Topic V21

Computer Arithmetic:
Multiplication and Division
Readings: (Section 3.3-3.4)

Multiplication (unsigned)

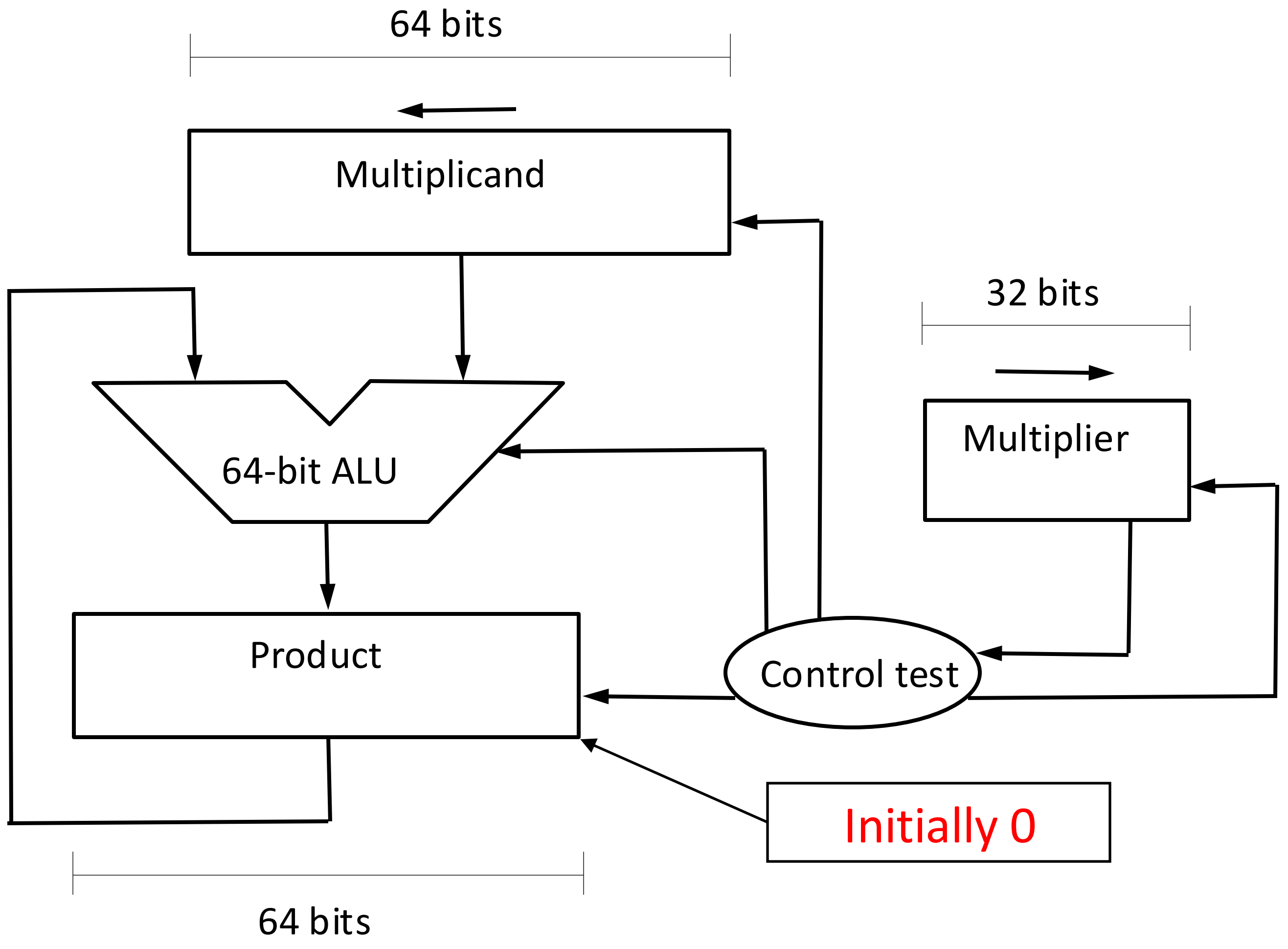
Start with the long-multiplication approach

```
multiplicand  ------>   1000
multiplier    ----x-->  1001
                    ---------
                        1000
                       0000
                      0000
                     1000
                    ---------
product       ---->  1001000
```

Length of the product is the sum of operand lengths

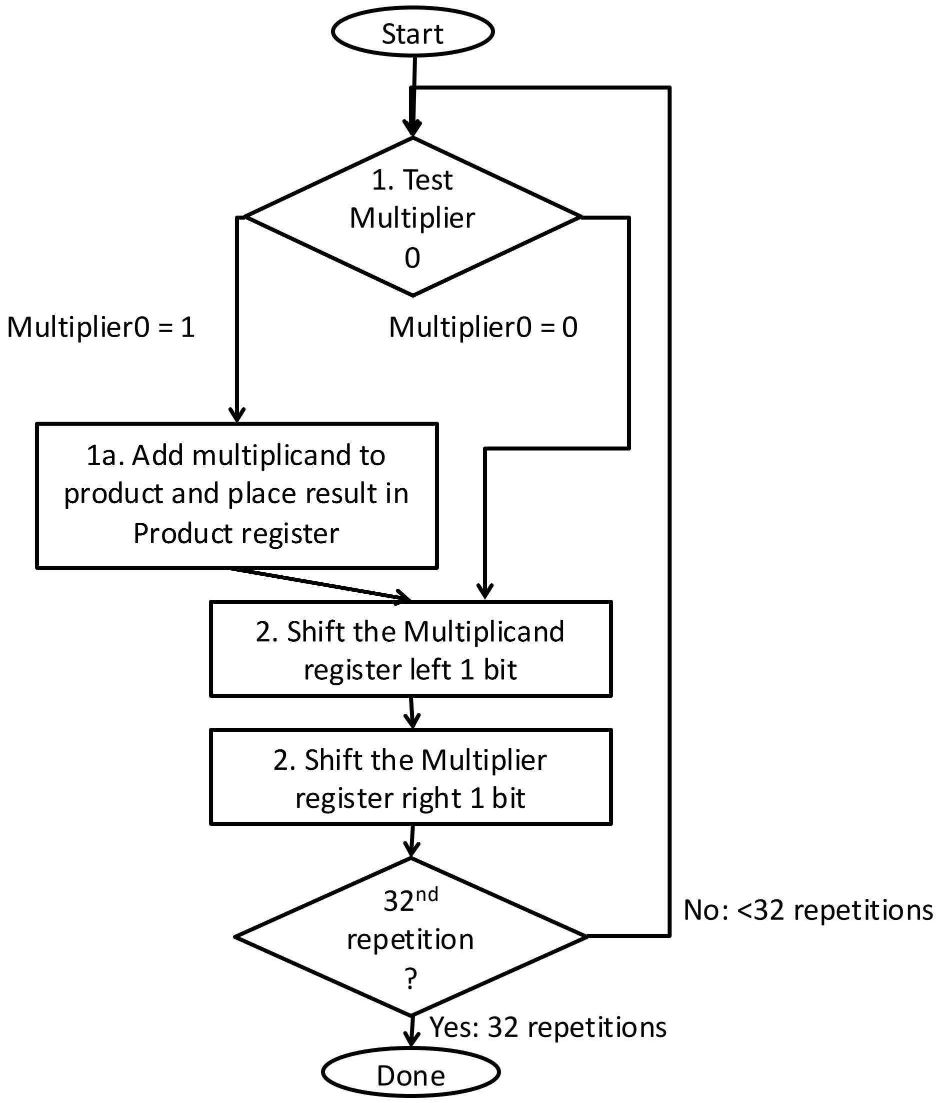
Multiplication Hardware

```
multiplicand          1000
multiplier          x 1001
                  --------
                      1000
                     0000
                    0000
                   1000
                  --------
product            1001000
```

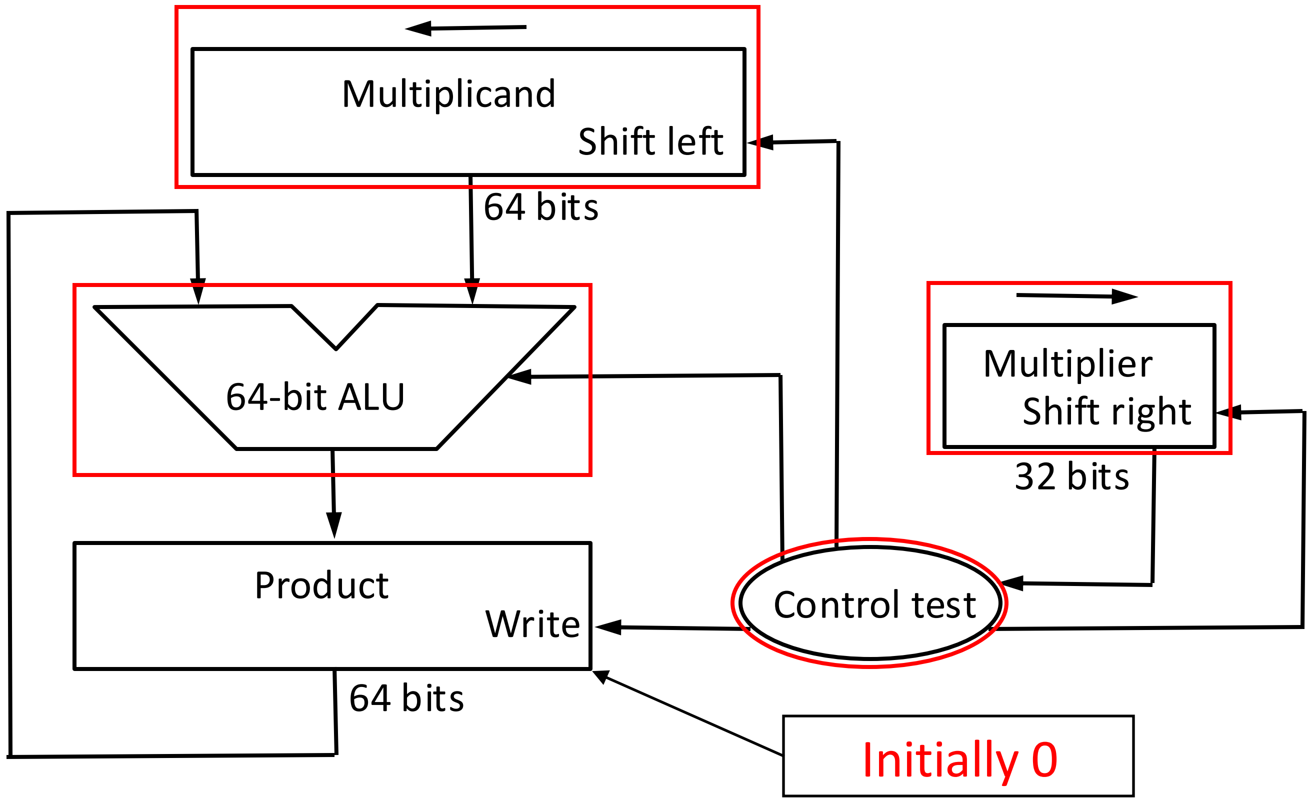


ALU ≡ Arithmetic and Logic Unit

Multiplication Hardware

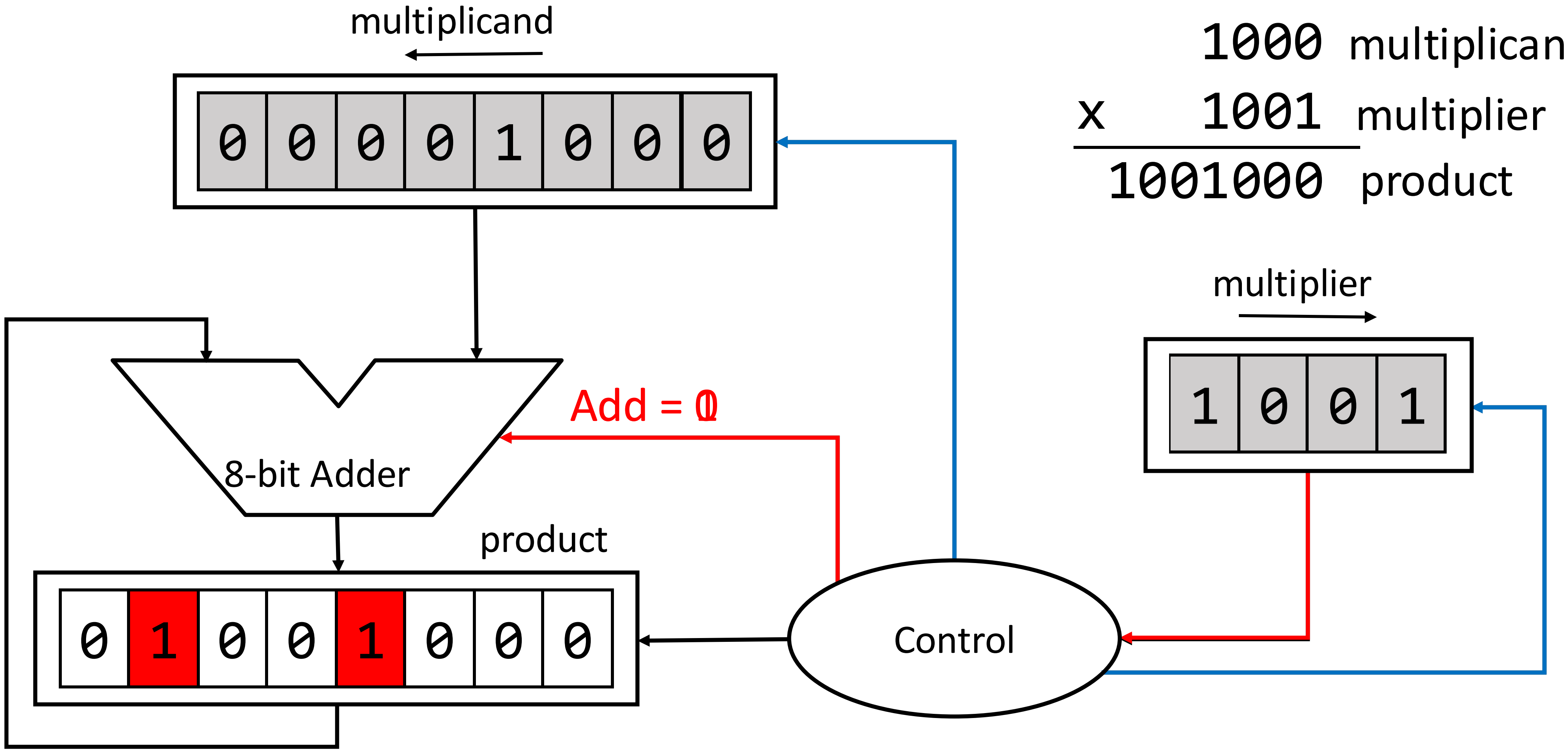


Start
1. Test Multiplier 0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place result in Product register
2. Shift the Multiplicand register left 1 bit
2. Shift the Multiplier register right 1 bit
32nd repetition ?
No: <32 repetitions
Yes: 32 repetitions
Done



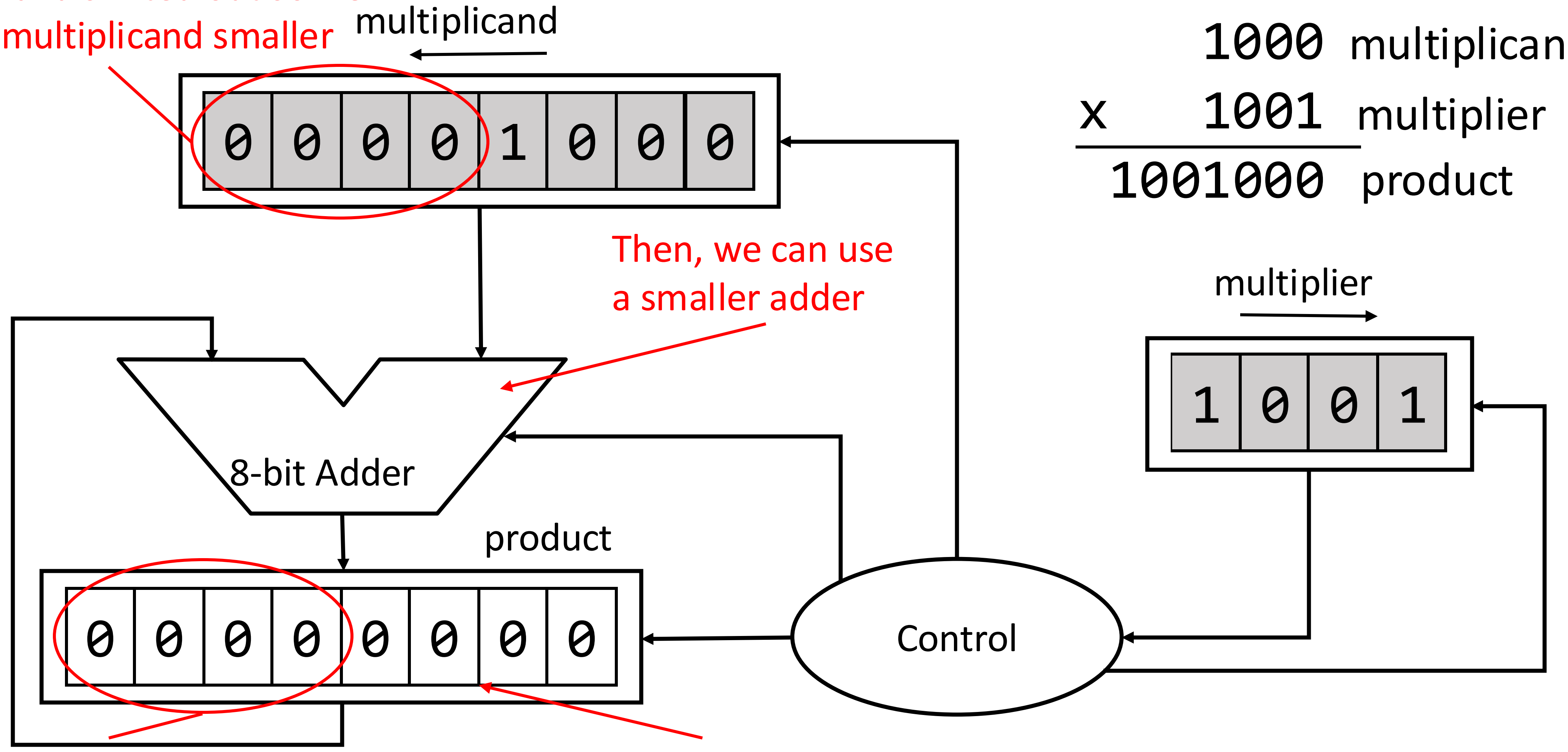
Multiplication Hardware

```
     1000  multiplicand
x    1001  multiplier
  1001000  product
```



Multiplication Hardware

This upper half was initialized with zeros and shifted out so we can make multiplicand smaller



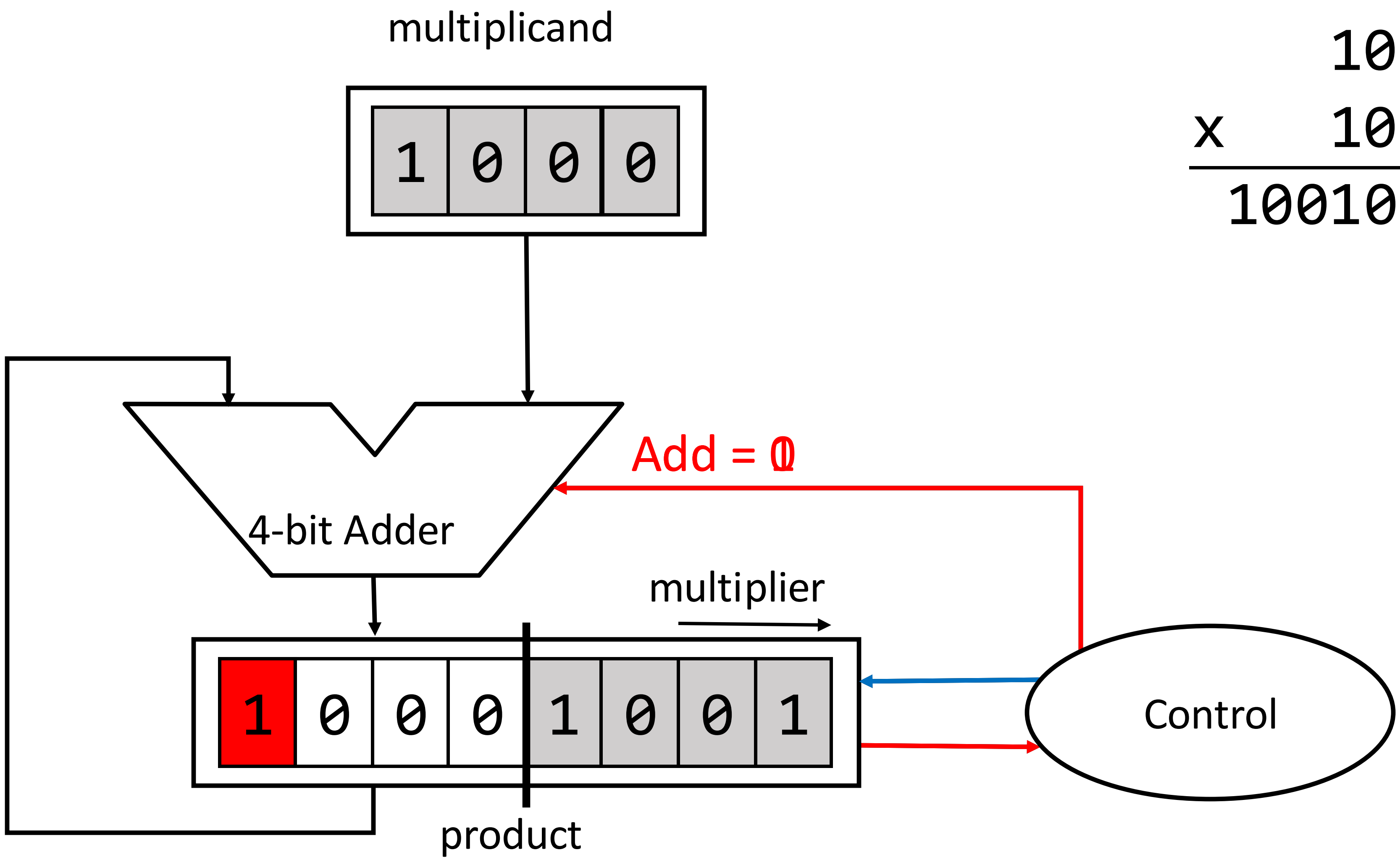
Then, we can use a smaller adder

$$\begin{array}{rl} 1000 & \text{multiplicand} \\ \times \quad 1001 & \text{multiplier} \\ \hline 1001000 & \text{product} \end{array}$$

Adder can write result into upper half of product and then we can shift to the right

Can put multiplier initially on the lower half of product because it will be shifted out

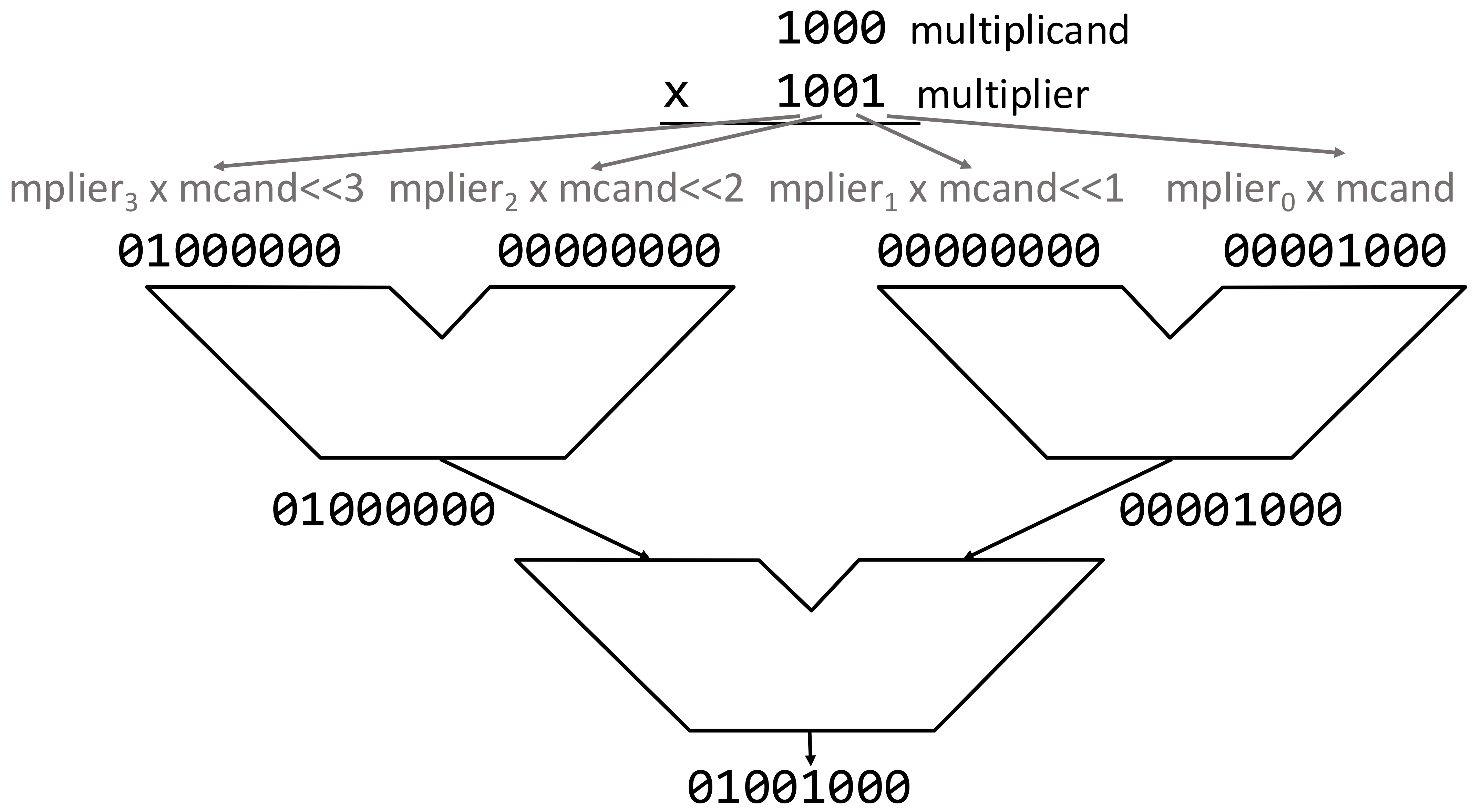
Multiplication Hardware



```
      1000  multiplicand
x     1001  multiplier
-----------
   1001000  product
```

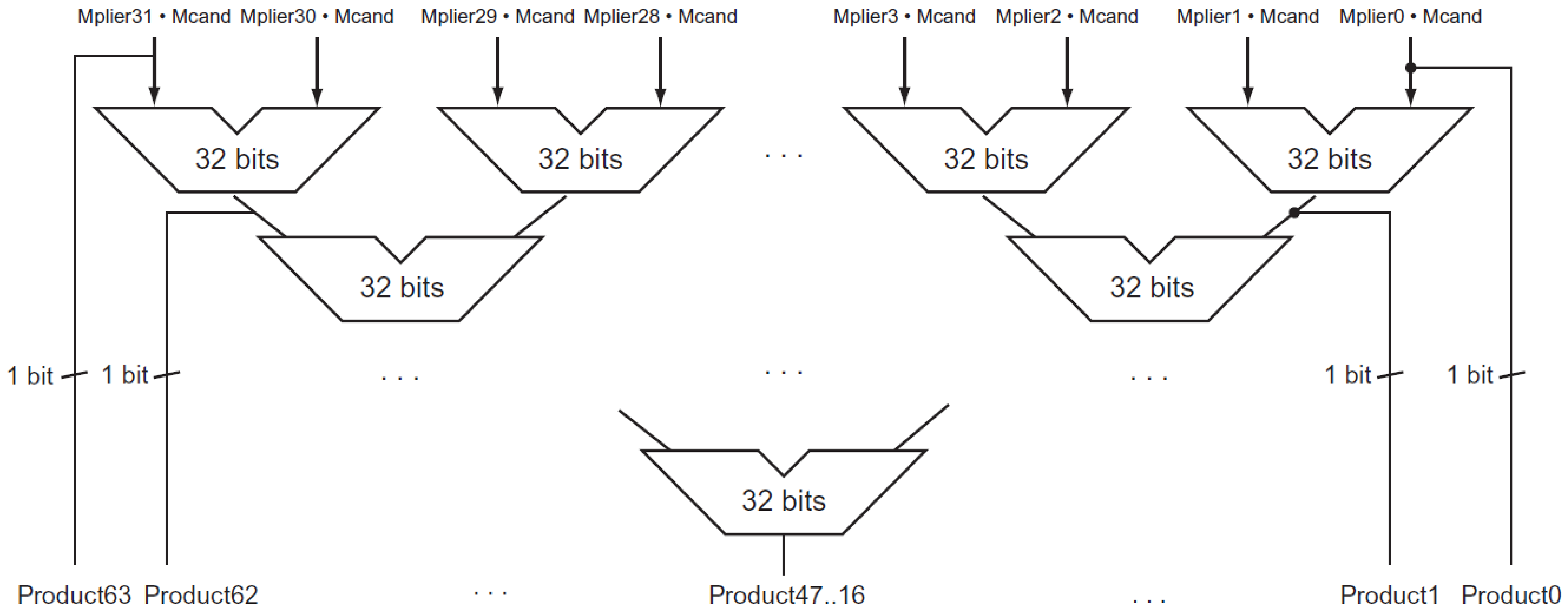
Final product after k shifts

Fast Multiplication (example)



Faster Multiplier

- Uses multiple adders
  - Cost/performance tradeoff



- Can be pipelined
  - Several multiplications performed in parallel

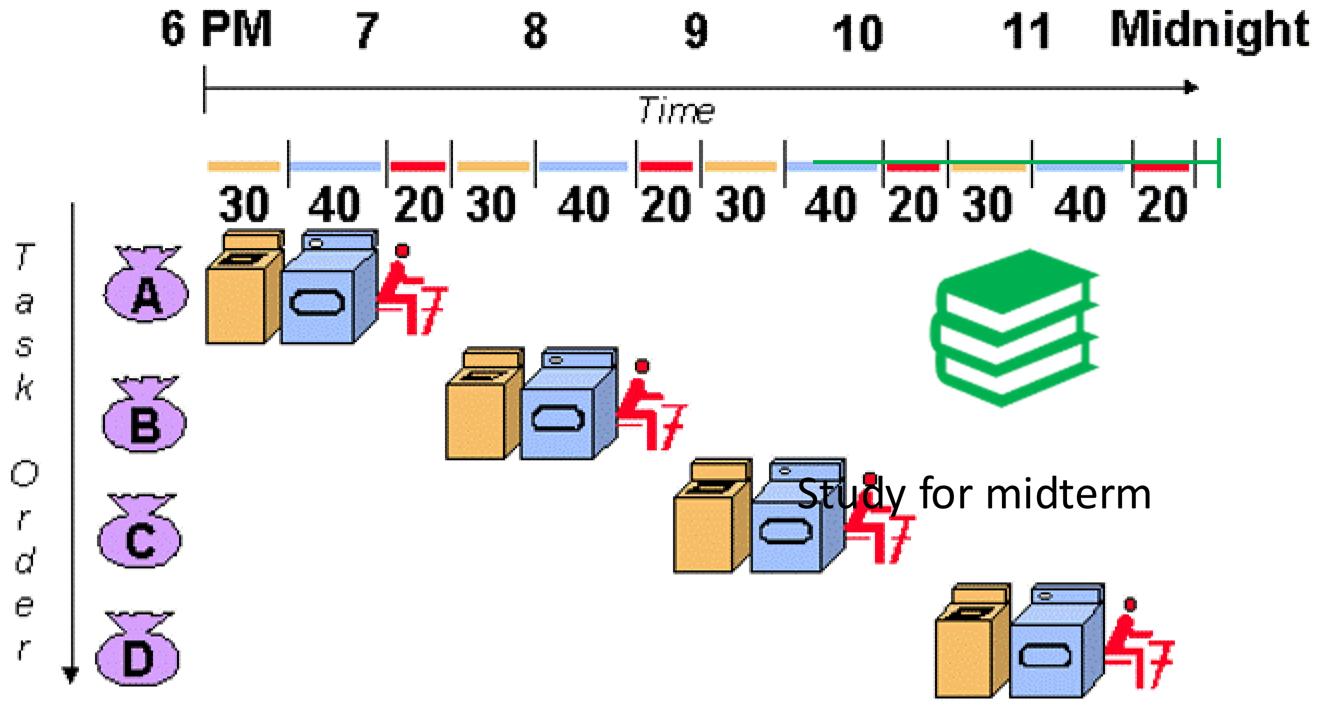
Pipelining

Laundry example – 4 loads of clothes

Washing takes 30 minutes

Drying takes 40 minutes

Folding takes 20 minutes



Study for midterm

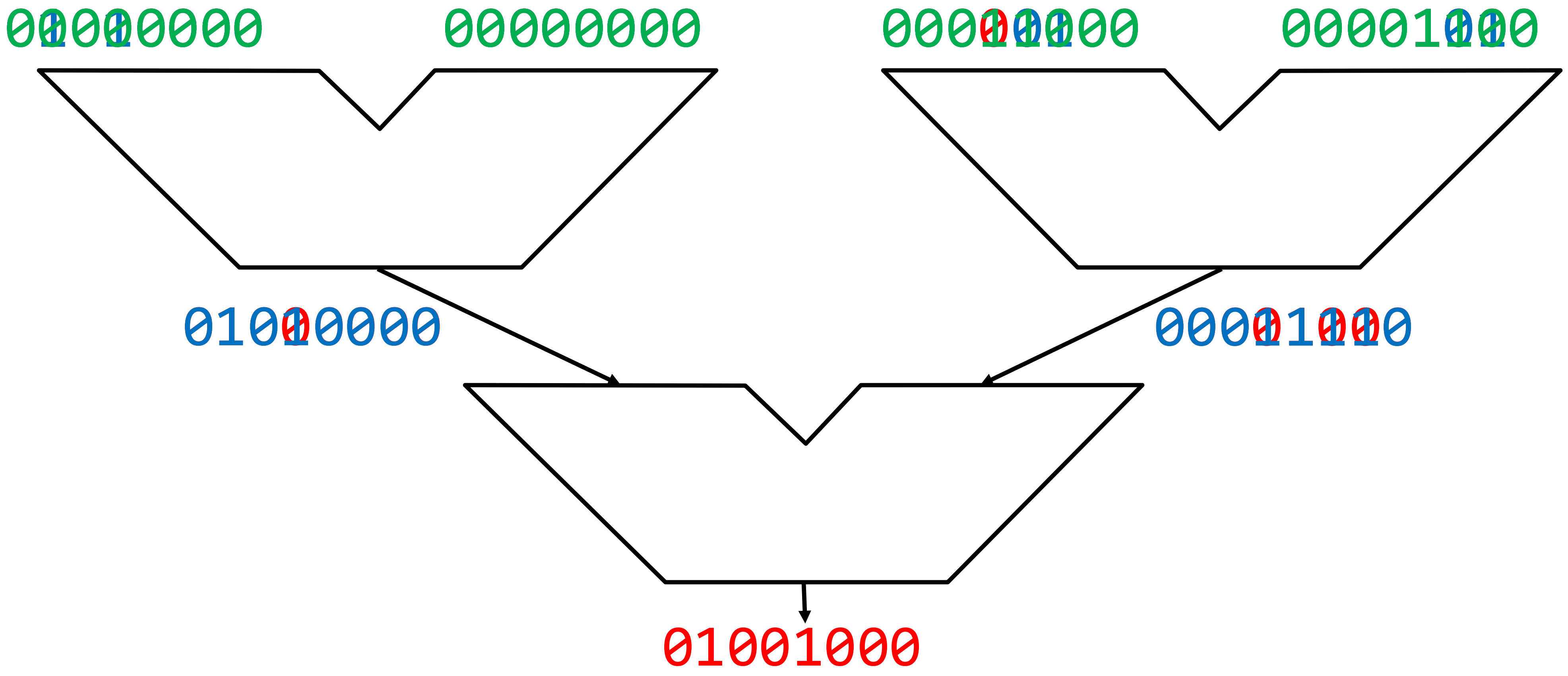
Pictures taken from: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/risc/pipelining/index.html

Pipelined Multiplication (example)

```
      1000  multiplicand
x     1001  multiplier
```

```
      1010  multiplicand
x     1011  multiplier
```

```
      1100  multiplicand
x     0011  multiplier
```



RISC-V Multiplication

Four instructions to produce a properly signed or unsigned 64-bit product

```
mul         rd, rs1, rs2
```

performs 32-bit x 32-bit multiplication and places lower 32-bits of the product in rd

```
mulh        rd, rs1, rs2
```

perform 32-bit x 32-bit multiplication but place the upper 32-bits of the product in rd

RISC-V Multiplication

```
mulhu    rd, rs1, rs2
```

perform 32-bit x 32-bit multiplication (both operands unsigned) place the upper 32-bits of the product in rd

```
mulhsu   rd, rs1, rs2
```

perform 32-bit x 32-bit multiplication (signed x unsigned) and place the upper 32-bits of the product in rd

64-bit = 32-bit x 32-bit RISC-V Multiplication

```
mulh rdh, rs1, rs2
mul  rdl, rs1, rs2
```

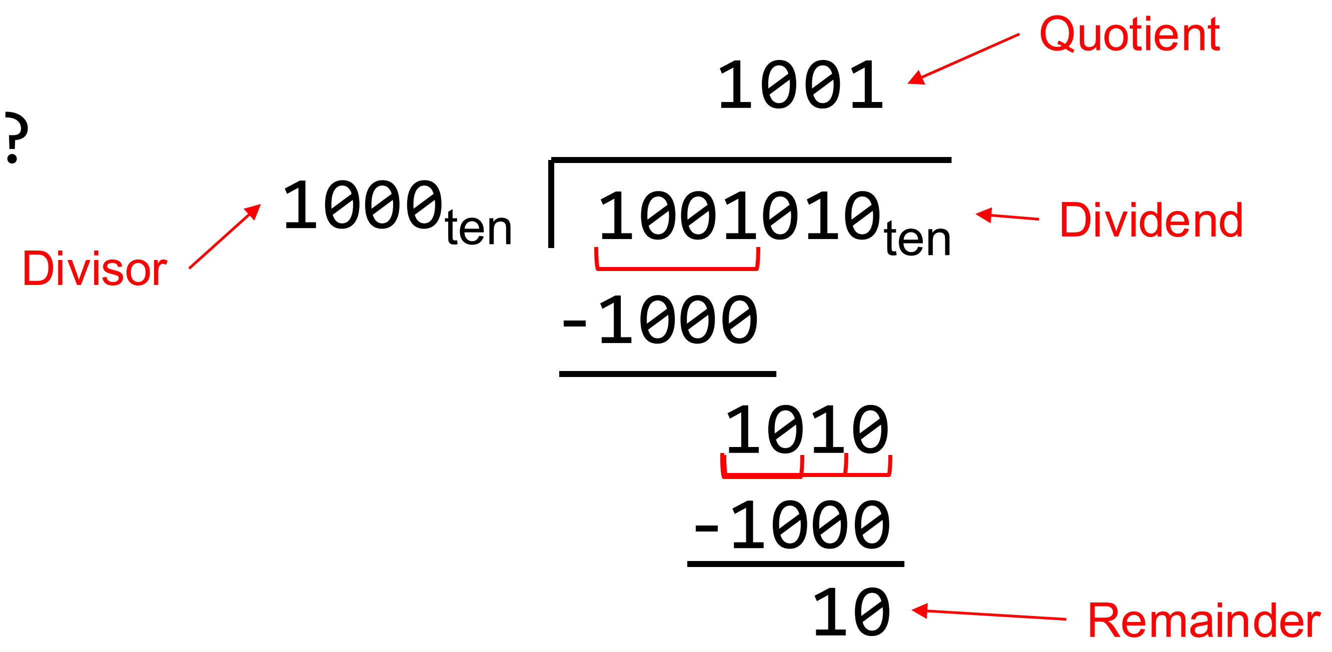
Source registers must appear in same order in both instructions

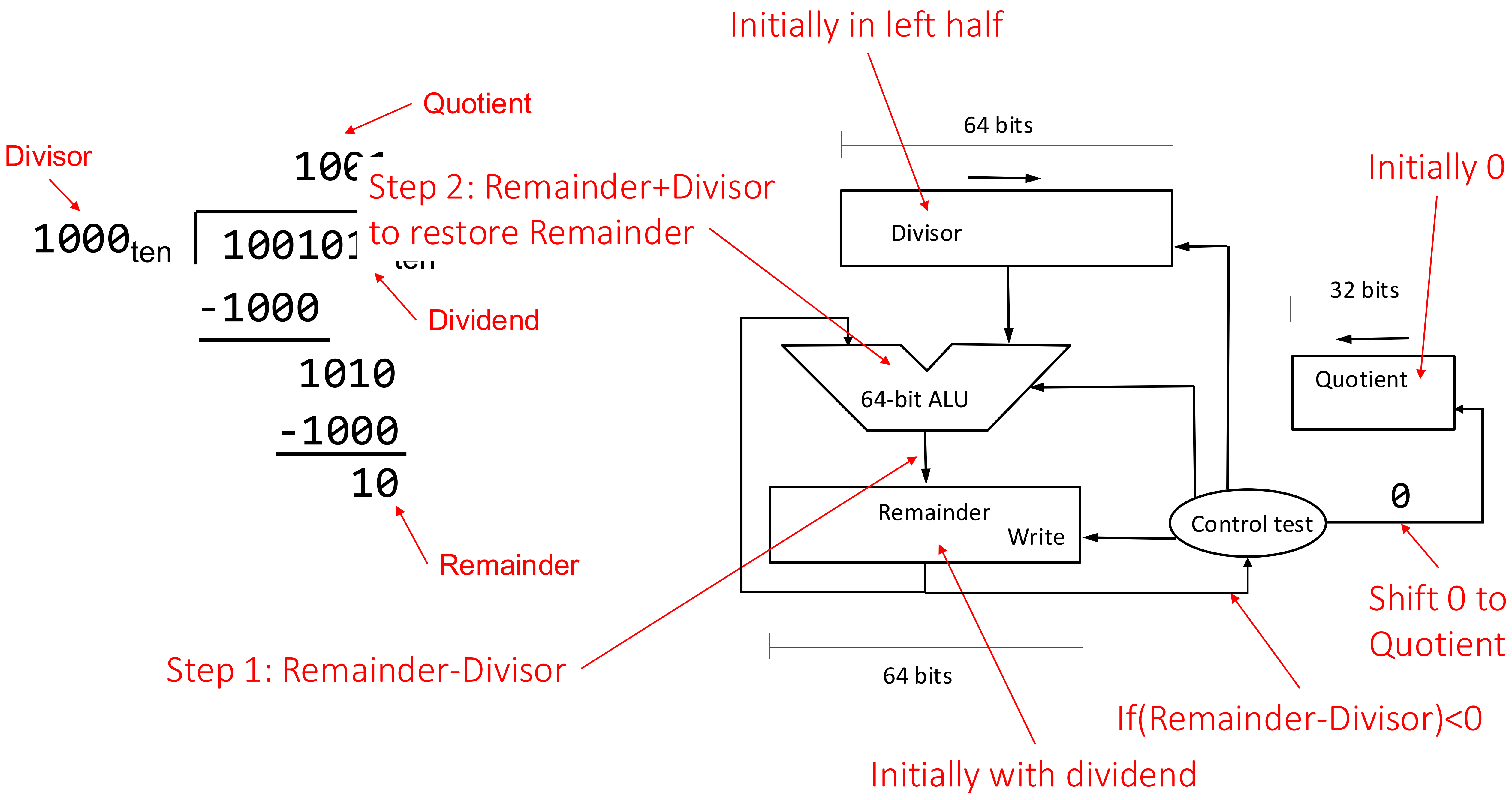
microarchitecture merges the two instructions into a single multiplication operation.

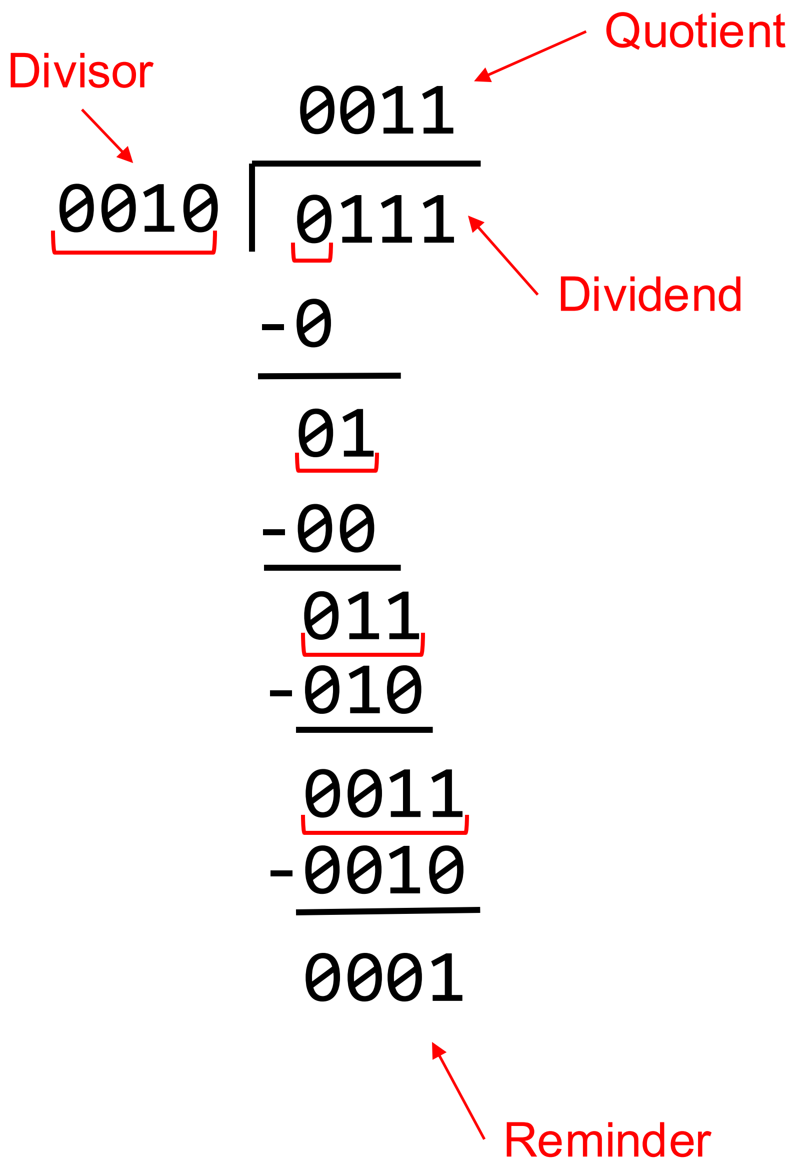
rdh cannot be the same register as either **rs1** or **rs2**

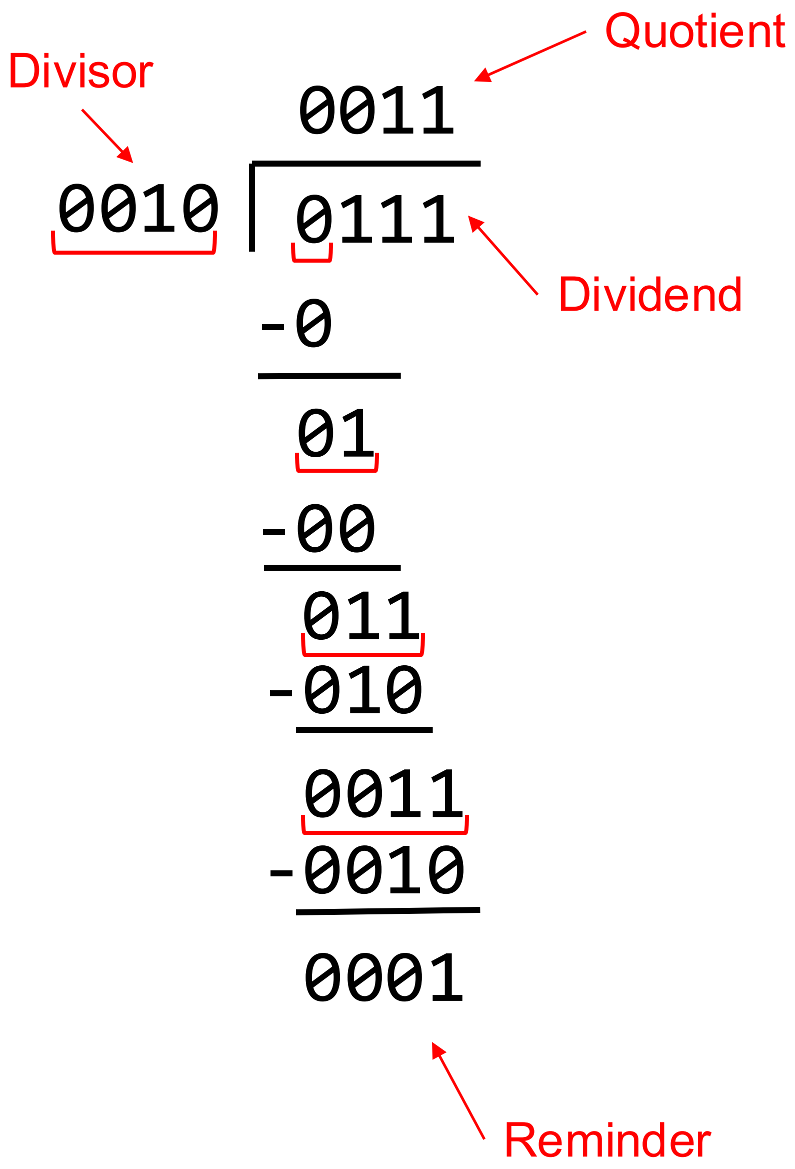
Long Division

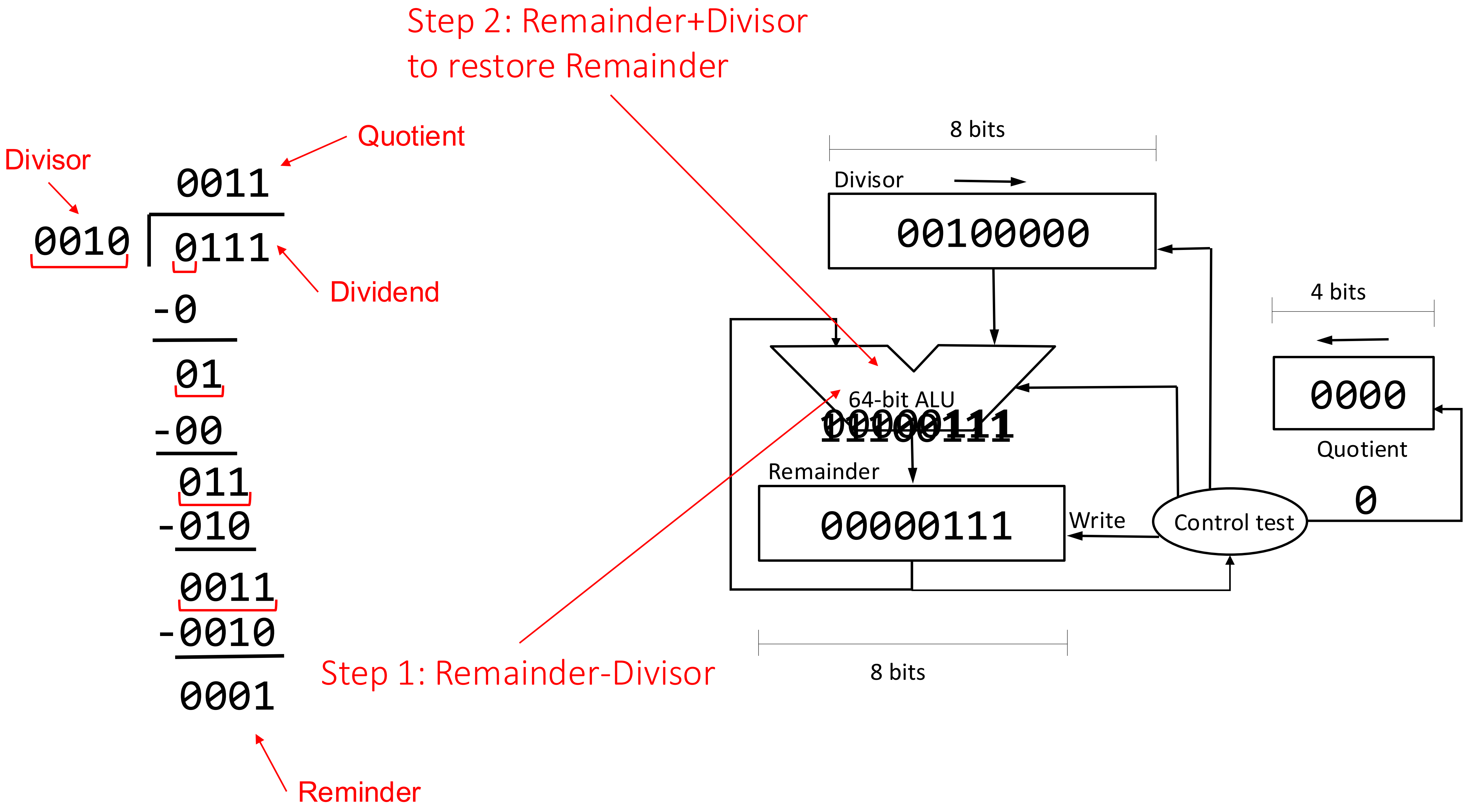
$$\frac{1001010_{ten}}{1000_{ten}} = ?$$

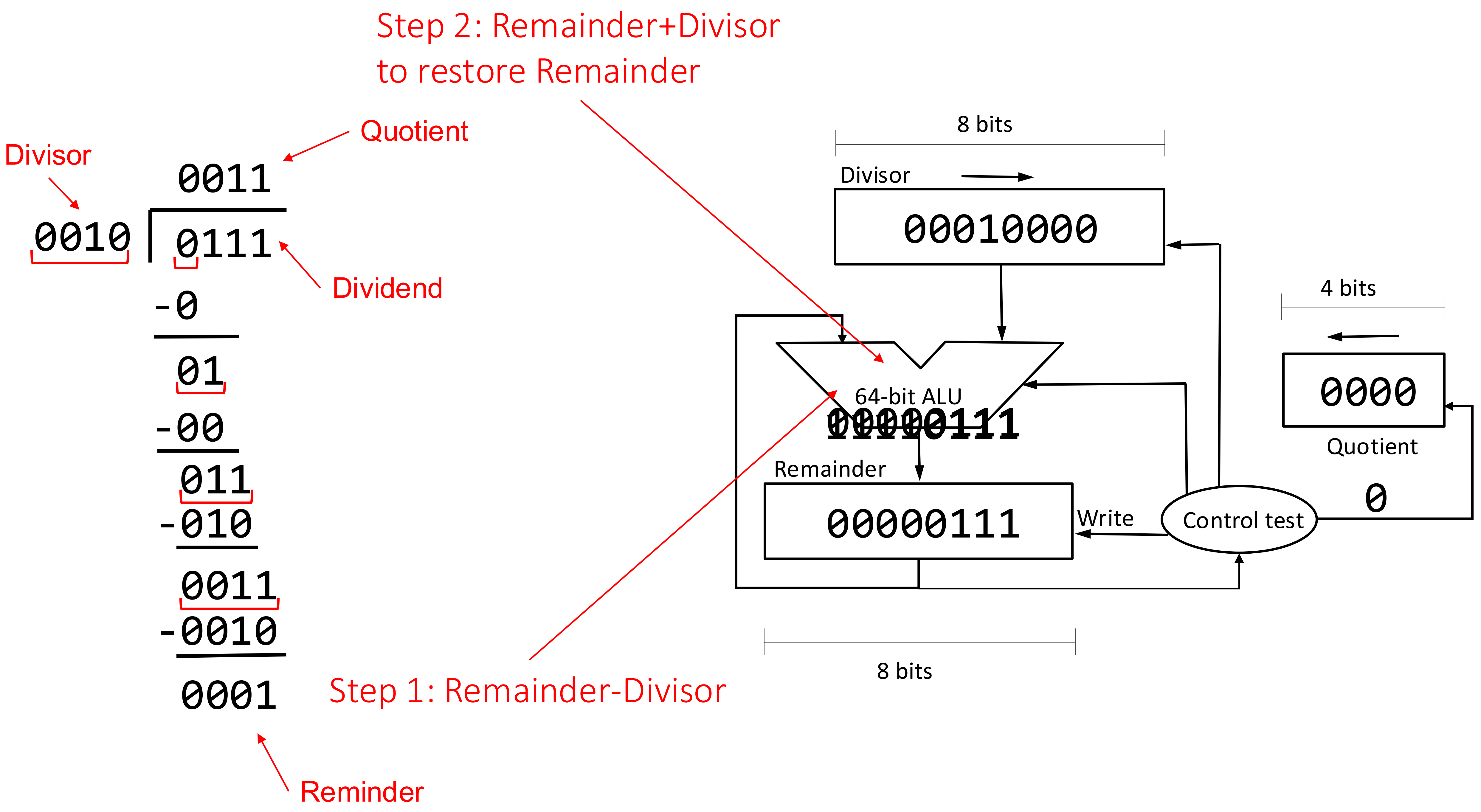



Divisor
Quotient
1000_{ten}
100
100101
-1000
1010
-1000
10
Dividend
Remainder
Step 2: Remainder+Divisor to restore Remainder
Step 1: Remainder-Divisor
Initially in left half
64 bits
Divisor
64-bit ALU
Remainder
Write
64 bits
Initially with dividend
Initially 0
32 bits
Quotient
Control test
0
Shift 0 to Quotient
If(Remainder-Divisor)<0


Quotient
Divisor
0011
0010
0111
Dividend
-0
01
-00
011
-010
0011
-0010
0001
Reminder


Quotient
Divisor
0011
0010
0111
Dividend
-0
01
-00
011
-010
0011
-0010
0001
Reminder


Step 2: Remainder+Divisor
to restore Remainder
Quotient
Divisor
0011
0010
0111
Dividend
-0
01
-00
011
-010
0011
-0010
0001
Step 1: Remainder-Divisor
Reminder
8 bits
Divisor
00100000
4 bits
64-bit ALU
11100111
00000111
0000
Quotient
Remainder
0
00000111
Write
Control test
8 bits


Step 2: Remainder+Divisor
to restore Remainder
Divisor
Quotient
0011
0010
0111
Dividend
-0
01
-00
011
-010
0011
-0010
0001
Reminder
Step 1: Remainder-Divisor
8 bits
Divisor
00010000
4 bits
0000
Quotient
64-bit ALU
11110111
00000111
Remainder
00000111
Write
Control test
0
8 bits

Step 2: Remainder+Divisor to restore Remainder

Quotient

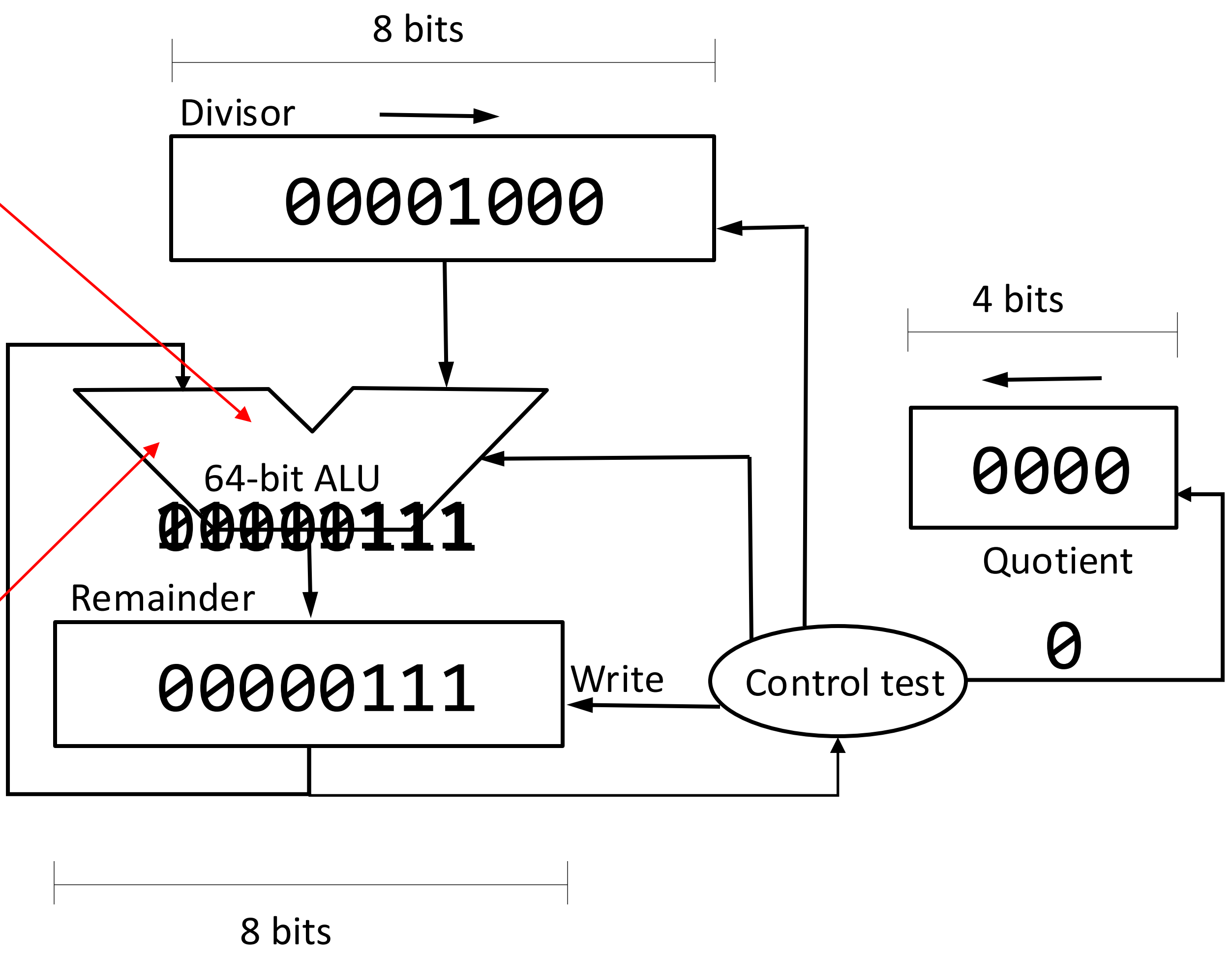
Divisor

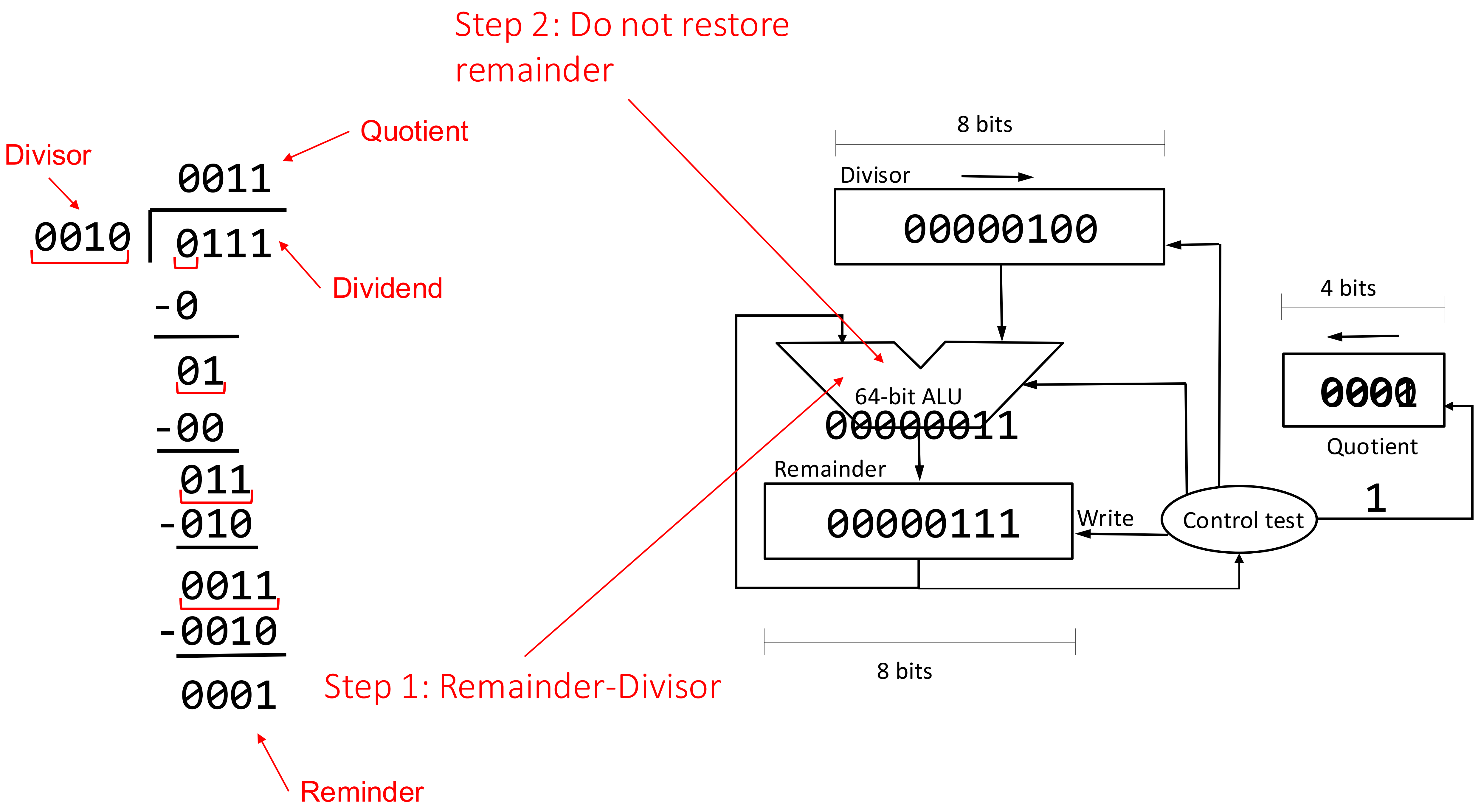
Dividend

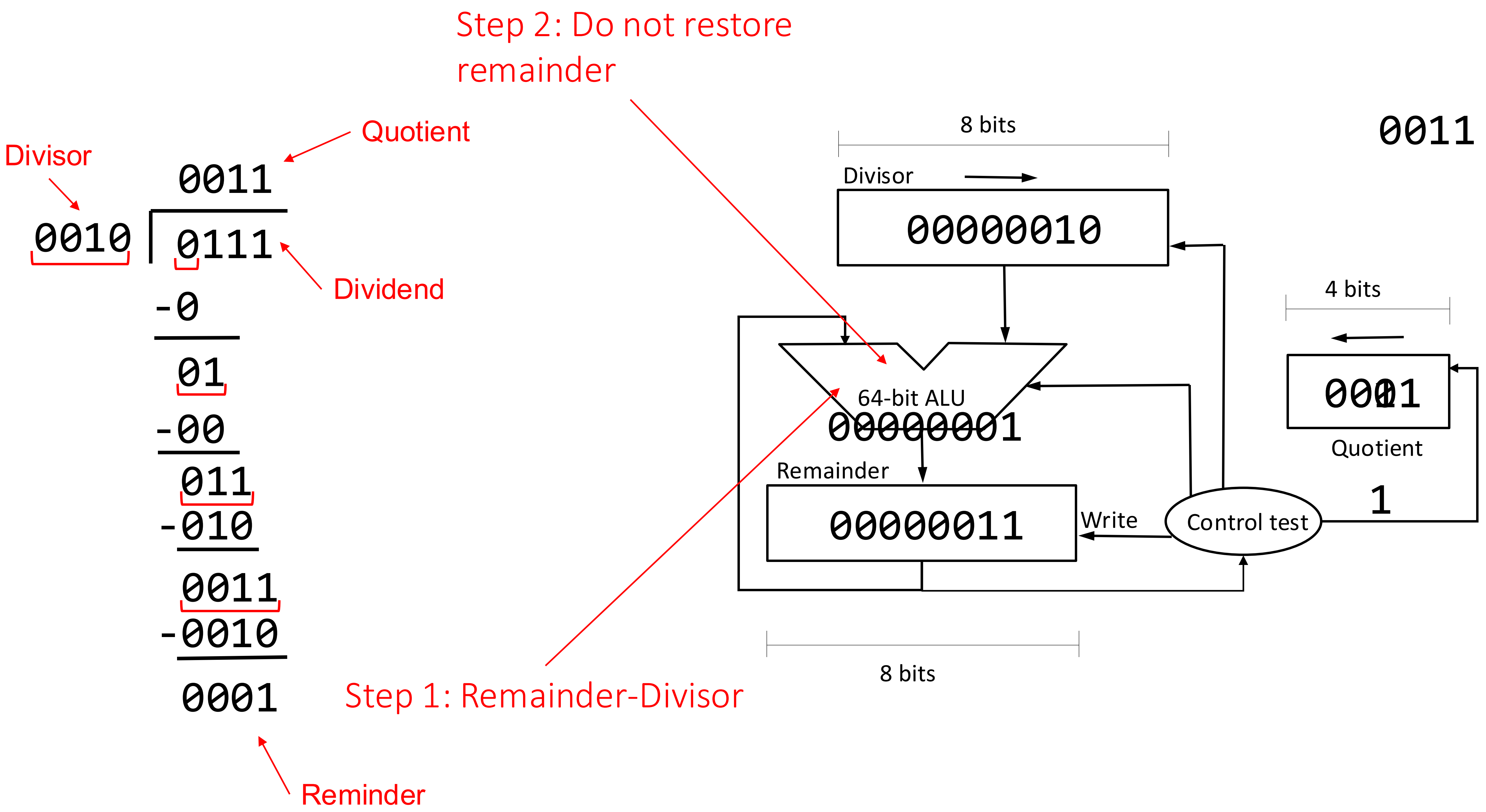
```
        0011
      ______
0010 | 0111
      -0
      ___
       01
      -00
      ___
       011
      -010
      ____
       0011
      -0010
      _____
       0001
```

Reminder

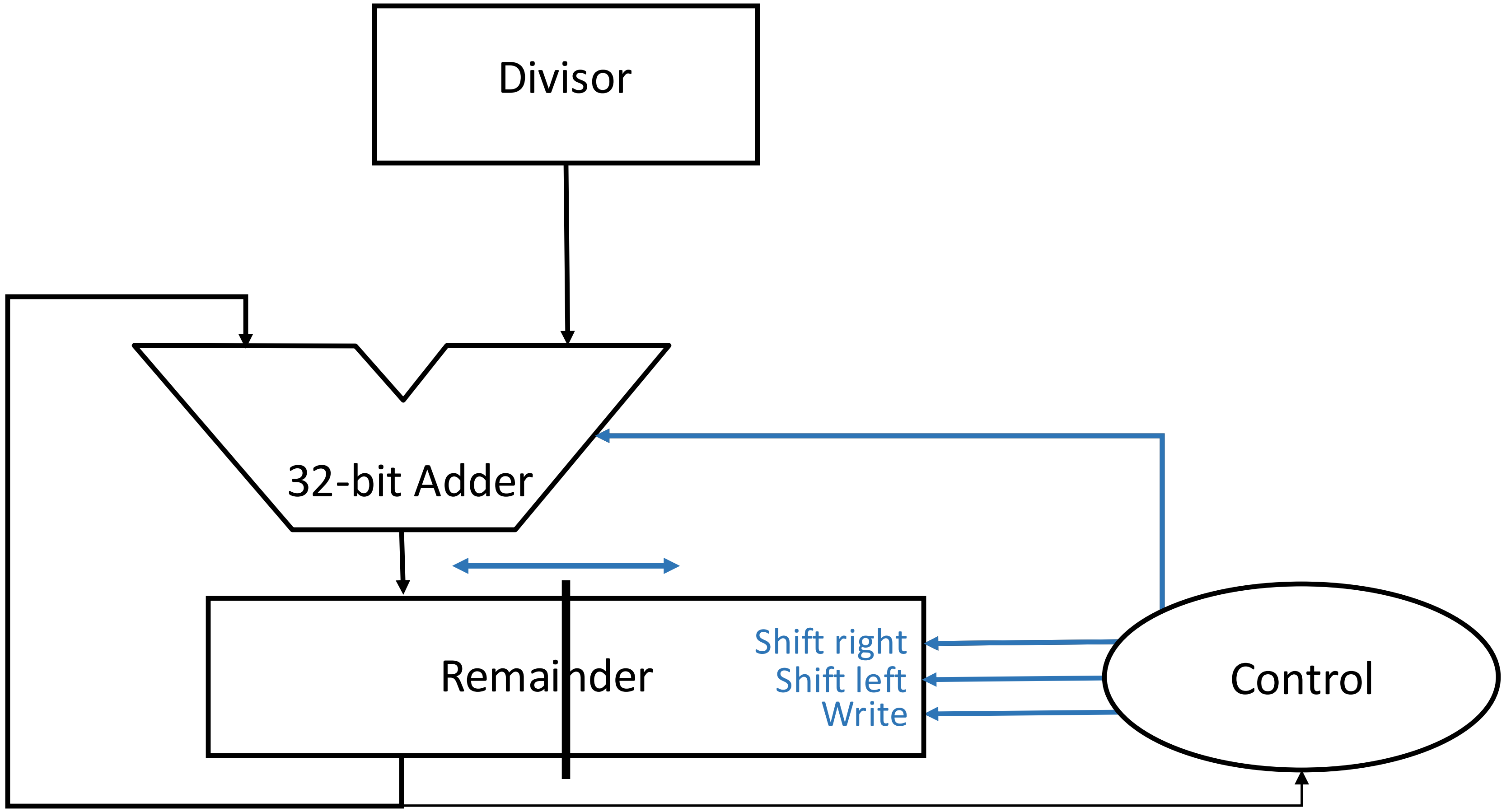
Step 1: Remainder-Divisor




Step 2: Do not restore remainder
Divisor
Quotient
0011
0010
0111
Dividend
-0
01
-00
011
-010
0011
-0010
0001
Reminder
Step 1: Remainder-Divisor
8 bits
Divisor
00000100
4 bits
0001
Quotient
1
64-bit ALU
00000011
Remainder
00000111
Write
Control test
8 bits


Step 2: Do not restore remainder
Divisor
Quotient
0011
0010
0111
Dividend
-0
01
-00
011
-010
0011
-0010
0001
Reminder
Step 1: Remainder-Divisor
8 bits
Divisor
00000010
64-bit ALU
00000001
Remainder
00000011
Write
Control test
8 bits
4 bits
0001
Quotient
1
0011

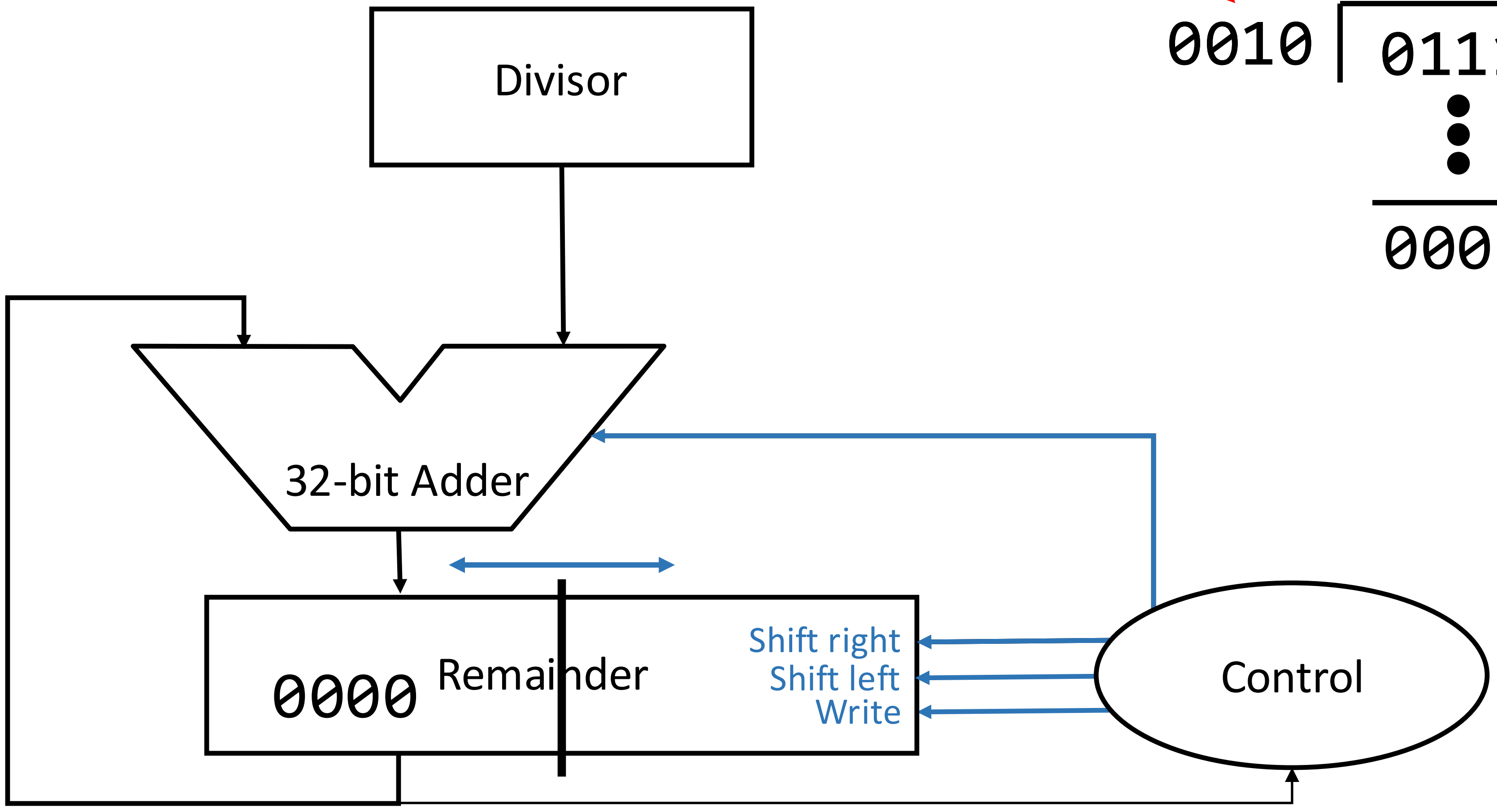
Division Hardware

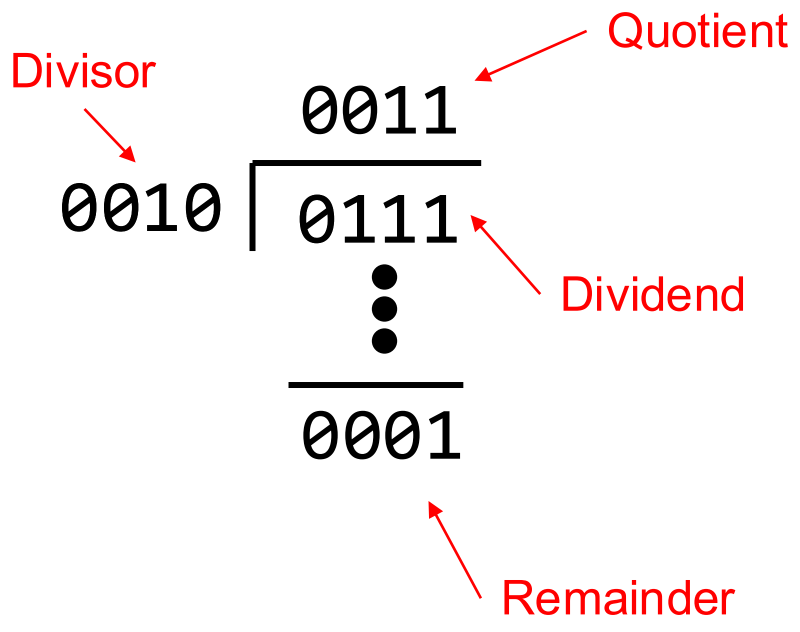


Algorithm description in slides 5 and 6 of:

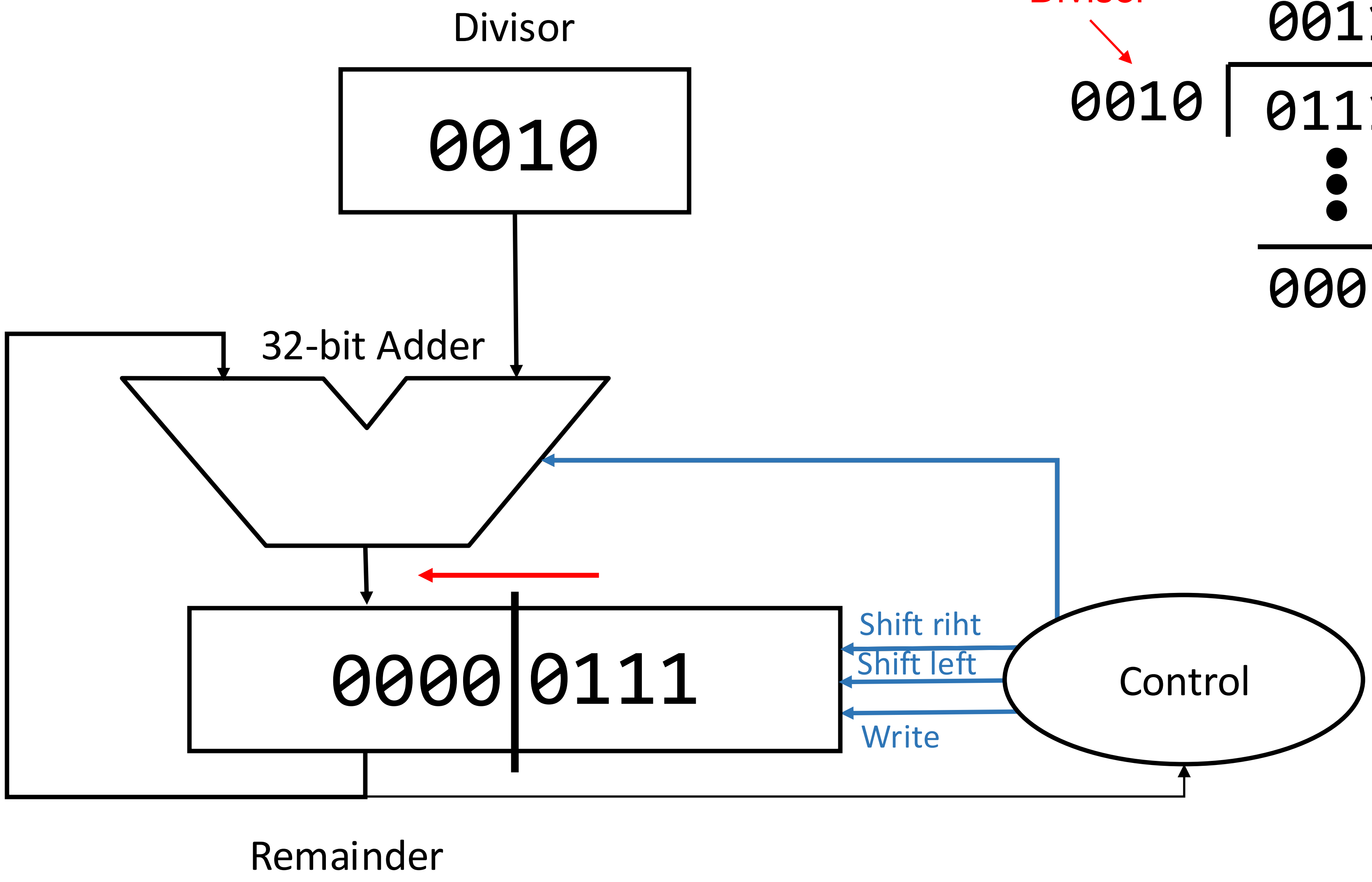
https://people.cs.pitt.edu/~cho/cs0447/currentsemester/handouts/lect-ch3p2_4up.pdf

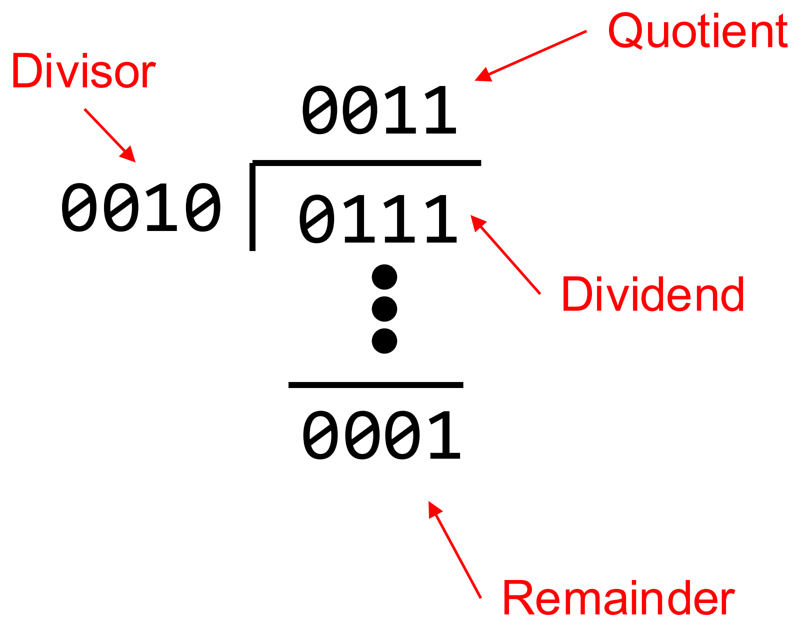
Division Hardware





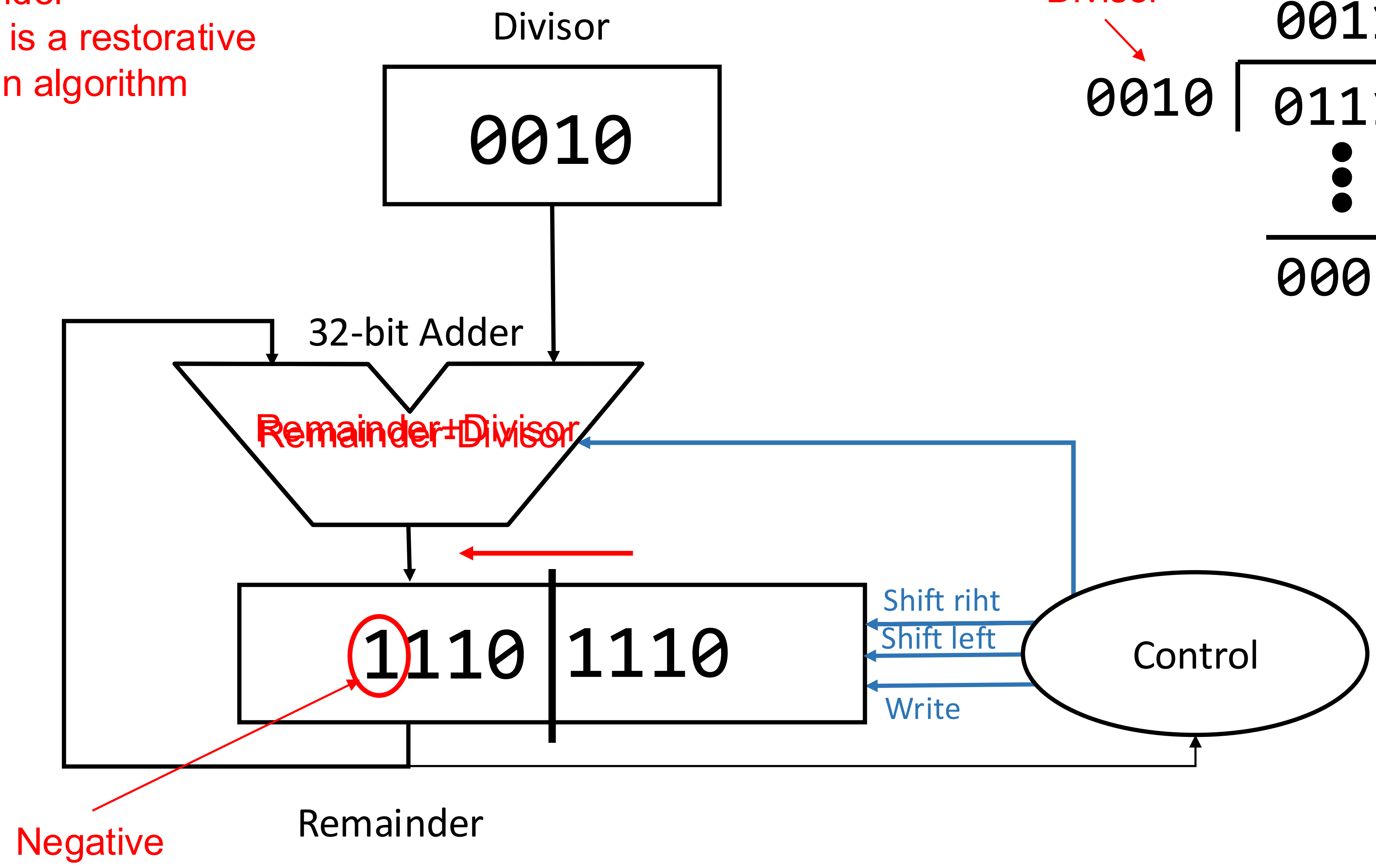
Division Hardware

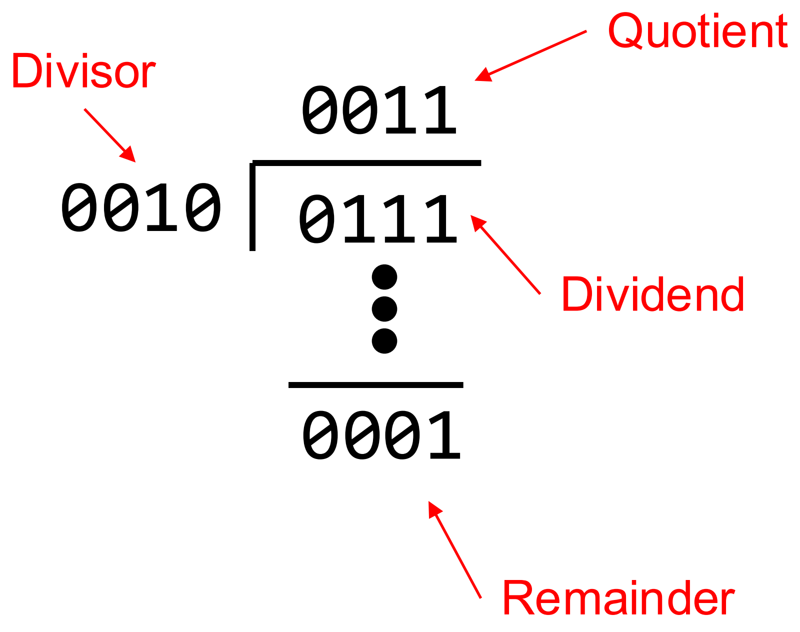




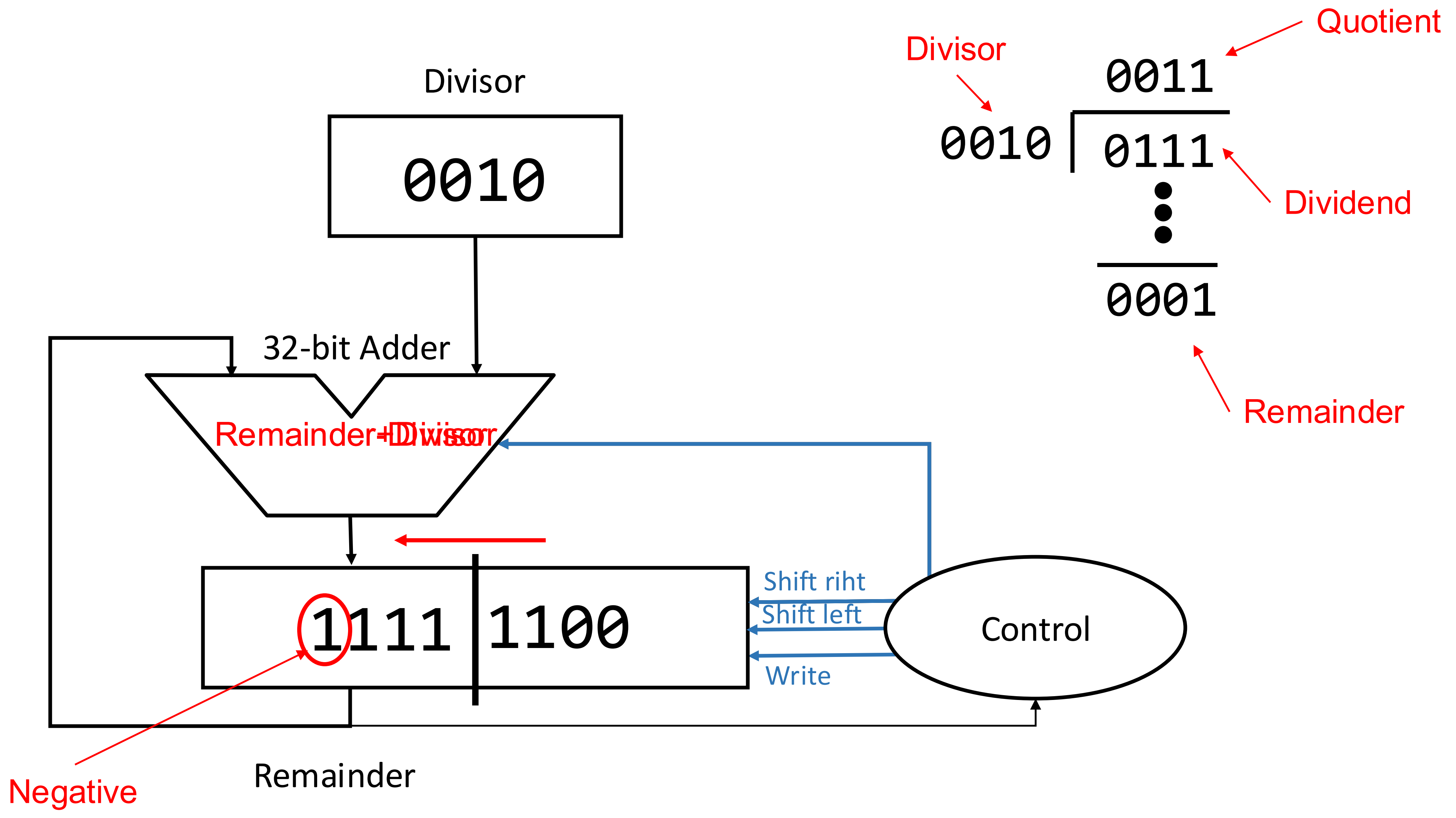
Division Hardware

This addition restores the remainder
⇒ this is a restorative division algorithm

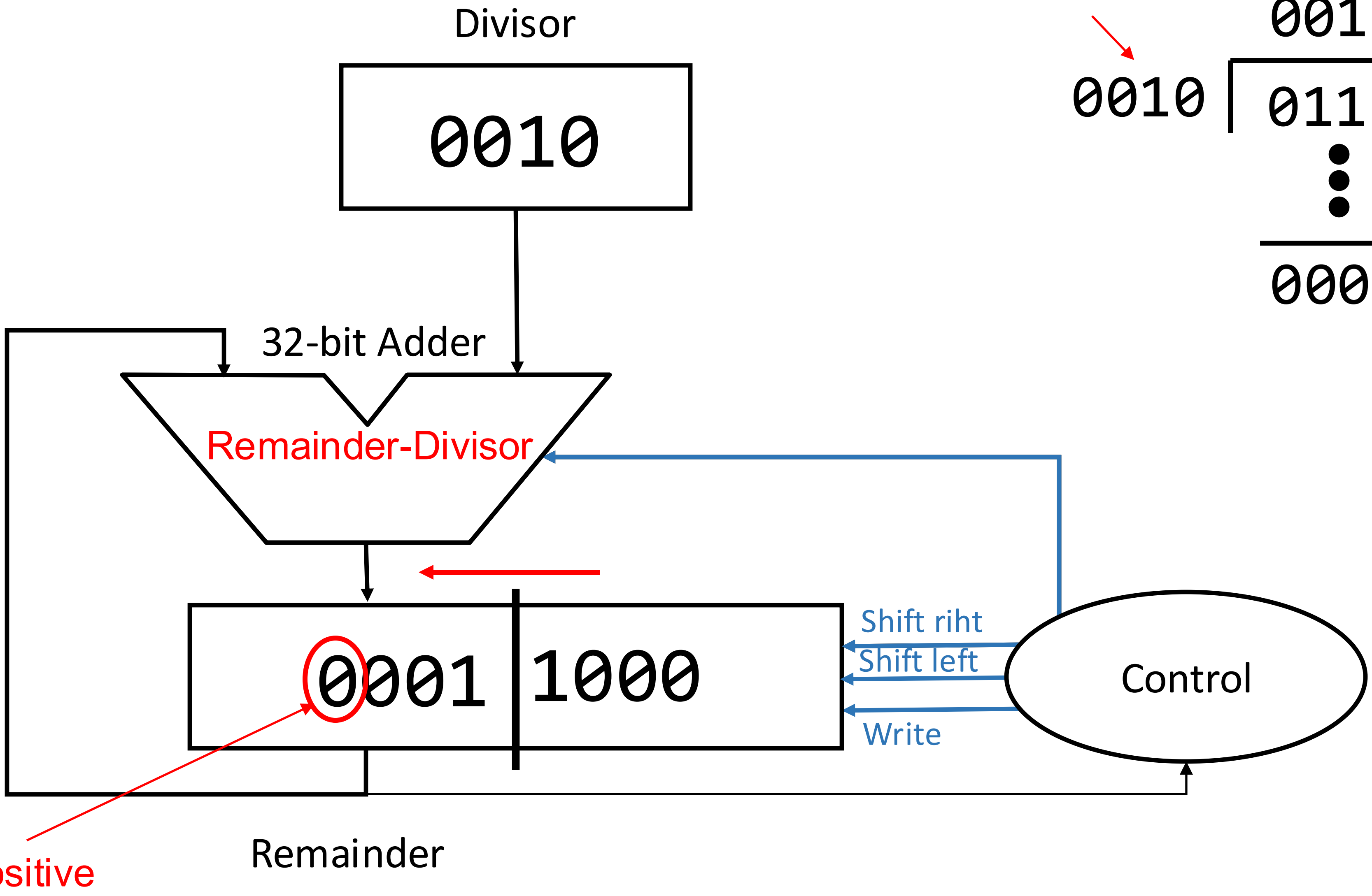


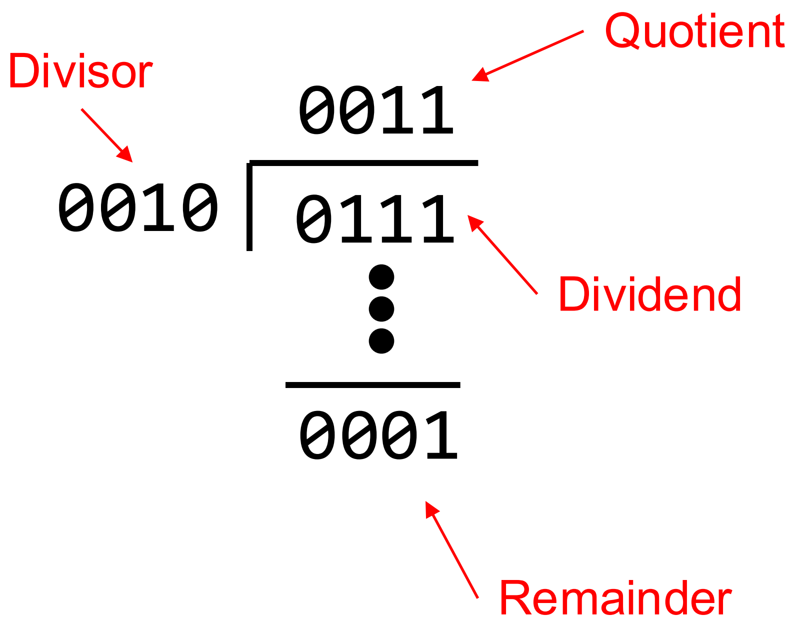


Division Hardware



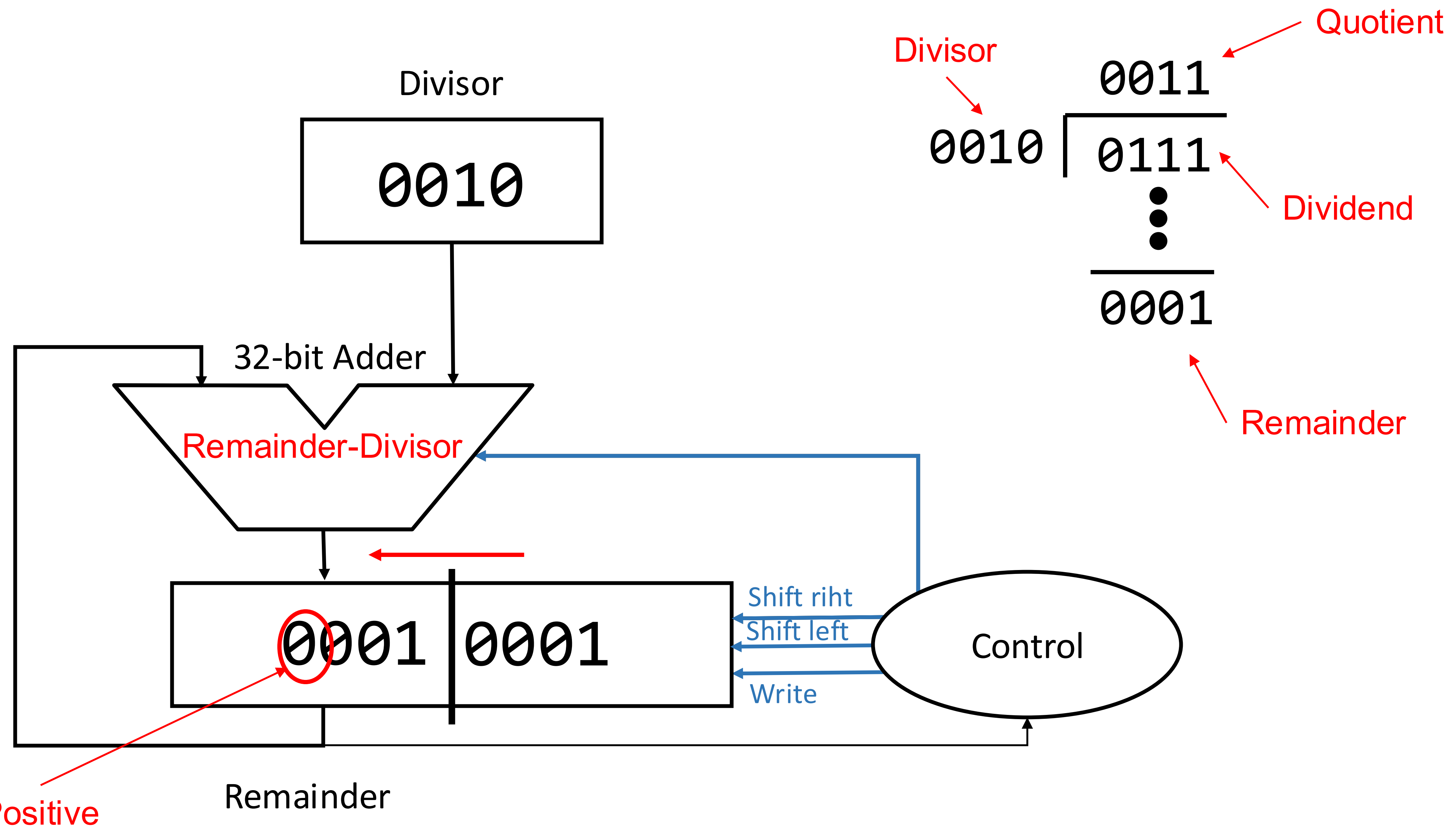
Division Hardware





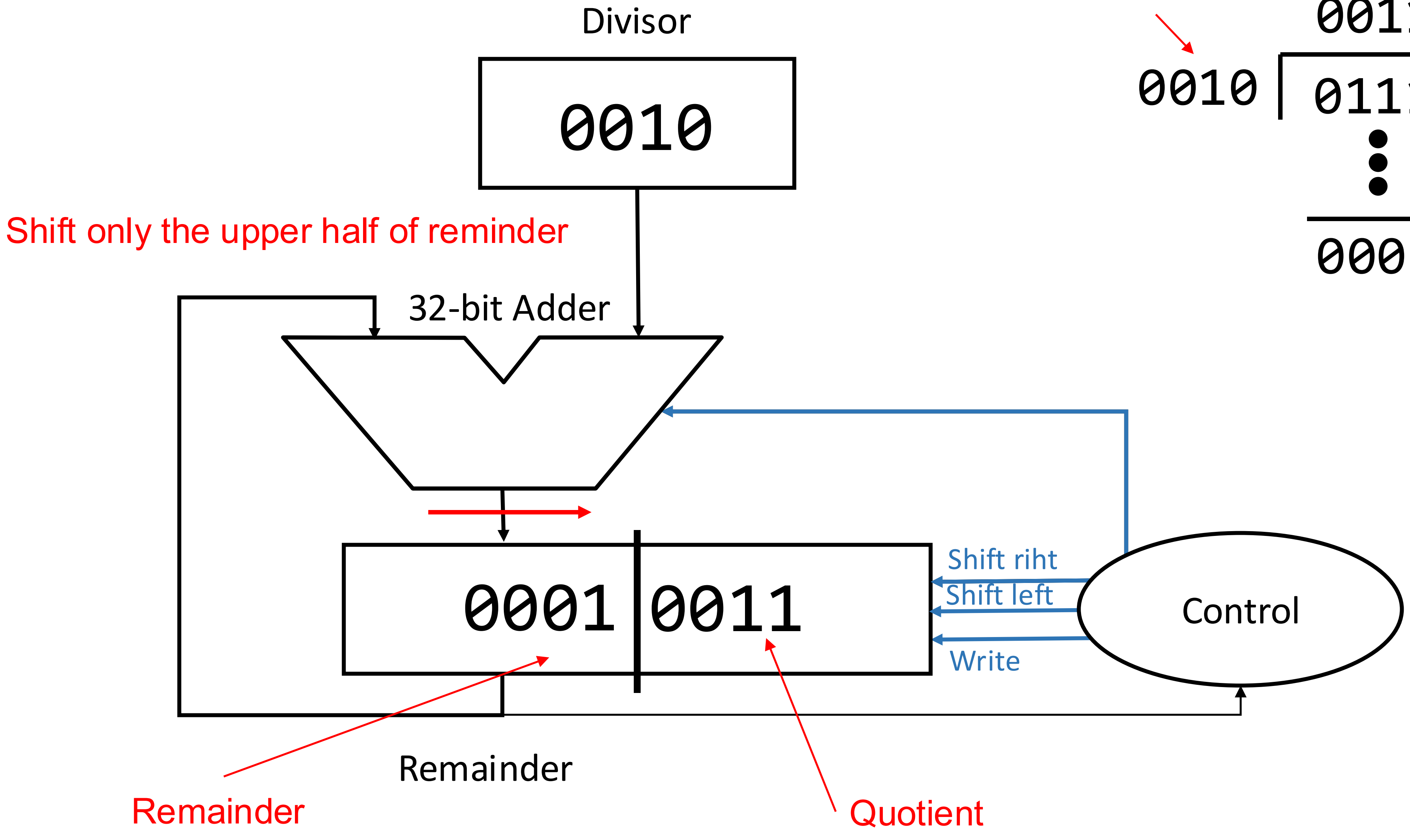
Do not restore the remainder

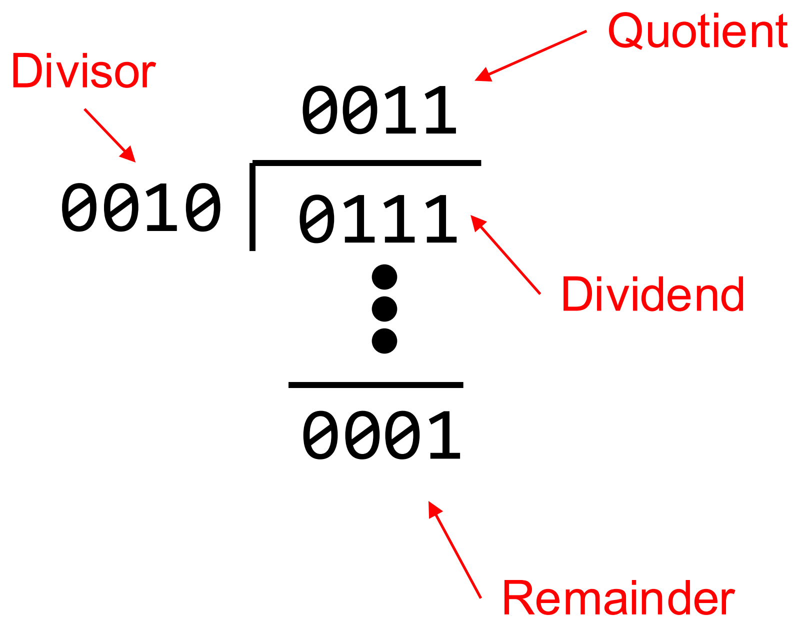
Division Hardware



Do not restore the remainder

Division Hardware





RISC-V Division

Two instructions for integer division and two instructions for remainder to handle both signed and unsigned integers

```
div     rd, rs1, rs2  /   divu    rd, rs1, rs2
```

R[rd] = (R[rs1] / R[rs2])

```
rem     rd, rs1, rs2  /   remu    rd, rs1, rs2
```

R[rd] = (R[rs1] % R[rs2])

RISC-V divide instructions ignore overflow and division-by-0

Software must perform checks if required