DHA Suffa University
Department of Computer Science
Computer Organization & Assembly Language
Spring 2021
Lab # 11 (Floating Point Operations)

Objective:

To deal with floating point operations in MIPS.

Floating-Point Operations:

The floating-point unit has 32 floating-point registers. These registers are numbered like the CPU registers. In the floating-point instructions we refer to these registers as $f0, $f1, and so on. Each of these registers is 32 bits wide. Thus, each register can hold one single-precision floating-point number. How can we use these registers to store double precision floating-point numbers? Because these numbers require 64 bits, register pairs are used to store them. This strategy is implemented by storing double-precision numbers in even-numbered registers. For example, **when we store a double-precision number in $f2, it is actually stored in registers $f2 and $f3.**

Floating-Point Register usage convention:

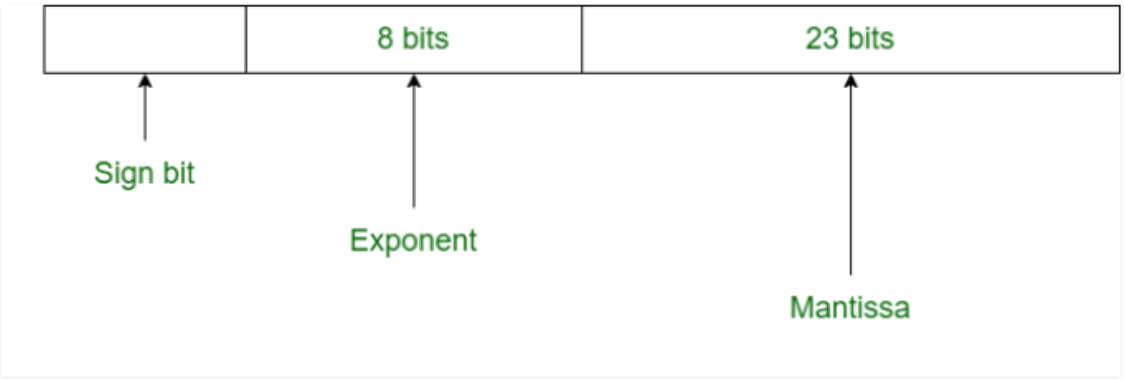
`$f0 – $f3`	Used to return values from a procedure
`$f4 – $f11`	Used as temporaries (caller saved) The called procedure can overwrite them.
`$f12 – $f15`	Used for parameter passing These registers are not preserved across procedure calls.
`$f16 – $f19`	Used as temporaries (caller-saved) The called procedure can overwrite them.
`$f20 – $f31`	Used as temporaries (callee saved) The called procedure cannot overwrite them.

Data Allocation Directives:

Directive	Type
.word	32-bit integer
.half	16-bit integer
.byte	8-bit integer
.float	32-bit IEEE floating point
.double	64-bit IEEE floating point
.space	Uninitialized memory block
.ascii	ASCII string
.asciiz	Null-terminated ASCII string

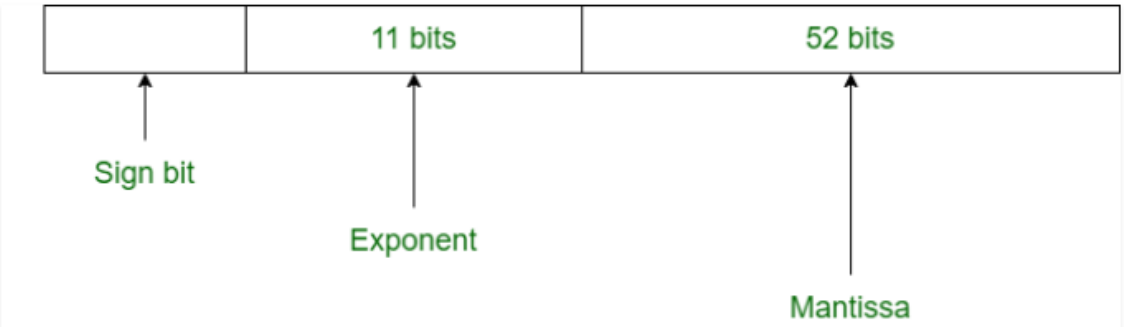
1. Single Precision:

Single Precision is a format proposed by IEEE for representation of floating-point number. It occupies 32 bits in computer memory.



2. Double Precision:

Double Precision is also a format given by IEEE for representation of floating-point number. It occupies 64 bits in computer memory.



Move Instructions for Floating-Point Operations:

mov.s FRdest,FRsrc

This instruction copies a single-precision floating-point number from the FRsrc to the Frdest register. If we want to copy a double-precision number, use mov.d instead. For e.g.

mov.d $f12,$f0

Above Instruction copies the double floating-point from $f0 to $f12 register.

Load and Store Instructions for Floating-Point Operations:

Following Instructions load the single precision floating point variable into FP register and double precision floating point variable into FP register respectively.

l.s FP_Register_Destination, address

l.d FP_Register_Destination, address

Similarly for Store instruction s.s and s.d are use for single and double precision floating point numbers respectively, as shown below:

s.s FP_Register_Source, Destination_address

s.d FP_Register_Source, Destination_address

Following Instructions load the single precision floating point constant value into FP register double precision floating point constant value into FP register respectively.

li.s $f0, 0.0

li.d $f2, 0.0

Arithmetic Instructions for Floating-Point Operations:

For Subtraction:

sub.s FRdest, FRsrc1,FRsrc2

sub.d FRdest,FRsrc1,FRsrc2

For Addition:

add.s FRdest,FRsrc1,FRsrc2

add.d FRdest,FRsrc1,FRsrc2

For Multiplication:

mul.s FRdest,FRsrc1,FRsrc2

mul.d FRdest,FRsrc1,FRsrc2

For Division:

div.s FRdest,FRsrc1,FRsrc2

div.d FRdest,FRsrc1,FRsrc2

Example 1

```
.data
string1:.asciiz  "\nAdd = "
string2:.asciiz  "\nSub = "
string3:.asciiz  "\nMul = "
string4:.asciiz  "\nDiv = "

number1: .float 2.5
number2: .float 3.3
.text

l.s $f1 number1
l.s $f2 number2

la $a0, string1
li $v0, 4
syscall
add.s $f3, $f1, $f2
mov.s $f12, $f3
li $v0, 2
syscall
```

```
la $a0, string2
li $v0, 4
syscall
sub.s $f3, $f1, $f2
mov.s $f12, $f3
li $v0, 2
syscall

la $a0, string3
li $v0, 4
syscall
mul.s $f3, $f1, $f2
mov.s $f12, $f3
li $v0, 2
syscall

la $a0, string4
li $v0, 4
syscall
div.s $f3, $f1, $f2
mov.s $f12, $f3
li $v0, 2
syscall


li $v0, 10
syscall
```

Example 2

```
.data
string1:.asciiz  "\nAdd = "
string2:.asciiz  "\nSub = "
string3:.asciiz  "\nMul = "
string4:.asciiz  "\nDiv = "

number1: .double 2.5
number2: .double 3.3
.text

l.d $f0 number1
l.d $f2 number2

la $a0, string1
li $v0, 4
syscall
add.d $f4, $f0, $f2
mov.d $f12, $f4
li $v0, 3
syscall

la $a0, string2
li $v0, 4
syscall
sub.d $f4, $f0, $f2
mov.d $f12, $f4
li $v0, 3
```

```
syscall

la $a0, string3
li $v0, 4
syscall
mul.d $f4, $f0, $f2
mov.d $f12, $f4
li $v0, 3
syscall

la $a0, string4
li $v0, 4
syscall
div.d $f4, $f0, $f2
mov.d $f12, $f4
li $v0, 3
syscall


li $v0, 10
syscall
```

Conditional Jumps for Floating-Point Operations:

Conditional jumps are performed in two stages

i). Comparison of FP values sets a code in a special register

ii). Branch instructions jump depending on the value of the code

Comparison:

- c.eq.s $f2, $f4 if $f2 == $f4 then code = 1 else code = 0
- c.le.s $f2, $f4 if $f2 <= $f4 then code = 1 else code = 0
- c.lt.s $f2, $f4 if $f2 < $f4 then code = 1 else code = 0

Branches:

- bc1f label if code == 0 then jump to label
- bc1t label if code == 1 then jump to label

Example 3

```
.data
string1:.asciiz  "\nNumbers are Equal\n"
string2:.asciiz  "\nNumbers are not Equal\n"

number1: .float 2.5
number2: .float 2.4
.text

l.s $f0 number1
l.s $f2 number2

c.eq.s $f0, $f2

bc1t true
```

```
la $a0, string2
li $v0, 4
syscall
b exit

true:
la $a0, string1
li $v0, 4
syscall

exit:
li $v0, 10
syscall
```

Converting floating point number to integer:

Let $f6 contains any floating point number:

```
cvt.w.s $f6, $f6        #f6 = (int) $f6
mfc1 $t0, $f6           #$t0 = $f6
```

Now $t0 contains the integer value.

Converting double precision floating point number to integer:

Let $f6 contains any floating point number:

```
cvt.w.d $f6, $f6        #f6 = (int) $f6
mfc1.d $t0, $f6         #$t0 = $f6
```

Now $t0 contains the integer value.

Converting integer to floating point number:

Let $t0 contains any integer number:

```
mtc1 $t0, $f2          #f2=$t0

cvt.s.w $f2, $f2       #$f2 = (float) $f2
```

Now $f2 contains the floating point number.

Converting integer to double precision floating point number:

Let $t0 contains any integer number:

```
mtc1.d $t0, $f2        #f2=$t0

cvt.d.w $f2, $f2       #$f2 = (double) $f2
```

Now $f2 contains the double precision floating point number.

MIPS Program for calculating the average of array:

```
.data
Array:   .float      24.0,87.0,34.0,23.0,42.0,67.0,76.0,12.0,92.0,85.0
N:       .float      10.0

.text
.globl main

main:
        la      $t0 , Array
        l.s     $f4 , N
        l.s     $f5 , 0($t0)

        li $t5, 1

        mtc1 $t5, $f6
        cvt.s.w $f6, $f6
        mtc1 $t5, $f9
        cvt.s.w $f9, $f9

loop :
        add     $t0     , $t0     , 4
        l.s     $f7     , 0($t0)
        add.s   $f5     , $f5     , $f7
        add.s   $f6,    , $f6     , $f9
        c.lt.s  $f6,    , $f4
        bc1t    loop
        div.s   $f8     , $f5     , $f4
```

```
li      $v0 , 2
mov.s   $f12 , $f8
syscall
li      $v0 , 10
syscall
```

LAB TASK 11

(1) Write a program in MIPS to reverse the array without using another array. Array should be of type double and it should contain only 10 values.

(2) Write a program that inputs an integer value and prints the number with its digits reversed. For example, given the number 1234, the program should print 4321.