BÁO CÁO THỨC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN 2

Họ và tên: Vũ Đức Hoàng Anh

MSSV: 20235658

1. Assignment 1:

- **Nhập chương trình:**

```
# Laboratory exercise 2, Assignment 1
.text
        addi s0, zero, 0x512    # s0 = 0 + 0x512; I-type: chỉ có thể lưu
                                # được hằng số có dấu 12 bits
        add s0, x0, zero        # s0 = 0 + 0; R-type: có thể dử dụng số
                                # hiệu thanhn ghi thay cho tên thanh ghi
```

- **Quan sát kết quả trên cửa sổ Register:**

Trạng thái	s0	pc
Ban đầu	0x00000000	0x00400000
Thực hiện addi	0x00000512	0x00400004
Thực hiện add	0x00000000	0x00400008

- **Lệnh addi s0, zero, 0x512**

  + Khuôn lệnh dạng: I-type (Immediate).
  + opcode: 0010011 (7 bit)
  + rd (s0): 0100 (5 bit)
  + funct3: 000 (3 bit)
  + rs1 (zero): 00000 (5 bit)
  + imm (0x512): 1010 0010 010 (12 bit)
  + Mã máy: **10100010010000000000010000010011**
  + Sau khi thực hiện câu lệnh này giá trị của s0 tăng từ 0x00000000 thành 0x00000512. Giá trị của pc tăng từ 0x00400000 thành 0x00400004.

- **Lệnh add s0, x0, zero**

  + Khuôn lệnh dạng: R-type (Register).

+ opcode: 0110011 (7 bit)
+ rd (s0): 0100 (5 bit)
+ rs1 (x0): 00000 (5 bit)
+ rs2 (zero): 00000 (5 bit)
+ funct3: 000 (3 bit)
+ funct7: 0000000 (7 bit)
+ Mã máy: **00000000000000000000010000110011**
+ Sau khi thực hiện câu lệnh này giá trị của s0 từ 0x00000512 quay lại 0x0000000. Giá trị của pc tăng từ 0x00400004 thành 0x00400008.

- **Thử nghiệm**: 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.

2. Assignment 2:

- **Nhập chương trình:**

```
# Laboratory Exrcise 2, Assignment 2
# Load 0x20232024 to s0 reister
.text
        lui s0, 0x20232      # s0 = 0x20232
        addi s0, s0, 0x024  # s0 = s0 + 0x024
```

- **Quan sát kết quả trên cửa sổ Register:**

Trạng thái	s0	pc
Ban đầu	0x00000000	0x00400000
Thực hiện lui	0x20232000	0x00400004
Thực hiện addi	0x20232024	0x00400008

- **Thực hiện lệnh lui s0, 0x20232**
+ Giá trị s0: 0x20232
+ Giá trị pc: Tăng thêm 4, giá trị sau khi thực hiện câu lệnh: 0x00400004

- **Thực hiện lệnh addi s0, s0, 0x024**
  + Giá trị s0: 0x20232024
  + Giá trị pc: Tăng thêm 4, giá trị sau khi thực hiện câu lệnh: 0x00400008
- **So sánh dữ liệu trong Data Segment và mã máy trong Text Segment:**

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

- **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.

3. Assignment 3:

- **Nhập chương trình:**

```
.text

        li s0, 0x20232024  # s0 = 0x20232024

        li s0, 0x20        # s0 = 0x20
```

- **Lệnh li s0, 0x20232024**
  + Lệnh tương đương với 2 lệnh :

```
        lui x8, 0x20232      # x8 = 0x20232
        addi x8, x8, 0x024 # x8 = x8 + 0x024
```

  + Khi lệnh được thực hiện nó tách thành 2 lệch có mã máy cấp thấp hơn. Lệnh *lui x8, 0x20232* gán giá trị 0x20232000 cho x8. Lệnh *addi x8, x8, 0x024* khiến cho x8 = 0x20232000 + 0x024.
  + Do giá trị của 0x20232024 là giá trị lớn hơn 12 bit nên được tách ra thành 2 phần và thực hiện lần lượt trên x8 trong mã máy
- **Lệnh li so, 0x20**

  + Lệnh tương đương với lệnh

```
        addi x8, x0, 0x20
```

  + Do giá trị của 0x20 là giá trị trong khoảng 12 bit nên nó được thực hiện bằng 1 lệnh cơ bản.

4. **Assignment 4:**

- **Nhập chương trình**

```
.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
```

- **Quan sát kết quả trên cửa sổ Register:**

Trạng thái	t1	t2	s0	pc
Ban đầu	0x00000000	0x00000000	0x00000000	0x00400004
Sau addi t1	0x00000005	0x00000000	0x00000000	0x00000008
Sau addi t2	0x00000005	0xffffffff	0x00000000	0x0000000c
Sau add s0	0x00000005	0xffffffff	0x0000000a	0x00000010
Sau add s0	0x00000005	0xffffffff	0x00000009	0x00000014

- **Nhận xét**

Có sự thay đổi của các thanh ghi:

+ Thanh t1 có giá trị trở thành 0x00000005
+ Thanh t2 có giá trị trở thành 0xffffffff
+ Thanh s0 thay đổi theo như kết quả giống khi thực hiện phép tính đã được giải thích như trong mã nguồn trên
+Thanh ghi pc cứ mỗi một lệnh tăng thêm 0x00000004
Kết quả đúng là 2*5 + (-1) = 9

⇨ Kết quả chương trình chạy đúng với kết quả thực tế.

- **Lệnh addi t1, zero, 5 (I-type):**

+ Khuôn dạng lệnh: I-type (Immediate).
+ opcode: 0010011 (7 bit)
+ rd (t1): 00110 (5 bit)
+ funct3: 000 (3 bit)
+ rs1 (zero): 00000 (5 bit)
+ imm (5): 000000000101 (12 bit) (được biểu diễn từ phải sang trái)
+ Mã máy: **000000000101 00000 000 00110 0010011**

- **Lệnh addi t2, zero, -1 (I-type):**
  +Khuôn dạng lệnh: I-type (Immediate).
  + opcode: 0010011 (7 bit)
  + rd (t2): 00111 (5 bit)
  + funct3: 000 (3 bit)
  + rs1 (zero): 00000 (5 bit)
  + imm (-1): 111111111111(12 bit) (được biểu diễn từ phải sang trái)
  + Mã máy: **111111111111 00000 000 00111 0010011**

- **Lệnh add s0, t1, t1 (R-type):**
  + Khuôn dạng lệnh: R-type (Register)
  + Opcode: add có opcode là 0110011
  + Thanh ghi đích (rd): 01000 (x8)
  + Thanh ghi nguồn 1 (rs1): 00110 (x6)
  + Thanh ghi nguồn 2 (rs2): 00110 (x6)
  + Funct3: Đối với lệnh add, funct3 là 000
  + Funct7: Đối với lệnh add, funct7 là 0000000
  + Mã máy: **0000000 00110 00110 000 01000 0110011**

- **Lệnh add s0, s0, t2 (R-type):**
  + Khuôn dạng lệnh: R-type (Register)
  + Opcode: add có opcode là 0110011
  + Thanh ghi đích (rd): 01000 (x8)
  + Thanh ghi nguồn 1 (rs1): 01000 (x8)
  + Thanh ghi nguồn 2 (rs2): 00111 (x7)
  + Funct3: Đối với lệnh add, funct3 là 000
  + Funct7: Đối với lệnh add, funct7 là 0000000
  + Mã máy: **0000000 00111 01000 000 01000 0110011**

5. Assignment 5:

- **Nhập chương trình**

```
#  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
```

- **Quan sát cửa sổ Register**

Trạng thái	t1	t2	S1	pc
Ban đầu	0x00000000	0x00000000	0x00000000	0x00400004
Sau addi t1	0x00000004	0x00000000	0x00000000	0x00000008
Sau addi t2	0x00000004	0x00000005	0x00000000	0x0000000c
Sau add s1	0x00000004	0x00000005	0x00000014	0x00000014

- **Nhận xét**

+ 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

- **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).

- **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.

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:
+ Thực hiện phép chia 10 / 3:
+ Kết quả thương số là 3, được lưu trong thanh ghi s1.
+ 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.

6. Assignment 6:

- **Nhập chương trình**

```
# Laboratory Exercise 2, Assignment 6
.data          # Khởi tạo biến (declare memory)
    X: .word 5      # Biến X, kiểu word (4 bytes), giá trị khởi tạo = 5
    Y: .word -1     # Biến Y, kiểu word (4 bytes), giá trị khởi tạo = -1
    Z: .word 0      # Biến Z, kiểu word (4 bytes), giá trị khởi tạo = 0

.text          # Khởi tạo lệnh (declare instruction)
# Nạp giá trị X và Y vào các thanh ghi
    la t5, X            # Lấy địa chỉ của X trong vùng nhớ chứa dữ liệu
    la t6, Y            # Lấy địa chỉ của Y
    lw t1, 0(t5)        # t1 = X
    lw t2, 0(t6)        # t2 = Y

# Tính biểu thức Z = 2X + Y với các thanh ghi
    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
```

- **Quan sát cửa số Register**

Thanh ghi	Giá trị ban đầu	Sau khi thực chương trình
t5	0x00000000	0x10010000
t6	0x00000000	0x10010008
t1	0x00000000	0x00000005
t2	0x00000000	0xffffffff
s0	0x00000000	0x1000000a
t4	0x00000000	0x10010020

+ Giá trị của thanh pc cứ mỗi bước tăng lên 4 đơn vị sau mỗi bước tính.

- **Giải thích sự thay đổi**

+ Thanh ghi t5: được dùng khi gọi lệnh la t5, X; có tác dụng nạp địa chỉ của biến X vào thanh t5

+ Thanh ghi t6: được dùng khi gọi lệnh la t6, Y; có tác dụng nạp địa chỉ của biến Y vào thanh t6

+ Thanh ghi t1: được dùng khi gọi lệnh lw t1, 0(t5); lấy giá trị của biến t1

lưu vào bộ nhớ (5)
+ Thanh ghi t2: được dùng khi gọi lệnh lw t2, 0(t6); lấy giá trị của biến t2 lưu vào bộ nhớ (-1)
+ Thanh ghi s0: được dùng khi gọi lệnh add s0, t1, t1;add s0, s0, t2; lưu giá trị của biểu thức Z = 2X + Y (10 khi gọi lệnh đầu, sau đó bằng 9 khi gọi lệnh tiếp)
+ Thanh ghi t4: được dùng khi gọi lệnh la t4, Z; có tác dụng nạp địa chỉ của biến Z vào thanh t4
+ Lệnh sw s0, 0(t4): Lấy giá trị lưu trong thanh s0 (9), lưu vào địa chỉ thanh t4 đang trỏ tới (biến Z); lệnh này không làm thay đổi giá trị thanh ghi trừ thanh pc
+ 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

- **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
+ auipc (Add Upper Immediate to PC): Lệnh này lấy 20 bit cao của một giá trị
tức thời, dịch trái 12 bit (tương đương nhân với 4096), và cộng kết quả với giá trị hiện tại của Program Counter (PC). Kết quả được lưu vào thanh ghi đích.
    - Công thức: rd = PC + (immediate << 12)
    - auipc được sử dụng để tính toán một địa chỉ gần với địa chỉ của nhãn.

+ addi (Add Immediate) / lw (Load Word): Lệnh thứ hai được sử dụng để điều
chỉnh giá trị trong thanh ghi đích để đạt được địa chỉ chính xác của nhãn.
    - addi: Thường được sử dụng nếu khoảng cách giữa PC (sau khi thực hiện auipc) và địa chỉ của nhãn đủ nhỏ (nằm trong khoảng -2048 đến +2047 byte). addi cộng một giá trị tức thời 12-bit (có dấu) với thanh ghi nguồn.
    - lw: Đôi khi được sử dụng nếu la đang được sử dụng để nạp địa chỉ

của một biến trong bộ nhớ, và lệnh tiếp theo là truy cập vào biến đó. Trong trường hợp này, lw sẽ nạp giá trị từ địa chỉ được tính toán (bởi auipc và phần bù) vào thanh ghi đích.