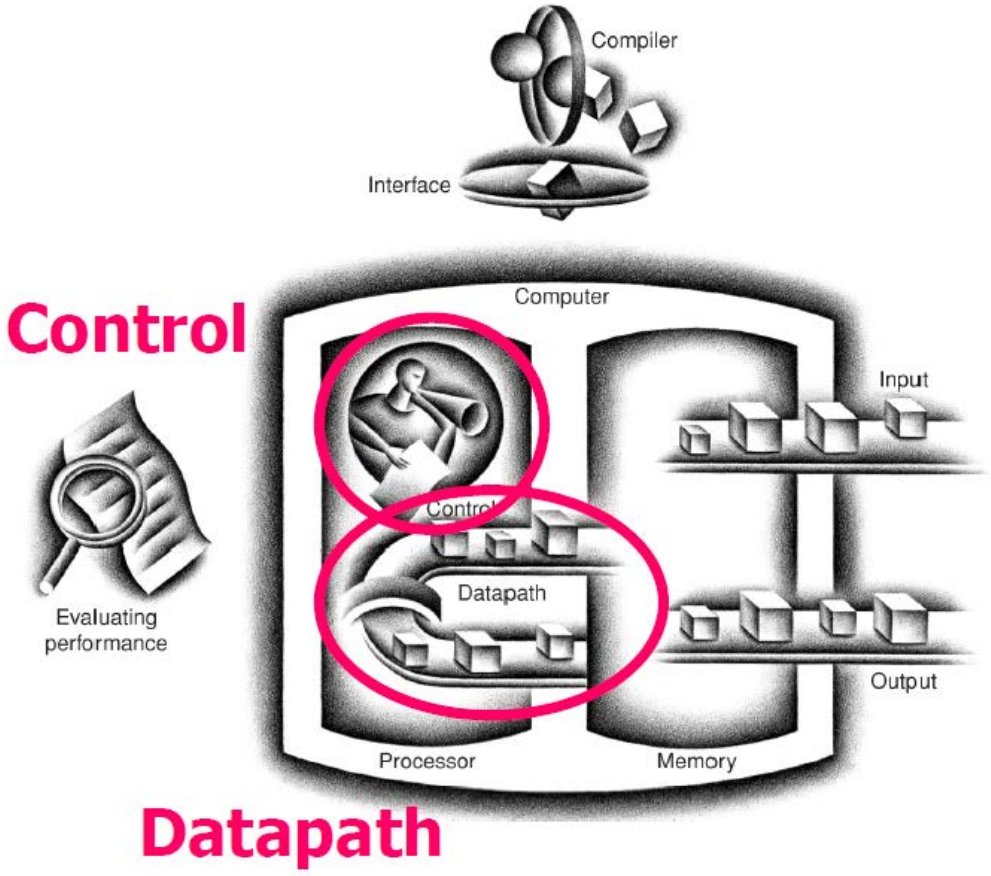
EE2011 Computer Organization

Lecture 7: The Processor - Datapath

Wen-Yen Lin, Ph.D.
Department of Electrical Engineering
Chang Gung University
Email: wylin@mail.cgu.edu.tw
May 2022

PPSoC
Parallel Processing and SoC Lab
Chang Gung University

The Processor: Datapath & Control



The Processor: Datapath & Control (Ch. 4.1)

- We're ready to look at an implementation of the MIPS
- Simplified to contain only:
  - memory-reference instructions: `lw, sw`
  - arithmetic-logical instructions: `add, sub, and, or, slt`
  - control flow instructions: `beq, j`
- Generic Implementation:
  - (PC) to supply instruction address
  - get the instruse the program counter uction from memory
  - read registers
  - use the instruction to decide exactly what to do
- All instructions use the ALU after reading the registers

  Why? memory-reference? arithmetic? control flow?
- After using ALU:
  - Memory-reference: access memory for load or store
  - Arithmetical-logical: write data from ALU back into register
  - Branches: change next instruction address based on comparison

Introduction

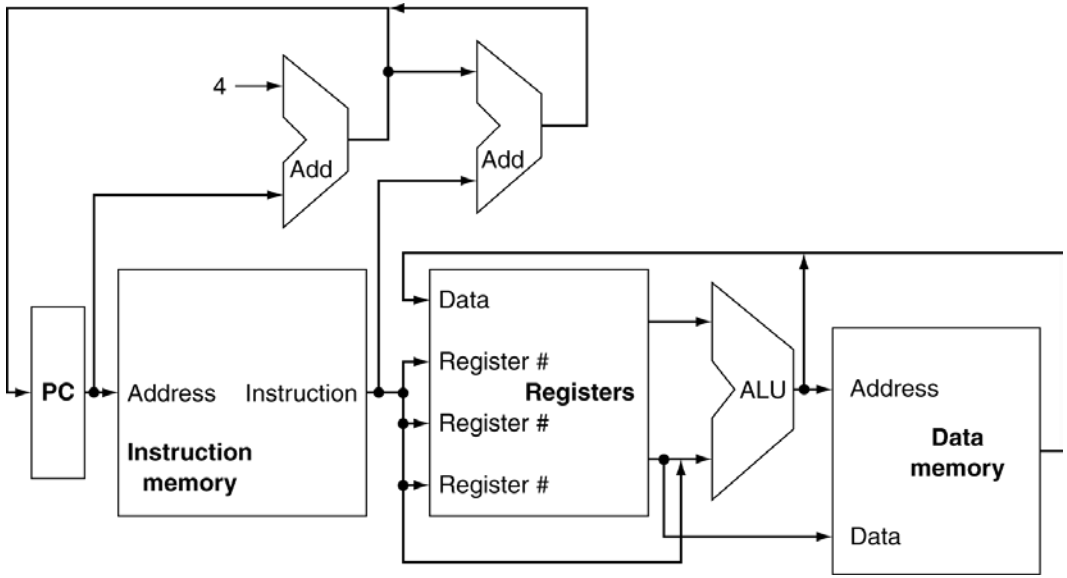
- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware
- We will examine two MIPS implementations
  - A simplified version
  - A more realistic pipelined version

Instruction Execution

- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC ← target address or PC + 4

CPU Overview ~ More Implementation Details (Fig. 4.1)

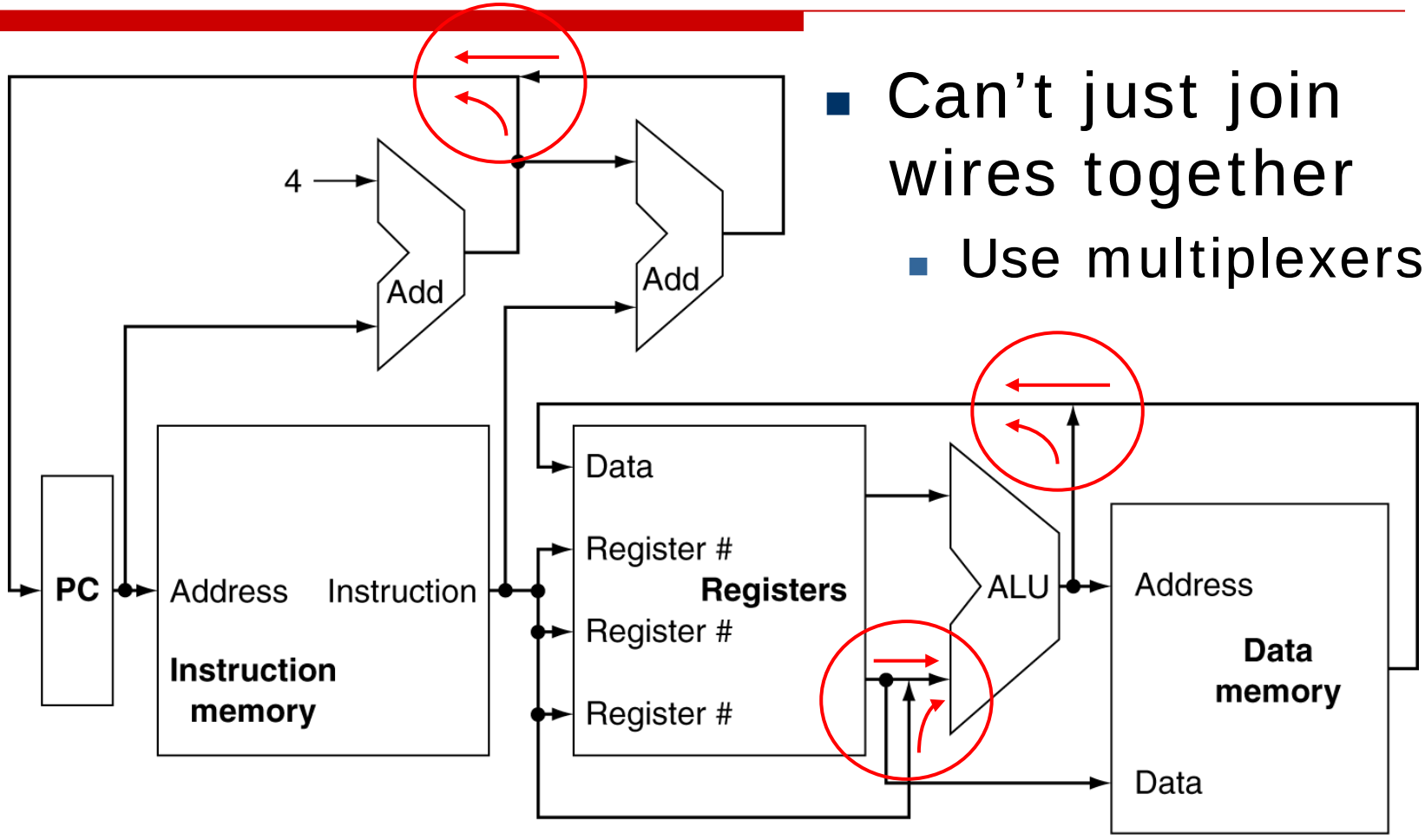
- Abstract / Simplified View:



Two types of functional units:

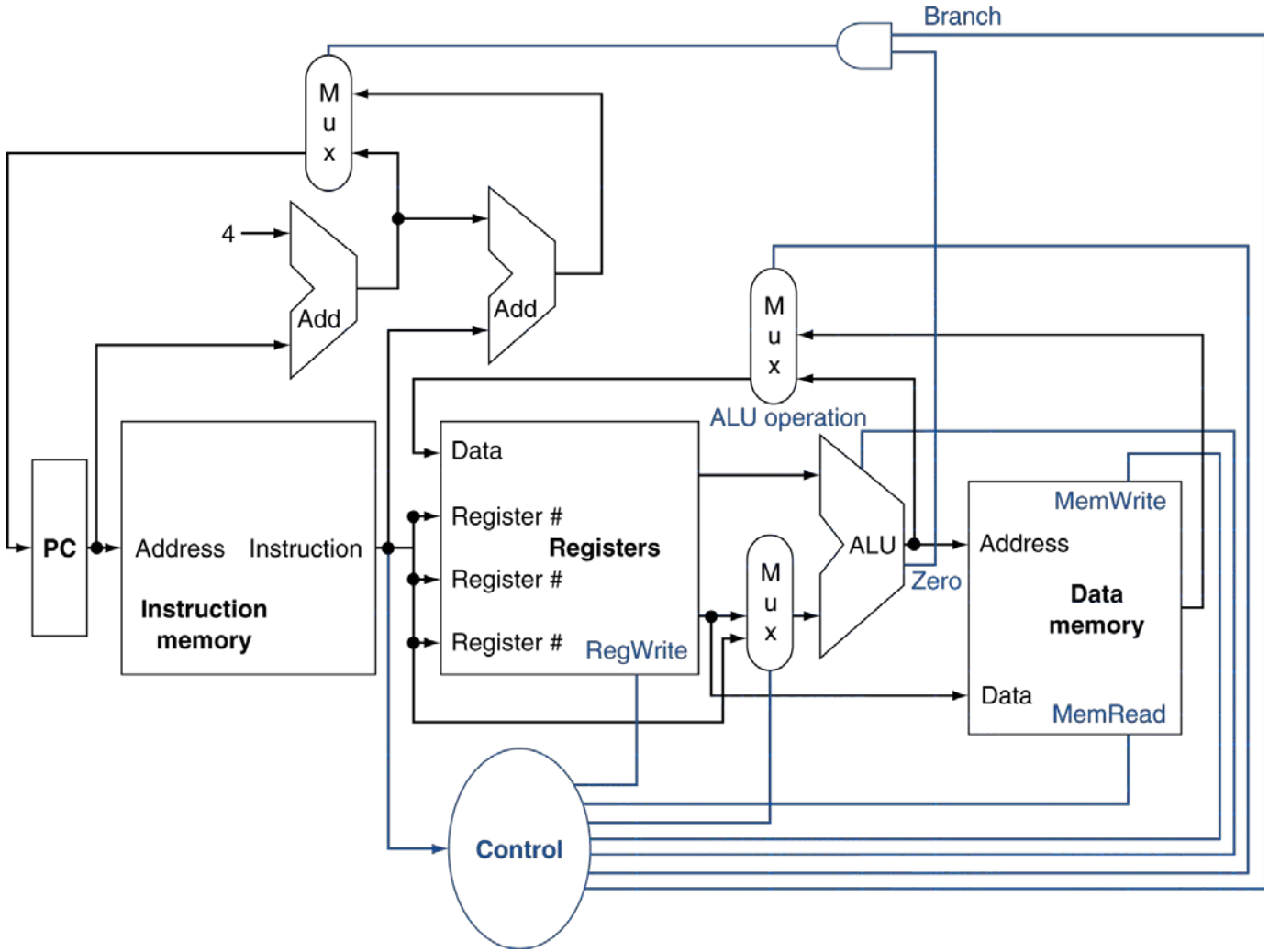
- ⇨ elements that operate on data values (combinational)
- ⇨ elements that contain state (sequential)

Multiplexers



- Can't just join wires together
  - Use multiplexers

Control (Fig. 4.2, To be discussed later ...)

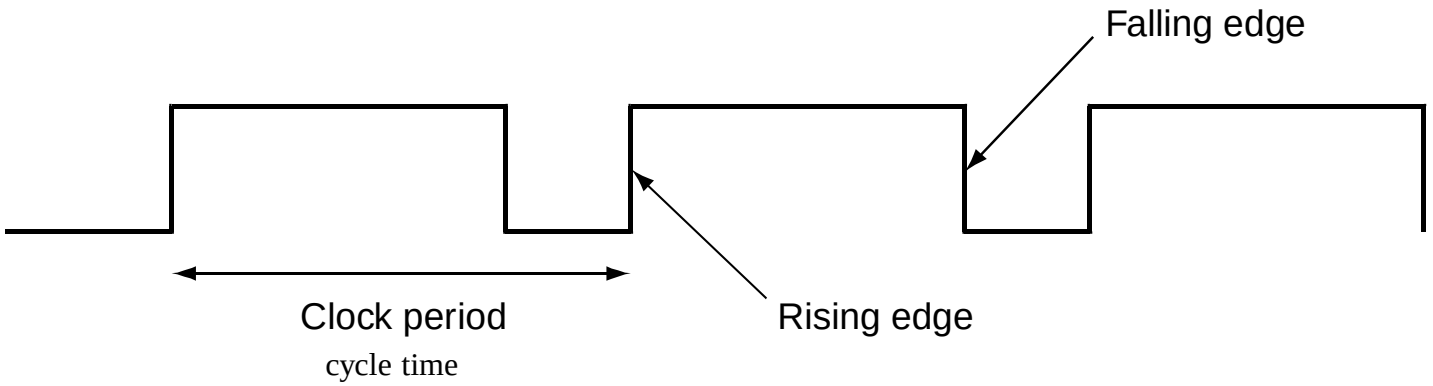


Logic Design Conventions (Ch. 4.2)

- Two types of logic elements in MIPS functional units
  - ⇨ Combinational: elements that operate on data values
  - ⇨ State or sequential: elements that contain state
    - A synchronous state element has at least
      - Two inputs
        - Data: value to be written
        - Clock: determining when the data value is written
      - One output:
        - Value that was written in an earlier clock cycle

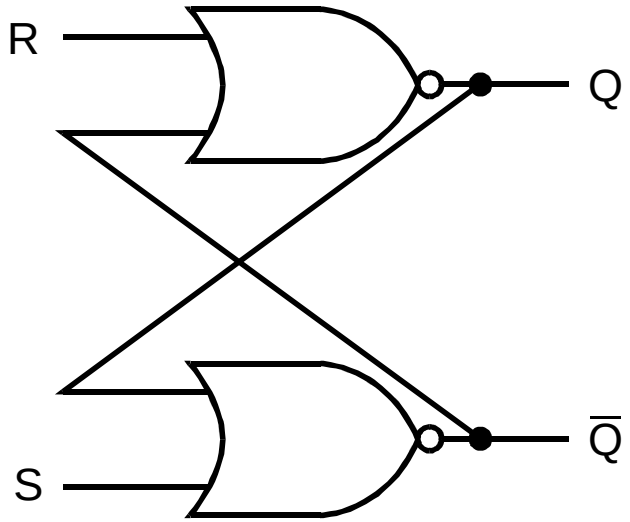
State Elements

- Unclocked/Asynchronous vs. Clocked/Synchronous
- Clocks used in synchronous logic
  - ⇨ when should an element that contains state be updated?



An Unclocked/Asynchronous state element

- The set-reset latch
  - ⇨ output depends on present inputs and also on past inputs (feedbacks)



Clocking Methodology (p. 261)

- Defines when signals can be read and written.
  - ⇨ Additional control signals may required to direct the operations, e.g. write enable, etc.
- Edge-triggered clocking
  - ⇨ Any values stored in the sequential logic elements are updated only on a clock edge
- Non-edge triggered clocking
  - ⇨ Any values stored in the sequential logic elements are updated when clock is asserted.
- Combinational logic
  - ⇨ Has its inputs coming from a set of state elements and its outputs written into a set of state elements
  - ⇨ The inputs are values that were written in a previous clock cycle, while the outputs are values that can be used in following clock cycle.

Latches and Flip-flops

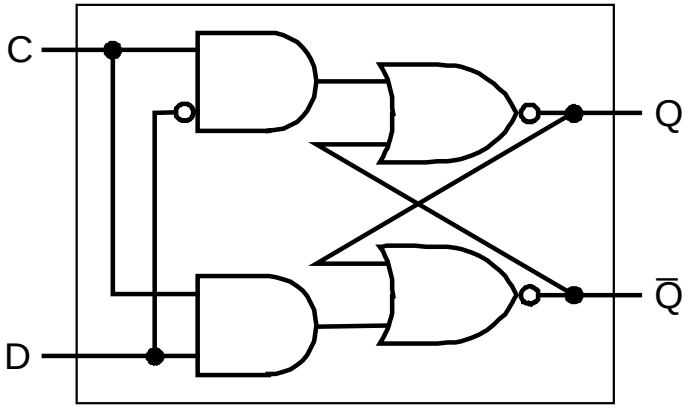
- Output is equal to the stored value inside the element (don't need to ask for permission to look at the value)
- Change of state (value) is based on the clock
- Latches: whenever the inputs change, and the clock is asserted
- Flip-flop: state changes only on a clock edge (edge-triggered methodology)

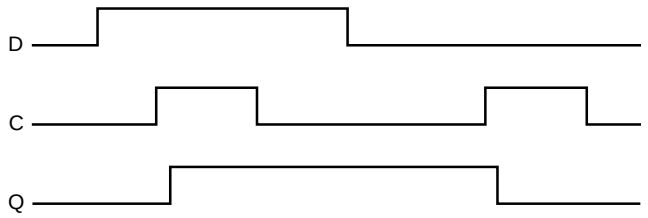
"logically true",
— could mean electrically low

A clocking methodology defines when signals can be read and written
— wouldn't want to read a signal at the same time it was being written

D-latch

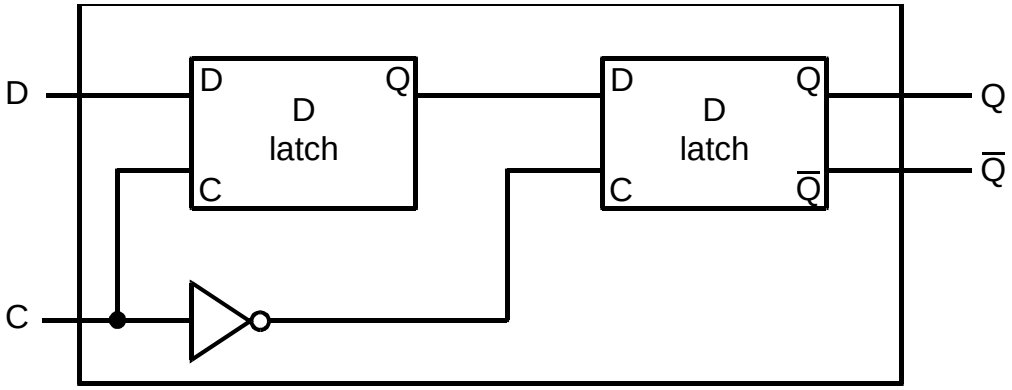
- Two inputs:
  - the data value to be stored (D)
  - the clock signal (C) indicating when to read & store D
- Two outputs:
  - the value of the internal state (Q) and it's complement

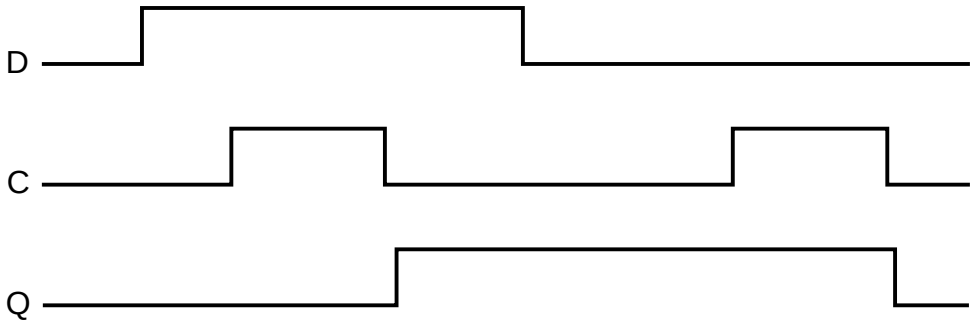




D flip-flop

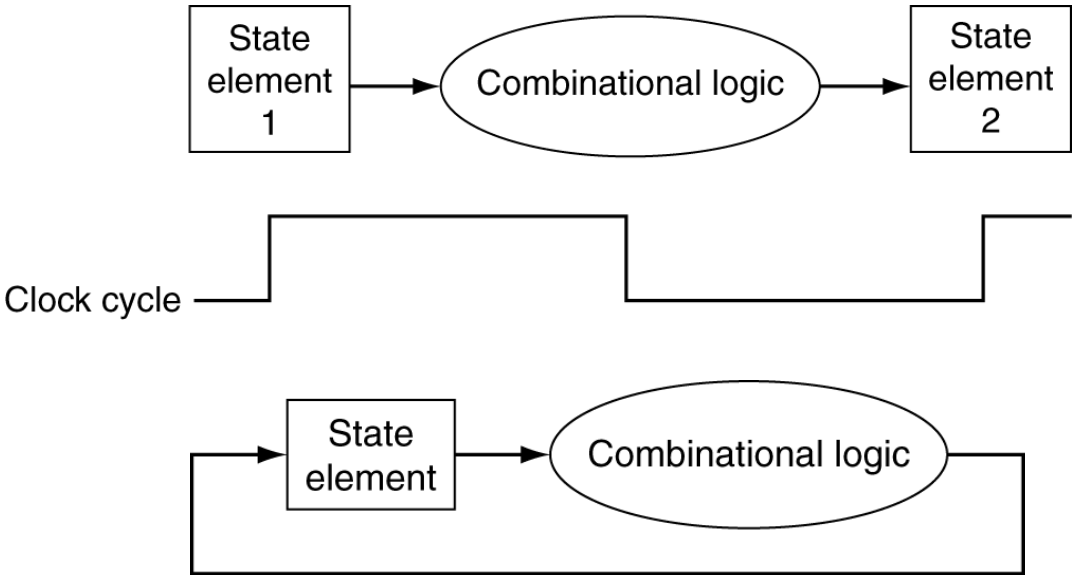
- Output changes only on the clock edge





Our Implementation (Fig. 4.3 & Fig. 4.4)

- An edge triggered methodology
- Typical execution:
  - ⇨ read contents of some state elements,
  - ⇨ send values through some combinational logic
  - ⇨ write results to one or more state elements



Building a Datapath (Ch. 4.3)

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

Simple Implementation

- Include the functional units we need for each instruction

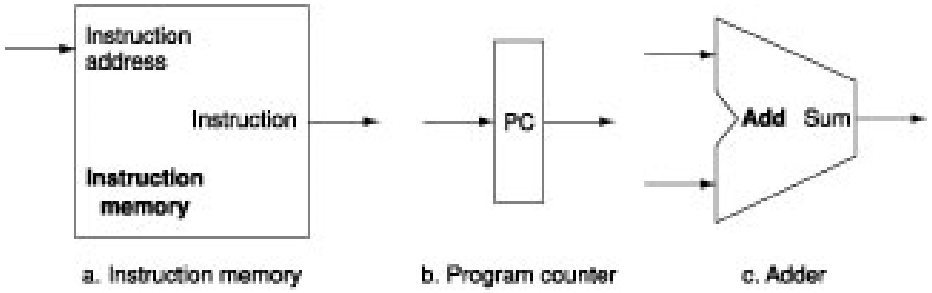


Figure 4.5 Instruction Fetching

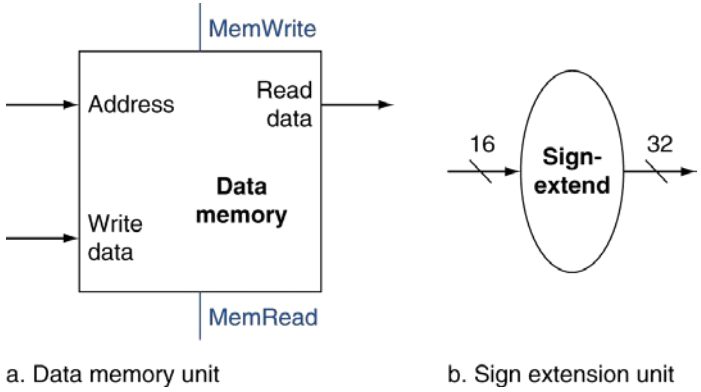


Figure 4.8 Load and Store operations

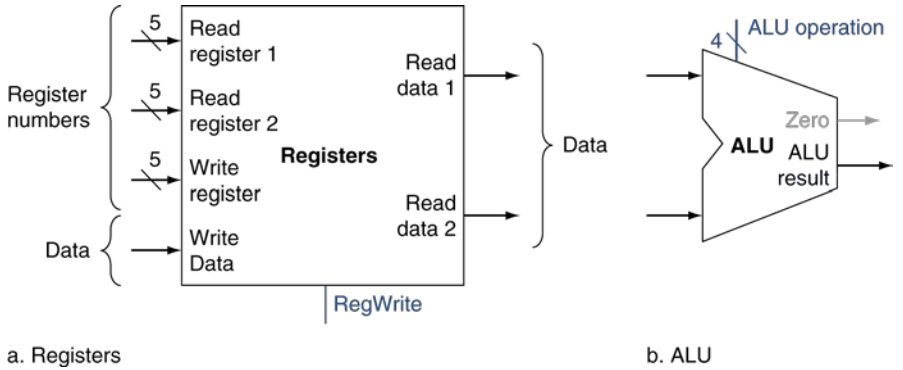
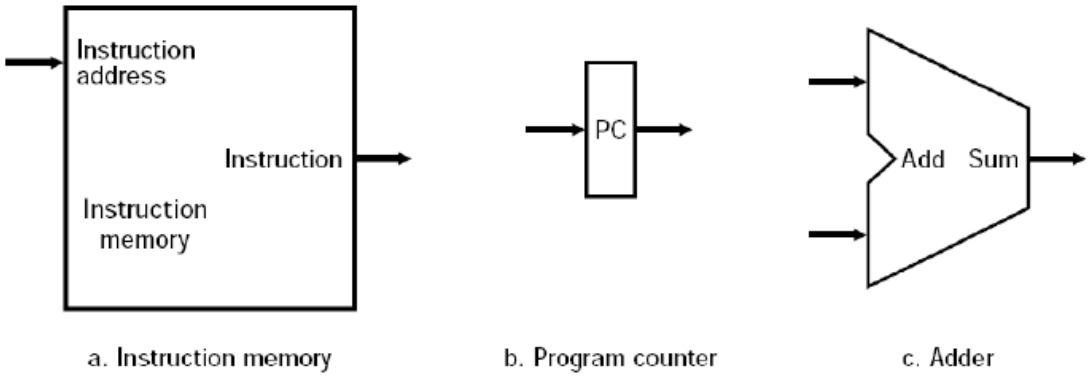


Figure 4.7 R-format ALU operations

Why do we need this stuff?

Building a Datapath

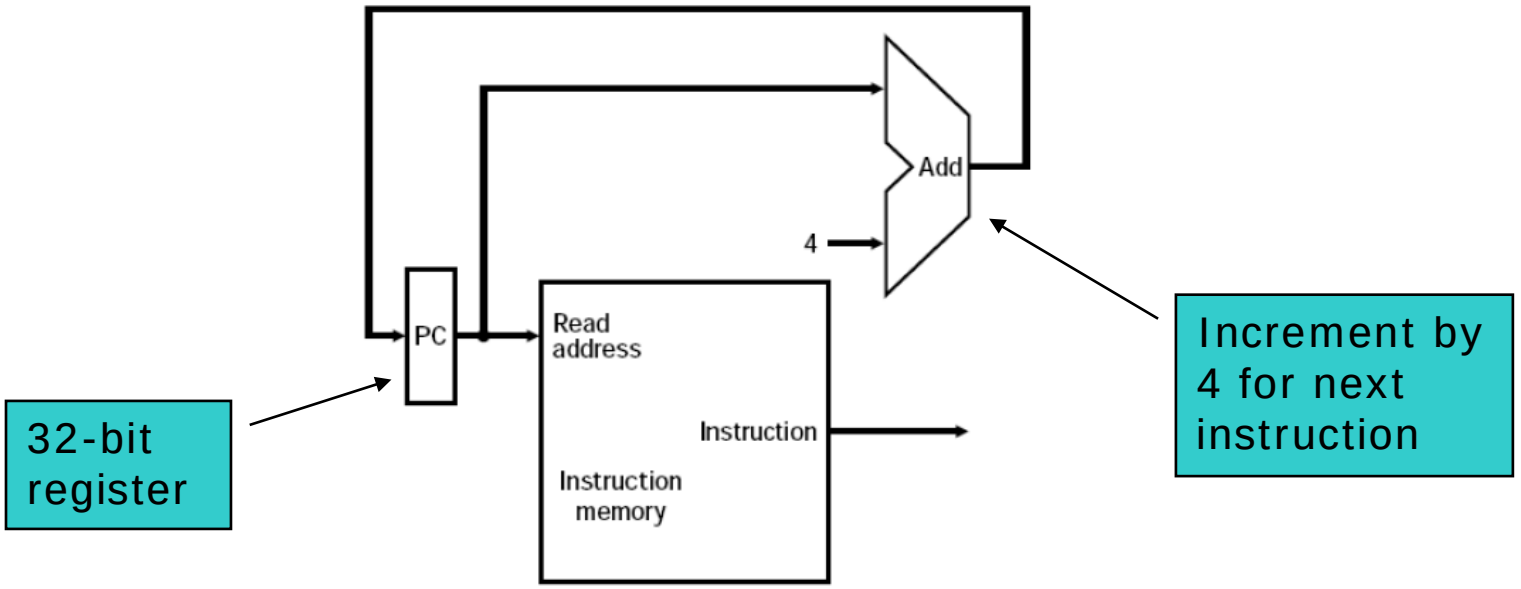
- The major components required to execute each class of instructions
  - Instruction memory
    - A state element, storing the instructions of a program
    - Holds and supplies instructions given an address
  - Program counter (PC)
    - A state element
    - Keeps the address of the instruction being executed
  - Adder
    - Combinational element
    - Increments the PC to the address of the next instruction
    - Can be built from the ALU by wiring the control lines



a. Instruction memory b. Program counter c. Adder

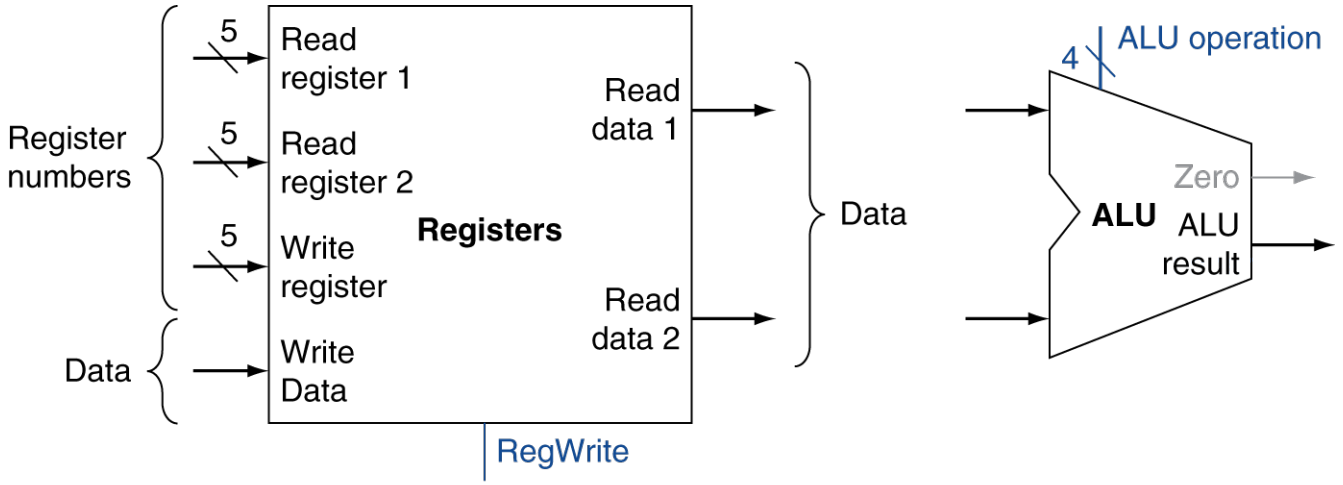
Instruction Fetching (Fig. 4.6)

- To execute an instruction, start by
  - ⇨ Fetching the instruction from memory.
  - ⇨ Incrementing PC to point to the next instruction, 4 bytes after.



R-Format Instructions (Fig. 4.7)

- Read two register operands
- Perform arithmetic/logical operation
- Write register result



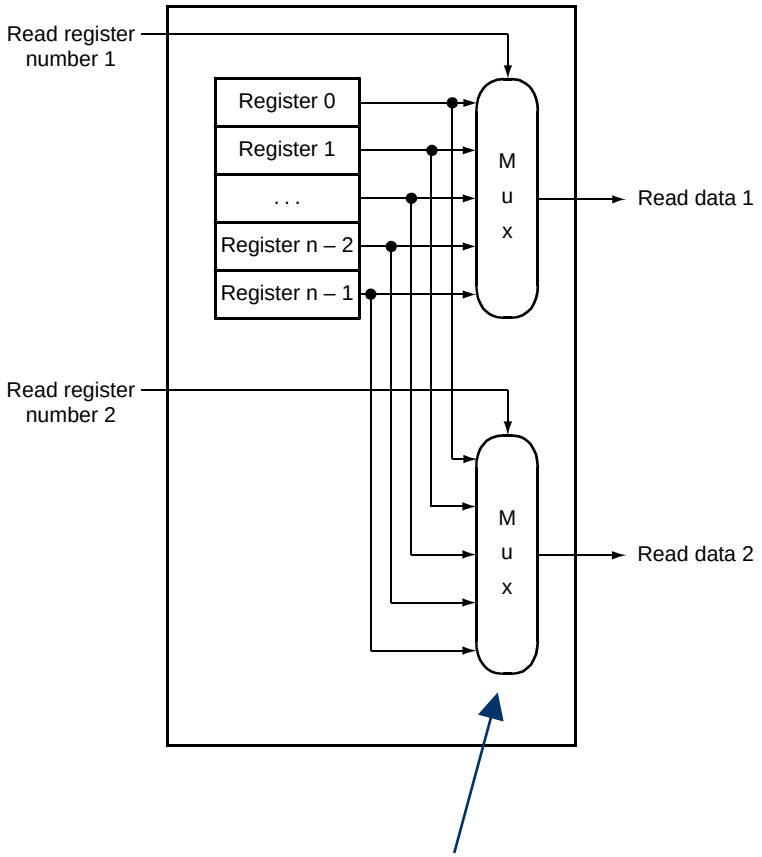
Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	32 bits
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction

Register files (p. 264)

- A collection of 32 32-bit registers in which any register can be read or written by specifying the number of the register.
- For R-format instruction, we have to read two data words from, and write one data word into the register file.
- To read a word from the register file, we need
  - An input to specify the register number to be read.
  - An output to carry the value that has been read.
- To write a word into the register file, we need
  - An input to specify the register number to be written
  - Any input to supply the data to be written.

Register File Implementation - Read

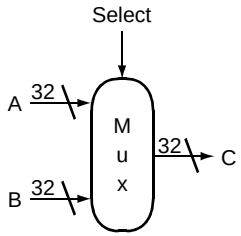
- Built using D flip-flops

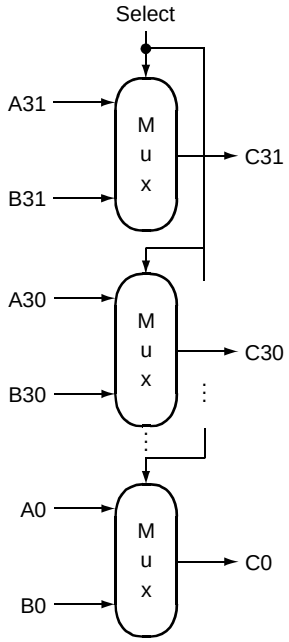


Do you understand? What is the "Mux" above?

Abstraction

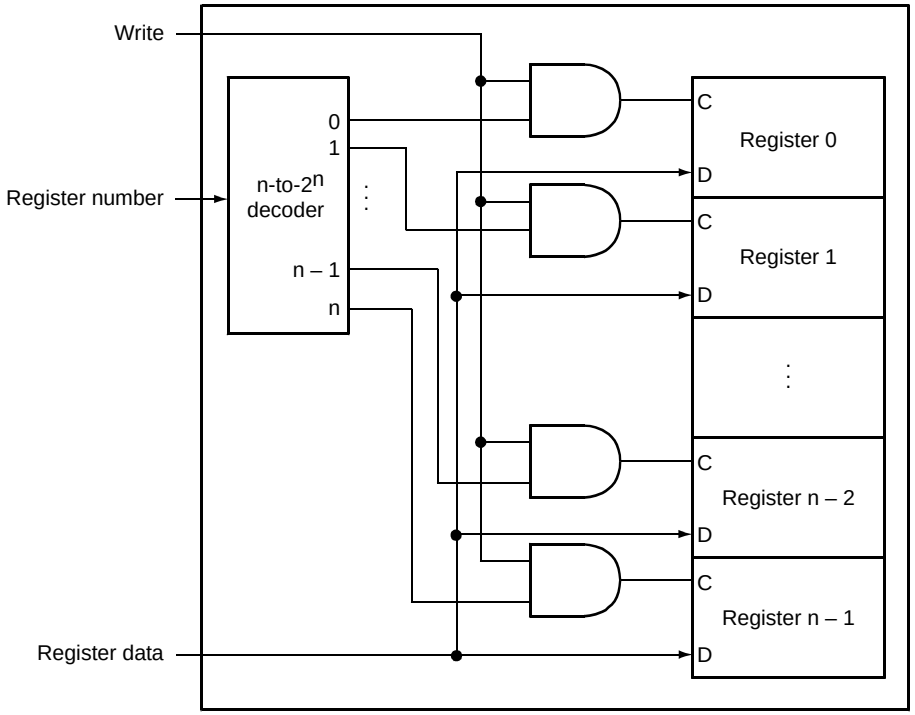
- Make sure you understand the abstractions!
- Sometimes it is easy to think you do, when you don't



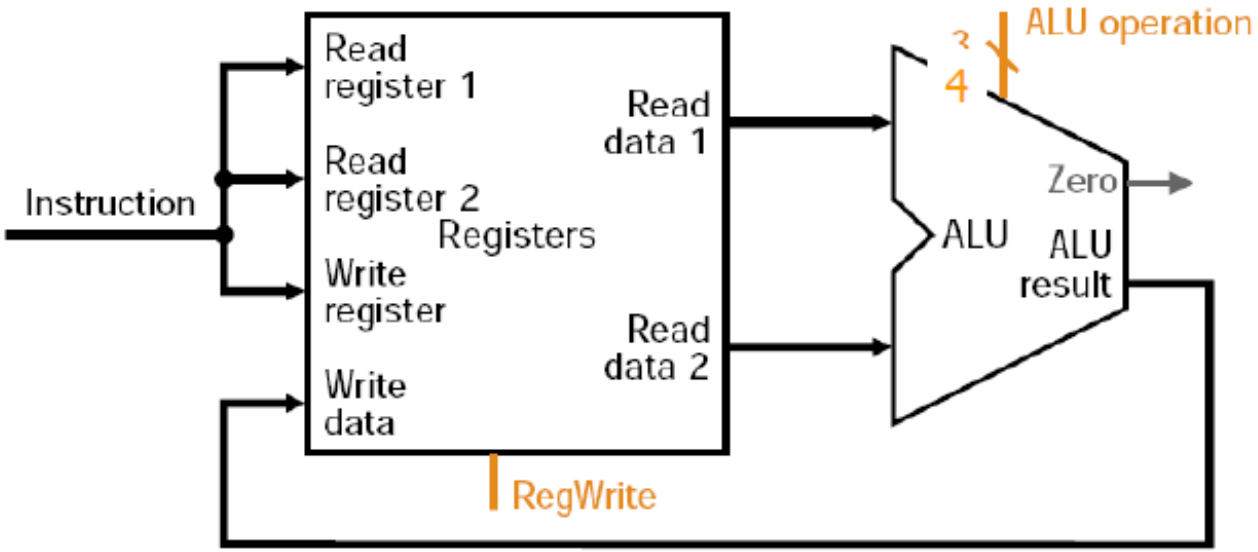


Register File Implementation - Write

- Note: we still use the real clock to determine when to write



Datapath for R-format Instructions



Name	Fields					
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R-format	op	rs	rt	rd	shamt	funct

I-format Instructions – Load & Store

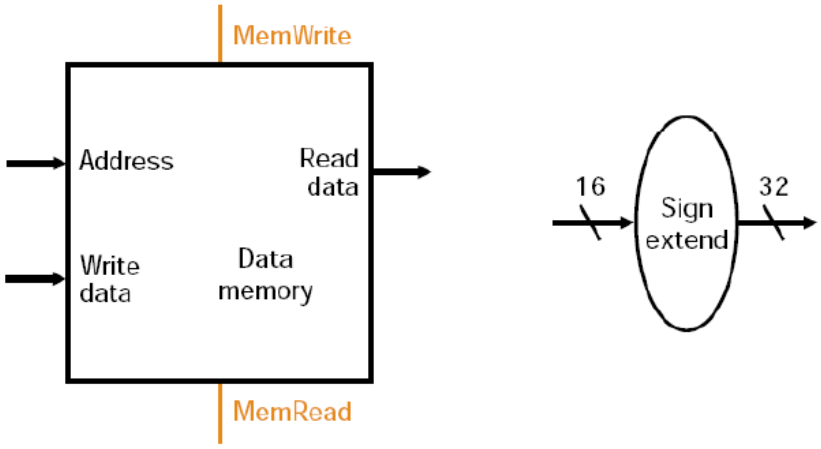
- I-format instructions
  - Load: lw rt, offset(rs)
  - Store: sw rt, offset(rs)
  - The memory address is obtained by adding the base register, rs, to the 16-bit signed offset field.

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	32 bits
I-format	op	rs	rt	address/immediate			Transfer, branch, imm.

Data Memory and Sign-Extension Units

- Data memory unit
  - A state element to read from (load) or write to (store)
  - Edge-triggered for writes

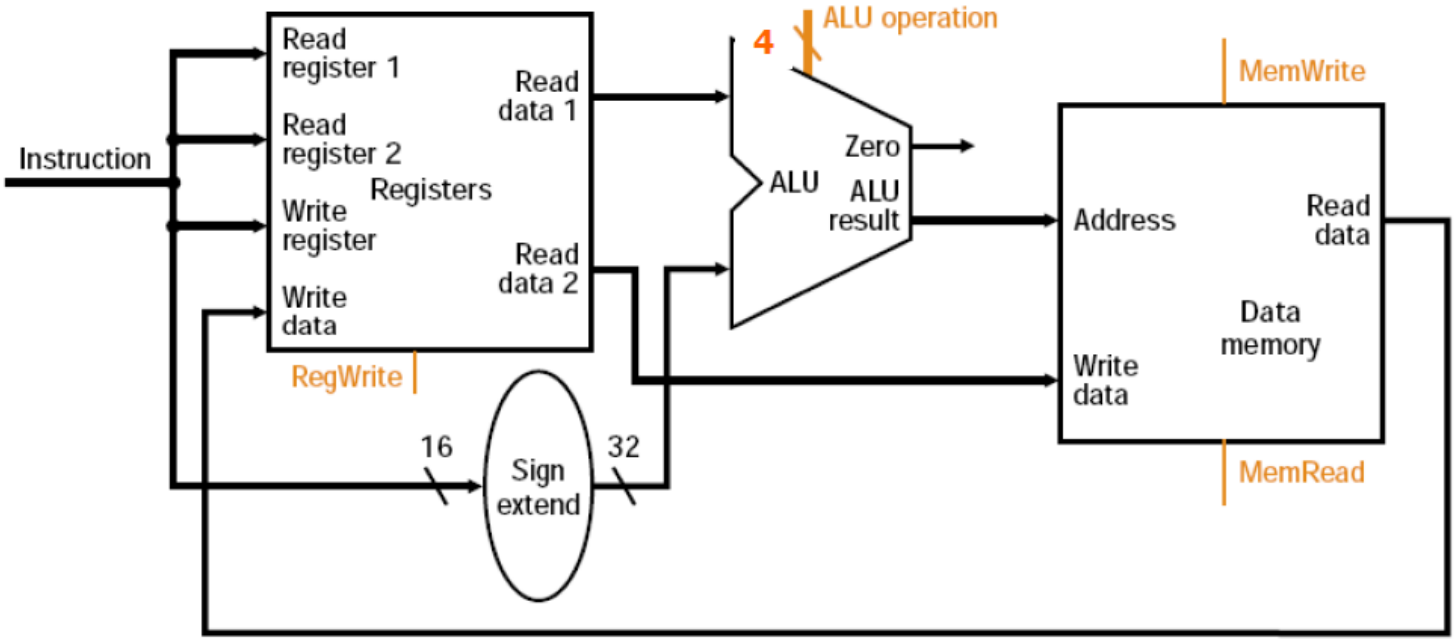
- Sign-extension unit
  - Combinational element, to sign-extend the 16-bit offset field to a 32-bit signed value



a. Data memory unit

b. Sign-extension unit

Datapath for I-format Instructions (Fig. 4.10)



Name	Fields					
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
I-format	op	rs	rt	address/immediate		

I-format Instructions – Branch (p. 266)

- beq rs, rt, offset
  - ⇨ Two registers are compared for equality
  - ⇨ A 16-bit offset is used to compute the branch target address relative to the PC

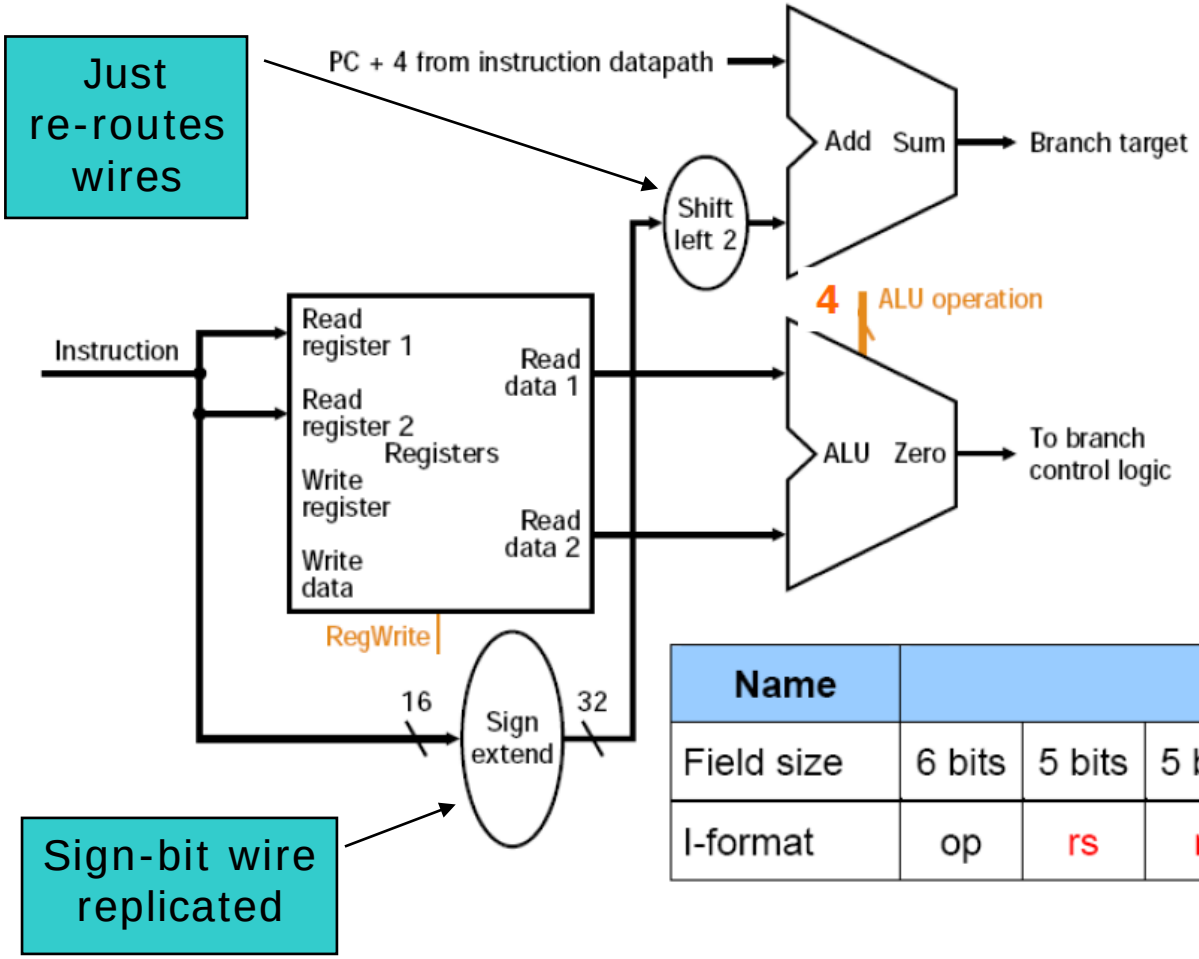
- Two details about branch instructions
  - ⇨ The base for the branch address calculation is the address of the instruction following the branch instruction, i.e. PC+4
  - ⇨ The offset is shifted left 2 bits to convert to a byte offset.

Name	Fields					
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
I-format	op	rs	rt	address/immediate		

Branch Datapath

- Two operations for the branch datapath
  - Compute the branch target address
  - Compare the register contents
- To perform the compare
  - Set the ALU operation to subtraction
  - Decide if the two register contents are equal by zero output
- If the condition is
  - True (branch taken): the branch target address becomes the new PC
  - Not True (branch not taken): the incremented PC replaces the current PC.

Branch Datapath (Fig. 4.9)



Name	Fields					
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
I-format	op	rs	rt	address/immediate		

J-Format Instructions - Jump

- Replace the lower 28 bits of the PC with the lower 26 bits of the instruction shifted left by 2 bits (word address to byte address)
- Leave the upper 4 bits of the PC unchanged.

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	32 bits
J-format	op	target address					Jump instruction

Composing the Elements (p. 268)

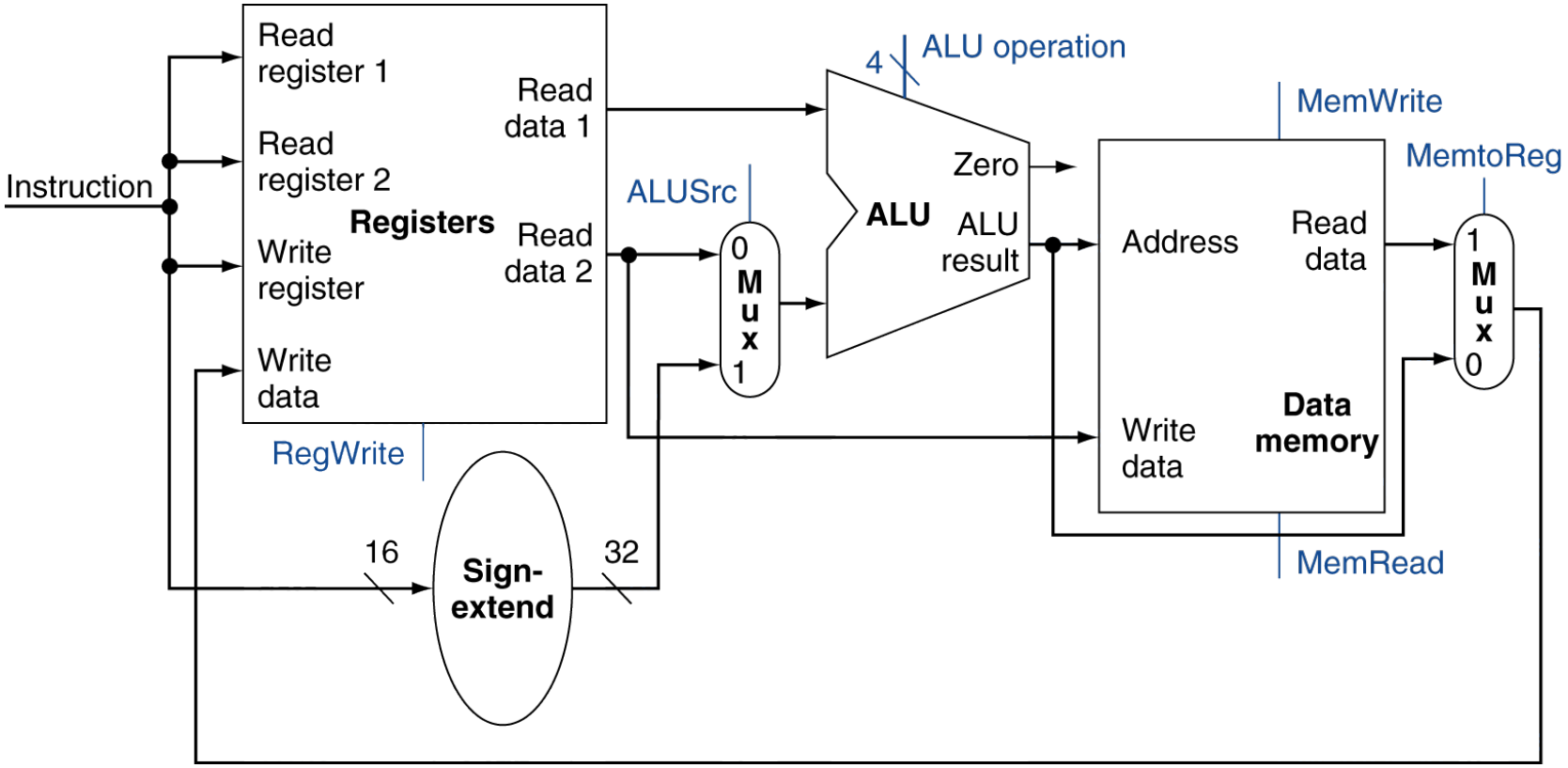
- First-cut data path does an instruction in one clock cycle
  - ⇨ Each datapath element can only do one function at a time
  - ⇨ Hence, we need separate instruction and data memories
- Use multiplexers where alternate data sources are used for different instructions

Single-Cycle Datapath

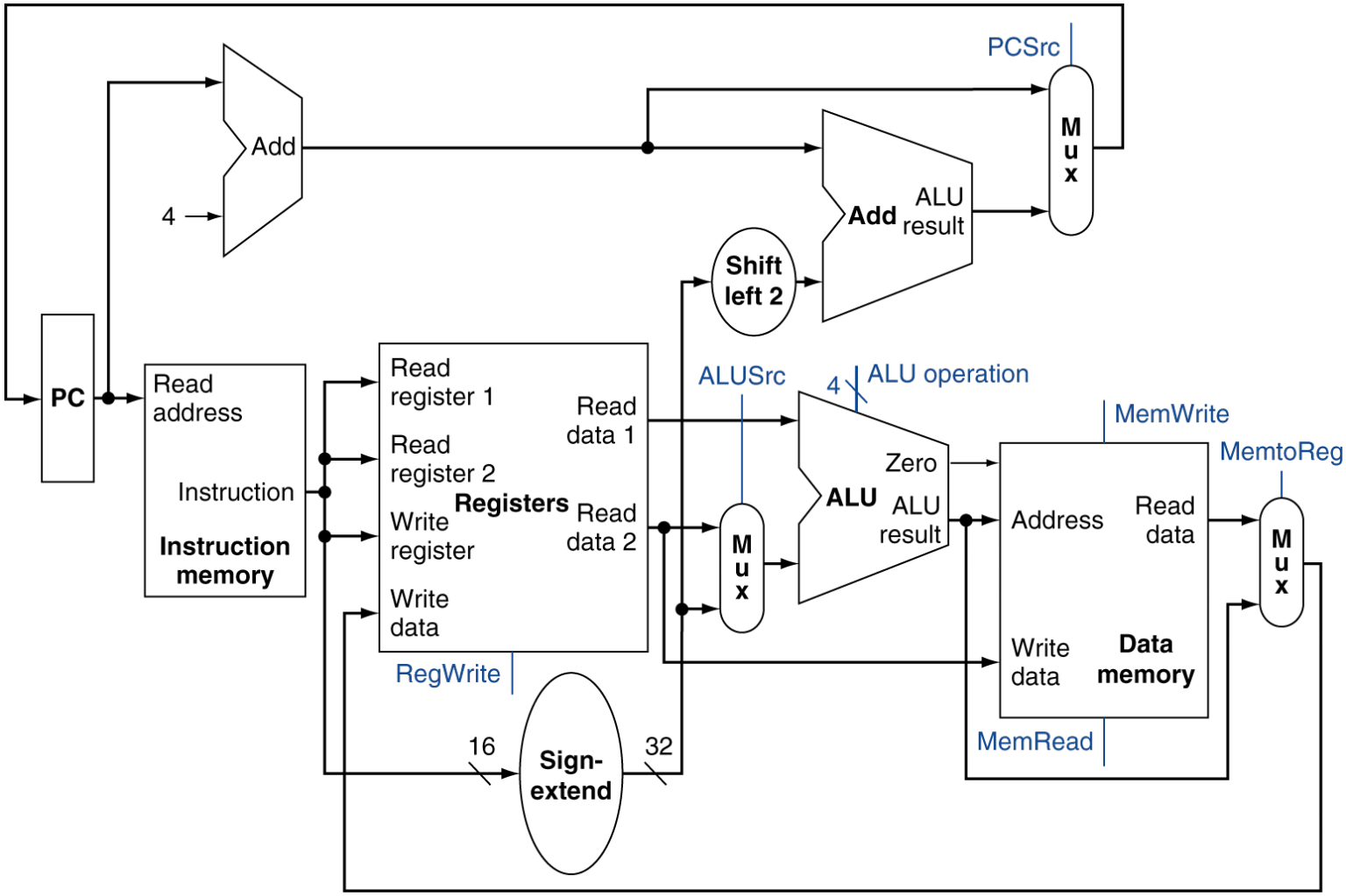
- All instructions are executed in one clock cycle.
- No datapath resource can be used more than once per instruction
- An element needed more than once must be duplicated
  - ⇨ Two memories: one for instruction and one for data
- Some functional units can be shared by different instructions
  - ⇨ We need to allow multiple connections to the input of an element and have a control signal select among the inputs.
  - ⇨ The selection is commonly done with a multiplexor

R-Type/Load/Store Datapath (Fig. 4.10)

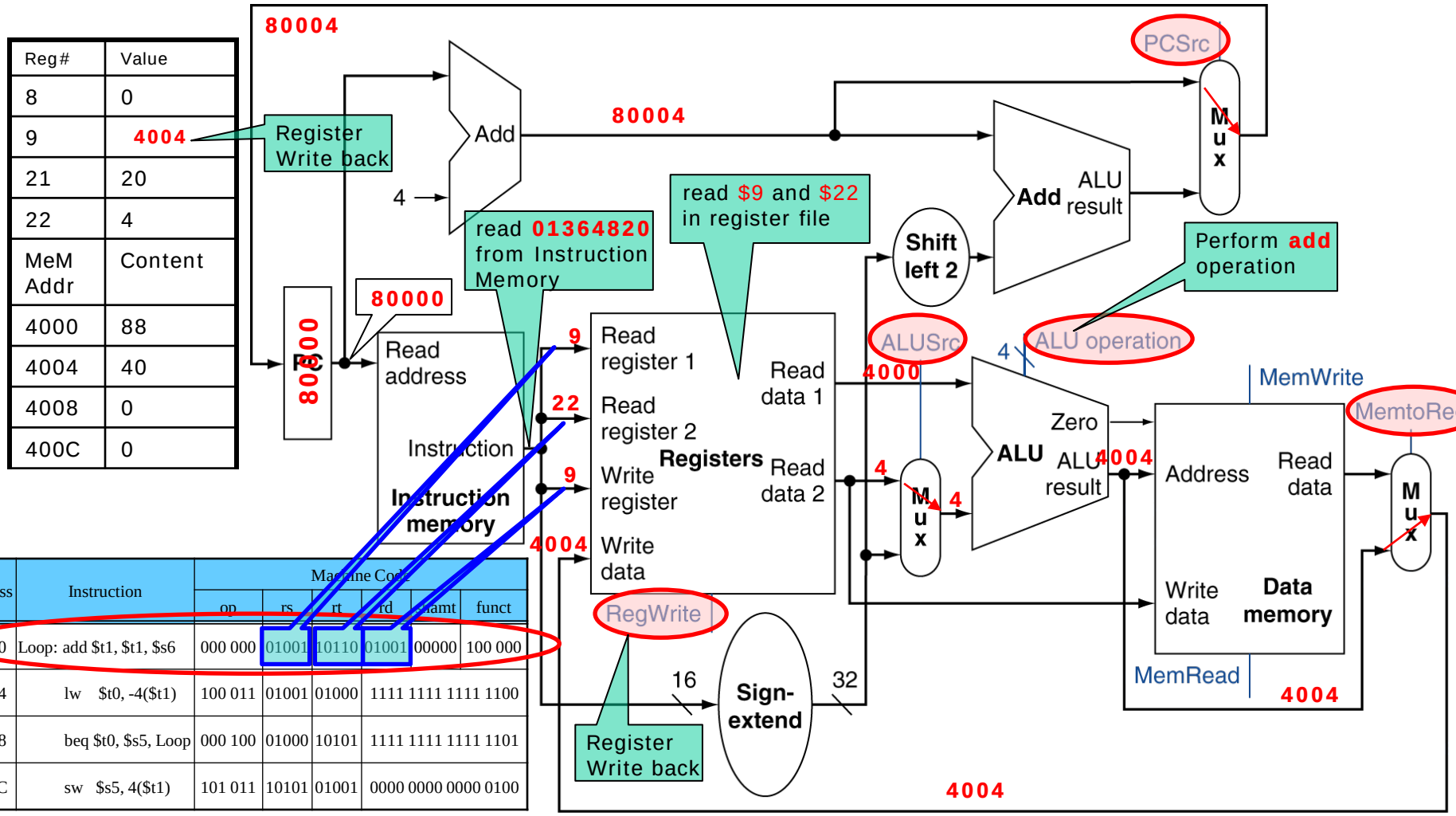
- Use multiplexors to stitch them together



Putting All Together for the Datapath (Fig. 4.11)



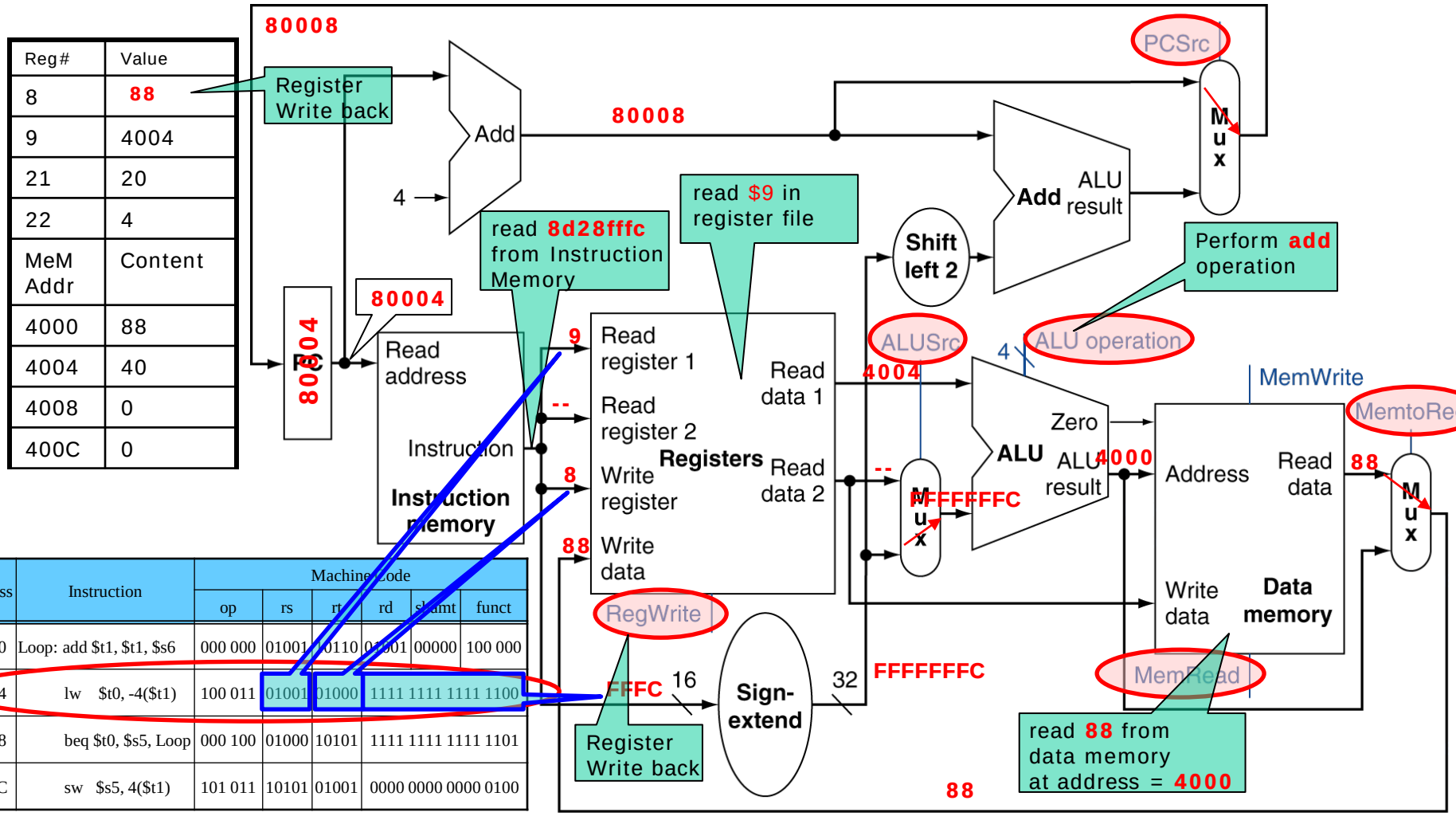
Example ~ How data flow through datapath (R-type Instruction: Add)



Reg#	Value
8	0
9	4004
21	20
22	4
MeM Addr	Content
4000	88
4004	40
4008	0
400C	0

Address	Instruction	Machine Code					
		op	rs	rt	rd	shamt	funct
80000	Loop: add $t1, $t1, $s6	000 000	01001	10110	01001	00000	100 000
80004	lw $t0, -4($t1)	100 011	01001	01000	1111 1111 1111 1100		
80008	beq $t0, $s5, Loop	000 100	01000	10101	1111 1111 1111 1101		
8000C	sw $s5, 4($t1)	101 011	10101	01001	0000 0000 0000 0100		

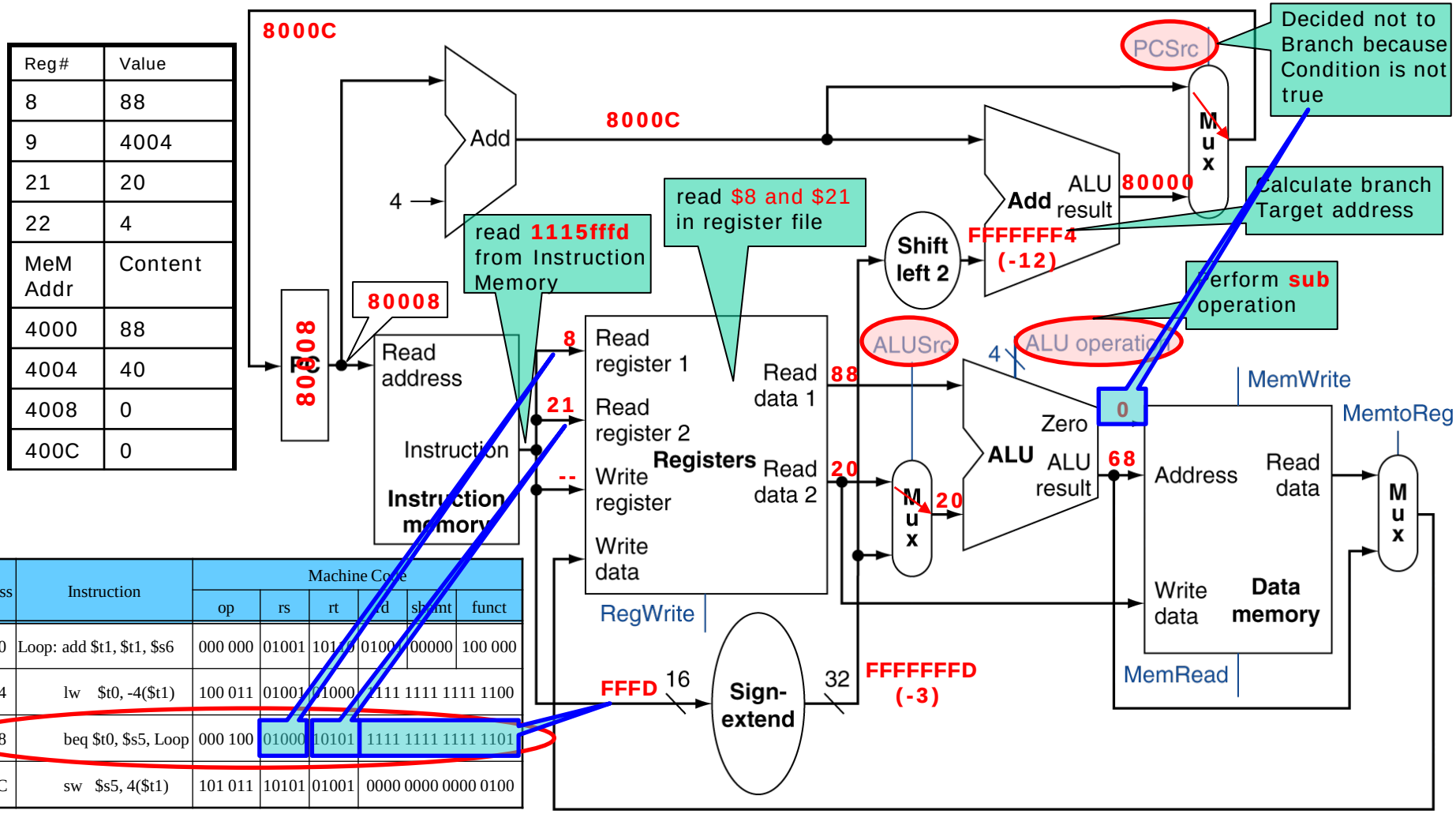
Example ~ How data flow through datapath (I-type Instruction: lw)



Reg#	Value
8	88
9	4004
21	20
22	4
MeM Addr	Content
4000	88
4004	40
4008	0
400C	0

Address	Instruction	Machine Code					
		op	rs	rt	rd	shamt	funct
80000	Loop: add $t1, $t1, $s6	000 000	01001	10110	01001	00000	100 000
80004	lw $t0, -4($t1)	100 011	01001	01000	1111 1111 1111 1100		
80008	beq $t0, $s5, Loop	000 100	01000	10101	1111 1111 1111 1101		
8000C	sw $s5, 4($t1)	101 011	10101	01001	0000 0000 0000 0100		

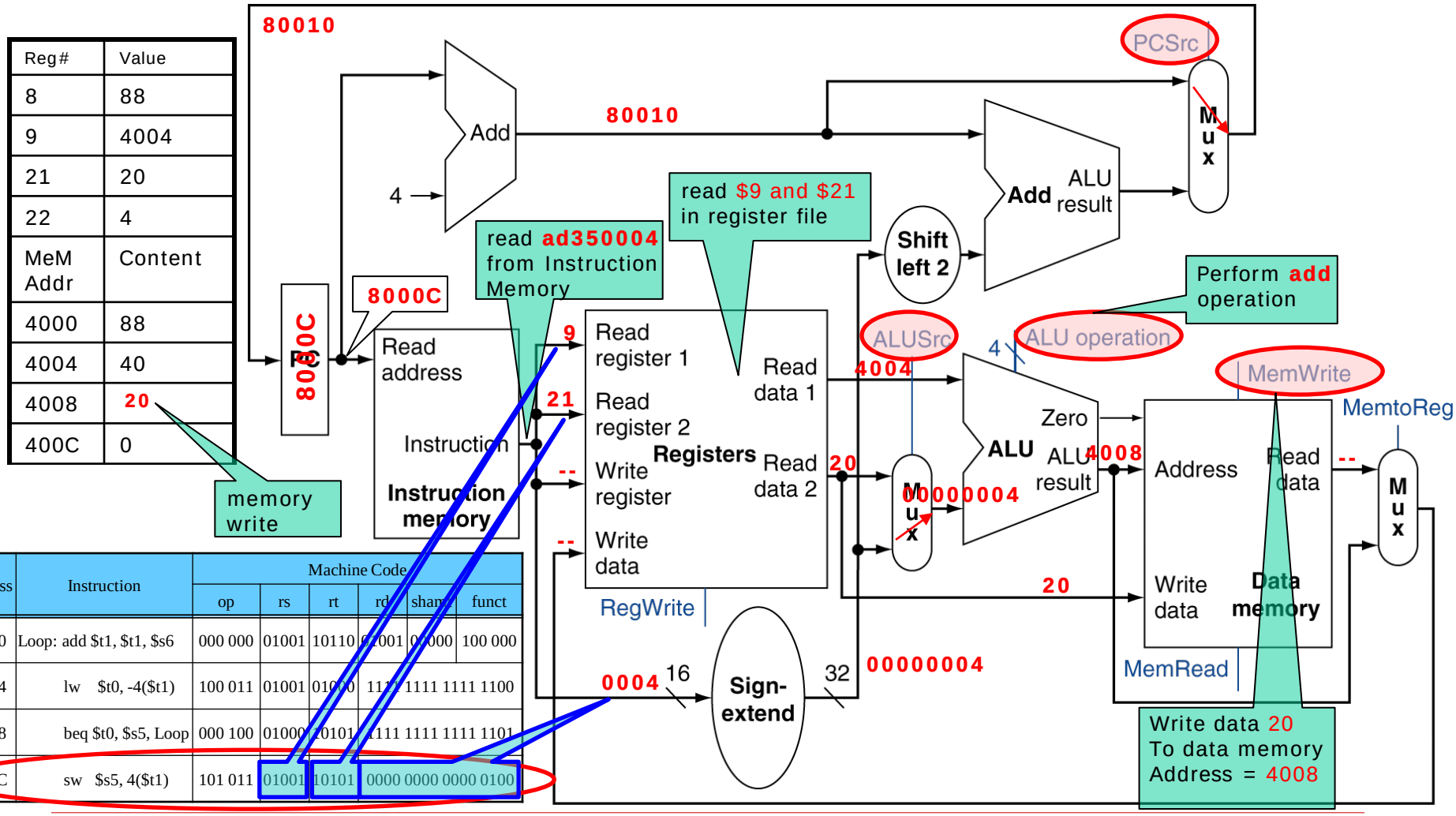
Example ~ How data flow through datapath (I-type Instruction: Beq)



Reg#	Value
8	88
9	4004
21	20
22	4
MeM Addr	Content
4000	88
4004	40
4008	0
400C	0

Address	Instruction	Machine Code					
		op	rs	rt	rd	shamt	funct
80000	Loop: add $t1, $t1, $s6	000 000	01001	10110	01001	00000	100 000
80004	lw $t0, -4($t1)	100 011	01001	01000	1111 1111 1111 1100		
80008	beq $t0, $s5, Loop	000 100	01000	10101	1111 1111 1111 1101		
8000C	sw $s5, 4($t1)	101 011	10101	01001	0000 0000 0000 0100		

Example ~ How data flow through datapath (I-type Instruction: sw)



Reg#	Value
8	88
9	4004
21	20
22	4
MeM Addr	Content
4000	88
4004	40
4008	20
400C	0

Address	Instruction	Machine Code					
		op	rs	rt	rd	shamt	funct
80000	Loop: add $t1, $t1, $s6	000 000	01001	10110	01001	00000	100 000
80004	lw $t0, -4($t1)	100 011	01001	01000	1111 1111 1111 1100		
80008	beq $t0, $s5, Loop	000 100	01000	10101	1111 1111 1111 1101		
8000C	sw $s5, 4($t1)	101 011	01001	10101	0000 0000 0000 0100		