数据的机器级表示与运算

整数的加/减法

主讲人: 邓倩妮

上海交通大学

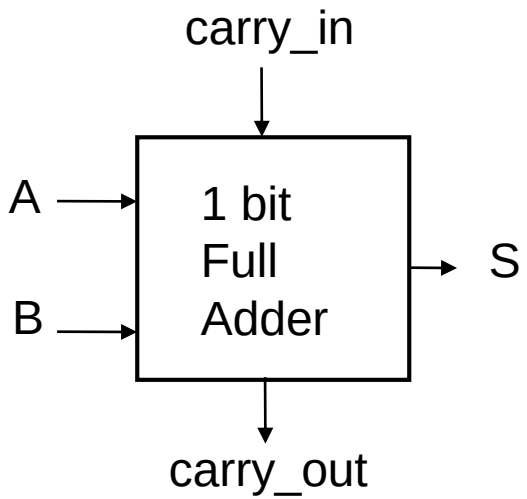
部分内容来自：

1. 《深入理解计算机系统》第三版，机械工业出版社，作者：Bryant,R.E.等
2. Computer Organization and Design, 4th Edition, Patterson & Hennessy

本节主要内容

- 加法器的工作原理
- 无符号数加/减法的实现
- 有符号数加/减法的实现
- 加/减法溢出的判断

1-bit Binary Adder （一位全加器）

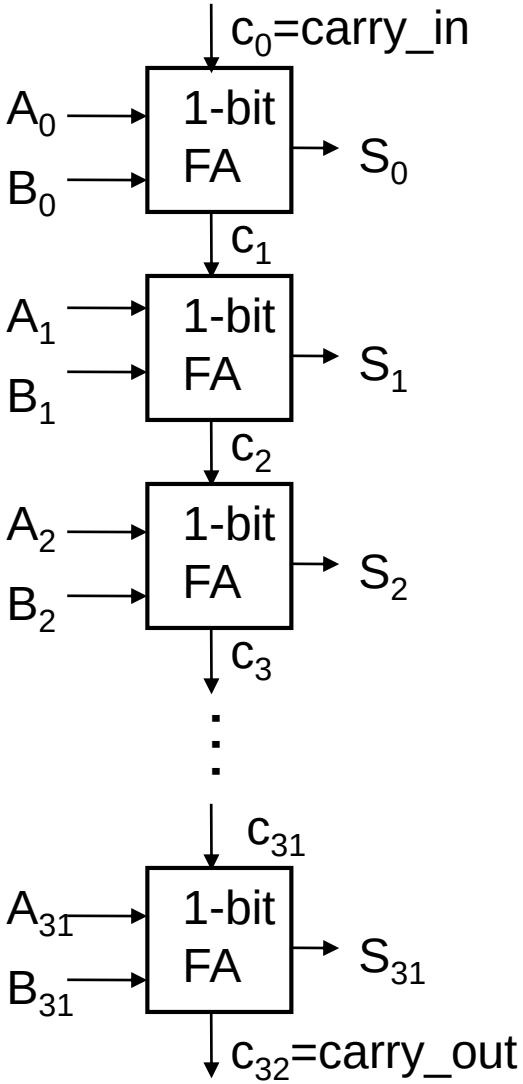


A	B	carry_in	carry_out	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S = A xor B xor carry_in

carry_out = A&B | A&carry_in | B&carry_in

构建 32-bit 加法器



- Just connect the carry-out of the least significant bit FA to the carry-in of the next least significant bit and connect . . .

- 串行行波加法器： Ripple Carry Adder (RCA)
  - 优点：简单
  - 缺点：速度慢

快速加法器

- 提前计算各位的进位输入
- 使得各位的加法运算能并行起来
- 提高多位加法器运算速度
- 请查阅参考资料

本节主要内容

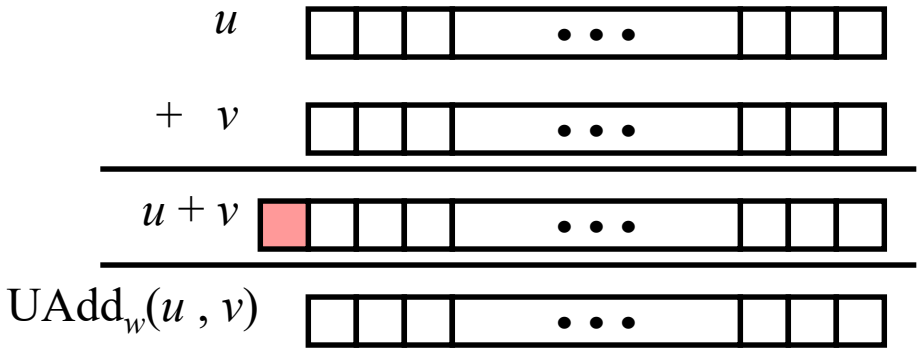
- 加法器的工作原理
- 无符号数加/减法的实现
- 有符号数加/减法的实现
- 加法溢出的判断

无符号加法

Operands: *w* bits

True Sum: *w*+1 bits

Discard Carry: *w* bits



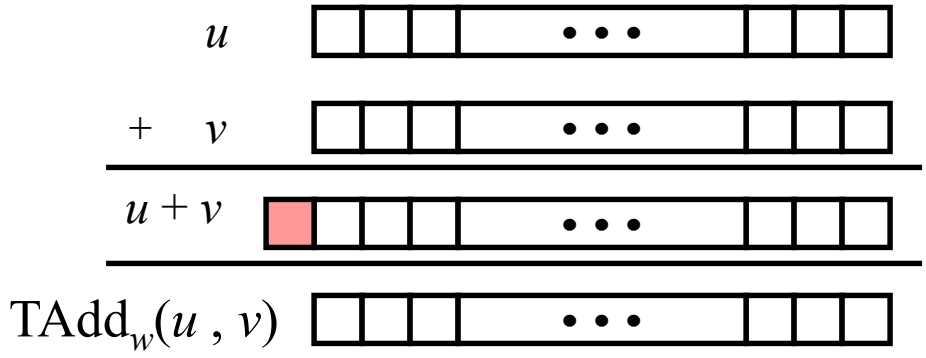
- 标准的加法：忽略最高位进位
  - 实现取模 （ 2^w ）运算 （字长w）
  - 加法： 0 → 1 → … → Umax → 0
  - s= $UAdd_w(u , v)$ $= u + v \mod 2^w$

补码加法

Operands: *w* bits

True Sum: *w*+1 bits

Discard Carry: *w* bits



- 无符号和有符号计算实现的操作是一样的：直接抛弃最高位的进位位
- 举例比较： Signed vs. unsigned addition in C:

```
int s, t, u, v;
s = (int) ((unsigned) u + (unsigned) v);
t = u + v
```

s == t 结果为 true

无符号减法

- 无符号数是一个模为2^w的计量系统 (w：字长)

  加法： 0 → 1 → … → Umax → 0

  减法： Umax → Umax -1 → … → 1 → 0 → Umax

- 实现取模 （2^w）运算
- 模运算系统：减法可以转化为加法
- $USub_w(u , v)= u - v + 2^w$

  $= u + (2^w - v) = u + \sim v + 1$

有符号数减法

- 补码加法
  - 根据补码加法公式，补码可以直接相加。
  - $[X]_{补} + [Y]_{补} = [X + Y]_{补}$
- 补码减法
  - 根据补码减法公式，补码减法可以转化为加法。

$[X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补} = [X-Y]_{补}$

$-[Y]_{补} = [-Y]_{补}$

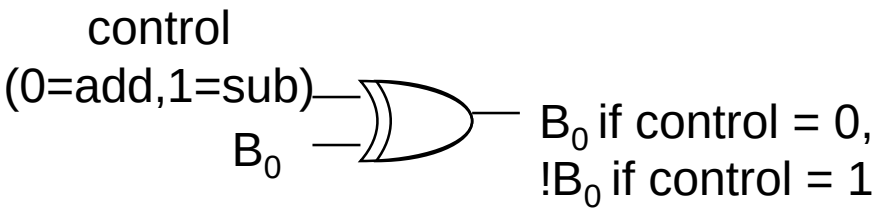
- 如何修改加法器电路构建一个 加/减法器?

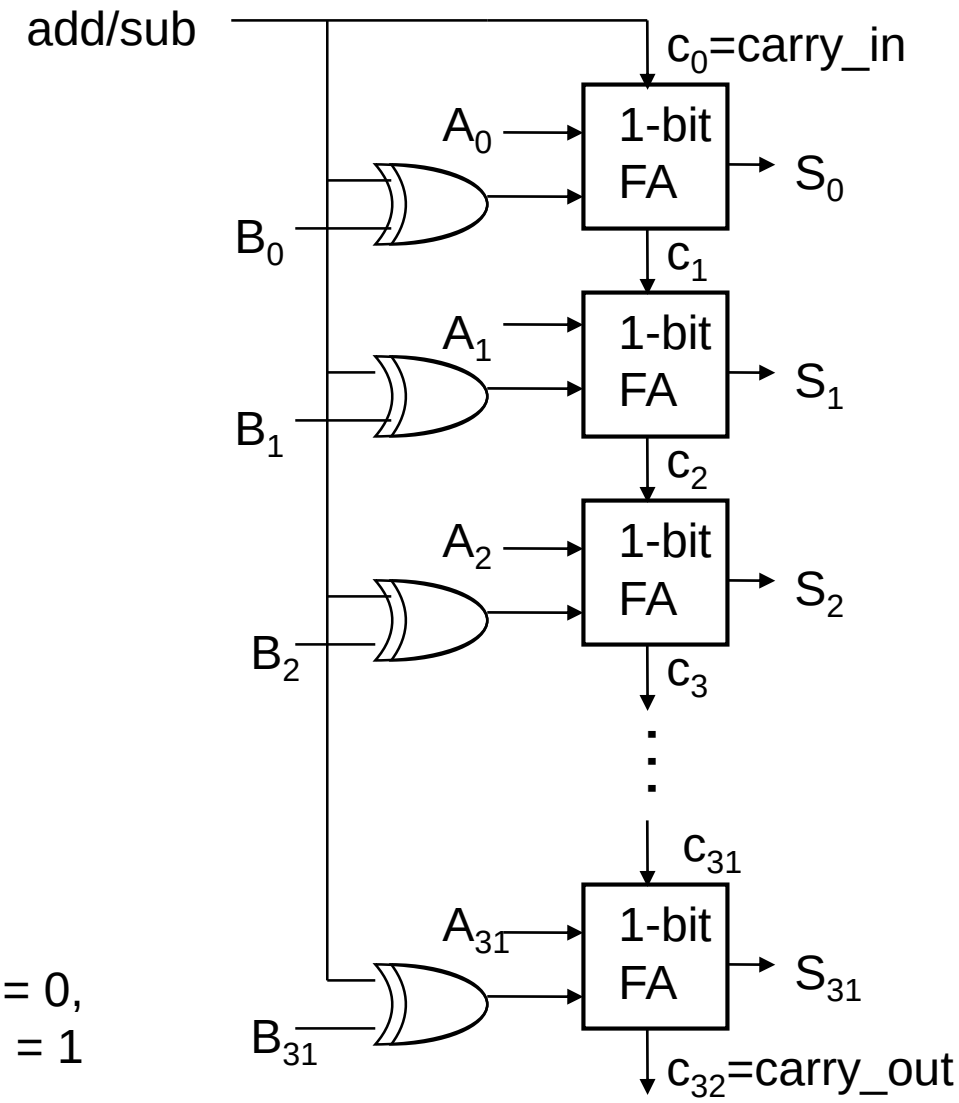
32-bit 串行进位的加/减法器

$[A-B]_{补} = [A]_{补} + [-B]_{补}$

- 已知$[B]_{补}$，求$[-B]_{补}$：
  - 取反，
  - 最低位加1

```
A     0111     →      0111
B   - 0110     →    + 1001
      0001            +    1
                      ------
                     1 0001
```





32-bit 串行进位的加/减法器（续）

1. 带符号数减法：

$[A-B]_{补} = [A]_{补} + [-B]_{补}$

- 已知$[B]_{补}$，求$[-B]_{补}$：
  - $[B]_{补}$取反， 最低位加1

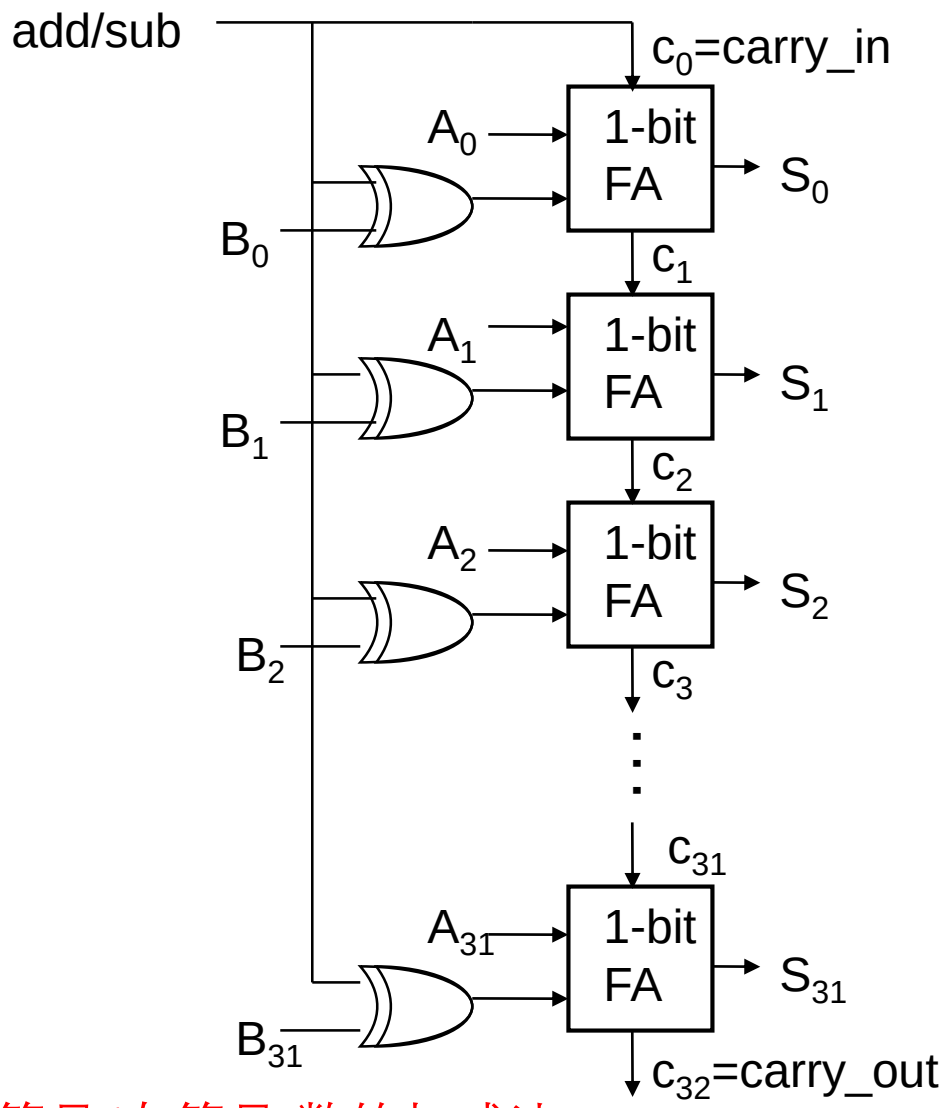
2. 无符号数减法：

$USub_w(u , v)$

$= u - v + 2^w$

$= u + (2^w - v)$

$= u + \sim v + 1$



此加/减法器同时适用于无符号/有符号 数的加减法

练习1

- 某计算机字长8位，其CPU中有一个8位加法器。已知无符号数x=69，y=38，现在要在该加法器中完成x-y运算，则该加法器的两个输入端信息和输入的低位进位信息分别为（）
- A.0100 0101, 0010 0110, 0
- B.0100 0101, 1101 1001, 1
- C.0100 0101, 1101 1010, 0
- D.0100 0101, 1101 1010, 1.

练习2

- 某计算机字长8位，其CPU中有一个8位加法器。已知有符号数x=-69，y=-38，现在要在该加法器中完成x+y运算，则该加法器的两个输入端信息和输入的低位进位信息分别为（）
- A.1011 1011, 1101 1010, 0
- B.1011 1011, 1101 1010, 1
- C.1011 1011, 0010 0101, 0
- D.1011 1011, 0010 0101, 1

本节主要内容

- 加法器的工作原理
- 无符号数加/减法的实现
- 有符号数加/减法的实现
- 加法溢出的判断

整数运算的溢出（Overflow）

- 整数在计算机内部通常用补码表示。
- 注意数据的表示范围
- **溢出：运算结果超出了表示范围**
- 例如： short i=23456;
- short j=23456; //最大值32767
- short k=i+j; //此时k的值是多少？//-18624

- C语言编译器规定：不报告此类错误
- 程序员必须自己保证程序中不出现这样的错误

有符号加法溢出的判断

- Overflow ：超出了数据表达的范围
- 符号相异的数相加、 符号相同的数相减, 不可能发生溢出

Operation	Operand A	Operand B	Result indicating overflow
A + B	⩾ 0	⩾ 0	< 0
A + B	< 0	< 0	⩾ 0
A - B	⩾ 0	< 0	< 0
A - B	< 0	⩾ 0	⩾ 0

补码加法的几种情况及其溢出检测

符号位进位C_f（C_n），最高位进位C_{n-1}

```
  0.10101          1.10101
+ 0.01000        + 1.11000
----------       ----------
  0.11101          1 1.01101
```

正常结果
$C_f = 0$，$C_{n-1} = 0$

符号位进位舍去，正常结果
$C_f = 1$，$C_{n-1} = 1$

```
  0.10101          1.00101
+ 0.11000        + 1.11000
----------       ----------
  1.01101          1 0.11101
```

正正得负，正溢出
$C_f = 0$，$C_{n-1} = 1$

负负得正，负溢出
$C_f = 1$，$C_{n-1} = 0$

单符号位的数溢出检测

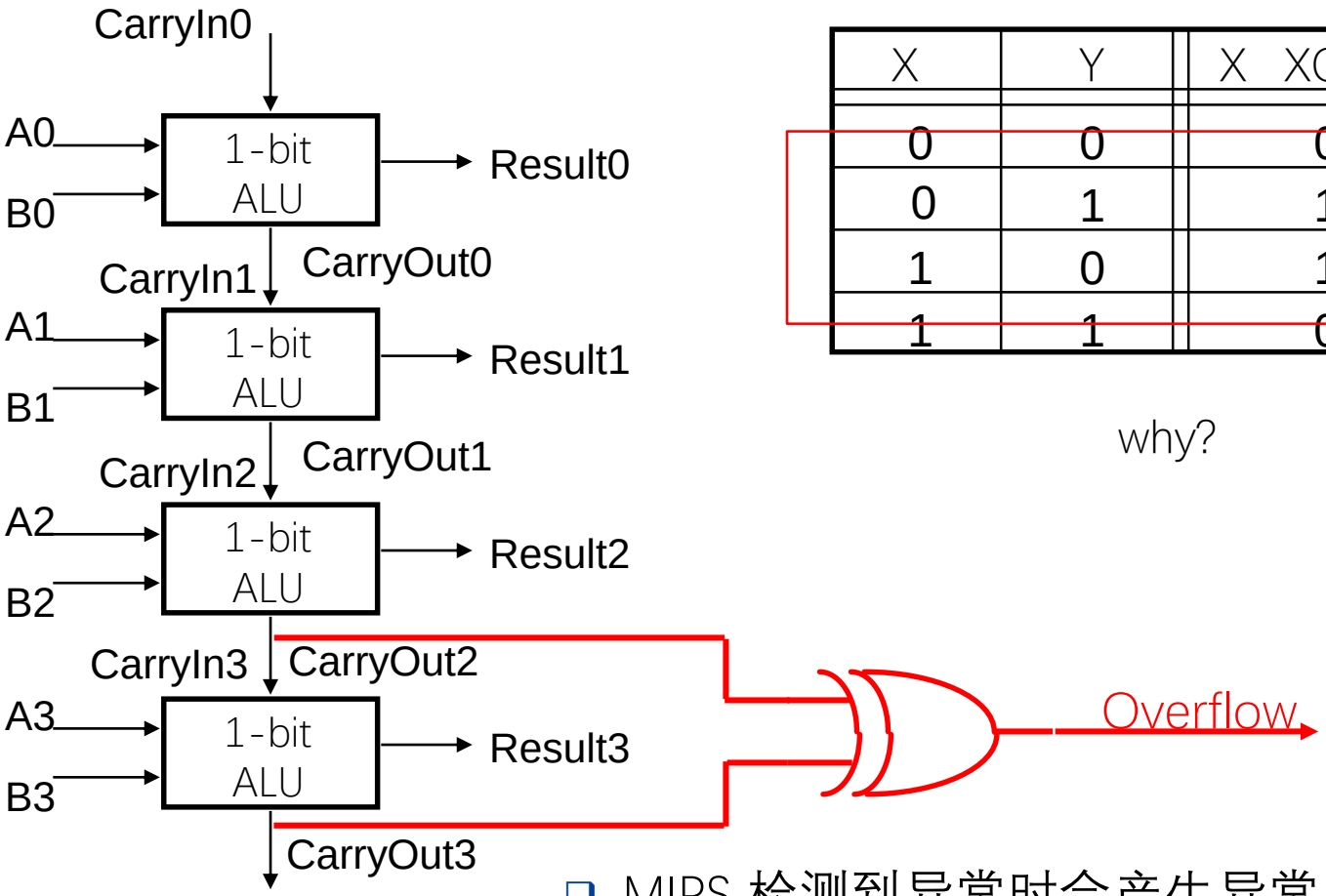
C_f	C_{n-1}	V
0	0	0
1	1	0
1	0	1
0	1	1

溢出信号V对应的真值表

$$V=C_f \oplus C_{n-1}$$

溢出判断逻辑

For a N-bit ALU: Overflow = CarryIn [N-1] XOR CarryOut [N-1]

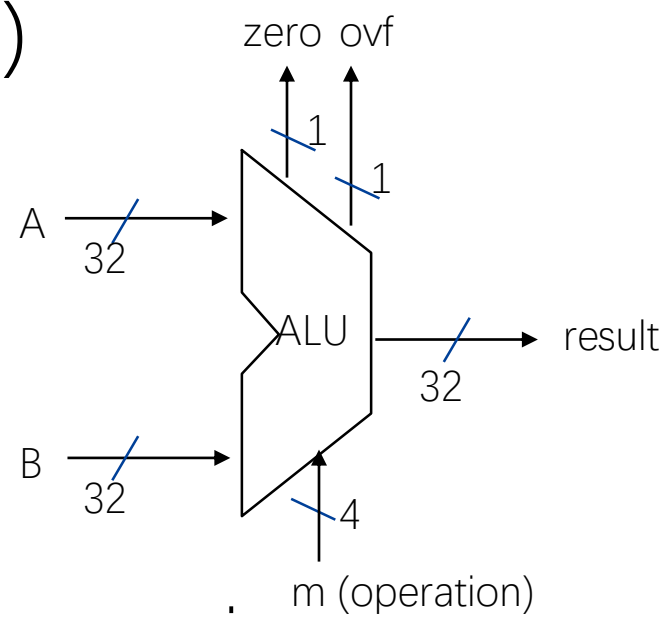


X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

why?

- MIPS 检测到异常时会产生异常（中断）
- EPC （异常程序计数器）保存导致异常的指令

MIPS Arithmetic Logic Unit (ALU)



- 支持的算术、逻辑运算指令
  - `add, addi, addiu, addu`
  - `sub, subu`
  - `mult, multu, div, divu`
  - `sqrt`
  - `and, andi, nor, or, ori, xor, xori`
  - `beq, bne, slt, slti, sltiu, sltu`

- 需要进行特殊处理：
  - 符号位扩展 sign extend – `addi, addiu, slti, sltiu`
  - 零扩展 zero extend – `andi, ori, xori`
  - 溢出检测 overflow detection – `add, addi, sub`

无符号 vs 有符号： 溢出的处理

- MIPS指令：
  - 有符号 add , addi, sub， 在溢出时产生异常
  - 无符号 addu, addiu, subu, 溢出时不会产生异常
- 无符号加法：硬件不检测溢出，将其忽略
  - 在程序中自行判断
    - 例如：结果比加数和被加数任何一个都小，则溢出；
- 有符号加法： 有硬件检验电路，有溢出信号
  - 也可能报异常：如Fortran
  - 可能被编译器忽略，例如C, Java；MIPS C编译器总是采用addu, addiu, subu, 来完成加法运算，而不考虑变量类型。程序员必须自己保证不出现错误。

小结

- MIPS支持的实现加法、减法功能的指令
  - add, addi, addiu, addu
  - sub, subu
- 计算机内部用一个ALU就能完成以上所有的指令
  - 无符号数、有符号数的加减法实现逻辑是一样的
- 溢出判断

  只对有符号数加法溢出有硬件检测

  C编译器忽略溢出信号，在程序中，无论是无符号加法、有符号加法，都要注意运算结果是否溢出，防止程序错误

谢谢！