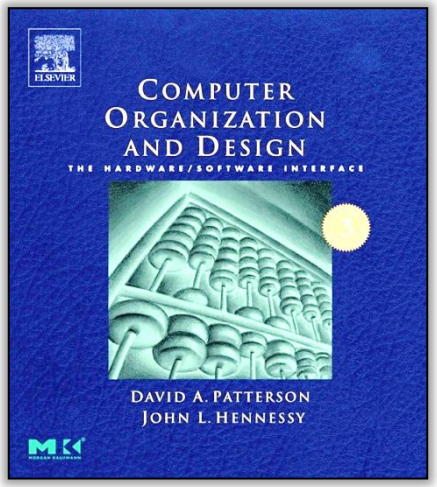
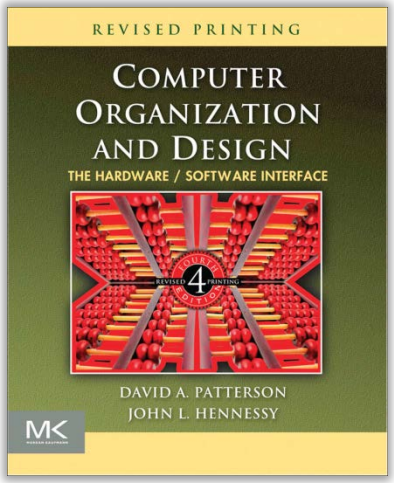
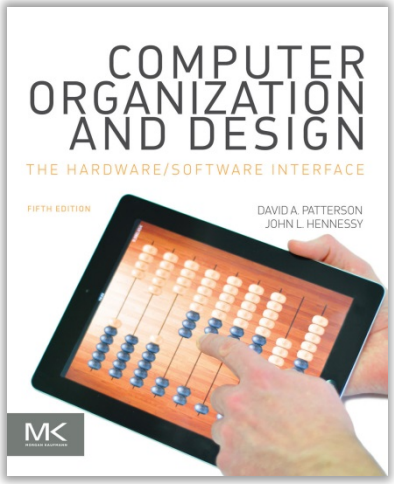
重点：算术逻辑单元 ALU

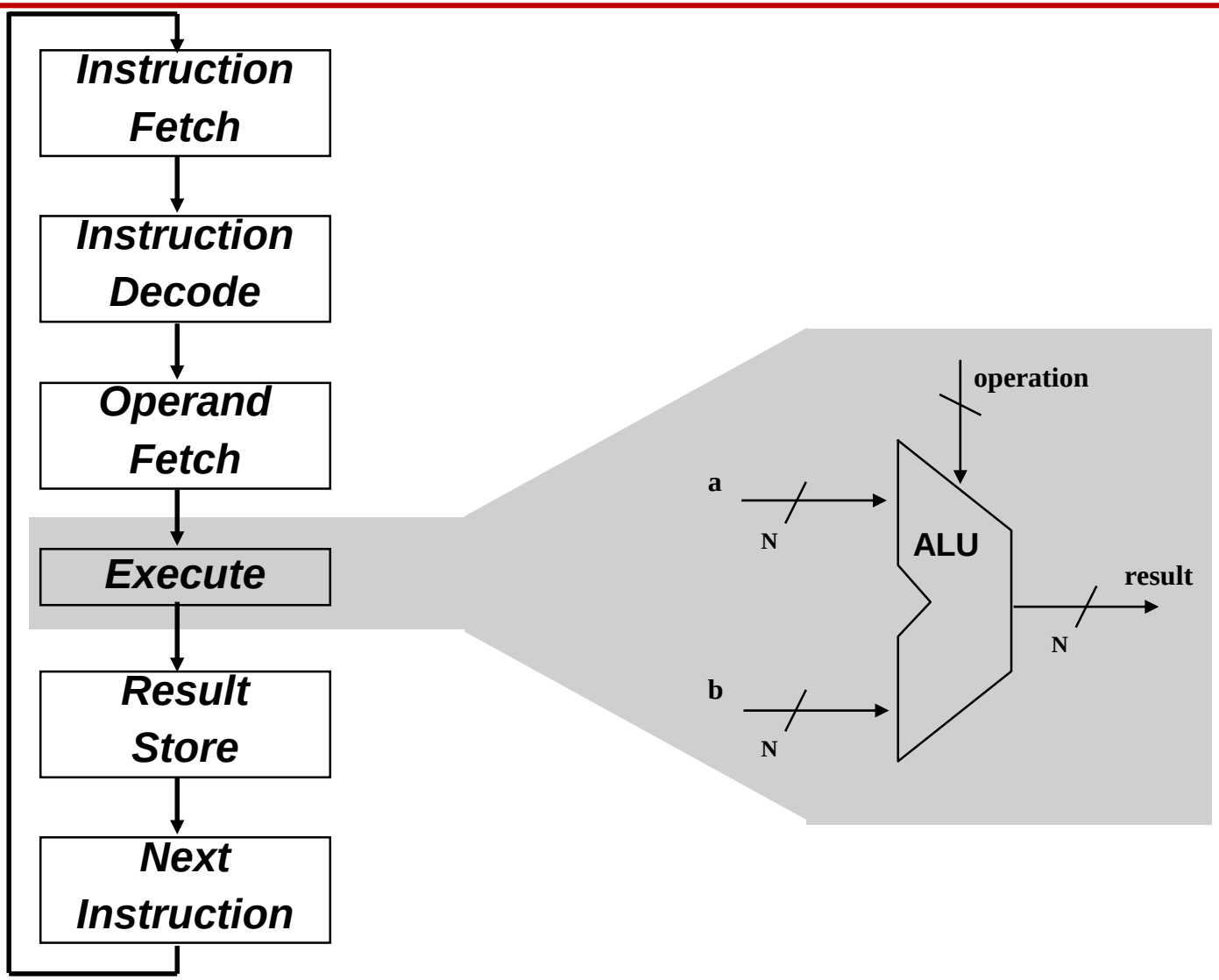
- **ALU / Arithmetic Logic Unit**
- **Appendix Chapter "The Basics of Logic Design" in COD**
  - **C**omputer **O**rganization and **D**esign, The Hardware/Software Interface
  - Appendix B of COD5E
  - Appendix C of COD4ER
  - Appendix B of COD3E
  - 注：教材第6.5节以ALU电路为主，这里强调逻辑结构和控制



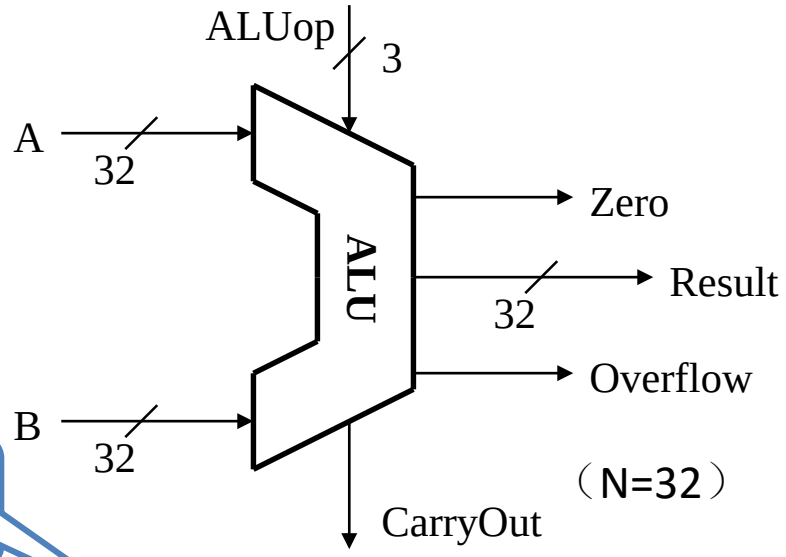




ALU: The key part of instruction execution



Designing an ALU



ALUop

ALU内部的动作

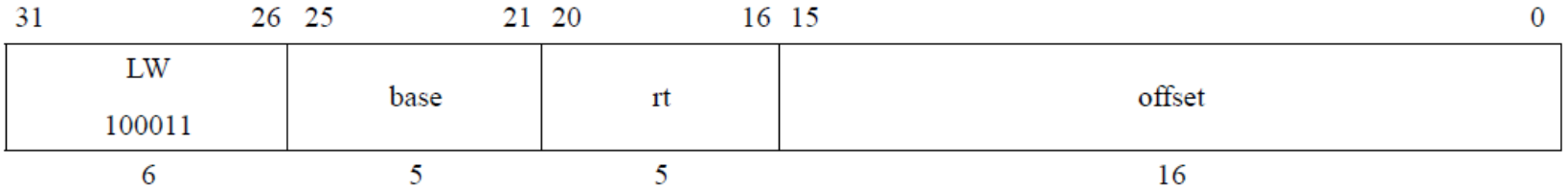
ALU control input	Function	Operations
000	And	and
001	Or	or
010	Add	add, lw, sw
110	Subtract	sub, beq
111	Slt	slt

ALU这些动作会和哪些指令有关

- 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

Load Word **LW**



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

Purpose:

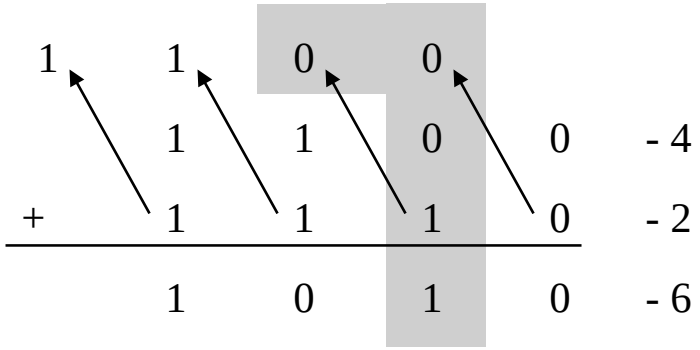
To load a word from memory as a signed value

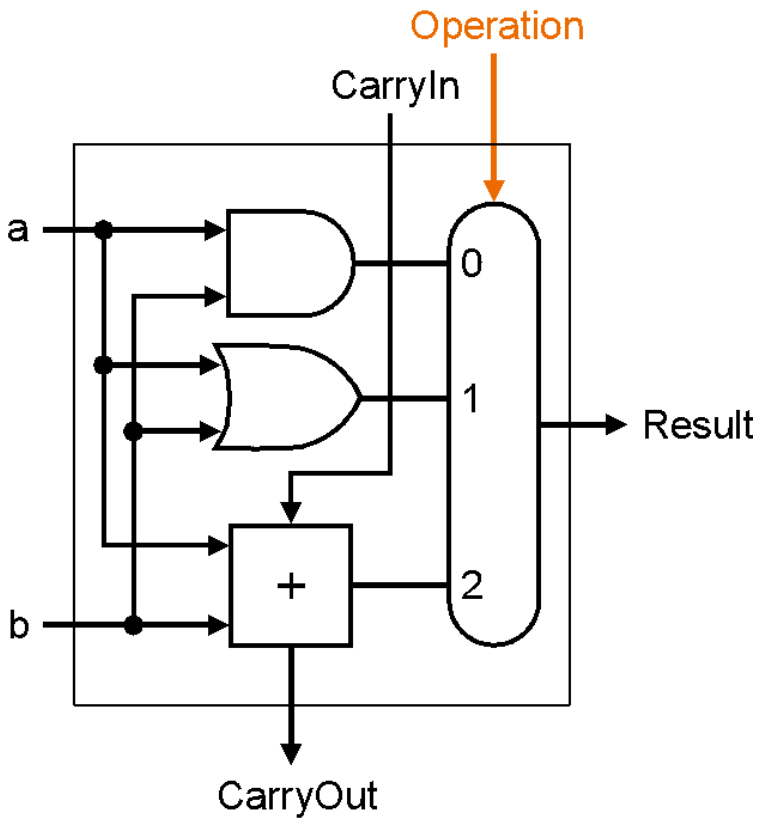
Description: `rt ← memory[base+offset]`

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.

A One-Bit ALU

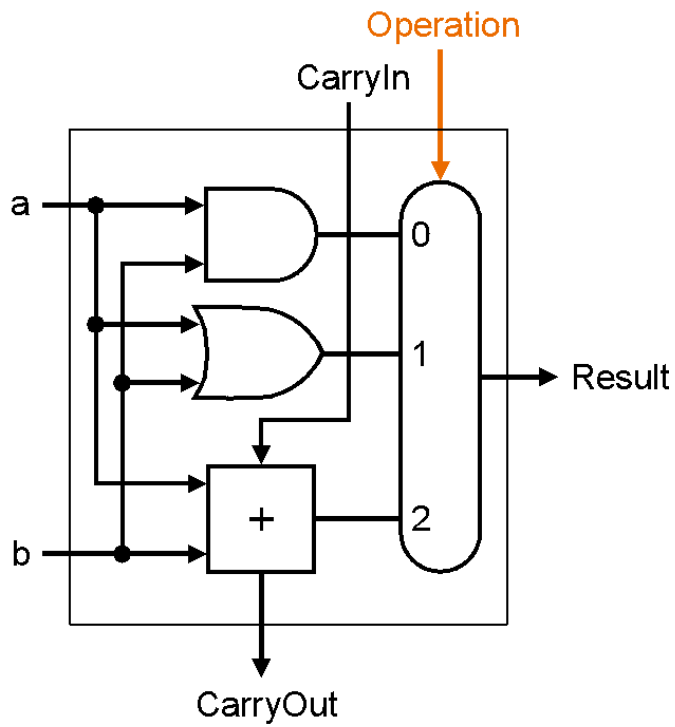
- This 1-bit ALU will perform AND, OR, and ADD



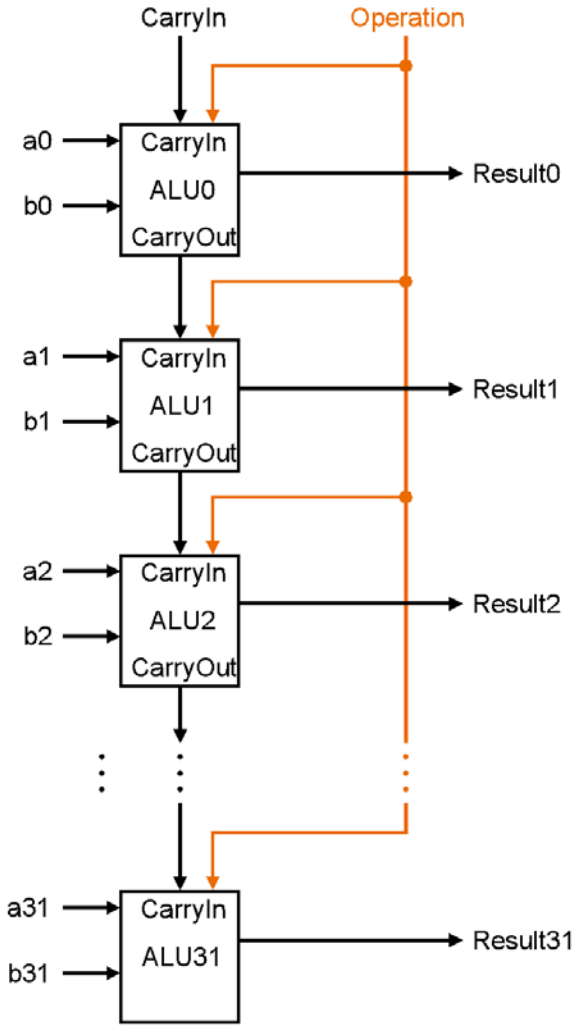


A 32-bit ALU

1-bit ALU

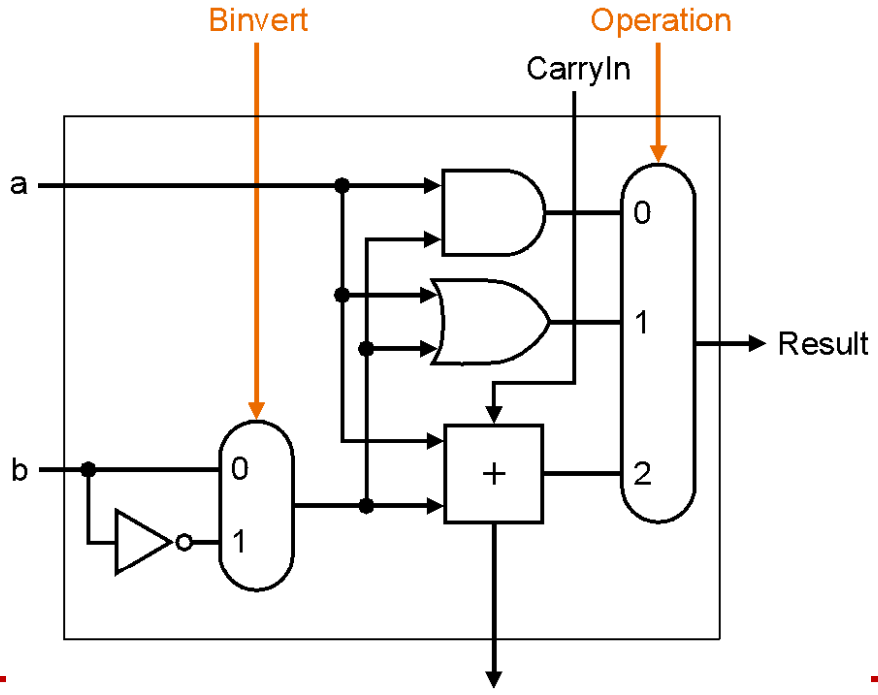


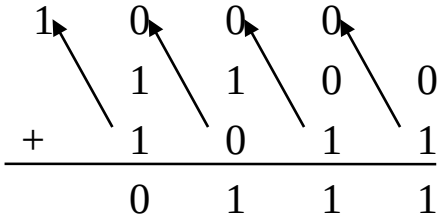
32-bit ALU



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 !B:**
  - A - B = A + (-B) = A + (!B + 1) = A + !B + 1

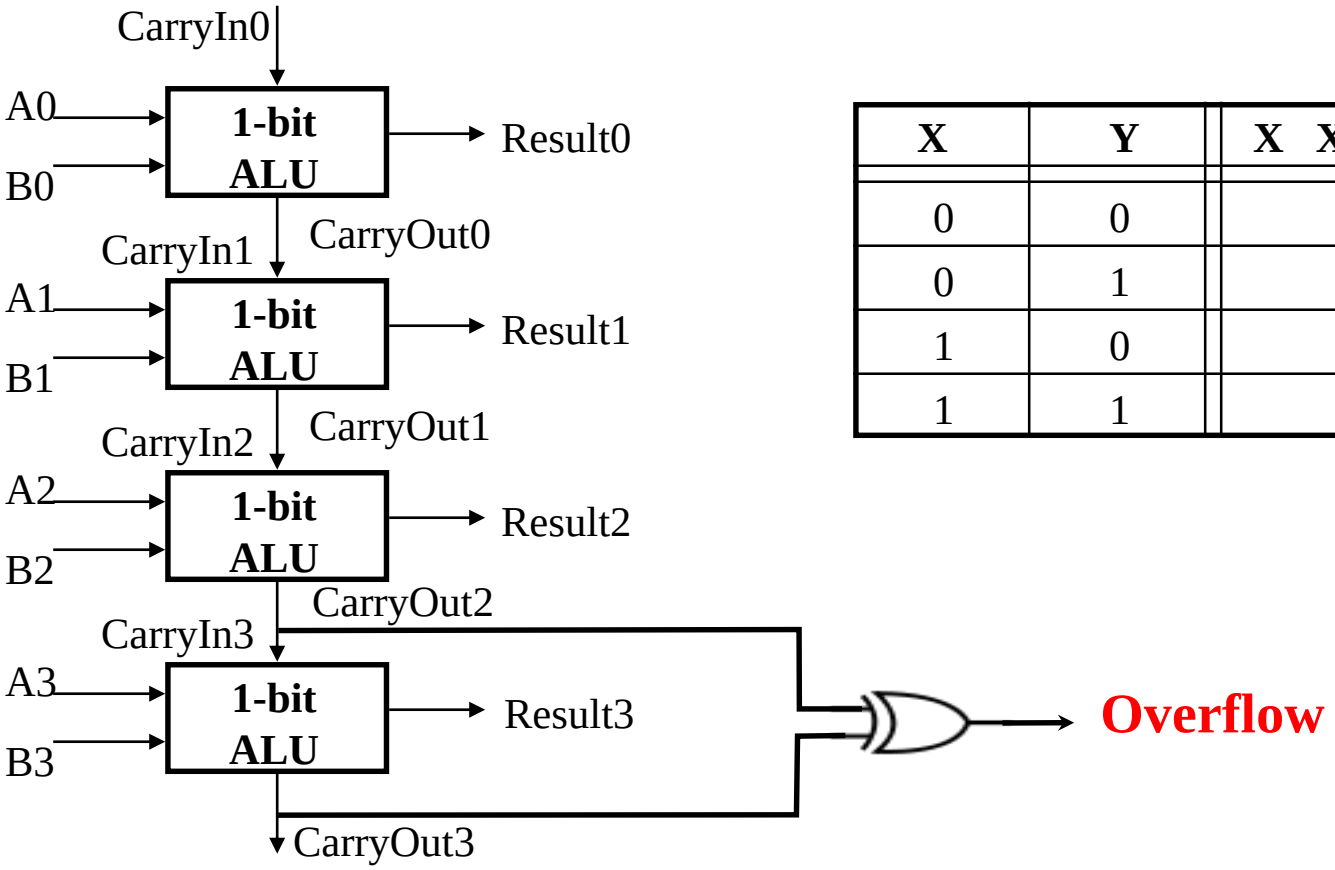




Overflow Detection Logic

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

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

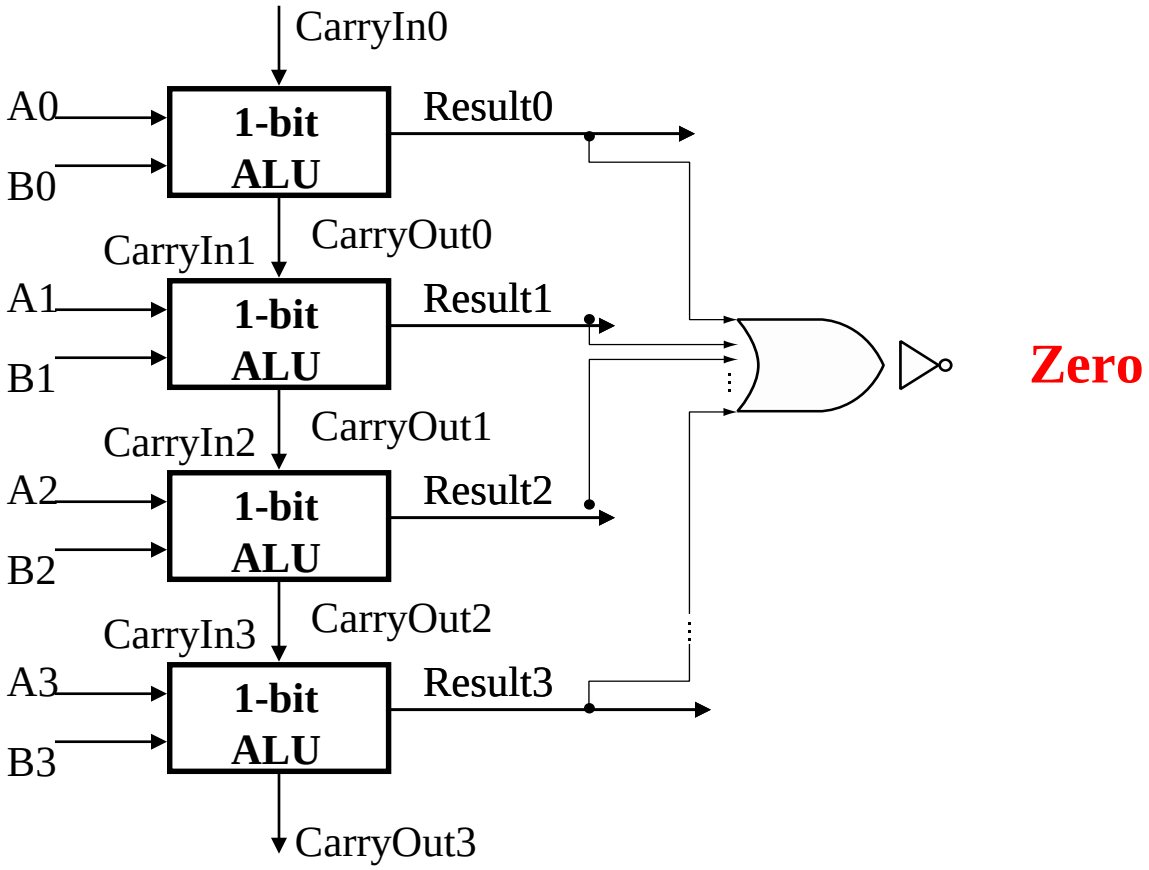


X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

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

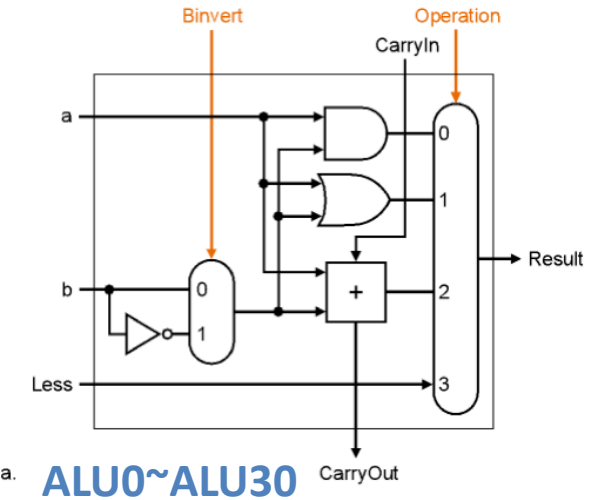


Set-on-less-than

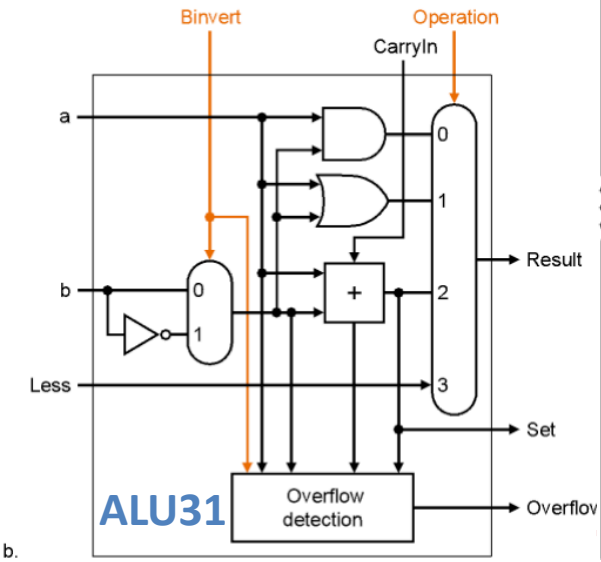
- **We are mostly there!**
  - A < B => (A - B) < 0
  - Do a subtract
- **If true, set LSB to 1, all others 0**
  - Use sign bit
    - route to bit 0 of result
    - all other bits zero
- **做减法的同时，若不溢出（Overflow=0），把ALU减法结果最高位符号位送至最低位ALU的Less端，其他位ALU的Less置零，最终输出所有的Less**
- **slt为有符号整数比较，最高符号位还需与Overflow进行异或操作才能供给最低位ALU的Less端**

Full ALU

要完成这几条指令时，ALU模块端口信号应该赋值成什么？

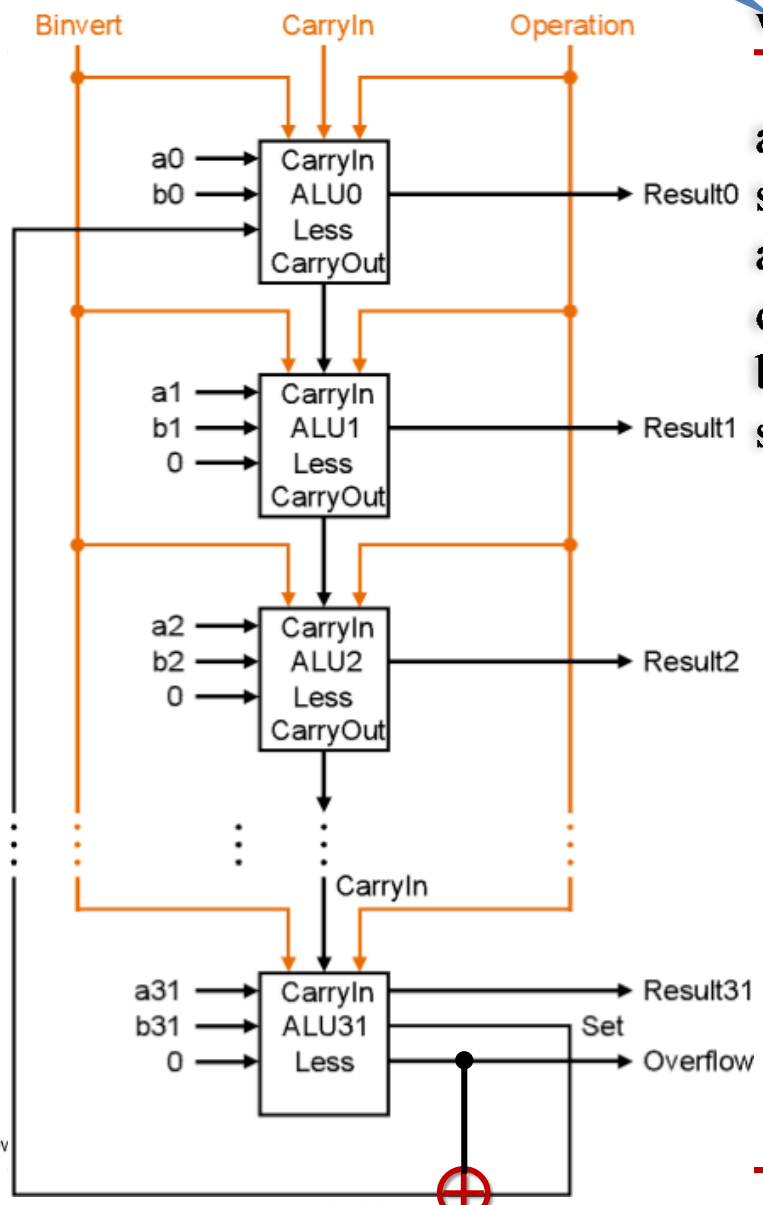


a. ALU0~ALU30



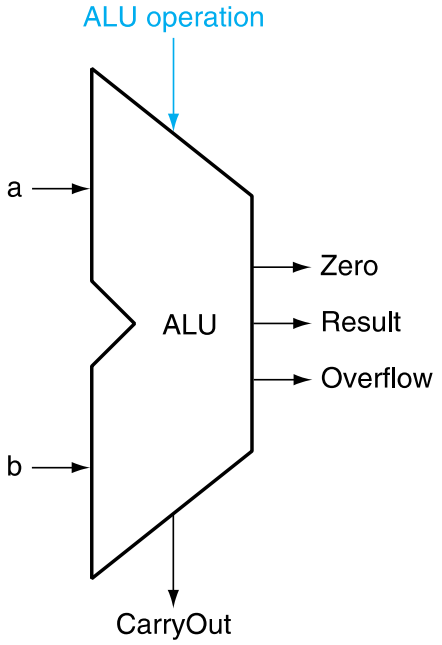
ALU31

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

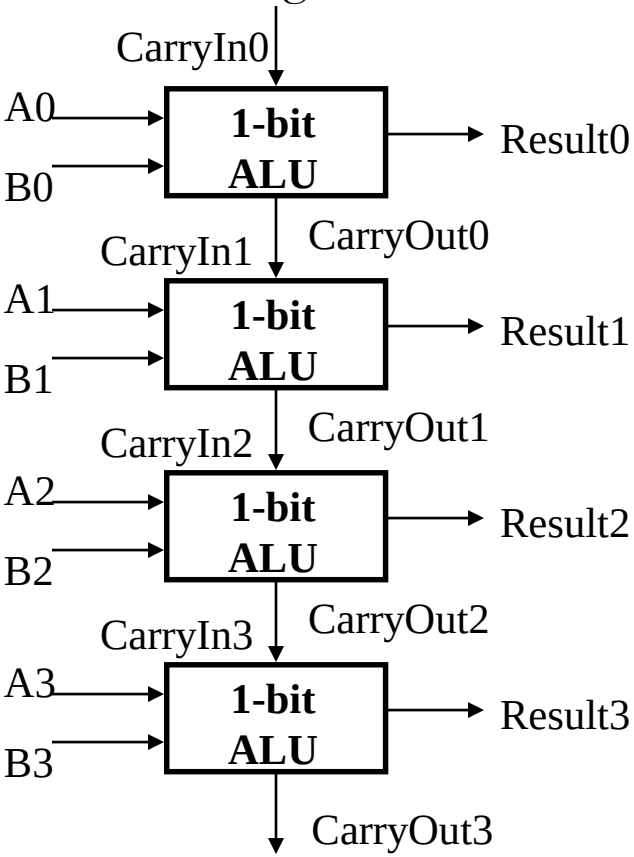


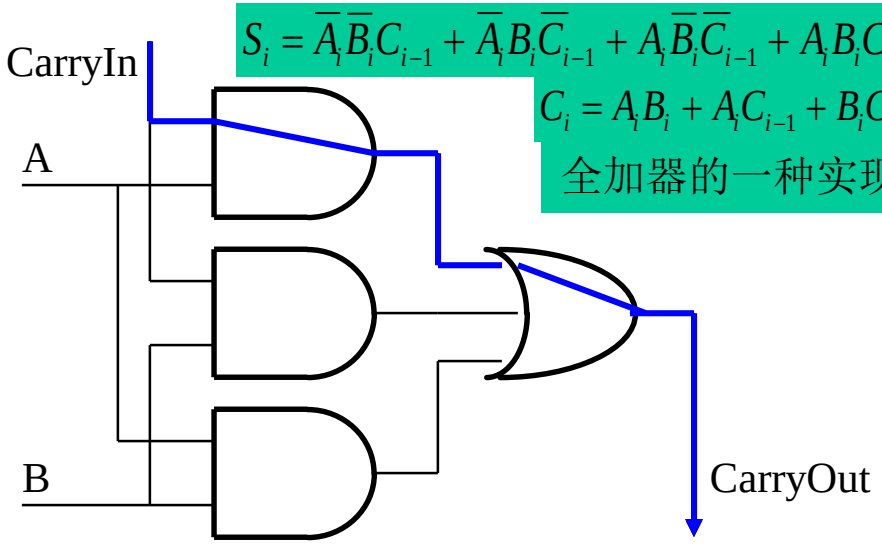
Two dedicated processor flags for carryout and overflow conditions

- https://en.wikipedia.org/wiki/Integer_overflow#Flags
- The carry (carryout) flag is set when the result of an addition or subtraction, considering the operands and result as unsigned numbers, does not fit in the given number of bits. This indicates an overflow with a carry or borrow from the most significant bit. An immediately following add with carry or subtract with borrow operation would use the contents of this flag to modify a register or a memory location that contains the higher part of a multi-word value.
- The overflow flag is set when the result of an operation on signed numbers does not have the sign that one would predict from the signs of the operands, e.g. a negative result when adding two positive numbers. This indicates that an overflow has occurred and the signed result represented in two's complement form would not fit in the given number of bits.

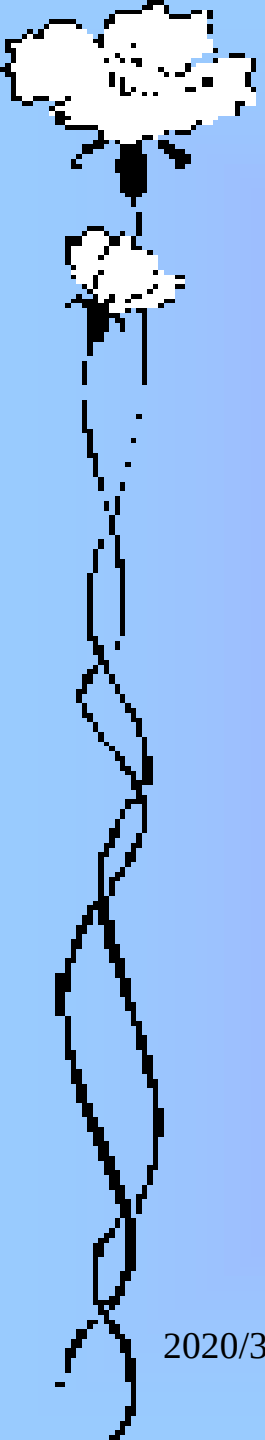
Ripple Carry Adder

- **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





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



附录：加法器

半加器和全加器

1、半加器

能对两个1位二进制数进行相加而求得和及进位的逻辑电路称为半加器.

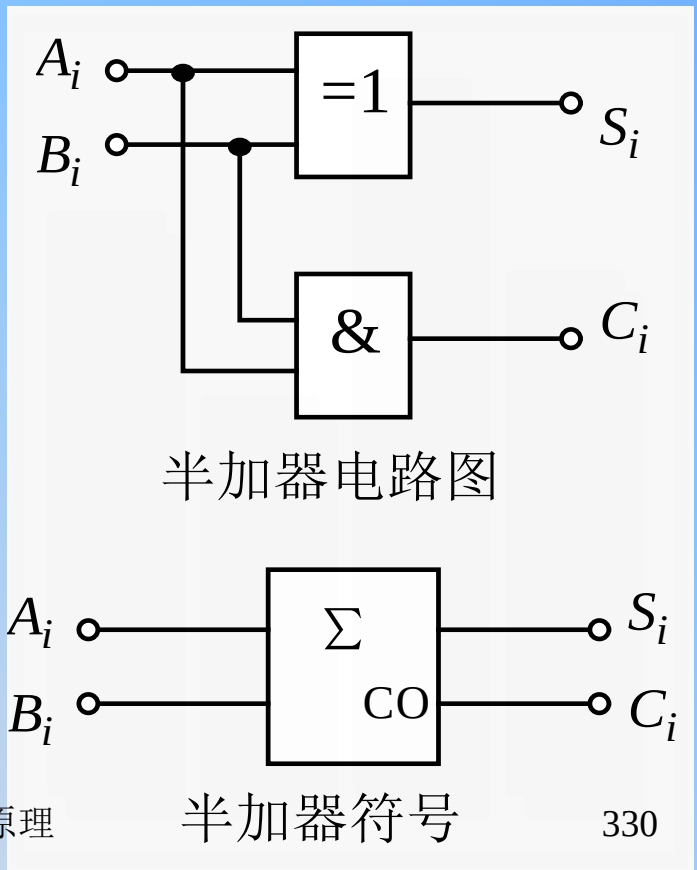
半加器真值表

A_i	B_i	S_i	C_i
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

加数　本位的和　向高位的进位

$$S_i = \overline{A}_i B_i + A_i \overline{B}_i = A_i \oplus B_i$$

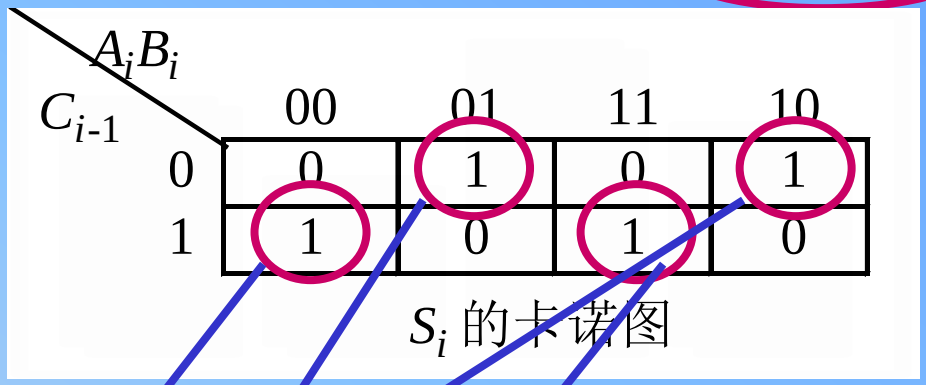
$$C_i = A_i B_i$$



2、全加器

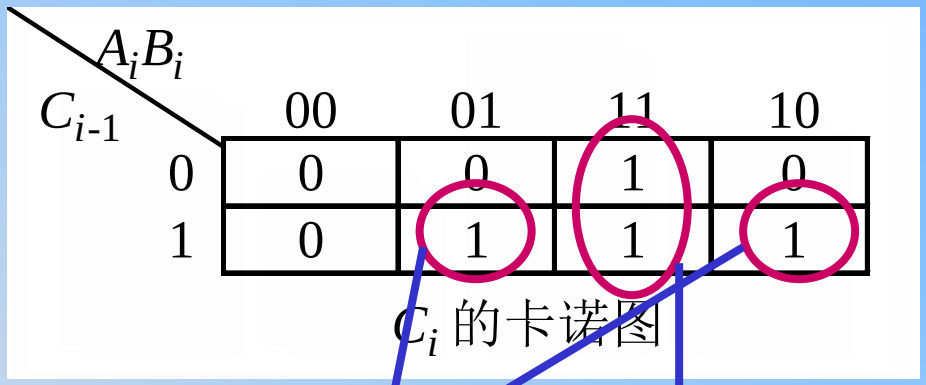
能对两个1位二进制数进行相加并考虑低位来的进位，即相当于3个1位二进制数相加，求得和及进位的逻辑电路称为全加器。

A_i	B_i	C_{i-1}	S_i	C_i
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



S_i 的卡诺图

$$S_i = m_1 + m_2 + m_4 + m_7 = A_i \oplus B_i \oplus C_{i-1}$$



C_i 的卡诺图

$$C_i = m_3 + m_5 + A_iB_i$$
$$= (A_i \oplus B_i)C_{i-1} + A_iB_i$$

A_i、B_i：加数，C_{i-1}：低位来的进位，S_i：本位的和，C_i：向高位的进位。

全加器的逻辑图和逻辑符号

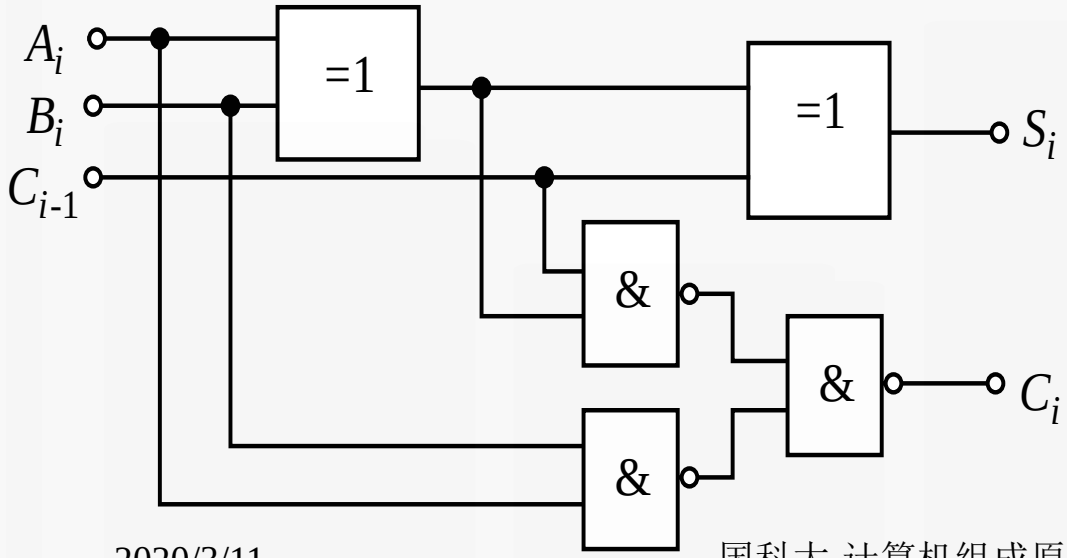
$$S_i = m_1 + m_2 + m_4 + m_7 = \overline{A}_i\overline{B}_iC_{i-1} + \overline{A}_iB_i\overline{C}_{i-1} + A_i\overline{B}_i\overline{C}_{i-1} + A_iB_iC_{i-1}$$

$$= \overline{A}_i(\overline{B}_iC_{i-1} + B_i\overline{C}_{i-1}) + A_i(\overline{B}_i\overline{C}_{i-1} + B_iC_{i-1}) = \overline{A}_i(B_i \oplus C_{i-1}) + A_i\overline{(B_i \oplus C_{i-1})}$$

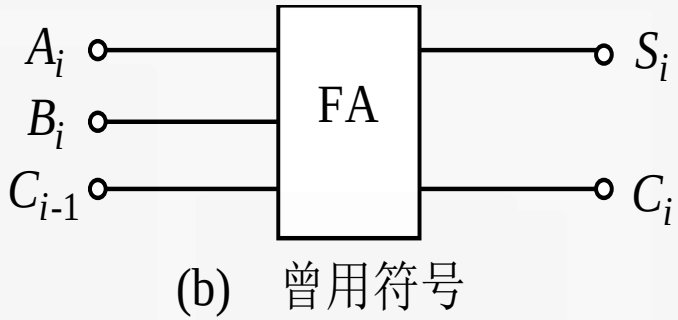
$$= A_i \oplus B_i \oplus C_{i-1}$$

$$C_i = m_3 + m_5 + A_iB_i = \overline{A}_iB_iC_{i-1} + A_i\overline{B}_iC_{i-1} + A_iB_i = (\overline{A}_iB_i + A_i\overline{B}_i)C_{i-1} + A_iB_i$$

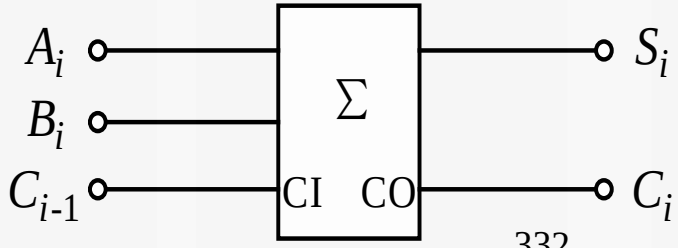
$$= (A_i \oplus B_i)C_{i-1} + A_iB_i$$



(a) 逻辑图



(b) 曾用符号

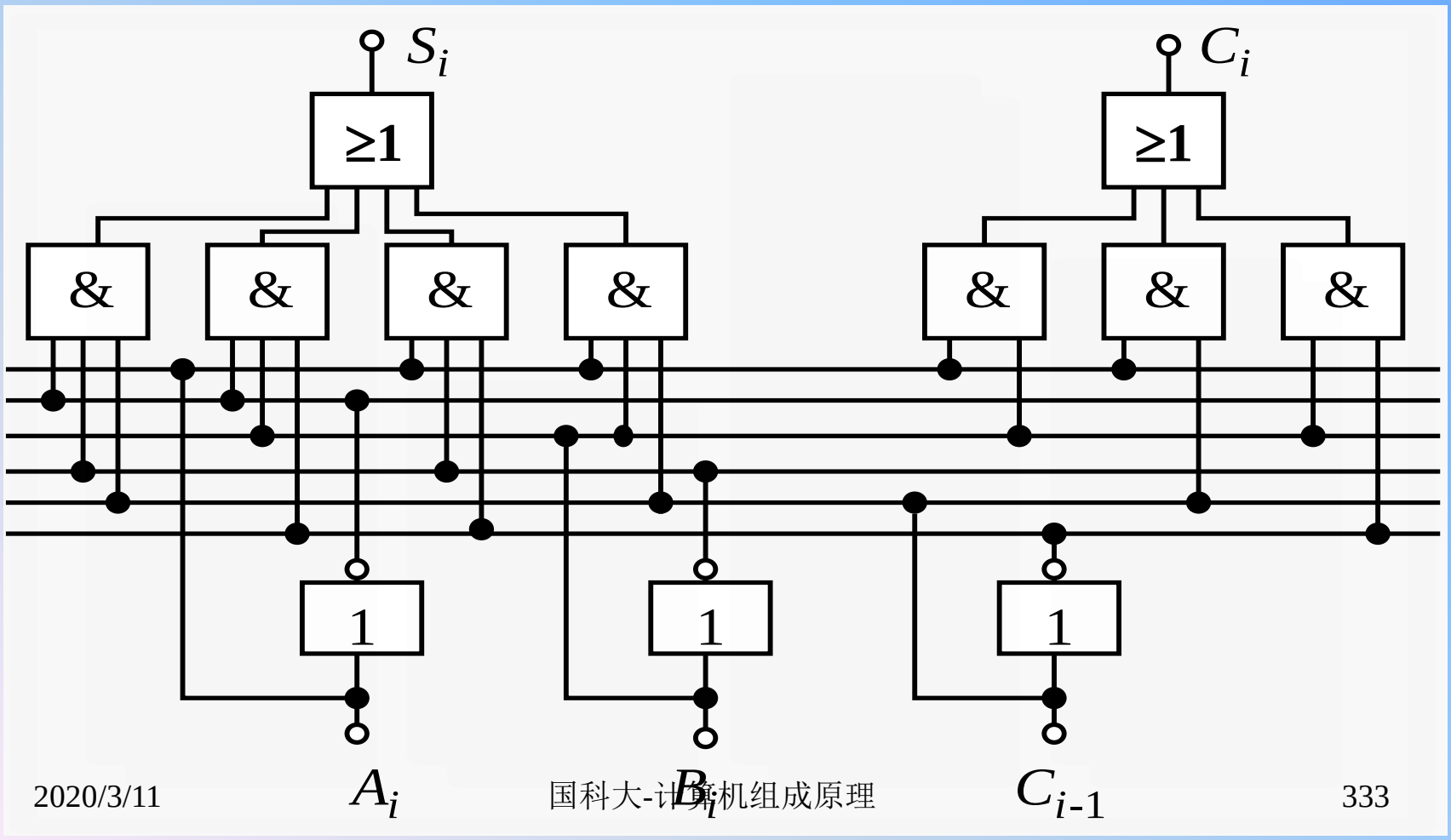


(c) 国标符号

用与门和或门实现

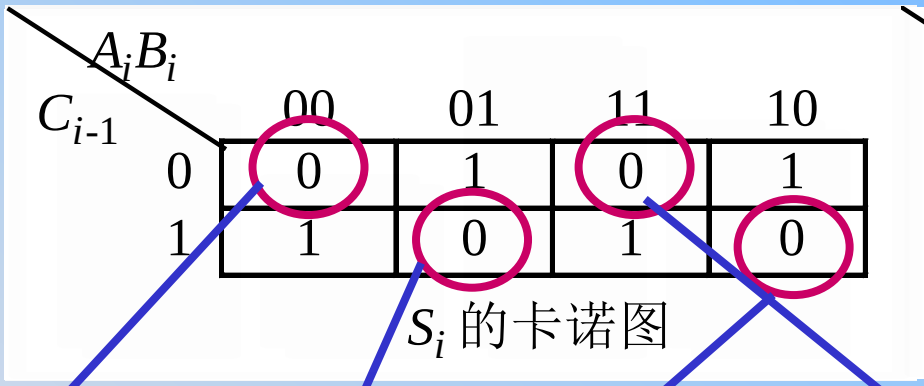
$$S_i = \overline{A}_i\overline{B}_iC_{i-1} + \overline{A}_iB_i\overline{C}_{i-1} + A_i\overline{B}_i\overline{C}_{i-1} + A_iB_iC_{i-1}$$

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

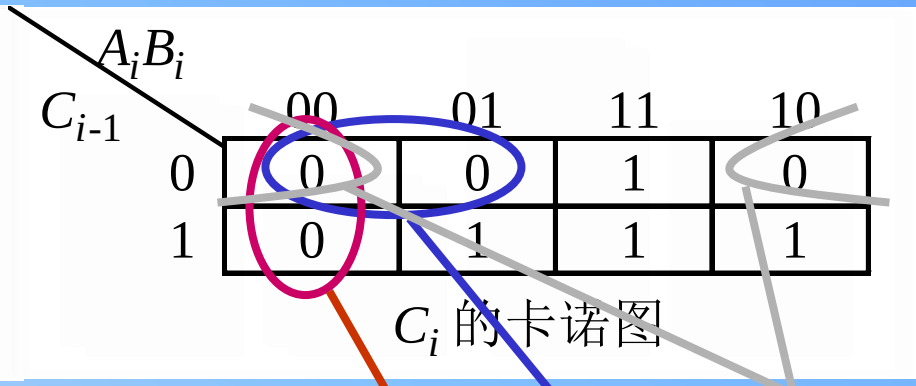


用与或非门实现

先求$\overline{S_i}$和$\overline{C_i}$。为此，合并值为0的最小项。



S_i 的卡诺图



C_i 的卡诺图

$$\overline{S_i} = \overline{A_i}\,\overline{B_i}\,\overline{C_{i-1}} + \overline{A_i}B_iC_{i-1} + A_i\overline{B_i}C_{i-1} + A_iB_i\overline{C_{i-1}}$$

$$\overline{C_i} = \overline{A_i}\,\overline{B_i} + \overline{A_i}\,\overline{C_{i-1}} + \overline{B_i}\,\overline{C_{i-1}}$$

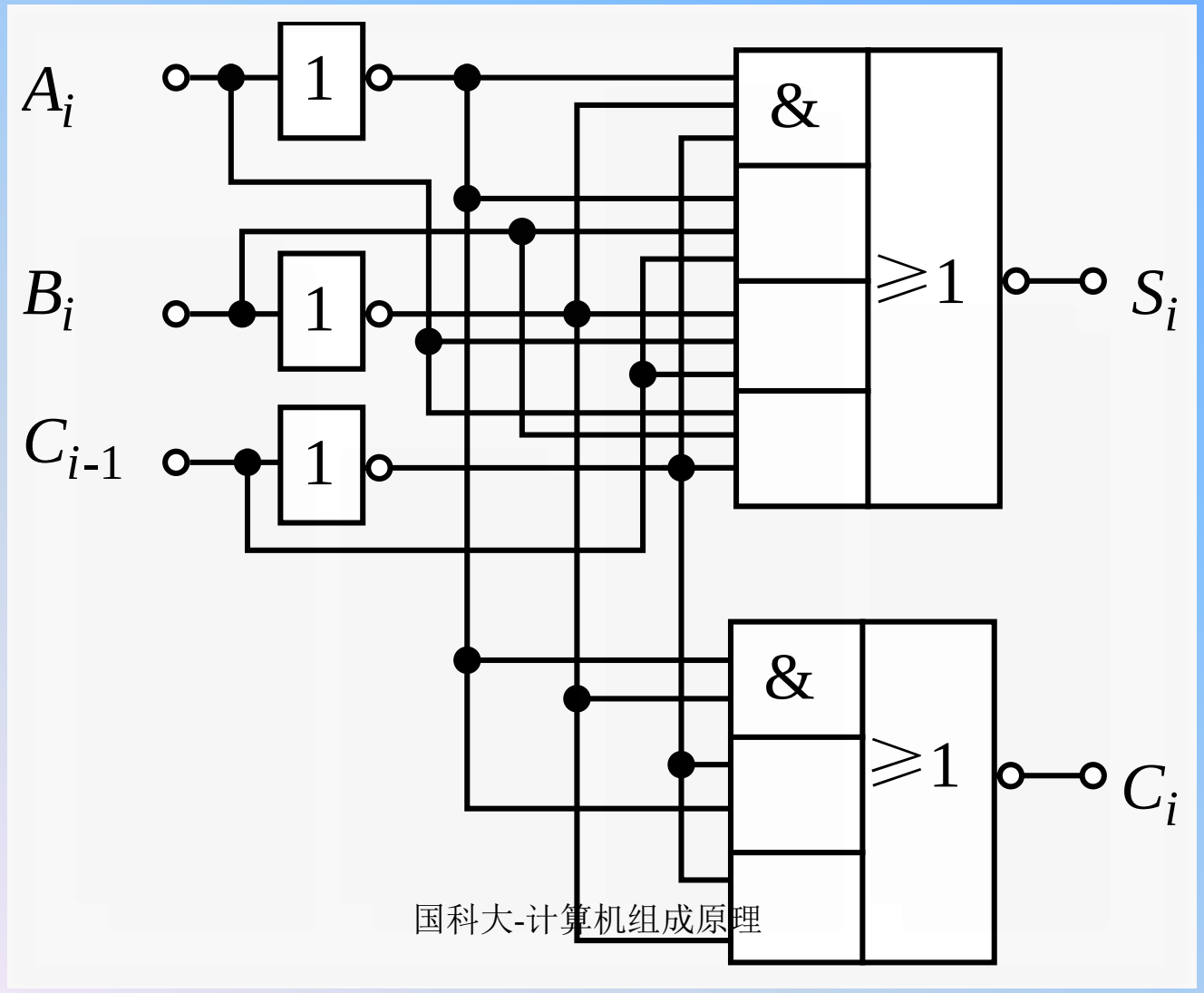
再取反，得：

$$\overline{S_i} = \overline{\overline{S_i}} = \overline{\overline{A_i}\,\overline{B_i}\,\overline{C_{i-1}} + \overline{A_i}B_iC_{i-1} + A_i\overline{B_i}C_{i-1} + A_iB_i\overline{C_{i-1}}}$$

$$C_i = \overline{\overline{C_i}} = \overline{\overline{A_i}\,\overline{B_i} + \overline{A_i}\,\overline{C_{i-1}} + \overline{B_i}\,\overline{C_{i-1}}}$$

$$\overline{S_i} = \overline{\overline{A_i}\,\overline{B_i}\,\overline{C_{i-1}} + \overline{A_i}\,B_i\,C_{i-1} + A_i\,\overline{B_i}\,C_{i-1} + A_i\,B_i\,\overline{C_{i-1}}}$$

$$C_i = \overline{\overline{A_i}\,\overline{B_i} + \overline{A_i}\,\overline{C_{i-1}} + \overline{B_i}\,\overline{C_{i-1}}}$$

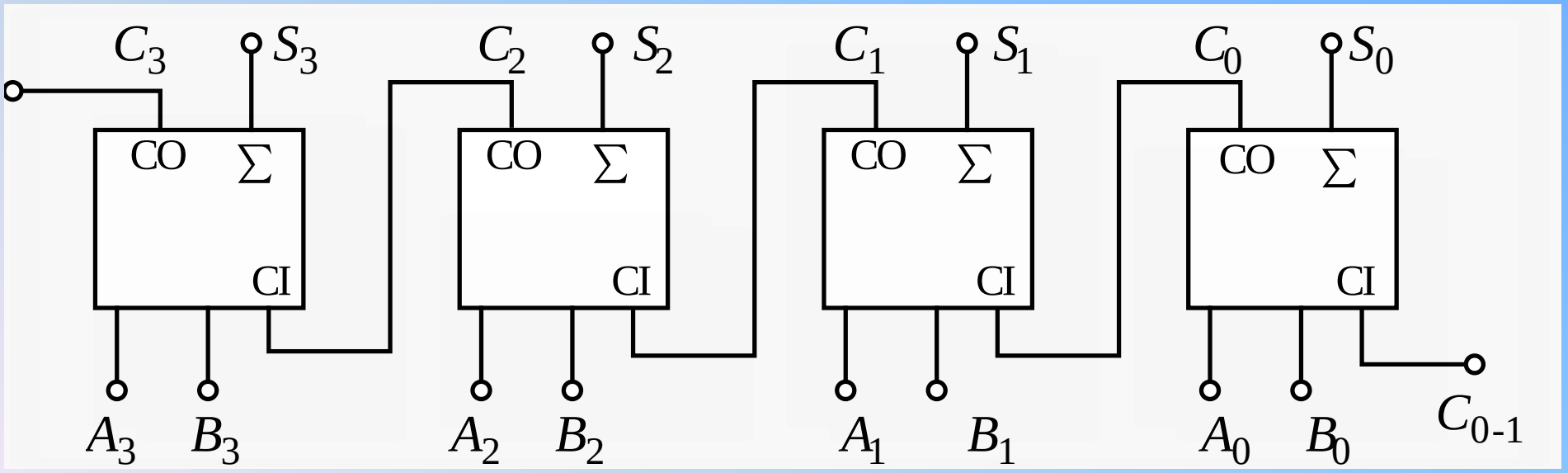


加法器

实现多位二进制数相加的电路称为加法器。

1、串行进位加法器

构成：把n位全加器串联起来，低位全加器的进位输出连接到相邻的高位全加器的进位输入。



特点：进位信号是由低位向高位逐级传递的，速度不高。

2、并行进位加法器（超前进位加法器）

进位生成项	$G_i = A_i B_i$	进位传递条件	$P_i = A_i \oplus B_i$
进位表达式	$C_i = A_i B_i + (A_i \oplus B_i) C_{i-1} = G_i + P_i C_{i-1}$		
和表达式	$S_i = A_i \oplus B_i \oplus C_{i-1} = P_i \oplus C_{i-1}$		

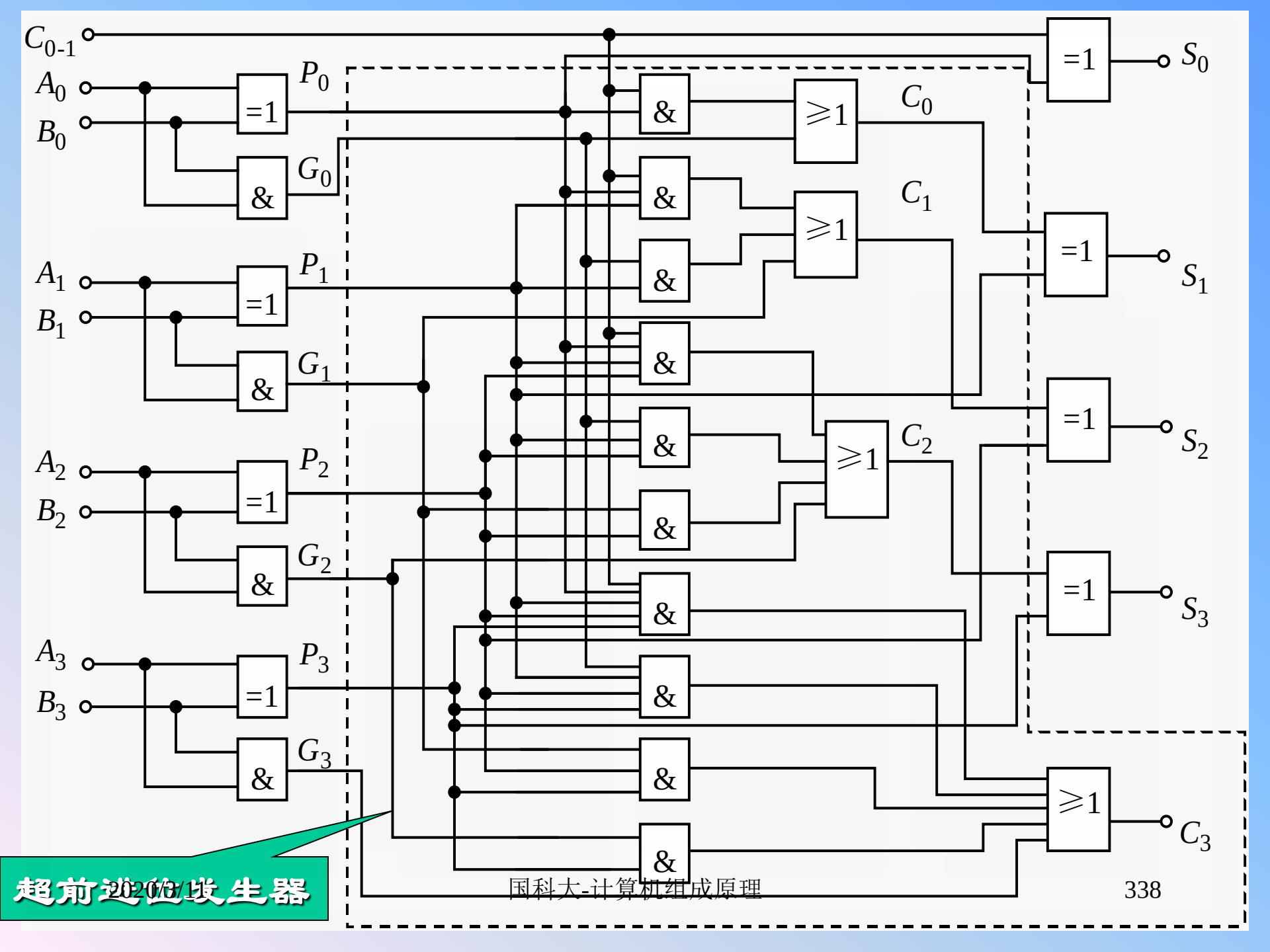
$$\begin{cases} S_0 = P_0 \oplus C_{0-1} \\ C_0 = G_0 + P_0 C_{0-1} \end{cases}$$

4位超前进位加法器递推公式

$$\begin{cases} S_1 = P_1 \oplus C_0 \\ C_1 = G_1 + P_1 C_0 = G_1 + P_1 G_0 + P_1 P_0 C_{0-1} \end{cases}$$

$$\begin{cases} S_2 = P_2 \oplus C_1 \\ C_2 = G_2 + P_2 C_1 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{0-1} \end{cases}$$

$$\begin{cases} S_3 = P_3 \oplus C_2 \\ C_3 = G_3 + P_3 C_2 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{0-1} \end{cases}$$


C_{0-1}
A_0
B_0
=1
P_0
&
G_0
A_1
B_1
=1
P_1
&
G_1
A_2
B_2
=1
P_2
&
G_2
A_3
B_3
=1
P_3
&
G_3
&
⩾1
C_0
C_1
C_2
C_3
=1
S_0
S_1
S_2
S_3
超前进位发生器