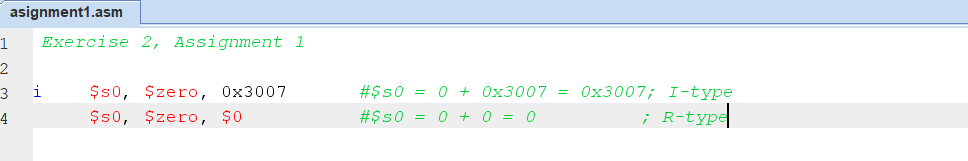
**Báo cáo thực hành tuần 2**

Sinh viên: Kiều Đăng Nam

MSSV: 20176830

**Asignment 1: lệnh gán số 16 bit**



1. Quan sát sự thay đổi thanh ghi

* Thanh ghi $s0 :







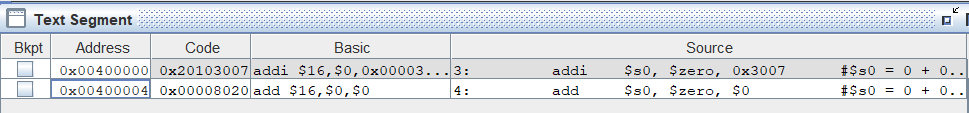
* Thanh ghi $pc :







1. Ở cửa sổ Text Segment, so sánh mã máy của các lệnh với khuôn dạng lệnh:



* Mã code của lệnh 1 sau khi chuyển sang nhị phân: 00100000|00000|10000|0011000000000111

Tương ứng với op = 8|$0|$16|0x00003007 🡪 Đúng như tập lệnh I quy định

* Mã code của lệnh 2 sau khi chuyển sang nhị phân:

000000 00000 00000 10000 00000 100000

Tương ứng với op = 0|$0|$0|$16||function = 32(add) 🡪 Đúng như tập lệnh R quy định

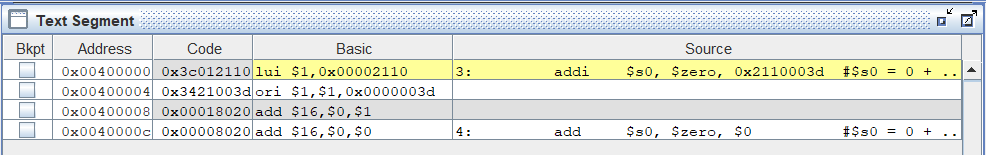
1. Sau khi sửa lại lệnh lui:

* Hiện tượng:

lệnh này đã đc tách ra làm 2 lệnh

lui $s1, 0x00002110

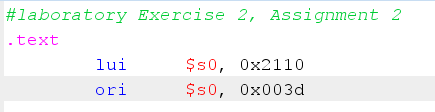
ori $s1, $s1, 0x0000003d



2 lệnh này tách số 0x2110003d làm 2 phần và gán vào $s1 sau đó lấy giá trị thanh ghi $s1 cộng với $0 kết quả thu được gán vào $s0 như bình thường

Giải thích : Khi thay thế bằng addi $s0, $zero, 0x2110003d thì do 0x2110003d lớn hơn 32 bit nên lệnh được chia làm 2 lệnh như trên.

**Asignment 2: lệnh gán số 32 bit**

****

1. Sự thay đổi giá trị của các thanh ghi:

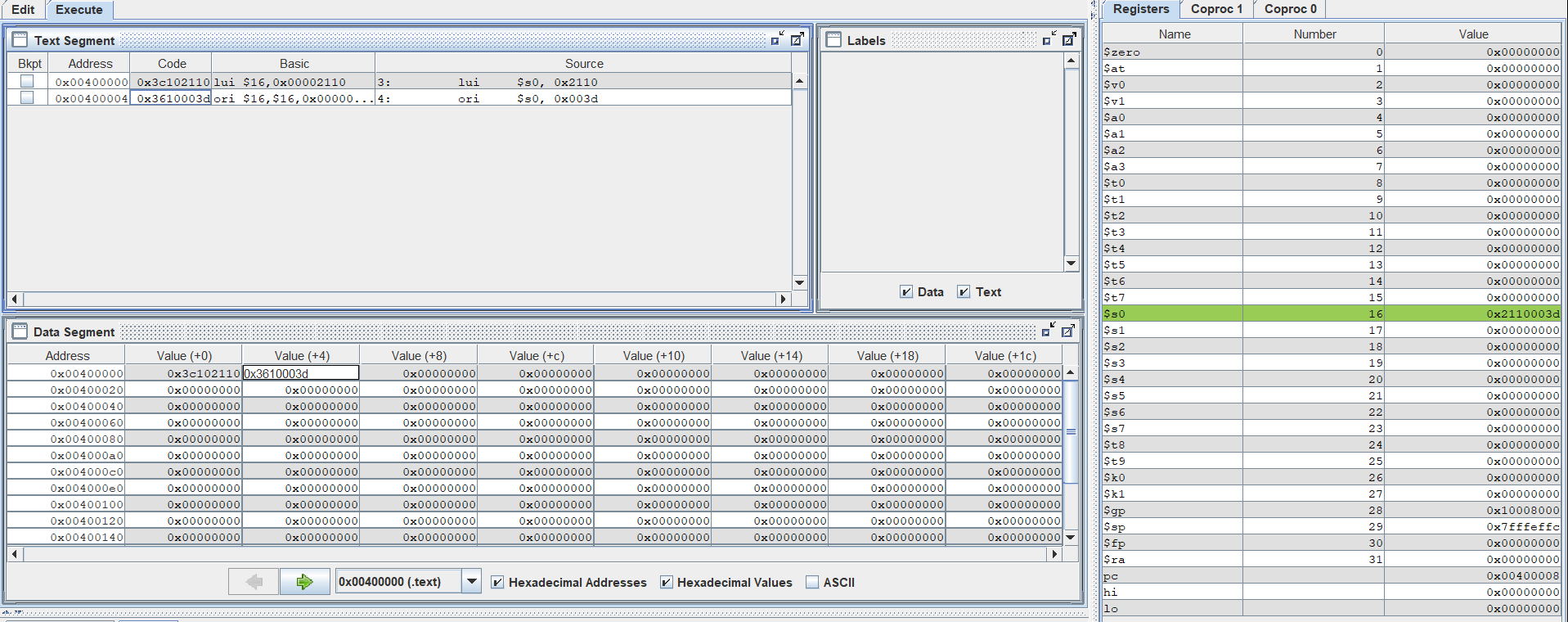
* Thanh ghi $s0: 0x0000000 -> 0x21100000 -> 0x2110003d
* Thanh ghi $pc: 0x0400000 -> 0x04000004 -> 0x04000008

1. Trong cửa sổ Data Segmeent, click vào hộp combo chuyển tới quan sát các byte trong vùng lện .text. Nhận xét:

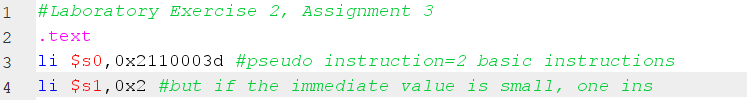
* Các byte đầu tiên ở vùng lệnh .text sẽ trùng với cột Code trong trong cửa sổ Text Segment.
* Cụ thể:

Tại address = 0x040000000 là địa chỉ của lệnh đầu tiên -> tương ứng giá trị đầu ( value (+0) ) sẽ là 0x3c102110 – trùng với mã code (cột Code trong Text Segment) của lệnh đầu tiên dung để load upper immediate – ghi hằng số 16 bit (2110) vào 2 byte cao của thanh ghi $s0, 2 byte thấp được gán = 0.

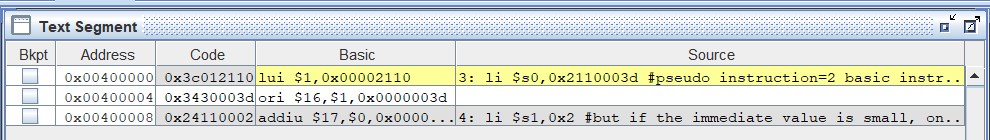
Khi address tăng lên 4 thì value(+4) = 0x361003d trùng với mã code của lệnh thứ 2 trong cửa sổ Text Segment để thực hiện lệnh ori.



**Asignment 3: lệnh gán (giả lệnh)**



Quan sát Text Segment:

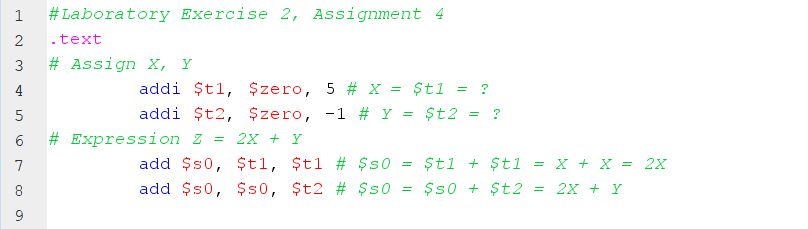


Nhận xét và giải thích:

* Lệnh li $s0, 0x2110003d được tách thành 2 lệnh:
  + lui $1, 0x00002110
  + ori $16, $s1, 0x0000003d
* Vì lệnh I của MIPS chỉ có tham số 16 bits, nên được tách thành 2 lệnh.

Lệnh li $s1, 0x2 có tham số nhỏ hơn 16 bits nên không cần tách

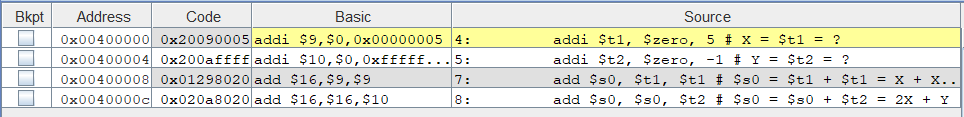
**Assignment 4: tính biểu thức 2x + y = ?**

****

1. Quan sát cửa sổ Registers

* Sau khi chạy lệnh đầu tiên thanh ghi $t1 được gán giá trị = 5
* Sau khi chạy lệnh thứ 2 thanh ghi $t2 được gán giá trị = -1
* Sau khi chạy lệnh thứ 3 thanh ghi $s0 được gán giá trị = 10 (do $s0 = $t1 + $t1, $t1 = 5)
* Sau khi chạy lệnh cuối cùng $s0 được gán giá trị = 9 🡪 Kết quả đúng

Cửa sổ Text Segment:



1. Mã máy lệnh addi: 0x20090005 có hợp ngữ là addi $9, $0, 5

Chuyển đổi mã máy sang nhị phân:001000|00000|01001|0000000000000101

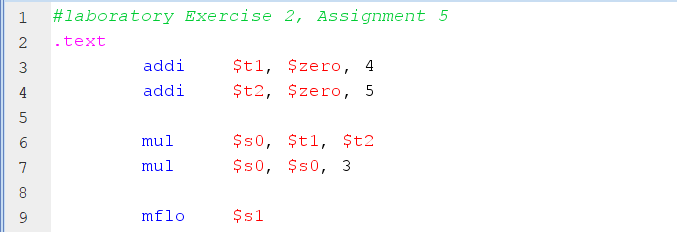
🡪Đúng với khuôn mẫu kiểu I

1. Mã máy lệnh add: 0x01298020

chuyển sang nhị phân -> 000000|01001|01001|10000|00000|100000

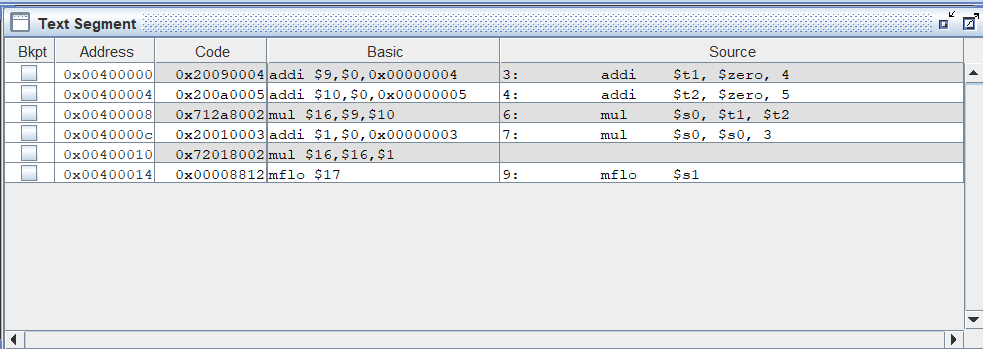
tương ứng với op = 0|$9|$9|$16||function(32) (add) 🡪 đúng với khuôn dạng kiểu R

**Asignment 5: Phép nhân**

****

1. Điều bất thường:

Các lệnh mã máy trong cửa sổ Text Segment:



Sự bất thường: cùng một lệnh mul ở dòng 6 và 7 nhưng lệnh mã máy lại khác nhau:



Giải thích:

* Lệnh mul ở dòng 6 thực hiện việc nhân giữa 2 thanh ghi nên lệnh mã máy trả về phép nhân <mul $16, $s9, S10>
* Lệnh mul ở dòng 7 thực hiện là nhân thanh ghi với một hằng số nên đầu tiên sẽ thực hiện lệnh addi để lưu giá trị hằng số vào thanh ghi thạm thời $at ($1) = $0 + 3. Sau đó mới thực hiện việc nhân giữa 2 thanh ghi

Kết của = $s0 \* $at

1. Debug, chạy từng lệnh và quan sát sự thay đổi giá trị của các thanh ghi

* Sự thay đổi giá trị của các thanh ghi:

+ Thanh ghi $s0: 0x0000000 -> (qua 2 phép gán giá trị cho $t1 và $t2) -> 0x00000014 (sau khi thực hiện phép nhân $t1 \* $t2 – 4\*5) -> 0x0000003c (sau khi thực hiện nhân $s0 \* $at = 20\*3).

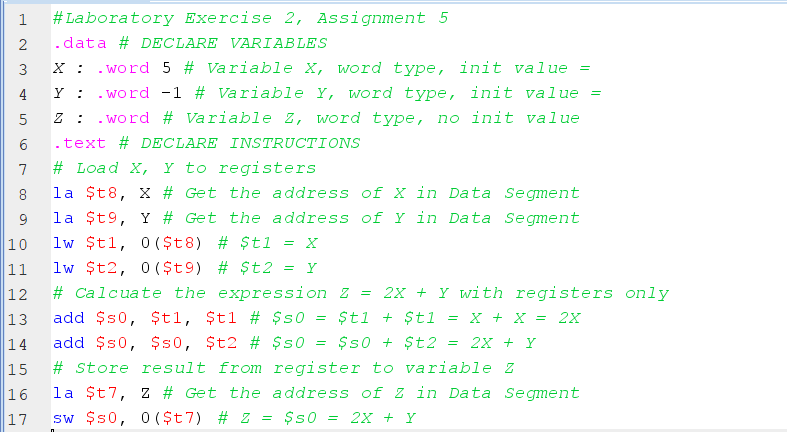
+ Thanh ghi lo : 0x00000000 -> (qua 2 phép gán giá trị cho $t1 và $t2) -> 0x00000014 (sau khi thực hiện phép nhân $t1 \* $t2 – 4\*5) -> 0x0000003c (sau khi thực hiện nhân $s0 \* $at = 20\*3).

+ Thanh ghi hi : không thay đổi do kết quả phép tính là số chỉ nằm trong phạm vi 32 bit (chỉ dung li để lưu kết quả).

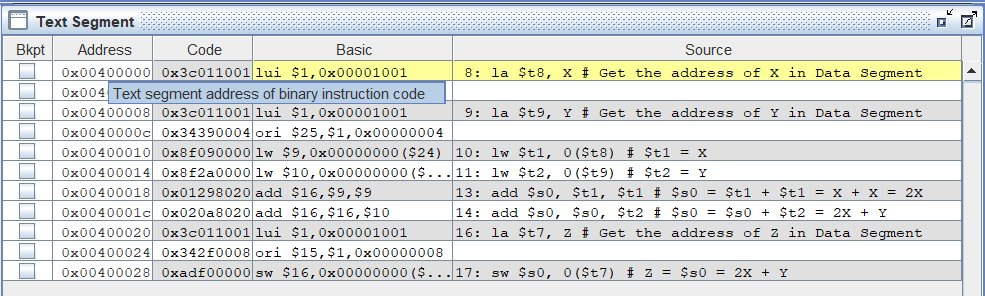
* Nhận xét kết quả: giá trị thanh ghi $s1 = 0x0000003c -> 60 (=3\*4\*5)

Kết quả là chính xác.

**Asignment 6: Tạo biến và truy cập biến**

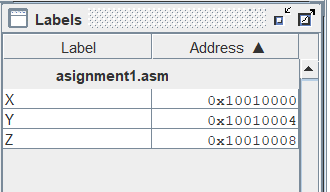
****

Cửa sổ Text Segment:



1. Lệnh la được tách thành 2 lệnh lui và ori để load địa chỉ của biến x vì địa chỉ biến x là 32 bits.

Cửa sổ Label:



1. Địa chỉ của x, y, z được tách thành 2 nửa

|  |  |  |  |
| --- | --- | --- | --- |
| Biến | Địa chỉ | Nửa trên | Nửa dưới |
| X | 0x10010000 | 0x00001001 | 0x00000000 |
| Y | 0x10010004 | 0x00001001 | 0x00000004 |
| Z | 0x10010008 | 0x00001001 | 0x00000008 |

1. Giá trị của x, y, z:

|  |  |
| --- | --- |
| Biến | Giá trị |
| X | 0x00000005 |
| Y | 0xffffffff |
| Z | 0x00000000 (không khởi tạo giá trị) |

1. Lệnh lw: load giá trị kiểu word từ thanh ghi $t8, $t9 vào thanh ghi $t1, $t2.
2. Lệnh sw: store giá trị kiểu word từ thanh ghi $s0 vào thanh ghi $t7.
3. Lệnh lb: load giá trị kiểu byte vào một thanh ghi.
4. Lệnh sb: store giá trị low-order 8 bits vào một thanh ghi.