Arithmetic for Computers: Multiplication and Division

~80% ~20%

Caiwen Ding
Department of Computer Science and Engineering
University of Connecticut

CSE3666: Introduction to Computer Architecture

Outline

- Multiplication
  - Multiplication of binary numbers
  - Multiplier
  - RISC-V multiplication instructions

- Division
  - Division of binary numbers
  - RISC-V division instructions

Reading: Sections 3.3 and 3.4

Multiplication

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

```
       1000     multiplicand   1
  ×    1001     multiplier     2
  ---------
       1000
      0000   +
     0000    +
    1000
  ---------
    1001000     product        3
```

Can you describe the steps?

Align the numbers

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

```
     1000              00001000
×    1001         ×        1001
---------          ------------
     1000              00001000
    0000               00000000
   0000                00000000
  1000                 01000000
---------          ------------
  1001000              01001000
```

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

LSb

MSb

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

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

Design an algorithm - 2

- Revise it

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

Do one iteration a cycle
Save product, multiplier,
and multiplicand in registers

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

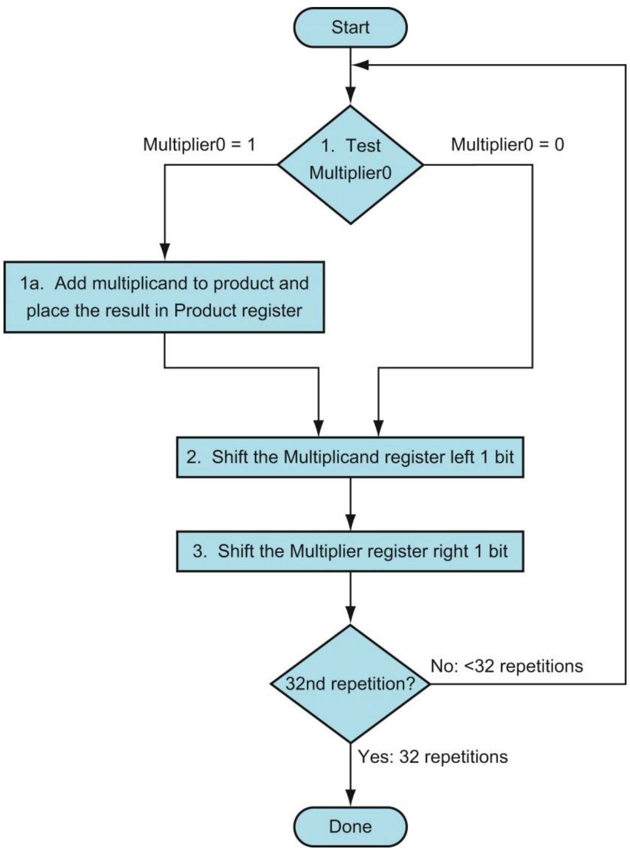
LSb

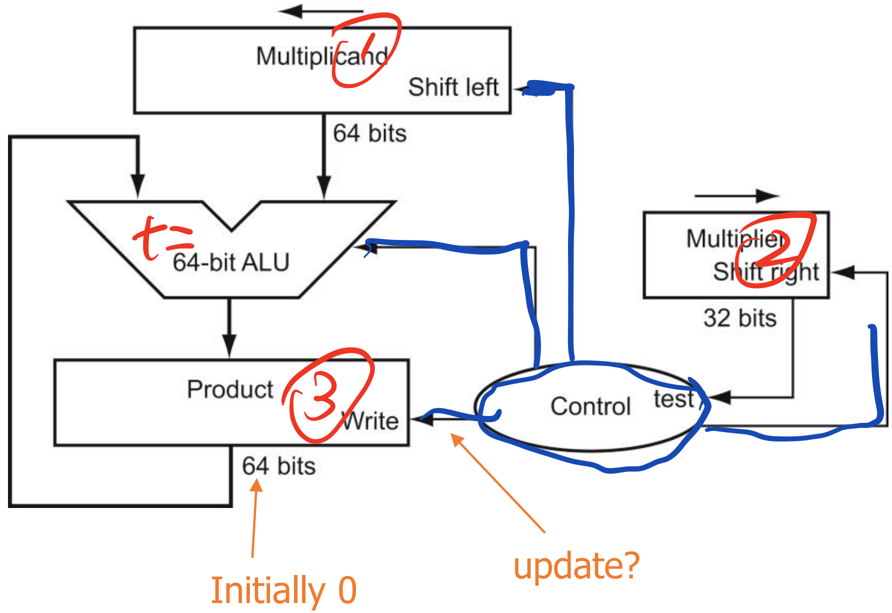
How many bits in each register?

→ check 1 bit

Multiplication Hardware for 32 bits

32-bit { adder
mult





Register values in 4-bit multiplier

Iteration	Multiplicand	Multiplier	Product
0	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

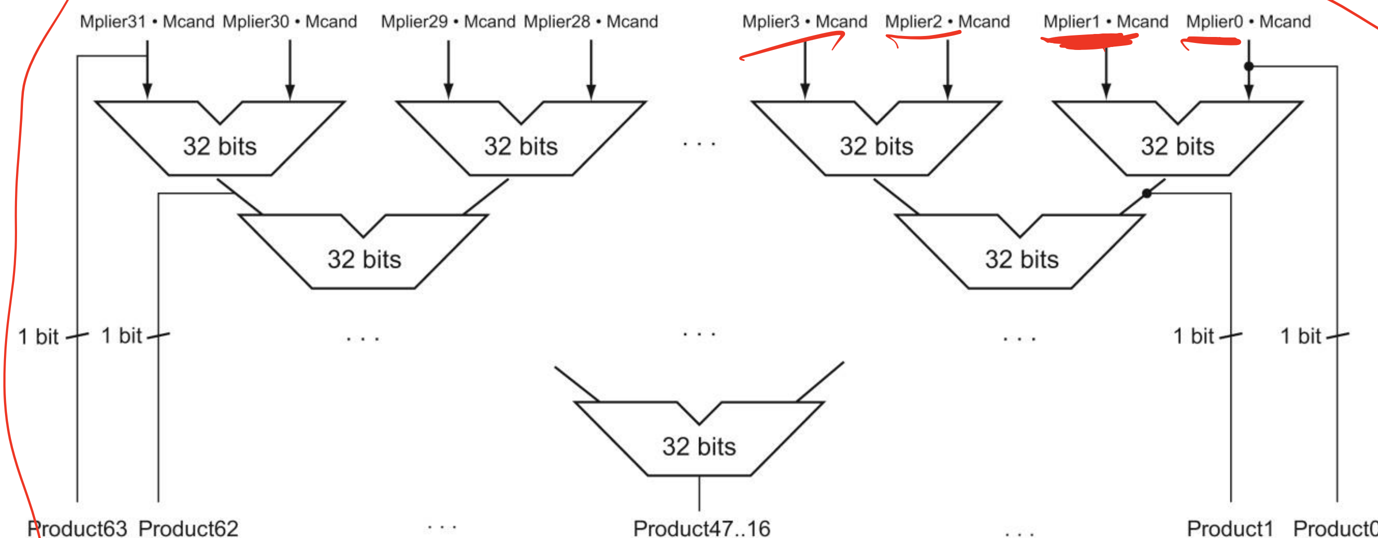
Iteration	Multiplicand	Multiplier	Product
0	0000 1000	1001	0000 0000
1	0001 0000	0100	0000 1000
2	0010 0000	0010	0000 1000
3	0100 0000	0001	0000 1000
4	1000 0000	0000	0100 1000

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



+16
+8
+4
+2
+1

Two’s complement multiplication Not exam

- 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



# higher 32 bits depend on signs of rs1 and rs2

                          rs1, 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

signed

$F = 2^4 - 1 = 15$

$FF = 2^8 - 1$

Suppose bits in both s1 and s2 are 0xFFFF FFFF. (−1) (signed)

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

$2^{32} - 1$ (unsigned)

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

What is the value in s3? 1

Show your answer in decimal.

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, x0, 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.

inp out

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.

Division

- Long division approach

- If divisor ≤ 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

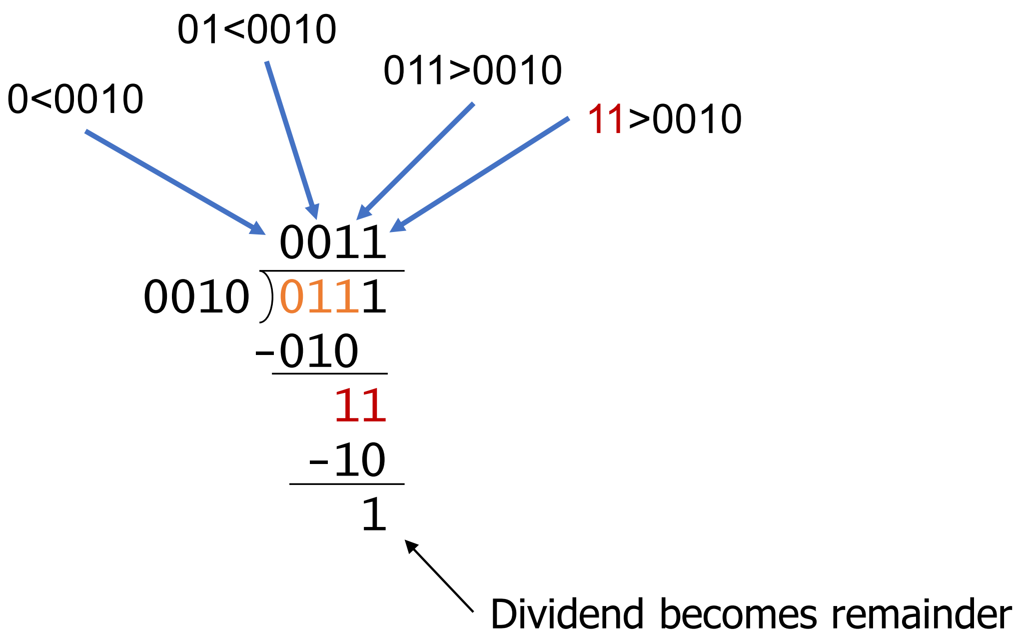
```
quotient
dividend              1001
divisor   1000 )1001010
                -1000
                 0010
                 00101
                 001010
                  -1000
remainder            10
```

One of the challenges is to align numbers

Example: 4-bit division

0b0111 / 0b0010

Numbers compared:



Subtraction is *performed* only when dividend >= divisor
Quotient bit is set to 1 in these cases