Name: Nguyễn Thanh Long

ID: ITITRG18012

**Lab 04**

**4.1.1)**

.data # the data segment

prompt: .asciiz "Input: "

output: .asciiz "Output: "

error: .asciiz "Unvalid input (0-15)\n"

annotation: .asciiz "0x"

.text # the code segment

.globl main

main:

li $t1, 0 # min is 0

li $t2, 15 # max is 15

li $t3, 10 # hex threshold

# print out the prompt

la $a0, prompt

li $v0, 4

syscall

# read in an integer

li $v0, 5

syscall

move $t0, $v0

# print out the result message

la $a0, output

li $v0, 4

syscall

# branch

blt $t0, $t1, printError # if$t0 < $t1 then printError

bgt $t0, $t2, printError # if$t0 > $21 then printError

blt $t0, $t3, printNum # if$t0 < $t1 then target

j printCharacter

jr $ra # return to caller (\_\_start)

# print 0x then number then stop

printNum:

# print the '0x'

la $a0, annotation

li $v0, 4

syscall

# print the number in t0

move $a0, $t0

li $v0, 1

syscall

jr $ra

# print out the leftover in t3

printCharacter:

# print the '0x'

la $a0, annotation

li $v0, 4

syscall

# add 55 to t0

addi $t0, $t0, 55

# print the number in t0 (but in character)

move $a0, $t0

li $v0, 11

syscall

jr $ra

# print out the "error" and stop

printError:

la $a0, error

li $v0, 4

syscall

jr $ra

**4.1.2)**

.data # the data segment

prompt: .asciiz "Input: "

output: .asciiz "Output: "

error: .asciiz "Unvalid input (0-15)\n"

annotation: .asciiz "0x"

.text # the code segment

.globl main

main:

# print out the prompt

la $a0, prompt

li $v0, 4

syscall

# read in an integer

li $v0, 5

syscall

move $t0, $v0

# print out the result message

la $a0, output

li $v0, 4

syscall

# branch

blt $t0, 0, printError # if$t0 < $t1 then printError

bgt $t0, 15, printError # if$t0 > $21 then printError

blt $t0, 10, printNum # if$t0 < $t1 then target

j printCharacter

jr $ra # return to caller (\_\_start)

# print 0x then number then stop

printNum:

# print the '0x'

la $a0, annotation

li $v0, 4

syscall

# print the number in t0

move $a0, $t0

li $v0, 1

syscall

jr $ra

# print out the leftover in t3

printCharacter:

# print the '0x'

la $a0, annotation

li $v0, 4

syscall

# add 55 to t0

addi $t0, $t0, 55

# print the number in t0 (but in character)

move $a0, $t0

li $v0, 11

syscall

jr $ra

# print out the "error" and stop

printError:

la $a0, error

li $v0, 4

syscall

jr $ra

****

****

****

**4.1.2)**

.data # the data segment

prompt: .asciiz "Input: "

output: .asciiz "Output: "

error: .asciiz "Unvalid input (0-15)\n"

annotation: .asciiz "0x"

.text # the code segment

.globl main

main:

# print out the prompt

la $a0, prompt

li $v0, 4

syscall

# read in an integer

li $v0, 5

syscall

move $t0, $v0

# print out the result message

la $a0, output

li $v0, 4

syscall

# make space in STACK

addi $sp, $sp, -4

# Save $ra into stack

sw $ra, 0($sp)

# Jal to PrintHex

jal printHex

# Load $ra out of stack

lw $ra, 0($sp)

# remove the stack

addi $sp, $sp, 4

jr $ra

# Procedure printHex(t0 input)

printHex:

# branch

blt $t0, 0, printError # if$t0 < $t1 then printError

bgt $t0, 15, printError # if$t0 > $21 then printError

blt $t0, 10, printNum # if$t0 < $t1 then target

j printCharacter

jr $ra

# print 0x

printAnnotation:

la $a0, annotation

li $v0, 4

syscall

jr $ra

# print 0x then number then stop

printNum:

# print the '0x'

jal printAnnotation

# print the number in t0

move $a0, $t0

li $v0, 1

syscall

j callTerminate

# print out the leftover in t3

printCharacter:

# print the '0x'

jal printAnnotation

# add 55 to t0

addi $t0, $t0, 55

# print the number in t0 (but in character)

move $a0, $t0

li $v0, 11

syscall

j callTerminate

# print out the "error" and stop

printError:

la $a0, error

li $v0, 4

syscall

j callTerminate

# call terminate

callTerminate:

li $v0, 10

syscall

**4.1.3)**

.data # the data segment

prompt1: .asciiz "Input: "

prompt2: .asciiz "Result: "

error: .asciiz "Invalid input!\n"

annotation: .asciiz "0x"

newline: .asciiz "\n"

.text # the code segment

.globl main

main:

# print out the prompt

la $a0, prompt1

li $v0, 4

syscall

# read in an integer

li $v0, 5

syscall

move $t0, $v0

# print out the result message

la $a0, prompt2

li $v0, 4

syscall

#passing parameter

move $a3, $t0

#call procedure

jal printHex32

mainLoop:

# print out the result message

la $a0, newline

li $v0, 4

syscall

j main

printHex32: #(int $a3)

# print the '0x'

la $a0, annotation

li $v0, 4

syscall

# for ( t1=0; t1<8; t1++)

#loop printHex32

li $t1, 0 # t1=0

ifPrintHex32:

blt $t1, 8, loopPrintHex32 # t1 < 8

# else restart a program

j mainLoop #return caller

loopPrintHex32:

# Ex: $a3 = 0x1234567AB

rol $a3, $a3, 4 # $a3 = 0x234567AB1

andi $t2, $a3, 0xF # get t2 = 0x1

# do printHex(0x1)

move $a1, $t2 # pass paramter for printHex

jal printHex

addi $t1, $t1, 1

j ifPrintHex32

printHex: # (int $a1)

# if (t0 < 0 || > 15) => ERROR

blt $a1, 0, print\_error

bgt $a1, 15, print\_error

blt $a1, 10, print\_int # VALID 0->15

#else

j print\_char

jr $ra # return to caller (\_\_start)

# print 0x then number then stop

print\_int:

# print the number in t0

move $a0, $a1

li $v0, 1

syscall

jr $ra # return to caller (\_\_start)

print\_char:

# add 55 to t0

addi $a1, $a1, 55

# print the number in t0 (but in character)

move $a0, $a1

li $v0, 11

syscall

jr $ra # return to caller (\_\_start)

# print out the "error" and stop

print\_error:

la $a0, error

li $v0, 4

syscall

jr $ra # return to caller (\_\_start)

****

4.2.1/

.data

prompt1: .asciiz "the final num is: "

.text

.globl main

main:

li $t1, 1

li $t2, 0

li $t3, 100

li $t5, 1

WHILE:

beq $t5, $t3, EXIT # branch to exit if $t5 = $t3

sub $t4, $t1, $t2 # $t4=$t1-$t2

move $t2, $t4 # copy $t4 to $t2

add $t1, $t1, $t2 # $t1=$t1+$t2

addi $t5, $t5, 1 # $t5=$t5+1

j WHILE

EXIT:

la $a0, prompt1

li $v0, 4

syscall

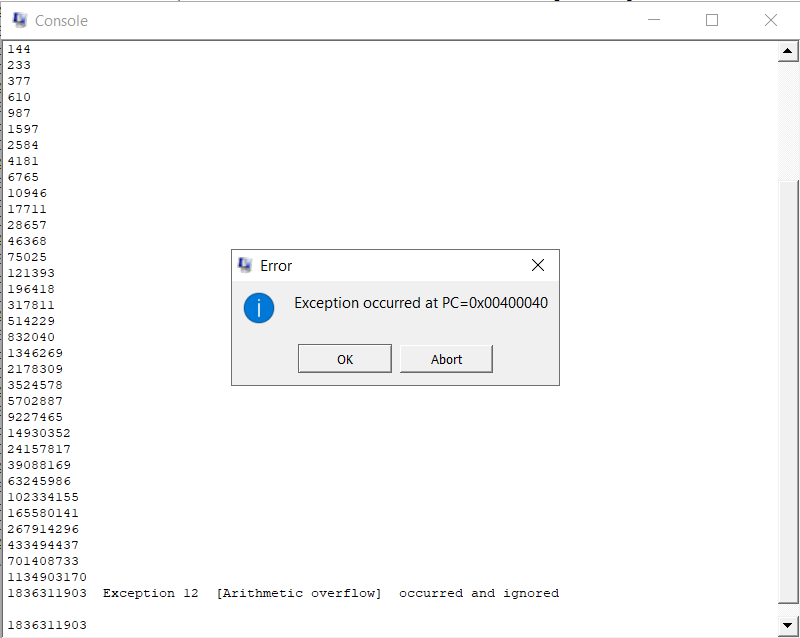
move $a0, $t1

li $v0, 1

syscall

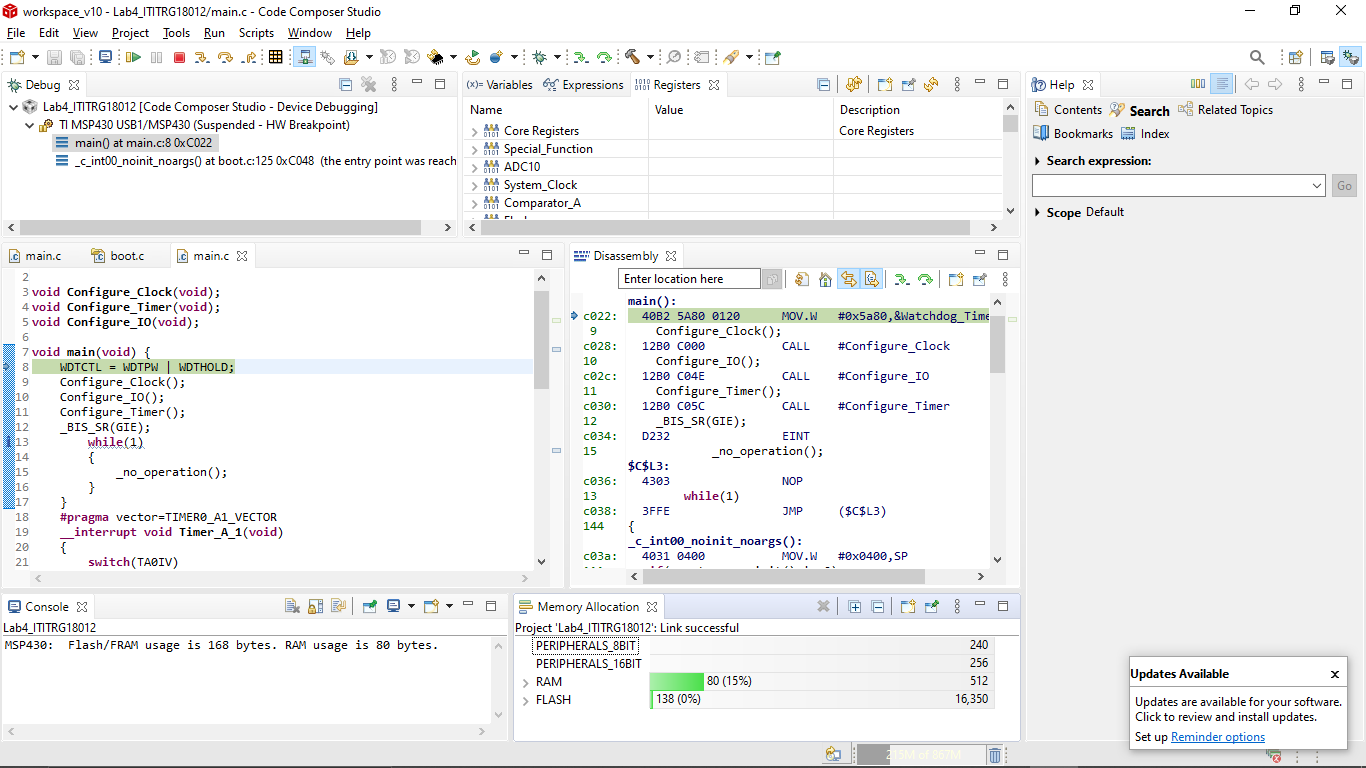
li $v0, 10

syscall

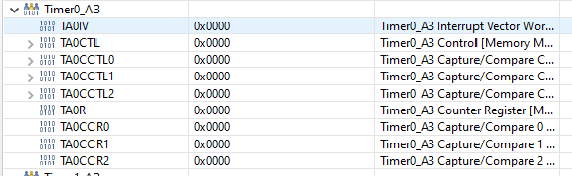


II/

**\*Evaluation memory allocation:**

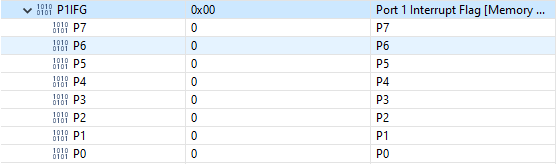


**\*Observe the register of Timer0\_A3 in Core registers:**

****

**Observe the register of Interrupt Flag in Core registers:**

**P1IFG :**



**P2IFG**

