



香港中文大學

The Chinese University of Hong Kong

CENG3420

Lab 1-1: RISC-V Assembly Language Programming I

Chen BAI, Su ZHENG

Department of Computer Science & Engineering

Chinese University of Hong Kong

cbai@cse.cuhk.edu.hk,

szheng22@cse.cuhk.edu.hk

Spring 2023

- 1 Introduction to Basic RISC-V Assembly Programming
- 2 RARS
- 3 System Service in RARS
- 4 Lab 1-1 Assignment

Introduction to Basic RISC-V Assembly Programming

- The RISC-V Instruction Set Manual Volume I: Unprivileged ISA
<https://riscv.org/technical/specifications/>

In all labs. of CENG3420, we focus on RV32I instructions.

An Example Program

```
1  .globl _start
2
3  .data
4  welcome_msg: .asciz "Welcome to ENG3420!\n"
5
6  .text
7  _start:
8      # STDOUT = 1
9      addi a0, x0, 1
10     # Load the address of `welcome_msg`
11     la a1, welcome_msg
12     # length of the string
13     addi a2, x0, 21
14     # linux write system call
15     addi a7, x0, 64
16     # Call linux service to output the string
17     ecall
18
```

The screenshot displays the Visual Studio Code interface during the execution of a program. The top menu bar includes File, Edit, Run, Settings, Tools, and Help. The main workspace is divided into several panels:

- Text Segment:** This panel shows the assembly code being executed. It includes columns for Bkpt, Address, Code, Basic, and Source. The code consists of several instructions, including `addi x10, x0, 1`, `la a1, welcome_msg`, `addi x11, x11, 0xffff...`, `addi x12, x0, 21`, `addi x17, x0, 0x00000040`, and `ecall`.
- Data Segment:** This panel shows the memory addresses and their corresponding values. It includes columns for Address, Value (+0), Value (+4), Value (+8), Value (+c), Value (+10), Value (+14), Value (+18), and Value (+1c). The values are mostly zero, indicating that the program has not yet written to memory.
- Control and Status:** This panel displays the state of the system, including registers and floating-point values. It includes columns for Name, Number, and Value. The registers are mostly zero, and the floating-point values are also zero.
- Messages:** This panel shows the output of the program. It includes a "Welcome to ENG3420!" message and a confirmation that the program is finished running.

The interface also features a toolbar at the top with various icons for file operations, editing, and running. A "Run speed at max (no interaction)" slider is visible in the top right corner.

- We can manipulate 32 architectural registers in assembly programming directly.
- We prefer using aliases to indicate registers.
- Instructions category
 - Load and store instructions
 - Bitwise instructions
 - Arithmetic instructions
 - Control transfer instructions
 - Pseudo instructions

Register Names and Descriptions

Table: Register names and descriptions

Register Names	ABI Names	Description
x0	zero	Hard-wired zero
x1	ra	Return address
x2	sp	Stack pointer
x3	gp	Global pointer
x4	tp	Thread pointer
x5	t0	Temporary / Alternate link register
x6-7	t1 - t2	Temporary register
x8	s0 / fp	Saved register / Frame pointer
x9	s1	Saved register
x10-11	a0-a1	Function argument / Return value registers
x12-17	a2-a7	Function argument registers
x18-27	s2-s11	Saved registers
x28-31	t3-t6	Temporary registers

Data types:

- All instructions are encoding in 32 bits
- Alias: byte (8 bits), halfword (2 bytes), word (4 bytes), double word (8 bytes)

Literals:

- numbers entered as is. *e.g.*, 12 in decimal, and 0xC in hexadecimal
- characters enclosed in single quotes. *e.g.*, 'b'
- strings enclosed in double quotes. *e.g.*, "A string"

Program Structure I

- Plain text file with data declarations, program code (name of file can be suffixed with *.asm*)
- Data declaration section is followed by program code section

Data Declarations

- Identified with assembler directive **.data**
- Declares variable names used in program
- Storage allocated in main memory (*e.g.*, RAM)
- `<name>: .<datatype> <value>`

Program Structure II

Code

- placed in section of text identified with assembler directive **.text**
- contains program code (instructions)
- starting point for code e.g. execution given label **start:**

Comments

Anything following # on a line

The structure of an assembly program looks like this:

Program outline

```
# Comment giving name of program and description
# Template.asm
# Bare-bones outline of RISC-V assembly language program

.globl _start

.data    # variable declarations follow this line
        # ...
.text    # instructions follow this line

_start: # indicates start of code
# ...

# End of program, leave a blank line afterwards is preferred
```

Instructions Overview I

LA: The Load Address (*la*) loads the location address of the specified SYMBOL.

Syntax

`la rd, SYMBOL`

Usage

```
.data
NumElements: .byte 6
.text
la x5, NumElements # assign addr[NumElements] to x5
```

LI: The Load Immediate (LI) loads a register (rd) with an immediate value given in the instruction.

Syntax

`li rd, CONSTANT`

Usage

```
li x5,100 # assign 100 to x5
```

LD: The Load Double word (LD) instruction does the fetching of 64-bit value from memory and loads into the destination register (rd).

Syntax

```
ld rd, offset(rs1)
```

Usage

```
ld x4, 1352(x9) # assign memory[x9+1352] to x4
```

SD: The Store Double word (SD) instruction does the copying of 64-bit value from register (rs2) and loads into the memory(rs1).

Instructions Overview III

Syntax

`sd rs2, offset(rs1)`

Usage

```
sd x4, 1352(x9) # assign mem[x9+1352] to x4
```

SLL: Shift Logical Left (SLL) performs logical left on the value in register (rs1) by the shift amount held in the register (rs2) and stores in (rd) register.

Syntax

`sll rd, rs1, rs2`

Usage

Instructions Overview IV

```
li x5, 4 # assign 4 to x5
li x3, 2 # assign 2 to x3
sll x1, x5, x3 # assign x5 << x3 to x1
```

SRL: Shift Logically Right (SRL) performs logical Right on the value in register (rs1) by the shift amount held in the register (rs2) and stores in (rd) register.

Syntax

srl rd, rs1, rs2

Usage

```
li x5, 1024 # assign 1024 to x5
li x3, 2     # assign 2 to x3
srl x1, x5, x3 # assign x5 >> x3 to x1
```


SLLI: Shift Logically Left Immediate (SLLI) performs logical left on the value in register (rs1) by the shift amount held in the register (imm) and stores in (rd) register.

Syntax

```
slli rd, rs1, imm
```

Usage

```
slli x1, x1, 3 # assign x1 << 3 to x1
```

SRLI: Shift Logically Right Immediate (SRLI) performs logical Right on the value in register (rs1) by the shift amount held in the register (imm) and stores in (rd) register.

Syntax

`srli rd, rs1, imm`

Usage

```
srli x1, x1, 1 # assign x1 >> 1 to x1
```

For more information about RISC-V instructions and assembly programming you can refer to:

- 1 Lecture slides and textbook.
- 2 **RARS** Help: F1
- 3 <https://github.com/riscv/riscv-asm-manual/blob/master/riscv-asm.md>
- 4 <https://web.eecs.utk.edu/~smarz1/courses/ece356/notes/assembly/>

RISC-V ISA Simulator – RARS

What is RARS

- **RARS is the RISC-V Assembler, Runtime and Simulator for RISC-V assembly language programs**
- **RARS** supports RISC-V IMFDN ISA base (riscv32 & riscv64).
- **RARS** supports debugging using breakpoints like *ebreak*.
- **RARS** supports side by side comparison from psuedo-instruction to machine code with intermediate steps.
- You need Java environment to run **RARS**

Download it here:

<https://github.com/TheThirdOne/rars/releases/tag/continuous>

Execute the command to start RARS: `java -jar <rars jar path>`

Launch RARS

```
cbai@hpc1:/research/dept8/gds/cbai/ta/rars$ java -jar rars.jar
```

Launch RARS

RARS Overview

Edit Run Settings Help
Run speed at max (no interaction)

Edit Execute

test.asm

```

88 # 12 "isa/rv64ui/srliw.S" 2
89
90
91 .test
92 .globl _start
93 _start: nop
94
95 #-----
96 # Arithmetic tests
97 #-----
98
99     test_2: li x1, 0xffffffff80000000
100            srliw x14, x1, 0
101            li x7, 0xffffffff80000000
102            li x0, 2
103            bne x14, x7, fail
104
105     test_3: li x1, 0xffffffff80000000
106            srliw x14, x1, 1
107            li x7, 0x0000000040000000
108            li x0, 3
109            bne x14, x7, fail
110
111     test_4: li x1, 0xffffffff80000000

```

Line: 100 Column: 18 | Show Line Numbers

Registers Floating Point Control and Status

Index	Name	Number	Value
x+		0	0x0000000000000000
x+		1	0x0000000000000000
x+		2	0x00000000ffff0000
f+		3	0x0000000010000000
f+		4	0x0000000000000000
f+		5	0x0000000000000000
f+		6	0x0000000000000000
f+		7	0x0000000000000000
f+		8	0x0000000000000000
f+		9	0x0000000000000000
f+		10	0x0000000000000000
f+		11	0x0000000000000000
f+		12	0x0000000000000000
f+		13	0x0000000000000000
f+		14	0x0000000000000000
f+		15	0x0000000000000000
f+		16	0x0000000000000000
f+		17	0x0000000000000000
f+		18	0x0000000000000000
f+		19	0x0000000000000000
f+		20	0x0000000000000000
f+		21	0x0000000000000000
f+		22	0x0000000000000000
f+		23	0x0000000000000000
f+		24	0x0000000000000000
f+		25	0x0000000000000000
f+		26	0x0000000000000000
f+		27	0x0000000000000000
f+		28	0x0000000000000000
f+		29	0x0000000000000000
f+		30	0x0000000000000000
f+		31	0x0000000000000000
+			0x0000000000004000

Messages Run IO

```

Assemble: assembling F:\Research\misc\IA\CEBQ420\tools\test.asm
Parsing in F:\Research\misc\IA\CEBQ420\tools\test.asm line 312 column 2: RARS does not recognize the global directive. Ignored.
Parsing in F:\Research\misc\IA\CEBQ420\tools\test.asm line 318 column 2: RARS does not recognize the global directive. Ignored.
Assemble: operation completed successfully.

```

Clear

File Edit Bus Settings Tools Help

Run speed at max (no interaction)

Editor **Execute**

Text Segment

Bit	Address	Code	Basic	Source
0x040000	0x00000013	add, r0, r0, 0	93	_start, nop
0x040004	0x00000015	li, r1, 0xffffffff00000000	99	test, 2, li, r1, 0xffffffff00000000
0x040008	0x00000019	add, r1, r1, 0	100	mov, r1, r1, 0
0x04000c	0x0000001d	li, r1, 0xffffffff00000000	101	li, r1, 0xffffffff00000000
0x040010	0x00000021	add, r1, r1, 0	102	li, r1, 0
0x040014	0x00000025	li, r1, 0	103	li, r1, 0
0x040018	0x00000029	li, r1, 0	104	li, r1, 0
0x04001c	0x0000002d	li, r1, 0	105	li, r1, 0
0x040020	0x00000031	li, r1, 0	106	li, r1, 0
0x040024	0x00000035	li, r1, 0	107	li, r1, 0
0x040028	0x00000039	li, r1, 0	108	li, r1, 0
0x04002c	0x0000003d	li, r1, 0	109	li, r1, 0
0x040030	0x00000041	li, r1, 0	110	li, r1, 0
0x040034	0x00000045	li, r1, 0	111	li, r1, 0

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
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
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010028	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001002c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010030	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010034	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010038	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001003c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010044	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010048	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001004c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010050	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

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

Messages **Run IO**

Assembly: assembling F:\Research\misc\TACER\32\tools\test.asm

Warning in F:\Research\misc\TACER\32\tools\test.asm line 312 column 2: RARS does not recognize the global directive. Ignored.

Warning in F:\Research\misc\TACER\32\tools\test.asm line 312 column 2: RARS does not recognize the global directive. Ignored.

Assembly: operation completed successfully.

Registers **Floating Point** **Control and Status**

Name	Number	Value
r0	0	0x0000000000000000
r1	1	0x0000000000000000
r2	2	0x0000000000000000
r3	3	0x0000000000000000
r4	4	0x0000000000000000
r5	5	0x0000000000000000
r6	6	0x0000000000000000
r7	7	0x0000000000000000
r8	8	0x0000000000000000
r9	9	0x0000000000000000
r10	10	0x0000000000000000
r11	11	0x0000000000000000
r12	12	0x0000000000000000
r13	13	0x0000000000000000
r14	14	0x0000000000000000
r15	15	0x0000000000000000
r16	16	0x0000000000000000
r17	17	0x0000000000000000
r18	18	0x0000000000000000
r19	19	0x0000000000000000
r20	20	0x0000000000000000
r21	21	0x0000000000000000
r22	22	0x0000000000000000
r23	23	0x0000000000000000
r24	24	0x0000000000000000
r25	25	0x0000000000000000
r26	26	0x0000000000000000
r27	27	0x0000000000000000
r28	28	0x0000000000000000
r29	29	0x0000000000000000
r30	30	0x0000000000000000
r31	31	0x0000000000000000
pc		0x0000000000000000

RARS Basic Introduction

The screenshot displays the RARS (RISC Assembler RISC Assembler) interface, which is divided into several panels:

- Tools panel:** Located at the top right, it contains a toolbar with icons for file operations (open, save, print, etc.) and a status bar indicating "Run speed at max (no interaction)".
- Source codes panel:** The central area displays the assembly code being processed. The code includes comments, directives, and instructions for a MIPS-like architecture. It starts with a comment "# 12 'isa/rv64ui/rvliw.5' 2", followed by a test section, a global directive, and several arithmetic tests.
- Registers panel:** Located on the right side, it shows a table of registers. The table has columns for Name, Number, and Value. It lists registers from \$zero to \$31, with their corresponding values in hexadecimal.
- Program information panel:** Located at the bottom, it displays messages and run information. It shows the assembly path, the file being processed, and the status of the assembly process.

The source code in the Source codes panel is as follows:

```
# 12 'isa/rv64ui/rvliw.5' 2
.test
.global _start
_start: nop

#-----
# Arithmetic tests
#-----

test_2: li $1, 0xffffffff80000000
rliw $14, $1, 0
li $7, 0xffffffff80000000
li $0, 2
bne $14, $7, fail

test_3: li $1, 0xffffffff80000000
rliw $14, $1, 1
li $7, 0x0000000004000000
li $0, 3
bne $14, $7, fail

test_4: li $1, 0xffffffff80000000
rliw $14, $1, 1
```

The Registers panel shows the following registers and values:

Name	Number	Value
\$zero	0	0x0000000000000000
\$ra	1	0x0000000000000000
\$sp	2	0x0000000000000000
\$gp	3	0x0000000000000000
\$tp	4	0x0000000000000000
\$t0	5	0x0000000000000000
\$t1	6	0x0000000000000000
\$t2	7	0x0000000000000000
\$t3	8	0x0000000000000000
\$t4	9	0x0000000000000000
\$t5	10	0x0000000000000000
\$t6	11	0x0000000000000000
\$t7	12	0x0000000000000000
\$t8	13	0x0000000000000000
\$t9	14	0x0000000000000000
\$s0	15	0x0000000000000000
\$s1	16	0x0000000000000000
\$s2	17	0x0000000000000000
\$s3	18	0x0000000000000000
\$s4	19	0x0000000000000000
\$s5	20	0x0000000000000000
\$s6	21	0x0000000000000000
\$s7	22	0x0000000000000000
\$s8	23	0x0000000000000000
\$s9	24	0x0000000000000000
\$s10	25	0x0000000000000000
\$s11	26	0x0000000000000000
\$s12	27	0x0000000000000000
\$s13	28	0x0000000000000000
\$s14	29	0x0000000000000000
\$s15	30	0x0000000000000000
\$s16	31	0x0000000000000000
\$s17	32	0x0000000000000000

The Program information panel shows the following messages:

```
Assembly: assembling F:\Research\misc\TA\CBM32420\tools\test.asm
Parsing in F:\Research\misc\TA\CBM32420\tools\test.asm line 212 column 2: RARS does not recognize the .global directive. Ignored.
Parsing in F:\Research\misc\TA\CBM32420\tools\test.asm line 318 column 2: RARS does not recognize the .global directive. Ignored.
Assembly: operation completed successfully.
```

File Edit Bus Settings Tools Help

Run speed at max (no interaction)

Tools panel

Text segment panel

Bit	Address	Code	Basic	Source
0x040000	0x00000013	addi r0, r0, 0	93	_start: nop
0x040004	0x00000014	lui r1, 0xffff0000	99	test_2: li r1, 0xffffffff00000000
0x040008	0x00000015	addi r1, r1, 0	100	movi r14, r1, 0
0x04000c	0x00000016	lui r1, 0xffff0000	101	li r1, 0xffffffff00000000
0x040010	0x00000017	addi r1, r1, 0	102	li r1, 0
0x040014	0x00000018	addi r1, r1, 0	103	li r1, 0
0x040018	0x00000019	addi r1, r1, 0	104	li r1, 0
0x04001c	0x0000001a	addi r1, r1, 0	105	li r1, 0
0x040020	0x0000001b	addi r1, r1, 0	106	li r1, 0
0x040024	0x0000001c	addi r1, r1, 0	107	li r1, 0
0x040028	0x0000001d	addi r1, r1, 0	108	li r1, 0
0x04002c	0x0000001e	addi r1, r1, 0	109	li r1, 0
0x040030	0x0000001f	addi r1, r1, 0	110	li r1, 0
0x040034	0x00000020	addi r1, r1, 0	111	li r1, 0

Data segment panel

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
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100101e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers panel

Registers	Floating Point	Control and Status
rsr		
r1		
r2		
r3		
r4		
r5		
r6		
r7		
r8		
r9		
r10		
r11		
r12		
r13		
r14		
r15		
r16		
r17		
r18		
r19		
r20		
r21		
r22		
r23		
r24		
r25		
r26		
r27		
r28		
r29		
r30		
r31		

Program information panel

Messages Run IO

Assembly: assembling F:\Research\bin\TA\CEB3420\tools\text.asm

Warning in F:\Research\bin\TA\CEB3420\tools\text.asm line 312 column 2: RARS does not recognize the global directive. Ignored.

Warning in F:\Research\bin\TA\CEB3420\tools\text.asm line 312 column 2: RARS does not recognize the global directive. Ignored.

Assembly: operation completed successfully.

Clear

- Create a new source file: Ctrl + N
- Close the current source file: Ctrl + W
- Assemble the source code: F3
- Execute the current source code: F5
- Step running: F7
- Instructions & System call query: F1

System Service in RARS

RARS provides a small set of operating system-like services through the system call (`ecall`) instruction. Register contents are not affected by a system call, except for result registers in some instructions.

- Load the service number (or number) in register `a7`.
- Load argument values, if any, in `a0`, `a1`, `a2` ..., as specified.
- Issue `ecall` instruction.
- Retrieve return values, if any, from result registers as specified.

System Calls in RARS II

Name	Number	Description	Inputs	Outputs
PrintInt	1	Prints an integer	a0 = integer to print	N/A
PrintFloat	2	Prints a float point number	fa0 = float to print	N/A
PrintString	4	Prints a null-terminated string to the console	a0 = the address of the string	N/A
ReadInt	5	Reads an int from input console	a0 = the int	N/A
ReadFloat	6	Reads a float from input console	fa0 = the float	N/A
ReadString	8	Reads a string from the console	a0 = address of input buffer, a1 = maximum number of characters to read	N/A
Open	1024	Opens a file from a path Only supported flags (a1), read-only (0), write-only (1) and write-append (9)	a0 = Null terminated string for the path, a1 = flags	a0 = the file descriptor or -1 if an error occurred
Read	63	Read from a file descriptor into a buffer	a0 = the file descriptor, a1 = address of the buffer, a2 = maximum length to read	a0 = the length read or -1 if error
Write	64	Write to a file descriptor from a buffer	a0 = the file descriptor, a1 = the buffer address, a2 = the length to write	a0 = the number of characters written
LSeek	62	Seek to a position in a file	a0 = the file descriptor, a1 = the offset for the base, a2 is the beginning of the file (0), the current position (1), or the end of the file (2)}	a0 = the selected position from the beginning of the file or -1 is an error occurred

An Example of System Calls in RARS I

An example shows how to use system calls in RARS

Using system call

```
# Comment giving name of program and description
# sys-call.asm
# Bare-bones outline of RISC-V assembly language program
    .globl _start

    .data
msg: .asciz "Hello, _world!\n"

    .text
_start:
    li a7, 4      # system call code for PrintString
    la a0, msg    # address of string to print
    ecall         # Use the system call
# End of program, leave a blank line afterwards is preferred
```

You can check the output in Run/IO of the program information panel.

An Example of System Calls in RARS II

- *li* loads a register with an immediate value given in the instruction.
- *la* loads an address of the specified symbol.
- *.asciz* emits the specified string within double quotes and includes the terminated zero character at the end.

Lab 1-1 Assignment

Write a RISC-V assembly program step by step as shown below:

- 1 Define two variables `var1` and `var2` which have initial value 15 and 19, respectively. (`var1 = 15` and `var2 = 19`)
- 2 Print MEMORY addresses of `var1` and `var2` using `syscall`.
- 3 Increase `var1` by 1 and multiply `var2` by 4.
- 4 Print `var1` and `var2` again.
- 5 Swap `var1` and `var2` and print them. (`var1` and `var2` are changed)

Submission Method:

Submit the source code and report **after** the whole lectures of Lab1 into **Blackboard**.

Some Tips

- ① Variables should be declared following the `.data` identifier.
- ② `<name>: .<datatype> <value>`
- ③ Use `la` instruction to access the RAM address of declared data.
- ④ Use `system` call to print integers.
- ⑤ Do not forget `exit` system call.
- ⑥ You should print a new line to distinguish outputs!