Báo cáo TH KTMT Tuần 2

Nguyễn Quý Đức - 20235682

Assignment 1:

.text

addi s0, zero, 0x512

# sau khi chạy lệnh này, value của **s0** tăng thành

# 0x00000512 và value của thanh ghi **pc** tăng từ

# 0x00400000 thành 0x00400004

add s0, x0, zero

# sau khi chạy lệnh này, value của **s0** quay

# 0x00000000 và value của thanh ghi **pc** là

# 0x00400004

Nếu sửa lại lệnh addi thành “addi s0, zero, 0x20232024” rồi chạy sẽ xuất hiện lỗi “0x20232024 operand is out of range” vì đây là một giá trị 32bit trong khi lệnh **addi** thuộc khuôn dạng lệnh I, sử dụng 12bits để biểu diễn signed int

Assignment 2:

.text

lui s0, 0x20232

# sau khi chạy lệnh này, value **s0** từ

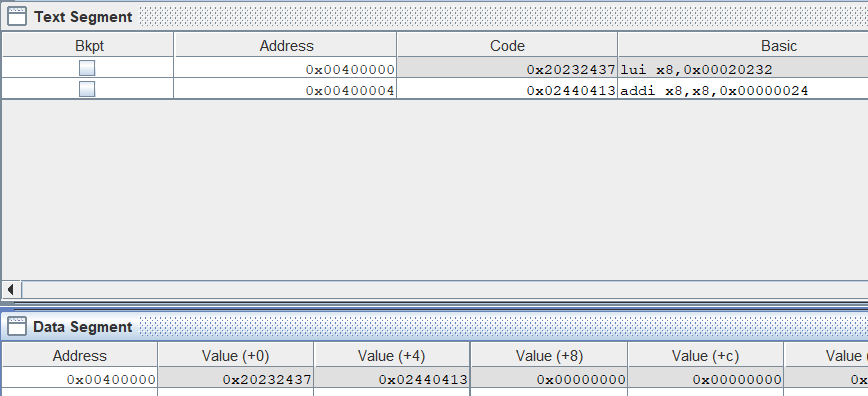
# 0x00000000 thành 0x20232000, **pc**

# từ 0x00400000 thành 0x00400004

addi s0, s0, 0x024

# sau khi chạy lệnh này, value **s0** thành

# 0x20232024, value **pc** thành 0x00400008



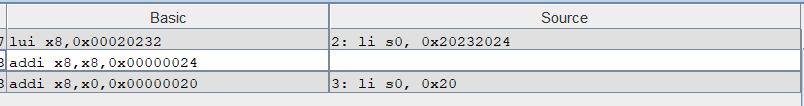
Dữ liệu trong Data Segment và dữ liệu trong Text Segment là giống nhau.

Assignment 3:

.text

li s0, 0x20232024

li s0, 0x20



Cột source hiển thị mã lệnh gốc bằng Assembly:

- li s0, 0x20232024: nạp giá trị 0x20232024 vào thanh ghi - s0 li s0, 0x20: nạp giá trị 0x20 vào thanh ghi s0.

Cột basic hiển thị các lệnh ở dạng mã máy cấp thấp hơn

- li s0, 0x20232024 được tách thành lui x8, 0x00020232 và addi x8, x8, 0x00000024 do đây là lệnh giả nạp giá trị lớn vào thanh ghi

- li s0, 0x20 được dịch thành addi x8, x0, 0x00000020

Assignment 4:

.text

# Assign X, Y into t1, t2 register

addi t1, zero, 5 # X = t1 = ?

addi t2, zero, -1 # Y = t2 = ?

# Expression Z = 2X + Y

add s0, t1, t1 # s0 = t1 + t1 = X + X = 2X

add s0, s0, t2 # s0 = s0 + t2 = 2X + Y

Sau mỗi lệnh, value các thanh ghi thay đổi như sau:

- addi t1, zero, 5:

t1: 0x00000000 -> 0x00000005

- addi t2, zero, -1:

t2: 0x00000000 -> 0xffffffff

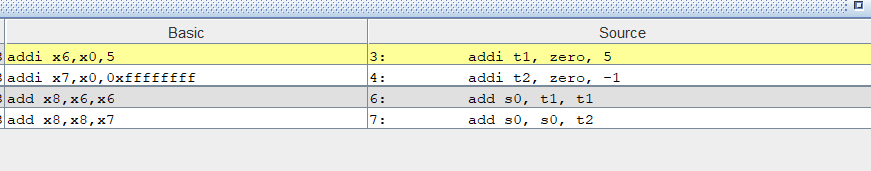
- add s0, t1, t1:

s0: 0x00000000 -> 0x0000000a

- add s0, s0, t2:

s0: 0x0000000a -> 0x00000009

Kết quả đúng 2\*5 + (-1) = 9



- addi x6, x0, 5:

* opcode = 0010011
* rd = 00110 (x6)
* funct3 = 000
* rs1 = 00000 (x0)
* imm = 0000 0000 0101

- addi x7, x0, 0xffffffff:

* imm = 1111 1111 1111

- add x8, x6, x6:

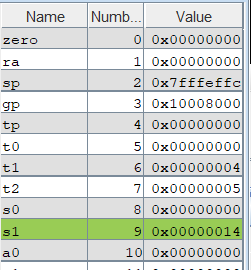
* opcode = 0110011
* rd = 01000 (x8)
* funct3 = 000
* rs1 = 00110 (x6)
* rs2 = 00110 (x6)
* funct7 = 0000000

- add x8, x8, x7:

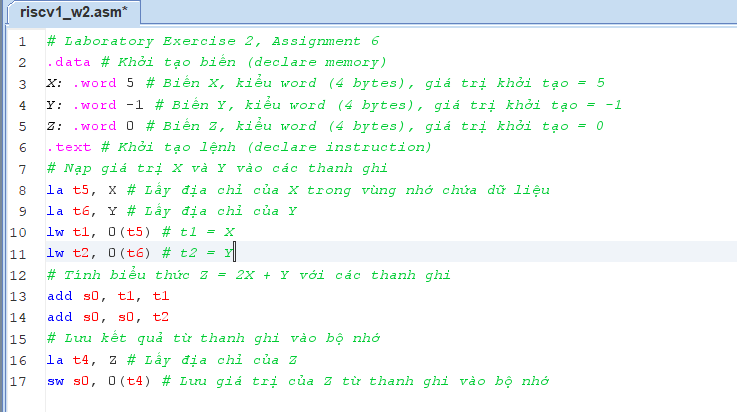
* rs1 = 01000 (x8)
* rs2 = 00111 (x7)

Assignment 5:

Đoạn lệnh trên thực hiện phép nhân hai số X=4 và Y=5. Đầu tiên, giá trị 4 được gán vào thanh ghi t1 và 5 được gán vào t2 bằng lệnh addi (cộng với 0). Sau đó, lệnh mul thực hiện phép nhân t1 với t2 và lưu kết quả 20 (0x14) vào thanh ghi s1.



Assignment 6:



Lêjnh la (load address) trong RISC-V là một lệnh giả (pseudo-instruction) và được dịch thành hai lệnh thực tế auipc và addi, giúp tính toán địa chỉ của biến dựa trên giá trị của Program Counter (PC).

Trong cửa sổ Labels của RARS, ta có thể nhấp đúp vào biến X, Y, Z để xem địa chỉ bộ nhớ tương ứng trong Data Segment, từ đó so sánh với giá trị khởi tạo trong mã nguồn.

Dùng công cụ gỡ lỗi trong RARS để chạy từng lệnh và quan sát sự thay đổi trong cửa sổ Registers, giúp hiểu rõ cách dữ liệu được truyền từ bộ nhớ vào thanh ghi và ngược lại.

Lệnh lw (load word) giúp tải dữ liệu 4 byte từ bộ nhớ vào thanh ghi, trong khi sw (store word) lưu dữ liệu 4 byte từ thanh ghi trở lại bộ nhớ.

Ngoài lw và sw, lệnh lb (load byte) và sb (store byte) chỉ thao tác với 1 byte, hữu ích khi làm việc với dữ liệu dạng char hoặc mảng byte.

Quy trình xử lý chung trong RISC-V bao gồm: nạp giá trị từ bộ nhớ vào thanh ghi (la, lw), thực hiện các phép toán trên thanh ghi, và lưu kết quả trở lại bộ nhớs (sw).