ECE232: Hardware Organization and Design

Lecture 8: Multiplication

MULTIPLY (unsigned)

- Paper and pencil example (unsigned):

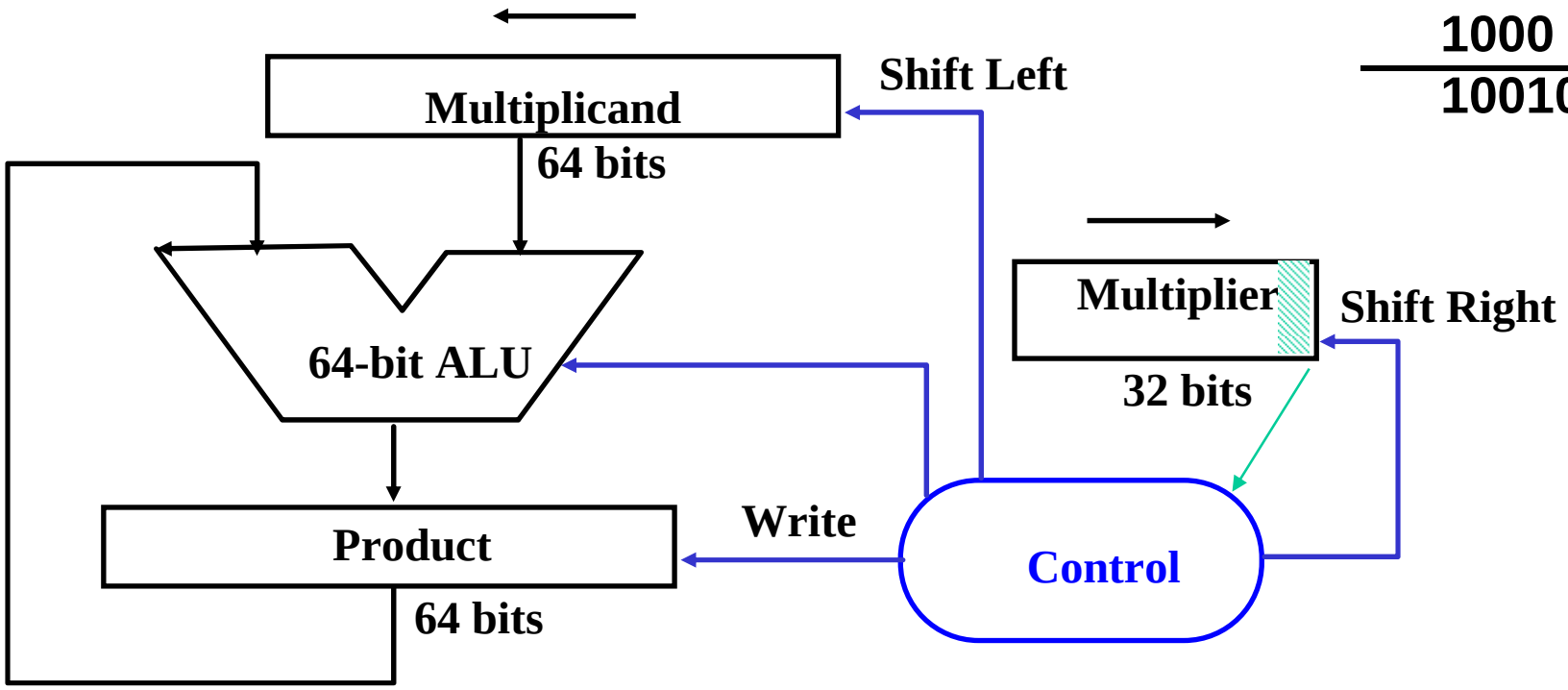
Multiplicand	**1000**
Multiplier	**1001**
	1000
	0000
	0000
	1000
Product	**1001000**

- m bits x n bits = m+n bit product
- Binary makes it easy:
  - 0 → place 0 (0 x multiplicand)
  - 1 → place a copy (1 x multiplicand)
- 3 versions of multiply hardware & algorithm:
  - successive refinement

Unsigned shift-add multiplier (version 1)

- 64-bit Multiplicand reg, 64-bit ALU,
  64-bit Product reg,
  32-bit multiplier reg

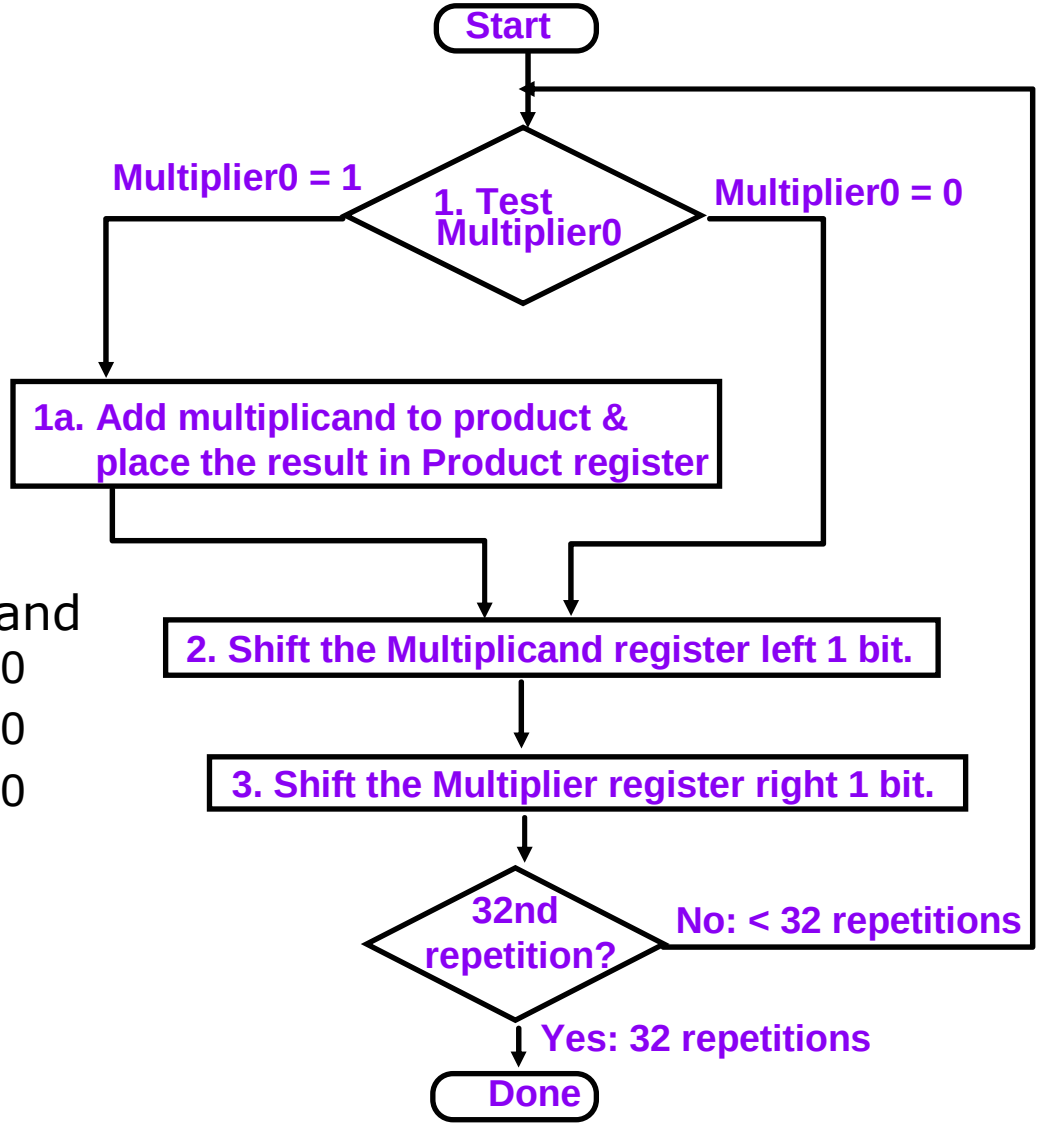
```
   1000
   1001
  ------
   1000
  0000
 0000
1000
------
1001000
```



Multiplier = datapath + control

Multiply Algorithm - Version 1: Control

- Product Multiplier Multiplicand
  0000 0000 0011 0000 0010
- 0000 0010 0001 0000 0100
- 0000 0110 0000 0000 1000
- 0000 0110



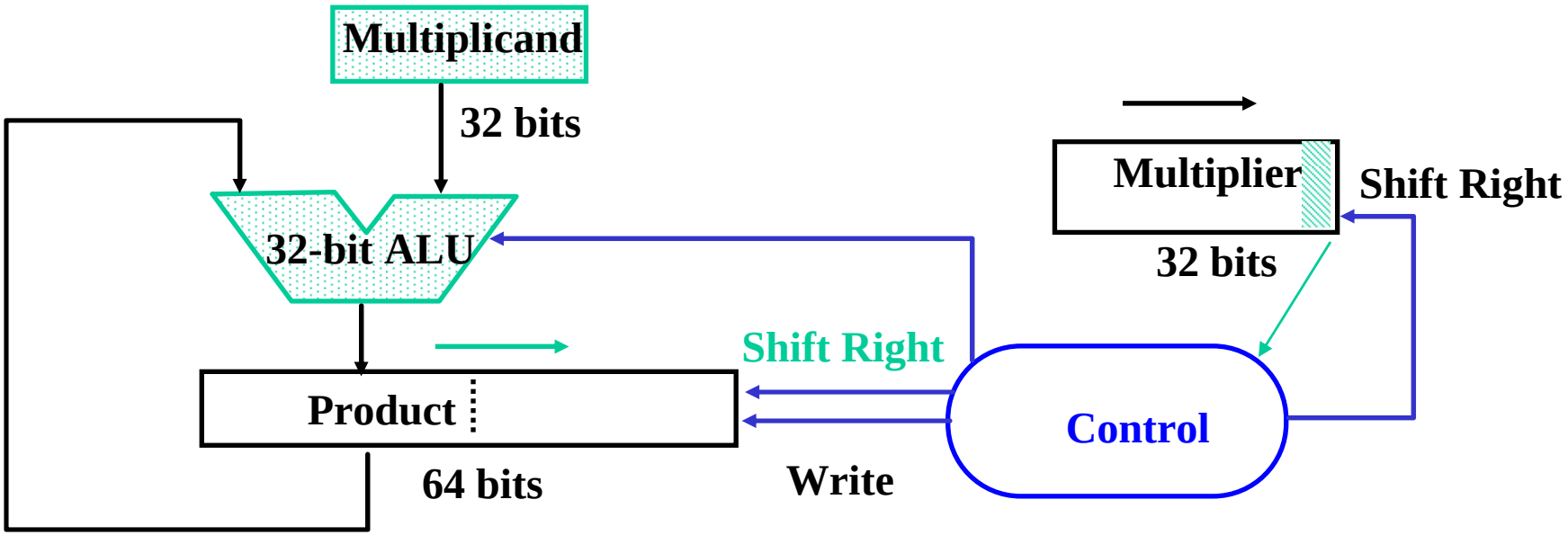
Observations on Multiply Version 1

- 1 cycle per step → 32x3 = ~ 100 cycles per multiply. However, One cycle per iteration can be saved by shifting multiplier and multiplicand in one cycle → 32x2
- 50% of the bits in multiplicand are 0
  → 64-bit adder is wasted
- 0s inserted in right of multiplicand as shifted to the left → least significant bits of product never changed once formed
- Instead of shifting multiplicand to left, shift product to the right

```
    1001
    1010
  ------
    0000
   1001
  0000
 1001
------
10100010
```

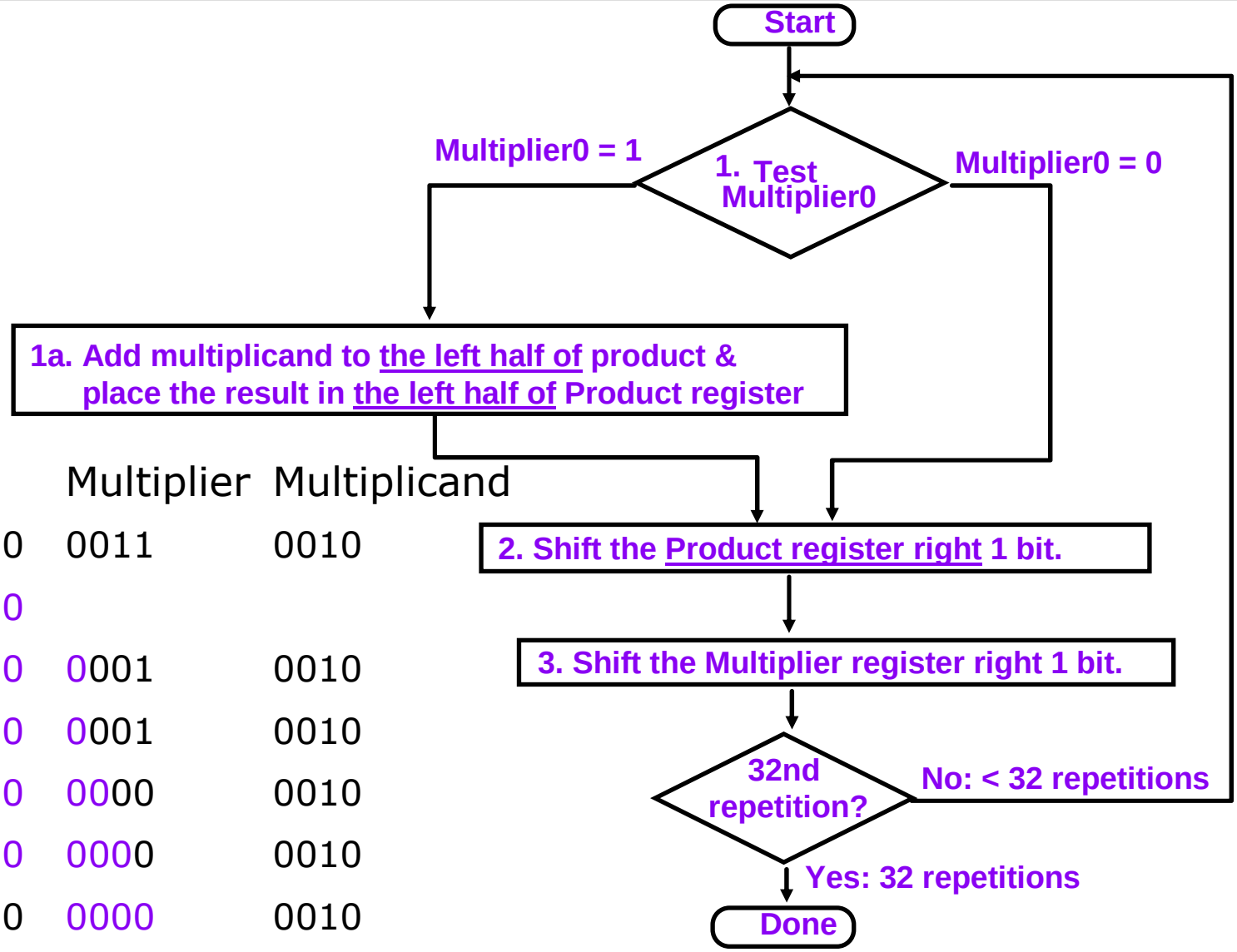
Multiply Hardware - Version 2

- 32-bit Multiplicand reg, 32 -bit ALU, 64-bit Product reg, 32-bit Multiplier reg



Multiply Algorithm - Version 2: Control

	Product	Multiplier	Multiplicand
1.	00000000	0011	0010
2.	00100000		
3.	00010000	0001	0010
4.	00110000	0001	0010
5.	00011000	0000	0010
6.	00001100	0000	0010
7.	00000110	0000	0010

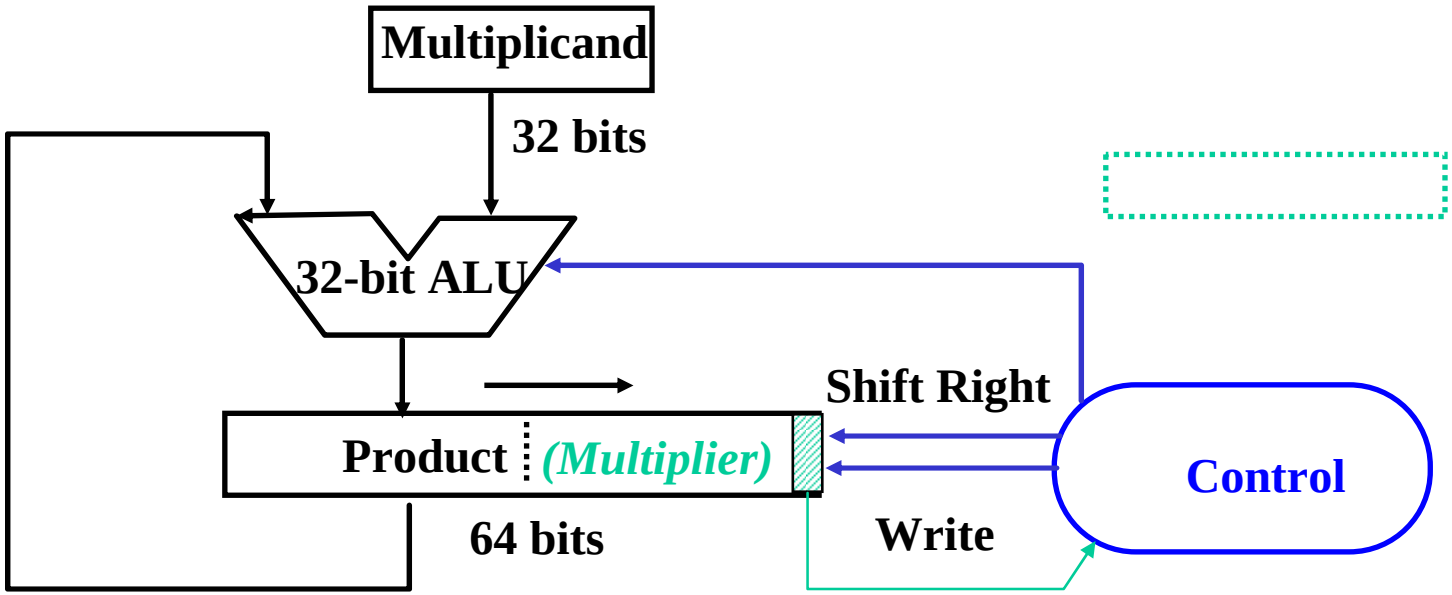


Multiply Hardware - Version 3

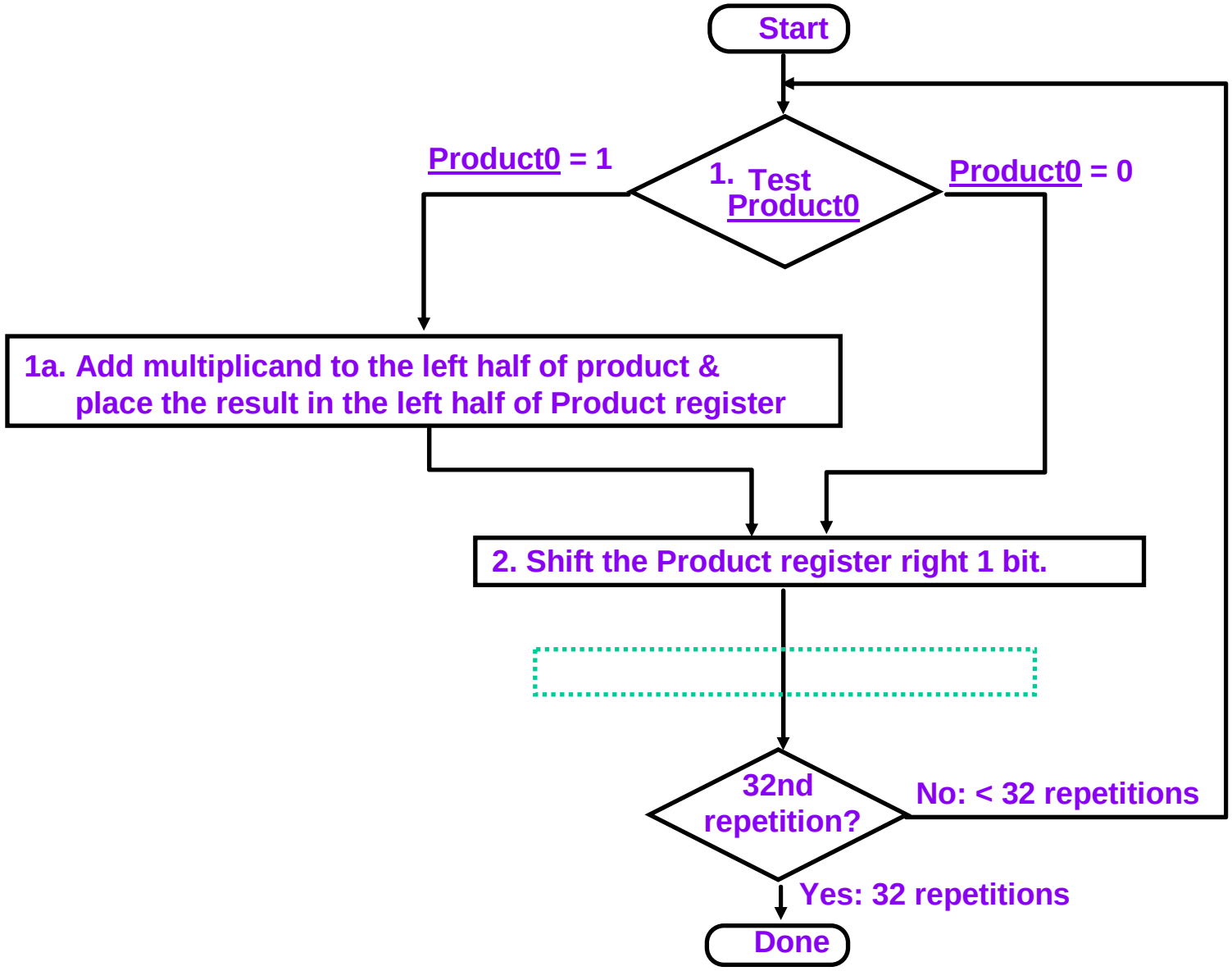
- Product register wastes space that exactly matches size of multiplier

  $\rightarrow$ combine Multiplier register and Product register

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



Multiply Algorithm - Version 3: Control



Observations on Multiply Version 3

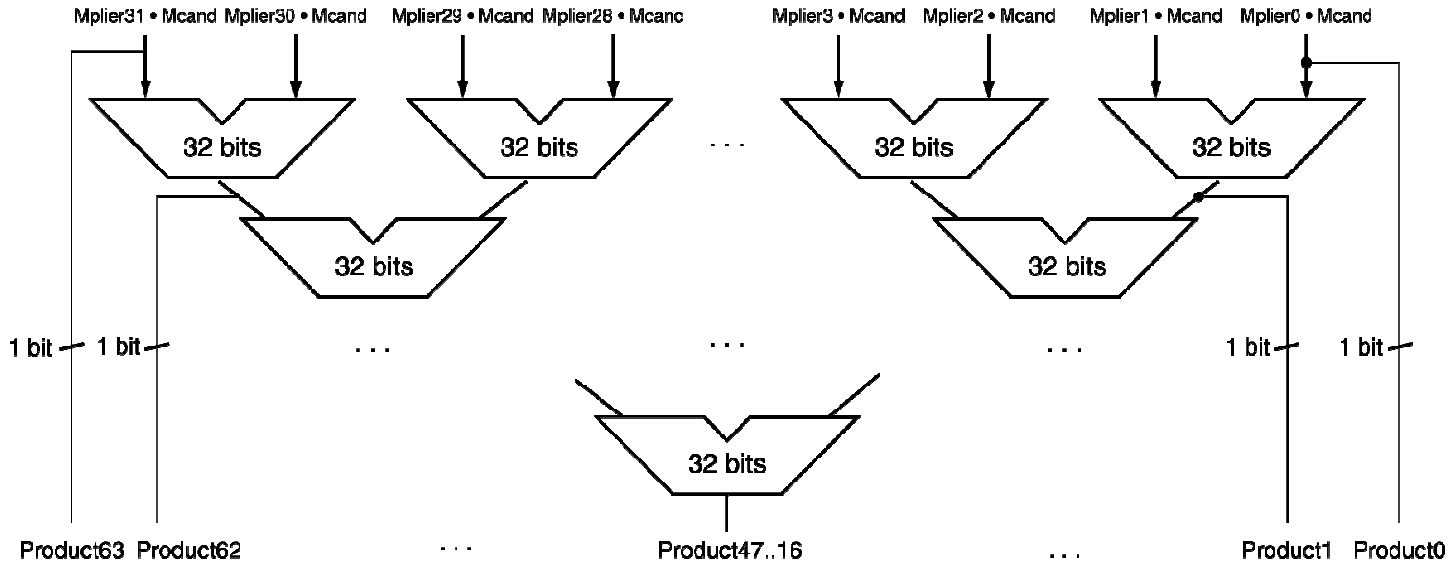
- 2 steps per bit because Multiplier & Product combined
- How can you make it faster?
- What about signed multiplication?
  - trivial solution: make both positive & complement product if one of operands is negative (leave out the sign bit, run for 31 steps)
  - apply definition of 2's complement
    - need to sign-extend partial products

A		1	0	1	1					-5
X	$\times$	0	0	1	1					3
$P^{(0)} = 0$		0	0	0	0					
$x_0 = 1 \Rightarrow$ Add A	$+$	1	0	1	1					
		1	0	1	1					
Shift		1	1	0	1	1				
$x_1 = 1 \Rightarrow$ Add A	$+$	1	0	1	1					
		1	0	0	0	1				
Shift		1	1	0	0	0	1			
$x_2 = 0 \Rightarrow$ Shift only		1	1	1	0	0	0	1		-15

Source: I. Koren, Computer Arithmetic Algorithms, 2nd Edition, 2002

Faster Multiplier

- Uses multiple adders
  - Cost/performance tradeoff



- Can be pipelined
  - Several multiplication performed in parallel

MIPS Multiplication

- Two 32-bit registers for product
  - HI: most-significant 32 bits
  - LO: least-significant 32-bits
- Instructions
  - `mult rs, rt  /  multu rs, rt`
    - 64-bit product in HI/LO
  - `mfhi rd  /  mflo rd`
    - Move from HI/LO to rd
    - Can test HI value to see if product overflows 32 bits
  - `mul rd, rs, rt`
    - Least-significant 32 bits of product –> rd

Summary

- Multiplication in computers generally takes longer than addition
- Lots of creative solutions for multiplier design
  - Minimize hardware
  - Best possible performance
- Note format of MIPs multiplication instructions
- Division similar