
EECE 321: Computer Organization

Mohammad M. Mansour
Dept. of Electrical and Compute Engineering
American University of Beirut

Lecture 18: MIPS Single-Cycle Processor
Implementation

Announcement

- Makeup session: Thursday from 5:00-6:30pm
 - Room: TBA by email

MIPS Processor Implementation

- Performance of a processor is determined by **inst. count**, **cc time**, and **CPI**
 - Compiler and ISA determine instruction count.
 - Implementation (micro-architecture) determines **cc time** and **CPI**.
- We'll construct the datapath and control unit for three different implementations of the MIPS ISA:
 - **Single-cycle implementation**: All operations take the same amount of time a single cycle.
 - **Pipelined implementation**: Lets a processor overlap the execution of several instructions, potentially leading to big performance gains.
- The implementation includes a subset of the core MIPS instruction set:

Arithmetic: **slt**, **or**, **and**, **sub**, **add**

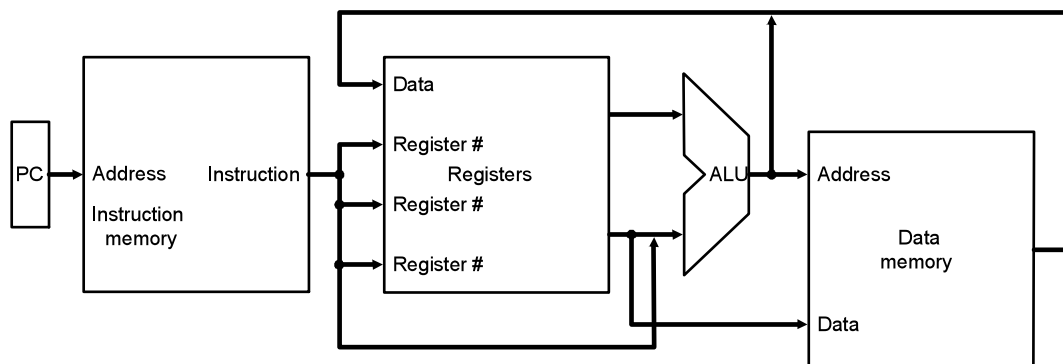
Data Transfer: **lw**, **sw**

Control: **beq**, **j**

- First we'll build a **single-cycle** implementation of this reduced instruction set piece by piece, and then combine pieces together to form datapath + control.
 - All instructions will execute in the same amount of time; this will determine the clock cycle time for the performance equations.
 - We'll explain the datapath first, and then design the control unit.

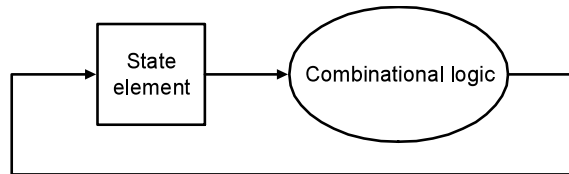
Implementation Overview

- Memory, ALU, and branch instructions have their first two steps identical:
 - Send PC to memory to fetch instruction
 - Read one (*lw*) or two registers (most others)
- There are even more similarities: All instructions use ALU after reading registers
 - Memory instructions use ALU for address calculation
 - ALU instructions use ALU for operation executions
 - Branch instructions use ALU for comparison
- After using ALU, actions required depend on the instruction class:
 - Memory instructions access memory to read/write data
 - ALU instructions write back results to a register
 - Branch instructions may need to change the next instruction address
- A high-level view of a MIPS implementation:

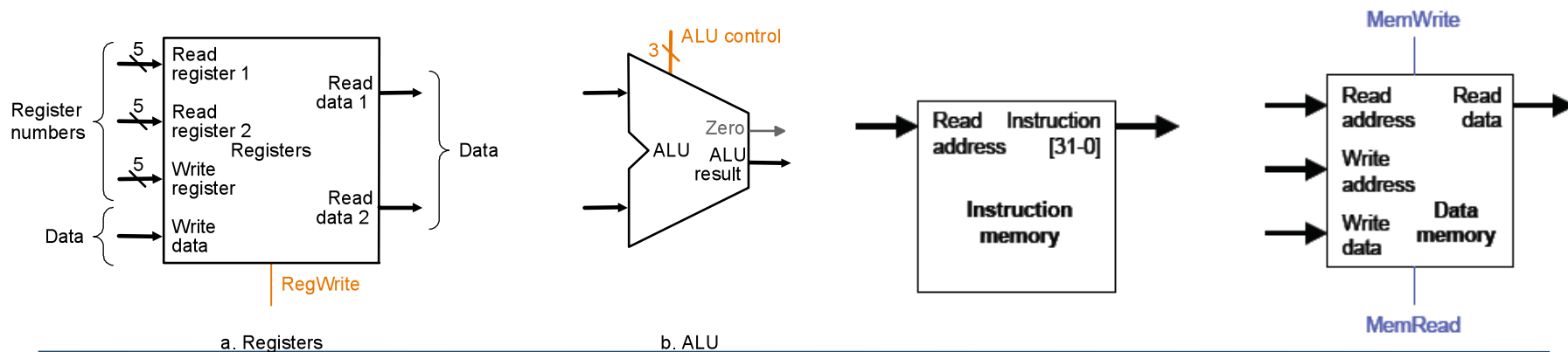


Conventions

- A computer is just a big complex state machine.
 - Registers, memory, hard disks and other storage form the state.
 - The processor keeps reading/updating the state, according to the instructions in some program.



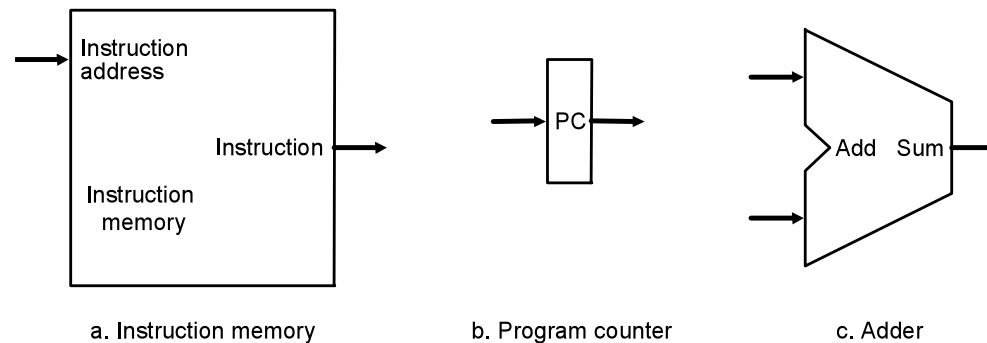
- Combinational elements to be used: ALU
- Sequential elements:
 - Register file
 - Instruction memory
 - Data memory
 - Note: Separating instructions from data by using instruction memory and data memory is referred to as Harvard Architecture.



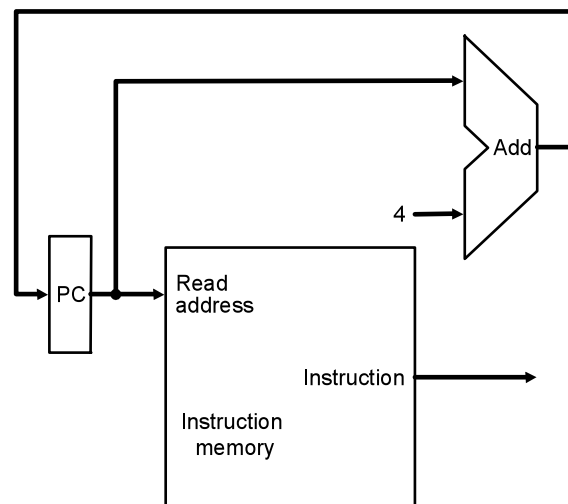
Building the Datapath: Instruction Fetch

- The processor is always in an infinite loop, fetching instructions from memory and executing them.
 - The program counter or PC register holds the address of the current instruction.
 - PC in MIPS should be incremented by four to read the next instruction in sequence.

Datapath elements
needed to fetch an
instruction

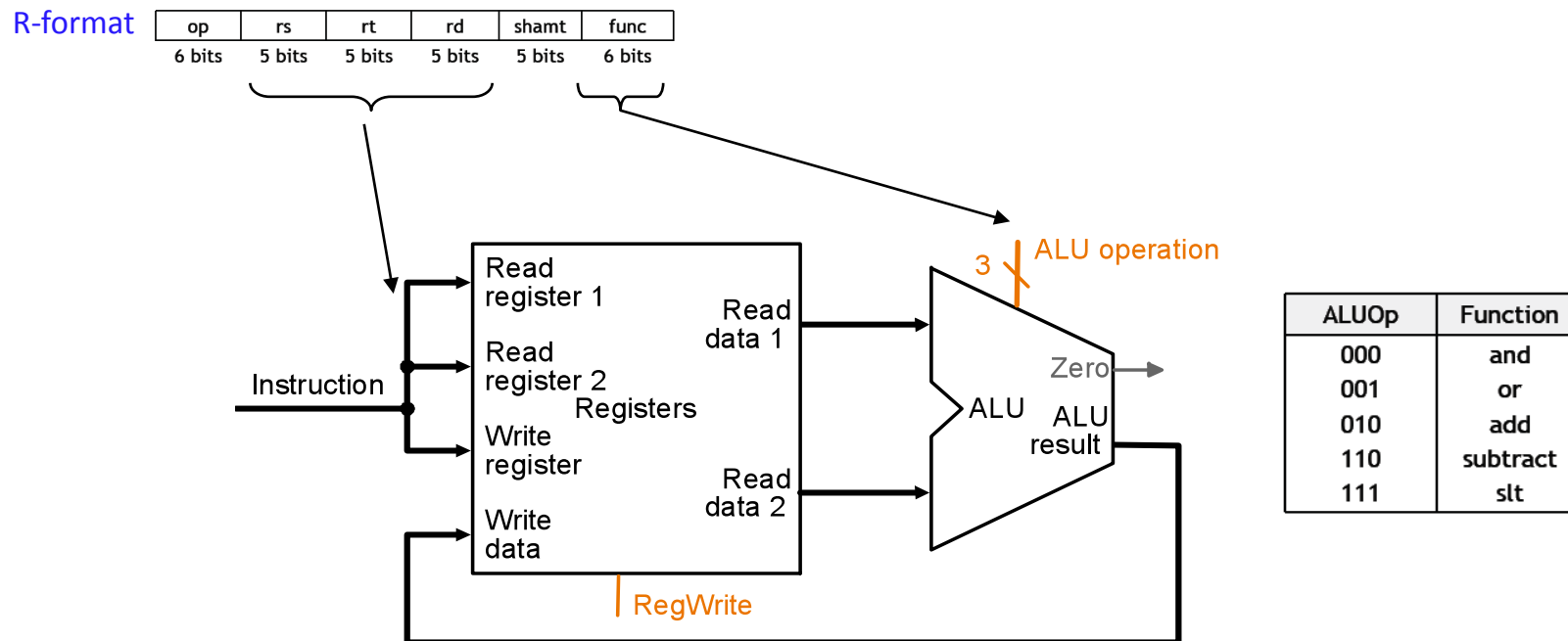


Datapath portion for
instruction fetch



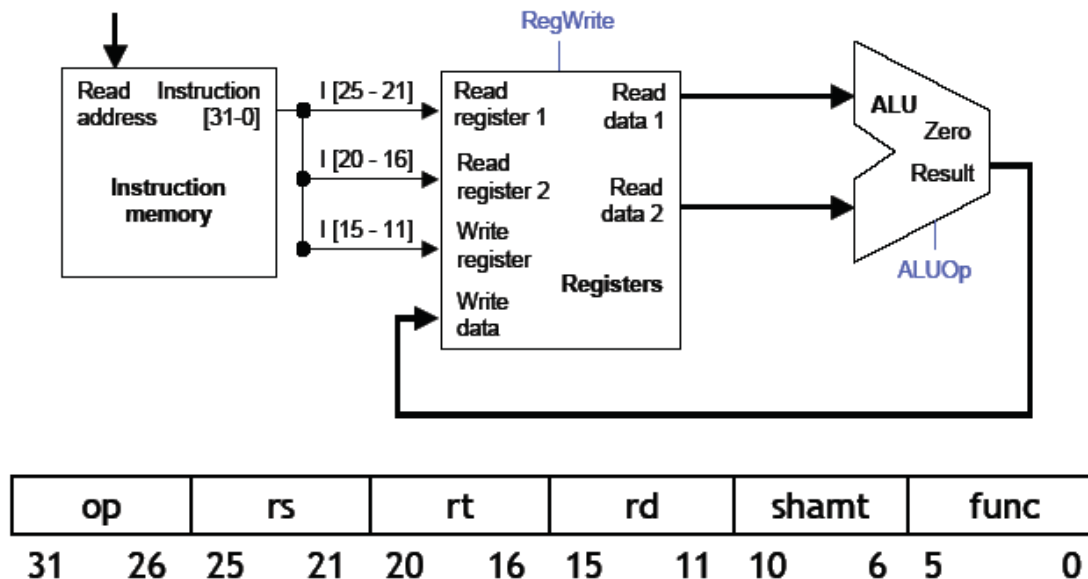
R-Format Instructions Datapath

- R-format instructions read 2 registers, perform an ALU operation, and write back result into a register.
 - Need a register file and an ALU
 - Register file outputs the contents of whatever register numbers are on the address inputs, but performs writes only when 'RegWrite' is enabled
 - Each register specifier is 5 bits long.
 - Can read from two registers at a time.
 - New values written into register file are available for next clock cycle.



Executing an R-Type Instruction

- Read an instruction from the instruction memory.
- The source registers, specified by instruction fields rs and rt, should be read from the register file.
- The ALU performs the desired operation.
- Its result is stored in the destination register, which is specified by field rd of the instruction word.

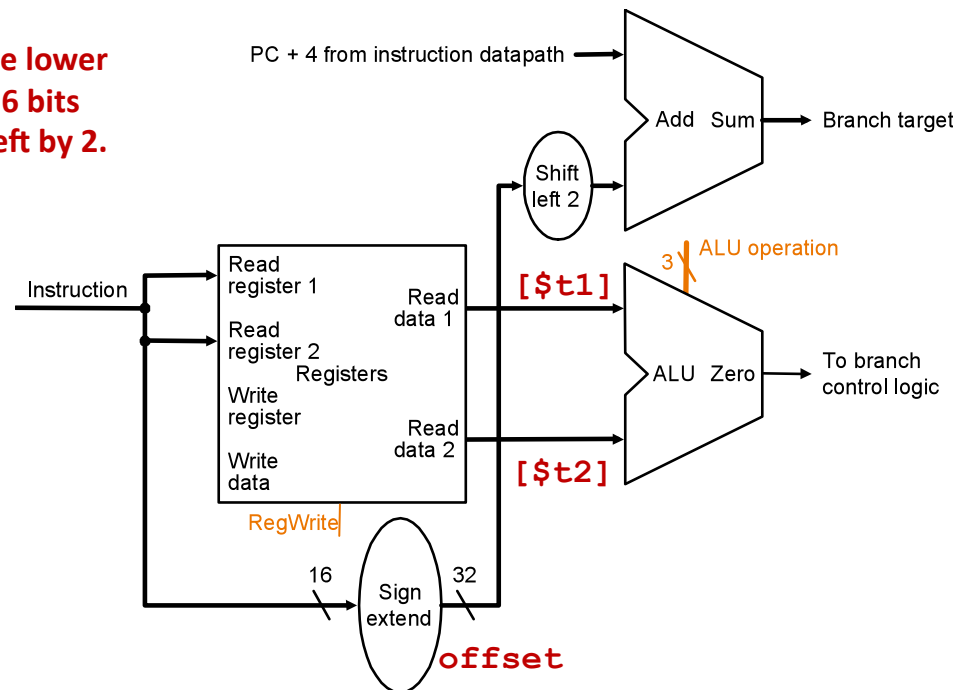




Datapath for Branch/Jump Instructions

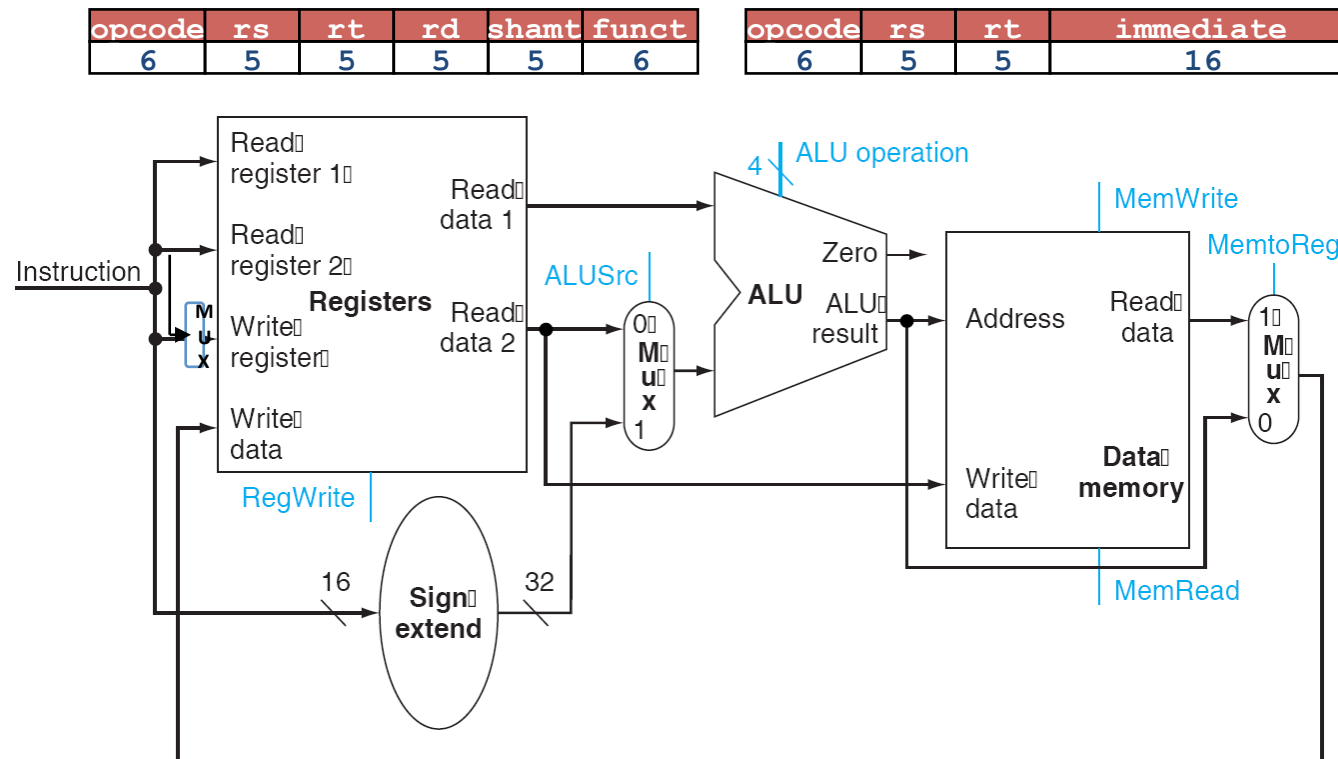
- Branch instructions: `beq/bne $t1,$t2,offset`
 - Decide if branch taken/not taken: Two registers are compared (using ALU, check zero output)
 - Branch target address: Use 16-bit offset to compute branch target address relative to the branch instruction address.
 - Need to add sign-extended offset to PC (+4)
 - Note that the offset field is shifted 2 bits to the left to get byte offset
 - Hence, we need a register file, ALU, sign-extender, and a data memory unit

For jumps, need to replace lower 28 bits of PC with lower 26 bits from instruction shifted left by 2.



Combining the Datapaths

- The previous datapaths are combined to implement *lw, sw, beq, add, sub, and, or, slt*.
- For this single-cycle datapath, all instructions execute in 1 cc, hence no resource can be used more than once per instruction. Hence,
 - Need separate instruction memory from data memory
 - Duplicate functional units that can't be shared.
- Combining R-Format and Memory datapaths:

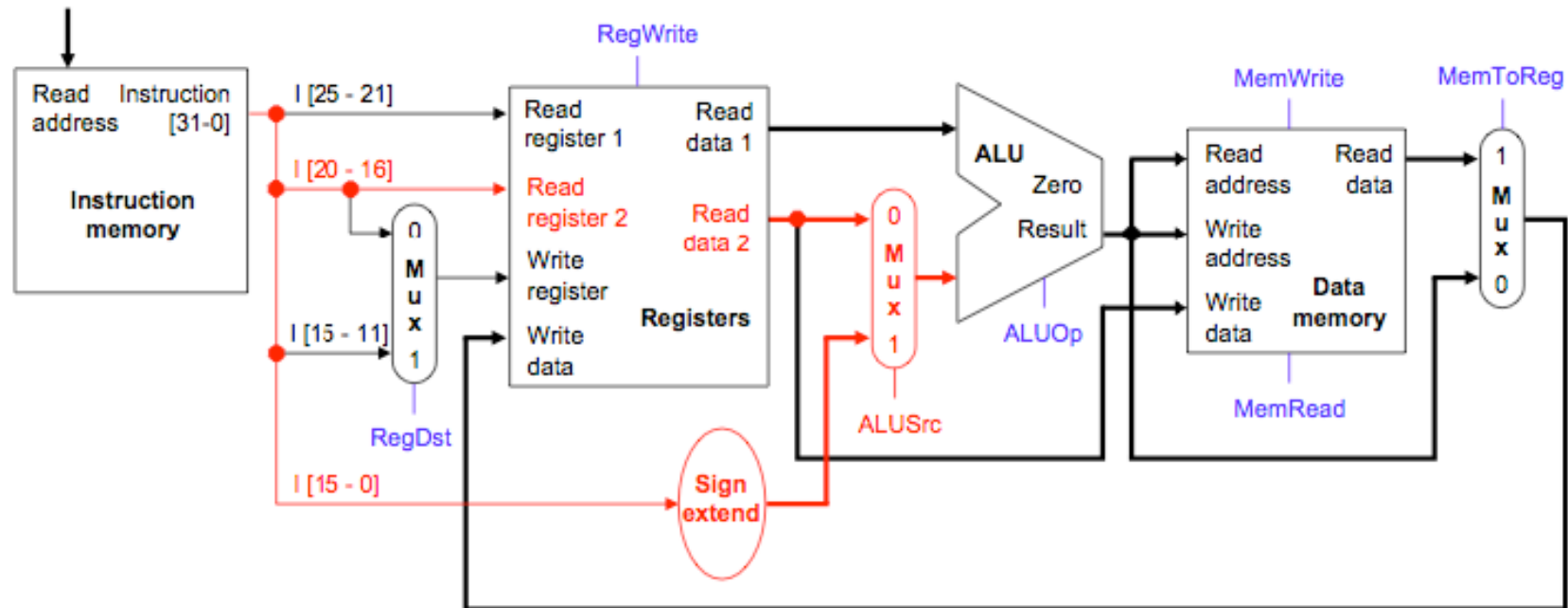


Executing a lw Instruction

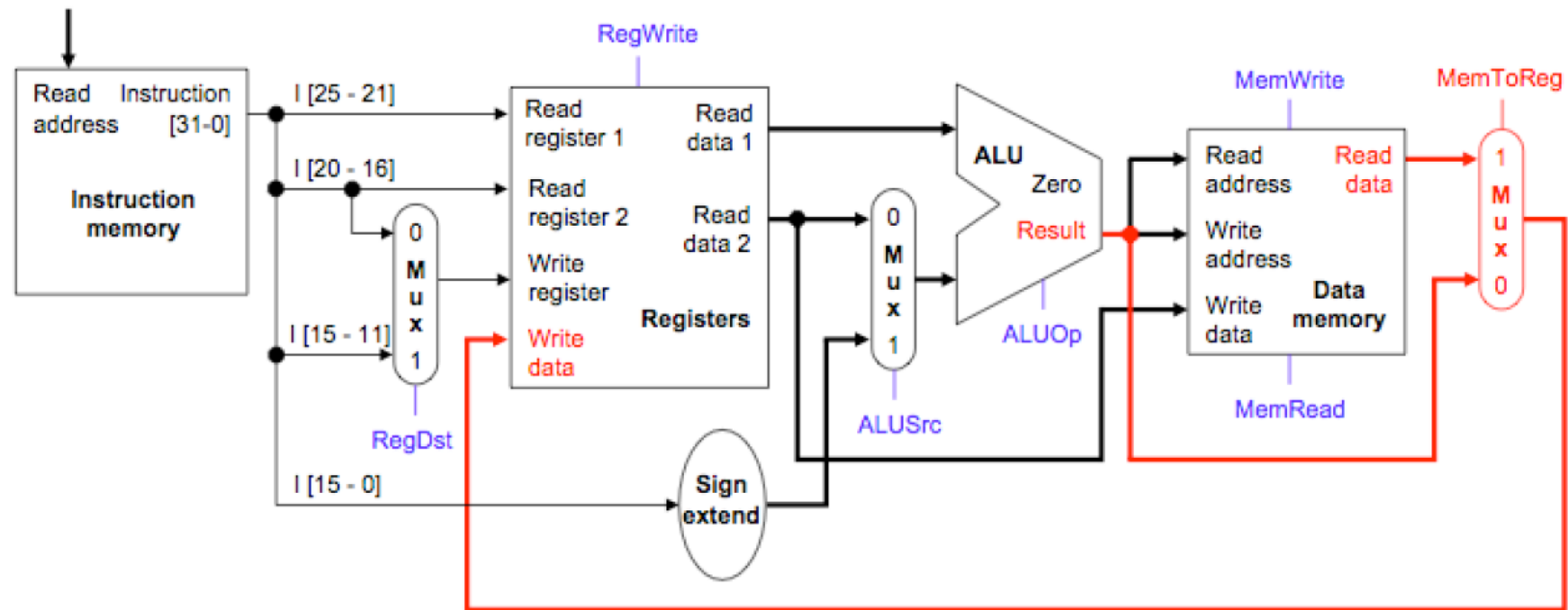
- Example:

lw \$t0, -4(\$sp)

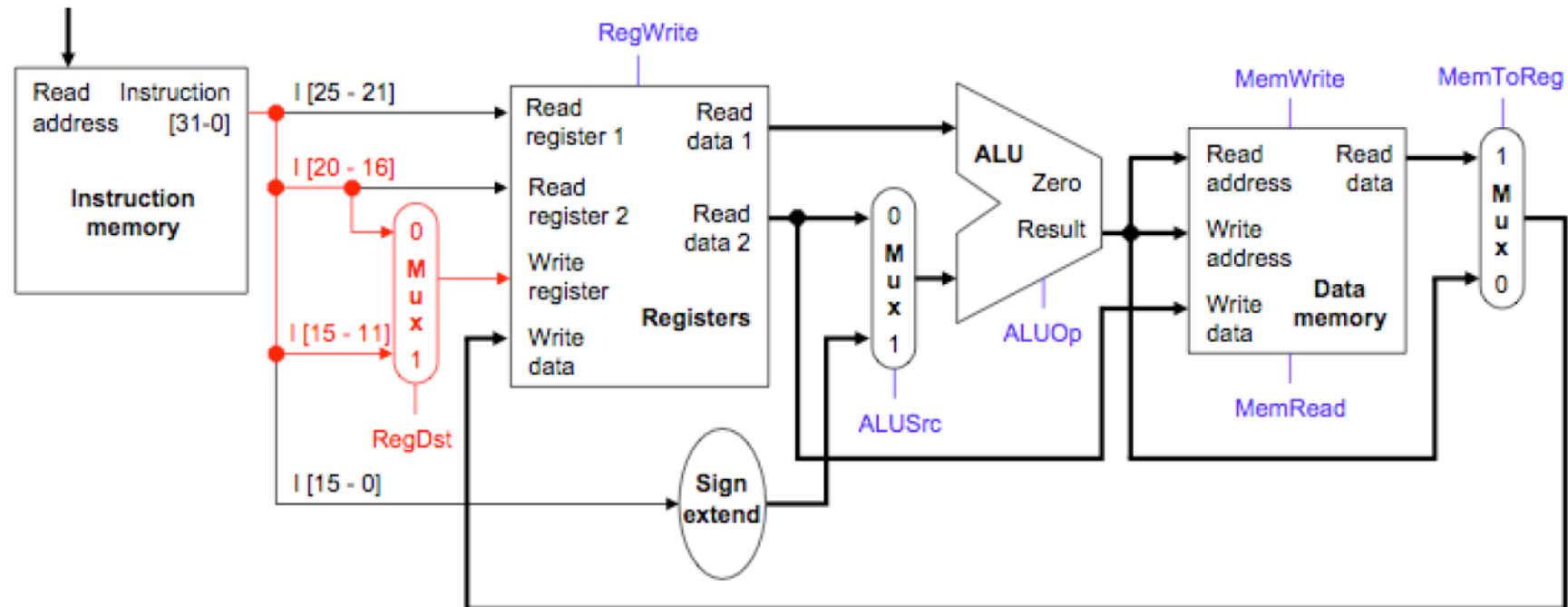
100011	11101	01000	1111 1111 1111 1100
31	26 25	21 20	16 15
			0



Executing a lw Instruction (cont'd)

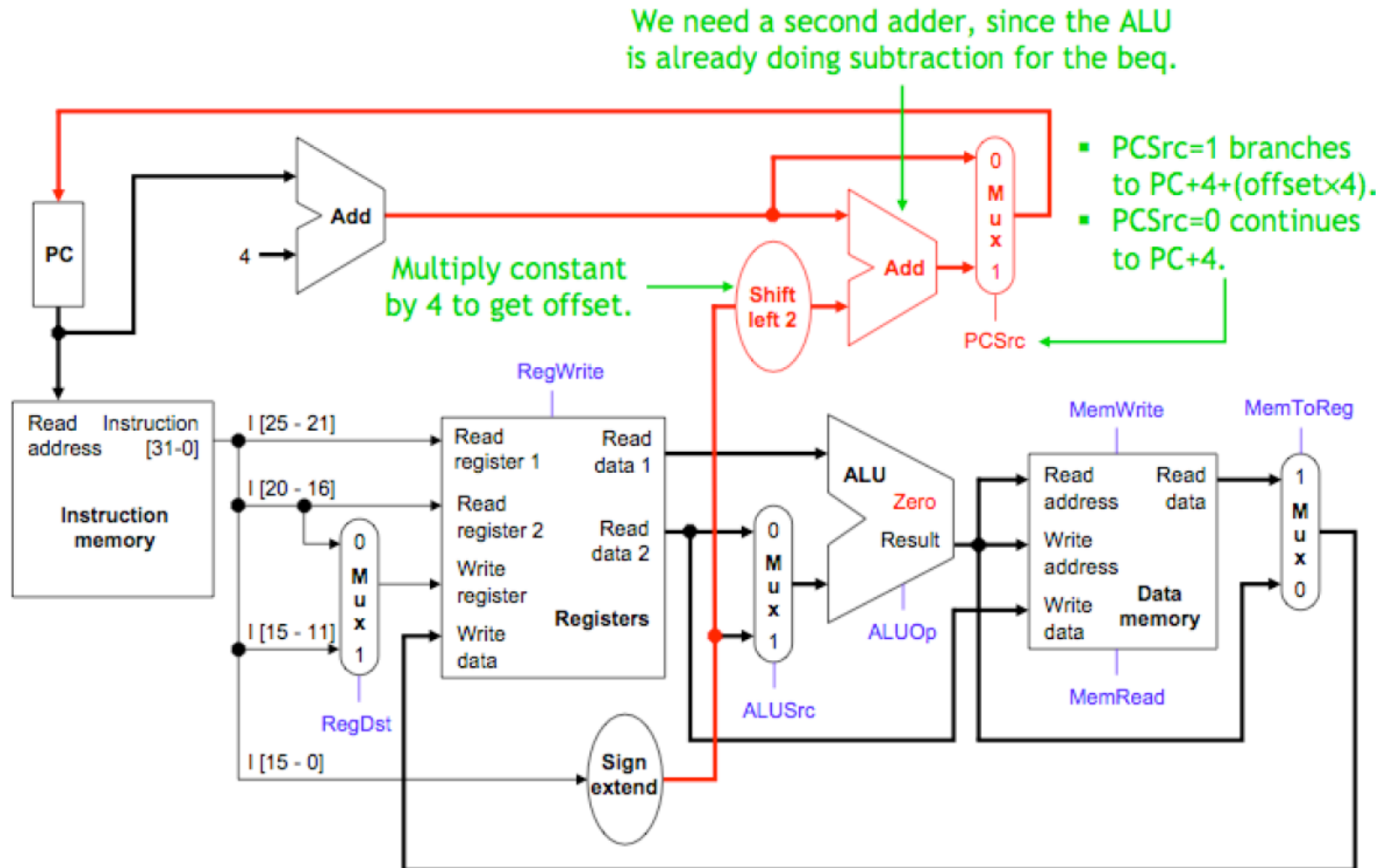


Executing a lw Instruction (cont'd)



Adding the Branch Datapath: `beq $s0, $s1, label`

- Branch instructions use the main ALU for comparison, but needs separate adder for branch target address computation.



The Final Datapath

