Laboratory Exercise 2 – Reports

Instruction Set, Basic Instructions, Directives

Trần Khánh Quỳnh – 20225762

1. Home assignment 1:

- Tên và ý nghĩa của 32 thanh ghi

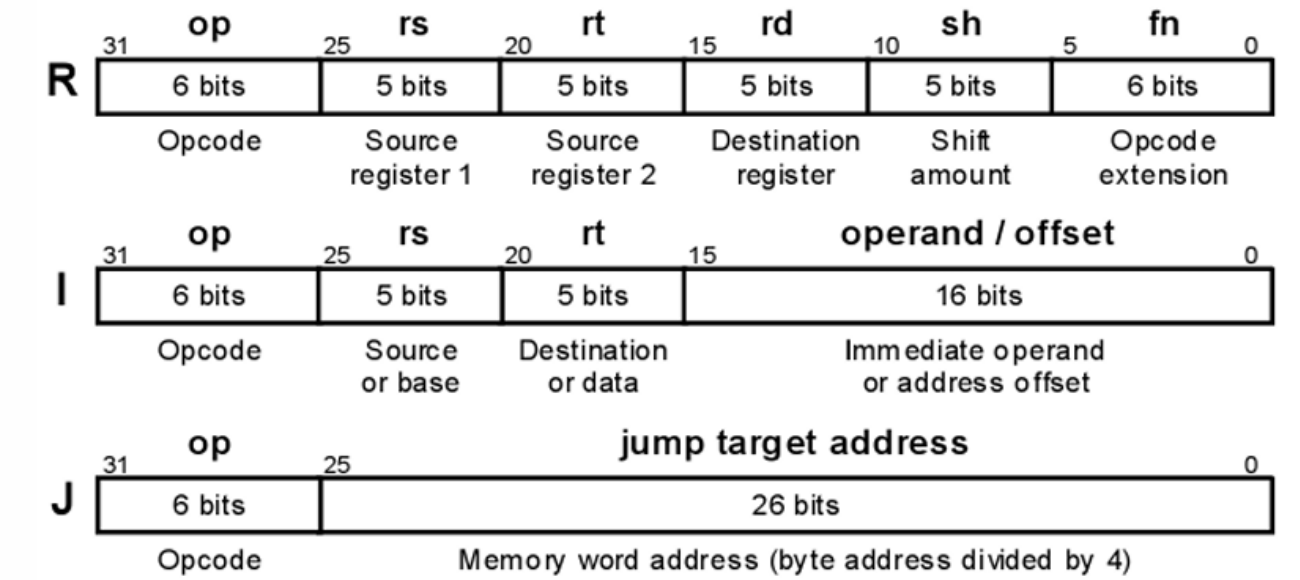
Tên thanh ghi	**Số**	**Ý nghĩa**
$zero	$0	Hằng số 0
$at	$1	Assembly temporary – giá trị tạm thời cho hợp ngữ
$v0 - $v1	$2 - $3	Giá trị trả về của hàm hoặc biểu thức
$a0 - $a3	$4 - $7	Tham số của hàm hoặc biểu thức
$t0 - $t7	$8 - $15	Thanh ghi tạm thời (không giữ giá trị trong quá trình gọi hàm)
$s0 - $s7	$16 - $23	Thanh ghi lưu trữ (giữ giá trị trong suốt quá trình gọi hàm)
$t8 - $t9	$24 - $25	Thanh ghi tạm thời
$k0 - $k1	$26 - $27	Dự trữ cho nhân OS
$gp	$28	Con trỏ toàn cục
$sp	$29	Con trỏ stack
$fp	$30	Con trỏ frame
$ra	$31	Địa chỉ trả về

-Các thanh ghi đặc biệt PC, HI, LO:

- Thanh ghi PC (Program Counter) ghi địa chỉ của lệnh sẽ được nạp vào từ bộ nhớ. Sau khi lệnh được nhận vào, nội dung thanh ghi PC sẽ tăng lên 4 để trỏ đến lệnh kế tiếp.

- Thanh ghi HI, LO: không phải thanh ghi đa năng. Kết quả của thao tác nhân sẽ được lưu trong thanh ghi này, với bit 0 – 31 là LO và 32 – 63 là LI. Đối với phép chia thì HI sẽ lưu thương và LO sẽ lưu phần dư của phép chia đó.

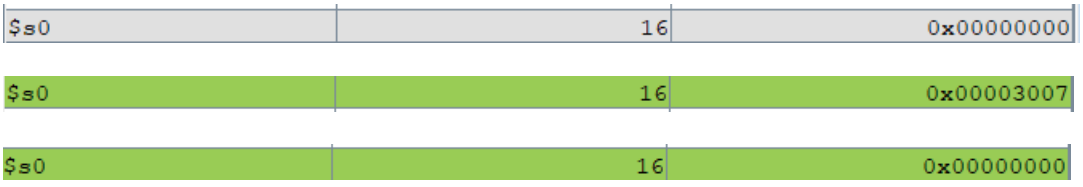
-Khuôn dạng 3 loại lệnh I, J, R



2. Home assignment 2

3. Assignment 1: Lệnh gán số 16-bit

-Sự thay đổi của thanh ghi $s0

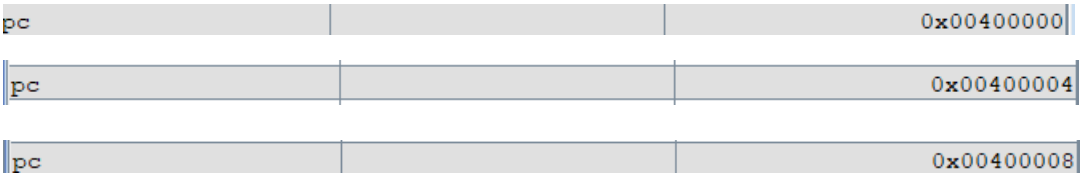


Nhận xét:

Thanh ghi $s0 chuyển từ 0x00000000 sang 0x00003007 sau đó lại trở về 0x00000000

Giải thích:

-Sự thay đổi của thanh ghi pc:



Nhận xét:

Thanh ghi $pc tăng lên 4 đơn vị sau mỗi câu lệnh: 0x00400000 -> 0x00400004 -> 0x00400008

Giải thích:

pc thay đổi để chuẩn bị nhận vào lệnh tiếp, mà độ dài lệnh trong kiến trúc MIPS là 4 bytes

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

`addi  $s0, $zero, 0x3007`

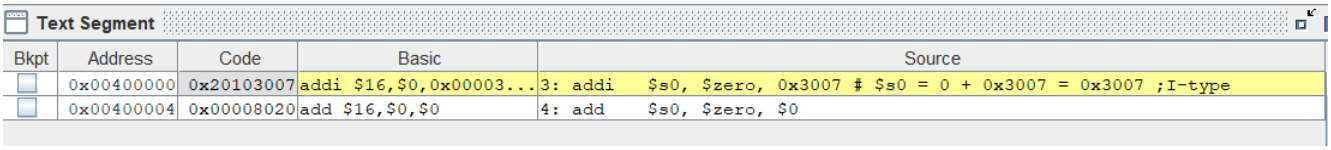
- Lệnh theo khuôn dạng I
- addi: 001000
- $s0: thanh số 16: 10000
- $zero: thanh đầu tiên: 00000
- 0x3007: 0011 0000 0000 0111

➔ Mã máy: 001000 | 00000 | 10000 | 0011 0000 0000 0111 = 0x20103007

`add  $s0, $zero, $0`

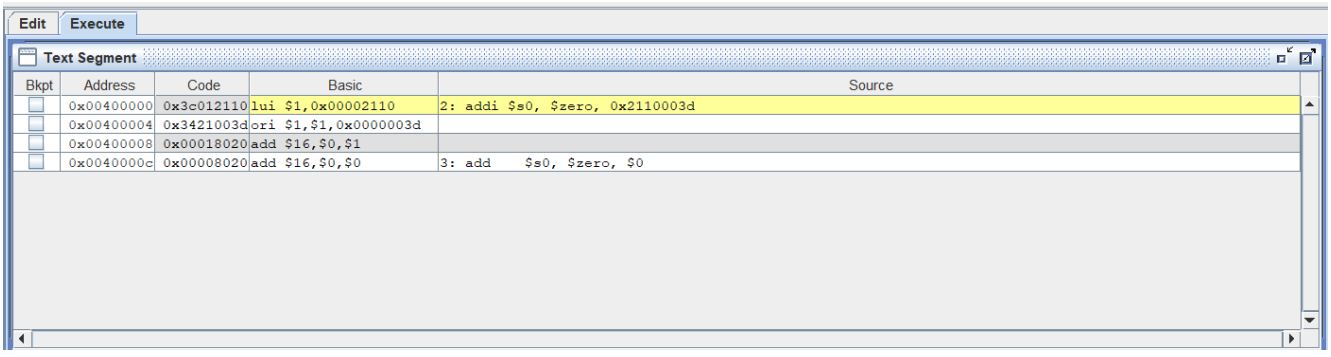
- Lệnh theo khuôn dạng R
- add: 000000
- $s0: thanh số 16: 10000 (Destination register)
- $zero: thanh đầu tiên: 00000 (Source register 1)
- $0: 00000 (Source register 2)

➔ Mã máy: 000000 | 00000 | 00000 | 10000 | 00000 | 10 0000 = 0x00008020



Kết luận: Các lệnh đúng như tập lệnh đã quy định

-Sửa lại lệnh addi: addi $s0, $zero, 0x2110003d



Nhận xét: thay vì hiện ra lệnh addi thì lệnh được chia ra làm 2 lệnh lui và ori, do lệnh theo khuôn dạng I cho phép tối đa 16 bit, mà ở đây là phép cộng với hằng số nguyên 32 bit (0x2110003d)

Lệnh lui: copy các bit 16-31 của hằng số vào $at làm trung gian

Name	Number	Value
$zero	0	0x00000000
$at	1	0x21100000

Lệnh ori: OR $1 và 0x0000003d rồi lưu vào $1

Name	Number	Value
$zero	0	0x00000000
$at	1	0x2110003d

Sau khi chạy xong lệnh lui và ori thì thực hiện lệnh add: cộng $0 ($zero) với $1 rồi lưu vào $16 (hay $s0). Tiếp theo đó chạy nốt lệnh add $s0, $zero, $0

4. **Assignment 2: Lệnh gán số 32-bit**

-Sự thay đổi của thanh ghi $s0

$s0	16	0x00000000
$s0	16	0x21100000
$s0	16	0x2110003d

Nhận xét: Thanh ghi $s0 chuyển từ 0x0000000 sang 0x21100000 rồi đến 0x2110003d
Lệnh lui copy nửa trên của hằng số vào $s0, vì vậy $s0 chuyển thành 0x21100000
Lệnh ori copy nửa dưới của hằng số vào $s0, vì vật $s0 chuyển thành 0x2110003d

-Sự thay đổi của thanh ghi pc

pc		0x00400000
pc		0x00400004
pc		0x00400008

Nhận xét: Sau mỗi một lệnh thì pc tăng lên 4, do mỗi lệnh trong MIPS có độ dài 4 bytes.

-Byte đầu tiên của vùng lệnh trong data segment trùng với cột code của các lệnh trong text segment.

5. **Assignment 3: Lệnh gán (giả lệnh)**

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c012110	lui $1,0x00002110	3: li $s0,0x2110003d #pseudo instruction=2 basic instructions
	0x00400004	0x3430003d	ori $16,$1,0x0000003d	
	0x00400008	0x24110002	addiu $17,$0,0x0000...	4: li $s1,0x2

-Dùng cùng giả lệnh li nhưng lệnh đầu được thực hiện bằng lui và ori, còn lệnh sau thì có thể thực hiện luôn mà không bị tách ra

Giải thích: 0x2110003d lớn hơn 16 bits, mà lại có khuôn dạng I nên phải chia ra thành lui và ori; còn 0x2 nhỏ hơn 16 bit nên có thể thực hiện ngay qua addiu

6. **Assignment 4: Tính biểu thức 2x + y**

-Thay đổi của cửa sổ Register

Nhận xét: đầu tiên $t1 đổi thành 0x00000005 sau khi chạy xong lệnh addi đầu tiên. Tiếp theo $t2 đổi thành 0xffffffff sau khi chạy xong lệnh addi thứ 2.
Với lệnh add đầu tiên, $s0 chuyển thành 0x0000000a, còn sau khi chạy nốt lệnh add thứ 2 thừ $s0 chuyển thành 0x00000009.

Sau khi kết thúc chương trình thì kết quả đúng: do $t1 có giá trị bằng 5, sau lệnh add đầu tiên lưu 2 lần giá trị của $t1 nên $s0 bằng 10 hay 0x0000000a. Với lệnh add thứ 2, $s0 lưu tổng của $s0 và $t2, mà $t2 có giá trị bằng -1 hay 0xffffffff. Vì vậy, kết quả cuối cùng lưu ở $s0 là 9 (2x + y = 2 x 5 + -1 = 9)

-Lệnh addi

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

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20090005	addi $9,$0,0x00000005	3: addi $t1, $zero, 5 # X = $t1 = ?
	0x00400004	0x200affff	addi $10,$0,0xfffff...	4: addi $t2, $zero, -1
	0x00400008	0x01298020	add $16,$9,$9	7: add $s0, $t1, $t1 # $s0 = $t1 + $t1 = X + X = 2X
	0x0040000c	0x020a8020	add $16,$16,$10	8: add $s0, $s0, $t2 # $s0 = $s0 + $t2 = 2X + Y

0x20090005 = 0010 00 | 00 000 | 0 1001 | 0000 0000 0000 0101
Theo khuôn mẫu của lệnh I:

- Opcode: 001000 = 8
- rs: 00000 ($zero)
- rt: 01001 ($9 hay $t1)
- imm :0101 = 5

➔ Lệnh tuân theo đúng khuôn mẫu của kiểu lệnh I

```
addi  $t2, $zero, -1
```

0x200afff = 0010 00| 00 000 | 0 1010 | 1111 1111 1111

Theo khuôn mẫu của lệnh I:

- Opcode: 001000 = 8
- rs: 00000 ($zero)
- rt: 01010 ($10 hay $t2)
- imm: 1111 1111 1111 = -1

➔ Lệnh tuân theo đúng khuôn mẫu của kiểu lệnh I

-Lệnh add

```
add   $s0, $t1, $t1
```

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20090005	addi $9,$0,0x00000005	3: addi $t1, $zero, 5 # X = $t1 = ?
	0x00400004	0x200affff	addi $10,$0,0xffffff...	4: addi $t2, $zero, -1
	0x00400008	0x01298020	add $16,$9,$9	7: add $s0, $t1, $t1 # $s0 = $t1 + $t1 = X + X = 2X
	0x0040000c	0x020a8020	add $16,$16,$10	8: add $s0, $s0, $t2 # $s0 = $s0 + $t2 = 2X + Y

0x01298020 = 0000 00 | 01 001 | 0 1001 | 1000 0 | 000 00 | 10 0000

Theo khuôn mẫu của lệnh R:

- opcode: 000000 = 0
- rs: 01001 ($t1 hay $9)
- rt: 01001 ($t1 hay $9)
- rd: 10000 ($s0 hay $16)
- sh: 00000
- fn: 100000 = 32 → lệnh add

➔ Lệnh tuân theo đúng khuôn mẫu kiểu lệnh R

```
add   $s0, $s0, $t2
```

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20090005	addi $9,$0,0x00000005	3: addi $t1, $zero, 5 # X = $t1 = ?
	0x00400004	0x200affff	addi $10,$0,0xffffff...	4: addi $t2, $zero, -1
	0x00400008	0x01298020	add $16,$9,$9	7: add $s0, $t1, $t1 # $s0 = $t1 + $t1 = X + X = 2X
	0x0040000c	0x020a8020	add $16,$16,$10	8: add $s0, $s0, $t2 # $s0 = $s0 + $t2 = 2X + Y

0x020a8020 = 0000 00| 10 000 |0 1010 | 1000 0 |000 00 | 10 0000

Theo khuôn mẫu của lệnh R:

- opcode: 000000 = 0
- rs: 10000 ($s0 hay $16)
- rt: 01010 ($t2 hay $10)
- rd: 10000 ($s0 hay $16)
- sh: 00000
- fn: 100000 = 32 → lệnh add

➔ Lệnh tuân theo đúng khuôn mẫu kiểu lệnh R

7. Assignment 5: Phép nhân

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20090004	addi $9,$0,0x00000004	5: addi $t1, $zero, 4 # X = $t1 = ?
	0x00400004	0x200a0005	addi $10,$0,0x00000005	6: addi $t2, $zero, 5 # Y = $t2 = ?
	0x00400008	0x712a8002	mul $16,$9,$10	8: mul $s0, $t1, $t2 # HI-LO = $t1 * $t2 = X * Y ; $s0 = LO
	0x0040000c	0x20010003	addi $1,$0,0x00000003	9: mul $s0, $s0, 3 # $s0 = $s0 * 3 = 3 * X * Y
	0x00400010	0x72018002	mul $16,$16,$1	
	0x00400014	0x00008812	mflo $17	11: mflo $s1

Nhận xét: Trước khi thực hiện lệnh mul thứ 2 thì xuất hiện thêm 1 lệnh addi. Mul chỉ thao tác với các thanh ghi nên gán 3 vào thanh ghi $1 (hay $at).

-Sự thay đổi của thanh ghi HI, LO:

+2 lệnh addi đầu tiên gán giá trị 4 và 5 vào thanh ghi $9 và $10, vì vậy thanh HI và LO vẫn là 0x00000000

+Lệnh mul đầu tiên tính toán giá trị của phép nhân, gán vào thanh ghi $s0, đồng thời cũng gán vào HI và LO (4 x 5 = 20 = 0x14 → thanh $s0 có giá trị 0x00000014 và thanh LO chuyển thành 0x00000014)

+Lệnh addi thứ 3 lưu tạm giá trị 3 vào thanh ghi $at

+Lệnh mul thứ 2 thực hiện phép nhân của giá trị trong thanh ghi $s0 và 3 (được lưu tạm trong $at). Kết quả được lưu vào $s0 nhưng cũng đồng thời lưu vào LO: 0x0000003c

+Lệnh mflo gán kết quả từ LO vào $s1

→Kết quả đúng

8. Assignment 6: Tạo biến và truy cập biến

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x00001001	10: la $t8, X # Get the address of X in Data Segment
	0x00400004	0x34380000	ori $24,$1,0x00000000	
	0x00400008	0x3c011001	lui $1,0x00001001	11: la $t9, Y # Get the address of Y in Data Segment
	0x0040000c	0x34390004	ori $25,$1,0x00000004	
	0x00400010	0x8f090000	lw $9,0x00000000($24)	12: lw $t1, 0($t8) # $t1 = X
	0x00400014	0x8f2a0000	lw $10,0x00000000($...	13: lw $t2, 0($t9)
	0x00400018	0x01298020	add $16,$9,$9	16: add $s0, $t1, $t1 # $s0 = $t1 + $t1 = X + X = 2X
	0x0040001c	0x020a8020	add $16,$16,$10	17: add $s0, $s0, $t2 # $s0 = $s0 + $t2 = 2X + Y
	0x00400020	0x3c011001	lui $1,0x00001001	19: la $t7, Z # Get the address of Z in Data Segment
	0x00400024	0x342f0008	ori $15,$1,0x00000008	
	0x00400028	0xadf00000	sw $16,0x00000000($...	20: sw $s0, 0($t7) # Z = $s0 = 2X + Y

-Lệnh la được biên dịch bởi hai lệnh lui và ori

-Cửa sổ labels:

+ Địa chỉ của 3 biến X, Y, Z không thay đổi trong quá trình biên dịch và cách nhau lần lượt 4

+ Các địa chỉ của các biến đều đúng với các giá trị gán cho chúng

-Lệnh lw và sw

+ Lệnh `lw $t1, 0($t8)`: đọc giá trị từ địa chỉ 0($t8) và lưu vào trong thanh ghi $t1

+Lệnh `sw $s0, 0($t7)`: lấy giá trị từ thanh ghi có địa chỉ $s0 để đưa vào bộ nhớ 0($t7)

+Tương tự với hai lệnh còn lại

→ Lệnh lw: tải dữ liệu từ bộ nhớ vào thanh ghi; lệnh sw: lưu dữ liệu từ thanh ghi vào bộ nhớ