CPT_S 260 Intro to Computer Architecture
Lecture 6

Integer Multiplication
January 26, 2022

Ganapati Bhat
School of Electrical Engineering and Computer Science
Washington State University

Announcements

- **Reminder: Homework 1 due on Monday**
- **TA Office hours**
- **Will take place over Zoom**
  - I will post the links
- **Nephi Duff**
  - Weekends 12:30 pm and 2:00 pm
- **Jessica Cuevas**
  - Fridays 2 pm to 3 pm

Recap: Sign Extension

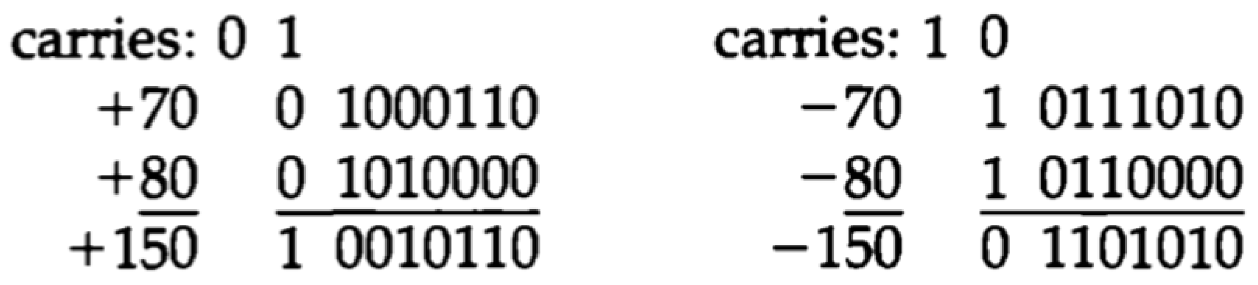
- Convert a binary number represented in N bits to a number represented with more than N bits
- For example, the computer has 32-bit memory, and the number is only 16 bits
- Sign extension allows us to convert
- Fill the lower order bits with the original number
- Copy the sign bit of the number to the left and fill the bits

Recap: Overflow

- The addition of two numbers with the same sign or subtraction of two numbers with different sign may cause overflow

- The condition wherein a result cannot be represented in allocated memory

- An overflow condition can be detected by observing the carry into the sign bit and carry out of the sign bit



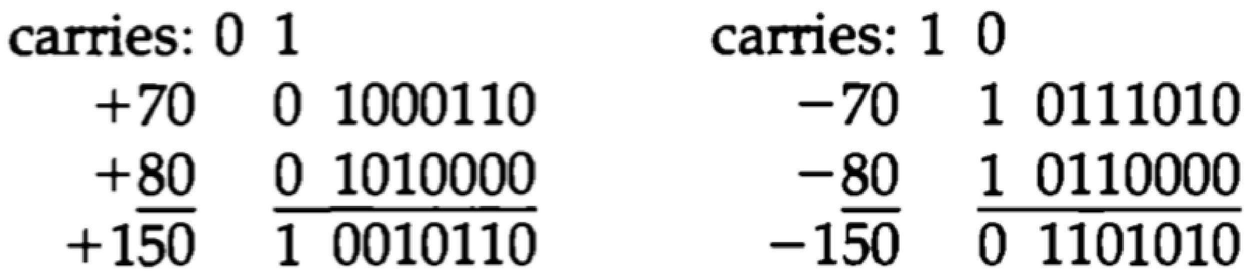
```
carries: 0  1                    carries: 1  0
     +70    0 1000110                 −70    1 0111010
     +80    0 1010000                 −80    1 0110000
    +150    1 0010110                −150    0 1101010
```

Recap: Conditions for Overflow

Operation	A	B	C
A + B = C	> 0	> 0	if (C <= 0)
	> 0	< 0	no overflow
	< 0	> 0	no overflow
	< 0	< 0	if (C >= 0)
A - B = C	> 0	> 0	no overflow
	> 0	< 0	if (C <= 0)
	< 0	> 0	if (C >= 0)
	< 0	< 0	no overflow

Overflow Detection in Binary

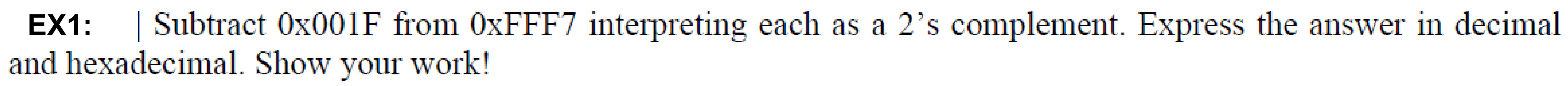
- **We need to compare three bits to check for overflow**
- **Needs a three-bit comparator**
- **Efficient method to check overflow**
- **Compare the carry in and carry out for the MSB**
  - If they are different then there is an overflow
  - If not, there is no overflow
- **Example**



```
carries: 0  1                carries: 1  0
     +70    0  1000110            −70    1  0111010
     +80    0  1010000            −80    1  0110000
    +150    1  0010110           −150    0  1101010
```

Another 2’s complement example

EX1:


| Subtract 0x001F from 0xFFF7 interpreting each as a 2’s complement. Express the answer in decimal and hexadecimal. Show your work!

EX2: Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate 185 – 122. Is there overflow, underflow, or neither?

Another 2’s complement example

[5 points] Subtract 0x001F from 0xFFF7 interpreting each as a 2’s complement. Express the answer in decimal and hexadecimal. Show your work!

0xFFF7 – 0x001F =
(1111 1111 1111 0111) – (0000 0000 0001 1111) =
(1111 1111 1111 0111) + (1111 1111 1110 0001) =
(1111 1111 1101 1000)

The answer in hexadecimal format is therefore 0xFFD8
To convert this number to decimal, we need to take a 2’s complement of the number because the sign bit is ‘1’ indicating that the result is negative. Thus,

0xFFD8 = (1111 1111 1101 1000) = - (0000 0000 0010 1000) = -40

Thus the answer in decimal format is ‘-40’.

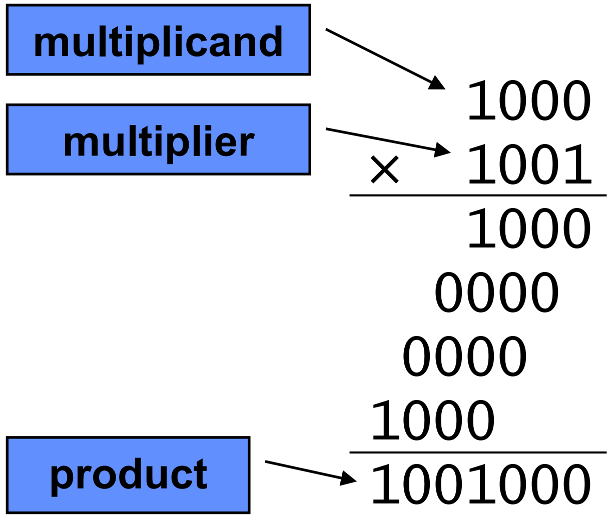
Multiplication

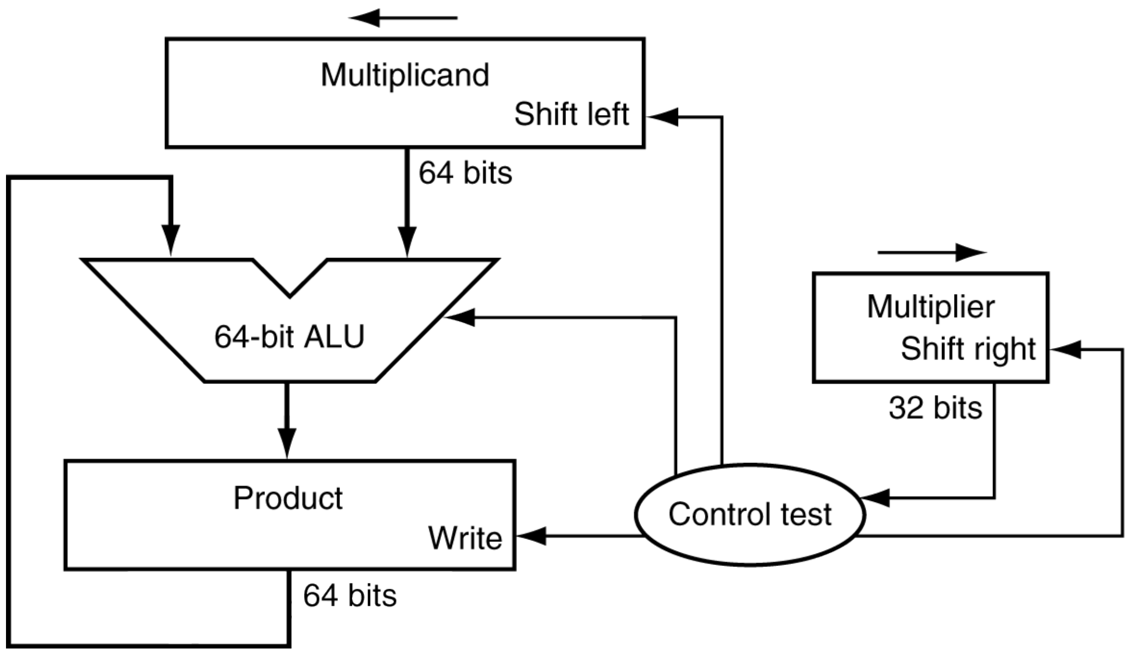
Base 10 –

$$\begin{array}{llr}
\text{Multiplicand} & & 1000_{\text{ten}} \\
\text{Multiplier} & \times & \underline{1001_{\text{ten}}} \\
 & & 1000\phantom{_{\text{ten}}} \\
 & & 0000\phantom{0_{\text{ten}}} \\
 & & 0000\phantom{00_{\text{ten}}} \\
 & & \underline{1000}\phantom{_{\text{ten}}} \\
\text{Product} & & 1001000_{\text{ten}}
\end{array}$$

An *n-bit multiplier* and *m-bit multiplicand* result in a product up to (n+m) bits.

Sequential Multiplier





Length of product is the sum of operand lengths

You will see ALUs with more details later!

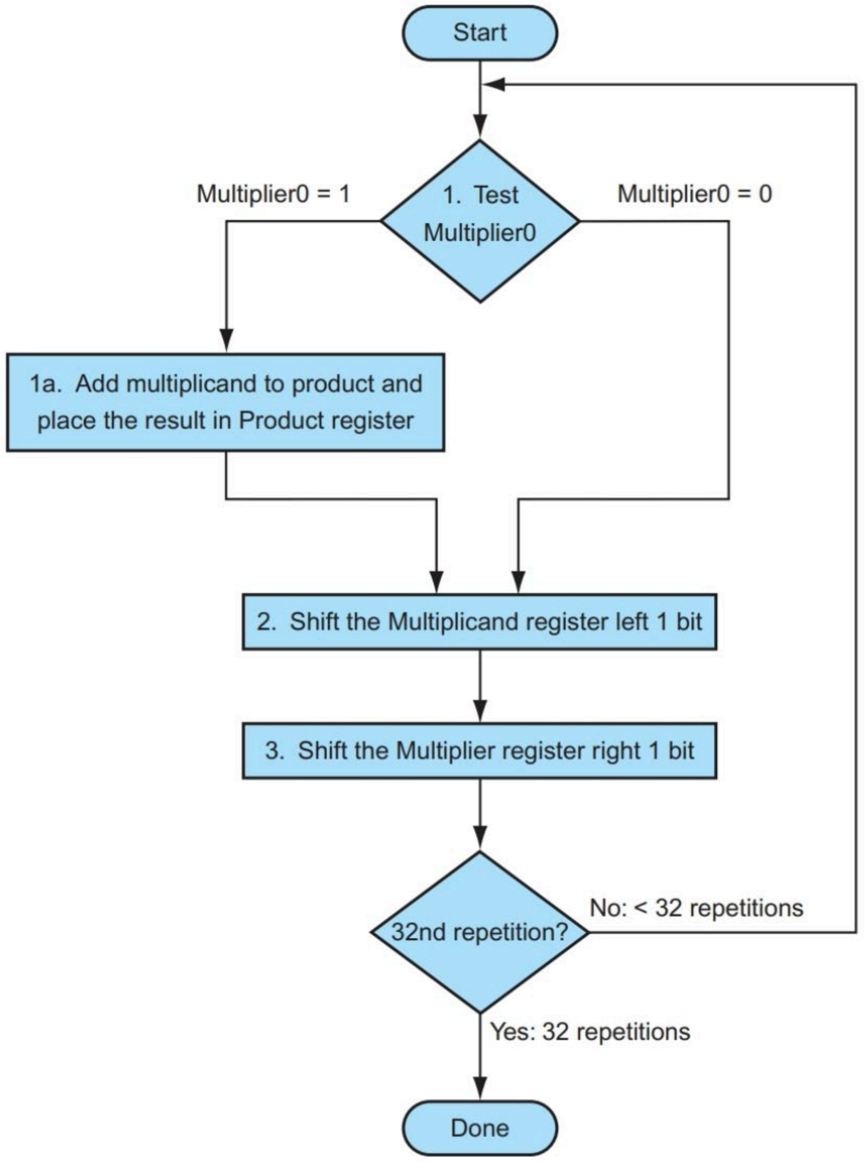
Sequential multiplier example

- **You are asked to multiply two binary numbers using the sequential multiplier shown before.**
- **The two binary numbers to multiply are 1010 and 10010.**
  - Show the value of different registers in each clock cycle. You can use the following table to do so.
  - How many bits is needed to save the result?

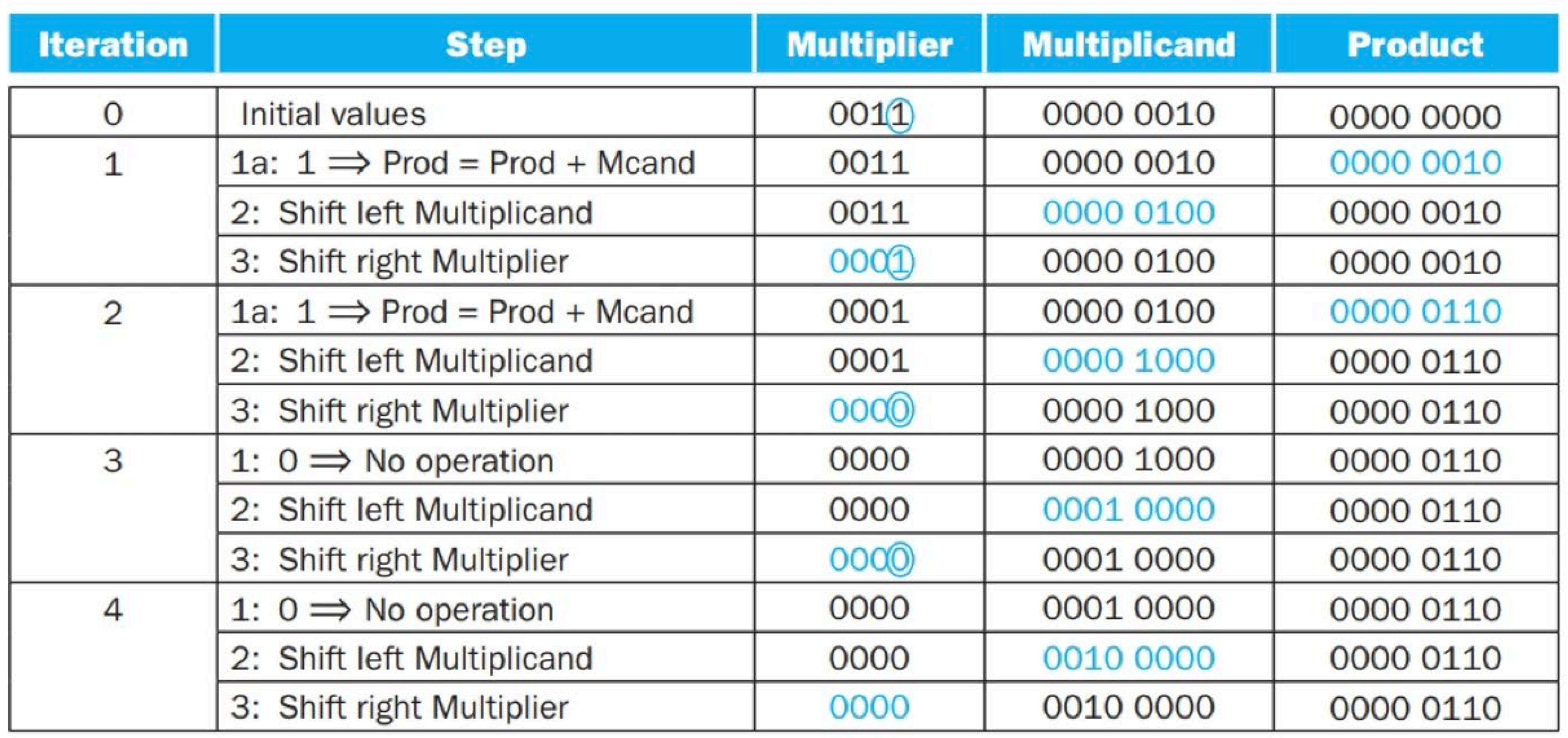
	Initial Values	Cycle 1	Cycle 2	Cycle 3	Cycle 4	Cycle 5
Product	000000000	000000000	000010100	000010100	000010100	010110100
Multiplicand	000001010	000010100	000101000	001010000	010100000	101000000
Multiplier	10010	01001	00100	00010	00001	00000

In each cycle, first look at 'Multiplier' in previous cycle. If LSB was '1' then add 'Multiplicand' to 'Product'. Then shift right 'Multiplier' and shift left 'Multiplicand' before going to the next cycle. For example, when in 'Cylcle1' we first look at LSB of 'Multiplier' in its initial value which is '0'. Thus we don't need to add 'Multiplicand' to 'Product'. But still we need to do both shift left and right before going to 'Cycle 2'.

Multiplication Algorithm

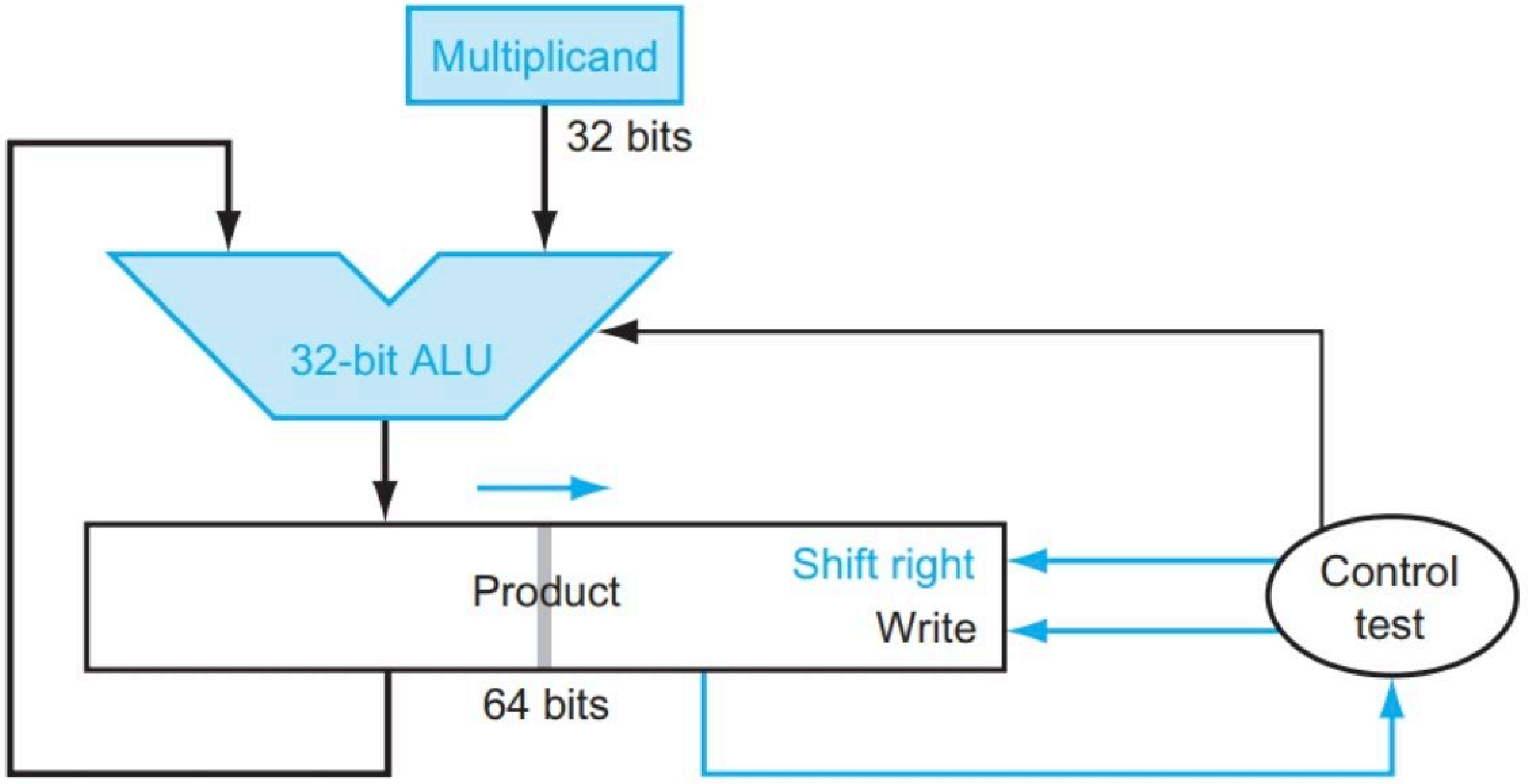

Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

Register Values per Clock Cycle



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

Parallel Architecture


Multiplicand
32 bits
32-bit ALU
Product
Shift right
Write
64 bits
Control test