CSCI 210: Computer Architecture
Lecture 24: Data path

Stephen Checkoway

Slides from Cynthia Taylor

Announcements

- Problem Set 8 Due Friday
- Lab 7 due Sunday

Computer Science

PROFESSOR BEERS



4:30-6PM

Thursday November 5th



Drinks on the department!

GENDER SEXUALITY AND FEMINIST STUDIES PROGRAM PRESENTS:

PREFER NOT TO SAY:

QUICK Bites

RECKONING WITH CHOICES IN COLLECTING AND VISUALIZING GENDER DATA



FRIDAY

NOV. 6TH

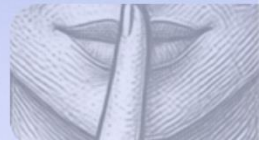
12:00- 12:55PM

KING 241



EMILY WANG

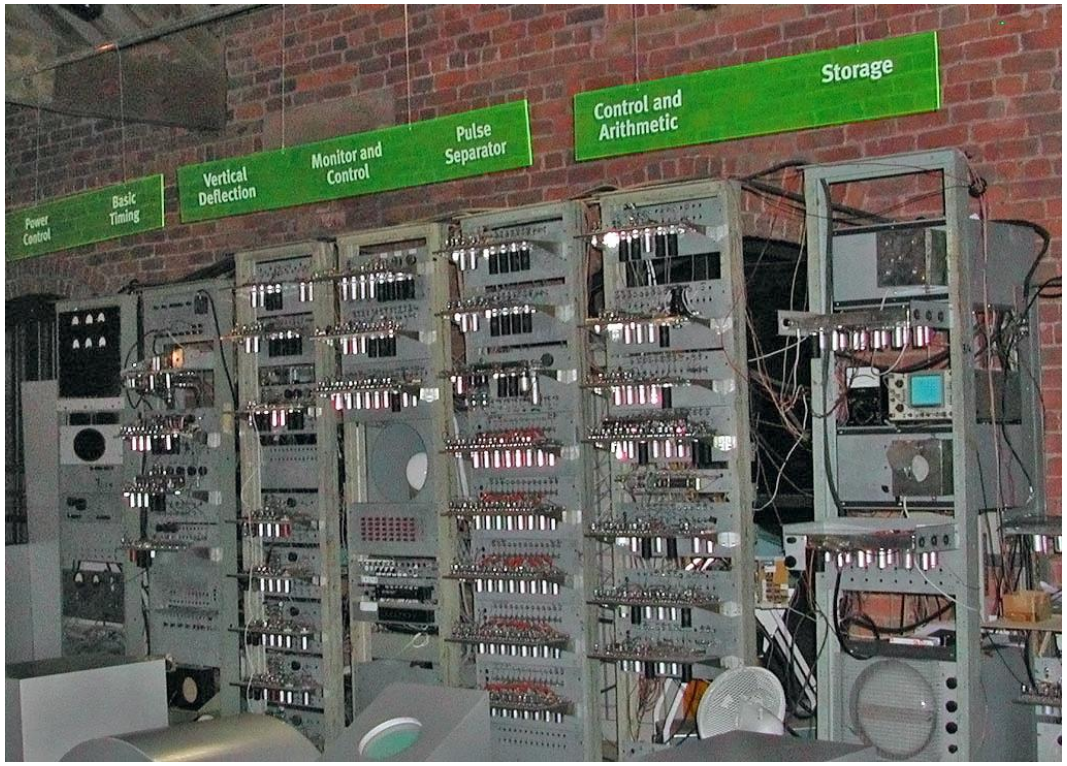
ASSISTANT PROFESSOR OF COMPUTER SCIENCE



Today’s Class

- The data path!

CS History: The Manchester Baby



- First stored-program computer
- Ran its first program on June 21, 1948
- Designed as a testbed for the first random-access memory
- Only arithmetic operations were addition and subtraction
- Its first program calculated the highest proper divisor of 2^{18} (262,144), by testing every integer from 2^{18} downwards
- This program was 17 instructions and took 52 minutes to run

First Manchester Baby program recreated in MIPS

```
main:
   lui   $s0, 4             # 2^18 = (2^2) << 16
   addi  $t0, $s0, -1       # t0 will hold the current divisor to test
outer:
   add   $t1, $s0, $zero    # t1 = 2^18
inner:
   sub   $t1, $t1, $t0
   beq   $t1, $zero, done   # t0 holds the highest divisor
   slt   $t2, $t1, $zero    # t2 = 1 if t1 < 0
   beq   $t2, $zero, inner # if t2 = 0, then go to inner
   addi  $t0, $t0, -1       # Try the next divisor
   j  outer
done:
   # t0 now holds the value of the largest divisor of 2^18
```

The Processor: Data path & Control

- We're ready to look at an implementation of RISC-V simplified to contain only:
  - memory-reference instructions: `ld, sd`
  - arithmetic-logical instructions: `add, sub, and, or`
  - control flow instructions: `beq`

Generic implementation

- Fetch
  - Use the program counter (PC) to supply instruction address
  - Get the instruction from memory
  - Update the program counter to the next instruction
- Decode instruction
  - Read registers
  - Read the instruction to decide how to execute
- Execute
  - Perform necessary data manipulation
  - Write to registers

To fetch an instruction and update the PC, what hardware do we need?

- Fetch
  - Use the program counter (PC) to supply instruction address
  - Get the instruction from memory
  - Update the program counter to the next instruction

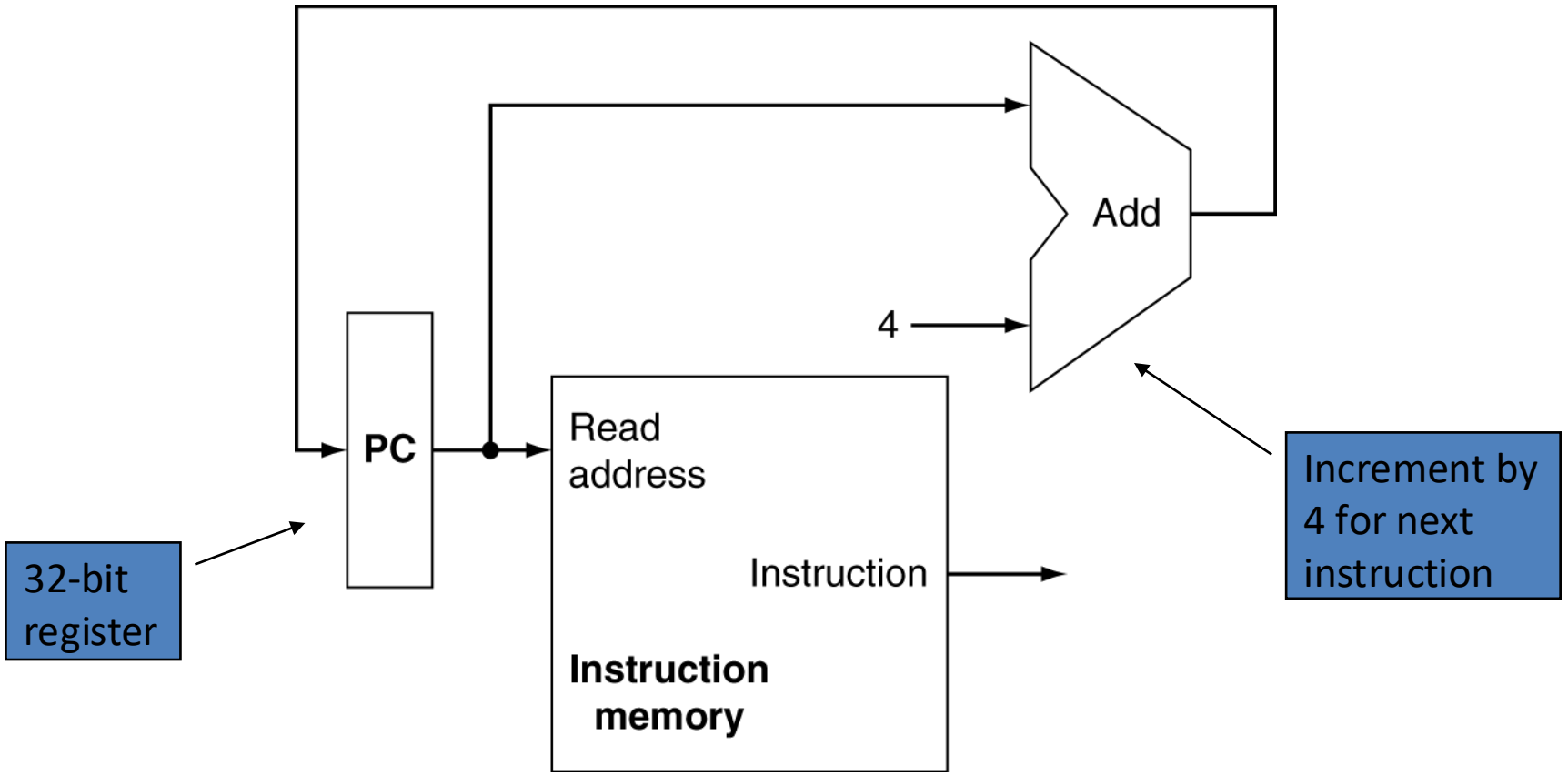
A. Register(s), Memory

B. Register(s), Adder, Memory

C. Register(s), ALU, Memory

D. More than this

Instruction Fetch



Generic implementation

- Fetch
  - Use the program counter (PC) to supply instruction address
  - Get the instruction from memory
  - Update the program counter to the next instruction
- Decode instruction
  - Read registers
  - Read the instruction to decide how to execute
- Execute
  - Perform necessary data manipulation
  - Write to registers

Registers for instructions

- `add t0, t1, t2` needs to read the values of registers `t1` and `t2` and write to register `t0`
- `ld t0, 8(t4)` needs to read one register and write one register
- `sd t0, -8(s0)` needs to read two registers and write zero registers

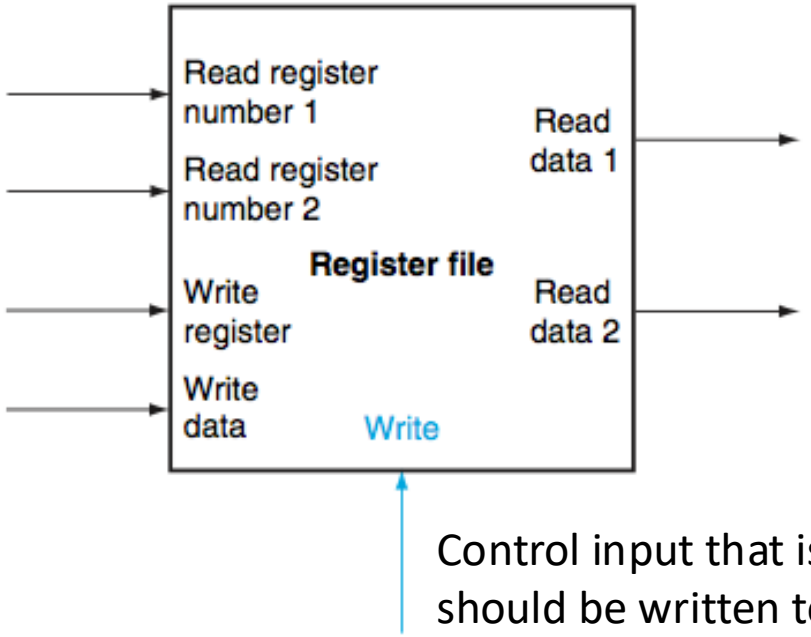
“Interface” for the register file

- We need the ability to read from up to 2 registers and write up to 1 register

Interface:

- Three 5-bit register select inputs (which come from rs, rt, rd)
- Two 64-bit data outputs (data in registers specified by rs and rt)
- One 64-bit data input (data to write to rd)
- One 1-bit control input (should input data be written to rd or not)

Register File



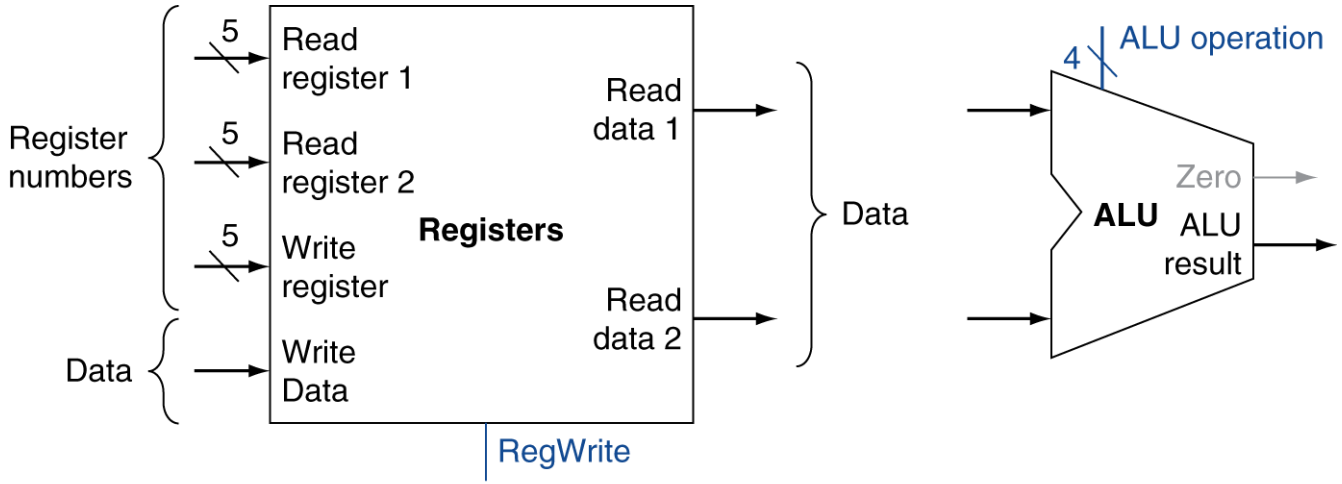
Control input that is 1 if the write data should be written to the register specified by write register

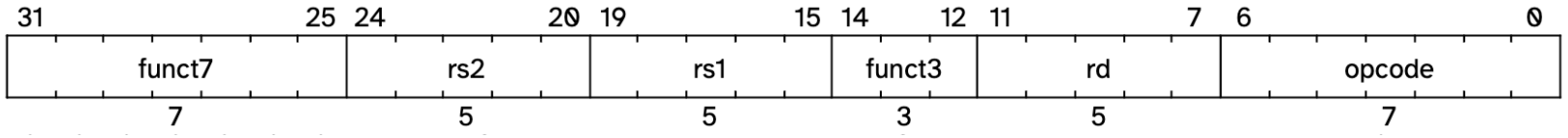
Generic implementation

- Fetch
  - Use the program counter (PC) to supply instruction address
  - Get the instruction from memory
  - Update the program counter to the next instruction
- Decode instruction
  - Read registers
  - Read the instruction to decide how to execute
- Execute
  - Perform necessary data manipulation
  - Write to registers

R-Format Instructions

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





Data memory

- `sub t0, t1, t2` does not read or write memory
- `ld t0, 0(s0)` reads 64 bits from memory
- `sd t0, 0(s0)` writes 64 bits to memory

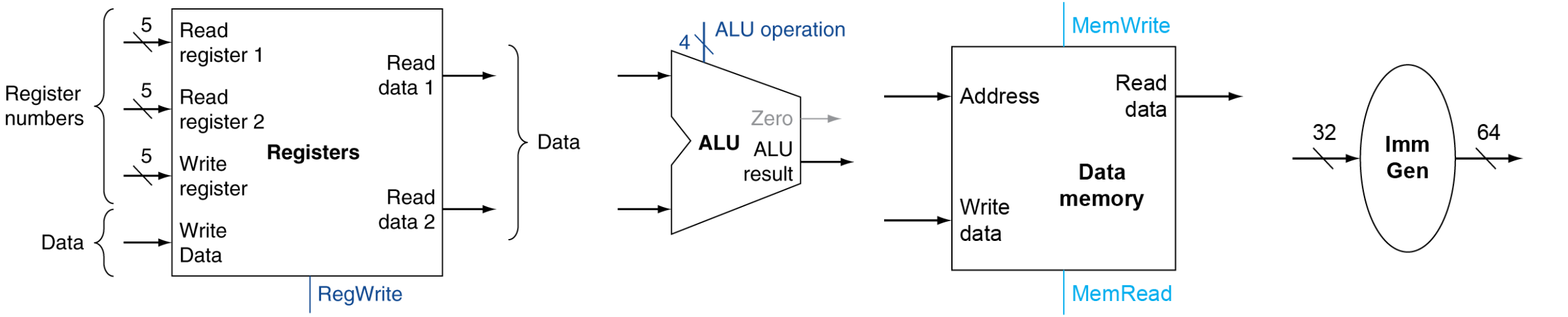
Which of these describes our interface for data memory? What do we need to support `ld 0(t3), t2` and `sd t4, 8(t5)`

	Data Inputs	Data Outputs	Select Inputs	Control Inputs
A	One 64-bit input	One 64-bit output	One 5-bit input	2 bits
B	None	One 64-bit output	Two 5-bit inputs	2 bits
C	One 64-bit input	One 64-bit output	One 64-bit input	2 bits
D	One 64-bit input	One 64-bit output	One 64-bit input	1 bit
E	One 64-bit input	One 64-bit output	One 5-bit input	1 bit

Data Inputs/Outputs are the data values read from/write to memory,
Select Inputs are the addresses we're reading/from writing to
Control Inputs are what operation the data memory does (load or store or others??)

Load/Store Instructions

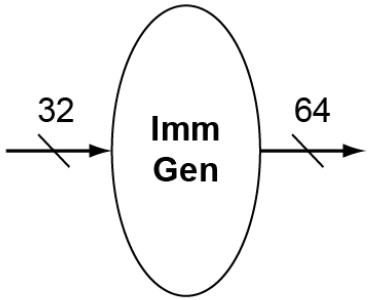
- Read register operands
- Calculate address using 12-bit offset
  - Use ALU, but sign-extend offset in the “immediate generation unit”
- Load: Read memory and update register
- Store: Write register value to memory



a. Registers | b. ALU | a. Data memory unit | b. Immediate generation unit

opcode	rs	rt	immediate

Immediate Generation Unit

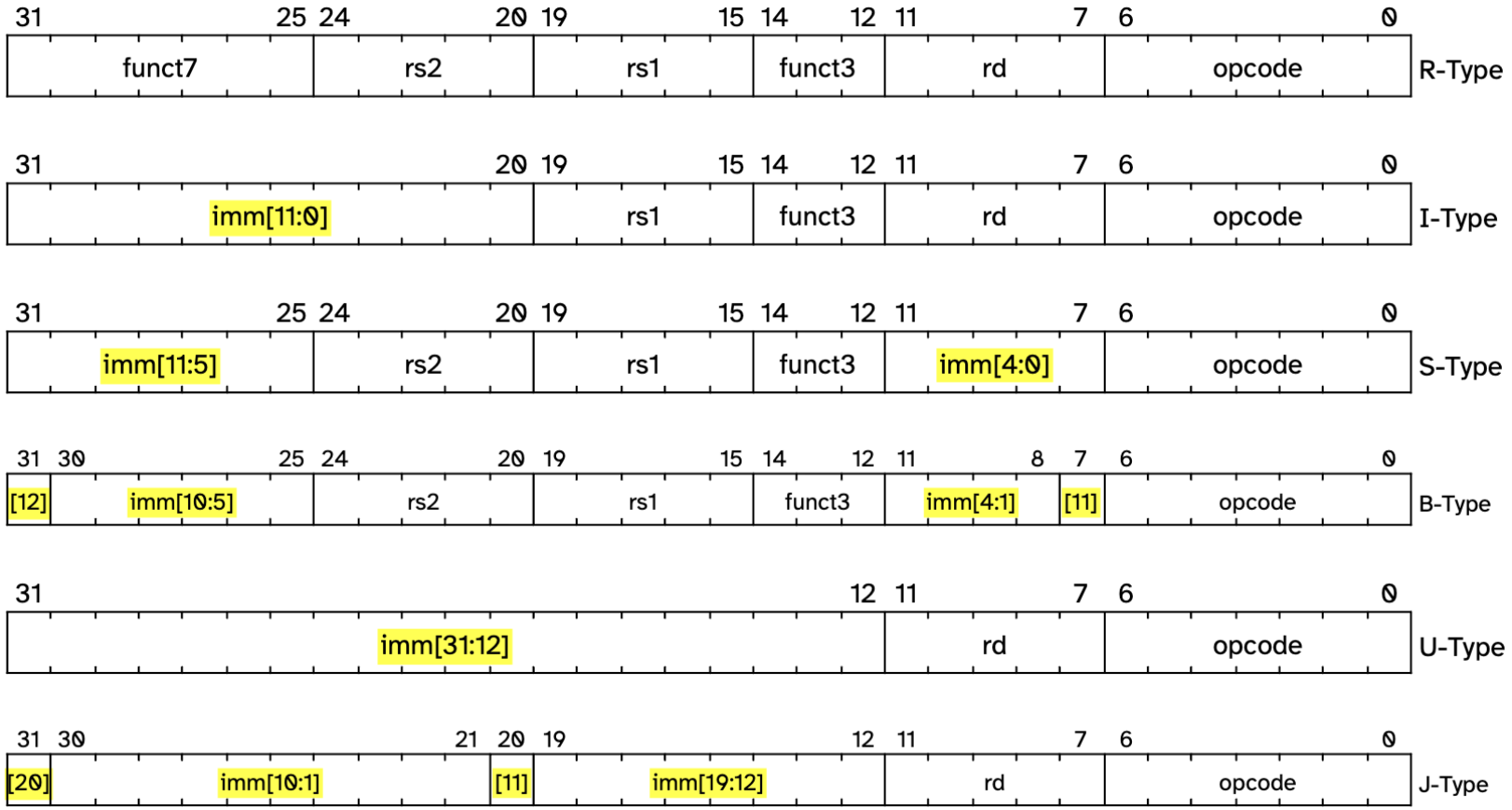


Input: 32 bits of the instruction

Output: sign-extended immediate value

Assembled from various immediate fields

Fields overlap where possible; sign bit is always instruction bit 31



Which is true about the ALU and the register file in RISC-V?

A. The ALU *always* performs an operation before accessing the register file

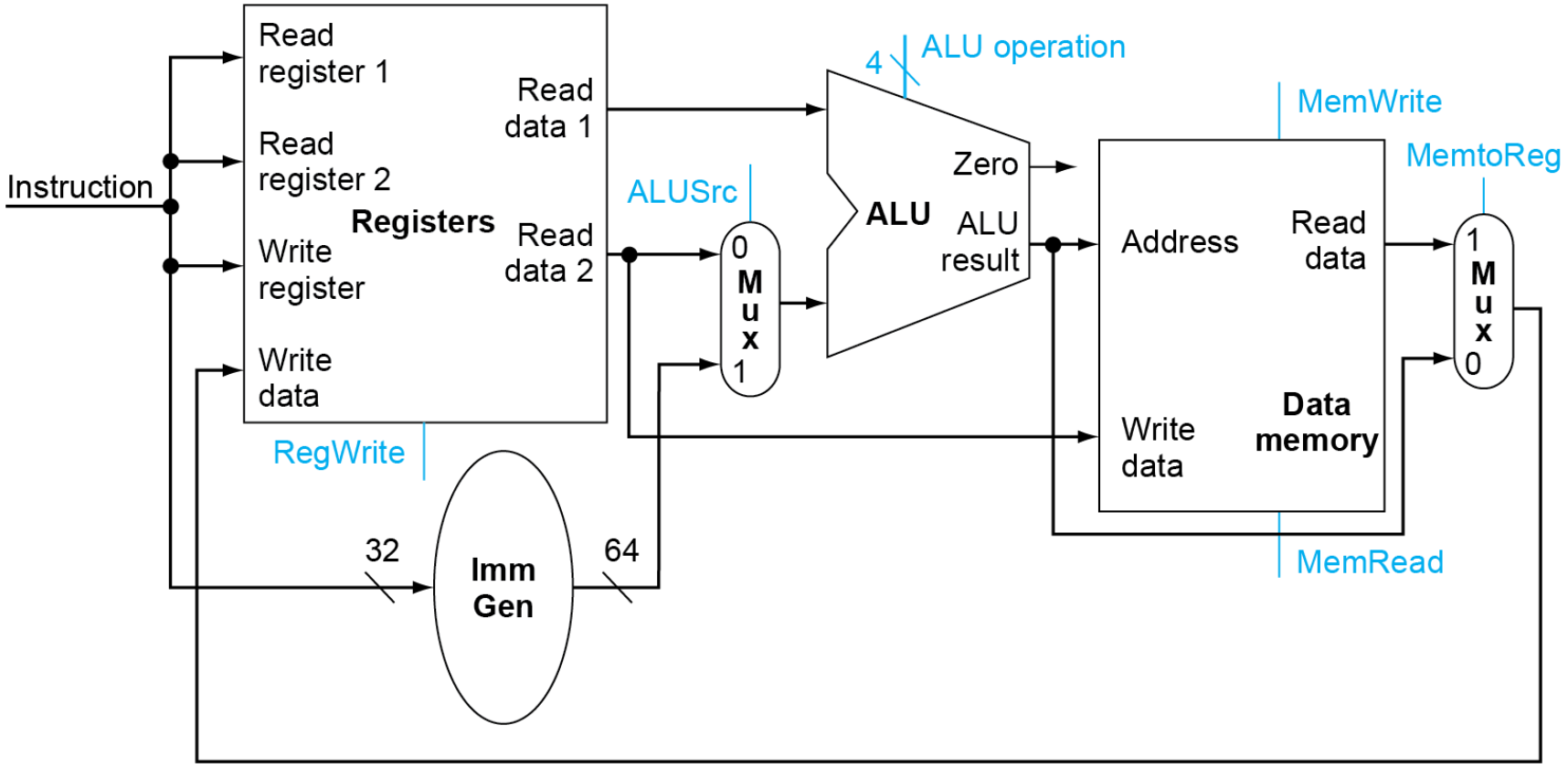
B. The ALU *sometimes* performs an operation before accessing the register file

C. The register file is *always* accessed before performing an ALU operation

D. The register file is *sometimes* accessed before performing an ALU operation

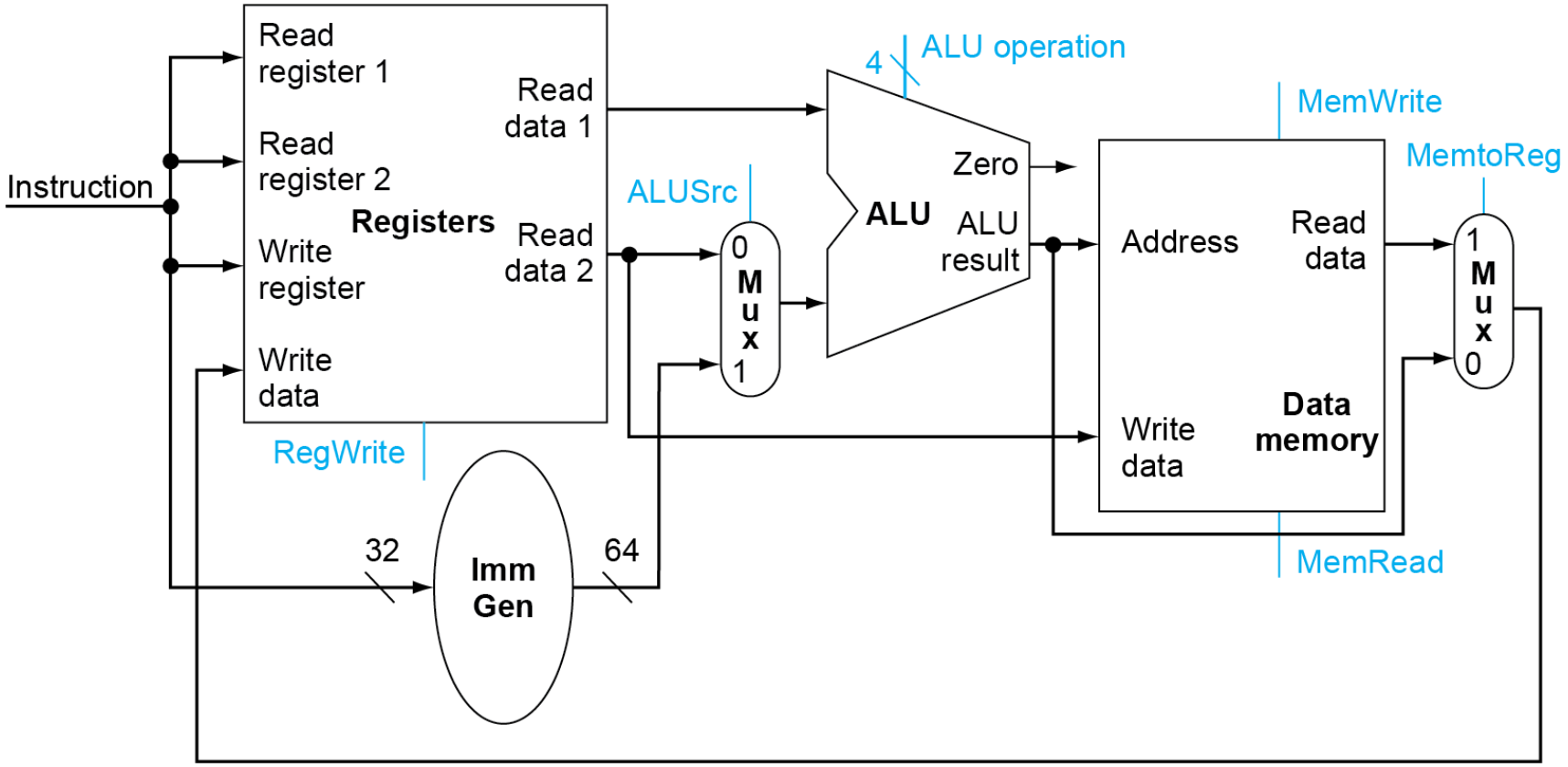
E. None of the above.

R-Type/Load/Store Datapath



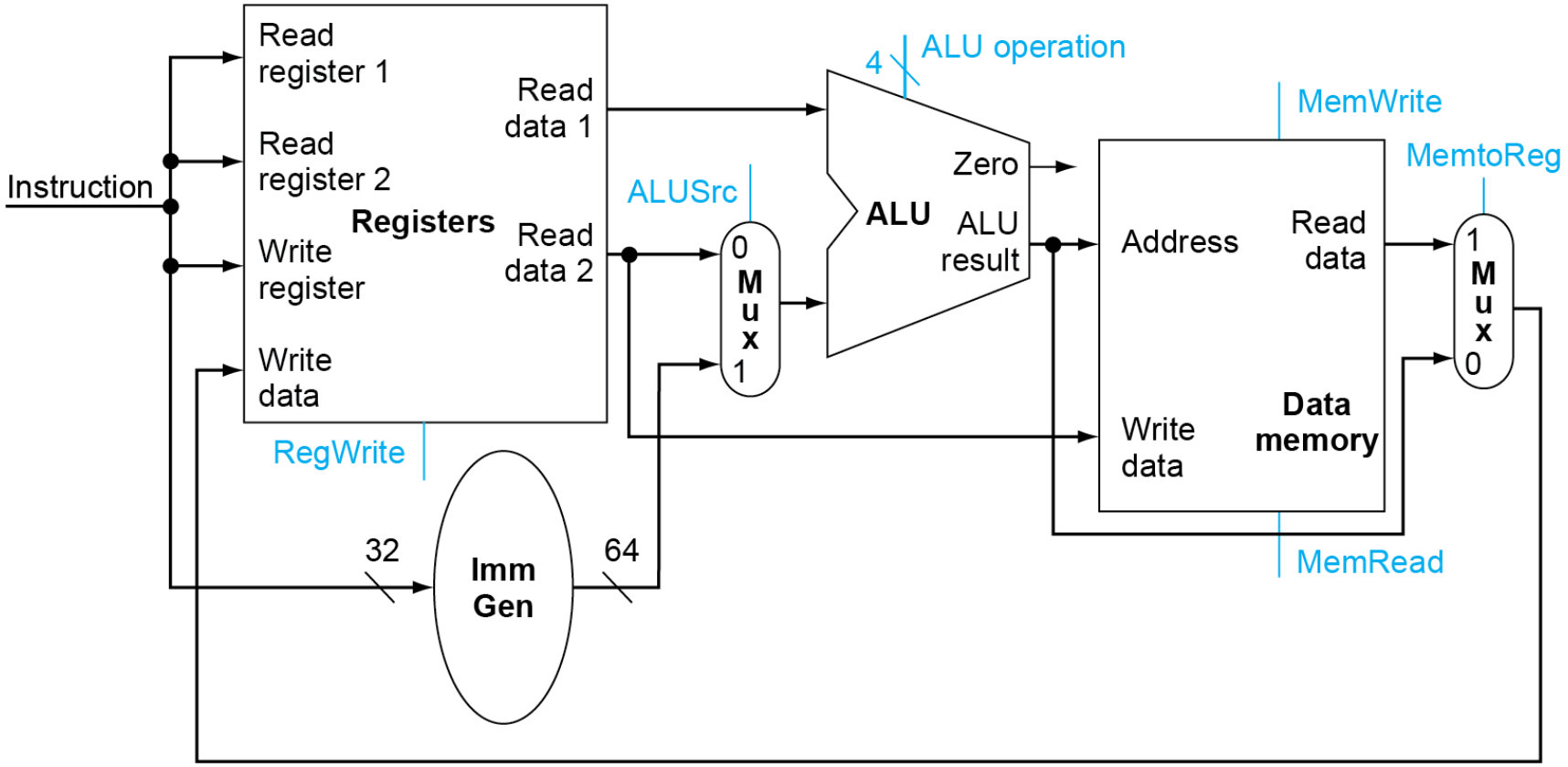
add x8, x8, x9

x8 holds 5
x9 holds 6



ld x10, 8(x12)

x12 holds 0x07AB8110
0x07AB8118 holds 105



Branch Instructions

- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend offset
  - Shift left 1 bits (word offset)
  - Add to the address of the instruction (pc-relative)

What hardware do we need for conditional branch instructions in our data path?

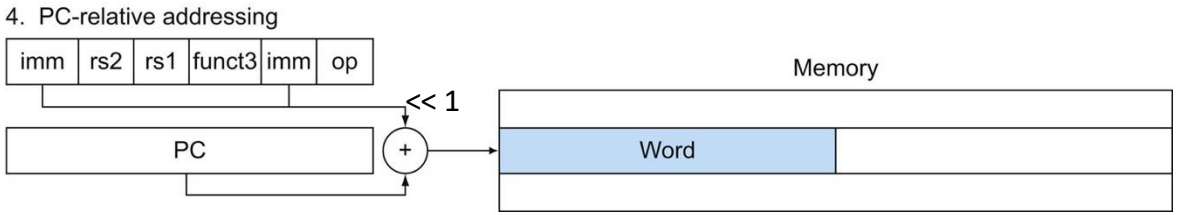
A. ALU

B. Registers and an ALU

C. Registers, ALU and Memory

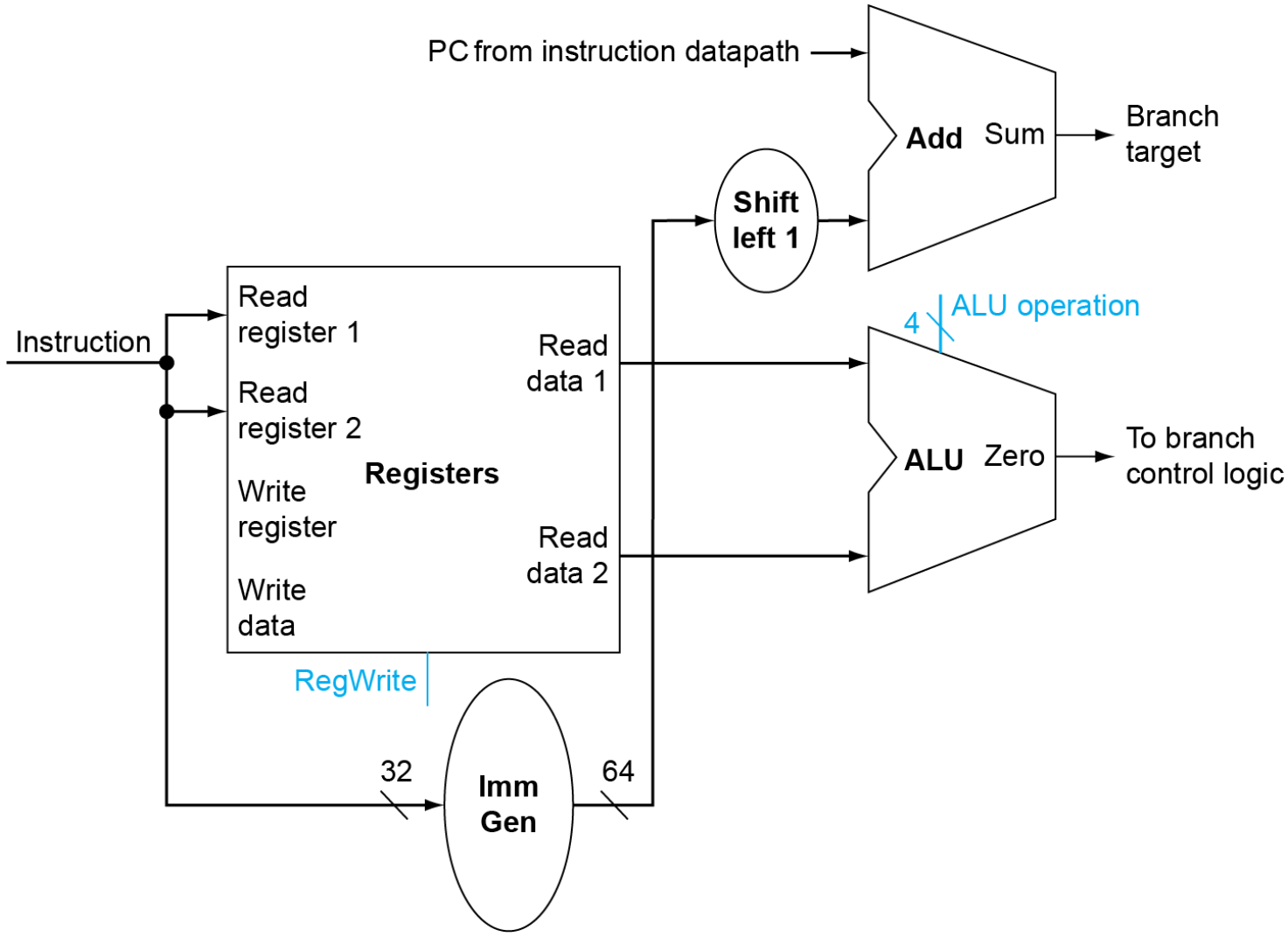
D. Registers, an ALU and an Adder

beq t2, t3, -16



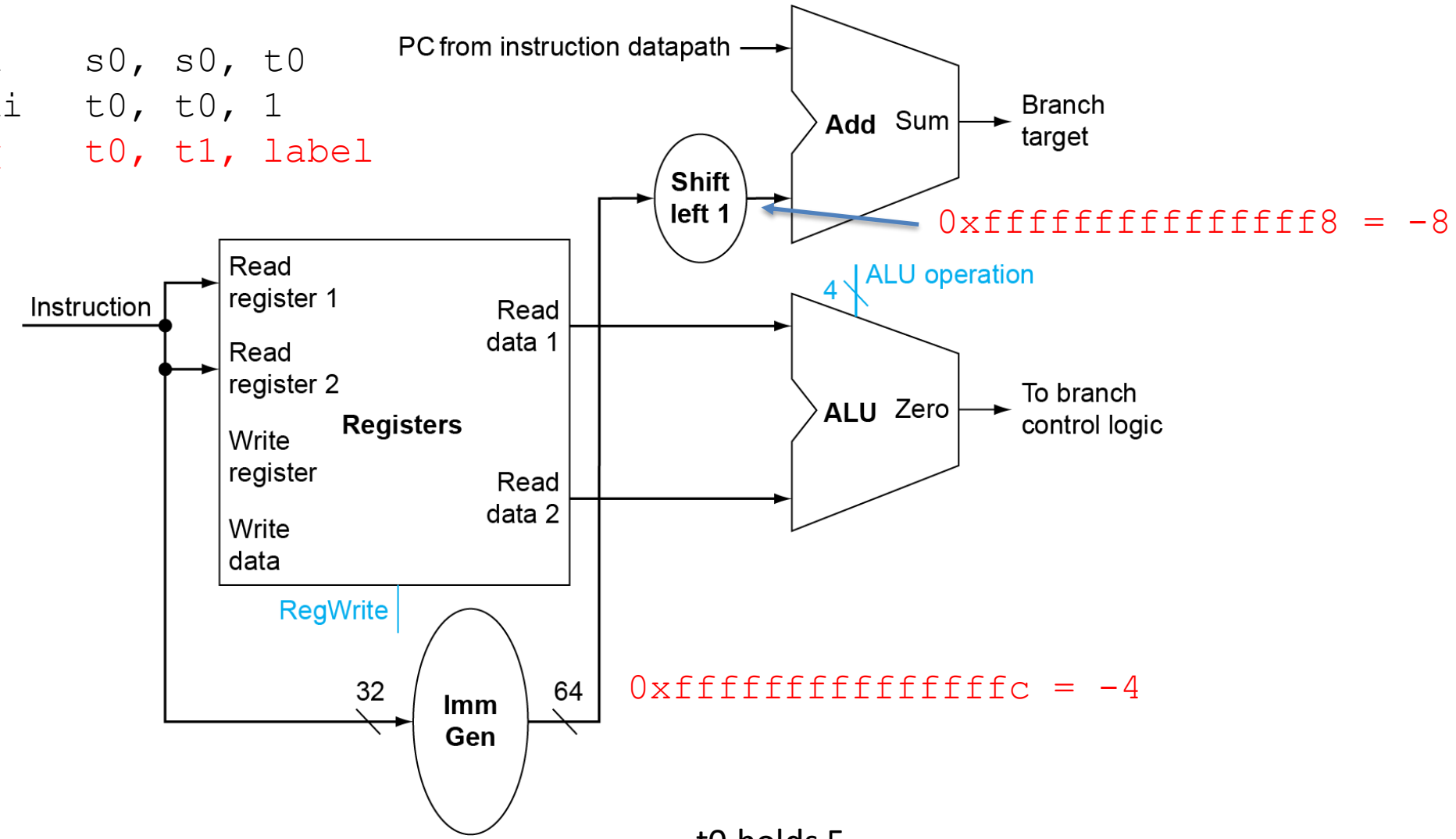
Read register operands
Compare operands
    Use ALU, subtract and check Zero output
Calculate target address
    Sign-extend offset
    Shift left 1 bits
    Add to PC

Branch Instructions



Branch Instructions

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



t0 holds 5
t1 holds 5

Datapath (still simplified a bit)

