香 港 中 文 大 學
The Chinese University of Hong Kong

CENG3420

Lab 1-1: RISC-V Assembly Language Programing 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

Outline

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

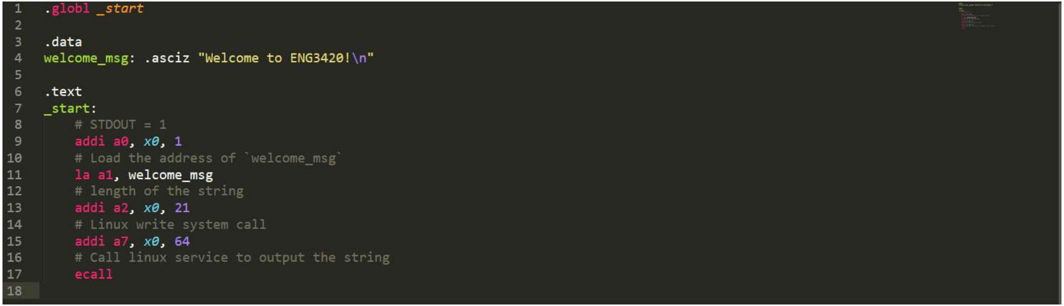
Introduction to Basic RISC-V Assembly Programing

Important Materials

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

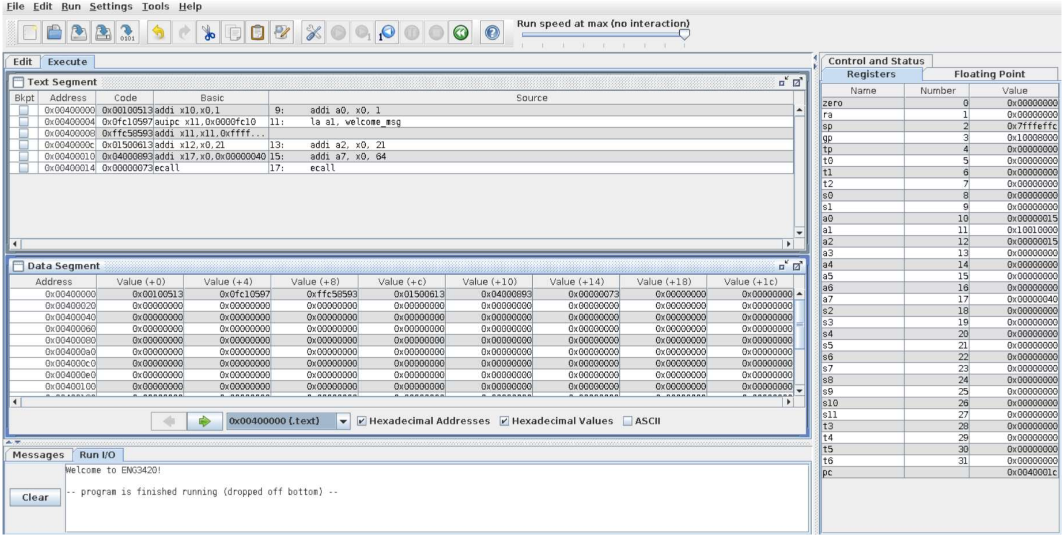
In all labs. of CENG3420, we focus on RV32I instructions.

An Example Program



```
.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



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

```
Welcome to ENG3420!

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

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
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

Registers

- 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 and Literals

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

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
```

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 immeidate 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
```

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

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
```

Instructions Overview V

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

Instructions Overview VI

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/`

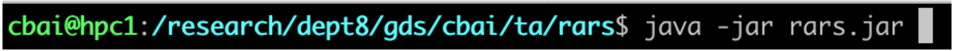
RISC-V ISA Simulator – RARS

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**

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

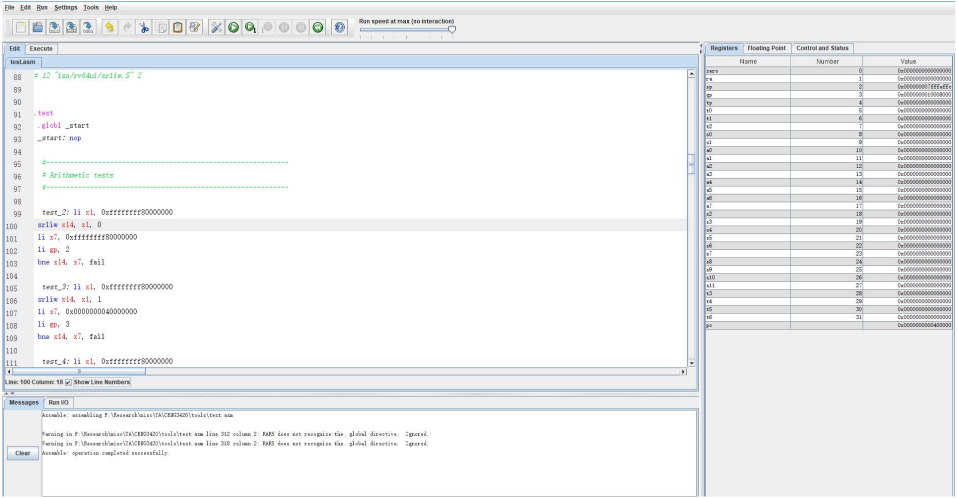
Launch RARS



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

Launch RARS

RARS Overview

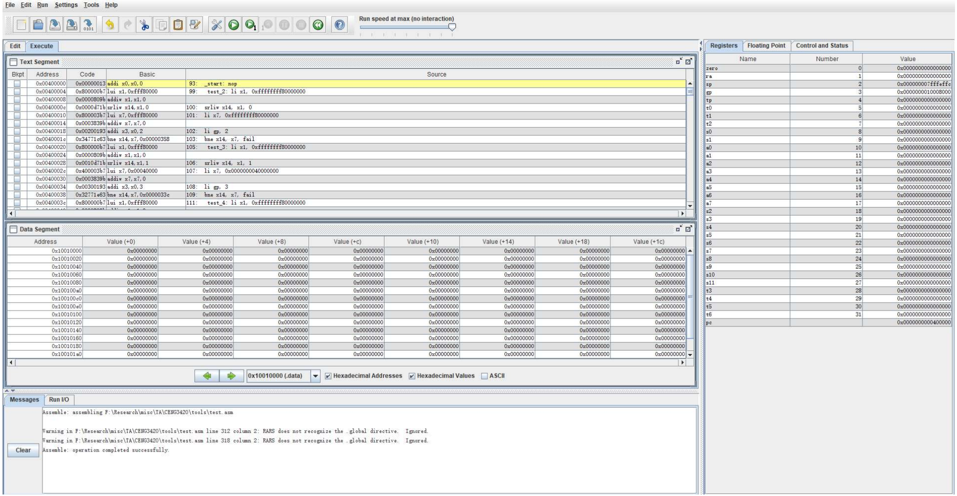


```
# 12 "isa/rv64ui/srliw.S" 2


.text
 .globl _start
 _start: nop

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

  test_2: li x1, 0xffffffff80000000
 srliw x14, x1, 0
 li x7, 0xffffffff80000000
 li gp, 2
 bne x14, x7, fail

  test_3: li x1, 0xffffffff80000000
 srliw x14, x1, 1
 li x7, 0x0000000040000000
 li gp, 3
 bne x14, x7, fail

  test_4: li x1, 0xffffffff80000000
```

Line: 100 Column: 18 ☑ Show Line Numbers

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007fffeffc
gp	3	0x0000000010008000
tp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
s0	8	0x0000000000000000
s1	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
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
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000000400000

```
Assemble: assembling F:\Research\misc\TA\CENG3420\tools\test.asm

Warning in F:\Research\misc\TA\CENG3420\tools\test.asm line 312 column 2: RARS does not recognize the .global directive.  Ignored.
Warning in F:\Research\misc\TA\CENG3420\tools\test.asm line 318 column 2: RARS does not recognize the .global directive.  Ignored.
Assemble: operation completed successfully.
```



Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x00000013	addi x0, x0, 0	93: _start: nop
□	0x00400004	0x800000b7	lui x1, 0xfff80000	99: test_2: li x1, 0xffffffff80000000
□	0x00400008	0x0000809b	addiw x1, x1, 0	
□	0x0040000c	0x0000d71b	srliw x14, x1, 0	100: srliw x14, x1, 0
□	0x00400010	0x800003b7	lui x7, 0xfff80000	101: li x7, 0xffffffff80000000
□	0x00400014	0x0003839b	addiw x7, x7, 0	
□	0x00400018	0x00200193	addi x3, x0, 2	102: li gp, 2
□	0x0040001c	0x34771c63	bne x14, x7, 0x00000358	103: bne x14, x7, fail
□	0x00400020	0x800000b7	lui x1, 0xfff80000	105: test_3: li x1, 0xffffffff80000000
□	0x00400024	0x0000809b	addiw x1, x1, 0	
□	0x00400028	0x0010d71b	srliw x14, x1, 1	106: srliw x14, x1, 1
□	0x0040002c	0x400003b7	lui x7, 0x00040000	107: li x7, 0x0000000040000000
□	0x00400030	0x0003839b	addiw x7, x7, 0	
□	0x00400034	0x00300193	addi x3, x0, 3	108: li gp, 3
□	0x00400038	0x32771e63	bne x14, x7, 0x0000033c	109: bne x14, x7, fail
□	0x0040003c	0x800000b7	lui x1, 0xfff80000	111: test_4: li x1, 0xffffffff80000000

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
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

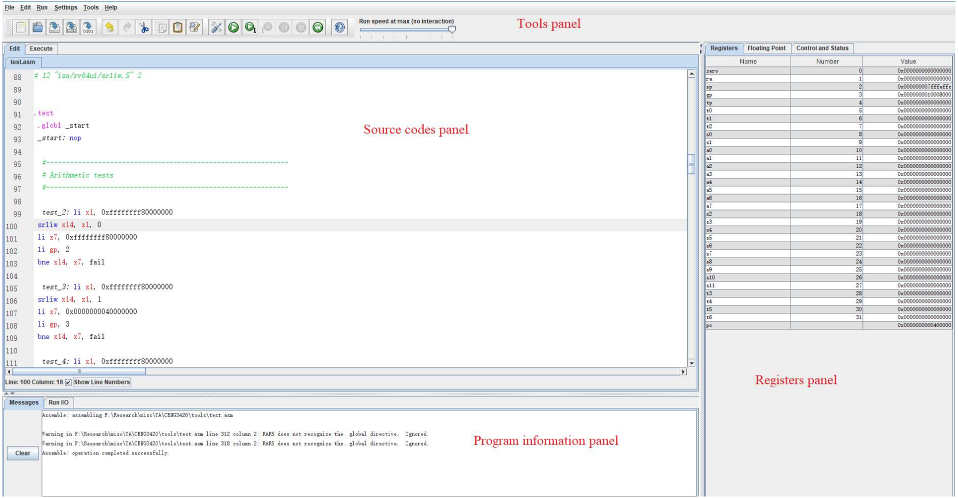
0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007fffeffc
gp	3	0x0000000010008000
tp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
s0	8	0x0000000000000000
s1	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
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
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000000400000

```
Assemble: assembling F:\Research\misc\TA\CENG3420\tools\test.asm

Warning in F:\Research\misc\TA\CENG3420\tools\test.asm line 312 column 2: RARS does not recognize the .global directive.  Ignored.
Warning in F:\Research\misc\TA\CENG3420\tools\test.asm line 318 column 2: RARS does not recognize the .global directive.  Ignored.
Assemble: operation completed successfully.
```

RARS execution panel

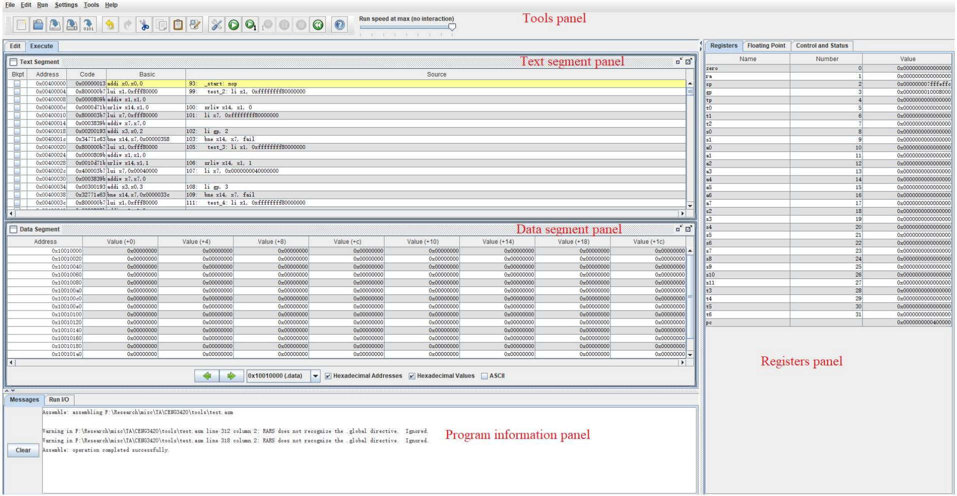
RARS Basic Introduction


File Edit Run Settings Tools Help
Run speed at max (no interaction)
Tools panel
Edit
Execute
test.asm
Source codes panel
Line: 100 Column: 18 ☑ Show Line Numbers
Messages
Run I/O
Clear
Program information panel
Registers
Floating Point
Control and Status
Registers panel

```
# 12 "isa/rv64ui/srliw.S" 2


.text
 .globl _start
 _start: nop

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

  test_2: li x1, 0xffffffff80000000
 srliw x14, x1, 0
 li x7, 0xffffffff80000000
 li gp, 2
 bne x14, x7, fail

  test_3: li x1, 0xffffffff80000000
 srliw x14, x1, 1
 li x7, 0x0000000040000000
 li gp, 3
 bne x14, x7, fail

  test_4: li x1, 0xffffffff80000000
```

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007fffeffc
gp	3	0x0000000010008000
tp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
s0	8	0x0000000000000000
s1	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
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
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000000400000

```
Assemble: assembling F:\Research\misc\TA\CENG3420\tools\test.asm

Warning in F:\Research\misc\TA\CENG3420\tools\test.asm line 312 column 2: RARS does not recognize the .global directive.  Ignored.
Warning in F:\Research\misc\TA\CENG3420\tools\test.asm line 318 column 2: RARS does not recognize the .global directive.  Ignored.
Assemble: operation completed successfully.
```



Run speed at max (no interaction)

Text Segment

Bkpt	Address	Code	Basic	Source
□	0x00400000	0x00000013	addi x0, x0, 0	93: _start: nop
□	0x00400004	0x800000b7	lui x1, 0xfff80000	99: test_2: li x1, 0xffffffff80000000
□	0x00400008	0x0000809b	addiw x1, x1, 0	
□	0x0040000c	0x0000d71b	srliw x14, x1, 0	100: srliw x14, x1, 0
□	0x00400010	0x800003b7	lui x7, 0xfff80000	101: li x7, 0xffffffff80000000
□	0x00400014	0x0003839b	addiw x7, x7, 0	
□	0x00400018	0x00200193	addi x3, x0, 2	102: li gp, 2
□	0x0040001c	0x34771c63	bne x14, x7, 0x00000358	103: bne x14, x7, fail
□	0x00400020	0x800000b7	lui x1, 0xfff80000	105: test_3: li x1, 0xffffffff80000000
□	0x00400024	0x0000809b	addiw x1, x1, 0	
□	0x00400028	0x0010d71b	srliw x14, x1, 1	106: srliw x14, x1, 1
□	0x0040002c	0x400003b7	lui x7, 0x00040000	107: li x7, 0x0000000040000000
□	0x00400030	0x0003839b	addiw x7, x7, 0	
□	0x00400034	0x00300193	addi x3, x0, 3	108: li gp, 3
□	0x00400038	0x32771e63	bne x14, x7, 0x0000033c	109: bne x14, x7, fail
□	0x0040003c	0x800000b7	lui x1, 0xfff80000	111: test_4: li x1, 0xffffffff80000000

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
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

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values □ ASCII

Name	Number	Value
zero	0	0x0000000000000000
ra	1	0x0000000000000000
sp	2	0x000000007fffeffc
gp	3	0x0000000010008000
tp	4	0x0000000000000000
t0	5	0x0000000000000000
t1	6	0x0000000000000000
t2	7	0x0000000000000000
s0	8	0x0000000000000000
s1	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
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
t3	28	0x0000000000000000
t4	29	0x0000000000000000
t5	30	0x0000000000000000
t6	31	0x0000000000000000
pc		0x0000000000400000

```
Assemble: assembling F:\Research\misc\TA\CENG3420\tools\test.asm

Warning in F:\Research\misc\TA\CENG3420\tools\test.asm line 312 column 2: RARS does not recognize the .global directive.  Ignored.
Warning in F:\Research\misc\TA\CENG3420\tools\test.asm line 318 column 2: RARS does not recognize the .global directive.  Ignored.
Assemble: operation completed successfully.
```

RARS execution panel

Shortcuts in Windows

- 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

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.

Lab 1-1 Assignment

Lab 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!