HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Computer Architecture Lab Final Project

Group 11

Supervised by:

Master Le Ba Vui

Assistant Teaching:

Nguyen Trung Kien

Group Members:

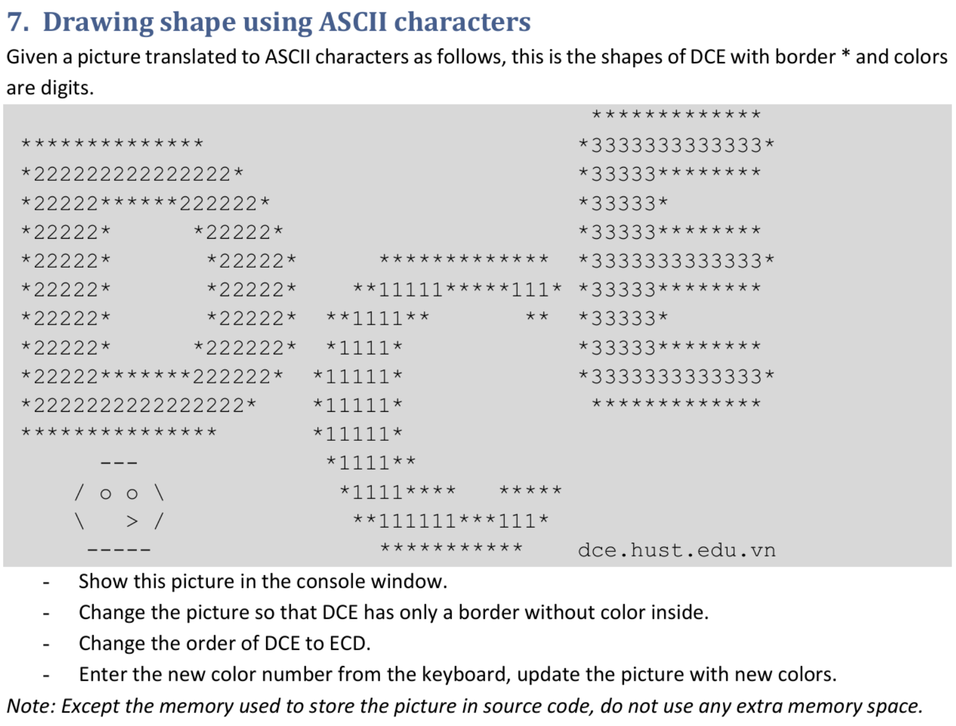
Pham Duy Hoang - 20226042

Nguyen Cong Minh - 20226056

Contents

1 Drawing shape using ASCII characters

1.1 Description:



7. Drawing shape using ASCII characters

Given a picture translated to ASCII characters as follows, this is the shapes of DCE with border * and colors are digits.

```
                                           *************
**************                            *3333333333333*
*222222222222222*                         *33333********
*22222******222222*                       *33333*
*22222*      *22222*                      *33333********
*22222*       *22222*      *************  *3333333333333*
*22222*       *22222*    **11111*****111* *33333********
*22222*       *22222*  **1111**       **  *33333*
*22222*      *222222*  *1111*             *33333********
*22222*******222222*  *11111*             *3333333333333*
*2222222222222222*    *11111*              *************
***************       *11111*
      ---              *1111**
    / o o \             *1111****   *****
    \   > /              **111111***111*
     -----                 ***********    dce.hust.edu.vn
```

- Show this picture in the console window.
- Change the picture so that DCE has only a border without color inside.
- Change the order of DCE to ECD.
- Enter the new color number from the keyboard, update the picture with new colors.

Note: Except the memory used to store the picture in source code, do not use any extra memory space.

Figure 1: Description for exercise 7

This picture and all the changes for this picture will be displayed on terminal screen while executing RISC-V code

1.2 Algorithms

1.2.1 Display the Picture in the Console Window

In the first task, we only need to print the picture as it is, without making any changes. The picture can be represented as 16 continuous strings, each with a size of 60 characters. To achieve this, the algorithm uses a single loop. During each iteration, the program prints the string stored at the address in `a0` by calling the system call `li a7, 4`. After printing, the program updates `t0 = t0 + 1` and `a0 = a0 + 60`, since the base addresses of two consecutive lines are separated by 60 (the size of each line). The loop terminates when `t0 = 16`.

1.2.2 Modify the Picture to Display Only the Border of `DCE`

This task requires printing the `DCE` pattern with only the border visible, removing the inner colored sections. Unlike the first task, we cannot print each line as-is. Instead, we consider the picture as a 16x60 matrix, where each cell stores a character. The algorithm employs a nested loop: the outer loop traverses the rows, and the inner loop traverses the columns of each row. Two registers are used to store the characters `'1'` and `'9'`, which help in checking whether a character represents a number (indicating color). For each character, if it is less than `'1'` or greater than `'9'`, the program prints it normally. Otherwise, it replaces the character with a space `' '` before printing it using system call `syscall 11`. This process ensures that only the border characters are printed.

1.2.3 Reorder the Picture from DCE to ECD

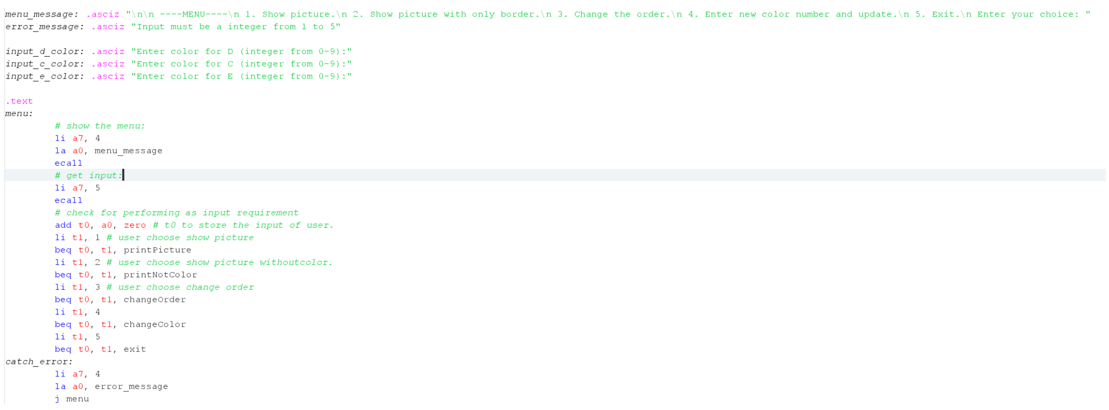
To solve this task, the focus is on the structure of the picture, specifically the starting and ending indices of each `'D'`, `'C'`, and `'E'` block, as well as the spaces separating them. The algorithm temporarily replaces the space characters (used as delimiters) with zeros to ensure that when printing a string from the base address, the process stops at the zero character. The program then prints each line in the new order: `'E'` (starting at index 43), `'C'` (starting at index 23), and `'D'` (starting at index 0). Between each block, it prints a space character `' '` to separate the words.

1.2.4 Update the Picture with New Colors Based on User Input

This task involves updating the `DCE` pattern with new colors based on user input. The user is prompted to enter a new color (a number between 0 and 9) for `D`, `C`, and `E`. If the input is out of range, the user is prompted to re-enter the value. The algorithm employs a nested loop: the outer loop traverses the rows, and the inner loop traverses the columns. For each column, if the index is less than 23, it updates the `D` block; if less than 43, it updates the `C` block; otherwise, it updates the `E` block. The user input is converted to its character equivalent by adding `'0'` before replacing the current character in the range `'0'` to `'9'`.

1.3 Code and Explanation

1.3.1 Menu for User Selection

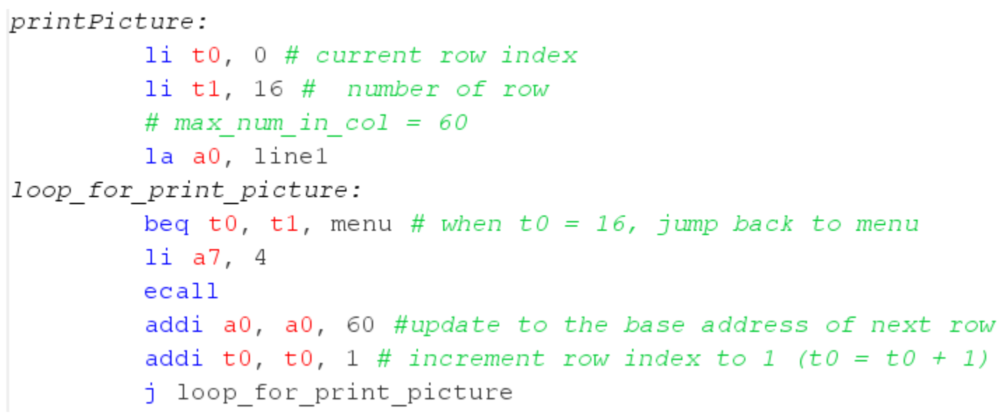


```
menu_message: .asciz "\n\n ----MENU----\n 1. Show picture.\n 2. Show picture with only border.\n 3. Change the order.\n 4. Enter new color number and update.\n 5. Exit.\n Enter your choice: "
error_message: .asciz "Input must be a integer from 1 to 5"

input_d_color: .asciz "Enter color for D (integer from 0-9):"
input_c_color: .asciz "Enter color for C (integer from 0-9):"
input_e_color: .asciz "Enter color for E (integer from 0-9):"

.text
menu:
        # show the menu:
        li a7, 4
        la a0, menu_message
        ecall
        # get input:
        li a7, 5
        ecall
        # check for performing as input requirement
        add t0, a0, zero # t0 to store the input of user.
        li t1, 1 # user choose show picture
        beq t0, t1, printPicture
        li t1, 2 # user choose show picture withoutcolor.
        beq t0, t1, printNotColor
        li t1, 3 # user choose change order
        beq t0, t1, changeOrder
        li t1, 4
        beq t0, t1, changeColor
        li t1, 5
        beq t0, t1, exit
catch_error:
        li a7, 4
        la a0, error_message
        j menu
```

Figure 2: Menu Screen

This code allows the user to select a task by entering a number from 1 to 5. Each number corresponds to a specific task. If the input is invalid (outside the range), the program jumps to the `catch_error` branch, displays the `error_message`, and returns to the menu branch.

1.3.2 Display the Original Picture



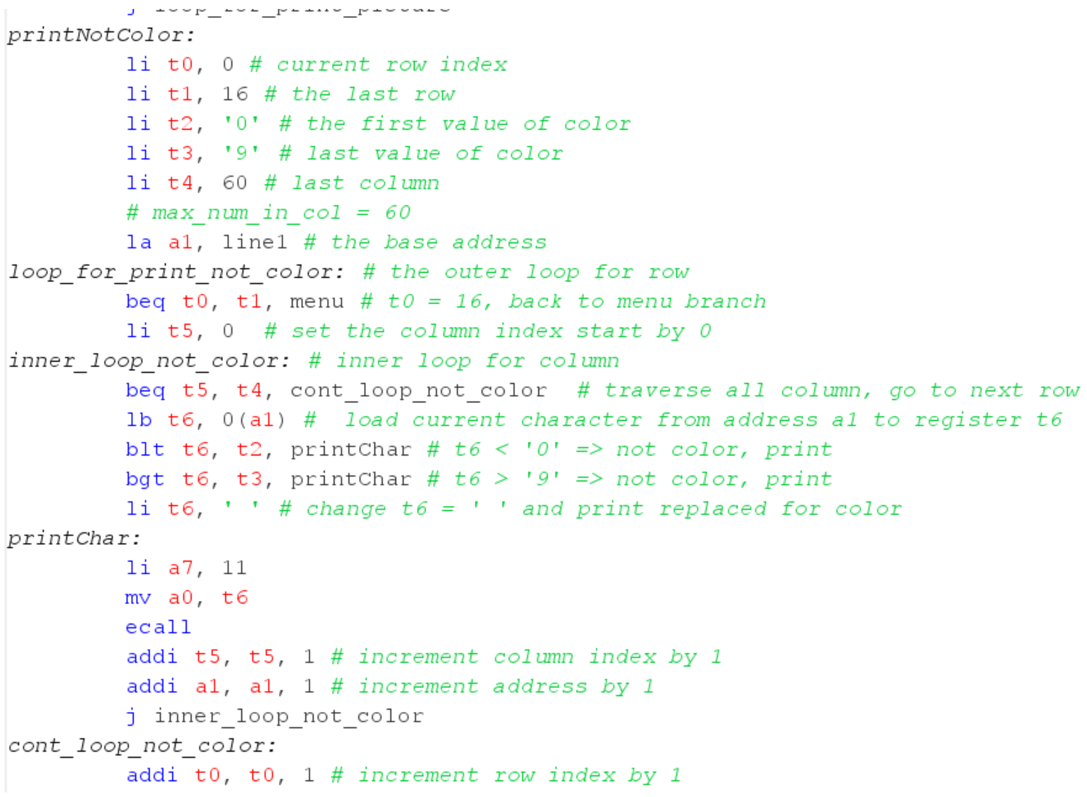
```
printPicture:
        li t0, 0 # current row index
        li t1, 16 #  number of row
        # max_num_in_col = 60
        la a0, line1
loop_for_print_picture:
        beq t0, t1, menu # when t0 = 16, jump back to menu
        li a7, 4
        ecall
        addi a0, a0, 60 #update to the base address of next row
        addi t0, t0, 1 # increment row index to 1 (t0 = t0 + 1)
        j loop_for_print_picture
```

Figure 3: Code for Displaying the Original Picture

This code uses a single loop to print the picture. It prints the string located at the base address `a0` and then updates `a0 = a0 + 60`, as each row is 60 bytes long. The counter `t0` is incremented by 1 during each iteration. The loop ends when `t0 = 16`, after which the program jumps back to the menu branch.

1.3.3 Display the Picture Without Colors

To perform this task, the program uses a nested loop. The outer loop traverses each row, while the inner loop processes each column in the current row. The register `t0` stores the current row, and `t5` stores the current column. For each character, if it is smaller than `'0'` or larger than `'9'`, the program directly jumps to `printChar`, which uses `syscall 11` to print the character. Otherwise, it assigns `t6 = ' '` and prints a space instead. The inner loop ends when `t5 = 60` (indicating the end of the row), and the outer loop terminates when `t0 = 16`. The program then jumps back to the menu to await the next user input.
Below is the detailed code:



```
        j loop_for_print_picture
printNotColor:
        li t0, 0 # current row index
        li t1, 16 # the last row
        li t2, '0' # the first value of color
        li t3, '9' # last value of color
        li t4, 60 # last column
        # max_num_in_col = 60
        la a1, line1 # the base address
loop_for_print_not_color: # the outer loop for row
        beq t0, t1, menu # t0 = 16, back to menu branch
        li t5, 0  # set the column index start by 0
inner_loop_not_color: # inner loop for column
        beq t5, t4, cont_loop_not_color  # traverse all column, go to next row
        lb t6, 0(a1) #  load current character from address a1 to register t6
        blt t6, t2, printChar # t6 < '0' => not color, print
        bgt t6, t3, printChar # t6 > '9' => not color, print
        li t6, ' ' # change t6 = ' ' and print replaced for color
printChar:
        li a7, 11
        mv a0, t6
        ecall
        addi t5, t5, 1 # increment column index by 1
        addi a1, a1, 1 # increment address by 1
        j inner_loop_not_color
cont_loop_not_color:
        addi t0, t0, 1 # increment row index by 1
```

Figure 4: Code for Printing Only the Border

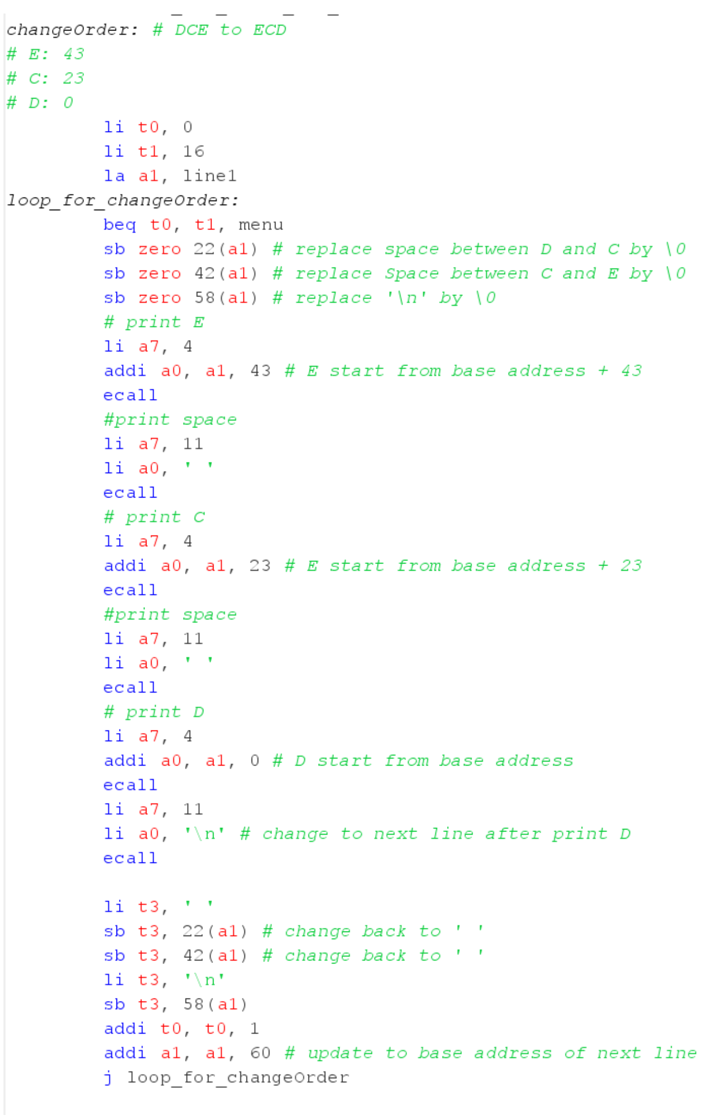
1.3.4 Reorder the Picture from `DCE` to `ECD`

This task involves rearranging the order of the letters `D`, `C`, and `E` in the picture so that they appear as `ECD`. To achieve this, we first analyze the structure of the picture:

The `'D'` block spans columns 0 to 21. The `'C'` block spans columns 23 to 41. The `'E'` block spans columns 43 to the end. The solution divides each line of the picture into three parts corresponding to these blocks. To handle the reordering efficiently, the program temporarily replaces the space characters (located at columns 22, 42, and 58) with zeros. This ensures that system call `syscall 4`, which prints strings, stops at these positions (interpreted as null terminators).

The algorithm proceeds as follows:

For each line, the program prints the sections in the new order: The `'E'` block, starting at `a1 + 43`. The `'C'` block, starting at `a1 + 22`. The `'D'` block, starting at `a1`. Between each block, a space character (`' '`) is printed for separation. After printing all three parts of a line, the program restores the original characters (spaces) at the positions temporarily replaced with zeros. The loop then updates the indices: `t0 = t0 + 1` (to move to the next row). `a1 = a1 + 60` (to move the base address to the next line). This process continues until all 16 rows of the picture are processed. The picture is then displayed in the desired `ECD` order. Below is the detailed code for this task.

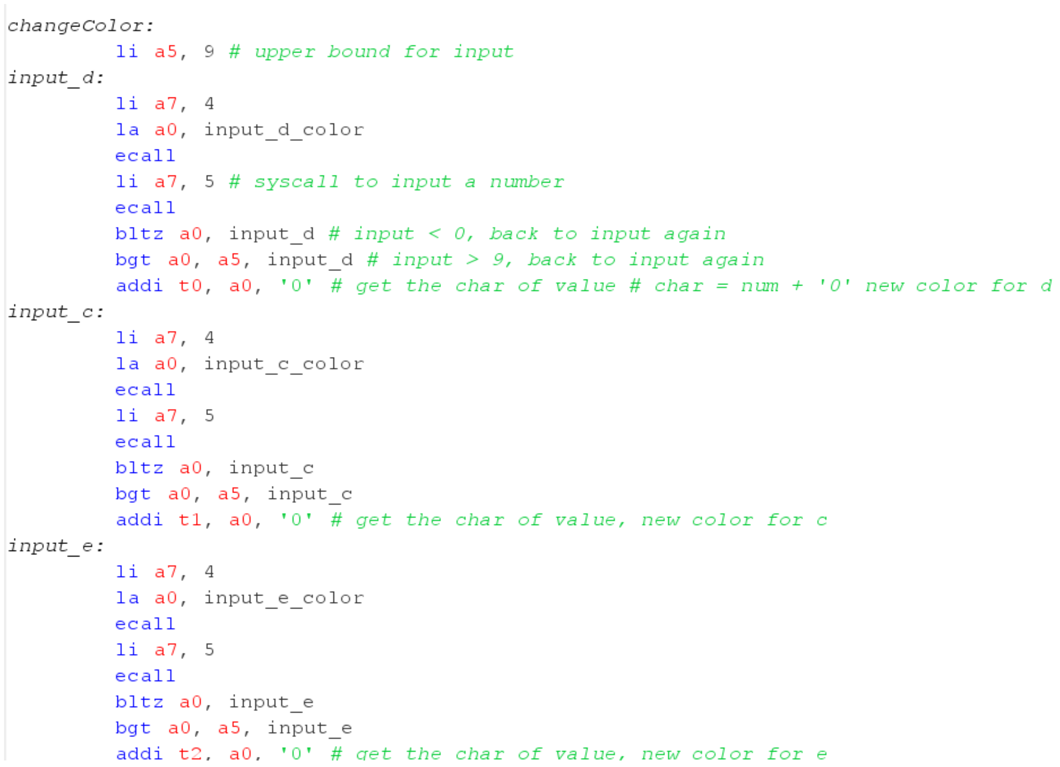


```
changeOrder: # DCE to ECD
# E: 43
# C: 23
# D: 0
        li t0, 0
        li t1, 16
        la a1, line1
loop_for_changeOrder:
        beq t0, t1, menu
        sb zero 22(a1) # replace space between D and C by \0
        sb zero 42(a1) # replace Space between C and E by \0
        sb zero 58(a1) # replace '\n' by \0
        # print E
        li a7, 4
        addi a0, a1, 43 # E start from base address + 43
        ecall
        #print space
        li a7, 11
        li a0, ' '
        ecall
        # print C
        li a7, 4
        addi a0, a1, 23 # E start from base address + 23
        ecall
        #print space
        li a7, 11
        li a0, ' '
        ecall
        # print D
        li a7, 4
        addi a0, a1, 0 # D start from base address
        ecall
        li a7, 11
        li a0, '\n' # change to next line after print D
        ecall

        li t3, ' '
        sb t3, 22(a1) # change back to ' '
        sb t3, 42(a1) # change back to ' '
        li t3, '\n'
        sb t3, 58(a1)
        addi t0, t0, 1
        addi a1, a1, 60 # update to base address of next line
        j loop_for_changeOrder
```

Figure 5: Code for change from DCE to ECD

1.3.5 Enter the color number from keyboard and update this

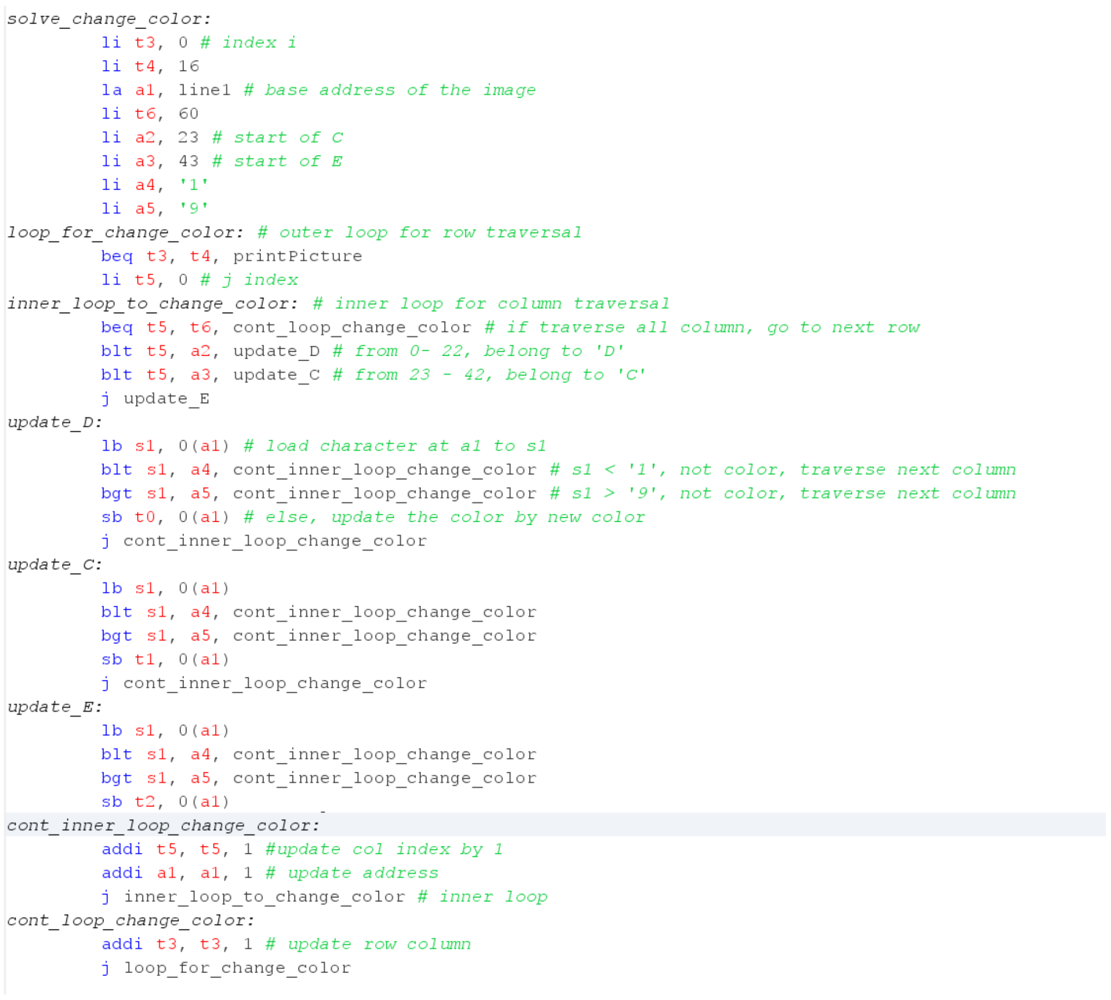
As performing 3 first tasks, we know that The `'D'` block spans columns 0 to 21. The `'C'` block spans columns 23 to 41. The `'E'` block spans columns 43 to the end. Then to solve it, we only need to perform a nested loop. The outer loop traverses row and inner loop traverses its columns. Then based on the current column position (specifying which words) and the character (is number or not) to performing replaced by the new number be entered by user for each word. After changed, it will jump to printPicture branch to print a new picture after updated. The updated one will completely replaced the original one by using this method.
Code for input new color:



```
changeColor:
        li a5, 9 # upper bound for input
input_d:
        li a7, 4
        la a0, input_d_color
        ecall
        li a7, 5 # syscall to input a number
        ecall
        bltz a0, input_d # input < 0, back to input again
        bgt a0, a5, input_d # input > 9, back to input again
        addi t0, a0, '0' # get the char of value # char = num + '0' new color for d
input_c:
        li a7, 4
        la a0, input_c_color
        ecall
        li a7, 5
        ecall
        bltz a0, input_c
        bgt a0, a5, input_c
        addi t1, a0, '0' # get the char of value, new color for c
input_e:
        li a7, 4
        la a0, input_e_color
        ecall
        li a7, 5
        ecall
        bltz a0, input_e
        bgt a0, a5, input_e
        addi t2, a0, '0' # get the char of value, new color for e
```

Figure 6: Code for input new color

This code allows user to input new color for each word. If user input out of range from 0 to 9, then it will back to the branch that you perform invalid input. Then, valid input will be converted to character by performing adding with '0'.
Code for performing change color



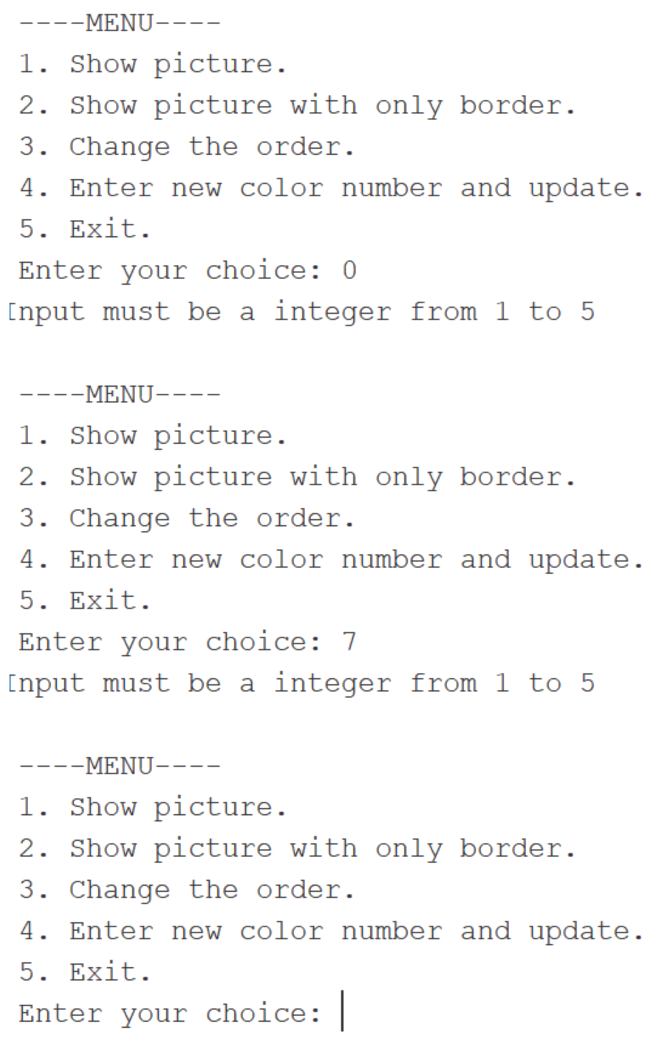
```
solve_change_color:
        li t3, 0 # index i
        li t4, 16
        la a1, line1 # base address of the image
        li t6, 60
        li a2, 23 # start of C
        li a3, 43 # start of E
        li a4, '1'
        li a5, '9'
loop_for_change_color: # outer loop for row traversal
        beq t3, t4, printPicture
        li t5, 0 # j index
inner_loop_to_change_color: # inner loop for column traversal
        beq t5, t6, cont_loop_change_color # if traverse all column, go to next row
        blt t5, a2, update_D # from 0- 22, belong to 'D'
        blt t5, a3, update_C # from 23 - 42, belong to 'C'
        j update_E
update_D:
        lb s1, 0(a1) # load character at a1 to s1
        blt s1, a4, cont_inner_loop_change_color # s1 < '1', not color, traverse next column
        bgt s1, a5, cont_inner_loop_change_color # s1 > '9', not color, traverse next column
        sb t0, 0(a1) # else, update the color by new color
        j cont_inner_loop_change_color
update_C:
        lb s1, 0(a1)
        blt s1, a4, cont_inner_loop_change_color
        bgt s1, a5, cont_inner_loop_change_color
        sb t1, 0(a1)
        j cont_inner_loop_change_color
update_E:
        lb s1, 0(a1)
        blt s1, a4, cont_inner_loop_change_color
        bgt s1, a5, cont_inner_loop_change_color
        sb t2, 0(a1)
cont_inner_loop_change_color:
        addi t5, t5, 1 #update col index by 1
        addi a1, a1, 1 # update address
        j inner_loop_to_change_color # inner loop
cont_loop_change_color:
        addi t3, t3, 1 # update row column
        j loop_for_change_color
```

Figure 7: Solve update color

This code will perform nested loop for each row. In each row, it checks each column. If column index that less than a2, then it belongs to 'D', it will jump to `update_D`. else if less than a3, then belongs to 'C', else, jump to update E.
In update function, it will check the character at this address. If it is not a number, then jump directly to `cont_inner_loop_change_color`, which will update column index and current address of picture, else, it will be replaced by the new character, which is stored at t0 ('D'), t1('C') and t2 ('E').

1.4 Demo

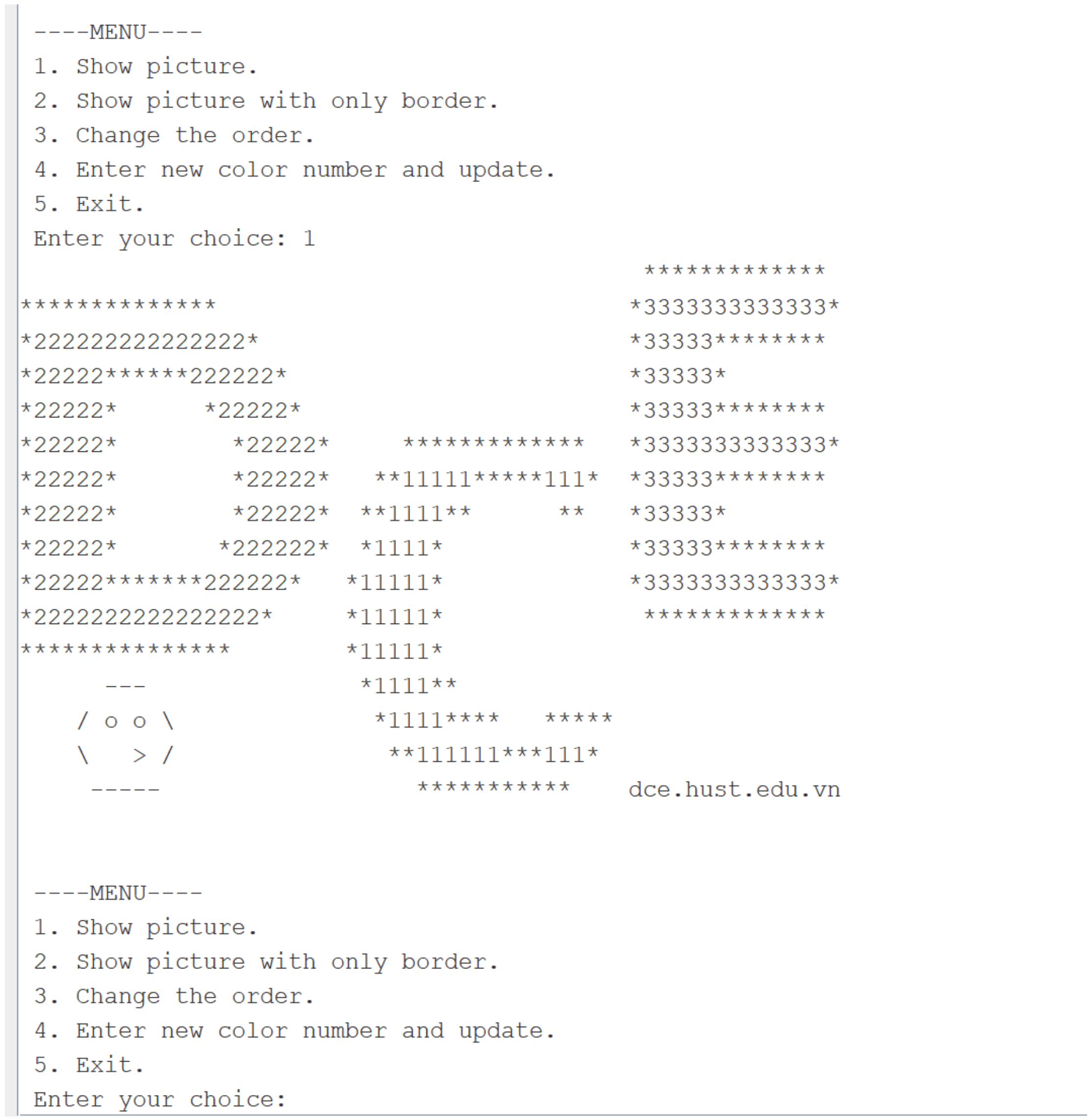
1.4.1 For menu



```
 ----MENU----
 1. Show picture.
 2. Show picture with only border.
 3. Change the order.
 4. Enter new color number and update.
 5. Exit.
 Enter your choice: 0
Input must be a integer from 1 to 5

 ----MENU----
 1. Show picture.
 2. Show picture with only border.
 3. Change the order.
 4. Enter new color number and update.
 5. Exit.
 Enter your choice: 7
Input must be a integer from 1 to 5

 ----MENU----
 1. Show picture.
 2. Show picture with only border.
 3. Change the order.
 4. Enter new color number and update.
 5. Exit.
 Enter your choice: 
```

Figure 8: Menu Demo

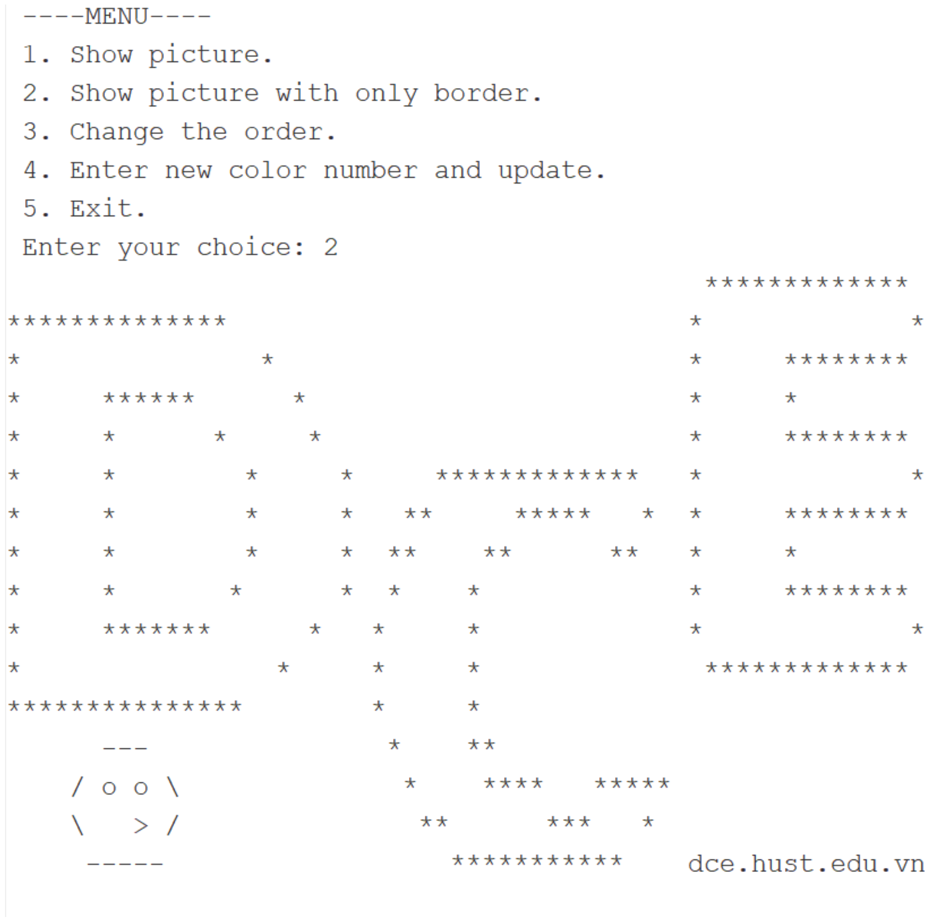
1.4.2 For display original picture



```
 ----MENU----
 1. Show picture.
 2. Show picture with only border.
 3. Change the order.
 4. Enter new color number and update.
 5. Exit.
 Enter your choice: 1
                                            *************
**************                             *3333333333333*
*222222222222222*                          *33333********
*22222******222222*                        *33333*
*22222*      *22222*                       *33333********
*22222*        *22222*     *************   *3333333333333*
*22222*        *22222*   **11111*****111*  *33333********
*22222*        *22222*  **1111**      **   *33333*
*22222*       *222222*  *1111*             *33333********
*22222*******222222*   *11111*             *3333333333333*
*2222222222222222*     *11111*              *************
***************        *11111*
      ---               *1111**
    / o o \              *1111****   *****
    \   > /               **111111***111*
     -----                  ***********    dce.hust.edu.vn


 ----MENU----
 1. Show picture.
 2. Show picture with only border.
 3. Change the order.
 4. Enter new color number and update.
 5. Exit.
 Enter your choice:
```

Figure 9: Display Original Demo

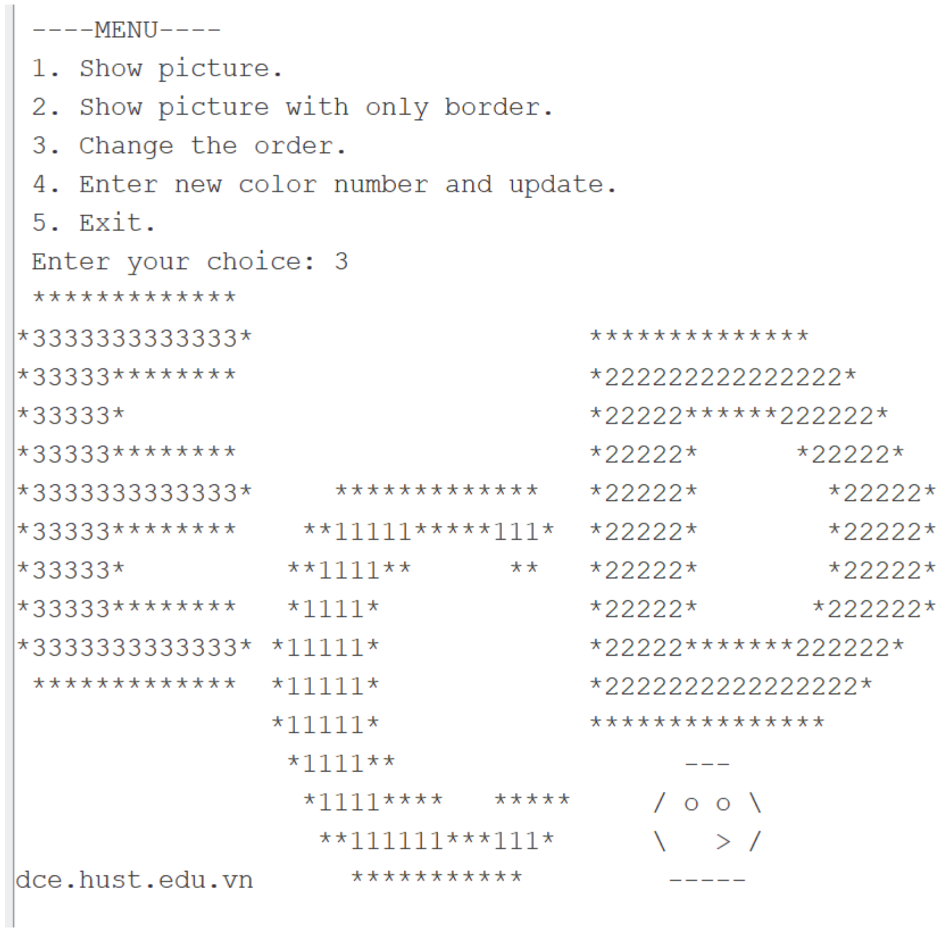
1.4.3 For display border picture



```
 ----MENU----
 1. Show picture.
 2. Show picture with only border.
 3. Change the order.
 4. Enter new color number and update.
 5. Exit.
 Enter your choice: 2
                                            *************
**************                             *             *
*               *                          *     ********
*     ******      *                        *     *
*     *      *     *                       *     ********
*     *        *     *     *************   *             *
*     *        *     *   **     *****   *  *     ********
*     *        *     *  **    **      **   *     *
*     *       *      *  *    *             *     ********
*     *******      *   *     *             *             *
*                *     *     *              *************
***************        *     *
      ---               *    **
    / o o \              *    ****   *****
    \   > /               **      ***   *
     -----                  ***********    dce.hust.edu.vn
```

Figure 10: Display Border Picture Demo

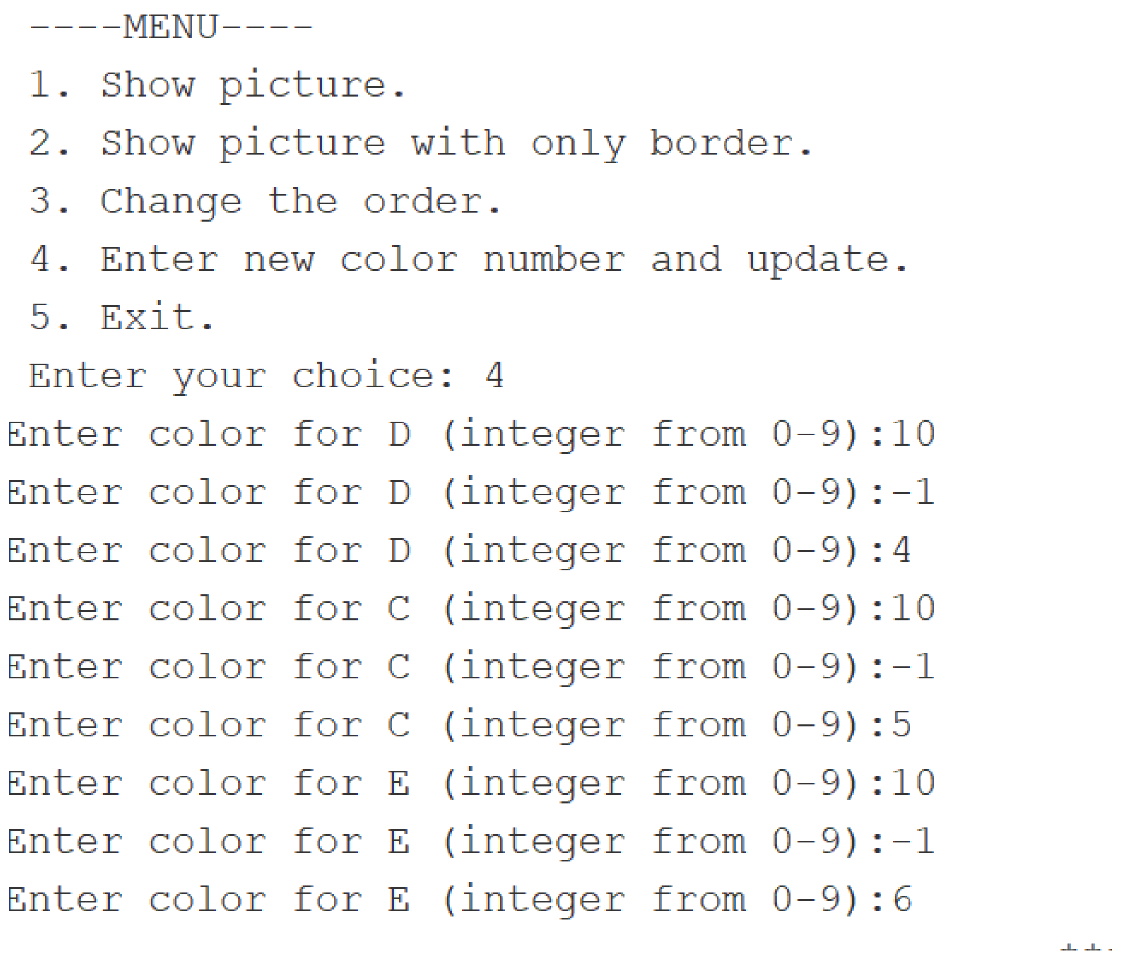
1.4.4 For ECD picture



```
 ----MENU----
 1. Show picture.
 2. Show picture with only border.
 3. Change the order.
 4. Enter new color number and update.
 5. Exit.
 Enter your choice: 3
 *************
*3333333333333*                     **************
*33333********                      *222222222222222*
*33333*                             *22222******222222*
*33333********                      *22222*      *22222*
*3333333333333*     *************   *22222*        *22222*
*33333********    **11111*****111*  *22222*        *22222*
*33333*          **1111**      **   *22222*        *22222*
*33333********   *1111*             *22222*       *222222*
*3333333333333* *11111*             *22222*******222222*
 *************  *11111*             *2222222222222222*
                *11111*             ***************
                 *1111**                  ---
                  *1111****   *****     / o o \
                   **111111***111*      \   > /
dce.hust.edu.vn       ***********        -----
```

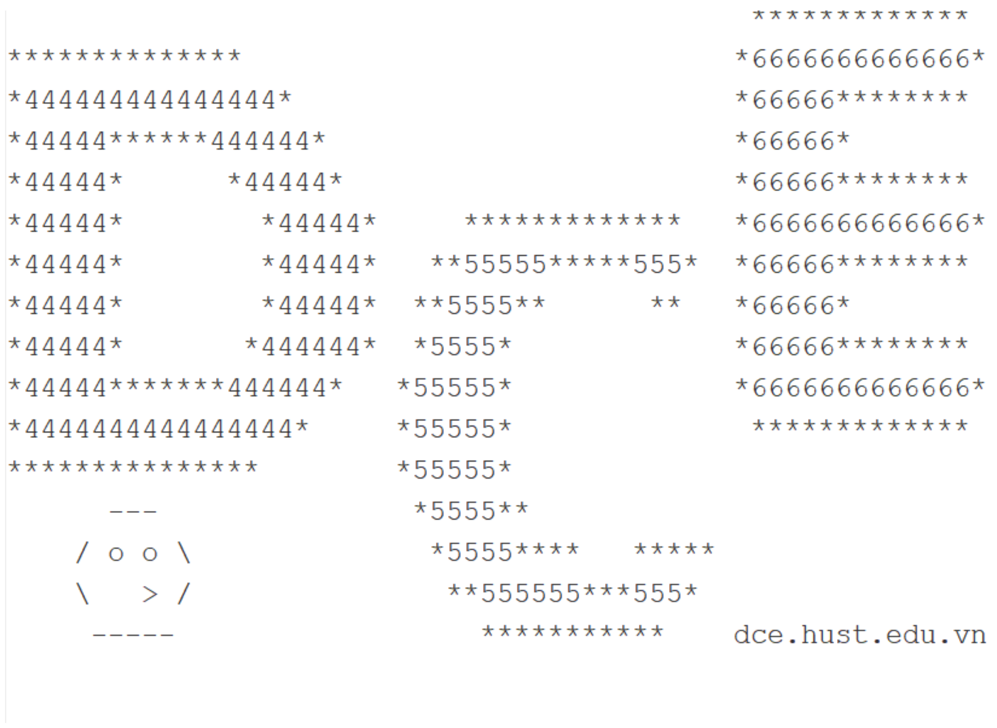
Figure 11: Display Border Picture Demo

1.4.5 For Update New Color



```
 ----MENU----
 1. Show picture.
 2. Show picture with only border.
 3. Change the order.
 4. Enter new color number and update.
 5. Exit.
 Enter your choice: 4
Enter color for D (integer from 0-9):10
Enter color for D (integer from 0-9):-1
Enter color for D (integer from 0-9):4
Enter color for C (integer from 0-9):10
Enter color for C (integer from 0-9):-1
Enter color for C (integer from 0-9):5
Enter color for E (integer from 0-9):10
Enter color for E (integer from 0-9):-1
Enter color for E (integer from 0-9):6
```

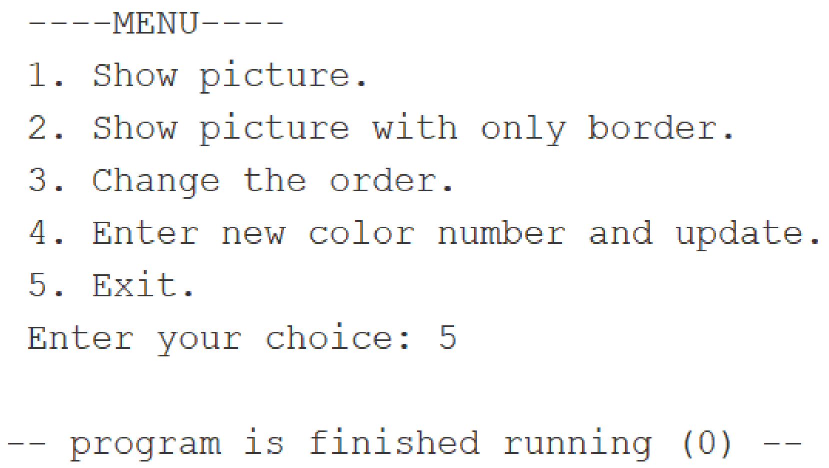
Figure 12: Update Input Demo



```
                                            *************
**************                             *6666666666666*
*444444444444444*                          *66666********
*44444******444444*                        *66666*
*44444*      *44444*                       *66666********
*44444*        *44444*     *************   *6666666666666*
*44444*        *44444*   **55555*****555*  *66666********
*44444*        *44444*  **5555**      **   *66666*
*44444*       *444444*  *5555*             *66666********
*44444*******444444*   *55555*             *6666666666666*
*4444444444444444*     *55555*              *************
***************        *55555*
      ---               *5555**
    / o o \              *5555****   *****
    \   > /               **555555***555*
     -----                  ***********    dce.hust.edu.vn
```

Figure 13: Update Result Demo

1.4.6 Exit

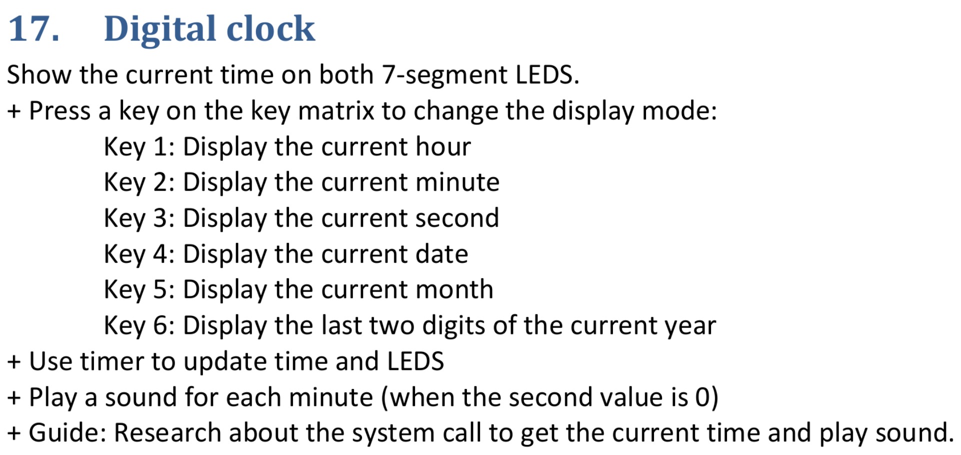


```
 ----MENU----
 1. Show picture.
 2. Show picture with only border.
 3. Change the order.
 4. Enter new color number and update.
 5. Exit.
 Enter your choice: 5

-- program is finished running (0) --
```

Figure 14: Exit

2 Digital Clock

2.1 Description



17. Digital clock

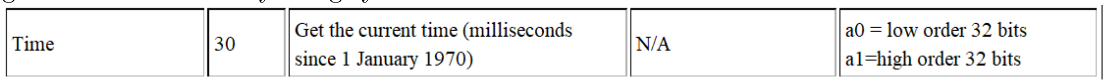
Show the current time on both 7-segment LEDS.

+ Press a key on the key matrix to change the display mode:
  - Key 1: Display the current hour
  - Key 2: Display the current minute
  - Key 3: Display the current second
  - Key 4: Display the current date
  - Key 5: Display the current month
  - Key 6: Display the last two digits of the current year
+ Use timer to update time and LEDS
+ Play a sound for each minute (when the second value is 0)
+ Guide: Research about the system call to get the current time and play sound.

2.2 Algorithms

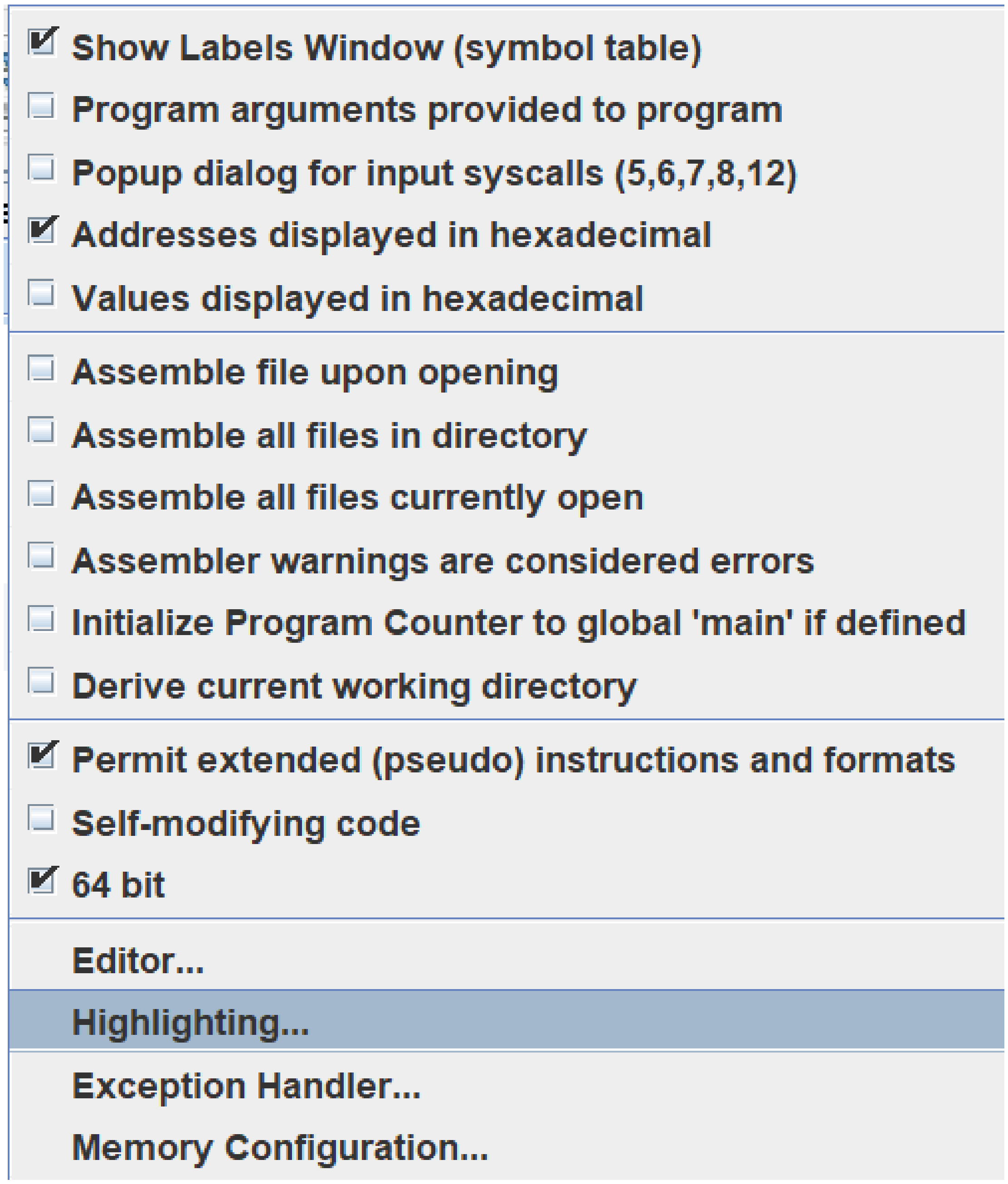
2.2.1 Display the current time using LEDs

In the first task, we need to display the current time with the LEDs. To start with, we need to get the current time by using syscall 30.



Time	30	Get the current time (milliseconds since 1 January 1970)	N/A	a0 = low order 32 bits
				a1=high order 32 bits

We have 32 first bits are stored in a0 and 32 higher bits are stored in a1. So we need 64 bits to solve this problem. So to make sure that the code can run correctly, please turn on the 64 bits in setting.



☑ Show Labels Window (symbol table)
☐ Program arguments provided to program
☐ Popup dialog for input syscalls (5,6,7,8,12)
☑ Addresses displayed in hexadecimal
☐ Values displayed in hexadecimal
☐ Assemble file upon opening
☐ Assemble all files in directory
☐ Assemble all files currently open
☐ Assembler warnings are considered errors
☐ Initialize Program Counter to global 'main' if defined
☐ Derive current working directory
☑ Permit extended (pseudo) instructions and formats
☐ Self-modifying code
☑ 64 bit

Editor...
Highlighting...
Exception Handler...
Memory Configuration...

After having a0 and a1, i need a register can have the form a1, a0 that have 64 bits. But when we turn on 64 bits, in a1 32 first bits are the data we need but the 32 higher bits are randomized. So we need to clear that!. We clear the 32 higher bits in a0 also. Then we can combine by using operation "OR".

After getting the time in register t1 for example. t1 will store the miliseconds from 1/1/1970 till now. So our task starts, we need to calculate the current year first. And the easiest way to do that is using a loop from 1970 till now, make sure handling the leap year and normal year, and minus the miliseconds in one year after each loop. After getting the current year, we will get the milisecond in this year!. So the next step is calculating the month. In calculating the month we need to make sure some exceptions like 31 days, 30 days, 29 days, 28 days. After that is calculating the current date, hour and seconds. It is much easier a work! To get the current date, we just need to divide by the miliseconds in one day and we get the current date, the same with hour and second.

One more thing need to be done in this section is GMT + 7. We all know that Vietname has GMT + 7 but the time in 1/1/1970 starts at GMT + 0, so we need to add 7 hours into register t1 at first

Our task nearly completed now. Next step is to display it in the LEDs, we just write a function named ”displayLED” with the above requirement.

2.2.2 Update the time by using Timer Tool

One more step in this is getting the time of the Timer Tool for updating the time. We just need to get the time and add it into a register t1 before a new loop

2.2.3 Play a sound when the seconds reach 0

For playing a sound, we can use system call 31 to get the sound

2.3 Code and explanation

2.3.1 Get the current time

```
# Get the current time in milliseconds
  li a7, 30               # System call number for getting the time (milliseconds)
  ecall                   # Make the system call
  mv t0, a0               # Lower 32 bits of the time
  mv t1, a1               # Higher 32 bits of the time


  # Combine lower and higher parts into a single 64-bit value
  slli t1, t1, 32         # Shift the higher 32 bits left by 32 positions
  slli t0, t0, 32
  srli t0, t0, 32
  or t1, t1, t0           # Combine with the lower 32 bits (bitwise OR)

  li a7, 36
  mv a0, t0
  ecall
```

To clear the higher 32 bits in a0, a1 we need to use ”slli t1, t1, 32”, that will make the lower 32 bits to become higher 32 bits, for a0 we need to become the lower 32 bits again so ”srli a0, a0, 32” is needed then we combine 2 register by using ”or t1, t1, a0” this will be the miliseconds from 1/1/1970 till now.

```
# t1 will store the value of the miliseconds from 1970 till this current year
li s11, 31536000000    # number of miliseconds of 365 days
li s10, 86400000       # number of miliseconds in 1 day
li s9, 3600000         # solve for GMT + 7
li s8, 7
li t2, 1970            # this have to store the current year
li t3, 0               # this store the number of seconds in this current year
li t4, 4               # temp to get the leap year

mul s9, s8, s9
add t1, t1, s9

calculate_year_loop:
      rem s9, t2, t4
      bne s9, zero, next_year_loop # if this is not leap go to next_year_loop
      add t3, t3, s10              # else add one day

      next_year_loop:
            add t3, t3, s11
            bgt t3, t1, end_year_loop
            addi t2, t2, 1
            j calculate_year_loop
end_year_loop:

# After running the loop, t2 will get the current year
```

The next step is to calculate the current year. In the calculate year loop, we need to compare the current miliseconds with the miliseconds in a year. If the current miliseconds is less than miliseconds in a year, break the loop and we get the current year.

The next thing to calculate is current month. We need to handle exception.

```
li s9, 2678400000      # number of miliseconds in 31 days
li s8, 2592000000      # number of miliseconds in 30 days
li s7, 2505600000      # number of miliseconds in 29 days
li s6, 2419200000      # number of miliseconds in 28 days
rem a2, t2, t4         # a2 = 0 means leap year, normal year otherwise
li a4, 1               # let's chosse t4 as the current month

calculate_month_loop:
      # month 1   31 days
      ble t1, s9, end_month_loop
      sub t1, t1, s9
      addi a4, a4, 1

      # month 2 28 - 29 days
      beq a4, zero, next_calculate_month
      ble t1, s7, end_month_loop
      sub t1, t1, s7
      addi a4, a4, 1
      j next_month_1

      next_calculate_month:
            ble t1, s6, end_month_loop
            sub t1, t1, s6
            addi a4, a4, 1
      next_month_1:

      # month 3 - 31 days
      ble t1, s9, end_month_loop
      sub t1, t1, s9
      addi a4, a4, 1

      # month 4 - 30 days
      ble t1, s8, end_month_loop
```

We need to get miliseconds in months has 31 days, 30 days, 29 days, 28 days, and do pretty the same with the calculate year loop.

The next is getting the date, hour, seconds.

```
li s10, 3600000   # s10 now store the number of seconds in 1 hour
div a5, t1, s10   # a5 stores the current hour

mul s11, a5, s10
sub t1, t1, s11   # update t1

li s10, 60000
div a6, t1, s10   # a6 stores the current minute

mul s11, a6, s10
sub t1, t1, s11

li s10, 1000
div a1, t1, s10
```

2.3.2 Display in the LED

First, we need to handle the key input from 1 to 6

```
li s7, 0x00000021
bne a0, s7, cont1

#########################################################
####### this will solve the key 1 (current hour)#########
#########################################################

li s0, 10
rem s1, a5, s0   # the first number of hour
div s2, a5, s0   # the second number of hour

# after this operations, the form we can get is s2 : s1   (s2 left LED and s1 RIGHT LED)

jal display_LED

j loop

cont1:

#########################################################
####### this will solve the key 2 (current minute)#######
#########################################################

li s7, 0x00000041
bne a0, s7, cont2

li s0, 10
rem s1, a6, s0
div s2, a6, s0

jal display_LED
j loop

cont2:
```

We just check each button and go to the display LED when each button from 1 to 6 is clicked.

In the display LED, we have 2 part: the left digit and right digit.

2.3.3 Updating time using Timer Tool

```
loop:      # this loop will update the time using timer tool

       li t1, 0
       li t2, 0

       li t0, TIMER_NOW
       lw t3, 0(t0)


       add s9, s8, t3


j new_loop
```

We just need to get the time in the timer tool and add to the s9 (current miliseconds) before going to the new loop. TIMER NOW is 0xFFFF0018 (Memory-mapped register for the current timer value)

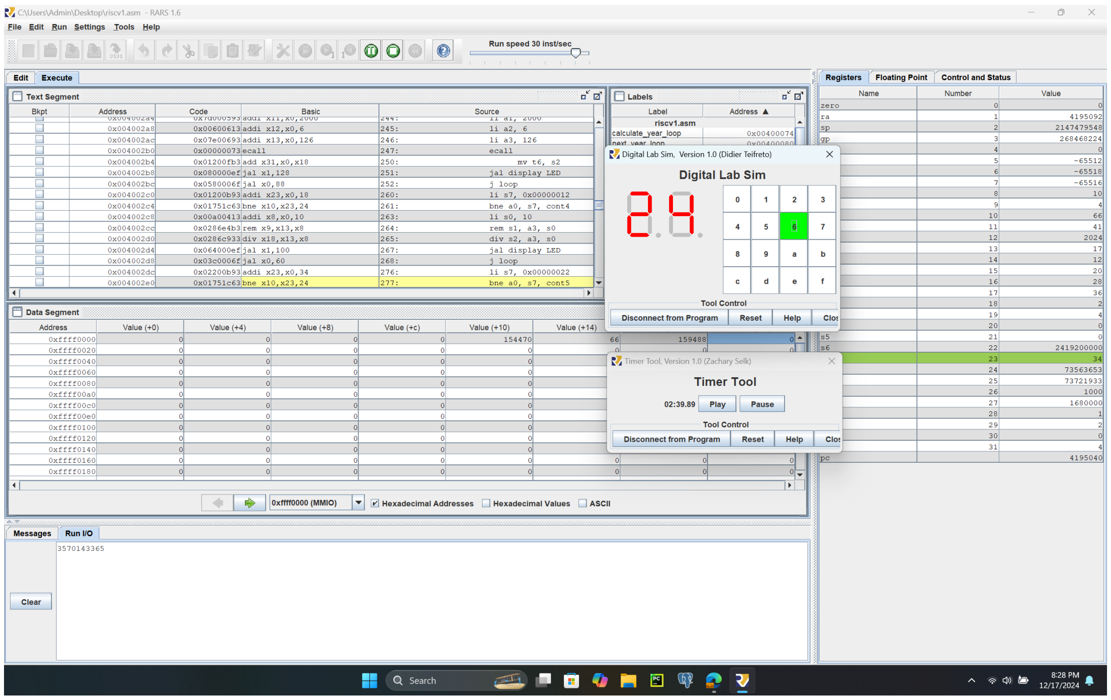
2.3.4 Play a sound

```
    loop:        # this loop will update the time using timer tool

         li t1, 0
         li t2, 0

         li t0, TIMER_NOW
         lw t3, 0(t0)


         add s9, s8, t3


    j new_loop
```

To play a sound, we need to use syscall 31, we just compare the seconds, if the new seconds is smaller the the previous seconds, we display a sound!

2.4 Demo

2.4.1 For year displaying



C:\Users\Admin\Desktop\riscv1.asm - RARS 1.6

Run speed 30 inst/sec

Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x004002a4	0x7d000593	addi x11,x0,2000	244: li a1, 2000
☐	0x004002a8	0x00600613	addi x12,x0,6	245: li a2, 6
☐	0x004002ac	0x07e00693	addi x13,x0,126	246: li a3, 126
☐	0x004002b0	0x00000073	ecall	247: ecall
☐	0x004002b4	0x01200fb3	add x31,x0,x18	250: mv t6, s2
☐	0x004002b8	0x080000ef	jal x1,128	251: jal display LED
☐	0x004002bc	0x0580006f	jal x0,88	252: j loop
☐	0x004002c0	0x01200b93	addi x23,x0,18	260: li s7, 0x00000012
☐	0x004002c4	0x01751c63	bne x10,x23,24	261: bne a0, s7, cont4
☐	0x004002c8	0x00a00413	addi x8,x0,10	263: li s0, 10
☐	0x004002cc	0x0286e4b3	rem x9,x13,x8	264: rem s1, a3, s0
☐	0x004002d0	0x0286c933	div x18,x13,x8	265: div s2, a3, s0
☐	0x004002d4	0x064000ef	jal x1,100	267: jal display LED
☐	0x004002d8	0x03c0006f	jal x0,60	268: j loop
☐	0x004002dc	0x02200b93	addi x23,x0,34	276: li s7, 0x00000022
☐	0x004002e0	0x01751c63	bne x10,x23,24	277: bne a0, s7, cont5

Labels

Label	Address ▲
riscv1.asm	
calculate_year_loop	0x00400074
next_year_loop	0x00400080

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0xffff0000	0	0	0	0	154470	66
0xffff0020	0	0	0	0	0	0
0xffff0040	0	0	0	0	0	
0xffff0060	0	0	0	0	0	
0xffff0080	0	0	0	0	0	
0xffff00a0	0	0	0	0	0	
0xffff00c0	0	0	0	0	0	
0xffff00e0	0	0	0	0	0	
0xffff0100	0	0	0	0	0	
0xffff0120	0	0	0	0	0	
0xffff0140	0	0	0	0	0	
0xffff0160	0	0	0	0	0	0
0xffff0180	0	0	0	0	0	0

0xffff0000 (MMIO) ☑ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Registers

Name	Number	Value
zero	0	0
ra	1	4195092
sp	2	2147479548
gp	3	268468224
	4	0
	5	-65512
	6	-65518
	7	-65516
	8	10
	9	4
	10	66
	11	41
	12	2024
	13	17
	14	12
	15	20
	16	28
	17	36
	18	2
	19	4
	20	0
s5	21	0
s6	22	2419200000
	23	34
	24	73563653
	25	73721933
	26	1000
	27	1680000
	28	1
	29	2
	30	0
	31	4
pc		4195040

Digital Lab Sim, Version 1.0 (Didier Teifreto)

Timer Tool, Version 1.0 (Zachary Selk)

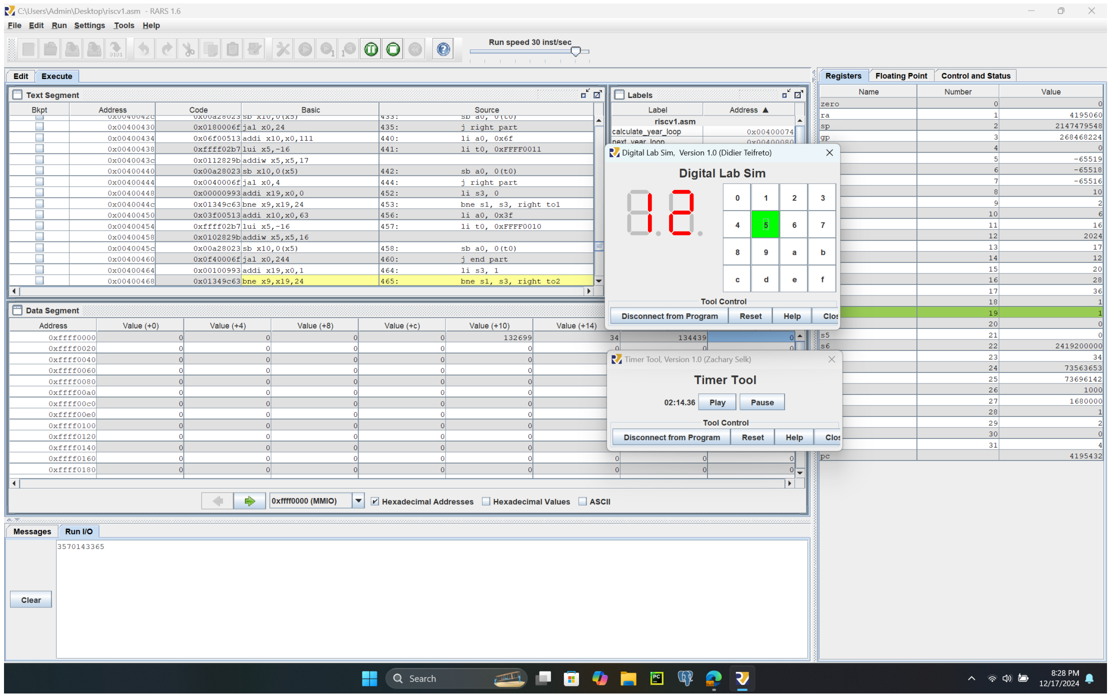
Timer Tool 02:39.89

Run I/O

```
3570143365
```

Figure 15: Displaying 2 last number of current year

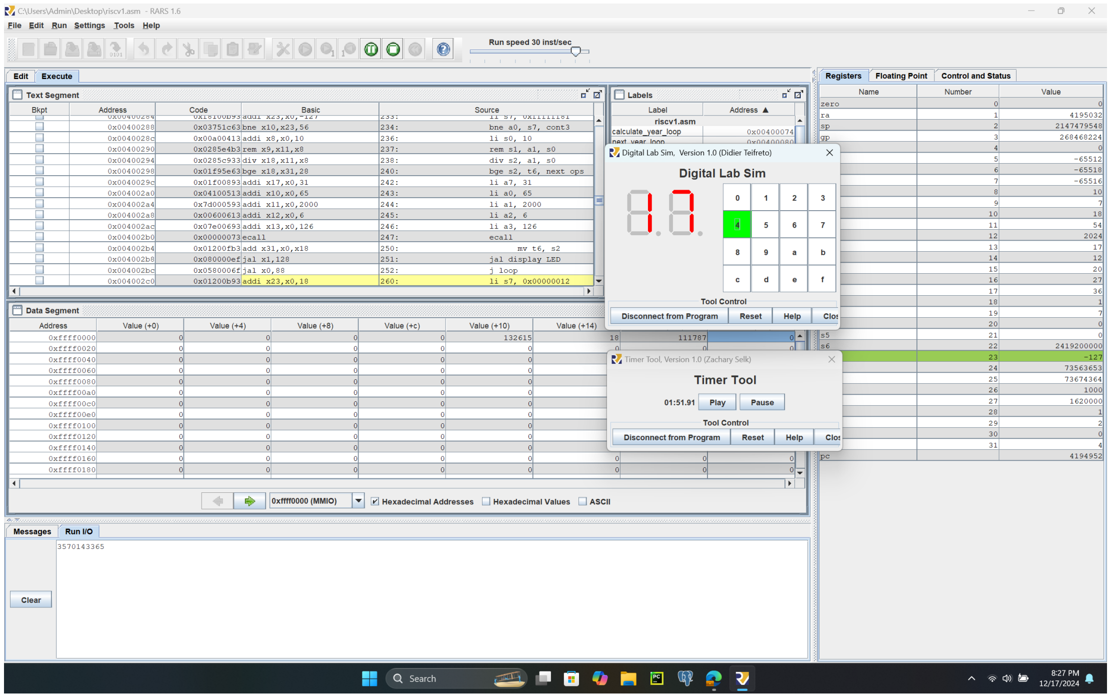
2.4.2 For month displaying



```
C:\Users\Admin\Desktop\riscv1.asm - RARS 1.6

Text Segment
Bkpt  Address     Code        Basic               Source
      0x00400430  0x0180006f  jal x0,24           435:   j right part
      0x00400434  0x06f00513  addi x10,x0,111     440:   li a0, 0x6f
      0x00400438  0xffff02b7  lui x5,-16          441:   li t0, 0xFFFF0011
      0x0040043c  0x0112829b  addiw x5,x5,17
      0x00400440  0x00a28023  sb x10,0(x5)        442:   sb a0, 0(t0)
      0x00400444  0x0040006f  jal x0,4            444:   j right part
      0x00400448  0x00000993  addi x19,x0,0       452:   li s3, 0
      0x0040044c  0x01349c63  bne x9,x19,24       453:   bne s1, s3, right to1
      0x00400450  0x03f00513  addi x10,x0,63      456:   li a0, 0x3f
      0x00400454  0xffff02b7  lui x5,-16          457:   li t0, 0xFFFF0010
      0x00400458  0x0102829b  addiw x5,x5,16
      0x0040045c  0x00a28023  sb x10,0(x5)        458:   sb a0, 0(t0)
      0x00400460  0x0f40006f  jal x0,244          460:   j end part
      0x00400464  0x00100993  addi x19,x0,1       464:   li s3, 1
      0x00400468  0x01349c63  bne x9,x19,24       465:   bne s1, s3, right to2

Data Segment
Address     Value (+0)  Value (+4)  Value (+8)  Value (+c)  Value (+10)  Value (+14)
0xffff0000           0           0           0           0       132699           34   134439
0xffff0020           0           0           0           0            0            0
...
0xffff0180           0           0           0           0            0            0

Timer Tool  02:14.36

Run I/O
3570143365

Registers
Name   Number  Value
zero        0           0
ra          1     4195060
sp          2  2147479548
gp          3   268468224
            4           0
            5      -65519
            6      -65518
            7      -65516
            8          10
            9           2
           10           6
           11          16
           12        2024
           13          17
           14          12
           15          20
           16          28
           17          36
           18           1
           19           1
           20           0
s5         21           0
s6         22  2419200000
           23          34
           24    73563653
           25    73696142
           26        1000
           27     1680000
           28           1
           29           2
           30           0
           31           4
pc                4195432
```

Figure 16: Displaying current month

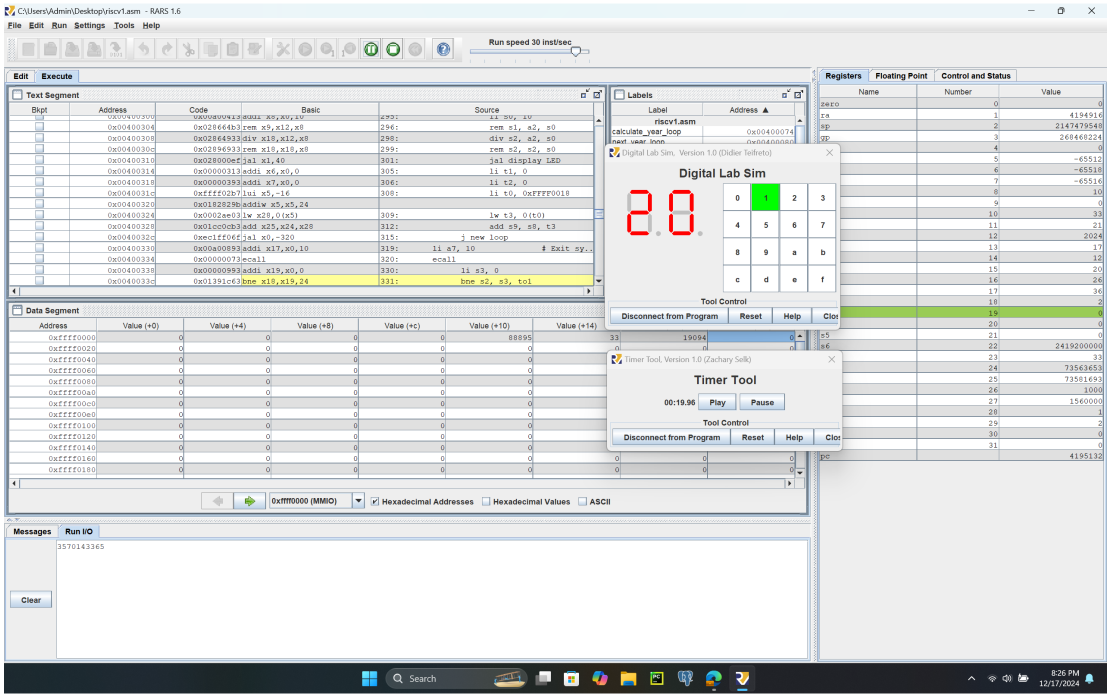
2.4.3 For date displaying



```
C:\Users\Admin\Desktop\riscv1.asm - RARS 1.6

Text Segment
Bkpt  Address     Code        Basic               Source
      0x00400288  0x03751c63  bne x10,x23,56      234:   bne a0, s7, cont3
      0x0040028c  0x00a00413  addi x8,x0,10       236:   li s0, 10
      0x00400290  0x0285e4b3  rem x9,x11,x8       237:   rem s1, a1, s0
      0x00400294  0x0285c933  div x18,x11,x8      238:   div s2, a1, s0
      0x00400298  0x01f95e63  bge x18,x31,28      240:   bge s2, t6, next ops
      0x0040029c  0x01f00893  addi x17,x0,31      242:   li a7, 31
      0x004002a0  0x04100513  addi x10,x0,65      243:   li a0, 65
      0x004002a4  0x7d000593  addi x11,x0,2000    244:   li a1, 2000
      0x004002a8  0x00600613  addi x12,x0,6       245:   li a2, 6
      0x004002ac  0x07e00693  addi x13,x0,126     246:   li a3, 126
      0x004002b0  0x00000073  ecall               247:   ecall
      0x004002b4  0x01200fb3  add x31,x0,x18      250:      mv t6, s2
      0x004002b8  0x080000ef  jal x1,128          251:   jal display LED
      0x004002bc  0x0580006f  jal x0,88           252:   j loop
      0x004002c0  0x01200b93  addi x23,x0,18      260:   li s7, 0x00000012

Data Segment
Address     Value (+0)  Value (+4)  Value (+8)  Value (+c)  Value (+10)  Value (+14)
0xffff0000           0           0           0           0       132615           18   111787
0xffff0020           0           0           0           0            0            0
...
0xffff0180           0           0           0           0            0            0

Timer Tool  01:51.91

Run I/O
3570143365

Registers
Name   Number  Value
zero        0           0
ra          1     4195032
sp          2  2147479548
gp          3   268468224
            4           0
            5      -65512
            6      -65518
            7      -65516
            8          10
            9           7
           10          18
           11          54
           12        2024
           13          17
           14          12
           15          20
           16          27
           17          36
           18           1
           19           7
           20           0
s5         21           0
s6         22  2419200000
           23        -127
           24    73563653
           25    73674364
           26        1000
           27     1620000
           28           1
           29           2
           30           0
           31           4
pc                4194952
```

Figure 17: Displaying current date

2.4.4 For hour displaying



C:\Users\Admin\Desktop\riscv1.asm - RARS 1.6

Run speed 30 inst/sec

Text Segment

```
Bkpt  Address     Code        Basic              Source
      0x00400304  0x028664b3  rem x9,x12,x8      296:        rem s1, a2, s0
      0x00400308  0x02864933  div x18,x12,x8     298:        div s2, a2, s0
      0x0040030c  0x02896933  rem x18,x18,x8     299:        rem s2, s2, s0
      0x00400310  0x028000ef  jal x1,40          301:        jal display LED
      0x00400314  0x00000313  addi x6,x0,0       305:        li t1, 0
      0x00400318  0x00000393  addi x7,x0,0       306:        li t2, 0
      0x0040031c  0xffff02b7  lui x5,-16         308:        li t0, 0xFFFF0018
      0x00400320  0x0182829b  addiw x5,x5,24
      0x00400324  0x0002ae03  lw x28,0(x5)       309:        lw t3, 0(t0)
      0x00400328  0x01cc0cb3  add x25,x24,x28    312:        add s9, s8, t3
      0x0040032c  0xec1ff06f  jal x0,-320        315:      j new loop
      0x00400330  0x00a00893  addi x17,x0,10     319:   li a7, 10          # Exit sy..
      0x00400334  0x00000073  ecall              320:   ecall
      0x00400338  0x00000993  addi x19,x0,0      330:      li s3, 0
      0x0040033c  0x01391c63  bne x18,x19,24     331:      bne s2, s3, to1
```

Data Segment

```
Address     Value (+0)  Value (+4)  Value (+8)  Value (+c)  Value (+10)  Value (+14)
0xffff0000           0           0           0           0        88895           33
0xffff0020           0           0           0           0            0            0
0xffff0040           0           0           0           0            0            0
0xffff0060           0           0           0           0            0            0
0xffff0080           0           0           0           0            0            0
0xffff00a0           0           0           0           0            0            0
0xffff00c0           0           0           0           0            0            0
0xffff00e0           0           0           0           0            0            0
0xffff0100           0           0           0           0            0            0
0xffff0120           0           0           0           0            0            0
0xffff0140           0           0           0           0            0            0
0xffff0160           0           0           0           0            0            0
0xffff0180           0           0           0           0            0            0
```

0xffff0000 (MMIO) ✓ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Labels: riscv1.asm — calculate_year_loop 0x00400074

Digital Lab Sim, Version 1.0 (Didier Teifreto)

Timer Tool, Version 1.0 (Zachary Selk) — 00:19.96

Run I/O

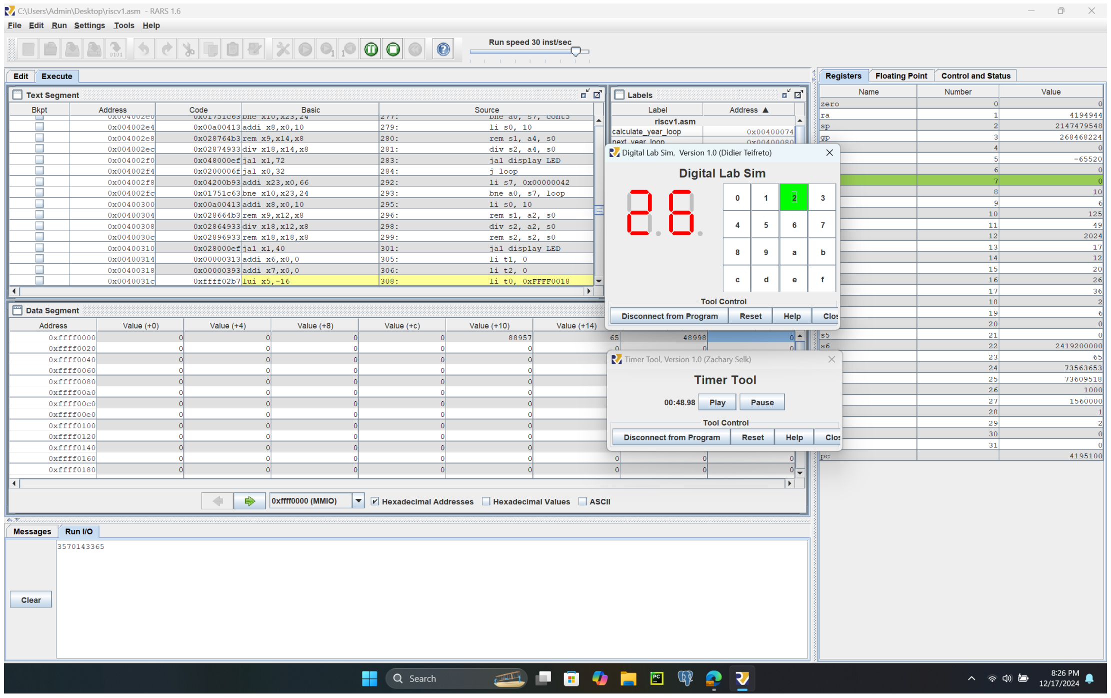
```
3570143365
```

Name	Number	Value
zero	0	0
ra	1	4194916
sp	2	2147479548
gp	3	268468224
	4	0
	5	-65512
	6	-65518
	7	-65516
	8	10
	9	0
	10	33
	11	21
	12	2024
	13	17
	14	12
	15	20
	16	26
	17	36
	18	2
	19	0
	20	0
s5	21	0
s6	22	2419200000
	23	33
	24	73563653
	25	73581693
	26	1000
	27	1560000
	28	1
	29	2
	30	0
	31	0
pc		4195132

File Edit Run Settings Tools Help
Edit Execute
Digital Lab Sim
0 1 2 3 4 5 6 7 8 9 a b c d e f
Tool Control
Disconnect from Program Reset Help Clos
Timer Tool
Play Pause
Registers Floating Point Control and Status
Messages Run I/O
Clear
Search
8:26 PM 12/17/2024

Figure 18: Displaying current hour

2.4.5 For minute displaying



C:\Users\Admin\Desktop\riscv1.asm - RARS 1.6

Run speed 30 inst/sec

Text Segment

```
Bkpt  Address     Code        Basic              Source
      0x004002e4  0x00a00413  addi x8,x0,10      279:        li s0, 10
      0x004002e8  0x028764b3  rem x9,x14,x8      280:        rem s1, a4, s0
      0x004002ec  0x02874933  div x18,x14,x8     281:        div s2, a4, s0
      0x004002f0  0x048000ef  jal x1,72          283:        jal display LED
      0x004002f4  0x0200006f  jal x0,32          284:        j loop
      0x004002f8  0x04200b93  addi x23,x0,66     292:        li s7, 0x00000042
      0x004002fc  0x01751c63  bne x10,x23,24     293:        bne a0, s7, loop
      0x00400300  0x00a00413  addi x8,x0,10      295:        li s0, 10
      0x00400304  0x028664b3  rem x9,x12,x8      296:        rem s1, a2, s0
      0x00400308  0x02864933  div x18,x12,x8     298:        div s2, a2, s0
      0x0040030c  0x02896933  rem x18,x18,x8     299:        rem s2, s2, s0
      0x00400310  0x028000ef  jal x1,40          301:        jal display LED
      0x00400314  0x00000313  addi x6,x0,0       305:        li t1, 0
      0x00400318  0x00000393  addi x7,x0,0       306:        li t2, 0
      0x0040031c  0xffff02b7  lui x5,-16         308:        li t0, 0xFFFF0018
```

Data Segment

```
Address     Value (+0)  Value (+4)  Value (+8)  Value (+c)  Value (+10)  Value (+14)
0xffff0000           0           0           0           0        88957           65
0xffff0020           0           0           0           0            0            0
0xffff0040           0           0           0           0            0            0
0xffff0060           0           0           0           0            0            0
0xffff0080           0           0           0           0            0            0
0xffff00a0           0           0           0           0            0            0
0xffff00c0           0           0           0           0            0            0
0xffff00e0           0           0           0           0            0            0
0xffff0100           0           0           0           0            0            0
0xffff0120           0           0           0           0            0            0
0xffff0140           0           0           0           0            0            0
0xffff0160           0           0           0           0            0            0
0xffff0180           0           0           0           0            0            0
```

0xffff0000 (MMIO) ✓ Hexadecimal Addresses ☐ Hexadecimal Values ☐ ASCII

Labels: riscv1.asm — calculate_year_loop 0x00400074

Digital Lab Sim, Version 1.0 (Didier Teifreto)

Timer Tool, Version 1.0 (Zachary Selk) — 00:48.98

Run I/O

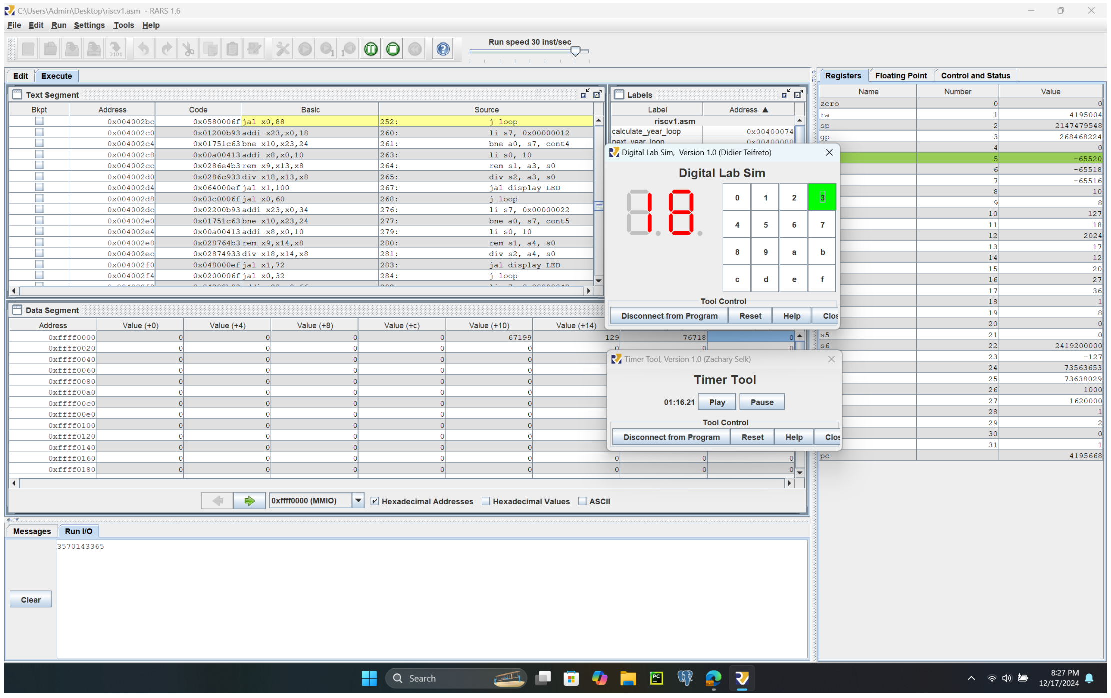
```
3570143365
```

Name	Number	Value
zero	0	0
ra	1	4194944
sp	2	2147479548
gp	3	268468224
	4	0
	5	-65520
	6	0
	7	0
	8	10
	9	6
	10	125
	11	49
	12	2024
	13	17
	14	12
	15	20
	16	26
	17	36
	18	2
	19	6
	20	0
s5	21	0
s6	22	2419200000
	23	65
	24	73563653
	25	73609518
	26	1000
	27	1560000
	28	1
	29	2
	30	0
	31	0
pc		4195100

File Edit Run Settings Tools Help
Edit Execute
Digital Lab Sim
0 1 2 3 4 5 6 7 8 9 a b c d e f
Tool Control
Disconnect from Program Reset Help Clos
Timer Tool
Play Pause
Registers Floating Point Control and Status
Messages Run I/O
Clear
Search
8:26 PM 12/17/2024

Figure 19: Displaying current minute

2.4.6 For second displaying



Text Segment

Bkpt	Address	Code	Basic	Source
☐	0x004002bc	0x0580006f	jal x0,88	252: j loop
☐	0x004002c0	0x01200b93	addi x23,x0,18	260: li s7, 0x00000012
☐	0x004002c4	0x01751c63	bne x10,x23,24	261: bne a0, s7, cont4
☐	0x004002c8	0x00a00413	addi x8,x0,10	263: li s0, 10
☐	0x004002cc	0x0286e4b3	rem x9,x13,x8	264: rem s1, a3, s0
☐	0x004002d0	0x0286c933	div x18,x13,x8	265: div s2, a3, s0
☐	0x004002d4	0x064000ef	jal x1,100	267: jal display LED
☐	0x004002d8	0x03c0006f	jal x0,60	268: j loop
☐	0x004002dc	0x02200b93	addi x23,x0,34	276: li s7, 0x00000022
☐	0x004002e0	0x01751c63	bne x10,x23,24	277: bne a0, s7, cont5
☐	0x004002e4	0x00a00413	addi x8,x0,10	279: li s0, 10
☐	0x004002e8	0x028764b3	rem x9,x14,x8	280: rem s1, a4, s0
☐	0x004002ec	0x02874933	div x18,x14,x8	281: div s2, a4, s0
☐	0x004002f0	0x048000ef	jal x1,72	283: jal display LED
☐	0x004002f4	0x0200006f	jal x0,32	284: j loop

Labels

Label	Address ▲
riscv1.asm	
calculate_year_loop	0x00400074
next_year_loop	0x00400080

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)		
0xffff0000	0	0	0	0	67199	129	76718	0
0xffff0020	0	0	0	0	0	0	0	0
0xffff0040	0	0	0	0	0			
0xffff0060	0	0	0	0	0			
0xffff0080	0	0	0	0	0			
0xffff00a0	0	0	0	0	0			
0xffff00c0	0	0	0	0	0			
0xffff00e0	0	0	0	0	0			
0xffff0100	0	0	0	0	0			
0xffff0120	0	0	0	0	0			
0xffff0140	0	0	0	0	0			
0xffff0160	0	0	0	0	0	0	0	0
0xffff0180	0	0	0	0	0	0	0	0

0xffff0000 (MMIO) [✓ Hexadecimal Addresses] ☐ Hexadecimal Values ☐ ASCII

Registers

Name	Number	Value
zero	0	0
ra	1	4195004
sp	2	2147479548
gp	3	268468224
	4	0
	5	-65520
	6	-65518
	7	-65516
	8	10
	9	8
	10	127
	11	18
	12	2024
	13	17
	14	12
	15	20
	16	27
	17	36
	18	1
	19	8
	20	0
s5	21	0
s6	22	2419200000
	23	-127
	24	73563653
	25	73638029
	26	1000
	27	1620000
	28	1
	29	2
	30	0
	31	1
pc		4195668

Run I/O

```
3570143365
```

Figure 20: Displaying current second