Topics

- Integer Addition and Subtraction (3.1, 3.2)
- ➔Integer Multiplication (3.3)
- Integer Division (3.4)
- Floating point arithmetic (3.5)

Multiplication is vexation, division is as bad; The rule of three doth puzzle me, and practice drives me mad.

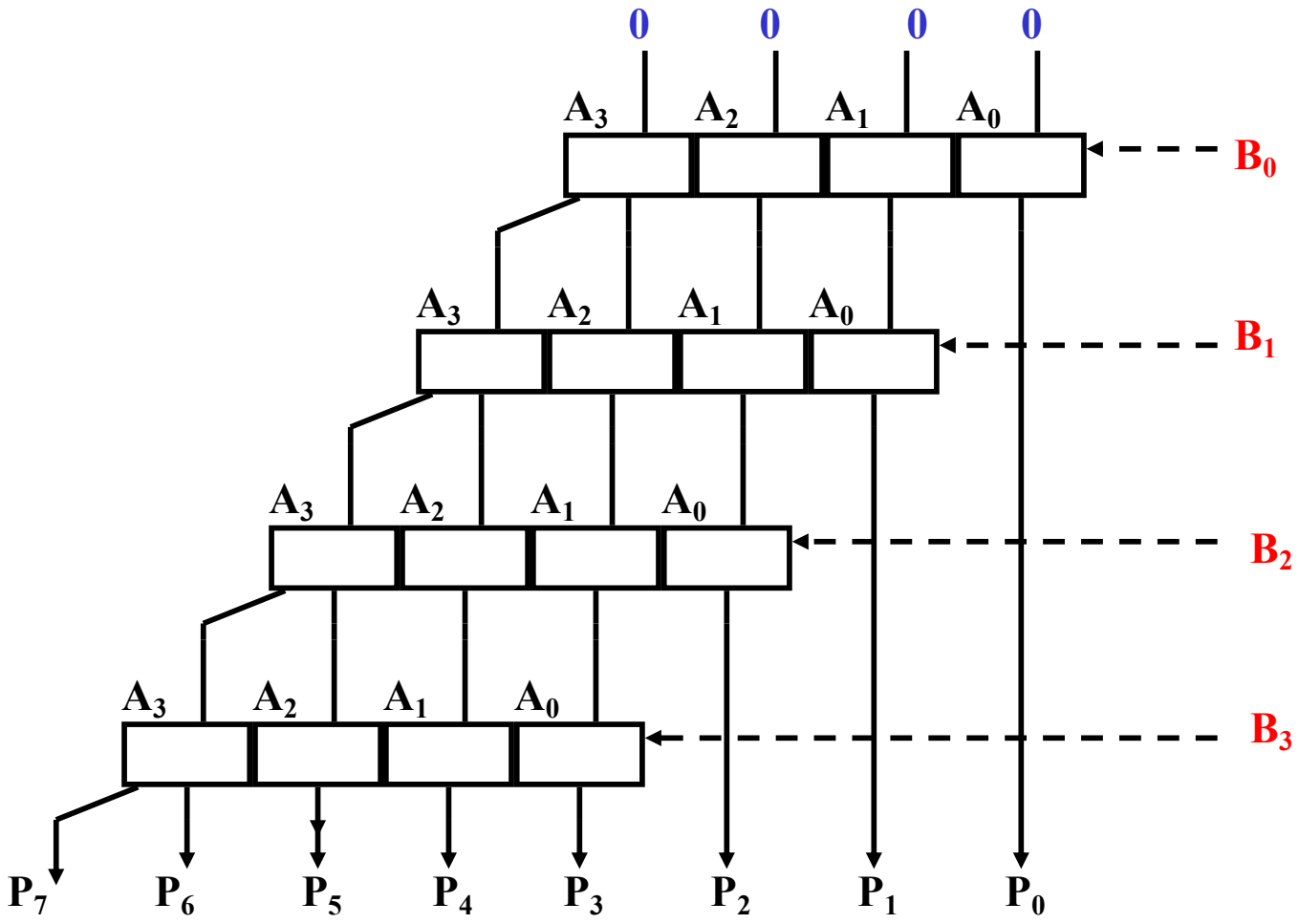
Anonymous, Elizabethan manuscript, 1570

Unsigned Multiplication

					0	1	0	1	1	0
					1	1	0	0	1	1
+					0	1	0	1	1	0
+				0	1	0	1	1	0	
+			0	0	0	0	0	0		
+		0	0	0	0	0	0			
+	0	1	0	1	1	0				
0	1	0	1	1	0					
1	0	0	0	1	1	0	0	0	1	0

- **`Multiplicand` is shifted left**
- **Check each bit of the `multiplier`**
  - shift right and then check LSB
- **Three registers:**
  - Multiplicand: 64 bits
  - Multiplier: 32 bits
  - Product: 64 bits

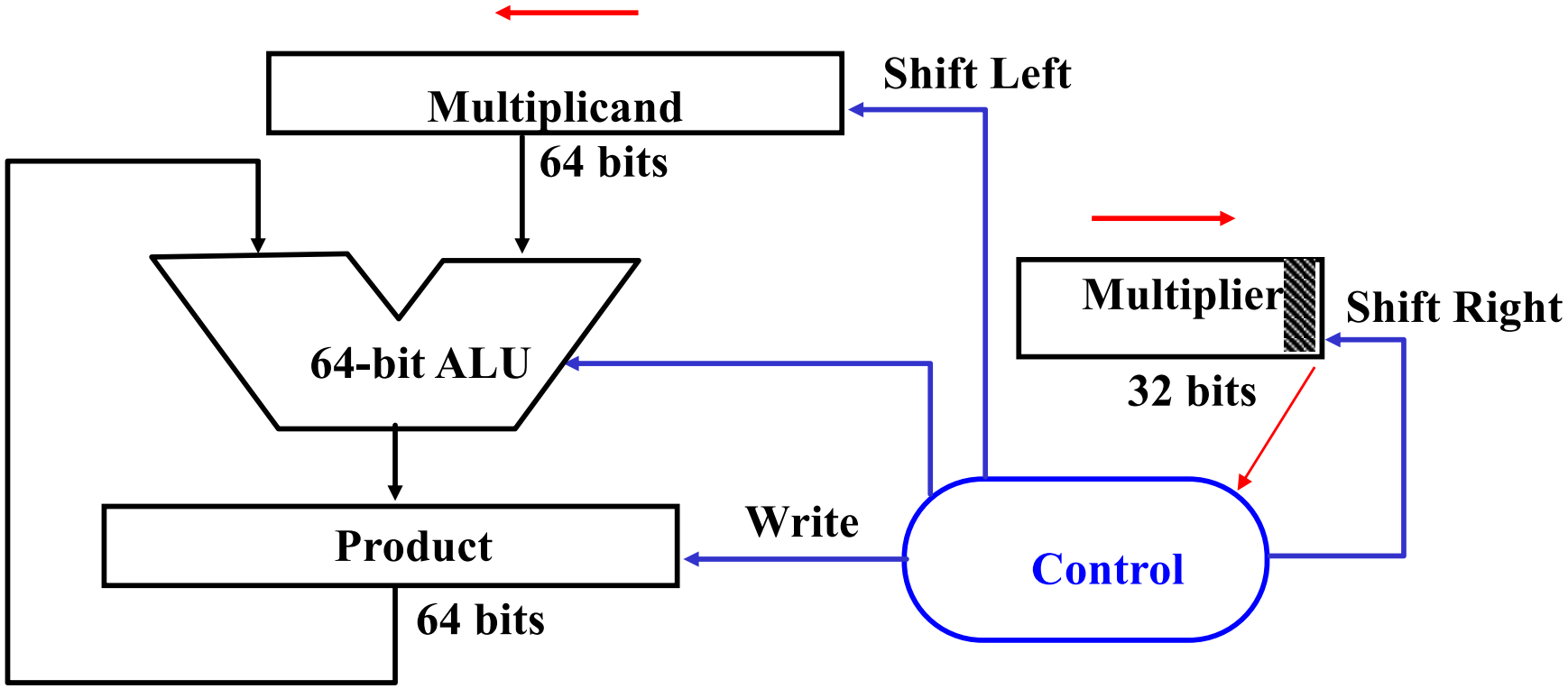
Unsigned Combinational Multiplier



- Stage i accumulates $A * 2^i$ if $B_i == 1$

Multiply Hardware (version 1)

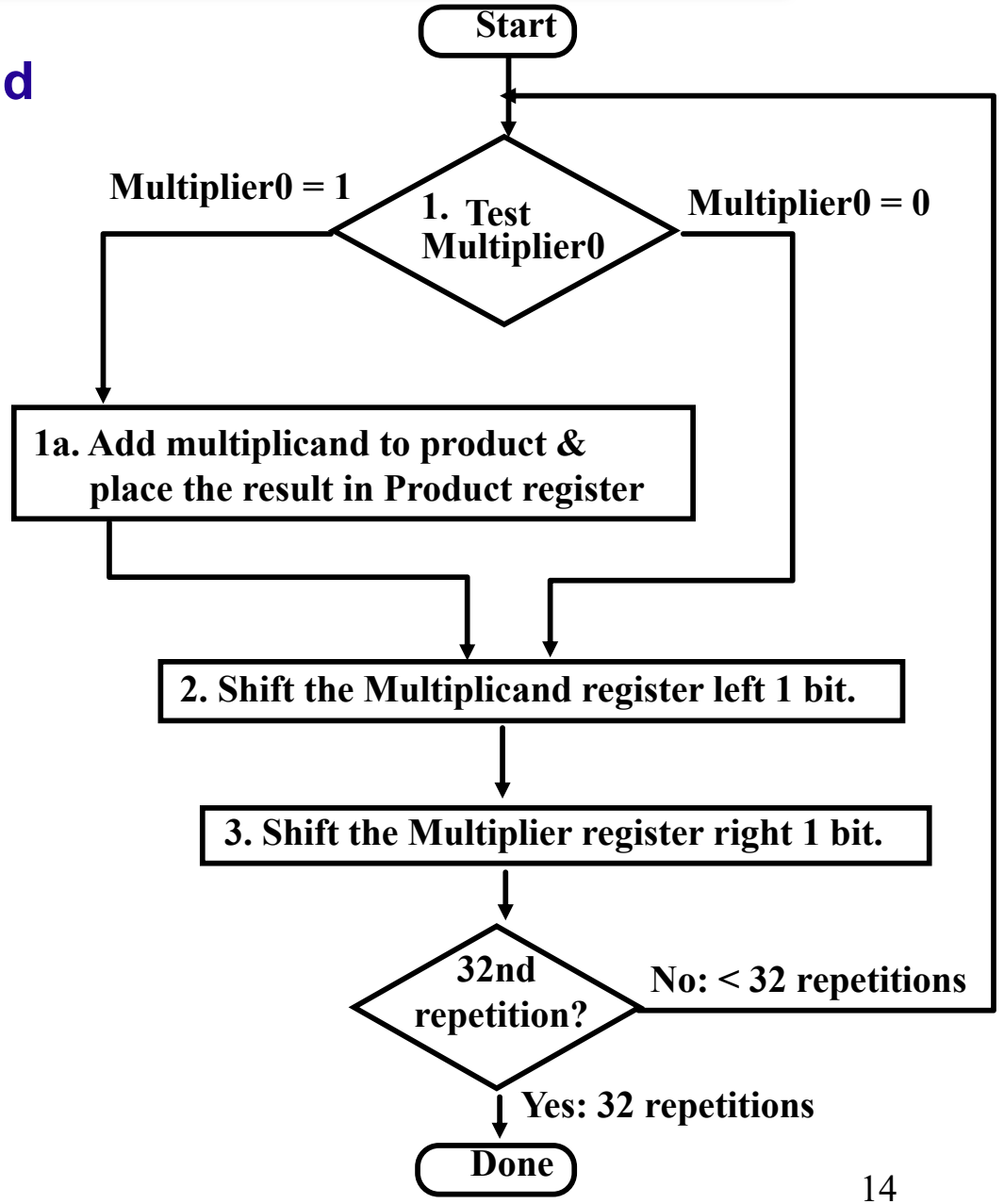
- Shift-add multiply
- 64-bit Multiplicand reg, 64-bit ALU, 64-bit Product reg, 32-bit multiplier reg



Multiply Algorithm Version 1

	Product	Multiplier	Multiplicand
0.	0000 0000	0011	0000 0010
1.	0000 0010	0001	0000 0100
2.	0000 0110	0000	0000 1000
3.	0000 0110	0000	0001 0000
4.	0000 0110	0000	0010 0000

0000 0110

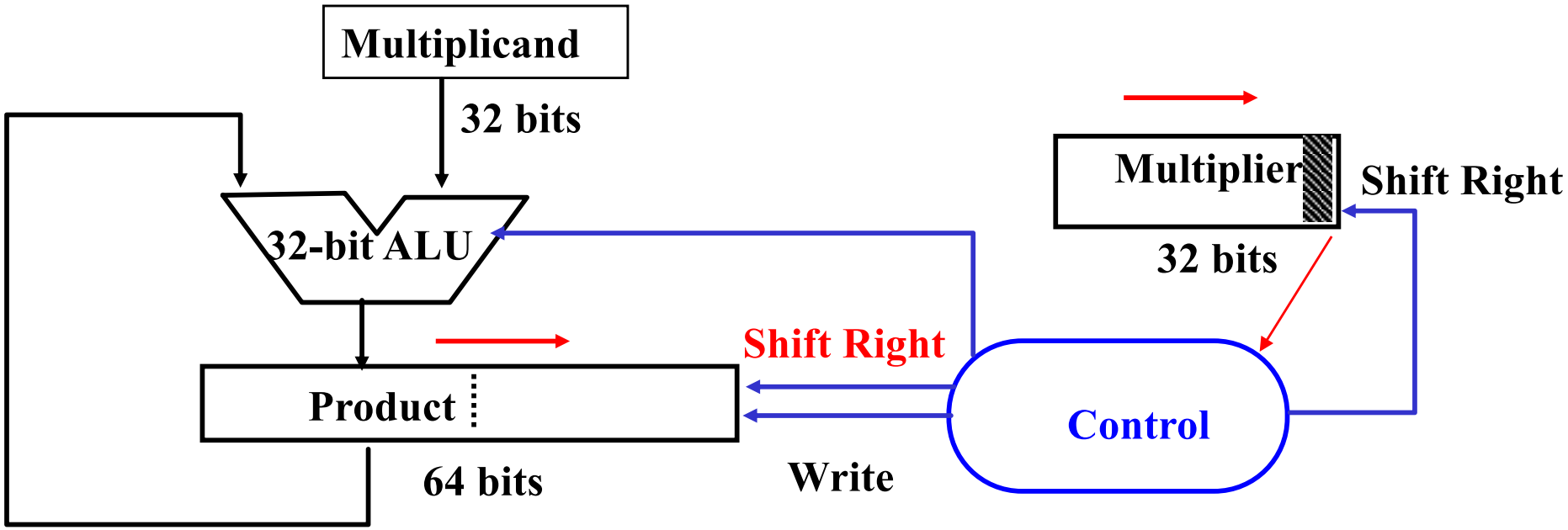


Observations on Multiply Version 1

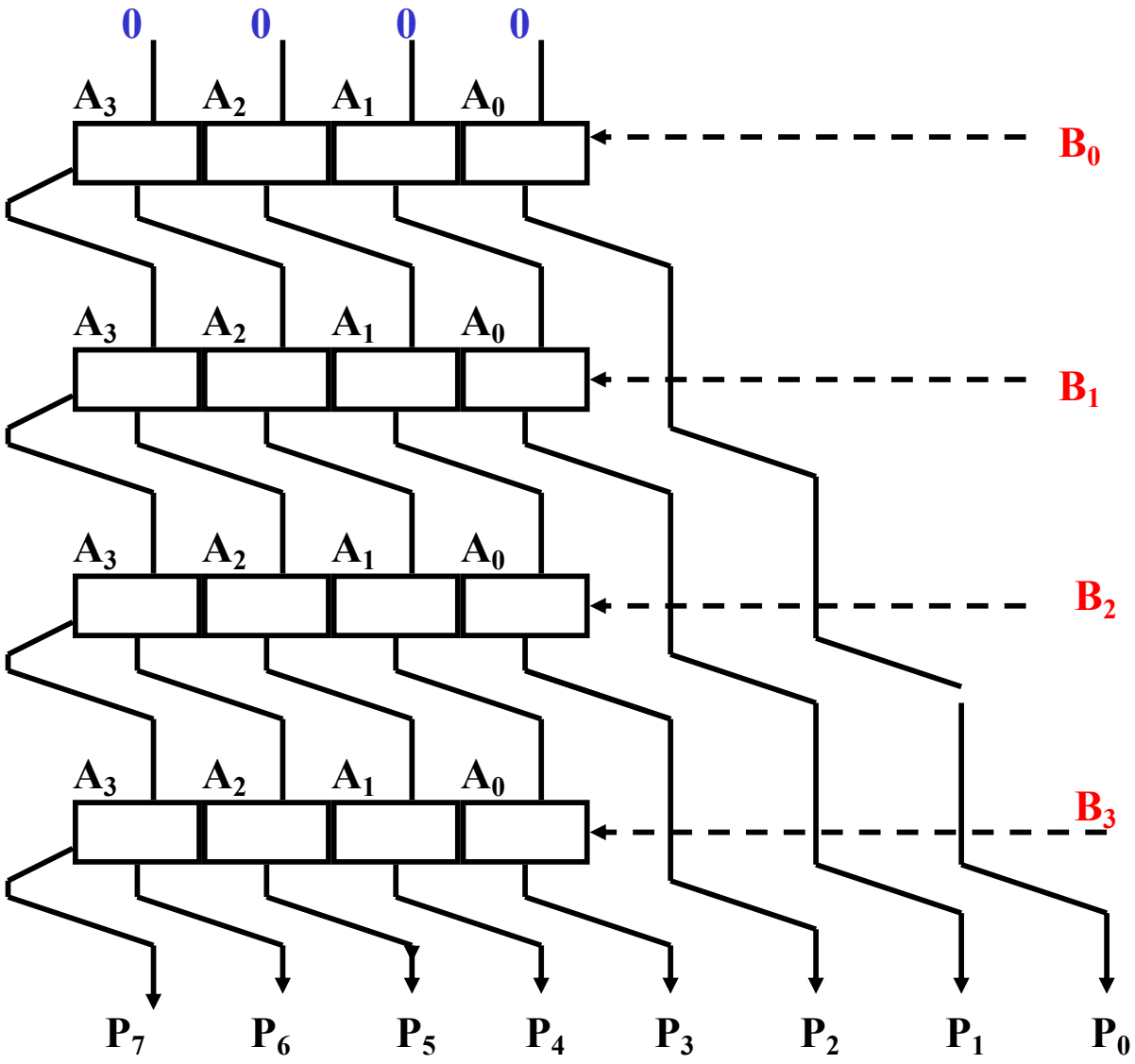
- 1 clock cycle per step => 96 clocks per multiply
- 1/2 bits in multiplicand always 0
  => 64-bit adder is wasted
- 0 is inserted when shift the multiplicand left
  => least significant bits of product never changed once formed

Multiply Hardware Version 2

- 32-bit Multiplicand reg, 32 -bit ALU, 64-bit Product reg, 32-bit Multiplier reg
- Instead of shifting multiplicand to left, shift product to right

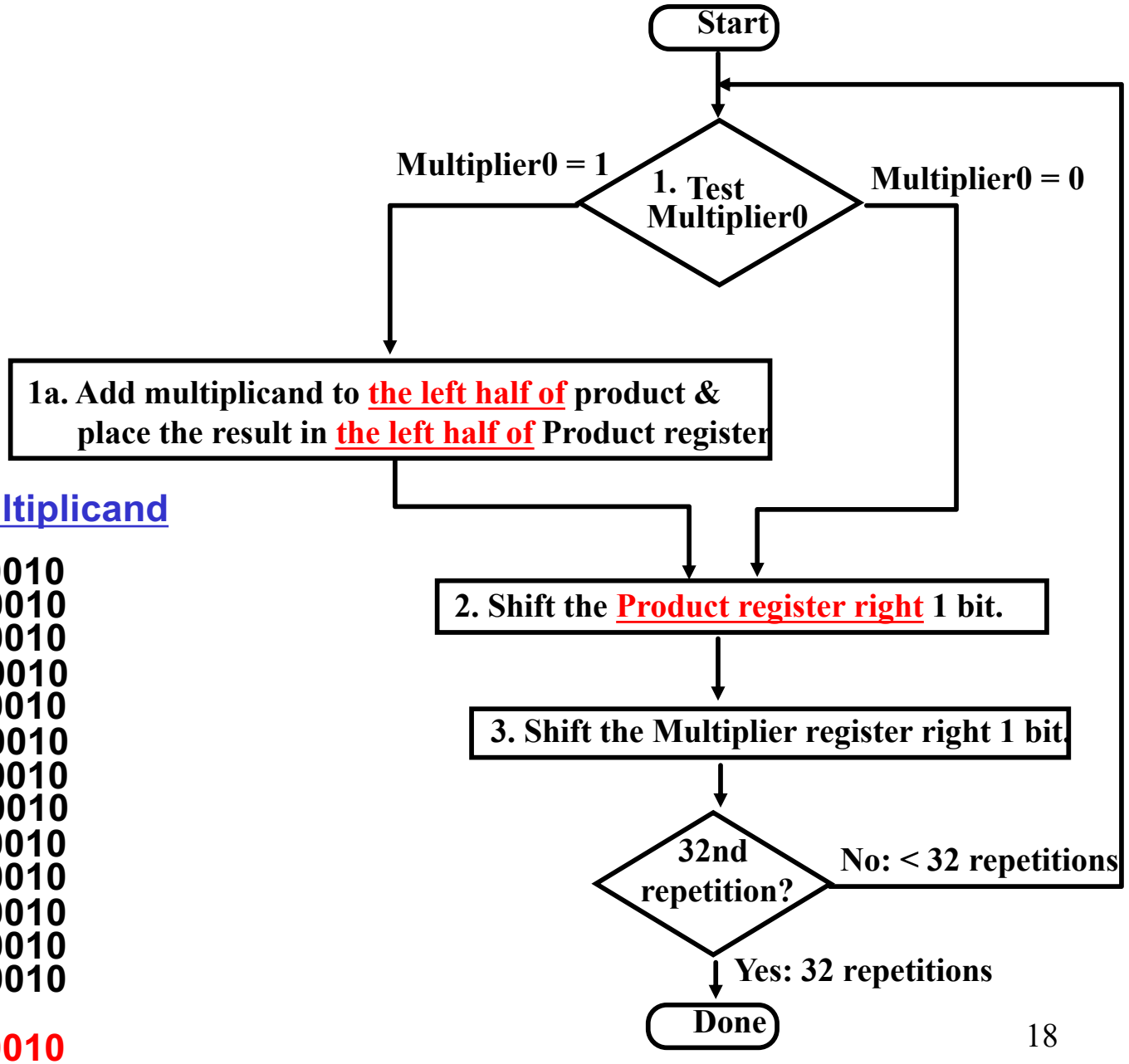


How does it work?



- Multiplicand stays still and product moves right

Multiply Algorithm Version 2



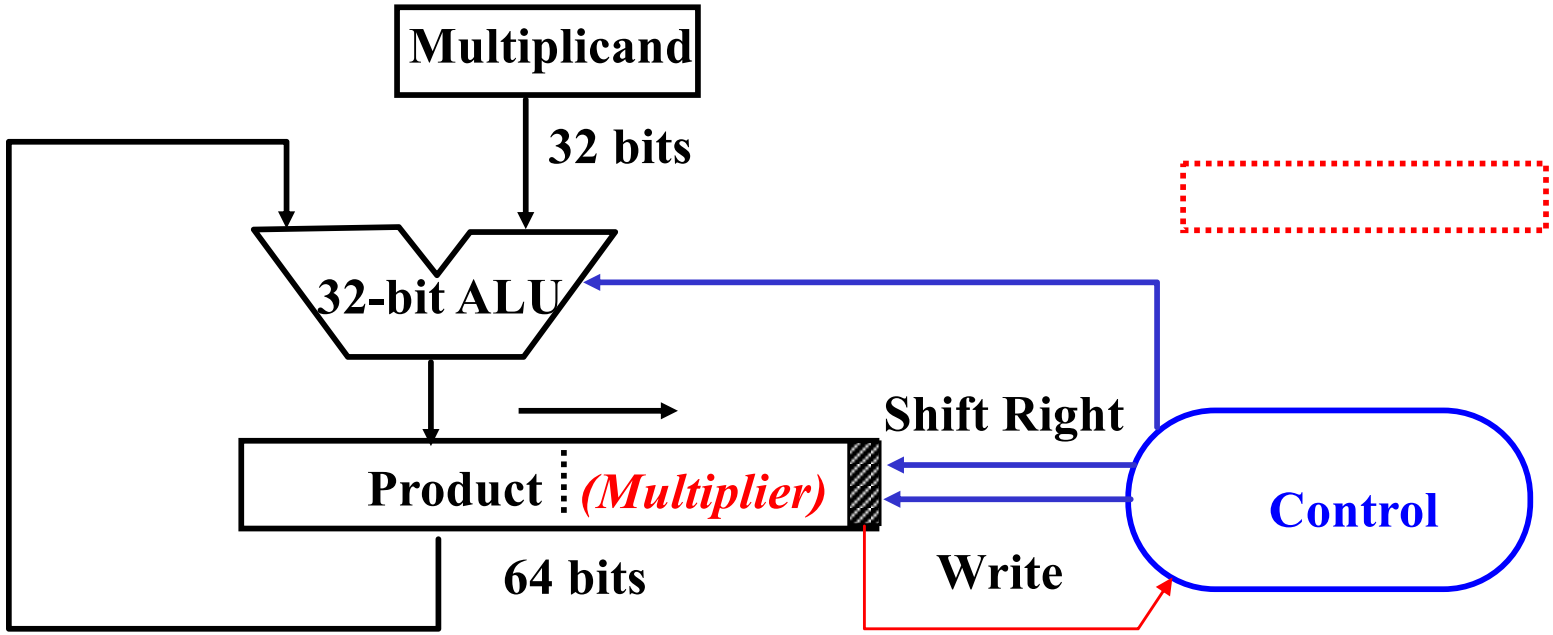
	Product	Multiplier	Multiplicand
	0000 0000	0011	0010
1:	0010 0000	0011	0010
2:	0001 0000	0011	0010
3:	0001 0000	0001	0010
1:	0011 0000	0001	0010
2:	0001 1000	0001	0010
3:	0001 1000	0000	0010
1:	0001 1000	0000	0010
2:	0000 1100	0000	0010
3:	0000 1100	0000	0010
1:	0000 1100	0000	0010
2:	0000 0110	0000	0010
3:	0000 0110	0000	0010
	0000 0110	0000	0010

Observations on Multiply Version 2

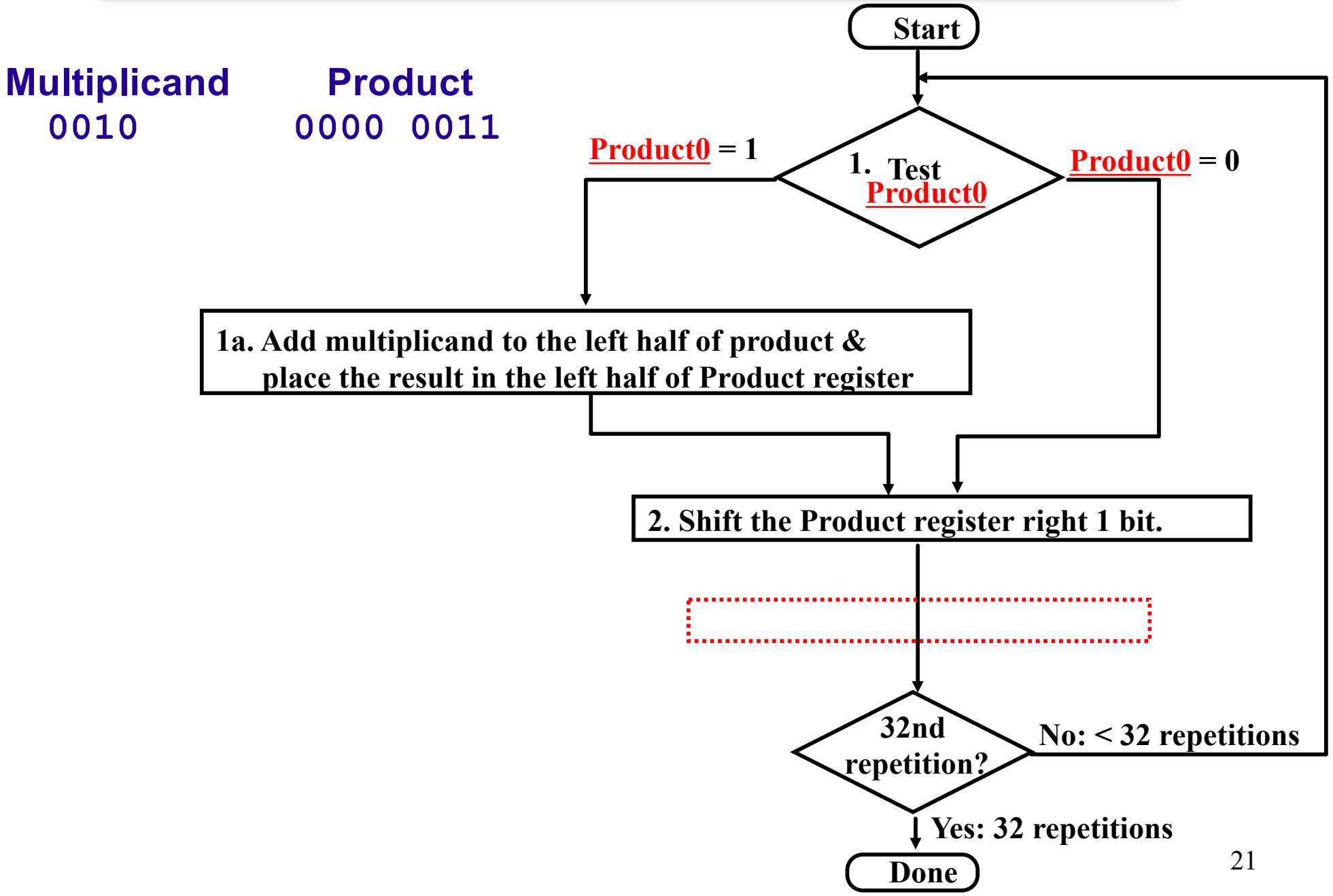
- Product register wastes space that exactly matches size of multiplier
- LSB of product never changed once formed
- LSB of multiplier will only be used once

Multiply Hardware Version 3

- 32-bit Multiplicand reg, 32 -bit ALU, 64-bit Product reg, (0-bit Multiplier reg)
- Combine Multiplier register and Product register



Multiply Algorithm Version 3



Signed Multiplication

- **Convert multiplier and multiplicand to positive numbers**
- **Remember the original signs**
- **Do unsigned multiplication**

Multiply Instructions in RISC-V

- **mul s0, s1, s2**
  - Multiply contents of register s1 by contents of register s2 and store low 32 bits of result in register s0

- **mulh s0, s1, s2**
  - Same but put the top 32 bits of the 64-bit result in s0

- **Complier has to generate code if overflow is a concern**
  - Use mulh to check – how?