

# Chapter-3

## "Arithmetic for Computers"

# Addition:  $(7)_{10} + (6)_{10}$

$$\begin{array}{r} & 1 \leftarrow & 1 \leftarrow & 0 \leftarrow \\ 0 & | & | & | \\ + & 0 & 1 & 1 \\ \hline 1 & (1) & 1 & (1) 0 & (0) 1 \end{array} \quad \left| \begin{array}{l} \text{\# bits with this color represent} \\ \text{carrying forward.} \end{array} \right.$$

# adding bits right to left.

# Subtraction:  $(7)_{10} - (6)_{10} = (7)_{10} + (-6)_{10}$

$$\begin{array}{r} 6 = 110 \\ + 6 = 0110 \\ = 1001 \\ + 1 \\ \hline - 6 = \underline{(1010)}_{2s} \end{array}$$

$$\begin{array}{r} + 7 = 1 \leftarrow 0 \leftarrow 1 \leftarrow 1 \leftarrow \\ - 6 = 1 \leftarrow 0 \leftarrow 0 \leftarrow 1 \leftarrow 0 \leftarrow 1 \\ \hline 1 & (1) & 0 & (1) & 0 & (1) & 0 & (0) & 1 \end{array}$$

## Overflow (signed)

### Different Approach

# Addition:

$\Rightarrow$  Add two same signed numbers  
if (answer also has same sign):

No overflow

else

Overflow

# After you get  
the answer check if  
it falls in the range  
or not.

$\Rightarrow$  if within range,  
no Overflow.

else,  
Overflow.

$\Rightarrow$  Sub two same signed number

Never Over Flow.

$\Rightarrow$  Add two different signed number

Never Over Flow.

$\Rightarrow$  Sub two different number

$$+A - (-B) = +A + B \Rightarrow \text{rule 1}$$

$$-A - (+B) = -A + (-B) \Rightarrow \text{rule 1}$$

# In case of unsigned integers, overflows are ignored.

are used to indicate memory addresses

# Compilers detect unsigned overflows using simple branch instruction.

$$\begin{array}{r} 6 = \boxed{110} \\ 7 = \boxed{111} \\ \hline 1101 \end{array}$$

logic: If the sum is less than either of the addends.

Overflow

if the difference is greater

than the minuend.

$$\begin{array}{r} 13 \\ - 3 \\ \hline 10 \end{array}$$

ALU = Arithmetic Logic Unit

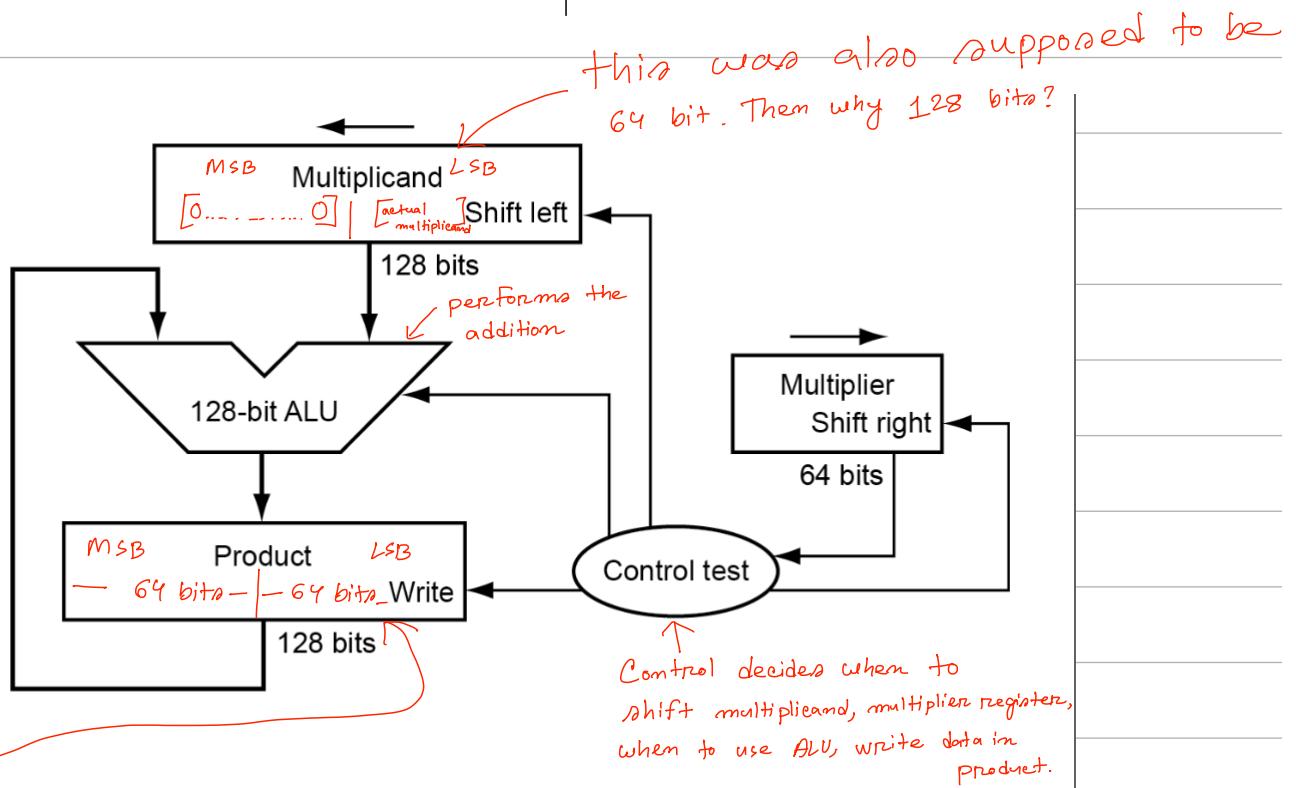
# Long Multiplication

$$\begin{array}{r}
 1000 \rightarrow \text{Multiplicand} \\
 \times 1001 \rightarrow \text{Multiplier} \\
 \hline
 1000 \\
 0000 \\
 0000 \\
 1000 \\
 \hline
 1001000 \rightarrow \text{Product}
 \end{array}$$

Maximum,

length of the product

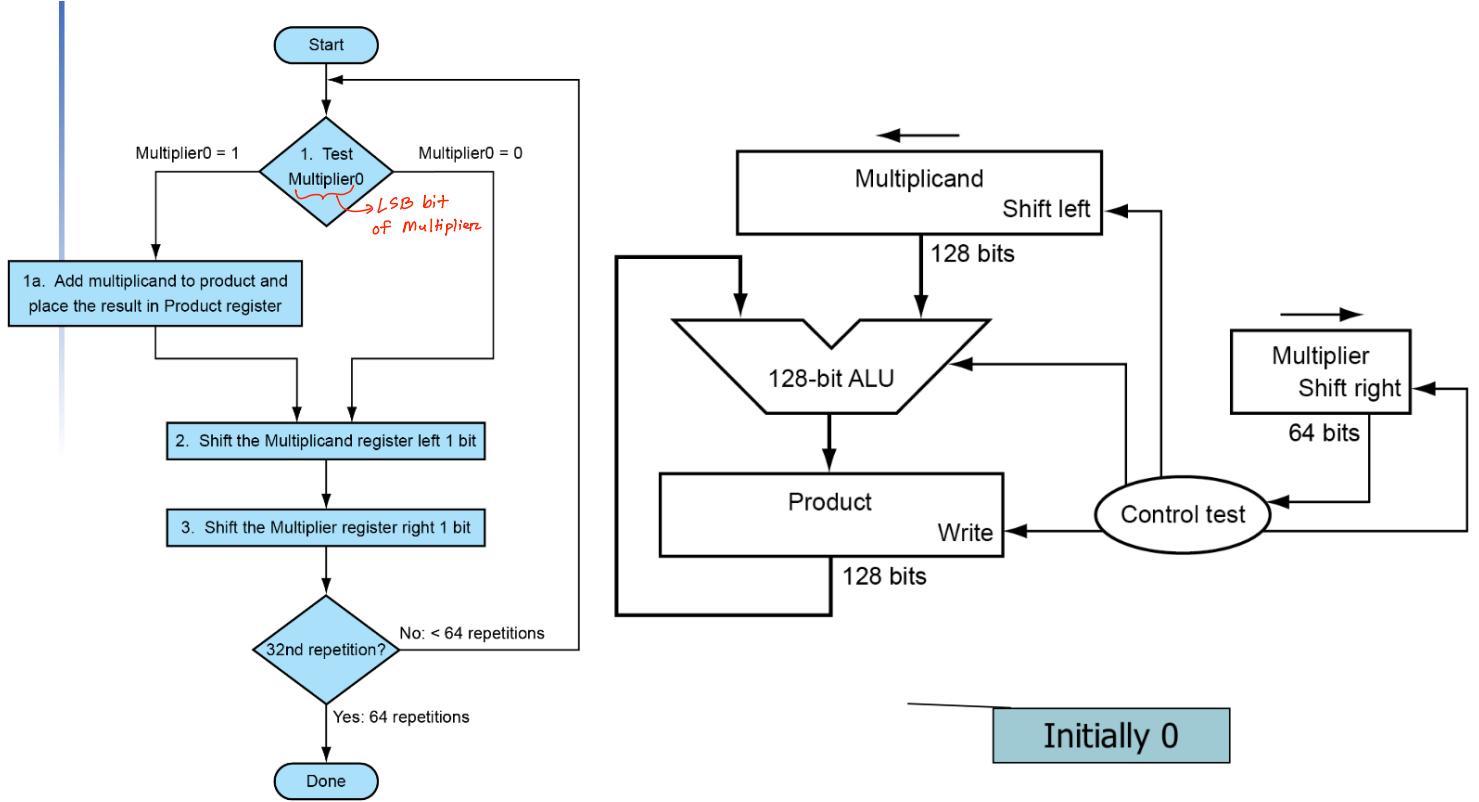
$$\begin{aligned}
 &= (\text{length of multiplicand} \\
 &+ \text{length of multiplier})
 \end{aligned}$$



# Multiply two 64 bit values, product length can be  $(64+64)=128$  bit.

But we do not have any 128 bit registers in RISC-V

Hence, we use two registers to store the 64 bit values

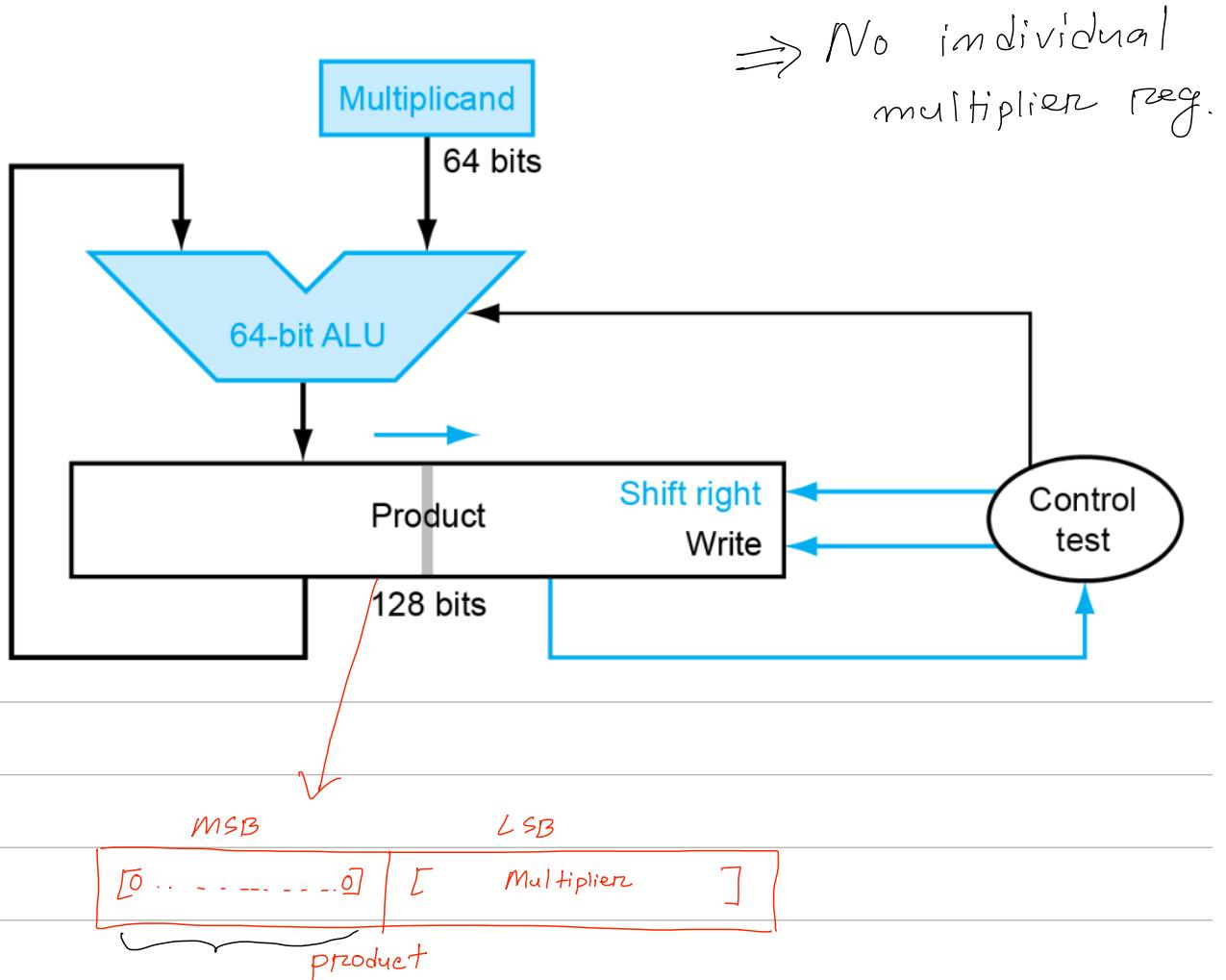


# Number of iteration = Number of bits in **Multiplier**

$\Rightarrow 100 \times 100$

	iteration	Multiplier	Multiplicand	Product
	0	1001	0000 1000	0000 0000
	1	1001	0000 1000	0000 1000
	1	1001	0001 0000	0000 1000
	1	0100	0001 0000	0000 1000
	2	0100	0001 0000	0000 1000
	2	0100	0010 0000	0000 1000
	2	0010	0010 0000	0000 1000
	3	0010	0010 0000	0000 1000
	3	0010	0100 0000	0000 1000
	3	0001	0100 0000	0000 1000
	4	0001	0100 0000	0100 1000
	4	0000	1000 0000	0100 1000

# Optimized Multiplication



# Number of iteration = Number of bits in Multiplier

Logic:

if (iteration <= multiplier bit length):

  if (multiplier[0] == 1):

    product\_MSB = Multiplicand + product\_MSB

    product = right shift product by 1

  elif (multiplier[0] == 0):

    product = right shift product by 1

$$8 \times 9 = 1000 \times 1001$$

iteration	multiplier	product
0	1000	0000 1001
1	1000	1000 1001 0100 0100
2	1000	0010 0010
3	1000	0001 0001
4	1000	1001 0001 0100 1000