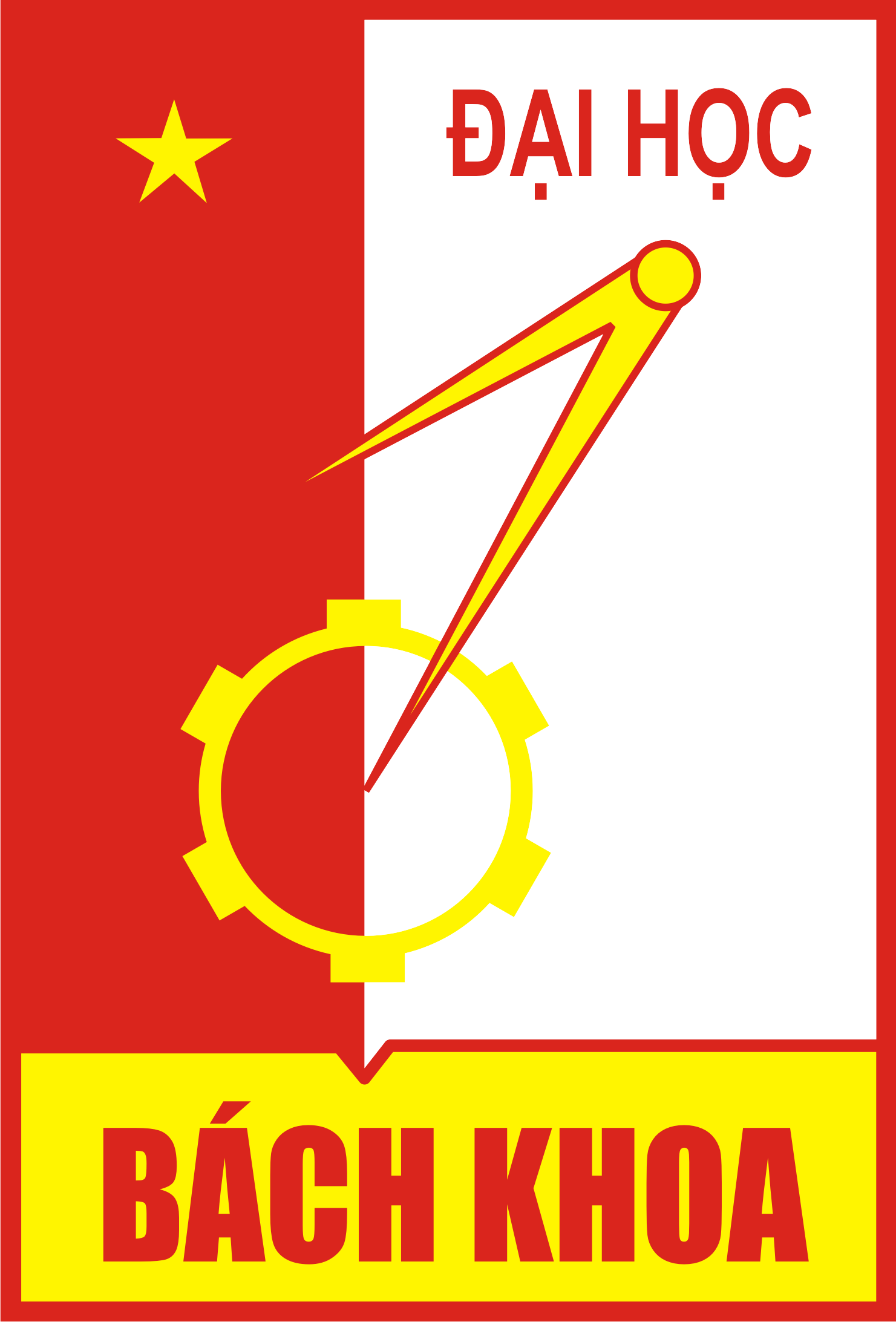
**ĐẠI HỌC BÁCH KHOA HÀ NỘI  
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**BẢN BÁO CÁO PROJECT**

**THỰC HÀNH KIẾN TRÚC MÁY TÍNH 2021.2**

***Đề tài:***

Xây dựng Máy tính bỏ túi bằng phần mềm MARS 4.5

Vẽhình trên màn hình Bitmap

**Giáo viên hướng dẫn**

*Lê Bá Vui*

**Sinh viên**

*Hoàng Trọng Tân 20194664*

*Phạm Nguyễn Gia Hòa 20194566*

**Bài 2: Vẽ hình trên màn hình Bitmap**

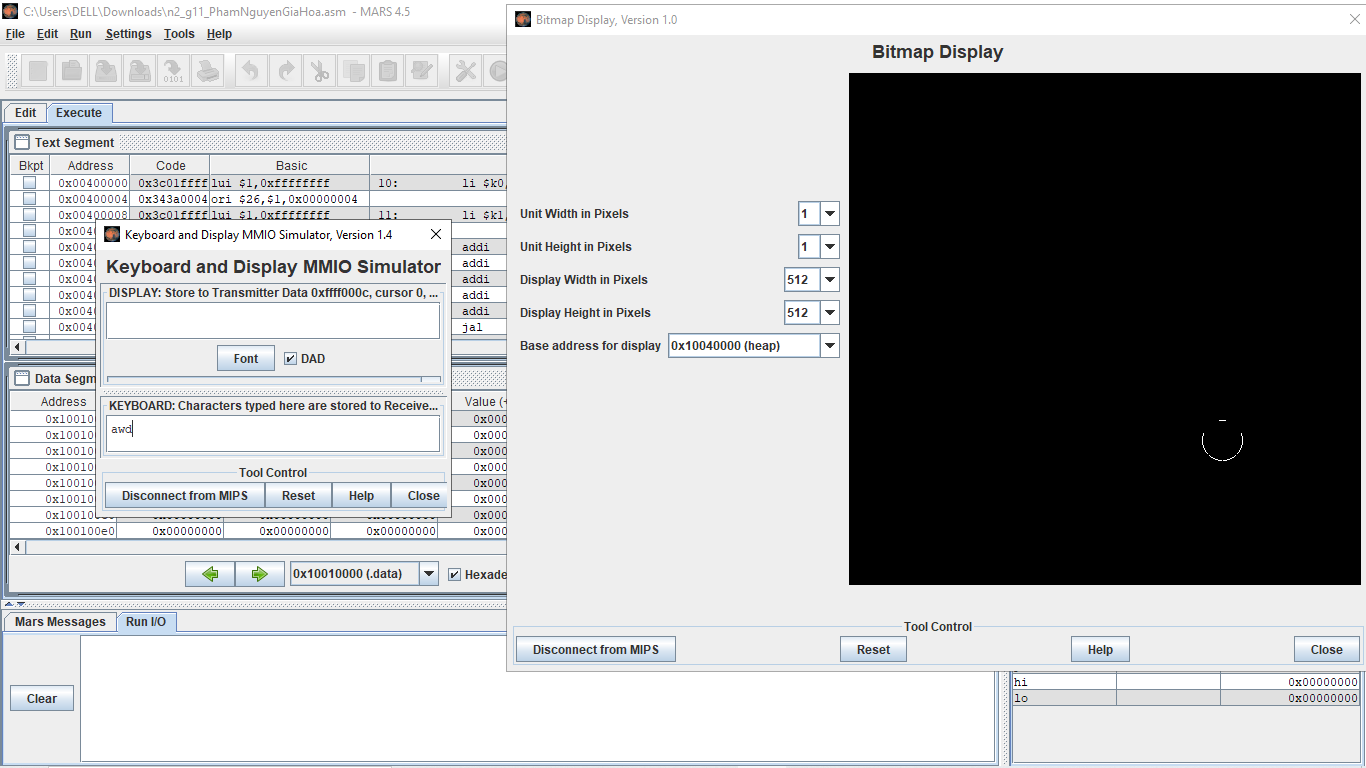
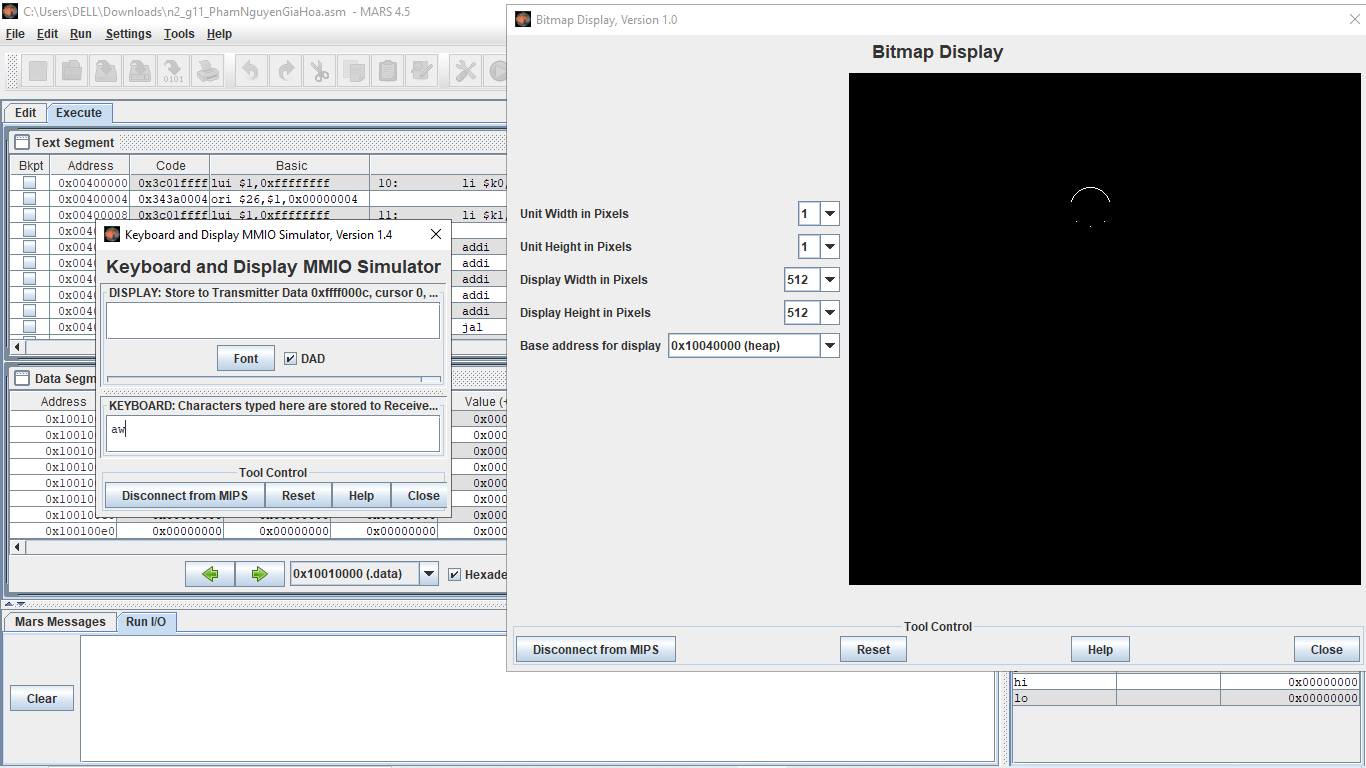
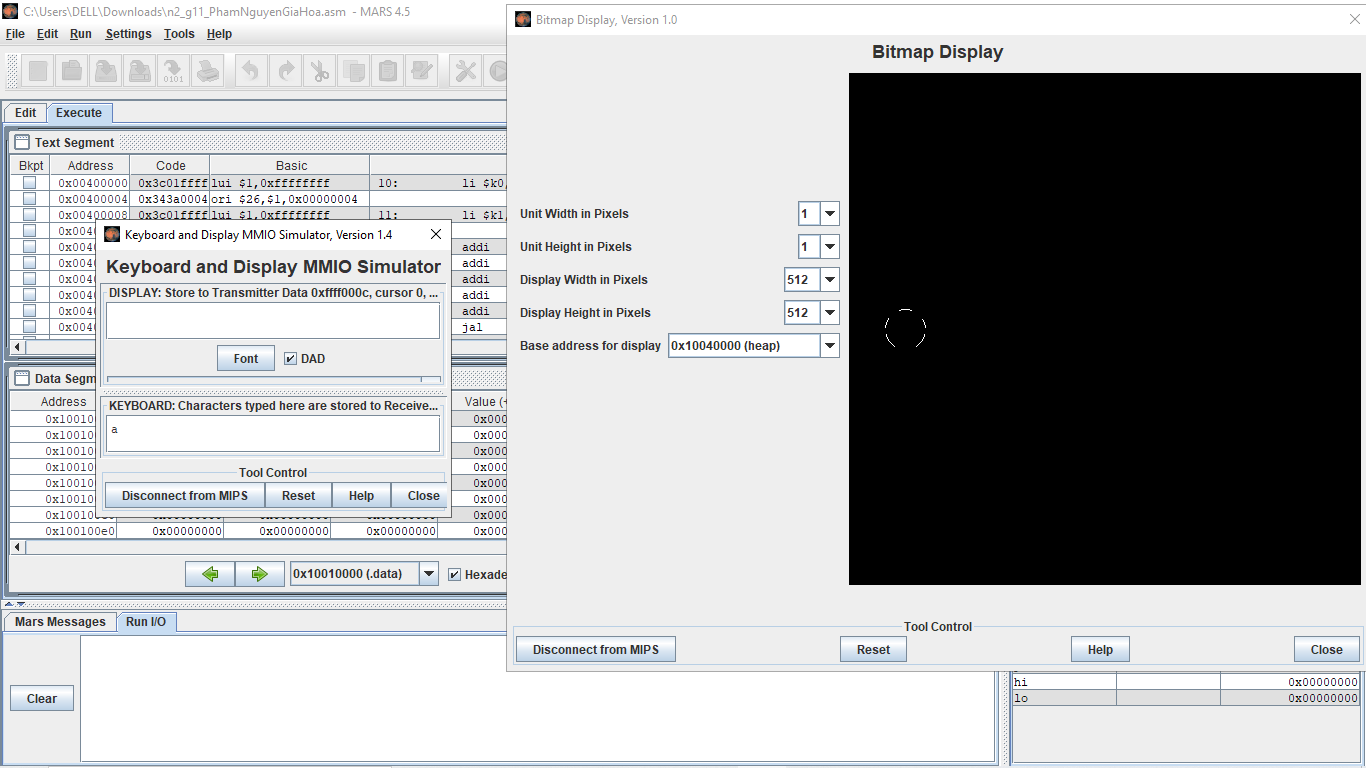
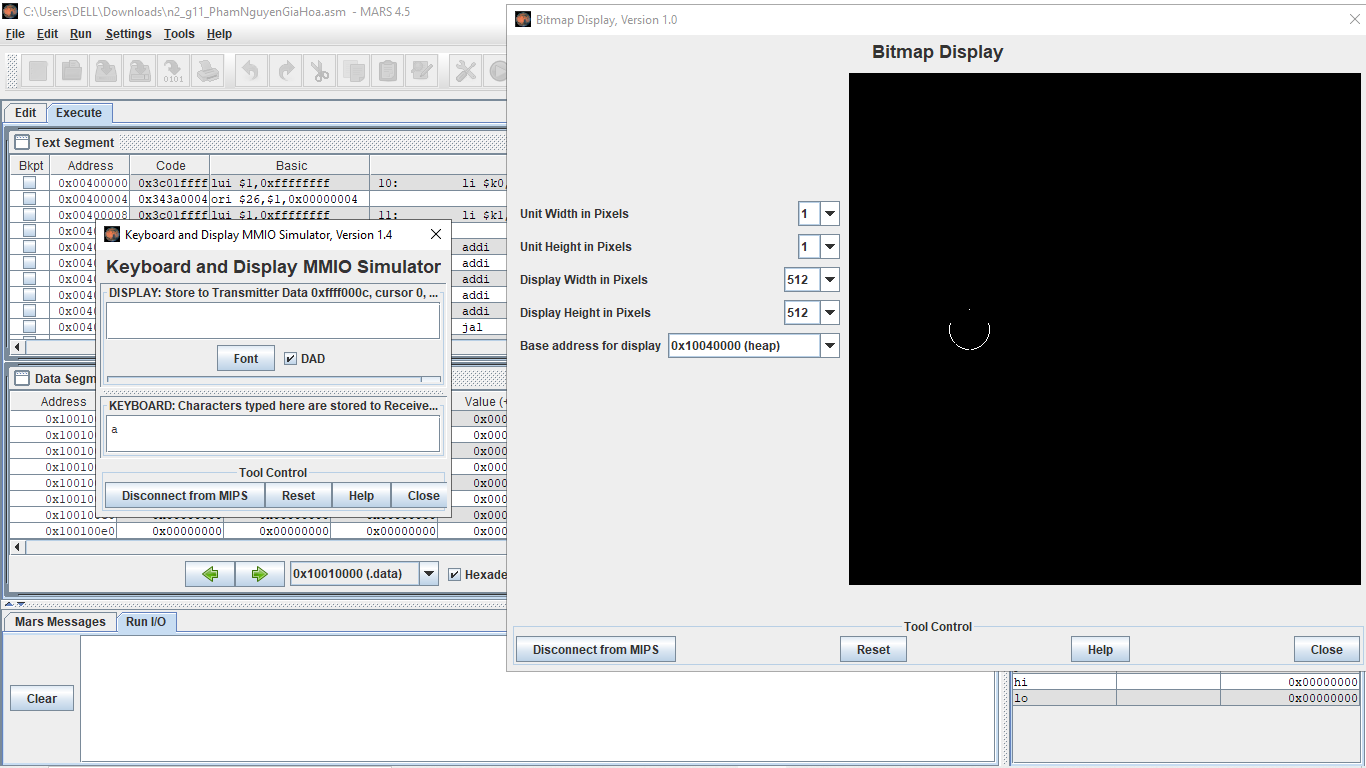
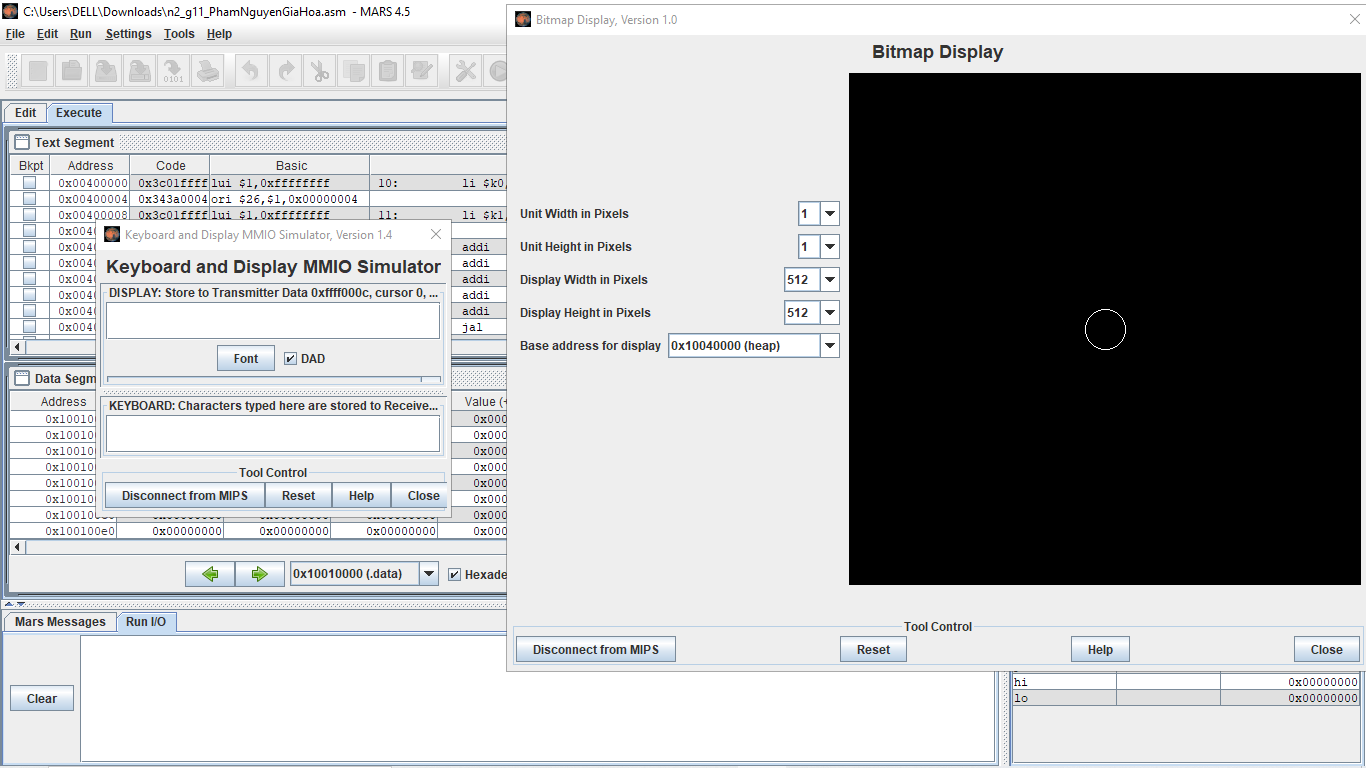
* **Phân tích cách làm:**

+ Vẽ đường tròn với vị trí giữa màn hình (x,y) = (256,256)

+ Nhập phím

+ Input :lưu mã ASCII của phím vào $k0

+ Xét phím nhập vào:

* Nếu chưa có nhảy đến input
* Nếu các phím nhập vào có mã ASCII bằng 97, 100, 115, 119 nhảy đến phần tương ứng: left, right, down, up và
  + Left: Xóa đường tron cũ, vẽ đường tròn mới dịch trái 1 (x=x-1), lặp lại liên tục đến khi
    - Tâm đường tròn có tọa độ x=20 thì sleep rồi reboundRight (thay đổi giá trị $k0=100 rồi nhảy đến input)
    - Nhập phím mới và nhảy đến input
  + Right: Tương tự
  + Down: Tương tự
  + Up: Tương tự
* Thuật toán: Brehasan (vẽ đường tròn)
* Kết quả chạy:

**Bài 10: Xây dựng Máy tính bỏ túi bằng phần mềm MARS 4.5**

* **Mô tả một số dữ liệu đầu vào:**

**+ digit:** Mảng chứa các mã code cần lưu vào thanh ghi 7 đoạn để hiển thị số tương ứng.

**+ key\_code:** Mảng chứa các mã code của các phím có thể được nhận từ Digital Lab Sim.

* **Mô tả các thanh ghi sử dụng chính:**

+ s0: Lưu giá trị lỗi.

0 : Không có lỗi.

-1: Lỗi chia cho 0.

-2: Lỗi tràn bộ nhớ do phép cộng.

-3: Lỗi tràn bộ nhớ do phép trừ.

-4: Lỗi tràn bộ nhớ do phép nhân.

+ s1: Lưu giá trị toán hạng thứ nhất, khởi tạo giá trị bằng 0.

+ s2: Lưu giá trị toán tử:

10: Toán tử cộng.

11: Toán tử trừ.

12: Toán tử nhân.

13: Toán tử chia.

14: Toán tử lấy đồng dư.

+ s3: Lưu giá trị toán hạng thứ hai, khởi tạo giá trị bằng 0.

* **Mô tả các chương trình con:**

**+ show\_int:**

CHỨC NĂNG: Hiển thị lên thanh 7 đoạn giá trị 2 chữ số cuối của toán hạng. Nếu chưa nhập toán tử thì hiển thị toán hạng 1, ngược lại hiển thị toán hạng 2. Sử dụng thanh ghi $a0 để lưu giá trị cần hiển thị. Nếu toán hạng thuộc [-9, -1] hiển thị dấu trừ ở đèn led 7 đoạn bên trái.

INPUT: Các thanh ghi s1, s2, s3. (Toán hạng 1, toán tử, toán hạng 2).

OUTPUT: Nếu s2 = -1 hiển thị 2 chữ số cuối của toán hạng s1, ngược lại hiển thị 2 chữ số cuối của toán hạng s3.

**+ mod\_10:**

CHỨC NĂNG: Đưa vào 1 số, trả về giá trị cuối của chữ số đó.

INPUT: Thanh ghi a1 chứa số cần tìm chữ số cuối.

OUTPUT: Thanh ghi v1 = a1 mod 10.

**+ input**

CHỨC NĂNG: Bật INTERUPT,

get\_cod: Kiểm tra kí tự nhập vào từ Digital Lab Sim, lưu giá trị phím được nhấn vào thanh ghi a0.

map\_key: Chuyển giá trị phím được nhấn (lưu ở thanh ghi a0) về đoạn [0, 15] lưu vào thanh ghi s4.

INPUT: Sử dụng INTERUPT bắt phím được nhấn, lưu vào thanh ghi a0.

OUTPUT: Thanh ghi s4 thuộc [0;15] tương ứng với phím được nhấn.

**+ process**

CHỨC NĂNG: Nhận vào giá trị từ thanh ghi s4 thuộc [0;15]. In phím, tính toán, trả về lỗi cho thanh ghi s0.

Nếu là số -> kiểm tra s2=-1, chưa nhập toán tử, thêm giá trị phím vừa nhập vào toán hạng thứ nhất. Ngược lại thêm giá trị phím vừa nhập vào toán hạng thứ 2.

Nếu là toán tử-> Nếu là toán tử đầu tiên -> in toán tử. Ngược lại, tính kết quả, in kết quả và toán tử. Nếu tính kết quả xảy ra lỗi -> in ra lỗi, in giá trị toán hạng thứ nhất. Gán giá trị toán tử về -1. Gán giá trị toán hạng 2 về 0.

Nếu là dấu bằng -> in dấu bằng, tính toán, in kết quả.

INPUT: Thanh ghi s4 thuộc [0;15].

OUTPUT: In ra toán tử, hoặc toán hạng, hoặc kết quả. Nếu tính kết quả xảy ra lỗi, trả về lỗi cho thanh ghi s0.

**+ calculate:**

CHỨC NĂNG: Nhận toán tử, toán hạng. Tính kết quả, báo lỗi

Nếu toán tử là dấu cộng, kiểm tra dấu 2 toán hạng. Nếu khác dấu -> thoát, ngược lại nếu kết quả khác dấu với toán hạng thứ nhất. Báo lỗi bằng thanh ghi s0.

Nếu toán tử là dấu trừ, kiểm tra dấu 2 toán hạng. Nếu cùng dấu thoát,

ngược lại nếu kết quả khác khác với toán hạng thứ nhất. Báo lỗi bằng thanh ghi s0.

Nếu toán tử là dấu nhân, nhân 2 toán hạng. Nếu nếu thanh ghi hi khác 0 báo lỗi bằng thanh ghi s0.

Nếu toán tử là dấu chia. Kiểm tra toán hạng. Nếu toán hạng 2 bằng 0, báo lỗi bằng thanh ghi s0.

* Nếu không lỗi thì thực hiện tính toán, lưu kết quả vào thanh ghi s1. Đặt thanh ghi s2=-1, s3=0.

INPUT: Thanh ghi s1, s2, s3. (Toán hạng thứ nhất, toán tử, toán hạng thứ 2).

OUTPUT: Thanh ghi s1 chứa kết quả

Thanh ghi s2 =-1, s3=0.

Thanh ghi s0 chứa mã lỗi.

**+ print\_operator**

CHỨC NĂNG: Nhận giá trị từ thanh ghi s4 thuộc [10,14] in ra toán tử tương ứng.

INPUT: Thanh ghi s4.

OUTPUT: Toán tử tương ứng với giá trị thanh ghi s4.

**+ handle\_exception**

CHỨC NĂNG: Nhận giá trị từ thanh ghi s0, in lỗi tương ứng, in kết quả trước khi xảy ra lỗi.

INPUT: Thanh ghi s0.

OUTPUT: Lỗi tương ứng và kết quả trước khi xảy ra lỗi.

* **Kết quả:**

**Graphical user interface, application, table, Excel

Description automatically generated**

Hình 1. Kết quả khi chương trình chạy bình thường.

Graphical user interface, application, Excel

Description automatically generated

Hình 2. Lỗi khi chia cho 0.

Graphical user interface, text, application, Excel

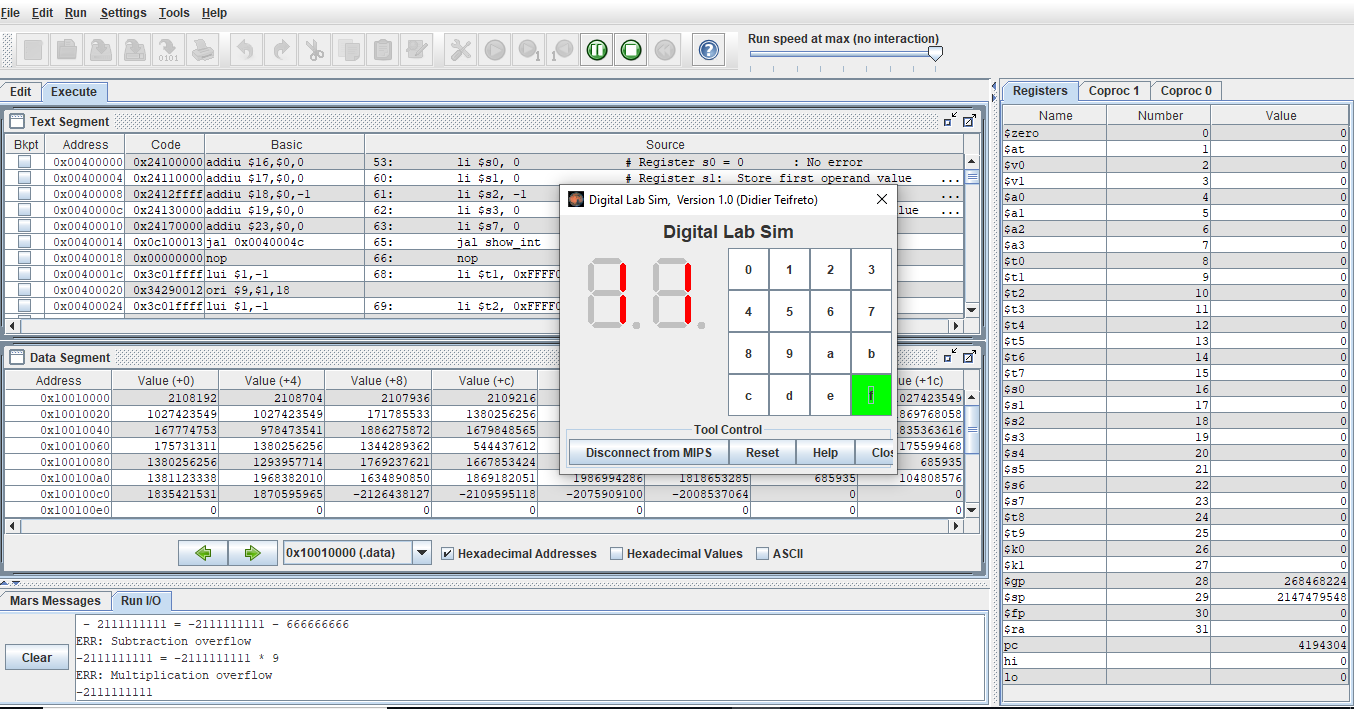
Description automatically generated

Hình 3. Lỗi khi nhập số quá lớn.

Graphical user interface, application

Description automatically generated

Hình 4. Lỗi khi trừ số tràn bộ nhớ



Hình 5: Lỗi khi nhân số tràn bộ nhớ.

Graphical user interface, application, Word

Description automatically generated

Hình 6. Lỗi khi cộng tràn bộ nhớ.

Bài 2 (mã nguồn)

.eqv KEY\_CODE 0xFFFF0004 # ASCII code to show, 1 byte

.eqv KEY\_READY 0xFFFF0000 # =1 if has a new keycode ?

# Auto clear after lw

.data

L : .asciiz "a"

R : .asciiz "d"

U: .asciiz "w"

D: .asciiz "s"

.text

li $k0, KEY\_CODE # chua k� tu nhap vao

li $k1, KEY\_READY # kiem tra da nhap phim nao chua

addi $s7, $0, 512 #store the width in s7

#circle:

addi $a0, $0, 256 #x = 256

addi $a1, $0, 256 #y = 256

addi $a2, $0, 20 #r = 20

addi $s0, $0, 0xFFFFFFFF

jal DrawCircle

nop

moving:

beq $t0,97,left #a

beq $t0,100,right #d

beq $t0,115,down #s

beq $t0,119,up #w

j Input

left:

addi $s0,$0,0x00000000 # delteCircle

jal DrawCircle

addi $a0,$a0,-1 #x= x-1

add $a1,$a1, $0 #y=y

addi $s0,$0,0xFFFFFFFF #drawNewCircle

jal DrawCircle

jal Pause # x=20

bltu $a0,20,reboundRight #??i h??ng

j Input

right:

addi $s0,$0,0x00000000 # delteCircle

jal DrawCircle

addi $a0,$a0,1 #x= x+1

add $a1,$a1, $0 #y=y

addi $s0,$0,0xFFFFFFFF #drawNewCircle

jal DrawCircle

jal Pause # x=492

bgtu $a0,492,reboundLeft #??i h??ng

j Input

up:

addi $s0,$0,0x00000000 # delteCircle

jal DrawCircle

addi $a1,$a1,-1 #y=y-1

add $a0,$a0,$0 #x=x

addi $s0,$0,0xFFFFFFFF #drawNewCircle

jal DrawCircle

jal Pause #when y=20

bltu $a1,20,reboundDown #??i h??ng

j Input

down:

addi $s0,$0,0x00000000 # delteCircle

jal DrawCircle

addi $a1,$a1,1 #y=y+1

add $a0,$a0,$0 #x=x

addi $s0,$0,0xFFFFFFFF #drawNewCircle

jal DrawCircle

jal Pause #when y=492

bgtu $a1,492,reboundUp #??i h??ng

j Input

reboundLeft:

li $t3 97

sw $t3,0($k0)

j Input

reboundRight:

li $t3 100

sw $t3,0($k0)

j Input

reboundDown:

li $t3 115

sw $t3,0($k0)

j Input

reboundUp:

li $t3 119

sw $t3,0($k0)

j Input

endMoving:...

Input:

ReadKey: lw $t0, 0($k0) # $t0 = [$k0] = KEY\_CODE

j moving

Pause:

addiu $sp,$sp,-4

sw $a0, ($sp)

la $a0,0 # speed =20ms

li $v0, 32 #syscall value for sleep

syscall

lw $a0,($sp)

addiu $sp,$sp,4

jr $ra

DrawCircle:

#a0 = cx

#a1 = cy

#a2 = radius

#s0 = colour

addiu $sp, $sp, -32

sw $ra, 28($sp)

sw $a0, 24($sp)

sw $a1, 20($sp)

sw $a2, 16($sp)

sw $s4, 12($sp)

sw $s3, 8($sp)

sw $s2, 4($sp)

sw $s0, ($sp)

#code goes here

sub $s2, $0, $a2 #error = -radius

add $s3, $0, $a2 #x = radius

add $s4, $0, $0 #y = 0 (s4)

DrawCircleLoop:

bgt $s4, $s3, exitDrawCircle #if y is greater than x, break the loop (while loop x >= y)

nop

#plots 4 points along the right of the circle, then swaps the x and y and plots the opposite 4 points

jal plot8points

nop

add $s2, $s2, $s4 #error += y

addi $s4, $s4, 1 #++y

add $s2, $s2, $s4 #error += y

blt $s2, 0, DrawCircleLoop #if error >= 0, start loop again

nop

sub $s3, $s3, 1 #--x

sub $s2, $s2, $s3 #error -= x

sub $s2, $s2, $s3 #error -= x

j DrawCircleLoop

nop

exitDrawCircle:

lw $s0, ($sp)

lw $s2, 4($sp)

lw $s3, 8($sp)

lw $s4, 12($sp)

lw $a2, 16($sp)

lw $a1, 20($sp)

lw $a0, 24($sp)

lw $ra, 28($sp)

addiu $sp, $sp, 32

jr $ra

nop

plot8points:

addiu $sp, $sp -4

sw $ra, ($sp)

jal plot4points

nop

beq $s4, $s3, skipSecondplot

nop

#swap y and x, and do it again

add $t2, $0, $s4 #puts y into t2

add $s4, $0, $s3 #puts x in to y

add $s3, $0, $t2 #puts y in to x

jal plot4points

nop

#swap them back

add $t2, $0, $s4 #puts y into t2

add $s4, $0, $s3 #puts x in to y

add $s3, $0, $t2 #puts y in to x

skipSecondplot:

lw $ra, ($sp)

addiu $sp, $sp, 4

jr $ra

nop

plot4points:

#plots 4 points along the right side of the circle, then swaps the cd and cy values to do the opposite side

#if statements are for optimisation, they work if the branches are removed

addiu $sp, $sp -4

sw $ra, ($sp)

#$a0 = a0 + s3, $a2 = a1 + s4

add $t0, $0, $a0 #store a0 (cx in t0)

add $t1, $0, $a1 #store a2 (cy in t1)

add $a0, $t0, $s3 #set a0 (x for the setpixel, to cx + x)

add $a2, $t1, $s4 #set a2 (y for setPixel to cy + y)

jal SetPixel #draw the first pixel

nop

sub $a0, $t0, $s3 #cx - x

#add $a2, $t1, $s4 #cy + y

beq $s3, $0, skipXnotequal0 #if s3 (x) equals 0, skip

nop

jal SetPixel #if x!=0 (cx - x, cy + y)

nop

skipXnotequal0:

sub $a2, $t1, $s4 #cy - y (a0 already equals cx - x

jal SetPixel #no if (cx - x, cy - y)

nop

add $a0, $t0, $s3

beq $s4, $0, skipYnotequal0 #if s4 (y) equals 0, skip

nop

jal SetPixel #if y!=0 (cx + x, cy - y)

nop

skipYnotequal0:

add $a0, $0, $t0

add $a2, $0, $t1

lw $ra, ($sp)

addiu $sp, $sp, 4

jr $ra

nop

SetPixel:

#a0 x

#a1 y

#s0 colour

addiu $sp, $sp, -20 # Save return address on stack

sw $ra, 16($sp)

sw $s1, 12($sp)

sw $s0, 8($sp) # Save original values of a0, s0, a2

sw $a0, 4($sp)

sw $a2, ($sp)

lui $s1, 0x1004 #starting address of the screen

sll $a0, $a0, 2 #multiply 4

add $s1, $s1, $a0 #x co-ord addded to pixel position

mul $a2, $a2, $s7 #multiply by width (s7 declared at top of program, never saved and loaded and it should never be changed)

mul $a2, $a2, 4 #myltiply by the size of the pixels (4)

add $s1, $s1, $a2 #add y co-ord to pixel position

sw $s0, ($s1) #stores the value of colour into the pixels memory address

lw $a2, ($sp) #retrieve original values and return address

lw $a0, 4($sp)

lw $s0, 8($sp)

lw $s1, 12($sp)

lw $ra, 16($sp)

addiu $sp, $sp, 20

jr $ra

nop

Bài 10 (mã nguồn):

# Project 10

# Group: 11

# Author: Hoang Trong Tan

.eqv IN\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0012

.eqv OUT\_ADRESS\_HEXA\_KEYBOARD 0xFFFF0014

.eqv SEVENSEG\_LEFT 0xFFFF0011 # The address of the left 7-segment led

.eqv SEVENSEG\_RIGHT 0xFFFF0010 # The address of the right 7-segment led

.data

add\_msg: .asciiz " + "

sub\_msg: .asciiz " - "

mul\_msg: .asciiz " \* "

div\_msg: .asciiz " / "

mod\_msg: .asciiz " % "

result\_msg: .asciiz " = "

msg\_new\_line: .asciiz "\n"

divider\_msg: .asciiz "=================\n"

div\_err\_msg: .asciiz "\nERR: Divie by zero!\n"

input\_overflow\_msg: .asciiz "\nERR: Input data overflow memory\n"

add\_overflow\_msg: .asciiz "\nERR: Plus memory overflow\n"

mul\_overflow\_msg: .asciiz "\nERR: Multiplication overflow\n"

sub\_overflow\_msg: .asciiz "\nERR: Subtraction overflow\n"

\_DOT: .byte 0x80 # Show dot sign

NEG: .byte 0x40 # Show minus sign if input Number is Negative

digit: # Digit: Display numbers by 7-segment led

ZERO: .byte 0x3F

ONE: .byte 0x06

TWO: .byte 0x5b

THREE: .byte 0x4f

FOUR: .byte 0x66

FIVE: .byte 0x6D

SIX: .byte 0x7D

SEVENT: .byte 0x07

EIGHT: .byte 0x7F

NINE: .byte 0x6F

# Key code of key pressed in Digi Lab Sim

key\_code: .byte 0x11, 0x21, 0x41, 0x81,

0x12, 0x22, 0x42, 0x82,

0x14, 0x24, 0x44, 0x84,

0x18, 0x28, 0x48, 0x88

.text

#---------------------------------------------------------------

# main: Use INTERUPT

# main\_loop

# Show nuber by 7-segment led

#---------------------------------------------------------------

main:

# Global variables

li $s0, 0 # Register s0 = 0 : No error

# Register s0 = -1 : ERR: Divide by 0

# Register s0 = -2 : ERR: Plus overflow

# Register s0 = -3 : ERR: Subtraction overflow

# Register s0 = -4 : ERR: Multiplitation overflow

# Register s0 = -5 : ERR: Input data overflow

li $s1, 0 # Register s1: Store first operand value default s1 = 0

li $s2, -1 # Register s2: Operator default s2 = -1

li $s3, 0 # Register s3: Store second operand value default s1 = -1

li $s7, 0

main\_loop:

jal show\_int # Show nuber by 7-segment led

nop

li $t1, IN\_ADRESS\_HEXA\_KEYBOARD

li $t2, OUT\_ADRESS\_HEXA\_KEYBOARD

jal input # Input data from Digi Lab Sim

nop # 0 - 9: Operand, 10 - 13: Operator, 15: Equal sign

# s4 0-15, s6 prev keypressed

jal process #

nop # Store operator value to $s2

# Store operand value to $s1 of $s3

# caculate when press f (=)

jal handle\_exception # Catch ERRORS

nop

main\_next:

j main\_loop # Use loop to input data

nop

end\_main:

# li $v0, 10

# syscall

#---------------------------------------------------------------

# Procedure show\_int

# brief: Show operand:

# - Haven't entered operator: s1

# - Have entered operator: s3

# If operand is nagative show '-' on the left position

# param[in] s1 first operand

# param[in] s2 operation

# param[in] s3 second operand

# param[in] $a1 value to shown in binary

#---------------------------------------------------------------

show\_int:

addi $sp, $sp, -4

sw $ra, 0($sp) # Push value of Register ra to Stack

load\_num:

add $a1, $s1, $zero # Default show the value of Register s1 on 7-segment led

bne $s2, -1, show\_s3 # if s2 is not default (entered operand), show the value of Register s3 on 7-segment led

nop

j show\_a1\_value

nop

show\_s3:

add $a1, $s3, $zero # Show the value of Register s3 on 7-segment led

show\_a1\_value:

jal mod\_10 # Procedure $v1 := $a1 mod 10

nop

la $t3, digit # $t1 := adress of Array Digit

add $t0, $t3, $v1

lb $a0, 0($t0) # Set value for segments

li $t0, SEVENSEG\_RIGHT # Assign port's address

sb $a0, 0($t0) # assign new value

bge $a1, 0, show\_left\_pos # If a1 > 0, show left position

nop

blt $a1, -9, show\_left\_pos # If a1 < -9, show left position

nop

# SHOW - (minus)

lb $a0, NEG # Show -

li $t0, SEVENSEG\_LEFT # Assign port's address

sb $a0, 0($t0) # assign new value

j end\_show\_int # Return

nop

show\_left\_pos:

div $a1, $a1, 10 # a1 = a1 / 10

jal mod\_10 # v1 = a1 mod 10

nop

add $t0, $t3, $v1

lb $a0, 0($t0) # Set value for segments

li $t0, SEVENSEG\_LEFT # Assign port's address

sb $a0, 0($t0) # assign new value

end\_show\_int:

lw $ra, 0($sp) # Load $ra

addi $sp, $sp, 4 # Pop $ra

jr $ra # Return main

nop

#---------------------------------------------------------------

# Procedure mod\_10:

# brief: v1 = abs(a1 - int(a1/10) \* 10)

# param[in] $a1 value to shown in binary

# param[out] $v1 = $a1 mod 10 in binary

#---------------------------------------------------------------

mod\_10:

add $v1, $a1, $zero # t0 := a2 = input

div $v1, $v1, 10 # v1 = input / 10

mul $v1, $v1, 10 # v1 = v1 \* 10

sub $v1, $a1, $v1 # v1 = a1 - v1

bge $a1, 0, end\_mod\_10 # return

nop

# mod 10 neg

sub $v1, $zero, $v1 # if v1 < 0 -> v1 = -v1

end\_mod\_10:

jr $ra # return

nop

#---------------------------------------------------------------

# Procedure input

# Digital lab sim

# - s4: keycode of key pressed

# - s5: Read row 0x01 0x02 0x04 0x08

# param[out] s4: key pressed [0..15], s4 = -1 when no key is pressed

# param[out] s6: prev key pressed

#---------------------------------------------------------------

input:

loop\_init:

li $t3, 0x80 # Turn on interupt bit

sb $t3, 0($t1)

loop:

sleep: addi $v0, $zero, 32

li $a0, 1 # sleep 1ms

syscall

b loop

nop

end\_input:

jr $ra # return

nop

#---------------------------------------------------------------

# Procedure process

# param[in]: s1: The first operand

# param[in]: s2: Operation

# param[in]: s3: The second operand

# param[in]: s4: pressed key

# param[in]: s6: prev pressed key

#---------------------------------------------------------------

process:

subi $sp, $sp, 4 # Push ra to Stack

sw $ra, 0($sp)

new\_key\_pressed: # 3 case: Equal sign || Operation || Number

beq $s4, 15, equal # case 1: Equal sign

nop

bgt $s4, 9, new\_operator # case 2: Operation

nop

number: # case 3: Number

bne $s2, -1, add\_s3 # if it is operation, input second operand

nop

add\_s1: # If there is no operator, input first operand

mul $t0, $s1, 10 # s1 = s1 \* 10

mfhi $t1 # move valute to $t1

beq $t1, 0, add\_s1\_ # if t1 == 0 -> ok

nop # != 0 -> overflow

# throw error

error\_input\_1:

li $s0, -5

j end\_process # return

nop

add\_s1\_: # if not overflow

move $s1, $t0

add $s1, $s1, $s4 # s1 = s1 + s4

andi $t1, $s1, 0x80000000 # t1 = s1 sign bit

bne $t1, $zero, error\_input\_1

nop

# print

li $v0, 1

add $a0, $s4, $zero

syscall

j end\_process # return

nop

add\_s3: # input to second operand

mul $t0, $s3, 10

mfhi $t1

beq $t1, 0, add\_s3\_ # check overflow

nop

# throw error

error\_input\_2:

li $s0, -5

j end\_process

nop

add\_s3\_:

move $s3, $t0

add $s3, $s3, $s4

# print

li $v0, 1

add $a0, $s4, $zero

syscall

andi $t1, $s3, 0x80000000 # t1 = s3 sign bit

bne $t1, $zero, error\_input\_2

nop

j end\_process # return

nop

new\_operator: # case: it is operator (+, -, \*, /, %)

#---------------- caculate result when second operator

beq $s7, $zero, new\_operator\_ # check the first operator

nop

jal calculate

nop

bne $s0, 0, end\_process # If have error, no PRINT

nop

# Print result

li $v0, 4

la $a0, result\_msg # show result msg

syscall

li $v0, 1

add $a0, $s1, $zero # show result ( in s1 )

syscall

#-----------------

# print

new\_operator\_:

addi $s7, $s7, 1 # count the operator

add $s2, $zero, $s4 # s2 = s4

jal print\_operator

nop

j end\_process

nop

equal: # Case: Equal sign =

jal calculate # Caculate

nop

bne $s0, 0, after\_print # If have error, no PRINT

nop

# Print result

li $v0, 4

la $a0, result\_msg # show result msg

syscall

li $v0, 1

add $a0, $s1, $zero # show result ( in s1 )

syscall

after\_print:

j end\_process # return

nop

end\_process:

lw $ra, 0($sp) # Pop ra from Stack

addi $sp, $sp, 4

jr $ra # return

nop

#------------------------------------------------------

# Procedure calculate

# Caculate the expression

# Param[in] $s1 operand 1

# Param[in] $s2 operator

# Param[in] $s3 operand 2

# Param[out]$s1 result, s2 := -1, s3 :=0

# Param[out]$s0 ERROR Code

#------------------------------------------------------

calculate:

# Check the operator

beq $s2, 10, calc\_add # s2 is plus

nop

beq $s2, 11, calc\_sub # s2 is subtraction

nop

beq $s2, 12, calc\_mul # s2 is multiplication

nop

beq $s2, 13, calc\_div # s2 is divide

nop

beq $s2, 14, calc\_mod # s2 is module

nop

j end\_calculate # if s2 is not operator

nop

calc\_add: # Plus

andi $t1, $s1, 0x80000000 # t1 = sign bit of s1

andi $t3, $s3, 0x80000000 # t3 = sign bit of s3

seq $t2, $t1, $t3 # t2 = 0 if t1 and  t3 are same sign

s1\_add\_s3:

addu $t0, $s1, $s3 # t0 = s1 + s3

andi $t4, $t0, 0x80000000 # t4 = bit

# handle error

bne $t2, 1, after\_check\_add\_err # If t1 and t3 are not same sign -> Don't need to check

nop

beq $t1, $t4, after\_check\_add\_err # if result is same sign with s1 -> NOT overflow

nop

throw\_add\_overflow:

li $s0, -2 # s0 = -2: is overflow

j calc\_after # reset and return

nop

after\_check\_add\_err:

add $s1, $s1, $s3 # s1 = s1 + s3

j calc\_after # reset s2 = -1, s3 = 0

nop

calc\_sub: # Subtraction

andi $t1, $s1, 0x80000000 # t1 = sign bit s1

andi $t3, $s3, 0x80000000 # t3 = sign bit s3

sne $t2, $t1, $t3 # t2 = 1 if t1 and  t3 are not same sign

bne $t2, 1, after\_check\_sub\_err # if t1 and t3 is same sign -> Not overflow

s1\_sub\_s3:

subu $t0, $s1, $s3 # t0 = s1 - s3

andi $t4, $t0, 0x80000000 # t4 = sign bit of t0

nop

beq $t1, $t4, after\_check\_sub\_err # if result and s1 are same sign -> NOT overflow

nop

throw\_sub\_overflow:

li $s0, -3 # s0 = -3: sub overflow

j calc\_after # reset and return

nop

after\_check\_sub\_err:

sub $s1, $s1, $s3 # s1 = s1 - s3

j calc\_after

nop

calc\_mul: # Multiplication

mul $t0, $s1, $s3 # t0 = s1 \* s3

mfhi $t1

beq $t1, 0, calc\_mul\_

nop

# handle error

li $s0, -4

j calc\_after

nop

calc\_mul\_:

move $s1, $t0 # s1 = t0

j calc\_after

nop

calc\_div: # Divide

bne $s3, 0, divisor\_neq\_zero # if not divide by zero

nop

# divisor\_eq\_zero: # if divide by zero

li $s0, -1 # throw error

j calc\_after

nop

divisor\_neq\_zero: # if s3 != 0

div $s1, $s1, $s3 # s1 = s1 / s3

j calc\_after

nop

calc\_mod: # Module

div $t0, $s1, $s3

mul $t0, $t0, $s3

sub $s1, $s1, $t0

bge $s1, 0, calc\_after # return

nop

# s1 is neg

sub $s1, $zero, $s1 # if s1 < 0 -> s1 = -s1

j calc\_after

nop

calc\_after: # Reset second operand, reset operator

li $s3, 0 # reset second operand

li $s2, -1 # reset operator = -1

end\_calculate:

jr $ra

nop

# ---------------------------------------------

# Procedure print\_operator

# Print operator

# param[in]: s1: phep tinh

# - Plus: s1 = 0xa

# - Subtraction: s1 = 0xb

# - ...

# ---------------------------------------------

print\_operator:

beq $s2, 10, print\_add # s2 == 10 : Plus

nop

beq $s2, 11, print\_sub # s2 == 11 : Subtraction

nop

beq $s2, 12, print\_mul # s2 == 12 : Multiplication

nop

beq $s2, 13, print\_div # s2 == 13 : Divide

nop

beq $s2, 14, print\_mod # s2 == 14 : Module

nop

j end\_print\_operator # End print operator

nop

print\_add:

la $a0, add\_msg

j print\_operator\_

nop

print\_sub:

la $a0, sub\_msg

j print\_operator\_

nop

print\_mul:

la $a0, mul\_msg

j print\_operator\_

nop

print\_div:

la $a0, div\_msg

j print\_operator\_

nop

print\_mod:

la $a0, mod\_msg

j print\_operator\_

nop

print\_operator\_:

li $v0, 4

syscall

end\_print\_operator:

jr $ra

nop

#------------------------------------------------------

# Handle eror

# Print ERR if has

# Param[in]: $s0 ERROR CODE

#----------------------------------------------------

handle\_exception:

beq $s0, -1, handle\_div\_err # ERR: Divide by zero

nop

beq $s0, -2, handle\_add\_overflow # ERR: Plus overflow

nop

beq $s0, -4, handle\_mul\_overflow

nop

beq $s0, -3, handle\_sub\_overflow

nop

beq $s0, -5, handle\_input\_overflow

nop

j reset\_err

nop

handle\_div\_err:

li $s3, 0 # reset second operand

la $a0, div\_err\_msg

li $v0, 4

syscall

li $v0, 1 # Print result

add $a0, $s1, $zero

syscall

j reset\_err

nop

handle\_add\_overflow:

li $s3, 0 # reset second operand

la $a0, add\_overflow\_msg

li $v0, 4

syscall

li $v0, 1 # Print result

add $a0, $s1, $zero

syscall

j reset\_err

nop

handle\_input\_overflow:

li $s3, 0 # reset second operand

la $a0, input\_overflow\_msg

li $v0, 4

syscall

li $v0, 1 # Print result

add $a0, $s1, $zero

syscall

j reset\_err

nop

handle\_sub\_overflow:

li $s3, 0 # reset second operand

la $a0, sub\_overflow\_msg

li $v0, 4

syscall

li $v0, 1 # Print result

add $a0, $s1, $zero

syscall

j reset\_err

nop

handle\_mul\_overflow:

li $s3, 0 # reset second operand

la $a0, mul\_overflow\_msg

li $v0, 4

syscall

li $v0, 1 # Print result

add $a0, $s1, $zero

syscall

j reset\_err

nop

reset\_err:

li $s0, 0 # reset error

end\_handle\_exception:

jr $ra

nop

.ktext 0x80000180

get\_cod:

li $t1, IN\_ADRESS\_HEXA\_KEYBOARD

li $t2, OUT\_ADRESS\_HEXA\_KEYBOARD

li $t3, 0x81 # Check Row 1

sb $t3, 0($t1)

lbu $a0, 0($t2)

bne $a0, $zero, continue

li $t3, 0x82 # Check Row 2

sb $t3, 0($t1)

lbu $a0, 0($t2)

bne $a0, $zero, continue

li $t3, 0x84 # Check Row 3

sb $t3, 0($t1)

lbu $a0, 0($t2)

bne $a0, $zero, continue

li $t3, 0x88 # Check Row 4

sb $t3, 0($t1)

lbu $a0, 0($t2)

bne $a0, $zero, continue

continue:

# Return value to $a0 = KEYPRESSED CODE

map\_key:

li $t7, 0 # index

la $t9, key\_code # load arr

map\_key\_loop:

add $t6, $t9, $t7

lbu $t8, 0($t6) # arr[i]

beq $a0, $t8, map\_key\_return\_value

nop

beq $t7, 15, end\_map\_key

nop

map\_key\_next:

addi $t7, $t7, 1 # index = index + 1

j map\_key\_loop

nop

map\_key\_return\_value:

add $s4, $t7, $zero # set return value

#li $v0, 1

#add $a0, $s4, $zero

#syscall

end\_map\_key:

next\_pc:mfc0 $at, $14

addi $at, $at, 4

mtc0 $at, $14

return: eret