

#### COMPUTER ORGANIZATION AND DE

The Hardware/Software Interface



# **Chapter 2**

# Instructions: Language of the Computer

```
(1) R-Type instructions: 3 registers of code add sub muft of volcode all start of all start (2) I - Type instructions: 2 registers produce with 6-bit opcode lw sw addi.

(3) Branch instruction: 2 registers beq bne

Ref: page 64 tx+bk.

- 80 - 86 -
```

#### Instruction Set

- The repertoire of instructions of a computer
- Different computers have different instruction sets
  - But with many aspects in common
- Early computers had very simple instruction sets
  - Simplified implementation
- Many modern computers also have simple instruction sets



#### The MIPS Instruction Set

- Used as the example throughout the book
- Stanford MIPS commercialized by MIPS Technologies (<u>www.mips.com</u>)
- Large share of embedded core market
  - Applications in consumer electronics, network/storage equipment, cameras, printers, ...
- Typical of many modern ISAs
  - See MIPS Reference Data tear-out card, and Appendixes B and E

# **Arithmetic Operations**

- Add and subtract, three operands
  - Two sources and one destination

```
add a, b, c # a gets b + c
```

- All arithmetic operations have this form
- Design Principle 1: Simplicity favours regularity
  - Regularity makes implementation simpler
  - Simplicity enables higher performance at lower cost



## **Arithmetic Example**

C code:

```
f = (g + h) - (i + j);
```

Compiled MIPS code:

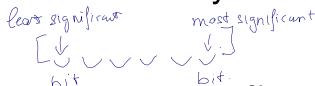
```
add t0, g, h # temp t0 = g + h add t1, i, j # temp t1 = i + j sub f, t0, t1 # f = t0 - t1
```

# Register Operands

MiPs has 32×32 bit registers Each 32 bit is a word! \$t0 -> \$t9: temporary value \$50 -> \$s7: saved value (input, on 1 put)

MIPS instruction uses 4 bytes for each instruction.

- Arithmetic instructions use register operands
- MIPS has a 32 × 32-bit register file in MiPs, unit is
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a "word"
- Assembler names fixed names for temp registers: only 10 registers
- G₁ \$t0, \$t1, ..., \$t9 for temporary values
- Design Principle 2: Smaller is faster
  - c.f. main memory: millions of locations



This is why MiPS does not increase their size more than 32 x 32.

are reserved.

Principle 1:

Simplicity favors

regularity

## Register Operand Example

C code:
 f = (g + h) - (i + j);
 f, ..., j in \$\$0, ..., \$\$4

Compiled MIPS code:

```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $s0, $t0, $t1
```

# **Memory Operands**

Load: In register dest, RAM\_source

# copy word (4 bytes) from source RAM to dest register.

store: sw register\_ Srg. RAM\_dest

# store word in source register into RAM destination.

Main memory used for composite data

Arrays, structures, dynamic data

To apply arithmetic operations



Store result from register to memory

Memory is byte addressed

Each address identifies an 8-bit byte

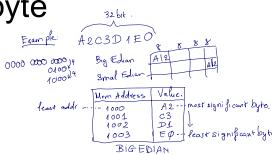
Words are aligned in memory

Address must be a multiple of 4

MIPS is 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**

- for array, the data must have been stored in memory.

  So we have to look their value to a temp

  register by process, Always times 4 to the.

  g = h + A[8]; index. when converting to MiPS code.
  - 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
```



## **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

```
Tw $t0, 32($s3) # Toad word
add $t0, $s2, $t0
sw $t0.48($s3) # store word
```

## Registers vs. Memory

- Registers are faster to access than memory (RAM).
- 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
- - Small constants are common
  - Immediate operand avoids a load instruction



#### **The Constant Zero**

- MIPS register 0 (\$zero) is the constant 0
  - Cannot be overwritten So there are 19 registers
- Useful for common operations
  - E.g., move between registers
     add \$t2, \$s1, \$zero

## **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<sup>n</sup> – 1

unsigned: 2<sup>n</sup>-1 biggest no. Signed: 2<sup>n</sup>-1 biggest no.

- Example
  - 0000 0000 0000 0000 0000 0000 0000 1011<sub>2</sub> = 0 + ... +  $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$ = 0 + ... + 8 + 0 + 2 + 1 =  $11_{10}$
- Using 32 bits
  - 0 to +4,294,967,295



#### 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  $\frac{0}{2}$   $\frac{0000}{0001}$   $\frac{0000}{0001}$   $\frac{1000}{0001}$   $\frac{1000}{0001}$   $\frac{1000}{0001}$   $\frac{1000}{0001}$   $\frac{1000}{0001}$   $\frac{1000}{0001}$   $\frac{1000}{0001}$
- Example -2<sup>31</sup> -> 2<sup>31</sup>-1
  - $= -2,147,483,648 + 2,147,483,644 = -4_{10}$
- Using 32 bits
  - -2,147,483,648 to +2,147,483,647

need to reload

#### **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



(1) Invert the digits in positive binary form

# Signed Negation

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

$$x + x = 1111...111_2 = -1$$
  
 $x + 1 = -x$ 

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

```
addi $t2, $t3, 5 # $t2= $t3+5; "add immediale"
16 register-dest, RAML STC
     # copylbyte at source RAM to low-order byte of
     # desdination register, and sign extend the rest 24 bit
     # with the msb of that 4 byte.
         Evample: $t0 = 0x12121212 Julia > [88, 77, 64, 55]
                  Q: lb $+0, 0($+1)
              "Ib" load 1 byte from $+1 +0 = $+1 = 88.
                           1 byte 1 byte 1 byte 1 byte.
                 16 takes the msb of 0x88 which is I and
                 fill up to the higher-order bytes (rest 24 bits) with 'I'
           => $+0= 1111 HH | 1111 HH 1111 | 1000 1000 |
                     = 0x ffffff 88
   Ih register dest, RAM- sic
        # similar to lb but load & bytes (least significant)
     beg $+0, $+1, target
           # branch to target if $+0 = $+1
```

#### **Unit 3 Hoemwork**

- 1. Convert following C code to MIPS Assembly code? A[8] =g + h A[3]? Suppose g, h, A are assigned to MIPS registers \$s1, \$s2, \$s3.
- 2. Use 2'complement to convert following binary numbers to decimal values?
  - (1) 1110 1101 1011 11012 (2) 1111 1001 1110 10112 ?
- 3. Convert following decimal numbers to 2's complement binary number?
  - (1) -810 (2) -1510

410-910=410+(-910) 410 = 0001 1001 1010. -910= 1100 0111 0010. (21s)

410 0000 1100 (21c)

- 4. What is the result in 2's complement?
  - (1) -510 + 310 (2) 410 910
  - 5. MIPS is using the 32 bits register. What is the biggest positive numbers in binary? in decimal? What is the most negative number in binary? and decimal?

```
$51 $2 $3
1) A[8] = g+h - A[3]
                                     810 = 0000 0011 0010 1010.
  lw $+0, 12($53)
                                     invert: 1111 1100 1101 0101
  sub $+0, $2, $+0
                                     +1 1111 1100 1101 0110.
  add $to, $s1,$to
  sw $10,32($53)
                                      - 1510.
2) 1110 1101 1011 1101
                                    1510 = 0000 0101 1110 0110
                                   invert => 1111 1010 0001 1001
   1110 11 01 1 MI 1100
                                    +1 => 1111 1010 0001 1010.
  0001. 0010 0100 0011.
                                 4\ -510 +310
                                    -510: 510 = 0001 1111 1110
  = -4675
                                          -510 = 1110 0000 0001
 1111 1001 1110 1011.
                                                 1110 0000 0010 (2's
 1111 1001 1110 1010.
                                           310 = 0001 0011 0110
0000 0110 0001 0101
                                                 11/1/00/1/1000 (010)
  =-1557
```

```
5) Biggest positive number
-in binary 0411 1111 1111 1111 1111 1111 1111
- in decimal 231-1 = 2147 1836 47
    Most negative number.
- in binary: MOO 0000 0000 0000 0000 0000
- in decimal: -231= -2147 4836 48
```

