Combinational Logic Design
Arithmetic Functions and Circuits

Overview

- Binary Addition
  - Half Adder
  - Full Adder
  - Ripple Carry Adder
  - Carry Look-ahead Adder
- Binary Subtraction
  - Binary Subtractor
  - Binary Adder-Subtractor
- Subtraction with Complements
  - Complements (2's complement and 1's complement)
  - Binary Adder-Subtractor
- Signed Binary Numbers
  - Signed Numbers
  - Signed Addition/Subtraction
  - Overflow Problem
- Binary Multipliers
- Other Arithmetic Functions

1-bit Addition

- Performs the addition of two binary bits.
- Four possible operations:
  - 0+0=0
  - 0+1=1
  - 1+0=1
  - 1+1=10
- Circuit implementation requires 2 outputs; one to indicate the ***sum*** and another to indicate the ***carry***.

Half Adder

- Performs 1-bit addition.
- Inputs: A_0, B_0
- Outputs: S_0, C_1
- Index indicates significance, 0 is for LSB and 1 is for the next higher significant bit.
- Boolean equations:
  - $S_0 = A_0B_0' + A_0'B_0 = A_0 \oplus B_0$
  - $C_1 = A_0B_0$

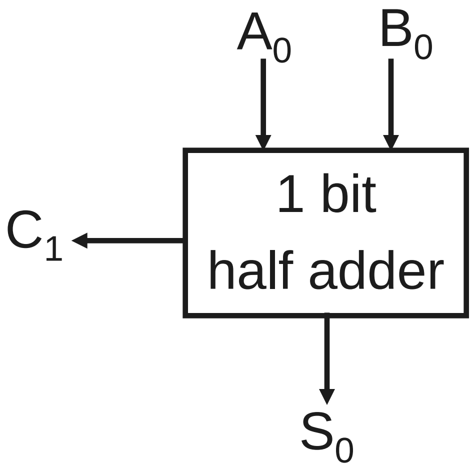
Truth Table

A_0	B_0	S_0	C_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

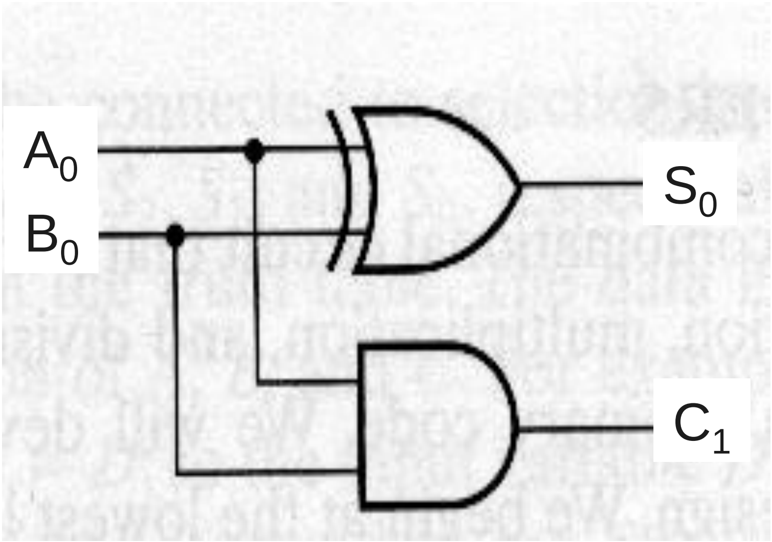
Half Adder (cont.)

- $S_0 = A_0B_0' + A_0'B_0 = A_0 \oplus B_0$
- $C_1 = A_0B_0$

Block Diagram



Logic Diagram



n-bit Addition

- Design an n-bit binary adder which performs the addition of two n-bit binary numbers and generates a n-bit sum and a carry out.

- Example: Let n=4

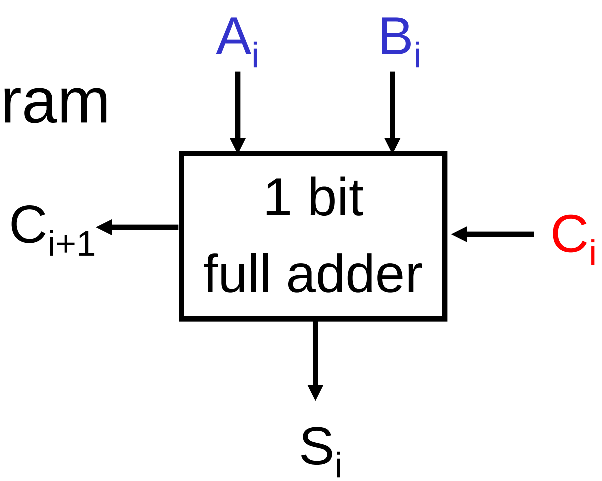
C_{out}	C_3	C_2	C_1	C_0		1	1	0	1	0
	A_3	A_2	A_1	A_0			1	1	0	1
	$+B_3$	B_2	B_1	B_0			+1	1	0	1
	------	------	------	------			----	----	----	----
	S_3	S_2	S_1	S_0			1	0	1	0

- **Notice that in each column we add 3 bits!**

Full Adder

- Combinational circuit that performs the additions of 3 bits (two bits and a carry-in bit).
- Full Adder is used for addition of n-bit binary numbers (for higher-order bit addition).

Block Diagram



Truth Table

A_i	B_i	C_i		S_i	C_{i+1}
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

Full Adder (cont.)

- K-maps:

K-map for S_i

$A_i \backslash B_iC_i$	00	01	11	10
0		1		1
1	1		1	

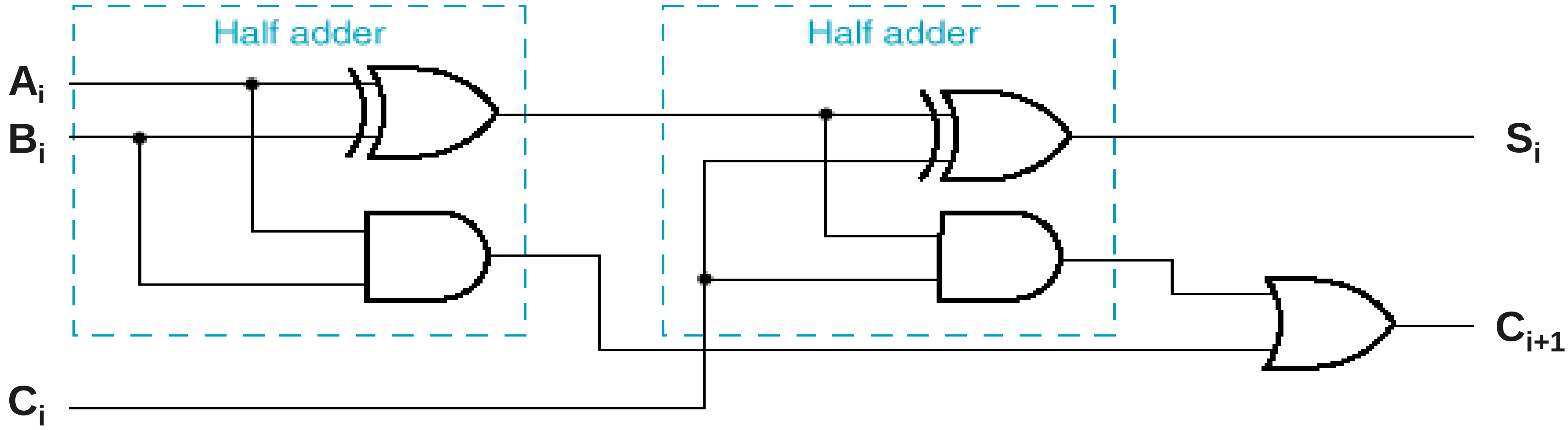
K-map for C_{i+1}

$A_i \backslash B_iC_i$	00	01	11	10
0			1	
1		1	1	1

- Boolean equations:
  - $C_{i+1} = A_iB_i + A_iC_i + B_iC_i$
  - $S_i \;= A_iB_i'C_i' + A_i'B_i'C_i + A_i'B_iC_i' + A_iB_iC_i$
    $= A_i \oplus B_i \oplus C_i$
- You can design full adder circuit **directly** from the above equations (requires 3 ANDs and 2 OR for C_{i+1} and 2 XORs for S_i)
- Can we do better?

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:
  - $\mathbf{C_{i+1}} = A_iB_i + A_iC_i + B_iC_i$
    $= A_iB_i + A_i(B_i+B_i')C_i + (A_i+A_i')B_iC_i$
    $= A_iB_i + A_iB_iC_i + A_iB_i'C_i + A_iB_iC_i + A_i'B_iC_i$
    $= A_iB_i(1+C_i+C_i) + C_i(A_iB_i' + A_i'B_i)$
    $= \mathbf{A_iB_i + C_i(A_i \oplus B_i)}$
  - $\mathbf{S_i = A_i \oplus B_i \oplus C_i}$

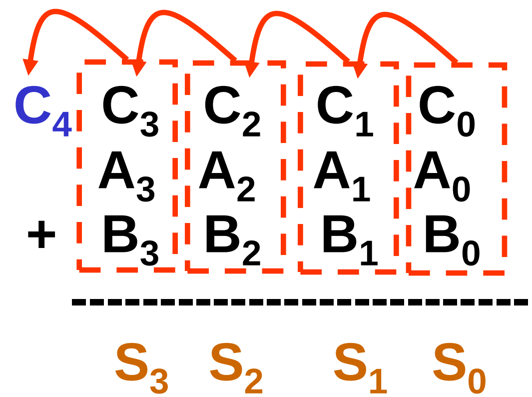


n-bit Combinational Adders

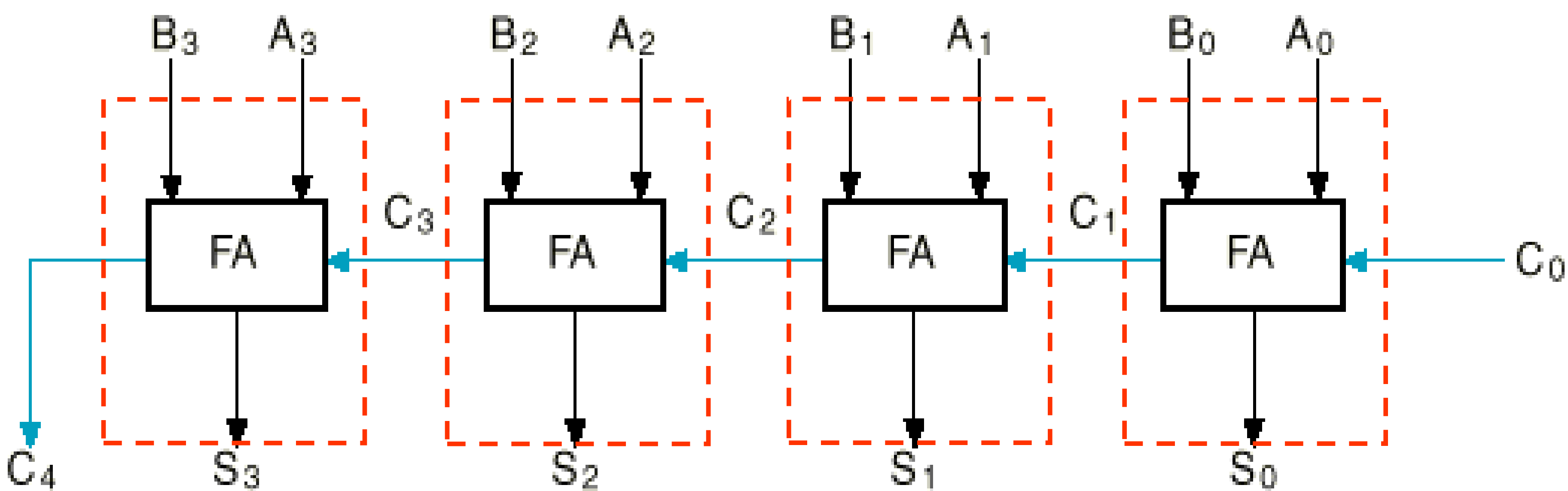
- Perform ***parallel*** addition of n-bit binary numbers.
- Ripple Carry Adder
  - Simple design.
  - Slow circuit. Why? (you'll see …)
- Carry Lookahead Adder
  - More complex than ripple-carry adder.
  - Reduces circuit delay.

n-bit Ripple Carry Adder

- Constructed using ***n*** 1-bit full adder blocks in parallel.
- Cascade the full adders so that the carry out from one becomes the carry in to the next higher bit position.

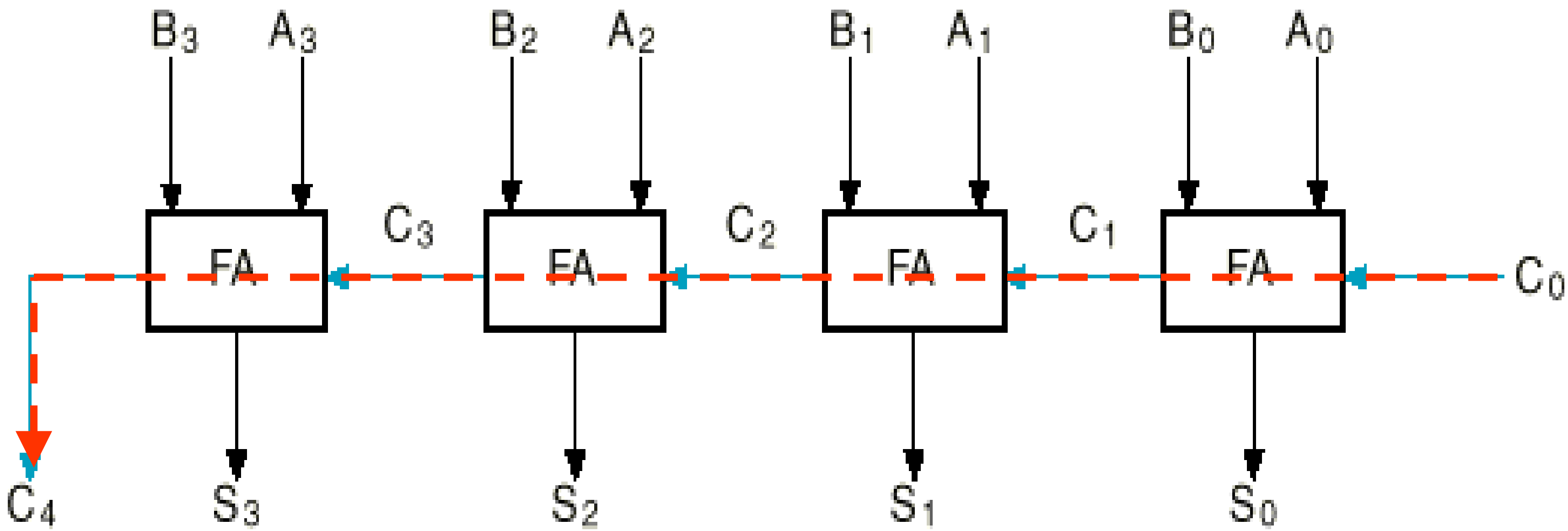


- Example: 4-bit Ripple Carry Adder



Ripple Carry Adder Delay

- Circuit delay in an n-bit ripple carry adder is determined by the delay on the carry path from the LSB (C_0) to the MSB (C_n).
- Let the delay in a 1-bit FA be Δ. Then, the delay of an n-bit ripple carry adder is nΔ.



Carry Look-ahead Adder

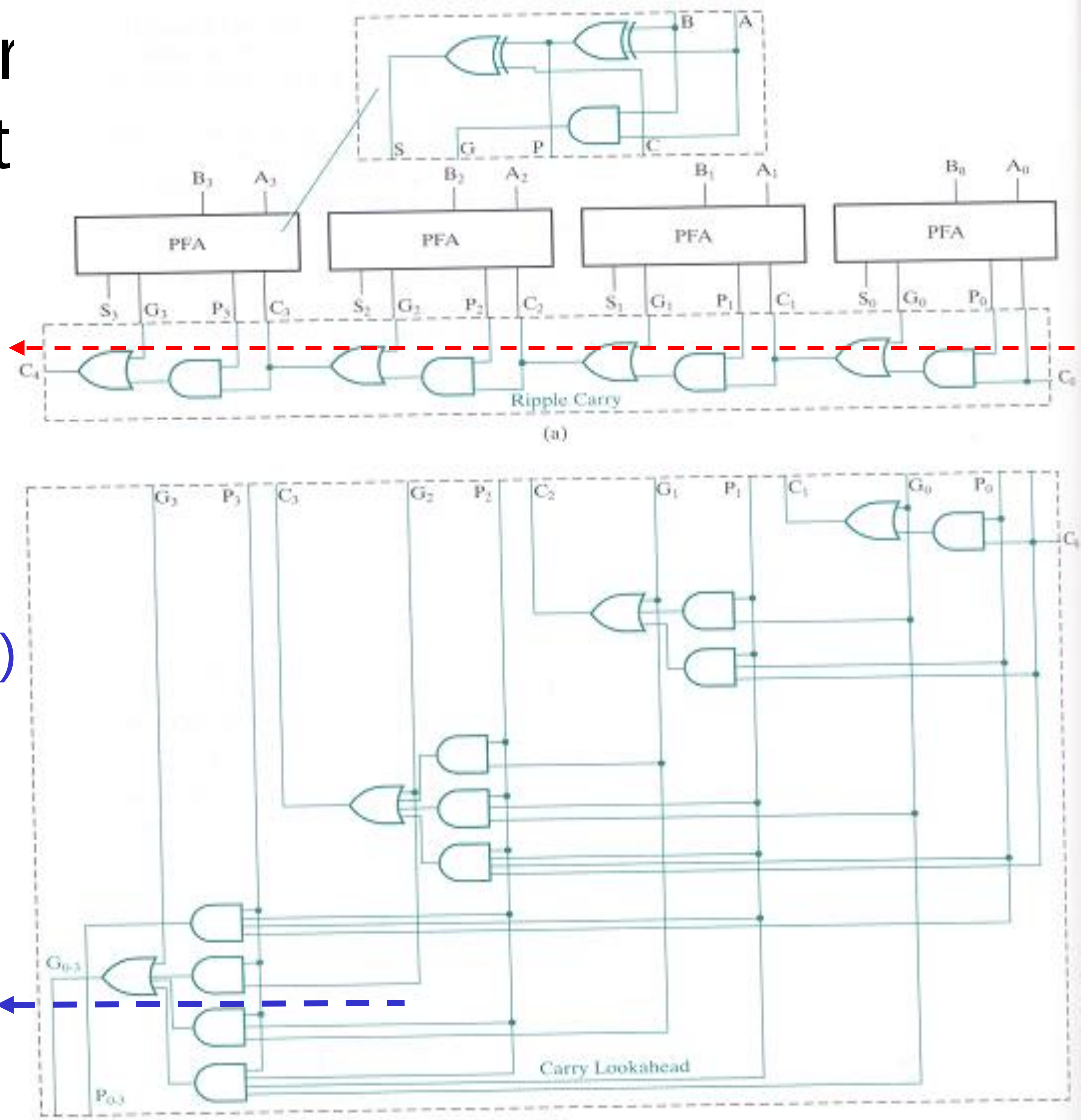
- Alternative design for a combinational n-bit adder.
- Reduced delay at the expense of more complex hardware.

← - Ripple Carry Delay (RD)

← - Carry Look-ahead Delay (LD)

LD < RD

- Study this circuit in detail using the textbook.



Binary Subtraction

- Unsigned numbers: minus sign is not explicitly represented.
- Given 2 binary numbers M and N, find M-N:
  - Case I: M $\geq$ N, thus, MSB of Borrow is 0

B	0 0 0 1 1 0		
M	− 1 1 1 1 0	− 30	
N	1 0 0 1 1	19	Result is Correct
Dif	0 1 0 1 1	11	

  - Case II: N > M, thus MSB of Borrow is 1

B	1 1 1 0 0 0		
M	− 1 0 0 1 1	− 19	
N	1 1 1 1 0	30	Result requires correction!
Dif	1 0 1 0 1	21	

Binary Subtraction (cont.)

- In Case II of the previous example, Dif= 19-30 = 21 = 19-30+2^5 (not correct).
- In general, if N > M, Dif = M-N+2^n, where n = # bits.
- To correct the magnitude of Dif, which should be N-M, calculate **2^n-(M-N+2^n) = N-M (correct)**.
- This is known as the 2's complement of Dif.
- To subtract two n-bit numbers, M-N, in base 2:
  - Find M-N.
  - If MSB of Borrow is 0, then M ≥ N. Result is positive and correct.
  - If MSB of Borrow is 1, then N > M. Result is negative and its magnitude must be corrected by subtracting it from 2^n (find its 2's complement).

Another Subtraction Example

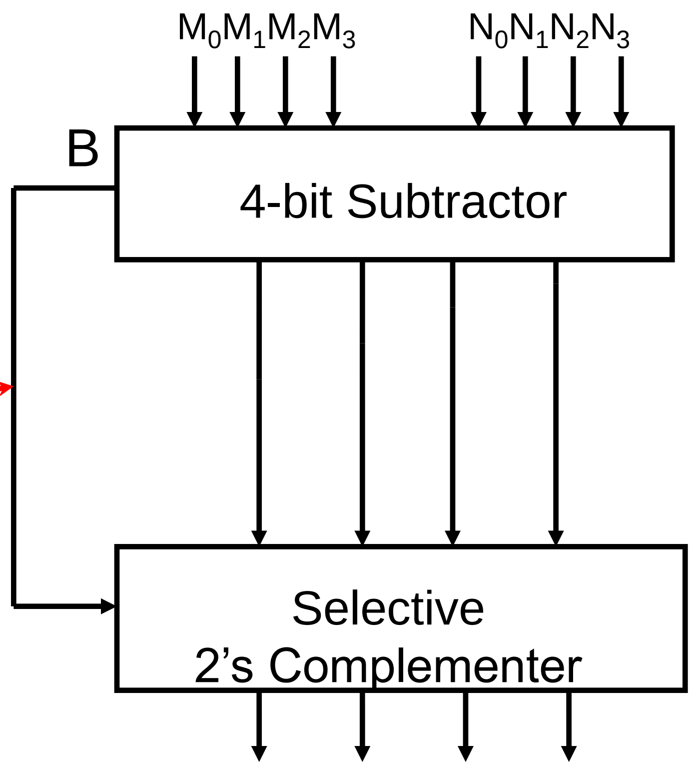
- Given M = 01100100 and N = 10010110, find M-N.

B	1 0 0 1 1 1 1 0 0		
M	0 1 1 0 0 1 0 0	100	
N	− 1 0 0 1 0 1 1 0	− 150	
Dif	1 1 0 0 1 1 1 0	206	(the result is negative)

2^n	1 0 0 0 0 0 0 0 0	256
Dif	− 1 1 0 0 1 1 1 0	− 206
	0 0 0 1 1 0 0 1 0	50

(corrected result, should be read as -50)

Block Diagram for Subtractor



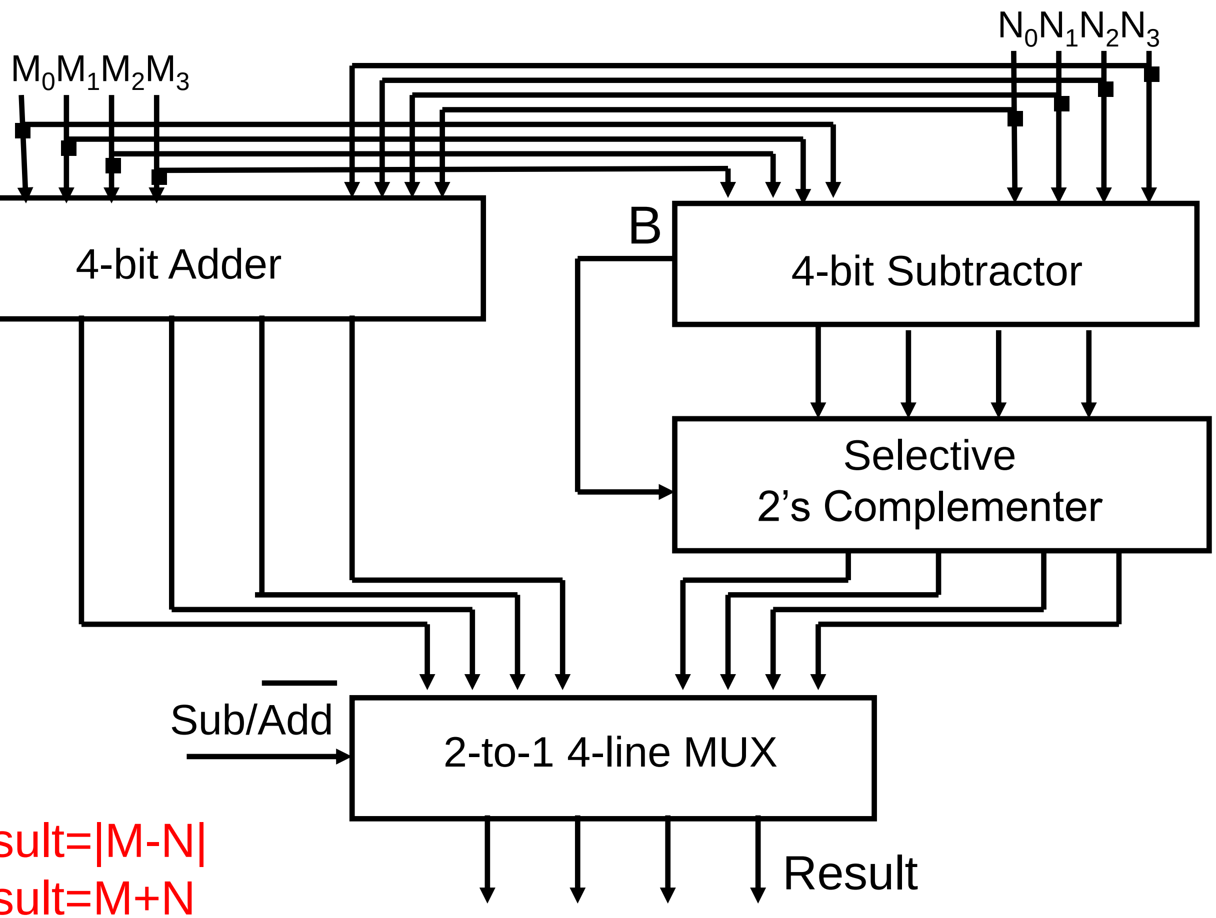
Subtract numbers M-N

If B = 1 then N>M

Correct the result if N>M
Enabled when B=1; otherwise, just pass the result from the subtractor

Not the best way to implement a subtractor circuit!

Block Diagram for Binary Adder-Subtractor



$Sub/\overline{Add}=1 \rightarrow$ Result=|M-N|

$Sub/\overline{Add}=0 \rightarrow$ Result=M+N

Again, not the <u>best</u> way to implement a Sub/Add circuit!

Complement Representations

- There are 2 types of complement representation of a number in base-***2 (binary)*** system:
  - *2's* complement
  - *1's* complement
- We have discussed this briefly at the beginning of the course (see Lecture 1).

2’s Complement

- For a positive *n*-bit number *N*, the *2*’s complement, *2*C(*N*), is given by:
  - $2C(N) = 2^n - N$
- Example: *N* = 1010
  - $2C(N) = 2^4 - N = 10000 - 1010_2 =$ **0110**
- Example: *N* = 11111
  - $2C(N) = 2^5 - N = 100000 - 11111 = 00001$
- Here’s an easier way to compute the *2’s* complement:
  1. Leave all least significant 0’s and first 1 unchanged
  2. Replace 0 with 1 and 1 with 0 in all remaining higher significant bits.
- Examples:

complement | unchanged

- N = 1010

0110

2’s complement on N

complement | unchanged

N = 01011000

10101000

2’s complement of N

1's Complement

- For a positive *n*-bit number *N*, the *1*'s complement, *1*C(N_2), is given by:
  - ***1C(N) = $(2^n-1) - N$***
- Example: *N* = 011
  - 1C(*N*) = $(2^3-1) - N$ = 111 – 011 = 100
- Example: *N* = 1010
  - 1C(*N*) = $(2^4-1) - N$ = 1111 – 1010 = 0101

- Observation1: 1's complement can be derived by just inverting all the bits in the number.
- Observation2: Compare 1's complement with 2's complement
  - $2^n-N = [(2^n-1) - N] + 1$
- Thus, the 2's complement can be obtained by deriving the 1's complement and adding 1 to it.
  - Example:
  - N = 1001
  - 2C(N) = 1C(N) + 1 = 0110 + 0001 = 0111

Subtraction with Complements

- To perform the subtraction M - N do:
  - Take the complement of N, i.e., **C(N)**
  - Perform addition M + C(N)
  - May need to correct the result
- We have discussed this briefly at the beginning of the course (see Lecture 1).

Subtraction with 2's complement

- If we use *2'*s complements to represent negative numbers:
  1. Form $R_I = M + 2C(N) = M + (2^n - N) = M - N + 2^n$.
  2. If there is a nonzero carry out of the addition, M ≥ N, so discard that carry and the remaining digits are the result R = *M-N*.
  3. Otherwise, M < N, so take the *2'*s complement of R_I $(=2^n - R_I = 2^n - (M - N + 2^n) = N - M)$, and attach a minus sign in front, *i.e.*, the result R is $-2C([R_I]_2) = -(N-M)$.
- A = 1010100 (84_{10}), B = 1000011 (67_{10})
- Find R = A-B:
  - 2C(B) = 0111101 (61_{10})
  - A+2C(B) = 1010100 + 0111101 = 10010001
  - Discard carry, R = 0010001 (17_{10}) ✓
- Find R = B-A:
  - 2C(A) = 0101100 (44_{10})
  - B+2C(A) = 1000011 + 0101100 = 1101111 (no carry, correction req.)
  - R = -2C(B+2C(A)) = -0010001 (-17_{10}) ✓

Subtraction with *1's* complement

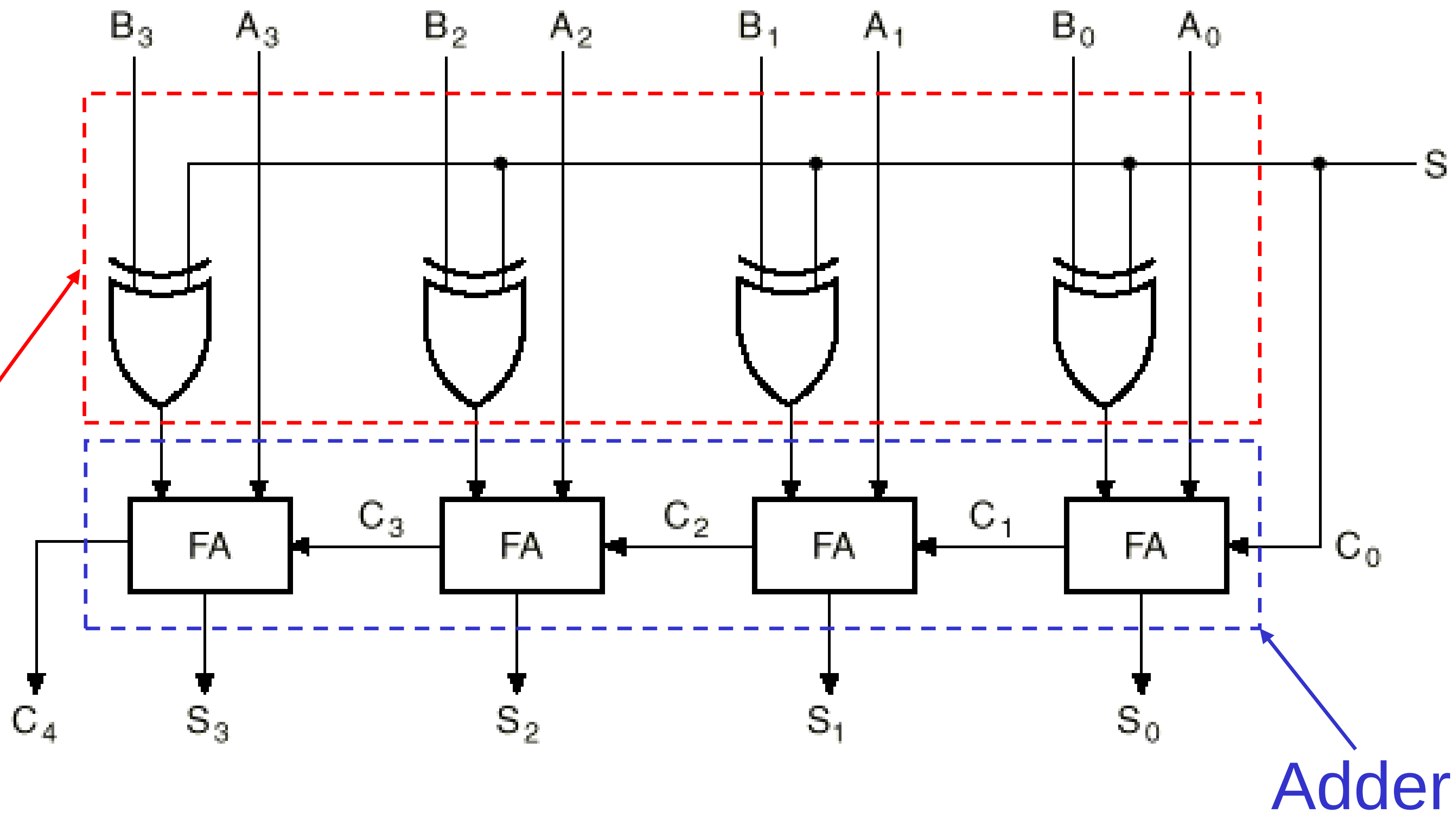
- If we use 1's complements to represent negative numbers:
  1. Form $R_I = M + 1C(N) = M + (2^n\text{-}1\text{-}N) = M - N + 2^n\text{-}1$.
  2. If there is a nonzero carry out of the addition, M ≥ N, so discard that carry and add 1 to the remaining digits. The result R = *M-N*.
  3. Otherwise, M < N, so take the *1's* complement of R_I $(=2^n\text{- }1\text{ - }R_I = 2^n\text{- }1\text{ - }(M - N + 2^n\text{-}1) = N - M$), and attach a minus sign in front, *i.e.*, the result R is $-1C([R_I]_2) = -(N\text{-}M)$.
- A = 1010100 (84_{10}), B = 1000011 (67_{10})
- Find R = A-B:
  - 1C(B) = 0111100 (60_{10})
  - A+1C(B) = 1010100 + 0111100 = 10010000
  - Discard carry and add 1,
R = 0010000 + 1 = 0010001 (17_{10}) ✓
- Find R = B-A:
  - 1C(A) = 0101011
  - B+1C(A) = 1000011 + 0101011 = 1101110 (no carry, correction needed)
  - R = -1C(B+1C(A)) = -0010001 (-17) ✓

Binary Adder/Subtractors

- If we perform subtraction using complements
  - we do addition instead of subtraction operation
  - we can use an adder with appropriate complementer for subtraction
- Actually, we can use an adder for both addition and subtraction:
  - Complement subtrahend for subtraction
  - Do not complement subtrahend for addition
- Thus, to form an adder-subtractor circuit, we only need a selective complementer and an adder.
- The subtraction *A*-*B* can be performed as follows:

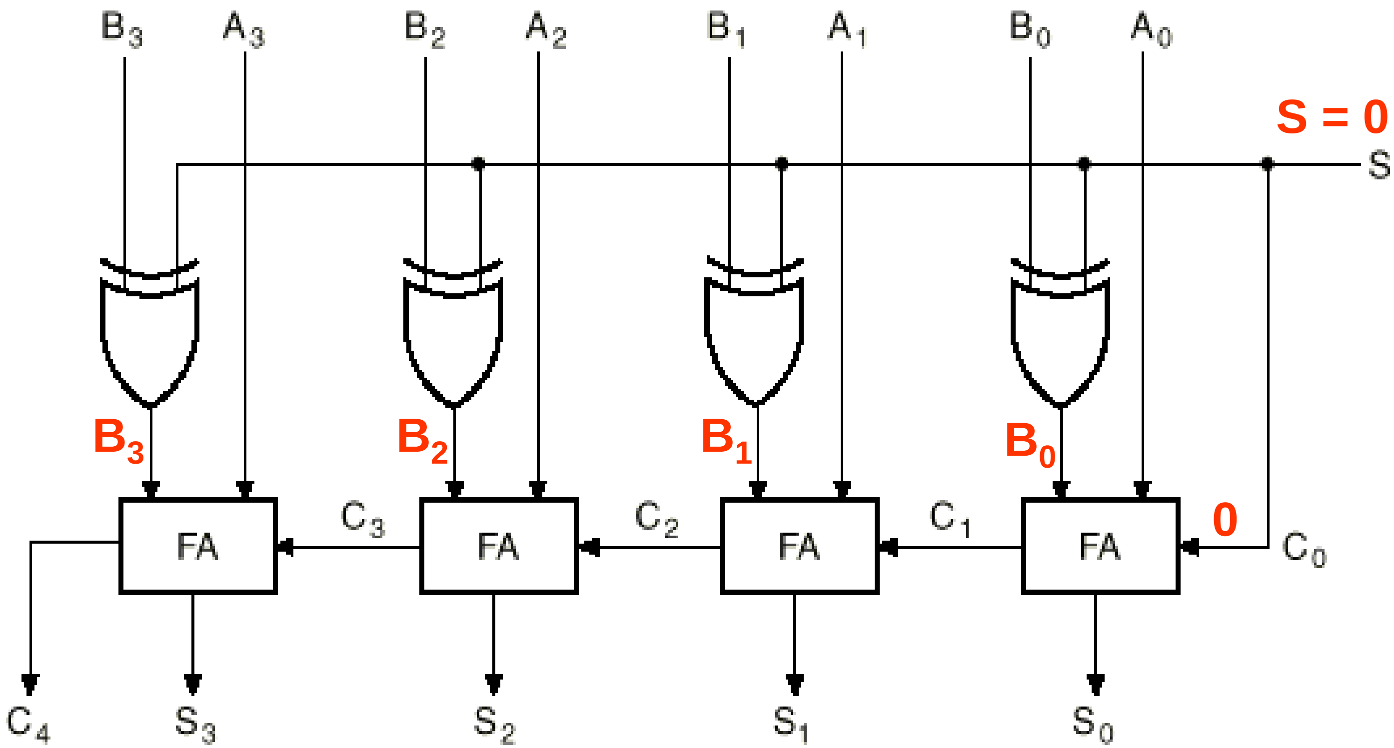
$$\begin{aligned} A-B &= A + 2C(B) \\ &= A + 1C(B) + 1 \\ &= A + B' + 1 \end{aligned}$$

4-bit Binary Adder-Subtractor using 2's Complement



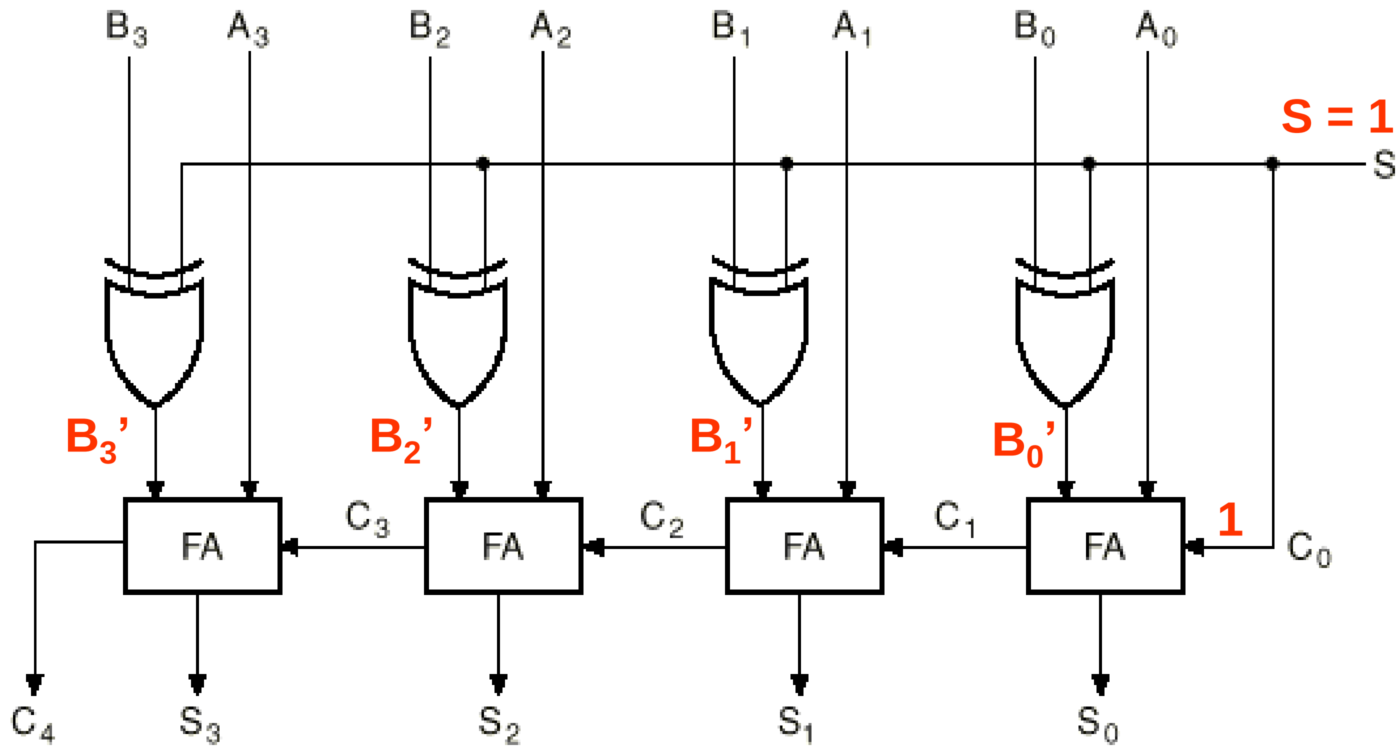
Selective complementer:
XOR gates act as programmable inverters

4-bit Binary Adder-Subtractor (cont.)



- When $S = 0$, the circuit performs ***A* + *B***. The carry in is 0, and the XOR gates simply pass B untouched.

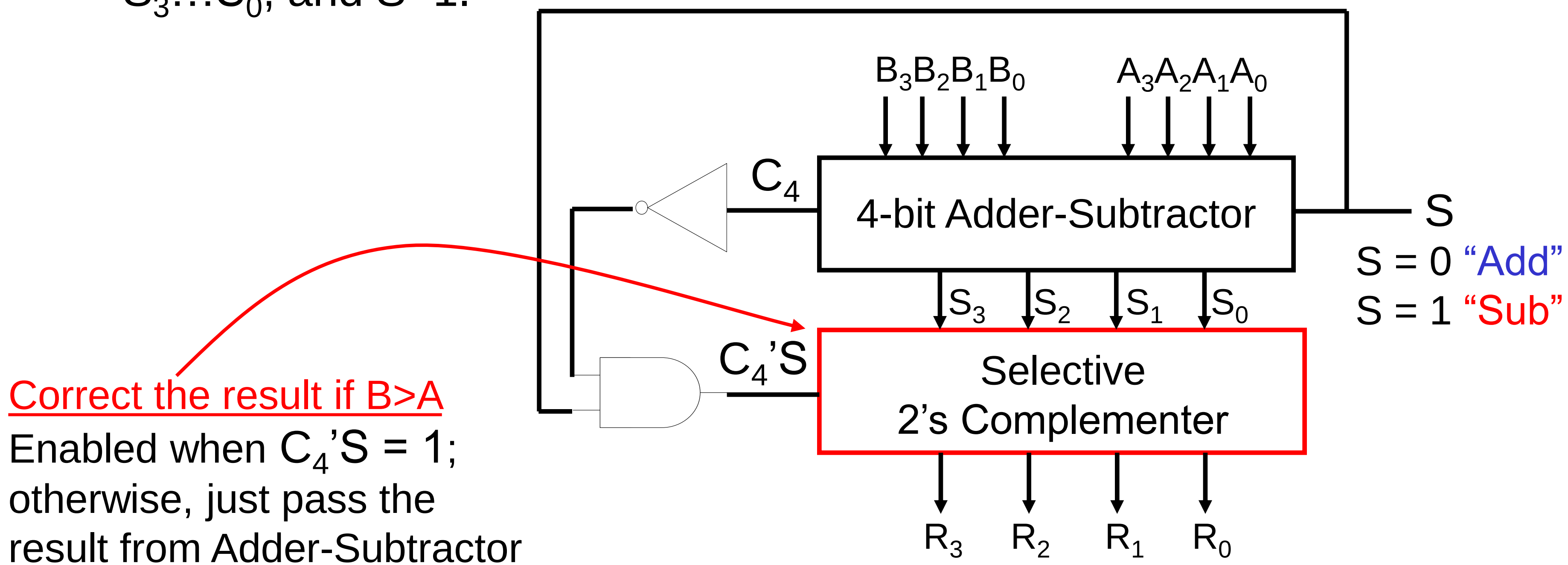
4-bit Binary Adder-Subtractor (cont.)



- When $S = 1$, the circuit performs ***A* - *B*** , i.e.,
A - B = A + *2C*(B) = A + *1C*(B) + 1 = **A + B' + 1**

4-bit Binary Adder-Subtractor (cont.)

- When we do subtraction, result may need to be corrected
  - If $C_4 = 0$ and $S = 1$, we must correct the result $S_3 \dots S_0$.
- Thus, we must compute 2's complement of $S_3 \dots S_0$:
  - Use a specialized 2's complement circuit or
  - Use the 4-bit Adder-Subtractor again, with $A_3 \dots A_0 = 0000$, $B_3 \dots B_0 = S_3 \dots S_0$, and S=1.



Signed Binary Numbers

- **Signed-magnitude representation:** Singed numbers are represented using the MSB of the binary number to indicate the number's sign:
  - If MSB is 0 → number is positive
  - If MSB is 1 → number is negative
- Do not confuse with unsigned numbers!
- For example:
  - -10_{10} is
    - 11010_2 in singed ("-" sing is indicated in MSB = 1)
- Another example:
  - 1011_2 is
    - 11_{10} in unsigned
    - -3_{10} in signed

Signed-Magnitude Addition-Subtraction

- To implement signed-magnitude addition and subtraction
  - separate the ***sign bit*** from the ***magnitude bits***
  - treat the magnitude bits as an unsigned number
  - do ordinary arithmetic
  - do correction if needed
- Example: M:00011001, N:10100101; find M+N
  - N is negative
  - so do M-N = 0011001-0100101 =1110100, with end borrow 1. This implies that M-N is a negative number,
  - so to correct find its 2's complement 0001100. Result is 10001100.

Signed-Complement System

- To avoid correction of the result, use the singed-complement representation of numbers
  - Signed-1's complement
  - Signed-2's complement

- Ex.: Use 8-bits to represent -9_{10} and 9_{10}
  - 9_{10} is 00001001_2 in any of the above representations
  - -9_{10} is:
    - 10001001_2 in singed-magnitude
    - 11110110_2 in singed-1's complement
    - 11110111_2 in singed-2's complement

Signed-Complement Addition

- Addition of two signed numbers in signed-2's complement form is obtained
  - by adding the two numbers **including the sign bits**.
  - carry out is discarded".

- Examples: (Assume 5-bit representations)

```
  0|1010 (+10)     0|1010 (+10)     1|0110 (-10)     1|0110 (-10)
+ 0|0101 (+5)    + 1|1011  (-5)   + 0|0101 (+5)    + 1|1011  (-5)
  0|1111 (+15)    10|0101  (+5)     1|1011  (-5)    11|0001 (-15)
```

Signed-Complement Subtraction

- Subtraction of two signed numbers in signed-2's complement form is obtained by
  - taking **the 2's complement** of the subtrahend **including sign bit**
  - add it to the minuend
  - Discard carry out

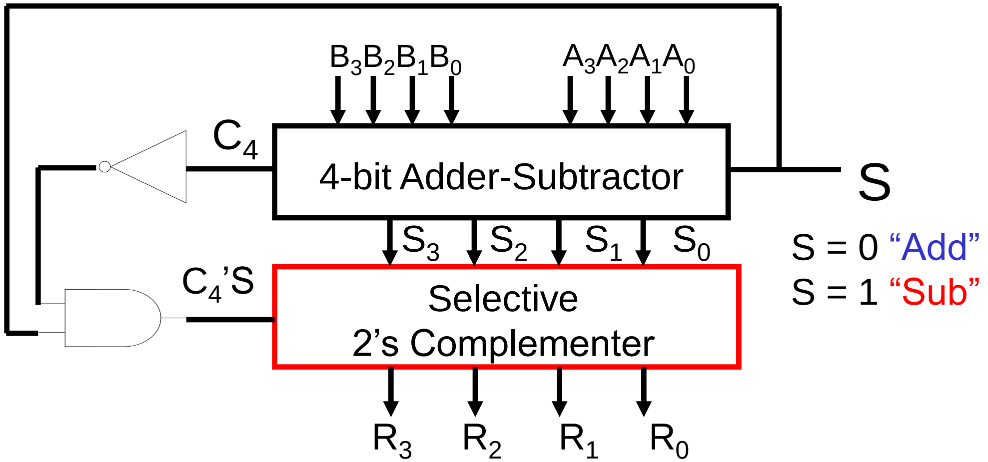
- Examples: (Assume 5-bit representations)

0|1010 (+10)   0|1010 (+10)   1|0110 (-10)   1|0110 (-10)
-0|0101 -(+5)   - 1|1011 -(-5)   - 0|0101 -(+5)   -1|1011 -(-5)

0|1010 (+10)   0|1010 (+10)   1|0110 (-10)   1|0110 (-10)
+1|1011 +(-5)   +0|0101 +(+5)   +1|1011 +(-5)   +0|0101 +(+5)
10|0101 (+5)   0|1111 (+15)   11|0001 (-15)   1|1011 (-5)

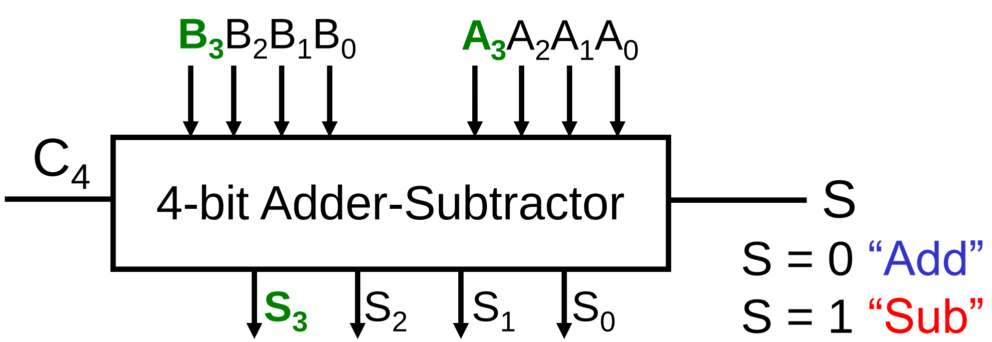
Binary Adder-Subtractor using 2's Complement Signed Numbers

- The circuit is simpler (correction is not needed):



Adder-Subtractor of
4-bit **unsigned** numbers.

Remove the correction circuit



Adder-Subtractor of
4-bit signed 2's complement numbers

The **4-th bit** in the numbers is interpreted as the sign bit.

The Overflow problem

- If the sum of two n-bit numbers results in an n+1 bit number, then an ***overflow*** conditions is said to occur.
- Detection of overflow can be implemented using either hardware or software.
- Detection depends on number system used: signed or unsigned.

The Overflow problem in Unsigned System

- Addition:
  - When Carry out is 1 we have overflow.
- Subtraction:
  - Can never occur. Magnitude of the result is always equal or smaller than the larger of the two numbers.
- → Not REALLY a problem!

- V = 1 indicates overflow condition when adding unsigned numbers.

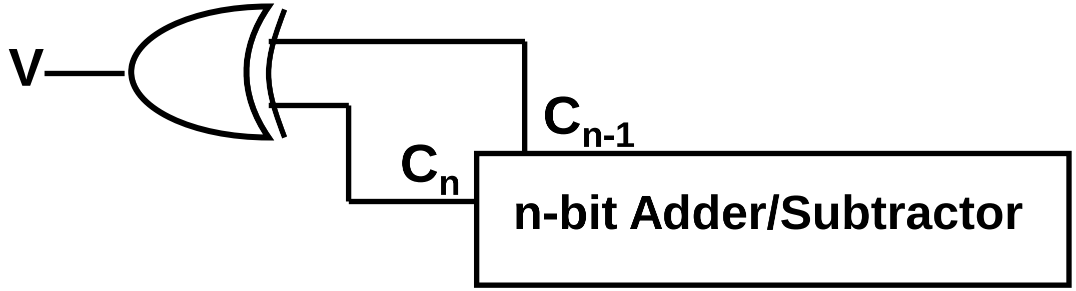
The Overflow problem in Signed-2's Complement

- Remember that the MSB is the sign. But, the sign is also added! Thus, a carry out equal to 1 does not necessarily indicate 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}).
- Example 1: Let $M=65_{10}$ and $N=65_{10}$ in an 8-bit signed-2's complement system.
  - $M = N = 01000001_2$
  - M+N = 10000010 with C_n=0. This is 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: Let $M=-65_{10}$ and $N=-65_{10}$ in an 8-bit signed-2's complement system.
  - $M = N = 10111111_2$
  - M+N = 01111110 with C_n=1. This is 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

- Overflow condition is detected by comparing the carry values into and out of the sign bit (C_n and C_{n-1}).

n-bit Adder/Subtractor with Overflow Detection Logic



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

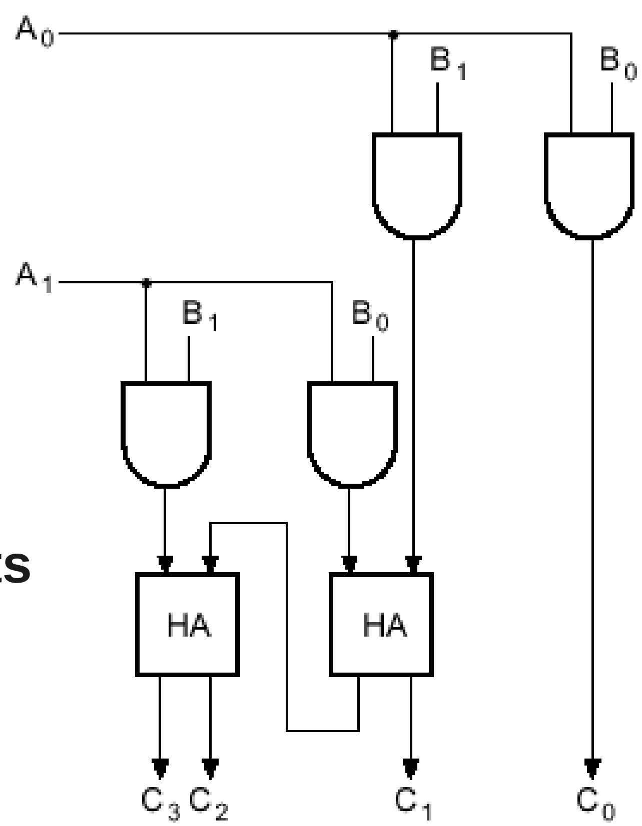
Binary Multiplier

- Binary multiplication resembles decimal multiplication:
  - *n*-bit multiplicand is multiplied by each bit of the *m*-bit multiplier, starting from LSB, to form *m* partial products.
  - Each successive partial product is shifted 1 bit to the left.
  - Derive result by addition the *m* rows of partial products.
  - The resultant product is a binary number that consists of *n* + *m* bits.
- Example:
  - Multiplicand **B = $(1011)_2$**
  - Multiplier **A = $(101)_2$**
  - Find Product **C = B x A:**

```
Multiplicand:     1011
Multiplier:     x  101
               -------
                  1011
              +  0000
                1011
               -------
Product:        110111
```

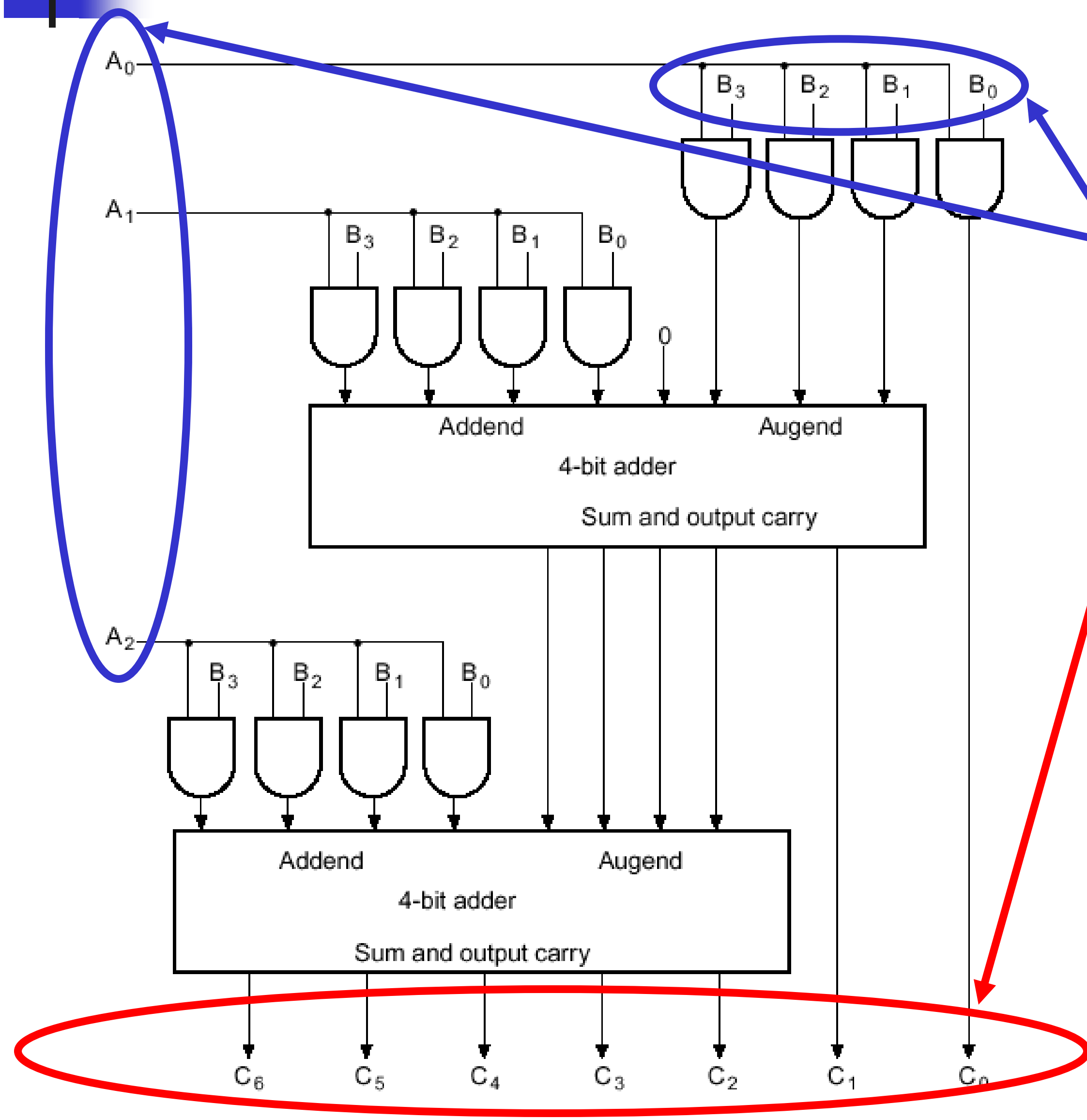
2-bit by 2-bit Binary Multiplier

		B_1	B_0
		A_1	A_0
		A_0B_1	A_0B_0
	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0



Half Adders are Sufficient since there is no Carry-in in addition to the two inputs to sum

4-bit by 3-bit Binary Multiplier

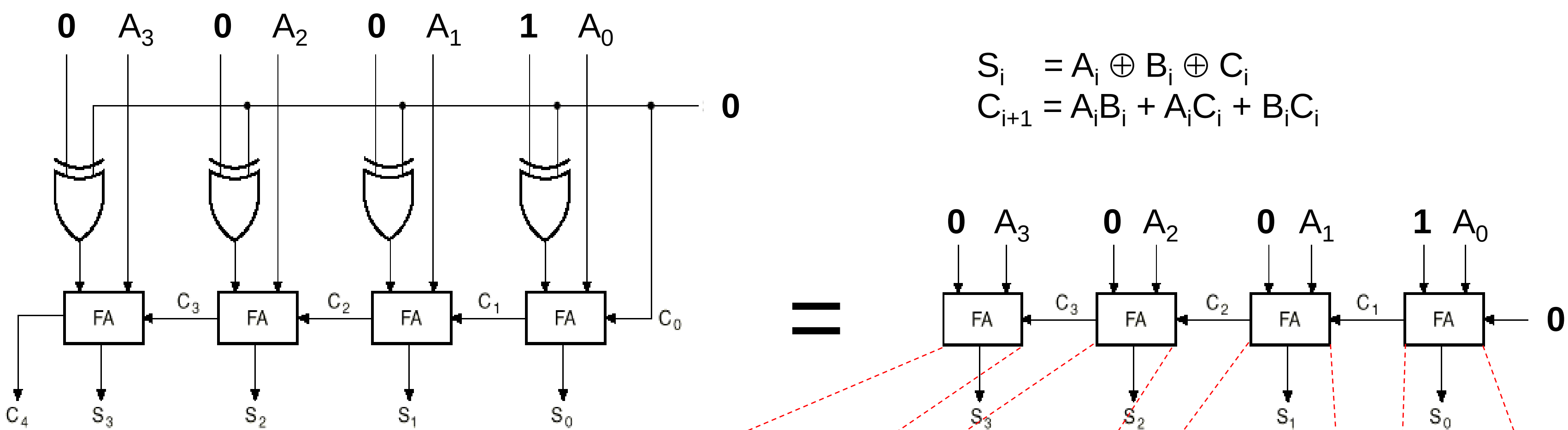


4 bit by 3 bit yields a 7 bit result

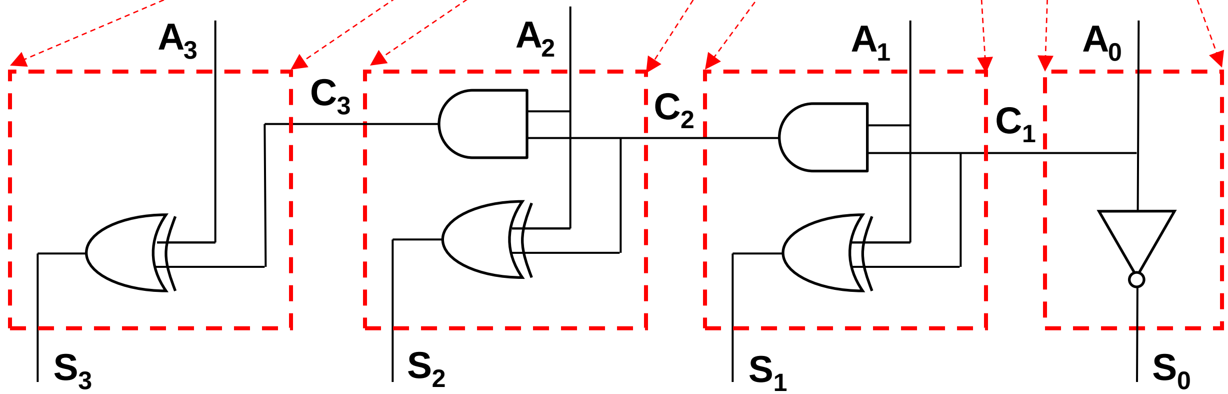
Other Arithmetic Functions

- Incrementing
- Decrementing
- Multiplication by Constant
- Division by Constant

Increment by 1



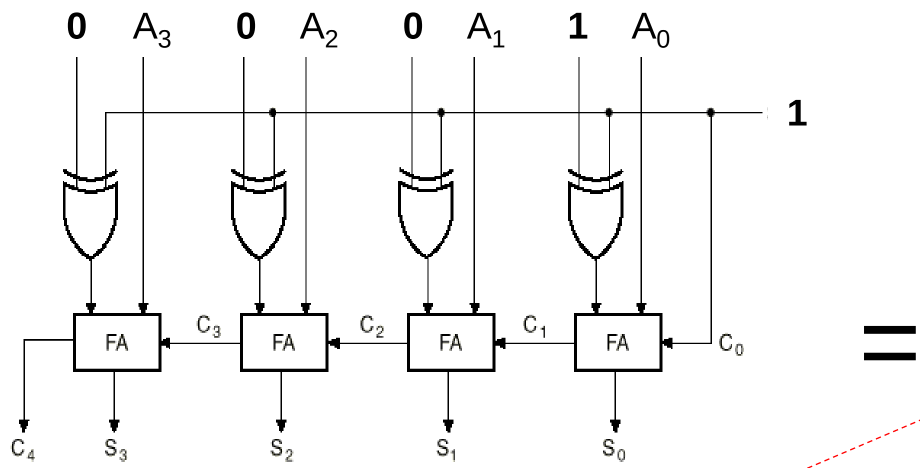
$S_i \;\; = A_i \oplus B_i \oplus C_i$
$C_{i+1} = A_iB_i + A_iC_i + B_iC_i$



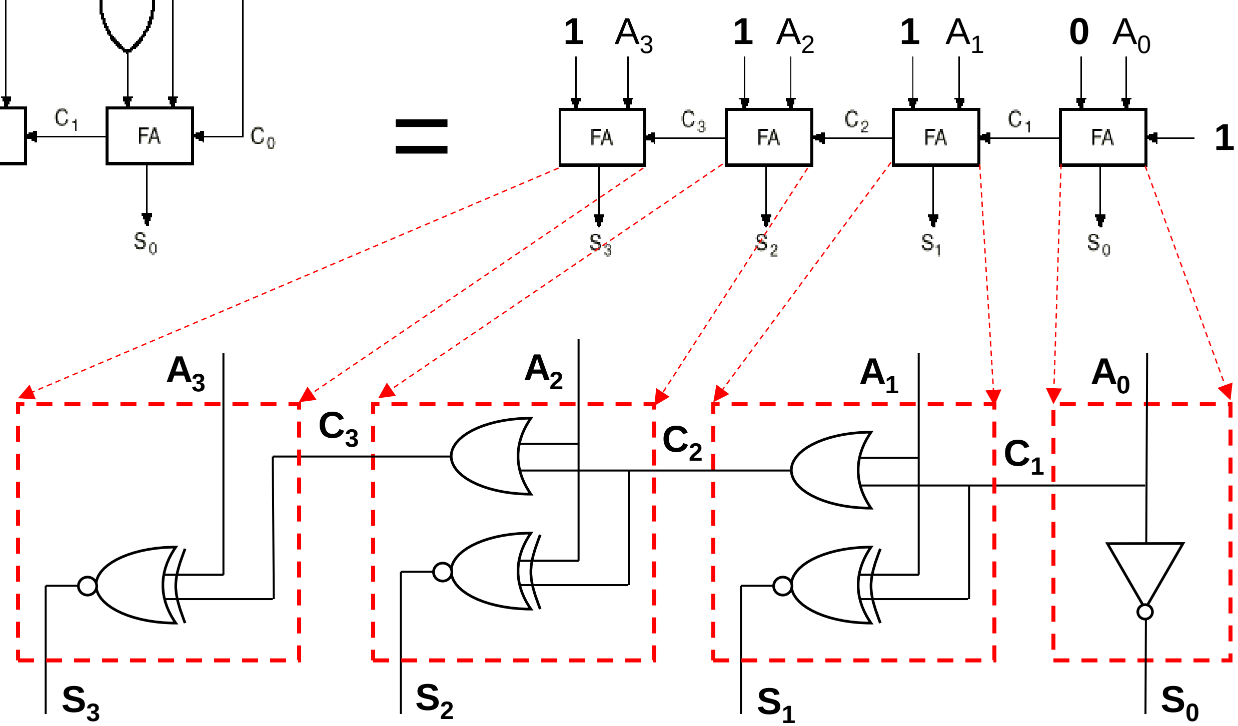
$S_{1,2,3} = A_i \oplus C_i$
$C_{2,3} = A_iC_i$

$S_0 = A_0'$
$C_1 = A_0$

Decrement by 1



$S_i \;\; = A_i \oplus B_i \oplus C_i$
$C_{i+1} = A_iB_i + A_iC_i + B_iC_i$

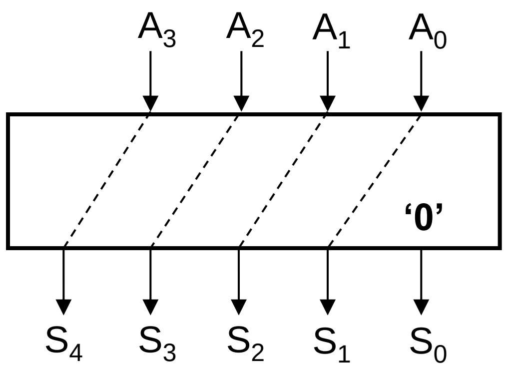


$S_{1,2,3} = (A_i \oplus C_i)'$
$C_{2,3} = A_i + C_i$

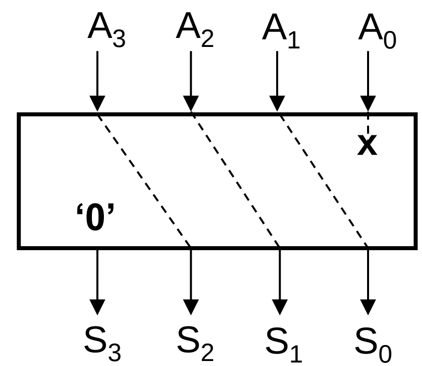
$S_0 \;\; = A_0'$
$C_1 \;\; = A_0$

Multiplication/Division by constant

Multiplication by 2 (shift left)



Division by 2 (shift right)



Multiplication by 2^n

Division by 2^n