

MIPS Processor

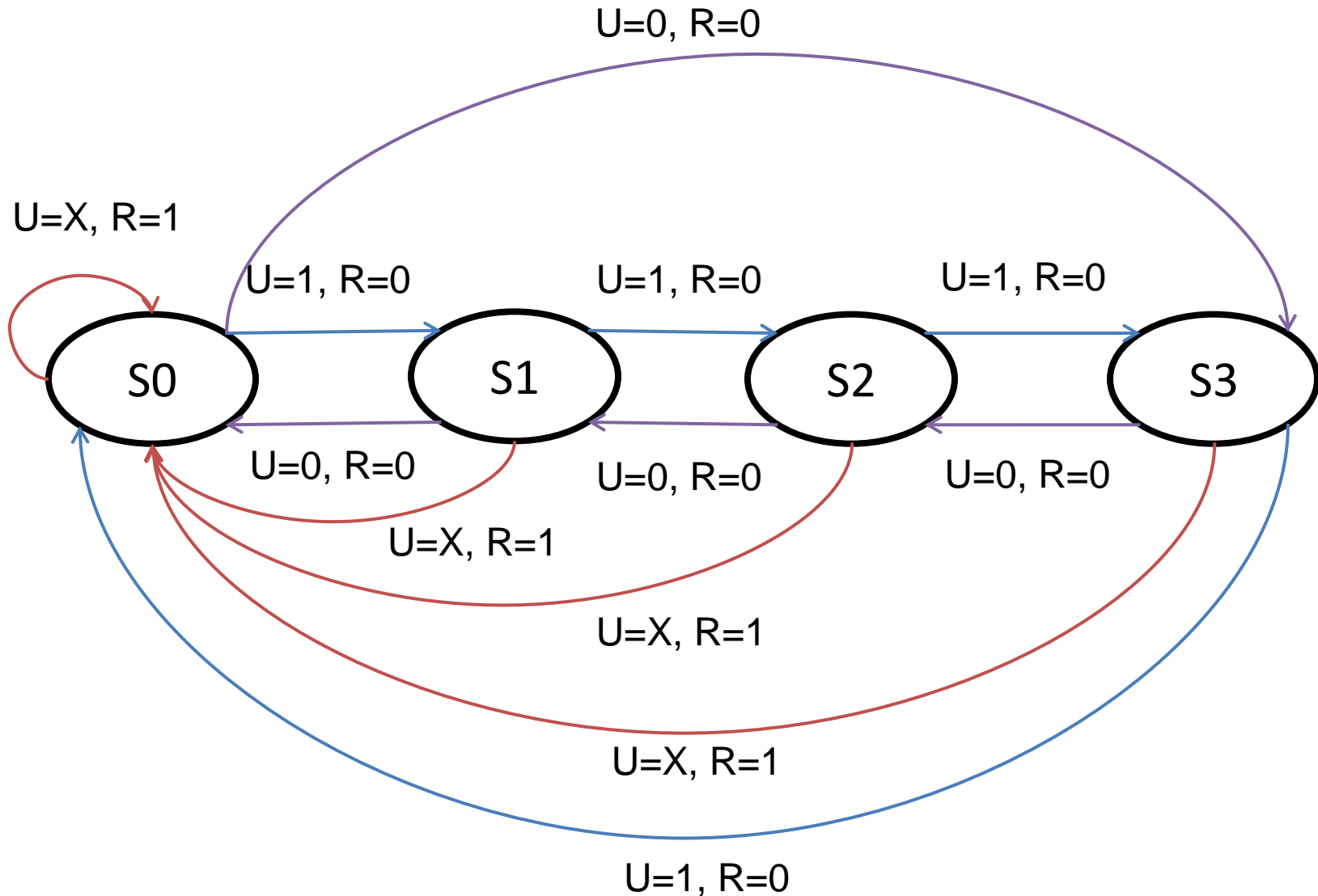
In Class Exercise Question

- Design a 2-bit up-down counter with a control signal U and a control signal R. If $U=0$, count down, else count up. If $R=1$, reset counter to 0, else count as appropriate. You don't have to worry about output, as the value of each bit of output is the same as the value of the counter.

In Class Exercise Answer

- This counter requires 4 states
- S0
 - Counter is at 00, named 00
- S1
 - Counter is at 01, named 01
- S2
 - Counter is at 10, named 10
- S3
 - Counter is at 11, named 11

In Class Exercise Answer



Q1	Q0	U	R		D1	D0
0	0	0	0		1	1
0	0	0 (X)	1		0	0
0	0	1	0		0	1
0	0	1 (X)	1		0	0
0	1	0	0		0	0
0	1	0 (X)	1		0	0
0	1	1	0		1	0
0	1	1 (X)	1		0	0
1	0	0	0		0	1
1	0	0 (X)	1		0	0
1	0	1	0		1	1
1	0	1 (X)	1		0	0
1	1	0	0		1	0
1	1	0 (X)	1		0	0
1	1	1	0		0	0
1	1	1 (X)	1		0	0

In Class Exercise Answer

- D0 =

UR \ Q1Q0	00	01	11	10
00	1			1
01				
11				
10	1			1

$$= \sim R \ \& \ \sim Q0$$

In Class Exercise Answer

- D1 =

UR \ Q1Q0	00	01	11	10
00	1		1	
01				
11				
10		1		1

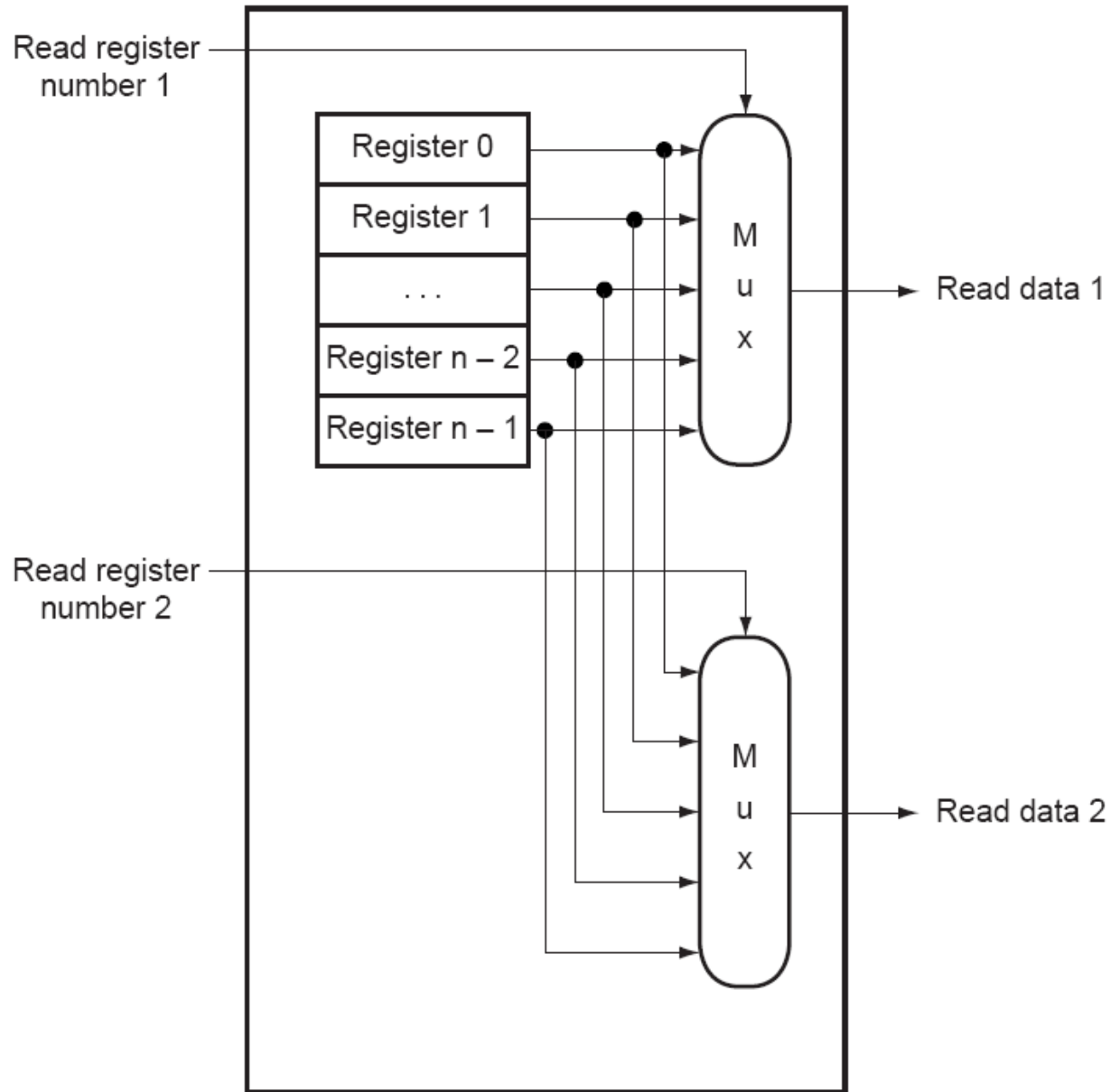
$$= (\sim Q1 \ \& \ \sim Q0 \ \& \ \sim U \ \& \ \sim R) \ | \ (\sim Q1 \ \& \ Q1 \ \& \ U \ \& \ \sim R) \ | \\ (Q1 \ \& \ Q0 \ \& \ \sim U \ \& \ \sim R) \ | \ (Q1 \ \& \ \sim Q0 \ \& \ U \ \& \ \sim R)$$

The Register File

- In MIPS, there are 32 Registers.
- In any given instruction, we will need to read up to two registers, and write to up to one register.
- Think registers as D flip-flops. Each register has 32 Dffs (one for each bit).
- The data signals are:
 - readReg1, readReg2: 5 bits. Used to specify which register to read.
 - writeReg: 5-bits. Used to specify which register to write.
 - writeData: 32-bits. The value that should be written into the writeRegister.
 - readData1, readData2: 32-bits. The value that will be read out of readRegister1 and readRegister2
- The control signals are:
 - RegWrite: whether to write or not.



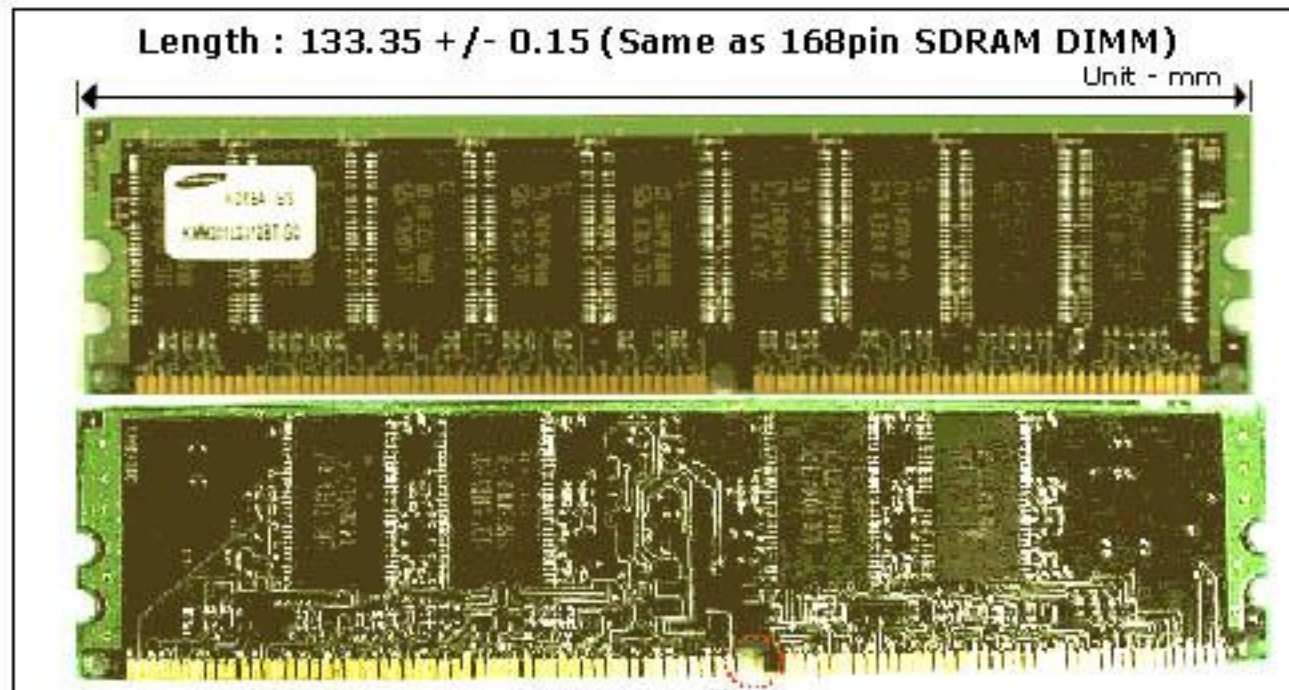
- This is for read.



To write to a register

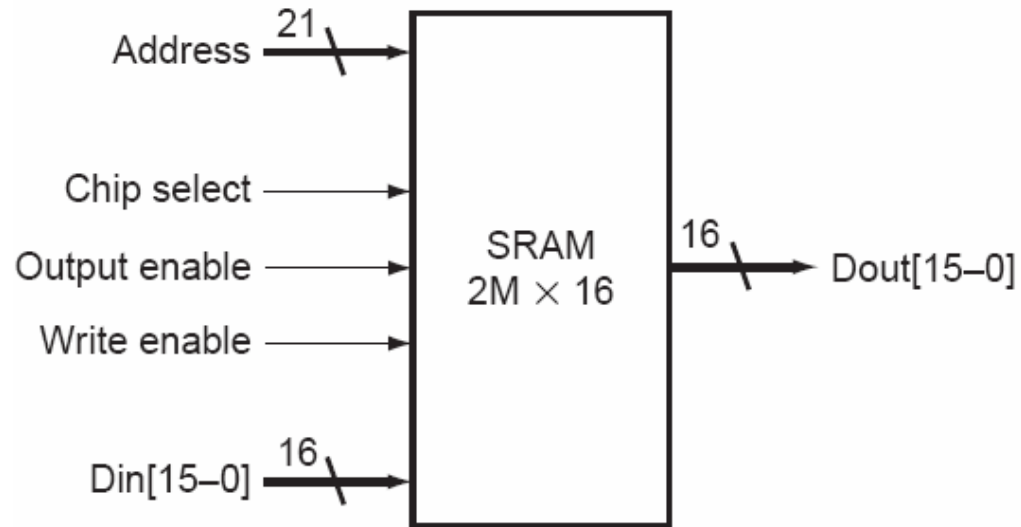
- The data is connected to every register.
- Use RegWrite, generate a ``LOAD'' signal for the register you want to write to.
 - Every register has a LOAD signal. If that signal is `1', new data will be set.
 - Only the target register's LOAD signal is `1'.

RAM



A RAM Example

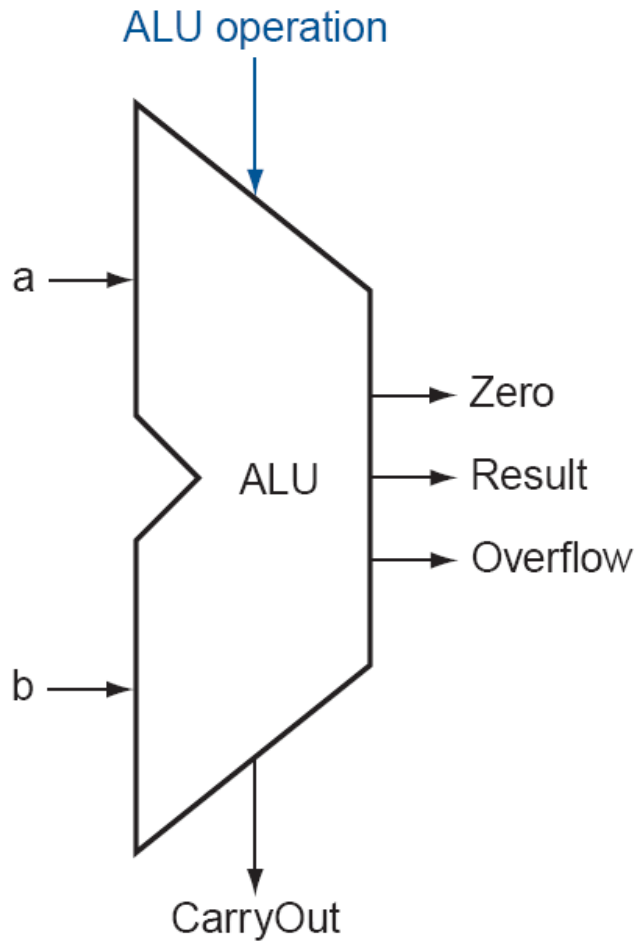
- RAM. Data signals:
 - address: 21-bits. If write, which location to write to. If read, which location to read from.
 - Din: 16-bits. The data that should be written into the location specified by the address.
 - Dout: 16-bits. The data that is read out of the location specified by the address
- RAM. Control signals:
 - Chip select: whether to use this chip or not.
 - Output enable: whether to enable output (output some voltage or in high-impedance state)
 - Write enable: whether to read or write.



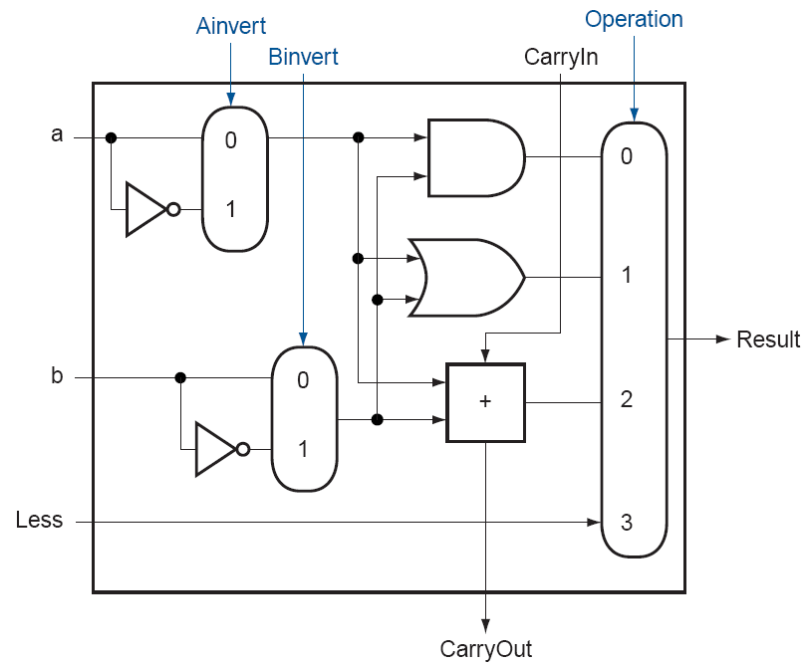
The processor

- We now know all the parts in the processor.
 - ALU
 - PC
 - Register file
- Also
 - RAM
- How to put them together? How to make them execute an instruction as we need?

ALU



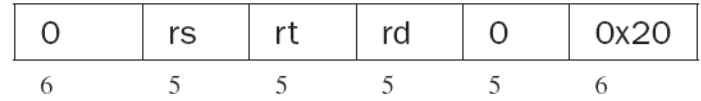
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR



Basic MIPS Implementation

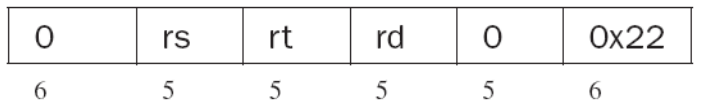
- We will focus on design of a basic MIPS processor that includes a subset of the core MIPS instruction set
 - The arithmetic-logic instructions:
 - add, sub, and, or, and slt
 - The memory-reference instructions:
 - load word and store word
 - The branching instructions:
 - branch equal and jump

add rd, rs, rt

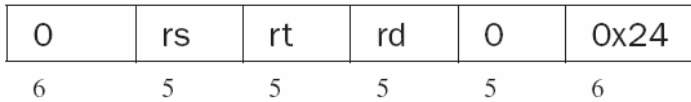


Put the sum of registers rs and rt into register rd.

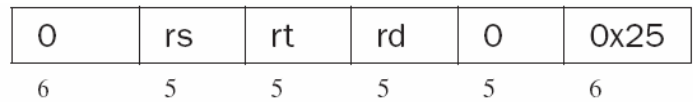
sub rd, rs, rt



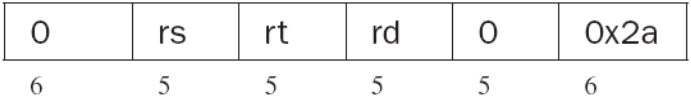
and rd, rs, rt



or rd, rs, rt

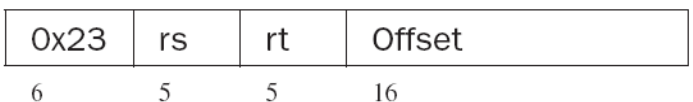


slt rd, rs, rt



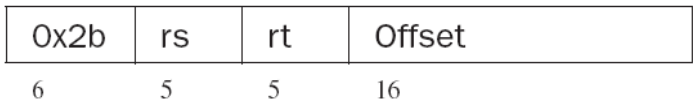
Set register rd to 1 if register rs is less than rt, and to 0 otherwise.

lw rt, address

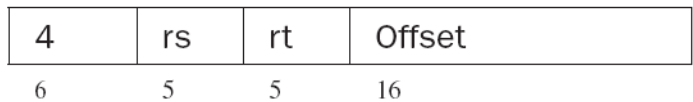


Load the 32-bit quantity (word) at address into register rt.

sw rt, address

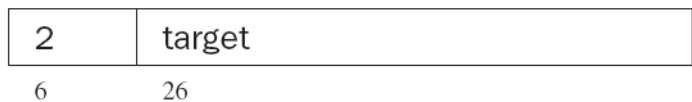


beq rs, rt, label



Conditionally branch the number of instructions specified by the offset if register rs equals rt.

j target



Unconditionally jump to the instruction at target.

The execution of an instruction

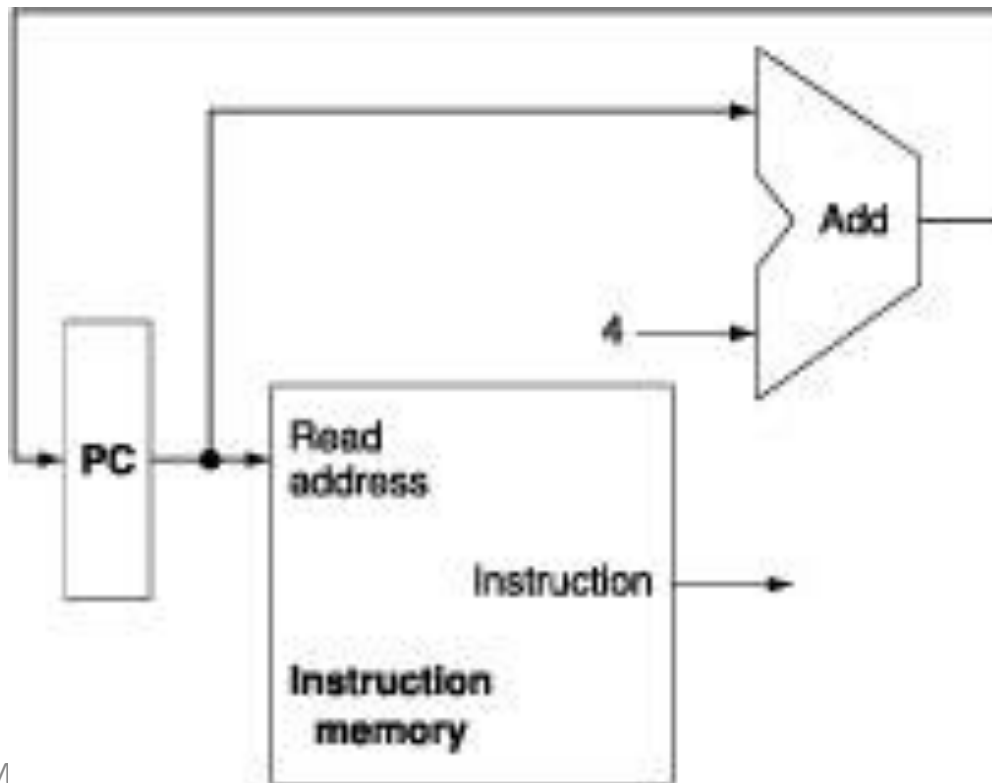
- First, we need to fetch the instruction at the address given by the current PC from instruction memory
- Second, we need to decode the instruction
- Third, we grab the values from the register file
- Fourth, we need to perform ALU operations on the values
- Fifth, the output from the ALU goes to back to the register file and/or data memory
- Sixth, based on the instruction, we need to update the PC in one of two ways
 - For sequential instructions, we then go the next instruction by setting $PC = PC + 4$.
 - For jump and branch instructions, PC will be changed based off the value from the ALU

MIPS Implementation Overview

- For every instruction, the first three steps are identical
 - Fetch the instruction from the memory according to the value of the program counter
 - Parse the fields of the instructions using the encodings
 - Read one or two registers
 - For load word, we need to read only one register
 - Most other instructions (except jump) require we read two registers
 - After the two steps, the actions required depend on the instructions
 - However, the actions are similar

Instruction Fetch and PC Increment

- Since for every instruction, the first step is to fetch the instruction from memory
 - In addition, for most instructions, the next instruction will be at $PC + 4$

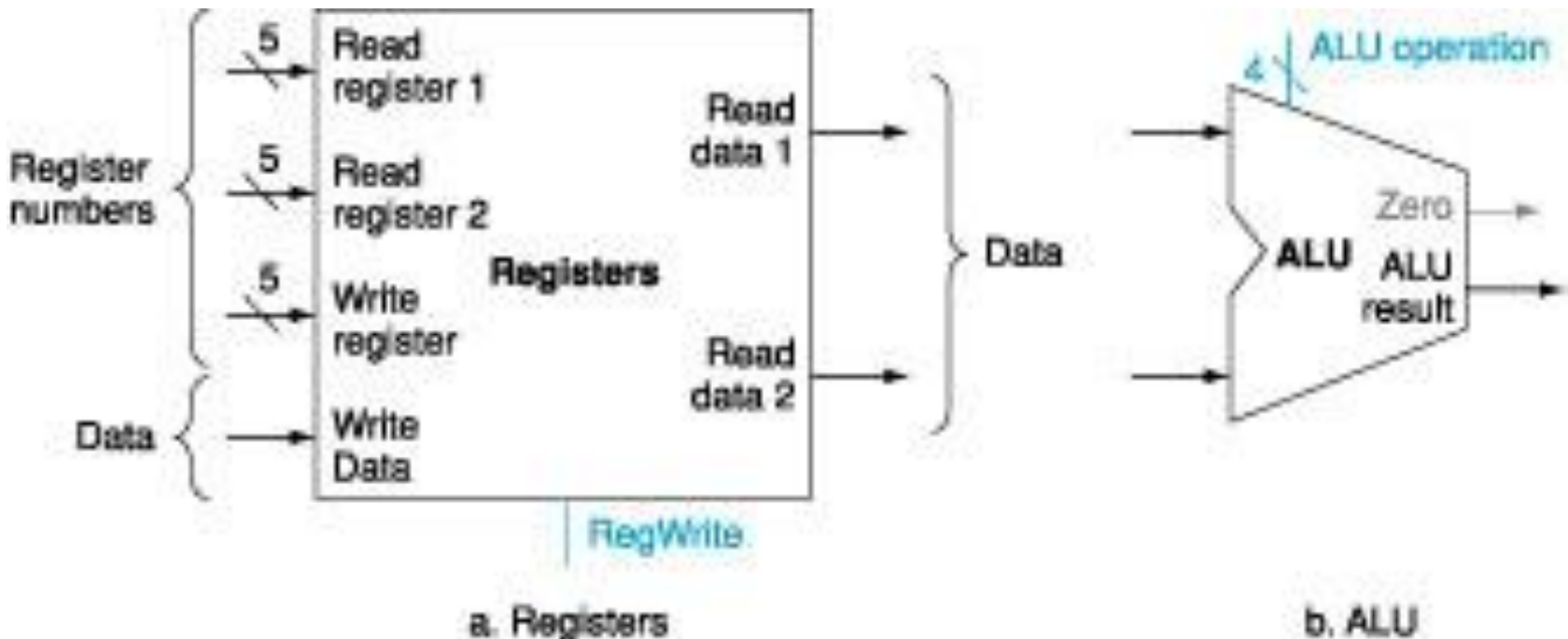


R-type Instructions

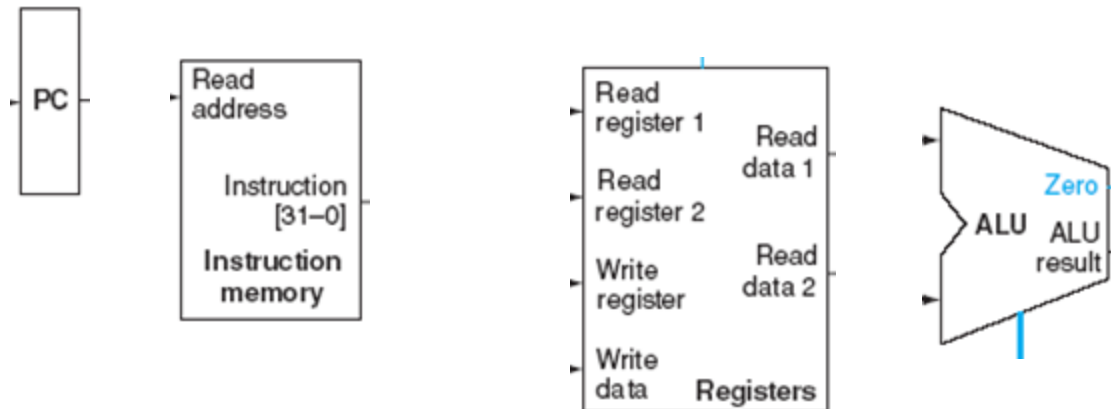
- Also called arithmetic-logical instructions
 - Including add, sub, and, or, and slt
 - Each one reads from two registers, performs an arithmetic or logical operation on the registers, and then write the result to a register

R-type Instructions

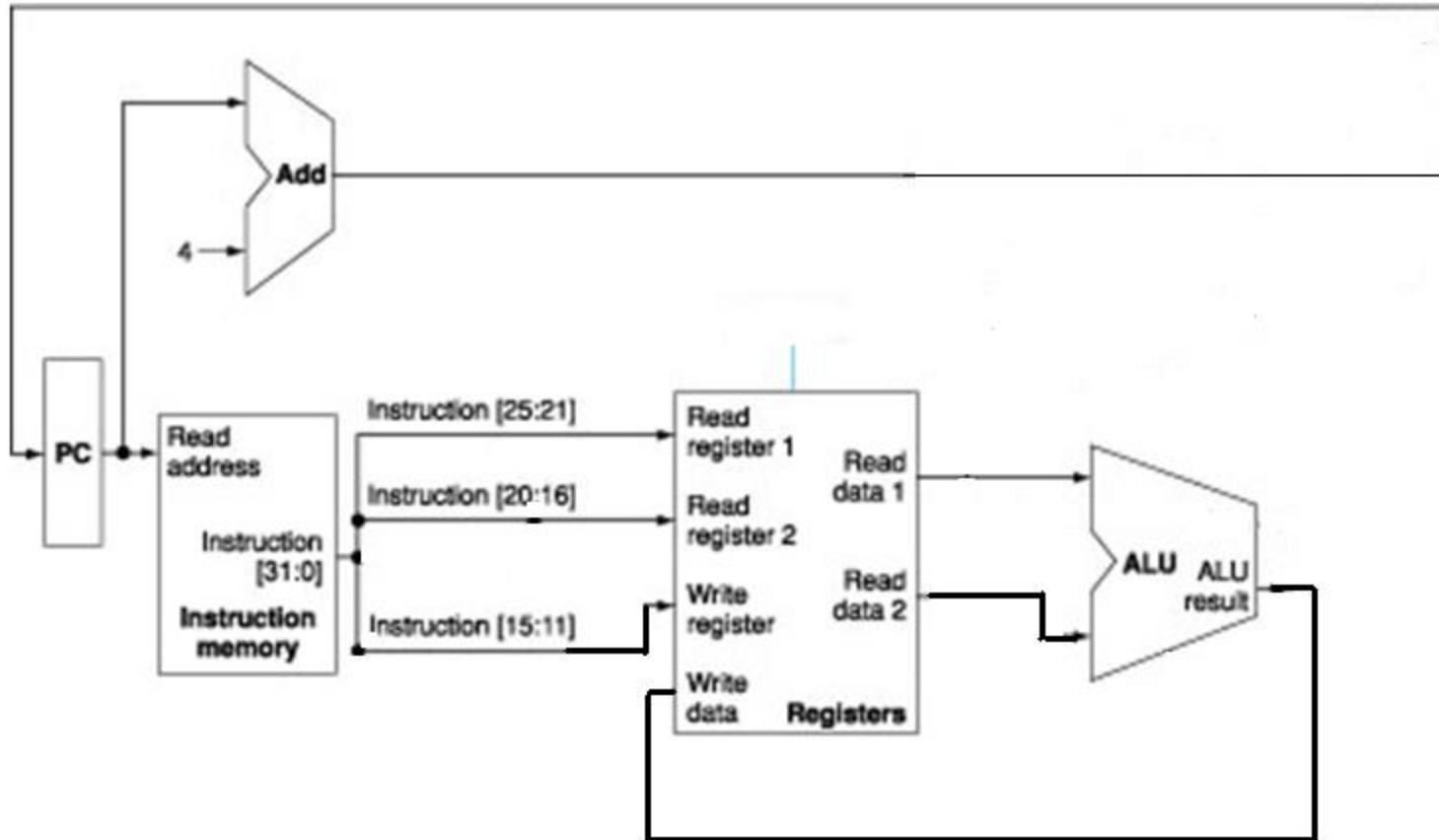
- Suppose the instruction is `add $t0, $t1, $t2`, what are the read reg1, read reg2, and write reg? What is the value of RegWrite? How to control the ALU to do add? ($\$t0 == \8)



Datapath only for R-type instructions



Datapath only for R-type instructions (Answer)

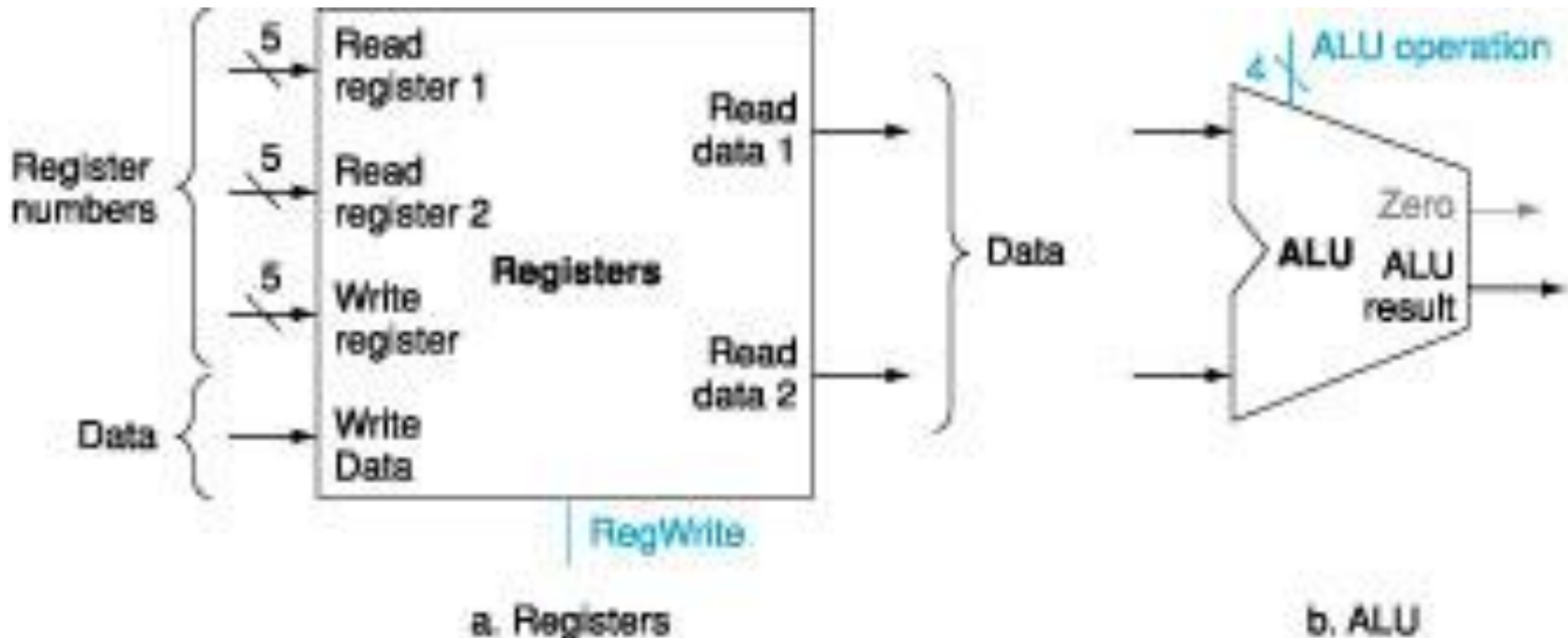


Data Transfer Instructions

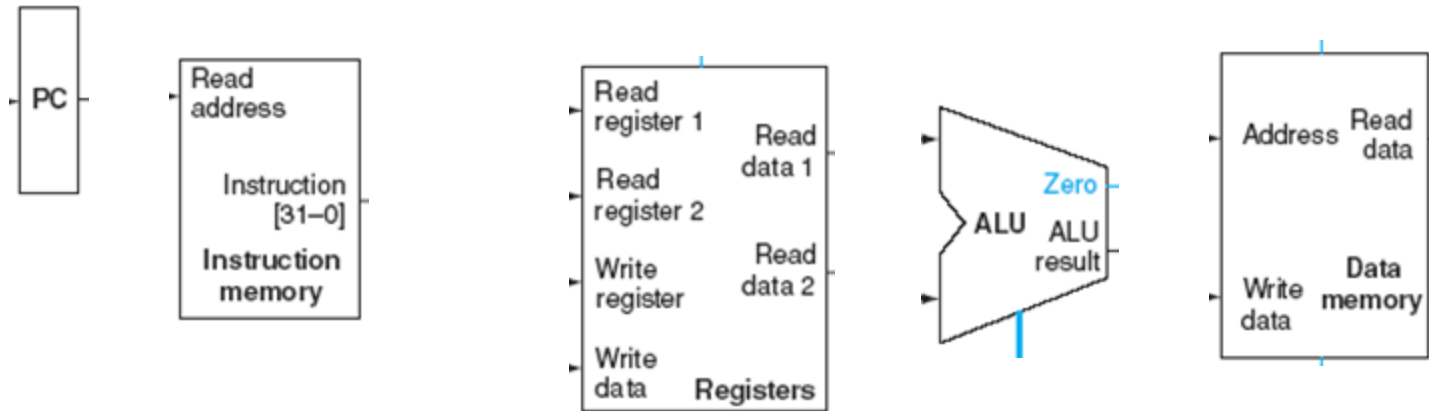
- Load word and store word instructions have the following general form
 - `lw $rt, offset_value($rs)`
 - opcode (6 bits) rs (5 bits) rt (5 bits) offset (16 bits)
 - `sw $rt, offset_value($rs)`
 - opcode (6 bits) rs (5 bits) rt (5 bits) offset (16 bits)
 - They need to compute a memory address by adding the base register to the sign-extended 16-bit offset

Load

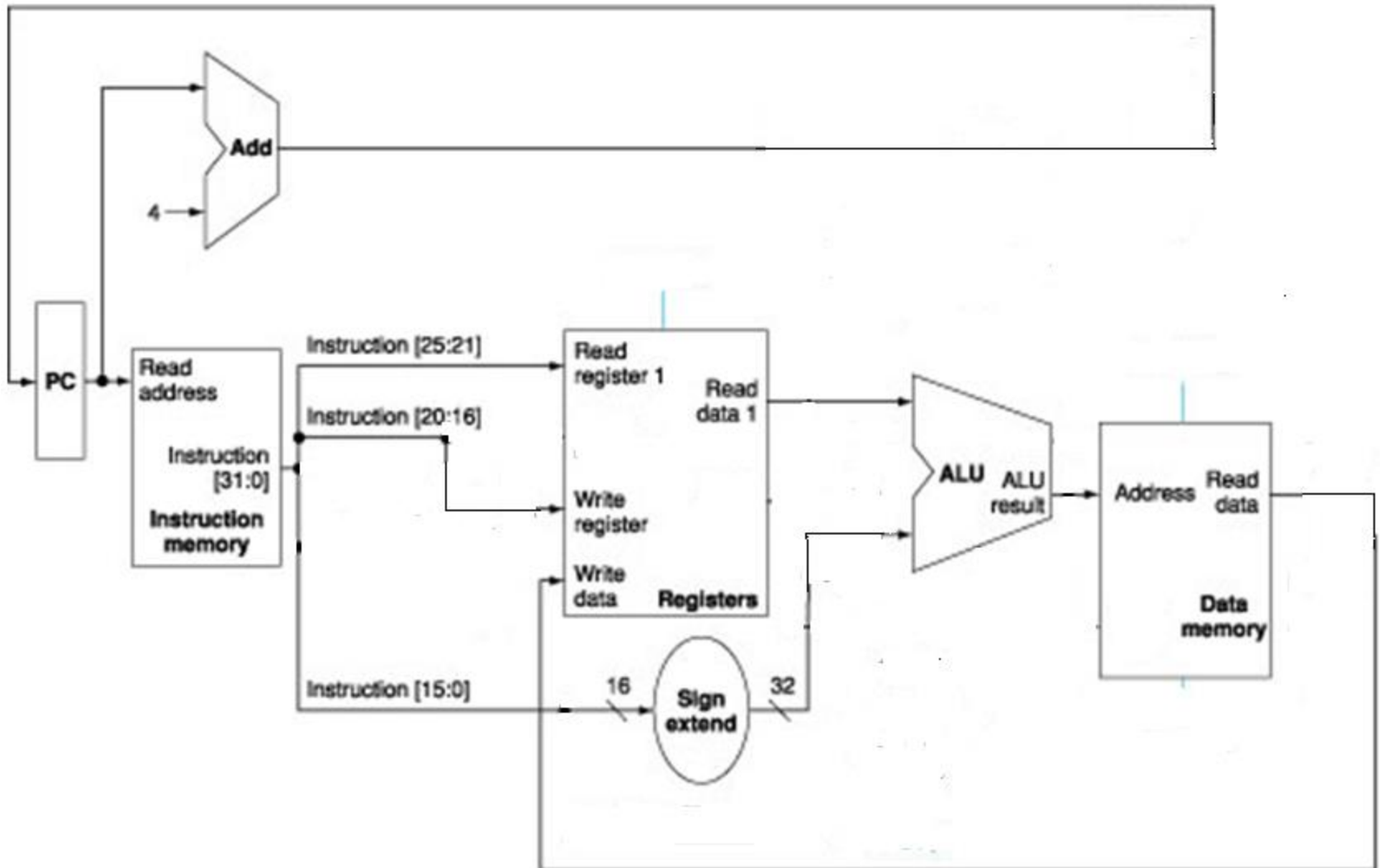
- For instruction `load $t0, 16($t1)`, what should the control signal be?



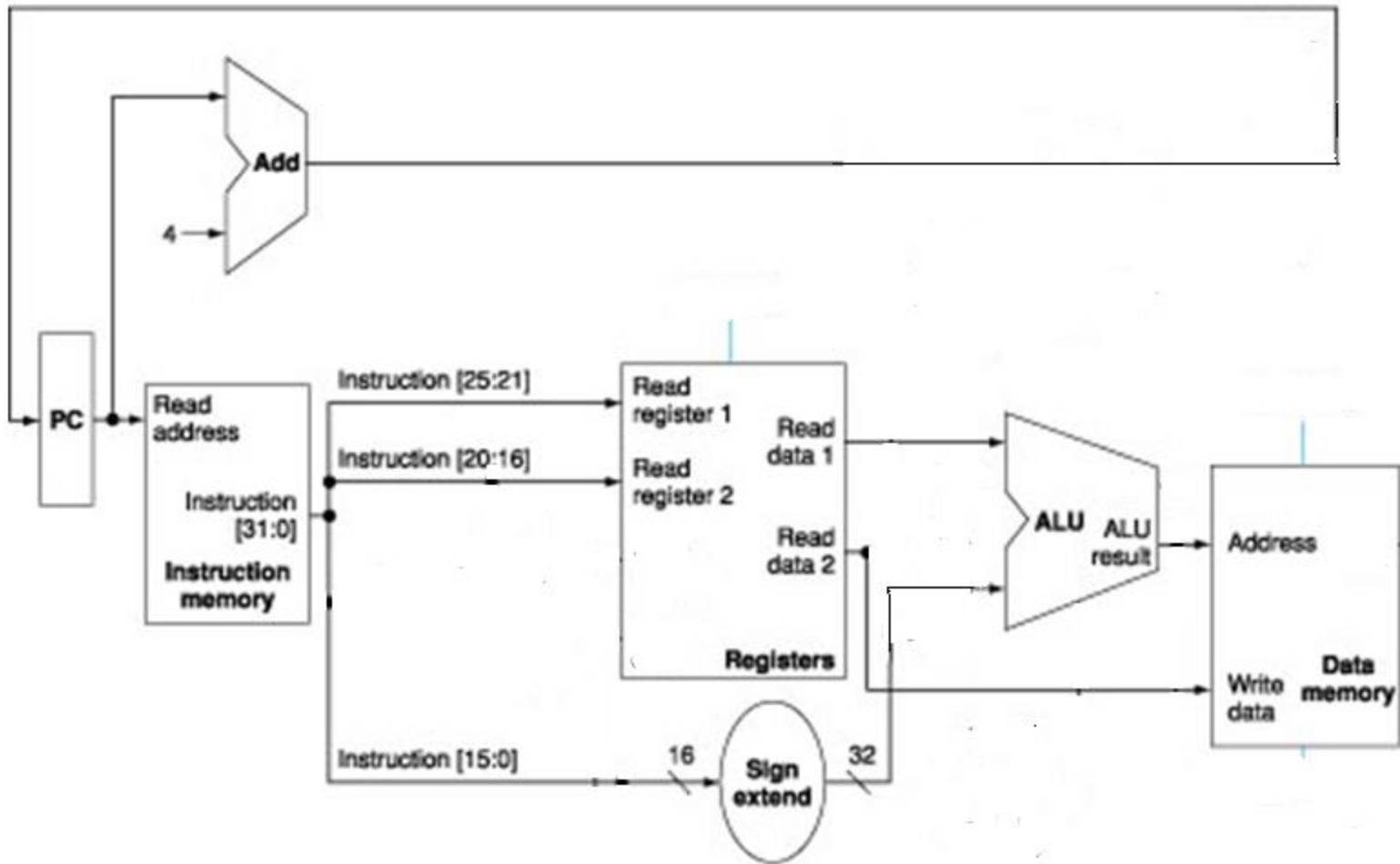
Data path



Data path only for lw (answer)



Data path only for sw (answer)

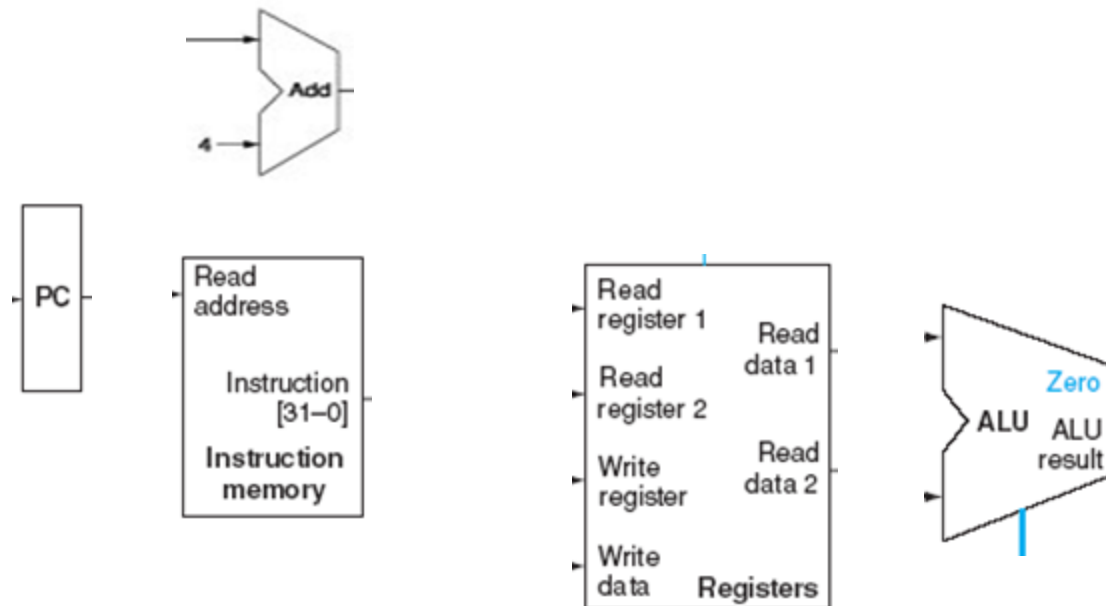


Branch Instruction

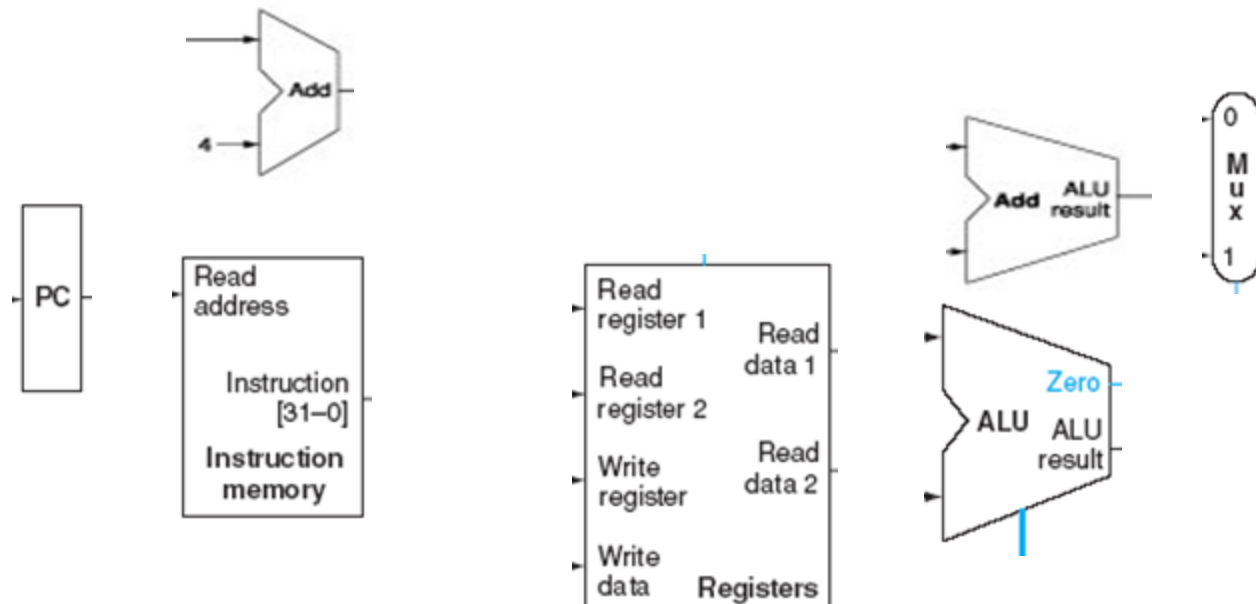
- Beq has three operands, two registers that are compared for equality and a 16-bit offset used to compute the branch-target address
 - `beq $rs, $rt, offset`
 - opcode (6 bits) rs (5 bits) rt (5 bits) offset (16 bits)
 - Note that the 16-bit offset is given in terms of instructions, not bytes and is relative to PC + 4

Designing a processor only for beq

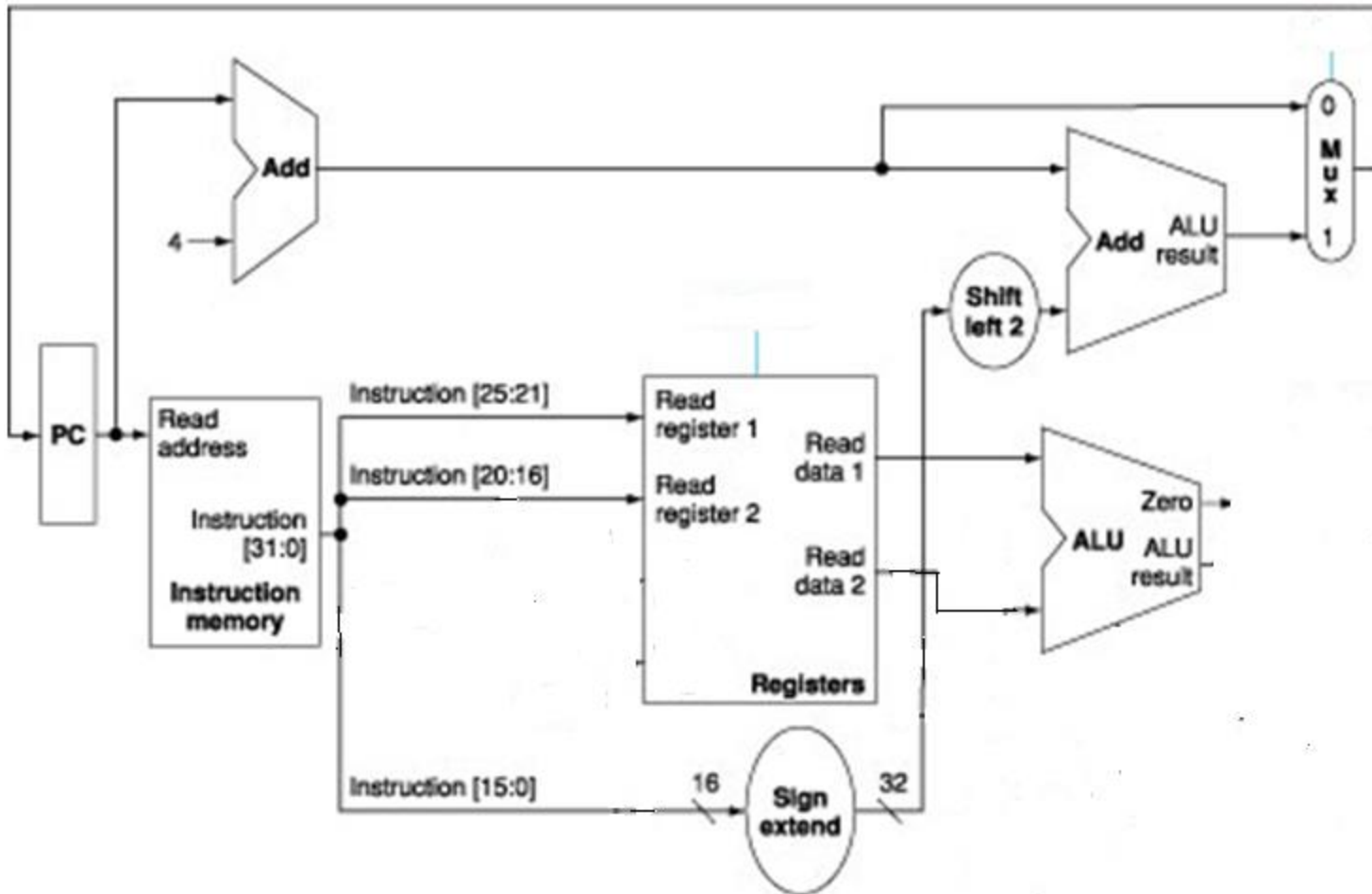
Are these enough? How many adders do we need? How to do the selection?



Designing a processor only for beq



Designing a processor only for beq (answer)



In class exercise