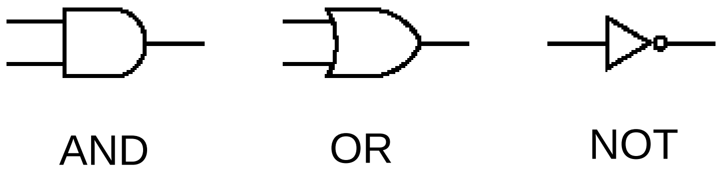
# Computer Architecture

Basics of Datapath

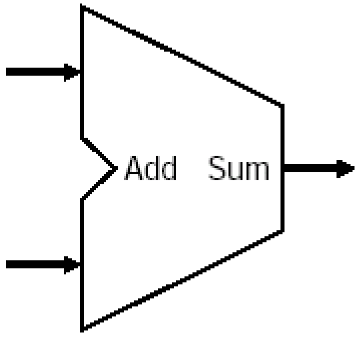
# Combinational Logic Elements

- Stateless logic
  - No embedded state (memory)
  - Output fully dependent on inputs
  - Any function possible
    - AND, OR, NAND, NOR, XOR, NOT…

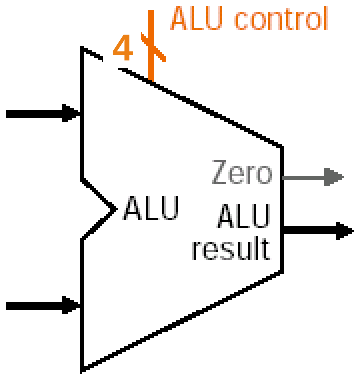


# Combinational Logic Elements

- Adder



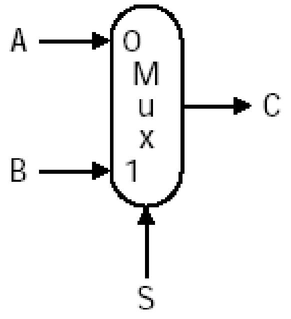
- ALU

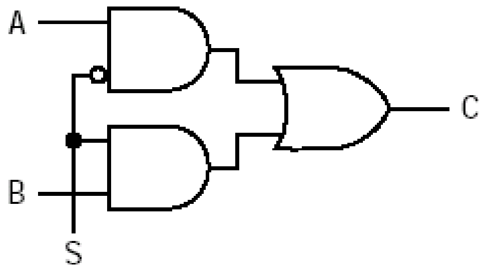


| ALU control input | Function |
|---|---|
| 0000 | AND |
| 0001 | OR |
| 0010 | add |
| 0110 | subtract |
| 0111 | set on less than |

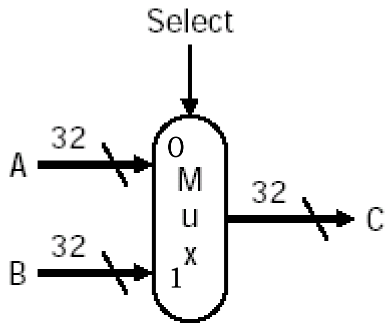
# Combinational Logic Elements

- Multiplexor
  - A two-input multiplexor



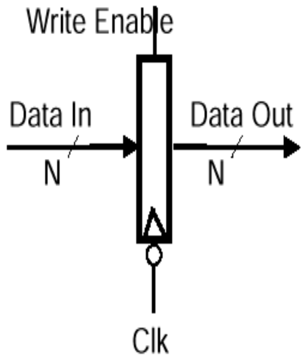


  - A 32-bit wide 2-to-1 multiplexor



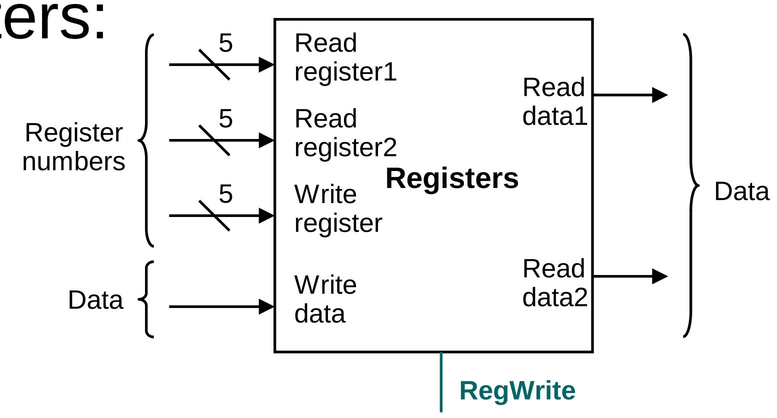
## Storage Elements

- Register
  - Similar to the D Flip Flop except
    - N-bit input and output
    - Write Enable input
  - Write Enable :
    - 0 : Data Out will not change
    - 1 : Data Out will become Data in
  - Stored data changes only on falling clock edge!



# Storage Elements

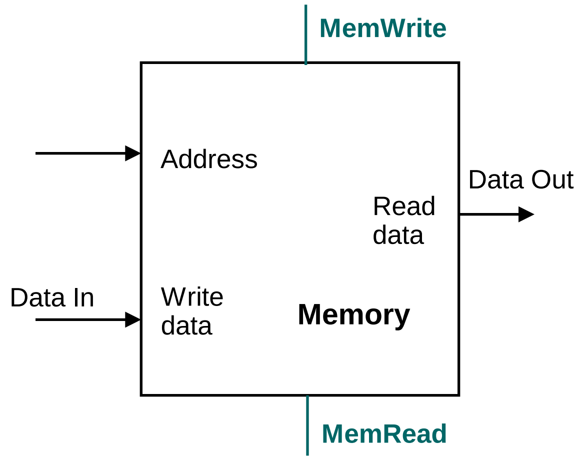
- Register File consists of 32 registers:
  - Two 32-bit output busses:
    - Read data1 and Read data2
  - One 32-bit input bus: Write data
  - Register 0 hard-wired to value 0



- Register is selected by:
  - Read register1 selects the register to put on Read data1
  - Read register2 selects the register to put on Read data2
  - Write register selects the register to be written via write data when RegWrite = 1
- Clock input (CLK)
  - The CLK input is a factor only for write operation (data changes only on falling clock edge)

# Storage Elements

- Memory has two busses:
  - One output bus : Read data (Data Out)
  - One input bus : Write data (Data In)



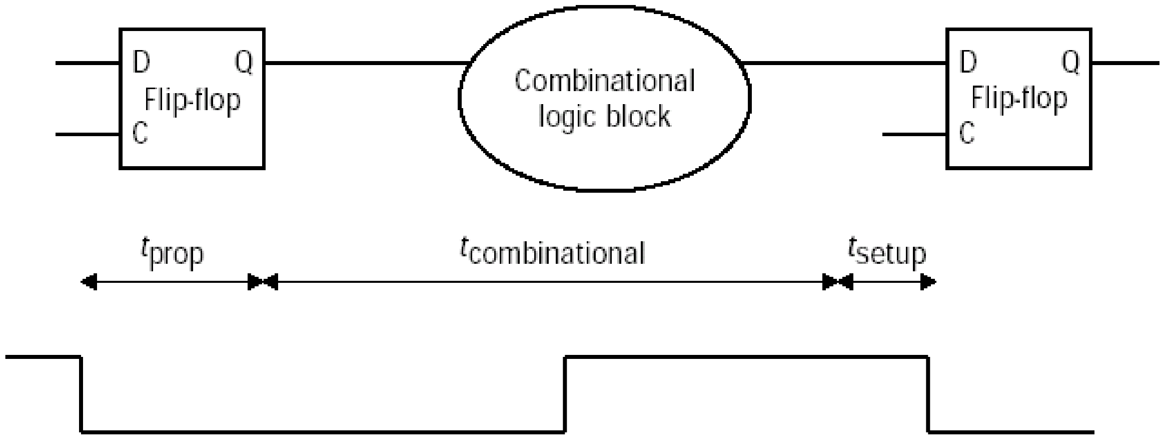
- Address selects
  - the word to put on Data Out when MemRead = 1
  - the word to be written via the Data In when MemWrite = 1

- Clock input (CLK)
  - The CLK input is a factor only for write operation (data changes only on falling clock edge)

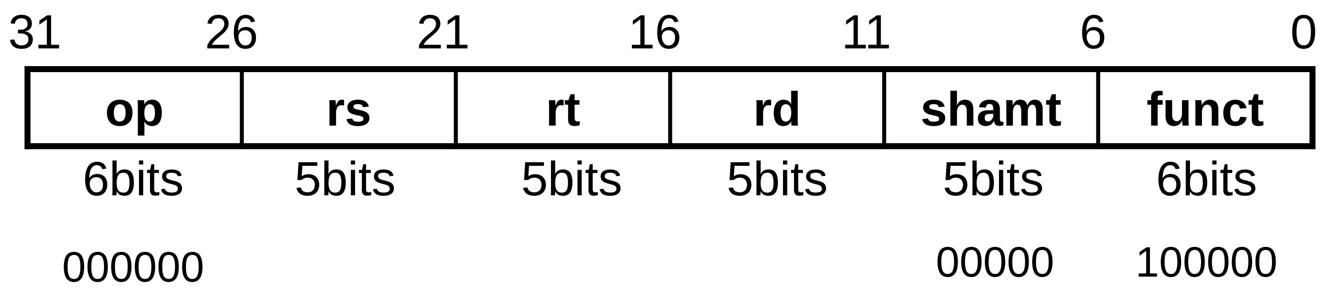
## Clocking

- All storage elements clocked by the same clock edge
  - Edge-triggered clocking (falling clock edge)
  - One clock period per clock cycle
  - Design always works if the clock is “slow enough”

$$\text{Cycle Time} = t_{prop} + t_{combinational} + t_{setup} + t_{skew}$$



# add Instruction



- add  rd, rs, rt

| | RTL Description |
|---|---|
| IR ← mem[PC]; | Fetch instruction from memory |
| R[rd] ← R[rs] + R[rt]; | ADD instruction |
| PC ← PC + 4; | Calculate next address |

# sub Instruction

31 26 21 16 11 6 0

| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 6bits | 5bits | 5bits | 5bits | 5bits | 6bits |
| 000000 | | | | 00000 | 100010 |

- sub rd, rs, rt

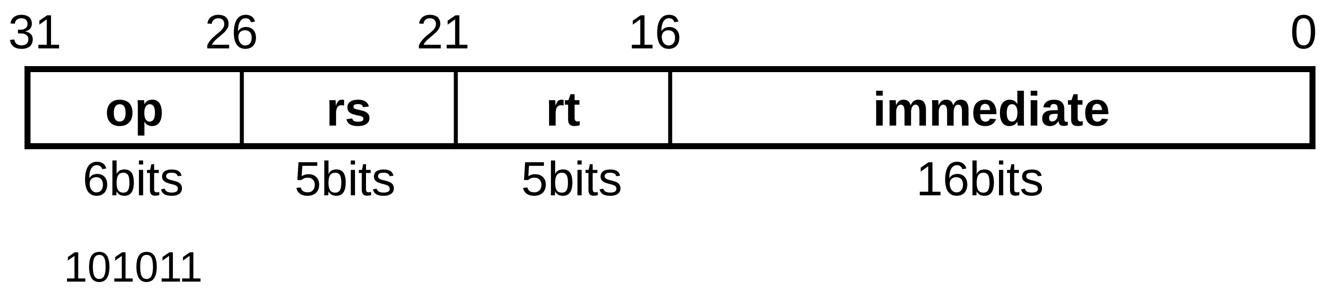
| | RTL Description |
|---|---|
| IR ← mem[PC]; | Fetch instruction from memory |
| R[rd] ← R[rs] + ~R[rt] + 1; | SUB instruction |
| PC ← PC + 4; | Calculate next address |

# lw Instruction

| 31 | 26 | 21 | 16 | 0 |
|---|---|---|---|---|
| **op** | **rs** | **rt** | **immediate** | |
| 6bits | 5bits | 5bits | 16bits | |
| 100011 | | | | |

| ❑ lw rt, rs, imm16 | RTL Description |
|---|---|
| IR ← mem[PC]; | Fetch instruction from memory |
| Addr ← R[rs] + SignExt(imm16); | Compute memory address |
| R[rt] ← Mem[Addr]; | Load data into register |
| PC ← PC + 4; | Calculate next address |

# sw Instruction



| ❑ sw rt, rs, imm16 | RTL Description |
|---|---|
| IR ← mem[PC]; | Fetch instruction from memory |
| Addr ← R[rs] + SignExt(imm16); | Compute memory address |
| Mem[Addr] ← R[rt]; | Store data into memory |
| PC ← PC + 4; | Calculate next address |

# beq Instruction

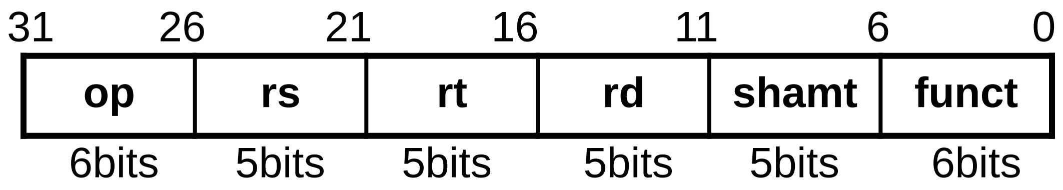
| 31     26 | 21 | 16 | 0 |
|---|---|---|---|
| **op** | **rs** | **rt** | **immediate** |
| 6bits | 5bits | 5bits | 16bits |
| 000100 | | | |

- beq rt, rs, imm16 — RTL Description

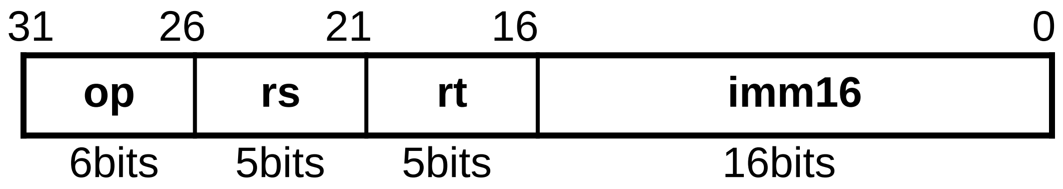
```
IR ← mem[PC];                                  Fetch instruction from memory
Cond ← R[rs] + ~R[rt] + 1;                     Compute conditional Cond
if (Cond == 0) then
        PC ← PC + 4 + (SignExt(imm16) << 2);
            (Branch if equal)
        else
        PC ← PC + 4;
            (Fall through otherwise)
```

# Instruction Format Summary

- Add, sub, and, or
  - add    rd,  rs,  rt
  - sub    rd,  rs,  rt
- Load, store
  - lw    rt,  rs,  imm16
  - sw    rt,  rs,  imm16
- Branch
  - beq    rt,  rs,  imm16

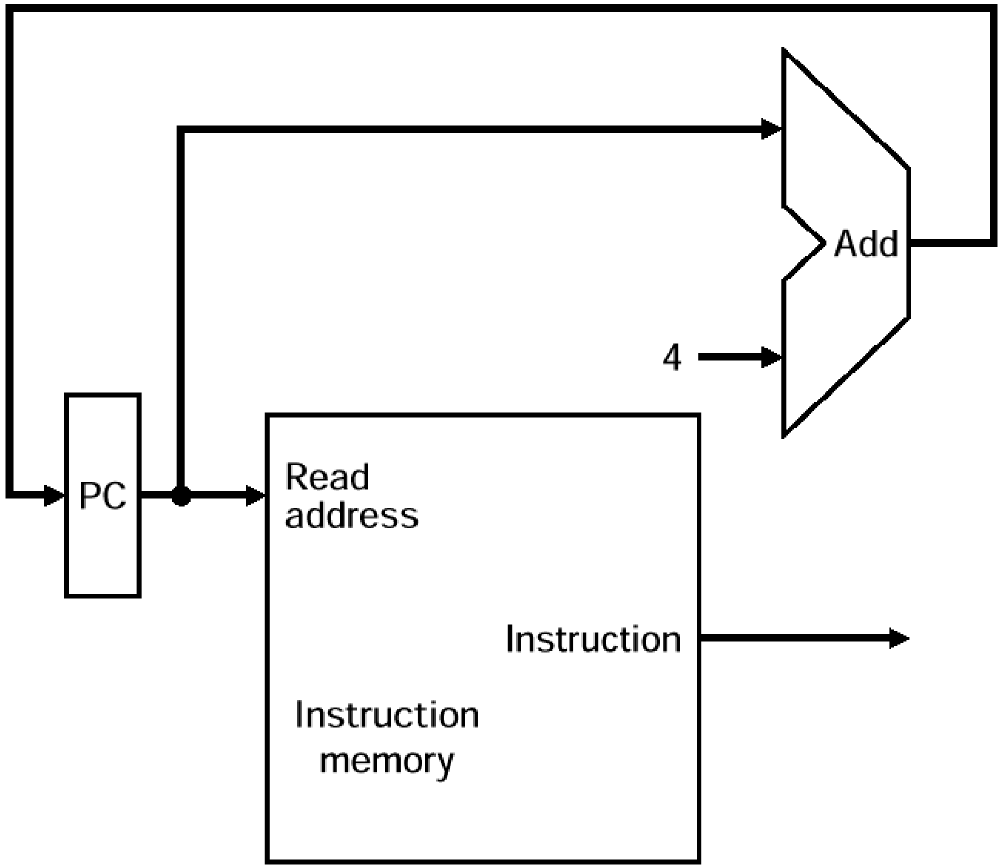


R-format Instruction

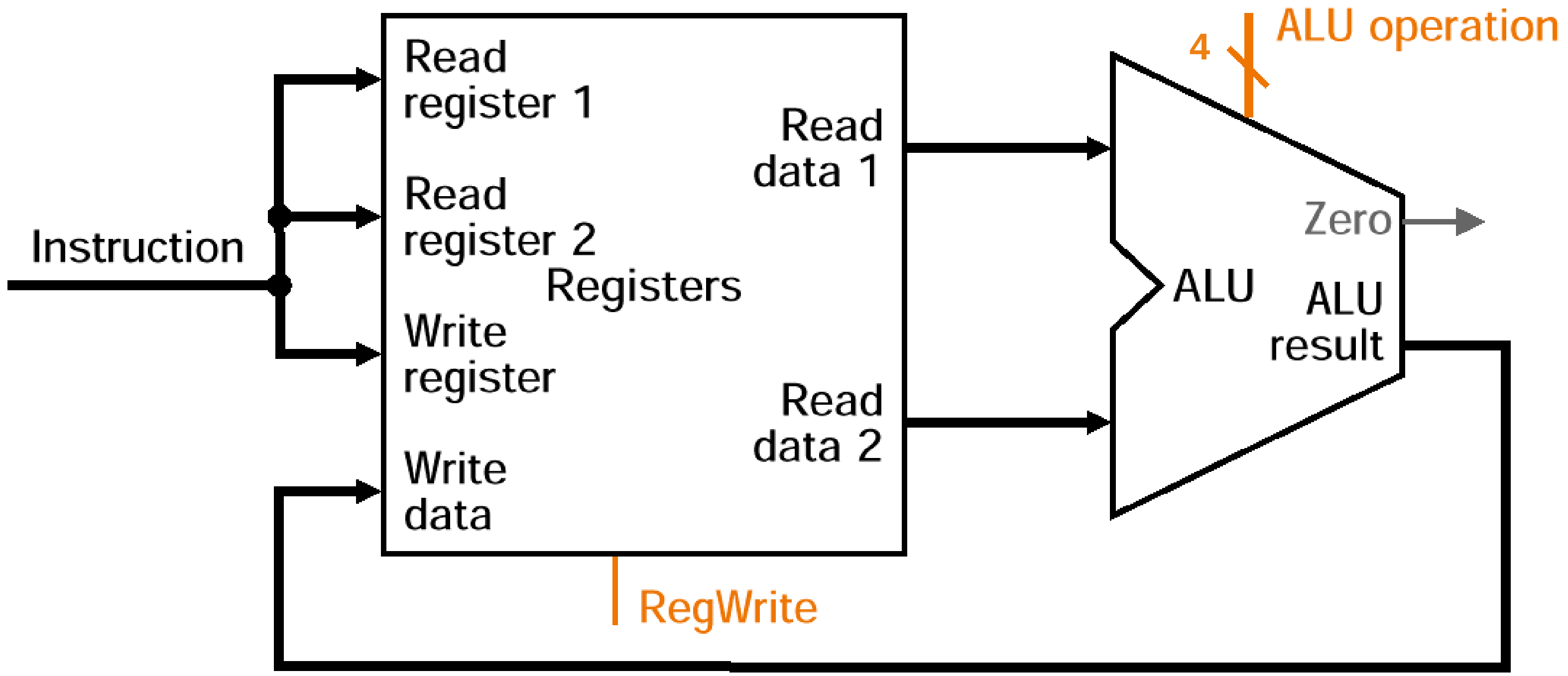


I-format Instruction

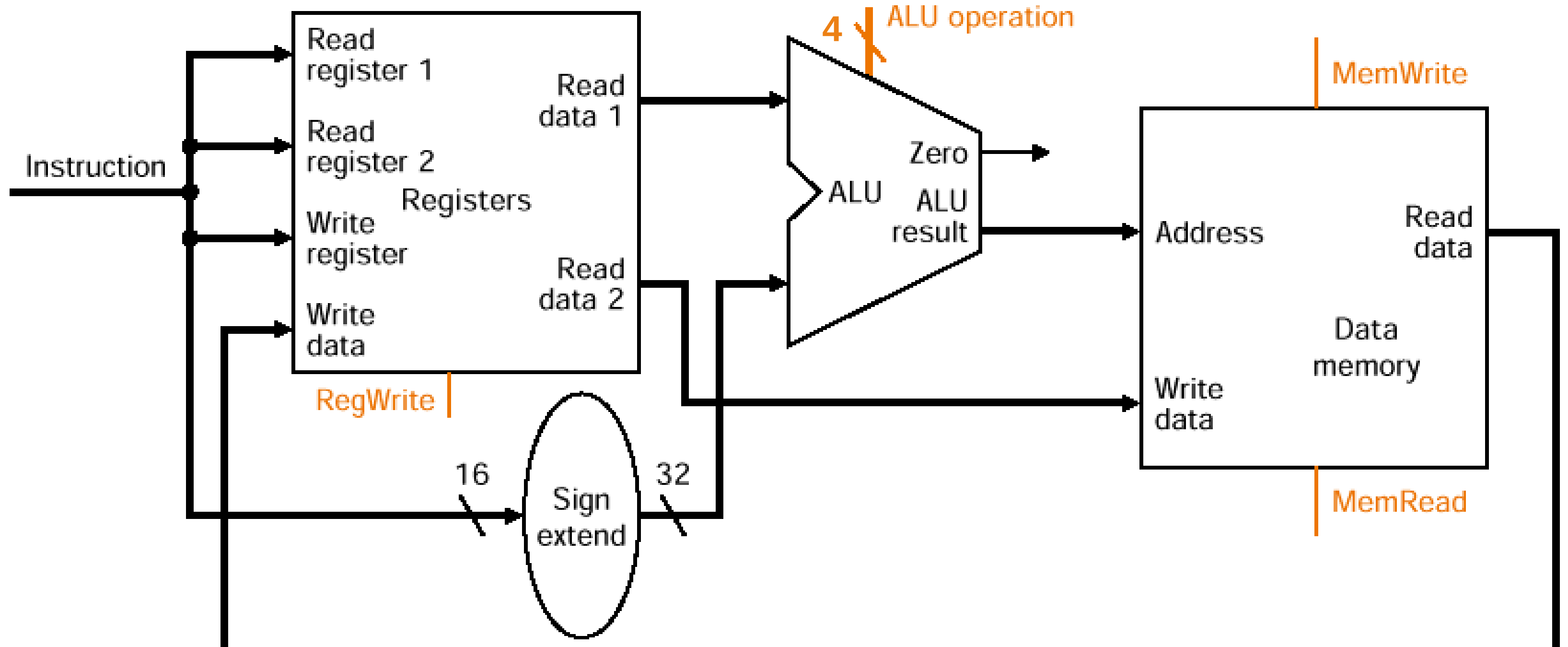
# PC-Related Datapath



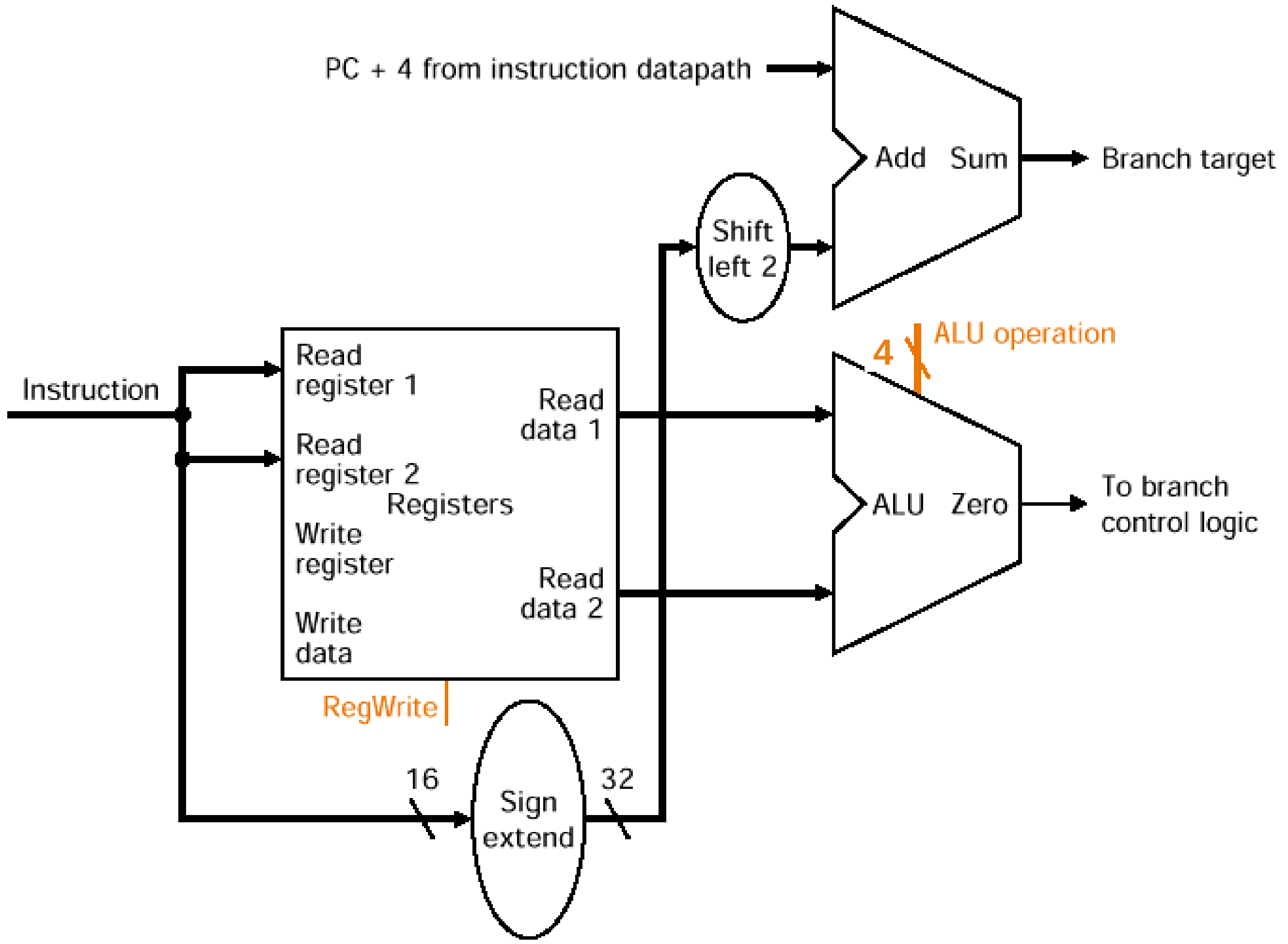
# Datapath for R-format Instructions



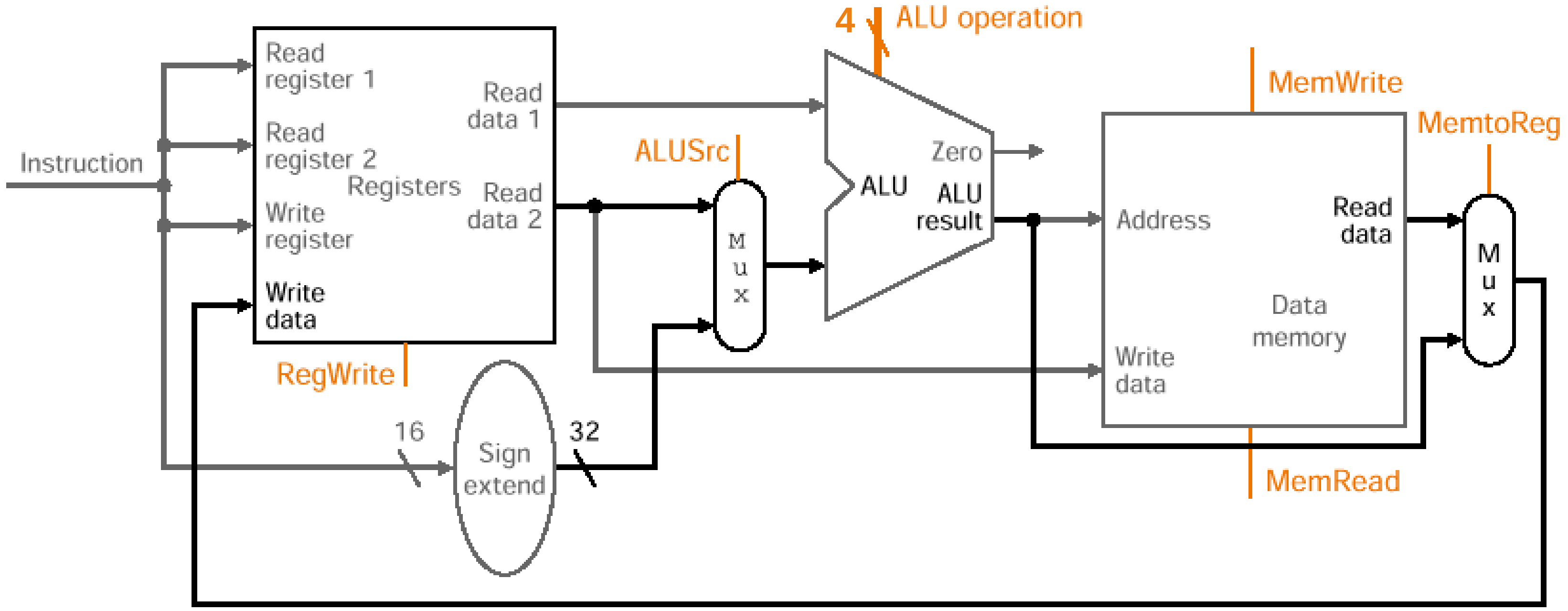
# Datapath for a Load or Store



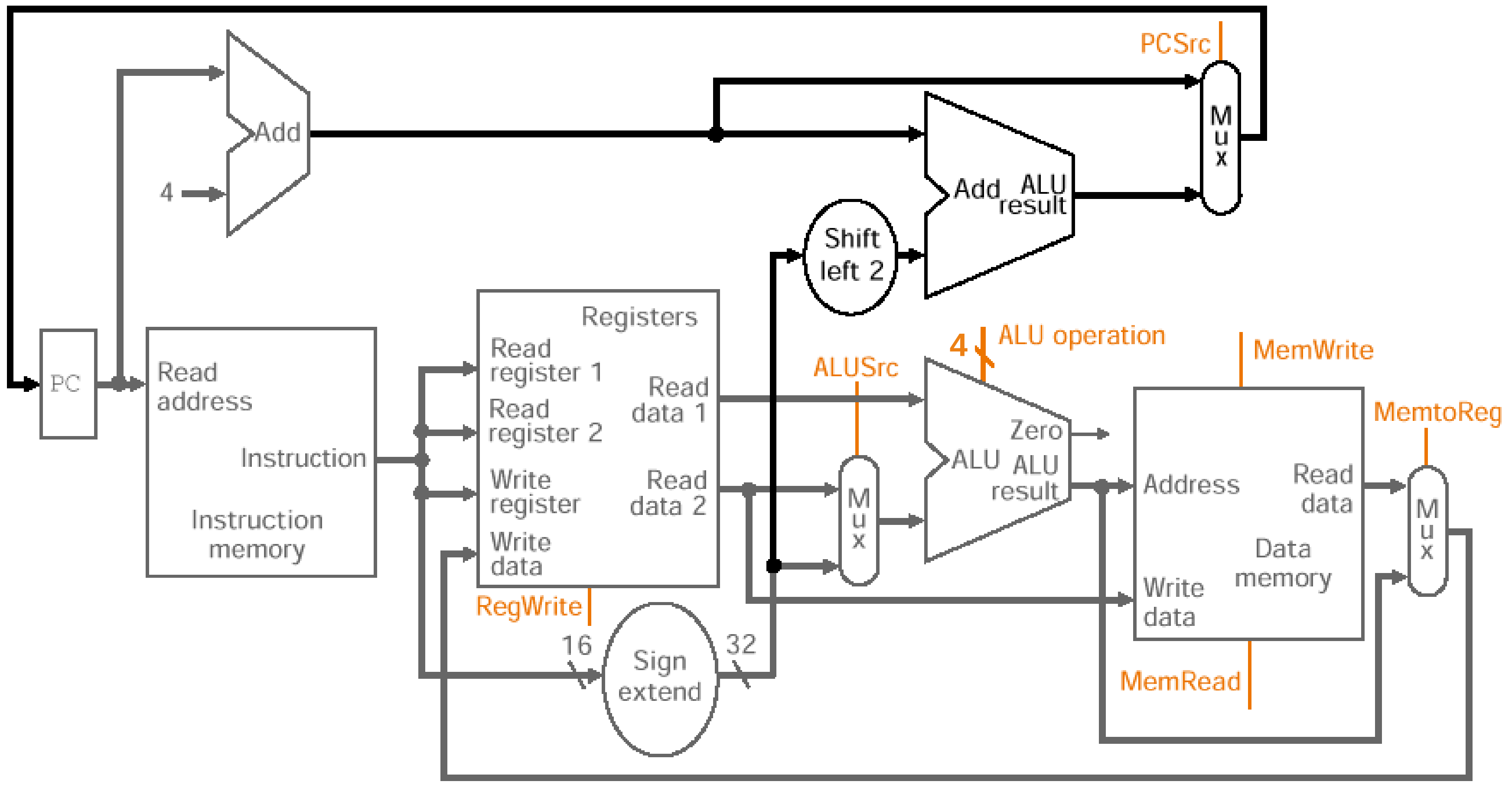
# Datapath for a Branch



# R-type + (Load or Store)



# R-type + (Load or Store) + Branch



PCSrc
Add
4
M u x
Add ALU result
Shift left 2
PC
Read address
Instruction
Instruction memory
Registers
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
RegWrite
16
Sign extend
32
ALUSrc
4
ALU operation
M u x
Zero
ALU
ALU result
MemWrite
Address
Read data
Data memory
Write data
MemRead
MemtoReg
M u x