POST LAB 6(A and B)

NAME - RAJENDRA SINGH

ROLL NO.- 111601017

QUESTION (1) Data transfer operations:

(a) Load all the temporary registers t0 to t9 with 4-bit hex value 0xF and saved registers s0 to s7 with the 4 bit hex value 0xA. Move data in t6 to s4 and s7 to t5

(b) Load all the temporary registers t0 to t9 with 16-bit hex value 0xFFFF and saved registers s0 to s7 with the 16 bit hex value 0xAAAA.

(c) Load all the temporary registers t0 to t9 with 32-bit hex value 0xFFFFFFFF and saved registers s0 to s7 with the 32 bit hex value 0xAAAAAAAA.

(d) Load all the temporary registers t0 to t9 and saved registers s0 to s7 with the 36 bit hex value 0x123456789.

Explain your observations in each of the above 4 cases.

```
# problem 1a



# Load all the temporary registers t0 to t9 with 4-bit hex value 0xF and saved
# registers s0 to s7 with the 4 bit hex value 0xA. Move data in t6 to s4 and s7 to t5

.text

.globl main

main:
        # load immediate instructions
        li $t0 , 0xF
        li $t1 , 0xF
        li $t2 , 0xF
        li $t3 , 0xF
        li $t4 , 0xF
        li $t5 , 0xF
        li $t6 , 0xF
        li $t7 , 0xF
        li $t8 , 0xF
        li $t9 , 0xF
li $s0 , 0xA
        li $s1 , 0xA
        li $s2 , 0xA
        li $s3 , 0xA
        li $s4 , 0xA
        li $s5 , 0xA
        li $s6 , 0xA
        li $s7 , 0xA


        # move the data from one register to another
        move $s4 , $t6
        move $t5 , $s7

        # exit instruction
        li $v0 , 10
        syscall
```

#PROBLEM 1(b) Load all the temporary registers t0 to t9 with 16-bit hex value 0xFFFF and saved
# registers s0 to s7 with the 16 bit hex value 0xAAAA.

```
.text

.globl main

main:
        # load immediate instructions
        li $t0 , 0xFFFF
        li $t1 , 0xFFFF
        li $t2 , 0xFFFF
        li $t3 , 0xFFFF
        li $t4 , 0xFFFF
        li $t5 , 0xFFFF
        li $t6 , 0xFFFF
        li $t7 , 0xFFFF
        li $t8 , 0xFFFF
        li $t9 , 0xFFFF
li $s0 , 0xAAAA
        li $s1 , 0xAAAA
        li $s2 , 0xAAAA
        li $s3 , 0xAAAA
        li $s4 , 0xAAAA
        li $s5 , 0xAAAA
        li $s6 , 0xAAAA
        li $s7 , 0xAAAA


        # move the data from one register to another
        #move $s4 , $t6
        #move $t5 , $s7

        # exit instruction
        li $v0 , 10
        syscall
```

```
# (c) Load all the temporary registers t0 to t9 with 32-bit hex value 0xFFFFFFFF and
# saved registers s0 to s7 with the 32 bit hex value 0xAAAAAAAA.

.text

.globl main

main:
        # load immediate instructions
        li $t0 , 0xFFFFFFFF
        li $t1 , 0xFFFFFFFF
        li $t2 , 0xFFFFFFFF
        li $t3 , 0xFFFFFFFF
        li $t4 , 0xFFFFFFFF
        li $t5 , 0xFFFFFFFF
        li $t6 , 0xFFFFFFFF
        li $t7 , 0xFFFFFFFF
        li $t8 , 0xFFFFFFFF
        li $t9 , 0xFFFFFFFF
li $s0 , 0xAAAAAAAA
        li $s1 , 0xAAAAAAAA
        li $s2 , 0xAAAAAAAA
        li $s3 , 0xAAAAAAAA
        li $s4 , 0xAAAAAAAA
        li $s5 , 0xAAAAAAAA
        li $s6 , 0xAAAAAAAA
        li $s7 , 0xAAAAAAAA


        # move the data from one register to another
        #move $s4 , $t6
        #move $t5 , $s7

        # exit instruction
        li $v0 , 10
        syscall
```

```
#1(d) Load all the temporary registers t0 to t9 and saved registers s0 to s7 with the 36
#bit hex value 0x123456789.




.text

.globl main

main:
        # load immediate instructions
        li $t0 , 0x123456789
        li $t1 , 0x123456789
        li $t2 , 0x123456789
        li $t3 , 0x123456789
        li $t4 , 0x123456789
        li $t5 , 0x123456789
        li $t6 , 0x123456789
        li $t7 , 0x123456789
        li $t8 , 0x123456789
        li $t9 , 0x123456789
li $s0 , 0x123456789
li $s1 , 0x123456789
li $s2 , 0x123456789
li $s3 , 0x123456789
li $s4 , 0x123456789
li $s5 , 0x123456789
li $s6 , 0x123456789
li $s7 , 0x123456789


# move the data from one register to another
#move $s4 , $t6
#move $t5 , $s7

# exit instruction
li $v0 , 10
syscall
```

(2) Storage of data in the data segment using ascii/asciiz:

Store 4 strings as indicated below. Load the address of each string into registers t0, t1, t2, t3 respectively. Run step by step and note how each string is getting stored and explain the changes occuring in the data segment:

```
str1: .ascii "123456789abcedef"
str2: .ascii "123456789ABCDEF"
str3: .asciiz "123456789abcedef"
str4: .asciiz "123456789ABCDEF"
```

```
# problem 2
# Store 4 strings as indicated below. Load the address of each string into registers t0, t1, t2,
# t3 respectively. Run step by step and note how each string is getting stored and explain
# the changes occuring in the data segment:

.data
	str1: .ascii "123456789abcedef"
	str2: .ascii "123456789ABCDEF"
	str3: .asciiz "123456789abcedef"
	str4: .asciiz "123456789ABCDEF"

.text
.globl main
main:
	# loading the address of each string
	la $t0 , str1
	la $t1 , str2
	la $t2 , str3
	la $t3 , str4

	# exit instruction
	li $v0 , 10
	syscall
```

QUESTION (3) Load and store a word, a half word and a byte:
Store the strings as indicated below:

```
str1: .ascii "123456789abcedef"
str2: .ascii "123456789ABCDEF"
```

(a) Load the first 3 bytes of str1 into t0, t1, t2 and store the values in registers t3, t4, t5 into last 3 bytes of str2.

(b) Load the first 2 half words of str2 into t6, t7 and store the value in the register t8 into last halfword of str1.

(c) Load the last word of str2 into t9 and store the value in the register s1 into the last word of str1.

```
# problem 3a
# (a) Load the first 3 bytes of str1 into t0, t1, t2 and store last 3 bytes of str2 into t3, t4,t5.
# (b) Load the first 2 half words of str2 into t6, t7 and store the last halfword of str1 into t8.
# (c) Load the last word of str2 into t9 and store the last word of str1 in s1 register.

#word is the in the multiple of 4 so the half wprd is in 2
#for accessing 1st, 2nd, or 3rd word, give the starting address of required word
.data
	str1: .ascii "123456789abcedef"
	str2: .ascii "123456789ABCDEF"

.text
.globl main

main:
	# loading the address of each string
	la $s0 , str1
	la $s1 , str2

	lb $t0 , 0($s0)
	lb $t1 , 1($s0)
	lb $t2 , 2($s0)
lb $t3 , 12($s1)
	lb $t4 , 13($s1)
	lb $t5 , 14($s1)

	# exit instruction
	li $v0 , 10
	syscall
```

```
# problem 3b
# (a) Load the first 3 bytes of str1 into t0, t1, t2 and store last 3 bytes of str2 into t3, t4,t5.
# (b) Load the first 2 half words of str2 into t6, t7 and store the last halfword of str1 into t8.
# (c) Load the last word of str2 into t9 and store the last word of str1 in s1 register.

.data
        str1: .ascii "123456789abcedef"
        str2: .ascii "123456789ABCDEF"

.text
.globl main

main:
        # loading the address of each string
        la $s0 , str1
        la $s1 , str2

        lb $t0 , 0($s0)
        lb $t1 , 1($s0)
        lb $t2 , 2($s0)

        sb $t3 , 12($s1)
        sb $t4 , 13($s1)
        sb $t5 , 14($s1)
```

```
lhu $t6 , 0($s1)
lhu $t7 , 2($s1)

lhu $t8 , 14($s0)

lw $t9 , 12($s1)
lw $s1 , 12($s0)



# exit instruction
li $v0 , 10
syscall
```

QUESTION (4) Illustrate the use of syscall codes for the following operations :
(a) Exiting a program

(b) Printing an integer and a string:
Store your roll number as memory word. Then display it on the Console. The display should be like:
My roll number is : <Your roll number>

(c) Reading and printing an integer and string:
Read your name and roll number during run time and then display the same on the Console. There should be prompts for entering the name and roll number. ie. The display should be like:

Please enter your name: <Enter your name>
Please enter your roll no.: <Enter your roll number>
Your name is : <Your name should be displayed here>
Your roll number is : < Your roll number should be displayed here>
Indicate the changes that you see in in data segment.

(d) Reading and printing a floating point number
Read a floating point number during run time and then display the same on the Console. ie. The display should be like:
Please enter a floating point number : <Enter the number>
You have entered : <The number should be displayed here>

```
#QUESTION 4
#reading and printing command by $v0 with various number
.data
        roll_no : .word 111601008
        name         : .asciiz "\nMy name is : bithack: "
        float_num : .float 0.0

        read_integer_prompt :.asciiz "\nEnter an integer: "
        read_string_prompt :.asciiz "\nEnter a string: "
        print_string :.asciiz "\nYou entered the string: "
        print_integer :.asciiz "\nYou entered the number : "
        read_float : .asciiz "\nEnter a floating point number: "
        print_float : .asciiz "\nYou entered a floating point number : "

# start of code section

.text
.globl main
        main:

        # print integer
        li $v0 , 1
        lw $a0 , roll_no   # integer to print
        syscall
```

```
# print string
        li $v0 , 4
        la $a0 , name
        syscall


#########################################################################
        # input the integer
        # print string
        li $v0 , 4
        la $a0 , read_integer_prompt
        syscall

        # read int from console
        li $v0 , 5
        syscall
        sw $v0 , roll_no

        # print string
        li $v0 , 4
        la $a0 , print_integer
        syscall
```

```
# print integer
        li $v0 , 1
        lw $a0 , roll_no    # integer to print
        syscall

################################################################################

        # input the string
        # print string
        li $v0 , 4
        la $a0 , read_string_prompt
        syscall

        # read string from console
        li $v0 , 8
        la $a0 , name
        syscall

        # print string
        li $v0 , 4
        la $a0 , print_string
        syscall
```

```
# print string
        li $v0 , 4
        la $a0 , name
        syscall

###################################################################

        # input the string
        # print string
        li $v0 , 4
        la $a0 , read_float
        syscall

        # read float from console
        li $v0 , 6
        syscall
        s.s $f0 , float_num

        # print float prompt
        li $v0 , 4
        la $a0 , print_float
        syscall
```

```
# print float num
        li $v0 , 2
        l.s $f12 , float_num
        syscall

########################################################################
        # exit
        li $v0 ,  10
        syscall
```

(5) Arithmetic operations:

Display the sum, difference, product and quotient and remainder of two numbers in the console.

(a) When both the numbers are positive

(b) When both the numbers are negative.

(c) When one number is positive and one is negative.

(d) Suppose your inputs are two hex numbers 0x08 and 0x04. Find the product (8*4) and quotient (8/4) without using the ‘mul’ and ‘div’ instruction.

For (a), (b), (c), the input numbers should be allowed to be given during run time. Use display statements wherever needed.
Remember that the arithmetic operations inside a computer is in 2’s complement.

```
#QUESTION 5
#sum by add, sub by sub, mul by mult(storing in both mfhi, mflo), div by div(storing mfhi->reminder, mflo->
Quotient) instruction,


.data
        prompt1 : .asciiz "\nEnter an integer: "
        prompt2 : .asciiz "\nEnter another integer: "
        prompt_sum : .asciiz "\nSum : "
        prompt_sub : .asciiz "\nSub : "
        prompt_mul : .asciiz "\nMul : "
        prompt_div : .asciiz "\nDiv : "

        prompt_rem : .asciiz "\nRem : "

        num1 : .word 0
        num2 : .word 0

# start of code section

.text
.globl main
        main:

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

```
#####################################################################################
# part a : both integer positive

# input the integer
        # print string
        li $v0 , 4
        la $a0 , prompt1
        syscall

        # read int from console
        li $v0 , 5
        syscall
        sw $v0 , num1

        # input the integer
        li $v0 , 4
        la $a0 , prompt2
        syscall

        # read int from console
        li $v0 , 5
        syscall
        sw $v0 , num2
```

```
lw $s0 , num1
lw $s1 , num2

# sum
li $v0 , 4
la $a0 , prompt_sum
syscall
add $t3 , $s0 , $s1
li $v0 , 1
move $a0 , $t3
syscall

# sub
li $v0 , 4
la $a0 , prompt_sub
syscall
sub $t3 , $s0 , $s1
li $v0 , 1
move $a0 , $t3
syscall
```

```
# mul
        li $v0 , 4
        la $a0 , prompt_mul
        syscall
        mult $s0 , $s1
        mfhi $t0
        mflo $t1
        li $v0 , 1
        move $a0 , $t0
        syscall
        li $v0 , 1
        move $a0 , $t1
        syscall

        # div
        li $v0 , 4
        la $a0 , prompt_div
        syscall
        div $s0 , $s1
        mfhi $t0     # remainder
        mflo $t1     # quotient
        li $v0 , 1
        move $a0 , $t1
        syscall
```

```
# rem
        li $v0 , 4
        la $a0 , prompt_rem
        syscall
        li $v0 , 1
        move $a0 , $t0
        syscall


############################
############################
##########
        # exit
        li $v0 ,  10
        syscall
```

(6) Illustration of logical operations-1:

Common data Questions:
The RGBA(Red, Green, Blue, Alpha) codes of the background colour of an image are given as {0xee, 0xff, 0xdd, 0x0}. They are stored in register t0 as $t0 = 0x0eeffdd0).
Use minimal number of instructions for all the operations.

(a) Extract the ‘red’ and ‘blue’ colour codes to registers t1 and t2 respectively using register t0 alone (ie. $t1=0xee and $t2= 0xdd). You are not allowed to directly load 0xee and 0xdd to the registers $t1 and $t2.

(b) Modify your code to extract the ‘green’ colour code to register t2. You should use only registers t0 and t2 for this operation. All the operations should be performed through values loaded in registers only.

(c) The colour code of ‘red’ needs to be changed from 0xee to 0xbb that of ‘green’ from 0xff to 0xcc. Extract the modified set of of colour codes into register t3 by using the previous value already present in register t0 (ie. 0x0eeffdd0).

(d) Set the Alpha code to 0x1 and store the modified set of colour codes in register t4. Other colour codes remain same as in t0.

```
#QUESTION 6
#we are extract the required part by anding it with 1 and rest with 0 and then shifting it correct position
.globl main
main:
	li $t0, 0x0eeffdd0		#storing original value

	and $t1, $t0, 0x0ff00000		#extracting red
	srl $t1, $t1, 20
	and $t2, $t0, 0x00000ff0		#extracting blue
	srl $t2, $t2, 4
	and $t2, $t0, 0x000ff000		#extracting green
	srl $t2, $t2, 12

	and $t3, $t0, 0xfbbccfff	# updating red from ee
					# to bb and green from
					# ff to cc,
					# rest remains same
	or $t3, $t3, 0x01100000
	or $t4, $t0, 0x00000001			#update alpha to 1

	li $v0, 10
	syscall
```

QUESTION (7) Illustration of logical operations-2:

You are provided with a LED array of 32 LEDs. Create a two way running led using MIPS assembly. It means that the LED initially get turned on in the one way and then in reverse direction.

It should work as follows. The leds are given numbers 0 to 31. Initially LED 2 should be ON, then 18, then 0 and then in the reverse order. The process need not be repeated.

(Hint: Use a 32 bit register. Make use of rotate instructions instead of shift and loop instructions)

#QUESTION 7

```
.text
.globl main
	main:
	# initially the LED 2 is ON
	li $t0 , 4
	# then rotate (18-2) positions left
	rol $t0 , $t0 , 16   # LED 18 is activated now
	# then rotate (18-0) positions right
	ror $t0 , $t0 , 18   # LED 0  is activated now
	# then rotate (18-0) positions left
	rol $t0 , $t0 , 18   # LED 18 is activated now
	# then rotate (18-2) positions right
	ror $t0 , $t0 , 16   # LED 2 is activated now

	# exit
	li $v0 ,  10
	syscall
```