TÉCNICO LISBOA

Computer Electronics

Lecture 15: Adder Circuits – Part 1

Typical ALU operations

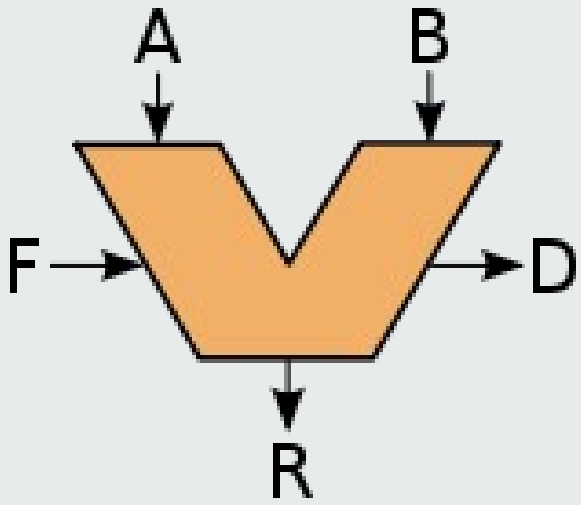
- Arithmetic: Addition, subtraction
- Logical: NOT, AND, OR, XOR, …
- Less common (normally performed by dedicated FUs):
  - Bit shifting: left, right (logical or arithmetic)
  - Multiplication
  - Division

Why only simple operations?

- Complex operations require more hardware whose cost must be justified
- Complex functions can be decomposed as a time sequence of simple operations
- Certain complex functions can be added as needed:
  - Dedicated IP cores integrated in the same chip
  - Dedicated chips integrated in the same board

The origin of the ALU

- John von Neumann proposed it
- EDVAC computer, 1945

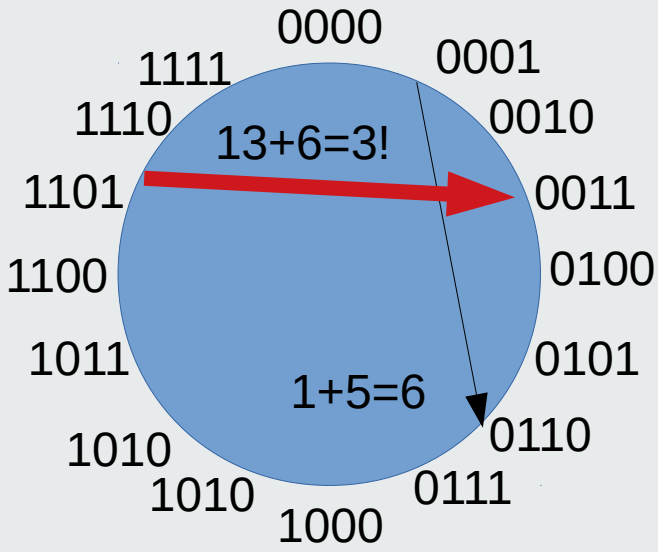


Integer number representation

- Binary: easy to implement and robust to noise
- Resolution: number of bits, N
- Unsigned numbers: 0, 1, 2, …., 2^N-1
- Signed numbers: -2^{N-1}, $-2^{N-1}+1$, …, -1, 0, +1, …, $2^{N-1}-1$
- Signed numbers: 2’s complement representation normally used

How to compute the 2's complement

- Method 0: (software)
  - complement2two(x) = 2^N-x
- Method 1: (hardware)
  - Invert all bits
  - Add 1
- Method 2: (hand, hardware?)
  - Scan number from right to left
  - Up to the first 1 inclusively: leave unchanged
  - Invert all remaining bits



TÉCNICO LISBOA

Adder circuits

- Addition is the most basic arithmetic operation
- Basis for more complex operations, like multiplication, division, etc
- Different possible implementations
  - Simple implementation: small and *cool* (low power), high combinational delay
  - Elaborate implementation: big and *hot* (high power), low combinational delay
- Technology used influences architecture choice
  - FPGA use the simplest, for example

Unsigned addition

$1010+0011=10_{10}+3_{10}$

1010 → 10

+0011 → 3

1101 → 13

0010 -> Carry

Overflow happens if result < some operand

1010 -> 10

+1010 -> 10

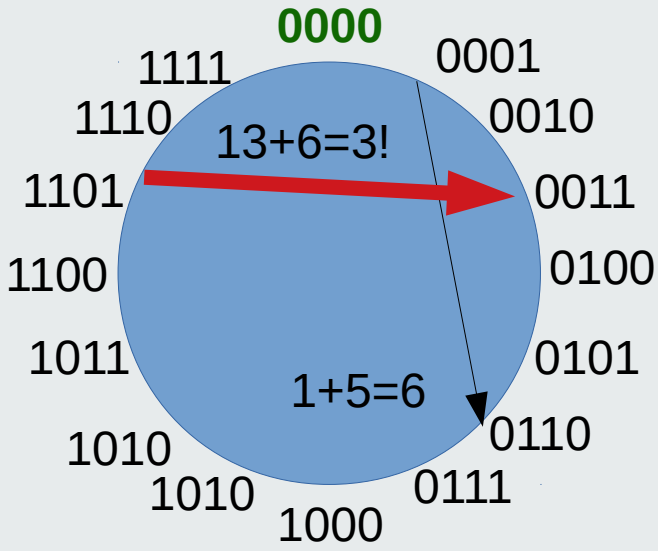
0100 -> 4 (!)

1010 -> Carry

Visualizing and computing <u>unsigned</u> addition and overflow

10 + 12 = 6!

```
  1010
+1100
--------
  0110
  1000 → carry on MSB is overflow
```



Signed addition

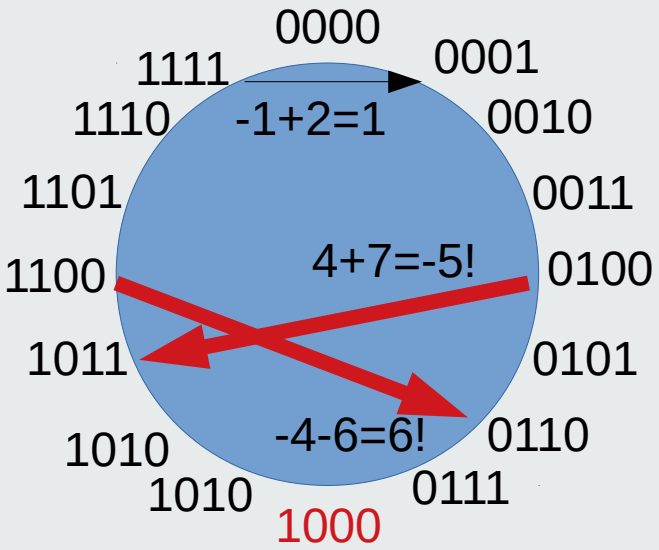
-3 - 2 = -5

```
  1101
+1110
-------
  1011
  1100  → carry
```

Overflow happens if addition of 2 positives (negatives) is a negative (positive)

```
  1010  → -6, negative
+1011  → -5, negative
--------
  0111  → 7!, positive!
  1010  → carry
```

Visualizing and computing signed addition and overflow



How to flag signed addition overflow in hardware?

Overflow = (result_sign XOR a_sign) AND (a_sign XNOR b_sign)

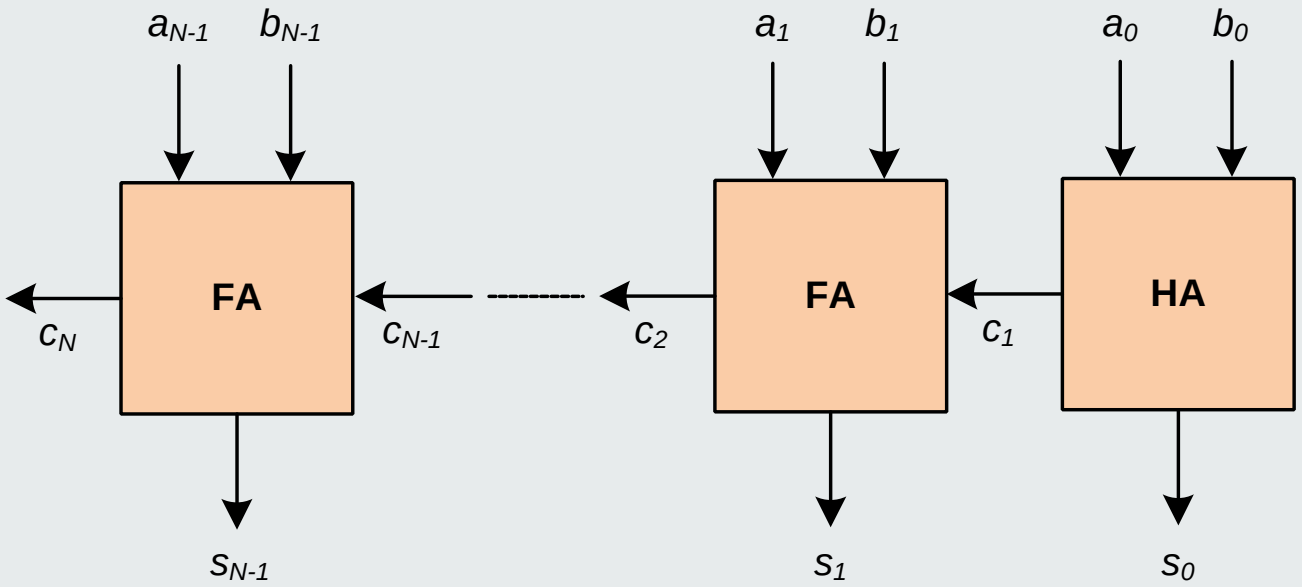
Equivalent to
Overflow = C(n-1) XOR C(n-2)

Can you prove this?

Unsigned vs. signed addition

- Same adder hardware is used!!
  - Greatest advantage of 2s complement!
  - Unsigned and signed interpreted differently but addition operation is the same
- Overflow detection
  - Unsigned: Carry(n-1)
  - Signed: Carry(n-1) XOR Carry(n-2)

Ripple carry adder



Half Adder (HA) truth table

- No input carry

$$s_i = a_i \overline{b_i} + \overline{a_i} b_i = a_i \oplus b_i$$

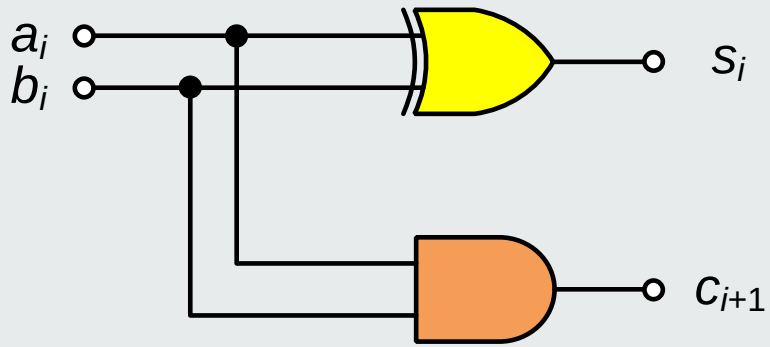
$$c_{i+1} = a_i b_i$$

Inputs		Outputs	
a_i	b_i	c_0	s_{i+1}
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Half adder logic circuit

$$s_i = a_i \overline{b_i} + \overline{a_i} b_i = a_i \oplus b_i$$

$$c_{i+1} = a_i b_i$$



Area: 2 gates
Delay: 1 gate

Full Adder truth table

- Input carry

$$s_i = a_i \oplus b_i \oplus c_i$$

$$c_{i+1} = a_i b_i + (a_i + b_i) c_i$$

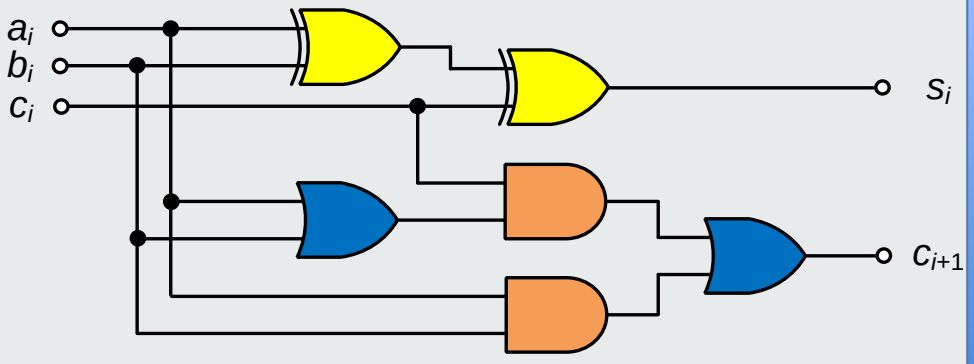
$a_i b_i = g_i$ Generate

$a_i + b_i = g_i$ Propagate

Inputs			Outputs	
a_i	b_i	c_i	c_0	s_{i+1}
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

Full Adder logic circuit

- Area: 6 (2-input) gates
- Propagation delay: 3 gate delays

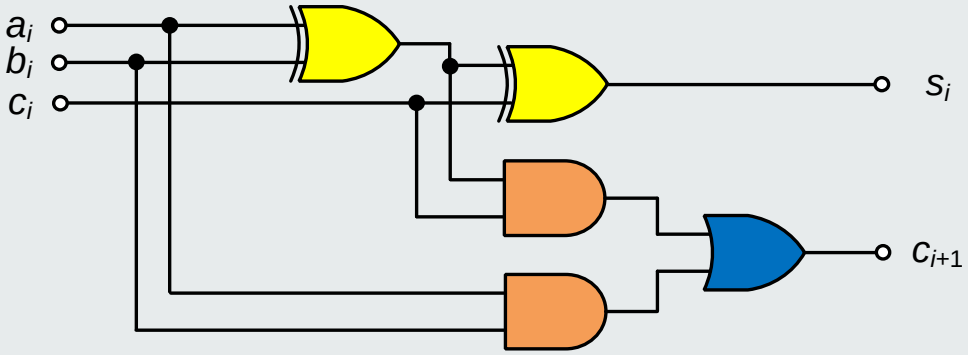


Carry delays will add up when blocks chained!

Full Adder simplification

$$c_{i+1} = a_i b_i + (a_i + b_i) c_i$$
$$c_{i+1} = a_i b_i + (a_i \oplus b_i + a_i b_i) c_i$$
$$c_{i+1} = a_i b_i (1 + c_i) + (a_i \oplus b_i) c_i$$
$$c_{i+1} = a_i b_i + (a_i \oplus b_i) c_i$$

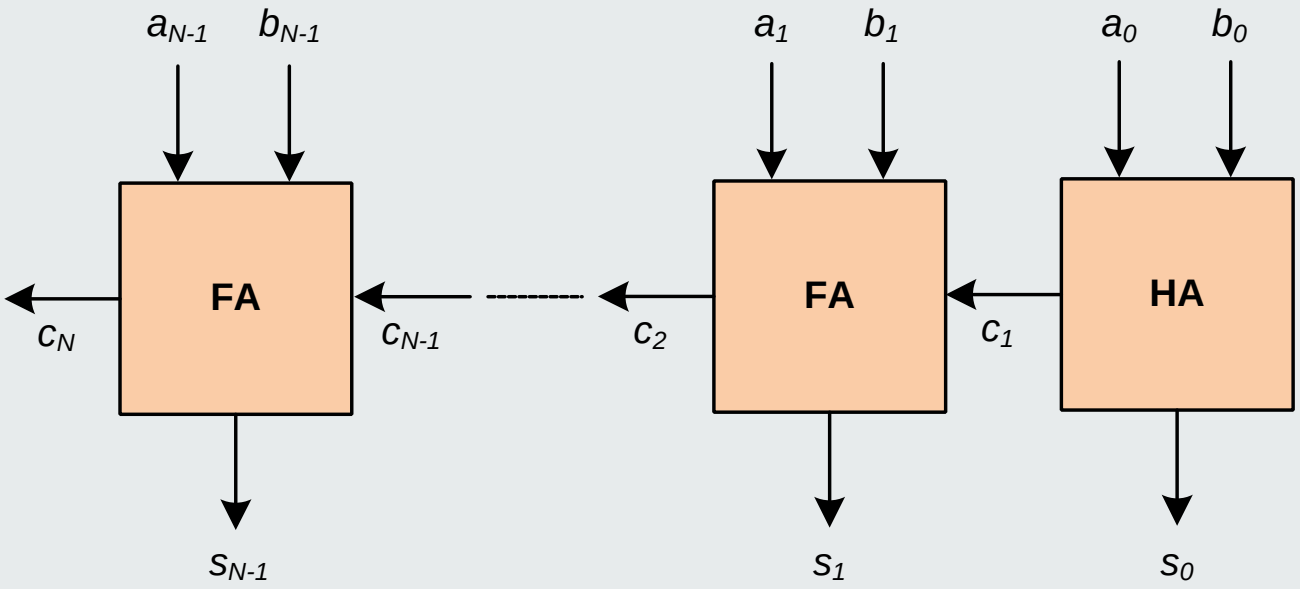
- Area: 5 gates (1 less!)
- Delay: 3 gates (same)



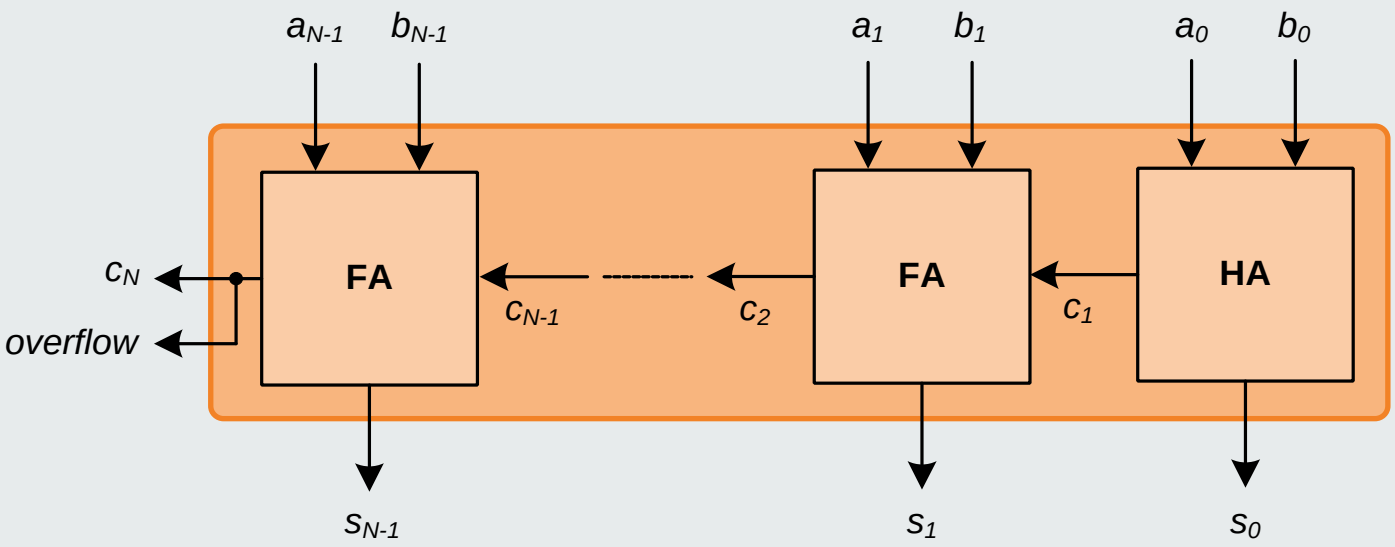
Carry delays will add up!

Ripple carry adder

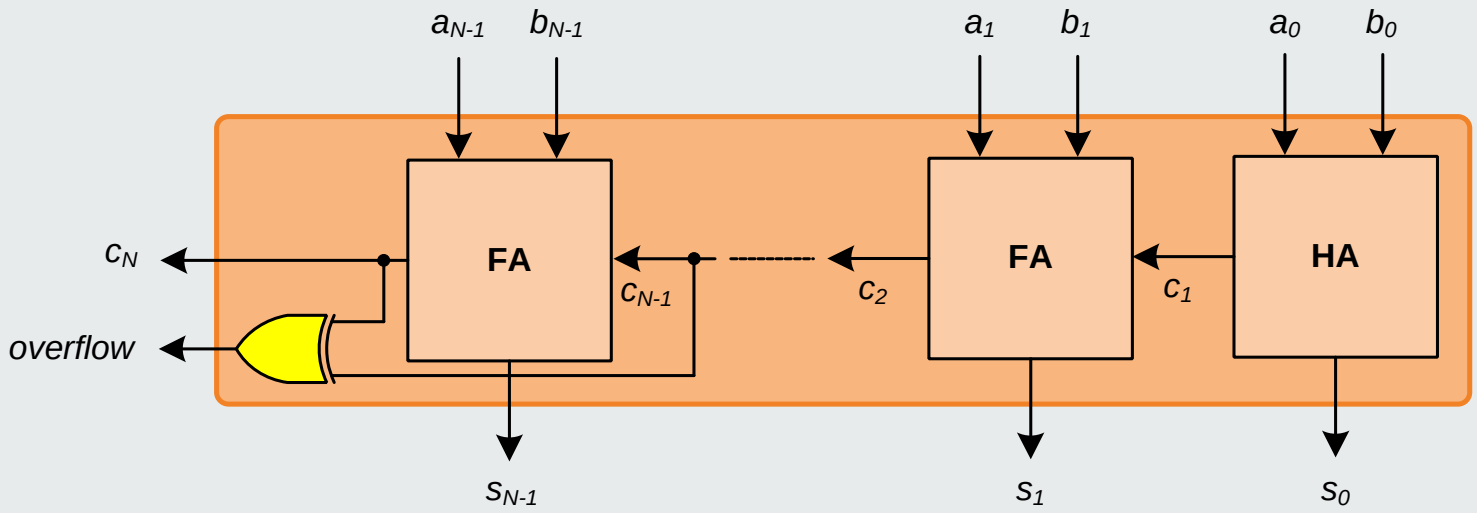
- Compute area and delay



RCA: compute unsigned overflow



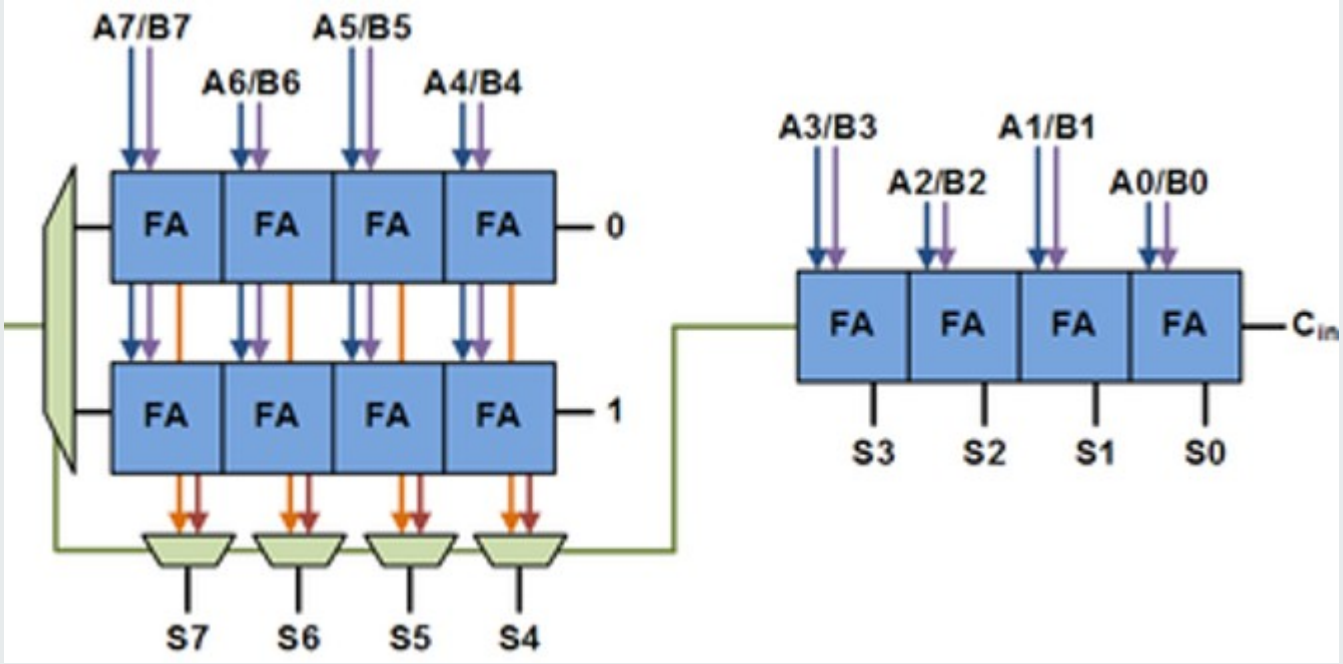
RCA: compute signed overflow



RCA: advantage and disadvantage

- Advantage: simplest adder circuit, lowest area and power dissipation
- Disadvantage: it is slow
  - Delay is O(N), where N is number of bits
  - Because carry computation is sequential:
    - Wait for C(i-1) to compute C(i)

Carry-select adder



TÉCNICO LISBOA

CSA algorithm summary

- Split N-bit addition into N/2-bit additions
- Use 1 RCA for lower part (A)
- Use 2 RCAs for higher part (B and C)
  - B assumes carry from lower part is 0
  - C assumes carry from lower part is 1
- Carry A selects the output from either B or C, which had been computed in parallel with A
- CSA delay is half RCA delay + mux delay
- CSA is 100% faster than RCA but 50% larger

Example 32-bit/4-stage:

A(CSA)/A(RCA)= (32+24)FA/32FA = 7/4
Delay(RCA) / Delay(CSA) = 32FA/8FA= 4

K-stage CSA