RISCV Assembly Programming

Contents

Ripes simulator

Introduction:

- Ripes is a graphical RISC-V simulator aimed at teaching and understanding computer architecture and assembly programming.

- Developed by Mads Sig Agerbæk.

Key Features:

- **Graphical Interface**: Easy-to-use graphical interface to visualize pipeline stages and register values.

- **Support for RISC-V ISA**: Full support for RV32I and extensions.

- **Interactive Simulation**: Step through execution cycles, view data flow, and track instruction progress.

- **Educational Focus**: Ideal for students learning about computer architecture, assembly language, and the RISC-V ISA.

Setting up Ripes simulator

Download and Installation:

- Visit the Ripes GitHub page to download the latest version.

Link: https://github.com/mortbopet/Ripes/releases

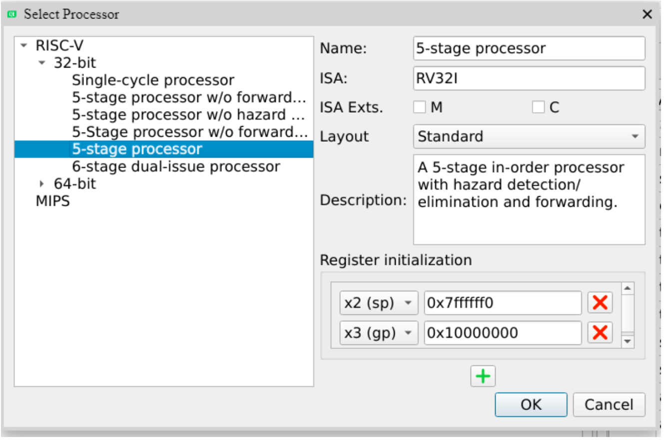
Initial Setup:

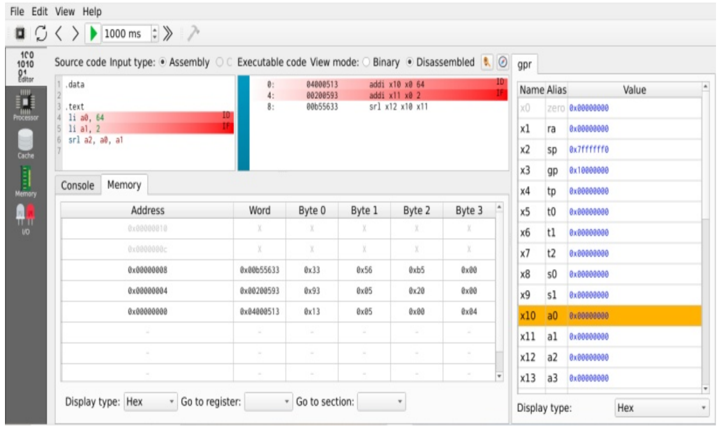
- Launch Ripes and select the RISC-V configuration (e.g., RV32I).
- Familiarize yourself with the main interface: code editor, register view, memory view, and pipeline diagram.

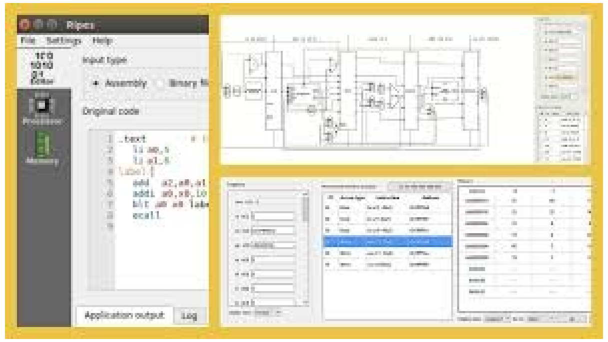
Online Ripes simulator:

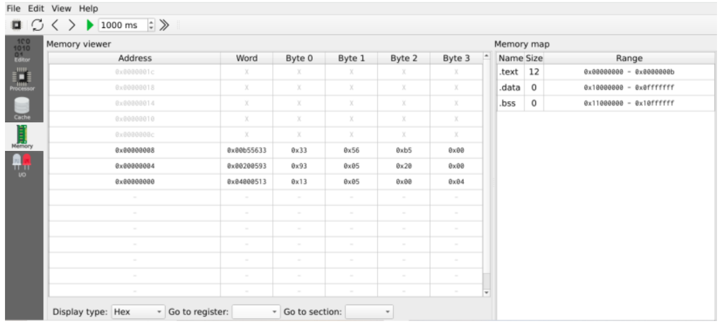
Write ripes.me in the google search bar and it will give you free online access to ripes.

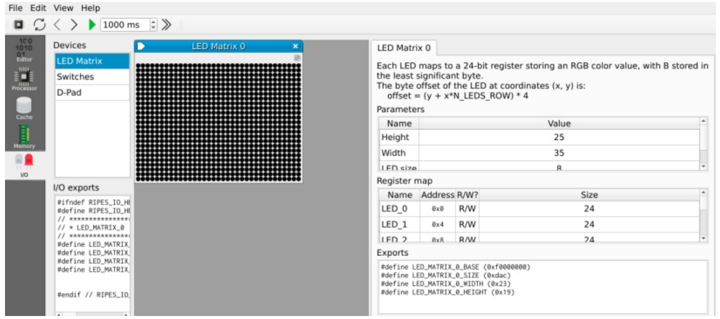
Ripes simulator











Contents

Basics Assembly Structure in Ripes

Creating a New Program:

- Open the code editor and start a new assembly file.
- Define the .data and .text sections.
- Write the entry point with the _start label.

Example:

```
.data
.text
.globl_start
_start:
# Your instructions here
```

Initializing Registers in Ripes

Loading Immediate Values:

- Initialization is the process of assigning a value to the Variable.

- Use the li (Load) instruction to initialize registers with specific values.

Example:

li t0, 10 # Load immediate value 10 into register t0

lui t3, 0x12345 #load 20 bits as a most significant bits in 32 bit register by lui

Declaring Registers in Ripes

Using Registers:

- Declaration tells the compiler about the existence of an entity in the program and its location.
- Understand the different types of registers (temporary t0-t6, saved s0-s11, argument a0-a7).

Example:

```
lw t0, s0 #load s0 register into t0 register
lw t6, s11 #load s11 register into t6 register
```

Contents

Logic Operations on Registers in Ripes

NOT Operation:

li a0, 5

not a1, a0 #(not): Inverts all the bits of the operand

AND Operation:

li a0, 9 li a1, 5

and a2, a1, a0 #(and): Sets each bit to 1 if both corresponding bits are 1.

OR Operation:

li a0, 9 li a1, 5

or a2, a1, a0 #(or): Sets each bit to 1 if at least one corresponding bit is 1

XOR Operation:

li a0, 9 li a1, 5

xor a2, a1, a0 #(xor): Sets each bit to 1 if only one of the corresponding bits is 1.

Contents

Arithmetic Operations in Ripes

Addition:

```
li a0, 10
li a1, 20
add a2, a0, a1 # a2 = a0 + a1
```

Subtraction:

```
li a0, 20
li a1, 10
sub a2, a0, a1 # a2 = a0 - a1
```

Multiplication (using shifts):

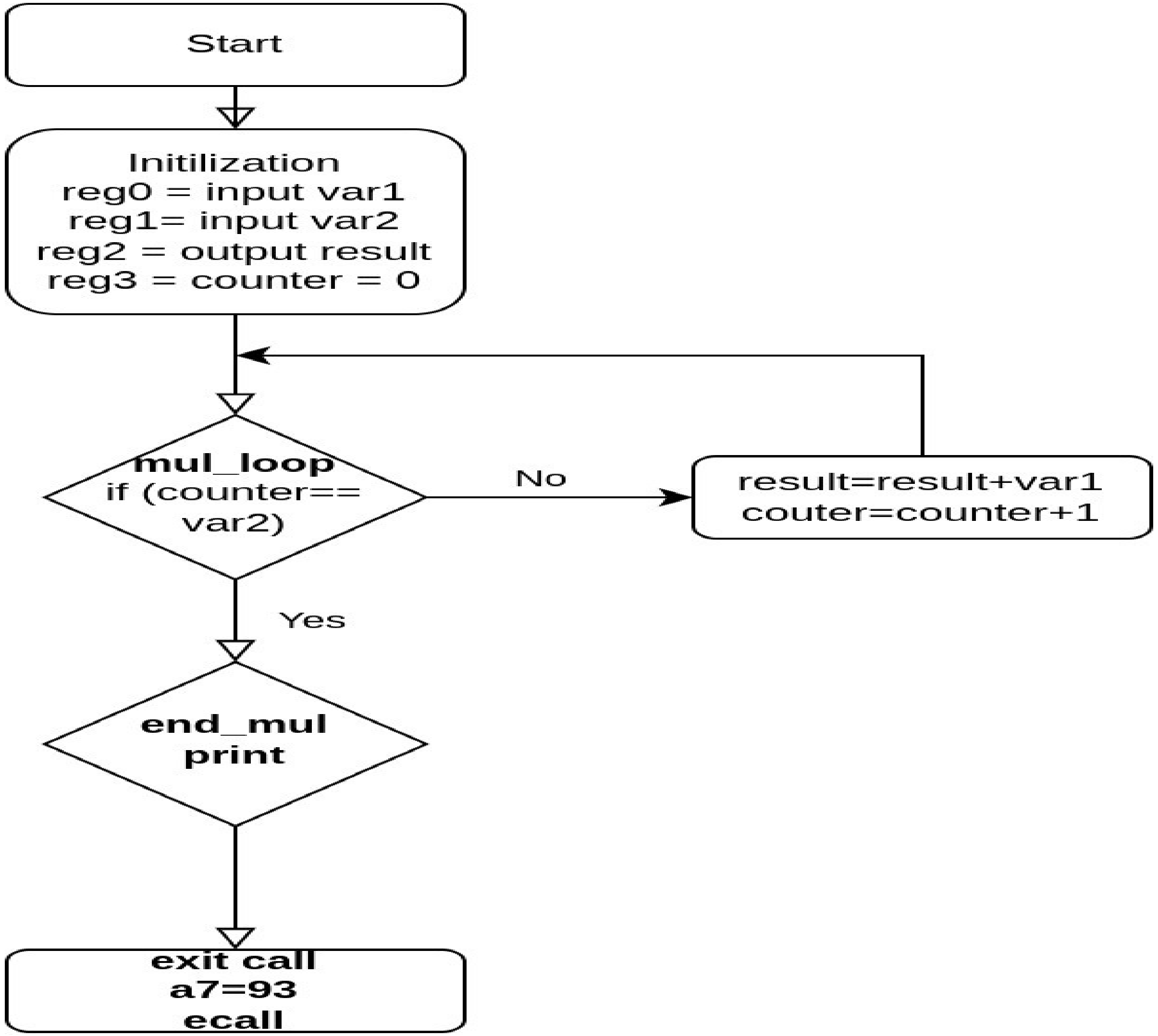
```
li a0, 9
li a1, 1
sll t0, a0, a1
# Logical shift left a0 by a1 positions
(9x2^1=18)
```

Division (using shifts):

```
li a0, 16
li a1,  3
srl t0, a0, a1
# Logical shift right a0 by a1 positions
(16x2^3=2)
```

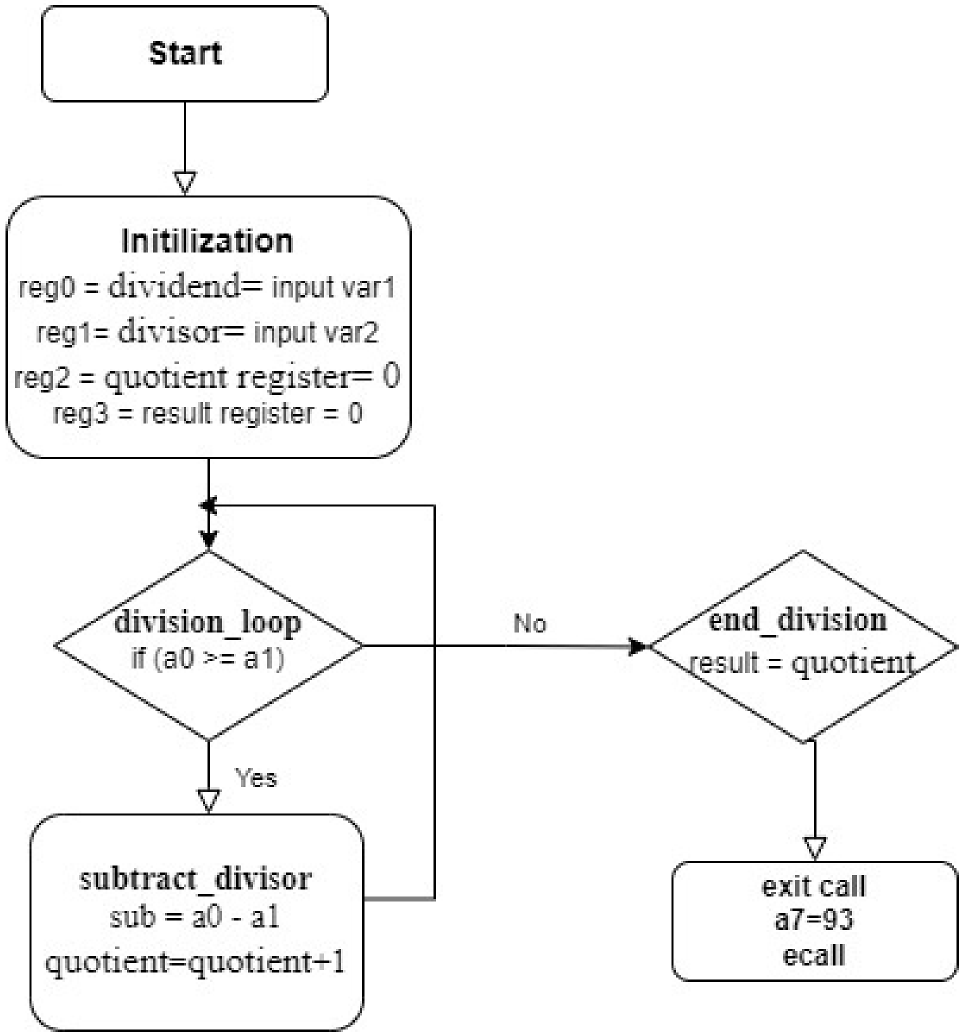
Contents

Multiplication with the Support of Addition (pseudo code)



Contents

Division with the Support of Subtraction (pseudo code)



Contents

Memory Access (Basic Concept)

Accessing memory beyond the registers involves using load and store instructions. This is typically done using lw (load word) and sw (store word).

Example:

```
.data
val1: .word 10
val2: .word 20
.text
.globl _start
 _start:
lw t0, val1 # Load the value at address val1 into t0
lw t1, val2 # Load the value at address val2 into t1
add t2, t0, t1 # Add the values in t0 and t1
sw t2, 0x100(t0) # Store the result at a specific memory address
```

Contents

- Ripes simulator
- Programming
  - Basics Assembly Structure
  - Declaring Registers
  - Initialing Registers
  - Logic Operating on Registers
  - Arithmetic Operations
  - Special Function API (Multiplication with the Support of Addition)
  - Division with the Support of Subtraction
  - Memory Access (Basic Concept)
  - External Peripheral Access (Basic Concept)

External Peripheral Access (Basic Concept)

Interfacing with peripherals (like timers, keyboards, displays) typically involves memory-mapped I/O, where peripheral registers are accessed via specific memory addresses.

Accessing Memory Mapped IO: Interact with peripherals via specific memory addresses.

Example:

```
li t0, 0x10008000 # Address of peripheral
li t1, 1 # Data to write
sw t1, 0(t0) # Write data to peripheral
```