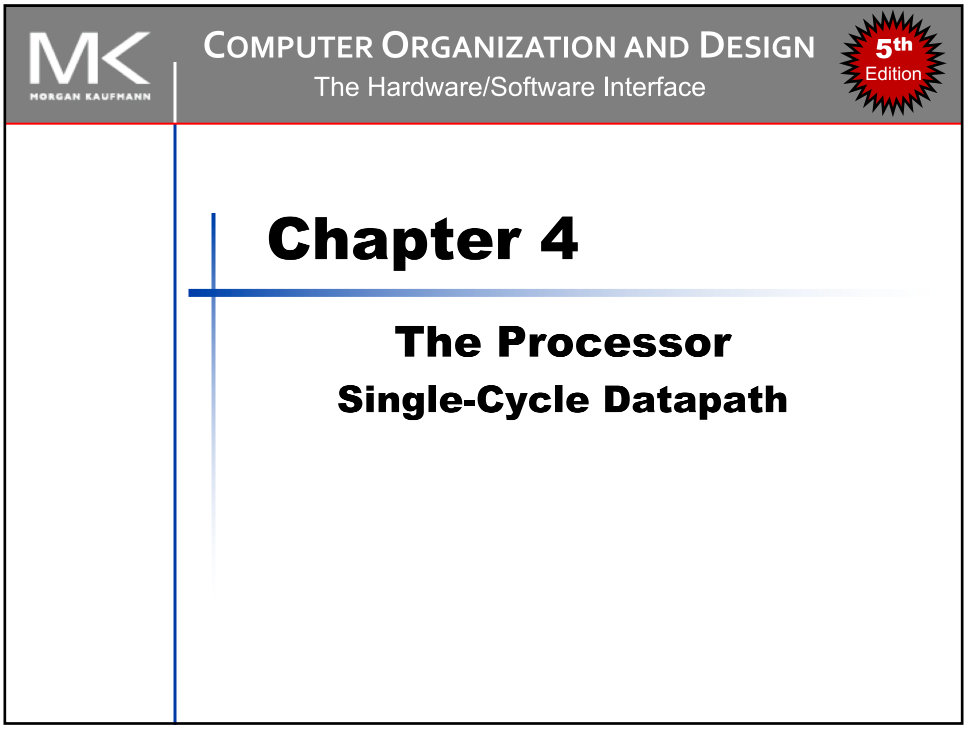


MORGAN KAUFMANN

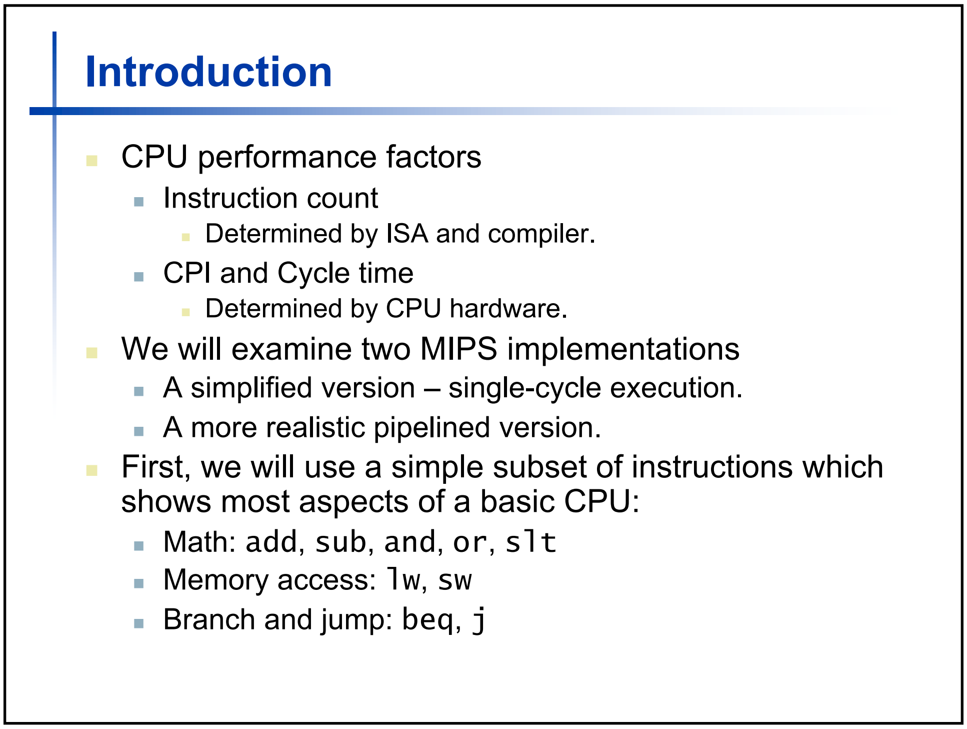
COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface

5th Edition

Chapter 4

The Processor

Single-Cycle Datapath

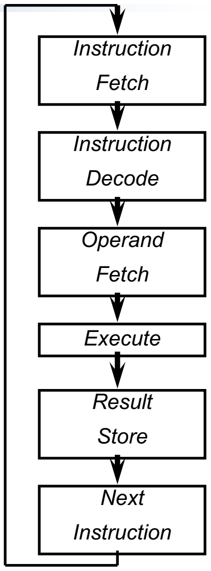


Introduction

- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler.
  - CPI and Cycle time
    - Determined by CPU hardware.
- We will examine two MIPS implementations
  - A simplified version – single-cycle execution.
  - A more realistic pipelined version.
- First, we will use a simple subset of instructions which shows most aspects of a basic CPU:
  - Math: `add`, `sub`, `and`, `or`, `slt`
  - Memory access: `lw`, `sw`
  - Branch and jump: `beq`, `j`

Stored Program Execution

- Get the instruction.
- Decide what kind of instruction it is.
- Get any necessary data.
- Execute the instruction.
- Store the result.
- Repeat forever.



Instruction Fetch

Instruction Decode

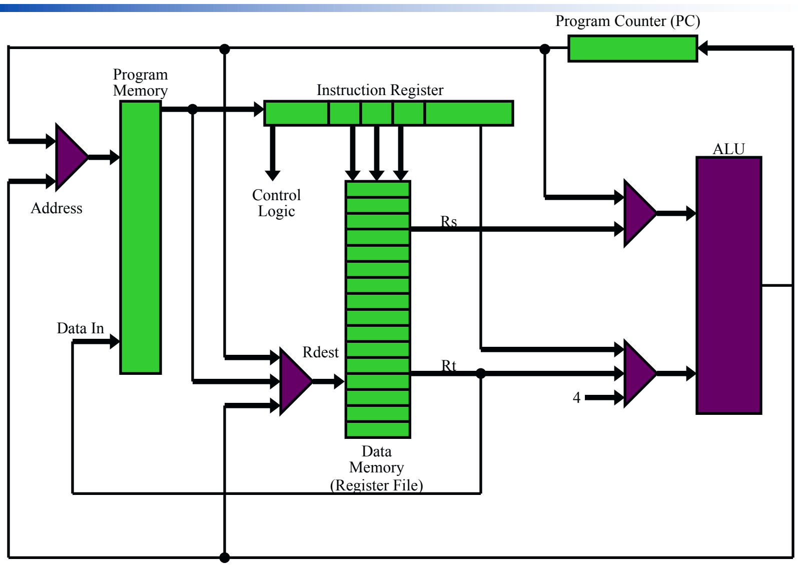
Operand Fetch

Execute

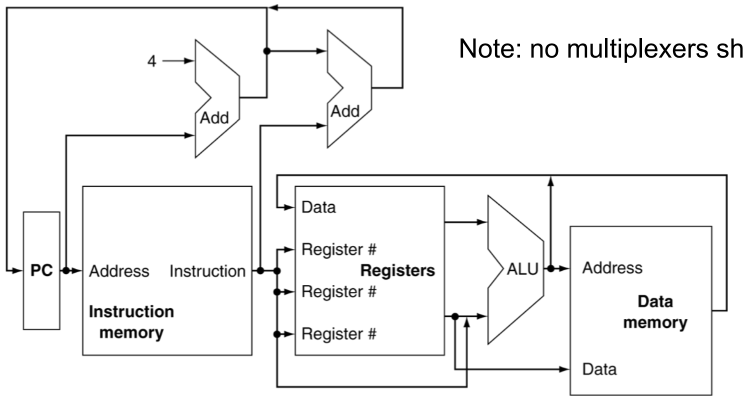
Result Store

Next Instruction

MIPS Fetch-Execute Processor Architecture



Big Picture Review

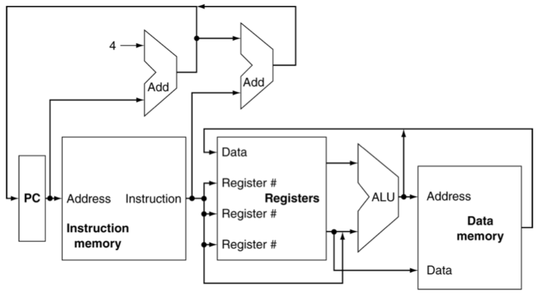


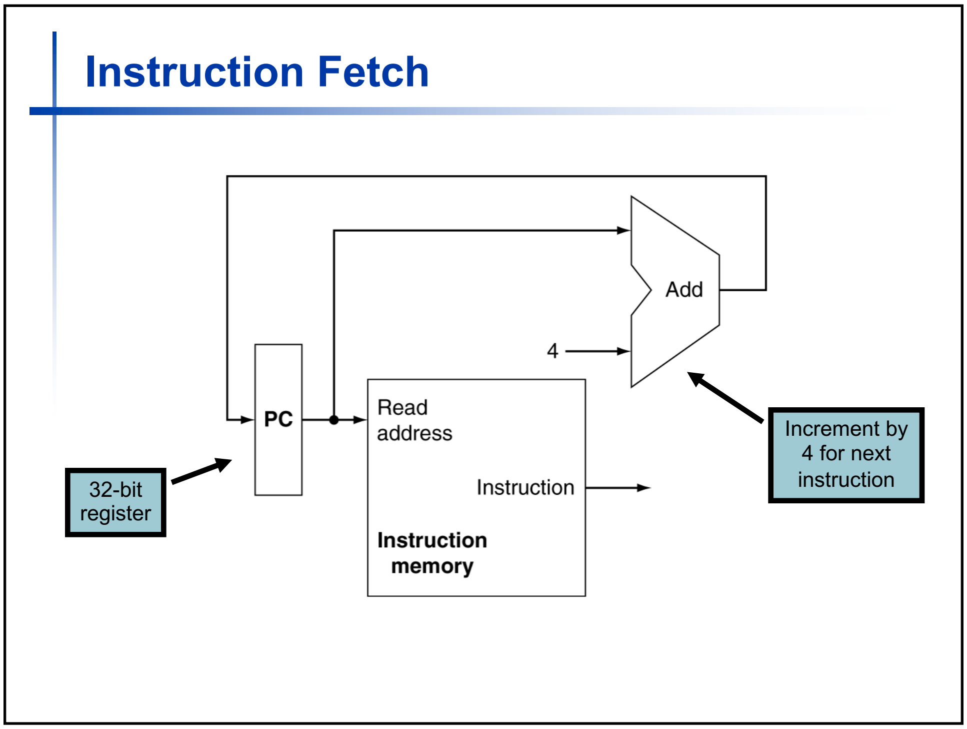
Note: no multiplexers shown.

- What is the role of the Add units?
- Explain the inputs to the register unit.
- Explain the inputs to the ALU.
- Explain the inputs to the data memory unit.

Building a Datapath

- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, mux's, memories, …
- We will build a MIPS datapath incrementally by refining the overview design shown earlier.

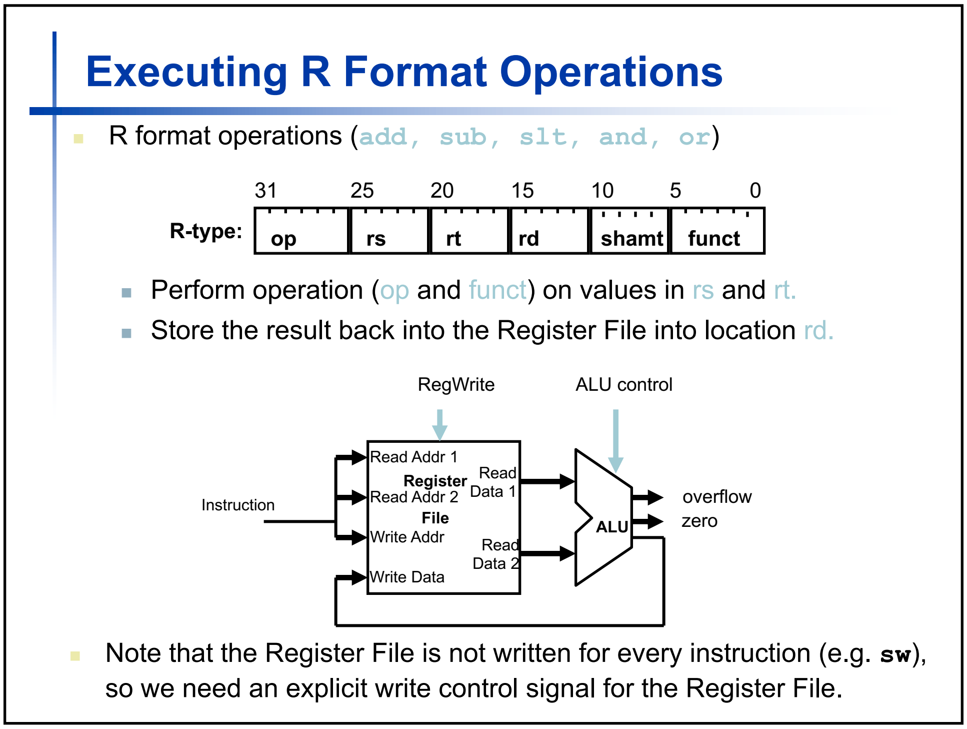




Instruction Fetch

32-bit register

Increment by 4 for next instruction



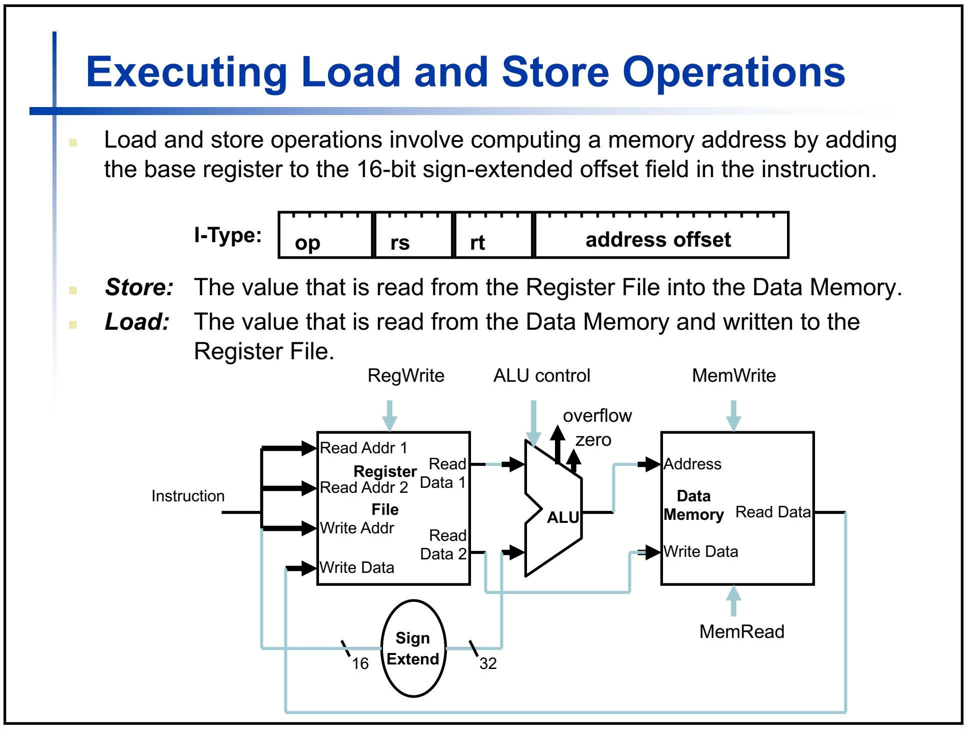
Executing R Format Operations

- R format operations (`add, sub, slt, and, or`)

	31	25	20	15	10	5	0
R-type:	**op**	**rs**	**rt**	**rd**	**shamt**	**funct**	

  - Perform operation (op and funct) on values in rs and rt.
  - Store the result back into the Register File into location rd.

- Note that the Register File is not written for every instruction (e.g. **`sw`**), so we need an explicit write control signal for the Register File.

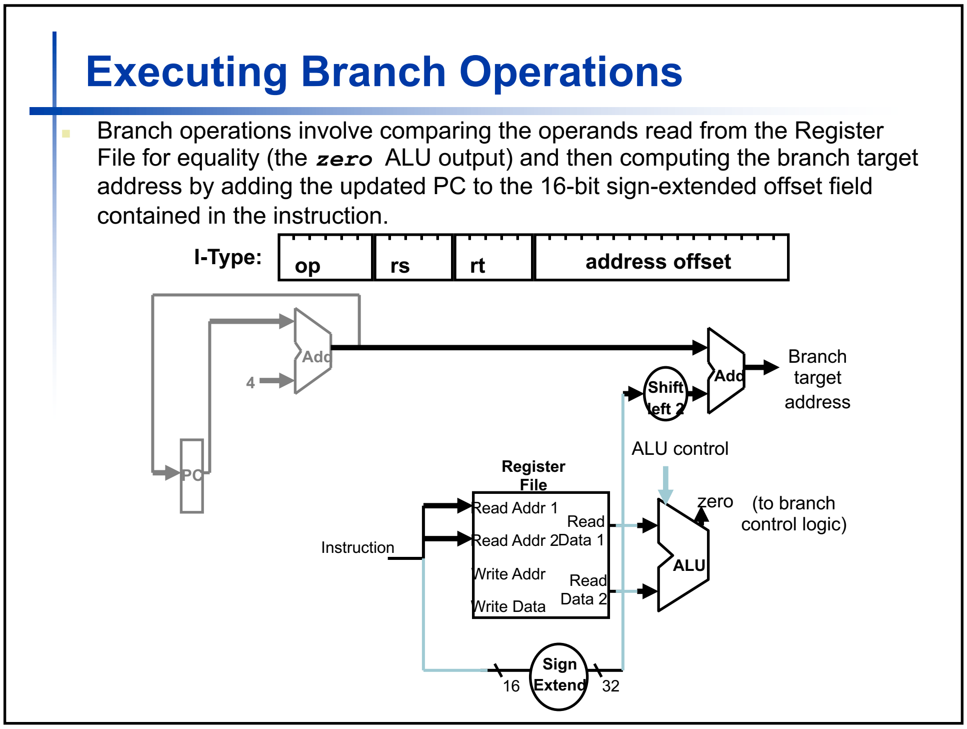


Executing Load and Store Operations

- Load and store operations involve computing a memory address by adding the base register to the 16-bit sign-extended offset field in the instruction.

I-Type:	op	rs	rt	address offset

- ***Store:*** The value that is read from the Register File into the Data Memory.
- ***Load:*** The value that is read from the Data Memory and written to the Register File.



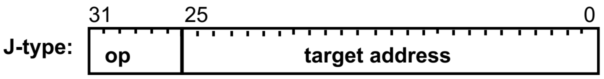
Executing Branch Operations

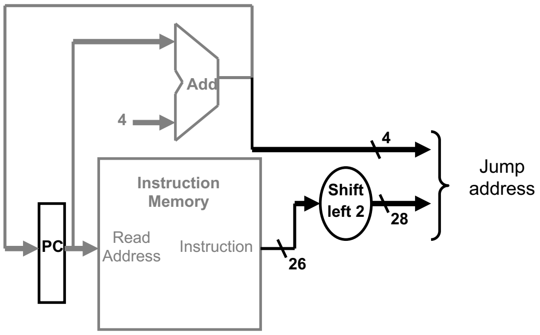
- Branch operations involve comparing the operands read from the Register File for equality (the `zero` ALU output) and then computing the branch target address by adding the updated PC to the 16-bit sign-extended offset field contained in the instruction.

I-Type:	op	rs	rt	address offset

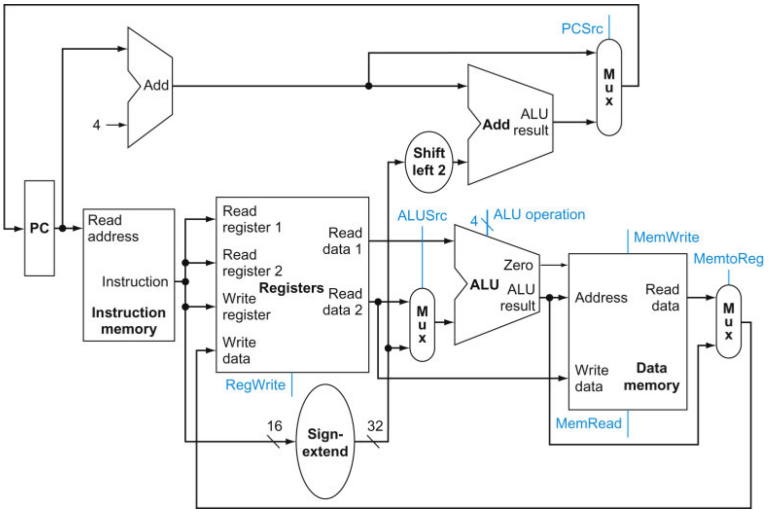
Executing Jump Operations

- A Jump operation involves replacing the lower 28 bits of the PC with the lower 26 bits of the fetched instruction shifted left by 2 bits.



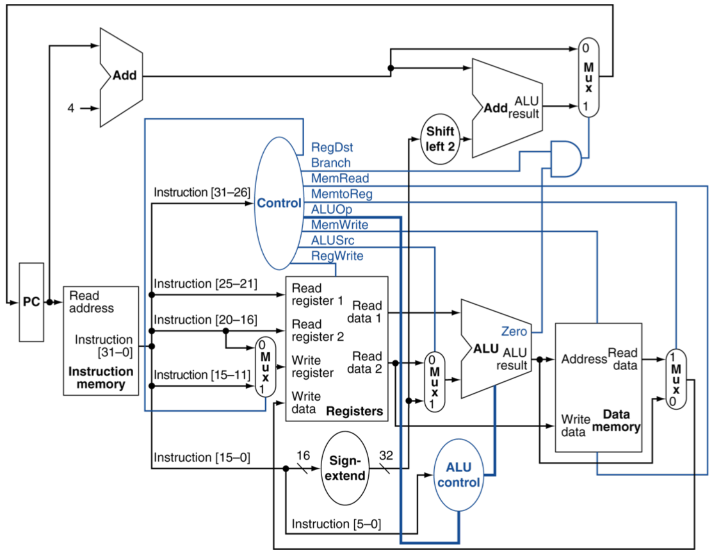


Overview With Control Signals



- The graphic above is referred to as the ***Datapath.***

More Detailed Datapath



Creating a Single Datapath from the Parts

- ***Single-cycle*** design – fetch, decode, and execute each instruction in one (and only one) clock cycle.
  - No datapath resource can be used more than once per instruction, so some must be duplicated (e.g., separate Instruction Memory and Data Memory, several adders).
  - Multiplexers needed at the input of shared elements, with control lines to do the selection.
  - Write signals to control writing to the Register File and Data Memory.
- Cycle time is determined by the length of the longest path, known as the ***critical path.***

Adding the Control

- The purpose of the ***controller*** is to control the flow of data.
- The controller determines which control signals to activate and when to activate them. The signals needed are dependant on the operation to be performed (Register, Branch or jump, or Memory read/write).

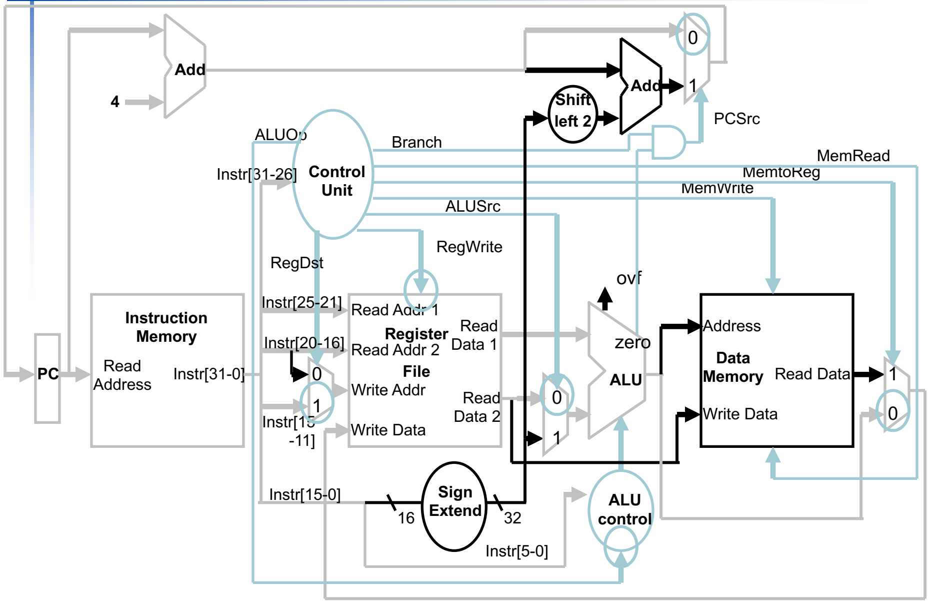
	31	25	20	15	10	5	0
R-type:	**op**	**rs**	**rt**	**rd**	**shamt**	**funct**	

	31	25	20	15	0
I-Type:	**op**	**rs**	**rt**	**address offset**	

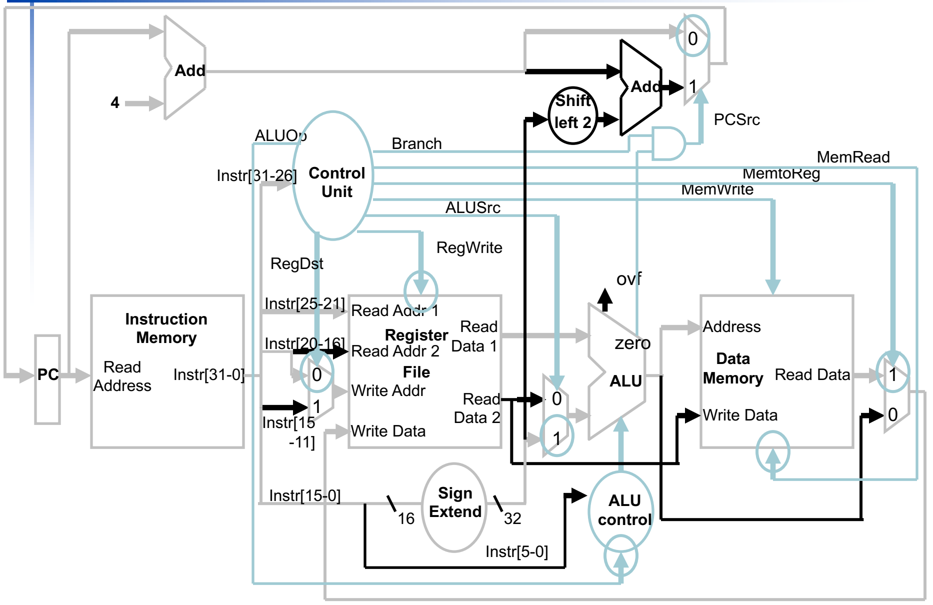
	31	25	0
J-type:	**op**	**target address**	

- Observations
  - The op field is always in bits 31-26.
  - The addresses of registers to be read are always specified by the rs (bits 25-21) and rt fields (bits 20-16); for lw and sw, rs is the base register.
  - The address of the register to be written is in one of two places – in rt (bits 20-16) for lw; in rd (bits 15-11) for R-type instructions.
  - The offset for beq, lw, and sw is always found in bits 15-0.

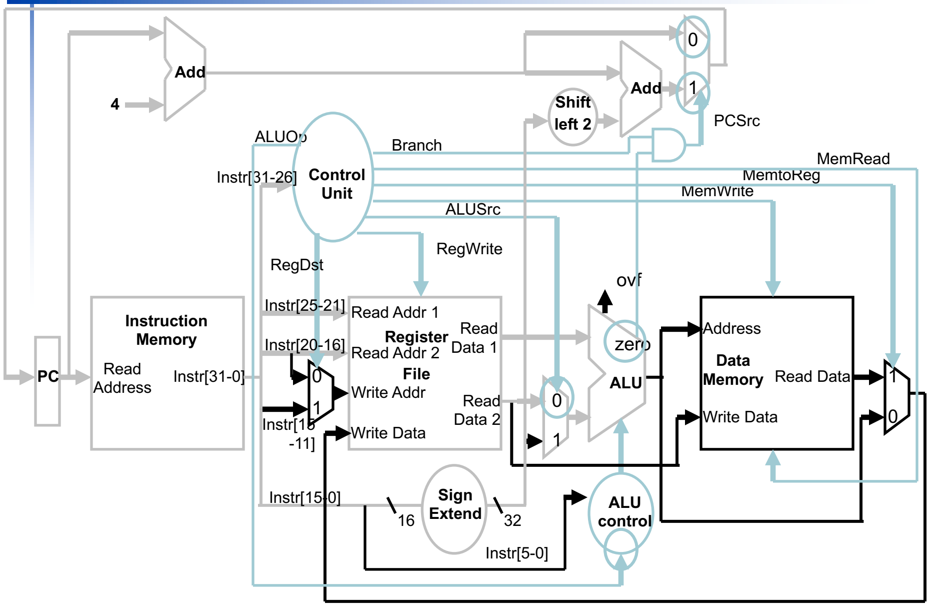
R-type Instruction



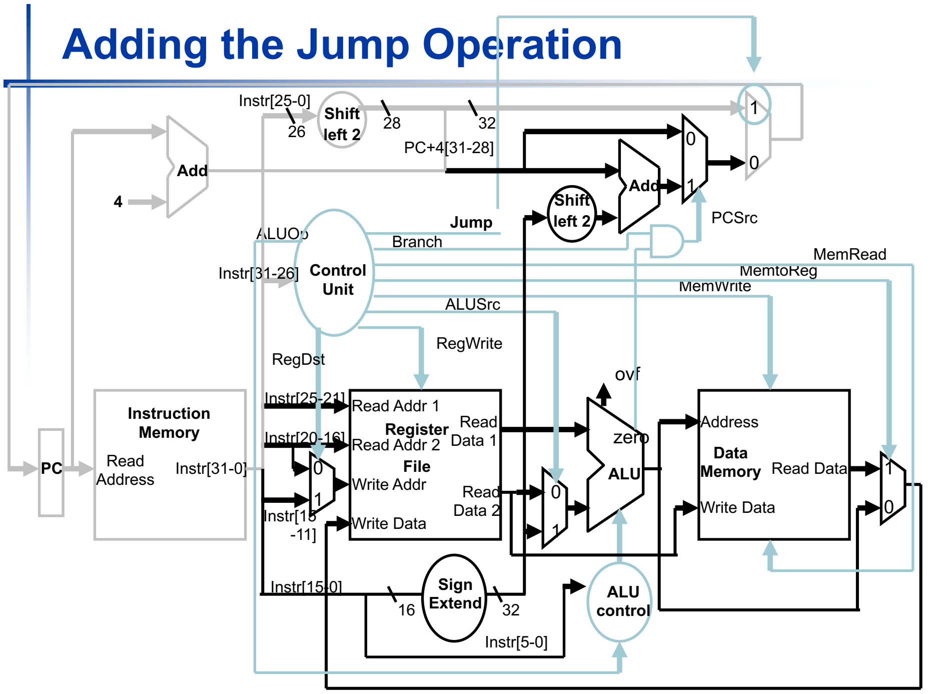
Load Word Instruction



Branch Instruction



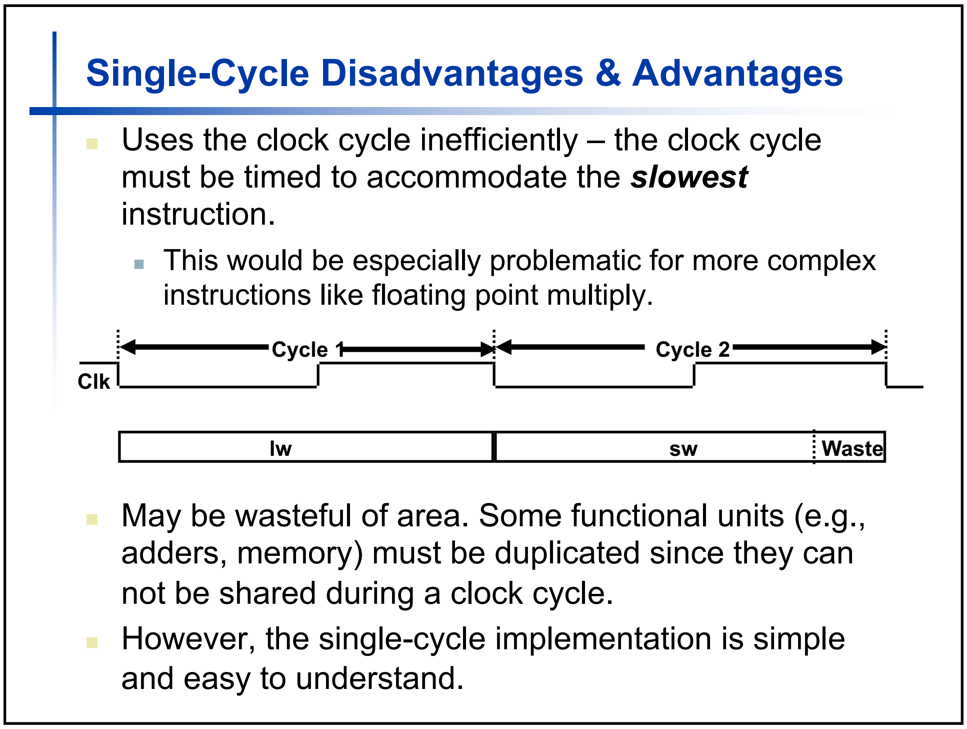
Adding the Jump Operation



Instruction Critical Paths

- Calculate the clock cycle time assuming negligible delays for multiplexers, control unit, sign extend, PC access, shift left 2, wires, setup and hold times:
  - Instruction and Data Memory (200 ps)
  - Register File access (reads or writes) (100 ps)
  - ALU and adders (200 ps)

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type	200	100	200		100	**600**
load	200	100	200	200	100	**800**
store	200	100	200	200		**700**
beq	200	100	200			**500**
jump	200					**200**

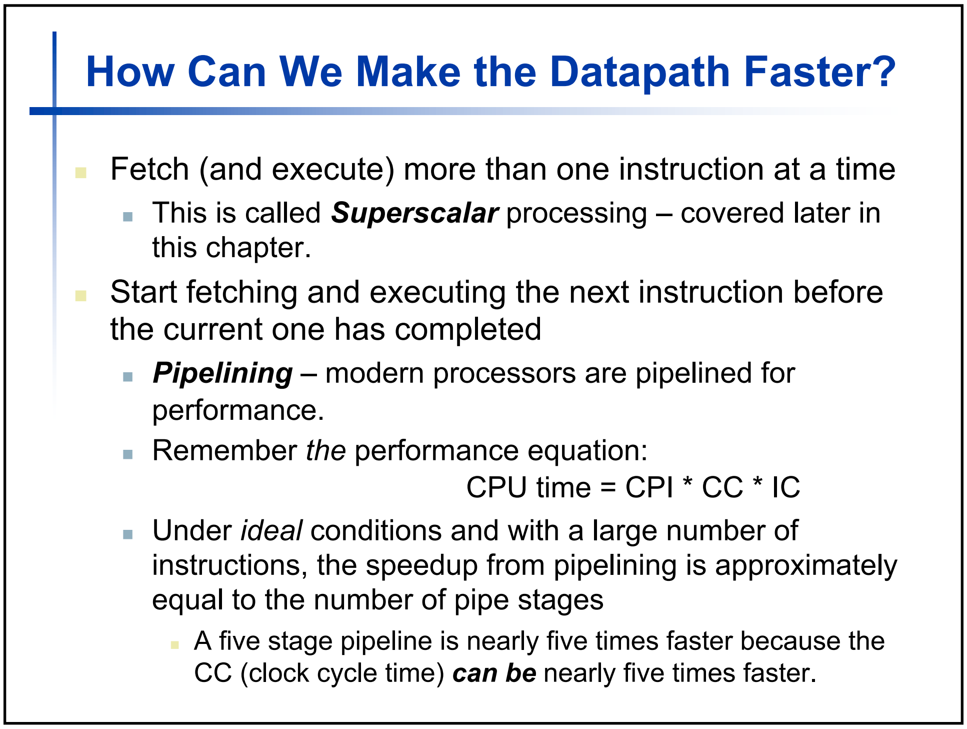


Single-Cycle Disadvantages & Advantages

- Uses the clock cycle inefficiently – the clock cycle must be timed to accommodate the ***slowest*** instruction.
  - This would be especially problematic for more complex instructions like floating point multiply.

Cycle 1
Cycle 2
Clk
lw
sw
Waste

- May be wasteful of area. Some functional units (e.g., adders, memory) must be duplicated since they can not be shared during a clock cycle.
- However, the single-cycle implementation is simple and easy to understand.

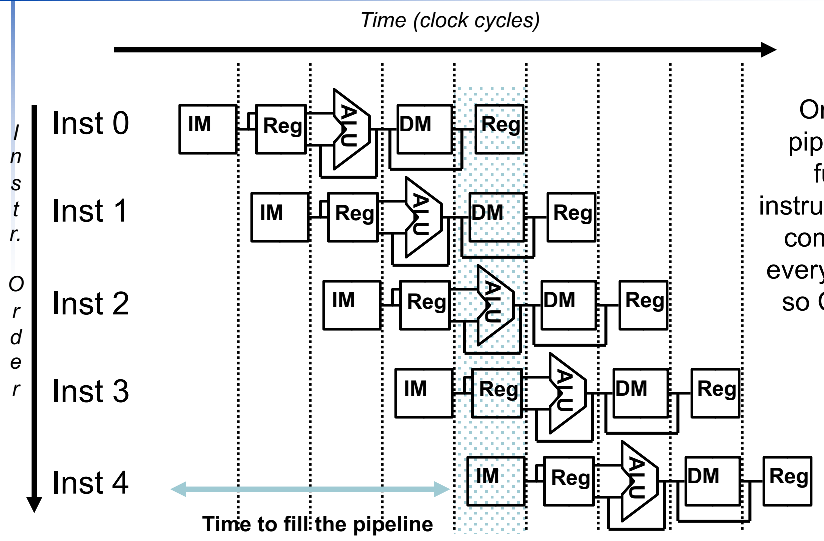


How Can We Make the Datapath Faster?

- Fetch (and execute) more than one instruction at a time
  - This is called ***Superscalar*** processing – covered later in this chapter.
- Start fetching and executing the next instruction before the current one has completed
  - ***Pipelining*** – modern processors are pipelined for performance.
  - Remember *the* performance equation:

    CPU time = CPI * CC * IC
  - Under *ideal* conditions and with a large number of instructions, the speedup from pipelining is approximately equal to the number of pipe stages
    - A five stage pipeline is nearly five times faster because the CC (clock cycle time) ***can be*** nearly five times faster.

Five Instruction Sequence



Once the pipeline is full, one instruction is completed every cycle, so CPI = 1

Summary

- Stored program execution.
- Single cycle execution
  - Data path design
    - R-type instructions.
    - I-type instructions.
    - J-type instructions.
- Next – pipelining.