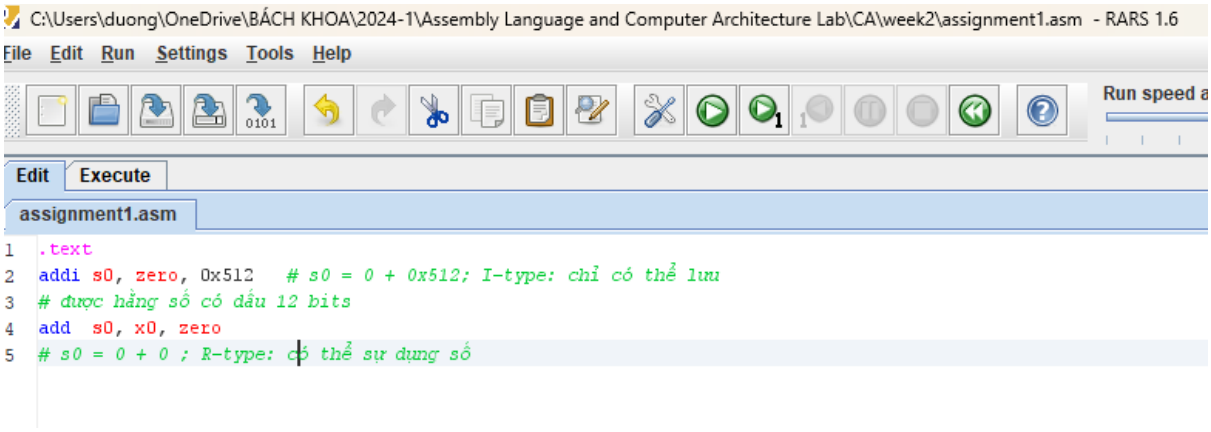
**Báo cáo thực hành Computer Architecture tuần 2**

**Họ và tên: Nguyễn Đình Dương**

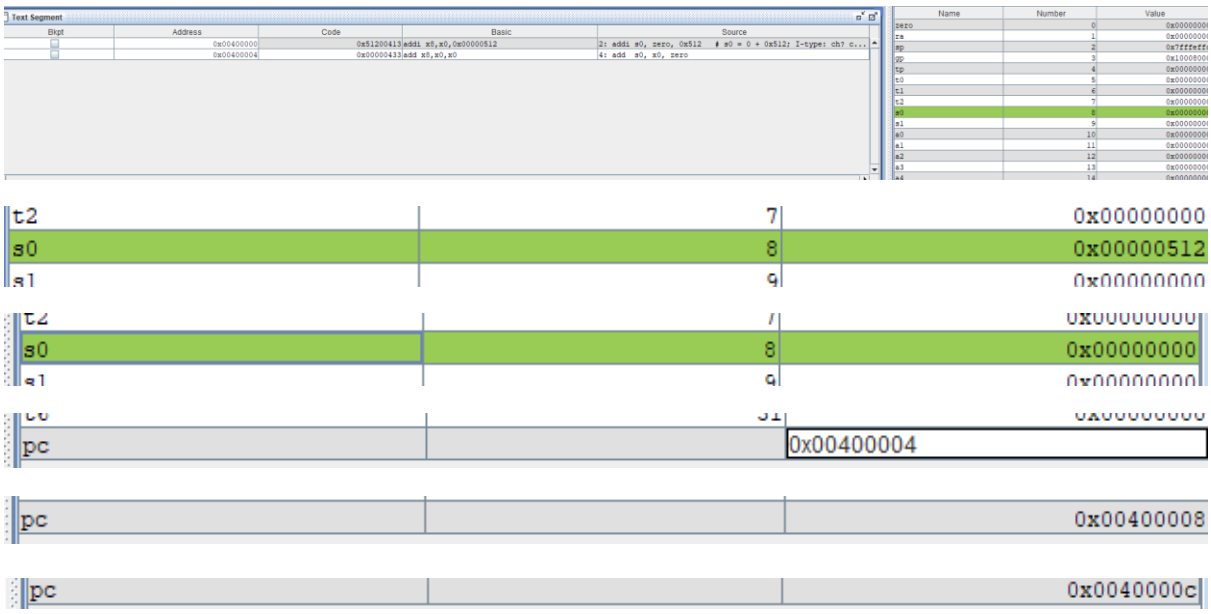
**MSSV: 20225966**

## Assignment 1

Nhập chương trình:




Quan sát các kết quả:




Nhận thấy:

- Sự thay đổi của thanh ghi s0: từ giá trị 0x00000000 chuyển thành 0x00000512 sau lệnh thứ nhất, rồi trở lại giá trị ban đầu sau lệnh thứ hai
- Sự thay đổi của thanh ghi pc: tăng thêm một khoảng có giá trị là 0x00000004 sau mỗi giá trị

Để dịch lệnh từ hợp ngữ sang mã máy, chúng ta sử dụng bảng mã lệnh và các định dạng trong tài liệu "The RISCV Instruction Set Manual".

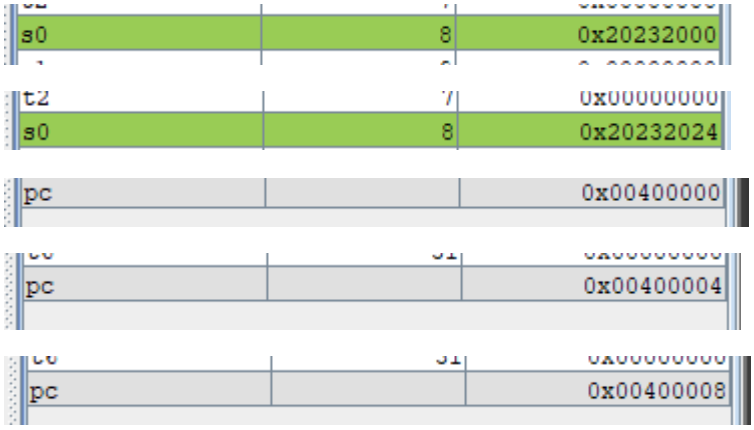
1. **Lệnh 1: addi s0, zero, 0x512:**
    - Khuôn dạng lệnh: I-type (Immediate).
    - opcode: 0010011 (7 bit)
    - rd (s0): 01000 (5 bit)
    - funct3: 000 (3 bit)
    - rs1 (zero): 00000 (5 bit)
    - imm (0x512): 10100010010 (12 bit) (được biểu diễn từ phải sang trái)
    - Mã máy: Kết hợp các thành phần trên lại, ta có mã máy cho lệnh này: **10100010010000000000010000010011**
2. **Lệnh 2: add s0, x0, zero:**
    - Khuôn dạng lệnh: R-type (Register).
    - Opcode: add có opcode là 0110011.
    - Thanh ghi đích (rd): s0 là x8.
    - Thanh ghi nguồn 1 (rs1): x0.
    - Thanh ghi nguồn 2 (rs2): zero, cũng là x0.
    - Funct3: Đối với lệnh add, funct3 là 000.
    - Funct7: Đối với lệnh add, funct7 là 0000000.
    - Mã máy: Từ đây, mã máy sẽ là **0000000 00000 00000 000 01000 0110011**.

Nếu ta sửa lệnh addi s0, zero, 0x512 thành giá trị ngoài phạm vi của số nguyên 12 bit (ví dụ, addi s0, zero, 0x812), công cụ sẽ báo lỗi vì giá trị đó vượt quá giới hạn của một số nguyên 12 bit có dấu.

- **Giải thích**: Lệnh addi chỉ hỗ trợ gán các số nguyên có dấu trong phạm vi từ -2048 đến 2047 (12 bit). Nếu ta cố gán giá trị lớn hơn 2047 hoặc nhỏ hơn -2048, lỗi sẽ xuất hiện vì giá trị đó không thể biểu diễn bằng 12 bit.

## Assignment 2

### 1. Quan sát giá trị của thanh ghi s0 và pc sau mỗi lệnh:




Sau lệnh lui s0, 0x20232:

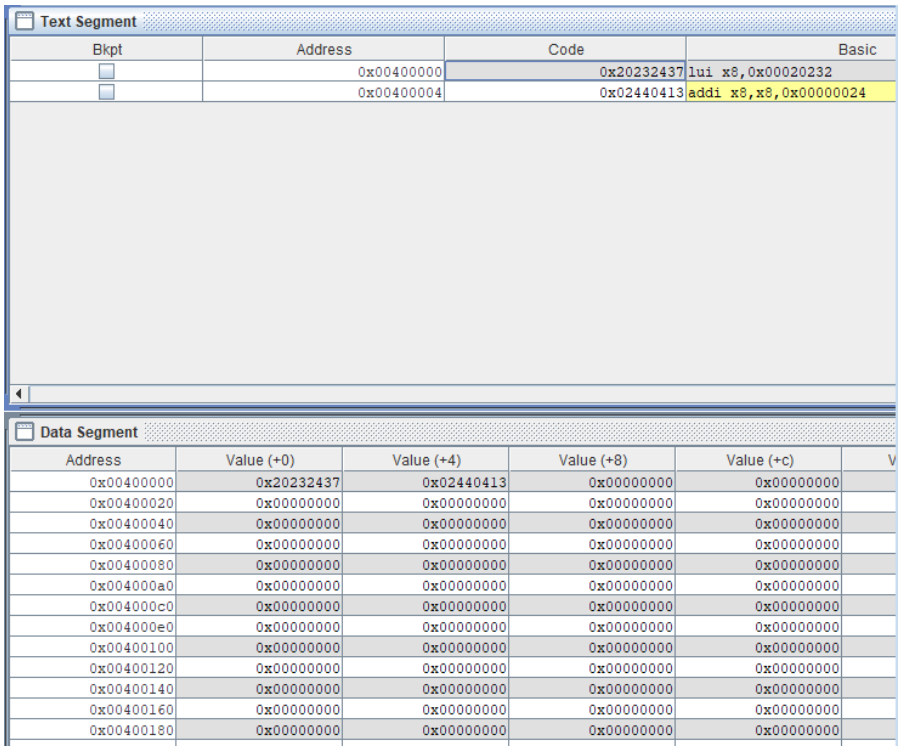
- Giá trị s0: 0x20232000
- Giá trị pc: Tăng thêm 4.

Sau lệnh addi s0, s0, 0x024:

- Giá trị s0: 0x20232024
- Giá trị pc: Tăng thêm 4.

### 2. So sánh dữ liệu trong Data Segment và mã máy trong Text Segment:

Các byte đầu tiên ở vùng lệnh trùng với cột Code(Mã máy theo Hexa) trong cửa sổ Text Segment ở phần thực thi



Text Segment

| Bkpt | Address | Code | Basic |
|---|---|---|---|
| | 0x00400000 | 0x20232437 | lui x8,0x00020232 |
| | 0x00400004 | 0x02440413 | addi x8,x8,0x00000024 |

Data Segment

| Address | Value (+0) | Value (+4) | Value (+8) | Value (+c) |
|---|---|---|---|---|
| 0x00400000 | 0x20232437 | 0x02440413 | 0x00000000 | 0x00000000 |
| 0x00400020 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00400040 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00400060 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00400080 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x004000a0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x004000c0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x004000e0 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00400100 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00400120 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00400140 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00400160 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |
| 0x00400180 | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |

**3. Giải thích về mở rộng dấu và tại sao cần tăng giá trị trong lui:**

Khi giá trị 12-bit của addi là số âm, mở rộng dấu (sign-extension) làm thay đổi giá trị. Vì vậy, cần tăng giá trị trong lệnh lui để bù đắp phần mở rộng dấu, đảm bảo số 32-bit đúng.

**4. Nạp số 32-bit 0xFEEDB987 vào thanh ghi:**

lui s0, 0xFEEDC  # s0 = 0xFEEDC000

addi s0, s0, 0x987 # s0 = 0xFEEDB987

## Assignment 3

Kết quả:

| Basic | Source |
|---|---|
| lui x8,0x00020232 | 2: li s0, 0x20232024 |
| addi x8,x8,0x00000024 | |
| addi x8,x0,0x00000020 | 3: li s0, 0x20 |

**Giải thích tại sao li tách thành 2 lệnh**

Lệnh li s0, 0x20232024: Lệnh này được giả lập bởi các lệnh thực:

- lui x8, 0x20232: Nạp phần 20 bit cao của số 0x20232024 vào thanh ghi s0 (được dịch là x8 trong mã máy).
- addi x8, x8, 0x24: Cộng thêm phần 12 bit thấp của số 0x20232024 vào thanh ghi s0.

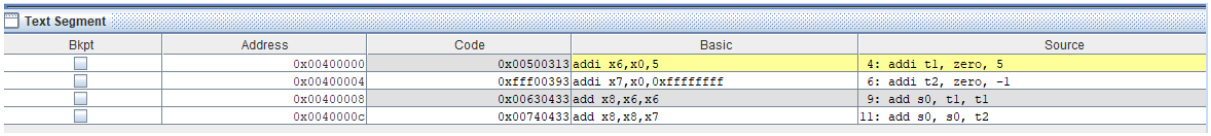
Lệnh li s0, 0x20: Đây là một giá trị nhỏ (trong phạm vi 12 bit), vì vậy chỉ cần một lệnh:

- addi x8, x0, 0x20: Nạp giá trị 0x20 vào thanh ghi s0 (tương ứng với x8)

## Assignment 4

```
# Laboratory Exercise 2, 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
```

Kết quả chạy:



Text Segment

| Bkpt | Address | Code | Basic | Source |
|---|---|---|---|---|
| | 0x00400000 | 0x00500313 | addi x6,x0,5 | 4: addi t1, zero, 5 |
| | 0x00400004 | 0xfff00393 | addi x7,x0,0xffffffff | 6: addi t2, zero, -1 |
| | 0x00400008 | 0x00630433 | add x8,x6,x6 | 9: add s0, t1, t1 |
| | 0x0040000c | 0x00740433 | add x8,x8,x7 | 11: add s0, s0, t2 |

| | | |
|---|---|---|
| t0 | 5 | 0x00000000 |
| t1 | 6 | 0x00000005 |
| t2 | 7 | 0xffffffff |
| s0 | 8 | 0x00000009 |

| | | |
|---|---|---|
| pc | | 0x0040000 |

| | | |
|---|---|---|
| pc | | 0x00400004 |

Có sự thay đổi của các thanh ghi: $t1 có giá trị trở thành 0x00000005, $t2 có giá trị trở thành 0xffffffff, $s0 thany đổi theo như kết quả giống khi thực hiện phép tính đã được giải thích như trong mã nguồn trên, còn thanh ghi pc cứ mỗi một lệnh tăng thêm 0x00000004.

**1. Lệnh addi t1, zero, 5 (I-type):**

- Mã máy: 0x00500093
- Cấu trúc I-type: imm[11:0] | rs1 | funct3 | rd | opcode

Phân tích:

  - **imm[11:0]**: 000000000101 (5)
  - **rs1**: 00000 (zero)
  - **funct3**: 000
  - **rd**: 01010 (t1)
  - **opcode**: 0010011 (addi)

Mã nhị phân:

**000000000101 00000 000 01010 0010011**

Kết quả:

**00000000010100000000010100010011**

**2. Lệnh addi t2, zero, -1 (I-type):**

- Mã máy: 0xfff00113
- Cấu trúc I-type: imm[11:0] | rs1 | funct3 | rd | opcode

Phân tích:

- **imm[11:0]**: 111111111111 (-1 trong bù 2)
- **rs1**: 00000 (zero)
- **funct3**: 000
- **rd**: 01011 (t2)
- **opcode**: 0010011 (addi)

Mã nhị phân:

**111111111111 00000 000 01011 0010011**

Kết quả:

**11111111111100000000010110010011**

**3. Lệnh add s0, t1, t1 (R-type):**

- Mã máy: 0x00208033
- Cấu trúc R-type: funct7 | rs2 | rs1 | funct3 | rd | opcode

Phân tích:

- **funct7**: 0000000
- **rs2**: 01010 (t1)
- **rs1**: 01010 (t1)
- **funct3**: 000
- **rd**: 01000 (s0)
- **opcode**: 0110011 (add)

Mã nhị phân:

**0000000 01010 01010 000 01000 0110011**

Kết quả:

**00000000101001010000010000110011**

**4. Lệnh add s0, s0, t2 (R-type):**

- Mã máy: 0x00208133
- Cấu trúc R-type: funct7 | rs2 | rs1 | funct3 | rd | opcode

Phân tích:

- **funct7**: 0000000

- **rs2**: 01011 (t2)
- **rs1**: 01000 (s0)
- **funct3**: 000
- **rd**: 01000 (s0)
- **opcode**: 0110011 (add)

Mã nhị phân:

**0000000 01011 01000 000 01000 0110011**

Kết quả:

**00000000101101000000010000110011**

## Assignment 5

```
#Laboratory Exercise 2, Assignment 5
.text
   # Assign X, Y into t1, t2 register
   addi t1, zero, 4          # X = t1 =?
   addi t2, zero, 5          # Y = t2 =?

   # Expression Z = X * Y
   mul s1, t1, t2           # s1 chứa 32 bit thấp
```

Kết quả:

Text Segment

| Bkpt | Address | Code | Basic | Source |
|---|---|---|---|---|
| | 0x00400000 | 0x00400313 | addi x6,x0,4 | 4: addi t1, zero, 4 # X = t1 =? |
| | 0x00400004 | 0x00500393 | addi x7,x0,5 | 5: addi t2, zero, 5 # Y = t2 =? |
| | 0x00400008 | 0x027304b3 | mul x9,x6,x7 | 8: mul s1, t1, t2 # s1 ch?a 32 bit th?p |

| | | |
|---|---|---|
| t1 | 6 | 0x00000004 |
| t2 | 7 | 0x00000005 |
| s0 | 8 | 0x00000000 |
| s1 | 9 | 0x00000014 |

| | | |
|---|---|---|
| pc | | 0x00400000 |

| | | |
|---|---|---|
| pc | | 0x00400004 |

- **t1 = 0x00000004** (tương đương giá trị thập phân 4) — đây là giá trị của biến X.
- **t2 = 0x00000005** (tương đương giá trị thập phân 5) — đây là giá trị của biến Y.

- **s1 = 0x00000014** (tương đương giá trị thập phân 20) — đây là kết quả của phép nhân 4 * 5 = 20, giá trị lưu trong s1.

**1. Giải thích chi tiết đoạn lệnh trong chương trình:**

*addi t1, zero, 4 # Gán giá trị 4 vào thanh ghi t1 (X = 4)*

*addi t2, zero, 5 # Gán giá trị 5 vào thanh ghi t2 (Y = 5)*

*mul s1, t1, t2 # Nhân t1 với t2, lưu kết quả 32 bit thấp vào thanh ghi s1*

**Giải thích từng lệnh:**

- **addi t1, zero, 4:**
  - Đây là lệnh cộng tức thời (I-type), gán giá trị 4 vào thanh ghi t1. Thanh ghi zero luôn có giá trị bằng 0, nên lệnh này thực tế là t1 = 0 + 4.
  - Kết quả: t1 = 4.
- **addi t2, zero, 5:**
  - Tương tự như lệnh trước, nhưng gán giá trị 5 vào thanh ghi t2.
  - Kết quả: t2 = 5.
- **mul s1, t1, t2:**
  - Đây là lệnh nhân (R-type) từ phần mở rộng RV32M (Multiply/Divide). Lệnh này nhân giá trị của hai thanh ghi t1 và t2, sau đó lưu **32 bit thấp** của kết quả vào thanh ghi s1.
  - Trong trường hợp này: s1 = 4 * 5 = 20.
  - Kết quả: s1 = 20.

Kết quả cuối cùng của phép nhân là **20**, được lưu trong thanh ghi s1, tương đương với kết quả trong ảnh là 0x00000014 (20).

**2. Tìm hiểu lệnh chia trong RISC-V**

Trong RISC-V, lệnh chia thuộc phần mở rộng **RV32M** với các lệnh chia cơ bản như sau:

- **div rd, rs1, rs2:** Chia hai số có dấu (signed), lưu kết quả thương (quotient) vào thanh ghi đích rd.
- **divu rd, rs1, rs2:** Chia hai số không dấu (unsigned), lưu kết quả thương vào thanh ghi đích rd.
- **rem rd, rs1, rs2:** Lấy phần dư của phép chia có dấu.
- **remu rd, rs1, rs2:** Lấy phần dư của phép chia không dấu.

**Ví dụ về lệnh chia:**

Dưới đây là đoạn code minh họa thực hiện phép chia và lấy phần dư:

```
addi t1, zero, 10      # t1 = 10
addi t2, zero, 3       # t2 = 3


div s1, t1, t2         # s1 = t1 / t2 (s1 = 10 / 3 = 3)
rem s2, t1, t2         # s2 = t1 % t2 (s2 = 10 % 3 = 1)
```

**Giải thích từng lệnh:**

- **addi t1, zero, 10:**
  - Gán giá trị 10 cho thanh ghi t1.
  - Kết quả: t1 = 10.
- **addi t2, zero, 3:**
  - Gán giá trị 3 cho thanh ghi t2.
  - Kết quả: t2 = 3.
- **div s1, t1, t2:**
  - Thực hiện phép chia có dấu, chia t1 cho t2 (10 / 3). Kết quả của phép chia là thương số 3, lưu vào thanh ghi s1.
  - Kết quả: s1 = 3.
- **rem s2, t1, t2:**
  - Thực hiện phép lấy phần dư của phép chia t1 cho t2 (10 % 3). Phần dư là 1, lưu vào thanh ghi s2.
  - Kết quả: s2 = 1.

**Luồng hoạt động của lệnh chia:**

1. **Thực hiện phép chia 10 / 3:**
   - Kết quả thương số là **3**, được lưu trong thanh ghi s1.
2. **Thực hiện phép lấy phần dư 10 % 3:**
   - Phần dư là **1**, được lưu trong thanh ghi s2.

**Kết quả của các thanh ghi:**

- **Thanh ghi s1:** Chứa giá trị **3**, là thương của phép chia 10 / 3.
- **Thanh ghi s2:** Chứa giá trị **1**, là phần dư của phép chia 10 % 3.

## Assignment 6

Code:

```
1  # Laboratory Exercise 2, Assignment 6
2  .data              # Khởi tạo biến (declare memory)
3     X: .word 5      # Biến X, kiểu word (4 bytes), giá trị khởi tạo = 5
4     Y: .word -1     # Biến Y, kiểu word (4 bytes), giá trị khởi tạo = -1
5     Z: .word 0      # Biến Z, kiểu word (4 bytes), giá trị khởi tạo = 0
6
7  .text           # Khởi tạo lệnh (declare instruction)
8     # Nạp giá trị X và Y vào các thanh ghi
9     la t5, X        # Lấy địa chỉ của X trong vùng nhớ chứa dữ liệu
0     la t6, Y        # Lấy địa chỉ của Y
1     lw t1, 0(t5)    # t1 = X
2     lw t2, 0(t6)    # t2 = Y
3
4     # Tính biểu thức Z = 2X + Y với các thanh ghi
5     add s0, t1, t1
6     add s0, s0, t2
7     # Lưu kết quả từ thanh ghi vào bộ nhớ
8     la t4, Z
9     # Lấy địa chỉ của Z
0     sw s0, 0(t4)    # Lưu giá trị của Z từ thanh ghi vào bộ nhớ
```

Kết quả:

Text Segment

| Bkpt | Address | Code | Basic | Source |
|---|---|---|---|---|
| | 0x00400000 | 0x0fc10f17 | auipc x30,0x0000fc10 | 9: la t5, X # L?y ??a ch? c?a X trong vùng ... |
| | 0x00400004 | 0x000f0f13 | addi x30,x30,0 | |
| | 0x00400008 | 0x0fc10f97 | auipc x31,0x0000fc10 | 10: la t6, Y # L?y ??a ch? c?a Y |
| | 0x0040000c | 0xffcf8f93 | addi x31,x31,0xfffffffc | |
| | 0x00400010 | 0x000f2303 | lw x6,0(x30) | 11: lw t1, 0(t5) # t1 = X |
| | 0x00400014 | 0x000fa383 | lw x7,0(x31) | 12: lw t2, 0(t6) # t2 = Y |
| | 0x00400018 | 0x00630433 | add x8,x6,x6 | 15: add s0, t1, t1 |
| | 0x0040001c | 0x00740433 | add x8,x8,x7 | 16: add s0, s0, t2 |
| | 0x00400020 | 0x0fc10e97 | auipc x29,0x0000fc10 | 18: la t4, Z |
| | 0x00400024 | 0xfe8e8e93 | addi x29,x29,0xfffffff8 | |
| | 0x00400028 | 0x008ea023 | sw x8,0(x29) | 20: sw s0, 0(t4) # L?u giá tr? c?a Z t? thanh gh... |

- **Lệnh la t5, X:**
  - Cơ chế hoạt động: Lệnh la (load address) được dùng để tải địa chỉ của một biến từ bộ nhớ vào thanh ghi. Nó thực hiện việc nạp địa chỉ của biến X vào thanh ghi t5.
  - Cách biên dịch: Lệnh này thường được dịch thành một hoặc hai lệnh như auipc (add upper immediate to PC) và addi để tính toán địa chỉ thực trong bộ nhớ.

Trước khi chạy:

| 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 | 0x00000000 |
| a1 | 11 | 0x00000000 |
| a2 | 12 | 0x00000000 |
| a3 | 13 | 0x00000000 |
| a4 | 14 | 0x00000000 |
| a5 | 15 | 0x00000000 |
| a6 | 16 | 0x00000000 |
| a7 | 17 | 0x00000000 |
| 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 | | 0x00400000 |

Sau khi chạy: t6, t1,t2,s0,t4 thay đổi giá trị. Pc cứ mỗi bước tang lên 4 đơn vị.

| Name | Number | Value |
|---|---|---|
| zero | 0 | 0x00000000 |
| ra | 1 | 0x00000000 |
| sp | 2 | 0x7fffeffc |
| gp | 3 | 0x10008000 |
| tp | 4 | 0x00000000 |
| t0 | 5 | 0x00000000 |
| t1 | 6 | 0x00000005 |
| t2 | 7 | 0xffffffff |
| s0 | 8 | 0x00000009 |
| s1 | 9 | 0x00000000 |
| a0 | 10 | 0x00000000 |
| a1 | 11 | 0x00000000 |
| a2 | 12 | 0x00000000 |
| a3 | 13 | 0x00000000 |
| a4 | 14 | 0x00000000 |
| a5 | 15 | 0x00000000 |
| a6 | 16 | 0x00000000 |
| a7 | 17 | 0x00000000 |
| 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 | 0x10010008 |
| t5 | 30 | 0x10010000 |
| t6 | 31 | 0x10010004 |
| pc | | 0x00400030 |

- Lệnh lw: Lấy địa chỉ của biến kiểu word và lưu vào 1 thanh ghi

- Lệnh sw: Lấy địa chỉ của biến kiểu word lưu vào bộ nhớ

- lb (load byte): Nạp 1 byte (8 bit) từ bộ nhớ vào thanh ghi, với việc mở rộng dấu (sign-extended) thành 32 bit.

- sb (store byte): Lưu 1 byte từ thanh ghi vào bộ nhớ.