



南京大學

NANJING UNIVERSITY

运算方法和运算部件

殷亚凤

智能软件与工程学院

苏州校区南雍楼东区225

yafeng@nju.edu.cn , <https://yafengnju.github.io/>



运算方法和运算部件

- 高级语言和机器指令中的运算
- 基本运算部件
- 定点数运算
- 整数乘除运算
- 浮点数运算





高级语言和机器指令中的运算

- **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

- 对一个**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：如何从一个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： $0x2C00$





高级语言和机器指令中的运算

- **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 “或” 运算
 - “&&” 表示 “与” 运算

- 例如，if ((x>y) && (i<100)) then

- “!” 表示 “非” 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



高级语言和机器指令中的运算

- **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

- 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。 (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。



高级语言和机器指令中的运算

• C语言程序中涉及的运算

• 位扩展和位截断运算

– 用途

- 在进行类型转换时，可能需要数据的扩展或截断

– 操作

- 没有专门的操作运算符，根据类型转换前后数据长短来确定是扩展还是截断
- “扩展”：短数转为长数；“截断”，长数转为短数

• 扩展

无符号数：0扩展，即：前面补0

带符号整数：符号扩展，即：前面补符号

• 截断

强行将一个长数的高位丢弃，故可能会发生“溢出”

例1：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;
unsigned short usi = si;
int i = si;
unsigned ui = usi;
```

```
si = -12345          CF C7
usi = 53191         CF C7
i = -12345         FF FF CF C7
ui = 53191         00 00 CF C7
```



高级语言和机器指令中的运算

• C语言程序中涉及的运算

• 位扩展和位截断运算

– 用途

- 在进行类型转换时，可能需要数据的扩展或截断

– 操作

- 没有专门的操作运算符，根据类型转换前后数据长短来确定是扩展还是截断
- “扩展”：短数转为长数；“截断”，长数转为短数

• 扩展

无符号数：0扩展，即：前面补0

带符号整数：符号扩展，即：前面补符号

• 截断

强行将一个长数的高位丢弃，故可能会发生“溢出”

例2：在大端机上执行后，
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，无法正确
表示！





高级语言和机器指令中的运算

MIPS指令中涉及的运算

表 3.1 MIPS 指令系统中涉及运算的部分指令

指令类型	指令名称	汇编形式举例	含义	所需运算
逻辑运算	and	and \$1, \$2, \$3	$\$1 = \$2 \& \$3$	按位与
	or	or \$1, \$2, \$3	$\$1 = \$2 \$3$	按位或
	nor	nor \$1, \$2, \$3	$\$1 = \sim(\$2 \$3)$	按位或非
	and immediate	andi \$1, \$2, 100	$\$1 = \$2 \& 100$	按位与
	or immediate	ori \$1, \$2, 100	$\$1 = \$2 100$	按位或
	shift left logical	sll \$1, \$2, 10	$\$1 = \$2 \ll 10$	逻辑左移
	shift right logical	srl \$1, \$2, 10	$\$1 = \$2 \gg 10$	逻辑右移

涉及到的操作数：32/16位 逻辑数

涉及到的操作：按位与 / 按位或 / 按位或非 / 左移 / 右移



高级语言和机器指令中的运算

MIPS指令中涉及的运算

定点 算术 运算*	shift right arithmetic	sra \$1, \$2, 10	$\$1 = \$2 \gg 10$	算术右移
	add	add \$1, \$2, \$3	$\$1 = \$2 + \$3$	整数加(判溢出)
	subtract	sub \$1, \$2, \$3	$\$1 = \$2 - \$3$	整数减(判溢出)
	add immediate	addi \$1, \$2, 100	$\$1 = \$2 + 100$	符号扩展、整数加(判溢出)
	sub immediate	subi \$1, \$2, 100	$\$1 = \$2 - 100$	符号扩展、整数减(判溢出)
	add unsigned	addu \$1, \$2, \$3	$\$1 = \$2 + \$3$	整数加(不判溢出)
	subtract unsigned	subu \$1, \$2, \$3	$\$1 = \$2 - \$3$	整数减(不判溢出)
	add immediate unsigned	addiu \$1, \$2, 100	$\$1 = \$2 + 100$	0 扩展、整数加(不判溢出)
	multiply	mult \$2, \$3	Hi, Lo = $\$2 \times \3	带符号整数乘
	multiply unsigned	multu \$2, \$3	Hi, Lo = $\$2 \times \3	无符号整数乘
	divide	div \$2, \$3	Lo = $\$2 \div \3 Hi = $\$2 \bmod \3	带符号整数除 Lo=商, Hi=余数
	divide unsigned	divu \$2, \$3	Lo = $\$2 \div \3 Hi = $\$2 \bmod \3	无符号整数除 Lo=商, Hi=余数

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 (有符号 / 无符号)。





高级语言和机器指令中的运算

• MIPS指令中涉及的运算

定点 数据 传送	load word	lw \$ 1,100(\$ 2)	$\$ 1 = \text{mem}[\$ 2 + 100]$	符号扩展并整数加
	store word	sw \$ 1,100(\$ 2)	$\text{mem}[\$ 2 + 100] = \$ 1$	符号扩展并整数加
	load half unsigned	lhu \$ 1,100(\$ 2)	$\$ 1 = \text{mem}[\$ 2 + 100]$	符号扩展并整数加,0 扩展
	store half	sh \$ 1,100(\$ 2)	$\text{mem}[\$ 2 + 100] = \$ 1$	符号扩展并整数加,符号扩展
	load byte unsigned	lbu \$ 1,100(\$ 2)	$\$ 1 = \text{mem}[\$ 2 + 100]$	符号扩展并整数加,0 扩展
	store byte	sb \$ 1,100(\$ 2)	$\text{mem}[\$ 2 + 100] = \$ 1$	符号扩展并整数加,符号扩展
	load upper immediate	lui \$ 1,100	$\$ 1 = 100 \times 2^{16}$	逻辑左移 16 位

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



高级语言和机器指令中的运算

MIPS指令中涉及的运算

指令类型	指令名称	汇编形式举例	含义	所需运算
浮点 算术 运算	FP add single	add.s \$f2, \$f4, \$f6	$\$f2 = \$f4 + \$f6$	单精度浮点加
	FP subtract single	sub.s \$f2, \$f4, \$f6	$\$f2 = \$f4 - \$f6$	单精度浮点减
	FP multiply single	mul.s \$f2, \$f4, \$f6	$\$f2 = \$f4 \times \$f6$	单精度浮点乘
	FP divide single	div.s \$f2, \$f4, \$f6	$\$f2 = \$f4 \div \$f6$	单精度浮点除
	FP add double	add.d \$f2, \$f4, \$f6	$\$f2 = \$f4 + \$f6$	双精度浮点加
	FP subtract double	sub.d \$f2, \$f4, \$f6	$\$f2 = \$f4 - \$f6$	双精度浮点减
	FP multiply double	mul.d \$f2, \$f4, \$f6	$\$f2 = \$f4 \times \$f6$	双精度浮点乘
	FP divide double	div.d \$f2, \$f4, \$f6	$\$f2 = \$f4 \div \$f6$	双精度浮点除

- 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： $\$f0, \$f1, \dots, \$f31$
 连续两个寄存器（一偶一奇）存放一个双精度浮点数



高级语言和机器指令中的运算

• MIPS指令中涉及的运算

浮点 数据 传送	load word corp.1 store word corp.1	lwcl \$ f1,100(\$ 2) swcl \$ f1,100(\$ 2)	\$ f1 = mem[\$ 2+100] mem[\$ 2+100] = \$ f1	符号扩展并整数加 符号扩展并整数加
----------------	---------------------------------------	--	--	----------------------

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）

涉及到定点操作：加 / 减（用于地址运算）

例：实现将两个浮点数从内存取出相加后再存回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)
```



高级语言和机器指令中的运算

• MIPS指令中涉及的运算

• 涉及到的操作数：

- 无符号整数、带符号整数
- 逻辑数
- 浮点数

• 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定点运算指令的思路：

首先实现一个能进行基本算术运算（加/减）和基本逻辑运算（与/或/或非）、并能生成基本条件码（ZF/VF/CF/NF）的**ALU**，再由ALU和移位器实现乘除运算器。



运算方法和运算部件

- 高级语言和机器指令中的运算
- **基本运算部件**
- 定点数运算
- 整数乘除运算
- 浮点数运算





基本运算部件

- 全加器**：输入为加数、被加数和低位进位 C_{in} ，输出为和 F 、进位 C_{out}

X_i	Y_i	C_{i-1}	F_i	C_i
A	B	C_{in}	F	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

真值表

$$F = \overline{A} \cdot \overline{B} \cdot C_{in} + \overline{A} \cdot B \cdot \overline{C_{in}} + A \cdot \overline{B} \cdot \overline{C_{in}} + A \cdot B \cdot C_{in}$$

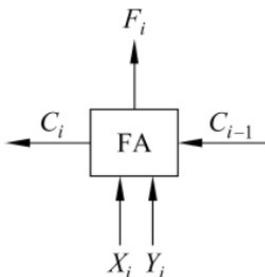
$$C_{out} = \overline{A} \cdot B \cdot C_{in} + A \cdot \overline{B} \cdot C_{in} + A \cdot B \cdot \overline{C_{in}} + A \cdot B \cdot C_{in}$$

化简后：

$$F = A \oplus B \oplus C_{in}$$

$$C_{out} = A \cdot B + A \cdot C_{in} + B \cdot C_{in}$$

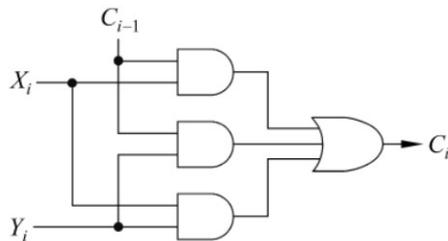
逻辑符号



全加和 F_i 的生成



全加进位 C_i 的生成





基本运算部件

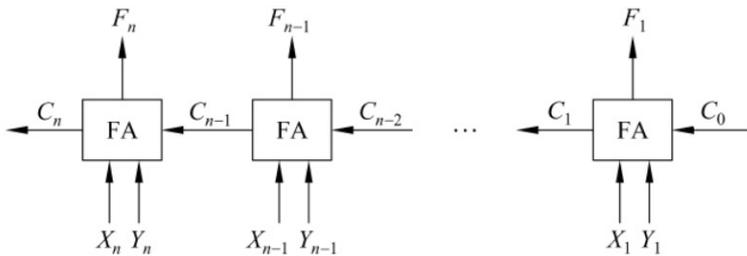
串行进位加法器

全加和、全加进位：

$$F = A \oplus B \oplus C_{in}$$

$$C_{out} = A \cdot B + A \cdot C_{in} + B \cdot C_{in}$$

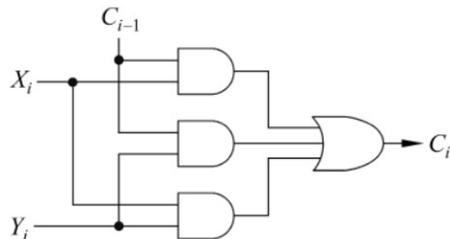
n位串行进位加法器



全加和Fi的生成



全加进位Ci的生成



- 求和Sum延迟为6ty；进位Carryout延迟为 2 ty
(假定一个与门/或门延迟为1ty，异或门的延迟则为 3 ty)

串行加法器的缺点：

进位按串行方式传递，速度慢！

问题：n位串行加法器从C0到Cn的延迟时间为多少？ **2n级门延迟！**

最后一位和数的延迟时间为多少？

2n+1级门延迟！



基本运算部件

- **并行进位加法器**

- **为什么用先行进位方式？**

串行进位加法器采用串行逐级传递进位，电路延迟与位数成正比关系。因此，现代计算机采用一种**先行进位 (Carry look ahead)方式**。

- **如何产生先行进位？**

定义辅助函数： $G_i = X_i Y_i \dots$ 进位生成函数

$P_i = X_i + Y_i \dots$ 进位传递函数

通常把实现上述逻辑的电路称为进位生成/传递部件

- **全加逻辑方程： $F_i = X_i \oplus Y_i \oplus C_i$ $C_{i+1} = X_i Y_i + (X_{i-1} + Y_{i-1}) C_{i-1} = G_i + P_i C_i$ ($i=0, 1, \dots, n$)**

设 $n=4$,则： $C_1 = G_0 + P_0 C_0$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$$

由上式可知：**各进位之间无等待**，相互独立并同时产生。通常把实现上述逻辑的电路称为4位先行进位部件（4位CLU）



基本运算部件

并行进位加法器

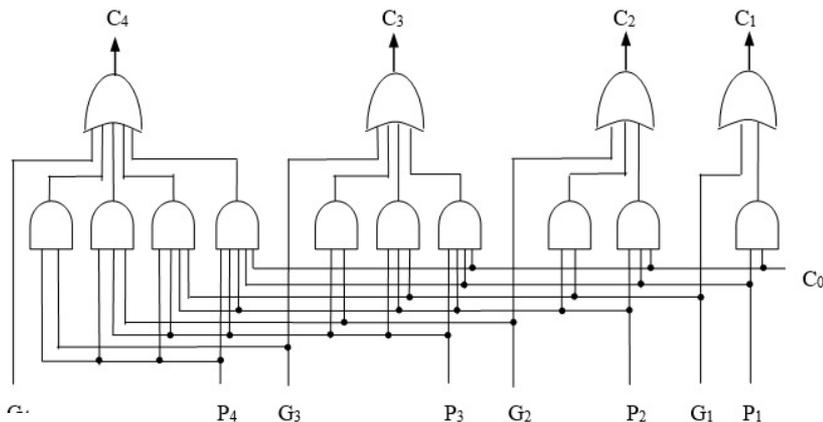
全先行进位加法器(CLA)：
所有进位独立并同时生成

$$C_1 = G_1 + P_1 C_0$$

$$C_2 = G_2 + P_2 C_1 = G_2 + P_2 G_1 + P_2 P_1 C_0$$

$$C_3 = G_3 + P_3 C_2 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_0$$

$$C_4 = G_4 + P_4 C_3 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_0$$

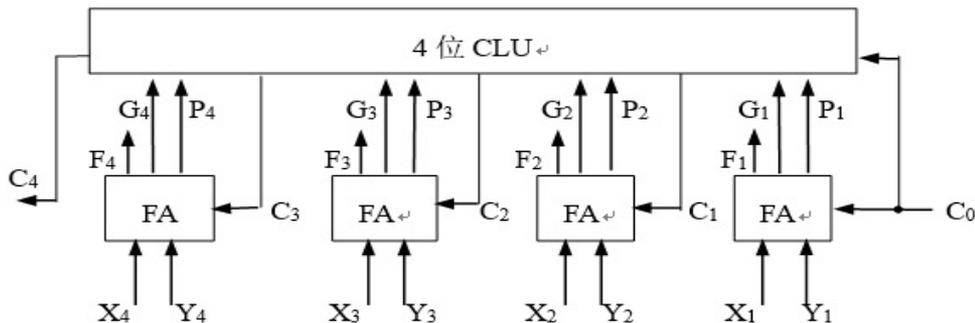


4位先行进位部件(CLU)

$$G_i = X_i Y_i$$

$$P_i = X_i + Y_i \quad (\text{或 } P_i = X_i \oplus Y_i)$$

$$F_i = X_i \oplus Y_i \oplus C_{i-1}$$



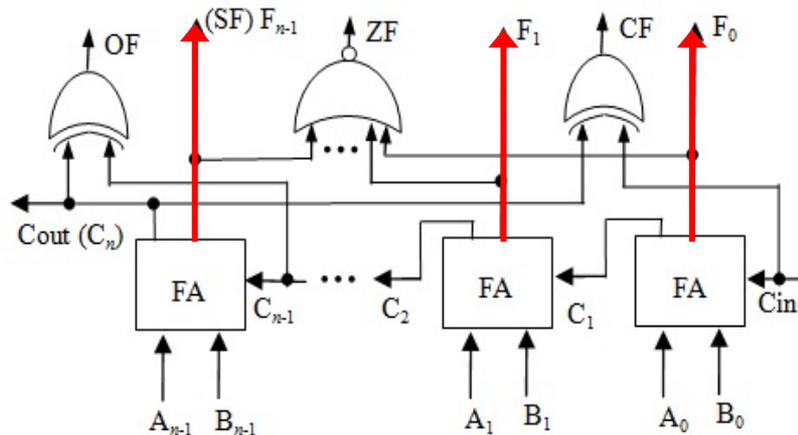
4位全先行进位加法器CLA



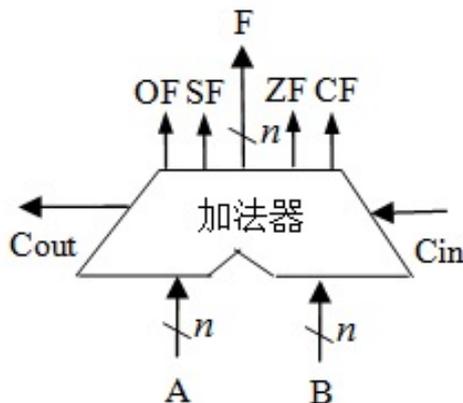
基本运算部件

带标志加法器

- n位加法器无法用于两个n位带符号整数（补码）相加，无法判断是否溢出
- 程序中经常需要比较大小，通过（在加法器中）做减法得到的标志信息来判断



带标志加法器的逻辑电路



带标志加法器符号

溢出标志OF : $OF = C_n \oplus C_{n-1}$

符号标志SF : $SF = F_{n-1}$

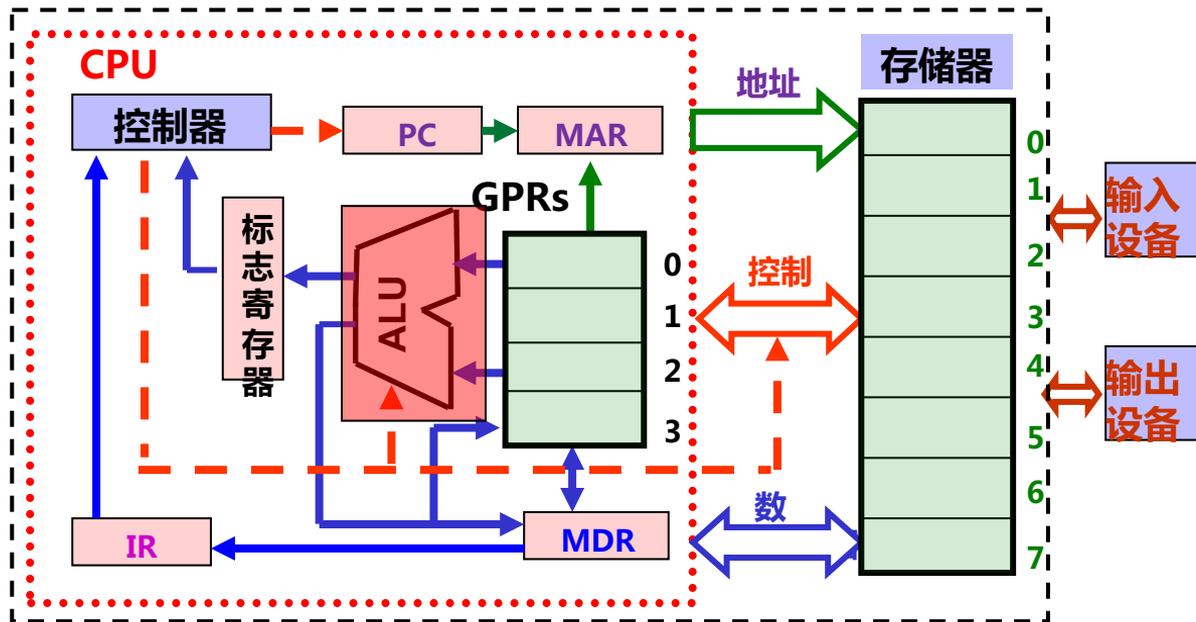
零标志ZF=1，当且仅当F=0；

进位/借位标志CF : $CF = C_{out} \oplus C_{in}$



基本运算部件

• 算术逻辑部件



CPU：中央处理器；
PC：程序计数器；
MAR：存储器地址寄存器
ALU：算术逻辑部件；
IR：指令寄存器；
MDR：存储器数据寄存器
GPRs：通用寄存器组
(由若干通用寄存器组成)



基本运算部件

• 算术逻辑部件

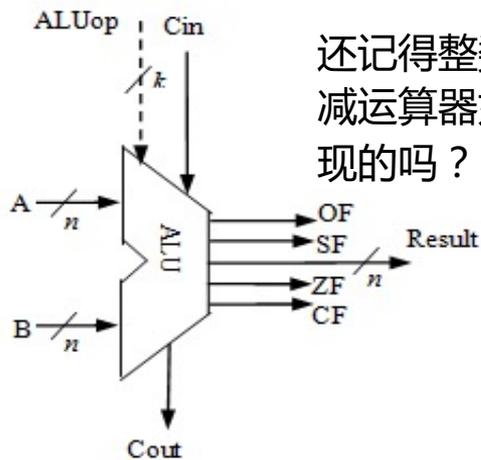
• 进行基本算术运算与逻辑运算

- 无符号整数加、减
- 带符号整数加、减
- 与、或、非、异或等逻辑运算

• 核心电路是整数加/减运算部件

• 输出除和/差等，还有标志信息

• 有一个操作控制端 (ALUop)，用来决定ALU所执行的处理功能。ALUop的位数k决定了操作的种类例如，当位数k为3时，ALU最多只有 $2^3=8$ 种操作。



还记得整数加/减运算器如何实现的吗？

ALU 符号

ALUop	Result	ALUop	Result	ALUop	Result	ALUop	Result
0 0 0	A加B	0 1 0	A与B	1 0 0	A取反	1 1 0	A
0 0 1	A减B	0 1 1	A或B	1 0 1	$A \oplus B$	1 1 1	未用

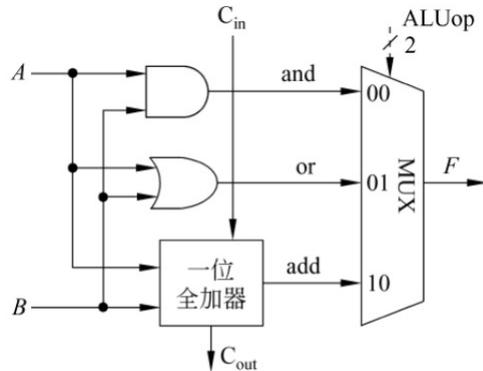


图 3.8 一位 ALU 结构



运算方法和运算部件

- 高级语言和机器指令中的运算
- 基本运算部件
- 定点数运算
- 整数乘除运算
- 浮点数运算





提问

Q & A

殷亚凤

智能软件与工程学院

苏州校区南雍楼东区225

yafeng@nju.edu.cn , <https://yafengnju.github.io/>



南京大學
NANJING UNIVERSITY