BÁO CÁO THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Lab 2 : Tập lệnh, các lệnh cơ bản, các chỉ thị biên dịch

Họ và tên	MSSV
Phạm Minh Hiển	20235705

Assignment 1: Lệnh gán số nguyên nhỏ 12-bit

```
# Laboratory Exercise 2, Assignment 1
.text
        addi s0, zero, 0x512
        #addi s0, zero, 0x20232024

        add s0, x0, zero
```

- Sự thay đổi của thanh ghi s0 và pc khi chạy từng lệnh :

Thời điểm	s0-value	pc-value
Ban đầu chưa chạy	0x00000000	0x00400000
Sau lệnh 1	0x00000512	0x00400004
Sau lệnh 2	0x00000000	0x00400008

Nhận xét :

- Sau mỗi lệnh giá trị của thanh ghi pc tăng thêm 4 byte để trỏ đến lệnh tiếp theo.
- Sau lệnh thứ nhất giá trị của thanh ghi s0 thay đổi do ta đã gán giá trị 0x512 dưới dạng 32-bit là 0x00000512.

- So sánh mã máy của các lệnh trên với khuôn dạng lệnh :

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x51200413	addi x8,x0,0x00000512	3: addi s0, zero, 0x512
	0x00400004	0x00000433	add x8,x0,x0	6: add s0, x0, zero

+ Lệnh addi :

- **opcode:** Lệnh addi có opcode = 0010011 .
- **funct3:** Lệnh addi có funct3 = 000.
- **rd:** Thanh ghi s0 có số hiệu là 8 , nên rd = 01000.
- **rs1:** Thanh ghi zero có số hiệu là 0, nên rs1 = 00000.
- **imm:** Hằng số 0x512 được chuyển sang nhị phân là 010100010010 (12-bit).

Kết hợp lại ta có mã máy : 0101 0001 0010 0000 0000 0100 0001 0011

Chuyển sang hexa : 0x51200413 -> Chính xác.

+ Lệnh add :

- **opcode:** Lệnh add có opcode = 0110011.
- **funct3:** Lệnh add có funct3 = 000.
- **funct7:** Lệnh add có funct7 = 0000000.
- **rd:** Thanh ghi s0 có số hiệu là 8, nên rd = 01000.
- **rs1:** Thanh ghi x0 (cũng là zero) có số hiệu là 0, nên rs1 = 00000.
- **rs2:** Thanh ghi zero có số hiệu là 0, nên rs2 = 00000.

Kết hợp lại ta có mã máy : 0000 0000 0000 0000 0000 0100 0011 0011

Chuyển sang hexa : 0x00000433 -> Chính xác.

- Sửa lại lệnh addi :

```
# Laboratory Exercise 2, Assignment 1
.text
        #addi s0, zero, 0x512
        addi s0, zero, 0x20232024

        add s0, x0, zero
```

+ Kết quả : Chương trình bị lỗi không thực hiện được.

+ Nguyên nhân : Lệnh addi chỉ chấp nhận hằng số 12-bit signed immediate. Giá trị 0x20232024 vượt quá phạm vi này (nó là một giá trị 32-bit), nên RARS sẽ báo lỗi khi biên dịch.

Assignment 2 : Lệnh gán số 32-bit

```
# Laboratory Exercise 2, Assignment 2
# Load 0x20232024 to s0 register
.text
        lui  s0, 0x20232

        addi s0, s0, 0x024
```

- Sự thay đổi của thanh ghi s0 và pc :

Thời điểm	s0-value	pc-value
Ban đầu chưa chạy	0x00000000	0x00400000
Sau lệnh 1	0x20232000	0x00400004
Sau lệnh 2	0x20232024	0x00400008

Nhân xét :

- Sau mỗi lệnh giá trị của thanh ghi pc tăng thêm 4 byte để trỏ đến lệnh tiếp theo.
- Sau lệnh **lui** giá trị thanh ghi s0 được thay đổi thành 0x20232000 (với 20232 là 20 bit trọng số cao), sau lệnh **addi** giá trị thanh ghi sẽ thêm 12 bit trọng số thấp là 024 và thành 0x20232024.

- So sánh dữ liệu trong Data Segment và mã máy Text Segment :

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20232437	lui x8,0x00020232	4: lui s0, 0x20232
	0x00400004	0x02440413	addi x8,x8,0x00000024	6: addi s0, s0, 0x024

Data Segment

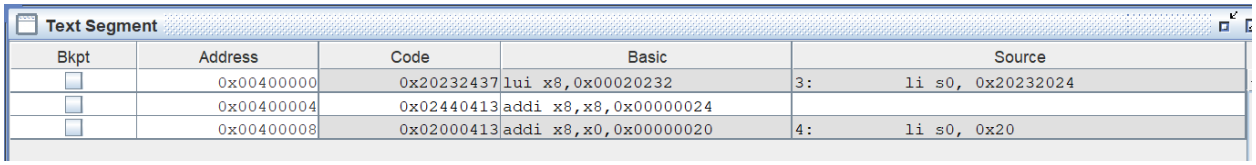
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x00400000	0x20232437	0x02440413	0x00000000	0x00000000
0x00400020	0x00000000	0x00000000	0x00000000	0x00000000
0x00400040	0x00000000	0x00000000	0x00000000	0x00000000
0x00400060	0x00000000	0x00000000	0x00000000	0x00000000
0x00400080	0x00000000	0x00000000	0x00000000	0x00000000
0x004000a0	0x00000000	0x00000000	0x00000000	0x00000000

Ta thấy value của địa chỉ 0x00400000 trong Data Segment là 0x20232437 tương ứng với mã máy của địa chỉ này trên Text Segment. Sau đó, giá trị tại địa chỉ này sau 4 byte là 0x02440413 cũng tương đồng với mã máy tại địa chỉ 0x00400004.

Assignment 3 : Lệnh gán (giả lệnh)

```
# Laboratory Exercise 2, Assignment 3
.text
        li s0, 0x20232024
        li s0, 0x20
```

- So sánh các lệnh ở cột Source và cột Basic trong cửa sổ Text Segment :



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20232437	lui x8,0x00020232	3: li s0, 0x20232024
	0x00400004	0x02440413	addi x8,x8,0x00000024	
	0x00400008	0x02000413	addi x8,x0,0x00000020	4: li s0, 0x20

Ta thấy bên cột Basic có 3 lệnh trong khi đó cột Source chỉ có 2 lệnh. Điều này xảy ra là do khi thực hiện lệnh giả **li** do 0x20232024 là giá trị 32-bit vượt quá 12-bit của lệnh **addi** nên lệnh giả **li** sẽ mở rộng thành 2 lệnh máy là **lui x8, 0x00020232** để nạp 20-bit trọng số cao vào s0 và dịch trái 12-bit (kết quả giá trị s0 lúc này là 0x20232000) và **addi x8, x8, 0x00000024** cộng tiếp 12 bit thấp vào s0 và cho ra kết quả thanh ghi s0 là 0x20232024. Sau đó, ta thấy lệnh **li s0, 0x20** thêm giá trị 0x20 (32) nằm trong phạm vi 12-bit của **addi** nên trực tiếp nạp vào s0, khi này giá trị thanh ghi s0 là 0x00000020.

Assignment 4 : Tính biểu thức 2x + y = ?

```
# Laboratory Exercise Assignment 4
.text
        addi t1, zero, 5
        addi t2, zero, -1

        add s0, t1, t1
        add s0, s0, t2
```

- Sự thay đổi giá trị của các thanh ghi :

Thời điểm	s0-value	pc-value	t1-value	t2-value
Ban đầu chưa chạy	0x00000000	0x00400000	0x00000000	0x00000000
Sau lệnh 1	0x00000000	0x00400004	0x00000005	0x00000000
Sau lệnh 2	0x00000000	0x00400008	0x00000005	0xffffffff
Sau lệnh 3	0x0000000a	0x0040000c	0x00000005	0xffffffff
Sau lệnh 4	0x00000009	0x00400010	0x00000005	0xffffffff

Nhận xét :

- Giá trị thanh ghi pc vẫn thêm 4 byte sau mỗi lệnh được thực hiện để chỉ đến lệnh kế tiếp.
- Ta thấy thanh ghi t1 sau lệnh thứ nhất đã được gán giá trị 5, thanh ghi t2 được gán giá trị -1 (0xffffffff trong hexa) sau lệnh 2.
- Giá trị thanh ghi s0 sau lệnh 3 (lệnh thực hiện lấy giá trị 2X) là 10 (0x0000000a) và cho ra kết quả chính xác là 9 sau khi thực hiện câu lệnh 4 (câu lệnh lấy 2X+Y).

- So sánh mã máy :

  + Lệnh addi thứ 1 :

- **opcode:** Lệnh addi có opcode = 0010011 .
- **funct3:** Lệnh addi có funct3 = 000.
- **rd:** Thanh ghi t1 có số hiệu là 6 , nên rd = 00110.
- **rs1:** Thanh ghi zero có số hiệu là 0, nên rs1 = 00000.
- **imm:** Hằng số 5 được chuyển sang nhị phân là 000000000101 (12-bit).

Kết hợp lại ta có mã máy : 0000 0000 0101 0000 0000 0011 0001 0011

Chuyển sang hexa : 0x00500313 -> Chính xác.

+ Lệnh addi thứ 2 :

- **opcode:** Lệnh addi có opcode = 0010011 .
- **funct3:** Lệnh addi có funct3 = 000.
- **rd:** Thanh ghi t2 có số hiệu là 7 , nên rd = 00111.
- **rs1:** Thanh ghi zero có số hiệu là 0, nên rs1 = 00000.
- **imm:** Hằng số -1 được chuyển sang nhị phân là 111111111111 (12-bit).

Kết hợp lại ta có mã máy : 1111 1111 1111 0000 0000 0011 1001 0011

Chuyển sang hexa : 0xfff00393 -> Chính xác.

+ Lệnh add thứ 1 :

- **opcode:** Lệnh add có opcode = 0110011 .
- **funct3:** Lệnh add có funct3 = 000.
- **funct7:** Lệnh add có funct7 = 0000000.
- **rd:** Thanh ghi s0 có số hiệu là 8 , nên rd = 01000.
- **rs1:** Thanh ghi t1 có số hiệu là 6, nên rs1 = 00110.
- **rs2:** Thanh ghi t1 có số hiệu là 6, nên rs2 = 00110.

Kết hợp lại ta có mã máy : 0000 0000 0110 0011 0000 0100 0011 0011

Chuyển sang hexa : 0x00630433 -> Chính xác.

+ Lệnh add thứ 2 :

- **opcode:** Lệnh add có opcode = 0110011 .
- **funct3:** Lệnh add có funct3 = 000.
- **funct7:** Lệnh add có funct7 = 0000000.
- **rd:** Thanh ghi s0 có số hiệu là 8 , nên rd = 01000.
- **rs1:** Thanh ghi s0 có số hiệu là 8, nên rs1 = 01000.
- **rs2:** Thanh ghi t2 có số hiệu là 7, nên rs2 = 00111.

Kết hợp lại ta có mã máy : 0000 0000 0111 0100 0000 0100 0011 0011

Chuyển sang hexa : 0x00740433 -> Chính xác.

Assignment 5 : Phép nhân

```
# Laboratory Exercise 5
.text
        addi t1, zero, 4
        addi t2, zero, 5

        mul s1, t1, t2
```

- Sự thay đổi của các thanh ghi :

Thời điểm	s1-value	pc-value	t1-value	t2-value
Ban đầu chưa chạy	0x00000000	0x00400000	0x00000000	0x00000000
Sau lệnh 1	0x00000000	0x00400004	0x00000004	0x00000000
Sau lệnh 2	0x00000000	0x00400008	0x00000004	0x00000005
Sau lệnh 3	0x00000014	0x0040000c	0x00000004	0x00000005

Nhận xét :

- Giá trị thanh ghi pc vẫn thêm 4 byte sau mỗi lệnh được thực hiện để chỉ đến lệnh kế tiếp.
- Thanh ghi t1 ban đầu có giá trị là 0, sau lệnh **addi t1, zero, 4** thì đã được gán giá trị mới là 4, tương tự thanh ghi t2 cũng có giá trị mới được gán là 5.
- Thanh ghi s1 sau lệnh **mul s1, t1, t2** thực hiện phép nhân 2 giá trị lưu trữ trong thanh ghi t1 và t2 lần lượt là 4 và 5 thì có giá trị là 0x00000014 (tức là 20), là kết quả chính xác cho phép nhân 2 giá trị 4 và 5.

- Lệnh chia :
  div : chia có dấu lấy thương (**divu** : không dấu)
  rem : chia có dấu lấy dư (**remu** : có dấu)

```
.text
        addi t0, zero, 10
        addi t1, zero, -3
        div  t2, t0, t1
        rem  t3, t0, t1
```

Cách hoạt động :
2 lệnh đầu thực hiện gán 2 giá trị 10 và -3 cho t0 và t1 để thực hiện phép chia 10 cho -3.

- **div :**
  - t2 = t0 / t1 = 10 / (-3) = -3 (làm tròn về 0), lúc này thanh ghi t2 và t1 có giá trị 0xfffffffd tương ứng với giá trị -3.
  - Lệnh thực hiện phép chia giá trị 10 trong t0 cho -3 trong t1 vào lưu giá trị vào t2.
- **rem :**
  - t3 = t0 % t1 = 10 - (-3)*(-3) = 10 - 9 = 1.
  - Lệnh thực hiện chia lấy dư và lưu vào t3.
- **Kết quả thanh ghi:**
  - t0 = 10
  - t1 = -3
  - t2 = -3
  - t3 = 1.

Assignment 6 : Tạo biến và truy cập biến

```
# Laboratory Exercise 2, Assignment 6
.data
        X: .word 5
        Y: .word -1
        Z: .word 0

.text
        la t5, X
        la t6, Y

        lw t1, 0(t5)    # t1 = X
        lw t2, 0(t6)    # t2 = Y
        add s0, t1, t1
        add s0, s0, t2
        # Lưu kết quả từ thanh ghi vào bộ nhớ
        la t4, Z
        # Lấy địa chỉ của Z
        sw s0, 0(t4)    # Lưu giá trị của Z từ thanh ghi vào bộ nhớ
```

- Lệnh **la** được biên dịch như thế nào? Giải thích cơ chế hoạt động

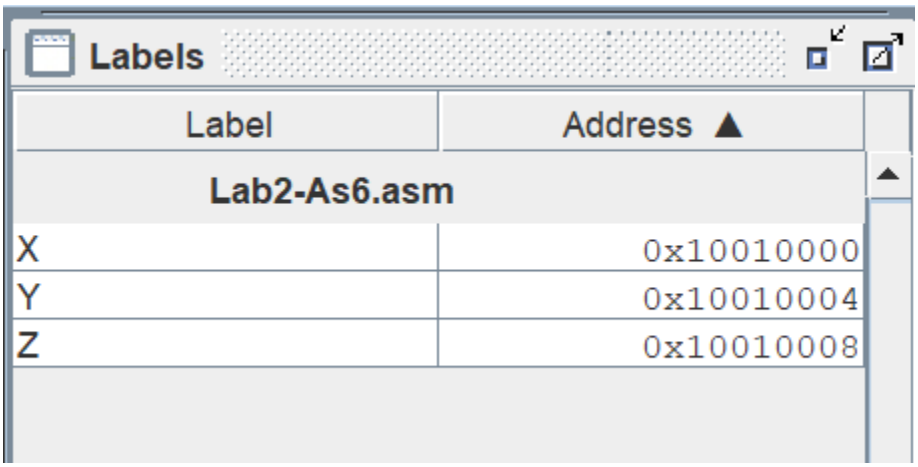
Trong RISC-V, **la** (load address) là một giả lệnh, được biên dịch thành 2 lệnh cơ bản:

- auipc (Add Upper Immediate to PC):
  Lệnh này tính địa chỉ cao 20 bit của nhãn (ví dụ: X, Y, Z) bằng cách cộng giá trị offset (từ PC) vào phần cao của địa chỉ.
  - Cú pháp: auipc rd, offset[31:12]
  - Ví dụ: auipc t5, 0x1000
- addi (Add Immediate):

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x0fc10f17	auipc x30,0x0000fc10	8: la t5, X
	0x00400004	0x000f0f13	addi x30,x30,0	
	0x00400008	0x0fc10f97	auipc x31,0x0000fc10	9: la t6, 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	13: add s0, t1, t1
	0x0040001c	0x00740433	add x8,x8,x7	14: add s0, s0, t2
	0x00400020	0x0fc10e97	auipc x29,0x0000fc10	16: la t4, Z
	0x00400024	0xfe8e8e93	addi x29,x29,0xffffffe8	
	0x00400028	0x008ea023	sw x8,0(x29)	18: sw s0, 0(t4) # Luu giá trị củ..

Ta thấy lệnh **la t5, X** được biên dịch thành **auipc x30, 0x0000fc10** và **addi x30, x30 ,0** lần lượt để tính địa chỉ cao dựa trên pc và thêm phần thấp của địa chỉ.



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x10010000	0x00000005	0xffffffff	0x00000009	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000

Thời điểm	pc-value	t1-value	t2-value	t5-value	t6-value
Ban đầu chưa chạy	0x00400000	0x00000000	0x00000000	0x00000000	0x00000000
Sau lệnh 1	0x00400004	0x00000000	0x00000000	0x10010000	0x00000000
Sau lệnh 2	0x00400008	0x00000000	0x00000000	0x10010000	0x00000000
Sau lệnh 3	0x0040000c	0x00000000	0x00000000	0x10010000	0x10010008
Sau lệnh 4	0x00400010	0x00000000	0x00000000	0x10010000	0x10010004
Sau lệnh 5	0x00400014	0x00000005	0x00000000	0x10010000	0x10010004
Sau lệnh 6	0x00400018	0x00000005	0xffffffff	0x10010000	0x10010004

Sau lệnh 7: s0 có giá trị 0x0000000a

Sau lệnh 8: s0 có giá trị 0x00000009

Sau lệnh 10: t4 có giá trị 0x10010020

Sau lệnh 11: t4 có giá trị 0x10010008

- Kiểm tra giá trị trong Data Segment:
  X chứa 0x00000005
  Y chứa 0xffffffff
  Z ban đầu chứa 0x00000000, sau khi chạy chương trình cập nhật thành 0x00000009(vì 2*5 +(-1) = 9).

- **lw (Load Word):** Tải giá trị từ bộ nhớ vào thanh ghi.
  - Ví dụ: lw t1, 0(t5) → t1 = giá trị tại địa chỉ [t5 + 0] (tức giá trị của X).
- **sw (Store Word):** Lưu giá trị từ thanh ghi vào bộ nhớ.
  - Ví dụ: sw s0, 0(t4) → Giá trị của s0 được lưu vào địa chỉ Z.

Lệnh lb và sb

- **lb (Load Byte):**
  - Tải 1 byte từ bộ nhớ vào thanh ghi, thực hiện **sign-extend** để mở rộng thành 32 bit.
  - Ví dụ: lb t0, 0(t5) → Tải byte tại địa chỉ t5, mở rộng dấu thành giá trị 32 bit.
- **sb (Store Byte):**
  - Lưu byte thấp nhất (8 bit) của thanh ghi vào bộ nhớ.
  - Ví dụ: sb t0, 0(t5) → Lưu byte thấp của t0 vào địa chỉ t5.