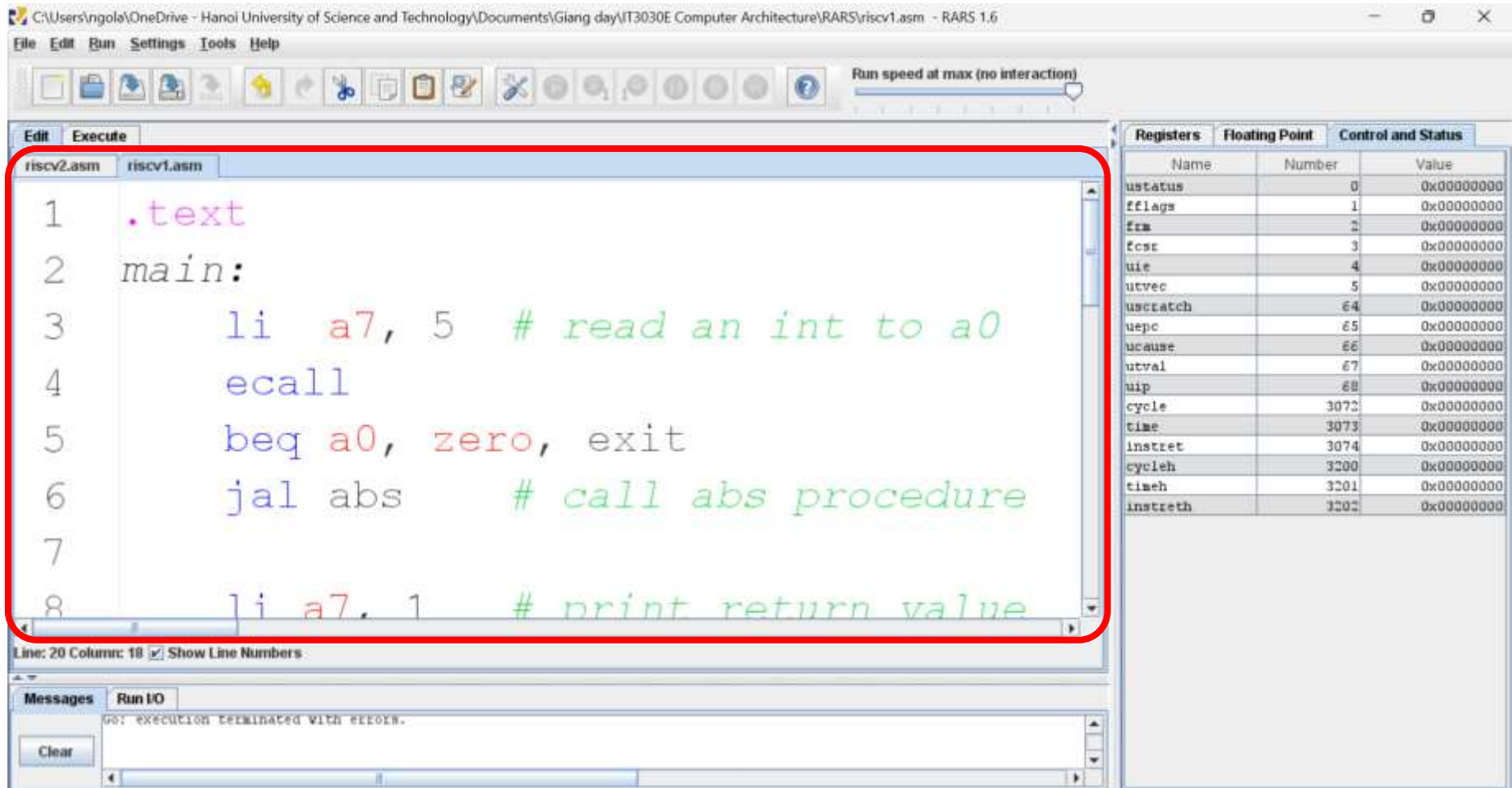

Exercise

Chapter 3. Instruction Set Architecture

RARS 1.6 Introduction



RARS 1.6 Introduction

C:\Users\ngola\OneDrive - Hanoi University of Science and Technology\Documents\Giang day\IT3030E Computer Architecture\RARS\riscv1.asm - RARS 1.6

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

☐ Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00500893	addi x17,x0,5	3: li a7, 5 # read an int to a0
	0x00400004	0x00000073	ecall	4: ecall
	0x00400008	0x00050af3	beq x10,x0,0x00000014	5: beq a0, zero, exit
	0x0040000c	0x018000ef	jal x1,0x00000018	6: jal abs # call abs procedure
	0x00400010	0x00100893	addi x17,x0,1	8: li a7, 1 # print return value
	0x00400014	0x00000073	ecall	9: ecall
	0x00400018	0xfe9ff06f	jal x0,0xffffffff	10: j main
	0x0040001c	0x00a00893	addi x17,x0,10	13: li a7, 10 # terminate
	0x00400020	0x00000073	ecall	14: ecall

☐ Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

Registers Floating Point Control and Status

Name	Number	Value
ustatus	0	0x00000000
fflags	1	0x00000000
frm	2	0x00000000
fcsr	3	0x00000000
mie	4	0x00000000
utvec	5	0x00000000
uscratch	64	0x00000000
uepc	65	0x00000000
ucause	66	0x00000000
utval	67	0x00000000
nip	68	0x00000000
cycle	3072	0x00000000
time	3073	0x00000000
instret	3074	0x00000000
cycleh	3200	0x00000000
timeh	3201	0x00000000
instreth	3202	0x00000000

Messages Run I/O

Assembler: operation completed successfully.

Clear

Hello World 1

.text #Code segment to store instructions

addi t1, zero, 2 # t1 = 2

addi t2, zero, 3 # t2 = 3

add t0, t1, t2 # t0 = t1 + t2

The screenshot shows the RARS 1.6 IDE interface. The main window displays the assembly code for the 'Hello World 1' program. The code is organized into a table with columns for Address, Code, Basic, and Source. The code segment is highlighted with a red box, and the address 0x00400000 is indicated in red text below it.

Code segment address: 0x00400000

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00200313	addi x6,x0,2	2: addi t1, zero, 2 #t1 = 2
	0x00400004	0x00300393	addi x7,x0,3	3: addi t2, zero, 3 #t2 = 3
	0x00400008	0x007302b3	add x5,x6,x7	4: add t0, t1, t2 #t0 = t1 + t2

The right-hand pane shows the Registers window, which lists the registers and their current values. The registers are organized into three columns: Name, Number, and Value. The registers are listed in ascending order of their values.

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeffc
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000005
t1	6	0x00000002
t2	7	0x00000003
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000

Directives

- ❑ Instruct assembler to allocate program objects.
- ❑ Program structure
 - .text: store objects in the code segment (instructions)
 - .data: store objects in data segment (static variables)
- ❑ Variable declaration (allocation)
 - .byte/.half/.word: store listed values as bytes/halves/words
 - .ascii/.asciz: string and null-terminated string
 - .space: reserved specified number of bytes
- ❑ List of directives: see **RARS 1.6 Help**

Hello World 2

```
.data #Data segment to store variables
    X: .word 100
    Y: .word 200
.text #Code segment to store instructions
    la    t0, X
    lw    t1, 0(t0)          #t1 <- X
    la    t0, Y
    lw    t2, 0(t0)          #t2 <- Y
    add   t0, t1, t2          #t0 <- X + Y
```


Hello World 1

C:\Users\ngola\OneDrive - Hanoi University of Science and Technology\Documents\Giang day\IT3030E Computer Architecture\RARS\HelloWorld1.asm - RARS 1.6

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x0fc10297	auipc x5,0x0000fc10	5: la t0, X
	0x00400004	0x00028293	addi x5,x5,0	
	0x00400008	0x0002a303	lw x6,0(x5)	6: lw t1, 0(t0) #t1 <- X
	0x0040000c	0x0fc10297	auipc x5,0x0000fc10	7: la t0, Y

Code segment address: 0x00400000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x000000e4	0x000000c8	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010004	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010008	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001000c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010010	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010018	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001001c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

X and Y

Data segment address: 0x10010000

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Messages Run I/O

Assembly: operation completed successfully.

Clear

Registers Floating Point Control and Status

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeffc
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
...	...	...

Exercise

- ❑ Draw the map of this data segment

.data

```
arr16: .space 16
arr:   .byte 'a','b','c'
st1:   .asciz "abcd"
X1:    .word 0xFFFFFFFF
st2:   .ascii "efgh"
X2:    .word 0xFFFFFFFF
st3:   .asciz "1234"
X3:    .word 0xFFFFFFFF
st4:   .asciz "5678"
X4:    .word 0xFFFFFFFF
```


❑ Data segment map

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x61636261	0x00646362	0xffffffff	0x68676665
0x10010020	0xffffffff	0x34333231	0x00000000	0xffffffff	0x38373635	0x00000000	0xffffffff	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Navigation: 0x10010000 (.data) ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	a c b a	\0 d c b		h g f e
0x10010020		4 3 2 1	\0 \0 \0 \0		8 7 6 5	\0 \0 \0 \0		\0 \0 \0 \0
0x10010040	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010060	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x10010080	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100a0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0
0x100100c0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

Navigation: 0x10010000 (.data) ☒ Hexadecimal Addresses ☒ Hexadecimal Values ☒ ASCII

Exercise 3.1

- ❑ 1.1: Write the RISC-V assembly code equivalent to the following C statement.

$$f = g + (h - 5);$$

- ❑ Assume that the variables f , g , and h , have already been placed/allocated in registers $x5$, $x6$, and $x7$ respectively.
- ❑ Note: a minimal number of RISC-V instructions should be used.
- ❑ 1.2: Write a complete assembly program that
 - | Declares integer variables f , g , h .
 - | Initialize values for f , g , h .
 - | Calculate and do the assignment: $f = g + (h - 5)$

.data

f:.word 0

g:.word 100

h:.word 300

.text

la x8, h

lw x7, 0(x8) #load h to x7

la x8, g

lw x6, 0(x8) #load g to x6

addi x7, x7, -5

add x5, x6, x7 #calculate

la x8, f

sw x5, 0(x8) #store to f

Exercise 3.2

- ❑ 2.1 Write the RISC-V assembly code equivalent to the below C statement.

$B[4] = A[i-j];$

- ❑ Assume that the variables i , and j are allocated in registers $x28$, and $x29$ respectively. A and B are two integer array, with the base address of A and B are in registers $x10$ and $x11$ respectively.
- ❑ 2.2 Write the complete program in RARS 1.6 that
 - | Declares two integer arrays A and B with at least 6 elements for each array.
 - | Do the operation $B[4] = A[i-j];$ with i and j in $x28$ and $x29$.

.data

A: .space 100 #25x4-byte integers each
B: .space 100

.text

```
la a0, A
la a1, B
li t0, 100
sw t0, 0(a0)      #initialize A[0]=100
sub t3, t3, t4      #i-j, now i and j are both 0
slli t4, t3, 2      #offset of A[i-j]
add a0, a0, t4      #a0 hold addr A[i-j]
lw t5, 0(a0)      #load A[i-j]
sw t5, 32(a1)      #store to B[8]
```

Exercise 3.3

- ❑ Reverse-compile the following RISC-V assembly code to the equivalent C code. Assume that the variables f, g, h, i, and j are allocated at registers x5, x6, x7, x28, and x29, respectively, and the base address of the arrays A and B are in registers x10 and x11.

```
slli x30, x5, 2
add x30, x10, x30
slli x31, x6, 2
add x31, x11, x31
lw x5, 0(x30)
addi x12, x30, 8
lw x30, 0(x12)
add x30, x30, x5
lw x30, 0(x31)
```

Exercise 3.4

- ❑ Find the format and the machine code of the following instruction:

`sw x5, 32(x30)`

RISC-V system services (syscall)

- ❑ Basic input/output services provided by RISC-V system.
 - | Print string, print character, print integer/floating point to output console.
 - | Read string, char, int, float/double from input console.
 - | Other function: random, dialog,...
- ❑ Usage:
 - | Step 1. Load the service number in register a7.
 - | Step 2. Load argument values, if any, in a0, a1, a2, a3, fa0, ... as specified.
 - | Step 3. Issue the ECALL instruction.
 - | Step 4. Retrieve return values, if any, from result registers as specified
- ❑ List of system services: see **RARS 1.6 Help**

System service: print integer

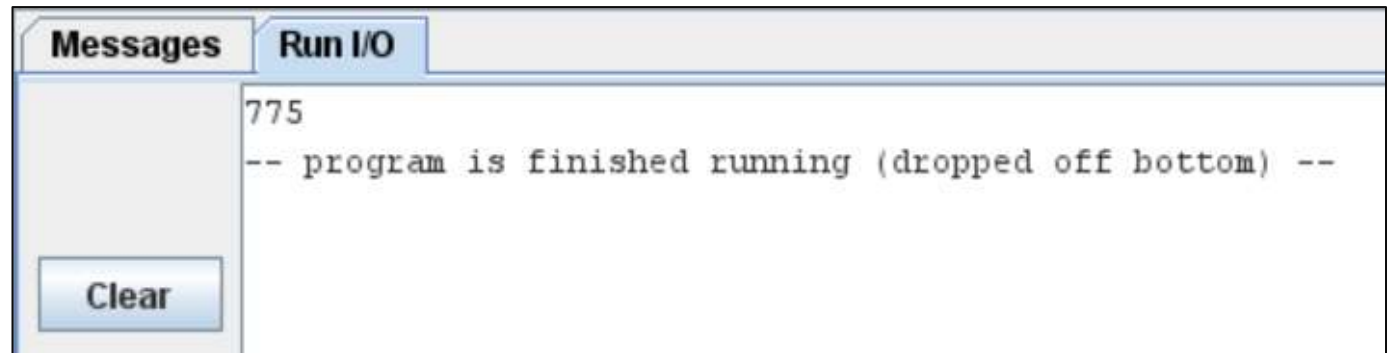
❑ Input params:

- | a7 = 1
- | a0 = [integer value to print].

❑ Example:

```
li a7, 1
li a0, 0x307
ecall
```

❑ Result:



System service: print string

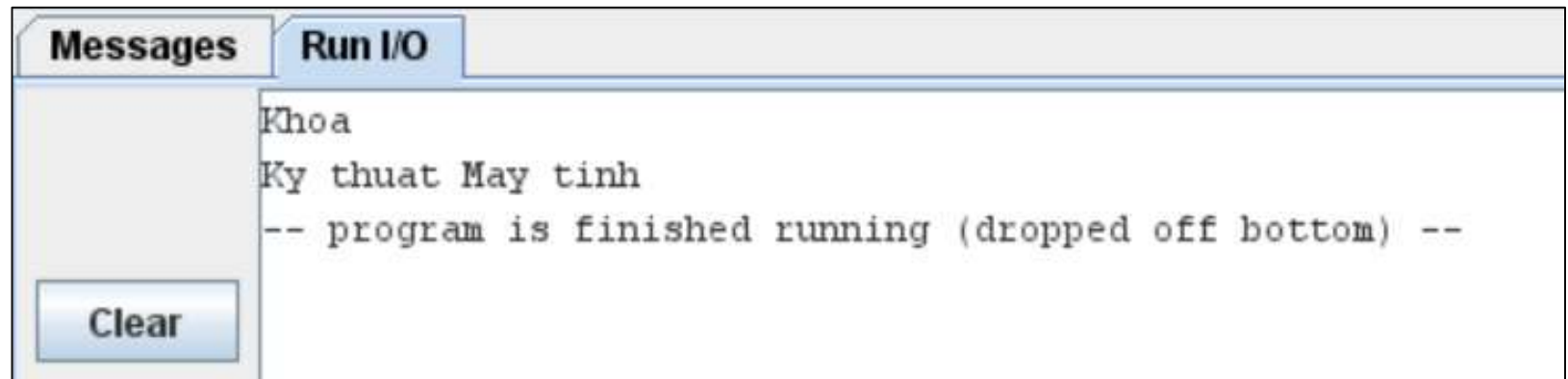
❑ Input params:

- | a7 = 4
- | a0 = address of the string to print

❑ Example:

```
.data
message: .asciz "Khoa \nKy thuat May tinh"
.text
li a7, 4
la a0, message
ecall
```

❑ Result



System service: read integer

- ❑ Input params

- | a7 = 5

- ❑ Return

- | a0 = integer value read

- ❑ Example:

```
li a7, 5  
ecall
```

- ❑ Result: try run the above code on RARS 1.6

- | The ecall instruction activates the input caret at the in/out console waiting for user input.
 - | User types in an integer and press Enter → the integer is loaded to a0.

Hello World 3

- ❑ Write a program
 - Print “Hello!” to console
 - Wait for user to press any key
 - Print “Have a good day!”

```
#simple syscall example for input/output
.data
    #this is where we declare variables
    hello: .asciz "Hello!\n"
    goodday: .asciz "Have a good day!\n"
.text
main:
    #this is where our code goes
    #print hello string
    li a7, 4
    la a0, hello
    ecall
    #wait for user to press on keyboard
    li a7, 12
    ecall
    #then say good day
    li a7, 4
    la a0, goodday
    ecall
endmain:
```

Exercise 3.5

- ❑ Write a program to:
 - | Input an integer X from console
 - | Output to screen the appropriate string:
 - “odd” if X is an odd number
 - “even” if X is an even number

Exercise 3.6

- ❑ 1. Write a subprogram/function to:
 - | Get an integer X as input argument
 - | Output the absolute value of X
- ❑ 2. Use the above function to write a program that reads an input integer from input console, then calculate and display the absolute value of that integer.
- ❑ 3. Modify the above program so that it repeats the above process continuously until it reads the value 0 from input console.

```
.text
main:
    li a7, 5                # read an int to a0
    ecall
    beq a0, zero, exit      # exit if input = 0
    jal abs                 # call abs procedure
    li a7, 1                # print return value
    ecall
    j main
exit: li a7, 10             # terminate, WHY is this necessary?
    ecall                  # see system service 10
end_main:

abs: bge a0, zero, done     # if a0>=0 done
    sub a0, zero, a0        # else negate a0
done:
    jr ra
```

Exercise 3.7

- ❑ Write a program to read a positive value n from console, then calculate and print the sum of all numbers from 1 to n .
- ❑ Implement the function of calculate sum as a procedure with below parameters:
 - | a7: n (upper boundary)
 - | a0: return value of $(1+2+..+n)$

Exercise 3.8

- ❑ Develop a simple implementation of the `strlen()` function

`size_t strlen(const char *str)`

- ❑ Write a program to input a string (max of 255 chars), then print out its length