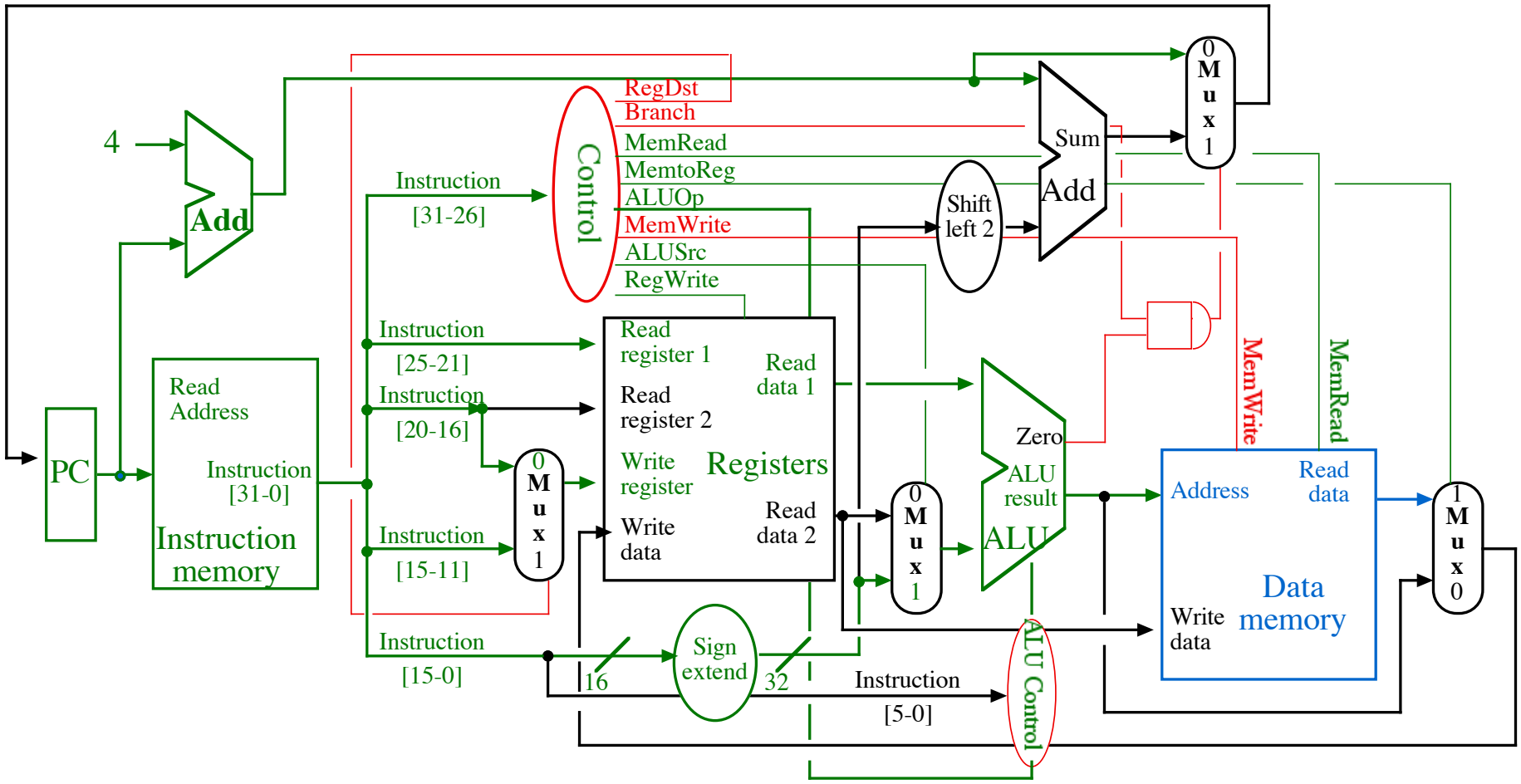
- I-Format instruction: Load word. `sw $t3, 64($s7)`
- ALU completes operation. `lw $t0, -128($s5)`



| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| I | op | rs | rt | 16 bit number | | |
| | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

Single-cycle Implementation (continued):

- I-Format instruction: Load word.
- Data memory reads contents of Address.
- Data sent to Mux.

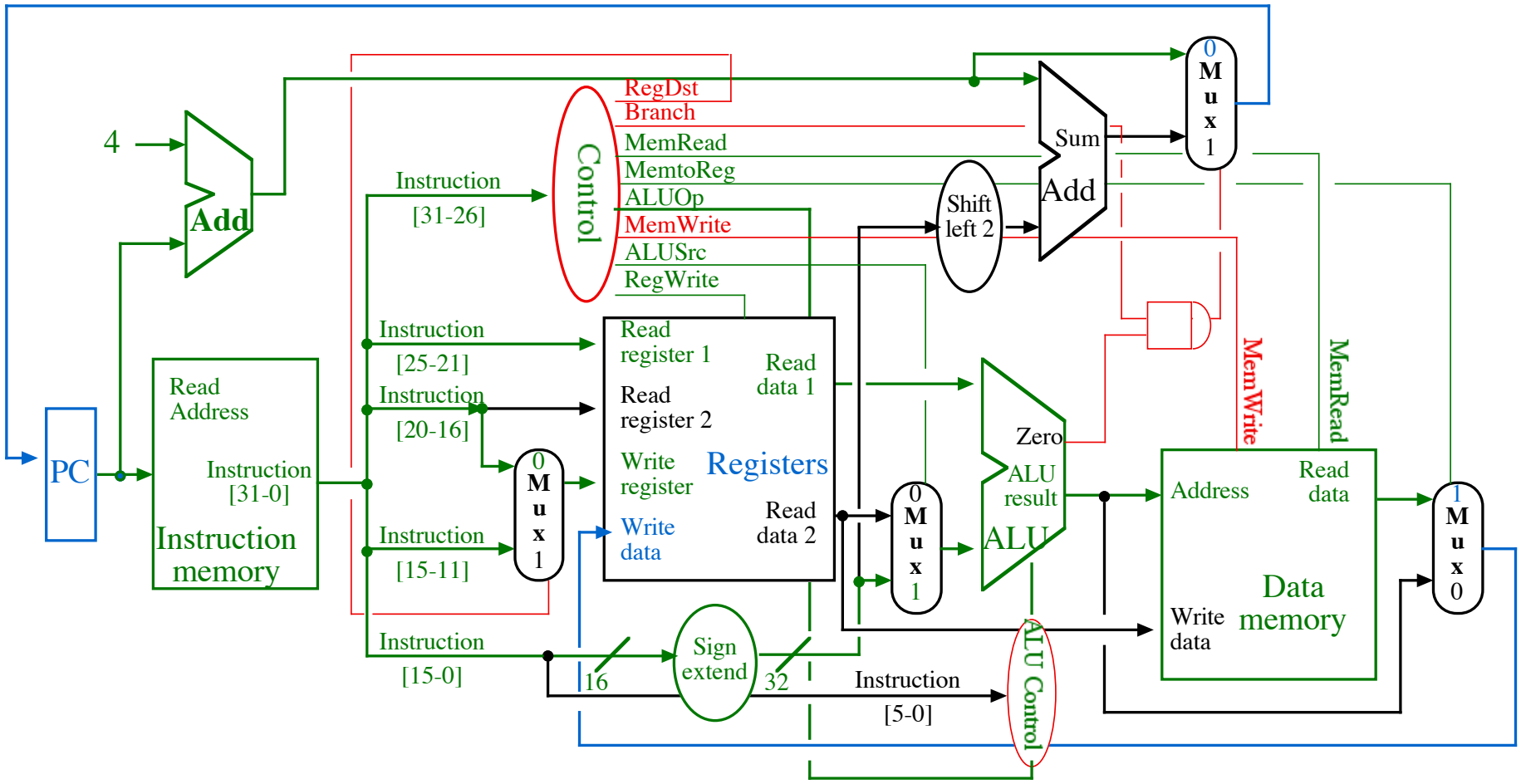


| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| I | op | rs | rt | 16 bit number | | |
| | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

Single-cycle Implementation (continued):

- I-Format instruction: Load word.
- Data sent from Mux to destination in Register File.
- PC+4 written to PC.

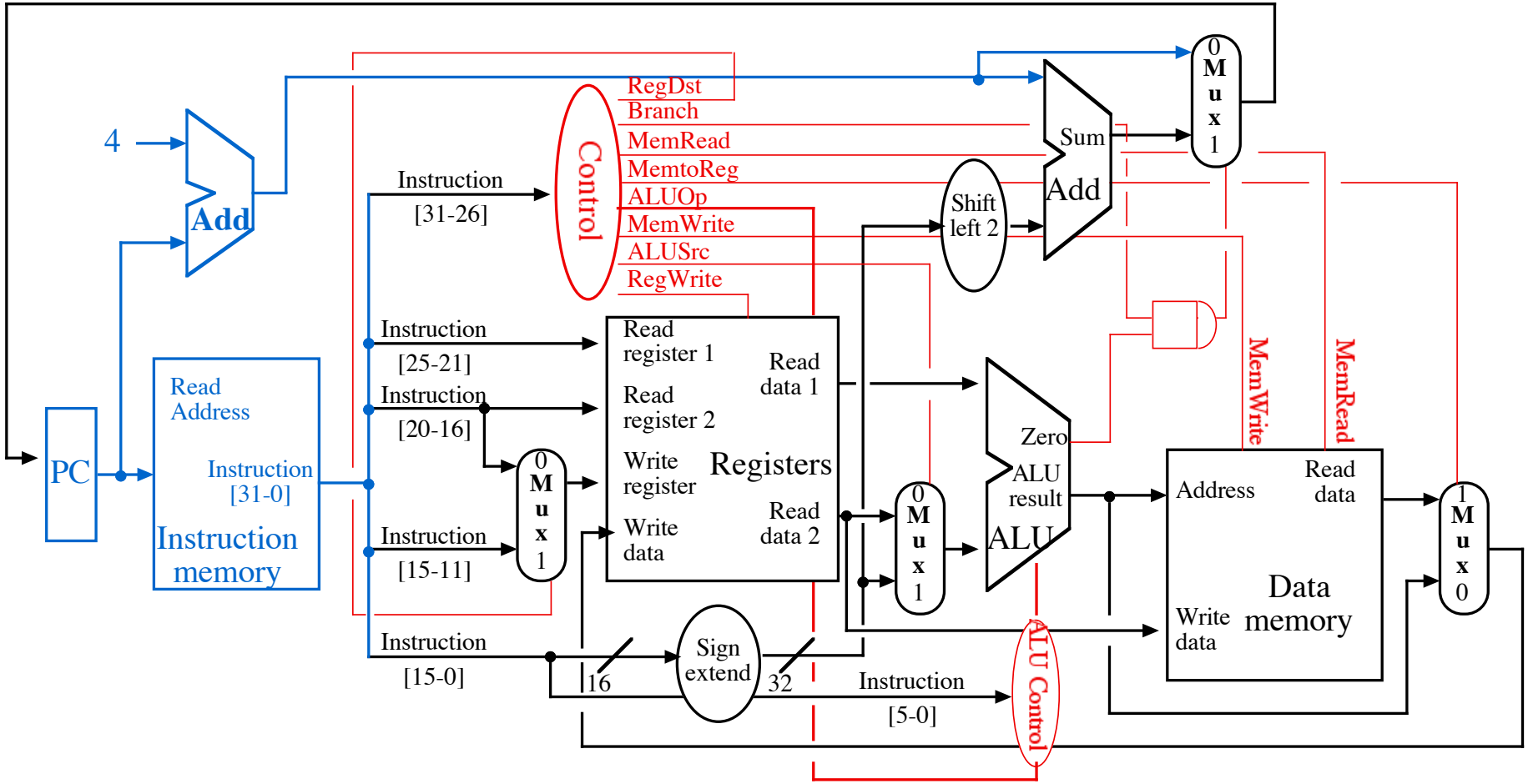
```
lw  $t0, -128($s5)
sw  $t0, -128($s5)
```

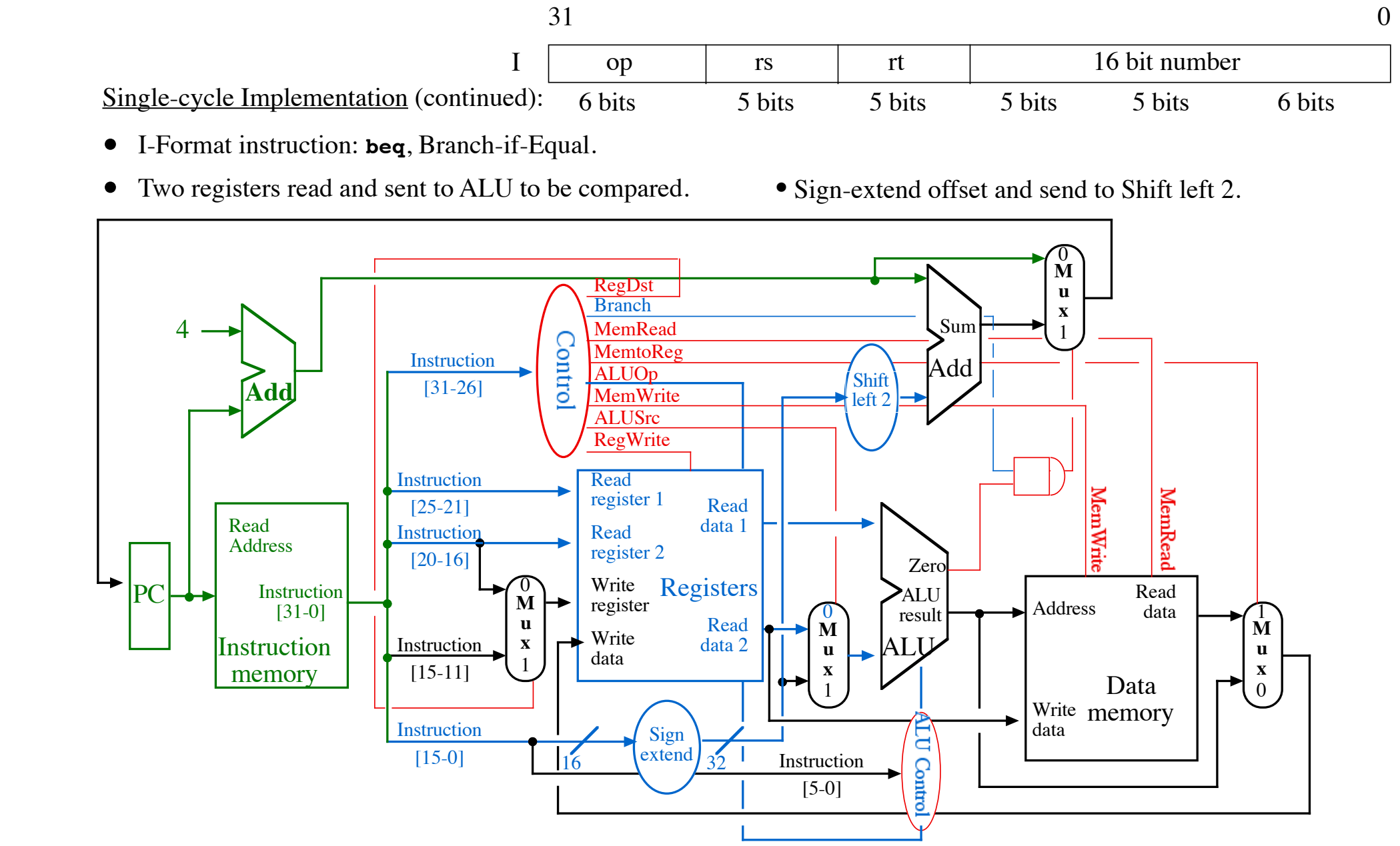


| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| I | op | rs | rt | 16 bit number | | |
| | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

## Single-cycle Implementation (continued):

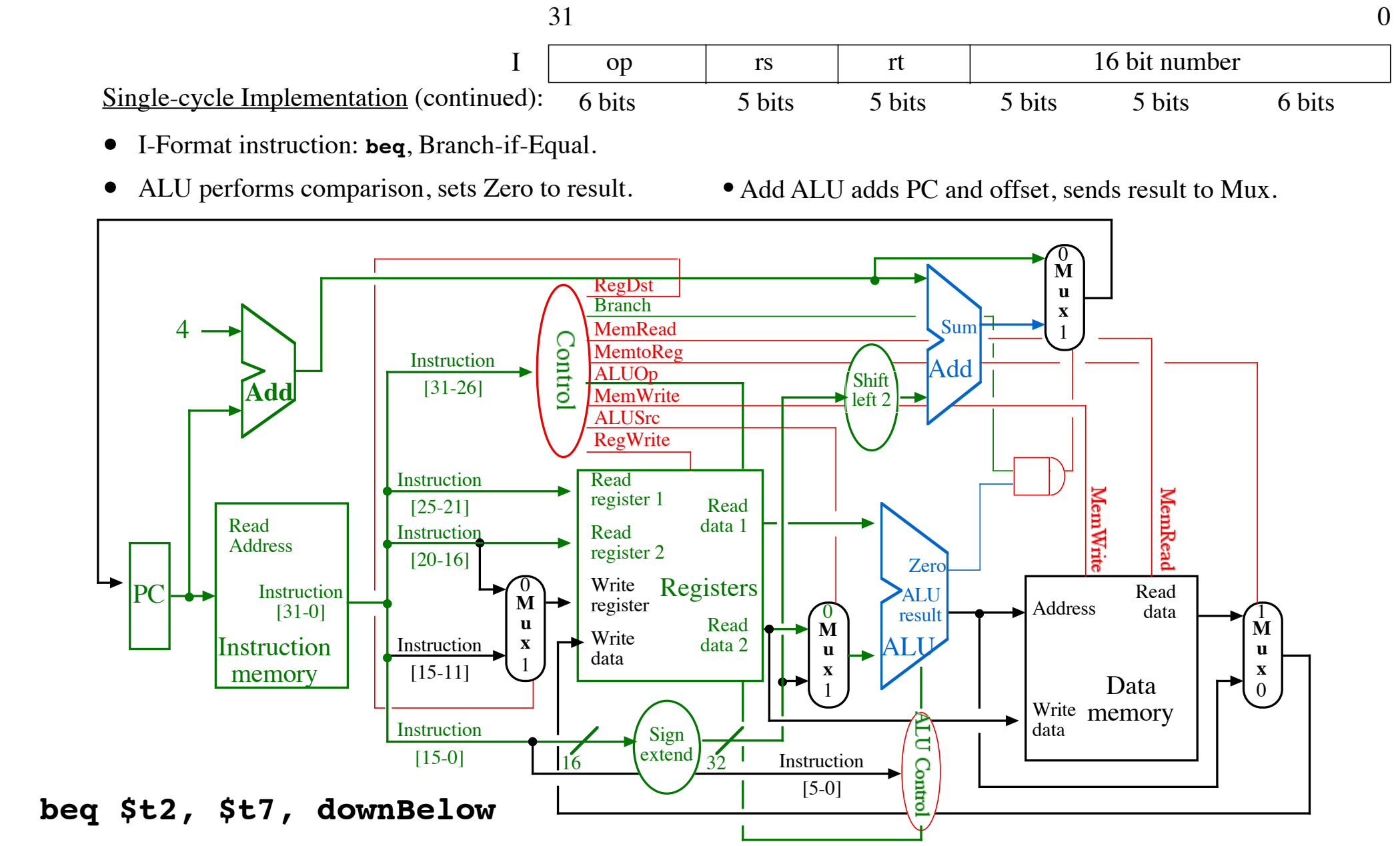
- I-Format instruction: **`beq`**, Branch-if-Equal.
- Fetch: Get instruction from memory and increment PC.
- Fetch is exactly the same as for R-Format.
- This is the same figure as Slide 11.






Single-cycle Implementation (continued):

- I-Format instruction: **`beq`**, Branch-if-Equal.
- Two registers read and sent to ALU to be compared.
- Sign-extend offset and send to Shift left 2.




Single-cycle Implementation (continued):

- I-Format instruction: **`beq`**, Branch-if-Equal.
- ALU performs comparison, sets Zero to result.
- Add ALU adds PC and offset, sends result to Mux.

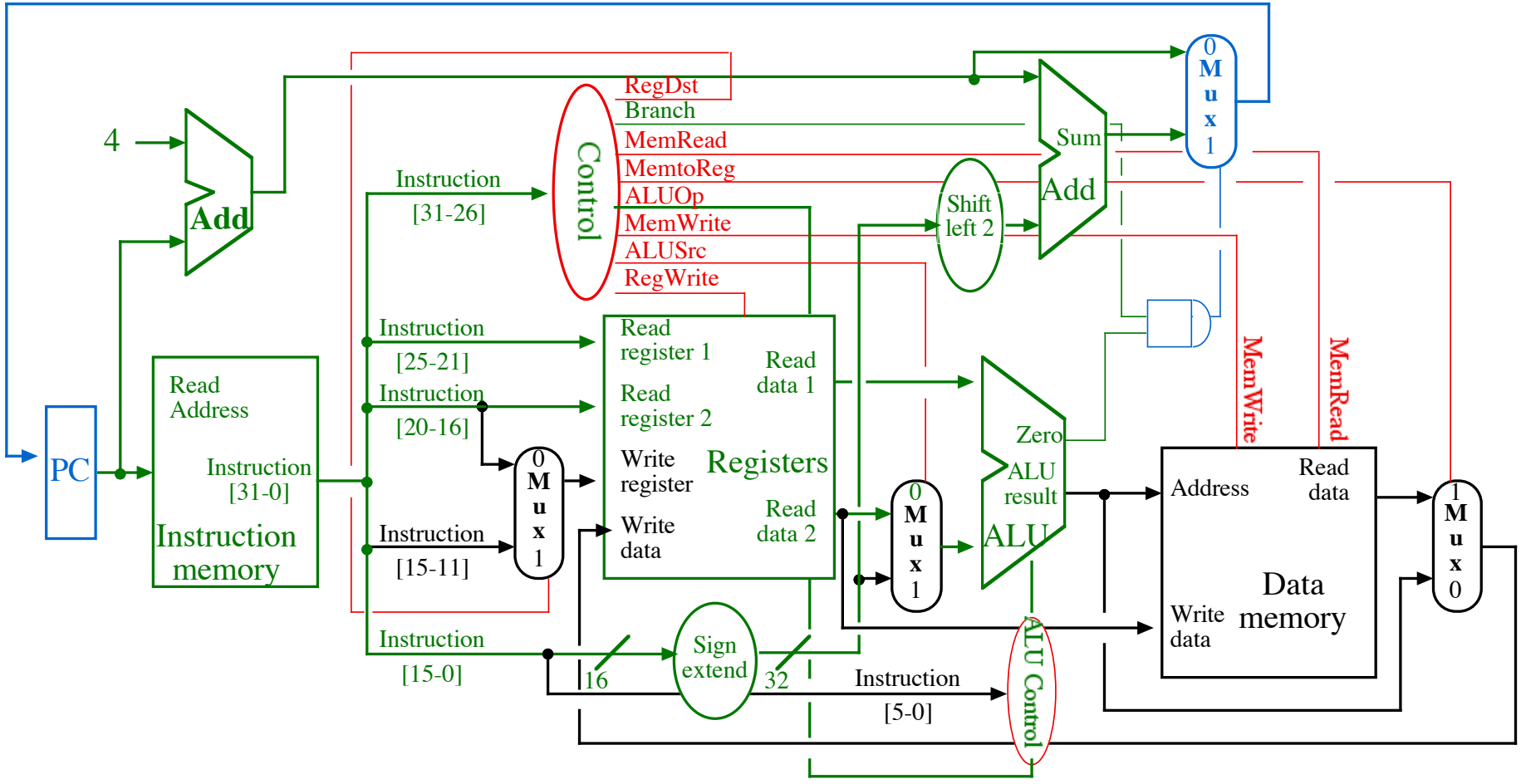


```
beq $t2, $t7, downBelow
```

| | 31 | | | | | 0 |
|---|---|---|---|---|---|---|
| I | op | rs | rt | 16 bit number | | |
| | 6 bits | 5 bits | 5 bits | 5 bits | 5 bits | 6 bits |

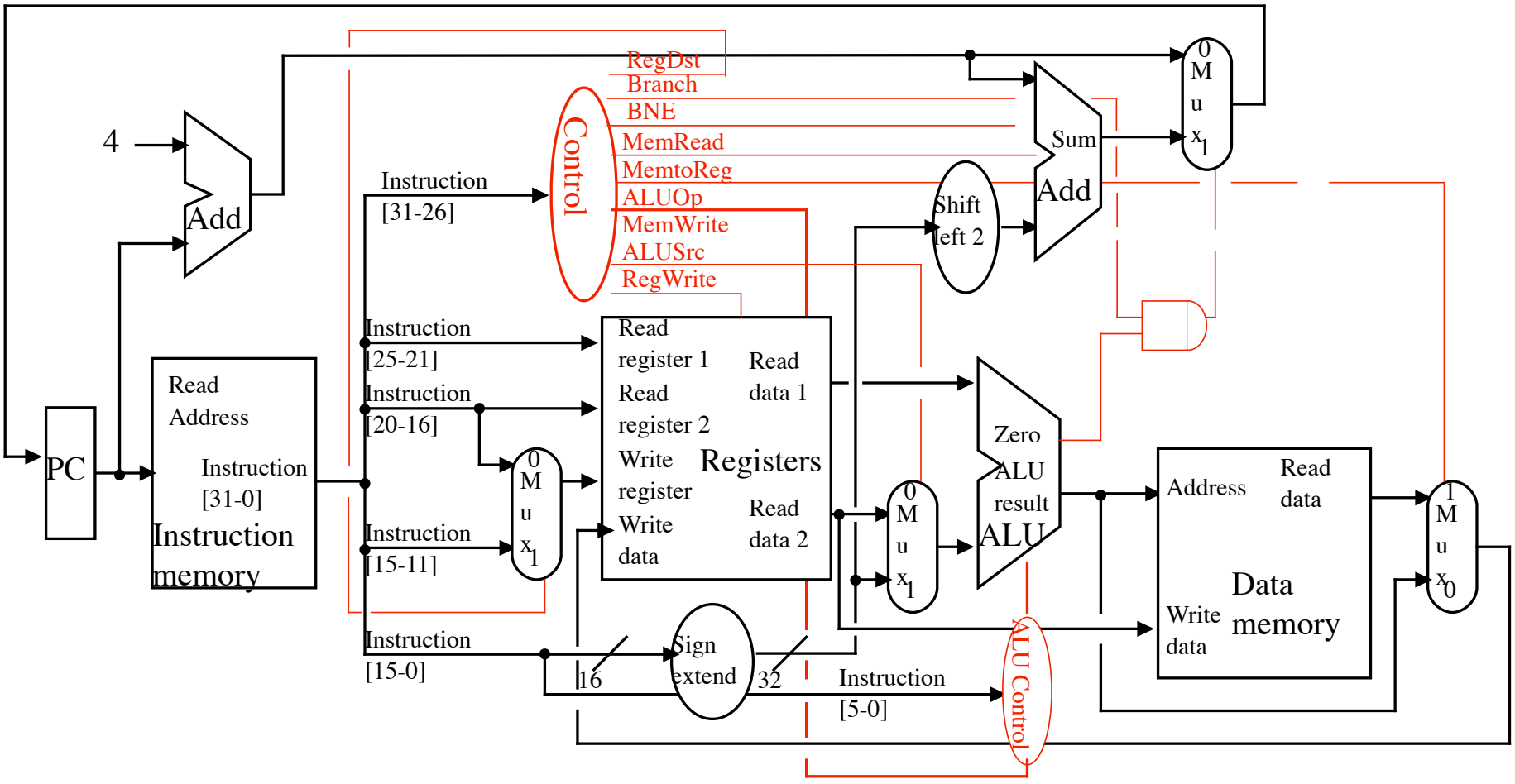
Single-cycle Implementation (continued):

- I-Format instruction: **`beq`**, Branch-if-Equal.
- AND Gate determines result of Branch AND Zero.
- Mux triggered to send correct result to PC.
- PC not updated until rising edge of clock cycle.



Single-cycle Implementation (continued):

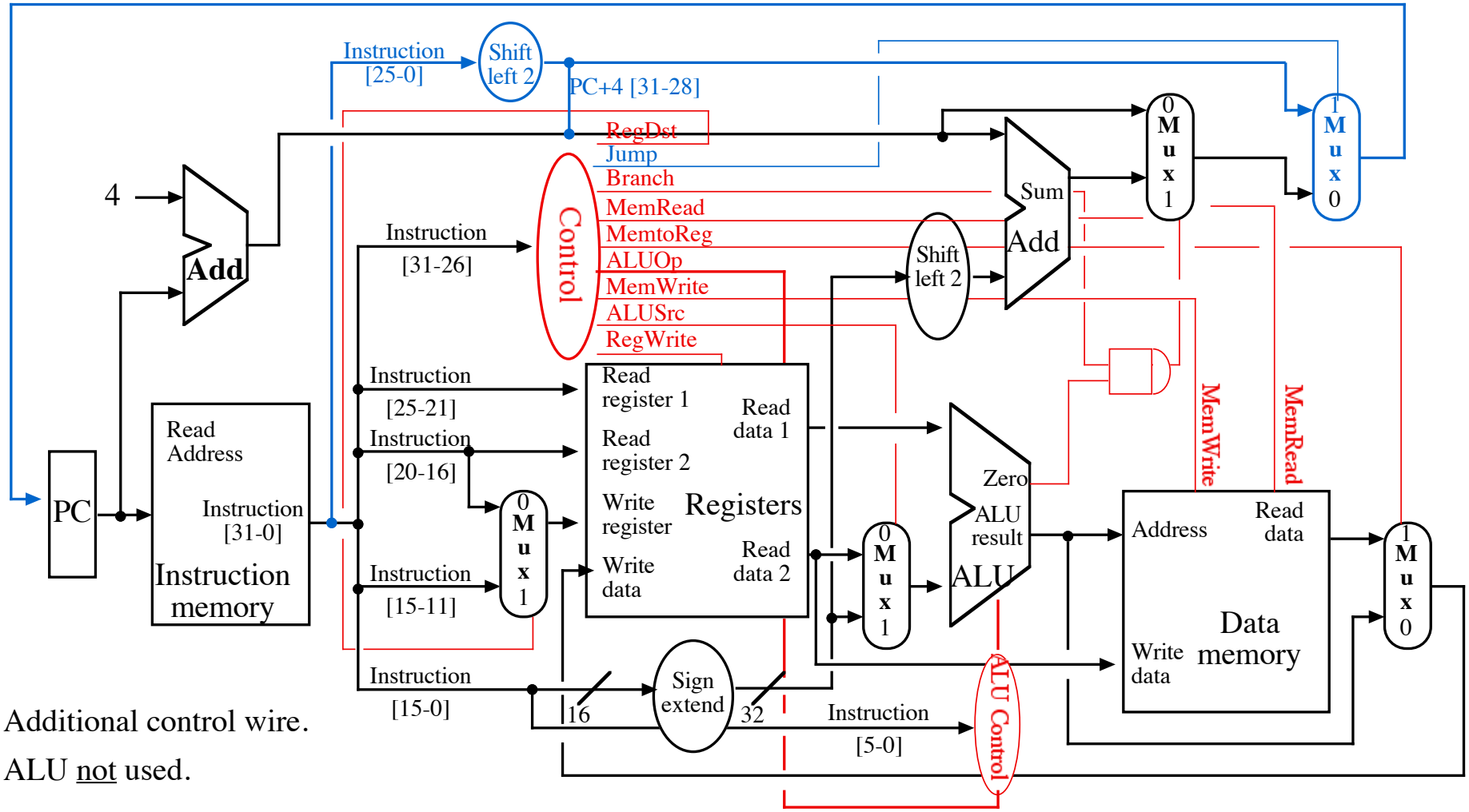
- I-Format instruction: **`bne`**, Branch-if-Not-Equal.
- What is changed/added?



| | 31 | | 0 |
|---|---|---|---|
| J | op | 26 bit address | |

Single-cycle Implementation (continued):

- J-Format instruction: `j`, jump.
- Extra items: Shift left 2, Mux.
- 26 bits shifted to give 28 bits + 4 bits from PC+4.




- Additional control wire.
- ALU not used.