INTRODUCTION TO COMPUTER ORGANIZATION

CCIT4026

Chapter 7: Single Cycle Process (Part 1)

Outline

- A Review on MIPS instruction set
- CPU Functional Unit
- Single Cycle Processor

MIPS Processor Architecture

- MIPS: Microprocessor without Interlocked Pipeline Stages
- Design a processor to **implement** a **subset** of the core MIPS instruction set:
  1. Arithmetic-logical instructions: `add, sub, and, or, slt`
  2. Memory-reference instructions: `lw, sw`
  3. Branch instructions: `beq, j`
  - This subset does not include all the instructions (e.g. shift, multiply, divide, etc.)

(1) Arithmetic-Logical Instruction

- **`sub reg1, reg2, reg3`**
  - Read register reg2 and reg3
  - Execute the addition to get reg2 - reg3
  - Write back the result into reg1

- **`add, and, or, slt`** are similar, only the execution steps are different

(2) Memory-reference Instruction

- `lw reg1, const_num(reg2)`
  - Read register reg2
  - Execute the addition to get reg2 + const
  - Access memory to get data based on the address
  - Write back the data into reg1

- `sw reg1, const_num(reg2)`
  - Read register reg1 and reg2
  - Execute the addition to get reg2 + const
  - Access memory to store data in reg1 based on the address

(3) Branch Instruction

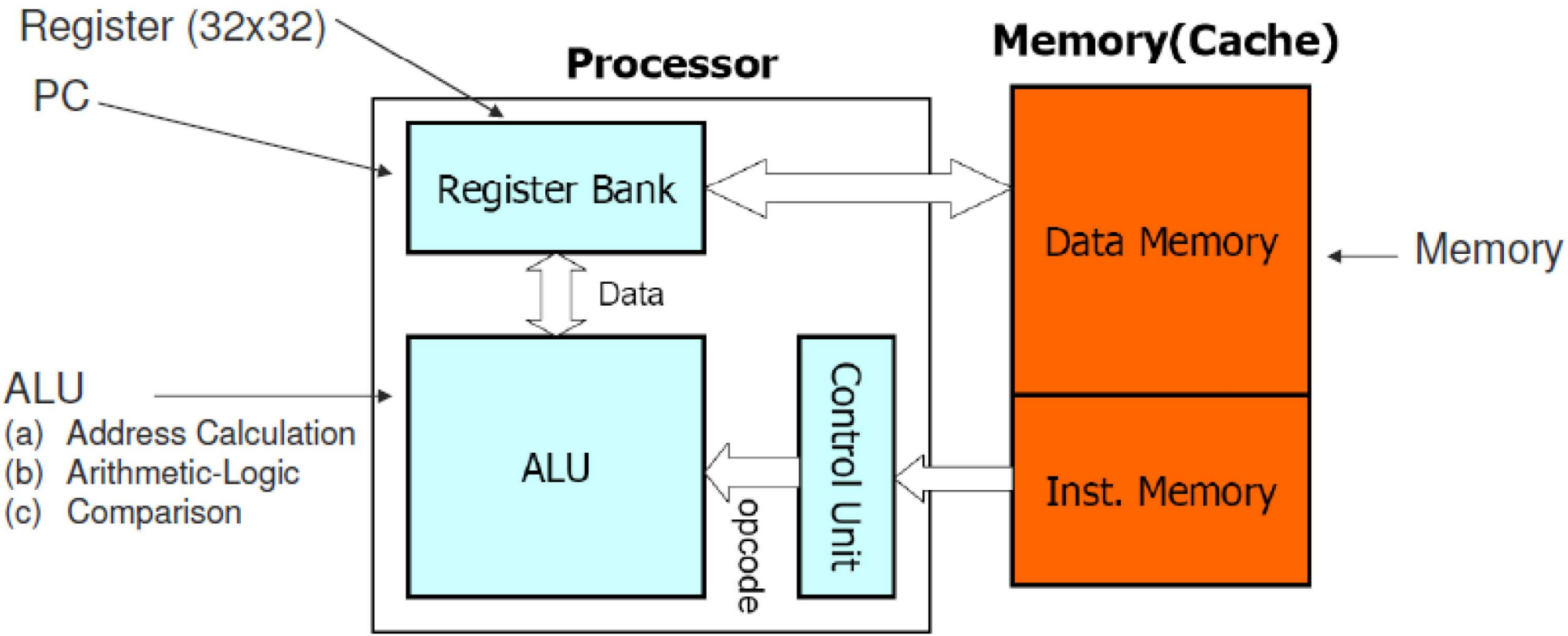
- **`beq reg1, reg2, const_num`**
  - Read register reg1 and reg2
  - Execute the comparison of reg1 and reg2
  - Update PC = PC + 4 + (const_num x 4), if reg1 is equal to reg2

- **`j const_num`**
  - Update PC = (PC & 0xf0000000) + const_num x 4
    - PC & 0xf0000000 keeps the upper 4 bits of PC untouched
    - const_num is 26 bits (J-format)

CPU Basic Functional components

- Register file for general-purpose registers
- Memory for instructions and data
- ALU for addition, subtraction, AND, OR, and comparison
- A functional unit for Program Counter
- Other functional units to calculate PC + 4, PC + 4 + (const_num x 4), and (PC & 0xf0000000) + const_num x 4
- Control unit to control other functional units
- Other miscellaneous functional units

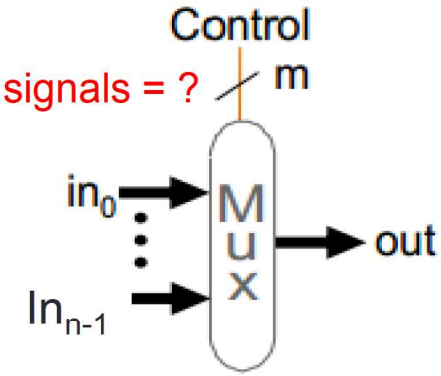
Datapath Components

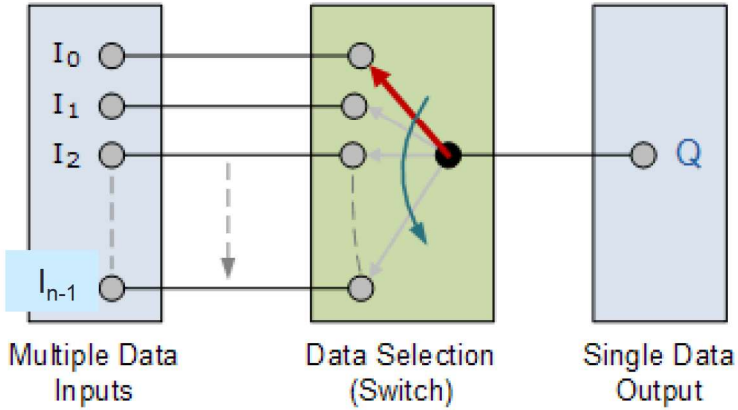


Multiplexer

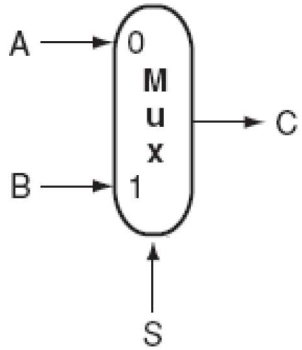
- Multiplexer (MUX) is a combinational logic
  - design to switch one of several input lines through a single common output line by control signal

How many controls signals = ?

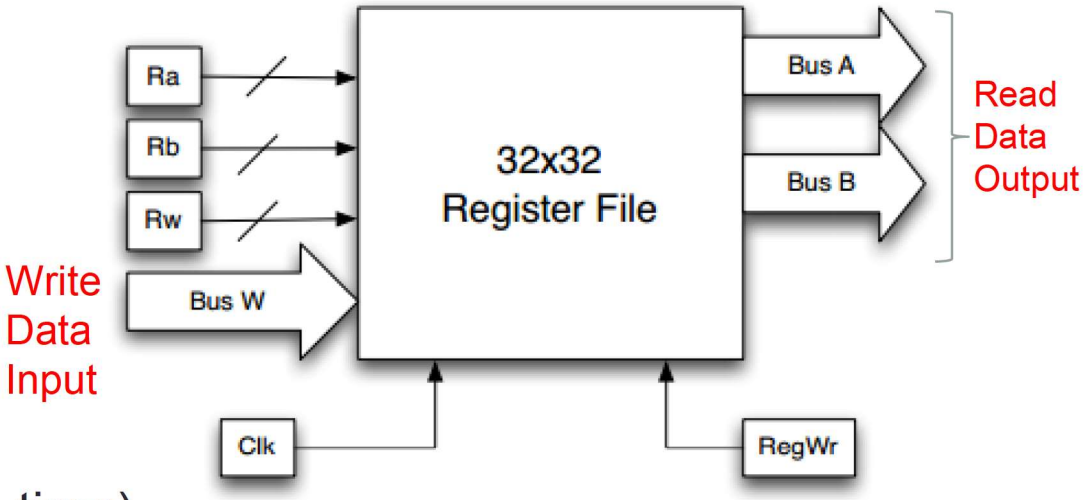




- For example: 2-to-1 mux,
  - Two data inputs: A and B
  - One control input: S
  - One data output: C
  - Boolean function: $C = A \cdot \bar{S} + B \cdot S$
    - i.e. if S is 0, C = A; if S is 1, C = B



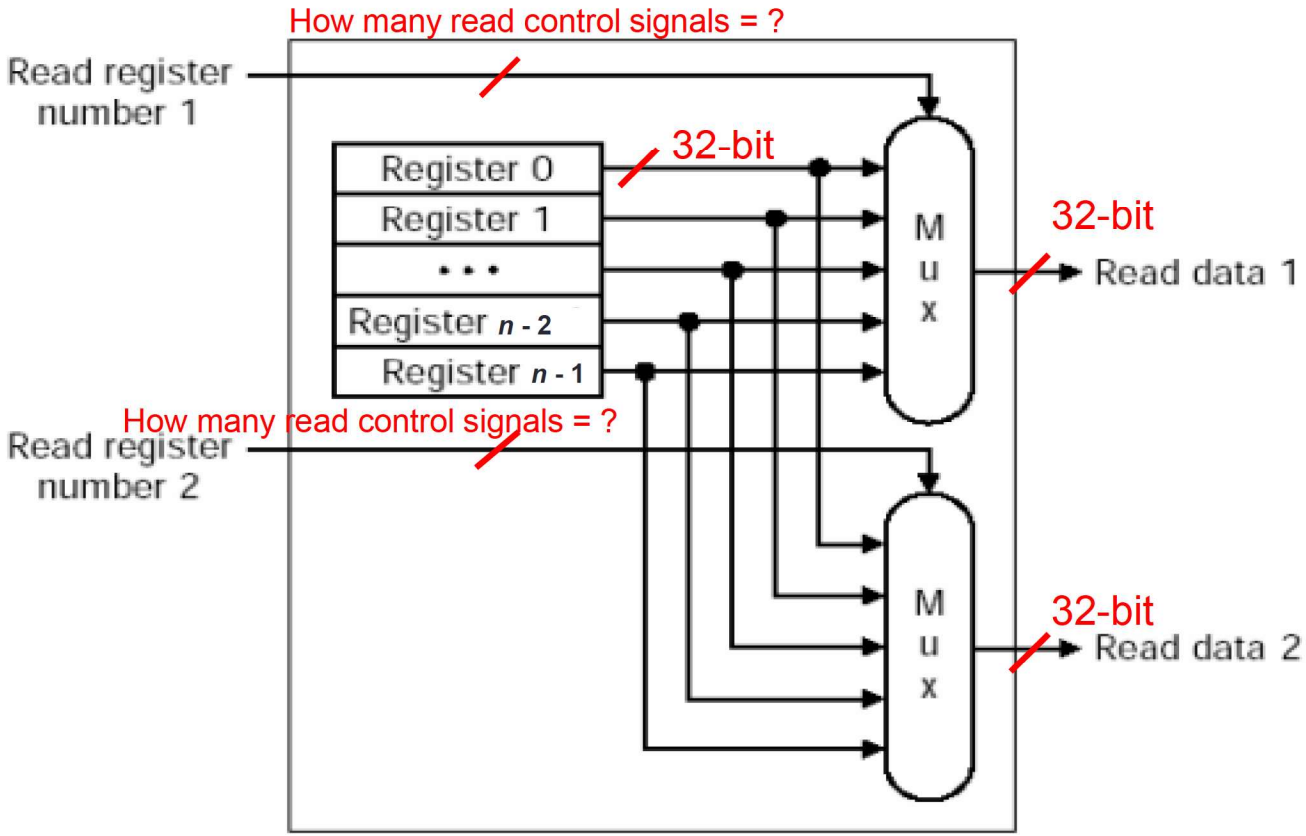
Register File



- 32x32 (32 32-bit) Register File
  a) Buses (32-bit wide)
     - Bus A, Bus B are read ports (allow to read 2 registers at a time)
     - Bus W is write port
  b) Register Address
     - Ra, Rb and Rw are the addresses of the data on Bus A, Bus B and Bus W respectively.
  c) RegWr
     - Set to 1 to enable write data on Bus W
  d) Clk
     - Time synchronization for write operation. For read, it is asynchronous.

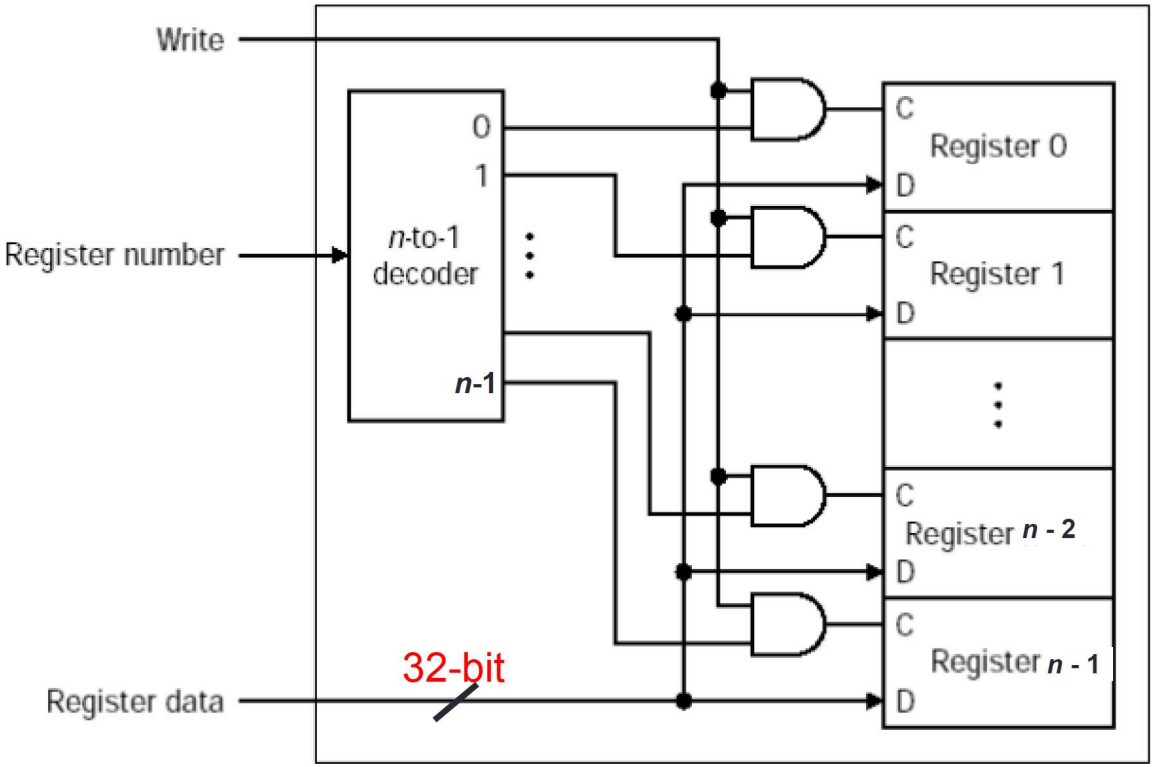
Register File

- To implement two read ports for a register file with *n* registers, we use two **n-to-1 multiplexers**.
- Each of which is 32 bits wide.



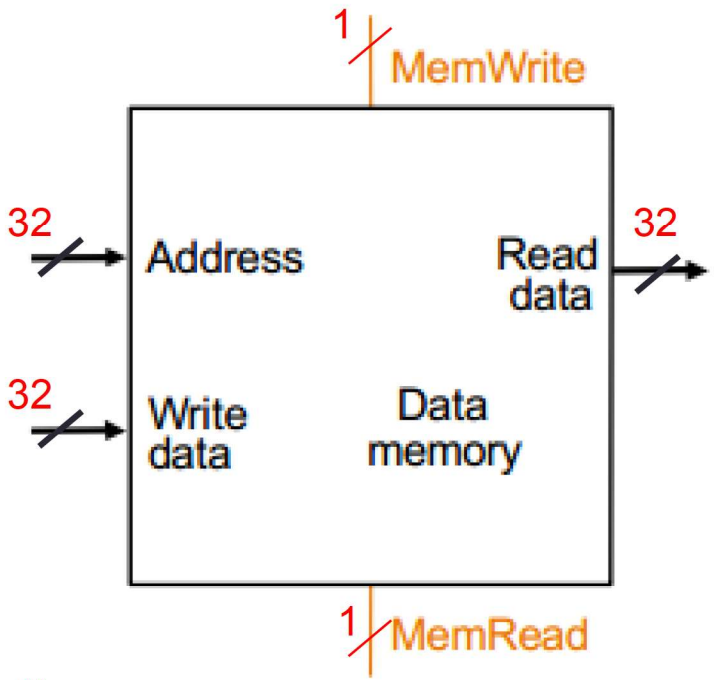
Register File

- To implement the write port for a register file with *n* registers, we use an **n-to-1 decoder** to generate a signal that can be used to determine which register to write.



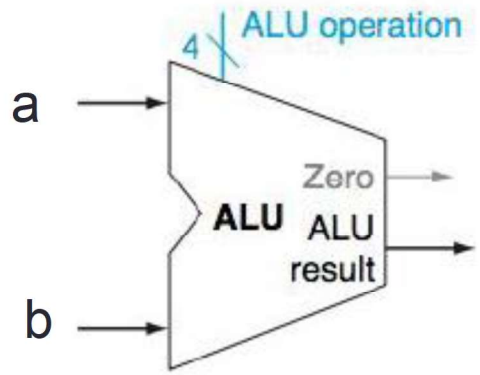
Memory

- Data memory (Load and Store)
  a) Buses (32-bit wide)
     - Write data
     - Read data
  b) 32-bit Address
  c) Memory control
     - MemWrite
     - MemRead
     - cannot read and write at the same time



Arithmetic Logic Unit (ALU)

- ALU performs binary arithmetic and logical operations
-

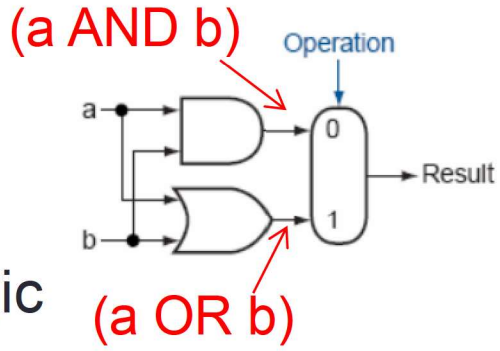


ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

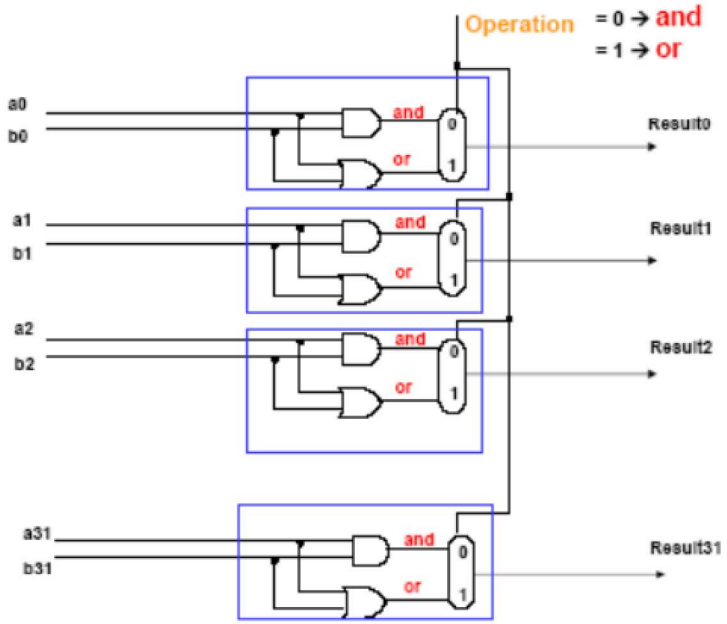
- Inputs:
  - a and b (2 x 32-bit)
  - Control (4-bit) for selecting operation (ALUOp)
- Outputs:
  - Result (32-bit)
  - Zero

Arithmetic Logic Unit (ALU)

- 1-bit ALU design
  - First, understand functionalities
  - Second, define the ports of a function unit
  - Third, design the internal logic as combinational logic

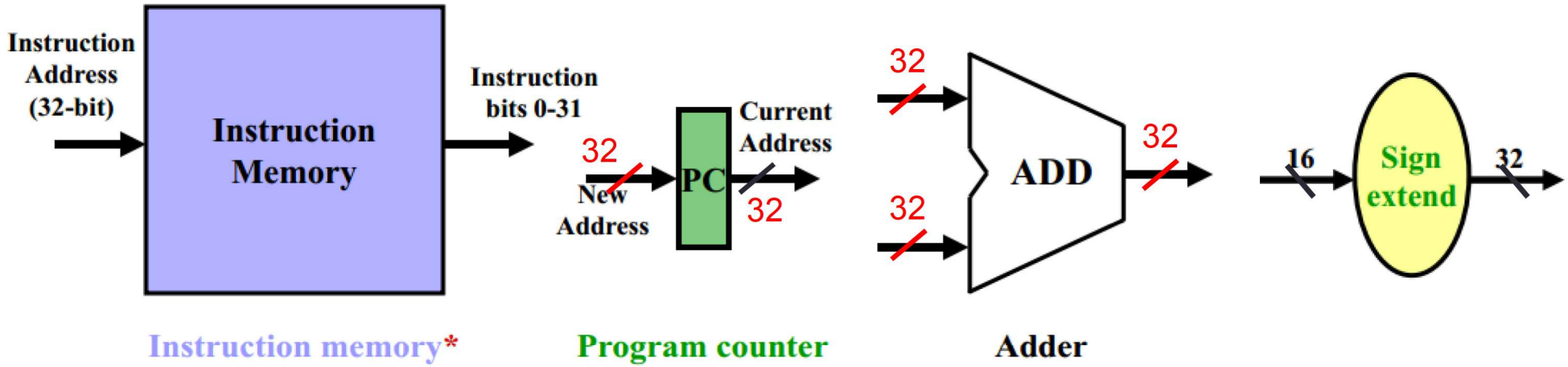


- 32-bit ALU
  - Can execute and, or, nor, add, sub, slt
  - First build a 1 bit ALU, and use 32 of them

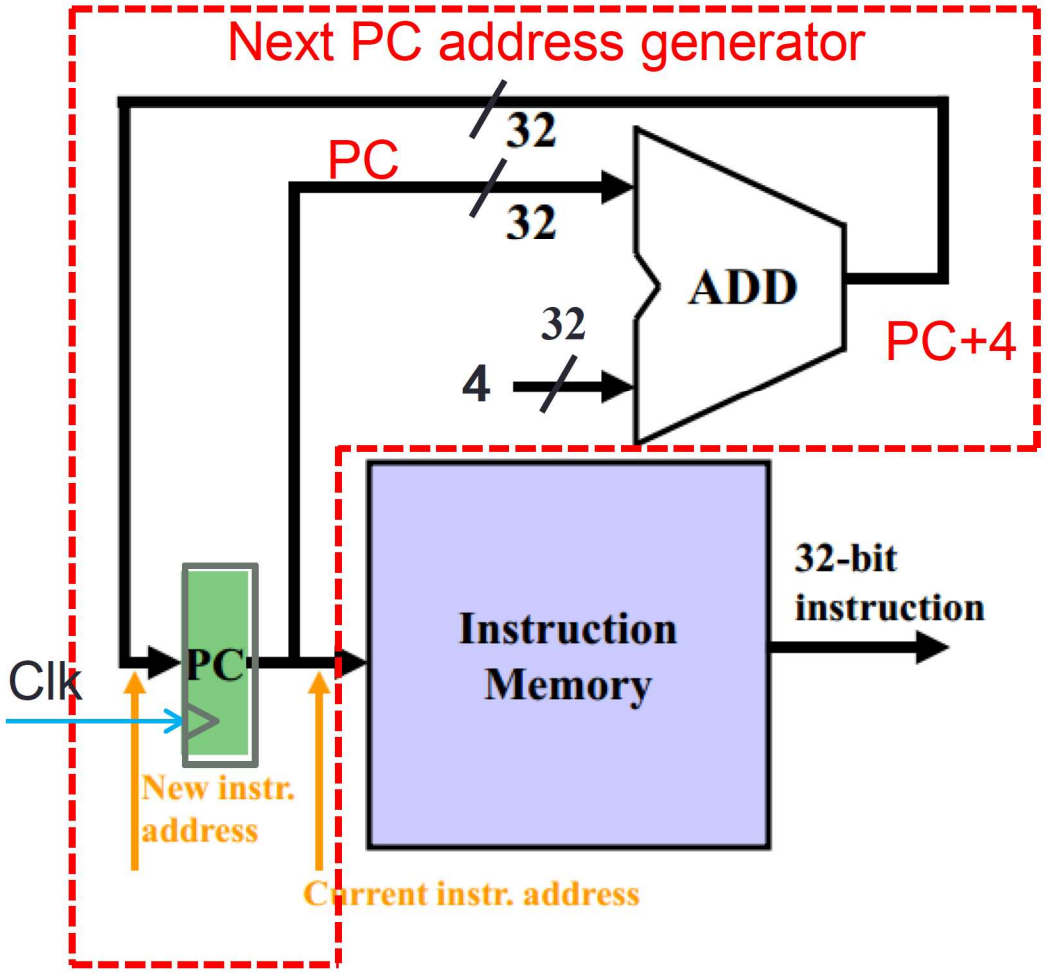


Single Cycle Processor

- More Building Blocks
  - **Instruction memory**: A memory unit that stores the instructions of a program supplies an instruction given its address
  - **Program counter**: A register storing the address of the instruction being executed
  - **Adder**: A unit that increments PC to form the address of next instruction
  - **Sign extend**: Convert 16-bit data to 32-bit data by appending the sign bit 16 times at the most significant side.



Single Cycle Processor

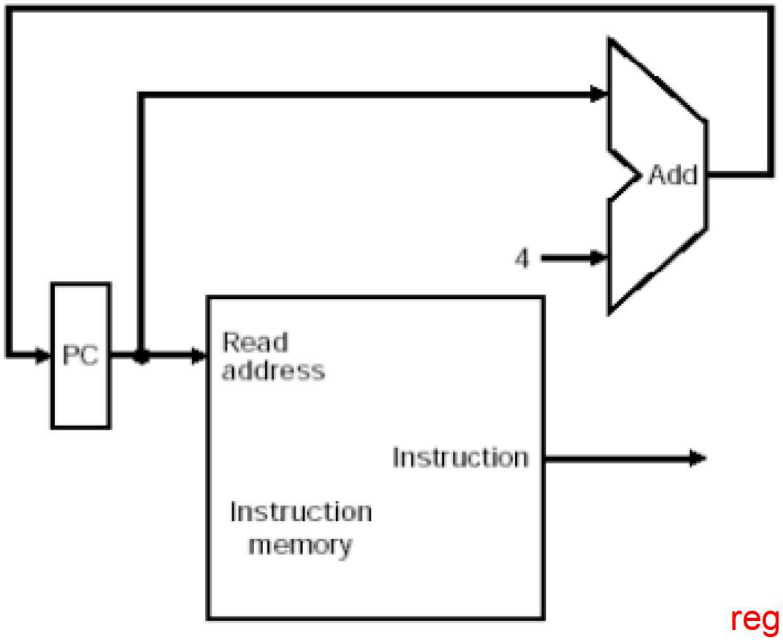


- The program counter (PC) is a register that addresses the next instruction in memory.
- New addr = current addr + 4

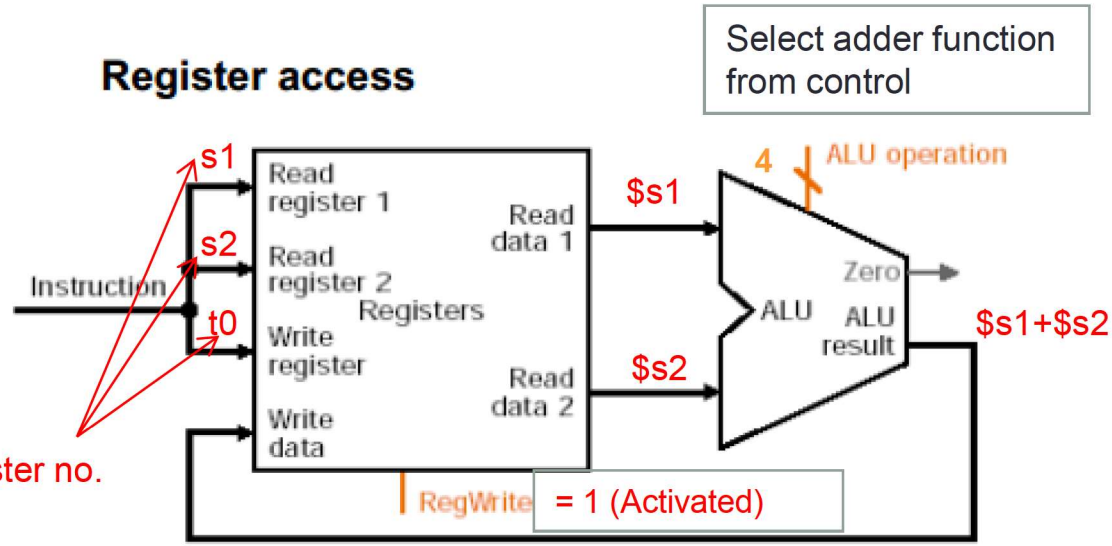
Example #1

(R-type) add $t0 $s1 $s2

opcode $s1 $s2 $t0



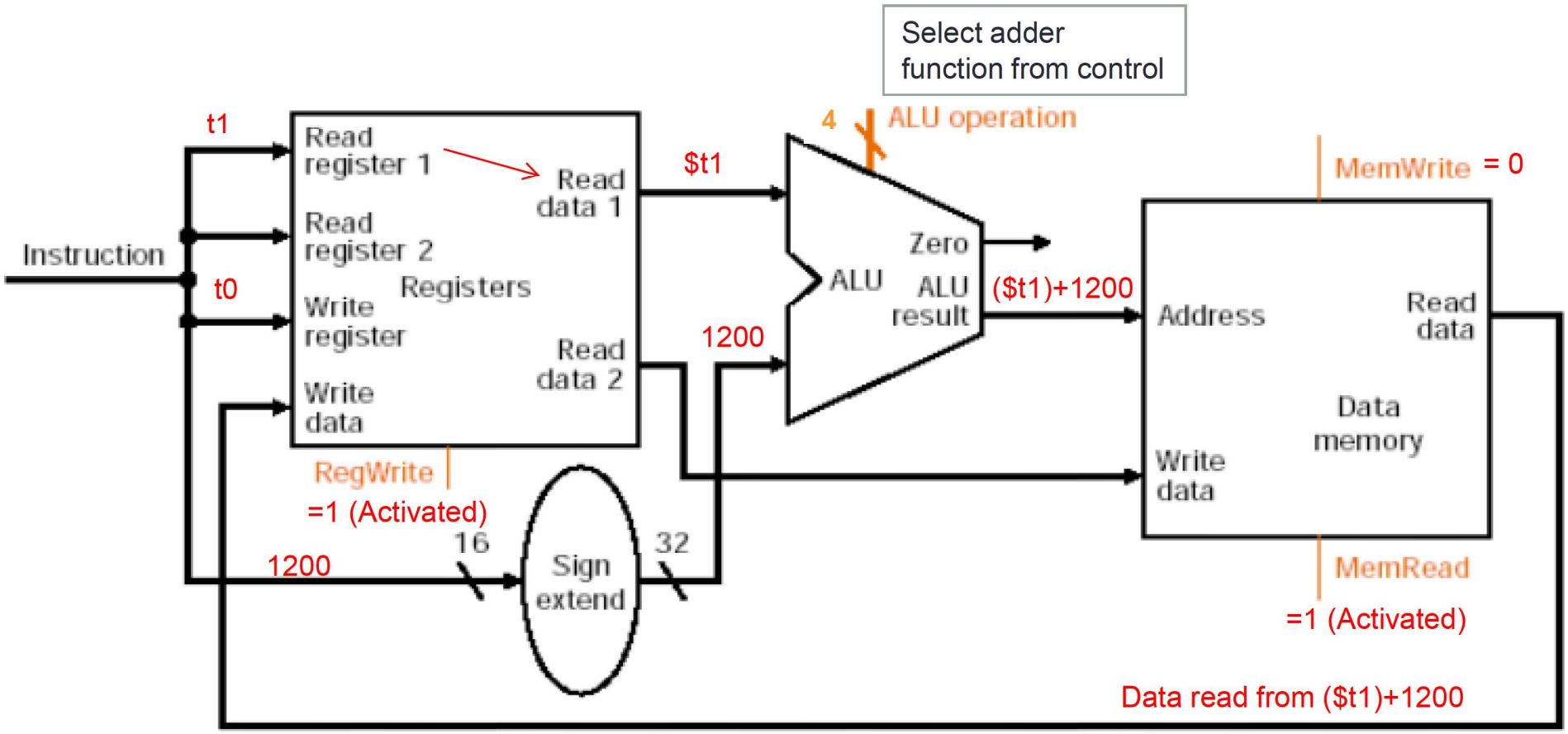
Instruction fetch and PC update



R-type operation: ALU

Example #2

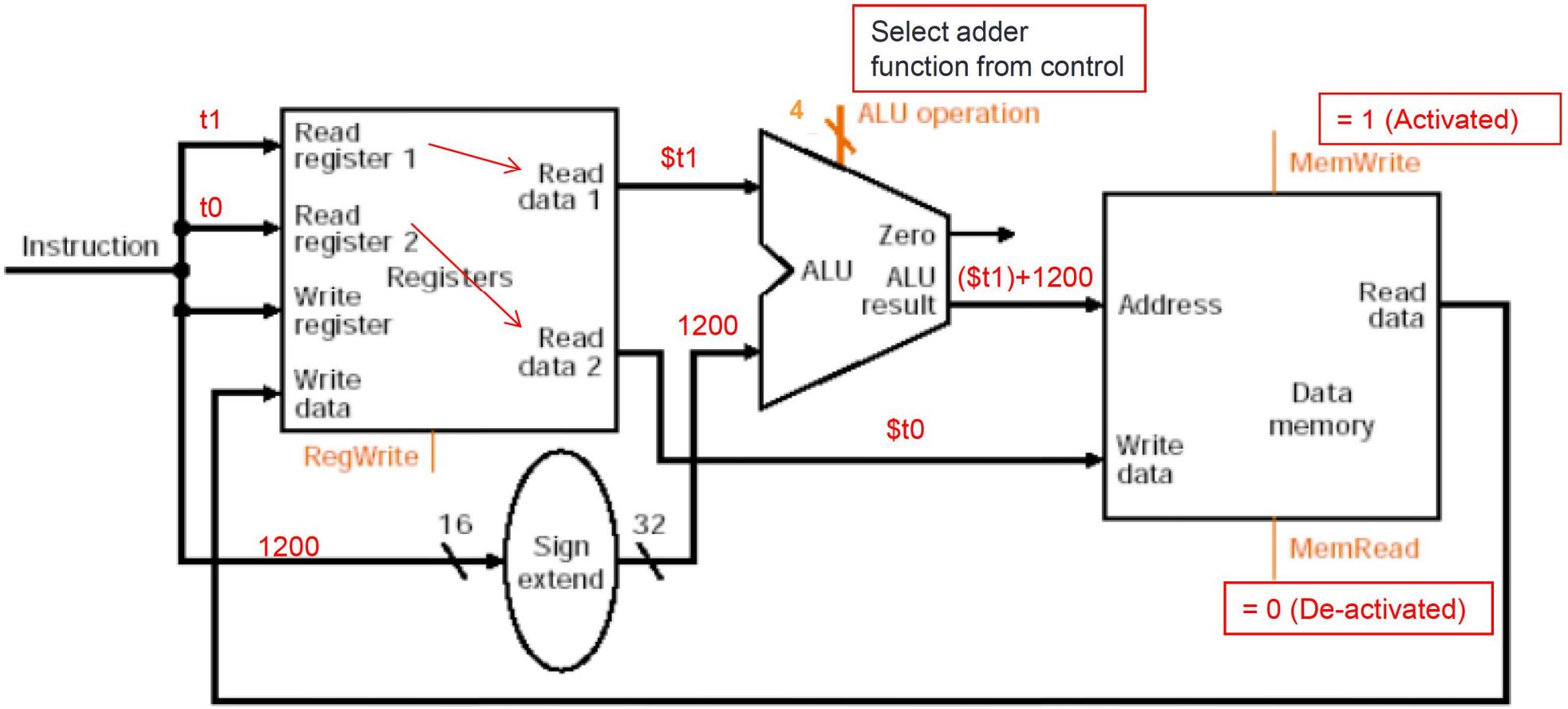
- ***(I-type) lw $t0, 1200($t1)***



* In load instruction, the ALU performs the address calculation.

Example #3

- ***(I-type) sw $t0, 1200($t1)***

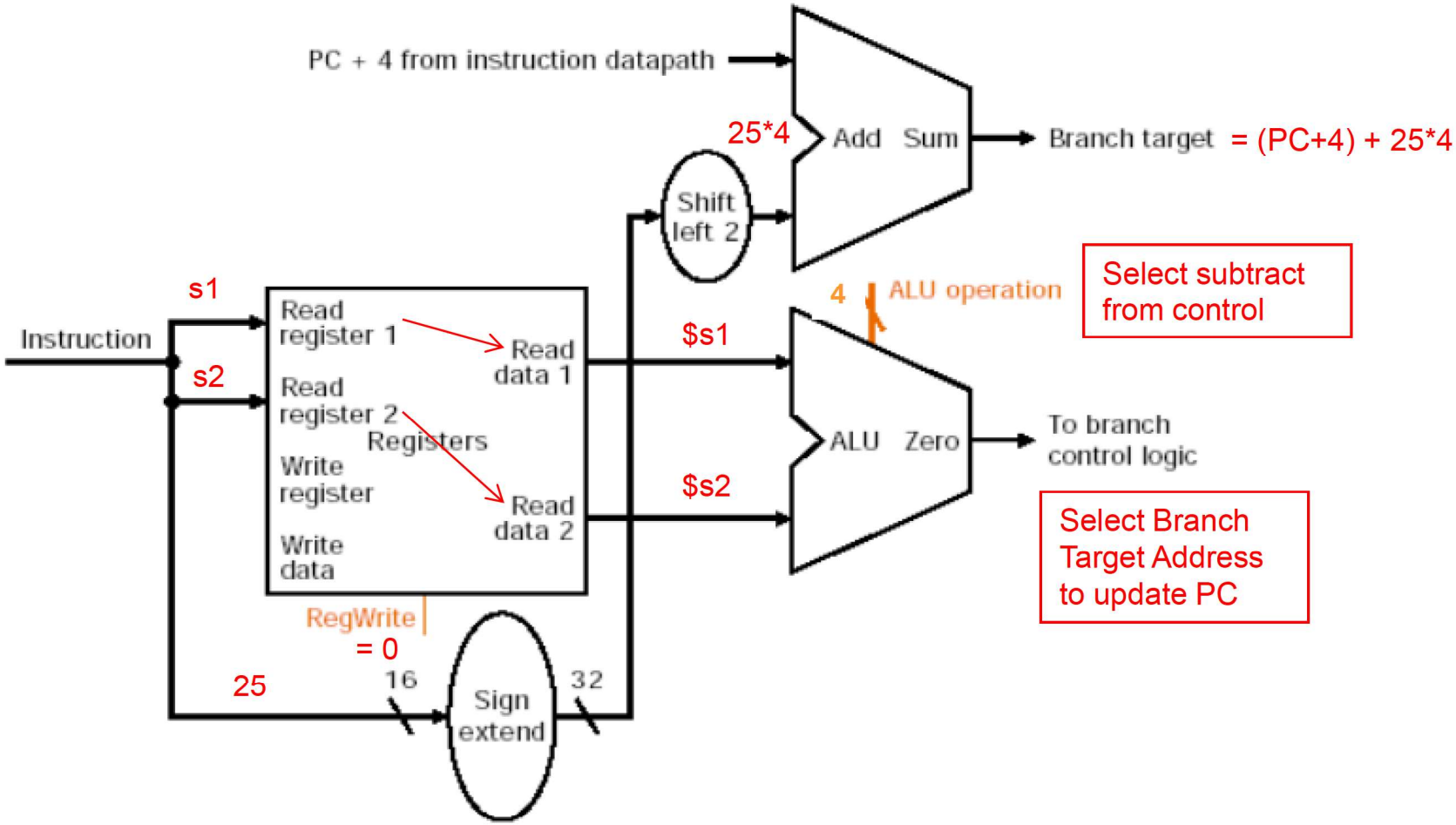


Example #4

To compare for EQUAL, the idea is to perform a subtraction and see if the result is ZERO

- ***(I-type) beq $s1, $s2, 25***

opcode $s1 $s2 25

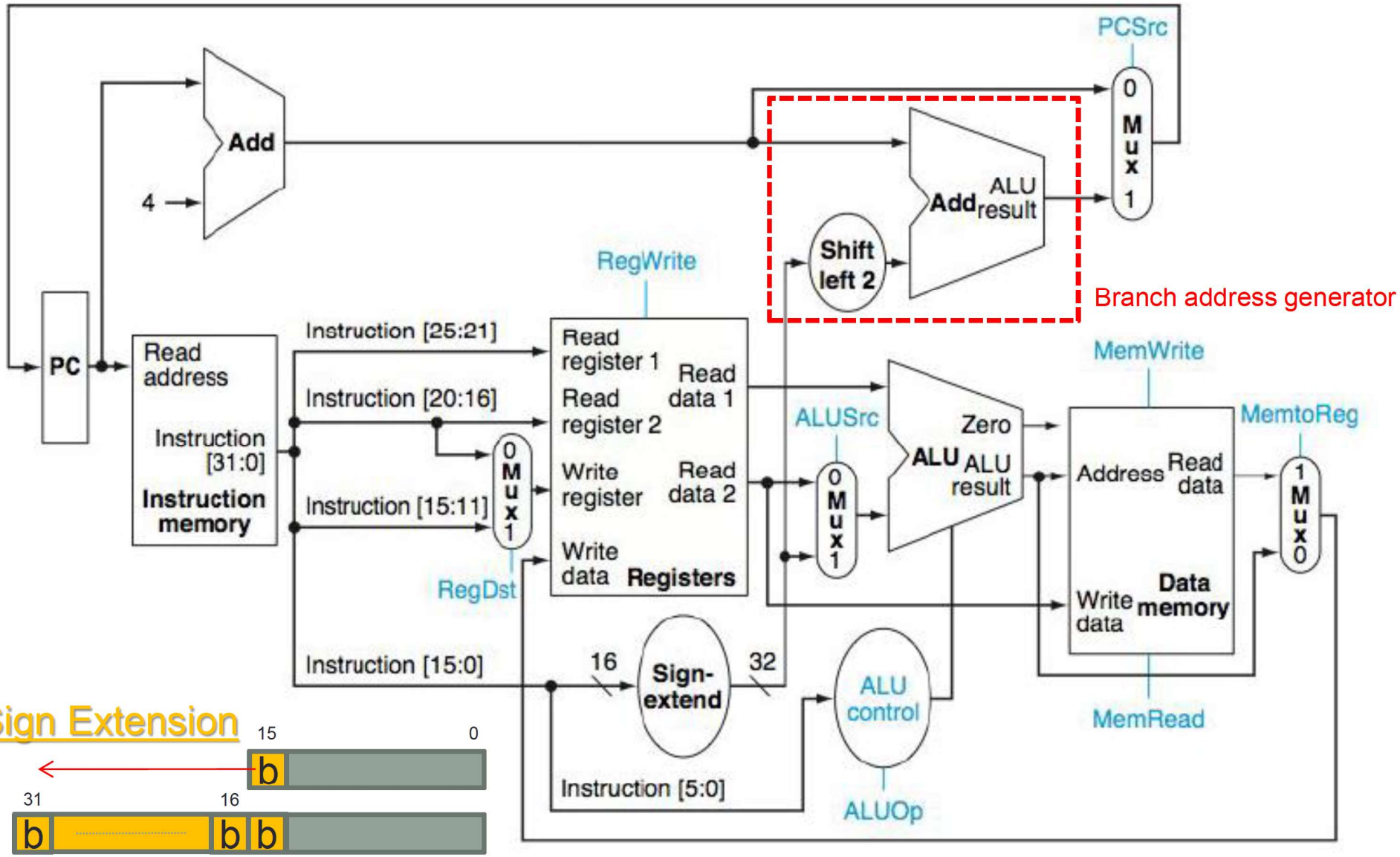
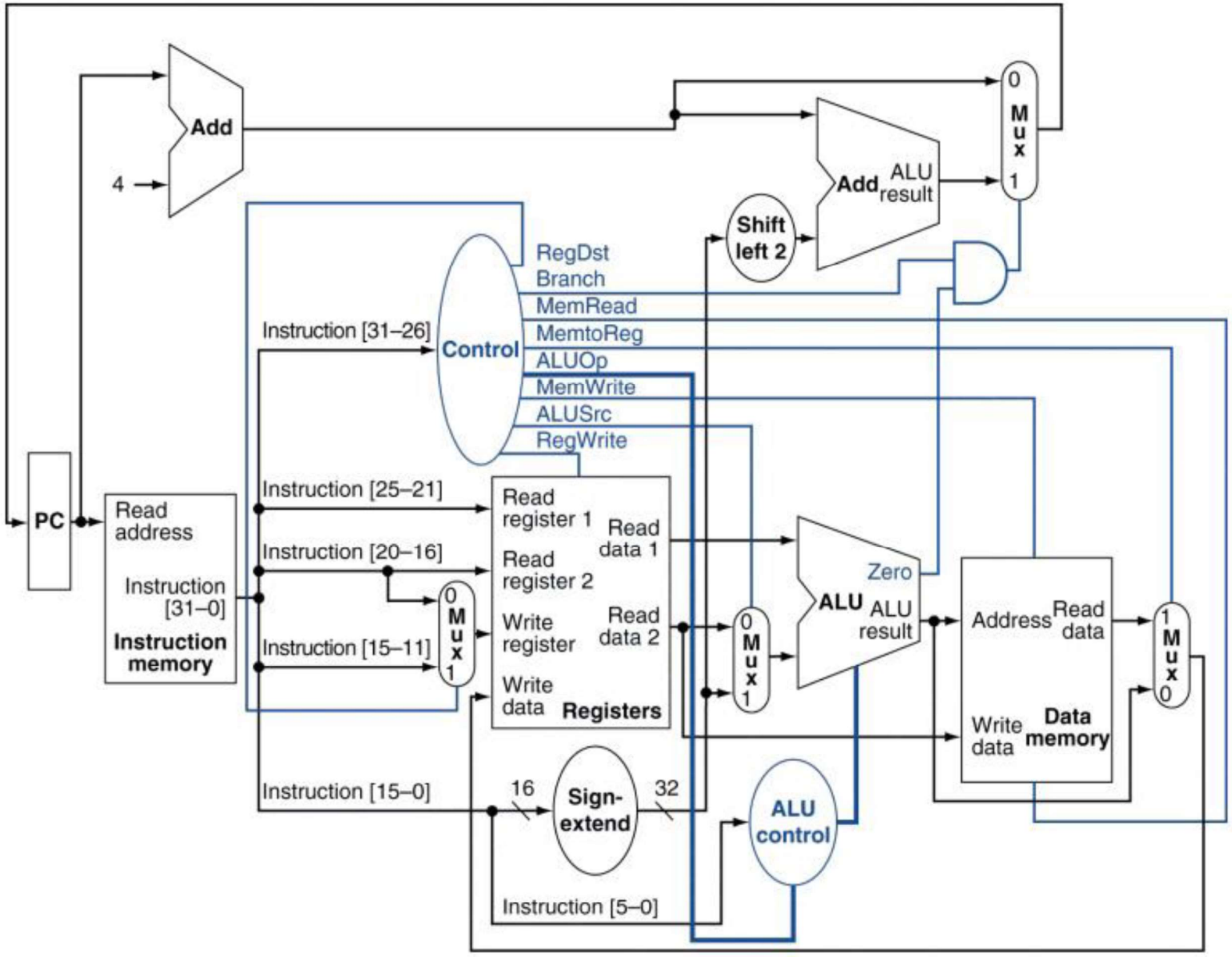


Single Cycle Processor

- Now, we need to combine them into a **single datapath**, by devising ways to share some of the resources (e.g., ALU) among the different instructions instead of duplicating them.

- Control and select the operation of the datapath by generating appropriate control signals (e.g. read/write signals for memory/registers, selection input for muxes, ALU control inputs)

Single Cycle Processor



Single Cycle Processor