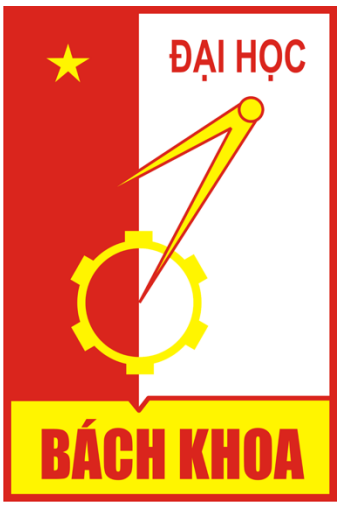
HANOI UNIVERSITY OF SCIENCE & TECHNOLOGY

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
Ngo Xuan Bach	ICT 01 K66	20215181
Nguyen Chi Long	ICT 02 K66	20210553

Instructor: MSc. Le Ba Vui

Teaching Assistant: Do Gia Huy

Hanoi, January 2024

Table of Contents

I. 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. Project implementation

a. Algorithm

- *Initial setup preprocessing:*

Step 1: Set up (X0,Y0) = (256,256) which is the center of bitmap screen, R = 16 (radius of the ball), deltaX = deltaY = 0 (no moving at start), speed = 2 (initial speed).

Step 2: Calculate relative location with respect to the center of the circle of points in the circle with formula (Xi,Yi) = (X0+i, Y0+sqrt(R^2 -i^2)). And store into array "circle_bound".

- *Keep track of user input and edges check:*

Step 3: Read the keyboard inputted by the user:

- If user inputted "a" set deltaX = - speed , deltaY = 0.

- If user inputted "d" set deltaX = speed, deltaY = 0.

- If user inputted "s" set deltaY = speed, deltaX = 0.

- If user inputted "w" set deltaY = -speed, deltaX = 0

- If user inputted "z" set speed = speed + 1 (add $s7, $s7, 1)

- If user inputted "x" set speed = speed - 1 (add $s7, $s7,-1) and also check the speed <= 0 or not with minimum value for speed is 1.

- Otherwise, keep the program running the same direction as before.

Step 4: Check whether the ball hit the edges or not by formula (0,0) < (X,Y) + R + (deltaX,deltaY) < 512. If hit the edges, reverse the direction of the ball, else keep running as before.

- *Draw the ball:*

Step 5: overwrite the previous circle using BLACK color.

Step 6: Move the center of the circle to next location with (X = X + deltaX, Y = Y + deltaY).

Step 7: Draw a circle in the new position with color YELLOW.

Step 8: Jump back to step 2.

- *Drawing the circle:*
  For each index and value in "circle_bound" which is (Xi,Yi) of the circle we have calculated, we draw in 4 quarters of the circle. In each quarter we draw two points. For example, in first quarter, we draw (Xo + i, Yo + j), (Xo + j, Yo + i) which is valid because "i" and "j" is exchangeable. Similarly for other quarter.

b. Code explanation

- Data section and define section:

- SCREEN: the bitmap screen.
- YELLOW, BLACK: colors for displaying the ball.
- KEY_A, KEY_S, KEY_D, KEY_W, KEY_Z: valid key press.
- KEY_CODE: address of user pressed key.
- KEY_READY: address of value indicate user pressed key.

```
1  .eqv SCREEN       0x10010000
2  .eqv YELLOW       0x00FFFF00
3  .eqv BLACK  0x00000000
4  .eqv KEY_A  0x00000061
5  .eqv KEY_S  0x00000073
6  .eqv KEY_D  0x00000064
7  .eqv KEY_W  0x00000077
8  .eqv KEY_Z        0x0000007A
9  .eqv KEY_X        0x00000078
10 .eqv KEY_CODE     0xFFFF0004
11 .eqv KEY_READY    0xFFFF0000
12
13 .data
14       circle_bound:     .space 100
15 # $a2, $a3: loop's iterators
```

- **Preprocess:** Setup initial values and preprocessing the circle points**.**

```
# $a2, $a3: loop's iterators
.text
      li $s0, 256       # X_0 = 256
      li $s1, 256       # Y_0 = 256
      li $s2, 16        # R = 16
      li $s3, 512       # width of screen
      li $s4, 512       # height of screen
      li $s5, 0         # deltaX = 0
      li $s6, 0         # deltaY = 0
      li $s7, 2         # speed

circle_bound_init: # use $t0, $t1, $t2, $t3, no need to keep
      la $t1, circle_bound
      li $a2, 0
      mul $t2, $s2, $s2 # $t2 = R^2
circle_bound_loop:
      ble $s2, $a2, end_circle_bound_loop
      mul $t3, $a2, $a2
      sub $t3, $t2, $t3 # $t3 = R^2-i^2
      move $t0, $t3 # $t0 = sqrt(R^2-i^2)
      jal sqrt
      sw $t0, 0($t1)
      add $a2, $a2, 1
      add $t1, $t1, 4
      j circle_bound_loop
end_circle_bound_loop:
```

- **Keep track of user input:** Check user input and jump to correct function to update direction of speed of the ball.

```
game_loop:
read_keyboard:
     lw $k1, KEY_READY                    # if a key is clicked, $k1 = 1
     beq $k1, $zero, position_check     # $k1 = 0 then run position check
     lw $k0, KEY_CODE
     beq $k0, KEY_A, case_a
     beq $k0, KEY_S, case_s
     beq $k0, KEY_D, case_d
     beq $k0, KEY_W, case_w
     beq $k0, KEY_Z, case_z
     beq $k0, KEY_X, case_x
     j position_check
case_a:
     jal move_left
     j position_check
case_s:
     jal move_down
     j position_check
case_d:
     jal move_right
     j position_check
case_w:
     jal move_up
     j position_check
case_z:
     jal speed_up
     j position_check
case_x:
     jal slow_down

```

```
# Note: in speed_up, we are based on the direction to call move again, so that the speed are immediately changed
move_left:
      sub $s5, $zero, $s7 # move left $s7 unit
      li $s6, 0
      jr $ra
move_right:
      add $s5, $zero, $s7 # move right $s7 unit
      li $s6, 0
      jr $ra
move_up:
      li $s5, 0
      sub $s6, $zero, $s7 # move up $s7 unit
      jr $ra
move_down:
      li $s5, 0
      add $s6, $zero, $s7 # move down $s7 unit
      jr $ra
speed_up: # $s7 += 1
      add $s7, $s7, 1

      jr $ra
slow_down: # $s7 -= 1, min speed: 1
      add $s7, $s7,-1
      blt $zero, $s7, back
      li $s7, 1
back:
      jr $ra
```

- **Edges hit checking:** Check whether the ball hit the edges or not. If hit reverse the ball direction, else keep the previous direction

```

# Check if the circle touches an edge or not. After checking 4 directions, draw the circle
position_check: # use $t0, no need to keep
check_right:
      add $t0, $s0, $s2
      add $t0, $t0, $s5       # $t0 = X_0 + R + deltaX: rightest point on the circle after this time step
      ble $t0, $s3, check_left # if $t0 < width of screen, no need to check more. If not, reverse the direction
      jal move_left
check_left:
      sub $t0, $s0, $s2
      add $t0, $t0, $s5       # $t0 = X_0 - R + deltaX: leftest point on the circle after this time step
      ble $zero, $t0, check_top# if 0 < $t0, no need to check more. If not, reverse the direction
      jal move_right
check_top:
      sub $t0, $s1, $s2
      add $t0, $t0, $s6       # t2 = Y_0 - R + deltaY: highest point on the circle after this time step
      ble $zero, $t0, check_bottom # if 0 < $t0, no need to check more. If not, reverse the direction
      jal move_down
check_bottom:
      add $t0, $s1, $s2
      add $t0, $t0, $s6       # t2 = Y_0 + R + deltaY: highest point on the circle after this time step
      ble $t0, $s4, draw            # if $t0 < height of screen, no need to check more. If not, reverse the direction
      jal move_up
```

- **Draw the ball:**
  - Overdraw the previous location: Draw the old location with color BLACK and then update the next location.

```
draw: # use $t0, $t1, no need to keep $t1 for child
      la $t0, BLACK
      jal draw_circle
      add $s0, $s0, $s5
      add $s1, $s1, $s6 # 2 lines: move the center to the new position after the time step: X = X + deltaX, Y = Y + deltaY
```

  - Draw the new location: Draw the old location with color YELLOW.

```
100         la $t0, YELLOW
101         jal draw_circle
102         li $v0, 32
103         li $t1, 50
104         syscall # Stop for a while: 50ms
105         j game_loop
106
```

- **Draw_circle function:** Loop through all pairs of relative position with respect to the center which have been stored in circle_bound. Draw the points in 4 quarters.

```
109 draw_circle: # use $t1, $t2, $t3, $t4, no need to keep
110         add $t9, $0, $ra #Luu lai gia tri cua $ra
111         la $t1, circle_bound
112         li $a2, 0
113 draw_circle_loop: # $t2 = circle_bound[i]
114         ble $s2, $a2, end_draw_circle_loop
115         lw $t2, 0($t1)
116
117         move $t3, $a2                   # i = $a0 = $t0(index cua mang)
118         move $t4, $t2                   # j = $a1
119         jal drawCirclePoint             # Lay toa do de ve (Xo + i, Yo + j), (Xo + j, Yo + i)
120         sub $t4, $zero, $t2
121         jal drawCirclePoint             # (Xo + i, Yo - j), (Xo + j, Yo - i)
122         sub $t3, $zero, $a2
123         jal drawCirclePoint             # (Xo - i, Yo - j), (Xo - j, Yo - i)
124         add $t4, $zero, $t2
125         jal drawCirclePoint             # (Xo - i, Yo + j), (Xo - j, Yo + i)
126
127         add $a2, $a2, 1
128         add $t1, $t1, 4
129         j draw_circle_loop
130 end_draw_circle_loop:
131         add $ra, $t9, $0 # tra lai gia tri $ra
132         jr $ra
```

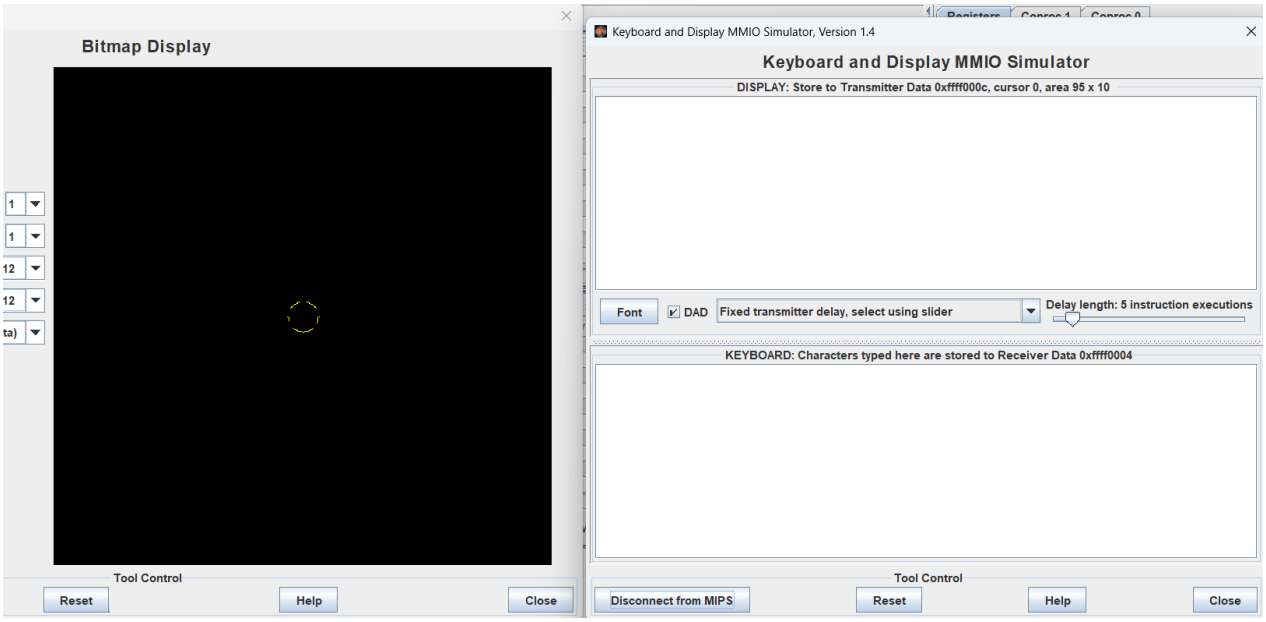
  - In each quarters, draw 2 points:

```
134 drawCirclePoint:
135
136         add $t5, $s0, $t3       # Xi = X0 + i
137         add $t6, $s1, $t4 # Yi = Y0 + j
138         mul $t6, $t6, $s3 # Yi * SCREEN_WIDTH
139         add $t5, $t5, $t6 # Yi * SCREEN_WIDTH + Xi (Toa do 1 chieu cua diem anh)
140         sll $t5, $t5, 2         # Dia chi tuong doi cua diem anh
141         sw $t0, SCREEN($t5)     # Ve diem anh
142         add $t5, $s0, $t4       # Xi = Xo + j
143         add $t6, $s1, $t3 # Yi = Y0 + i
144         mul $t6, $t6, $s3 # Yi * SCREEN_WIDTH
145         add $t5, $t5, $t6 # Yi * SCREEN_WIDTH + Xi (Toa do 1 chieu cua diem anh)
146         sll $t5, $t5, 2         # Dia chi tuong doi cua diem anh
147         sw $t0, SCREEN($t5)     # Ve diem anh
148
149         jr $ra
150
```

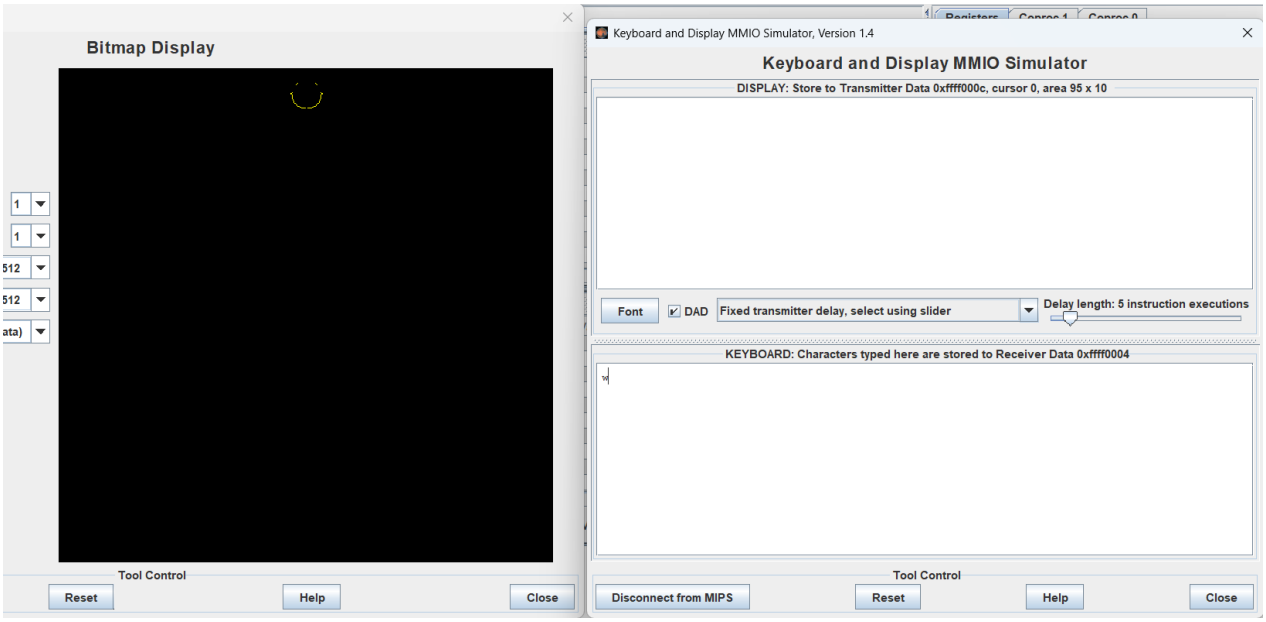
3. Results demonstration

The demonstration results is not sufficient to this task, I will represent it directly to you when we met up for reporting.

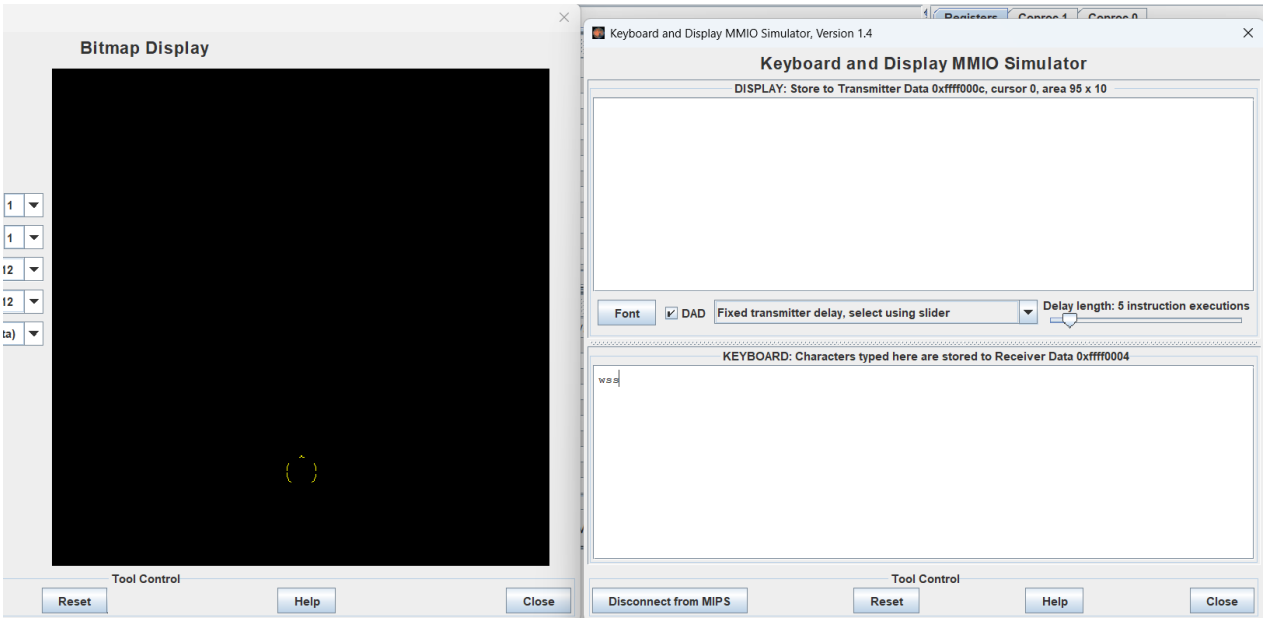
- **Starting state:**



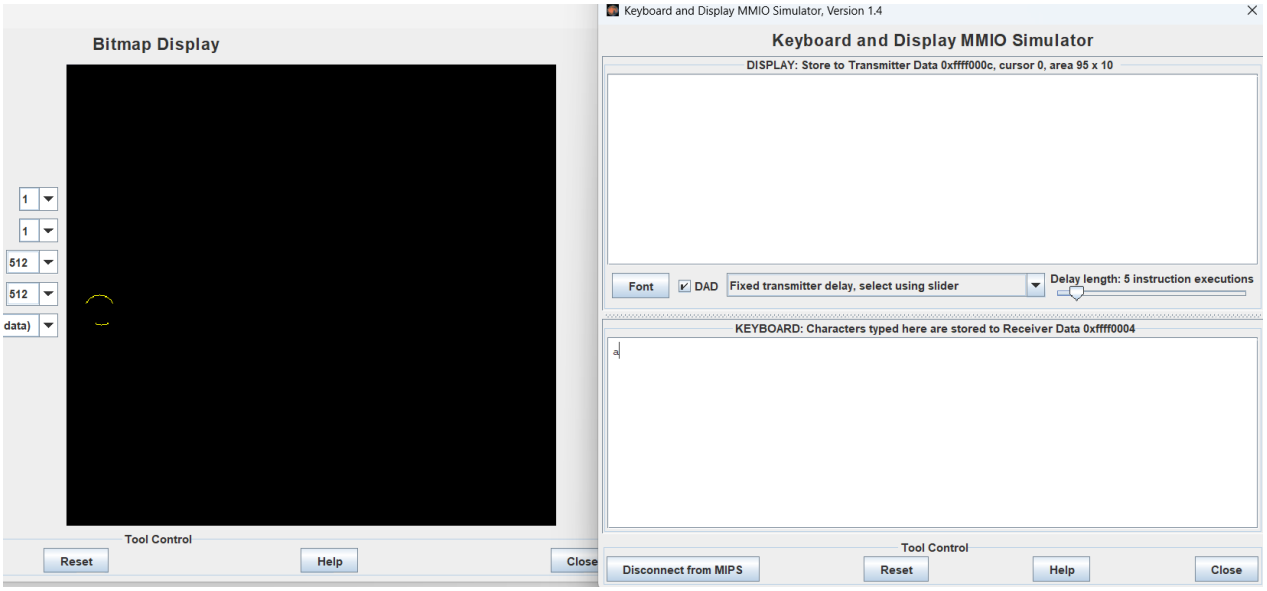
- **Move up:**



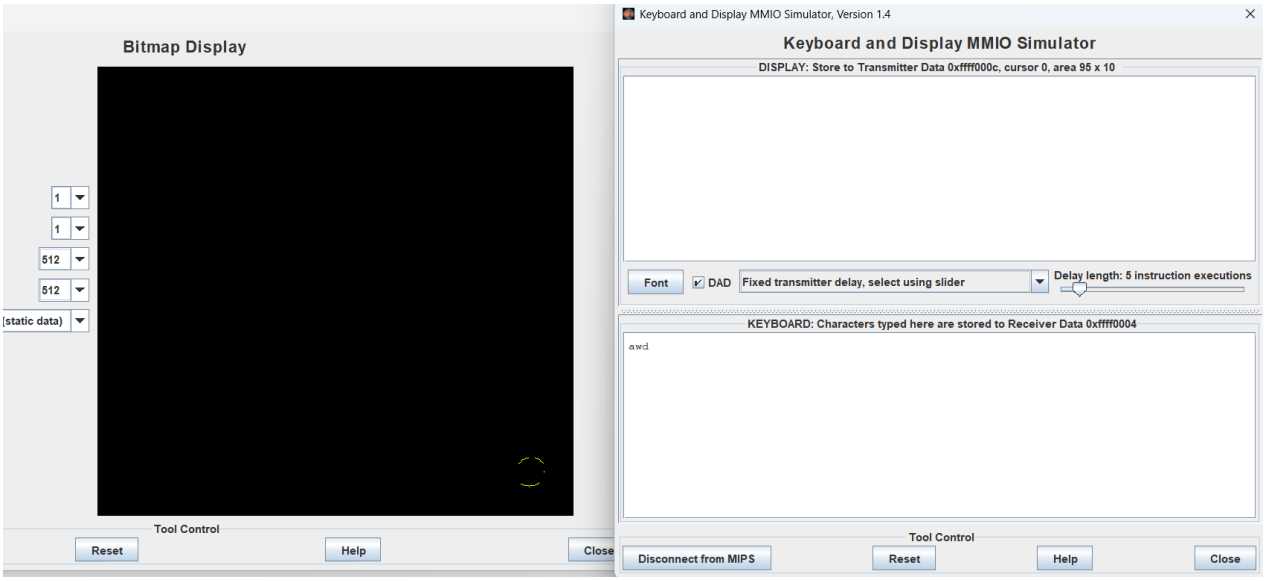
- **Move down:**



- **Move left:**



- **Move right**



II. Task 8: RAID 5 simulation

1. Problem description

The RAID5 drive system requires at least 3 hard disks, in which parity data will be stored on 3 drives as shown below. Write a program to simulate the operation of RAID 5 with 3 drives, assuming each data block has 4 characters. The interface is as shown in the example below. Limit the length of the input string to a multiple of 8.

In this example, a string is entered from the keyboard (DCE.****ABCD1234HUSTHUST) will be divided in to blocks of 4 byte. First 4 bytes “DCE.” stored in Disk 1, next 4 bytes “****” stored Disk 2, data stored in Disk 3 is 4 parity bytes computed from 2 first blocks 6e = ’D’ xor ‘*’ ; 69 = ’C’ xor ‘*’; 6f = ’E’ xor ‘*’ ; 04 = ’.’ xor ‘*’

```
Nhap chuoi ki tu : DCE.****ABCD1234HUSTHUST
     Disk 1                Disk 2                Disk 3
 --------------        --------------        --------------
|      DCE.      |      |      ****      |      [[ 6e,69,6f,04]]
|      ABCD      |      [[ 70,70,70,70]]      |      1234      |
[[ 00,00,00,00]]      |      HUST      |      |      HUST      |
 --------------        --------------        --------------
```

2. Problem implementation

- *Initial setup preprocessing:*

Step 1: Set up the string objects for display, initialize string s and construct an array to hold their hex code equivalents for later utilization.

```
.data
inputMessage: .asciiz "Nhap chuoi ki tu : "
outputDisk: .asciiz "     Disk 1                Disk 2                Disk 3\n"
outputLine: .asciiz " --------------        --------------        --------------\n"
s: .space 1001
hex: .byte '0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'
errorMessage: .asciiz "Error: Length of the string must be divisible by 8\n"
```

Step 2: Print the inputMessage, and get input s with maximum length 1001.

```
input:
    li $v0, 4
    la $a0, inputMessage
    syscall
    li $v0, 8
    la $a0, s
    li $a1, 1001
    syscall
```

Step 3: Check if length of the input string is divisible by 8. If not, jump back to the input. Until when the string is valid, the first part of the output is printed.

```
# $s0: size of string s
    la $t0, s
    li $s0, 0
check_length_loop:
    lb $t1, ($t0)
    beq $t1, 10, after_check_length_loop
    add $t0, $t0, 1
    add $s0, $s0, 1
    j check_length_loop

after_check_length_loop:
    rem $t0, $s0, 8
    beq $t0, $zero, valid_string
    li $v0, 4
    la $a0, errorMessage
    syscall
    j input
valid_string:
    li $v0, 4
    la $a0, outputDisk
    syscall
    li $v0, 4
    la $a0, outputLine
    syscall
```

- *Print output:*

Step 4: For each 8 bits, we calculate the results of bit[i] xor bit[i+4] and store them into $t2, $t3, $t4 and $t5. We see that, there are 3 kinds of line of data block which are printed, so we check the remainder of the iterator when divided to 8, and choose the form to print based on it.

```
# $t2, $t3, $t4, $t5: xors of 4 pairs
    la $t0, s
    li $t1, 0
print_disk_memory_loop:
    beq $t1, $s0, after_print_disk_memory_loop

    lb $t6, 0($t0)
    lb $t7, 4($t0)
    xor $t2, $t6, $t7

    lb $t6, 1($t0)
    lb $t7, 5($t0)
    xor $t3, $t6, $t7

    lb $t6, 2($t0)
    lb $t7, 6($t0)
    xor $t4, $t6, $t7

    lb $t6, 3($t0)
    lb $t7, 7($t0)
    xor $t5, $t6, $t7

    rem $t6, $t1, 24
    beq $t6, 0, print_8_char1
    beq $t6, 8, print_8_char2
    beq $t6, 16, print_8_char3
```

Step 5: Print 8 characters in the form of the problem.

Here, we print the first block and it contains the first 4 bits count from address $t0. Here, 124 is the code of '|' in ASCII code.

```
print_8_char1:
    li $v0, 11
    li $a0, 124
    syscall
    jal print_5_spaces
    lb $t7, 0($t0)
    add $a0, $t7, $zero
    syscall
    lb $t7, 1($t0)
    add $a0, $t7, $zero
    syscall
    lb $t7, 2($t0)
    add $a0, $t7, $zero
    syscall
    lb $t7, 3($t0)
    add $a0, $t7, $zero
    syscall
    jal print_5_spaces
    li $a0, 124
    syscall

    jal print_6_spaces
```

The last 4 bits are printed with the same code.

```
94
95      li $a0, 124
96      syscall
97      jal print_5_spaces
98      lb $t7, 4($t0)
99      add $a0, $t7, $zero
100     syscall
101     lb $t7, 5($t0)
102     add $a0, $t7, $zero
103     syscall
104     lb $t7, 6($t0)
105     add $a0, $t7, $zero
106     syscall
107     lb $t7, 7($t0)
108     add $a0, $t7, $zero
109     syscall
110     jal print_5_spaces
111     li $a0, 124
112     syscall
113
114     jal print_6_spaces
```

The last block is printed with a different algorithm. After getting xor results in $t2, $t3, $t4, $t5, we need to convert it into 2 hex characters. We get the first char by shifting right 4 units, and get the second char by using AND operation with 0xf. After printing, we jump to after_print_8_char to continue the printing loop.

91 is the code of '[', 32 is the code of ' ', 44 is the code of ',', 93 is the code of ']' in ASCII.

Other functions print_8_char2 and print_8_char3's algorithm are the same, except the position of the blocks of code.

```
    li $a0, 91
    syscall
    li $a0, 91
    syscall

    li $a0, 32
    syscall

    srl $s2, $t2, 4
    and $s3, $t2, 0x0000000f

    lb $s4, hex($s2)
    move $a0, $s4
    syscall
    lb $s4, hex($s3)
    move $a0, $s4
    syscall

    li $a0, 44
    syscall

    srl $s2, $t3, 4
    and $s3, $t3, 0x0000000f
```

```
    lb $s4, hex($s2)
    move $a0, $s4
    syscall
    lb $s4, hex($s3)
    move $a0, $s4
    syscall

    li $a0, 44
    syscall

    srl $s2, $t4, 4
    and $s3, $t4, 0x0000000f

    lb $s4, hex($s2)
    move $a0, $s4
    syscall
    lb $s4, hex($s3)
    move $a0, $s4
    syscall

    li $a0, 44
    syscall

    srl $s2, $t5, 4
    and $s3, $t5, 0x0000000f
```

```
    lb $s4, hex($s2)
    move $a0, $s4
    syscall
    lb $s4, hex($s3)
    move $a0, $s4
    syscall

    li $a0, 93
    syscall
    li $a0, 93
    syscall

    li $a0, 10
    syscall
    j after_print_8_char
```

```
after_print_8_char:
    add $t0, $t0, 8
    add $t1, $t1, 8
    j print_disk_memory_loop

```

Step 6: Print the last part of output and terminate the program.

```
after_print_disk_memory_loop:
    li $v0, 4
    la $a0, outputLine
    syscall
    li $v0, 10
    syscall

```

- *Other functions:*

print_5_spaces:

```
print_5_spaces:
    li $s1, 0       # Initialize loop counter

print_5_spaces_loop:
    beq $s1, 5, print_5_spaces_end  # Exit loop when counter reaches 5
    la $a0, 32  # Load address of space character
    syscall         # Print the space
    addi $s1, $s1, 1  # Increment counter
    j print_5_spaces_loop           # Jump back to the beginning of the loop

print_5_spaces_end:
    jr $ra          # Return from the function
```

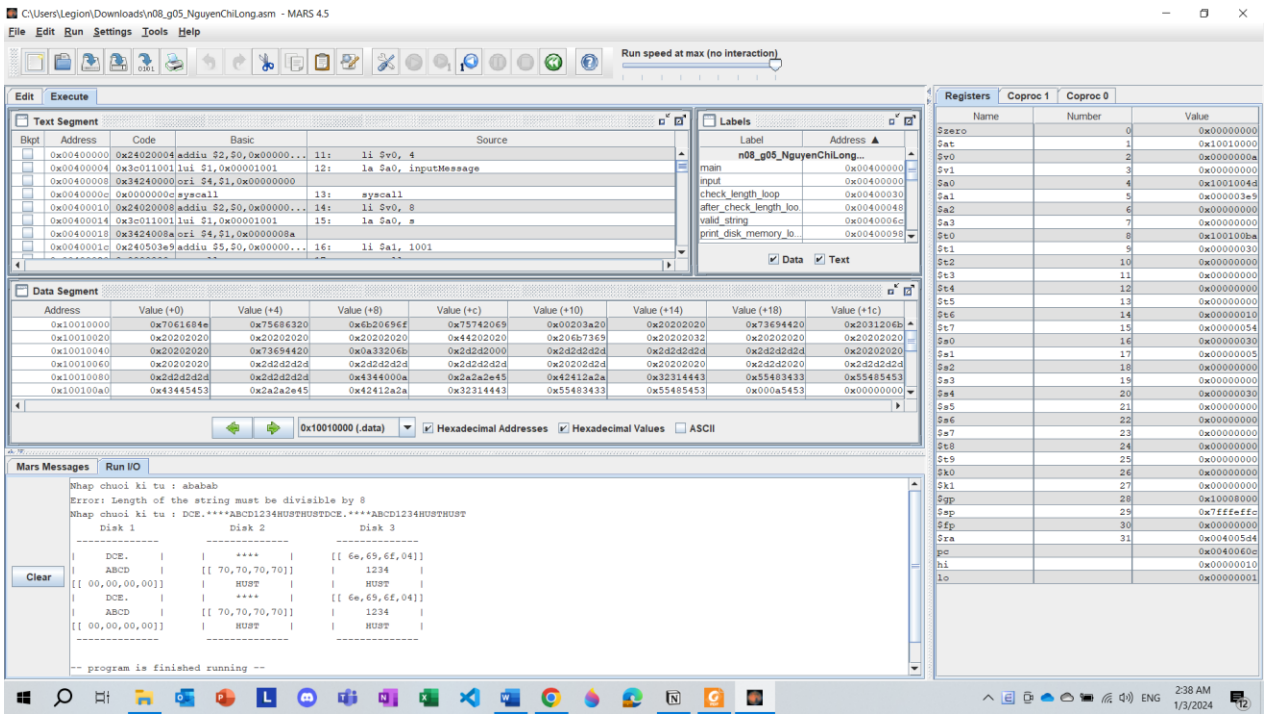
print_6_spaces:

```
print_6_spaces:
    li $s1, 0       # Initialize loop counter

print_6_spaces_loop:
    beq $s1, 6, print_6_spaces_end  # Exit loop when counter reaches 5
    la $a0, 32  # Load address of space character
    syscall         # Print the space
    addi $s1, $s1, 1  # Increment counter
    j print_6_spaces_loop           # Jump back to the beginning of the loop

print_6_spaces_end:
    jr $ra          # Return from the function
```

3. Results demonstration


C:\Users\Legion\Downloads\n08_g05_NguyenChiLong.asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
Bkpt Address Code Basic Source
0x00400000 0x24020004 addiu $2,$0,0x00000... 11: li $v0, 4
0x00400004 0x3c011001 lui $1,0x00001001 12: la $a0, inputMessage
0x00400008 0x34240000 ori $4,$1,0x00000000
0x0040000c 0x0000000c syscall 13: syscall
0x00400010 0x24020008 addiu $2,$0,0x00000... 14: li $v0, 8
0x00400014 0x3c011001 lui $1,0x00001001 15: la $a0, s
0x00400018 0x3424008a ori $4,$1,0x0000008a
0x0040001c 0x240503e9 addiu $5,$0,0x00000... 16: li $a1, 1001
Labels
Label Address
n08_g05_NguyenChiLong...
main 0x00400000
input 0x00400000
check_length_loop 0x00400030
after_check_length_loo 0x00400048
valid_string 0x0040006c
print_disk_memory_lo... 0x00400098
Data Text
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14) Value (+18) Value (+1c)
0x10010000 0x7061684e 0x75686320 0x6b20696f 0x75742069 0x00203a20 0x20202020 0x73694420 0x2031206b
0x10010020 0x20202020 0x20202020 0x20202020 0x44202020 0x206b7369 0x20202032 0x20202020 0x20202020
0x10010040 0x20202020 0x73694420 0x0a33206b 0x2d2d2000 0x2d2d2d2d 0x2d2d2d2d 0x2d2d2d2d 0x20202020
0x10010060 0x20202020 0x2d2d2d2d 0x2d2d2d2d 0x2d2d2d2d 0x20202d2d 0x20202020 0x2d2d2020 0x2d2d2d2d
0x10010080 0x2d2d2d2d 0x2d2d2d2d 0x4344000a 0x2a2a2e45 0x42412a2a 0x32314443 0x55483433 0x55485453
0x100100a0 0x43445453 0x2a2a2e45 0x42412a2a 0x32314443 0x55483433 0x55485453 0x000a5453 0x00000000
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII
Registers Coproc 1 Coproc 0
Name Number Value
$zero 0 0x00000000
$at 1 0x10010000
$v0 2 0x0000000a
$v1 3 0x00000000
$a0 4 0x1001004d
$a1 5 0x000003e9
$a2 6 0x00000000
$a3 7 0x00000000
$t0 8 0x100100ba
$t1 9 0x00000030
$t2 10 0x00000000
$t3 11 0x00000000
$t4 12 0x00000000
$t5 13 0x00000000
$t6 14 0x00000010
$t7 15 0x00000054
$s0 16 0x00000030
$s1 17 0x00000005
$s2 18 0x00000000
$s3 19 0x00000000
$s4 20 0x00000030
$s5 21 0x00000000
$s6 22 0x00000000
$s7 23 0x00000000
$t8 24 0x00000000
$t9 25 0x00000000
$k0 26 0x00000000
$k1 27 0x00000000
$gp 28 0x10008000
$sp 29 0x7fffeffc
$fp 30 0x00000000
$ra 31 0x004005d4
pc 0x0040060c
hi 0x00000010
lo 0x00000001
Mars Messages Run I/O
Clear
Nhap chuoi ki tu : ababab
Error: Length of the string must be divisible by 8
Nhap chuoi ki tu : DCE.****ABCD1234HUSTHUSTDCE.****ABCD1234HUSTHUST
Disk 1 Disk 2 Disk 3
DCE. **** [[6e,65,6f,04]]
ABCD [[70,70,70,70]] 1234
[[00,00,00,00]] HUST HUST
DCE. **** [[6e,65,6f,04]]
ABCD [[70,70,70,70]] 1234
[[00,00,00,00]] HUST HUST
-- program is finished running --
ENG 2:38 AM 1/3/2024