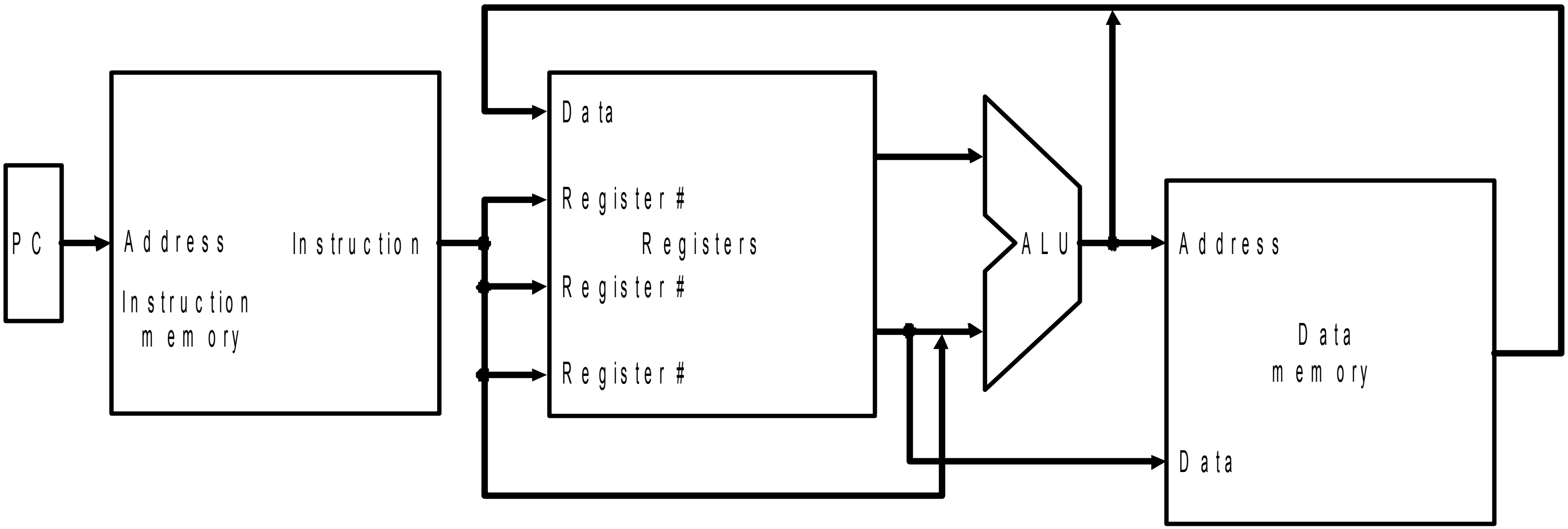
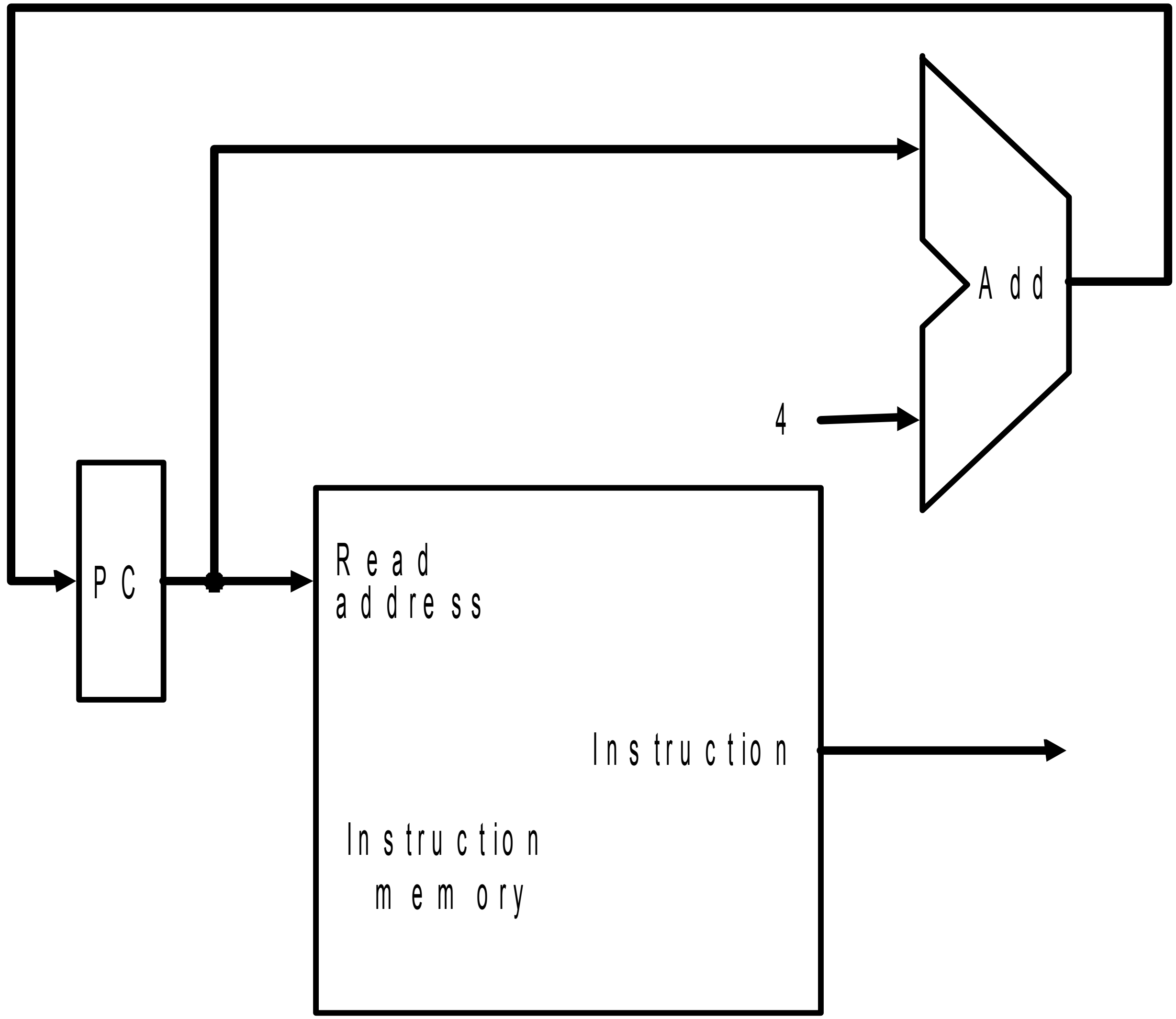
- **MIPS Instruction Set Processor**

# Basic Abstract View of the Data Path

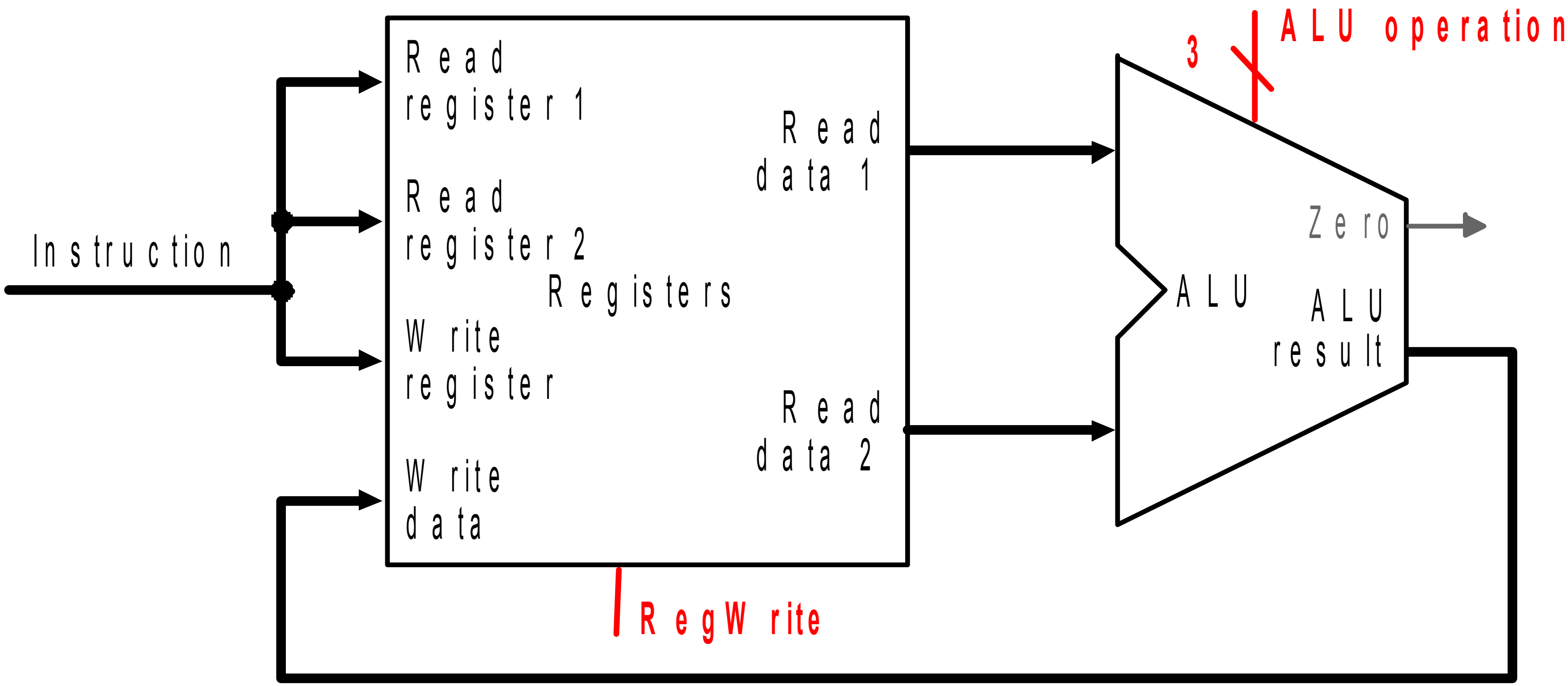



**Shows common functions for most instructions**

# Data Path for Instruction Fetching

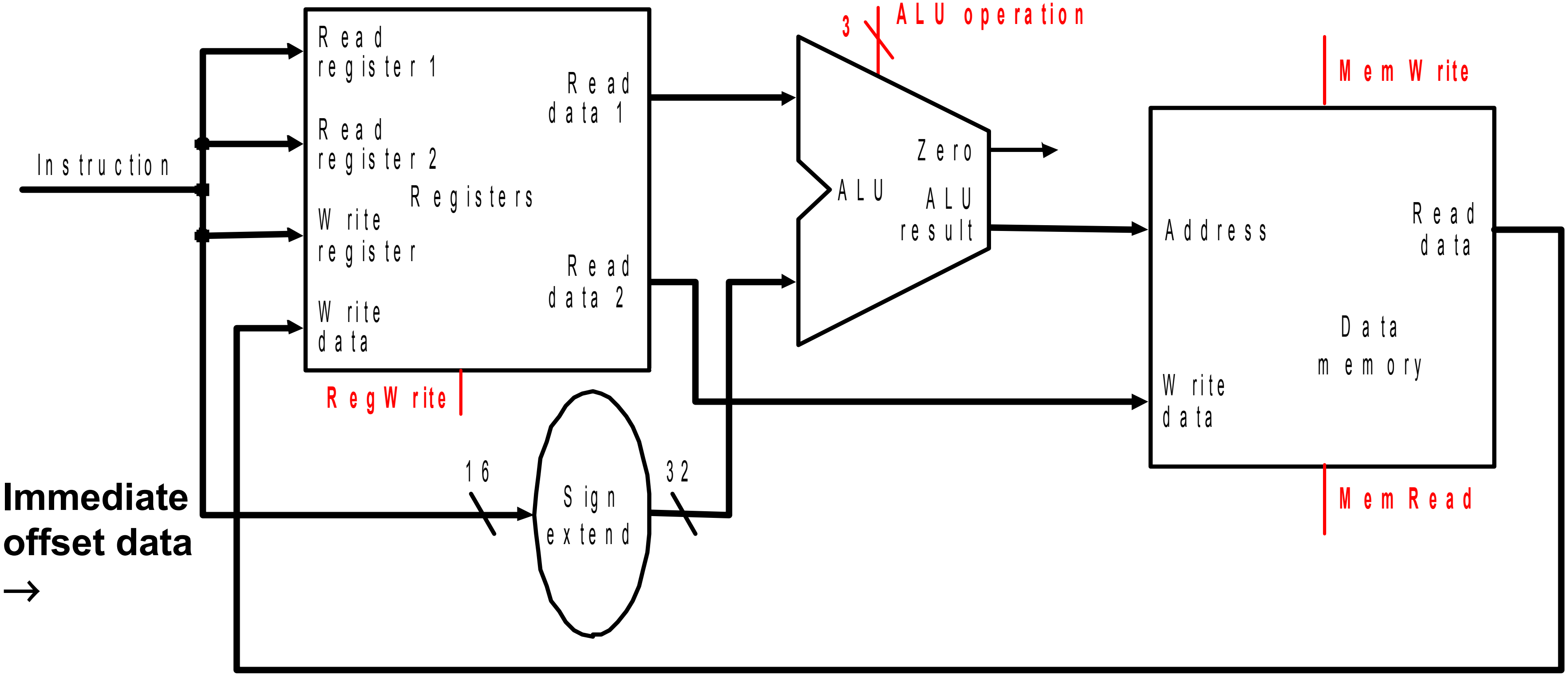


# Basic Data Path for R-type Instruction




**Red lines are for control signals generated by the controller**

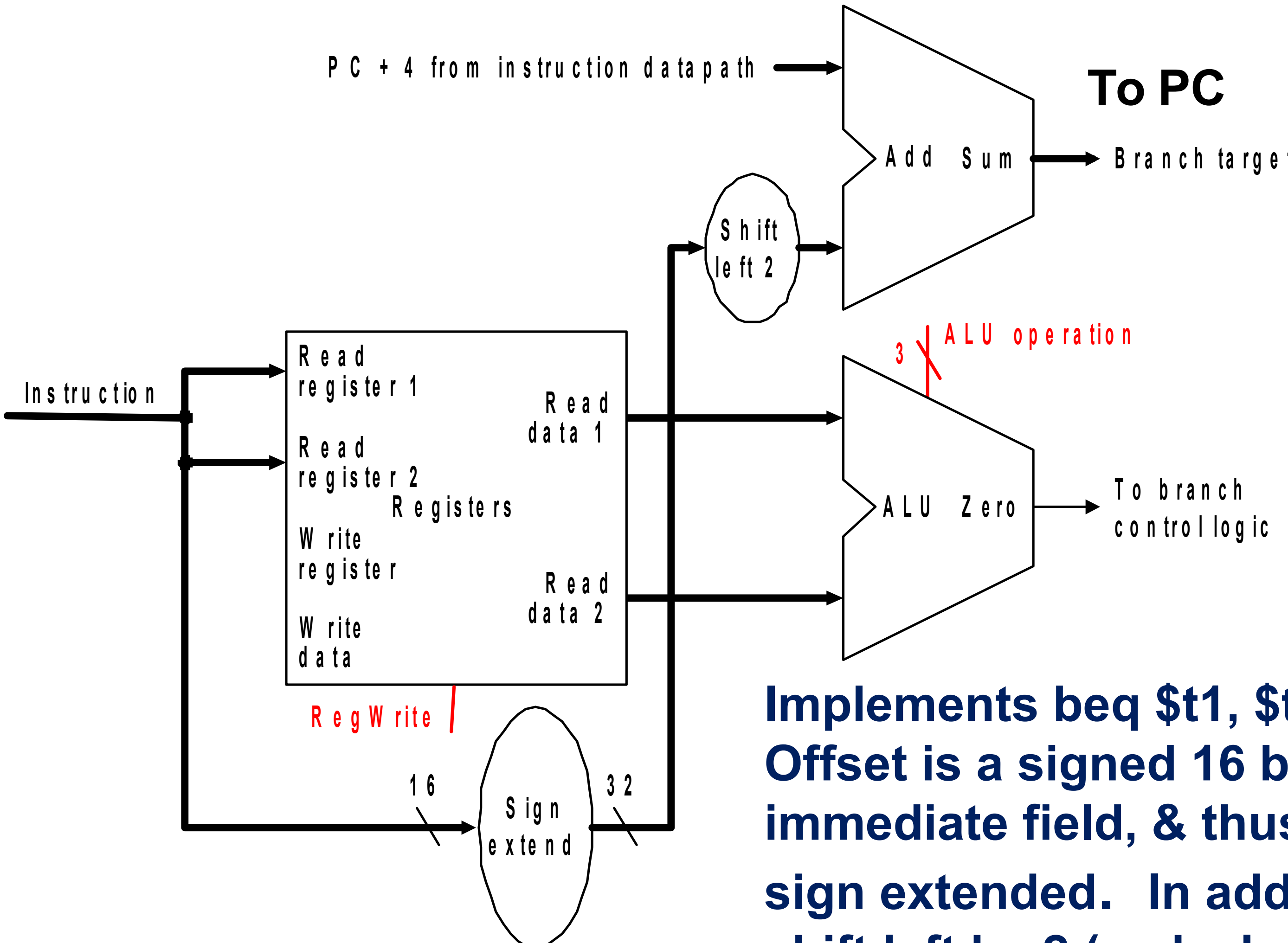
# Adding the Data Path for lw & sw Instruction




**Immediate offset data →**

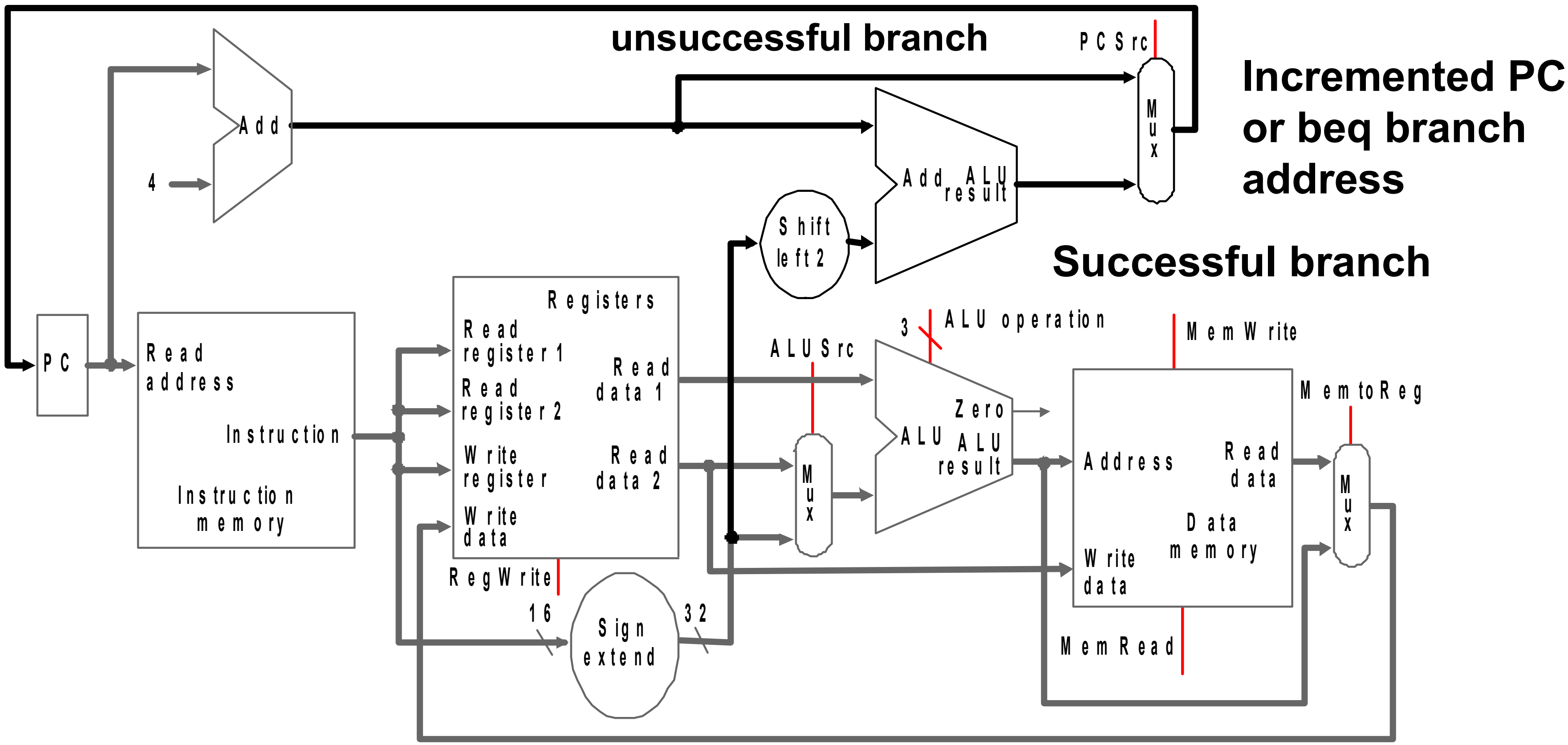
**Implements:**
**lw $t1, offset_value($t2)**
**sw $t1, offset_value($t2)**
**The offset value is a 16-bit signed immediate field & must be sign extended to 32 bits**

# Adding the Data Path for beq Instruction




**Implements beq $t1, $t2, offset**
**Offset is a signed 16 bit immediate field, & thus must be sign extended. In addition we shift left by 2 (make low bits are 00) to address to a word boundary**

# Putting It All Together




j instruction to be added later

Need control circuits to drive control lines in red.

Two control units will be designg: ALU Control & “Main Control

| Instruction | RegDst | RegWrite | ALUSrc | MemRead | MemWrite | MemToReg | PCSrc | ALU operation |
|---|---|---|---|---|---|---|---|---|
| R-format | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0000 (and)<br>0001 (or)<br>0010 (add)<br>0110 (sub) |
| lw | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0010 (add) |
| sw | X | 0 | 1 | 0 | 1 | X | 0 | 0010 (add) |
| beq | x | 0 | 0 | 0 | 0 | X | 1 or 0 | 0110 (sub) |

# Control

- We next add the control unit that generates
  - write signal for each state element
  - control signals for each multiplexer
  - ALU control signal
- Input to control unit: instruction opcode and function code

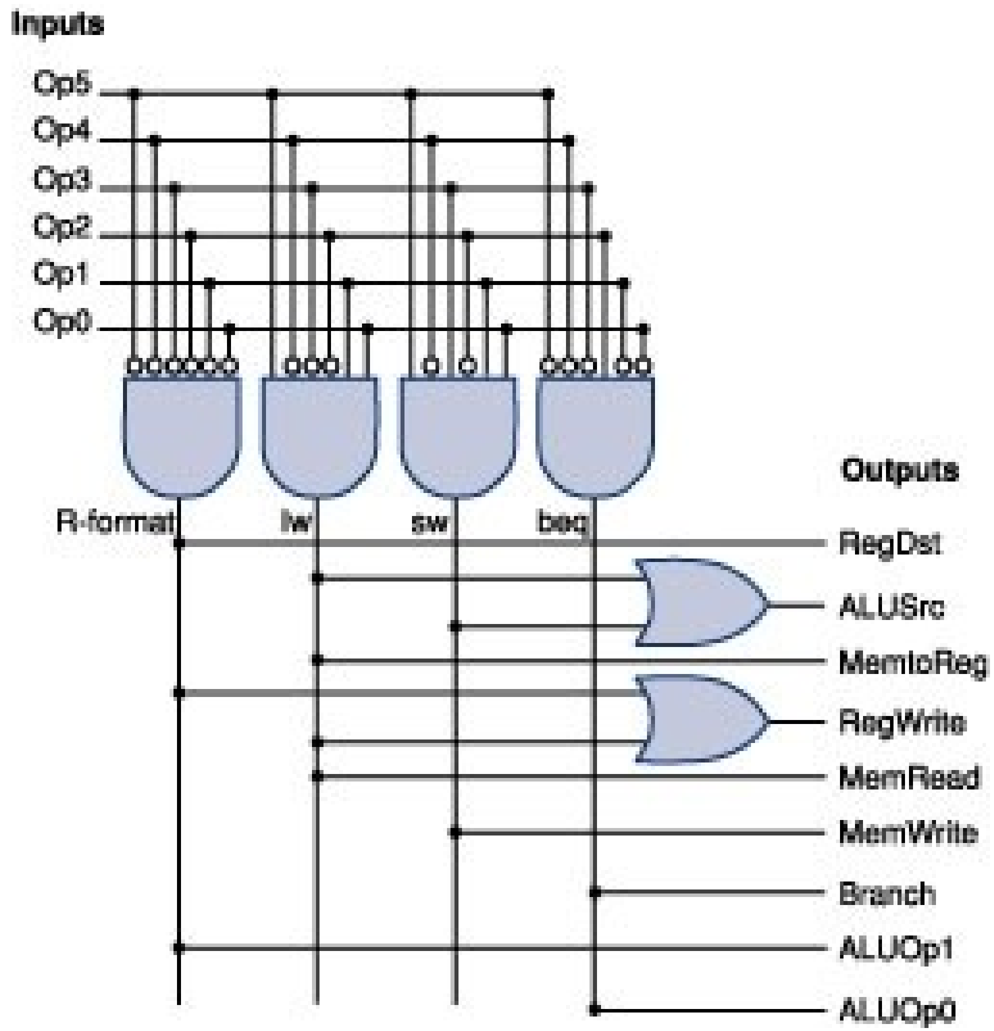
# Control Unit

- Divided into two parts
  - Main Control Unit
    - Input: 6-bit opcode
    - Output: all control signals for Muxes, RegWrite, MemRead, MemWrite and a 2-bit ALUOp signal
  - ALU Control Unit
    - Input: 2-bit ALUOp signal generated from Main Control Unit and 6-bit instruction function code
    - Output: 4-bit ALU control signal

# Truth Table for Main Control Unit

| 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 |

# Main Control Unit



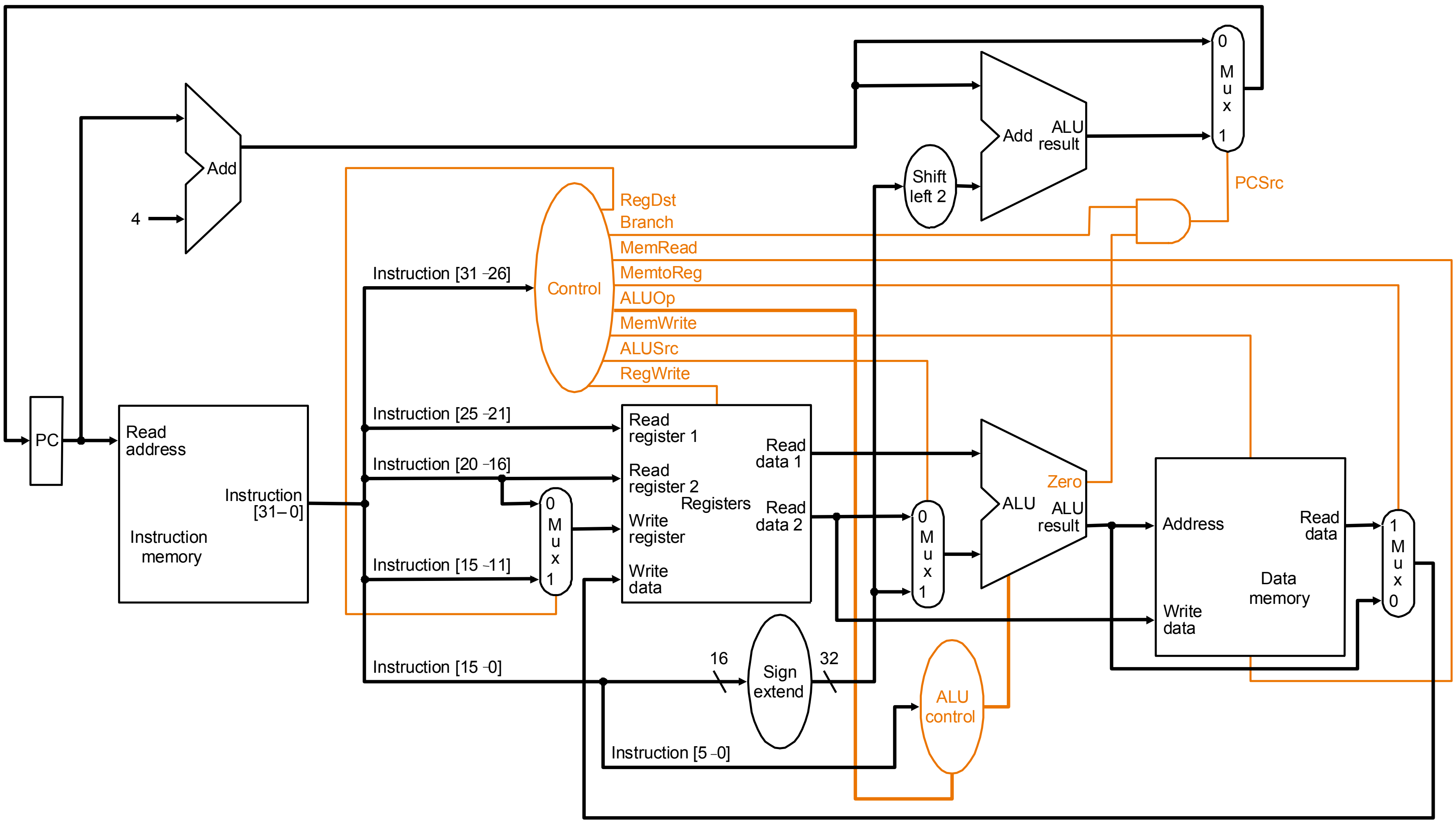
# ALU Control Unit

- Must describe hardware to compute 4-bit ALU control input given
  - 2-bit ALUOp signal from Main Control Unit
  - function code for arithmetic
- Describe it using a truth table (can turn into gates):

# ALU Control bits

| Instruction opcode | ALUOp | Instruction operation | Funct field | Desired ALU action | ALU control input |
|---|---|---|---|---|---|
| LW | 00 | load word | XXXXXX | add | 0010 |
| SW | 00 | store word | XXXXXX | add | 0010 |
| Branch equal | 01 | branch equal | XXXXXX | subtract | 0110 |
| R-type | 10 | add | 100000 | add | 0010 |
| R-type | 10 | subtract | 100010 | subtract | 0110 |
| R-type | 10 | AND | 100100 | and | 0000 |
| R-type | 10 | OR | 100101 | or | 0001 |
| R-type | 10 | set on less than | 101010 | set on less than | 0111 |

# Putting It All Together Again

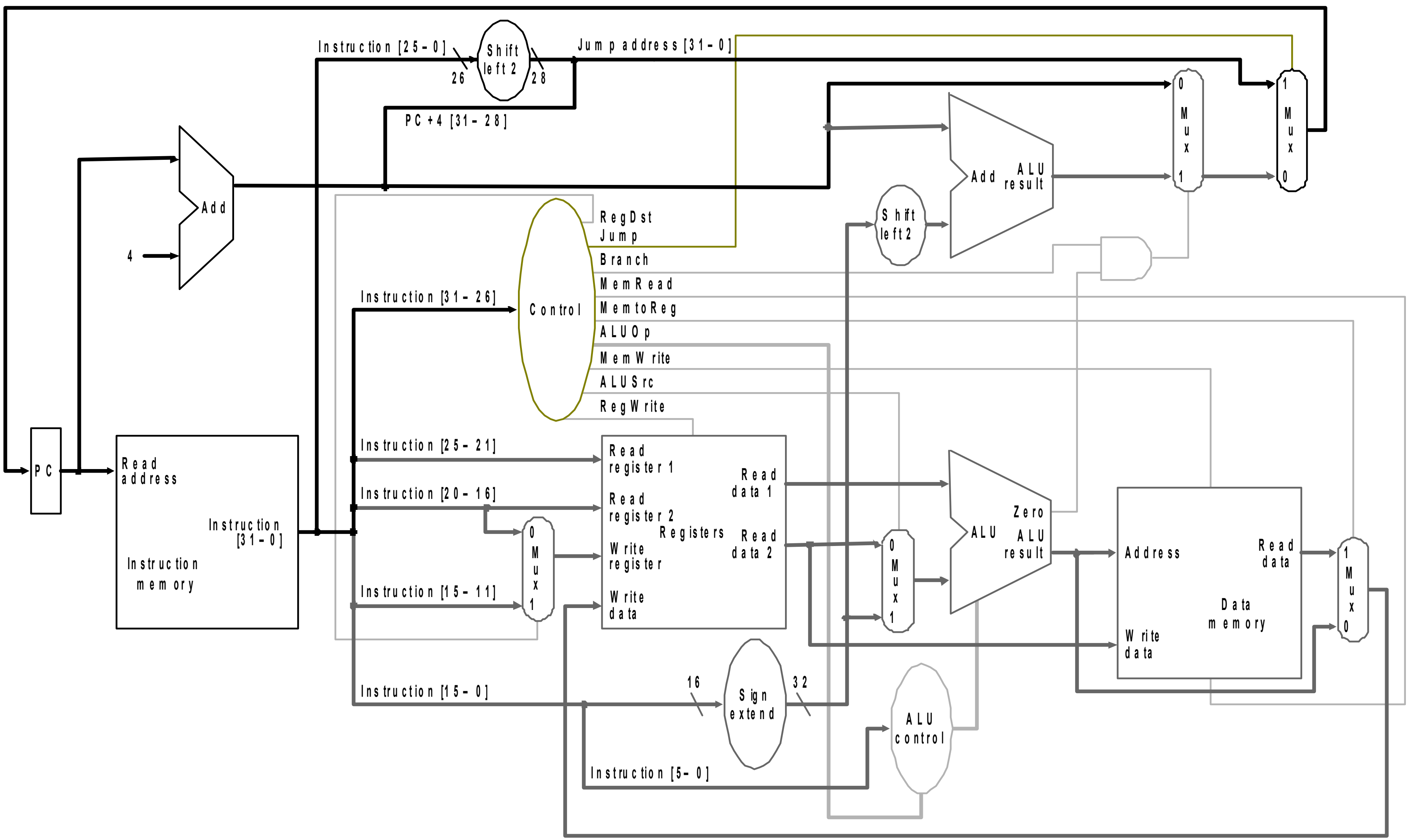



Use this for R-type, memory, & beq instructions scenarios.

# Addition of the Unconditional Jump

- We now add one more op code to our single cycle design:
  - Op code 2: “j”
  - The format is op field 28-31 is a “2”
  - Remaining 26 low bits is the immediate target address
- The full 32 bit target address is computed by concatenating:
  - Upper 4 bits of PC+4
  - 26 bit immediate field of the jump instruction
  - Bits 00 in the lowest positions (word boundary)
  - See text chapter 3, p. 150
- An additional control line from the main controller will have to be generated to select this “new” instruction
- A two bit shifter is also added to get the two low order zeros

# Final Design with jump Instruction



**Thanks!!**