

Name	Register number	Usage
\$zero	0	The constant value 0
\$v0-\$v1	2-3	Values for results and expression evaluation
\$a0-\$a3	4-7	Arguments
\$t0-\$t7	8-15	Temporaries
\$s0-\$s7	16-23	Saved
\$t8-\$t9	24-25	More temporaries
\$gp	28	Global pointer
\$sp	29	Stack pointer
\$fp	30	Frame pointer
\$ra	31	Return address

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add \$s1,\$s2,\$s3	$s1 = s2 + s3$	Three register operands
	subtract	sub \$s1,\$s2,\$s3	$s1 = s2 - s3$	Three register operands
	add immediate	addi \$s1,\$s2,20	$s1 = s2 + 20$	Used to add constants
Data transfer	load word	lw \$s1,20(\$s2)	$s1 = \text{Memory}[s2 + 20]$	Word from memory to register
	store word	sw \$s1,20(\$s2)	$\text{Memory}[s2 + 20] = s1$	Word from register to memory
	load half	lh \$s1,20(\$s2)	$s1 = \text{Memory}[s2 + 20]$	Halfword memory to register
	load half unsigned	lhu \$s1,20(\$s2)	$s1 = \text{Memory}[s2 + 20]$	Halfword memory to register
	store half	sh \$s1,20(\$s2)	$\text{Memory}[s2 + 20] = s1$	Halfword register to memory
	load byte	lb \$s1,20(\$s2)	$s1 = \text{Memory}[s2 + 20]$	Byte from memory to register
	load byte unsigned	lbu \$s1,20(\$s2)	$s1 = \text{Memory}[s2 + 20]$	Byte from memory to register
	store byte	sb \$s1,20(\$s2)	$\text{Memory}[s2 + 20] = s1$	Byte from register to memory
	load linked word	ll \$s1,20(\$s2)	$s1 = \text{Memory}[s2 + 20]$	Load word as 1st half of atomic swap
	store condition. word	sc \$s1,20(\$s2)	$\text{Memory}[s2+20]=s1; s1=0 \text{ or } 1$	Store word as 2nd half of atomic swap
	load upper immed.	lui \$s1,20	$s1 = 20 * 2^{16}$	Loads constant in upper 16 bits
Logical	and	and \$s1,\$s2,\$s3	$s1 = s2 \& s3$	Three reg. operands; bit-by-bit AND
	or	or \$s1,\$s2,\$s3	$s1 = s2 s3$	Three reg. operands; bit-by-bit OR
	nor	nor \$s1,\$s2,\$s3	$s1 = \sim (s2 s3)$	Three reg. operands; bit-by-bit NOR
	and immediate	andi \$s1,\$s2,20	$s1 = s2 \& 20$	Bit-by-bit AND reg with constant
	or immediate	ori \$s1,\$s2,20	$s1 = s2 20$	Bit-by-bit OR reg with constant
	shift left logical	sll \$s1,\$s2,10	$s1 = s2 \ll 10$	Shift left by constant
	shift right logical	srl \$s1,\$s2,10	$s1 = s2 \gg 10$	Shift right by constant
Conditional branch	branch on equal	beq \$s1,\$s2,25	if ($s1 == s2$) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne \$s1,\$s2,25	if ($s1 \neq s2$) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt \$s1,\$s2,\$s3	if ($s2 < s3$) $s1 = 1$; else $s1 = 0$	Compare less than; for beq, bne
	set on less than unsigned	sltu \$s1,\$s2,\$s3	if ($s2 < s3$) $s1 = 1$; else $s1 = 0$	Compare less than unsigned
	set less than immediate	slti \$s1,\$s2,20	if ($s2 < 20$) $s1 = 1$; else $s1 = 0$	Compare less than constant
	set less than immediate unsigned	sltiu \$s1,\$s2,20	if ($s2 < 20$) $s1 = 1$; else $s1 = 0$	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr \$ra	go to \$ra	For switch, procedure return
	jump and link	jal 2500	$sra = PC + 4$; go to 10000	For procedure call

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

Instruction Set Architecture: MIPS

Instruction Formats, Logical Operations and Branching

Dr. Ahmed H. Zahran

WGB 182

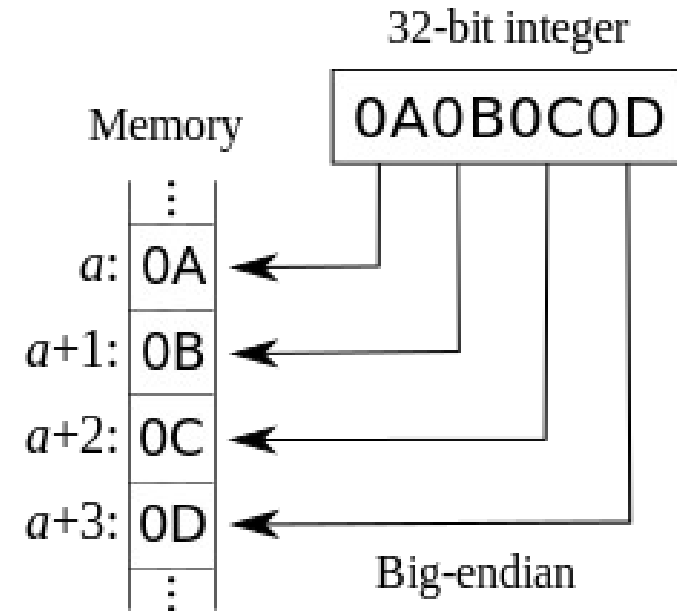
a.zahran@cs.ucc.ie

Objectives

- Explore Key MIPS instructions and their formats
- Review Integer Number Representations
- Explore branching instructions for flow control

Memory Operands

- Main memory used for composite data
 - Arrays, structures, dynamic data
- **Data transfer instructions**
 - 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
 - Word address must be a multiple of 4
- MIPS is Big Endian
 - 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`

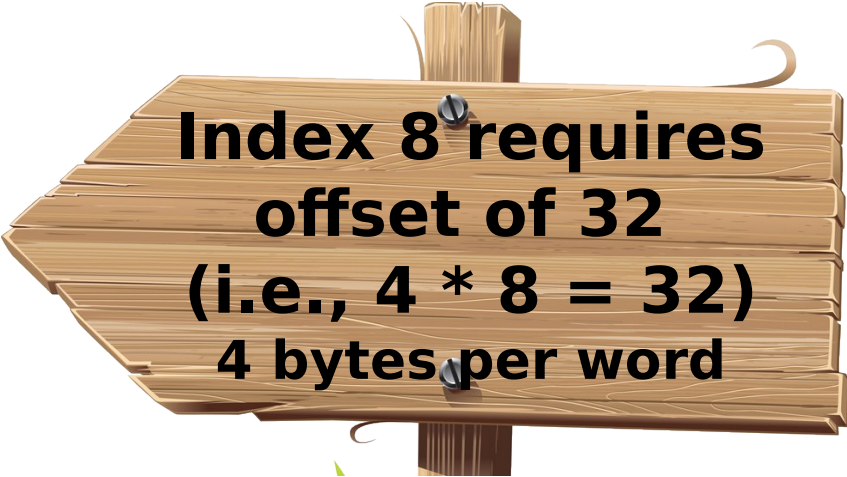
- Compiled MIPS code:

lw `$t0, 32($s3)` # load word

add `$s1, $s2, $t0`

offset

base register



**Index 8 requires
offset of 32
(i.e., $4 * 8 = 32$)
4 bytes per word**

Memory Operand Example 2

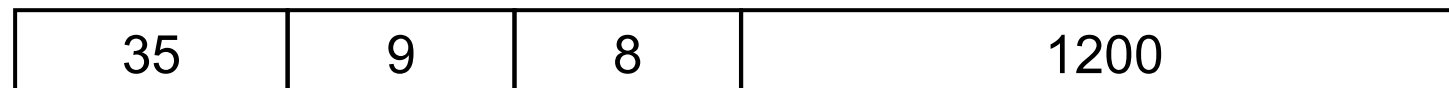
- C code:
A[12] = h + A[8];
 - h in \$s2, base address of A in \$s3
- Compiled MIPS code:
 - **lw** \$t0, 32 (\$s3) # load word
 - **add** \$t0, \$s2, \$t0 # perform addition
 - **sw** \$t0, (\$s3) # store word Offset = ???

How would this change if I am processing a byte array?

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
- Example: ***lw*** \$t0, 1200(\$t1)



Register vs. Memory

- Registers are faster to access than memory
- Operating on memory data requires loads and stores
 - More instructions to be executed
- Compiler must use registers for variables as much as possible
 - Only spill to memory for less frequently used variables
 - Register optimization is important!

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

- ***Design Principle 3: Make the common case fast***

- Small constants are common
 - Immediate operand avoids a load instruction

The Zero Register

- MIPS register 0 (\$zero) is the constant 0
 - Hardwired (Cannot be overwritten)
- Useful for common operations

add \$t2, \$s1, \$zero

Binary Representation of Numbers

Positive numbers

Negative numbers

Unsigned Binary 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: 0 to $+2^n - 1$
- Using 32 bits
 - 0 to +4,294,967,295
- Example

$$\begin{aligned} & \bullet 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2 \\ & = 0 + \dots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ & = 0 + \dots + 8 + 0 + 2 + 1 = 11_{10} \end{aligned}$$

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:

- Negative -2^{n-1} to -1
- Positive 0 to $+2^{n-1} - 1$

- Using 32 bits

- $-2,147,483,648$ to $+2,147,483,647$

- Example

$$\begin{aligned} & \bullet 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2 \\ & = -1 \times 2^{31} + 1 \times 2^{30} + \dots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\ & = -2,147,483,648 + 2,147,483,644 = -4_{10} \end{aligned}$$

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 $1 \rightarrow 0, 0 \rightarrow 1$

$$x + \bar{x} = 1111 \dots 111_2 = -1$$

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

- Example: negate +2
 - $+2 = 0000 \ 0000 \dots 0010_2$
 - $-2 = 1111 \ 1111 \dots 1101_2 + 1$
 $= 1111 \ 1111 \dots 1110_2$

Signed Extension

- Representing a number using more bits
 - **unsigned values:** extend with 0s
 - **Signed numbers:** replicate the sign bit to the left
- Examples: 8-bit to 16-bit
 - +2: 0000 0010 => 0000 0000 0000 0010
 - -2: 1111 1110 => 1111 1111 1111 1110
- In MIPS instruction set
 - ***addi***: extend immediate value
 - ***lb*, *lh***: extend loaded byte/halfword

Logical Operations

- Instructions for *bitwise* manipulation

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

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

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

OR Operations

- Useful to **set** some bits to 1 in a word and 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

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

Reading

- Sections 2.5, 2.6, 2.7