EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 5_1: ALU

Xiaolin Xu
Department of ECE
Northeastern University

Binary Multiplier

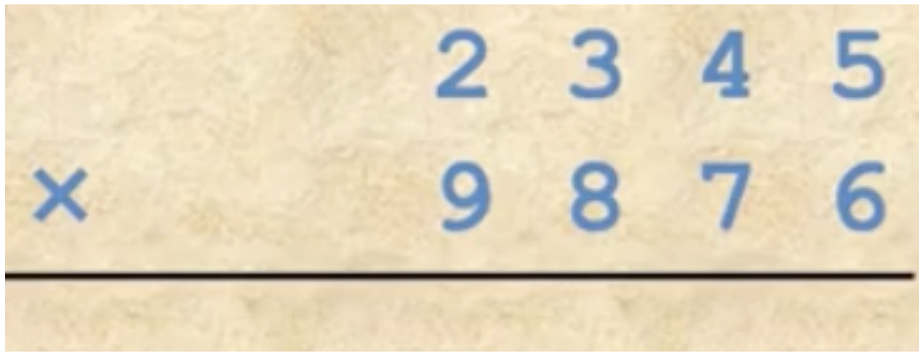
- Design
- Optimization
- What else?

An Example with Decimal Numbers

- Review this process
  - We need some look-up-table, i.e., 5*6 = 30

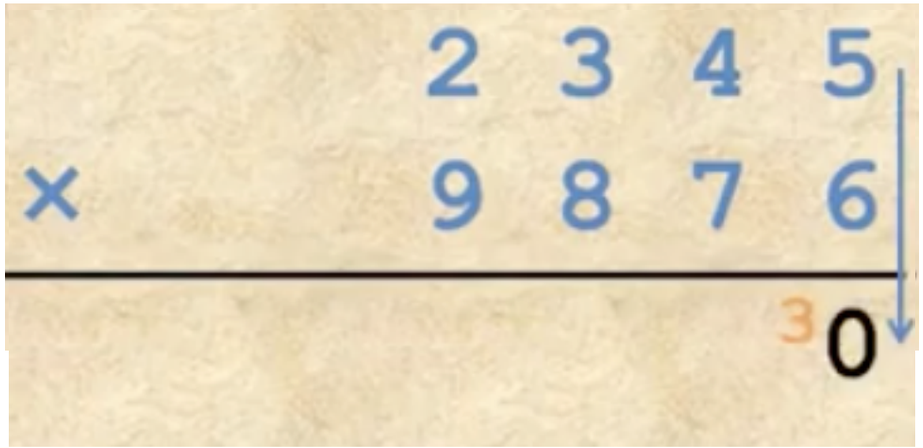
An Example with Decimal Numbers

- Review this process
  - We need some look-up-table, i.e., 5*6 = 30



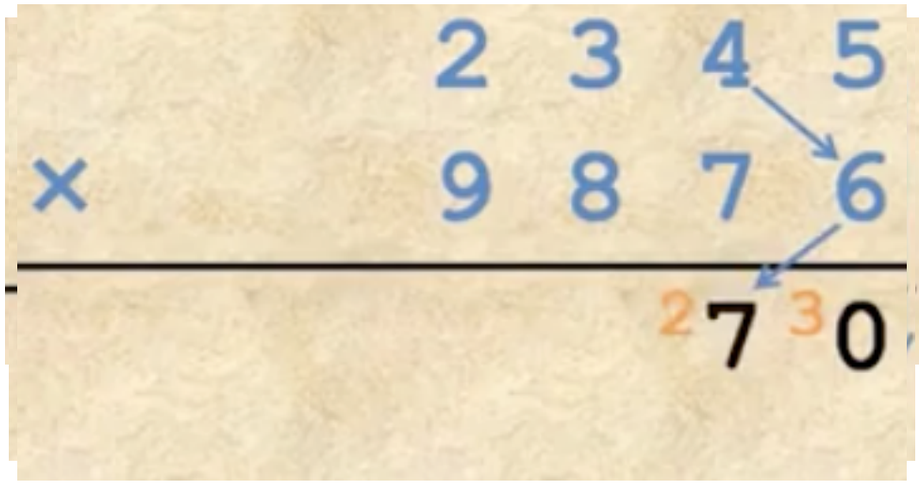
An Example with Decimal Numbers

- Review this process
  - We need some look-up-table, i.e., 5*6 = 30



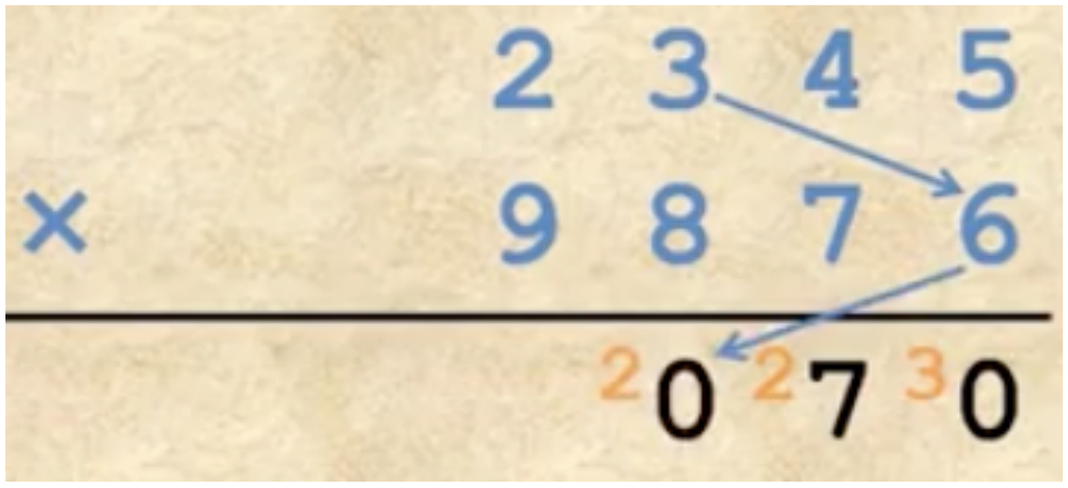
An Example with Decimal Numbers

- Review this process
  - We need some look-up-table, i.e., 5*6 = 30



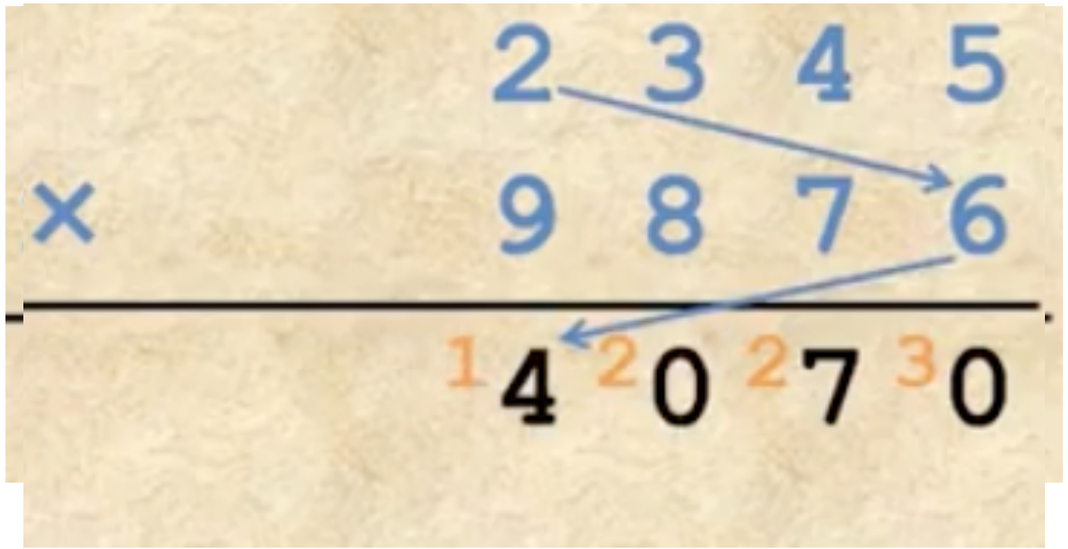
An Example with Decimal Numbers

- Review this process
  - We need some look-up-table, i.e., 5*6 = 30



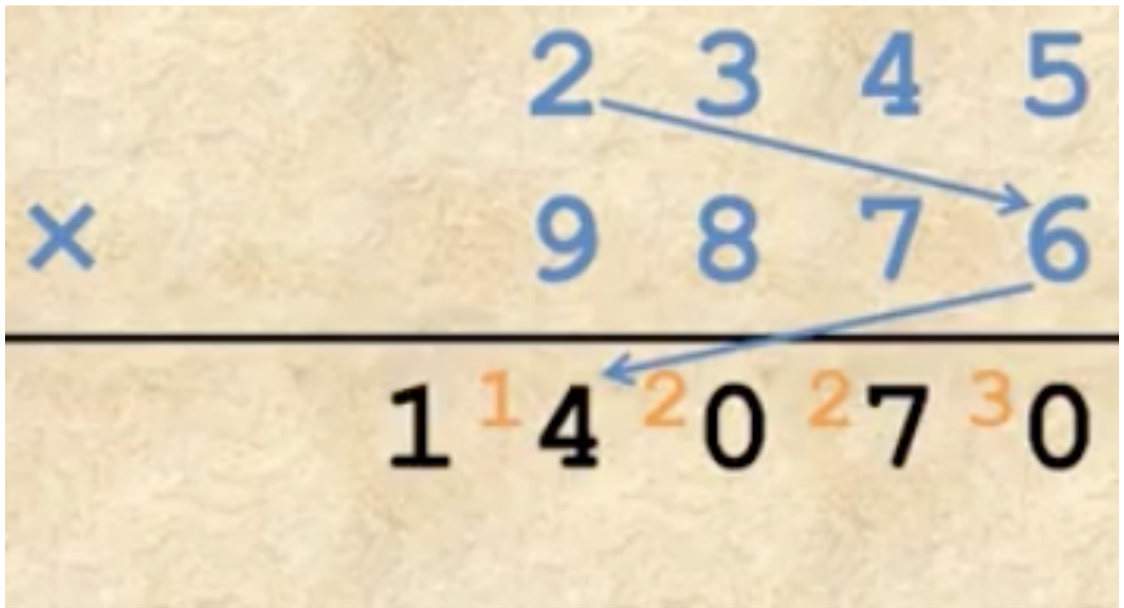
An Example with Decimal Numbers

- Review this process
  - We need some look-up-table, i.e., 5*6 = 30



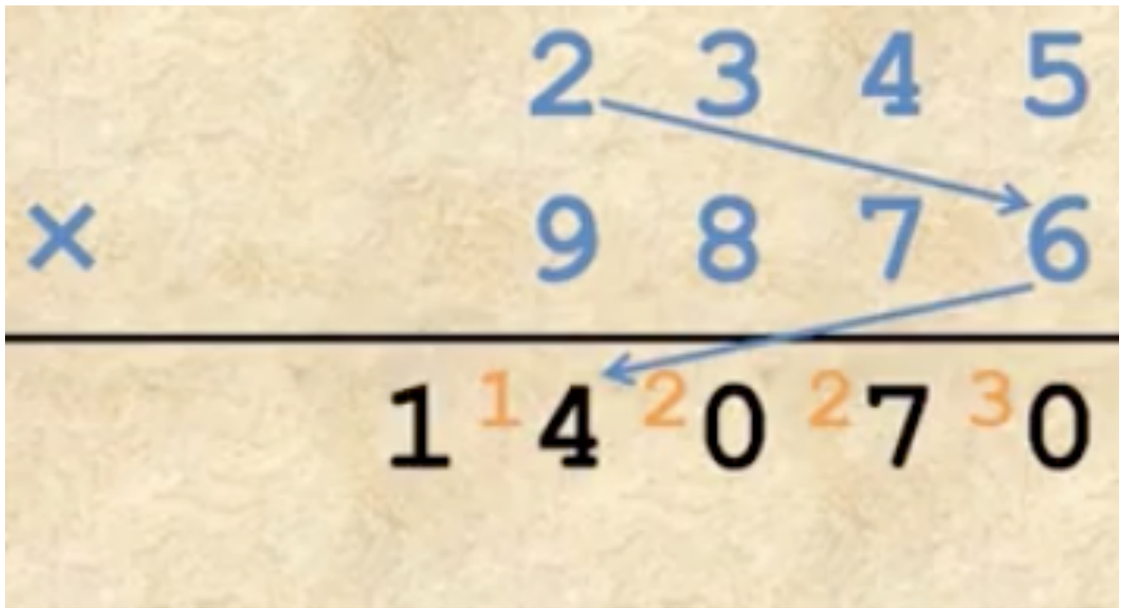
An Example with Decimal Numbers

- Review this process
  - We need some look-up-table, i.e., 5*6 = 30



An Example with Decimal Numbers

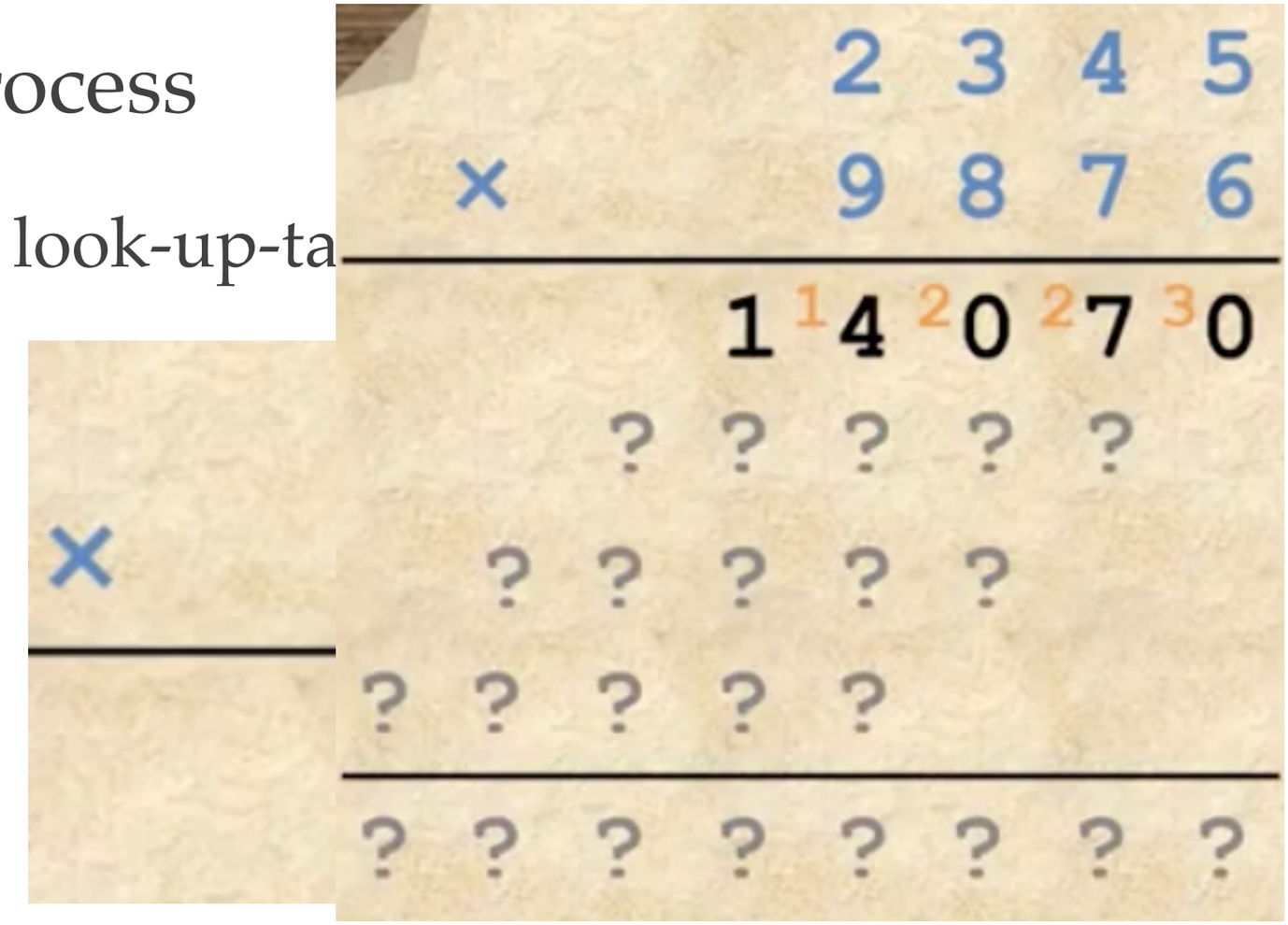
- Review this process
  - We need some look-up-table, i.e., 5*6 = 30



- There will be many more such operation!

An Example with Decimal Numbers

- Review this process
  - We need some look-up-ta



- There will be many more such operation!

Any Quick, Straightforward Solution?

- A simpler example

Any Quick, Straightforward Solution?

- A simpler example

```
        1 0 0 0
  x     1 0 0 1
-------------------
```

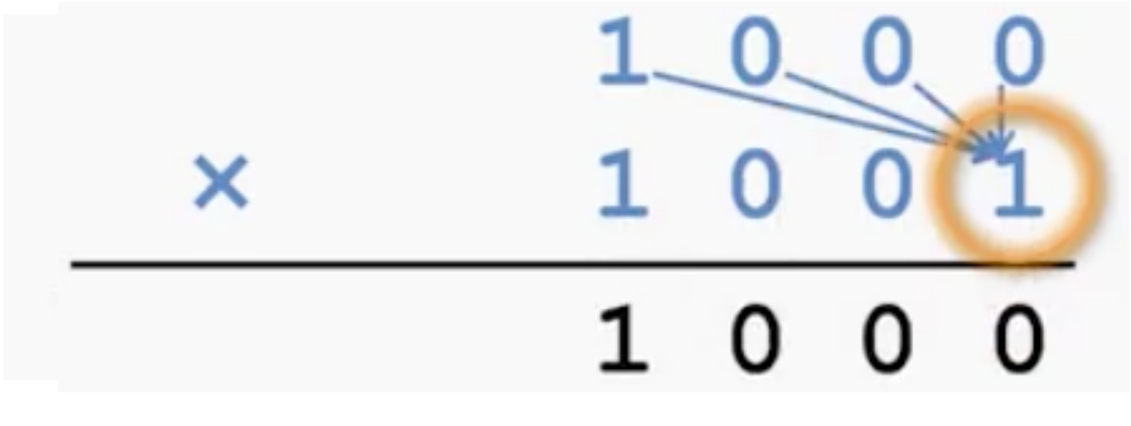
Any Quick, Straightforward Solution?

- A simpler example

```
      1 0 0 0
  x   1 0 0 1
-------------
```

Any Quick, Straightforward Solution?

- A simpler example



Any Quick, Straightforward Solution?

- A simpler example

```
          1  0  0  0
   x      1  0  0  1
-------------------
          1  0  0  0
       0  0  0  0
```

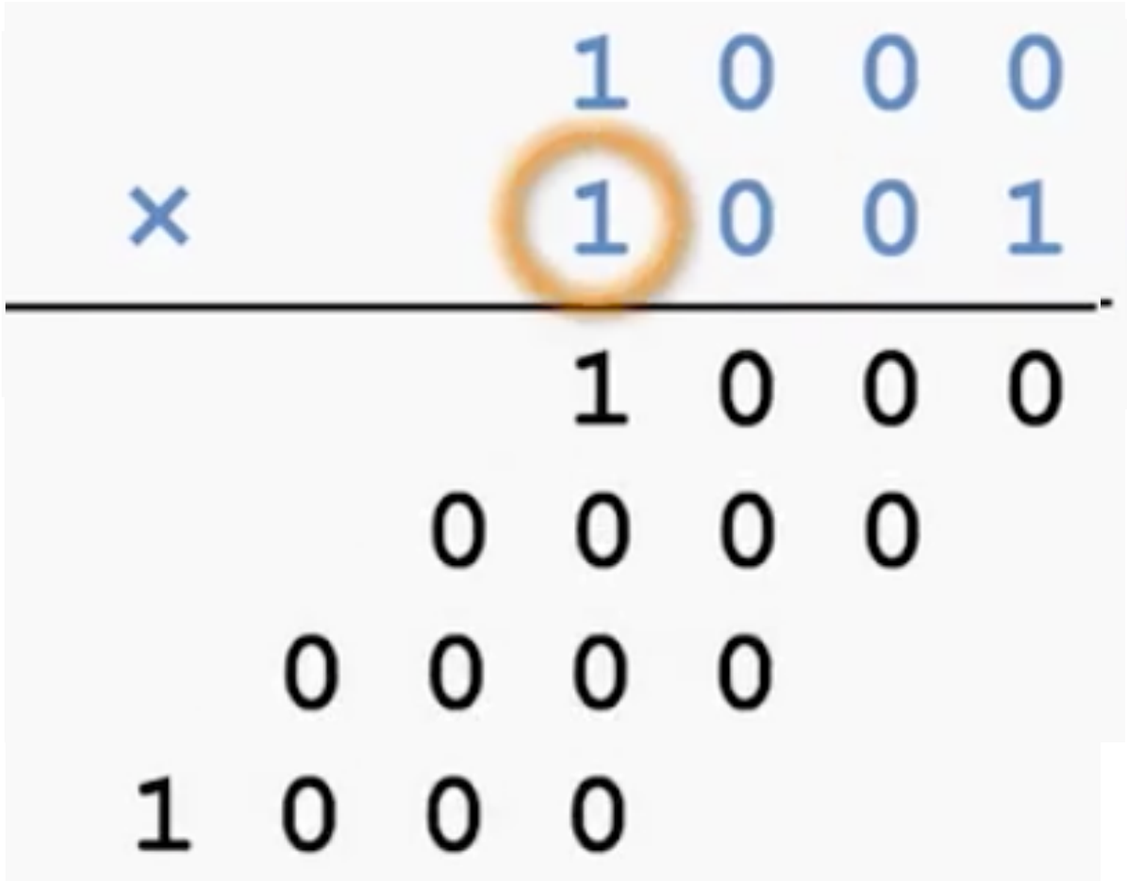
Any Quick, Straightforward Solution?

- A simpler example

```
          1  0  0  0
   x      1  0  0  1
-------------------
          1  0  0  0
       0  0  0  0
    0  0  0  0
```

Any Quick, Straightforward Solution?

- A simpler example



Any Quick, Straightforward Solution?

- A simpler example

```
          1  0  0  0
   ×      1  0  0  1
-------------------
          1  0  0  0
       0  0  0  0
    0  0  0  0
 1  0  0  0
-------------------
 1  0  0  1  0  0  0
```

Any Quick, Straightforward Solution?

- A simpler example

```
          1  0  0  0
   ×      1  0  0  1
-------------------
          1  0  0  0
       0  0  0  0
    0  0  0  0
 1  0  0  0
-------------------
 1  0  0  1  0  0  0
```

- What we learned from there?

Summary from the Previous Example

Summary from the Previous Example

```
          1 0 0 0
    ×     1 0 0 1
-----------------
          1 0 0 0
        0 0 0 0
      0 0 0 0
    1 0 0 0
-----------------
    1 0 0 1 0 0 0
```

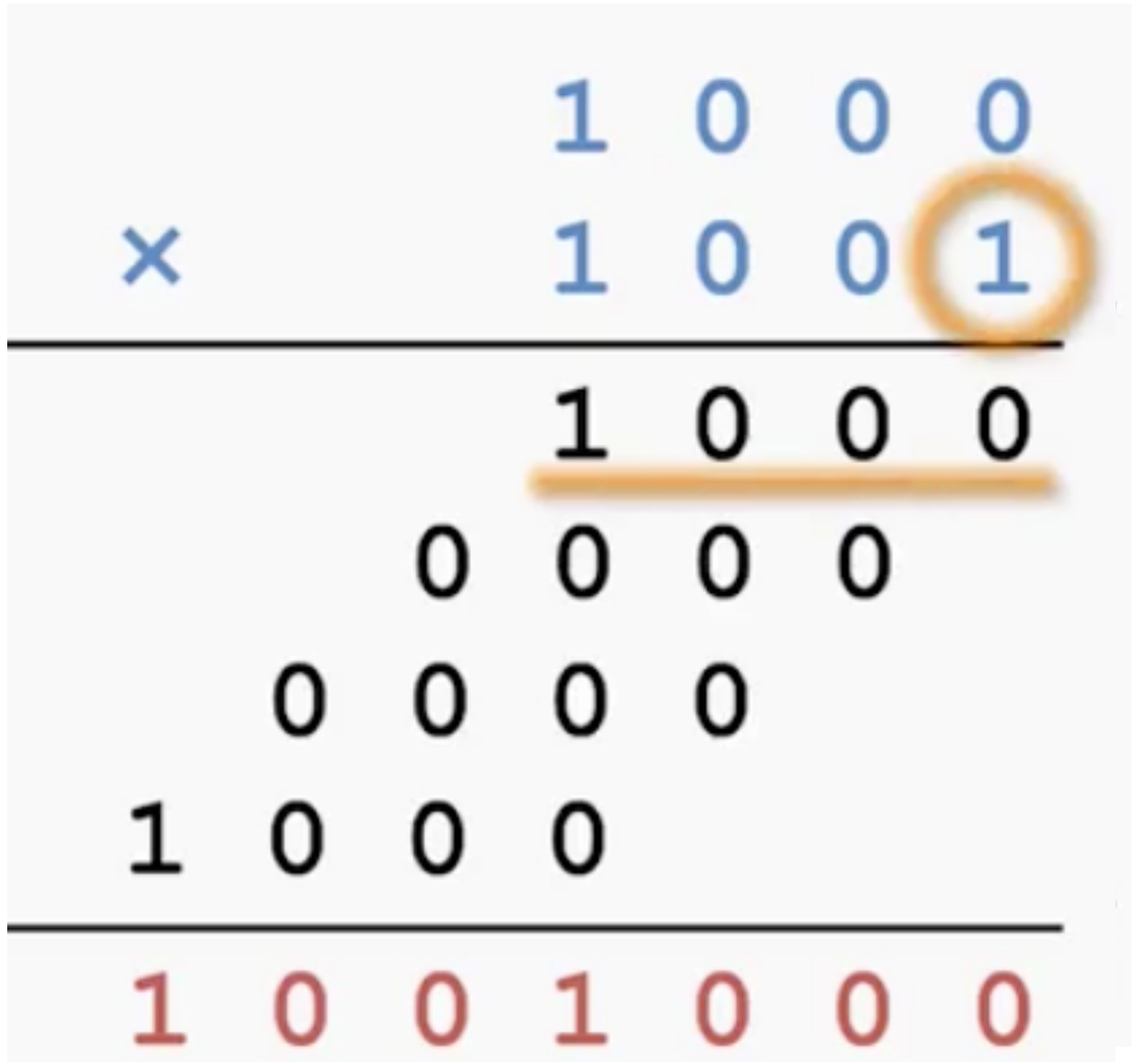
Summary from the Previous Example

- Multiplier = 1

```
          1 0 0 0
    ×     1 0 0 1
-----------------
          1 0 0 0
        0 0 0 0
      0 0 0 0
    1 0 0 0
-----------------
  1 0 0 1 0 0 0
```

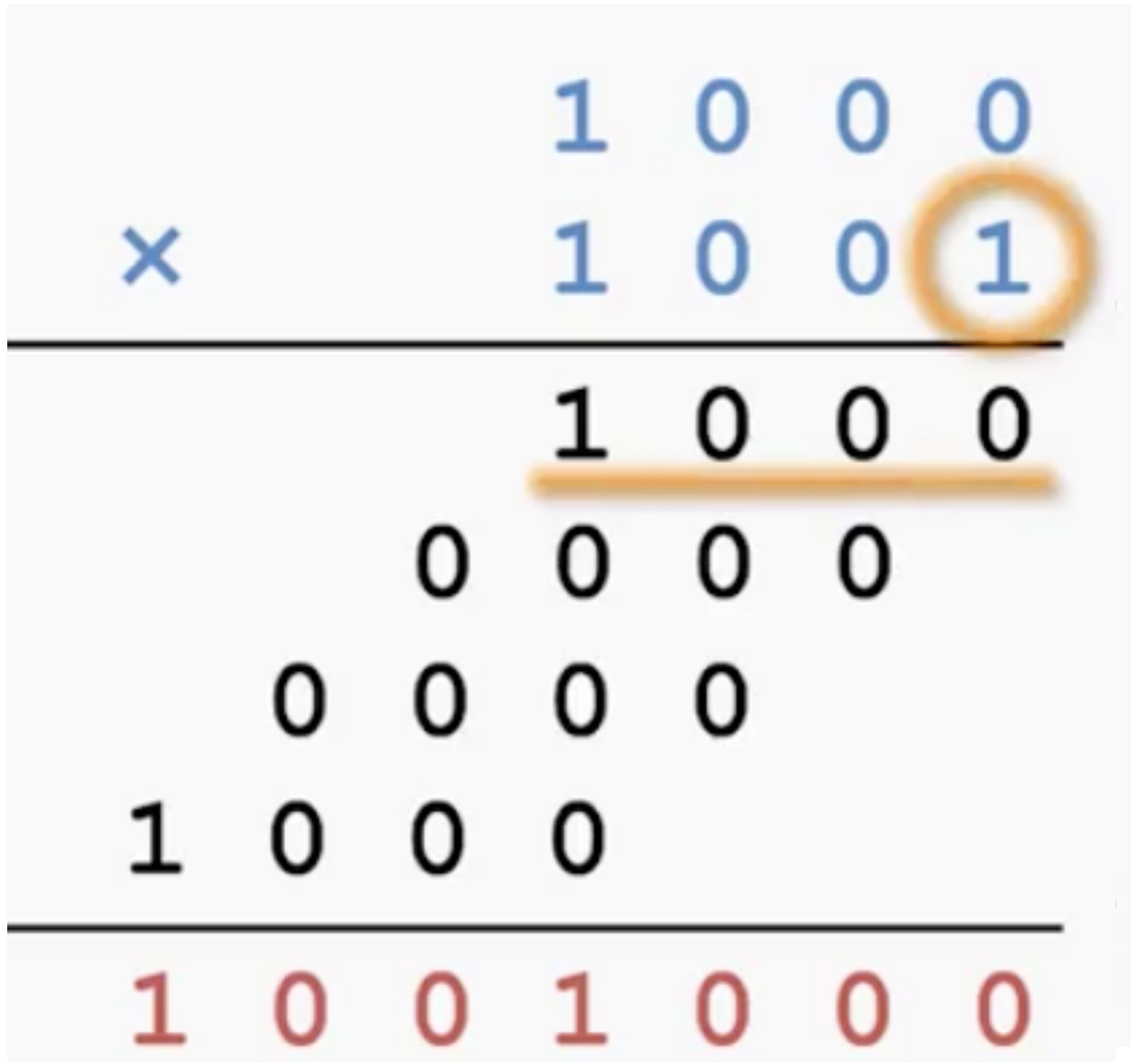
Summary from the Previous Example

- Multiplier = 1



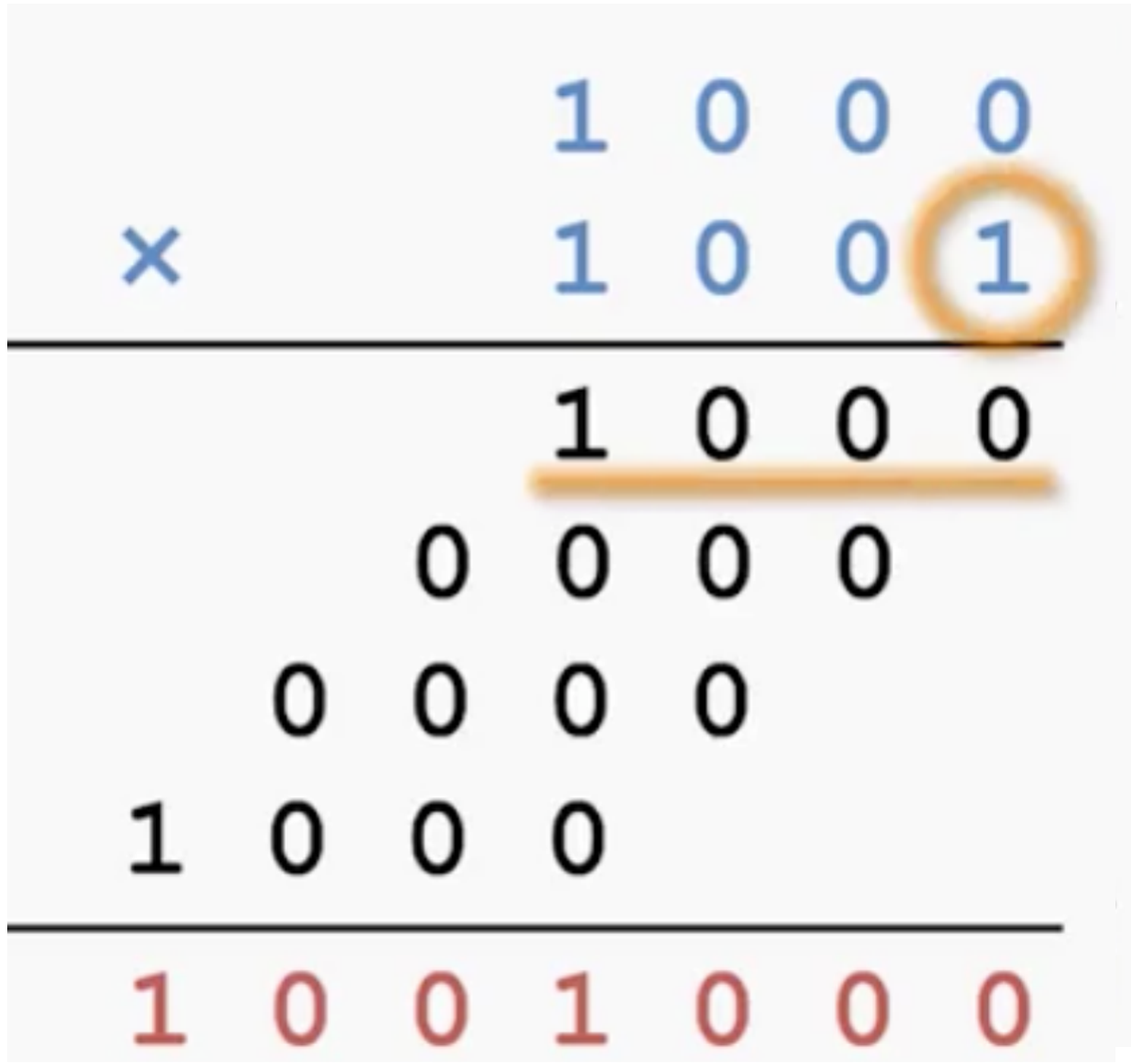
Summary from the Previous Example

- Multiplier = 1
  - Directly apply multiplicand



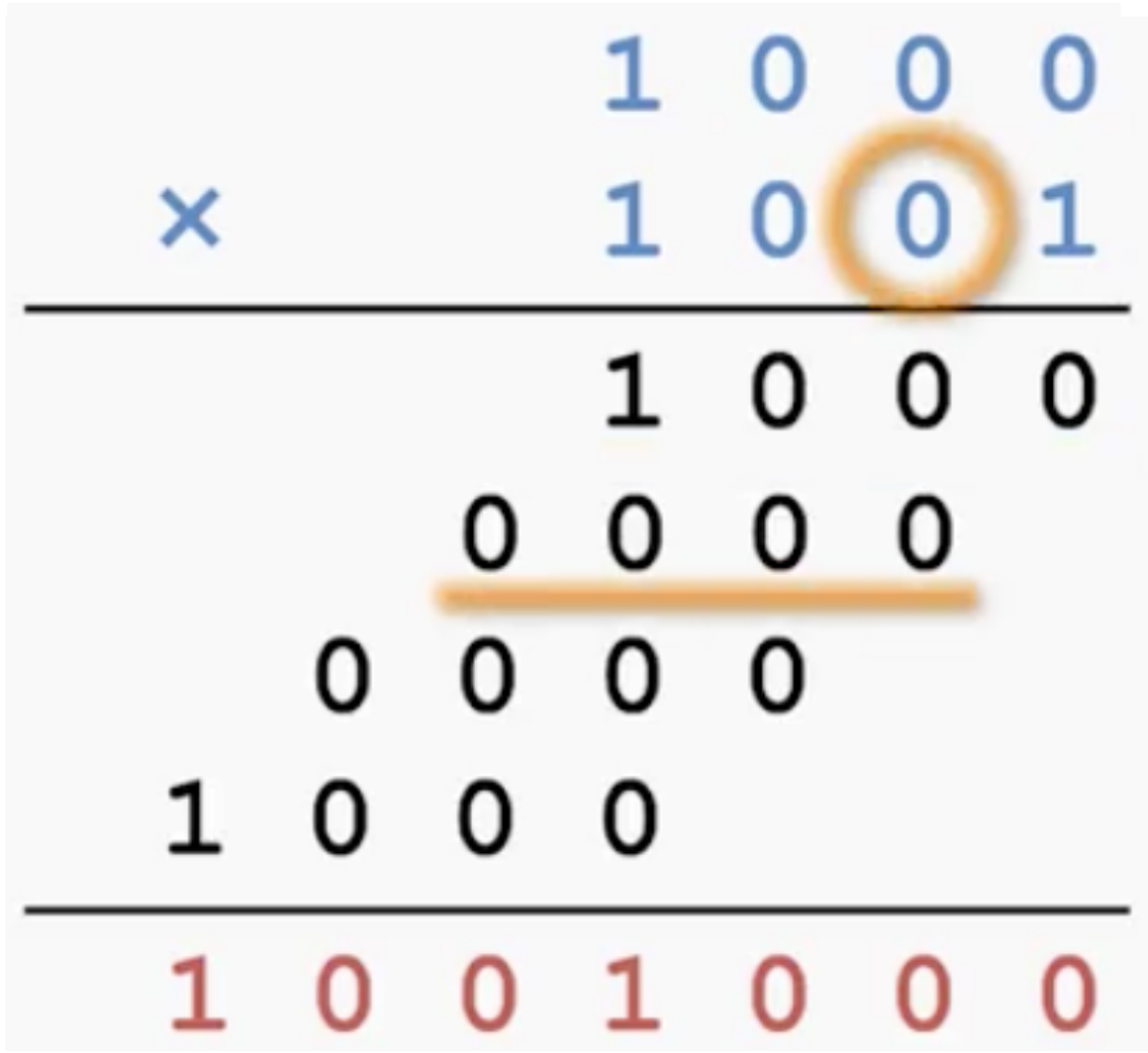
Summary from the Previous Example

- Multiplier = 1
  - Directly apply multiplicand
- Multiplier = 0



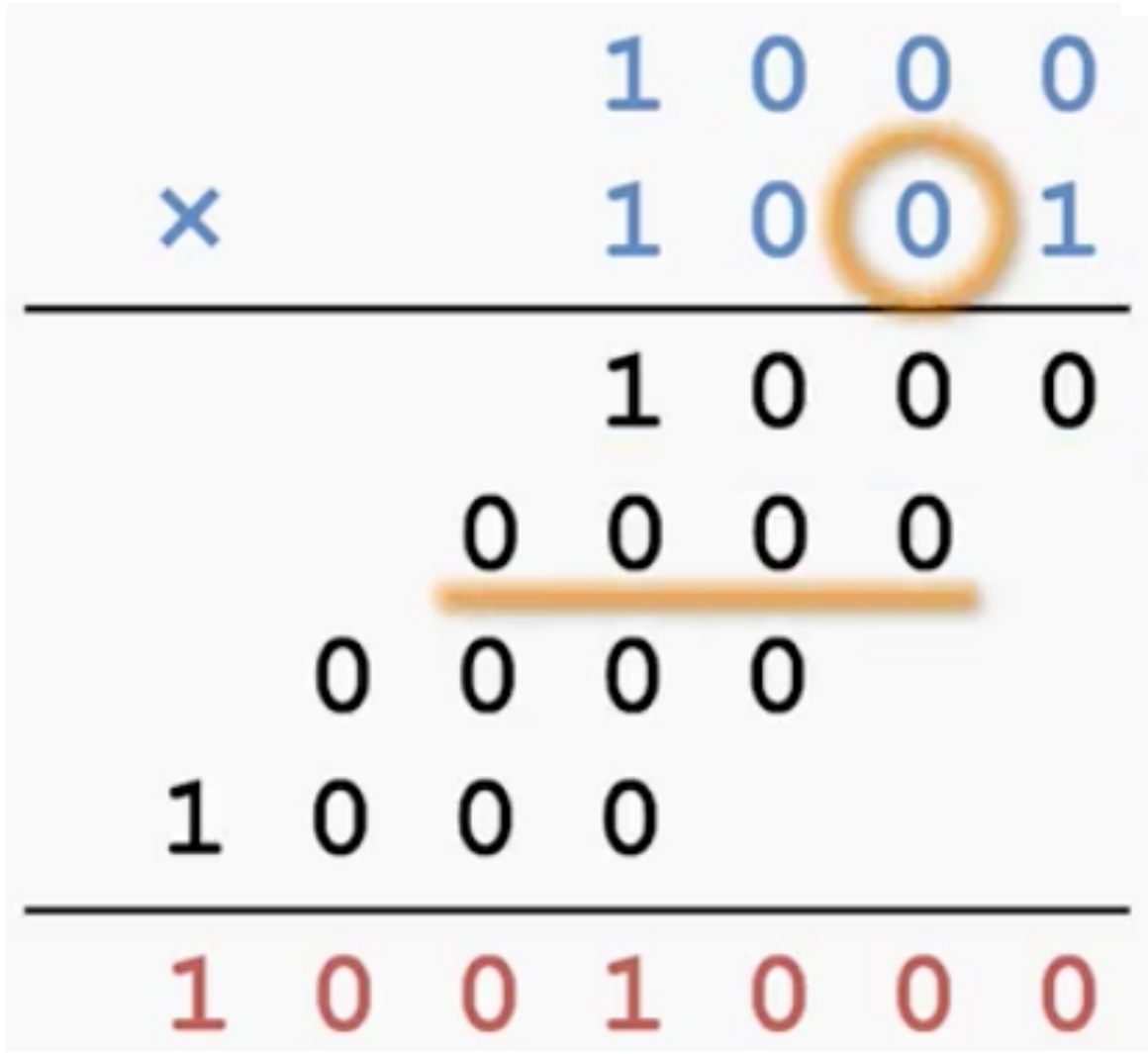
Summary from the Previous Example

- Multiplier = 1
  - Directly apply multiplicand
- Multiplier = 0



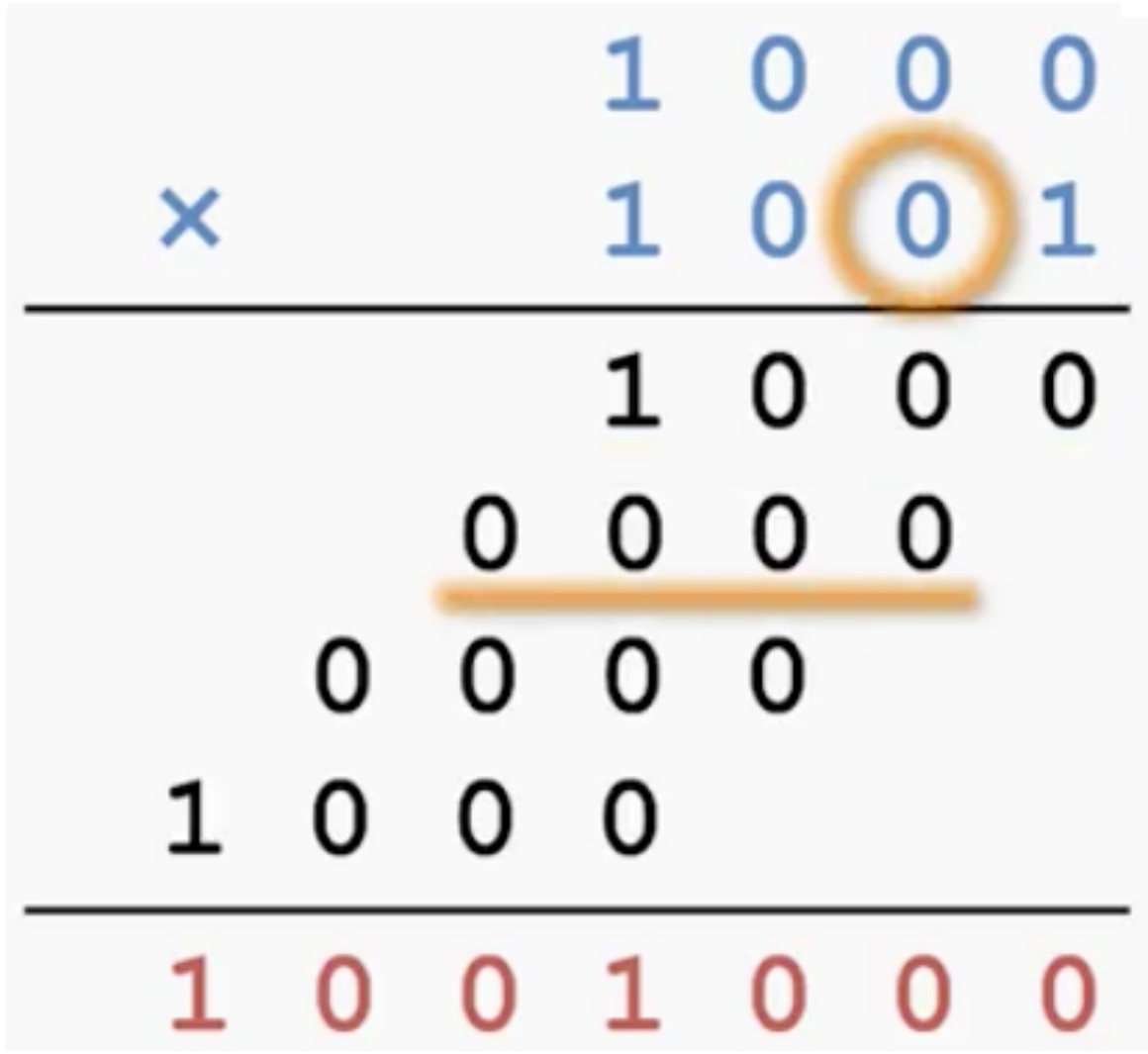
Summary from the Previous Example

- Multiplier = 1
  - Directly apply multiplicand
- Multiplier = 0
  - Directly apply all-0



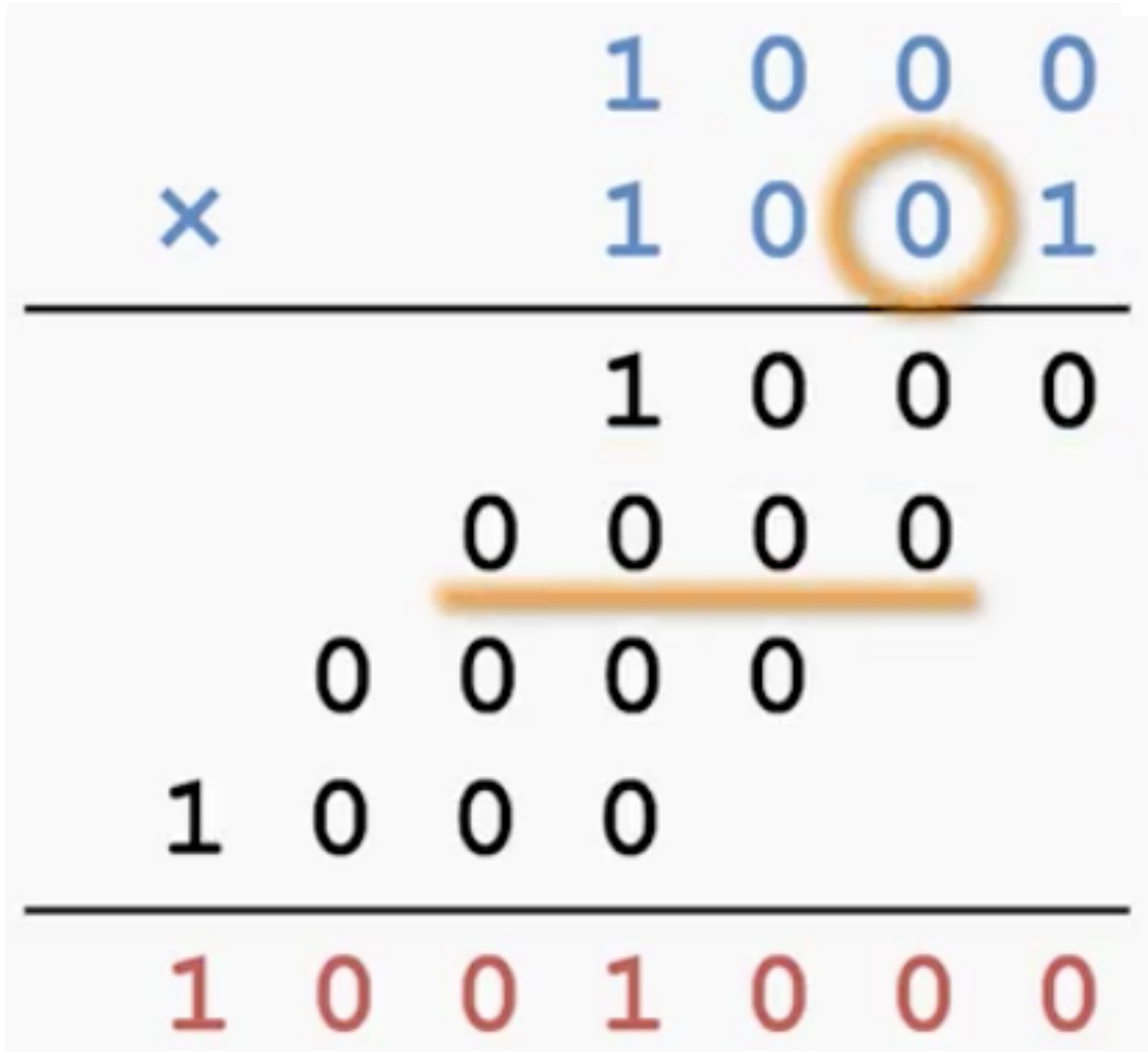
Summary from the Previous Example

- Multiplier = 1
  - Directly apply multiplicand
- Multiplier = 0
  - Directly apply all-0
  - No more look-up-table checking



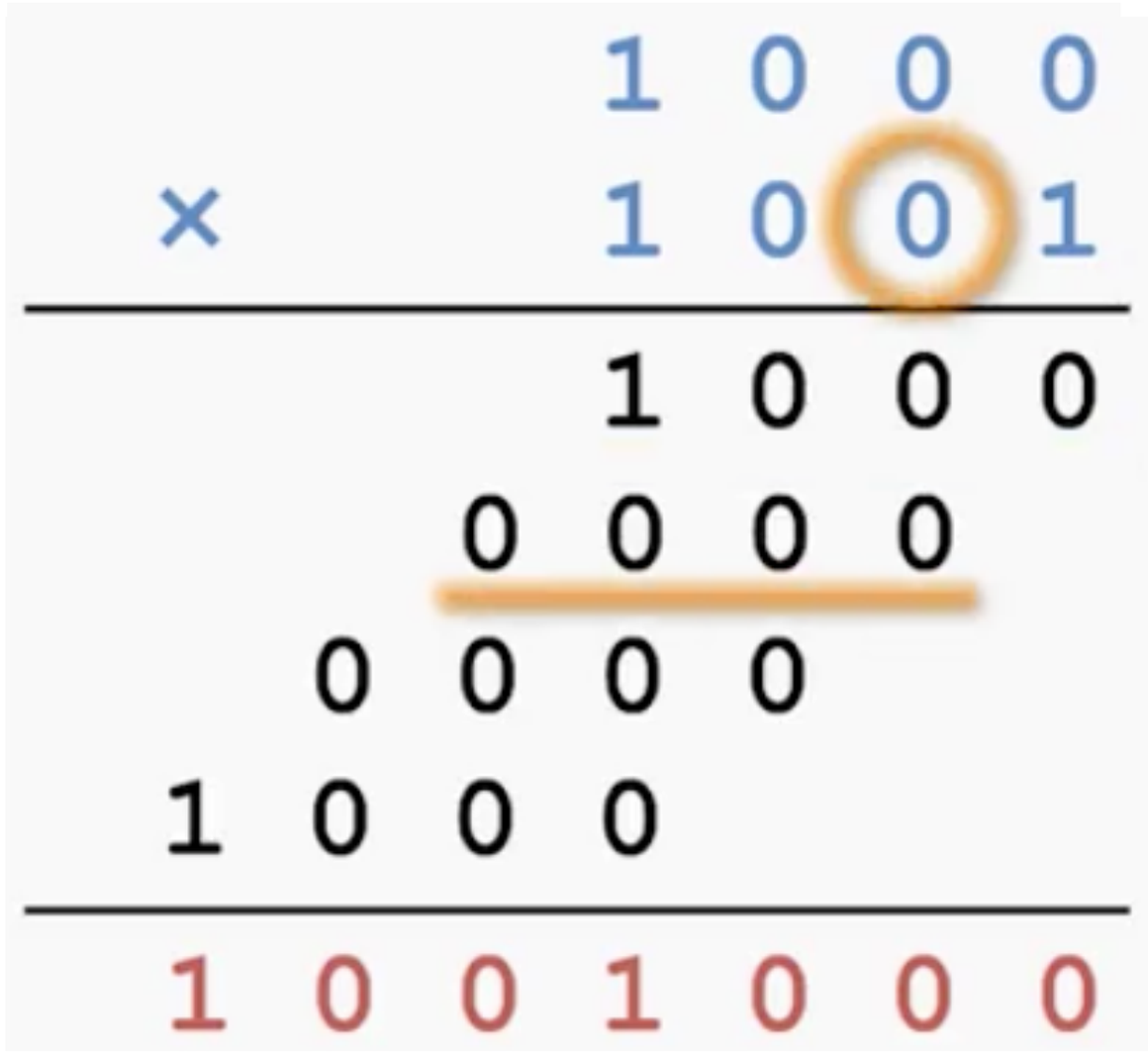
Summary from the Previous Example

- Multiplier = 1
  - Directly apply multiplicand
- Multiplier = 0
  - Directly apply all-0
  - No more look-up-table checking
  - This example is applicable to binary!



Summary from the Previous Example

- Multiplier = 1
  - Directly apply multiplicand
- Multiplier = 0
  - Directly apply all-0
  - No more look-up-table checking
  - This example is applicable to binary!
  - An important reason why our computer use binary data!



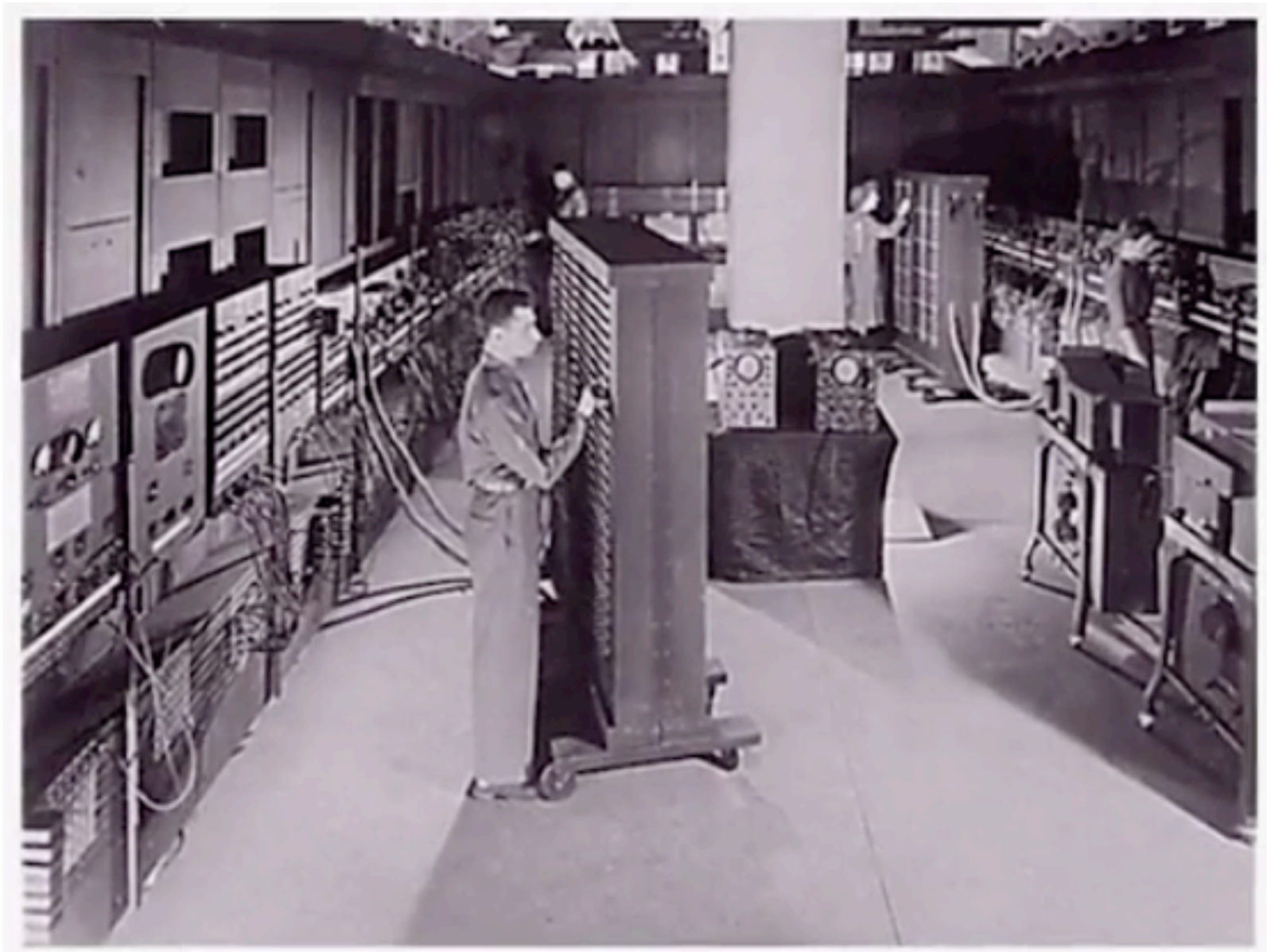
Between Decimal and Binary

Between Decimal and Binary

- ENIAC, a computer based on decimal

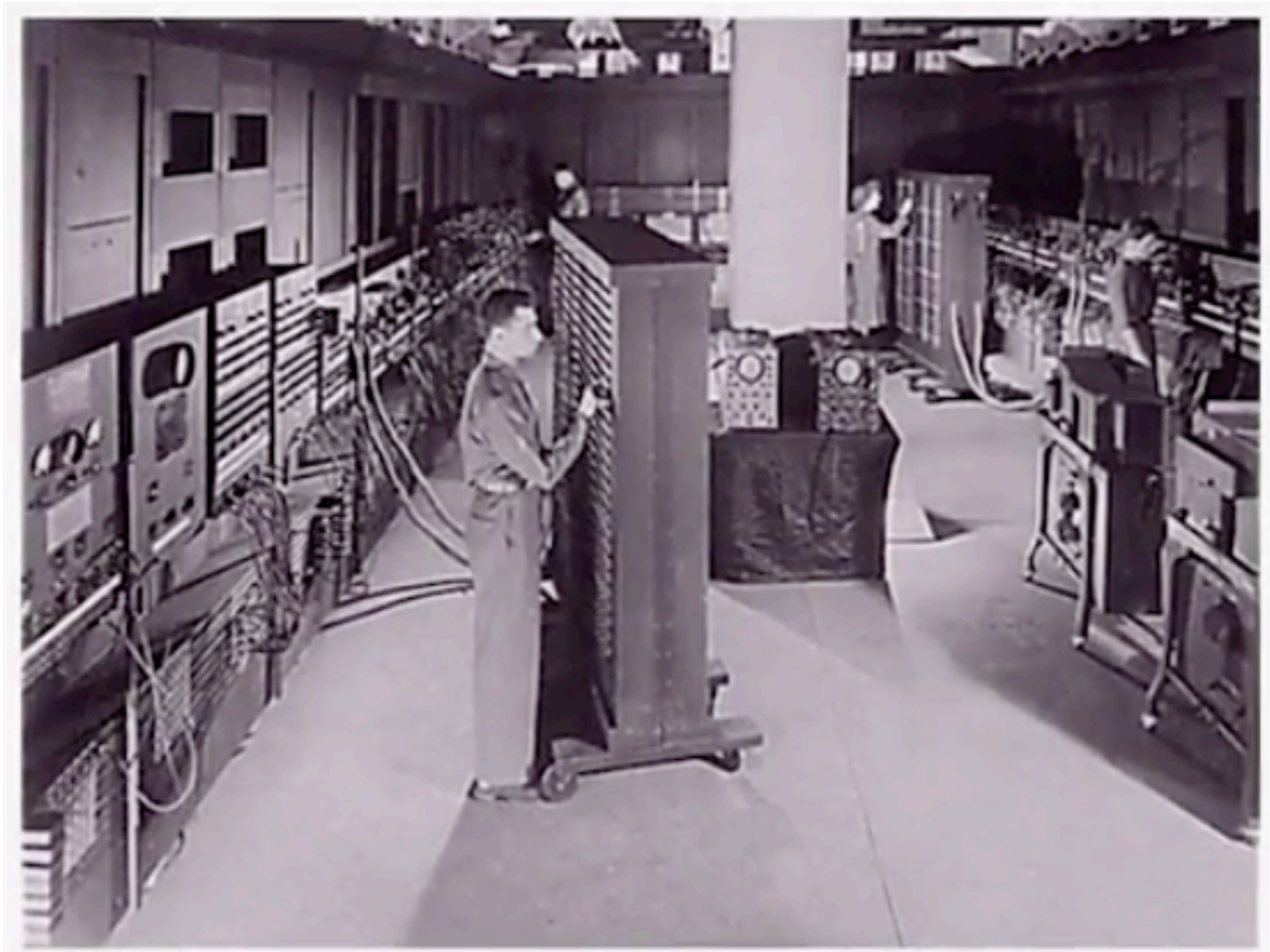
Between Decimal and Binary

- ENIAC, a computer based on decimal



Between Decimal and Binary

- ENIAC, a computer based on decimal
- EDVAC, binary computing

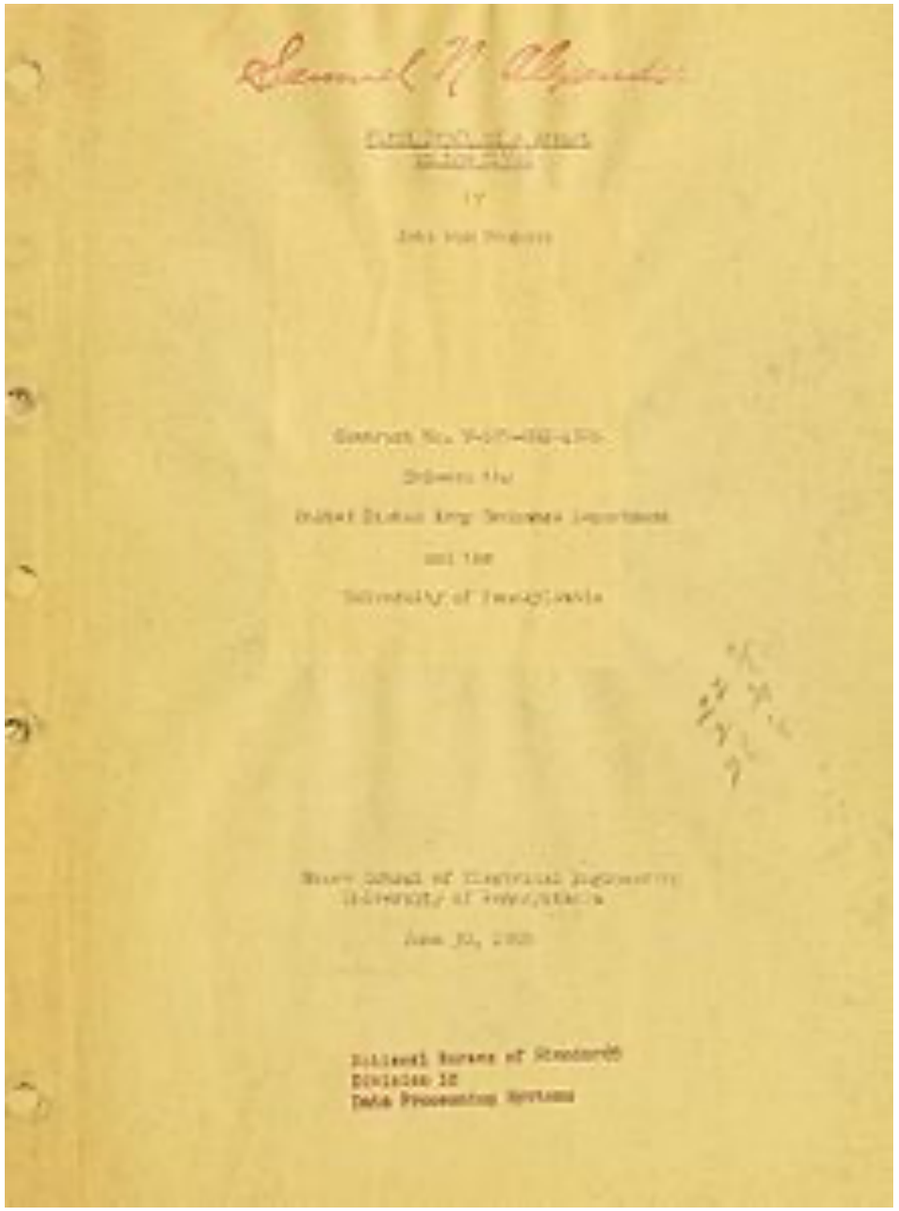


Between Decimal and Binary

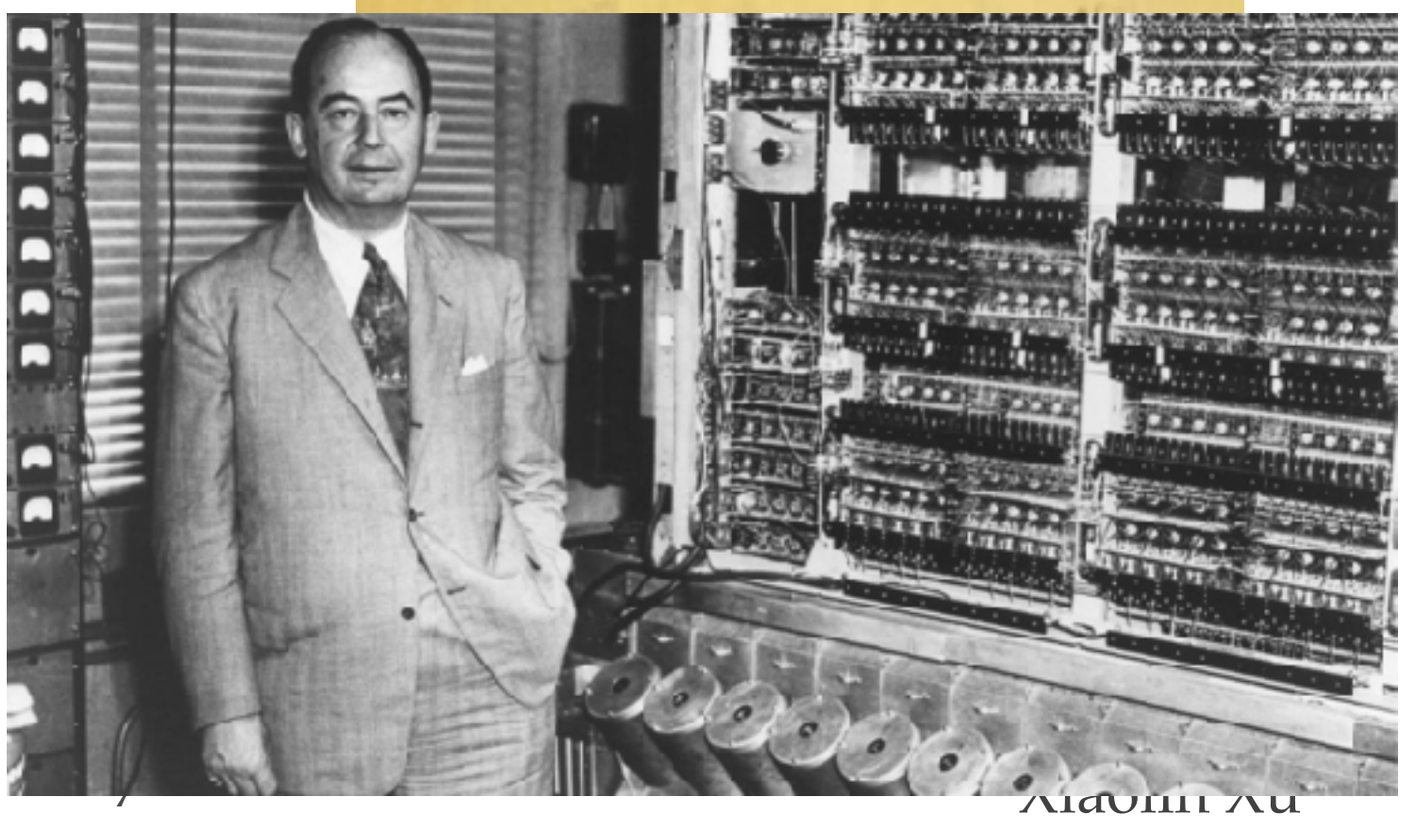
- ENIAC, a computer based on decimal
- EDVAC, binary computing

Between Decimal and Binary

Between Decimal and Binary

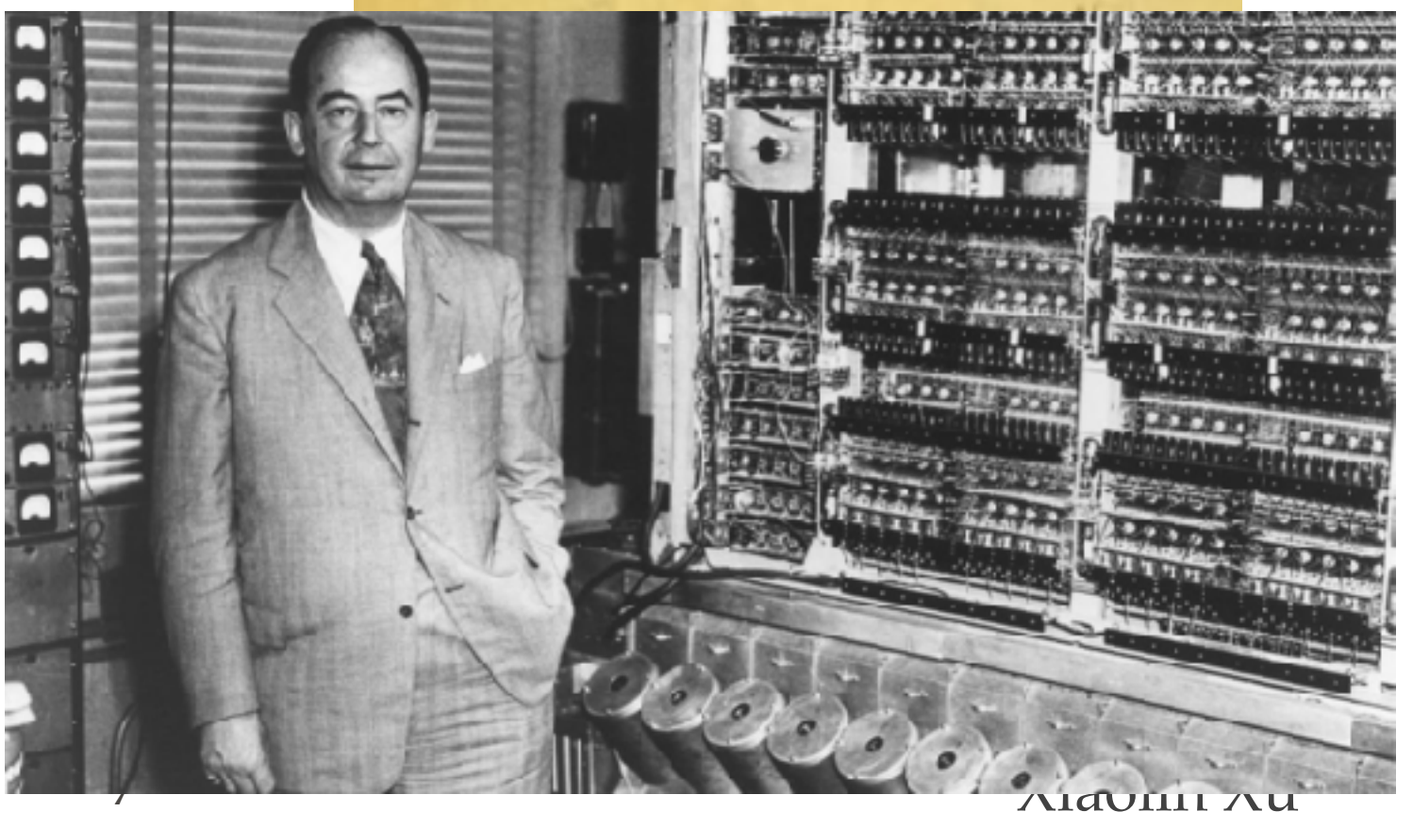


Between Decimal and Binary



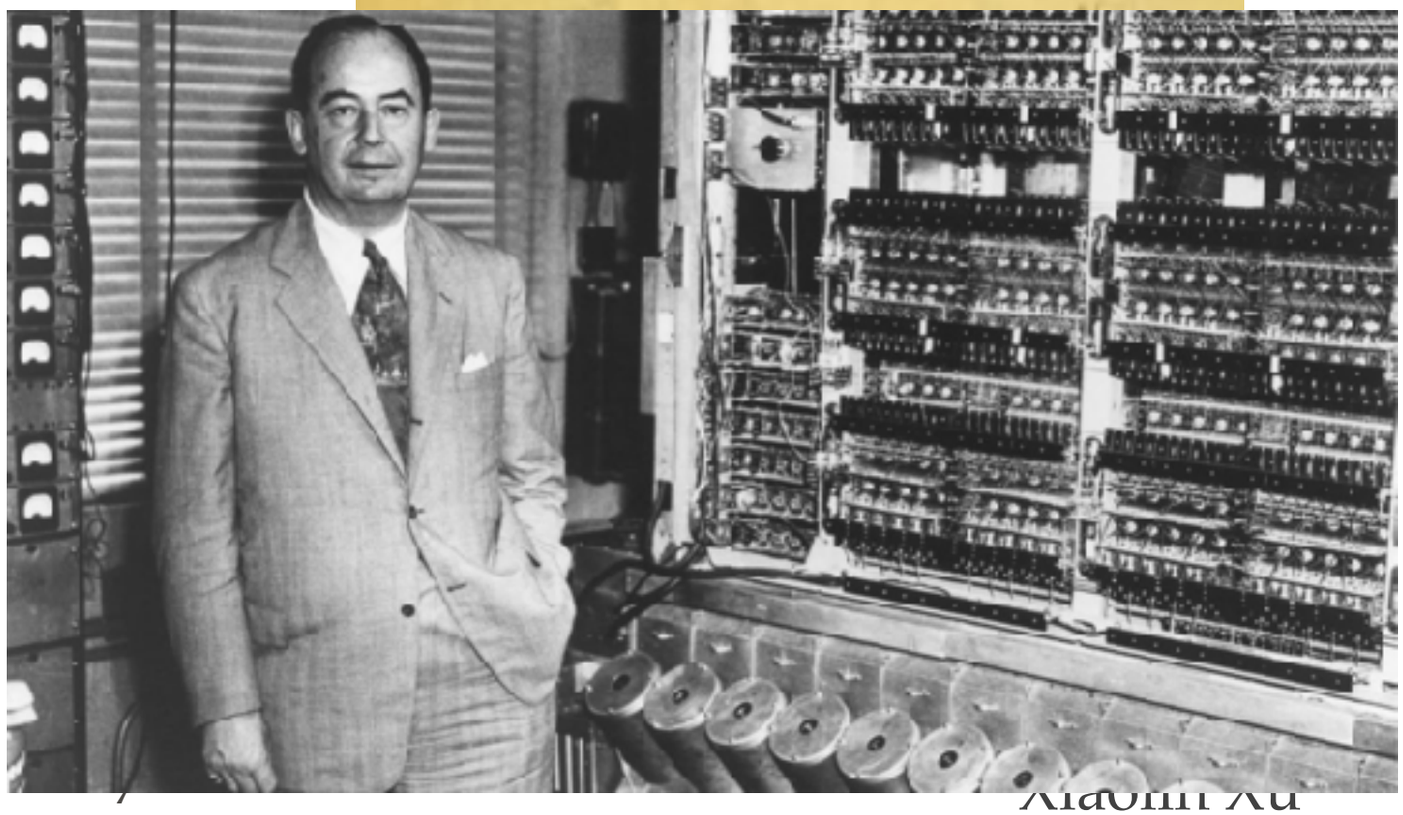
Between Decimal and Binary

- Transistor is a “all-or-non” device
  - Thus, only two values are reasonable



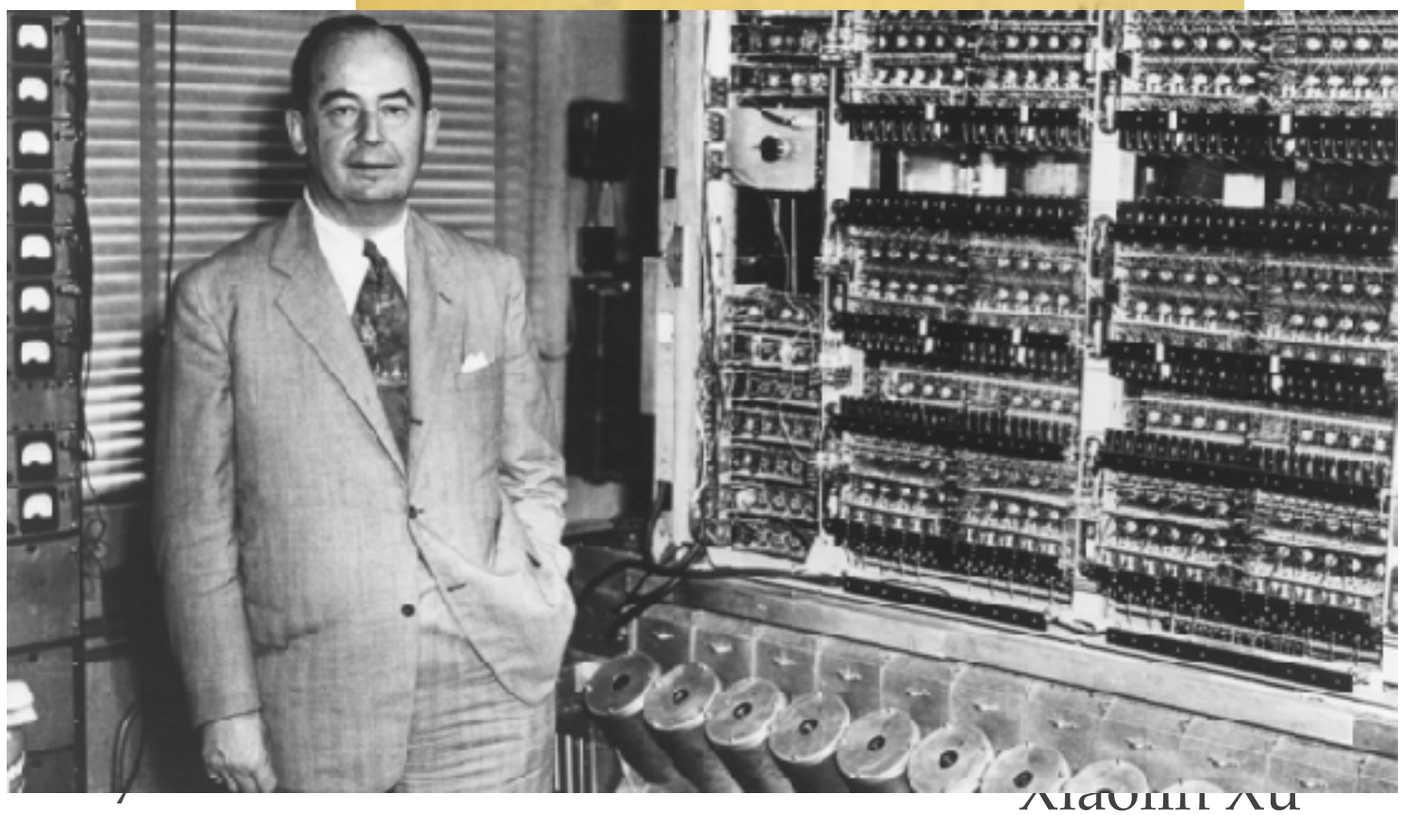
Between Decimal and Binary

- Transistor is a “all-or-non” device
  - Thus, only two values are reasonable
- Binary can significantly simplify the multiplication and division
  - No more look-up-table
  - No more intermediate value to be stored



Between Decimal and Binary

- Transistor is a “all-or-non” device
  - Thus, only two values are reasonable
- Binary can significantly simplify the multiplication and division
  - No more look-up-table
  - No more intermediate value to be stored
- But remember, decimal is for human being
  - We need converter! I.e., keyboard, monitor, etc.



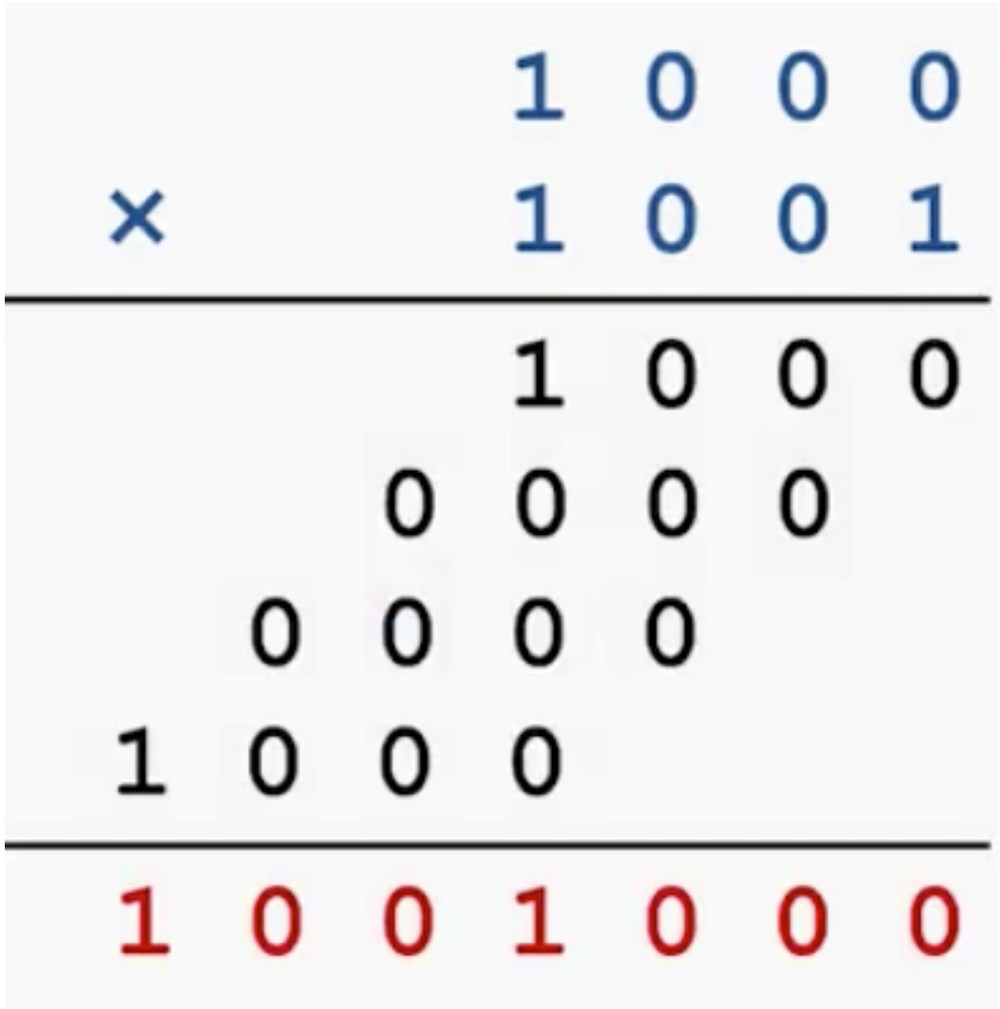
Hardware Implementation

Hardware Implementation

```
          1 0 0 0
    ×     1 0 0 1
-------------------
          1 0 0 0
        0 0 0 0
      0 0 0 0
    1 0 0 0
-------------------
  1 0 0 1 0 0 0
```

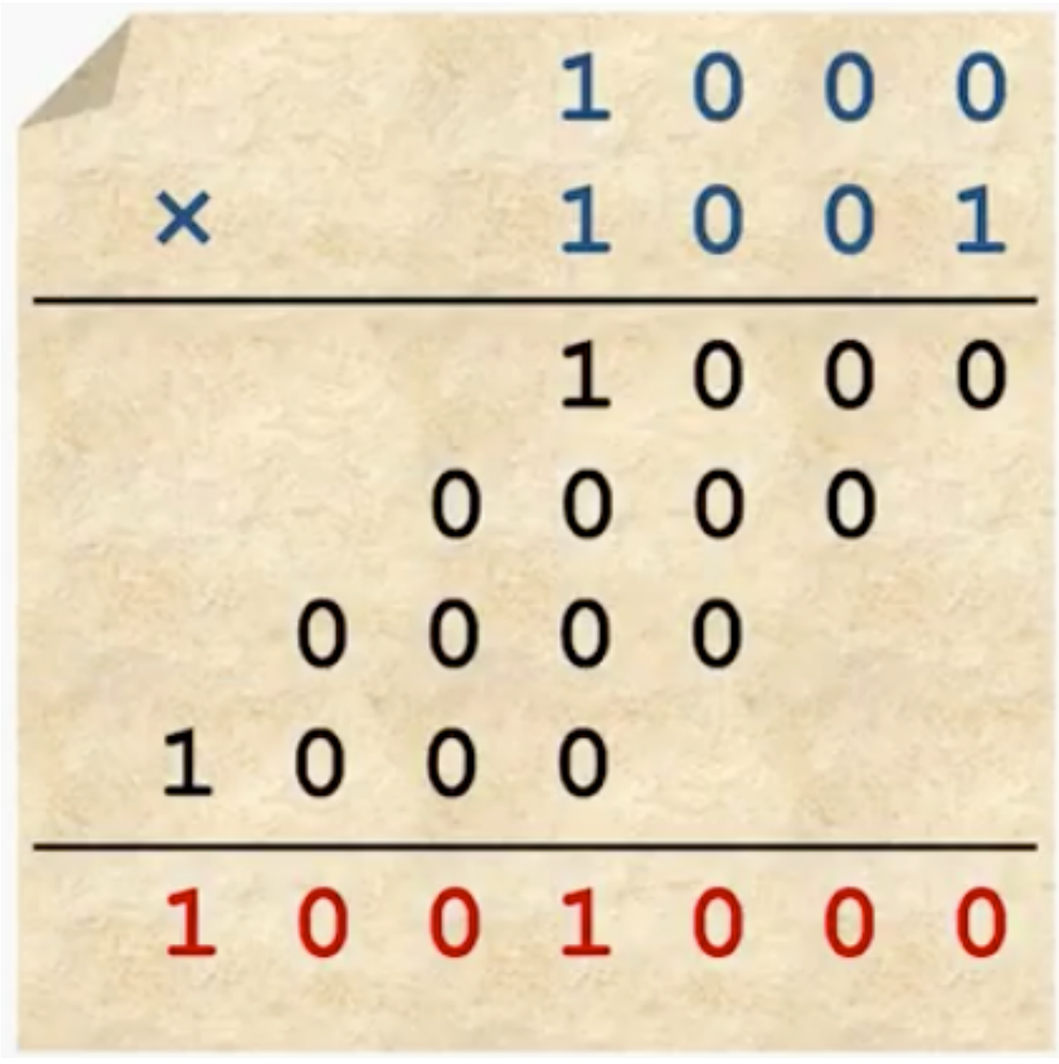
Hardware Implementation

- Different from what we do on a paper?



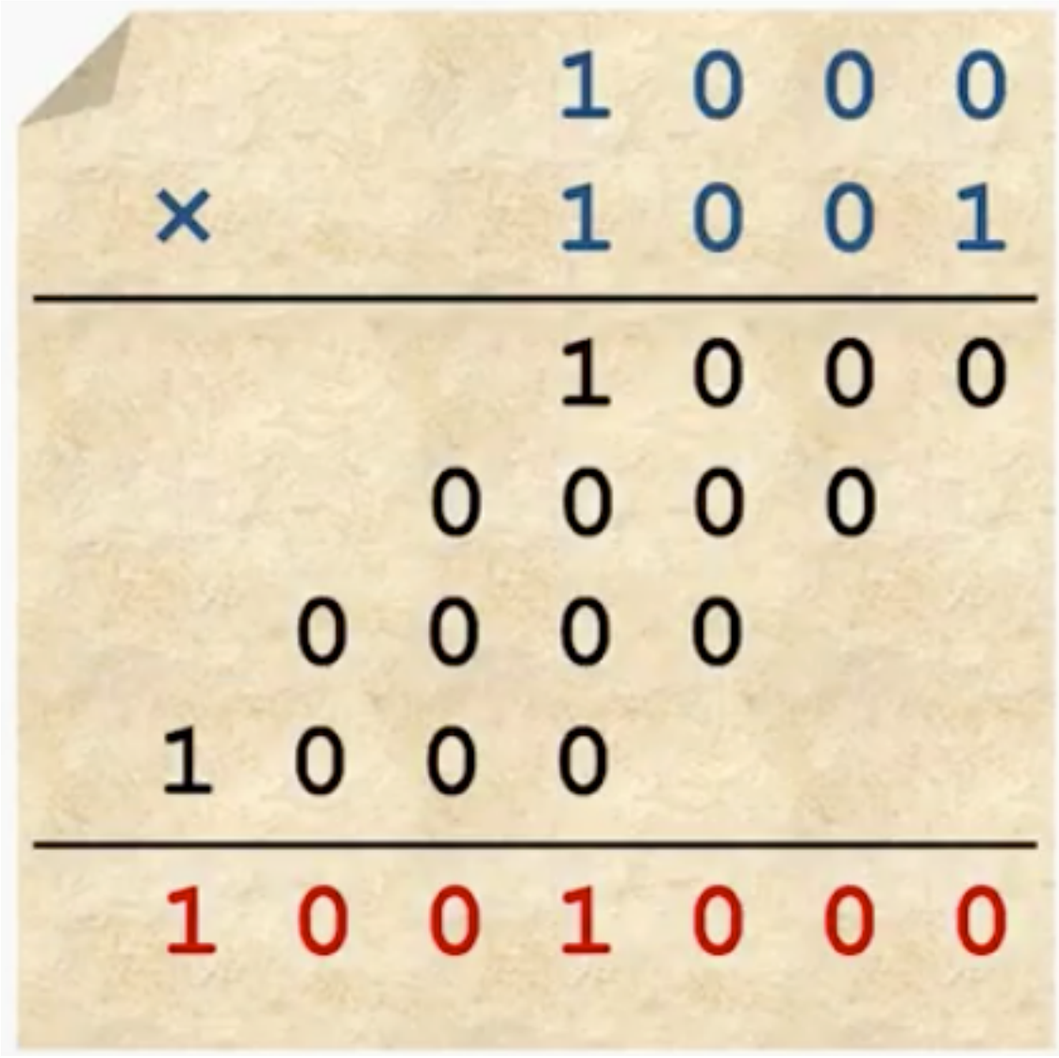
Hardware Implementation

- Different from what we do on a paper?



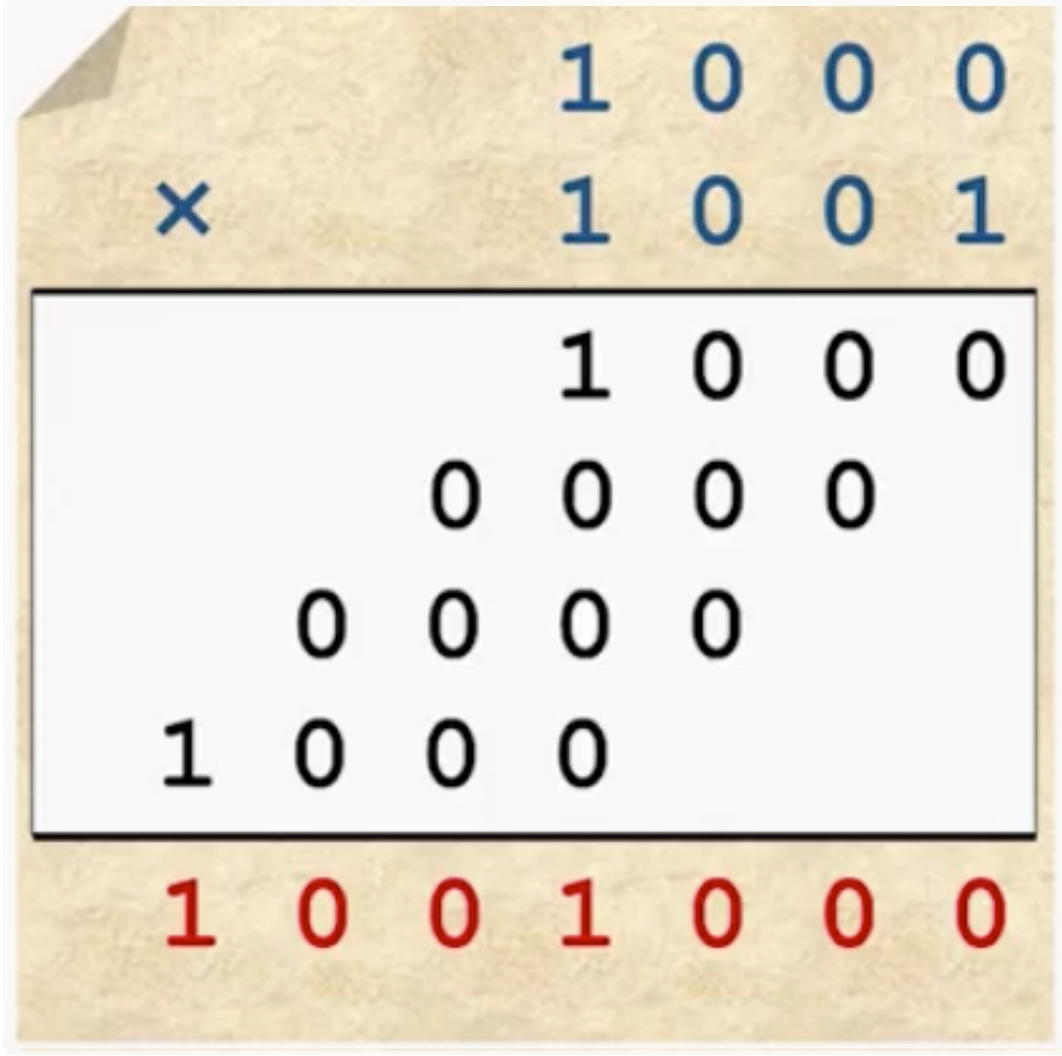
Hardware Implementation

- Different from what we do on a paper?
- Paper to computer
  - "Smaller paper size"
  - Only has space for
    - Multiplicand
    - Multiplier
    - Product
- We have eraser as well!



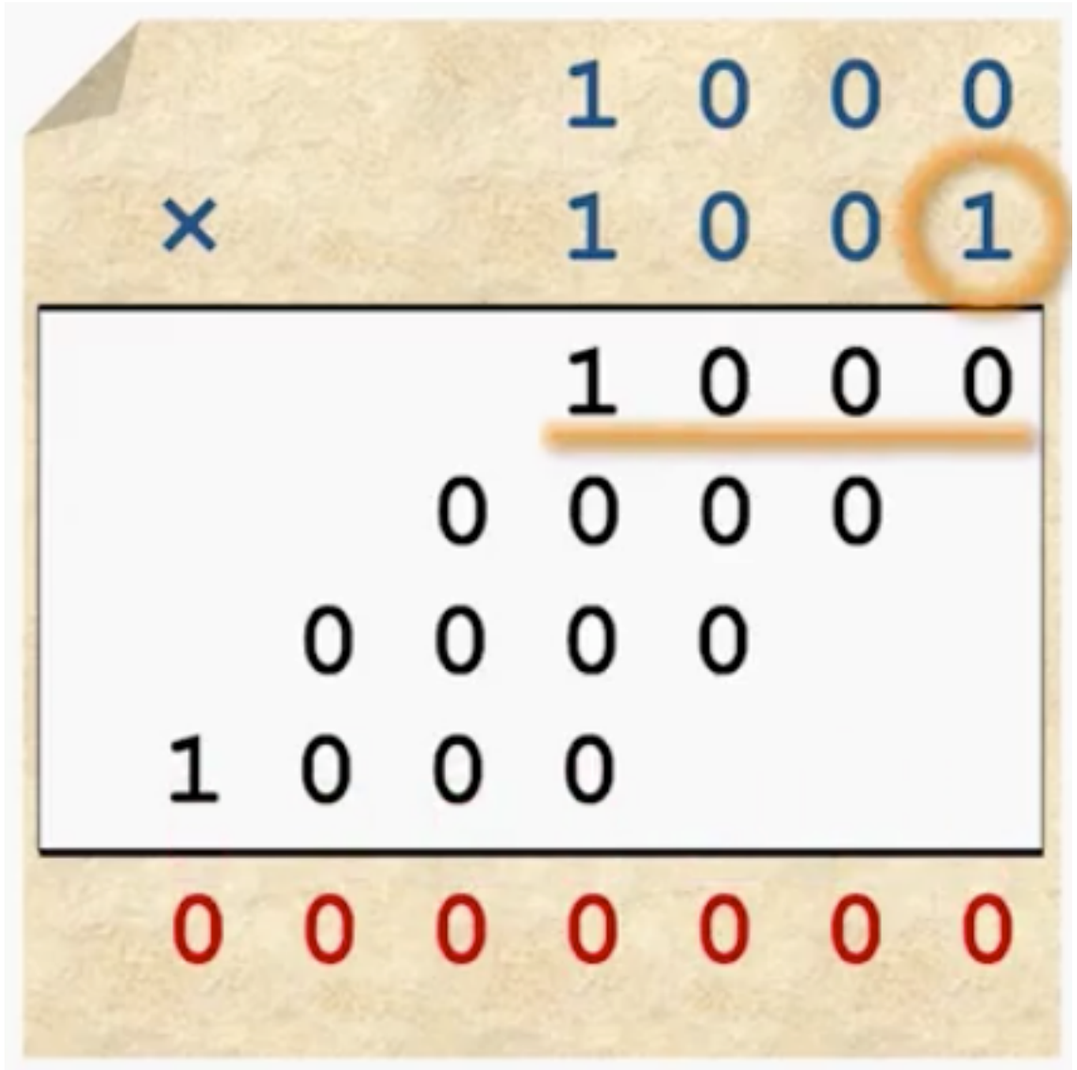
Hardware Implementation

- Different from what we do on a paper?
- Paper to computer
  - "Smaller paper size"
  - Only has space for
    - Multiplicand
    - Multiplier
    - Product
- We have eraser as well!

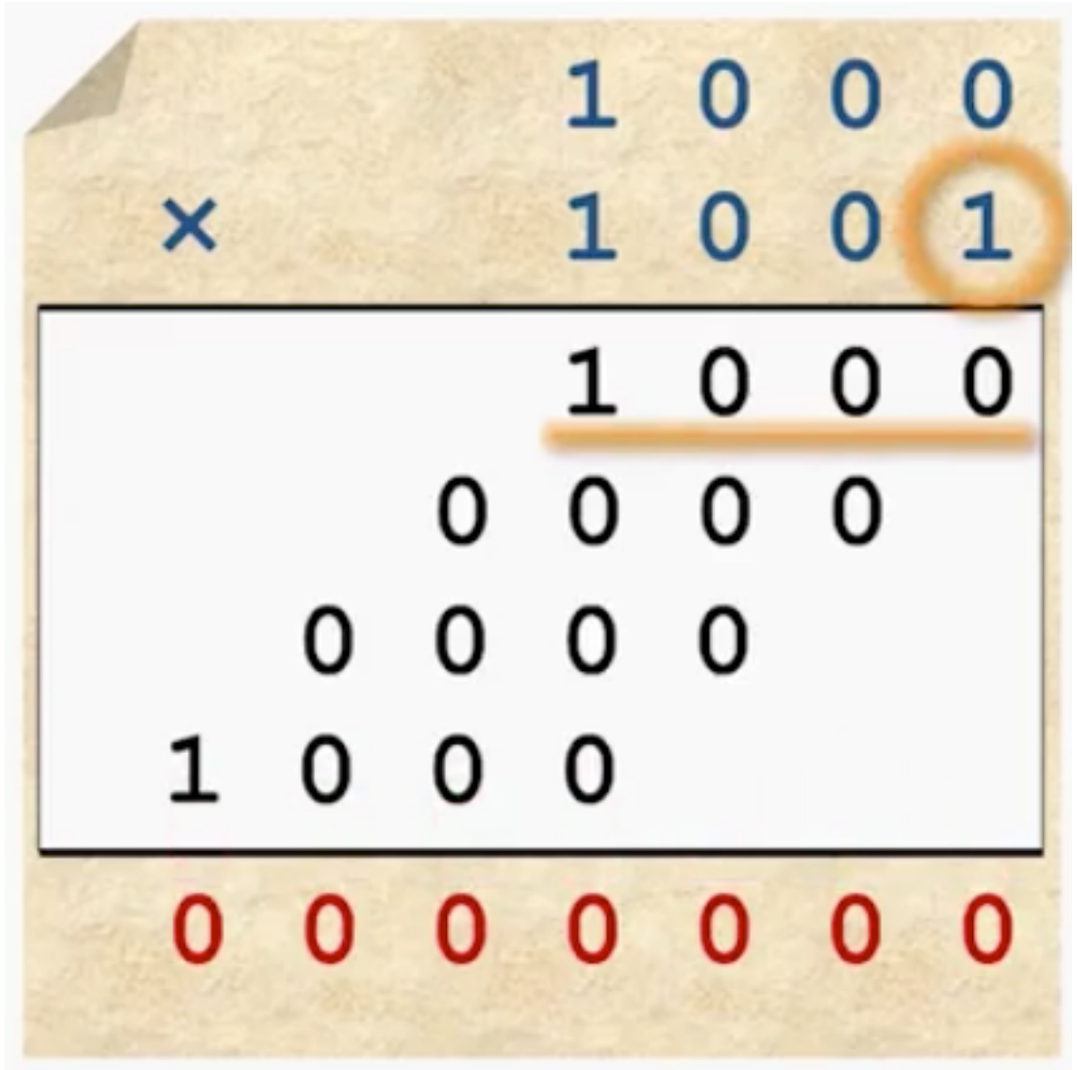


Hardware Implementation

Hardware Implementation

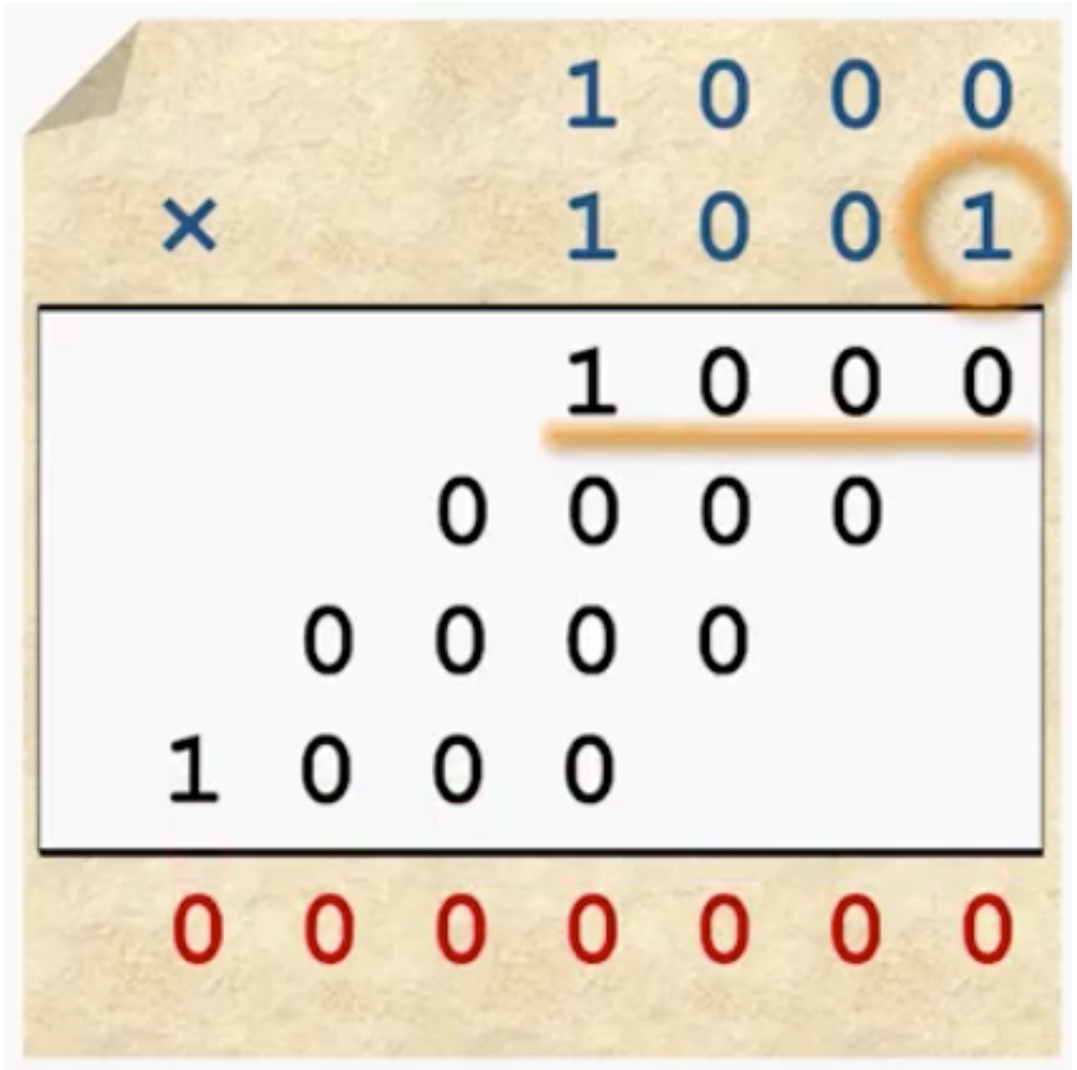


Hardware Implementation



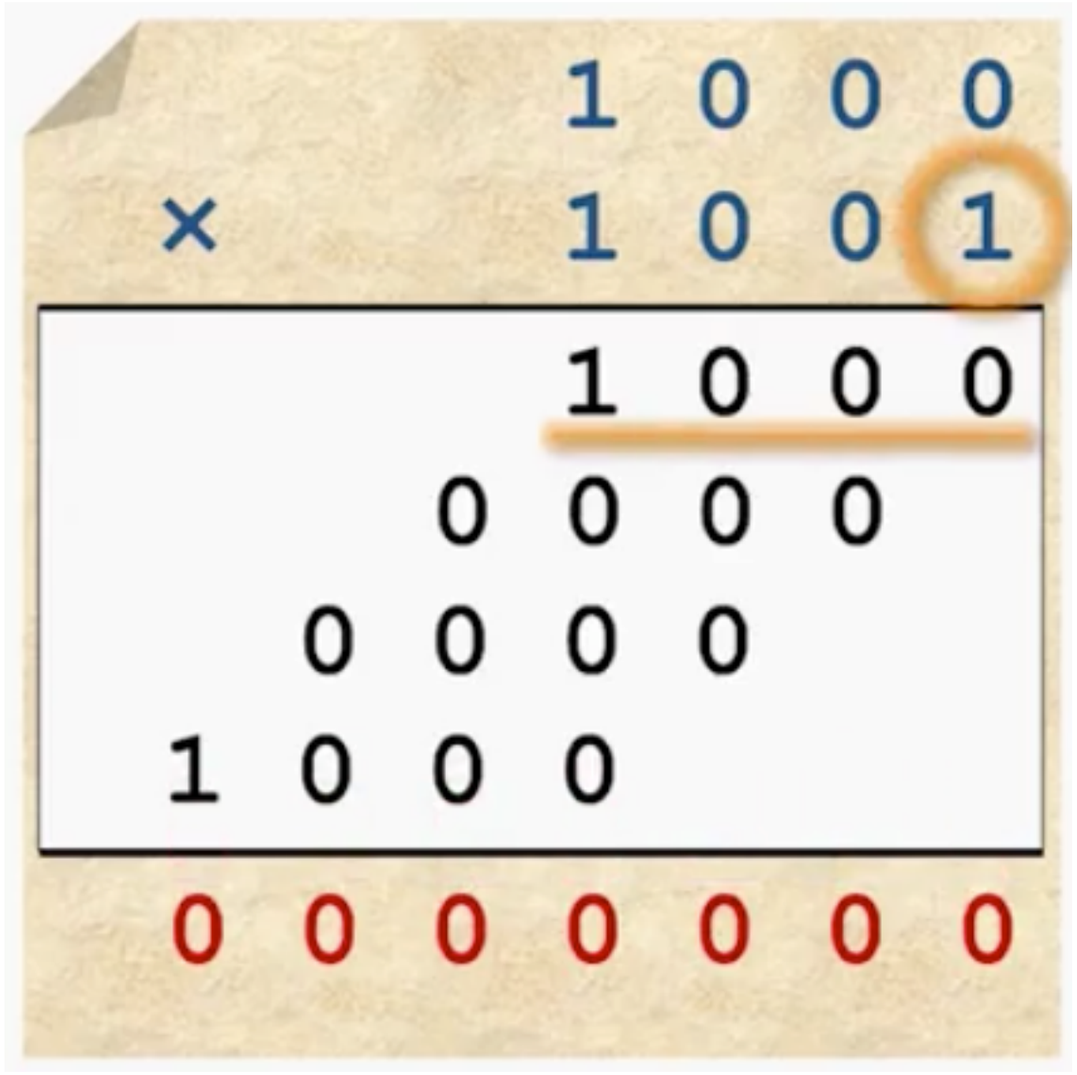
Hardware Implementation

- 1, Product marked as all-0



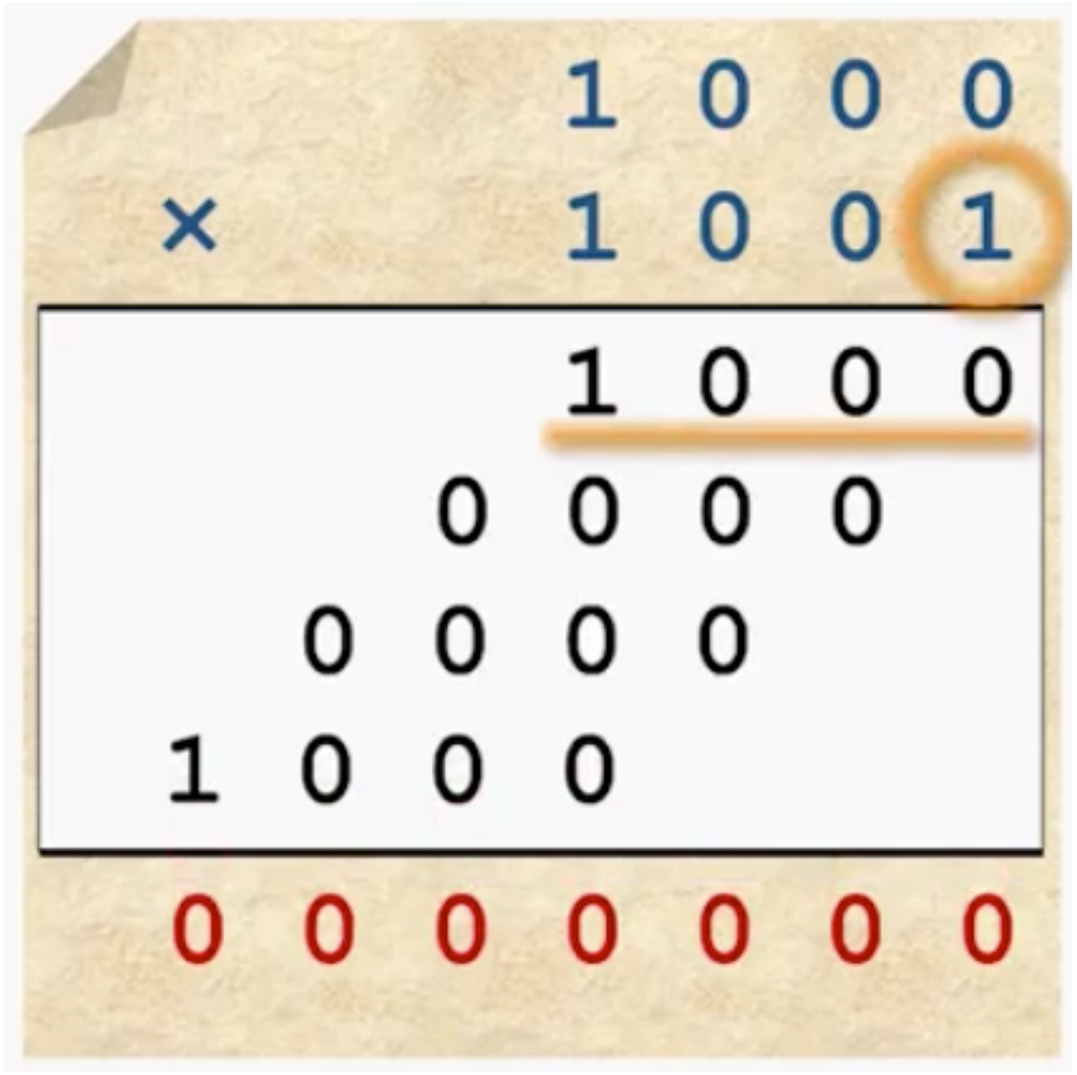
Hardware Implementation

- 1, Product marked as all-0
- 2, Multiplier = 1
  - Directly apply multiplicand
  - However, no space for it!
  - Use eraser, store it as product (temporary!)



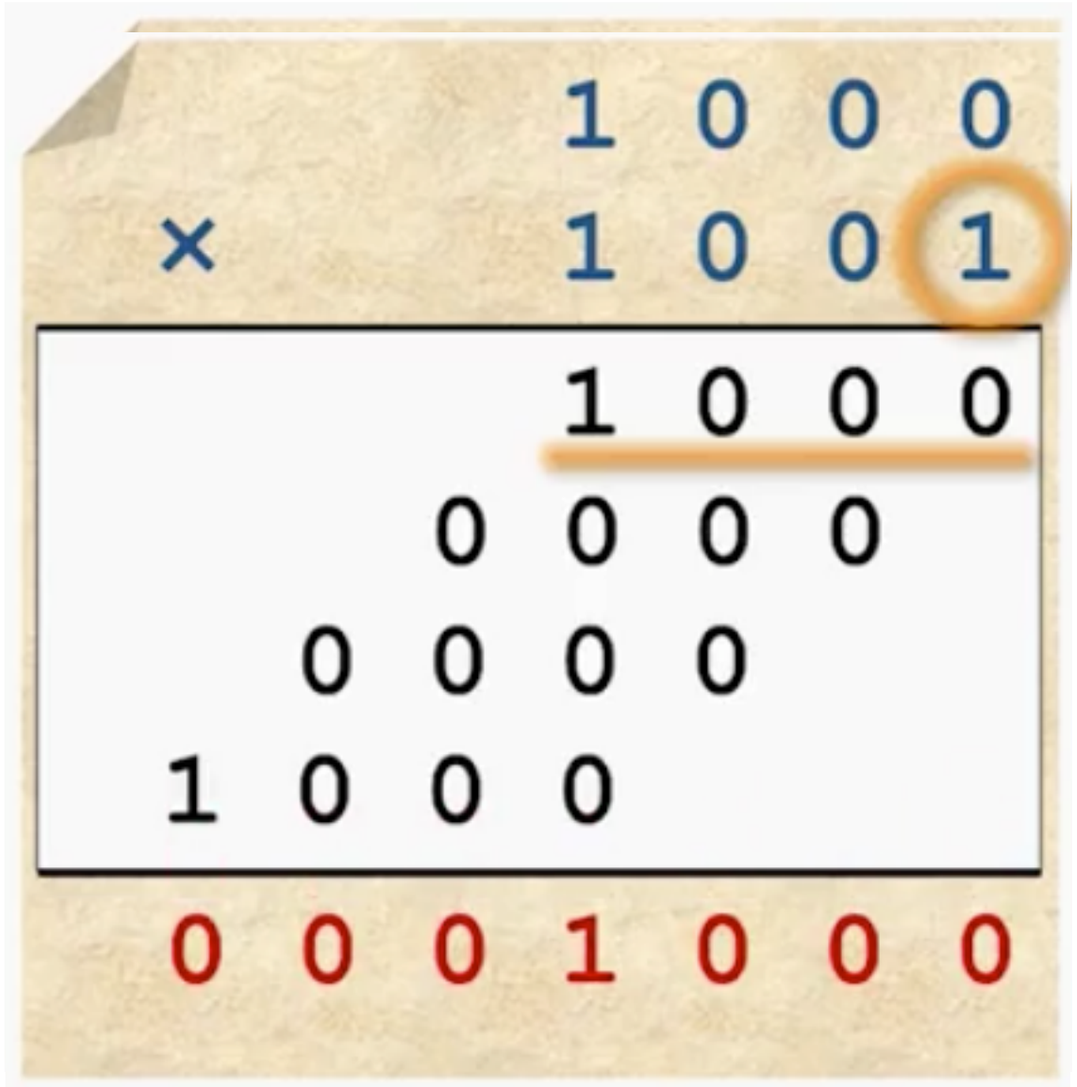
Hardware Implementation

- 1, Product marked as all-0
- 2, Multiplier = 1
  - Directly apply multiplicand
  - However, no space for it!
  - Use eraser, store it as product (temporary!)
- 3, We need shift-to-left now
  - No space for it anymore…
  - How to proceed?



Hardware Implementation

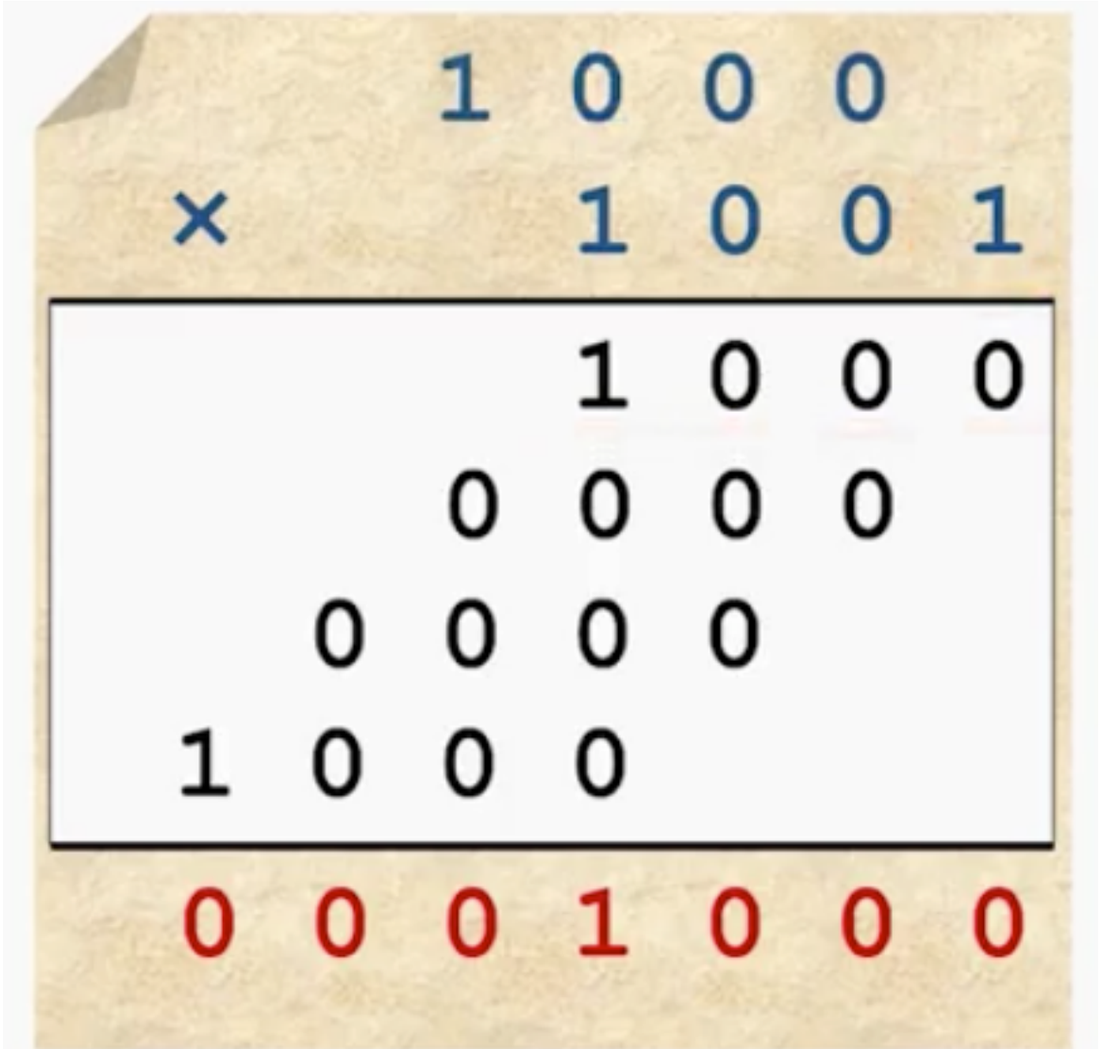
- 1, Product marked as all-0
- 2, Multiplier = 1
  - Directly apply multiplicand
  - However, no space for it!
  - Use eraser, store it as product (temporary!)
- 3, We need shift-to-left now
  - No space for it anymore…
  - How to proceed?



Hardware Implementation

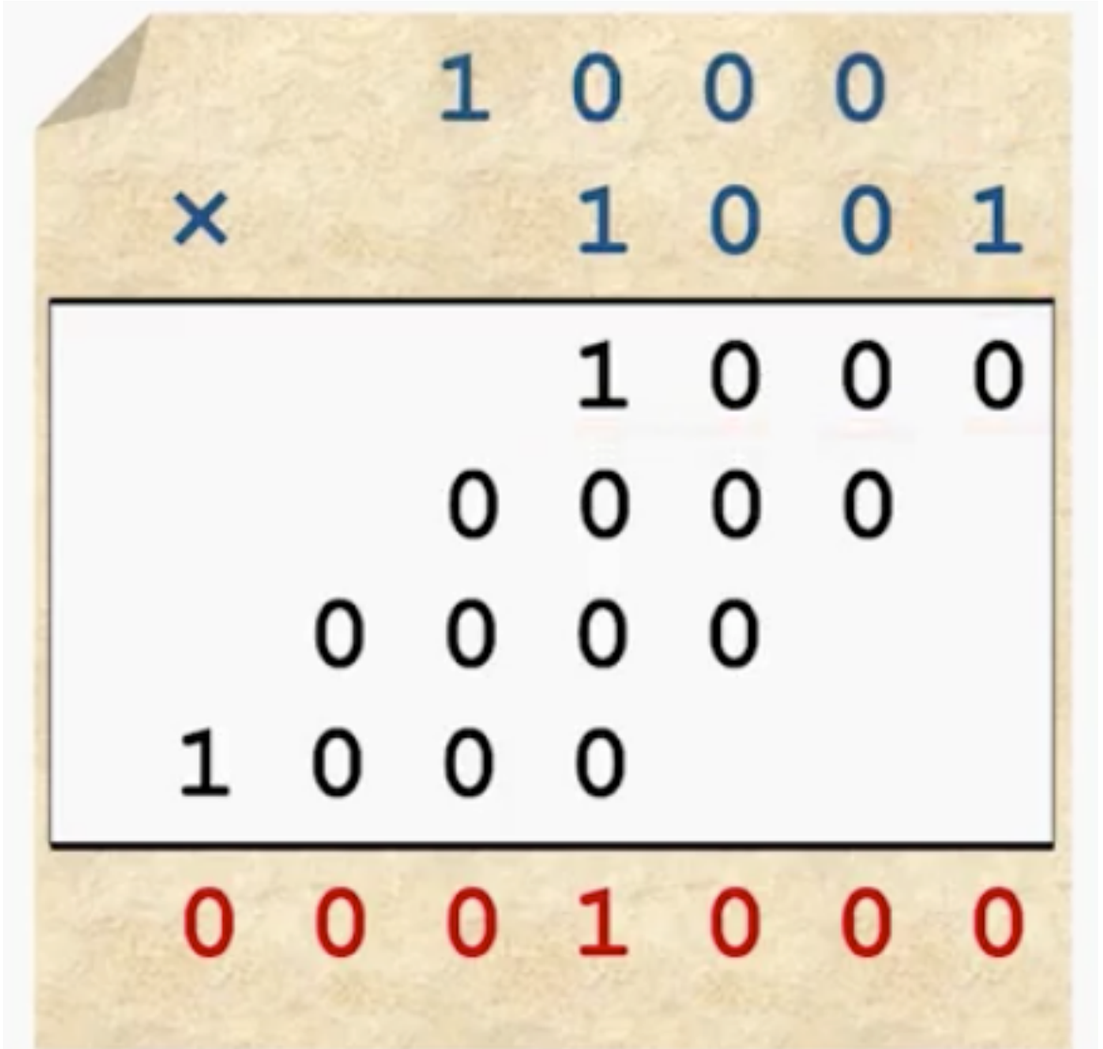
Hardware Implementation

- 4, Shifting the multiplicand instead
  - Calculate as step 1-3
  - If all-0
    - Just like we already added it to the product, or did nothing
  - If not all-0
    - Adding it to the product



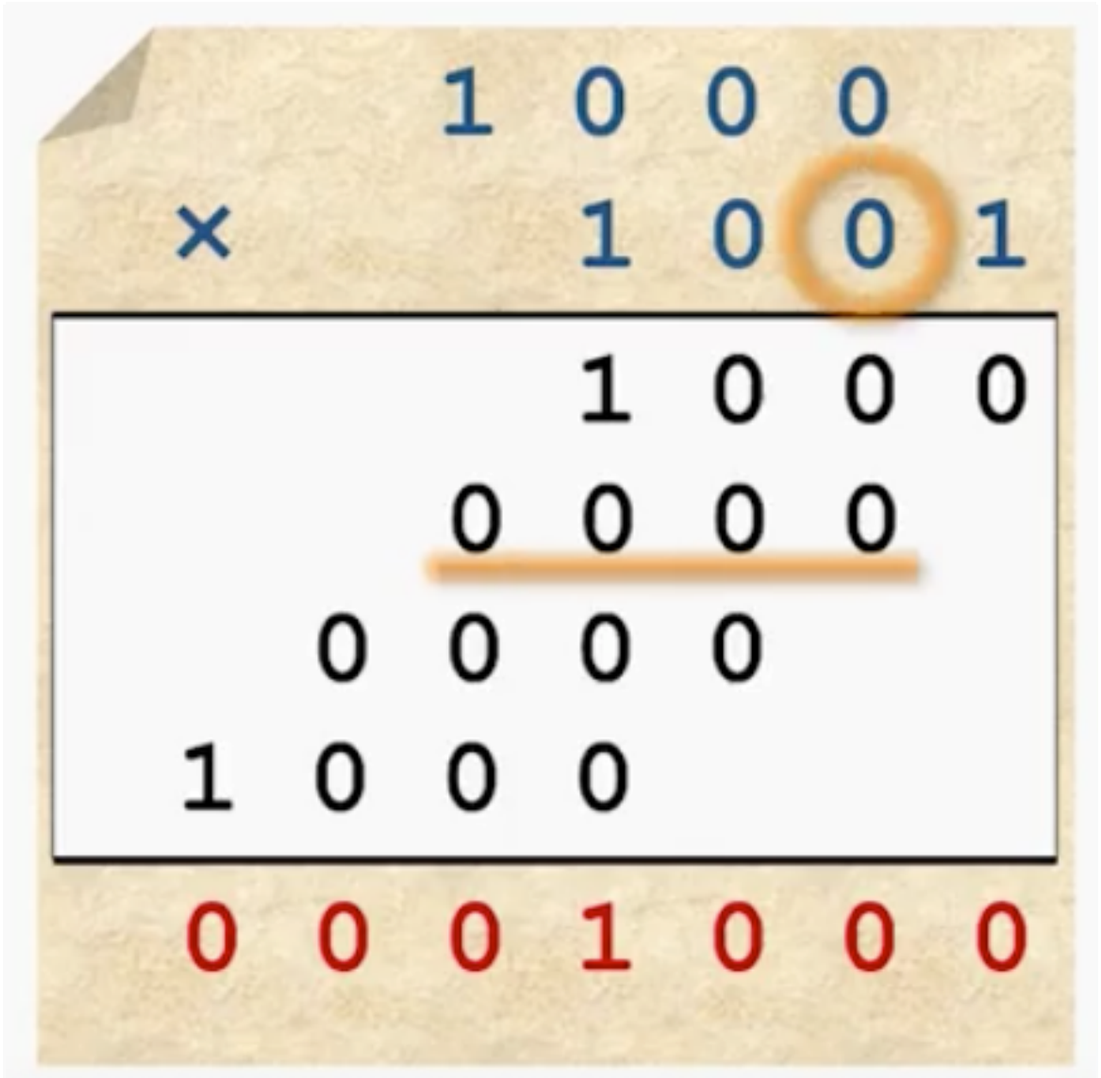
Hardware Implementation

- 4, Shifting the multiplicand instead
  - Calculate as step 1-3
  - If all-0
    - Just like we already added it to the product, or did nothing
  - If not all-0
    - Adding it to the product
- 5, Repeat step 4



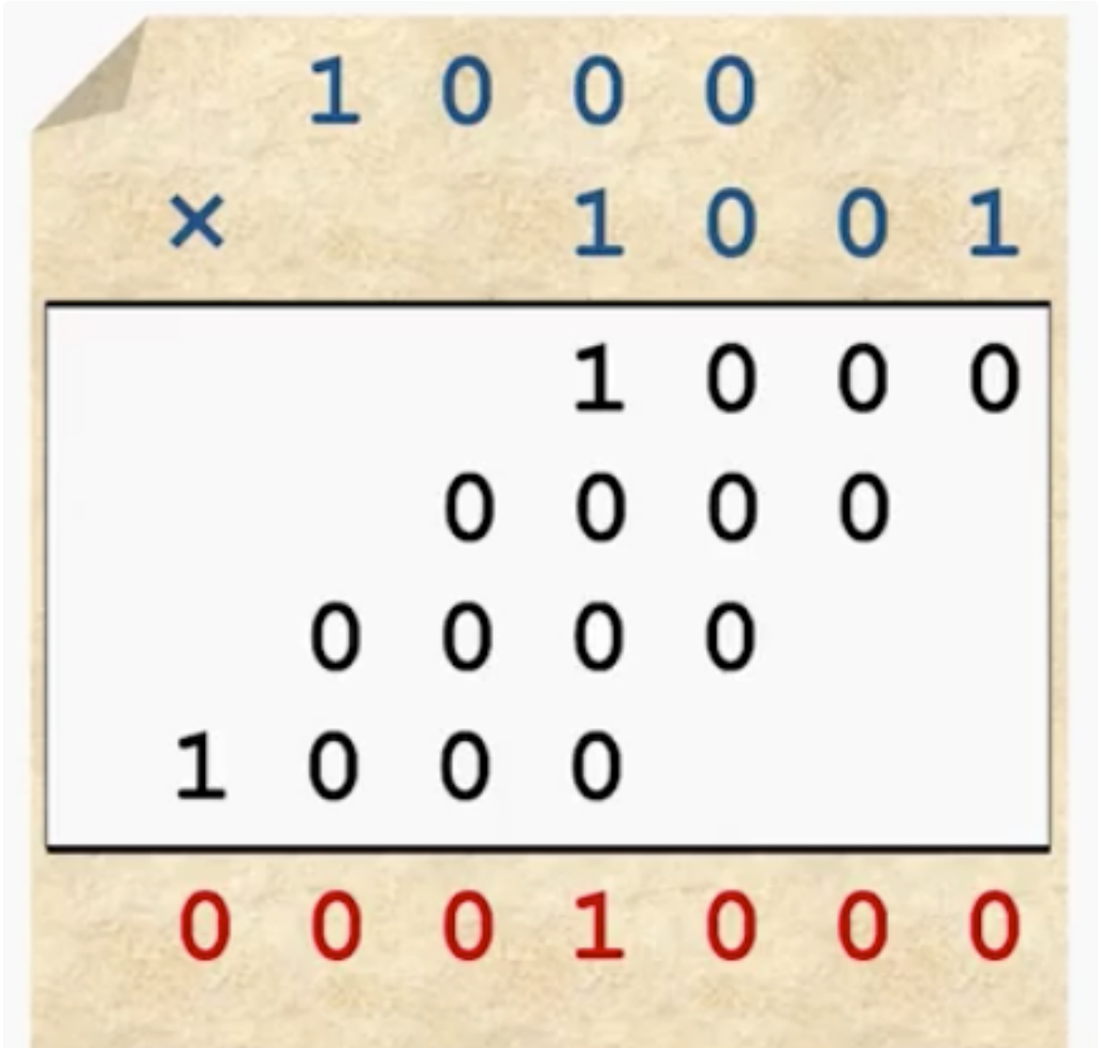
Hardware Implementation

- 4, Shifting the multiplicand instead
  - Calculate as step 1-3
  - If all-0
    - Just like we already added it to the product, or did nothing
  - If not all-0
    - Adding it to the product
- 5, Repeat step 4



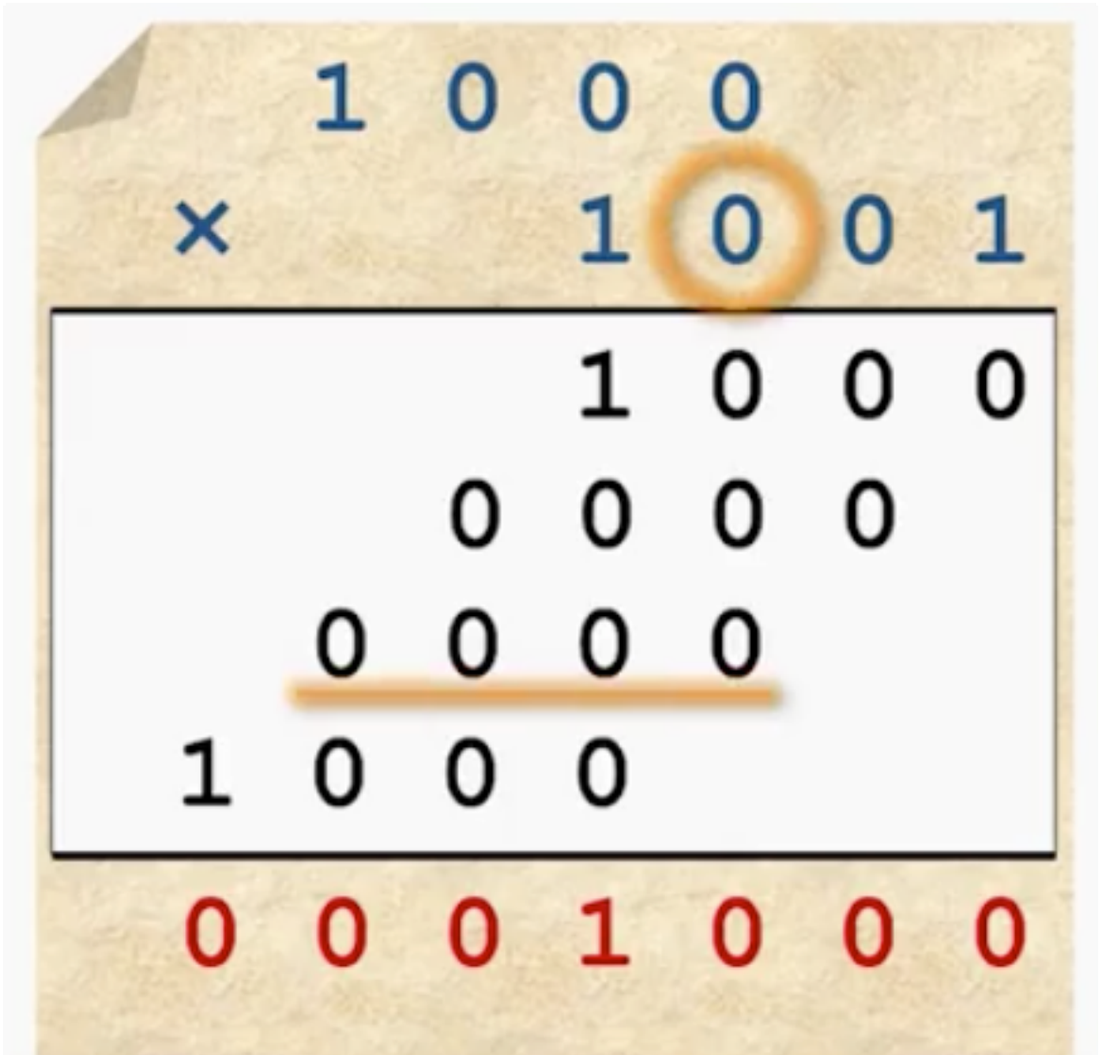
Hardware Implementation

- 4, Shifting the multiplicand instead
  - Calculate as step 1-3
  - If all-0
    - Just like we already added it to the product, or did nothing
  - If not all-0
    - Adding it to the product
- 5, Repeat step 4



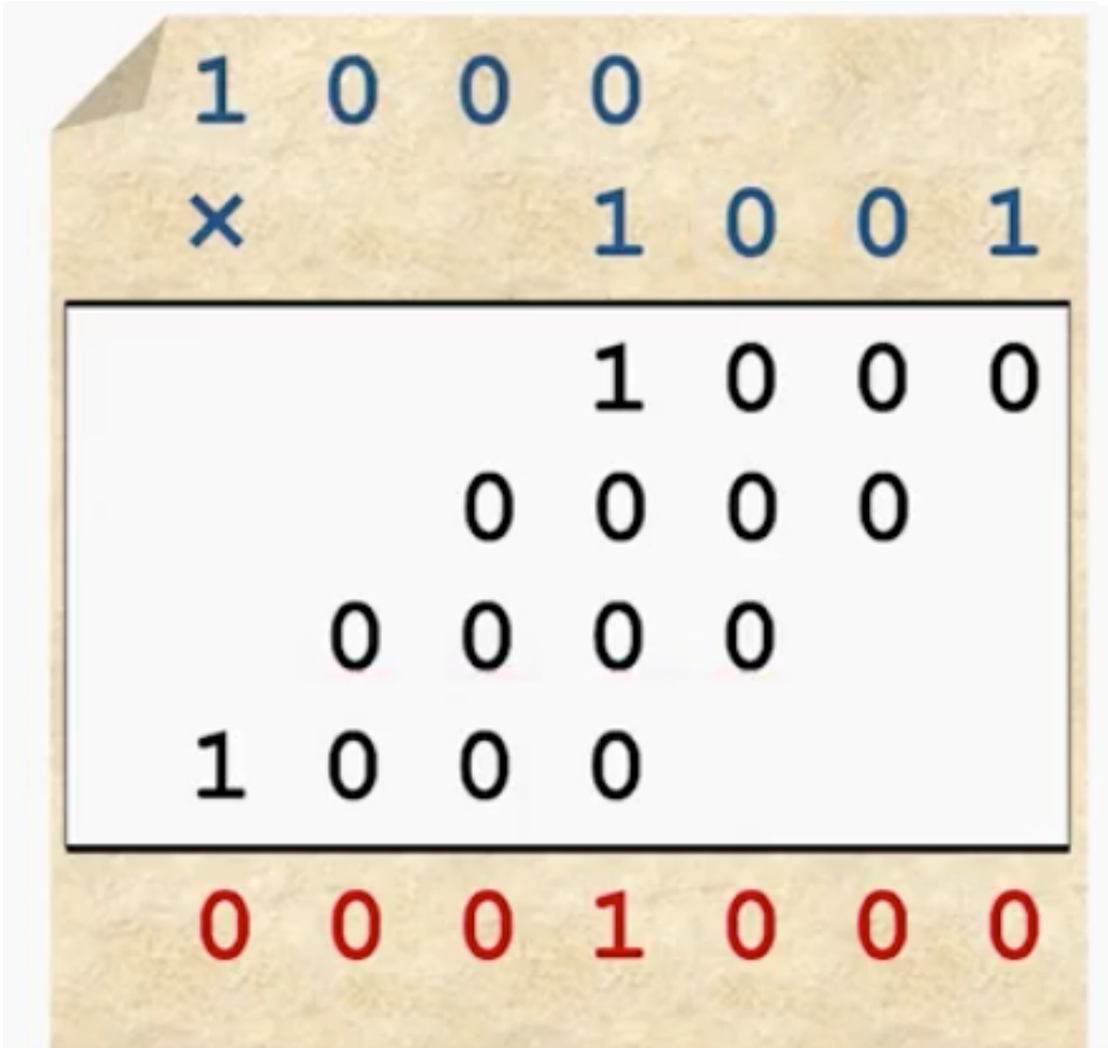
Hardware Implementation

- 4, Shifting the multiplicand instead
  - Calculate as step 1-3
  - If all-0
    - Just like we already added it to the product, or did nothing
  - If not all-0
    - Adding it to the product
- 5, Repeat step 4



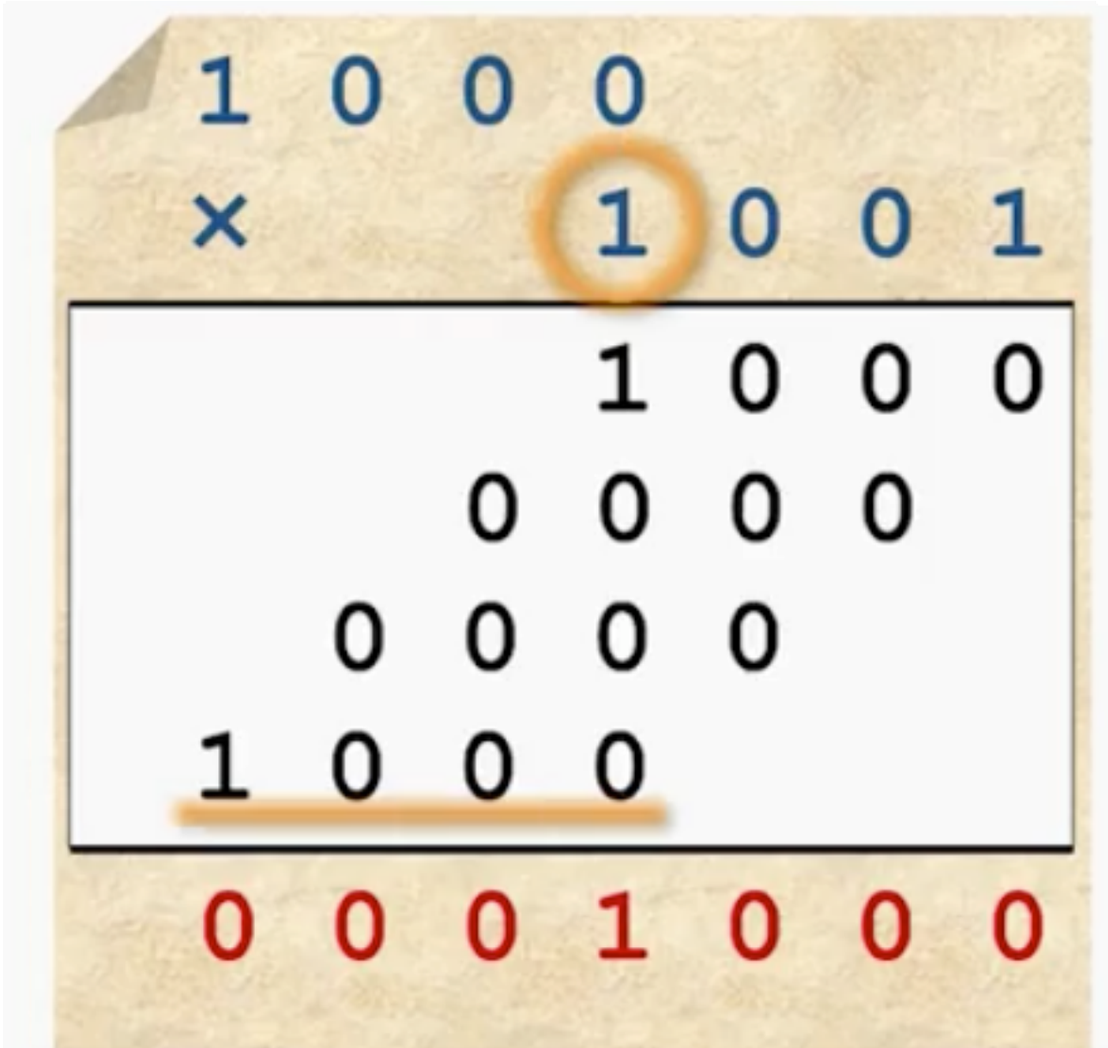
Hardware Implementation

- 4, Shifting the multiplicand instead
  - Calculate as step 1-3
  - If all-0
    - Just like we already added it to the product, or did nothing
  - If not all-0
    - Adding it to the product
- 5, Repeat step 4



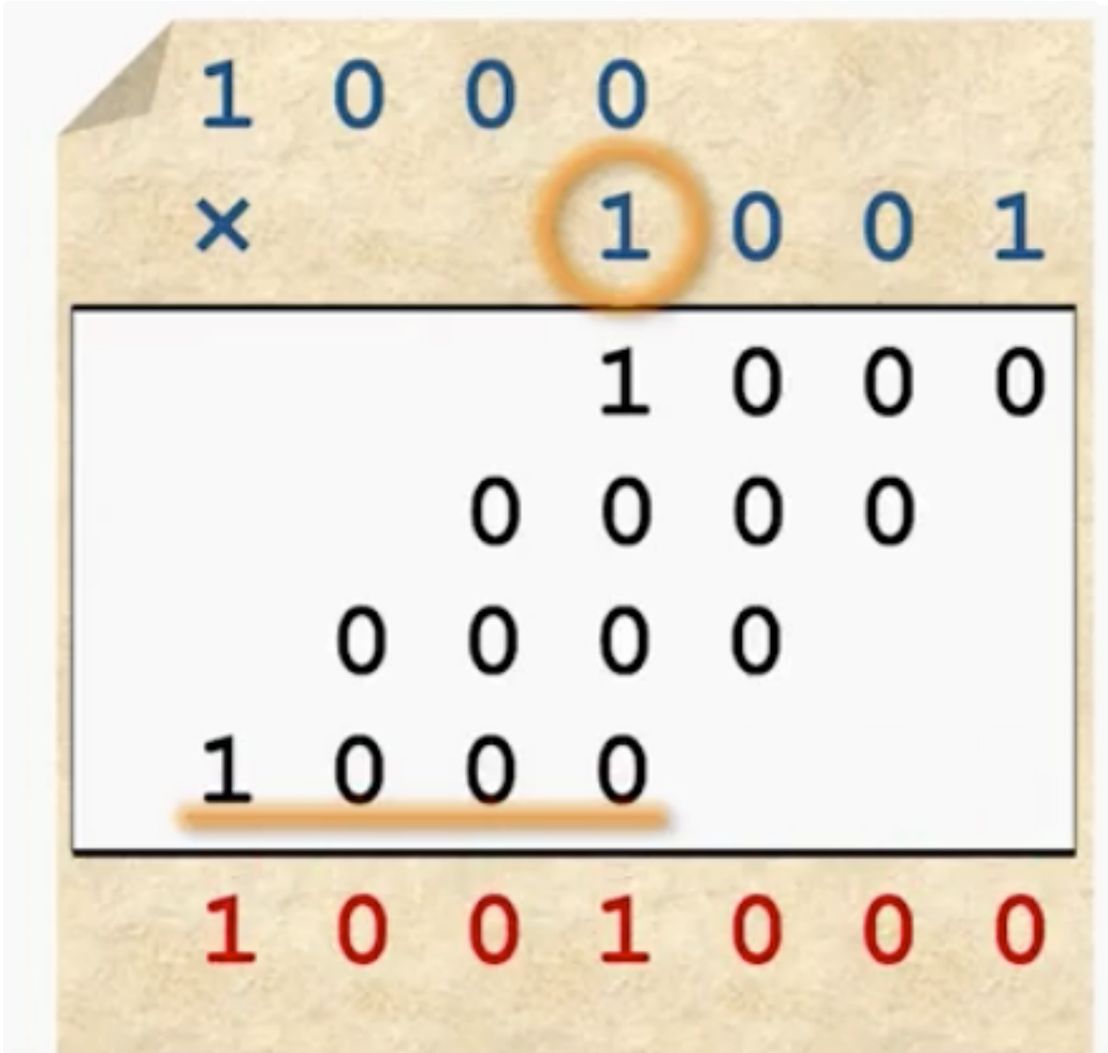
Hardware Implementation

- 4, Shifting the multiplicand instead
  - Calculate as step 1-3
  - If all-0
    - Just like we already added it to the product, or did nothing
  - If not all-0
    - Adding it to the product
- 5, Repeat step 4

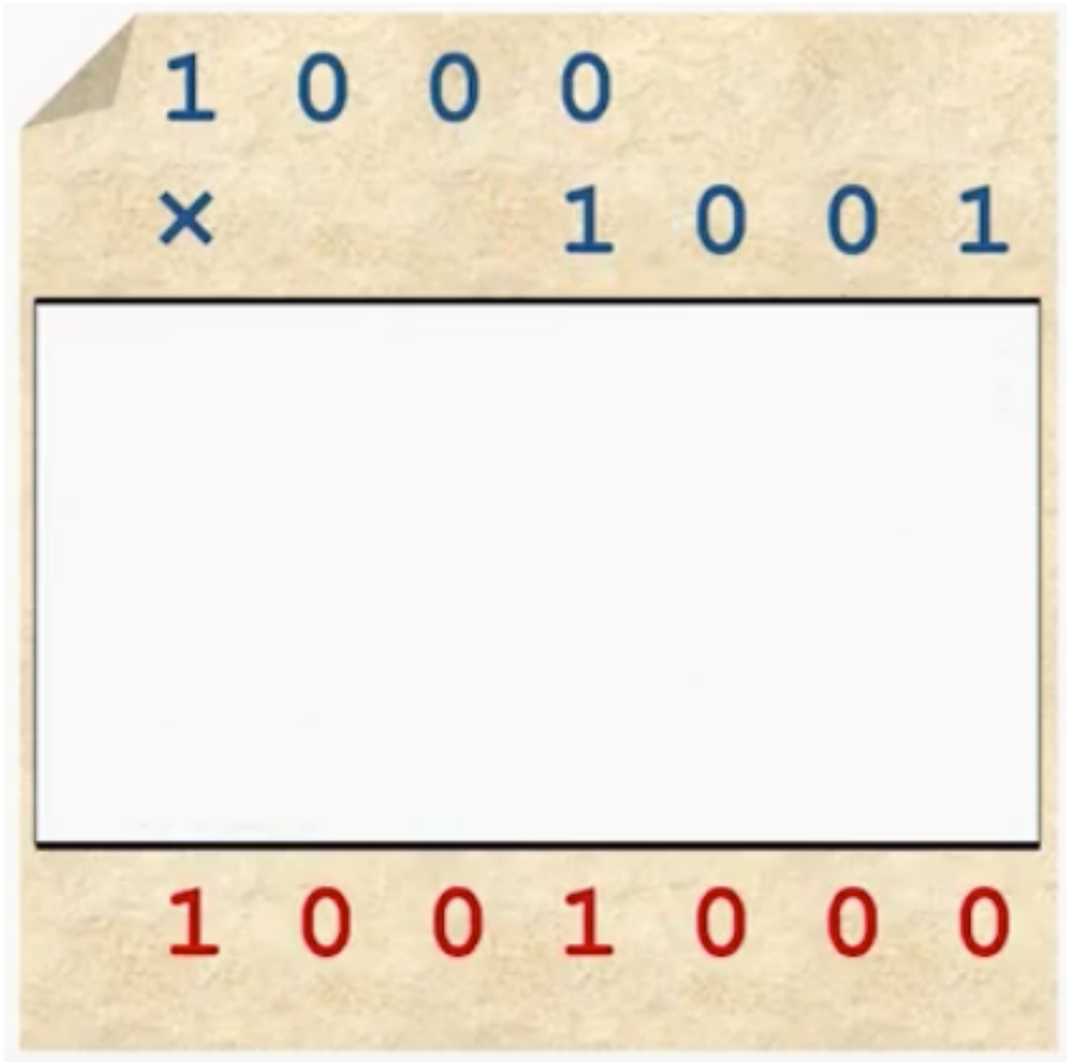


Hardware Implementation

- 4, Shifting the multiplicand instead
  - Calculate as step 1-3
  - If all-0
    - Just like we already added it to the product, or did nothing
  - If not all-0
    - Adding it to the product
- 5, Repeat step 4

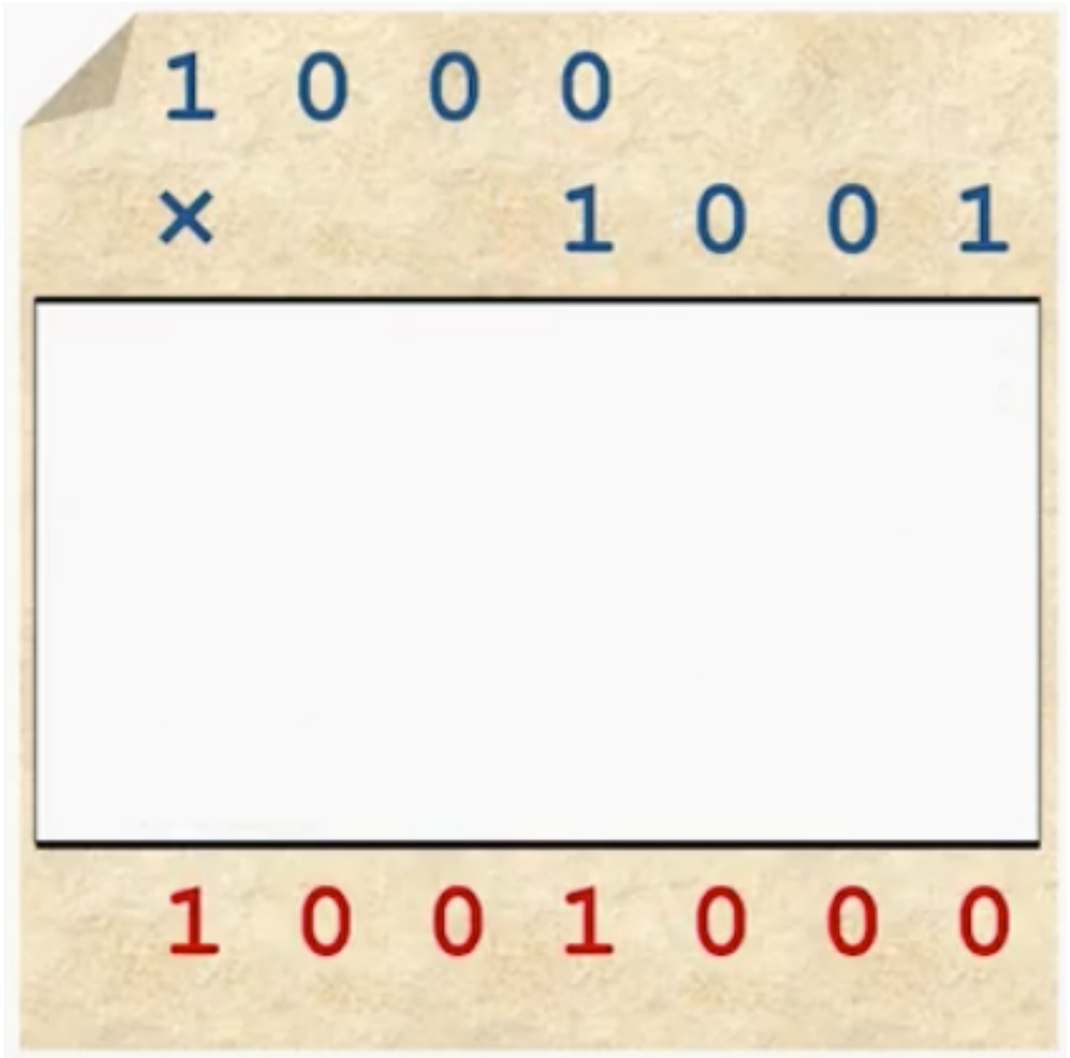


Hardware Implementation



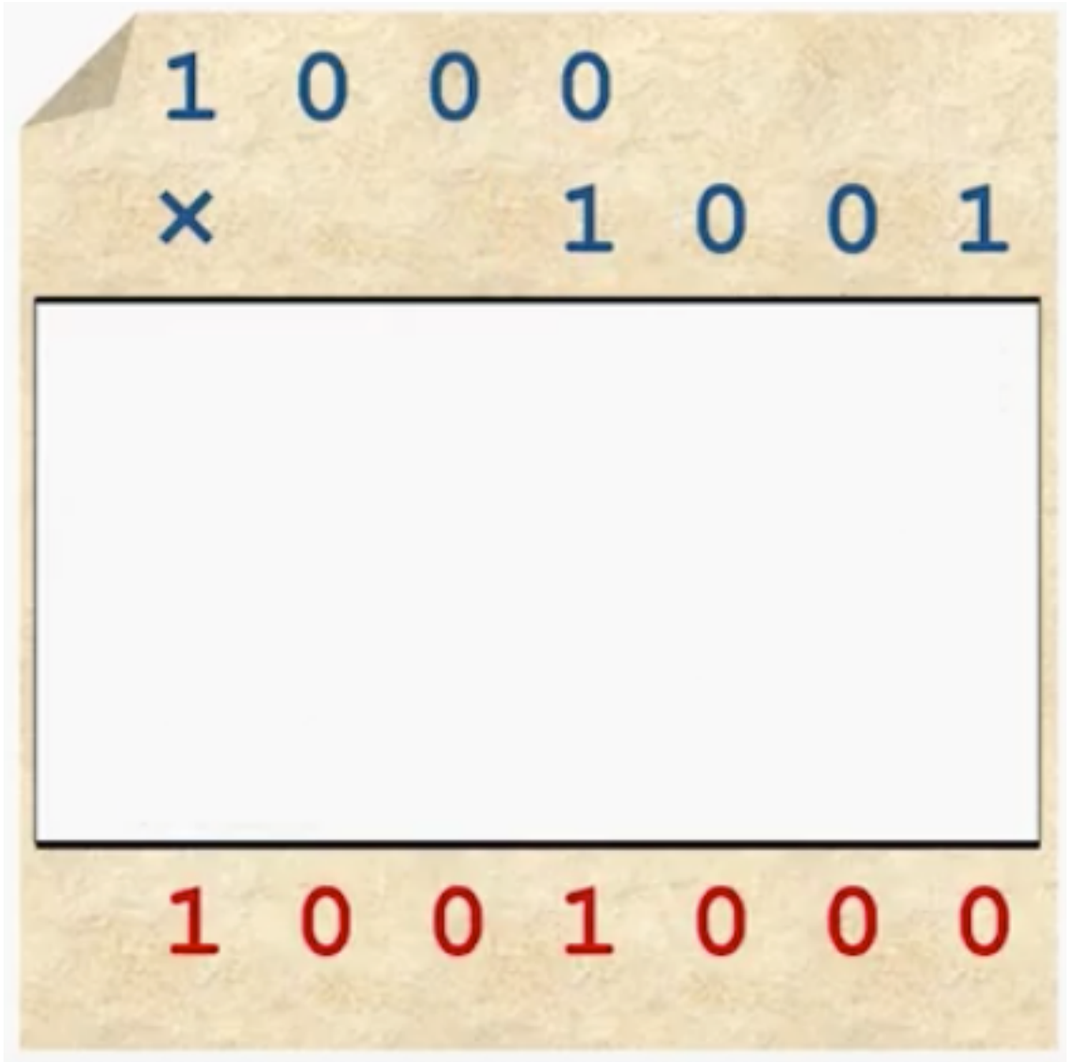
Hardware Implementation

- 6, All bit of multiplier checked?



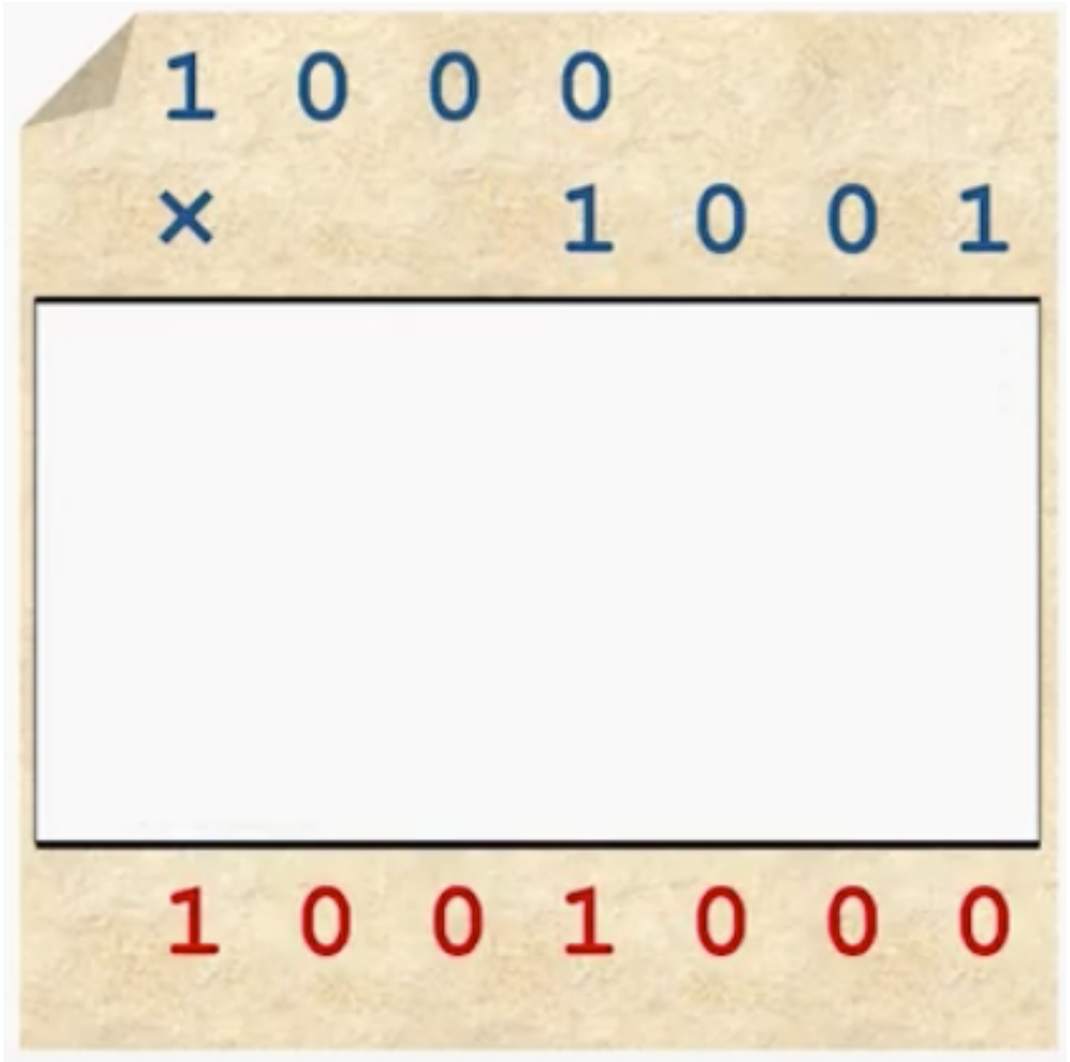
Hardware Implementation

- 6, All bit of multiplier checked?
  - No, repeat step 4



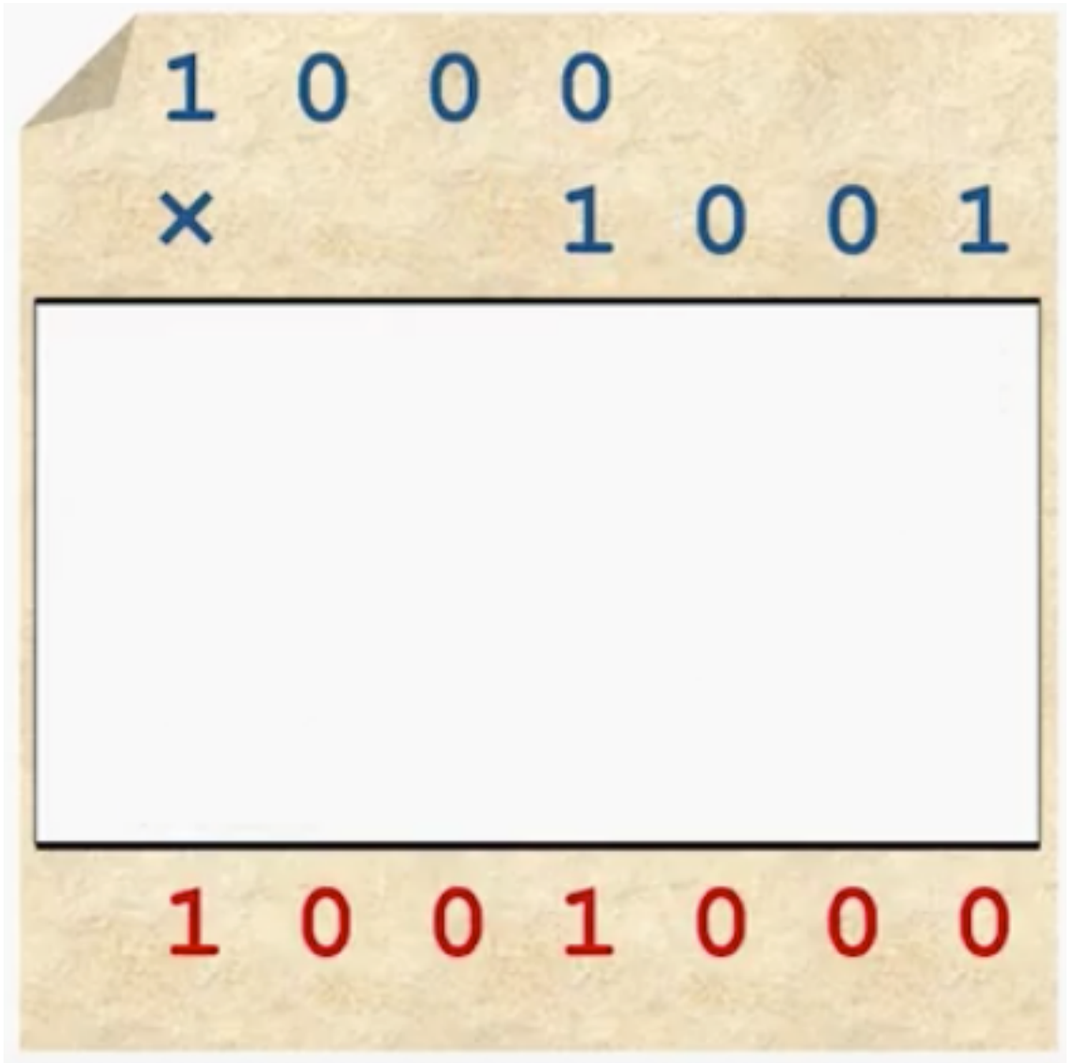
Hardware Implementation

- 6, All bit of multiplier checked?
  - No, repeat step 4
  - Yes, done!



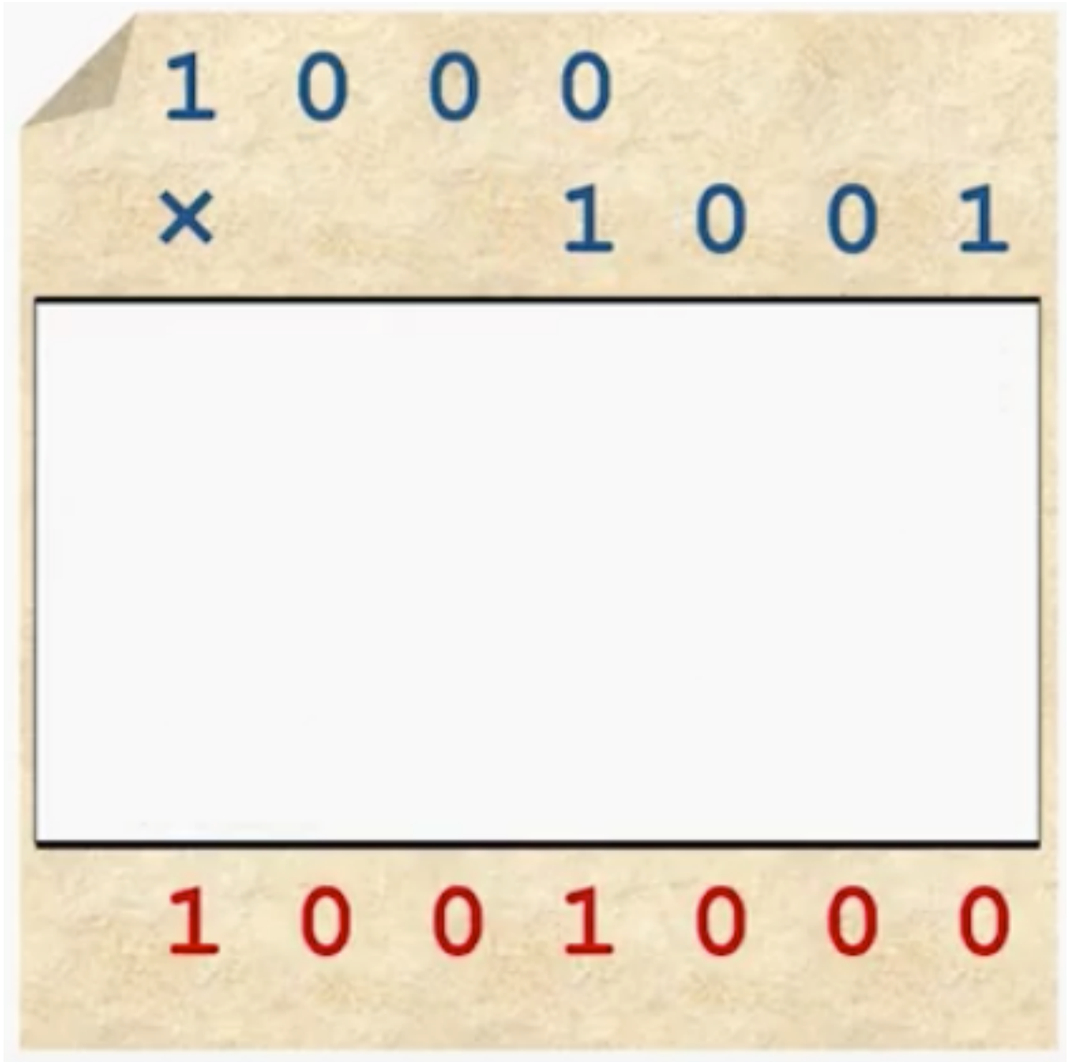
Hardware Implementation

- 6, All bit of multiplier checked?
  - No, repeat step 4
  - Yes, done!
- The proposed method does not need extra space



Hardware Implementation

- 6, All bit of multiplier checked?
  - No, repeat step 4
  - Yes, done!
- The proposed method does not need extra space
- Suitable for hardware implementation



Circuit Implementation

$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$

Circuit Implementation

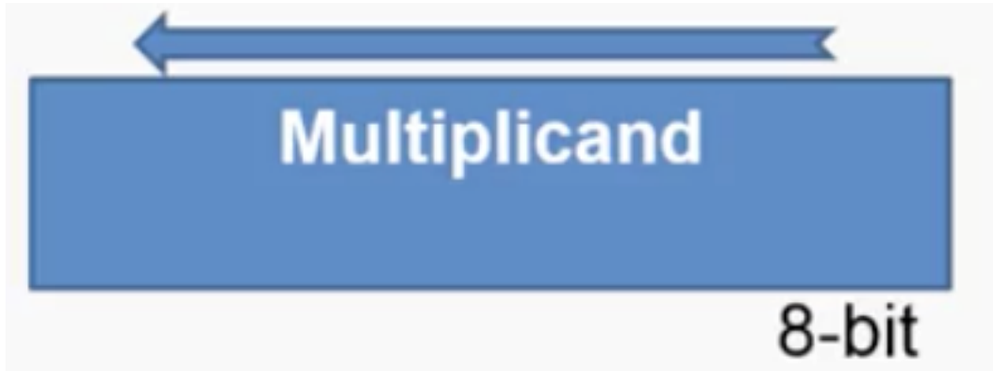
- 8-bit register for multiplicand

$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$

Circuit Implementation

- 8-bit register for multiplicand

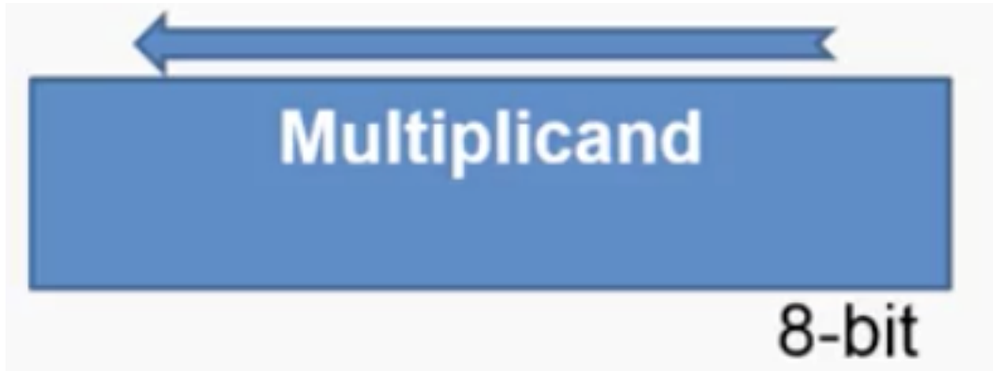
$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$



Circuit Implementation

- 8-bit register for multiplicand
  - With left shift function

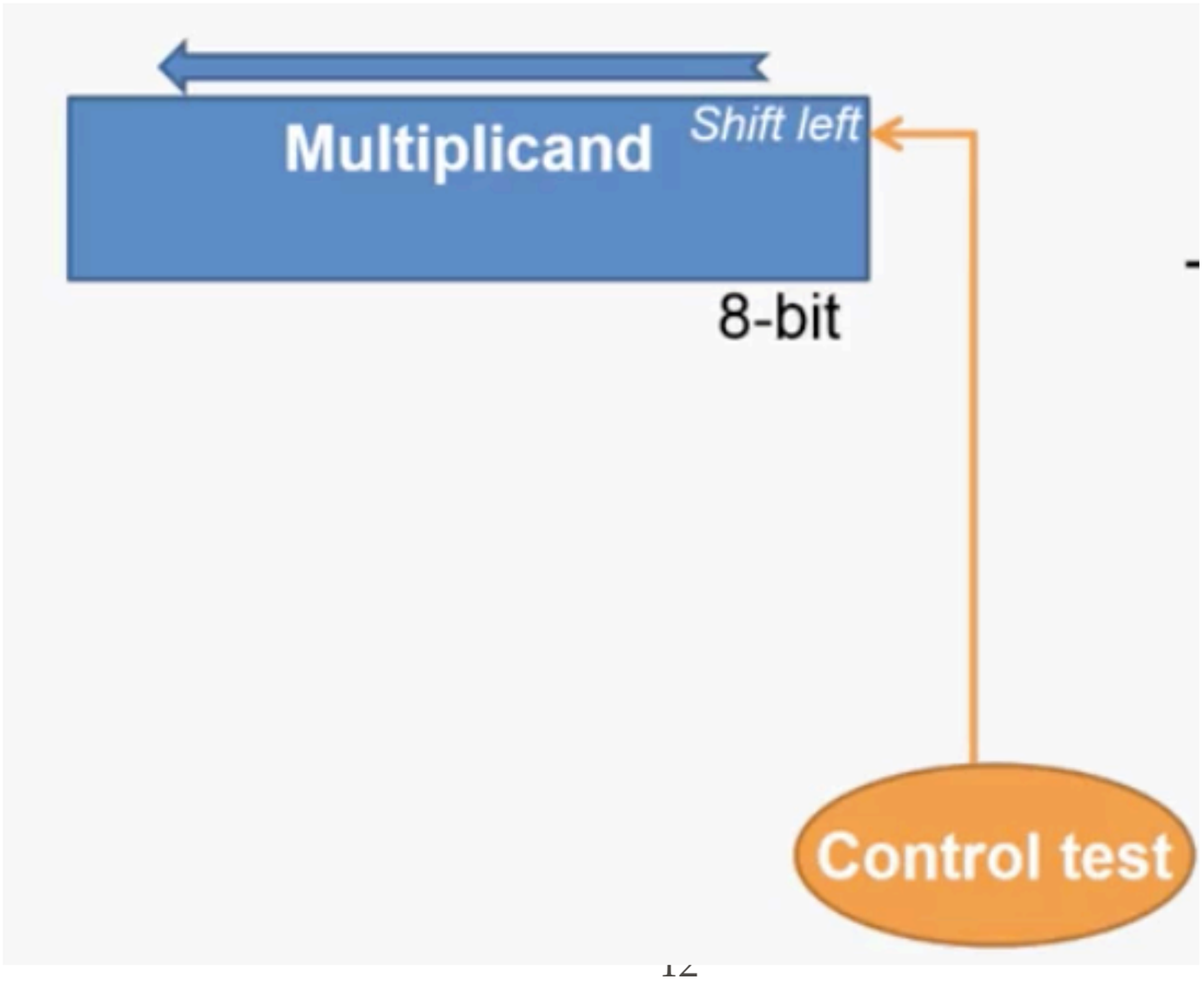
$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$



Circuit Implementation

- 8-bit register for multiplicand
  - With left shift function

$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$



Circuit Implementation

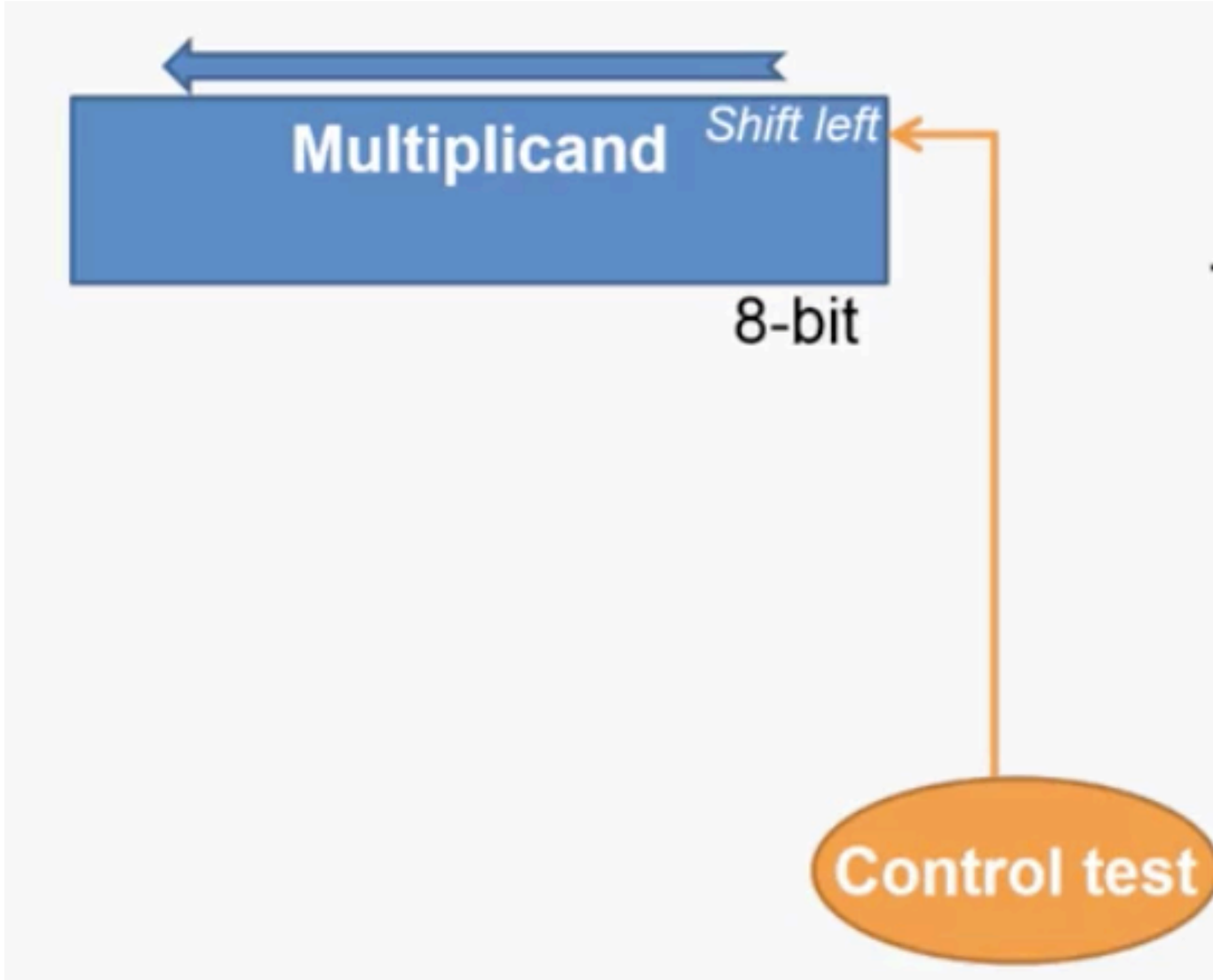
- 8-bit register for multiplicand
  - With left shift function

$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$

Circuit Implementation

- 8-bit register for multiplicand
  - With left shift function

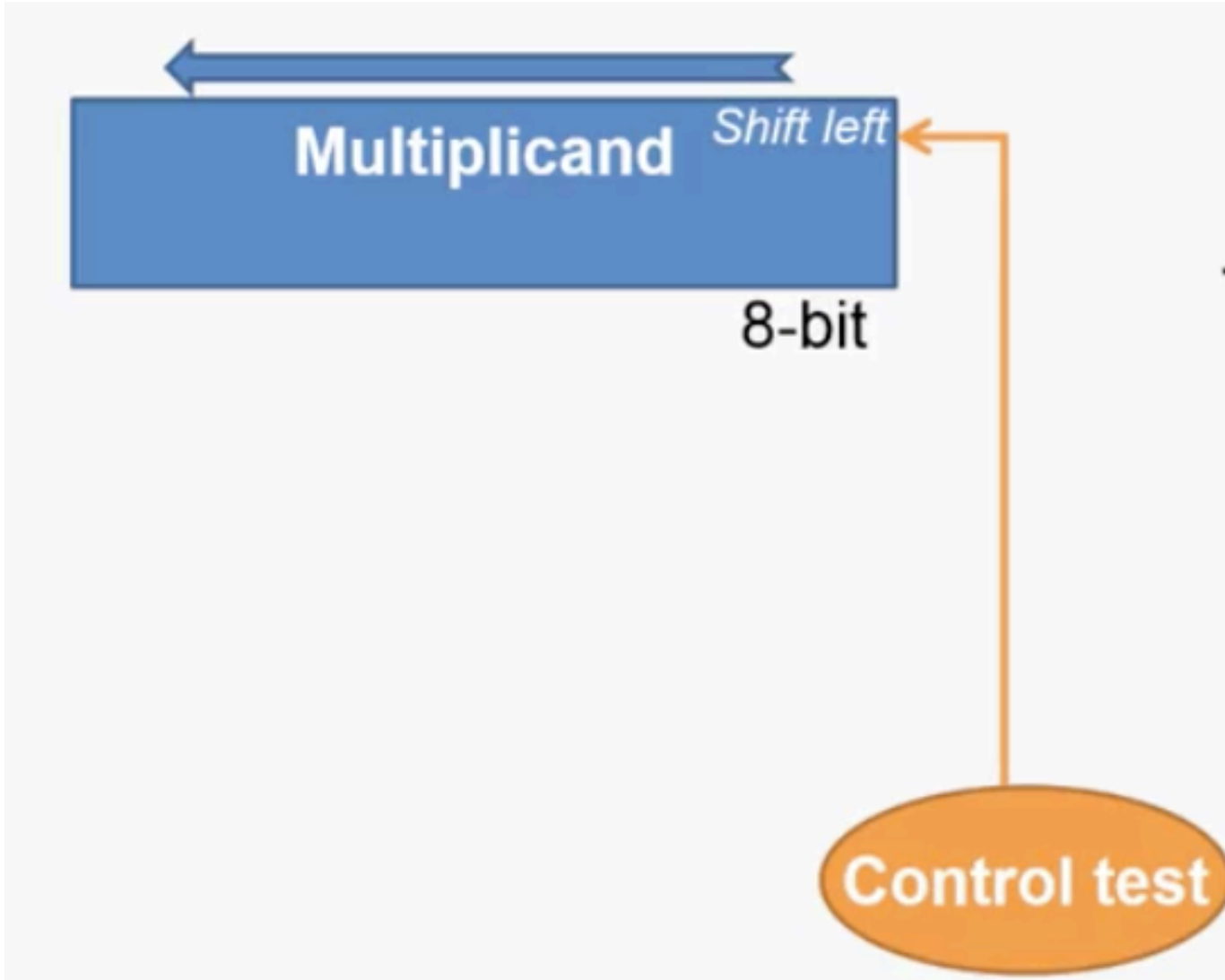
$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$



Circuit Implementation

- 8-bit register for multiplicand
  - With left shift function
- 8-bit register for product

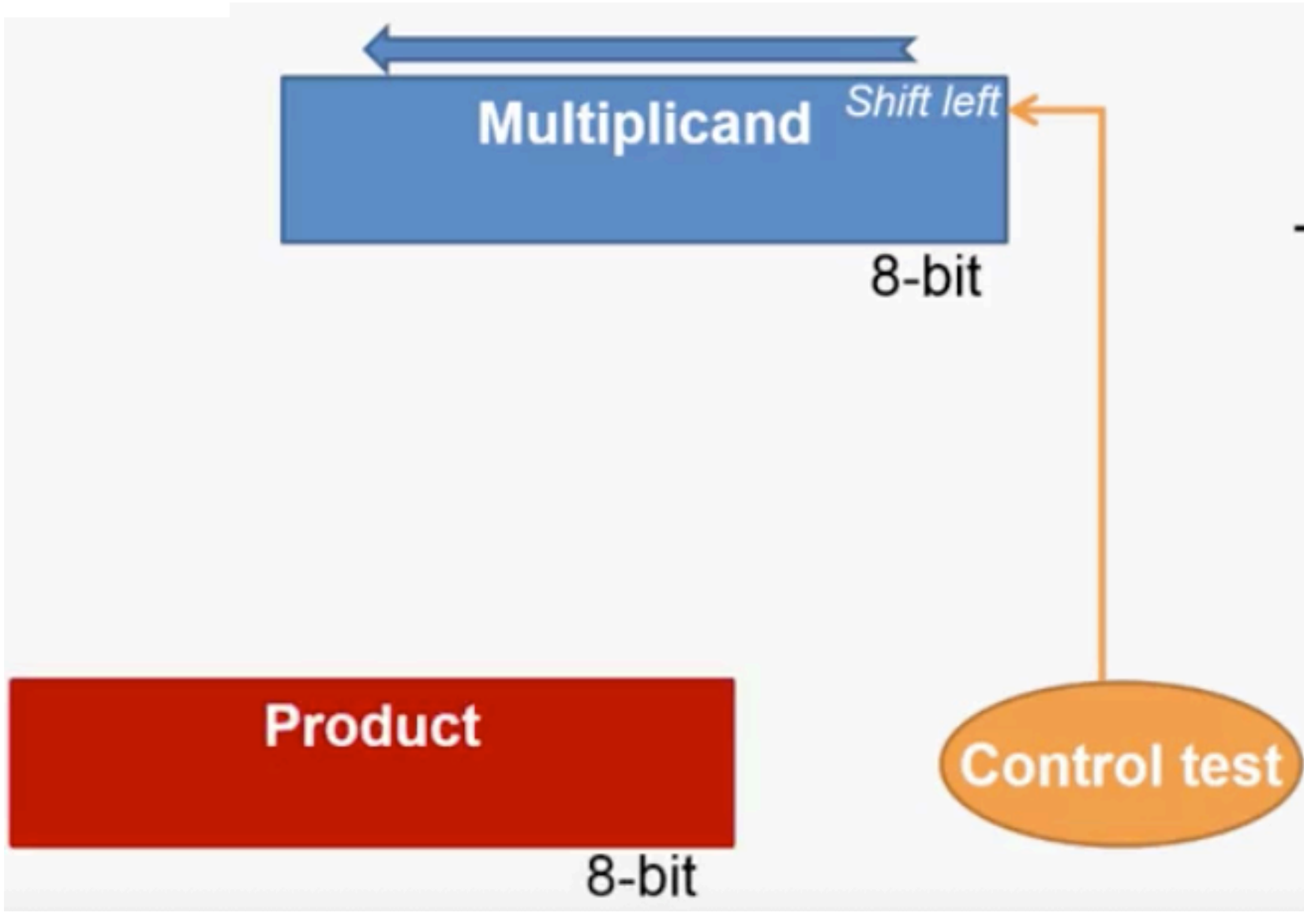
$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$



Circuit Implementation

- 8-bit register for multiplicand
  - With left shift function
- 8-bit register for product

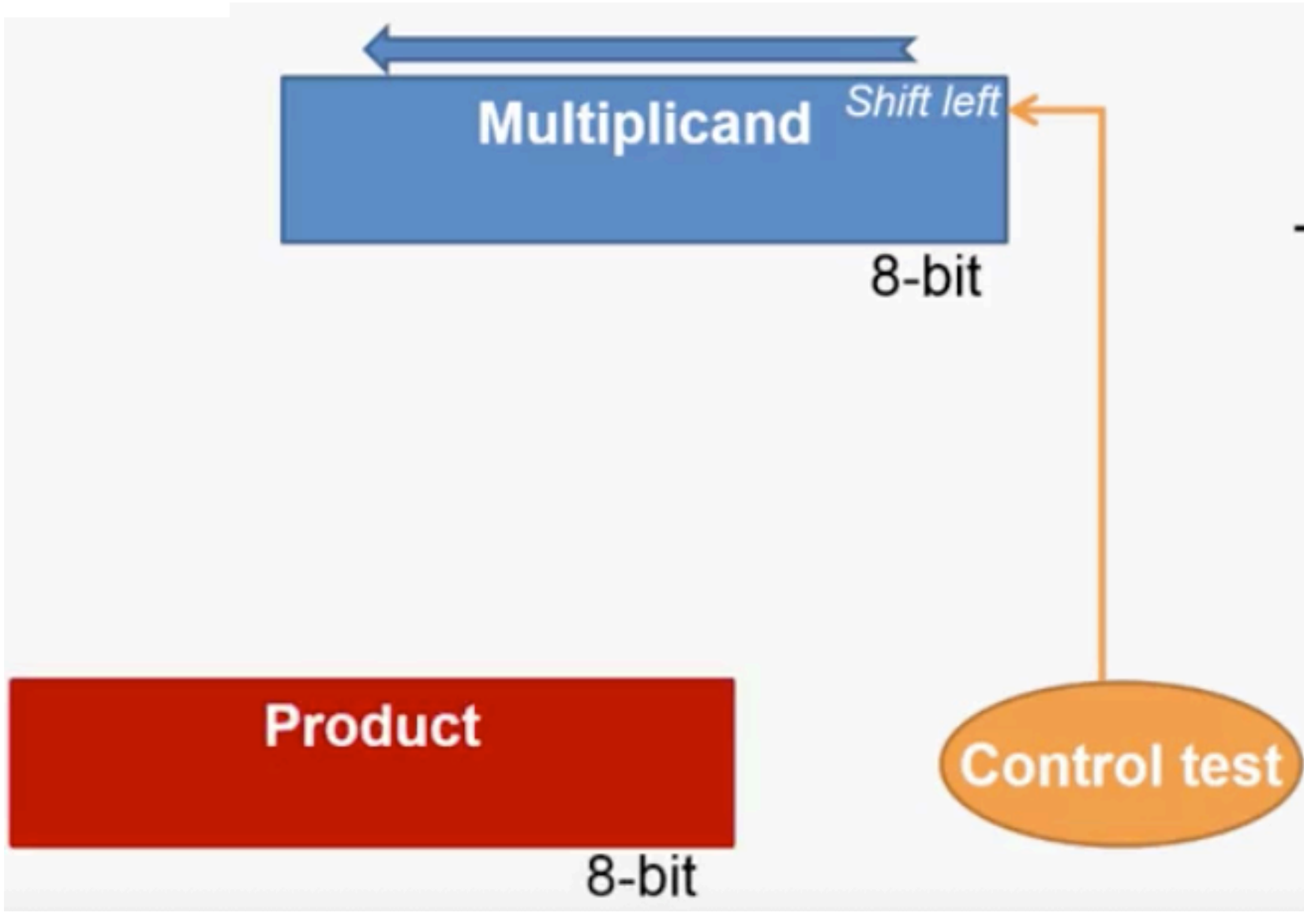
$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$



Circuit Implementation

- 8-bit register for multiplicand
  - With left shift function
- 8-bit register for product
- What else?

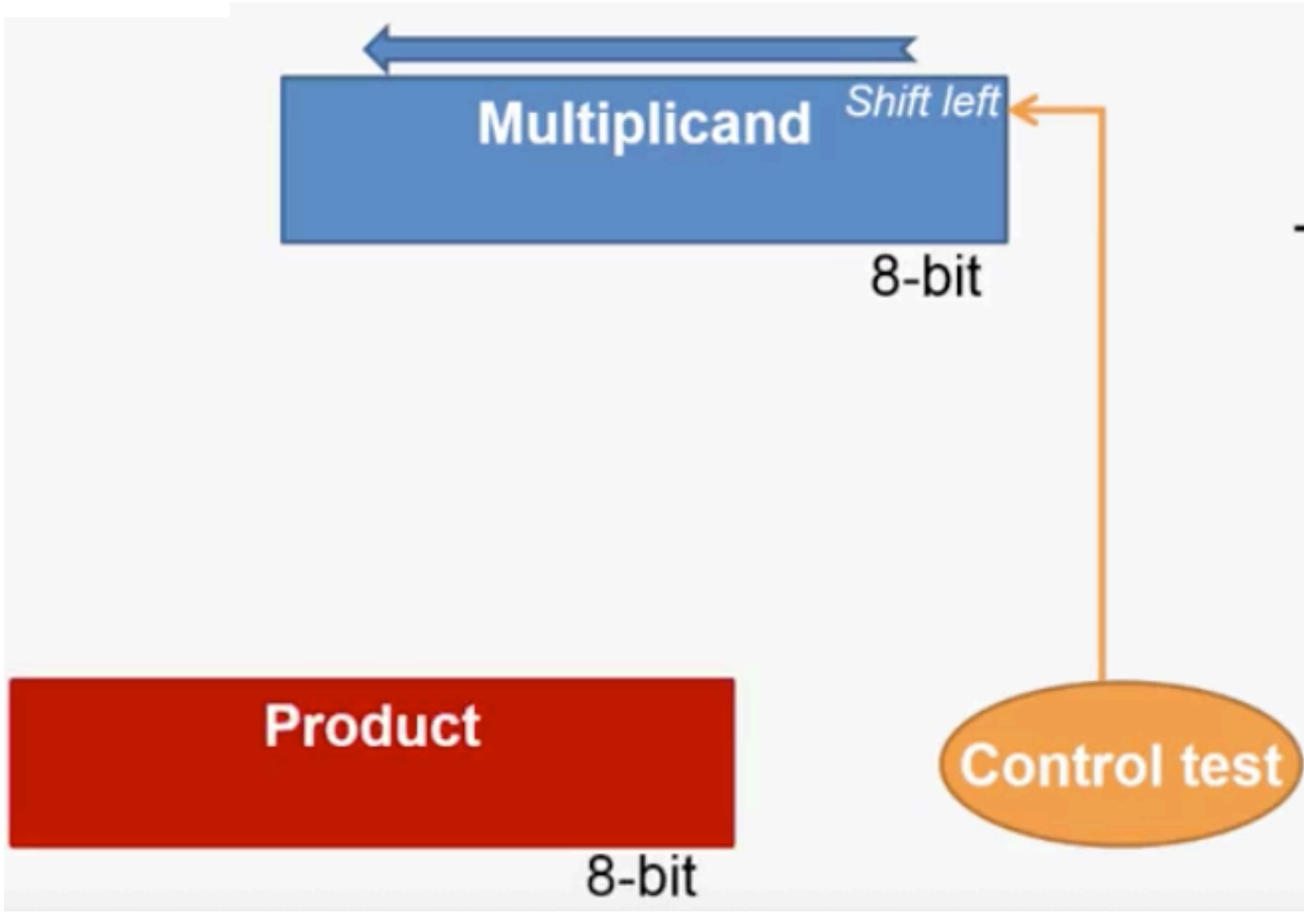
$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$



Circuit Implementation

- 8-bit register for multiplicand
  - With left shift function
- 8-bit register for product
- What else?
- 8-bit Adder

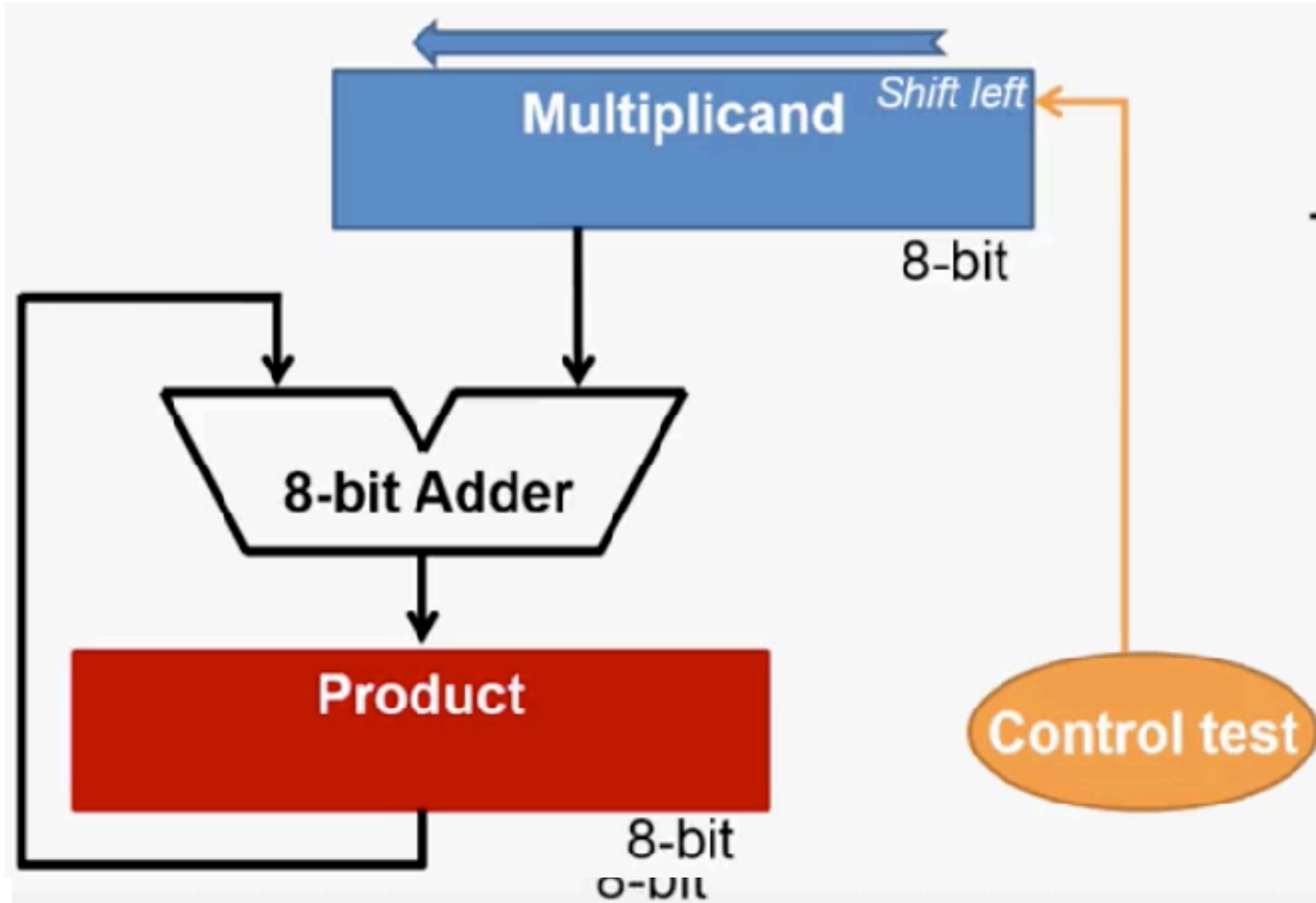
$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$



Circuit Implementation

- 8-bit register for multiplicand
  - With left shift function
- 8-bit register for product
- What else?
- 8-bit Adder

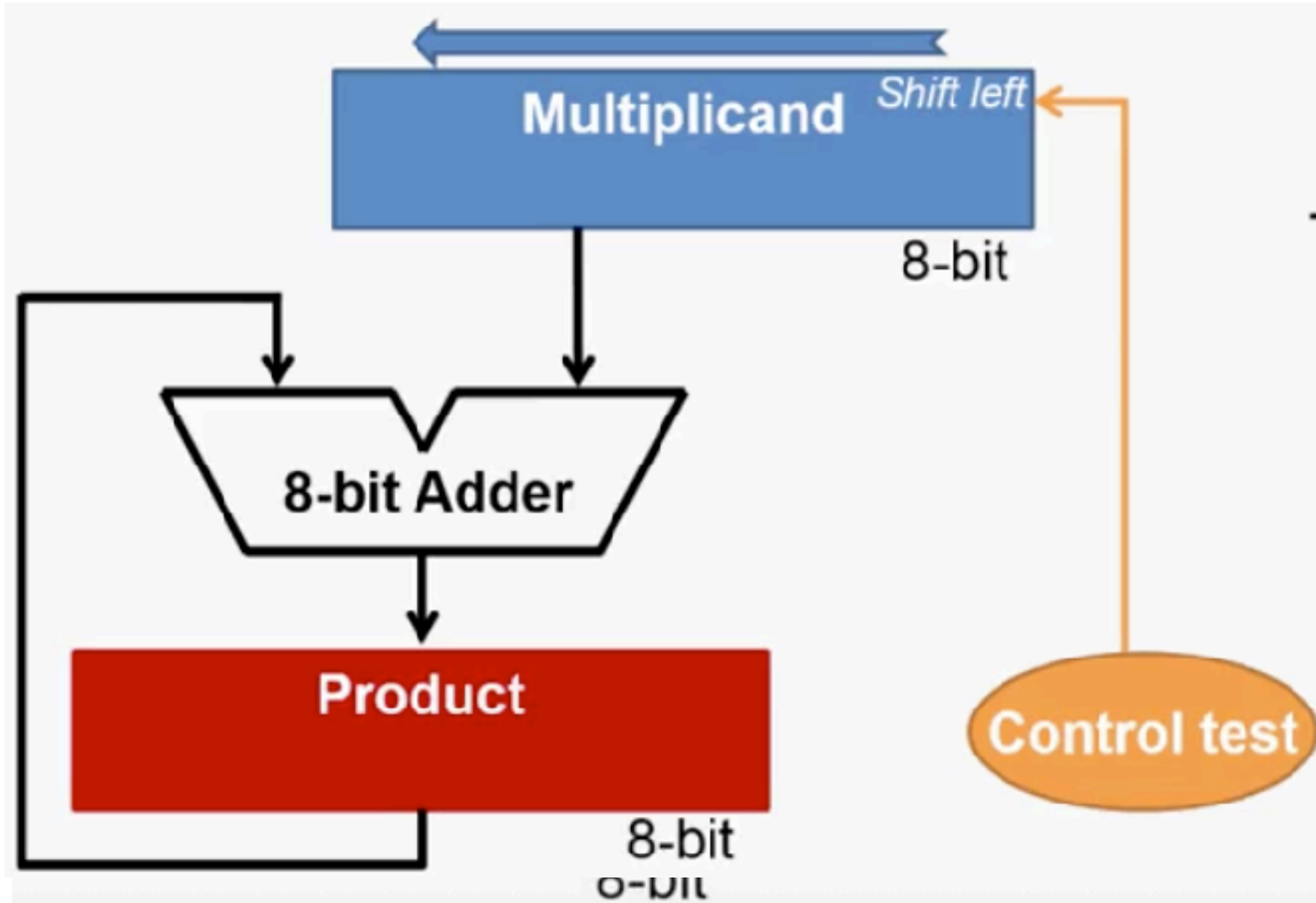
$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$



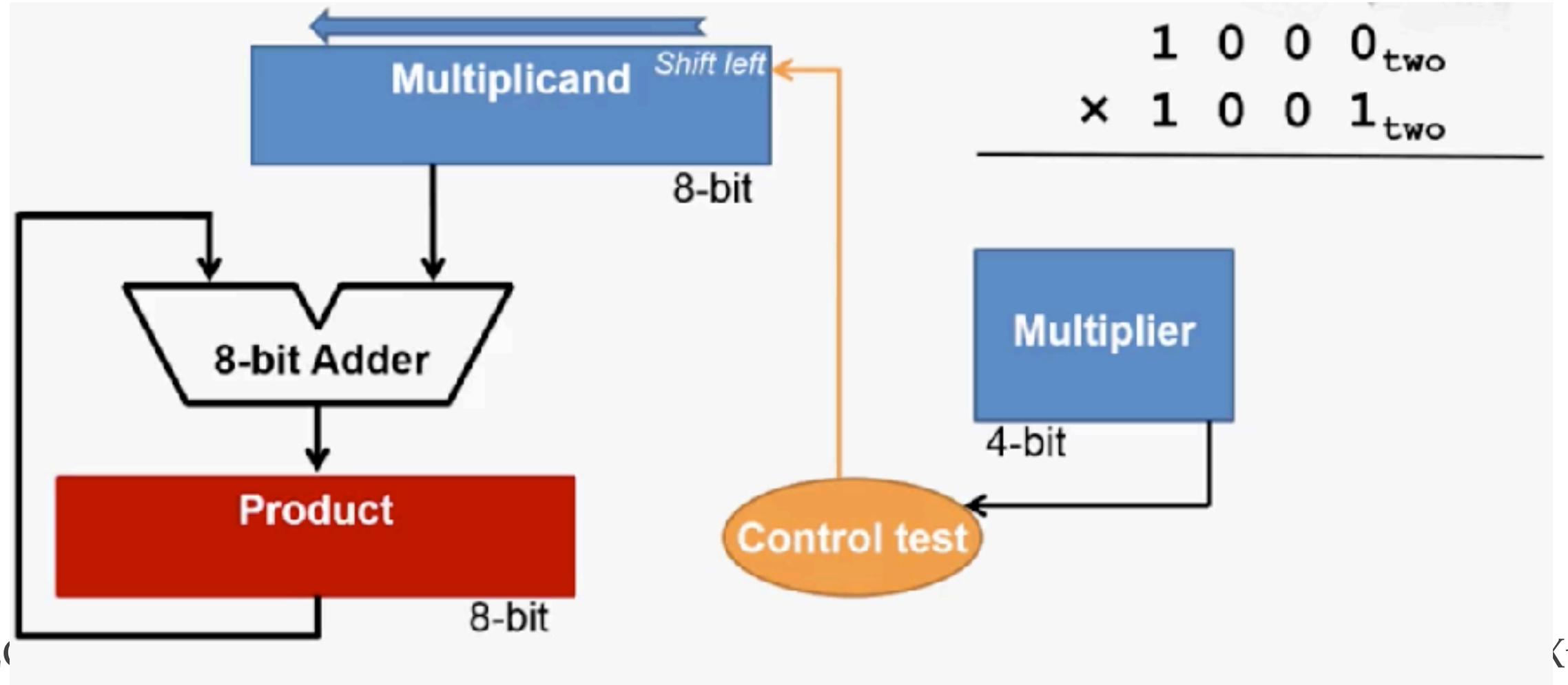
Circuit Implementation

- 8-bit register for multiplicand
  - With left shift function
- 8-bit register for product
- What else?
- 8-bit Adder
- What else?

$$\begin{array}{r} 1\ 0\ 0\ 0_{two} \\ \times\ 1\ 0\ 0\ 1_{two} \\ \hline \end{array}$$

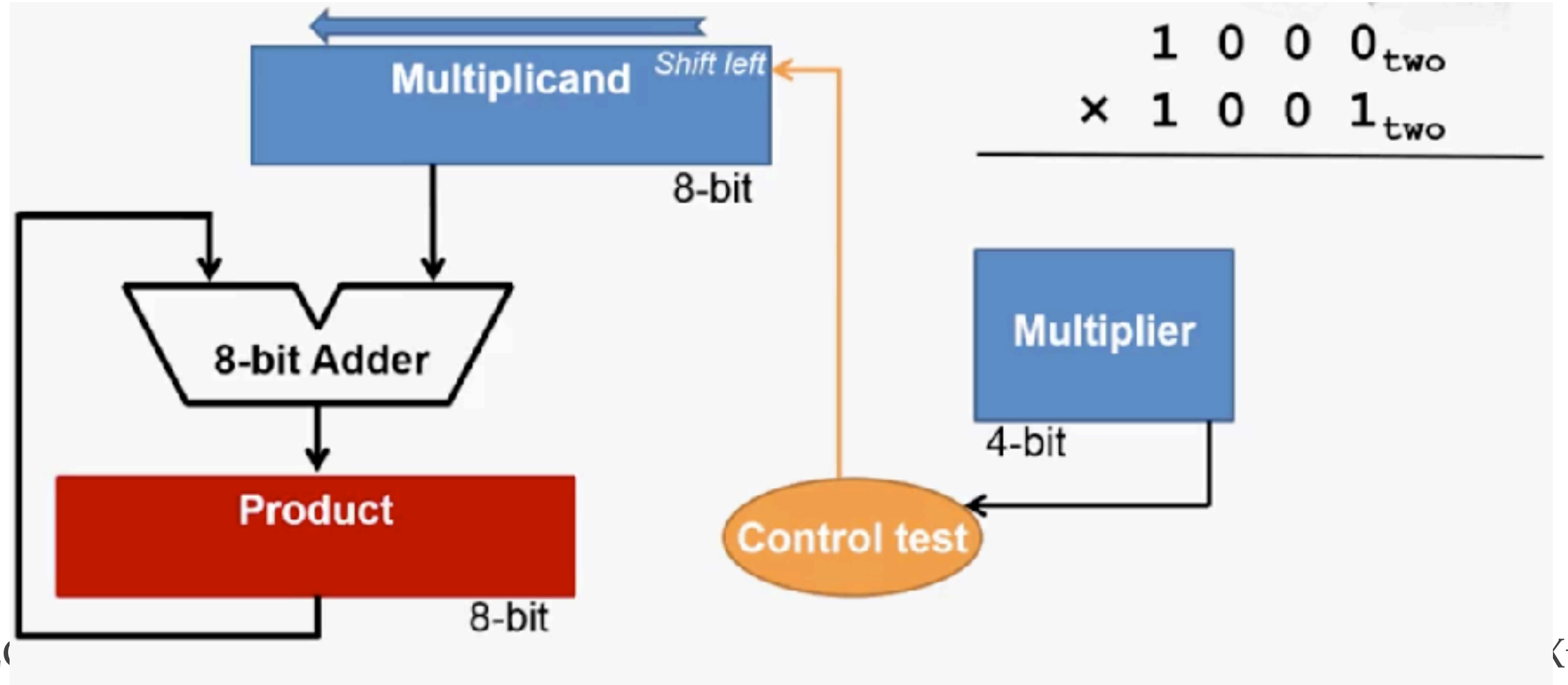


Circuit Implementation



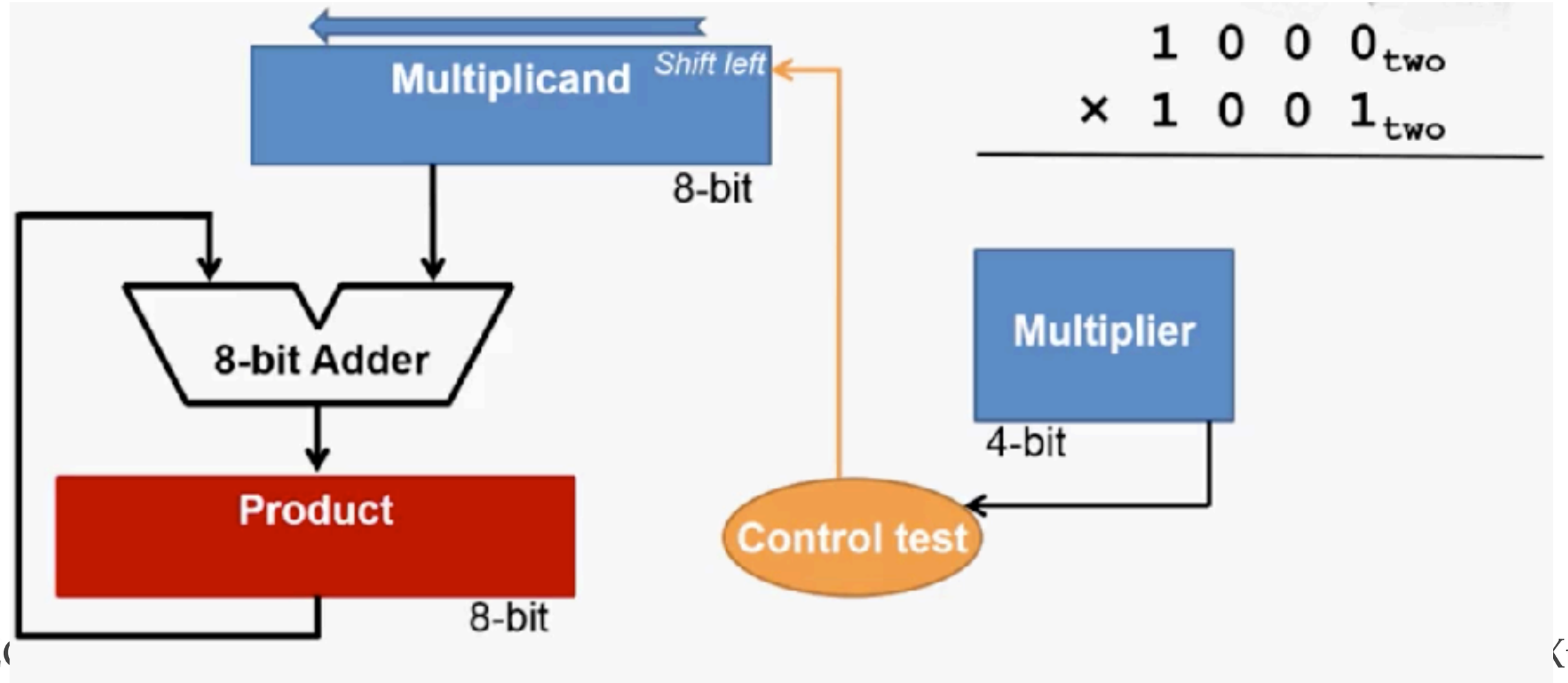
Circuit Implementation

- 4-bit register for multiplier



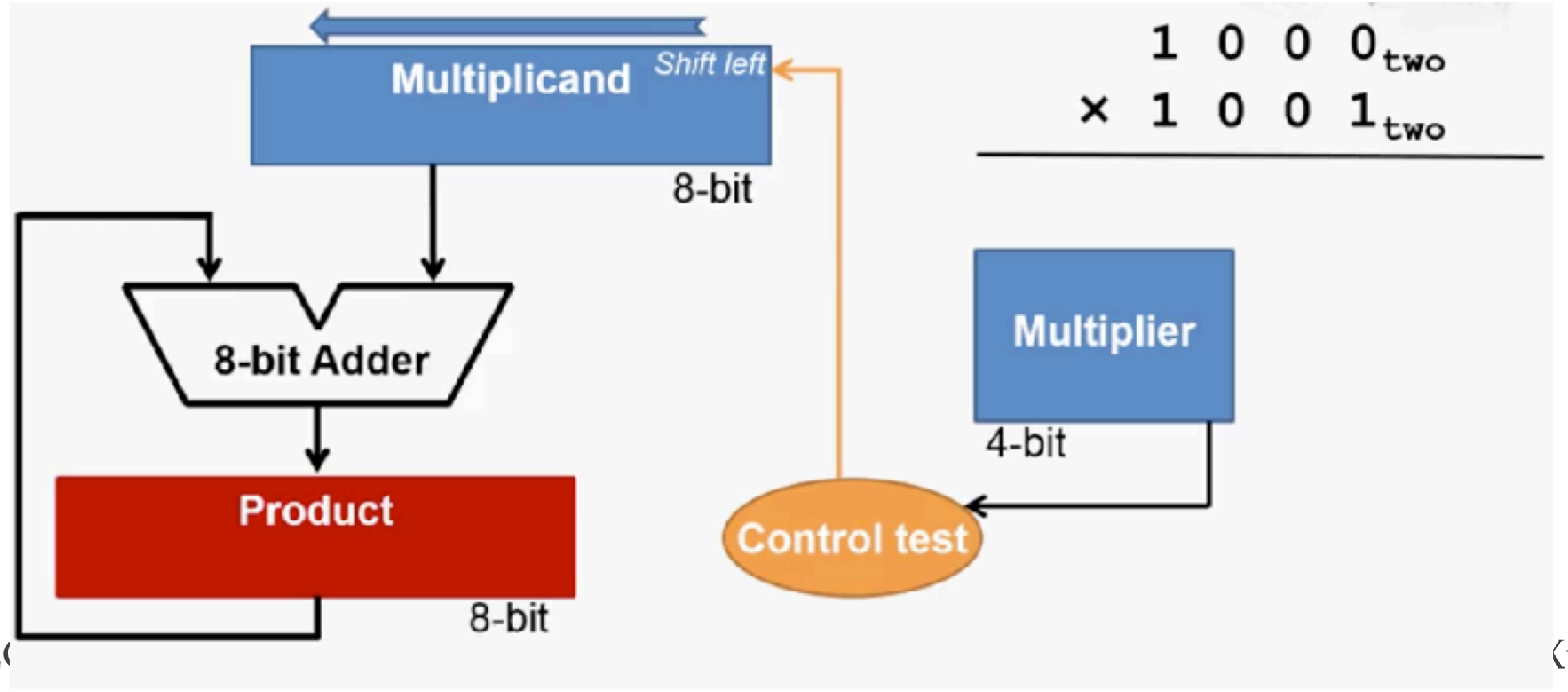
Circuit Implementation

- 4-bit register for multiplier
  - What is the data from multiplier to control test?

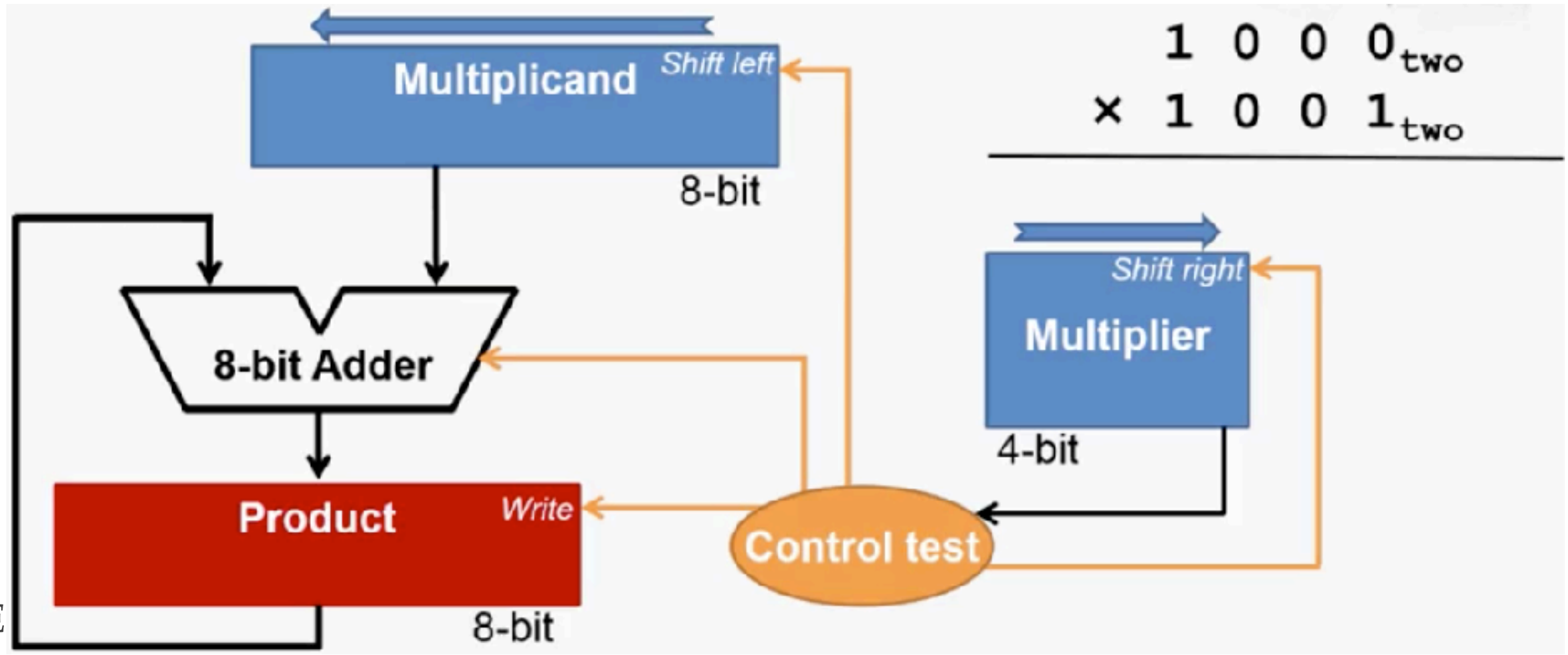


Circuit Implementation

- 4-bit register for multiplier
  - What is the data from multiplier to control test?
    - LSB == 0? Shift to left? Else?

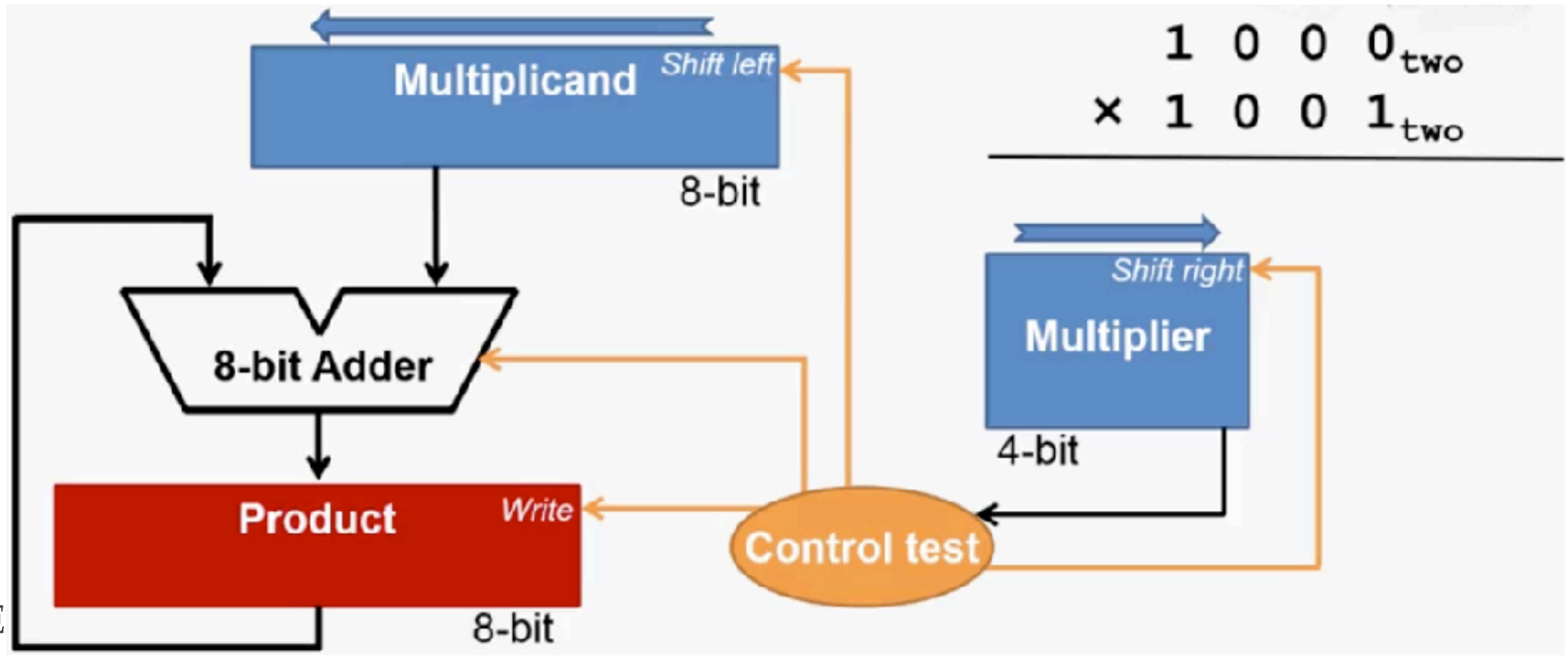


Circuit Implementation



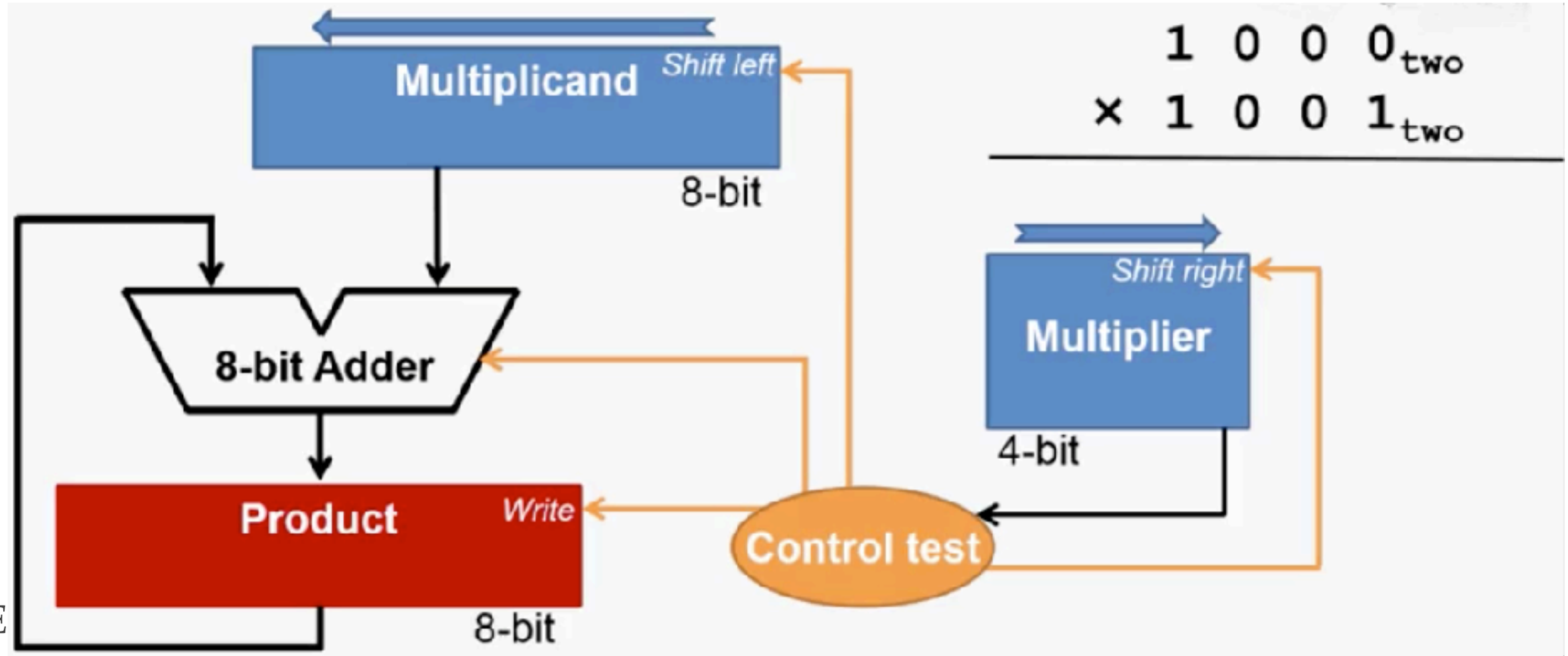
Circuit Implementation

- Adder executed?



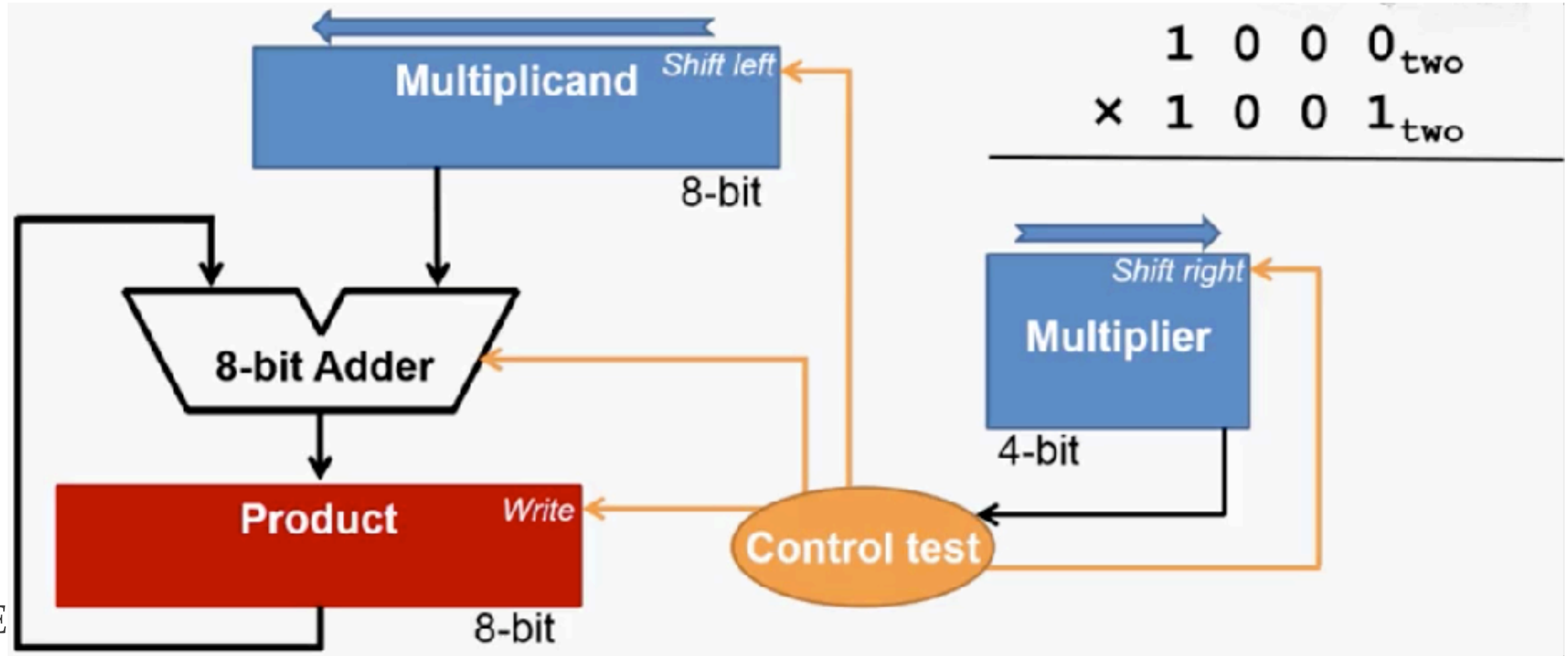
Circuit Implementation

- Adder executed?
- Writing operation executed?



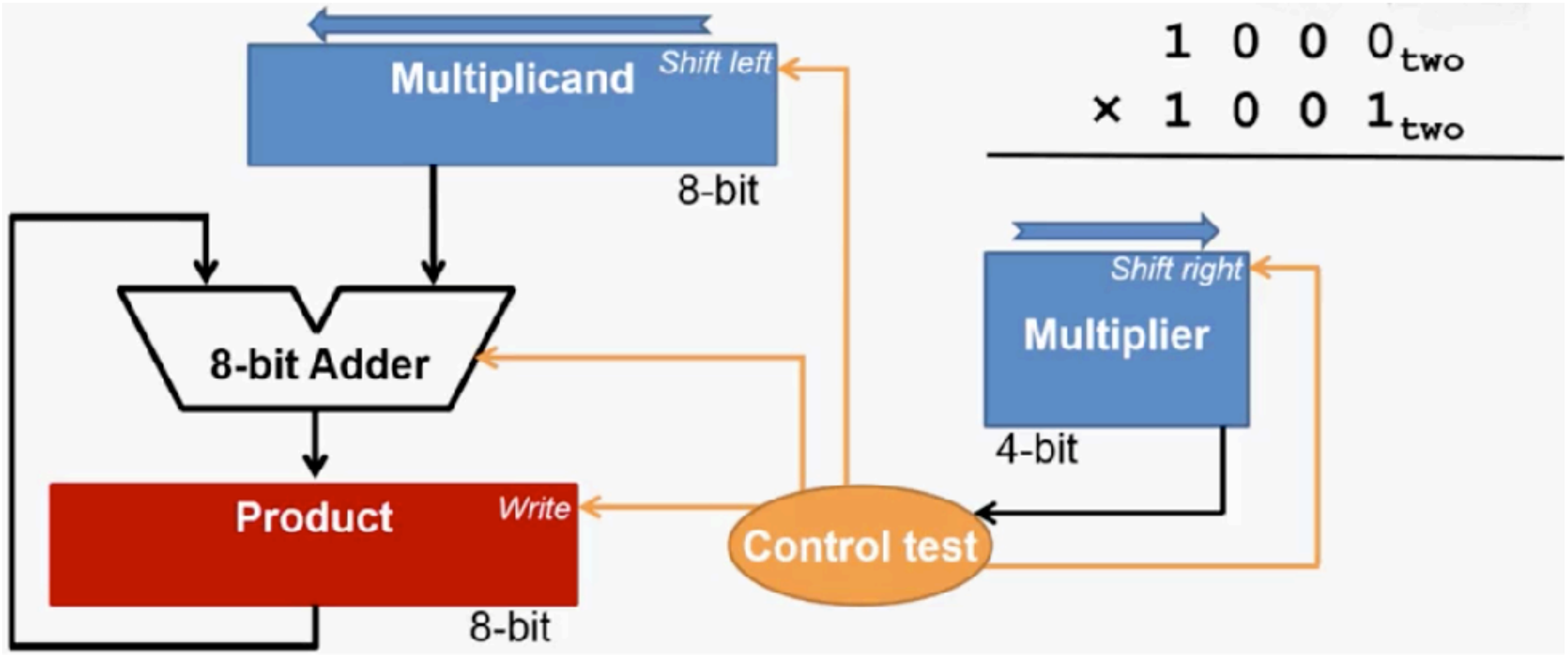
Circuit Implementation

- Adder executed?
- Writing operation executed?
- What else?

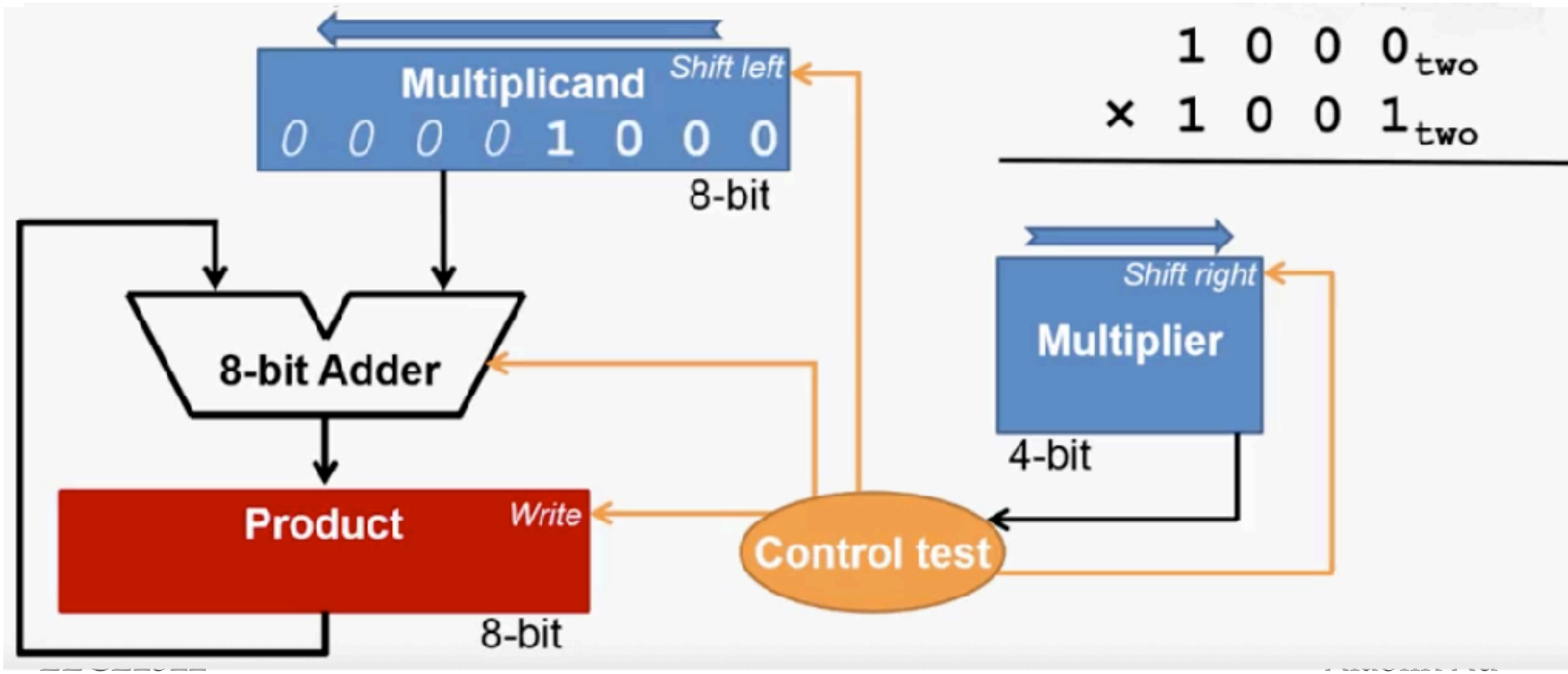


Initialization

Initialization

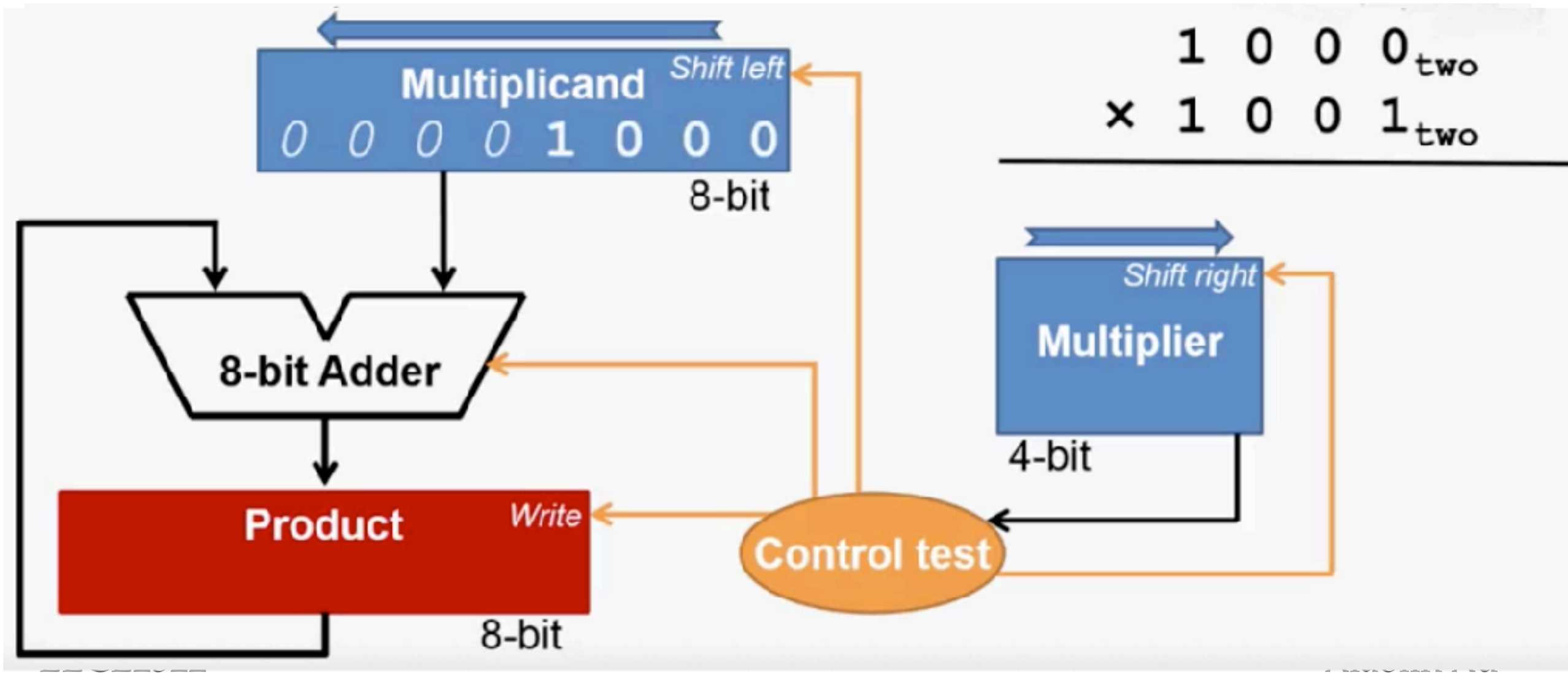


Initialization

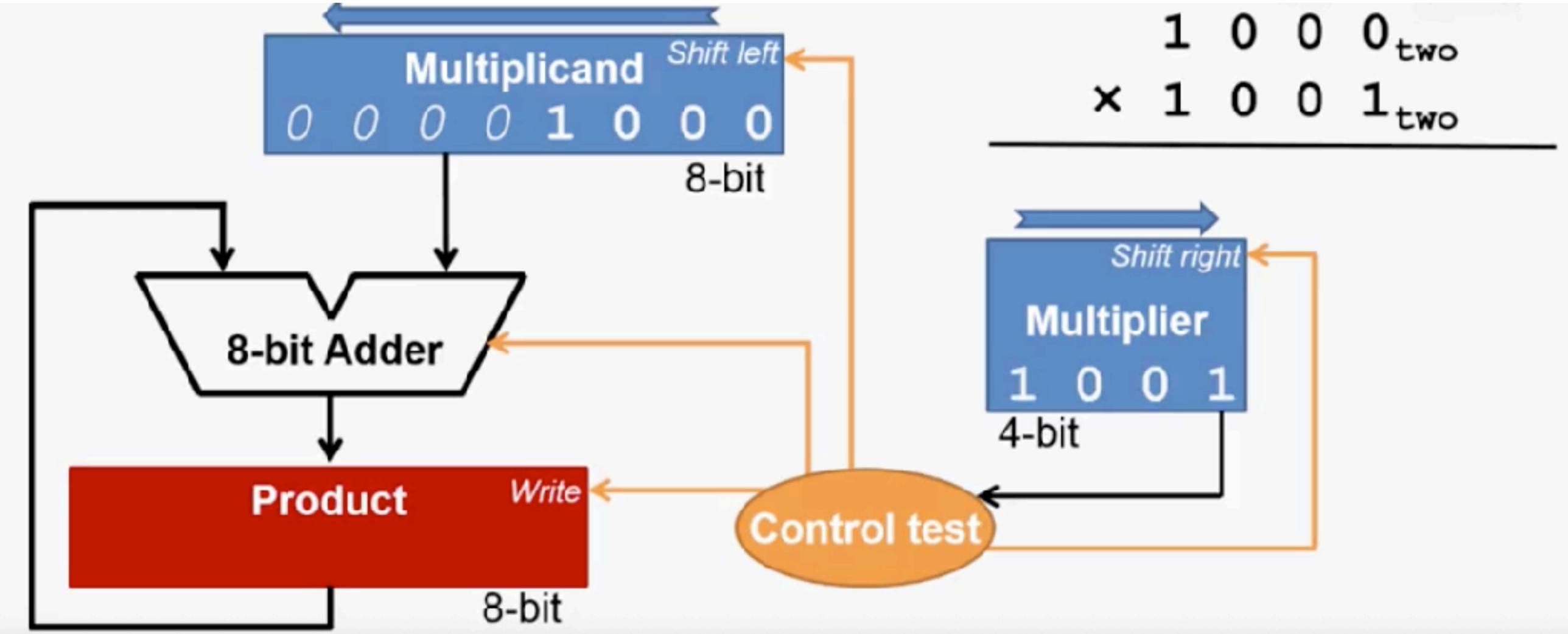


Initialization

- Load multiplicand (note the difference between *0000* and **1000**)

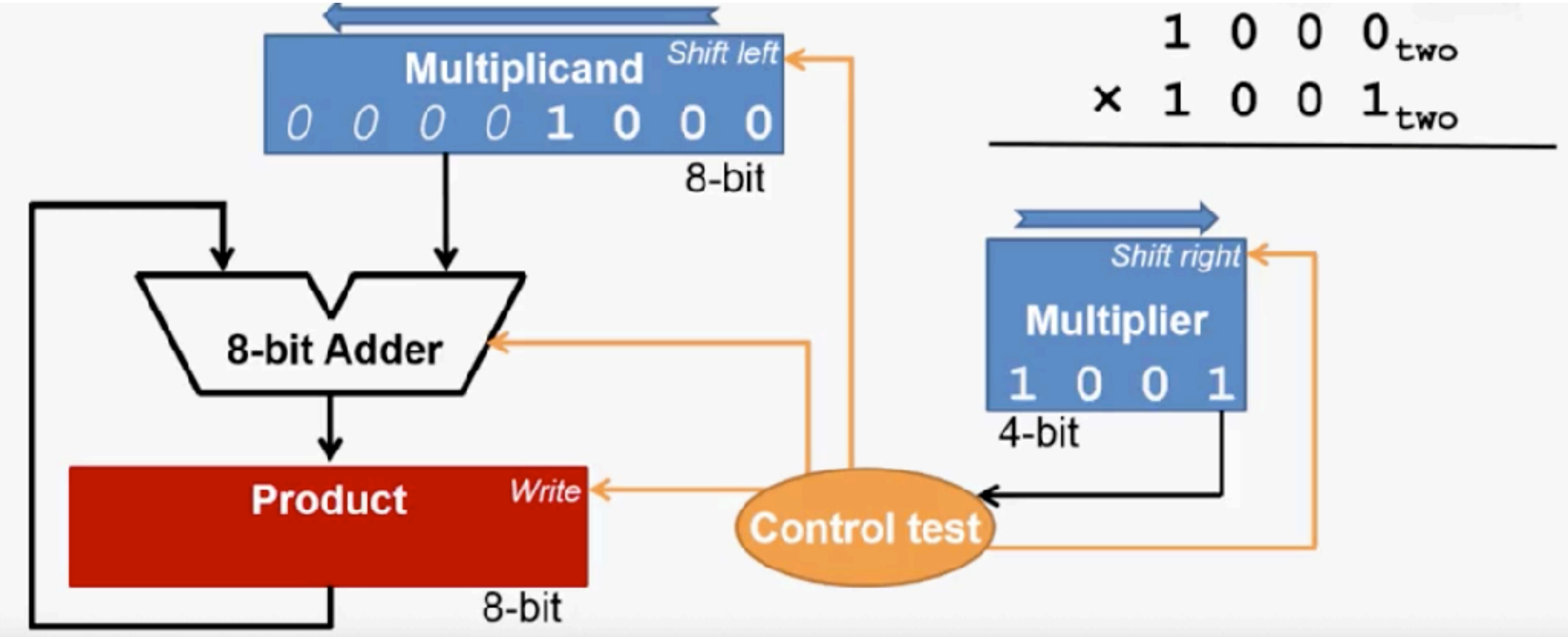


Initialization



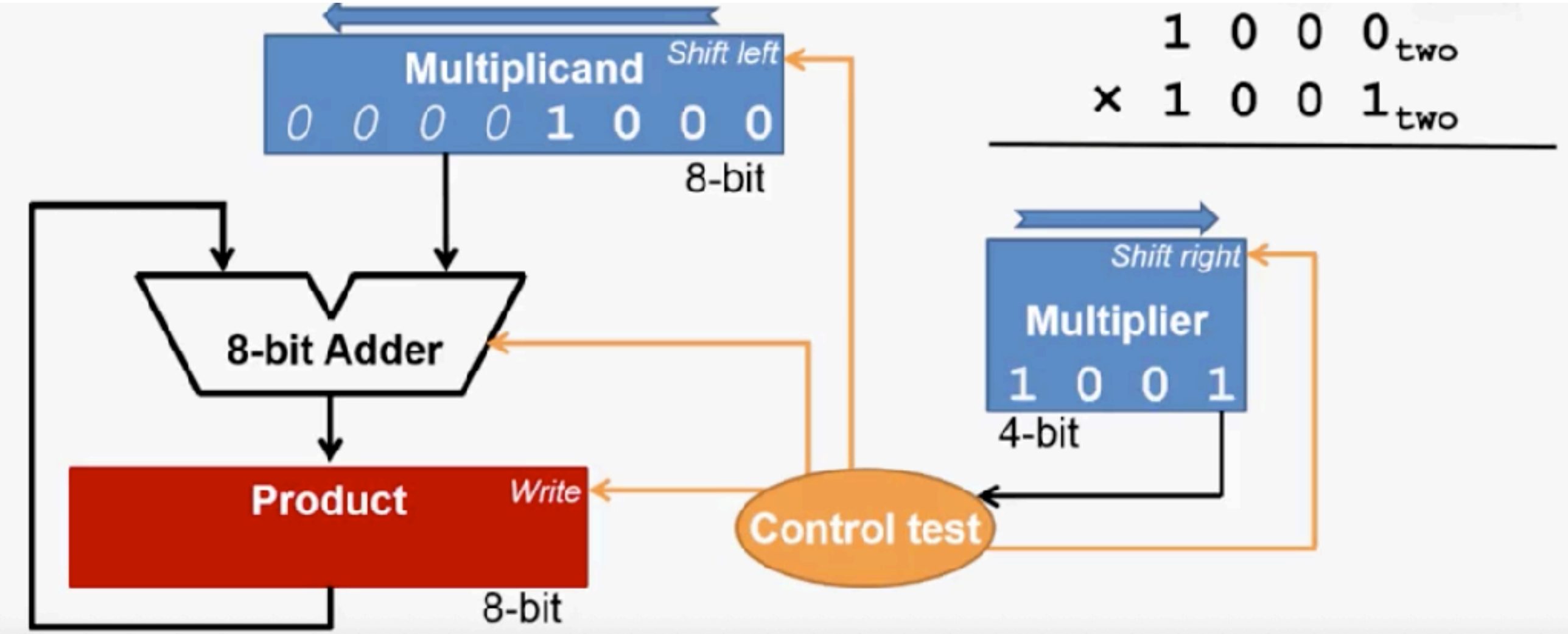
Initialization

- Load multiplier



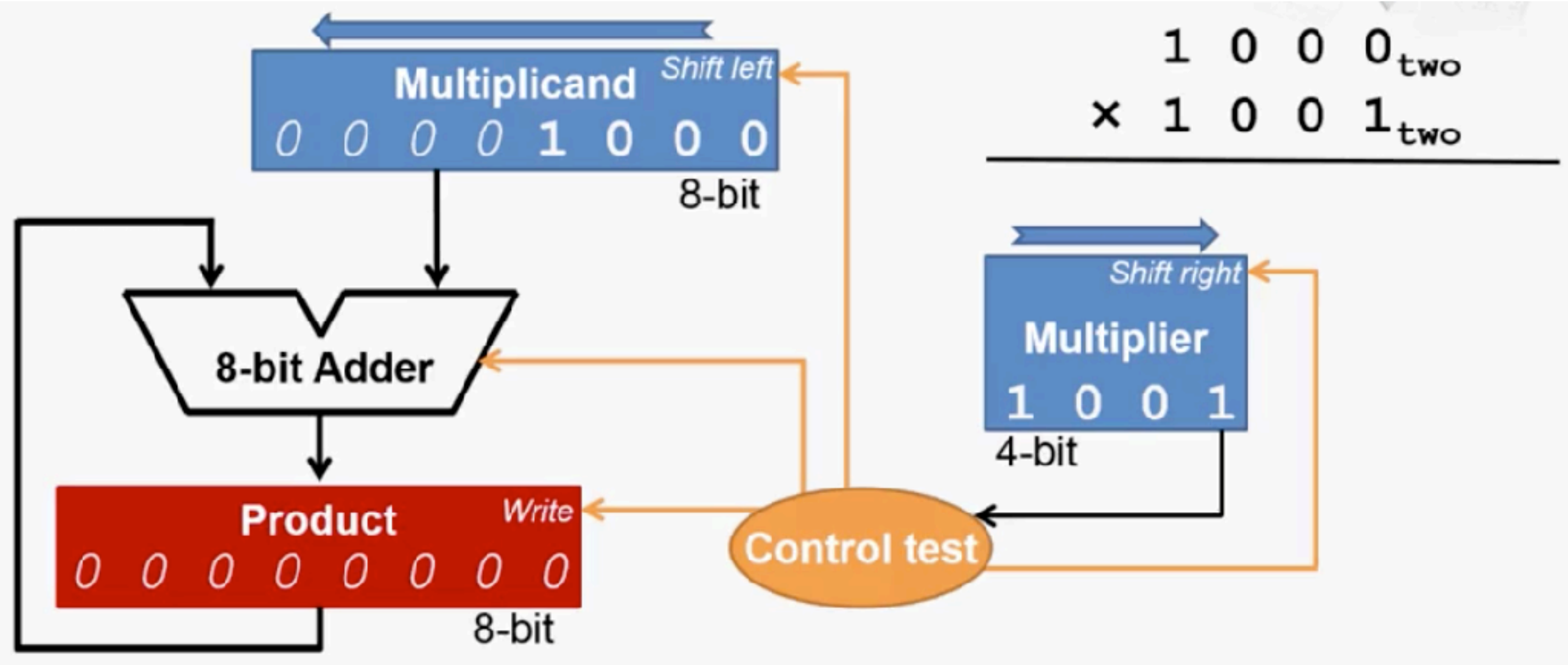
Initialization

- Load multiplier
- Initialize product (note the value is *00000000*)



Initialization

- Load multiplier
- Initialize product (note the value is *00000000*)

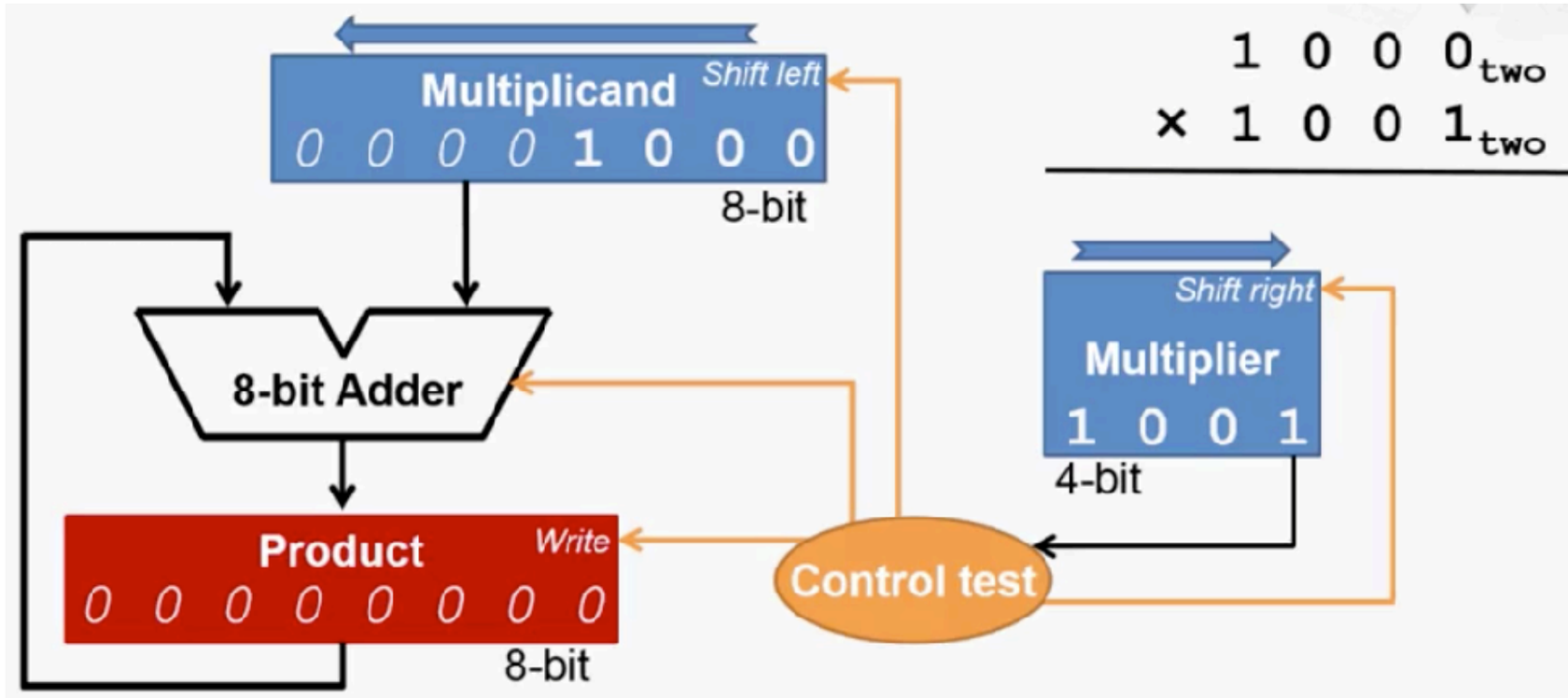


Step 1

- Recall the operation with paper and pencil

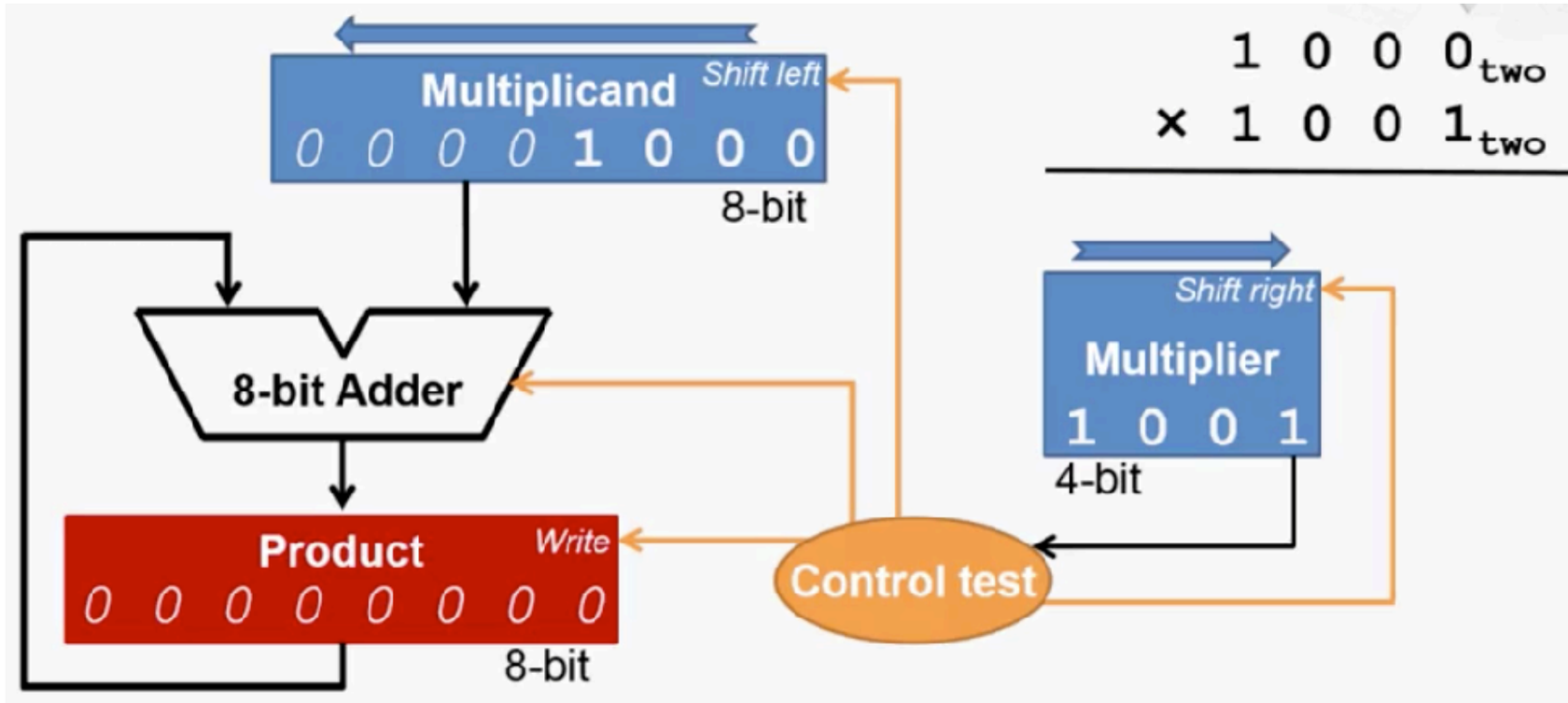
Step 1

- Recall the operation with paper and pencil



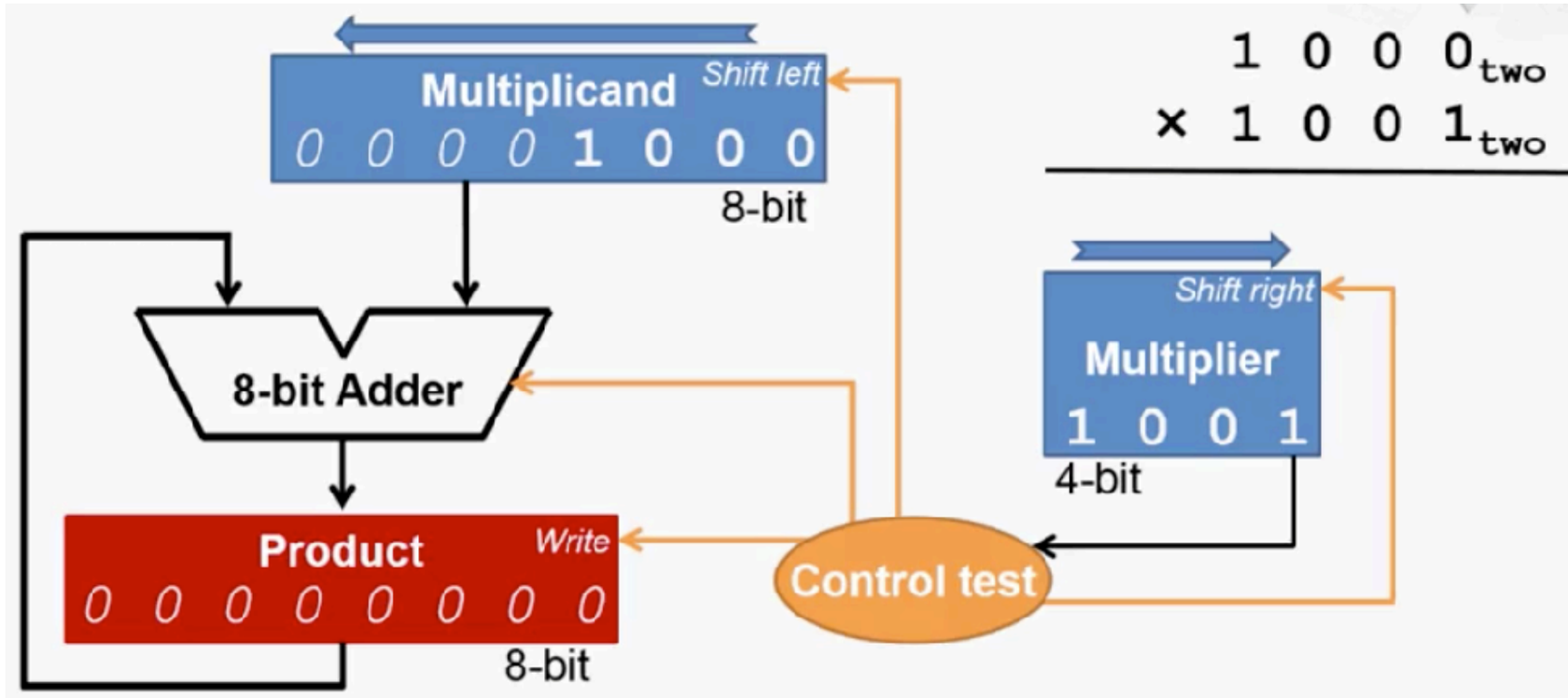
Step 1

- Recall the operation with paper and pencil
- What should we do?



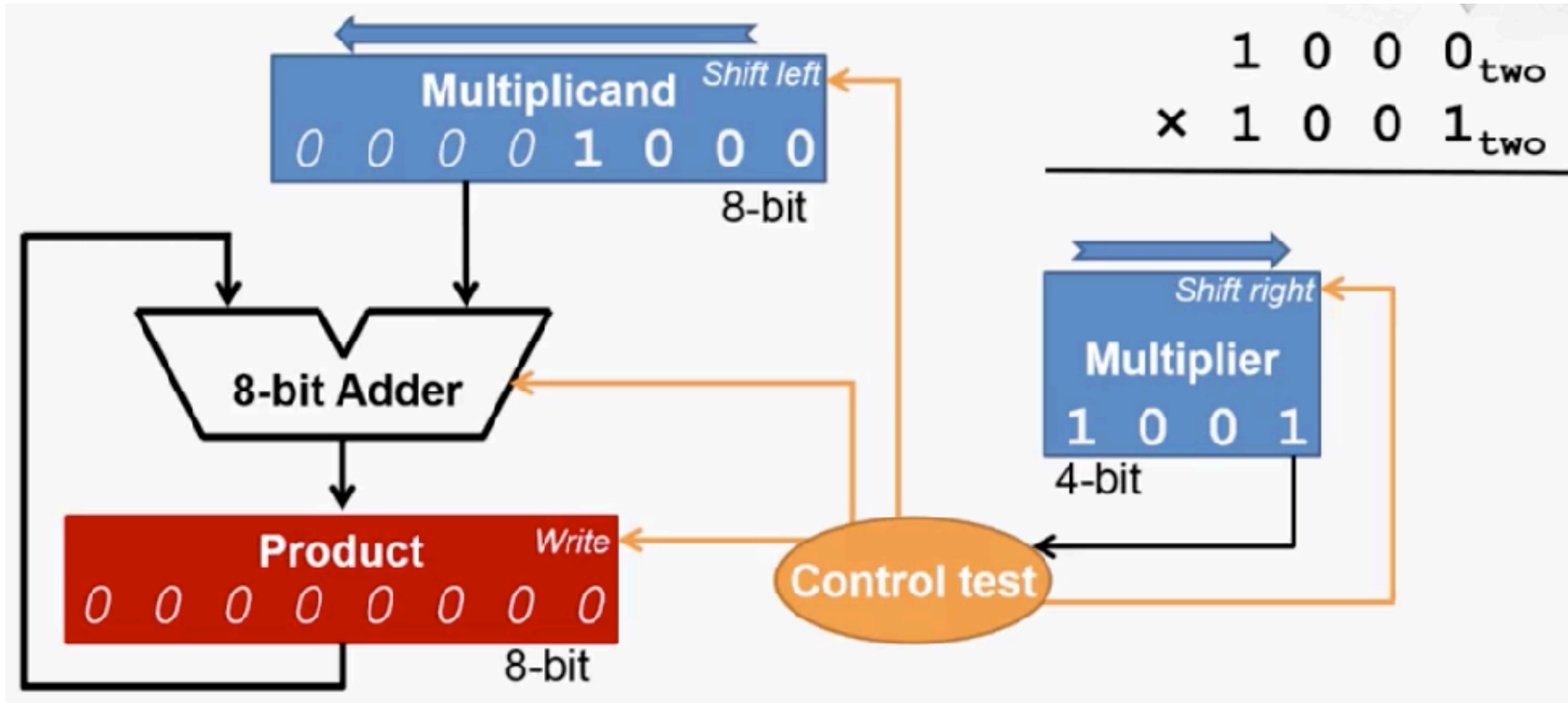
Step 1

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier



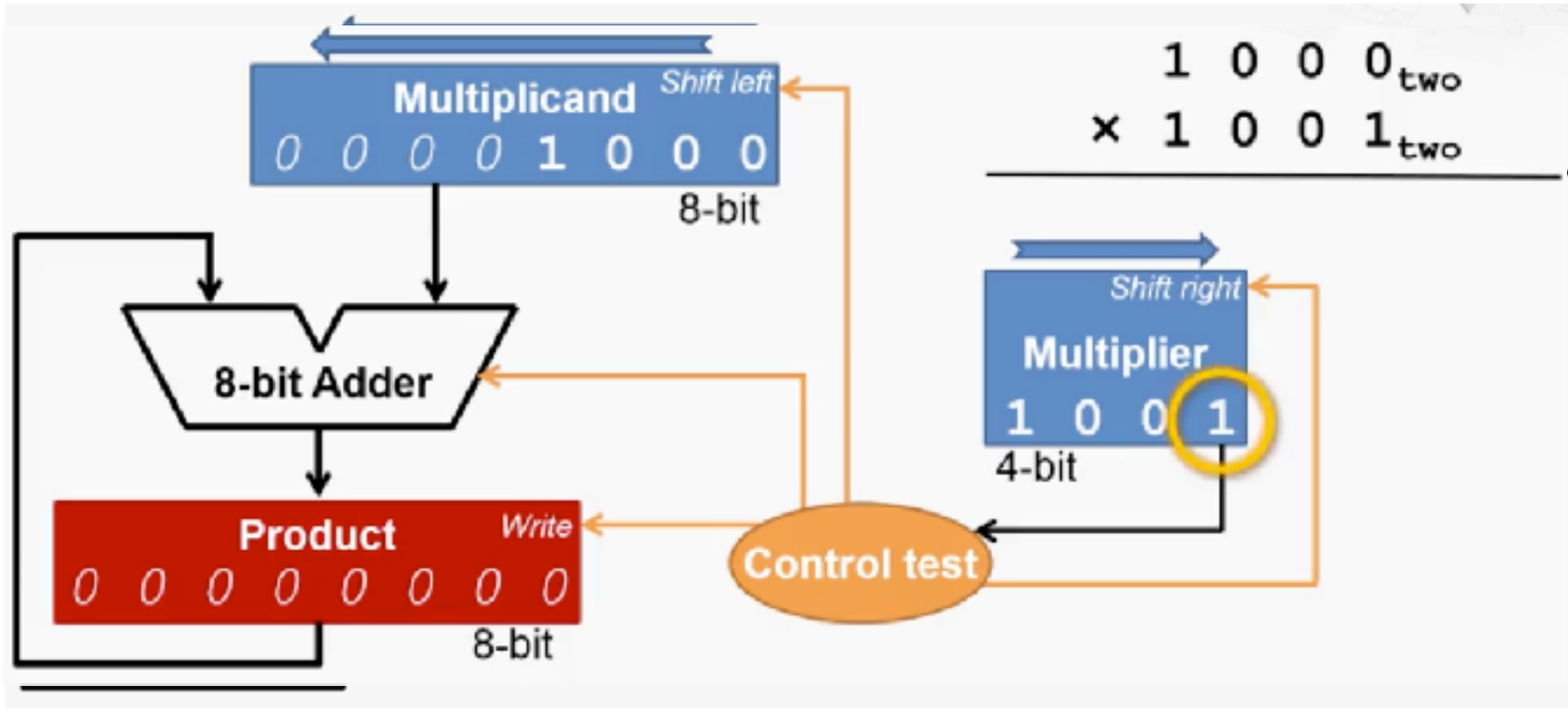
Step 1

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier
    - If 1



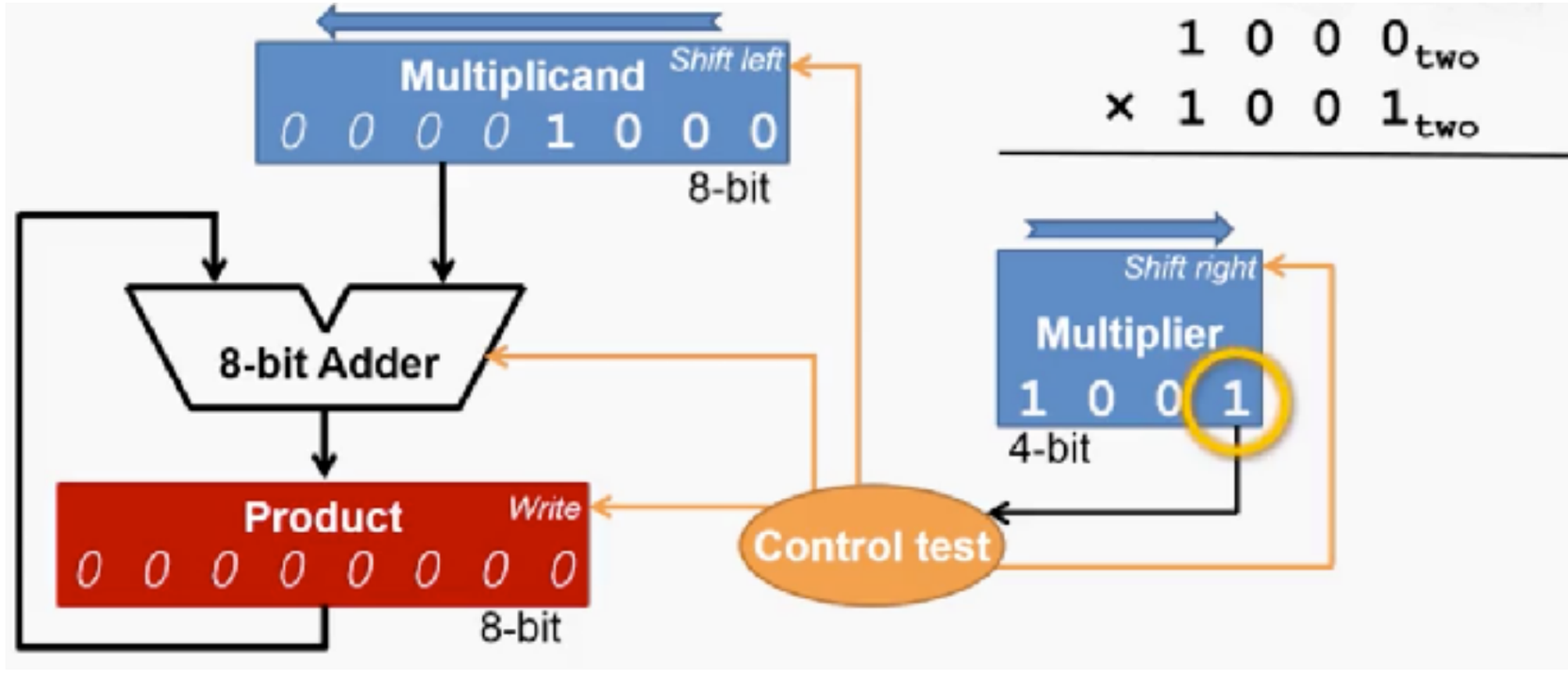
Step 1

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier
    - If 1



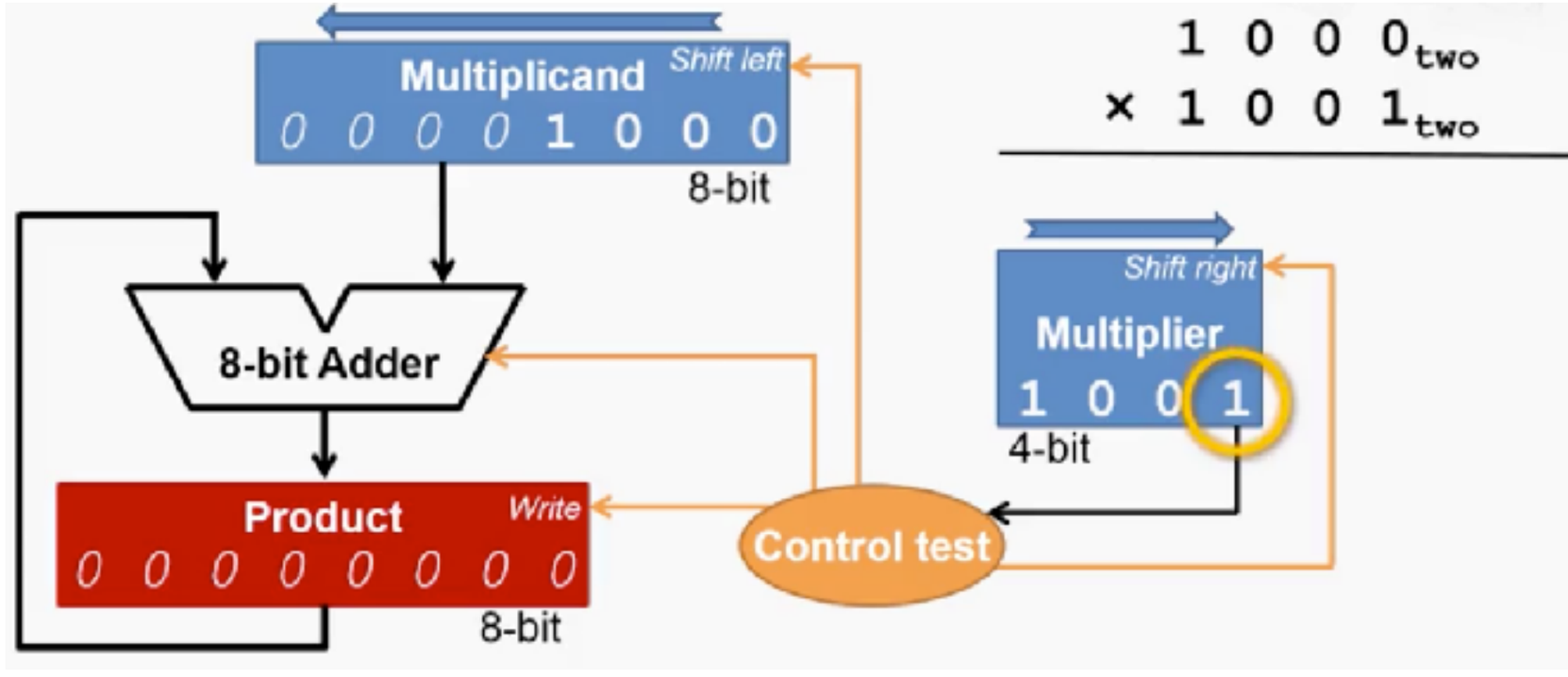
Step 1(a)

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier



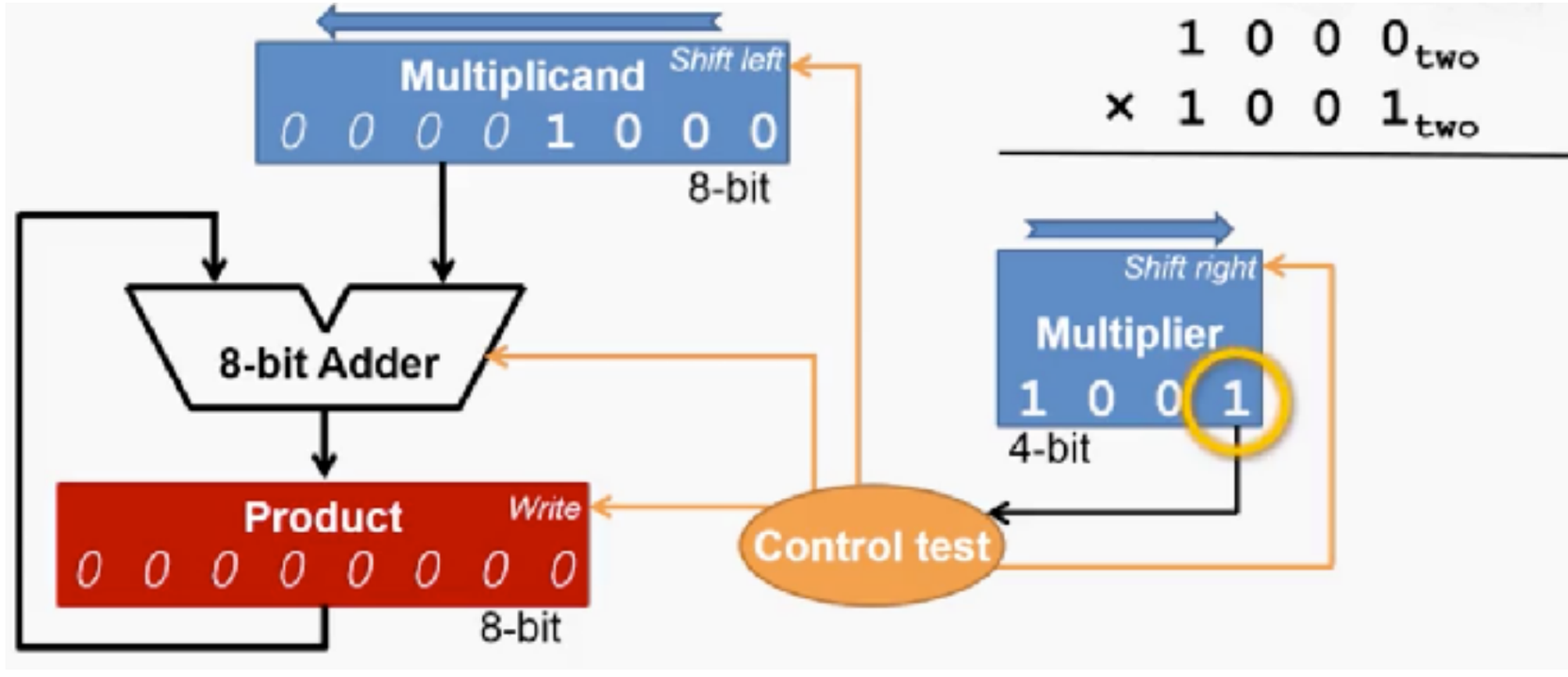
Step 1(a)

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier
    - If 1



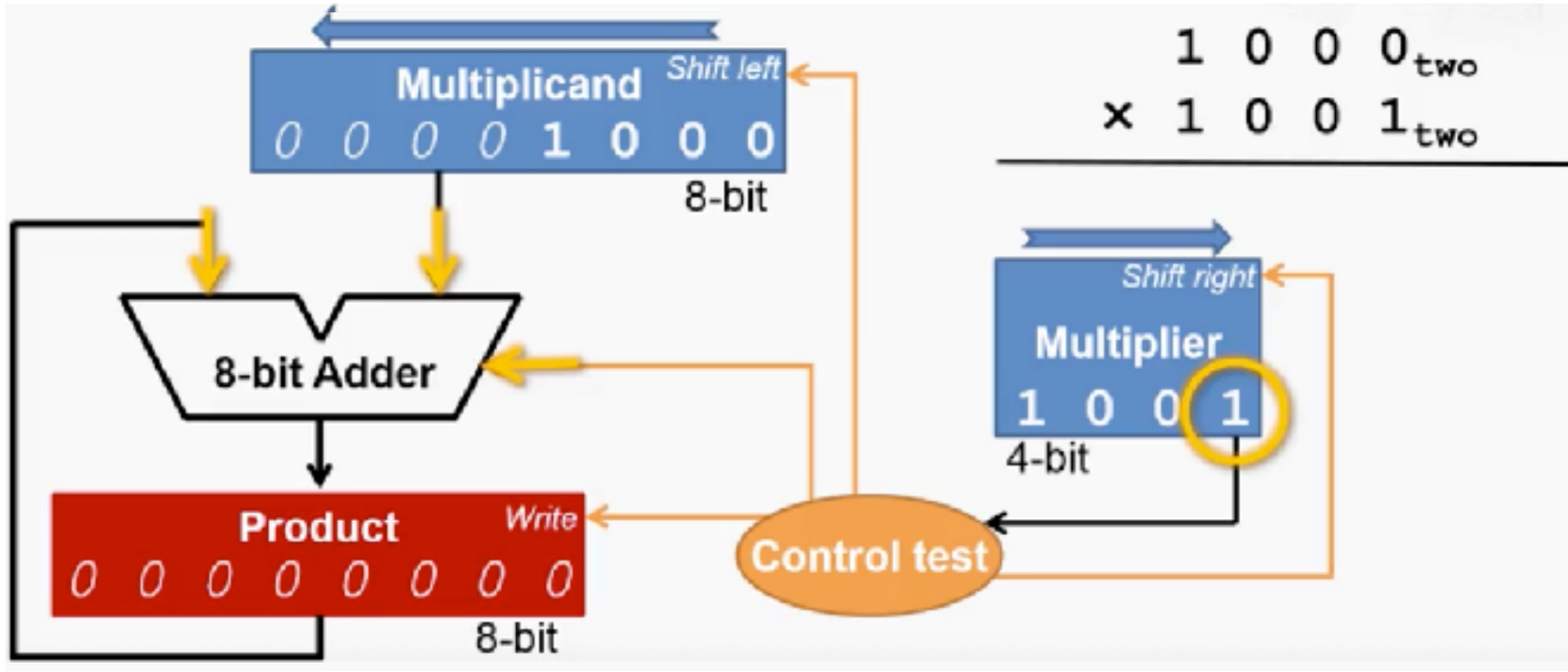
Step 1(a)

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier
    - If 1
    - Adding multiplicand and product



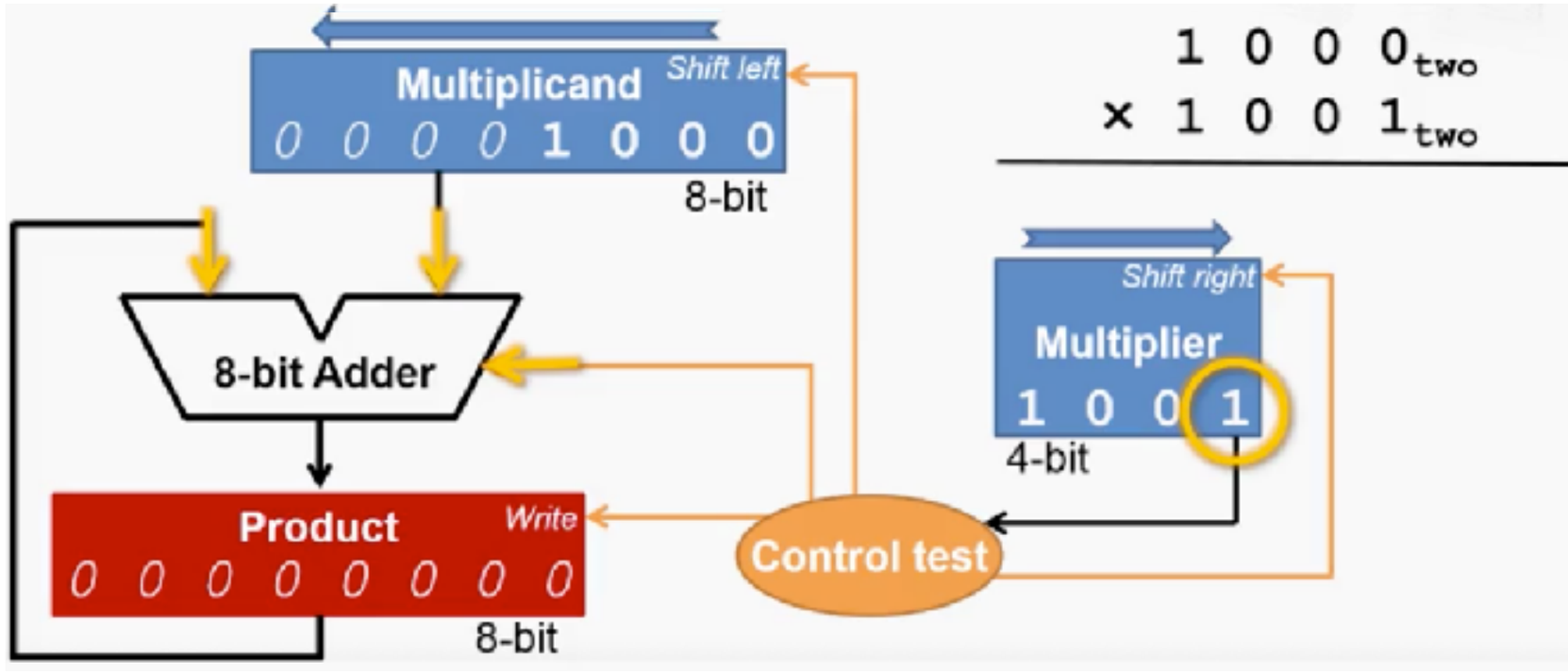
Step 1(a)

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier
    - If 1
    - Adding multiplicand and product



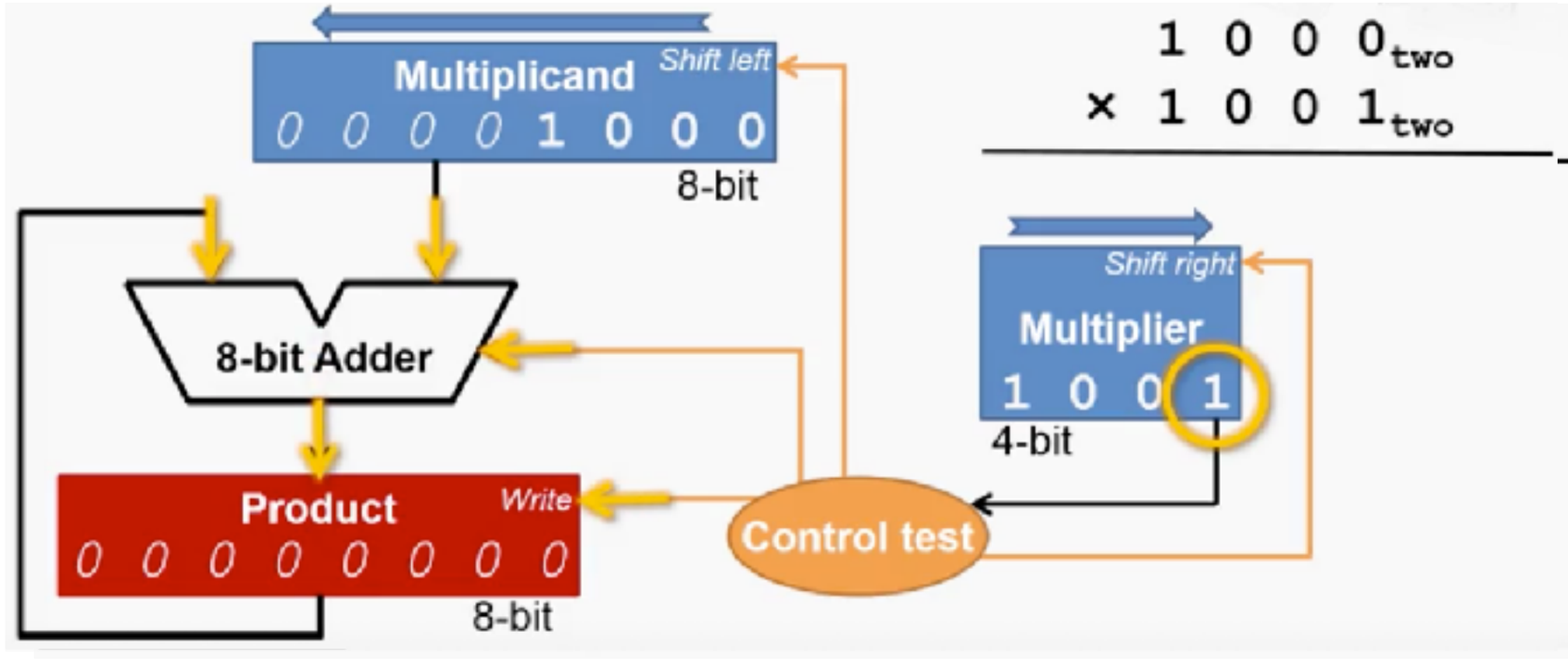
Step 1(a)

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier
    - If 1
    - Adding multiplicand and product
    - Store the value into product register



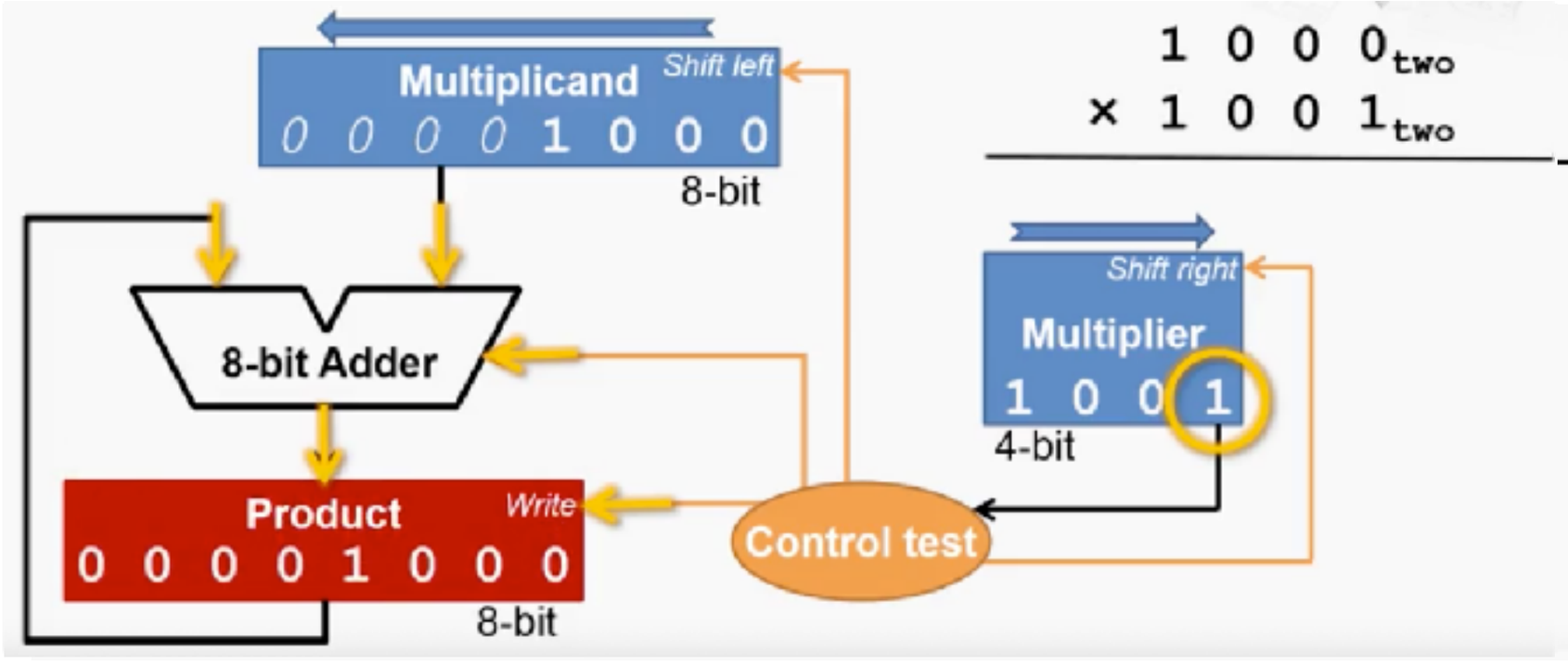
Step 1(a)

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier
    - If 1
    - Adding multiplicand and product
    - Store the value into product register

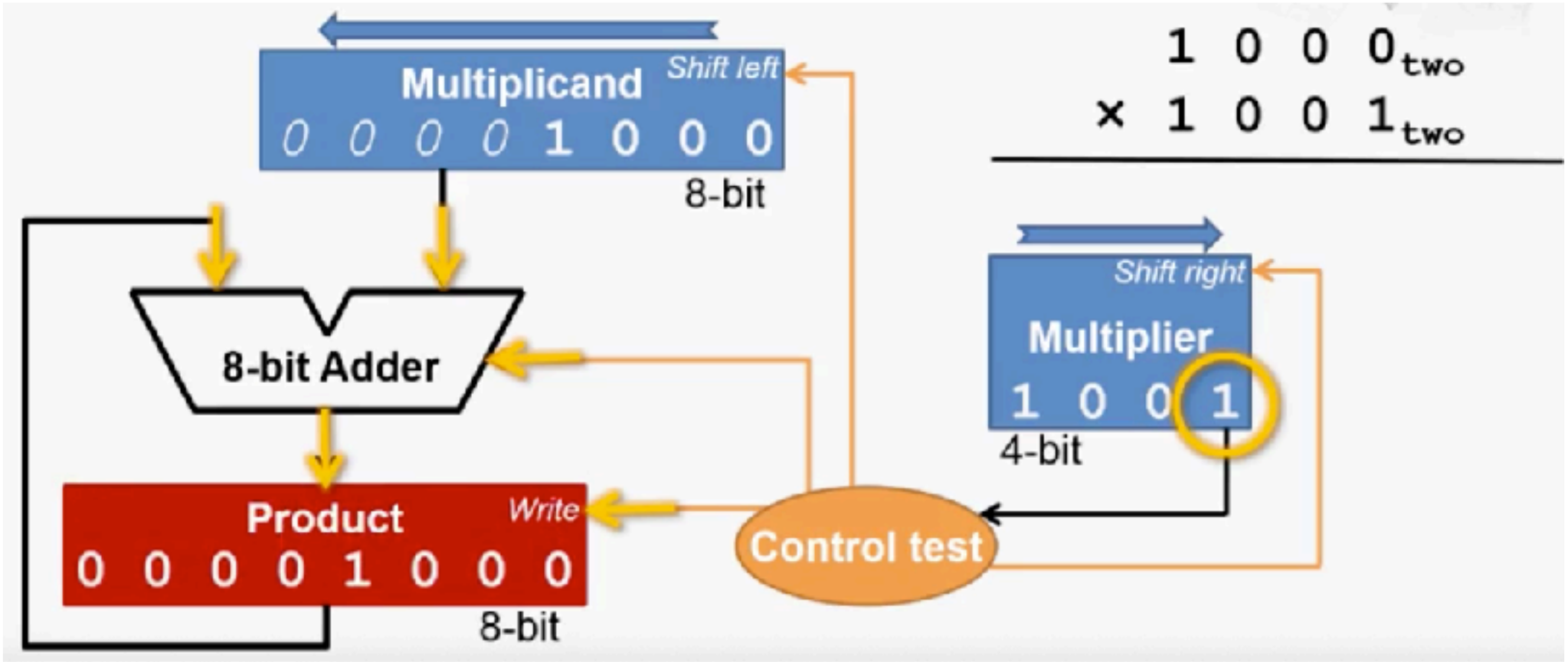


Step 1(a)

- Recall the operation with paper and pencil
- What should we do?
  - Check the LSB of multiplier
    - If 1
    - Adding multiplicand and product
    - Store the value into product register

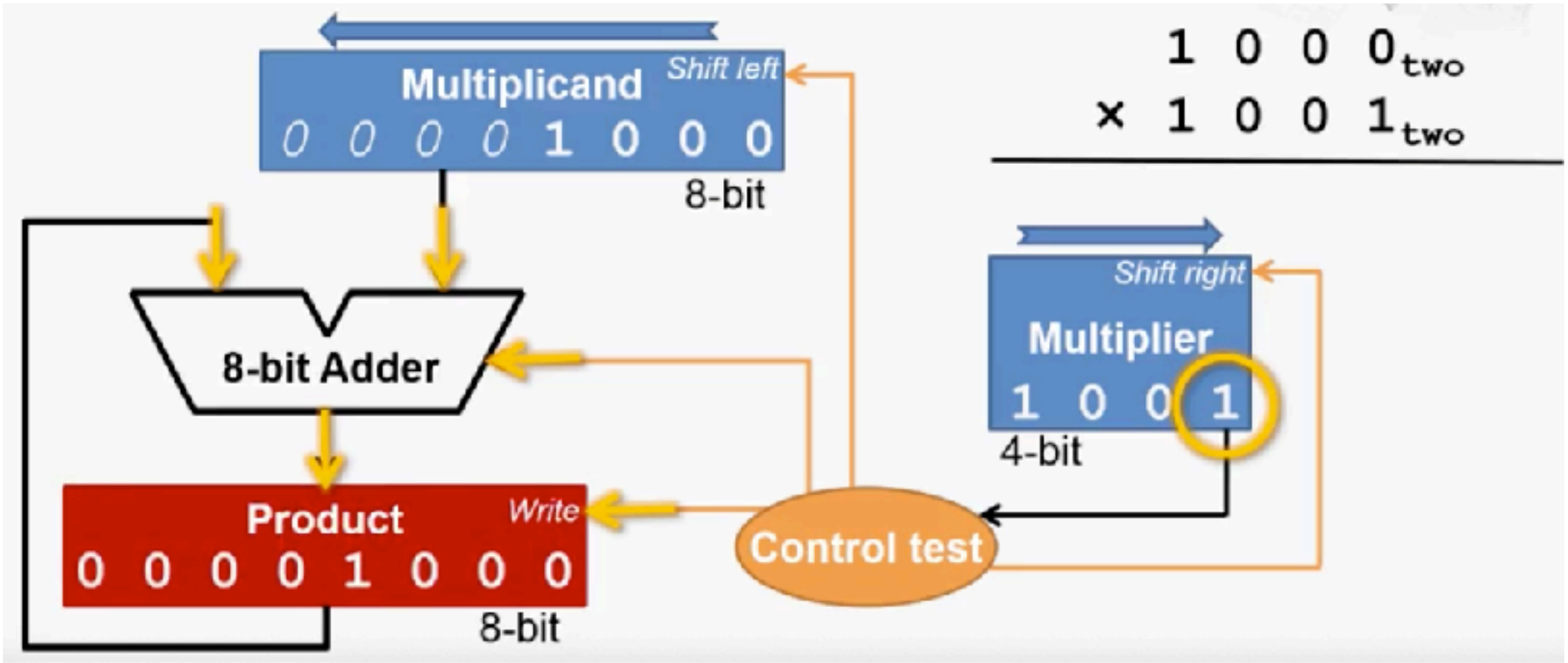


Step 2


1 0 0 0two
× 1 0 0 1two
Multiplicand
Shift left
0 0 0 0 1 0 0 0
8-bit
8-bit Adder
Shift right
Multiplier
1 0 0 1
4-bit
Product
Write
0 0 0 0 1 0 0 0
8-bit
Control test

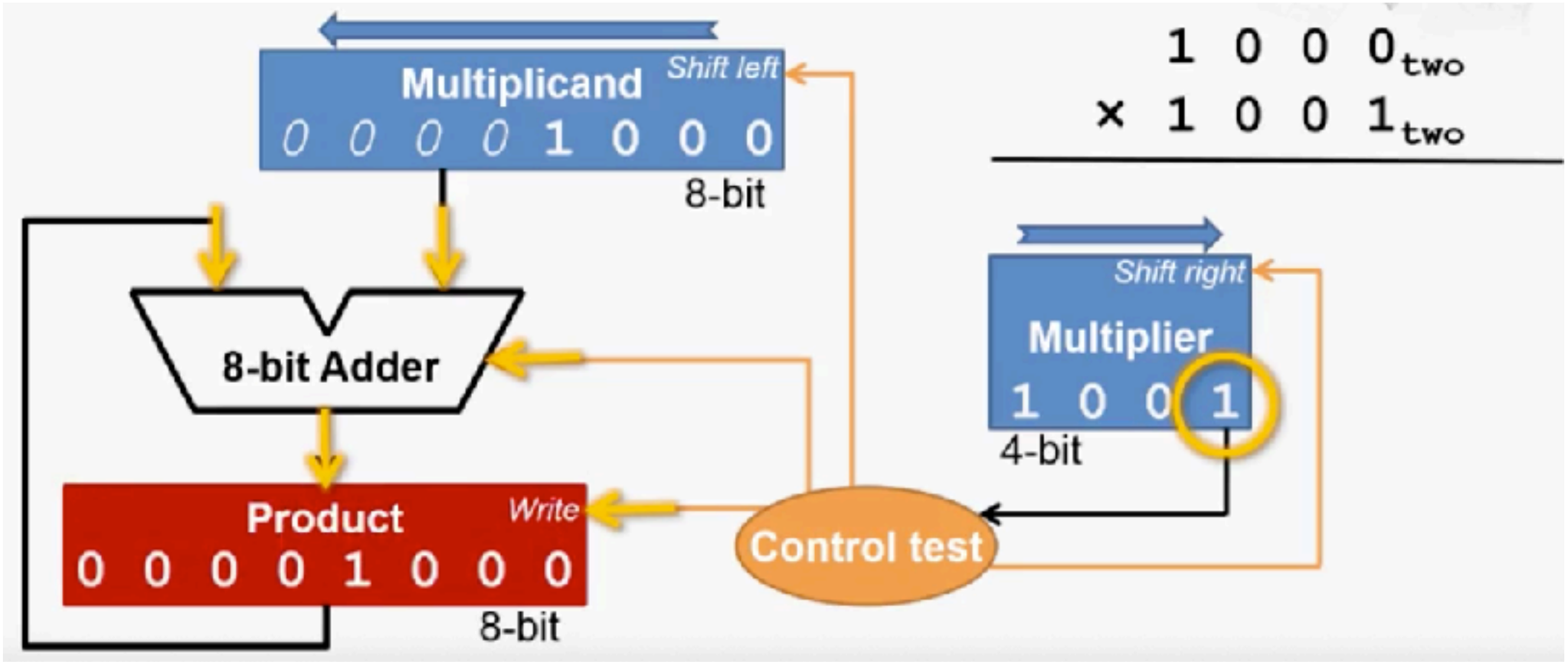
Step 2

- What should we do as step 2?



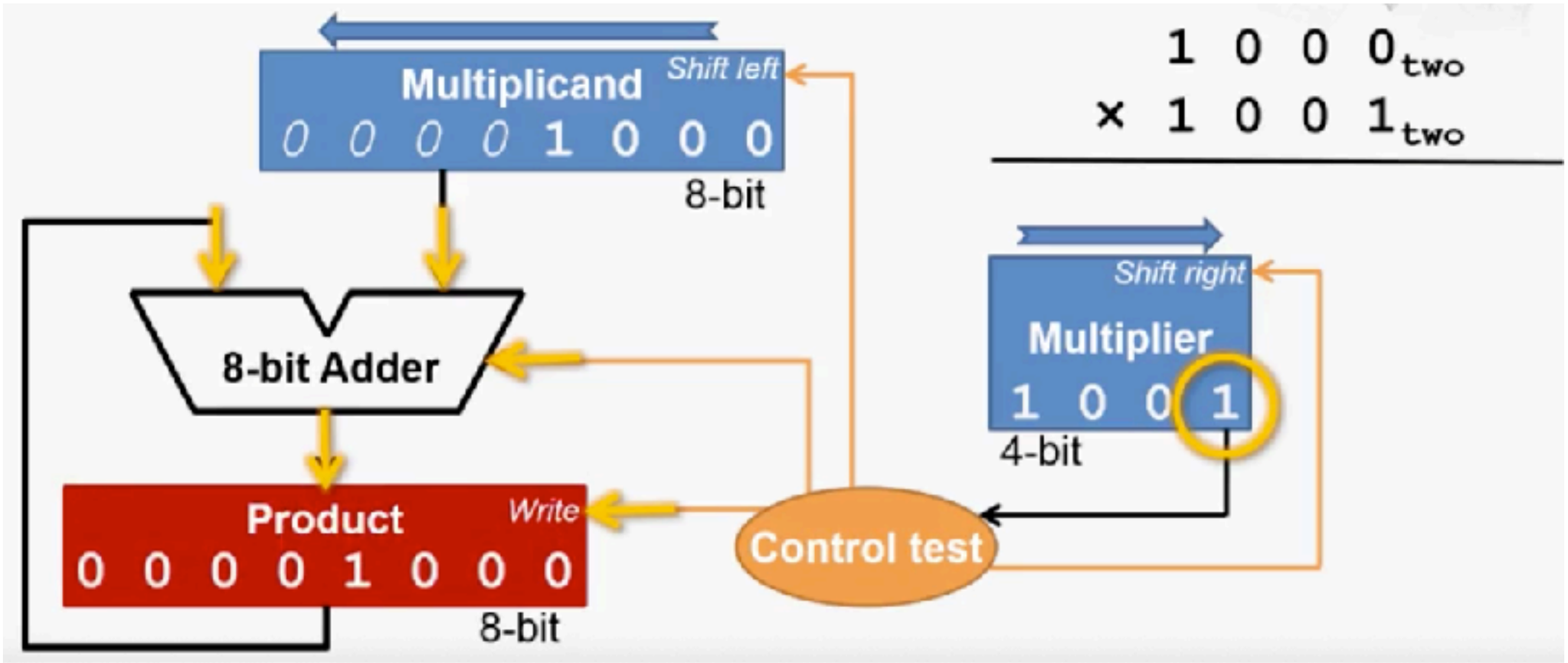
Step 2

- What should we do as step 2?
  - Recall what we did with the emulation



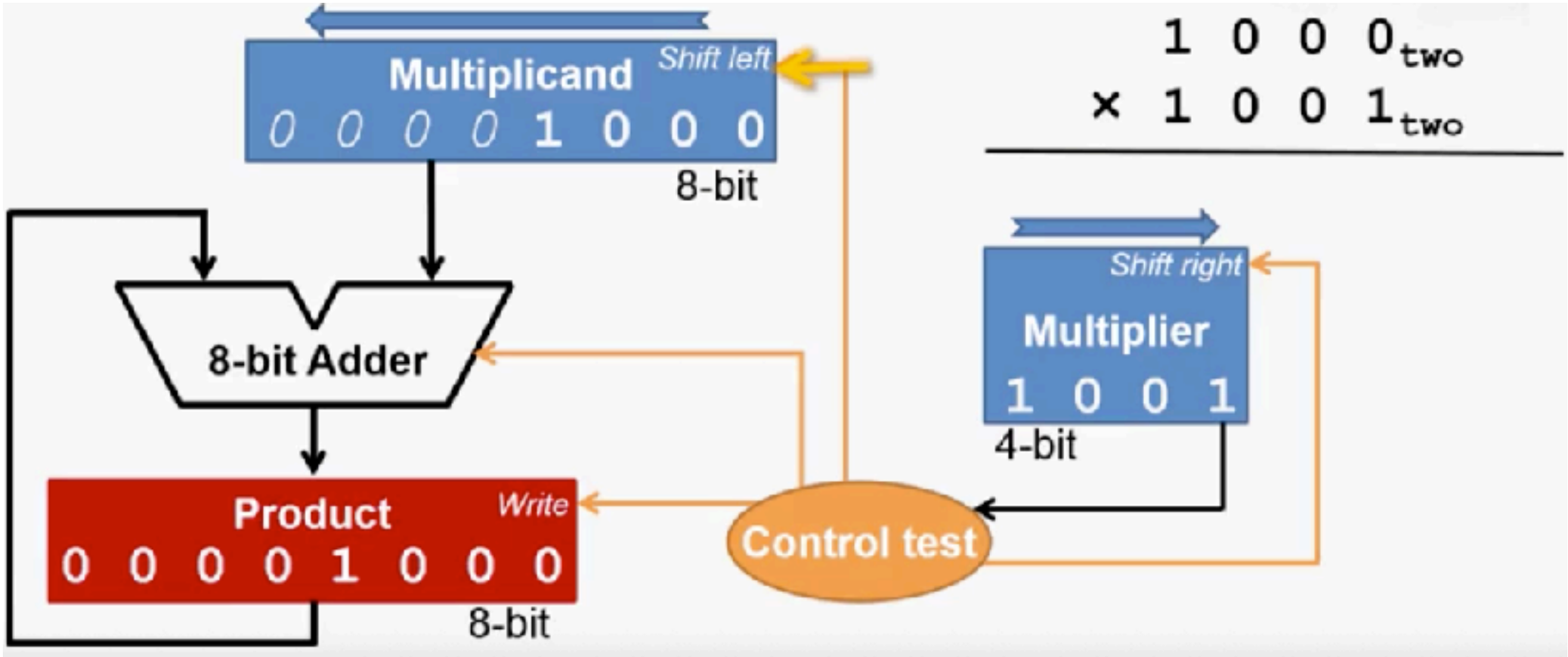
Step 2

- What should we do as step 2?
  - Recall what we did with the emulation
  - Left shift!



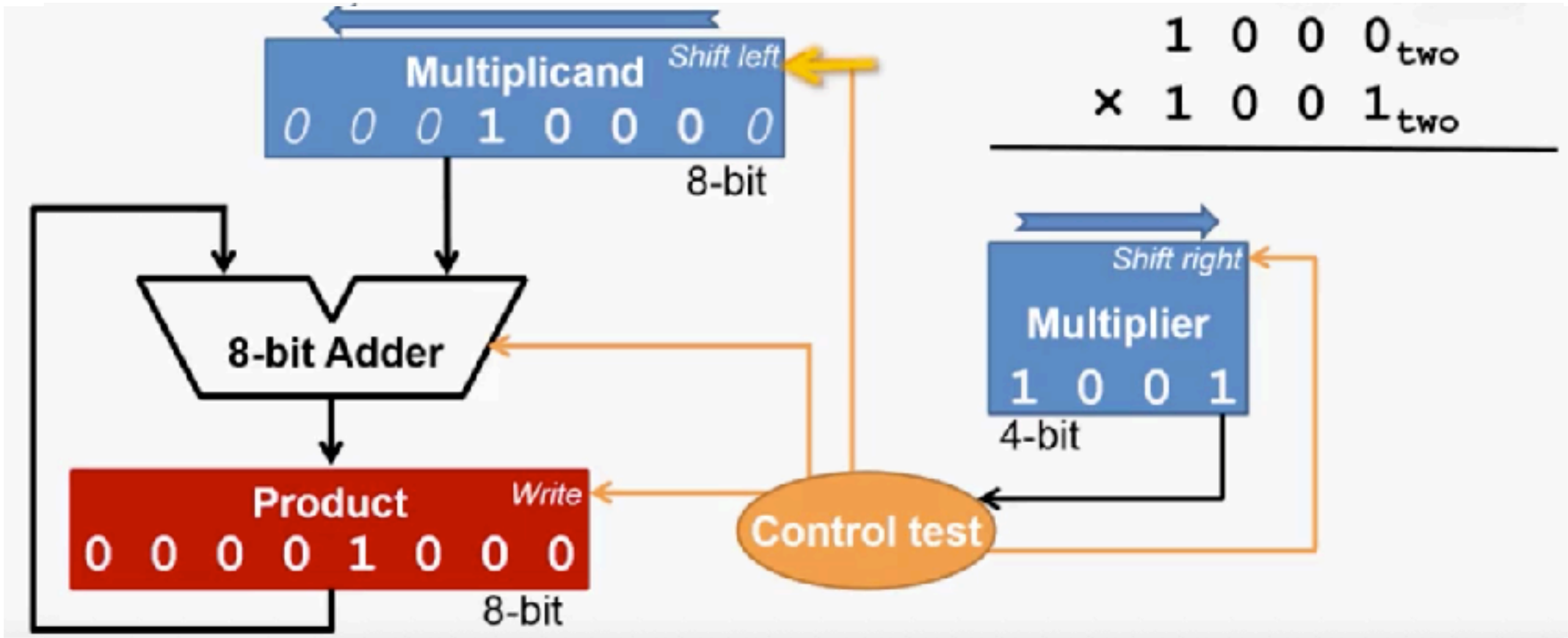
Step 2

- What should we do as step 2?
  - Recall what we did with the emulation
  - Left shift!

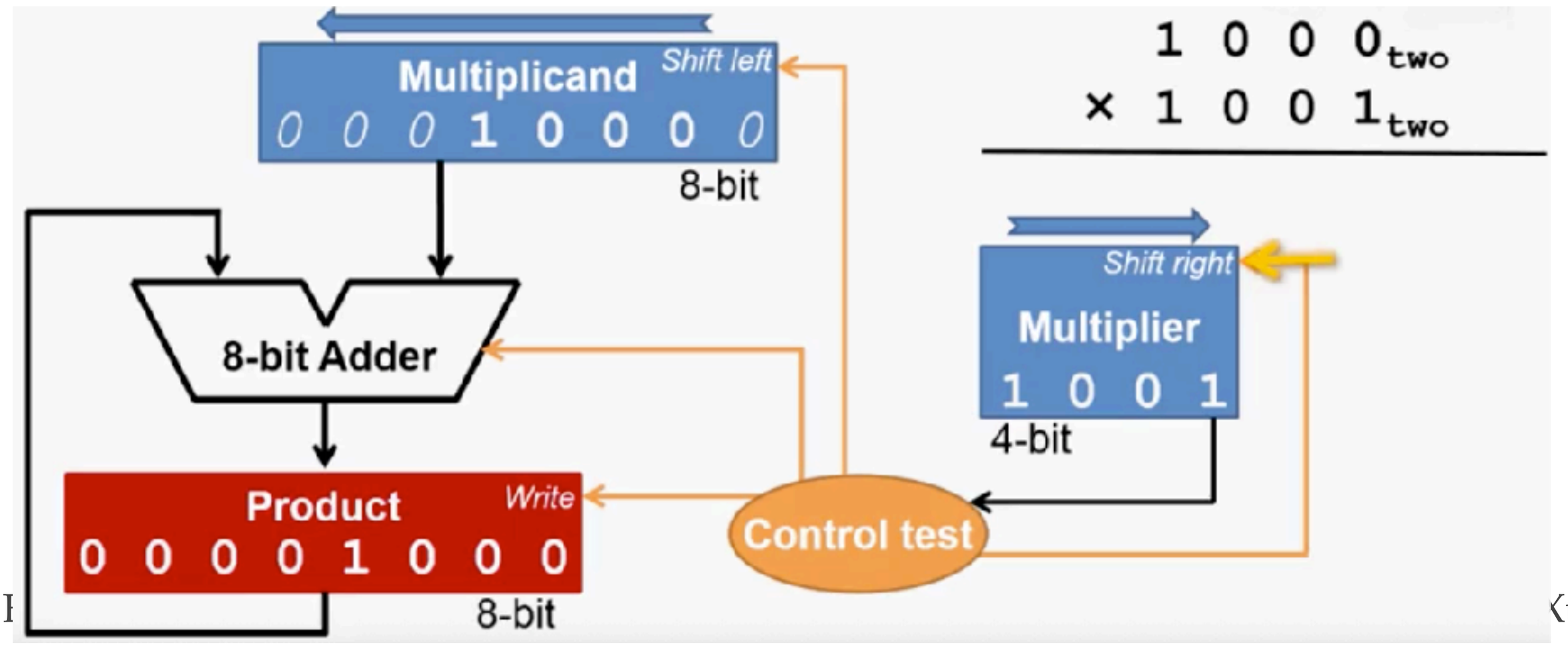


Step 2

- What should we do as step 2?
  - Recall what we did with the emulation
  - Left shift!

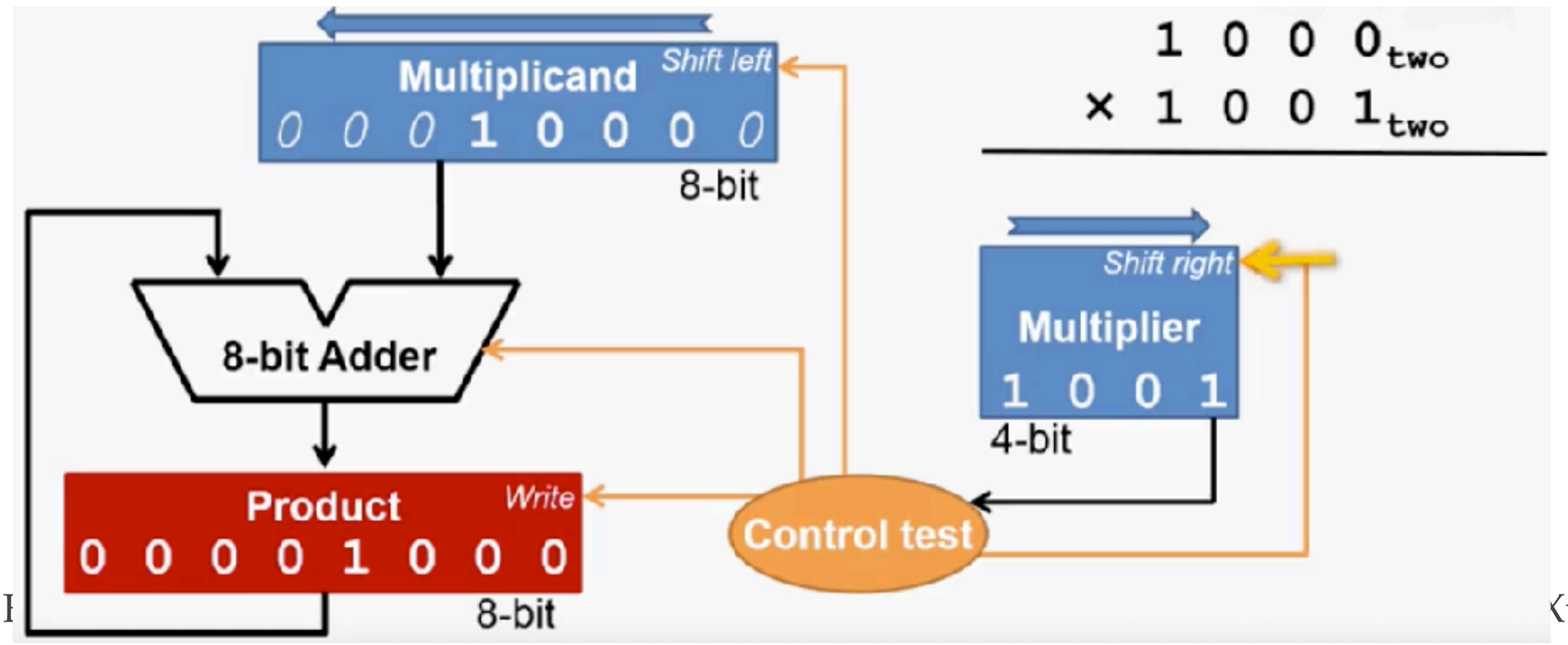


Step 3


1 0 0 0 two
× 1 0 0 1 two
Multiplicand
Shift left
0 0 0 1 0 0 0 0
8-bit
8-bit Adder
Shift right
Multiplier
1 0 0 1
4-bit
Product
Write
0 0 0 0 1 0 0 0
8-bit
Control test

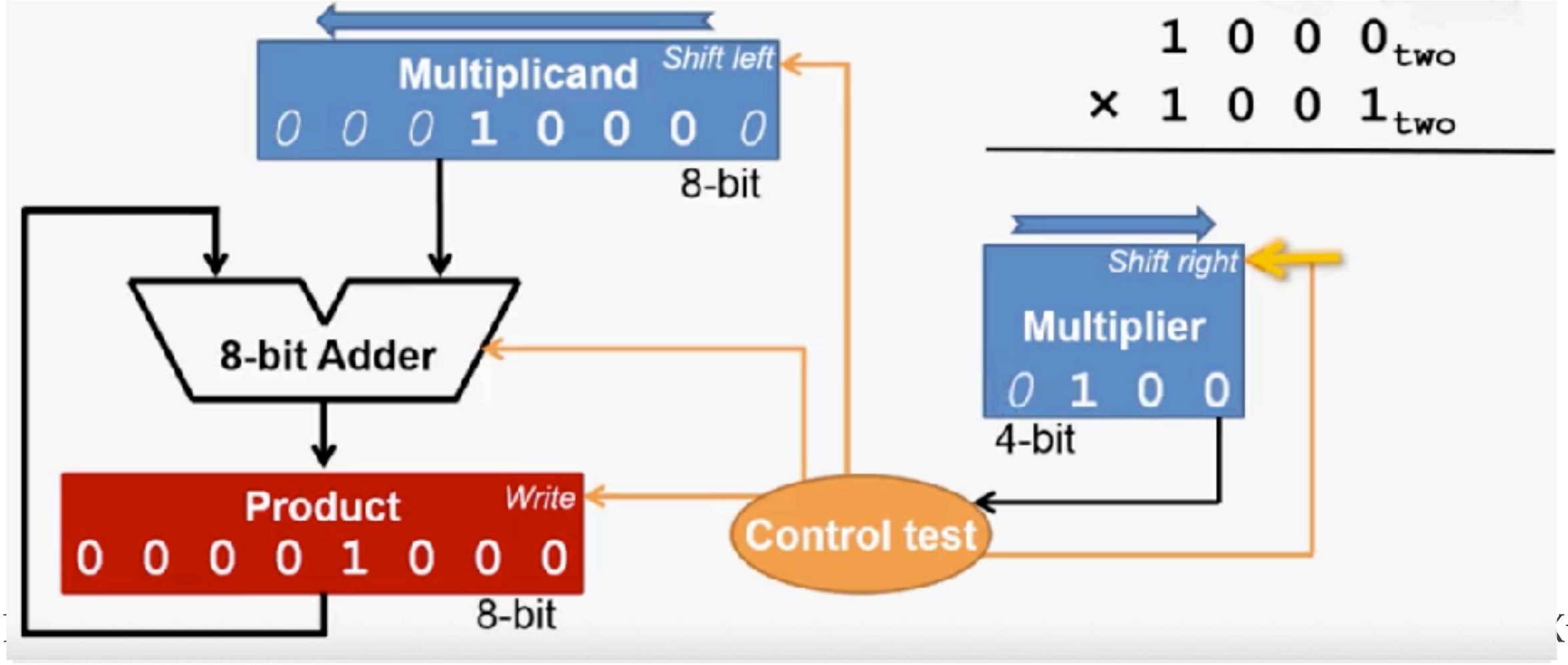
Step 3

- Right shift of multiplier



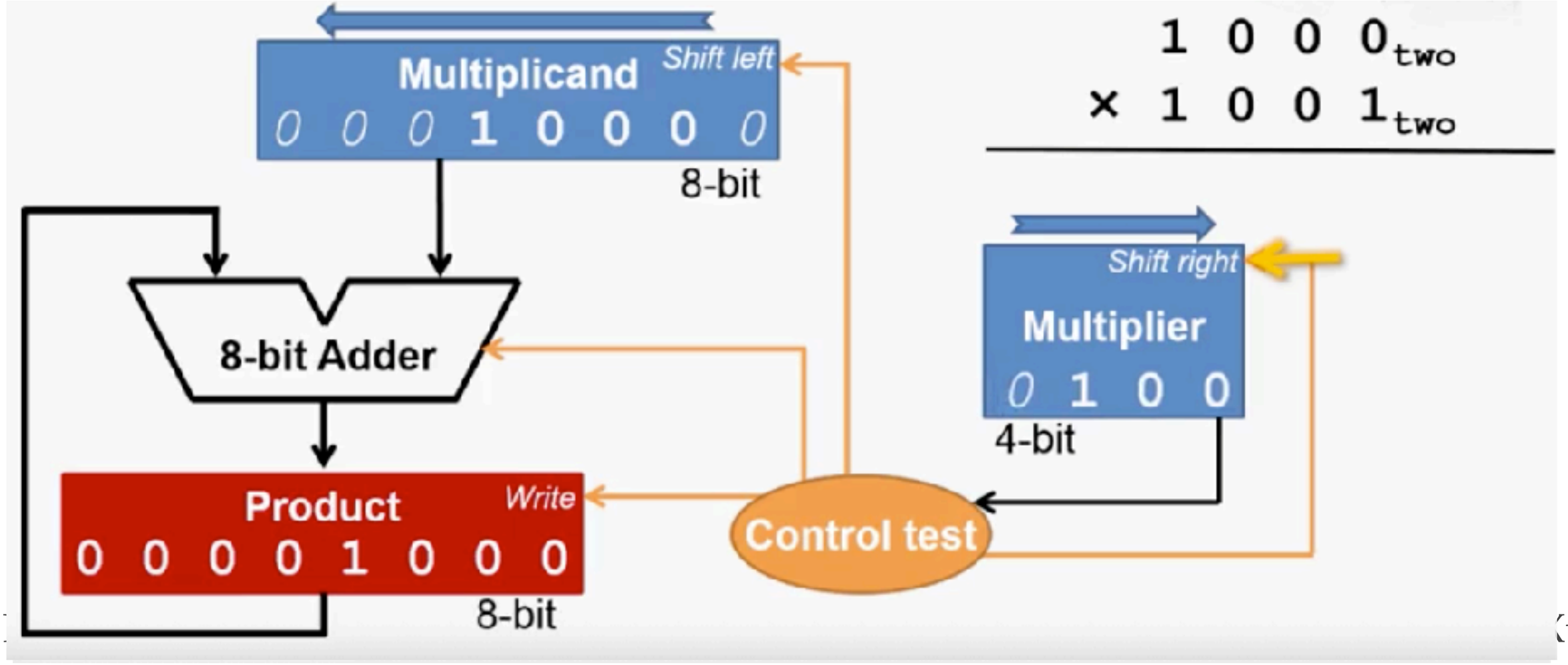
Step 3

- Right shift of multiplier



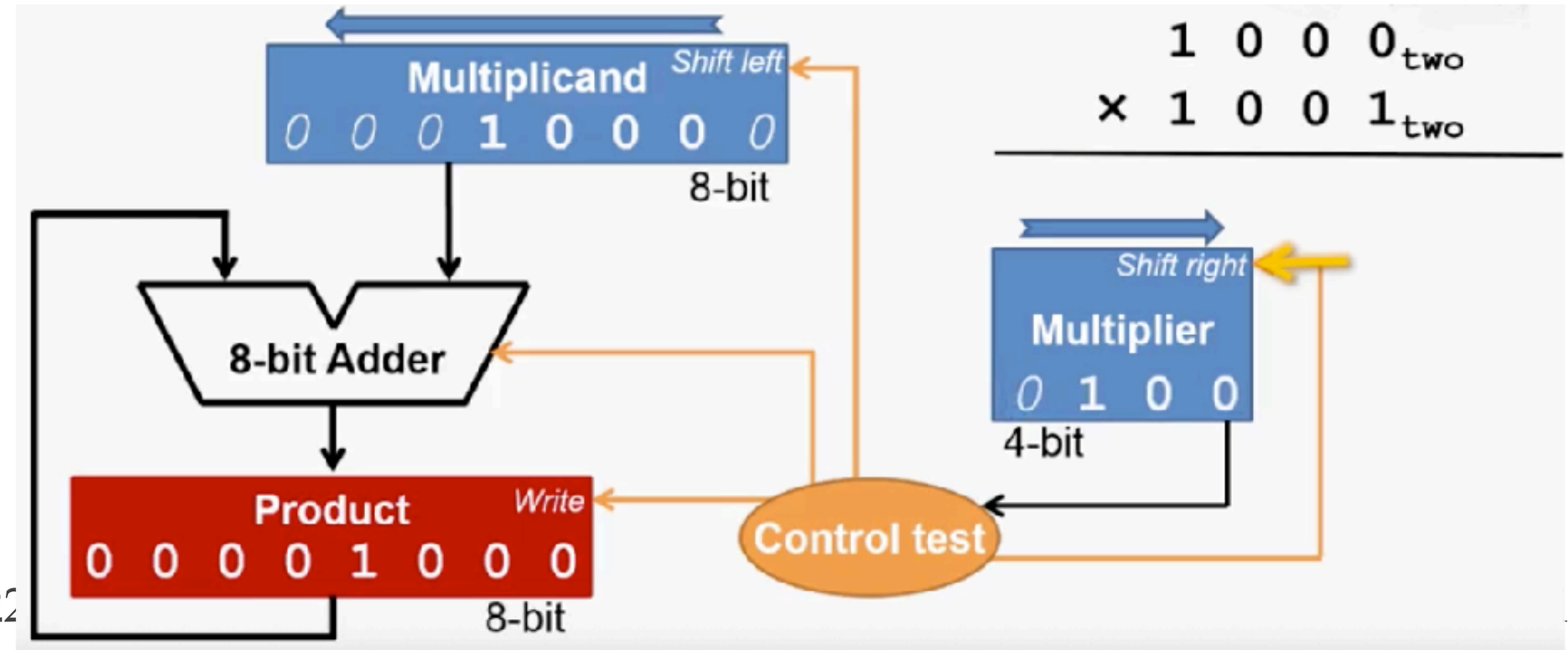
Step 3

- Right shift of multiplier
  - Instead of shifting, we can set up indicator to check which data we operate in last round, however, complicated! Now, only LSB!



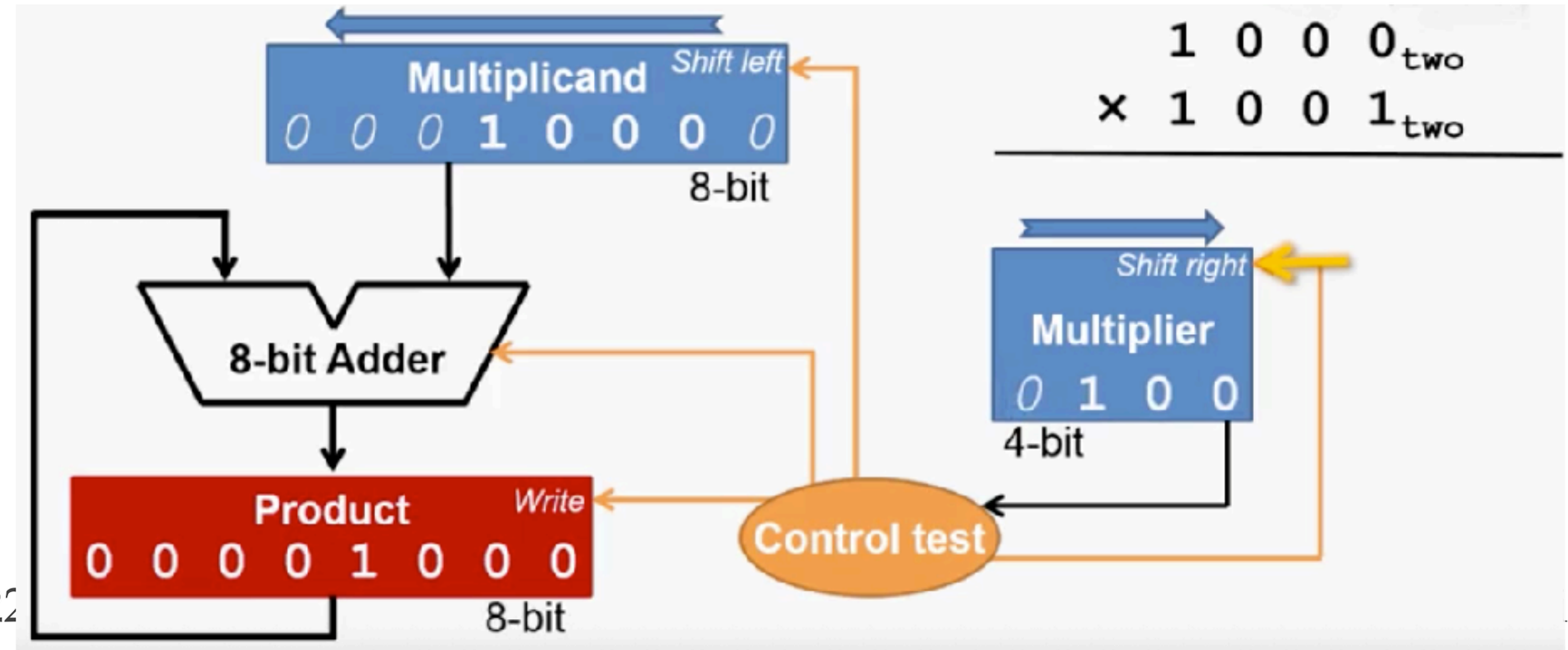
Analysis of Step 2 and 3

- Why we can do the right shift of multiplier?



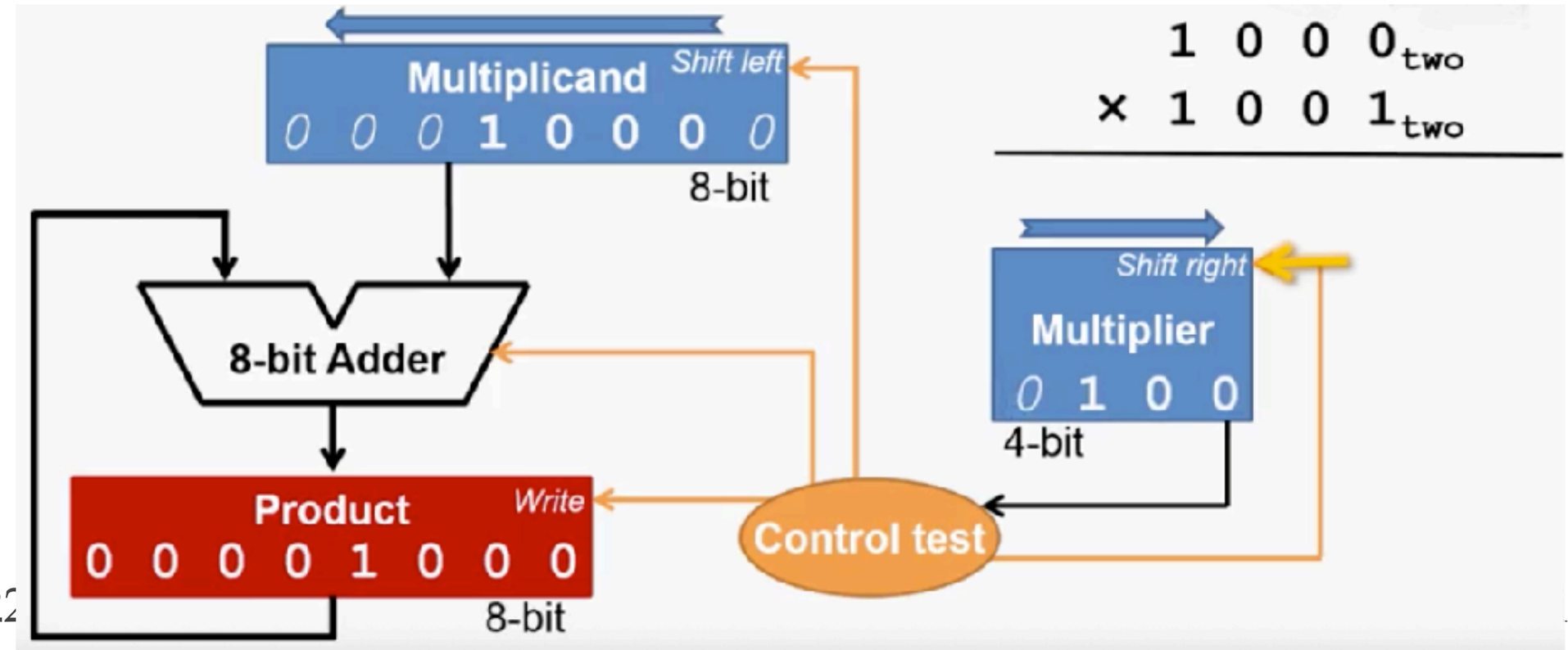
Analysis of Step 2 and 3

- Why we can do the right shift of multiplier?
  - The data has no use anymore



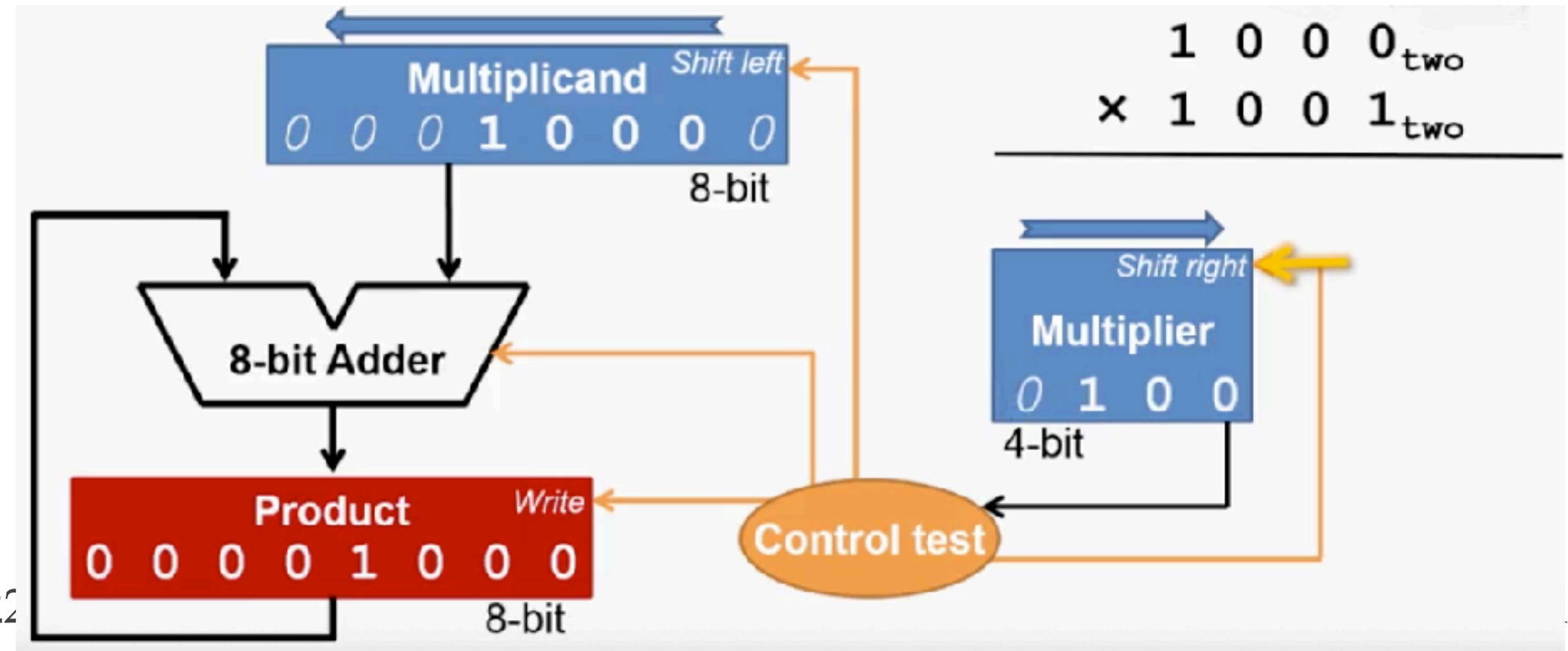
Analysis of Step 2 and 3

- Why we can do the right shift of multiplier?
  - The data has no use anymore
- Step 2 and 3 are making preparation for next round!

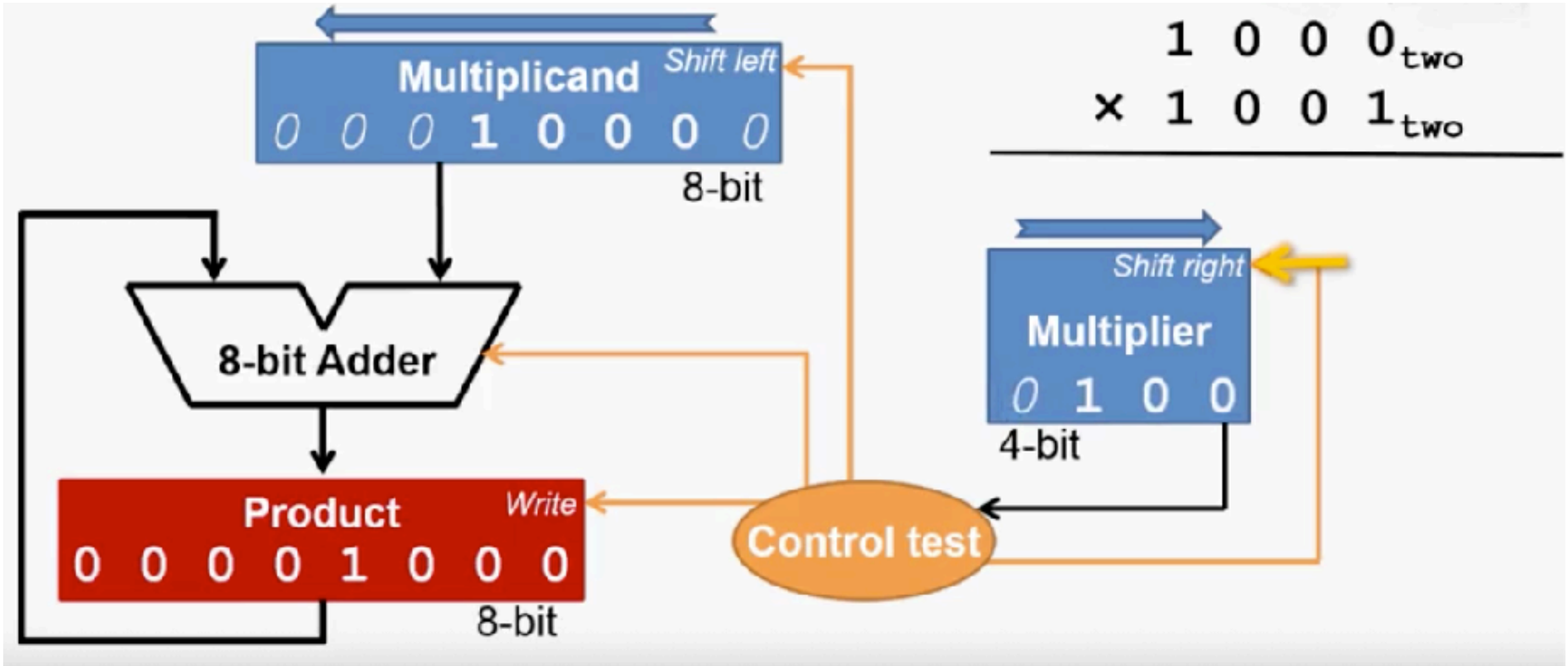


Analysis of Step 2 and 3

- Why we can do the right shift of multiplier?
  - The data has no use anymore
- Step 2 and 3 are making preparation for next round!
- But hold on, do we have next round or not?

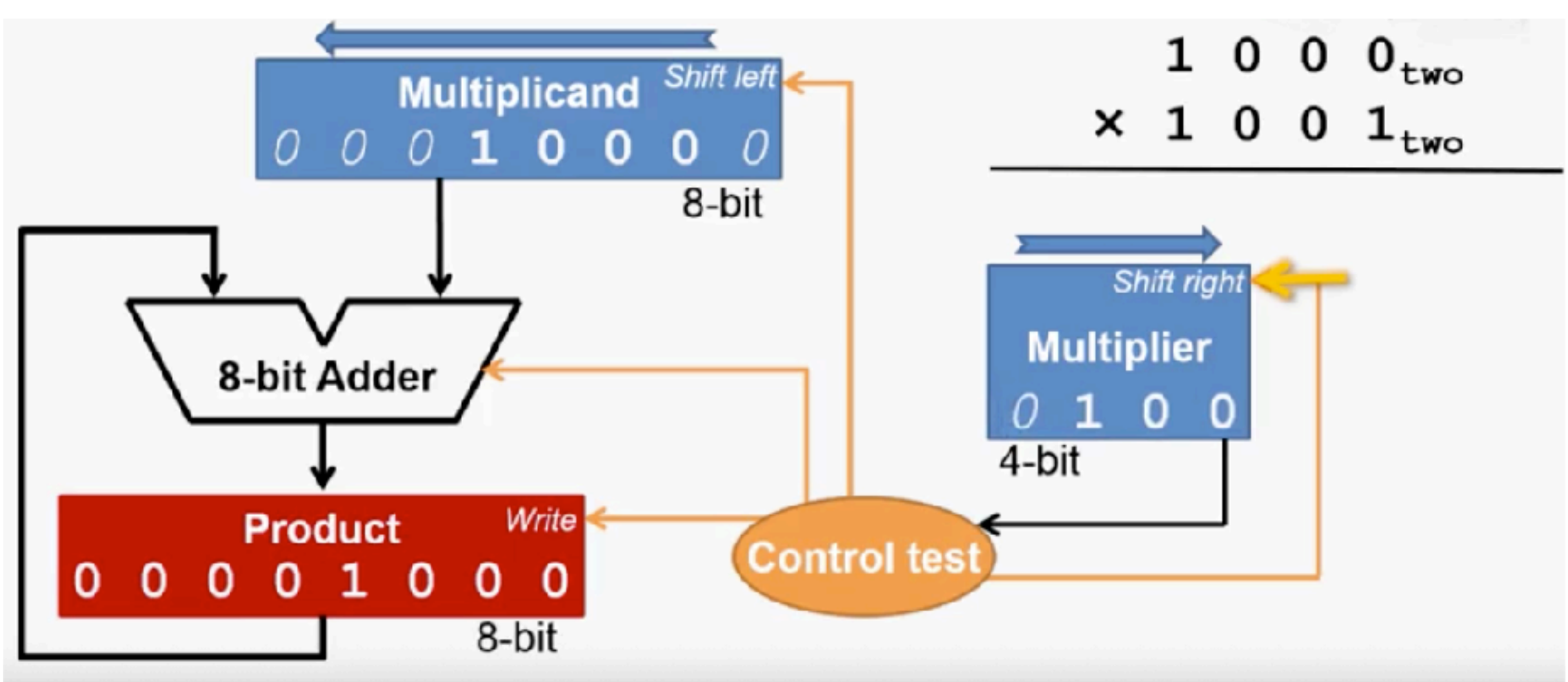


Step 4


1 0 0 0 two
× 1 0 0 1 two
Multiplicand
Shift left
0 0 0 1 0 0 0 0
8-bit
8-bit Adder
Shift right
Multiplier
0 1 0 0
4-bit
Product
Write
0 0 0 0 1 0 0 0
8-bit
Control test

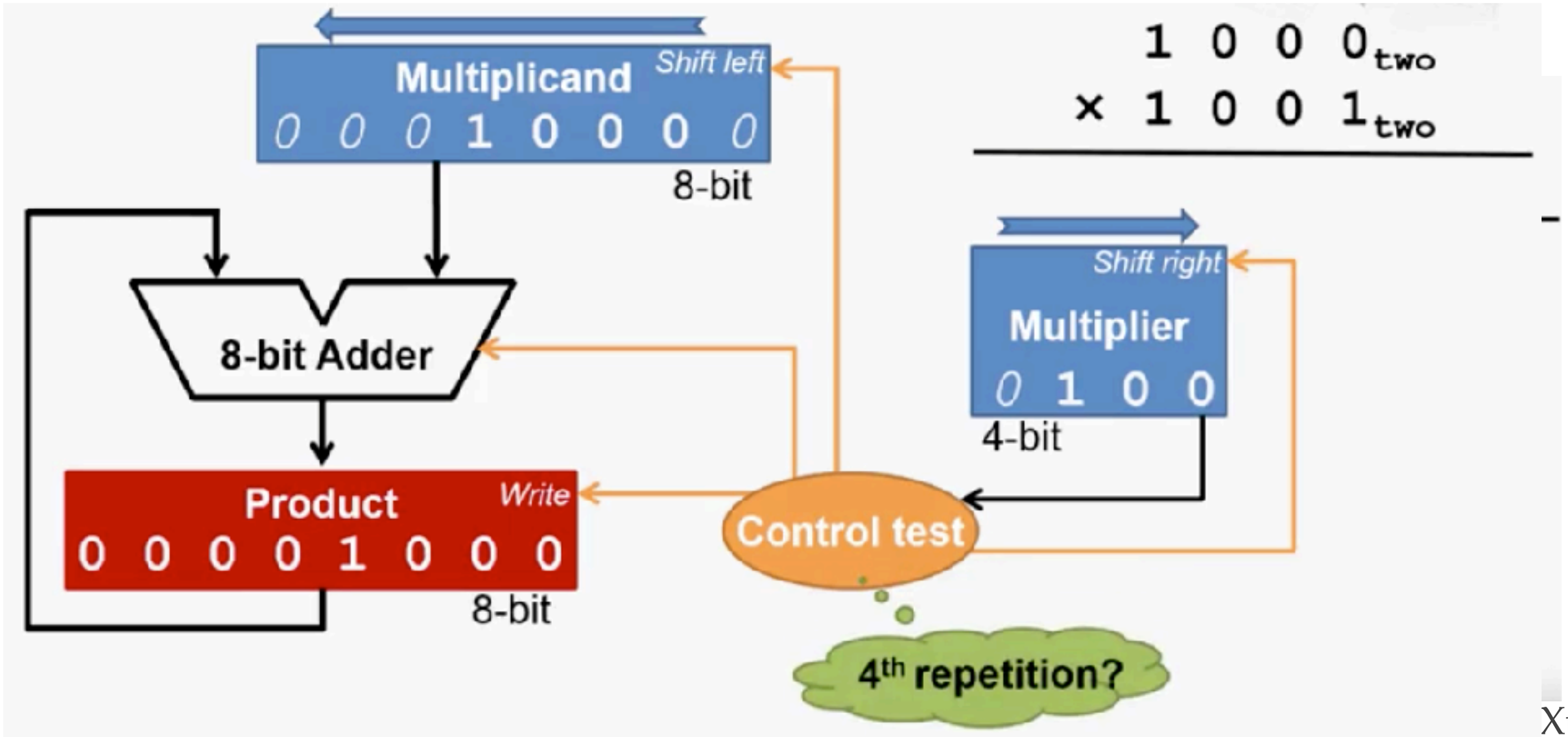
Step 4

- Control test module decide whether it is already the last round? I.e., the 4th round?



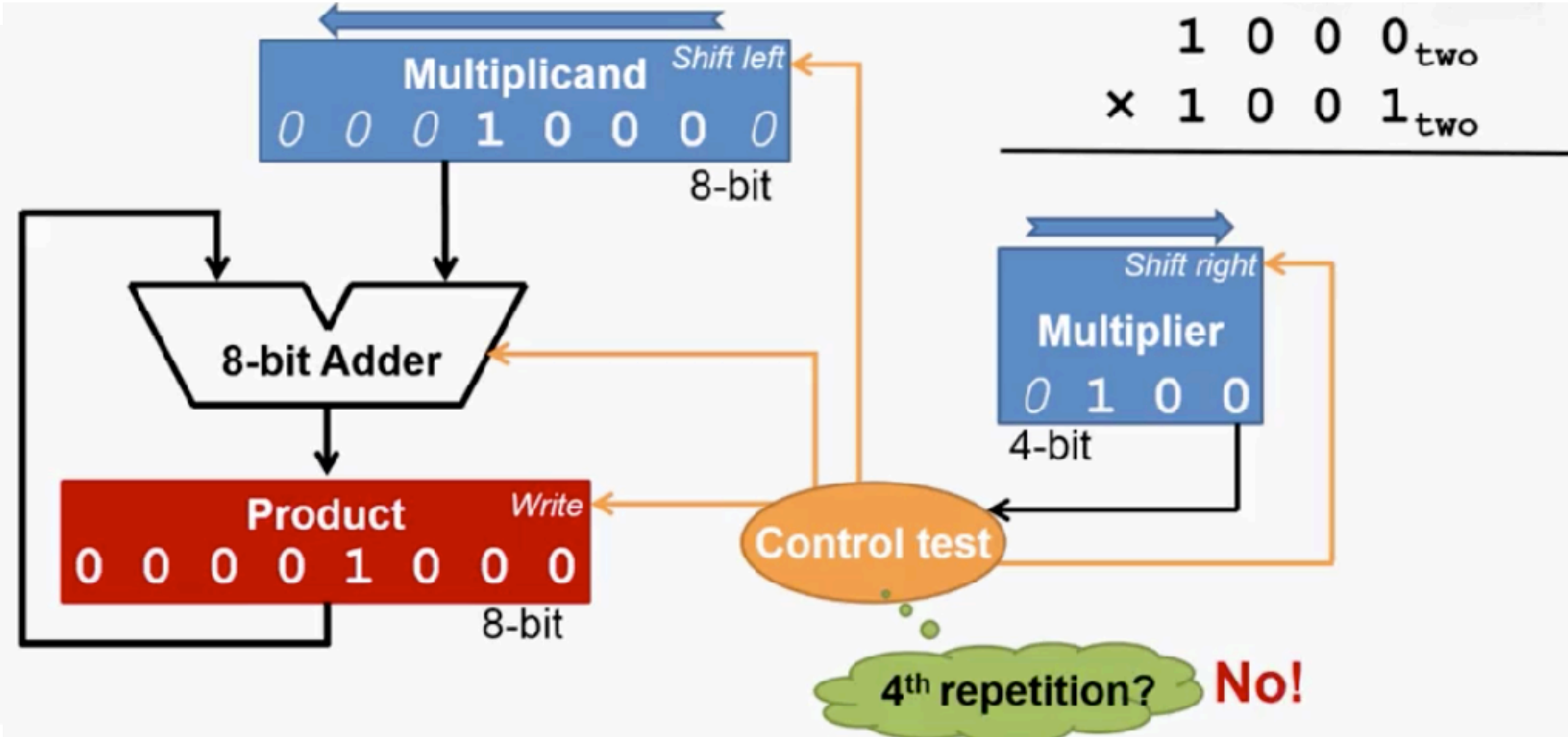
Step 4

- Control test module decide whether it is already the last round? I.e., the 4th round?



Step 4

- Control test module decide whether it is already the last round? I.e., the 4th round?



Control Test

- Another important module
- Repetition checking
- Simply counter can do it

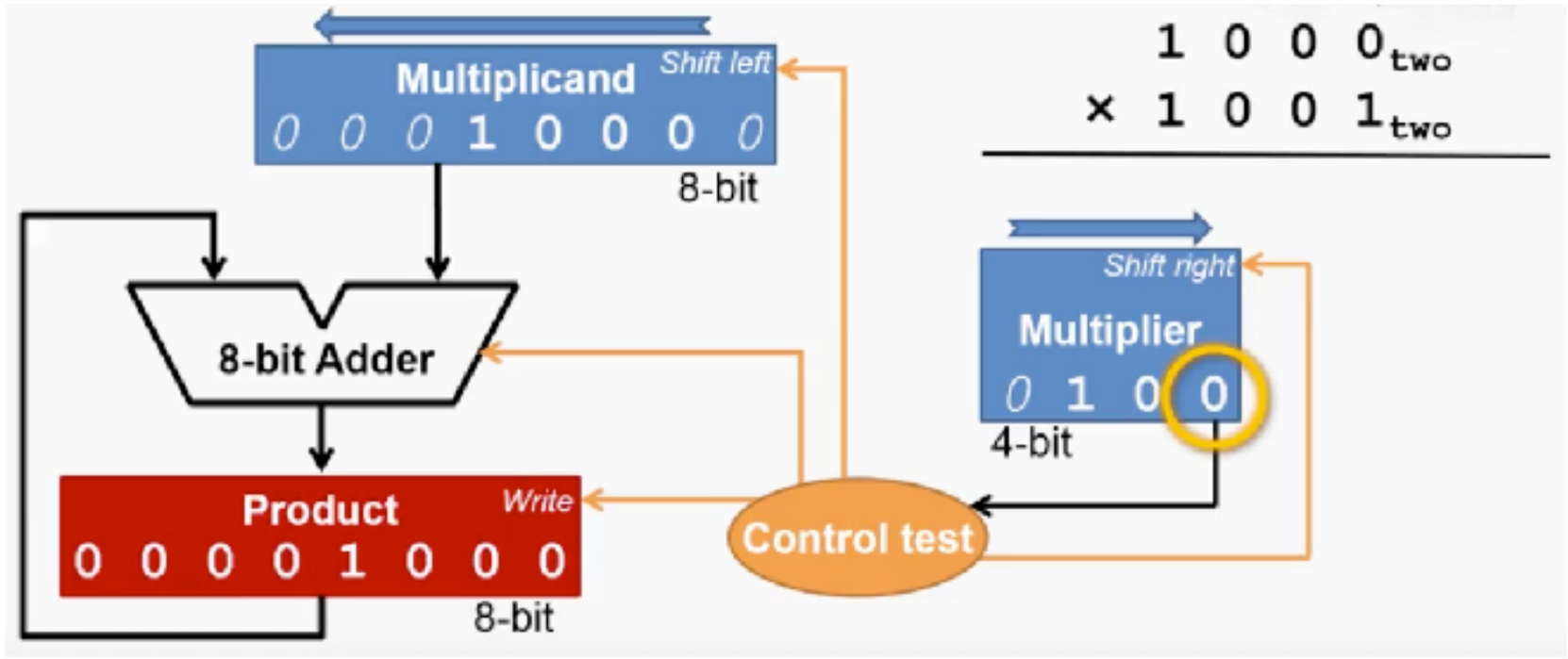
Step 1 of Round 2

Step 1 of Round 2

- 1, Check the LSB of multiplier

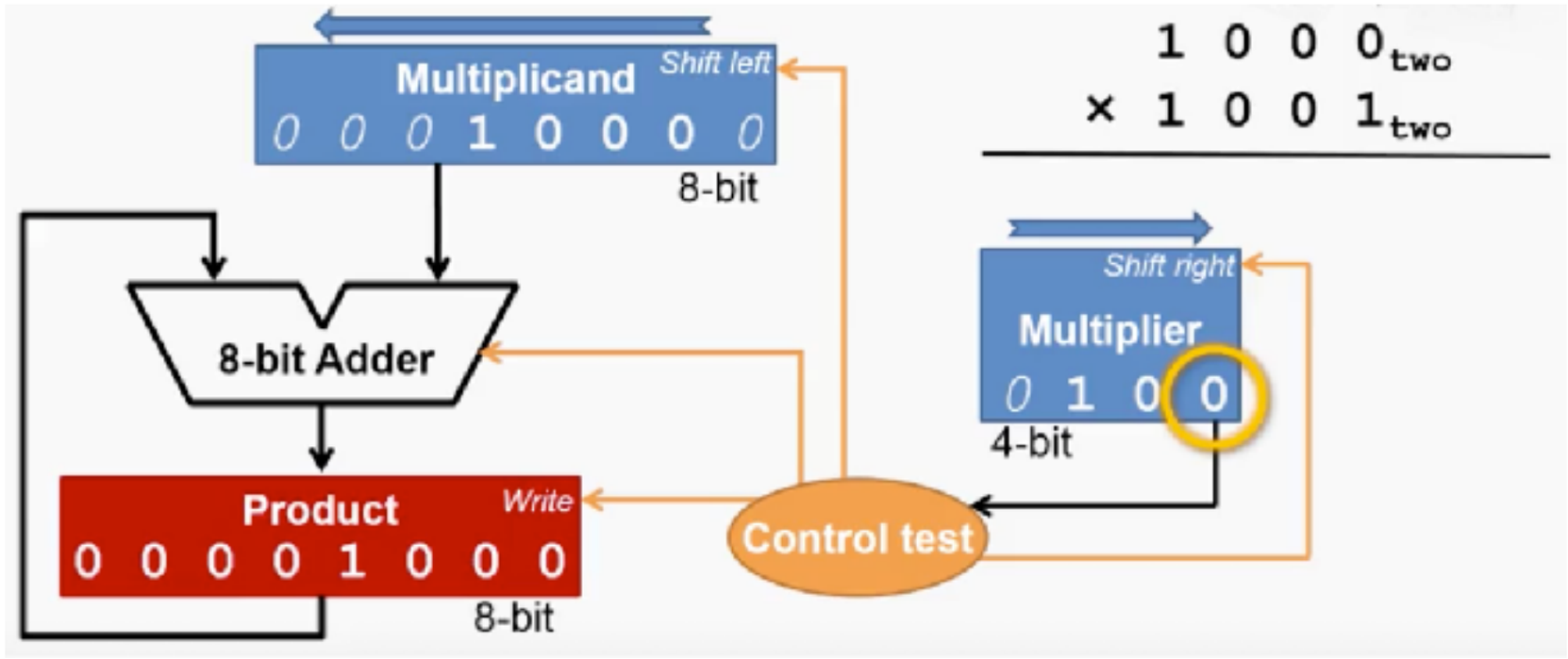
Step 1 of Round 2

- 1, Check the LSB of multiplier



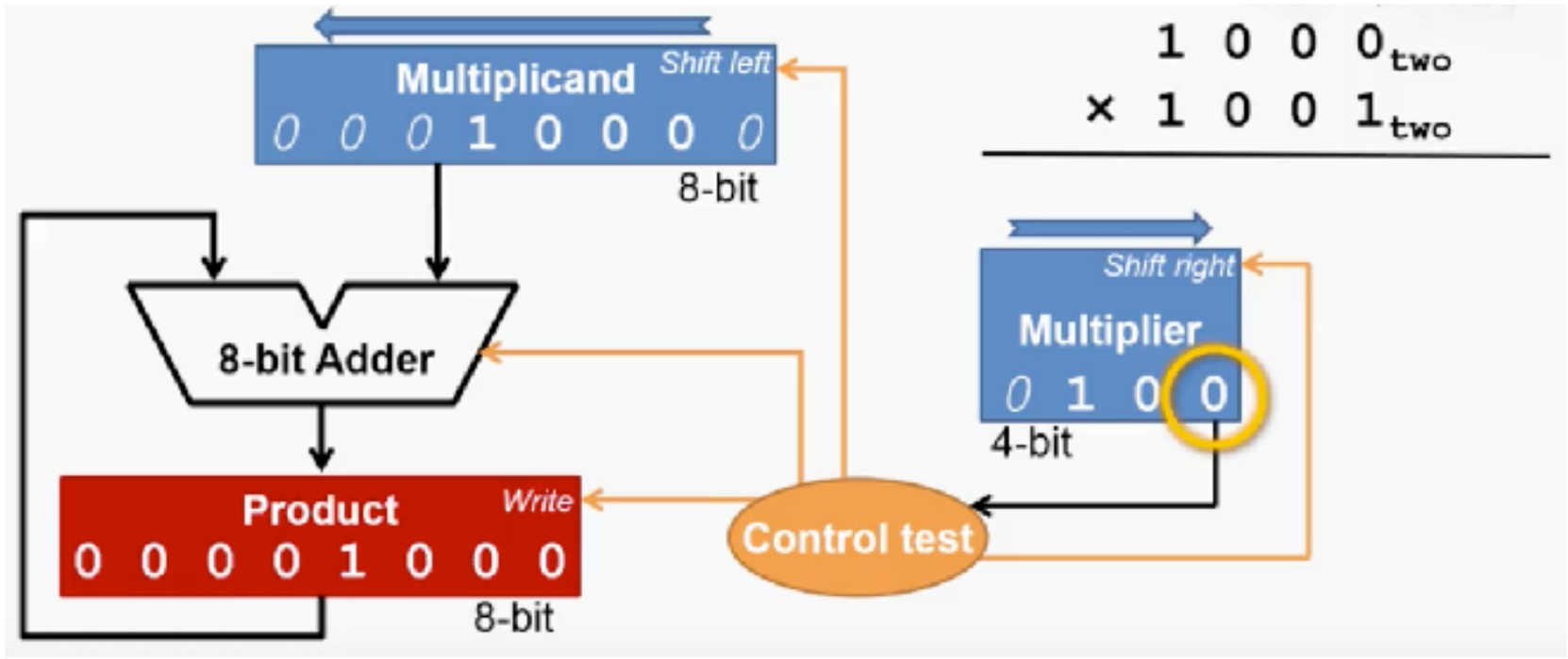
Step 1 of Round 2

- 1, Check the LSB of multiplier
  - If 1, do what?



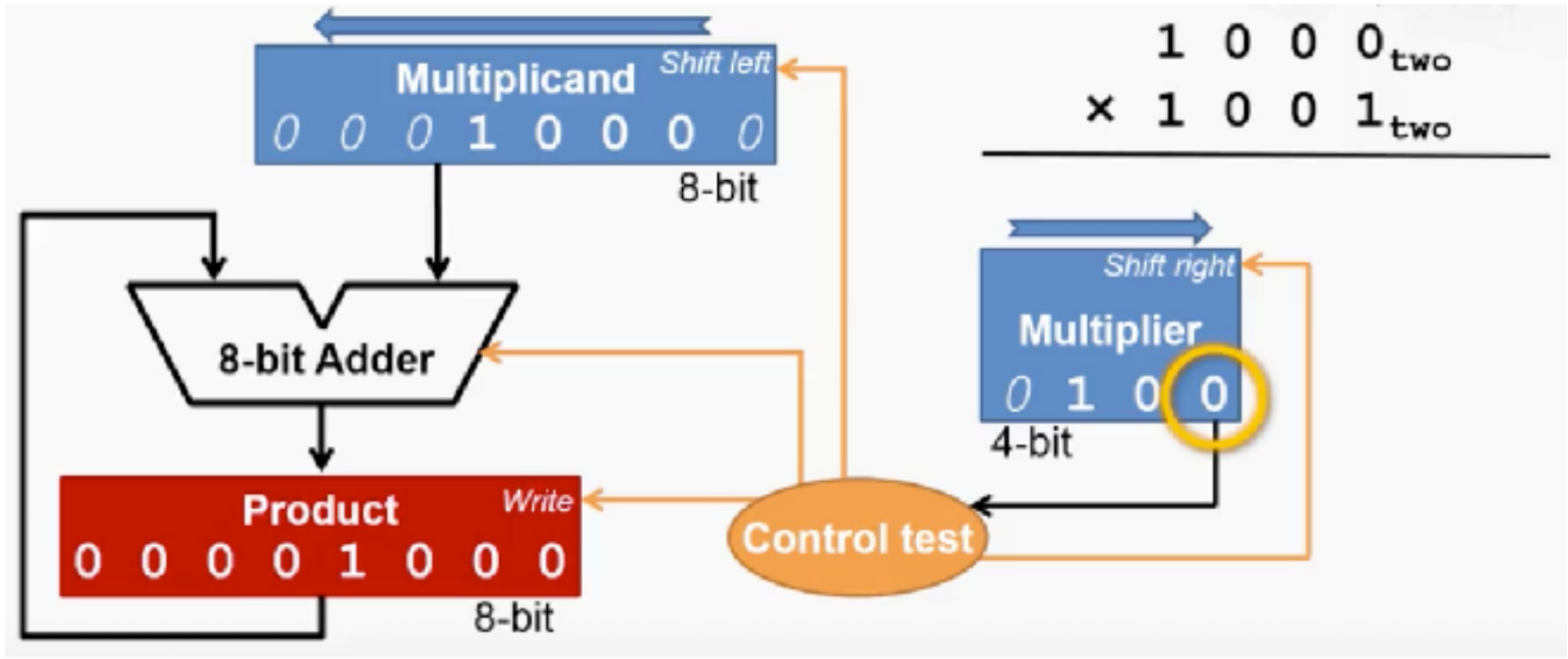
Step 1 of Round 2

- 1, Check the LSB of multiplier
  - If 1, do what?
  - If 0, do what?



Step 1 of Round 2

- 1, Check the LSB of multiplier
  - If 1, do what?
  - If 0, do what?
- We get 0, do nothing, no step 2 of round 2

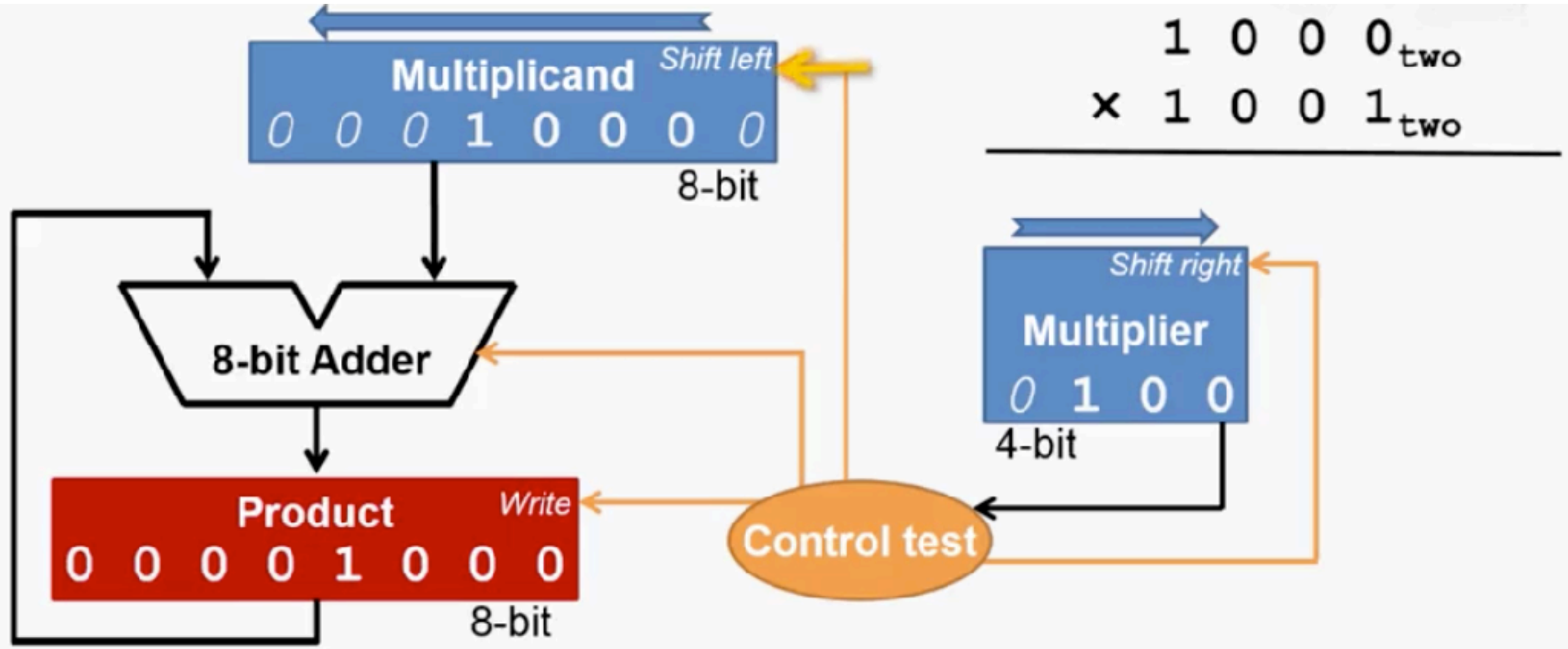


Step 2 of Round 2

- What does this step do?
  - Left shift of multiplicand

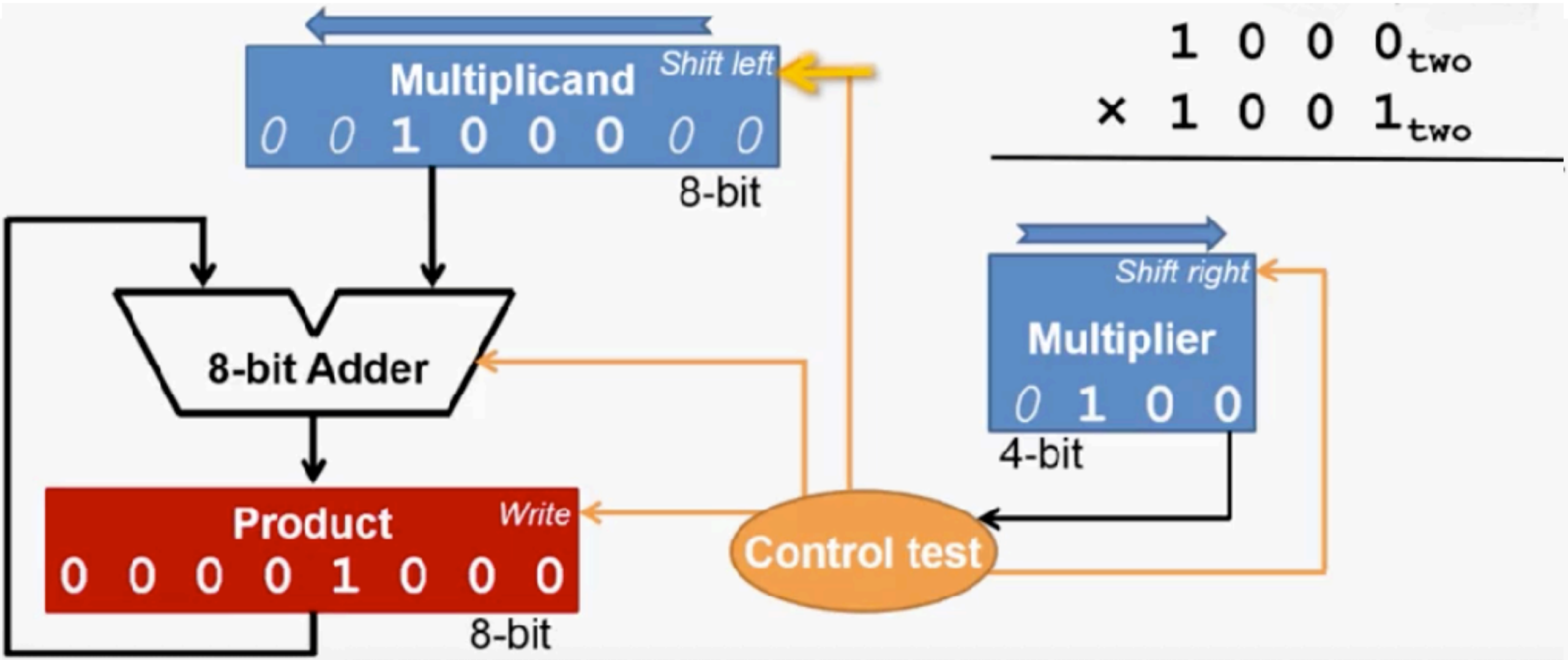
Step 2 of Round 2

- What does this step do?
  - Left shift of multiplicand



Step 2 of Round 2

- What does this step do?
  - Left shift of multiplicand

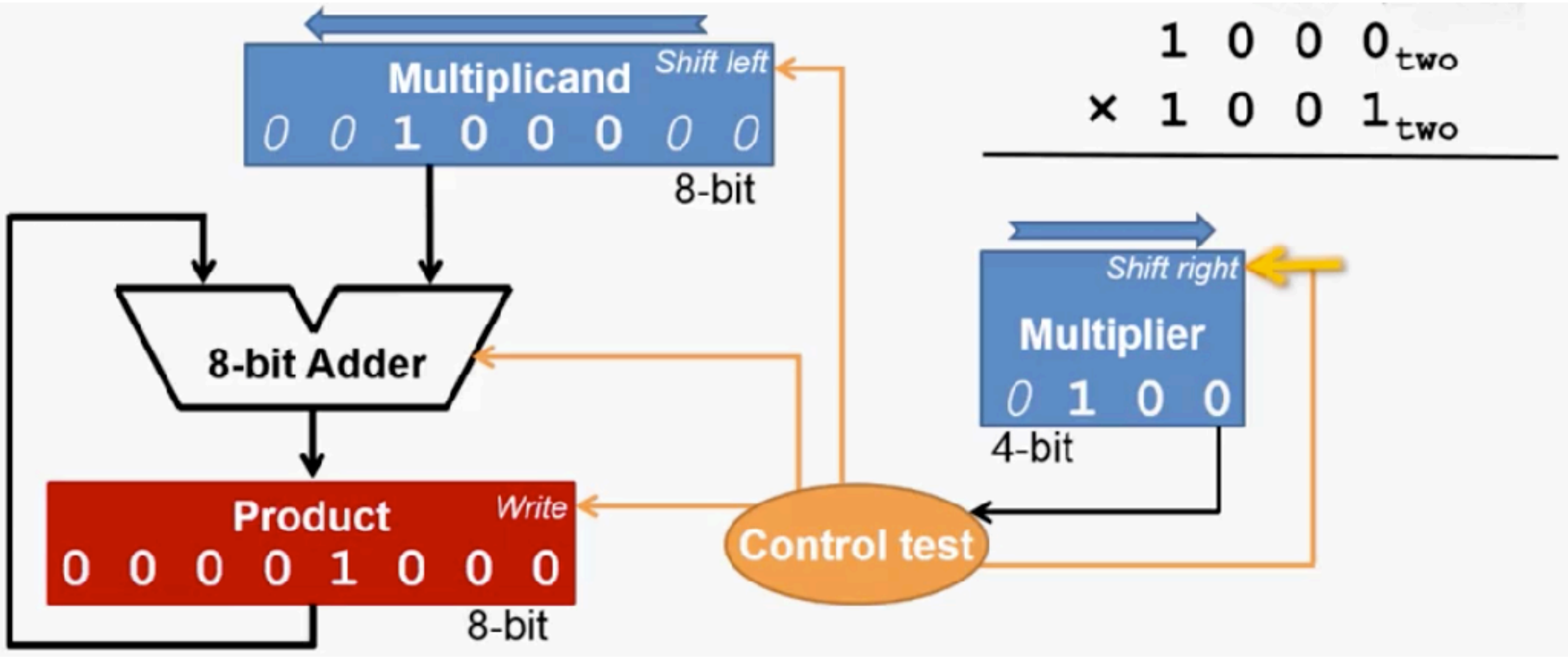


Step 3 of Round 2

- Shift right of multiplier

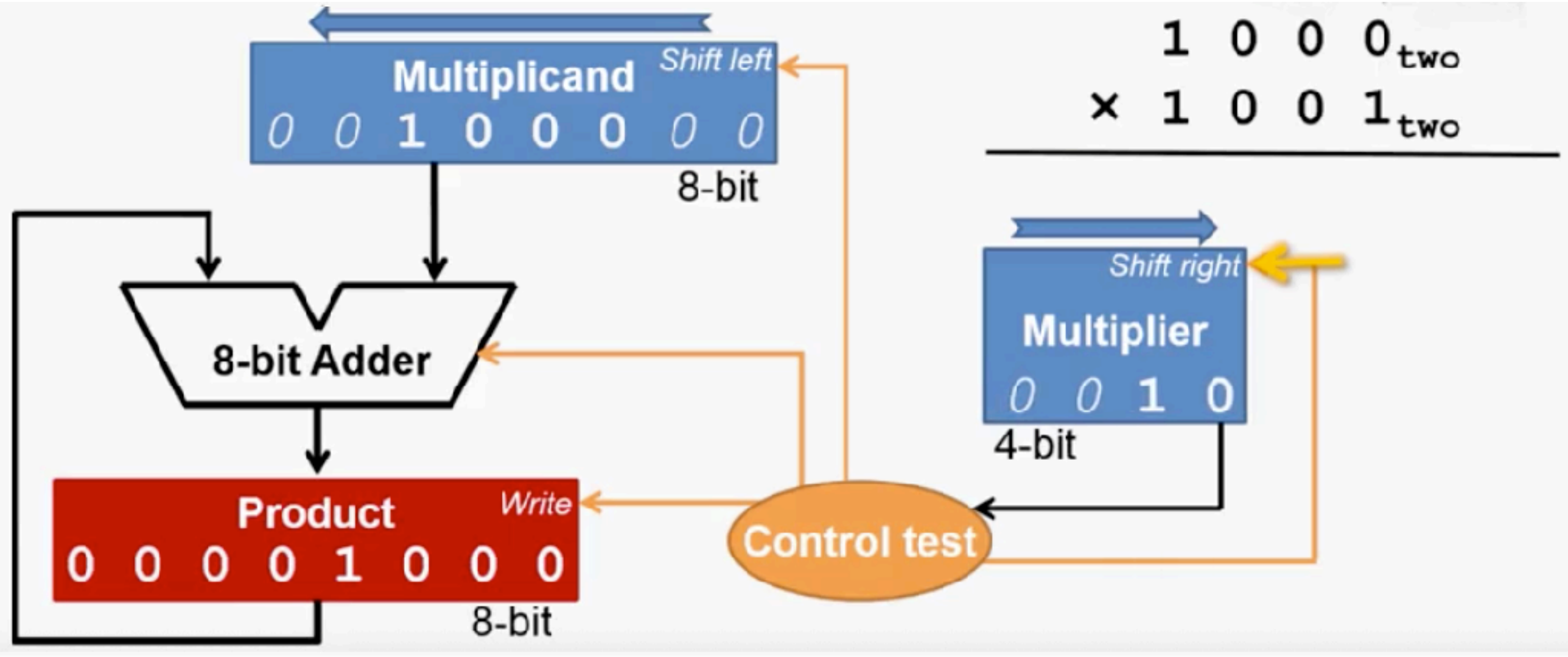
Step 3 of Round 2

- Shift right of multiplier



Step 3 of Round 2

- Shift right of multiplier



Step 4 of Round 2

Step 4 of Round 2

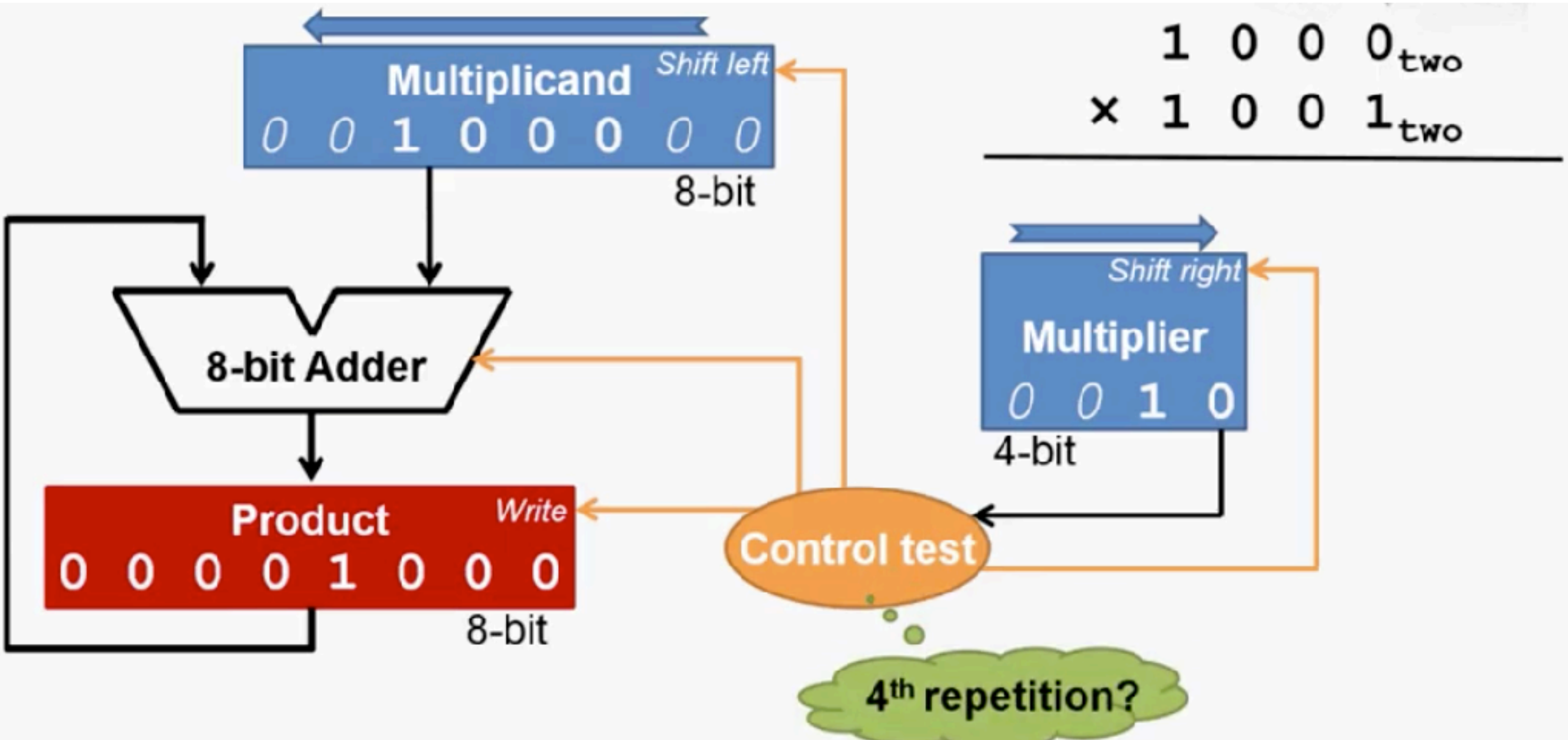
- What does this step do?

Step 4 of Round 2

- What does this step do?
  - Repetition checking!

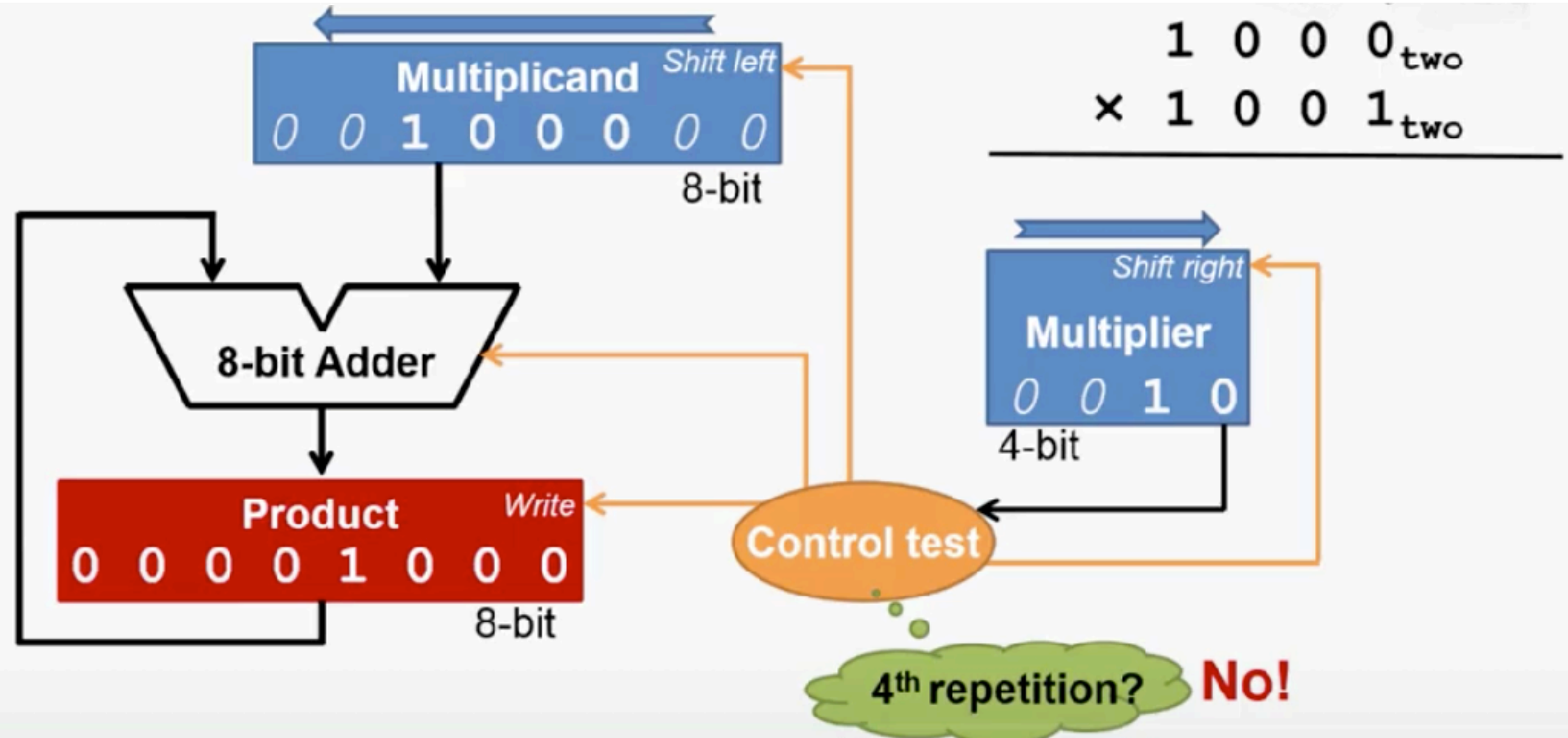
Step 4 of Round 2

- What does this step do?
  - Repetition checking!

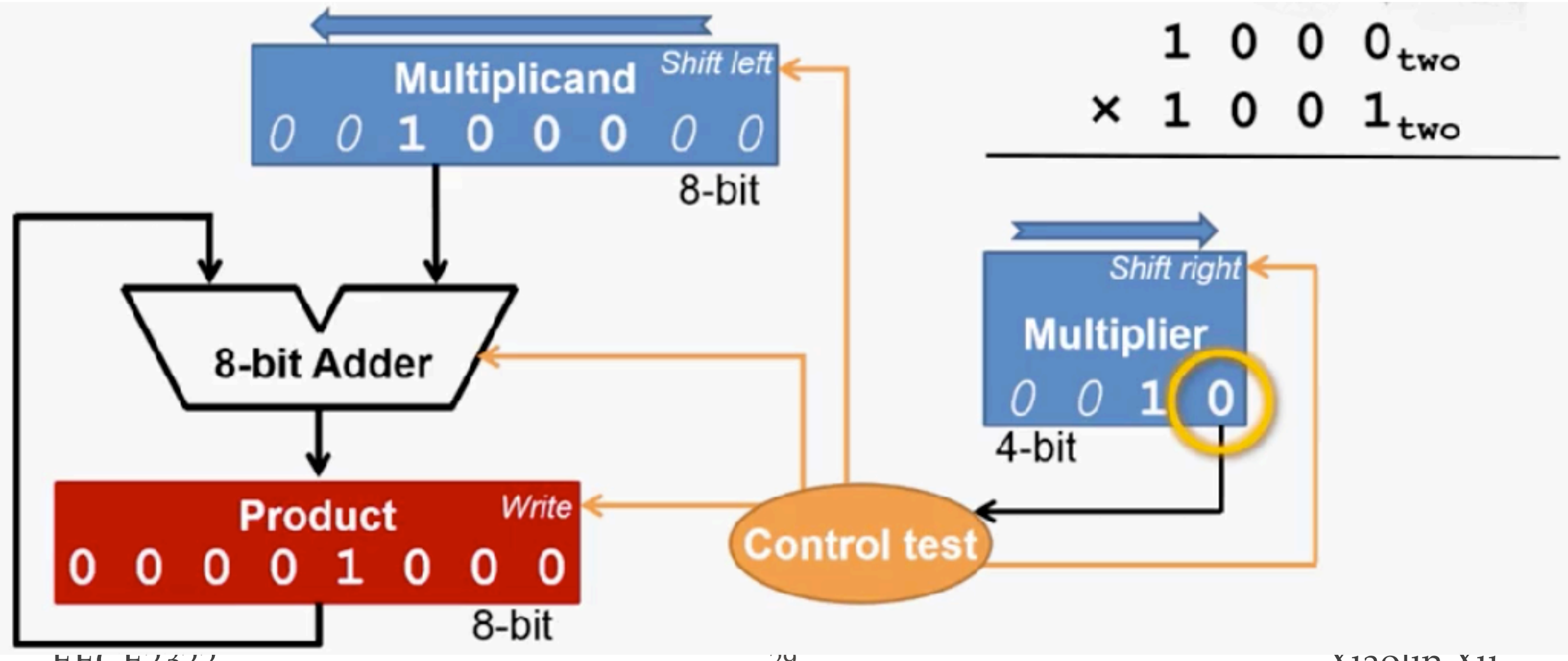


Step 4 of Round 2

- What does this step do?
  - Repetition checking!

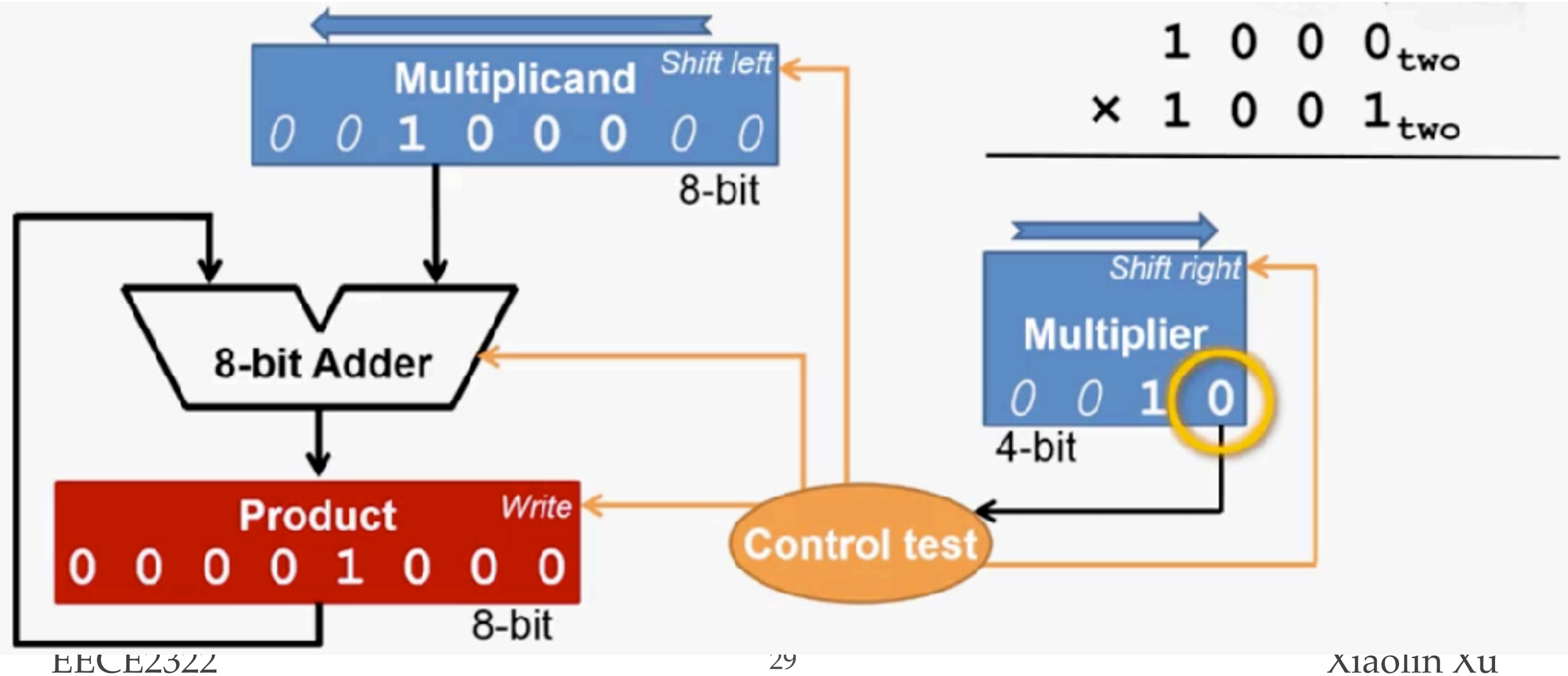


Step 1 of Round 3



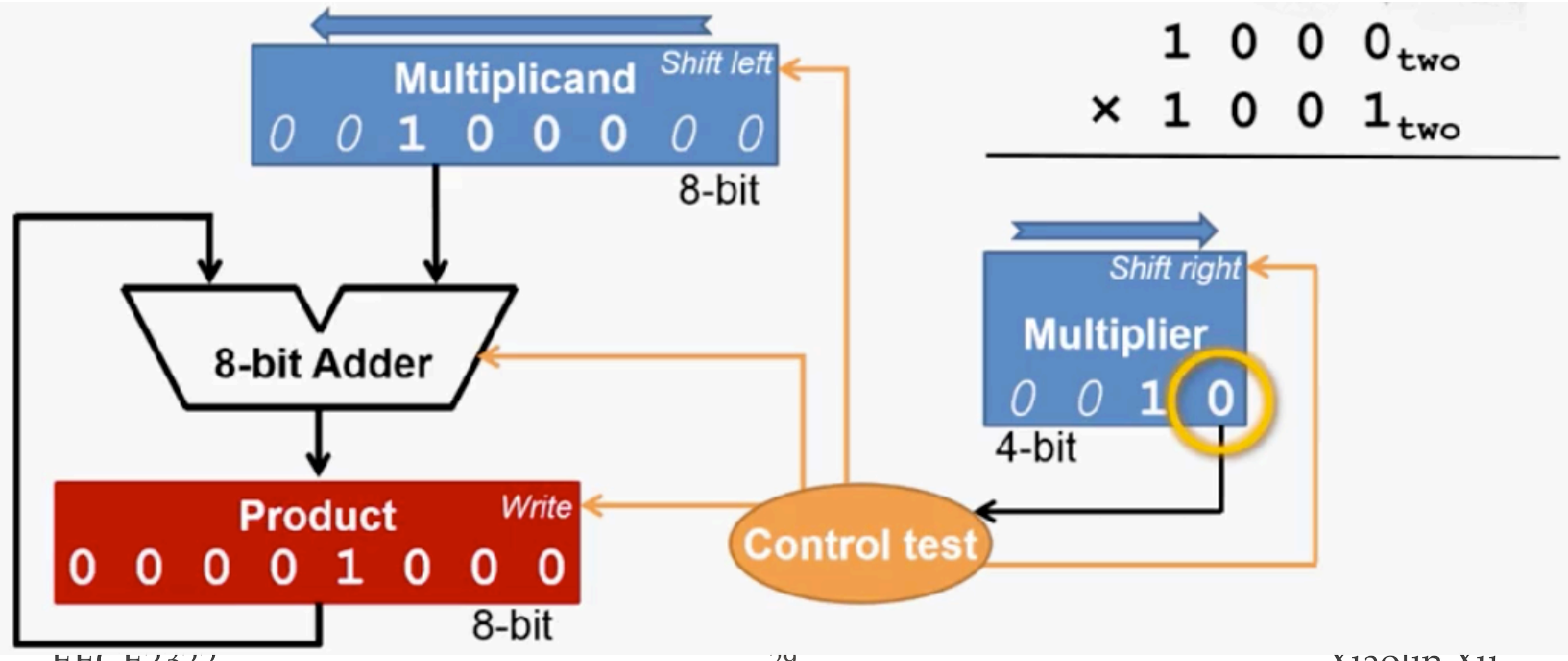
Step 1 of Round 3

- Checking the LSB of multiplier

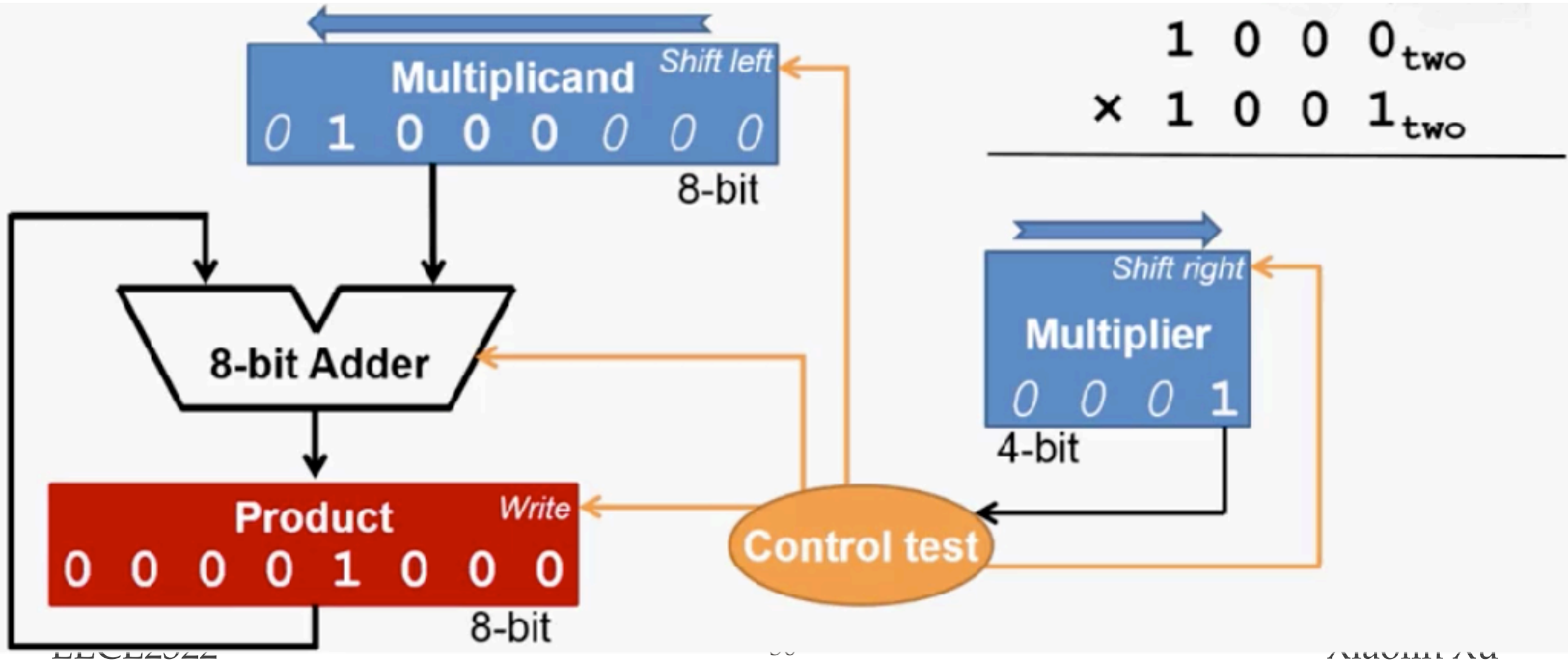


Step 1 of Round 3

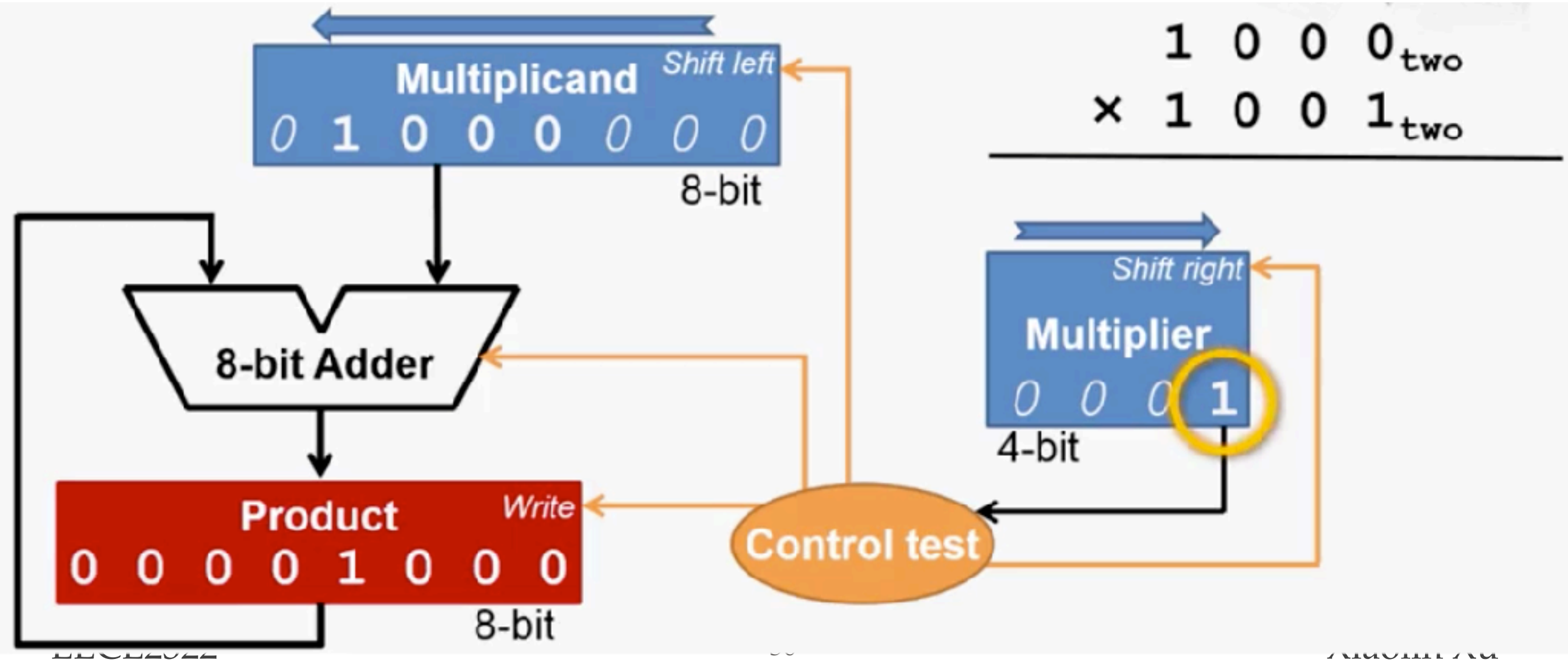
- Checking the LSB of multiplier
- Repeat in mind what happens for each step of round 3



Step 1 of Round 4

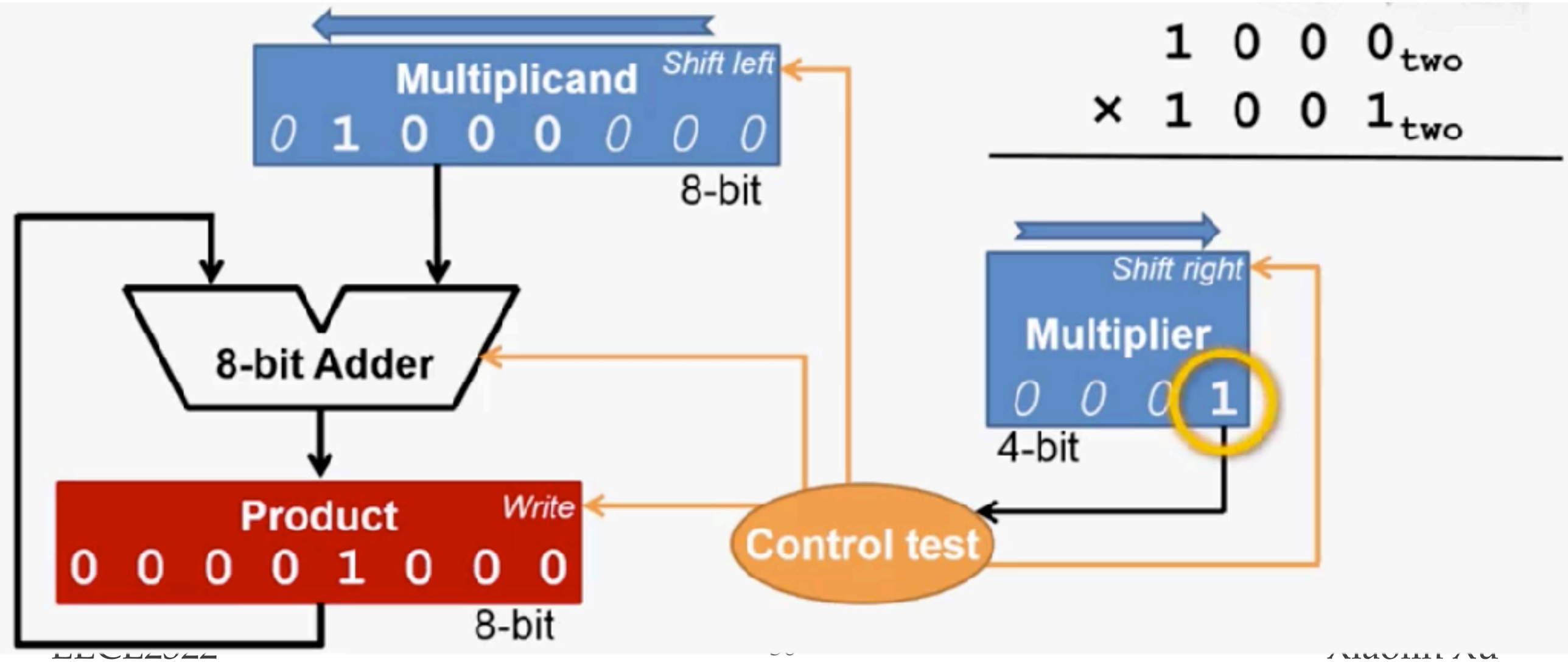


Step 1 of Round 4

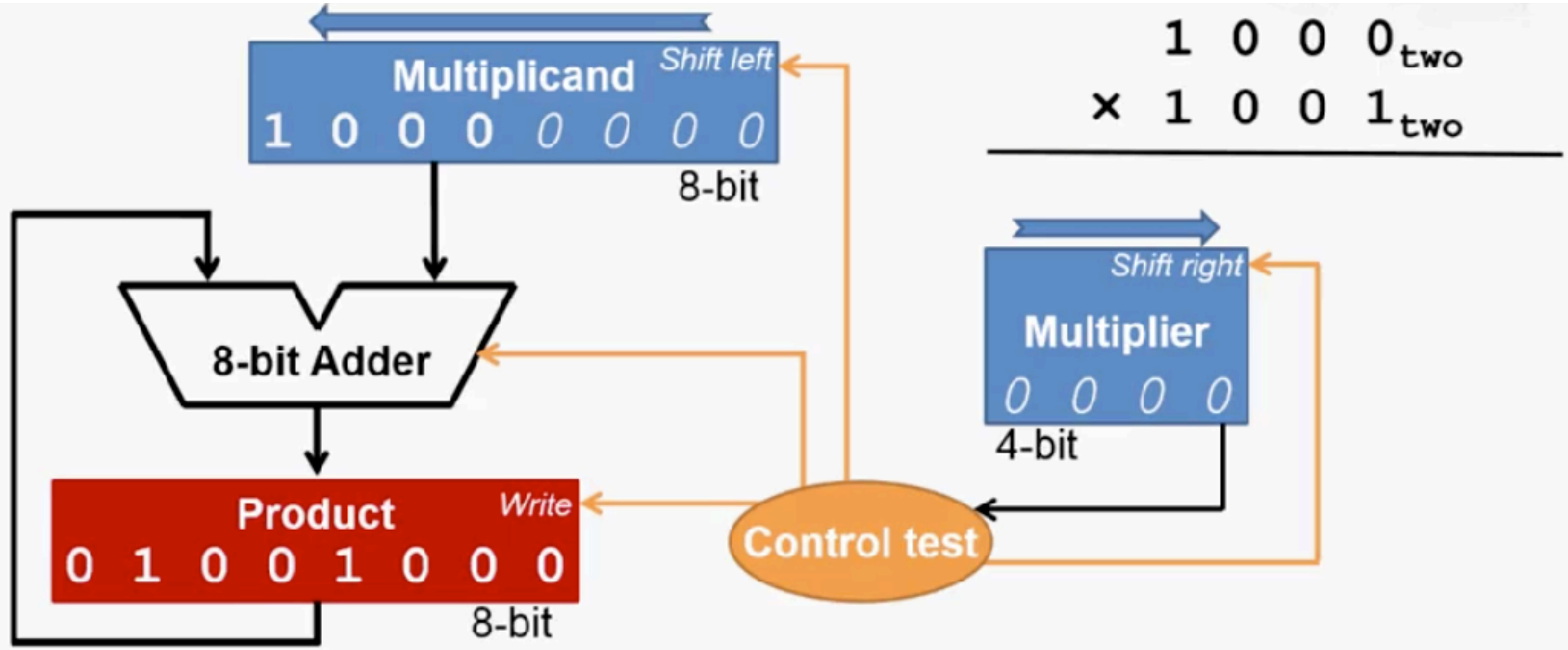


Step 1 of Round 4

- You should remember what is the step 1 now.

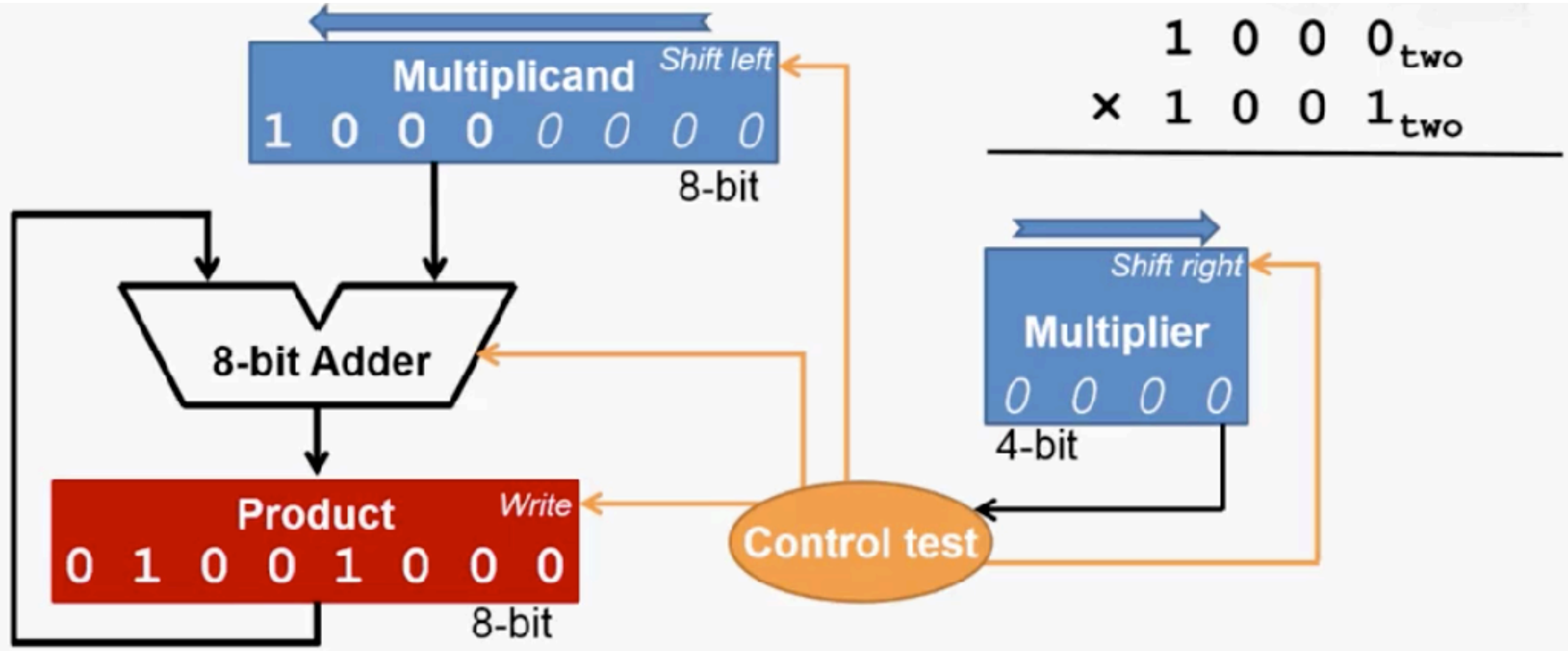


Step 4 of Round 4



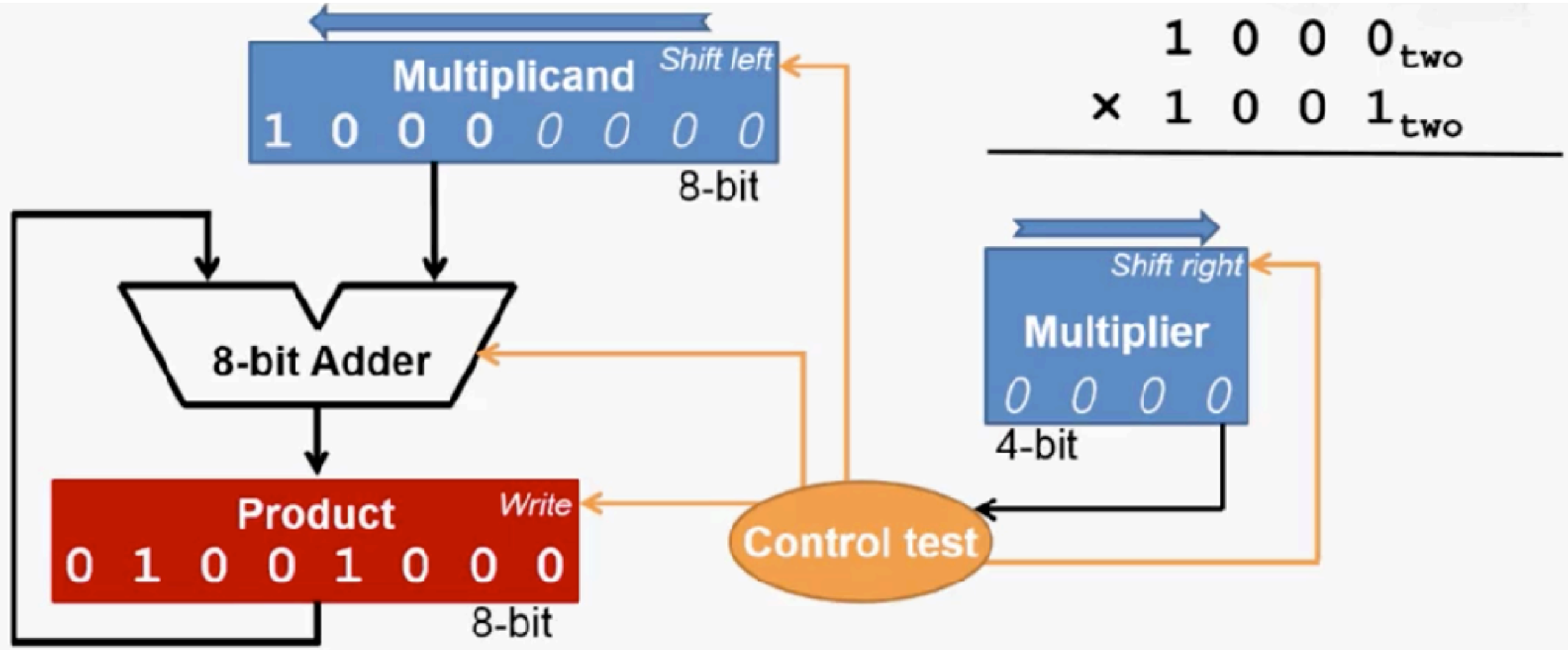
Step 4 of Round 4

- Multiplicand now has **1000***0000* (note the difference)



Step 4 of Round 4

- Multiplicand now has **1000***0000* (note the difference)
- All rounds done! However, hardware does not know it!

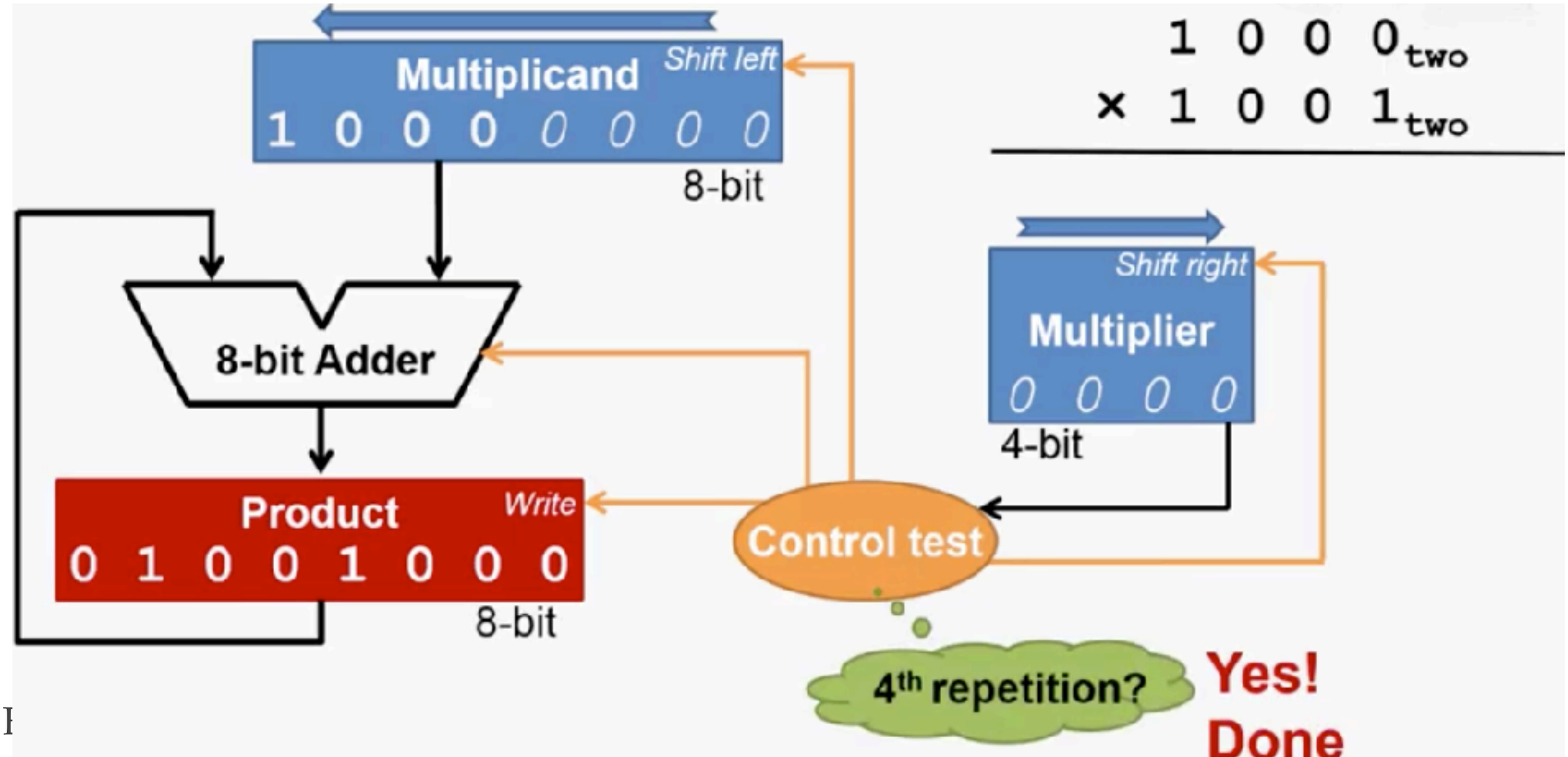


Step 4 of Round 4

- Multiplicand now has **1000***0000* (note the difference)
- All rounds done! However, hardware does not know it!

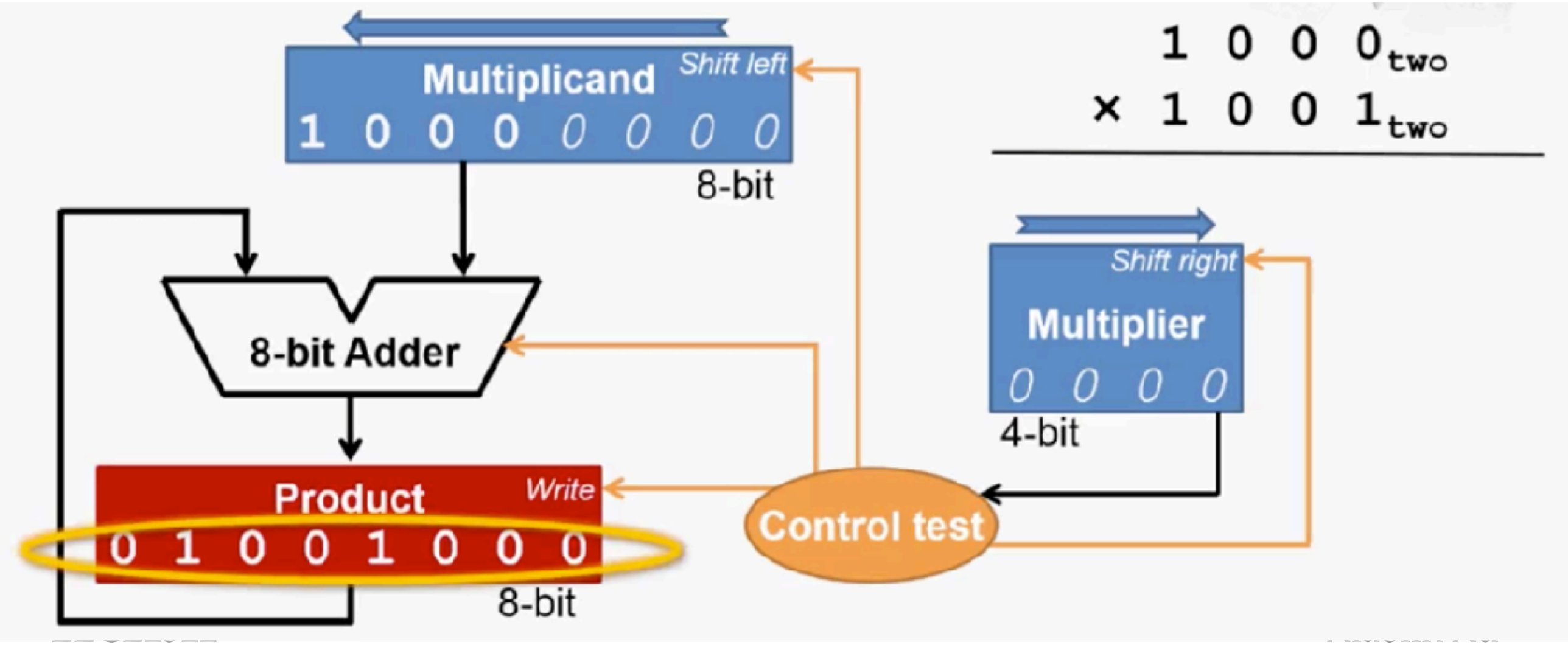
Step 4 of Round 4

- Multiplicand now has **1000***0000* (note the difference)
- All rounds done! However, hardware does not know it!



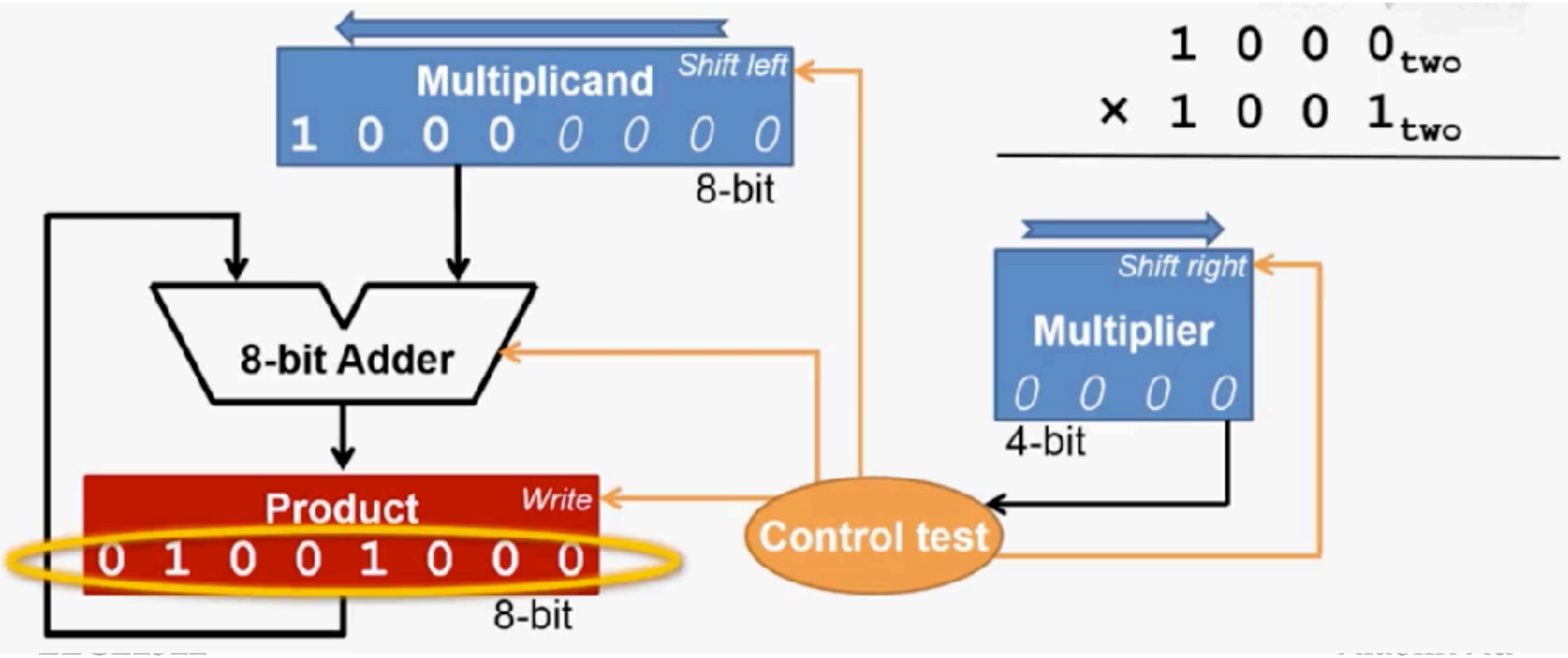
I

Results from All the 4 Rounds



Results from All the 4 Rounds

- 01001000, correct!



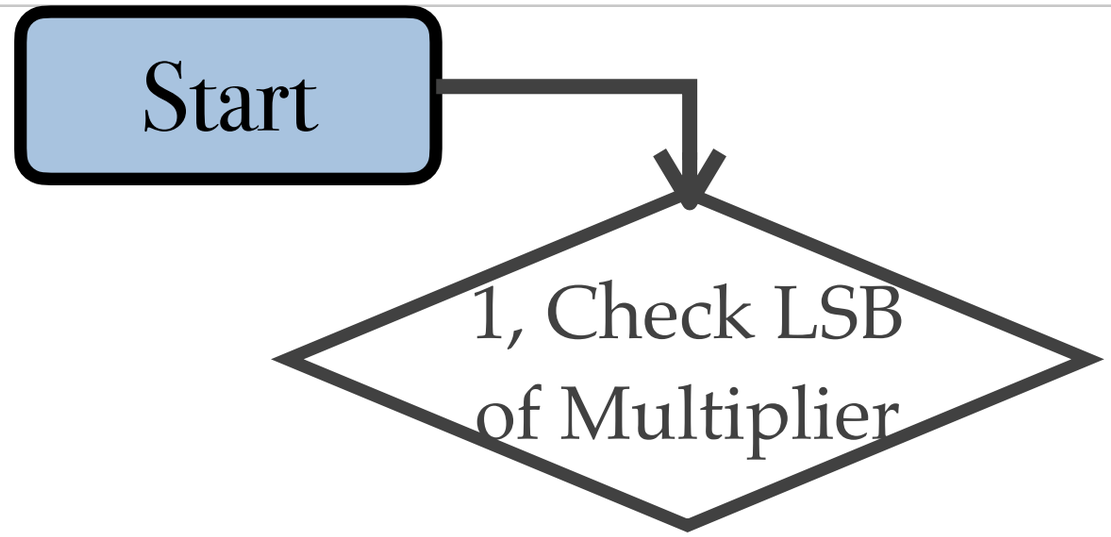
Summary

- Very straightforward hardware implementation
  - Each round repeated the same operation
  - No manual change in between
    - Simply left/right shifting
    - Simply checking repetition
  - Adder does not need to worries about aligning the two inputs

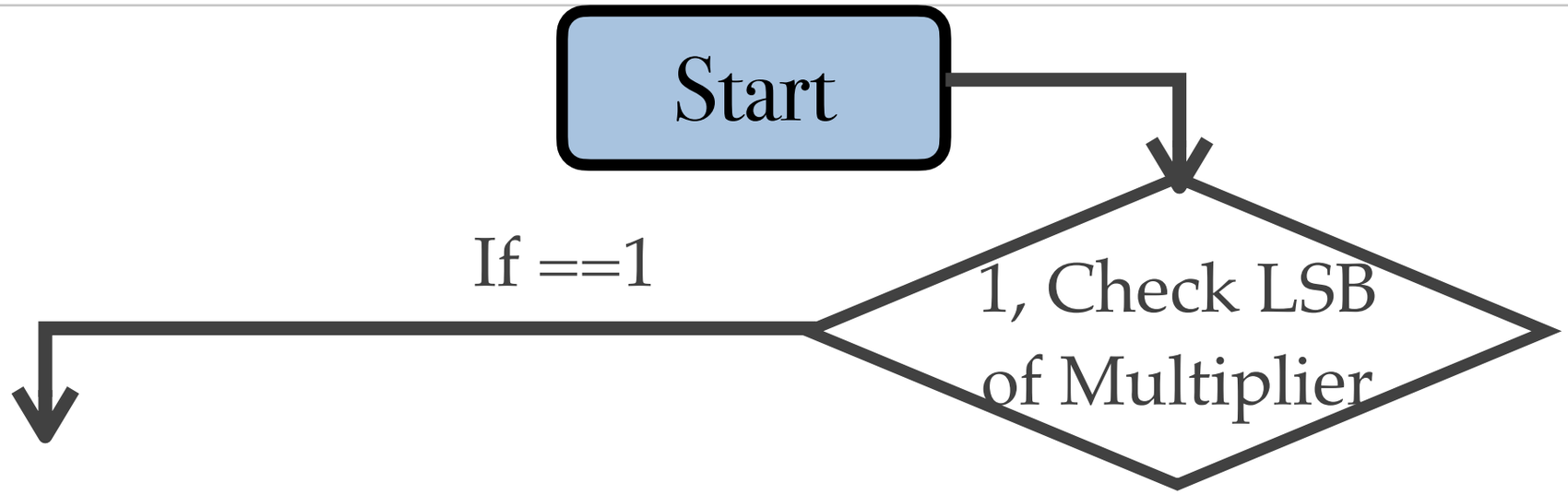
What about N-bit Multiplication?

What about N-bit Multiplication?

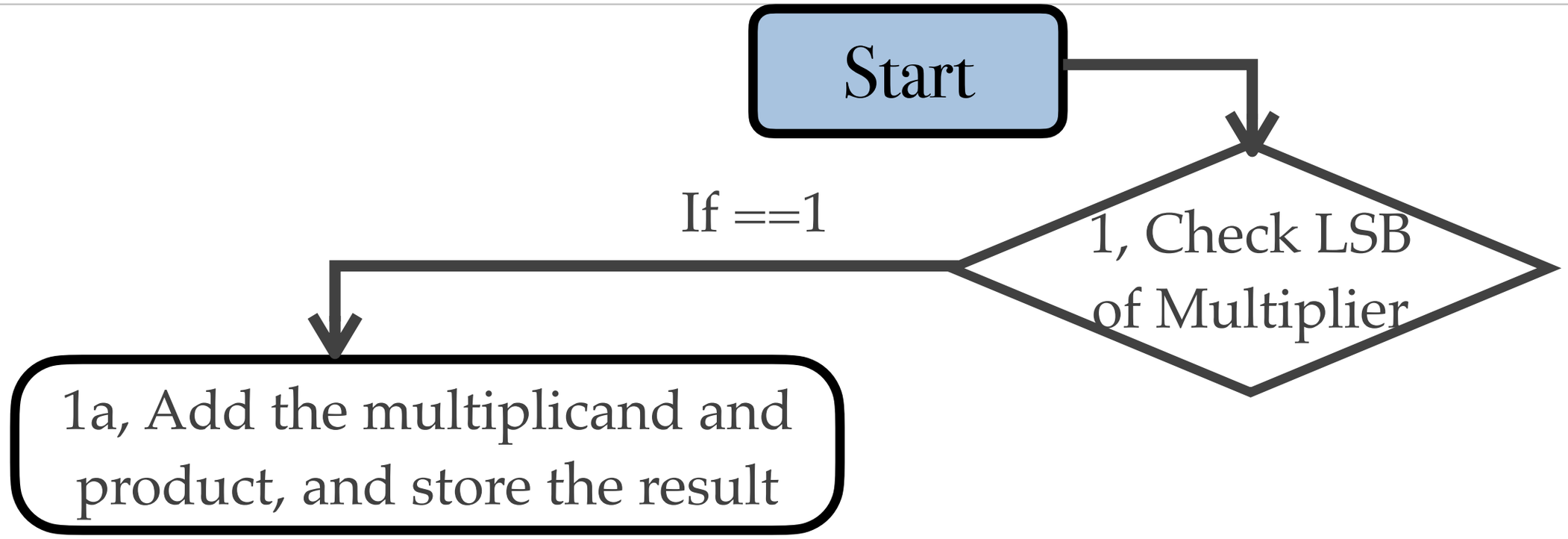
What about N-bit Multiplication?



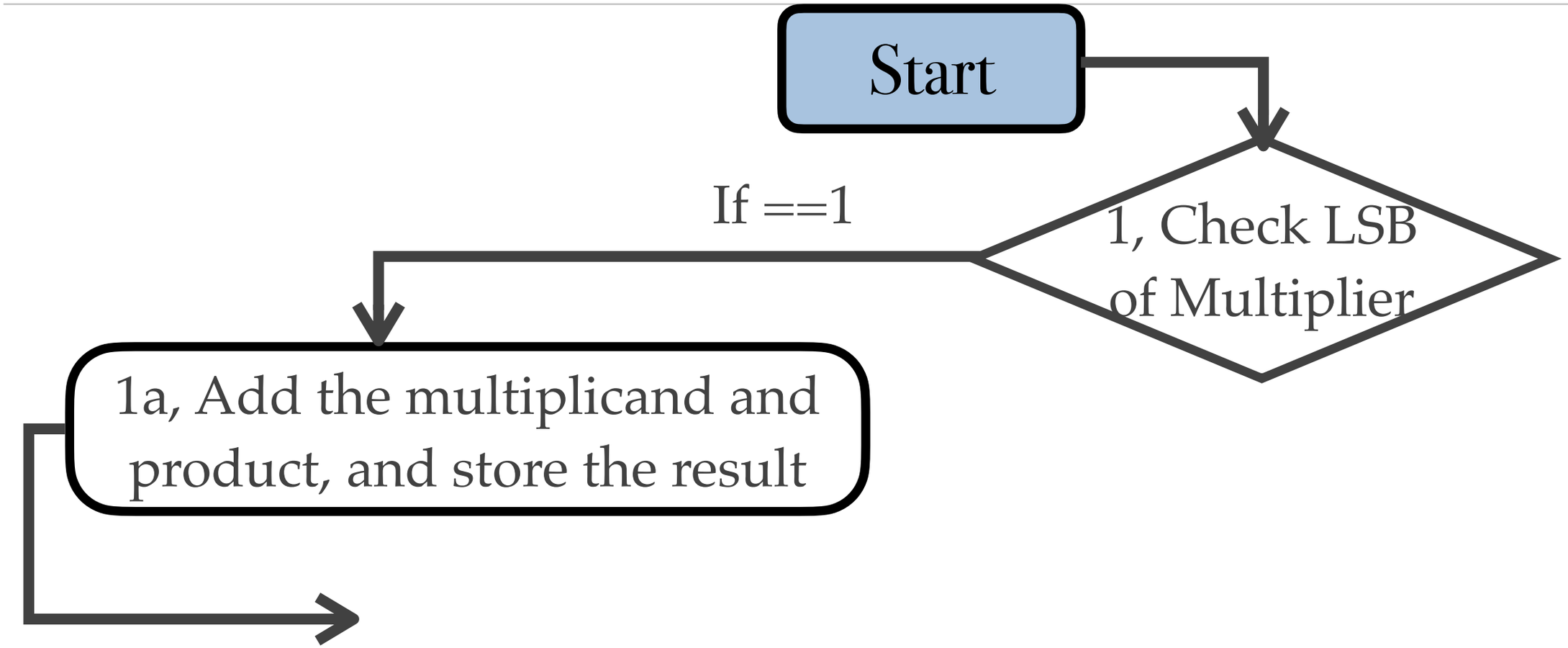
What about N-bit Multiplication?



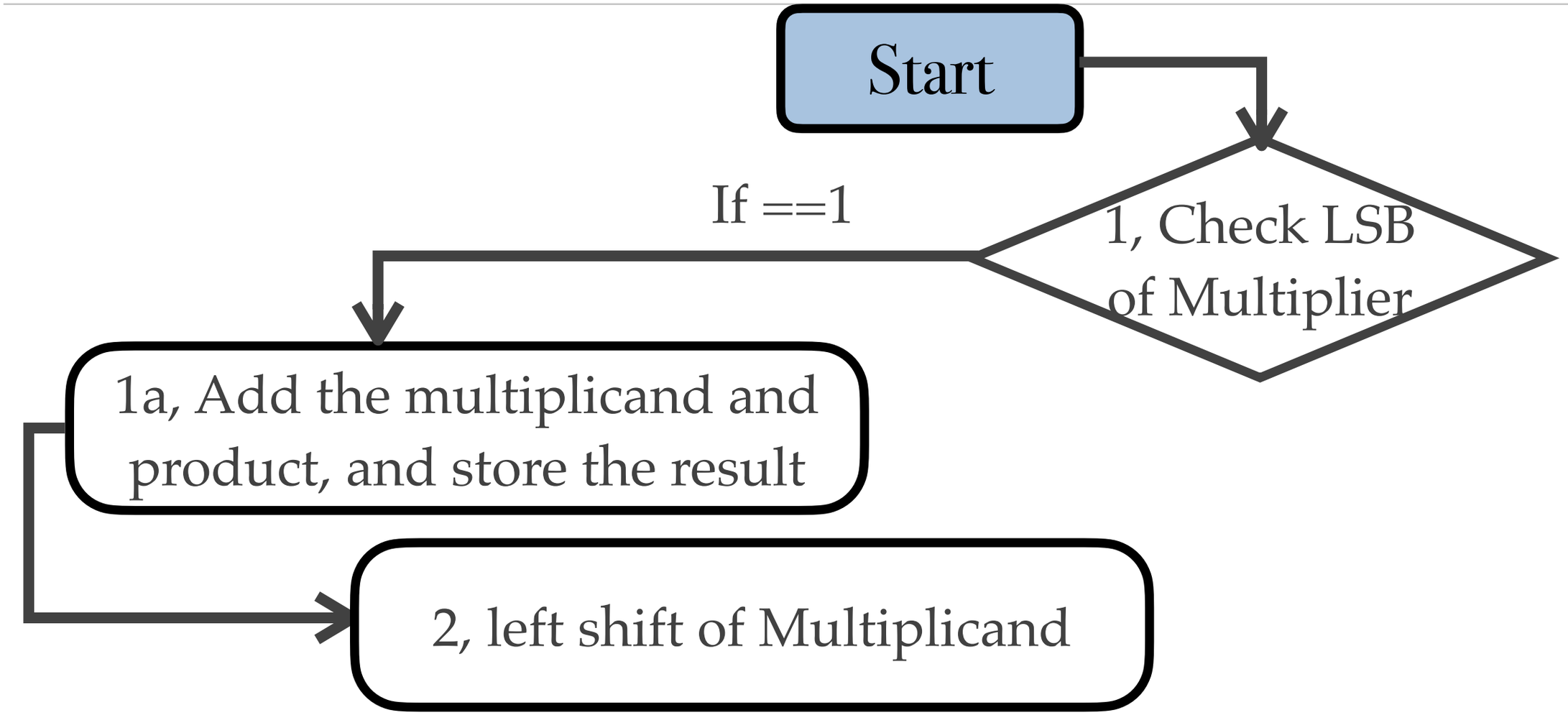
What about N-bit Multiplication?



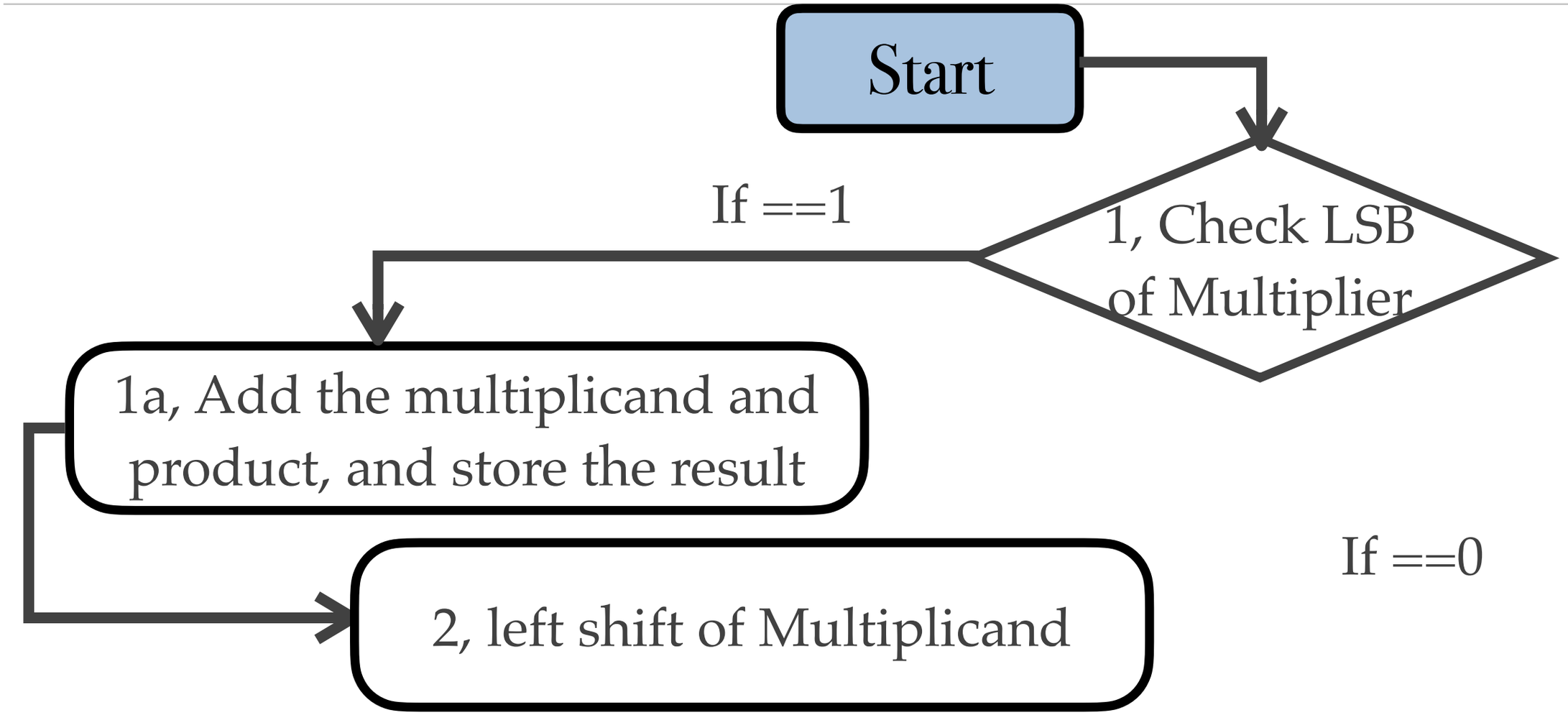
What about N-bit Multiplication?



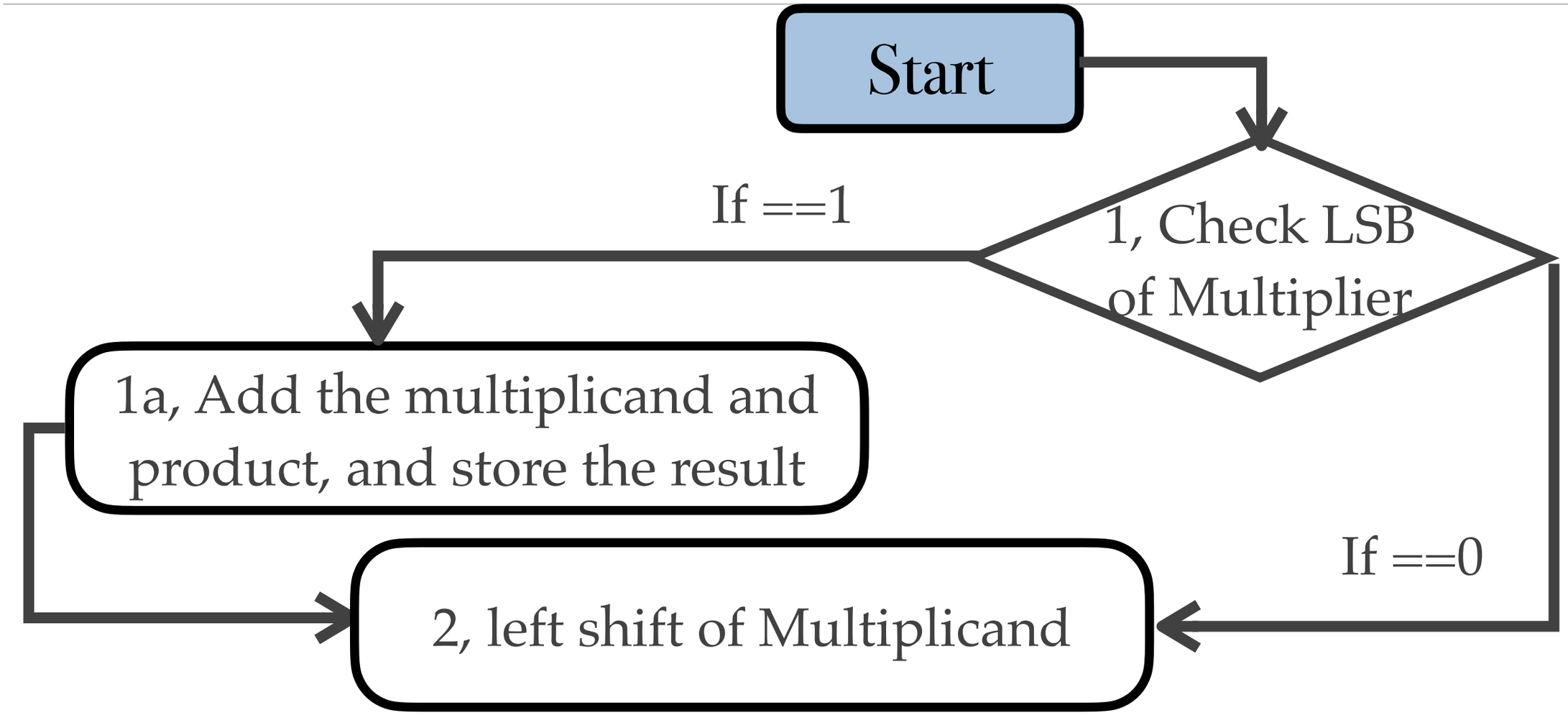
What about N-bit Multiplication?



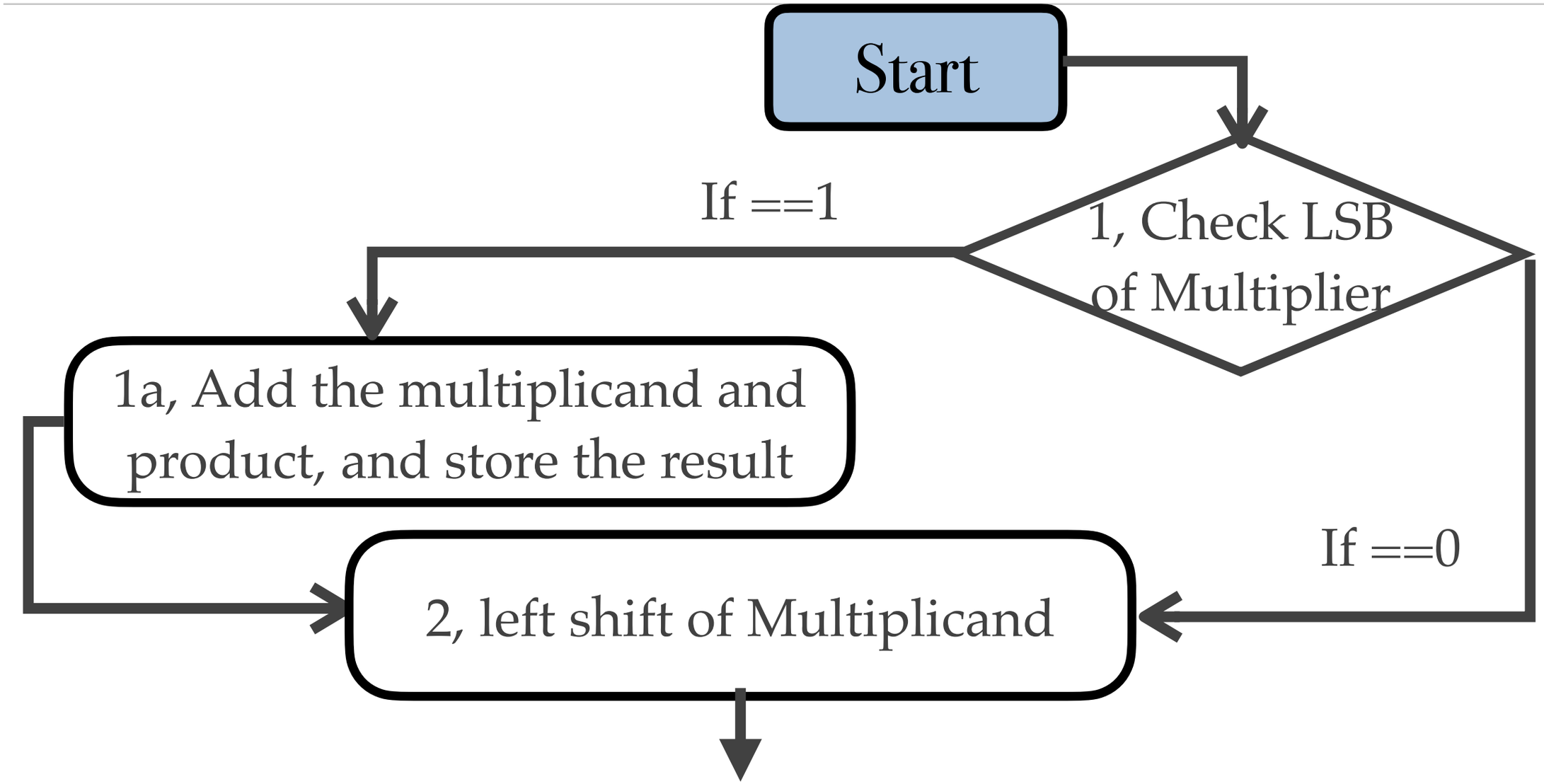
What about N-bit Multiplication?



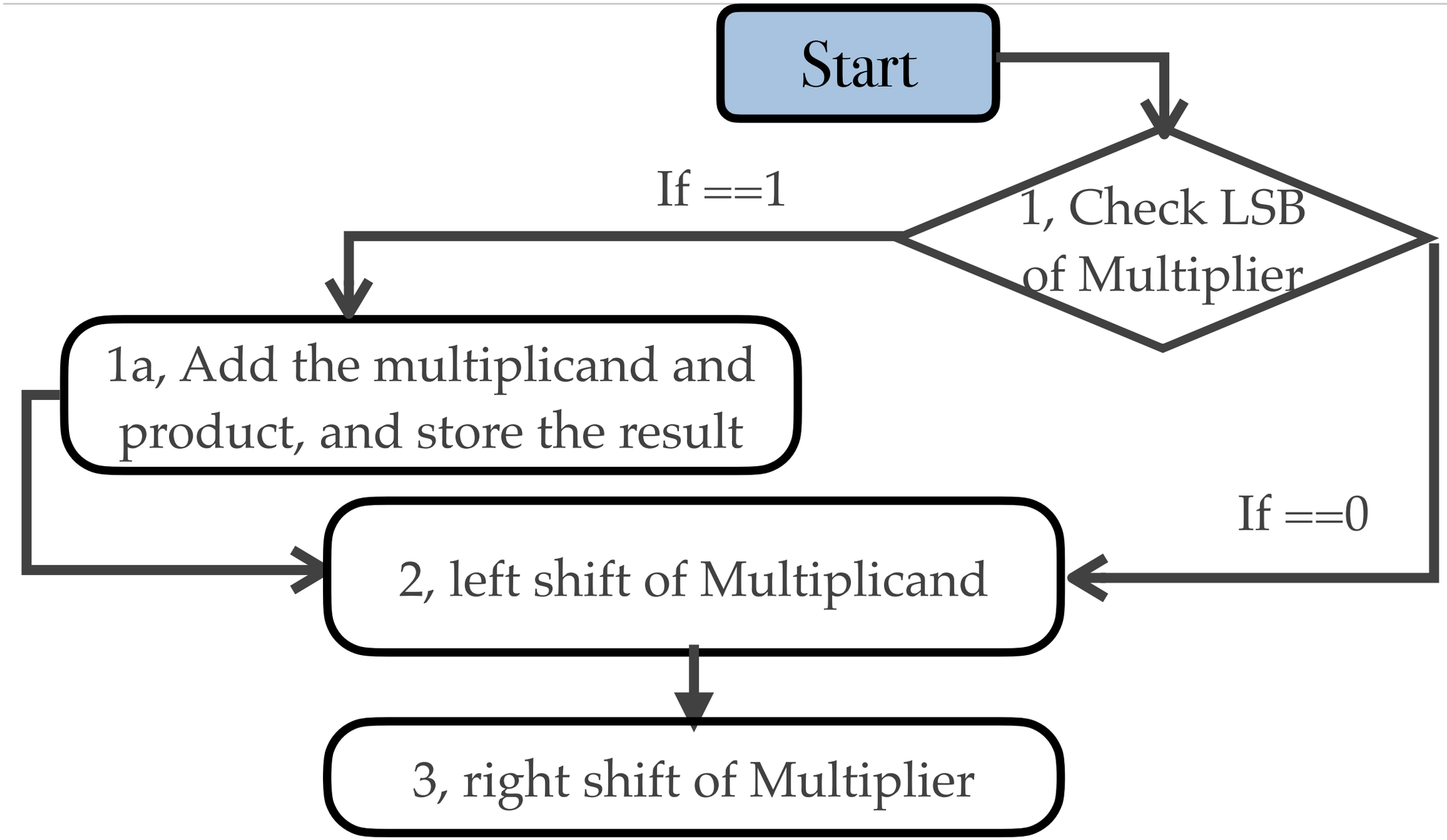
What about N-bit Multiplication?



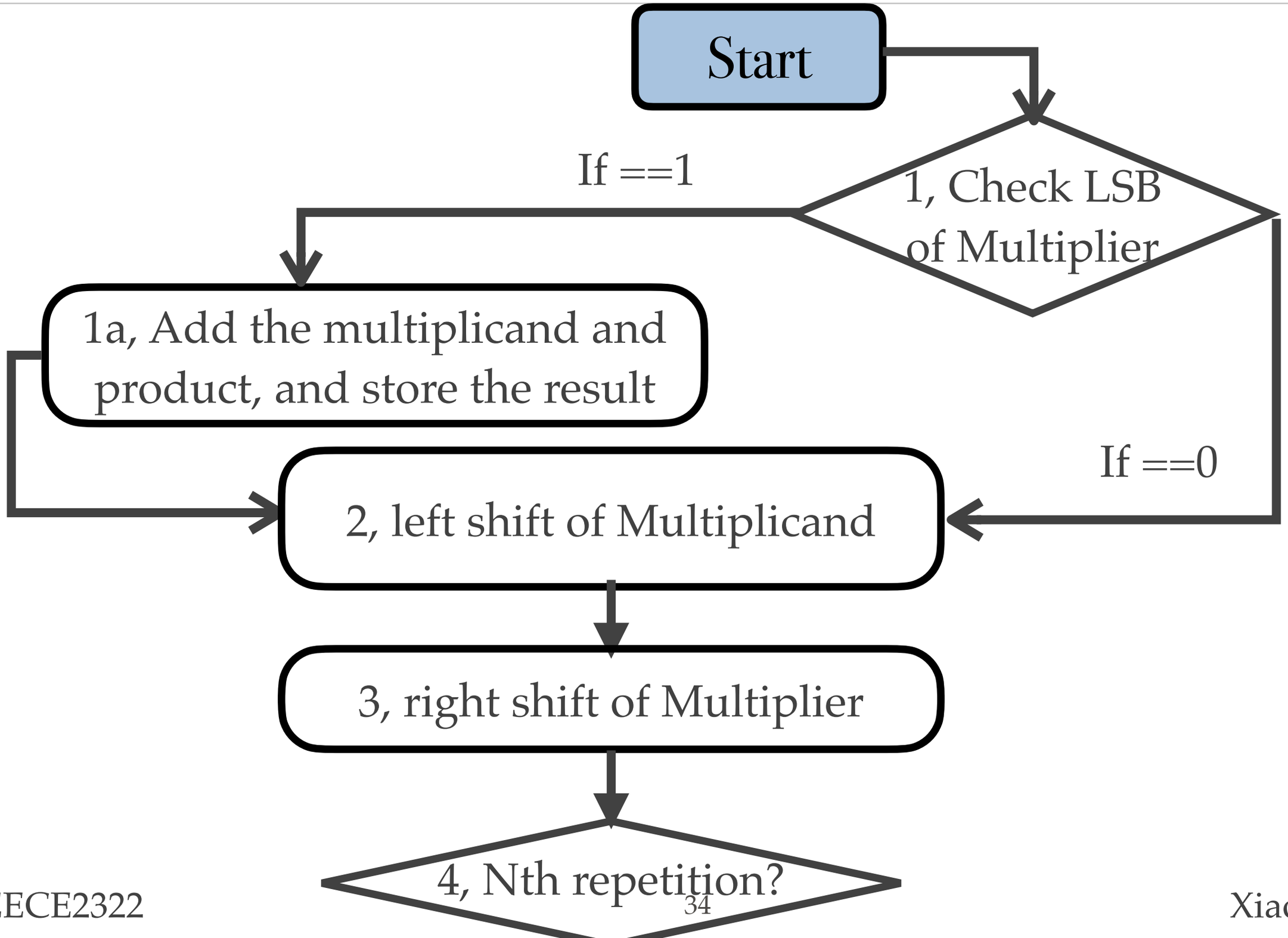
What about N-bit Multiplication?



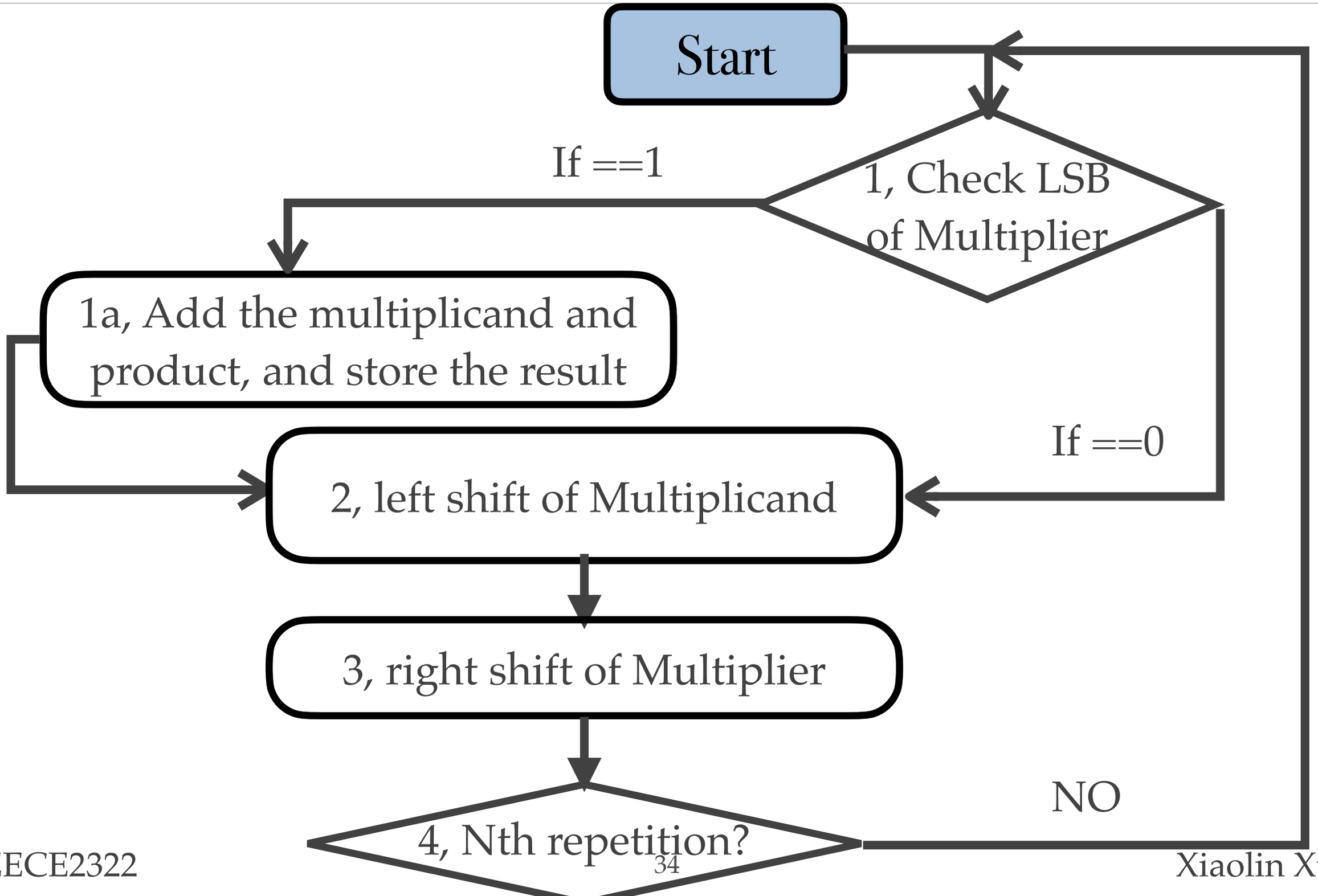
What about N-bit Multiplication?



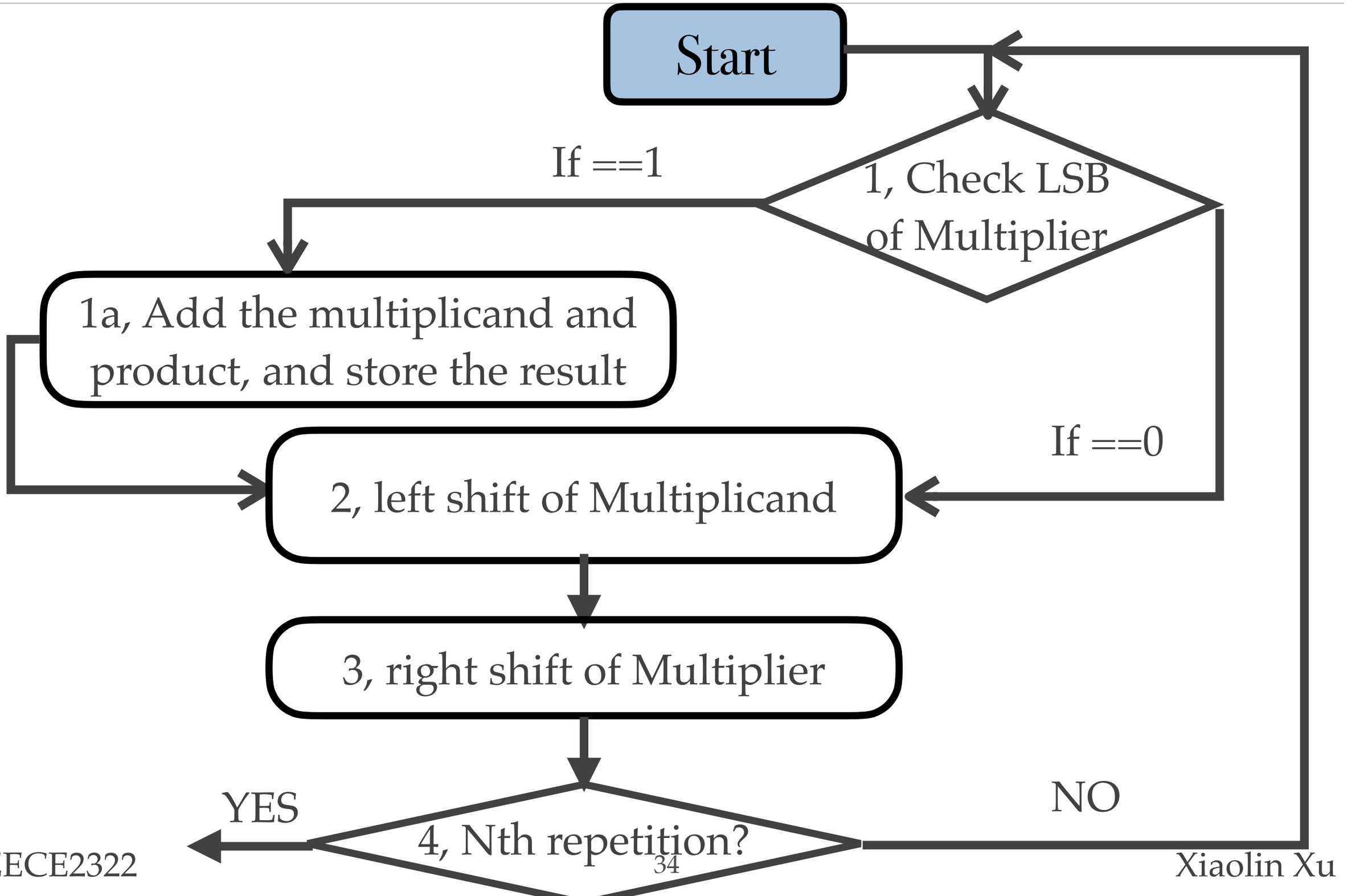
What about N-bit Multiplication?



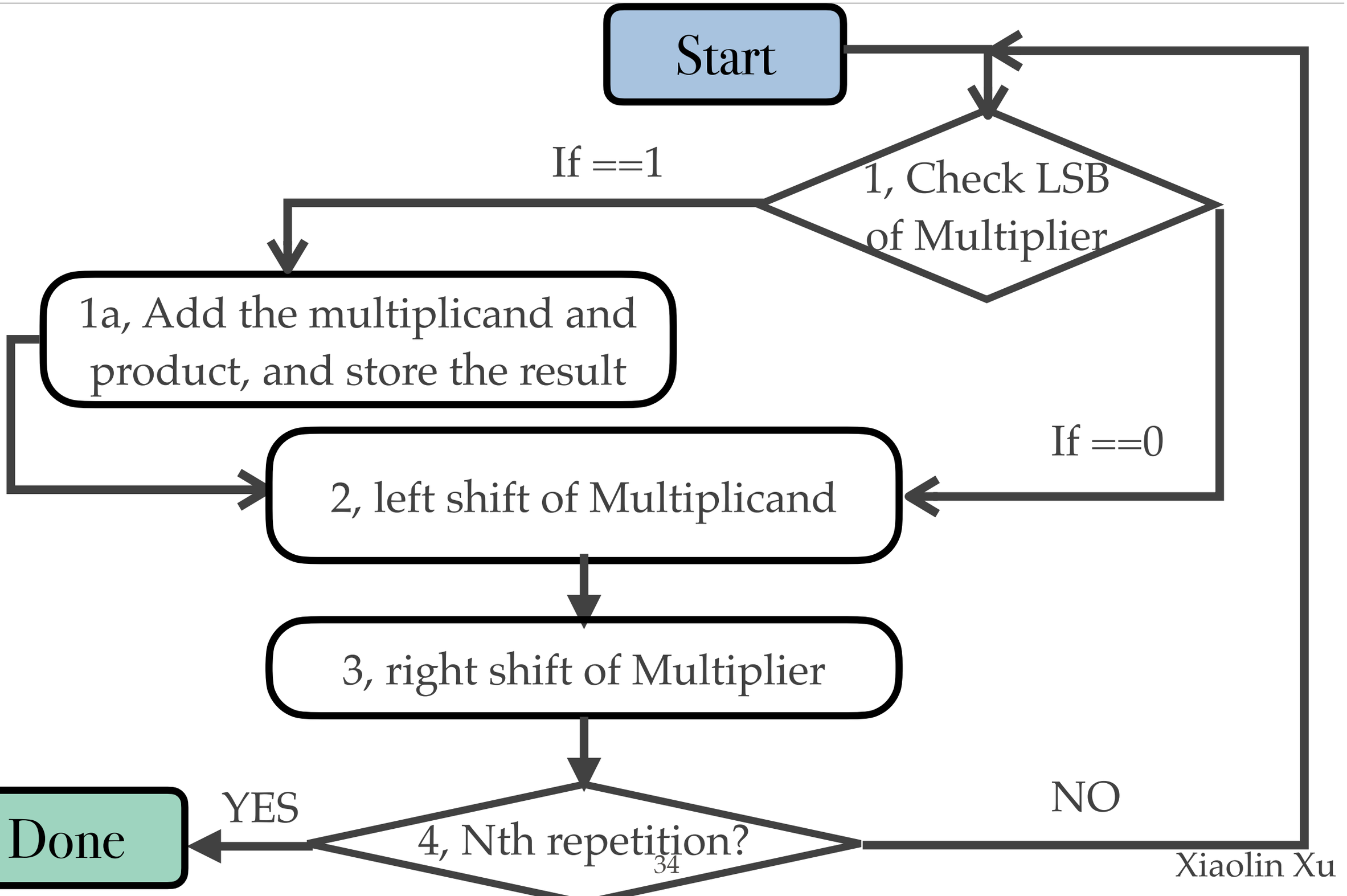
What about N-bit Multiplication?



What about N-bit Multiplication?



What about N-bit Multiplication?



EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 5_2: MIPS ISA

Xiaolin Xu
Department of ECE
Northeastern University