计算机组成与结构
—运算方法习题课

李瑞　副教授
蒋志平　讲师
计算机科学与技术学院

题3.2 / 3.3 / 3.9

说明单符号号位检测溢出的方法?

说明双符号位检测举出的方法?

当两个同符号的数相加或异符号的数相减，结果的SF和CF进行啥运算为1时，表示运算的结果产生溢出?

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}为最高数值位向符号位的进位，C_n是符号位向更高位的进位，则溢出判断条件：

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

$$\begin{cases} C_{n-1}C_n = 00, \textbf{Non-Overflow} \\ C_{n-1}C_n = 00, \textbf{Non-Overflow} \\ C_{n-1}C_n = 10, \textbf{Negative Overflow} \\ C_{n-1}C_n = 01, \textbf{Positive Overflow} \end{cases}$$

向更高进位

65 + 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	溢出

0 1

65 + 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	溢出

0 1

正溢出

-65 + -67

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

1 0

-65 + -85

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

1 0

负溢出

65 + 16

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

0 0

-65 - 5

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

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：$OF = CF \oplus SF$

$$\begin{cases} CFSF = 00,\textbf{Non-Overflow} \\ CFSF = 00,\textbf{Non-Overflow} \\ CFSF = 10,\textbf{Negative Overflow} \\ CFSF = 01,\textbf{Positive Overflow} \end{cases}$$

CPU符号标志SF

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

CPU进位标志CF

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

- 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：$OF = S_1 \oplus S_2$

$$\begin{cases} S_1S_2 = 00,\textbf{Non-Overflow} \\ S_1S_2 = 11,\textbf{Non-Overflow} \\ S_1S_2 = 10,\textbf{Negative Overflow} \\ S_1S_2 = 01,\textbf{Positive Overflow} \end{cases}$$

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

65 + 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

-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	81

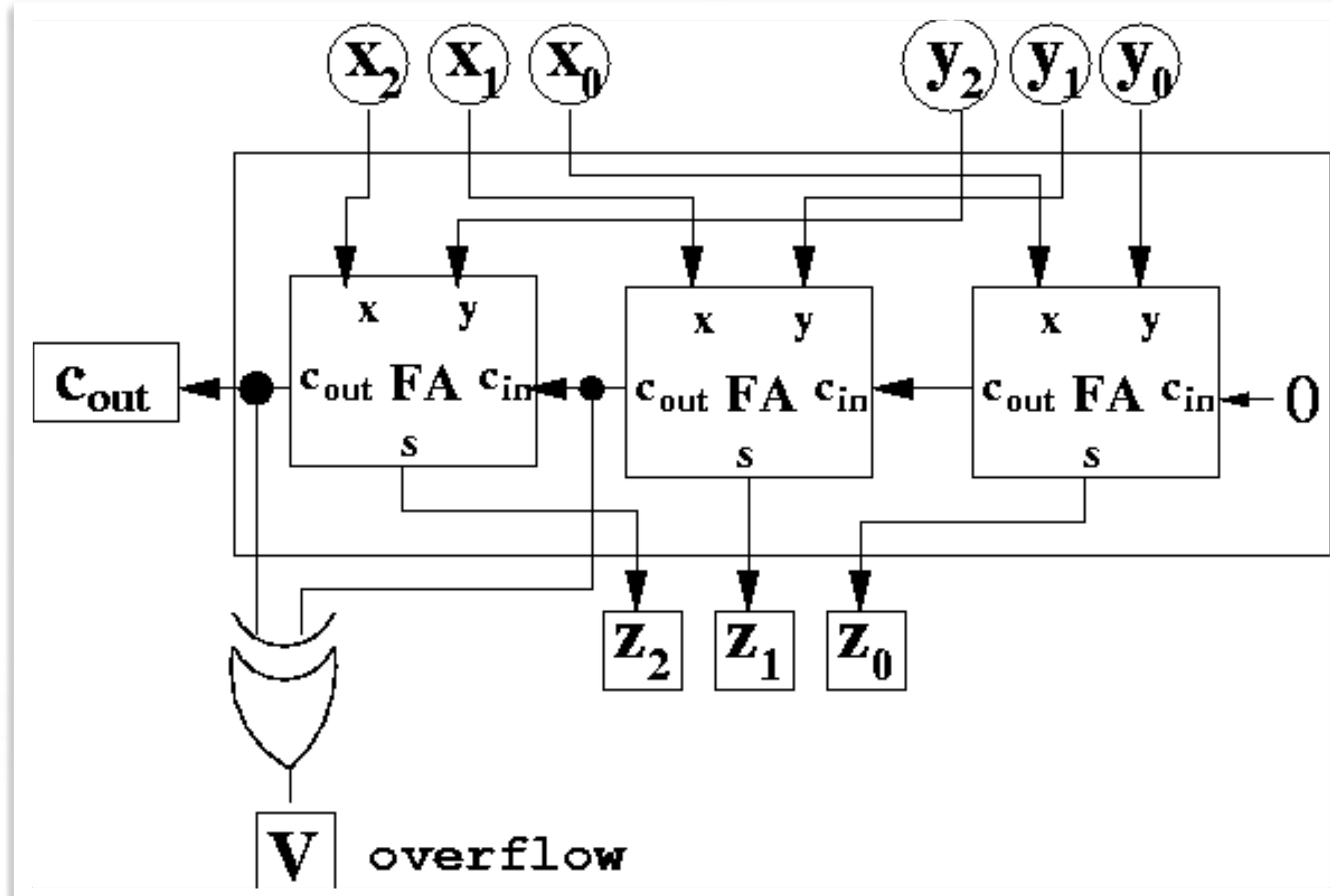
-65 - 5

	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	-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

题3.17

计算下面的加法结果，并判断溢出

1. x=0.01001　y = -0.10111
2. x=0.10010　y = 0.11000
3. x=-0.01101　y = 0.00101
4. x=-0.11011　y = -0.10010

题3.20

分别用原码1位乘法和booth法计算下面乘法

1. $x=-0.1101 \quad y = 0.0110$
2. $x=-0.1110 \quad y = -0.1101$

“表格式计算”—乘法

- 以原码一位乘法为例：竖式是直观的，步骤步骤累加是可理解的，表格式是一眼看上去懵逼的...

通过部分积D累加逐层结果

第一层 加1101	0 0 0 0	<-部分积D，初值为0
	+ 1 1 0 1	D = D + X
	1 1 0 1	
第二层 加1101	0 1 1 0 1	D右移
	+ 1 1 0 1 0	D = D>>1 + X
	1 0 0 1 1 1	
第三层 加0000	1 0 0 1 1 1	D右移
	+ 0 0 0 0 0 0	D = D>>1 + X
	1 0 0 1 1 1	
第四层 加1101	0 1 0 0 1 1 1	D右移
	+ 1 1 0 1 0 0 0	D = D>>1 + X
	1 0 0 0 1 1 1 1	

<<是左移操作， >>是右移操作

Y总是顶格对齐，D总右移，等效于竖式中X左移

2位符号，防止溢出时右移出现负数

部分积D的符号
（建议至少2位）

部分积D

粗线分隔D和Y

乘数Y

符号		D				Y		Y_0	操作说明
0	0	0	0	0	0	1	0	1	1	D/Y初始化
0	0	1	1	0	1	1	0	1	1	+[X]原
0	0	1	1	0	1	1	0	1	1	
0	0	0	1	1	0	1	1	0	1	D/Y右移1位
0	0	1	1	0	1	0	1	0	1	+[X]原
0	1	0	0	1	1	1	1	0	1	
0	0	1	0	0	1	1	1	1	0	D/Y右移1位
0	0	0	0	0	0			1	0	+0
0	0	1	0	0	1	1	1	1	0	
0	0	0	1	0	0	1	1	1	1	D/Y右移1位
0	0	1	1	0	1				1	+[X]原
0	1	0	0	0	1	1	1	1	1	
0	0	1	0	0	0	1	1	1	1	D/Y右移1位

乘数用尽，计算结束

Booth乘法（补码乘法）

```
          0 0 1 1 0 1
×         0 0 1 1 1 0
--------------------
      0 0 0 1 1 0 1
      0 0 1 1 0 1
    0 0 1 1 0 1
--------------------
0 0 0 1 0 1 1 0 1 1 0
```

<—略用心机需要3步

v.s

运用技巧只需2步—>

在此处减 $|X|$

在此处加 $|X|$

```
          0 0 1 1 0 1
×         0 0 1 1 1 0
--------------------
  1 1 1 1 1 0 0 1 1
  0 0 1 1 0 1
--------------------
  0 0 1 0 1 1 0 1 1 0
```

问题来了：如何高效地识别“一串1”的开始和结束？

Booth算法：

- 从右向左，遇到“1 0”就是“一串1”的开始，在1的位置减 $|X|$；
- 从右向左，遇到“0 1”就是“一串1”的结束，在0的位置加 $|X|$；
- 中间的“00”和“11”都不管！

y_i	y_{i-1}	操 作
0	0	部分积＋0，右移1位
0	1	部分积＋$[X]_补$，右移1
1	0	部分积＋$[-X]_补$，右移1
1	1	部分积＋0，右移1位

Booth乘法（补码乘法）

Booth算法：

- 从右向左，遇到“1 0”就是“一串1”的开始，在1的位置减 $|X|$；
- 从右向左，遇到“0 1”就是“一串1”的结束，在0的位置加 $|X|$；
- 中间的“00”和“11”都不管！

【例】X＝0.1010，Y＝－0.1101，利用Booth法求积。

【解】：

$[X]_补 = 00.1010$

$[-X]_补 = 11.0110$

$[Y]_补 = 11.0011$

$\therefore [X \cdot Y]_补 = 1.01111110$

[Y]补含1位符号

-1位置尾数

符号		D				Y				Y_0	Y_{-1}	操作说明
0	0	0	0	0	0	1	0	0	1	1	0	D/Y初始化
1	1	0	1	1	0	1	0	0	1	1	0	尾数10，“串头”，-X
1	1	0	1	1	0	1	0	0	1	1	0	
1	1	1	0	1	1	0	1	0	0	1	1	尾数11，跳过
1	1	1	1	0	1	1	0	1	0	0	1	D/Y右移1位
0	0	1	0	1	0	0	0	1	0	0	1	尾数01，“串尾”，+X
0	0	0	1	1	1	1	0	1	0	0	1	
0	0	0	0	1	1	1	1	0	1	0	0	尾数00，跳过
0	0	0	0	0	1	1	1	1	0	1	0	D/Y右移1位
1	1	0	1	1	0	0	0	0	0	1	0	尾数10，“串头”，-X
1	1	0	1	1	1	1	1	1	0	1	0	
1	1	1.	0	1	1	1	1	1	1	0	1	D/Y右移2位

结束首位为符号

乘数用尽，计算结束

在此处减 $|X|$

题3.22

用原码加减交替法计算下面除法

1. x=-0.10101　y = -0.11011

2. x=0.1001110001 y = -0.10101

关于除法的“表格式计算”

解决方案二：无脑直减，减多了原地不恢复（加减交替法）

例，[X]原=0.1011, [Y]原=1.1101，求商/余数

减成负值商0

```
                 0 .1  1  0  1
 .1  1  0  1 ) .1  0  1  1
                1  1  0  1
            - 0  0  1  0  0
            +    1  1  0  1
                 1  0  0  1  0
                    1  1  0  1
                       1  0  1  0
                       1  1  0  1
                          - 1  1  0
                       +  1  1  0  1
              .0  0  0  0  0  1  1  1
```

符号		余数（R）				商	操作说明
0	0	1	0	1	1		R初始化
1	1	0	0	1	1		无脑直减，负了商0
1	1	1	1	1	0	0	
1	1	1	1	0	0		R左移1位
0	0	1	1	0	1		+D，正了商1
0	0	1	0	0	1	1	
0	1	0	0	1	0		R左移1位
1	1	0	0	1	1	1	无脑直减，正了商1
0	0	0	1	0	1		
0	0	1	0	1	0		R左移1位
1	1	0	0	1	1		无脑直减，负了商0
1	1	1	1	0	1	0	
1	1	1	0	1	0		R左移1位
0	0	1	1	0	1	1	，正了商1
0	0	0	1	1	1		

定点数运算—除法

解决方案三：补码交替加减法

例，[X]=-0.10001011, [Y]=0.1110，求商及余数

【解】

$[X]_补 = 1.01110101$

$[Y]_补 = 0.1110$

$[-Y]_补 = 1.0010$

1. R与D同号，-D
2. R与D异号，+D
3. 当新余数R与D相同时，商1
4. 当新余数R与D不同时，商0；
5. R左移1位，下一轮回到1
6. 除不尽时，商恒置1

符号		余数（R）								商	操作说明
1	1	0	1	1	1	0	1	0	1		R初始化
0	0	1	1	1	0	0	0	0	0		R与D异号，+D
0	0	0	1	0	1	0	1	0	1	1	R与D同号，商1
0	0	1	0	1	0	1	0	1	0		R左移1位
1	1	0	0	1	0	0	0	0	0		R与D同号，-D
1	1	1	1	0	0	1	0	1	0	0	R与D异号，商0
1	1	1	0	0	1	0	1	0	0		R左移1位
0	0	1	1	1	0	0	0	0	0		R与D异号，+D
0	0	0	1	1	1	0	1	0	0	1	R与D同号，商1
0	0	1	1	1	0	1	0	0	0		R左移1位
1	1	0	0	1	0	0	0	0	0		R与D同号，-D
0	0	0	0	0	0	1	0	0	0	1	R与D同号，商1
0	0	0	0	0	1	0	0	0	0		R左移1位
1	1	0	0	1	0	0	0	0	0		R与D同号，-D
1	1	0	0	1	1	0	0	0	0	0	R与D异号，商0
1	1	0	0	1	1	0	0	0	0	1	末尾恒1

$[X \div Y]_补 = 1.01101$

余数 $= 1.0011 \times 2^{-4}$

题3.26

浮点字长12位，阶码5位（移码表示），尾数7位（补码表示），计算下面数字的加减法

1. x=11/16*2^-4   y = 35/64*2^-3

2. x=0.101101*2^-11B   y = -0.100101*2^-1B

浮点运算—加减法

- So, 浮点加减法4步操作：
  - 对阶（低阶对齐到高阶）
  - 运算（补码+/-法）
  - 规格化（左/右移）
  - 舍入处理（截断法/恒置1/四舍五入）