

## 3. Instruction Set Architecture – The MIPS architecture

---

EECS 370 – Introduction to Computer Organization – Winter 2007

Prof. Valeria Bertacco & Prof. Scott Mahlke

EECS Department  
University of Michigan in Ann Arbor, USA

© V. Bertacco & S. Mahlke, 2007  
The material in this presentation cannot be  
copied in any form without our written permission

### Instruction Set Architecture (ISA) Design Lectures

---

- ❑ Lecture 2: Storage types and addressing modes
- ❑ **Lecture 3: MIPS architecture**
- ❑ Lecture 4: Calling functions / passing arguments
- ❑ Lecture 5: Translation software; libraries, VMs

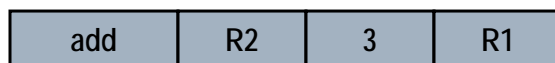
## Recap (Storage)

- ❑ Registers
  - Small array of storage locations in processor
  - Fast access
  - Direct addressing only
  - digression: the PC register
- ❑ Memory
  - Large array of storage locations
  - Slow access
  - Many addressing modes (direct, indirect, double indirect, reg indirect, base+displmt)

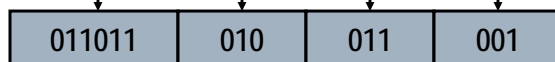
## Recap (Assembly/Machine Code)

- ❑ Computers store program instructions the same way they store data.
- ❑ Each instruction is encoded as a number
  - Opcode field: what instruction to perform.
  - Operand fields: what data to perform it on.

Assembly code



Machine code



## LC-2Kx Processor

- ❑ 32-bit processor
  - Instructions are 32 bits
  - Integer registers are 32 bits
- ❑ 8 registers
- ❑ supports 65536 words of memory (addressable space)
  
- ❑ 8 instructions
  - add, nand, lw, sw, beq, jalr, halt, noop

## Assembly Format

- Format:

**label** <white> **instr** <white> **field0** <white> **field1** <white> **field2** <white> **comments**

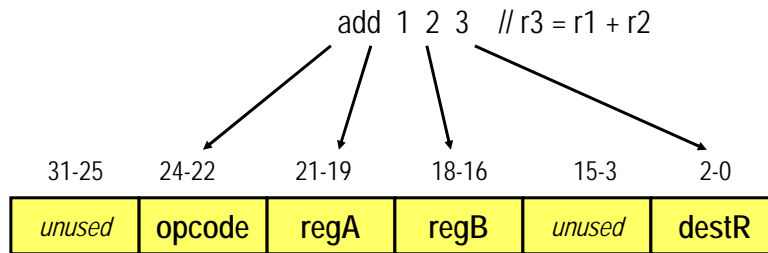
- Labels: Max of 6 characters, starts with a letter {A..Z,a..z} followed by letters or numbers.

- Instruction: add, nand, lw, sw, beq, jalr, noop, halt and “.fill”

0	lw	0	1	five	load reg1 with 5 (uses symbolic address)	
1	lw	1	2	3	load reg2 with -1 (uses numeric address)	
2	start	add	1	2	1	decrement reg1
3	beq	0	1	2	goto end of program when reg1 equals 0	
4	beq	0	0	start	go back to the beginning of the loop	
5	noop					
6	done	halt			end of program	
7	five	.fill	5			
8	neg1	.fill	-1			
9	stAddr	.fill	start		will contain the address of start (2)	

## Instruction Encoding

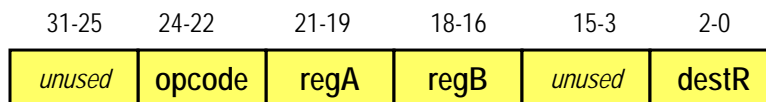
- Instruction set architecture defines the mapping of assembly instructions to machine code



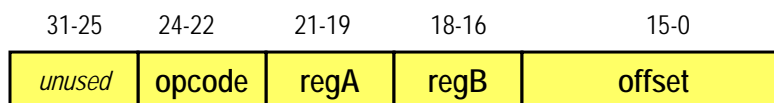
## Instruction Formats

- Positional organization of bits (Implies nothing about bit values!!!)

- R type instructions (add, nand)



- I type instructions (lw, sw, beq)

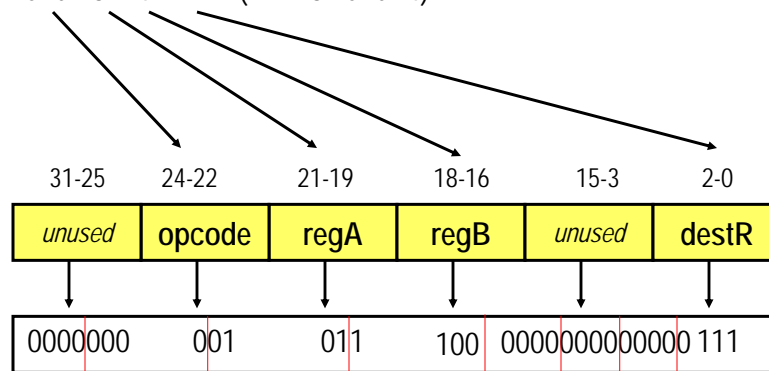


## Bit Encodings

- ❑ Opcode encodings
  - add (000), nand (001), lw (010), sw (011), beq (100), jalr (101), halt (110), noop (111)
- ❑ Register values
  - Just encode the register number (r2 = 010)
- ❑ Immediate values
  - Just encode the values (Remember to give all the available bits a value!!)

## Example Encoding - nand

- ❑ nand 3 4 7 (r7 = r3 nand r4)

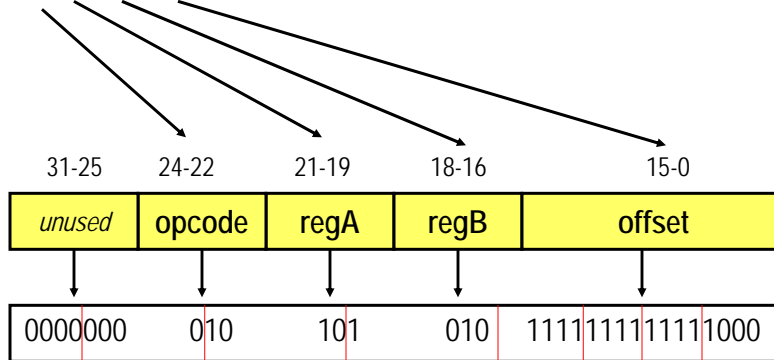


Convert to Hex → 0x005C0007

Convert to Dec → 6029319

### Example Encoding - lw

lw 5 2 -8 (r2 = M[r5 + -8])



Convert to Hex → 0x00AFFF8

Convert to Dec → 11206648

### Mini-review: Representing Negative Numbers

2's complement representation

Positive numbers:

- 1 0000 0001
- 2 0000 0010
- 3 0000 0011
- 4 0000 0100
- 5 0000 0101
- 6 0000 0110
- 7 0000 0111
- 8 0000 1000
- 9 0000 1001

Negative numbers:

- 1 1111 1111
- 2 1111 1110
- 3 1111 1101
- 4 1111 1100
- 5 1111 1011
- 6 1111 1010
- 7 1111 1001
- 8 1111 1000
- 9 1111 0111

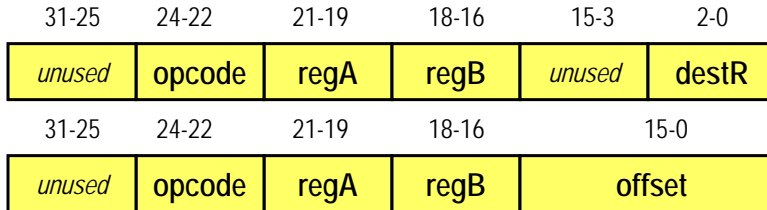
Neg r5, #4

4 = 0000 0100

complement 1111 1011  
+ 1  
-----  
1111 1100 = -4

### Class Problem 1

- ❑ Compute the encoding in Hex for:
  - add 3 7 3 (r3 = r3 + r7) (add = 000)
  - sw 1 5 67 (M[r1+67] = r5) (sw = 011)



### Mini-review: What is a *shift*?

- ❑ C/C++
    - a = b >> 2;
    - c = d << 4;
- b = 01101110**

**b = 00110111**
- ❑ MIPS
    - slr \$3, \$2, 2
    - sll \$5, \$4, 4
  - ❑ LC2K
    - ...

## MIPS Instruction Set

- Three main types of instructions:
  1. Arithmetic
    - Add, subtract, multiply, divide
    - Logical: and, or, shift, rotate, etc.
    - Compare : equal, lt, le, ne, etc.
  2. Memory access
    - Load, store
  3. Sequencing / control flow
    - Jump, branch, function call, return

## MIPS Arithmetic Instructions – type 1

- Format: three register operand fields
  - e.g., add \$3, \$4, \$7
- C-code example:  $f = (g + h) - (i + j)$

\$3→g  
\$4→h  
\$5→i  
\$6→j

```
add t0, g, h
add t1, i, j
sub f, t0, t1
```



```
add $1, $3, $4
add $2, $5, $6
sub $7, $1, $2
```

## MIPS Arithmetic Instructions – type 2

- Format: two register operand fields and an immediate constant (16 bit)
  - e.g., `addi $3, $4, 10`
- C-code example: `f = g × 10`

```
multi $7, $5, 10
```

## What About immediates Larger Than 16 Bits??

- Use 2 instructions
  - Load upper bits (load upper immediate, `lui`)
  - Add in lower bits (add immediate, `addi`)
- C-code example: `f = 0x10002`

```
lui    $7, 1  
addi   $7, $7, 2
```

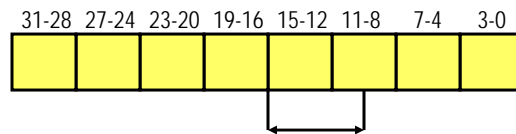
0x10000
0x10002

## MIPS Arithmetic Instructions - recap

- abs rd, rs // absolute value
- add rd, rs, rt // add
- addi rd, rs, imm // add immediate
- and rd, rs, rt // logical AND of bits
- div rd, rs, rt // divide
- mult rd, rs, rt // multiply
- **neg rd, rs // negate (pseudo-instruction)**
- nor rd, rs, rt // logical NOR of bits
- sll rd, rs, imm // shift left logical
- sub rd, rs, rt // sub rt from rs
- subi rd, rs, imm // sub immediate from rs
- srl rd, rs, imm // shift right logical
- **li rd, imm // load immediate (pseudo-instruction)**

## Class Problem 2

- Show the C and MIPS assembly for extracting the value in bits 15:10 from a 32-bit integer variable



Want these bits

Remember each hex digit is 4 bits

## MIPS Memory Instructions

- ❑ Supports base + displacement mode only
  - Base is a register
  - Offset is a 16-bit immediate
  
- ❑ Format: 2 registers (dest, base) and 16-bit immediate (offset)
  - Example:  
lw \$3, 1000(\$4) // load word  
Retrieves 32-bit value from memory location ( $\$4+1000$ ) and puts the result into \$3

## Load Instruction Sizes

How much data is retrieved from memory at the given address?

- ❑ lw \$3, 1000(\$4)
  - retrieve a word (32 bits) from address ( $\$4+1000$ )
  
- ❑ lh \$3, 1000(\$4)
  - retrieve a halfword (16 bits) from address ( $\$4+1000$ )
  
- ❑ lb \$3, 1000(\$4)
  - retrieve a byte (8 bits) from address ( $\$4+1000$ )

## Sign/Zero Extension

- ❑ Registers in MIPS are 32 bits!
- ❑ So what happens when you load 8 or 16 bits?

- Sign extend if the load is signed



- Zero extend if the load is unsigned

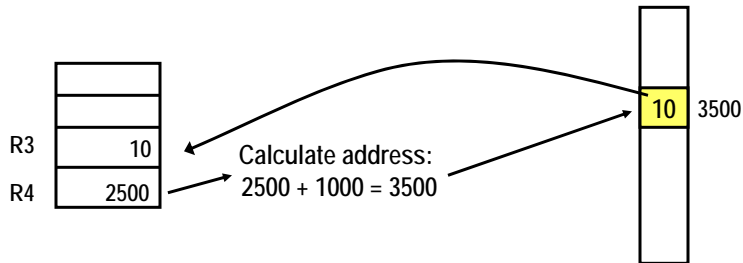


## MIPS Memory Instructions - recap

- ❑ Load instructions
  - lb \load byte signed (load 8 bits, sign extend)
  - lbu \load byte unsigned (load 8 bits, zero extend)
  - lh \load halfword (load 16 bits, sign extend)
  - lhu \load halfword unsigned (load 16 bits, zero extend)
  - lw \load word (load 32 bits, no extension)
  
- ❑ Store instructions (No sign/zero extension for stores)
  - sb \$3, 1000(\$4) \store 8 LSBs of \$3 to M[\$4+1000]
  - sh \$3, 1000(\$4) \store 16 LSBs of \$3 to M[\$4+1000]
  - sw \$3, 1000(\$4) \store all 32 bits of \$3 to M[\$4+1000]

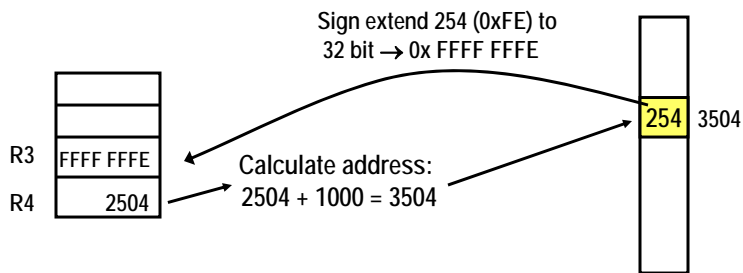
### Load Instruction in Action

- ❑ `lb $3, 1000($4)` // load byte  
Retrieves 8-bit value from memory location ( $\$4+1000$ ) and puts the result into  $\$3$  (sign extended)



### Load Instruction in Action – other example

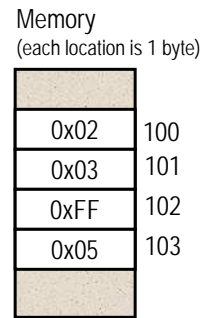
- ❑ `lb $3, 1000($4)` // load byte  
Retrieves 8-bit value from memory location ( $\$4+1000$ ) and puts the result into  $\$3$  (sign extended)



## Example Code Sequence

What is the final state of memory once you execute the following instruction sequence?

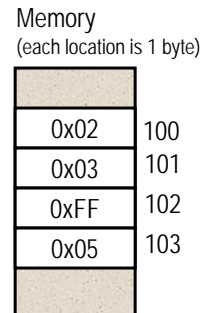
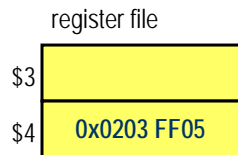
lw	\$4, 100(\$0)
lb	\$3, 102(\$0)
sw	\$3, 100(\$0)
sb	\$4, 102(\$0)



## Example Code Sequence – insn 1

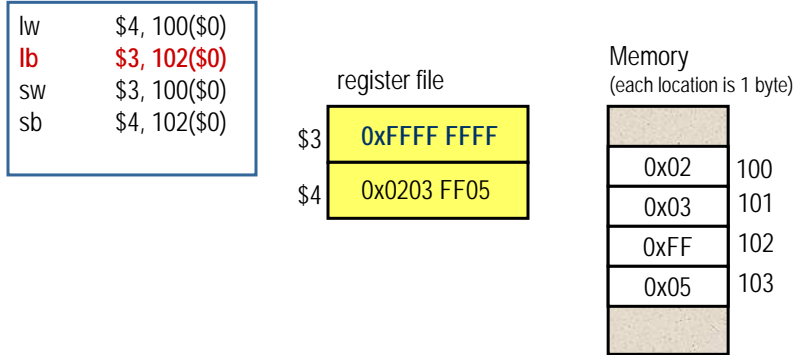
What is the final state of memory once you execute the following instruction sequence?

<b>lw</b>	<b>\$4, 100(\$0)</b>
lb	\$3, 102(\$0)
sw	\$3, 100(\$0)
sb	\$4, 102(\$0)



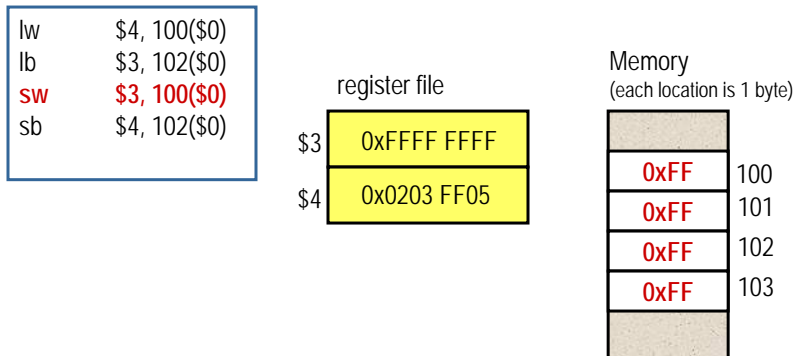
### Example Code Sequence – insn 2

What is the final state of memory once you execute the following instruction sequence?



### Example Code Sequence – insn 3

What is the final state of memory once you execute the following instruction sequence?



### Example Code Sequence – insn 4

What is the final state of memory once you execute the following instruction sequence?

lw	\$4, 100(\$0)
lb	\$3, 102(\$0)
sw	\$3, 100(\$0)
<b>sb</b>	<b>\$4, 102(\$0)</b>

register file

\$3	0xFFFF FFFF
\$4	0x0203 FF05

Memory  
(each location is 1 byte)

	0xFF	100
	0xFF	101
	<b>0x05</b>	102
	0xFF	103

### Class Problem 3

What is the final state of memory once you execute the following instruction sequence?

lhu	\$3, 100(\$0)
lb	\$4, 102(\$0)
sw	\$3, 100(\$0)
sh	\$4, 102(\$0)

register file

\$3	
\$4	

Memory  
(each location is 1 byte)

	0xFF	100
	0xFF	101
	0x77	102
	0xFF	103

## Converting C to Assembly

---

- ❑ Memory layout → memory addresses
- ❑ Branches
- ❑ Procedure calls
- ❑ Expression trees
- ❑ Register allocation
- ❑ This topic (along with procedure calls) is the focus of the next lecture, but let's do a brief warmup

## Converting C to assembly – example 1

---

Write MIPS Assembly Code for the following C Expression:

C: `a = b + names[ i ];`

Assume that **a** is in \$1, **b** is in \$2, **i** is in \$3, and the array **names** starts at address 1000 and holds 32-bit integers.

```
mult    $5, $3, 4        // calculate array offset
lw      $4, 1000($5)     // load names[i]
add     $1, $2, $4       // calculate b + names[i]
```

## Converting C to assembly – example 2

---

Write MIPS Assembly Code for the following C Expression:

```
class { int a; char b, c; } y;
```

```
y.a = y.b + y.c;
```

Assume that a pointer to **y** is in \$1.

```
lb    $2, 4($1) // load y.b
lb    $3, 5($1) // load y.c
add   $4, $2, $3 // calculate y.b+y.c
sw    $4, 0($1) // store y.a
```