Chapter 2

Instructions: Language of the Computer

① R-Type instructions : 3 registers
add sub mult div
slt and or
opcode = 0

② I-Type instructions : 2 registers
lw sw addi
opcode <> 0

③ Branch instruction : 2 registers
beq bne

all start with 6-bit opcode.

Ref: page 64. txt bk.
— 80 —
— 86 —

Instruction Set

- The repertoire of instructions of a computer
  collection
- 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 (www.mips.com)
- 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

output input comment

```
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

MK MORGAN KAUFMANN

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 32x32 bit registers
Each 32bit is a 'word'!
$t0 → $t9 : temporary value.
$s0 → $s7 : saved value (input, output)

- Arithmetic instructions use register operands
- MIPS has a 32 × 32-bit register file
  - Use for frequently accessed data
  - Numbered 0 to 31
  - 32-bit data called a "word"

4bytes

→ in MIPS, unit is byte (8 bit).
MIPS instruction uses 4 bytes for each instruction.

- Assembler names
  - G1 $t0, $t1, …, $t9 for temporary values
  - G2 $s0, $s1, …, $s7 for saved variables

fixed names for temp registers. : only 10 registers are reserved.

for input/output

: only 8 registers are reserved.

- *Design Principle 2:* Smaller is faster
  - c.f. main memory: millions of locations

this is why MIPS does not increase their size. more than 32×32.

Principle 1: Simplicity favors regularity

least significant bit

most significant bit

Register Operand Example

- C code:

```
$s0      $s1    $s2        $s3      $s4
f = (g + h) - (i + j);
```

  - f, …, j in \$s0, …, \$s4
- Compiled MIPS code:

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

Memory Operands

Load / Store Instructions:

load: lw register_dest, RAM_source
copy word (4 bytes) from source RAM to dest register.

store: sw register_src, RAM_dest
store word in source register into RAM destination.

- 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

load data : lw
save data : sw

- 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

Example: $\overbrace{\text{A2C3D1E0}}^{\text{32 bit}}$

0000 0000 0000)4
0100)4
1000

	8	8	8	8
Big Edian	A2			
Smal Edian				A2

least addr

Mem Address	Value.
1000	A2
1001	C3
1002	D1
1003	EØ

most significant byte.
least significant byte.

BIG EDIAN

Memory Operand Example 1

- C code:

  for array, the data must have been stored in memory. So we have to load their value to a temp register b4 process. Always times 4 to the index. when converting to MIPS code.

  $s1 $s2 $s3

  ```
  g = h + A[8];
  ```

  - g in $s1, h in $s2, base address of A in $s3
- Compiled MIPS code:
  - x4
  - 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:

$s2 $s3

```
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
```

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
```

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

Principle 1: Simplicity favors regularity.
Principle 2: Smaller runs faster.
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 so there are 19 registers
- Useful for common operations
  - E.g., copy or move between registers

```
add $t2, $s1, $zero
```

Unsigned Binary Integers

- Given an n-bit number

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

- Range: 0 to $+2^n - 1$

unsigned : $2^n - 1$ biggest no.

signed : $2^{n-1} - 1$ biggest no.

- 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

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

  $\underbrace{}$ $\to 2^{31} \to 2^{31} - 1$

  L:

  - $1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1100_2$

    MS LS.

    $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^2 + 0\times2^1 + 0\times2^0$

    $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = -4_{10}$

- Using 32 bits
  - –2,147,483,648 to +2,147,483,647

		negate	2s complement	
0	0000			
1	0001	$\longrightarrow$	1110.	−1
2	0010	$\longrightarrow$	1101.	−2
3	0011	$\longrightarrow$	1100	−3
4	0100	$\longrightarrow$	1011.	−4

need to reload from new slides

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

How to calculate 2s' complement:
(1) Invert the digits in positive binary form
$2_{10} = 0010$, invert $\rightarrow$ 1101
(2) Then add 1.
$1101 + 1 = 1110 = -2_{10}$

Signed Negation

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

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

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

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

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

```
addi  $t2, $t3, 5  # $t2 = $t3 + 5 ; "add immediate"

lb  register_dest, RAM_src
   # copy 1 byte at source RAM to low-order byte of
   # destination register, and sign extend the rest 24 bit.
   # with the msb of that 1 byte.
     Example:  $t0 = 0x12121212
               $t1 = 0x1000 0000  --value-->  [88, 77, 66, 55]
          Q: lb $t0, 0($t1)
        'lb' load 1 byte from $t1 + 0 = $t1 = 88.
         0x88 = 0b1000 1000
   =>  $t0 = |   ?    |   ?    |   ?    | 1000 1000 |
               1 byte   1 byte   1 byte    1 byte.
         'lb' takes the msb of 0x88 which is 1 and.
          fill up to the higher-order bytes (rest 24 bits)
          with '1'
   =>  $t0 = 1111 1111 | 1111 1111 | 1111 1111 | 1000 1000 |
           = 0xffffff88

lh  register_dest, RAM_src
   # similar to lb but load 2 bytes (least significant)

beq  $t0, $t1, target
      # branch to target if $t0 = $t1
blt         "           <   "
ble         "           ≤   "
bgt         "           >   "
bge         "           ≥   "
bne         "           <>  "
```

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
- 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 ?

$s3 $s1 $s2 $s3
1) A[8] = g + h - A[3]

```
lw  $t0, 12($s3)
sub $t0, $s2, $t0
add $t0, $s1, $t0
sw  $t0, 32($s3).
```

2) 1110 1101. 1011 1101.
= 1
1110 1101 1011 1100
positive form:
0001. 0010 0100 0011.
= -4675

1111 1001 1110 1011.
− 1
1111 1001 1110 1010.
invers 0000 0110 0001 0101.
= -1557

3) -810
810 = 0000 0011 0010 1010.
invert: 1111 1100 1101 0101
+1 1111 1100 1101 0110.

-1510.
1510 = 0000 0101 1110 0110.
invert => 1111 1010 0001 1001.
+1 => 1111 1010 0001 1010.

4) -510 + 310
-510: 510 = 0001 1111 1110
-510 = 1110 0000 0001
+ 1
1110 0000 0010 (2's)
+
310 = 0001 0011 0110
1111 0011 1000 (2's)

410 - 910 = 410 + (-910).
410 = 0001 1001 1010.
+ -910 = 1100 0111 0010. (2's)
1110 0000 1100 (2's).

5) Biggest positive number
- in binary 0111 1111 1111 1111 1111 1111 1111 1111.
- in decimal $2^{31}-1$ = 2147 4836 47

Most negative number.
- in binary: 1100 0000 0000 0000 0000 0000 0000 0000
- in decimal: -2^{31} = -2147 4836 48