

CENG3420

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

**Chen BAI**

Department of Computer Science and Engineering  
The Chinese University of Hong Kong

`cbai@cse.cuhk.edu.hk`

Spring 2021



香港中文大學

The Chinese University of Hong Kong

# Overview

RARS

Assembly Programing

System Service in RARS

Lab Assignment



# Overview

RARS

Assembly Programing

System Service in RARS

Lab Assignment



# 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 and/or *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/download/continuous/rars\\_345c17b.jar](https://github.com/TheThirdOne/rars/releases/download/continuous/rars_345c17b.jar)

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



# RARS Overview

The screenshot displays the RARS application interface. The main window contains assembly code for a test program. The registers window on the right shows the state of registers saxe through s31. The messages window at the bottom shows the output of the assembly process.

```
# 32 "iss/rv64ui/rliw.S" 2
88
89
90
91 .test
92 .globl _start
93 _start: nop
94
95 -----
96 # Arithmetic tests
97 #-----
98
99 test_0: li s1, 0xffffffff80000000
100 rliw s14, s1, 0
101 li s7, 0xffffffff80000000
102 li sp, 2
103 bne s14, s7, fail
104
105 test_3: li s1, 0xffffffff80000000
106 rliw s14, s1, 1
107 li s7, 0x0000000040000000
108 li sp, 3
109 bne s14, s7, fail
110
111 test_4: li s1, 0xffffffff80000000
```

Registers	Floating Point	Control and Status
Name	Number	Value
saxe	0	0x0000000000000000
s1	1	0x0000000000000000
s2	2	0x00000000ffffffffff
s3	3	0x0000000100000000
s4	4	0x0000000000000000
s5	5	0x0000000000000000
s6	6	0x0000000000000000
s7	7	0x0000000000000000
s8	8	0x0000000000000000
s9	9	0x0000000000000000
s10	10	0x0000000000000000
s11	11	0x0000000000000000
s12	12	0x0000000000000000
s13	13	0x0000000000000000
s14	14	0x0000000000000000
s15	15	0x0000000000000000
s16	16	0x0000000000000000
s17	17	0x0000000000000000
s18	18	0x0000000000000000
s19	19	0x0000000000000000
s20	20	0x0000000000000000
s21	21	0x0000000000000000
s22	22	0x0000000000000000
s23	23	0x0000000000000000
s24	24	0x0000000000000000
s25	25	0x0000000000000000
s26	26	0x0000000000000000
s27	27	0x0000000000000000
s28	28	0x0000000000000000
s29	29	0x0000000000000000
s30	30	0x0000000000000000
s31	31	0x0000000000000000
pc		0x0000000000000000

Line: 100 Column: 10 Show Line Numbers

Messages Run ID

asmbl: assembling F:\Users\h\source\TAVC880420\tools\test.asm

Warning in F:\Users\h\source\TAVC880420\tools\test.asm line 312 column 2: RARS does not recognize the .global directive. Ignored.

Warning in F:\Users\h\source\TAVC880420\tools\test.asm line 318 column 2: RARS does not recognize the .global directive. Ignored.

asmbl: operation completed successfully.

Clear





# RARS Basic introduction

The screenshot displays the RARS application interface, which is divided into several panels:

- Source codes panel:** Contains assembly code for a test program. The code includes comments and instructions for testing various instructions like `li`, `srlw`, and `lsw`.
- Registers panel:** A table showing the state of registers. The registers are listed in two columns: `Registers` and `Control and Status`. The `Registers` column includes `name` and `Number`, and the `Control and Status` column includes `Value`. The registers are numbered from 0 to 31.
- Program information panel:** A messages window at the bottom showing the output of the assembler. It indicates that the assembly operation completed successfully.

The interface also features a menu bar (File, Edit, Run, Settings, Tools, Help), a toolbar, and a status bar at the bottom.

Registers	Floating Point	Control and Status
Name	Number	Value
saxx	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x00000000ffffffffff
gp	3	0x0000000100000000
tp	4	0x0000000000000000
00	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
g0	8	0x0000000000000000
t1	9	0x0000000000000000
a0	10	0x0000000000000000
a1	11	0x0000000000000000
a2	12	0x0000000000000000
a3	13	0x0000000000000000
a4	14	0x0000000000000000
a5	15	0x0000000000000000
a6	16	0x0000000000000000
a7	17	0x0000000000000000
a8	18	0x0000000000000000
a9	19	0x0000000000000000
a10	20	0x0000000000000000
a11	21	0x0000000000000000
a12	22	0x0000000000000000
a13	23	0x0000000000000000
a14	24	0x0000000000000000
a15	25	0x0000000000000000
a16	26	0x0000000000000000
a17	27	0x0000000000000000
a18	28	0x0000000000000000
a19	29	0x0000000000000000
a20	30	0x0000000000000000
a21	31	0x0000000000000000
pc		0x0000000000000000

```
# 12 "src/vv64ui/srlw.S" 2
88
89
90
91 .test
92 _globl _start
93 _start: nop
94
95 -----
96 # Arithmetic tests
97 #-----
98
99 test_0: li x1, 0xffffffff80000000
100 srlw x14, x1, 0
101 li x7, 0xffffffff80000000
102 li sp, 2
103 bne x14, x7, fail
104
105 test_3: li x1, 0xffffffff80000000
106 srlw x14, x1, 1
107 li x7, 0x0000000040000000
108 li sp, 3
109 bne x14, x7, fail
110
111 test_4: li x1, 0xffffffff80000000
```

Line: 100 Column: 10 Show Line Numbers

Messages Run ID

asmcli: assembling F:\assembler\src\VV64UI\tools\test.asm

Warning in F:\assembler\src\VV64UI\tools\test.asm line 88 column 2: RARS does not recognize the global directive. Ignored.

Warning in F:\assembler\src\VV64UI\tools\test.asm line 88 column 2: RARS does not recognize the global directive. Ignored.

asmcli: operation completed successfully.





Tools panel

Edit Execute

**Text Segment**

Flag	Address	Code	Basic	Source
	0x04000000	0x0000001c	addr r0, r0, 0	91_start_nop
	0x04000004	0x00000017	ldr r1, 0xffff0000	90_test_2 li r1, 0xffffffff00000000
	0x04000008	0x00000039	addr r1, r1, 0	
	0x0400000c	0x00000017	ldr r1, r1, r1, 0	100_rliw r14, r1, 0
	0x04000010	0x00000017	ldr r7, 0xffff0000	101 li r7, 0xffffffff00000000
	0x04000014	0x00000039	addr r1, r1, 0	
	0x04000018	0x0000001c	addr r2, r0, 0	102 li r0, 2
	0x0400001c	0x2471e038	ldr r14, r1, 0x00000388	103 ldr r14, r1, #0
	0x04000020	0x00000017	ldr r1, 0xffff0000	105_test_3 li r1, 0xffffffff00000000
	0x04000024	0x00000039	addr r1, r1, 0	
	0x04000028	0x00000017	rliw r14, r1, 1	106_rliw r14, r1, 1
	0x0400002c	0x00000039	addr r1, r1, 0x00040000	107 li r1, 0x0000000400000000
	0x04000030	0x00000039	addr r1, r1, 0	
	0x04000034	0x03001083	addr r3, r0, 3	108 li r0, 3
	0x04000038	0x3271e038	ldr r14, r1, 0x00000388	109 ldr r14, r1, #0
	0x0400003c	0x00000017	ldr r1, 0xffff0000	111_test_4 li r1, 0xffffffff00000000

Text segment panel

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)	Value (+10)	Value (+14)	Value (+18)	Value (+1C)
0x10101000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101004	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101008	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1010100c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101010	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101018	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1010101c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101028	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1010102c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101030	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101034	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101038	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1010103c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10101000 (Data) Hexadecimal Addresses Hexadecimal Values ASCII

Data segment panel

**Messages** Run IO

```

Assembly: assembling F:\Research\misc\TA\CE93420\test\test.asm
Warning in F:\Research\misc\TA\CE93420\test\test.asm line 312 column 2: RARS does not recognize the global directive. Ignored.
Warning in F:\Research\misc\TA\CE93420\test\test.asm line 318 column 2: RARS does not recognize the global directive. Ignored.
Assembly: operation completed successfully.
  
```

Clear

Program information panel

Registers	Floating Point	Control and Status
Name	Number	Value
r0x0	0	0x0000000000000000
r0x1	1	0x0000000000000000
r0x2	2	0x0000000000000000
r0x3	3	0x0000000000000000
r0x4	4	0x0000000000000000
r0x5	5	0x0000000000000000
r0x6	6	0x0000000000000000
r0x7	7	0x0000000000000000
r0x8	8	0x0000000000000000
r0x9	9	0x0000000000000000
r0x10	10	0x0000000000000000
r0x11	11	0x0000000000000000
r0x12	12	0x0000000000000000
r0x13	13	0x0000000000000000
r0x14	14	0x0000000000000000
r0x15	15	0x0000000000000000
r0x16	16	0x0000000000000000
r0x17	17	0x0000000000000000
r0x18	18	0x0000000000000000
r0x19	19	0x0000000000000000
r0x20	20	0x0000000000000000
r0x21	21	0x0000000000000000
r0x22	22	0x0000000000000000
r0x23	23	0x0000000000000000
r0x24	24	0x0000000000000000
r0x25	25	0x0000000000000000
r0x26	26	0x0000000000000000
r0x27	27	0x0000000000000000
r0x28	28	0x0000000000000000
r0x29	29	0x0000000000000000
r0x30	30	0x0000000000000000
r0x31	31	0x0000000000000000
r0x32	32	0x0000000000000000
r0x33	33	0x0000000000000000
r0x34	34	0x0000000000000000
r0x35	35	0x0000000000000000
r0x36	36	0x0000000000000000
r0x37	37	0x0000000000000000
r0x38	38	0x0000000000000000
r0x39	39	0x0000000000000000
r0x40	40	0x0000000000000000
r0x41	41	0x0000000000000000
r0x42	42	0x0000000000000000
r0x43	43	0x0000000000000000
r0x44	44	0x0000000000000000
r0x45	45	0x0000000000000000
r0x46	46	0x0000000000000000
r0x47	47	0x0000000000000000
r0x48	48	0x0000000000000000
r0x49	49	0x0000000000000000
r0x50	50	0x0000000000000000
r0x51	51	0x0000000000000000
r0x52	52	0x0000000000000000
r0x53	53	0x0000000000000000
r0x54	54	0x0000000000000000
r0x55	55	0x0000000000000000
r0x56	56	0x0000000000000000
r0x57	57	0x0000000000000000
r0x58	58	0x0000000000000000
r0x59	59	0x0000000000000000
r0x60	60	0x0000000000000000
r0x61	61	0x0000000000000000
r0x62	62	0x0000000000000000
r0x63	63	0x0000000000000000
r0x64	64	0x0000000000000000
r0x65	65	0x0000000000000000
r0x66	66	0x0000000000000000
r0x67	67	0x0000000000000000
r0x68	68	0x0000000000000000
r0x69	69	0x0000000000000000
r0x70	70	0x0000000000000000
r0x71	71	0x0000000000000000
r0x72	72	0x0000000000000000
r0x73	73	0x0000000000000000
r0x74	74	0x0000000000000000
r0x75	75	0x0000000000000000
r0x76	76	0x0000000000000000
r0x77	77	0x0000000000000000
r0x78	78	0x0000000000000000
r0x79	79	0x0000000000000000
r0x80	80	0x0000000000000000

Registers panel



# Basic introduction

- ▶ 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



# Overview

RARS

Assembly Programing

System Service in RARS

Lab Assignment



# Registers

- ▶ We can manipulate 32 general purpose 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



# Stack Pointer Register

## Stack pointer register

In RISC-V architecture, x2 register is use as Stack Pointer  $sp0$  and holds the base address of the stack.

Stack base address must aligned to 4-bytes. Failing which, a load / store alignment fault may arise.



# Global Pointer Register

## Global pointer register

Data is allocated to the memory when it is globally declared in an application. Using pc-relative or absolute addressing mode leads to utilization of extra instructions, thus increasing the code size.

In order to decrease the code size, RISC-V places all the global variables in a particular area which is pointed to, using the x3 *gp* register. The x3 register will hold the base address of the location where the global variables reside.



# Thread Pointer Register

## Thread pointer register

The x1 *ra* register is used to save the subroutine / function return addresses. Before a subroutine call is performed, x1 is explicitly set to the subroutine return address which is usually  $pc + 4$ .

The standard software calling convention uses x1 register to hold the return address on a function call.



# Argument Register

## Argument register

In RISC-V, 8 argument registers, namely, x10 to x17 are used to pass arguments in a subroutine / function. Before a subroutine call is made, the arguments to the subroutine are copied to the argument registers. The stack is used in case the number of arguments exceeds 8.



# Data Types and Literals

## Data types:

- ▶ Instructions are all 32 bits
- ▶ 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

- ▶ Just plain text file with data declarations, program code (name of file can be suffixed with *.asm* in **RARS**)
- ▶ Data declaration section 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



# Program Structure III

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
```



# 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
```



# An Example Program

File Edit Run Settings Tools Help

Run speed at max (no interaction)

**Edit Execute**

**Text Segment**

Blkpt	Address	Code	Basic	Source
	0x00400000	0x00100513	addi x10,x0,1	9: addi a0, x0, 1
	0x00400004	0x0fc10597	auipc x11,0x0000fc10	11: la a1, welcome_msg
	0x00400008	0xffc58593	addi x11,x11,0xffff...	
	0x0040000c	0x01500613	addi x12,x0,21	13: addi a2, x0, 21
	0x00400010	0x04000893	addi x17,x0,0x00000040	15: addi a7, x0, 64
	0x00400014	0x00000073	ecall	17: ecall

**Data Segment**

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00400000	0x00100513	0x0fc10597	0xffc58593	0x01500613	0x04000893	0x00000073	0x00000000	0x00000000
0x00400020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x004000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x004000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x004000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x00400000 (.text) Hexadecimal Addresses Hexadecimal Values ASCII

**Control and Status**

Registers		Floating Point	
Name	Number		Value
zero	0		0x00000000
ra	1		0x00000000
sp	2		0x7ffffffc
gp	3		0x10000000
tp	4		0x00000000
t0	5		0x00000000
t1	6		0x00000000
t2	7		0x00000000
s0	8		0x00000000
s1	9		0x00000000
a0	10		0x00000015
a1	11		0x10010000
a2	12		0x00000015
a3	13		0x00000000
a4	14		0x00000000
a5	15		0x00000000
a6	16		0x00000000
a7	17		0x00000040
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
t6	31		0x00000000
pc			0x0040001c

**Messages Run I/O**

Welcome to ENG3420!

-- program is finished running (dropped off bottom) --

Clear



# 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 memory[NumElements] to x5
```

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

## Syntax

```
li rd, CONSTANT
```



# Instructions Overview II

## 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
```

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

## Syntax

```
li rd, CONSTANT
```

## Usage



# Instructions Overview IV

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

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

```
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.



# Instructions Overview V

## 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



# Instructions Overview VI

```
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

```
srl rd, rs1, imm
```

## Usage

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



# More Information

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/>



# Overview

RARS

Assembly Programing

System Service in RARS

Lab Assignment



# System Calls in RARS I

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 filedescriptor from a buffer	a0 = the file descriptor, a1 = the buffer address, a2 = the length to write	a0 = the number of charcters 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



# Overview

RARS

Assembly Programing

System Service in RARS

Lab Assignment



# Lab Assignment

Write an assembly program with the following requirements:

1. Define two variables `var1` and `var2` which have initial value 15 and 19, respectively.
2. Print RAM addresses of `var1` and `var2` using `syscall`.
3. Increase `var1` by 1 and multiply `var2` by 4.
4. Print `var1` and `var2`.
5. Swap `var1` and `var2` and print them.

## Submission Method:

Submit the source code and report **after** the whole Lab1, onto [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!

