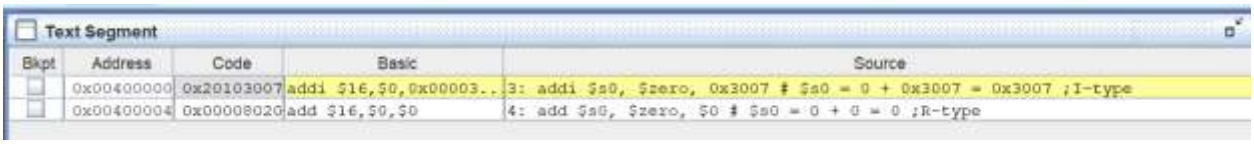
# Assignment 1: lệnh gán số 16-bit




Sau khi thực hiện dòng lệnh đầu tiên:

- Giá trị thanh $s0 thay đổi từ 0x00000000 thành 0x00003007 do dòng lệnh đầu tiên đã thay đổi giá trị thanh ghi $s0.

- giá trị pc thay đổi từ 0x0040000 thành 0x00400004 vì pc lưu địa chỉ dòng lệnh tiếp theo.

Sauk hi thực hiện câu lệnh 2

- Giá trị thanh $s0 thay đổi từ 0x00003007 thành 0x00000000 do dòng lệnh đầu tiên đã thay đổi giá trị thanh ghi $s0 thành 0.

- pc tiếp tục nhảy đến địa chỉ câu lệnh tiếp theo.

Kiểm tra mã máy:

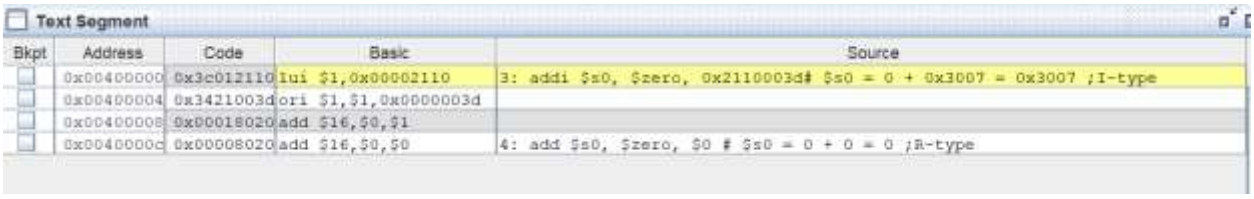
- addi $s0, $zero, 0x3007 :lệnh I , opcode 8

=> mã máy : 0010 0000 0001 0000 0011 0000 0000 0111-> 0x20103007 đúng với khuôn dạng

- add $s0, $zero, $0 : lệnh R opcode 0/32

=> mã máy : 0000 0000 0000 0000 1000 0000 0010 0000->0x00008020 đúng với khuôn dạng

Sửa lại chương trình:




Thay vì chạy 2 lệnh như cũ chương trình thực hiện 4 lệnh do con số mà chương trình phải thực hiện trong phép toán vượt quá mức biểu diễn của 16bit.

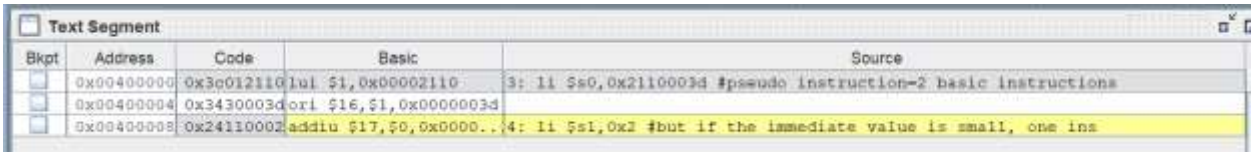
# Assignment 2: lệnh gán số 32-bit


Sau câu lệnh đầu tiên $s0 thay đổi từ 0x0000000 thành 0x2110000 do câu lệnh đầu tiên được sử dụng để nạp giá trị ngay lập tức vào 16 bit cao nhất của thanh ghi đích ở đây 0x2110 là giá trị được nạp vào 16 bit cao nhất.

Sau câu lệnh 2 $s0 thay đổi thành 0x2110003d do câu lệnh thực hiện phép toán Ỏ giữa trị cũ 0x21100000 và 0x003d ,kết quả là 0x2110003d lưu vào $s0.

Các byte đầu tiên ở vùng lệnh trùng với cột value(+0) và value(+4) của cửa sổ text segment.

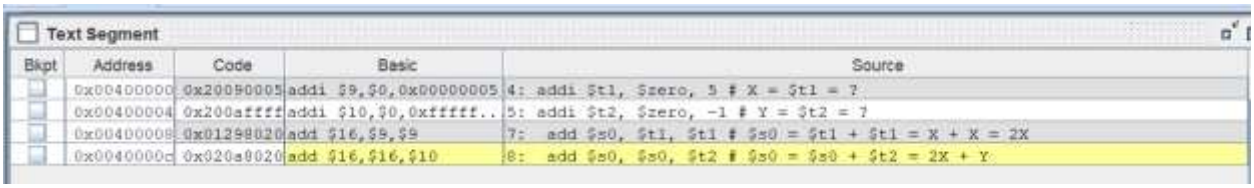
# Assignment 3: lệnh gán (giả lệnh)




Mặc dù lệnh ta soạn là ‘li’ nhưng khi thực hiện chương trình ,leenghj thực hiện lại lài ‘lui’,’ori’ và ‘addiu’,ddieuf này có thể giải thích như sau:

Lệnh ‘li’ là một lệnh giả mạo và không phải là một lệnh thực sự trong cấu trúc MIPS,khi biên soạn lệnh ‘li’ chương trình sẽ tự hiểu đó là lệnh gán ngay lập tức giá trị nào đó tới đích,tùy vào giá trị đó (16 bit hoặc 32 bit) mà chương trình sẽ đưa ra cách thức gán giá trị khác nhau. Cụ thể ở trên với giá trị 0x2110003d chương trình sử dụng cách thức gán 32 bit,còn giá trị 0x2 thì sử dụng ‘addiu’ cho số 16 bit.

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




Sau câu lệnh 1 giá trị thanh $t1 thay đổi từ 0x00000000 thành 0x00000005 do câu lệnh gán giá trị ‘5’ cho thanh ghi $t1

Sau câu lệnh 2 giá trị thanh $t2 thay đổi từ 0x00000000 thành 0xffffffff do câu lệnh gán giá trị ‘-1’ cho thanh ghi $t2

Sau câu lệnh 3 giá trị thanh $s0 thay đổi từ 0x00000000 thành 0x0000000a do câu lệnh thực hiện cộng giá trị $t1 với chính nó rồi lưu vào thanh ghi $s0

Sau câu lệnh 3 giá trị thanh $s0 thay đổi từ 0x00000000 thành 0x00000009 do câu lệnh thực hiện cộng giá trị $s0 với $t2 rồi lưu vào thanh ghi $s0

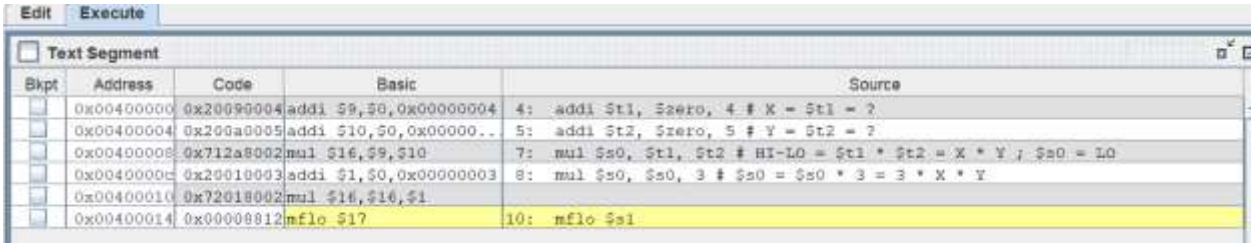
Đối với lệnh addi, hợp ngữ và mã máy có dạng như sau:

addi $t1, $zero, 5

0x20010005

=>mỗi lệnh addi sẽ có một mã máy duy nhất tương ứng với giá trị ngay lập tức và các thanh ghi cụ thể được sử dụng(0x0005 và 5)

# Assignment 5: phép nhân



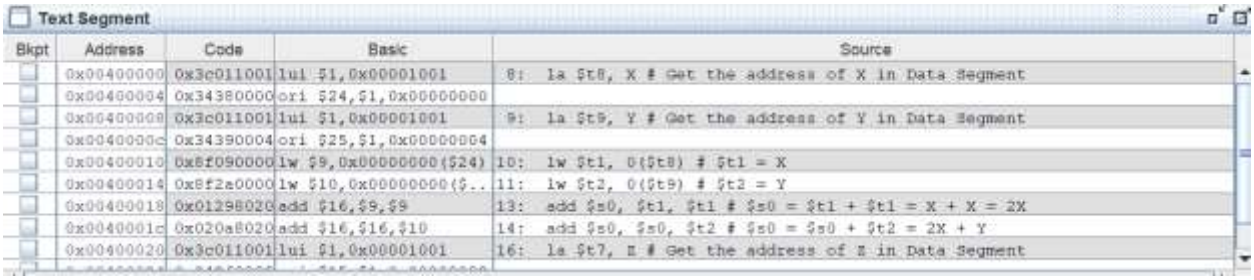

Do khi thực hiện phép nhân 2 số 32 bit sẽ ra một kết quả có độ dài tối đa là 64-bit, vậy nên máy sẽ lưu kết quả này vào 2 thanh ghi $hi và $lo.

Ta thấy điều bất thường là ở câu lệnh mul $s0, $s0, 3, khi nó được chuyển thành: addi $1, $0, 0x00000003 mul $16, $16, $1

Ta thấy rằng lệnh mul chỉ có thể thực hiện được với giá trị thanh ghi nên khi truyền tham số trực tiếp nó sẽ lưu tham số đó vào thanh ghi $1, sử dụng lệnh add sau đó mới thực hiện lệnh mul với thanh ghi $1.

Kết quả sẽ được lưu lại vào thanh ghi $lo, câu lệnh cuối mflo $s1 giúp ta có thể lấy giá trị từ thanh ghi $lo và gán vào thanh ghi $1. Điều này giải thích tại sao cả thanh ghi $s0 và thanh ghi $s1 đều có giá trị là 60.

# Assignment 6: tạo biến và truy cập biến



Text Segment

| Bkpt | Address | Code | Basic | Source |
|---|---|---|---|---|
| | 0x00400000 | 0x3c011001 | lui $1,0x00001001 | 8: la $t8, X # Get the address of X in Data Segment |
| | 0x00400004 | 0x34380000 | ori $24,$1,0x00000000 | |
| | 0x00400008 | 0x3c011001 | lui $1,0x00001001 | 9: la $t9, Y # Get the address of Y in Data Segment |
| | 0x0040000c | 0x34390004 | ori $25,$1,0x00000004 | |
| | 0x00400010 | 0x8f090000 | lw $9,0x00000000($24) | 10: lw $t1, 0($t8) # $t1 = X |
| | 0x00400014 | 0x8f2a0000 | lw $10,0x00000000($.. | 11: lw $t2, 0($t9) # $t2 = Y |
| | 0x00400018 | 0x01298020 | add $16,$9,$9 | 13: add $s0, $t1, $t1 # $s0 = $t1 + $t1 = X + X = 2X |
| | 0x0040001c | 0x020a8020 | add $16,$16,$10 | 14: add $s0, $s0, $t2 # $s0 = $s0 + $t2 = 2X + Y |
| | 0x00400020 | 0x3c011001 | lui $1,0x00001001 | 16: la $t7, Z # Get the address of Z in Data Segment |

Lệnh la được biên dịch thành cặp lệnh lui và ori Ví dụ: la $t8, X sẽ được dịch thành lui $1, 0x00001001 ori $24, $1, 0x00000000 • Địa chỉ của các biến: X: 0x10010000 Y: 0x10010004 Z: 0x10010008

Bài toán trên là bài toán giải phương trình Z = 2X + Y.

Lệnh lw có vai trò lưu địa chỉ của biến vào thanh ghi. • Lệnh sw có vai trò gán giá trị của một thanh ghi vào một địa chỉ