

# Arithmetic for Computers: Multiplication and Division

*not HW Design  
(hard) wave*



Z. Jerry Shi

Department of Computer Science and Engineering  
University of Connecticut

CSE3666: Introduction to Computer Architecture

# Outline

---

- Multiplication
  - Multiplication of binary numbers
  - Multiplier (HW)
  - RISC-V multiplication instructions
- Division
  - Division of binary numbers
  - Division hardware
  - RISC-V division instructions

Reading: Sections 3.3 and 3.4

# Multiplication

- Start with long-multiplication approach
  - Similar to multiplication in decimal

A hand-drawn diagram illustrating the long multiplication of two binary numbers, 1000 and 1001. The multiplicand (1000) and multiplier (1001) are at the top. The multiplier's '1' in the second position from the right is circled in green. Below the multiplier, four partial products are shown, each shifted one position to the left: 1000, 0000, 0000, and 1000. The first partial product (1000) is underlined in green. The second and third partial products (0000) are underlined in red. The fourth partial product (1000) is underlined in orange. Arrows point from the labels 'multiplicand' and 'multiplier' to their respective numbers. Another arrow points from the label 'product' to the final result, 1001000, which is underlined in orange. The entire calculation is enclosed in a hand-drawn orange box.

$$\begin{array}{r} 1000 \\ \times 1001 \\ \hline 1000 \\ 0000 \\ 0000 \\ 1000 \\ \hline 1001000 \end{array}$$

← multiplicand  
← multiplier  
← product

Can you describe the steps?

# Align the numbers

- Fill the blanks with 0
  - It is easier to do additions

$$\begin{array}{r} 1000 \\ \times 1001 \\ \hline 1000 \\ 0000 \\ 0000 \\ 1000 \\ \hline 1001000 \end{array}$$

$$\begin{array}{r} 00001000 \\ \times 1001 \\ \hline 00001000 \\ 00000000 \\ 00000000 \\ 01000000 \\ \hline 01001000 \end{array}$$

How can we get these numbers?

Number of bits in product is doubled

# Design an algorithm

- Let us do it in multiple steps, one addition in each step

```
1. product = 0
2. For i in 0 .. n-1 (bits in multiplier)
  2.1 t = multiplicand * multiplier[i]
  2.2 product += t
  2.3 multiplicand <<= 1
```

		Product
	00001000	00000000
×	1001	
	00001000	+ 00001000
	00000000	= 00001000
	00000000	+ 00000000
	00000000	= 00001000
	01000000	+ 00000000
	01001000	= 00001000
		+ 00001000
		= 01001000

# Design an algorithm - 2

- The version on the previous slide

```
1. product = 0
2. For i in 0 .. n-1
  2.1 t = multiplicand * multiplier[i]
  2.2 product += t
  2.3 multiplicand <<= 1
```

- Revised

```
1. product = 0
2. For i in 0 .. n-1
  2.1a t = product + multiplicand
  2.1b If multiplier[i] == 1
    product = t
  2.2 multiplier >>= 1
  2.3 multiplicand <<= 1
```

There is a loop

We can do one iteration per cycle  
Save product, multiplier, and  
multiplicand in registers.

How many bits in each register?

# Multiplication Hardware for 32 bits

```
product = 0
```

```
For i in 0 .. 31
```

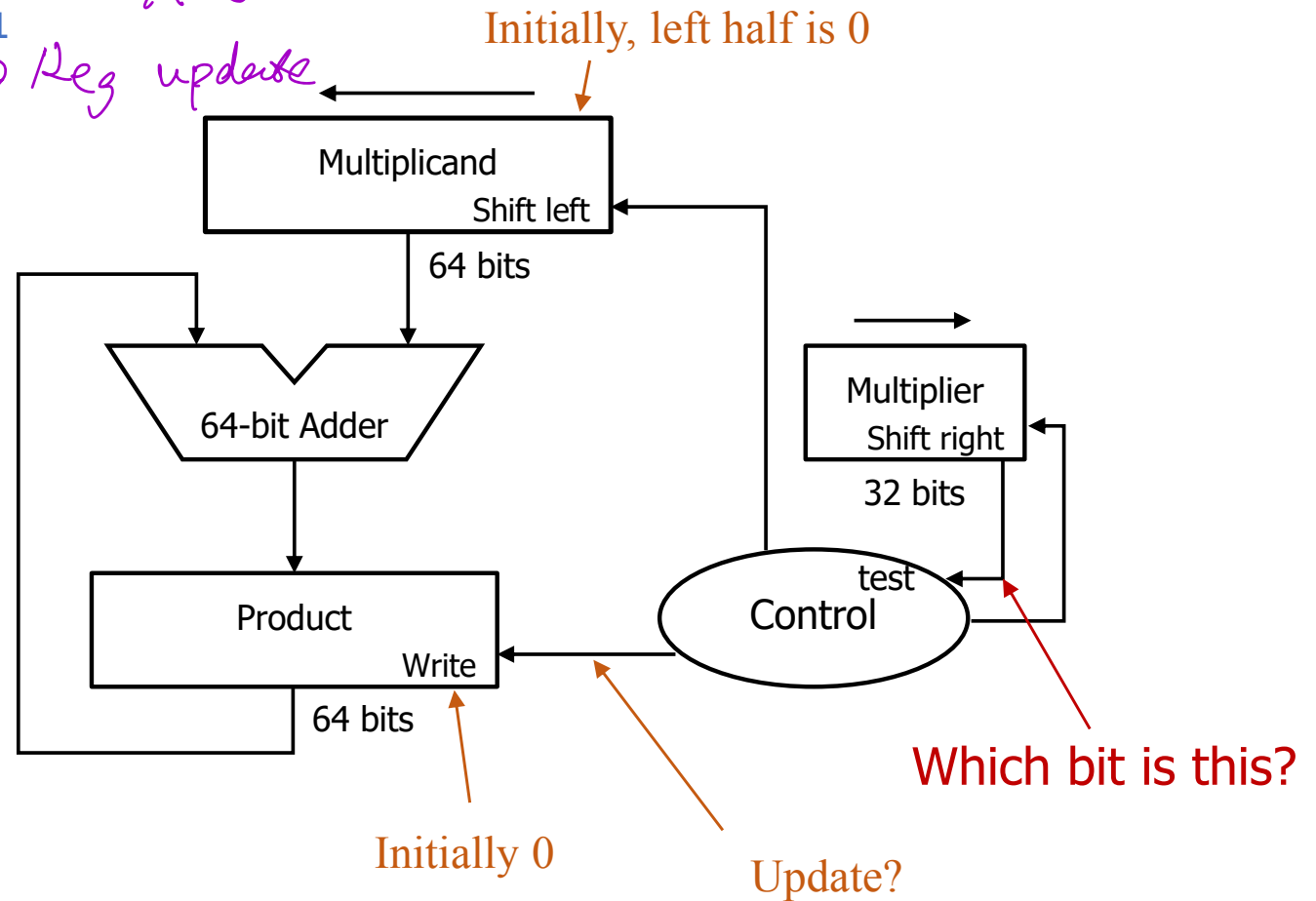
```
  t = product + multiplicand ALU
```

```
  If multiplier[0] == 1
```

```
    product = t → Reg update
```

```
  multiplier >>= 1
```

```
  multiplicand <<= 1
```



Clock and a few other signals are not shown.

# Register values in 4-bit multiplier

Iteration	Multiplicand	Multiplier	Product
→ 0(load)	0000 1000	1001	0000 0000
→ 1	0001 0000	0100	0000 1000
→ 2			
→ 3			
→ 4			

In each iteration:

- Multiplicand is added to product if the LSB of multiplier is 1
- Multiplicand is shifted left (prepare for adding in the next iteration)
- Multiplier is shifted right (discarding the bits already checked)

The values are the ones saved into registers at the beginning of cycles



# Register values in 4-bit multiplier

Iteration	Multiplicand	Multiplier	Product
0(load)	0000 1000	1001	0000 0000
1	0001 0000	0100	0000 1000
2	0010 0000	0010	0000 1000
3	0100 0000	0001	0000 1000
4	1000 0000	0000	0100 1000

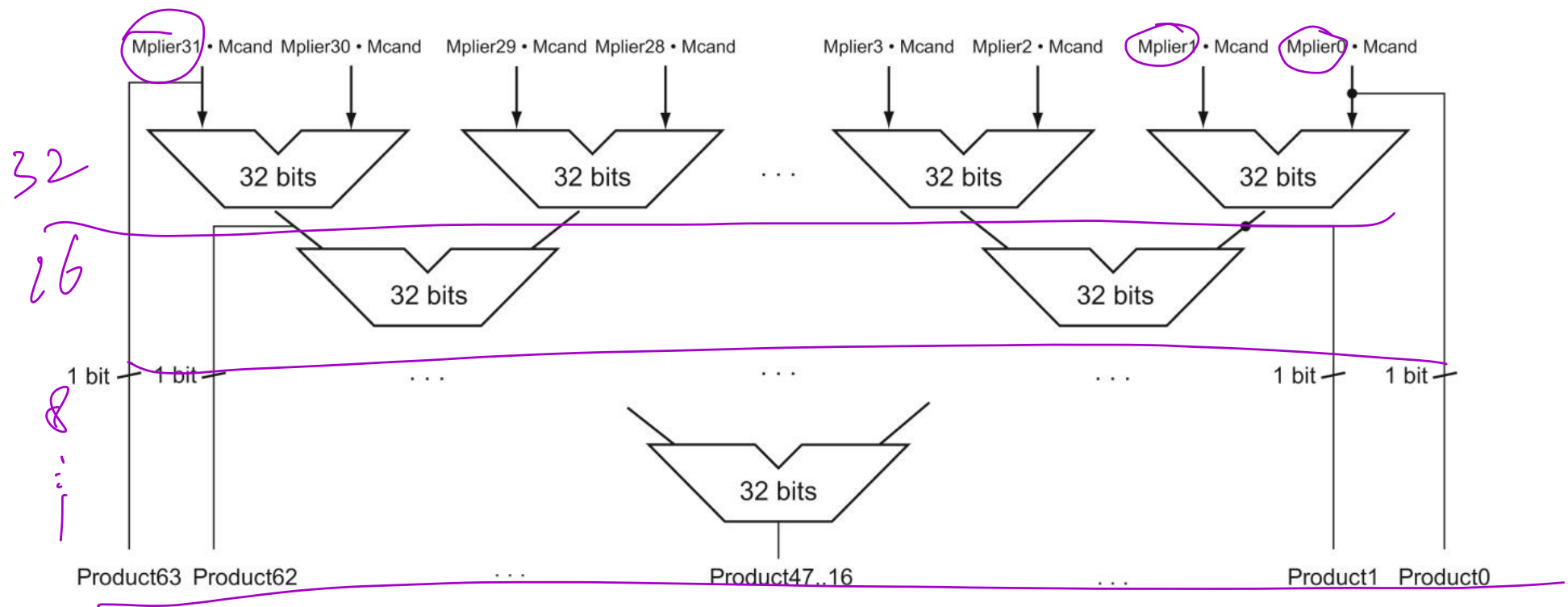
In each iteration:

- Multiplicand is added to product if the LSB of multiplier is 1
- Multiplicand is shifted left (prepare for adding in the next iteration)
- Multiplier is shifted right (discarding the bits already checked)

# Faster Multiplier

- Uses multiple adders
  - Cost/performance tradeoff
- Can be pipelined
  - Several multiplication performed in parallel

5-stage



# Two's complement multiplication

---

- Compute sign separately
- Booth's multiplication algorithm
  - Invented by Andrew Donald Booth in 1951
- Previous methods can work for **the lower half**

# RISC-V Multiplication Instructions

# lower 32 bits of the product

mul

rd, rs1, rs2

add

(64)

# higher 32 bits depend on signs of rs1 and rs2

mulh

rd, rs1, rs2

# both are signed

mulhu rd, rs1, rs2

# both are unsigned

mulhsu rd, rs1, rs2

# rs1 is signed, rs2 is unsigned

# Question

Suppose bits in both s1 and s2 are 0xFFFF FFFF.

Compute the product of s1 and s2 and save the lower 32 bits of the product in s3.

```
mul    s3, s1, s2
```

What is the value in s3? ( )

Show your answer in decimal.

Handwritten diagram illustrating the multiplication of two 32-bit values (0xFFFF FFFF) to produce a 64-bit result (0xFFFFFFFF00000000). The diagram shows the two input values as 1111...1111, their product as 0xFFFFFFFF00000000, and the lower 32 bits of the product highlighted with an arrow and the label "lower 32-bit".

# RISC-V Division Instructions

---

# signed

div rd, rs1, rs2 # rs1 / rs2

rem rd, rs1, rs2 # rs1 % rs2

# unsigned

divu rd, rs1, rs2

remu rd, rs1, rs2

- No divide-by-0 checking
  - Software must perform checks if required

## Example

---

Convert the following pseudocode to RISC-V assembly code.

`s1` is a signed number.

`if s1 is divisible by 7, go to L1`

## Example

---

Convert the following pseudocode to RISC-V assembly code.  
s1 is a signed number.

if s1 is divisible by 7, go to L1

```
addi  t0, t0, 7
rem   t1, s1, t0
beq   t1, x0, L1
```

No need to use div/rem if the divisor is a power of 2



## div and mod with negative numbers

---

$n$  : dividend,  $d$  : divisor,  $q$  : quotient,  $r$  : remainder.

n	d	q	r
7	3	2	1
-7	3	-2	-1
7	-3	-2	1
-7	-3	2	-1

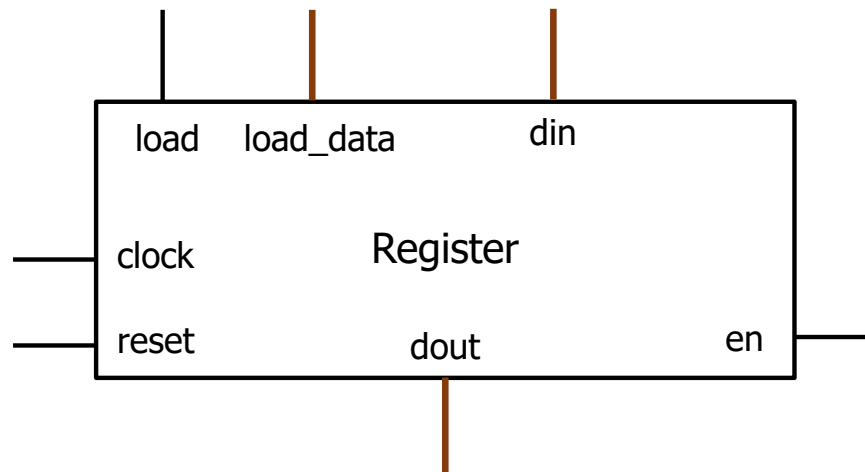
$$-(n / d) = (-n) / d = n / (-d)$$

$r$  always have the same sign as  $n$ .

Adjust in software if you want mathematically correct answers.

# Registers used in lab

---



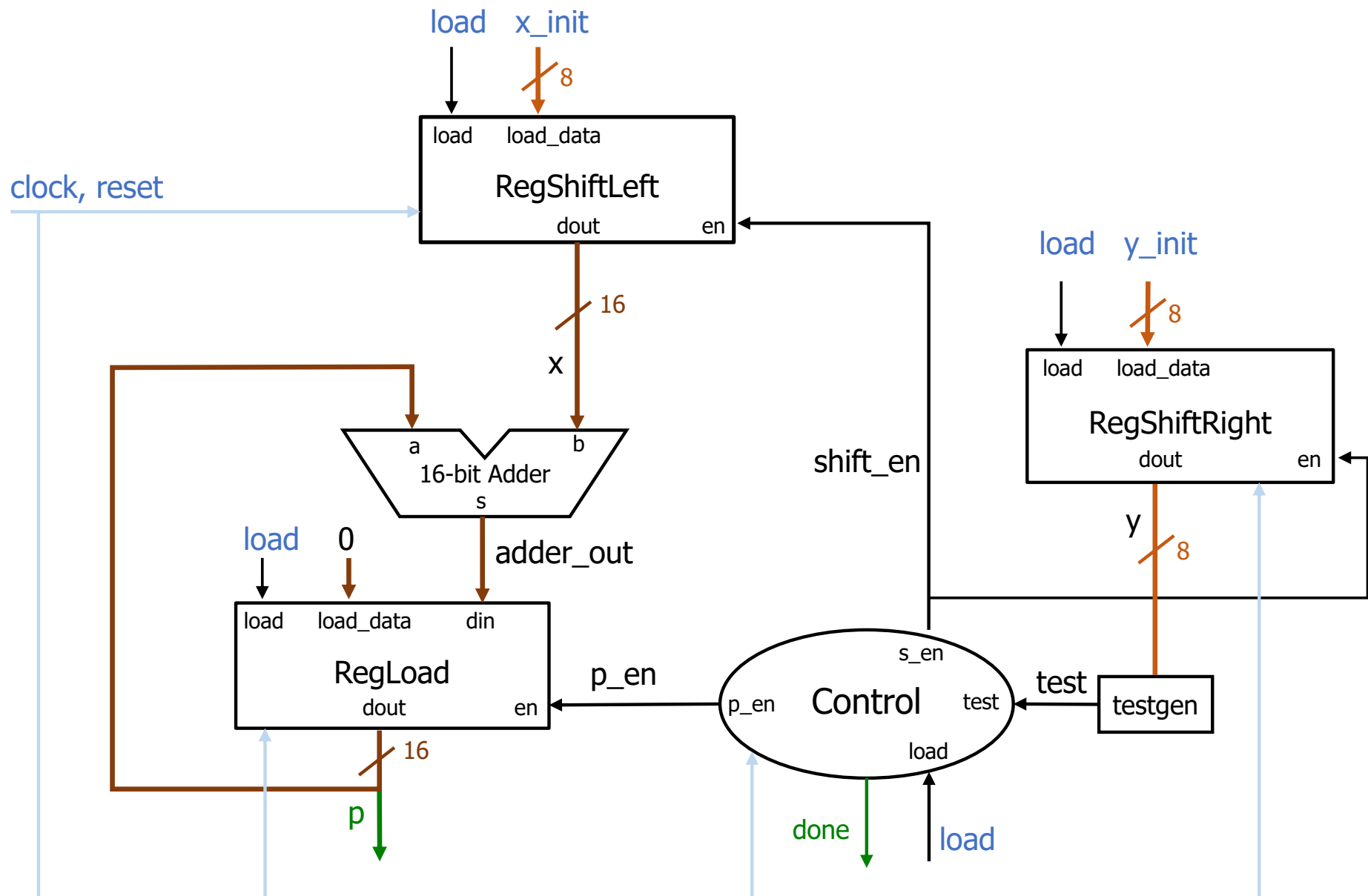
load == 1: load\_data is saved in the register  
load == 0 and enable == 1: din is saved  
load == 0 and enable == 0: dout does not change

Shift left register and Shift right register also have load and load\_data, but do not have din.

If load is 1, load\_data is saved in the register

If load is 0 and enable is 1, dout is shift left (or right)

# Multiplication Hardware for lab



# Division (no Hardware in Exam)

- Long division approach
- If divisor  $\leq$  bits from dividend

Yes

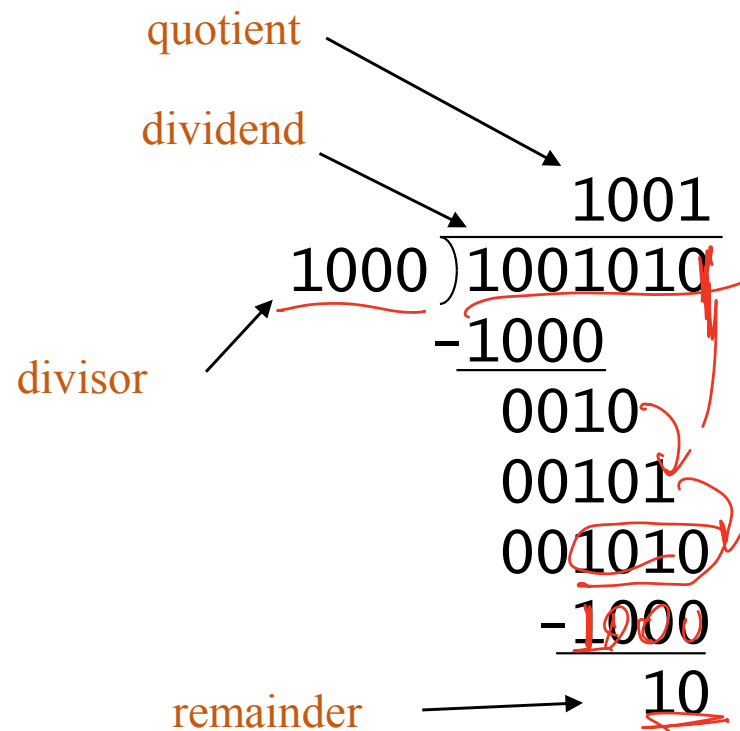
Set quotient bit to 1

Subtract divisor from dividend

No

Set quotient bit to 0

Bring down next bit in dividend



One of the challenges is to align numbers

## Example: 4-bit division

0b0111 / 0b0010

## Numbers compared:

Compared:

0010  $\overline{) 00100111}$

0010

00000111

0010

00000111

0010

00000011

0010

00000001

Annotations:

- Dividend has 8 bits
- 0000 < 0010 no subtraction
- 0001 < 0010 no subtraction
- 0011 > 0010 subtraction is done
- 0011 > 0010 subtraction is done
- Dividend becomes remainder

Subtraction is *performed* only when dividend  $\geq$  divisor  
Quotient bit is set to 1 in these cases