CSCI 210: Computer Architecture
Lecture 25: Data Path 2

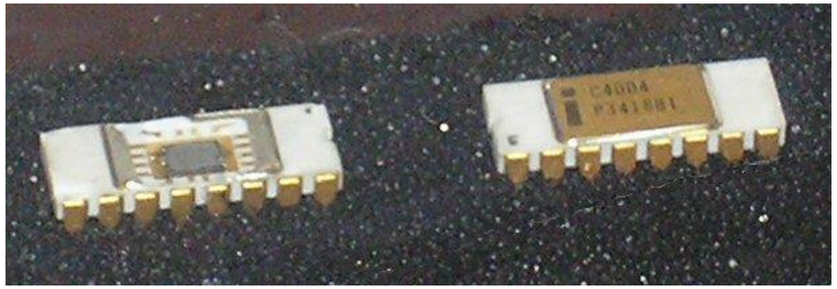
Stephen Checkoway

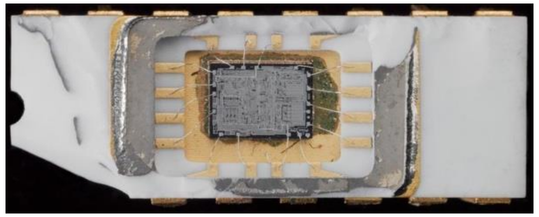
Slides from Cynthia Taylor

Announcements

- Problem Set 8 Due TONIGHT 10pm
  - Problem Sets 5 and 6 graded, resubmits up
- Lab 7 due Monday 10 pm
- Computational Skills Hours, King 225
  - Wednesday, Sunday 7 – 9 pm
- Cynthia’s Student Hours King 139 B
  - Monday 4:30 – 5:30 pm
  - Thursday 10 am – noon

CS History: Intel 4004





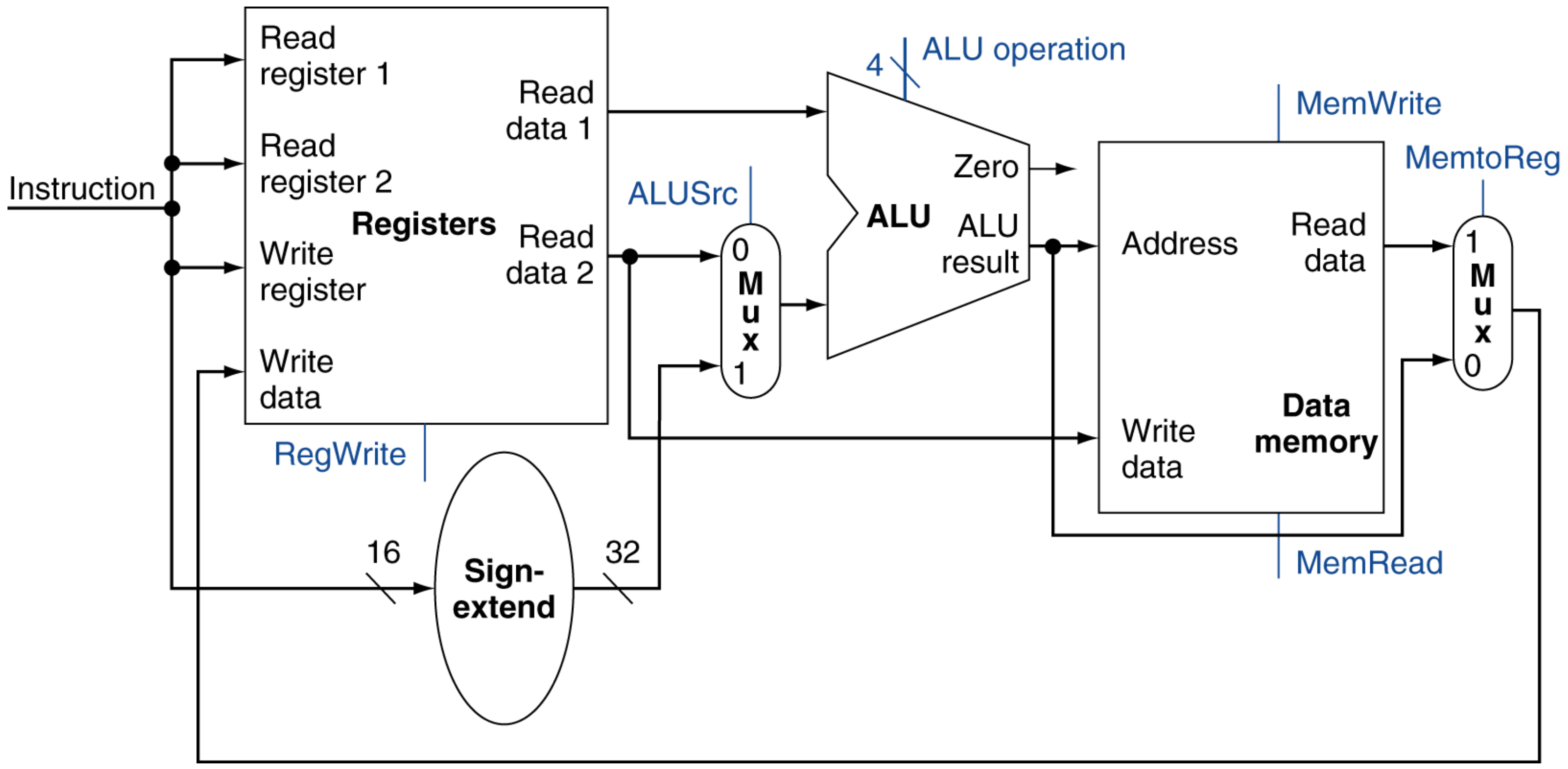
- First commercially available microprocessor (single chip with both data processing logic and control)
- Released in 1971
- Had 12-bit addresses, 8-bit instructions, and 4-bit data words
- 16 4-bit registers
- Designed for Binary-Coded Decimal, in which every decimal digit is stored as a 4-bit value
  - Still present in x86

The Processor: Datapath & Control

- We're ready to look at an implementation of MIPS simplified to contain only:
  - memory-reference instructions: `lw,  sw`
  - arithmetic-logical instructions: `add,  sub,  and,  or,  slt`
  - control flow instructions: `beq`

lw $t1, 4($t0)

$t0 is register 8, $t1 is register 9
$t0 holds 0x07AB8110
0x07AB8114 holds 12



op = lw	rs = 8	rt = 9	imm = 4

Branch Instructions

- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend offset
  - Shift left 2 bits (word offset)
  - Add to PC + 4
    - Already calculated during instruction fetch

Conditional Branch Instructions Require

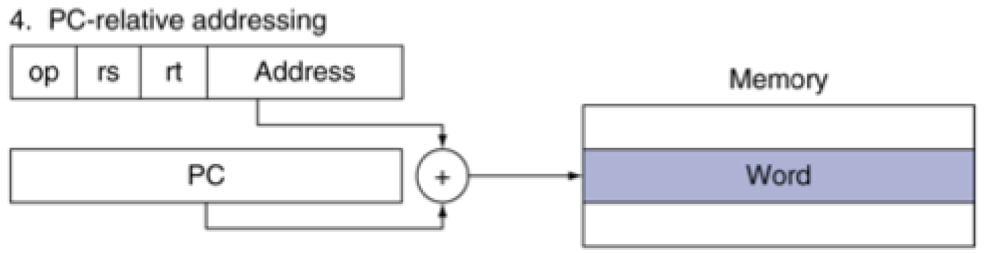
A. ALU

B. Registers and an ALU

C. Registers, ALU and Memory

D. Registers, an ALU and an Adder

beq $t2, $t3, 0x4F35



Read register operands

Compare operands

    Use ALU, subtract and check Zero output

Calculate target address

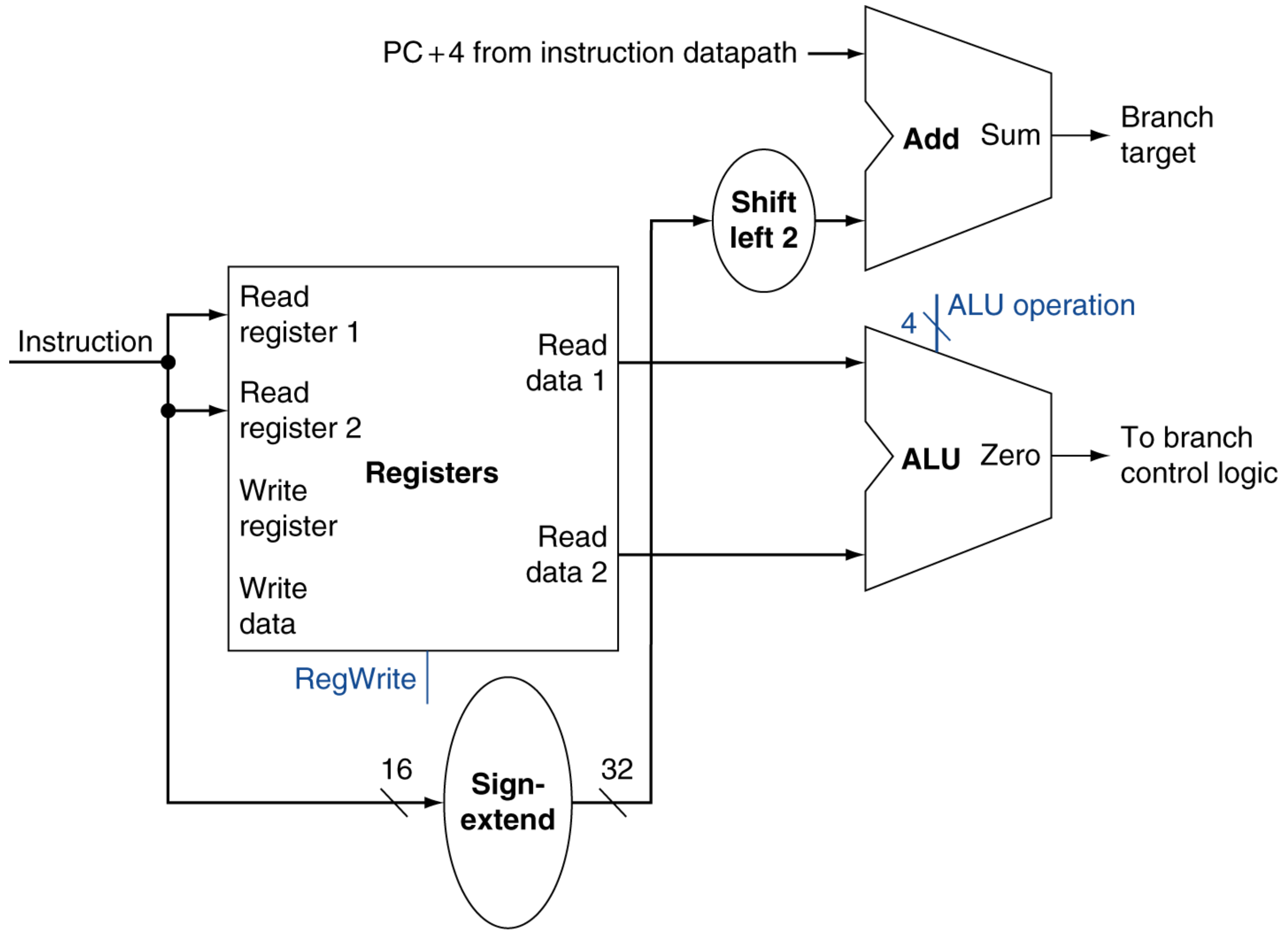
    Sign-extend offset

    Shift left 2 bits (word offset)

    Add to PC + 4

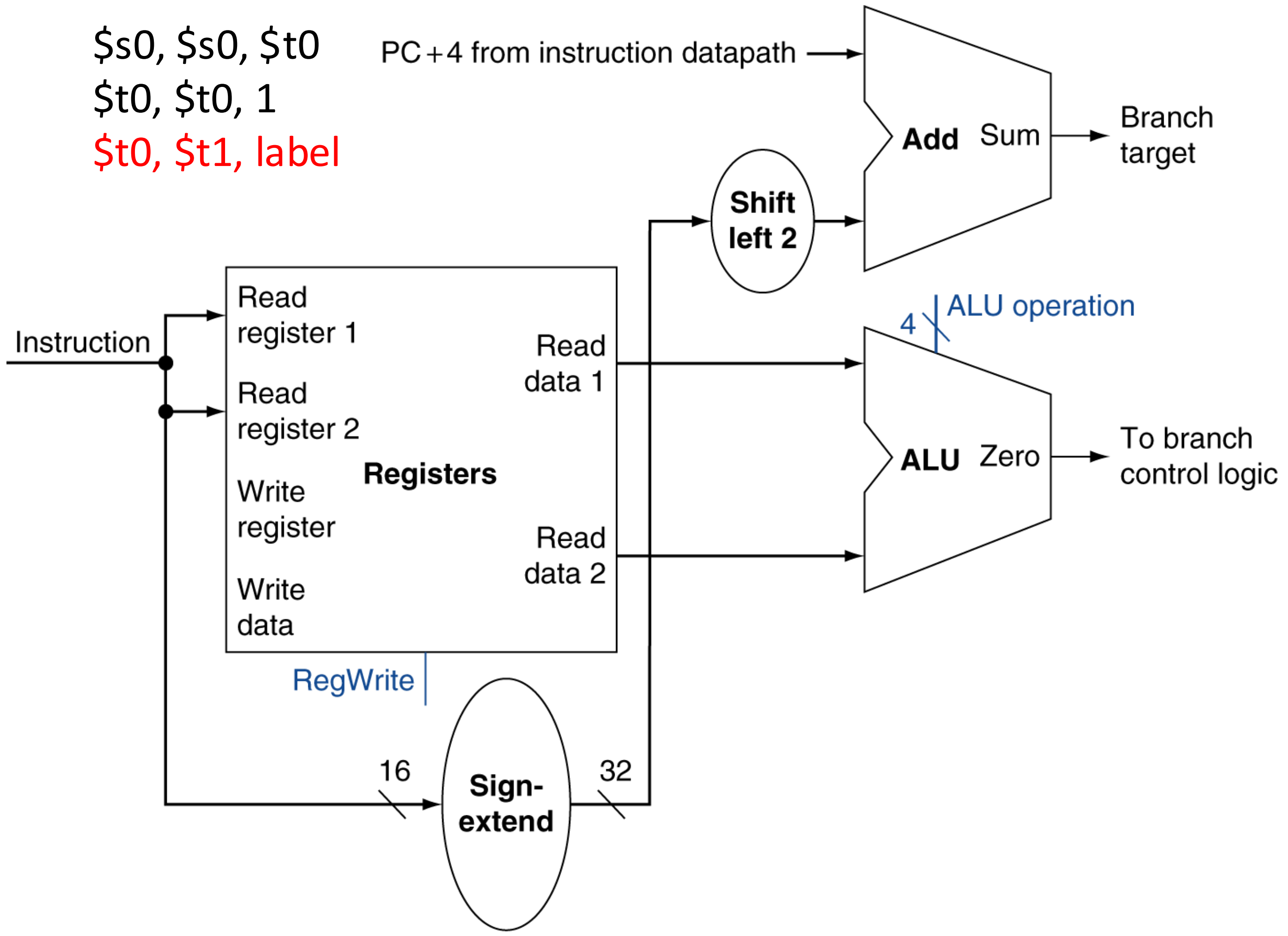
        Already calculated during instruction fetch

Branch Instructions



Branch Instructions

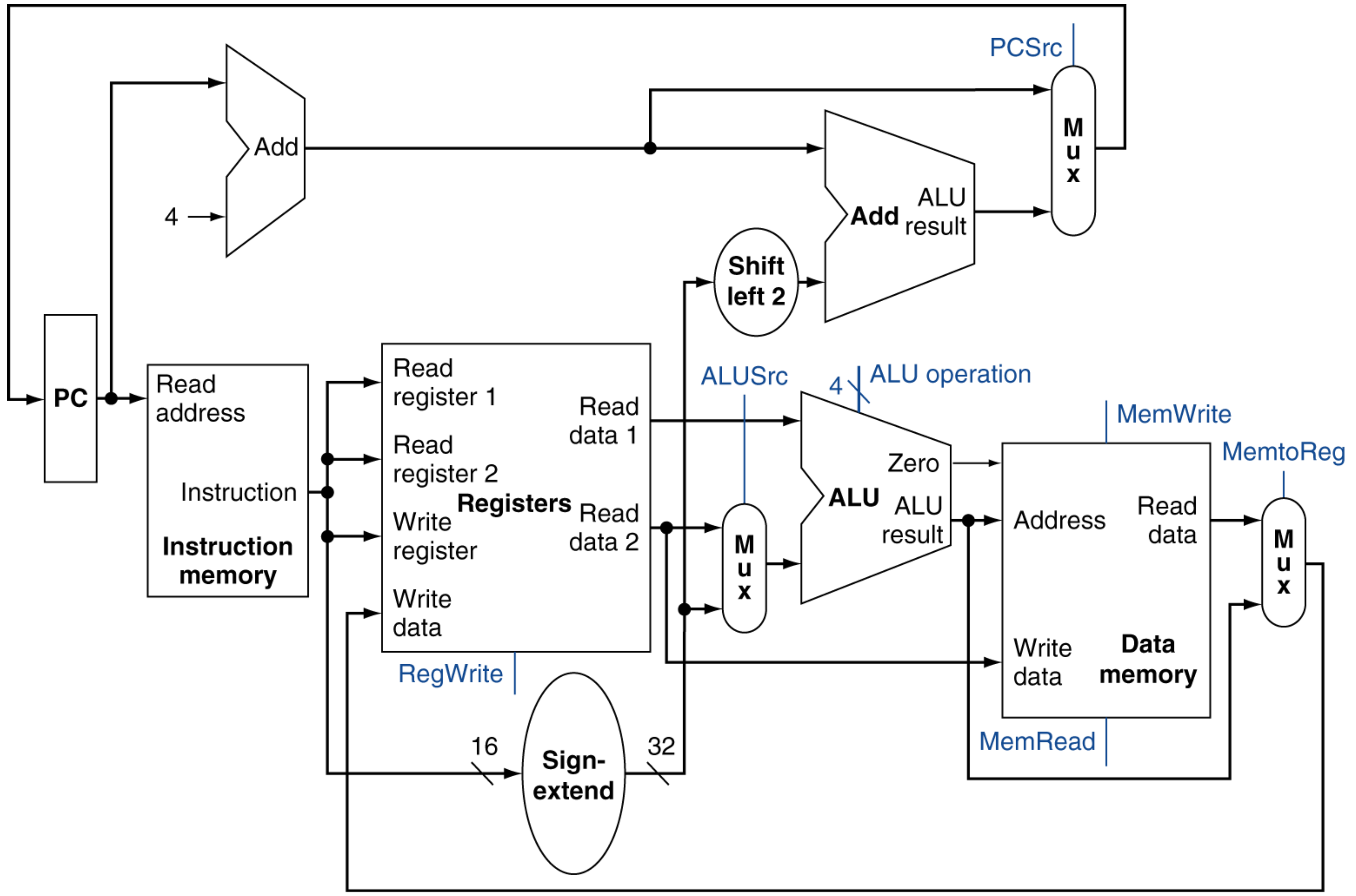
```
0x4045A130  label:  add       $s0, $s0, $t0
0x4045A134          addi      $t0, $t0, 1
0x4045A138          beq       $t0, $t1, label
```



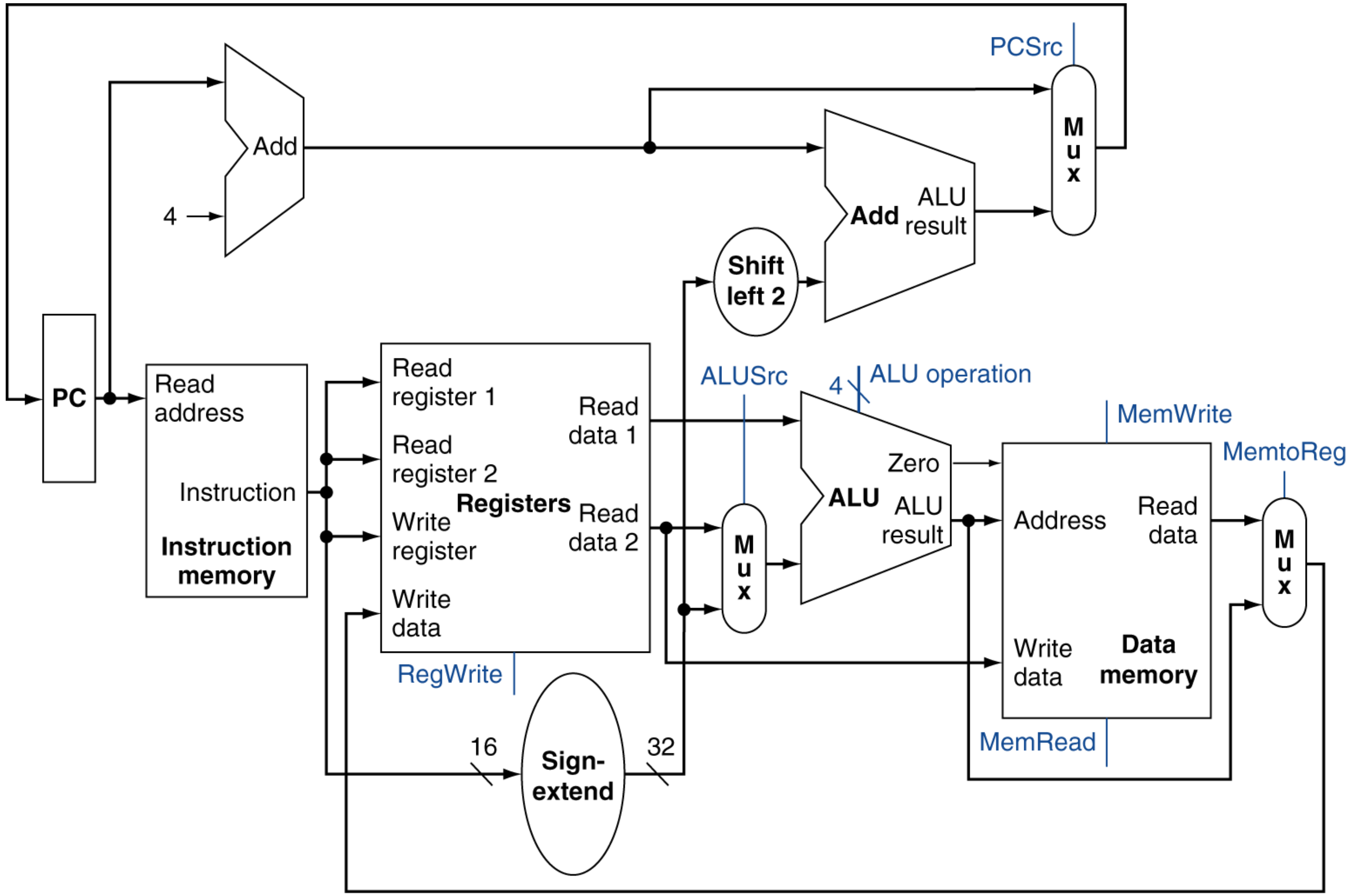
op = 0x04	rs = 8	rt = 9	imm = 0xFFFD

$t0 holds 5
$t1 holds 5

Datapath (still simplified a bit)



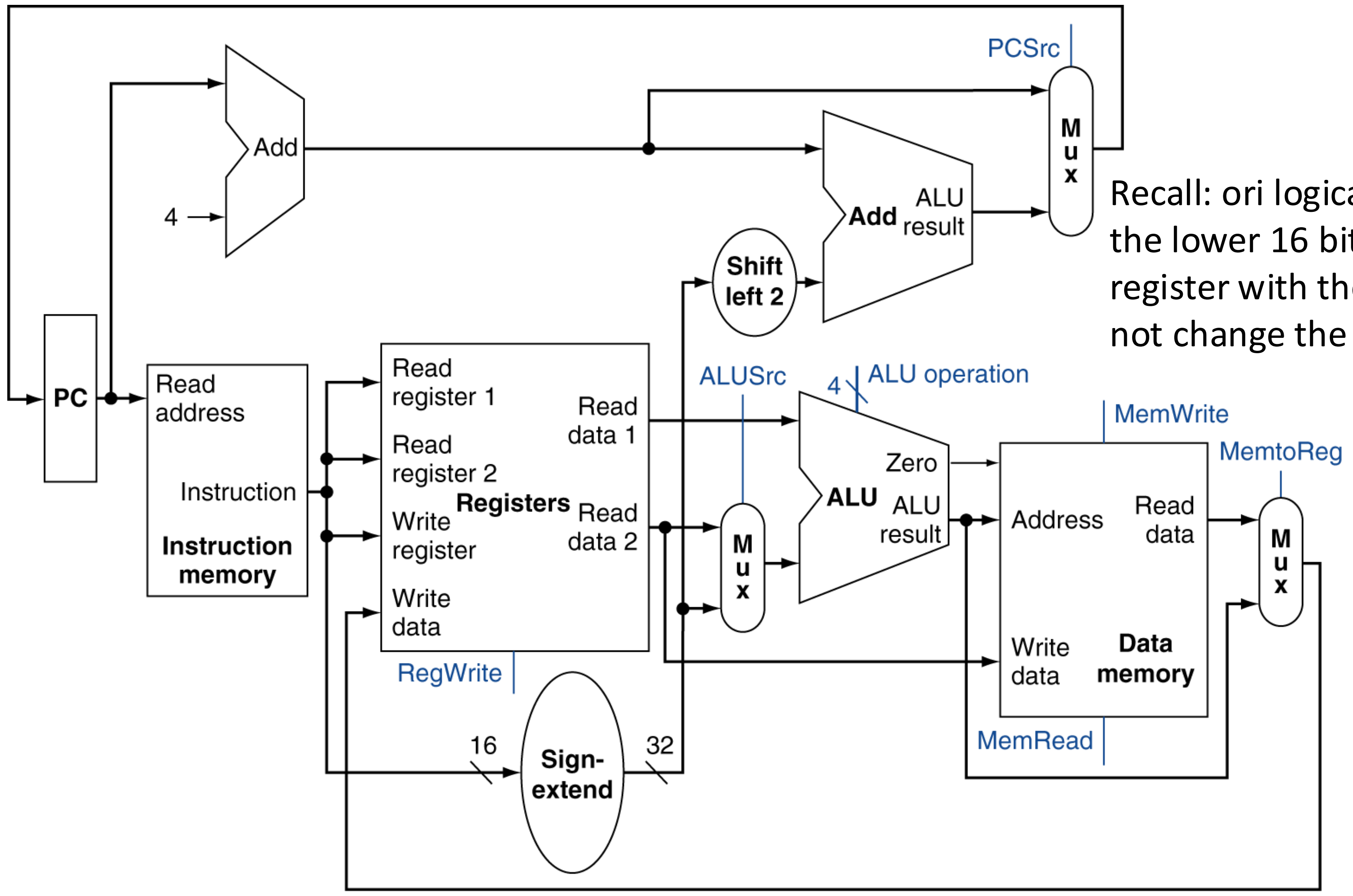
addi $t1, $t0, -1



$t0 holds 10

op = 0x08	rs = 8	rt = 9	imm = 0xFFFF

What do we need to add to support ori?

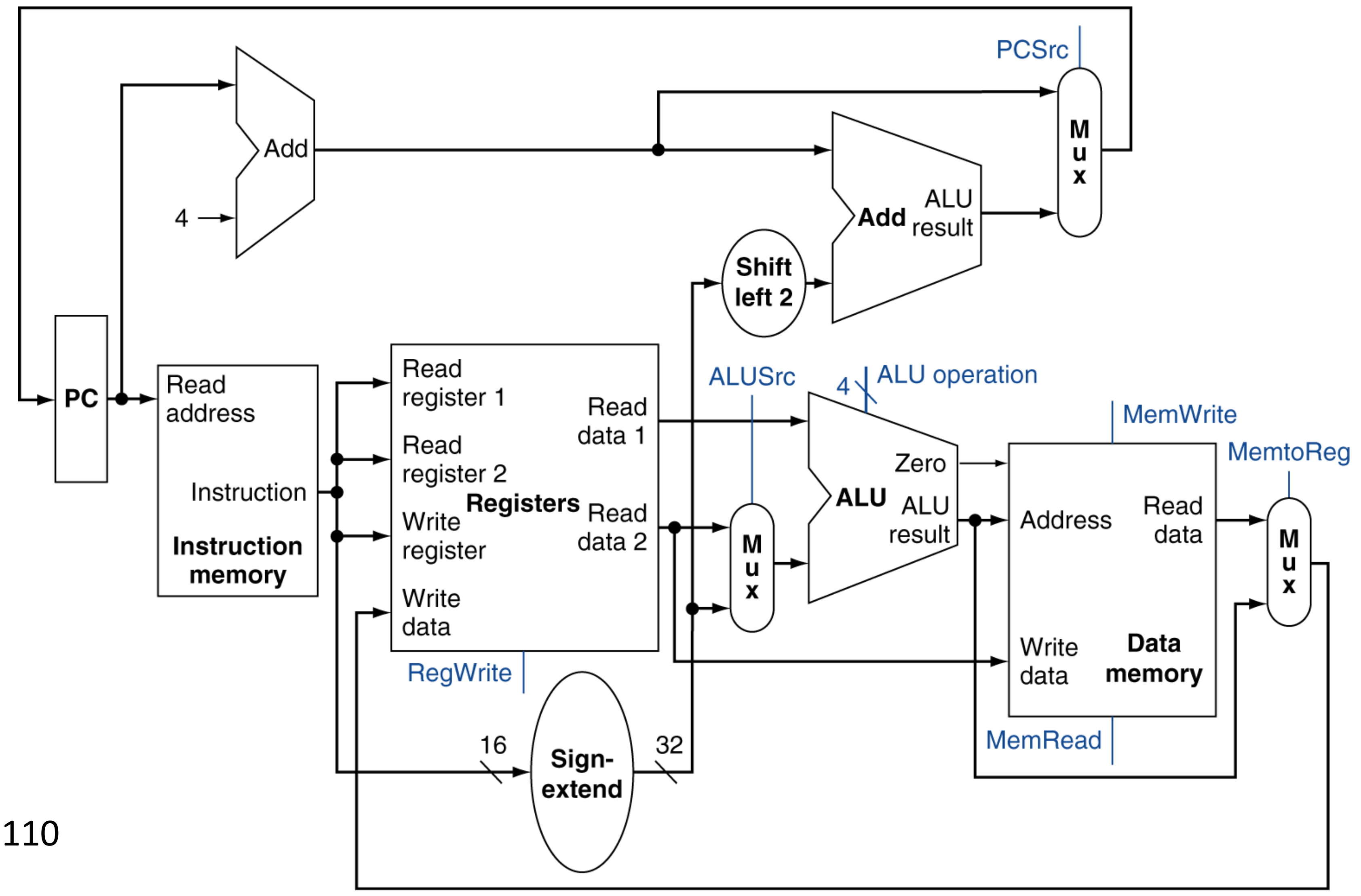


Recall: ori logically ors the lower 16 bits of the specified register with the immediate. It does not change the upper 16 bits.

ori $t0, $t1, -3
$t1 holds 5

op = 0x0D	rs = 9	rt = 8	imm = 0xFFFD

sw $t1, 8($t0)



$t0 holds 0x07AB8110
$t1 holds 5

op = 0x2B	rs = 8	rt = 9	imm = 0x0008

Composing the Elements

- Data path does an instruction in one clock cycle
  - Each data path element can only do one function at a time
  - Hence, we need separate instruction and data memories, ALU and adders, etc
- Use multiplexers where alternative data sources are used for different instructions

Key Points

- CPU is just a collection of state and combinational logic
- We just designed a very rich processor, at least in terms of functionality
- ET = IC * CPI * Cycle Time

Reading

- Next lecture: Control Path
  - Section 5.4