EE 357 Unit 2a

Multiplication Techniques

Learning Objectives

- Perform by hand the different methods for unsigned and signed multiplication
- Understand the various digital implementations of a multiplier along with their tradeoffs
  - Sequential add and shift method
  - Basic combinational array multiplier
  - Booth and/or Bit-Pair multiplier

Add and Shift Method (Sequential)

Booth's Coding and Bit-Pair Recoding

MULTIPLICATION TECHNIQUES

Unsigned Multiplication Review

- Same rules as decimal multiplication
- Multiply each bit of Q by M shifting as you go
- An m-bit * n-bit mult. produces an m+n bit result (i.e. n-bit * n-bit produces 2*n bit result)
- Notice each partial product is a shifted copy of M or 0 (zero)

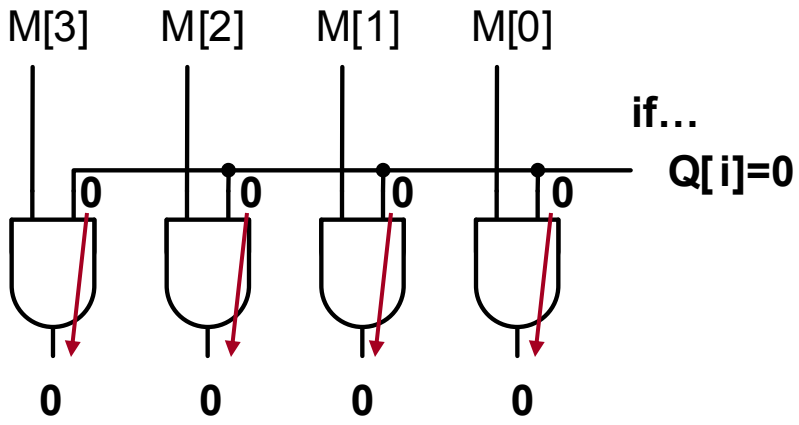
```
    1010   M (Multiplicand)
  * 1011   Q (Multiplier)
  ------
    1010
   1010_   PP(Partial
  0000__      Products)
+ 1010
  ------
 01101110   P (Product)
```

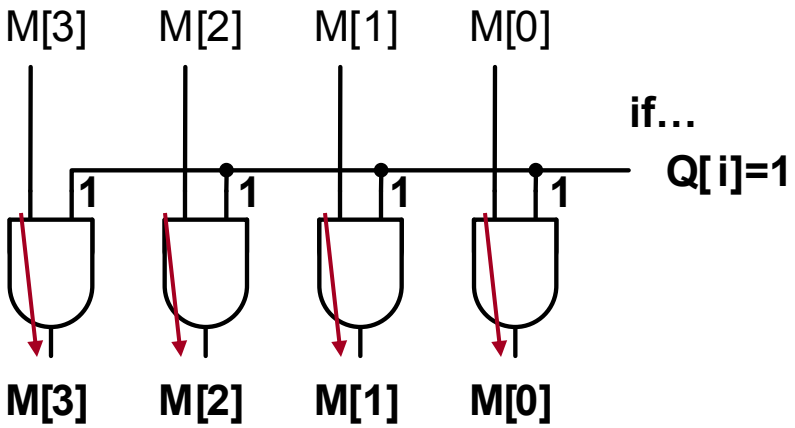
Multiplication Techniques

- A multiplier unit can be
  - Purely Combinational: Each partial product is produced in parallel and fed into an array of adders to generate the product
  - Sequential and Combinational: Produce and add 1 partial product at a time (per cycle)

Combinational Multiplier

- Partial Product (PP_i) Generation
  - Multiply Q[i] * M
    - if Q[i]=0 => PP_i = 0
    - if Q[i]=1 => PP_i = M
  - AND gates can be used to generate each partial product



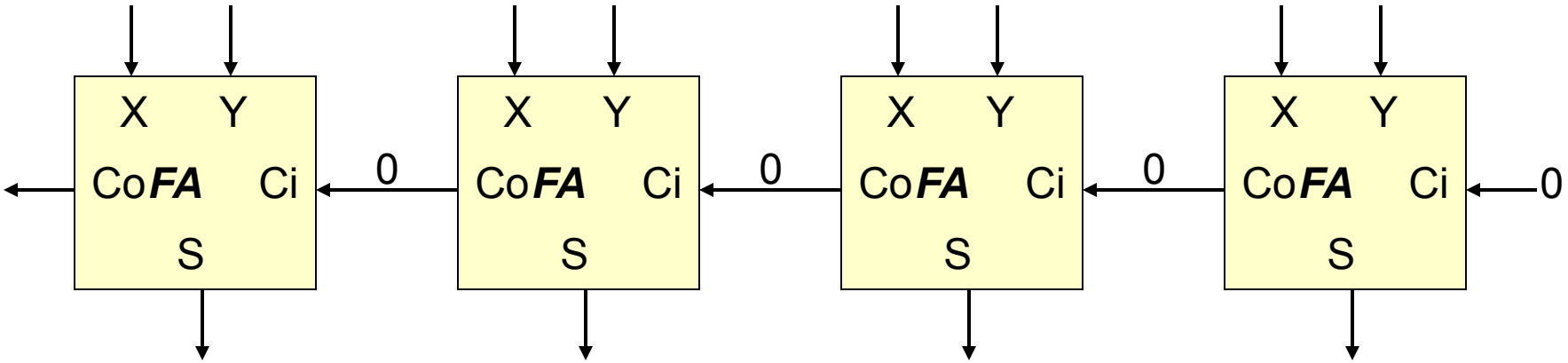


Combinational Multiplier

- Partial Products must be added together
- Combinational multipliers suffer from long propagation delay through the adders
  - propagation delay is proportional to the number of partial products (i.e. number of bits of input) and the width of each adder

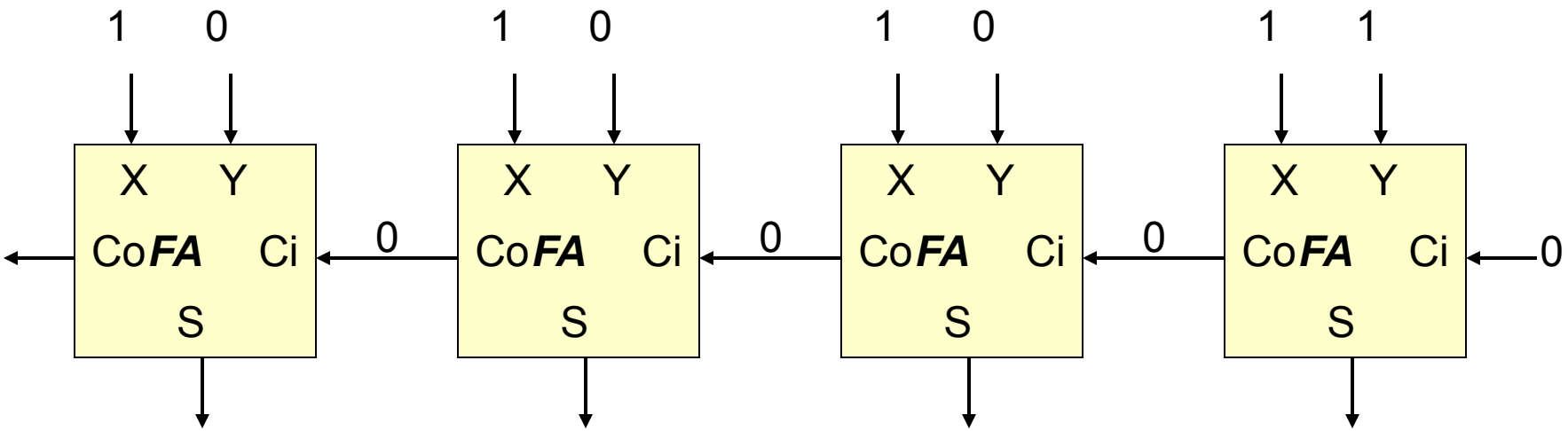
Adder Propagation Delay

```
  1111
+ 0001
------
```



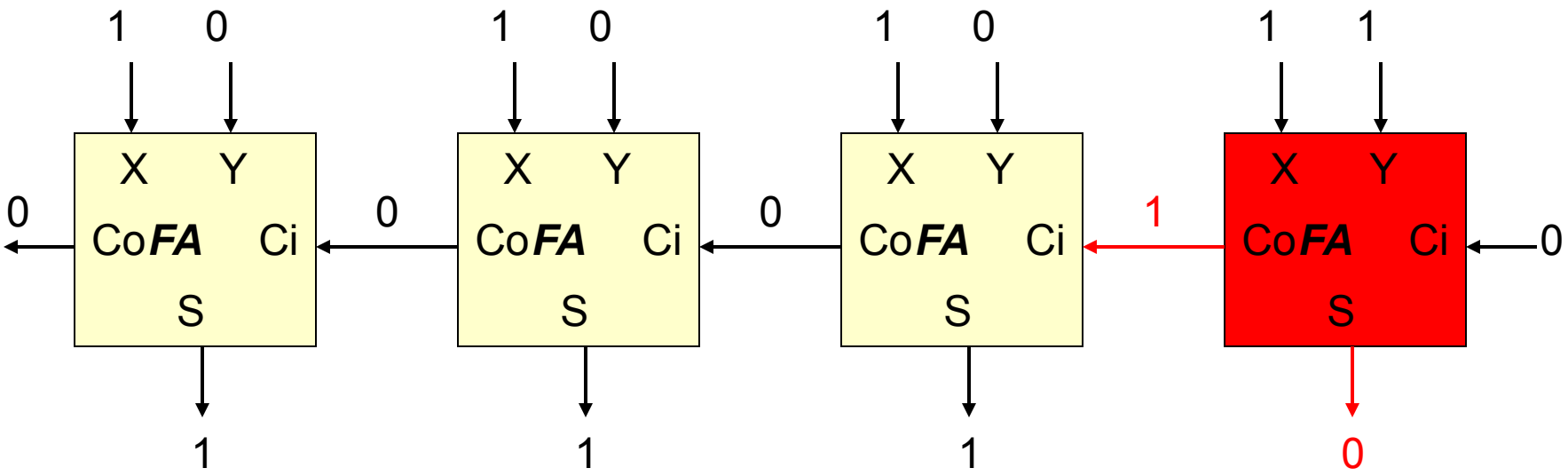
Adder Propagation Delay

```
  1111
+ 0001
------
```



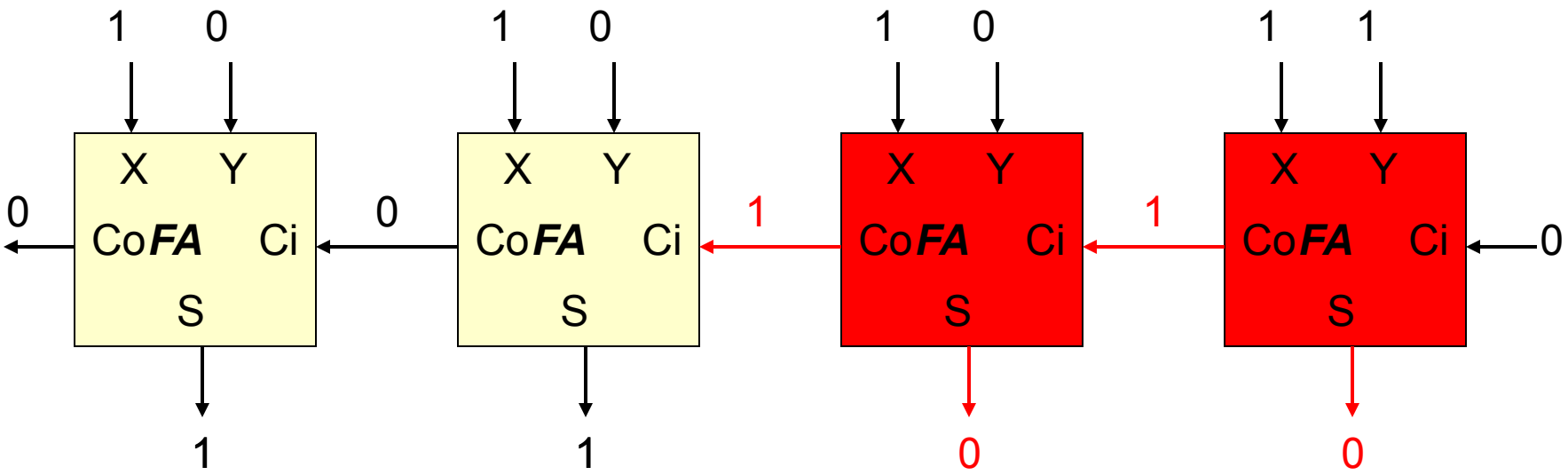
Adder Propagation Delay

```
  1111
+ 0001
------
```



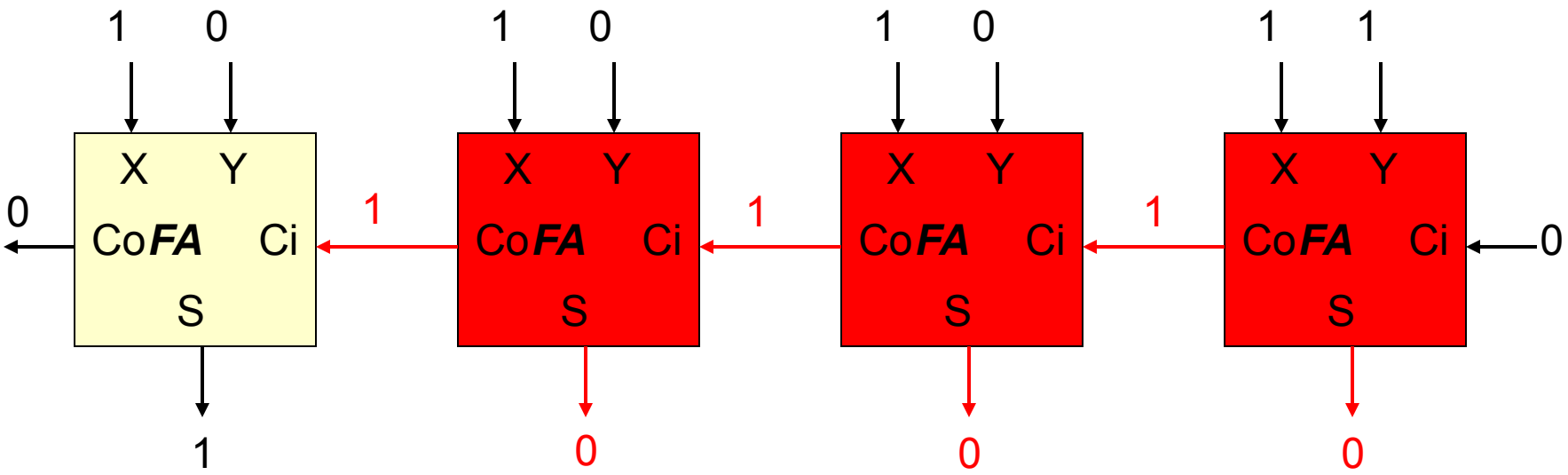
Adder Propagation Delay

```
  1111
+ 0001
------
```



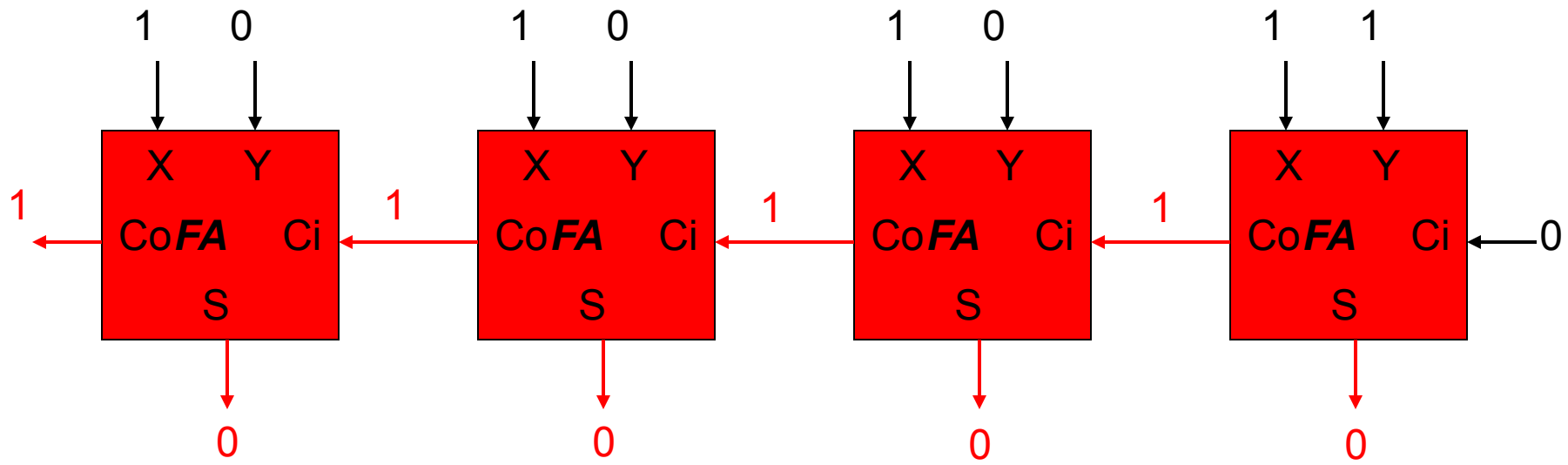
Adder Propagation Delay

```
  1111
+ 0001
------
```



Adder Propagation Delay

```
  1111
+ 0001
------
```

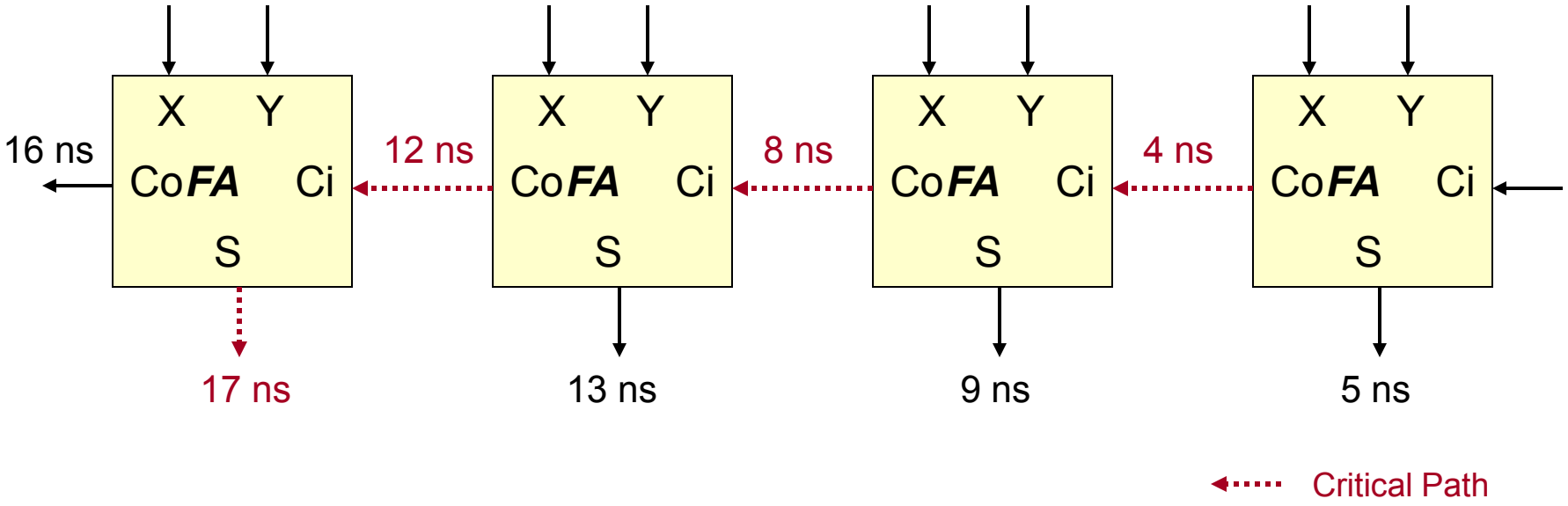


Critical Path

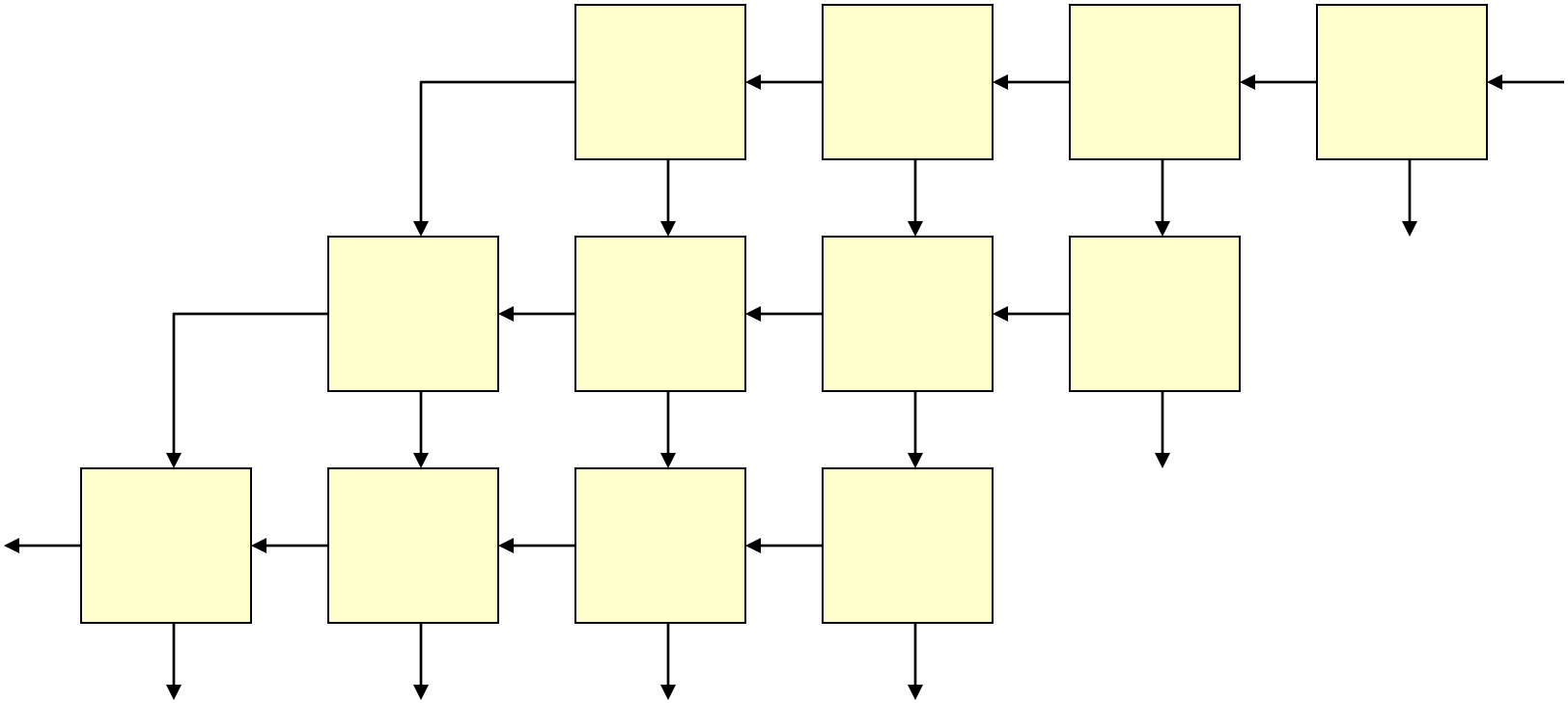
- Critical Path = Longest possible delay path

Assume t_{sum} = 5 ns,

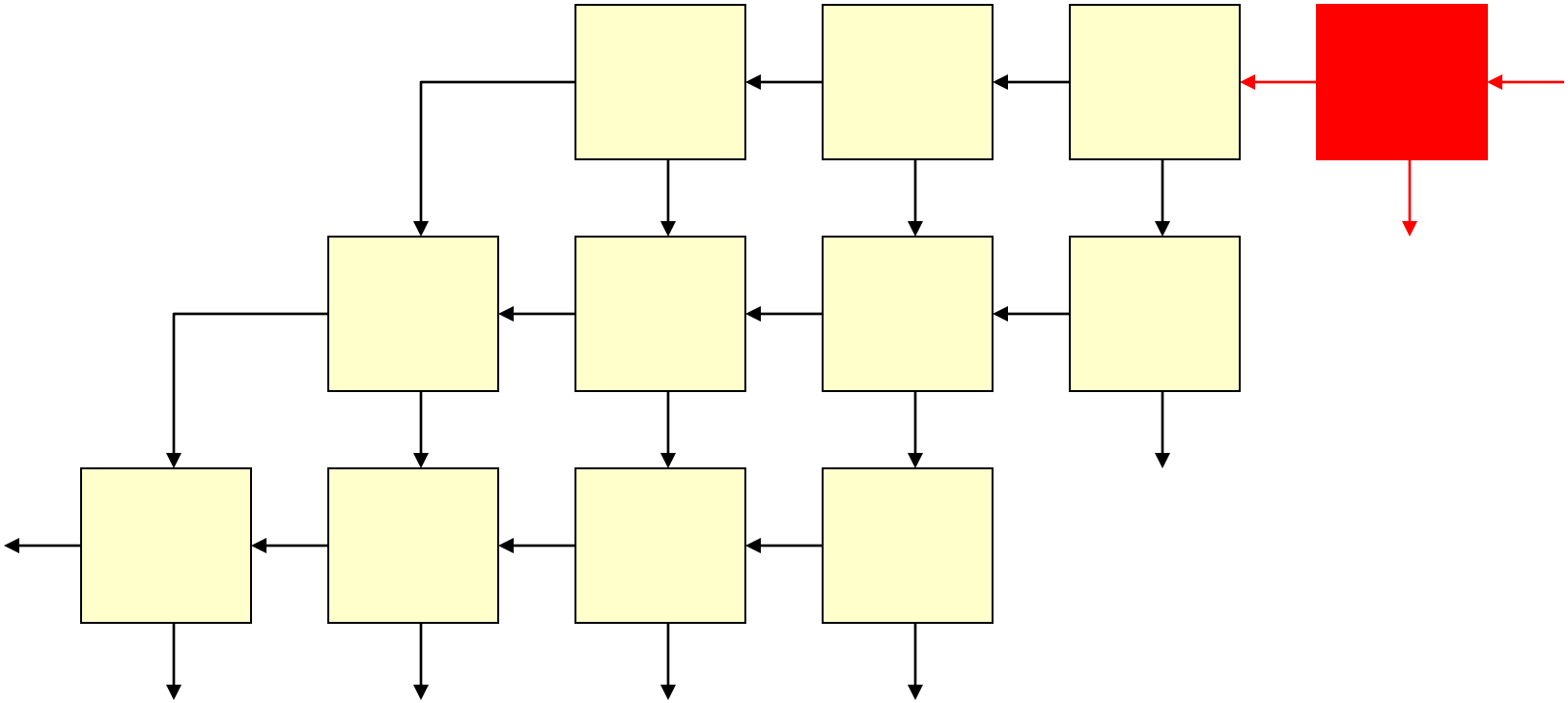
t_{carry} = 4 ns



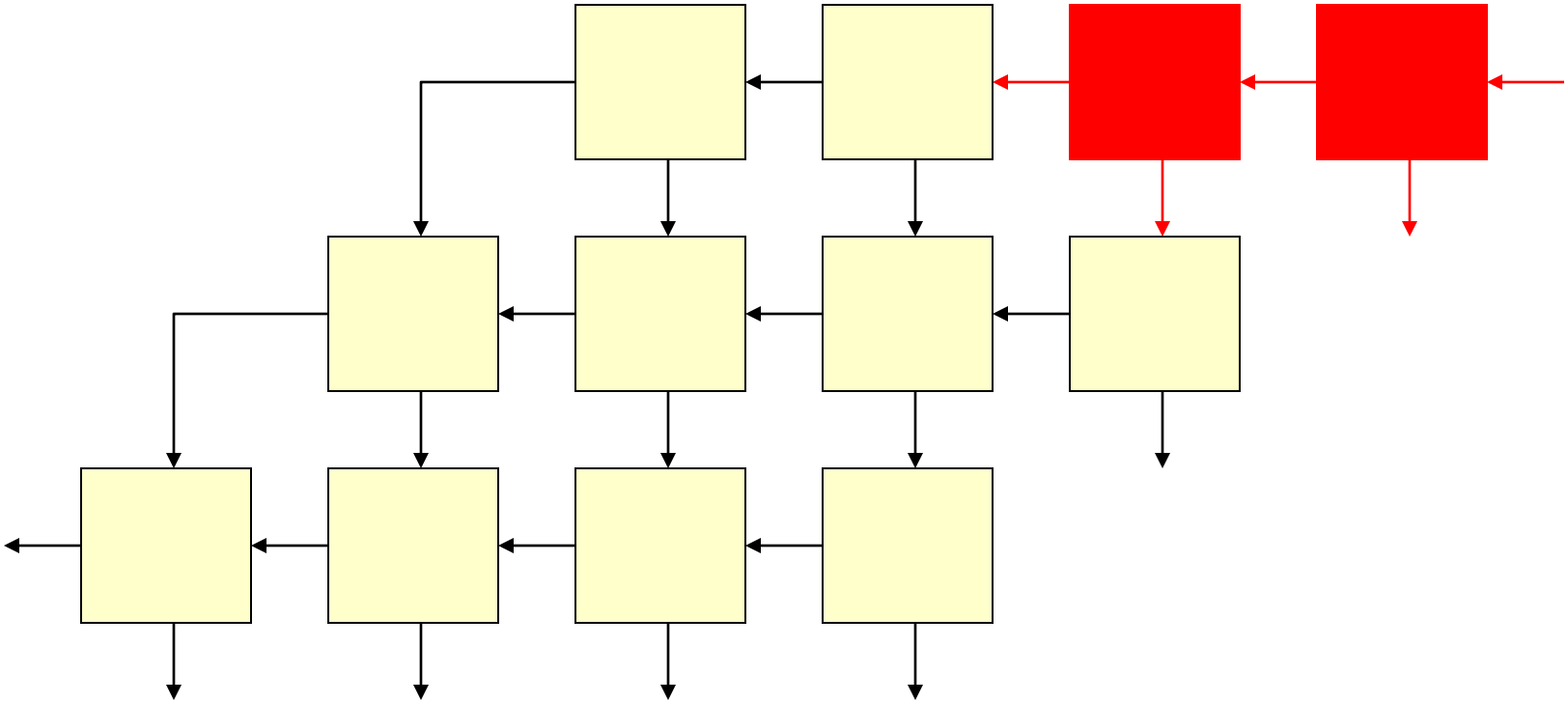
Combinational Multiplier



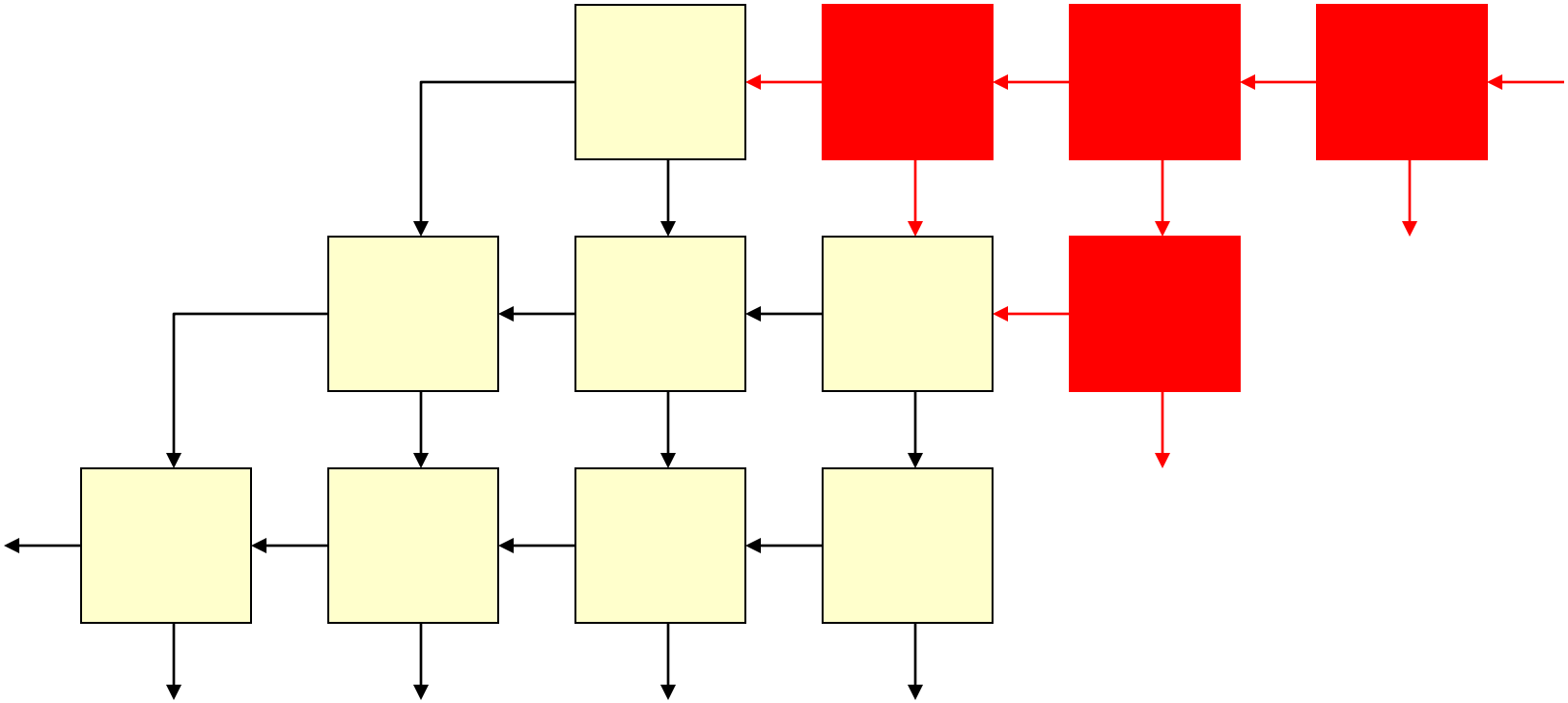
Combinational Multiplier



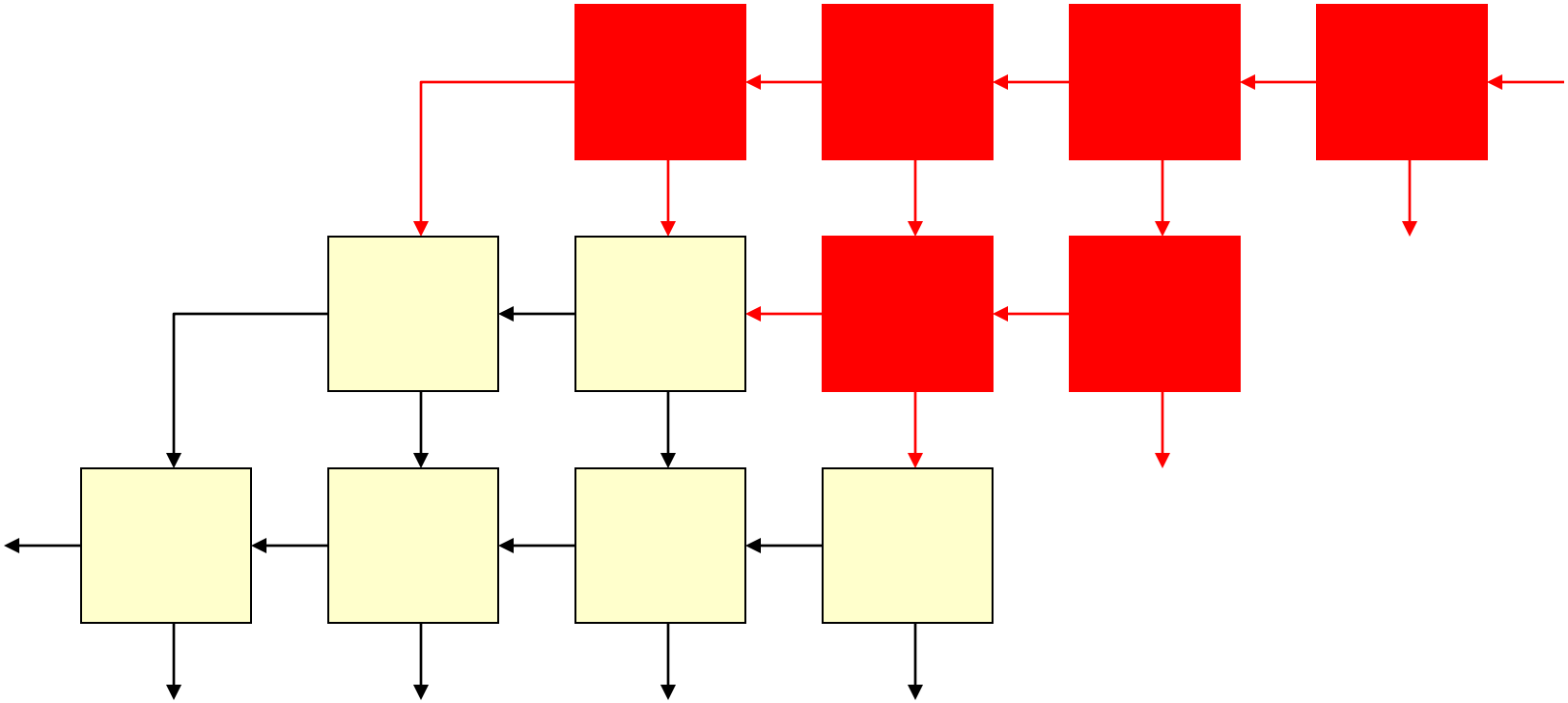
Combinational Multiplier



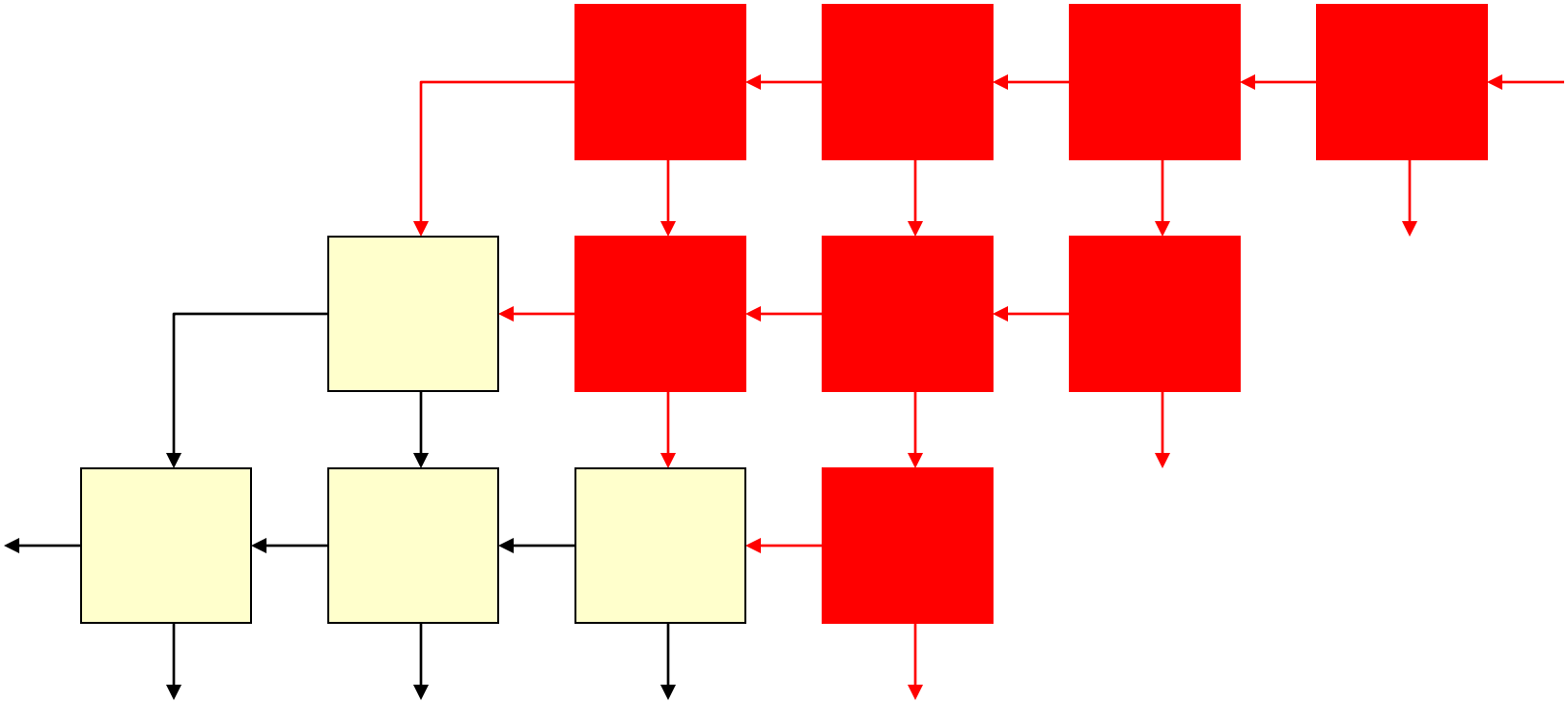
Combinational Multiplier



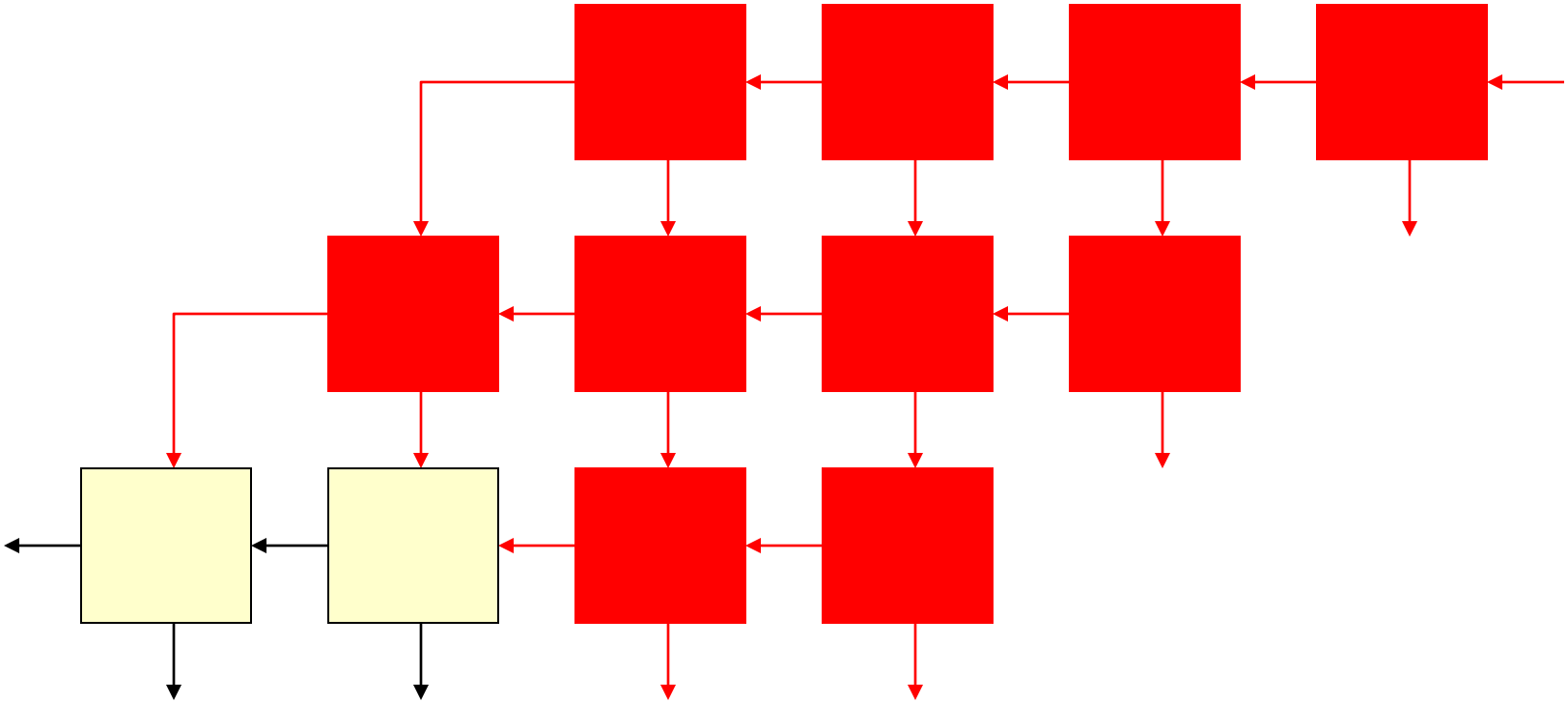
Combinational Multiplier



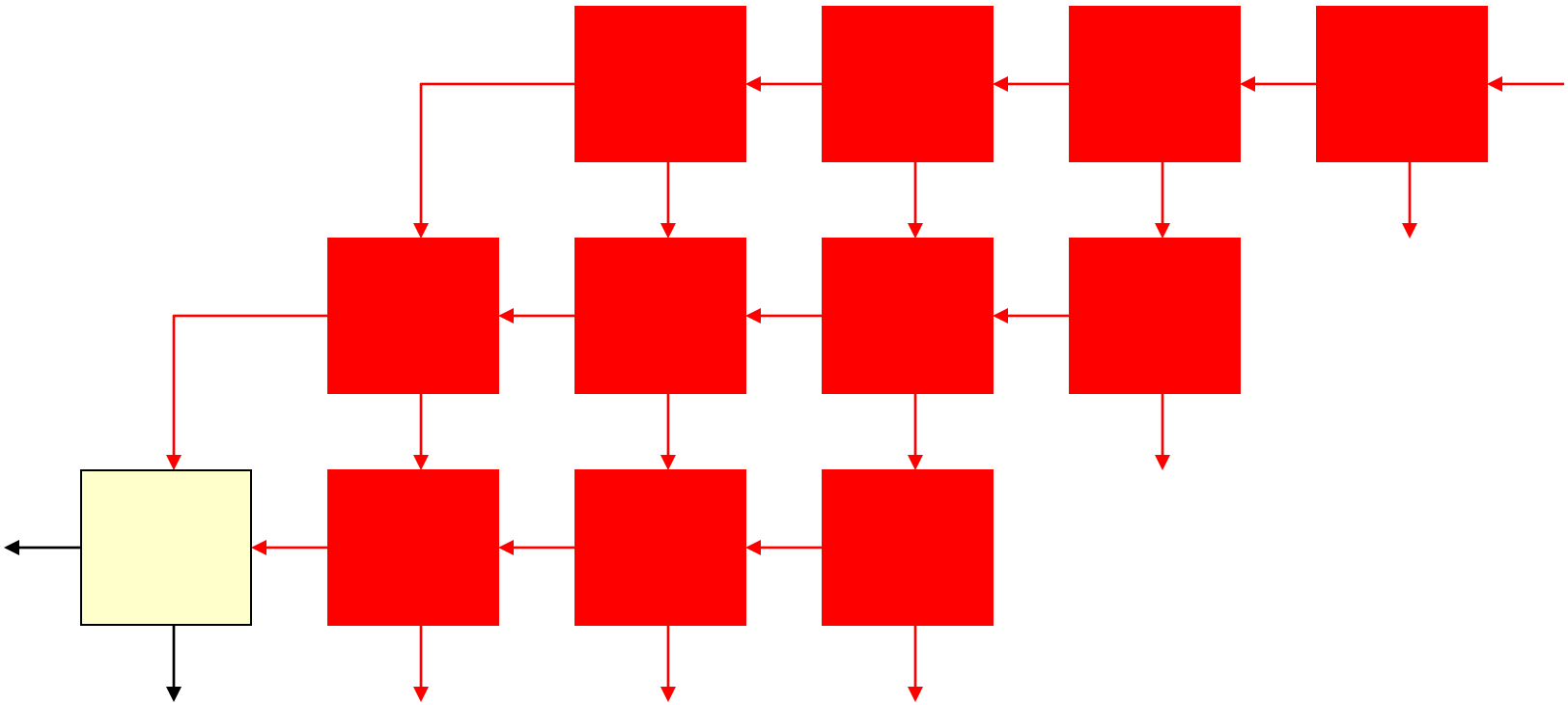
Combinational Multiplier



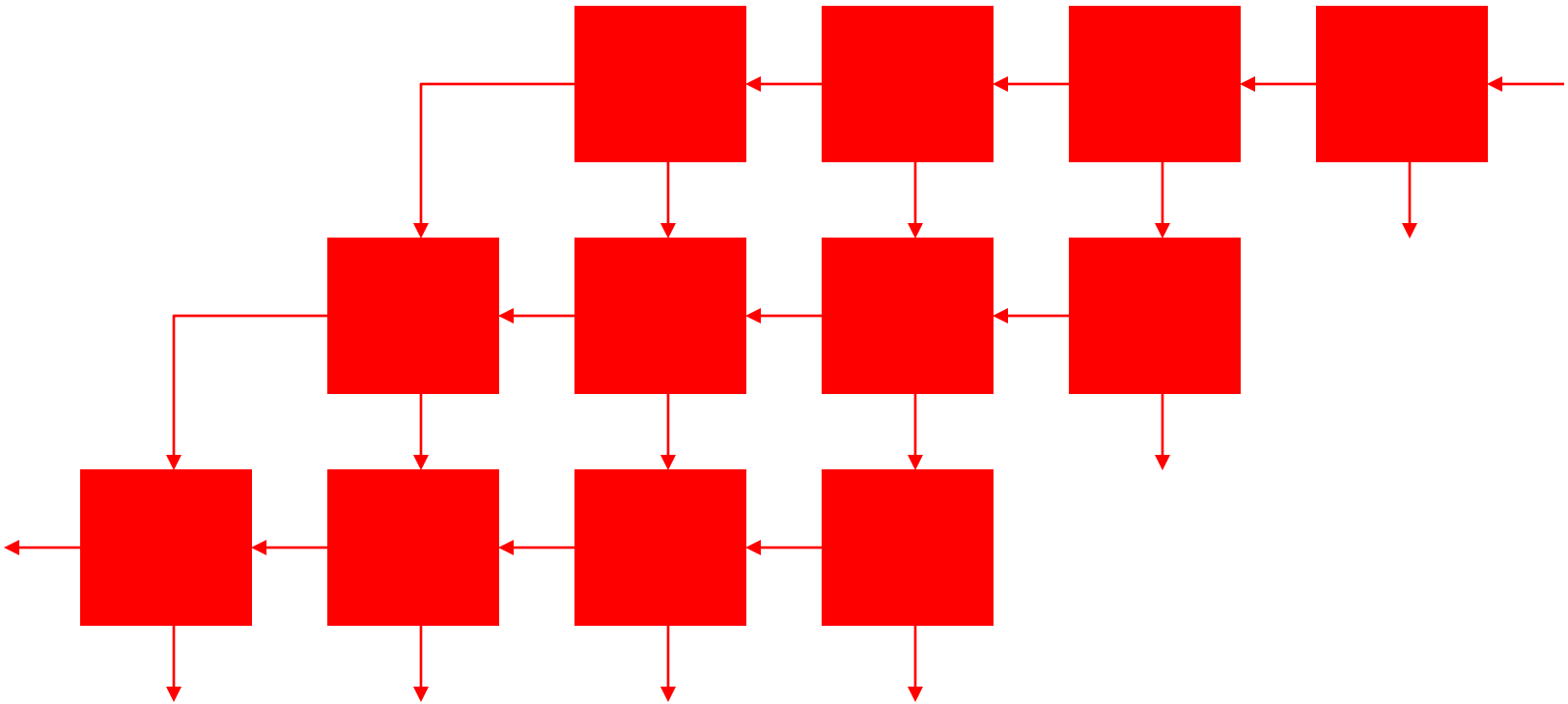
Combinational Multiplier



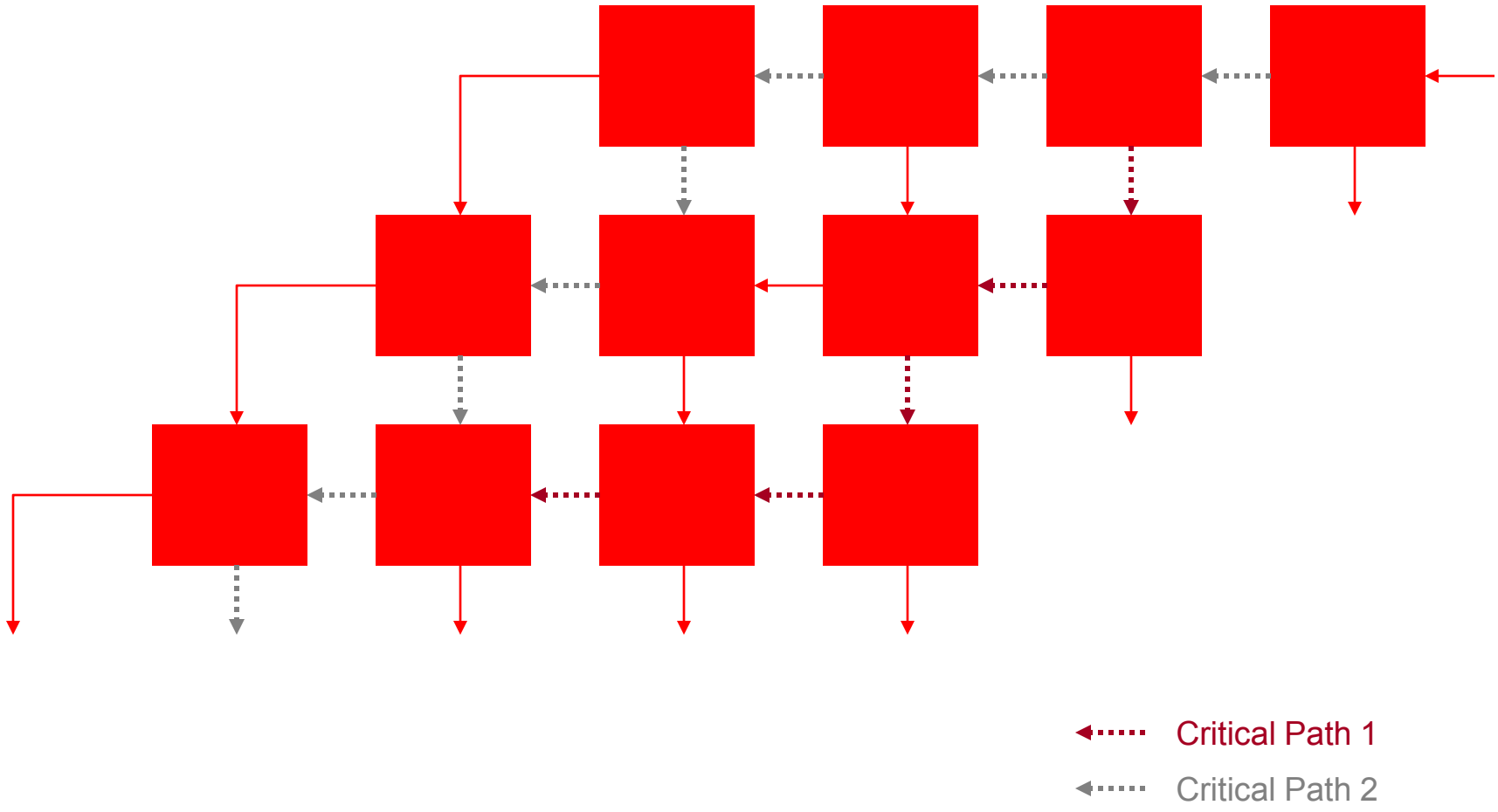
Combinational Multiplier



Combinational Multiplier



Critical Paths



Combinational Multiplier Analysis

- Large Area due to (n-1) m-bit adders
  - n-1 because the first adder adds the first two partial products and then each adder afterwards adds one more partial product
- Propagation delay is in two dimensions
  - proportional to m+n

Sequential Multiplier

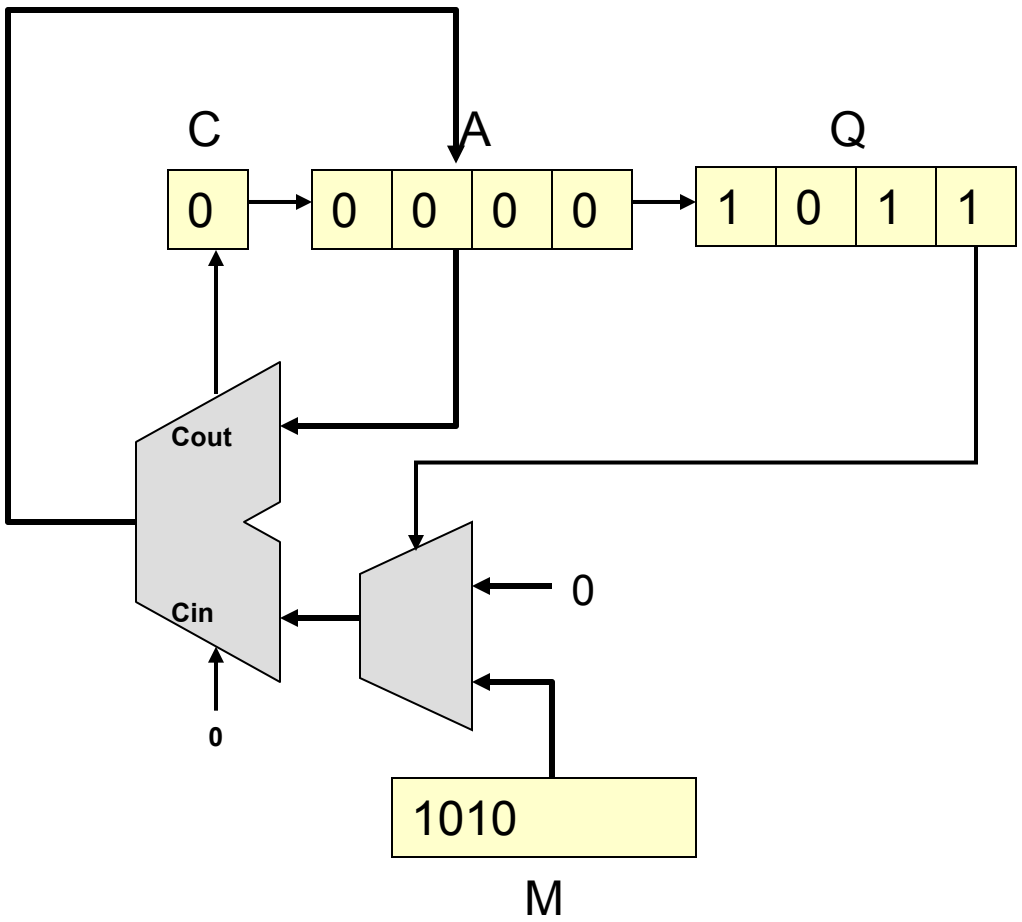
- Use 1 adder to add a single partial product per clock cycle keeping a running sum

Add and Shift Method

- Sequential algorithm
- n-bit * n-bit multiply
- Adds 1 partial product per clock
- Shift running sum 1-bit right each clock
- Three *n*-bit Registers, 1 Adder
- At start:
  - M = Multiplicand
  - Q = Multiplier
  - A = Answer => initialized to 0
- After completion
  - A and Q concatenate to form 2*n*-bit answer

Add and Shift Hardware

```
  1010 = M
* 1011 = Q
```



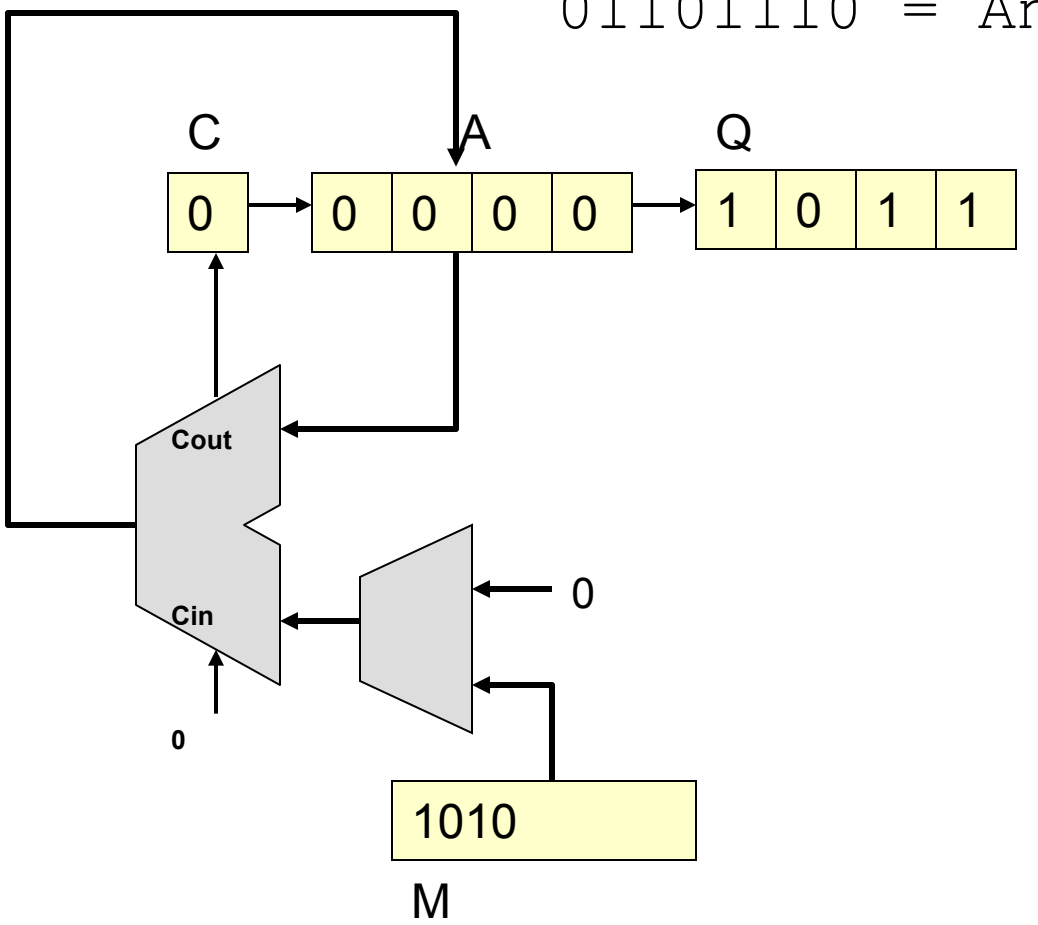
Add and Shift Algorithm

- C=0, A=0
- Repeat the following n-times
  - If Q[0] = 0, A = A+0
    Else if Q[0] = 1, A= A+M
  - Shift right 1-bit (0→C→A→Q)

$$\begin{array}{r} 1010 \\ * \ 1011 \\ \hline \end{array}$$

Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
--------------
01101110 = Ans
```



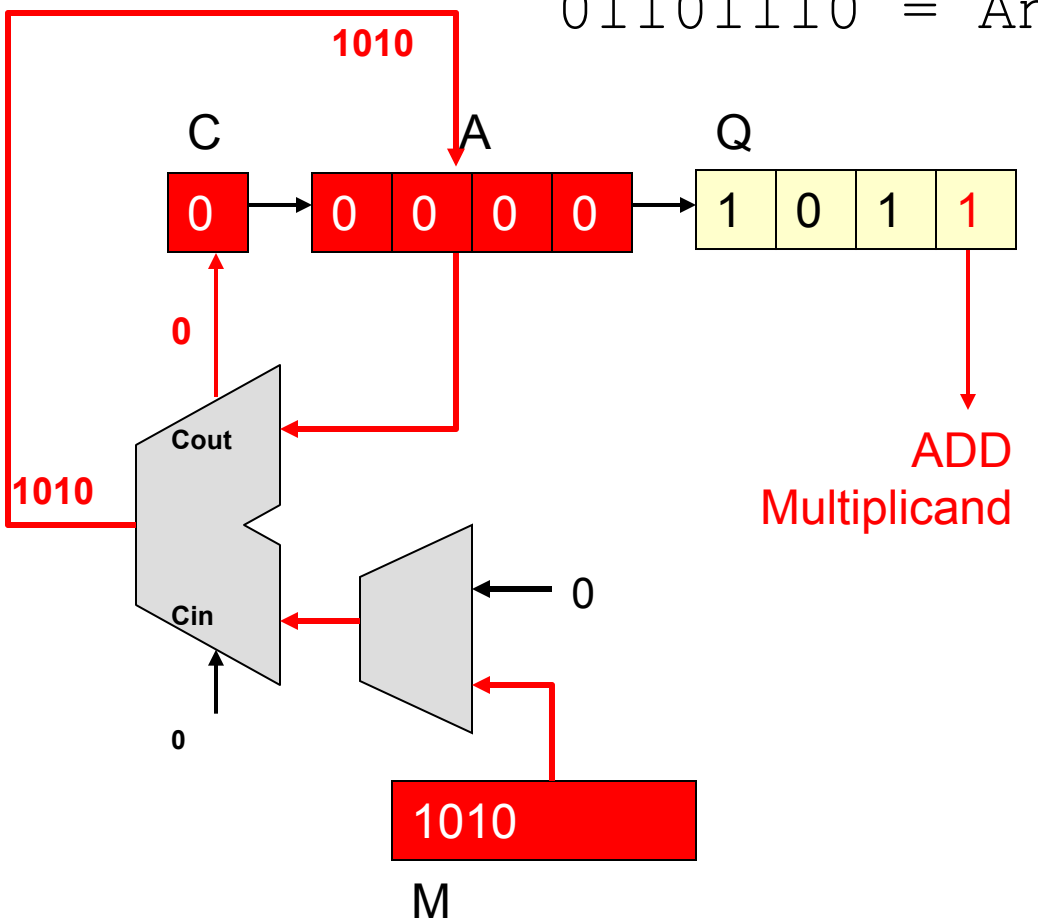
M = 1010

C	A	Q
0	0000	1011

Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
----------
01101110 = Ans
```

```
    1010
  * 1011
--------
  + 1010
--------
    1010
```



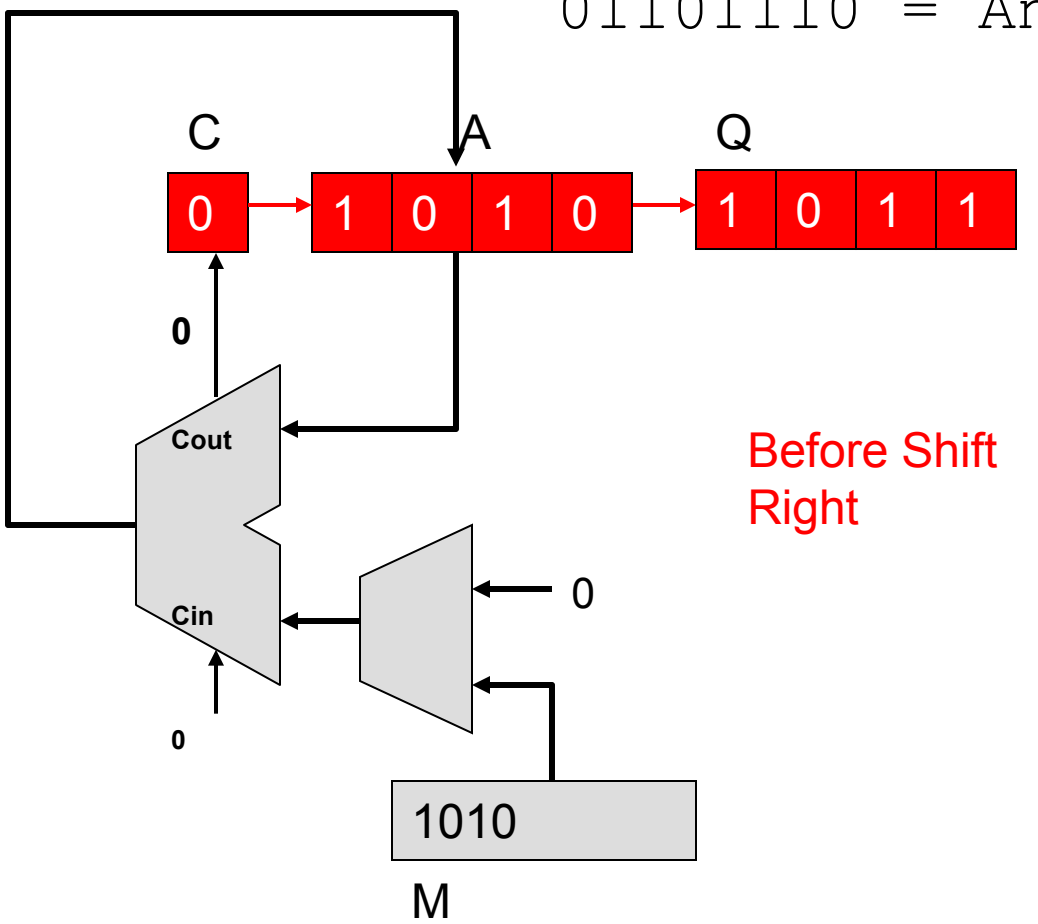
M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add

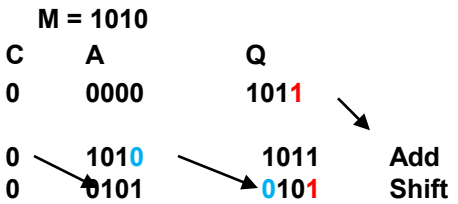
Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
----------
01101110 = Ans
```

```
    1010
  * 1011
-------
  + 1010
-------
    1010
```



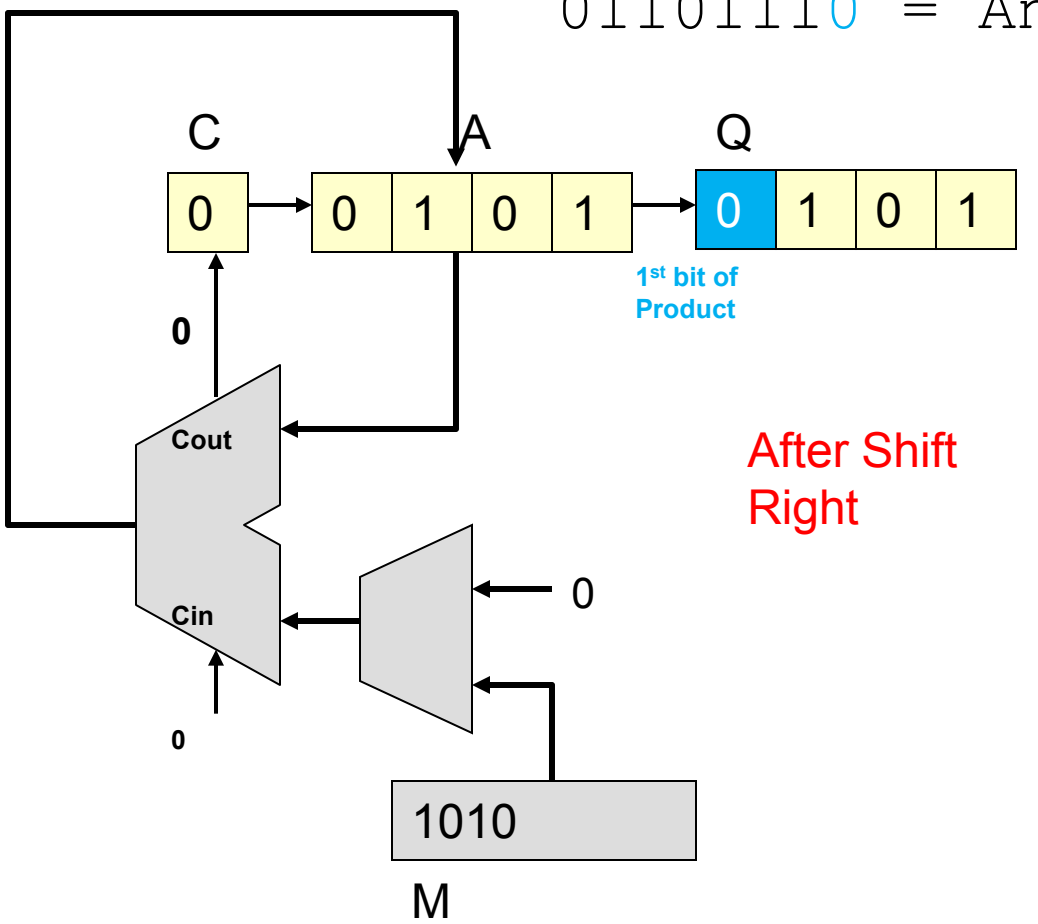
Before Shift Right



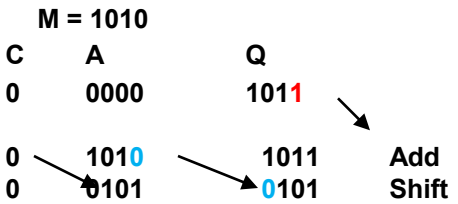
Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
----------
01101110 = Ans
```

```
    1010
  * 1011
-------
  + 1010
-------
    1010
```



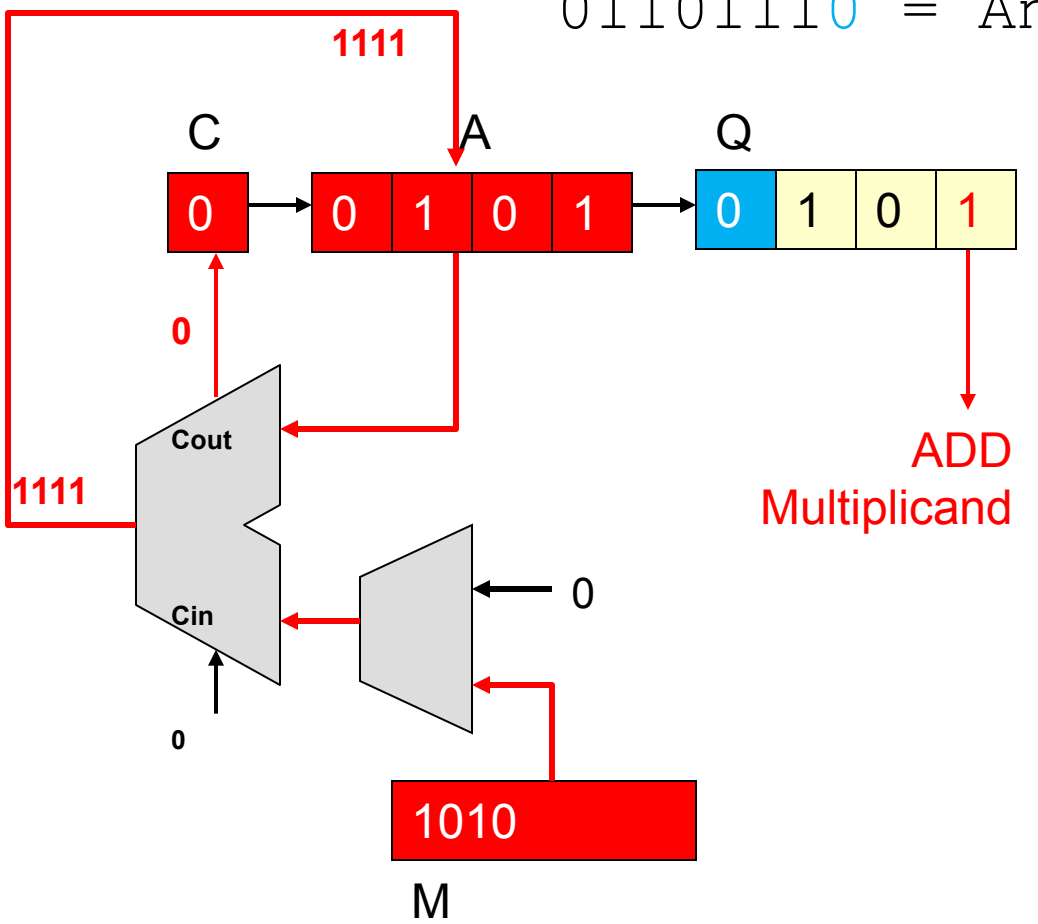
After Shift Right



Add and Shift Multiplication

```
  1010 = M
* 1011 = Q
----------
01101110 = Ans
```

```
   1010
 * 1011
------
   1010
------
   1010
 + 1010-
------
  011110
```



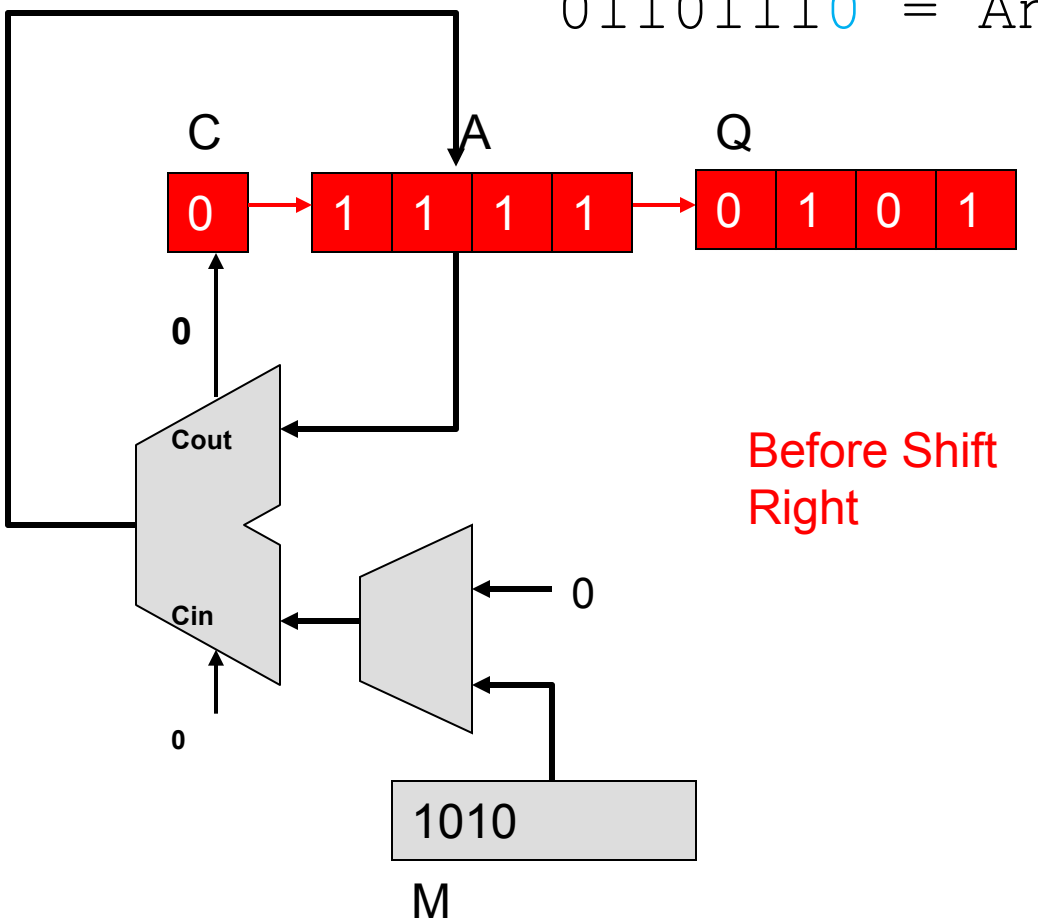
M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add
0	0101	0101	Shift
0	1111	0101	Add

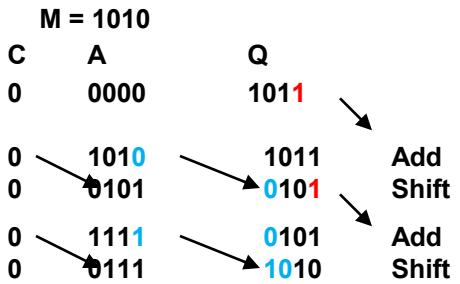
Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
--------------
01101110 = Ans
```

```
    1010
  * 1011
--------
    1010
--------
    1010
  + 1010-
--------
  011110
```



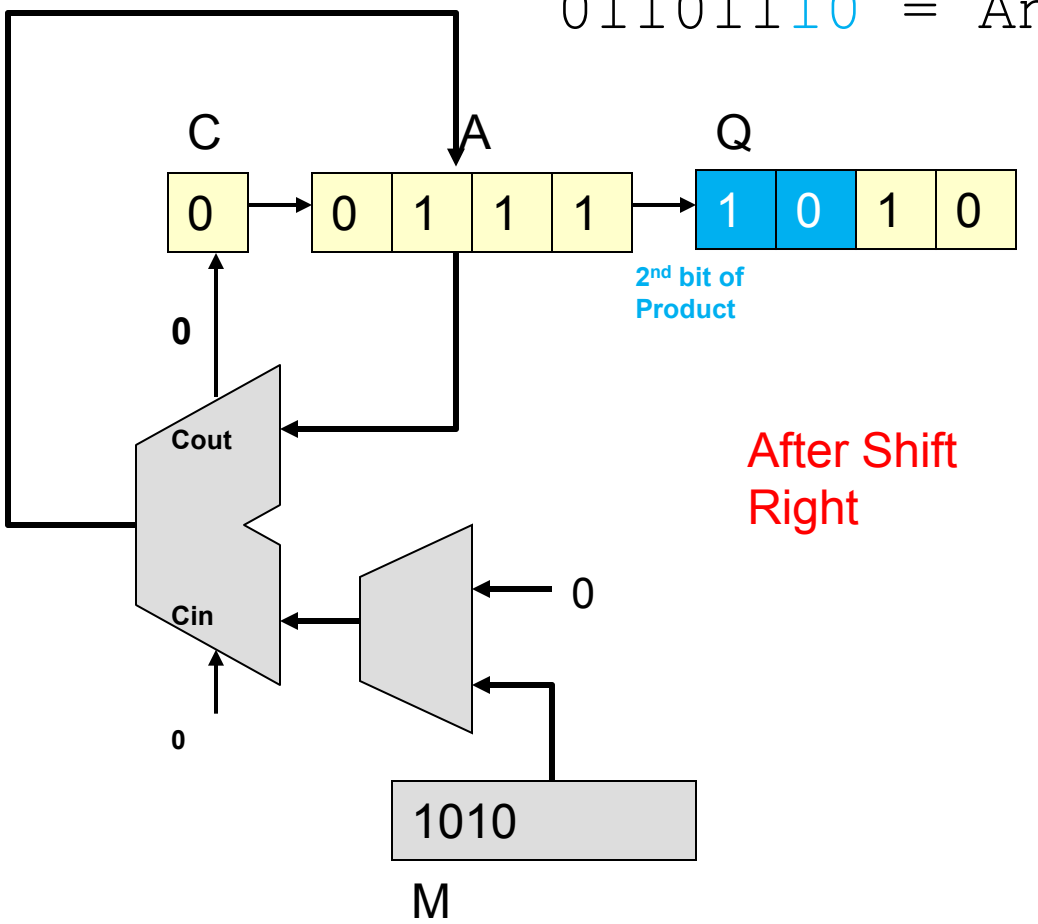
Before Shift Right



Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
----------
01101110 = Ans
```

```
    1010
  * 1011
--------
    1010
--------
    1010
  + 1010-
--------
  011110
```



After Shift Right

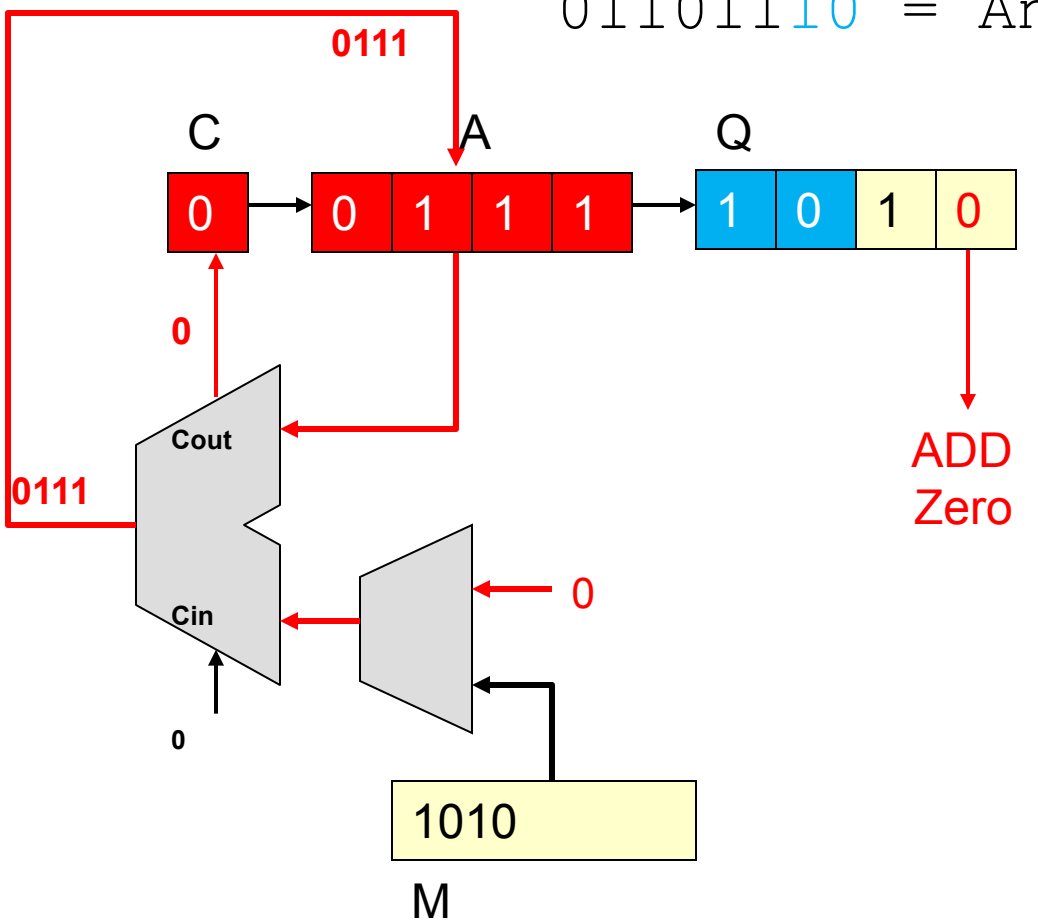
M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add
0	0101	0101	Shift
0	1111	0101	Add
0	0111	1010	Shift

Add and Shift Multiplication

```
  1010 = M
* 1011 = Q
----------
01101110 = Ans
```

```
    1010
  * 1011
-------
  + 1010
-------
    1010
  + 1010-
-------
   011110
 + 0000--
-------
  0011110
```



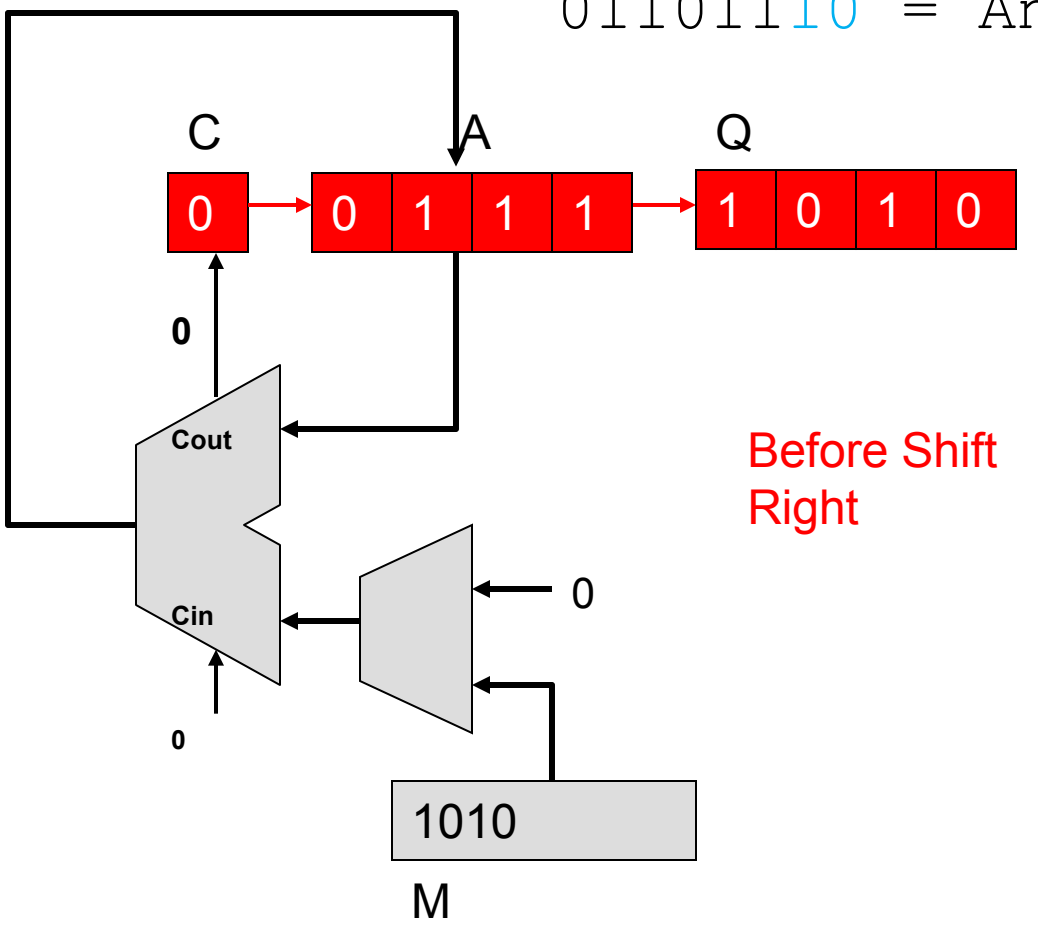
M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add
0	0101	0101	Shift
0	1111	0101	Add
0	0111	1010	Shift
0	0111	1010	No Add

Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
----------
01101110 = Ans
```

```
    1010
  * 1011
----------
  + 1010
----------
    1010
  + 1010-
----------
   011110
 + 0000--
----------
  0011110
```



Before Shift Right

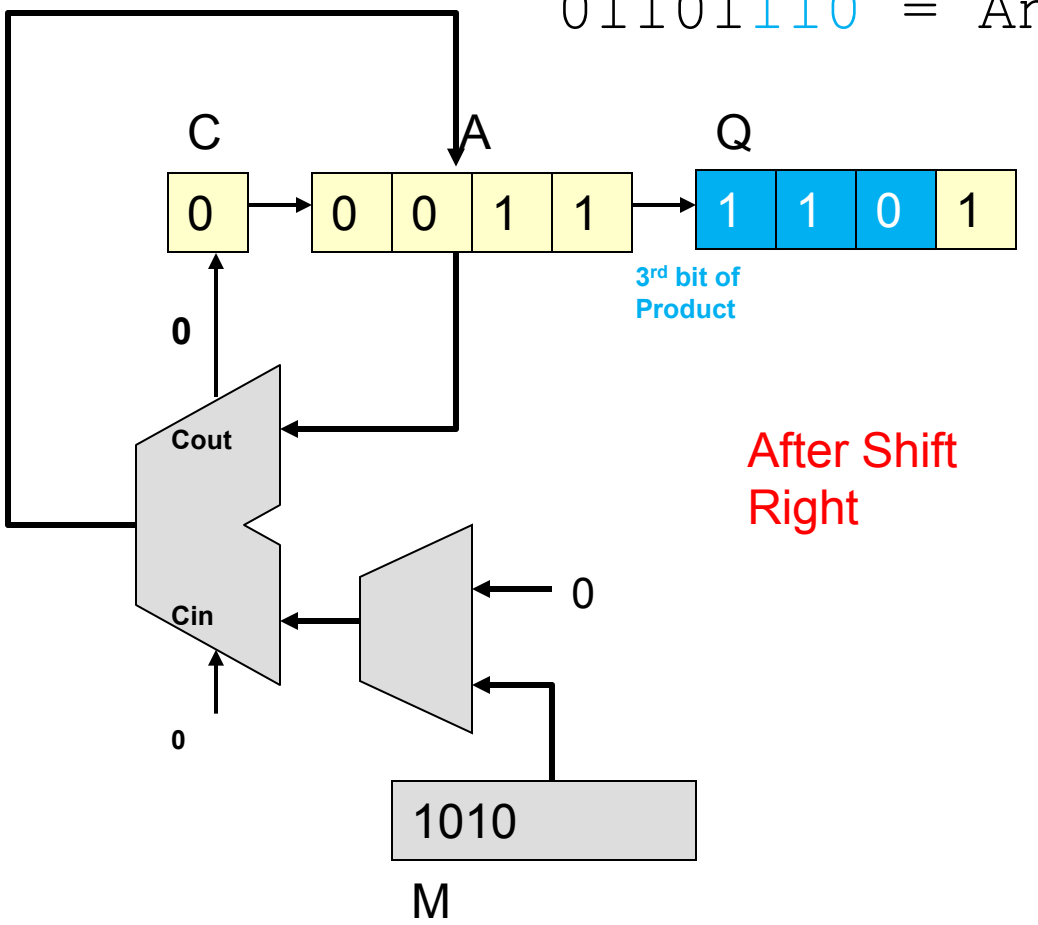
M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add
0	0101	0101	Shift
0	1111	0101	Add
0	0111	1010	Shift
0	0111	1010	No Add
0	0011	1101	Shift

Add and Shift Multiplication

```
  1010 = M
* 1011 = Q
----------
01101110 = Ans
```

```
    1010
  * 1011
----------
  + 1010
----------
    1010
  + 1010-
----------
   011110
 + 0000--
----------
  0011110
```



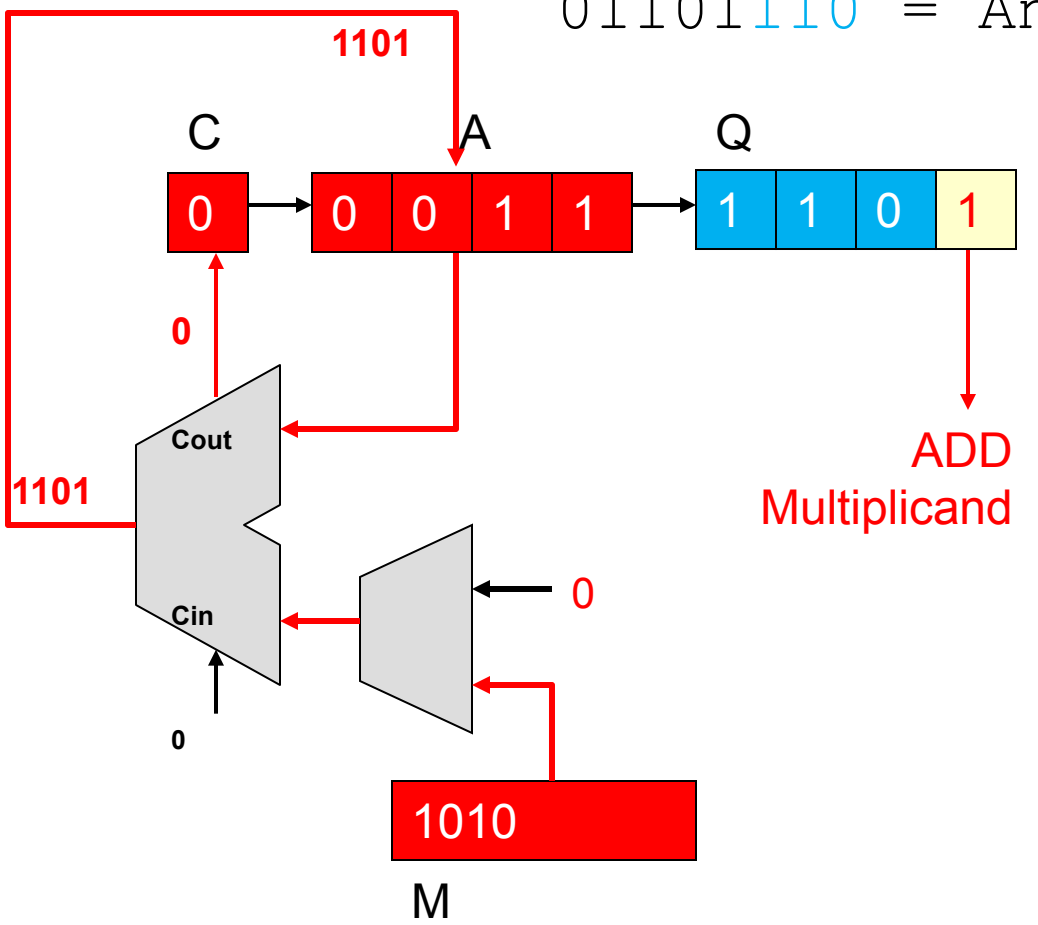
After Shift Right

M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add
0	0101	0101	Shift
0	1111	0101	Add
0	0111	1010	Shift
0	0111	1010	No Add
0	0011	1101	Shift

Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
--------------
01101110 = Ans
```



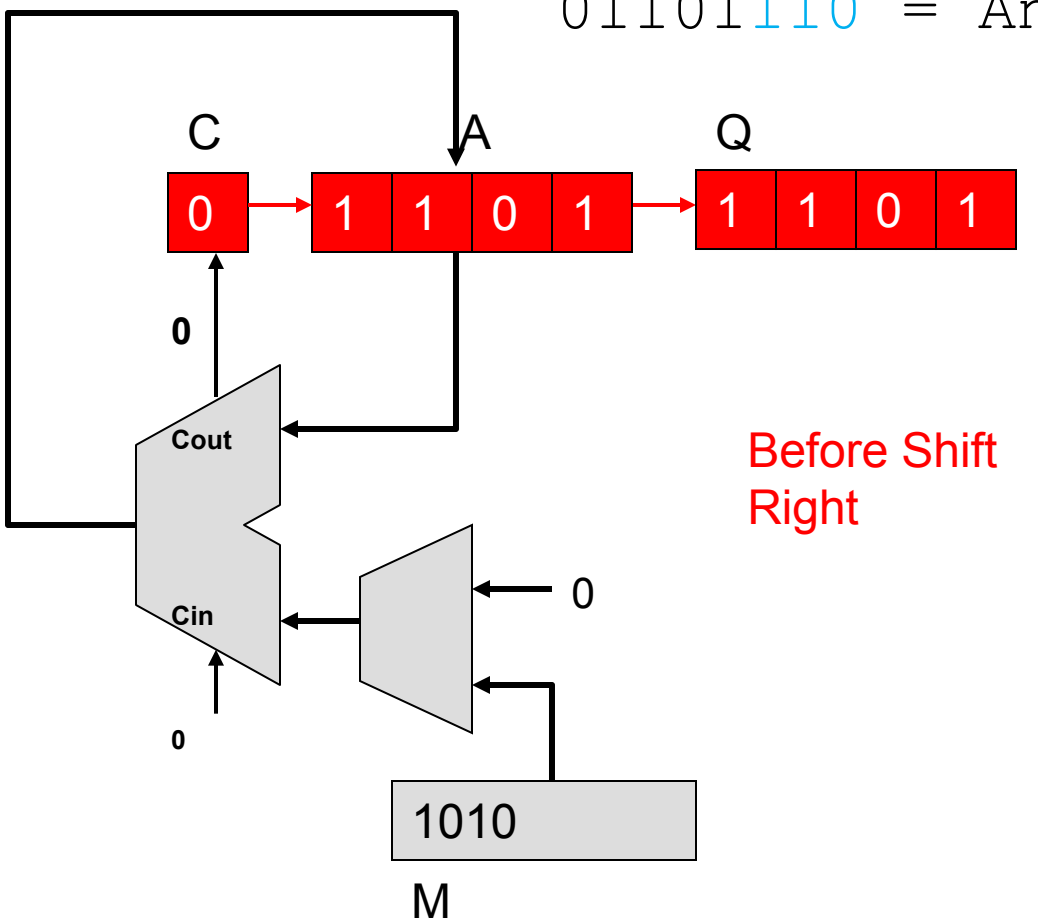

```
    1010
  * 1011
----------
  + 1010
----------
    1010
  + 1010-
----------
   011110
  + 0000--
----------
   0011110
 + 1010---
----------
  01101110
```

M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add
0	0101	0101	Shift
0	1111	0101	Add
0	0111	1010	Shift
0	0111	1010	No Add
0	0011	1101	Shift
0	1101	1101	Add

Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
--------------
01101110 = Ans
```



Before Shift Right

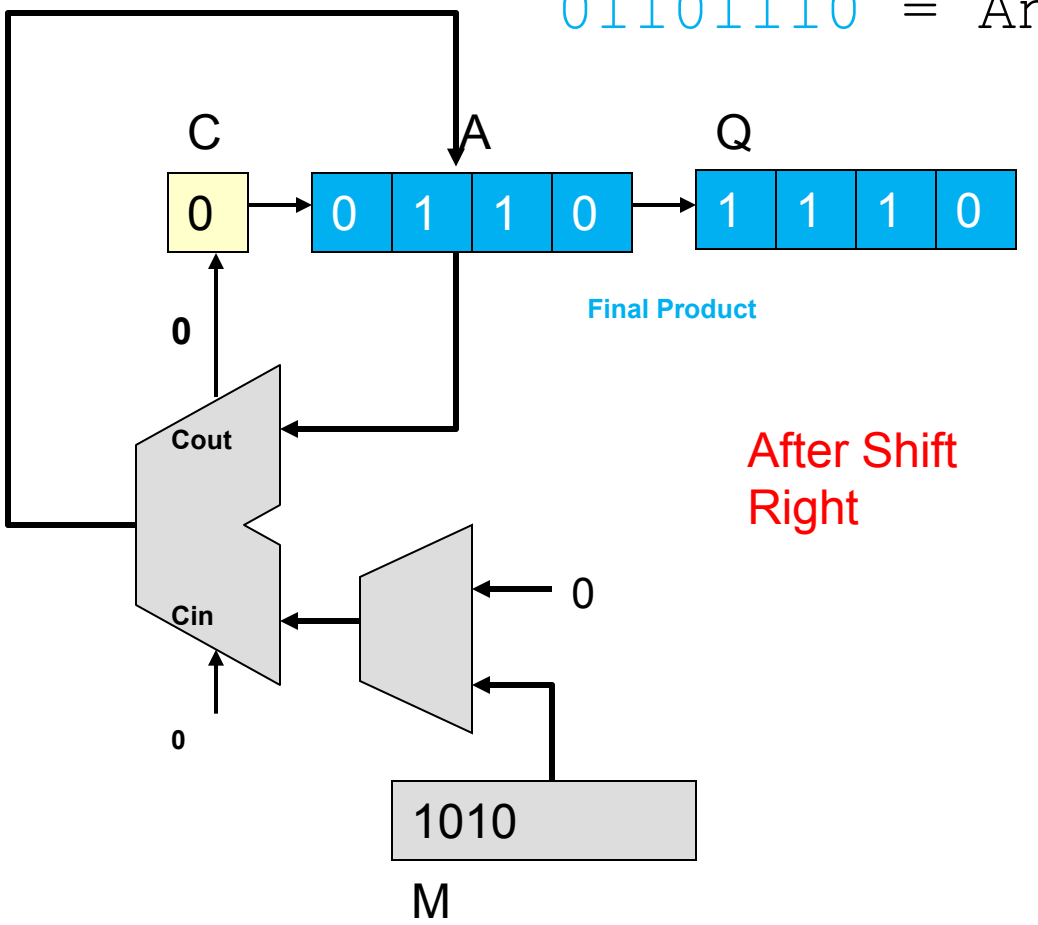
```
     1010
   * 1011
----------
   + 1010
----------
    1010
  + 1010-
----------
   011110
 + 0000--
----------
  0011110
+ 1010---
----------
 01101110
```

M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add
0	0101	0101	Shift
0	1111	0101	Add
0	0111	1010	Shift
0	0111	1010	No Add
0	0011	1101	Shift
0	1101	1101	Add
0	0110	1110	Shift

Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
--------------
01101110 = Ans
```



After Shift Right

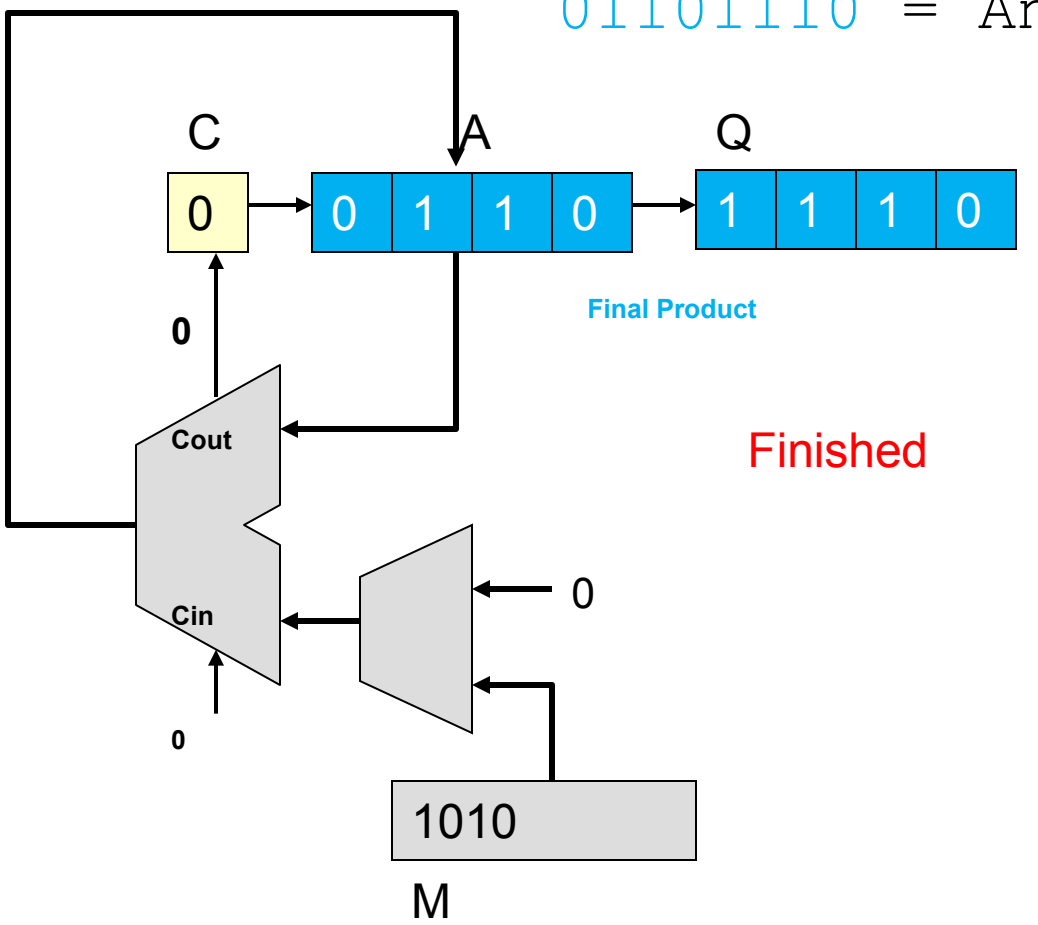
```
    1010
  * 1011
----------
  + 1010
----------
   1010
 + 1010-
----------
  011110
+ 0000--
----------
 0011110
+ 1010---
----------
 01101110
```

M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add
0	0101	0101	Shift
0	1111	0101	Add
0	0111	1010	Shift
0	0111	1010	No Add
0	0011	1101	Shift
0	1101	1101	Add
0	0110	1110	Shift

Add and Shift Multiplication

```
   1010 = M
 * 1011 = Q
--------------
01101110 = Ans
```



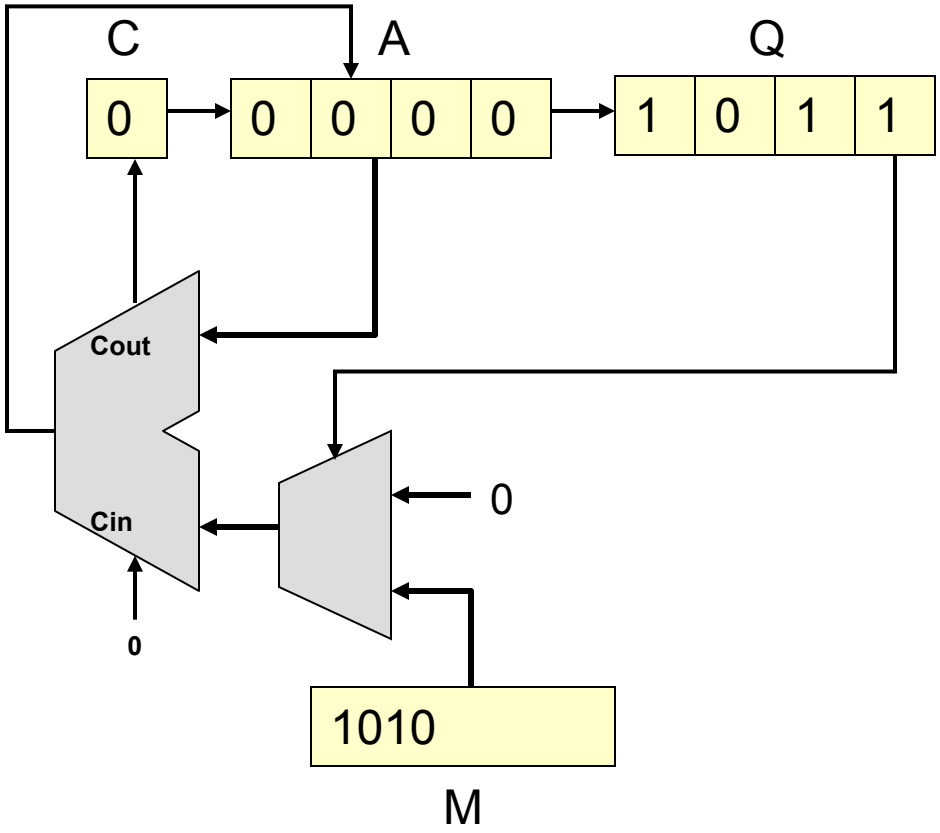
Finished

```
    1010
  * 1011
----------
  + 1010
----------
   1010
 + 1010-
----------
  011110
+ 0000--
----------
 0011110
+ 1010---
----------
 01101110
```

M = 1010

C	A	Q	
0	0000	1011	
0	1010	1011	Add
0	0101	0101	Shift
0	1111	0101	Add
0	0111	1010	Shift
0	0111	1010	No Add
0	0011	1101	Shift
0	1101	1101	Add
0	0110	1110	Shift
	0110	1110	= 110_{10}

1101 * 0101 Example



C= 0	M=1101 A=0000	Q=0101	Description
0	1101	0101	A=A+M
0	0110	1010	Shift Right C,A,Q
0	0110	1010	A=A+0
0	0011	0101	Shift Right C,A,Q
1	0000	0101	A=A+M
0	1000	0010	Shift Right C,A,Q
0	1000	0010	A=A+0
0	0100	0001	Shift Right C,A,Q

Sequential Multiplier Analysis

- Pros:
  - Smaller Area due to the use of only 1 adder
- Cons:
  - Slow to execute (2 cycles per bit of the multiplier)

Multipliers

- Most multipliers are combinational
  - Faster than sequential method
- To increase speed
  - Make faster adders
  - Reduce # of additions (i.e. # of partial products)…See Bit Pair Recoding later in this lecture

Signed Multiplication Techniques

- When adding signed (2's comp.) numbers, some new issues arise
- Must sign extend partial products (out to 2n bits)

**Without Sign Extension…
Wrong Answer!**

```
      1001  = -7
    * 0110  = +6
    ------
      0000
     1001_
    1001__
+  0000___
  ---------
  00110110  = +54
```

**With Sign Extension…
Correct Answer!**

```
      1001  = -7
    * 0110  = +6
    ------
  00000000
  1111001_
  111001__
+ 00000___
  ---------
  11010110  = -42
```

Signed Multiplication Techniques

- Also, must worry about negative multiplier
  - MSB of multiplier has negative weight
  - If MSB=1, multiply by -1 (i.e. take 2's comp. of multiplicand)

**With Sign Extension but w/o consideration of MSB…
Wrong Answer!**

```
    1100  =  -4
  * 1010  =  -6
  --------
  00000000
  1111100_
  000000__
+ 11100___
  --------
  11011000  =  -40
```

**With Sign Extension and w/ consideration of MSB…
Correct Answer!**

Place Value: -8
Multiply by -1

```
    1100  =  -4
  * 1010  =  -6
  --------
  00000000
  1111100_
  000000__
+ 00100___
  --------
  00011000  =  +24
```

Booth's Algorithm

- Used for signed multiplication
  - Works regardless of sign of either operand
- Forms the basis for an optimization known as bit-pair recoding
- Requires recoding of multiplier
  - 0's and 1's → 0's, 1's, -1's
  - Must still sign extend partial products

Booth Recoding

- Recode the multiplier using the truth table below
- Start at MSB of the multiplier and work towards the LSB
  - Bit i = current bit
  - Bit i-1 = bit to the right of current bit

Bit *i*	Bit *i-1*	Booth's Value
0	0	0
1	1	0
0	1	+1
1	0	-1

Notice Booth's Value is equal to the change from bit i to bit (i-1)…or really [bit i-1] – [bit i]

Booth's Algorithm

Original Problem:

```
    0 1 0 1 0     (+10)
*   1 0 0 1 1     (-13)
-----------------
```

Booth's Algorithm

Original Problem:

```
    0 1 0 1 0     (+10)
*   1 0 0 1 1     (-13)
```

-1

Bit *i*	Bit *i-1*	Booth's Value
0	0	0
1	1	0
0	1	+1
1	0	-1

Booth's Algorithm

Original Problem:

```
    0 1 0 1 0     (+10)
* 1 0 0 1 1     (-13)
-----------
```

```
-1 0
```

Bit *i*	Bit *i-1*	Booth's Value
0	0	0
1	1	0
0	1	+1
1	0	-1

Booth's Algorithm

Original Problem:

```
    0 1 0 1 0     (+10)
* 1 0 0 1 1     (-13)
```

```
-1  0+1
```

Bit *i*	Bit *i-1*	Booth's Value
0	0	0
1	1	0
0	1	+1
1	0	-1

Booth's Algorithm

Original Problem:

```
    0 1 0 1 0     (+10)
* 1 0 0 1 1     (-13)
```

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

Bit *i*	Bit *i-1*	Booth's Value
0	0	0
1	1	0
0	1	+1
1	0	-1

Booth's Algorithm

Original Problem:

```
   0  1  0  1  0      (+10)
*  1  0  0  1  1  0   (-13)
```

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

Bit *i*	Bit *i-1*	Booth's Value
0	0	0
1	1	0
0	1	+1
1	0	-1

Booth's Algorithm

Original Problem:

```
          0 1 0 1 0    (+10)
        * 1 0 0 1 1    (-13)
        ---------------

          0 1 0 1 0
        * -1 0+1 0-1
        ------------
  1 1 1 1 1 1 0 1 1 0
  0 0 0 0 0 0 0 0 0
  0 0 0 0 1 0 1 0
  0 0 0 0 0 0 0
+ 1 1 0 1 1 0
-----------------------
  1 1 0 1 1 1 1 1 1 0    (-130)
```

Bit *i*	Bit *i-1*	Booth's Value
0	0	0
1	1	0
0	1	+1
1	0	-1

Booth's Example

Original Problem:

```
      1 0 1 1 1    (-9)
    * 0 0 1 1 0    (+6)
    ---------------
```

```
      1 0 1 1 1
    * 0+1 0-1 0
    -----------
  0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 1 0 0 1
  0 0 0 0 0 0 0 0
  1 1 1 0 1 1 1
+ 0 0 0 0 0 0
-------------------
  1 1 1 1 0 0 1 0 1 0    (-54)
```

Bit *i*	Bit *i-1*	Booth's Value
0	0	0
1	1	0
0	1	+1
1	0	-1

Booth's Algorithm Summary

- Recode multiplier (Q) and perform multiplication
- Works for positive or negative operands

Booth's Algorithm Summary

- Recode multiplier (Q) and perform multiplication
- Works for positive or negative operands

How fast is multiplication?

Still required to add (n-1) partial products

Bit-Pair Recoding

- Works on top of Booth's algorithm
- Cuts # of partial products in half
- Method
  - Group every 2 Booth encoded multiplier bits
  - Convert to: -2,-1,0,+1,+2
  - Perform multiplication

Bit Pair Recoding

Original Problem:

```
   0 1 0 1 0    (+10)
* 1 0 1 1 1    (-9)
-------------
```

Bit Pair Recoding

Original Problem:

```
    0  1  0  1  0      (+10)
*   1  0  1  1  1  0   (-9)
```

Booth's Algorithm

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

Bit Pair Recoding

Original Problem:

```
    0  1  0  1  0      (+10)
*   1  0  1  1  1  0   (-9)
```

Group in pairs of two starting from LSB

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

Bit Pair Recoding

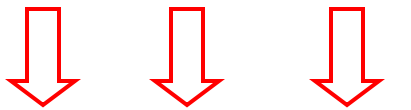
Original Problem:

```
   0  1  0  1  0      (+10)
*  1  0  1  1  1  0   (-9)
```


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

Find pair equivalent using binary place values (e.g. 0*2 + (-1)*1 = -1)

2 1 2 1 2 1 Place values



```
 -1   +2   -1
```

Bit Pair Recoding

Bit-Pair Recoded Problem:

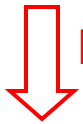
```
                      0 1 0 1 0       (+10)
                  * -1   +2  -1       (-9)
                  ---------------
      1   1 1
    1 1 1 1 1 1 0 1 1 0
    0 0 0 1 0 1 0 0 - -
  + 1 1 0 1 1 0 - - - -
  ---------------------
    1 1 1 0 1 0 0 1 1 0               (-90)
```

This zero is added for multiplication by 2

Bit Pair Recoding Example

Bit-Pair Recoded Problem:

```
    1  1  1  1  0      (-2)
*   1  1  0  1  0  0   (-6)
```



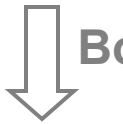
Booth's Algorithm

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

Bit Pair Recoding Example

Original Problem:

```
   1  1  1  1  0      (-2)
*  1  1  0  1  0  0   (-6)
```




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

Find pair equivalent using place values (e.g. -1*2 + 0*1 = -1)

2 1 2 1 2 1 Place values

```
   0   -1   -2
```

Bit Pair Recoding Example

Bit-Pair Recoded Problem:

```
                    1  1  1  1  0     (-2)
                 *  0    -1    -2     (-6)
              ---------------------
      0  0  0  0  0  0  0  1  0  0
      0  0  0  0  0  0  1  0  -  -
   +  0  0  0  0  0  0  -  -  -  -
   ---------------------------------
      0  0  0  0  0  0  1  1  0  0     (+12)
```

This zero is added for multiplication by 2

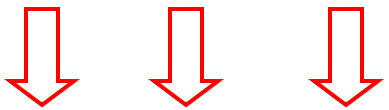
Bit Pair Recoding Example

Original Problem:

```
   1  1  0  1  0      (-6)
*  0  1  0  0  1  0   (+9)
```


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

2 1 2 1 2 1 *Place values*



```
 +1   -2   +1
```

Bit Pair Recoding Example

Bit-Pair Recoded Problem:

```
                    1 1 0 1 0       (-6)
                  *+1   -2   +1     (+9)
                  -------------
        1       1
    1 1 1 1 1 1 1 0 1 0
    0 0 0 0 1 1 0 0 - -
  + 1 1 1 0 1 0 - - - -
  ---------------------
    1 1 1 1 0 0 1 0 1 0       (-54)
```

This zero is added for multiplication by 2