
Lecture 6-2: Arithmetic and Logic Operations and ALU - 1

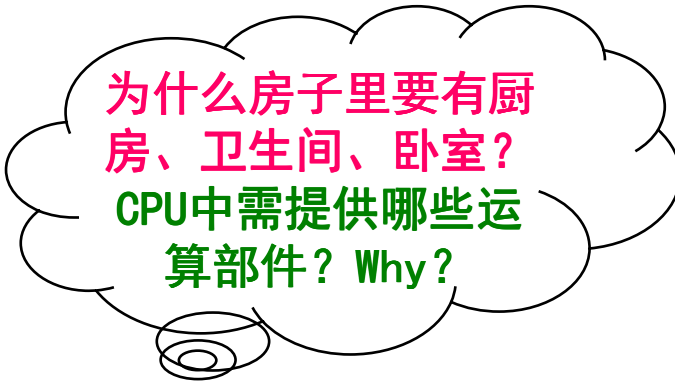
运算方法和运算部件

不同层次程序员看到的运算及**ALU**

不同层次程序员看到的运算及ALU

主要内容

- ◆ 高级语言程序中涉及的运算（以C语言为例）
 - 整数算术运算、浮点数算术运算
 - 按位、逻辑、移位、位扩展和位截断
- ◆ 指令集中涉及到的运算（以MIPS为例）
 - 涉及到的定点数运算
 - 算术运算
 - 带符号整数运算：取负 / 符号扩展 / 加 / 减 / 乘 / 除 / 算术移位
 - 无符号整数运算：0扩展 / 加 / 减 / 乘 / 除
 - 逻辑运算
 - 逻辑操作：与 / 或 / 非 / ...
 - 移位操作：逻辑左移 / 逻辑右移
 - 涉及到的浮点数运算：加、减、乘、除
- ◆ 基本运算部件ALU的设计



为什么房子里要有厨房、卫生间、卧室？
CPU中需提供哪些运算部件？Why？

人的需要！ 程序的需要！

C语言程序中涉及的运算

◆ 算术运算（最基本的运算）

- 无符号数、带符号整数、浮点数的+、-、*、/ 运算

◆ 按位运算

• 用途

- 对位串实现“掩码”（mask）操作或相应的其他处理
（主要用于对多媒体数据或控制信息进行处理）

• 操作

- 按位或：“|”
- 按位与：“&”
- 按位取反：“~”
- 按位异或：“^”

问题：如何从16位采样数据y中提取高位字节，并使低字节为0？

可用“&”实现“掩码”操作： $y \& 0xFF00$

例如，当 $y=0x2C0B$ 时，通过掩码操作得到结果为： $0x2C00$

C语言程序中涉及的运算

◆ 逻辑运算

• 用途

- 用于关系表达式的运算

例如，if (x>y and i<100) then中的“and”运算

• 操作

- “||”表示“OR”运算
- “&&”表示“AND”运算

例如，if ((x>y) && (i<100)) then

- “!”表示“NOT”运算

• 与按位运算的差别

- 符号表示不同：& ~ && ; | ~ || ;
- 运算过程不同：按位 ~ 整体
- 结果类型不同：位串 ~ 逻辑值

C语言程序中涉及的运算

◆ 移位运算

• 用途

- 提取部分信息
- 扩大或缩小数值的2、4、8...倍

• 操作

- 左移:: $x \ll k$; 右移: $x \gg k$
- 不区分是逻辑移位还是算术移位, 由x的类型确定
- 无符号数: 逻辑左移、逻辑右移
 - 高(低)位移出, 低(高)位补0, 可能溢出!
 - 问题: 何时可能发生溢出? 如何判断溢出?
 - 若高位移出的是1, 则左移时发生溢出
- 带符号整数: 算术左移、算术右移
 - 左移: 高位移出, 低位补0。可能溢出!
 - 溢出判断: 若移出的位不等于新的符号位, 则溢出。
 - 右移: 低位移出, 高位补符, 可能发生数据丢失。

C语言程序中涉及的运算

◆ 位扩展和位截断运算

• 用途

- 类型转换时可能需要数据扩展或截断

• 操作

- 没有专门操作运算符，根据类型转换前后数据长短确定是扩展还是截断
- 扩展：短转长
 - 无符号数：0扩展，前面补0
 - 带符号整数：符号扩展，前面补符
- 截断：长转短
 - 强行将高位丢弃，故可能发生“溢出”

例1（扩展操作）：在大端机上输出si, usi, i, ui的十进制和十六进制值是什么？

```
short si = -12345;      si = -12345    CF C7
unsigned short usi = si; usi = 53191    CF C7
int i = si;             i = -12345     FF FF CF C7
unsigned ui = usi;      ui = 53191     00 00 CF C7
```

例2（截断操作）：i和j是否相等？

```
int i = 53191;
short si = (short) i;
int j = si;
```

不相等！

```
i = 53191    00 00 CF C7
si = -12345   CF C7
j = -12345    FF FF CF C7
```

原因：对i截断时发生了“溢出”，即：53191截断为16位数时，有效数据丢失，无法被正确表示！

如何实现高级语言源程序中的运算？

- ◆ 总结：C语言程序中的基本数据类型及其基本运算类型
 - 基本数据类型
 - 无符号数、带符号整数、浮点数、位串、字符（串）
 - 基本运算类型
 - 算术、按位、逻辑、移位、扩展和截断
- ◆ 计算机如何实现高级语言程序中的运算？
 - 将各类表达式编译（转换为）指令序列
 - 计算机直接执行指令来完成运算

例：C语言赋值语句“**f = (g+h) - (i+j);**”中变量i、j、f、g、h由编译器分别分配给MIPS寄存器\$t0~\$t4。寄存器\$t0~\$t7对应8~15，上述程序段对应的MIPS机器代码和汇编表示（#后为注释）如下：

000000 01011 01100 01101 00000 100000 add \$t5, \$t3, \$t4 # g+h

000000 01000 01001 01110 00000 100000 add \$t6, \$t0, \$t1 # i+j

000000 01101 01110 01010 00000 100010 sub \$t2, \$t5, \$t6 # f=(g+h)-(i+j)

下面以MIPS为例看：指令中会提供哪些运算？能否完全支持高级语言需求？

MIPS定点算术运算指令

<i>Instruction</i>	<i>Example</i>	<i>Meaning</i>	<i>Comments</i>
add	add \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; exception possible
subtract	sub \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; exception possible
add immediate	addi \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; exception possible
add unsigned	addu \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; no exceptions
subtract unsigned	subu \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; no exceptions
add imm. unsign.	addiu \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; no exceptions
multiply	mult \$2,\$3	Hi, Lo = $\$2 \times \3	64-bit signed product
multiply unsigned	multu \$2,\$3	Hi, Lo = $\$2 \times \3	64-bit unsigned product
divide	div \$2,\$3	Lo = $\$2 \div \3 , Hi = $\$2 \bmod \3	Lo = quotient, Hi = remainder
divide unsigned	divu \$2,\$3	Lo = $\$2 \div \3 , Hi = $\$2 \bmod \3	Unsigned quotient & remainder

涉及到的操作数：32/16位 无符号数， 32/16位带符号数

涉及到的操作：加 / 减 / 乘 / 除（带符号数 / 无符号数）

MIPS 逻辑运算指令

and	and \$s1,\$s2,\$s3	$\$s1 = \$s2 \ \& \ \$s3$
or	or \$s1,\$s2,\$s3	$\$s1 = \$s2 \ \ \$s3$
nor	nor \$s1,\$s2,\$s3	$\$s1 = \sim (\$s2 \ \ \$s3)$
and immediate	andi \$s1,\$s2,100	$\$s1 = \$s2 \ \& \ 100$
or immediate	ori \$s1,\$s2,100	$\$s1 = \$s2 \ \ 100$
shift left logical	sll \$s1,\$s2,10	$\$s1 = \$s2 \ \ll \ 10$
shift right logical	srl \$s1,\$s2,10	$\$s1 = \$s2 \ \gg \ 10$

涉及到的操作数：32/16位 逻辑数（位串）

涉及到的操作：按位与 / 按位或 / 按位或非 / 左移 / 右移

MIPS定点比较和分支指令

branch on equal	beq \$s1,\$s2,25	if (\$s1 == \$s2) go to PC + 4 + 100
branch on not equal	bne \$s1,\$s2,25	if (\$s1 != \$s2) go to PC + 4 + 100
set on less than	slt \$s1,\$s2,\$s3	if (\$s2 < \$s3) \$s1 = 1; else \$s1 = 0
set less than immediate	slti \$s1,\$s2,100	if (\$s2 < 100) \$s1 = 1; else \$s1 = 0
set less than unsign	sltu \$s1,\$s2,\$s3	if (\$s2 < \$s3) \$s1 = 1; else \$s1 = 0
set less than immediate unsigned	sltiu \$s1,\$s2,100	if (\$s2 < 100) \$s1 = 1; else \$s1 = 0

涉及到的操作数：32/16位 无符号数， 32/16位带符号数

涉及到的操作：大小比较和相等比较（有符号 / 无符号）

通过减法运算实现“比较”操作！

MIPS定点数据传送指令

load word	lw \$s1,100(\$s2)	\$s1 = Memory[\$s2 + 100]
store word	sw \$s1,100(\$s2)	Memory[\$s2 + 100] = \$s1
load half unsigned	lhu \$s1,100(\$s2)	\$s1 = Memory[\$s2 + 100]
store half	sh \$s1,100(\$s2)	Memory[\$s2 + 100] = \$s1
load byte unsigned	lbu \$s1,100(\$s2)	\$s1 = Memory[\$s2 + 100]
store byte	sb \$s1,100(\$s2)	Memory[\$s2 + 100] = \$s1
load upper immediate	lui \$s1,100	\$s1 = 100 * 2 ¹⁶

涉及到的操作数： 32/16位带符号数（偏移量可以是负数）

涉及到的操作： 加 / 减 / 符号扩展 / 0扩展

MIPS中的浮点算术运算指令

FP add single	<code>add.s \$f2,\$f4,\$f6</code>	$\$f2 = \$f4 + \$f6$
FP subtract single	<code>sub.s \$f2,\$f4,\$f6</code>	$\$f2 = \$f4 - \$f6$
FP multiply single	<code>mul.s \$f2,\$f4,\$f6</code>	$\$f2 = \$f4 \times \$f6$
FP divide single	<code>div.s \$f2,\$f4,\$f6</code>	$\$f2 = \$f4 / \$f6$
FP add double	<code>add.d \$f2,\$f4,\$f6</code>	$\$f2 = \$f4 + \$f6$
FP subtract double	<code>sub.d \$f2,\$f4,\$f6</code>	$\$f2 = \$f4 - \$f6$
FP multiply double	<code>mul.d \$f2,\$f4,\$f6</code>	$\$f2 = \$f4 \times \$f6$
FP divide double	<code>div.d \$f2,\$f4,\$f6</code>	$\$f2 = \$f4 / \$f6$

MIPS提供专门的浮点数寄存器：

- 32个32位单精度浮点数寄存器：\$f0, \$f1,, \$f31
 - 连续两个寄存器（一偶一奇）存放一个双精度浮点数
- 涉及到的浮点操作数： 32位单精度 / 64位双精度浮点数
- 涉及到的浮点操作： 加 / 减 / 乘 / 除

MIPS中的浮点数传送指令

load word copr. 1	lwcl \$f1, 100(\$s2)	\$f1 = Memory[\$s2 + 100]
store word copr. 1	swcl \$f1, 100(\$s2)	Memory[\$s2 + 100] = \$f1

涉及到的浮点操作数： 32位单精度浮点数

涉及到的浮点操作： 传送操作（与定点传送一样）

还涉及到定点操作： 加 / 减（用于地址运算）

例：将两个浮点数从内存取出相加后再存到内存的指令序列为：

lwcl \$f1, x(\$s1)

lwcl \$f2, y(\$s2)

add.s \$f4, \$f1, \$f2

swlc \$f4, z(\$s3)

MIPS中的浮点数比较和分支指令

branch on FP true	bclt 25	if (cond == 1) go to PC + 4 + 100
branch on FP false	bclf 25	if (cond == 0) go to PC + 4 + 100
FP compare single (eq,ne,lt,le,gt,ge)	c.lt.s \$f2,\$f4	if (\$f2 < \$f4) cond = 1; else cond = 0
FP compare double (eq,ne,lt,le,gt,ge)	c.lt.d \$f2,\$f4	if (\$f2 < \$f4) cond = 1; else cond = 0

涉及到的浮点操作数： 32位单精度浮点数/64位双精度浮点数

涉及到的浮点操作： 比较操作（用 减法来实现比较）

还涉及到的定点操作： 加 / 减（用于地址运算）

有一个专门的浮点标志**cond**，无需在指令中明显给出**cond**

MIPS指令考察的结果

◆ 涉及到的操作数：

- 无符号整数、带符号整数
- 逻辑数（位串）
- 浮点数

完全能够支持高级语言
对运算的需求！！

◆ 涉及到的运算

- 定点数运算
 - 带符号整数运算：取负 / 符号扩展 / 加 / 减 / 乘 / 除 / 算术移位
 - 无符号整数运算：0扩展 / 加 / 减 / 乘 / 除
- 逻辑运算
 - 逻辑操作：与 / 或 / 非 / ...
 - 移位操作：逻辑左移 / 逻辑右移
- 浮点数运算：加、减、乘、除

实现MIPS定点和浮点运算指令的思路：

先实现一个能进行基本算术运算（加/减）和基本逻辑运算、并生成基本条件码（ZF/OF/CF/NF）的ALU，再由ALU和移位器实现乘、除、浮点运算器。

ALU是运算部件的核心！以下介绍ALU的实现。