CS2100
COMPUTER ORGANIZATION:
ALU DESIGN

Hamid Sarbazi-Azad

Visiting Professor

Arithmetic Circuits

1. Binary Arithmetic Circuits
   - Addition/Subtraction
   - Multiplication
   - Division

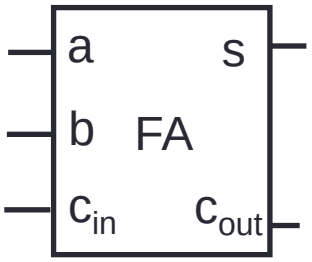
2. Decimal Arithmetic Circuits
   - Addition/Subtraction
   - Multiplication
   - Division

3. Floating-point Arithmetic Circuits
   - Addition/Subtraction
   - Multiplication
   - Division

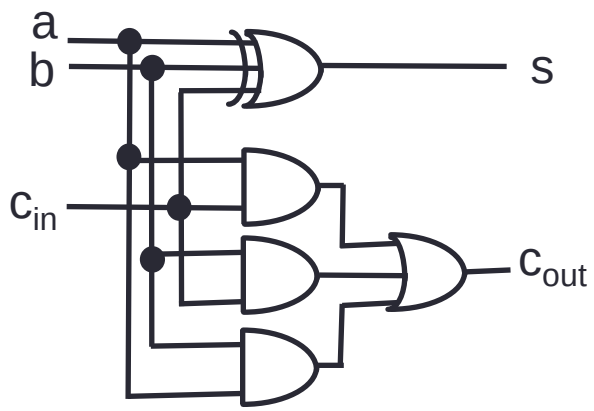
4. Special Purpose Arithmetic Circuits

Binary Addition/Subtraction

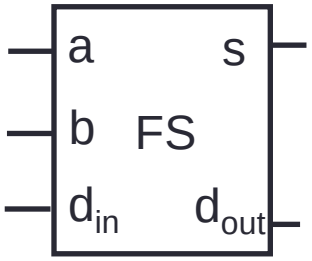
A Full-Adder (FA) adds three bits a, b and c (also named c_{in} to indicate input-carry for multibit addition) and generates 2 bits s (result bit) and c (also named c_{out}).



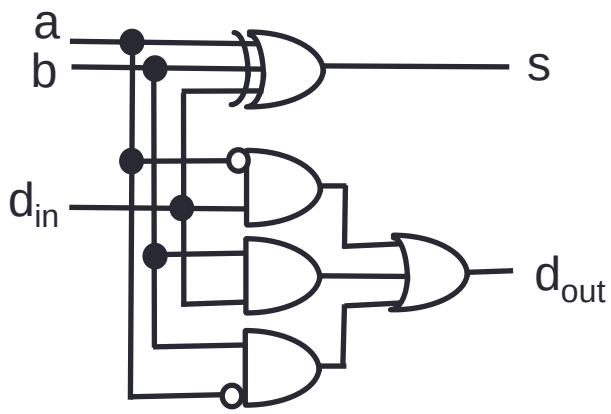
a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



A Full-Subtractor (FS) reduces b and d (also named d_{in} to indicate borrow in for multibit subtraction) from a and generates 2 bits s (result bit) and d_{out} (borrow out).



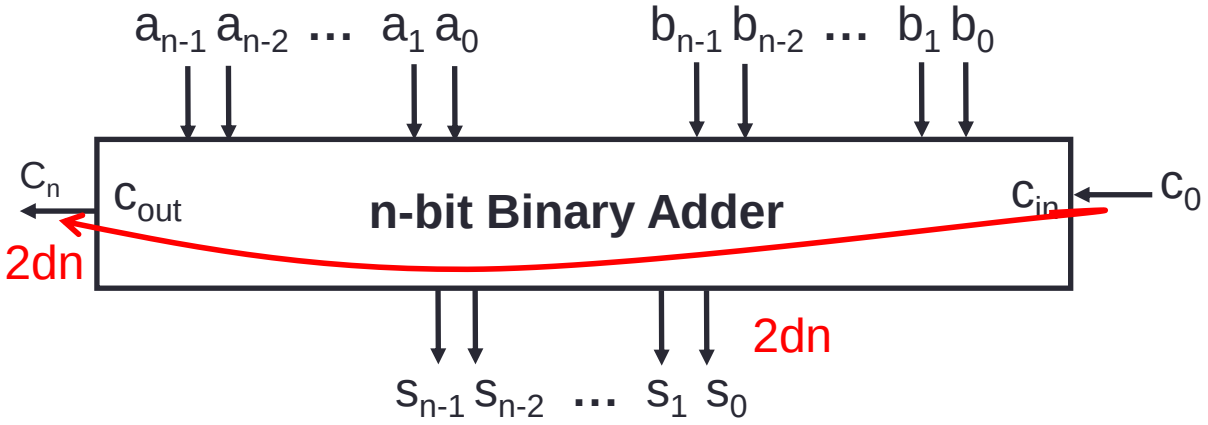
a	b	d_{in}	d_{out}	s
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

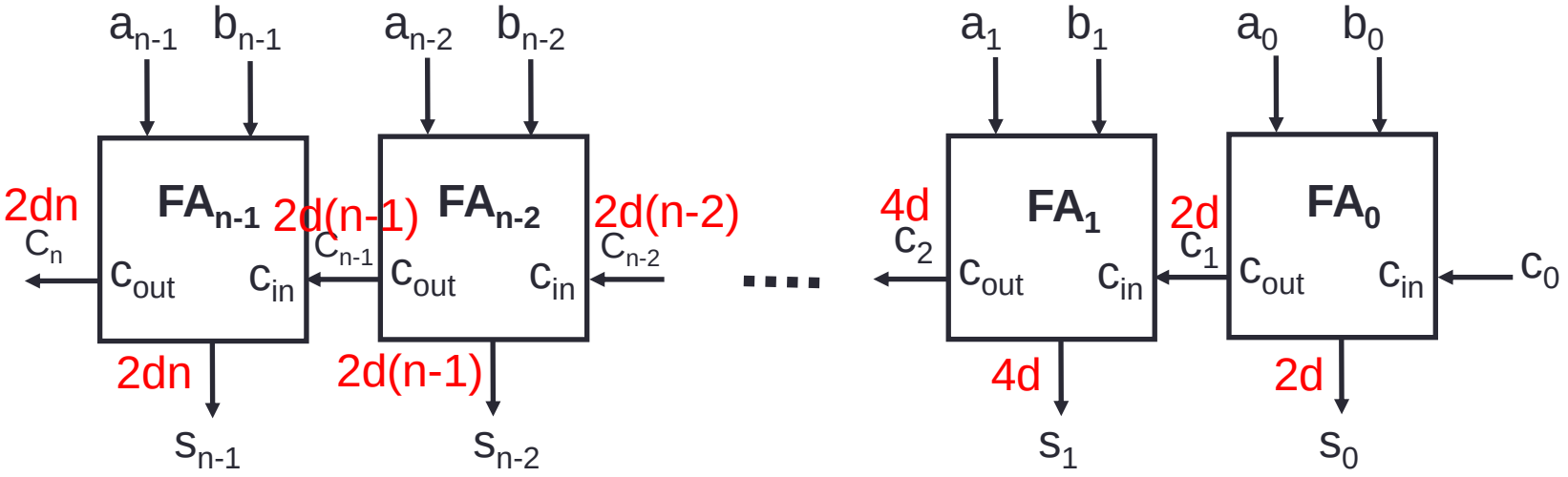


Carry Ripple Adder (Parallel Adder)

An n-bit carry-ripple adder (CRA) can be easily implemented with n cascaded FAs.
We add A = $a_{n-1}a_{n-2}\ldots a_1a_0$ and B = $b_{n-1}b_{n-2}\ldots b_1b_0$ and generate S = $s_{n-1}s_{n-2}\ldots s_1s_0$.

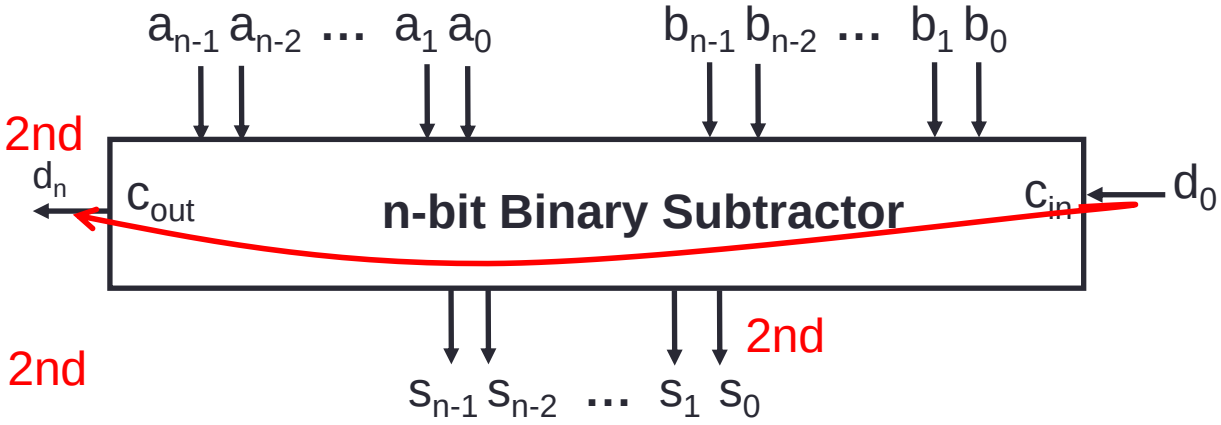
$t_{\text{n-bit Carry-Ripple Adder}} = 2nd$



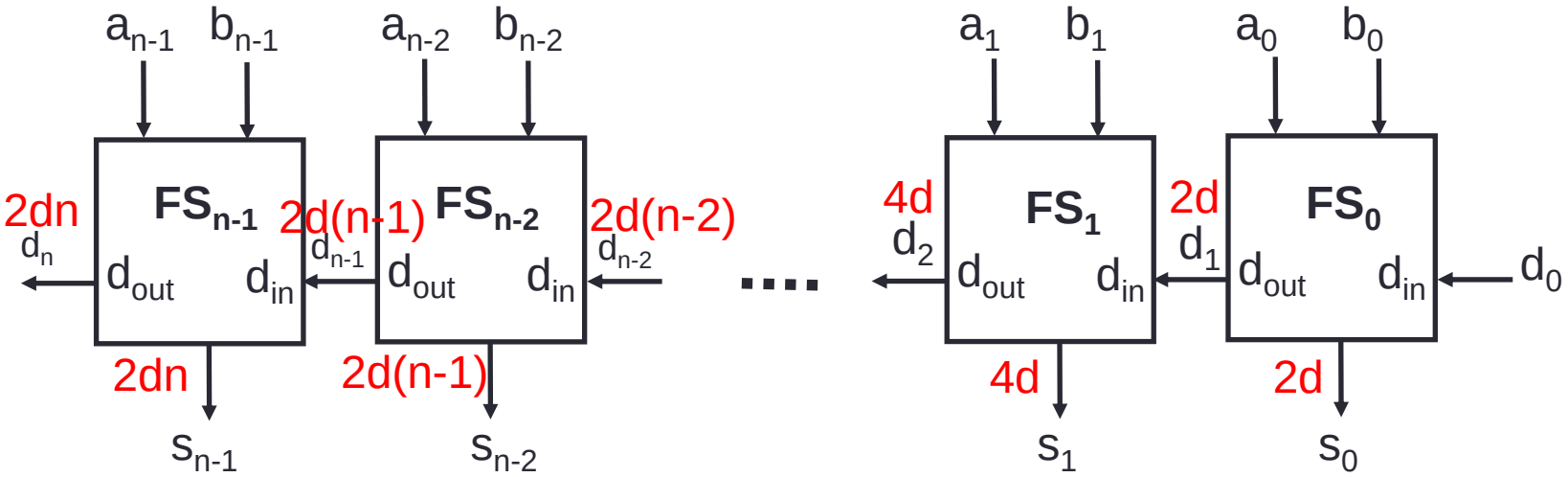


Borrow Ripple Subtractor (Parallel Subtractor)

An n-bit borrow-ripple subtractor (BRS) can be implemented with n cascaded FSs.
We subtract $B = b_{n-1}b_{n-2}\ldots b_1b_0$ from $A = a_{n-1}a_{n-2}\ldots a_1a_0$ to generate $S = s_{n-1}s_{n-2}\ldots s_1s_0$.



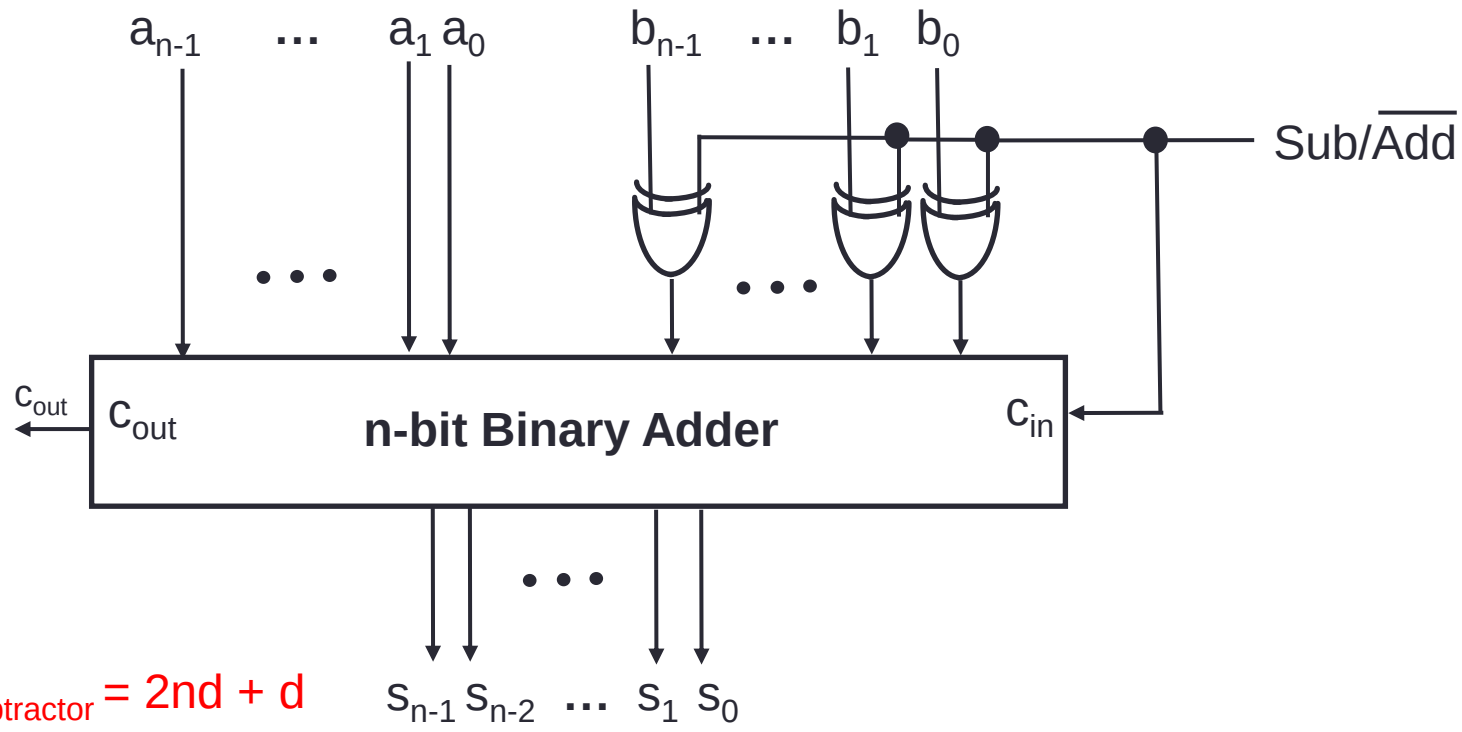
$t_{\text{n-bit Borrow-Ripple Subtractor}} = 2nd$



Parallel Adder/Subtractor

An n-bit parallel adder/subtractor can be implemented with n cascaded FAs and n XOR gates.

We have $B = b_{n-1}b_{n-2}\ldots b_1b_0$ and $A = a_{n-1}a_{n-2}\ldots a_1a_0$ and generate $S = s_{n-1}s_{n-2}\ldots s_1s_0 = A+B$, if Sub/$\overline{\text{Add}}$ = 0, and $S = A - B$ if Sub/$\overline{\text{Add}}$ = 1, in 2's complement number system.



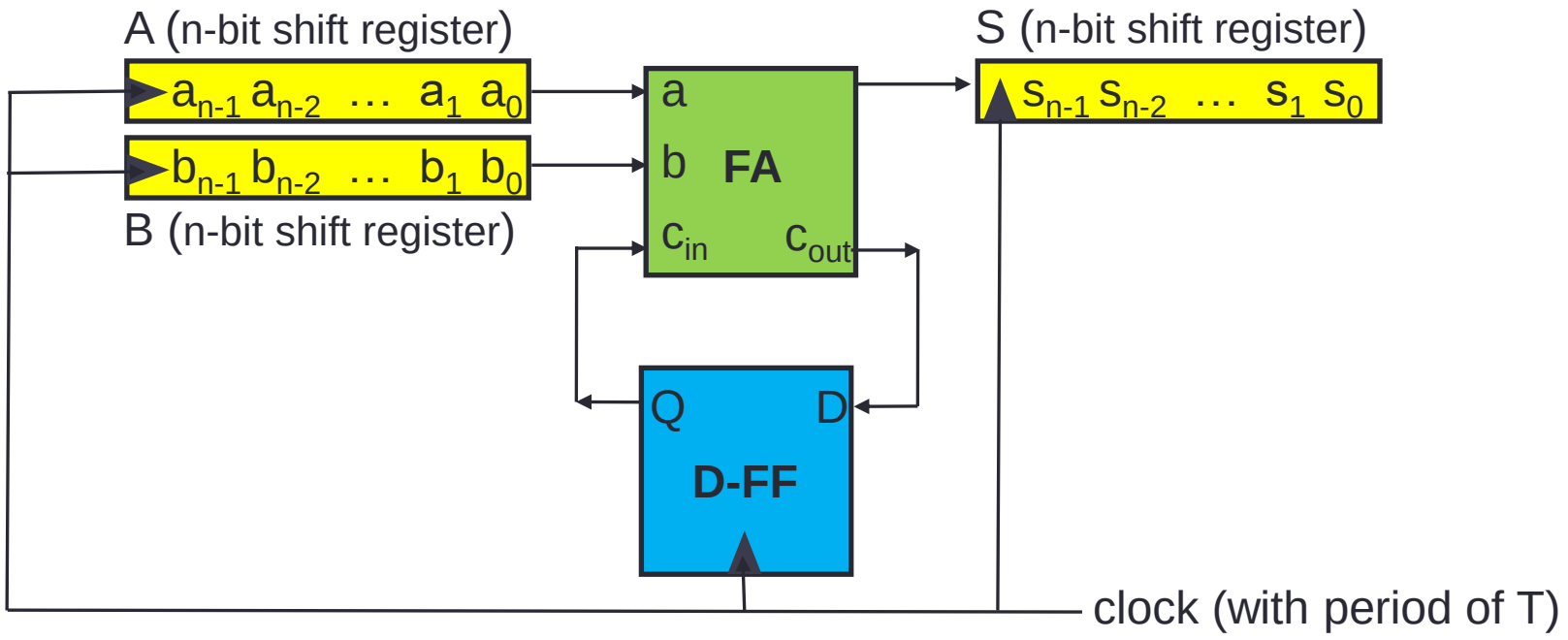
$t_{\text{n-bit parallel adder/subtractor}} = 2nd + d$

Serial Binary Adder

Adding two binary numbers can be done in a serial manner using the below circuit.

$A\ (a_{n-1}a_{n-2}\ldots a_1a_0) + B\ (b_{n-1}b_{n-2}\ldots b_1b_0) = S\ (s_{n-1}s_{n-2}\ldots s_1s_0)$

$t_{\text{n-bit Serial Adder}} = n\,T > n\,(2d + t_{FF})$

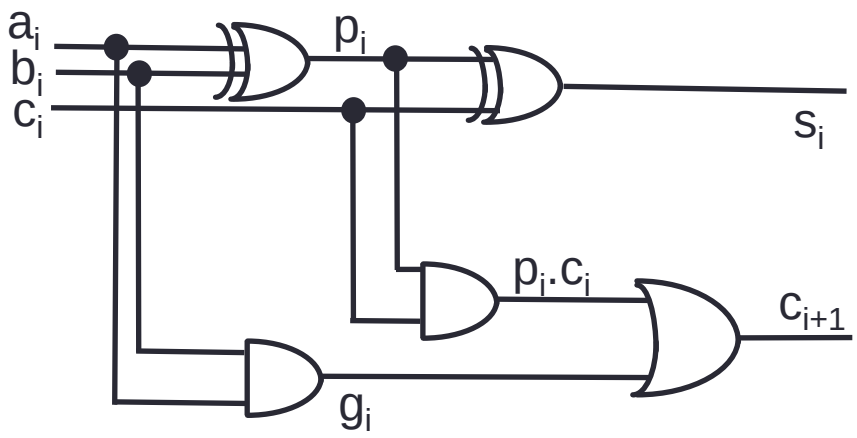


Faster Adders: Carry Look Ahead Adder

The main obstacle with speeding up addition is the carry bit rippled at different stages.

→ If carry is generated faster, the addition can be realized faster!

Lets look inside the logic of a full-adder and follow the carry generation logic in an n-bit adder.



Let:

$p_i = a_i \oplus b_i \quad g_i = a_i b_i$

→ $s_i = p_i \oplus c_i \quad c_{i+1} = g_i + p_i c_i$

We can now write:

$c_1 = g_0 + p_0 c_0$

$c_2 = g_1 + p_1 c_1 = g_1 + p_1(g_0 + p_0 c_0) = g_1 + p_1 g_0 + p_1 p_0 c_0$

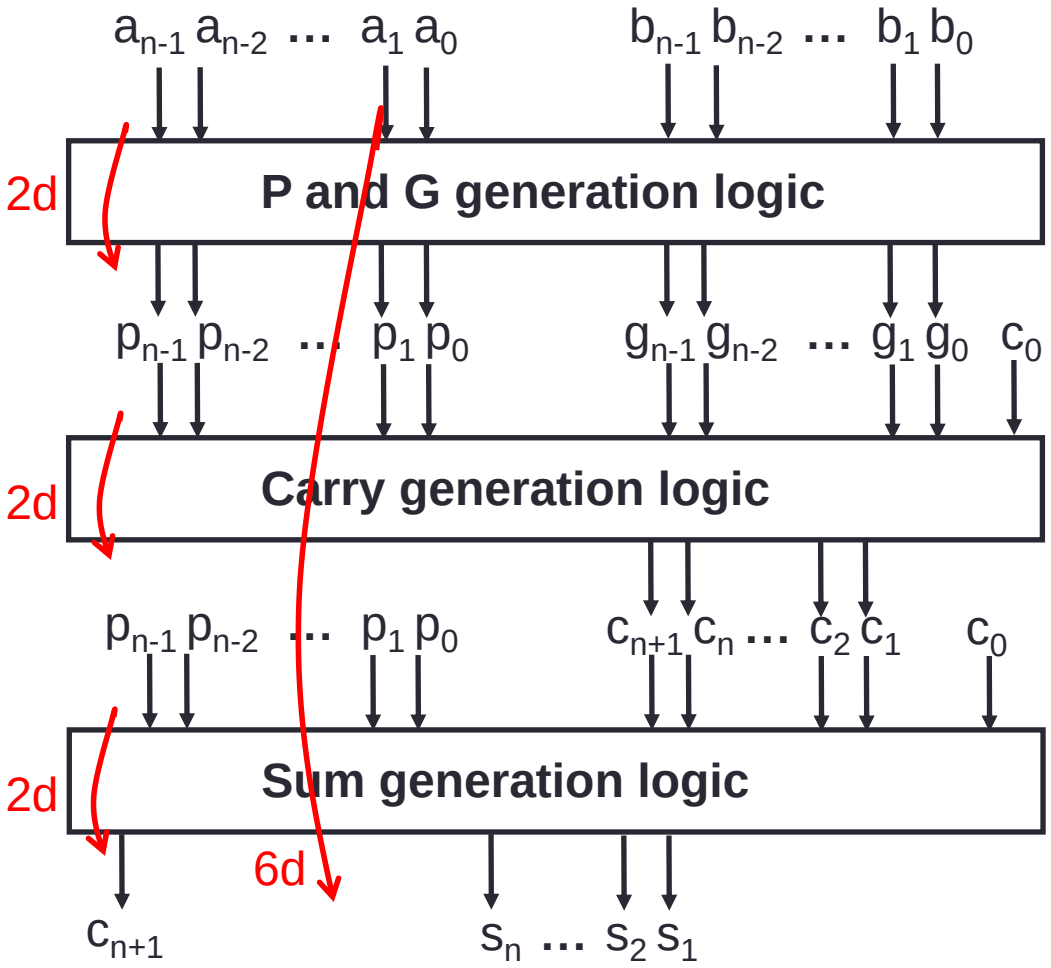
$c_3 = g_2 + p_2 c_2 = g_2 + p_2(g_1 + p_1 g_0 + p_1 p_0 c_0) = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$

…

→ $c_i = \Sigma_{j=0,1,\ldots,i-1} (g_{i-1} \Pi_{k=j,j+1,\ldots,i-1} p_k)$ where $g_0 = c_0$.

Faster Adders: Carry Look Ahead Adder

Now, we can add two n-bit numbers using CLA idea as:

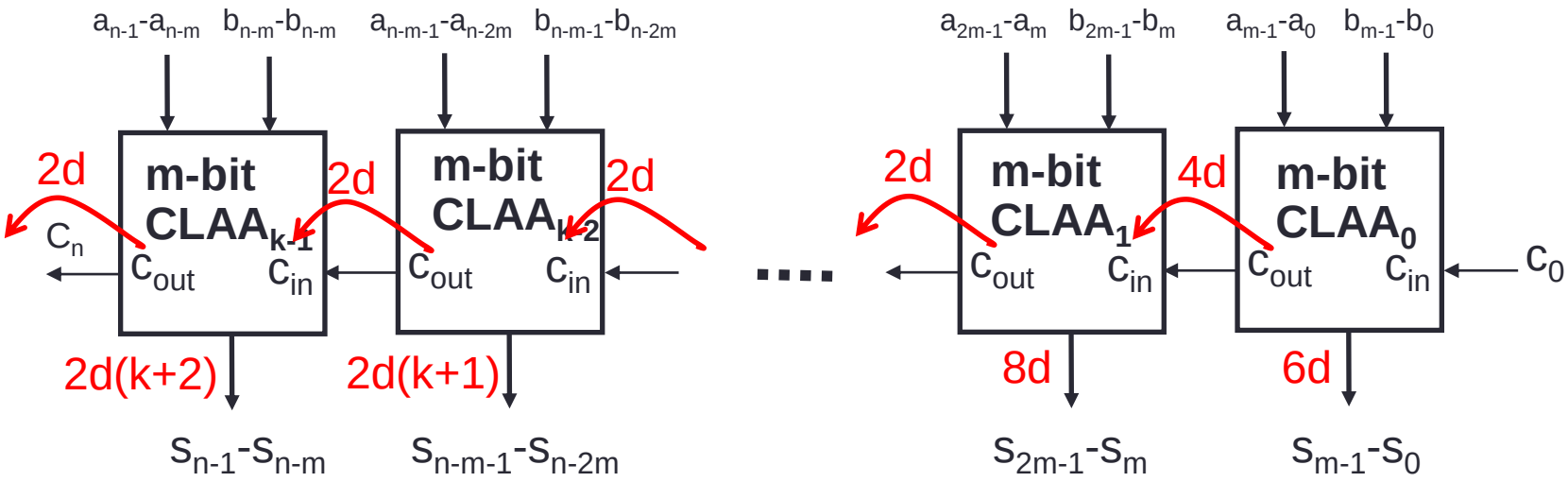


Limited fan-in and fan-out in any available integration technology force us to keep n small ☹

We can cascade m-bit CLAAs to build an n-bit adder ☺

Faster Adders: Carry Look Ahead Adder

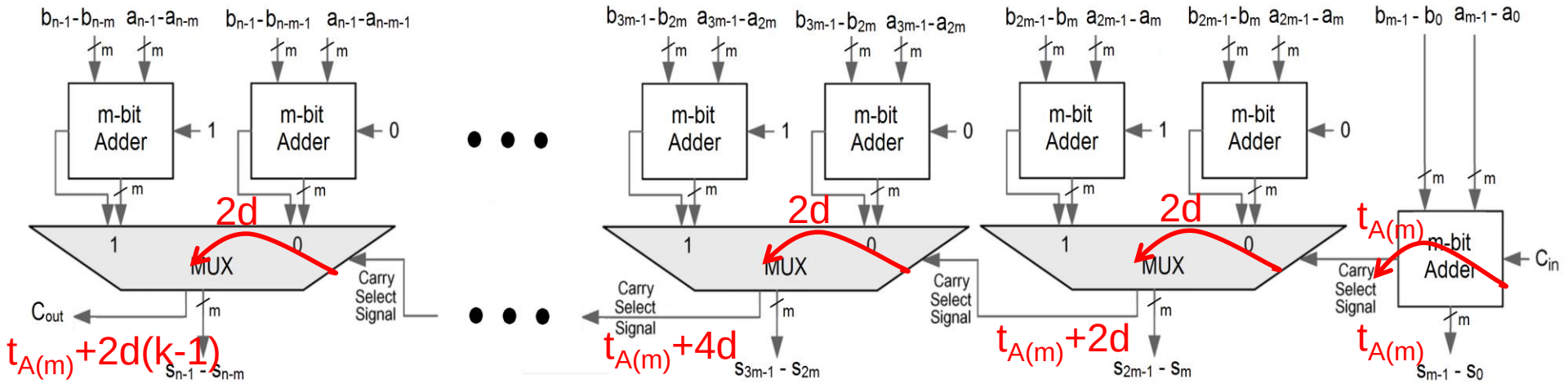
An n-bit binary adder using m-bit CLAAs can be easily implemented with $k = \lceil n/m \rceil$ cascaded CLAAs.



$$t_{\text{m-bit CLA-based n-bit Adder}} = 2d + 2dk + 2d = 4d + 2d\lceil n/m \rceil$$

Faster Adders: Carry-Select Adder

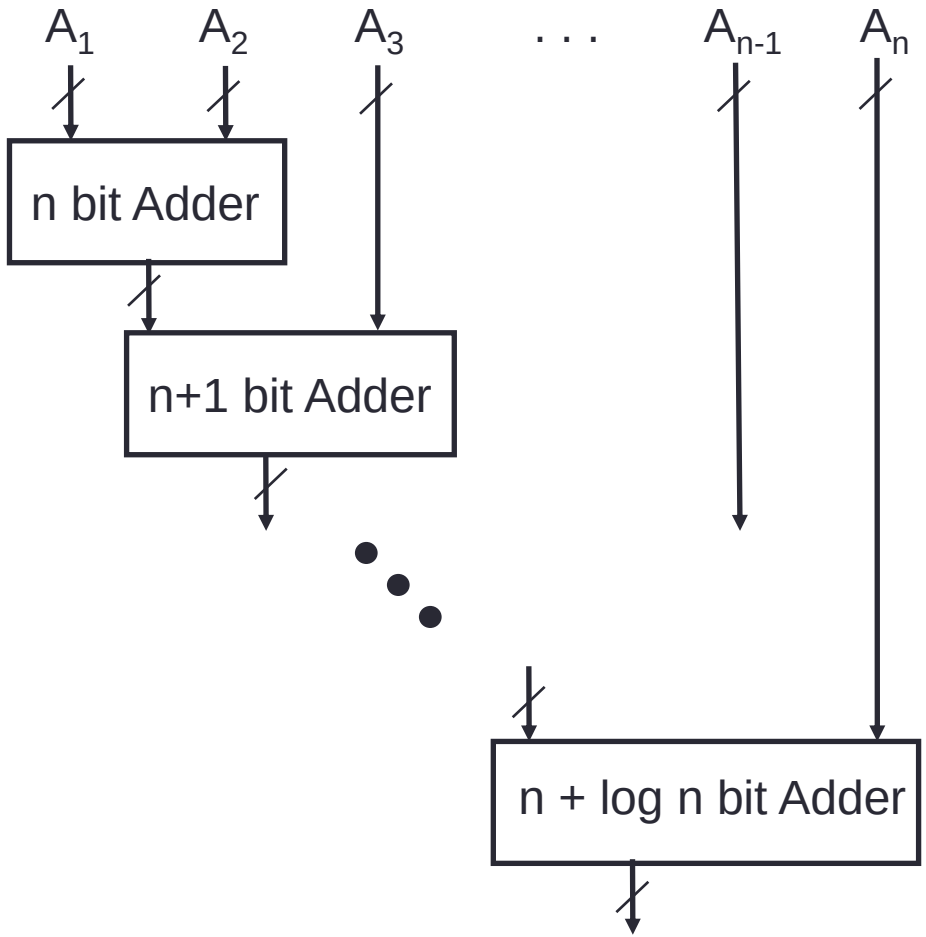
An n-bit carry-select adder (CSLA) using m-bit adders can be easily implemented with (2k-1) x m-bit adders and (k-1) x 2-input multiplexers (k = $\lceil n/m \rceil$).



$$t_{\text{m-bit based n-bit Adder}} = 2d(k-1) + t_{A(m)}$$

Multi-Operand Addition

Suppose we have more than two inputs to add. The simplest way to add n numbers is to add them in order as:

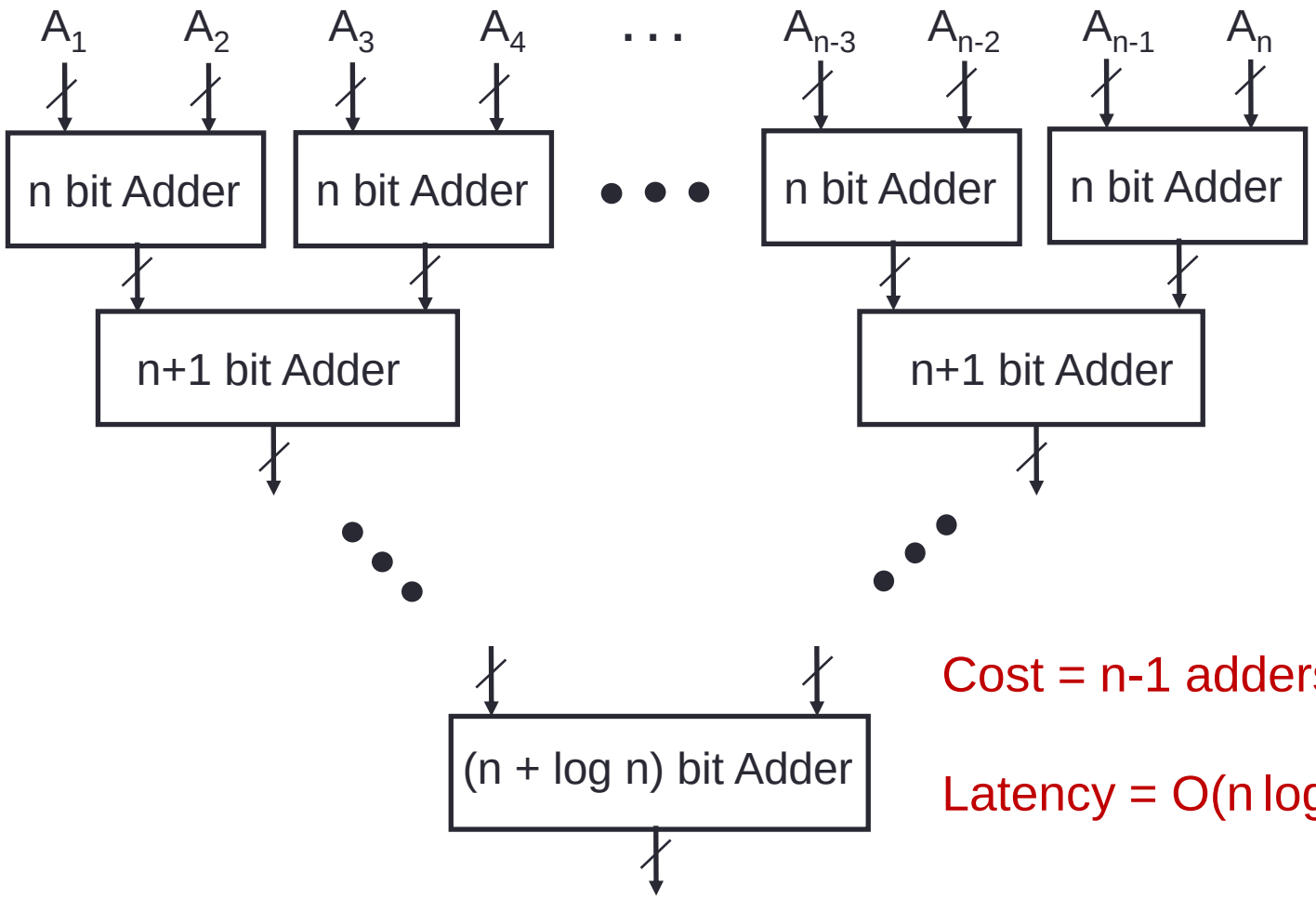


Cost = n-1 adders

Latency = $O(n^2)$ gate delay

Multi-Operand Addition

A faster design can use a binary tree based structure as:

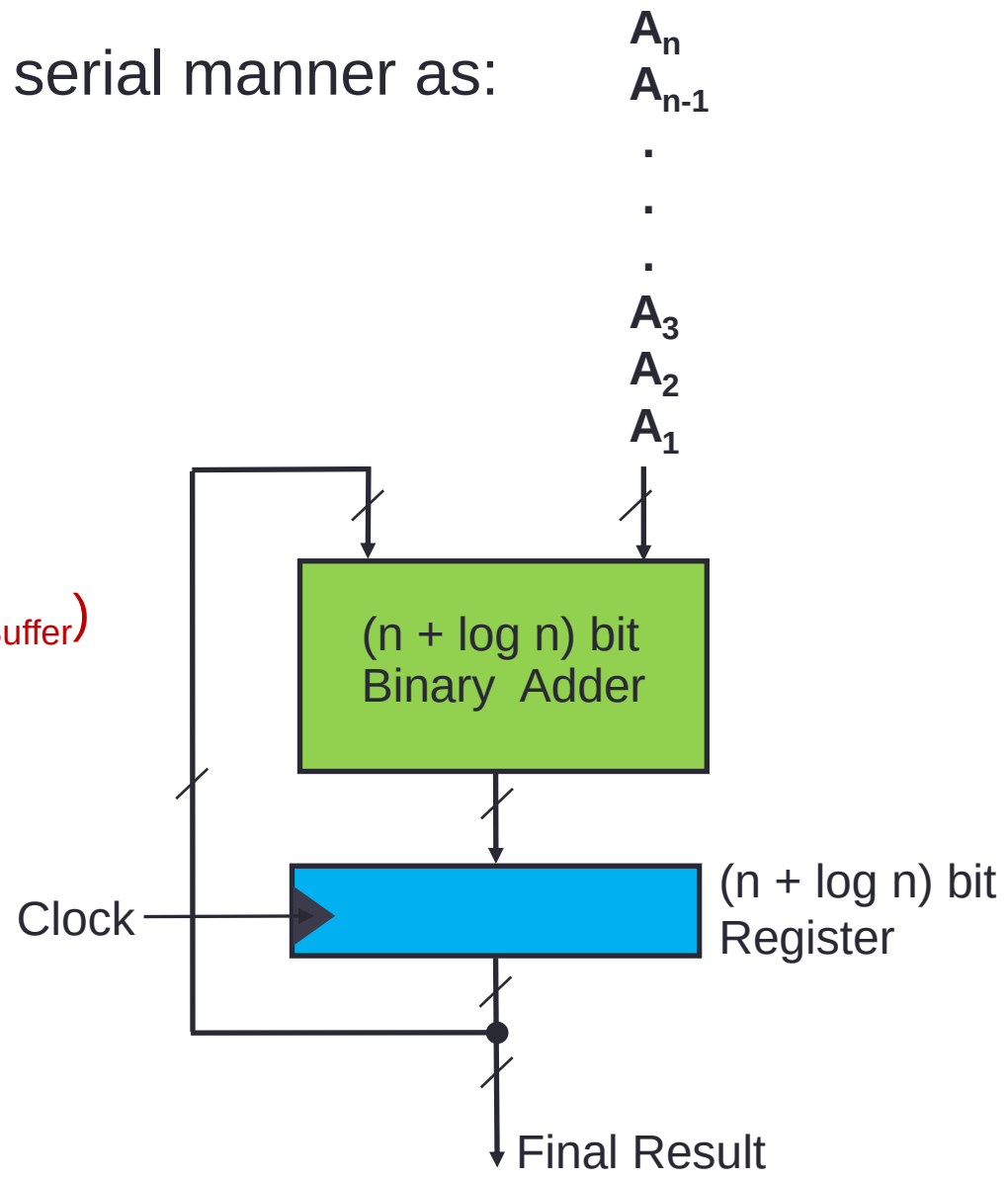


Cost = n-1 adders

Latency = O(n log n) gate delay

Multi-Operand Addition

Addition can be realized in a serial manner as:



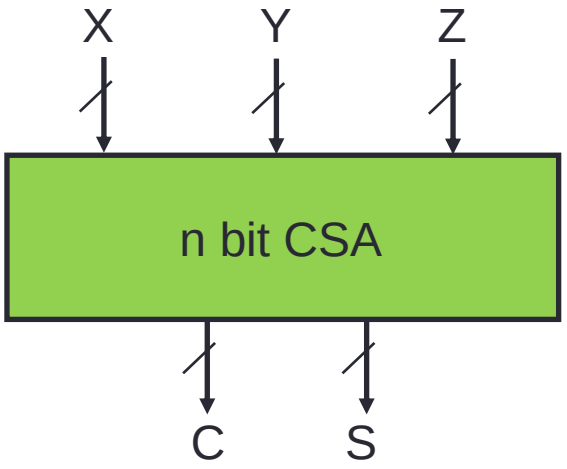
Cost = (n + log n) bit adder

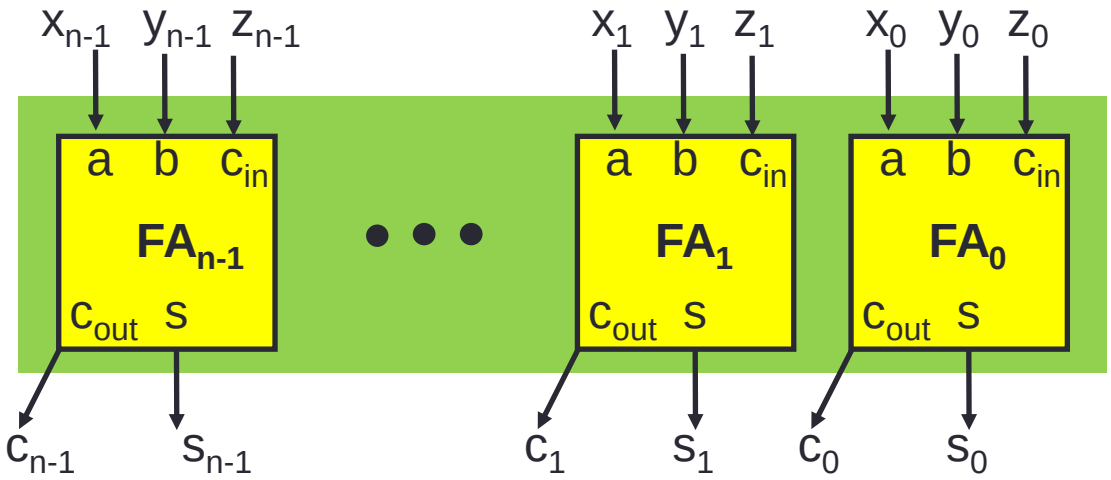
Latency = $nT > n(t_{(n+\log n)\text{-bit Adder}} + t_{Buffer})$

Multi-Operand Addition

Faster multi-operand addition can be done using carry save adders (CSA).

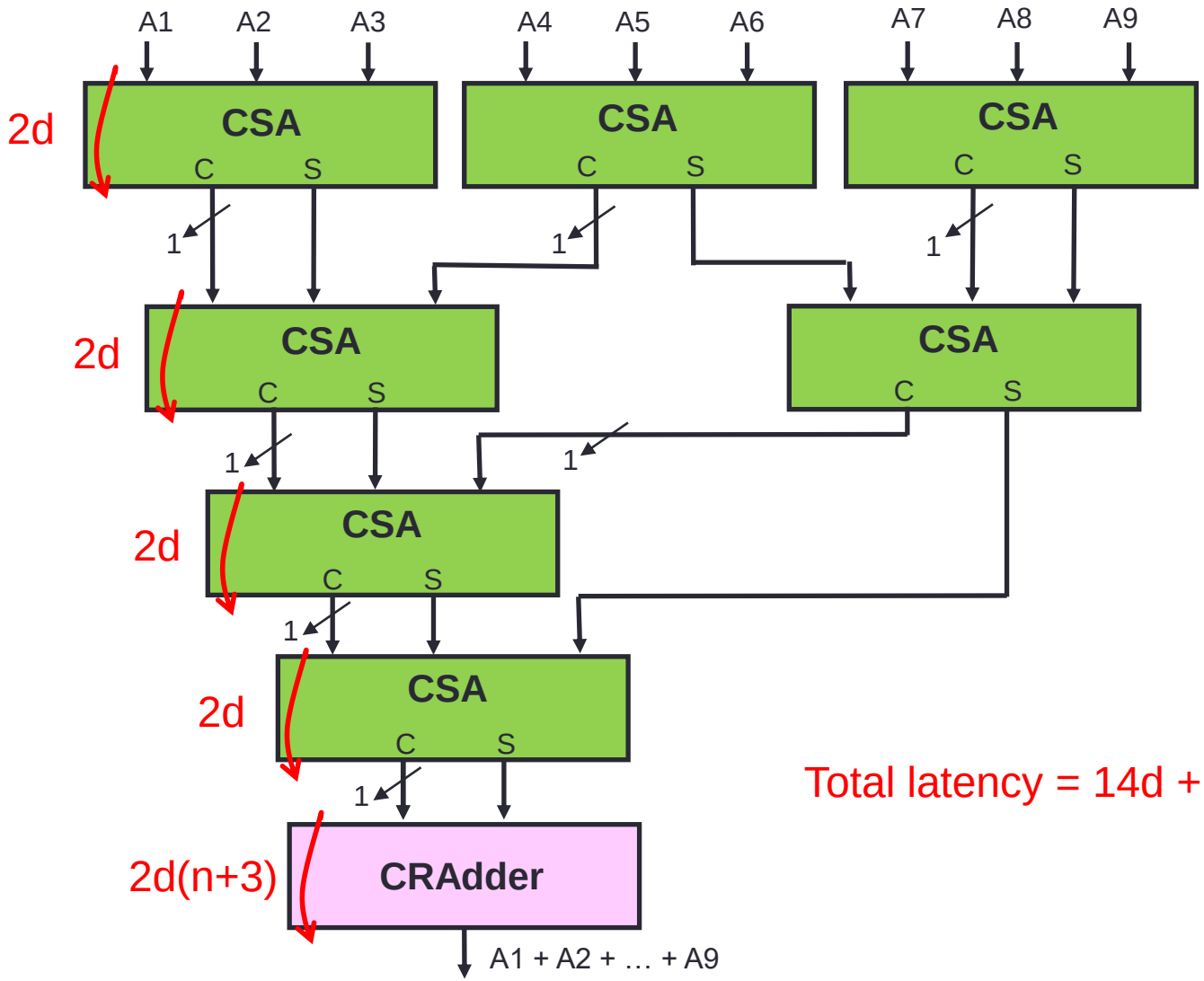
Any CSA reduces 3 inputs to 2.
It is also called a 3-to-2 Compressor.





Multi-Operand Addition

Using CSAs we can add n input numbers as:



Total latency = 14d + 2nd

Unsigned Binary Multiplication

Recall the pen and paper technique we use to multiply n-bit number A and m-bit number B to generate (m+n)-bit number P = A x B.

$$A = a_{n-1} \cdots a_1 a_0$$
$$B = b_{m-1} \cdots b_1 b_0$$

$$M_0 = a_{n-1}b_0 \cdots a_1 b_0 \; a_0 b_0$$
$$M_1 = a_{n-1}b_1 \cdots a_1 b_1 \; a_0 b_1$$
$$\vdots$$
$$M_{m-1} = a_{n-1}b_{m-1} \cdots a_1 b_{m-1} \; a_0 b_{m-1}$$

$$p_{m+n-1} \; p_{m+n-2} \cdots p_2 \; p_1 \; p_0$$

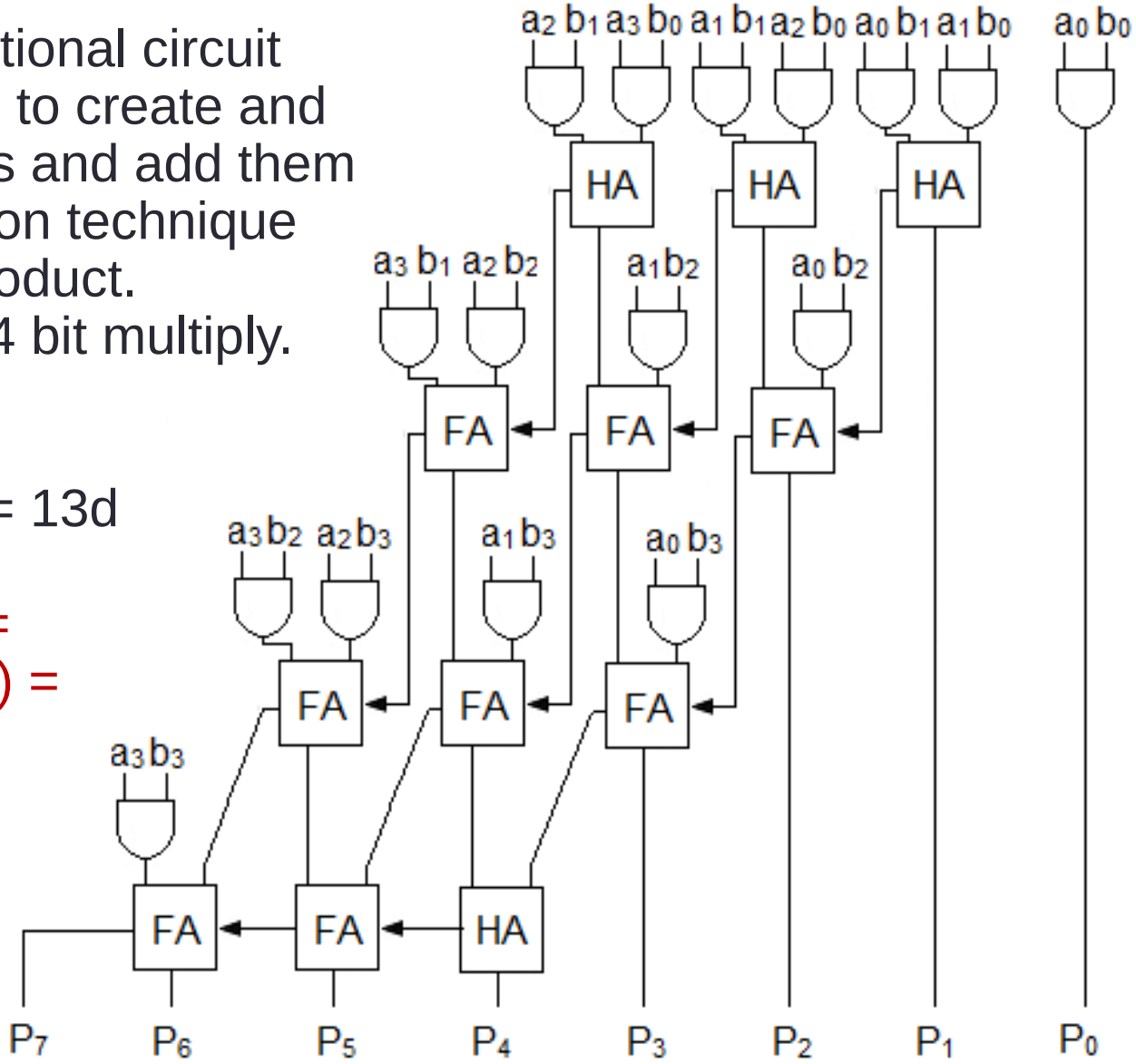
Unsigned Binary Multiplication

We can use a combinational circuit (called Multiplier Array) to create and add the partial products and add them using carry-save addition technique to generate the final product.
Here we show it for 4x4 bit multiply.

$Latency_{4x4\ bit\ multiplier} =$
$d + 2d + 2d + 2d + 6d = 13d$

→ $Latency_{nxn\ bit\ multiplier} =$
$d + 2d(n-1) + 2d(n-1) =$
$4nd - 3d$

$Cost_{nxn\ bit\ multiplier} =$
n^2 ANDs + n HAs +
$((n-1)^2-1)$ FAs



Unsigned Binary Multiplication

Multiplication can be done in a serial manner.

Consider n-bit numbers A($a_{n-1}a_{n-2} \ldots a_0$) and B($b_{n-1}b_{n-2} \ldots b_0$).

We can write:

$$A \times B = (A \times 2^{n-1} \times b_{n-1}) + (A \times 2^{n-2} \times b_{n-2}) + \ldots + (A \times 2^{1} \times b_1) + (A \times b_0)$$

$$= [A \ll (n-1)] \times b_{n-1} + [A \ll (n-2)] \times b_{n-2} + \ldots + [A \ll 1] \times b_1 + [A \ll 0] \times b_0$$

$$= \sum_{i=0..n-1} [A \ll i] \times b_i$$

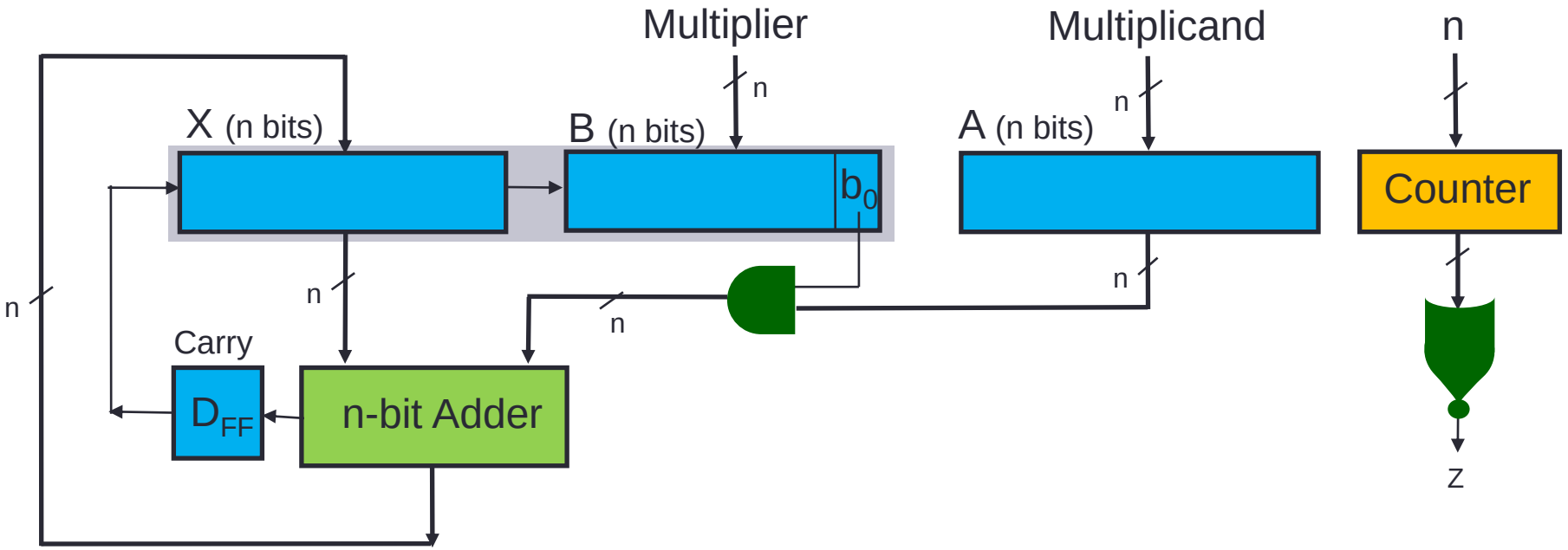
For shifting A for n-1 bits to the left, we need a 2n-bit register to keep the partial products.

Alternatively, we can shift the accumulator, that accumulates the partial products, to the right. → a tricky circuit to save hardware ☺

Unsigned Binary Multiplication

Add and Shift multiplier uses the following circuit (data path):

In the beginning: A and B keep the n-bit multiplicand and multiplier.
At the end: Register pairs X:B contain the 2n-bit multiply result.

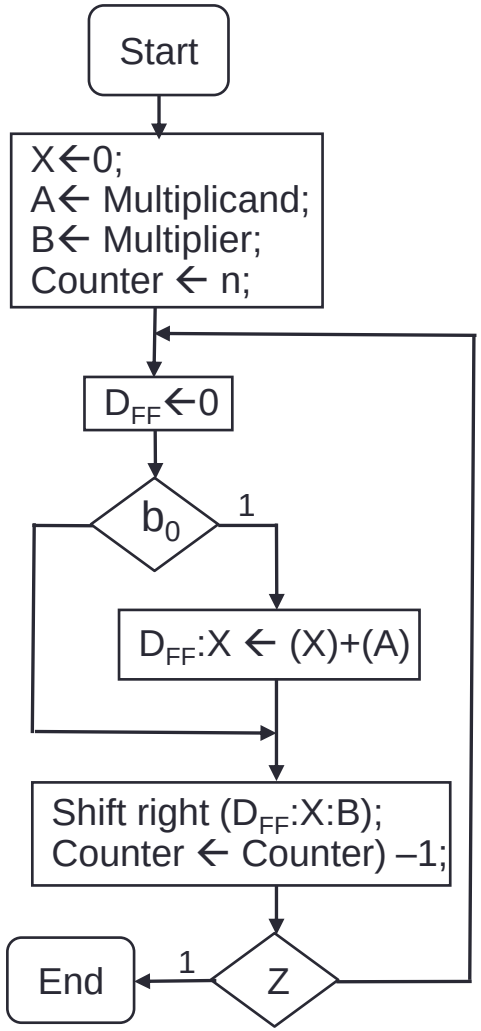


Unsigned Binary Multiplication

Add & Shift multiplication algorithm is shown.
Example 1: Follow Add & Shift method step by step to multiply 4-bit numbers 9 and 11. (A=1001)

D_{FF}	X	B	b_0	Counter	Operation
0	0000	1011		4	Add
0	1001	1011		4	Shift
0	0100	1101		3	Add
0	1101	1101		3	Shift
0	0110	1110		2	Shift
0	0011	0111		1	Add
0	1100	0111		1	Shift
0	0110	0011		0	End.

99 = 9 x 11

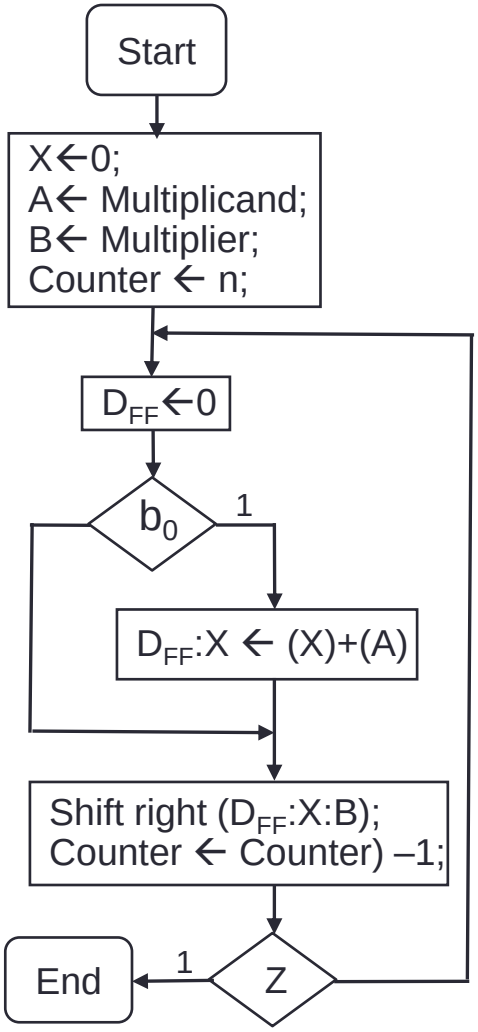


Unsigned Binary Multiplication

Add & Shift multiplication algorithm is shown.
Example 2: Follow Add & Shift method step by step to multiply 4-bit numbers 12 and 6. (A=1100)

D_{FF}	X	B	b_0	Counter	Operation
0	0000	0110	0	4	Shift
0	0000	0011	1	3	Add
0	1100	0011	1	3	Shift
0	0110	0001	1	2	Add
1	0010	0001	1	2	Shift
0	1001	0000	0	1	Shift
0	0100	1000	0	0	End.

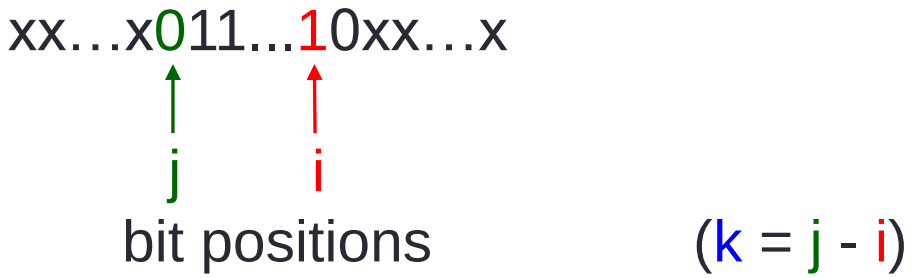
72 = 12 x 6



Unsigned Binary Multiplication

Multiplication based on Booth's scheme.

Assume in the n-bit multiplier we have a k-bit pattern of 1s:



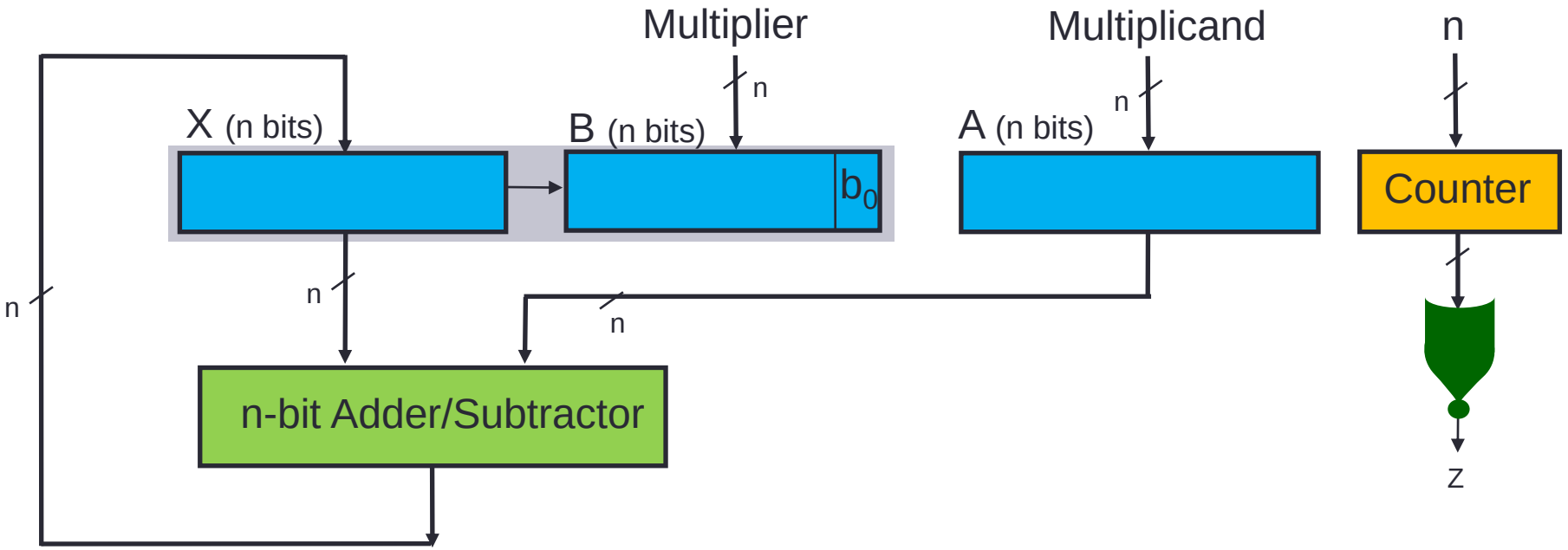
For the k-bit string 11..11, we must add A x $(2^i+2^{i+1}+\ldots+2^{j-1})$ to the accumulator. Obviously: $2^i+2^{i+1}+\ldots+2^{j-1} = 2^j - 2^i$

So, instead of adding A to the accumulator at positions i, i+1, …, j-1, we can subtract A from accumulator for the first 1 at position i and add A to the accumulator for the 0 at position j.

Unsigned Binary Multiplication

Booth's based multiplier uses the following circuit (data path):

In the beginning: A and B keep the n-bit multiplicand and multiplier.
At the end: Register pairs X:B contain the 2n-bit multiply result.



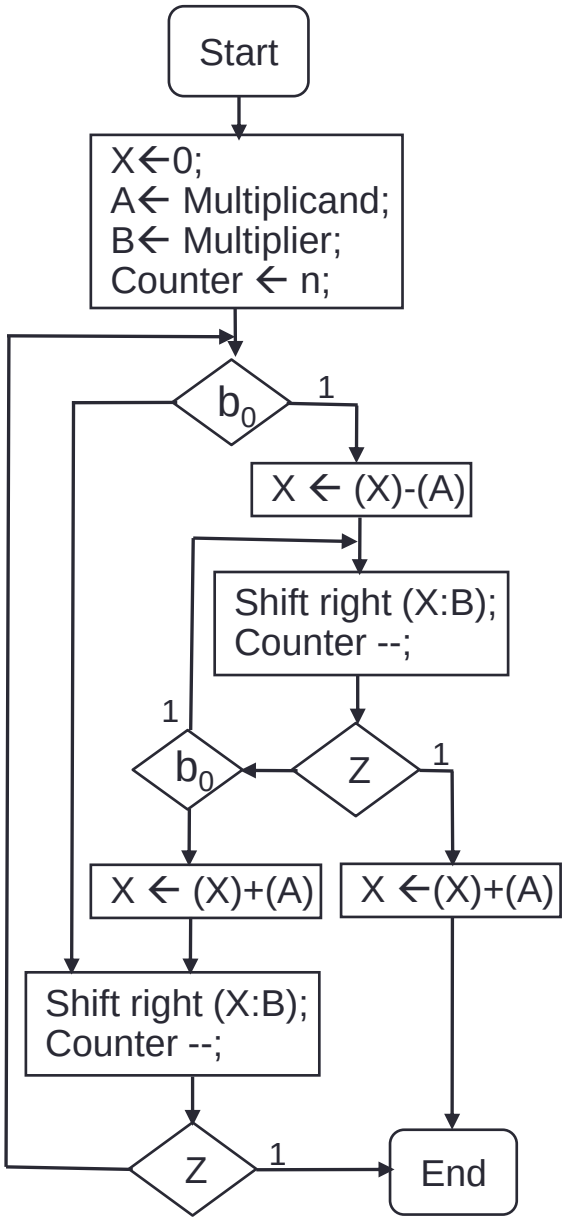
Unsigned Binary Multiplication

Booth's based multiplication algorithm is shown.

Example 1: Follow Booth's based multiplication algorithm step by step to multiply 4-bit numbers 7 and 6. (A=0111)

X	B	b_0	Counter	Operation
0000	0110		4	Shift
0000	0011		3	Subtract
1001	0011		3	Shift
1100	1001		2	Shift
1110	0100		1	Add
0101	0100		1	Shift
0010	1010		0	End.

42 = 7 x 6



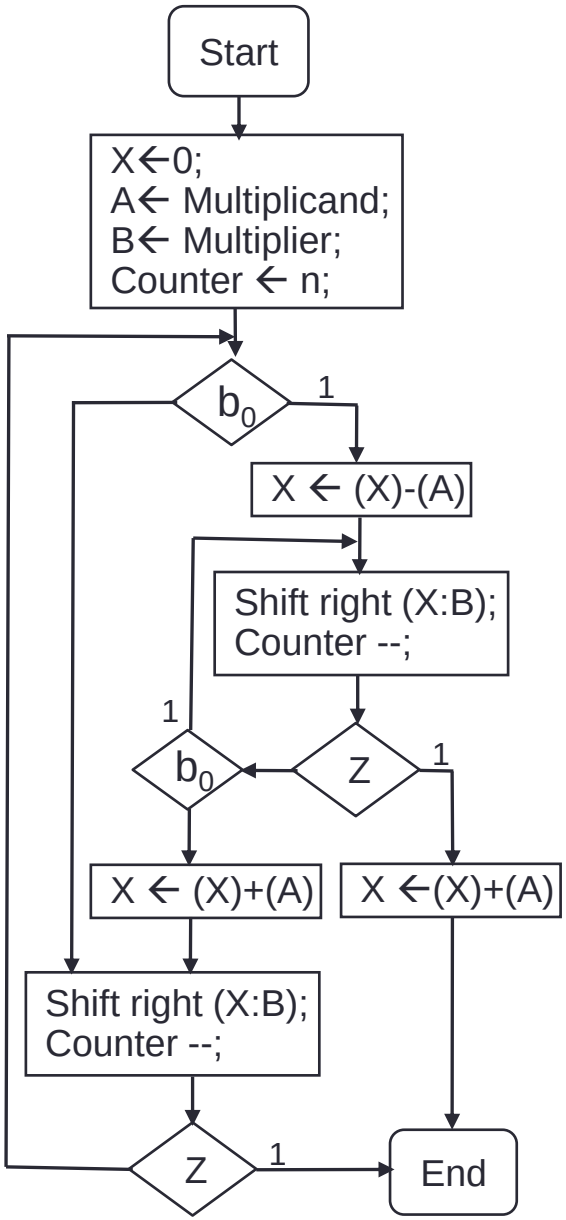
Unsigned Binary Multiplication

Booth's based multiplication algorithm is shown.

Example 2: Follow Booth's based multiplication algorithm step by step to multiply 4-bit numbers 5 and 11. (A=0101)

X	B	b_0	Counter	Operation
0000	1011	1	4	Subtract
1011	1011	1	4	Shift
1101	1101	1	3	Shift
1110	1110	0	2	Add
0011	1110	0	2	Shift
0001	1111	1	1	Subtract
1100	1111	1	1	Shift
1110	0111	1	0	z=1→ correction
0011	0111	1	0	End.

55 = 5 x 11



Signed Binary Multiplication

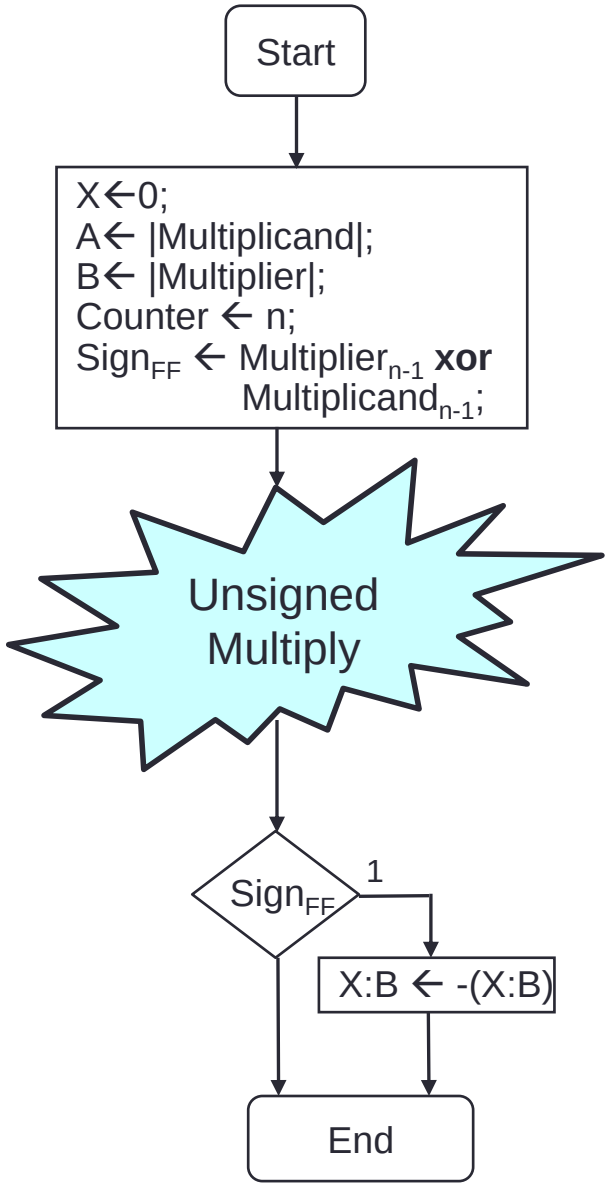
Signed multiplication can be done in two way:

- Indirect
- Direct

In indirect scheme: we first make the two input numbers positive and keep the sign bit of product. Then, we use one of the algorithms for unsigned multiplication to multiply the two positive numbers. At the end, the sign bit of product is applied to the result to make the final signed result.

→ We need some circuits:

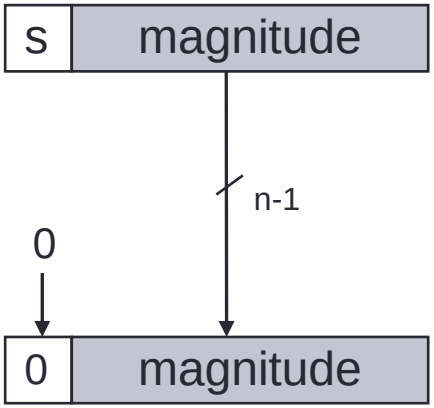
- to extract the absolute value of any signed input number
- to negate a positive number



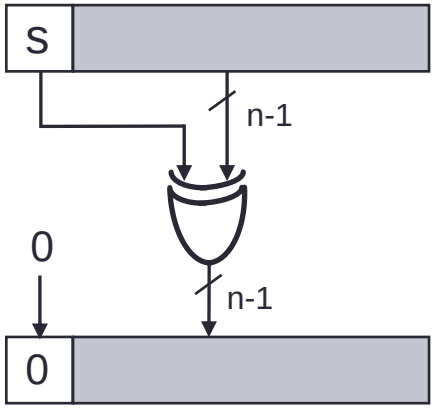
Signed Binary Multiplication

Circuit to extract the absolute value, in different representation systems:

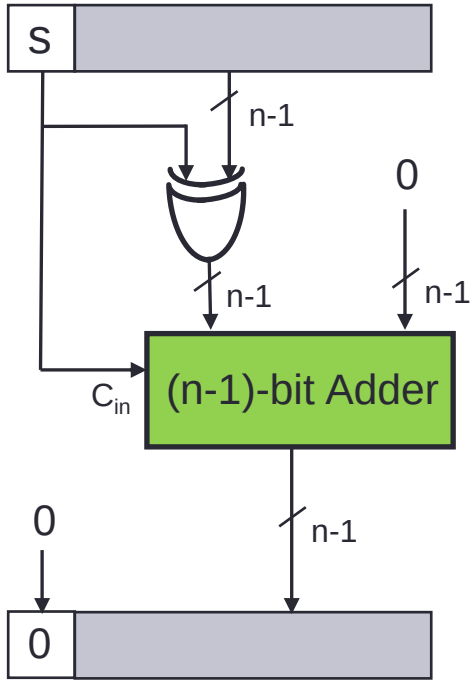
Sign-Magnitude



1's Complement



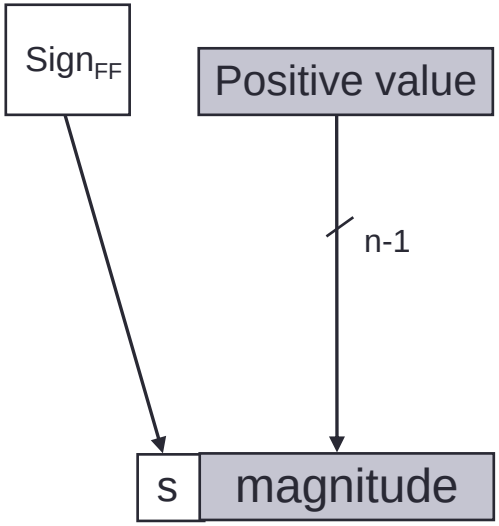
2's Complement



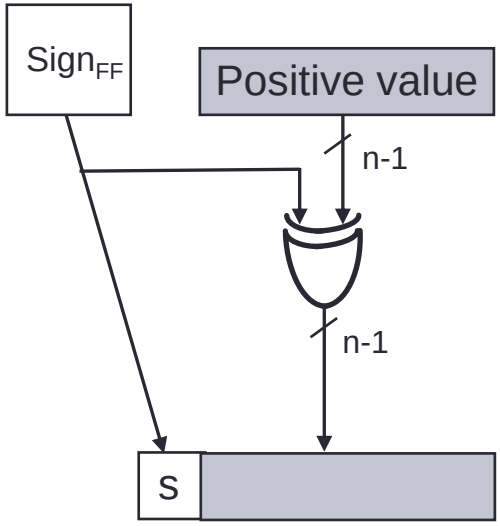
Signed Binary Multiplication

Circuit to negate a positive value for different representation systems:

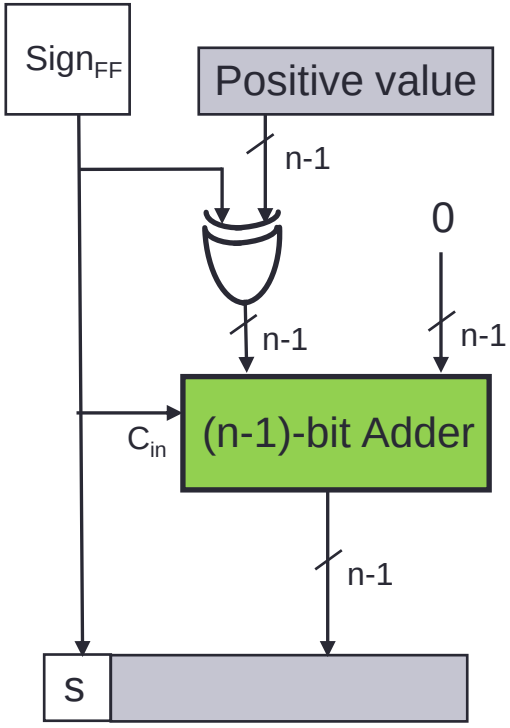
Sign-Magnitude



1's Complement



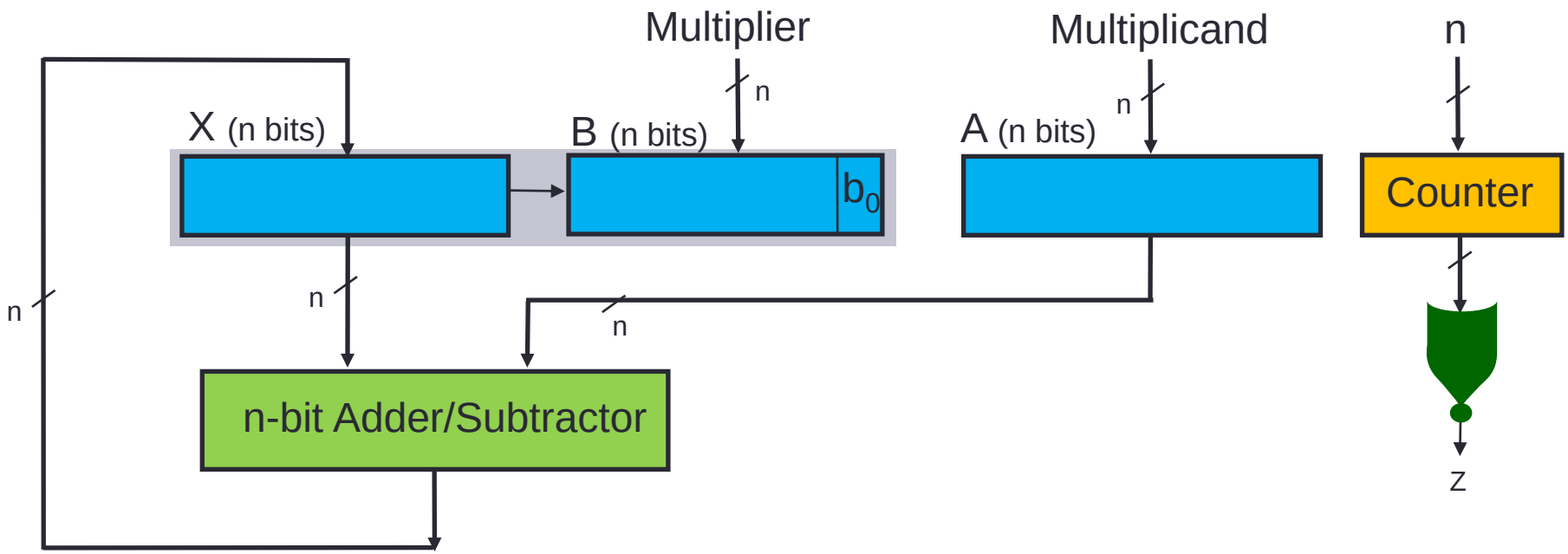
2's Complement



Signed Binary Multiplication

Booth's Algorithm: A Direct Signed Multiplication Method

The datapath used here is exactly what we used for unsigned multiplication.



Signed Binary Multiplication

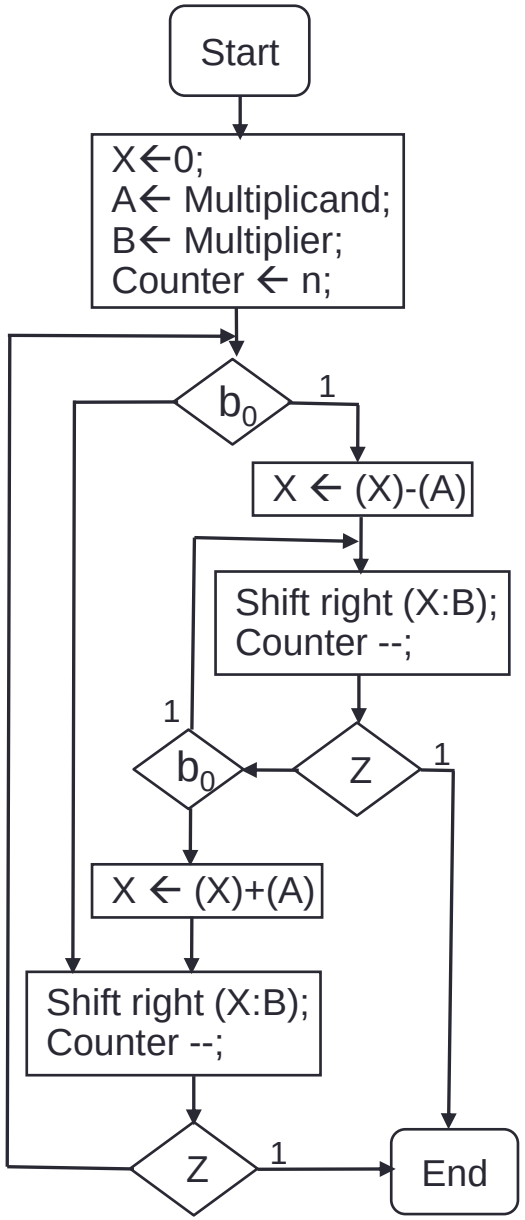
The acting chary for the Booth's signed multiplication scheme is shown.

Example 1: Follow Booth's signed multiplication algorithm step by step to multiply 4-bit numbers 7 and -5. (A=0111)

X	B	b_0	Counter	Operation
0000	1011	1	4	Subtract
1001	1011	1	4	Shift
1100	1101	1	3	Shift
1110	0110	1	2	Add
0101	0110	0	2	Shift
0010	1011	1	1	Subtract
1011	1011	1	1	Shift
1101	1101	1	0	End.

- 00100011

-35 = 7 x (-5)



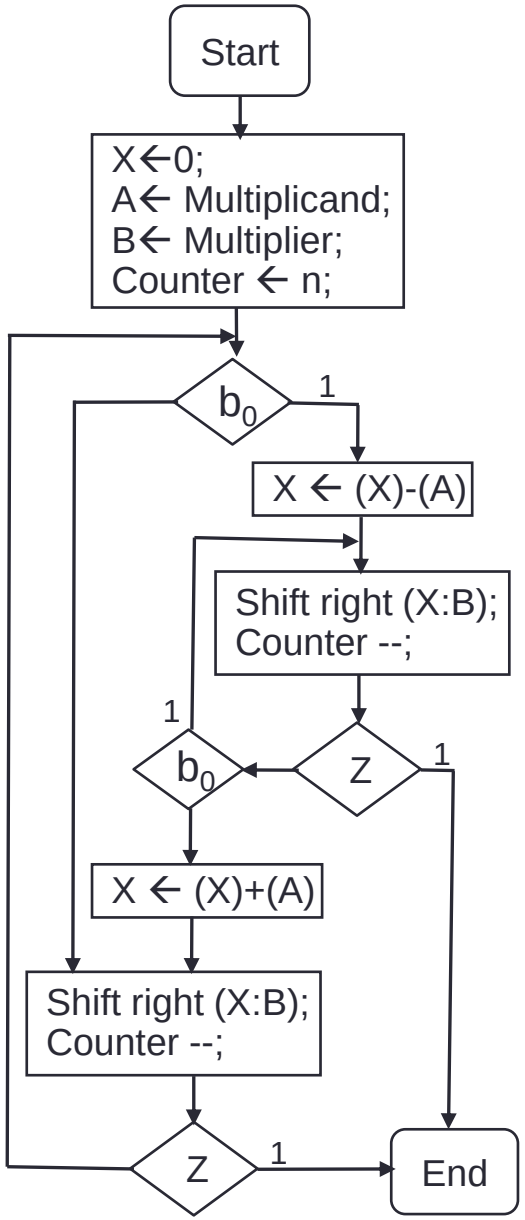
Signed Binary Multiplication

The acting chary for the Booth's signed multiplication scheme is shown.

Example 2: Follow Booth's signed multiplication algorithm step by step to multiply 4-bit numbers -7 and 6. (A=1001)

X	B	b_0	Counter	Operation
0000	0110		4	Shift
0000	0011		3	Subtract
0111	0011		3	Shift
0011	1001		2	Shift
0001	1100		1	Add
1010	1100		1	Shift
1101	0110		0	End.

- 00101010 **-42 = -7 x 6**



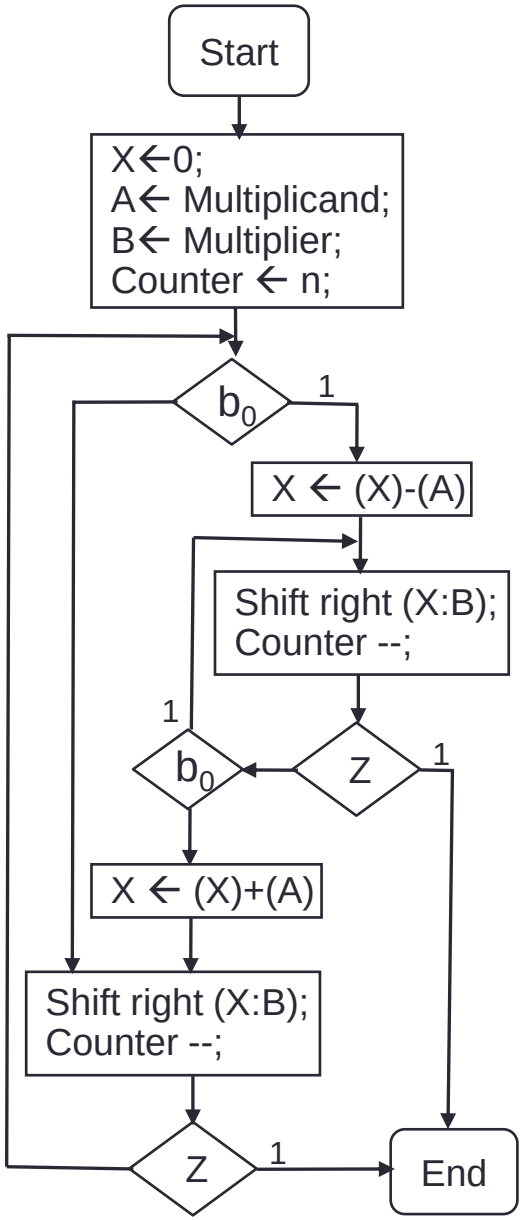
Signed Binary Multiplication

The acting chart for the Booth's signed multiplication scheme is shown.

Example 3: Follow Booth's signed multiplication algorithm step by step to multiply 4-bit numbers -7 and -5. (A=1001)

X	B	b_0	Counter	Operation
0000	1011		4	Subtract
0111	1011		4	Shift
0011	1101		3	Shift
0001	1110		2	Add
1010	1110		2	Shift
1101	0111		1	Subtract
0100	0110		1	Shift
0010	0011		0	End.

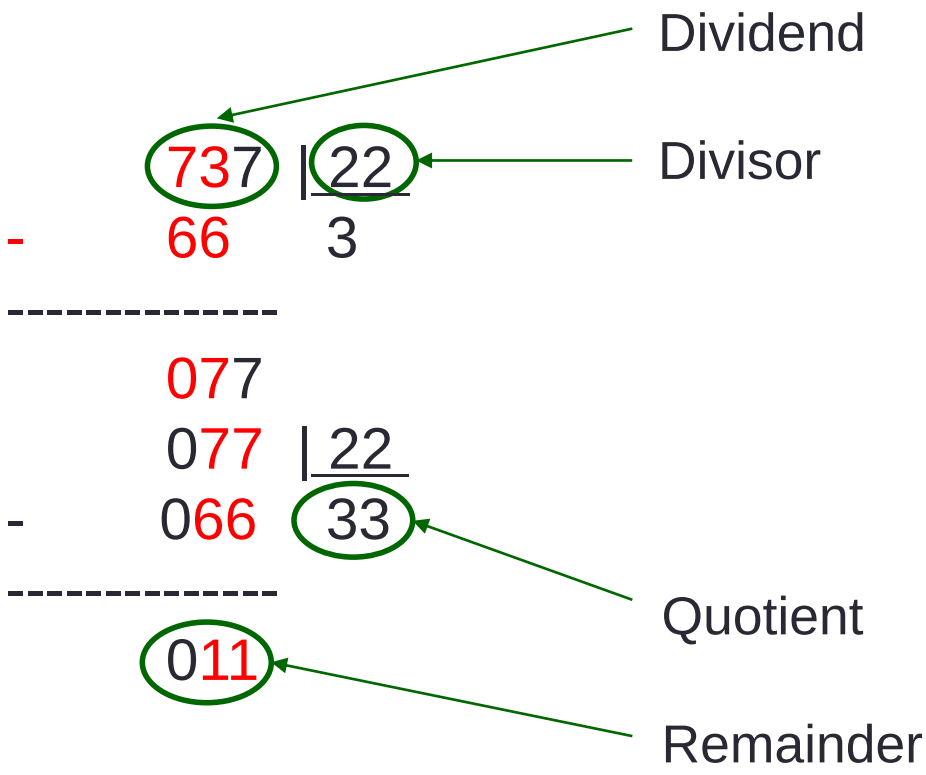
35 = -7 x (-5)



Unsigned Binary Division

Recall the pen & paper technique to divide two numbers.

Example: Divide 736 (Dividend) by 22 (Divisor).



Unsigned Binary Division

The pen & paper technique uses comparison for subtract possibility of divisor from dividend.

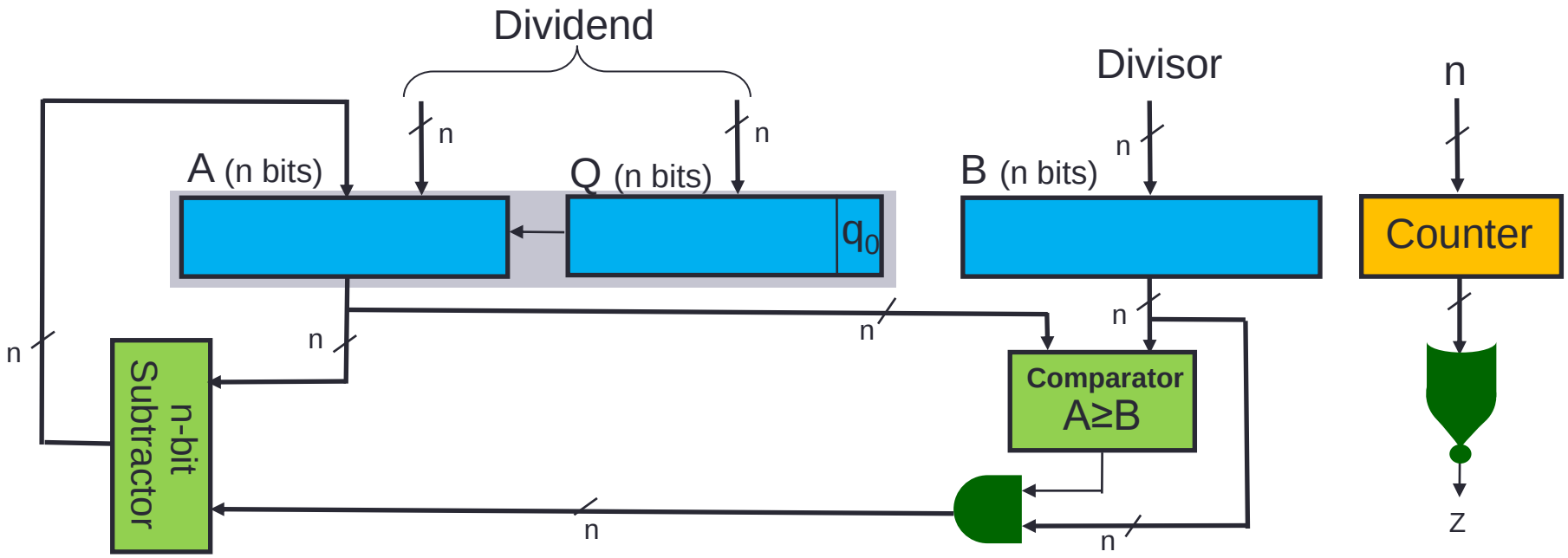
Division methods include:

- Comparison method
- Restoring method
- Non-restoring method

Unsigned Binary Division

Comparison method uses the following circuit (datapath):

In the beginning: Register pairs A and Q keep the 2n-bit dividend and register B keeps the n-bit divisor.
At the end: Register Q contains the n-bit quotient and register A contains the n-bit remainder.



Unsigned Binary Division

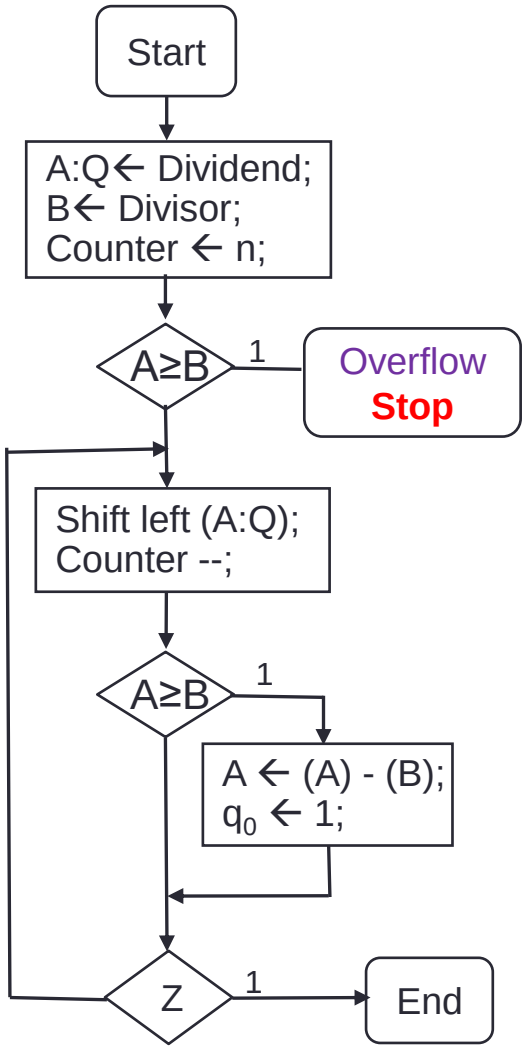
Comparison method algorithm is shown in the chart.

Example 1: Follow comparison method algorithm step by step to divide 37 by 5 (4-bit number). (B=0101)

A	Q	q_0	Counter	Operation
0010	0101		4	A<B → No Overflow
0010	0101		4	Shift
0100	1010		3	Shift
1001	0100		2	Subtract & set q_0
0100	0101		2	Shift
1000	1010		1	Subtract & set q_0
0011	1011		1	Shift
0111	0110		0	Subtract & set q_0
0010	0111		0	End.

2 = 37 – 7x5

7 = 37 / 5



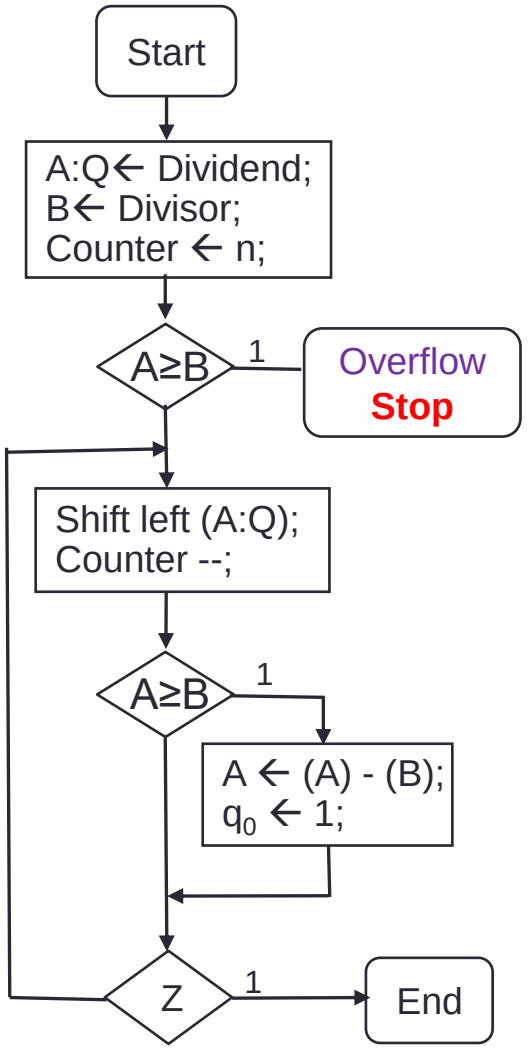
Unsigned Binary Division

Comparison method algorithm.

Example 2: Follow comparison method algorithm step by step to divide 44 by 8 (4-bit number). (B=1000)

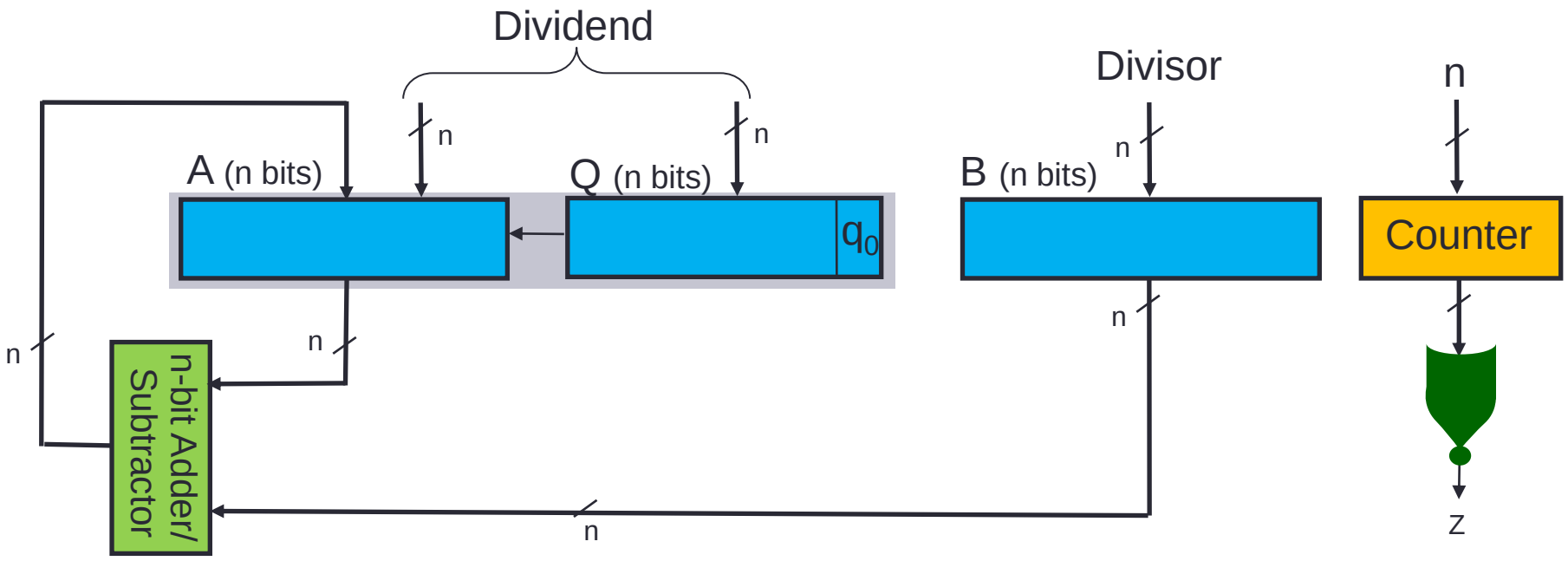
A	Q	q_0	Counter	Operation
0010	1100	0	4	A<B → No Overflow
0010	1100	0	4	Shift
0101	1000	0	3	Shift
1011	0000	0	2	Subtract & set q_0
0011	0001	1	2	Shift
0110	0010	0	1	Shift
1100	0100	0	0	Subtract & set q_0
0100	0101	1	0	End.

4 = 44 – 8x5 **5 = 44 / 8**



Unsigned Binary Division

Restoring method division algorithm uses the subtractor for comparison. It has the following circuit (data path).



Unsigned Binary Division

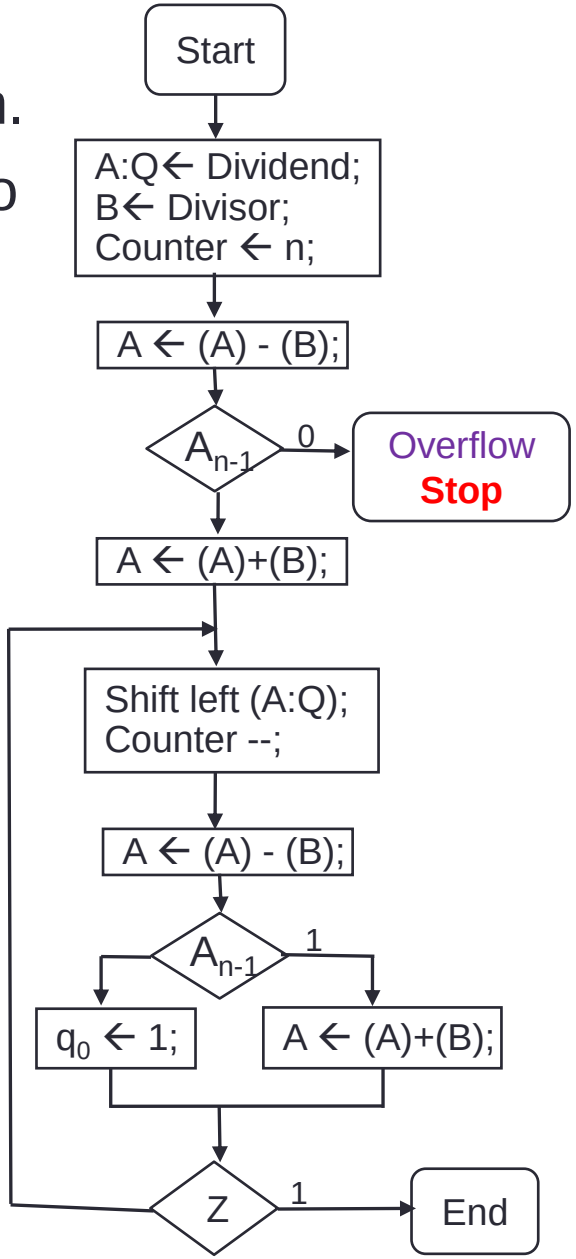
The chart of Restoring method division algorithm.

Example 2: Follow restoring method algorithm to divide 76 by 7 (4-bit number). (B=0111)

A	Q	q_0	Counter	Operation
0100	1100		4	Subtract
1101	1100		4	No Overflow→ Restore
0100	1100		4	Shift
1001	1000		3	Subtract
0010	1000		3	$q_0 \leftarrow 1$
0010	1001		3	Shift
0101	0010		2	Subtract
1110	0010		2	Restore
0101	0010		2	Shift
1010	0100		1	Subtract
0011	0100		1	$q_0 \leftarrow 1$
0011	0101		1	Shift
0110	1010		0	Subtract
1111	1010		0	Restore
0110	1010		0	End.

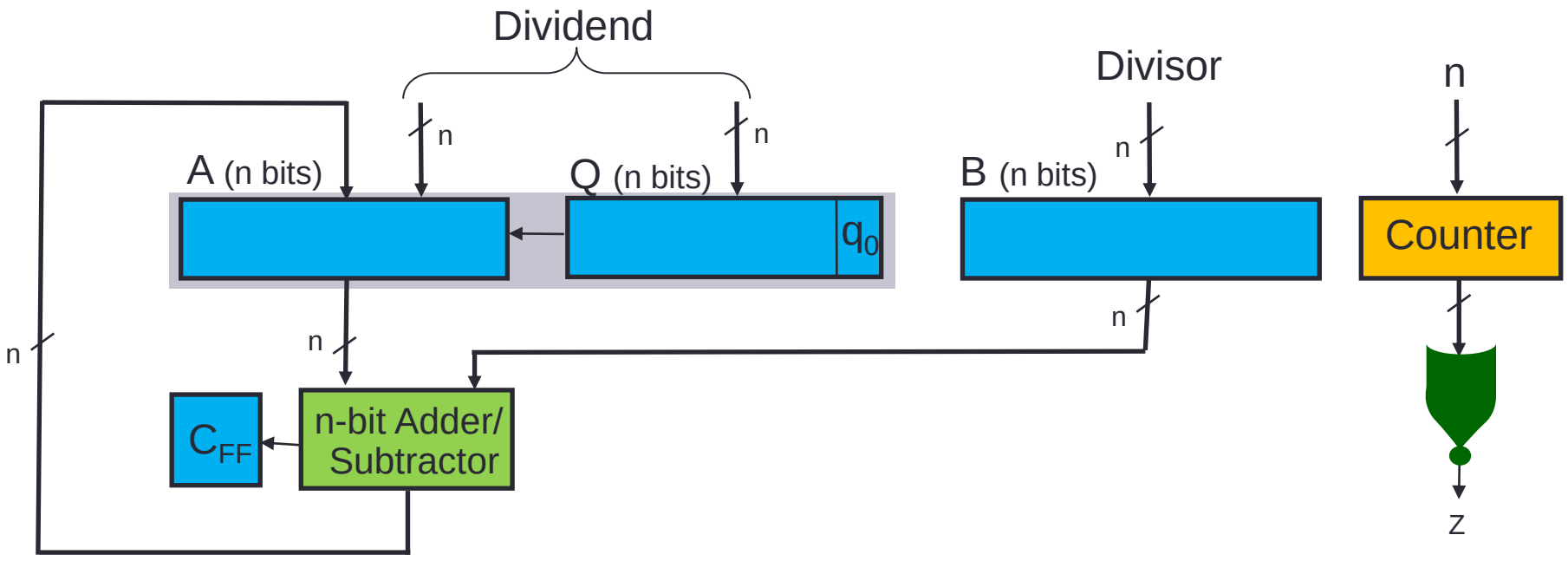
6 = 76 – 10x7

10 = 76 / 7



Unsigned Binary Division

Non-restoring method division algorithm uses the subtractor for comparison but do not restore the result if negative.
It has the following circuit (data path).



Unsigned Binary Division

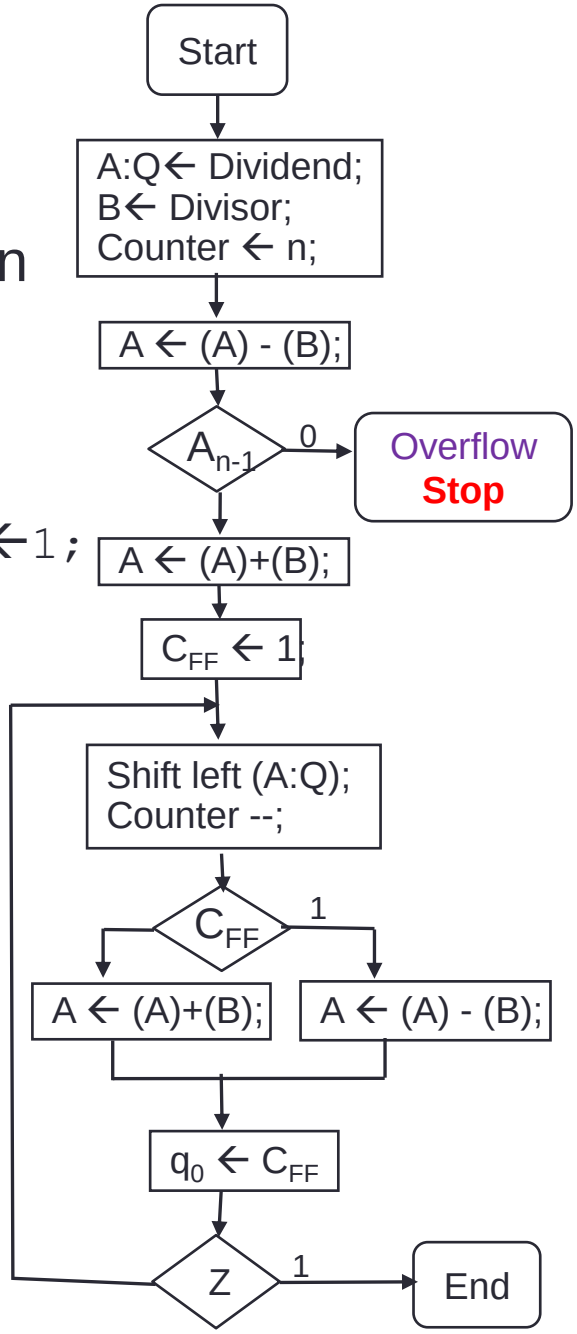
The chart of non-restoring method is shown.

Example 1: Follow non-restoring method division to divide 59 by 5 (4-bit number). (B=0101)

C_{FF}	A	Q	q_0	Counter	Operation
–	0011	1011		4	Subtract
0	1110	1011		4	No-overflow→Restore; C_{FF}←1;
1	0011	1011		4	Shift
1	0111	0110		3	Subtract (C_{FF}=1)
1	0010	0110		3	q_0 ←C_{FF};
1	0010	0111		3	Shift
1	0100	1110		2	Subtract (C_{FF}=0)
0	1111	1110		2	q_0 ←C_{FF};
0	1111	1110		2	Shift
0	1111	1100		1	Add (C_{FF}=1)
1	0100	1100		1	q_0 ←C_{FF};
1	0100	1101		1	Shift
1	1001	1010		0	Subtract (C_{FF}=1)
1	0100	1010		0	q_0 ←C_{FF};
1	0100	1011		0	End.

4 = 59 – 11x5

11 = 59 / 5



Unsigned Binary Division

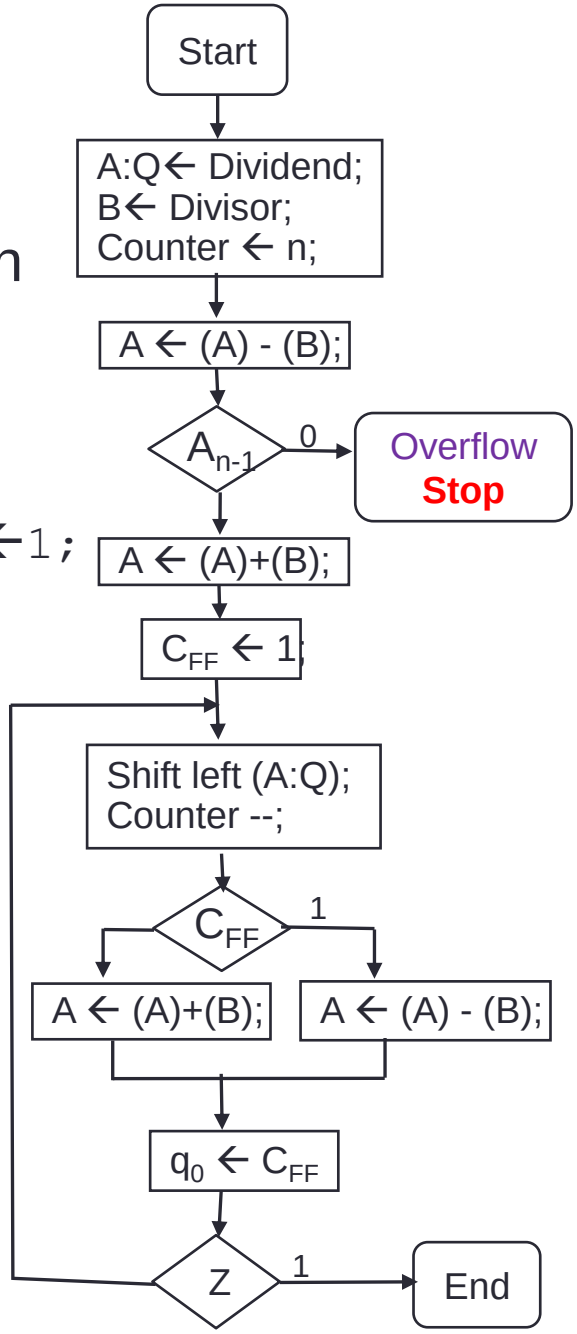
The chart of non-restoring method is shown.

Example 2: Follow non-restoring method division to divide 45 by 6 (4-bit number). (B=0110)

c_{FF}	A	Q	q_0	Counter	Operation
–	0010	1101		4	Subtract
0	1100	1101		4	No-overflow→Restore; C_{FF}←1;
1	0010	1101		4	Shift
1	0101	1010		3	Subtract (C_{FF}=0)
0	1111	1010		3	q_0 ←C_{FF};
0	1111	1010		3	Shift
0	1111	0100		2	Add (C_{FF}=1)
1	0101	0100		2	q_0 ←C_{FF};
1	0101	0101		2	Shift
1	1010	1010		1	Subtract (C_{FF}=1)
1	0100	1010		1	q_0 ←C_{FF};
1	0100	1011		1	Shift
1	1001	0110		0	Subtract (C_{FF}=1)
1	0011	0110		0	q_0 ←C_{FF};
1	0011	0111		0	End.

3 = 45 – 7x6

7 = 45 / 6



Decimal Arithmetic

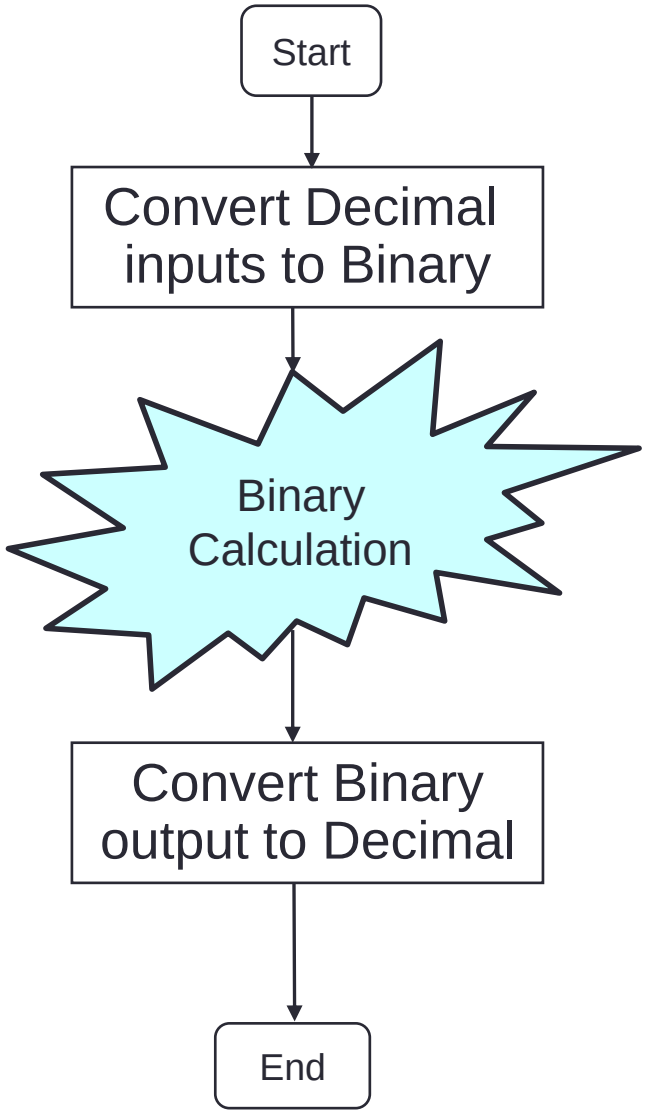
Decimal arithmetic can be done in two way:

- Indirect
- Direct

In Indirect scheme, we first convert the input decimal numbers to binary. Then, we use one of the algorithms for binary arithmetic we studied. At the end, the binary output is converted to decimal.

→ We need some circuits:

- Decimal to binary convertor
- Binary to decimal convertor



Decimal Arithmetic

Decimal to binary convertor:

The pen & paper technique uses successive divisions by 2 keeping the remainders.

Example: Convert the BCD number 74 to its binary equivalent.

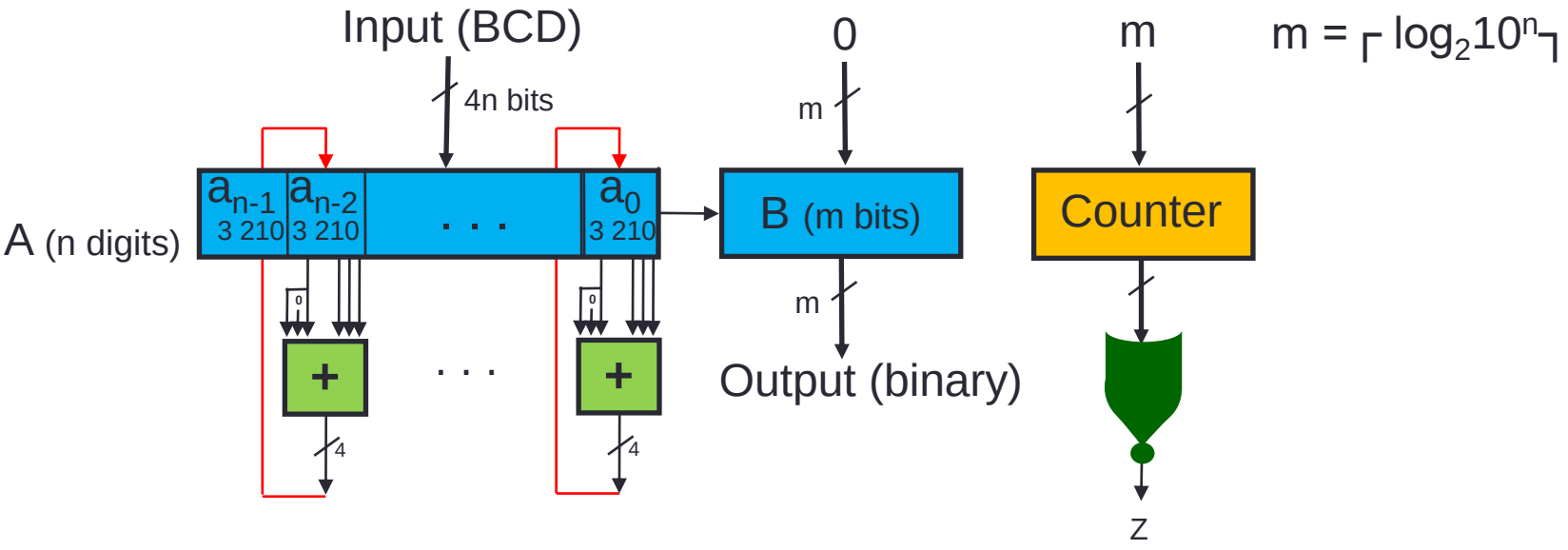
	Remainder
74 / 2 = 37	0
37 / 2 = 18	1
18 / 2 = 9	0
9 / 2 = 4	1
4 / 2 = 2	0
2 / 2 = 1	0
1 / 2 = 0	1

→ Writing the remainder values down to up, the binary equivalent will be: 1001010

Decimal Arithmetic

Decimal to binary convertor:

Consider the following data path. At start, input BCD number is in A register and at the end the output binary is in B register.



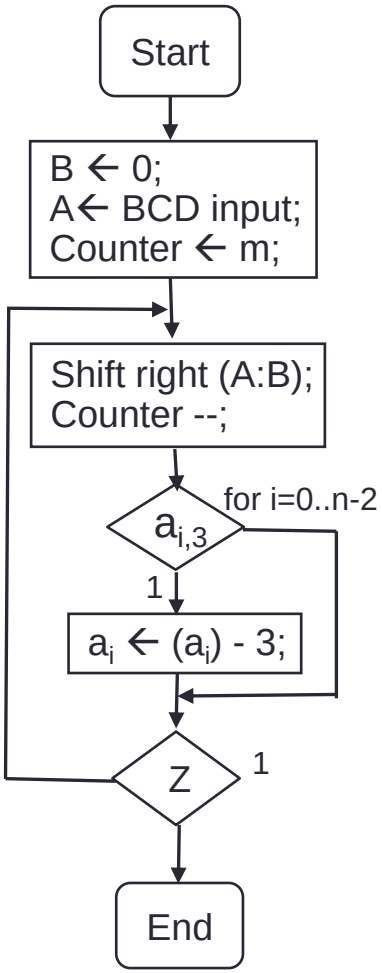
Decimal Arithmetic

Operation chart of decimal-to-binary convertor is shown.

Example 1: Follow BCD-to-BIN conversion to convert 371 to binary.

A	B	Counter	Operation
0011 0111 0001	0000000000	10	Shift
0001 1011 1000	1000000000	9	Correct a_1 a_0
0001 1000 0101	1000000000	9	Shift
0000 1100 0010	1100000000	8	Correct a_1
0000 1001 0010	1100000000	8	Shift
0000 0100 1001	0110000000	7	Correct a_0
0000 0100 0110	0110000000	7	Shift
0000 0010 0011	0011000000	6	Shift
0000 0001 0001	1001100000	5	Shift
0000 0000 1000	1100110000	4	Correct a_0
0000 0000 0101	1100110000	4	Shift
0000 0000 0010	1110011000	3	Shift
0000 0000 0001	0111001100	2	Shift
0000 0000 0000	1011100110	1	Shift
0000 0000 0000	0101110011	0	End.

371 = 3+16+32+64+256



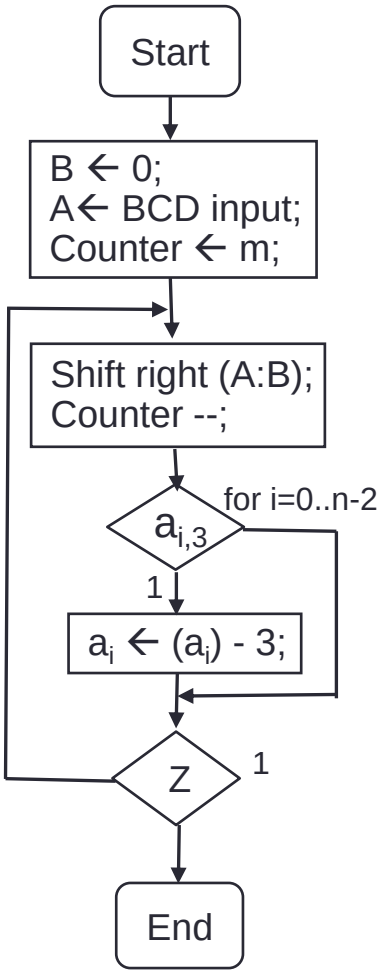
Decimal Arithmetic

Operation chart of decimal-to-binary convertor is shown.

Example 2: Follow BCD-to-BIN conversion to convert 555 to binary.

A	B	Counter	Operation
0101 0101 0101	0000000000	10	Shift
0010 1010 1010	1000000000	9	Correct a_1 a_0
0010 0111 0111	1000000000	9	Shift
0001 0011 1011	1100000000	8	Correct a_1
0001 0011 1000	1100000000	8	Shift
0000 1001 1100	0110000000	7	Correct a_1 a_0
0000 0110 1001	0110000000	7	Shift
0000 0011 0100	1011000000	6	Shift
0000 0001 1010	0101100000	5	Correct a_0
0000 0001 0111	0101100000	5	Shift
0000 0000 1011	1010110000	4	Correct a_0
0000 0000 1000	1010110000	4	Shift
0000 0000 0100	0101011000	3	Shift
0000 0000 0010	0010101100	2	Shift
0000 0000 0001	0001010110	1	Shift
0000 0000 0000	1000101011	0	End.

555 = 3+8+32+512



Decimal Arithmetic

Binary to decimal convertor:

The pen & paper technique uses multiplication of each bit of the input binary number to its position weight and adding it to the accumulator in decimal format.

Example: Convert the binary number 0011001100 to its BCD equivalent.

$0011001100 = 1x2^2 + 1x2^3 + 1x2^6 + 1x2^7$

$= 4 + 8 + 64 + 128$

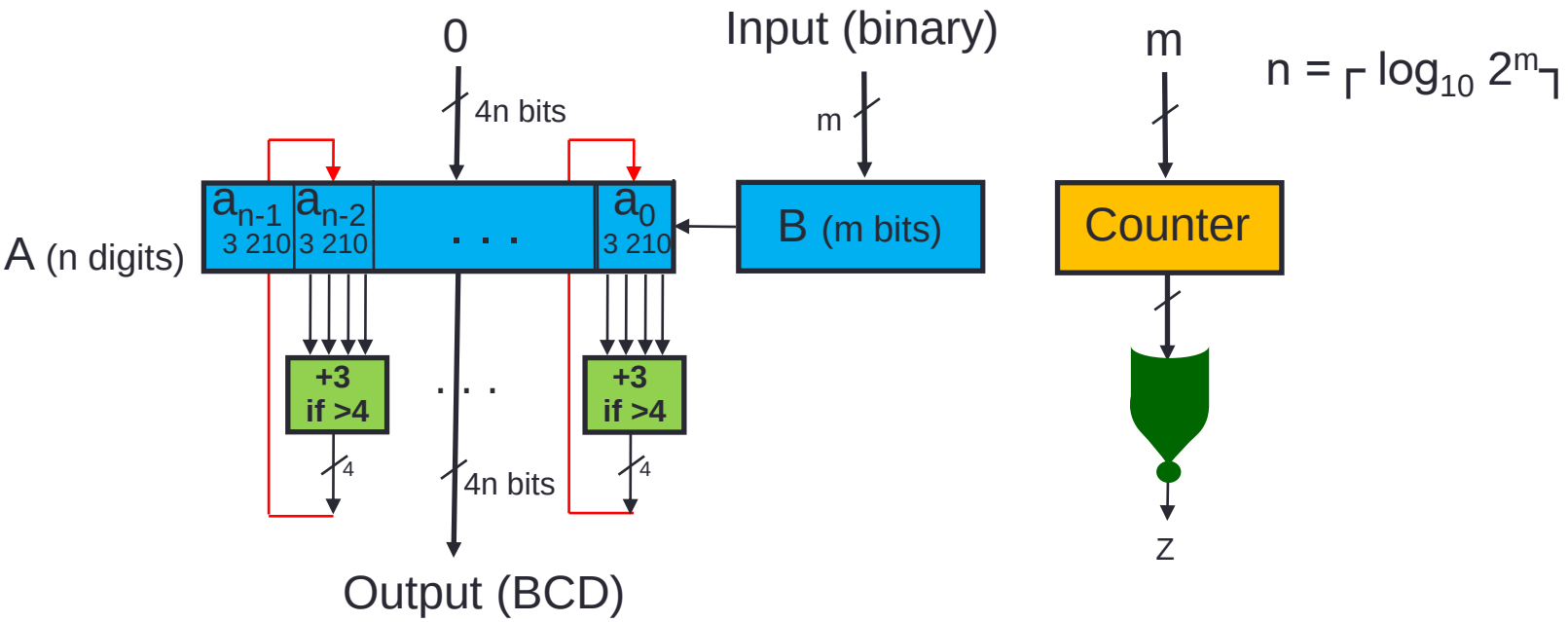
$= 204_{(10)}$

$= 0010\ 0000\ 0100\ _{(BCD)}$

Decimal Arithmetic

Binary to decimal convertor:

Consider the following data path. At start, input binary number is in B register and at the end the output BCD number is in A register.



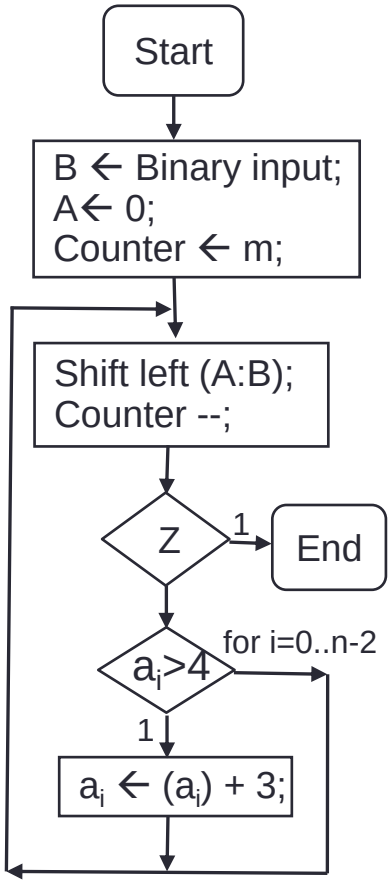
Decimal Arithmetic

Operation chart of binary-to-decimal convertor is shown.

Example 1: Follow BIN-to-BCD conversion to convert 0111011100 to binary.

A	B	Counter	Operation
0000 0000 0000	0111011100	10	Shift
0000 0000 0000	1110111000	9	Shift
0000 0000 0001	1101110000	8	Shift
0000 0000 0011	1011100000	7	Shift
0000 0000 0111	0111000000	6	Correct a_0
0000 0000 1010	0111000000	6	Shift
0000 0001 0100	1110000000	5	Shift
0000 0010 1001	1100000000	4	Correct a_0
0000 0010 1100	1100000000	4	Shift
0000 0101 1001	1000000000	3	Correct a_1 a_0
0000 1000 1100	1000000000	3	Shift
0001 0001 1001	0000000000	2	Correct a_0
0001 0001 1100	0000000000	2	Shift
0010 0011 1000	0000000000	1	Correct a_0
0010 0011 1011	0000000000	1	Shift
0100 0111 0110	0000000000	0	End.

476 = 4+8+16+64+128+256



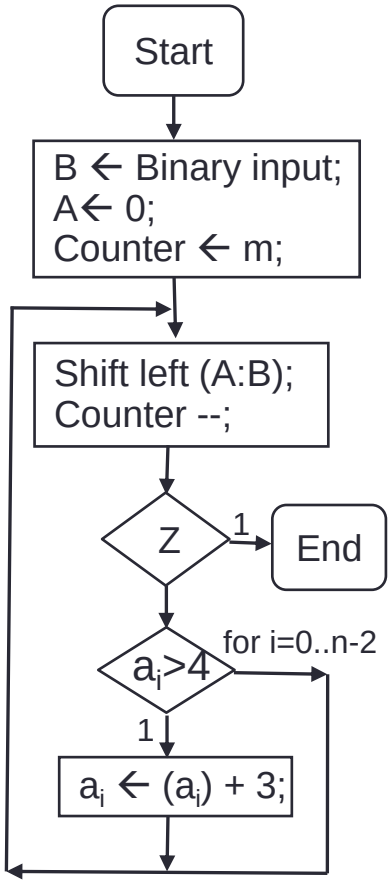
Decimal Arithmetic

Operation chart of binary-to-decimal convertor is shown.

Example 2: Follow BIN-to-BCD conversion to convert 0011111100 to binary.

A	B	Counter	Operation
0000 0000	0000 0011111100	10	Shift
0000 0000	0000 0111111000	9	Shift
0000 0000	0000 1111110000	8	Shift
0000 0000	0001 1111100000	7	Shift
0000 0000	0011 1111000000	6	Shift
0000 0000	0111 1110000000	5	Correct a_0
0000 0000	1010 1110000000	5	Shift
0000 0001	0101 1100000000	4	Correct a_0
0000 0001	1000 1100000000	4	Shift
0000 0011	0001 1000000000	3	Shift
0000 0110	0011 0000000000	2	Correct a_1
0000 1001	0011 0000000000	2	Shift
0001 0010	0110 0000000000	1	Correct a_0
0001 0010	1001 0000000000	1	Shift
0010 0101	0010 0000000000	0	End.

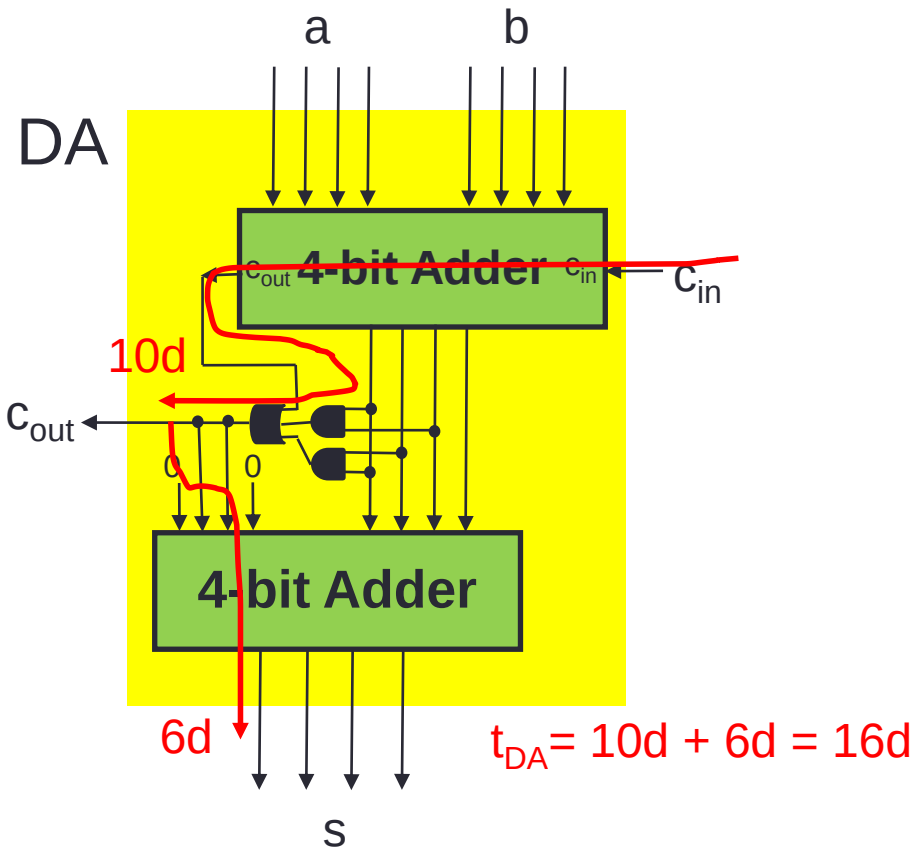
252 = 4+8+16+32+64+128



Decimal Arithmetic

Direct Method: Addition

One-digit decimal adder



All possible results of adding two decimal (BCD) digits with carry

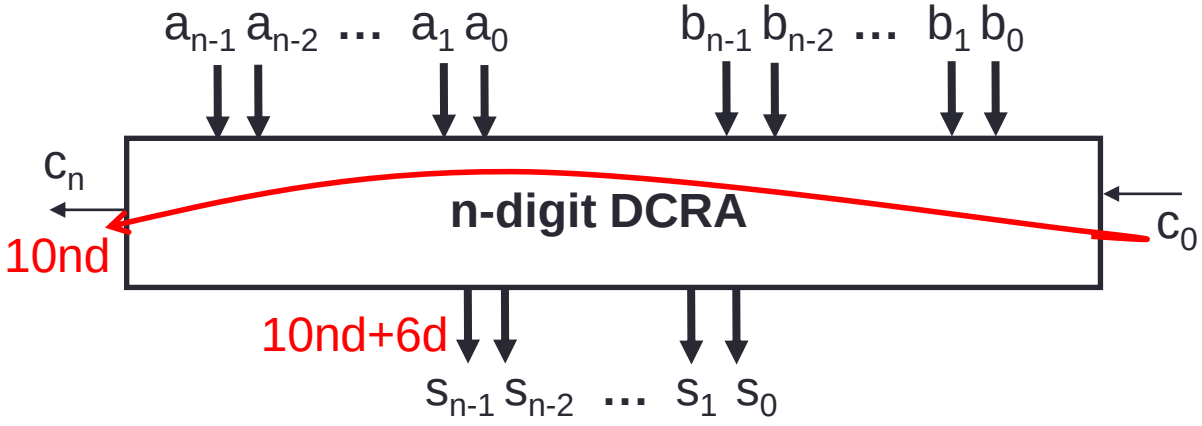
$a + b + c_{in}$			c_{out} s
0 0000		→	0 0000
0 0001		→	0 0001
0 0010		→	0 0010
……		→	……
0 1001		→	0 1001
0 1010	+6	→	1 0000
0 1011	+6	→	1 0001
0 1100	+6	→	1 0010
0 1101	+6	→	1 0011
0 1110	+6	→	1 0100
0 1111	+6	→	1 0101
1 0000	+6	→	1 0110
1 0001	+6	→	1 0111
1 0010	+6	→	1 1000
1 0011	+6	→	1 1001

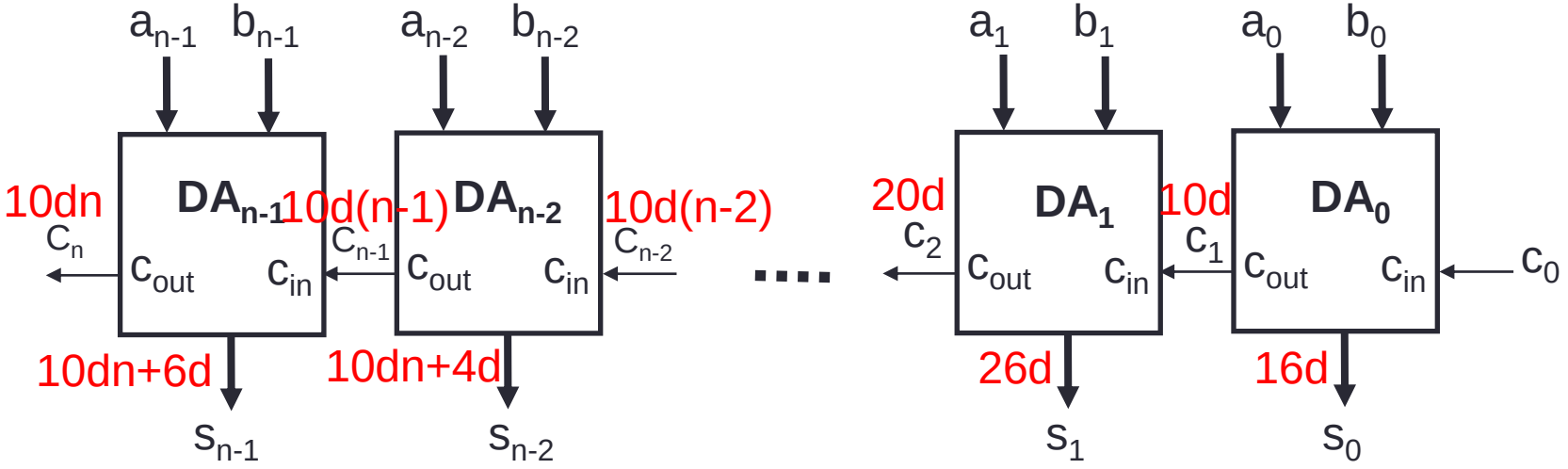
Decimal Arithmetic

Direct Method: n-digit Decimal Parallel Adder

An n-digit decimal carry-ripple adder (DCRA) can be easily implemented with n cascaded DAs.

$t_{\text{n-digit DCRA}} = 10nd + 6d$





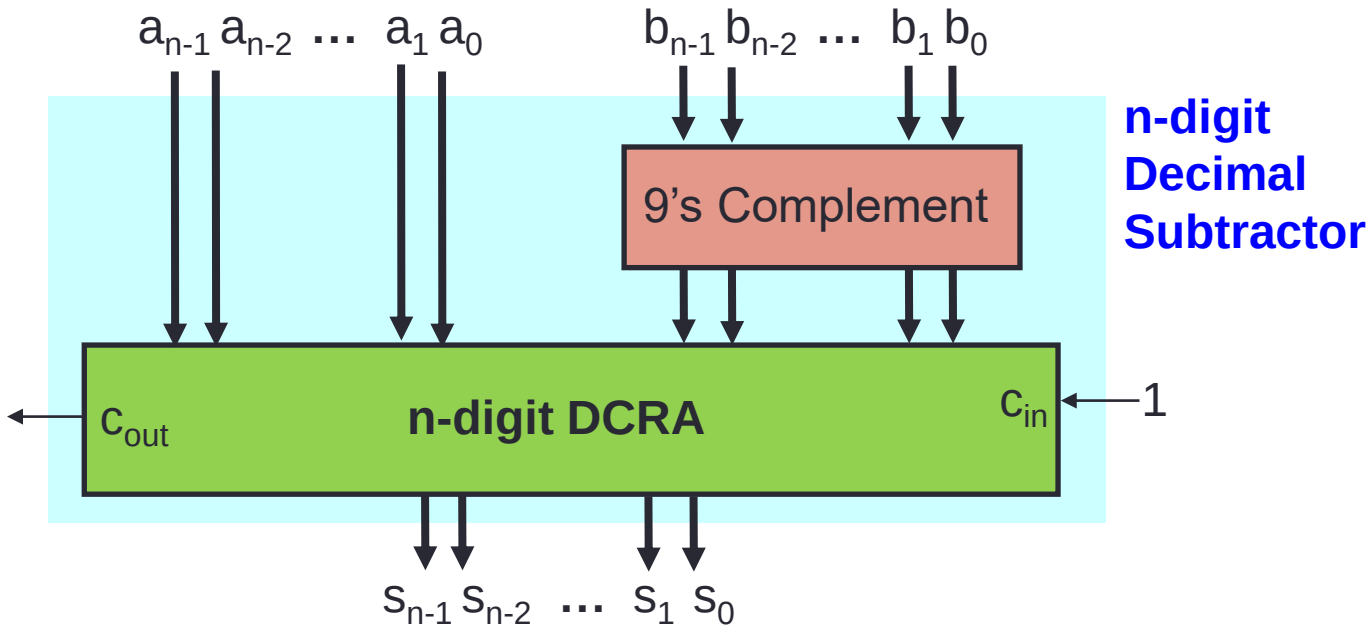
Decimal Arithmetic

Direct Method: n-digit Decimal Parallel Subtractor

An n-digit decimal subtractor can be built by n-digit DCRA.

$t_{\text{n-digit decimal subtractor}} = t_{\text{9's complement}} + 10nd + 6d$

If decimal code is self-complement then, $t_{\text{9's complement}} = d$



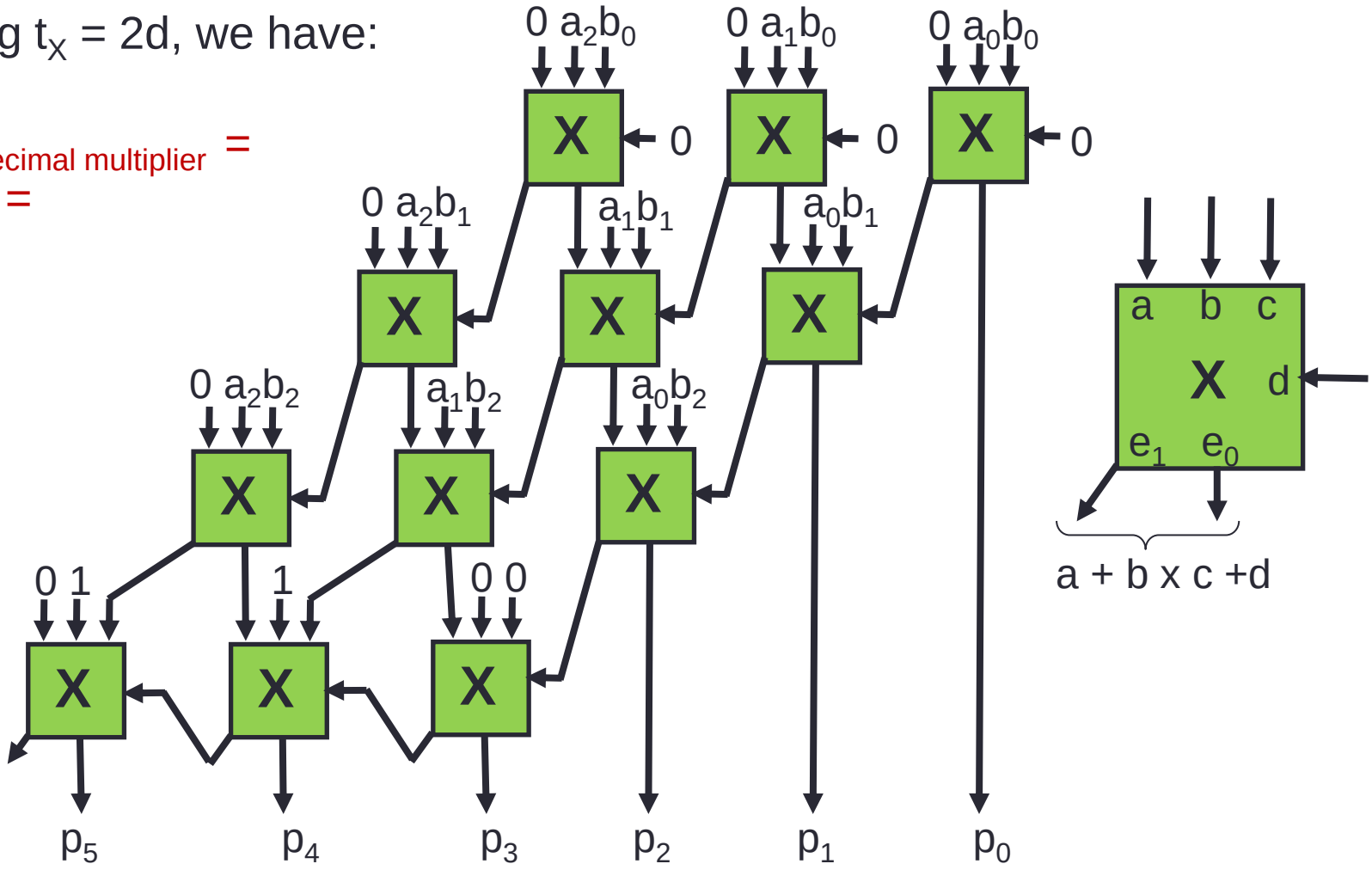
Decimal Arithmetic

Direct Method: Combinational Multiplier

Example: Design a 3x3 digit multiplier.

Assuming t_X = 2d, we have:

$T_{\text{nxn digit decimal multiplier}}$ = $2(n-1)\ t_X$ = $4(n-1)$ d

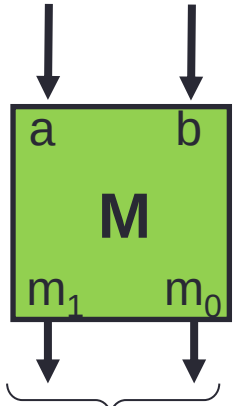


Decimal Arithmetic

Direct Method: Sequential Multiplier

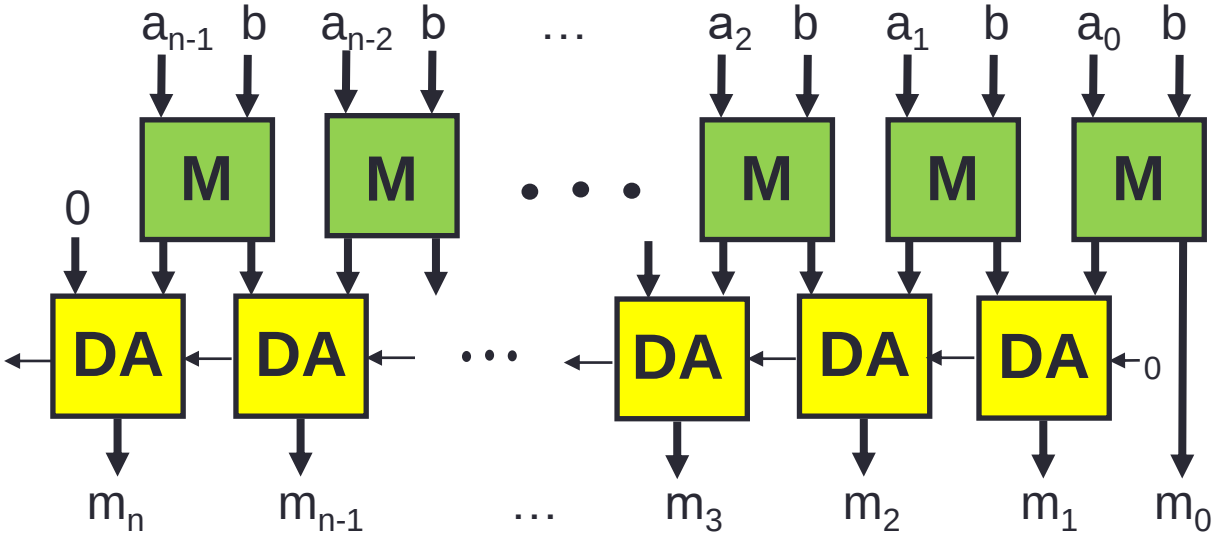
To implement Shift & Add method, we need to build and nx1 digit decimal multiplier.

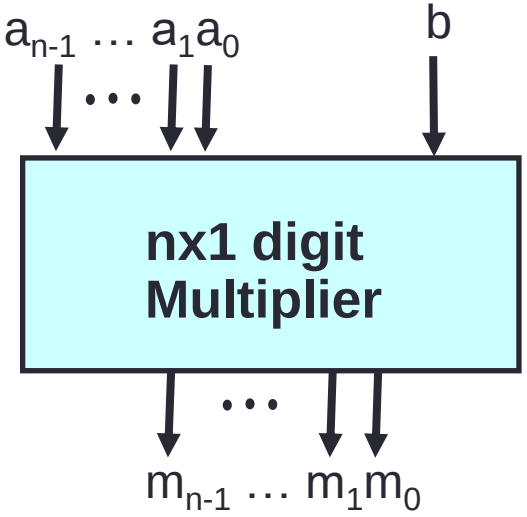
A one-digit decimal multiplier, M, can be built as a 2-layer combinational circuit with latency 2d.



$t_M = 2d$

Now, using M and DA cells we can design an nx1 digit decimal multiplier.





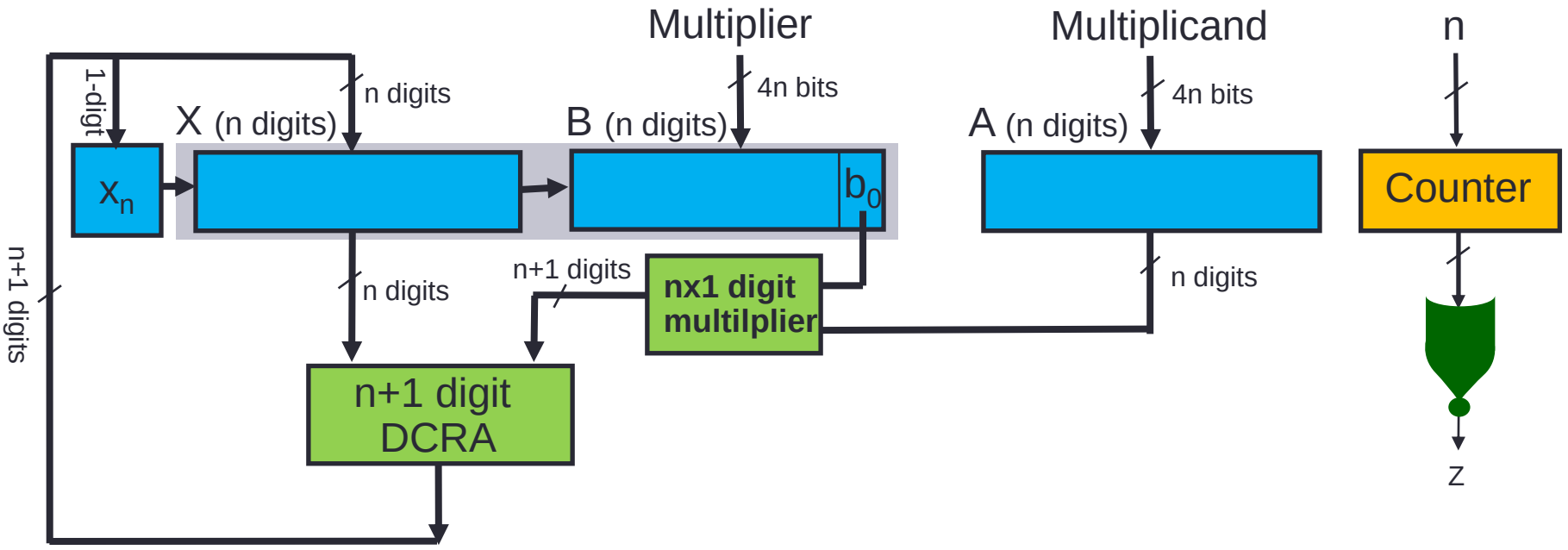
→ $T_{\text{nx1 digit multiplier}} = 2d + 10nd + 6d = 10nd + 8d$

Decimal Arithmetic

Direct Method: Sequential Multiplier

Add & Shift multiplier uses the following circuit (data path):

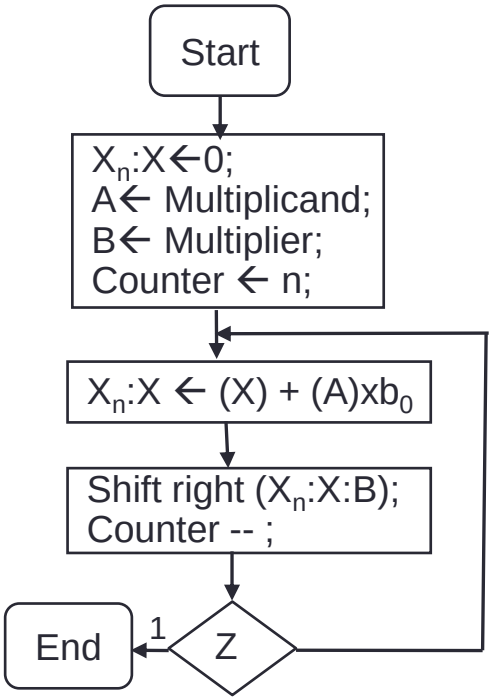
In the beginning: A and B keep the n-digit multiplicand and multiplier.
At the end: Register pairs X:B contain the 2n-digit multiply result.



Decimal Arithmetic

Direct Method: Sequential Multiplier

Add & Shift decimal multiplication algorithm is shown.



Example 1: Follow Add & Shift method step by step to multiply 3-digit numbers 123 and 117. (A=123)

X_n	X	B	b_0	Counter	Operation
0000	0000000000000	0001000101111		3	Add 7x123 to X
0000	1000011000001	0001000101111		3	Shift 1 digit
0000	0000100000110	0001000100001		2	Add 1x123 to X
0000	0010000001001	0001000100001		2	Shift 1 digit
0000	0000001000000	1001000100001		1	Add 1x123 to X
0000	0001010000011	1001000100001		1	Shift 1 digit
0000	0000000010100	0011100100001		0	End.

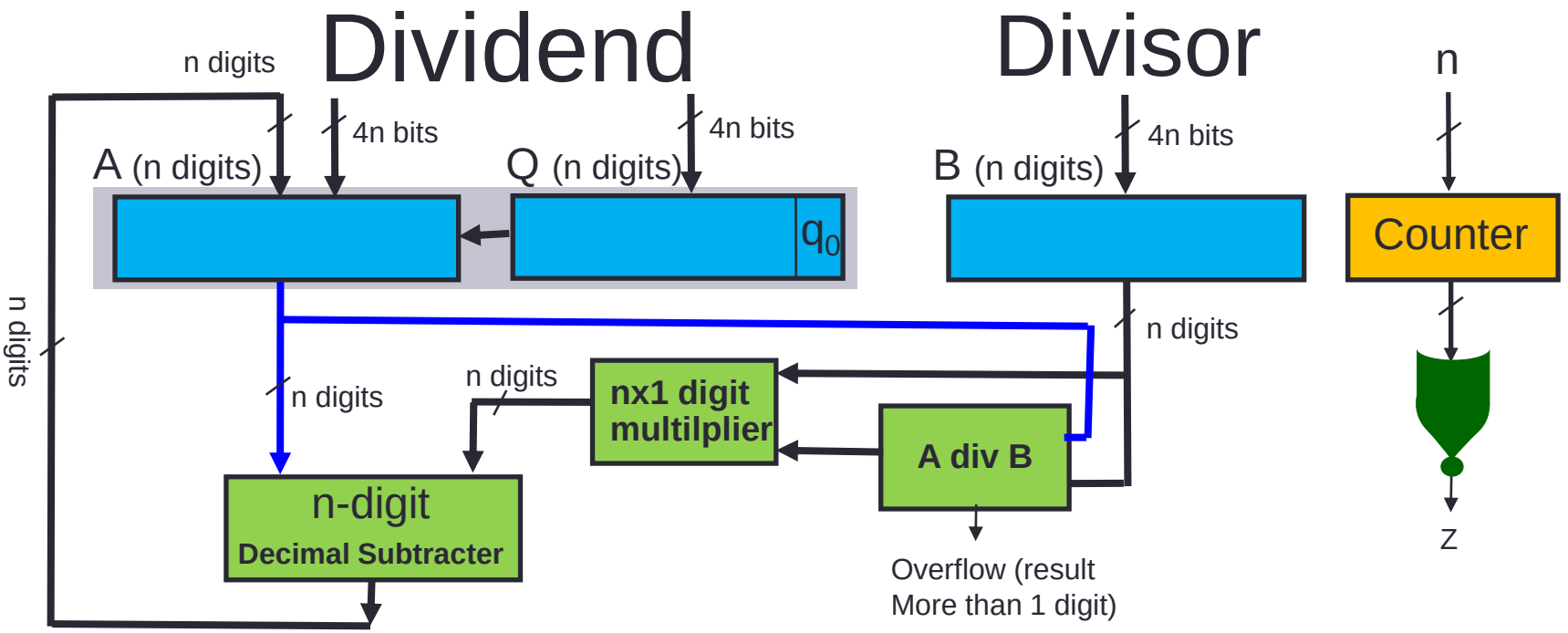
14391 = 123 x 117

Decimal Arithmetic

Direct Method: Sequential Divider

A sequential decimal divider uses the following circuit (data path):

In the beginning: A and Q keep the 2n-digit dividend and B keeps the n-digit divisor.
At the end: Register Q contain the n-digit quotient and A contains the n-digit remainder.

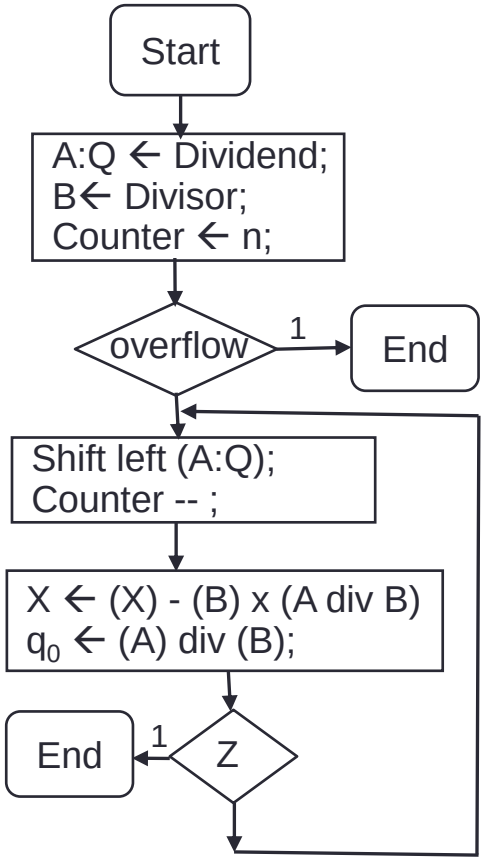


Decimal Arithmetic

Direct Method: Sequential Divider

The division algorithm is shown.

Example 2: Follow the division algorithm step by step to divide decimal 785 by decimal 11.



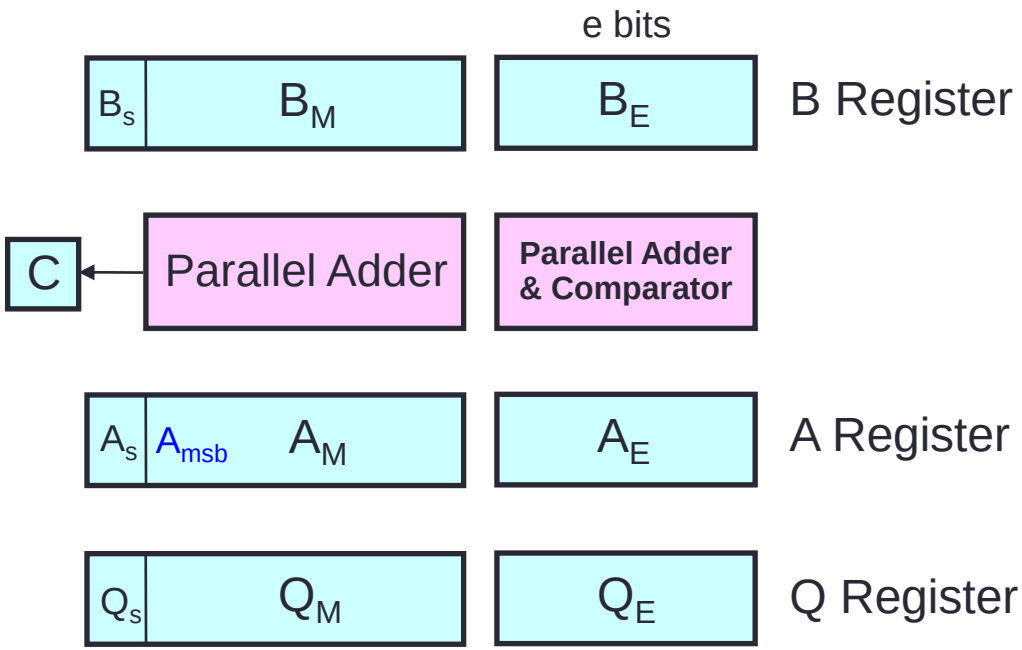
A	Q	q_0	Counter	Operation
00000111	10000101		2	No overflow
01111000	01010000		2	Shift 1 digit
01111000	01010000		1	X ← 78 – 7x11; q_0←7;
00000001	01010111		1	Shift 1 digit
00010101	01110000		0	X ← 15 – 1x11; q_0←1;
00000100	01110001		0	End.

remainder=4 quotient= 71

Floating-point Arithmetic

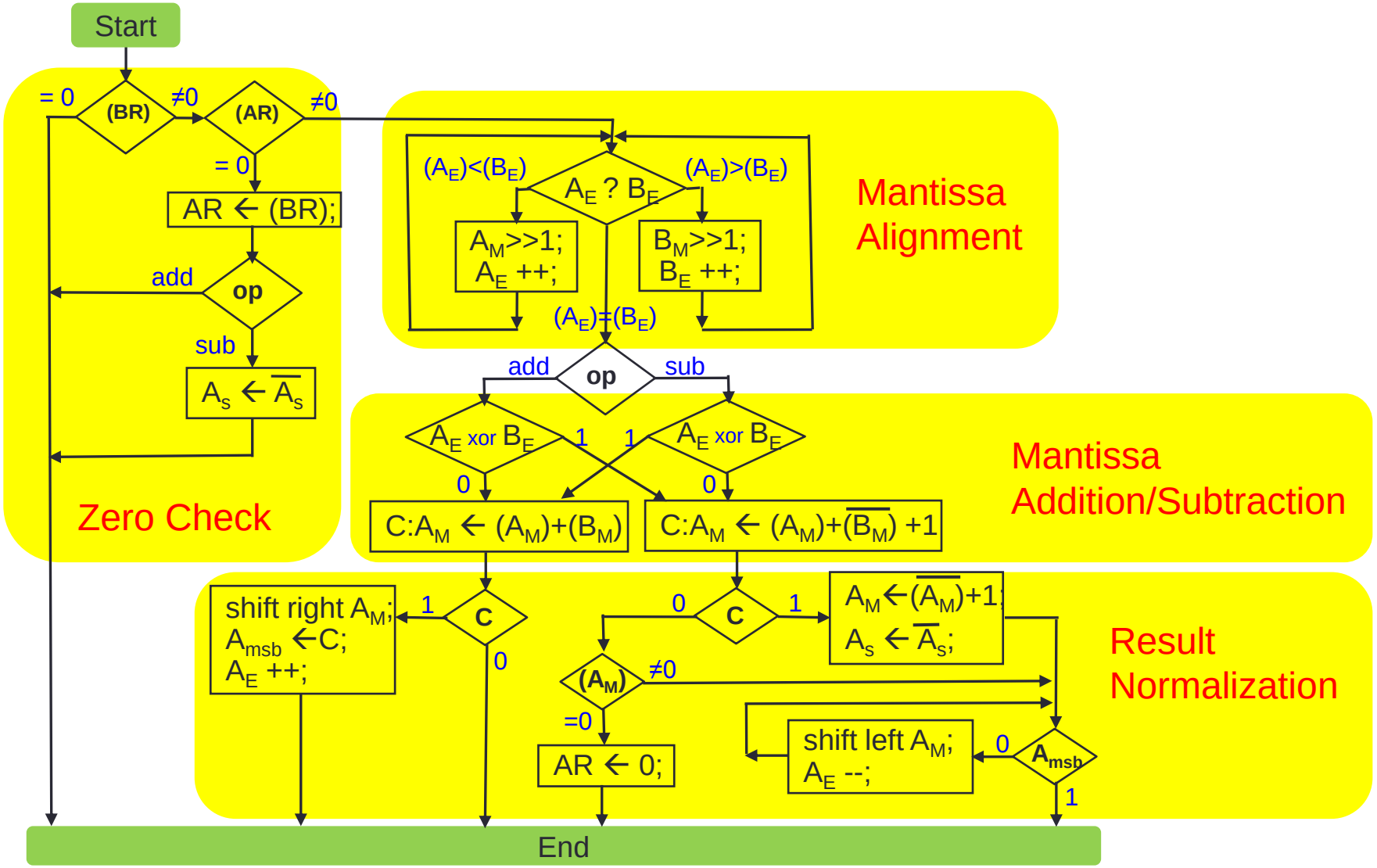
Consider the following hardware used to describe different arithmetic operations on floating-point numbers.

Each floating-point number has 2 parts:
- Mantissa (sign-magnitude representation) and
- Exponent (excess-2^{e-1} biased exponent).



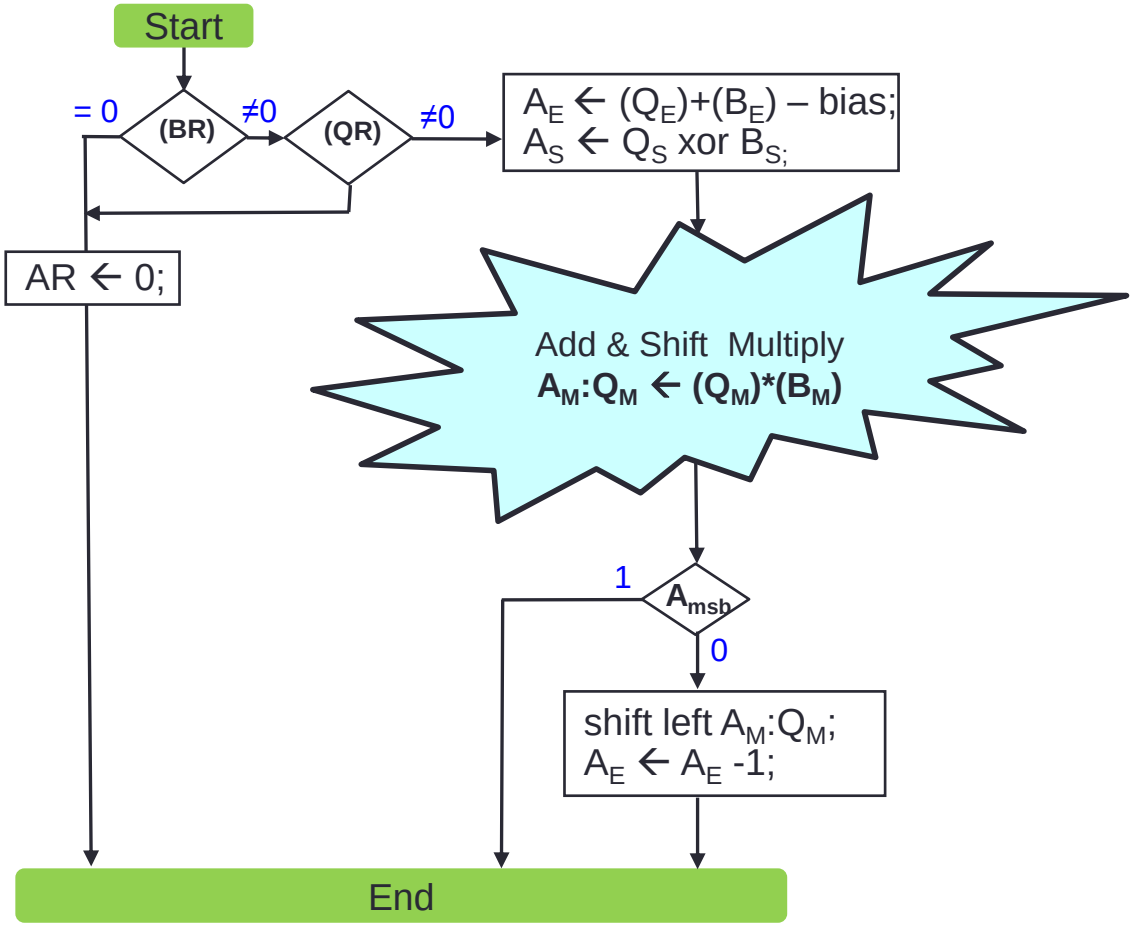
Floating-point Arithmetic

Floating-Point Addition/Subtraction: AR ← (AR) ± (BR)



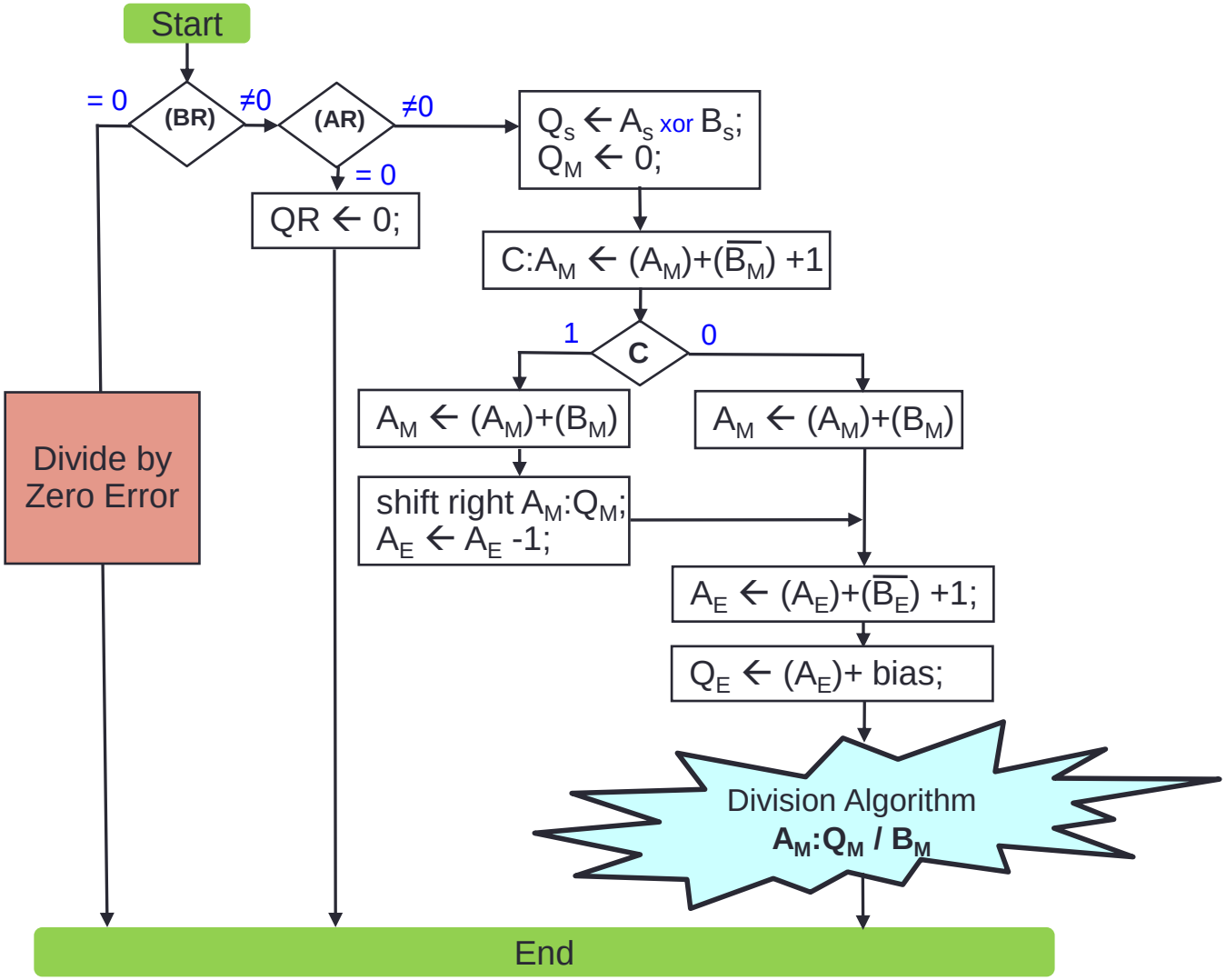
Floating-point Arithmetic

Floating-Point Multiplication: AR ← (QR) x (BR)



Floating-point Arithmetic

Floating-Point Division: QR ← (AR) / (BR)

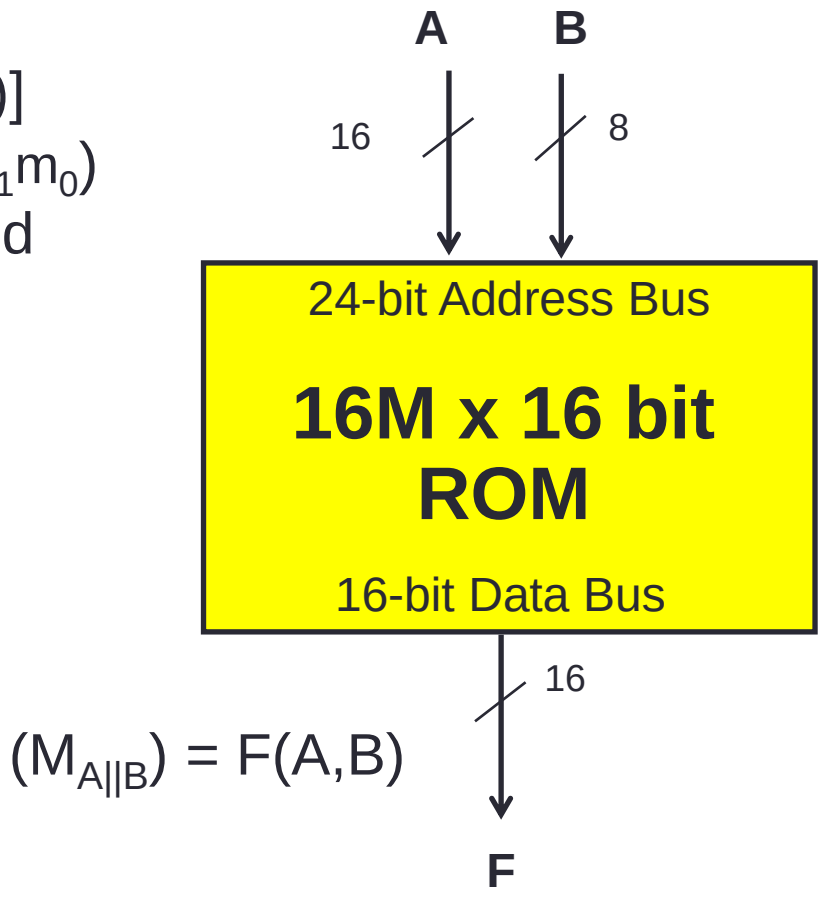


Special Purpose Arithmetic

When a very complex function is computed for few inputs, we can use ROMs instead of complex arithmetic circuits.

Example: Compute function
F (A,B) = A^B – sin(A) / [exp(B) – sqrt(A)]
for A (s:$e_5e_4e_3e_2e_1e_0$:$m_8m_7m_6m_5m_4m_3m_2m_1m_0$) being a 16-bit floating-point number and B ($b_7b_6b_5b_4.b_3b_2b_1b_0$) being an 8-bit fixed-point number.

Output is also a 16-bit floating-point number.



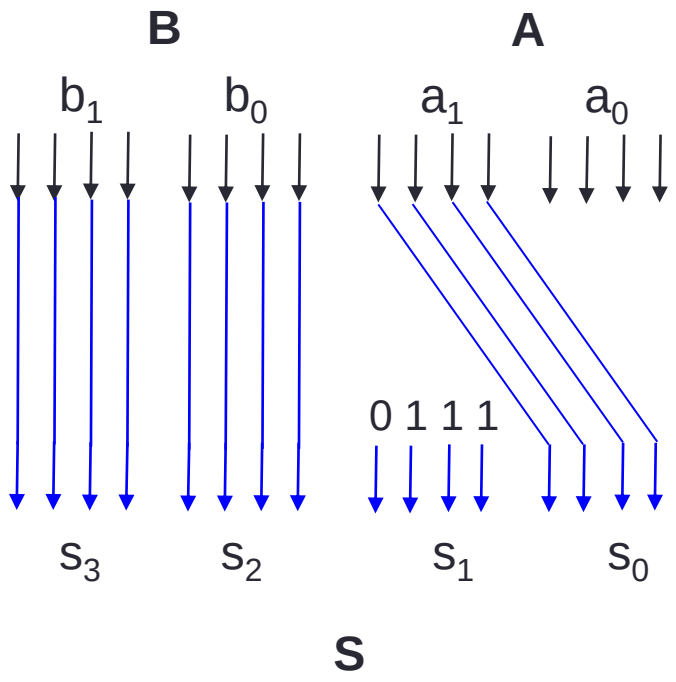
Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

Example 1: Design a circuit to calculate decimal output

$$S = 100 \times B + A/10 + 70$$

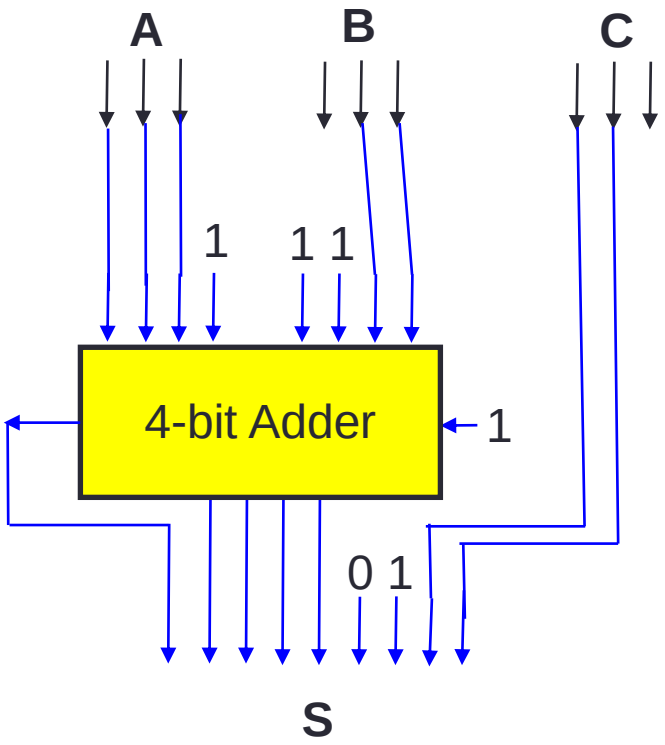
given 2-digit BCD numbers A and B.



Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

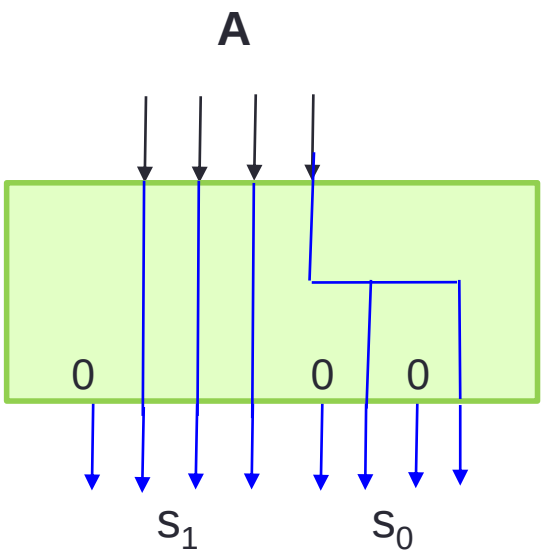
Example 2: Given 3-bit binary numbers A, B and C, design a circuit to compute binary output S = 16 (2A + B%4 + 14) + C/2 + 4 using a 4-bit binary adder.



Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

Example 3: Design a circuit to calculate decimal output S = 5 x A given 1-digit BCD number A.



S = 5 x A

A	S
0000	0000 0000
0001	0000 0101
0010	0001 0000
0011	0001 0101
0100	0010 0000
0101	0010 0101
0110	0011 0000
0111	0011 0101
1000	0100 0000
1001	0100 0101

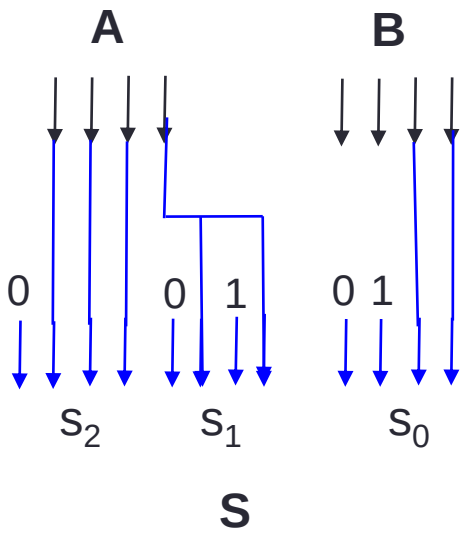
Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

Example 4: Design a circuit to calculate decimal output

$$S = 50 \times A + B\%4 + 24$$

given 1-digit BCD numbers A and B.



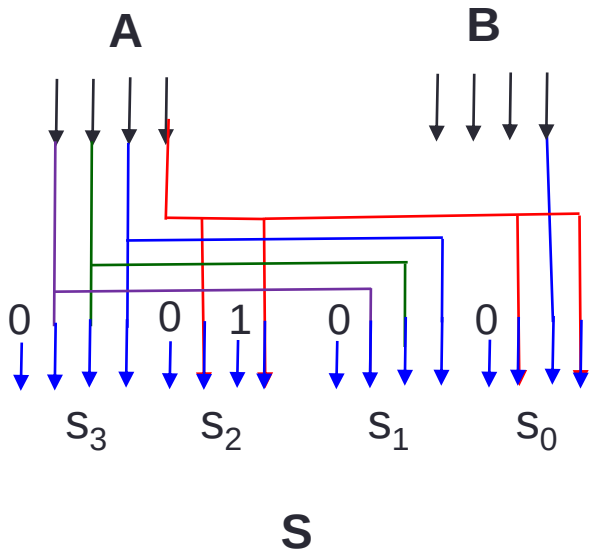
Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

Example 5: Design a circuit to calculate 4-digit decimal output

$$S = 505 \times A + 2 \times (B\%2) + 200$$

given 1-digit BCD numbers A and B.



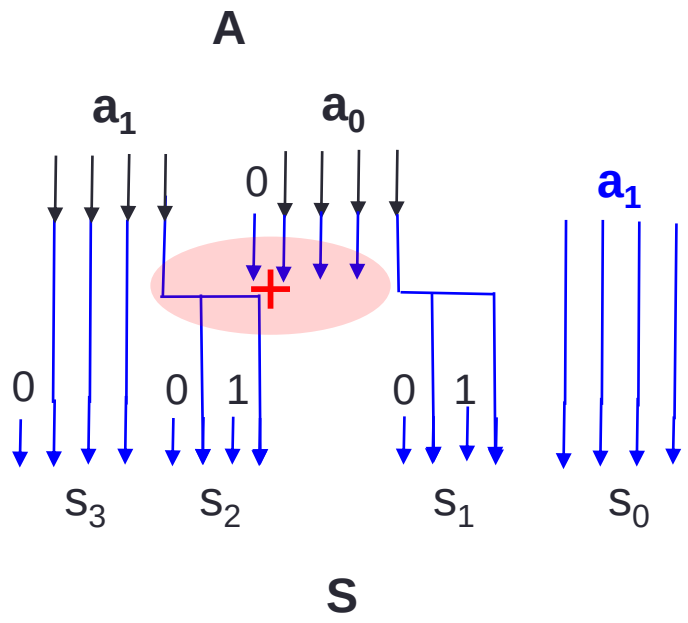
Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

Example 6: Design a circuit to calculate decimal output

$$S = 50 \times A + A/10 - 100 \times [(A\%10)/2] + 220$$

given a 2-digit BCD number A.



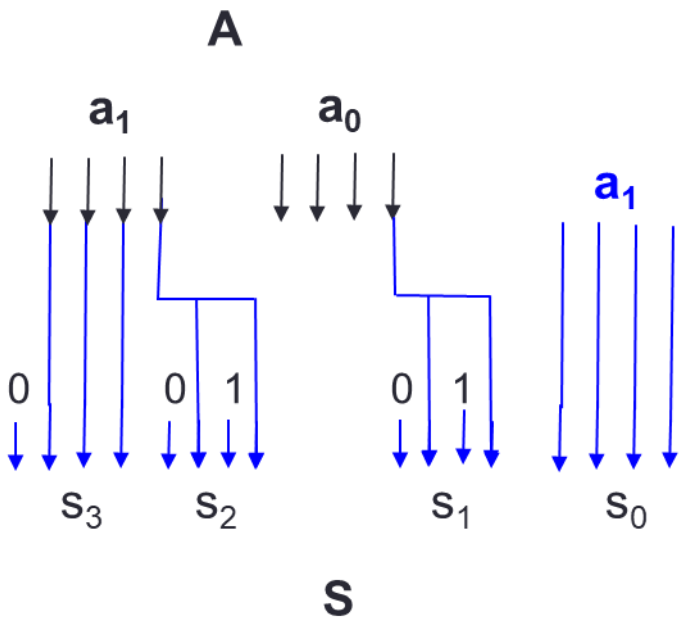
Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

Example 6: Design a circuit to calculate decimal output

$$S = 50 \times A + A/10 - 100 \times [(A\%10)/2] + 220$$

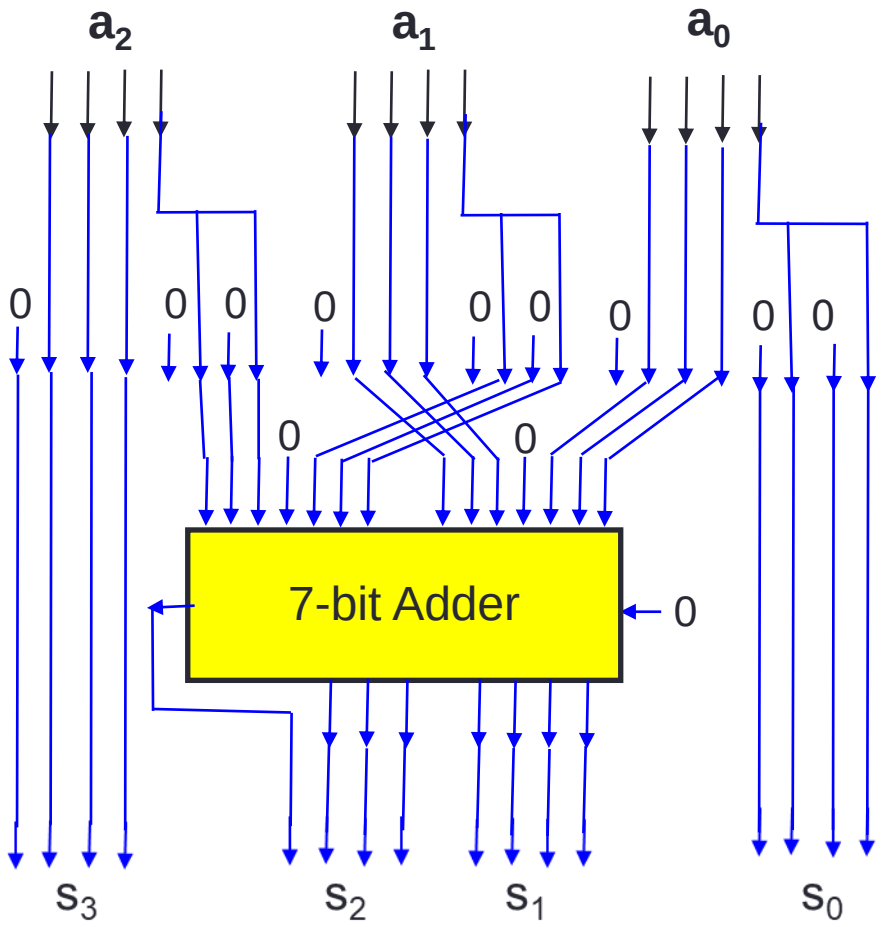
given a 2-digit BCD number A.



Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

Example 7: Design a circuit to calculate decimal output 5 x A given a 3-digit BCD number A.

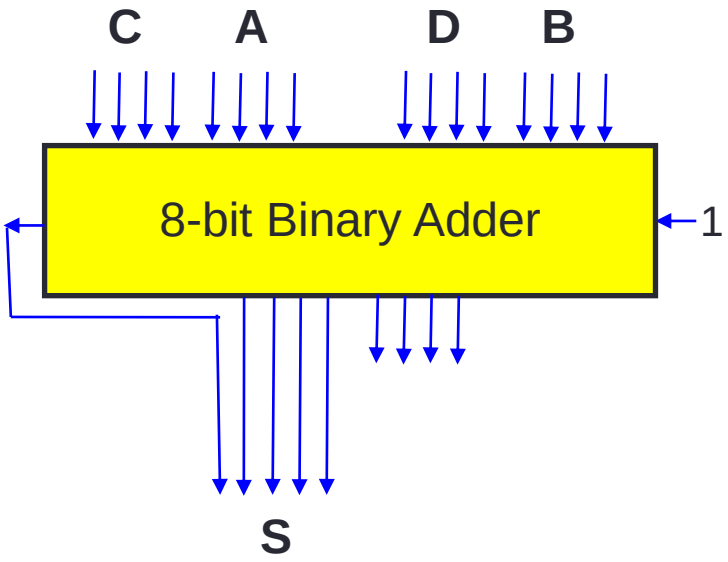


Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

Example 8: Using an 8-bit binary adder and given 4-bit pure binary numbers A, B, C, and D, design a circuit to generate S:

If (A+B)>14 then S = C+D+1 else S = C+D;

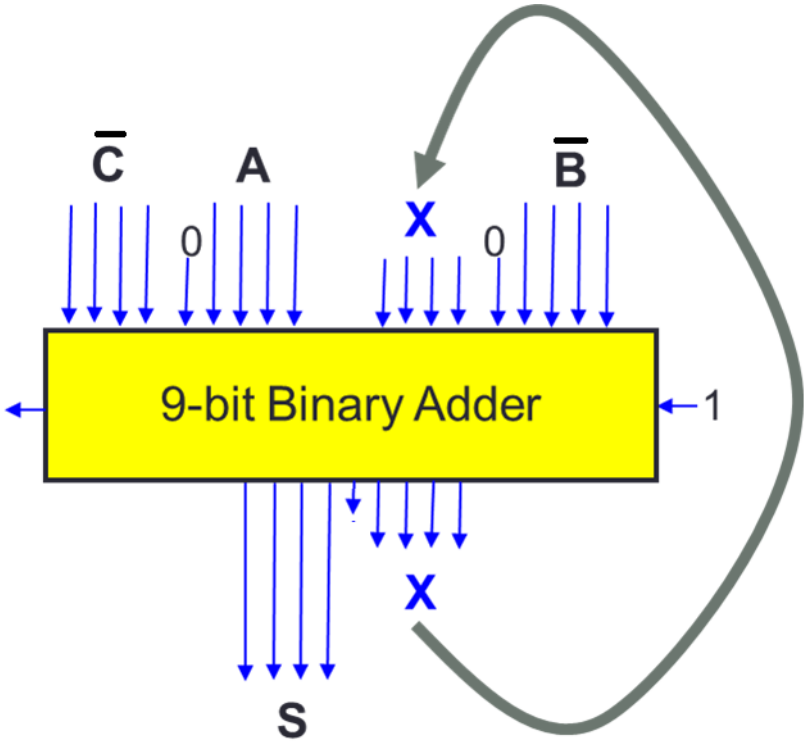


Special Purpose Arithmetic

When inputs are of special formats/sizes or fixed, we might design more efficient and simpler circuits.

Example 9: Using a 9-bit binary adder and given 4-bit 2's complement numbers A, B, and C, design a circuit to generate S:

$$S = A - B - C - 1$$



END OF SLIDES

QUESTIONS?