COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface

EECS 2021

Computer Organization
Fall 2015

The slides are based on the publisher slides and contribution from Profs Amir Asif and Peter Lian
The slides will be modified, annotated, explained on the board, and sometimes corrected in the class

Based on slides by the author and prof. Mary Jane Irwin of PSU.

Byte/Halfword Operations

- Could use bitwise operations
- MIPS byte/halfword load/store
  - String processing is a common case

```
lb rt, offset(rs)      lh rt, offset(rs)
```

  - Sign extend to 32 bits in rt

```
lbu rt, offset(rs)     lhu rt, offset(rs)
```

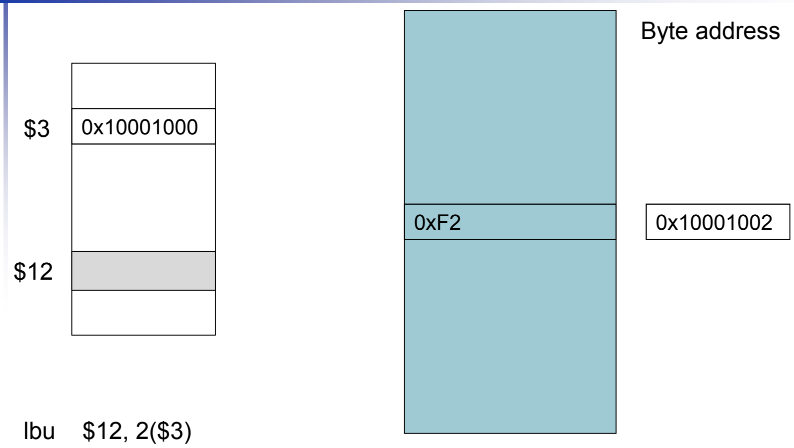
  - Zero extend to 32 bits in rt

```
sb rt, offset(rs)      sh rt, offset(rs)
```

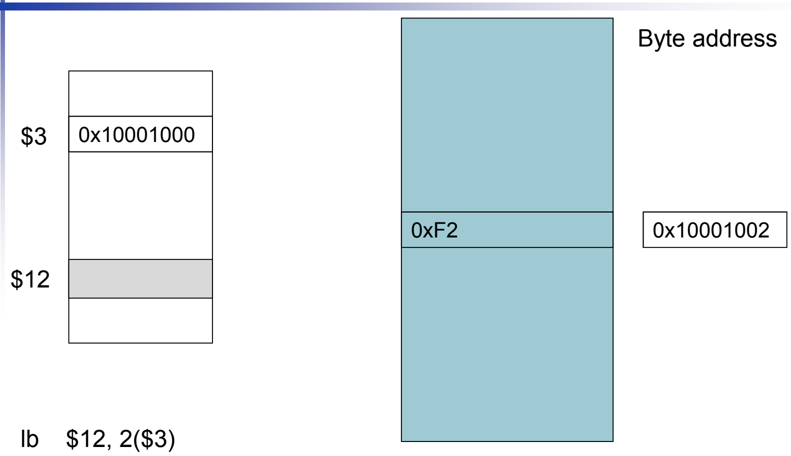
  - Store just rightmost byte/halfword

Chapter 2 — Instructions: Language of the Computer — 2

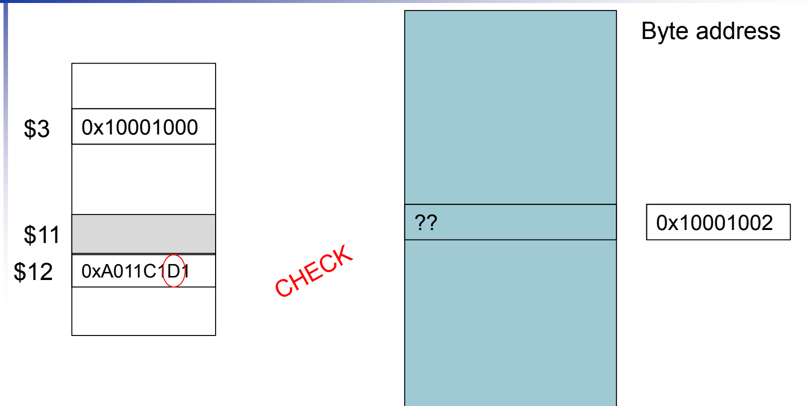
Example



Example



Example



sb $12, 2($3)

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:
  - Index 8 requires offset of 32
    - 4 bytes per word

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

offset　　base register

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
```

Chapter 2 — Instructions: Language of the Computer — 7

Registers 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!

Chapter 2 — Instructions: Language of the Computer — 8

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 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`

Unsigned Binary Integers

§2.4 Signed and Unsigned Numbers

- Given an n-bit number

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: 0 to $+2^n - 1$
- Example
  - $0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1011_2$
    $= 0 + \ldots + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$
    $= 0 + \ldots + 8 + 0 + 2 + 1 = 11_{10}$
- Using 32 bits
  - 0 to +4,294,967,295

Chapter 2 — Instructions: Language of the Computer — 11

2s-Complement Signed Integers

- Given an n-bit number

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

- Range: -2^{n-1} to $+2^{n-1} - 1$
- Example
  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$
    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$
    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$
- Using 32 bits
  - –2,147,483,648 to +2,147,483,647

Chapter 2 — Instructions: Language of the Computer — 12

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

Chapter 2 — Instructions: Language of the Computer — 13

Signed Negation

- Complement and add 1
  - Complement means 1 → 0, 0 → 1

$$x + \overline{x} = 1111\ldots111_2 = -1$$

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

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

Chapter 2 — Instructions: Language of the Computer — 14

2's Complement

	2'sc binary	decimal
-2^3 =	1000	-8
$-(2^3 - 1)$ =	1001	-7
	1010	-6
	1011	-5
	1100	-4
	1101	-3
	1110	-2
	1111	-1
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
$2^3 - 1$ =	0111	7

complement all the bits

0101
and add a 1

0110 (6)

Chapter 2 — Instructions: Language of the Computer — 15

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

Chapter 2 — Instructions: Language of the Computer — 16