# Lab 10. Peripheral devices

## *Goals*

After this laboratory excersice, students should understand the method to control peripheral devices via simulation tools.

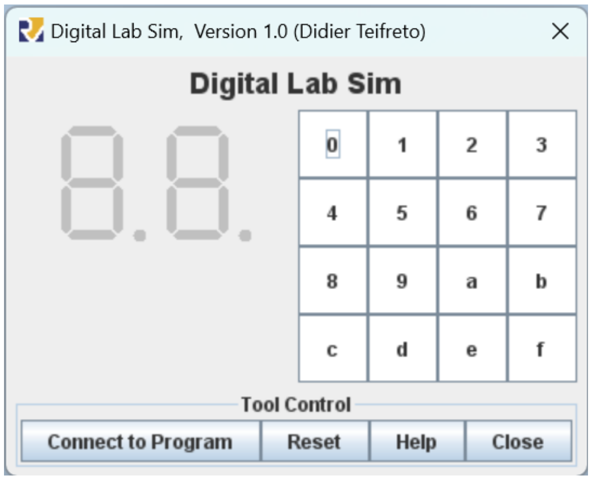
## *Literature*

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.

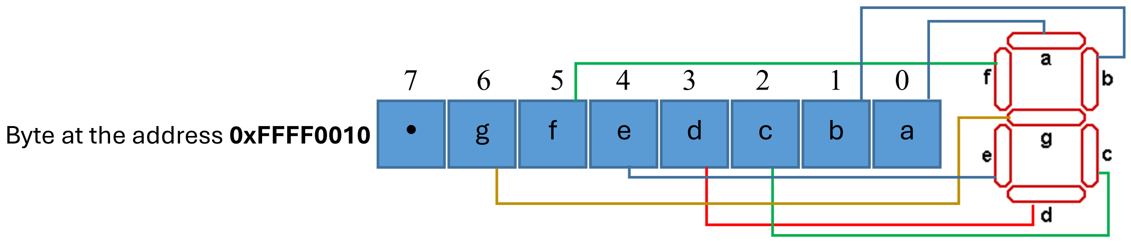
## Home Assignment 1 – LED PORT

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



To open the 7-segment LEDs, click Tools/Digital Lab Sim at the menu bar.

Open Help to understand how to manipulate the 7-segment LEDs.

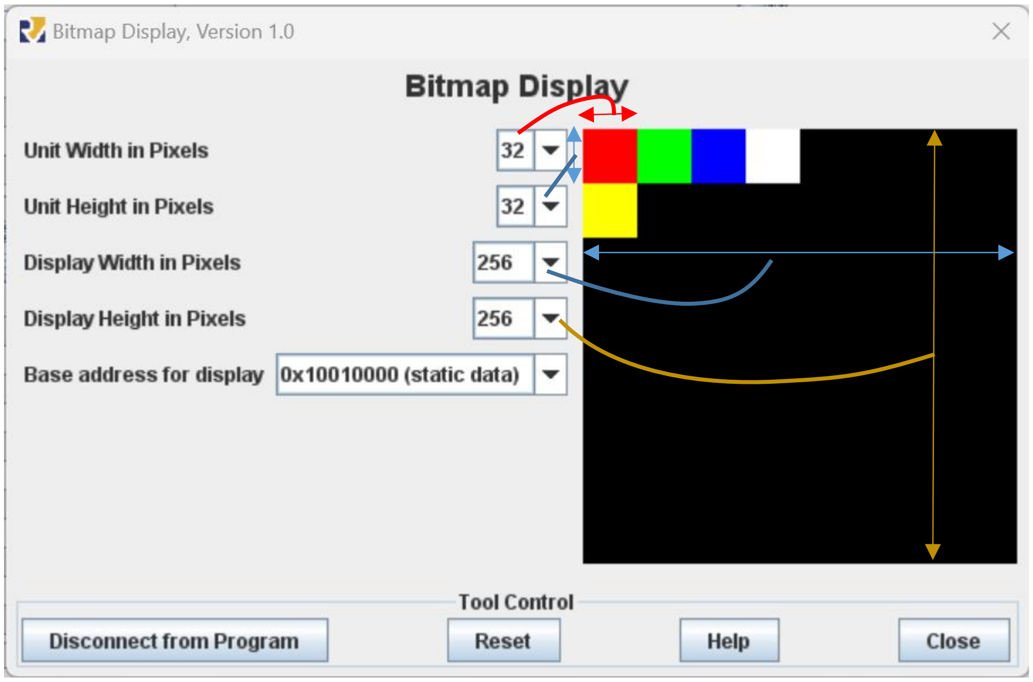


```
.eqv SEVENSEG_LEFT    0xFFFF0011    # Address of the LED on the left
                                    #     Bit 0 = segment a
                                    #     Bit 1 = segment b
                                    #     ...
                                    #     Bit 7 = dot sign
.eqv SEVENSEG_RIGHT   0xFFFF0010    # Address of the LED on the right

.text
main:
    li      a0,  0x06               # Set value for 7 segments
    jal     SHOW_7SEG_LEFT          # Show the result
    li      a0,  0x3F               # Set value for 7 segments
    jal     SHOW_7SEG_RIGHT         # Show the result
exit:
    li      a7, 10
    ecall
end_main:

# ---------------------------------------------------------------
# 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
    jr      ra

# ---------------------------------------------------------------
# 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
    jr   ra
```

## Home Assignment 2 – BITMAP DISPLAY

Bitmap Display is similar with a graphic monitor, in which Windows OS draws windows, start button… To do that, developers should calculate color of all bitmap pixels on thee 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 RARS, in the menu bar, click Tools / Bitmap Display to open the screen simulator.



Unit Width in Pixels: 32
Unit Height in Pixels: 32
Display Width in Pixels: 256
Display Height in Pixels: 256
Base address for display: 0x10010000 (static data)

| 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 represents one memory word in a contiguous address starting with the specified base address (in above figure, base address is 0x10010000).

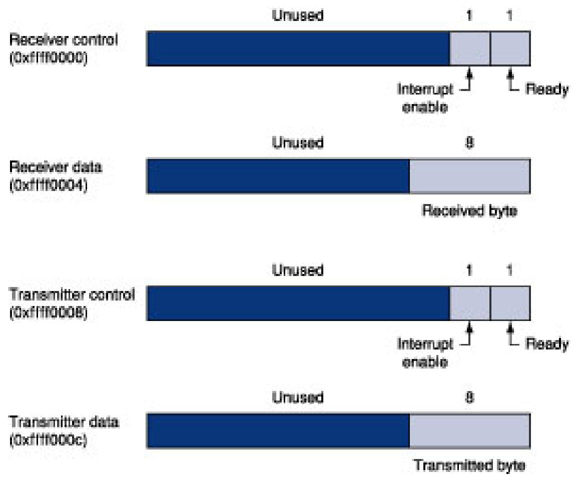
The value stored in that word will be interpreted as a 24-bit RGB.

```
.eqv MONITOR_SCREEN 0x10010000  # Start address of the bitmap display
.eqv RED            0x00FF0000  # Common color values
.eqv GREEN          0x0000FF00
.eqv BLUE           0x000000FF
.eqv WHITE          0x00FFFFFF
.eqv YELLOW         0x00FFFF00
.text
    li   a0, MONITOR_SCREEN     # Load address of the display

    li   t0, RED
    sw   t0, 0(a0)

    li   t0, GREEN
    sw   t0, 4(a0)
    li   t0, BLUE
    sw   t0, 8(a0)

    li   t0, WHITE
    sw   t0, 12(a0)

    li   t0, YELLOW
    sw   t0, 32(a0)
```

## Home Assignment 3 – 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 RARS' Tools menu or as a stand-alone application.

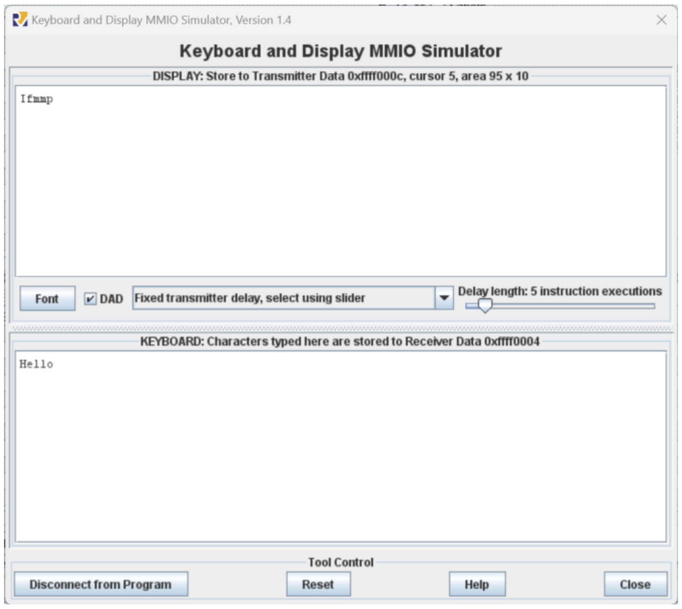
While the tool is connected to the program, each keystroke in the text area causes the corresponding ASCII code to be placed in the Receiver Data register (low-order byte of memory word 0xffff0004), and the Ready bit to be set to 1 in the Receiver Control register (low-order bit of 0xffff0000). The Ready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


Receiver control (0xffff0000)
Unused
1
1
Interrupt enable
Ready
Receiver data (0xffff0004)
Unused
8
Received byte
Transmitter control (0xffff0008)
Unused
1
1
Interrupt enable
Ready
Transmitter data (0xffff000c)
Unused
8
Transmitted byte

```
.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   a0, KEY_CODE
    li   a1, KEY_READY
    li   s0, DISPLAY_CODE
    li   s1, DISPLAY_READY
```

```
loop:
WaitForKey:
    lw      t1, 0(a1)               # t1 = [a1] = KEY_READY
    beq     t1, zero, WaitForKey    # if t1 == 0 then Polling
ReadKey:
    lw      t0, 0(a0)               # t0 = [a0] = KEY_CODE
WaitForDis:
    lw      t2, 0(s1)               # t2 = [s1] = DISPLAY_READY
    beq     t2, zero, WaitForDis    # if t2 == 0 then polling
Encrypt:
    addi    t0, t0, 1               # change input key
ShowKey:
    sw      t0, 0(s0)               # show key
    j       loop
```

The expected result of the execution:



Keyboard and Display MMIO Simulator, Version 1.4

Keyboard and Display MMIO Simulator

DISPLAY: Store to Transmitter Data 0xffff000c, cursor 5, area 95 x 10

```
Ifmmp
```

[✓] DAD — Fixed transmitter delay, select using slider — Delay length: 5 instruction executions

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

```
Hello
```

## Assignment 1

Implement the program in Home Assignment 1, change the values so that the last two digits of StudentID will be displayed on the LEDs.

## Assignment 2

Write a program that lets user enter a character from the keyboard and the program will print the last two digits of the ASCII code of the characters.

## Assignment 3

Implement the program in Home Assignment 2, and then update the code so that it can draw a chess board.

## Assignment 4

Implement the program in Home Assignment 3, then update the code so that it can be executed as follows:
Enter a lowercase character => Display the corresponding uppercase character.
Enter an uppercase character => Display the corresponding lowercase character.
Enter a digit => Display the same digit
Enter another character => Display “*”
The program will be exited if “exit” is entered.

## Assignment 5

Write a program that allows the user to enter 2 points with coordinates (x1, y1) and (x2, y2) (x1 is different from x2 and y1 is different from y2), draw and color a rectangle with 2 corners being the 2 entered points with a red border 1 unit wide and a green background. For example, with (x1, y1) = (3, 3) and (x2, y2) = (18, 11), or (x1, y1) = (3, 11) and (x2, y2) = (18, 3), we will have the result as the following figure.