# Computer Organization

## Lab7 MIPS(6) - Floating-Point Processing

2021 Spring term

wangw6@sustech.edu.cn

# Topics

- **Floating-Point Number**
  - IEEE 745 On Floating-point Number
- **Registers of Coprocessor 1**
- **Floating-point Instructions**
  - Load & Store, Move
  - Computational
  - Relational and Branch ...
- **Practices**

# IEEE 745 On Floating-Point Number

$$\pm 1.xxxxxxx_2 \times 2^{yyyy}$$

负数补码

| | single: 8 bits<br>double: 11 bits | 运算量小于求 single: 23 bits<br>double: 52 bits |
|---|---|---|
| S | Exponent (yyyy+Bias) | Fraction (xxxx) |

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

**For single-precision float data:**

**Exponents 0000_0000 and 1111_1111 reserved**

**Bias in Exponent is: 0111_1111**

**For double-precision float data:**

**Exponents 000_0000_0000 and 111_1111_1111 reserved**

**Bias in Exponent is: 011_1111_1111**

# IEEE 745 On Floating-Point Number continued

a 10
12
4+8

```
.data
fneg1:      .float  -1
wneg1:      .word -1
fpos1:      .float   1
wpos1:      .word  1
```

1.0000000 1100
19
4

$\pm 1.xxxxxxx_2 \times 2^{yyyy}$

single: 8 bits
double: 11 bits

single: 23 bits
double: 52 bits

| S | Exponent (yyyy+Bias) | Fraction (xxxx) |
|---|---|---|

$(1+2^{-20}+2^{-21})$
$\times 2^{-127}$

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

(1+
$2^{-127}$
1.0000000000000000000000C $\times 2^{-127}$

| Label | Address ▲ |
|---|---|
| **float_rw.asm** | |
| fneg1 | 0x10010000 |
| wneg1 | 0x10010004 |
| fpos1 | 0x10010008 |
| wpos1 | 0x1001000c |

- -1 = $(-1)^1 \times (1+0) \times 2^0$
  s: 1, exponent: 0 + 0111_1111, fraction: 0
- 1 = $(-1)^0 \times (1+0) \times 2^0$
  s: 0, exponent: 0 + 0111_1111, fraction: 0

0+127
0011 1111 1000 0000 ...

**Data Segment**

| Address | Value (+0) | Value (+4) | Value (+8) | Value (+c) |
|---|---|---|---|---|
| 0x10010000 | 0xbf800000 | 0xffffffff | 0x3f800000 | 0x00000001 |

1011 1111 1000 0000 0000 ......

0+127



# Coprocessor 1 in MIPS

What's the difference between lwc1and ldc1?
Which demo would trigger the exception?
Which demo would get the right answer?

高比特段
奇数寄存器

| Name | Float |
|---|---|
| $f0 | 0x00000000 |
| $f1 | 0xbf800000 |
| $f2 | 0x00000000 |
| $f3 | 0x3f800000 |

Registers | Coproc 1 | Copro

```
Runtime exception at 0x00400004: first register must be even-numbered
```

```
Runtime exception at 0x00400010: all registers must be even-numbered
```

```
.data                #demo1
      fneg1:    .float -1
      fpos1:    .float 1
.text
      lwc1 $f1,fneg1
      lwc1 $f3,fpos1
      add.s $f12,$f1,$f3

      li $v0,2
      syscall
      li $v0,10
      syscall
```

浮点必须
放在 coproc1
的寄存器中

✓

```
.data                #demo2
      fneg1:    .double -1
      fpos1:    .double 1
.text
      ldc1 $f1,fneg1
      ldc1 $f3,fpos1
      add.d $f12,$f1,$f3

      li $v0,3
      syscall
      li $v0,10
      syscall
```

double

在$f12中找data

✗

```
.data                #demo3
      fneg1:    .double -1
      fpos1:    .double 1
.text
      ldc1 $f0,fneg1
      ldc1 $f2,fpos1
      add.d $f11,$f0,$f2

      li $v0,3
      syscall
      li $v0,10
      syscall
```

✗

# Floating-Point Instructions

The **floating-point coprocessor** has these classes of **instructions**:

- **Load and Store** Instructions: Load values and move data between memory and coprocessor registers.

  **lwc1**,**ldc1**; **swc1**,**sdc1**; ...etc

- **Move** Instructions: Move data between registers.

  Conditional Flag 0 同在C1中

  **mtcl**, **mfc1**; VS **mov.s**,**mov.d**; ...etc

  c.eq.d r1, r2

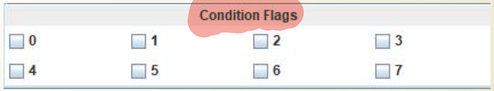
  c.eq.d 1, r1, r2

- **Computational** Instructions: Do arithmetic and logical operations on values in coprocessor registers. 按位运算.逻辑运算不分整与浮.

  add.s,add.d; sub.s,sub.d; mul.s,mul.d; div.s,div.d; ...etc

- **Relational** Instructions: **Compare** two floating-point values and set conditional flag

  c.eq.s, c.eq.d; c.le.s,c.le.d; c.lt.s,c.lt.d; ...etc




- **Conditional jumping** instructions:

  不加数字就判断 flag 0.

  bc1f # conditional jump while conditional flag is 0(false)

  bc1t; # conditional jump while conditional flag is 1 (true)

- **Convert Instructions: floor.w.d, floor.w.s; ceil.w.d, ceil.w.d; cvt.d.s** not a number

# Infinite vs NaN (floating-point)

Which one will get an infinite value?

Which one will get the NaN?

| | Sign (31) | Exponent (30–23) | Mantissa (22–0) |
|---|---|---|---|
| 93000000 | 0 | 0001 1010 | 101 1000 1011 0001 0001 |
| 0 | 0 | 0000 0000 | 000 0000 0000 0000 0000 |
| +Infinity | 0 | 1111 1111 | 000 0000 0000 0000 0000 |
| −Infinity | 1 | 1111 1111 | 000 0000 0000 0000 0000 |
| Quiet NaN | x | 1111 1111 | 0xx xxxx xxxx xxxx xxxx |
| Signaling NaN | x | 1111 1111 | 1xx xxxx xxxx xxxx xxxx |

除以0/负数开根号

```
.data
	sdata: .word 0xff7f7fff
	fneg1: .float -1
.text
	lw $t0,sdata
	mtc1 $t0,$f1
	mul.s $f12,$f1,$f1

	li $v0,2
	syscall

	lwc1 $f2,fneg1
	mul.s $f12,$f12,$f2

	li $v0,2
	syscall

	li $v0,10
	syscall
```

$sdata^2$

正无穷

负无穷

```
.data
	sdata: .word 0xffff7fff
	fneg1: .float -1
.text
	lw $t0,sdata
	mtc1 $t0,$f1
	mul.s $f12,$f1,$f1

	li $v0,2
	syscall

	lwc1 $f2,fneg1
	div.s $f12,$f12,$f2

	li $v0,2
	syscall

	li $v0,10
	syscall
```

NaN

# Demo 1

```
.include "macro_print_str.asm"
.data
        f1: .float 12.625    12.525
.text
        lwc1 $f0,f1
        floor.w.s $f1,$f0
        ceil.w.s $f2,$f0
        round.w.s $f3,$f0

        print_string("orignal float: ")
        print_float($f0)

        print_string("\nafter floor:")
        print_float($f1)

        print_string("\nafter ceil:")
        print_float($f2)

        print_string("\nafter round:")
        print_float($f3)

        end
```

What's the output of current demo after running?
Why?
How to change the code to get correct output?

```
.macro print_float(%fr)
        addi $sp,$sp,-8
        swc1 $f12,4($sp)
        sw $v0,0($sp)

        mov.s $f12,%fr
        li $v0,2
        syscall

        lw $v0,0($sp)
        lwc1 $f12,4($sp)
        addi $sp,$sp,8
.end_macro
```

```
orignal float: 12.625
after floor:1.7E-44
after ceil:1.8E-44
after round:1.8E-44
-- program is finished running --
```

```
orignal float: 12.625      12.500
after floor:12             12
after ceil:13              13
after round:13             12
-- program is finished running --
```

> 0.5 向上取整

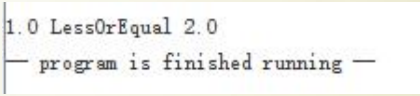
≤ 0.5 舍去

# Demo2

```
##piece 1/2 of code##
.include "macro_print_str.asm"
.data
	str1: .asciiz   "str1:"
	fd1:  .float     1.0
	dd1: .double 2.0
.text

	li $v0 2
	syscall




	bc1t printLe
	j printGt
```

```
##piece 2/2 of code##
printLe:
	print_string(  “is less or equal than ")
	j printSecondData
printGt:
	print_string(" is larger than")
printSecondData:
	li $v0,3
	syscall

	end
```

The output is expected to be the following screenshot, please complete the code.



# Practices

$4-\frac{4}{3}+\frac{4}{5}$

- 1. Calculate the value of π from the infinite series: $\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$

  Input a double-precision float number which represents a precision threshold. 界限

  Your program should terminate when the difference between two successive 起点, iterations is smaller than the precision threshold.

  Print the value of π (as double-precision float).

  f0 — precision

  f2 — 每次值　f4 分母

- 2. Given a single-precision float number 'x' and a positive integer 'r'. Round up 'x' to a number which keeps 'r' digits after the decimal point. Print the processing results and the final results.

  f8 分子=-4.0

  f6 — 上一次

  For example, suppose 'x' is 1.5671

  - if 'r' is 2, print 1.57;
  - if 'r' is 0, print 2;
  - if 'r' is 3, print 1.567;

  f10 : $(-1)^n$

  f16 : 2

  f12 — result

  f18 : (−1)

  1 $1.5671 \times 10^3 = 1567.1$

  2 $1567.1 \xrightarrow{取整} 1567$

  3 $1567 / 10 = 156$

④ $1567 - 156 \times 10 = 7$ ⑤ $1 < 5$ $156/100 = 1.56$.

$f_1 \sim f_{12}$

$v_0 \to r$

$f_0 \to x$

$S_0$ 计数

$f_1: 10^3$ $f_2: 10^2$ $f_3: 10$ $f_4: 10^2$

①② $f_2 = f_0 \times f_1 \xrightarrow{\text{取整}} f_2^{15} \to t_0\ (1567)$ 15

$1.57 \times 10 = 15.7$

③ $t_0 / 10 = t_1\ (156)$ 1

$15 - 10 = 5$

④ $t_2 = t_1 \times 10$ $t_0 = t_0 - t_2\ (7)$

⑤ $t_0 \geq 5$ $t_1$++ move $t_0$

$t_0 < 5$ $t_1$ 2 $\Rightarrow f_{12}$ 2 $\Rightarrow f_{12} = f_{12} / f_2$. 转换成float

$2/1 = 2$,

# Tips:

Single

| | 31 | 30 23 | 22 0 |
|---|---|---|---|
| | Sign | Exponent | Mantissa |
| 93000000 | 0 | 0001 1010 | 101 1000 1011 0001 0001 |
| 0 | 0 | 0000 0000 | 000 0000 0000 0000 0000 |
| +Infinity | 0 | 1111 1111 | 000 0000 0000 0000 0000 |
| −Infinity | 1 | 1111 1111 | 000 0000 0000 0000 0000 |
| Quiet NaN | x | 1111 1111 | 0xx xxxx xxxx xxxx xxxx |
| Signaling NaN | x | 1111 1111 | 1xx xxxx xxxx xxxx xxxx |

High-order word Low-order word

Double

| | 31 | 30 20 | 19 0 31 0 |
|---|---|---|---|
| | Sign | Exponent | Mantissa |
| 93000000 | 0 | 000 0001 1010 | 1011 0001 0110 0010 0010 1000 0000 .... |
| 0 | 0 | 000 0000 0000 | 0000 0000 0000 0000 0000 0000 .... |
| +Infinity | 0 | 111 1111 1111 | 0000 0000 0000 0000 0000 0000 .... |
| −Infinity | 1 | 111 1111 1111 | 0000 0000 0000 0000 0000 0000 .... |
| Quiet NaN | x | 111 1111 1111 | 0xxx xxxx xxxx xxxx xxxx xxxx .... |
| Signaling NaN | x | 111 1111 1111 | 1xxx xxxx xxxx xxxx xxxx .... |

reference from “see in MIPS”

Registers | Coproc 1 | Coproc 0

| Name | Float | Double |
|---|---|---|
| $f0 | 0x00000000 | 0x0000000000000000 |
| $f1 | 0x00000000 | |
| $f2 | 0x00000000 | 0x0000000000000000 |
| $f3 | 0x00000000 | |
| $f4 | 0x00000000 | 0x0000000000000000 |
| $f5 | 0x00000000 | |
| $f6 | 0x00000000 | 0x0000000000000000 |
| $f7 | 0x00000000 | |
| $f8 | 0x00000000 | 0x0000000000000000 |
| $f9 | 0x00000000 | |
| $f10 | 0x00000000 | 0x0000000000000000 |
| $f11 | 0x00000000 | |
| $f12 | 0x00000000 | 0x4000000000000000 |
| $f13 | 0x40000000 | |
| $f14 | 0x00000000 | 0x3ff0000000000000 |
| $f15 | 0x3ff00000 | |
| $f16 | 0x00000000 | 0x0000000000000000 |
| $f17 | 0x00000000 | |
| $f18 | 0x00000000 | 0x0000000000000000 |
| $f19 | 0x00000000 | |
| $f20 | 0x00000000 | 0x0000000000000000 |
| $f21 | 0x00000000 | |
| $f22 | 0x00000000 | 0x0000000000000000 |
| $f23 | 0x00000000 | |
| $f24 | 0x00000000 | 0x0000000000000000 |
| $f25 | 0x00000000 | |
| $f26 | 0x00000000 | 0x0000000000000000 |
| $f27 | 0x00000000 | |
| $f28 | 0x00000000 | 0x0000000000000000 |
| $f29 | 0x00000000 | |
| $f30 | 0x00000000 | 0x0000000000000000 |
| $f31 | 0x00000000 | |

Condition Flags

- [x] 0
- [ ] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5
- [ ] 6
- [ ] 7

registers and flags in coprocessor 1

2. Given a single-precision float number 'x' and a positive integer 'r'. Round up 'x' to a number which keeps 'r' digits after the decimal point. Print the processing results and the final results.

For example, suppose 'x' is 1.5671

- if 'r' is 2, print 1.57;
- if 'r' is 0, print 2;
- if 'r' is 3, print 1.567;