COMP1047 Lab Week 04

1. Convert the Hexadecimal number 3A.F0 into single precision floating point representation. Show its corresponding 32-bit word in Hex format.

   Solution
   3A.F0 = (0011 1010. 1111 0000)2 = (-1)0 * 1.110101111 * 2^5
   Sign: 0
   Fraction: 110101111
   Biased Exponent = Actual Exponent + Bias = 5+127 = 132 = (1000 0100)2
   Single-Float Representation = (0 1000 0100 1101 0111 1000 0000 0000 000)2
   = (426B C000)h

2. Work out the IEEE 754 single-precision representations for the following numbers.
   (a) 10.25 (b) -128.6

   Solution
   (a)
   Step 1: Convert decimal 10.25 into binary correspondence
   10 (1010)2
   0.25 (0.01)2
   10.25 (1010.01)2
   Step 2: Normalized scientific notation for binary correspondence
   10.25 (-1)0 * 1.01001 * 2^3
   Step 3: IEEE 754 Single-Floating Point Standard
   Sign: 0
   Fraction: 01001
   Biased Exponent = Actual Exponent + Bias
   = 3 + 127 = 130 (1000 0010)2
   The IEEE 754 single-floating point representation for 10.25 is
   0 1000 0010 0100 1000 0000 0000 0000 000
   Step 4: IEEE 754 Double-Floating Point Standard
   Sign: 0
   Fraction: 01001
   Biased Exponent = Actual Exponent + Bias
   = 3 + 1023 = 1026 (1000 0000 010)2
   The IEEE 754 double-floating point representation for 10.25 is
   0 1000 0000 010 0100 1000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

   (b)
   Step 1: Convert decimal -128.6 into binary correspondence
   128 (1000 0000)2
   0.6 (0.1001 1001 1001 ...)2
   (Notes: Digits in fraction part repeats forever and 0.6 cannot be expressed exactly by finite digits.)
   -128.6 (1000 0000.1001 1001 1001 ...)2

Step 2: Normalized scientific notation for binary correspondence

-128.6 (-1)1 *(1.000 0000 1001 1001 1001 ...) * 2^7

Step 3: IEEE 754 Single-Floating Point Standard

Sign: 1

Fraction: 000 0000 1001 1001 1001 ...

Biased Exponent = Actual Exponent + Bias

= 7 + 127 = 134 (1000 0110)2

The IEEE 754 single-floating point representation for -128.6 is

1 1000 0110 0000 0001 0011 0011 0011 001

Step 4: IEEE 754 Double-Floating Point Standard

Sign: 1

Fraction: 0000 0001 0011 0011 0011 ...

Biased Exponent = Actual Exponent + Bias

= 7 + 1023 = 1030 (100 0000 0110)2

The IEEE 754 single-floating point representation for -128.6 is

1 100 0000 0110 0000 0001 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011 0011

3. Write a MIPS program, using .word or .float directive to store both numbers into the memory and then print them out to check whether your results in the previous question are correct. *Hint: You may need to use instruction lwc1.*

Solution

```
.data
#single-floating point representations for 10.25 and -128.6 respectively
SingleNum: .word 0x41240000 0xC3009999
#double-floating point representations for 10.25 and -128.6 respectively
DoubleNum:.word 0x00000000 0x40248000 0x33333333 0xC0601333
.text
.globl main

main:
# print out single-floating point number 10.25
la $t0, SingleNum #load the base address
lwc1 $f12, 0($t0) #load the single-floating point representation for 10.25 into $f12
li $v0, 2 # print single-floating point 10.25
syscall

# print out single-floating point number -128.6
lwc1 $f12, 4($t0) #load the single-floating point representation for -128.6 into $f12
li $v0, 2 # print single-floating point -128.6
syscall

# print out double-floating point number 10.25
la $a0, DoubleNum # load the base address
lwc1 $f12, 0($a0)
lwc1 $f13, 4($a0) # load the double-floating point representation for 10.25 into
$f12&$f13
li $v0, 3 # print double-floating point 10.25
```

```
syscall

# print out double-floating point number -128.6
lwc1 $f12, 8($a0)
lwc1 $f13, 12($a0) # load the double-floating point representation for -128.6 into
$f12&$f13
li $v0, 3 # print double-floating point -128.6
syscall

li $v0, 10 # exit
syscall
```

4. Write a short MIPS program to complete the addition and multiplication of the two numbers (10.25 and -128.6), and then print the results out.

Solution

```
.data
SingleNum: .float 10.25 -128.6
DoubleNum: .double 10.25 -128.6
.text
.globl main

main:
la $a0, SingleNum # load the base address
lwc1 $f2, 0($a0) # load the single-floating point 10.25 into $f2
lwc1 $f3, 4($a0) # load the single-floating point -128.6 into $f3
#Print out the sum of single-floating point numbers 10.25 and -128.6
add.s $f12, $f3, $f2 # f12 := f2 + f3
li $v0, 2 # print single-floating point addition (-128.6 + 10.25)
syscall

#Print out the product of single-floating point numbers 10.25 and -128.6
mul.s $f12, $f3, $f2 # f12 := f2 * f3
li $v0, 2 # print single-floating point multiplication (-128.6 * 10.25)
syscall

la $a0, DoubleNum # load the base address
l.d $f2, 0($a0) # load the double-floating point 10.25 into $f2, $f3
l.d $f4, 8($a0) # load the double-floating point -128.6 into $f4, $f5
#Print out the sum of double-floating point numbers 10.25 and -128.6
add.d $f12, $f4, $f2 # f12 := f2 + f4
li $v0, 3 # print double-floating point addition (-128.6 + 10.25)
syscall

#Print out the product of double-floating point numbers 10.25 and -128.6
mul.d $f12, $f4, $f2 # f12 := f2 * f4
li $v0, 3 # print double-floating point multiplication (-128.6 * 10.25)
syscall

li $v0, 10 # exit
syscall
```

5. Write a program in MIPS32 assembly language which reads two numbers x and y from the console, calculates, then prints x + 2y + 32769. *Hint: no multiplication is necessary and proper user prompts are expected.*

Solution

```
.data
prompt1: .asciiz "Please input x: "
prompt2: .asciiz "Please input y: "
rs_string: .asciiz "The result of (x - 2y + 32769) is: "
.text
.globl main

main:
# prompt for input
la $a0, prompt1 # prompt x
li $v0, 4
syscall

li $v0, 5 # read input x
syscall

or $s0, $zero, $v0 # Save x to s0
la $a0, prompt2 # prompt y
li $v0, 4
syscall

li $v0, 5 # read input y
syscall

or $s1, $zero, $v0 # Save y to s1
la $a0, rs_string # The result is
li $v0, 4
syscall

# calculation
sll $s1, $s1, 1 # 2y
sub $s0, $s0, $s1 # x - 2y
addi $a0, $s0, 32769 # a0 = x - 2y + 32769
li $v0, 1 # output result
syscall

# exit
```

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

There are several ways to deal with the problem. One possibility is to use oat type:

```
.data
prompt1: .asciiz "Please input x: "
prompt2: .asciiz "Please input y: "
rs_string: .asciiz "The result of (x - 2y + 32769) is: "
.text
.globl main

main:
# prompt for input
la $a0, prompt1 # prompt x
li $v0, 4
syscall

li $v0, 6 # read input x
syscall

mov.s $f2, $f0 # save to f2
la $a0, prompt2 # prompt y
li $v0, 4
syscall

li $v0, 6 # read input y
syscall

mov.s $f4, $f0 # save to f4
add.s $f4, $f4, $f4 # 2y
sub.s $f2, $f2, $f4 # x-2y
li.s $f6, 32769.0
add.s $f2, $f2, $f6
mov.s $f12, $f2
la $a0, rs_string # The result is
li $v0, 4
syscall

li $v0, 2 # output result
syscall

# exit
li $v0, 10
syscall
```

6. Write a program in MIPS32 assembly language which reads three integer numbers x, y and z from the console, then calculates and prints out *m*, the minimum of the three. The following C segment shows how *m* can be calculated:

```
m = x;
if (m > y) m = y;
if (m > z) m = z;
```

Solution

```
.data
prompt1:.asciiz "Please input x: "
prompt2:.asciiz "Please input y: "
prompt3:.asciiz "Please input z: "
rs_string:.asciiz "The minimum number is: "
.text
.globl main

main:
  # prompt for input
  la $a0, prompt1 # prompt x
  li $v0, 4
  syscall

  li $v0, 5     # read input x
  syscall

  or $s0, $zero, $v0  # Save x to s0
  la $a0, prompt2 # prompt y
  li $v0, 4
  syscall

  li $v0, 5     # read input y
  syscall

  or $s1, $zero, $v0     # Save y to s1
  la $a0, prompt3 # prompt z
  li $v0, 4
  syscall

  li $v0, 5     # read input z
  syscall

  or $s2, $zero, $v0     # Save y to s1
  # Print rs_string first
  la $a0, rs_string
  li $v0, 4
  syscall
```

```
    # calculation
    move $a0, $s0    # m = x
    slt $t0, $s1, $a0  # t0 = 1 if s1 < s0
    beq $t0, $zero, compare2 # jump to compare2 if s1 >= s0

    move $a0, $s1       #if y<m, m=y

# compare smaller one in (s0, s1) with s2
compare2:
    slt $t0, $s2, $a0 #s2 < a0
    beq $t0, $zero, print_res # jump to print_res if s2 >= s0
    or $a0, $s2, $0        # if z<m, m = z

print_res:
    li $v0, 1       #output result
    syscall
    li $v0, 10   # exit
    syscall
```

7. Write a program in MIPS assembly language to read two integer numbers A and B. The program should indicate if one of these numbers is multiple of the other one.

Solution

```
.data
prompt1: .asciiz "Please input A: "
prompt2: .asciiz "Please input B: "
AofB_string: .asciiz "A is the multiple of B.\n"
BofA_string: .asciiz "B is the multiple of A.\n"
no_string: .asciiz "They are not the multiple of each other.\n"
.text
.globl main

main:
## prompt for input
la $a0, prompt1 # prompt A
li $v0, 4
syscall
li $v0, 5 # read input A
syscall
or $s0, $zero, $v0 # save A to s0

la $a0, prompt2 # prompt B
li $v0, 4
syscall
li $v0, 5 # read input B
syscall
or $s1, $zero, $v0 # save B to s1

## calculation
```

```
div $s0, $s1 # Lo = $s0 / $s1, Hi = $s0 mod $s1
mfhi $t0 # move quantity in special register Hi to $t0: $t0 = Hi, i.e. the remainder
beq $t0, $zero, AofB # if the remainder is 0, jump to branch AofB

div $s1, $s0 # Lo = $s1 / $s0, Hi = $s1 mod $s0
mfhi $t0 # move quantity in special register Hi to $t0: $t0 = Hi, i.e. the remainder
beq $t0, $zero, BofA # if the remainder is 0, jump to branch BofA

no: # otherwise, move to branch no
la $a0, no_string # "They are not the multiple of each other."
li $v0, 4
syscall
j exit # exit the program

AofB:
la $a0, AofB_string # "A is the multiple of B."
li $v0, 4
syscall
j exit

BofA:
la $a0, BofA_string # "B is the multiple of A."
li $v0, 4
syscall

exit: # exit the program
li $v0, 10
syscall
```