Topic 3: Building the ALU

Processor units

- ALU (arithmetic logic unit)： performs arithmetic operations

Registers： to hold data that is being processed

Control unit： fetches each instruction

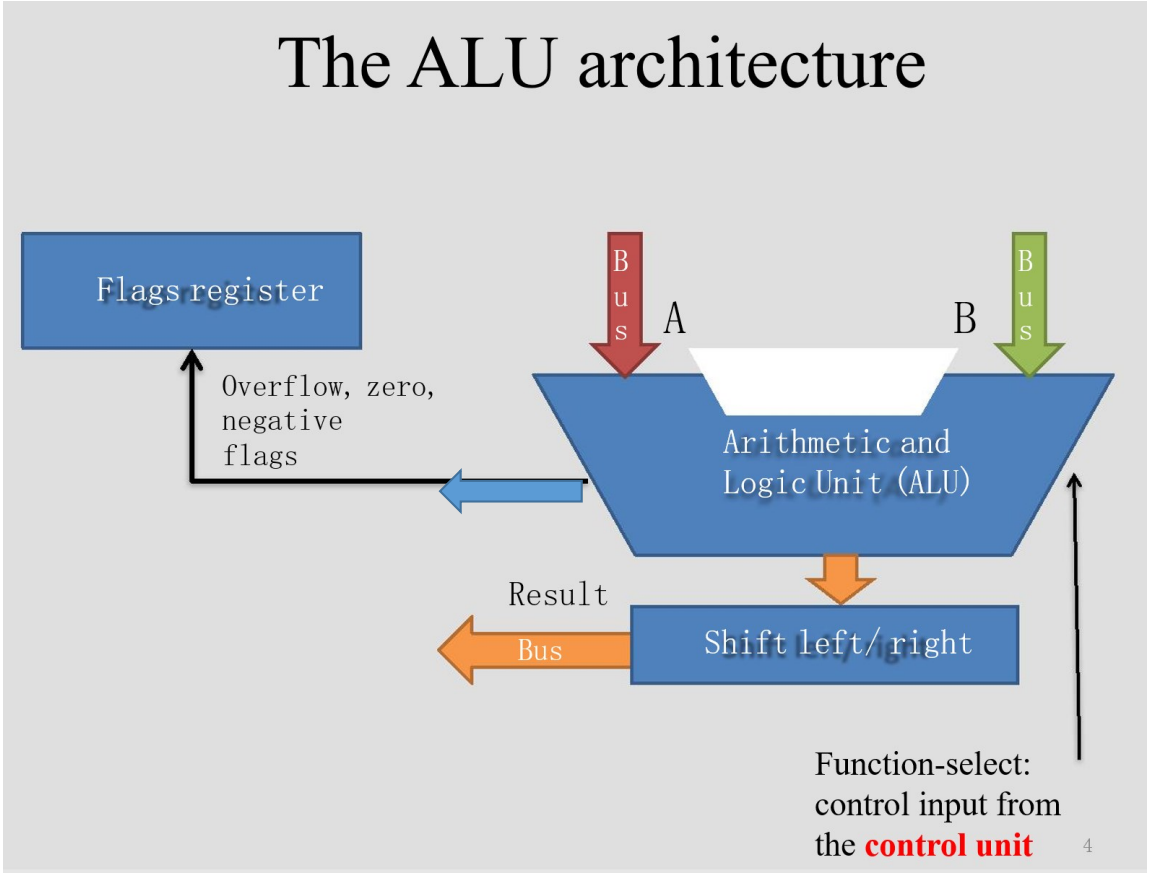
ALU units

Arithmetic and logic operations units:

-add, subtract, multiply, divide, shift of integers, …
and, or, not, xor, shift to left or right…
comparisons: <, ≤, =, ≠, ≥, >: how to do?

-status flag register (ZF, SF, OF)

- Adder is the basic part for ALU
  bit: A+B (A,B is binary digit)
  -> multiple addition: AB+CD



half adder implement

HALF ADDER

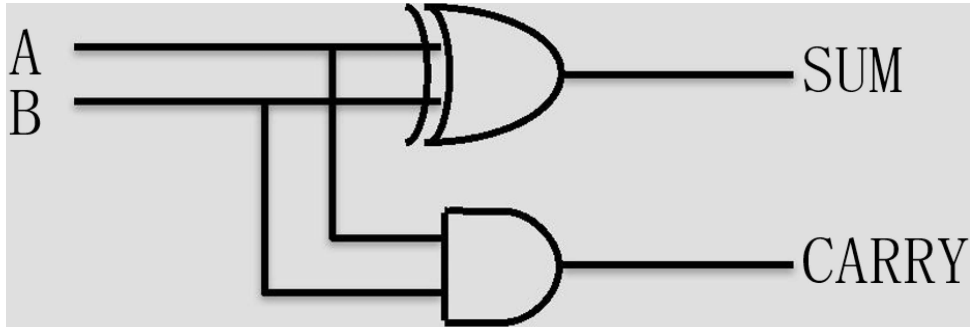
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

SUM

A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

CARRY

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1



Use design rule to design full adder

A	B	C	S	D
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

S=A’B’C+A’BC’+AB’C’+ABC

D=A’BC+AB’C+ABC’+ABC

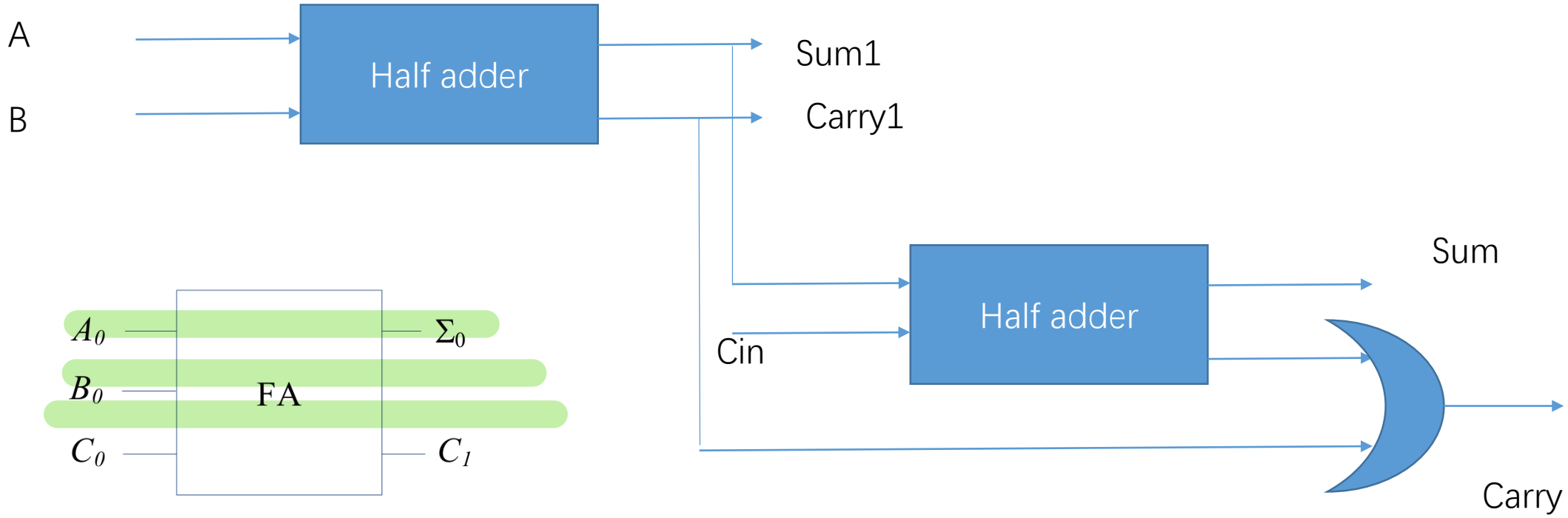
it is useless to use kanoral for this one

Use two's half adder to design a full adder

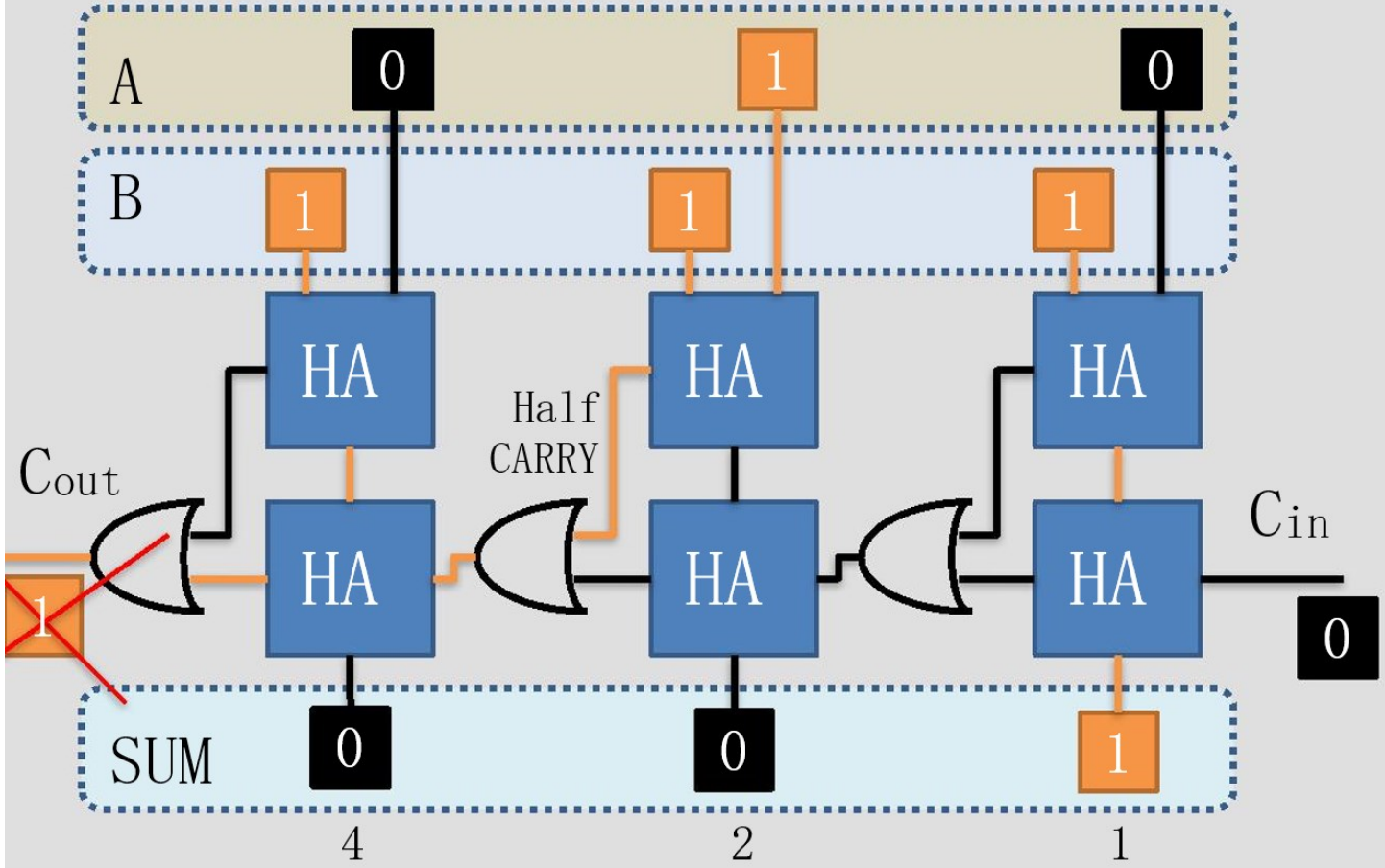
S=A'B'C+A'BC'+AB'C'+ABC=C'(A'B+AB')+C(A'B'+AB)=C'(A ⊕ B)+C(A ⊕B)'=C⊕(A⊕B)

[Use de Morgan law A'B'+AB=(A'B+AB')' Convert them to basic gate form(XOR)

D=A'BC+AB'C+ABC'+ABC =AB+C(A'B+AB')=AB+C(A⊕B)



Multi-stage full adder with ripple carry



2

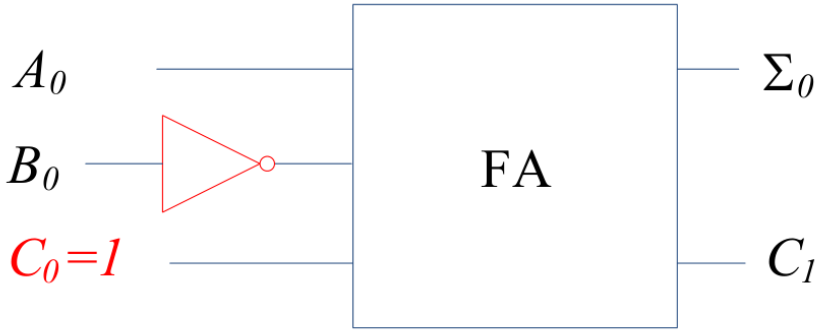
−1

= 1

How to design subtractor

method 1

A-B=A+(-B)=A+(B’+1)

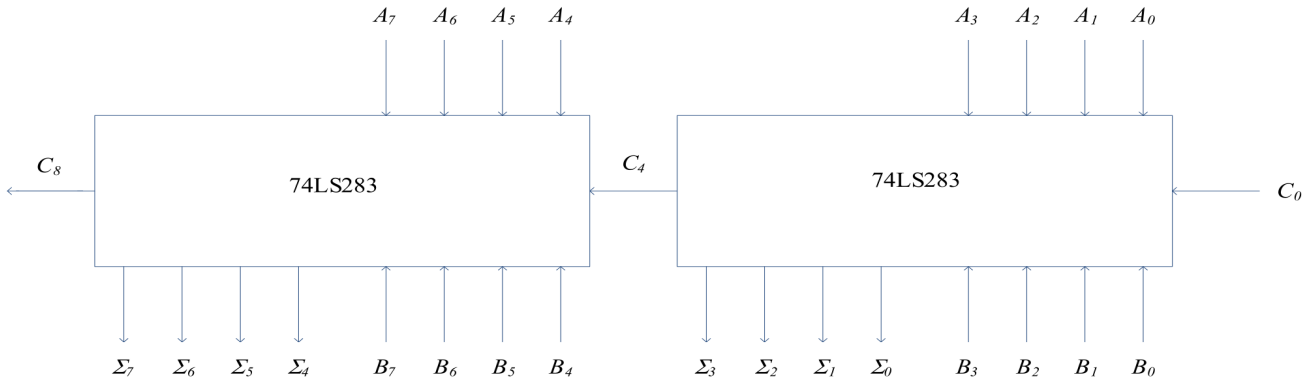


method 2

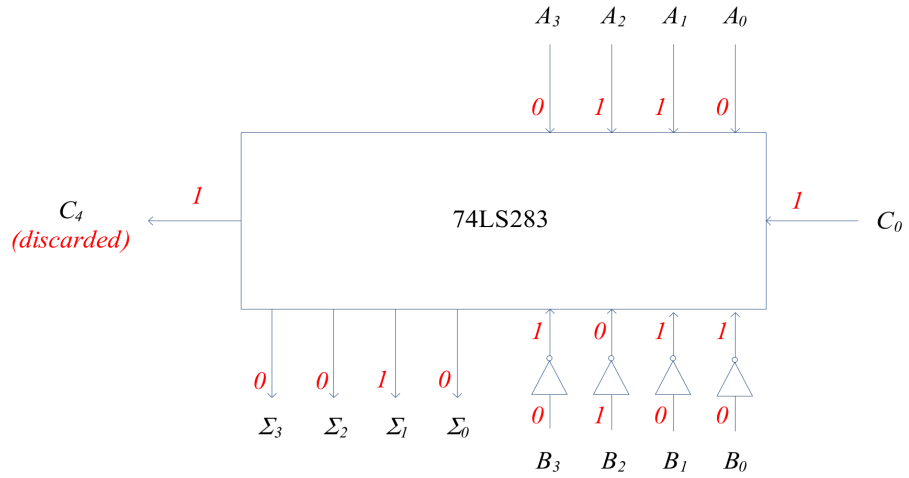
A	B	D	B	
0	0	0	0	
1	0	1	0	
0	1	1	1	
1	1	0	0	

D=AB’+A’B

B=A’B



8 bits additions



4 bits subtraction

The status flags register

reflects an aspect of the outcome of the most recent ALU operation

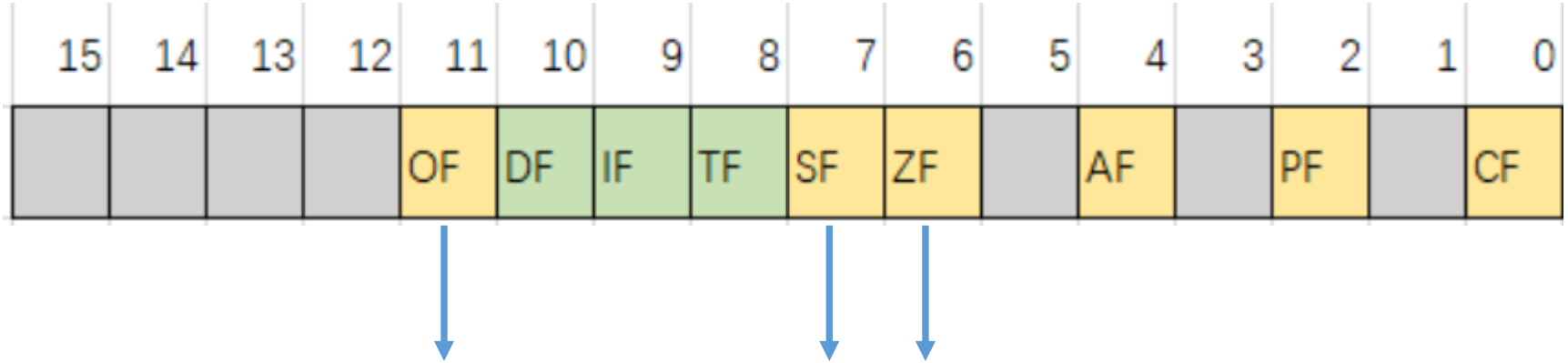
Zero flag

Sign flag

Overflow flag

Judge the conditional branch according to flag register

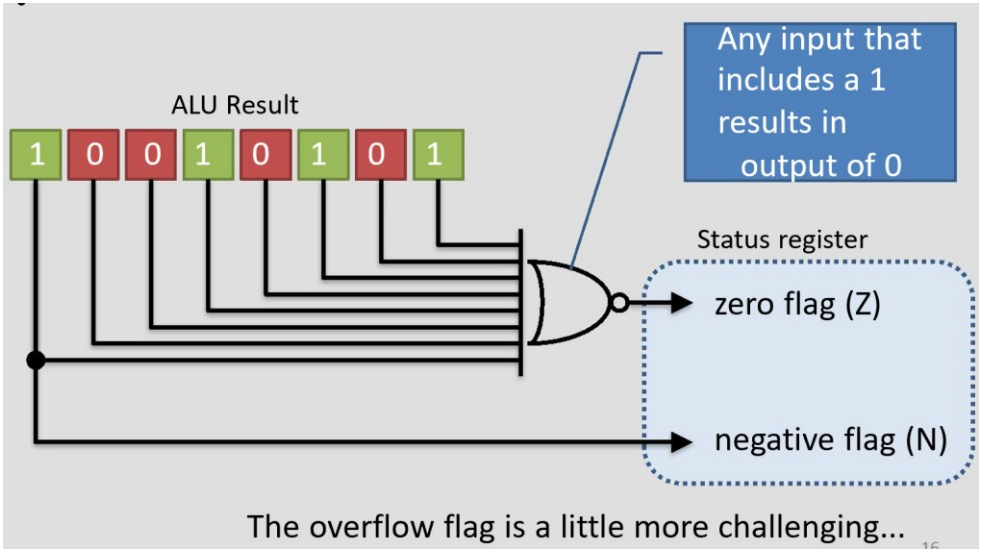
-if(A>0)
-if(A==0)
-if(A<0)



1 means the result status exist

Zero flags : and all bits

Sign flags : Check the Sign bit of that number



Overflow flag: the result is too large to store

++ -> +

-- -> -

Carry flag:carry in or carry out

Data range

- For unsigned number N bits
  data range:0 -(2^N-1)
- For signed number
  data range: -(2^{N-1})-(2^{N-1}-1)

- Example
- What is overflow
- 4 bits, -8→7
- 4+4=1000(8)→-8, -5+(-5)=0111=7

How to design logic gates to detect overflow

1. S,A,B
2. Cin, Cout(most left-side bit)

A	B	C_{in}	S	C_{out}	OF
0	0	0	0	0	0
0	0	1	1	0	1
1	0	0	1	0	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	1	0	1	0
1	1	0	0	1	1
1	1	1	1	1	0

A'B'S + ABS' CinCout' + Cin'Cout

1. Cin=0, Cout=1, A=B=1 overflow: two negative number addition
   leftmost bits are both 1 （A and B are 1, we are adding two negative numbers and, getting a positively-signed result.
2. Cin=1, C_{out}=0, A=B=0; overflow: two positive number addition
   leftmost bits are both 0 (A and B are 0), we are adding two positive numbers and getting a negatively-signed result
3. Other case: no overflow

a. A=B=1, Cin=1,Cout=1, two negative number addition, sum is negative
   A=0, B=1; A=1,B=0, or S=0; result can be positive and negative
   (no overflow for different sign number addition: -8→+7, 7-8=-1)

b. Cin=0, Cout=0
   where the leftmost bits are both 0,we are adding two positive and result is also positive. or where one leftmost bit are either 0 or 1
   One positive plus one negative, result is negative

- 4+5
- 0100
- +0101
- 1001（9） Cin=1, Cout=0

- -6-5
- 1010
- +1011 Cin =0, Cout=1
- 0001（1）

- -3-4
- 1101
- 1100
- 1001（-7）

-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

ALU other operation

- Comparisons

  We can use 2's complement gives us subtraction
  subtraction allows us to do comparisons: <, >, =
  do "trial subtraction", then check for +ve, -ve, or zero result)

- Multiplication and division?

  Multiplication: shifting addition
  Division: subtraction

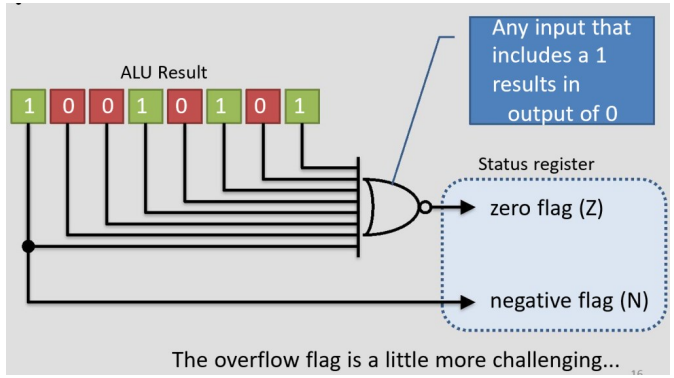
- Logic operations can easily be built directly from logic gates

  AND, OR, NOT, ...

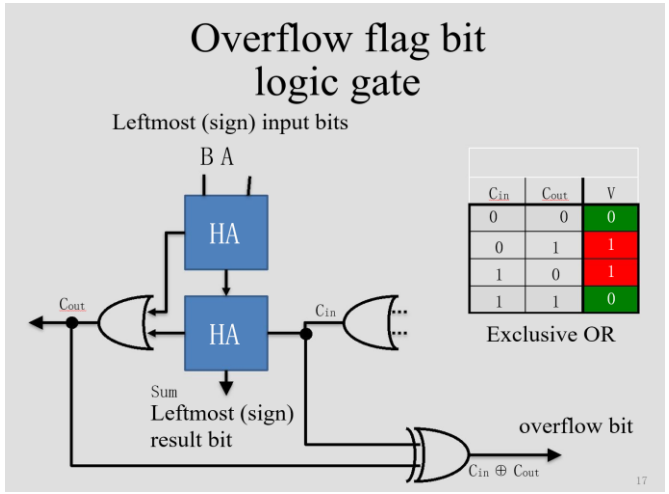
Bit shifting

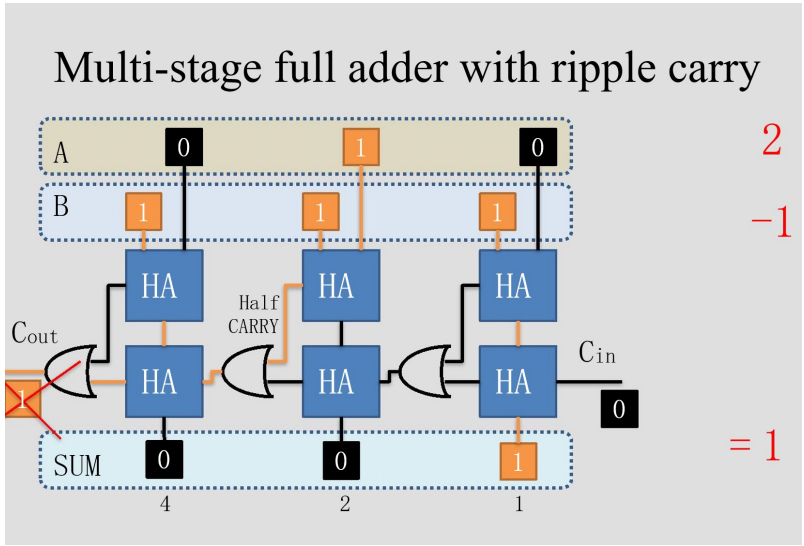
As we've already seen, bit shifting enables simple multiplication/ division by powers of two, and can also speed up addition-based methods of multiplication, Movement of a bit pattern left or right is also widely used in control systems and pattern matching

Basic ALU Logic Design

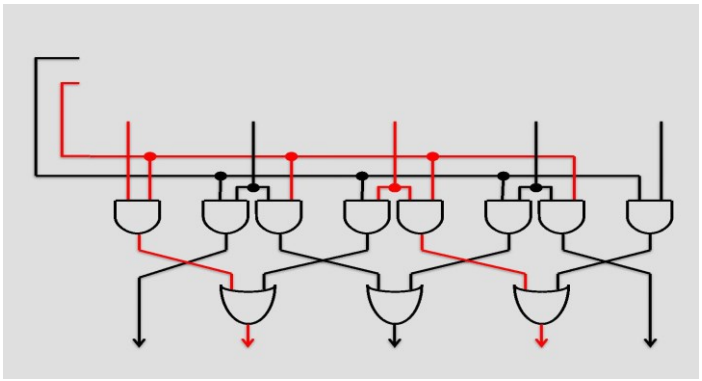


Status Flag





Adder



Shift