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	Student ID
Mai Duc An	ICT 01-K66	20210008
Bui Duc Viet	ICT 01- K66	20215254

Instructor: MSc. Le Ba Vui

Teaching assistant: Do Gia Huy

Table of Contents

Task 5: Infix and postfix expression

1. Problem introduction

   Create a program that can calculate an expression by evaluating the postfix expression.

   Requirements:

   - Enter an infix expression from the console, for example: 9 + 2 + 8 * 6
   - Print it in the postfix representation, for example: 9 2 + 8 6 * +
   - Calculate and display the result to the console screen

   The operand must be an integer between 0 and 99

   Operators include addition, subtraction, multiplication, division, division with remainder and parenthesis

2. Project implementation

   2.1 Main idea for problem

   - Using stack to solve the problem

   2.2 Detail algorithm

   - Enter infix expression from keyboard and store it in memory. Check the condition to see if the number is within the range from 0 to 99.
   - Create a memory space to store the result of the expression in postfix form
   - Convert from infix expression to postfix
     - Initialize an empty stack
     - Scan each element from left to right in the infix expression
     - If an operand is encountered, append it to the result string
     - If an opening parenthesis is encountered, push it onto the stack
     - If encountering an operator o1, perform the following streps:
       - As long as there is another operator o2 at the top of the stack, and the precedence of o1 is less than or equal to the precedence of o2, pop o2 from the stack and append it to the result
       - Push o1 onto the stack
     - If encountering a closing parenthesis, continuously pop operators from the stack and append them to the result until an opening parenthesis is encountered and removed from the stack

- Once the entire infix expression has been traversed, sequentially pop all remaining operators (if any) from the stack and append them to the result string
- Print the result string (postfix expression) to the screen

- To evaluate the value of the postfix expression:
  - Initialize an empty stack
  - Read each element from left to right in the postfix expression
  - If the element is an operand, push its value onto the stack
  - If the element is an operator, pop 2 values (y and x) from the stack, apply the operator to these 2 values, and push the result back onto the stack
  - Continue this process until the stack contains only one element, which is the result of the expression
  - Print the result of the expression to the screen

2.3 Code explanation

- Get the infix expression

```
input_infix:
        li $v0, 54
        la $a0, msg_read_infix              # address of string of message
        la $a1, infix                               # address of input buffer
        la $a2, 256                         # maximum number of characters
        syscall
        li $v0, 4
        la $a0, msg_print_infix             # address of string to print
        syscall
        li $v0, 4
        la $a0, infix               # address of string to print
        syscall
```

- Convert infix expression to postfix

```
convert_postfix:
        li $s0, 0                           # index j of infix
        li $s1, 0                           # index i of postfix
        li $s2, -1                          # index k of stack
        li $a3, 0                           # lparcount = 0
        li $s3, 0                           # element to push to postfix
        lb $t0, infix($s0)
        beq $t0, '-', lt0_error             # check the first number is nagative
        nop
```

- Scan each element

```
loop_infix:
      lb $t0, infix($s0)                # $t0 = infix[j]
      beq $t0, $0, end_loop_infix       # $t0 = 0 -> end loop
      nop
      beq $t0, '\n', end_loop_infix
      nop
      beq $t0, ' ', remove_space1
      nop
# if $t0 is operand, to arrange according to priority of operands than push to stack
      beq $t0, '+', consider_plus_minus
      nop
      beq $t0, '-', consider_plus_minus
      nop
      beq $t0, '*', consider_mul_div
      nop
      beq $t0, '%', consider_mul_div
      nop
      beq $t0, '/', consider_mul_div
      nop
      beq $t0, '(', consider_lpar       # check nagative number if get '('
      nop
      beq $t0, ')', consider_rpar1
      nop
# if $t0 is operand, push to postfix immediately
      sb $t0, postfix($s1)        # postfix[i] = $t0
      addi $s1, $s1, 1                  # i++
```

- Scan for the next character if number is in range 9-100 and check the operand is in range 0-100 or not

```
# if 9 < number < 100 -> look ahead for a character
loop_continue:
       addi $s0, $s0, 1                  # j++
       lb $t2, infix($s0)                # $t2 = infix[j]
       beq $t2, '0', continue
       nop
       beq $t2, '1', continue
       nop
       beq $t2, '2', continue
       nop
       beq $t2, '3', continue
       nop
       beq $t2, '4', continue
       nop
       beq $t2, '5', continue
       nop
       beq $t2, '6', continue
       nop
       beq $t2, '7', continue
       nop
       beq $t2, '8', continue
       nop
       beq $t2, '9', continue
       nop
       beq $t2, ' ', loop_continue
       nop
```

```
        li $s3, ' '
        sb $s3, postfix($s1)        # postfix[i] = ' '
        addi $s1, $s1, 1                    # i++
        j loop_infix
        nop
remove_space1:
        addi $s0, $s0, 1                    # i++
        j loop_infix
        nop
# check the next character
continue:
        addi $t3, $s0, 1                    # $t3 = j++
        lb $t4, infix($t3)                  # infix[$t3] = $t4

# if greater than 99 -> branch to error alert
check_gt99:
        beq $t4, '0', gt99_error
        nop
        beq $t4, '1', gt99_error
        nop
        beq $t4, '2', gt99_error
        nop
        beq $t4, '3', gt99_error
        nop
        beq $t4, '4', gt99_error
        nop
        beq $t4, '5', gt99_error
        nop
        beq $t4, '6', gt99_error
        nop
        beq $t4, '7', gt99_error
        nop
        beq $t4, '8', gt99_error
        nop
        beq $t4, '9', gt99_error
        nop
        sb $t2, postfix($s1)                # else postfix[i] = $t2
        addi $s1, $s1, 1                    # i++
        li $s3, ' '
        sb $s3, postfix($s1)                # postfix[i] = ' '
        addi $s1, $s1, 1                    # i++
        addi $s0, $s0, 1                    # j++
        j       loop_infix
        nop
gt99_error:
        li $v0, 55
        la $a0, msg_error1                  # address of error message
        syscall
        j       input_infix
        nop
```

- Function handling whenever ‘-’ or ‘+’ is encountered

```
consider_plus_minus:
        beq $s2, -1, push_op        # if stack is null, push this operand to stack
        nop
        lb $t9, stack($s2)
        beq $t9, '(', push_op               # if peek of stack is '(', push this operand to stack
        nop
        lb $t1, stack($s2)          # else pop all operands out of stack then push this operand to stack
        sb $t1, postfix($s1)
        addi $s2, $s2, -1                   # k--
        addi $s1, $s1, 1                    # i++
        li $s3, ' '
        sb $s3, postfix($s1)        # postfix[i] = ' '
        addi $s1, $s1, 1                    # i++
        j consider_plus_minus
        nop
```

- Function handling whenever ‘*’ or ‘/’ is encountered

```
consider_mul_div:
      beq $s2, -1, push_op        # if stack is null, push this operator to stack
      nop
      lb $t9, stack($s2)
      beq $t9, '+', push_op             # if peek of stack is '+', '-', '(', push this operator to stack
      nop
      beq $t9, '-', push_op
      nop
      beq $t9, '(', push_op
      nop
      lb $t1, stack($s2)          # else pop all operands out of stack then push this operand to stack
      sb $t1, postfix($s1)
      addi $s2, $s2, -1                 # k--
      addi $s1, $s1, 1                  # i++
      li $s3, ' '
      sb $s3, postfix($s1)        # postfix[i] = ' '
      addi $s1, $s1, 1                  # i++
      j consider_mul_div
      nop
```

- Function handling whenever opening parenthesis is encountered

```
consider_lpar:
      addi $a3, $a3, 1                  # lparcount++
      addi $t3, $s0, 1                  # $t3 = j++
      lb $t4, infix($t3)                # $t4 = infix[j]
      beq $t4, '-', lt0_error           # if infix[j] == '-' -> negative value -> branch to error elert
      nop
      j     push_op                           # else push to stack
      nop
```

- Function handling whenever closing parenthesis is encountered

```
consider_rpar1:
       addi $a3, $a3, -1                  # lparcount--
       j      consider_rpar
       nop
consider_rpar:
       beq $s2, -1, push_op        # if stack is null, push right parentheses to stack
       nop
       lb $t1, stack($s2)          # else pop operand out of stack
       sb $t1, postfix($s1)        # postfix[i] = $t1
       addi $s2, $s2, -1                  # k--
       addi $s1, $s1, 1                   # i++
       beq $t1, '(', push_op              # until get '(' then push to stack
       j consider_rpar
       nop
```

- Push operator, operand to stack

```
push_op:
       addi $s2, $s2, 1                   # k++
       sb $t0, stack($s2)                 # stack[k] = $t0
       addi $s0, $s0, 1                   # j--
       j loop_infix
       nop
# pop the other operators of stack then push to postfix
end_loop_infix:
       beq $s2, -1, remove_parentheses
       nop
       lb $t0, stack($s2)          # $t0 = stack[k]
       sb $t0, postfix($s1)        # postfix[i] = $t0
       addi $s2, $s2, -1                  # k--
       addi $s1, $s1, 1                   # i++
       li $s3, ' '
       sb $s3, postfix($s1)        # postfix[i] = ' '
       addi $s1, $s1, 1                   # i++
       j end_loop_infix
       nop
# remove parentheses procedure
       li $s6, 0                          # index of postfix
       li $s7, 0                          # index of postfix_
remove_parentheses:
       lb $t5, postfix($s6)        # $t5 = postfix[i]
       addi $s6, $s6, 1                   # i++
       beq $t5, '(', remove_parentheses
```

```
275         nop
276         beq $t5, ')', remove_parentheses
277         nop
278         beq $t5, 0, print_postfix          # if get a space character -> branch to print postfix expression
279         nop
280         sb $t5, postfix_($s7)              # postfix_[j] = $t5
281         addi $s7, $s7, 1                   # j++
282         j remove_parentheses
283         nop
```

- Print postfix expression

```
284 # print postfix procedure
285 print_postfix:
286         bne $a3, 0, error1                 # if lparcount != 0 -> branch to error alert
287         nop
288         li $v0, 4
289         la $a0, msg_print_postfix          # address of message
290         syscall
291         li $v0, 4
292         la $a0, postfix_                   # address of final result
293         syscall
294         li $v0, 4
295         la $a0, msg_enter
296         syscall
297         j calculate_postfix
298         nop
299 error1:
300         li $v0, 55
301         la $a0, msg_error4
302         syscall
303         li $v0, 10                         # exit
304         syscall
```

- Calculate the postfix expression

```
314 calculate_postfix:
315         li $s1, 0                          # i = 0
316 loop_postfix:
317         lb $t0, postfix_($s1)              # $t0 = postfix[i]
318         beq $t0, $0, print_result          # if $t0 = '\0' -> branch to print result
319         nop
320         beq $t0, ' ', remove_space
321         nop
322 # if character is number -> check the next character
323         beq $t0, '0', next
324         nop
325         beq $t0, '1', next
326         nop
327         beq $t0, '2', next
328         nop
329         beq $t0, '3', next
330         nop
331         beq $t0, '4', next
332         nop
333         beq $t0, '5', next
334         nop
335         beq $t0, '6', next
336         nop
337         beq $t0, '7', next
338         nop
339         beq $t0, '8', next
340         nop
```

```
        beq $t0, '9', next
nop
operand:
        lw $t6, -8($sp)                 # load the first value (a) from stack pointer
        lw $t7, -4($sp)                 # load the second value (b) from stack pointer
        addi $sp, $sp, -8
        beq $t0, '+', add_func          # if $t0 = '+' -> branch to add function
        nop
        beq $t0, '-', sub_func          # if $t0 = '-' -> branch to sub function
        nop
        beq $t0, '*', mul_func          # if $t0 = '*' -> branch to mul function
        nop
        beq $t0, '%', mod_func          # if $t0 = '%' -> branch to mod function
        nop
        beq $t0, '/', div_func          # if $t0 = '/' -> branch to div function
        nop
        addi $s1, $s1, 1                # i++
        j loop_postfix
# if get a space then remove it
remove_space:
        addi $s1, $s1, 1                # i++
        j loop_postfix
        nop
```

```
next:
       addi $s1, $s1, 1                 # i++
       lb $t2, postfix_($s1)            # $t2 = postfix_[i]
       beq $t2, '0', push_number
       nop
       beq $t2, '1', push_number
       nop
       beq $t2, '2', push_number
       nop
       beq $t2, '3', push_number
       nop
       beq $t2, '4', push_number
       nop
       beq $t2, '5', push_number
       nop
       beq $t2, '6', push_number
       nop
       beq $t2, '7', push_number
       nop
       beq $t2, '8', push_number
       nop
       beq $t2, '9', push_number
       nop
       addi $t0, $t0, -48               # if number < 10 then convert character from char to number
       sw $t0, 0($sp)
       addi $sp, $sp, 4
       j loop_postfix
```

```
        nop
push_number:
        addi $t0, $t0, -48                  # convert character from char to number
        addi $t2, $t2, -48                  # convert character from char to number
        mul $t3, $t0, 10
        add $t3, $t3, $t2                   # $t3 = 10 * $t0 + $t2
        sw $t3, 0($sp)                      # $sp = $t3
        addi $sp, $sp, 4
        addi $s1, $s1, 1                    # i++
        j loop_postfix
```

- Function to add, sub, mul, div and mod

```
add_func:
      add $t6, $t6, $t7                 # $t6 = $t6 + $t7
      sw $t6, 0($sp)                    # $sp = $t6
      addi $sp, $sp, 4
      addi $s1, $s1, 1                  # i++
      j loop_postfix
      nop
```

```
sub_func:
        sub $t6, $t6, $t7                   # $t6 = $t6 - $t7
        sw $t6, 0($sp)                      # $sp = $t6
        addi $sp, $sp, 4
        addi $s1, $s1, 1                    # i++
        j loop_postfix
        nop

mul_func:
        mul $t6, $t6, $t7                    # $t6 = $t6 * $t7
        sw $t6, 0($sp)                       # $sp = $t6
        addi $sp, $sp, 4
        addi $s1, $s1, 1                     # i++
        j loop_postfix
```

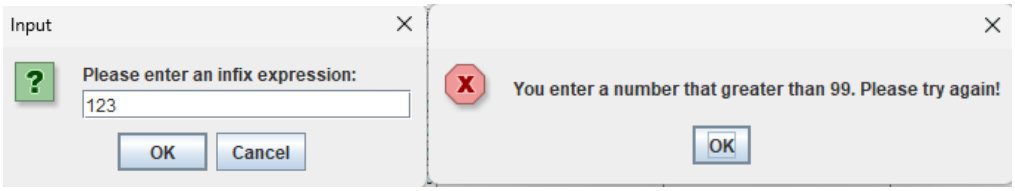
```
mod_func:
        beq $t7, 0, invalid_divisor         # if the divisor == 0 -> branch to error elert
        nop
        div $t6, $t7                        # $t9 = $t6 % $t7
        mfhi $t9
        sw $t9, 0($sp)                      # $sp = $t6
        addi $sp, $sp, 4
        addi $s1, $s1, 1                    # i++
        j loop_postfix
        nop


div_func:
        beq $t7, 0, invalid_divisor         # if the divisor == 0 -> branch to error elert
        nop
        div $t6, $t6, $t7                   # $t6 = $t6/$t7
        sw $t6, 0($sp)                      # $sp = $t6
        addi $sp, $sp, 4
        addi $s1, $s1, 1                    # i++
        j loop_postfix
        nop
invalid_divisor:
        li $v0, 55
        la $a0, msg_error3
        syscall
        #j input_infix
        li $v0, 10                          # exit
        syscall
```

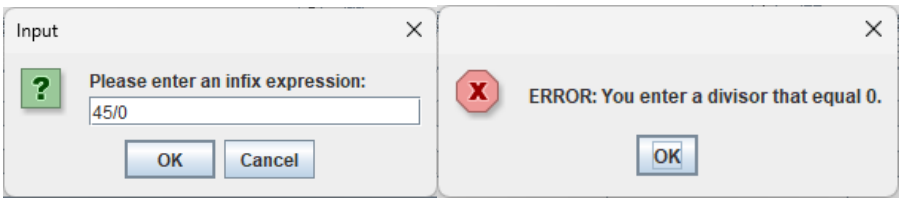
3. Demo project

a. Exception cases

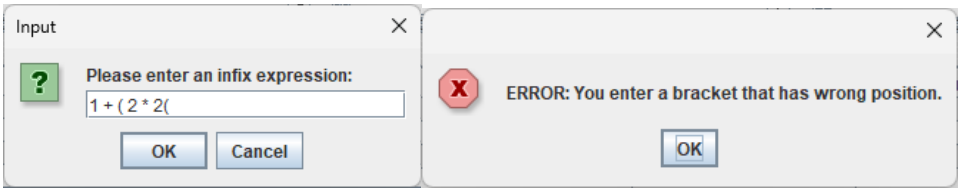
- Input is out of range 0-100



- Divide by 0



- Parenthesis



b. Results

9 + 2 + 8 * 6

```
Infix expression: 9 + 2 + 8 * 6
Postfix expression: 9 2 + 8 6 * +
Result of the expression: 59
```

(22 + (10 – 4)) * ((11 – 5) / (4 + 3 -2))

```
Infix expression: (22+(10-4))*((11-5)/(4+3-2))
Postfix expression: 22 10 4 -+ 11 5 - 4 3 + 2 -/ *
Result of the expression: 28
```

1+2-3*4+5*6-7%8

```
Infix expression: 1 + 2 - 3 * 4 + 5 * 6 - 7 % 8
Postfix expression: 1 2 + 3 4 * - 5 6 * + 7 8 % -
Result of the expression: 14
```

12 + (12 * (3+4) - (5*6-7)) + 5 % 2

```
Infix expression: 12 + (12 * (3+4) - (5*6-7)) + 5 % 2
Postfix expression: 12 12 3 4 + * 5 6 * 7 -- + 5 2 % +
Result of the expression: 74
```

(79 73) / 2 + (39 /13)*(98-87)

```
Infix expression: (79 + 73) / 2 + (39 /13)*(98-87)
Postfix expression: 79 73 + 2 / 39 13 / 98 87 - * +
Result of the expression: 109
```

Task 2: Moving a ball on the BITMAP Display

1. Problem introduction

- Create a program that display a movable and bouncing ball on the bitmap screen. If the ball hit an edge, it will move in the opposite direction.

- The movement of the ball depends on the key pressed from keyboard (W for moving up, S for moving down, D for moving left, A for moving right, Z for speeding up and X for slowing down)

2. Project implementation

2.1 Main idea for problem

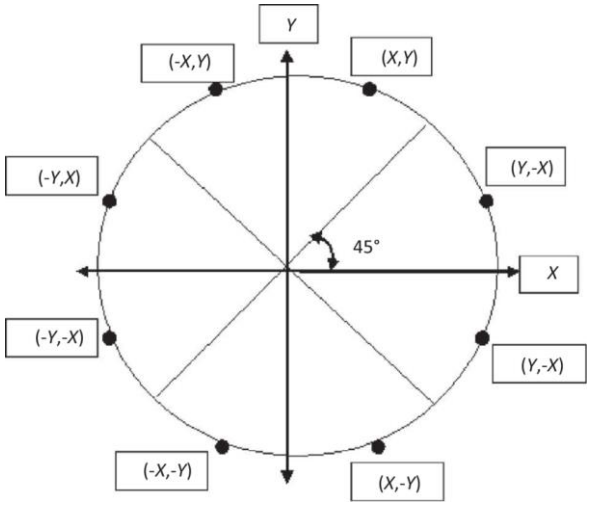
- Using Keyboard and Display MIMO Simulator to detect the key pressed.
- The movement of the ball can be represented by drawing a circle in the new position and draw another circle with the same color with screen in the recent position.

2.2 Detail algorithm

- Drawing circle: Setup initial value for center of the circle (x, y), radius (R), moving distance of the circle and sleep time of each movement.

  . Create a pointer to save data about all the point in the circle in circle array.

  . Loop the value x-coordinate from 0 to R, calculate the value y-coordinate by the formular y = $\sqrt{R^2 - x^2}$. Because x and y are positive, so we also create point (-x, y); (x, -y); (-x, -y) and then we also swap value of x and y so we have 8 point to draw a circle. Save these points to array and we finished data of the circle.



- Check if the ball touch to the edge of the screen. If the coordinate of the center plus R is greater than the bounded value of the screen (0 for lower bound and 511 for upper bound), the ball will move in the opposite direction.
- Draw a new circle: Delete the old circle by change the color of the recent circle to black. Change the color to yellow, update new value for point in the circle and redraw.

2.3 Code explanation

n02_BuiDucViet_20215254.asm*

```
7   circle_end:     .word   1       # Pointer point to the end of the cycle
8   circle:         .word           # Pointer point to the array storing data of the cycle
9   .text
10  initial_declare:
11          addi    $s0, $0, 255    # x = 255
12          addi    $s1, $0, 255    # y = 255
13          addi    $s2, $0, 0      # dx = 0
14          add     $s3, $0, $0     # dy = 0
15          addi    $s4, $0, 20     # r = 20
16          addi    $a0, $0, 50     # t = 50ms/frame
17          jal     circle_data
```

- This code is used to setup initial value for some variable: $s0 is x-coordinate of center, $s1 is y-coordinate of the center, initial setup is the center of the screen (255,255)
- $s2 (dx), $s3 (dy) is the movement distance along x and y axis. (equal 0 at first)
- $s4 is the radius of the circle, $a0 is the sleeping time.

```
19  input:
20          li      $k0, KEY_READY  # Check whether there is input data
21          lw      $t0, 0($k0)
22          bne     $t0, 1, edge_check
23          jal     direction_change
```

- Check the input from the keyboard. If there is a key pressed, program will move to ‘edge_check’ part to check if the ball hit the edges.

```
26  edge_check:
27
28  right:
29          bne     $s2, 1, left
30          j       check_right
31
32  left:
33          bne     $s2, -1, down
34          j       check_left
35
36  down:
37          bne     $s3, 1, up
38          j       check_down
39
40  up:
41          bne     $s3, -1, move_circle
42          j       check_up
43
```

```
181 check_right:
182         add     $t0, $s0, $s4   # Set $t0 to the right side of the circle
183         beq     $t0, 511, reverse_direction     # Reverse direction if the side has touched the edge
184         j       move_circle     # Return if not
185
186 check_left:
187         sub     $t0, $s0, $s4   # Set $t0 to the left side of the circle
188         beq     $t0, 0, reverse_direction       # Reverse direction if the side has touched the edge
189         j       move_circle     # Return if not
190
191 check_down:
192         add     $t0, $s1, $s4   # Set $t0 to the down side of the circle
193         beq     $t0, 511, reverse_direction     # Reverse direction if the side has touched the edge
194         j       move_circle     # Return if not
195
196 check_up:
197         sub     $t0, $s1, $s4   # Set $t0 to the up side of the circle
198         beq     $t0, 0, reverse_direction       # Reverse direction if the side has touched the edge
199         j       move_circle     # Return if not
```

- First, check the direction of the movement. If the ball is moving to the left, only check with the left-bound, if the ball is moving to the right, only check with the right-bound. Do the same way with the case of moving up and down.

- In the check function (check_right, check_left, check_up, check_down), create a temporary variable to store the value of the right-most point of the circle (coordinate of center + radius) to check.

```
move_circle:
        add     $s5, $0, $0     # Set color to black
        jal     draw_circle     # Erase the old circle

        add     $s0, $s0, $s2   # Update new value for center point of the cycle
        add     $s1, $s1, $s3
        li      $s5, 0x00FFFF00 # Set color to yellow
        jal     draw_circle     # Draw the new circle

```

- Code for moving the circle: set color to the black and draw a circle. Next, update the new center, change the color to yellow and redraw a new circle.

```
circle_data:
        addi    $sp, $sp, -4    # Save $ra
        sw      $ra, 0($sp)
        la      $s5, circle     # $s5 becomes the pointer of the "circle" array
        mul     $a3, $s4, $s4   # $a3 = r^2
        add     $s7, $0, $0     # pixel x (px) = 0

pixel_data_loop:
        bgt     $s7, $s4, data_end
        mul     $t0, $s7, $s7   # $t0 = px^2
        sub     $a2, $a3, $t0   # $a2 = r^2 - px^2 = py^2
        jal     root            # $a2 = py

        add     $a1, $0, $s7    # $a1 = px
        add     $s6, $0, $0     # After saving (px, py), (-px, py), (-px, -py), (px, -py), we swap px and py, then save (-py, px), (py, px), (py, -px), (-py, -px)

```

```
symmetric:
        beq     $s6, 2, finish
        jal     pixel_save      # px, py >= 0
        sub     $a1, $0, $a1
        jal     pixel_save      # px <= 0, py >= 0
        sub     $a2, $0, $a2
        jal     pixel_save      # px, py <= 0
        sub     $a1, $0, $a1
        jal     pixel_save      # px >= 0, py <= 0

        add     $t0, $0, $a1    # Swap px and -py
        add     $a1, $0, $a2
        add     $a2, $0, $t0
        addi    $s6, $s6, 1
        j       symmetric

finish:
        addi    $s7, $s7, 1
        j       pixel_data_loop

data_end:
        la      $t0, circle_end
        sw      $s5, 0($t0)     # Save the end address of the "circle" array
        lw      $ra, 0($sp)
        addi    $sp, $sp, 4
        jr      $ra
```

- This paragraph is used to store data of all point in a circle. First, assign $s7 (px) = 0 to loop from 0 to R.
- In the pixel_data_loop block: calculate $a2 (py) by the root block. Storing 7 other points by change sign and swap value of px and py to the circle array.
- The storing end when px increase to greater than R.

```
134 pixel_save:
135         sw      $a1, 0($s5)     # Store px in the "circle" array
136         sw      $a2, 4($s5)     # Store py in the "circle" array
137         addi    $s5, $s5, 8     # Move the pointer to a null block
138         jr      $ra
```

- Function for save the pixel x and y.

```
140 direction_change:
141         li      $k0, KEY_CODE
142         lw      $t0, 0($k0)
143
144 case_d:
145         bne     $t0, 'd', case_a
146         addi    $s2, $0, 1      # dx = 1
147         add     $s3, $0, $0     # dy = 0
148         jr      $ra
149
150 case_a:
151         bne     $t0, 'a', case_s
152         addi    $s2, $0, -1     # dx = -1
153         add     $s3, $0, $0     # dy = 0
154         jr      $ra
155
156 case_s:
157         bne     $t0, 's', case_w
158         add     $s2, $0, $0     # dx = 0
159         addi    $s3, $0, 1      # dy = 1
160         jr      $ra
```

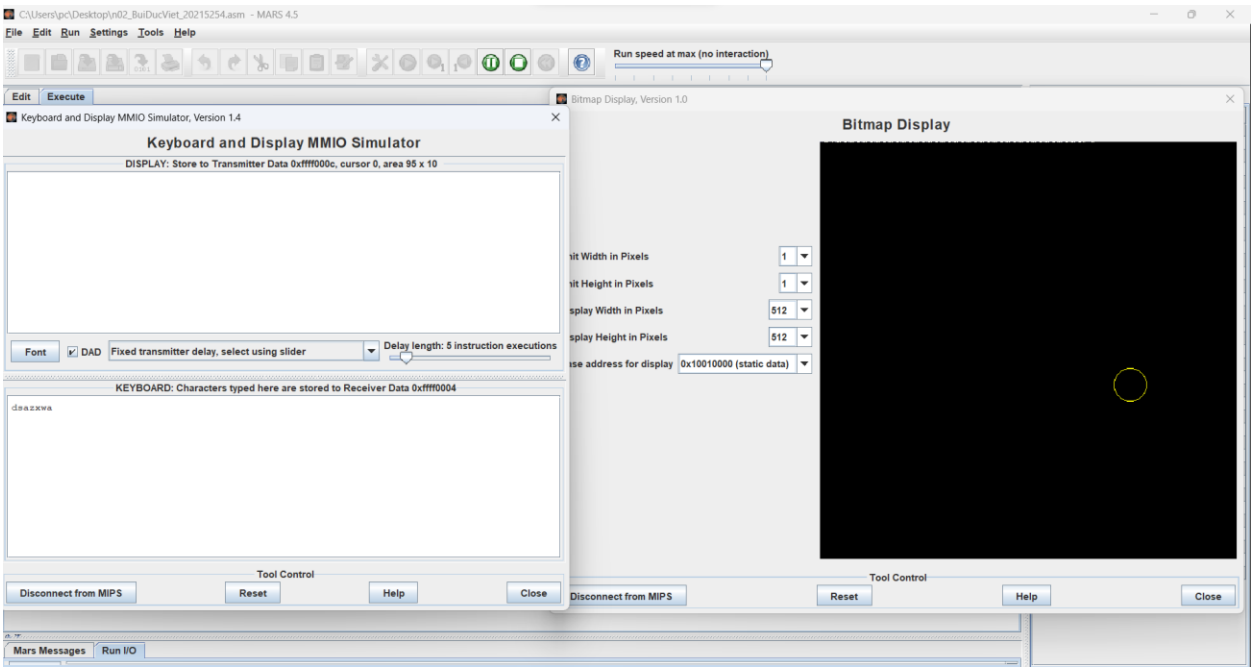
```
162 case_w:
163         bne     $t0, 'w', case_x
164         add     $s2, $0, $0     # dx = 0
165         addi    $s3, $0, -1     # dy = -1
166         jr      $ra
167
168 case_x:
169         bne     $t0, 'x', case_z
170         addi    $a0, $a0, 10    # t += 10
171         jr      $ra
172
173 case_z:
174         bne     $t0, 'z', default
175         beq     $a0, 0, default # Only reduce t when t >= 0
176         addi    $a0, $a0, -10   # t -= 10
177
178 default:
179         jr      $ra
180
```

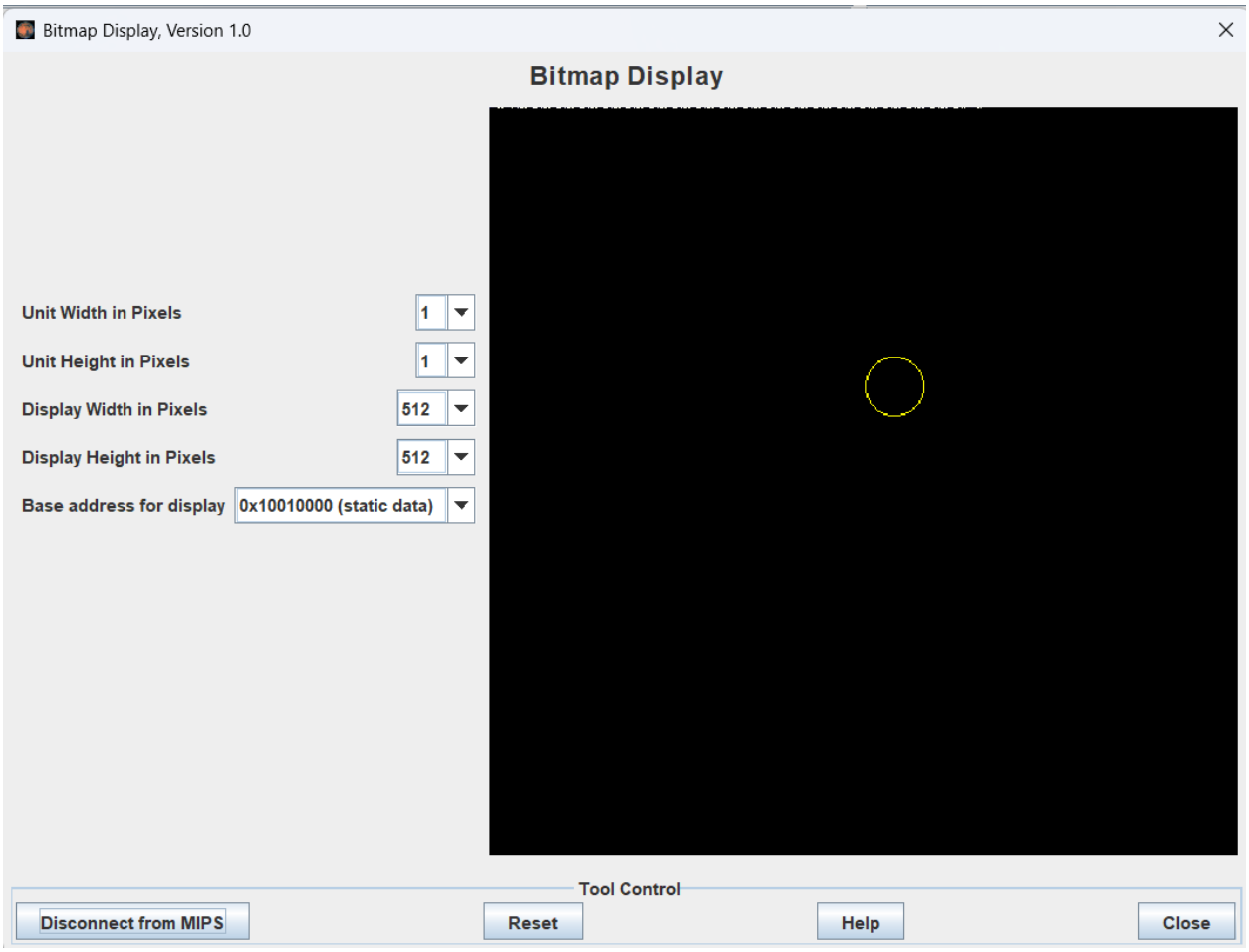
- Check the key pressed from keyboard to update the movement of the circle by updating the moving distance $s2 (dx), $s3(dy). If the option is ‘x’ or ‘z’, update the value of sleep time.

```
206 draw_circle:
207         addi    $sp, $sp, -4    # Save $ra
208         sw      $ra, 0($sp)
209         la      $s6, circle_end
210         lw      $s7, 0($s6)     # $s7 becomes the end address of the "circle" array
211         la      $s6, circle     # $s6 becomes the pointer to the "circle" array
212
213 draw_loop:
214         beq     $s6, $s7, draw_end      # Stop when $s6 = $s7
215         lw      $a1, 0($s6)             # Get px
216         lw      $a2, 4($s6)             # Get py
217         jal     pixel_draw
218         addi    $s6, $s6, 8             # Get to the next pixel
219         j       draw_loop
220
221 draw_end:
222         lw      $ra, 0($sp)
223         addi    $sp, $sp, 4
224         jr      $ra
225
226 pixel_draw:
227         li      $t0, SCREEN_MONITOR
228         add     $t1, $s0, $a1           # final x (fx) = x + px
229         add     $t2, $s1, $a2           # fy = y + py
230         sll     $t2, $t2, 9             # $t2 = fy * 512
231         add     $t2, $t2, $t1           # $t2 += fx
232         sll     $t2, $t2, 2             # $t2 *= 4
233         add     $t0, $t0, $t2
234         sw      $s5, 0($t0)
235         jr      $ra
```

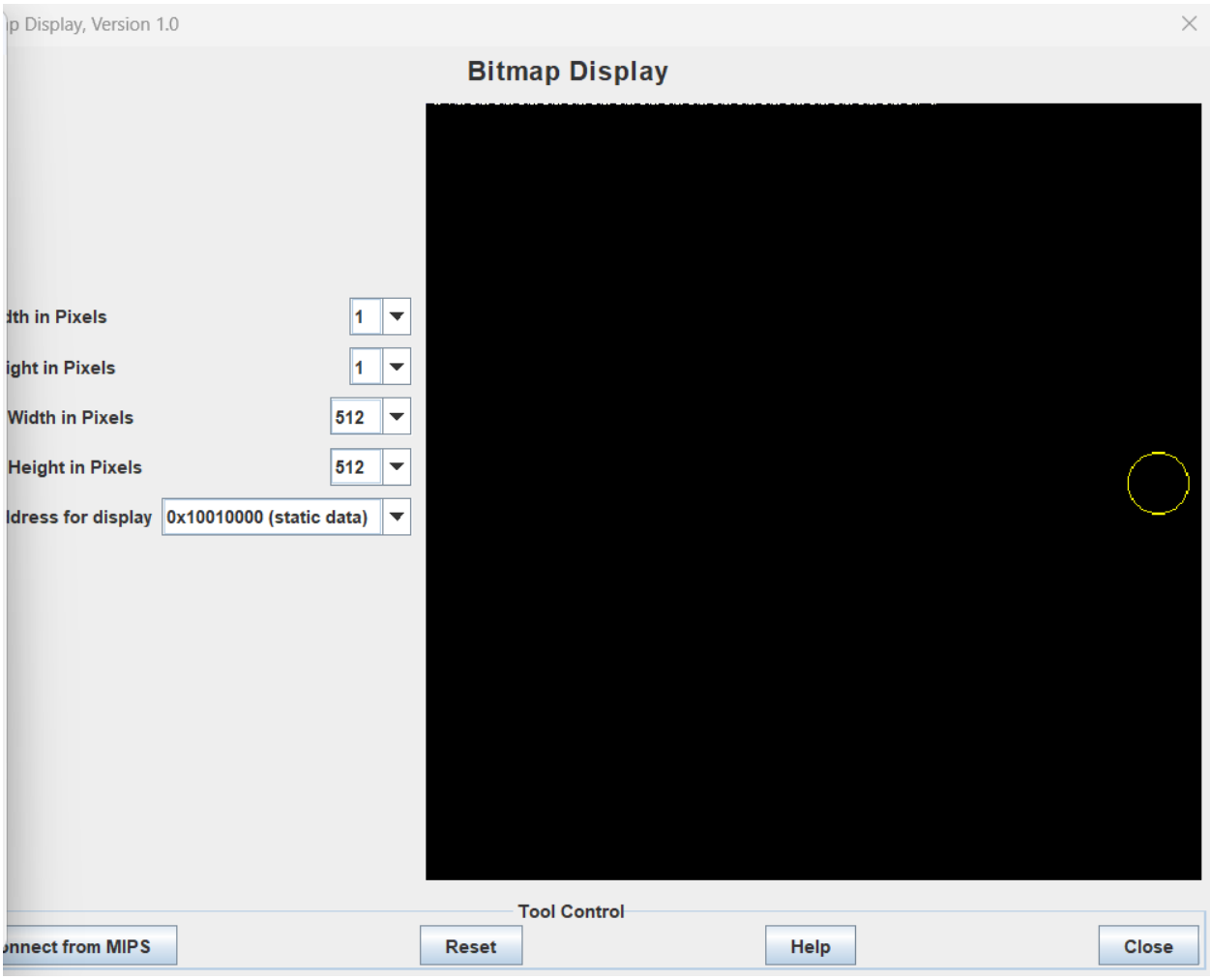
- Store the address of the start of circle array $s6 and address of the end of circle array $s7. Move to the block ‘pixel_draw’.
- Loop for all point, get 2 temporary variables $a1 to store px, $a2 to store py. Calculate the position of the pixel by 2 coordinate px, py. Draw this pixel with color.

3. Demo project





- BITMAP Display screen



- The ball bounces to the opposite direction when hit the edges.