



香港中文大學
The Chinese University of Hong Kong

CENG3420

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

Chen BAI

Department of Computer Science & Engineering

Chinese University of Hong Kong

cbai@cse.cuhk.edu.hk

Spring 2022

- 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

- 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

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>`

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
```

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

Bkpt	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

0x00400000 (Text) Hexadecimal Addresses Hexadecimal Values ASCII

Control and Status

Registers

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
s0	10	0x00000015
s1	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 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).

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/download/continuous/rars_f0c874c.jar

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

RARS Overview

The screenshot displays the RARS application interface. The main window shows assembly code for a test program. The code includes directives like #12, #Arithmetic tests, and instructions like li, ori, and hwi. The right panel shows a register table with columns for Name, Number, and Value. The bottom panel shows messages from the assembler, including warnings about unrecognized global directives and a successful completion message.

```
test.asm
88 # 12 "isa/rv64ui/srliw.5" 2
89
90
91 .test
92 .globl _start
93 _start: nop
94
95 #-----
96 # Arithmetic tests
97 #-----
98
99 test_2: li s1, 0xffffffff80000000
100 srliw s14, s1, 0
101 li s7, 0xffffffff80000000
102 li s0, 2
103 hwi s14, s7, fail
104
105 test_3: li s1, 0xffffffff80000000
106 srliw s14, s1, 1
107 li s7, 0x0000000040000000
108 li s0, 3
109 hwi s14, s7, fail
110
111 test_4: li s1, 0xffffffff80000000
112 li
```

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0	0x0000000000000000	
ra	1	0x0000000000000000	
sp	2	0x00000007ffffffffff	
gp	3	0x0000000000000000	
tp	4	0x0000000000000000	
cp	5	0x0000000000000000	
t0	6	0x0000000000000000	
t1	7	0x0000000000000000	
t2	8	0x0000000000000000	
t3	9	0x0000000000000000	
t4	10	0x0000000000000000	
t5	11	0x0000000000000000	
t6	12	0x0000000000000000	
t7	13	0x0000000000000000	
t8	14	0x0000000000000000	
t9	15	0x0000000000000000	
s0	16	0x0000000000000000	
s1	17	0x0000000000000000	
s2	18	0x0000000000000000	
s3	19	0x0000000000000000	
s4	20	0x0000000000000000	
s5	21	0x0000000000000000	
s6	22	0x0000000000000000	
s7	23	0x0000000000000000	
s8	24	0x0000000000000000	
s9	25	0x0000000000000000	
s10	26	0x0000000000000000	
s11	27	0x0000000000000000	
s12	28	0x0000000000000000	
s13	29	0x0000000000000000	
s14	30	0x0000000000000000	
s15	31	0x0000000000000000	
pc		0x0000000000000000	

Messages Run IO

```
Assemble: assembling F:\Research\misc\TA\CEB32420\tools\test.asm
Parsing in F:\Research\misc\TA\CEB32420\tools\test.asm line 212 column 2: RARS does not recognize the global directive: Ignored.
Parsing in F:\Research\misc\TA\CEB32420\tools\test.asm line 218 column 2: RARS does not recognize the global directive: Ignored.
Assemble: operation completed successfully.
```

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Execute

Test Segment

Bit	Address	Code	Basic	Source
0x040000	0x00000013	add r0, r0, 0	91	_start: nup
0x040004	0x00000015	lsw st, 0xffff0000	99	test_2: li st, 0xffffffff00000000
0x040008	0x00000019	add st, st, 0	100	
0x04000c	0x0000001b	swi st, st, 0	101	li st, 0xffffffff00000000
0x040010	0x0000001d	lsw st, st, 0	102	li gp, 2
0x040014	0x0000001f	add st, st, 0	103	lsw st4, st, fail
0x040018	0x00000021	lsw st, st, 0xffff0000	105	test_3: li st, 0xffffffff00000000
0x04001c	0x00000023	add st, st, 0	106	swi st4, st, 1
0x040020	0x00000025	lsw st, st, 0xffff0000	107	li st, 0x0000000400000000
0x040024	0x00000027	add st, st, 0	108	li gp, 3
0x040028	0x00000029	lsw st4, st, fail	109	lsw st4, st, fail
0x04002c	0x0000002b	lsw st, st, 0xffff0000	111	test_4: li st, 0xffffffff00000000

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x101000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101004	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101008	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10100c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101010	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101018	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101028	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10102c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101030	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101034	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101038	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10103c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers Floating Point Control and Status

Name	Number	Value
r0	0	0x0000000000000000
r1	1	0x0000000000000000
r2	2	0x00000000ffff0000
r3	3	0x0000000100000000
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		0x0000000400000000

Messages Run ID

```

Assemble: assembling F:\Research\msvc\IA62\test\test.asm
Parsing in F:\Research\msvc\IA62\test\test.asm line 312 column 2: RARS does not recognize the global directive. Ignored.
Parsing in F:\Research\msvc\IA62\test\test.asm line 312 column 2: RARS does not recognize the global directive. Ignored.
Assemble: operation completed successfully.
  
```

Clear

RARS Basic Introduction

The screenshot displays the RARS application window with three main panels:

- Tools panel:** Located at the top right, it contains a toolbar with various icons and a status indicator that reads "Run speed at max (no interaction)".
- Source codes panel:** The central area shows the assembly source code for a test program. The code includes directives like `.text`, `.globl _start`, and `_start: nop`, followed by arithmetic tests using `li`, `and`, and `and` instructions. Comments indicate sections for arithmetic tests.
- Registers panel:** Located on the right side, it displays a table of registers. The table has columns for Name, Number, and Value. The registers listed are `zero` through `pc`, with their corresponding values shown in hexadecimal.
- Program information panel:** Located at the bottom, it shows messages from the assembler. The messages indicate that the assembler is running in a specific directory and that it does not recognize the `.globl` directive, but the operation completed successfully.

```
test.asm
88 # 32 "isa/rv64ui/rvliw.5"
89
90
91 .text
92 .globl _start
93 _start: nop
94
95 #-----
96 # Arithmetic tests
97 #-----
98
99 test_2: li s1, 0xffffffff80000000
100 andiw s14, s1, 0
101 li s7, 0xffffffff80000000
102 li s0, 2
103 and s14, s7, fail
104
105 test_3: li s1, 0xffffffff80000000
106 andiw s14, s1, 1
107 li s7, 0x0000000040000000
108 li s0, 3
109 and s14, s7, fail
110
111 test_4: li s1, 0xffffffff80000000
112 andiw s14, s1, 1
```

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x00000007ffffffffff
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
a0	25	0x0000000000000000
a1	26	0x0000000000000000
a2	27	0x0000000000000000
a3	28	0x0000000000000000
a4	29	0x0000000000000000
a5	30	0x0000000000000000
a6	31	0x0000000000000000
pc		0x0000000040000000

Messages

```
Assembly: assembling F:\Research\misc\TA\CEB32420\tools\test.asm
Parsing in F:\Research\misc\TA\CEB32420\tools\test.asm line 312 column 2: RARS does not recognize the .globl directive. Ignored.
Parsing in F:\Research\misc\TA\CEB32420\tools\test.asm line 318 column 2: RARS does not recognize the .globl directive. Ignored.
Assembly: operation completed successfully.
```

Edit Execute

Test Segment

Bit#	Address	Code	Basic	Source
0x040000	0x00000013	addi r0, r0, 0	91	_start.nop
0x040004	0x00000015	lui r1, 0xffff0000	99	test_2: li r1, 0xffffffff00000000
0x040008	0x00000019	addi r0, r0, 0	100	
0x04000c	0x0000001b	orli r14, r14, 0	100	li r1, 0xffffffff00000000
0x040010	0x00000019	lui r1, 0xffff0000	101	li r1, 0xffffffff00000000
0x040014	0x00000019	addi r1, r1, 0	102	li gp, 2
0x040018	0x00000019	addi r1, r1, 0	103	lui r14, r1, f0
0x04001c	0x00000015	lui r1, 0xffff0000	105	test_3: li r1, 0xffffffff00000000
0x040020	0x00000019	addi r1, r1, 0	106	orli r14, r1, 1
0x040024	0x0000001b	orli r14, r14, r1, 1	106	orli r14, r1, 1
0x040028	0x00000013	lui r1, 0xffff0000	107	li r1, 0xffffffff00000000
0x04002c	0x00000019	addi r1, r1, 0	108	li gp, 3
0x040030	0x00000015	lui r1, 0xffff0000	108	lui r14, r1, f0
0x040034	0x00000015	lui 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)
0x101000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101004	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101008	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10100c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101010	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101018	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10101c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101028	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10102c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101030	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101034	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x101038	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10103c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Data segment panel

Registers Floating Point Control and Status

Name	Number	Value
pc	0	0x0000000000000000
r4	1	0x0000000000000000
fp	2	0x00000000ffff0000
gp	3	0x0000000000000000
r0	4	0x0000000000000000
r1	5	0x0000000000000000
r2	6	0x0000000000000000
r3	7	0x0000000000000000
r0	8	0x0000000000000000
r1	9	0x0000000000000000
r2	10	0x0000000000000000
r3	11	0x0000000000000000
r4	12	0x0000000000000000
r5	13	0x0000000000000000
r6	14	0x0000000000000000
r7	15	0x0000000000000000
r8	16	0x0000000000000000
r9	17	0x0000000000000000
r0	18	0x0000000000000000
r1	19	0x0000000000000000
r2	20	0x0000000000000000
r3	21	0x0000000000000000
r4	22	0x0000000000000000
r5	23	0x0000000000000000
r6	24	0x0000000000000000
r7	25	0x0000000000000000
r8	26	0x0000000000000000
r9	27	0x0000000000000000
r0	28	0x0000000000000000
r1	29	0x0000000000000000
r2	30	0x0000000000000000
r3	31	0x0000000000000000
pc		0x0000000000000000

Registers panel

Messages Run ID

Assembly: assembling F:\Research\misc\IACER93420\tools\text.asm

Warning in F:\Research\misc\IACER93420\tools\text.asm line 312 column 2: RARS does not recognize the global directive. Ignored.

Warning in F:\Research\misc\IACER93420\tools\text.asm line 312 column 2: RARS does not recognize the global directive. Ignored.

Assembly: operation completed successfully.

Program information panel

- 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 decriptor 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.

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**.

- 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!