CS/ECE 5381/7381 Computer Architecture Spring 2023

Dr. Manikas

Computer Science

Lecture 6: Feb. 9, 2023

Assignments

- Quiz 3 – due Sat., Feb. 11 (11:59 pm)
  - Covers concepts from Module 3 (this week)

Quiz 3 Details

- The quiz is open book and open notes.
- You are allowed 90 minutes to take this quiz.
- You are allowed 2 attempts to take this quiz - your highest score will be kept.
  - Note that some questions (e.g., fill in the blank) will need to be graded manually
- Quiz answers will be made available 24 hours after the quiz due date.

Instruction Set Principles

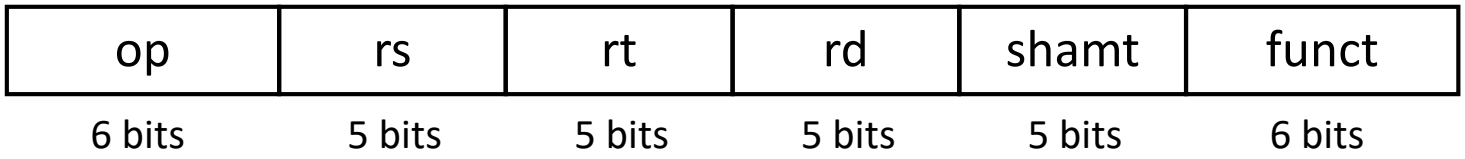
(Appendix A, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Outline

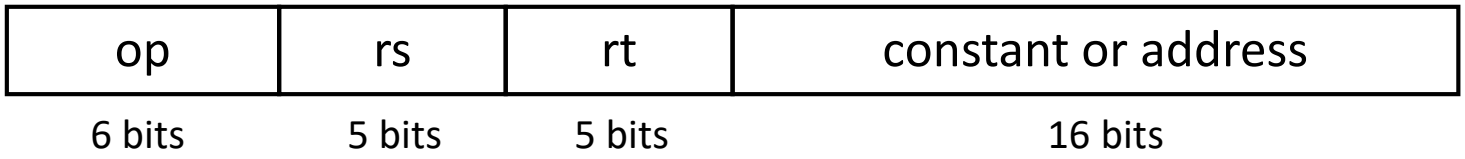
- A.9 MIPS Architecture
  - MIPS Instruction Set
  - MIPS Processor Design

MIPS R-format Instructions



- Instruction fields
  - op: operation code (opcode)
  - rs: first source register number
  - rt: second source register number
  - rd: destination register number
  - shamt: shift amount (00000 for now)
  - funct: function code (extends opcode)

MIPS I-format Instructions



- Immediate arithmetic and load/store instructions
  - rt: destination or source register number
  - Constant: -2^{15} to $+2^{15} - 1$
  - Address: offset added to base address in rs
- *Design Principle 4:* Good design demands good compromises
  - Different formats complicate decoding, but allow 32-bit instructions uniformly
  - Keep formats as similar as possible

RECALL INSTRUCTION TYPE:

① DATA MOVEMENT

MIPS: lw, sw

② ARITHMETIC/LOGIC

MIPS: ARITH = add, sub

LOGIC: and, or

③ CONTROL FLOW

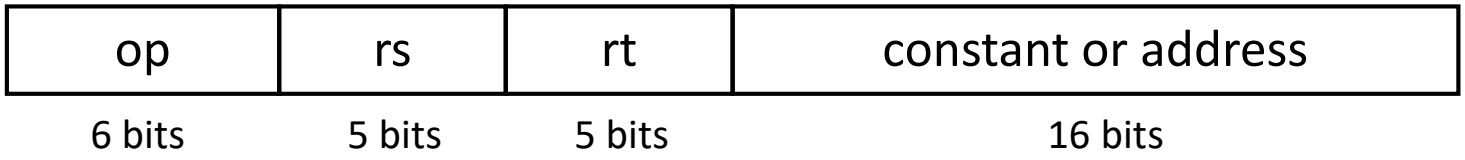
- BRANCHING OPERATIONS
- HOW DOES MIPS DO THESE?

Conditional Operations

- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially
- `beq rs, rt, L1` branch equal
  - if (rs == rt) branch to instruction labeled L1;
- `bne rs, rt, L1` branch not equal
  - if (rs != rt) branch to instruction labeled L1;
- `j L1`
  - unconditional jump to instruction labeled L1

Branch Addressing

- Branch instructions specify
  - Opcode, two registers, target address
- Most branch targets are near branch
  - Forward or backward



- PC-relative addressing
  - Target address = PC + offset × 4
  - PC already incremented by 4 by this time

It's a byte a word

Example A.9-1

Translating C to MIPS.

Assume that C variables are 32-bits and are assigned to MIPS registers as follows: h = $s1

Also assume that the base addresses of C arrays are stored in the following MIPS registers: A: $s2

What is the corresponding MIPS code for the following C statement?

```
A[6] = h;
```

GIVEN : A[6] = h ; C/C++ CODE

STORE OPERATION

ALSO : h = $s1 (MIPS REGISTER)

BASE ADDRESS OF ARRAY A IN $s2

MIPS CODE? STORE => sw

SYNTAX : sw $s1 , 24 ($s2)

↑ h — h

↑ 24 — OFFSET

↑ $s2 — BASE FOR A

OFFSET = 6 × 4 = 24

Example A.9-2

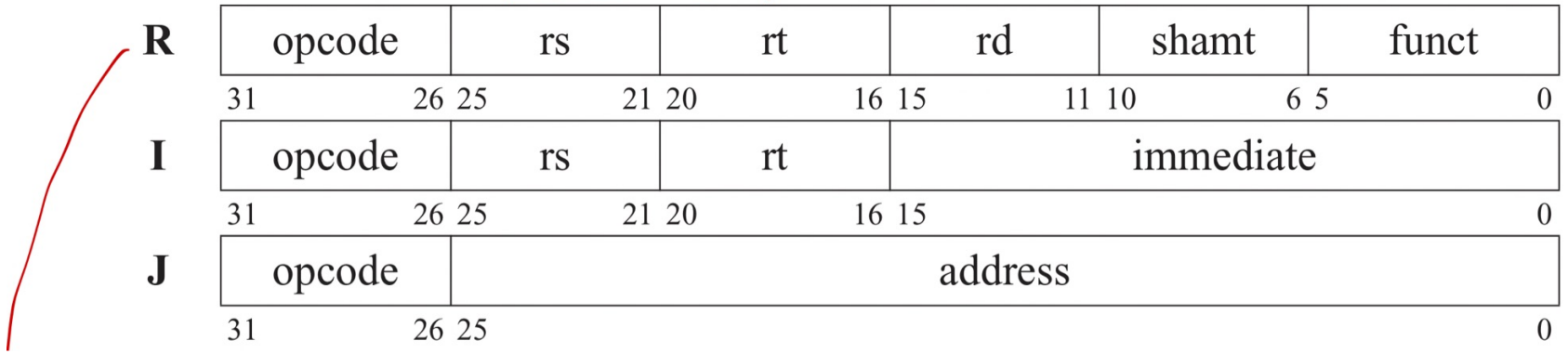
What MIPS instruction does this bit string represent?

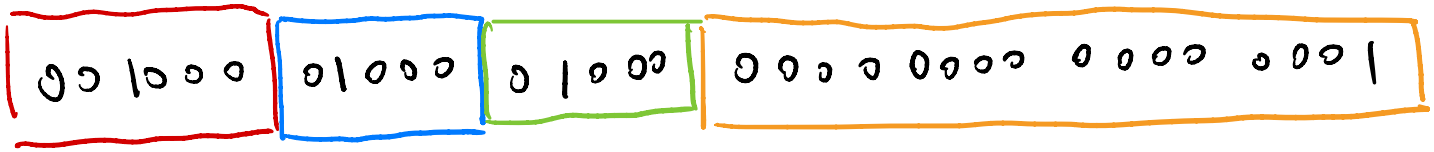
0010 0001 0000 1000 0000 0000 0000 0001

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	`add`	R	R[rd] = R[rs] + R[rt]	(1)	$0 / 20_{hex}$
Add Immediate	`addi`	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}

BASIC INSTRUCTION FORMATS





FIRST 6 BITS

OP CODE

rs = REG $01000_2 = 8$

$\underline{001000}_2 = 8_{10}$ WHO HAS OPCODE 8 ?

This is addi I-FORMAT

OPERATION : R[rt] = R[rs] + CONST.

rs = REG $01000_2 = 8_{10}$ ⇒ $t0

rt = REG $01000_2 = 8_{10}$ ⇒ $t0

CONSTANT = 0000 0000 0000 $0001_2 = 1_{10}$

MIPS INSTRUCTION : addi $t0, $t0, 1

i++

Example A.9-3

Given the following MIPS instruction:

```
sub $t0, $t1, $t2
```

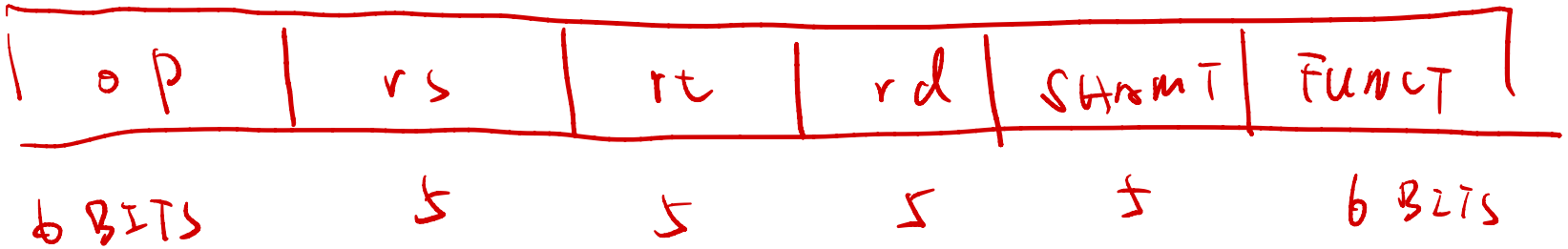
Show the **binary**, then **hexadecimal**, representation of this instruction

sub $t0, $t1, $t2

Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1) 0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]	0 / 23_{hex}

MIPS REF, SHEET

R-FORMAT



OPERATION: R[rd] = R[rs] − R[rt]

$t0 $t1 $t2

OPCODE = 0 = 000000_2 (6BITS)

FUNCT = 22_{16} = 100010_2 (6BITS)

SHAMT = 0 = 00000_2 (5BITS)

rs = $t1 => REG 9 => 01001_2 (5 Bits)

rt = $t2 => REG 10 => 01010_2

rd = $t0 => REG 8 => 01000_2

from the sheet

Fill in fields

op	rs	rt	rd	shamt	funct
000000	01001	01010	01000	00000	100010 $_2$

Binary — 32 Bits

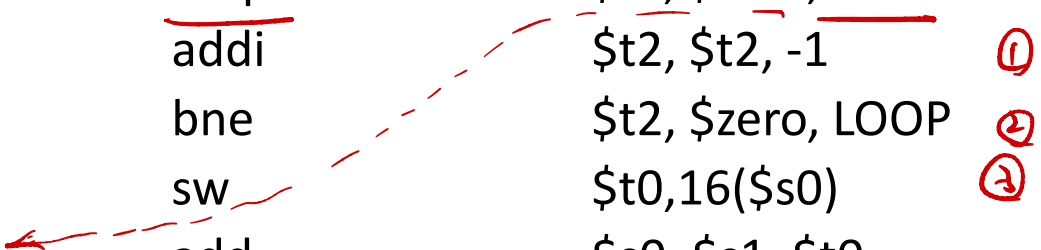
000000 | 01001 | 01010 | 01000 | 00000 | 100010

Hex 0 1 2 8 4 0 2 2_{16}

Example A.9-4

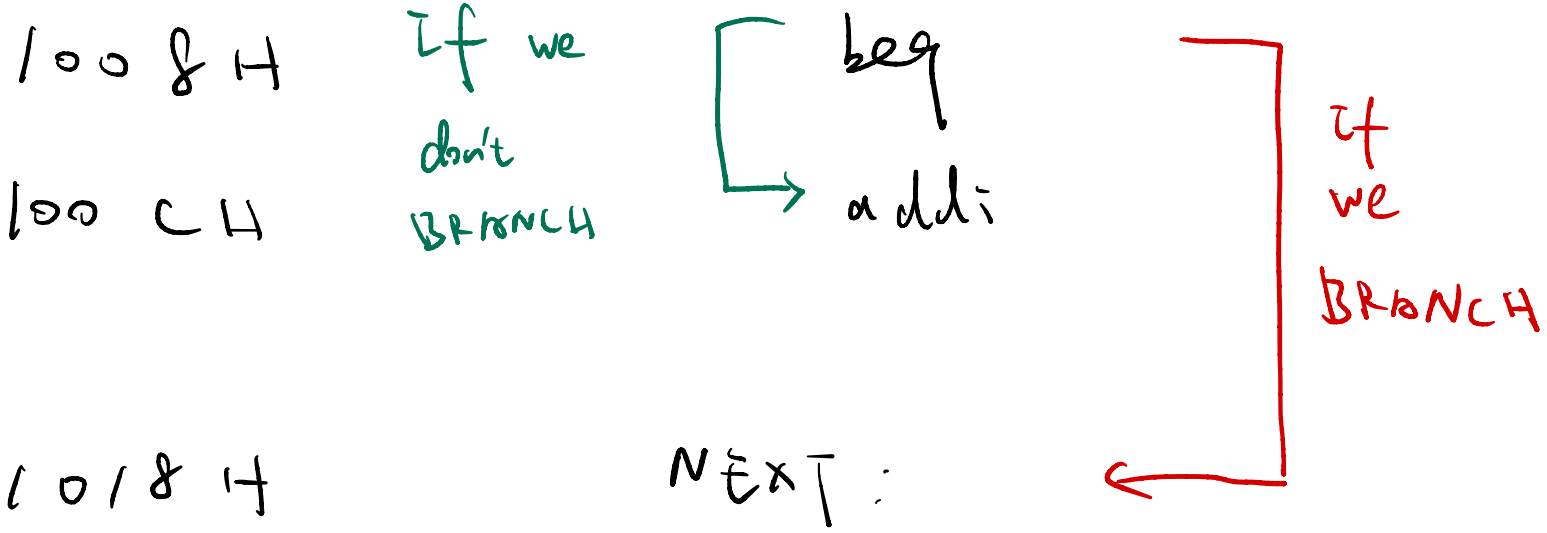
Assume we have the following MIPS code, starting at memory address 1000H:

1000H	LOOP:	addi	$t1, $t1, 4
1004H		sub	$t0, $t3, $t4
1008H		beq	$t0, $zero, NEXT
100CH		addi	$t2, $t2, -1
1010H		bne	$t2, $zero, LOOP
1014H	DONE:	sw	$t0,16($s0)
1018H	NEXT:	add	$s0, $s1, $t0



If we use PC relative addressing for loop, what is the constant for **NEXT** in the **beq** instruction?

beq $t0, $zero, NEXT what is this?



PC RELATIVE EQUATION:

TARGET ADDRESS = PC + 4 (OFFSET)

If we DON'T BRANCH,

ADDRESS OF NEXT INSTRUCTION

= 100 CH ⇒ PC

IF We DO BRANCH, ADDRESS OF

NEXT INSTRUCTION = 101 2H

⇒ TARGET ADDRESS

WHAT IS OFFSET? ITEM WE PUT IN "NEXT"

FOR beq.

TARGET ADDRESS = PC + 4 (OFFSET)

1018H 100CH

$$\text{OFFSET} = \frac{\text{TARGET ADDRESS} - \text{PC}}{4}$$

$$= \frac{1018H - 100CH}{4}$$

$$\begin{array}{r} 1018H \\ -\ 100CH \\ \hline 18H \end{array} = -CH = \begin{array}{r} 24_{10} \\ -\ 12_{10} \\ \hline 12 \end{array}$$

$$\text{OFFSET} = \frac{12}{4} = \boxed{3}$$

→ skip 3 lines.

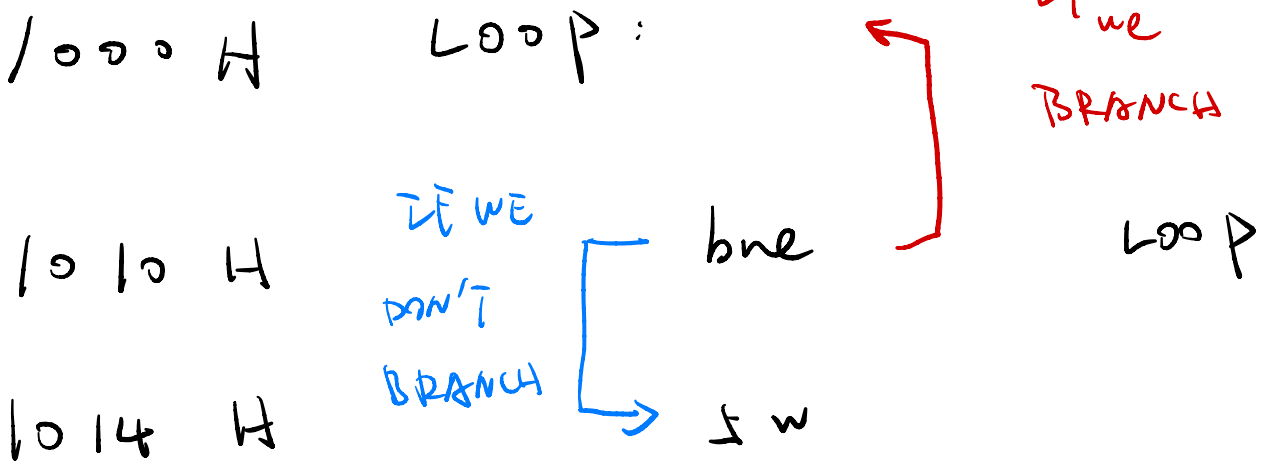
make sense.

Example A.9-5

Assume we have the following MIPS code, starting at memory address 1000H:

```
1000H   LOOP:   addi   $t1, $t1, 4
1004H           sub    $t0, $t3, $t4
1008H           beq    $t0, $zero, NEXT
100CH           addi   $t2, $t2, -1
1010H           bne    $t2, $zero, LOOP
1014H   DONE:   sw     $t0,16($s0)
1018H   NEXT:   add    $s0, $s1, $t0
```

If we use PC relative addressing for loop, what is the constant for **LOOP** in the **bne** instruction?



TARGET: 1000 H PC: 1014 H

OFFSET = (TARGET - PC) / 4

1000 H - 1014 H

14 H

14 H = 20$_{10}$ ⇒ -20

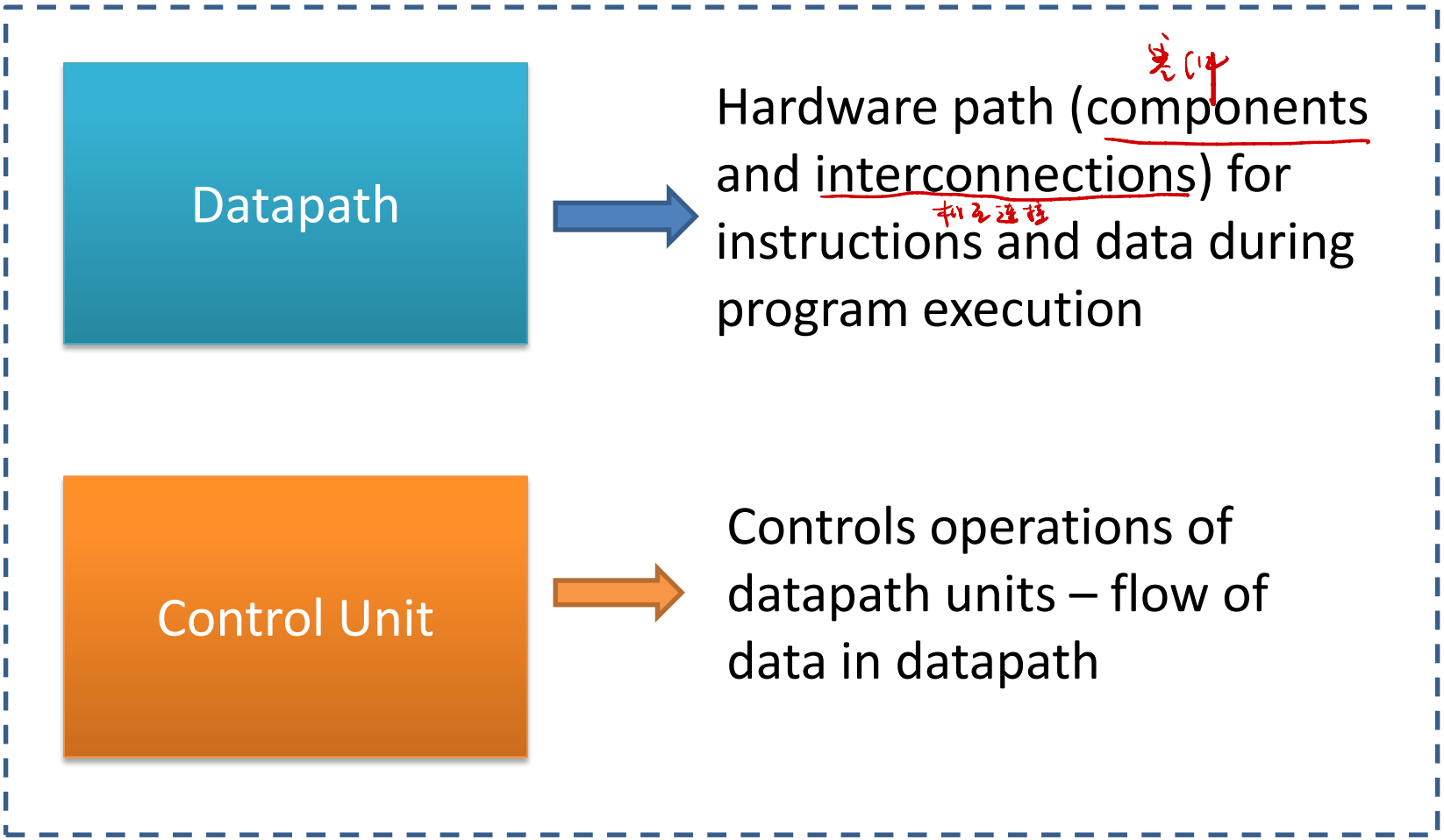
-00 H

$$\text{OFFSET} = \frac{-20}{4} = \boxed{-5}$$

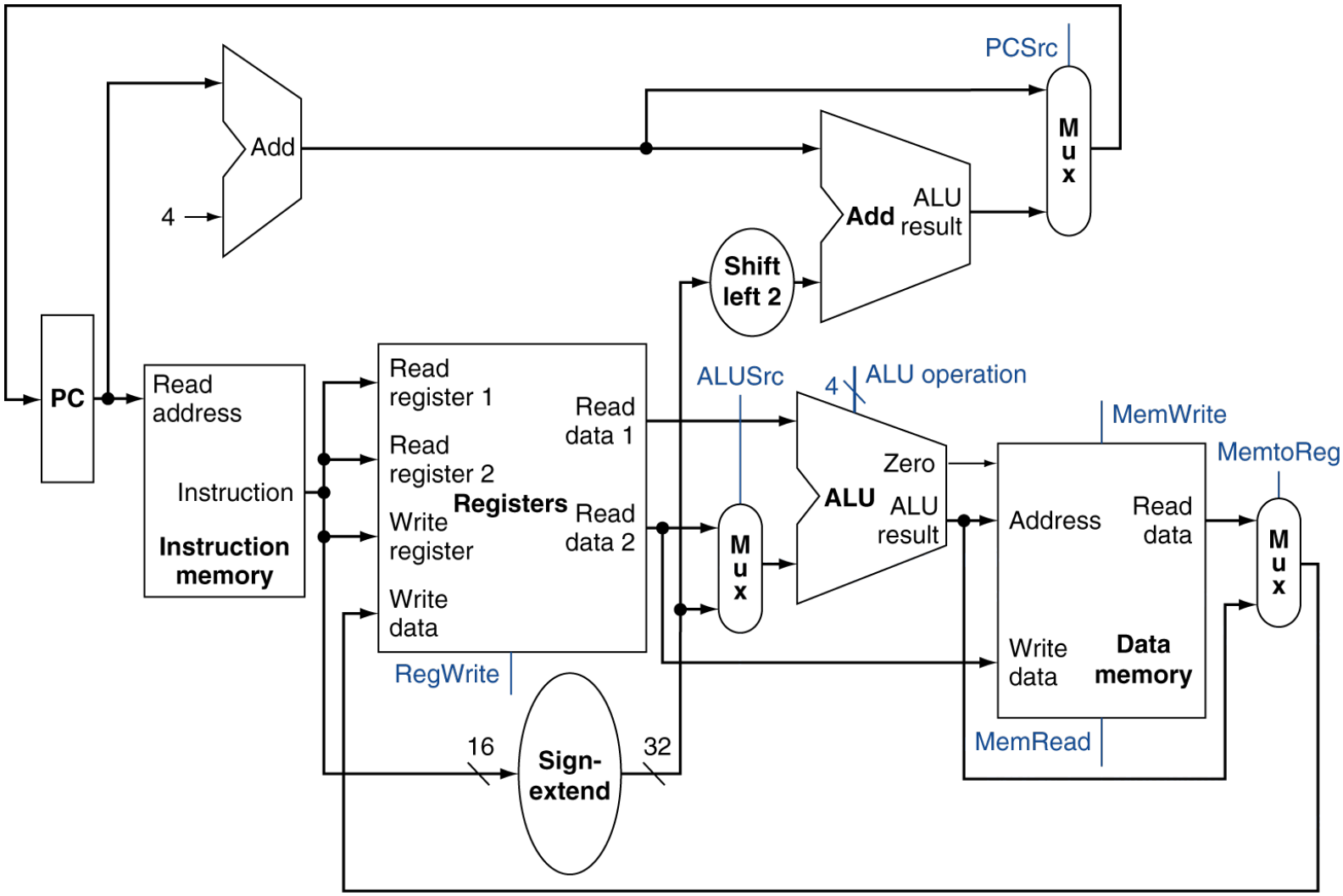
Outline

- A.9 MIPS Architecture
  - MIPS Instruction Set
  - MIPS Processor Design

Processor Units



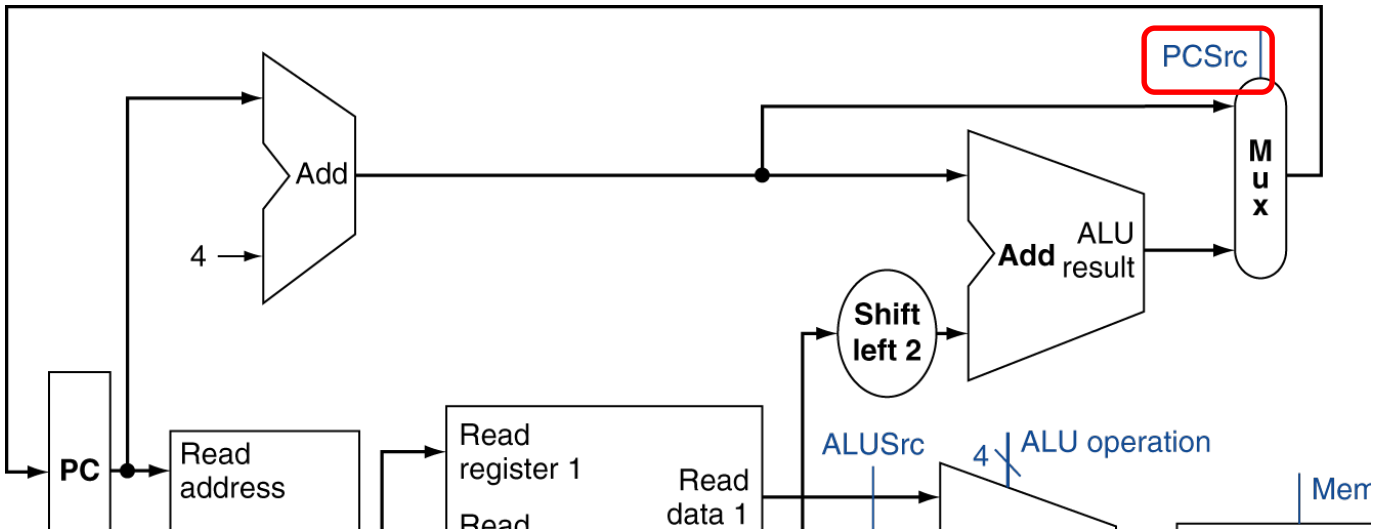
MIPS Datapath



Datapath Control Signals

- What are our basic control signals?
  - PCSrc
  - RegWrite
  - ALUSrc
  - MemWrite
  - MemRead
  - MemtoReg
  - ALU operation (4 bits)

PCSrc

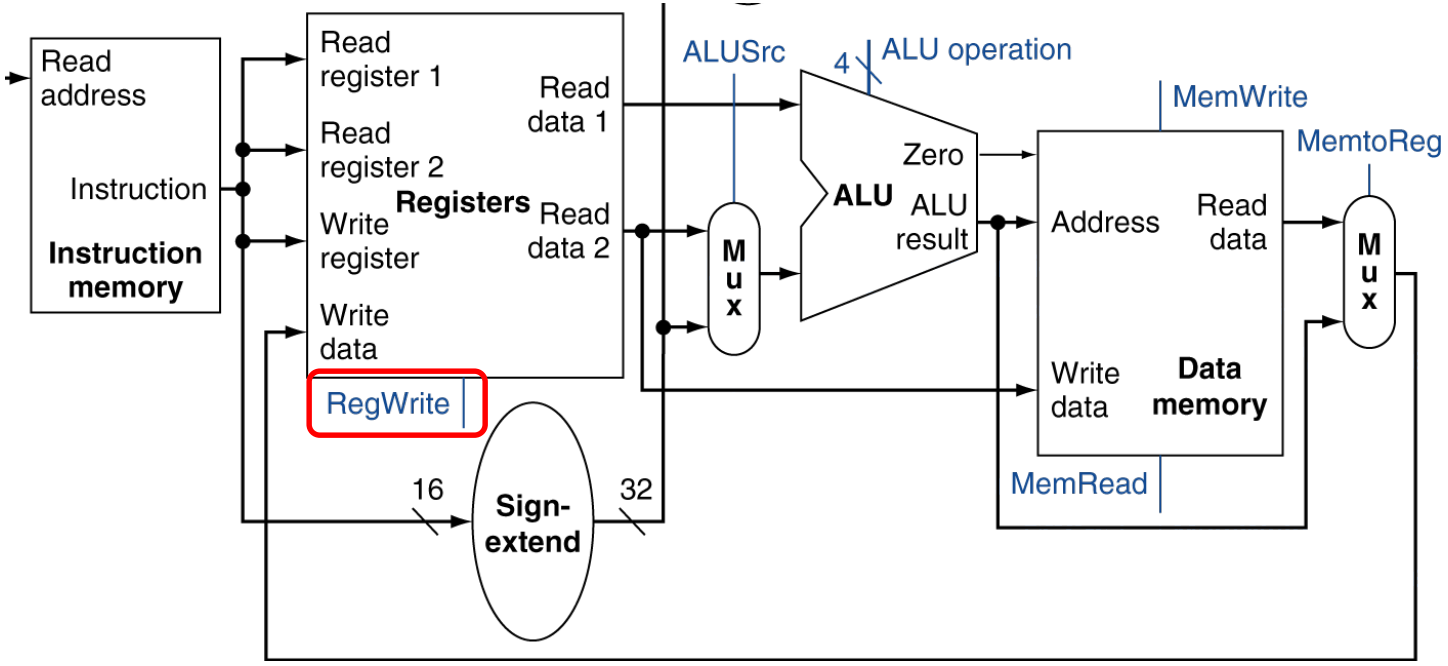


zero
We're not branching

Value	Effect
0	PC <= PC + 4
1	PC <= PC + 4 + offset

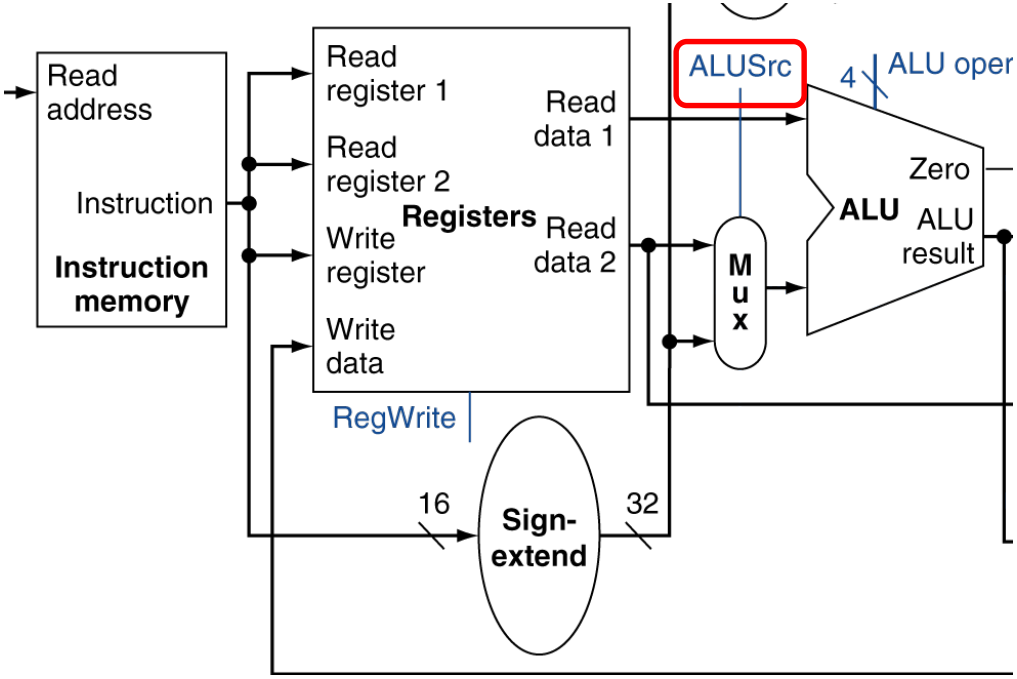
We're branching

RegWrite



Value	Effect
0	none
1	R[Write register] <= Write data

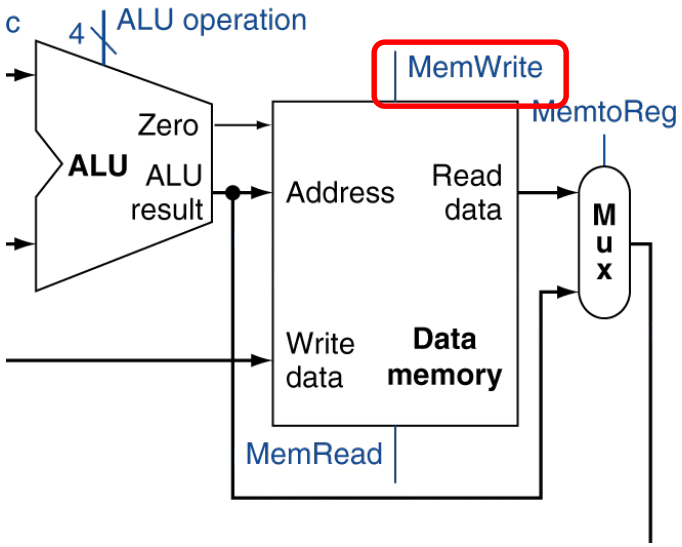
ALUSrc



Value	Effect
0	2nd ALU operand <= Read data 2
1	2nd ALU operand <= Sign-extended constant

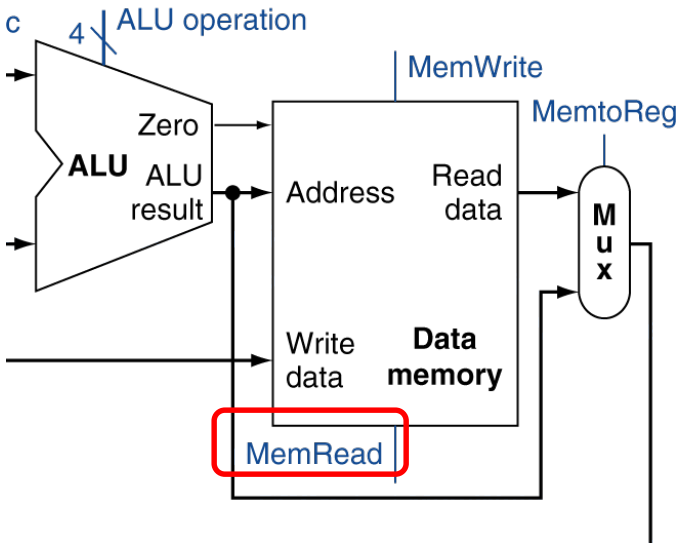
Here are some instructions for data memory.

MemWrite



Value	Effect
0	none
1	Mem[Address] <= Write data

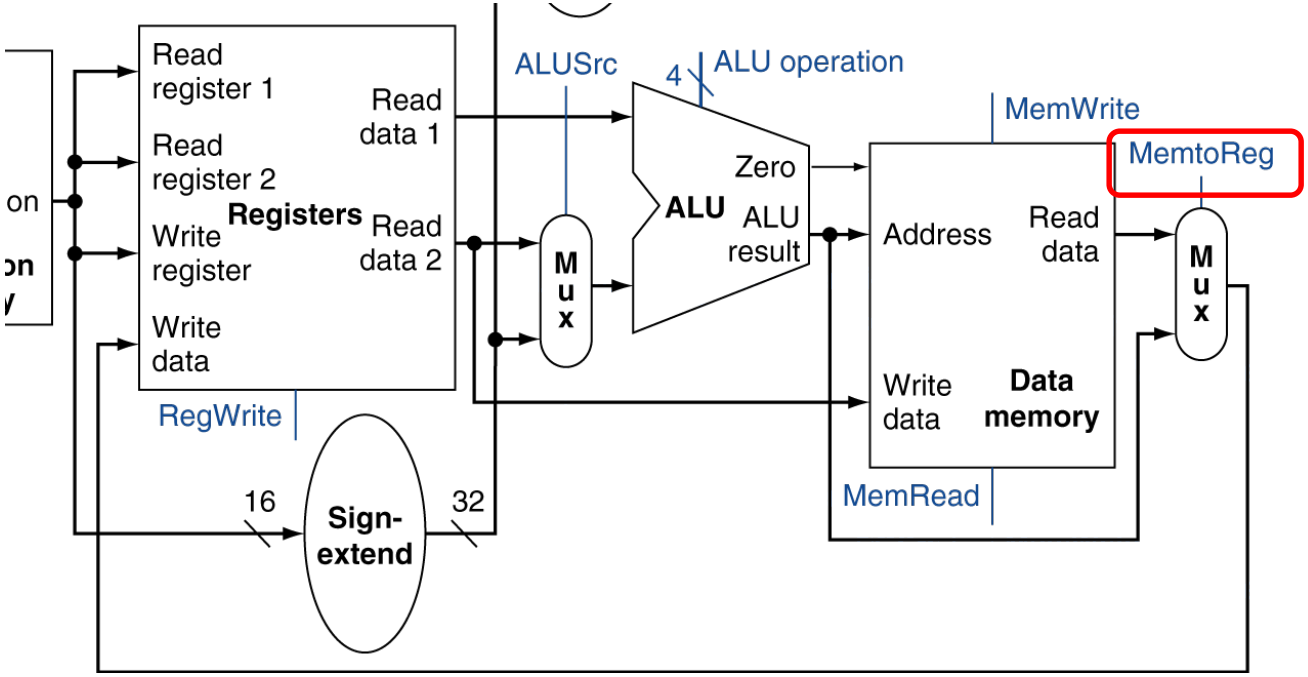
MemRead



load word

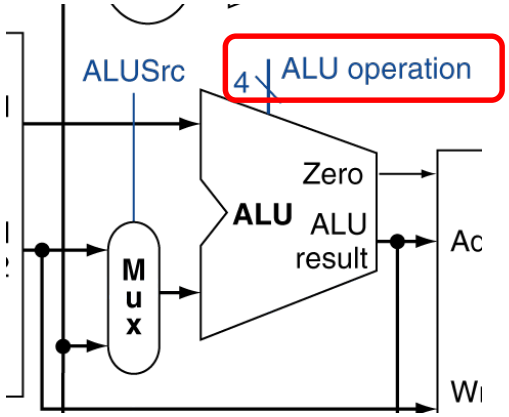
Value	Effect
0	none
1	Read data <= Mem[Address]

MemtoReg



Value	Effect
1	Write data <= Read data
0	Write data <= ALU result

ALU operation (4 bits)



Note that the ALU operation control has 4 bits – what are the specific bit patterns and operations?

ALU Control

- ALU used for
  - Load/Store: F = add
  - Branch: F = subtract
  - R-type: F depends on funct field

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

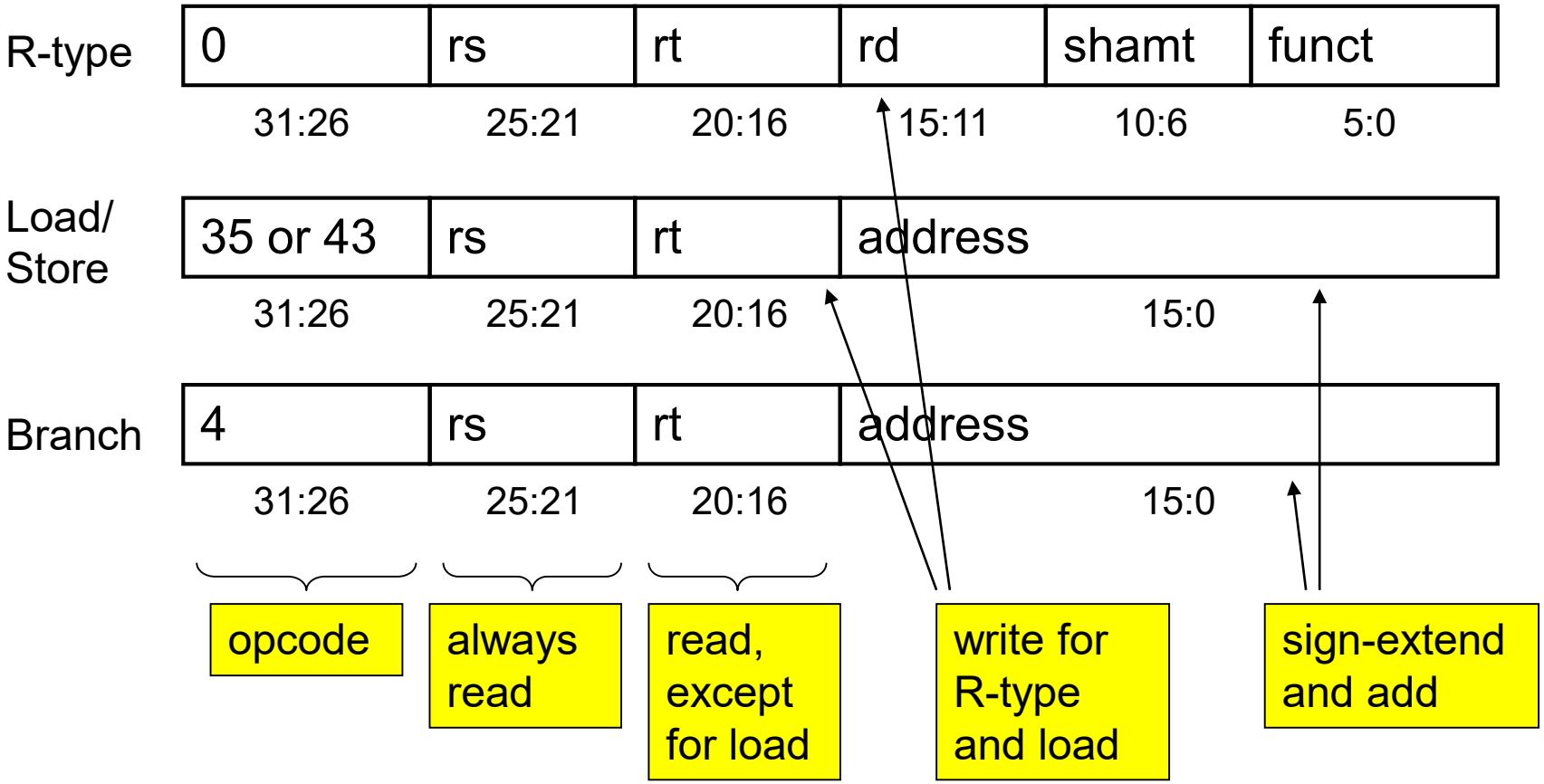
ALU Control

- Assume 2-bit ALUOp derived from opcode
  - Combinational logic derives ALU control

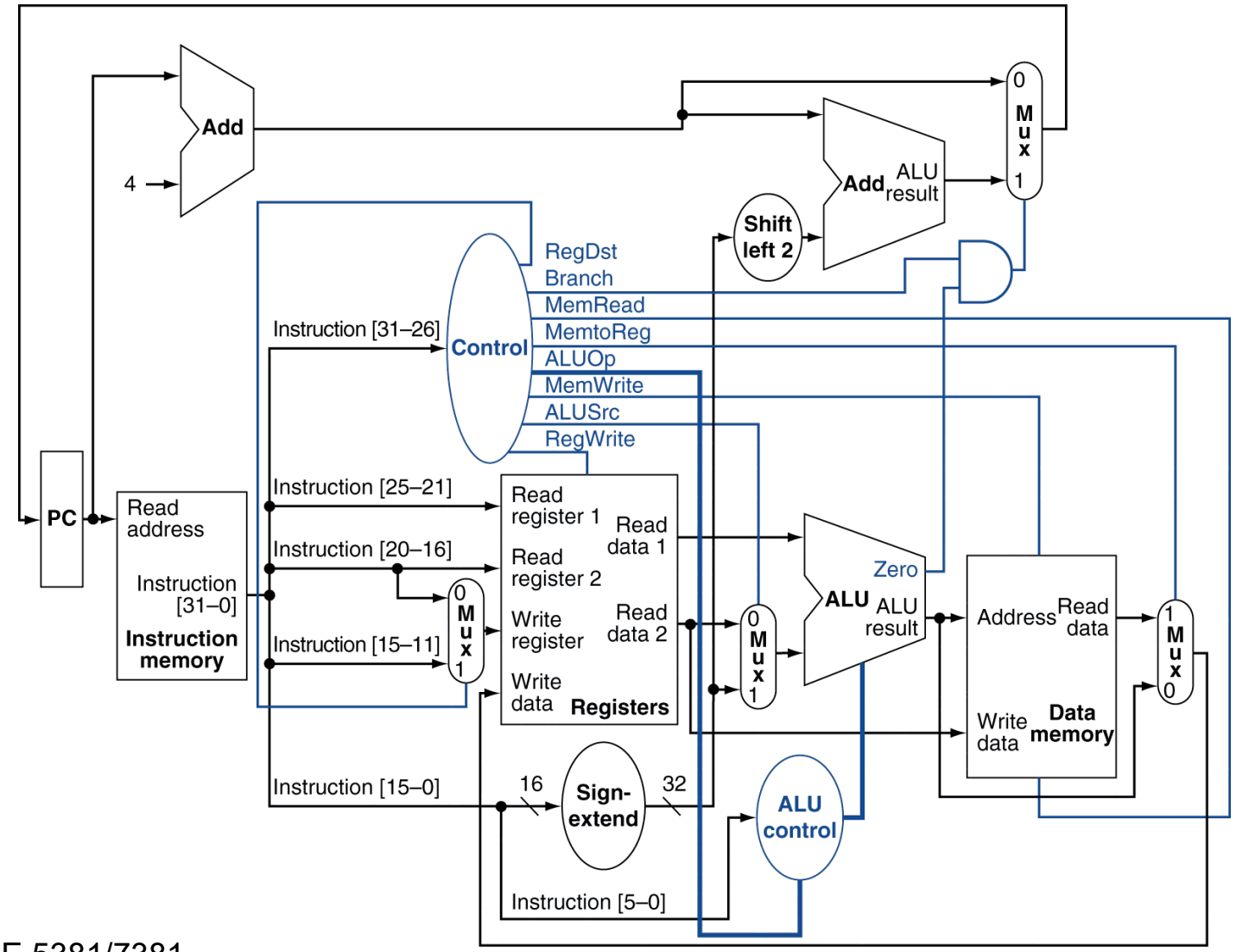
opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

The Main Control Unit

- Control signals derived from instruction



Datapath With Control

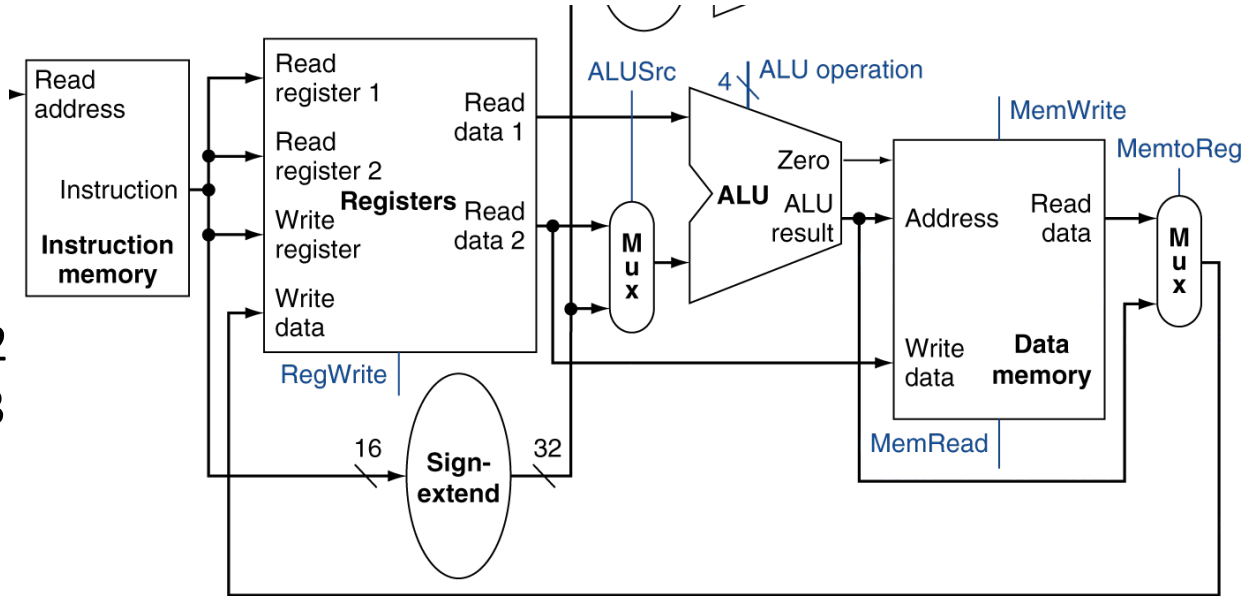


Simple Implementation Scheme

- We will look at datapath operations and controls for three MIPS instructions
  1. add
  2. lw
  3. beq

add $t1,$t2,$t3

Read Register 1 rs = $t2
Read Register 2 rt = $t3
Write Register rd = $t1

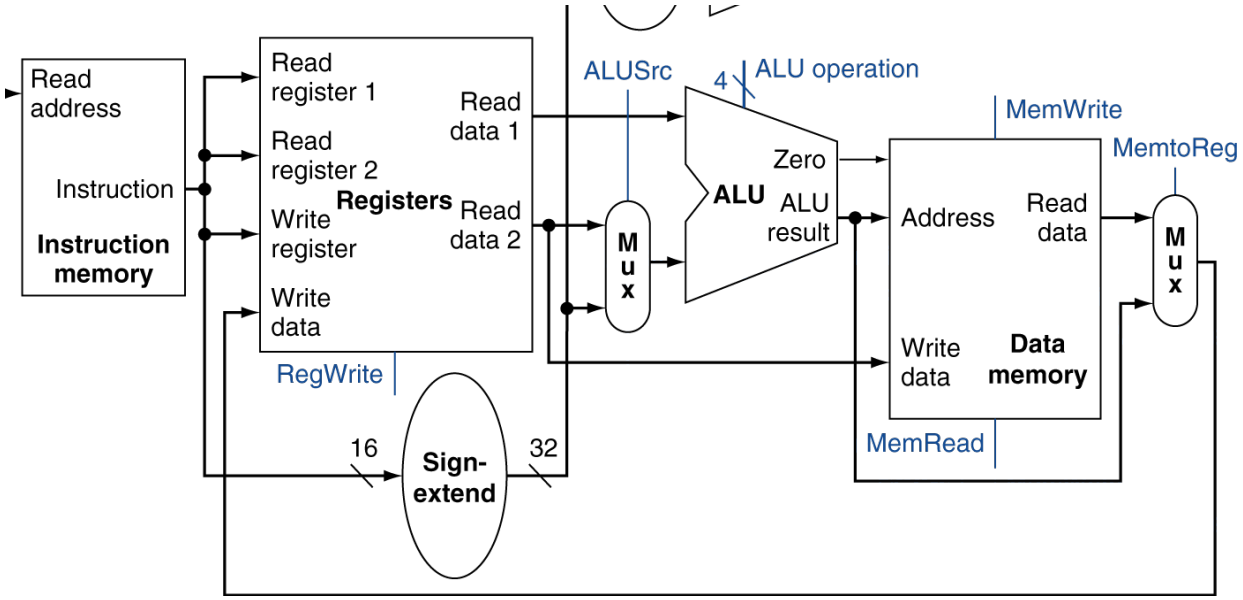


Control signal **RegDst** = 1; result stored in register specified by rd

ALUSrc = 0; second ALU input is Read data 2

ALU Operation = 0010; add

add $t1,$t2,$t3

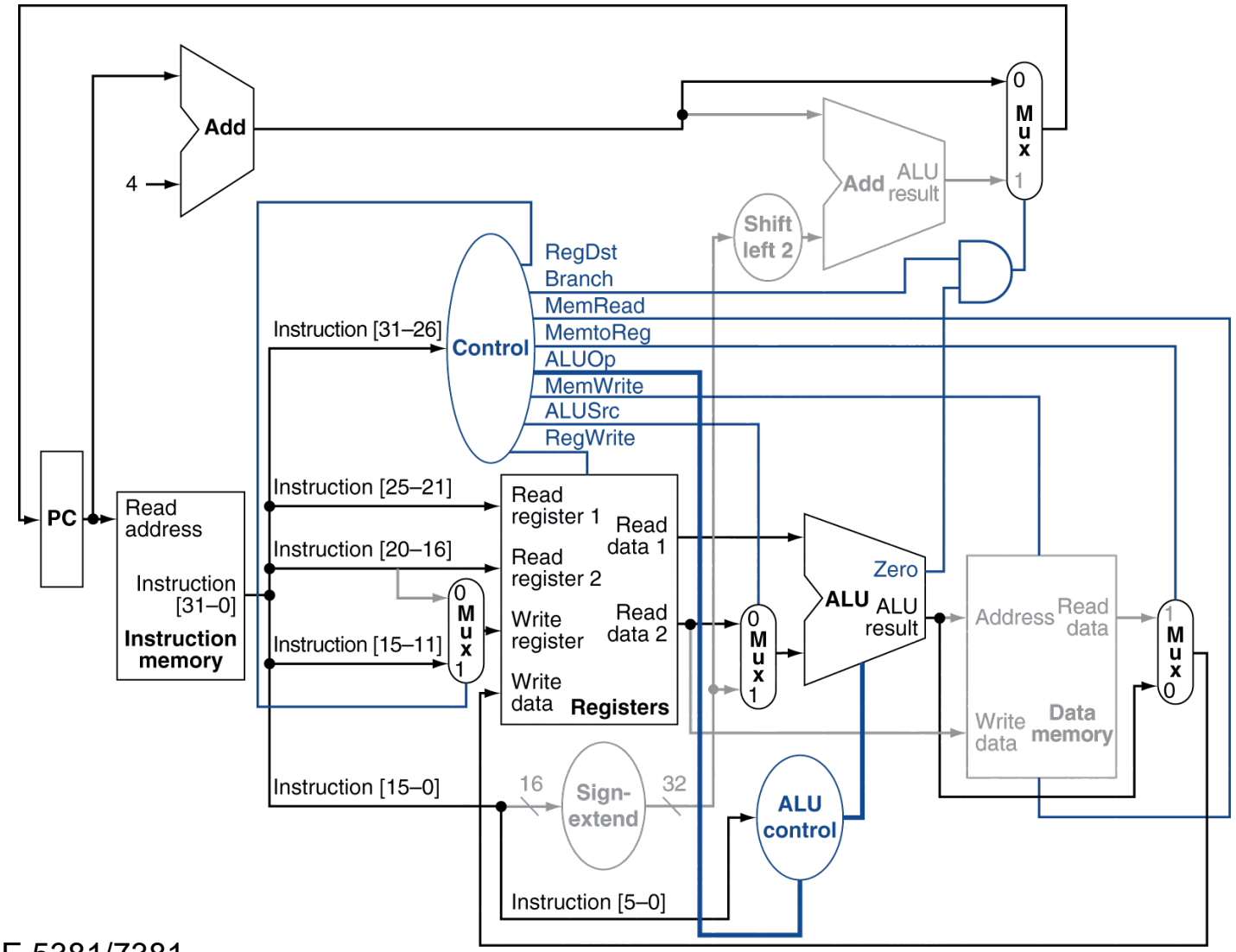


ALU Result = Read Data 1 + Read Data 2

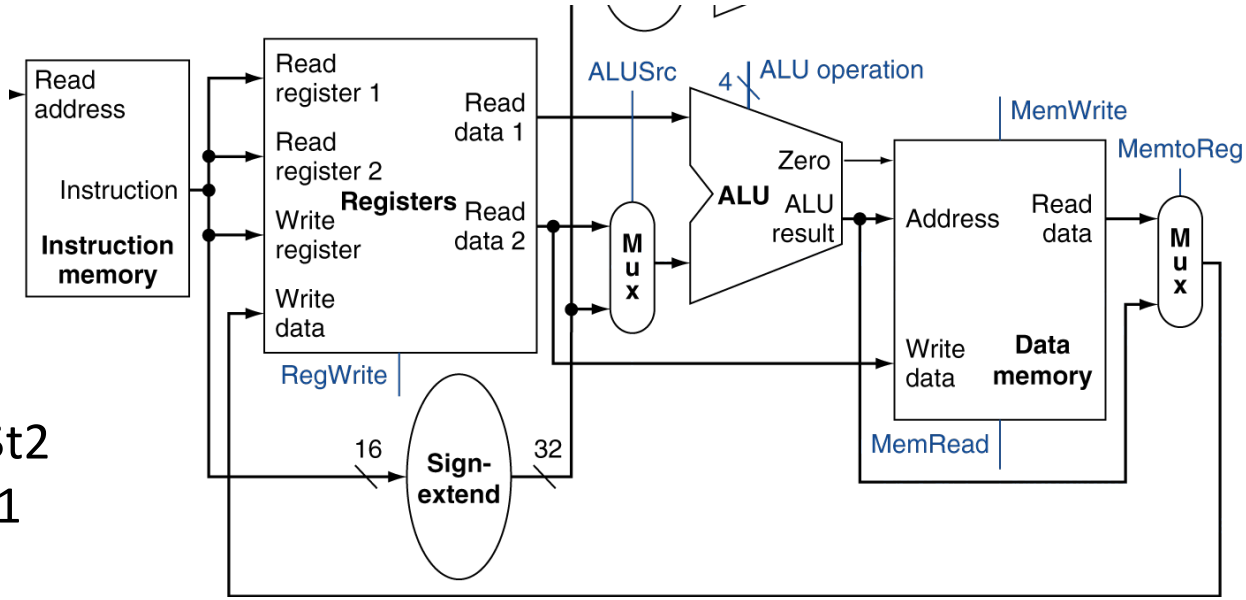
MemtoReg = 0 ALU Result passes back to Write Data in registers

RegWrite = 1 write ALU Result to register specified by rd

R-Type Instruction



lw $t1,OFFSET($t2)



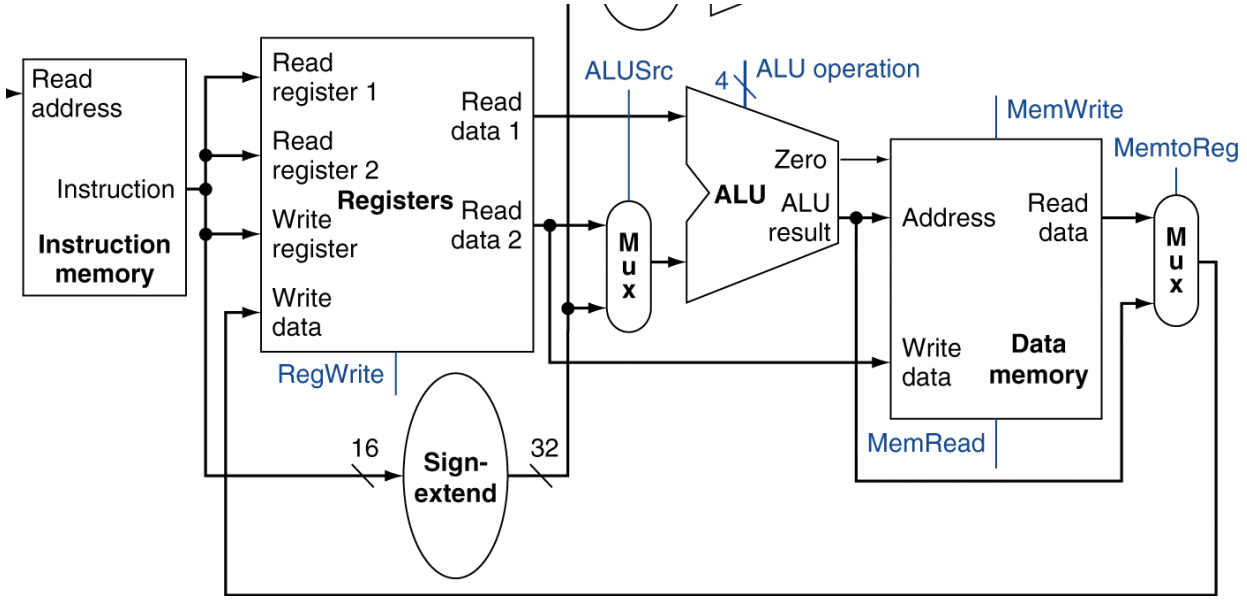
Read Register 1 rs = $t2
Write Register rt = $t1

Control signal **RegDst** = 0 result stored in register specified by rt

ALUSrc = 1 second ALU input is Sign-Extended address from instruction

ALU Operation = 0010 add

lw $t1,OFFSET($t2)



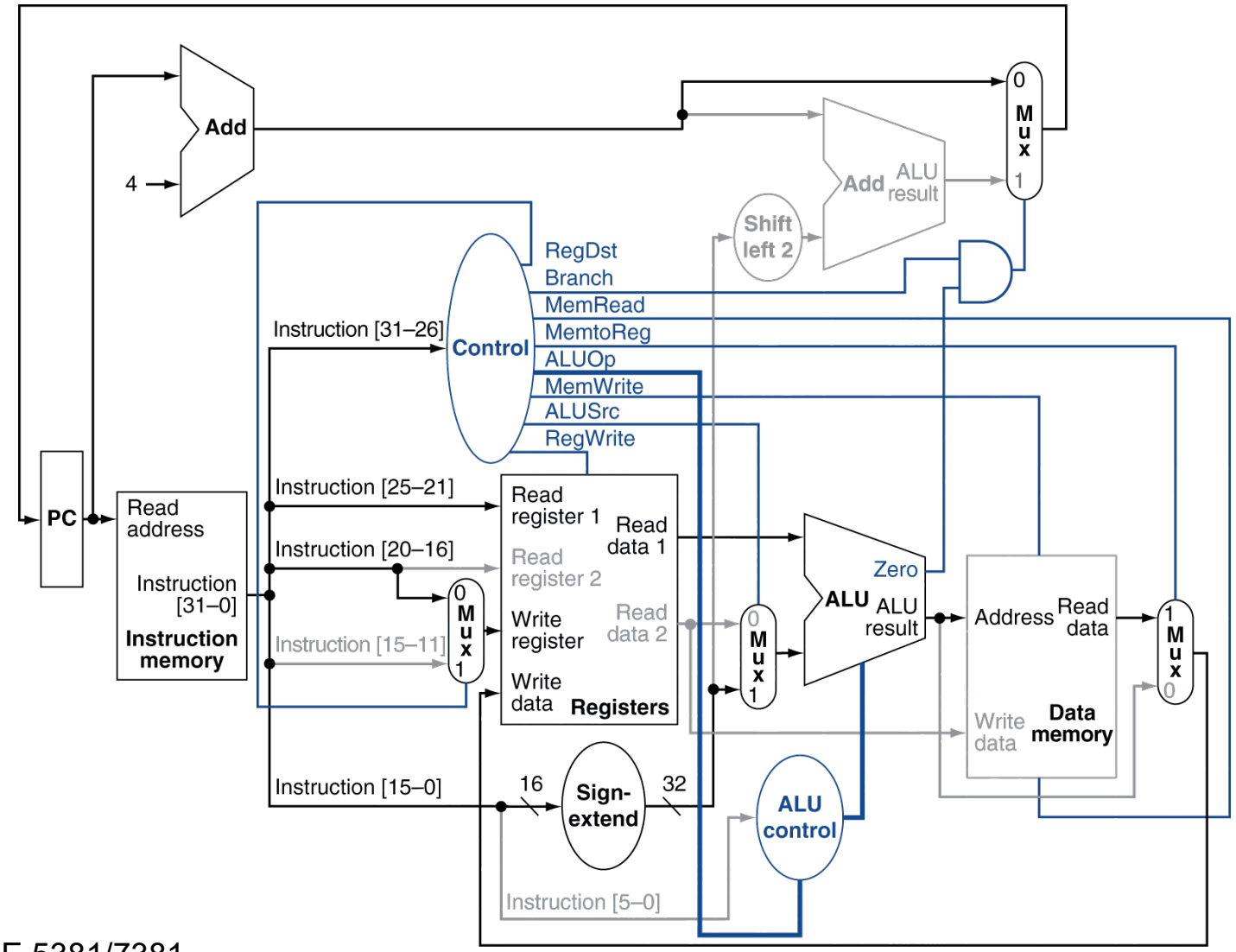
ALU Result = Read Data 1 + sign-extended OFFSET value

MemRead = 1 Read data from Data Memory

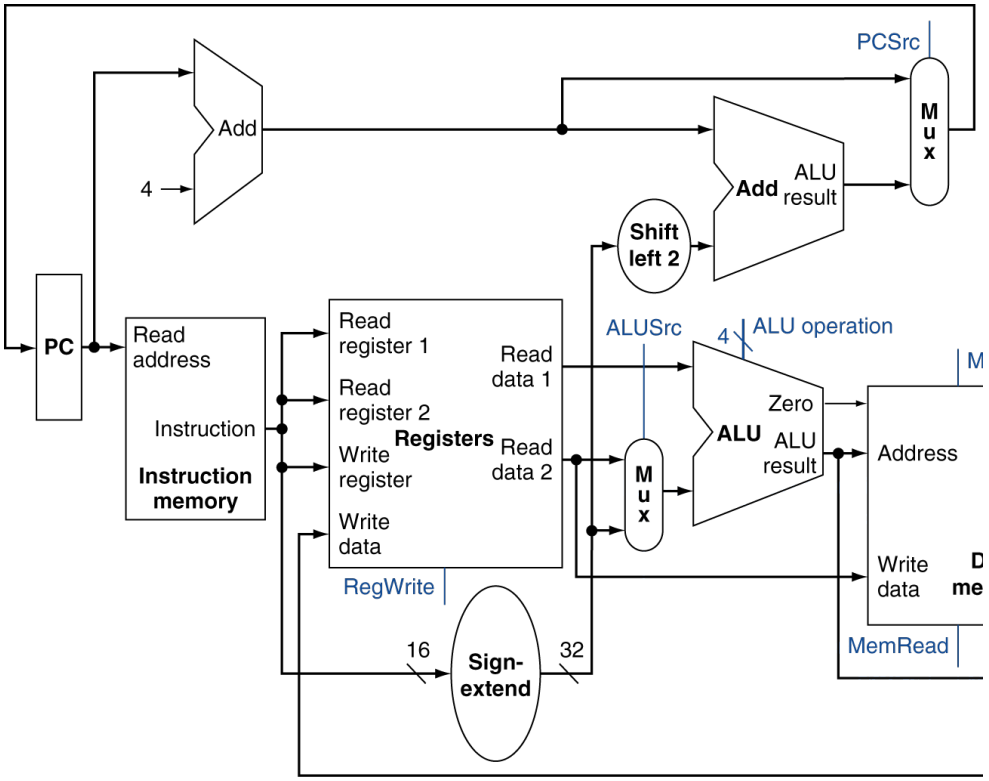
MemtoReg = 1 ALU Result becomes Data Memory address. Read Data passes back to Write Data in registers.

RegWrite = 1 write ALU Result to register specified by rt

Load Instruction



beq $t1, $t2, OFFSET



Read Register 1 rs = $t1
Read Register 2 rt = $t2

ALUSrc = 0 second ALU input is Read Register 2

ALU Operation = 0110 sub

beq $t1, $t2, OFFSET

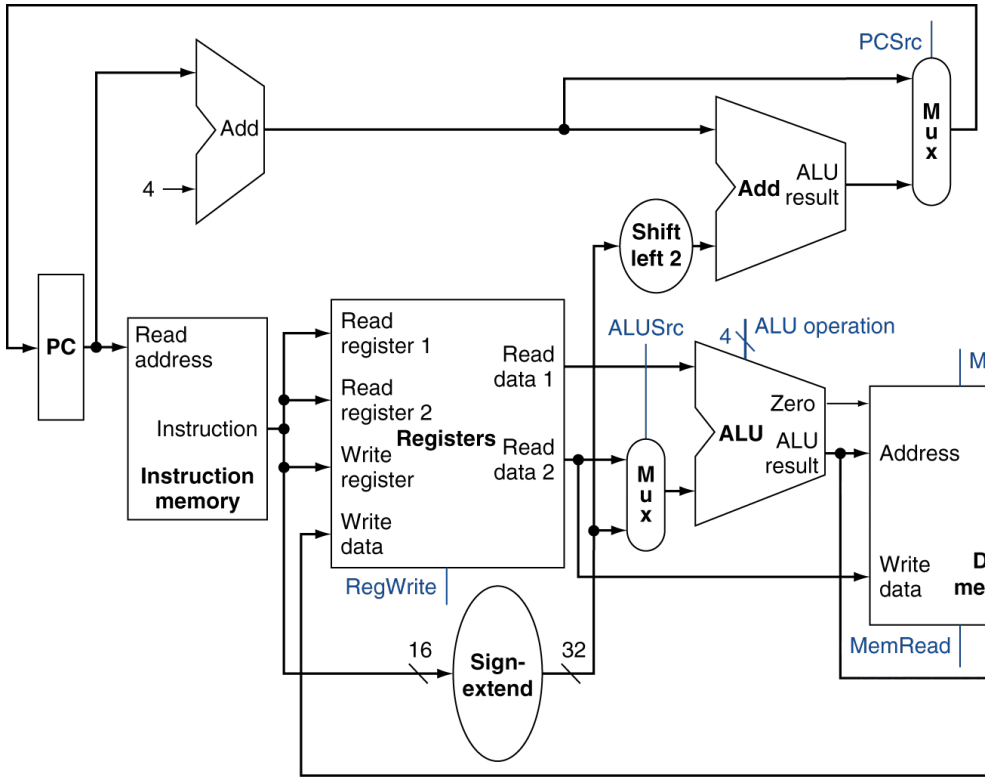
Add ALU gets PC + Sign-extended OFFSET shifted left by 2

If R[t1] = R[t2], ALU Zero = 1 and PCSrc = 1

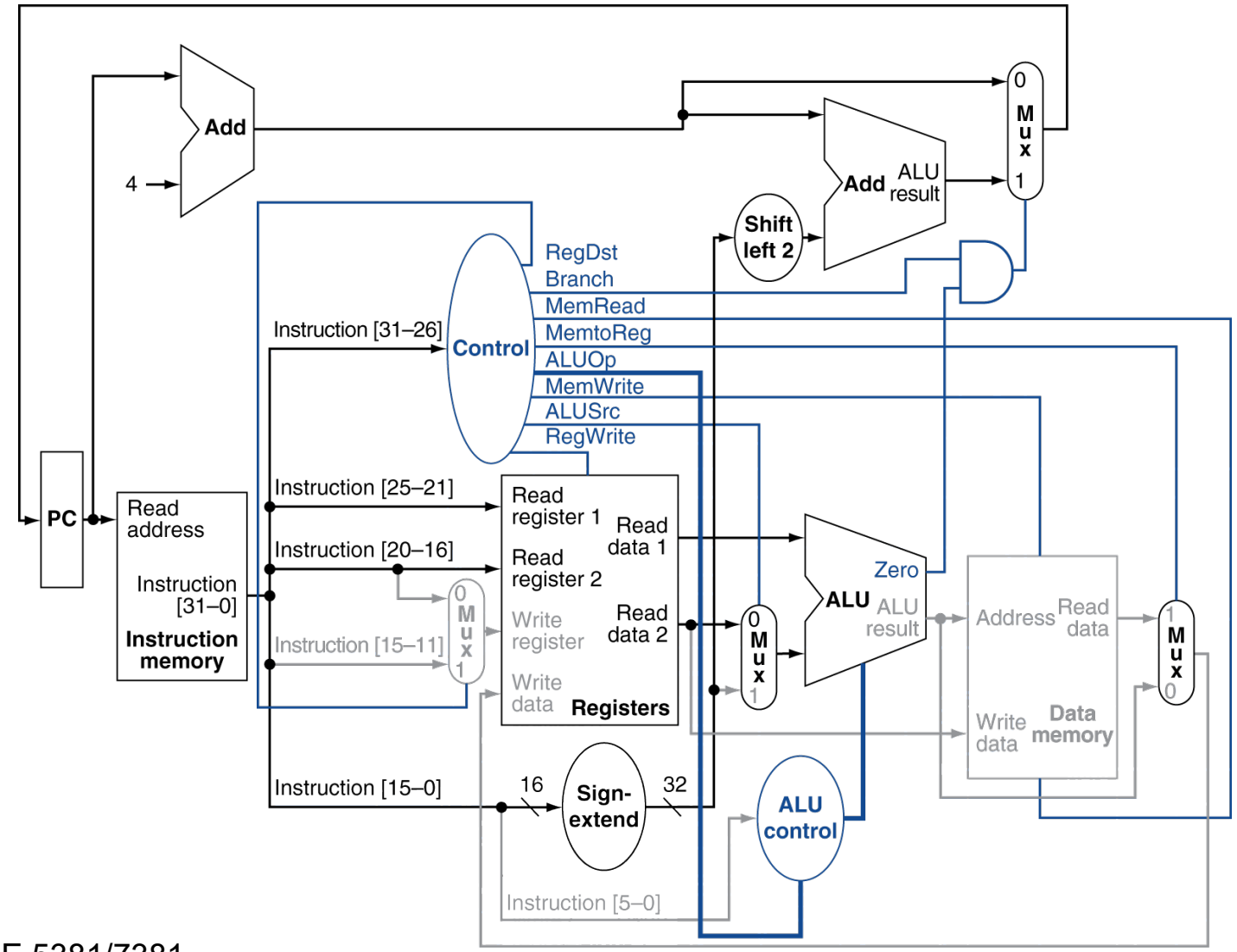
PC ← Add ALU result

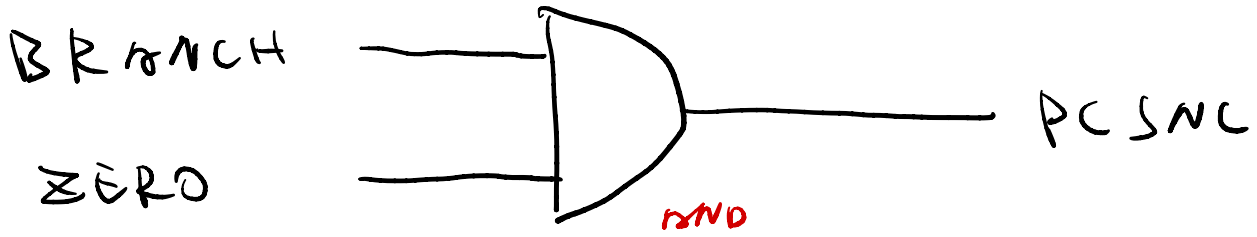
Else,PCSrc = 0

PC ← PC + 4



Branch-on-Equal Instruction





IS THIS A BRANCH INSTRUCTION?

BRANCH	ZERO	PC SNC
0	0	0
0	1	0
1	0	0
1	1	1

NOT A BRANCH INSTRUCTION

RESULT

beq : rs ≠ rt

beq : rs = rt

Branch condition

ALU RESULT

BRANCH

Pipelining: Basic and Intermediate Concepts

(Appendix C, Hennessy and Patterson)
Note: some course slides adopted from publisher-provided material