

Lê Thành An

20235631

Thực hành kiến trúc máy tính

Báo cáo thực hành

Bài 2. Tập lệnh, các lệnh cơ bản, các chỉ thị biên dịch

Assignment 1: Lệnh gán số nguyên nhỏ 12-bit

Code:

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 sử dụng số

hiệu thanh ghi thay cho tên thanh ghi

Khởi đầu:

s0	8	0x00000000
pc		0x00400000

Chạy bước 1:

s0	8	0x00000512
pc		0x00400004

Chạy bước 2:

s0	8	0x00000000
pc		0x00400008

Chạy bước 3:

s0	8	0x00000000
pc		0x0040000c

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x51200413	addi x8,x0,0x00000512	2: addi s0, zero, 0x512

Mã máy 0x51200413 trong hệ nhị phân là 0101 0001 0010 0000 0000 0100 0001 0011.

Chia mã này ra:

- funct7: 0101000 (7 bits đầu tiên)
- rs1: 00010 (5 bits tiếp theo, tức là 2 trong thập phân)
- funct3: 000 (3 bits tiếp theo)
- rd: 00101 (5 bits tiếp theo, tức là 5 trong thập phân)
- opcode: 0010011 (7 bits cuối cùng)

Dựa trên các trường này, ta có:

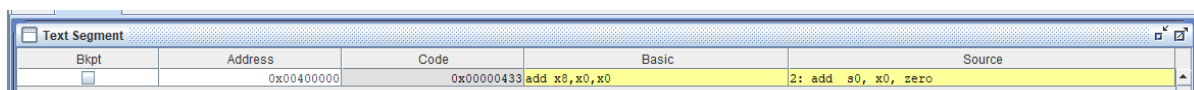
- funct7: 0101000 có thể tương ứng với lệnh ADD hoặc SUB tùy thuộc vào các bit khác.
- rs1: 2 (tức là x2)
- rd: 5 (tức là x5)
- funct3: 000 tương ứng với lệnh ADD.

Nhận xét: Mã máy của lệnh đúng với khuôn dạng lệnh đã qui định.

Sự thay đổi của thanh s0 và pc:

Bước	Thanh s0	Thanh pc
Khởi đầu	0x00000000	0x00400000
1	0x00000512	0x00400004
2	0x00000512	0x00400008

Lệnh: add s0, x0, zero



Bkpt	Address	Code	Basic	Source
	0x00400000	0x00000433	add x8, x0, x0	2: add s0, x0, zero

Mã máy 0x00000413 trong hệ nhị phân là 0000 0000 0000 0000 0000 0100 0011 0011.

Phân tích mã máy:

Opcode (7 bit): 0110011 (Lệnh R-type)

funct7 (7 bit): 0000000 (Không có tác dụng đặc biệt trong lệnh này)

rs2 (5 bit): 00000 (Thanh ghi nguồn thứ hai x0)

rs1 (5 bit): 00000 (Thanh ghi nguồn thứ nhất x0)

funct3 (3 bit): 000 (Lệnh ADD)

rd (5 bit): 00000 (Thanh ghi đích x0)

Nhận xét: Mã máy của lệnh đúng với khuôn dạng lệnh đã qui định.

```
1 .text
2 addi s0, zero, 0x20232024
3 add s0, x0, zero
```

Khi chạy đoạn lệnh sau thì xuất hiện lỗi sau:

```
Error in C:\Users\COMPUTER\Desktop\THKTM\week2.1.asm line 3 column 16: "0x20232024": operand is out of range
Assemble: operation completed with errors.
```

Lệnh addi thuộc khuôn dạng lệnh I, sử dụng 12 bits để biểu diễn số nguyên có dấu (imm[11:0]). Do đó lệnh addi chỉ có thể sử dụng để gán số nguyên có dấu trong phạm vi 12 bits (từ -2048 đến 2047).

Assignment 2: Lệnh gán số 32-bit

Code:

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

Khởi đầu:

s0	8	0x00000000
pc		0x00400000

Chạy bước 1:

s0	8	0x20232000
----	---	------------

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

Chạy bước 2:

s0	8	0x20232024
----	---	------------

pc		0x00400008
----	--	------------

Chạy bước 3:

s0	8	0x20232024
----	---	------------

pc		0x0040000c
----	--	------------

Edit Execute	
Text Segment	
Bkpt	Address Code Basic Source
<input type="checkbox"/>	0x00400000 0x20232437 lui x8,0x00020232 4: lui s0, 0x20232
<input type="checkbox"/>	0x00400004 0x02440413 addi x8,x8,0x00000024 6: addi s0, s0, 0x024 # s0 = s0 + 0x024

Data Segment									
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)	
0x00400000	0x20232437	0x02440413	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x00400020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x00400040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x00400060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x00400080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x004000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x004000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x004000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x00400100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	
0x00400120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	

← →

0x00400000 (.text)

☒ Hexadecimal Addresses ☒ Hexadecimal Values ☐ ASCII

Nhận xét: Dữ liệu từ Text Segment và Data Segment trùng nhau.

Assignment 3: Lệnh gán (giả lệnh)

Code:

Laboratory Exercise 2, Assignment 3

.text

li s0, 0x20232024

li s0, 0x20

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20232437	lui x8,0x00020232	3: li s0, 0x20232024
<input type="checkbox"/>	0x00400004	0x02440413	addi x8,x8,0x00000024	
<input type="checkbox"/>	0x00400008	0x02000413	addi x8,x0,0x00000020	4: li s0, 0x20

- Lệnh li không xuất hiện trực tiếp trong cột Basic vì li là một lệnh giả (pseudo-instruction), RARS sẽ thay thế li bằng 1 hoặc nhiều lệnh thực để đạt chức năng tương tự.

- Ví dụ ở lệnh li s0, 0x20232024, vì 0x20232024 là một giá trị 32-bit lớn và không thể nạp trực tiếp nên phải sử dụng 2 lệnh lui x8, 0x00020232 và addi x8, x8, 0x00000024.

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

Code:

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

Giá trị các thanh ghi:

t1	6	0x00000005
t2	7	0xffffffff
s0	8	0x0000000a
s0	8	0x00000009

Đáp án là $2X + Y = 2*5 + (-1) = 9$ (đúng)

Text Segment					
Offset	Address	Code	Basic	Source	
4	0x00400000	addi x6,x0,5		4: addi t1, zero, 5 # X = t1 = ?	
5	0x00400004	addi x7,x0,0xffffffff		5: addi t2, zero, -1 # Y = t2 = ?	
7	0x00400008	add x8,x6,x6		7: add s0, t1, t1 # s0 = t1 + t1 = X + X = 2X	
8	0x0040000c	add x8,x8,x7		8: add s0, s0, t2 # s0 = s0 + t2 = 2X + Y	

- Kiểm nghiệm:

+ addi x6, x0, 5: $x6 = x0 + 5 = 5$ (đúng)

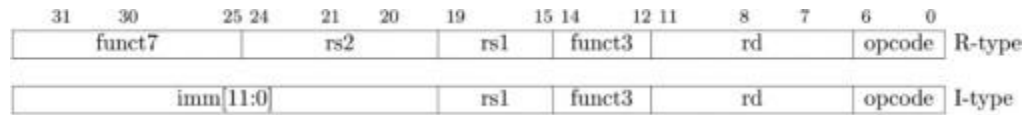
+ addi x7, x0, 0xffffffff: $x7 = x0 + 0xffffffff = -1$ (đúng)

+ add x8, x6, x6: $x8 = x6 + x6 = 0x0000000a = 10$ (đúng)

+ add x8, x8, x7: $x8 = x8 + x7 = 10 - 1 = 9$ (đúng)

=> Đúng với I-type và R-type:

add t1,t2,t3	Addition: set t1 to (t2 plus t3)
addi t1,t2,-100	Addition immediate: set t1 to (t2 plus signed 12-bit immediate)



Register-type:

- funct7 xác định chính xác loại lệnh
- funct3 xác định phép toán tử
- rs1, rs2: chỉ định thanh ghi chứa toán hạng thứ 1 và thứ 2
- rd: thanh ghi đích (nơi in ra kết quả)
- opcode: xác định loại lệnh

Immediate-type:

- imm[11:0]: immediate, một hằng số 12-bit, sử dụng như 1 toán hạng.
- rs1: chỉ định thanh ghi chứa toán hạng thứ 1.
- funct3: xác định thao tác cụ thể.
- rd: thanh ghi đích (nơi in ra kết quả)
- opcode: xác định loại lệnh.

Assignment 5: Phép nhân

Code:

```
# 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
```

Lệnh 1: Gán $t1 = 0 + 4 = 4$

Lệnh 2: Gán $t2 = 0 + 5 = 5$

Lệnh 3: Gán $s1 = t1 * t2 = 20$

Kết quả ở thanh ghi hiển thị chính xác

Khởi đầu:

t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000

Chạy bước 1:

t1	6	0x00000004
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000

Chạy bước 2:

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

Chạy bước 3:

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

Chạy bước 4:

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

Phép chia:

```
.text  
addi s1, zero, 20  
addi t1, zero, 5  
div t2, s1, t1
```

Lệnh 1: Gán $s1 = 0 + 20 = 20$

Lệnh 2: Gán $t1 = 0 + 5 = 5$

Lệnh 3: Gán $t2 = s1/t1 = 4$

Kết quả ở thanh ghi hiển thị chính xác

Khởi đầu:

t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000

Chạy bước 1:

t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000014

Chạy bước 2:

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

Chạy bước 3:

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

Chạy bước 4:

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

Assignment 6: Tạo biến và truy cập biến

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 v

Lệnh la (load address) không phải 1 lệnh của tập lệnh mà là 1 lệnh giả (pseudo-instruction), nó sẽ được biên dịch thành 2 lệnh auipc và addi.

Ví dụ ở lệnh la t5, X sẽ biên dịch thành 2 lệnh:

- auipc x30, 0x0000fc10 : Gán x30 = địa chỉ pc (0x00400000) + (0x0000fc10 << 12) (do auipc chỉ lấy giá trị 20 bit cao)

- addi x30, x30, 0 : Cộng thêm vào x30 = địa chỉ của x (0x10010000) – x30 hiện tại

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x0fc10f17	auipc x30,0x0000fc10	8: la t5, X # L?y ??a ch? c?a X trong vùng nh? ch?a d? li?u
<input type="checkbox"/>	0x00400004	0x000f0f13	addi x30,x30,0	
<input type="checkbox"/>	0x00400008	0x0fc10f97	auipc x31,0x0000fc10	9: la t6, Y # L?y ??a ch? c?a Y
<input type="checkbox"/>	0x0040000c	0xffcf8f93	addi x31,x31,0xfffffff3	

Địa chỉ và giá trị của X, Y, Z:

t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
t4	29	0x00000000
t5	30	0x10010000
t6	31	0x00000000
pc		0x00400004

Chạy bước 3:

t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
t4	29	0x00000000
t5	30	0x10010000
t6	31	0x10010008
pc		0x0040000c

Chạy bước 4:

t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
t4	29	0x00000000
t5	30	0x10010000
t6	31	0x10010004
pc		0x00400010

Chạy bước 5:

t1	6	0x00000005
t2	7	0x00000000
s0	8	0x00000000
t4	29	0x00000000
t5	30	0x10010000
t6	31	0x10010004
pc		0x00400014

Chạy bước 6:

t1	6	0x00000005
t2	7	0xffffffff
s0	8	0x00000000

t4	29	0x00000000
t5	30	0x10010000
t6	31	0x10010004
pc		0x00400018

Chạy bước 7:

t1	6	0x00000005
t2	7	0xffffffff
s0	8	0x0000000a

t4	29	0x00000000
t5	30	0x10010000
t6	31	0x10010004
pc		0x0040001c

Chạy bước 8:

t1	6	0x00000005
t2	7	0xffffffff
s0	8	0x00000009
t4	29	0x00000000
t5	30	0x10010000
t6	31	0x10010004
pc		0x00400020

Chạy bước 9:

t1	6	0x00000005
t2	7	0xffffffff
s0	8	0x00000009
t4	29	0x10010020
t5	30	0x10010000
t6	31	0x10010004
pc		0x00400024

Chạy bước 10:

t1	6	0x00000005
t2	7	0xffffffff
s0	8	0x00000009
t4	29	0x10010008
t5	30	0x10010000
t6	31	0x10010004
pc		0x00400028

Chạy bước 11:

t1	6	0x00000005
t2	7	0xffffffff
s0	8	0x00000009
t4	29	0x10010008
t5	30	0x10010000
t6	31	0x10010004
pc		0x0040002c

Chạy bước 12:

t1	6	0x00000005
t2	7	0xffffffff
s0	8	0x00000009
t4	29	0x10010008
t5	30	0x10010000
t6	31	0x10010004
pc		0x00400030

- Lệnh lw t1, 0(t5) lấy giá trị từ địa chỉ t5 lưu vào t1
- Lệnh sw s0, 0(t4) lấy giá trị từ s0 lưu vào địa chỉ mà t4 trỏ đến
- Các lệnh lb, sb tương tự lw, sw nhưng chỉ lấy giá trị 1 byte