# Adder

# Adder

- **Review 01_numbers.ppt**

# اعمال ریاضی باینری: جمع

- قوانین: مانند جمع دسیمال
- با این تفاوت که 10 = 1+1 ← تولید نقلی
  - 0+0 = 0c0 (sum 0 with carry 0)
  - 0+1 = 1+0 = 1c0
  - 1+1 = 0c1
  - 1+1+1 = 1c1

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

| | | | | | | |
|---|---|---|---|---|---|---|
| Carry | 1 | 1 | 1 | 1 | 1 | 0 |
| Augend | 0 | 0 | 1 | 0 | 0 | 1 |
| Addend | 0 | 1 | 1 | 1 | 1 | 1 |
| Result | 1 | 0 | 1 | 0 | 0 | 0 |

# نمایش اعداد

$N^* = 2^n - N$

## مکمل 2:

Example: Twos complement of 7

$$\begin{array}{rr} 2^4 = & 10000 \\ \text{sub } 7 = & \underline{0111} \\ & 1001 \end{array} = \text{repr. of } -7$$

Example: Twos complement of -7

$$\begin{array}{rr} 2^4 = & 10000 \\ \text{sub } -7 = & \underline{1001} \\ & 0111 \end{array} = \text{repr. of } 7$$

Shortcut method:

Twos complement = bitwise complement + 1

0111 -> 1000 + 1 -> 1001 (representation of -7)

1001 -> 0110 + 1 -> 0111 (representation of 7)

# جمع و تفريق
# مكمل 2

| | | | |
|---|---|---|---|
| 4 | 0100 | -4 | 1100 |
| + 3 | 0011 | + (-3) | 1101 |
| 7 | 0111 | -7 | 11001 |

**If**
**(carry-in to sign = carry-out )**
**then ignore carry**

**if**
**(carry-in ≠ carry-out)**
**then overflow**

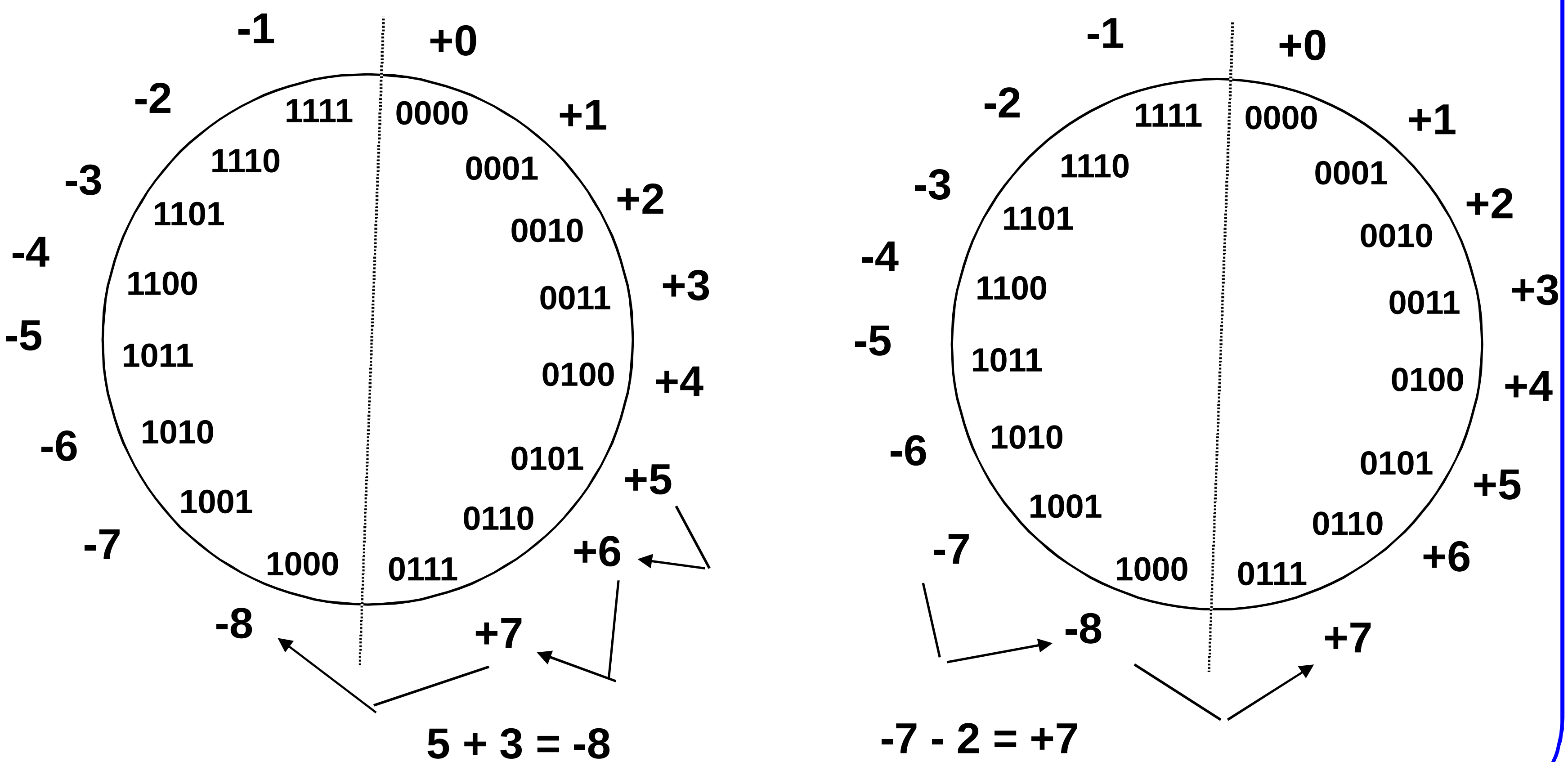
| | | | |
|---|---|---|---|
| 4 | 0100 | -4 | 1100 |
| - 3 | 1101 | + 3 | 0011 |
| 1 | 10001 | -1 | 1111 |

**Simpler addition scheme makes twos complement the most common choice for integer number systems within digital systems**

*Overflow Conditions* سرریز

**Add two positive numbers to get a negative number**

**or two negative numbers to get a positive number**



## *Overflow Conditions*

سرریز

| | |
|---|---|
| | 0 1 1 1 |
| 5 | 0 1 0 1 |
| 3 | 0 0 1 1 |
| -8 | 1 0 0 0 |

Overflow

| | |
|---|---|
| | 1 0 0 0 |
| -7 | 1 0 0 1 |
| -2 | 1 1 0 0 |
| 7 | 1 0 1 1 1 |

Overflow

| | |
|---|---|
| | 0 0 0 0 |
| 5 | 0 1 0 1 |
| 2 | 0 0 1 0 |
| 7 | 0 1 1 1 |

No overflow

| | |
|---|---|
| | 1 1 1 1 |
| -3 | 1 1 0 1 |
| -5 | 1 0 1 1 |
| -8 | 1 1 0 0 0 |

No overflow

**Overflow when carry in to sign ≠ carry out**

# Half Adder (HA)

- Add single bits

| Ai | Bi | Sum | Carry |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

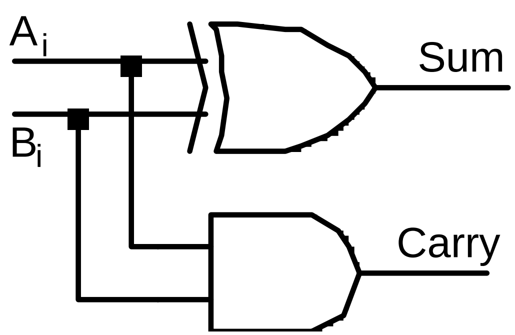
| Bi \ Ai | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

$$\text{Sum} = \overline{Ai}\,Bi + Ai\,\overline{Bi}$$

$$= Ai \oplus Bi$$

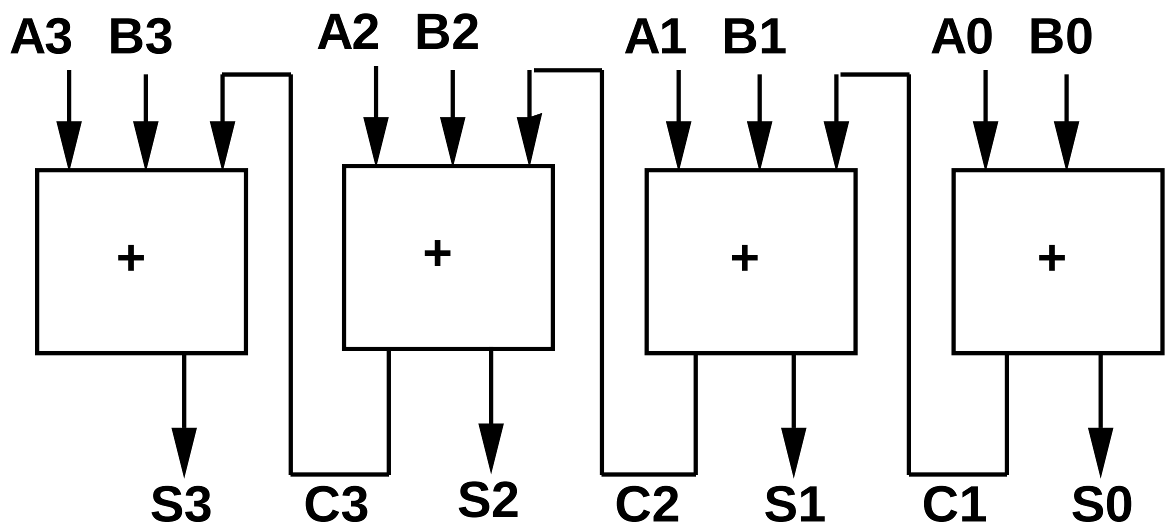
| Bi \ Ai | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

$$\text{Carry} = Ai \,.\, Bi$$



# Full Adder

- Add multiple bits



# Full Adder (FA)

| A | B | $C_I$ | S | $C_O$ |
|---|---|---|---|---|
| 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

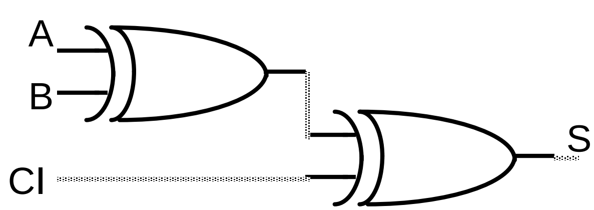
| $C_I$ \ A B | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

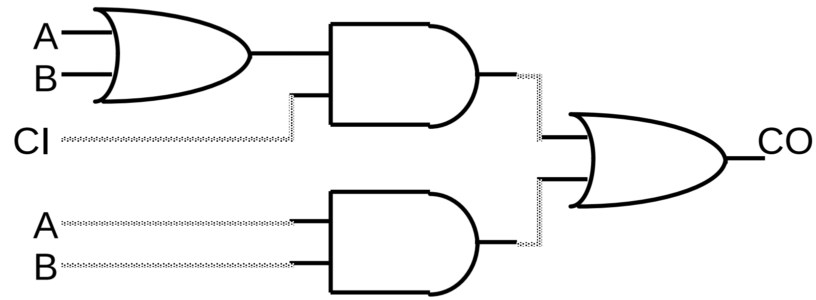
$C_O$

| $C_I$ \ A B | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

$S = C_I \text{ xor } A \text{ xor } B$

$C_O = B\,C_I + A\,C_I + A\,B = C_I\,(A + B) + A\,B$

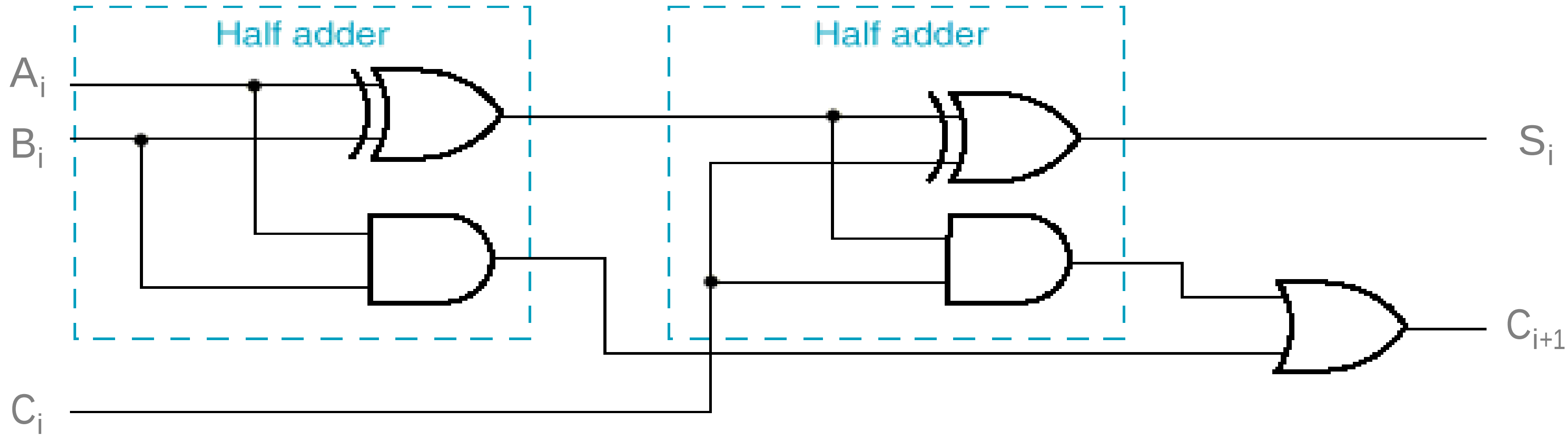


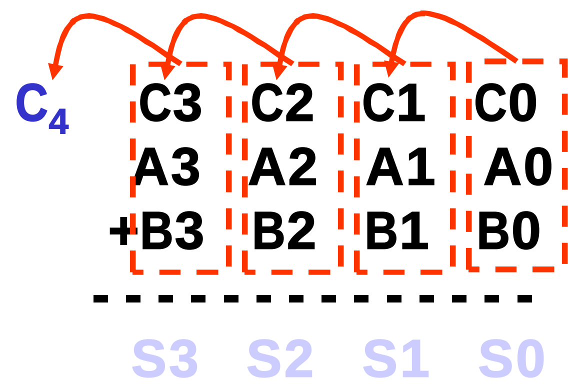
# Full Adder Using 2 Half Adders

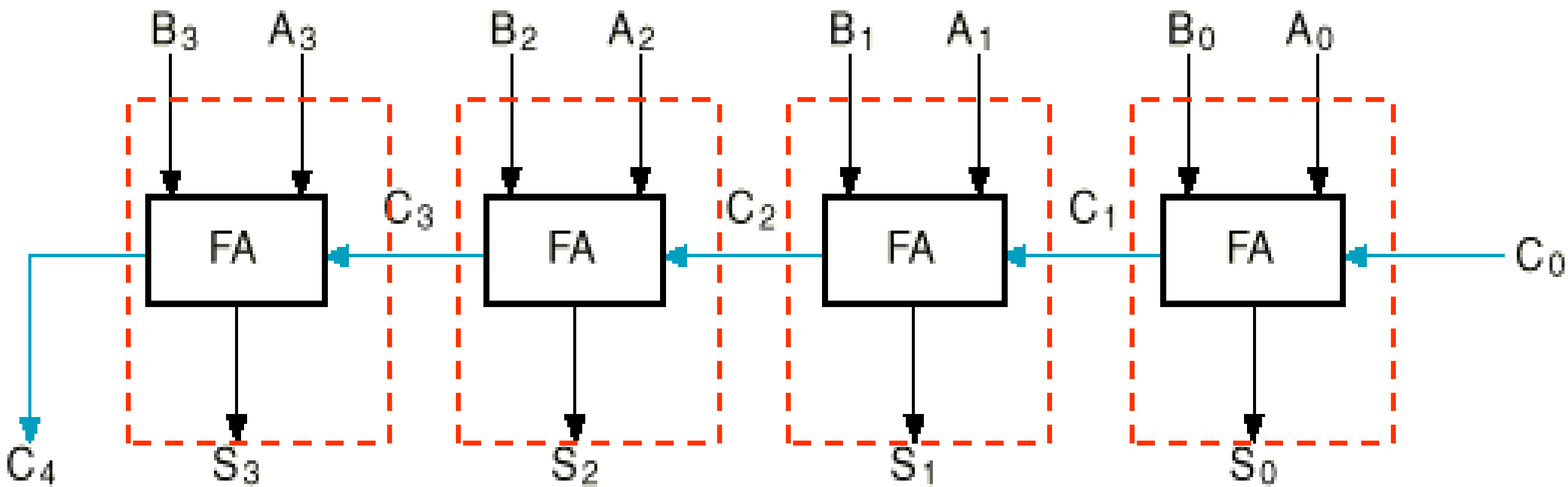
- A full adder can also be realized with two half adders and an OR gate, since $C_{i+1}$ can also be expressed as:
- $C_{i+1} = A_iB_i + (Ai \oplus B_i)C_i$

and $S_i = (A_i \oplus B_i) \oplus C_i$



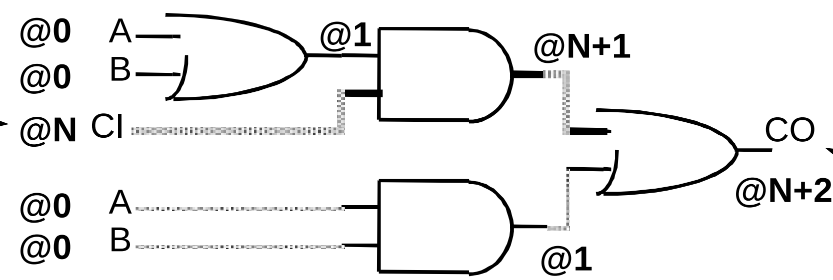
# Example: 4-bit Ripple Carry Adder



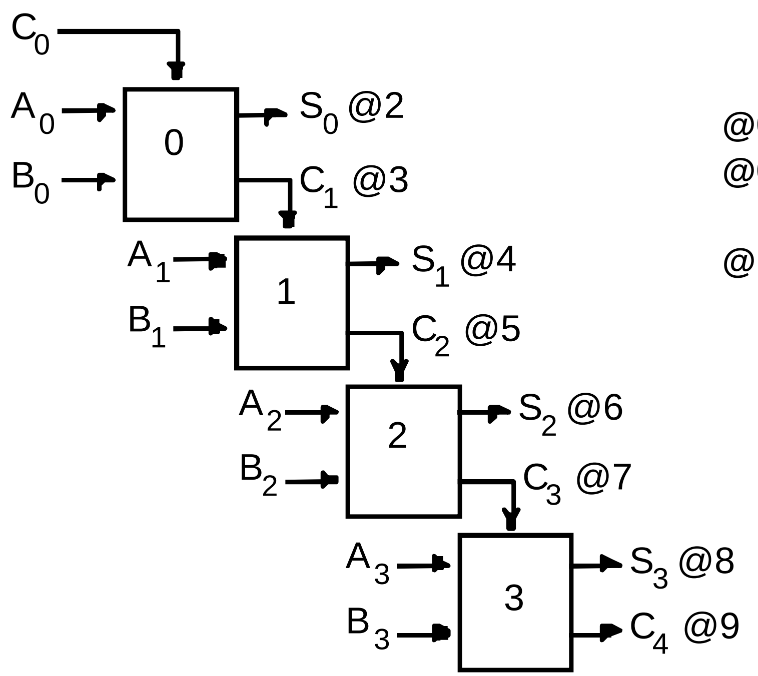
# Delay Analysis of Ripple Adder

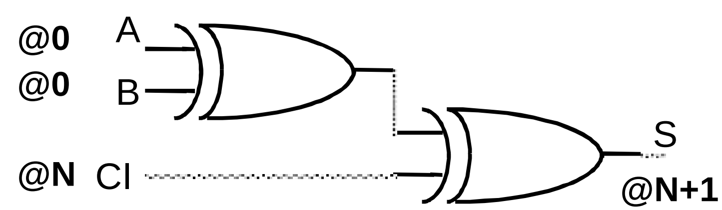
**late arriving signal**




**two gate delays to compute CO**

**4 stage adder**



# Carry Lookahead Adder

- **Carry Generate**
  - **$G_i = A_i B_i$** *must generate carry when A = B = 1*
- **Carry Propagate**
  - **$P_i = A_i \text{ xor } B_i$** *carry-in will equal carry-out here*

- ***Sum and Carry can be reexpressed in terms of generate/propagate:***

$$S_i = A_i \text{ xor } B_i \text{ xor } C_i = P_i \text{ xor } C_i$$

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i$$

$$= A_i B_i + C_i (A_i + B_i)$$

$$= A_i B_i + C_i (A_i \text{ xor } B_i)$$

$$= G_i + C_i P_i$$

# Carry Lookahead Adder

$C1 = G0 + P0\ C0$

$C2 = G1 + P1\ C1$
$\quad = G1 + P1\ (G0 + P0\ C0)$
$\quad = G1 + P1\ G0 + P1\ P0\ C0$

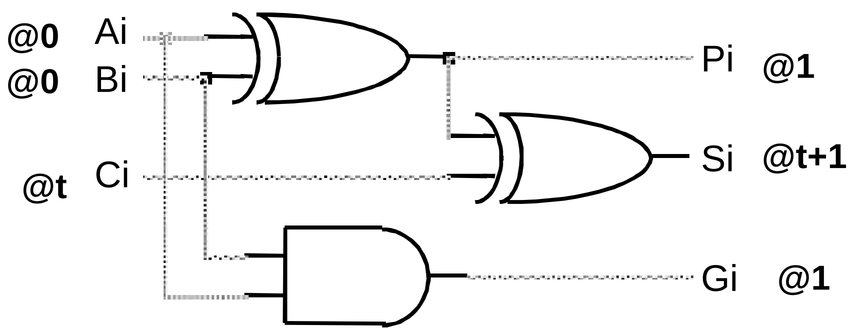
$C3 = G2 + P2\ C2$
$\quad = G2 + P2\ (G1 + P1\ G0 + P1\ P0\ C0)$
$\quad = G2 + P2\ G1 + P2\ P1\ G0 + P2\ P1\ P0\ C0$

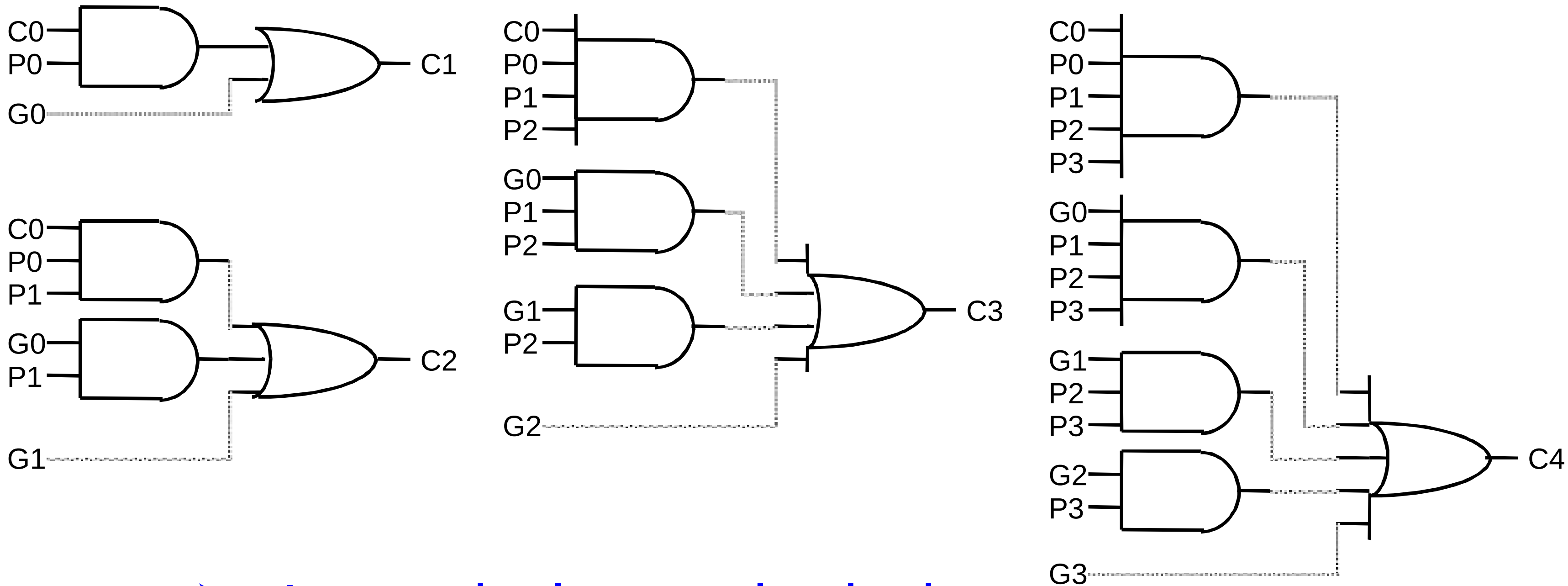
$C4 = G3 + P3\ C3$
$\quad = G3 + P3\ G2 + P3\ P2\ G1 + P3\ P2\ P1\ G0 + P3\ P2\ P1\ P0\ C0$

- Each of the carry equations can be implemented in a two-level logic network

- Variables are the adder inputs and carry in to stage 0!

# CLA

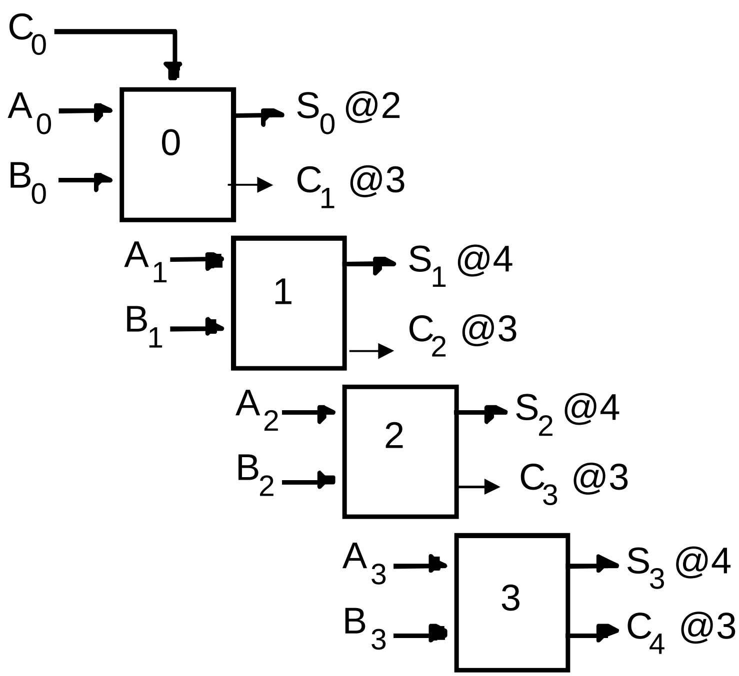


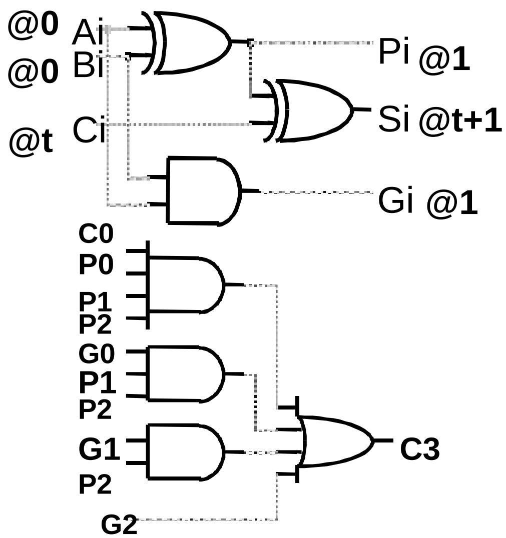

- Increasingly complex logic

# Delay Analysis of CLA

- Ci's are generated independent of N



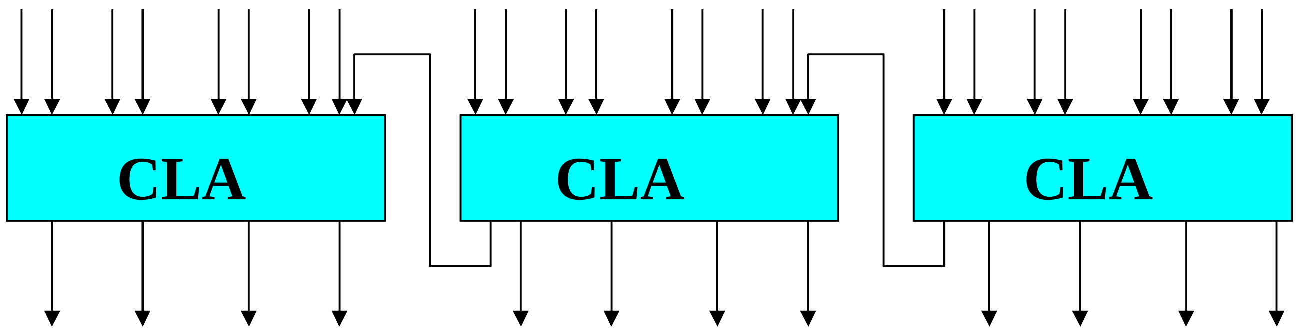

**4 stage adder**




**final sum and carry**

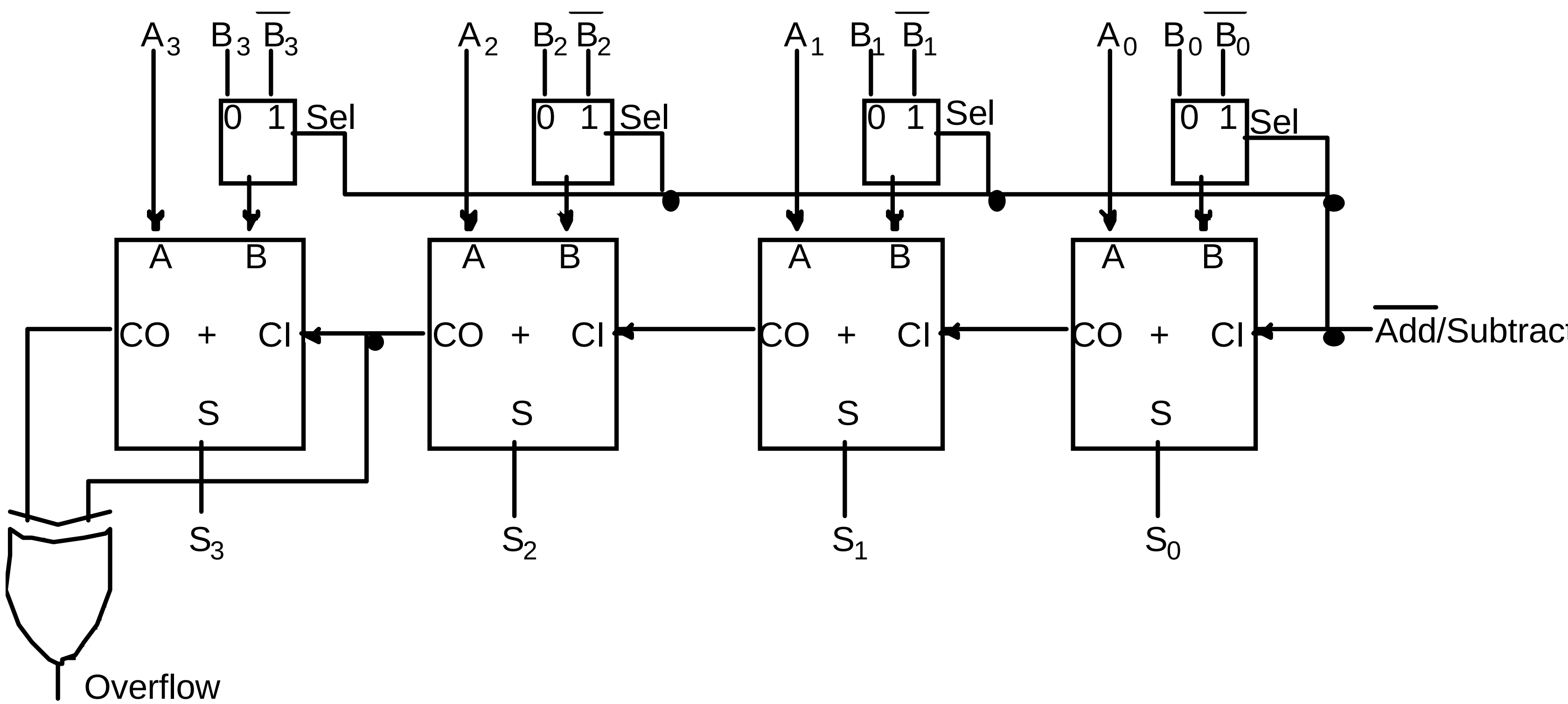
- NOTE: This assumes all gate delay are same.
- Not true, delays depend on fan-ins and fan-out

# Cascaded CLA

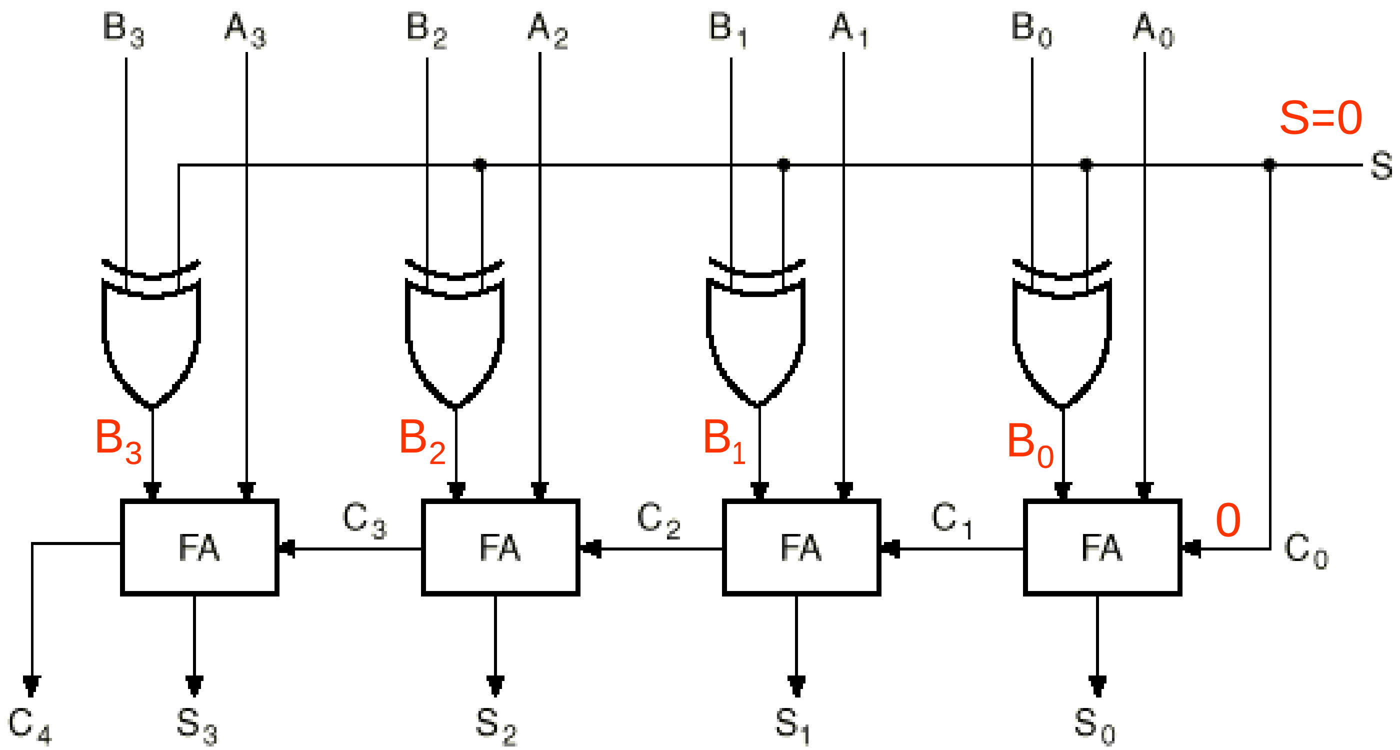


# Adder/Subtractor

**A - B = A + (-B) = A + B' + 1**

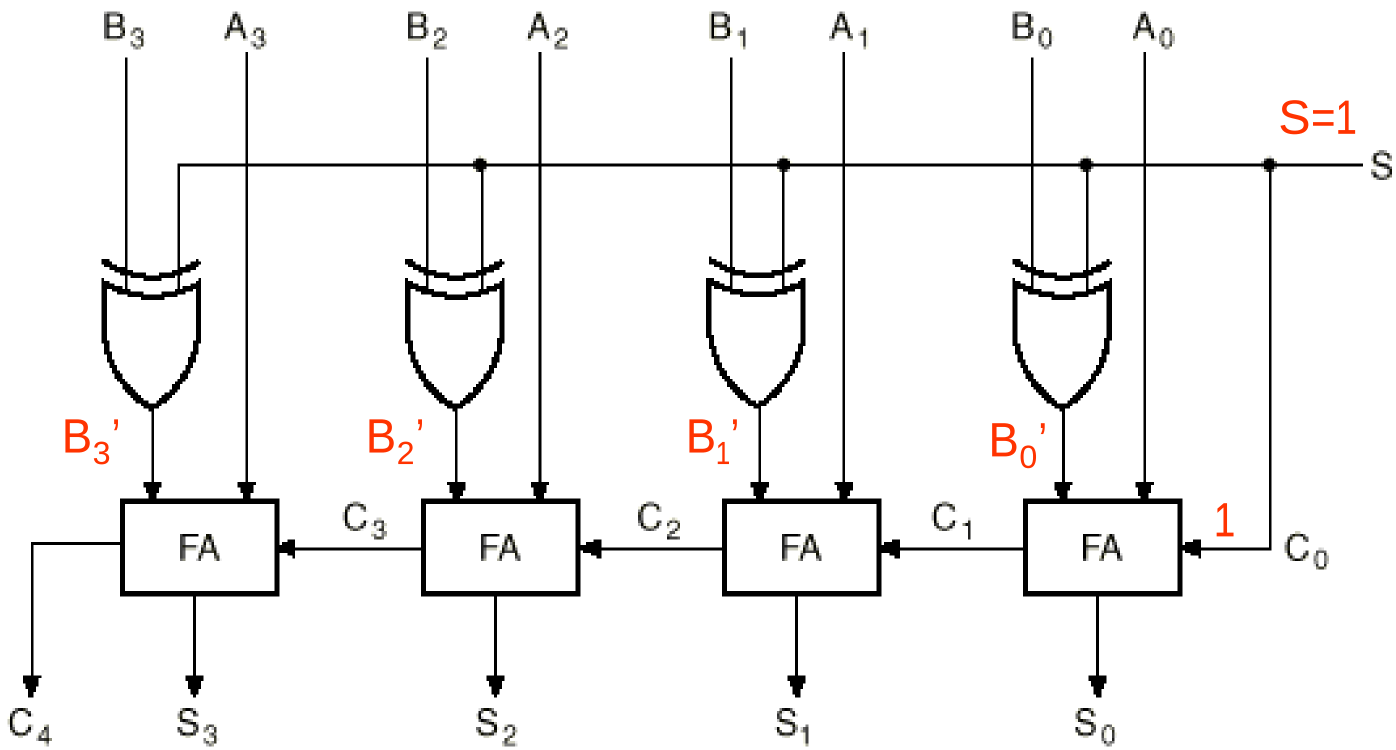


# 4-bit Binary Adder/Subtractor (cont.)




S=0 selects addition

# 4-bit Binary Adder/Subtractor (cont.)




S=1 selects subtraction

# Overflow

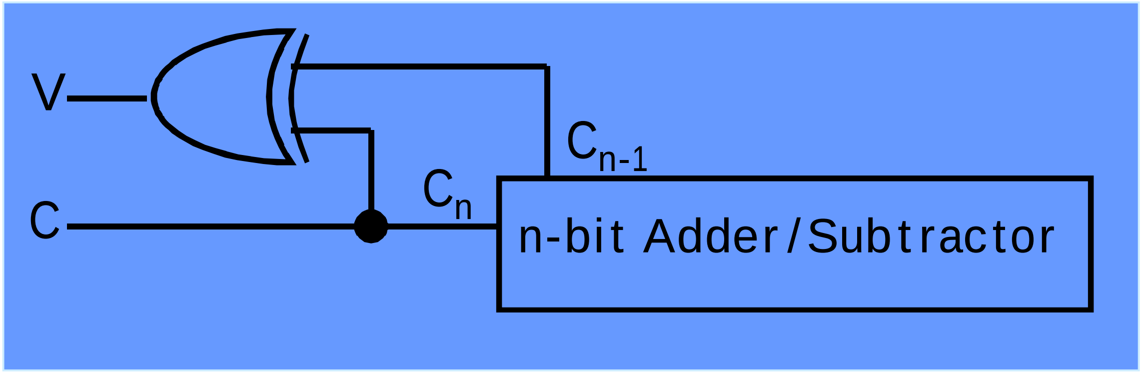
- Overflow can occur ONLY when both numbers have the same sign.

- This condition can be detected when the carry out ($C_n$) is <u>different</u> than the carry at the previous position ($C_{n-1}$).

# The Overflow problem in Signed-2's Complement (cont.)

- **Example 1: $M=65_{10}$ and $N=65_{10}$ (In an 8-bit 2's complement system).**
  - $M = N = 01000001_2$
  - M+N = 10000010 with $C_n$=0. (clearly wrong!)
  - Bring $C_n$ as the MSB to get $010000010_2$ ($130_{10}$) which is correct, but requires 9-bits → overflow occurs.
- **Example 2: $M=-65_{10}$ and $N=-65_{10}$**
  - $M = N = 10111111_2$
  - M+N = 01111110 with $C_n$=1. (wrong again!)
  - Bring $C_n$ as the MSB to get $101111110_2$ ($-130_{10}$) which is correct, but also requires 9-bits → overflow occurs.

# Overflow Detection in Signed-2's Complement



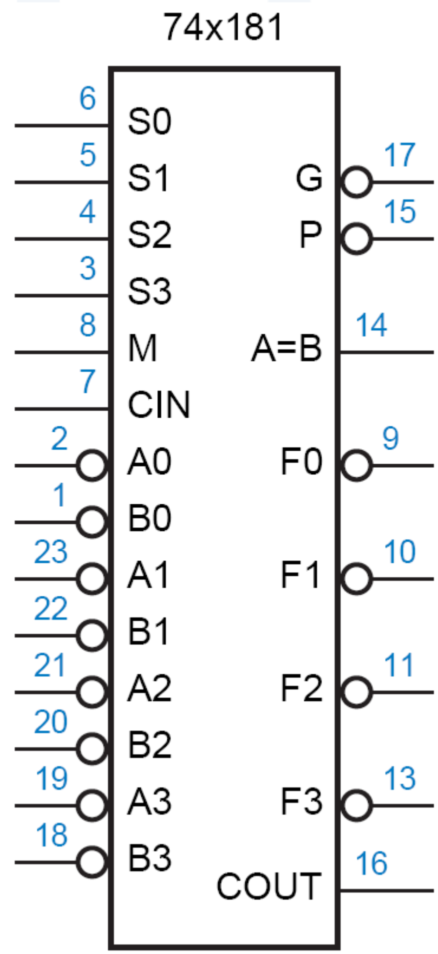

- C =1 indicates overflow condition when adding/subtr. unsigned numbers.
- V=1 indicates overflow condition when adding/subtr. signed-2's complement numbers

# 4-Bit ALU

- **74181 TTL ALU**
  - Arithmetic-Logic Unit

| Selection | | | | M = 1 | M = 0, Arithmetic Functions | |
|---|---|---|---|---|---|---|
| S3 | S2 | S1 | S0 | Logic Function | Cn = 0 | Cn = 1 |
| 0 | 0 | 0 | 0 | F = not A | F = A minus 1 | F = A |
| 0 | 0 | 0 | 1 | F = A nand B | F = A B minus 1 | F = A B |
| 0 | 0 | 1 | 0 | F = (not A) + B | F = A (not B) minus 1 | F = A (not B) |
| 0 | 0 | 1 | 1 | F = 1 | F = minus 1 | F = zero |
| 0 | 1 | 0 | 0 | F = A nor B | F = A plus (A + not B) | F = A plus (A + not B) plus 1 |
| 0 | 1 | 0 | 1 | F = not B | F = A B plus (A + not B) | F = A B plus (A + not B) plus 1 |
| 0 | 1 | 1 | 0 | F = A xnor B | F = A minus B minus 1 | F = (A + not B) plus 1 |
| 0 | 1 | 1 | 1 | F = A + not B | F = A + not B | F = A minus B |
| 1 | 0 | 0 | 0 | F = (not A) B | F = A plus (A + B) | F = (A + not B) plus 1 |
| 1 | 0 | 0 | 1 | F = A xor B | F = A plus B | F = A plus (A + B) plus 1 |
| 1 | 0 | 1 | 0 | F = B | F = A (not B) plus (A + B) | F = A (not B) plus (A + B) plus 1 |
| 1 | 0 | 1 | 1 | F = A + B | F = (A + B) | F = (A + B) plus 1 |
| 1 | 1 | 0 | 0 | F = 0 | F = A | F = A plus A plus 1 |
| 1 | 1 | 0 | 1 | F = A (not B) | F = A B plus A | F = AB plus A plus 1 |
| 1 | 1 | 1 | 0 | F = A B | F= A (not B) plus A | F = A (not B) plus A plus 1 |
| 1 | 1 | 1 | 1 | F = A | F = A | F = A  plus 1 |

# 4-Bit ALU


74x181
6
S0
5
S1
4
S2
3
S3
8
M
7
CIN
2
A0
1
B0
23
A1
22
B1
21
A2
20
B2
19
A3
18
B3
17
G
15
P
14
A=B
9
F0
10
F1
11
F2
13
F3
16
COUT

# BCD Addition

**Addition:**

5 = 0101

3 = 0011

1000 = 8

5 = 0101

8 = 1000

1101 = 13!

Problem when digit sum exceeds 9

**Solution: add 6 (0110) if sum exceeds 9!**

5 = 0101

8 = 1000

1101

6 = 0110

1 0011 = 1 3 in BCD

9 = 1001

7 = 0111

1 0000 = 16 in binary

6 = 0110

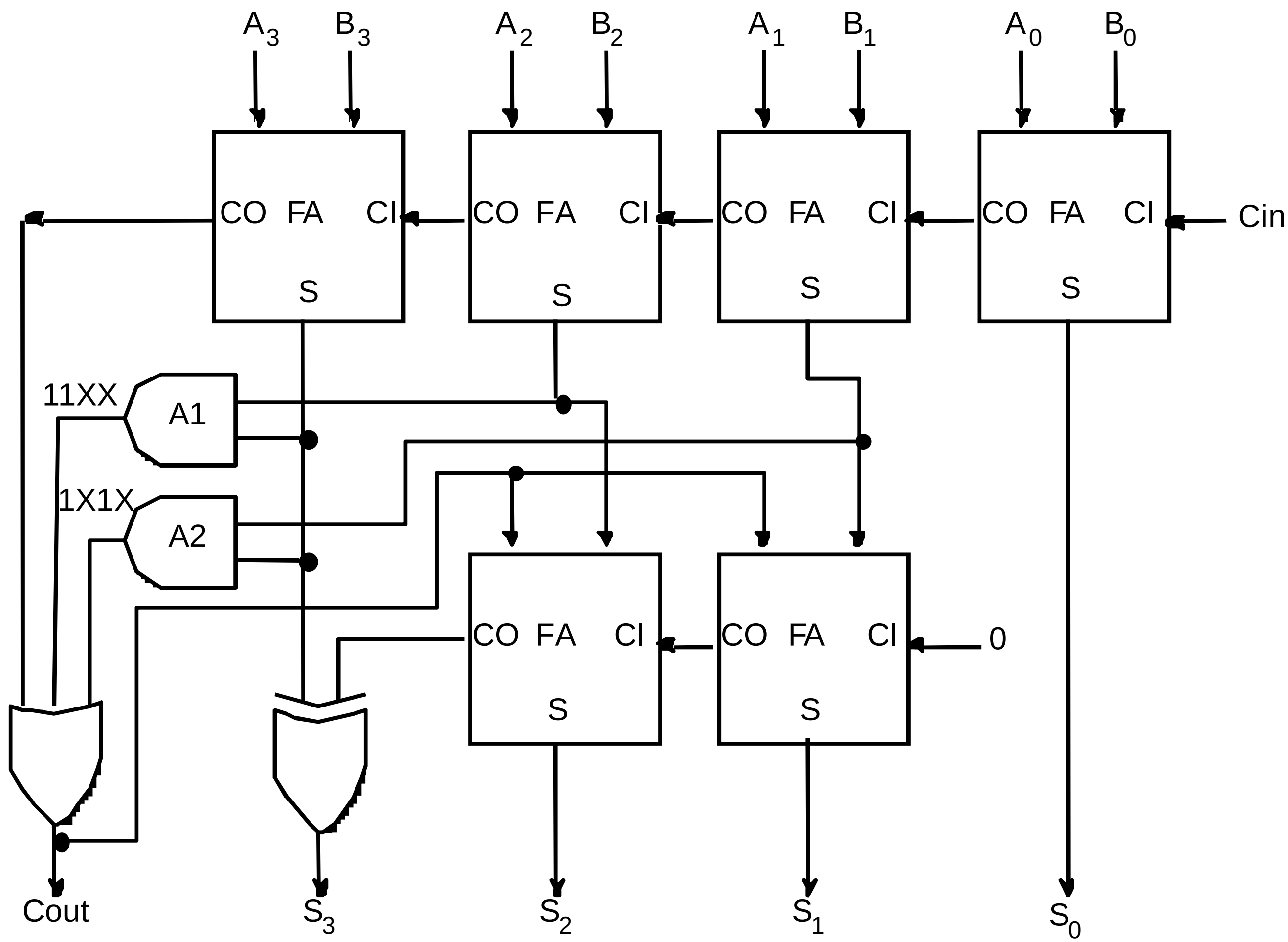
1 0110 = 1 6 in BCD

# اعداد در مبناهای مختلف

| Decimal (base 10) | Binary (base 2) | Octal (base 8) | Hexadecimal (base 16) |
|---|---|---|---|
| 00 | 0000 | 00 | 0 |
| 01 | 0001 | 01 | 1 |
| 02 | 0010 | 02 | 2 |
| 03 | 0011 | 03 | 3 |
| 04 | 0100 | 04 | 4 |
| 05 | 0101 | 05 | 5 |
| 06 | 0110 | 06 | 6 |
| 07 | 0111 | 07 | 7 |
| 08 | 1000 | 10 | 8 |
| 09 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

**Add 0110 to sum whenever it exceeds 1001 (11XX or 1X1X)**

# BCD Adder




**Add 0110 to sum whenever it exceeds 1001 (11XX or 1X1X)**