SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

****

COMPUTER ARCHITECTURE LAB

Final Project Report

Instructor: MSc. Le Ba Vui

Name	Student ID	Project title
Nguyen Nam Anh	20215177	Curiosity Marsbot
Pham Tuan Tai	20215240	Simple Calculator

Hanoi, 2024

2. Curiosity Marsbot

a) Problem:

Curiosity Marsbot runs on Mars, remotely controlled by developers from Earth by sending control

messages from the key matrix with the following code:

Control code	Meaning
1b4	Start moving
c68	Stop moving
444	Turn left 90 degrees with the current direction
666	Turn right 90 degrees with the current direction
dad	Start to leave the trace
cbc	Stop to leave the trace
999	Follow the reverse route without leaving a trace and accept the control code until the end of the route.

After receiving the control code, Curiosity Marsbot does not proceed immediately but must wait for the activation command from the Keyboard. There are 3 commands:

Command	Meaning
Enter	Complete receiving the control code, Marsbot takes the action.
Delete	Clear the receiving control code
Space	Repeat the last taken control code.

b) How to use:

- Compile the program

- Open digital lab sim tool, Keyboard and display MMIO Simulator and Marsbot run the program.
- Enter code in digital lab and enter command in keyboard MMIO. The marsbot will perform action base on code and command

c) Method and Algorithm.

To run, the program must enable the interruption of Keyboard matrix 4x4 of Digital Lab.

The program will in infinite loop, waiting for the input key of keyboard MMIO. When a button in Digital Lab is pressed, an interrupt will raise and allow the program to check for key press in Digital Lab.

The program will start with Init function: by rotating to bot to 90 degrees. Allow it to go to the right when the user makes it move.

WaitForKey function: waiting for the input key of keyboard MMIO. When a key is pressed, the program will identify the command and perform execution:

- Enter: The program will execute based on the control_code parameter. If the code is not available, errors will be printed. If not, marsbot will perform action base on control_code. After a successful run, the program will save code into prev_control_code parameter.
- Delete: Run the strClear function to reset control_code parameter.
- Space: Copy prev_control_code value to control_code.

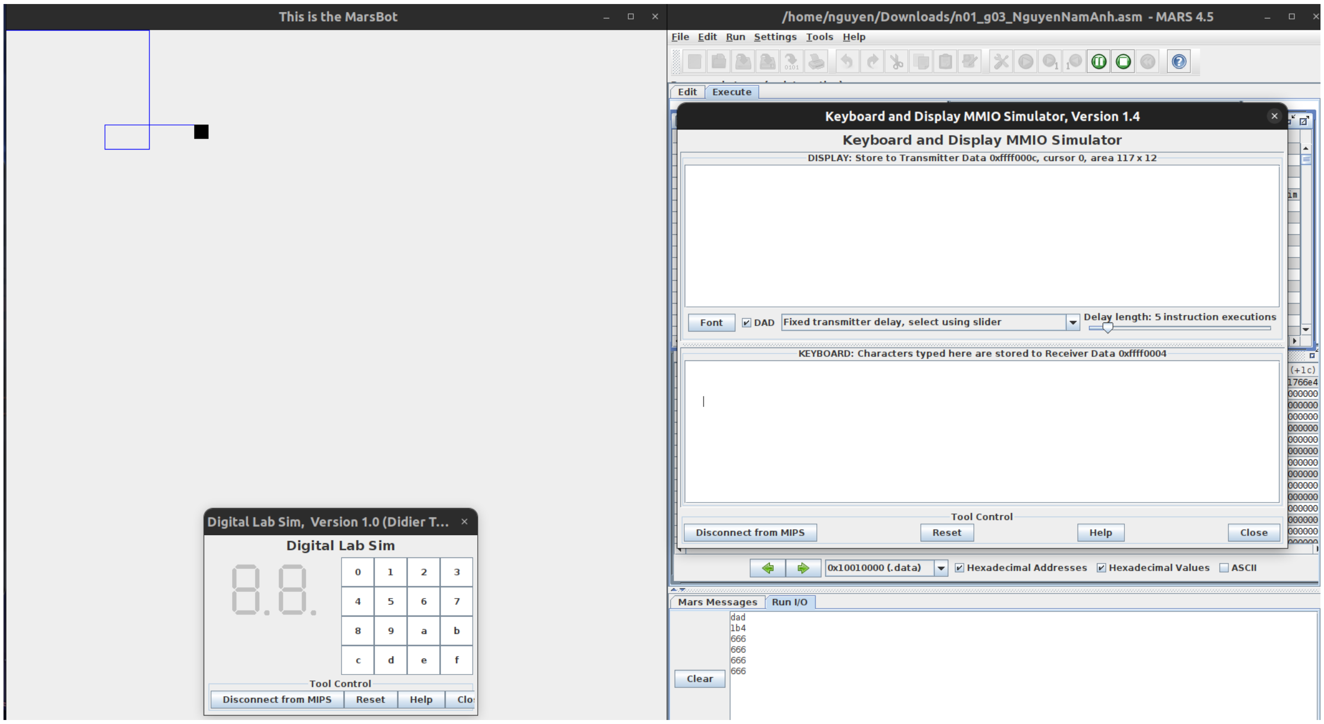
If a key in Digital Lab is pressed, the program will be interrupted and run getCode function to get the input key. The key will be appended to control_code.

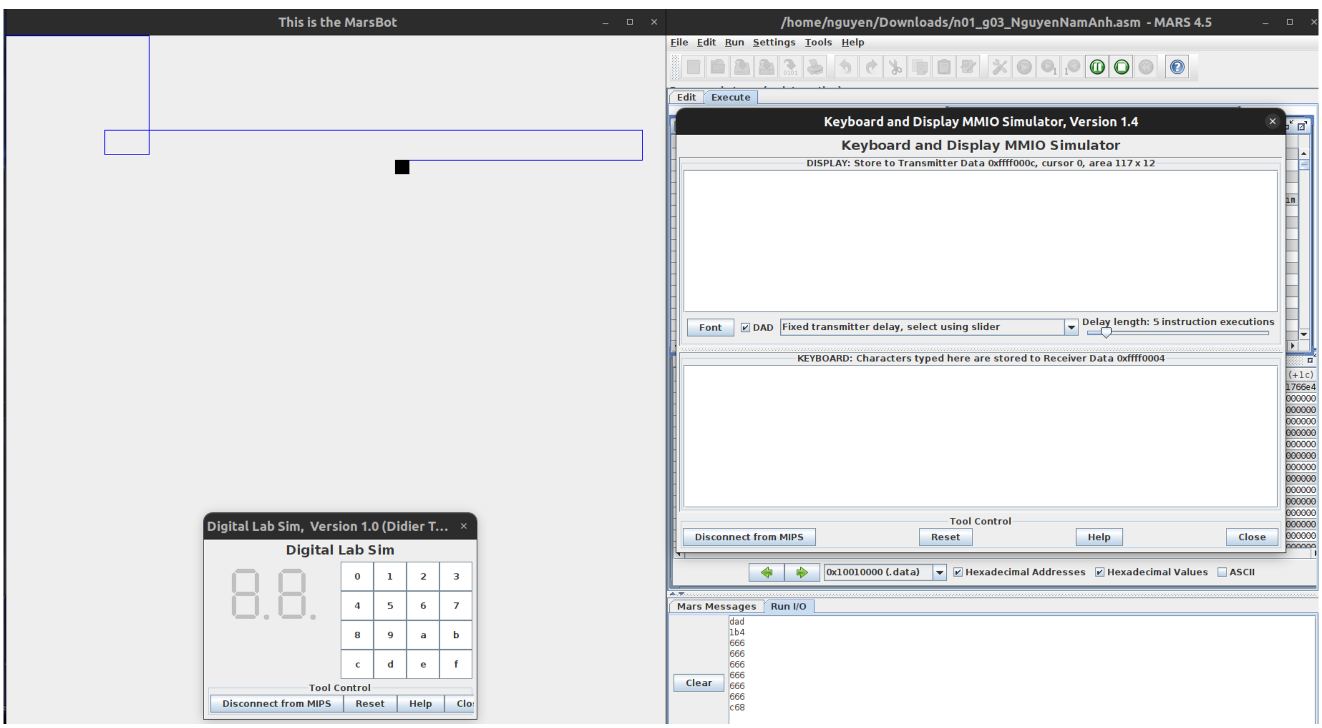
During the runtime, each time a Marsbot performs an action, the coordinate and angle will be saved in an array history. Allow trace back the action when the bot has to follow the reverse trace.

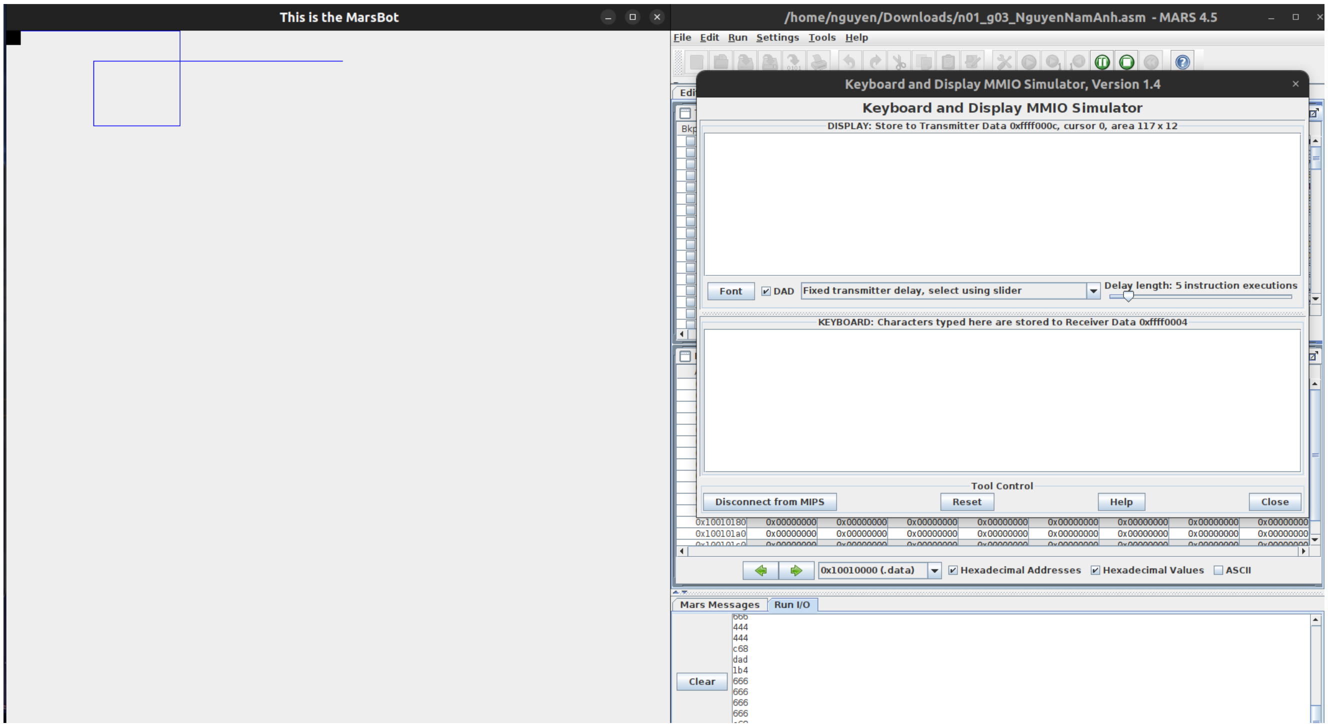
d) Demonstration

- The marbot started moving (‘1b4’) and turn left (‘666’).

- “Space” button is pressed three times, allow it to form a rectangle.







2. Simple Calculator

a) Problem:

Use Key Matrix and 7-segments LEDs to implement a simple calculator that support +, -, *, /, % with

integer operands.

- Press a for addition

- Press b for subtraction

- Press c for multiplication

- Press d for division

- Press e for division with remainder

- Press f to get the result

Detail requirements:

- When pressing digital key, show the last two digits on LEDs. For example, press 1 → show 01, press 2 → show 12, press 3 → show 23.

- After entering an operand, press + - * / % to select the operation.

- After pressing f (=) , calculate and show two digits at the right of the result on LEDs.

- Can calculate continuously (use Calculator on Windows for reference)

b) How to use:

- Compile the program
- Open digital lab sim tool and run the program
- Click the numbers on the keyboard and the corresponding number will appear on the seven segment display
- Choose an operator (details see part a) Problem)
- Choose the number again, if the user doesn't click a number key, then the number will be default to zero

  Click f to compute the expression, the last 2 digit of the result will be display on the seven segment display. The whole equation is printed out in the console.

c) Method and Algorithm.

In order to calculate continuously, the program must run through an infinite loop in which a calculation is done.

Initially, value **0x80** is stored at the **IN_ADDRESS_HEXA_KEYBOARD** to enable interrupt on each key press. When a key is pressed, the assembler will jump to the **.ktext** section, in which calculations is done.

- Scan the keyboard and get the code of the pressed keyboard
- Convert the keyboard code to a real number and display on seven segment code
- Check whether it is a number or an operator
    - If it is a number:

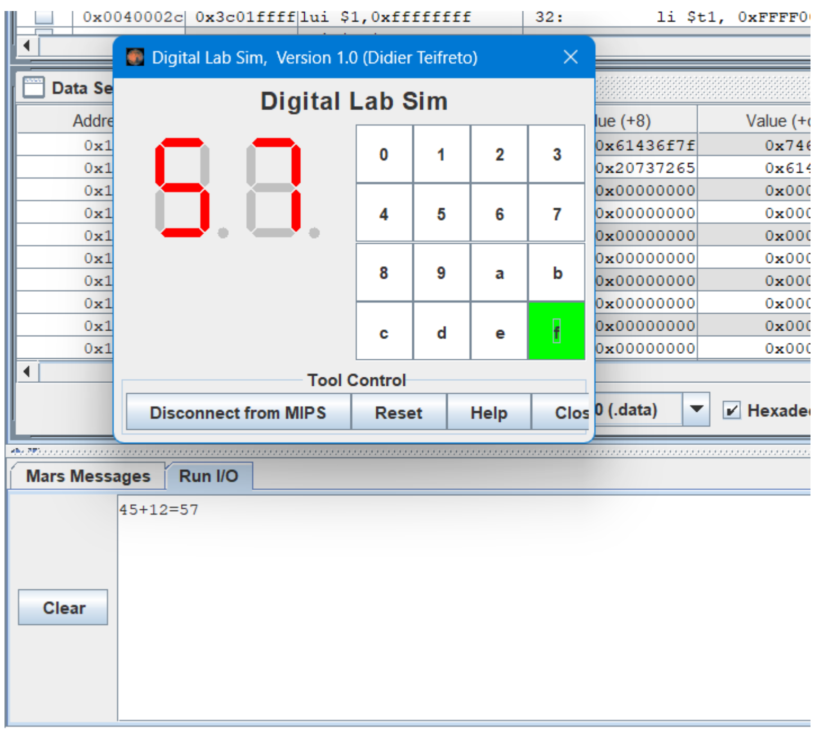
- Push the number to a stack
- Display it on the seven segments display
- Exit the handler

- If it is an operator:
  - Check if the operator is "=" sign, if not, push to operator to the memory, change **$s0** to 1, now the next number will be second operand and then end the exception
  - If the operator is "=", continue

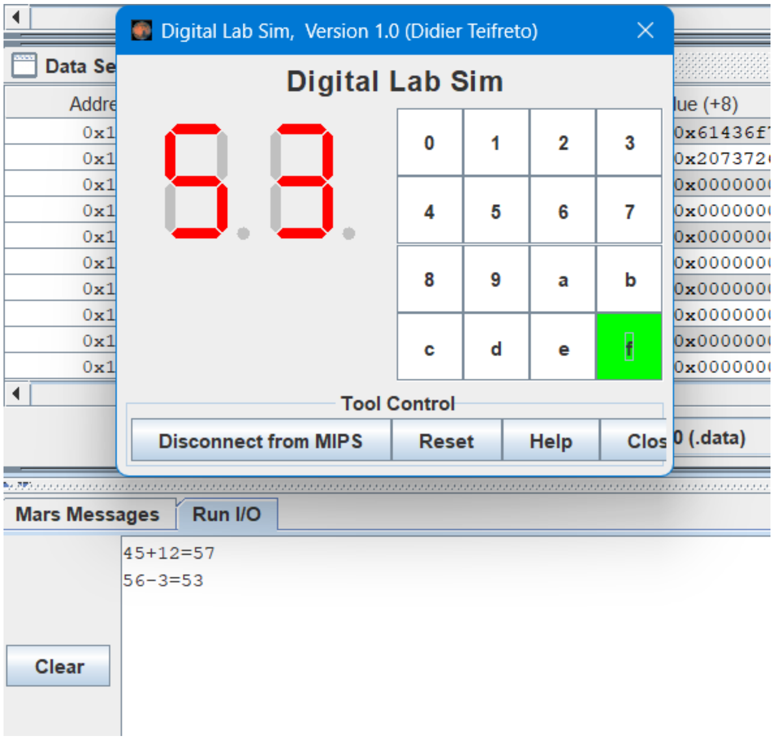
- After pressing the equal sign, the program will display the answer to the seven segments display.

d) Demonstration

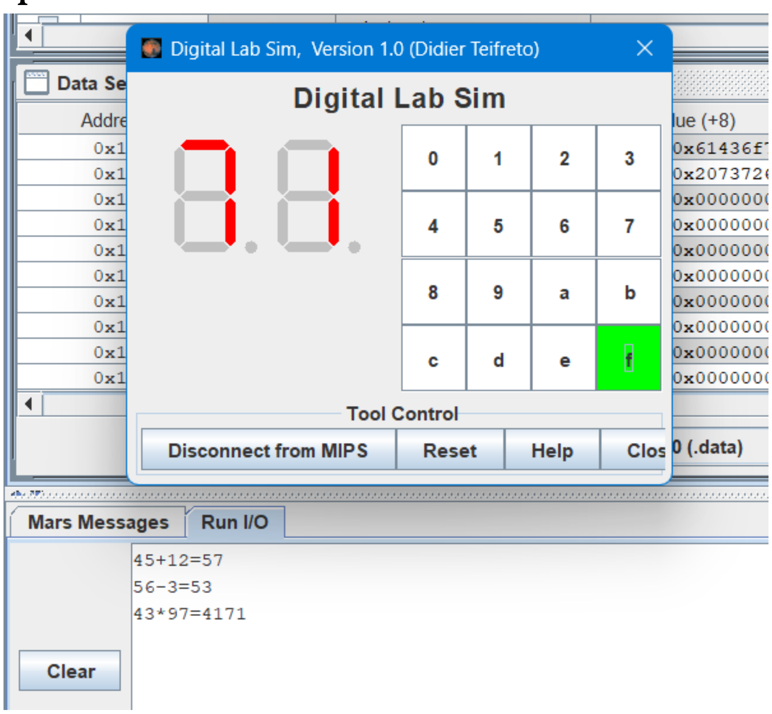
- Addition



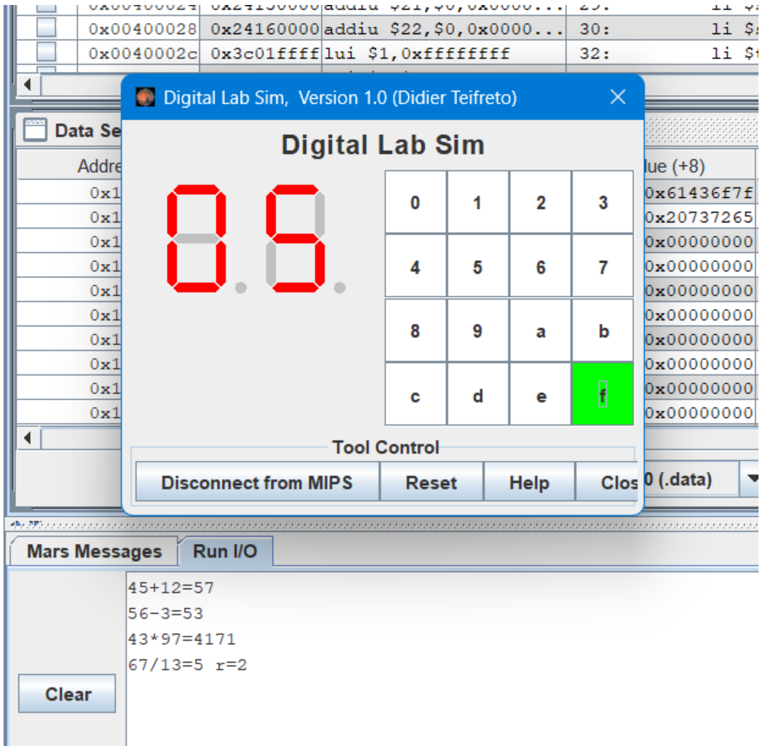
- Subtraction



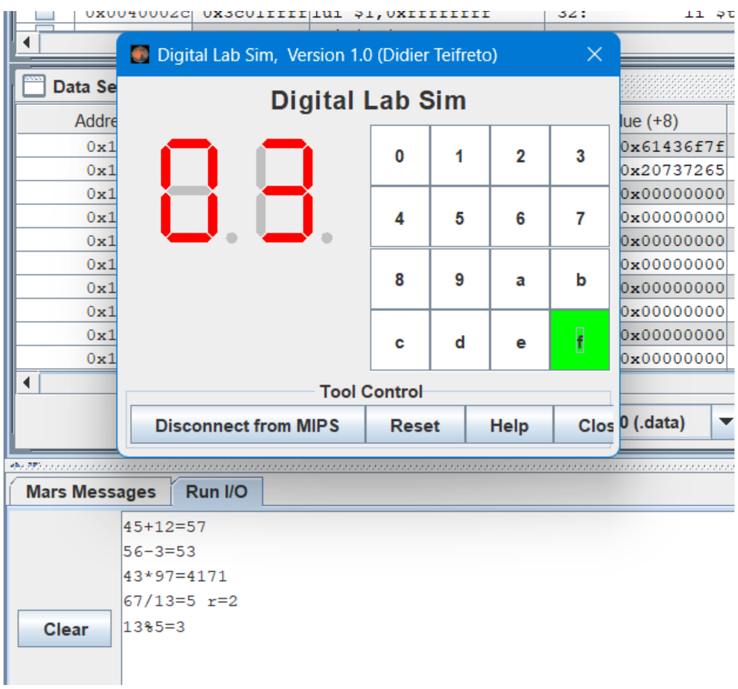
- Multiplication



- Division



- Modulus



3. Source code

SIMPLE CALCULATOR

```
.eqv IN_ADDRESS_HEXA_KEYBOARD    0xFFFF0012

.eqv OUT_ADDRESS_HEXA_KEYBOARD   0xFFFF0014

.eqv SEVENSEG_LEFT 0xFFFF0011          # left LED

.eqv SEVENSEG_RIGHT 0xFFFF0010         # right LED


.data

     zero:  .byte 0x3f

     one:   .byte 0x6

     two:   .byte 0x5b

     three: .byte 0x4f

     four:  .byte 0x66

     five:  .byte 0x6d

     six:   .byte 0x7d

     seven: .byte 0x7

     eight: .byte 0x7f

     nine:  .byte 0x6f

     mess1:  .asciiz "Cannot calculate negative numbers \n"

     mess2: .asciiz "Cannot divide by zero \n"


.text
```

```
main:

      li $t0,SEVENSEG_LEFT        # $t0: value of left LED

        li $t5,SEVENSEG_RIGHT           # $t1: value of right LED

        li $s0,0                        # check input 0: number, 1:
operation, 2: terminate key

        li $s1,0                   # number displayed in left LED

        li $s2,0                   # number displayed in right LED

        li $s3,0                   # representing operation: 1:add,
2:sub, 3:mul, 4:div

        li $s4,0                        # first num

        li $s5,0                   # second num

        li $s6,0                   # result

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

      li $t1, IN_ADDRESS_HEXA_KEYBOARD

      li $t2, OUT_ADDRESS_HEXA_KEYBOARD

      li $t3, 0x80                 #enable keyboard interrupt

      sb  $t3, 0($t1)

      li $t7,0                     #the value of displaying number

      li $t4,0                #byte for displaying on LED (1->9)

storefirstvalue:

      li $t7,0                #first display bit

      addi $sp,$sp,4               #push to stack

        sb $t7,0($sp)

      lb $t4,zero                  #first displaying bit

      addi $sp,$sp,4          #push to stack
```

```
        sb $t4,0($sp)
loop1:
      nop
      nop
      nop
      nop
      b loop1
endloop1:
end_main:
      li $v0,10
      syscall
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# GENERAL INTERRUPT SERVED ROUTINE for all interrupts
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.ktext 0x80000180
process:
      jal checkrow1              #check rows if there is key press
      bnez $t3,convertrow1       #t3 != 0 --> key pressed convert to 
led
      nop
      jal checkrow2
      bnez $t3,convertrow2
      nop
      jal checkrow3
      bnez $t3,convertrow3
```

```
      nop
      jal checkrow4
      bnez $t3,convertrow4
checkrow1:
      addi $sp,$sp,4
         sw $ra,0($sp)
         li $t3,0x81         # enable interrupt
         sb $t3,0($t1)
         jal getvalue        # get the position ( col and row ) if
pressed
         lw $ra,0($sp)
         addi $sp,$sp,-4
         jr $ra
checkrow2:
      addi $sp,$sp,4
         sw $ra,0($sp)
      li $t3,0x82      # enable interrupt for row 2
         sb $t3,0($t1)
         jal getvalue
         lw $ra,0($sp)
         addi $sp,$sp,-4
         jr $ra
checkrow3:
      addi $sp,$sp,4
         sw $ra,0($sp)
```

```
      li $t3,0x84     # enable interrupt for row 3
        sb $t3,0($t1)
        jal getvalue
        lw $ra,0($sp)
        addi $sp,$sp,-4
        jr $ra
checkrow4:
      addi $sp,$sp,4
        sw $ra,0($sp)
      li $t3,0x88     # enable interrupt for row 4
        sb $t3,0($t1)
        jal getvalue
        lw $ra,0($sp)
        addi $sp,$sp,-4
        jr $ra
getvalue:
      addi $sp,$sp,4
        sw $ra,0($sp)
        li $t2,OUT_ADDRESS_HEXA_KEYBOARD  #adress contains position of
the key pressed
        lb $t3,0($t2)             #load
        lw $ra,0($sp)
        addi $sp,$sp,-4
        jr $ra
convertrow1:               #convert from position to number
```

```
	beq $t3,0x11,case_zero			#0x11 -->row 1 col 1--> 0
	beq $t3,0x21,case_one
	beq $t3,0x41,case_two
	beq $t3,0xffffff81,case_three
case_zero:
	lb $t4,zero		#convert
	li $t7,0		#t7= t4
	j done
case_one:
	lb $t4,one
	li $t7,1
	j done
case_two:
	lb $t4,two
	li $t7,2
	j done
case_three:
	lb $t4,three
	li $t7,3
	j done
convertrow2:
	beq $t3,0x12,case_four
	beq $t3,0x22,case_five
	beq $t3,0x42,case_six
	beq $t3,0xffffff82,case_seven
```

```
case_four:
	lb $t4,four
	li $t7,4
	j done
case_five:
	lb $t4,five
	li $t7,5
	j done
case_six:
	lb $t4,six
	li $t7,6
	j done
case_seven:
	lb $t4,seven
	li $t7,7
	j done
convertrow3:
	beq $t3,0x14,case_eight
	beq $t3,0x24,case_nine
	beq $t3 0x44,case_a
	beq $t3 0xffffff84,case_b
case_eight:
	lb $t4,eight
	li $t7,8
	j done
```

```
case_nine:

      lb $t4,nine

      li $t7,9

      j done

case_a:    #addition

      addi $a3,$zero,1

      addi $s0,$s0,1          #check variable turns to 1 (operator)

      bne $s3,0,setnextoperator

      addi $s3,$zero,1 #operator type =  1(addition)


      j setfirstnumber        #convert 2 byte that are being displayed
on 2 led to number to calculate

case_b: #subtraction

      addi $a3,$zero,2

      addi $s0,$s0,1

      bne $s3,0,setnextoperator

      addi $s3,$zero,2

      j setfirstnumber

convertrow4:

      beq $t3,0x18,case_c

      beq $t3,0x28,case_d

      beq $t3,0x48,case_e

      beq $t3 0xffffff88,case_f

case_c: #multiplication

      addi $a3,$zero,3
```

```
	addi $s0,$s0,1
	bne $s3,0,setnextoperator
	addi $s3,$zero,3
	j setfirstnumber
case_d: #division
	addi $a3,$zero,4
	addi $s0,$s0,1
	bne $s3,0,setnextoperator
	addi $s3,$zero,4
	j setfirstnumber

case_e: #modular
	addi $a3, $zero, 5
	addi $s0, $s0, 1
	bne $s3, 0, setnextoperator
	addi $s3, $zero, 5
	j setfirstnumber



setfirstnumber:			# calculate the displaying value
	mul $s4,$s2,10		# s4=s2*10+s1
	add $s4,$s4,$s1
	j done

case_f:  #press =
```

```
setsecondnumber:  #calculate second number that displaying
      mul $s5,$s2,10         # s5=s2*10+s1
      add $s5,$s5,$s1
      beq $s3,1,addition         # s3=1--> addition
      beq $s3,2,subtraction
      beq $s3,3,multiplication
      beq $s3,4,division
      beq $s3, 5, modular
addition:
      add $s6,$s5,$s4
      li $s3,0
      j printadd
      nop             # s6=s5+s4

printadd:
      li $v0, 1
      move $a0, $s4
      syscall

      li $v0, 11
      li $a0, '+'
      syscall

      li $v0, 1
      move $a0, $s5
```

```
	syscall

	li $v0, 11
	li $a0, '='
	syscall

	li $v0, 1
	move $a0, $s6
	syscall

	li $v0, 11
	li $a0, '\n'
	syscall
	li $s7,100
	div $s6,$s7
	mfhi $s6	# only takes 2 last digit of result to led
	j splitnumber	# split to display on LED
	nop

subtraction:
	sub $s6,$s4,$s5	# s6=s4-s5
	li $s3,0
	blt $s6,0,subneg
	j printsub
	nop
```

```
printsub:
	li $v0, 1
	move $a0, $s4
	syscall

	li $v0, 11
	li $a0, '-'
	syscall

	li $v0, 1
	move $a0, $s5
	syscall

	li $v0, 11
	li $a0, '='
	syscall

	li $v0, 1
	move $a0, $s6
	syscall

	li $v0, 11
	li $a0, '\n'
	syscall
	j splitnumber
```

```
	nop
multiplication:
	mul $s6,$s4,$s5     # s6=s4*s5
	li $s3,0
	j printmul
	nop
printmul:
	li $v0, 1
	move $a0, $s4
	syscall

	li $v0, 11
	li $a0, '*'
	syscall

	li $v0, 1
	move $a0, $s5
	syscall

	li $v0, 11
	li $a0, '='
	syscall

	li $v0, 1
	move $a0, $s6
```

```
	syscall

	li $v0, 11
	li $a0, '\n'
	syscall
	li $s7,100
	div $s6,$s7
	mfhi $s6	# chi lay 2 chu so sau cùng cua ket qua in ra
	j splitnumber	# chuyen den ham chia ket qua thanh 2 chu so
de hien thi len tung led
	nop
division:
	beq $s5,0,div0
	li $s3,0
	div $s4,$s5		# s6=s4/s5
	mflo $s6
	mfhi $s7
	j printdiv
	nop
printdiv:
	li $v0, 1
	move $a0, $s4
	syscall

	li $v0, 11
```

```
li $a0, '/'
syscall

li $v0, 1
move $a0, $s5
syscall

li $v0, 11
li $a0, '='
syscall

li $v0, 1
move $a0, $s6
syscall

li $v0, 11
li $a0, ' '
syscall

li $v0, 11
li $a0, 'r'
syscall

li $v0, 11
li $a0, '='
```

```
	syscall

	li $v0, 1
	move $a0, $s7
	syscall

	li $v0, 11
	li $a0, '\n'
	syscall
	j splitnumber
	nop
modular:
	beq $s5,0,div0
	li $s3,0
	div $s4,$s5		# s6=s4/s5
	mfhi $s6
	j printmod
	nop
printmod:
	li $v0, 1
	move $a0, $s4
	syscall

	li $v0, 11
	li $a0, '%'
```

```
        syscall

        li $v0, 1
        move $a0, $s5
        syscall

        li $v0, 11
        li $a0, '='
        syscall

        li $v0, 1
        move $a0, $s6
        syscall

        li $v0, 11
        li $a0, ' '
        syscall

        li $v0, 11
        li $a0, '\n'
        syscall
        j splitnumber
        nop
div0:
        li $v0, 55
```

```
	la $a0, mess2
	li $a1, 0
	syscall
	j resetled
subneg:
	li $v0, 55
	la $a0, mess1
	li $a1, 0
	syscall
	j resetled

splitnumber:    #split the last 2 digits to display on each LED
	li $t8,10
	div $s6,$t8    #s6/10
	mflo $t7       #t7 = result
	jal convert    #convert number to LED
	  #---------
	  sb $t4,0($t0)  # left LED
	add $sp,$sp,4
	sb $t7,0($sp)       #push to stack
	add $sp,$sp,4
	sb $t4,0($sp)      #push to stack
	add $s2,$t7,$zero   #s1 = value of left LED

	#----------
```

```
      mfhi $t7       #t7= remainder
      jal convert
         sb $t4,0($t5)  #right LED
            add $sp,$sp,4
      sb $t7,0($sp)       #push to stack
      add $sp,$sp,4
      sb $t4,0($sp)       #push to stack
      add $s1,$t7,$zero   #s1 = value of left LED
         j resetled
convert:
      addi $sp,$sp,4
         sw $ra,0($sp)
         beq $t7,0,case_0
         beq $t7,1,case_1
         beq $t7,2,case_2
         beq $t7,3,case_3
         beq $t7,4,case_4
         beq $t7,5,case_5
         beq $t7,6,case_6
         beq $t7,7,case_7
         beq $t7,8,case_8
         beq $t7,9,case_9
case_0:
      lb $t4,zero    #t4=zero
      j finishconvert
```

```
case_1:
	lb $t4,one
	j finishconvert
case_2:
	lb $t4,two
	j finishconvert
case_3:
	lb $t4,three
	j finishconvert
case_4:
	lb $t4,four
	j finishconvert
case_5:
	lb $t4,five
	j finishconvert
case_6:
	lb $t4,six
	j finishconvert
case_7:
	lb $t4,seven
	j finishconvert
case_8:
	lb $t4,eight
	j finishconvert
case_9:
```

```
	lb $t4,nine
	j finishconvert
finishconvert:
	lw $ra,0($sp)
	addi $sp,$sp,-4
	jr $ra
done:
	beq $s0,1,resetled   #s0=1-->operator-->reset led
loadtoleftled:   # display left LED
	lb $t6,0($sp)       #load from stack
	add $sp,$sp,-4
	lb $t8,0($sp)
	add $sp,$sp,-4
	add $s2,$t8,$zero   #s2 = value of left LED
	sb $t6,0($t0)       # display
loadtorightled:
	sb $t4,0($t5)
	add $sp,$sp,4
	sb $t7,0($sp)
	add $sp,$sp,4
	sb $t4,0($sp)
	add $s1,$t7,$zero   #s1 = value of right LED
	j finish
resetled:
	li $s0,0           #s0=0--> wait for next number
```

```
   li $t8,0
  addi $sp,$sp,4
   sb $t8,0($sp)
   lb $t6,zero        # push zero
  addi $sp,$sp,4
   sb $t6,0($sp)
finish:
  j end_exception
  nop
end_exception:
  # return to start of the loop instead of where the interrupt
occur, since the loop doesn't do meaningful thing
  la $a3, loop1
  mtc0 $a3, $14
  eret
setnextoperator:
setsecondnumber1:  #find second number
  mul $s5,$s2,10         # s5=s2*10+s1
  add $s5,$s5,$s1
  beq $s3,1,add1         # s3=1--> addition
  beq $s3,2,sub1
  beq $s3,3,mul1
  beq $s3,4,div1
  beq $s3,5,mod1
add1:
```

```
      add $s6,$s5,$s4
      j printadd1
      nop               # s6=s5+s4

printadd1:
      li $v0, 1
      move $a0, $s4
      syscall

      li $v0, 11
      li $a0, '+'
      syscall

      li $v0, 1
      move $a0, $s5
      syscall

      li $v0, 11
      li $a0, '='
      syscall

      li $v0, 1
      move $a0, $s6
      syscall
```

```
	li $v0, 11
	li $a0, '\n'
	syscall
	li $s7,100
	div $s6,$s7
	mfhi $s6	# chi lay 2 chu so cuoi cua ket qua de in ra led
	j splitnumber1	# chuyen den ham chia ket qua thanh 2 chu so
de hien thi len tung led
	nop

sub1:
	sub $s6,$s4,$s5	# s6=s4-s5
	blt $s6,0,subneg1
	j printsub1
	nop
printsub1:
	li $v0, 1
	move $a0, $s4
	syscall

	li $v0, 11
	li $a0, '-'
	syscall

	li $v0, 1
```

```
	move $a0, $s5
	syscall

	li $v0, 11
	li $a0, '='
	syscall

	li $v0, 1
	move $a0, $s6
	syscall

	li $v0, 11
	li $a0, '\n'
	syscall
	j splitnumber1       # chuyen den ham chia ket qua thanh 2 chu so de hien thi len tung led
	nop
mul1:
	mul $s6,$s4,$s5     # s6=s4*s5
	j printmul1
	nop
printmul1:
	li $v0, 1
	move $a0, $s4
	syscall
```

```
	li $v0, 11
	li $a0, '*'
	syscall

	li $v0, 1
	move $a0, $s5
	syscall

	li $v0, 11
	li $a0, '='
	syscall

	li $v0, 1
	move $a0, $s6
	syscall

	li $v0, 11
	li $a0, '\n'
	syscall
	li $s7,100
	div $s6,$s7
	mfhi $s6	# chi lay 2 chu so sau cùng cua ket qua in ra
	j splitnumber1	# chuyen den ham chia ket qua thanh 2 chu so de hien thi len tung led
```

```
	nop
div1:
	beq $s5,0,div01
	div $s4,$s5		# s6=s4/s5
	mflo $s6
	mfhi $s7
	j printdiv1
	nop
printdiv1:
	li $v0, 1
	move $a0, $s4
	syscall

	li $v0, 11
	li $a0, '/'
	syscall

	li $v0, 1
	move $a0, $s5
	syscall

	li $v0, 11
	li $a0, '='
	syscall
```

```
	li $v0, 1
	move $a0, $s6
	syscall

	li $v0, 11
	li $a0, ' '
	syscall

	li $v0, 11
	li $a0, 'r'
	syscall

	li $v0, 11
	li $a0, '='
	syscall

	li $v0, 1
	move $a0, $s7
	syscall

	li $v0, 11
	li $a0, '\n'
	syscall
	j splitnumber1		# chuyen den ham chia ket qua thanh 2 chu so de hien thi len tung led
```

```
	nop
mod1:
	beq $s5,0,div01
	div $s4,$s5		# s6=s4/s5
	mfhi $s6
	j printmod1
	nop
printmod1:
	li $v0, 1
	move $a0, $s4
	syscall

	li $v0, 11
	li $a0, '%'
	syscall

	li $v0, 1
	move $a0, $s5
	syscall

	li $v0, 11
	li $a0, '='
	syscall

	li $v0, 1
```

```
	move $a0, $s6
	syscall
div01:
	li $v0, 55
	la $a0, mess2
	li $a1, 0
	syscall
	j resetled1
subneg1:
	li $v0, 55
	la $a0, mess1
	li $a1, 0
	syscall
	j resetled1
splitnumber1:   #divide the result into 2 digit to display
	li $t8,10
	div $s6,$t8    #s6/10
	mflo $t7       #t7 = result
	jal convert1
	  #---------
	add $sp,$sp,4
	sb $t7,0($sp)        #push to stack
	add $sp,$sp,4
	sb $t4,0($sp)       #push to stack
	add $s2,$t7,$zero
```

```
	#----------
	mfhi $t7
	jal convert1
	     add $sp,$sp,4
	sb $t7,0($sp)
	add $sp,$sp,4
	sb $t4,0($sp)
	add $s1,$t7,$zero
	  j resetled1     #ham reset lai led
convert1:
	addi $sp,$sp,4
	  sw $ra,0($sp)
	  beq $t7,0,case_01    #t7=0 -->ham chuyen 0 thanh bit zero hien
thi len led
	  beq $t7,1,case_11
	  beq $t7,2,case_21
	  beq $t7,3,case_31
	  beq $t7,4,case_41
	  beq $t7,5,case_51
	  beq $t7,6,case_61
	  beq $t7,7,case_71
	  beq $t7,8,case_81
	  beq $t7,9,case_91
case_01:   #ham chuyen 0 thanh bit zero hien thi len led
```

```
	lb $t4,zero	#t4=zero
	j finishconvert1 #ket thuc
case_11:
	lb $t4,one
	j finishconvert1
case_21:
	lb $t4,two
	j finishconvert1
case_31:
	lb $t4,three
	j finishconvert1
case_41:
	lb $t4,four
	j finishconvert1
case_51:
	lb $t4,five
	j finishconvert1
case_61:
	lb $t4,six
	j finishconvert1
case_71:
	lb $t4,seven
	j finishconvert1
case_81:
	lb $t4,eight
```

```
	j finishconvert1
case_91:
	lb $t4,nine
	j finishconvert1
finishconvert1:
	lw $ra,0($sp)
	addi $sp,$sp,-4
	jr $ra
done1:
	beq $s0,1,resetled1
resetled1:
	li $s0,0
	  li $t8,0
	addi $sp,$sp,4
	  sb $t8,0($sp)
	  lb $t6,zero
	addi $sp,$sp,4
	  sb $t6,0($sp)
	  mul $s4,$s2,10          # s4=s2*10+s1
	add $s4,$s4,$s1
	beq $a3,1,setadd
	nop
	beq $a3,2,setsub
	nop
	beq $a3,3,setmul
```

```
      nop
      beq $a3,4,setdiv
      nop
      beq $a3,5, setmod
      nop
setadd: addi $s3,$zero,1
      j finish1
      nop
setsub: addi $s3,$zero,2
      j finish1
      nop
setmul: addi $s3,$zero,3
      j finish1
      nop
setdiv: addi $s3,$zero,4
      j finish1
      nop
setmod: addi $s3, $zero, 5
      j finish1
      nop

finish1:
      j end_exception1
      nop
end_exception1:
```

```
	la $a3, loop1
	mtc0 $a3, $14
	eret
```

Curiosity Marsbot

```
# eqv for Digital Lab Sim
.eqv KEY_0 0x11
.eqv KEY_1 0x21
.eqv KEY_2 0x41
.eqv KEY_3 0x81
.eqv KEY_4 0x12
.eqv KEY_5 0x22
.eqv KEY_6 0x42
.eqv KEY_7 0x82
.eqv KEY_8 0x14
.eqv KEY_9 0x24
.eqv KEY_a 0x44
.eqv KEY_b 0x84
.eqv KEY_c 0x18
```

```
.eqv KEY_d 0x28

.eqv KEY_e 0x48

.eqv KEY_f 0x88


# eqv for Keyboard

.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012

.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014

.eqv KEY_CODE 0xFFFF0004   # ASCII code from keyboard, 1 byte

.eqv KEY_READY 0xFFFF0000  # = 1 if has a new keycode ?

                           # Auto clear after lw


# eqv for Mars bot

.eqv HEADING 0xffff8010          # Integer: An angle between 0 and 359

                # 0 : North (up)

                                 # 90: East (right)

                                 # 180: South (down)

                                 # 270: West (left)

.eqv MOVING 0xffff8050           # Boolean: whether or not to move

.eqv LEAVETRACK 0xffff8020 # Boolean: whether or not to leave a track

.eqv WHEREX 0xffff8030           # Integer: Current x-location of
MarsBot

.eqv WHEREY 0xffff8040           # Integer: Current y-location of
MarsBot


#----------------------------------------------------------------------
----------
```

```
.data
# CODE
MOVE_CODE: .asciiz "1b4" # command code
STOP_CODE: .asciiz "c68"
TURN_LEFT_CODE: .asciiz "444"
TURN_RIGHT_CODE: .asciiz "666"
TRACK_CODE: .asciiz "dad"
UNTRACK_CODE: .asciiz "cbc"
GOBACKWARD_CODE: .asciiz "999"

error_msg:  .asciiz "Invalid command code: "


# HISTORY
# save history before changing direction

x_history: .word 0 : 16 # = 16 for easier debugging
y_history: .word 0 : 16

# For rotation
a_history: .word 0 : 16
l_history: .word 4          # history length

a_current: .word 0          # current alpha
```

```
is_going: .word 0

is_tracking: .word 0


# Code properties

control_code: .space 8     # input command code

code_length: .word 0            # input command length


prev_control_code: .space 8     # store previous input code


.text


main:

li $k0, KEY_CODE

li $k1, KEY_READY


li $t1, IN_ADRESS_HEXA_KEYBOARD # enable the interrupt of Digital Lab
Sim

li $t3, 0x80   # bit 7 = 1 to enable

sb $t3, 0($t1)


# run at start of program

init:

# increase length history by 4

# (as saving current state: x = 0; y = 0; a = 90)
```

```
lw $t7, l_history # l_history += 4
addi $t7, $zero, 4
sw $t7, l_history

li $t7, 90
sw $t7, a_current # a_current = 90 -> head to the right
jal ROTATE
nop

sw $t7, a_history # a_history[0] = 90

j waitForKey

# Function: print error to console
printError:
li $v0, 4
la $a0, error_msg
syscall

printCode:
li $v0, 4
la $a0, control_code
syscall
j resetInput
```

```
repeatCode:
# copy from the prev_control_code
jal strCpy1
j checkCode

resetInput:
jal strClear
nop

# Take input
waitForKey:
lw $t5, 0($k1)   # $t5 = [$k1] = KEY_READY
beq $t5, $zero, waitForKey  # if $t5 == 0 -> Polling
nop
beq $t5, $zero, waitForKey

readKey:
lw $t6, 0($k0)   # $t6 = [$k0] = KEY_CODE
# if $t6 == 'DEL' -> reset input
beq $t6, 0x8, resetInput

# if $t6 == 'SPACE' -> reset copy from previous input and
# go to checkCode label
beq $t6, 0x20, repeatCode
```

```
# if $t6 != 'ENTER' -> Polling
bne $t6, 0x0a, waitForKey
nop
bne $t6, 0x0a, waitForKey

checkCode:
lw $s2, code_length   # code_length != 3 -> invalid code
bne $s2, 3, printError

la $s3, MOVE_CODE
jal strcmp
beq $t0, 1, go

la $s3, STOP_CODE
jal strcmp
beq $t0, 1, stop

la $s3, TURN_LEFT_CODE
jal strcmp
beq $t0, 1, turnLeft

la $s3, TURN_RIGHT_CODE
jal strcmp
beq $t0, 1, turnRight
```

```
la $s3, TRACK_CODE
jal strcmp
beq $t0, 1, track

la $s3, UNTRACK_CODE
jal strcmp
beq $t0, 1, untrack

la $s3, GOBACKWARD_CODE
jal strcmp
beq $t0, 1, goBackward
nop

j printError



# Perform function and print code
go:
jal strCpy2
jal GO
j printCode

stop:
jal strCpy2
jal STOP
```

```
j printCode

track:
jal strCpy2
jal TRACK
j printCode

untrack:
jal strCpy2
jal UNTRACK
j printCode


turnRight:
jal strCpy2
lw $t7, is_going
lw $s0, is_tracking

jal STOP
nop
jal UNTRACK
nop

la $s5, a_current
lw $s6, 0($s5)  # $s6 is heading at now
```

```
addi $s6, $s6, 90 # increase alpha by 90*
sw $s6, 0($s5)  # update a_current

jal saveHistory
jal ROTATE

beqz $s0, noTrack1
nop
jal TRACK
noTrack1: nop

beqz $t7, noGo1
nop
jal GO
noGo1:
nop
j printCode


turnLeft:
jal strCpy2
lw $t7, is_going
lw $s0, is_tracking

jal STOP
```

```
nop
jal UNTRACK
nop

la $s5, a_current
lw $s6, 0($s5)  # $s6 is heading at now
addi $s6, $s6, -90 # decrease alpha by 90*
sw $s6, 0($s5)  # update a_current

jal saveHistory
jal ROTATE

beqz $s0, noTrack2
nop
jal TRACK
noTrack2: nop

beqz $t7, noGo2
nop
jal GO
noGo2:
nop
j printCode
```

```
goBackward:
jal strCpy2
li $t7, IN_ADRESS_HEXA_KEYBOARD # Disable interrupts when going backward
      sb $zero, 0($t7)

lw $s5, l_history  # $s5 = code_length
jal UNTRACK
jal GO

goBackward_turn:
addi $s5, $s5, -4   # code_length--
lw $s6, a_history($s5)  # $s6 = a_history[code_length]
addi $s6, $s6, 180  # $s6 = the reverse direction of alpha
sw $s6, a_current
jal ROTATE
nop

goBackward_toTurningPoint:
lw $t9, x_history($s5)  # $t9 = x_history[i]
lw $t7, y_history($s5)  # $t9 = y_history[i]

get_x:
li $t8, WHEREX   # $t8 = x_current
lw $t8, 0($t8)
```

```
bne $t8, $t9, get_x  # x_current == x_history[i]
nop
bne $t8, $t9, get_x

get_Y:
li $t8, WHEREY   # $t8 = y_current
lw $t8, 0($t8)

bne $t8, $t7, get_Y  # y_current == y_history[i]
nop
bne $t8, $t7, get_Y  # y_current == y_history[i]

beq $s5, 0, goBackward_end  # l_history == 0
nop    # -> end

j goBackward_turn   # else -> turn

goBackward_end:
jal STOP
sw $zero, a_current  # update heading
jal ROTATE

addi $s5, $zero, 4
sw $s5, l_history  # reset l_history = 0
```

```
j printCode

#------------------------------------------------------------
# saveHistory()
#------------------------------------------------------------

saveHistory:
addi $sp, $sp, 4   # backup
sw $t1, 0($sp)
addi $sp, $sp, 4
sw $t2, 0($sp)
addi $sp, $sp, 4
sw $t3, 0($sp)
addi $sp, $sp, 4
sw $t4, 0($sp)
addi $sp, $sp, 4
sw $s1, 0($sp)
addi $sp, $sp, 4
sw $s2, 0($sp)
addi $sp, $sp, 4
sw $s3, 0($sp)
addi $sp, $sp, 4
sw $s4, 0($sp)
```

```
lw $s1, WHEREX   # s1 = x

lw $s2, WHEREY   # s2 = y

lw $s4, a_current  # s4 = a_current


lw $t3, l_history  # $t3 = l_history

sw $s1, x_history($t3)  # store: x, y, alpha

sw $s2, y_history($t3)

sw $s4, a_history($t3)


addi $t3, $t3, 4   # update lengthPath

sw $t3, l_history


lw $s4, 0($sp)   # restore backup

addi $sp, $sp, -4

lw $s3, 0($sp)

addi $sp, $sp, -4

lw $s2, 0($sp)

addi $sp, $sp, -4

lw $s1, 0($sp)

addi $sp, $sp, -4

lw $t4, 0($sp)

addi $sp, $sp, -4

lw $t3, 0($sp)

addi $sp, $sp, -4

lw $t2, 0($sp)
```

```
addi $sp, $sp, -4
lw $t1, 0($sp)
addi $sp, $sp, -4

saveHistory_end:
jr $ra

#=====================================================================
==========
# Procedure for Mars bot
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# GO()
#-----------------------------------------------------------
GO:
addi $sp, $sp, 4   # backup
sw $at, 0($sp)
addi $sp, $sp, 4
sw $k0, 0($sp)

li $at, MOVING   # change MOVING port
addi $k0, $zero, 1  # to logic 1,
sb $k0, 0($at)   # to start running

li $t7, 1   # is_going = 0
sw $t7, is_going
```

```
lw $k0, 0($sp)   # restore back up
addi $sp, $sp, -4
lw $at, 0($sp)
addi $sp, $sp, -4

GO_end:
jr $ra

#-----------------------------------------------------------
# STOP()
#-----------------------------------------------------------
STOP:
addi $sp, $sp, 4   # backup
sw $at, 0($sp)

li $at, MOVING   # change MOVING port to 0
sb $zero, 0($at)  # to stop

sw $zero, is_going  # is_going = 0

lw $at, 0($sp)   # restore back up
addi $sp, $sp, -4

STOP_end:
```

```
jr $ra

#------------------------------------------------------------
# TRACK()
#------------------------------------------------------------
TRACK:
addi $sp, $sp, 4   # backup
sw $at, 0($sp)
addi $sp, $sp, 4
sw $k0, 0($sp)

li $at, LEAVETRACK  # change LEAVETRACK port
addi $k0, $zero,1  # to logic 1,
sb $k0, 0($at)   # to start tracking

addi $s0, $zero, 1
sw $s0, is_tracking

lw $k0, 0($sp)   # restore back up
addi $sp, $sp, -4
lw $at, 0($sp)
addi $sp, $sp, -4

TRACK_end:
jr $ra
```

```
#------------------------------------------------------------
# UNTRACK()
#------------------------------------------------------------
UNTRACK:
addi $sp, $sp, 4  # backup
sw $at, 0($sp)

li $at, LEAVETRACK # change LEAVETRACK port to 0
sb $zero, 0($at) # to stop drawing tail

sw $zero, is_tracking

lw $at, 0($sp)  # restore back up
addi $sp, $sp, -4

UNTRACK_end:
jr $ra

#------------------------------------------------------------
# ROTATE()
#------------------------------------------------------------
ROTATE:
addi $sp, $sp, 4  # backup
sw $t1, 0($sp)
```

```
addi $sp, $sp, 4

sw $t2, 0($sp)

addi $sp, $sp, 4

sw $t3, 0($sp)


li $t1, HEADING # change HEADING port

la $t2, a_current

lw $t3, 0($t2)  # $t3 is heading at now

sw $t3, 0($t1)  # to rotate robot


lw $t3, 0($sp)  # restore back up

addi $sp, $sp, -4

lw $t2, 0($sp)

addi $sp, $sp, -4

lw $t1, 0($sp)

addi $sp, $sp, -4


ROTATE_end:

jr $ra


#======================================================================
==========

# Procedure for string

#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

# strcmp()
```

```
# - input: $s3 = string to compare with control_code
# - output: $t0 = 0 if not equal, 1 if equal
#-----------------------------------------------------------
strcmp:
addi $sp, $sp, 4   # back up
sw $t1, 0($sp)
addi $sp, $sp, 4
sw $s1, 0($sp)
addi $sp,$sp,4
sw $t2, 0($sp)
addi $sp, $sp, 4
sw $t3, 0($sp)

xor $t0, $zero, $zero  # $t1 = return value = 0
xor $t1, $zero, $zero  # $t1 = i = 0

strcmp_loop:
beq $t1, 3, strcmp_equal  # if i = 3 -> end loop -> equal
nop

lb $t2, control_code($t1)  # $t2 = control_code[i]

add $t3, $s3, $t1  # $t3 = s + i
lb $t3, 0($t3)   # $t3 = s[i]
```

```
beq $t2, $t3, strcmp_next  # if $t2 == $t3 -> continue the loop
nop

j strcmp_end

strcmp_next:
addi $t1, $t1, 1
j strcmp_loop

strcmp_equal:
add $t0, $zero, 1  # i++

strcmp_end:
lw $t3, 0($sp)   # restore the backup
addi $sp, $sp, -4
lw $t2, 0($sp)
addi $sp, $sp, -4
lw $s1, 0($sp)
addi $sp, $sp, -4
lw $t1, 0($sp)
addi $sp, $sp, -4

jr $ra

#------------------------------------------------------------
```

```
# strClear()
#------------------------------------------------------------
strClear:
addi $sp, $sp, 4   # backup
sw $t1, 0($sp)
addi $sp, $sp, 4
sw $t2, 0($sp)
addi $sp, $sp, 4
sw $s1, 0($sp)
addi $sp, $sp, 4
sw $t3, 0($sp)
addi $sp, $sp, 4
sw $s2, 0($sp)

lw $t3, code_length   # $t3 = code_length
addi $t1, $zero, -1  # $t1 = -1 = i

strClear_loop:
addi $t1, $t1, 1   # i++
sb $zero, control_code  # control_code[i] = '\0'

bne $t1, $t3, strClear_loop # if $t1 <=3 resetInput loop
nop

sw $zero, code_length  # reset code_length = 0
```

```
strClear_end:
lw $s2, 0($sp)   # restore backup
addi $sp, $sp, -4
lw $t3, 0($sp)
addi $sp, $sp, -4
lw $s1, 0($sp)
addi $sp, $sp, -4
lw $t2, 0($sp)
addi $sp, $sp, -4
lw $t1, 0($sp)
addi $sp, $sp, -4

jr $ra

#------------------------------------------------------------
# strCpy1(): copy value from prev to current code
#------------------------------------------------------------
strCpy1:
addi $sp, $sp, 4   # backup
sw $t1, 0($sp)
addi $sp, $sp, 4
sw $t2, 0($sp)
addi $sp, $sp, 4
sw $s1, 0($sp)
```

```
addi $sp, $sp, 4
sw $t3, 0($sp)
addi $sp, $sp, 4
sw $s2, 0($sp)

li $t2, 0
# load address of control_code
la $s1, control_code

# load address of prev_control_code
la $s2, prev_control_code

strCpy1_loop:
beq $t2, 3, strCpy1_end

# $t1 as control_code[i]
lb $t1, 0($s2)
sb $t1, 0($s1)

addi $s1, $s1, 1
addi $s2, $s2, 1
addi $t2, $t2, 1

j strCpy1_loop
```

```
strCpy1_end:
# reset code length
li $t3, 3
sw $t3, code_length

lw $s2, 0($sp)   # restore backup
addi $sp, $sp, -4
lw $t3, 0($sp)
addi $sp, $sp, -4
lw $s1, 0($sp)
addi $sp, $sp, -4
lw $t2, 0($sp)
addi $sp, $sp, -4
lw $t1, 0($sp)
addi $sp, $sp, -4

jr $ra


#-----------------------------------------------------------
# strCpy2(): copy value from current code to prev code
#-----------------------------------------------------------
strCpy2:
addi $sp, $sp, 4   # backup
sw $t1, 0($sp)
```

```
addi $sp, $sp, 4
sw $t2, 0($sp)
addi $sp, $sp, 4
sw $s1, 0($sp)
addi $sp, $sp, 4
sw $t3, 0($sp)
addi $sp, $sp, 4
sw $s2, 0($sp)

li $t2, 0
# load address of prev_control_code
la $s1, prev_control_code

# load address of control_code
la $s2, control_code

strCpy2_loop:
beq $t2, 3, strCpy2_end

# $t1 as control_code[i]
lb $t1, 0($s2)
sb $t1, 0($s1)

addi $s1, $s1, 1
addi $s2, $s2, 1
```

```
addi $t2, $t2, 1

j strCpy2_loop

strCpy2_end:
lw $s2, 0($sp)   # restore backup
addi $sp, $sp, -4
lw $t3, 0($sp)
addi $sp, $sp, -4
lw $s1, 0($sp)
addi $sp, $sp, -4
lw $t2, 0($sp)
addi $sp, $sp, -4
lw $t1, 0($sp)
addi $sp, $sp, -4

jr $ra

#=====================================================================
==========
# GENERAL INTERRUPT SERVED ROUTINE for all interrupts
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
.ktext 0x80000180
#-------------------------------------------------------
# SAVE the current REG FILE to stack
```

```
#--------------------------------------------------------
backup:
addi $sp, $sp, 4
sw $ra, 0($sp)

addi $sp, $sp, 4
sw $t1, 0($sp)

addi $sp, $sp, 4
sw $t2, 0($sp)

addi $sp, $sp, 4
sw $t3, 0($sp)

addi $sp, $sp, 4
sw $a0, 0($sp)

addi $sp, $sp, 4
sw $at, 0($sp)
addi $sp, $sp, 4
sw $s0, 0($sp)
addi $sp, $sp, 4
sw $s1, 0($sp)
addi $sp, $sp, 4
sw $s2, 0($sp)
```

```
addi $sp, $sp, 4
sw $t4, 0($sp)
addi $sp, $sp, 4
sw $s3, 0($sp)
#---------------------------------------------------------
# Processing
#---------------------------------------------------------
getCode:
li $t1, IN_ADRESS_HEXA_KEYBOARD
li $t2, OUT_ADRESS_HEXA_KEYBOARD

# scan row 1
li $t3, 0x81
sb $t3, 0($t1)
lbu $a0, 0($t2)
bnez $a0, getCodeInChar

# scan row 2
li $t3, 0x82
sb $t3, 0($t1)
lbu $a0, 0($t2)
bnez $a0, getCodeInChar

# scan row 3
li $t3, 0x84
```

```
sb $t3, 0($t1)
lbu $a0, 0($t2)
bnez $a0, getCodeInChar

# scan row 4
li $t3, 0x88
sb $t3, 0($t1)
lbu $a0, 0($t2)
bnez $a0, getCodeInChar

getCodeInChar:
beq $a0, KEY_0, case_0
beq $a0, KEY_1, case_1
beq $a0, KEY_2, case_2
beq $a0, KEY_3, case_3
beq $a0, KEY_4, case_4
beq $a0, KEY_5, case_5
beq $a0, KEY_6, case_6
beq $a0, KEY_7, case_7
beq $a0, KEY_8, case_8
beq $a0, KEY_9, case_9
beq $a0, KEY_a, case_a
beq $a0, KEY_b, case_b
beq $a0, KEY_c, case_c
beq $a0, KEY_d, case_d
```

```
beq $a0, KEY_e, case_e
beq $a0, KEY_f, case_f

case_0:
li $s0, '0'  # $s0 store code in char type
j storeCode
case_1:
li $s0, '1'
j storeCode
case_2:
li $s0, '2'
j storeCode
case_3:
li $s0, '3'
j storeCode
case_4:
li $s0, '4'
j storeCode
case_5:
li $s0, '5'
j storeCode
case_6:
li $s0, '6'
j storeCode
case_7:
```

```
li $s0, '7'
j storeCode
case_8:
li $s0, '8'
j storeCode
case_9:
li $s0, '9'
j storeCode
case_a:
li $s0, 'a'
j storeCode
case_b:
li $s0, 'b'
j storeCode
case_c:
li $s0, 'c'
j storeCode
case_d:
li $s0, 'd'
j storeCode
case_e:
li $s0, 'e'
j storeCode
case_f:
li $s0, 'f'
```

```
j storeCode

storeCode:
la  $s1, control_code
la $s2, code_length
lw $s3, 0($s2)   # $s3 = strlen(control_code)
addi $t4, $t4, -1   # $t4 = i

storeCodeLoop:
addi $t4, $t4, 1
bne $t4, $s3, storeCodeLoop
add $s1, $s1, $t4  # $s1 = control_code + i
sb $s0, 0($s1)   # control_code[i] = $s0

addi $s0, $zero, '\n'  # add '\n' character to end of string
addi $s1, $s1, 1
sb $s0, 0($s1)

addi $s3, $s3, 1
sw $s3, 0($s2)   # update code_length

#---------------------------------------------------------
# Evaluate the return address of main routine
# epc <= epc + 4
#---------------------------------------------------------
```

```
next_pc:
mfc0 $at, $14  # $at <= Coproc0.$14 = Coproc0.epc
addi $at, $at, 4  # $at = $at + 4 (next instruction)
mtc0 $at, $14  # Coproc0.$14 = Coproc0.epc <= $at
#---------------------------------------------------------
# RESTORE the REG FILE from STACK
#---------------------------------------------------------
restore:
lw $s3, 0($sp)
addi $sp, $sp, -4
lw $t4, 0($sp)
addi $sp, $sp, -4
lw $s2, 0($sp)
addi $sp, $sp, -4
lw $s1, 0($sp)
addi $sp, $sp, -4
lw $s0, 0($sp)
addi $sp, $sp, -4
lw $at, 0($sp)
addi $sp, $sp, -4
lw $a0, 0($sp)
addi $sp, $sp, -4
lw $t3, 0($sp)
addi $sp, $sp, -4
lw $t2, 0($sp)
```

```
addi $sp, $sp, -4

lw $t1, 0($sp)

addi $sp, $sp, -4

lw $ra, 0($sp)

addi $sp, $sp, -4

return: eret # Return from exception
```