中国科学院大学
University of Chinese Academy of Sciences

B0911006Y-01　2024-2025学年春季学期

计算机组成原理

Principles of Computer Organization

第 8 讲　计算机中数的运算II

定点加减法及溢出判断；ALU结构

主讲教师：石　侃

shikan@ict.ac.cn

2025年3月19日

二、加减法运算

6.3

1. 补码加减运算公式

(1) 加法

整数 $[A]_{补} + [B]_{补} = [A+B]_{补}$ （mod 2^{n+1}）

小数 $[A]_{补} + [B]_{补} = [A+B]_{补}$ （mod 2）

(2) 减法

$$A-B = A+(-B)$$

整数 $[A-B]_{补} = [A+(-B)]_{补} = [A]_{补} + [-B]_{补}$ (mod 2^{n+1})

小数 $[A-B]_{补} = [A+(-B)]_{补} = [A]_{补} + [-B]_{补}$ (mod 2)

连同符号位一起相加，符号位产生的进位自然丢掉

2. 举例

例 6.18　设 $A = 0.1011$，$B = -0.0101$

求 $[A + B]_{补}$　　验证

解：

$$
\begin{array}{rl}
 & [A]_{补} = 0.1011 \\
+ & [B]_{补} = 1.1011 \\
\hline
 & [A]_{补} + [B]_{补} = 10.0110 = [A + B]_{补}
\end{array}
$$

$$
\begin{array}{r}
0.1011 \\
-\ 0.0101 \\
\hline
0.0110
\end{array}
$$

$\therefore A + B = 0.0110$

例 6.19　设 $A = -9$，$B = -5$

求 $[A+B]_{补}$　　验证

解：

$$
\begin{array}{rl}
 & [A]_{补} = 1,0111 \\
+ & [B]_{补} = 1,1011 \\
\hline
 & [A]_{补} + [B]_{补} = 11,0010 = [A + B]_{补}
\end{array}
$$

$$
\begin{array}{r}
-\ 1001 \\
+\ -\ 0101 \\
\hline
-\ 1110
\end{array}
$$

$\therefore A + B = -1110$

例 6.20 设机器数字长为 8 位（含 1 位符号位）且 $A = 15$，$B = 24$，用补码求 $A - B$

解：$A = 15 = 0001111$

$B = 24 = 0011000$

$[A]_{补} = 0, 0001111$　　$[B]_{补} = 0, 0011000$

$+ [-B]_{补} = 1, 1101000$

$[A]_{补} + [-B]_{补} = 1, 1110111 = [A - B]_{补}$

$\therefore A - B = -1001 = -9$

练习 1　设 $x = \frac{9}{16}$　$y = \frac{11}{16}$，用补码求 $x+y$

$x + y = -0.1100 = -\frac{12}{16}$　　错

练习 2　设机器数字长为 8 位（含 1 位符号位）且 $A = -97$，$B = +41$，用补码求 $A - B$

$A - B = +1110110 = +118$　　错

(−97)−(41)= −138 超出了8位机器字长的表示范围☹

3. 溢出判断

$A = -97,\ B = +41$

$[A-B]_{补} = [A]_{补} + [-B]_{补}$

$= \overset{0}{1}0011111$

$+\ 11010111$

$= \boxed{1}01110110 = +118$

(1) 一位符号位判溢出

参加操作（不论加减）的 两个数（减法时即为被减数和“求补”以后的减数）符号相同，而其结果的符号与原操作数的符号不同，即为溢出

硬件实现

异或操作

符号位的进位 ⊕ 最高有效位的进位 = 1　　溢出

如

$1 \oplus 0 = 1$
$0 \oplus 1 = 1$ } 有 溢出

$0 \oplus 0 = 0$
$1 \oplus 1 = 0$ } 无 溢出

(2) 两位符号位判溢出

$$[x]_{补'} = \begin{cases} x & 1 > x \geq 0 \\ 4 + x & 0 > x \geq -1 \end{cases} \quad (\text{mod } 4)$$

双符号位补码，或称变形补码，用于溢出判断和浮点数中的阶码运算

$$[x]_{补'} + [y]_{补'} = [x + y]_{补'} \quad (\text{mod } 4)$$

$$[x - y]_{补'} = [x]_{补'} + [-y]_{补'} \quad (\text{mod } 4)$$

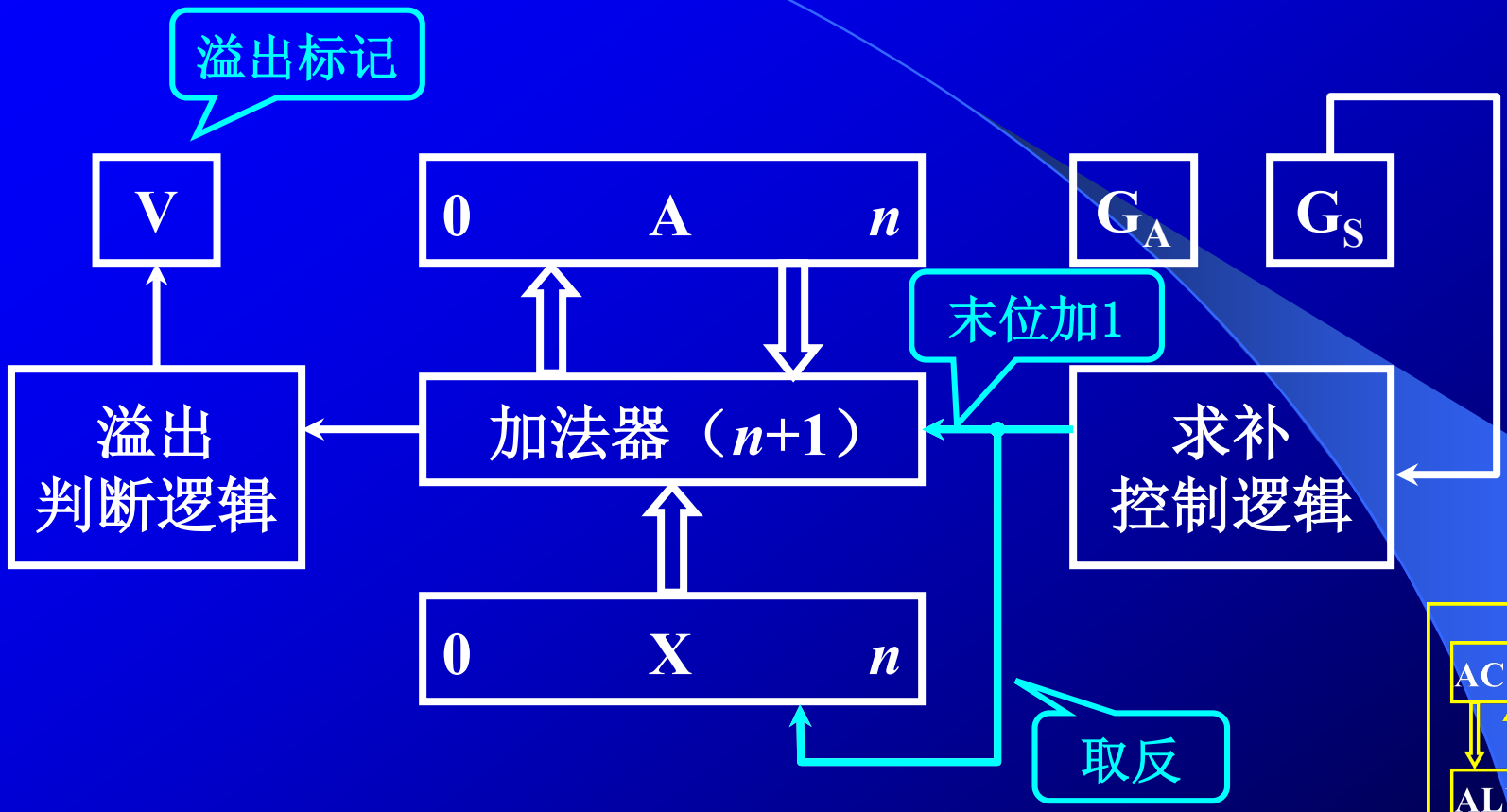
小数或整数都是一样的判断规则

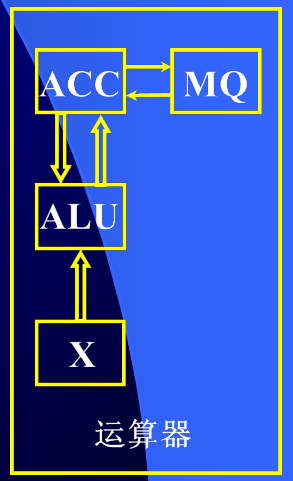
结果的双符号位 相同　　未溢出　　00, ×××××
11, ×××××

结果的双符号位 不同　　溢出　　10, ×××××
01, ×××××

最高符号位（左符） 代表其 真正的符号

4. 补码加减法的硬件配置



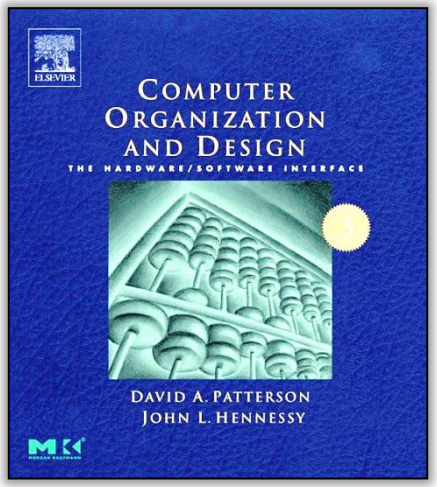


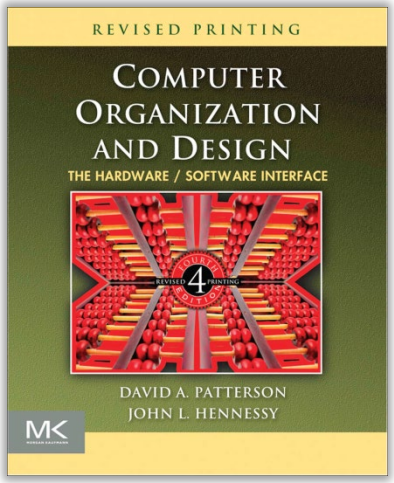
A（或称ACC）、X 均 n+1 位寄存器

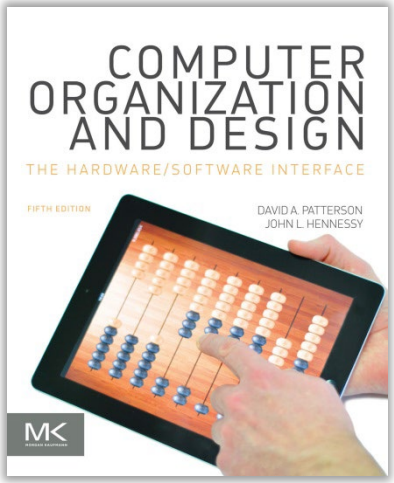
G_A 为加法标记；用减法标记 G_S 控制求补逻辑

算术逻辑单元 ALU

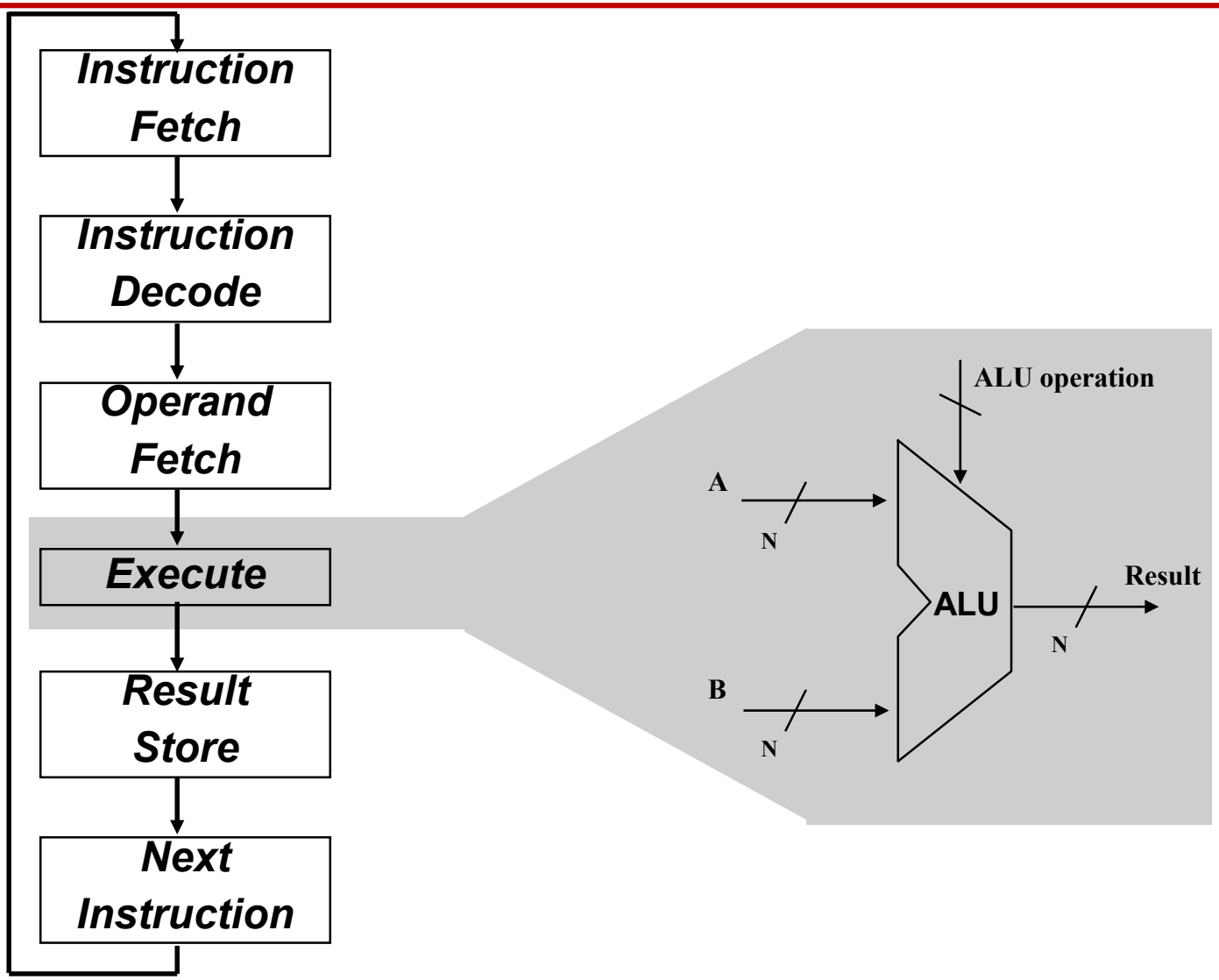
- ALU / Arithmetic Logic Unit
- Appendix Chapter "The Basics of Logic Design" in COD
  - Computer Organization and Design, The Hardware/Software Interface
  - Appendix B of COD5E
  - Appendix C of COD4ER
  - Appendix B of COD3E
  - 注：教材第6.5节以ALU电路为主，这里强调逻辑结构和控制



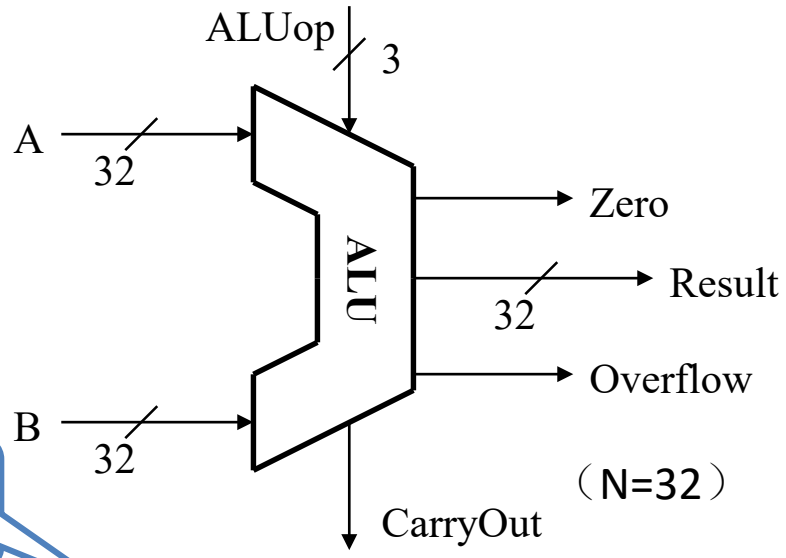




ALU: The key part of instruction execution



Designing an ALU



ALUop

ALU内部的动作

ALU control input	Function	Instruction
000	And	and
001	Or	or
010	Add	add, lw, sw
110	Subtract	sub, beq
111	Slt	slt

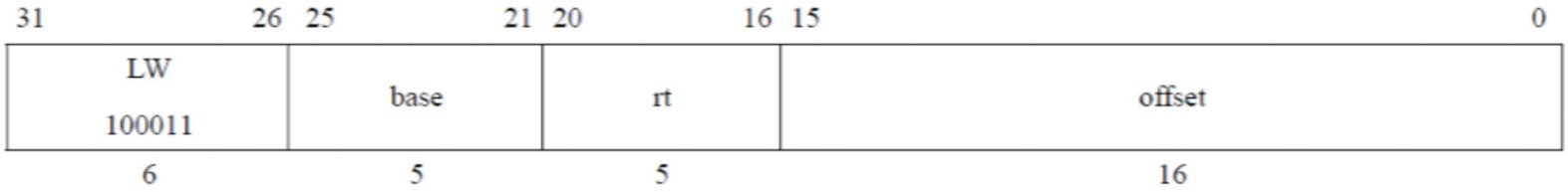
哪些指令会和相应的ALU内部动作有关

- lw/sw: load word / store word
- beq: branch on equal
- slt: set on less than (有符号整数比较)

LW in MIPS32 Instruction Set

Mnemonic/助记符

Load Word LW



Format: `LW rt, offset(base)` MIPS32 (MIPS I)

Purpose:

To load a word from memory as a signed value

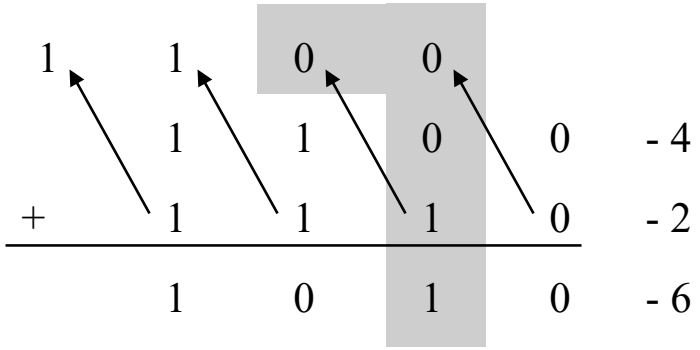
Description: `rt ← memory[base+offset]`

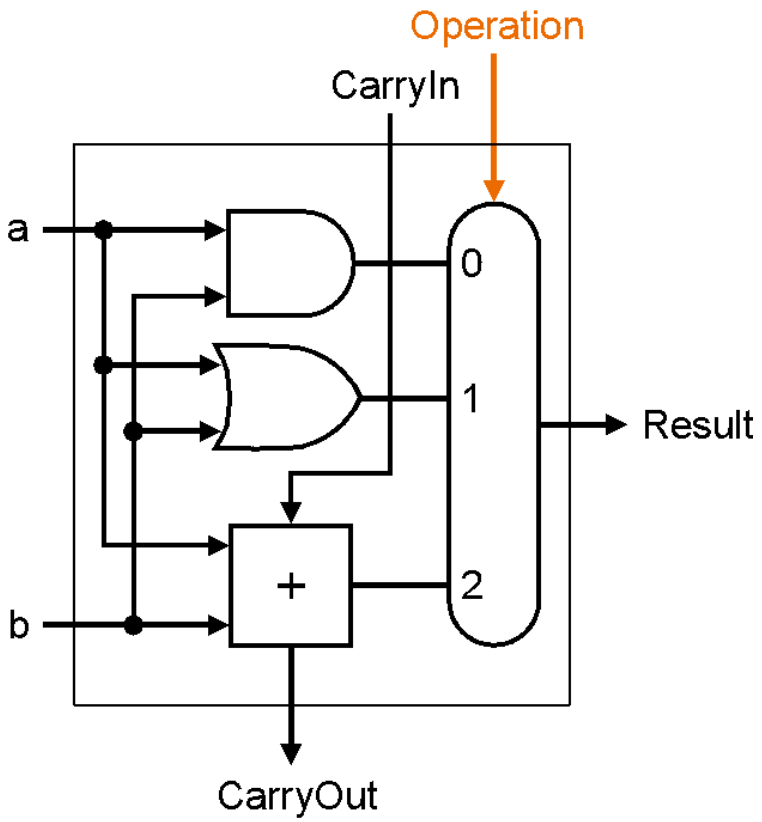
The contents of the 32-bit word at the memory location specified by the aligned effective address are fetched, sign-extended to the GPR register length if necessary, and placed in GPR *rt*. The 16-bit signed *offset* is added to the contents of GPR *base* to form the effective address.

来自MIPS32指令手册《MIPS32™ Architecture For Programmers Volume II, Revision 0.95》（MIPS_Vol 2.pdf），第130页

1-bit ALU

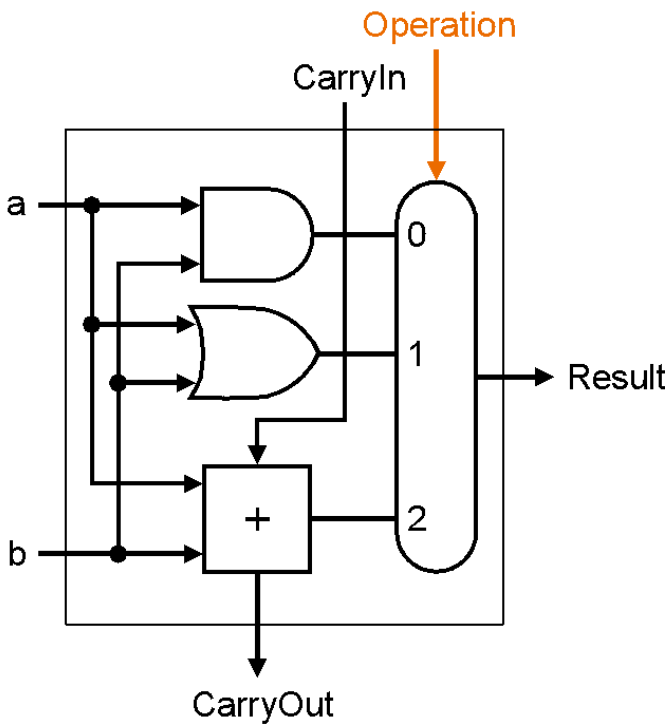
- This 1-bit ALU will perform AND, OR, and ADD



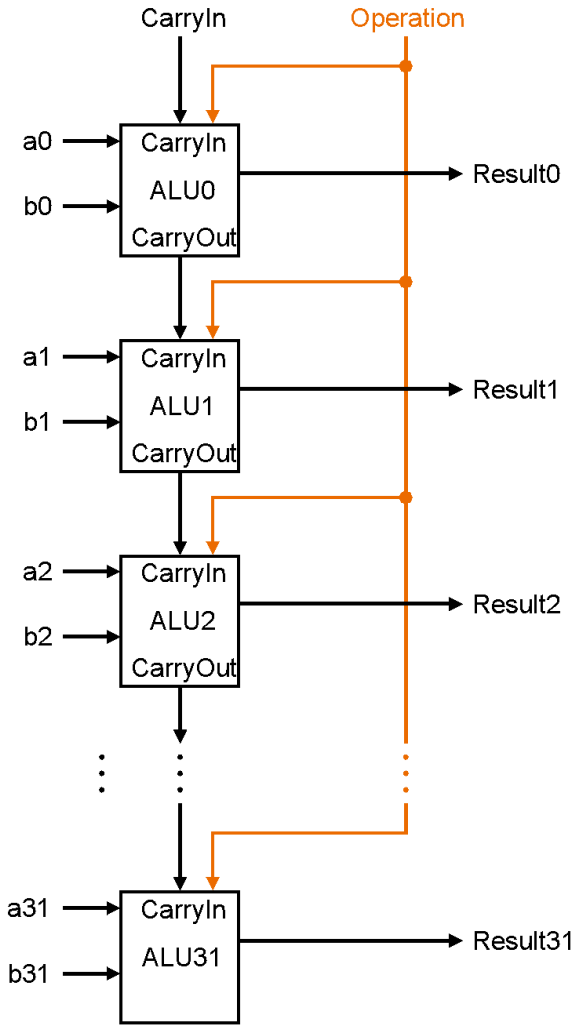


32-bit ALU

1-bit ALU



32-bit ALU

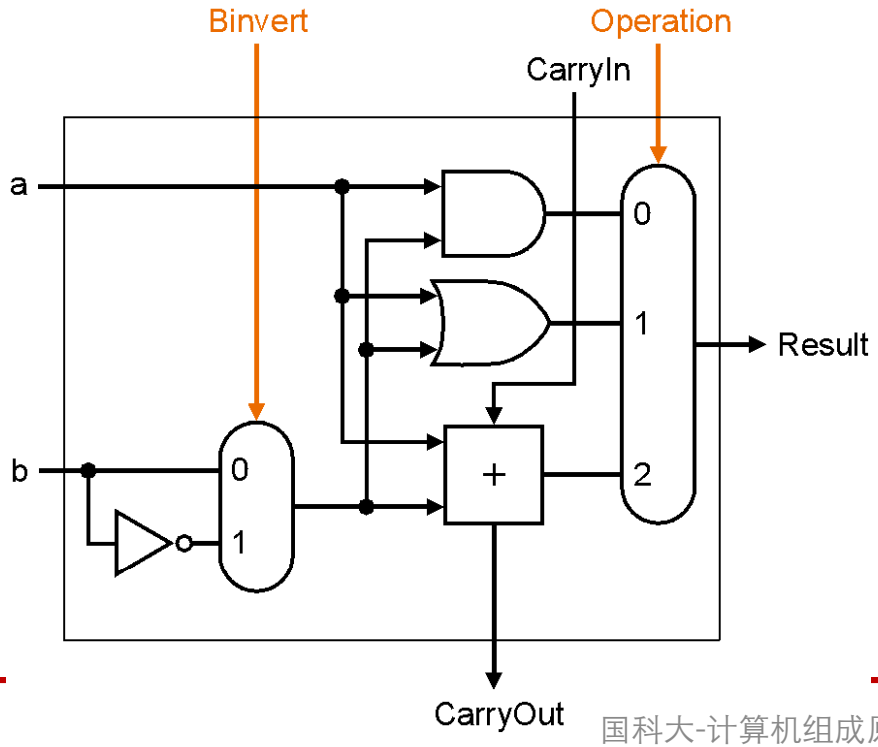


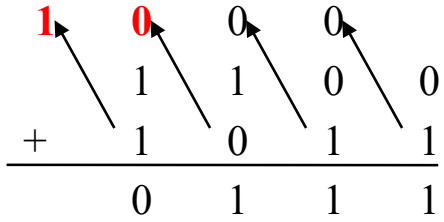
How About Subtraction?

- **Keep in mind the following:**
  - (A - B) is the same as: A + (-B)
  - 2's Complement negate: Take the inverse of every bit and add 1
- **Bit-wise inverse of B is defined as !B here:**
  - A - B = A + (-B) = A + (!B + 1) = A + !B + 1

对减数进行求补操作：
按位取反并加1

注意：Verilog中两个
单目运算符的不同
!（逻辑取非）
~（按位取反）

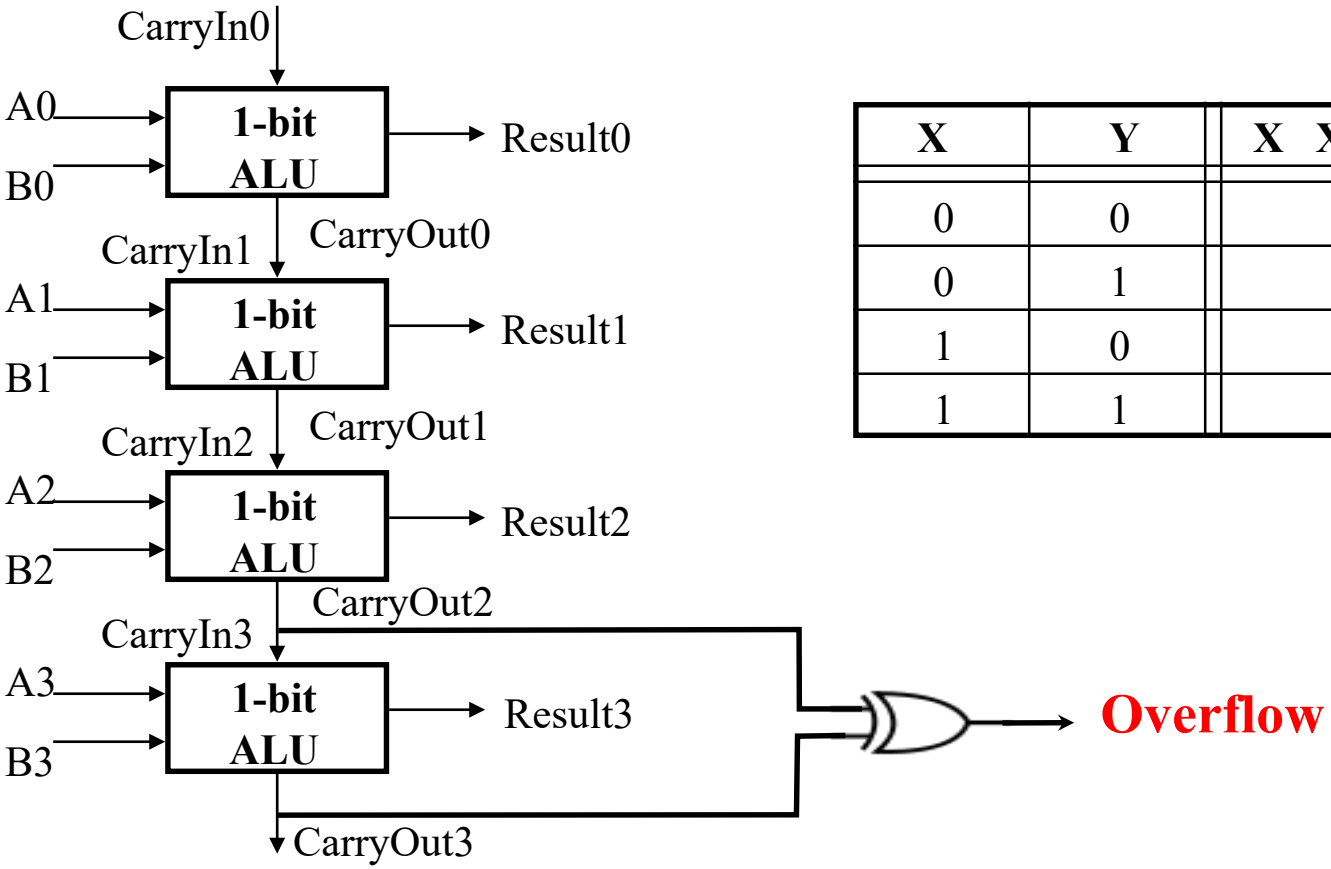




Overflow Detection Logic

- **Carry into MSB ! = Carry out of MSB**

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

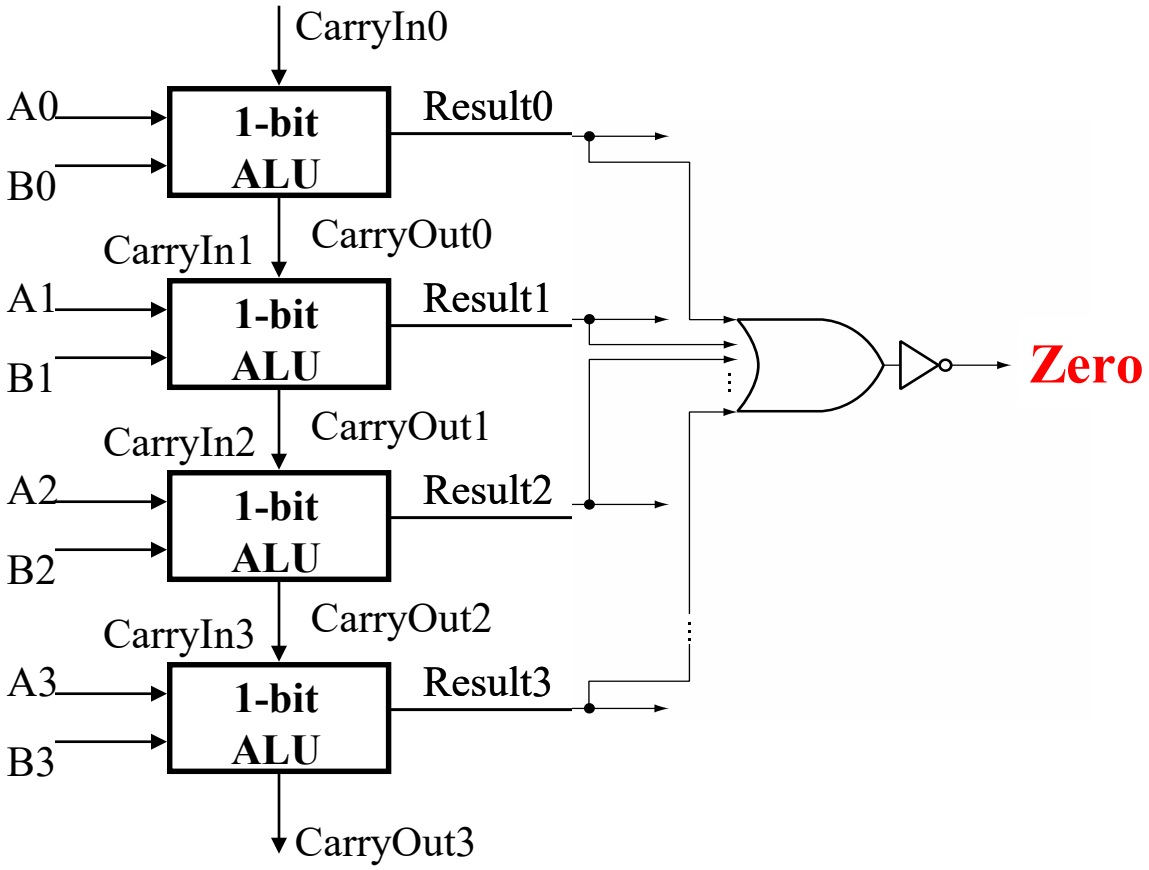


X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Zero Detection Logic

- **Zero Detection Logic is just one BIG NOR gate**

  Any non-zero input to the NOR gate will cause its output to be zero

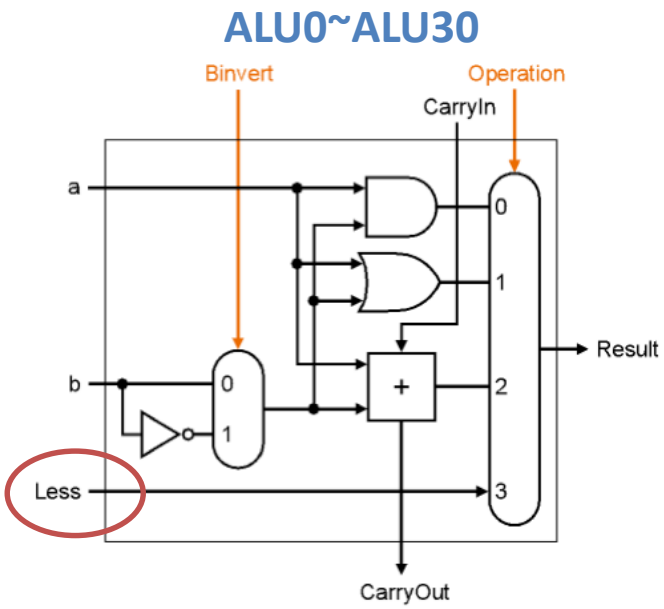


Slt：Set-on-less-than

- **We are mostly there!**
- **What is less-than in an Slt operation?**
  - A < B => (A - B) < 0
  - Do a subtract
- **If true, set LSB of result to 1 and all other bits to 0**
  - Use sign bit
    - route the sign bit to bit#0 of result
    - all other bits are set to zero

LSB：least-significant bit

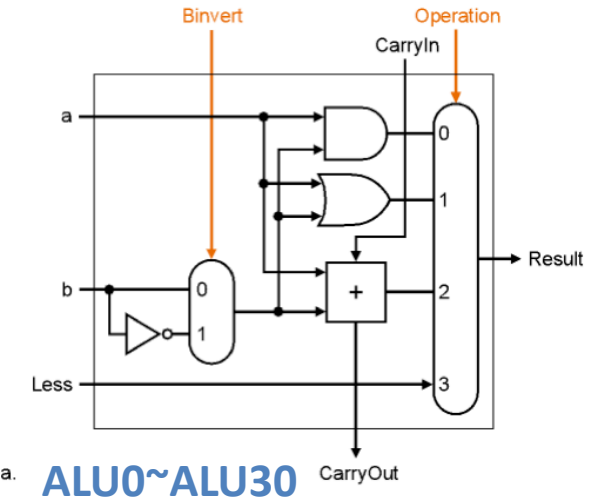
MSB：most-significant bit



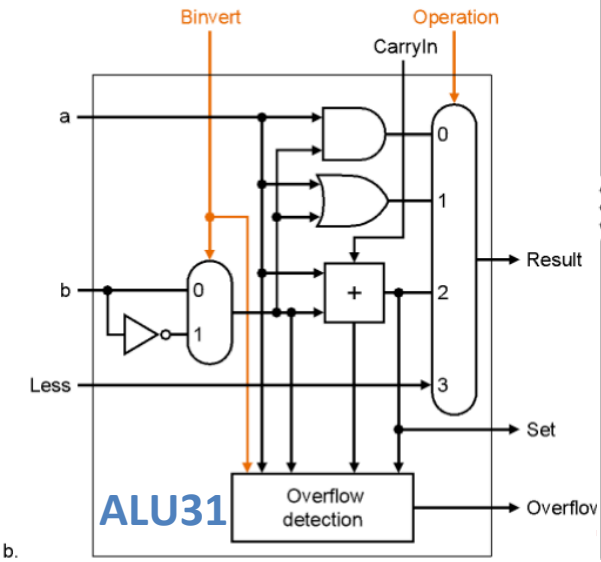
- **做减法的同时，若不溢出（Overflow=0），把ALU减法结果符号位送至最低位ALU的Less输入端，其他位ALU的Less输入置零，最终输出所有的Less到Result**
- **Slt为有符号整数比较，最高符号位还需与Overflow进行异或操作才能供给最低位ALU的Less输入端**

Full ALU (一种可能的实现)

要执行这几条指令时，ALU模块端口控制信号应赋值成什么？

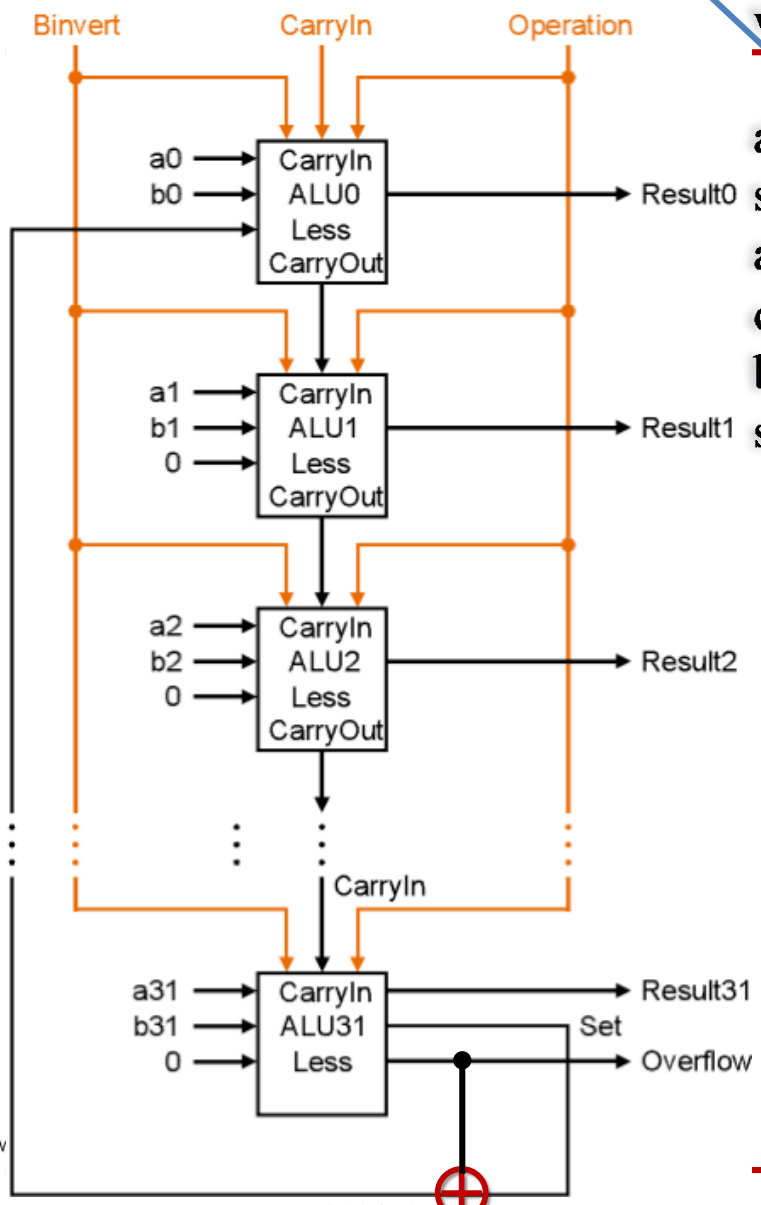


a. ALU0~ALU30



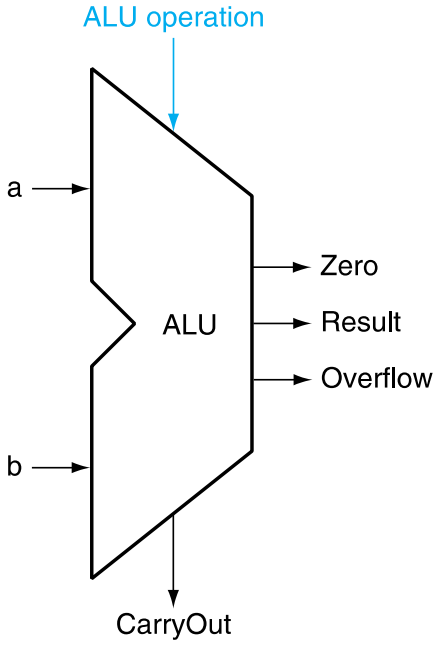
ALU31

b.



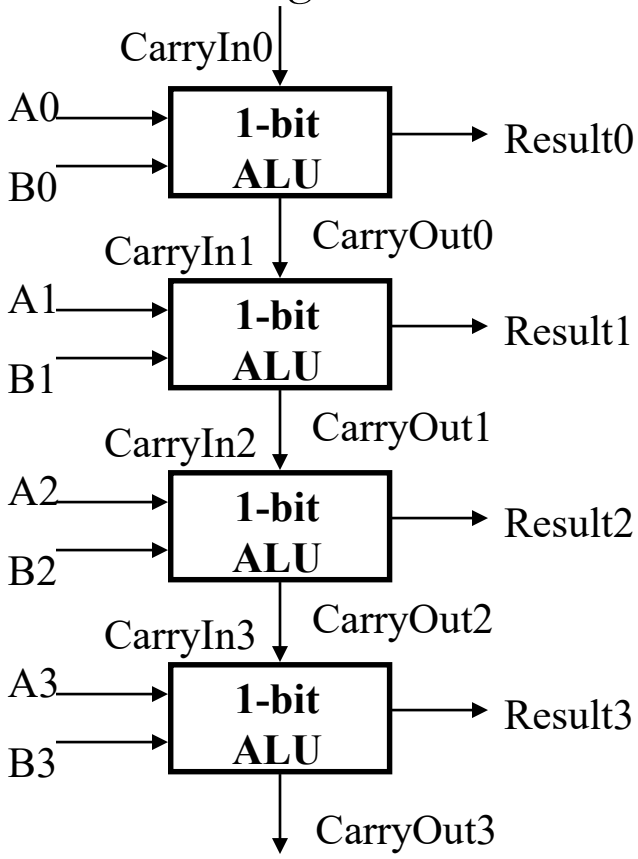
what signals accomplish:

	Binvert	CIn	Oper
add?	0	0	10
sub?	1	1	10
and?	0	0	00
or?	0	0	01
beq?	1	1	10
slt?	1	1	11



Ripple Carry Adder (行波进位加法器)

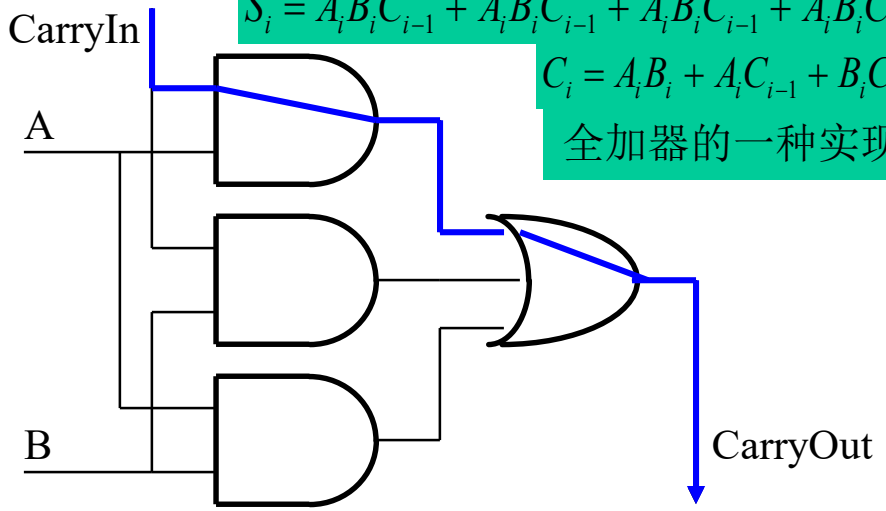
- **The adder we just built is called a "Ripple Carry Adder"**
  - The carry bit may have to propagate from LSB to MSB
  - Disadvantage - worst case delay for an N-bit RC adder: 2N-gate delay



$$S_i = \overline{A}_i\overline{B}_iC_{i-1} + \overline{A}_iB_i\overline{C}_{i-1} + A_i\overline{B}_i\overline{C}_{i-1} + A_iB_iC_{i-1}$$

$$C_i = A_iB_i + A_iC_{i-1} + B_iC_{i-1}$$

全加器的一种实现



The point -> ripple carry adders are slow. Faster addition schemes are possible that *accelerate* the movement of the carry from one end to the other.
详见唐朔飞教材6.5.2