Chap.2 Instructions：Language of the Computer

Chap.2 指令系统：计算机的语言

- 2.1 Introduction
- 2.2 Operations of the Computer Hardware
- 2.3 Operands of the Computer Hardware
- 2.4 Signed and Unsigned Numbers
- 2.5 Representing Instructions in the Computer
- 2.6 Logical Operations
- 2.7 Instructions for Making Decisions
- 2.8 Supporting Procedures in Computer Hardware
- 2.9 Communicating with People

Chap.2： Instructions: Language of the Computer

Chap.2 指令系统：计算机的语言

- 2.10 MIPS Addressing for 32-Bit Immediates and More Complex Addresses
- 2.11 Parallelism and instructions: Synchronization
- 2.12 Translating and Starting a Program
- 2.13 A C Sort Example to Put it All Together
- 2.14 Arrays versus Pointers
- 2.15 Advanced Material: Compiling C and Interpreting Java
- 2.16 Real Stuff：ARM Instructions
- 2.17 Real Stuff：x86 Instructions

2.1 Introduction

设计目标

- computer designers have a common goal：

  To find a language that makes it easy to build the hardware and the compiler while maximizing performance and minimizing cost and power.

- Instruction / Instruction Set：指令 / 指令集

  the words of a computer's language are called instructions, and its vocabulary is called an instruction set.

2.4 Signed and Unsigned Numbers

two's complement representation

有符号和无符号数的表示：2的补码

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = 0_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = 1_{ten}$

$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = 2_{ten}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = 2{,}147{,}483{,}645_{ten}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = 2{,}147{,}483{,}646_{ten}$

$0111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = 2{,}147{,}483{,}647_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_2 = -2{,}147{,}483{,}648_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_2 = -2{,}147{,}483{,}647_{ten}$

$1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0010_2 = -2{,}147{,}483{,}646_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1101_2 = -3_{ten}$

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110_2 = -2_{ten}$

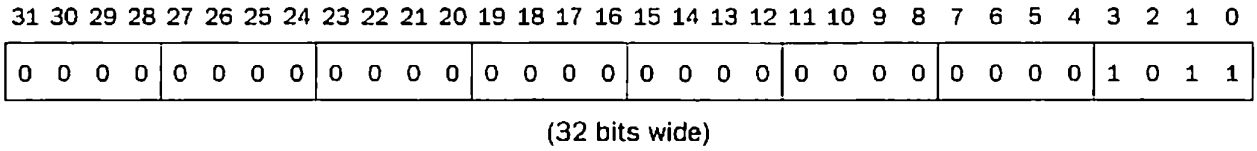
$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_2 = -1_{ten}$

2.4 Signed and Unsigned Numbers

two's complement representation

有符号和无符号数的表示：2的补码

We number the bits 0, 1, 2, 3, . . . from *right to left* in a word. The drawing below shows the numbering of bits within a MIPS word and the placement of the number 1011_{two}: Bit31act as the *sign bit*



$$Value = (bit31 \times (-2^{31})) + (bit30 \times 2^{30}) + (bit29 \times 2^{29}) + ... + (bit1 \times 2^{1}) + (bit0 \times 2^{0})$$

2.4 Signed and Unsigned Numbers

two's complement representation

有符号和无符号数的表示：2的补码

■ Negation Shortcut 求负

因为： $x + \overline{x} = 111...111_{two} = -1$

所以： $x + \overline{x} + 1 = 0$

即： $\overline{x} + 1 = -x$

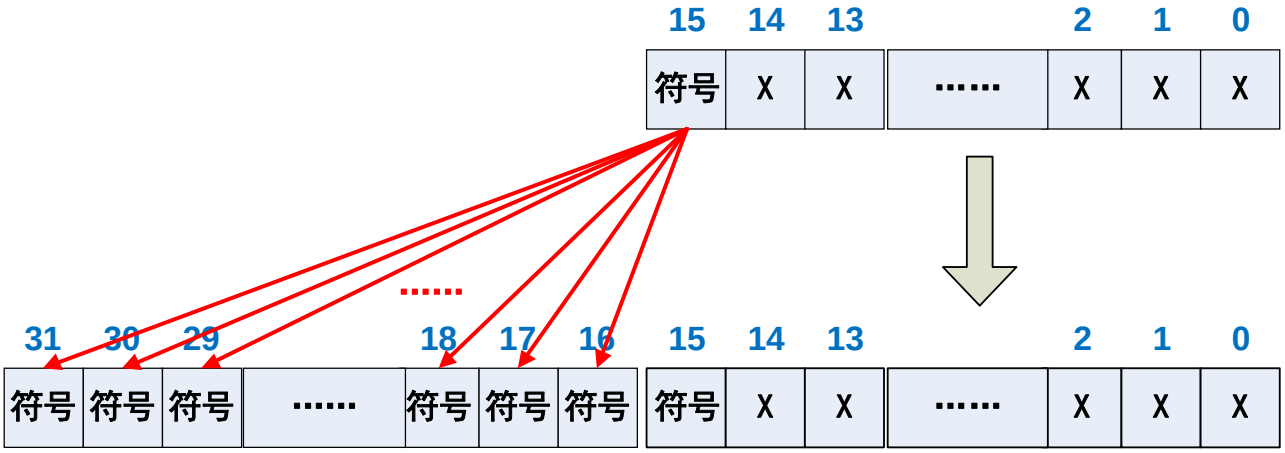
该方法对 正变负、负变正 均适用。

2.4 Signed and Unsigned Numbers

two's complement representation

有符号和无符号数的表示：2的补码

■ Sign Extension Shortcut 符号扩展



2.4 Signed and Unsigned Numbers

Summary and Check Yourself

The main point of this section is that we need to represent both positive and negative integers within a computer word, and although there are pros and cons to any option, the overwhelming choice since 1965 has been two's complement.

What is the decimal value of this 64-bit two's complement number?

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1000_{two}$

1) -4_{ten}

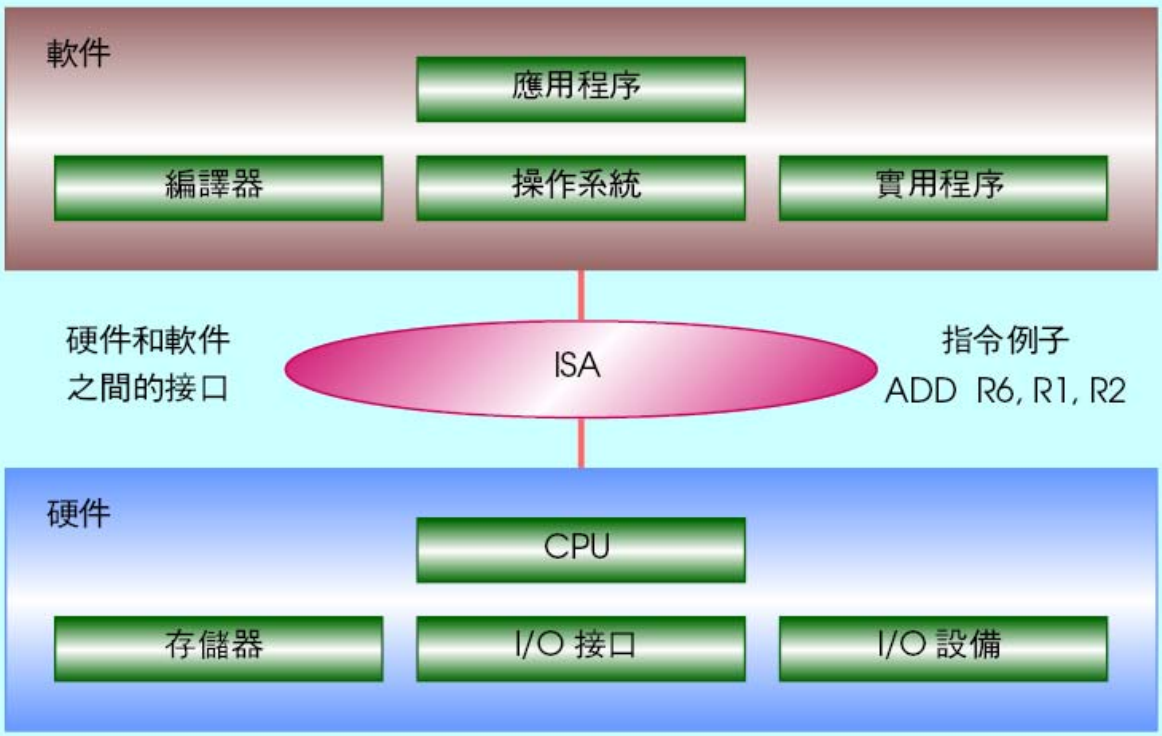
2) -8_{ten}

3) -16_{ten}

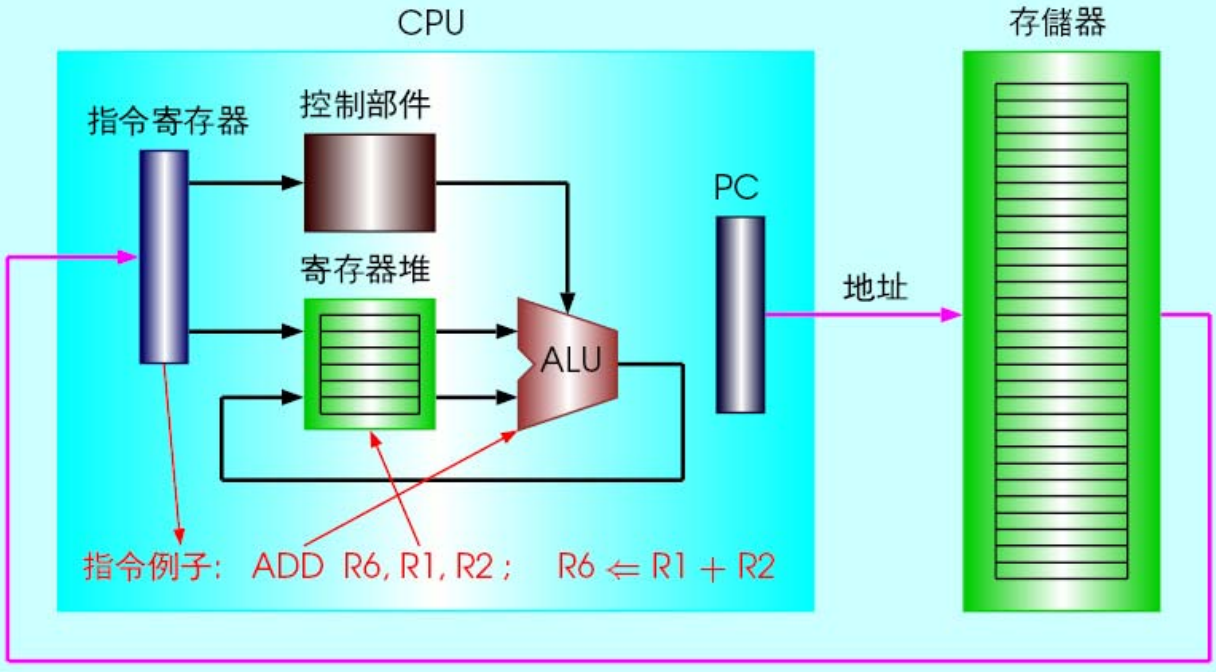
4) $18,446,744,073,709,551,609_{ten}$

注：可参考另外PPT

2.5 Representing Instructions in the Computer



CPU和存储器



2.5 Representing Instructions in the Computer

EXAMPLE Translating a MIPS Assembly Instruction into a Machine Instruction

■ add $t0, $s1, $s2

The decimal representation is

0	17	18	8	0	32

This instruction can also be represented as fields of binary numbers as opposed to decimal:

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
R型指令	操作数1	操作数2	目的操作数		add加法运算

2.5 Representing Instructions in the Computer

关于指令格式和机器语言

■ 指令格式/ Instruction Format

□ The layout of the instruction is called the instruction format.

□ 指令一般由多个字段组成，总长度相对固定，可设计有一种或几种长度。MIPS所有指令都为32bit长。

■ 机器指令/ Machine Language

□To distinguish from assembly language, we call the numeric version of instructions machine language and a sequence of such instructions machine code.

□ 用二进制或十六进制编码表示的指令。

■ 16进制数/ Hexadecimal

□ 以16为基的一种数据表示形式。

instruction format
A form of representation of an instruction composed of fields of binary numbers.

machine language
Binary representation used for communication within a computer system.

hexadecimal
Numbers in base 16.

补充说明：关于数的16进制表示

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

FIGURE 2.4 The hexadecimal-binary conversion table. Just replace one hexadecimal digit by the corresponding four binary digits, and vice versa. If the length of the binary number is not a multiple of 4, go from right to left.

- 分别采用0,1,2,3,4,5,6,7,8,9,a/A, b/B, c/C, d/D, e/E, f/F表示16进制数的16个数字符号。（要非常熟悉）

【例】数的16进制表示方式

EXAMPLE

Binary to Hexadecimal and Back

ANSWER

Convert the following hexadecimal and binary numbers into the other base:

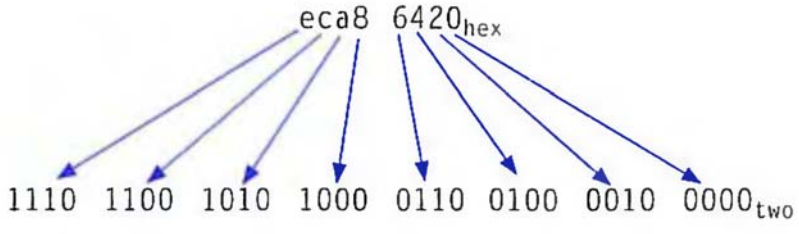
eca8 6420_{hex}

0001 0011 0101 0111 1001 1011 1101 1111_{two}

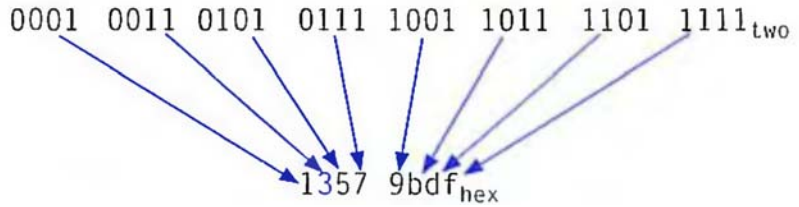
Using Figure 2.4, the answer is just a table lookup one way:

C和JAVA语言的表达方式：

0xnnnn



And then the other direction:



MIPS的R-型（Register/寄存器型）指令格式

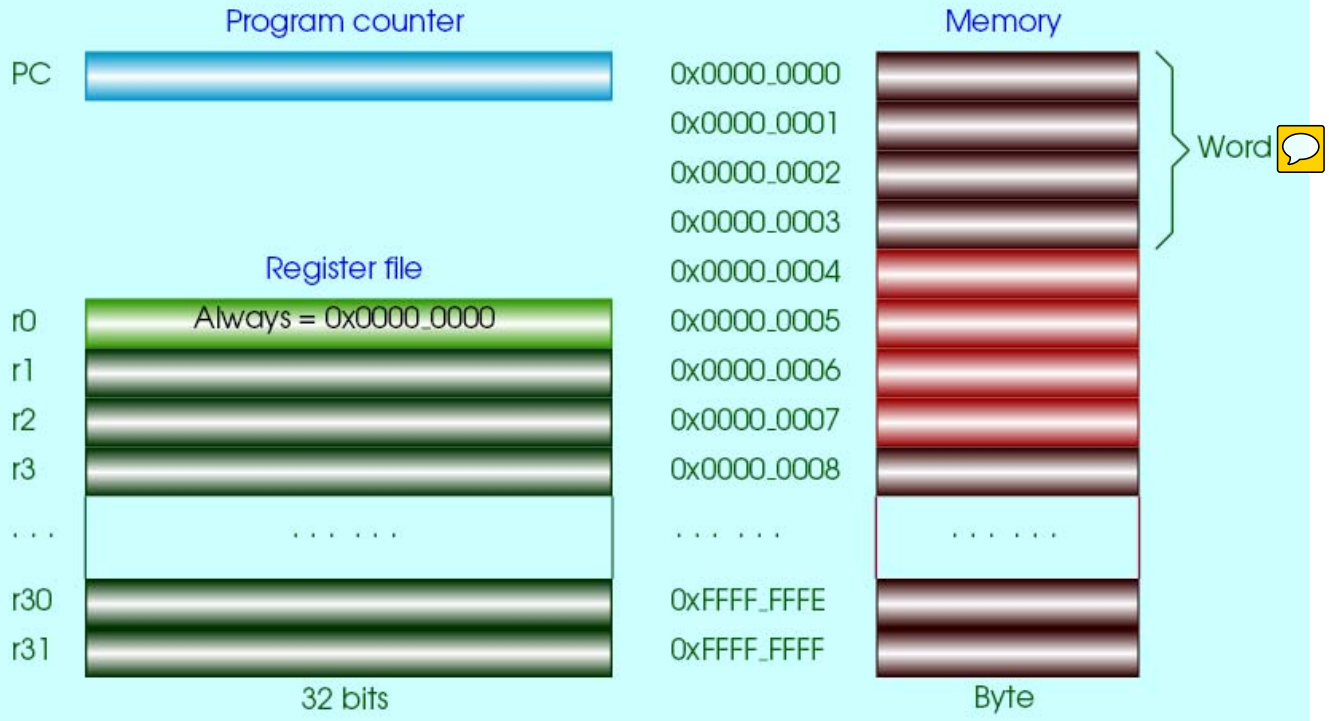
MIPS Fields MIPS R型指令的格式及各个字段的含义

MIPS fields are given names to make them easier to discuss:

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

- *op:* Basic operation of the instruction, traditionally called the **opcode**.
- *rs:* The first register source operand.
- *rt:* The second register source operand.
- *rd:* The register destination operand. It gets the result of the operation.
- *shamt:* Shift amount. (Section 2.6 explains shift instructions and this term; it will not be used until then, and hence the field contains zero in this section.)
- *funct:* Function. This field, often called the *function code*, selects the specific variant of the operation in the op field.

MIPS CPU的寄存器和存储器

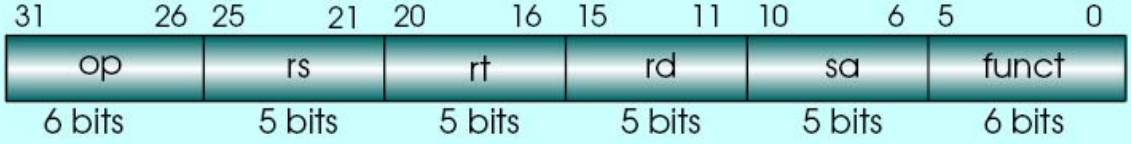


MIPS指令的功能分类

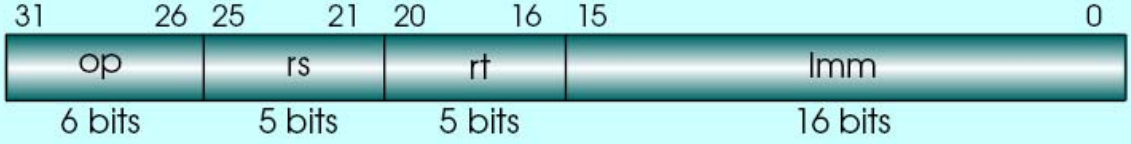
- 算術運算
  add, sub, addi, addu, mul, mulu, div, divu
- 邏輯運算
  and, or, andi, ori, sll, srl
- 數據傳送
  lw, sw, lb, lbu, sb
- 條件轉移
  beq, bne, bnez, slt, slti, sltu, sltiu
- 無條件跳轉
  j, jr, jal

MIPS指令的三种格式

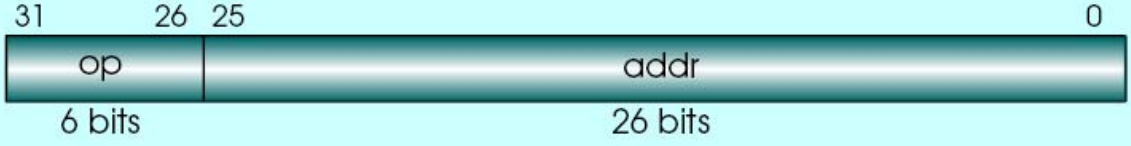
R 類型指令 (rs, rt, rd: 寄存器號, sa: 移位位數)



I 類型指令 (rs, rt: 寄存器號, imm: 立即數)



J 類型指令 (addr: 跳轉目標地址)



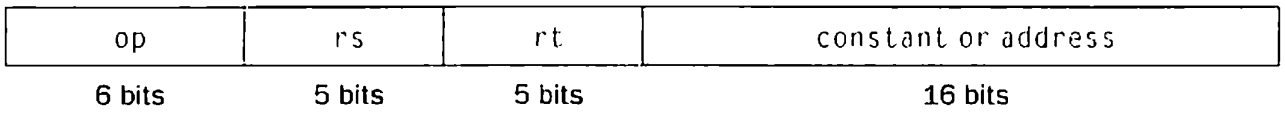
MIPS的 I-型（Immediate / 立即数型）指令格式

设计原理4：良好的设计即是合理的折衷

Design Principle 4: Good design demands good compromises.

The compromise chosen by the MIPS designers is to keep all instructions the same length, thereby requiring different kinds of instruction formats for different kinds of instructions. For example, the format above is called *R-type* (for register) or *R-format*. A second type of instruction format is called *I-type* (for immediate) or *I-format* and is used by the immediate and data transfer instructions. The fields of I-format are

MIPS的 I-型（Immediate/立即数型）指令格式



The 16-bit address means a load word instruction can load any word within a region of $\pm 2^{15}$ or 32,768 bytes ($\pm 2^{13}$ or 8192 words) of the address in the base register rs. Similarly, add immediate is limited to constants no larger than $\pm 2^{15}$.

2.5 Representing Instructions in the Computer

MIPS 指令格式举例

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

FIGURE 2.5 MIPS instruction encoding. In the table above, "reg" means a register number between 0 and 31, "address" means a 16-bit address, and "n.a." (not applicable) means this field does not appear in this format. Note that add and sub instructions have the same value in the op field; the hardware uses the funct field to decide the variant of the operation: add (32) or subtract (34).

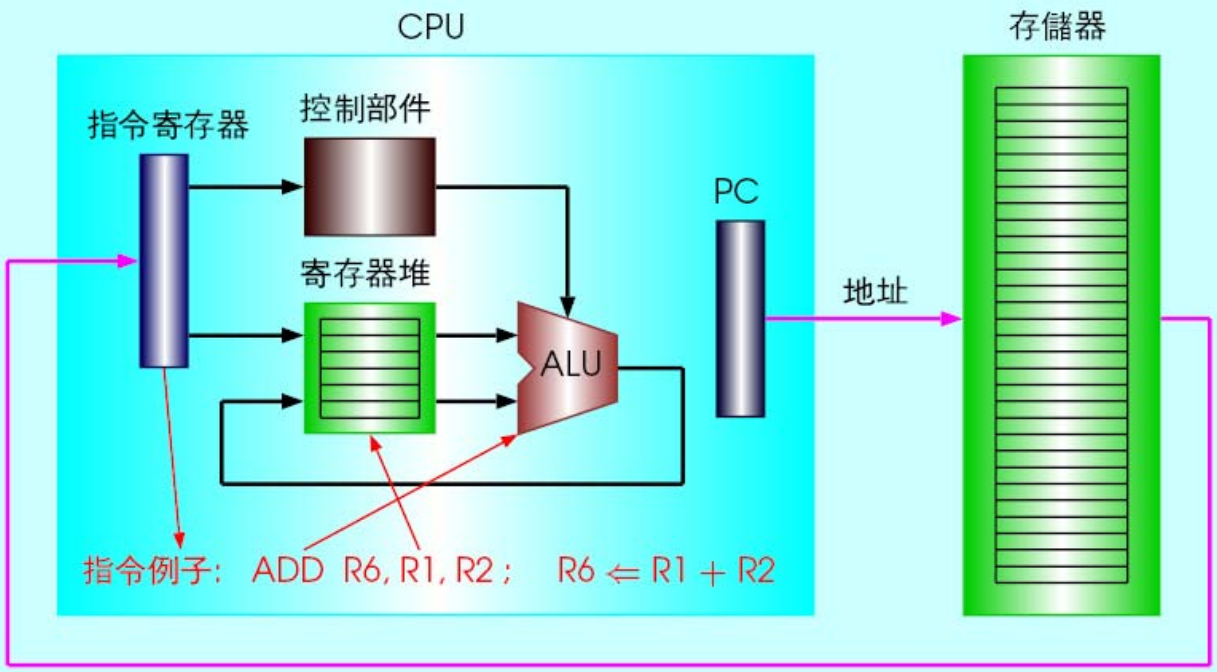
MIPS的部分指令

```
add  rd, rs, rt   ; rd <-- rs + rt
sub  rd, rs, rt   ; rd <-- rs - rt
and  rd, rs, rt   ; rd <-- rs & rt
or   rd, rs, rt   ; rd <-- rs | rt
xor  rd, rs, rt   ; rd <-- rs ^ rt
sll  rd, rt, sa   ; rd <-- rt << sa
srl  rd, rt, sa   ; rd <-- rt >> sa (logical)
sra  rd, st, sa   ; rd <-- rt >> sa (arithmetic)
jr   rs           ; PC <-- rs
addi rt, rs, imm  ; rt <-- rs + (sign)imm
andi rt, rs, imm  ; rt <-- rs & (zero)imm
ori  rt, rs, imm  ; rt <-- rs | (zero)imm
xori rt, rs, imm  ; rt <-- rs ^ (zero)imm
lw   rt, imm(rs)  ; rt <-- memory[rs + (sign)imm]
sw   rt, imm(rs)  ; memory[rs + (sign)imm] <-- rt
beq  rs, rt, imm  ; if (rs == rt) PC <-- PC + 4 + (sign)imm << 2
bne  rs, rt, imm  ; if (rs != rt) PC <-- PC + 4 + (sign)imm << 2
lui  rt, imm      ; rt <-- imm << 16
j    addr         ; PC <-- (PC+4)[31..28],addr<<2
jal  addr         ; $31 <-- PC+4; PC <-- (PC+4)[31..28],addr << 2
```

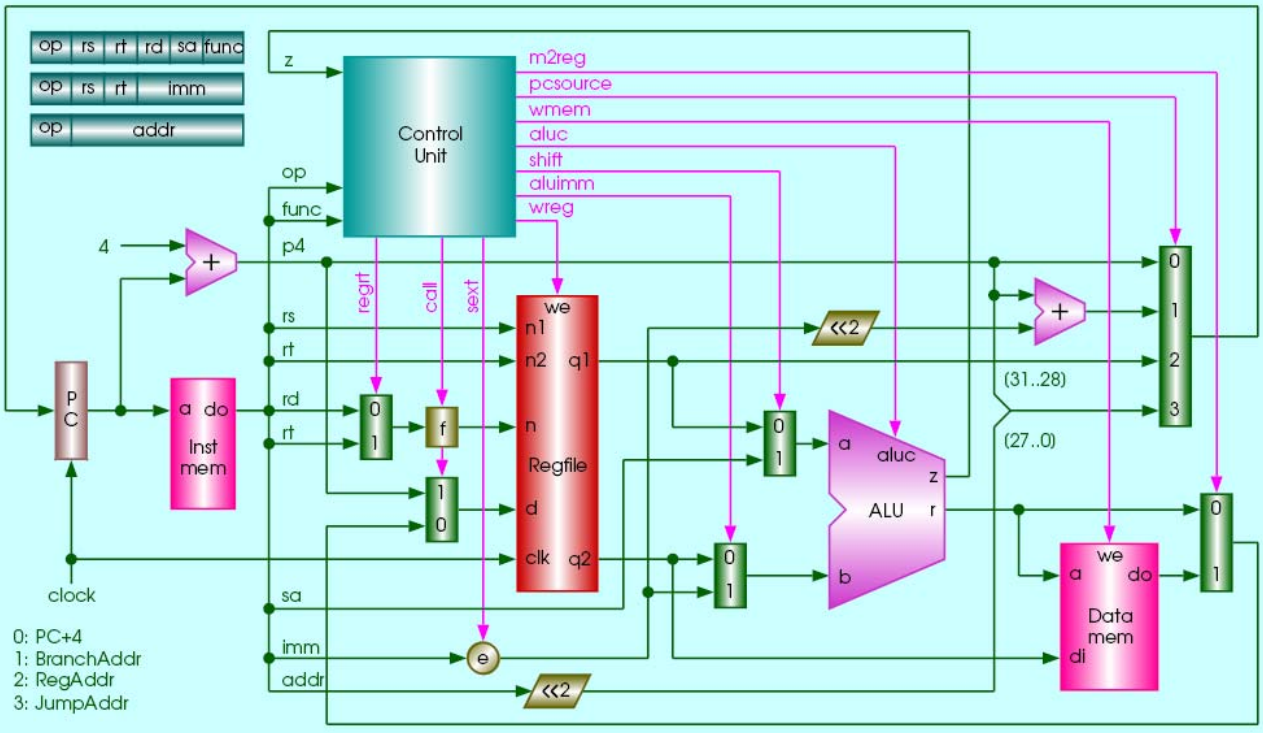

MIPS的部分指令编码

```
op     rs    rt    rd    sa    func   ; R format instructions
000000 rs    rt    rd    00000 100000 ; add  rd, rs, rt
000000 rs    rt    rd    00000 100010 ; sub  rd, rs, rt
000000 rs    rt    rd    00000 100100 ; and  rd, rs, rt
000000 rs    rt    rd    00000 100101 ; or   rd, rs, rt
000000 rs    rt    rd    00000 100110 ; xor  rd, rs, rt
000000 00000 rt    rd    sa    000000 ; sll  rd, rt, sa
000000 00000 rt    rd    sa    000010 ; srl  rd, rt, sa
000000 00000 rt    rd    sa    000011 ; sra  rd, st, sa
000000 rs    00000 00000 00000 001000 ; jr   rs
op     rs    rt    imm                ; I format instructions
001000 rs    rt    imm                ; addi rt, rs, imm
001100 rs    rt    imm                ; andi rt, rs, imm
001101 rs    rt    imm                ; ori  rt, rs, imm
001110 rs    rt    imm                ; xori rt, rs, imm
100011 rs    rt    imm                ; lw   rt, imm(rs)
101011 rs    rt    imm                ; sw   rt, imm(rs)
000100 rs    rt    imm                ; beq  rs, rt, imm
000101 rs    rt    imm                ; bne  rs, rt, imm
001111 00000 rt    imm                ; lui  rt, imm
op     addr                          ; J format instructions
000010 addr                          ; j    addr
000011 addr                          ; jal  addr
```

CPU和存储器



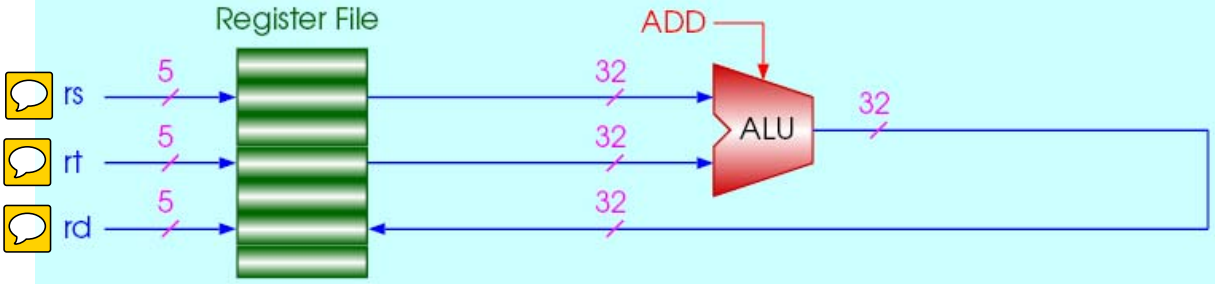
单周期CPU+指令存储器+数据存储器



add指令（Add）

```
add   rd, rs, rt   ; rd = rs + rt
```

op	rs	rt	rd	sa	funct
000000	rs	rt	rd	00000	100000
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit

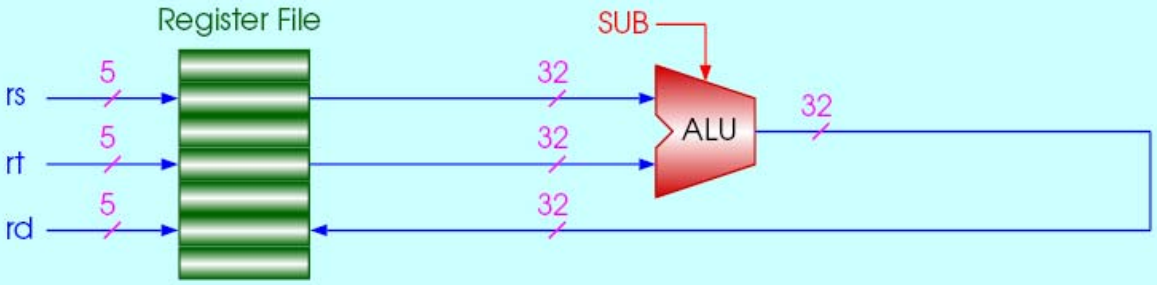


指令的意義：寄存器rs中的數據和寄存器rt中的數據相加，結果存放在寄存器rd中

sub指令（Subtract）

```
sub   rd, rs, rt   ; rd <-- rs - rt
```

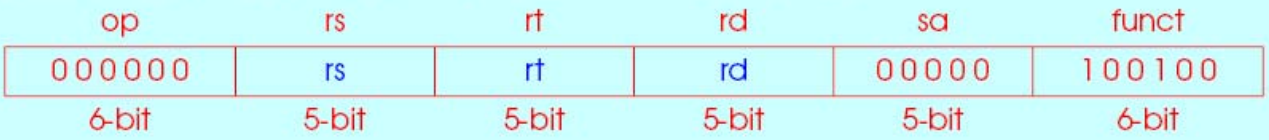
op	rs	rt	rd	sa	funct
000000	rs	rt	rd	00000	100010
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit

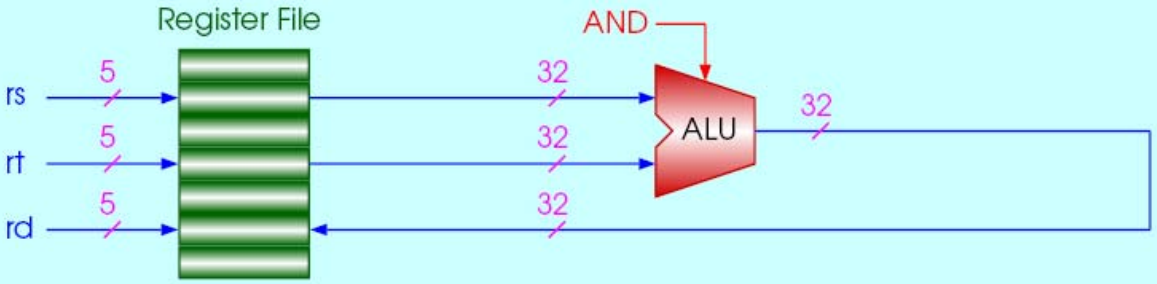


指令的意義：寄存器rs中的數據和寄存器rt中的數據相減，結果存放在寄存器rd中

and指令（And）

```
and   rd, rs, rt   ; rd <-- rs & rt
```

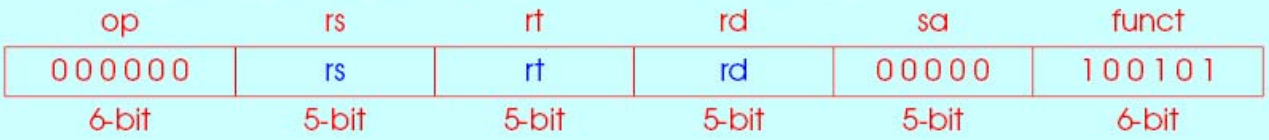


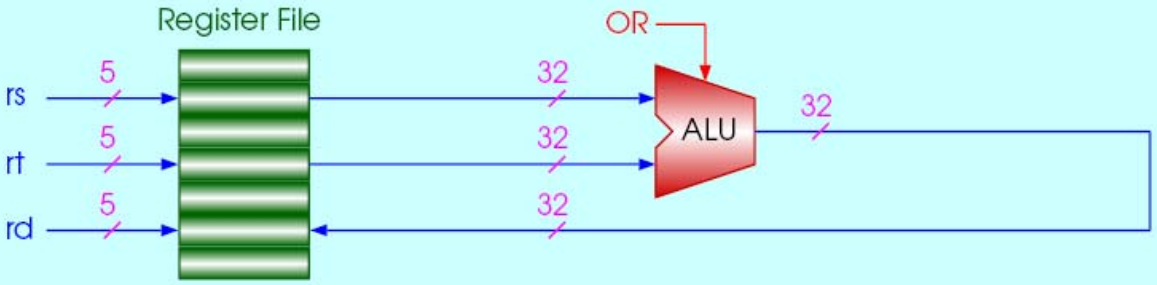


指令的意義: 寄存器rs中的數據和寄存器rt中的數據相與, 結果存放在寄存器rd中

or指令（Or）

```
or    rd, rs, rt   ; rd <-- rs | rt
```



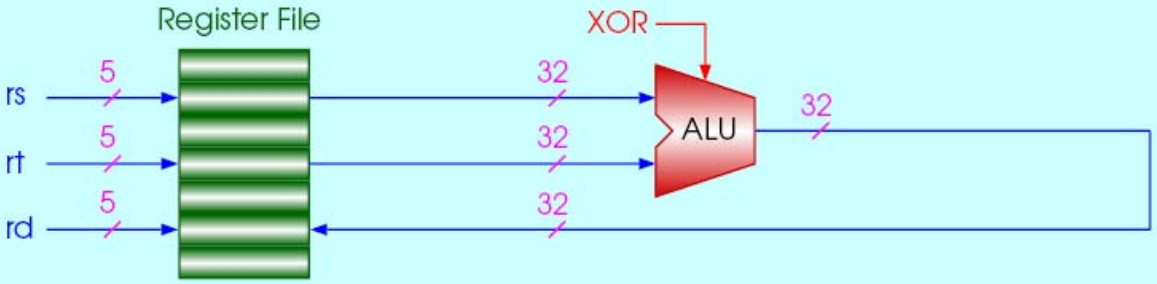


指令的意義: 寄存器rs中的數據和寄存器rt中的數據相或, 結果存放在寄存器rd中

xor 指令 (Exclusive Or)

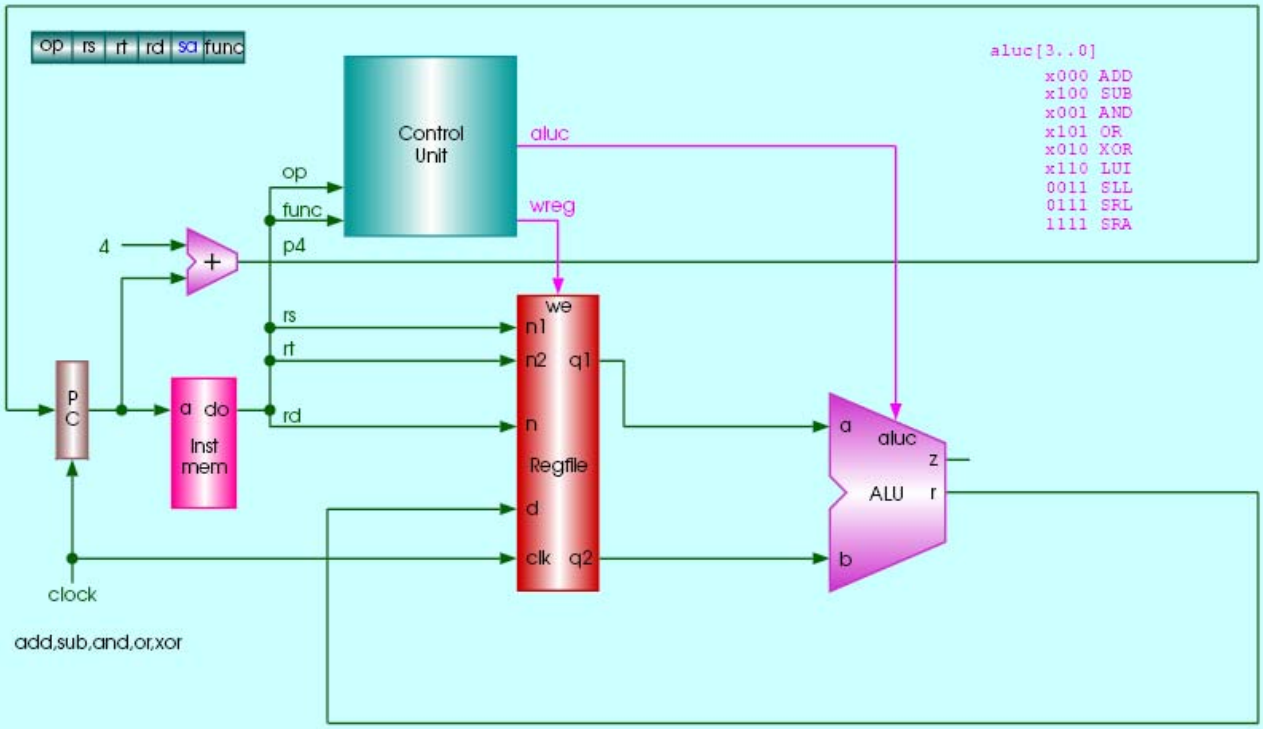
```
xor  rd, rs, rt  ; rd <-- rs ^ rt
```

op	rs	rt	rd	sa	funct
000000	rs	rt	rd	00000	100110
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit



指令的意義: 寄存器rs中的數據和寄存器rt中的數據相异或, 結果存放在寄存器rd中

【例】一种能够执行add/sub/and/or/xor指令的CPU电路/datapath

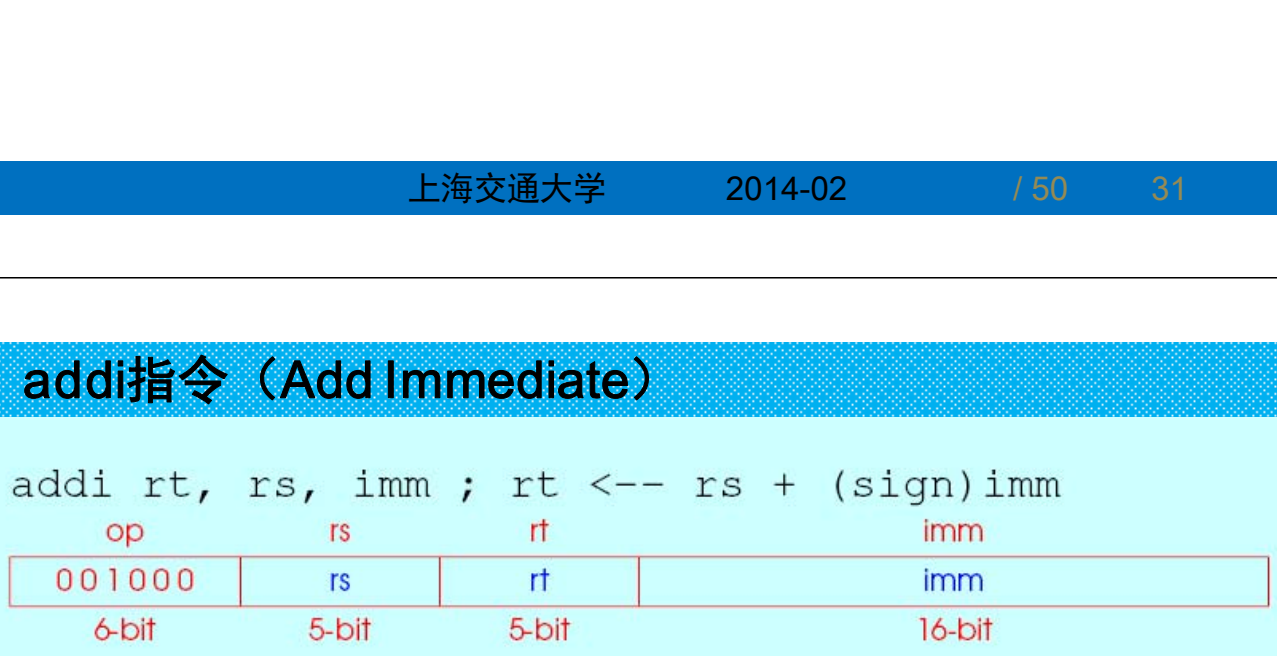


MIPS I-型（Immediate / 立即数型）

指令格式及编码举例

addi指令（Add Immediate）

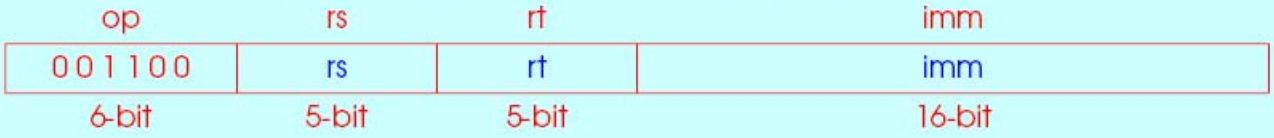
```
addi rt, rs, imm ; rt <-- rs + (sign)imm
```

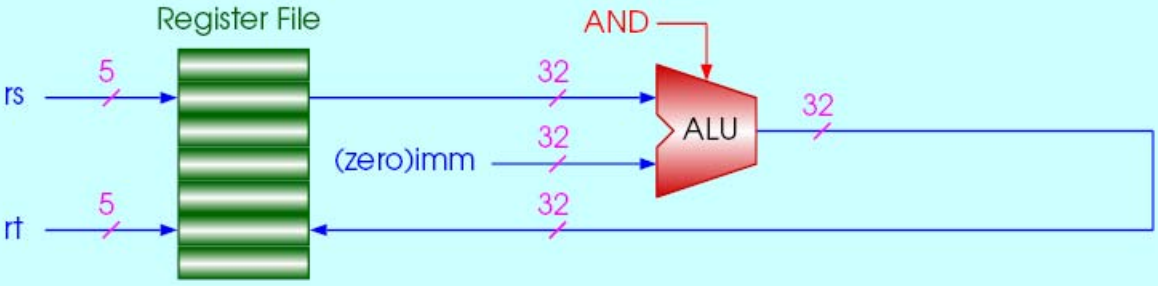


指令的意義: 寄存器rs中的數據和2的補碼表示的立即數imm相加，結果存放在寄存器rt中

andi指令（And Immediate）

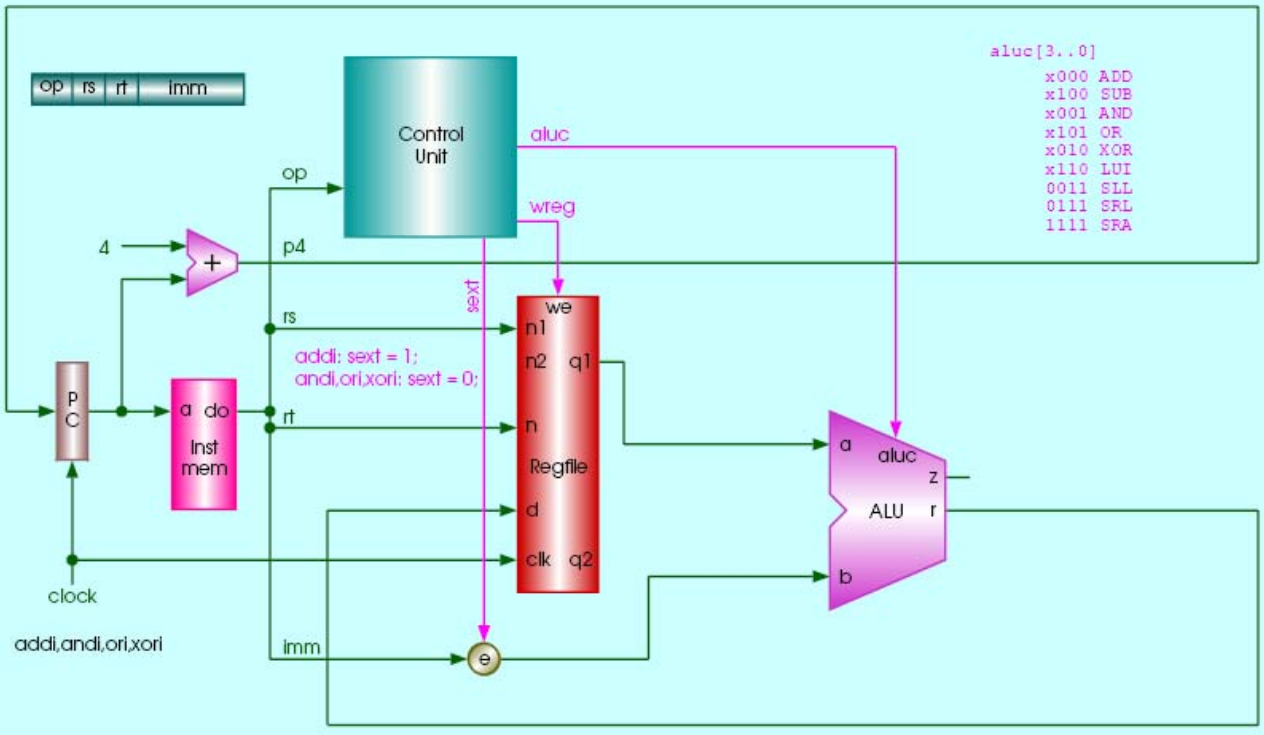
```
andi rt, rs, imm ; rt <-- rs & (zero)imm
```





指令的意義: 寄存器rs中的數據和無符號立即數imm相與, 結果存放在寄存器rt中

【例】一种能够执行addi/andi/ori/xori指令的CPU电路/datapath



符号扩展和零扩展

- `addi rt, rs, imm ; rt <-- rs + (sign)imm`

  指令的意義: 寄存器rs中的數據和2的補碼表示的立即數imm相加, 結果存放在寄存器rt中.....(符號擴展)

- `andi rt, rs, imm ; rt <-- rs & (zero)imm`

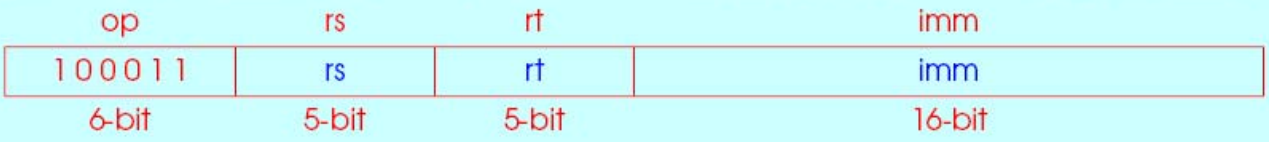
  指令的意義: 寄存器rs中的數據和無符號立即數imm相與, 結果存放在寄存器rt中.....(零擴展)

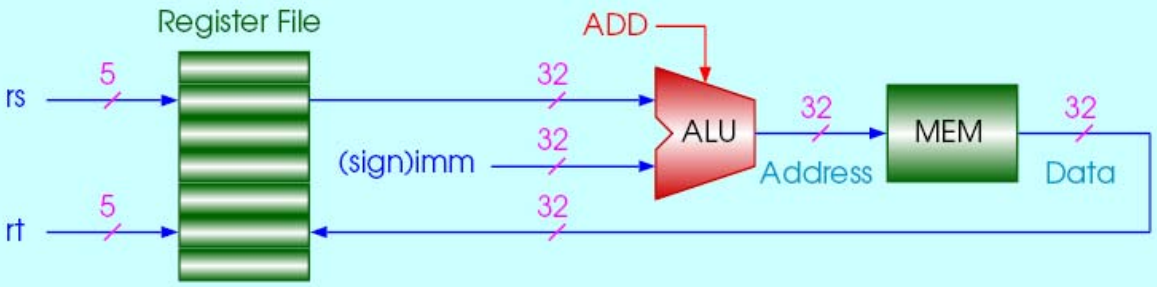
- 符號擴展和零擴展

擴展	16位輸入	32位輸出
符號擴展	0xxxxxxxxxxxxxxx (正)	0000000000000000xxxxxxxxxxxxxxx (負)
	1xxxxxxxxxxxxxxx (負)	1111111111111111xxxxxxxxxxxxxxx (正)
零擴展	xxxxxxxxxxxxxxxx	0000000000000000xxxxxxxxxxxxxxxx

lw指令（Load Word）

`lw rt, imm(rs) ; rt <-- memory[rs + (sign)imm]`



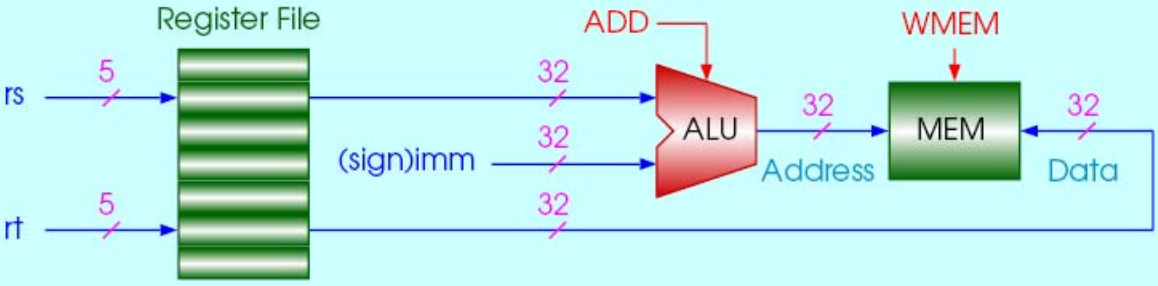


指令的意義: 寄存器rs中的數據和2的補碼表示的立即數imm相加, 得到存儲器地址, 用這個地址訪問存儲器, 把得到的存儲器數據寫入寄存器rt中

sw指令（Store Word）

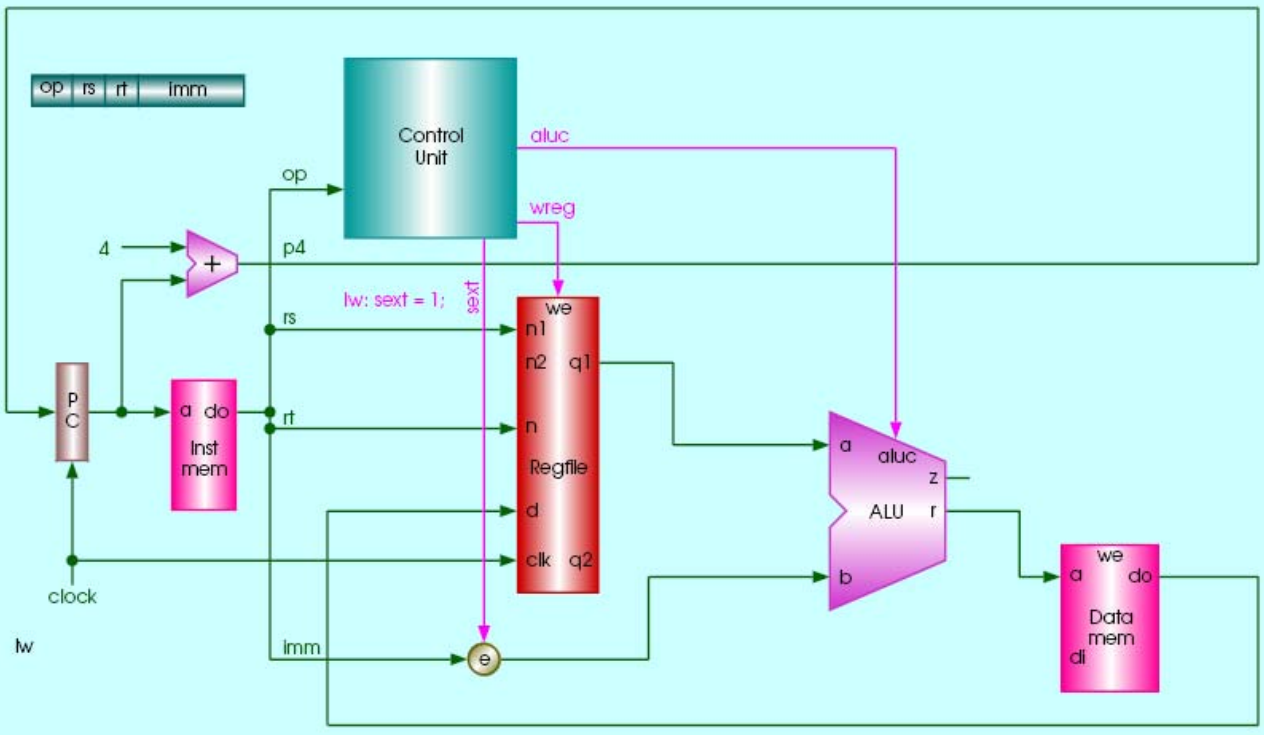
```
sw   rt, imm(rs)   ; memory[rs + (sign)imm] <-- rt
```

op	rs	rt	imm
101011	rs	rt	imm
6-bit	5-bit	5-bit	16-bit

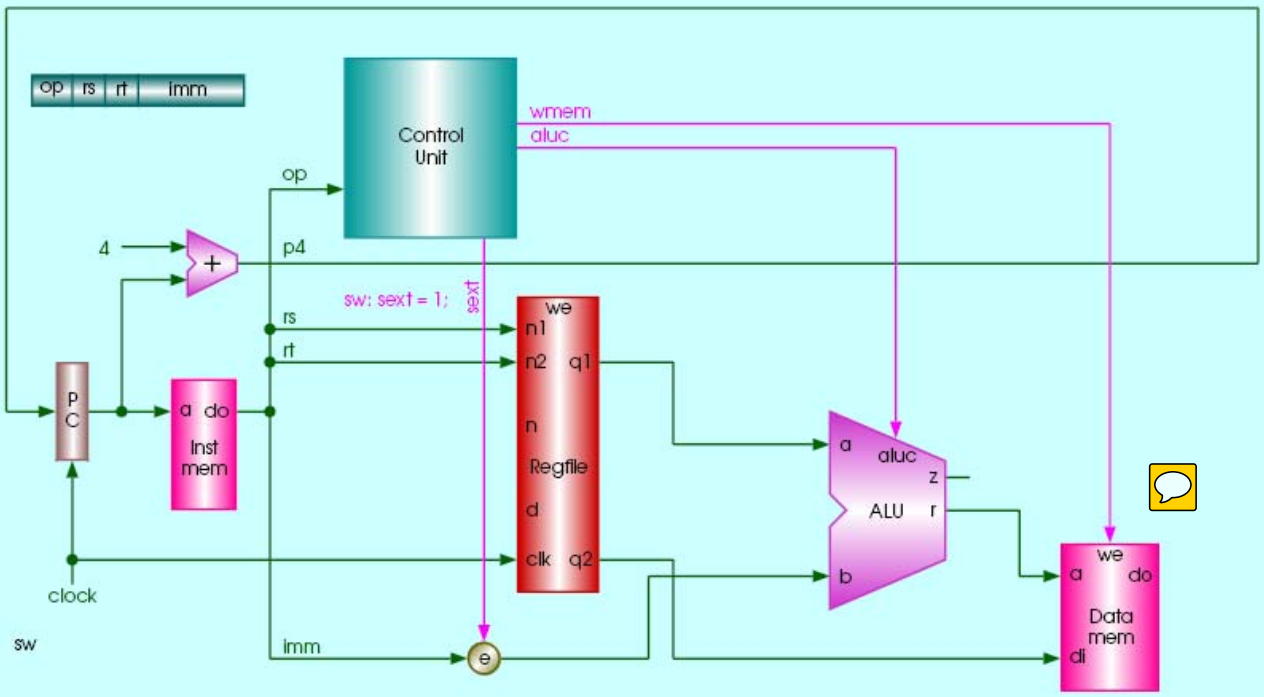


指令的意義: 寄存器rs中的數據和2的補碼表示的立即數imm相加，得到存儲器地址，把寄存器rt中的數據寫入這個地址的存儲器中

【例】一种能够执行lw指令的CPU电路/datapath



【例】一种能够执行sw指令的CPU电路/datapath



2.5 Representing Instructions in the Computer

EXAMPLE

Translating MIPS Assembly Language into Machine Language

We can now take an example all the way from what the programmer writes to what the computer executes. If `$t1` has the base of the array `A` and `$s2` corresponds to `h`, the assignment statement

```
A[300] = h + A[300];
```

is compiled into

```
lw   $t0,1200($t1)# Temporary reg $t0 gets A[300]
add  $t0,$s2,$t0  # Temporary reg $t0 gets h + A[300]
sw   $t0,1200($t1) # Stores h + A[300] back into A[300]
```

What is the MIPS machine language code for these three instructions?

【例】Translating MIPS Assembly Language into Machine Language

ANSWER

For convenience, let's first represent the machine language instructions using decimal numbers. From Figure 2.5, we can determine the three machine language instructions:

op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

机器指令序列的二进制形式：

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

2.5 Representing Instructions in the Computer

MIPS机器语言示例

MIPS machine language

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
addi	I	8	18	17	100			addi $s1,$s2,100
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

FIGURE 2.6 MIPS architecture revealed through Section 2.5. The two MIPS instruction formats so far are R and I. The first 16 bits are the same: both contain an *op* field, giving the base operation; an *rs* field, giving one of the sources; and the *rt* field, which specifies the other source operand, except for load word, where it specifies the destination register. R-format divides the last 16 bits into an *rd* field, specifying the destination register; the *shamt* field, which Section 2.6 explains; and the *funct* field, which specifies the specific operation of R-format instructions. I-format combines the last 16 bits into a single *address* field.

2.5 Representing Instructions in the Computer

The BIG Picture

关于stored-program（存储程序）思想

Today's computers are built on two key principles:

1. Instructions are represented as numbers.
2. Programs are stored in memory to be read or written, just like numbers.

These principles lead to the *stored-program* concept; its invention let the computing genie out of its bottle. Figure 2.7 shows the power of the concept; specifically, memory can contain the source code for an editor program, the corresponding compiled machine code, the text that the compiled program is using, and even the compiler that generated the machine code.

One consequence of instructions as numbers is that programs are often shipped as files of binary numbers. The commercial implication is that computers can inherit ready-made software provided they are compatible with an existing instruction set. Such "binary compatibility" often leads industry to align around a small number of instruction set architectures.

关于stored-program（存储程序）思想

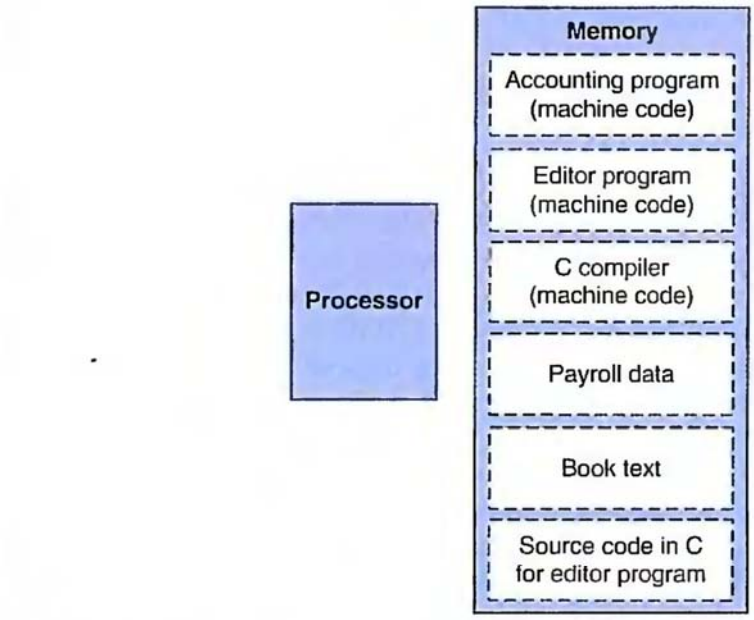


FIGURE 2.7 The stored-program concept. Stored programs allow a computer that performs accounting to become, in the blink of an eye, a computer that helps an author write a book. The switch happens simply by loading memory with programs and data and then telling the computer to begin executing at a given location in memory. Treating instructions in the same way as data greatly simplifies both the memory hardware and the software of computer systems. Specifically, the memory technology needed for data can also be used for programs, and programs like compilers, for instance, can translate code written in a notation far more convenient for humans into code that the computer can understand.

Check Yourself

What MIPS instruction does this represent? Chose from one of the four options below.

op	rs	rt	rd	shamt	funct
0	8	9	10	0	34

1. `add $s0, $s1, $s2`
2. `add $s2, $s0, $s1`
3. `add $s2, $s1, $s0`
4. `sub $s2, $s0, $s1`

Acknowledgements

- These slides contain material developed and copyright by:
  - —Arvind (MIT)
  - —Krste Asanovic (MIT/UCB)
  - —John Kubiatowicz (UCB)
  - —David Patterson (UCB)
  - —Geng Wang (SJTU)
  - —Yanmin Zhu (SJTU)
  - —Li Yamin(Hosei Univ.)