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

FINAL PROJECT REPORT

IT3280E – ASSEMBLY LANGUAGE AND COMPUTER ARCHITECTURE LAB

Course information

Course ID	Course title	Class ID
IT3280E	Assembly language and Computer architecture Lab	143684

Student information

Student's full name	Class	StudentID
Tran Anh	ICT 02 K66	20215179
Nguyen Tho Dat	ICT 02 K66	20215193

Instructor: MSc. Le Ba Vui

Teaching Assistant: Do Gia Huy

Hanoi, January 2024

Table of Contents

I. Task 10: Simple calculator

1. Problem description

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)

2. Instructions for running the program

a) Instructions

Step 1: Run the program, open Digital Lab Sim, Connects to MIPS.

Step 2: Enter the first number -> enter the operand -> enter the second number

Step 3: Enter 'f' (alias for '=')

b) Node

- *Wait for the Digital Lab Sim to load the number correctly (the key on the screen turn green)*
- *Exception handling*
  - *Didive by zero*: The program tracks the second number entered by the user. If it is '0' and the operand is '/' or '%', the program will raise the error.
- *Consecutively enter 1 number and 2 operands, the program will consider the second number is '0' and calculate the result with the first operand.*
- *The program currently cannot consecutively calculate many operands at once (one calculation contains only one operand)*
- *When handling '-' operand, if the result is negative (<0), the program will show '00' on the LED and normal printing at RUN I/O.*

3. Code explanation

Initialization: Initialize used values on the program

```
.eqv    SEVENSEG_LEFT   0xFFFF0011      # Address of the LEFT LED
.eqv    SEVENSEG_RIGHT  0xFFFF0010      # Address of the RIGHT LED
.eqv    IN_ADDRESS_HEXA_KEYBOARD        0xFFFF0012
.eqv    OUT_ADDRESS_HEXA_KEYBOARD       0xFFFF0014

.data
# Values corresponding to LED digits
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         "Can not divide by 0!\n"

.text
main:

Init:
        li      $t0, SEVENSEG_LEFT      # Variable contains value of the LEFT LED
        li      $t5, SEVENSEG_RIGHT     # Variable contains value of the RIGHT LED
        li      $s0, 0                  # Variable contains type of input: (0: digit), (1: operand)
        li      $s1, 0                  # Variable on the LEFT LED
        li      $s2, 0                  # Variable on the RIGHT LED
        li      $s3, 0                  # Variable contains type of operand (1: +, 2: -, 3: *, 4: /, 5: %)
        li      $s4, 0                  # The first number
        li      $s5, 0                  # The second number
        li      $s6, 0                  # Result
        li      $t9, 0                  # Temp value

        li      $t1, IN_ADDRESS_HEXA_KEYBOARD   # Variable contronls keyboard rows and enable keyboard interrupt
        li      $t2, OUT_ADDRESS_HEXA_KEYBOARD  # Variable contains key locations
        li      $t3, 0x80                       # bit used to enable keyboard interrupt and enable check each keyboard row
        sb      $t3, 0($t1)
        li      $t7, 0                  # Variable contains value of number on the LED
        li      $t4, 0                  # Byte showed on LED (0->9)
```

Main

Main sectiion:

- Set up keyboard input and interrupt handling
- The program enters an infinite loop waiting for keyboard interrupts

```
43  First_value:
44          li      $t7, 0                  # Value of needed to be showed initial bit
45          addi    $sp,$sp, 4              # Push to stack
46          sb      $t7, 0($sp)
47          lb      $t4, zero               # First bit to be showed
48          addi    $sp, $sp, 4             # Push to stack
49          sb      $t4, 0($sp)
50
51  Loop1:
52          nop
53          nop
54          nop
55          nop
56          b       Loop1           # Wait for interrupt
57          nop
58          nop
59          nop
60          nop
61          b       Loop1
62          nop
63          nop
64          nop
65          nop
66          b       Loop1
67  end_loop1:
68
69  # Handle interrupt
70  # -> Show clicked key on the LED
71  # Check each row for clicked row
72  end_main:
73          li      $v0, 10
74          syscall
75
```

Interrupt Handling:

- Upon an interrupt, the program checks each keyboard row to determine the pressed key
- Exstract the pressed key
- Depend on the clicked key, the programs sets various variables to perform mathematical operations

```
77  .ktext 0x80000180
78  # If row contains clicked key
79  # -> Move to that row
80          jal     check_row1              # Check row 1
81          bnez    $t3, convert_row1       # t3 != 0 -> clicked key, find clicked key on the row -> exstract that key
82          nop
83
84          jal     check_row2              # The same go to row 2...
85          bnez    $t3, convert_row2
86          nop
87
88          jal     check_row3
89          bnez    $t3, convert_row3
90          nop
91
92          jal     check_row4
93          bnez    $t3, convert_row4
94
```

```
95  # Functions check for clicked key on the row or not
96  check_row1:
97          addi    $sp, $sp, 4
98          sw      $ra, 0($sp)             # Store -> can be changed
99          li      $t3, 0x81               # Execute interrupt
100         sb      $t3, 0($t1)
101         jal     Get_value               # Get location of clicked key (if existed)
102         lw      $ra, 0($sp)
103         addi    $sp, $sp, -4
104         jr      $ra
```

```
136 # Exstract the value of clicked key
137 Get_value:
138         addi    $sp, $sp, 4
139         sw      $ra, 0($sp)
140         li      $t2, OUT_ADDRESS_HEXA_KEYBOARD  # Address containing location of clicked key
141         lb      $t3, 0($t2)                     # Load the location
142         lw      $ra, 0($sp)
143         addi    $sp, $sp, -4
144         jr      $ra
145
146 # Convert from location -> value of the LED
147 convert_row1:
148         beq     $t3, 0x11, case_0               # Digit 0
149         beq     $t3, 0x21, case_1               # Digit 1
150         beq     $t3, 0x41, case_2               # Digit 2
151         beq     $t3, 0xffffff81, case_3         # Digit 3
152 case_0:
153         lb      $t4,zero        # t4 = 0, value of "0" on Digital Lab Sim
154         li      $t7,0           # t7 = 0
155         j       update_tg
156 case_1:
157         lb      $t4,one         # So on
158         li      $t7,1
159         j       update_tg
160 case_2:
161         lb      $t4,two
162         li      $t7,2
163         j       update_tg
164 case_3:
165         lb      $t4, three
166         li      $t7, 3
167         j       update_tg
168
```

```
# Convert the number on LED -> value of the second number
set_second_number:
        beq     $s3, 1, addition
        beq     $s3, 2, substraction
        beq     $s3, 3, multiplication
        beq     $s3, 4, division
        beq     $s3, 5, find_remainder

addition:
        add     $s6, $s5, $s4
        li      $s3, 0
        li      $t9, 0
        j       print_addition
        nop
```

Display Results and Resetting:

- Convert the result into digits and displays them on the left and right of LED
- Reset varios variables and LEDs for the next input

```
print_addition:
        li      $v0, 1
        move    $a0, $s4
        syscall
        li      $s4, 0          # reset the first number to 0

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

        li      $v0, 1
        move    $a0, $s5
        syscall
        li      $s5, 0          # Reset the second number to 0

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

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

        li      $v0, 11
        li      $a0, '\n'
        syscall
        li      $s7, 100
        div     $s6, $s7
        mfhi    $s6             # Exstract 2 last digit of the result
        j show_result_in_led    # Show it
        nop
```

Handling exception:

- Prompt “Can not divide by 0” when meeting this error

```
402 divide_by_0:
403         li      $v0, 55
404         la      $a0, mess1
405         li      $a1, 0
406         syscall
407         j       reset_led
408
```

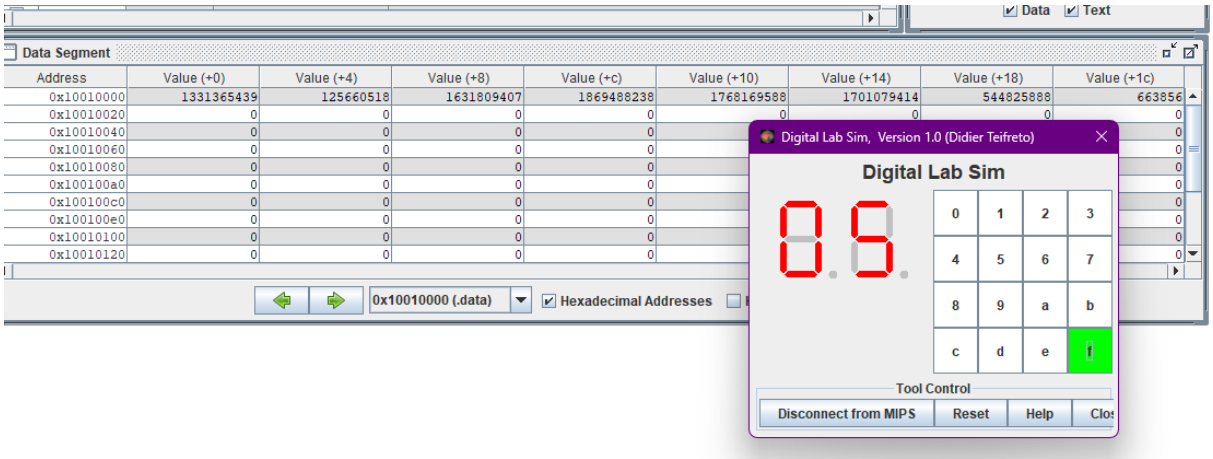
Source Code:

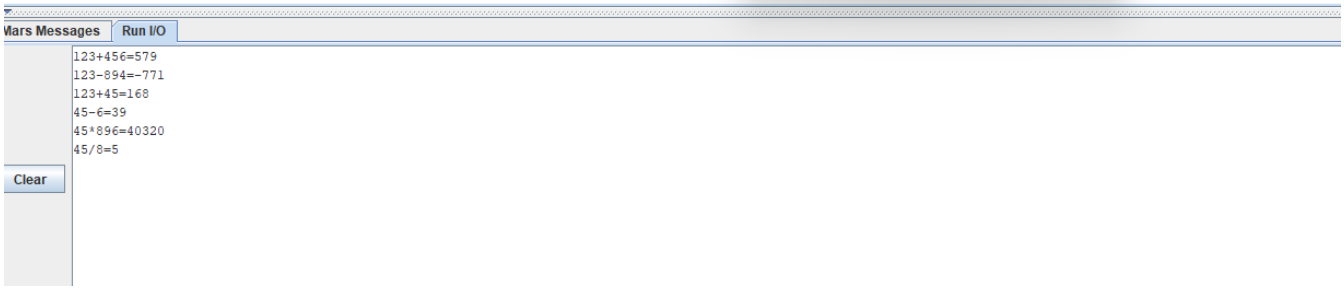
Task10_Gr4_TranAnh_20215179.asm

4. Result demonstration

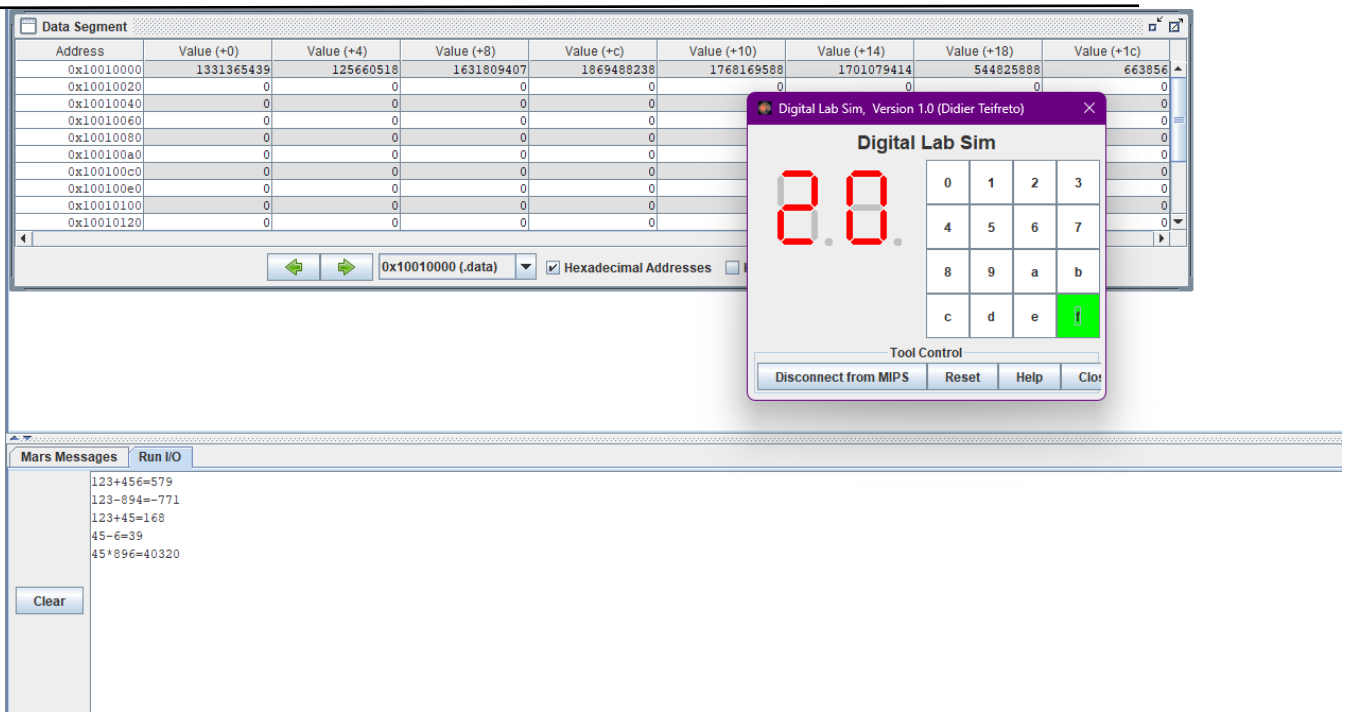
a) Normal cases:

45 / 8 = 5

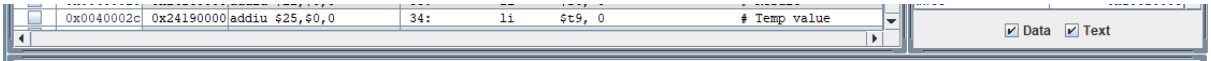


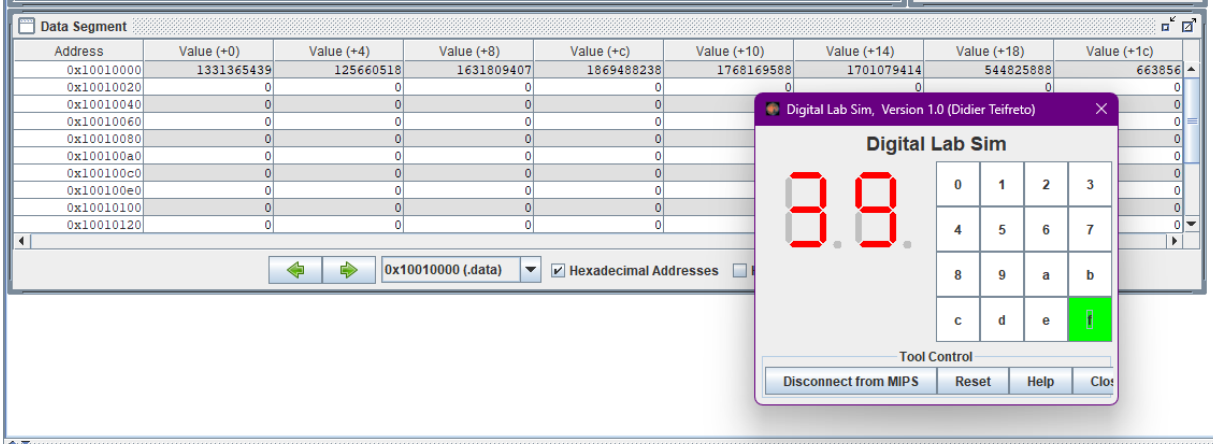


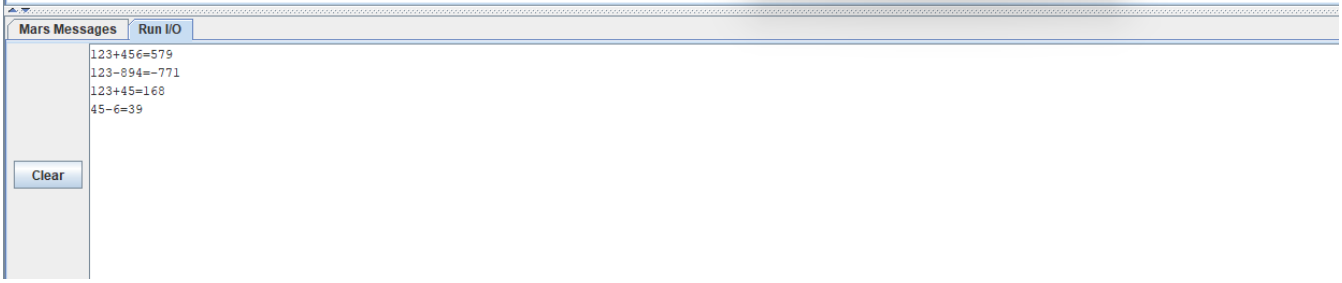
45 * 896 = 40320



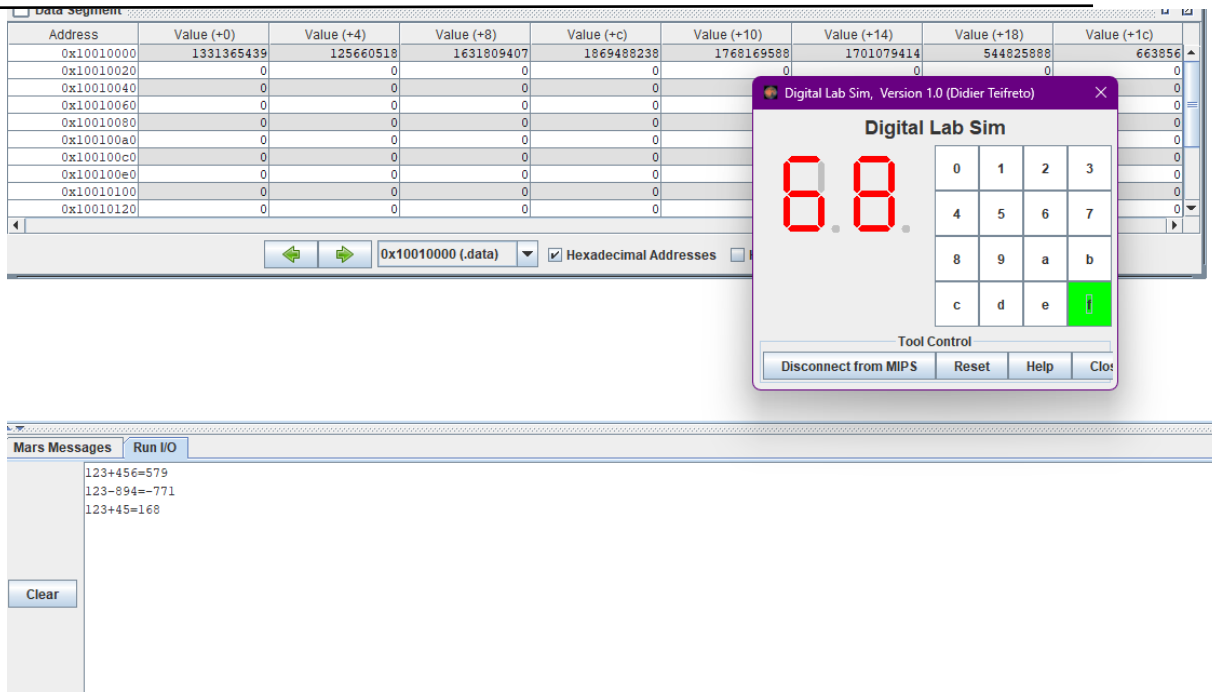
45 – 6 = 39





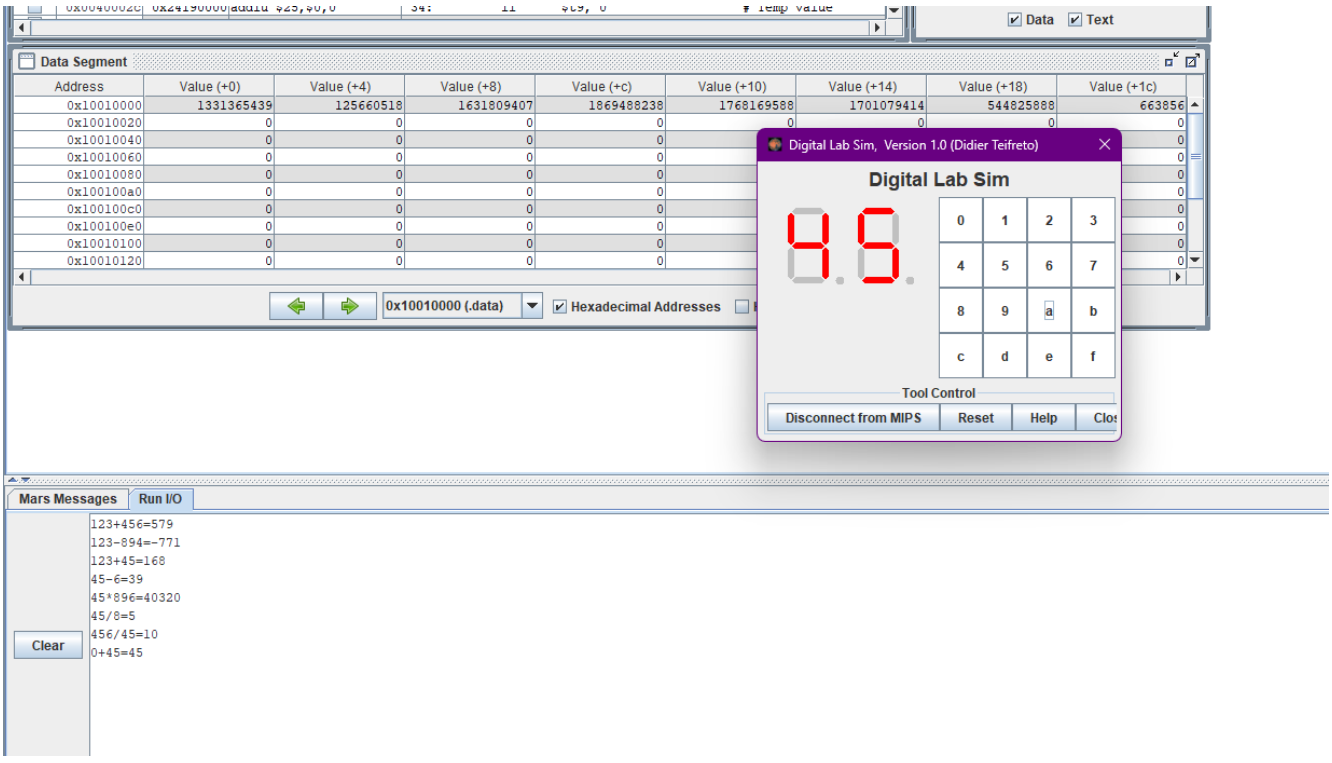


123 + 45 = 168

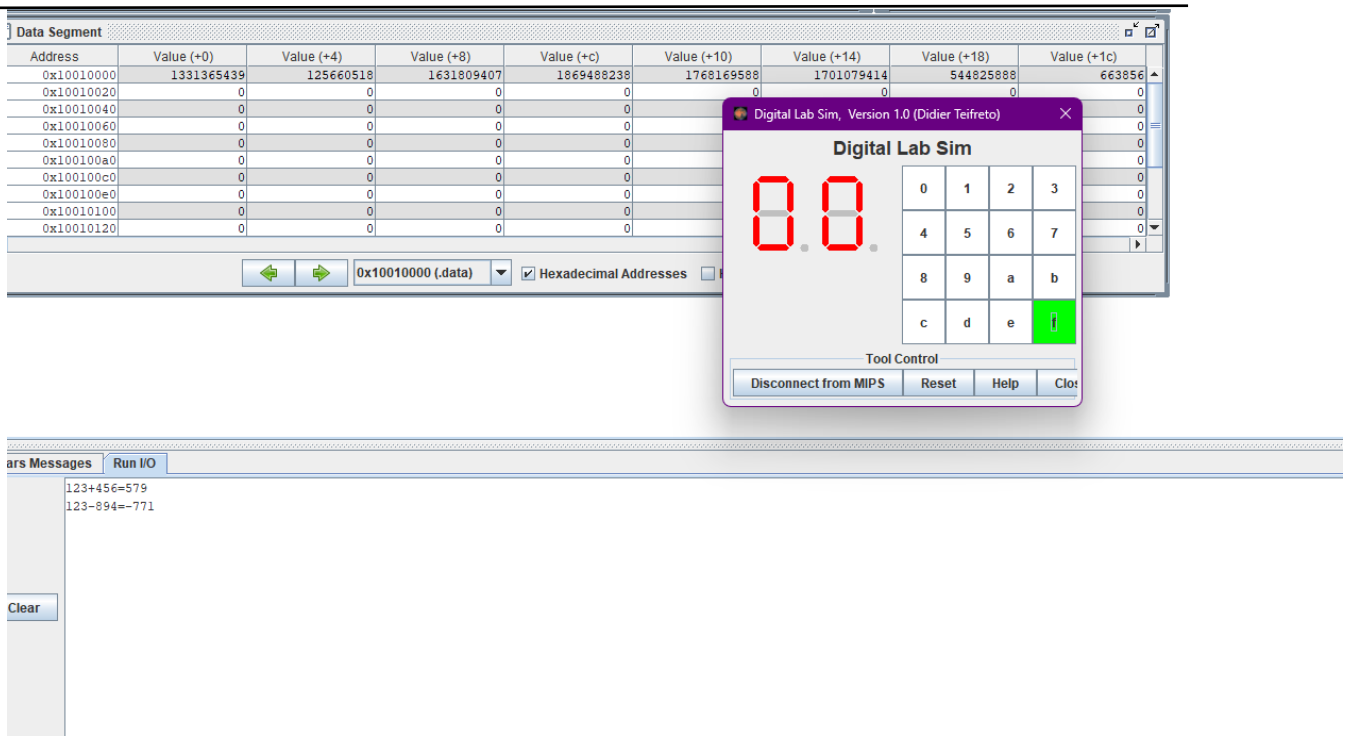


b) Handling exception:

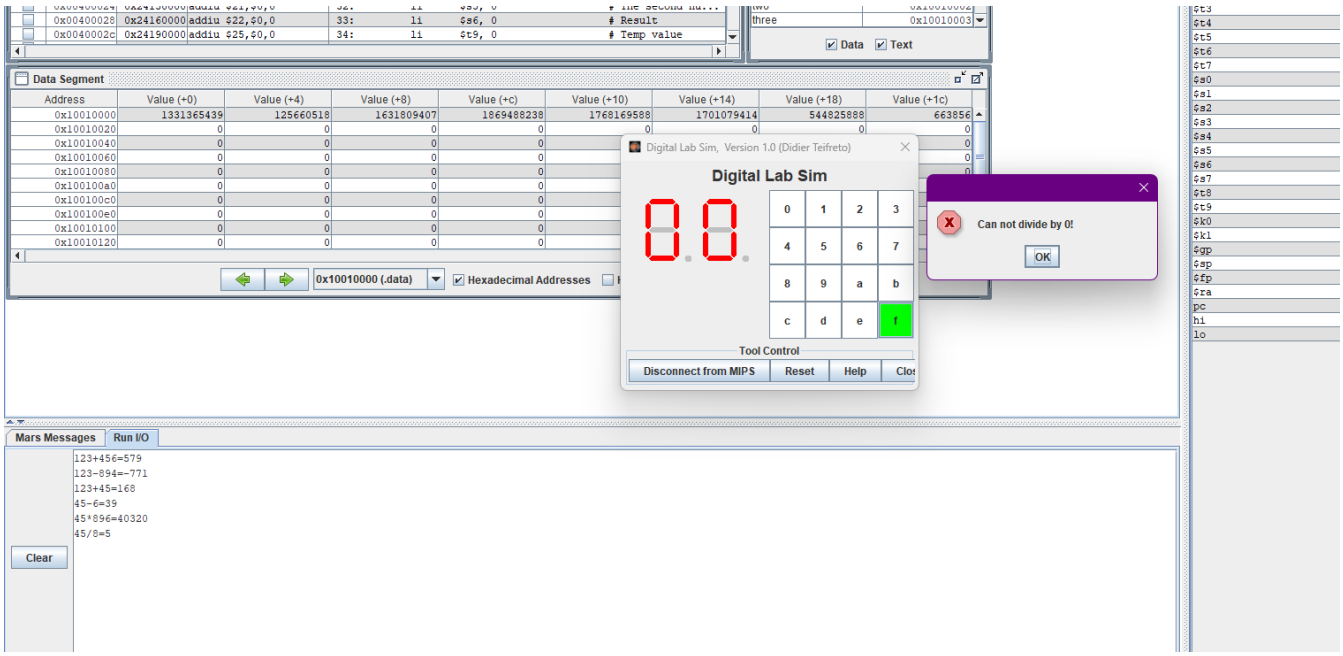
Consecutively enter



Negative result



Divide by 0



II. Task 2: Moving a ball in the bitmap display:

1. Problem description:

Create a program that displays a movable round ball on the bitmap screen. If the ball touches the edge of the screen, it will move in the opposite direction. Requirement:

- Set display width and height to 512 pixels, unit width and height to 1 pixel.
- The direction of movement depends on the key pressed from the keyboard. (W moves up, S moves down, A moves left, D moves right, Z speeds up, X slows down).
- The default position is the center of the screen.

2. Instructions for running the program:

- **Step 1:** Assemble the program, open "Bitmap Display" and "Keyboard and Display MMIO Simulator", Connects to MIPS.
- **Step 2:** Set the Display Height in Pixels to 512 on the Bitmap Display
- **Step 3**: Enter W moves up, S moves down, A moves left, D moves right, Z speeds up, X slows down into "Keyboard and Display MMIO Simulator" and watch the circle move.

3. Algorithm:

- Initialization.
  - Center of the circle coordinate: x, y.
  - Radius of the circle: R
  - Direction of the circle: dx = 1 (right), -1 (left); dy = 1 (down), -1 (up).
  - The array that contains the coordinate of pixels surrounding the center to make a circle: circle.
  - Each pixel coordinates to make a circle: px, py.
- Find the coordinates of each pixel relative to the center of the circle.
  - For px = 1 to R, $py^2 = R^2 - px^2$ => px, py
  - Then other 3 pixels on the other sides of the circle (-px,py), (px,-py), (-px,-py)
  - Save all the coordinate to the circle array.
- Input and read the input on the keyboard.
- Each case of key pressed: W moves up, S moves down, A moves left, D moves right, Z speeds up, X slows down
- Check if the circle hits the edge, if yes, then reverse the direction

4. Code Explanation:

a) Initialization:

Initialize the value

```
.eqv MONITOR_SCREEN 0x10010000 #Dia chi bat dau cua bo nho man hinh

.eqv KEY_CODE   0xFFFF0004
.eqv KEY_READY  0xFFFF0000

.data
circle:         .word

.text
initialize:
        # center
        li      $s0, 256                # x = 256
        li      $s1, 256                # y = 256
        # direction that the circle is moving
        li      $s2, 1                  # dx = 1
        li      $s3, 0                  # dy = 0
        # radius
        li      $s4, 20                 # R = 20
        # sleep time
        li      $a0, 50                 # t = 50
        jal     circle_push
```

Then do the procedure to push into the circle array

b) Circle push

Initialize some value for calculation

```
circle_push:
        addi    $sp, $sp, -4
        sw      $ra, 0($sp)
        la      $s5, circle     # $s5 = pointer of the "circle" array
        mul     $a3, $s4, $s4   # $a3 = R*R
        add     $s7, $0, $0     # px = 0
```

Calculation to find the coordinate of each pixel relative to the center of the circle.

Loop for R times,

$py^2 = R^2 - px^2$ => px, py

```
circle_cal_loop:
        bgt     $s7, $s4, circle_end
        mul     $t0, $s7, $s7   # $t0 = px^2
        sub     $a2, $a3, $t0   # $a2 = R^2 - px^2 = py^2
        beqz    $a2, cal_continue
        jal     root            # $a2 = py
cal_continue:
        move    $a1, $s7        # $a1 = px
        li      $s6, 0
```

If py = 0 then continue else

Procedure to find the square root of py^2

```
191   root:
192           li      $t0, 0
193           li      $t1, 0
194           move    $t2, $a2
195           div     $t3, $a2, 2
196   root_loop:
197           div     $t4, $a2, $t2
198           add     $t4, $t2, $t4
199           div     $t2, $t4, 2
200           addi    $t1, $t1, 1
201           blt     $t1, $t3, root_loop
202           move    $a2, $t2
203           jr      $ra
204
```

Save all the coordinates to the array then repeat the circle_push

```
159   # Saving (px, py), (-px, py), (-px, -py), (px, -py)
160   push:
161           jal     push_save
162           sub     $a1, $0, $a1
163           jal     push_save
164           sub     $a2, $0, $a2
165           jal     push_save
166           sub     $a1, $0, $a1
167           jal     push_save
168   # then save (-py, px), (py, px), (py, -px), (-py, -px)
169           move    $t0, $a1        # Swap px and -py
170           move    $a1, $a2
171           move    $a2, $t0
172           addi    $s6, $s6, 1
173           beq     $s6, 2, push_finish
174           j       push
175   push_finish:
176           addi    $s7, $s7, 1
177           j       circle_cal_loop
178
179   push_save:
180           sw      $a1, 0($s5)     # Store px
181           sw      $a2, 4($s5)     # Store py
182           addi    $s5, $s5, 8     # Move the pointer
183           jr      $ra
```

After finding all the coordinates, save the end address for later use.

```
185   circle_end:
186         move    $v1, $s5        # Save the end address of the "circle" array
187         lw      $ra, 0($sp)
188         addi    $sp, $sp, 4
189         jr      $ra
190
```

Input and read the input on the keyboard, check edge and cases for different keys pressed.

```
24  input:
25          li      $k0, KEY_READY
26          lw      $t0, 0($k0)
27          bne     $t0, 1, hit_edge
28          li      $k0, KEY_CODE
29          lw      $t0, 0($k0)
30          beq     $t0, 'a', pressed_a
31          beq     $t0, 'd', pressed_d
32          beq     $t0, 's', pressed_s
33          beq     $t0, 'w', pressed_w
34          beq     $t0, 'x', pressed_x
35          beq     $t0, 'z', pressed_z
```

Find which edge the circle moving toward to

```
66  hit_edge:
67          beq     $s2, 1, right_edge
68          beq     $s2, -1, left_edge
69          beq     $s3, -1, up_edge
70          beq     $s3, 1, down_edge
71          j       move_circle
```

Check if the pixel closest to the edge hit the edge yet

If yes, then reverse the direction of the circle

```
73  right_edge:
74          add     $t0, $s0, $s4   # Rightest side of the circle
75          beq     $t0, 511, reverse
76          j       move_circle
77
78  left_edge:
79          sub     $t0, $s0, $s4   # Leftest side of the circle
80          beq     $t0, 1, reverse
81          j       move_circle
82
83  down_edge:
84          add     $t0, $s1, $s4   # Downest side of the circle
85          bge     $t0, 511, reverse
86          j       move_circle
87
88  up_edge:
89          sub     $t0, $s1, $s4   #Upest side of the circle
90          ble     $t0, 1, reverse
91          j       move_circle
92
93  reverse:
94          sub     $s2, $0, $s2    # dx = -dx
95          sub     $s3, $0, $s3    # dy = -dy
96          j       move_circle
```

Move the circle by erasing the old circle then set the center's coordinate to the direction the circle is moving.

```
98  move_circle:
99          li      $s5, 0          # Set color to black
100         jal     draw_circle     # Erase the old circle
101
102         add     $s0, $s0, $s2   # Set the center of the new circle
103         add     $s1, $s1, $s3
104         li      $s5, 0x00FFFF00 # Set color to yellow
105         jal     draw_circle     # Draw the new circle
106
```

Get the coordinate in the circle array then draw the pixels.

```
112 draw_circle:
113         addi    $sp, $sp, -4    # Save $ra
114         sw      $ra, 0($sp)
115         la      $s6, circle     # pointer to the circle array
116
117 draw_loop:
118         beq     $s6, $v1, draw_end      # Stop when $s6 = $v1 (pointer at the end of the array)
119         lw      $a1, 0($s6)             # Get px
120         lw      $a2, 4($s6)             # Get py
121         jal     draw
122         addi    $s6, $s6, 8             # Move the pointer
123         j       draw_loop
124
125 draw_end:
126         lw      $ra, 0($sp)
127         addi    $sp, $sp, 4
128         jr      $ra
```

Find the coordinate relative to the monitor screen.

```
draw:
        li      $t0, MONITOR_SCREEN
        add     $t1, $s0, $a1
        add     $t2, $s1, $a2
        sll     $t2, $t2, 9     # Move to y coordinate
        add     $t2, $t2, $t1   # Move to x coordinate
        sll     $t2, $t2, 2     # Multiply by 4 for address
        add     $t0, $t0, $t2
        sw      $s5, 0($t0)
        jr      $ra
```

Each case of keys pressed the check again if the circle hit the edge.

```
pressed_a:
        li      $s2, -1         # dx = -1
        li      $s3, 0          # dy = 0
        j       hit_edge

pressed_d:
        li      $s2, 1          # dx = 1
        li      $s3, 0          # dy = 0
        j       hit_edge

pressed_s:
        li      $s3, 1          # dy = 1
        li      $s2, 0          # dx = 0
        j       hit_edge

pressed_w:
        li      $s3, -1         # dy = -1
        li      $s2, 0          # dx = 0
        j       hit_edge

```

Cases whether speed up or slow down the circle: adding sleep time or subtracting sleep time

```
pressed_x:
        addi    $a0, $a0, 10    # t += 10
        j       hit_edge

pressed_z:
        beq     $a0, 0, hit_edge
        addi    $a0, $a0, -10   # t -= 10
        j       hit_edge
```

Loop over from the input of the keyboard.

```
107 loop:
108         li $v0, 32              # Sleep
109         syscall
110         j       input           # Renew the cycle
```

5. Result: