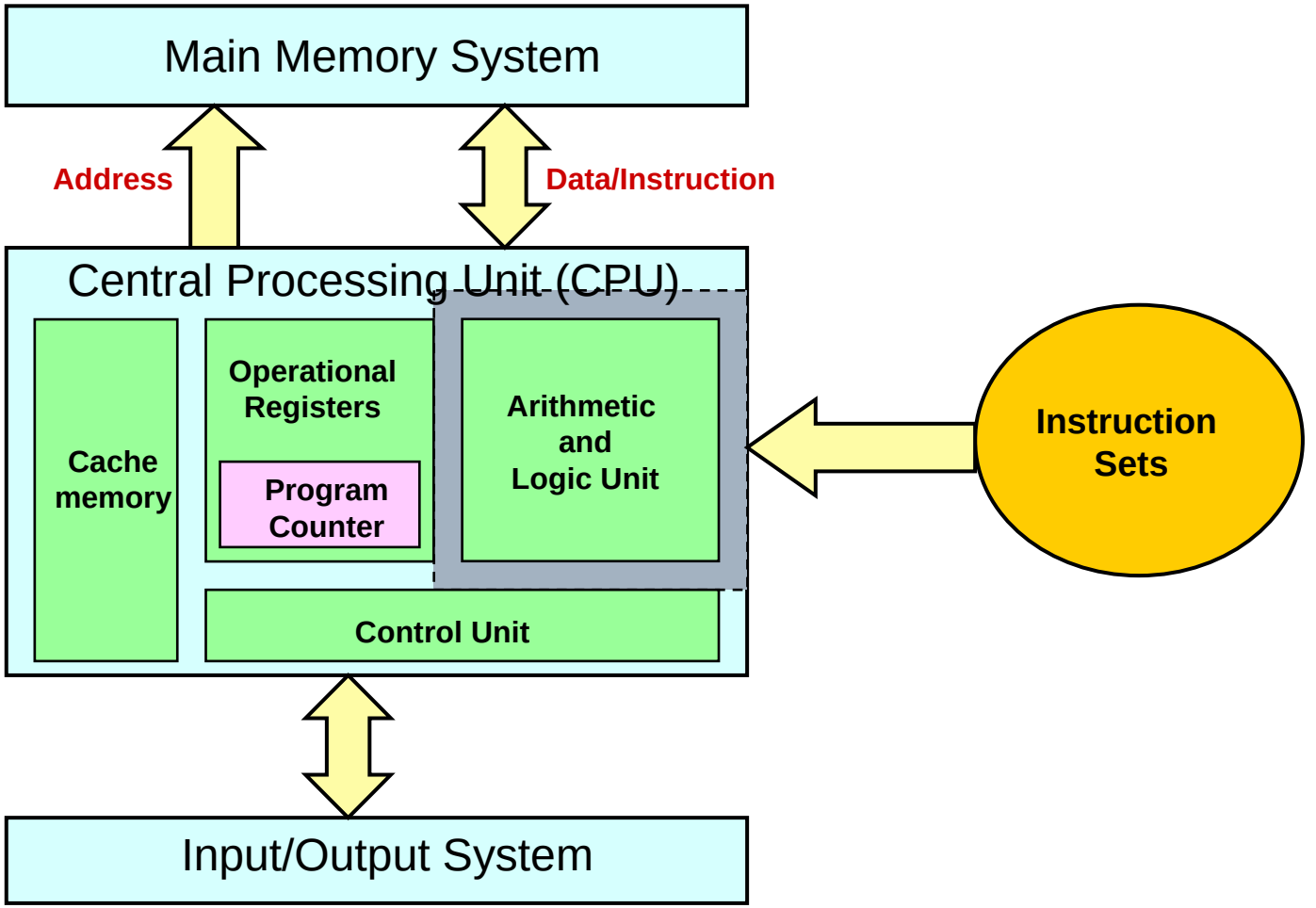
Chapter 7
Arithmetic Logic

Jin-Fu Li
Department of Electrical Engineering
National Central University
Jungli, Taiwan

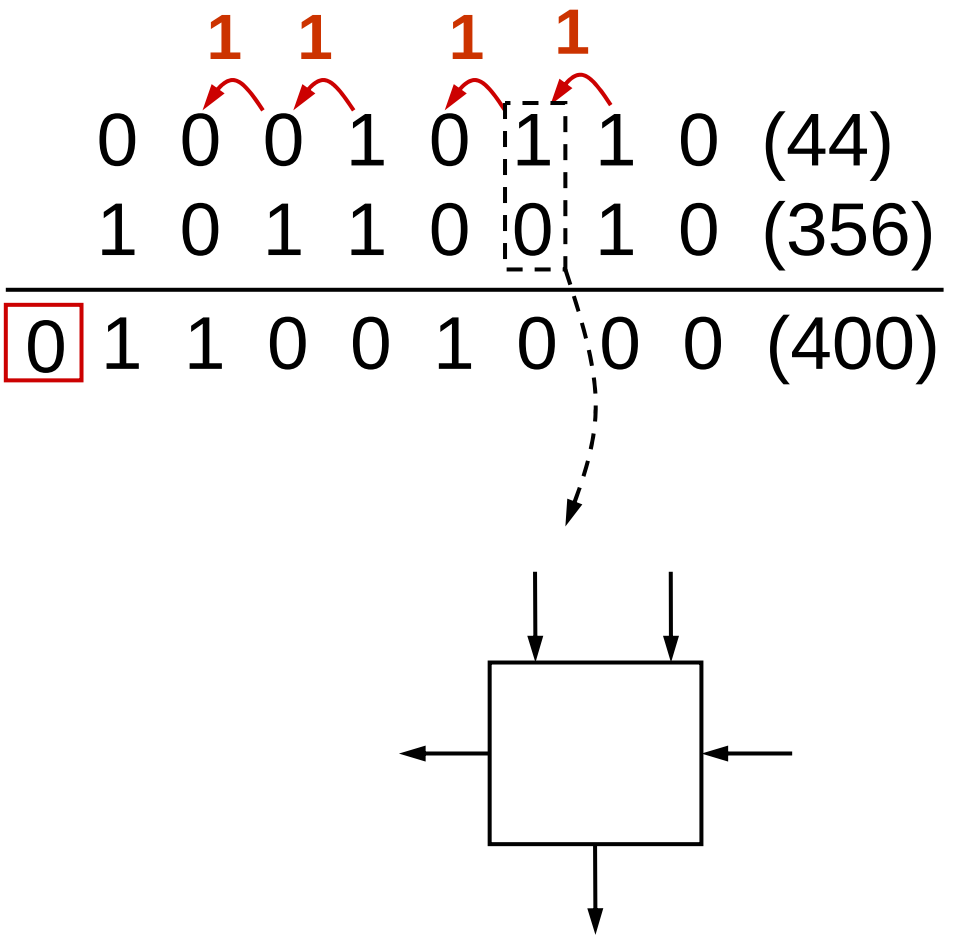
Outline

- Addition and Subtraction of Signed Numbers
- Multiplication of Positive Numbers

Content Coverage



An Example of Binary Addition

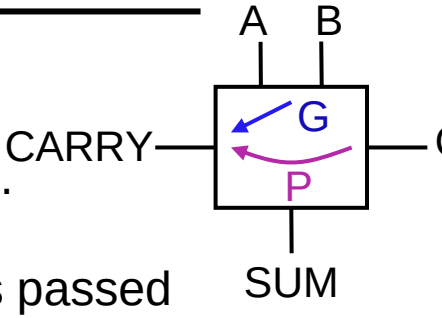


1-bit Full Adder

Adder Truth Table

C	A	B	A.B(G)	A+B(P)	A⊕B	SUM	CARRY
0	0	0	0	0	0	0	0
0	0	1	0	1	1	1	0
0	1	0	0	1	1	1	0
0	1	1	1	1	0	0	1
1	0	0	0	0	0	1	0
1	0	1	0	1	1	0	1
1	1	0	0	1	1	0	1
1	1	1	1	1	0	1	1

Generate Signal G(A.B): occurs when a carry output (CARRY) is internally generated within the adder .



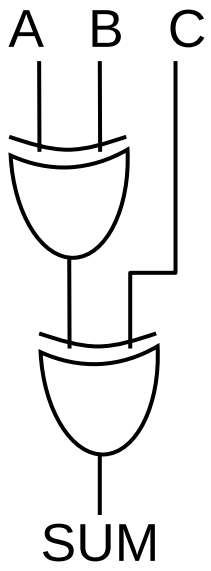
Propagate Signal P(A+B): when it is true, the carry in signal C is passed to the carry output (CARRY) when C is true

Logic for a 1-bit Full Adder

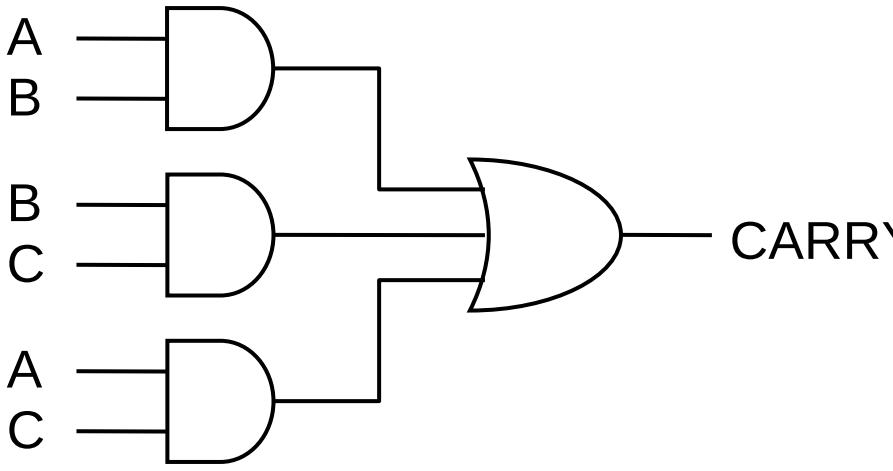
SUM=A⊕ B⊕ C

CARRY=AB+AC+BC

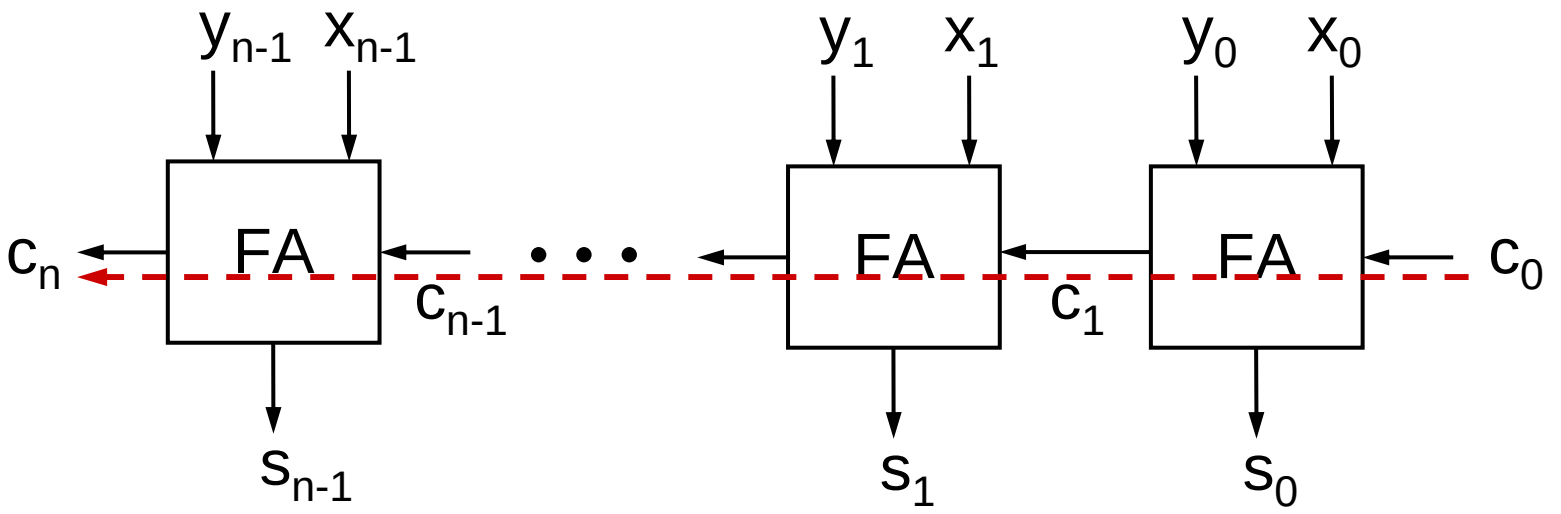
Single-bit schematic of SUM



Single-bit schematic of CARRY



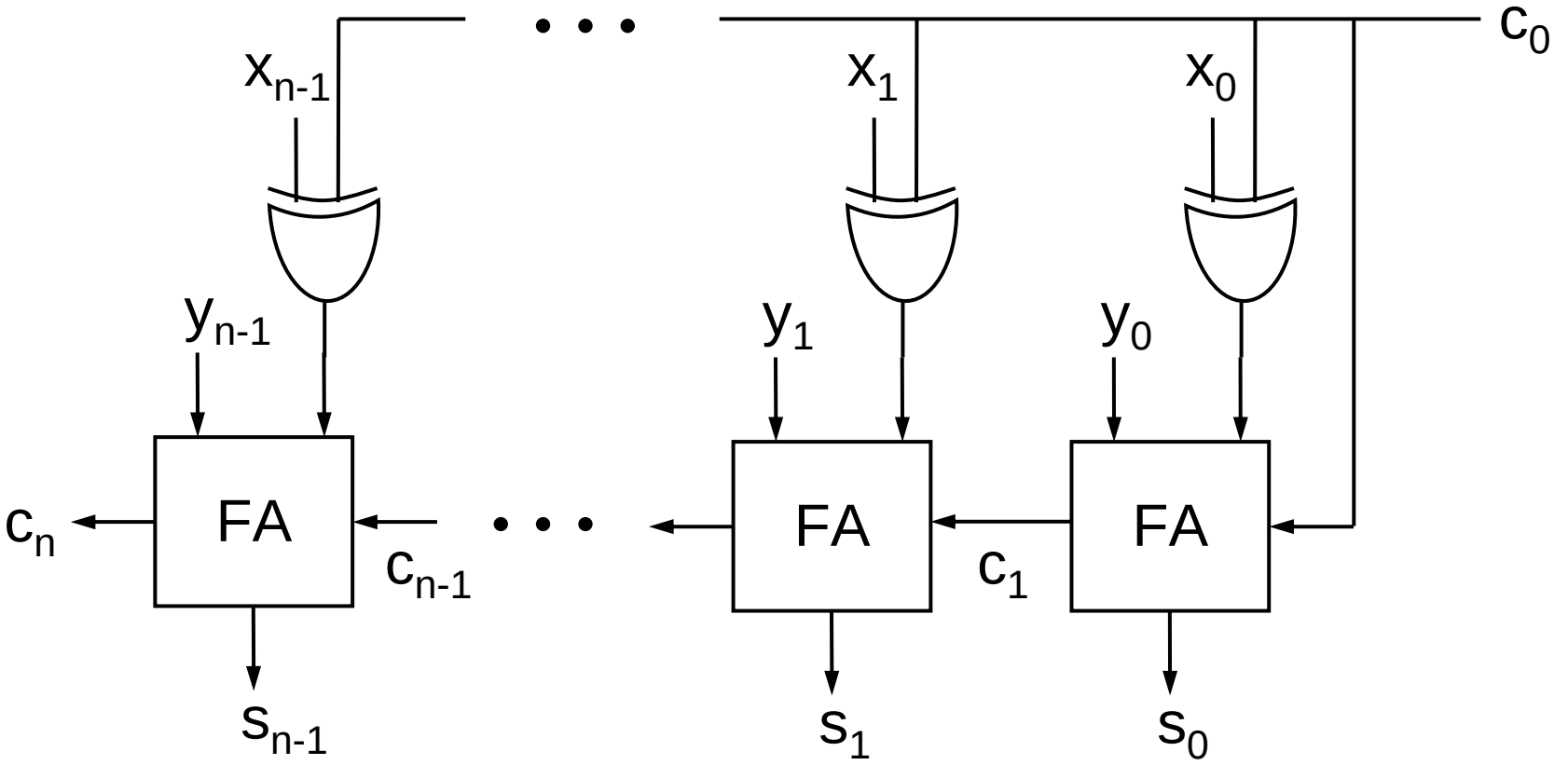
An N-bit Ripple Carry Adder



Disadvantage: long delay time

Binary Addition-Subtraction

Y-X=Y+X'+1



Carry-LookAhead Addition

$C_{i+1}=A_iB_i+(A_i+B_i)C_i=G_i+P_iC_i$

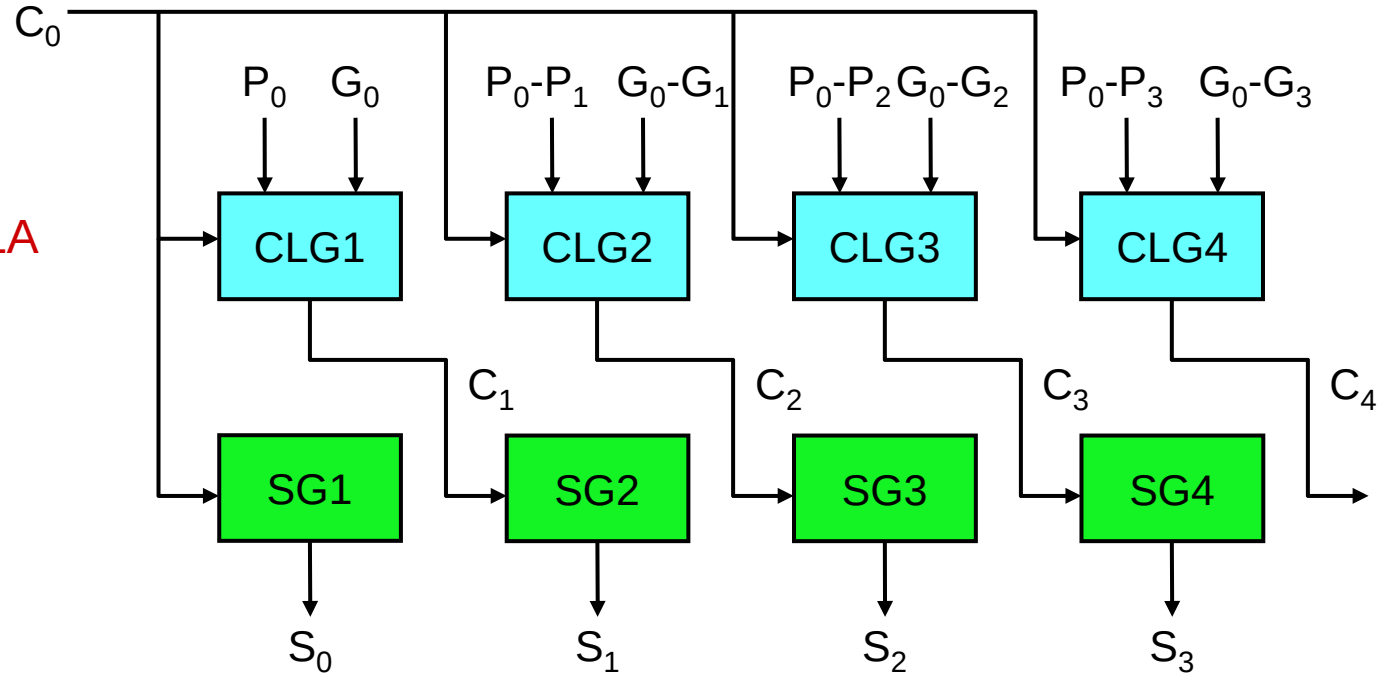
$C_1=G_0+P_0C_0$

$C_2=G_1+P_1G_0+P_1P_0C_0$

$C_3=G_2+P_2G_1+P_2P_1G_0+P_2P_1P_0C_0$

$C_4=G_3+P_3G_2+P_3P_2G_1+P_3P_2P_1G_0+P_3P_2P_1P_0C_0$

4-bit CLA



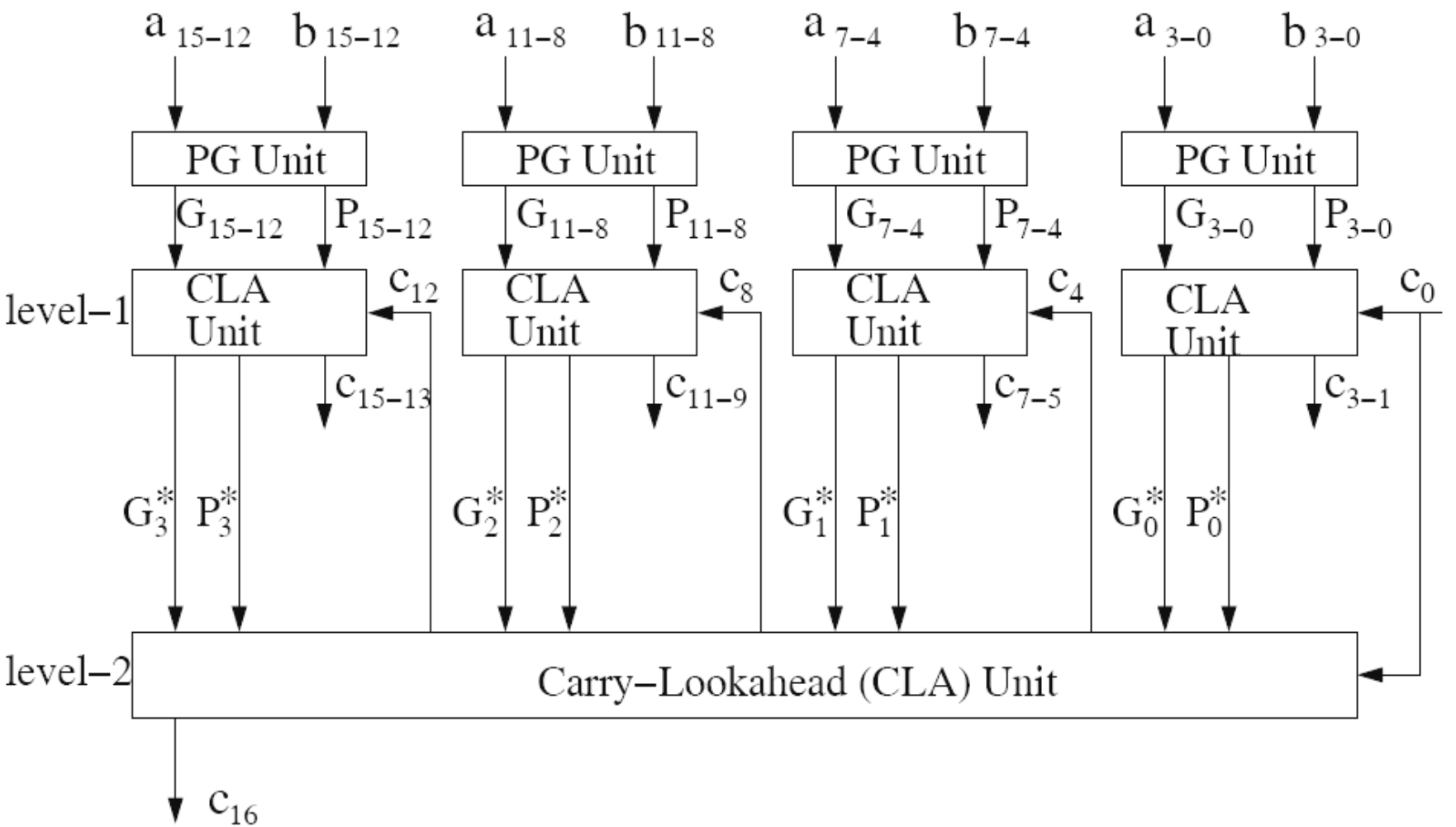
Multi-Level Carry LookAhead Addition

$$
\begin{aligned}
c_1 &= G_0 + P_0 c_0 \\
c_2 &= G_1 + G_0 P_1 + P_1 P_0 c_0 \\
&\vdots \\
c_n &= G_n + P_n G_{n-1} + P_n P_{n-1} G_{n-2} + \cdots \\
&\quad + P_n P_{n-1} \ldots P_0 c_0 .
\end{aligned}
$$

$$
\begin{aligned}
G_0^* &= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 , \\
P_0^* &= P_0 P_1 P_2 P_3 .
\end{aligned}
$$

$$
\begin{aligned}
c_4 &= G_0^* + c_0 P_0^* , \\
c_8 &= G_1^* + G_0^* P_1^* + c_0 P_0^* P_1^* , \\
c_{12} &= G_2^* + G_1^* P_2^* + G_0^* P_1^* P_2^* + c_0 P_0^* P_1^* P_2^* , \\
c_{16} &= G_3^* + G_2^* P_3^* + G_1^* P_2^* P_3^* + G_0^* P_1^* P_2^* P_3^* \\
&\quad + c_0 P_0^* P_1^* P_2^* P_3^* .
\end{aligned}
$$

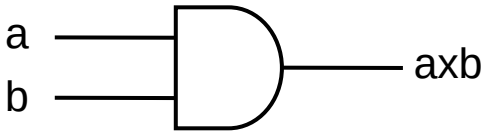
Multi-Level Carry LookAhead Addition



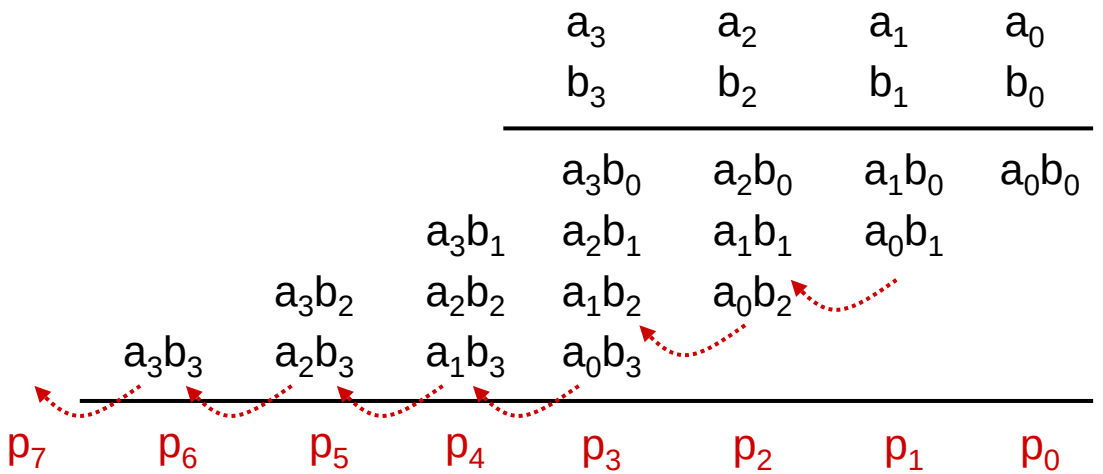
Multiplication of Positive Numbers

Bit-level multiplier

a	b	axb
0	0	0
0	1	0
1	0	0
1	1	1



Multiplication of two 4-bit words



Array Multiplication

- Consider two unsigned binary integers X and Y

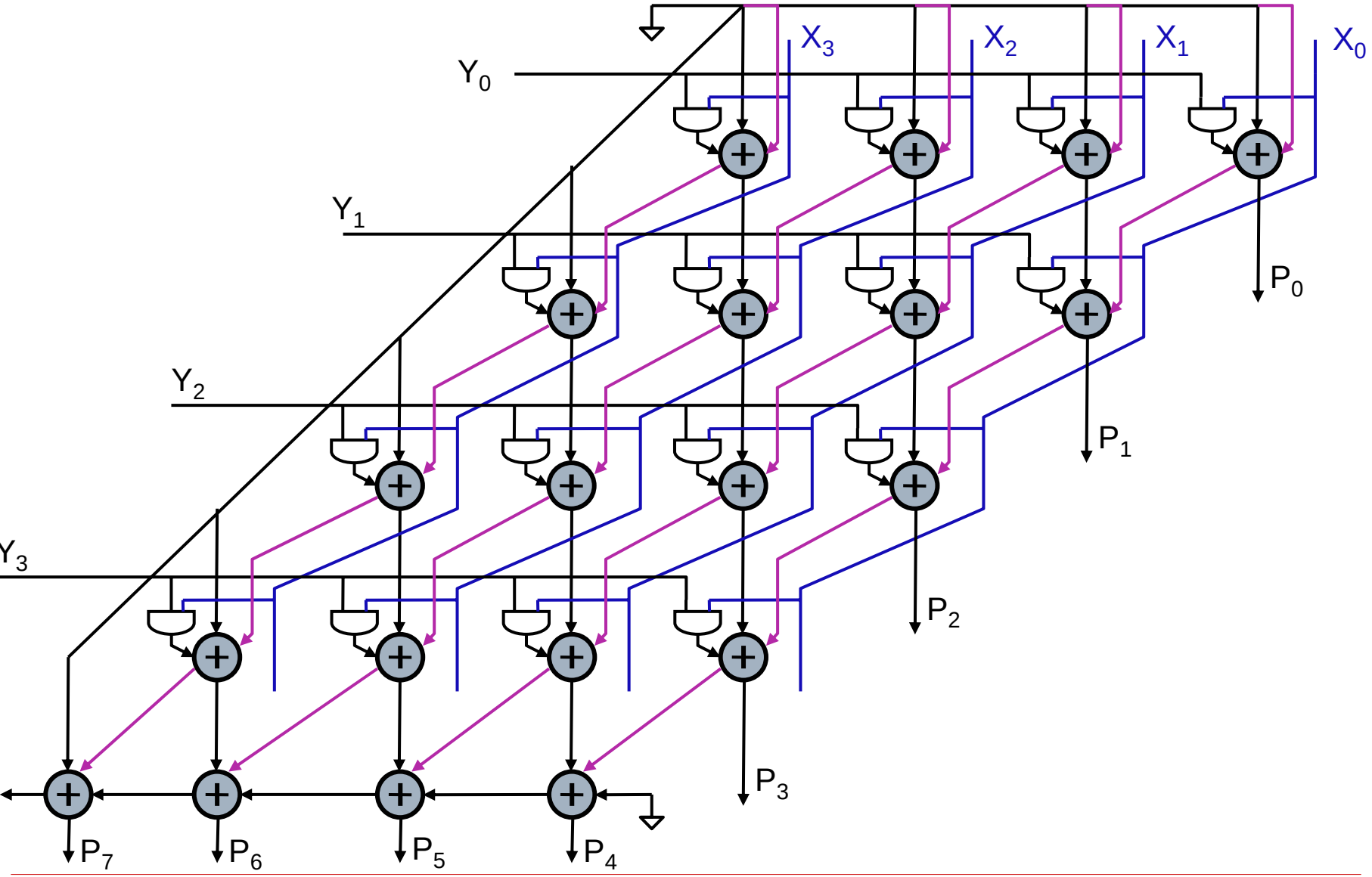
$$X = \sum_{i=0}^{n-1} X_i 2^i \qquad Y = \sum_{j=0}^{n-1} Y_j 2^j$$

$$P = X \times Y = \sum_{i=0}^{n-1} X_i 2^i \cdot \sum_{j=0}^{n-1} Y_j 2^j$$

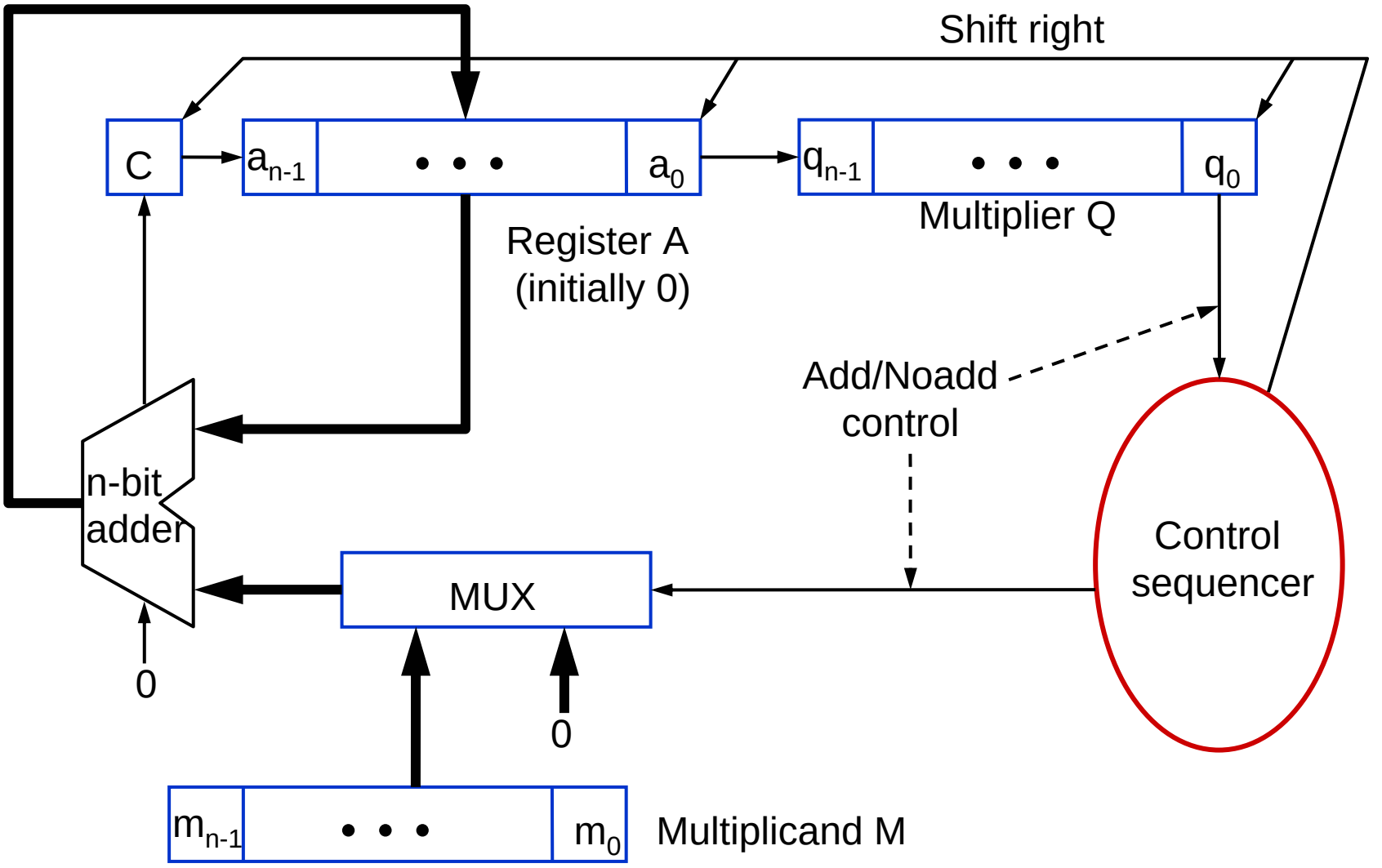
$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (X_i Y_j) 2^{i+j}$$

$$= \sum_{k=0}^{n+n-1} P_k 2^k$$

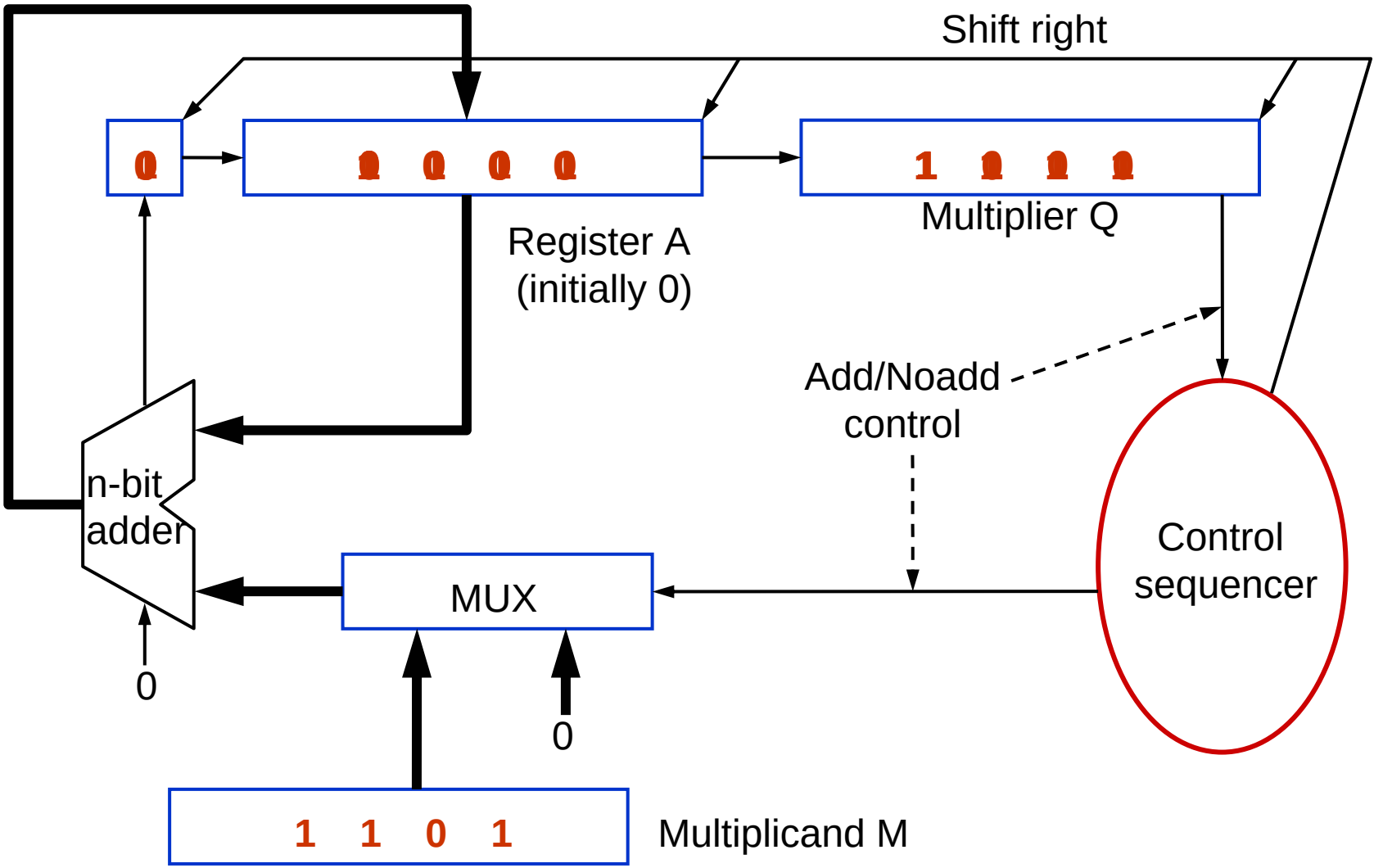
Array Multiplication



Register-Based Multiplication



Example of Register-Based Multiplication

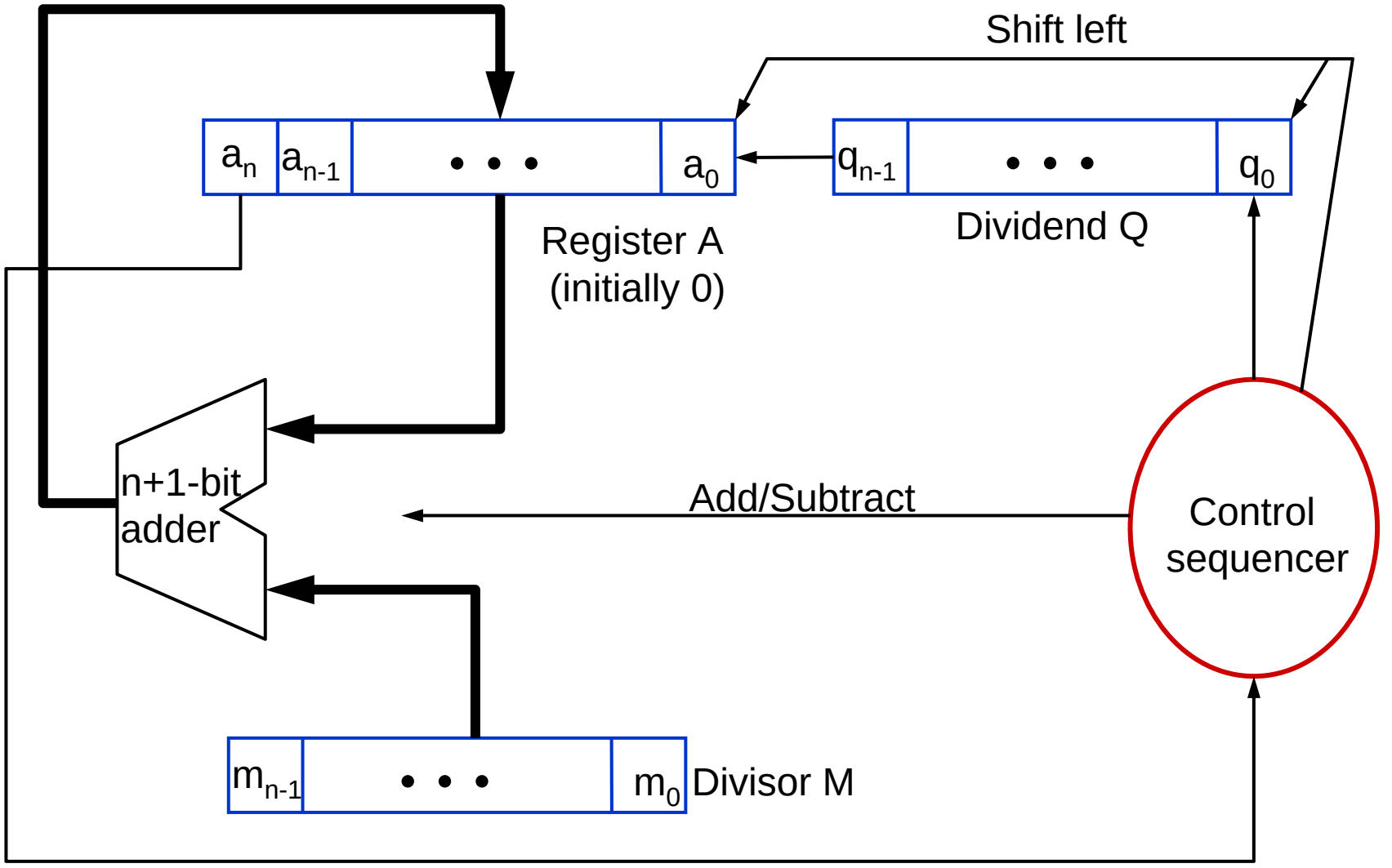


Integer Division

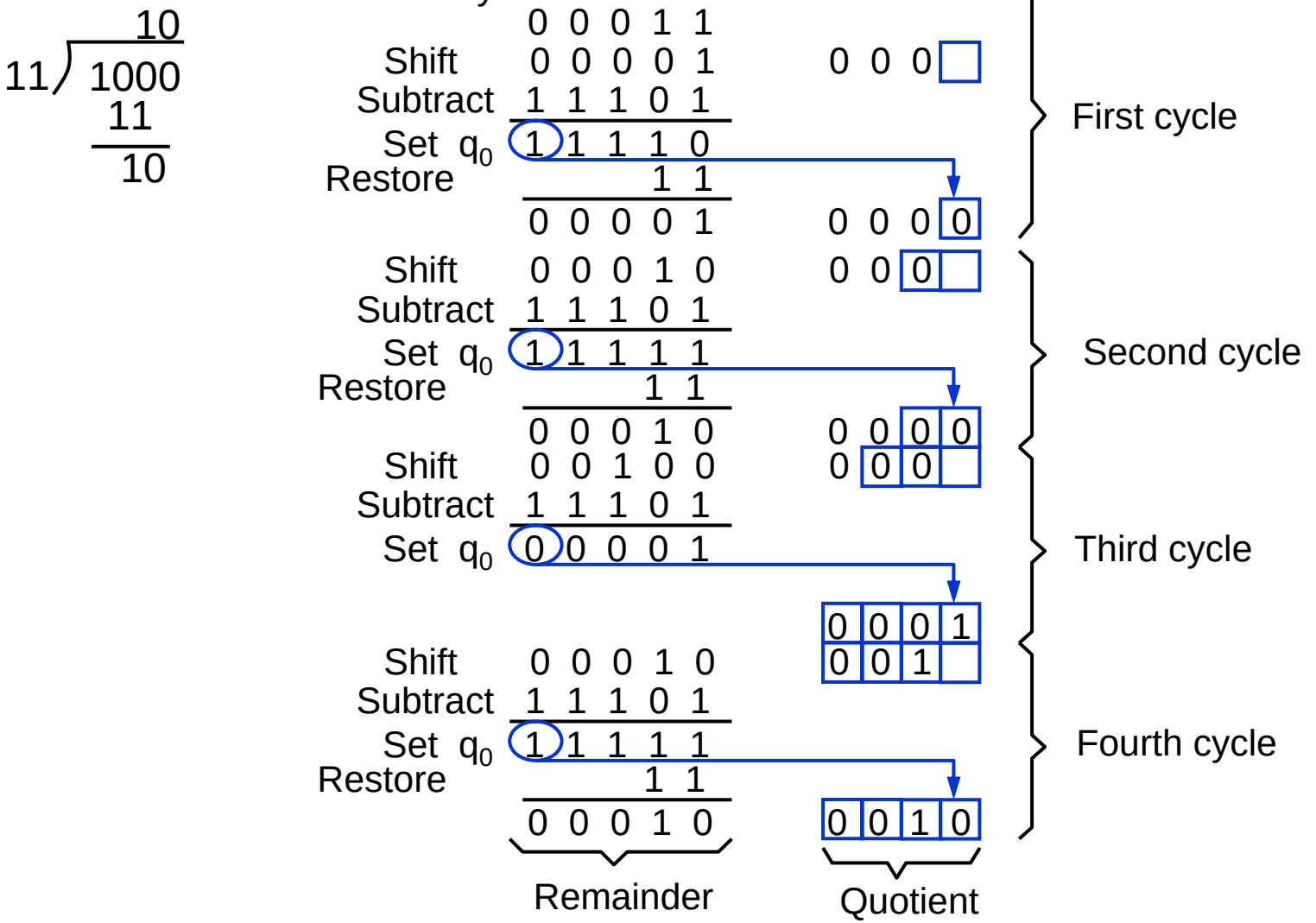
```
      21
   ------
13 ) 274
     26
   ----
     14
     13
   ----
      1
```

```
          10101
      ----------
1101 ) 100010010
       1101
       ----
        10000
        1101
        ----
         1110
         1101
         ----
            1
```

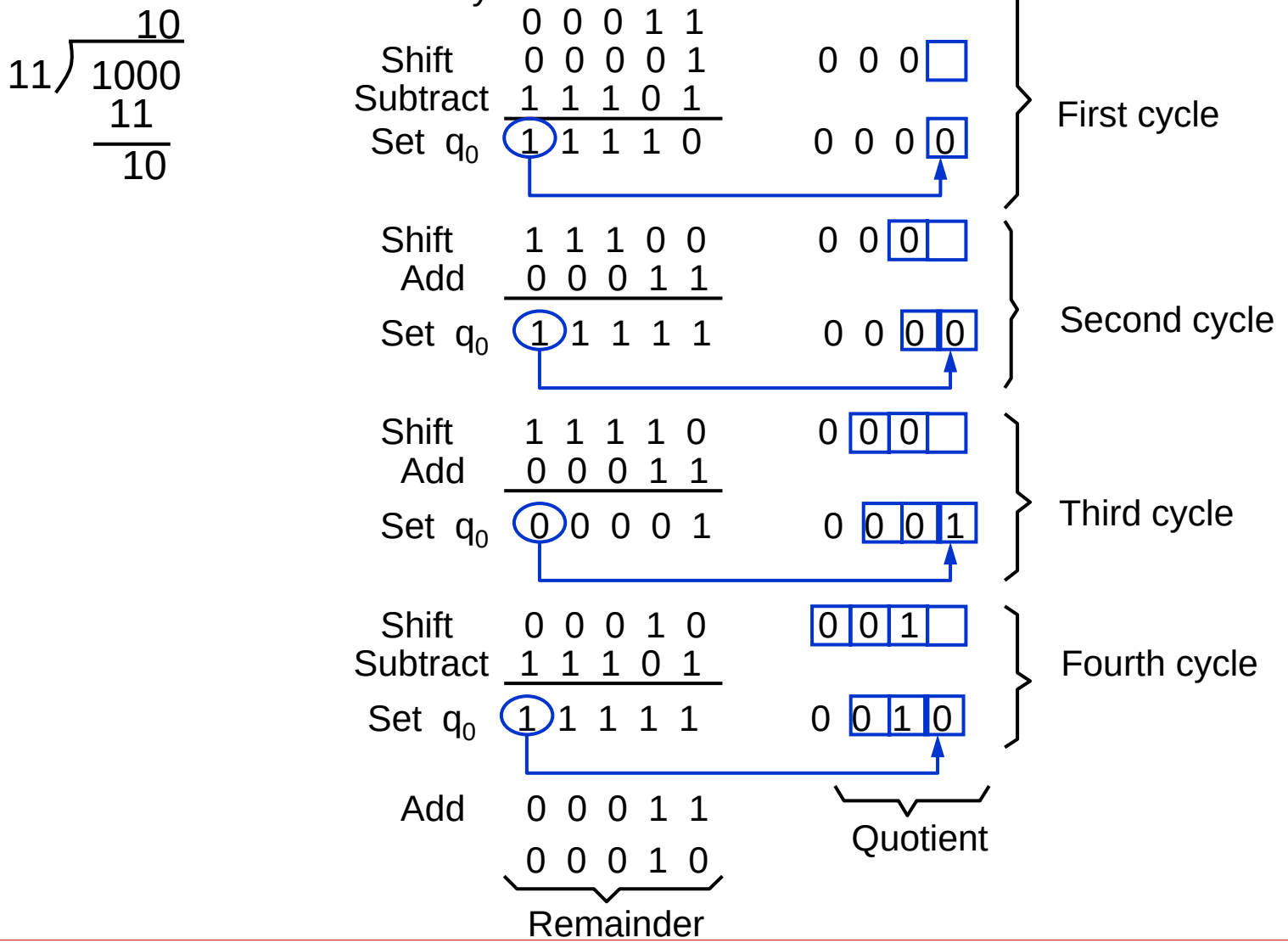
Register-Based Division



A Restoring-Division Example



A Nonrestoring-Division Example



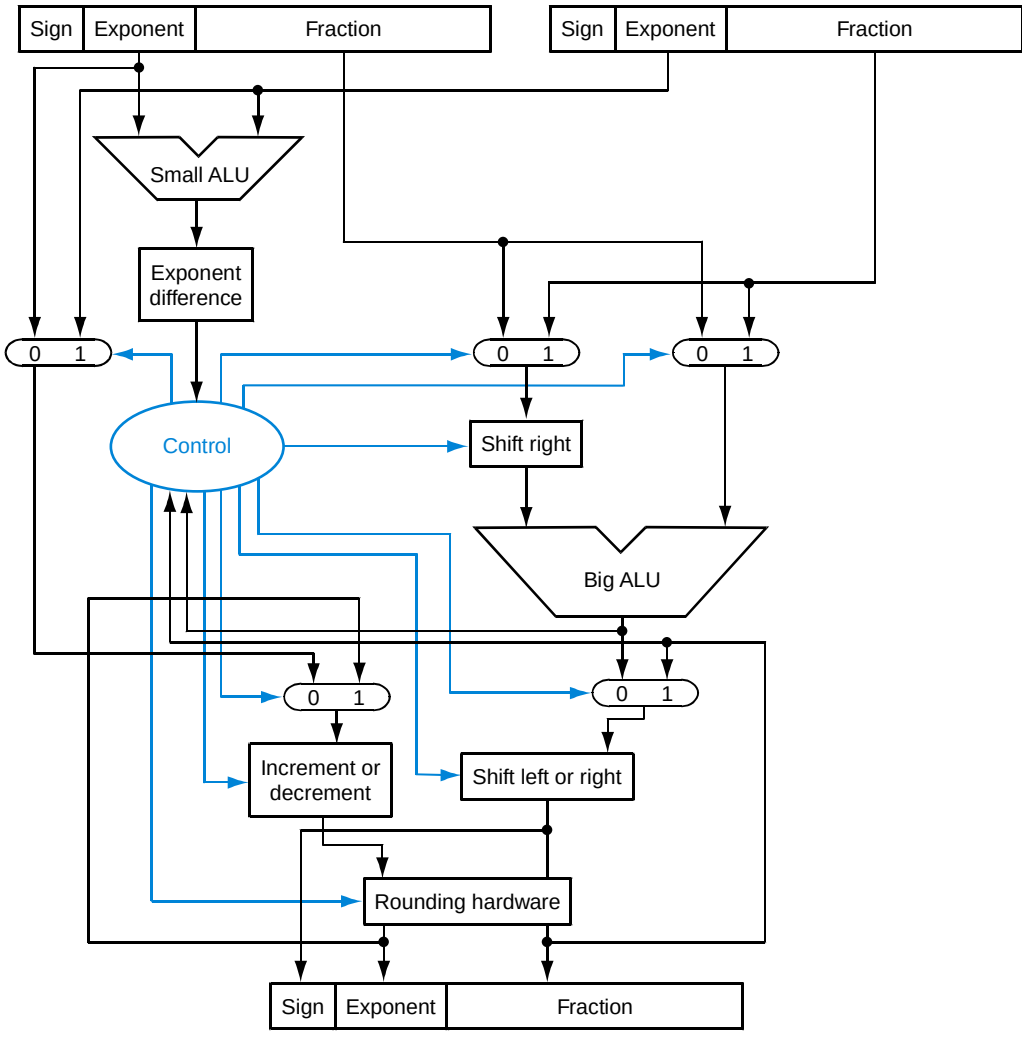
Floating Point (a brief look)

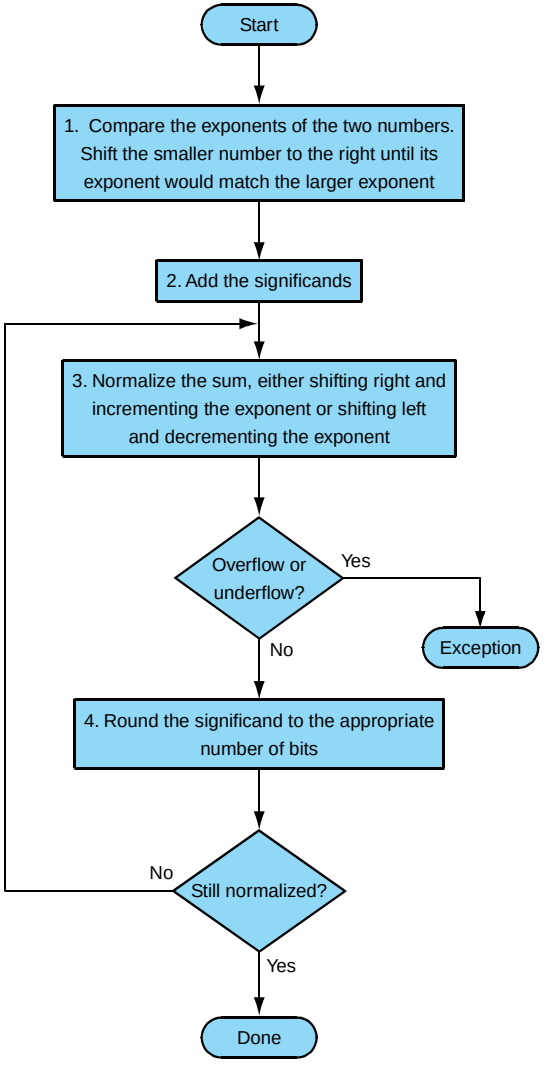
- We need a way to represent
  - numbers with fractions, e.g., 3.1416
  - very small numbers, e.g., .000000001
  - very large numbers, e.g., 3.15576×10^9
- Representation:
  - sign, exponent, significand: $(-1)^{sign} \times significand \times 2^{exponent}$
  - more bits for significand gives more accuracy
  - more bits for exponent increases range
- IEEE 754 floating point standard:
  - single precision: 8 bit exponent, 23 bit significand
  - double precision: 11 bit exponent, 52 bit significand

IEEE 754 Floating-Point Standard

- Leading “1” bit of significand is implicit
- Exponent is “biased” to make sorting easier
  - all 0s is smallest exponent all 1s is largest
  - bias of 127 for single precision and 1023 for double precision
  - summary: $(-1)^{sign} \cdot (1+significand) \cdot 2^{exponent - bias}$
- Example:
  - decimal: -.75 = - (½ + ¼)
  - binary: -.11 = -1.1 x 2^{-1}
  - floating point: exponent = 126 = 01111110
  - IEEE single precision: `10111111010000000000000000000000`

Floating Point Addition





Floating Point Complexities

- Operations are somewhat more complicated
- In addition to overflow we can have “underflow”
- Accuracy can be a big problem
  - IEEE 754 keeps two extra bits, guard and round
    - For example, add 2.56x100 to 2.34x102→2.37x102 (with guard and round bits) or 2.36 (without guard and round bits)
  - four rounding modes
  - positive divided by zero yields “infinity”
  - zero divide by zero yields “not a number”
  - other complexities