# Signed Multiplication

## Booth's Algorithm

**STEP 1:** Depending on current & prev. bits do following

- 00: No arithmetic operation
- 01: Add mcand to left half of product
- 10: Subtract mcand from left half of product
- 11: No arithmetic operation.

**STEP 2:** Shift product register right 1-bit
(Extend sign when product is shifted)
→ arithmetic right shift

initially product has multiplier
↳ 0 prev. bit

Example: $2x-3 = \overbrace{0010}^{\text{mcand}} \times \overbrace{1101}^{\text{multiplier}}$

| I | step | Mcand | product |
|---|---|---|---|
| ∅ | initial values | 0010 | 0000 1101 ↳ prev=∅ |
| 1 | prod = prod - mcand | | 0000 1101<br>- 0010<br>1110 1101 ← |

prod → 1111 0110 ↳ 1

---

2 prod = prod + mcand

prod →

```
  1111 0110
+ 0010
-----------
→ 0001 0110

  0000 1011   ↳ 0
```

---

3 prod = prod - mcand → prod

```
  0000 1011
- 0010
-----------
  1110 1011

  1111 0101   ↳ 1
```

---

4 No arithm. opt.

prod →

-6

1111 1010

---

1111 1010 → 0000 0101 → 0000 0110

(6)

# Multiply in MIPS

MIPS has 2 instr → mult
→ multu (multiply unsigned)

To fetch integer 32-bit product
→ mflo → move from lo
→ mfhi → move from hi

Ex:
```
mult  $s1, $s2   # Result hi-lo
mflo  $s3        # $s3 ← lo
mfhi  $s4        # $s4 ← hi
```

```
mul   $s5, $s2, $s3   ⇒ pseudo instr
```

Translates →
```
mult  $s2, $s3
mflo  $s5
```

$s1 x $s2 ⇒ ⇒ [ hi (32) | lo (32) ] → s3
$s4

## Division

Dividend | Divisor
---|---
| Quotient
Remainder

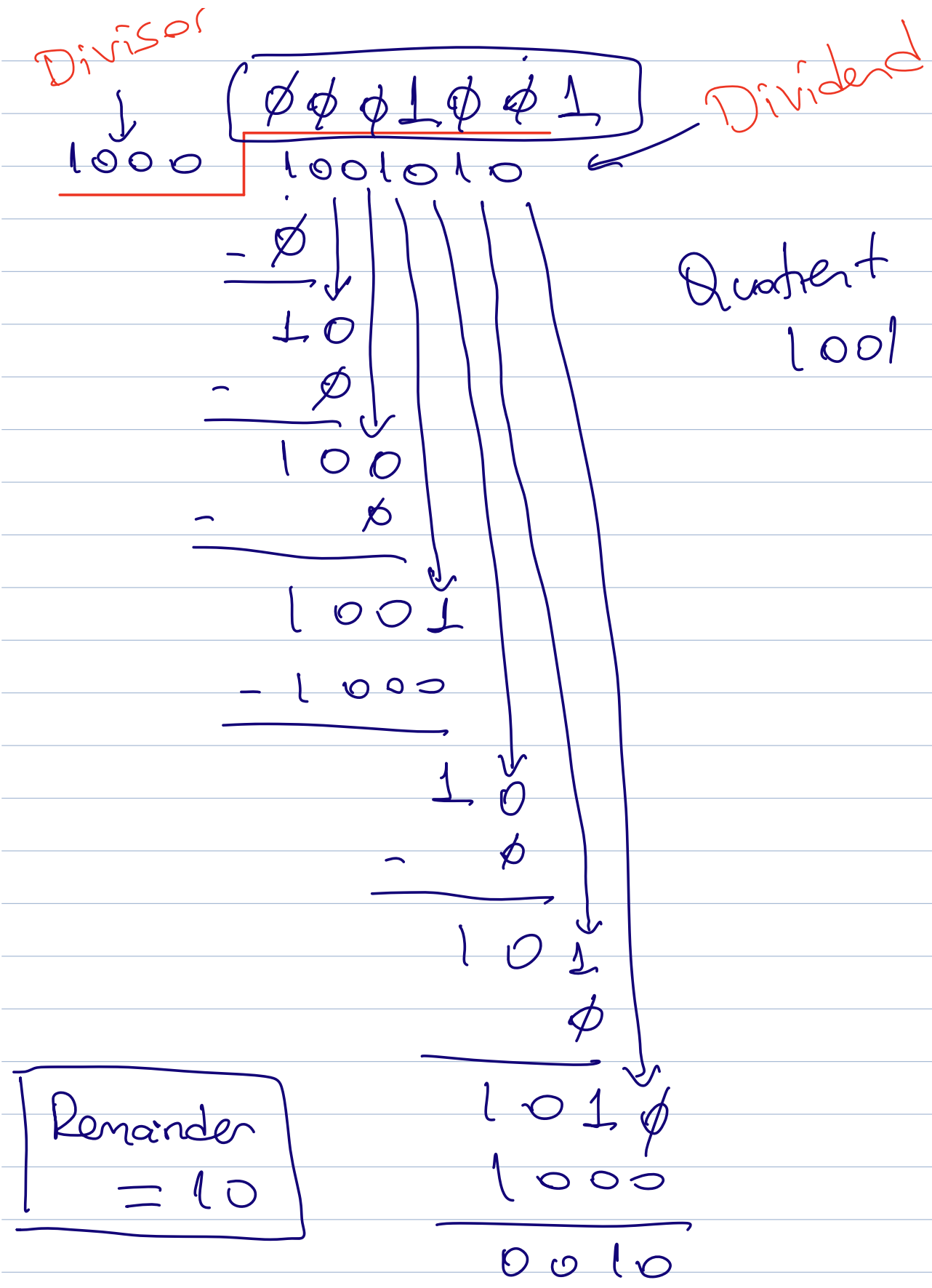
Dividend = Diviser × Quotient + Remander

## Binary Division

Divided → 1001010
Divisor → 1000

Main idea

* how big a number can be subtracted from
* create a digit of the quotient on each attempt.


Divisor
1000
00010001
Dividend
1001010
- 0
10
- 0
100
- 0
1001
- 1000
10
- 0
101
0
1010
1000
0010
Quotient
1001
Remander
= 10

# First Method (Slide 14)

32-bit Quotient set to 0

Each iteration → move divisor right one bit

Initially

* divisor placed left half of 64-bit divisor register
* remainder is initialized with dividend.

Example: Divide 7 by 2.

0000 0111 → dividend

0010 → divisor

| I (→ iteration) | Step | Quotient | Divisor | Remainder |
|---|---|---|---|---|
| 0 | initial values | 0000 | 0010 0000 | 0000 0111 |
| 1 | Rem = Rem − Divisor | | | 0000 0111<br>− 0010 0000<br>1110 0111 |

| | | | |
|---|---|---|---|
| Restore & ← Quotient → Divisor | 0000 | 0001 0000 | 0000 0111 |
| 2 Rem = Rem − Divisor<br>Rem < 0<br>Restore ← Quotient → Divisor | 0000 | 0000 1000 | 0000 0111<br>− 0001 0000<br>1111 0111<br>0000 0111 |
| 3 Rem = Rem − Divisor<br>Rem < 0<br>Restore ← Quotient → Divisor | 0000 | 0000 0100 | 0000 0111<br>− 0000 1000<br>1111 1111<br>0000 0111 |
| 4 Rem = Rem − Divisor ← Quotient (LSB=1) → Divisor | 0001 | 0000 0010 | 0000 0111<br>− 0000 0100<br>0000 0011 |

| | | | | |
|---|---|---|---|---|
| 5 | Rem = Rem − Divisor<br>←<br>Quotient (LSB=1) | 0011<br>. | 0000 0001 ✓ | 0000 0011<br>− 0000 0010<br>0000 0001 |

Quotient = 0011

Remainder = 0000 0001

Algorithm Requires (n+1) steps

Problems & Solutions

* Half of divisor and ALU used (cut size in Half)

* Shift remainder left instead of shift divisor to right

(same alignment & simplify the hardware for ALU)

Quotient eliminated & combine with right half of remainder

# Floating Point Representation

Scientific Notation = single digit to the left of the decimal point.

Normalized = A number in scientific notation that has no leading 0's

$1.0 \times 10^{-9}$ normalized number

$0.1 \times 10^{-8}$
$10.0 \times 10^{-10}$
) not a normalized number

## IEEE Floating-Point Format

Exponent before significant
→ simplify sorting of floating point numbers using integer integer instr.

Negative exp ⇒ challeye to simplified sorting
2's complerent → look like a big number
neg. exponent

## Desirable Notation :

Most neg. exponent $(\emptyset\emptyset\ldots\emptyset)_2$

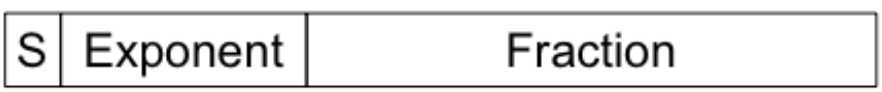
Most positive exp. $(11\ldots1)_2$

Biased Notation: 127 , 1023

↙ Bias for single precision

↓ Bias for double precision

single: 8 bits single: 23 bits
double: 11 bits double: 52 bits




$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

If we number the bits of the fraction from left to right s1, s2, s3 ... then, the value is:

$$(-1)^S \times \left[1 + (s1 \times 2^{-1}) + (s2 \times 2^{-2}) + (s3 \times 2^{-3}) \ldots\right] \times 2^{(exp - bias)}$$

**Question** $-0.75 \rightarrow$ To Binary (Single Precision)

$0.75 \times 2 = 1 + 0.50$

$0.50 \times 2 = 1 + 0.00$

$= 0.11$ → $2^{-2}$

↳ $2^{-1}$

$-0.75 = (-1)^1 \times 0.11$

$= (-1)^1 \times 1.1 \times 2^{-1}$

Bias: $126 - 127$

$= (-1)^1 \times 1.1 \times 2^{126-127}$

Fraction = $100\ldots0$ (23 bits)

exp = $01111110$ (8 bits)

SINGLE = 1 0 1111 1110 1000 ... 0

0.4375 ⇒ Binary

$0.4375 \times 2 = 0 + 0.875$

$0.875 \times 2 = 1 + 0.75$

$0.75 \times 2 = 1 + 0.50$

$0.50 \times 2 = 1 + 0.0$

$0.4375 = 0.0111$ ($2^{-1}$, $2^{-2}$, $2^{-3}$, $2^{-4}$)

$= 1/4 + 1/8 + 1/16$

$= 0.25 + 0.125 + 0.0625$

$= 0.375 + 0.0625$

$= 0.4375$