

# 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:

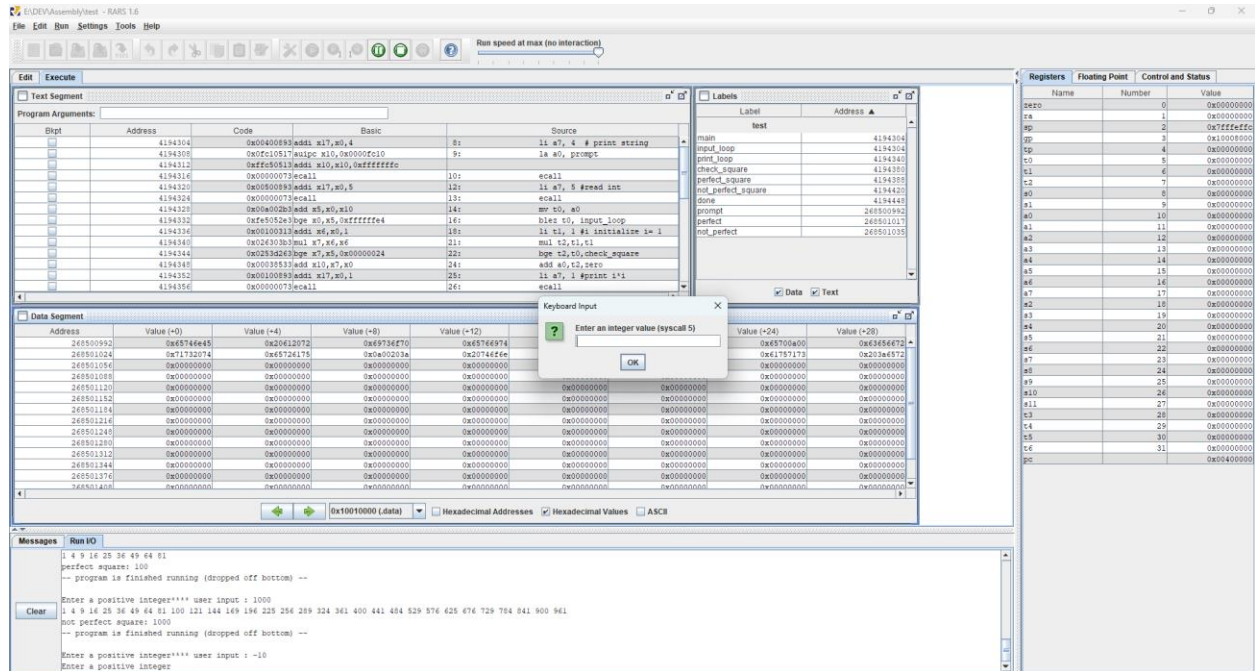
- After exiting the loop, if  $i * i$  equals n, it prints that n is a perfect square.

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

Done:

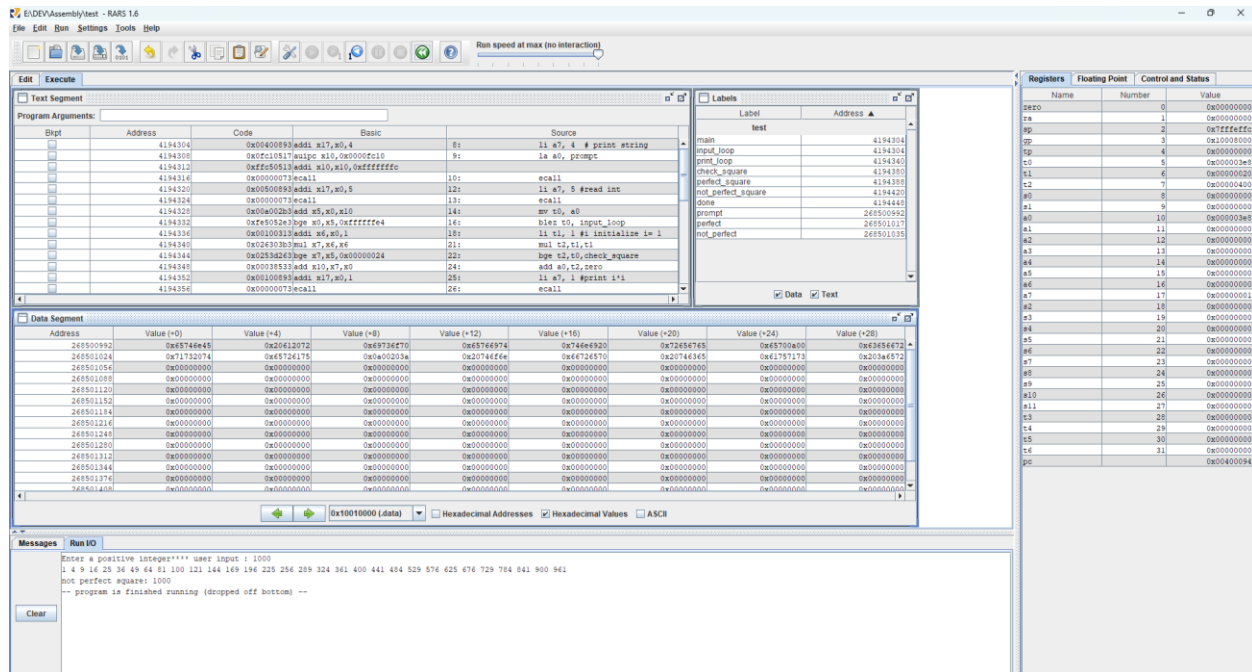
- The program exits using the exit syscall.

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:

- 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:**
  - o The program prompts the user to enter the number of elements (n) and then the elements themselves.
- **Array Storage and Sum Calculation:**
  - o 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:**
  - o After calculating, the program prints both sums with appropriate labels ("Sum of positive numbers:" and "Sum of negative numbers:").
- **Exit:**
  - o The program exits gracefully after printing the results.

#### Result:

ENDEVAssemblytest - RARS 1.6

File Edit Run Settings Tools Help

Run speed at max (no interaction)

**Edit Execute**

Text Segment

Program Arguments:

Offset	Address	Code	Basic	Source
4194304	0x0000893	addi w17,w1,4	10: li a7, 4	# Syscall fo...
4194308	0x0001051	swipc w10,0x0000fc10	11: la a0, prompt_n	
4194312	0x0000813	addi w10,w10,0x0000ffff	12: ecall	
4194316	0x0000007	ecall	12: ecall	
4194320	0x0000893	addi w17,w1,5	15: li a7, 5	# read int
4194324	0x0000007	ecall	16: ecall	
4194328	0x000002b	addi w5,w5,w10	17: wr w5, a0	# w5 = n
4194332	0x0005543	bge w5,a5,0x000000b0	19: bgez w5, exit	# Exit if n ...
4194336	0x0000893	addi w17,w1,4	22: li a7, 4	
4194340	0x0001051	swipc w10,0x0000fc10	23: la a0, prompt_elements	# Syscall for ...
4194344	0x0000813	addi w10,w10,0x0000ffff	24: ecall	
4194348	0x0000007	ecall	24: ecall	
4194352	0x00000a1	addi w5,w5,0	26: li w5, 0	# a0 = possum
4194356	0x0000049	addi w5,w5,0	27: li w5, 0	# a1 = negsum

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
26550992	0x65746e45	0x6742072	0x756e2065	0x72656266	0x02666f20	0x6d656c65	0x73746e45	0x450203a
26551024	0x7265746e	0x62674207	0x746e6266	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551056	0x020203a7	0x62674207	0x746e6266	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551088	0x72656266	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551120	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551152	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551184	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551216	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551248	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551280	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551312	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551344	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551376	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551408	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a

Hexadecimal Addresses Hexadecimal Values ASCII

**Registers** Floating Point Control and Status

Name	Number	Value
\$zero	0	0x00000000
\$ra	1	0x00000000
\$sp	2	0x7fffffc0
\$gp	3	0x10000000
\$tp	4	0x00000000
\$t0	5	0x00000000
\$t1	6	0x00000000
\$t2	7	0x00000000
\$t3	8	0x00000000
\$t4	9	0x00000000
\$t5	10	0x00000000
\$t6	11	0x00000000
\$t7	12	0x00000000
\$t8	13	0x00000000
\$t9	14	0x00000000
\$s0	15	0x00000000
\$s1	16	0x00000000
\$s2	17	0x00000000
\$s3	18	0x00000000
\$s4	19	0x00000000
\$s5	20	0x00000000
\$s6	21	0x00000000
\$s7	22	0x00000000
\$s8	23	0x00000000
\$s9	24	0x00000000
\$a0	25	0x00000000
\$a1	26	0x00000000
\$a2	27	0x00000000
\$a3	28	0x00000000
\$a4	29	0x00000000
\$a5	30	0x00000000
\$a6	31	0x00000000
\$a7	32	0x00000000

**Messages** Run IO

Enter the number of elements: \*\*\*\* user input : 5  
Enter the elements: \*\*\*\* user input : 1  
\*\*\*\* user input : 2  
\*\*\*\* user input : 3  
\*\*\*\* user input : 4  
\*\*\*\* user input : 5  
Sum of positive numbers: 8  
Sum of negative numbers: -7  
-- program is finished running (0) --

## Handle possible exceptions

I consider the case that  $n \leq 0$  so that no element so that do not do anything.

ENDEVAssemblytest - RARS 1.6

File Edit Run Settings Tools Help

Run speed at max (no interaction)

**Edit Execute**

Text Segment

Program Arguments:

Offset	Address	Code	Basic	Source
4194304	0x0000893	addi w17,w1,4	10: li a7, 4	# Syscall fo...
4194308	0x0001051	swipc w10,0x0000fc10	11: la a0, prompt_n	
4194312	0x0000813	addi w10,w10,0x0000ffff	12: ecall	
4194316	0x0000007	ecall	12: ecall	
4194320	0x0000893	addi w17,w1,5	15: li a7, 5	# read int
4194324	0x0000007	ecall	16: ecall	
4194328	0x000002b	addi w5,w5,w10	17: wr w5, a0	# w5 = n
4194332	0x0005543	bge w5,a5,0x000000b0	19: bgez w5, exit	# Exit if n ...
4194336	0x0000893	addi w17,w1,4	22: li a7, 4	
4194340	0x0001051	swipc w10,0x0000fc10	23: la a0, prompt_elements	# Syscall for ...
4194344	0x0000813	addi w10,w10,0x0000ffff	24: ecall	
4194348	0x0000007	ecall	24: ecall	
4194352	0x00000a1	addi w5,w5,0	26: li w5, 0	# a0 = possum
4194356	0x0000049	addi w5,w5,0	27: li w5, 0	# a1 = negsum

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
26550992	0x65746e45	0x6742072	0x756e2065	0x72656266	0x02666f20	0x6d656c65	0x73746e45	0x450203a
26551024	0x7265746e	0x62674207	0x746e6266	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551056	0x020203a7	0x62674207	0x746e6266	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551088	0x72656266	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551120	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551152	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551184	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551216	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551248	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551280	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551312	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551344	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551376	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a
26551408	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x020203a7	0x73746e45	0x450203a

Hexadecimal Addresses Hexadecimal Values ASCII

**Registers** Floating Point Control and Status

Name	Number	Value
\$zero	0	0x00000000
\$ra	1	0x00000000
\$sp	2	0x7fffffc0
\$gp	3	0x10000000
\$tp	4	0x00000000
\$t0	5	0x00000000
\$t1	6	0x00000000
\$t2	7	0x00000000
\$t3	8	0x00000000
\$t4	9	0x00000000
\$t5	10	0x00000000
\$t6	11	0x00000000
\$t7	12	0x00000000
\$t8	13	0x00000000
\$t9	14	0x00000000
\$s0	15	0x00000000
\$s1	16	0x00000000
\$s2	17	0x00000000
\$s3	18	0x00000000
\$s4	19	0x00000000
\$s5	20	0x00000000
\$s6	21	0x00000000
\$s7	22	0x00000000
\$s8	23	0x00000000
\$s9	24	0x00000000
\$a0	25	0x00000000
\$a1	26	0x00000000
\$a2	27	0x00000000
\$a3	28	0x00000000
\$a4	29	0x00000000
\$a5	30	0x00000000
\$a6	31	0x00000000
\$a7	32	0x00000000

**Messages** Run IO

\*\*\*\* user input : -5  
Sum of positive numbers: 8  
Sum of negative numbers: -7  
-- program is finished running (0) --

Enter the number of elements: \*\*\*\* user input : -1  
-- program is finished running (0) --

## 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 upper 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

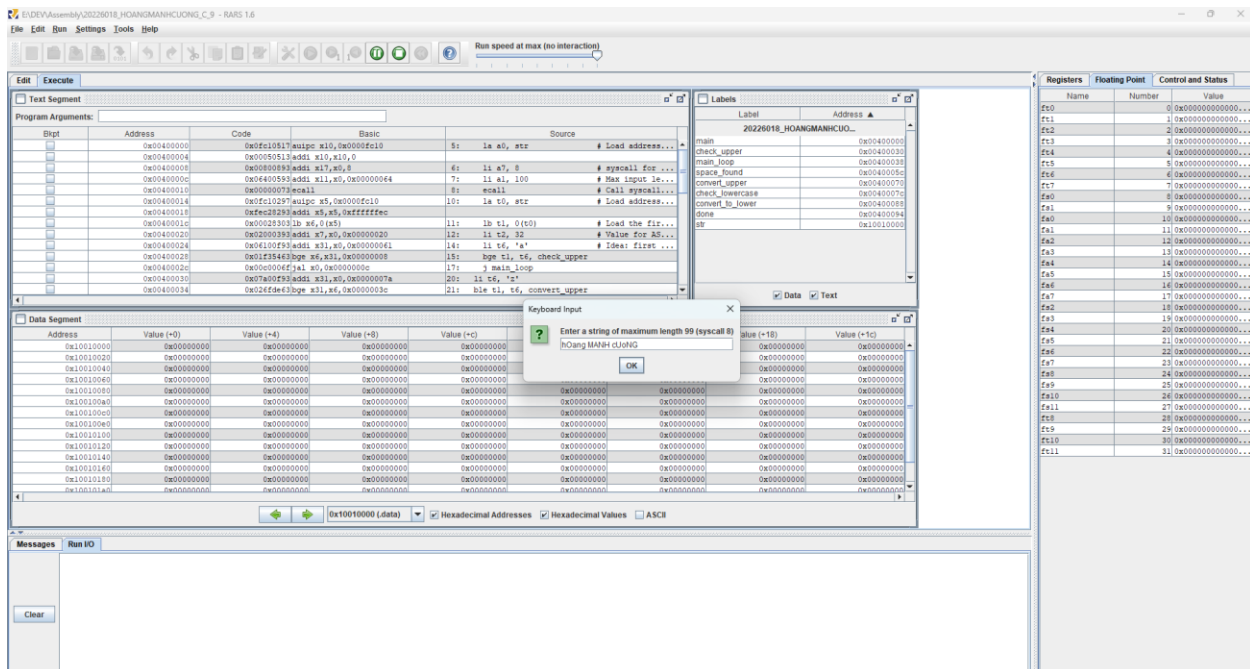
```

## 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, 4 and li a1, 100:** Prepares syscall 4 (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) --
<input type="button" value="Clear"/>	