计算机组成与结构
—运算方法1

计算机科学与技术学院

温故 —关于数制表示

- 56的8421格式BCD码是?
- 0101 0110
- 简述ASCII码的特性?
- 7bit（存入1Byte），编码128个字符(32个控制字符+94个可打印符号)
- 只覆盖了英语字符
- 简述Unicode的特性?
- 全面性：覆盖了世界上几乎所有的字符系统
- 唯一性：给每一个字符一个唯一编码，不论语言、平台、程序
- 扩展性：设计了多种编码方案（如UTF-8、UTF-16、UTF-32)
- 兼容性：兼性现有编码系统，例如，它的前128个字符与ASCII码一致

温故 —关于数制表示

- 简单奇偶校验原理?
- 通过计算数据中“1”的个数是奇数还是偶数来判断数据的正确性
- “1110111”在奇/偶校验编码时，如何设置校验位?
- 奇校验： 11110111
- 偶校验： 01110111
- 简述海明码原理?
- 核心基于奇偶校验，将每位数据的分配在不同的校验组合中
- 海明码通过计算校验位数值来确定错误的位置。

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		...
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

- 当海明码校验位输出为0101时，如何解读?
- 第5位出错

几个小练习

- 2024的IEEE 754 32位浮点如何表示?
- 用奇 and 偶校验编码原码2024?
- 用海明码编码原码2024?
- 用CRC(10011)编码原码2024?

2024的IEEE 754 32位浮点如何表示?

- 1为符号+8为阶码+31尾数
  - 尾数用原码表示，但小数点前隐含一个1
  - 阶码有移码表示，但移码使用非常规的$+2^n-1$格式
    - 在常规的移码计算之后，额外-1

2024= 011111101000=+1.1111101×2^{10}

符号位0

阶码10+127=10001001

尾数11111010000000000000000

0100 0100 1111 1101 0000 0000 0000 0000

32位浮点数为：44FD0000

用奇 and 偶校验编码原码2024

2024= 0111 1110 1000

奇校验 0 0111 1110 1000

偶校验 1 0111 1110 1000

采用偶校验，用海明码编码原码2024

2024= 0111 1110 1000

0X1111110X100X0XX

0X1111110X100X0X0

0X1111110X100X010

0X1111110X1001010

0X111111001001010

00111111001001010

0 0111 1110 0100 1010=7E4A

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		…
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

循环冗余检验(CRC)

- 循环冗余校验：

  Cyclic Redundancy Check，简称 CRC。
- 通过某种数学运算建立数据和校验位之间的约定关系。
- 编码及译码：
  - 发送端：
    - 被校验数据除以生成多项式；
    - 被校验数据拼接余数，结果作为发送数据。
  - 接收端：接收数据除以生成多项式。
    - 可以除尽，编码正确；
    - 除不尽，余数指明出错位所在的位置。

循环冗余检验(CRC)

■ 采用模2算术运算：

- 通过模2减法实现模2除法；
- 以模2加法将所得余数拼接在被校验数据的后面，形成能除尽的被校验数据。

■ 生成多项式应满足的要求：

- 任何一位发生错误都应使余数不为0；
- 不同位发生错误应当使余数不同；
- 应满足余数循环规律。

■ 生成多项式的表示：

如，生成多项式G = 1011_2，表示生成多项式为

$G(X) = X^3 + X + 1$

循环冗余检验(CRC)

符号及约定：

- **被校验数据(被除数)为F(X)；**
- **约定的生成多项式(除数)为G(X)；**
  - **发送方和接收方使用同一个生成多项式G(X)**
  - **G(X)的首位和最后一位的系数必须为1**
- **所产生的余数为R(X)。**

循环冗余检验(CRC)

发送端，CRC的编码方法：

① 将被校验数据(共k位)的有效信息F(X)左移 r 位，得到

F(X)×X^r。

② 选取一个 r＋1 位的生成多项式G(X)，对F(X)×X^r作模2除法

：

$$F(X)\times X^r/G(X)=Q(X)+R(X)/G(X)$$

③ 将F(X)与R(X)相拼接。

$$F(X)\times X^r+R(X)=F(X)\times X^r-R(X)=Q(X)\times G(X)$$

拼接了校验码的数据必定能被约定的G(X)所除尽。

循环冗余检验(CRC)

接收端，CRC的译码方法：

将接收到的编码字除以约定的生成多项式**G(X)**：

- 余数为**0**，则传输没有错误。
- 余数不为**0**，则某一位出错。
  - 余数代码与出错位序号之间有唯一的对应关系：根据余数找到出错位；
  - 将出错位取反即可纠错。

循环冗余检验(CRC)

```
          1010101
      ___________
1011 ) 1001010000
       1011
       ----
        1001
        1011
        ----
          1000
          1011
          ----
           1100
           1011
           ----
            111
```

【例】

① 假设信息字节为

　F＝1001010_2；

② 选取G＝1011_2；

③ 将F左移l－1位，

　形成 F′＝1001010000_2；

④ 用 F′做被除数、G 做除数，

　进行模2除法。

　忽略商，余数为R＝111_2。

⑤ 把余数加到 F' 中，组成要发送的信息M：1001010000_2＋111_2＝1001010111_2。

⑥ 接收器采用相反的过程对接收的信息 M 进行解码和校验。M 应该可以被 G 严格整除。

循环冗余检验(CRC)

【例】

接收器：解码校验（正确的情况）

```
              1010101
      ┌──────────────
1011 )1001010111
      1011
      ────
       1001
       1011
       ────
        1001
        1011
        ────
         1011
         1011
         ────
         0000
```

接收器：解码校验（1位出错的情况）

```
              1010100
      ┌──────────────
1011 )1001011111
      1011
      ────
       1001
       1011
       ────
        1011
        1011
        ────
         0011
```

循环冗余检验(CRC)

(7，4)循环码编码、余数与出错位置的关系

$G(X)=1011_2$	编码举例1		编码举例2		余数	出错位置
	数据位 6543	校验位 210	数据位 6543	校验位 210		
正确	1001	110	1100	010	0 0 0	无
错误	1001	111	1100	011	0 0 1	0
	1001	100	1100	000	0 1 0	1
	1001	010	1100	110	1 0 0	2
	1000	110	1101	010	0 1 1	3
	1011	110	1110	010	1 1 0	4
	1101	110	1000	010	1 1 1	5
	0001	110	0100	010	1 0 1	6

循环冗余检验(CRC)

- **CRC**的生成多项式的阶数越高，误判的概率就越小。
- 常用的**4**个标准多项式：
  - **CRC-12**：

    $G(X)=X^{12}+X^{11}+X^{3}+X^{2}+X+1$
  - **CRC-16（ANSI）**：

    $G(X)=X^{16}+X^{15}+X^{2}+1$
  - **CRC-CCITT（ITU-T）**：

    $G(X)=X^{16}+X^{12}+X^{5}+1$
  - **CRC-32**：

    $G(X)=X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}$

    $+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$

几个小练习

- 520的IEEE 754浮点如何表示?
- 用奇 and 偶校验编码原码520?
- 用海明码编码原码520?
- 用CRC(10011)编码原码520?

用CRC(10011)编码原码2024?

2024= 111 1110 1000

10011

CRC编码结果:

111 1110 1000 0011

```
10011 | 111111010000000
        10011
        011001
         10011
         010100
          10011
          0011110
            10011
            011010
             10011
             010010
              10011
              000010000
                  10011
                  00011
```

本节学习要点

- 定点数（整数、纯小数）的四则运算
  - 加减法
- 算数逻辑部分
  - 单元电路
  - 算法逻辑单元ALU
  - 运算器的结构
- 定点数乘法
  - 原码乘法（一位/二位乘法）

定点数运算—加减法

定点数运算—加法减法

- **无符号加法很简单，只需注意进位…**

原码加法，6+7=13

	0 1 1 0	6
+	0 1 1 1	7
	1 1 0 1	13

原码加法，13+7=20

	1 1 0 1	13
+	0 1 1 1	7
	1 0 1 0 0	20

溢出

定点数运算—加减法

- 无符号加法很简单，只需注意进位…
- 减法比较麻烦（包括 加负数）

原码减法，13-7=6

0 1 1 0 1	13
- 0 0 1 1 1	7
0 0 1 1 0	6

借位机制麻烦

原码减法，-7+13=6

1 0 1 1 1	-7
+ 0 1 1 0 1	13
0 0 1 0 0	4

错误！

定点数运算—加减法

- 无符号加法很简单，只需注意进位…
- 减法比较麻烦（包括 加负数）
- 如何简化减法?
  - 对减数预处理
    - 方案一：反码减法

原码减法，13-7=6

	0	1	1	0	1	13
-	0	0	1	1	1	7
		0	1	1	0	6

反码减法，13-7=6

	0	1	1	0	1	13
+	1	1	0	0	0	8
	0	0	1	0	1	5
+					1	1
	0	0	1	1	0	6

-7的反码

反码结果需+1

定点数运算—加减法

- 无符号加法很简单，只需注意进位…
- 减法比较麻烦（包括 加负数）
- 如何简化减法?
  - 对减数预处理
    - 方案一：反码减法
    - 方案二：补码减法

原码减法，13-7=6

0 1 1 0 1	13
- 0 0 1 1 1	7
0 1 1 0	6

反码减法，13-7=6

0 1 1 0 1	13
+ 1 1 0 0 0	8
0 0 1 0 1	5
+ 1	1
0 0 1 1 0	6

补码减法，13-7=6

0 1 1 0 1	13
+ 1 1 0 0 1	9
0 0 1 1 0	6

-7的补码

定点数运算—加减法

- 无符号加法很简单，只需注意进位…
- 减法比较麻烦（包括 加负数）
- 如何简化减法?
  - 对减数预处理
    - 方案一：反码减法
    - 方案二：补码减法
- 补码加法统一加减法
  - [X + Y]补 = [X]补 + [Y]补
  - [X - Y]补 = [X]补 + [-Y]补

原码减法，13-7=6

0 1 1 0 1	13
- 0 0 1 1 1	7
0 1 1 0	6

反码减法，13-7=6

0 1 1 0 1	13
+ 1 1 0 0 0	8
0 0 1 0 1	5
+ 1	1
0 0 1 1 0	6

补码减法，13-7=6

0 1 1 0 1	13
+ 1 1 0 0 1	9
0 0 1 1 0	6

7的补码

定点数运算—加减法

- 补码加减法统一运算规则：
  - 都用补码表示
  - 减法变成加负数
  - 符号也参与运算
- 结果就是和/差的补码

- 例，用补码计算：
  - 63 + 35
  - -63 + (-35)
  - 63 - 35

- 解：

$[63]_{补} = 00111111$

$[-63]_{补} = 11000001$

$[35]_{补} = 00100011$

$[-35]_{补} = 11011101$

63 + 35

	0	0 1 1 1 1 1 1	63
+	0	0 1 0 0 0 1 1	35
	0	1 1 0 0 0 1 0	98

-63 + (-35)

	1	1 0 0 0 0 0 1	-63
+	1	1 0 1 1 1 0 1	-35
1	1	0 0 1 1 1 1 0	-98

63 - 35

	0	0 1 1 1 1 1 1	63
+	1	1 0 1 1 1 0 1	-35
1	0	0 0 1 1 1 0 0	28

这多出来的1怎么搞？

这多出来的1怎么搞？

定点数运算—加减法

- 除了补码，移码也能加减
- 运算规则：
  - 都用移码表示
  - 减法变为加负数
  - 符号也参与计算
  - 运算结果符号位取反
- 结果就是和/差的移码

- 例，用移码计算：
  - 63 + 35
  - -63 + (-35)
  - 63 - 35

- 解：

[63]移 = 10111111

[- 63]移 = 01000001

[35]移 = 10100011

[- 35]移 = 01011101

63 + 35

	1	0 1 1 1 1 1 1	63
+	1	0 1 0 0 0 1 1	35
	1 0	1 1 0 0 0 1 0	-30
+	0 1	0 0 0 0 0 0 0	
	1 1	1 1 0 0 0 1 0	98

-63 + (-35)

	0	1 0 0 0 0 0 1	-63
+	0	1 0 1 1 1 0 1	-35
	1	0 0 1 1 1 1 0	30
+	1	0 0 0 0 0 0 0	
	1 0	0 0 1 1 1 1 0	-98

63 - 35

	1	0 1 1 1 1 1 1	63
+	0	1 0 1 1 1 0 1	-35
	0	0 0 1 1 1 0 0	-100
+	1	0 0 0 0 0 0 0	
	1	0 0 1 1 1 0 0	28

定点数运算—加法减法

- **溢出导致两个问题：**
  - **数据部分是否越界？**
  - **符号如何确定？**
- **粗暴解决方案：**
  - **扩大数值位数**
- **实用解决方案：**
  - **判断是否溢出**

63 + 85

	0	0	1	1	1	1	1	1		63
+	0	1	0	1	0	1	0	1		85
	1	0	0	1	0	1	0	0		-101

63 + 67

	0	1	0	0	0	0	0	1		65
+	0	1	0	0	0	0	1	1		67
	1	0	0	0	0	1	0	0		-124

- **溢出判断方法：**
  - **双符号位（变形补码）**
  - **进位判别法**
  - **符号位与进位标志判别法**

定点数运算—加减法

- **双符号位法**
  - “**一个符号不够，两个来凑**”
  - **符号位** S **变成** S_2S_1
    - **正数**->00
    - **负数**->11
- **溢出判断条件：** $OF = S1 \oplus S2$

S_1S_2 = 00,Non-Overflow

S_1S_2 = 11,Non-Overflow

S_1S_2 = 10,Negative Overflow

S_1S_2 = 01,Positive Overflow

双符号位

65 + 67

0 0	1 0 0 0 0 0 1	65
+ 0 0	1 0 0 0 0 1 1	67
0 1	0 0 0 0 1 0 0	溢出

正溢出

63 + 85

0 0	0 1 1 1 1 1 1	63
+ 0 0	1 0 1 0 1 0 1	85
0 1	0 0 1 0 1 0 0	溢出

正溢出

-65 + -67

1 1	0 1 1 1 1 1 1	-65
+ 1 1	0 1 1 1 1 0 1	-67
1 0	0 1 1 1 1 0 0	溢出

负溢出

-65 + -85

1 1	0 1 1 1 1 1 1	-65
+ 1 1	0 1 0 1 0 1 1	-85
1 0	1 1 0 1 0 1 0	溢出

负溢出

65 + 16

0 0	1 0 0 0 0 0 1	65
+ 0 0	0 0 0 1 0 0 0	16
0 0	1 0 0 1 0 0 1	溢出

无溢出

-65 + -5

1 1	0 1 1 1 1 1 1	-65
+ 1 1	1 1 1 1 0 1 1	-5
1 1	0 1 1 1 0 1 0	无溢

无溢出

定点数运算—加减法

- 进位判别法
- “其实就是双符号位法的变形”
- 设C_{n-1}为高最高数值位向符号位的进位

C_n为符号位向更高位的进位

则溢出判断条件：

$$OF = C_{n-1} \oplus C_n$$

$C_{n-1}\ C_n = 00$,Non-Overflow

$C_{n-1}\ C_n = 11$,Non-Overflow

$C_{n-1}\ C_n = 10$,Negative Overflow

$C_{n-1}\ C_n = 01$,Positive Overflow

65 + 67

```
    0|1 0 0 0 0 0 1 | 65
+   0|1 0 0 0 0 1 1 | 67
-------------------------
  0 1|0 0 0 0 1 0 0 | 溢出
  0 1
```

C_n　C_{n-1}

63 + 85

```
    0|0 1 1 1 1 1 1 | 63
+   0|1 0 1 0 1 0 1 | 85
-------------------------
  0 1|0 0 1 0 1 0 0 | 溢出
  0 1
```

正溢出

-65 + -67

```
  1 1|0 1 1 1 1 1 1 | -65
+ 1 1|0 1 1 1 1 0 1 | -67
-------------------------
  1 0|0 1 1 1 1 0 0 | 溢出
  1 0
```

负溢出

-65 + -85

```
  1 1|0 1 1 1 1 1 1 | -65
+ 1 1|0 1 0 1 0 1 1 | -85
-------------------------
  1 0|1 1 0 1 0 1 0 | 溢出
  1 0
```

负溢出

65 + 16

```
  0 0|1 0 0 0 0 0 1 | 65
+ 0 0|0 0 0 1 0 0 0 | 16
-------------------------
  0 0|1 0 0 1 0 0 1 | 溢出
  0 0
```

无溢出

-65 + -5

```
  1 1|0 1 1 1 1 1 1 | -65
+ 1 1|1 1 1 1 0 1 1 | -5
-------------------------
  1 1|0 1 1 1 0 1 0 | 无溢
  1 1
```

无溢出

定点数运算—加减法

- 符号位与进位标志判别法
  - CPU的处理器状态字（Processor State Word, PSW）描述众多运算状态，如溢出标志(Overflow Flag, OF)，进位标志(Carrier Flag, CF), 符号标志(Sign Flag)等…
  - 当运算发生进位时，CF置1
  - “把CF，SF纳入计算”
  - 溢出判断条件：$OF = CF \oplus SF$

CFSF = 00,Non-Overflow

CFSF = 11,Non-Overflow

CFSF = 10,Negative Overflow

CFSF = 01,Positive Overflow

CPU符号标志SF

CPU进位标志CF

65 + 67

	0	1 0 0 0 0 0 1	65
+	0	1 0 0 0 0 1 1	67
0 1		0 0 0 0 1 0 0	溢出

65 + 85

	0	0 1 1 1 1 1 1	63
+	0	1 0 1 0 1 0 1	85
0 1		0 0 1 0 1 0 0	溢出

正溢出

-65 + -67

	1	0 1 1 1 1 1 1	-65
+	1	0 1 1 1 1 0 1	-67
1 0		0 1 1 1 1 0 0	溢出

-65 + -85

	1	0 1 1 1 1 1 1	-65
+	1	0 1 0 1 0 1 1	-85
1 0		1 1 0 1 0 1 0	溢出

负溢出

65 + 16

	0	1 0 0 0 0 0 1	65
+	0	0 0 0 1 0 0 0	16
0 0		1 0 0 1 0 0 1	81

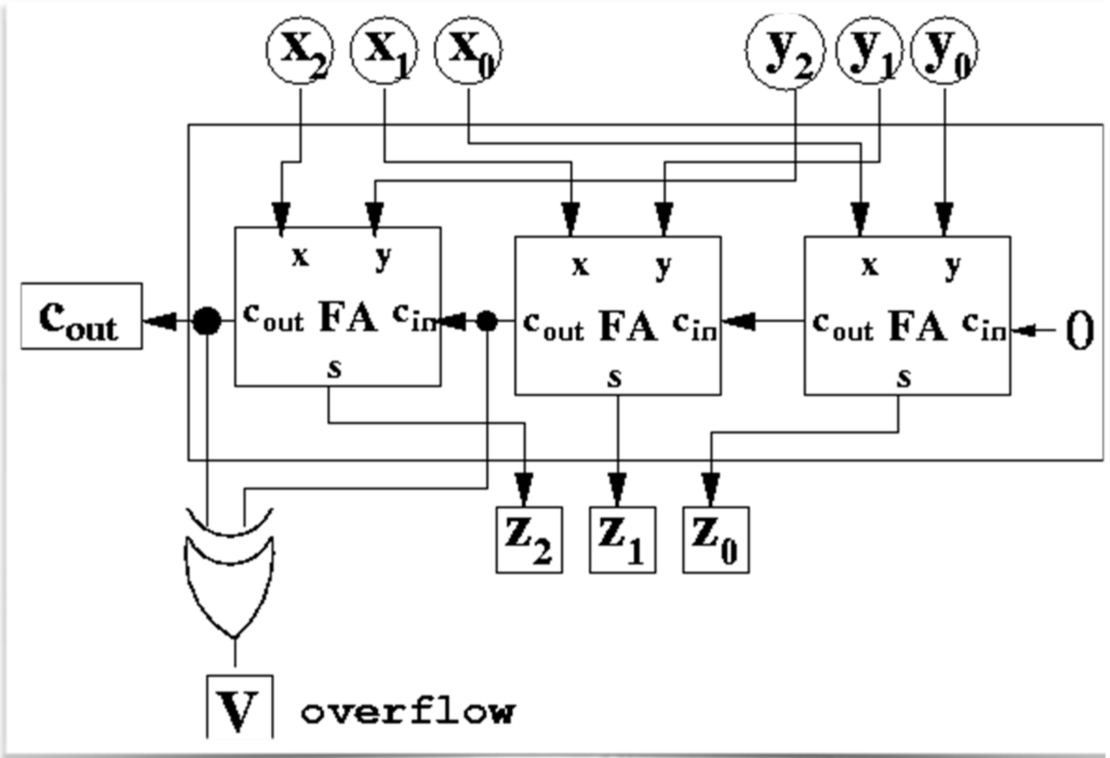
-65 - 5

	1	0 1 1 1 1 1 1	-65
+	1	1 1 1 1 0 1 1	-5
1 1		0 1 1 1 0 1 0	-70

无溢出

定点数运算—加减法

- **那么CPU实际是怎么做的?**
  - **SF和CF输入组合逻辑电路判断出来的~**
- **Then ?**
  - **溢出标志OF = On**
  - **[maybe]产生溢出中断**



3个全加器级联

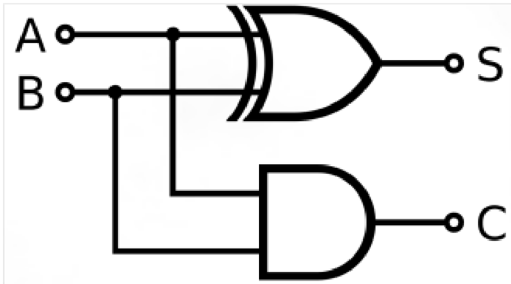
加法器逻辑电路

加法器逻辑电路

- **加法器**
  - **一位加法器**
    - **半加器(Half Adder)**
    - **全加器(Full Adder)**
  - **多位加法器**
    - **行波加法器（RCA）**
    - **先行进位加法器（CLA）**
  - **BCD加法器**

加法器逻辑电路

- **半加器(Half Adder)**
  - **没有进位输入(Carry In)的加法器**
  - **S_i为和，C_{i+1}为进位**
  - **电路逻辑为：**
    - $S_i = A \oplus B$
    - $C_{i+1} = A \cdot B$



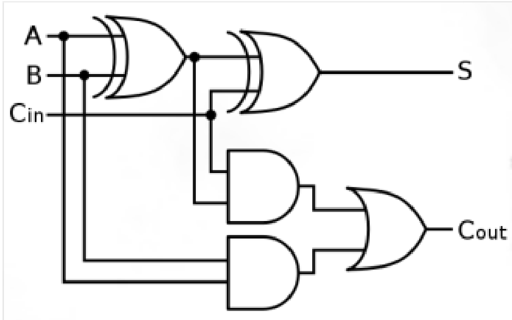
一位半加器

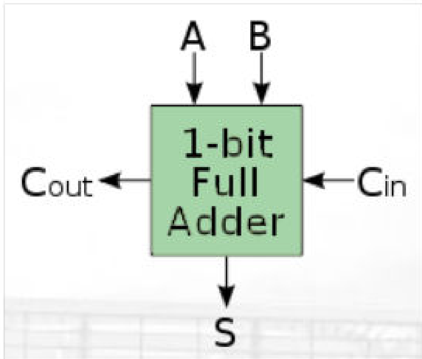
Inputs		Outputs	
A	B	C	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

半加器真值表

加法器逻辑电路

- **全加器(Full Adder)**
  - **包含进位输入(Carry In)的加法器**
  - **S_i为和**
  - **C_{in}为进位输入，C_{out}为进位输出**
  - **电路逻辑为：**
    - $S_i = A \oplus B \oplus C$
    - $C_{out} = A \cdot B + (A \oplus B) \cdot C_{in}$





一位全加器

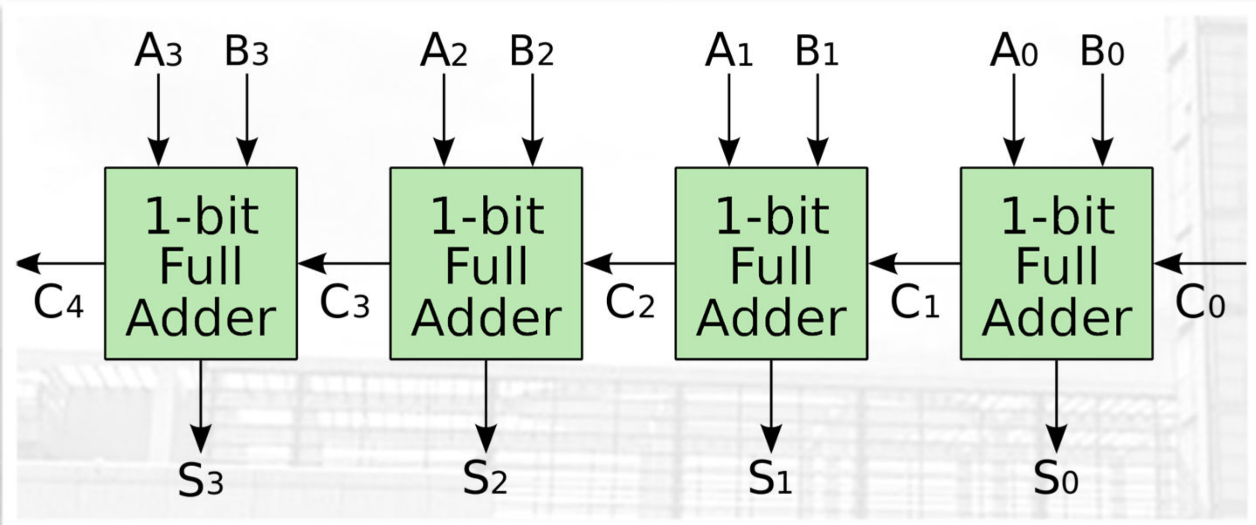
Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

全加器真值表

加法器逻辑电路

行波多位加法器(Ripple-Carry Adder, RCA)

- **自低位向高位，依次计算、进位，计算、进位…**
  - **so-called “ripple carry”**
- **RCA结构简单，但有明显缺点：计算时延大，**$n\Delta t$



RCA多位加法器

加法器逻辑电路

先行进位加法器(Carry-Lookahead Adder, CLA)

回顾Let一下1位全加器的进位逻辑C_{i+1}

$$C_{i+1} = A \cdot B + (A \oplus B) \cdot C_i$$

令：$P_i = A \oplus B$

$G_i = A \cdot B$

则：$C_{i+1} = G_i + P_i \cdot C_i$

$$C_1 = G_0 + P_0 C_{in}$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

C_2与C_1无关

C_3与C_2无关

C_4与C_3无关

所有进位C可以根据P序列和G序列直接求得

加法器逻辑电路

先行进位加法器(Carry-Lookahead Adder, CLA)

回顾Let一下1位全加器的进位逻辑C_{i+1}

$C_{i+1} = A \cdot B + (A \oplus B) \cdot C_i$

令：$P_i = A \oplus B$

$G_i = A \cdot B$

则：$C_{i+1} = G_i + P_i \cdot C_i$

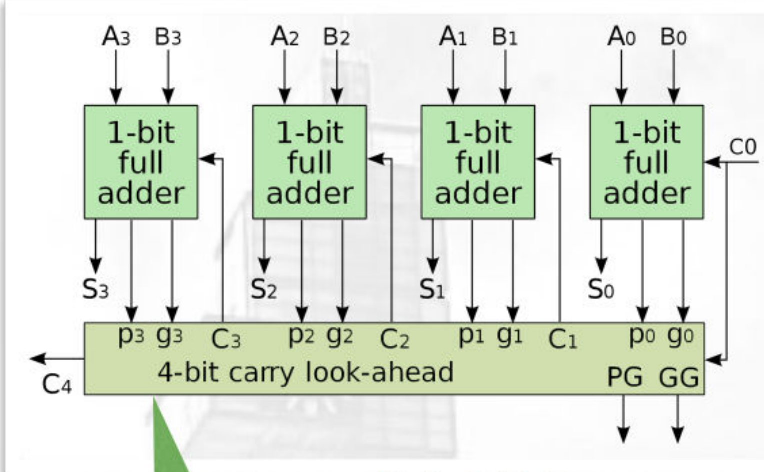
$C_1 = G_0 + P_0C_{in}$

$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1P_0C_{in}$

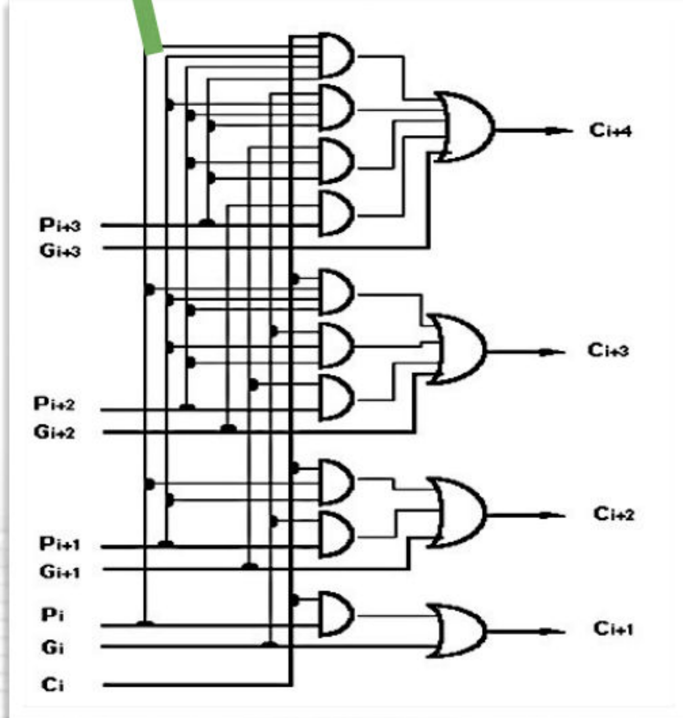
$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C$

$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C$

所有进位C可以根据P序列和G序列直接求得



CLA 4位加法器



4bit 进位生成单元

加法器逻辑电路

BCD加法器

16	0001 0110
+05	+0000 0101
	0001 1011
21	+0000 0110
	0010 0001

进位

无进位

产生进位

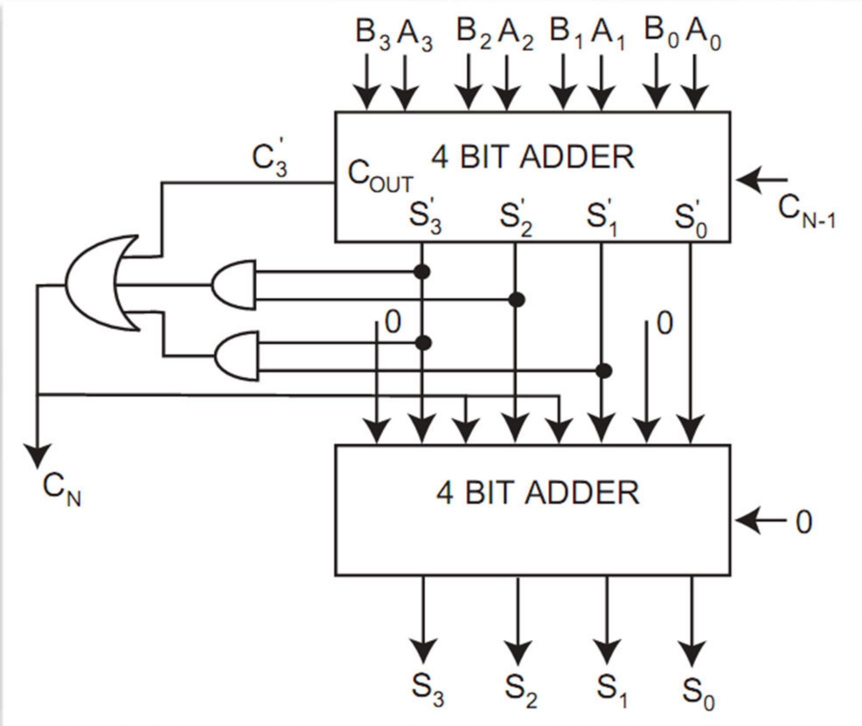
21

10进制有进位	10D	11D	12D	13D	14D	15D
16进制无进位	0AH	0BH	0CH	0DH	0EH	0FH
+06H	06H	06H	06H	06H	06H	06H
16进制有进位	10H	11H	12H	13H	14H	15H

对应

加法器逻辑电路

- **BCD加法器**
  - **两个4bit加法器构成**
    - **一个加法器算数值**
    - **另一个加法器用来+06H ….**



4bit BCD加法器

加法器逻辑电路

现代的CPU用啥加法器?

- **五花八门…**
  - **精确加法器**
    - **CLA / RCA**
    - **Kogge-Stone / Brent-Kung / Han-Carlson (CLA的高级形态)**
  - **非精确加法器 Speculative Adder**
    - **Insight: “很少有很长的进位链，所以赌某段无进位，…”**
    - **95%正确，比最快的精确加速器快平均1.5x**
    - **有额外电路快速检查结果是否正确（只能判断正确)**
- **取舍因素? 设计过程 / 性能 / 面积 / 延时 / 功耗 ….**
- **区区一个加法器，直到2020年，还在不断地发展…**

定点数运算—乘法

定点数运算—乘法

原码一位乘法就是模拟手工乘法:

- **符号**$S_m = S_A \oplus S_b$, **数值**$|m| = |A| \times |B|$

例，[X]原=0.1101, [Y]原=1.1011，求积

解： 符号，$S = 0 \oplus 1 = 1$；**数值部分如下:**

模拟手工乘法

$$
\begin{array}{r}
1\ 1\ 0\ 1 \\
\times \quad 1\ 0\ 1\ 1 \\
\hline
\left.\begin{array}{r}
1\ 1\ 0\ 1 \\
1\ 1\ 0\ 1\ \ \ \\
0\ 0\ 0\ 0\ \ \ \ \ \ \\
1\ 1\ 0\ 1\ \ \ \ \ \ \ \ \
\end{array}\right\} \text{如何描述这一过程?} \\
\hline
1\ 0\ 0\ 0\ 1\ 1\ 1\ 1
\end{array}
$$

定点数运算—乘法

原码一位乘法就是模拟手工乘法:

- 符号$S_m = S_A \oplus S_b$, 数值$|m| = |A| \times |B|$

例，[X]原=0.1101, [Y]原=1.1011，求积

解：符号，$S = 0 \oplus 1 = 1$；数值部分如下：

模拟手工乘法

$$
\begin{array}{cccccccc}
 & & & & 1 & 1 & 0 & 1 \\
\times & & & & 1 & 0 & 1 & 1 \\
\hline
 & & & & 1 & 1 & 0 & 1 \\
 & & & 1 & 1 & 0 & 1 & \\
 & & 0 & 0 & 0 & 0 & & \\
 & 1 & 1 & 0 & 1 & & & \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
$$

如何描述这一过程?

1. 设一临时变量$D = 0$
2. 若Y末位为1，$D = D + X$
3. D右移一位；Y右移一位
4. 若Y有剩余位数，回到第2步
5. 结束

定点数运算—乘法

原码一位乘法就是模拟手工乘法:

- 符号$S_m = S_A \oplus S_b$, 数值$|m| = |A| \times |B|$

例，[X]原=0.1101, [Y]原=1.1011，求积

解：符号，$S = 0 \oplus 1 = 1$；数值部分如下:

模拟手工乘法

```
        1 1 0 1
X       1 0 1 1
---------------
        1 1 0 1
      1 1 0 1
    0 0 0 0
  1 1 0 1
---------------
1 0 0 0 1 1 1 1
```

1. 设一临时变量$D = 0$
2. 若Y末位为1，$D = D + X$
3. D右移一位；Y右移一位
4. 若Y有剩余位数，回到第2步
5. 结束

Y

	D				A			A_0	操作
0	0	0	0	0	1	0	1	1	A_0=1, +X
+0	1	1	0	1					
0	1	1	0	1					
0	0	1	1	0	1	1	0	1	→右移一次
+0	1	1	0	1					A_0=1, +X
1	0	0	1	1	1	1	0	1	
0	1	0	0	1	1	1	1	0	→右移一次
0	0	0	0	0					A_0=0, +0
0	1	0	0	1	1	1	1	0	
0	0	1	0	0	1	1	1	1	→右移一次
+0	1	1	0	1					A_0=1, +X
1	0	0	0	1	1	1	1	1	
0	1	0	0	0	1	1	1	1	→右移一次

原码一位乘法运算过程

最终结果，[X]原x[Y]原=1.10001111

定点数运算—乘法

练习，[X]原=0.110111, [Y]原=1.1001，求积 1.01 1110 1111

1. 设一临时变量$D = 0$
2. 若Y末位为1，$D = D + X$
3. D右移一位；Y右移一位
4. 若Y有剩余位数，回到第2步
5. 结束

	D						Y				操作
0	0	0	0	0	0	0	1	0	0	1	A_0=1，+X
+0	1	1	0	1	1	1					
0	1	1	0	1	1	1					
0	0	1	1	0	1	1	1	1	0	0	右移一次
0	0	0	1	1	0	1	1	1	1	0	右移一次
0	0	0	0	1	1	0	1	1	1	1	右移一次 A_0=1，+X
+0	1	1	0	1	1	1					
0	1	1	1	1	0	1	1	1	1	1	
0	0	1	1	1	1	0	1	1	1	1	右移一次

定点数运算—乘法

原码二位乘法，

一次乘两位数字？这个在十进制下很反人类啊…，但在二进制下，似乎还好…

$$
\begin{array}{rl}
1\ 1\ 0\ 1 & \\
\times \qquad 1\ 0\ 0\ 1 & \\
\hline
1\ 1\ 0\ 1 & +|X| \\
1\ 1\ 0\ 1\ 0 & +2|X| \\
\hline
0\ 1\ 1\ 1\ 0\ 1\ 0\ 1 &
\end{array}
\qquad
\begin{array}{rl}
1\ 1\ 0\ 1 & \\
\times \qquad 1\ 1\ 0\ 0 & \\
\hline
0\ 0\ 0\ 0 & +0 \\
1\ 0\ 0\ 1\ 1\ 1 & +3|X| \\
\hline
1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 &
\end{array}
$$

规则似乎很直观：

$$
\begin{cases}
Y_{i+1}Yi = 00, +0 \\
Y_{i+1}Yi = 01, +|X| \\
Y_{i+1}Yi = 10, +2|X| \\
Y_{i+1}Yi = 11, +3|X|
\end{cases}
$$

定点数运算—乘法

原码二位乘法，

一次乘两位数字？这个在十进制下很反人类啊…，但在二进制下，似乎还好…

```
           1 1 0 1
×          1 0 0 1
------------------
           1 1 0 1   ← +|X|
       1 1 0 1 0     ← +2|X|
------------------
   0 1 1 1 0 1 0 1
```

```
           1 1 0 1
×          1 1 0 0
------------------
           0 0 0 0   ← +0
   1 0 0 1 1 1       ← +3|X|
------------------
   1 0 0 1 1 1 0 0
```

规则似乎很直观：

$$
\begin{cases}
Y_{i+1}Yi = 00, +0 \\
Y_{i+1}Yi = 01, +|X| \\
Y_{i+1}Yi = 10, +2|X| \\
Y_{i+1}Yi = 11, +3|X|
\end{cases}
$$

规则还可以更进一步优化：

$$+3|X| = +4|X| - |X|$$

⇒先减$|X|$，平移2步后+ $|X|$

```
               0 0 1 1 0 1
×              0 0 1 1 0 0
--------------------------
               0 0 0 0 0 0
+4X        1 1 0 0 1 1       ← −|X|
       0 0 1 1 0 1           ← +|X|
--------------------------
   0 1 1 0 0 1 1 1 0 0
```

定点数运算—乘法

原码二位乘法

规则还可以更进一步优化：

$+3|X| \quad = +4|X| - |X|$

$\Rightarrow$**先减**$|X|$**，平移2步后**$+\ |X|$

$\Rightarrow$

标记是否要$+4|X|$

Y_{i+1}	Y_i	C	操 作
0	0	0	+0，右移2次，C=0
0	0	1	+|X|，右移2次，C=0
0	1	0	+|X|，右移2次，C=0
0	1	1	+2|X|，右移2次，C=0
1	0	0	+2|X|，右移2次，C=0
1	0	1	−|X|，右移2次，C=1
1	1	0	−|X|，右移2次，C=1
1	1	1	+0，右移2次，C=1

定点数运算—乘法

原码二位乘法

标记是否要$+4|X|$

Y_{i+1}	Y_i	C	操　作
0	0	0	+0，　右移2次，C＝0
0	0	1	+|X|，　右移2次，C＝0
0	1	0	+|X|，　右移2次，C＝0
0	1	1	+2|X|，　右移2次，C＝0
1	0	0	+2|X|，　右移2次，C＝0
1	0	1	－|X|，　右移2次，C＝1
1	1	0	－|X|，　右移2次，C＝1
1	1	1	+0，　右移2次，C＝1

例，[X]原=0.1001111, [Y]原=1.100111，求积

解：[2X]补=01.001110; [-X]补=1.011001

符号位	D	A	操作
0 0 0	0 0 0 0 0 0	1 0 0 1 1 1	C=0
1 1 1	0 1 1 0 0 1		−X
1 1 1	0 1 1 0 0 1		C=1
1 1 1	1 1 0 1 1 0	0 1 1 0 0 1	→右移二次
0 0 1	0 0 1 1 1 0		C=1,+2X
0 0 1	0 0 0 1 0 0		C=0
0 0 0	0 1 0 0 0 1	0 0 0 1 1 0	→右移二次
0 0 1	0 0 1 1 1 0		C=0,+2X
0 0 1	0 1 1 1 1 1		C=0
0 0 0	0 1 0 1 1 1	1 1 0 0 0 1	→右移二次

图 3-9　例 3-12 的二位乘法的运算过程

最终结果，[X]原x[Y]原=1.010111110001

定点数运算—乘法

练习，[X]原=0.110110, [Y]原=1.101101，求积

解：[2X]补=01.101100; [-X]补=1.001010