CENG3420
Lab 1-1: RISC-V Assembly Language Programing

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

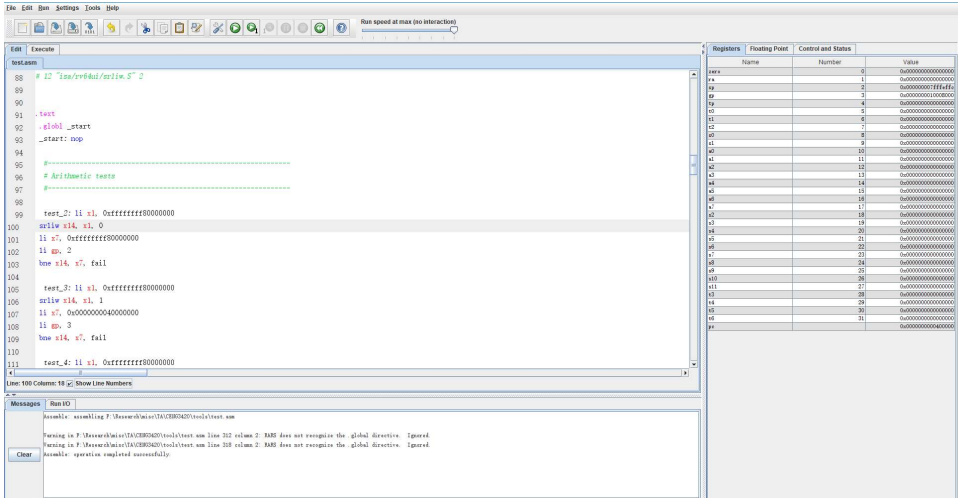
Lab Assignment

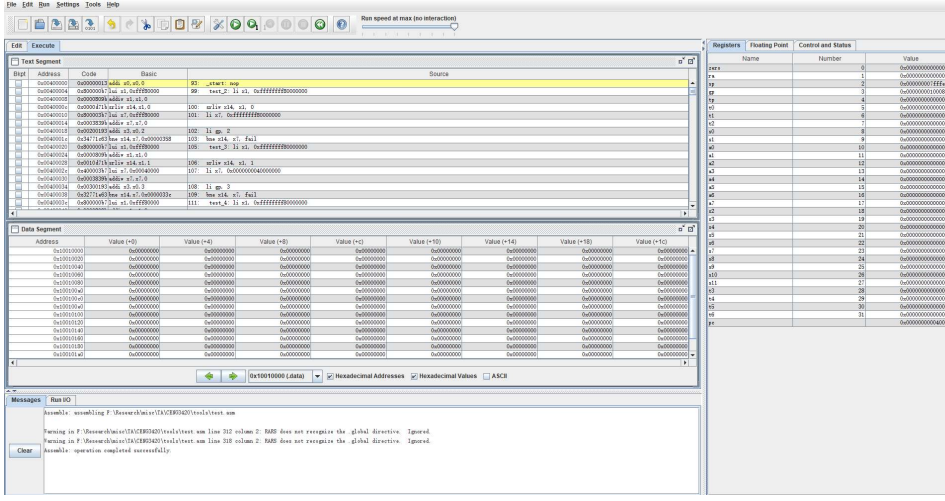
What is RARS

- **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**

Dowload it here: `https://github.com/TheThirdOne/rars/releases/download/continuous/rars_345c17b.jar`
Execute the command to start RARS: java -jar <rars jar path>

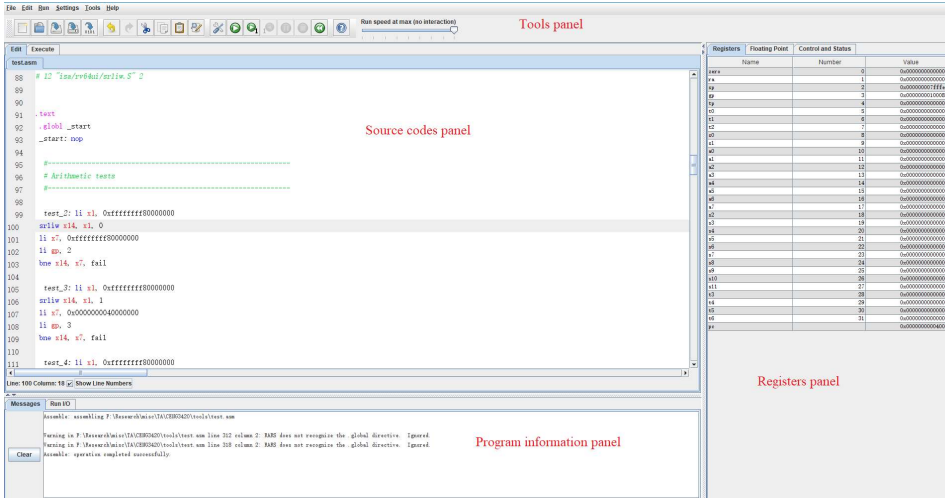
RARS Overview

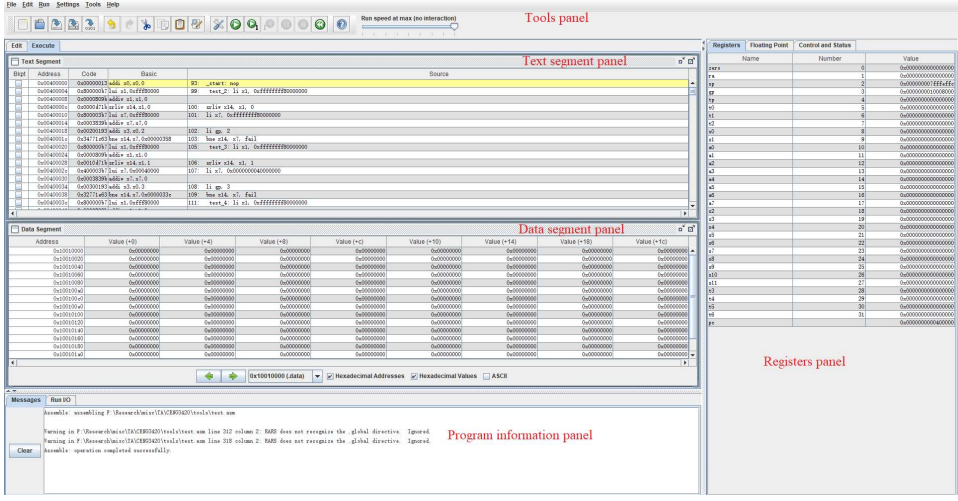




RARS execution panel

RARS Basic introduction





RARS execution 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

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

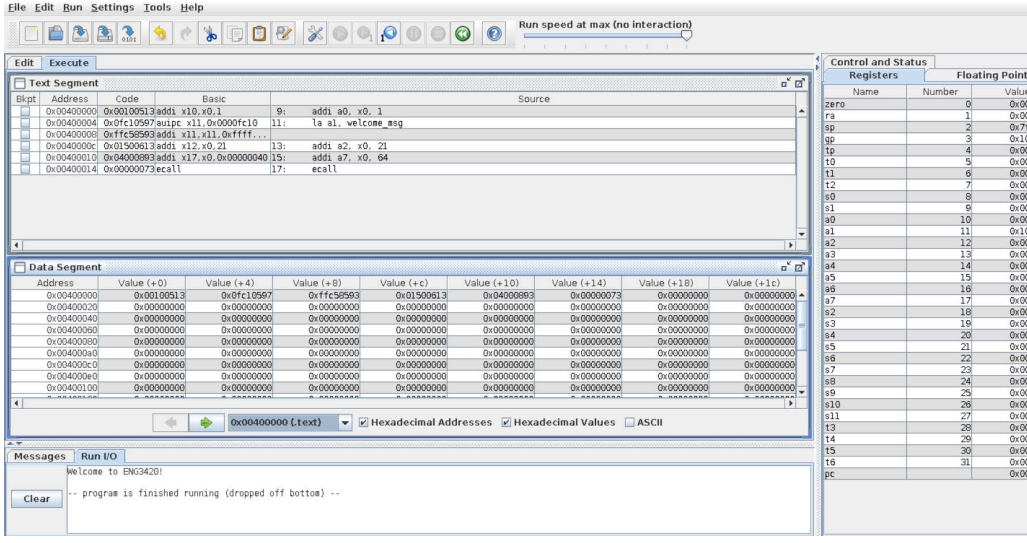
```
.globl _start

.data
welcome_msg: .asciz "Welcome to ENG3420!\n"

.text
_start:
    # STDOUT = 1
    addi a0, x0, 1
    # Load the address of `welcome_msg`
    la a1, welcome_msg
    # length of the string
    addi a2, x0, 21
    # Linux write system call
    addi a7, x0, 64
    # Call linux service to output the string
    ecall

```

An Example Program



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 immeidate value given int 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 immeidate value given int 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 amountheld 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

srli rd, rs1, imm

Usage

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

More Information

For more information about RISC-V instructions and assembly programing 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

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 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 begining 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!