

**CPS 104**  
**Computer Organization and Programming**  
**Lecture-12: Memory elements, Buses, Registers.**  
**Integer Arithmetic.**

**Oct. 15, 1999**

**Dietolf (Dee) Ramm**

**<http://www.cs.duke.edu/~dr/cps104.html>**

# Overview of Today's Lecture:

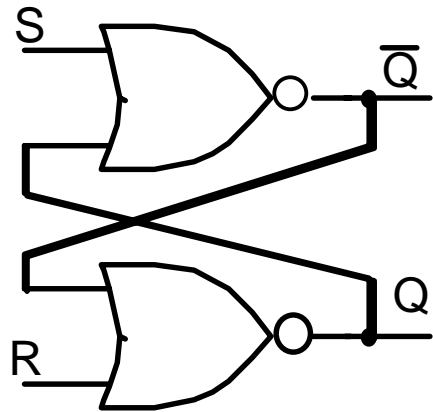
- **Memory elements:**
  - \* **Latch,**
  - \* **Data-FlipFlop**
  - \* **Registers**
- **Tristate drivers and bus interconnects.**
- **Integer Multiplication and Division.**

**Read Appendix B**

# Memory Elements

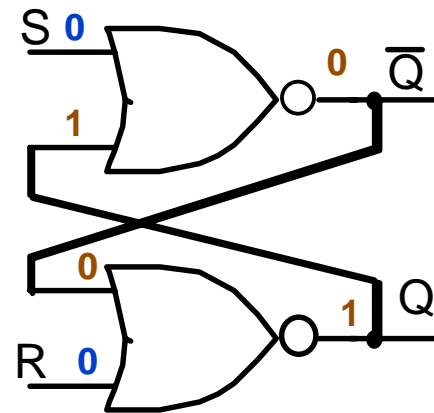
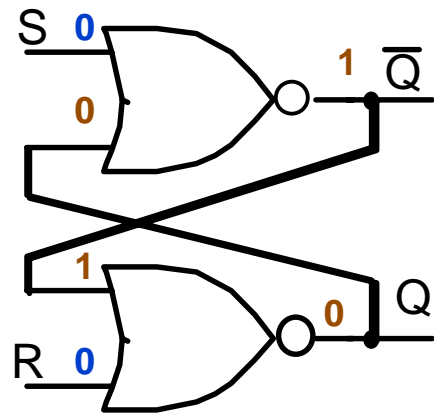
- All the circuit we looked at so far are **combinational circuits**: the output is a Boolean function of the inputs.
- We need circuits that can remember values. (registers)
- The output of the circuit is a function of the input AND a function of a stored values (state) .
- Circuits with memory are called **sequential circuits**.

## Reset-Set Latch

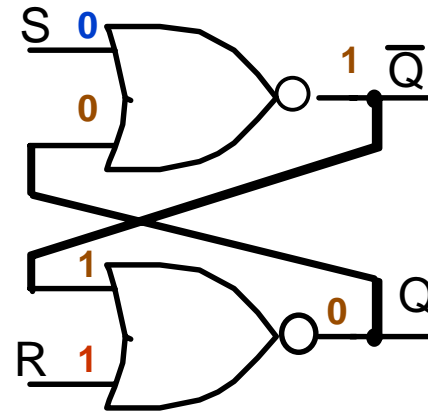
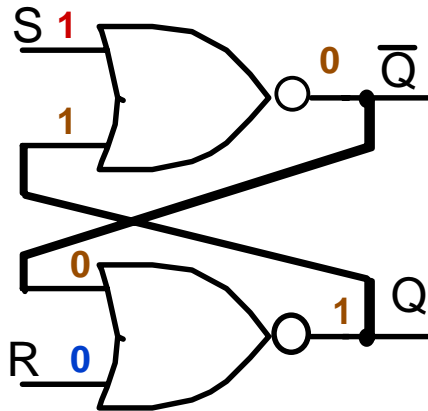


R	S	Q
0	0	Q
0	1	1
1	0	0
1	1	-

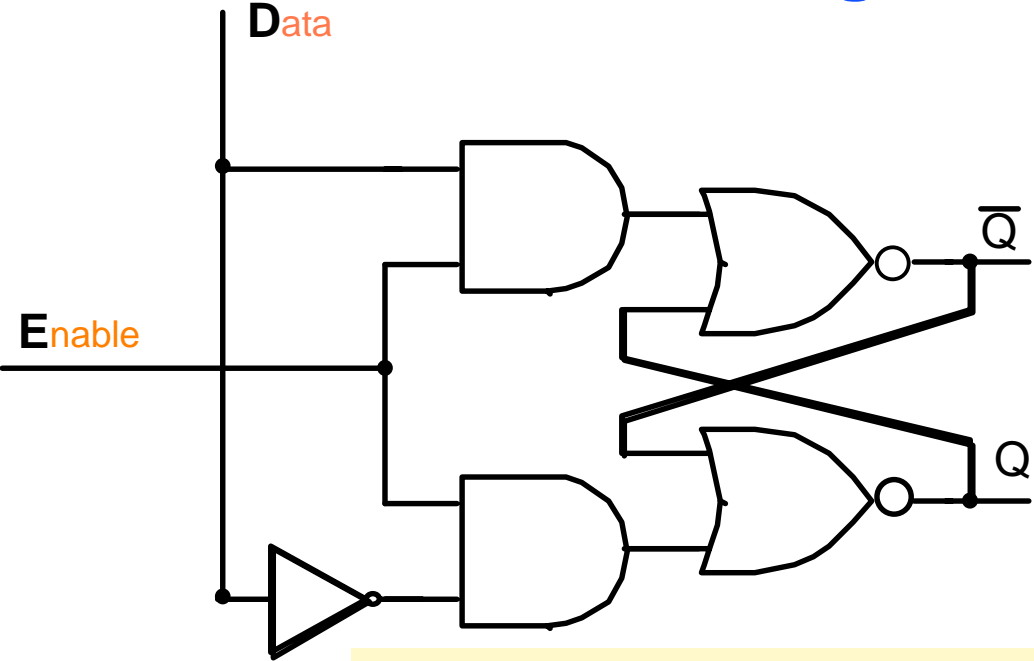
## Rest-Set Latch (cont.)



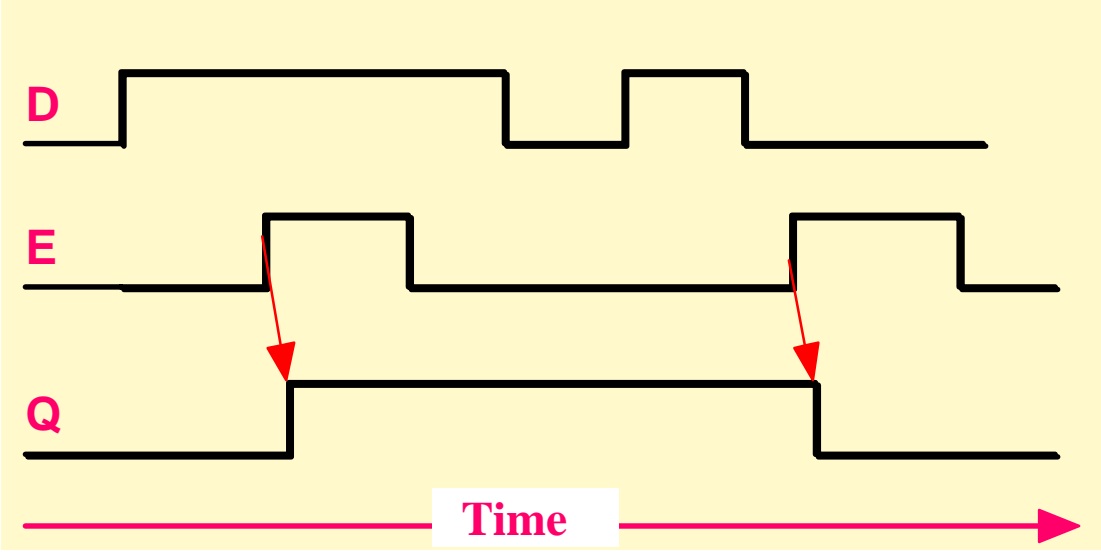
## Reset-Set Latch (cont.)



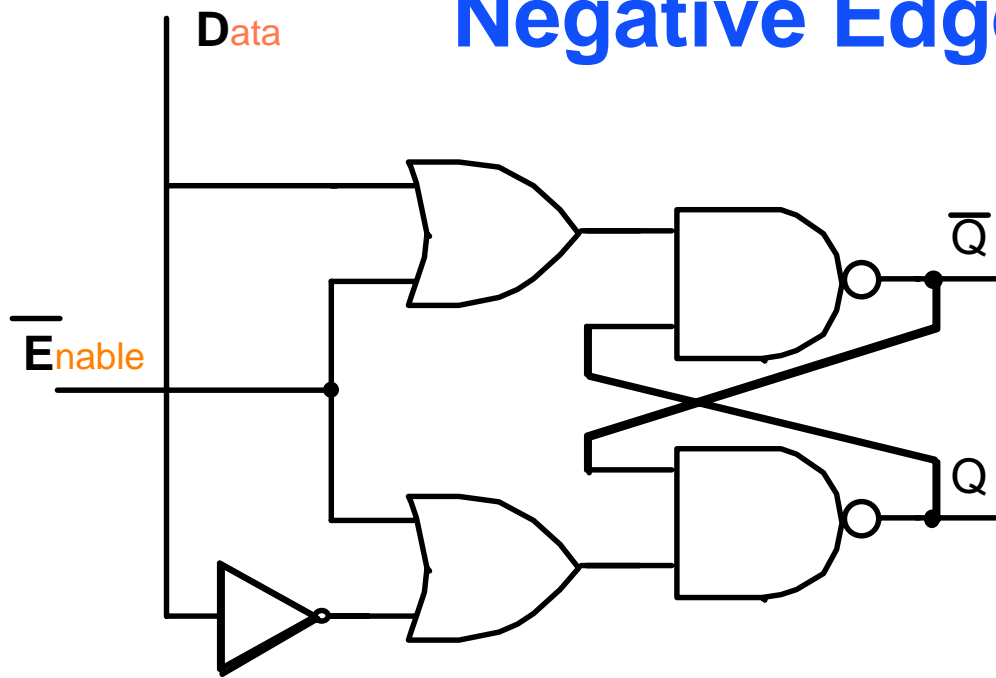
# Positive Edge Data-Latch



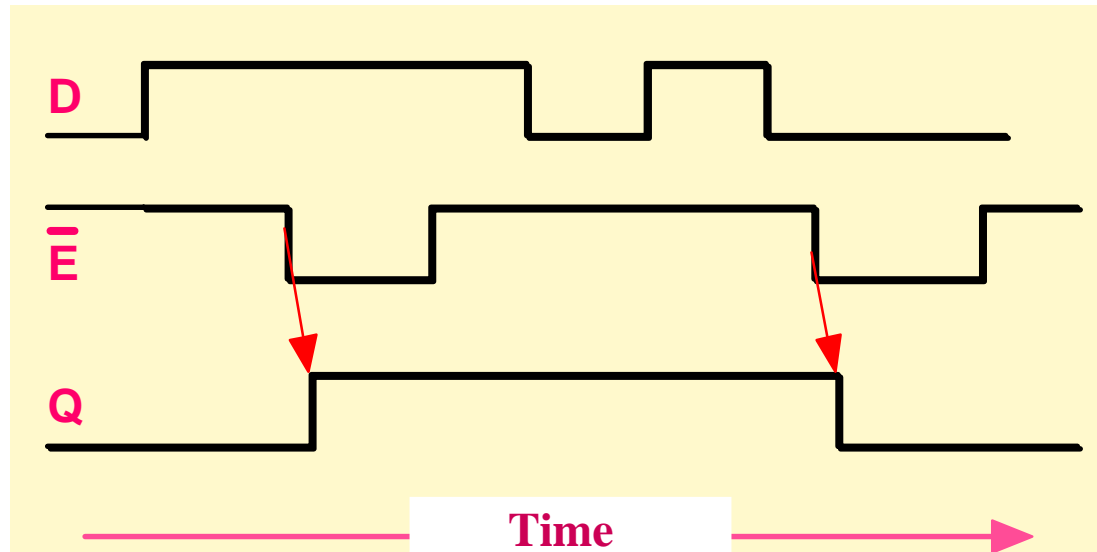
D	E	Q
0	1	0
1	1	1
-	0	Q



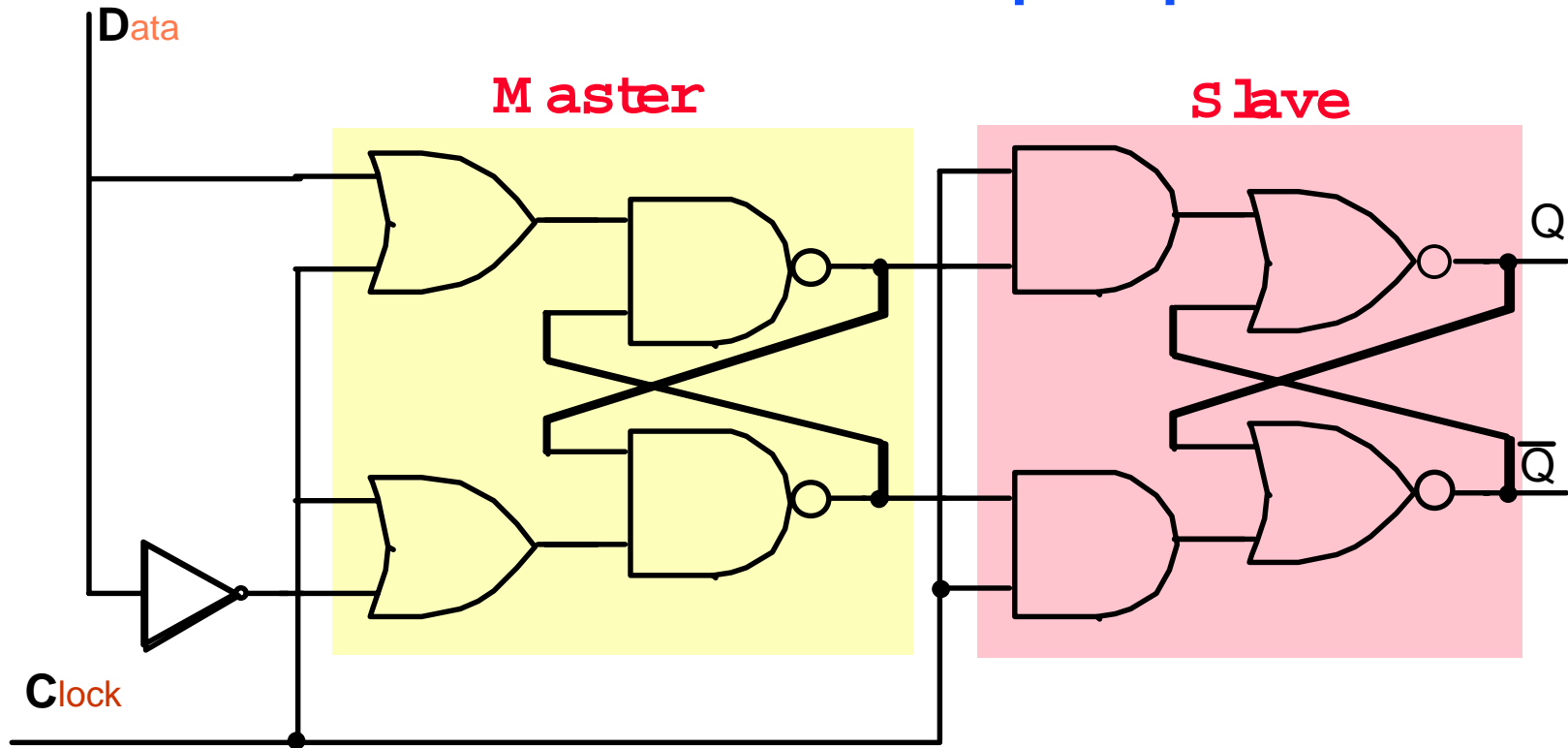
# Negative Edge D-Latch



D	E	Q
0	1	0
1	1	1
-	0	Q

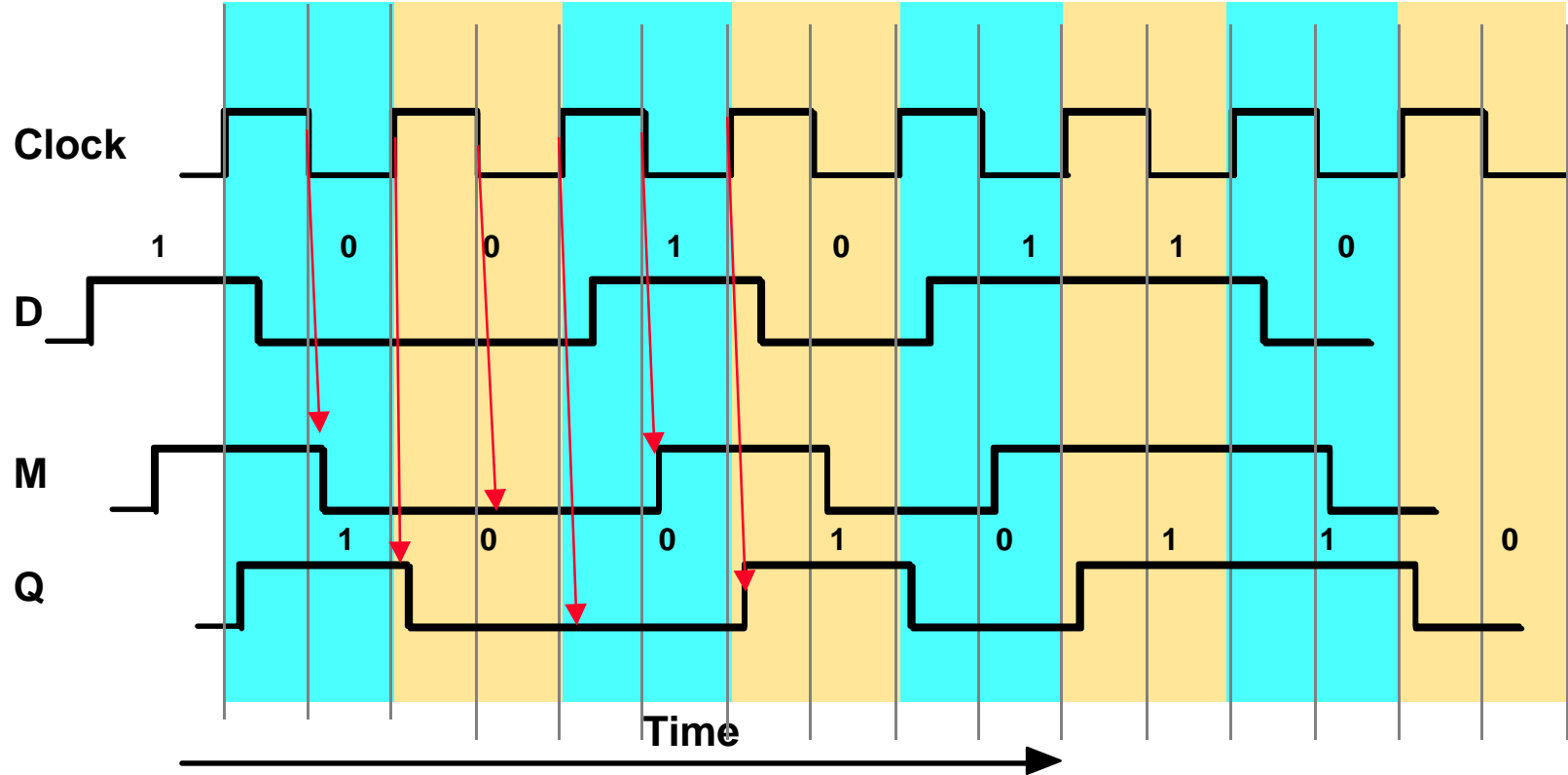
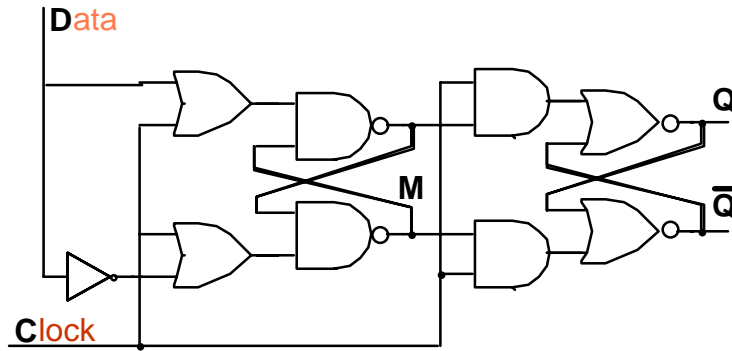


# Master-Slave Data-Flip-Flop



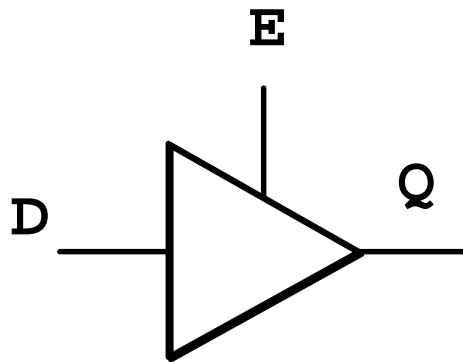
- On  $C \downarrow$  D is transferred to the master stage and the slave is stable.
- On  $C \uparrow$  the Master stage is transferred into the slave stage (output), and the master stage is stable.

# DFF Timing



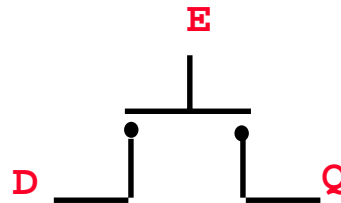
# Tri-State Driver

- The Tri-State driver is like a (one directional) switch:
  - \* When the Enable is on ( $E=1$ ) it transfers the input to the output.
  - \* When the Enable is off ( $E=0$ ) it disconnects the output.



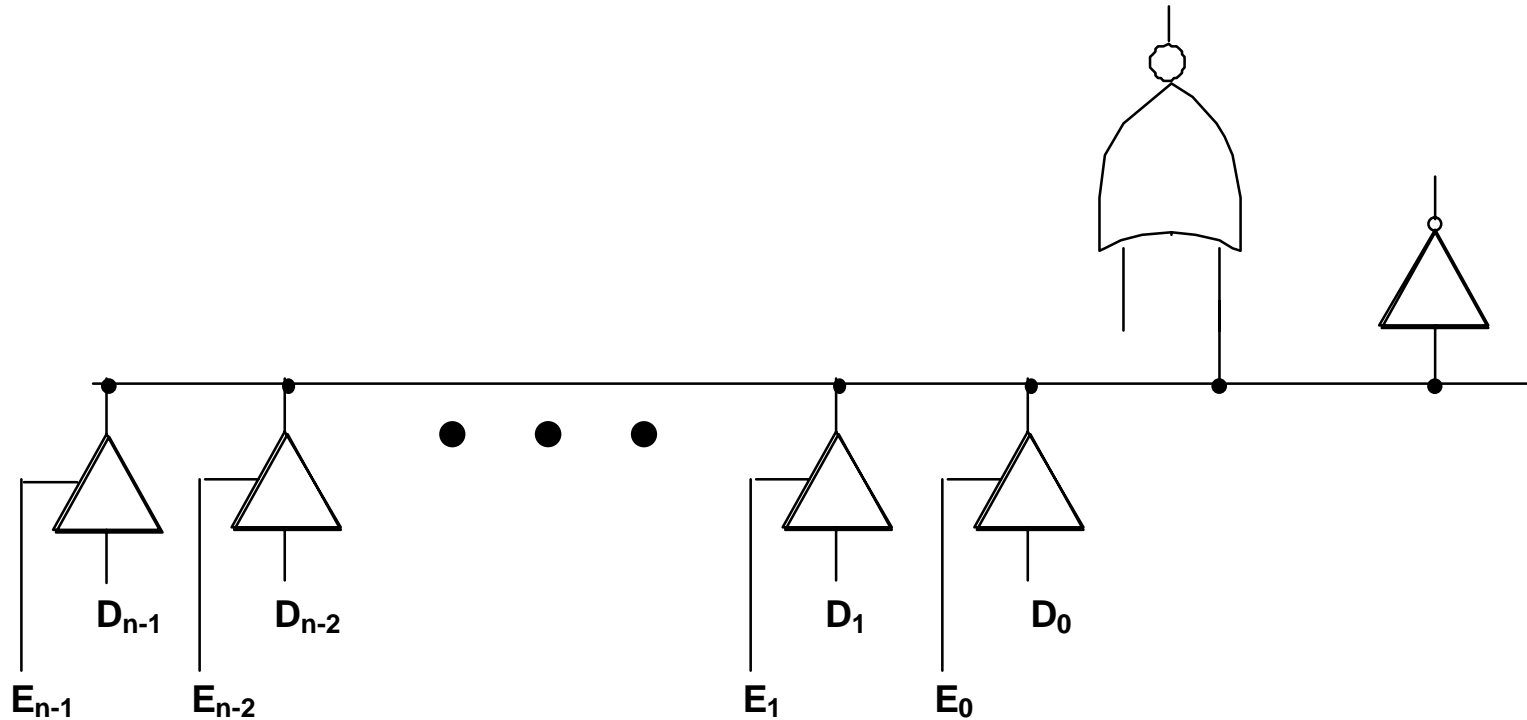
D	E	Q
0	1	0
1	1	1
-	0	Z

**Z :- High Impedance**

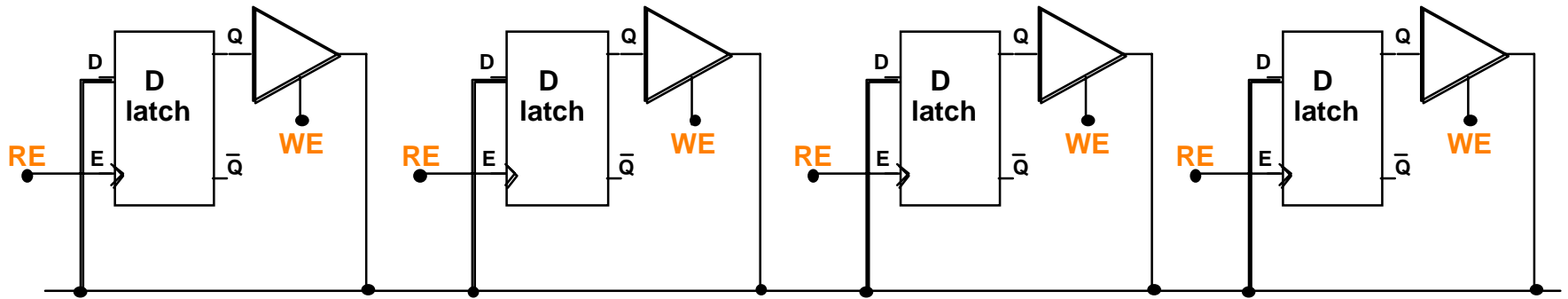


# Bus Connections

- The Bus: Many to many connections.
- Mutual exclusion: **At most one Enable is on!**

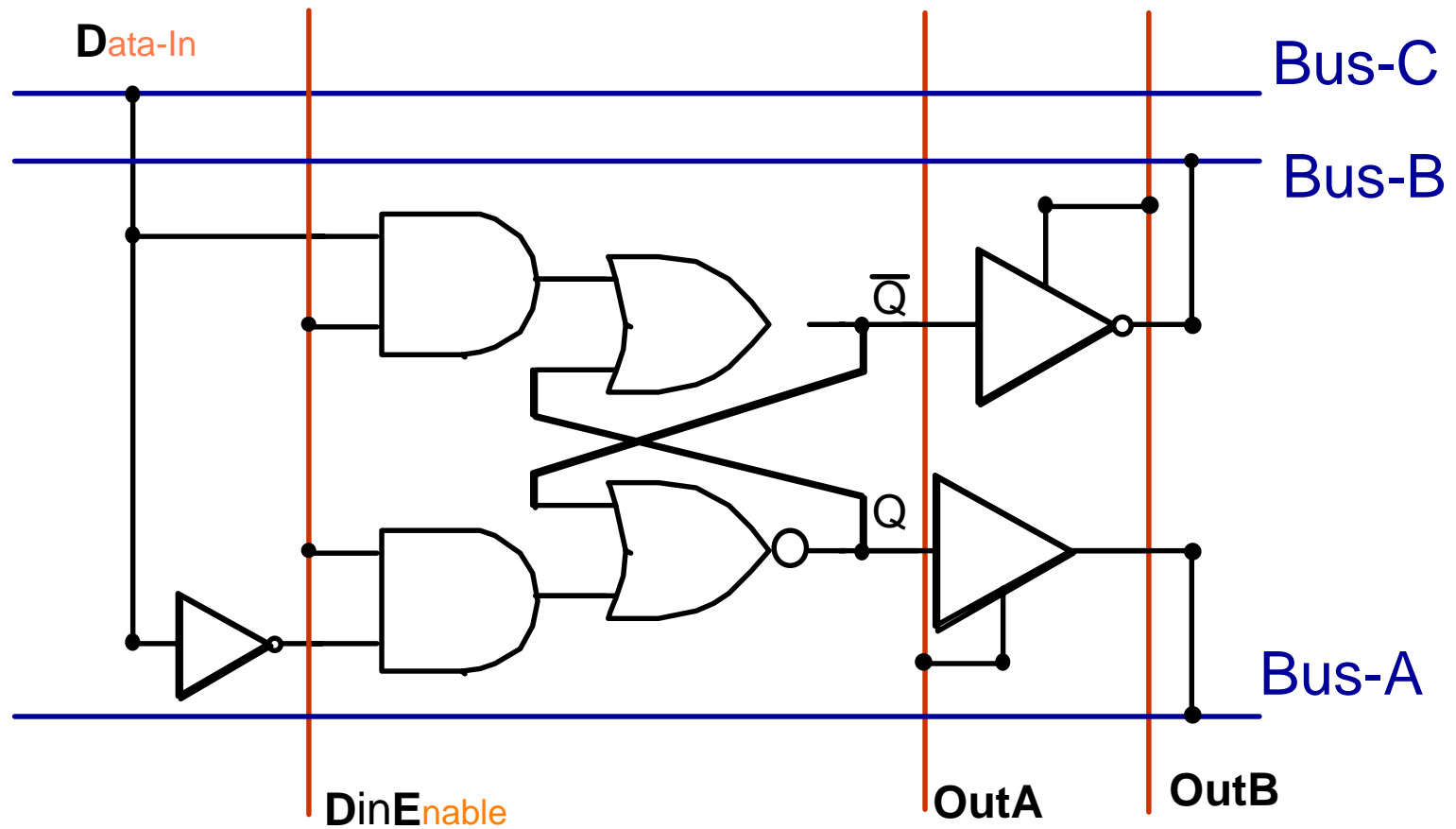


## Register Cells on a bus

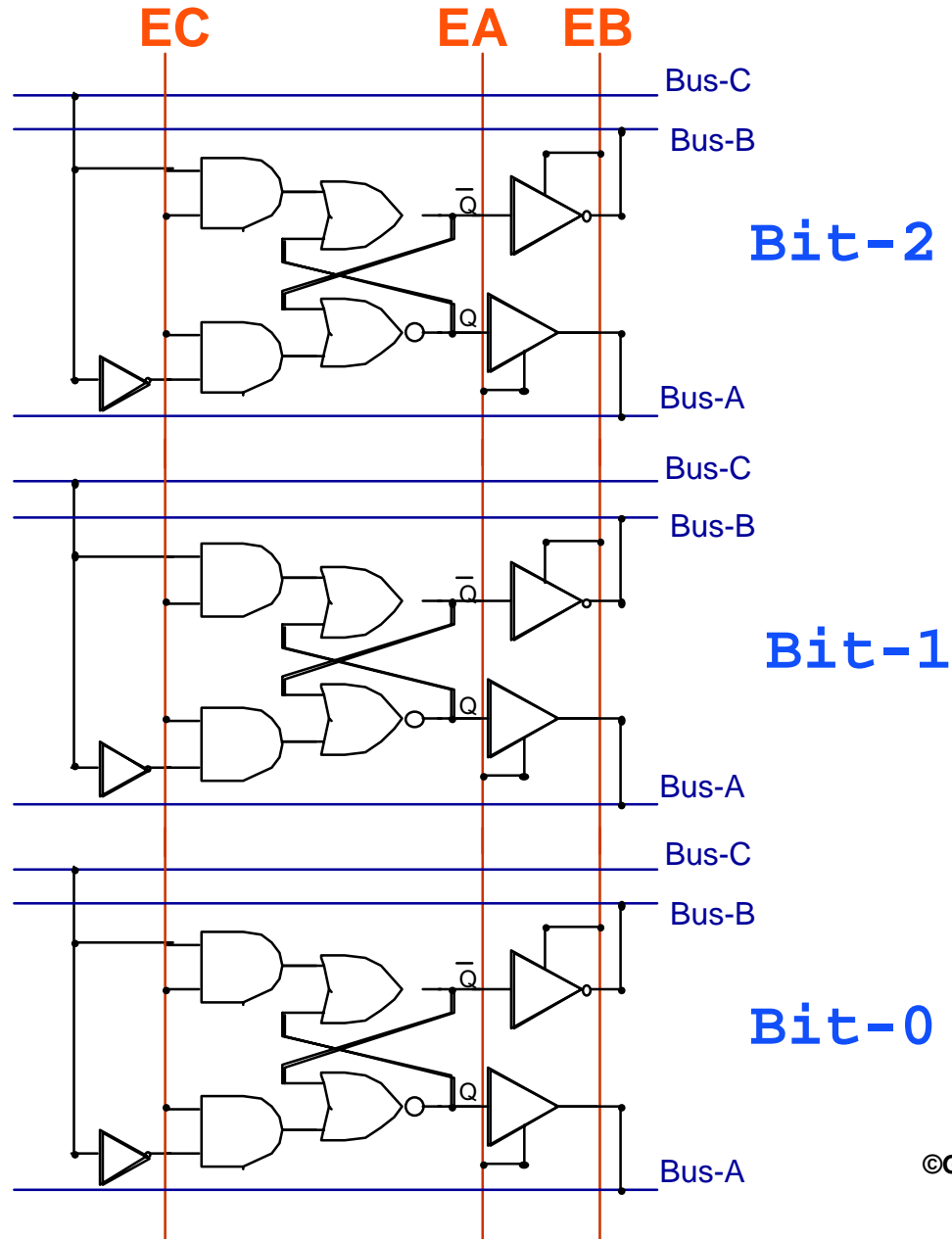


One can “source” and “sink” from any cell on the bus by activating the right controls (**WE** and **RE**).

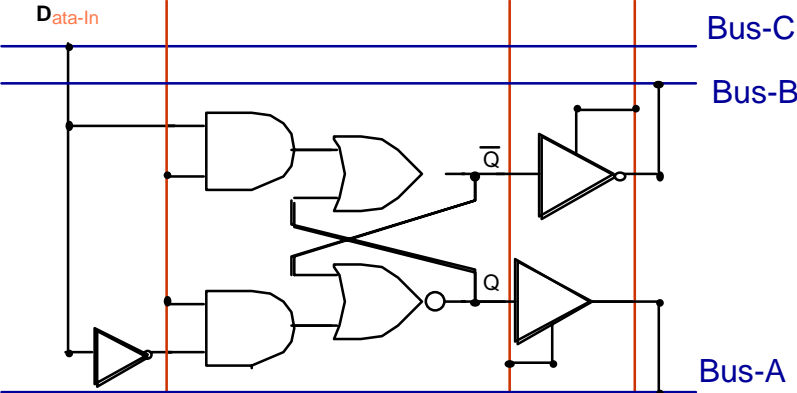
# 3-Port Register Cell



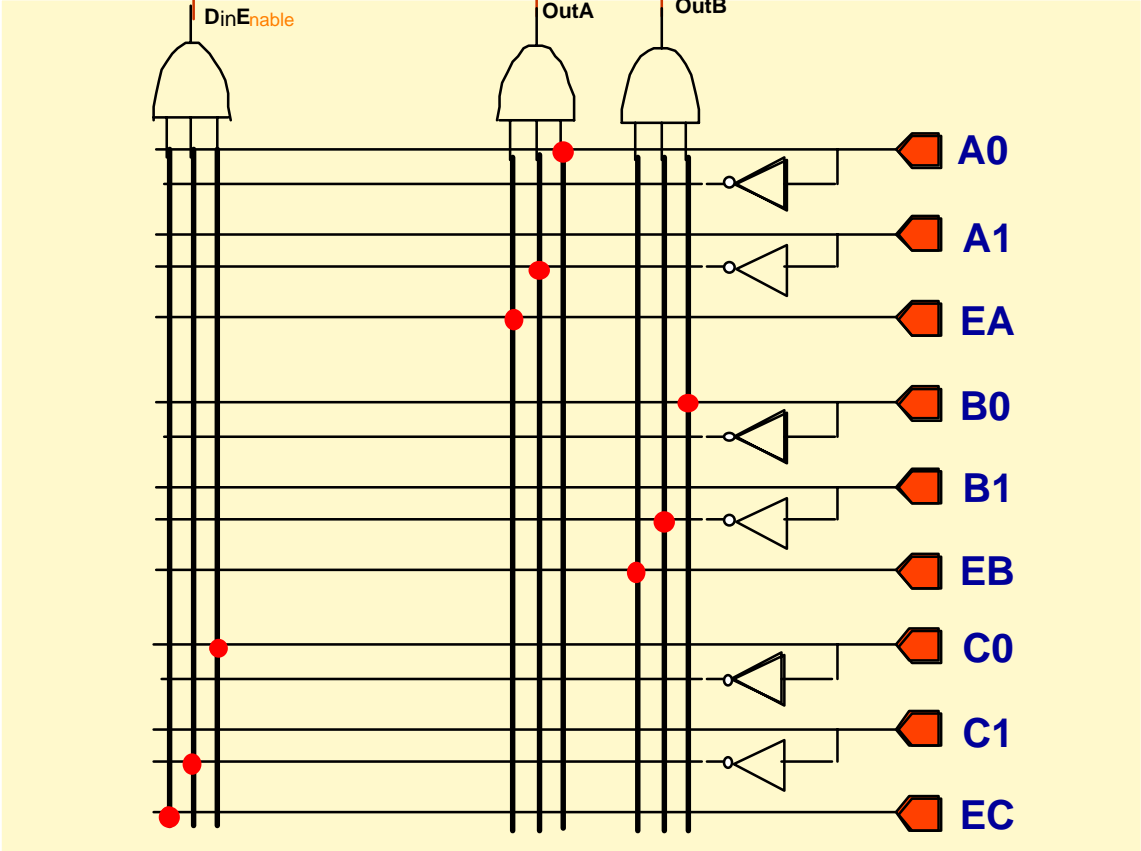
# 3-Port Register



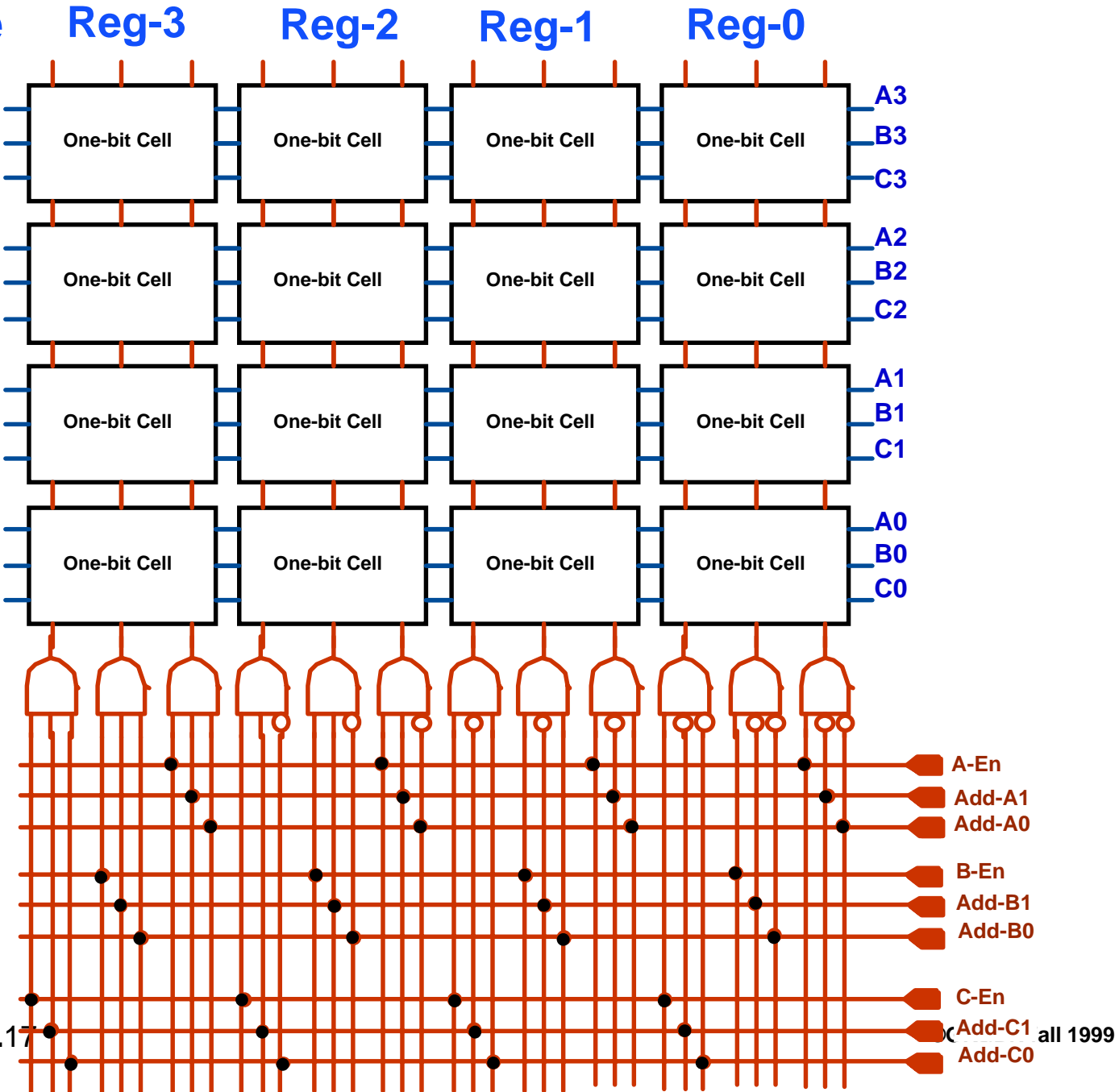
# Address Decode circuit



Register address: 01



# Register File



## Summary

- So far we saw how to take a Boolean function and generate a circuit that “realize” the function.
- We learned to construct circuits that can **add** and **subtract**.
- We learned about the **ALU**: a circuit that can **add, subtract, detect overflow, compare**, and do **bit-wise operations (AND, OR, NOT)**
- Saw how to construct **a shifter** circuit.
- Learned about the memory elements: **RS-Latch, D-Latches** and **D-Flip-flops**.
- Learned about **Tri-State** drivers and **BUS** Communication. (many-to many)
- Learned about how to construct a **register file**.
- Saw how **control signals** can modify what the circuit will do with **inputs**.
  - \* **Examples:** ALU, Shift, Register read-write, ...

# Integer Multiplication

- **Product = Multiplicand x Multiplier**
- **Example: 0011 x 0101**

Multiplicand	0 0 1 1
Multiplier	0 1 0 1
	<hr/>
	0 0 1 1
	0 0 0 0 0
	0 0 1 1 0 0
	0 0 0 0 0 0 0
	<hr/>
Product	0 0 0 1 1 1 1

# Multiplication Algorithm #1

- From Right-Left:
  - \* If multiplier digit = 1: add (shifted) copy of multiplicand to result.
  - \* If multiplier digit = 0: add 0 to result.
- 32 steps when multiplier is 32-bit number.
- Example:  $3_{10} \times 5_{10}$  or  $0011_2 \times 0101_2$   
Product =  $00001111_2$

# Multiplication Algorithm #1

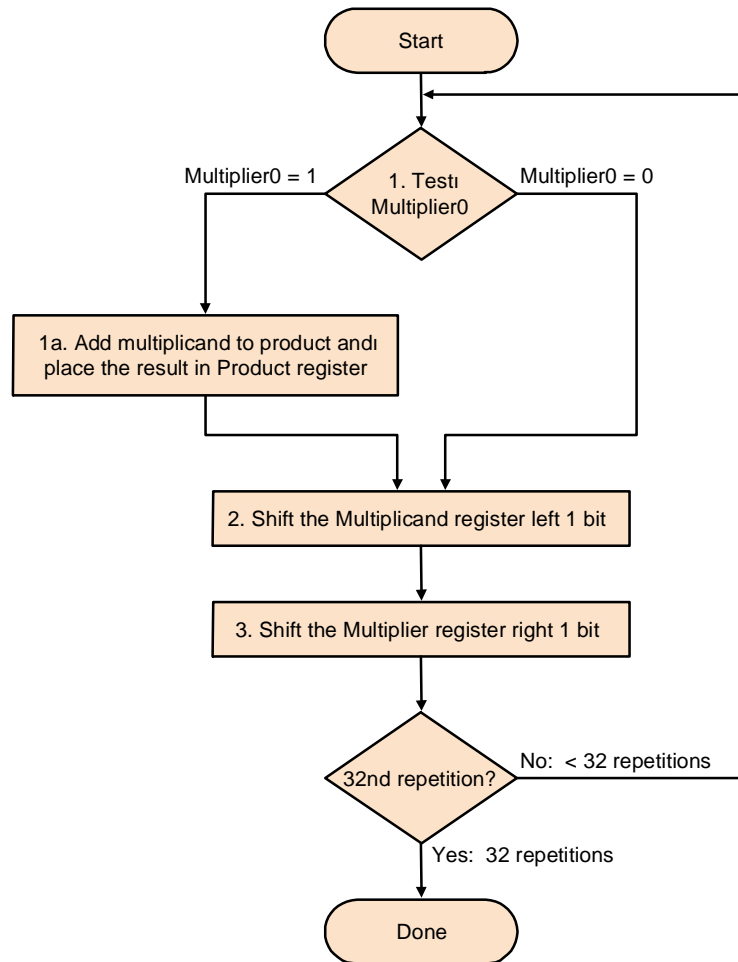


Figure Copyright Morgan Kaufmann

# Multiplication Hardware #1

- Multiplicand starts in right half of register

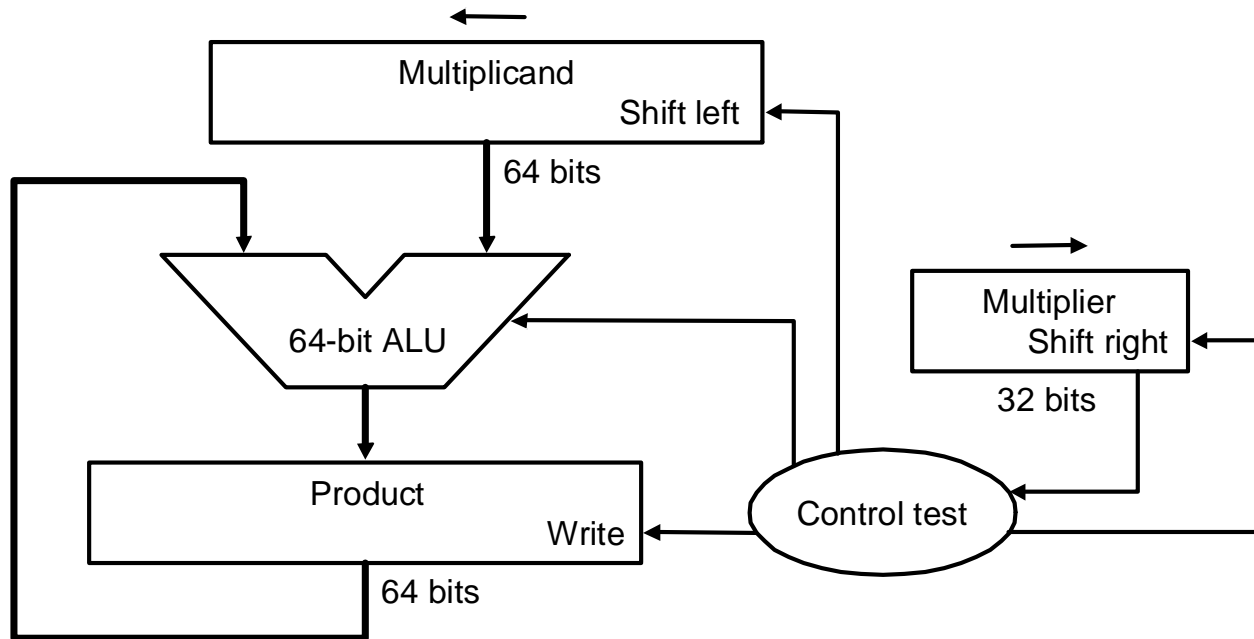
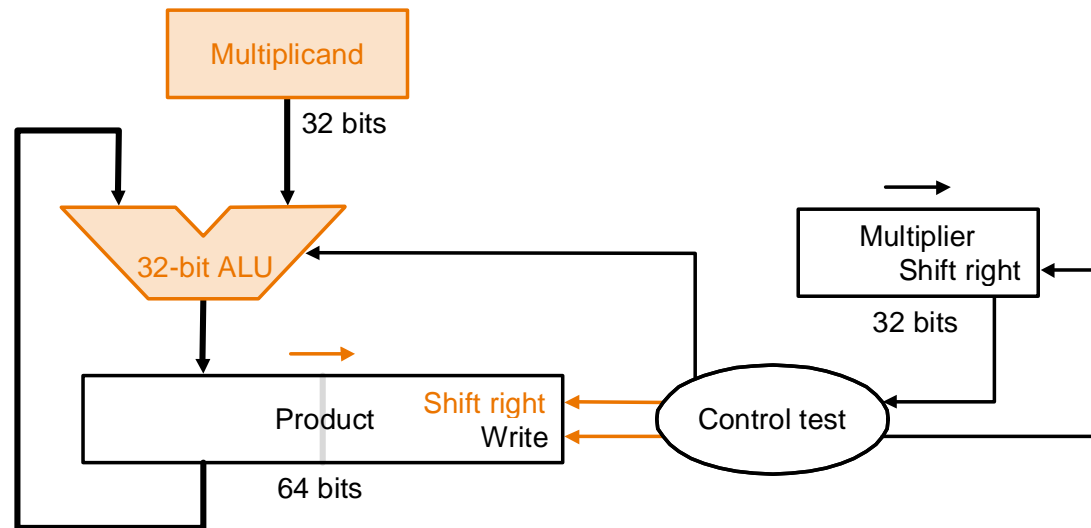


Figure Copyright Morgan Kaufmann

## Multiplication Hardware #2

- Shift Multiplicand Left ~ Shift Product Right
- Only need 32 bits for multiplicand



- Possible to combine multiplier and product registers

Figure Copyright Morgan Kaufmann

# Multiplication Algorithm #2

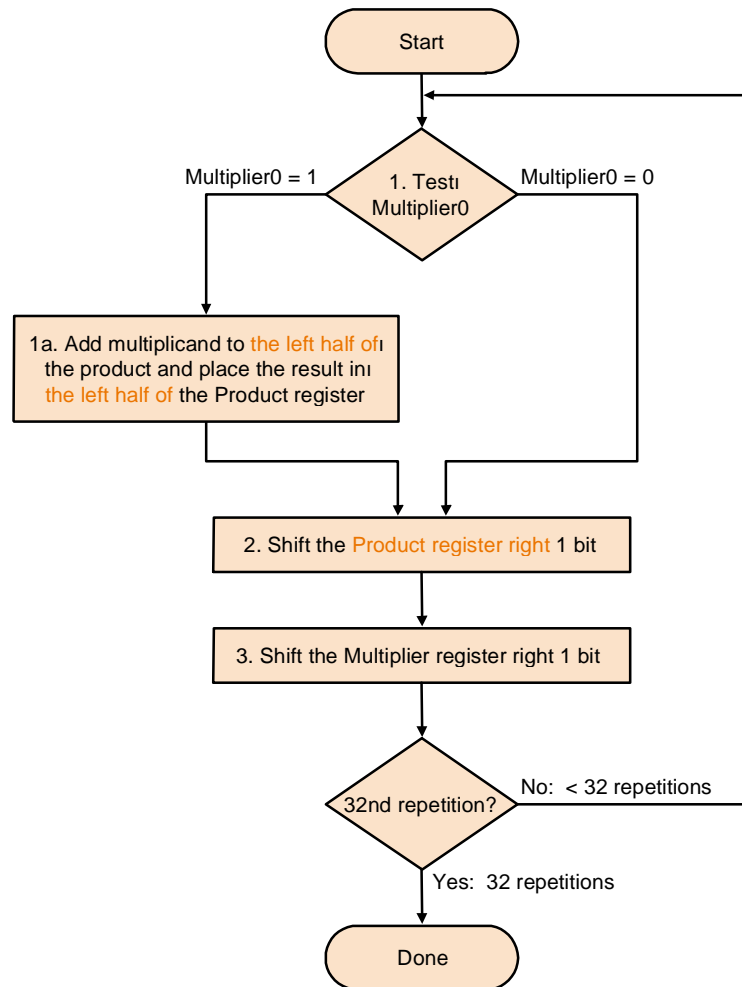


Figure Copyright Morgan Kaufmann

# Booth Encoding

- **Observation:**
  - \* Can write number as difference of two numbers.
  - \* In particular: Can replace a string of 1s with initial subtract when we see a 1, and then an add when we see the bit AFTER the last 1
- **Example 1:  $7_{10}$** 
  - \*  $7_{10} = -1_{10} + 8_{10}$
  - \*  $0111_2 = -0001_2 + 1000_2$
- **Example 2:  $110_{10} = 01101110_2$** 
  - \*  $110_{10} = (-2_{10} + 16_{10}) + (-32_{10} + 128_{10})$
  - \*  $01101110_2 = (-00000010_2 + 00010000_2) + (-00100000_2 + 10000000_2)$
- **Works for signed numbers as well!**

## Booth's Algorithm

- Similar to previous multiply algorithm.
- (Current, Previous) bits of Multiplier:
  - \* 0,0: middle of string of 0s; do nothing
  - \* 0,1: end of a string of 1s; add multiplicand
  - \* 1,0; start of string of 1s; subtract multiplicand
  - \* 1,1: middle of string of 1s; do nothing
- Shift Product/Multiplier right 1 bit (as before)

## Signed Multiplication

- **Convert negative numbers to positive and remember the original signs.**
- **In 2s-complement, can multiply directly using Booth's Algorithm.**
  - \* **Sign extend when shifting.**



# Division Hardware #1

- Divisor starts in left half of divisor register

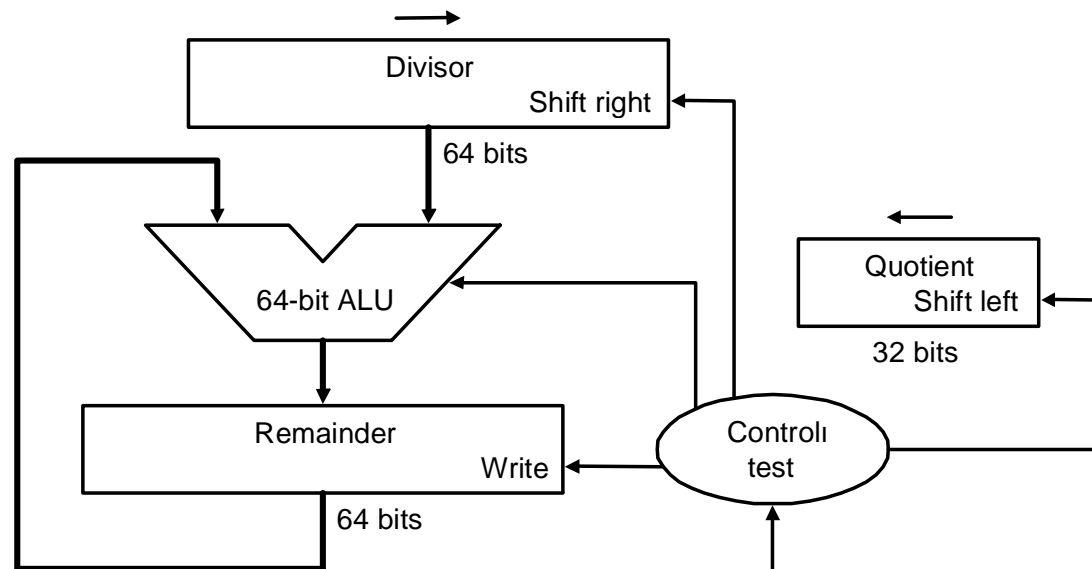


Figure Copyright Morgan Kaufmann

## Division (contd.)

- **Similar to multiplication**
  - \* **Shift remainder left instead of shifting divisor right**
  - \* **Combine quotient register with right half of remainder register**
- **Signed Division**
  - \* **Remember the signs and negate quotient if different.**
  - \* **Make sign of remainder match the dividend**
- **Same hardware can be used for both multiply and divide.**
  - \* **Need 64-bit register that can shift left and right**
  - \* **ALU that adds or subtracts**
  - \* **Optimizations possible**