Multiplication and Division

Agenda

- Multiplication (3.3)
- Division (3.4)

Multiplications

Revisit Integer Multiplication

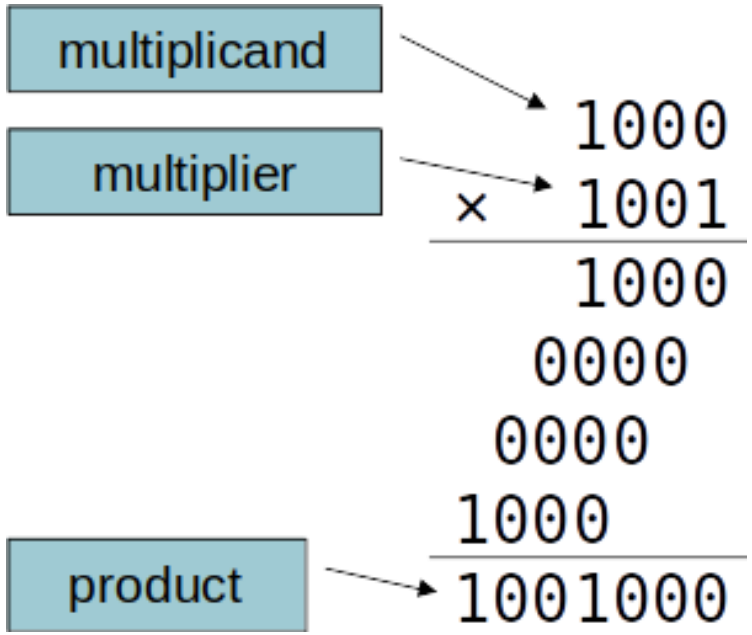
- *P = A x B*
  - *A: Multiplicand*
  - *B: multiplier*
  - *P: product*

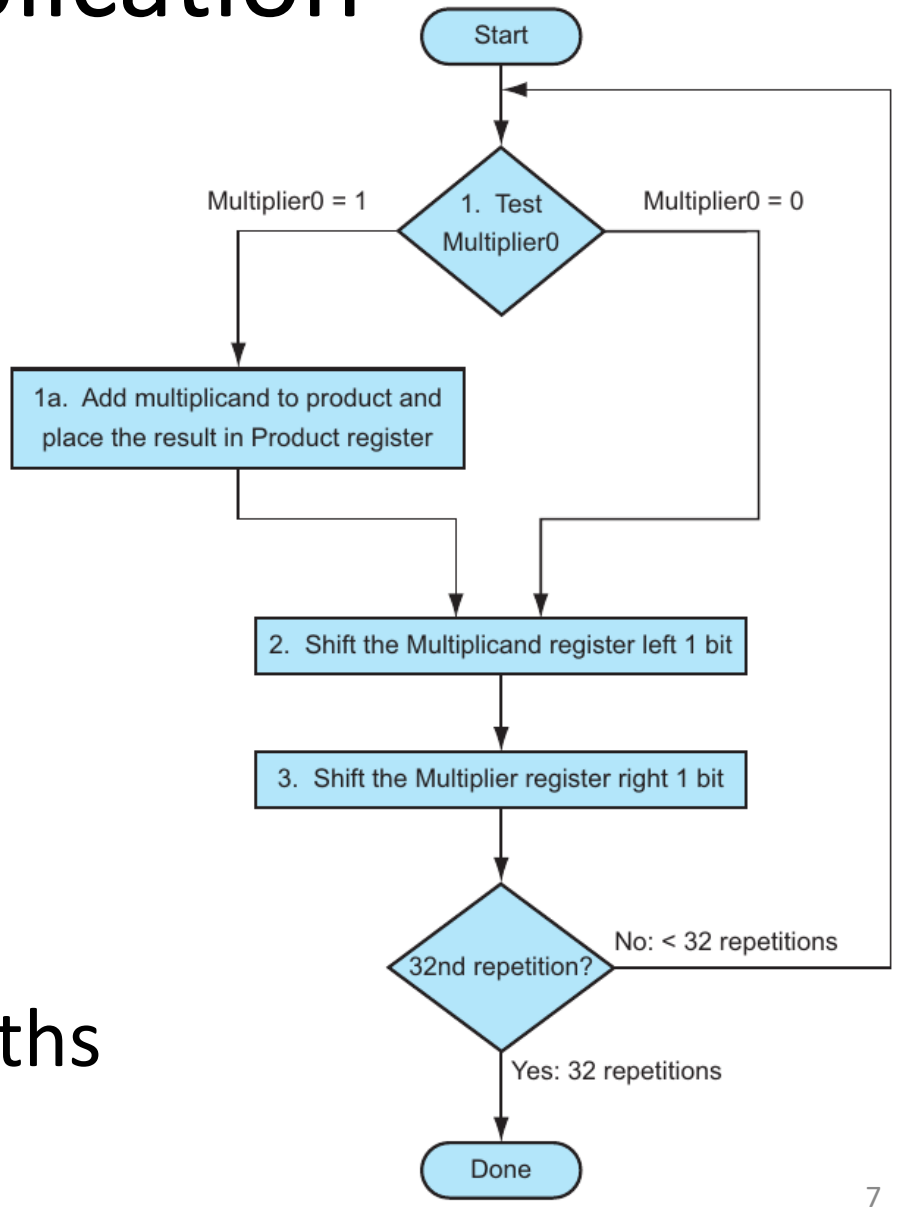
$$\begin{array}{rll} & A_{n-1} \ldots A_1 A_0 & \text{“multiplicand”} \\ \times & B_{n-1} \ldots B_1 B_0 & \text{“multiplier”} \\ \hline P_{2n-1} \ldots & P_{n-1} \ldots P_1 P_0 & \text{“product”} \end{array}$$

- Multiplicand is multiplied by each bit of the multiplier
- Each intermediate result is added up to get the final *product.*

Unsigned Binary Multiplication

- Multiplying each bit is easy because it is either 1 or 0

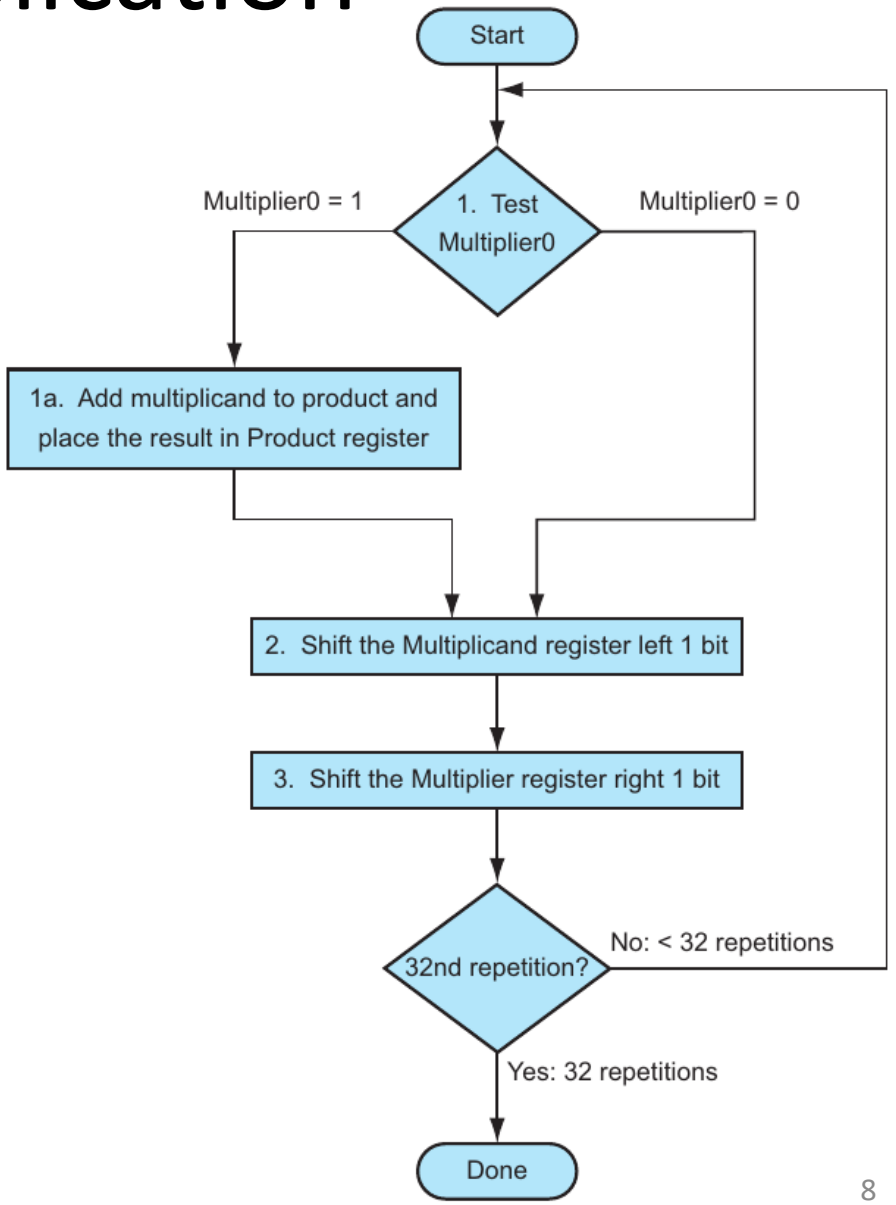




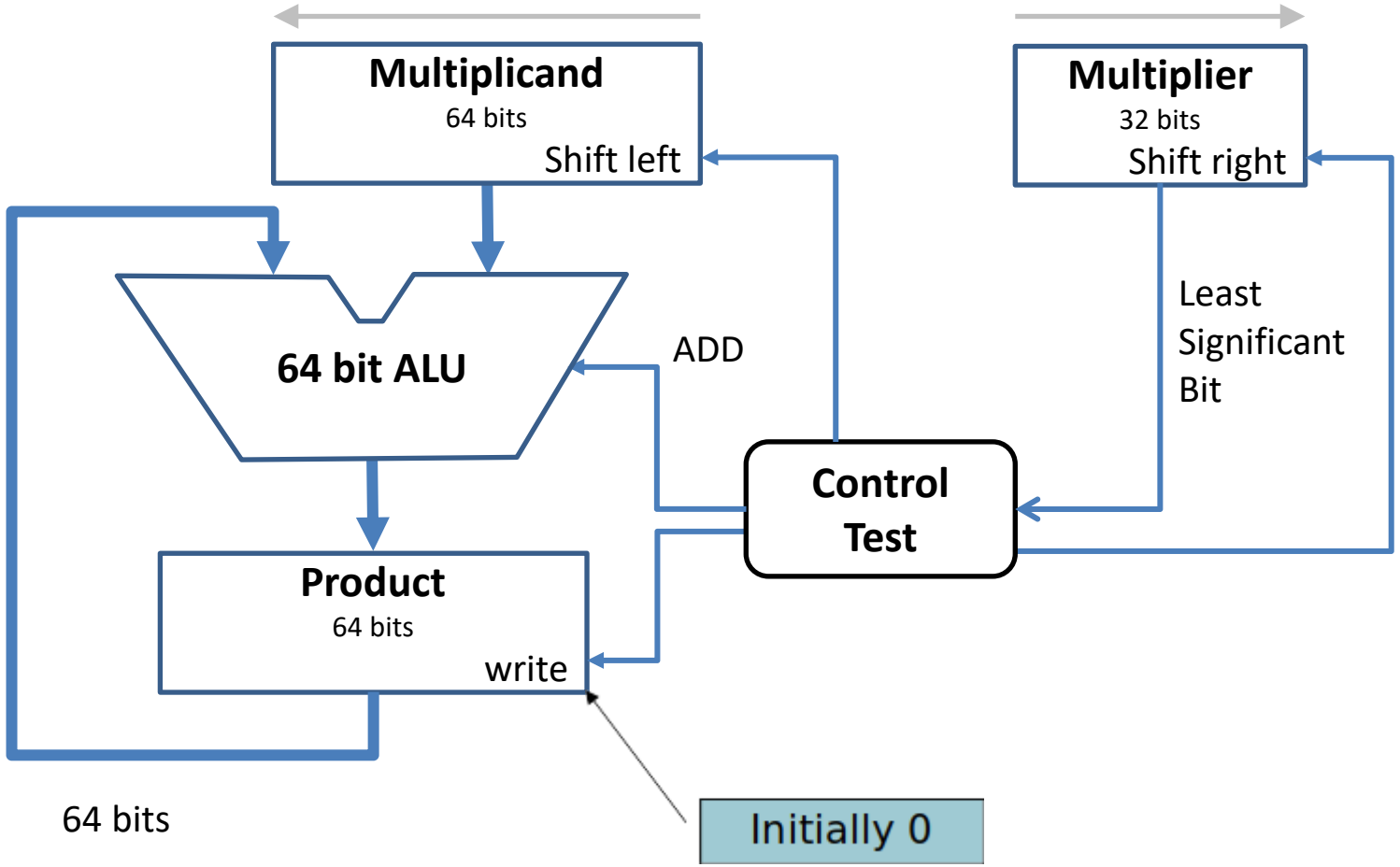
- Length of product = sum of operand lengths

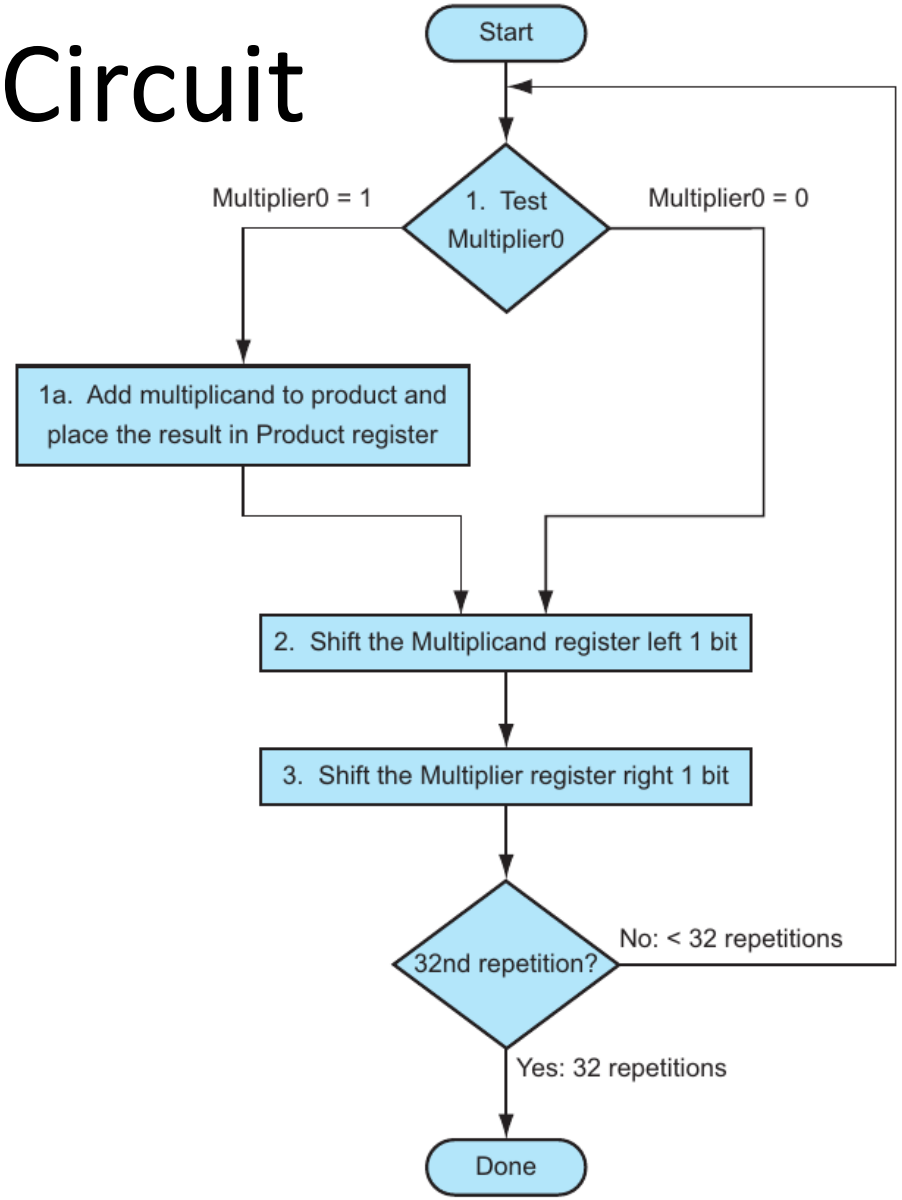
Unsigned Binary Multiplication

- **Multiplicand**: 64 bits shift (left) register
- **Multiplier**: 32-bit shift (right) register to read the LSB on each add
- **Product**: 64-bit register
- **Adder circuit**: 64-bit ALU
- **Counter:** keep track of how many times we shift (in this case can be 5 bit counter register)



Simple Multiplication Circuit





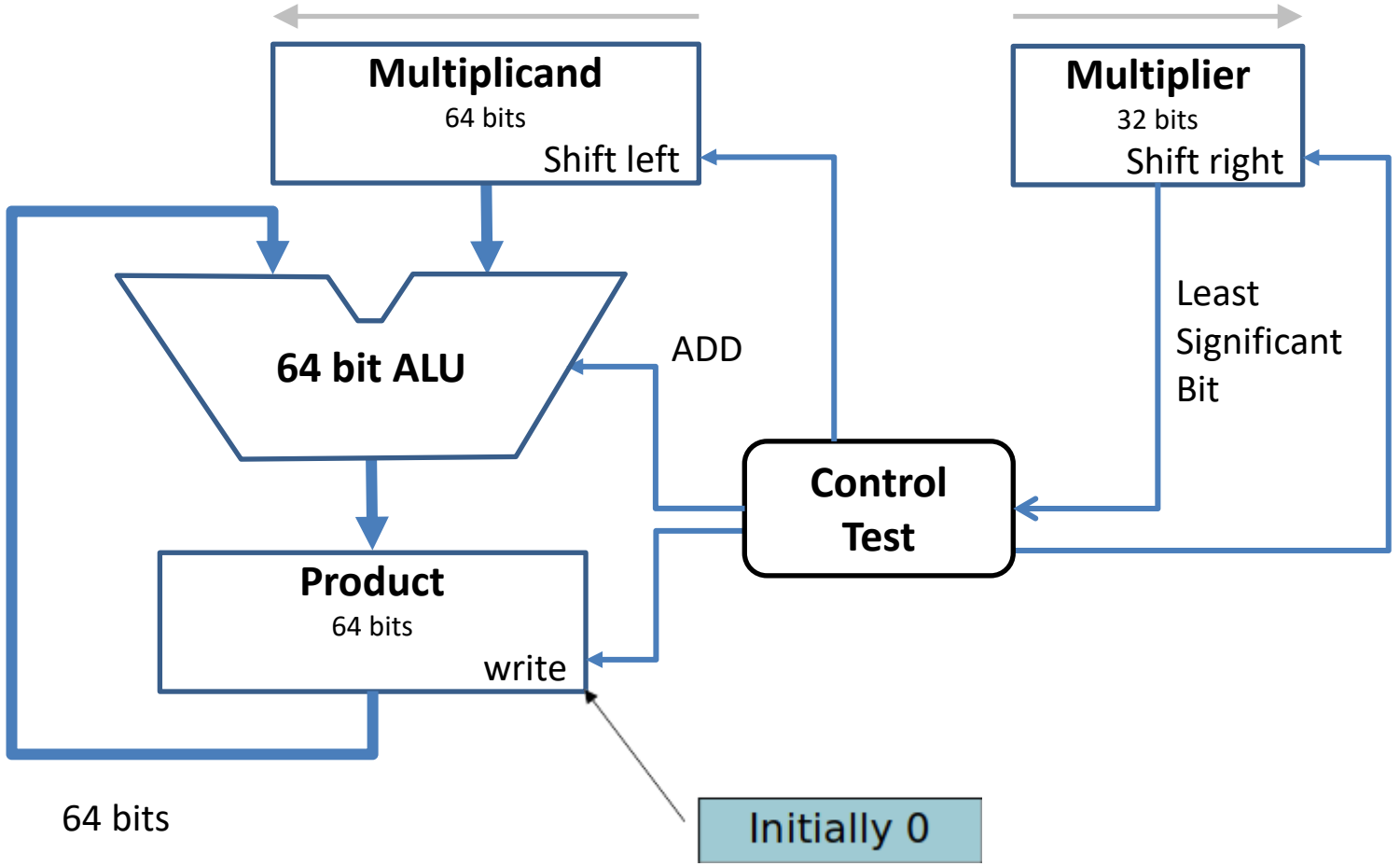
Repeat, shifting 32 times! Good enough? How is this wasteful?

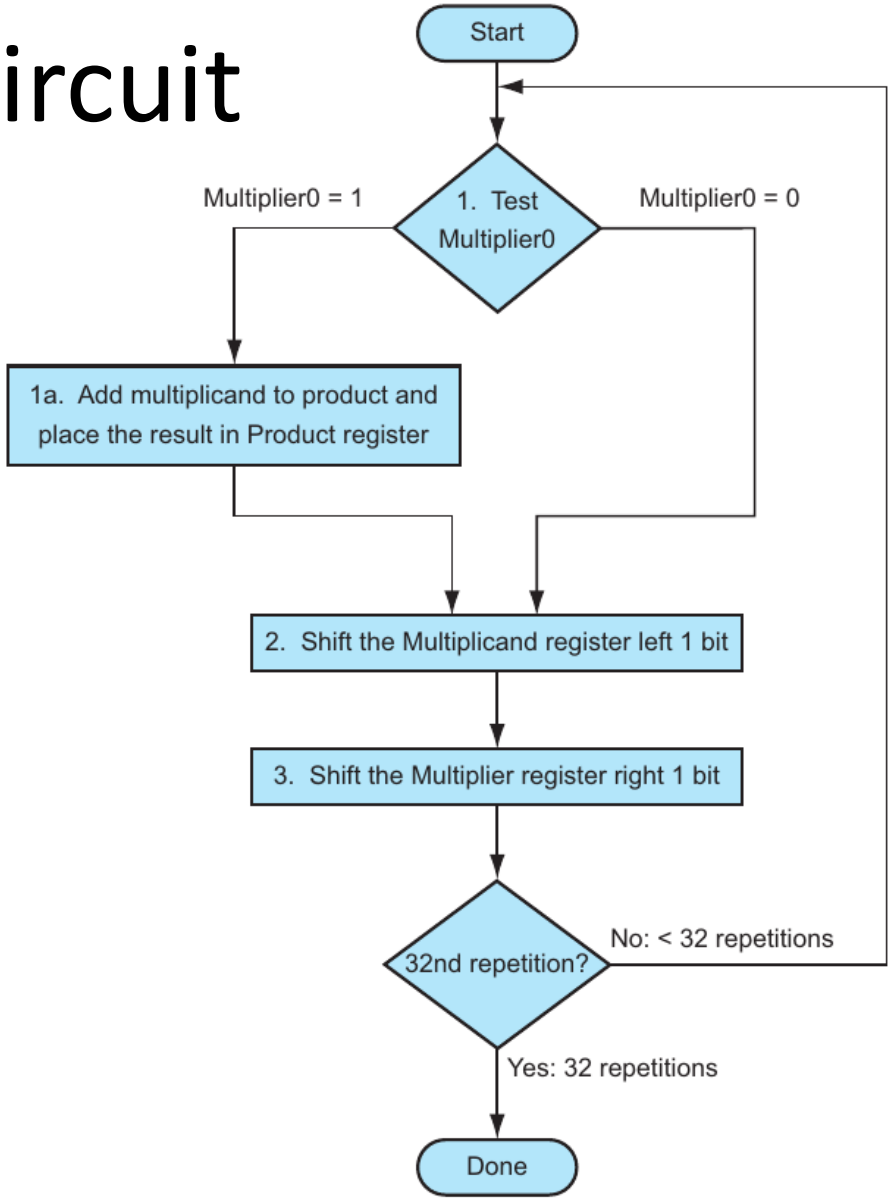
Simple Multiplication Circuit

4-bit Example
Multiplicand: 0010
Multiplier: 0011
Repetition: 4

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⟹ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⟹ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⟹ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⟹ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

Simple Multiplication Circuit





Repeat, shifting 32 times! Good enough? How is this wasteful?

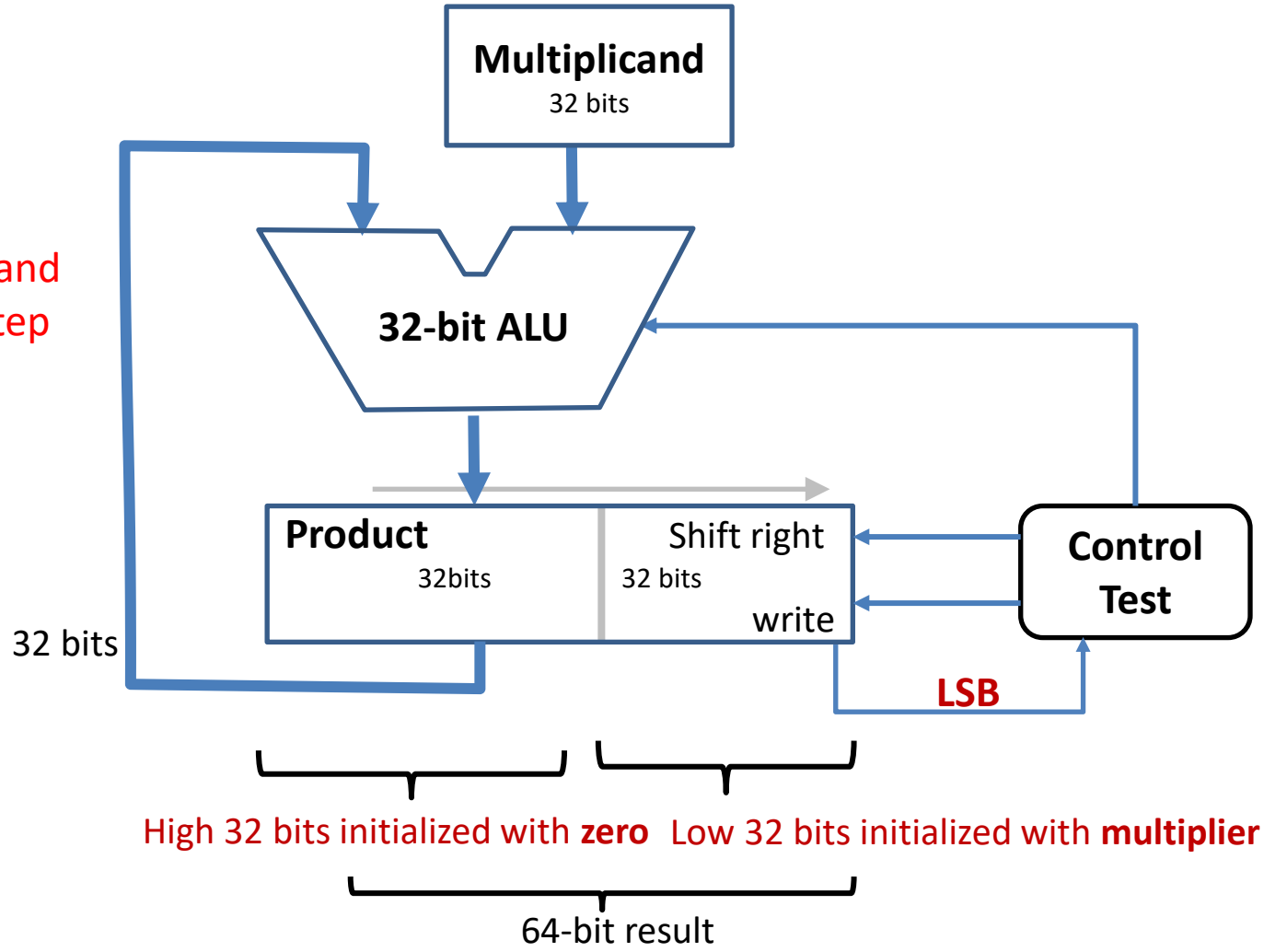
Refined Multiplication Circuit

4-bit Example
Multiplicand: 0010
Multiplier: 0011
Repetition: 4

Note that there is no shift left to the multiplicand as shifting right the product register at each step performs that alignment process.

Iteration	Step	Multiplicand	Product
0	Initial values	0010	0000 0011
1	Product= Product + Multiplicand	0010	0010 0011
	Shift right the product register	0010	0001 0001
2	Product= Product + Multiplicand	0010	0011 0001
	Shift right the product register	0010	0001 1000
3	Shift right the product register	0010	0000 1100
4	Shift right the product register	0010	0000 0110

Product Multiplier



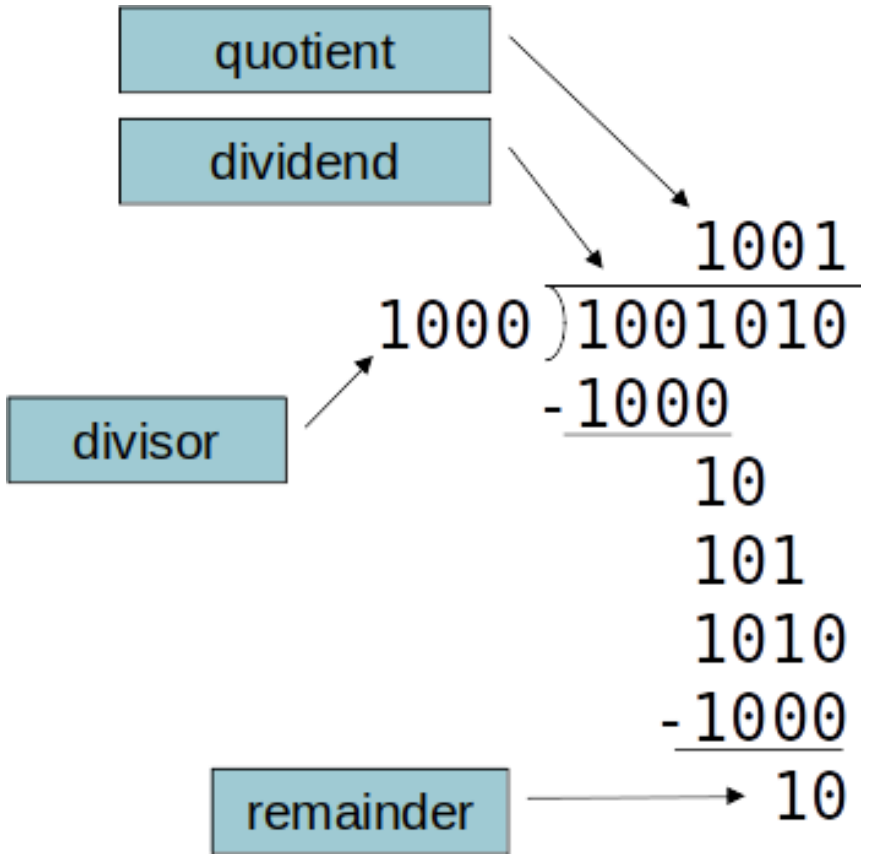
Signed Multiplication

- Convert the multiplier and the multiplicand to positive numbers and remember the original signs
- Multiply positive numbers
- The sign of the product is negative if the signs of the operands were different
- There are faster ways to do multiplication, but outside of scope of this course

Division

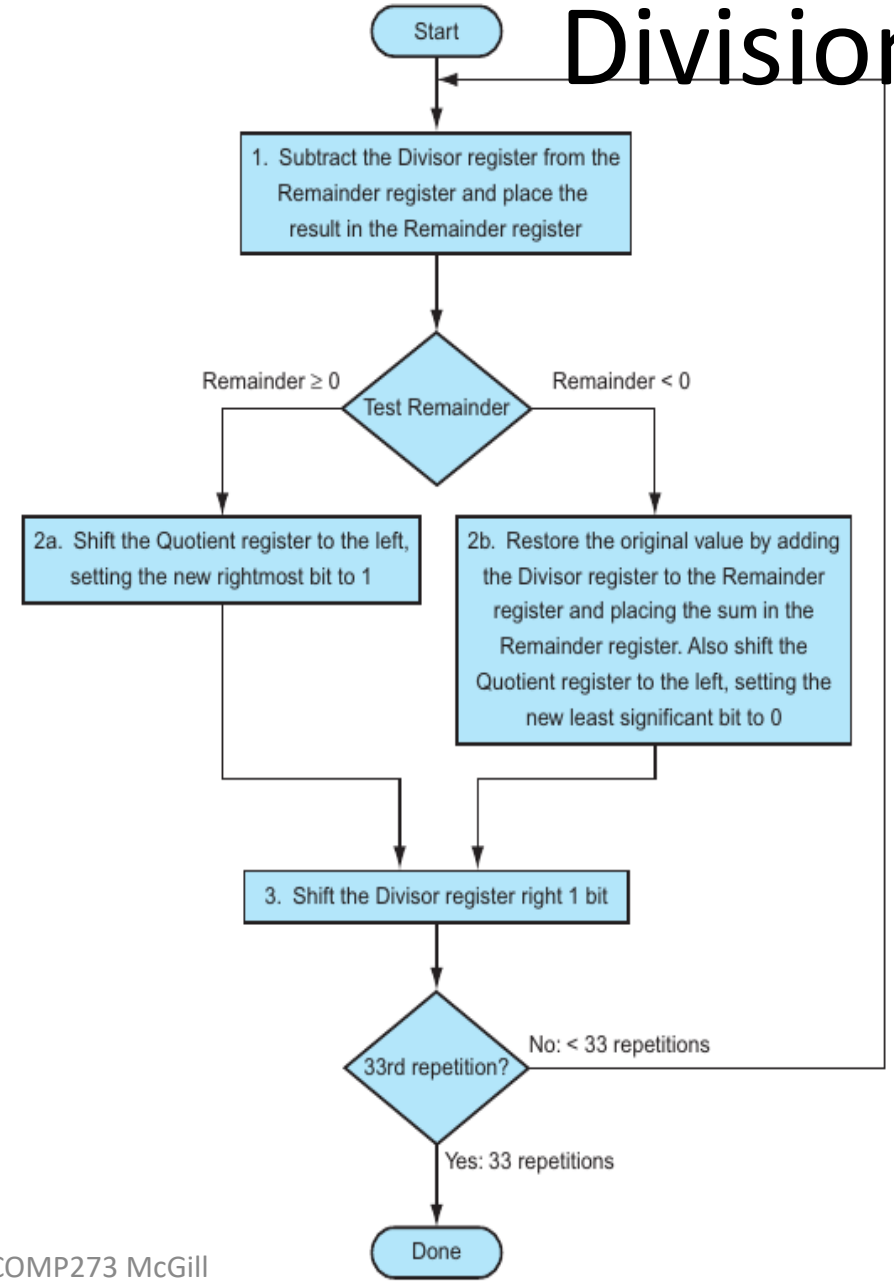
Division for Positive Numbers

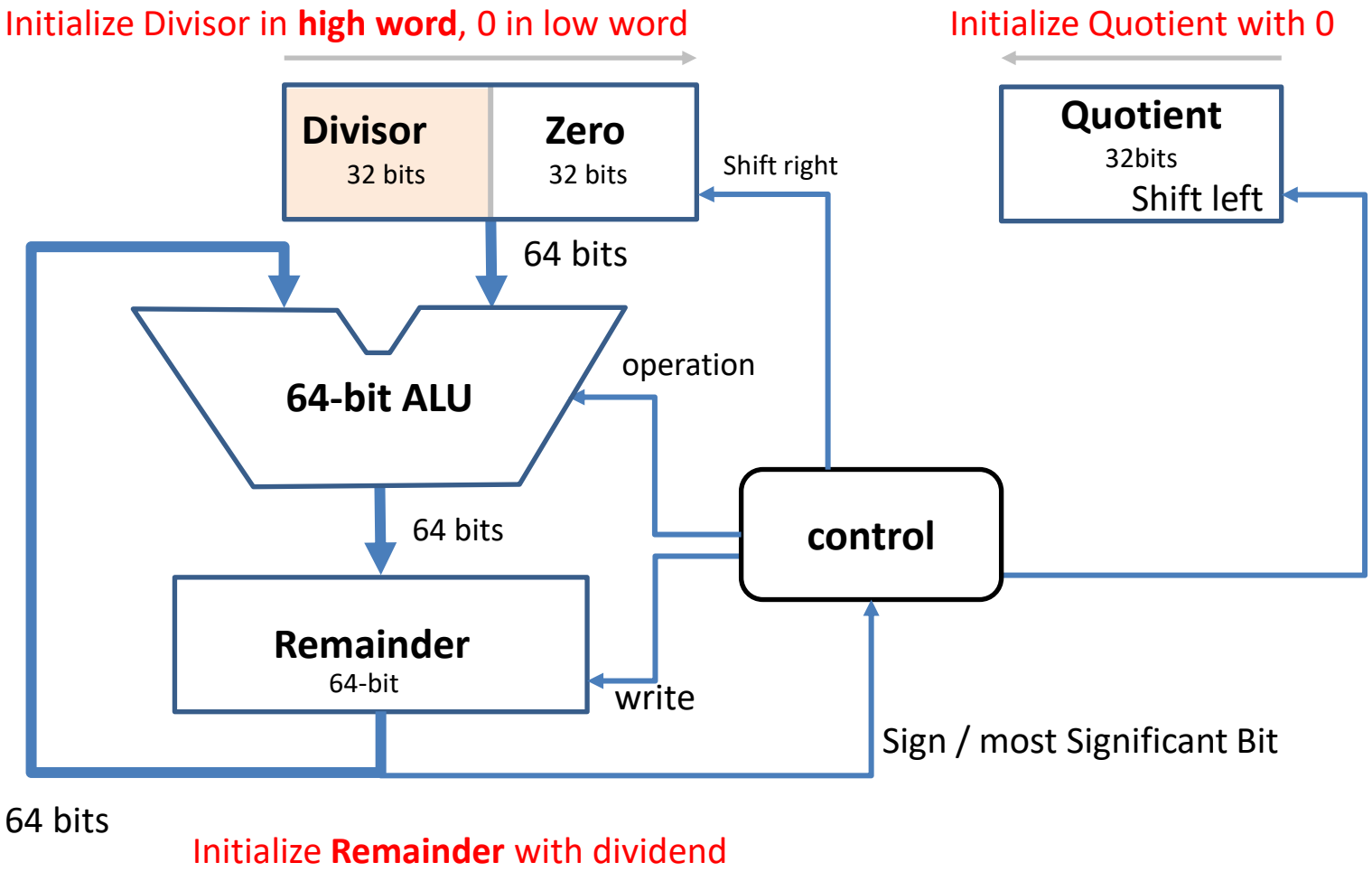
- Binary division is similar, and in many respects easier!
- Approach
  - If divisor <= dividend, 1 in quotient, subtract the divisor from the dividend
  - If divisor > dividend, 0 in quotient, bring down next dividend bit



dividend / divisor = quotient, remainder

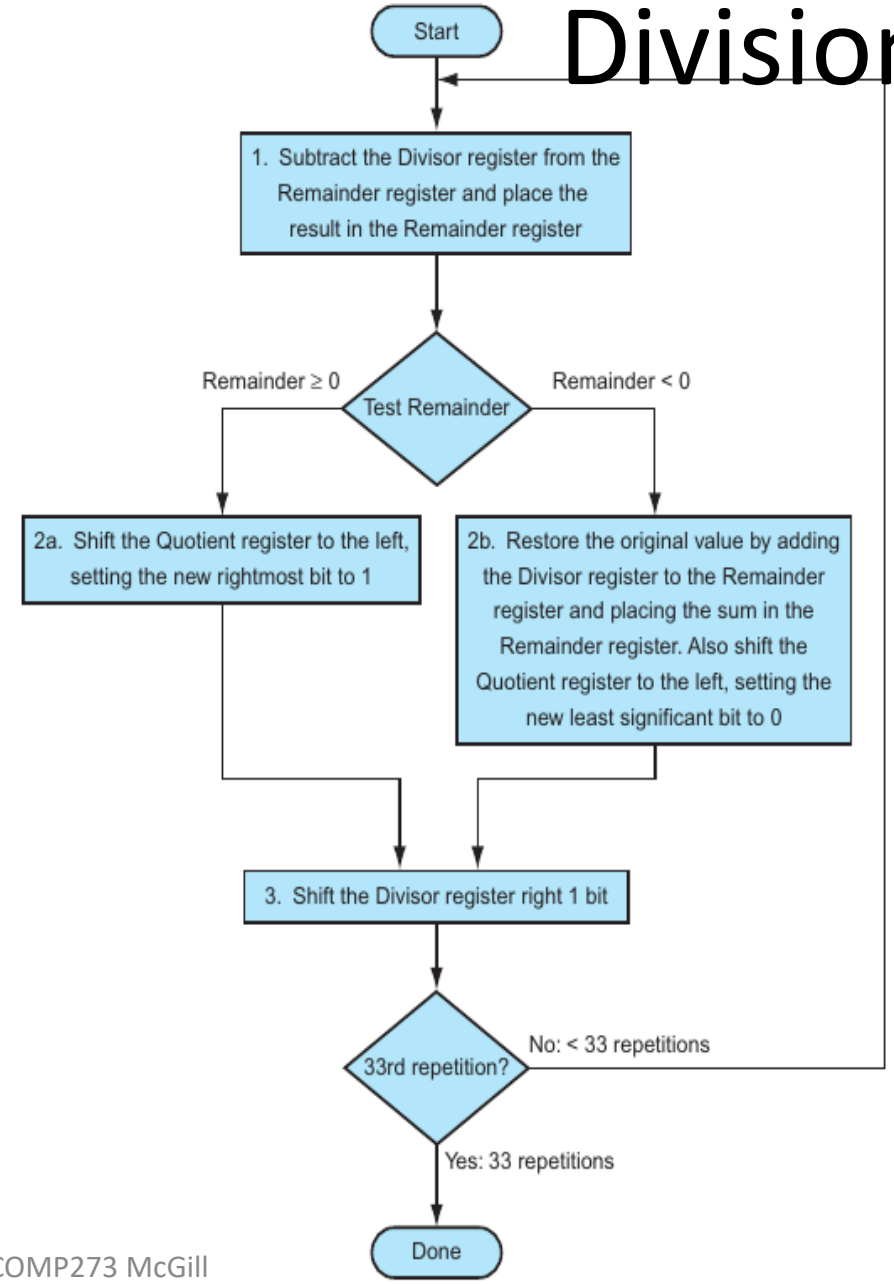
Division for Positive Numbers

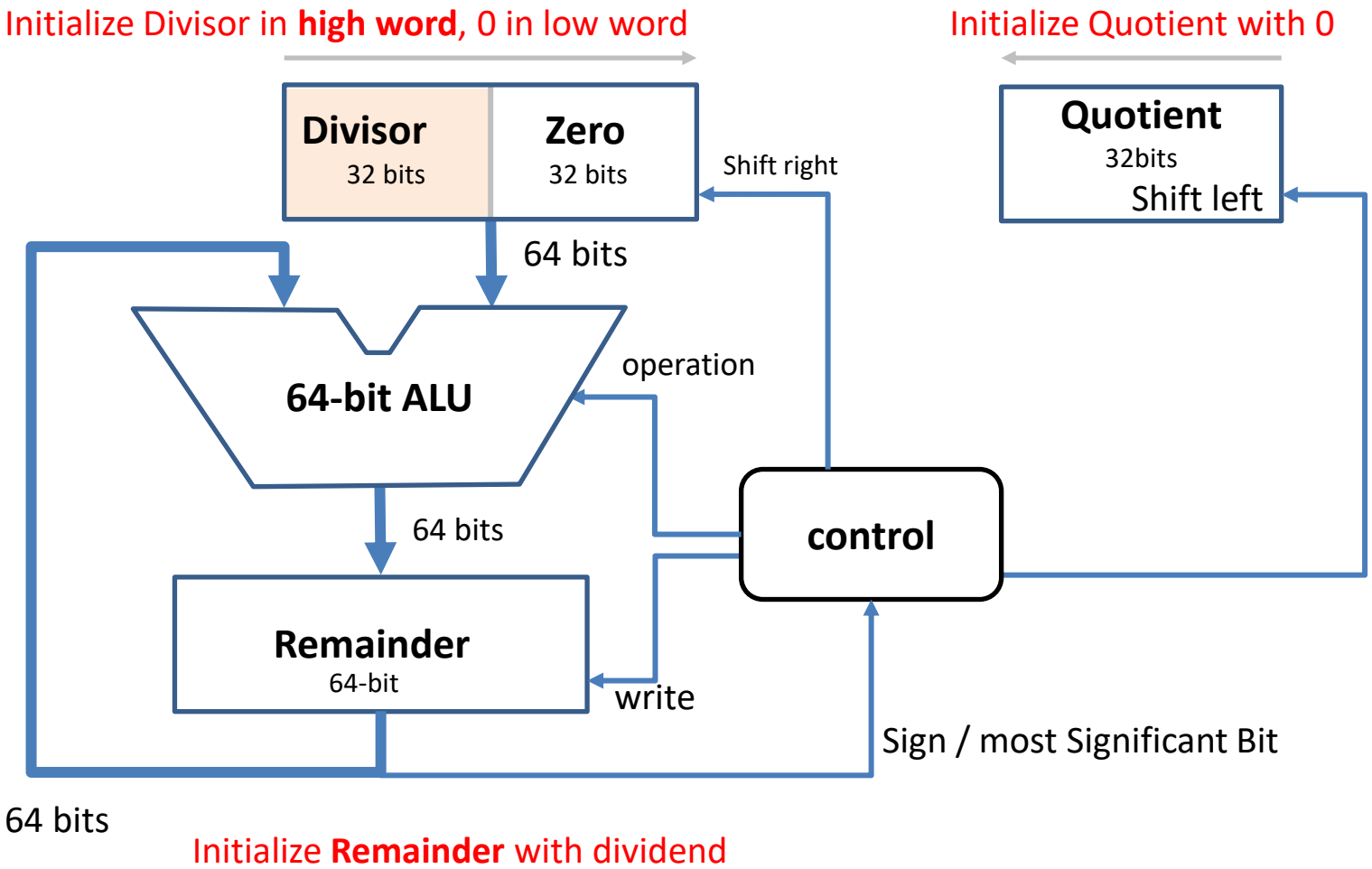




Division for Positive Numbers

4-bit Example
Dividend: 0111
Divisor: 0010
Repetition: 5





Division for Positive Numbers

4-bit Example
Dividend: 0111
Divisor: 0010
Repetition: 5 (i.e., n+1)

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Signed Division

- Simple solution
  - Divide using absolute values
  - Negate the quotient if the signs disagree
- What should be the sign of the remainder?

  Dividend = Quotient * Divisor + Remainder

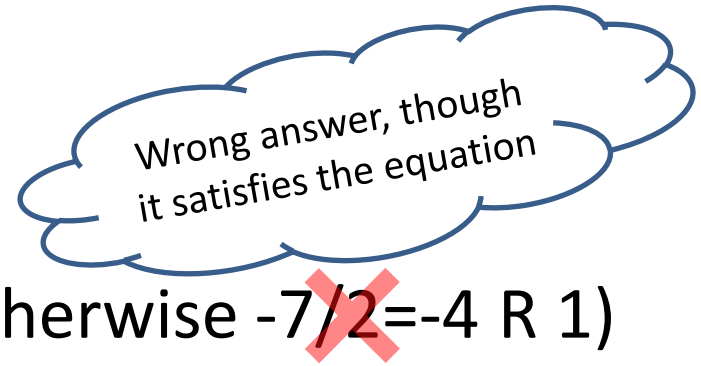
  7/2 = 3 R 1

  -7/2 =-3 R -1 (needs to be a negative remainder, otherwise -7/2=-4 R 1)

  Rule: Dividend and remainder must have the same signs

  7/-2 = -3 R 1

  -7/2 = -3 R -1



Review and more information

- Multiplication (Section 3.3)
- Division (Section 3.4)