

CS/ECE 5381/7381
Computer Architecture
Spring 2023

Dr. Manikas

Computer Science

Lecture 5: Feb. 7, 2023

Instruction Set Principles

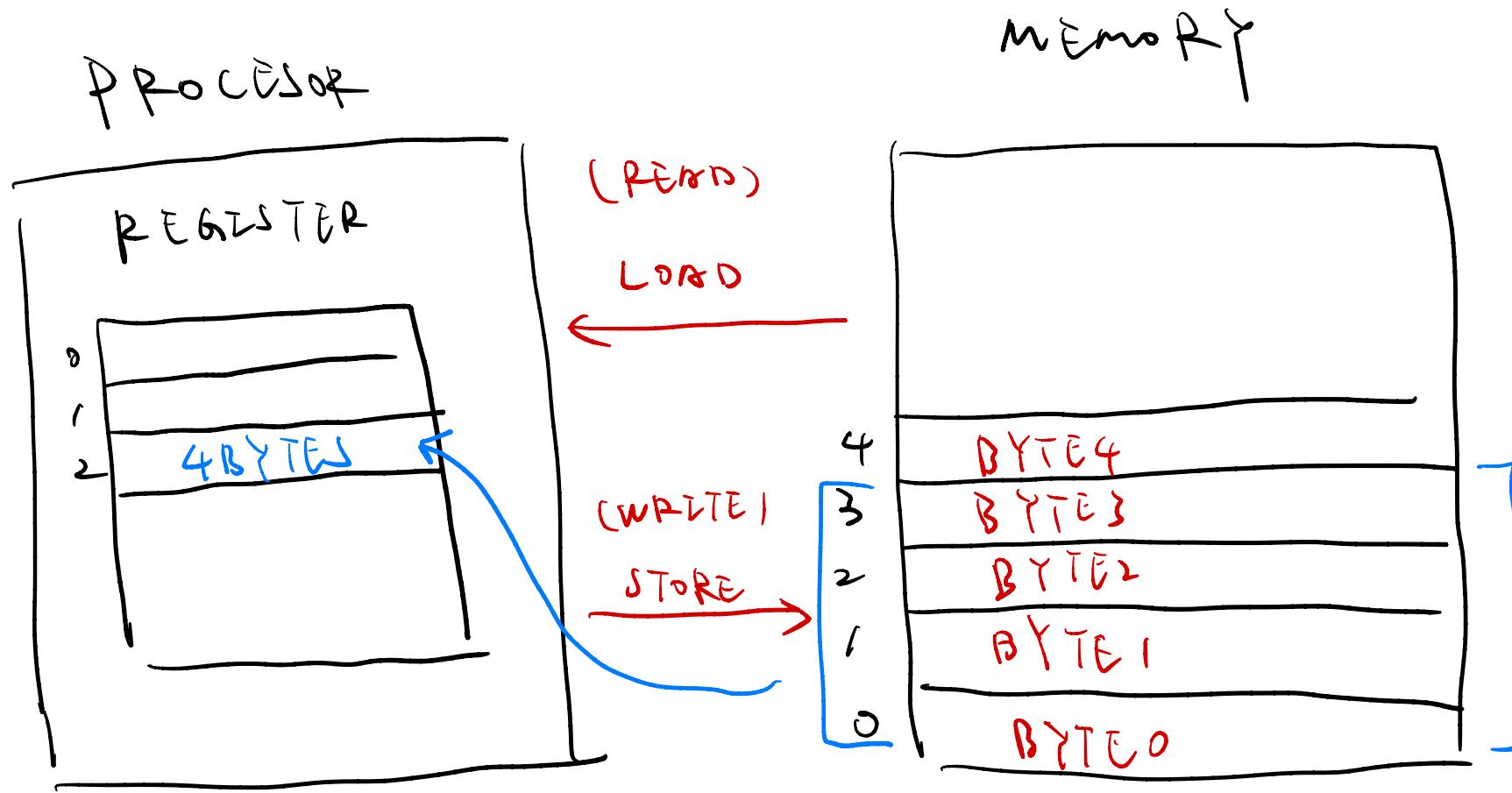
(Appendix A, Hennessy and Patterson)

Note: some course slides adopted from
publisher-provided material

Outline

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

REMINDER : MEMORY DATA TRANSFERS .



MIPS REGISTER

SIZE = 32 BITS

= 4 BYTES

Memory Operands

정수 -

- Main memory used for composite data
 - Arrays, structures, dynamic data
- To apply arithmetic operations
 - Load values from memory into registers
 - Store result from register to memory
- Memory is byte addressed
 - Each address identifies an 8-bit byte
- Words are aligned in memory
 - Address must be a multiple of 4
- MIPS is BigEndian
 - Most-significant byte at least address of a word
 - c.f. Little Endian: least-significant byte at least address

Memory Operand Example 1

- C code:

g = h + A[8];

- g in \$s1, h in \$s2, base address of A in \$s3

- Compiled MIPS code:

$\$s0 - \$s7 \quad 16-23$ saved Temporaries

- Index 8 requires offset of 32

- 4 bytes per word

lw \$t0, 32(\$s3) # load word
add \$s1, \$s2, \$t0

$\$t0 - \$t7 \quad 8-15$ Temporaries

$\$t8 - \$t9 \quad 24-25$ Temporaries.

临时变量；常数表达式

$g = h + \underline{\alpha[8]}$ C/C++ / JAVA
 ↑ ↑ ↓
 Register memory location

NEED TO **Read** VALUE $\alpha[8]$ FROM MEMORY

(LOAD) INTO MIPS REGISTER

↑
 32 BITS
 ↓

MIPS INSTRUCTION : **lw** (LOAD WORD)

(DESTINATION) (BASE REGISTER)

Syntax: **lw** TARGET REGISTER, OFFSET

TARGET (DESTINATION) REGISTER

- MIPS REGISTER WHERE WE ARE

LOADING WORD FROM MEMORY

BASE REGISTER

- REGISTER THAT CONTAINS ADDRESS OF

FIRST ELEMENT IN ARRAY = $a[0]$

OFFSET

VALUE ADDED TO BASE ADDRESS

TO FIND OUR ARRAY ELEMENT ($a[8]$)

We want $\alpha[8]$

Assume we are given that

- base address of α is stored

in register $\$s3$

- we want to load $\alpha[8]$ value

in register $\$t0$

\Rightarrow to $\$t0$, OFFSET($\$s3$) ??

OFFSET FOR $\&i[8]$?

$\&i[i]$

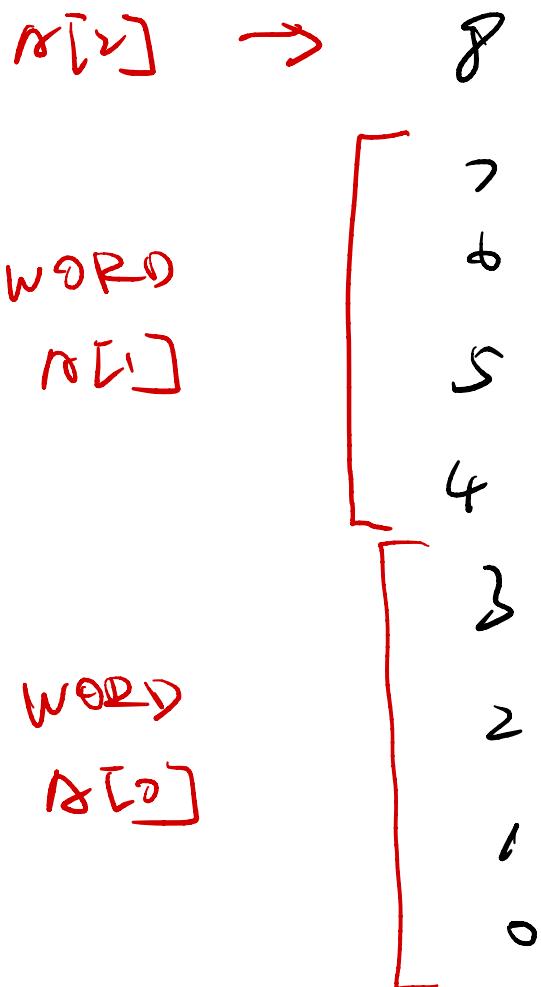
OFFSET

$= 4i$

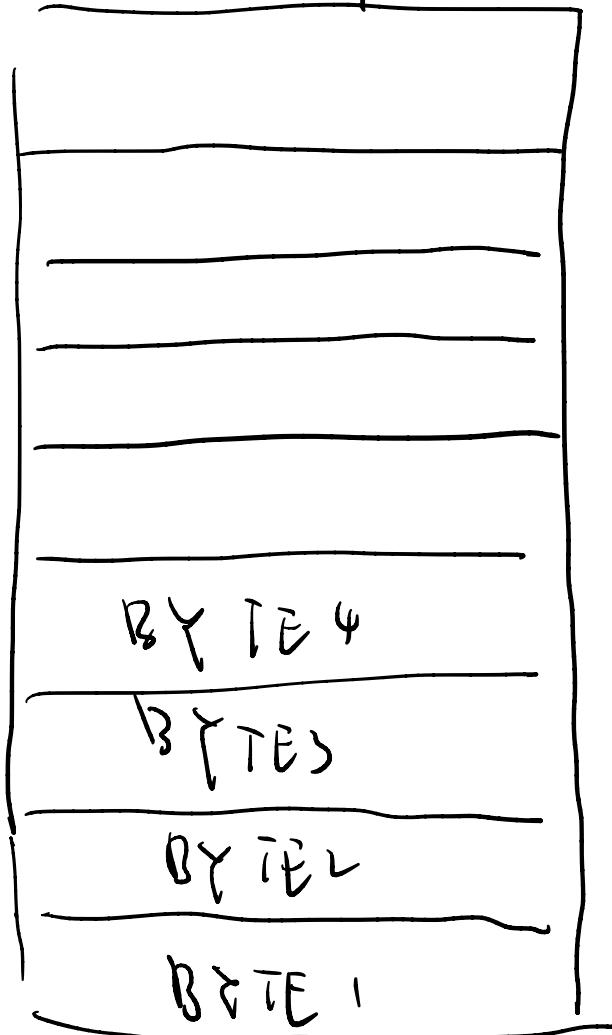
$\&i[8]$

OFFSET = 4 · 8

$= 32$



MEMORY



To send $r[8]$ into REGISTER $$t0$

$\Rightarrow lw \ $t0, 32($s3)$

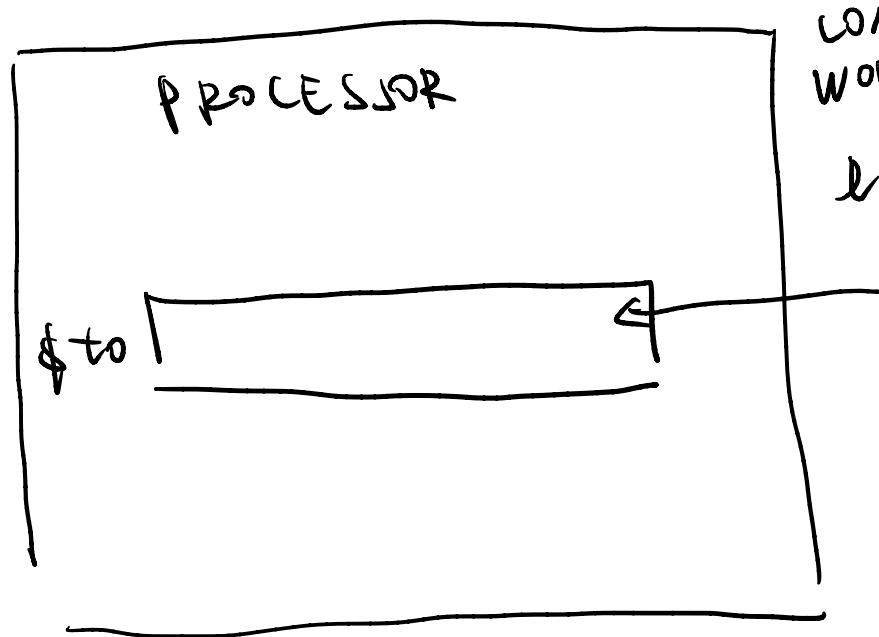
$\uparrow \quad \uparrow$

ADDRESS OF $r[0]$

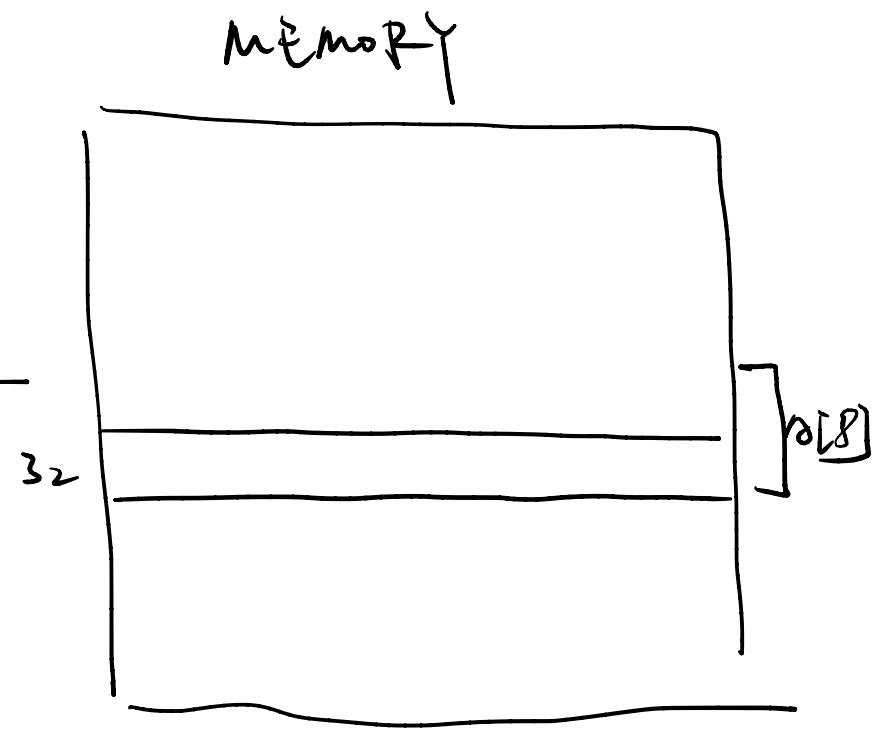
OFFSET TO GET $r[8]$

MIPS
DESTINATION

REGISTER

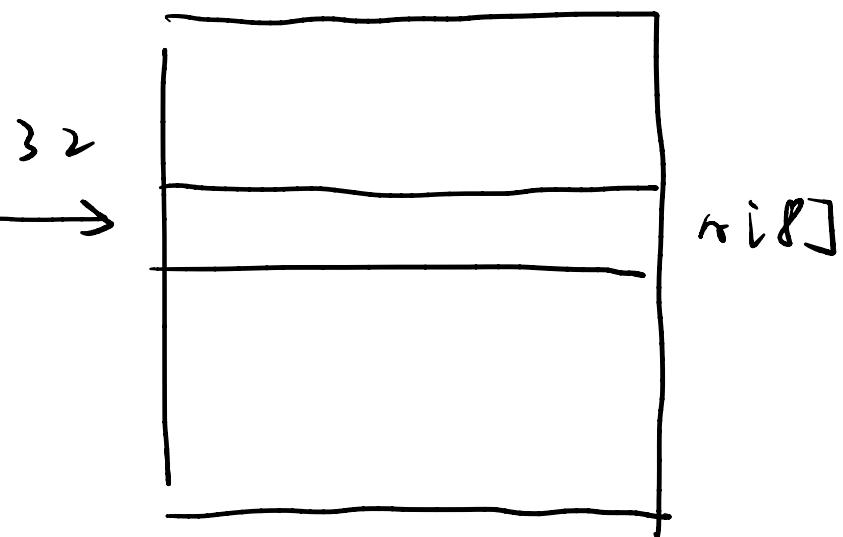


WORD
WORD
 lw



OTHER DIRECTION?

STORE WORD (sw)



Syntax: sw \$t0, 32(\$s3)

Memory Operand Example 2

- C code:

A[12] = h + A[8];

– h in \$s2, base address of A in \$s3

- Compiled MIPS code:

– Index 8 requires offset of 32

```
lw    $t0, 32($s3)      # load word  
add $t0, $s2, $t0  
sw    $t0, 48($s3)      # store word
```

for [var] = h + K[8];
↑
OFFSET
= 4 × 8 = 48
[offset = $4 \times 8 = 32$
Read from memory : lW

WRITE TO MEMORY : sw

Immediate Operands

- Constant data specified in an instruction

```
addi $s3, $s3, 4
```

- No subtract immediate instruction

— Just use a negative constant

```
addi $s2, $s1, -1
```

IMMEDIATE OPERAND

C/C++ / JAVA : $i++;$

($i = i + 1$) (LOAD $\$1 = i$)

MIPS : addi \$s1, \$s1, 1

$i--;$ ($i = i - 1$)

MIPS : addi \$s1, \$s1, -1

add vs addi

SECOND
OPERAND

add \$t0, \$t1, \$t2

VOLUME REGD
FROM REGISTER

can change

addi \$t0, \$t0, 1

CONSTANT

The Constant Zero

- MIPS register 0 (\$zero) is the constant 0
 - Cannot be overwritten
- Useful for common operations
 - E.g., move between registers
`add $t2, $s1, $zero`

CONSTANT \$ZERO (=0)

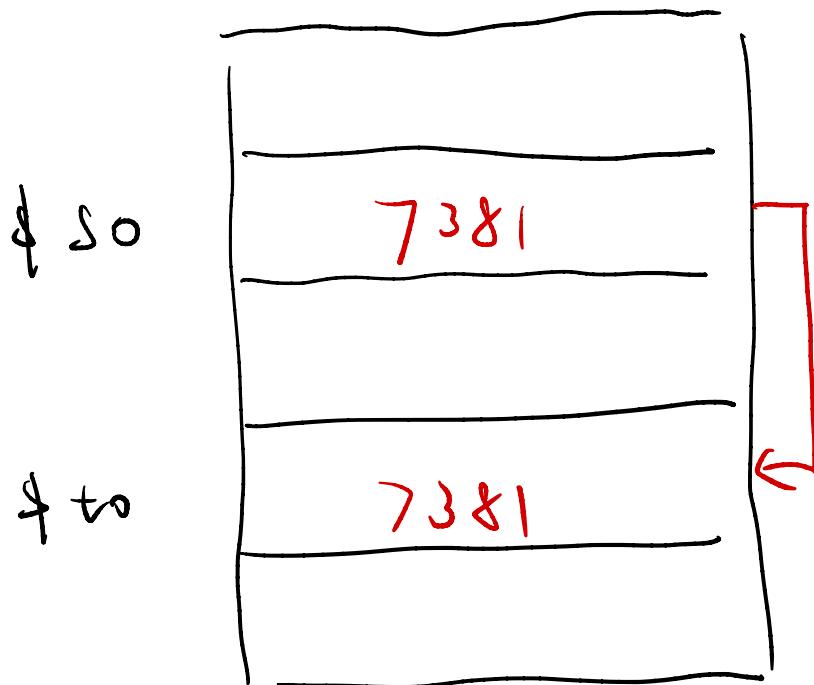
Why

IS THIS USEFUL?

addi \$t0, \$t0,

add \$t0, \$t0, \$zero

$\Rightarrow RL[\$t0] \leftarrow RL[\$t0] + 0$



WHAT TO COPY VALUE
HERE

BUT: MIPS DOES NOT

HAVE A copy INSTRUCTION

2s-Complement Signed Integers

- Given an n-bit number

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

- Range: -2^{n-1} to $+2^{n-1} - 1$
- Example
 - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
 $= -1 \times 2^{31} + 1 \times 2^{30} + \dots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$
 $= -2,147,483,648 + 2,147,483,644 = -4_{10}$
- Using 32 bits
 - $-2,147,483,648$ to $+2,147,483,647$

2s-Complement Signed Integers

- Bit 31 is sign bit
 - 1 for negative numbers
 - 0 for non-negative numbers
- $-(-2^{n-1})$ can't be represented
- Non-negative numbers have the same unsigned and 2s-complement representation
- Some specific numbers
 - 0: 0000 0000 ... 0000
 - -1: 1111 1111 ... 1111
 - Most-negative: 1000 0000 ... 0000
 - Most-positive: 0111 1111 ... 1111

Signed Negation

- Complement and add 1
 - Complement means $1 \rightarrow 0, 0 \rightarrow 1$

$$x + \overline{x} = 1111\dots111_2 = -1$$

$$\overline{x} + 1 = -x$$

- Example: negate +2

- $+2 = 0000\ 0000\dots 0010_2$

- $-2 = 1111\ 1111\dots 1101_2 + \underline{1}$
 $= 1111\ 1111\dots 1110_2$

Sign Extension

- Representing a number using more bits
 - Preserve the numeric value
- In MIPS instruction set
 - addi: extend immediate value
 - lb, lh: extend loaded byte/halfword
 - beq, bne: extend the displacement
- Replicate the sign bit to the left
 - c.f. unsigned values: extend with 0s
- Examples: 8-bit to 16-bit
 - +2: 0000 0010 => 0000 0000 0000 0010
 - -2: 1111 1110 => 1111 1111 1111 1110

W~~A~~~~T~~

EXTEND SIGN ?

SIGN BIT
↓

0 = POSITIVE
1 = NEGATIVE

+≥ : 8 BITS

0 0 0 0 0 0 1 0

-≥ : 8 BITS

1 1 1 1 1 1 1 0

-≥ : EXTEND TO 16 BITS

SIGN BIT

⇒ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0

✓

WHAT IF WE DON'T EXTEND SIGN?

⇒ 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 X

PROCESSOR THINKS THIS IS POSITIVE

Representing Instructions

- Instructions are encoded in binary
 - Called machine code
- MIPS instructions
 - Encoded as 32-bit instruction words
 - Small number of formats encoding operation code (opcode), register numbers, ...
 - Regularity!
- Register numbers
 - \$t0 – \$t7 are reg's 8 – 15
 - \$t8 – \$t9 are reg's 24 – 25
 - \$s0 – \$s7 are reg's 16 – 23

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	
	31	26 25	21 20	16 15	11 10	6 5
I	opcode	rs	rt		immediate	
	31	26 25	21 20	16 15		
J	opcode				address	
	31	26 25				

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy.

this is how the processor wants you
to encode these instructions
so how do i translate my assembly instruction?

MIPS Reference

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	
	31	26 25	21 20	16 15	11 10	6 5
I	opcode	rs	rt			immediate
	31	26 25	21 20	16 15		
J	opcode					address
	31	26 25				

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy.


this is how the processor wants you
to encode these instructions
so how do I translate my assembly ^{instruction?}

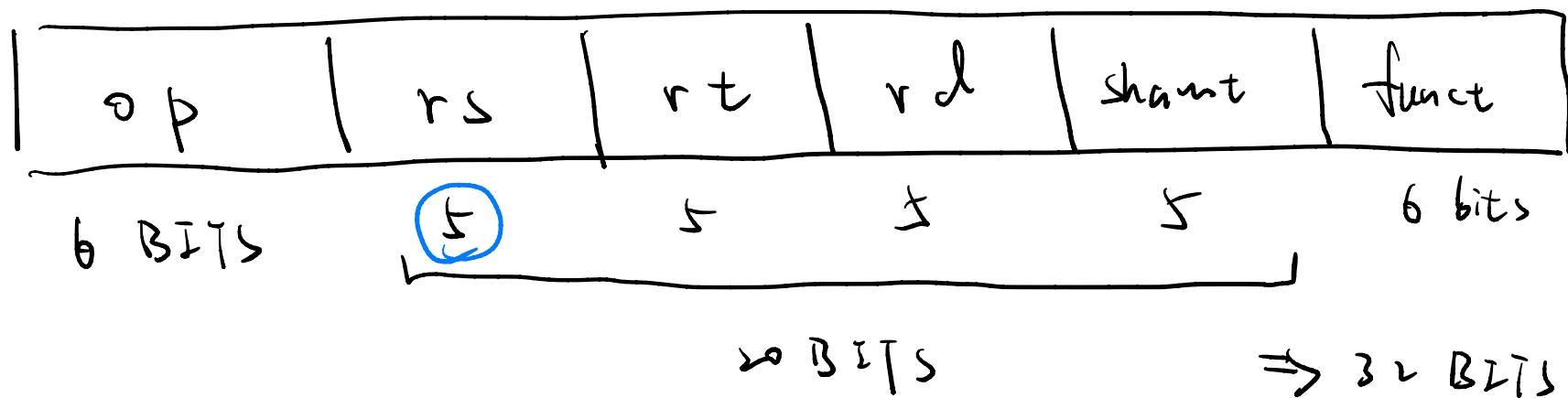
MIPS R-format Instructions



- Instruction fields

- op: operation code (opcode) *6 bits*
 $2^6 \text{ operations} = 64 \text{ operations}$
- 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)

R FORMAT



OP : OPERATION CODE 6 BITS

$$\geq^6 \text{ OPERATIONS} = 64 \text{ OPERATIONS}$$

FUNCT : EXTENDS OP CODE \Rightarrow ALLOWS MORE OPERATION

REGISTERS : rs, rt, rd $\geq^5 = 32$

MIPS HAS 32 REGISTERS

CORE INSTRUCTION SET			OPCODE / FUNCT (Hex)
NAME, MNEMONIC	FOR- MAT	OPERATION (in Verilog)	
Add	add R R[rd] = R[rs] + R[rt]	(1) 0 / 20 _{hex}	

R-format Example

op	rs	rt	rd	shamt	funct
----	----	----	----	-------	-------

6 bits 5 bits 5 bits 5 bits 5 bits 6 bits

add \$t0, \$s1, \$s2 $\$s2 = \$t0 + \$s1$

20_{hex} $20_{hex} = 32_{10}$

special	rs	rt	rd	shifting	add
	\$s1	\$s2	\$t0	0	add

op $0 \quad 17 \quad 18 \quad 8 \quad 0 \quad 32$

000000	10001	10010	01000	00000	100000
--------	-------	-------	-------	-------	--------

$$10001 + 10010 = 100011$$

$$0000001000110010010000000100000_2 = 02324020_{16}$$

\$t0-\$t7	8-15	Temporaries	$\$t=8$	No
\$s0-\$s7	16-23	Saved Temporaries	$\$s0=16; \$s1=17; \$s2=18$	Yes

Hexadecimal

- Base 16
 - Compact representation of bit strings
 - 4 bits per hex digit

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

- Example: eca8 6420
 - 1110 1100 1010 1000 0110 0100 0010 0000
e c a 8 b 4 2 0

like Load word ..

MIPS I-format Instructions

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

- 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

Logical Operations

- Instructions for bitwise manipulation

操作

Operation	C	Java	MIPS
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bitwise AND	&	&	and, andi
Bitwise OR			or, ori
Bitwise NOT	~	~	nor

抽出

插入

- Useful for extracting and inserting groups of bits in a word

Shift Operations

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- shamt: how many positions to shift
- Shift left logical
 - Shift left and fill with 0 bits
 - sll by i bits multiplies by 2^i
- Shift right logical
 - Shift right and fill with 0 bits
 - srl by i bits divides by 2^i (unsigned only)

SHIFT LEFT?

$$3_{10} : \underbrace{0\ 0\ 0\ 0\ 0\ 0\ 1\ 1}_2$$

SHIFT LEFT BY 1 BIT

$$0\ 0\ 0\ 0\ 0\ 1\ 1\ 0_2 = 6_{10} = 3 \times 2 = 3 \times 2^1$$

SHIFT LEFT BY 2 BITS

$$0\ 0\ 0\ 0\ 1\ 1\ 0\ 0_2 = 12_{10} = 3 \times 4 = 3 \times 2^2$$

SHIFT LEFT BY i Bits = MULTIPLY BY 2^i

Shift Right?

$$12_{10} : 0000\ 1100_2$$

Shift Right By 2 Bits

$$\Rightarrow 0000\ 0011_2 = 3_{10} = \frac{12}{2^2}$$

shift right by i Bits = $\frac{\text{DIVIDE}}{\text{By } 2^i}$

AND Operations

- Useful to mask bits in a word
 - Select some bits, clear others to 0

and \$t0, \$t1, \$t2

\$t2	0000 0000 0000 0000 0000 1101 1100 0000
\$t1	0000 0000 0000 0000 0011 1100 0000 0000
\$t0	0000 0000 0000 0000 0000 1100 0000 0000

$$\$t0 = \$t1 \text{ AND } \$t2$$

OR Operations

- Useful to include bits in a word
 - Set some bits to 1, leave others unchanged
or \$t0, \$t1, \$t2

\$t2	0000 0000 0000 0000 0000 1101 1100 0000
\$t1	0000 0000 0000 0000 0011 1100 0000 0000
\$t0	0000 0000 0000 0000 0011 1101 1100 0000

Example Applications

- How to represent text in computer?
 - ASCII: American Standard Code for Information Interchange
- Case conversions: upper to lower, lower to upper
转置
 - Use AND, OR with bit masks

ASCII: Alphabetic codes

	000	001	010	011	100	101	110	111
0000	NULL	DLE		0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EDT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

How to convert between cases?

$20H \rightarrow$ $\begin{array}{r} 2 \\ - - - \\ 0010 \end{array}$ $\begin{array}{r} 0 \\ - - - \\ 000 \end{array}$

Upper to lower: OR with $20H$ = 010 0000

Lower to upper: AND with $5FH$ = 101 1111

$5FH \rightarrow$ $\begin{array}{r} 5 \\ - - - \\ 0101 \end{array}$ $\begin{array}{r} 15 \\ - - - \\ 1111 \end{array}$

Example: A to a: Upper to lower
 with $20H$

$$\begin{array}{r} 100\ 0001 \\ +\ 010\ 0000 \\ \hline 110\ 0001 \end{array}$$

ori \$t0, \$t1, 0x20

z to Z

$$\begin{array}{r} 111\ 1010 \\ * 101\ 1111 \\ \hline 101\ 1010 \end{array}$$

andi \$t0, \$t1, 0x5F

NOT Operations

- Useful to invert bits in a word
 - Change 0 to 1, and 1 to 0
- MIPS has NOR 3-operand instruction
 - $a \text{ NOR } b == \text{NOT} (a \text{ OR } b)$

nor \$t0, \$t1, \$zero

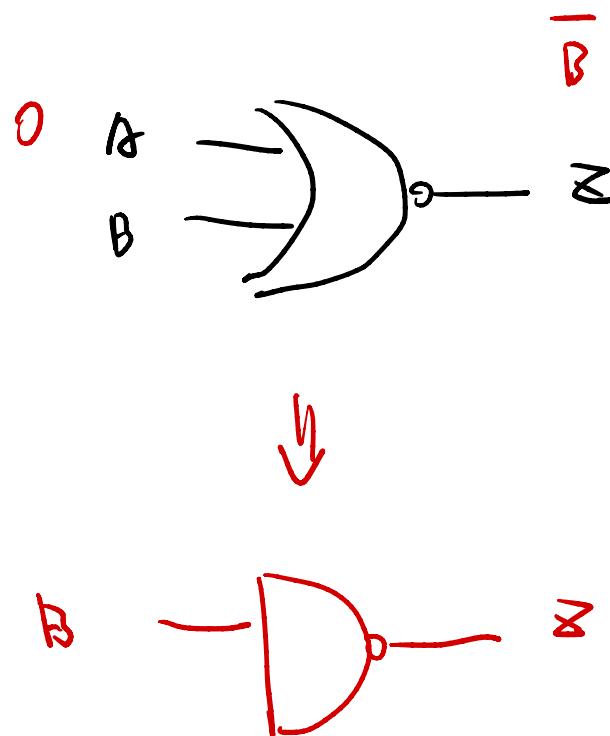
Register 0: always
read as zero

\$t1 0000 0000 0000 0000 0011 1100 0000 0000

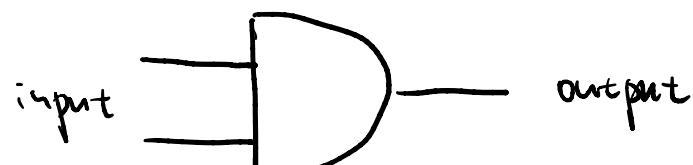
\$t0 1111 1111 1111 1111 1100 0011 1111 1111

NOT using NOR?

A	B	OR	NOR
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

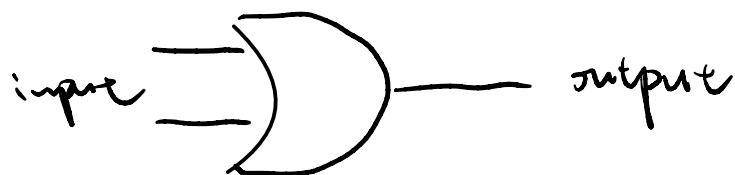


AND



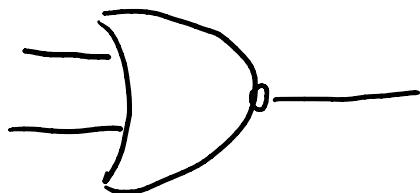
AND	0	1
0	0	0
1	0	1

OR



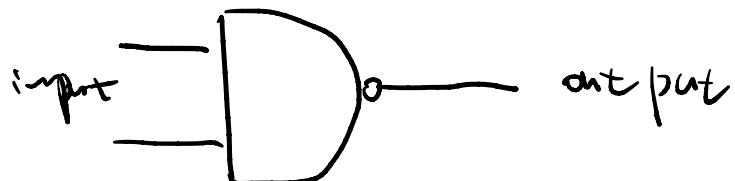
OR	0	1
0	0	1
1	1	1

NOR

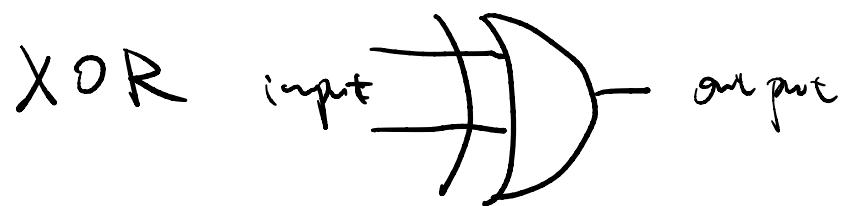


NOR	0	1
0	1	0
1	0	0

NAND



NAND	0	1
0	1	1
1	1	0



OR + AND + NOT

XOR	0	1
0	0	1
1	1	0

Conditional Operations

- Branch to a labeled instruction if a condition is true
 - Otherwise, continue sequentially
- **beq rs, rt, L1**
 - if ($rs == rt$) branch to instruction labeled L1;
- **bne rs, rt, L1**
 - 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 = $PC + offset \times 4$
 - PC already incremented by 4 by this time

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;

Example A.9-2

What MIPS instruction does this bit string represent?

0010 0001 0000 1000 0000 0000 0000 0001

Example A.9-3

Given the following MIPS instruction:

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

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

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?

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?

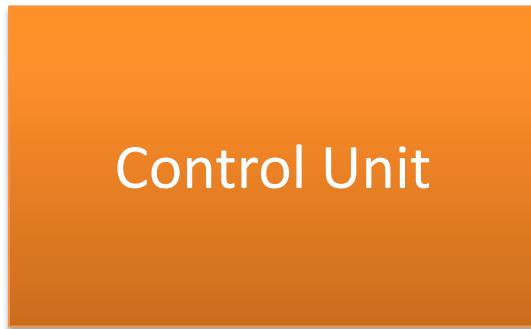
Outline

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

Processor Units

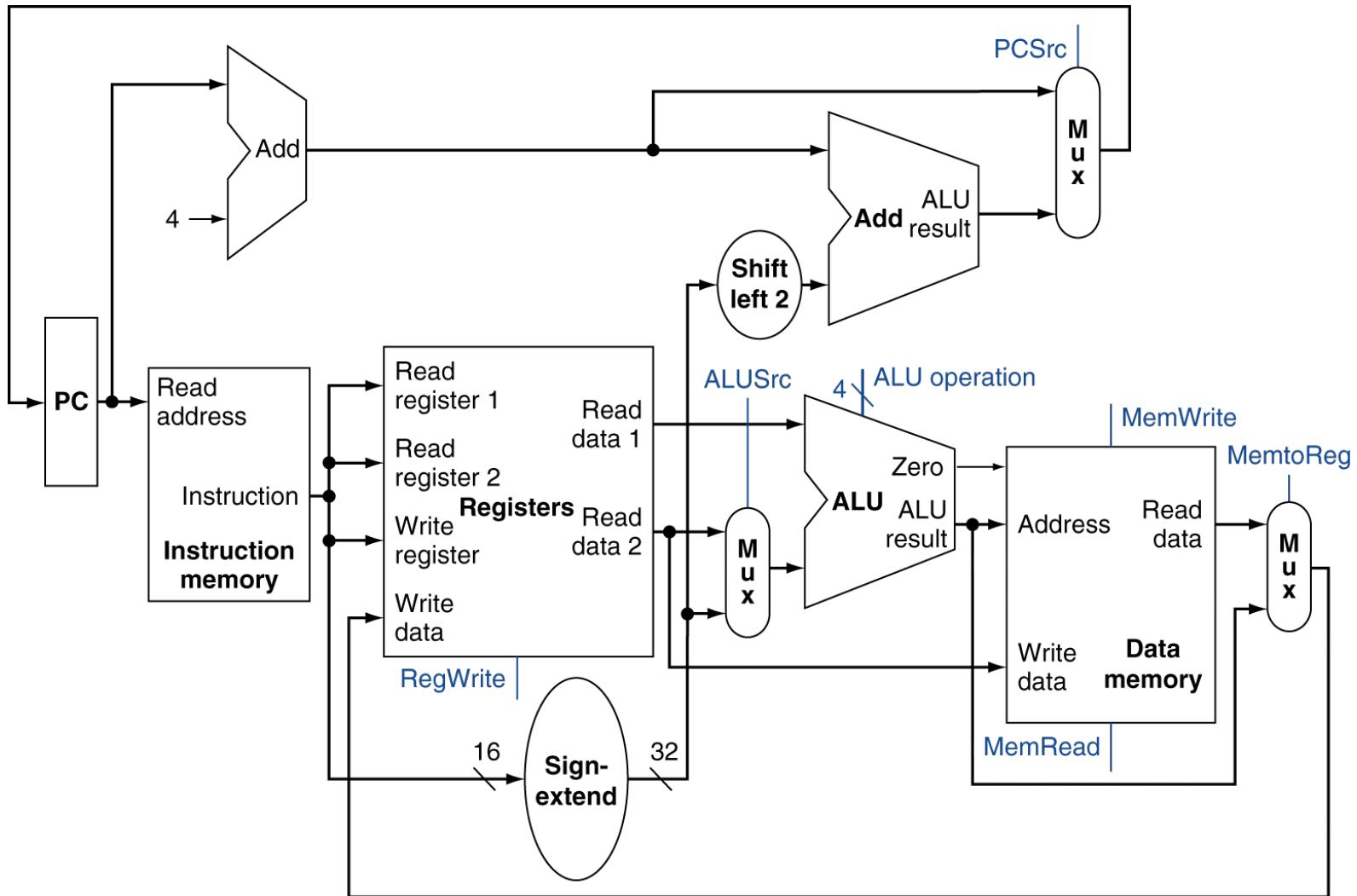


Hardware path (components and interconnections) for instructions and data during program execution



Controls operations of datapath units – flow of data in datapath

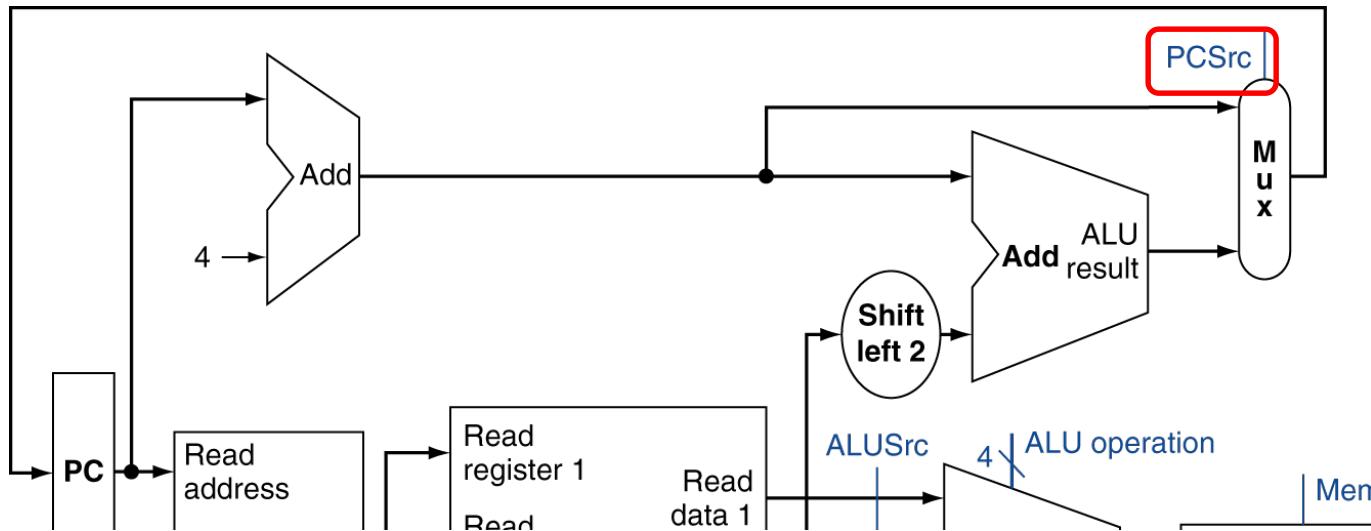
MIPS Datapath



Datapath Control Signals

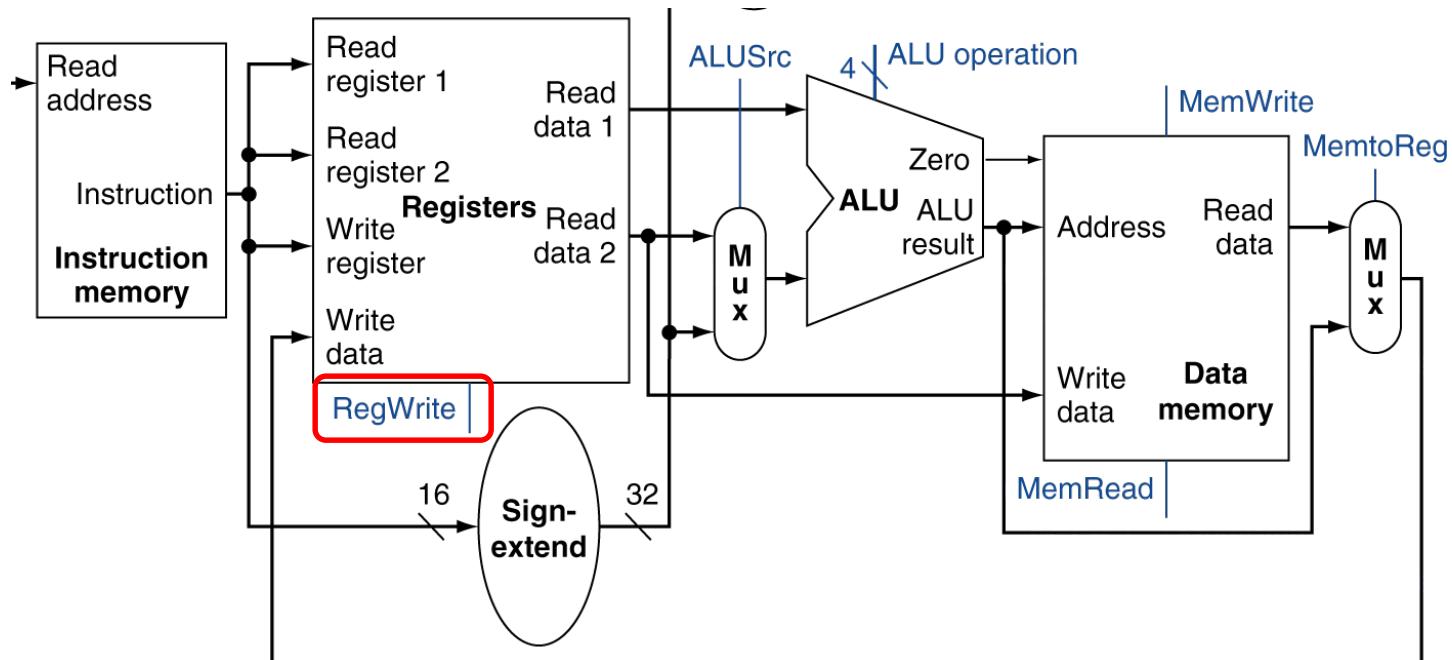
- What are our basic control signals?
 - PCSrc
 - RegWrite
 - ALUSrc
 - MemWrite
 - MemRead
 - MemtoReg
 - ALU operation (4 bits)

PCSrc



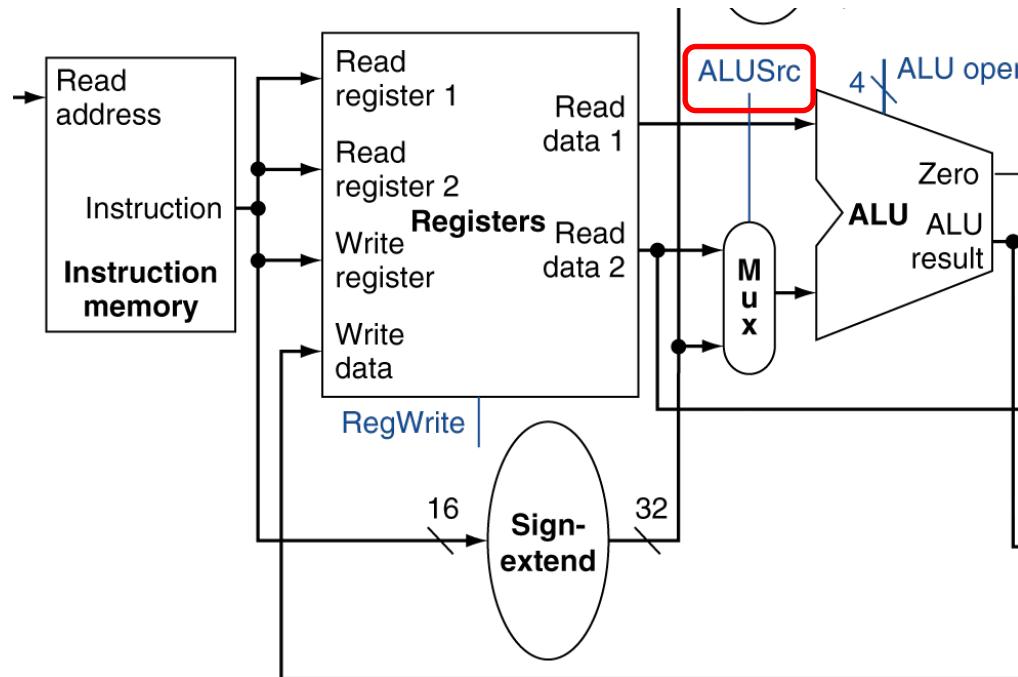
Value	Effect
0	$PC \leq PC + 4$
1	$PC \leq PC + 4 + \text{offset}$

RegWrite



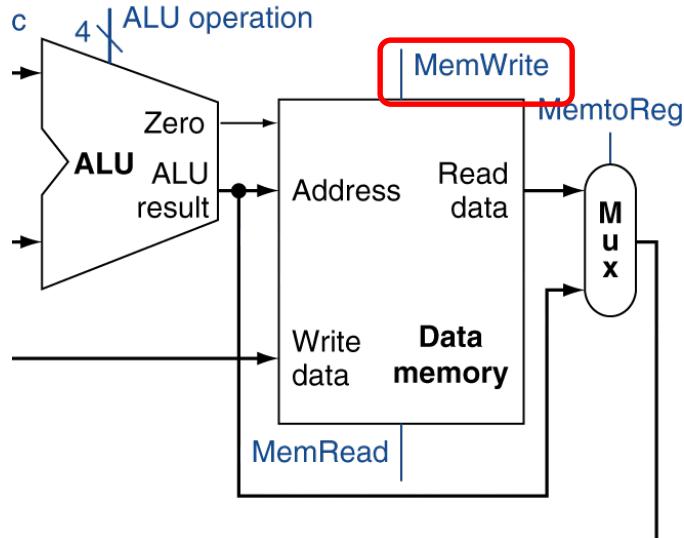
Value	Effect
0	none
1	$R[\text{Write register}] \leq \text{Write data}$

ALUSrc



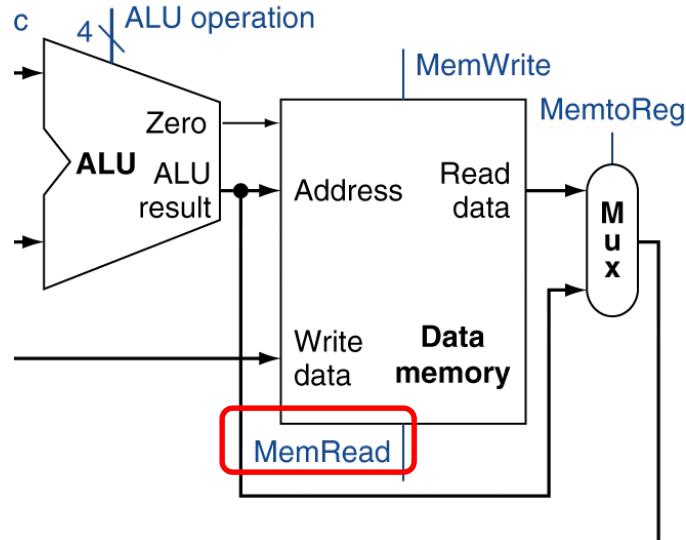
Value	Effect
0	2 nd ALU operand <= Read data 2
1	2 nd ALU operand <= Sign-extended constant

MemWrite



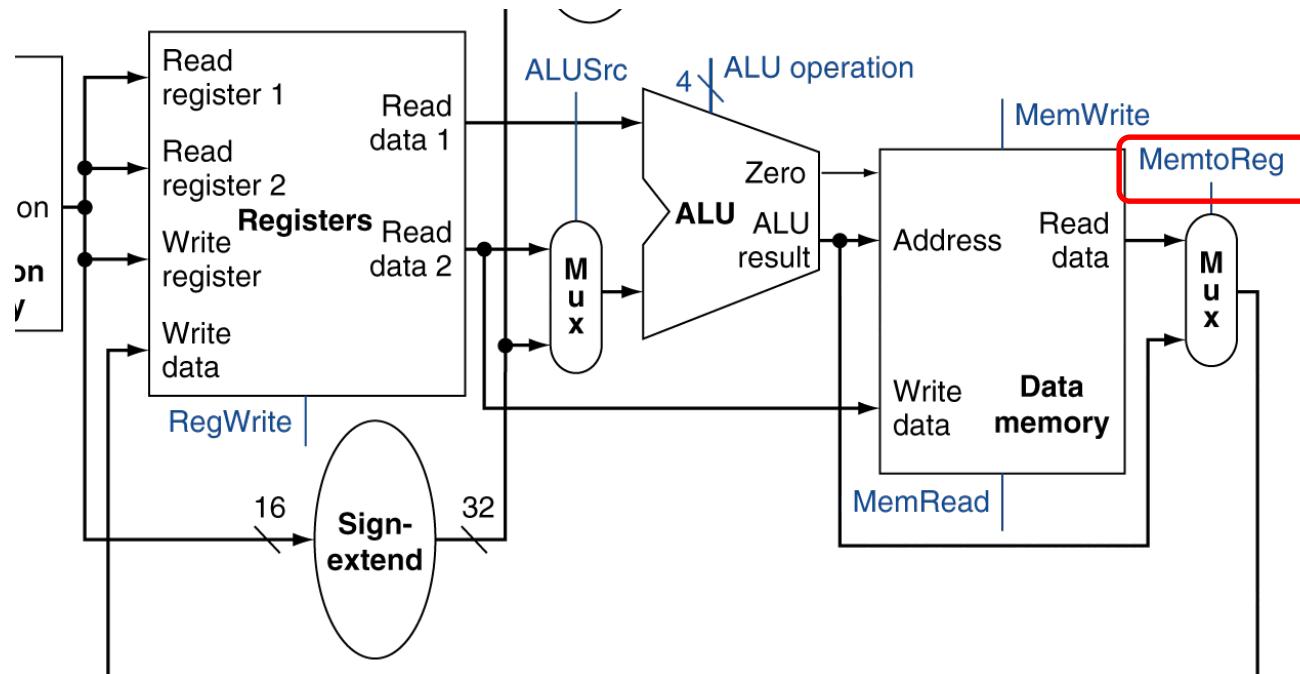
Value	Effect
0	none
1	$\text{Mem}[\text{Address}] \leq \text{Write data}$

MemRead



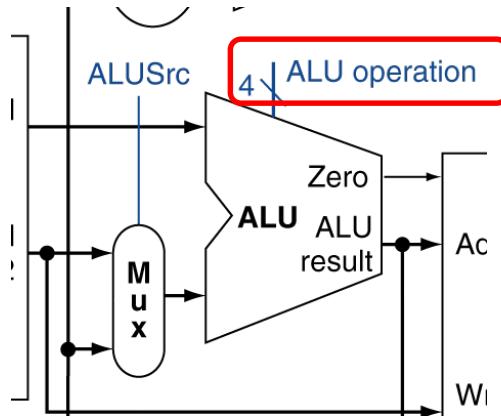
Value	Effect
0	none
1	Read data <= Mem[Address]

MemtoReg



Value	Effect
1	Write data \leq Read data
0	Write data \leq 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

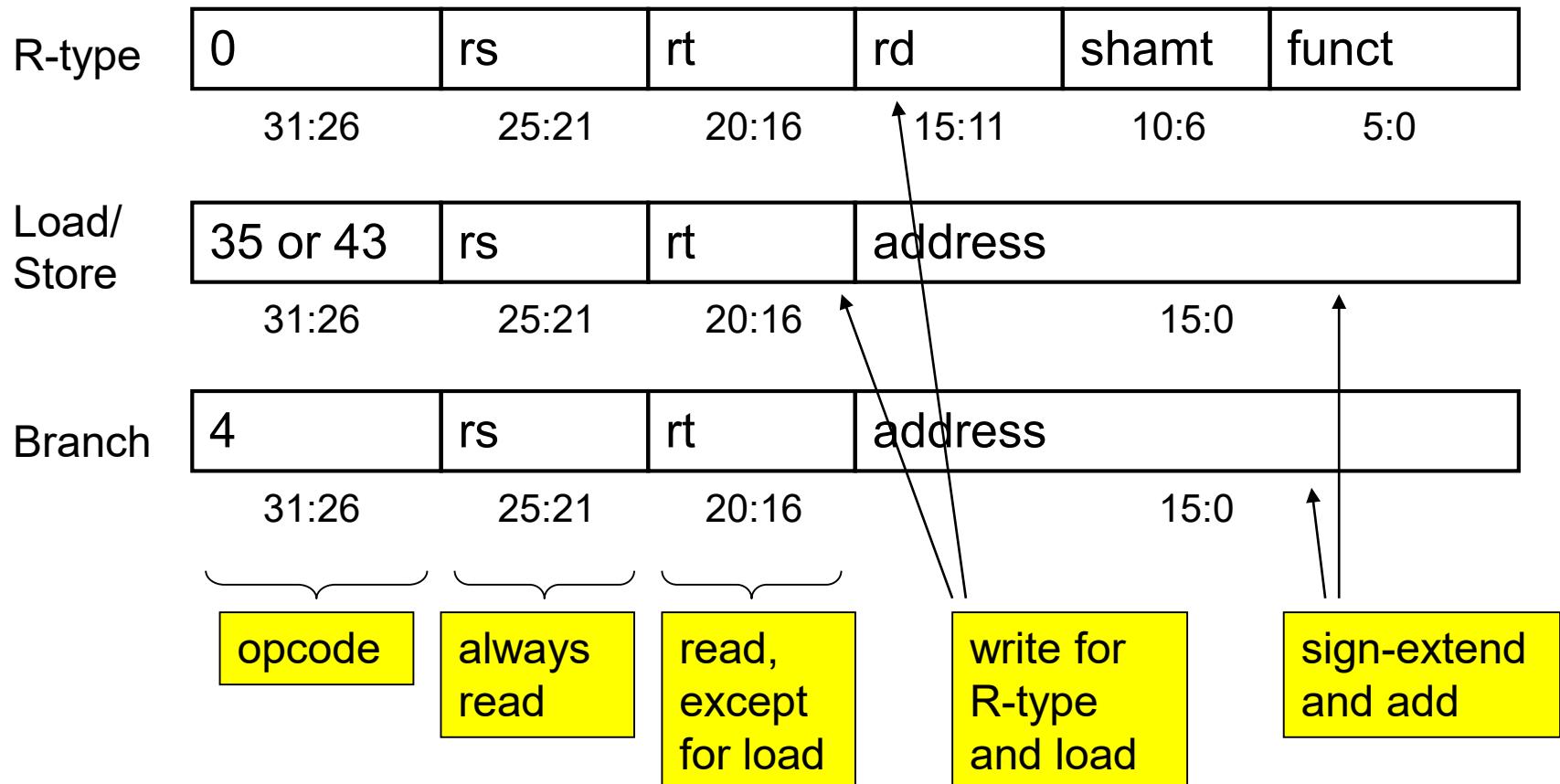
ALU Control

- 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

