Chapter 3 - Arithmetic for Computers

1. Integer

1.1. Overflow

1.1.1. Overflow conditions

Add

- `(+) add (-)` –> No overflow
- `(+) add (+)` , <u>result sign</u> is 1 -> **Overflow**
- `(-) add (-)` , <u>result sign</u> is 0 -> **Overflow**

Subtract

- (+) subtract (+), or (-) subtract (-) -> No overflow
- (+) subtract (-), <u>result sign</u> is 1 -> **Overflow**
- (-) subtract (+), <u>result sign</u> is 0 -> **Overflow**

1.1.2. How MIPS deals with overflow

<u>on overflow:</u>

- save PC in EPC (exception program counter) register (for returning later)
- jump to predefined handler address (to handle this overflow, if any)
- use `mfc0` instruction to retrieve EPC value to return after corrective action

1.2. Multiplication

MIPS allows multiplication of two 32-bit registers, the product is stored into 2 registers:

- Upper part (aka <u>most significant</u> 32 bits): HI
- Lower part (aka <u>least significant</u> 32 bits): LO

Formula: `mult rs, rt`

To read result (2 x 32-bit registers)

- `mfhi rd` : move HI to rd
- `mflo rd` : move LO to rd

<u>Note</u>: `mul rd, rs, rt` does work too, but the result only stores least-significant 32-bit (i.e. LO)

1.3. Division

MIPS also uses HI/LOW for division but in a different way:

- HI: stores the remainder, in 32-bit.
- LO: stores the quotient, in 32-bit.

Formula: `div rs, rt`

1.4. Floating Point

1.4.1. Scientific Notation

Normalzied form

- Decimal value must be in the form of $\pm x.yyy * 10^{zzz}$. There must be only 1 digit on the left of the floating point.

- Binrary value must be in the form of $\pm 1.xxxxxx_2 * 2^{yyyy}$. There must be value '1' on the left of the floating point.

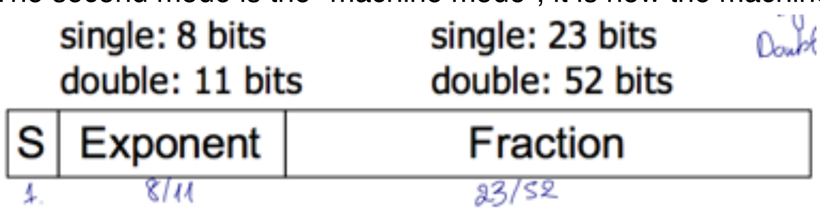
IEEE Std 754-1985

There are 2 "modes" (personal opinions) to represent binary floating number:

- The first mode is the "human mode", it is how we present the number in writing, and it is the normalized form described above: $\pm 1.xxxxxx_2 * 2^{yyyy}$

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

- The second mode is the "machine mode", it is how the machine reads and operates on floating point.



1.4.1.1. To convert from "human mode" to "machine mode":

- Make sure the value in "human mode" **is already normalized** before proceed.
- Record Sign bit (S) as 1 if this is a negative number, 0 if positive.
- The **fraction** is taken from the matissa of the floating number in "human mode". For e.g. fraction of 1.10101 is 10101. This goes straight to the Fraction bit section, starts from **LEFT MOST BIT TO THE RIGHT** on bit 10 for single precision, or bit 13 for double precision.
- The Exponent value of the bit section is the **binary value of the Exponent in the human mode, plus the Bias**, which is 127 for Single Precision, and 1023 for Double Precision.

Invert the process to get "human mode" writing from "machine mode" floating binary.

1.4.1.2. Smallest and Largest in Single-precision

Exponents 00000000 and 11111111 reserved

NOT ALLOWED

Smallest value

in machine code

in IEEE format

- Exponent: 00000001
  ⇒ actual exponent = 1 – 127 = –126
- Fraction: 000…00 ⇒ significand = 1.0
- $1.0 \times 2^{-126} \approx 1.2 \times 10^{-38}$

Largest value

- exponent: 11111110
  ⇒ actual exponent = 254 – 127 = +127
- Fraction: 111…11 ⇒ significand ≈ 2.0
- $2.0 \times 2^{+127} \approx 3.4 \times 10^{+38}$

1.4.1.3. Smallest and Largest in Double-precision

Exponents 0000…00 and 1111…11 reserved

Smallest value

- Exponent: 00000000001
  ⇒ actual exponent = 1 – 1023 = –1022
- Fraction: 000…00 ⇒ significand = 1.0
- $1.0 \times 2^{-1022} \approx 2.2 \times 10^{-308}$

Largest value

- Exponent: 11111111110
  ⇒ actual exponent = 2046 – 1023 = +1023
- Fraction: 111…11 ⇒ significand ≈ 2.0
- $2.0 \times 2^{+1023} \approx 1.8 \times 10^{+308}$

1.4.2. Practice

1.4.2.1. Convert Base 10 Floating number to IEEE Binary

Represent –0.75

significand. (if an asks to digits → 1.100)

- $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$ ← Manul calculation
- sign.
- S = 1
- Fraction = $1000\ldots00_2$
- Exponent = –1 + Bias
  - Single: –1 + 127 = 126 = 01111110_2
  - Double: –1 + 1023 = 1022 = 01111111110_2

Single: 1011111101000…00 ← Store in Mem

Double: 1011111111101000…00

1.4.2.2. Convert IEEE Binary to Base 10 Floating number

What number is represented by the single-precision float

exp.

1 10000001 01000…00

- S = 1
- Fraction = 01000…00_2 ⇒ 1.01
- Fxponent = 10000001_2 = 129 → 129 − 127 = 2.

$= (-1)^1 \times 1.01 \times 2^2$

1.01_2.

$$x = (-1)^1 \times (1 \cancel{+ 01_2}) \times 2^{(129-127)}$$

$$= (-1) \times 1\cancel{.25 \times 2^2} \quad 101_2 \times 2^0$$

$$= -5.0$$

to convert $(-1)^1 \times 1.01 \times 2^2$ to base 10
⇒ change $2^2 \to 2^0$, shift $1.01 \to 101_2$.
$101_2 = 5$ ⇒ add the sign → −5

1.4.2.3. Addition of Floating Point numbers in Base 10

Consider a 4-digit decimal example

- $9.999 \times 10^1 + 1.610 \times 10^{-1}$

1. Align decimal points $\boxed{\text{smaller to bigger one.}}$
   - Shift fractional number on smaller exponent
   - $9.999 \times 10^1 + 0.016 \times 10^1$
2. Add significands
   - $9.999 \times 10^1 + 0.016 \times 10^1 = 10.015 \times 10^1$
3. Normalize result & check for over/underflow
   - 1.0015×10^2
4. Round and renormalize if necessary
   - 1.002×10^2

1.4.2.4. Addition of Floating Point numbers in Base 2

Now consider a 4-digit binary example

- $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ (0.5 + –0.4375)

1. Align binary points

- Shift number with smaller exponent
- $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$

2. Add significands

- $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$

3. Normalize result & check for over/underflow

- $1.000_2 \times 2^{-4}$, with no over/underflow

4. Round and renormalize if necessary

- $1.\underbrace{000_2 \times 2^{-4}}$ (no change) = 0.0625

Convert floating binary number to base 10

$1.\underbrace{000_2 \times 2^{-4}}$ (no

$1.000_2 \times 2^{-4}$

	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Value:	1	0.5	0.25	.125	.0625
Bits:	0	0	0	0	1

↑

$\Rightarrow 1.000_2 \times 2^{-4} = 0.0625$

1.4.2.5. Multiplication of Floating Point numbers in Base 10

- Consider a 4-digit decimal example
  - $1.110 \times 10^{10} \times 9.200 \times 10^{-5}$
- 1. Add exponents
  - For biased exponents, subtract bias from sum
  - New exponent = 10 + –5 = 5
- 2. Multiply significands
  - $1.110 \times 9.200 = 10.212 \Rightarrow 10.212 \times 10^{5}$
- 3. Normalize result & check for over/underflow
  - 1.0212×10^{6}
- 4. Round and renormalize if necessary
  - 4 digits 1.021×10^{6}
- 5. Determine sign of result from signs of operands
  - $+1.021 \times 10^{6}$

1.4.2.6. Multiplication of Floating Point numbers in Base 2

Now consider a 4-digit binary example

- $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$ (0.5 × –0.4375)

1. Add exponents
   - Unbiased: –1 + –2 = –3
   - Biased: (–1 + 127) + (–2 + 127) = –3 + 254 – 127 = –3 + 127
2. Multiply significands
   - $1.000_2 \times 1.110_2 = 1.110_2 \Rightarrow 1.110_2 \times 2^{-3}$
3. Normalize result & check for over/underflow
   - $1.110_2 \times 2^{-3}$ (no change) with no over/underflow
4. Round and renormalize if necessary — if qn does not ask, we keep 4 digit.
   - $1.110_2 \times 2^{-3}$ (no change)
5. Determine sign: +ve × –ve ⇒ –ve
   - $-1.110_2 \times 2^{-3} = -0.21875$

1.4.3. Load/Store Floating Point Instructions in MIPS

Using different instructions:

- `lwc1`, `ldc1`, `swc1`, `sdc1`
  - e.g., `ldc1 $f8, 32($sp)`

load word floating.

and different registers:

32 single-precision: \$f0, \$f1, … \$f31

1.4.4. Other Operational Floating Point Instructions

Single-precision arithmetic
- `add.s`, `sub.s`, `mul.s`, div.s
  - e.g., `add.s $f0, $f1, $f6`

Double-precision arithmetic
- `add.d`, `sub.d`, `mul.d`, `div.d`
  - e.g., `mul.d $f4, $f4, $f6`

Single- and double-precision comparison
- `c.`*xx*`.s`, `c.`*xx*`.d` (*xx* is eq, `lt`, `le`, …)
- Sets or clears FP condition-code bit
  - e.g. `c.lt.s $f3, $f4`

Branch on FP condition code true or false
- `bc1t`, `bc1f`
  - e.g., `bc1t TargetLabel`

2. Other Notes

- Shift Left/Right only works correctly on unsigned numbers, **NOT** signed number.
- MIPS ISA: uses mostly 54 core instructions, the rests are less frequent.

3. Excercise

1. Use the MIPS algorithm in this chapter to get the binary sum for these two decimal numbers, verify your result is correct. $5_{10}+9_{10} = ?_2$

2. Use the MIPS algorithm in this chapter to get the binary difference for these two decimal numbers, verify your result is correct. $5_{10}-9_{10} = ?_2$

3. Use the MIPS algorithm in this chapter to get the binary product for these two decimal numbers, verify your result is correct. $5_{10} \times 9_{10} = ?_2$

4. Use the MIPS algorithm in this chapter to get the binary quotient and binary remainder for these two decimal numbers, verify your result is correct. $9_{10}/2_{10} = ?_2$

1. Use the IEEE754 binary floating point format to show the decimal number "-1.37510" in single and double precision binary floating point format.
2. What decimal number is represented by this IEEE754 single precision floating number ?
   1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 02
3. Add following two decimal floating numbers together, use the binary normalized scientific notation and show the steps, keep the result significand in 4 digits, the exponent in two decimal digits: -0.810 0.62510
4. Multiply following two decimal floating numbers together, use the binary normalized scientific notation and show the steps, keep the result significand in 4 digits, the exponent in two decimal digits: -0.810 0.62510