# Laboratory Exercise 10

## *Goals*

After this laboratory exercise, you should understand the method to control peripheral devices via simulators.

## *Literature*

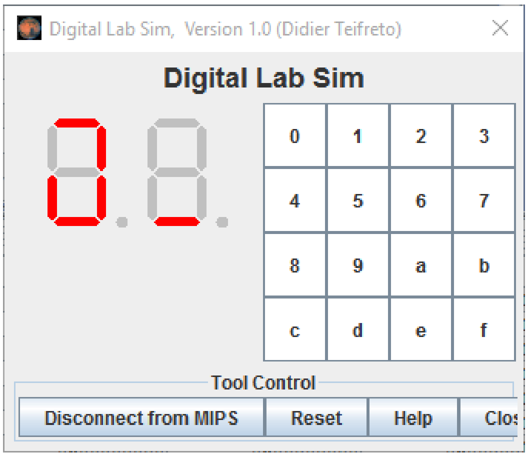
How does the CPU communicate with input and output devices such as the monitor or keyboard?

There are several ways. Intel machines have special instructions named in and out that communicate with I/O ports. These instructions are usually disabled for ordinary users, but they are used internally for communicating with I/O devices. This is called port-mapped I/O. However, we are going to look at a different method in which I/O devices have access to memory. The CPU can place data in memory that can be read by the I/O devices; likewise, the I/O devices can place data in memory for the CPU. This is called memory-mapped I/O or MMIO. (For more information, see P&H page 588 or Appendix B.8, or look it up online!)

## *Assignments at Home and at Lab*

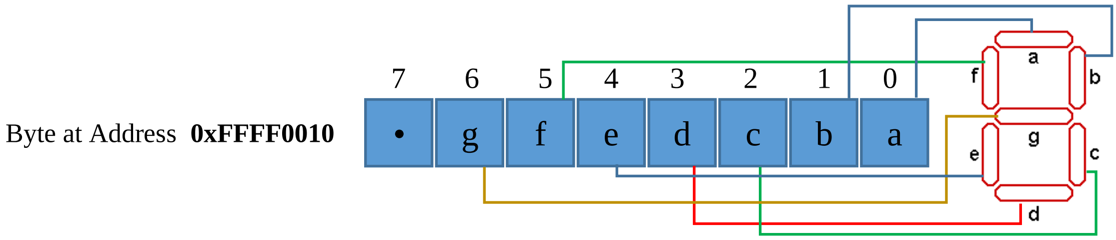
### Home Assignment 1 – LED PORT

Write a program using assembly language to show numbers from 0 to F to the 7-seg led.




To view the 7-segs, at the menu bar, click /Tools/Digi Lab Sim

Click Help to understand how to turn on the 7-seg led.

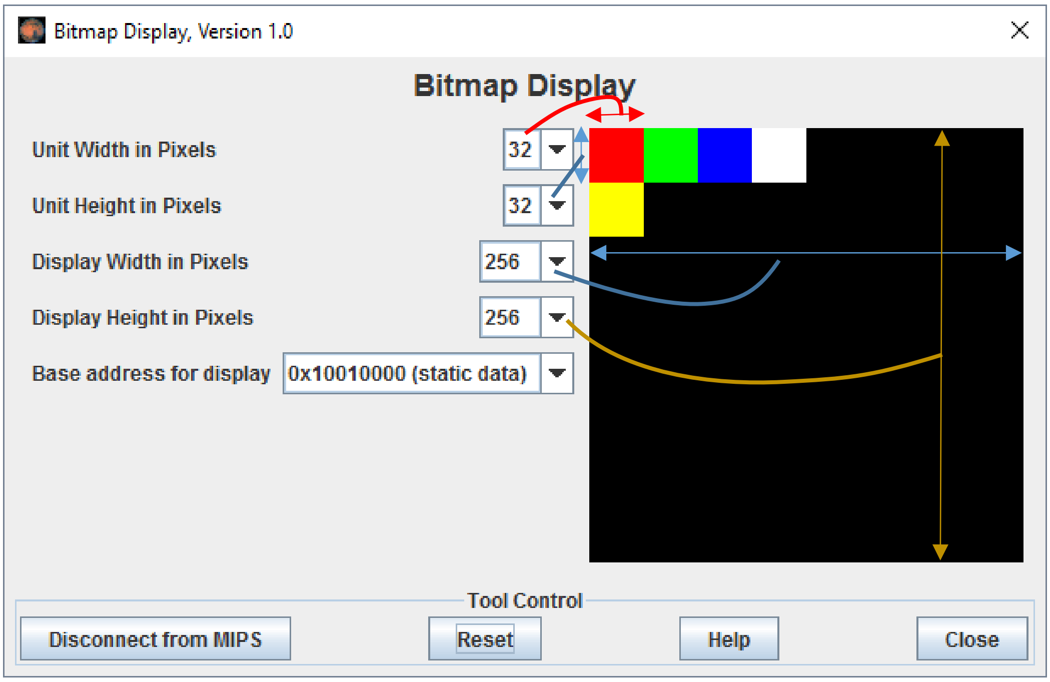



```
.eqv SEVENSEG_LEFT    0xFFFF0011    # Dia chi cua den led 7 doan trai.
                                    #     Bit 0 = doan a;
                                    #     Bit 1 = doan b; ...
                                    #     Bit 7 = dau .
.eqv SEVENSEG_RIGHT   0xFFFF0010    # Dia chi cua den led 7 doan phai

.text
main:
          li     $a0,  0x8                 # set value for segments
          jal    SHOW_7SEG_LEFT            # show
          nop
          li     $a0,  0x1F                # set value for segments
          jal    SHOW_7SEG_RIGHT           # show
          nop
exit:     li     $v0, 10
          syscall
endmain:

#---------------------------------------------------------------
# Function  SHOW_7SEG_LEFT : turn on/off the 7seg
# param[in]  $a0   value to shown
# remark     $t0 changed
#---------------------------------------------------------------
SHOW_7SEG_LEFT:  li   $t0,  SEVENSEG_LEFT # assign port's address
                 sb   $a0,  0($t0)        # assign new value
                 nop
                 jr   $ra
                 nop

#---------------------------------------------------------------
# Function  SHOW_7SEG_RIGHT : turn on/off the 7seg
# param[in]  $a0   value to shown
# remark     $t0 changed
#---------------------------------------------------------------
SHOW_7SEG_RIGHT: li   $t0,  SEVENSEG_RIGHT # assign port's address
                 sb   $a0,  0($t0)         # assign new value
                 nop
                 jr   $ra
                 nop
```

## Home Assignment 2 – BITMAP DISPLAY

Bitmap Display like the graphic monitor, in which Windows OS draws windows, start button… To do that, developer should calculate color of all bitmap pixels on the screen and store these color value to the screen memory. Wherever we change a value in screen memory, the color of the respective pixel on the screen will be changed.

In MARS, in menu bar, click Tools / Bitmap Display to open the screen simulator.




b

| 0 | R | G | B | |
|---|---|---|---|---|
| 00 | FF | 00 | 00 | 0x10010000 - pixel 0 |
| 00 | 00 | FF | 00 | 0x10010004 - pixel 1 |
| 00 | 00 | 00 | 00 | 0x10010008 - pixel 2 |
| 00 | FF | FF | FF | 0x1001000C - pixel 3 |

Each rectangular unit on the display represent s one memory word in a contiguous address space starting with the specified base address (in above figure, base address is 0x10010000)

Value stored in that word will be interpreted as a 24-bit RGB

```
.eqv MONITOR_SCREEN 0x10010000    #Dia chi bat dau cua bo nho man hinh
.eqv RED            0x00FF0000    #Cac gia tri mau thuong su dung
.eqv GREEN          0x0000FF00
.eqv BLUE           0x000000FF
.eqv WHITE          0x00FFFFFF
.eqv YELLOW         0x00FFFF00
.text
   li $k0, MONITOR_SCREEN         #Nap dia chi bat dau cua man hinh

   li $t0, RED
   sw  $t0, 0($k0)
   nop

   li $t0, GREEN
   sw  $t0, 4($k0)
   nop

   li $t0, BLUE
   sw  $t0, 8($k0)
   nop

   li $t0, WHITE
   sw  $t0, 12($k0)
```

```
   nop

   li $t0, YELLOW
   sw  $t0, 32($k0)
   nop

   li $t0, WHITE
   lb $t0, 42($k0)
   nop
```

## Home Assignment 3 – MARSBOT RIDER

The MarsBot is a virtual robot that has a very simple mode of operation. It travels around in two-dimensional space, optionally leaving a trail, or track, as it goes. It uses five words in memory:[1]

| Name | Address | Meaning |
|---|---|---|
| HEADING | 0xffff8010 | Integer: An angle between 0 and 359 |
| LEAVETRACK | 0xffff8020 | Boolean (0 or non-0): whether or not to leave a track |
| WHEREX | 0xffff8030 | Integer: Current x-location of the MarsBot |
| WHEREY | 0xffff8040 | Integer: Current y-location of the MarsBot |
| MOVING | 0xffff8050 | Boolean: whether or not to move |

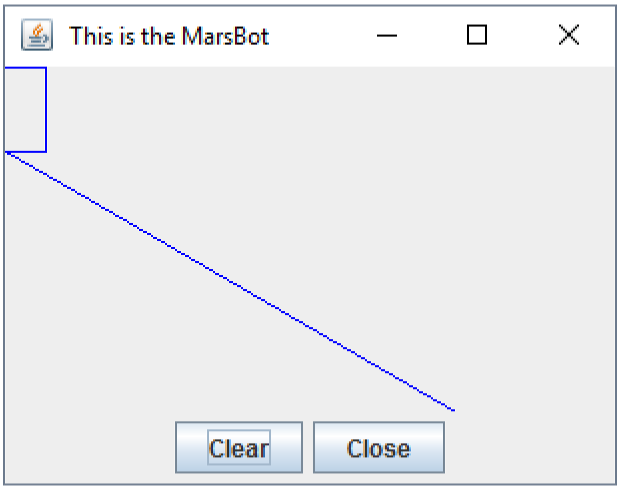
The CPU can place commands in the HEADING, LEAVETRACK, and MOVE locations; the robot can then change its direction of travel (using the HEADING value), turn on or turn off the pen" drawing the line (using the LEAVE-TRACK value), and can halt or resume moving (using the MOVING value).

```
.eqv  HEADING    0xffff8010    # Integer: An angle between 0 and 359
                               # 0 : North (up)
                               # 90: East (right)
                               # 180: South (down)
                               # 270: West  (left)
.eqv  MOVING     0xffff8050    # Boolean: whether or not to move
.eqv  LEAVETRACK 0xffff8020    # Boolean (0 or non-0):
                               #    whether or not to leave a track
.eqv  WHEREX     0xffff8030    # Integer: Current x-location of
MarsBot
.eqv  WHEREY     0xffff8040    # Integer: Current y-location of
MarsBot


.text
main:    jal     TRACK           # draw track line
         nop
         addi    $a0, $zero, 90  # Marsbot rotates 90* and start
running
         jal     ROTATE
         nop
         jal     GO
         nop
```

[1] http://cs.allegheny.edu/~rroos/cs210f2013

```
sleep1: addi    $v0,$zero,32    # Keep running by sleeping in 1000 ms
        li      $a0,1000
        syscall

        jal     UNTRACK         # keep old track
        nop
        jal     TRACK           # and draw new track line
        nop

goDOWN: addi    $a0, $zero, 180 # Marsbot rotates 180*
        jal     ROTATE
        nop

sleep2: addi    $v0,$zero,32    # Keep running by sleeping in 2000 ms
        li      $a0,2000
        syscall

        jal     UNTRACK         # keep old track
        nop
        jal     TRACK           # and draw new track line
        nop

goLEFT: addi  $a0, $zero, 270   # Marsbot rotates 270*
        jal   ROTATE
        nop

sleep3: addi    $v0,$zero,32    # Keep running by sleeping in 1000 ms
        li      $a0,1000
        syscall

        jal     UNTRACK         # keep old track
        nop
        jal     TRACK           # and draw new track line
        nop

goASKEW:addi  $a0, $zero, 120   # Marsbot rotates 120*
        jal   ROTATE
        nop

sleep4: addi    $v0,$zero,32    # Keep running by sleeping in 2000 ms
        li      $a0,2000
        syscall

        jal     UNTRACK         # keep old track
        nop
        jal     TRACK           # and draw new track line
        nop
end_main:

#-----------------------------------------------------------
# GO procedure, to start running
# param[in]    none
#-----------------------------------------------------------
GO:     li    $at, MOVING     # change MOVING port
        addi  $k0, $zero,1    # to  logic 1,
        sb    $k0, 0($at)     # to start running
        nop
        jr    $ra
        nop
#-----------------------------------------------------------
# STOP procedure, to stop running
# param[in]    none
#-----------------------------------------------------------
```

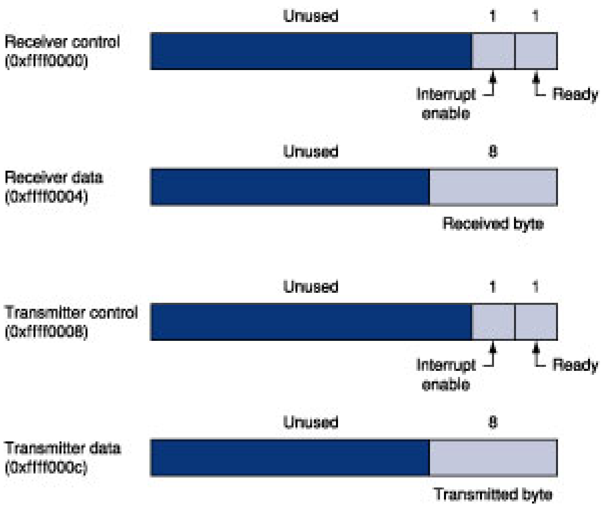
```
STOP:    li     $at, MOVING      # change MOVING port to 0
         sb     $zero, 0($at)    # to stop
         nop
         jr     $ra
         nop
#-----------------------------------------------------------
# TRACK procedure, to start drawing line
# param[in]     none
#-----------------------------------------------------------
TRACK:   li     $at, LEAVETRACK # change LEAVETRACK port
         addi   $k0, $zero,1     # to  logic 1,
         sb     $k0, 0($at)      # to start tracking
         nop
         jr     $ra
         nop
#-----------------------------------------------------------
# UNTRACK procedure, to stop drawing line
# param[in]     none
#-----------------------------------------------------------
UNTRACK:li     $at, LEAVETRACK # change LEAVETRACK port to 0
         sb     $zero, 0($at)    # to stop drawing tail
         nop
         jr     $ra
         nop
#-----------------------------------------------------------
# ROTATE procedure, to rotate the robot
# param[in]     $a0, An angle between 0 and 359
#                    0 : North (up)
#                    90: East  (right)
#                   180: South (down)
#                   270: West  (left)
#-----------------------------------------------------------
ROTATE: li     $at, HEADING     # change HEADING port
         sw     $a0, 0($at)      # to rotate robot
         nop
         jr     $ra
         nop
```



## Home Assignment 4 – KEYBOARD and DISPLAY MMIO

Use this program to simulate Memory-Mapped I/O (MMIO) for a keyboard input device and character display output device. It may be run either from MARS' Tools menu or as a stand-alone application.

While the tool is connected to MIPS, each keystroke in the text area causes the corresponding ASCII code to be placed in the Receiver Data register (low-order byte of memory word 0xffff0004), and the Ready bit to be set to 1 in the Receiver Control register (low-order bit of 0xffff0000). The Ready bit is automatically reset to 0 when the MIPS program reads the Receiver Data using an 'lw' instruction.

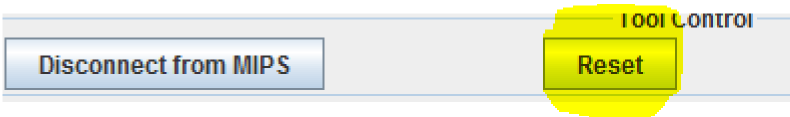



```
.eqv KEY_CODE    0xFFFF0004        # ASCII code from keyboard, 1 byte
.eqv KEY_READY   0xFFFF0000        # =1 if has a new keycode ?
                                   # Auto clear after lw

.eqv DISPLAY_CODE    0xFFFF000C    # ASCII code to show, 1 byte
.eqv DISPLAY_READY   0xFFFF0008    # =1 if the display has already to do
                                   # Auto clear after sw

.text
             li   $k0,  KEY_CODE
             li   $k1,  KEY_READY

             li   $s0, DISPLAY_CODE
             li   $s1, DISPLAY_READY

loop:        nop

WaitForKey:  lw   $t1, 0($k1)              # $t1 = [$k1] = KEY_READY
             nop
             beq  $t1, $zero, WaitForKey # if $t1 == 0 then Polling
             nop
             #-----------------------------------------------------
ReadKey:     lw   $t0, 0($k0)              # $t0 = [$k0] = KEY_CODE
             nop
             #-----------------------------------------------------
WaitForDis:  lw   $t2, 0($s1)              # $t2 = [$s1] = DISPLAY_READY
              nop
```

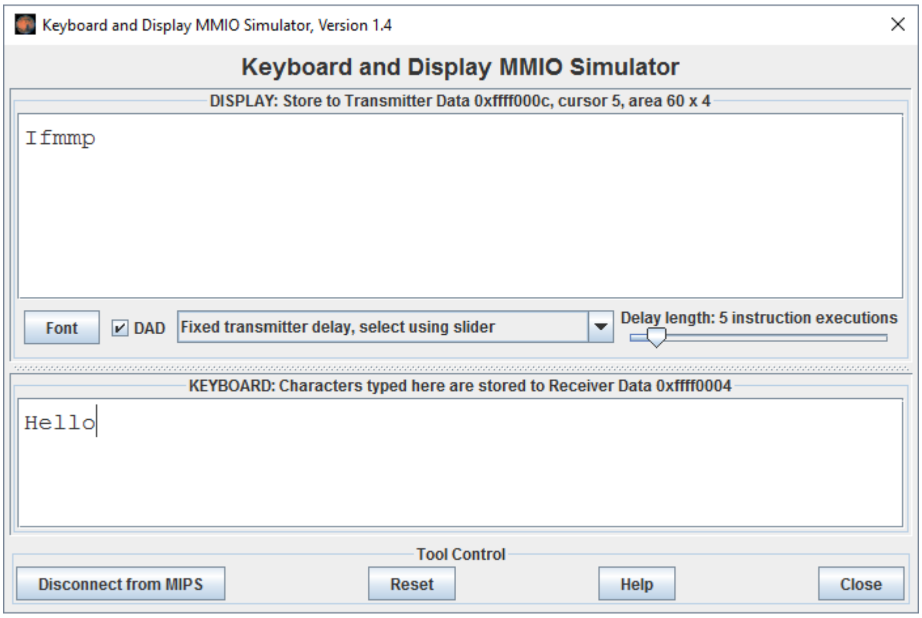
```
            beq  $t2, $zero, WaitForDis # if $t2 == 0 then Polling
            nop
            #----------------------------------------------------
Encrypt:    addi $t0, $t0, 1            # change input key
            #----------------------------------------------------
ShowKey:    sw $t0, 0($s0)              # show key
            nop
            #----------------------------------------------------
            j loop
            nop
```

**Warning: Must execute as below**

1. Click Run 




2. Click Reset




## Assignment 1

Create a new project, type in, and build the program of Home Assignment 1. Show different values on LED

## Assignment 2

Create a new project, type in, and build the program of Home Assignment 2. Draw something.

## Assignment 3

Create a new project, type in, and build the program of Home Assignment 3. Make the Bot run and draw a triangle by tracking.

## Assignment 4

Create a new project, type in, and build the program of Home Assignment 4. Read key char and terminate the application when receiving “exit” command.