Computer Architecture

CH3 Computer Arithmetic (I)

Prof. Ren-Shuo Liu

NTHU EE

Fall 2017

Applications

Data Structures / Algorithms

Programming

Software

Hardware

CH2 — Instruction Set Archtecture

CH3, 4, 5 — Organization & Architecture

} Computer Architecture

Logic Design

Digital Electronics

Solid-State Electronics

Outline

- Overview
- Integer operations
  - Bit-wise logical operations
  - Additions, subtractions
  - Comparisons
  - Multiplications
  - Divisions
- Floating point operations
  - Additions, subtractions
  - Multiplications

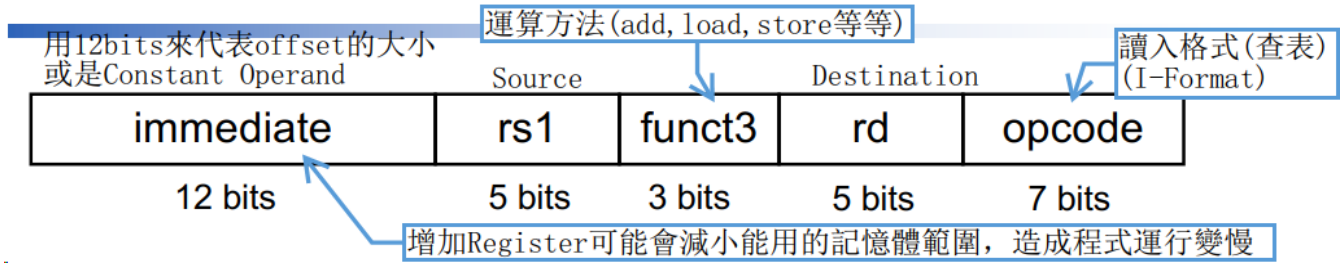
Outline

- **Overview**
  - Arithmetic logic unit (ALU)
  - Integer ~~Inter~~ instructions
  - Logic gates
  - Value representation and interpretation
- Integer operations
- Floating point operations

Common Integer Instructions

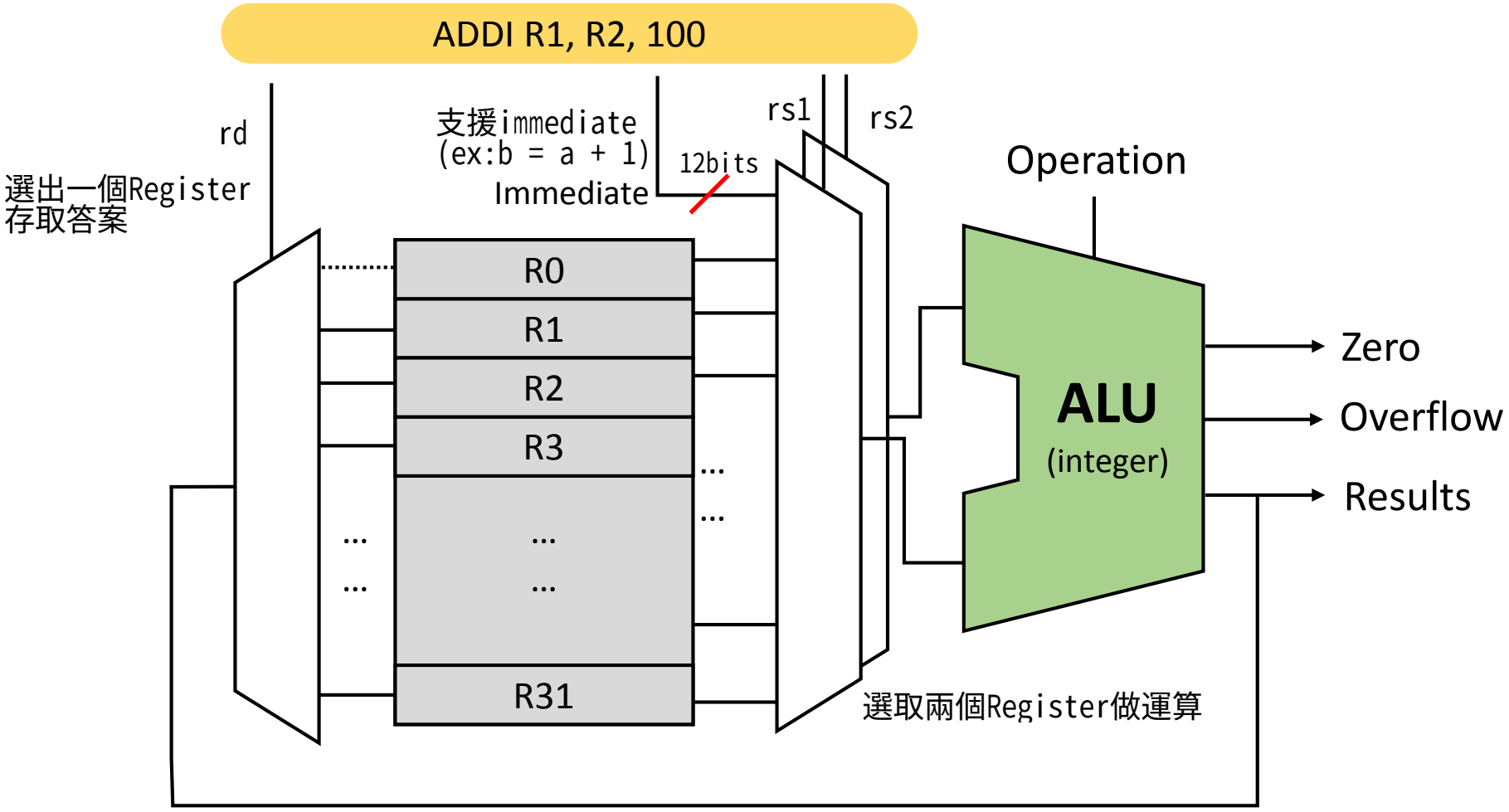
- Bit-wise logical
  - and, andi
  - or, ori
  - xor, xori
  - nor
- Arithmetic
  - add, addi
  - sub
  - mult
  - div
- Comparisons
  - slt, slti, sltu, sltiu

and: 做bitwise的AND運算
andi: 做bitwise AND運算，其中一項直接賦值，而不用存到Register裡面(Ex: a = b & 0xff)

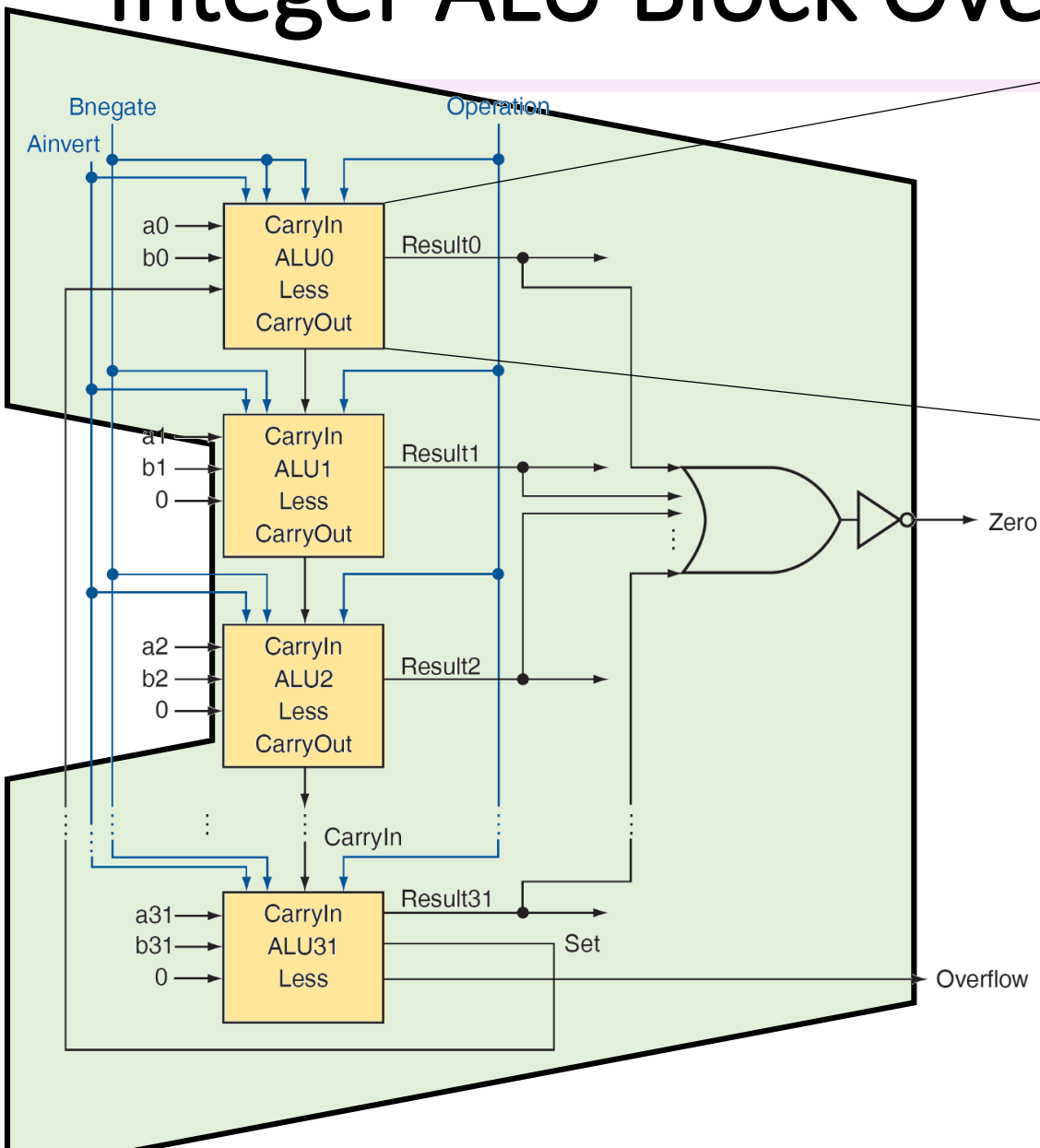


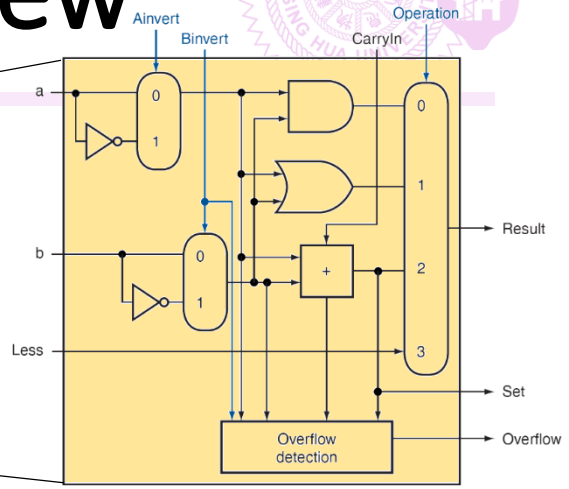
兩個unsigned數字做比較

Arithmetic Logic Unit (ALU)

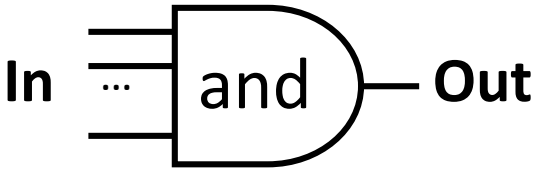


Integer ALU Block Overview

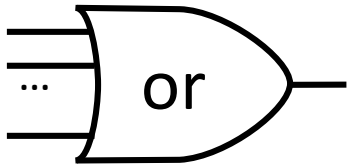




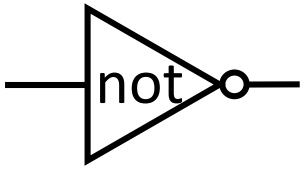
Logic Gates



In	Out
all 1's	1
others	0



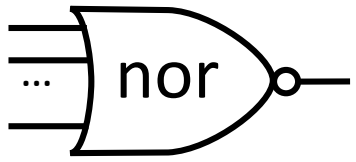
In	Out
all 0's	0
others	1



In	Out
0	1
1	0

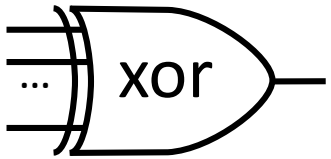
Logic Gates (Cont'd)

OR Gate + NOR Gate



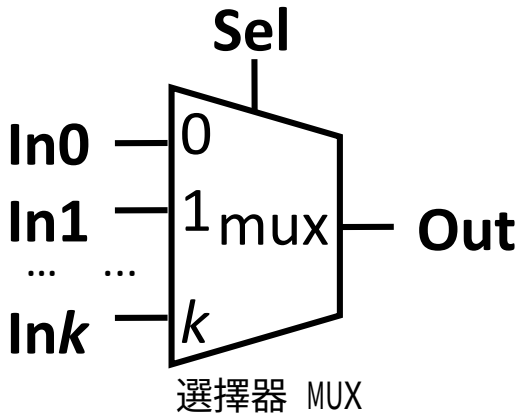
In	Out
all 0's	1
others	0

Exclusive OR



奇數個1輸出為1

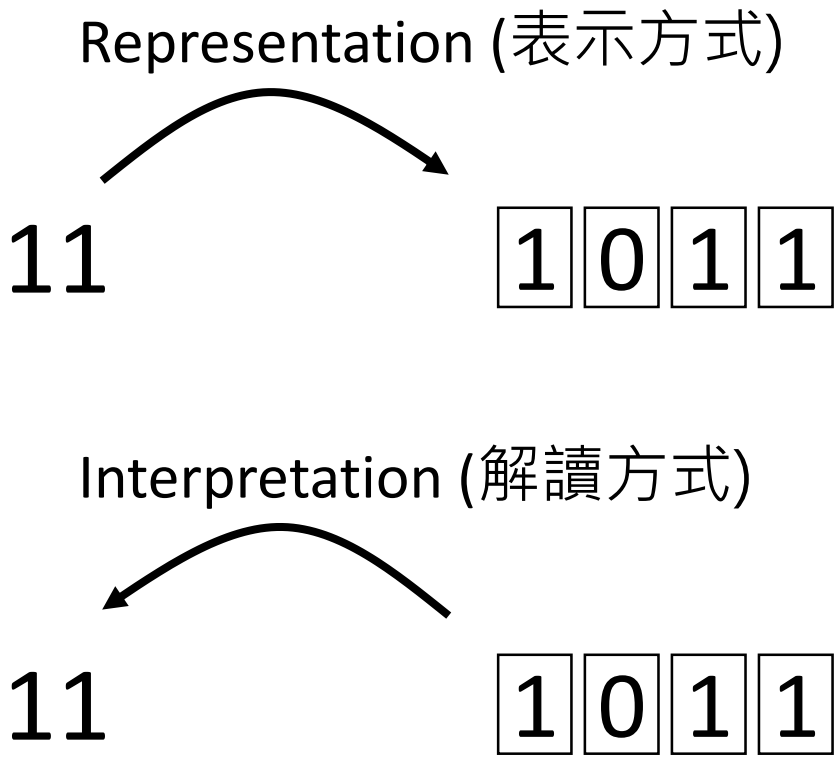
In	Out
odd num. of 1's	1
others	0



選擇器 MUX

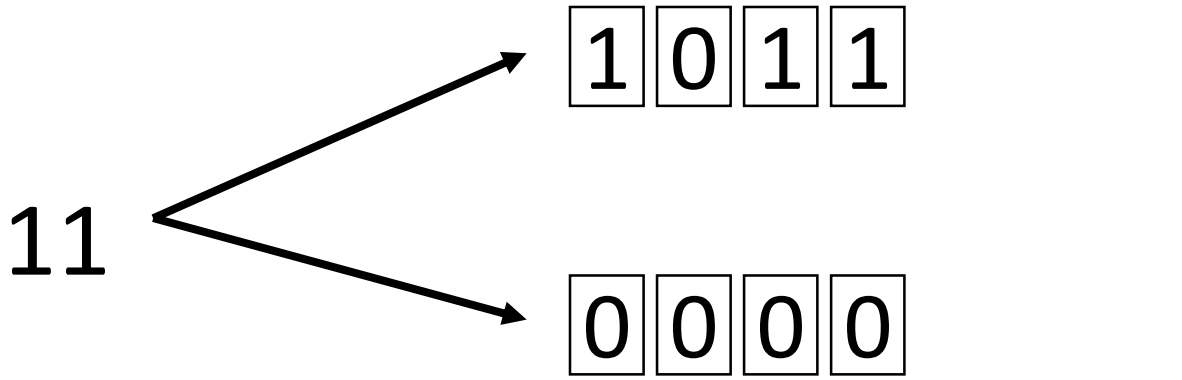
Sel	Out
0	In0
1	In1
…	…
k	In*k*

Value Representation and Interpretation



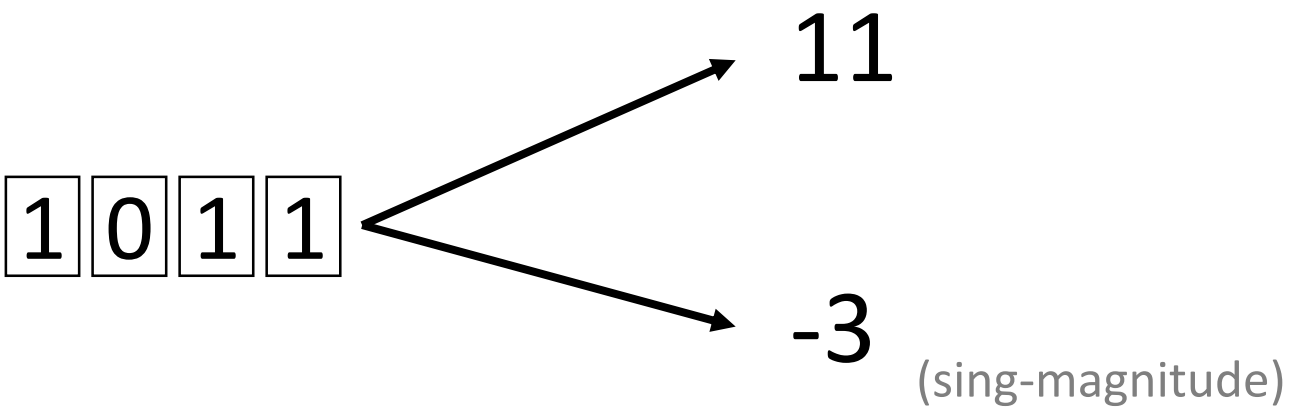
Value Representation and Interpretation

Two representation strategies (兩種表示方式)

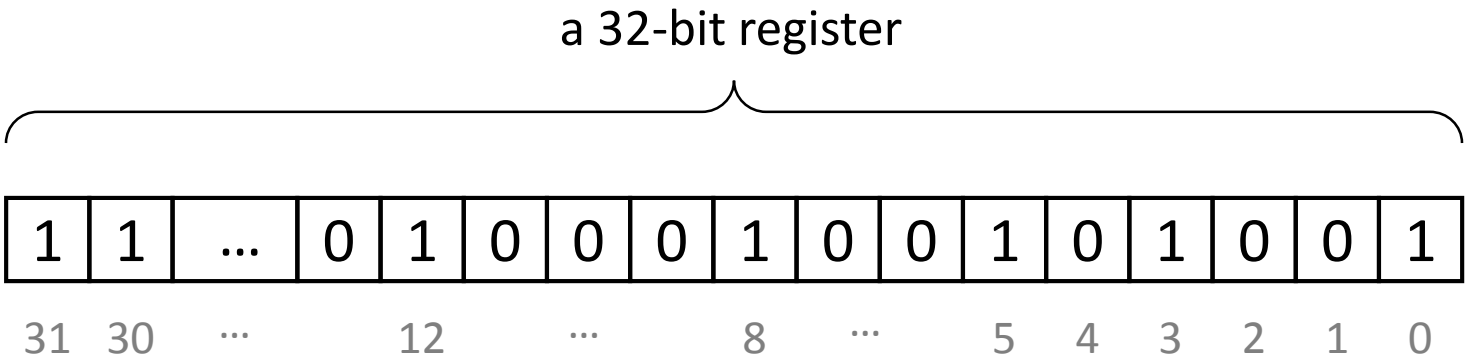


Value Representation and Interpretation

Two interpretation strategies (兩種解讀方式)



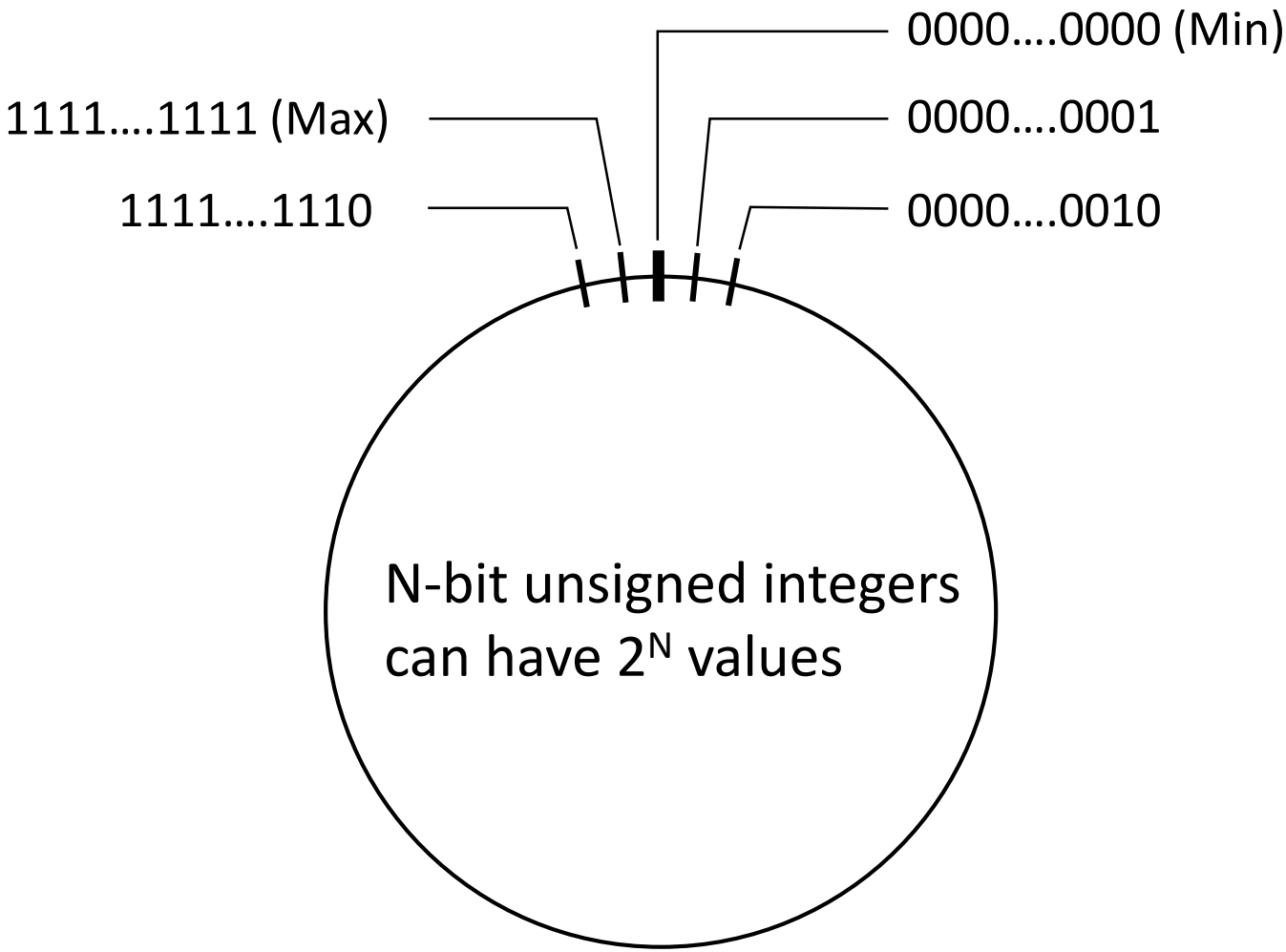
Binary Unsigned Integers



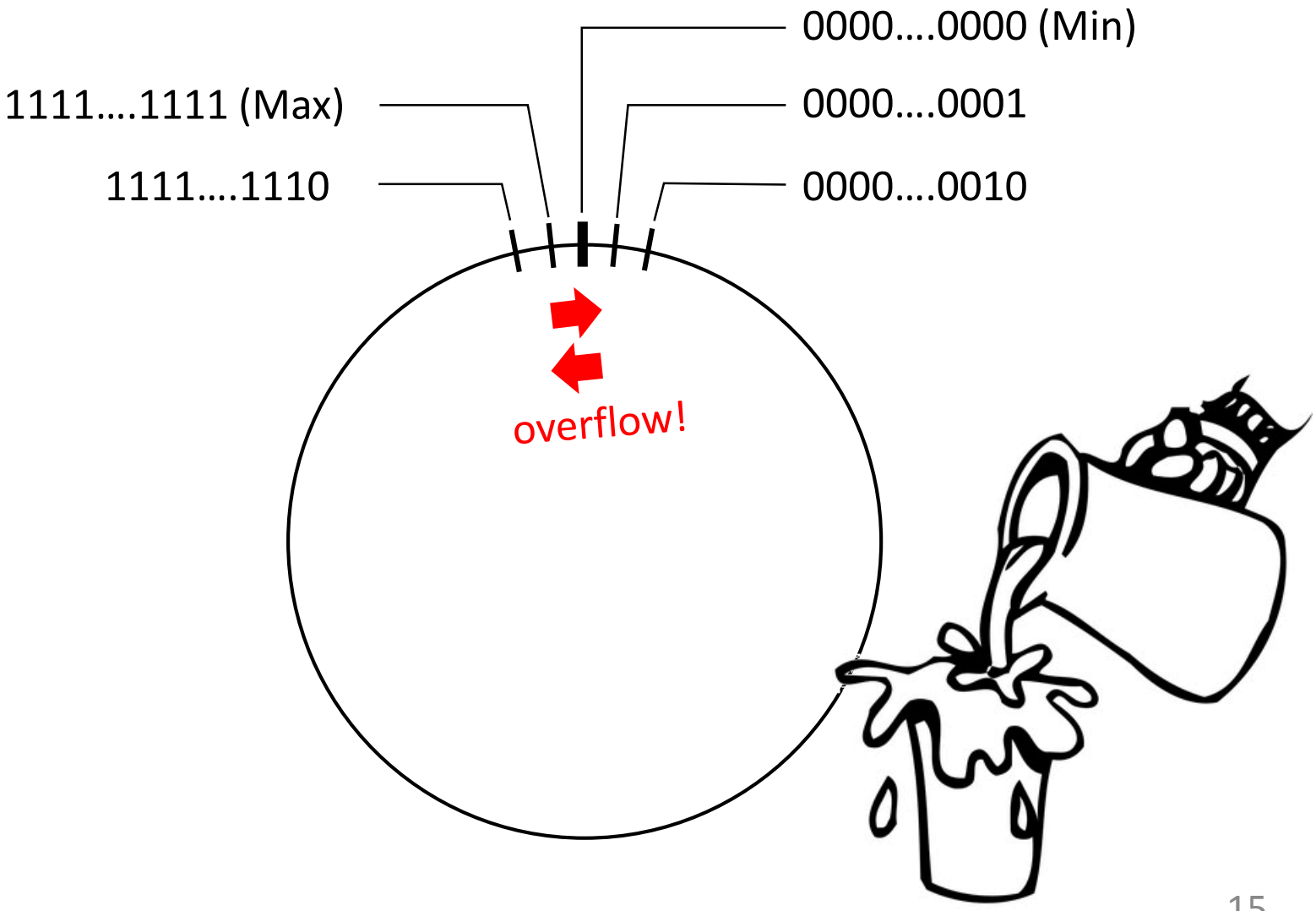
- Above is interpreted as $(2^{31}+2^{30}+2^{12}+2^{8}+2^{5}+2^{3}+2^{0})$
- Range
  - Min: 0
  - Max: ~~$2^{33}-1$~~

    2^32-1

Unsigned Integers (Cont'd)



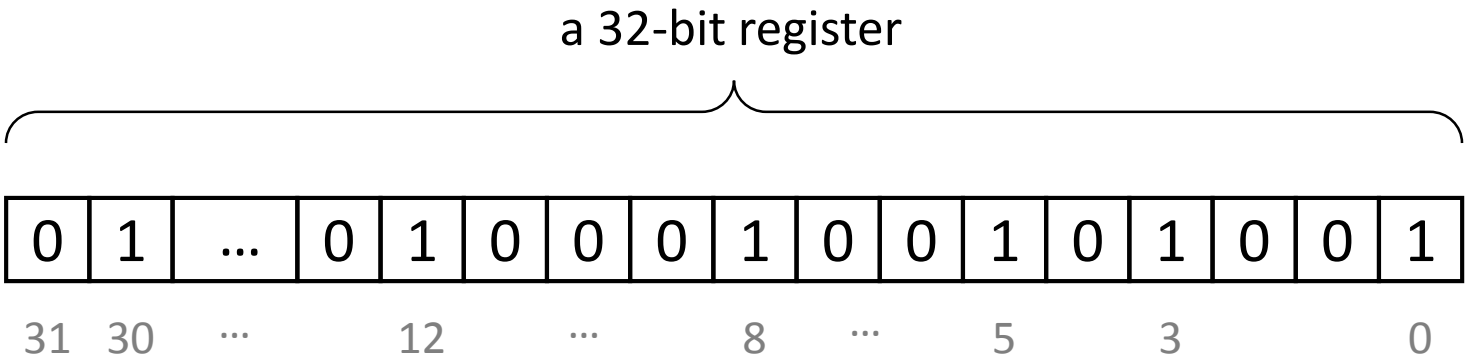
Unsigned Integers (Cont'd)



Signed Integers Signed(有正負)

- Many widely-used representations
  - Biased/offset/excess-K
  - Signed magnitude
  - Two's complement

Biased / Offset / Excess-K

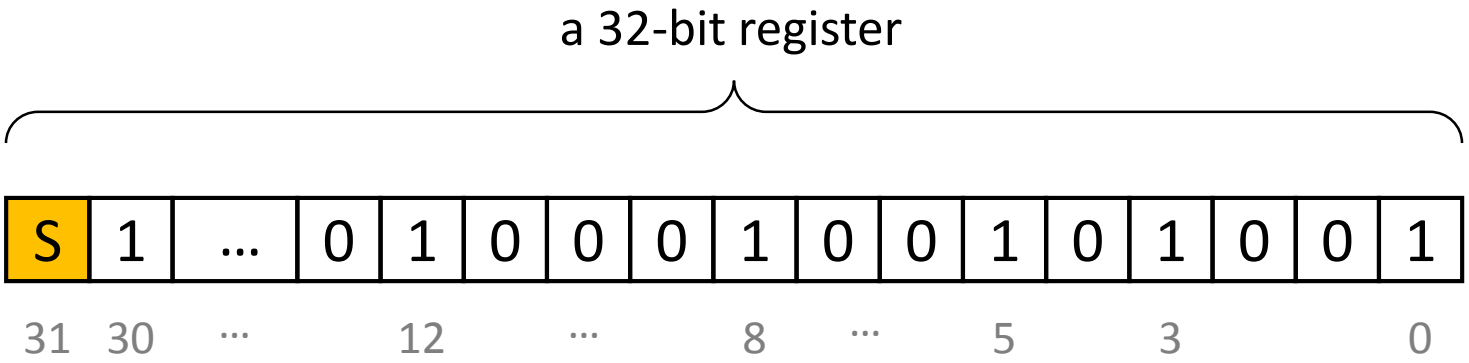


- Above is interpreted as
  - $(2^{30}+2^{12}+2^{8}+2^{5}+2^{3}+2^{0})$ - K
    - Where K is a predefined value

先減去K

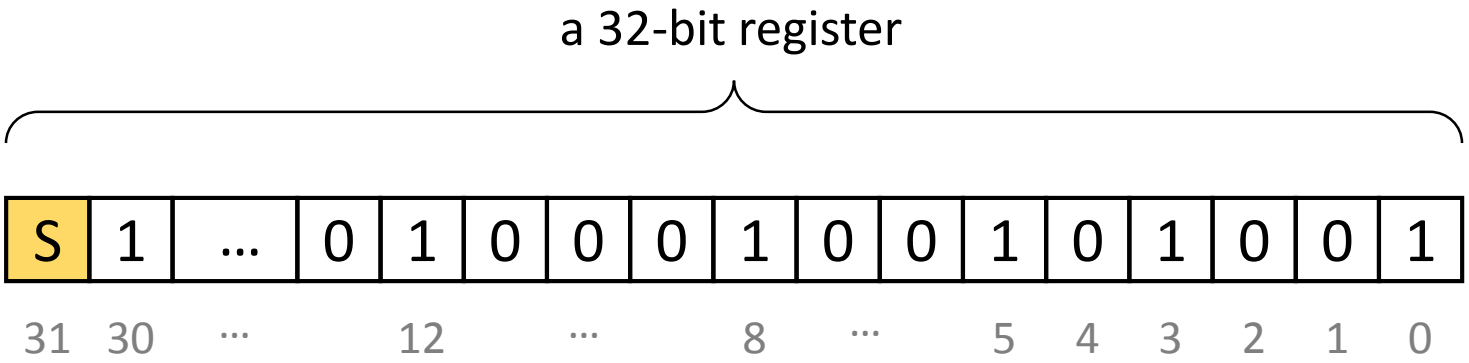
- Range
  - Most negative: -K
  - Most positive: 2^{32} - K - 1

Sign Magnitude Representation



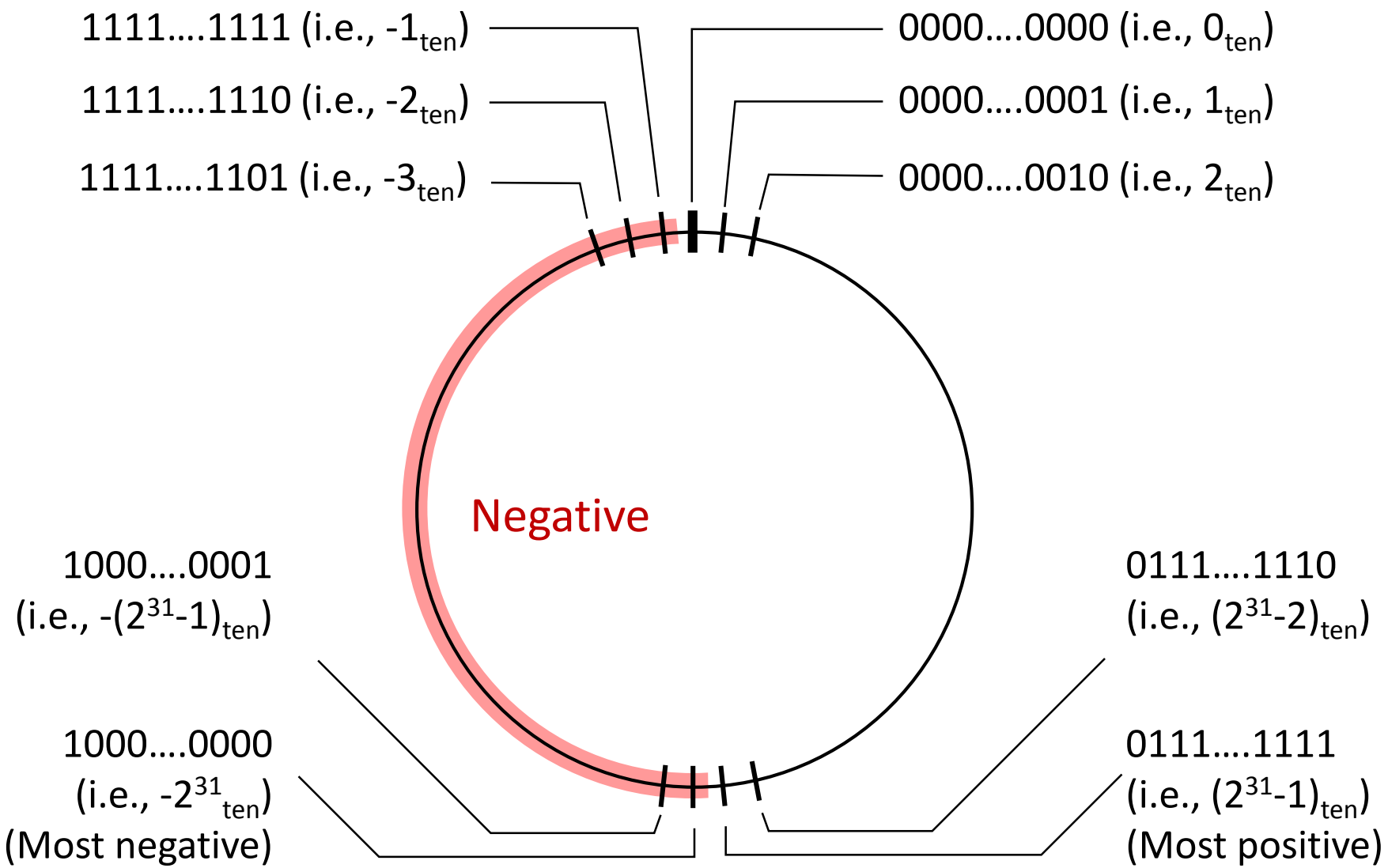
- Above is interpreted as
  - S × $(2^{30}+2^{12}+2^{8}+2^{5}+2^{3}+2^{0})$
    - $(2^{30}+2^{12}+2^{8}+2^{5}+2^{3}+2^{0})$ if sign == 0 (i.e., positive)
    - $-(2^{31}-(2^{30}+2^{12}+2^{8}+2^{5}+2^{3}+2^{0}))$ if sign == 1 (i.e., negative)
- Range
  - Most negative: $-2^{31}+1$
  - Most positive: $2^{31}-1$

2's Compliment Signed Integers

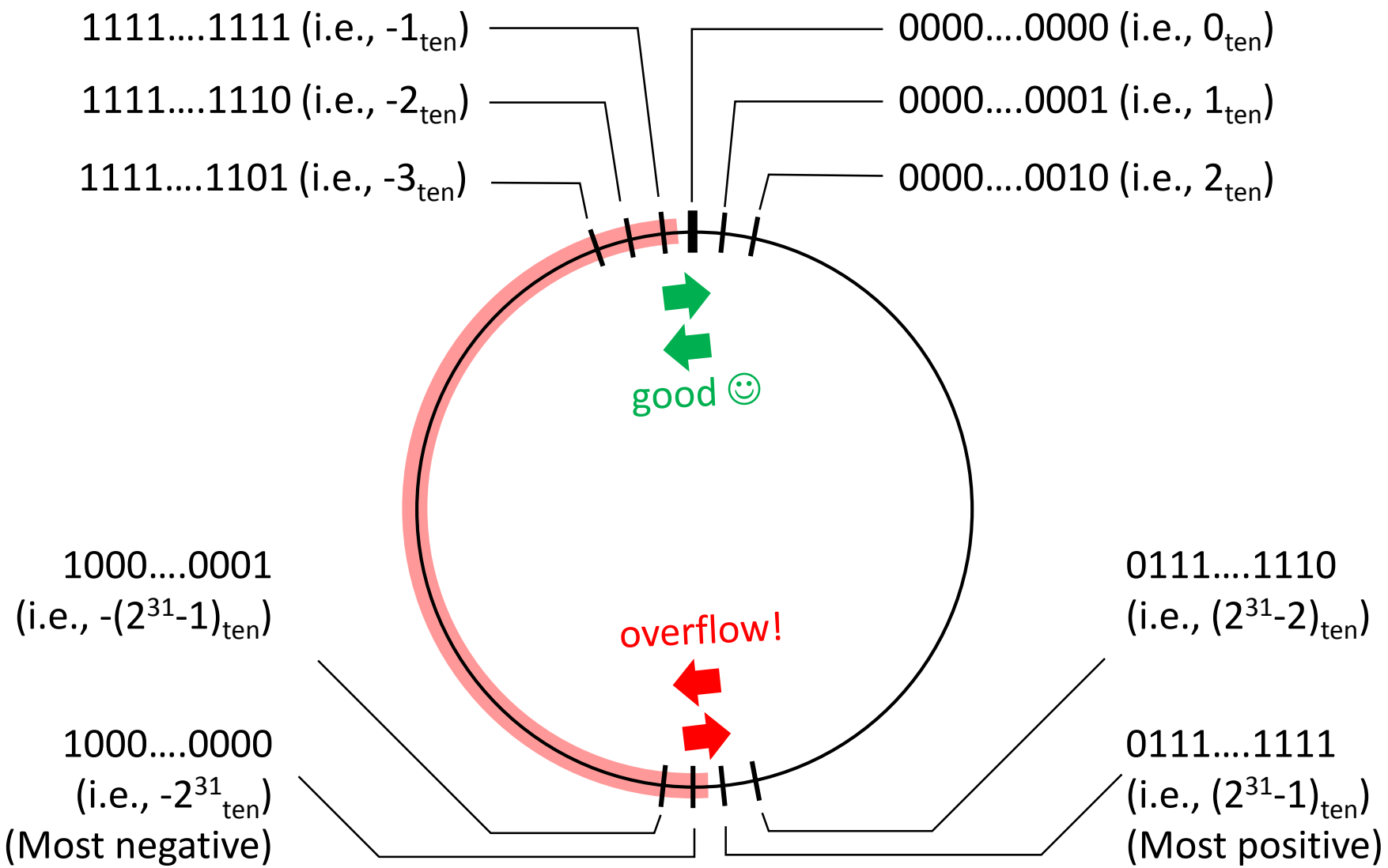


- Above is interpreted as
  - $S \times (-2^{31}) + (2^{30}+2^{12}+2^{8}+2^{5}+2^{3}+2^{0})$
    - $(2^{30}+2^{12}+2^{8}+2^{5}+2^{3}+2^{0})$ if sign == 0 (i.e., positive)
    - $-(2^{31}-(2^{30}+2^{12}+2^{8}+2^{5}+2^{3}+2^{0}))$ if sign == 1 (i.e., negative)
- Range
  - Most negative: $-(2^{31})$
  - Most positive: $2^{31}-1$

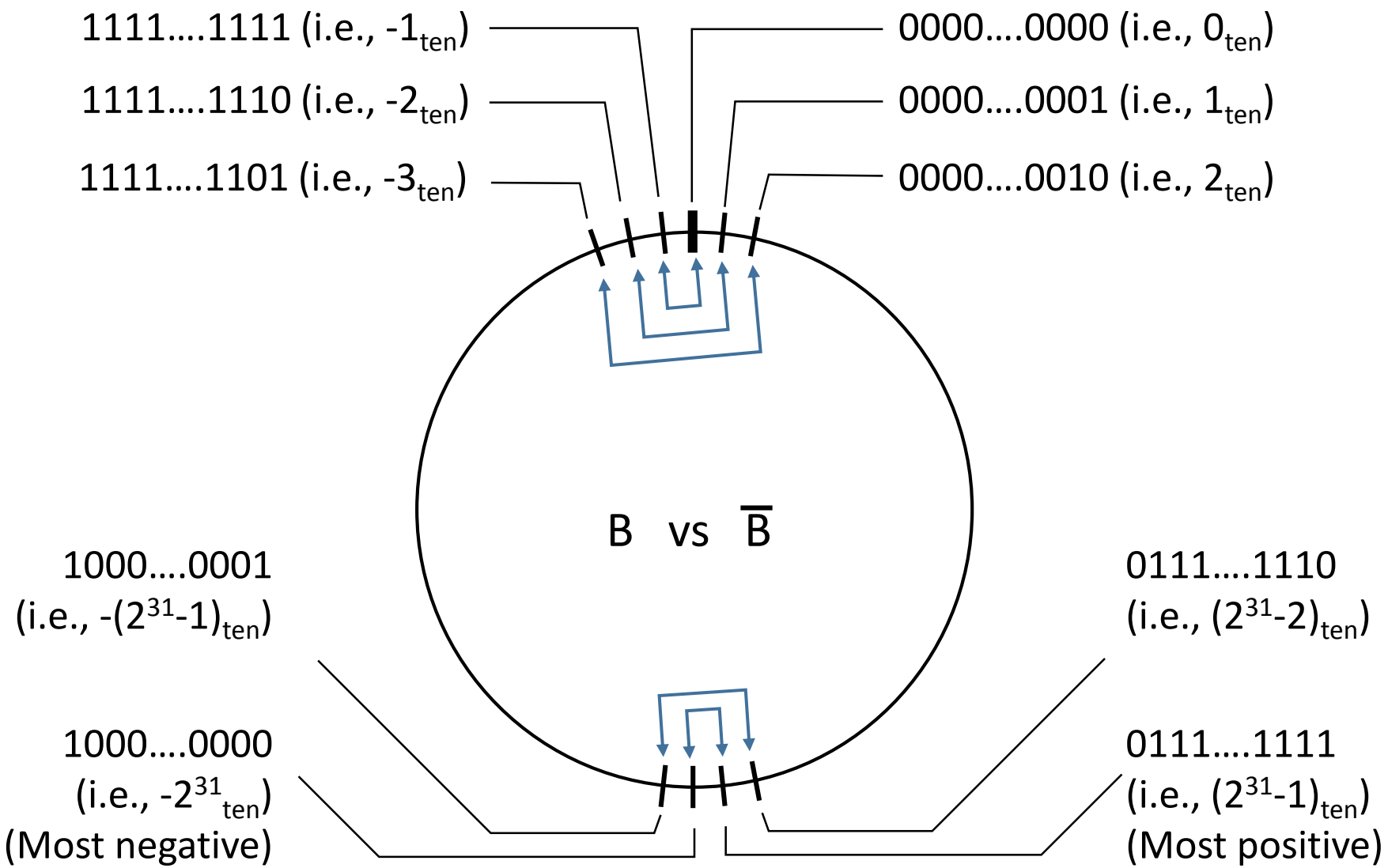
2's Compliment Signed Integers (Cont'd)



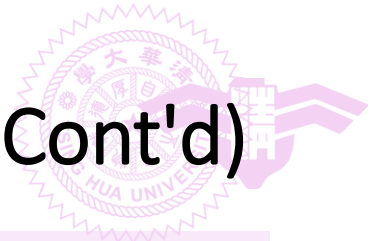
2's Compliment Signed Integers (Cont'd)



2's Compliment Signed Integers (Cont'd)



2's Compliment Signed Integers (Cont'd)



1111….1111 (i.e., $\mathbf{-1}_{ten}$) ⟷ 0000….0000 (i.e., $\mathbf{0}_{ten}$)

1111….1110 (i.e., $\mathbf{-2}_{ten}$) ⟷ 0000….0001 (i.e., $\mathbf{1}_{ten}$)

1111….1101 (i.e., $\mathbf{-3}_{ten}$) ⟷ 0000….0010 (i.e., $\mathbf{2}_{ten}$)

$$-B = \overline{B} + 1 = 2^N - B$$

$$\overline{B} = -(B + 1) = 2^N - (B + 1)$$

1000….0001 (i.e., $\mathbf{-(2^{31}-1)}_{ten}$) ⟷ 0111….1110 (i.e., $\mathbf{(2^{31}-2)}_{ten}$)

1000….0000 (i.e., $\mathbf{-2^{31}}_{ten}$) ⟷ 0111….1111 (i.e., $\mathbf{(2^{31}-1)}_{ten}$)

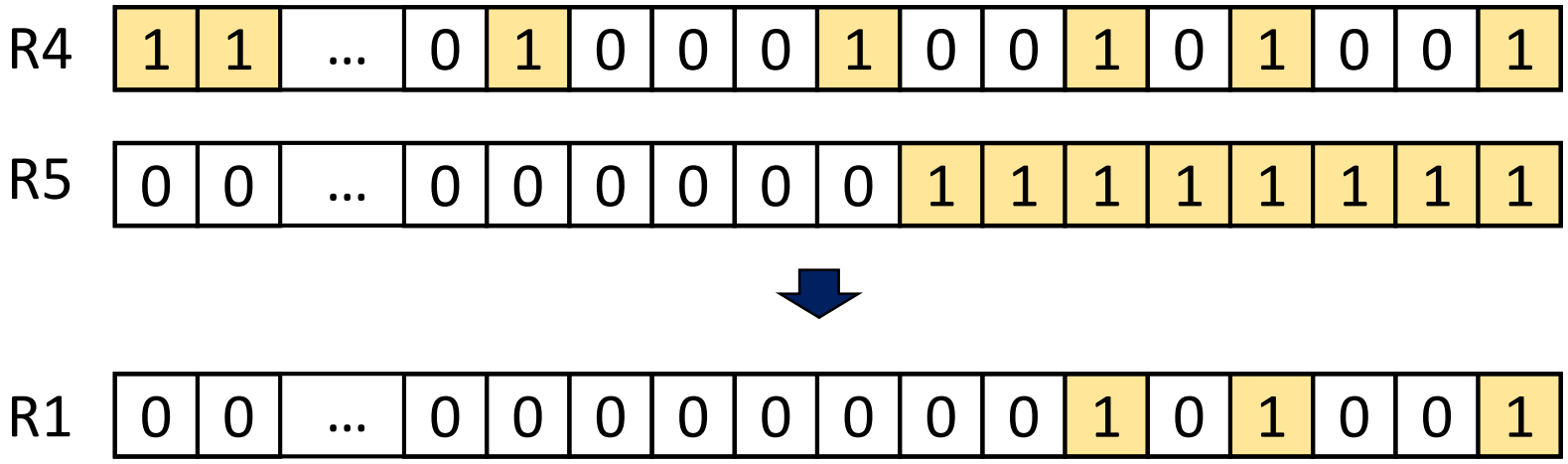
(Most negative) (Most positive)

Outline

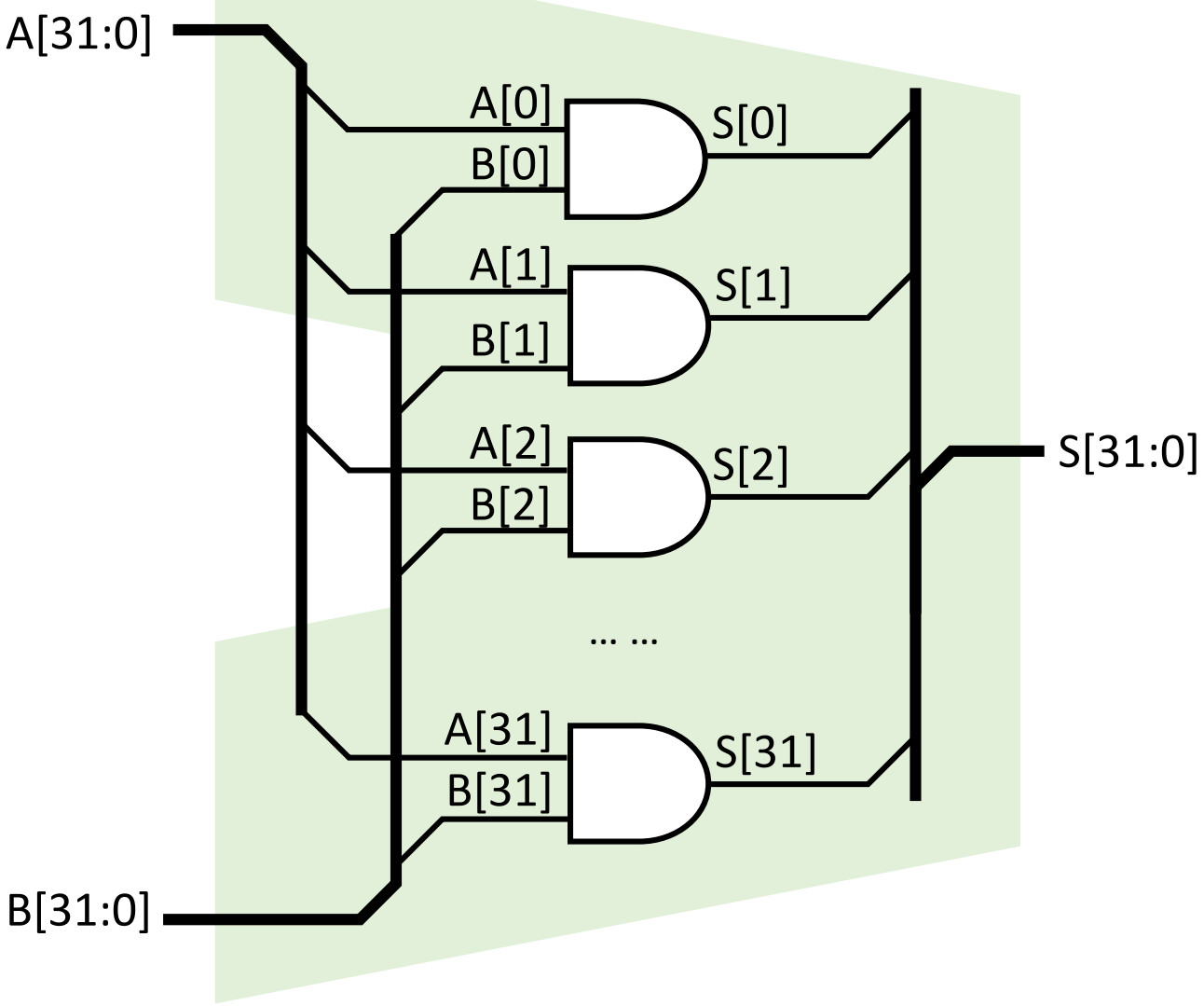
- Overview
- Integer operations
  - Bit-wise logical operations
  - Additions, subtractions
  - Comparisons
  - Multiplications
  - Divisions
- Floating point operations
  - Additions, subtractions
  - Multiplications

Bit-Wise Logical Operations

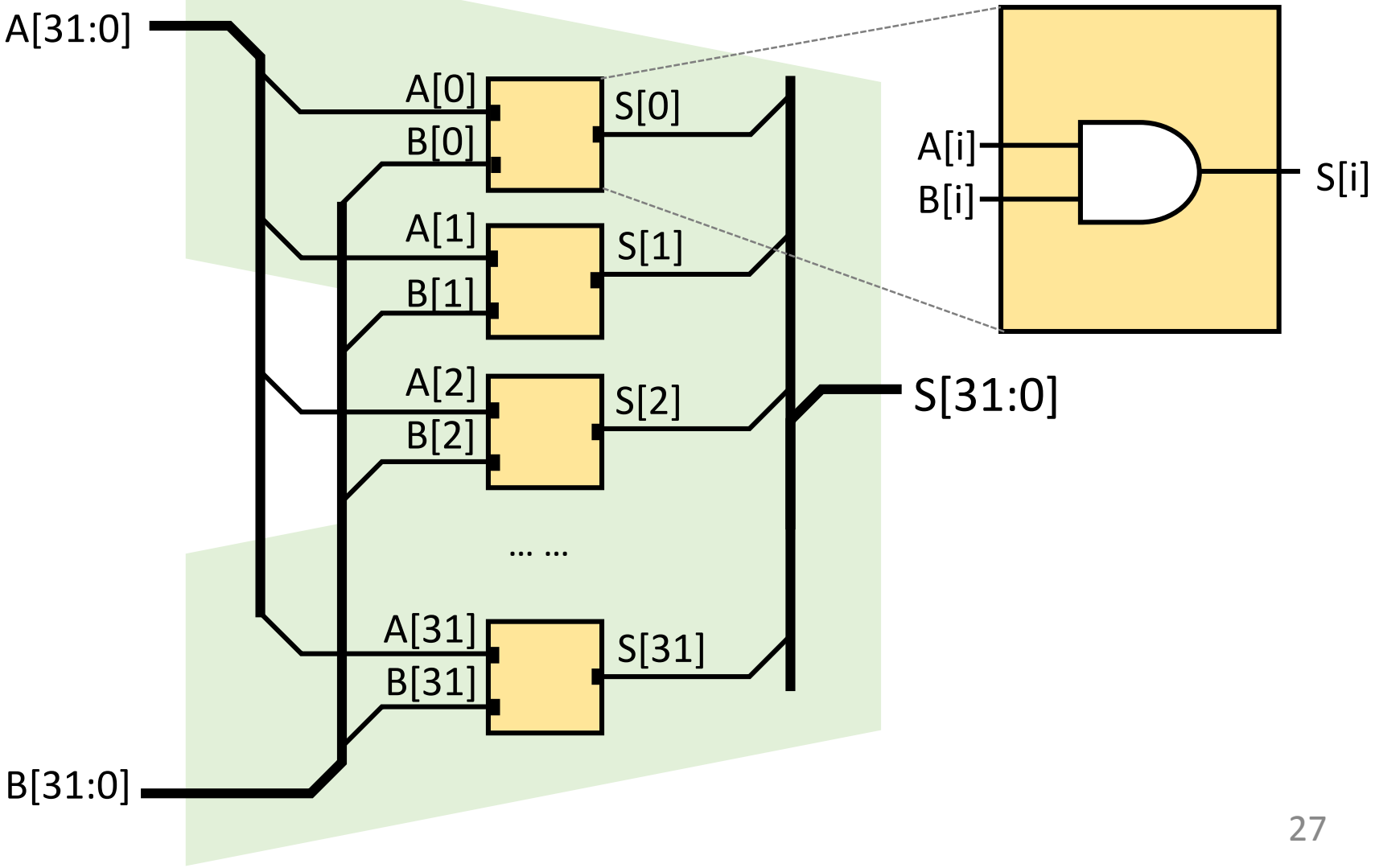
- Common CPU instructions
  - and, andi, or, ori, nor, xor, xori
- Example
  - and R1, R4, R5 // R1 = (R4 & R5)



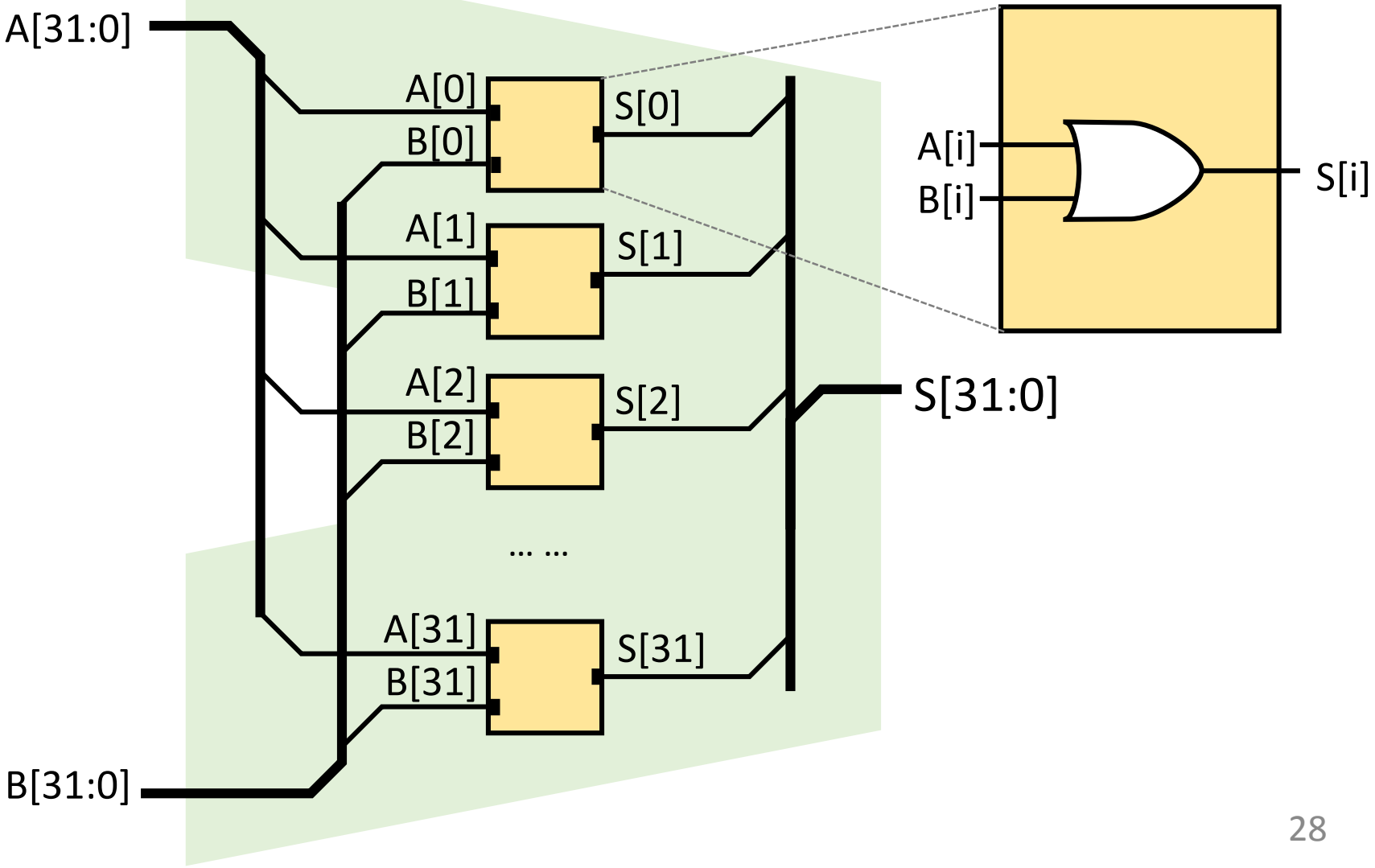
Supporting AND is Easy



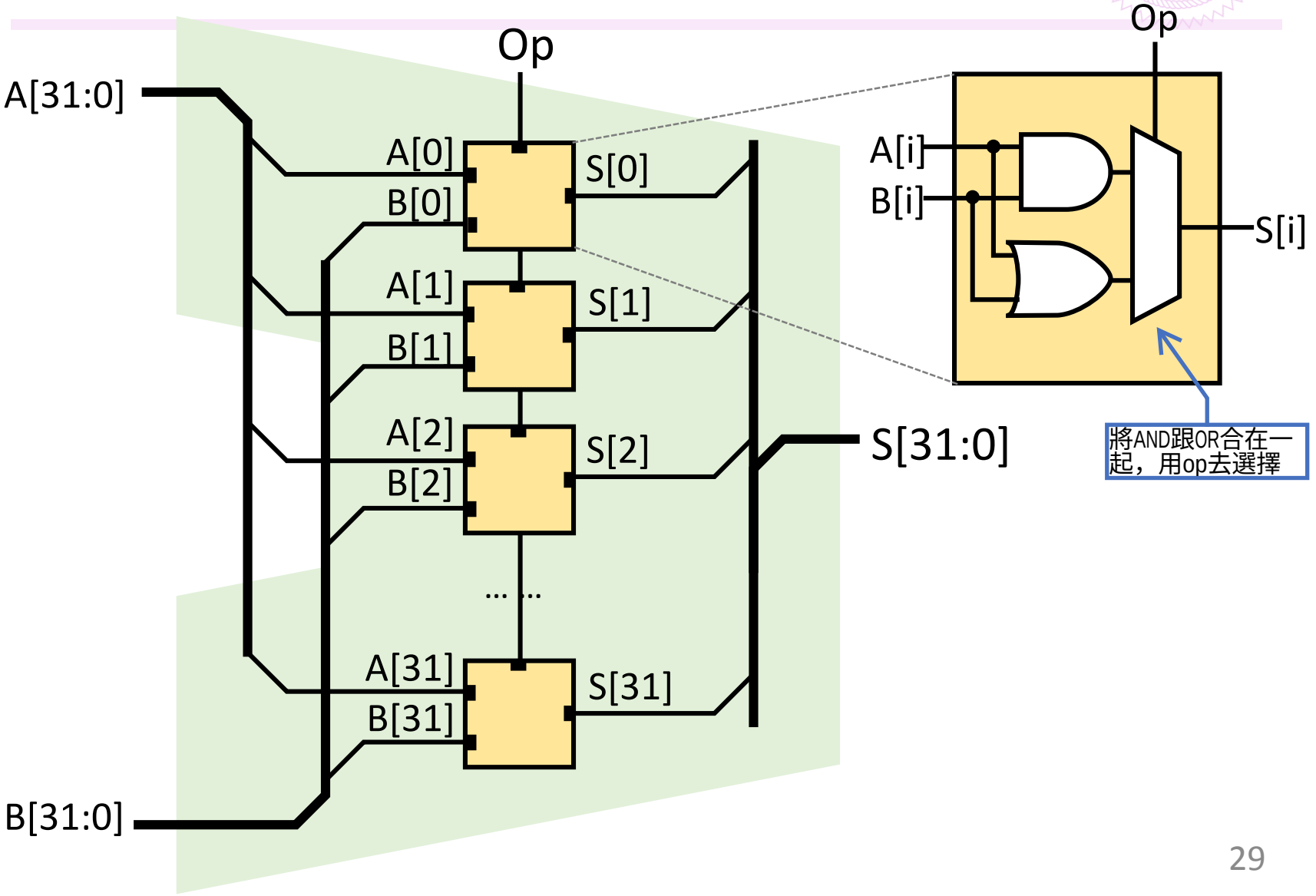
Support AND (Block Diagram)



Support OR

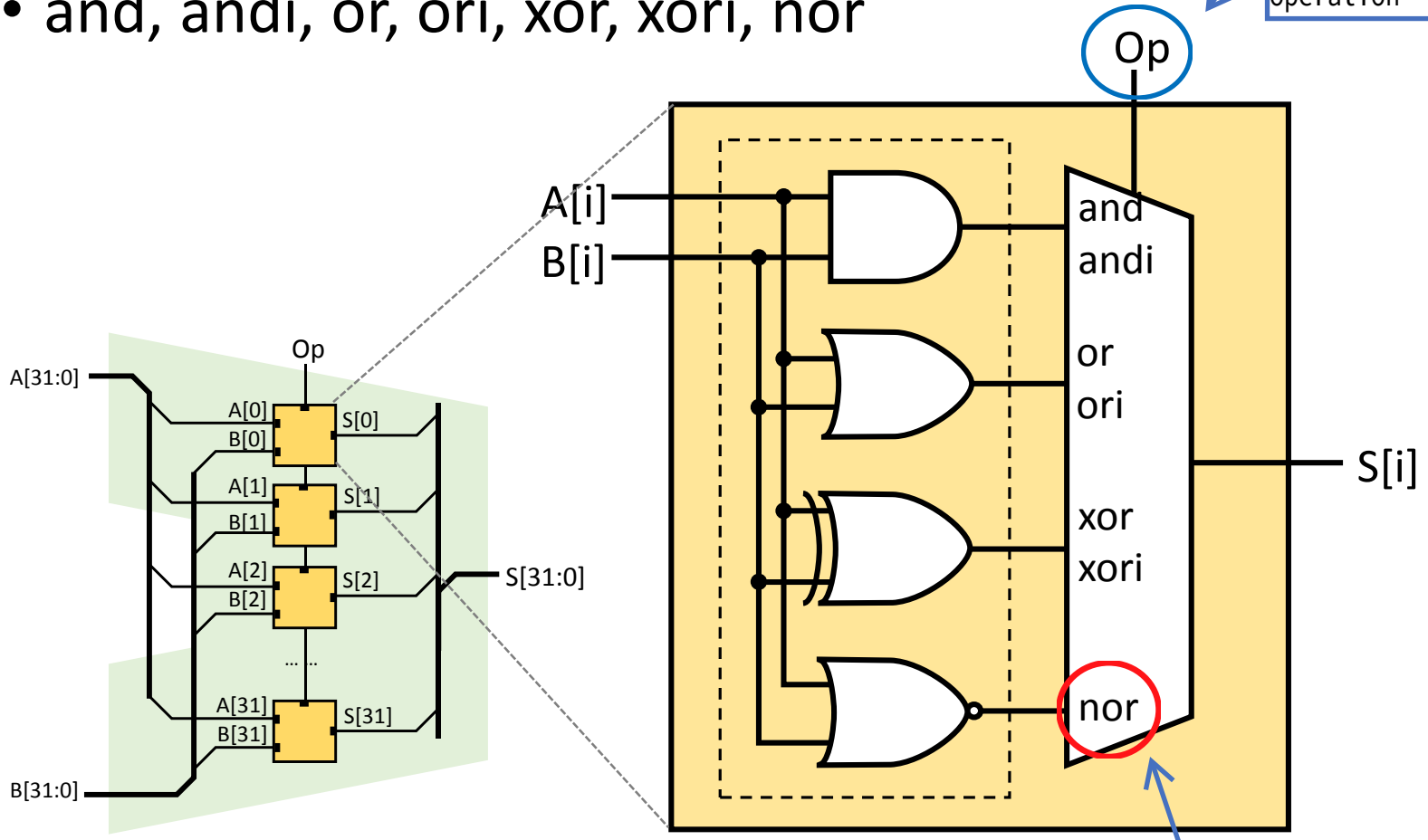


Support Both AND and OR

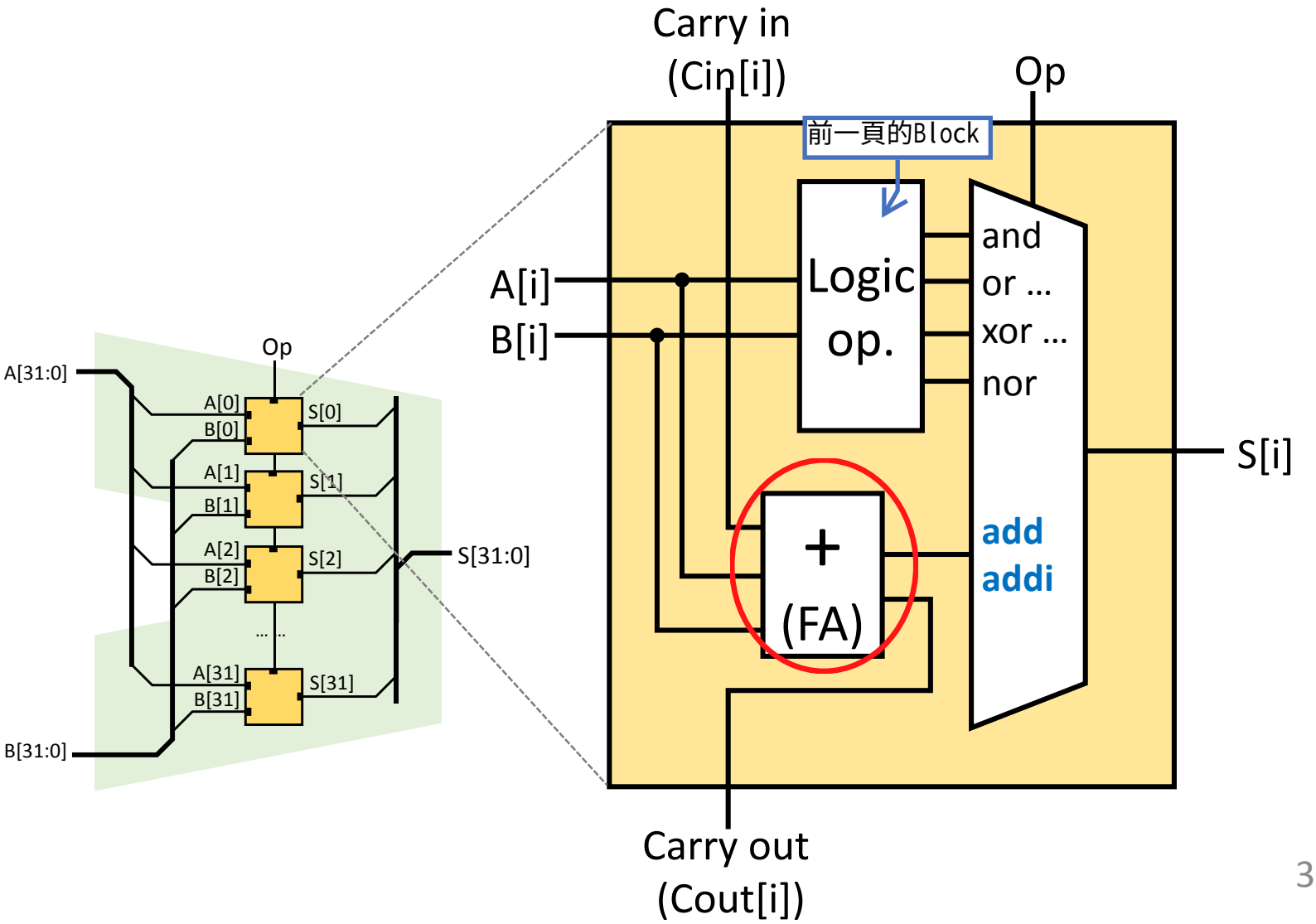


Support All Logical Instructions

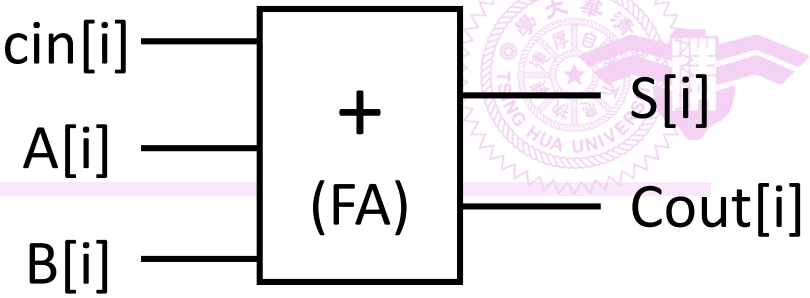
- and, andi, or, ori, xor, xori, nor



Further Support ADD



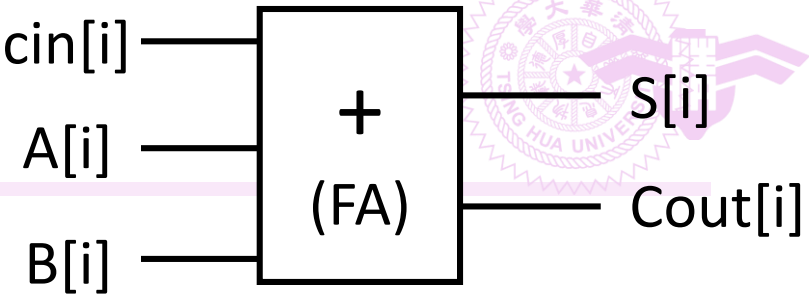
1-Bit Full Adder (FA)



- Outputs = the sum of the three input bits

Cin[0]	A[i]	B[i]	Cout[i]	S[i]
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

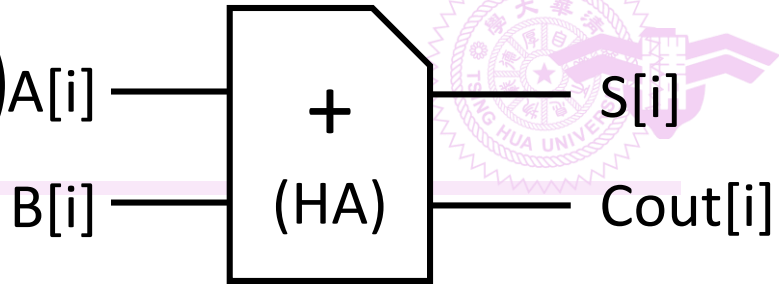
1-Bit Full Adder (FA)



- Some observatios
  - Three inputs are interchangeable
  - Outputs = the number of 1's in the inputs
  - S is 1 if the number of 1's is odd (i.e., XOR)
  - Cout (進位) is 1 if the number of 1's ≥ 2

Cin[0]	A[i]	B[i]	Cout[i]	S[i]
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

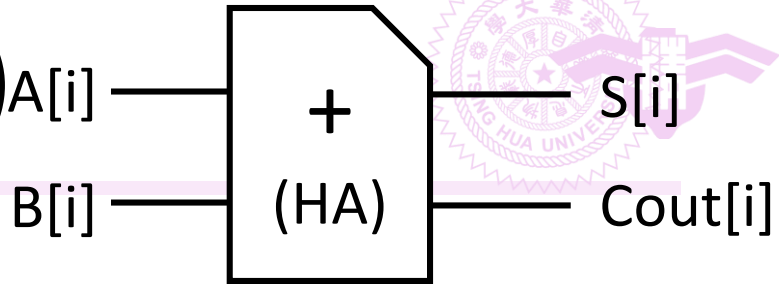
1-Bit Half Adder (HA)



- Outputs = the sum of the two input bits

A[i]	B[i]	Cout[i]	S[i]
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

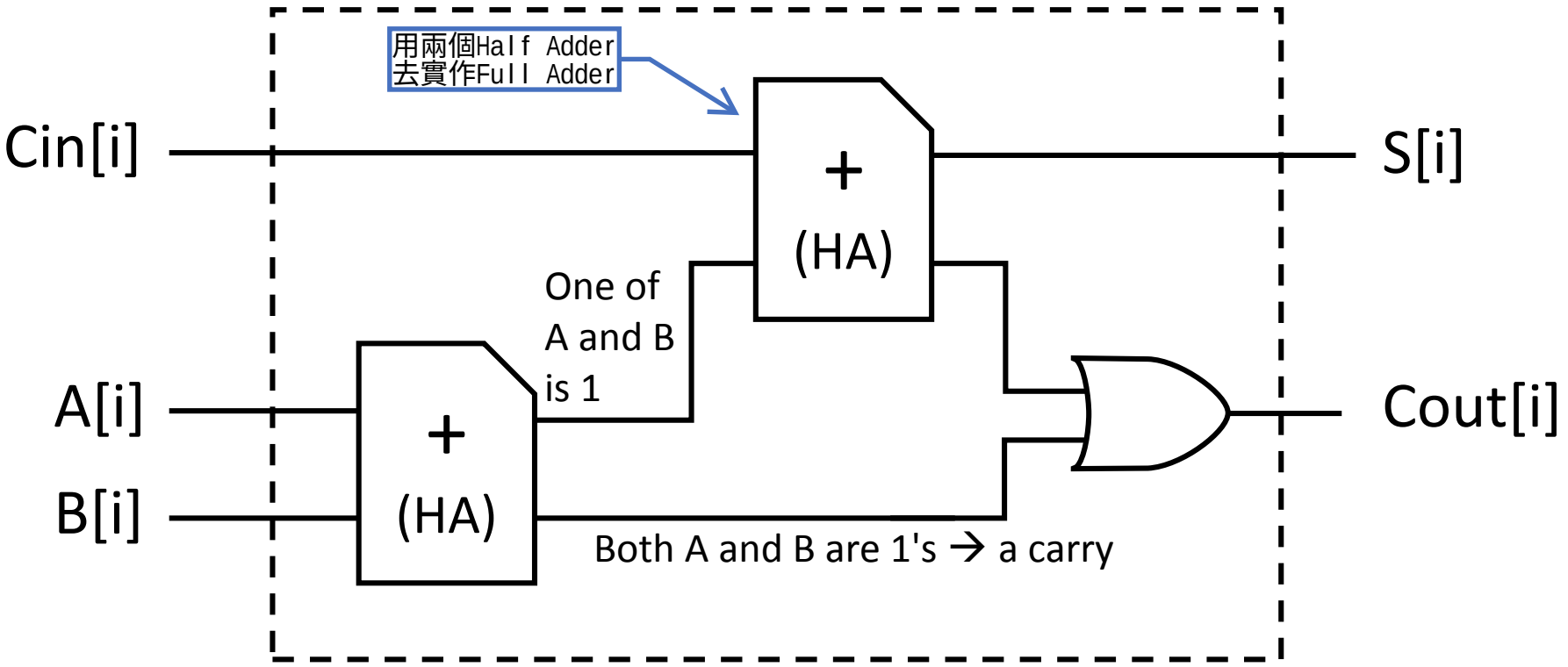
1-Bit Half Adder (HA)



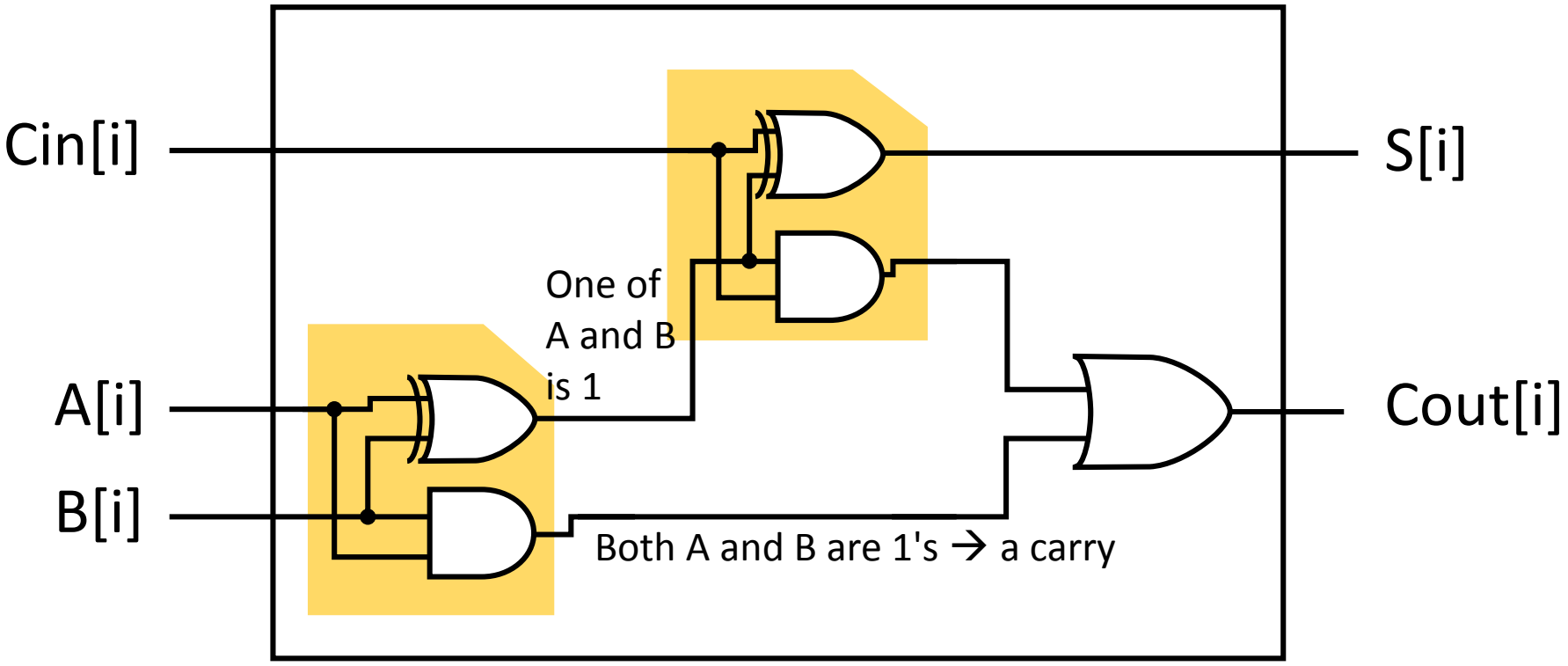
- Some observations
  - Two inputs are interchangeable
  - Outputs = the number of 1's in the inputs
  - S is 1 if the number of 1's is odd (i.e., XOR)
  - Cout (進位) is 1 if both the inputs are 1's (i.e., AND)

A[i]	B[i]	Cout[i]	S[i]
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

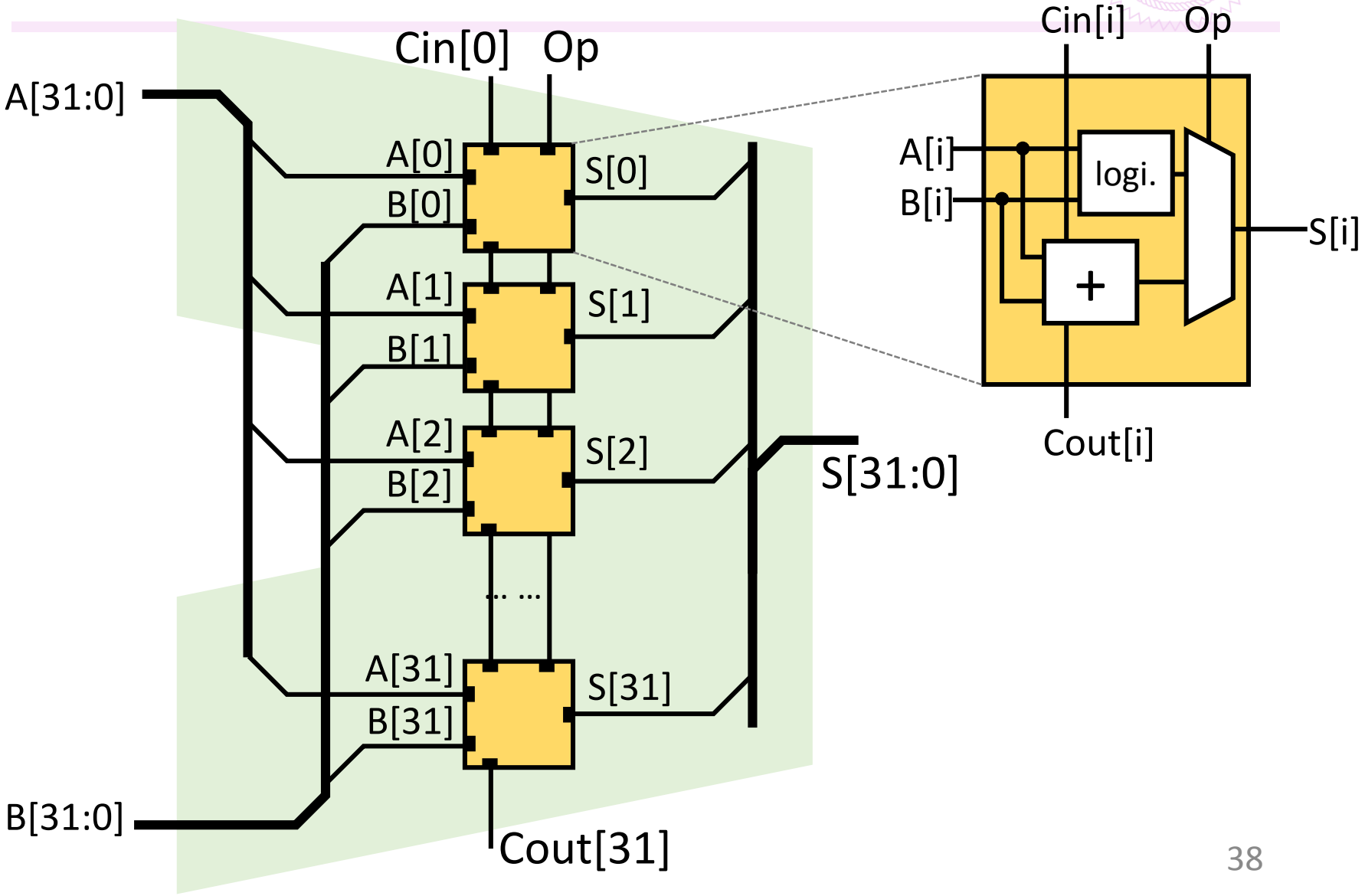
Building a 1-Bit Full Adder



Building a 1-Bit Full Adder (Cont'd)

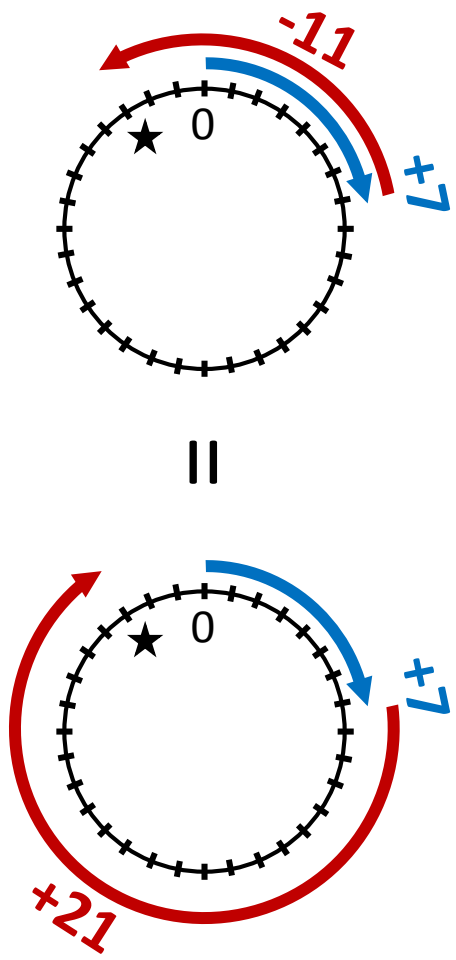


Addition


Cin[0]
Op
A[31:0]
A[0]
B[0]
S[0]
A[1]
B[1]
S[1]
A[2]
B[2]
S[2]
S[31:0]
A[31]
B[31]
S[31]
B[31:0]
Cout[31]
Cin[i]
Op
A[i]
B[i]
logi.
+
S[i]
Cout[i]

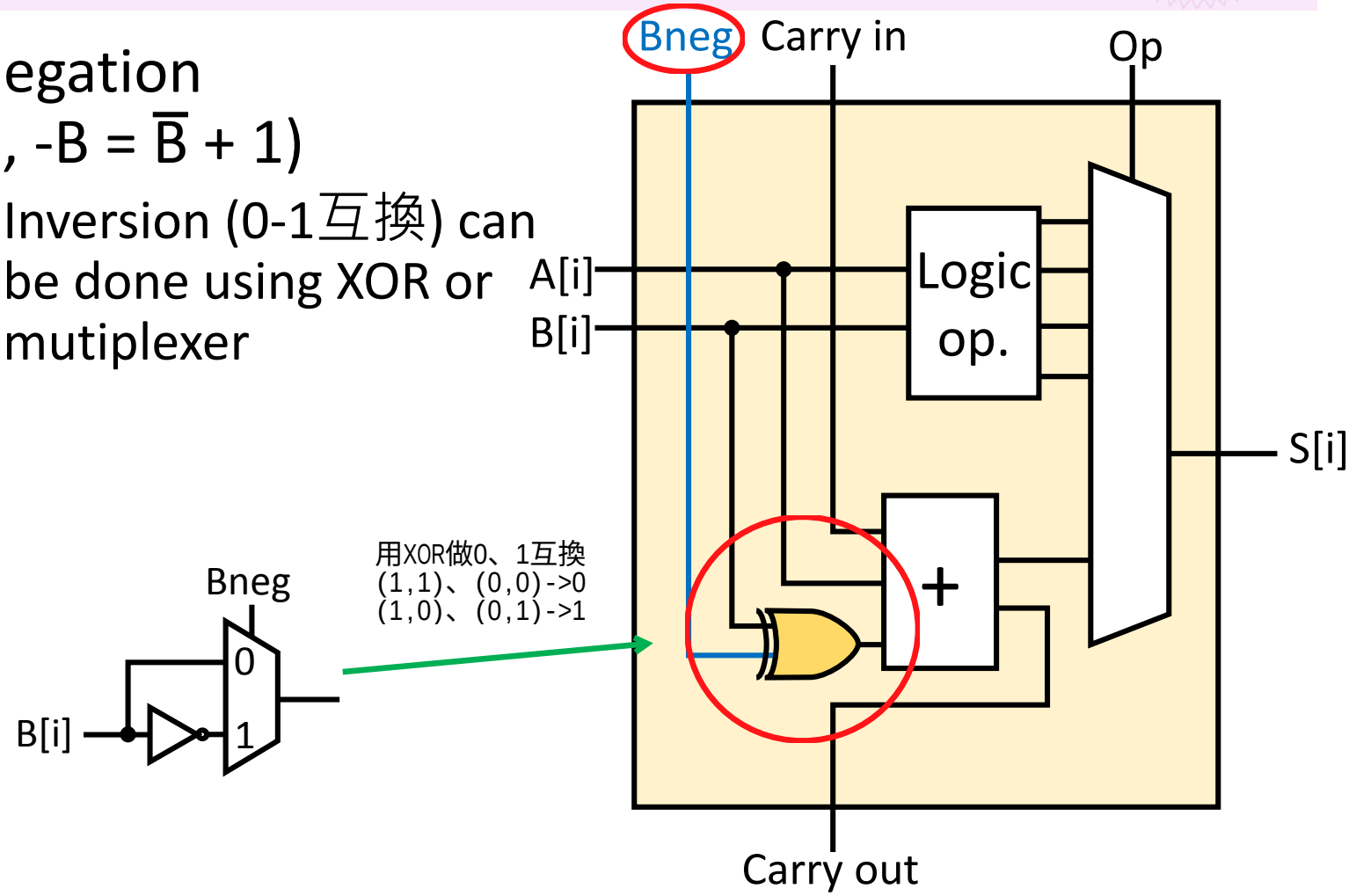
Subtraction

- Subtraction can be done using an addition and an negation(取負)
  - A - B
    = A + (-B)
    = A + ($\overline{B}$ + 1)
- E.g., in 5-bit (modulo 32) arithmetic
  - **7** - **11**
    = 7 + (-11)
    = 7 + 21 // 32-11 = 21; ($\overline{11}$+1) = 21
    = -4 // interpreted as singed



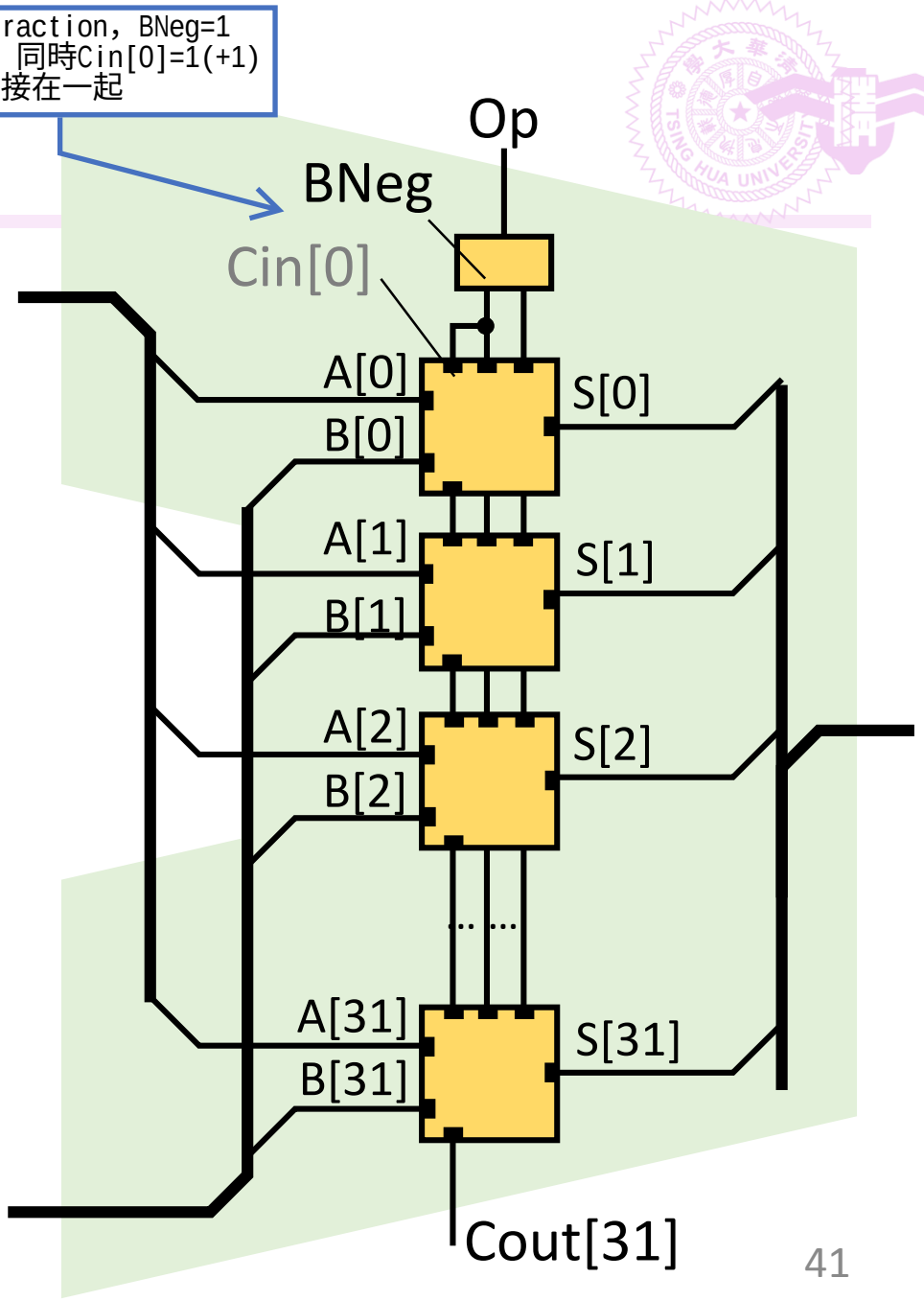
Subtraction

- B negation
  (i.e, -B = $\overline{B}$ + 1)
  - Inversion (0-1互換) can be done using XOR or mutiplexer



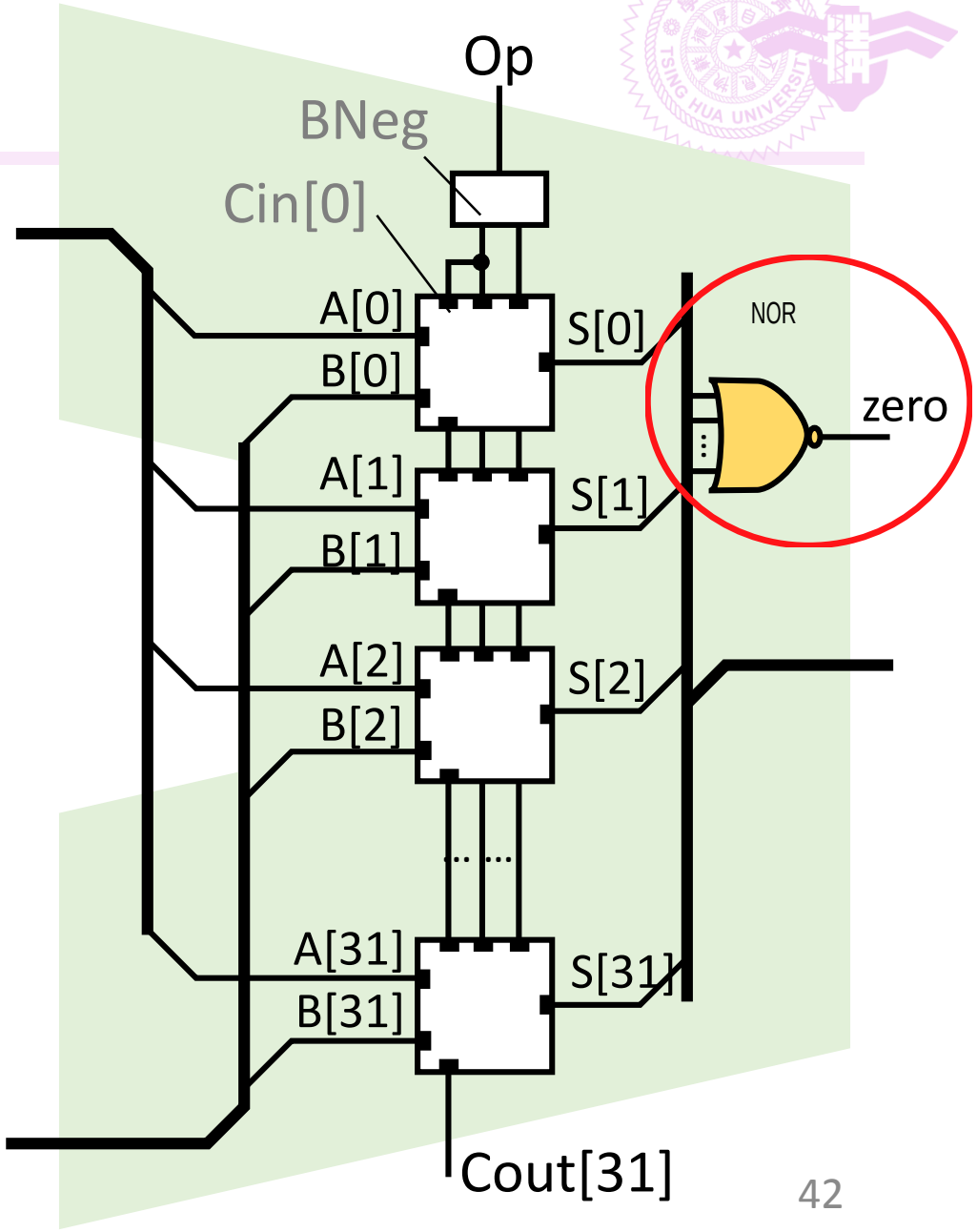
Subtraction

- B negation (i.e, -B = $\overline{B}$ + 1)
  - +1 is done by using Cin[0]



Zero Detection

- Indicate whether the output 32-bit result is a zero



Overflow Detection

- Before describing overflow detection methods, some things need to be clarified
  - Overflow detection methods are different for signed and unsigned operations
  - Comparisons (SLT and SLTU) are related to overflow detection

Overflow Detection (Signed Add)

A	B	S=A+B	overflow?
+	+	-	Yes
-	-	+	Yes
+	+	+	No
-	-	-	No
+	-	(overflow cannot occur)	
-	+		
0	+		
0	-		
+	0		
-	0		

Overflow Detection (Signed Add)

- According to the above descriptions

兩個正數相加後變成負的(Overflow)

A[31]	B[31]	Cin[31]	Cout[31]	S[31]	Overflow
0	0	0	0	0	0
0	0	1	0	1	**Yes**
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	**Yes**
1	1	1	1	1	0

傳統作法：將兩個Case做OR Gate去輸出是否Overflow
(先3個做AND Gate)

兩個負數相加後變成正的(Overflow)

Overflow Detection (Signed Add)

- Tricky approach

Tricky作法:觀察Cin和Cout發現兩者不相等就Overflow

A[31]	B[31]	Cin[31]	Cout[31]	S[31]	Overflow
0	0	0	0	0	0
0	0	1	0	1	**Yes**
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	0	**Yes**
1	1	1	1	1	0

Overflow Detection (Unsigned Add)

A	B	A + B overflow?
non-zero	non-zero	If (A+B) produces a carry
0	non-zero	(cannot occur)
non-zero	0	
0	non-zero	

Overflow condition: If (A+B) produces a carry

Overflow Detection (Unsigned Sub)

A	B	A - B = A + (-B) overflow?
non-zero	non-zero	If (A-B) > A (i.e., if A + (-B) does not produce a carry)
0	non-zero	
non-zero	0	(cannot occur)
0	0	

只要A-B<0對於unsigned來說就是overflow

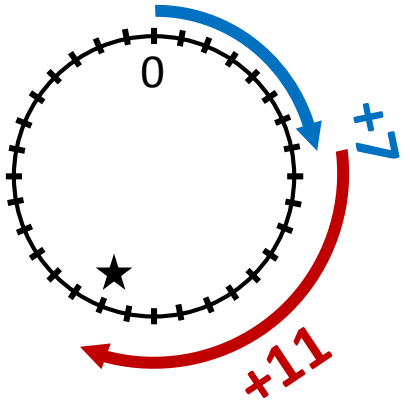
Overflow condition: If A + (-B) does not produce a carry

Unsigned Integers和2's complement都是用一樣的加法電路，但是在判斷overflow有些不同。

Overflow Detection (Unsigned Operands)

- 7 + 11

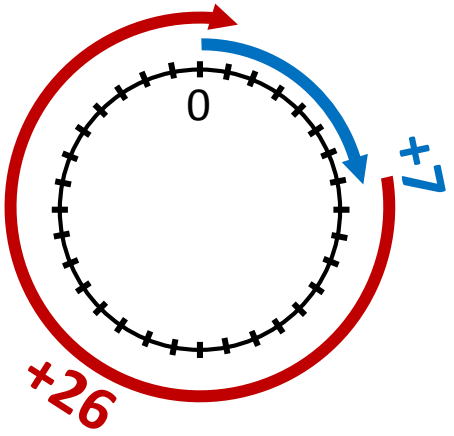
```
  0 0 1 1 1
+ 0 1 0 1 1
-----------
0 1 0 0 1 0
```



- 7 + 26 (overflow)

```
  0 0 1 1 1
+ 1 1 0 1 0
-----------
1 0 0 0 0 1
```

carry==1 during unsigned add indicates overflow

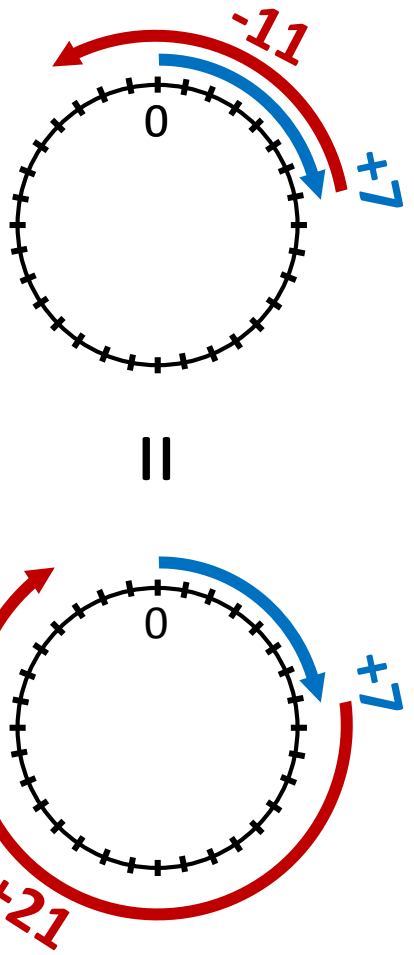


Overflow Detection (Unsigned Operands)

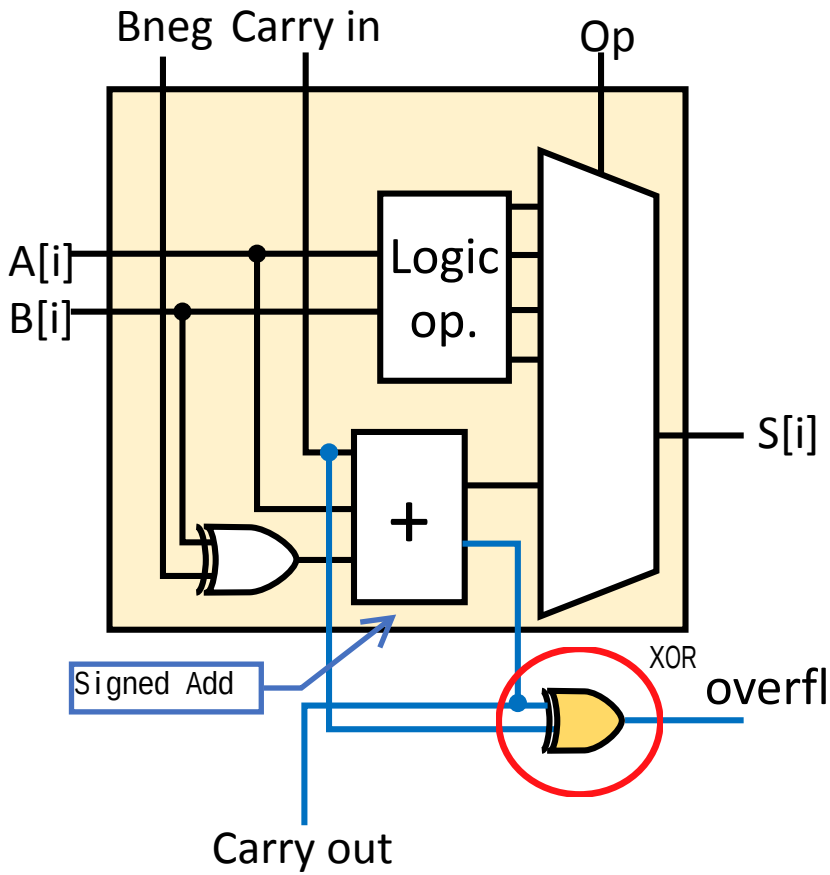
- 7 - 11
  = 7 + (-11)
  = 7 + 21 (11'+1)
  = 28 ☹

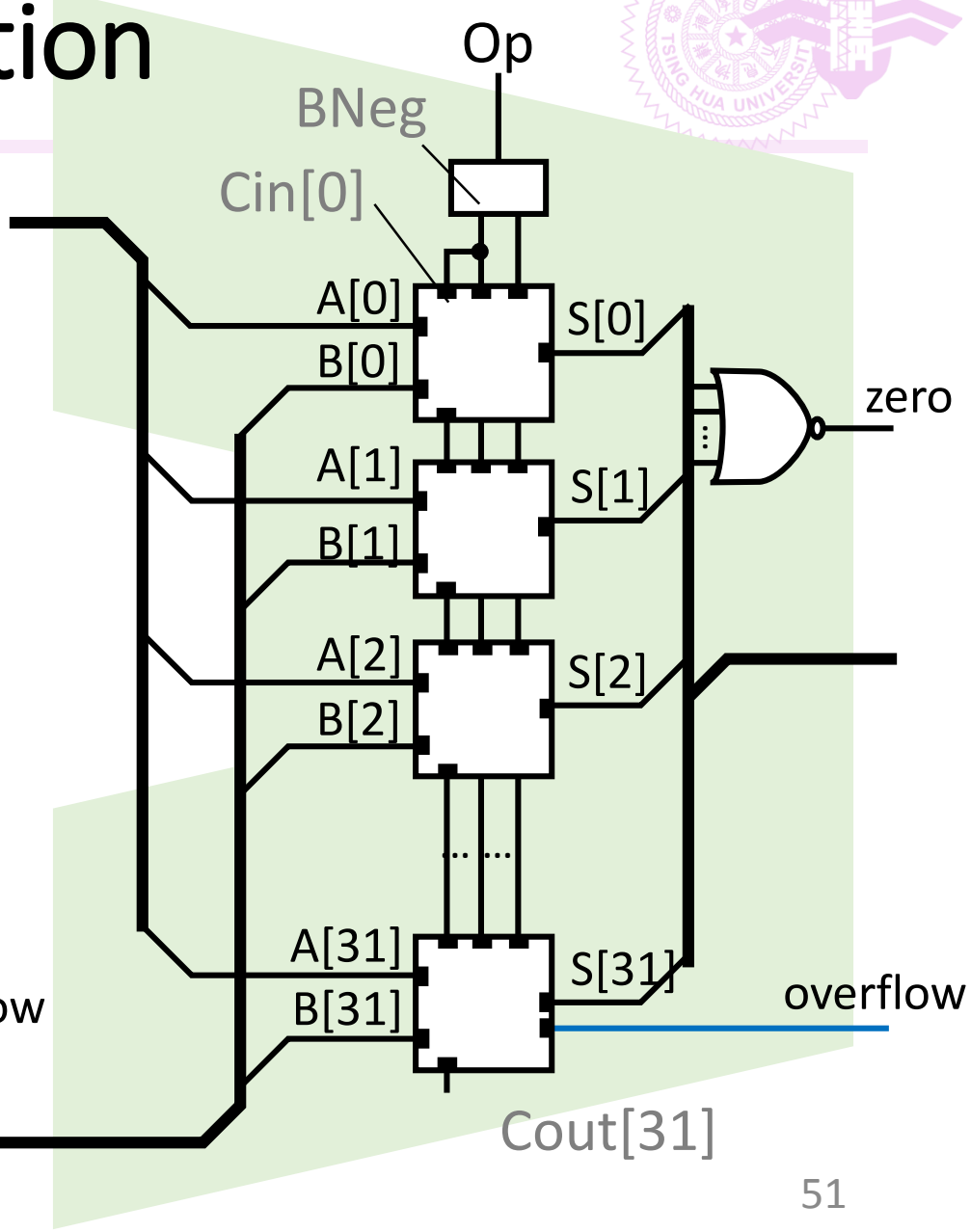
carry==0 during unsigned sub indicates overflow

```
  0 0 1 1 1
+ 1 0 1 0 1
-----------
0 1 1 1 0 0
```

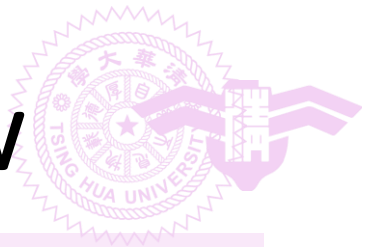


Overflow Detection





More Discussions on Overflow



- Different languages have different designs

繞回1(對2^n取餘數)

Language	Unsigned integer	Signed integer
Ada	modulo the type's modulus	raise Constraint_Error
C/C++	modulo power of two	undefined behavior
C#	modulo power of 2 in unchecked context; System.OverflowException is raised in checked context	
Java	NA	ignored
Python 2	NA	convert to long
Seed7	NA	raise OVERFLOW_ERROR
Scheme	NA	convert to bigNum
Smalltalk	NA	convert to LargeInteger
Swift	Causes error unless using special overflow operators.	

wikipedia

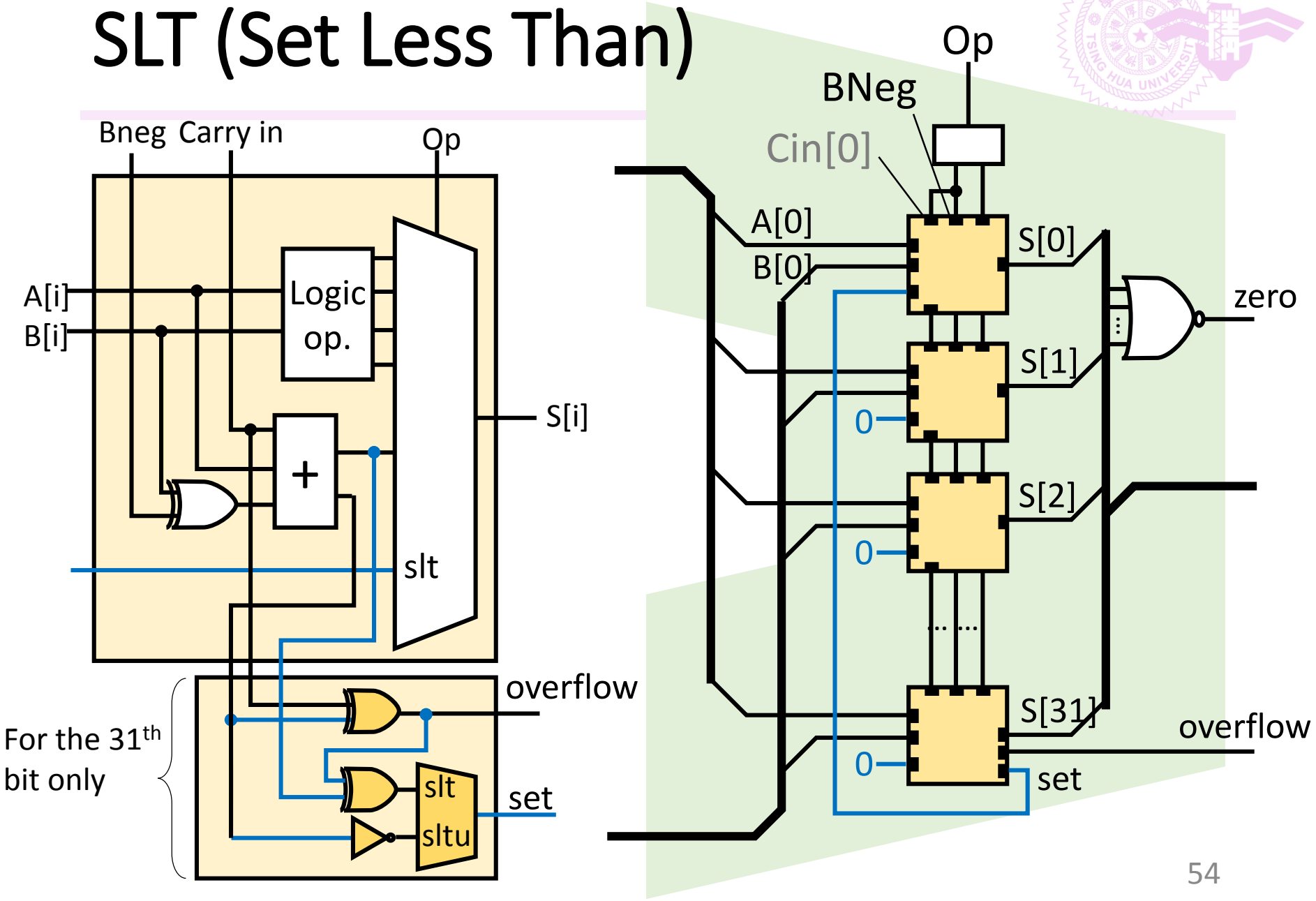
SLT and SLTU (Set Less Than)

- Requirements
  - ALU outputs 1 (i.e., 0000….0001) if A < B, otherwise outputs 0
- Methods
  - Leverage SUB and SUBU hardware

所以最後set要接回去第一個bit

	Operation	Overflow	Sign(S) (i.e., S[31])	Carry out (i.e., S[32])	Outcome
SLT	S=A-B= A+$\overline{B}$+1	0	1 (A-B<0)	Don't Care	1 (A<B)
			0		0 (A≧B)
		1 (本來A-B>=0 卻Overflow)	1		0 (A≧B)
			0		1 (A<B)
SLTU Unsigned		Don't care		1 (No Carry ->Overflow (A-B<0))	0 (A≧B)
				0	1 (A<B)

SLT (Set Less Than)



Summary

- Bit-wise logical
  - ✔ and, andi
  - ✔ or, ori
  - ✔ xor, xori
  - ✔ nor
- Arithmetic
  - ✔ add, addi
  - ✔ sub, subu
    - mul
    - div
- Comparisons
  - ✔ slt, slti, sltu, sltiu

Outline

- Overview
- Integer operations
  - Bit-wise logical operations
  - Additions, subtractions
  - Comparisons
  - Multiplications
  - Divisions
- Floating point operations
  - Additions, subtractions
  - Multiplications