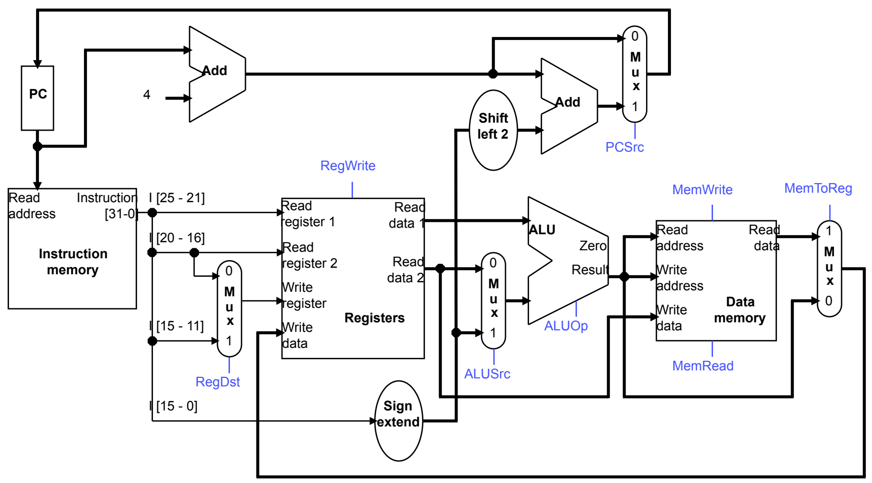
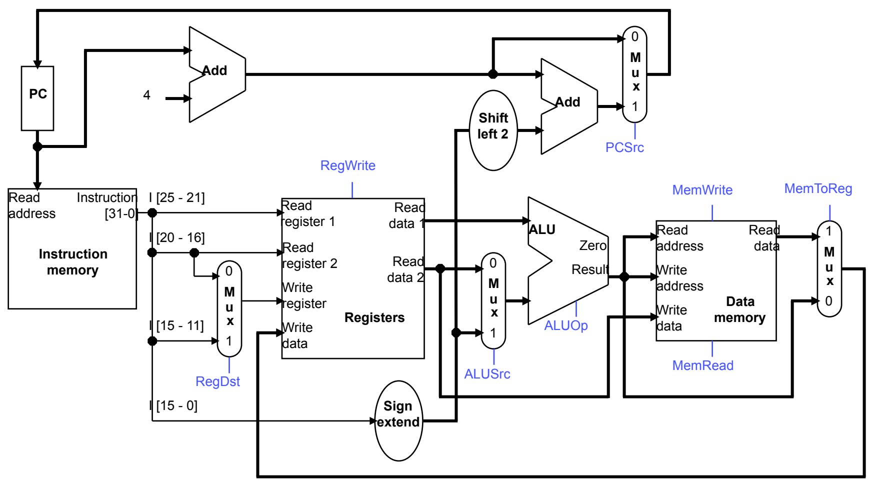
Review: What is it? What does it do?



1

slti $4, $5, 6



2

Control

The control unit is responsible for setting all the control signals so that each instruction is executed properly

- ❖ The control unit's input is the 32-bit instruction word
- ❖ The outputs are values for the blue control signals in the datapath

Most of the signals can be generated from the instruction opcode alone, and not the entire 32-bit word

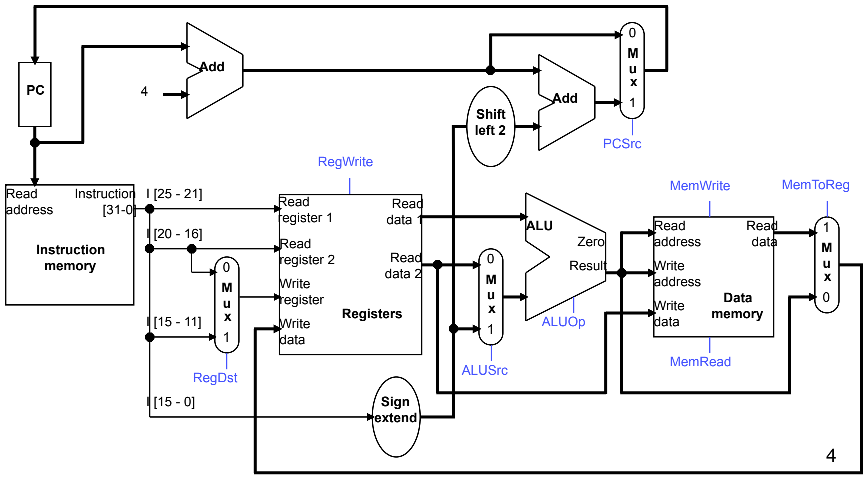
To illustrate the relevant control signals, we will show the route that is taken through the datapath by R-type, lw, sw and beq instructions

3

R-type instruction path

R-type instructions include add, sub, and, or, and slt

ALUOp is determined by the instruction's "func" field

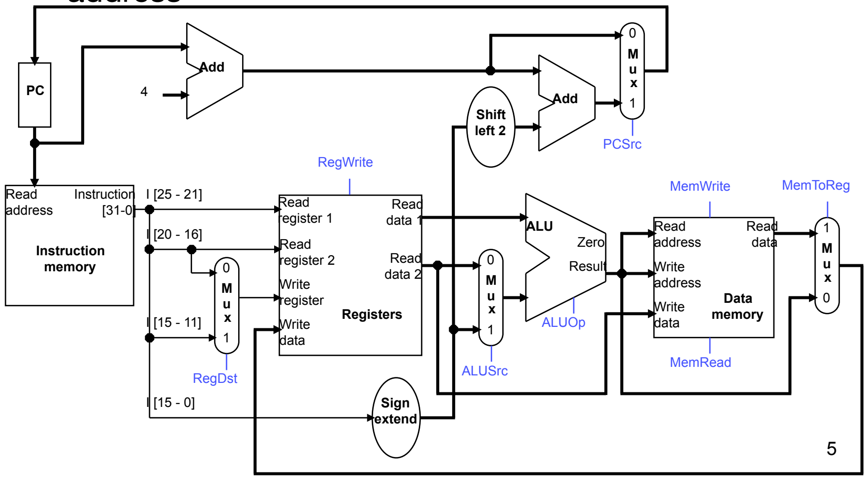


4

lw instruction path

An example load instruction is lw $t0, –4($sp)

ALUOp must be 010 (add) to compute the effective address

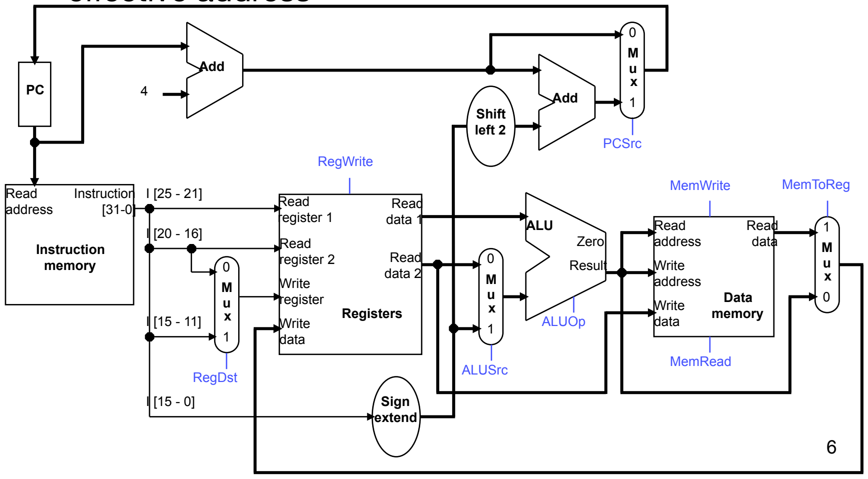


5

sw instruction path

An example store instruction is sw $a0, 16($sp)

ALUOp must be 010 (add) again to compute the effective address

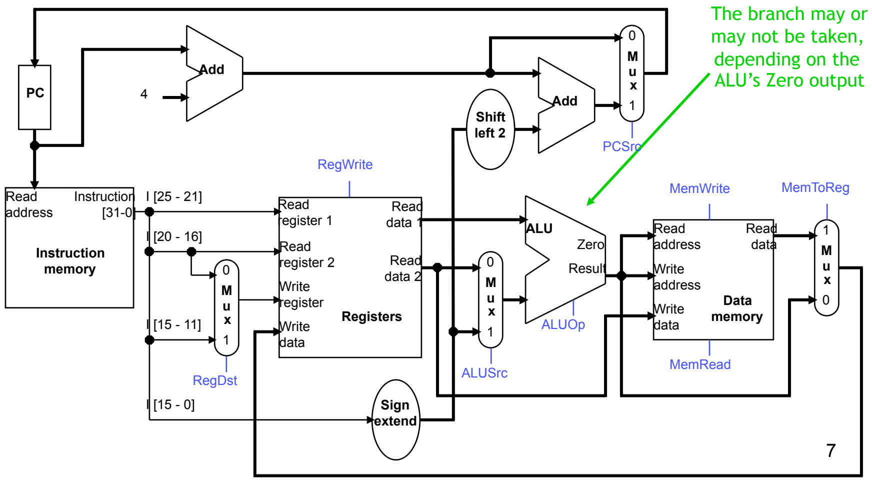


6

beq instruction path

A sample branch instruction is beq $at, $0, offset.

The ALUOp is 110 (subtract) to test for equality



The branch may or may not be taken, depending on the ALU's Zero output

7

Control signal table

Operation	RegDst	RegWrite	ALUSrc	ALUOp	MemWrite	MemRead	MemToReg
add	1	1	0	010	0	0	0
sub	1	1	0	110	0	0	0
and	1	1	0	000	0	0	0
or	1	1	0	001	0	0	0
slt	1	1	0	111	0	0	0
lw	0	1	1	010	0	1	1
sw	X	0	1	010	1	0	X
beq	X	0	0	110	0	0	X

sw and beq are the only instructions that don't write any registers

lw and sw are the only instructions that use the constant field. They also depend on the ALU to compute the effective memory address

ALUOp for R-type instructions depends on the instructions' func field

The PCSrc control signal (not listed) should be set if the instruction is beq *and* the ALU's Zero output is true

8

Generating control signals

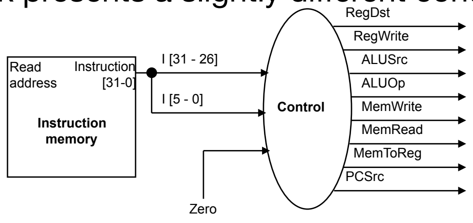
The control unit needs 13 bits of inputs

- ❖ Six bits make up the instruction's opcode
- ❖ Six bits come from the instruction's func field
- ❖ It also needs the Zero output of the ALU

The control unit generates 10 bits of output, corresponding to the signals mentioned earlier

You can build the actual circuit by using big K-maps, big Boolean algebra, or big circuit design programs

The textbook presents a slightly different control unit



9

Logic Array

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

10

A Closer Look At the Operation

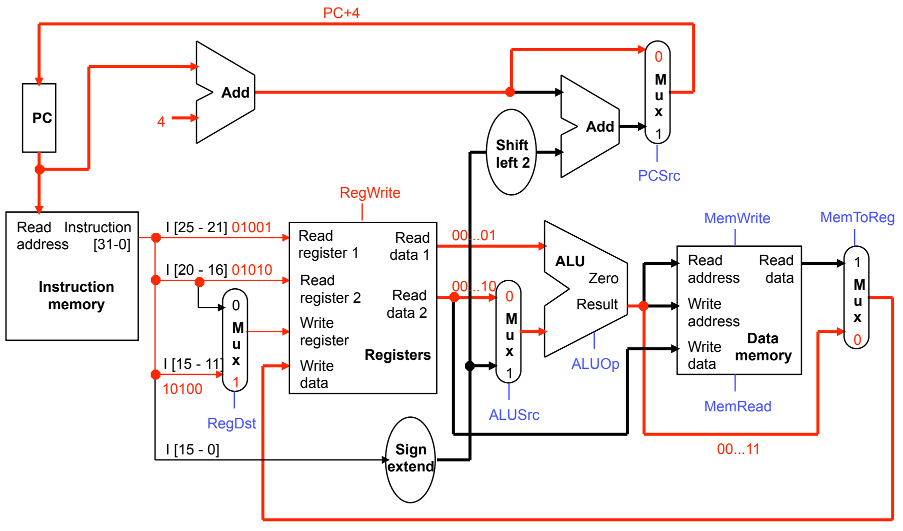
Consider the instruction add $t1, $t1, $t2

000000	01001	01010	01001	00000	1000000
opcode	rs	rt	rd	shamt	func

- Assume $t1 and $t2 initially contain 1 and 2 respectively.
- Executing this instruction involves several steps.
  1. The instruction word is read from the instruction memory, and the program counter is incremented by 4
  2. The sources $t1 and $t2 are read from the register file
  3. The values 1 and 2 are added by the ALU
  4. The result (3) is stored back into $t1 in the register file

11

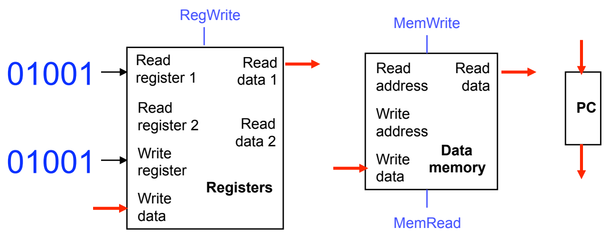
The add moving through the datapath



12

State Elements

- In an instruction like add $t1, $t1, $t2, how do we know $t1 is not updated until *after* its original value is read?



13

The datapath and the clock

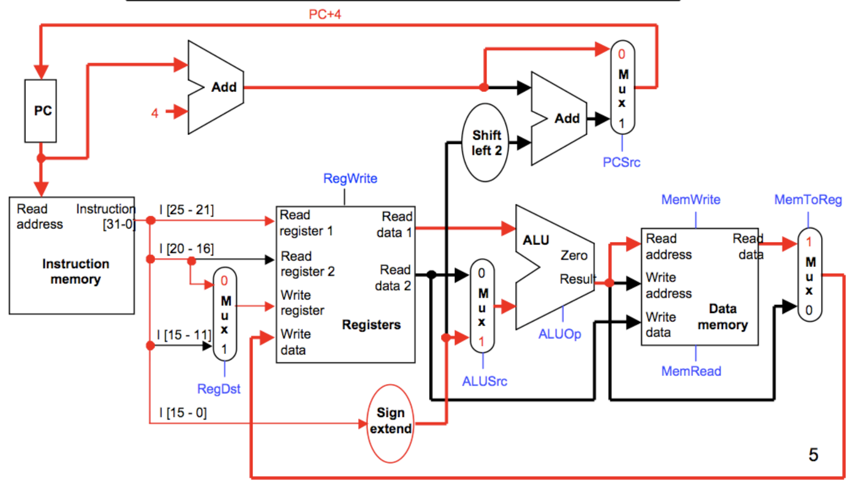
- STEP 1: A new instruction is loaded from memory. The control unit sets the datapath signals appropriately so that
  - ❖ registers are read,
  - ❖ ALU output is generated,
  - ❖ data memory is read and
  - ❖ branch target addresses are computed
- STEP 2:
  - ❖ The register file is updated for arithmetic or lw instructions
  - ❖ Data memory is written for a sw instruction
  - ❖ The PC is updated to point to the next instruction
- In a single-cycle datapath everything in Step 1 must complete within one clock cycle.

14

The lw moves through the datapath

Consider lw $t0, –4($sp) execution

100011	11101	01000	1111 1111 1111 1100

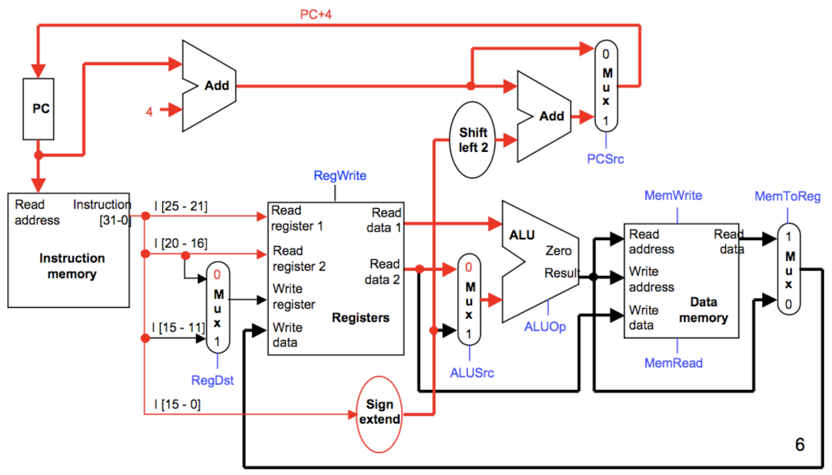


5

The beq moves through the datapath

Recall beq $at, $0, offset compute branch addr: PC + 4 + (offset x 4)

000100	00001	00000	0000 0000 0000 0011



6

Next Steps

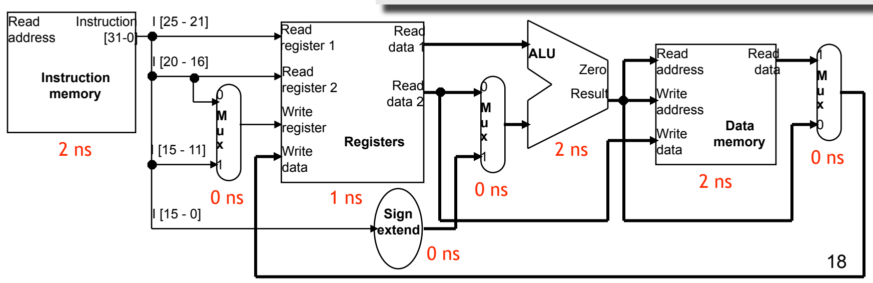
- Designing a computer is only the first step
  - ❖Next, we must consider how fast it runs,
  - ❖And how to make it run faster
- We have a “single cycle” design that assumes all operations complete within one clock cycle

The slowest instruction...

- If all instructions must complete within 1 clock cycle, then the cycle time >= *slowest* instruction
- For example, lw $t0, –4($sp) needs 8ns, assuming the delays shown here

reading the instruction memory	2ns
reading the base register $sp	1ns
computing memory address $sp-4	2ns
reading the data memory	2ns
storing data back to $t0	1ns

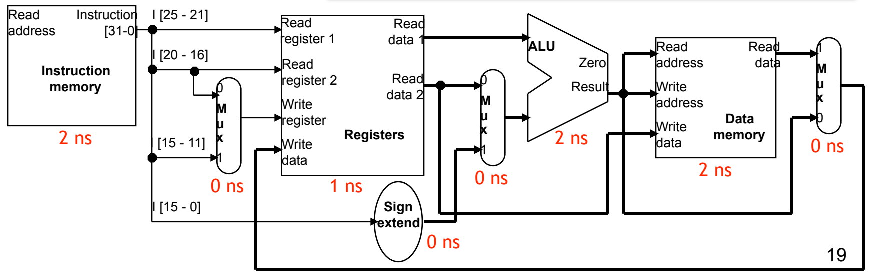
8ns



...determines the clock cycle time

- If we make the cycle time 8ns *every* instruction will take 8ns, even if they don't need that much time
- For example, the instruction add $s4, $t1, $t2 really needs just 6ns

reading the instruction memory	2ns	6ns
reading registers $t1 and $t2	1ns	
computing $t1 + $t2	2ns	
storing the result into $s0	1ns	



Summary

A datapath contains all the functional units and connections necessary to implement an instruction set architecture

❖ For our single-cycle implementation, we use two separate memories, an ALU, some extra adders, and lots of multiplexers

❖ MIPS is a 32-bit machine, so most of the buses are 32-bits wide

The control unit tells the datapath what to do, based on the instruction that's currently being executed

❖ Our processor has ten control signals that regulate the datapath

❖ The control signals can be generated by a combinational circuit with the instruction's 32-bit binary encoding as input

Next, we'll see the performance limitations of this single-cycle machine and try to improve upon it