ASSEMBLY REPORT

MID-TERM

Full name: Hoàng Mạnh Cường

Student ID: 20226018

Date: 2024

Assignment A7:

7. Write a function to check if a number is a perfect square. Then enter a positive integer N from the keyboard, print out all the square numbers (or perfect squares) less than N.

```
.data
 prompt: .asciz "Enter a positive integer"
 perfect: .asciz "\nperfect square: "
 not_perfect: .asciz "\nnot perfect square: "
.text
 main:
     input_loop:
        li a7, 4  #print string
        la a0, prompt
        ecall

        li a7, 5 #read int
        ecall
        mv t0, a0

        blez t0, input_loop

        li t1, 1 #i initialize i= 1
     # print the perfect squares
     print_loop:
        mul t2,t1,t1
        bge t2,t0,check_square

        add a0,t2,zero
        li a7, 1 #print i*i
        ecall

        li a0, 32 # print space
        li a7, 11 #print char
```

```
        ecall

        addi t1, t1 , 1 # i = i+1
        j print_loop

    check_square:
        beq t2, t0 perfect_square # check that i*i == n?
        j not_perfect_square
    perfect_square:
        la a0, perfect
        li a7, 4 # print string
        ecall

        mv a0,t0
        li a7, 1 #print n
        ecall
        j done
    not_perfect_square:
        la a0, not_perfect
        li a7, 4 # print string
        ecall

        mv a0,t0
        li a7, 1 #print n
        ecall
    done:
        li a7, 10
        ecall
```

Idea:

Prompt for Input:

- The program prompts the user to enter a positive integer using a do-while loop (simulated by input_loop).
- The integer is read into register t0 and checked to be positive; if not, it is re-prompts the user.

Print Perfect Squares Less Than n:

- i is initialized to 1, and i * i is calculated in t2.
- If i * i < n, the program prints i * i followed by a space, increments i, and repeats.

Check if n is a Perfect Square:

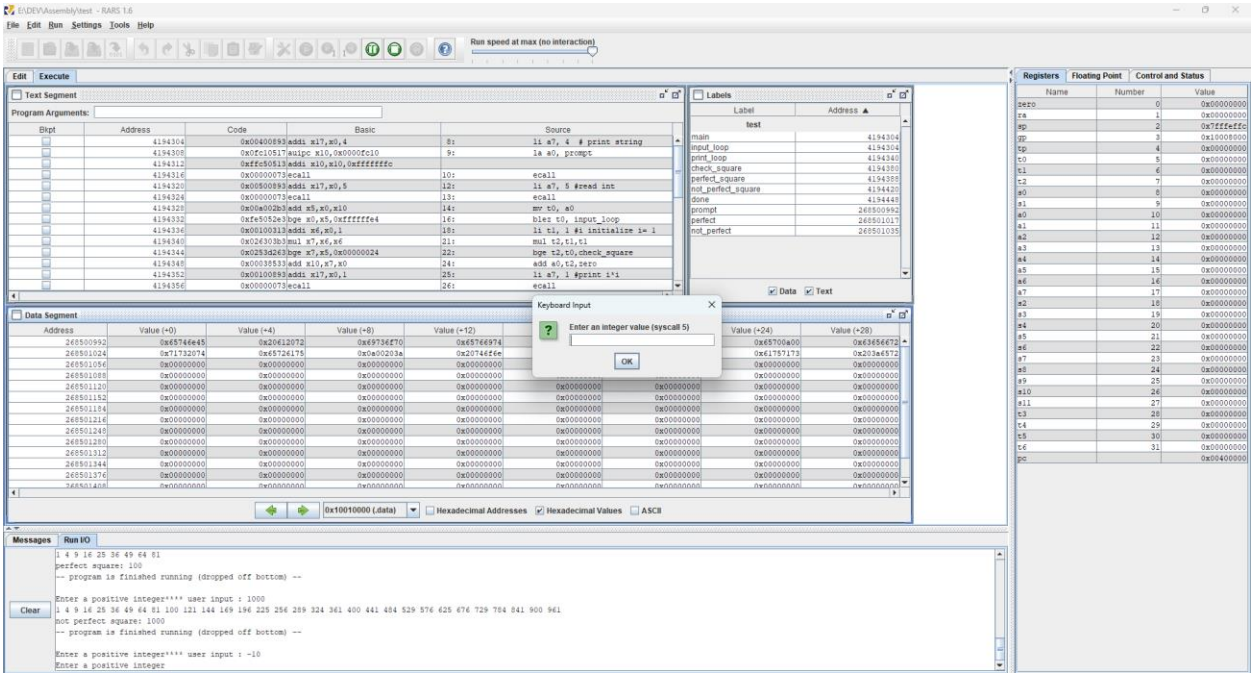
- After exiting the loop, if i * i equals n, it prints that n is a perfect square.

- Otherwise, it prints that n is not a perfect square.

Done:

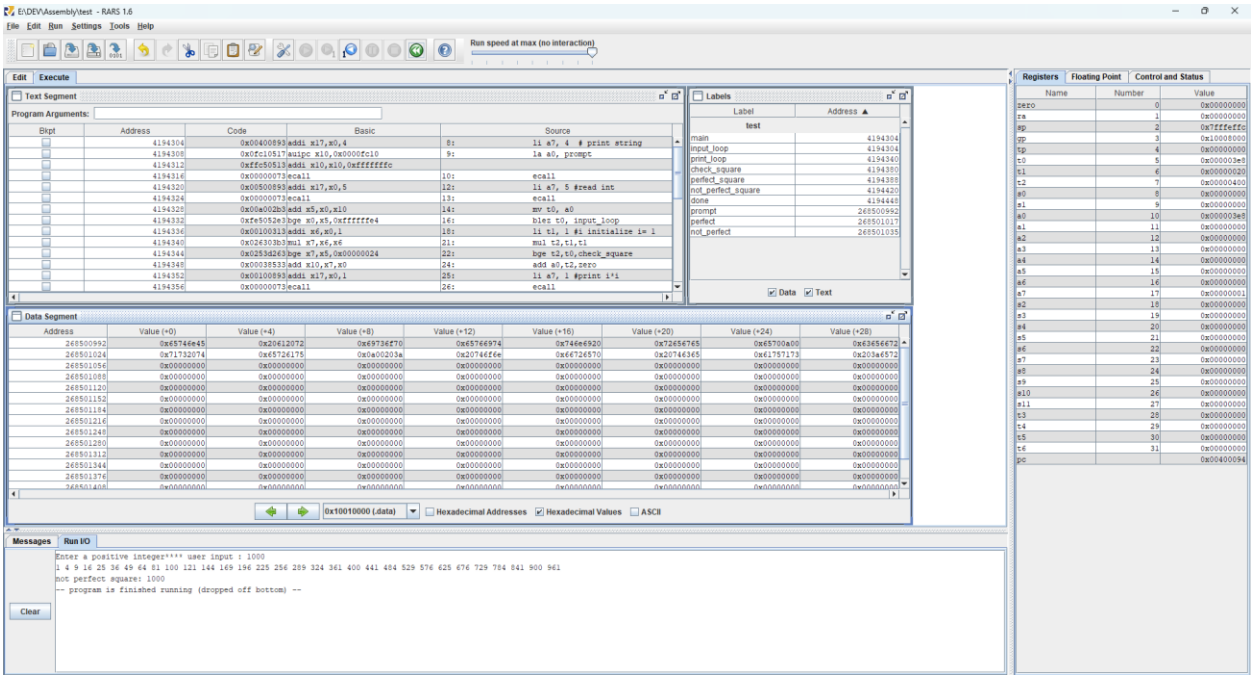
- **The program exits using the exit syscall.**

Handle possible exceptions:



In case of inputting the non-positive integer, you need to input again until this is positive integer. (in this picture, when you input -10 (negative integer) the system forces you to input again.

Result:



Assignment B7:

7. Enter an array of integers from the keyboard. Print out the sum of the negative elements and the sum of the positive elements in the array.

```
.data
prompt_n: .asciz "Enter the number of elements: "
prompt_elements: .asciz "Enter the elements: "
result_pos: .asciz "Sum of positive numbers: "
result_neg: .asciz "Sum of negative numbers: "
arr: .space 400                # Allocate space for up to 100 integers (assuming
max input size)

.text
main:
    li a7, 4                   # Syscall for print_string number of element
    la a0, prompt_n
    ecall

    li a7, 5                   # read int
    ecall
    mv t0, a0                  # t0 = n

    blez t0, exit              # Exit if n <= 0
```

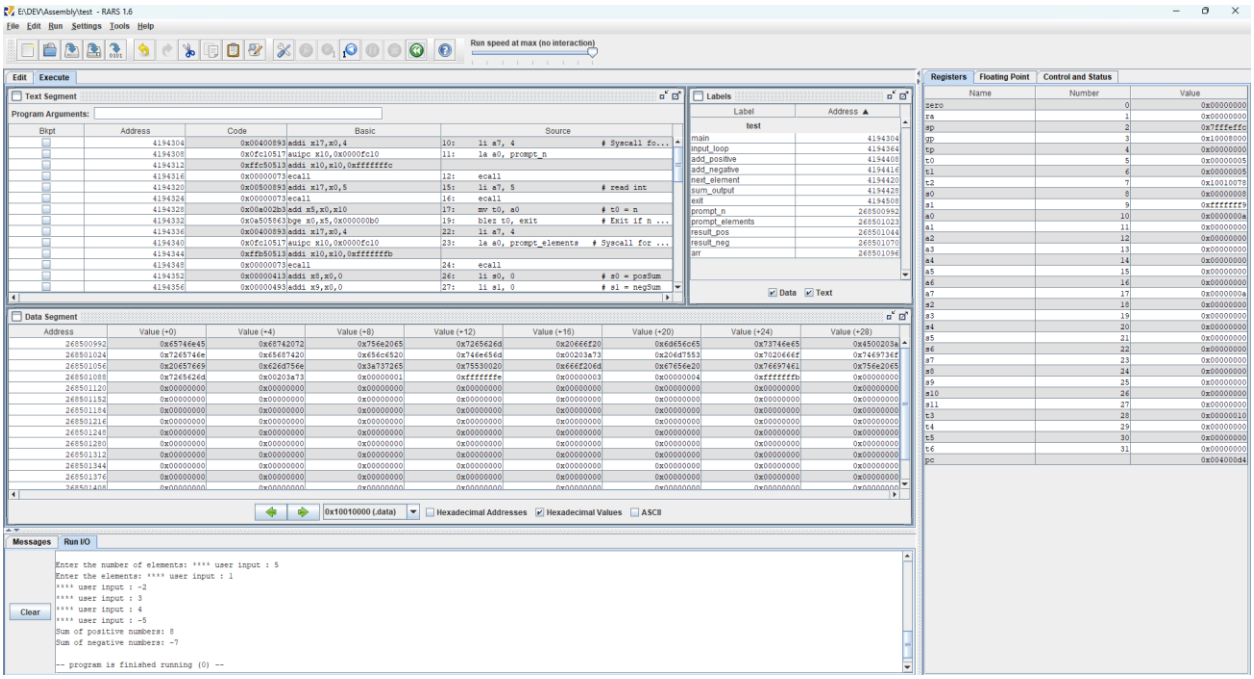
```
    li a7, 4
    la a0, prompt_elements   # Syscall for print_string Enter the elements:
    ecall

    li s0, 0                  # s0 = posSum
    li s1, 0                  # s1 = negSum
    li t1, 0                  # t1 = index counter i

input_loop:
    bge t1, t0, sum_output    # If i >= n, go to output

    li a7, 5                 # Read integer element
    ecall

    # Calculate offset and store input in array
    la t2, arr                # Load base address of array
    slli t3, t1, 2            # t3 = i * 4 (calculate offset)
    add t2, t2, t3            # t2 = arr + offset
    sw a0, 0(t2)              # Store the value in arr[i]

    # Check if the element is positive or negative
    bgtz a0, add_positive     # If element > 0, add to posSum
    bltz a0, add_negative     # If element < 0, add to negSum
    j next_element            # If element == 0, skip

add_positive:
    add s0, s0, a0            # posSum += element
    j next_element

add_negative:
    add s1, s1, a0            # negSum += element

next_element:
    addi t1, t1, 1            # Move to the next index (i++)
    j input_loop              # Repeat the loop


sum_output:
    # Print "Sum of positive numbers: "
    li a7, 4
    la a0, result_pos
    ecall

    # Print posSum
```

```
    mv a0, s0                   # Load posSum into a0 for printing
    li a7, 1                    # Syscall for print_int
    ecall

    # Print newline
    li a0, 10                   # Newline character (ASCII 10)
    li a7, 11                   # Syscall for print_char
    ecall

    # Print "Sum of negative numbers: "
    li a7, 4
    la a0, result_neg
    ecall

    # Print negSum
    mv a0, s1                   # Load negSum into a0 for printing
    li a7, 1                    # Syscall for print_int
    ecall

    # Print newline
    li a0, 10
    li a7, 11
    ecall
exit:
    li a7, 10                   # Exit syscall
    ecall
```

Idea:

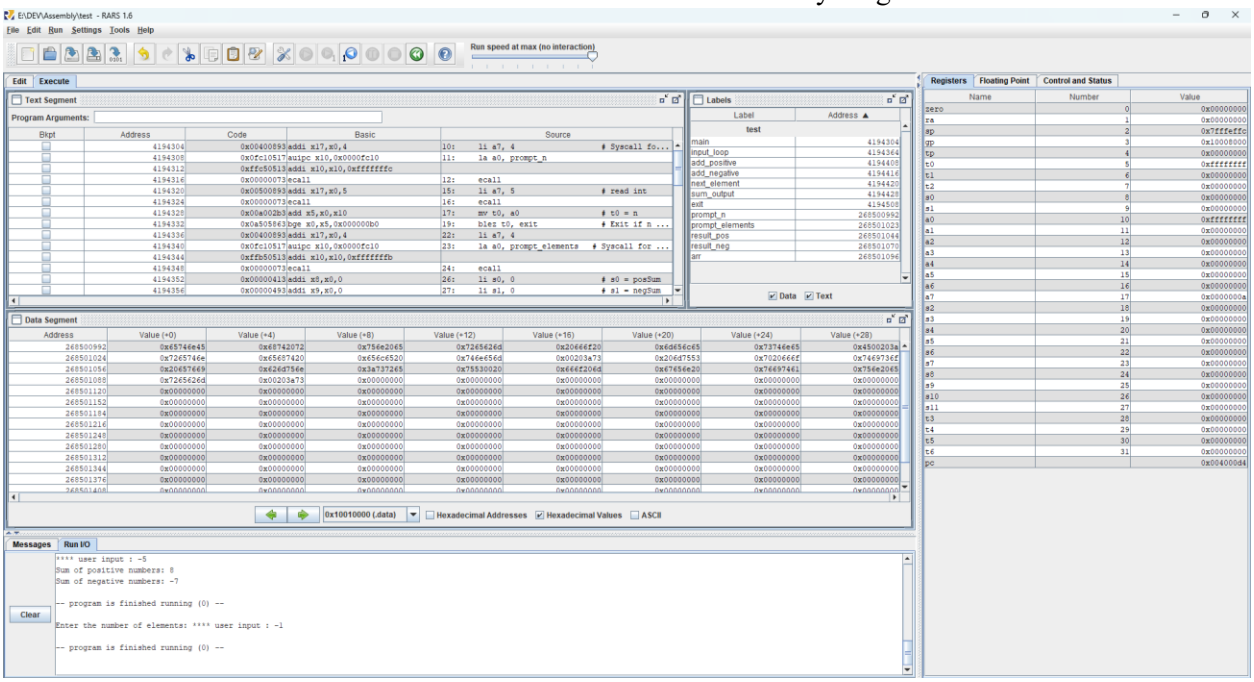
- **Prompt for Input**:
  - The program prompts the user to enter the number of elements (n) and then the elements themselves.
- **Array Storage and Sum Calculation**:
  - Each integer is read and stored in the array arr. If it's positive, it's added to posSum (register s0); if it's negative, it's added to negSum (register s1).
- **Output Sums**:
  - After calculating, the program prints both sums with appropriate labels ("Sum of positive numbers:" and "Sum of negative numbers:").
- **Exit**:
  - The program exits gracefully after printing the results.

Result:



Handle possible exceptions

I consider the case that n <= 0 so that no element so that do not do anything.



Assignment C9:

9. Enter a string, convert the first letter of each word to uppercase and the remaining letters to lowercase. For instance, enter the string "xIn chAO cac bAn", then the result is "Xin Chao Cac Ban".

```
.data
    str: .space 100
.text
 main:
    la a0, str                # Load address of str into a0
    li a7, 8                  # syscall for read_string
    li a1, 100                # Max input length
    ecall                     # Call syscall to read string

    la t0, str                # Load address of the string
    lb t1, 0(t0)              # Load the first character of the string
    li t2, 32                 # Value for ASCII case conversion

    li t6, 'a'           # Idea: first we check the first element that is uper or
not
    bge t1, t6, check_upper

    j main_loop

 check_upper:
  li t6, 'z'
  ble t1, t6, convert_upper

 main_loop:
    addi t0,t0,1        # Move to the next character
    lb t1,0(t0)         # Load the next character
    li t5 ' '
    li t4 '\n'
    li t6, 'A'
    beq t1, t4, done        # End of string check
    beq t1, t5, space_found # If found the space then go to space found and addi
to 1 then check

    bge t1,t6, check_lowercase # Check this element (không phải chữ cái đầu tiên
sau dấu cách) is lowercase or not.
    j main_loop
```

```
space_found:
    addi t0,t0,1
    lb t1, 0(t0)
    li t6, 'a'
    bge t1, t6, check_upper
   j main_loop

 convert_upper:
    sub t1, t1, t2          # Convert to uppercase
    sb t1, 0(t0)            # Store back in str
    j main_loop

check_lowercase:
   li t6, 'Z'
   ble t1, t6, convert_to_lower
   j main_loop

convert_to_lower:
    add t1, t1, t2          # Convert to lowercase
    sb t1, 0(t0)            # Store back in str
    j main_loop

 done:
    # Print the result
    la a0, str              # Load address of str into a0 for print_string
    li a7, 4                # syscall for print_string in RARS
    ecall                   # Call syscall to print the string

    # Exit the program
    li a7, 10               # syscall for exit in RARS
    ecall
```

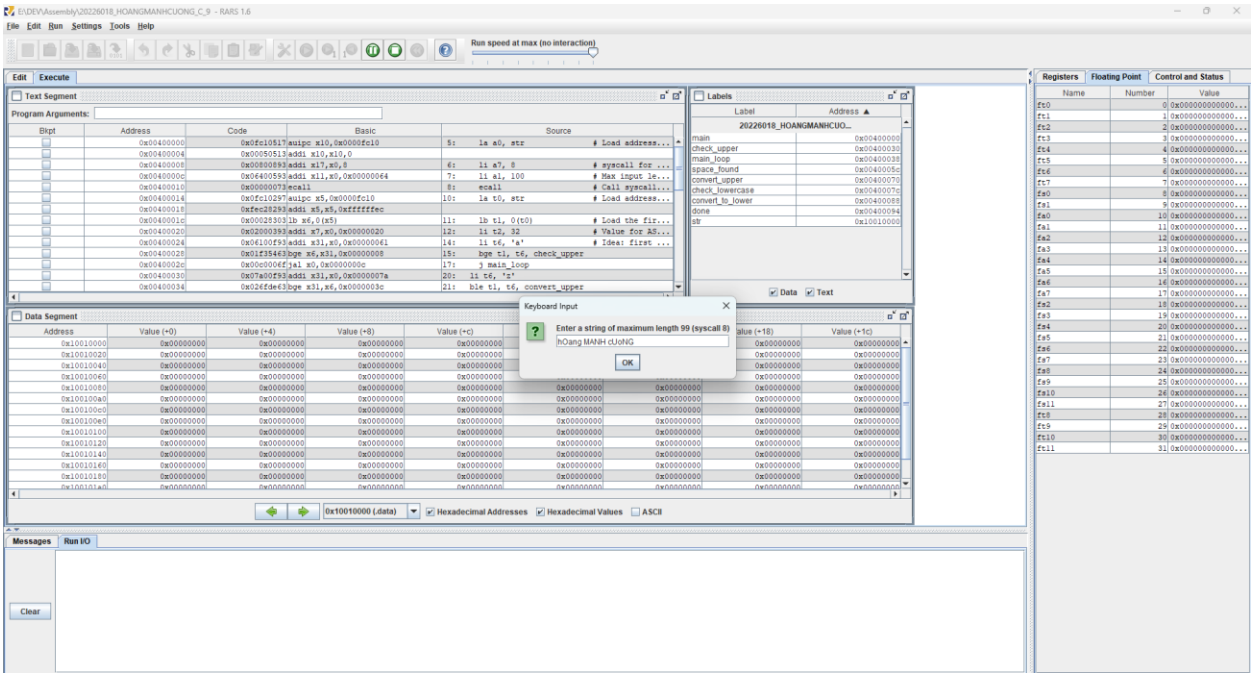
1. **Explain:**

- **Data Section:**
  - str: .space 100: Allocates 100 bytes for the string input.
- **Text Section:**
  - Input and Setup:
    - **la a0, str**: Loads the address of str into a0 (the register expected by the RARS syscall for reading).
    - **li a7, 8** and **li a1, 100**: Prepares syscall 8 (read string) with a maximum length of 100 characters.
    - **ecall**: Invokes the syscall to read the string from the user into str.
- **Looping Through Characters:**

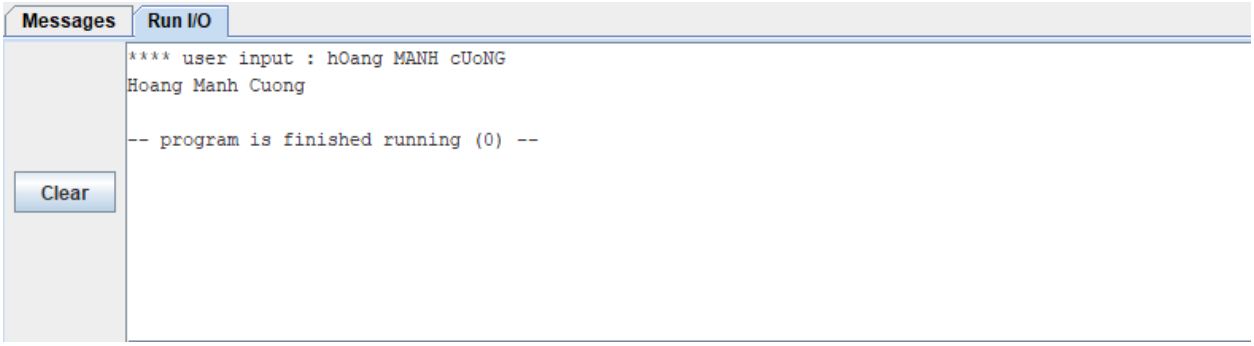
- **la t0, str**: Initializes t0 with the starting address of str to iterate through the string.
- **lb t1, 0(t0)**: Loads the first character of the string into t1.
- Checking and Converting the First Character:
  - If the first character is lowercase (a to z), it's converted to uppercase.
- Main Loop (main_loop):
  - The code iterates through each character until it finds a newline **(\n),** which indicates the end of the string.

- **Character Conversion Logic:**
  - **Identifying Word Boundaries:**
    - If a space (' ') is found, it indicates a boundary between words, so the loop moves to the next character and checks if it's lowercase to make it uppercase.
  - **Uppercase Conversion (convert_upper**):
    - Converts a lowercase letter to uppercase by subtracting 32 (ASCII offset).
  - **Lowercase Conversion (convert_to_lower):**
    - Converts an uppercase letter to lowercase by adding 32 (ASCII offset).
- **Printing the Result:**
  - After processing, the modified string is printed by invoking syscall 4 (print_string).
- **Exiting the Program:**
  - Uses syscall 10 to terminate the program.

First input:



Result:


Messages
Run I/O
**** user input : hOang MANH cUoNG
Hoang Manh Cuong
-- program is finished running (0) --
Clear