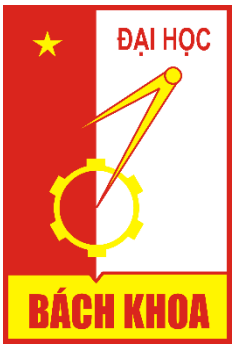
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



FINAL PROJECT REPORT

COMPUTER ARCHITECTURE LAB

Supervisor: Le Ba Vui

Group 18

Trinh Giang Nam 20215229

Nguyen Huu Phuc 20215234

Hanoi, 2024

TABLE OF CONTENTS

Project 1: Curiosity Marsbot

I.1. 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 a 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.

I.2. Project Implementation

I.2.1. Algorithm

Step 1: Press the keys on the Digital Lab Sim, storing each character in a character string.

Step 2: Enter/Delete in the Keyboard & Display MMIO Simulator:

- If Enter is pressed: check if the entered control code is valid in terms of length and matches one of the predefined codes.

- If Delete is pressed: delete the currently entered control code.
- If Space is pressed: repeat the last taken control code.

Step 3: Execute the entered control code.

Step 4: If the entered code is for turning left (right), save the x, y coordinates, and angle before turning into three arrays of integers sequentially (x_history, y_history, a_history).

Step 5: Print the entered control code on the screen and repeat Step 1.

I.2.2. Subprogram

- In main:
    + setStartHeading: set the initial angle of Marsbot
    + printErrorMsg: print the error message
    + printCmd: print the control code
    + resetInput: delete the pressed control code to prepare for the next one
    + repeatInput: repeat the previous control code
    + waitForKey: wait for key pressing in Digital Lab Sim
    + readKey: read the character pressed from keyboard and Display MIMO Simulator
    + checkCmd: check if the control code is valid
    + go, stop, turnLeft, turnRight, track, untrack, goBackward: execute the control code
- Marsbot function:
    + GO, STOP: control Marsbot to move or stop and store the state in isGoing
    + ROTATE: control Marsbot to rotate according to the angle stored in a_current
    + TRACK, UNTRACK: control Marsbot to leave track or not, store the state in isTracking
    + saveHistory: store the coordinate of x, y and the current angle before ROTATE
- String function:
    + strcmp: compare the string in $s3 with the current control code, return a boolean value in $t0
    + strClear: delete the current control code
    + strCpyCurToPrev/strCpyPrevToCur: copy from current/previous to previous/current string

I.2.3. Codesheet

```
# 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		# assign expected row index into the byte at the address 0xFFFF0012

.eqv	OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014	#  read byte at the address 0xFFFF0014, to detect which key button was pressed

.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
        x_history:              .word 0 : 16    # 16: size of each element
        y_history:              .word 0 : 16    # debugging
        a_history:              .word 0 : 16
        l_history:              .word 4         # history length

        a_current:              .word 0         # current alpha

        isGoing:        .word 0
        isTracking:             .word 0

        cmdCode:                .space 8                # input command code
        cmdLen:                 .word 0         # input command length
        prev_cmdCode: .space 8                  # prev code

        MOVE_CODE:              .asciiz "1b4"   # control 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"
```

```
        invalidCmd_msg:         .asciiz "Invalid command 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 interrupt
        sb $t3, 0($t1)

setStartHeading:
        lw      $t7, l_history    # l_history += 4
        addi    $t7, $zero, 4     # to save x = 0; y = 0; a = 90
        sw      $t7, l_history

        li      $t7, 90
        sw      $t7, a_current    # a_current = 90 -> heading down
        jal     ROTATE
        nop

        sw      $t7, a_history    # a_history[1] = 90

        j       waitForKey

printErrorMsg:
        li      $v0, 4
        la      $a0, invalidCmd_msg
        syscall
```

```
printCmd:       li      $v0, 4
                la      $a0, cmdCode
                syscall
                j       resetInput

repeatCmd:      # copy from the prev_cmdCode
                jal strCpyPrevToCur
                j checkCmd

resetInput:     jal     strClear
                nop

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
                beq     $t6, 0x7f , resetInput      # if $t6 == 'DEL' -> reset input
                beq     $t6, 0x20, repeatCmd        # if $t6 == 'SPACE' -> repeat input

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

checkCmd:       lw      $s2, cmdLen                 # cmdLen != 3 -> invalid cmd
                bne     $s2, 3, printErrorMsg

                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       printErrorMsg

#-------------------------------------------------------------
go:     jal     strCpyCurToPrev
        jal     GO
```

```
        j       printCmd

#---------------------------------------------------------
stop:   jal     strCpyCurToPrev
        jal     STOP
        j       printCmd

#---------------------------------------------------------
track:  jal     strCpyCurToPrev
        jal     TRACK
        j       printCmd

#---------------------------------------------------------
untrack:        jal     strCpyCurToPrev
        jal     UNTRACK
        j       printCmd

#---------------------------------------------------------
turnRight:      jal     strCpyCurToPrev
        lw      $t7, isGoing
        lw      $s0, isTracking

        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       printCmd

#-------------------------------------------------------------
turnLeft:       jal     strCpyCurToPrev
        lw      $t7, isGoing
        lw      $s0, isTracking

        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 # if not tracking then skip track
        nop
        jal     TRACK
        noTrack2:       nop

        beqz    $t7, noGo2      # if not going then skip go
        nop
        jal     GO
        noGo2: nop

        j       printCmd

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

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

goBackward_turn:
        addi    $s5, $s5, -4                # cmdLen--
        lw      $s6, a_history($s5)                 # $s6 = a_history[cmdLen]
        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       printCmd

#---------------------------------------------------------
# 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                  # isGoing = 0
        sw      $t7, isGoing

        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, isGoing          # isGoing = 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, isTracking

        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, isTracking

        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 cmdCode
# - 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      $t2, cmdCode($t1)               # $t2 = cmdCode[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, cmdLen                     # $t3 = cmdLen
        addi    $t1, $zero, -1          # $t1 = -1 = i

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

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

        sw      $zero, cmdLen           # reset cmdLen = 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

# strCpyPrevToCur(): copy value from prev to current code
#-------------------------------------------------------
strCpyPrevToCur:
        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 cmdCode
        la $s1, cmdCode

        # load address of prev_cmdCode
        la $s2, prev_cmdCode

strCpyPrevToCur_loop:
        beq $t2, 3, strCpyPrevToCur_end

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

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

        j strCpyPrevToCur_loop

strCpyPrevToCur_end:
```

```
        # reset code length
        li $t3, 3
        sw $t3, cmdLen

        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


#-------------------------------------------------------------
# strCpyCurToPrev(): copy value from current code to prev code
#-------------------------------------------------------------
strCpyCurToPrev:
        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_cmdCode
        la $s1, prev_cmdCode

        # load address of cmdCode
        la $s2, cmdCode

strCpyCurToPrev_loop:
        beq $t2, 3, strCpyCurToPrev_end

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

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

        j strCpyCurToPrev_loop

strCpyCurToPrev_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
#-------------------------------------------------------
get_cod:        li      $t1, IN_ADRESS_HEXA_KEYBOARD
        li      $t2, OUT_ADRESS_HEXA_KEYBOARD

scan_row1:      li      $t3, 0x81
        sb      $t3, 0($t1)
        lbu     $a0, 0($t2)
        bnez    $a0, get_code_in_char

scan_row2:      li      $t3, 0x82
        sb      $t3, 0($t1)
        lbu     $a0, 0($t2)
        bnez    $a0, get_code_in_char

scan_row3:      li      $t3, 0x84
        sb      $t3, 0($t1)
        lbu     $a0, 0($t2)
        bnez    $a0, get_code_in_char

scan_row4:      li      $t3, 0x88
        sb      $t3, 0($t1)
        lbu     $a0, 0($t2)
        bnez    $a0, get_code_in_char

get_code_in_char:
        beq     $a0, KEY_0, case_0
```

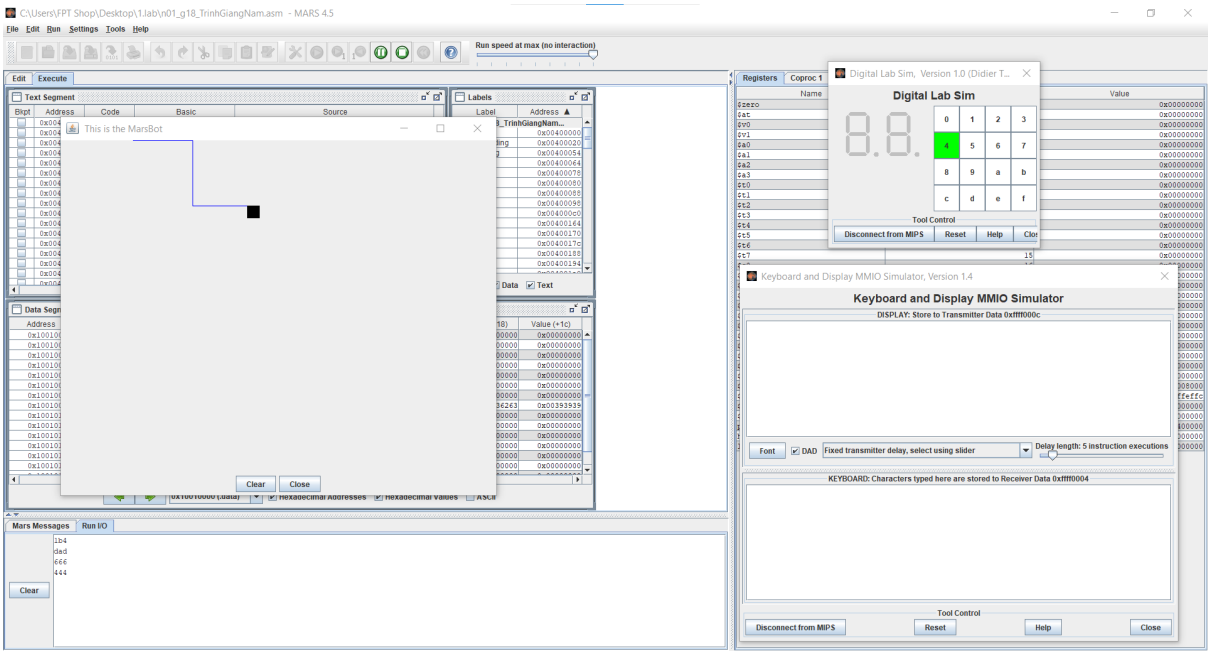
```
        beq     $a0, KEY_1, case_1
        beq     $a0, KEY_2, case_2
        beq     $a0, KEY_3, case_3
        beq     $a0, KEY_4, case_4
        beq     $a0, KEY_5, case_5
        beq     $a0, KEY_6, case_6
        beq     $a0, KEY_7, case_7
        beq     $a0, KEY_8, case_8
        beq     $a0, 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       store_code
case_1: li      $s0, '1'
        j       store_code
case_2: li      $s0, '2'
        j       store_code
case_3: li      $s0, '3'
        j       store_code
case_4: li      $s0, '4'
        j       store_code
case_5: li      $s0, '5'
        j       store_code
case_6: li      $s0, '6'
        j       store_code
case_7: li      $s0, '7'
```

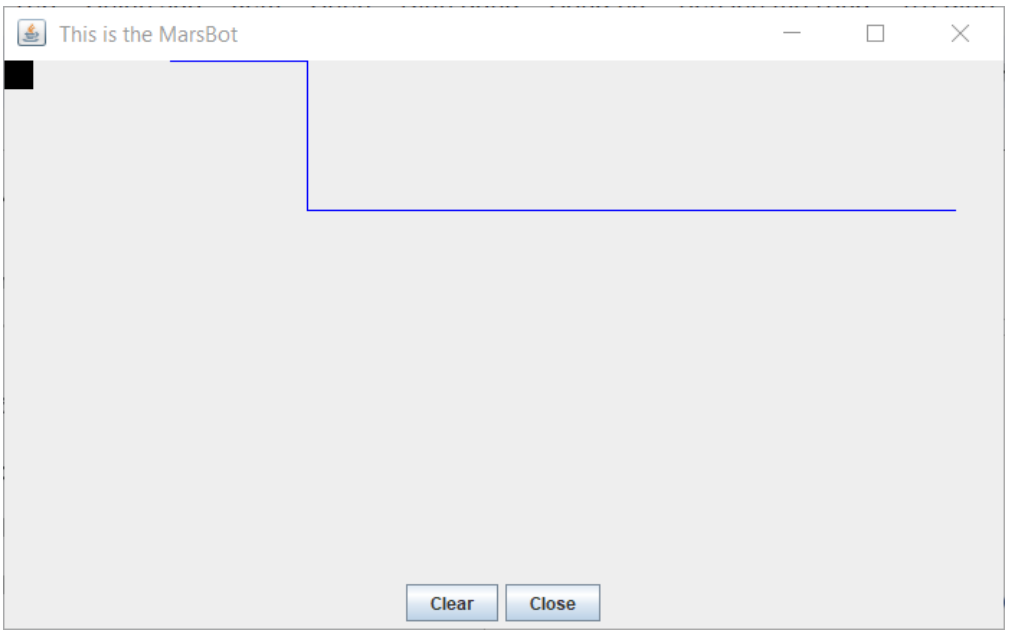
```
        j       store_code
case_8: li      $s0, '8'
        j       store_code
case_9: li      $s0, '9'
        j       store_code
case_a: li      $s0, 'a'
        j       store_code
case_b: li      $s0, 'b'
        j       store_code
case_c: li      $s0, 'c'
        j       store_code
case_d: li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

store_code:     la      $s1, cmdCode
        la      $s2, cmdLen
        lw      $s3, 0($s2)                     # $s3 = strlen(cmdCode)
        addi    $t4, $t4, -1            # $t4 = i

store_code_loop:
        addi $t4, $t4, 1
        bne     $t4, $s3, store_code_loop
        add     $s1, $s1, $t4           # $s1 = cmdCode + i
        sb      $s0, 0($s1)                     # cmdCode[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 cmdLen

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

I.3. Result





Project 10: MIPS - Digital Lab Sim simple calculator

II.1. 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

II.2. Project implementation

II.2.1. Algorithm

The program will be a continuous loop on detector keys hit on the Digital Lab Sim interface. Initially, you enter the first operand, and it's saved into a register memory. If you continue calculating with another operator by pressing = and that operator, the result is calculated and saved into the register memory before you enter it. Else, if you press =, it will return the calculation result and stop calculating.

II.2.2. Codesheet

```
.eqv IN_ADDRESS_HEXA_KEYBOARD 0xFFFF0012

.eqv OUT_ADDRESS_HEXA_KEYBOARD 0xFFFF0014

.eqv SEVENSEG_LEFT 0xFFFF0011
```

```
.eqv SEVENSEG_RIGHT 0xFFFF0010


.data
# 7-segment display values for digits 0-9
SEGMENT_VALUES: .word 0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x79


.text


main:
   li $t1, IN_ADDRESS_HEXA_KEYBOARD
   li $t2, OUT_ADDRESS_HEXA_KEYBOARD


   li $s0, 0 #s0 is first variable
   li $s1, 0 #s1 is second variable
   li $s2, 0 #s2 is unit number display code
   li $s3, 0
   li $s4, 0 #s4 is 10- number display code
   li $s5, 0
   li $s7, 0 #s7 is the operation memory
   j loop
print0:
   li $a0, 0
   j printChar
print1:
   li $a0, 1
   j printChar
```

```
print2:
   li $a0, 2
   j printChar
print3:
   li $a0, 3
   j printChar
print4:
   li $a0, 4
   j printChar
print5:
   li $a0, 5
   j printChar
print6:
   li $a0, 6
   j printChar
print7:
   li $a0, 7
   j printChar
print8:
   li $a0, 8
   j printChar
print9:
   li $a0, 9
   j printChar

displayReset:
```

```
    li $s0, 0


    li $s4, 0
    li $s2, 0


    lw $s5, SEGMENT_VALUES($s4)
    jal SHOW_7SEG_LEFT
    lw $s3, SEGMENT_VALUES($s2)
    jal SHOW_7SEG_RIGHT
    j cont


swapVar:
    la $s1, ($s0)
    j displayReset
printErr:
    li $s4, 40
    lw $s5, SEGMENT_VALUES($s4)
    jal SHOW_7SEG_LEFT
    li $s2, 40
    lw $s3, SEGMENT_VALUES($s2)
    jal SHOW_7SEG_RIGHT
    j exit
printPlus:
    li $s7, 1
    beq $s1, 0, swapVar
    add $s1, $s0, $s1
```

```
    j displayReset
printMinus:
    li $s7, 2
    beq $s1, 0, swapVar
    slt $a3, $s0, $s1
    beq $a3, 0, printErr
    sub $s1, $s1, $s0
    j displayReset
printMul:
    li $s7, 3
    beq $s1, 0, swapVar
    mul $s1, $s0, $s1
    j displayReset
printDiv:
    li $s7, 4
    beq $s1, 0, swapVar
    beq $s0, 0, printErr
    div $s1, $s0
    mflo $s1
    j displayReset
printMod:
    li $s7, 5
    beq $s1, 0, swapVar
    beq $s0, 0, printErr
    div $s1, $s0
    mfhi $s1
```

```
    j displayReset


Plus:
    add $s1, $s0, $s1
    li $s0, 0
    j next
Minus:
    slt $a3, $s0, $s1
    beq $a3, 0, printErr
    sub $s1, $s1, $s0
    li $s0, 0
    j next
Mul:
    mul $s1, $s0, $s1
    li $s0, 0
    j next
Div:
    beq $s0, 0, printErr
    div $s1, $s0
    mflo $s1
    li $s0, 0
    j next
Mod:
    beq $s0, 0, printErr
    div $s1, $s0
    mfhi $s1
```

```
    li $s0, 0
    j next

printEq: #Get the value of s1 to screen
    li $a0, '='
    beq $s7, 1, Plus
    beq $s7, 2, Minus
    beq $s7, 3, Mul
    beq $s7, 4, Div
    beq $s7, 5, Mod

    next:
    li $t3, 4
    li $s6, 100
    div $s1, $s6
    mfhi $t6
    li $s6, 10
    div $t6, $s6
    mfhi $t7            # Last digit
    mflo $t6            # 10- digit

    # Display the 10- number
    li $t3, 4
    la $s4, ($t6)
    mul $s4, $s4, $t3
    lw $s5, SEGMENT_VALUES($s4)
```

```
    jal SHOW_7SEG_LEFT

    # Display the last number
    li $t3, 4           # Each element in SEGMENT_VALUES is a word (4 bytes)
    la $s2, ($t7)
    mul $s2, $s2, $t3   # Multiply the last digit by 4 to get the offset
    lw $s3, SEGMENT_VALUES($s2)
    jal SHOW_7SEG_RIGHT

    j cont

printChar:
    # Print the character
    la $s4, ($s2)
    lw $s5, SEGMENT_VALUES($s4)
    jal SHOW_7SEG_LEFT

    # Calculate the address offset for SEGMENT_VALUES array
    li $t3, 4           # Each element in SEGMENT_VALUES is a word (4 bytes)
    la $s2, ($a0)
    mul $s2, $s2, $t3   # Multiply the last digit by 4 to get the offset
    lw $s3, SEGMENT_VALUES($s2)
    jal SHOW_7SEG_RIGHT

    # Load the 2 stored variables s0, s1 for calculation
```

```
    mul $s0, $s0, 10
    add $s0, $s0, $a0

printTerminal:
    j cont

SHOW_7SEG_LEFT:
    li $t0, SEVENSEG_LEFT
    sb $s5, 0($t0)
    jr $ra
    #j printTerminal

SHOW_7SEG_RIGHT:
    li $t0, SEVENSEG_RIGHT
    sb $s3, 0($t0)
    jr $ra
    #j printTerminal

print:
    # Check each possible value of $a0 and print the corresponding character
    beq $a0, 0x11, print0
    beq $a0, 0x21, print1
    beq $a0, 0x41, print2
    beq $a0, 0xffffff81, print3
    beq $a0, 0x12, print4
    beq $a0, 0x22, print5
```

```
    beq $a0, 0x42, print6
    beq $a0, 0xffffff82, print7
    beq $a0, 0x14, print8
    beq $a0, 0x24, print9
    beq $a0, 0x44, printPlus
    beq $a0, 0xffffff84, printMinus
    beq $a0, 0x18, printMul
    beq $a0, 0x28, printDiv
    beq $a0, 0x48, printMod
    beq $a0, 0xffffff88, printEq
    j printTerminal

loop_rows:
        li $t3, 0x01
        li $t4, 1
        li $a0, 500
        li $v0, 32
        syscall

loop:
        beq $t4, 5, loop_rows
        sb $t3, 0($t1)  # must reassign expected row
        lb $a0, 0($t2)  # read scan code of key button
        # Print the row and scan code

        bne $a0, 0, print
```

```
cont:
    # Increment row and check if all rows have been processed
    beq $t4, 5, loop_rows
    sll $t3, $t3, 1   # Shift left to the next row
    addi $t4, $t4, 1
    li $a0, 100
    li $v0, 32
    syscall
    j loop

  # End of program
exit:
  li $v0, 10
  syscall
```

II.2.3. Code explanation

The program starts with j loop where it starts detecting pressed buttons across the whole DLS interface.

```
loop_rows:
    li $t3, 0x01
    li $t4, 1
    li $a0, 500
    li $v0, 32
    syscall

loop:
    beq $t4, 5, loop_rows
```

```
    sb $t3, 0($t1)  # must reassign expected row
    lb $a0, 0($t2)  # read scan code of key button
    # Print the row and scan code

    bne $a0, 0, print
```

Once the user pressed a button, the program will check which button they have pressed for its according numbers or operator.

```
print:
  # Check each possible value of $a0 and print the corresponding character
  beq $a0, 0x11, print0
  beq $a0, 0x21, print1
  beq $a0, 0x41, print2
  beq $a0, 0xffffff81, print3
  beq $a0, 0x12, print4
  beq $a0, 0x22, print5
  beq $a0, 0x42, print6
  beq $a0, 0xffffff82, print7
  beq $a0, 0x14, print8
  beq $a0, 0x24, print9
  beq $a0, 0x44, printPlus
  beq $a0, 0xffffff84, printMinus
  beq $a0, 0x18, printMul
  beq $a0, 0x28, printDiv
  beq $a0, 0x48, printMod
  beq $a0, 0xffffff88, printEq
  j printTerminal
```

Initially, the first operand is handled by print1, print2, print…, print9 which are all hooked into printChar to display the input numbers on the screen. Things get interesting at the first operator input. Without saving the first operand input, once you press =, the memory

numbers will all disappear, giving no chance to continue the calculation. A solution to tackle that problem is to introduce the operator memory $t7. Other things in the code below include the displayReset block for resetting the screen to 00, printErr block for printing errors, and the swapVar function for pasting the information of $s0 to $s1.

```
displayReset:
   li $s0, 0

   li $s4, 0
   li $s2, 0

   lw $s5, SEGMENT_VALUES($s4)
   jal SHOW_7SEG_LEFT
   lw $s3, SEGMENT_VALUES($s2)
   jal SHOW_7SEG_RIGHT
   j cont

swapVar:
   la $s1, ($s0)
   j displayReset
printErr:
   li $s4, 40
   lw $s5, SEGMENT_VALUES($s4)
   jal SHOW_7SEG_LEFT
   li $s2, 40
   lw $s3, SEGMENT_VALUES($s2)
   jal SHOW_7SEG_RIGHT
   j exit
printPlus:
   li $s7, 1
   beq $s1, 0, swapVar
```

```
    add $s1, $s0, $s1
    j displayReset
printMinus:
    li $s7, 2
    beq $s1, 0, swapVar
    slt $a3, $s0, $s1
    beq $a3, 0, printErr    #Not letting variables to not be a negative number
    sub $s1, $s1, $s0
    j displayReset
printMul:
    li $s7, 3
    beq $s1, 0, swapVar
    mul $s1, $s0, $s1
    j displayReset
printDiv:
    li $s7, 4
    beq $s1, 0, swapVar
    beq $s0, 0, printErr     #Not letting divisor be 0
    div $s1, $s0
    mflo $s1
    j displayReset
printMod:
    li $s7, 5
    beq $s1, 0, swapVar
    beq $s0, 0, printErr    #Not leading divisor be 0
    div $s1, $s0
    mfhi $s1
    j displayReset
```

Once more, we come to handling the Equal button. The $t7 we established earlier now helps us when the user press the equal button after completing inputting. It helps memorize

which was the operator used. But we can't reuse the above printMinus, printPlus… blocks as they all have the displayReset block which we can't use. So, the Plus, Minus,… is there instead.

```
 Plus:
  add $s1, $s0, $s1
  li $s0, 0
  j next
Minus:
  slt $a3, $s0, $s1
  beq $a3, 0, printErr
  sub $s1, $s1, $s0
  li $s0, 0
  j next
Mul:
  mul $s1, $s0, $s1
  li $s0, 0
  j next
Div:
  beq $s0, 0, printErr
  div $s1, $s0
  mflo $s1
  li $s0, 0
  j next
Mod:
  beq $s0, 0, printErr
  div $s1, $s0
  mfhi $s1
  li $s0, 0
  j next

printEq: #Get the value of s1 to screen
```

```
li $a0, '='
beq $s7, 1, Plus
beq $s7, 2, Minus
beq $s7, 3, Mul
beq $s7, 4, Div
beq $s7, 5, Mod

next:
li $t3, 4
li $s6, 100
div $s1, $s6
mfhi $t6
li $s6, 10
div $t6, $s6
mfhi $t7                    # Last digit
mflo $t6                    # 10- digit

# Display the 10- number
li $t3, 4
la $s4, ($t6)
mul $s4, $s4, $t3
lw $s5, SEGMENT_VALUES($s4)
jal SHOW_7SEG_LEFT

# Display the last number
li $t3, 4          # Each element in SEGMENT_VALUES is a word (4 bytes)
la $s2, ($t7)
mul $s2, $s2, $t3     # Multiply the last digit by 4 to get the offset
lw $s3, SEGMENT_VALUES($s2)
jal SHOW_7SEG_RIGHT
```

j cont

II.3. Result demonstration

II.3.1. Input guide

For normal calculation with 2 operands: <operand 1><operator><operand 2> =

Continuous calculation:
<operand 1><operator 1><operand 2> = <operator 2><operand 3> = … (there are unhandled exceptions on subtraction, division and mod so not recommend to use this feature yet)

II.3.2. Normal cases

Input: 3+2=

Output: 5

Input: 6*7=*8=

Output: 36

Input: 155%15=-2=

Output: 3

II.3.3. Exception cases

Input: 3-4=

Output: EE

Input: 5/0=
Output: EE

Input: 14+15=/0=
Output: EE (unhandled)