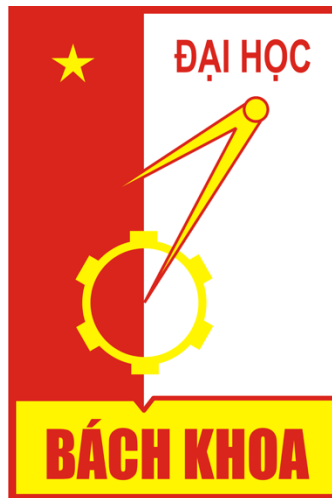


HANOI UNIVERSITY OF SCIENCE & TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



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	3
1. Problem description.....	3
2. Project implementation.....	3
a. Algorithm	3
b. Code explanation.....	4
3. Results demonstration	7
II. Task 8: RAID 5 simulation.....	10
1. Problem description.....	10
2. Problem implementation	10
3. Results demonstration	15



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 $(X_0, Y_0) = (256, 256)$ which is the center of bitmap screen, $R = 16$ (radius of the ball), $\Delta X = \Delta Y = 0$ (no moving at start), $\text{speed} = 2$ (initial speed).

Step 2: Calculate relative location with respect to the center of the circle of points in the circle with formula $(X_i, Y_i) = (X_0 + i, Y_0 + \sqrt{R^2 - i^2})$. 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 $\Delta X = -\text{speed}$, $\Delta Y = 0$.
- If user inputted "d" set $\Delta X = \text{speed}$, $\Delta Y = 0$.
- If user inputted "s" set $\Delta Y = \text{speed}$, $\Delta X = 0$.
- If user inputted "w" set $\Delta Y = -\text{speed}$, $\Delta X = 0$.
- If user inputted "z" set $\text{speed} = \text{speed} + 1$ (add \$s7, \$s7, 1)
- If user inputted "x" set $\text{speed} = \text{speed} - 1$ (add \$s7, \$s7, -1) and also check the $\text{speed} \leq 0$ 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 $(0, 0) < (X, Y) + R + (\Delta X, \Delta Y) < 512$. 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 ($X = X + \text{delta}X$, $Y = Y + \text{delta}Y$).

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 (X_i, Y_i) 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 ($X_o + i$, $Y_o + j$), ($X_o + j$, $Y_o + i$) 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.

```
15 # $a2, $a3: loop's iterators
16 .text
17     li $s0, 256      # X_0 = 256
18     li $s1, 256      # Y_0 = 256
19     li $s2, 16       # R = 16
20     li $s3, 512      # width of screen
21     li $s4, 512      # height of screen
22     li $s5, 0        # deltaX = 0
23     li $s6, 0        # deltaY = 0
24     li $s7, 2        # speed
25
26 circle_bound_init: # use $t0, $t1, $t2, $t3, no need to keep
27     la $t1, circle_bound
28     li $a2, 0
29     mul $t2, $s2, $s2 # $t2 = R^2
30 circle_bound_loop:
31     ble $s2, $a2, end_circle_bound_loop
32     mul $t3, $a2, $a2
33     sub $t3, $t2, $t3 # $t3 = R^2 - i^2
34     move $t0, $t3 # $t0 = sqrt(R^2 - i^2)
35     jal sqrt
36     sw $t0, 0($t1)
37     add $a2, $a2, 1
38     add $t1, $t1, 4
39     j circle_bound_loop
40 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.

```
41 game_loop:
42 read_keyboard:
43     lw $k1, KEY_READY          # if a key is clicked, $k1 = 1
44     beq $k1, $zero, position_check # $k1 = 0 then run position check
45     lw $k0, KEY_CODE
46     beq $k0, KEY_A, case_a
47     beq $k0, KEY_S, case_s
48     beq $k0, KEY_D, case_d
49     beq $k0, KEY_W, case_w
50     beq $k0, KEY_Z, case_z
51     beq $k0, KEY_X, case_x
52     j position_check
53 case_a:
54     jal move_left
55     j position_check
56 case_s:
57     jal move_down
58     j position_check
59 case_d:
60     jal move_right
61     j position_check
62 case_w:
63     jal move_up
64     j position_check
65 case_z:
66     jal speed_up
67     j position_check
68 case_x:
69     jal slow_down
70
```

```

153 # Note: in speed_up, we are based on the direction to call move again, so that the speed are immediately changed
154 move_left:
155     sub $s5, $zero, $s7 # move left $s7 unit
156     li $s6, 0
157     jr $ra
158 move_right:
159     add $s5, $zero, $s7 # move right $s7 unit
160     li $s6, 0
161     jr $ra
162 move_up:
163     li $s5, 0
164     sub $s6, $zero, $s7 # move up $s7 unit
165     jr $ra
166 move_down:
167     li $s5, 0
168     add $s6, $zero, $s7 # move down $s7 unit
169     jr $ra
170 speed_up: # $s7 += 1
171     add $s7, $s7, 1
172
173     jr $ra
174 slow_down: # $s7 -= 1, min speed: 1
175     add $s7, $s7, -1
176     blt $zero, $s7, back
177     li $s7, 1
178 back:
179     jr $ra
180

```

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

```

70
71 # Check if the circle touches an edge or not. After checking 4 directions, draw the circle
72 position_check: # use $t0, no need to keep
73 check_right:
74     add $t0, $s0, $s2
75     add $t0, $t0, $s5 # $t0 = X_0 + R + deltaX: rightest point on the circle after this time step
76     ble $t0, $s3, check_left # if $t0 < width of screen, no need to check more. If not, reverse the direction
77     jal move_left
78 check_left:
79     sub $t0, $s0, $s2
80     add $t0, $t0, $s5 # $t0 = X_0 - R + deltaX: leftest point on the circle after this time step
81     ble $zero, $t0, check_top # if 0 < $t0, no need to check more. If not, reverse the direction
82     jal move_right
83 check_top:
84     sub $t0, $s1, $s2
85     add $t0, $t0, $s6 # $t0 = Y_0 - R + deltaY: highest point on the circle after this time step
86     ble $zero, $t0, check_bottom # if 0 < $t0, no need to check more. If not, reverse the direction
87     jal move_down
88 check_bottom:
89     add $t0, $s1, $s2
90     add $t0, $t0, $s6 # $t0 = Y_0 + R + deltaY: lowest point on the circle after this time step
91     ble $t0, $s4, draw # if $t0 < height of screen, no need to check more. If not, reverse the direction
92     jal move_up

```

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

```

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

```

- o 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
```

- o In each quarters, draw 2 points:

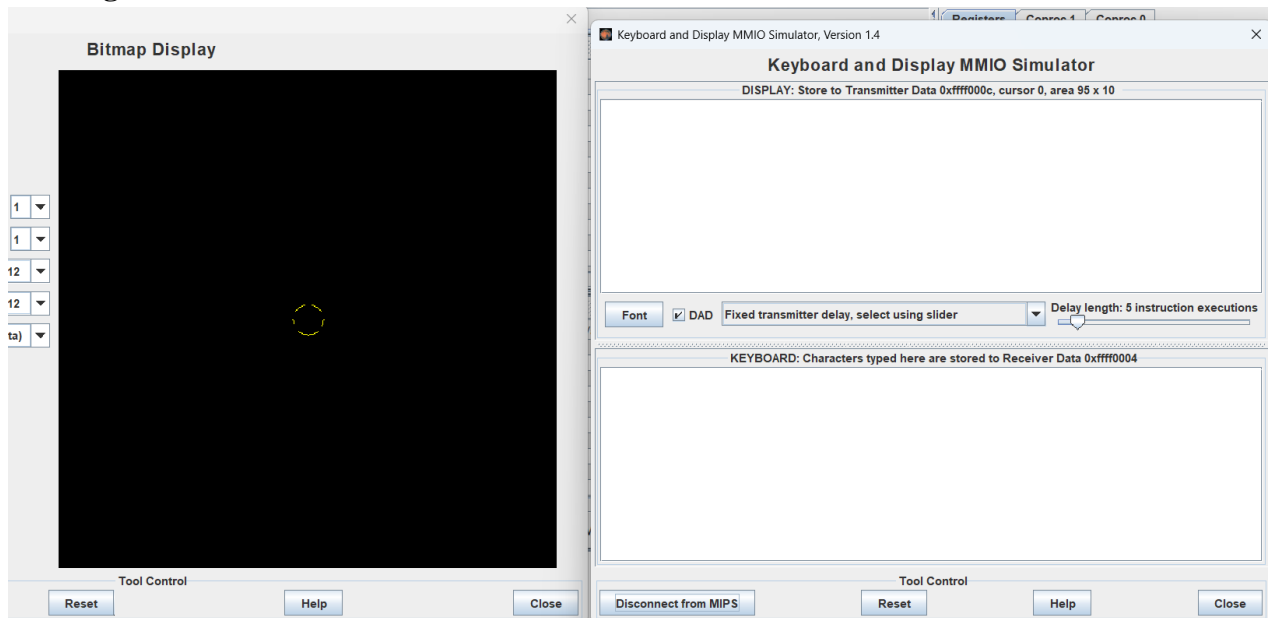
```
134 drawCirclePoint:
135
136     add $t5, $s0, $t3      # Xi = X0 + i
137     add $t6, $s1, $t4      # Yi = Y0 + j
138     mul $t6, $t6, $s3      # Yi * SCREEN_WIDTH
139     add $t5, $t5, $t6      # Yi * SCREEN_WIDTH + Xi (Toa do 1 chieu cua diem anh)
140     sll $t5, $t5, 2        # Dia chi tuong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

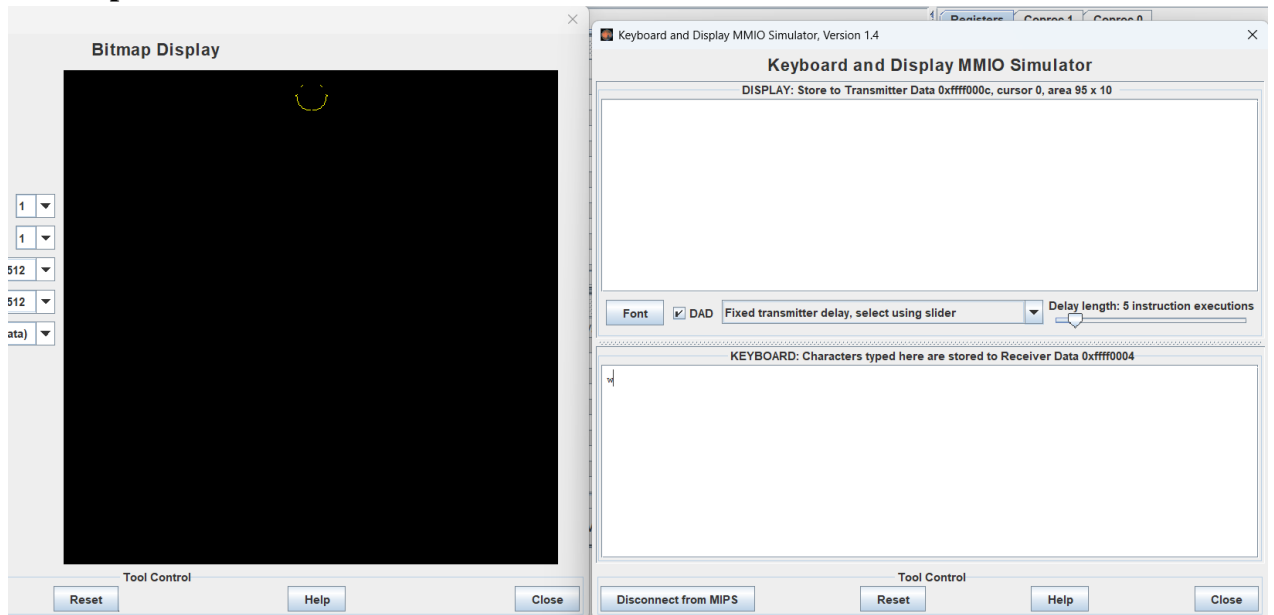
The demonstration results is not sufficient to this task, I will represent it directly to you when we met up for reporting.



- Starting state:

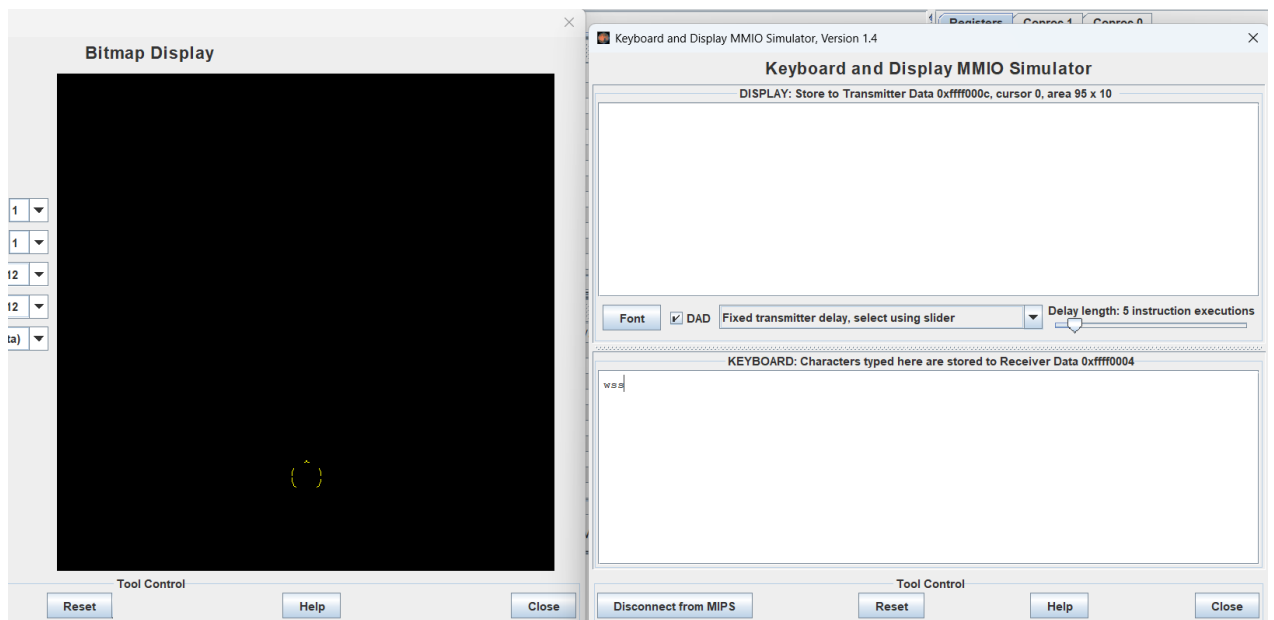


- Move up:

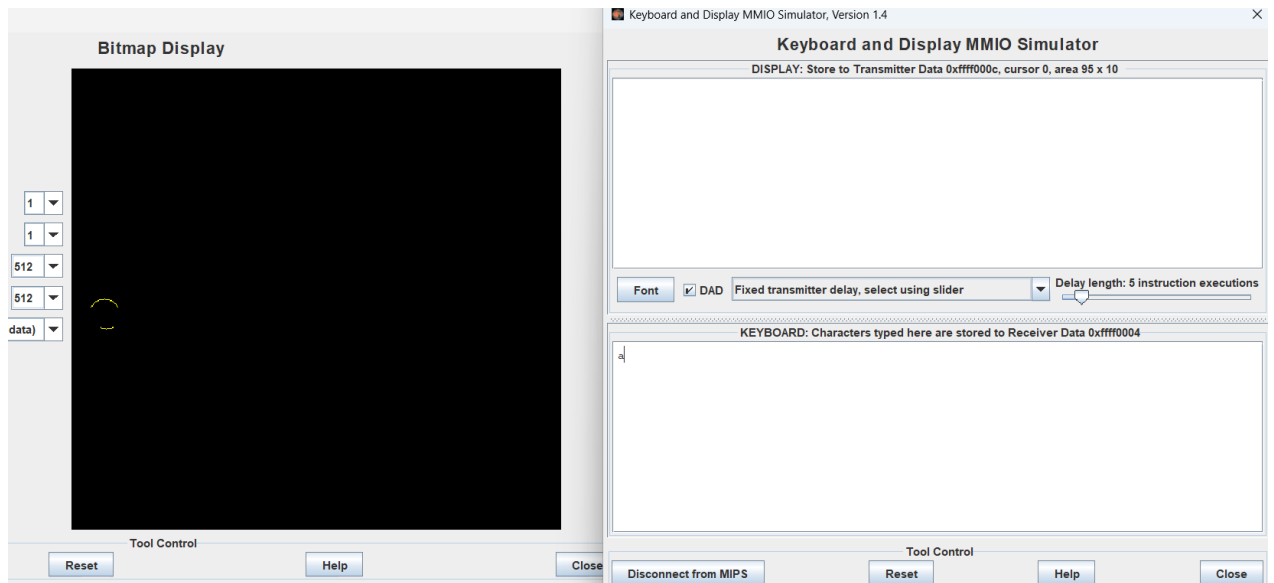




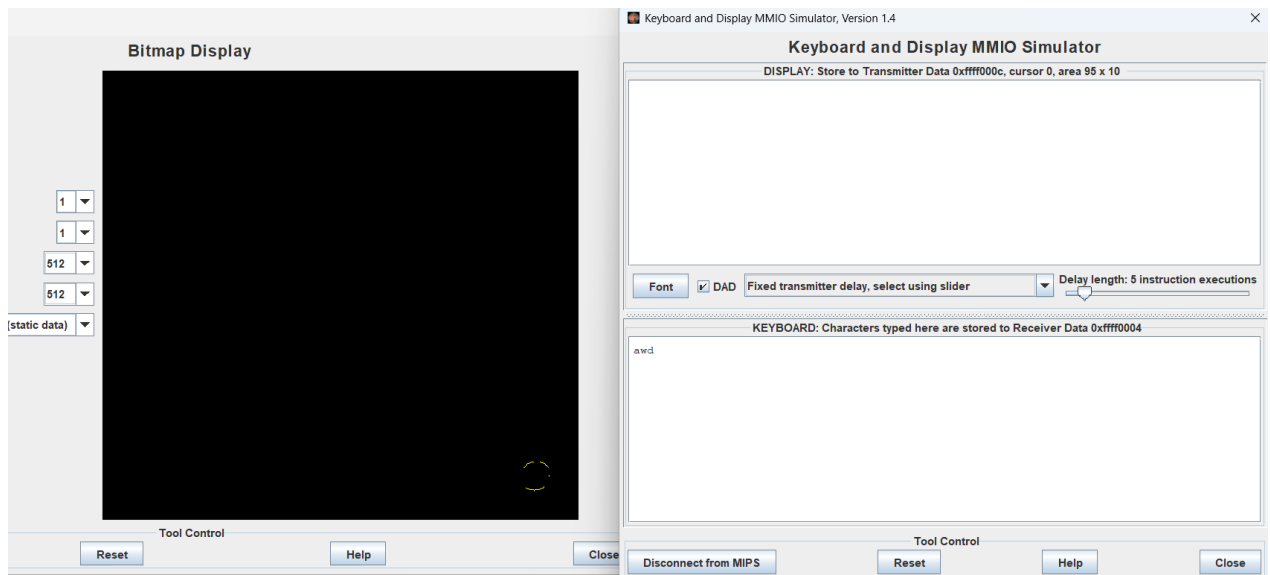
- Move down:



- 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 $6e = 'D' \text{ xor } '*' ; 69 = 'C' \text{ xor } '*' ; 6f = 'E' \text{ xor } '*' ; 04 = '.' \text{ xor } '*'$

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.

```
1 |.data
2 |inputMessage: .asciiiz "Nhap chuoi ki tu : "
3 |outputDisk: .asciiiz "      Disk 1      Disk 2      Disk 3\n"
4 |outputLine: .asciiiz " -----\n"
5 |s: .space 1001
6 |hex: .byte '0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'
7 |errorMessage: .asciiiz "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.

```
19 # $s0: size of string s
20     la $t0, s
21     li $s0, 0
22 check_length_loop:
23     lb $t1, ($t0)
24     beq $t1, 10, after_check_length_loop
25     add $t0, $t0, 1
26     add $s0, $s0, 1
27     j check_length_loop
28
29 after_check_length_loop:
30     rem $t0, $s0, 8
31     beq $t0, $zero, valid_string
32     li $v0, 4
33     la $a0, errorMessage
34     syscall
35     j input
36 valid_string:
37     li $v0, 4
38     la $a0, outputDisk
39     syscall
40     li $v0, 4
41     la $a0, outputLine
42     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.



```
44 # $t2, $t3, $t4, $t5: xors of 4 pairs
45     la $t0, s
46     li $t1, 0
47 print_disk_memory_loop:
48     beq $t1, $s0, after_print_disk_memory_loop
49
50     lb $t6, 0($t0)
51     lb $t7, 4($t0)
52     xor $t2, $t6, $t7
53
54     lb $t6, 1($t0)
55     lb $t7, 5($t0)
56     xor $t3, $t6, $t7
57
58     lb $t6, 2($t0)
59     lb $t7, 6($t0)
60     xor $t4, $t6, $t7
61
62     lb $t6, 3($t0)
63     lb $t7, 7($t0)
64     xor $t5, $t6, $t7
65
66     rem $t6, $t1, 24
67     beq $t6, 0, print_8_char1
68     beq $t6, 8, print_8_char2
69     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.



```
116     li $a0, 91
117     syscall
118     li $a0, 91
119     syscall
120
121     li $a0, 32
122     syscall
123
124     srl $s2, $t2, 4
125     and $s3, $t2, 0x0000000f
126
127     lb $s4, hex($s2)
128     move $a0, $s4
129     syscall
130     lb $s4, hex($s3)
131     move $a0, $s4
132     syscall
133
134     li $a0, 44
135     syscall
136
137     srl $s2, $t3, 4
138     and $s3, $t3, 0x0000000f
139
140     lb $s4, hex($s2)
141     move $a0, $s4
142     syscall
143     lb $s4, hex($s3)
144     move $a0, $s4
145     syscall
146
147     li $a0, 44
148     syscall
149
150     srl $s2, $t4, 4
151     and $s3, $t4, 0x0000000f
152
153     lb $s4, hex($s2)
154     move $a0, $s4
155     syscall
156     lb $s4, hex($s3)
157     move $a0, $s4
158     syscall
159
160     li $a0, 44
161     syscall
162
163     srl $s2, $t5, 4
164     and $s3, $t5, 0x0000000f
165
166     lb $s4, hex($s2)
167     move $a0, $s4
168     syscall
169     lb $s4, hex($s3)
170     move $a0, $s4
171     syscall
172
173     li $a0, 93
174     syscall
175     li $a0, 93
176     syscall
177
178     li $a0, 10
179     syscall
180     j after_print_8_char
```



```
404 after_print_8_char:
405     add $t0, $t0, 8
406     add $t1, $t1, 8
407     j print_disk_memory_loop
408
```

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

```
409 after_print_disk_memory_loop:
410     li $v0, 4
411     la $a0, outputLine
412     syscall
413     li $v0, 10
414     syscall
415
```

- *Other functions:*

print_5_spaces:

```
416 print_5_spaces:
417     li $s1, 0          # Initialize loop counter
418
419 print_5_spaces_loop:
420     beq $s1, 5, print_5_spaces_end # Exit loop when counter reaches 5
421     la $a0, 32          # Load address of space character
422     syscall             # Print the space
423     addi $s1, $s1, 1    # Increment counter
424     j print_5_spaces_loop      # Jump back to the beginning of the loop
425
426 print_5_spaces_end:
427     jr $ra              # Return from the function
```

print_6_spaces:

```
429 print_6_spaces:
430     li $s1, 0          # Initialize loop counter
431
432 print_6_spaces_loop:
433     beq $s1, 6, print_6_spaces_end # Exit loop when counter reaches 5
434     la $a0, 32          # Load address of space character
435     syscall             # Print the space
436     addi $s1, $s1, 1    # Increment counter
437     j print_6_spaces_loop      # Jump back to the beginning of the loop
438
439 print_6_spaces_end:
440     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)

Text Segment

Offset	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	0x3424000a	ori \$4,\$1,0x0000000a		
0x0040001c	0x240503e9	addiu \$5,\$0,0x00000...	16: li \$a1, 1001	

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x7061694a	0x75666320	0x6b20696f	0x75742069	0x00203a20	0x20202020	0x73694420	0x2031206b
0x10010020	0x20202020	0x20202020	0x20202020	0x42020202	0x206b7369	0x20202032	0x20202020	0x20202020
0x10010040	0x20202020	0x73694420	0x0a33206b	0x2d2d2d2d	0x2d2d2d2d	0x2d2d2d2d	0x2d2d2d2d	0x20202020
0x10010060	0x20202020	0x2d2d2d2d	0x2d2d2d2d	0x2d2d2d2d	0x20202020	0x2d2d2d2d	0x2d2d2d2d	0x2d2d2d2d
0x10010080	0x2d2d2d2d	0x2d2d2d2d	0x4340000a	0x2a2a2e45	0x42412a2a	0x32314443	0x55485453	0x000a5453
0x100100a0	0x43445453	0x2a2a2e45	0x42412a2a	0x32314443	0x55485453	0x000a5453	0x000a5453	0x00000000

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x10010040
\$a1	5	0x000003e9
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x1001008a
\$t1	9	0x00000030
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000010
\$t7	15	0x00000054
\$a0	16	0x00000030
\$a1	17	0x00000005
\$a2	18	0x00000000
\$a3	19	0x00000000
\$a4	20	0x00000030
\$a5	21	0x00000000
\$a6	22	0x00000000
\$a7	23	0x00000000
\$t0	24	0x00000000
\$t1	25	0x00000000
\$t2	26	0x00000000
\$t3	27	0x00000000
\$t4	28	0x10008000
\$t5	29	0x7fffffc
\$t6	30	0x00000000
\$t7	31	0x0040054d
\$ra		0x0040054d
\$sp		0x00000000
\$fp		0x00000000
\$s0		0x00000000
\$s1		0x00000000
\$s2		0x00000000
\$s3		0x00000000
\$s4		0x00000000
\$s5		0x00000000
\$s6		0x00000000
\$s7		0x00000000
\$s8		0x00000000
\$s9		0x00000000
\$s10		0x00000000
\$s11		0x00000000
\$s12		0x00000000
\$s13		0x00000000
\$s14		0x00000000
\$s15		0x00000000
\$s16		0x00000000
\$s17		0x00000000
\$s18		0x00000000
\$s19		0x00000000
\$s20		0x00000000
\$s21		0x00000000
\$s22		0x00000000
\$s23		0x00000000
\$s24		0x00000000
\$s25		0x00000000
\$s26		0x00000000
\$s27		0x00000000
\$s28		0x00000000
\$s29		0x00000000
\$s30		0x00000000
\$s31		0x00000000
\$s32		0x00000000
\$s33		0x00000000
\$s34		0x00000000
\$s35		0x00000000
\$s36		0x00000000
\$s37		0x00000000
\$s38		0x00000000
\$s39		0x00000000
\$s40		0x00000000
\$s41		0x00000000
\$s42		0x00000000
\$s43		0x00000000
\$s44		0x00000000
\$s45		0x00000000
\$s46		0x00000000
\$s47		0x00000000
\$s48		0x00000000
\$s49		0x00000000
\$s50		0x00000000
\$s51		0x00000000
\$s52		0x00000000
\$s53		0x00000000
\$s54		0x00000000
\$s55		0x00000000
\$s56		0x00000000
\$s57		0x00000000
\$s58		0x00000000
\$s59		0x00000000
\$s60		0x00000000
\$s61		0x00000000
\$s62		0x00000000
\$s63		0x00000000
\$s64		0x00000000
\$s65		0x00000000
\$s66		0x00000000
\$s67		0x00000000
\$s68		0x00000000
\$s69		0x00000000
\$s70		0x00000000
\$s71		0x00000000
\$s72		0x00000000
\$s73		0x00000000
\$s74		0x00000000
\$s75		0x00000000
\$s76		0x00000000
\$s77		0x00000000
\$s78		0x00000000
\$s79		0x00000000
\$s80		0x00000000
\$s81		0x00000000
\$s82		0x00000000
\$s83		0x00000000
\$s84		0x00000000
\$s85		0x00000000
\$s86		0x00000000
\$s87		0x00000000
\$s88		0x00000000
\$s89		0x00000000
\$s90		0x00000000
\$s91		0x00000000
\$s92		0x00000000
\$s93		0x00000000
\$s94		0x00000000
\$s95		0x00000000
\$s96		0x00000000
\$s97		0x00000000
\$s98		0x00000000
\$s99		0x00000000
\$s100		0x00000000
\$s101		0x00000000
\$s102		0x00000000
\$s103		0x00000000
\$s104		0x00000000
\$s105		0x00000000
\$s106		0x00000000
\$s107		0x00000000
\$s108		0x00000000
\$s109		0x00000000
\$s110		0x00000000
\$s111		0x00000000
\$s112		0x00000000
\$s113		0x00000000
\$s114		0x00000000
\$s115		0x00000000
\$s116		0x00000000
\$s117		0x00000000
\$s118		0x00000000
\$s119		0x00000000
\$s120		0x00000000
\$s121		0x00000000
\$s122		0x00000000
\$s123		0x00000000
\$s124		0x00000000
\$s125		0x00000000
\$s126		0x00000000
\$s127		0x00000000
\$s128		0x00000000
\$s129		0x00000000
\$s130		0x00000000
\$s131		0x00000000
\$s132		0x00000000
\$s133		0x00000000
\$s134		0x00000000
\$s135		0x00000000
\$s136		0x00000000
\$s137		0x00000000
\$s138		0x00000000
\$s139		0x00000000
\$s140		0x00000000
\$s141		0x00000000
\$s142		0x00000000
\$s143		0x00000000
\$s144		0x00000000
\$s145		0x00000000
\$s146		0x00000000
\$s147		0x00000000
\$s148		0x00000000
\$s149		0x00000000
\$s150		0x00000000
\$s151		0x00000000
\$s152		0x00000000
\$s153		0x00000000
\$s154		0x00000000
\$s155		0x00000000
\$s156		0x00000000
\$s157		0x00000000
\$s158		0x00000000
\$s159		0x00000000
\$s160		0x00000000
\$s161		0x00000000
\$s162		0x00000000
\$s163		0x00000000
\$s164		0x00000000
\$s165		0x00000000
\$s166		0x00000000
\$s167		0x00000000
\$s168		0x00000000
\$s169		0x00000000
\$s170		0x00000000
\$s171		0x00000000
\$s172		0x00000000
\$s173		0x00000000
\$s174		0x00000000
\$s175		0x00000000
\$s176		0x00000000
\$s177		0x00000000
\$s178		0x00000000
\$s179		0x00000000
\$s180		0x00000000
\$s181		0x00000000
\$s182		0x00000000
\$s183		0x00000000
\$s184		0x00000000
\$s185		0x00000000
\$s186		0x00000000
\$s187		0x00000000
\$s188		0x00000000
\$s189		0x00000000
\$s190		0x00000000
\$s191		0x00000000
\$s192		0x00000000
\$s193		0x00000000
\$s194		0x00000000
\$s195		0x00000000
\$s196		0x00000000
\$s197		0x00000000
\$s198		0x00000000
\$s199		0x00000000
\$s200		0x00000000
\$s201		0x00000000
\$s202		0x00000000
\$s203		0x00000000
\$s204		0x00000000
\$s205		0x00000000
\$s206		0x00000000
\$s207		0x00000000
\$s208		0x00000000
\$s209		0x00000000
\$s210		0x00000000
\$s211		0x00000000
\$s212		0x00000000
\$s213		0x00000000
\$s214		0x00000000
\$s215		0x00000000
\$s216		0x00000000
\$s217		0x00000000
\$s218		0x00000000
\$s219		0x00000000
\$s220		0x00000000
\$s221		0x00000000
\$s222		0x00000000
\$s223		0x00000000
\$s224		0x00000000
\$s225		0x00000000
\$s226		0x00000000
\$s227		0x00000000
\$s228		0x00000000
\$s229		0x00000000
\$s230		0x00000000
\$s231		0x00000000
\$s232		0x00000000
\$s233		0x00000000
\$s234		0x00000000
\$s235		0x00000000
\$s236		0x00000000
\$s237		0x00000000
\$s238		0x00000000
\$s239		0x00000000
\$s240		0x00000000
\$s241		0x00000000
\$s242		0x00000000
\$s243		0x00000000
\$s244		0x00000000
\$s245		0x00000000
\$s246		0x00000000
\$s247		0x00000000
\$s248		0x00000000
\$s249		0x00000000
\$s250		0x00000000
\$s251		0x00000000
\$s252		0x00000000
\$s253		0x00000000
\$s254		0x00000000
\$s255		0x00000000
\$s256		0x00000000
\$s257		0x00000000
\$s258		0x00000000
\$s259		0x00000000
\$s260		0x00000000
\$s261		0x00000000
\$s262		0x00000000
\$s263		0x00000000
\$s264		0x00000000
\$s265		0x00000000
\$s266		0x00000000
\$s267		0x00000000
\$s268		0x00000000
\$s269		0x00000000
\$s270		0x00000000
\$s271		0x00000000
\$s272		0x00000000
\$s273		0x00000000
\$s274		0x00000000
\$s275		0x00000000
\$s276		0x00000000
\$s277		0x00000000
\$s278		0x00000000
\$s279		0x00000000
\$s280		0x00000000
\$s281		0x00000000
\$s282		0x00000000
\$s283		0x00000000
\$s284		0x00000000
\$s285		0x00000000
\$s286		0x00000000
\$s287		0x00000000
\$s288		0x00000000
\$s289		0x00000000
\$s290		0x00000000
\$s291		0x00000000
\$s292		0x00000000
\$s293		0x00000000
\$s294		0x00000000
\$s295		0x00000000
\$s296		0x00000000
\$s297		0x00000000
\$s298		0x00000000
\$s299		0x00000000
\$s300		0x000