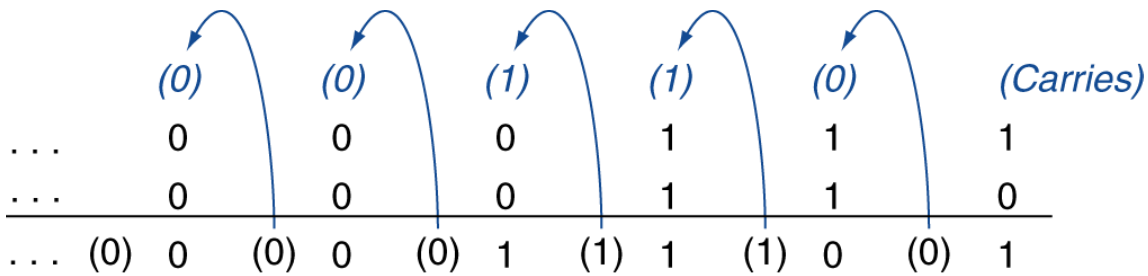
Chapter 3. Arithmetic for Computers

Binary Addition

- Adding 6_{10} and 7_{10}

```
    … 0000 0000 0000 0111
+   … 0000 0000 0000 0110
-------------------------
    … 0000 0000 0000 1101
```



- Overflow in addition
  - Not possible when signs are different
    - Magnitude of result must be smaller than operands

Binary Subtraction

- Subtracting 6 from 7: 7 - 6 = 7 + (-6)

```
  … 0000 0000 0000 0111
+ … 1111 1111 1111 1010
-----------------------
  … 0000 0000 0000 0001
```

2's complement representation of -6

- No overflow if signs of both operands are the same
- Overflow possible if operands have different signs

Overflow Detection

- Detecting overflow for 2's complement numbers
  - ex) In addition, it is the overflow if sign bit becomes 1

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

- Overflow of Unsigned Integers
  - Addition overflows if the sum is less than either of the addends
    - Overflow if A+B=S and S < A or S < B
  - Subtraction overflows if the difference is greater than the minuend
    - Overflow if A-B=S and S > A
- How to handle overflow differs from language to language
  - C, Java: ignore integer overflow
  - Ada, Fortran: overflow must be notified to program

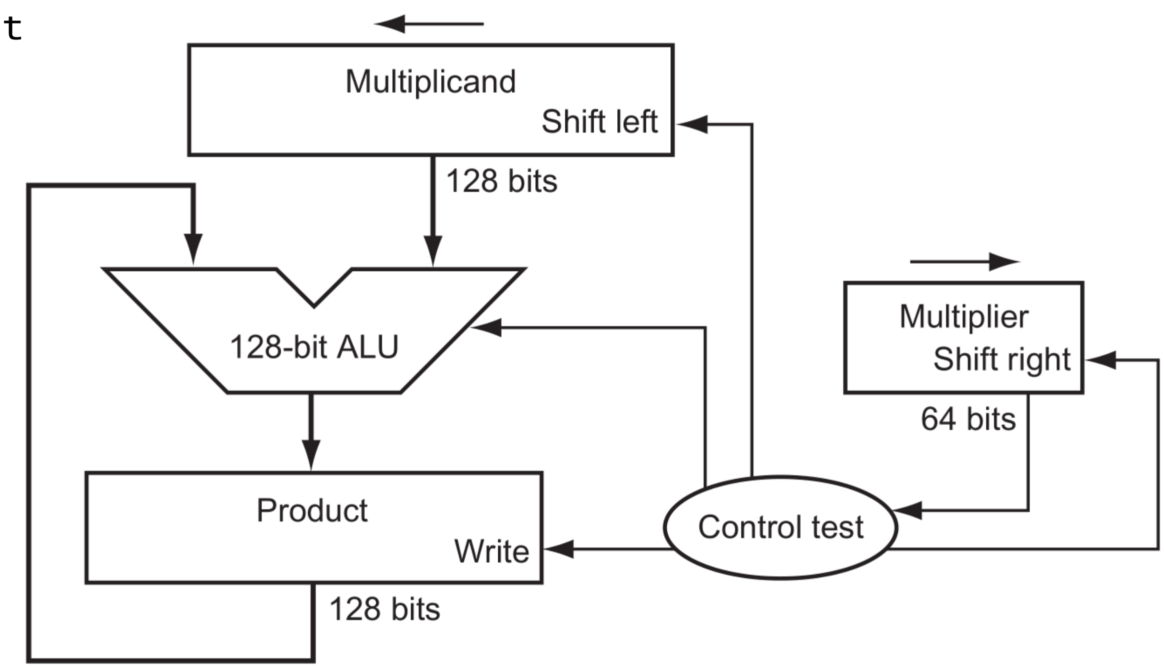
Multiplication

- Steps of multiplication

multiplicand		1000_{ten}
multiplier	×	1001_{ten}
		1000
		0000
		0000
		1000
product		10010000_{t}

Max Length of product =
length of multiplicand
+ length of multiplier

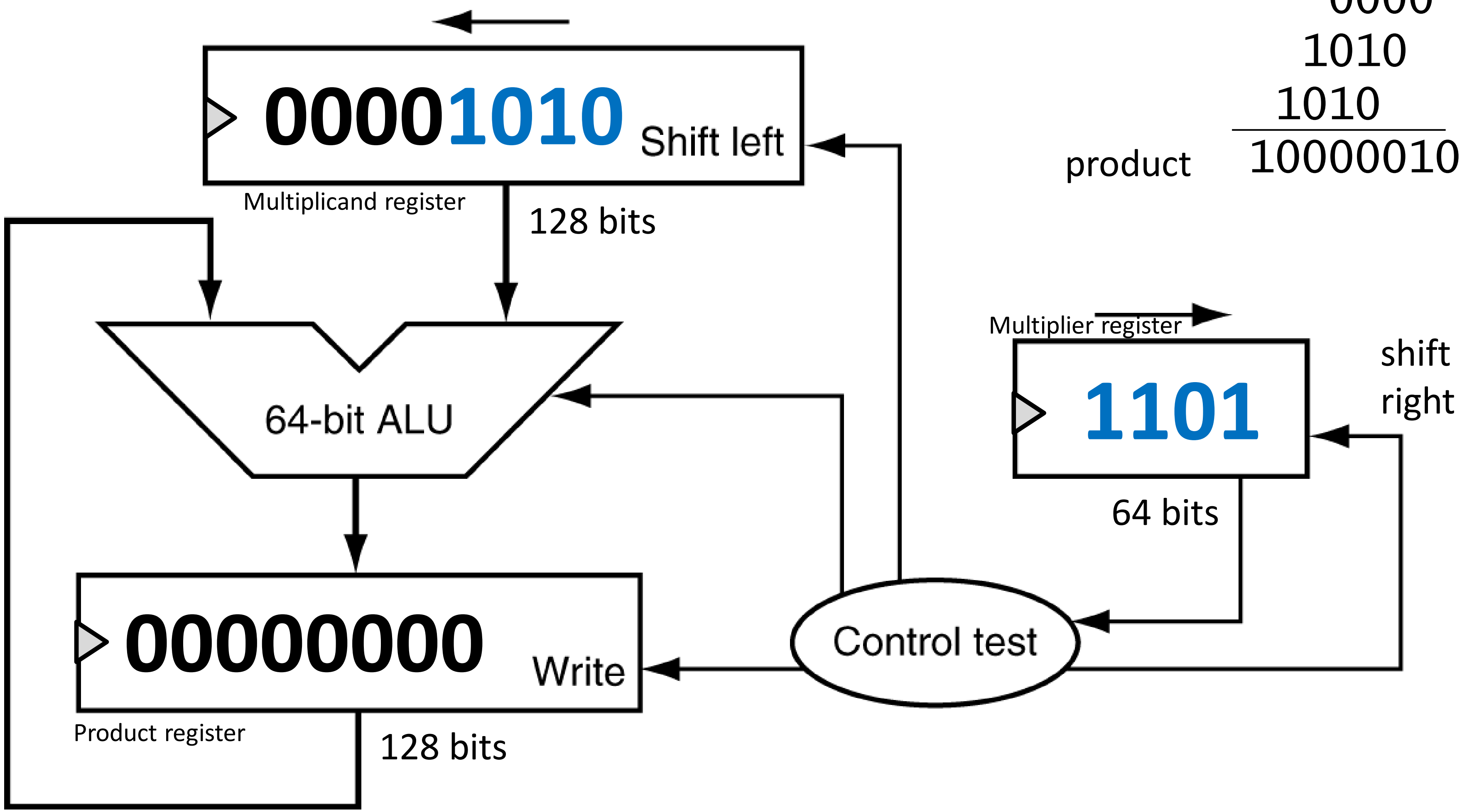
- Multiplication Hardware
  - First version
  - Sequential processing



Multiplication

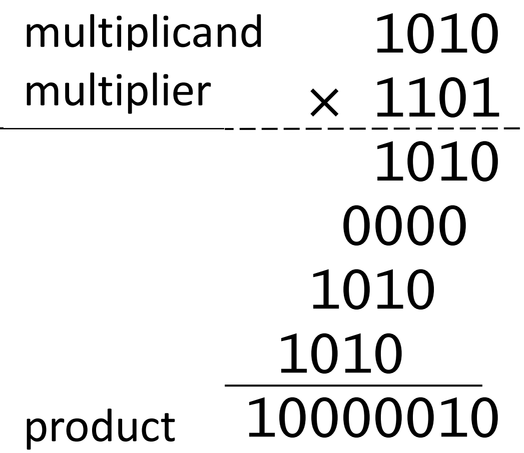
multiplicand	1010
multiplier	× 1101
	1010
	0000
	1010
	1010
product	10000010

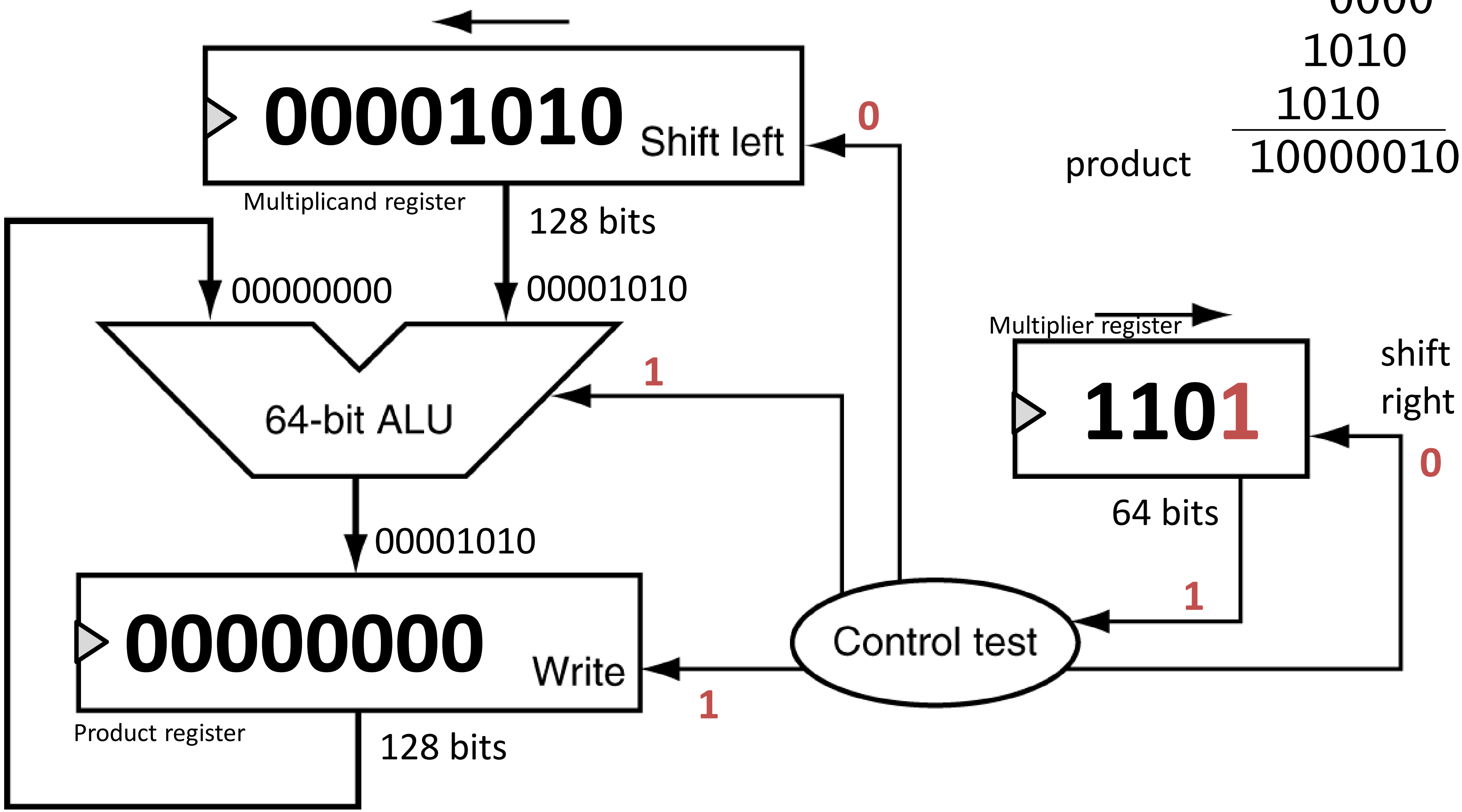
Initialization



Multiplication

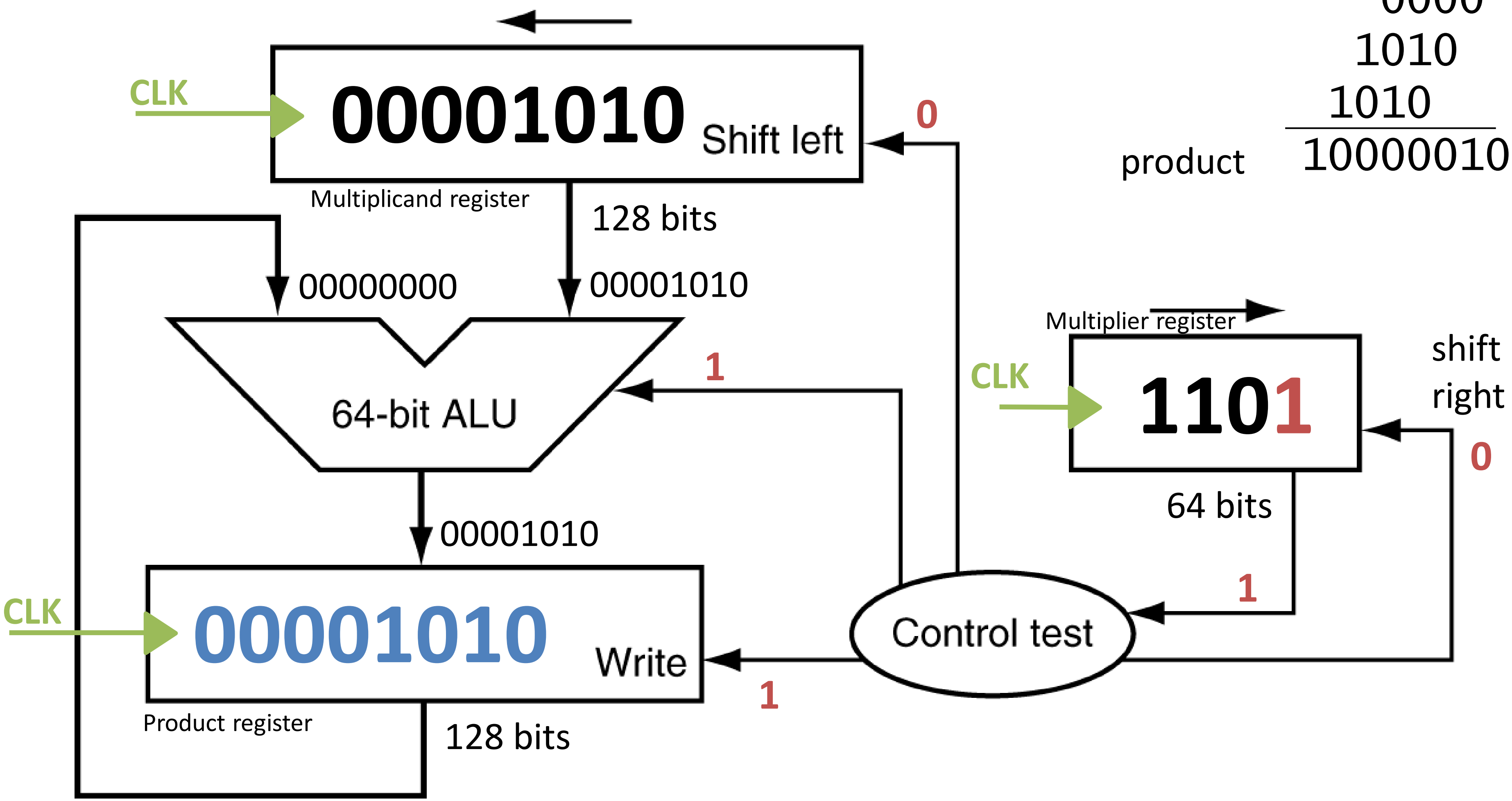
Iteration 1: Control signal set-up

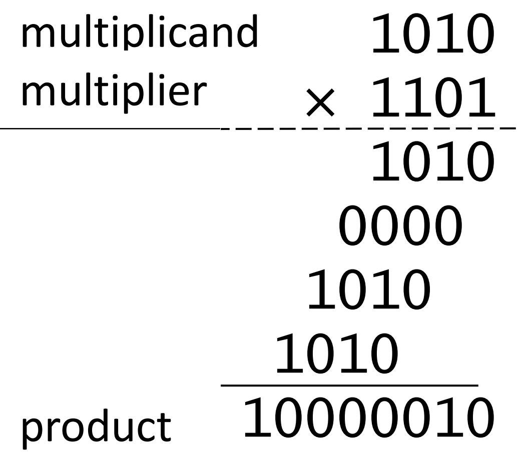




Multiplication

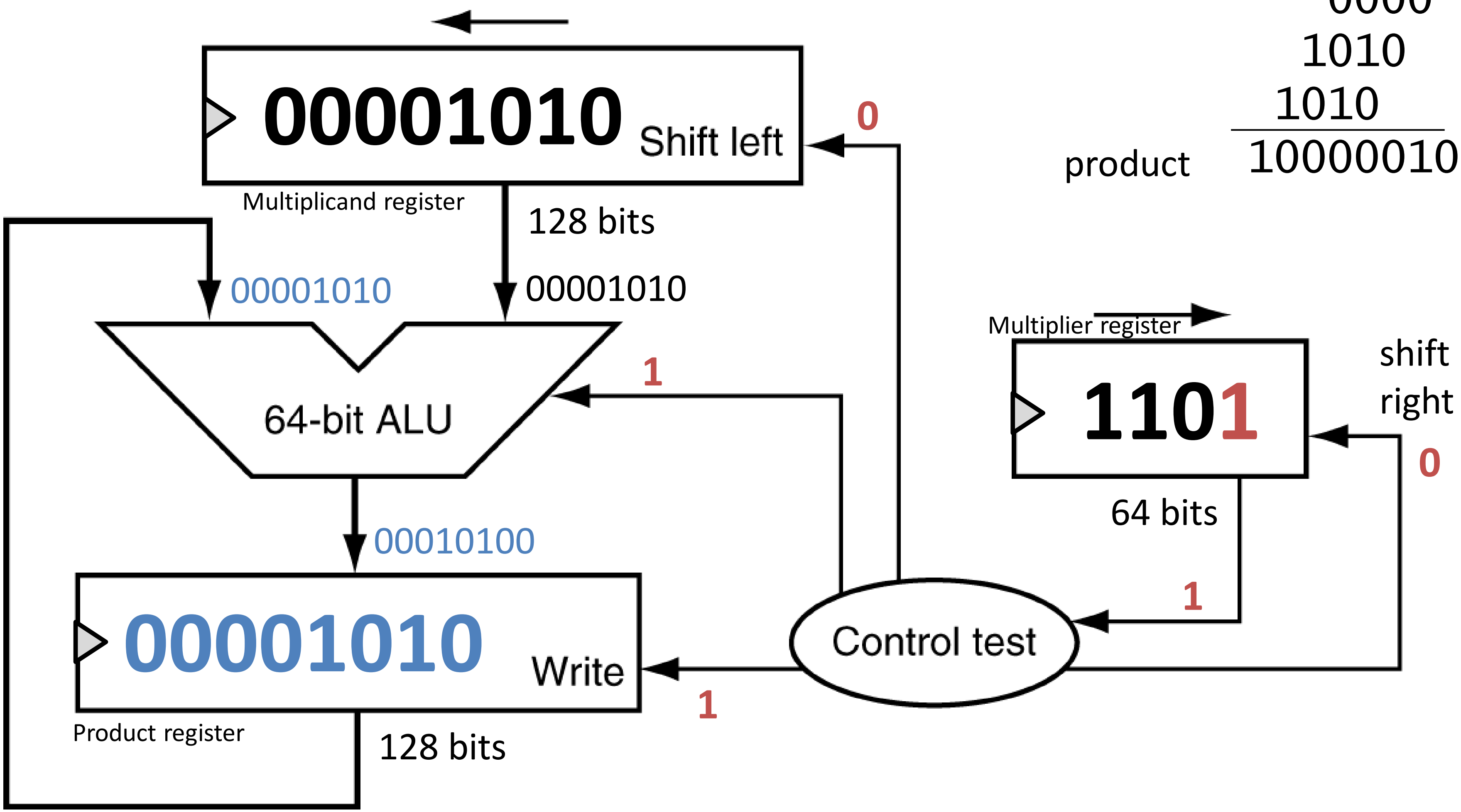
Iteration 1: Clock signal firing





Multiplication

Iteration 1: Product register updated

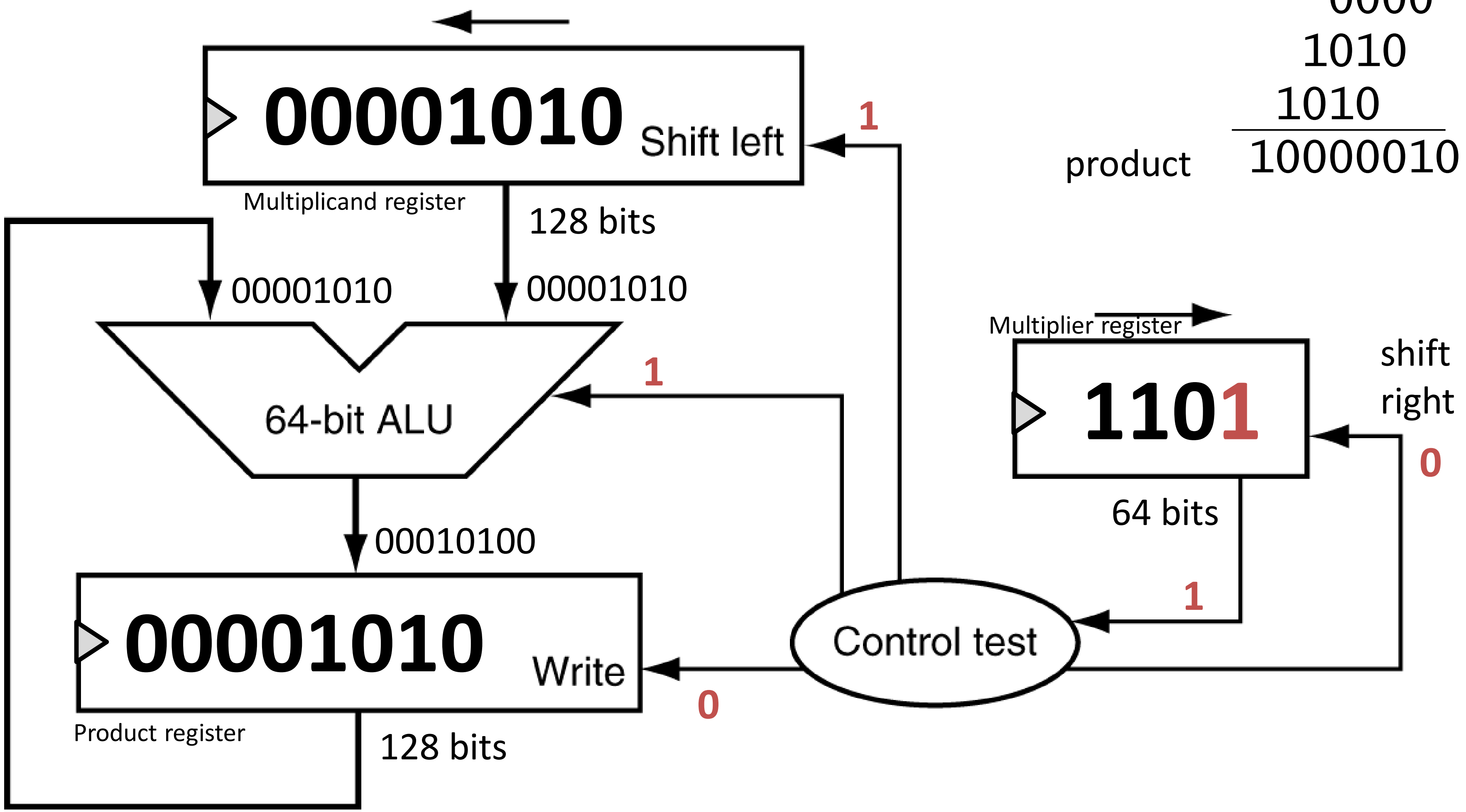


multiplicand	1010
multiplier	× 1101
	1010
	0000
	1010
	1010
product	10000010

Multiplication

multiplicand		1010
multiplier	×	1101
		1010
		0000
		1010
		1010
product		10000010

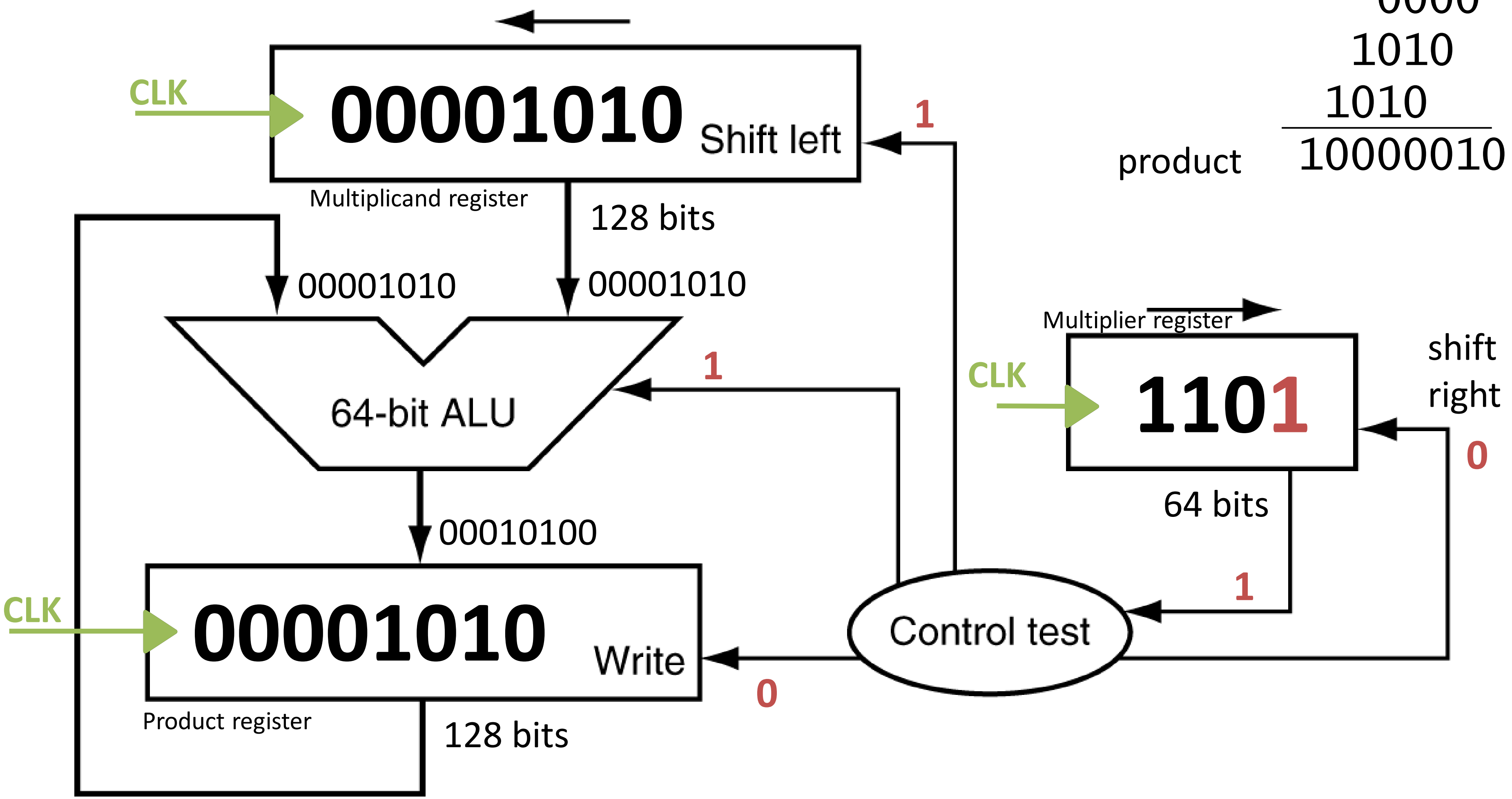
Iteration 1: Control signal for shifting multiplicand



Multiplication

multiplicand		1010
multiplier	×	1101
		1010
		0000
		1010
		1010
product		10000010

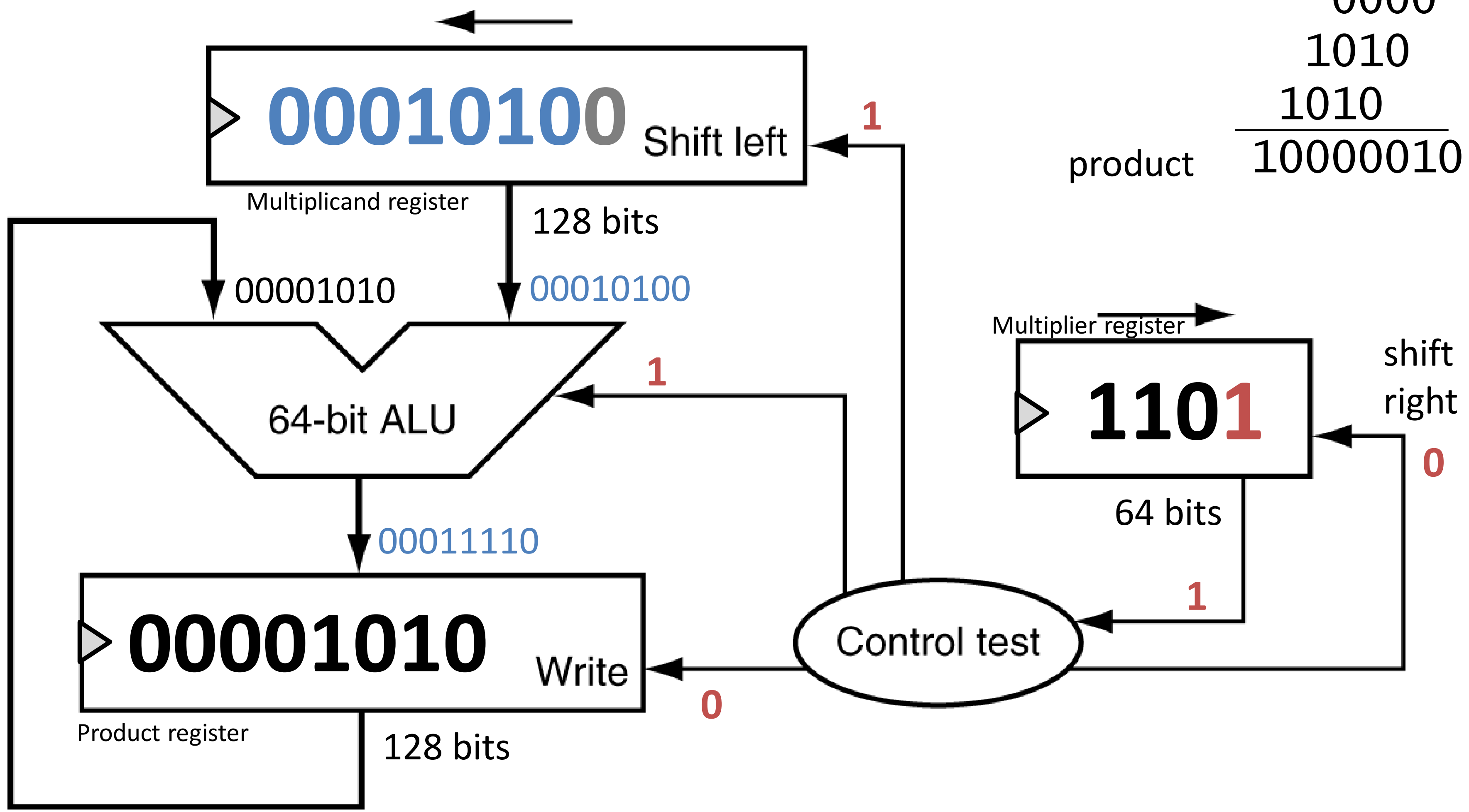
Iteration 1: Clock firing



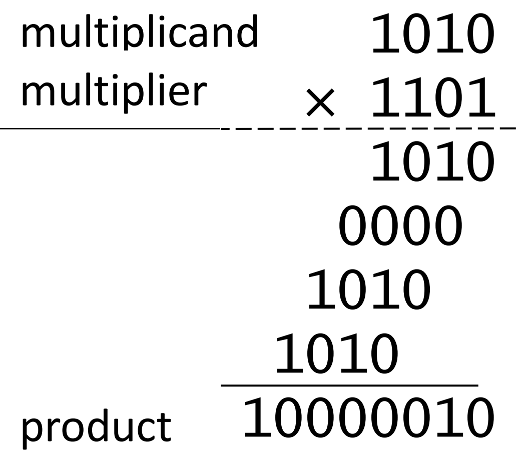
Multiplication

multiplicand	1010
multiplier	× 1101
	1010
	0000
	1010
	1010
product	10000010

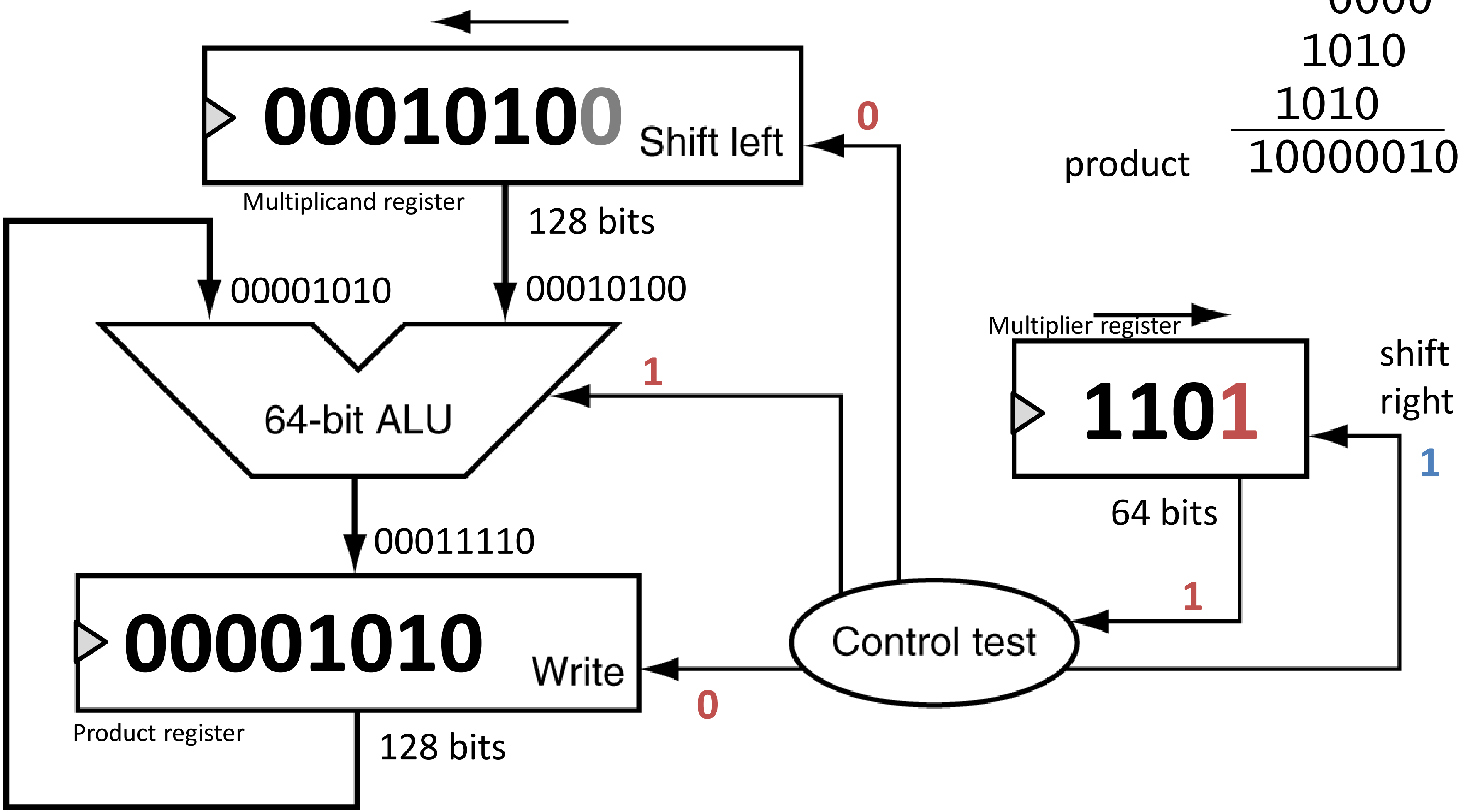
Iteration 1: Multiplicand shifted left



Multiplication



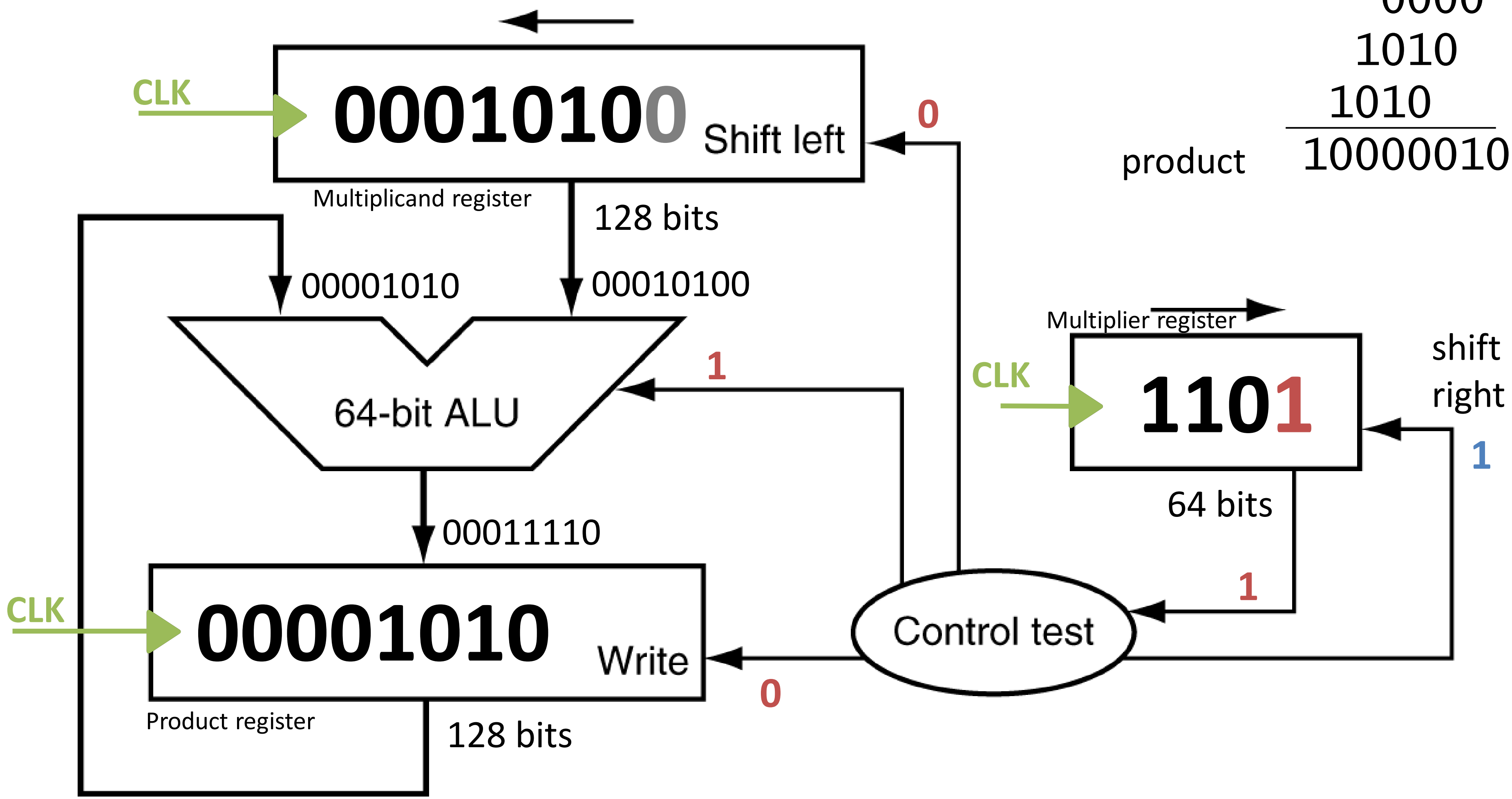
Iteration 1: Control signal for shifting multiplier



Multiplication

```
multiplicand       1010
multiplier       × 1101
-----------------------
                   1010
                  0000
                 1010
                1010
-----------------------
product         10000010
```

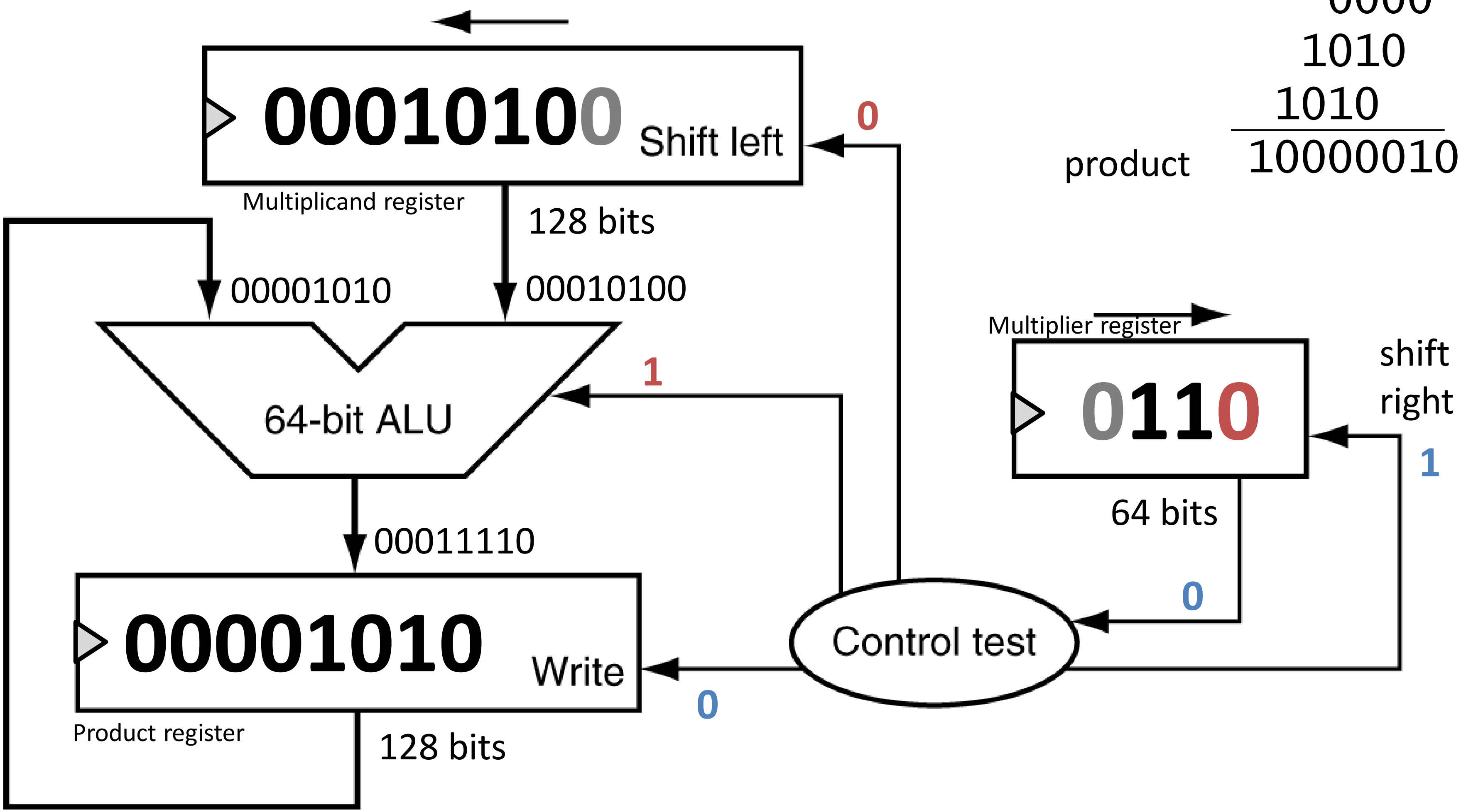
Iteration 1: Clock firing



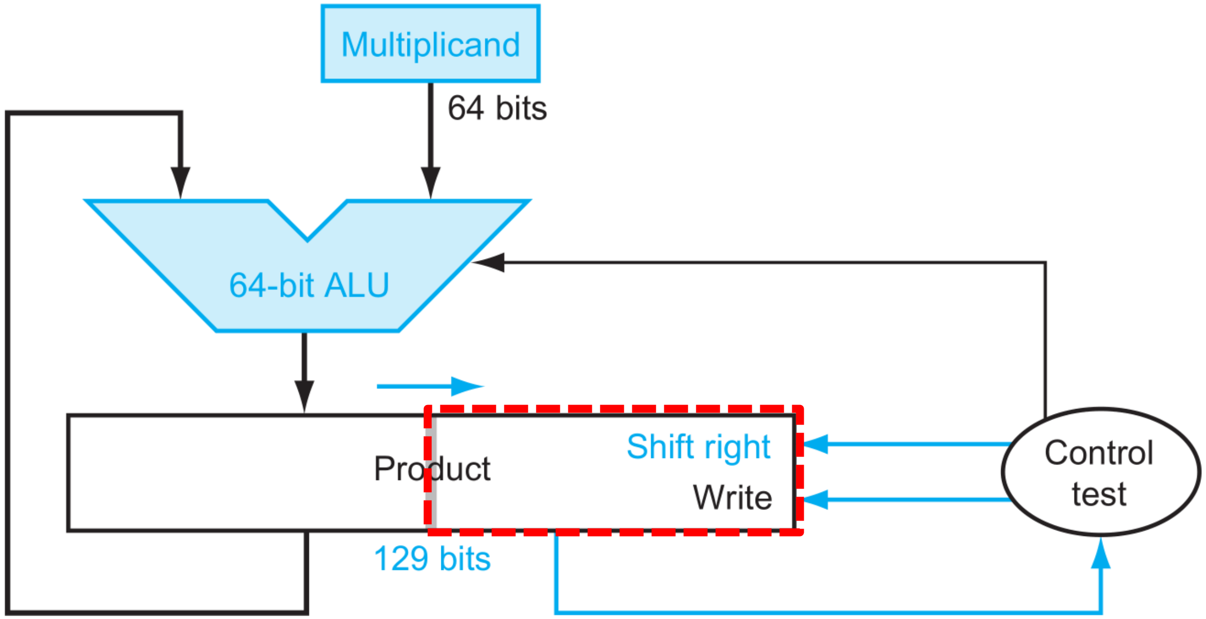
Multiplication

multiplicand		1010
multiplier	×	1101
		1010
		0000
		1010
		1010
product		10000010

Iteration 1: Multiplier shifted



Improving the Multiplication

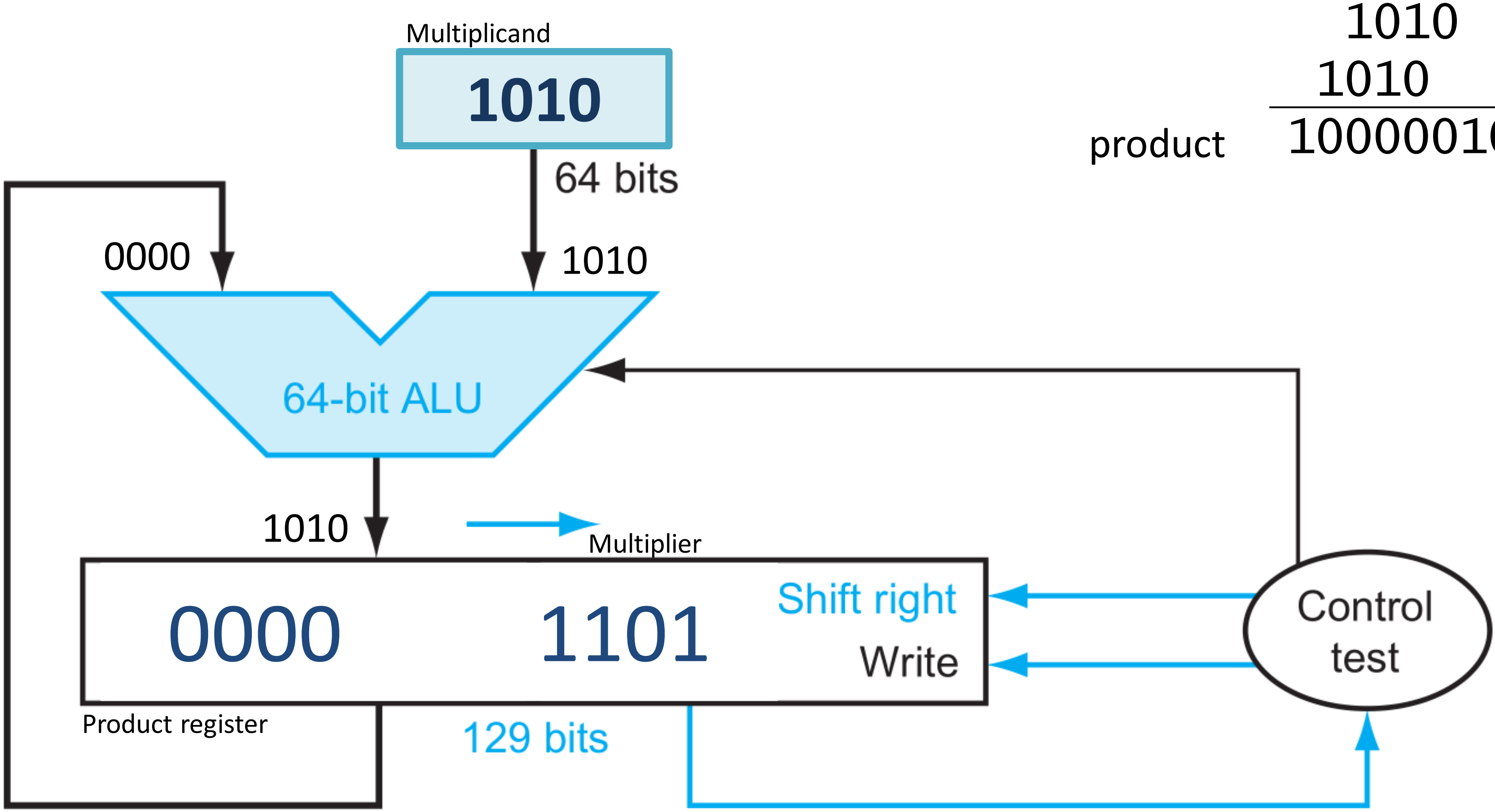


- Multiplicand register, ALU, Multiplier register are 64 bits
- Product register is 129 bits
- Multiplier placed in the right half of product register
- Addition and shifts occur in parallel

Improving the Multiplication

multiplicand	1010
multiplier	× 1101
	1010
	0000
	1010
	1010
product	10000010

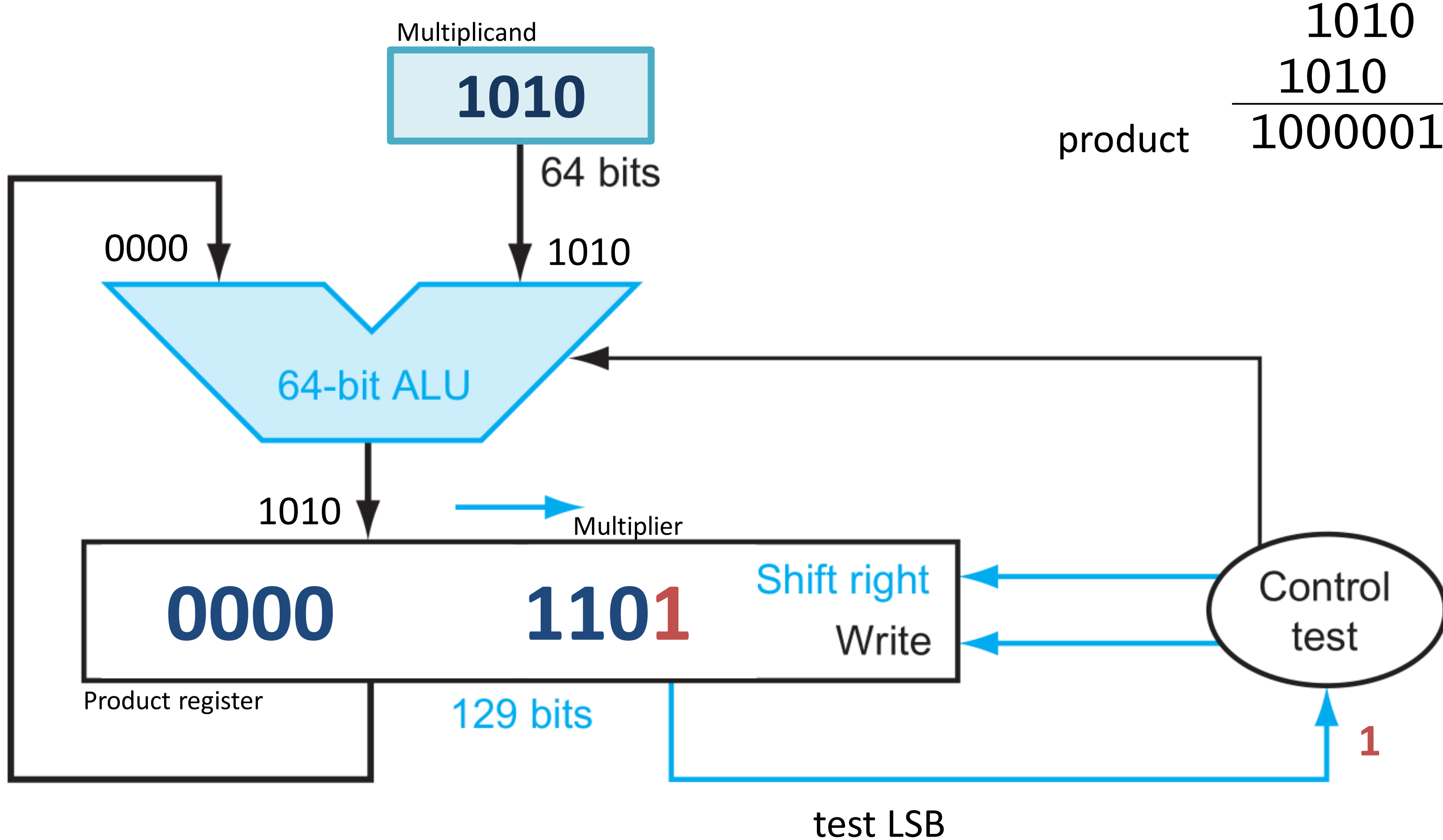
Initialization



Improving the Multiplication

multiplicand	1010
multiplier	× 1101
	1010
	0000
	1010
	1010
product	10000010

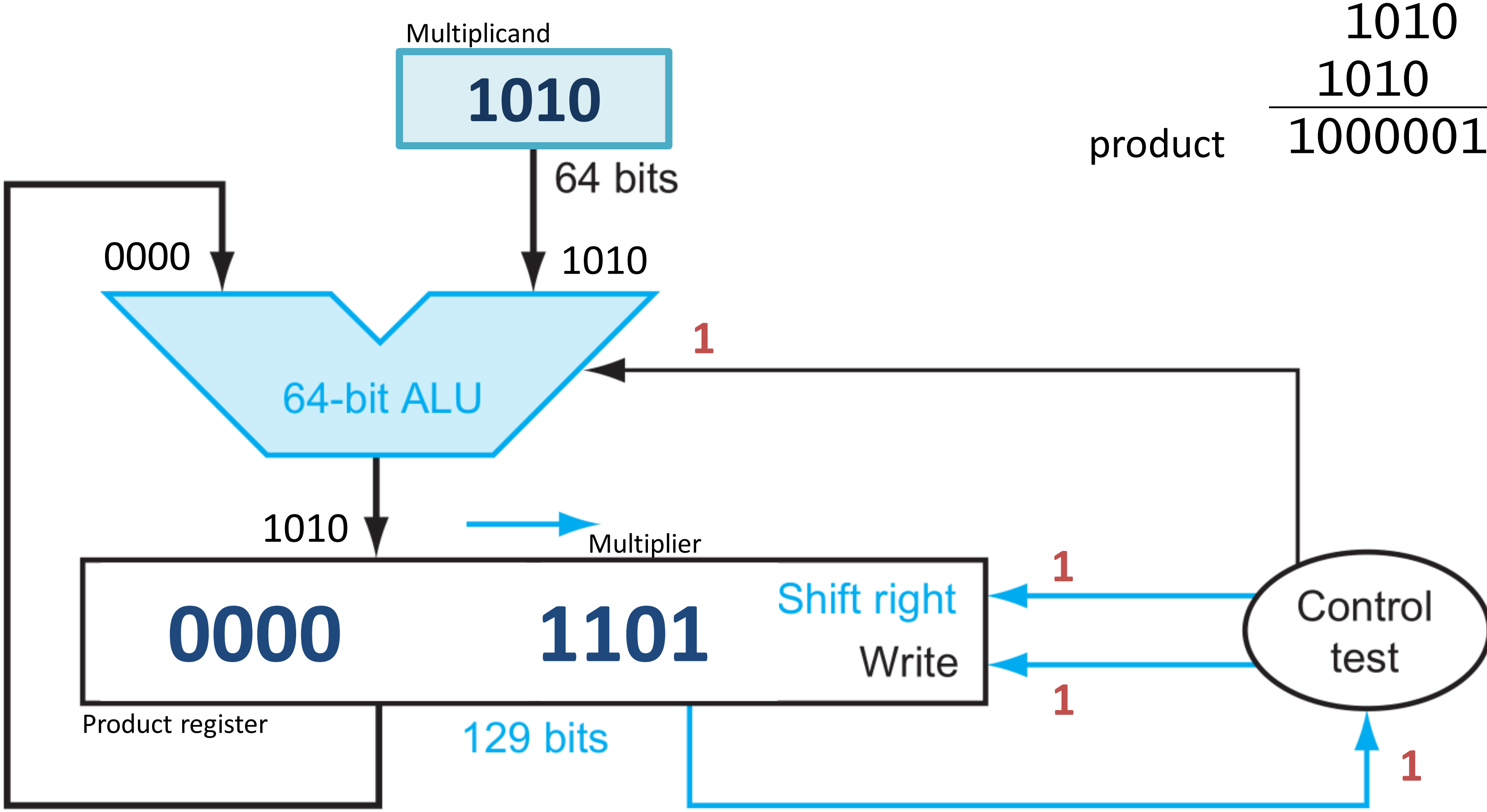
Iteration 1: Test LSB



Improving the Multiplication

multiplicand	1010
multiplier	× 1101
	1010
	0000
	1010
	1010
product	10000010

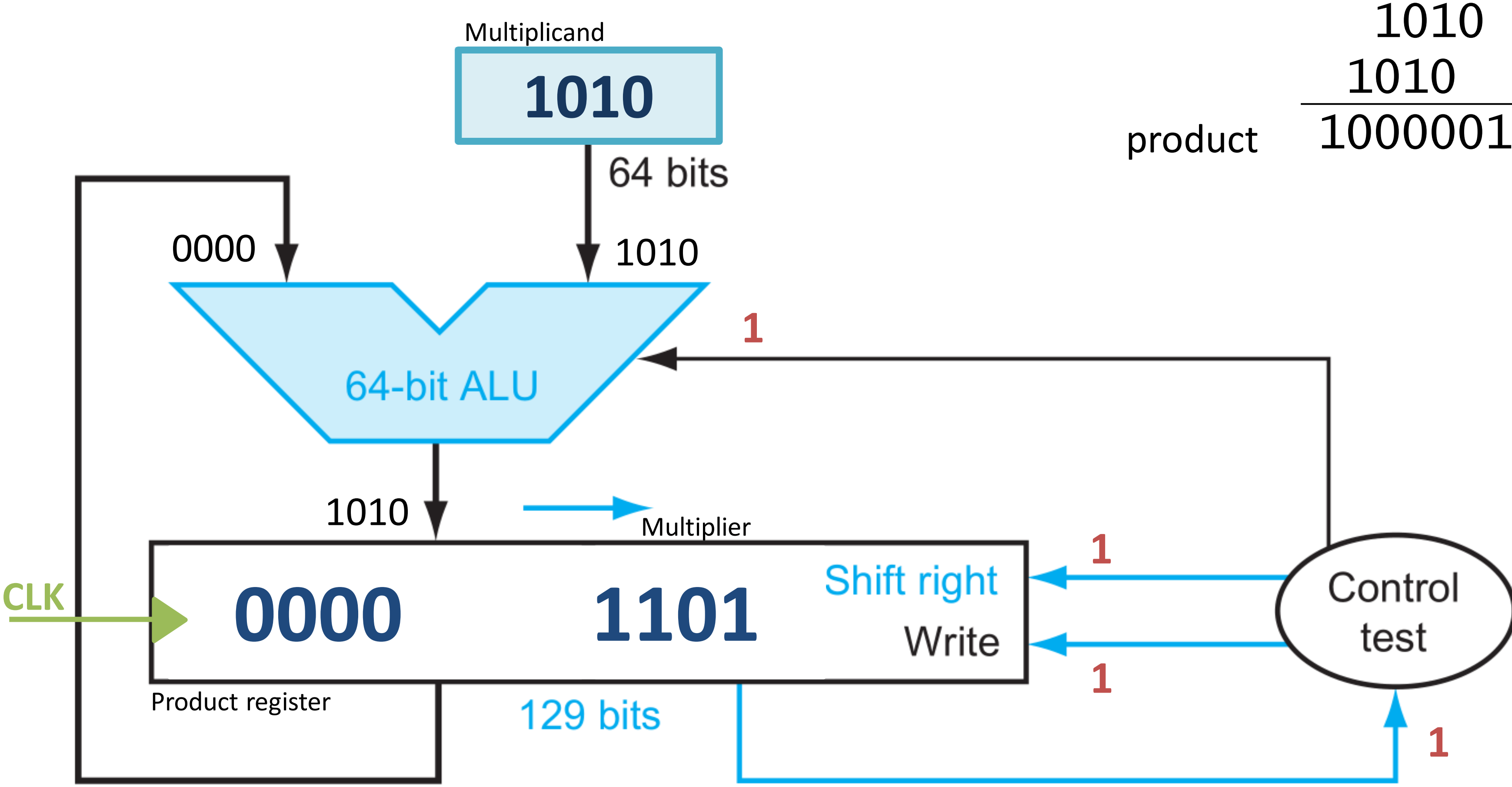
Iteration 1: Control signal setup



Improving the Multiplication

```
multiplicand          1010
multiplier          × 1101
---------------------------
                      1010
                     0000
                    1010
                   1010
             -------------
product            10000010
```

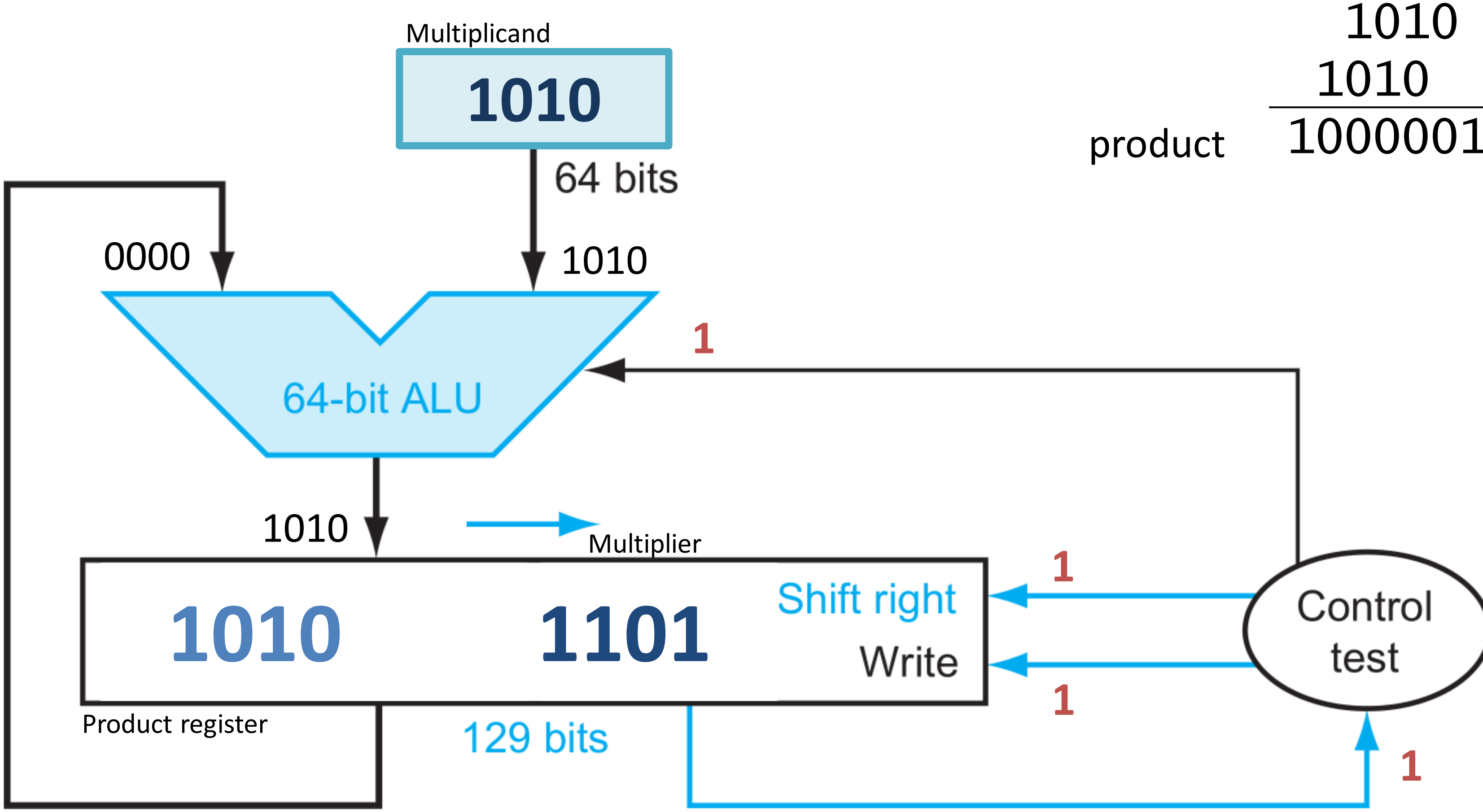
Iteration 1: Clock firing



Improving the Multiplication

```
multiplicand        1010
multiplier        × 1101
------------------------
                    1010
                   0000
                  1010
                 1010
------------------------
product          10000010
```

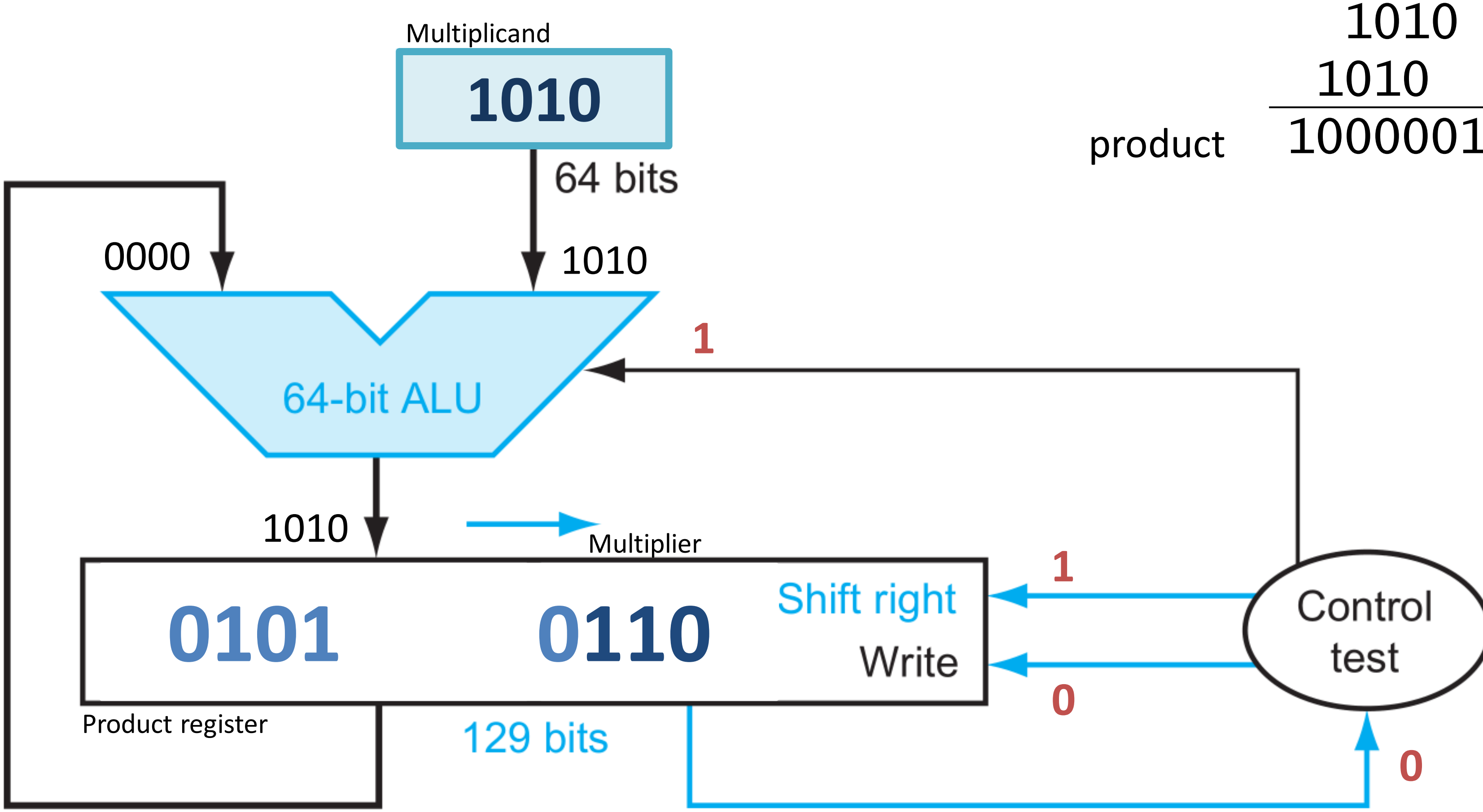
Iteration 1: Multiplicand added



Improving the Multiplication

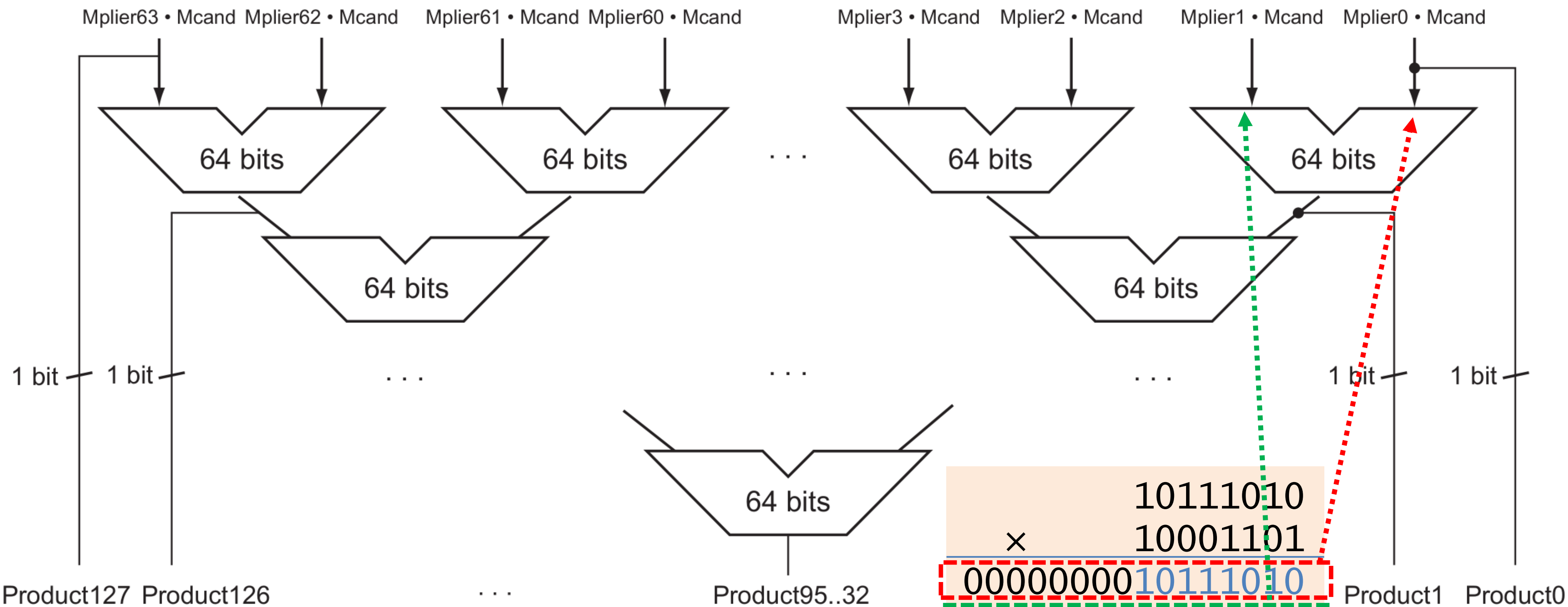
```
multiplicand        1010
multiplier        × 1101
--------------------------
                    1010
                   0000
                  1010
                 1010
              ------------
product         10000010
```

Iteration 1: Product register shifted



Faster Multiplication

- Use multiple adders since multiplier is known at the beginning




```
          10111010
×         10001101
  0000000010111010
  0000000000000000
  0000001011101000
  0000010111010000
  0000000000000000
  0000000000000000
  0000000000000000
  0101110100000000
  0110011001110010
```

- RISC-V instructions for multiplication
  - Multiply: `mul rdl, rs1, rs2`
  - Multiply high: `mulh rdh, rs1, rs2`
  - Multiply high unsigned: `mulhu`
  - Multiply high signed unsigned: `mulhsu`

    `mulhsu rdh, rs1, rs2`

    signed

Division

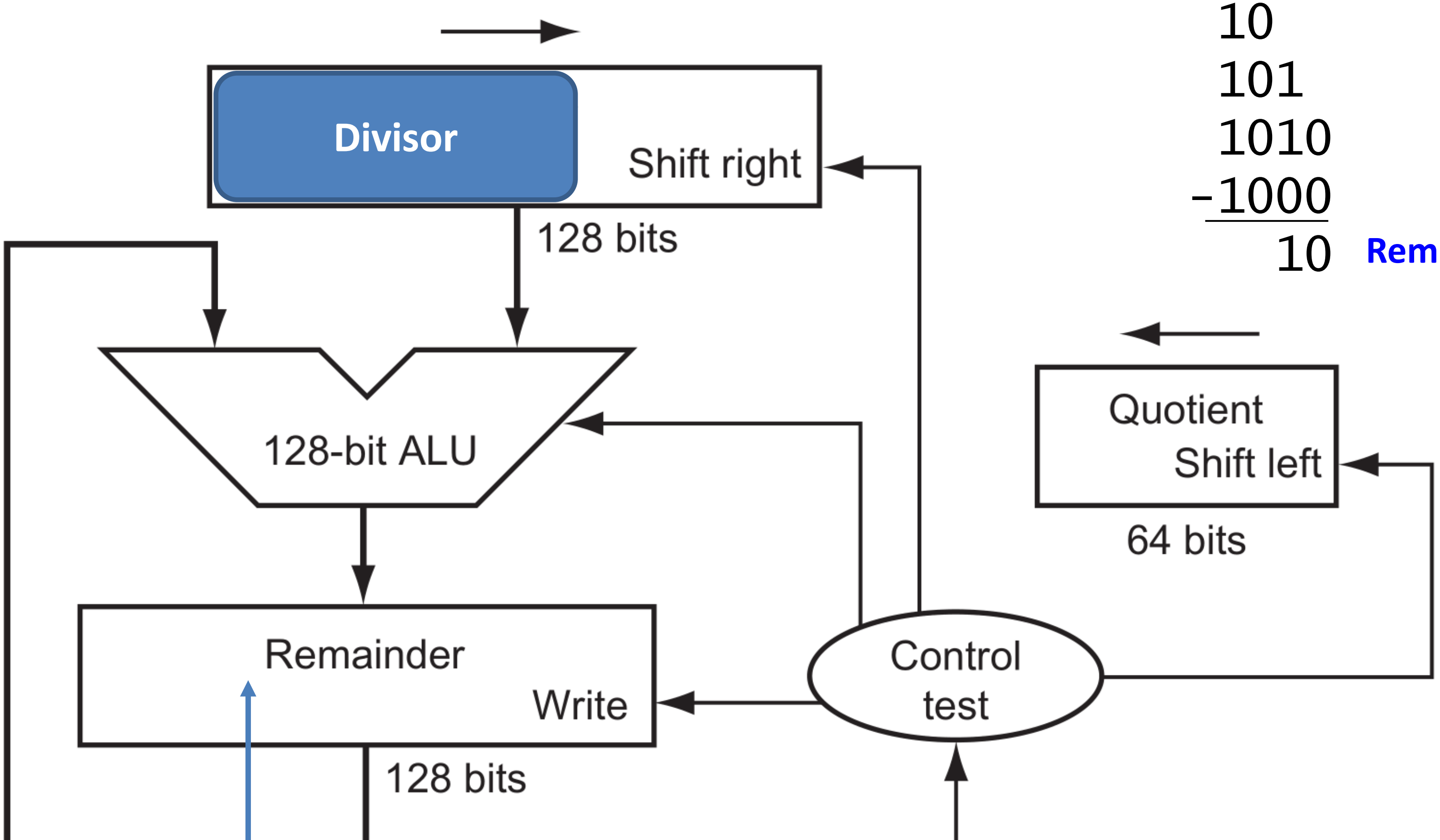
```
                1001      Quotient
Divisor
1000 ) 1001010            Dividend
      -1000
          10
          101
          1010
         -1000
            10            Remainder
```

Dividend=Quotient x Divisor + Remainder

- Algorithm
  - Subtract Divisor from Dividend (=remainder)
    - Result goes into remainder register
  - If remainder>=0, write 1 to quotient
  - If remainder<0, restore by adding divisor to remainder. Write 0 to quotient.
  - Shift divisor right by 1 bit position
  - Repeat this 33 times.

Division Hardware

```
                    1001      Quotient
Divisor   1000 ) 1001010      Dividend
                -1000
                    10
                    101
                    1010
                   -1000
                      10      Remainder
```

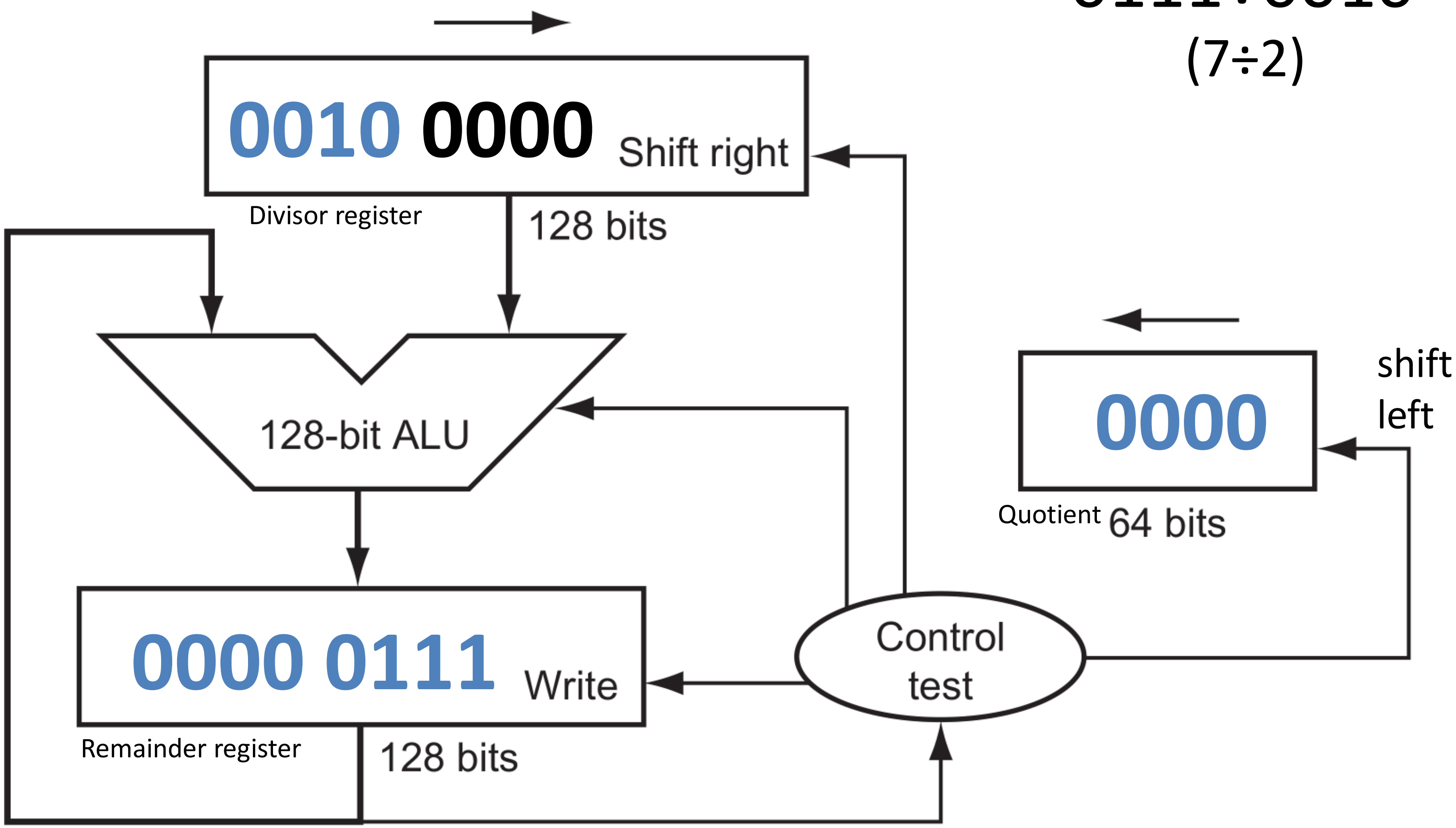


Dividend is place here

Division Hardware

Initialization

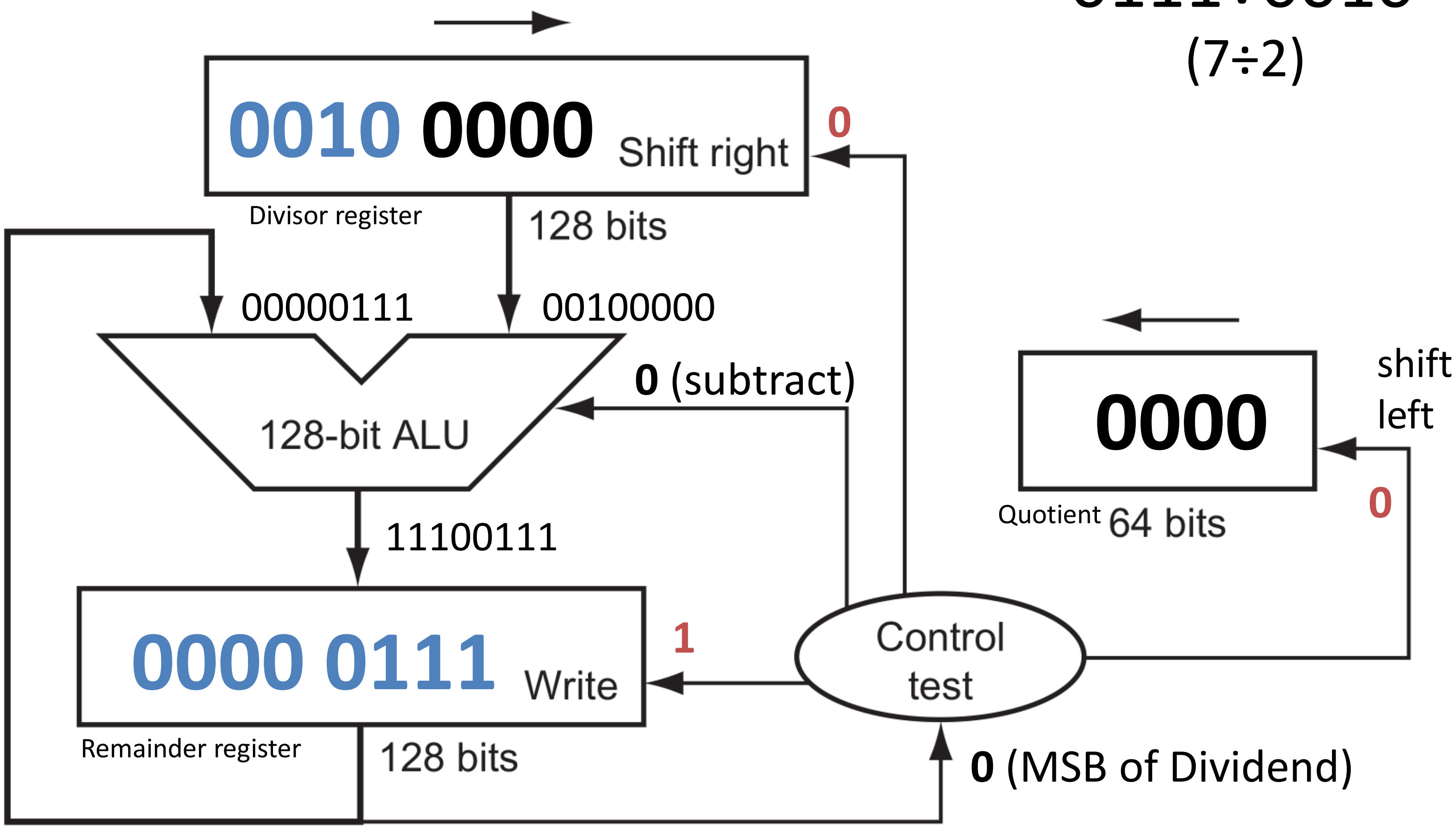
0111÷0010
(7÷2)



Division Hardware

Iteration 1: Control signal set-up

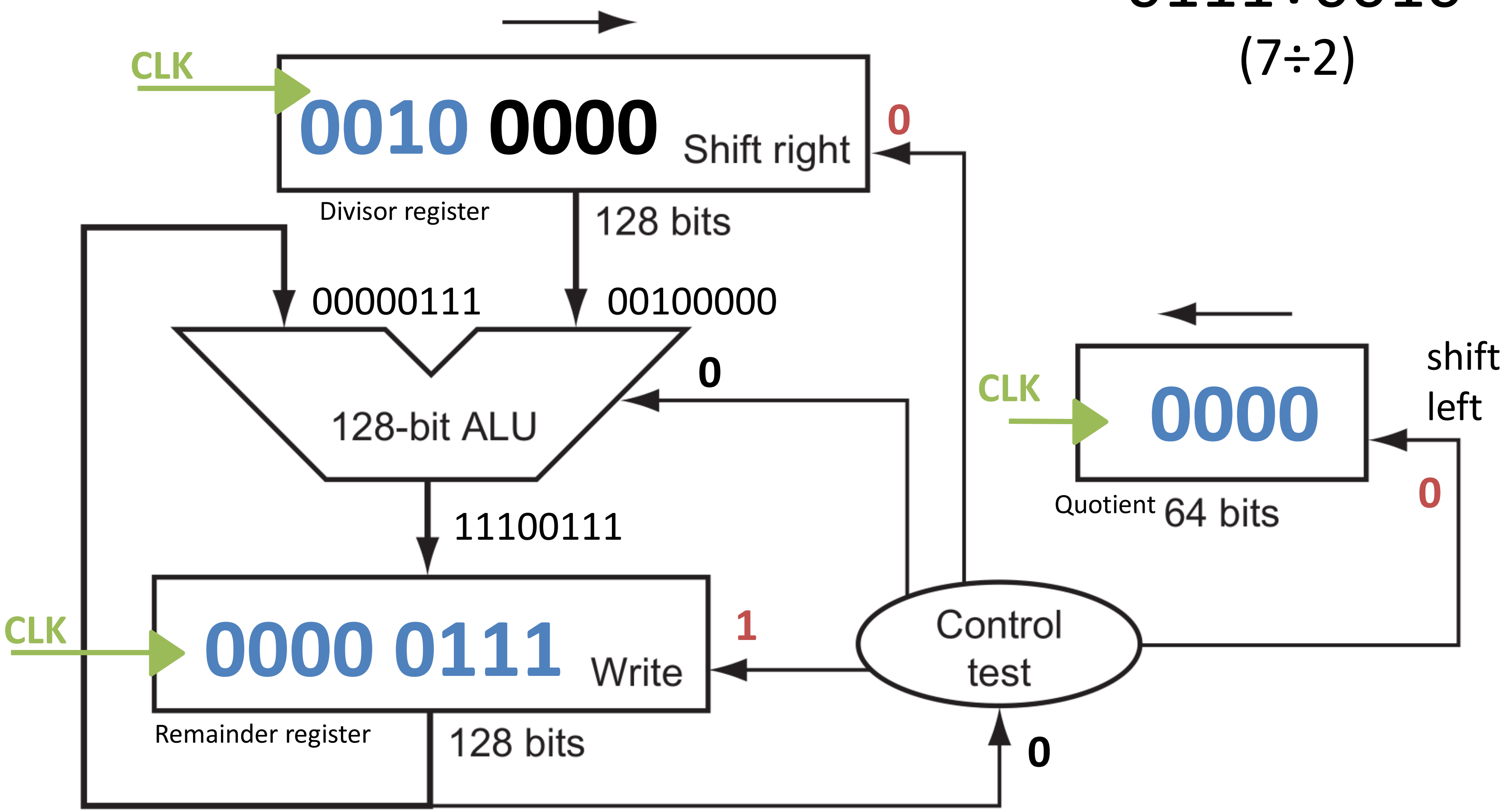
0111÷0010
(7÷2)



Division Hardware

Iteration 1: Clock firing (& Remainder updated)

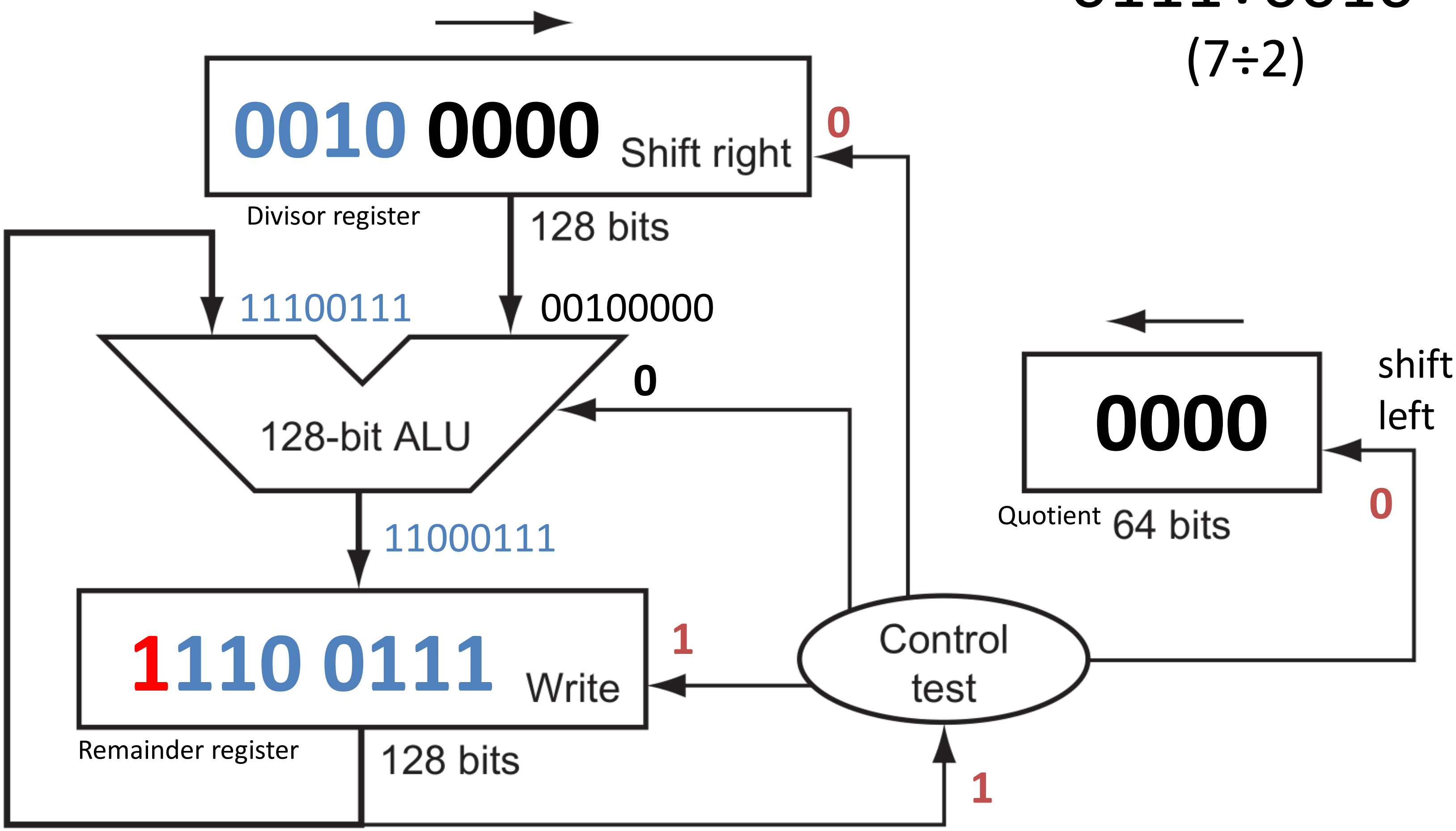
0111÷0010
(7÷2)



Division Hardware

Iteration 1: Test MSB of remainder

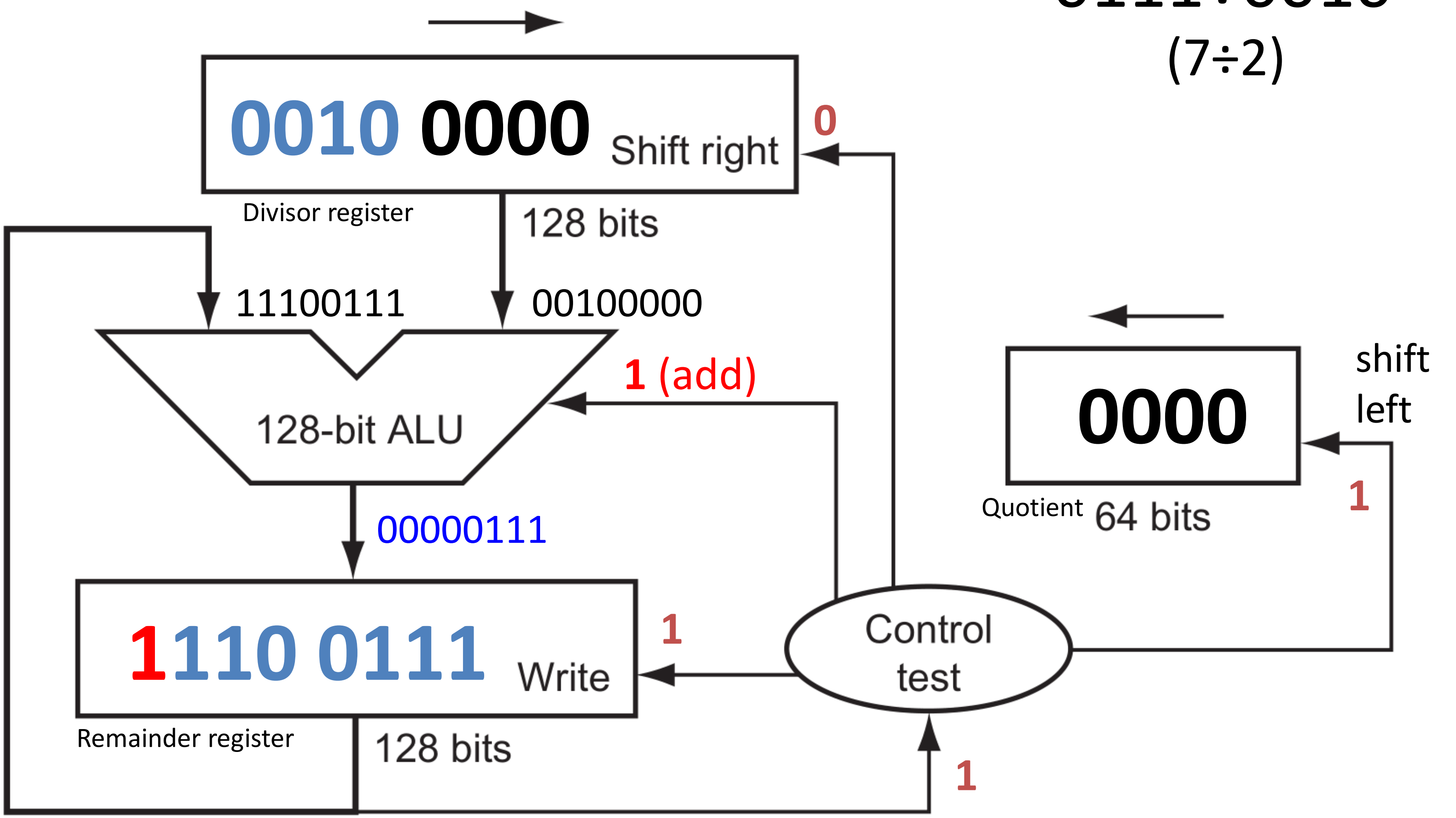
0111÷0010
(7÷2)



Division Hardware

Iteration 1: Control signal for ALU addition ready

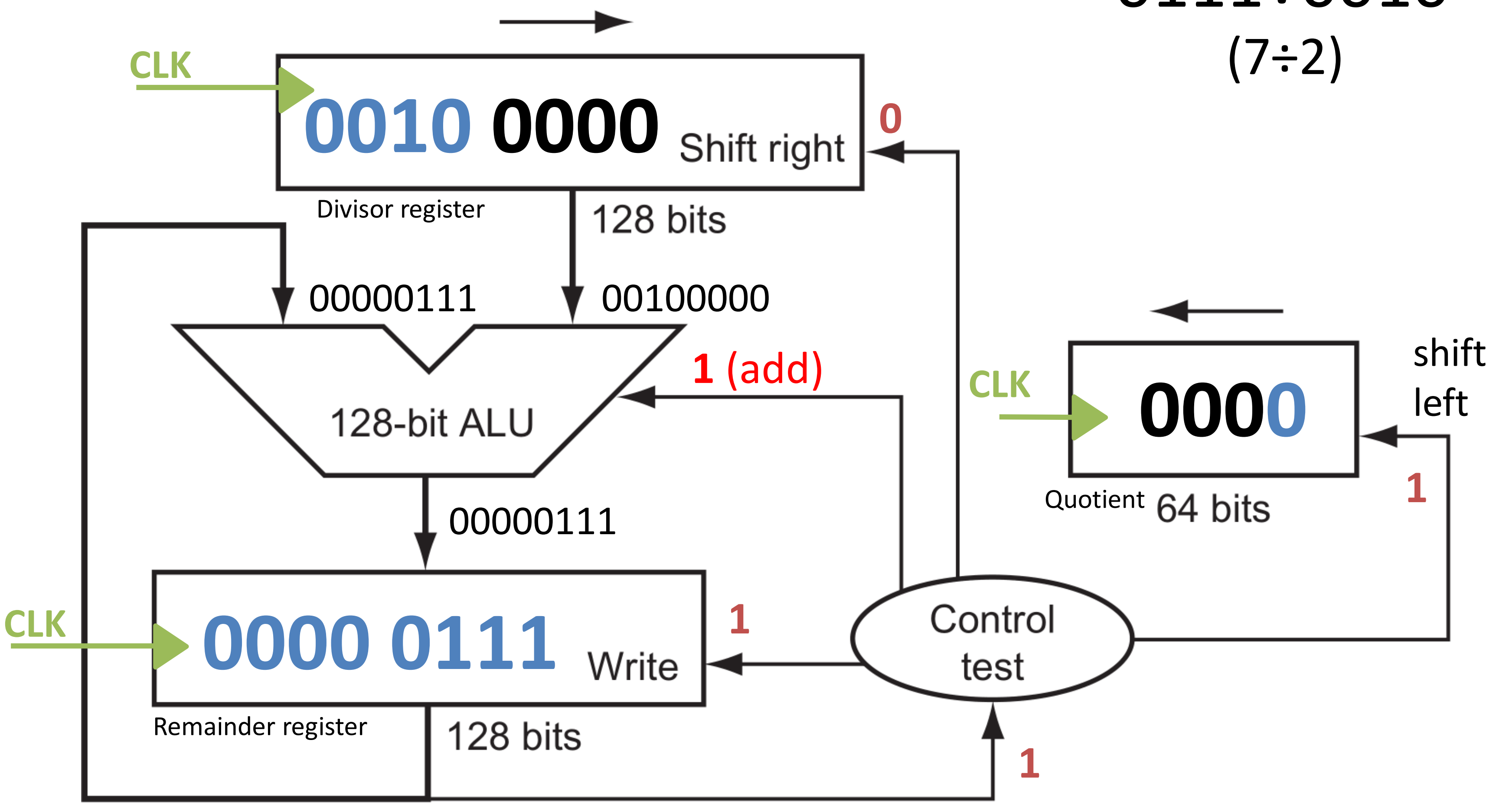
0111÷0010
(7÷2)



Division Hardware

Iteration 1: Clock firing (& Remainder restored)

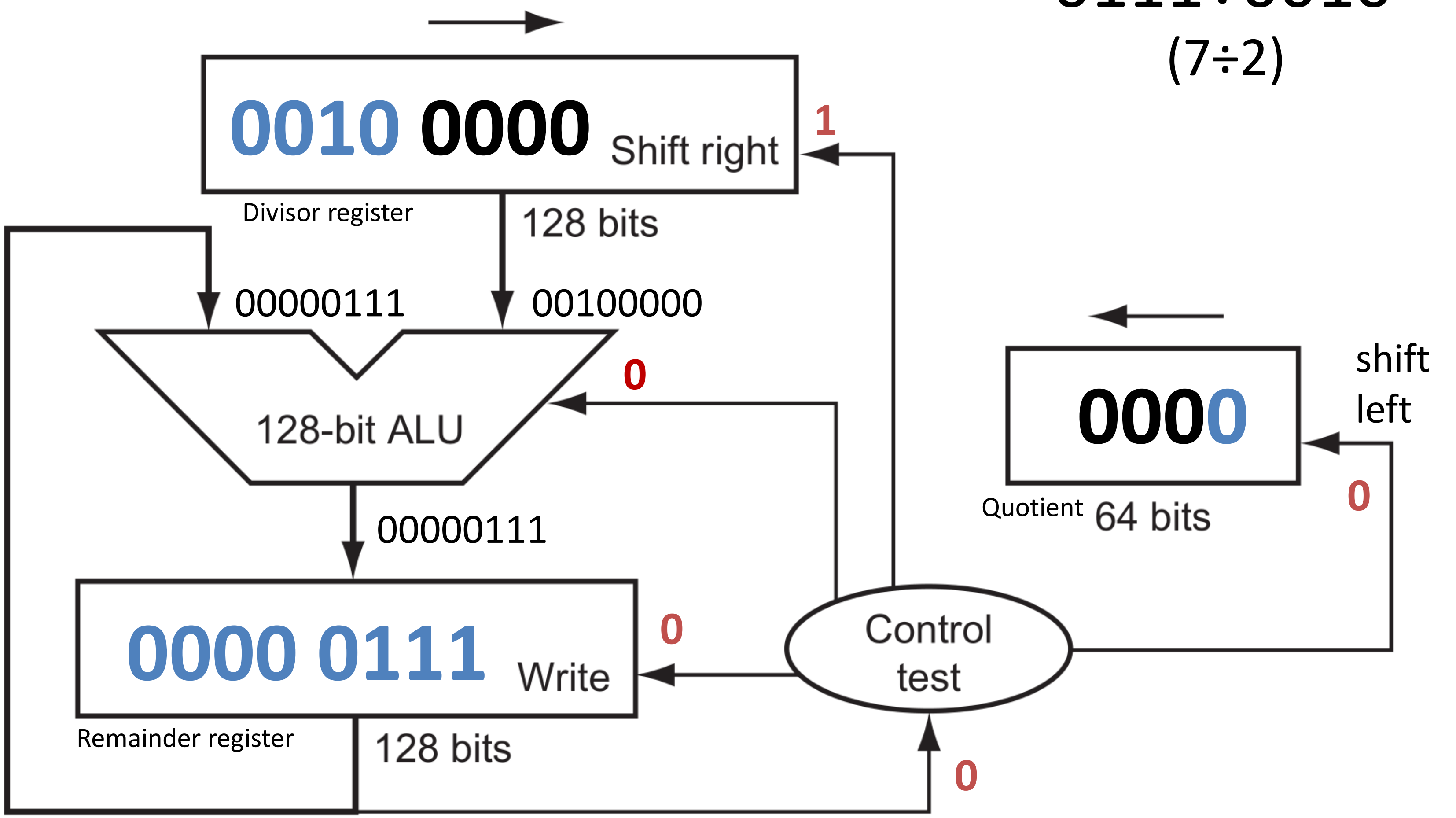
0111÷0010
(7÷2)



Division Hardware

Iteration 1: Control signal for divisor shift ready

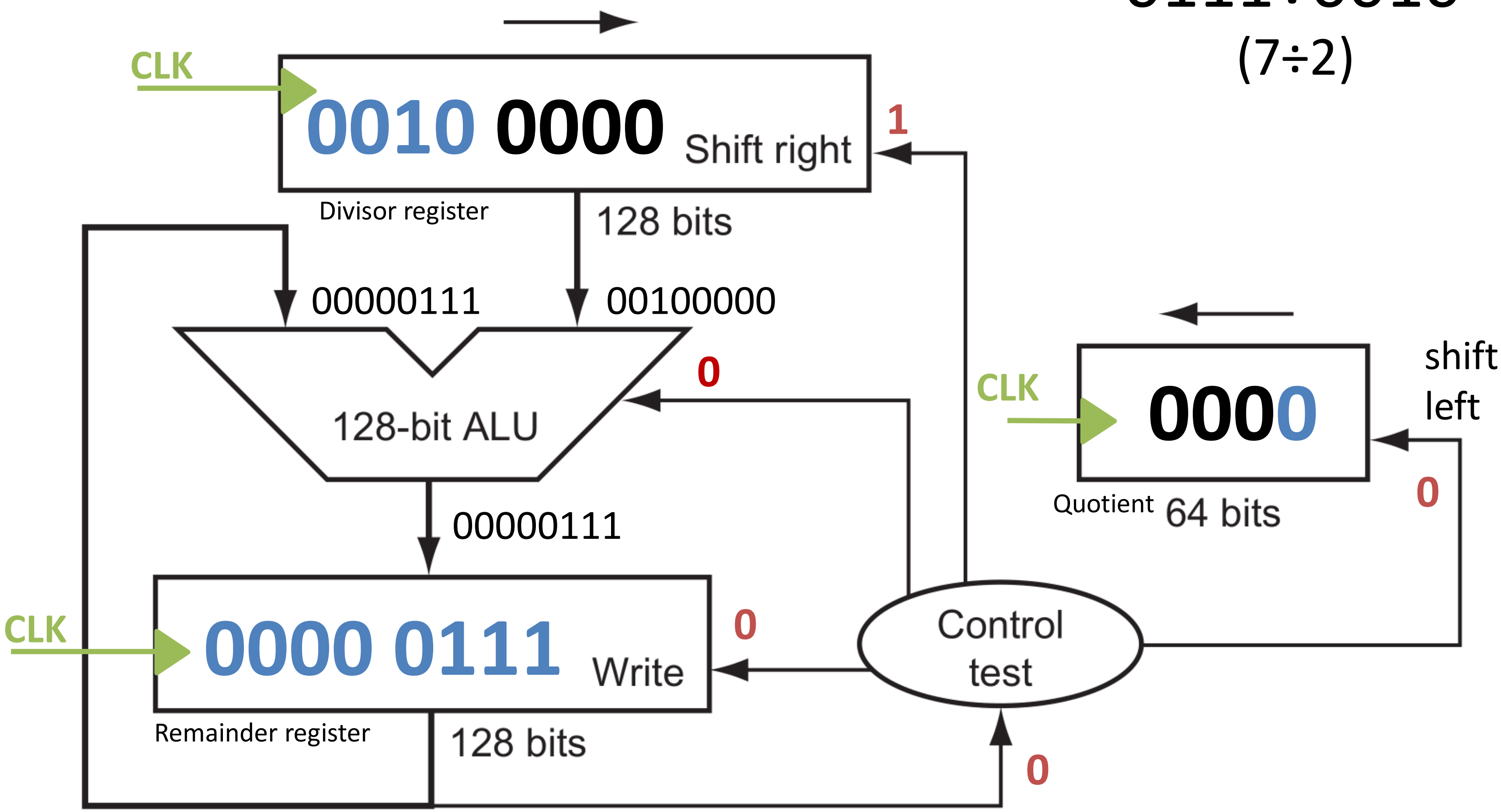
0111÷0010
(7÷2)



Division Hardware

Iteration 1: Clock firing

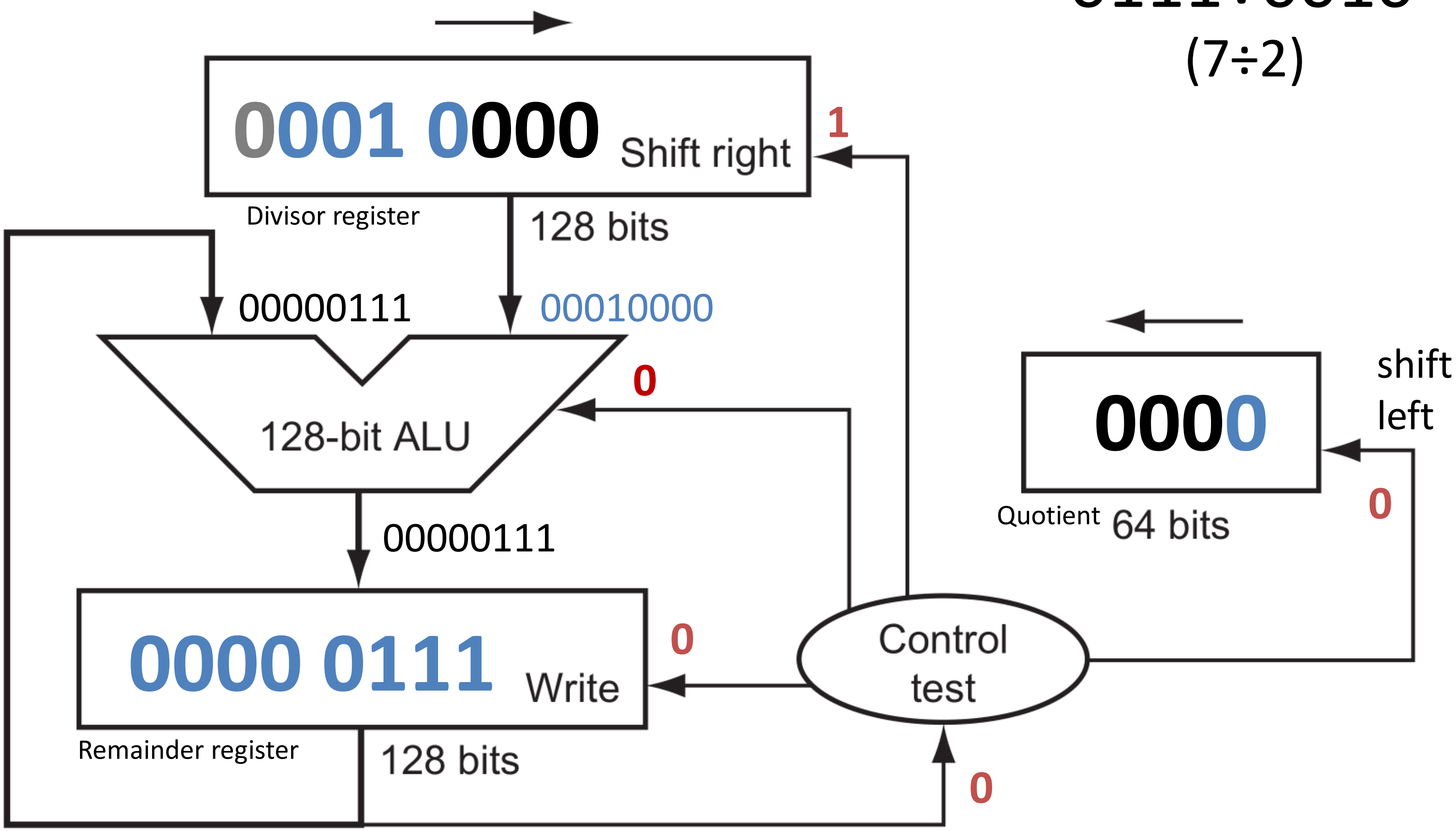
0111÷0010
(7÷2)



Division Hardware

Iteration 1: Divisor shifted

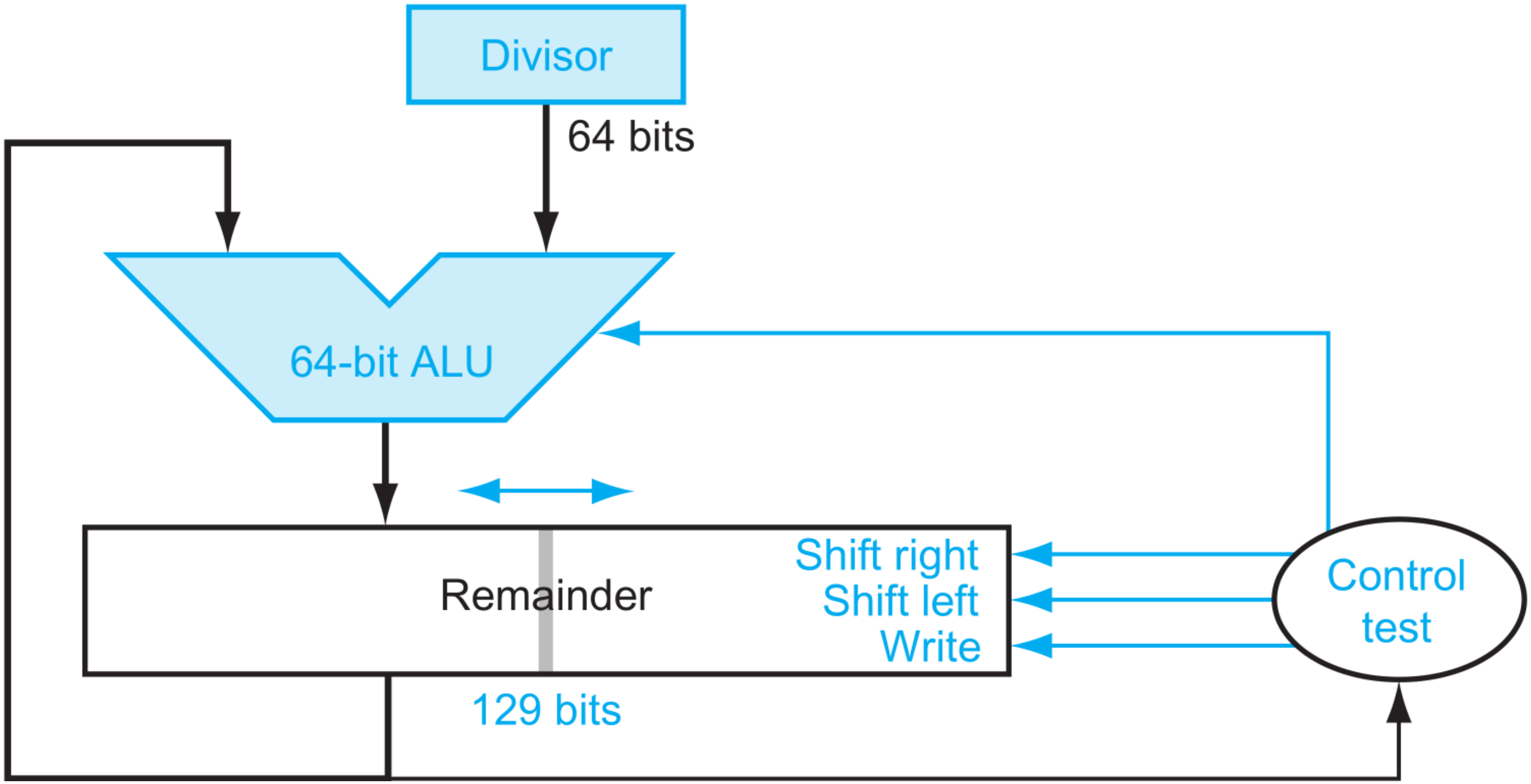
0111÷0010
(7÷2)



Division Example 7÷2 (0111÷0010)

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

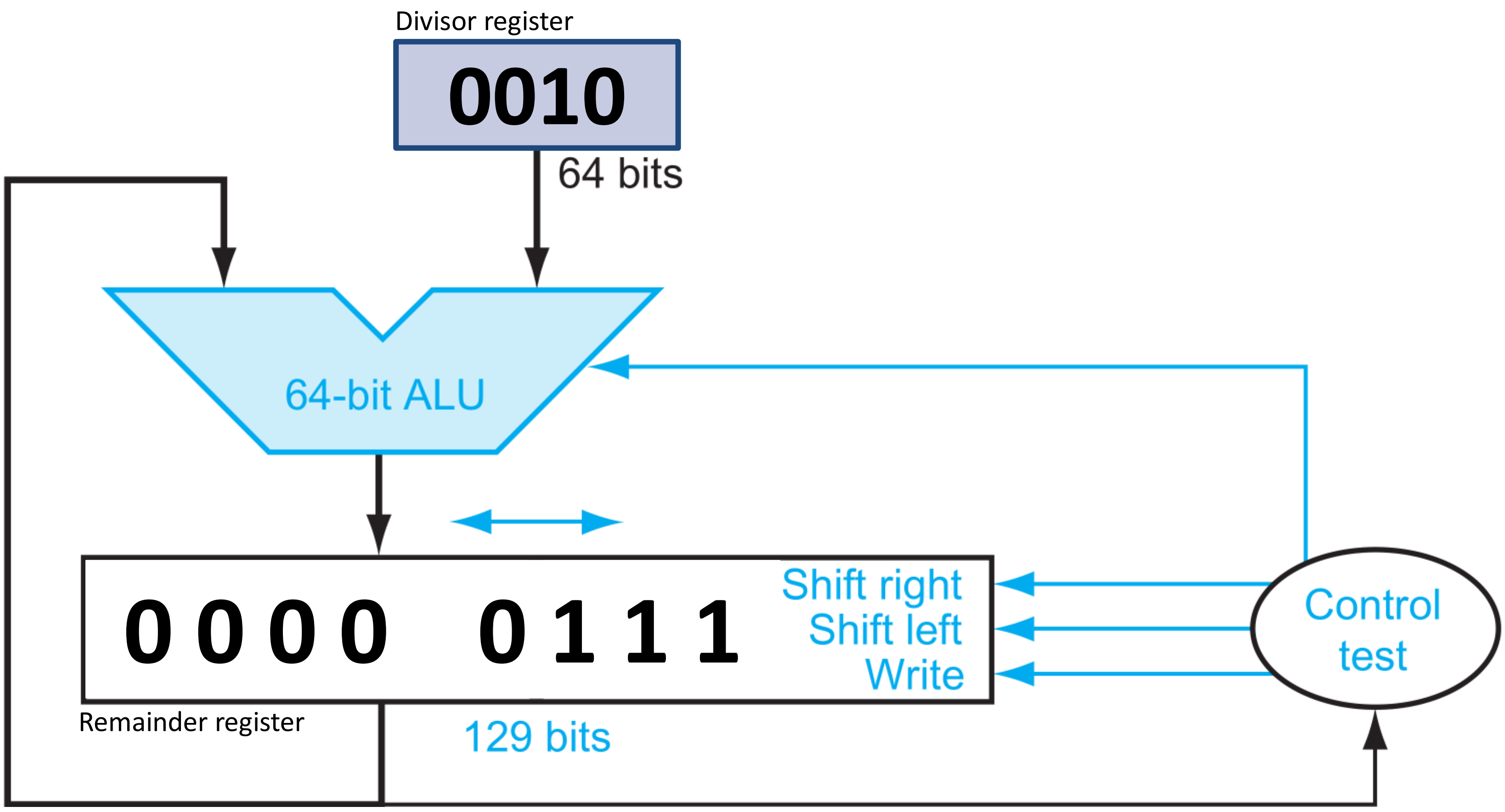
Improved Division Hardware



Improved Division Hardware

Iteration 0

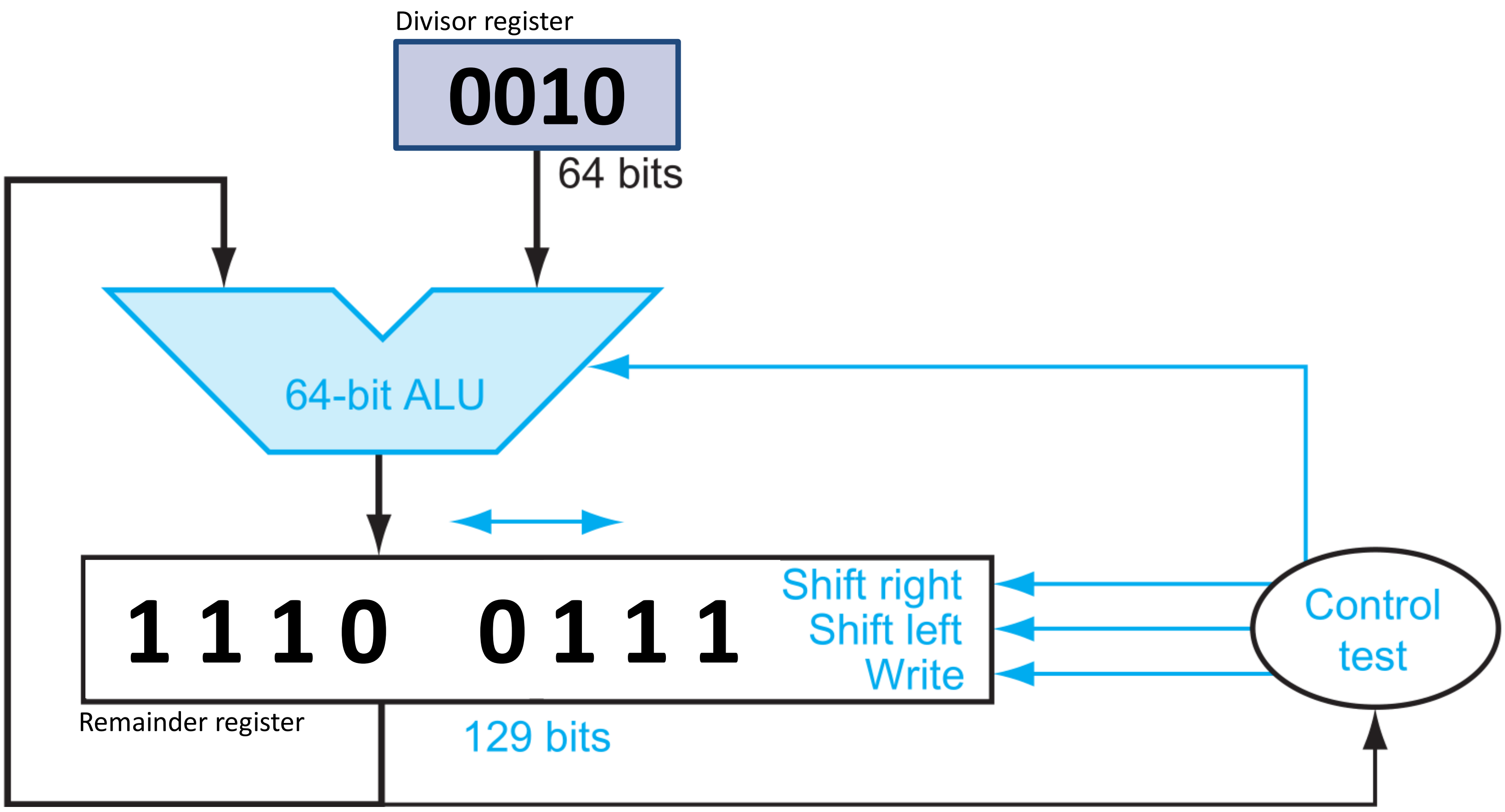
Setting up registers



Improved Division Hardware

Iteration 1

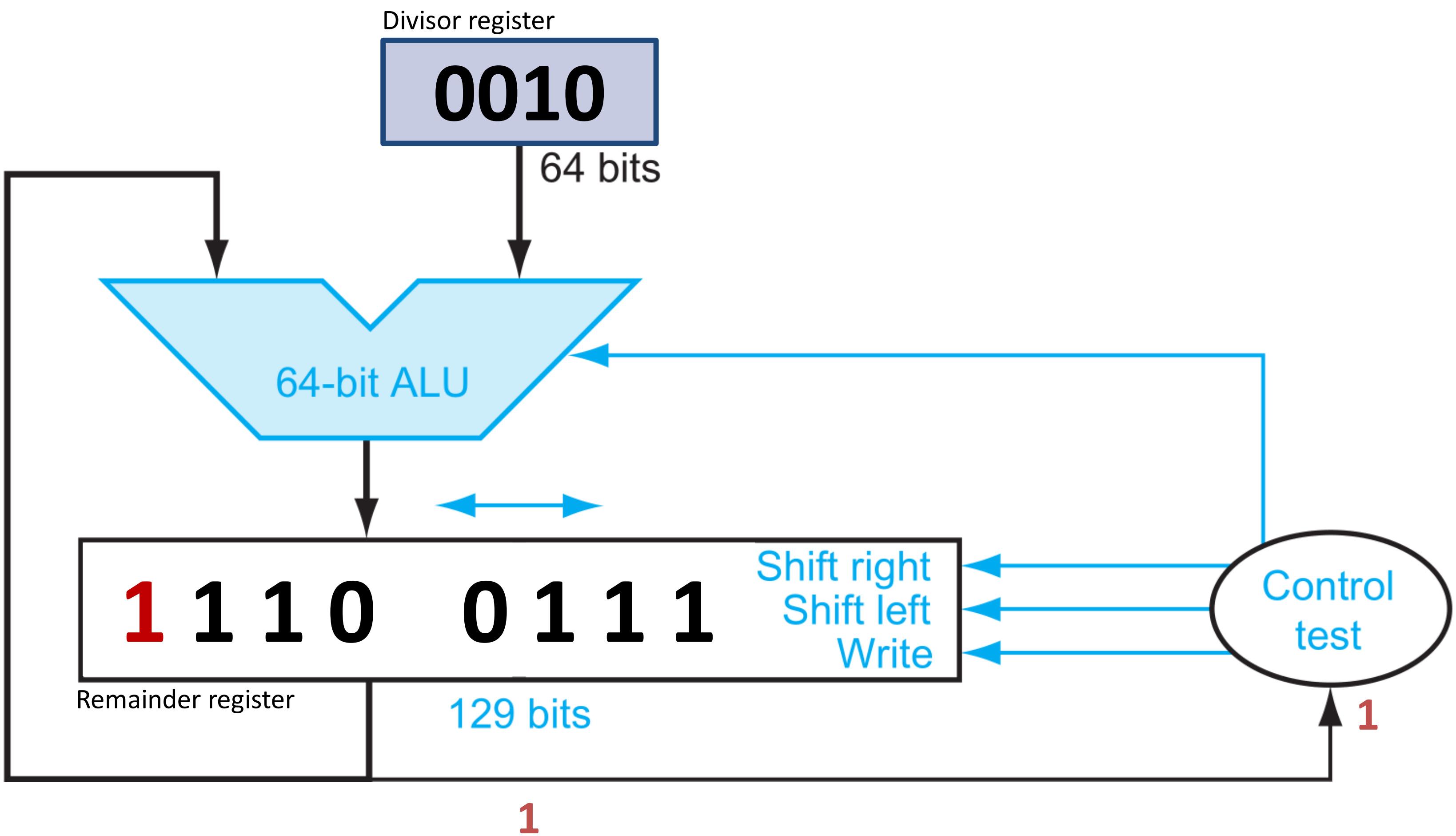
Left of the remainder = Left of the remainder - divisor



Improved Division Hardware

Iteration 1

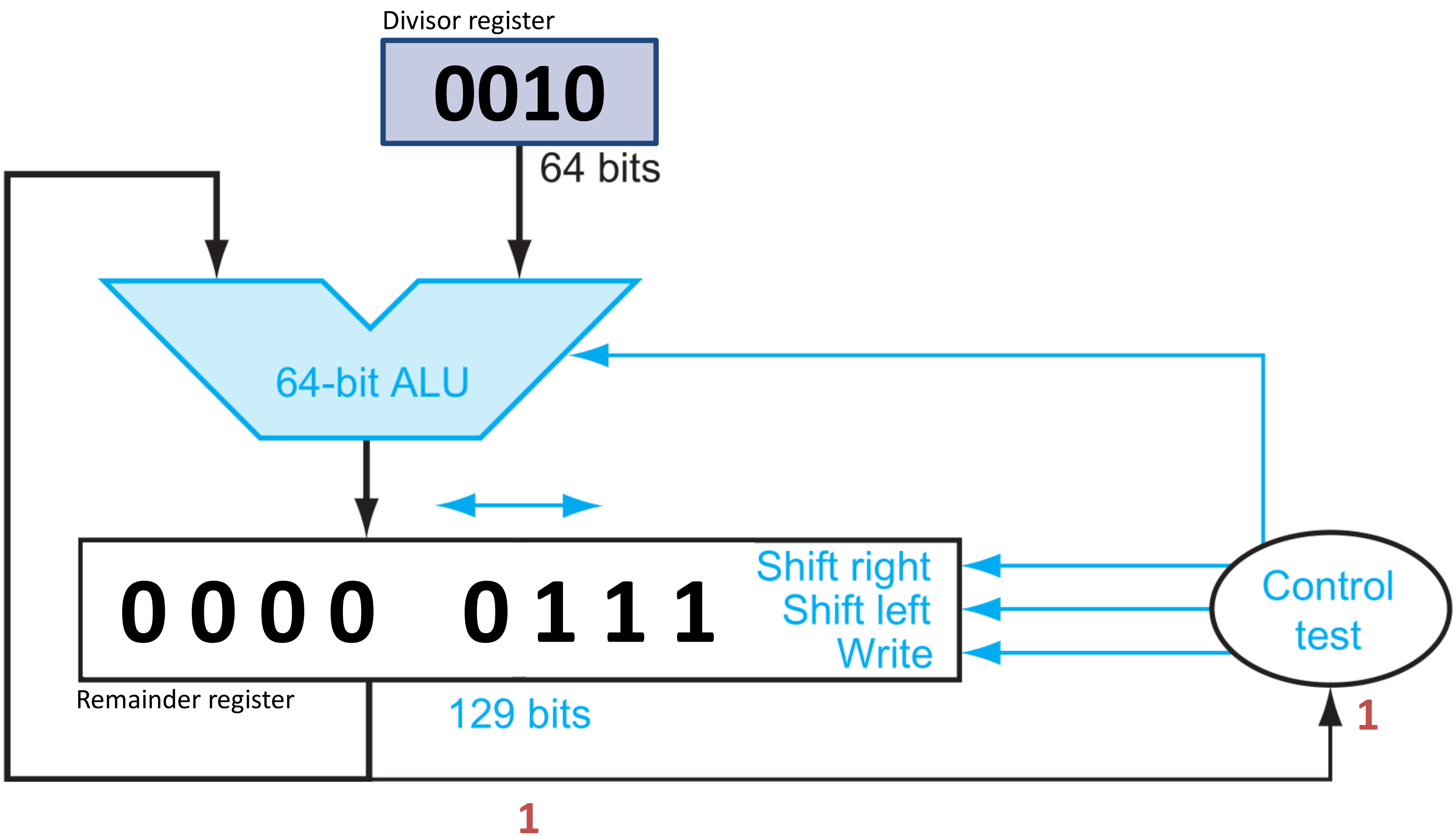
Test MSB to check if it is 1



Improved Division Hardware

Iteration 1

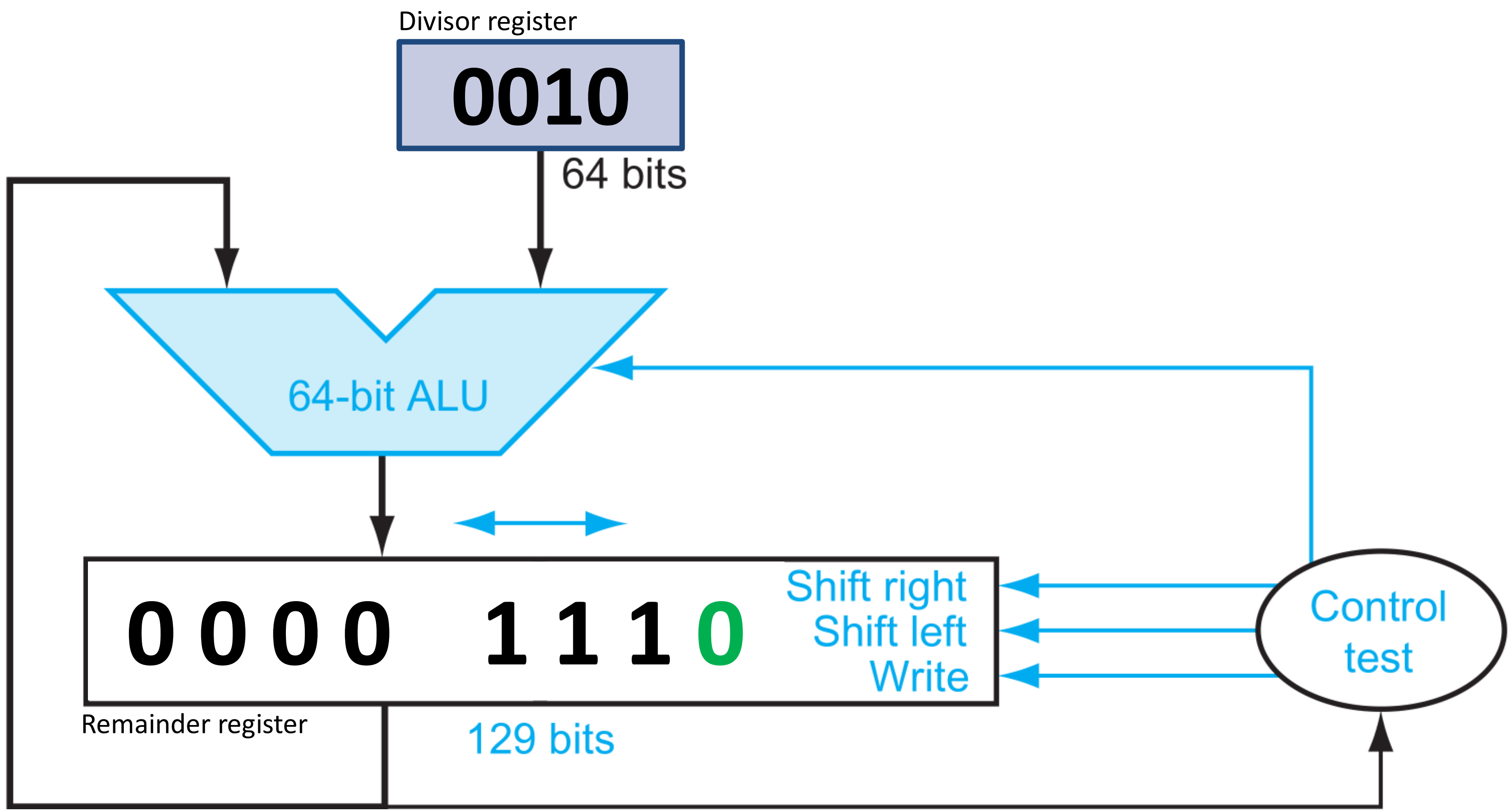
Restore by adding the divisor back to the remainder



Improved Division Hardware

Iteration 1

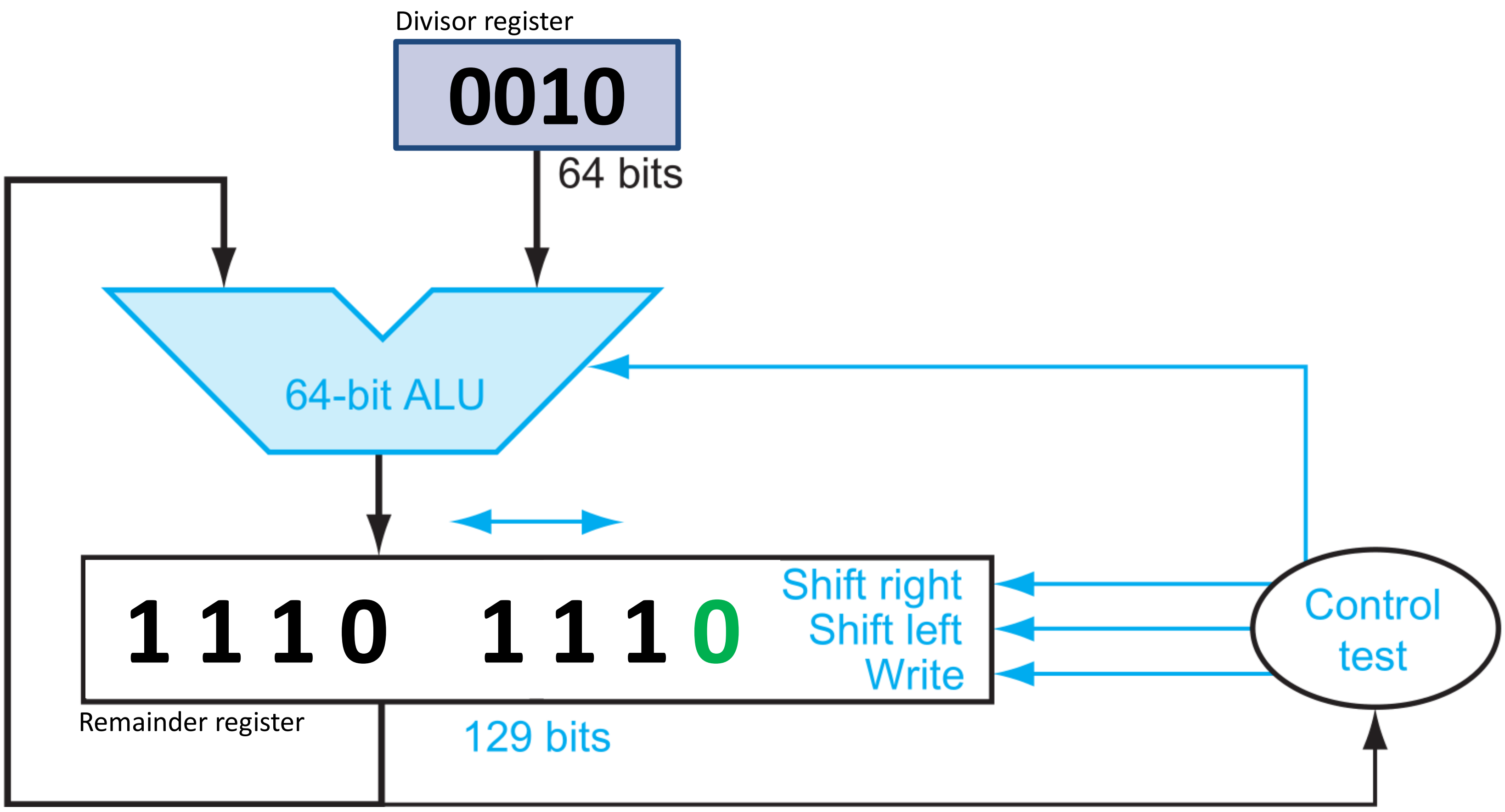
Shift left the remainder



Improved Division Hardware

Iteration 2

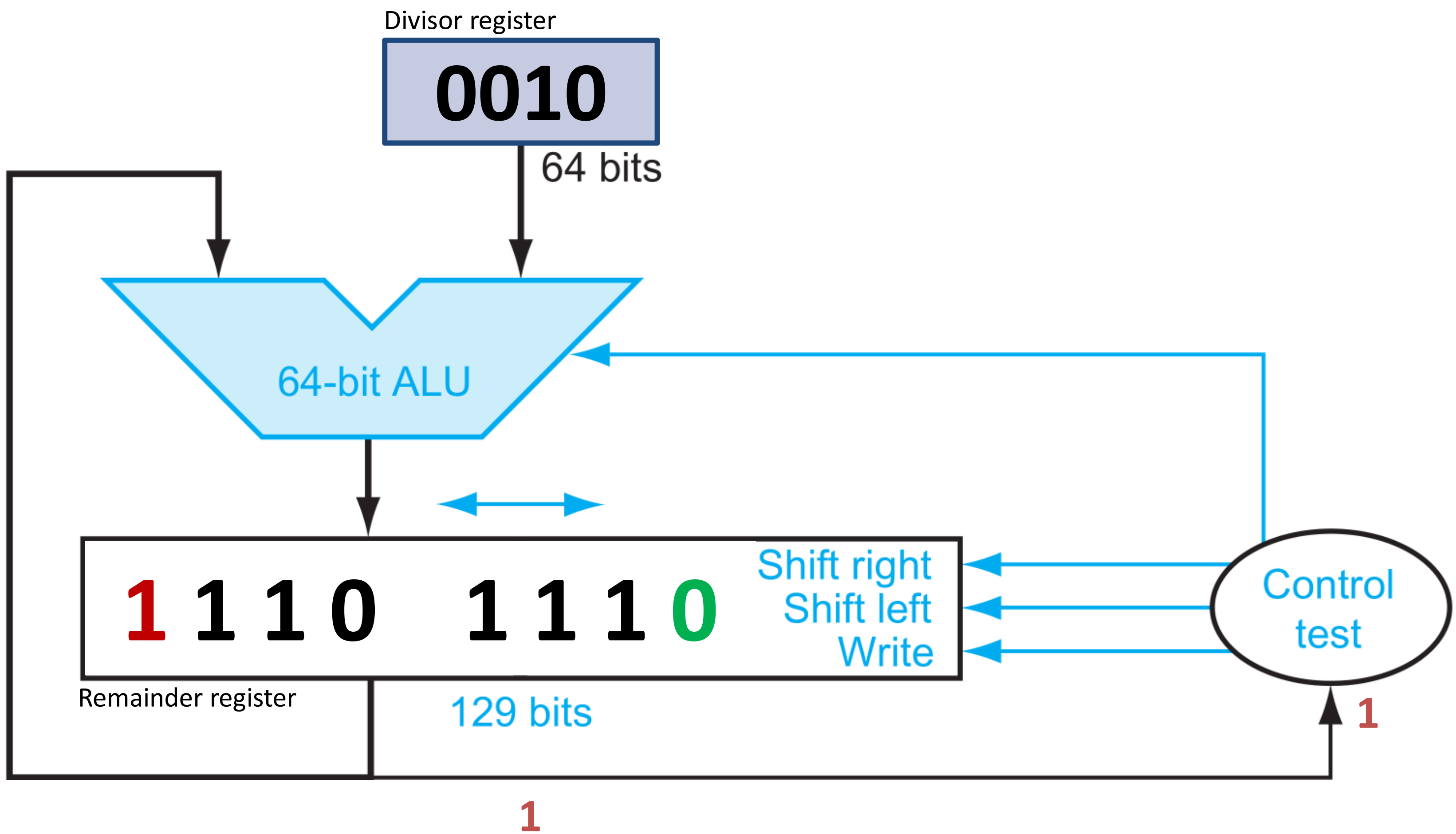
Left of the remainder = Left of the remainder - divisor



Improved Division Hardware

Iteration 2

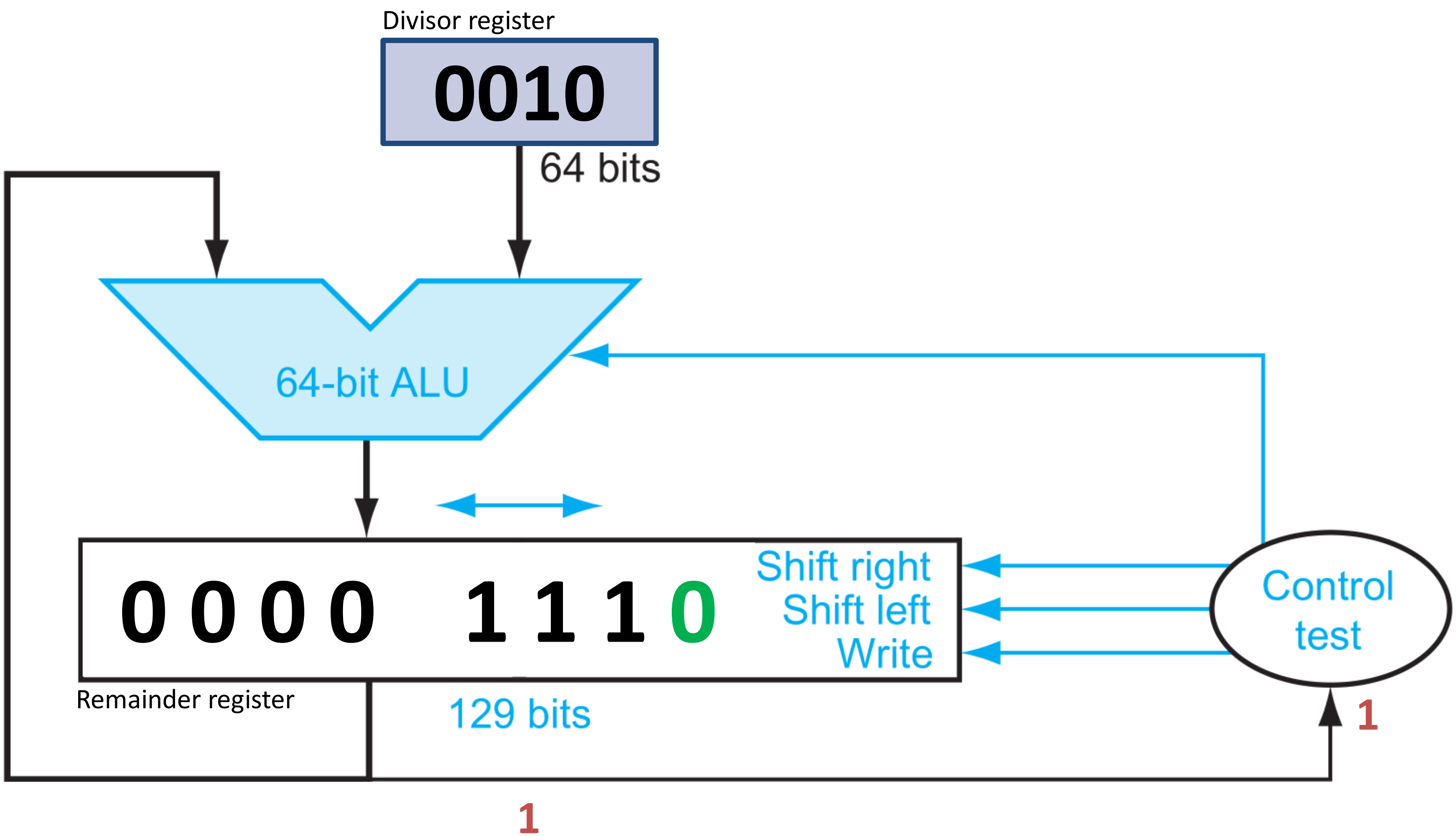
Test MSB to check if it is 1



Improved Division Hardware

Iteration 2

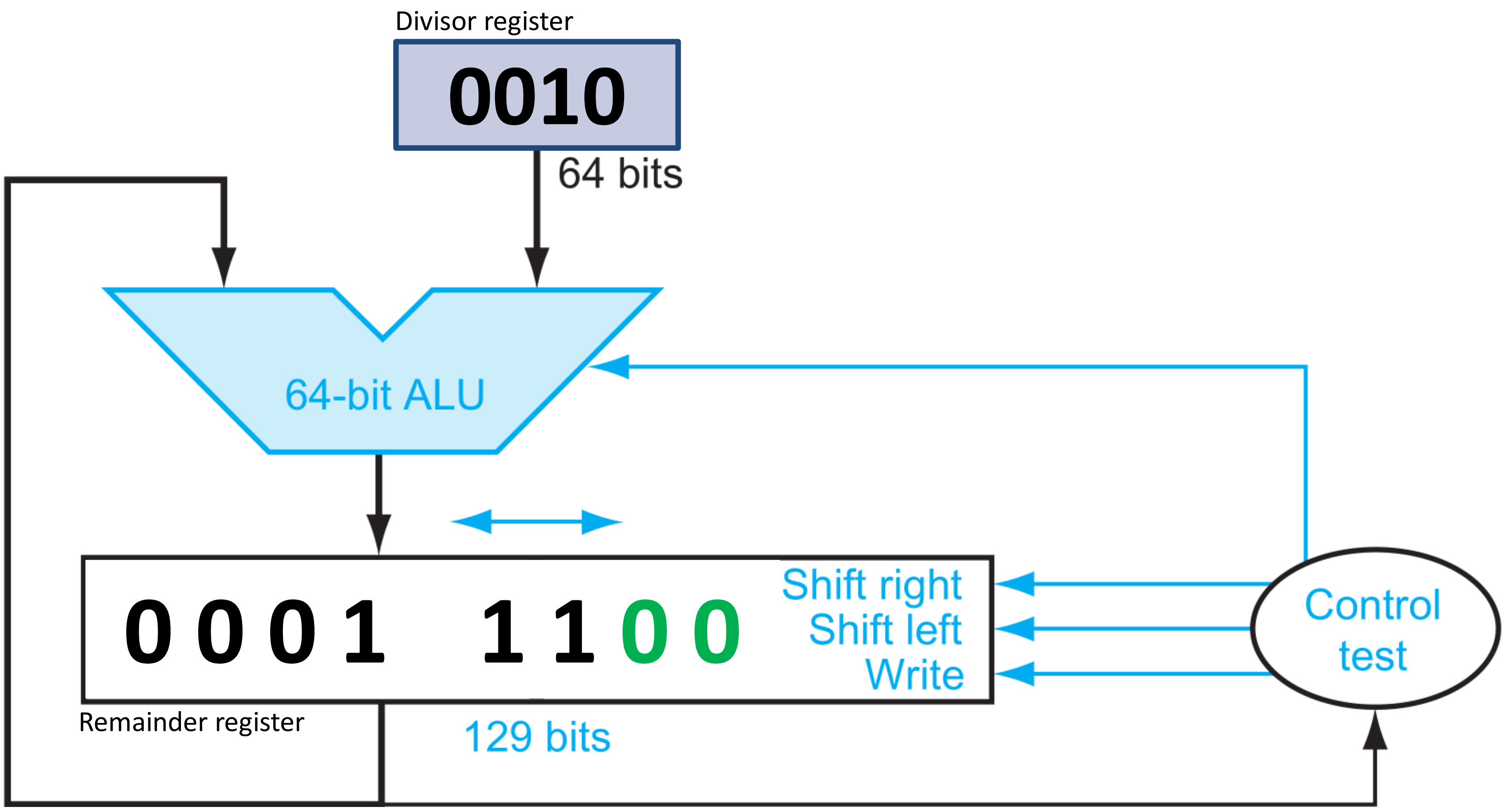
Restore by adding the divisor back to the remainder



Improved Division Hardware

Iteration 2

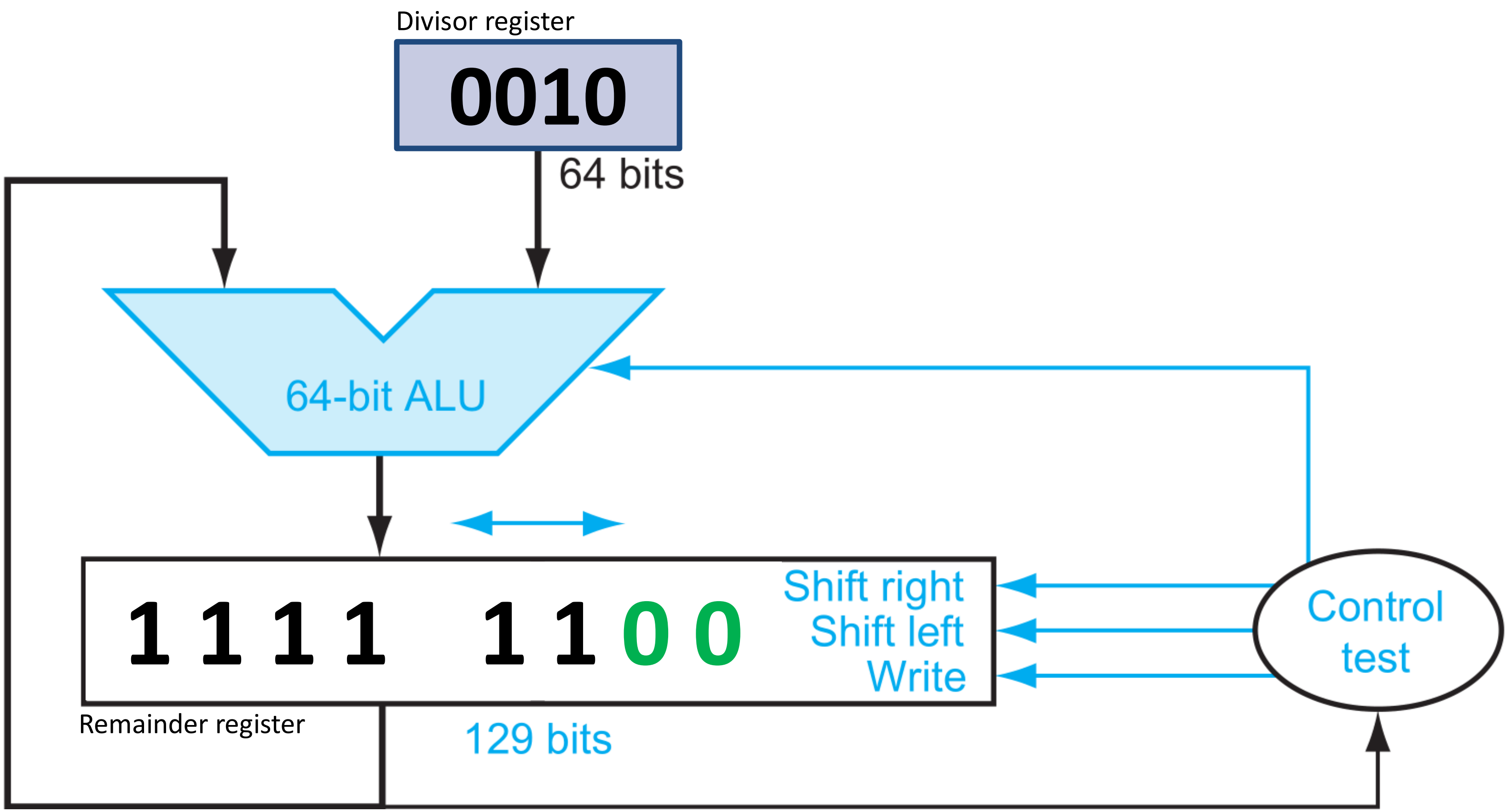
Shift left the remainder



Improved Division Hardware

Iteration 3

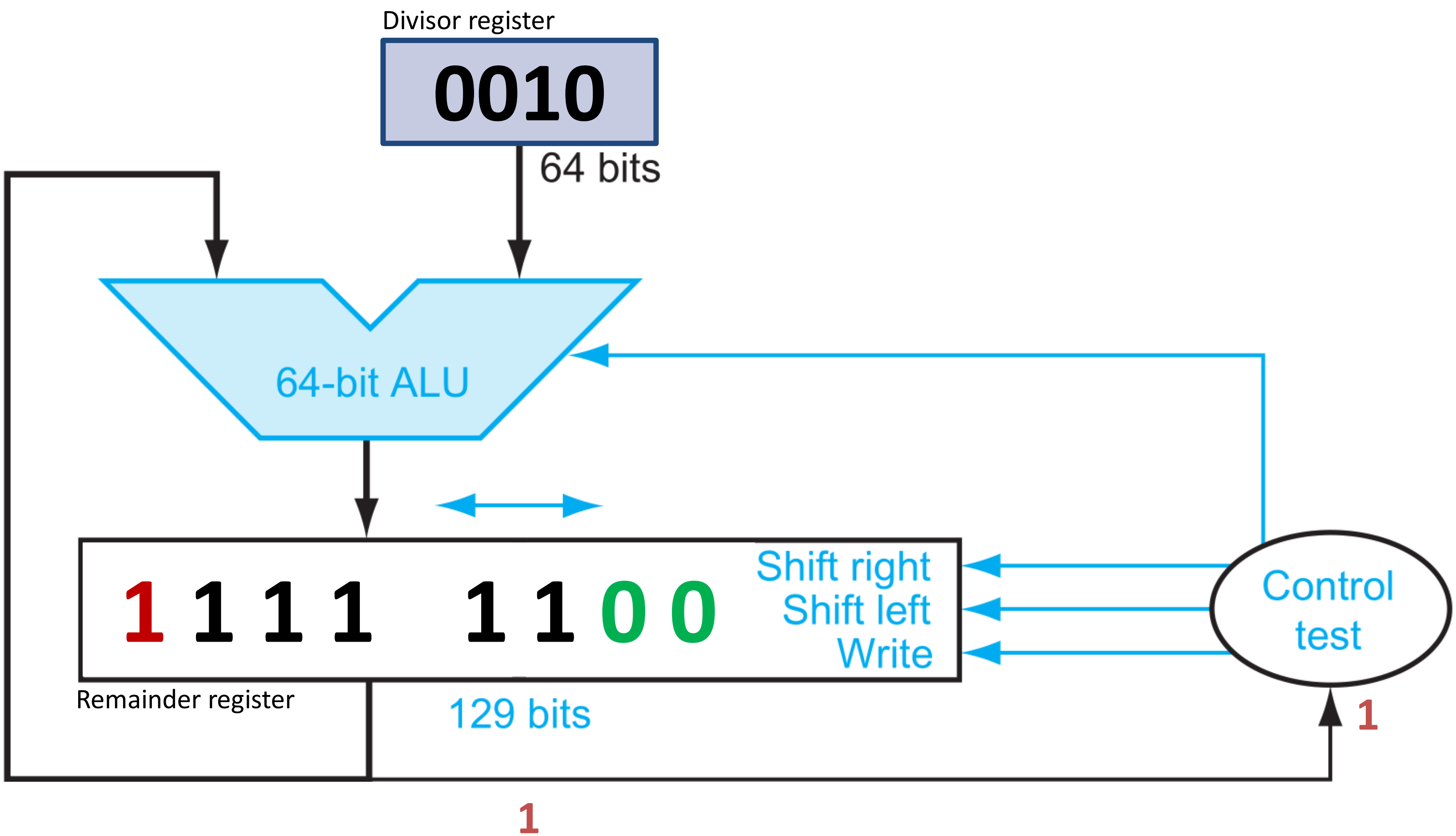
Left of the remainder = Left of the remainder - divisor



Improved Division Hardware

Iteration 3

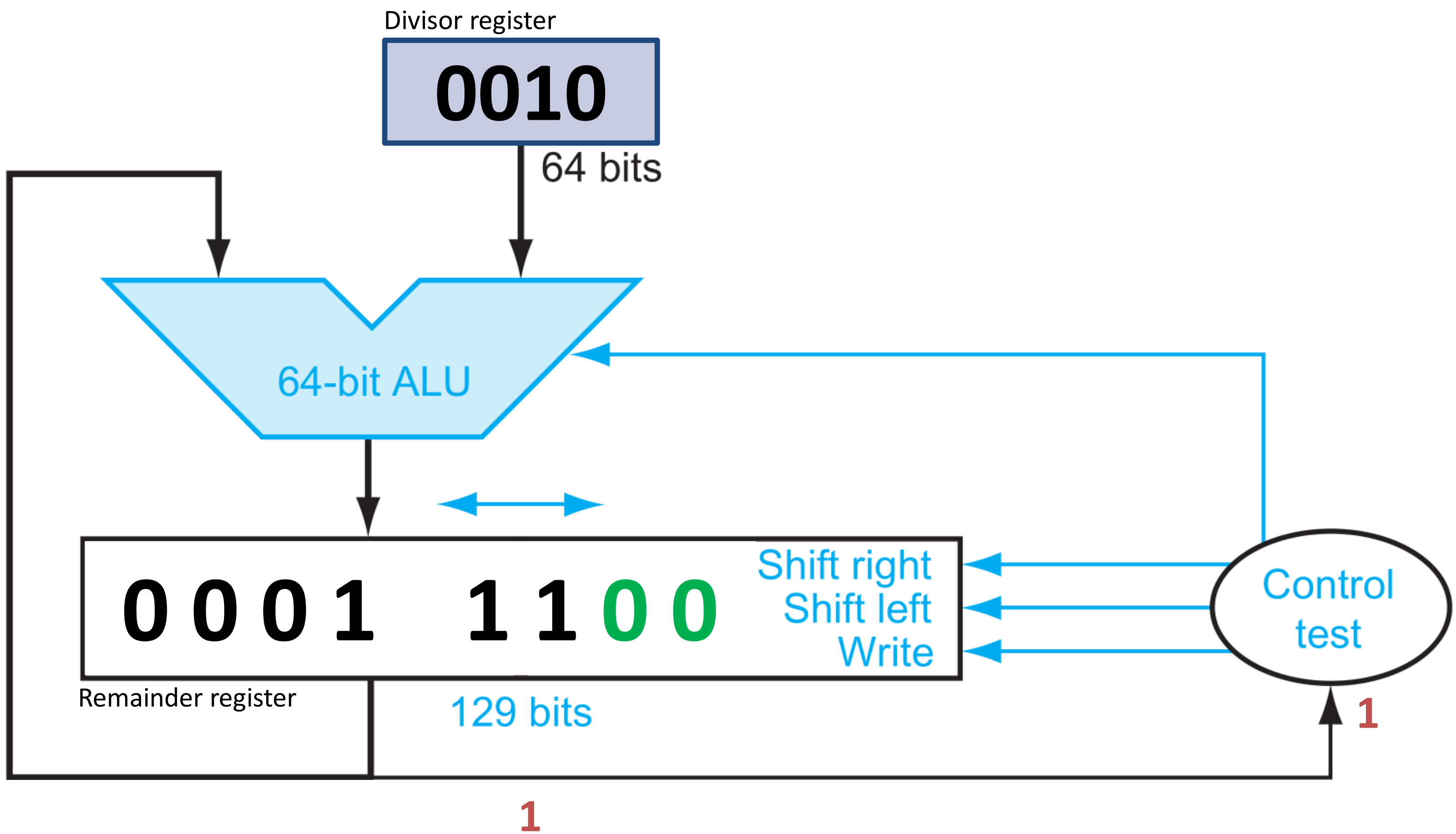
Test MSB to check if it is 1



Improved Division Hardware

Iteration 3

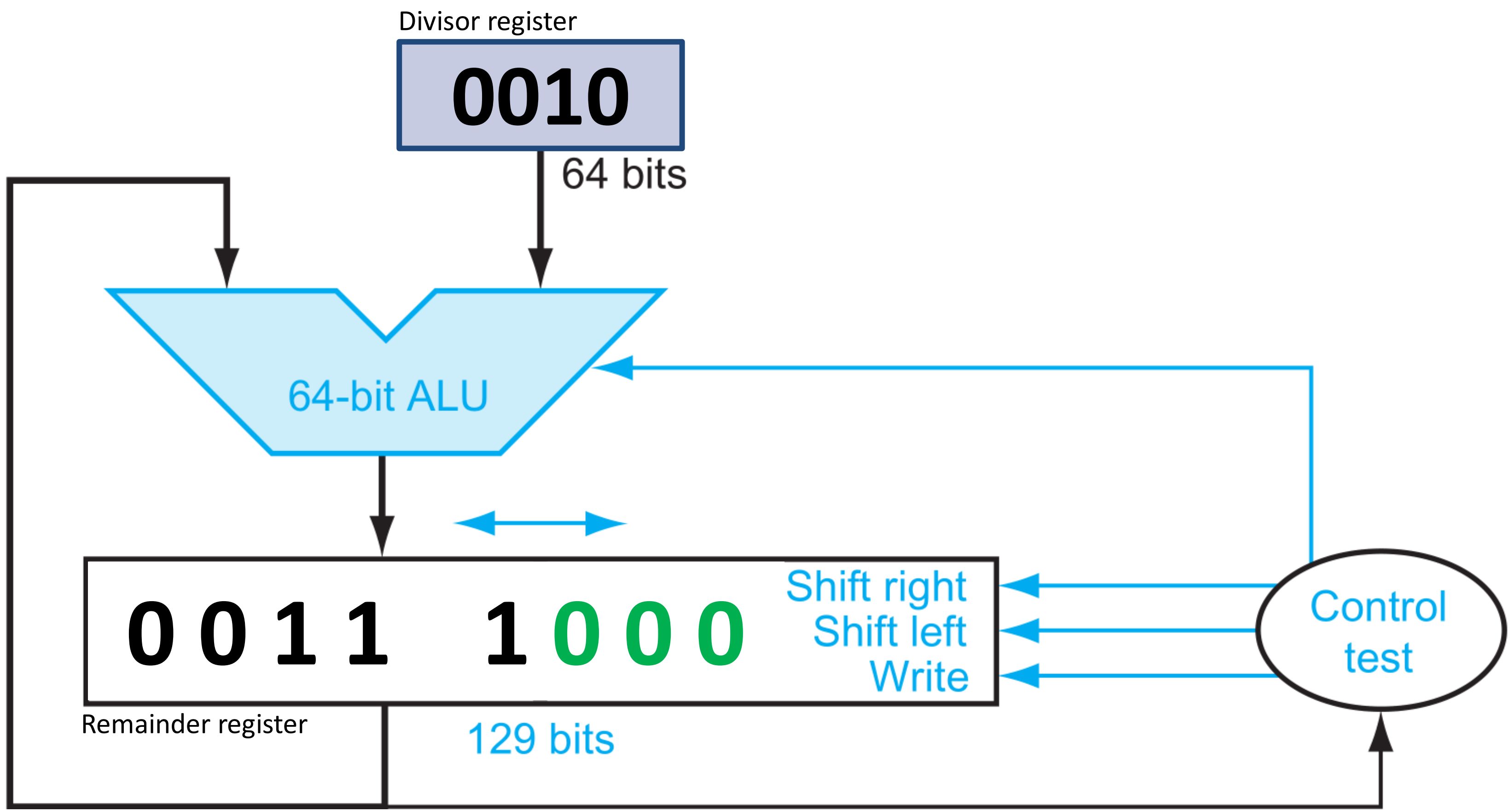
Restore by adding the divisor back to the remainder



Improved Division Hardware

Iteration 3

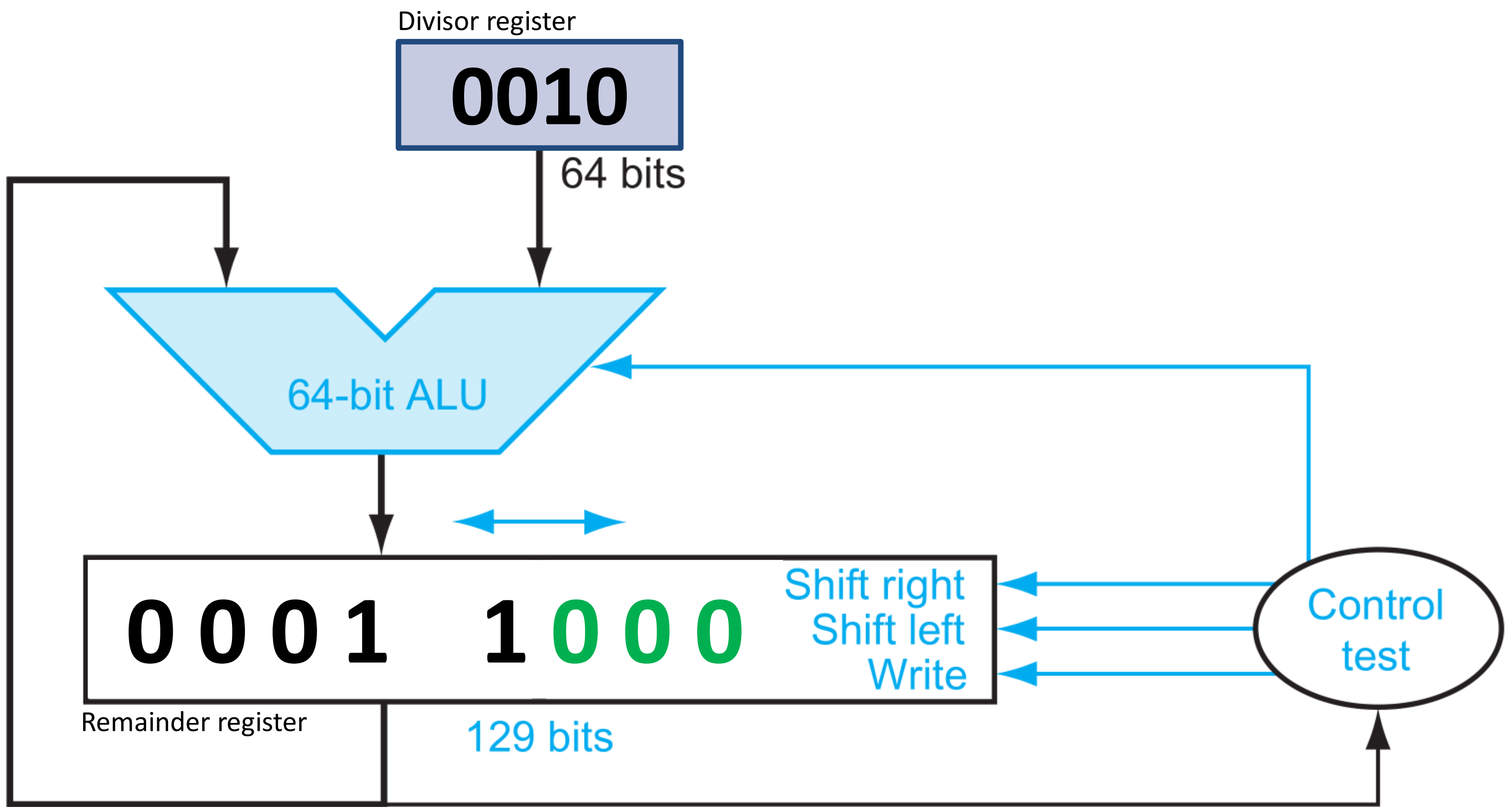
Shift left the remainder



Improved Division Hardware

Iteration 4

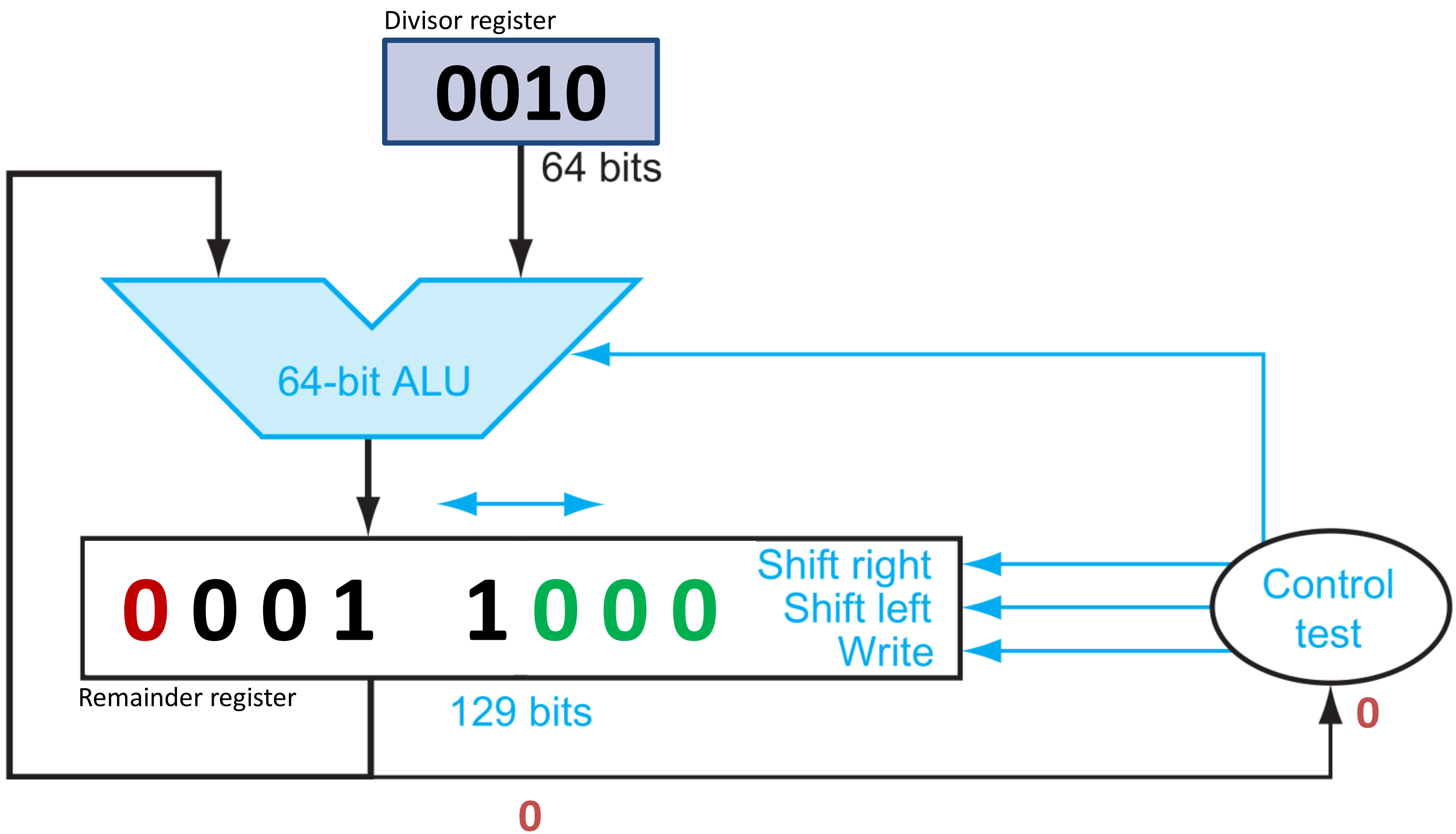
Left of the remainder = Left of the remainder - divisor



Improved Division Hardware

Iteration 4

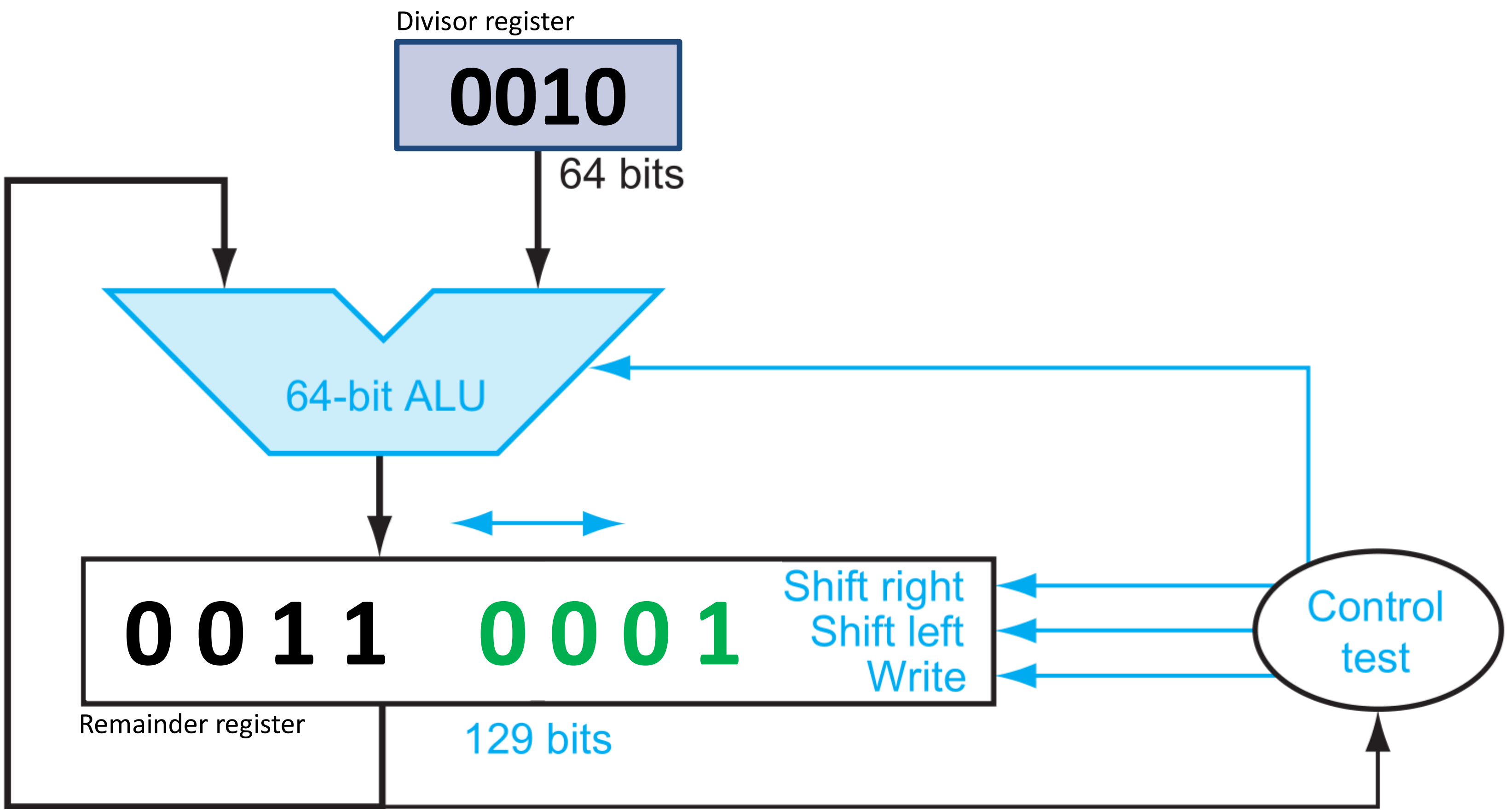
Test MSB to check if it is 1



Improved Division Hardware

Iteration 4

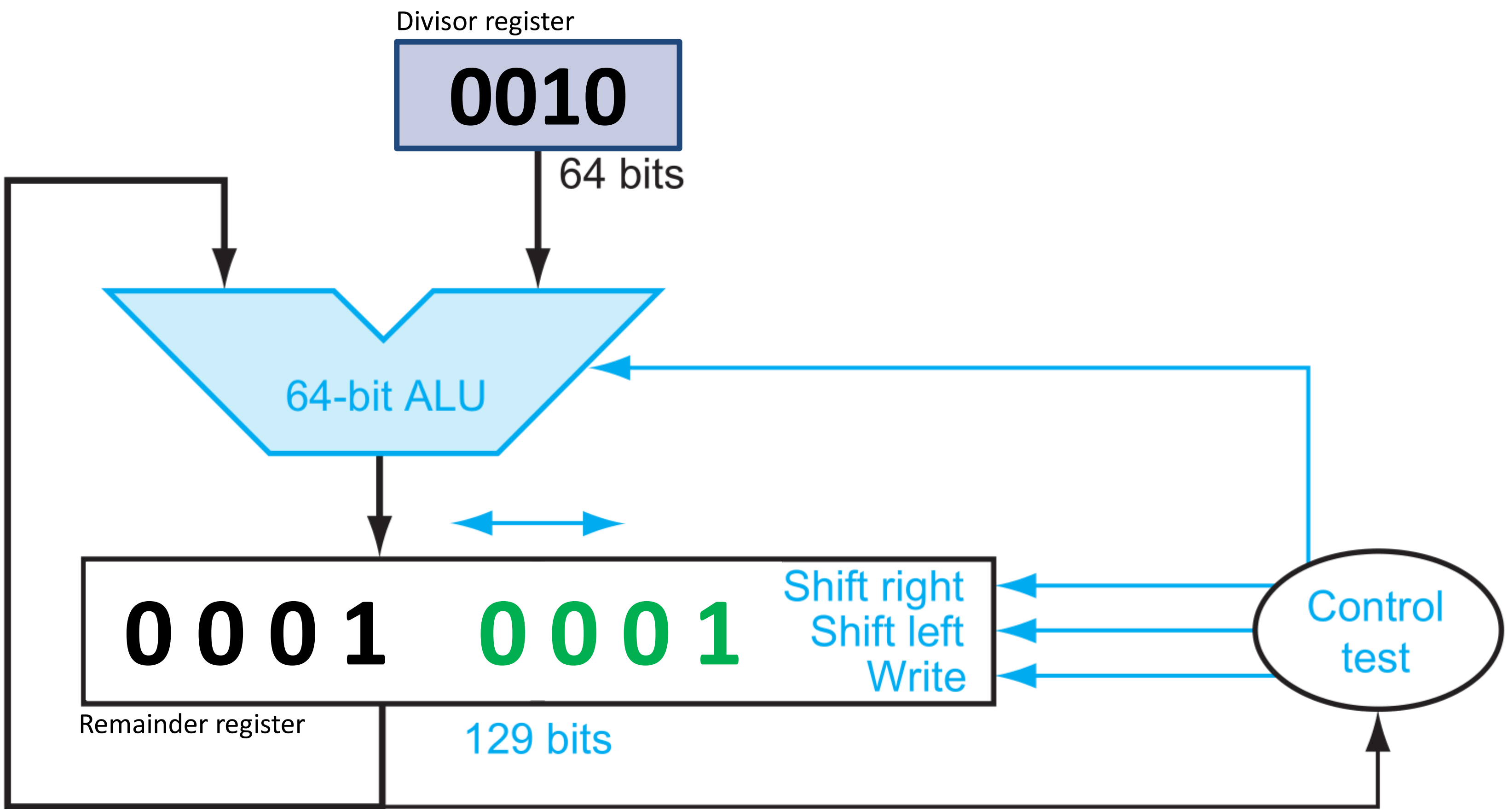
Shift left the remainder



Improved Division Hardware

Iteration 5

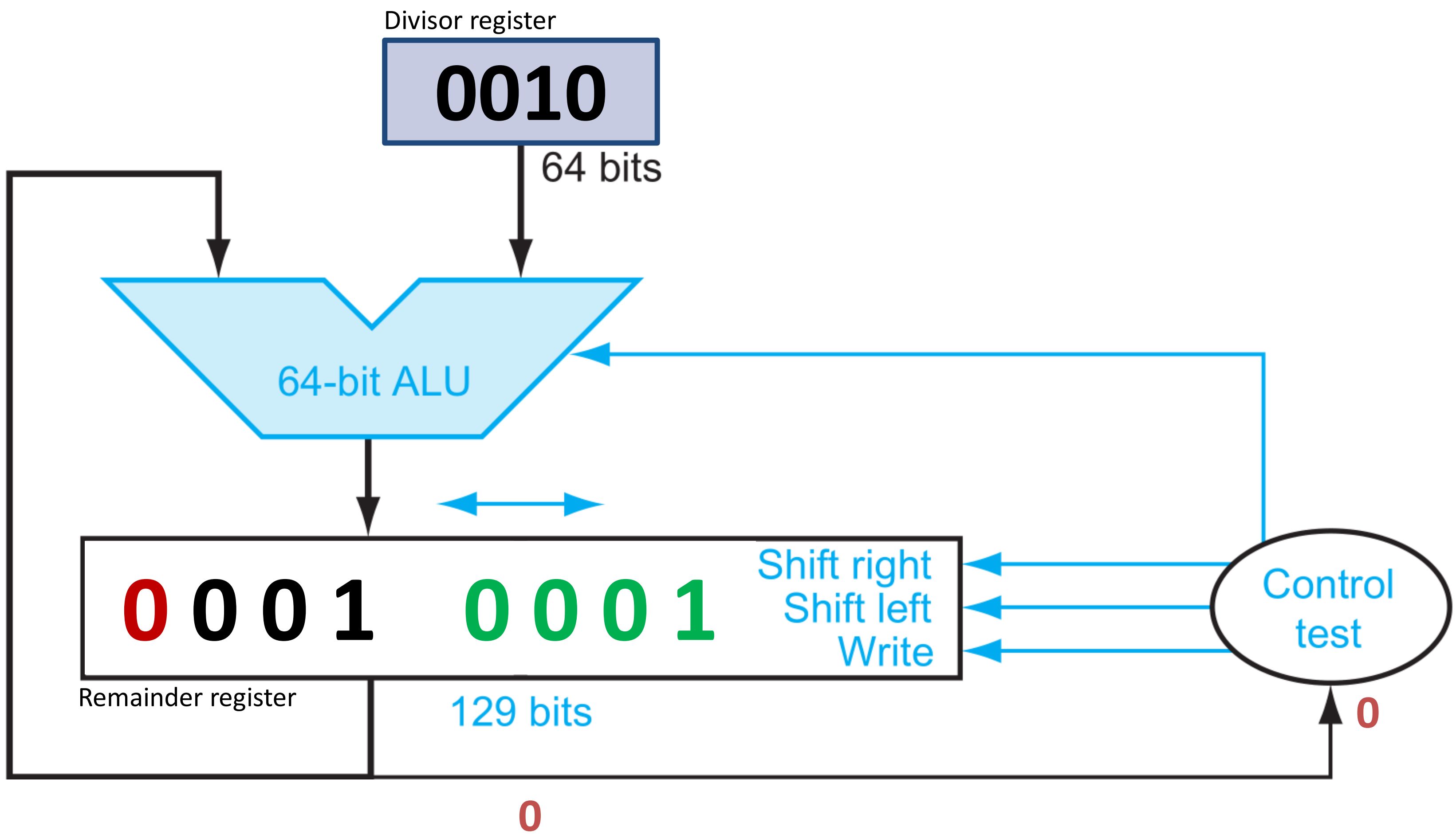
Left of the remainder = Left of the remainder - divisor



Improved Division Hardware

Iteration 5

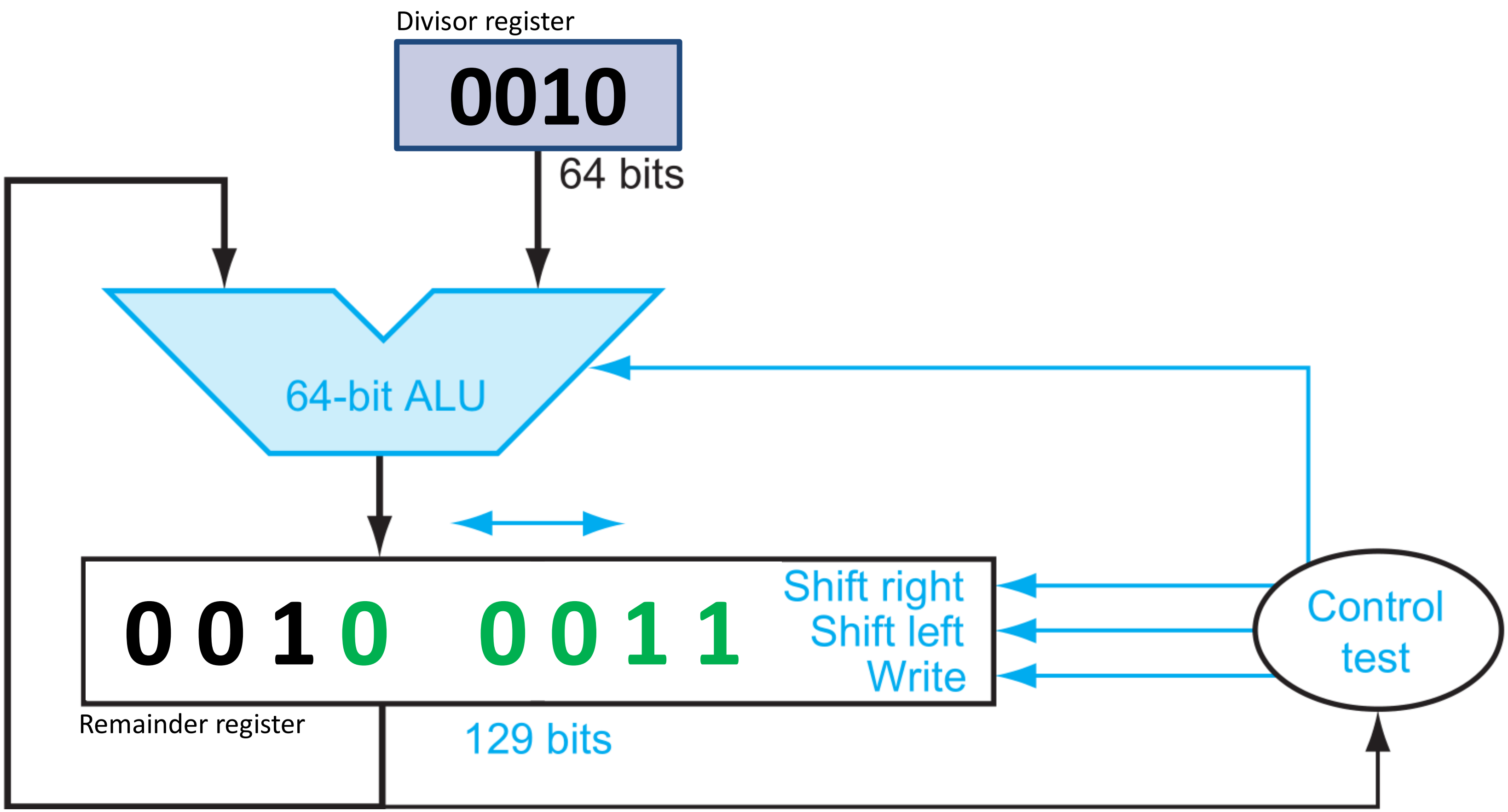
Test MSB to check if it is 1



Improved Division Hardware

Iteration 5

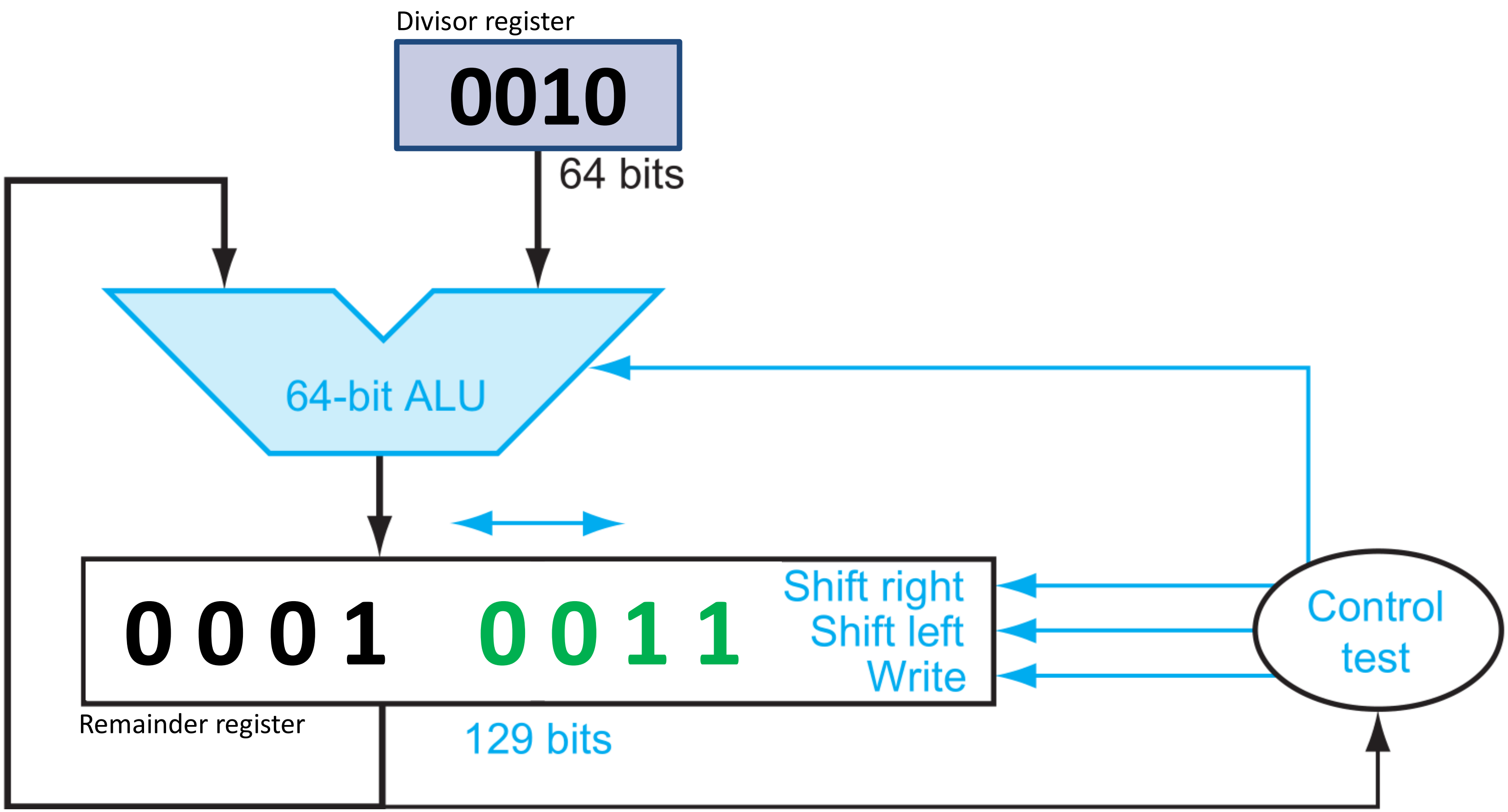
Shift left the remainder



Improved Division Hardware

Last Adjustment

Shift right the left-half of the remainder

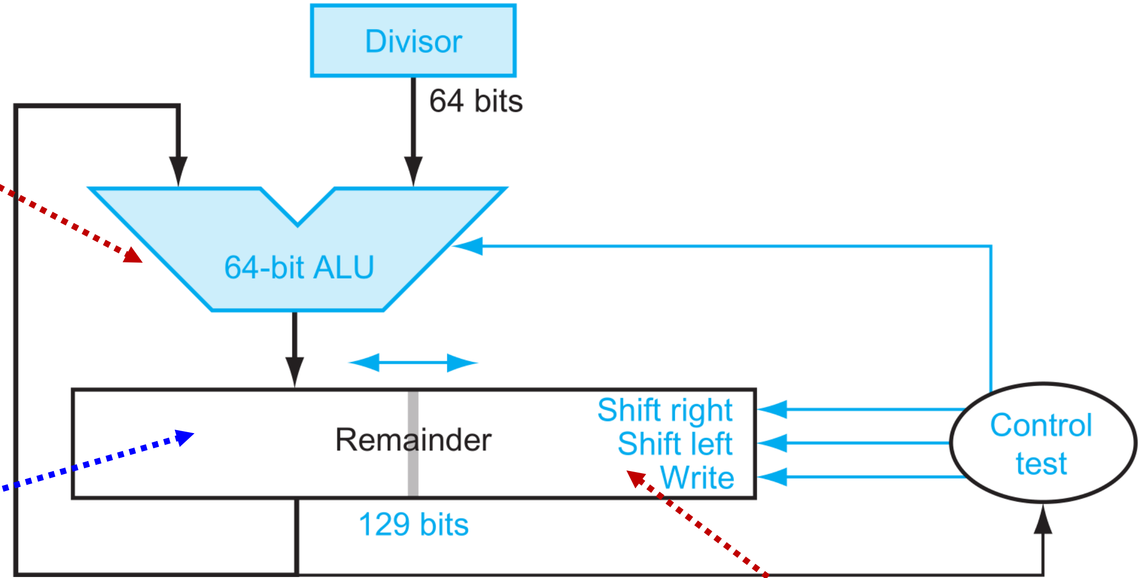


Improved Division HW

Iteration	Step	Divisor	Remainder (Quotient will place in right half)	
0	Initial Values	0010	0000 : 0111	
1	1: Rem = Rem - Div	0010	1110 : 0111	
	2: Rem < 0 → + Div, sll Q, Q0=0	0010	0000 : 1110	**restore**
2	1: Rem = Rem - Div	0010	1110 : 1110	
	2: Rem < 0 → + Div, sll Q, Q0=0	0010	0001 : 1100	**restore**
3	1: Rem = Rem - Div	0010	1111 : 1100	
	2: Rem < 0 → + Div, sll Q, Q0=0	0010	0011 : 1000	**restore**
4	1: Rem = Rem - Div	0010	0001 : 1000	
	2: Rem >= 0 → sll Q, Q0=1	0010	0011 : 0001	
5	1: Rem = Rem - Div	0010	0001 : 0001	
	2: Rem >= 0 → sll Q, Q0=1	0010	0010 : 0011	
6	Shift right the left half of remainder	0010	0001 : 0011	

Improved Division Hardware

need subtraction capability



32bit divisor subtracted from left half of remainder

Quotient goes here

- Compare with mult H/W