

Bài thực hành số 10

Lớp: 139365 – Học phần: Thực Hành Kiến Trúc Máy Tính
Đào Minh Nhật – 20215107

Assignment 1:

Code:

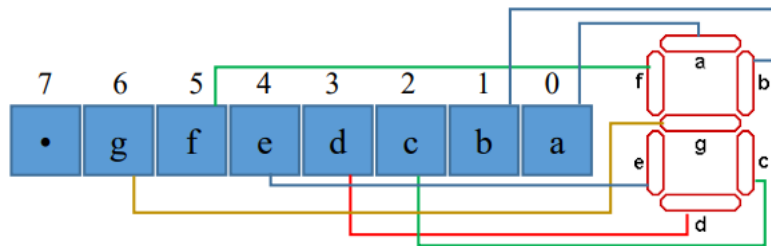
```
1  # Lab 10, Assignment 1
2  .eqv SEVENSEG_LEFT 0xFFFF0011 # Địa chỉ của đèn led 7 đoạn trái.
3
4  .eqv SEVENSEG_RIGHT 0xFFFF0010 # Địa chỉ của đèn led 7 đoạn phải
5
6  .text
7  main:
8      li $a0, 0x3F # set value for segments
9      jal SHOW_7SEG_LEFT # show
10     nop
11     li $a0, 0x07 # set value for segments
12     jal SHOW_7SEG_RIGHT # show
13     nop
14  exit: li $v0, 10
15         syscall
16  endmain:
17
18  SHOW_7SEG_LEFT:
19     li $t0, SEVENSEG_LEFT # assign port's address
20     sb $a0, 0($t0) # assign new value
21     nop
22
23     jr $ra
24     nop
25
26  SHOW_7SEG_RIGHT:
27     li $t0, SEVENSEG_RIGHT # assign port's address
28     sb $a0, 0($t0) # assign new value
29     nop
30     jr $ra
31     nop
```

Giải thích:

Địa chỉ của đèn led 7 bên trái: 0xFFFF0011

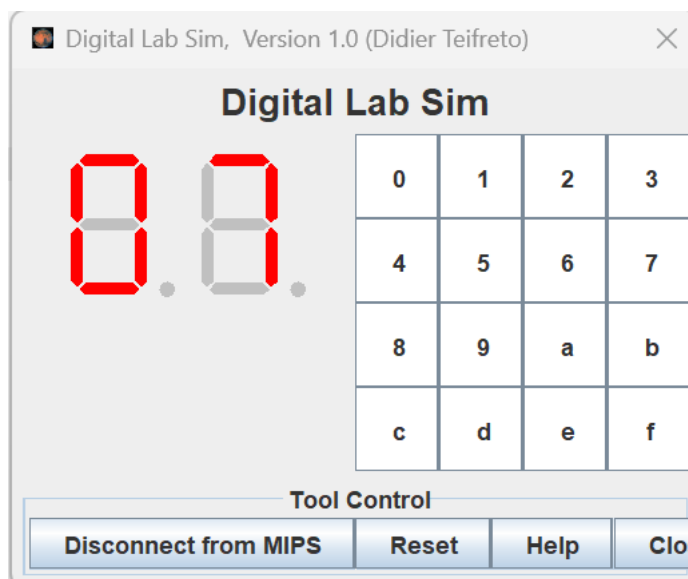
Địa chỉ của đèn led 7 bên phải: 0xFFFF0010

\$a0: giá trị hiển thị các thanh led



Với \$a0 = 0x3F (00111111) thì đèn sẽ hiển thị số 0

Với \$a0 = 0x07 (00000111) thì đèn sẽ hiển thị số 7



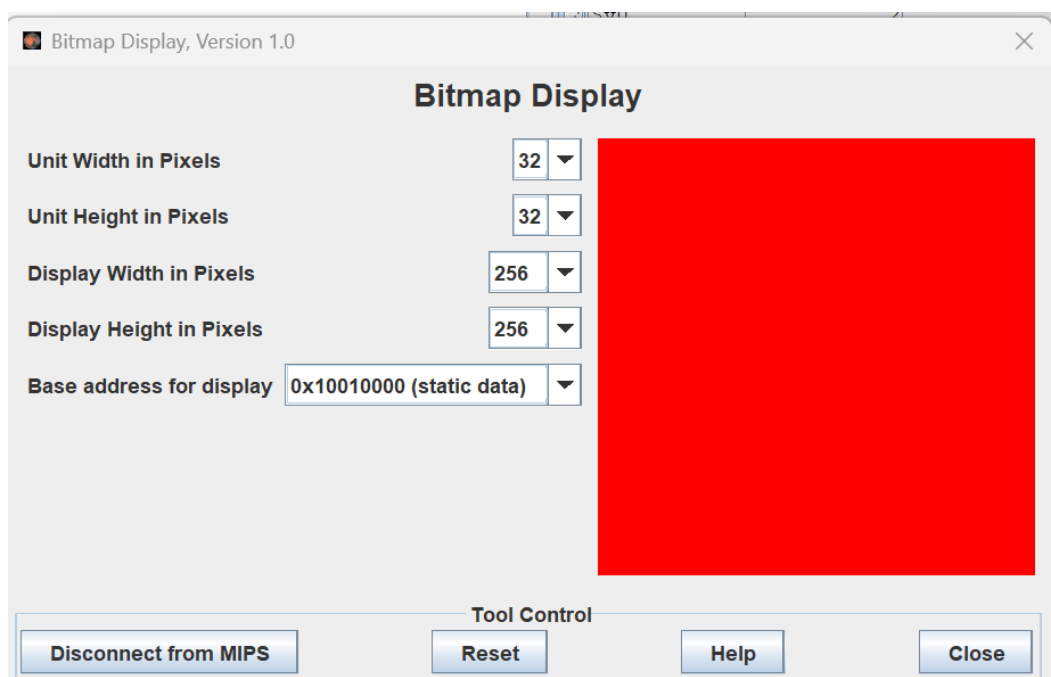
Assignment 2

Code:

Chương trình in ra toàn màn hình màu đỏ

```
1  # Lab 10, Assignment 2
2  .eqv MONITOR_SCREEN 0x10010000 #Địa chỉ bắt đầu của bộ nhớ màn hình
3  .eqv RED 0x00FF0000
4
5  .text
6      li $k0, MONITOR_SCREEN #Nạp địa chỉ bắt đầu của màn hình
7
8      li $t0, RED
9      li $s0, 0
10     li $s1, 256
11
12 scan_full_red:
13     add $t1, $k0, $s0
14
15     sw $t0, 0($t1)
16
17     addi $s0, $s0, 4
18     bne $s0, $s1, scan_full_red
19 end_scan_full_red:
```

Kết quả:



Assignment 3

Code:

```
1  # Lab 10, Assignment 3
2  .eqv HEADING 0xffff8010 # Integer: An angle between 0 and 359
3                               # 0 : North (up)
4                               # 90: East (right)
5                               # 180: South (down)
6                               # 270: West (left)
7  .eqv MOVING 0xffff8050 # Boolean: whether or not to move
8  .eqv LEAVETRACK 0xffff8020 # Boolean (0 or non-0):
9                               # whether or not to leave a track
10 .eqv WHEREX 0xffff8030 # Integer: Current x-location of MarsBot
11 .eqv WHEREY 0xffff8040 # Integer: Current y-location of MarsBot
12
13 .text
14 main:
15     jal TRACK # draw track line
16     nop
17     addi $a0, $zero, 90 # Marsbot rotates 90* and start running
18     jal ROTATE
19     nop
20     jal GO
21     nop
22
23 sleep1: addi $v0, $zero, 32 # Keep running by sleeping in 1000 ms
24         li $a0, 1000
25
26
27     jal UNTRACK # keep old track
28     nop
29     jal TRACK # and draw new track line
30     nop
31
32 goDOWN: addi $a0, $zero, 180 # Marsbot rotates 180*
33         jal ROTATE
34         nop
35
36 sleep2: addi $v0, $zero, 32 # Keep running by sleeping in 2000 ms
37         li $a0, 2000
38         syscall
39         jal UNTRACK # keep old track
40         nop
41         jal TRACK # and draw new track line
42         nop
43
44 goLEFT: addi $a0, $zero, 270 # Marsbot rotates 270*
45         jal ROTATE
46         nop
47
```

```

48 sleep3: addi $v0,$zero,32 # Keep running by sleeping in 1000 ms
49         li $a0,1000
50         syscall
51         jal UNTRACK # keep old track
52         nop
53         jal TRACK # and draw new track line
54         nop
55
56 goASKEW: addi $a0, $zero, 120 # Marsbot rotates 120*
57         jal ROTATE
58         nop
59
60 sleep4: addi $v0,$zero,32 # Keep running by sleeping in 2000 ms
61         li $a0,2000
62         syscall
63
64         jal UNTRACK # keep old track
65         nop
66         jal TRACK # and draw new track line
67         nop
68 end_main:
69

```

```

70 GO:     li $at, MOVING # change MOVING port
71         addi $k0, $zero,1 # to logic 1,
72         sb $k0, 0($at) # to start running
73         nop
74         jr $ra
75         nop
76
77 STOP:   li $at, MOVING # change MOVING port to 0
78         sb $zero, 0($at) # to stop
79         nop
80         jr $ra
81         nop
82
83 TRACK:  li $at, LEAVETRACK # change LEAVETRACK port
84         addi $k0, $zero,1 # to logic 1,
85         sb $k0, 0($at) # to start tracking
86         nop
87         jr $ra
88         nop
89
90 UNTRACK:li $at, LEAVETRACK # change LEAVETRACK port to 0
91         sb $zero, 0($at) # to stop drawing tail
92         nop
93         jr $ra

```

```

94         nop
95
96 ROTATE: li $at, HEADING # change HEADING port
97         sw $a0, 0($at) # to rotate robot
98         nop
99         jr $ra
100        nop

```

Giải thích:

Dòng 15: Thực hiện hàm TRACK để vẽ đường kẻ

Hàm TRACK (dòng 83-88):

Thay đổi đến cổng LEAVETRACK và truyền vào logic 1 ($\$k0 = 1$) thực hiện vẽ đường kẻ.

Dòng 17-18: Khởi tạo $\$a0 = 90$ để thực hiện hướng di chuyển sang phải với hàm rotate.

Hàm ROTATE (Dòng 96-100):

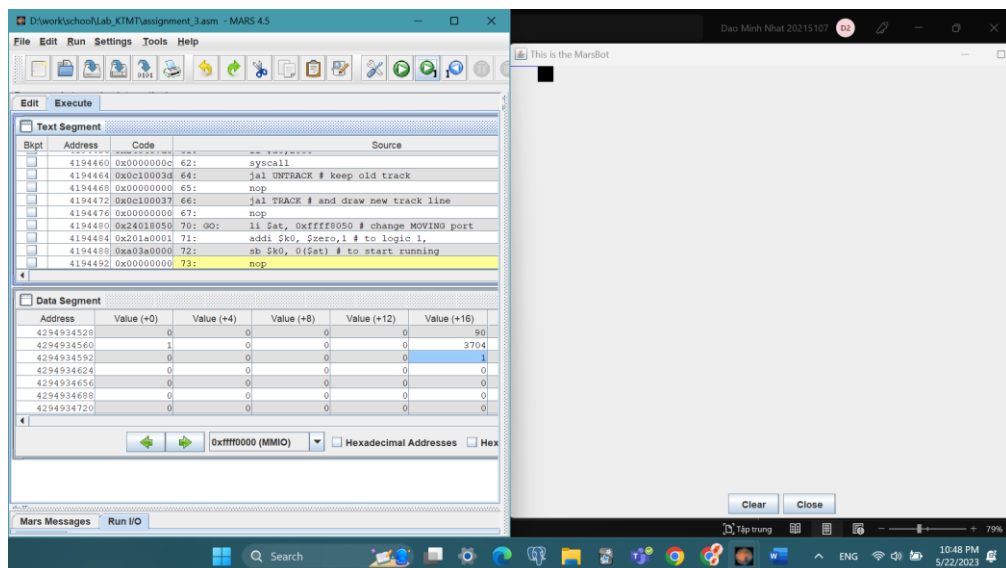
Thay đổi đến cổng HEADING

Dòng 20: Thực hiện hàm GO để bắt đầu chạy

Hàm GO (dòng 70-75):

Thay đổi đến cổng MOVING, và truyền vào logic 1 ($\$k0 = 1$) để bắt đầu di chuyển

⇒ Sau khi thực hiện 3 hàm TRACK để vẽ đường, ROTATE (90) để chọn hướng di chuyển sang phải và GO để chạy thì ta sẽ thấy con robot di chuyển sang bên phải với vạch kẻ xanh trên đường đã di chuyển qua.



Dòng 23-25: hàm sleep với $v0 = 32$ và $a0 = 1000$ sẽ giúp cho chương trình nghỉ 1000ms.

Dòng 27: Thực hiện hàm UNTRACK để bỏ đường kẻ.

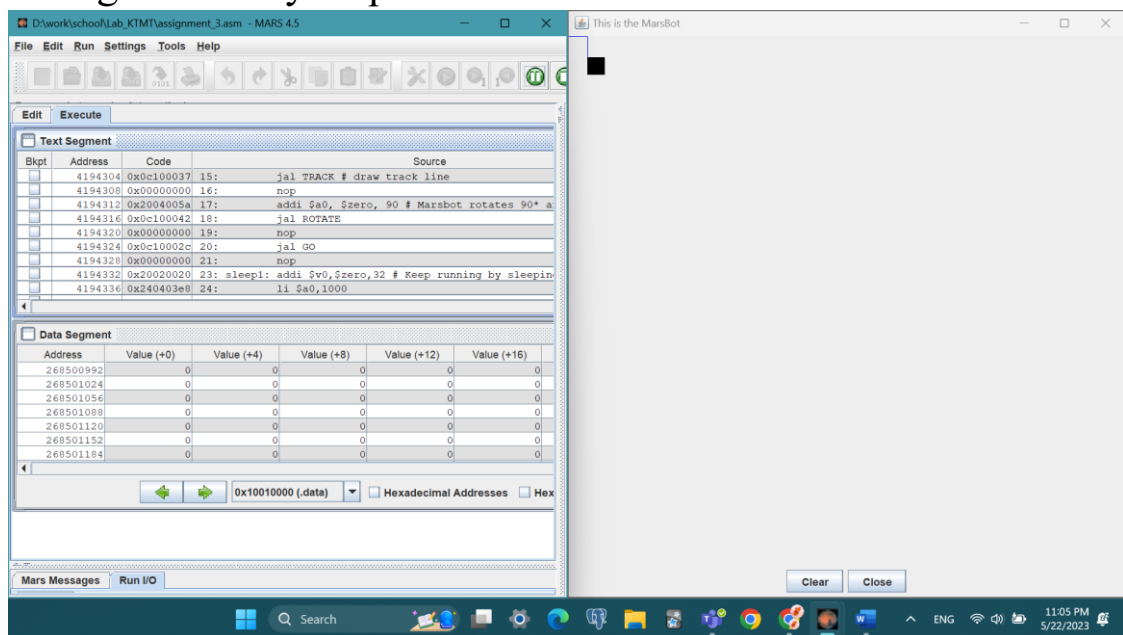
Hàm UNTRACK:

Dòng 90-94: Thay đổi đến cổng LEAVETRACK và truyền vào logic 0 để dừng in ra đường kẻ.

Dòng 29: Thực hiện hàm TRACK để vẽ đường kẻ.

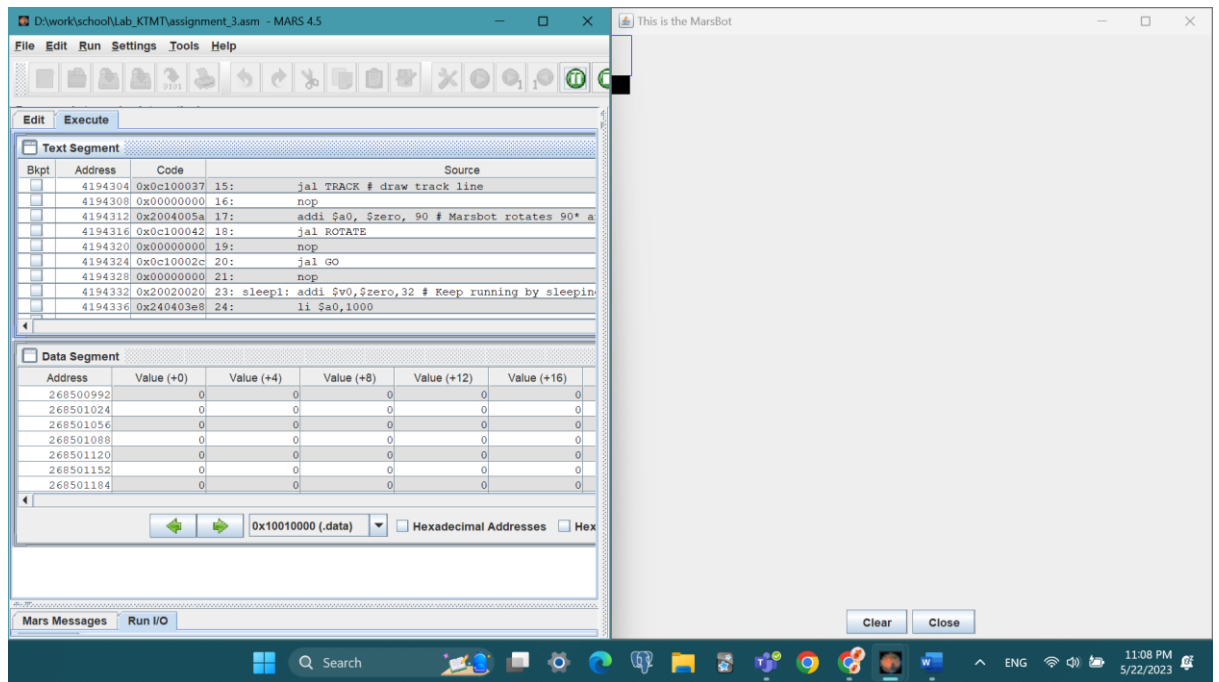
Dòng 32: Thực hiện hàm goDOWN với hàm ROTATE tham số truyền vào là $\$a0 = 180$ để robot di chuyển xuống dưới.

⇒ Sau khi thực hiện hàm sleep1 để có thời gian nghỉ để chuyển hướng, ROTATE (180) để chọn hướng di chuyển xuống dưới thì ta sẽ thấy con robot di chuyển xuống dưới với vạch kẻ xanh trên đường đã di chuyển qua.

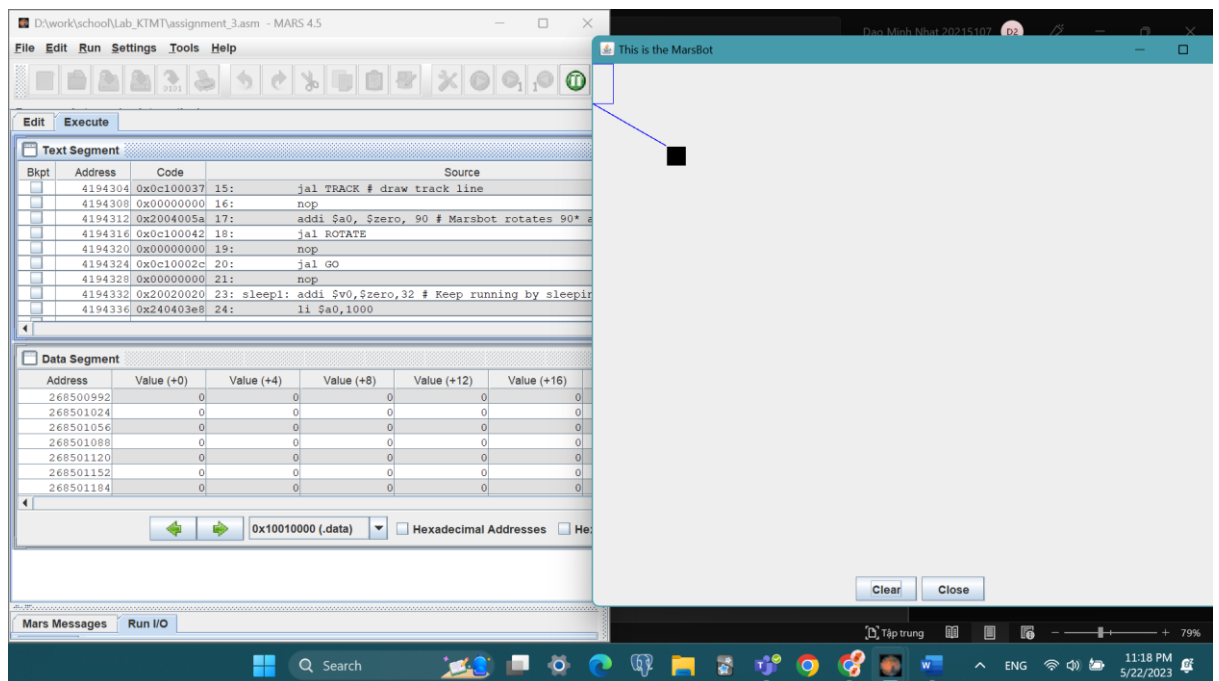


Dòng 36-46: tương tự với lệnh sleep2 (2000ms thời gian nghỉ) và goLEFT(với ROTATE (270) để di chuyển sang trái)

⇒ Khi đó robot sẽ di chuyển sang trái với đường kẻ.



Dòng 48-58: Hàm sleep3 và goASEW(với ROTATE (120)), thì robot sẽ di chuyển theo đường chéo từ trên xuống dưới và từ trái sang phải



Assignment 4

Code:

```
1  # Lab 10, Assignment 4
2  .eqv KEY_CODE 0xFFFF0004 # ASCII code from keyboard, 1 byte
3  .eqv KEY_READY 0xFFFF0000 # =1 if has a new keycode ?
4  # Auto clear after lw
5  .eqv DISPLAY_CODE 0xFFFF000C # ASCII code to show, 1 byte
6  .eqv DISPLAY_READY 0xFFFF0008 # =1 if the display has already to do
7  # Auto clear after sw
8  .text
9      li $k0, KEY_CODE
10     li $k1, KEY_READY
11
12     li $s0, DISPLAY_CODE
13     li $s1, DISPLAY_READY
14 loop: nop
15
16 WaitForKey:
17     lw $t1, 0($k1) # $t1 = [$k1] = KEY_READY
18     nop
19     beq $t1, $zero, WaitForKey # if $t1 == 0 then Polling
20     nop
21
22 ReadKey:
23     lw $t0, 0($k0) # $t0 = [$k0] = KEY_CODE
24     nop
25     beq $t0, 'd', Exit
26
27 WaitForDis:
28     lw $t2, 0($s1) # $t2 = [$s1] = DISPLAY_READY
29     nop
30     beq $t2, $zero, WaitForDis # if $t2 == 0 then Polling
31     nop
32
33 Encrypt:
34     addi $t0, $t0, 1 # change input key
35
36 ShowKey:
37     sw $t0, 0($s0) # show key
38     nop
39
40     j loop
41     nop
42 Exit:
43     li $v0, 10
44     syscall
```

Giải thích:

Hàm WaitForKey là hàm kiểm tra xem đã có gì được nhập chưa.

Hàm ReadKey là để đọc kí tự vừa nhập, và kiểm tra xem kí tự nhập vào có phải là kí tự ‘d’ không nếu có thì chuyển tới exit.

Hàm WaitForDis để kiểm tra xem kí tự đã có thể hiển thị chưa.

Hàm Encrypt để tăng giá trị vừa nhập vào 1 rồi in ra màn hình.

Kết quả:

The image shows a MIPS simulator interface with two main windows. The left window displays assembly code for a program, and the right window shows a keyboard input simulation.

Text Segment:

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c01ffff	lui \$1, 0xffffffff	8: li \$k0, 0xffffffff
	0x00400004	0x343a0004	ori \$26, \$1, 0x00000004	
	0x00400008	0x3c01ffff	lui \$1, 0xffffffff	9: li \$k1, 0xffffffff
	0x0040000c	0x343b0000	ori \$27, \$1, 0x00000000	
	0x00400010	0x3c01ffff	lui \$1, 0xffffffff	11: li \$a0, 0xffffffff
	0x00400014	0x3430000c	ori \$16, \$1, 0x0000000c	
	0x00400018	0x3c01ffff	lui \$1, 0xffffffff	12: li \$s1, 0xffffffff
	0x0040001c	0x34310008	ori \$17, \$1, 0x00000008	
	0x00400020	0x00000000	nop	13: loop: nop
	0x00400024	0xef900000	lw \$9, 0x00000000(\$27)	14: lw \$t1, 0(\$k1)
	0x00400028	0x00000000	nop	17: nop
	0x0040002c	0x1120ffff	beq \$9, \$0, 0xffffffff	18: beq \$t1, \$zero, loop
	0x00400030	0x00000000	nop	19: nop
	0x00400034	0xef400000	lw \$8, 0x00000000(\$26)	22: lw \$t0, 0(\$k0)
	0x00400038	0x00000000	nop	23: nop
	0x0040003c	0x20010064	addi \$1, \$0, 0x00000064	24: beq \$t0, 'd', Exit
	0x00400040	0x10280000	beq \$1, \$8, 0x00000000	

Data Segment:

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000

Keyboard and Display MMIO Simulator, Version 1.4

DISPLAY: Store to Transmitter Data 0xffff000c, cursor 8, area 95 x 10

h ifmpt

Font ☒ DAD Fixed transmitter delay, select using slider Delay length: 5 instruction executions

KEYBOARD: Characters typed here are stored to Receiver Data 0xffff0004

hello

Tool Control: Disconnect from MIPS, Reset, Help, Close