Bài 4. Các lệnh số học và logic

Họ và tên: Nguyễn Thành Duy

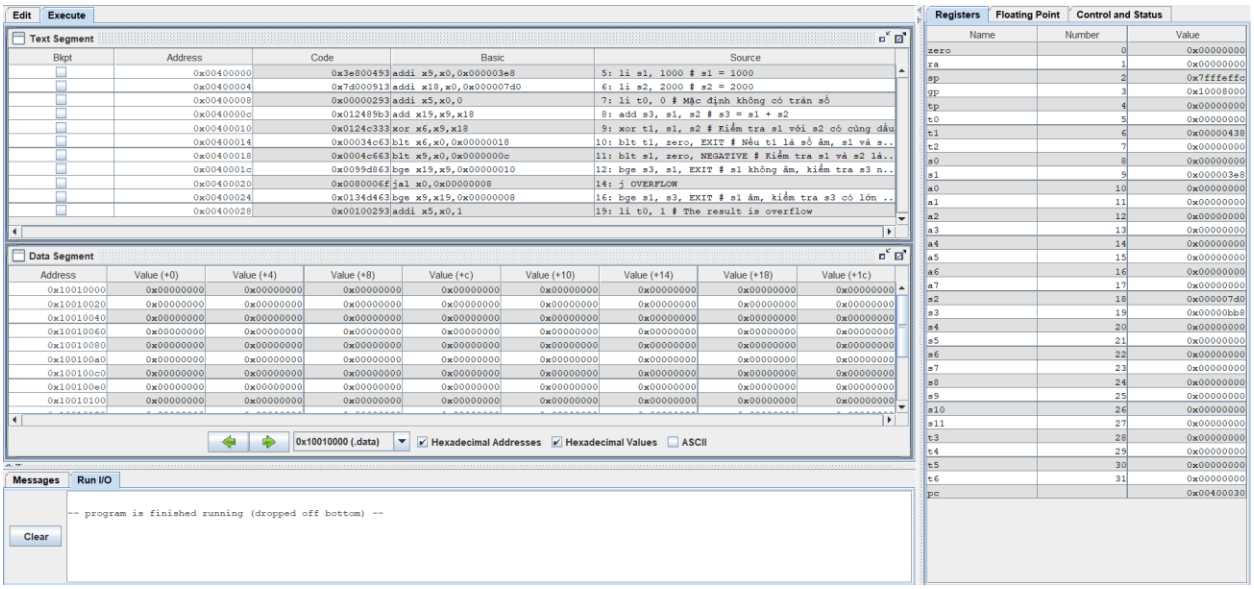
MSSV: 20235696

Assignment 1

1. Trường hợp 2 số dương và tổng không tràn dấu

Code:

```
# Laboratory Exercise 4, Home Assignment 1
.text
# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
# Thuật toán xác định tràn số
li s1, 1000 # s1 = 1000
li s2, 2000 # s2 = 2000
li t0, 0 # Mặc định không có tràn số
add s3, s1, s2 # s3 = s1 + s2
xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu
blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu
blt s1, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm
bge s3, s1, EXIT # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
# Nếu s3 >= s1, không tràn số
j OVERFLOW
NEGATIVE:
bge s1, s3, EXIT # s1 âm, kiểm tra s3 có lớn hơn s1 không
# Nếu s1 >= s3, không tràn số
OVERFLOW:
li t0, 1 # The result is overflow
EXIT:
```

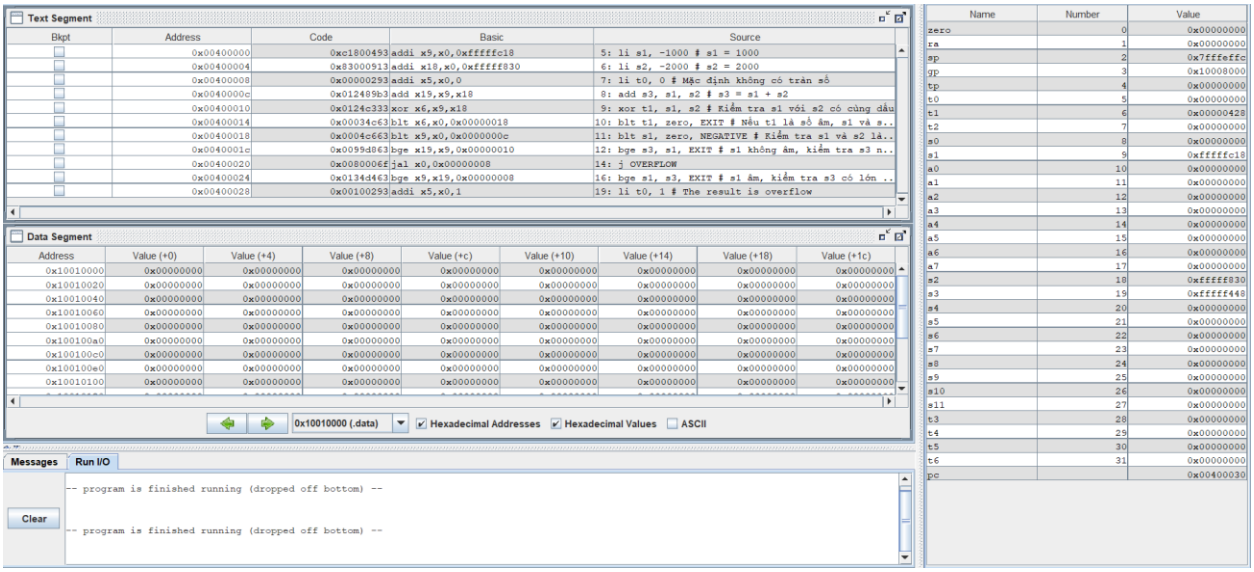


Kết quả: Giá trị thanh ghi t0 = 0 -> không xảy ra hiện tượng tràn số thì kết quả tổng s3 = s1 + s2 = 1000 + 2000 = 3000 (Đúng như dự đoán)

2. Trường hợp 2 số âm, tổng không tràn số

Code:

```
# Laboratory Exercise 4, Home Assignment 1
.text
# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
# Thuật toán xác định tràn số
li s1, -1000 # s1 = 1000
li s2, -2000 # s2 = 2000
li t0, 0 # Mặc định không có tràn số
add s3, s1, s2 # s3 = s1 + s2
xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu
blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu
blt s1, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm
bge s3, s1, EXIT # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
# Nếu s3 >= s1, không tràn số
j OVERFLOW
NEGATIVE:
bge s1, s3, EXIT # s1 âm, kiểm tra s3 có lớn hơn s1 không
# Nếu s1 >= s3, không tràn số
OVERFLOW:
li t0, 1 # The result is overflow
EXIT:
```

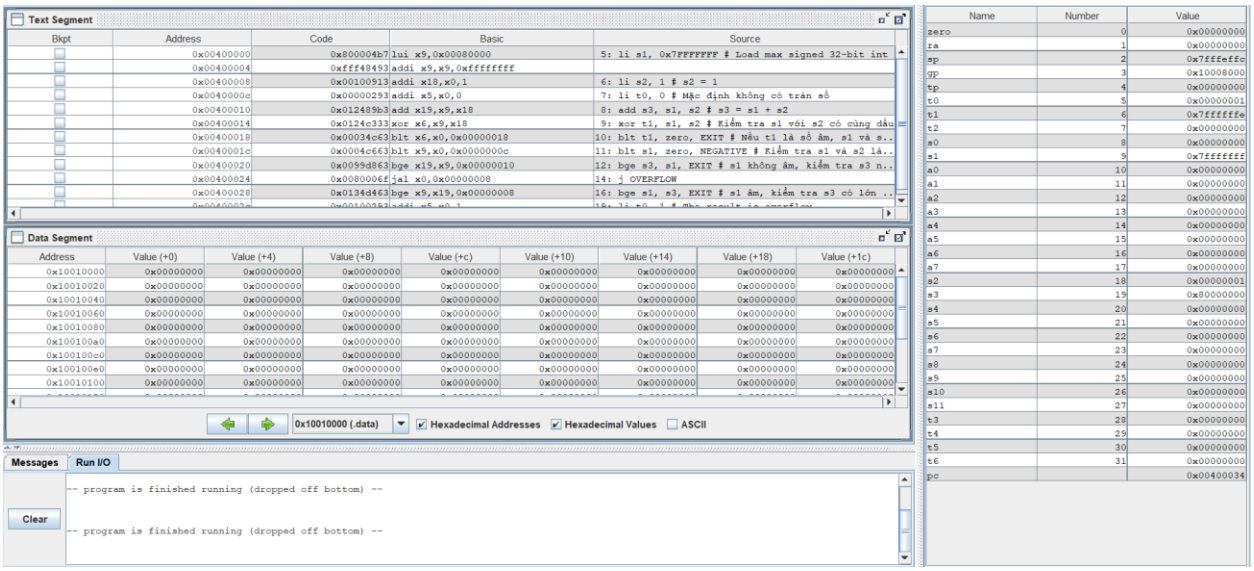


Kết quả: Giá trị thanh ghi t0 = 0 -> không xảy ra hiện tượng tràn số nên s3 = s1 + s2 = -1000 – 2000 = -3000 (Đúng như mong muốn)

3. Trường hợp 2 số dương và bị tràn số

Code:

```
# Laboratory Exercise 4, Home Assignment 1
.text
# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
# Thuật toán xác định tràn số
li s1, 0x7FFFFFFF # Load max signed 32-bit int
li s2, 1 # s2 = 1
li t0, 0 # Mặc định không có tràn số
add s3, s1, s2 # s3 = s1 + s2
xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu
blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu
blt s1, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm
bge s3, s1, EXIT # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
# Nếu s3 >= s1, không tràn số
j OVERFLOW
NEGATIVE:
bge s1, s3, EXIT # s1 âm, kiểm tra s3 có lớn hơn s1 không
# Nếu s1 >= s3, không tràn số
OVERFLOW:
li t0, 1 # The result is overflow
EXIT:
```

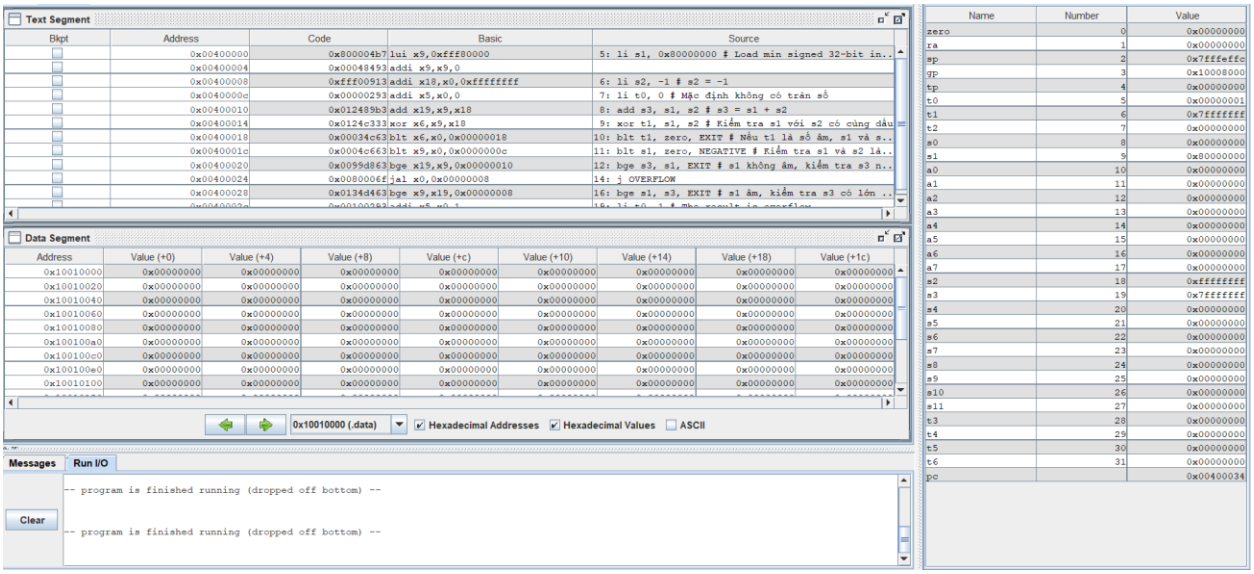


Kết quả: Giá trị thanh ghi t0 = 1 -> Xảy ra hiện tượng tràn số, Giá trị thanh ghi tổng s3 = 0x80000000 (tức -2,147,483,648 – giá trị nhỏ nhất có thể biểu diễn)

4. Trường hợp 2 số âm, tràn số

Code:

```
# Laboratory Exercise 4, Home Assignment 1
.text
# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
# Thuật toán xác định tràn số
li s1, 0x80000000 # Load min signed 32-bit int (-2,147,483,648)
li s2, -1 # s2 = -1
li t0, 0 # Mặc định không có tràn số
add s3, s1, s2 # s3 = s1 + s2
xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu
blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu
blt s1, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm
bge s3, s1, EXIT # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
# Nếu s3 >= s1, không tràn số
j OVERFLOW
NEGATIVE:
bge s1, s3, EXIT # s1 âm, kiểm tra s3 có lớn hơn s1 không
# Nếu s1 >= s3, không tràn số
OVERFLOW:
li t0, 1 # The result is overflow
EXIT:
```

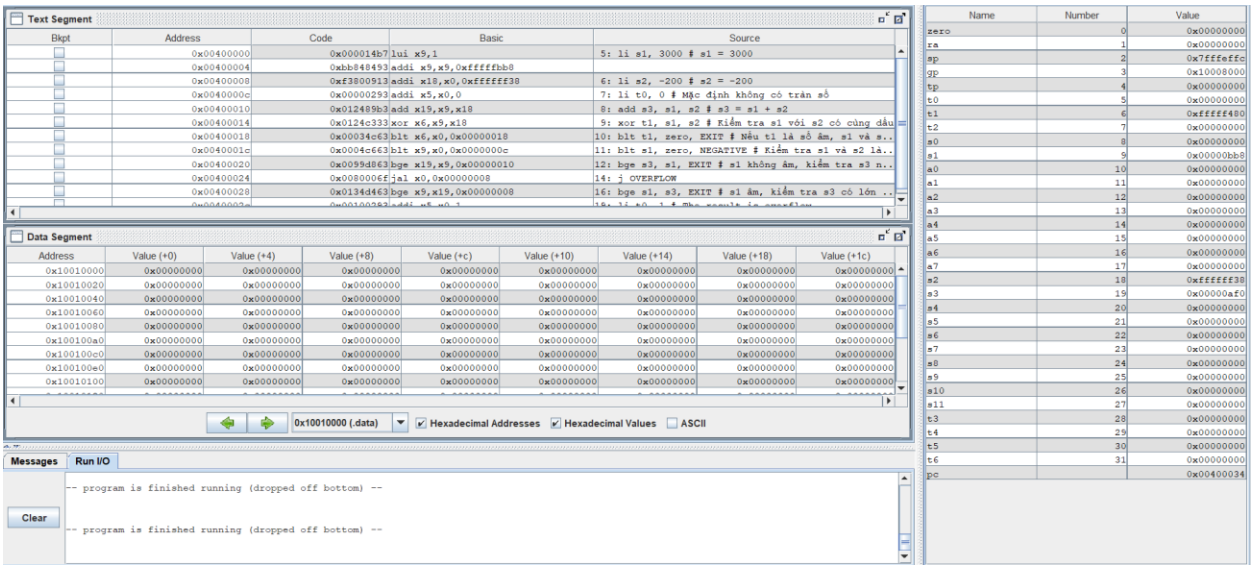


Kết quả: t0 = 1 -> Xảy ra hiện tượng tràn số, Giá trị thanh ghi tổng s3 = 0x7fffffff (tức 2,147,483,647)

5. Trường hợp 2 số trái dấu, không tràn số

Code:

```
# Laboratory Exercise 4, Home Assignment 1
.text
# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
# Thuật toán xác định tràn số
li s1, 3000 # s1 = 3000
li s2, -200 # s2 = -200
li t0, 0 # Mặc định không có tràn số
add s3, s1, s2 # s3 = s1 + s2
xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu
blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu
blt s1, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm
bge s3, s1, EXIT # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
# Nếu s3 >= s1, không tràn số
j OVERFLOW
NEGATIVE:
bge s1, s3, EXIT # s1 âm, kiểm tra s3 có lớn hơn s1 không
# Nếu s1 >= s3, không tràn số
OVERFLOW:
li t0, 1 # The result is overflow
EXIT:
```



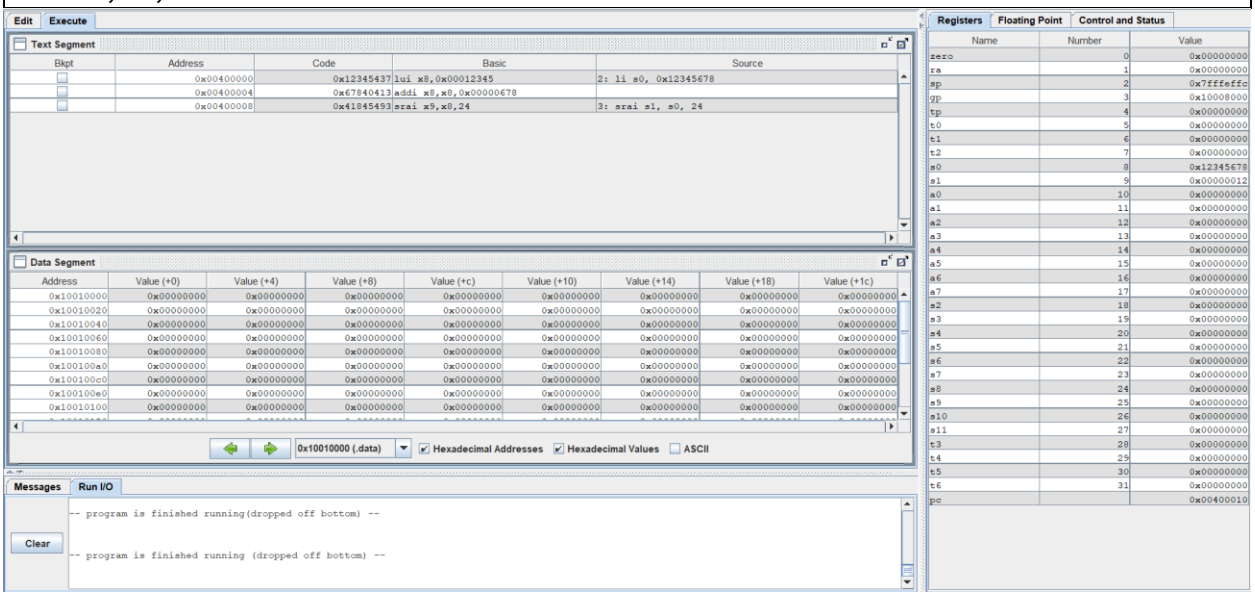
Kết quả: Giá trị thanh ghi t0 = 0 -> không xảy ra hiện tượng tràn số.

Assignment 2

1. Trích xuất MSB của thanh ghi s0

Code:

```
.text
li s0, 0x12345678
srai s1, s0, 24
```

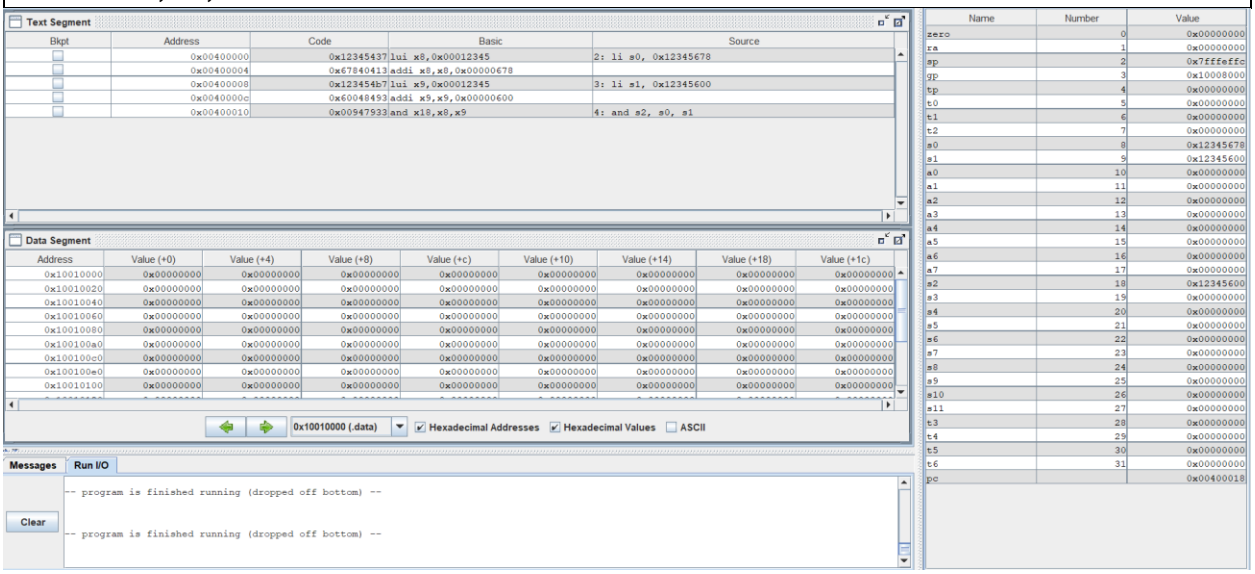


Kết quả: Thanh ghi s1 = 0x00000012

2. Xóa LSB của thanh ghi s0

Code:

```
.text
li s0, 0x12345678
li s1, 0x12345600
and s2, s0, s1
```

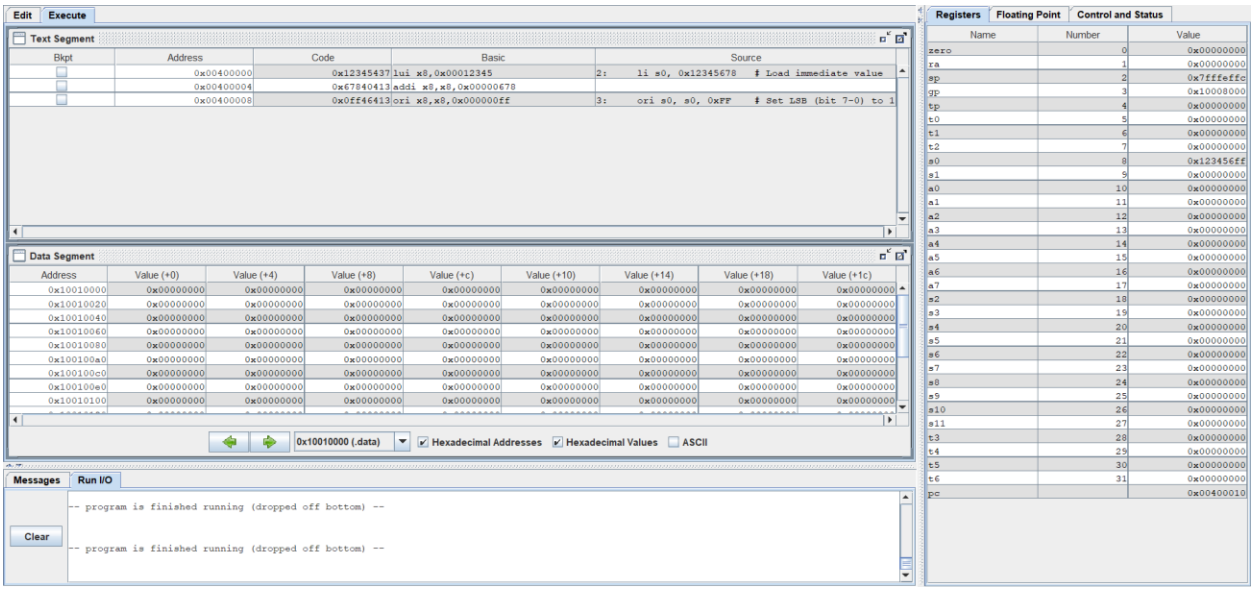


Kết quả: s2 = 0x12345600

3. Thiết lập LSB của thanh ghi s0 (bit 7 đến bit 0 được thiết lập bằng 1)

Code:

```
.text
  li s0, 0x12345678   # Load immediate value
  ori s0, s0, 0xFF    # Set LSB (bit 7-0) to 1
```

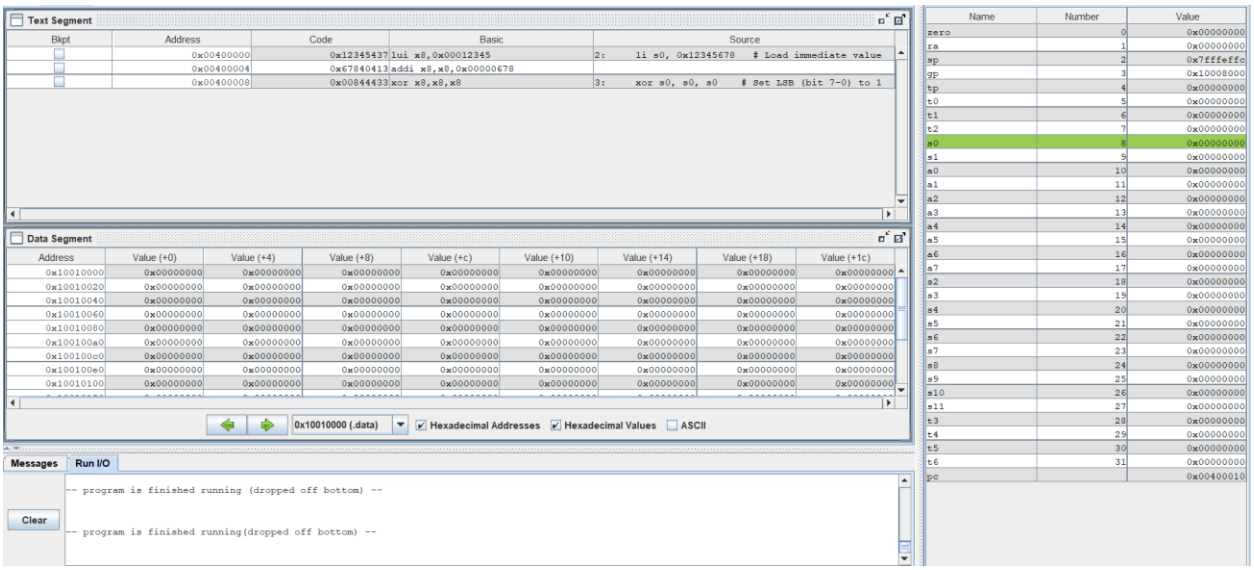


Kết quả: Giá trị thanh ghi s0 = 0x123456ff (với **bit 7 đến bit 0 được thiết lập bằng 1**.)

4. Xóa thanh ghi s0 bằng cách dùng các lệnh logic (s0 = 0)

Code:

```
.text
  li s0, 0x12345678   # Load immediate value
  xor s0, s0, s0    # Set LSB (bit 7-0) to 1
```

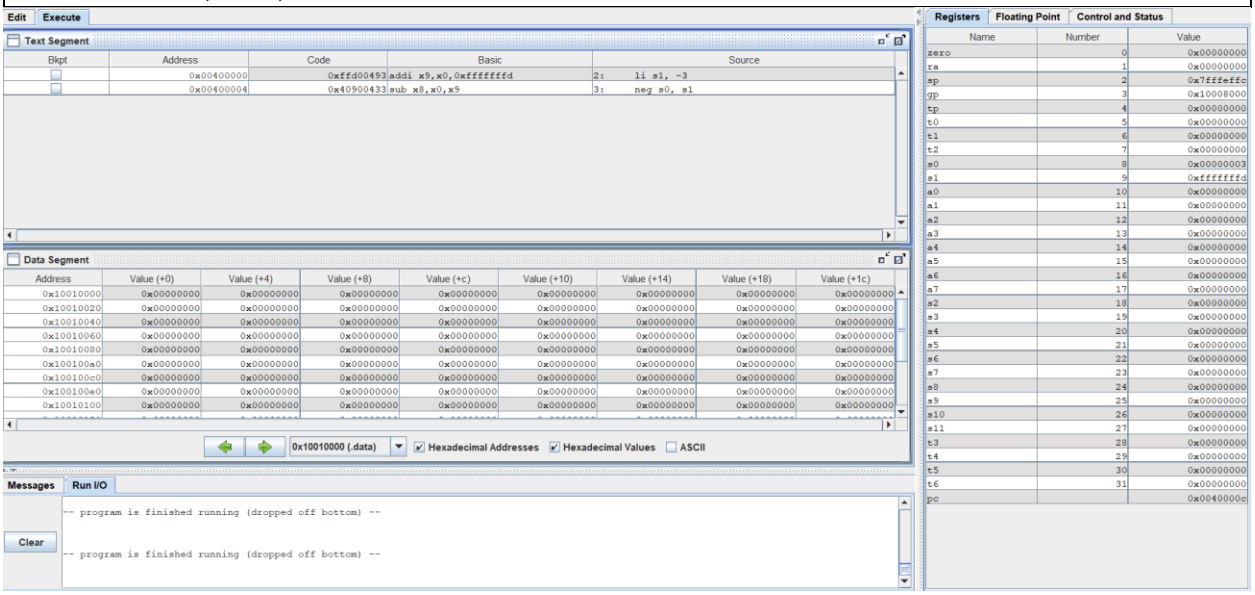


Kết quả: Thanh ghi s0 bị xóa về 0

Assignment 3

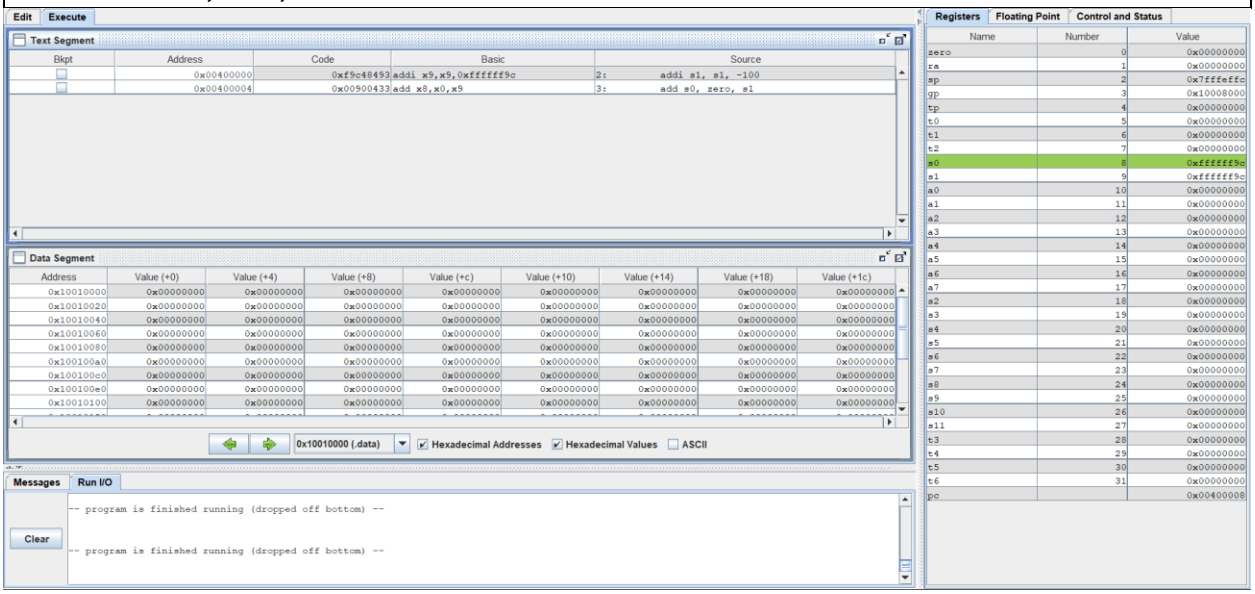
a. neq s0, s1

```
.text
    addi s1, s1, -100
    sub s0, zero, s1
```



b. mv s0, s1

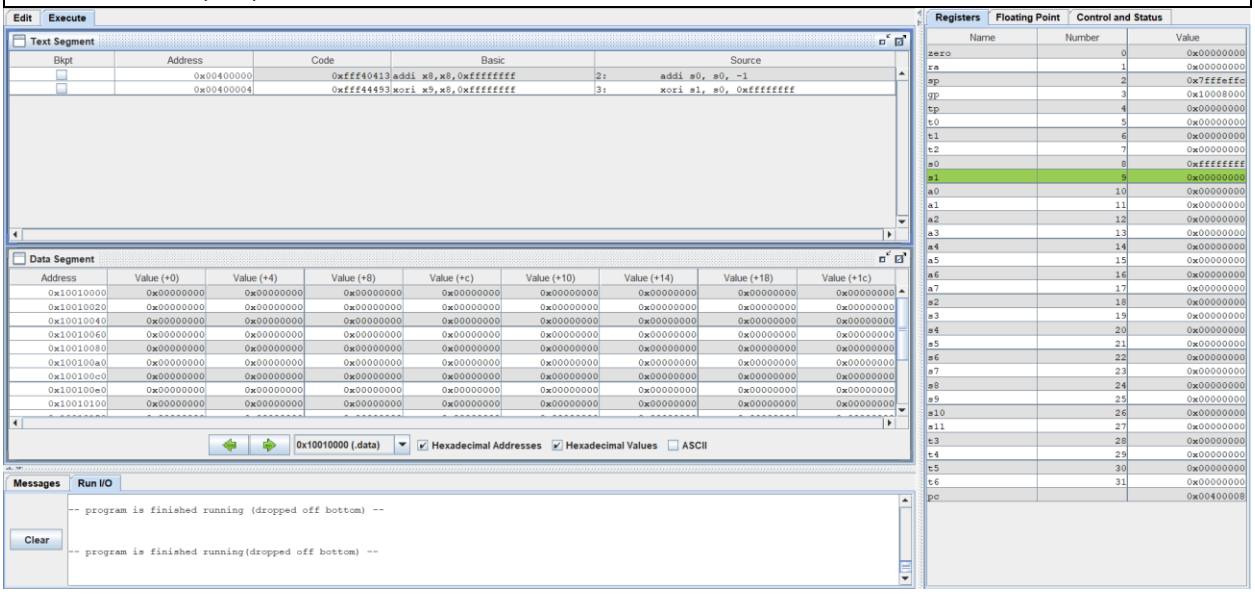
```
.text
    addi s1, s1, -100
    add s0, zero, s1
```



c. Not s0

Code:

```
.text
        addi s0, s0, -1
        xori s1, s0, 0xffffffff
```

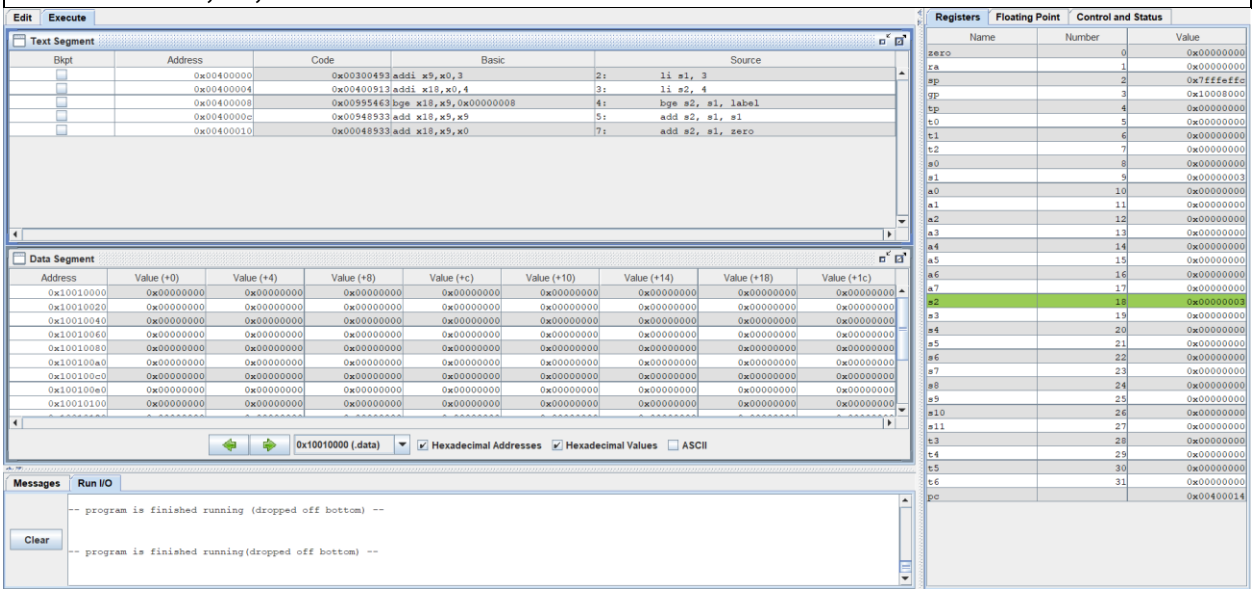


Kết quả: Các bit đã bị đảo 1->0

d. ble s1, s2, label

Code:

```
.text
        li s1, 3
        li s2, 4
        bge s2, s1, label
        add s2, s1, s1
label:
        add s2, s1, zero
```



Kết quả: s2 = s1 = 3 do s1 <= s2 nên nhảy tới label để thực hiện. Nếu ngược lại s1 > s2 thì sẽ thực hiện s2 = 2s1 = 4.

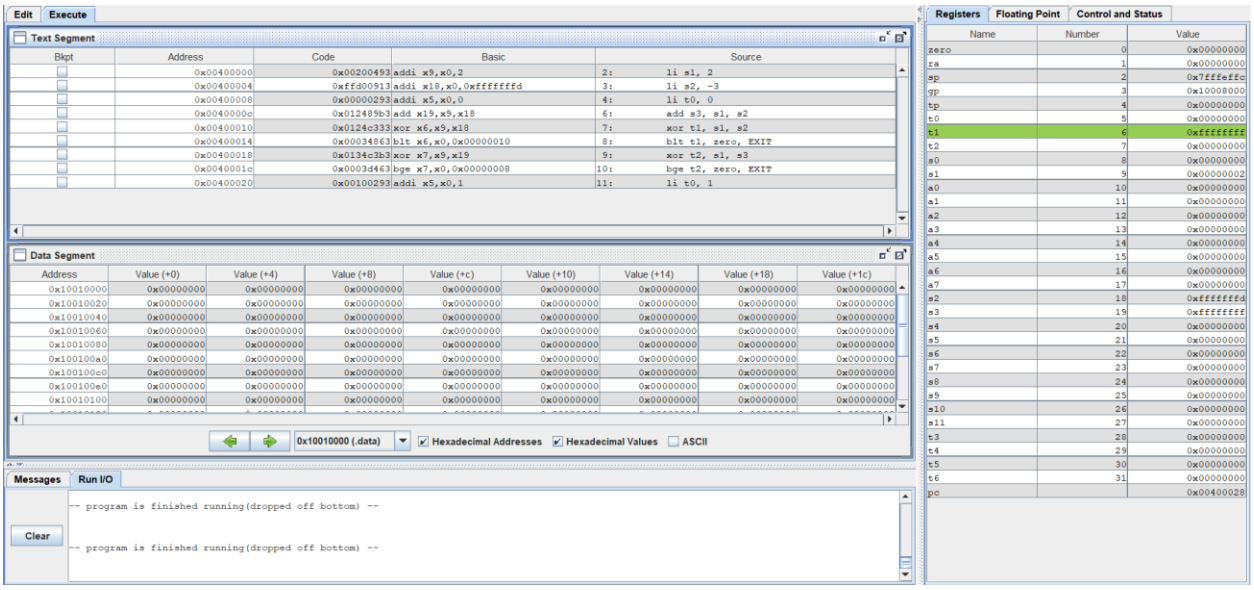
Assignment 4

1. Trường hợp 2 số trái dấu

Code:

```
.text
        li s1, 2
        li s2, -3
        li t0, 0

        add s3, s1, s2
        xor t1, s1, s2
        blt t1, zero, EXIT
        xor t2, s1, s3
        bge t2, zero, EXIT
        li t0, 1
```

```
EXIT:
```

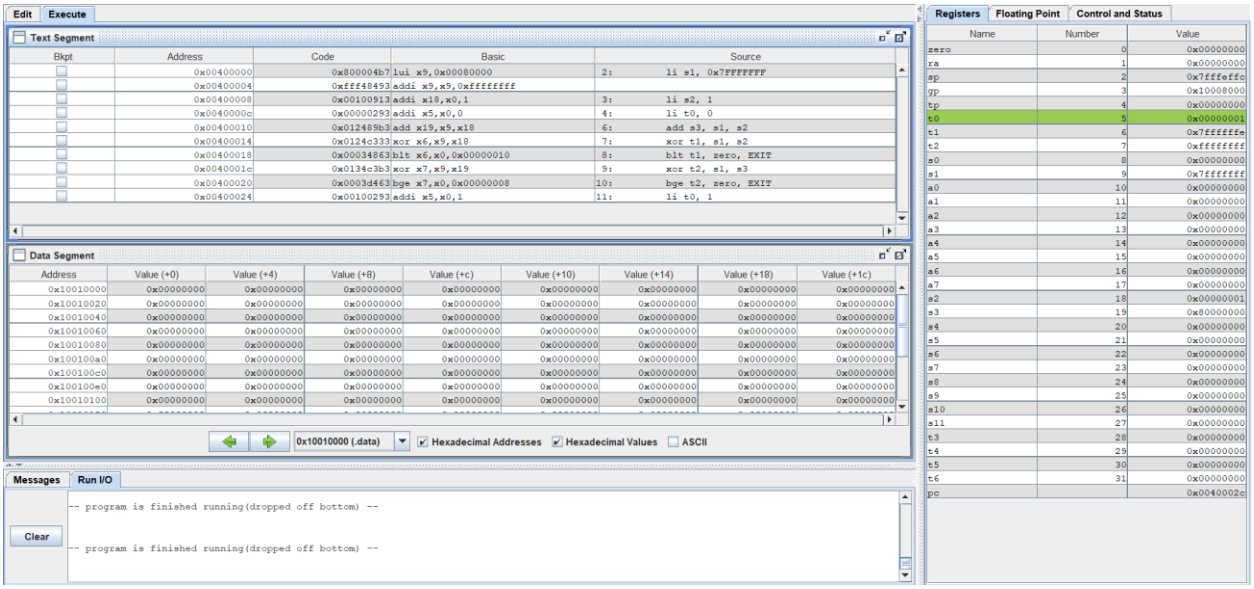


Kết quả: t0 = 0 -> không xảy ra hiện tượng tràn số.

2. Trường hợp 2 số cùng dấu tràn số

Code:

```
.text
        li s1, 0x7FFFFFFF
        li s2, 1
        li t0, 0

        add s3, s1, s2
        xor t1, s1, s2
        blt t1, zero, EXIT
        xor t2, s1, s3
        bge t2, zero, EXIT
        li t0, 1
EXIT:
```

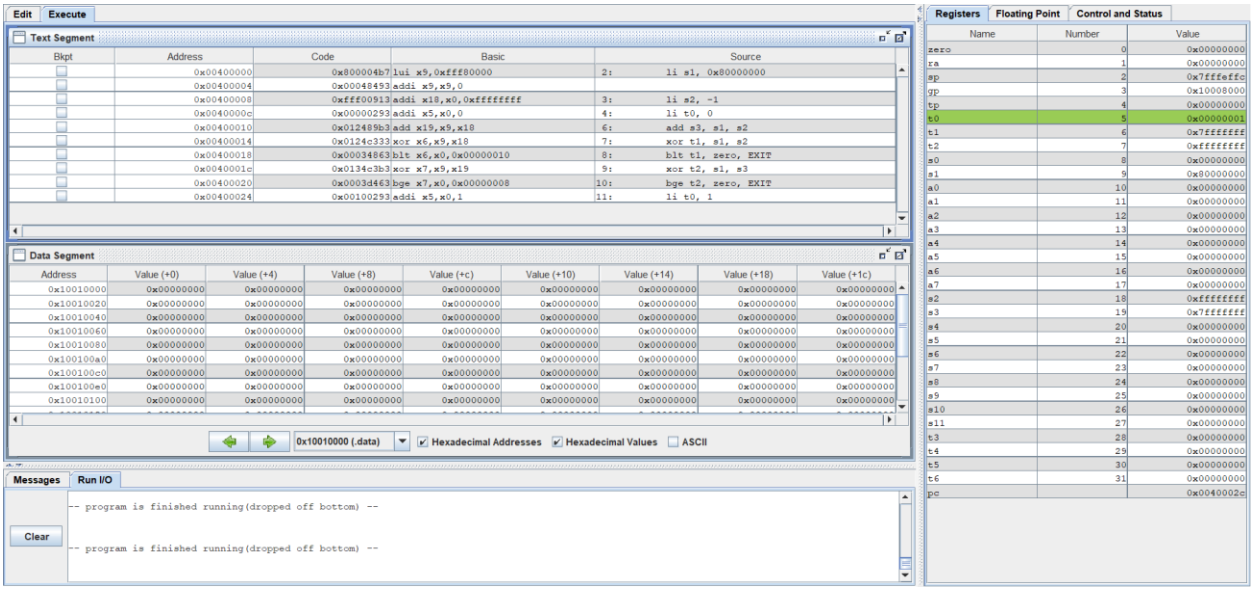


Kết quả: t0 = 1 -> xảy ra hiện tượng tràn số

3. Trường hợp 2 số âm tràn số

Code:

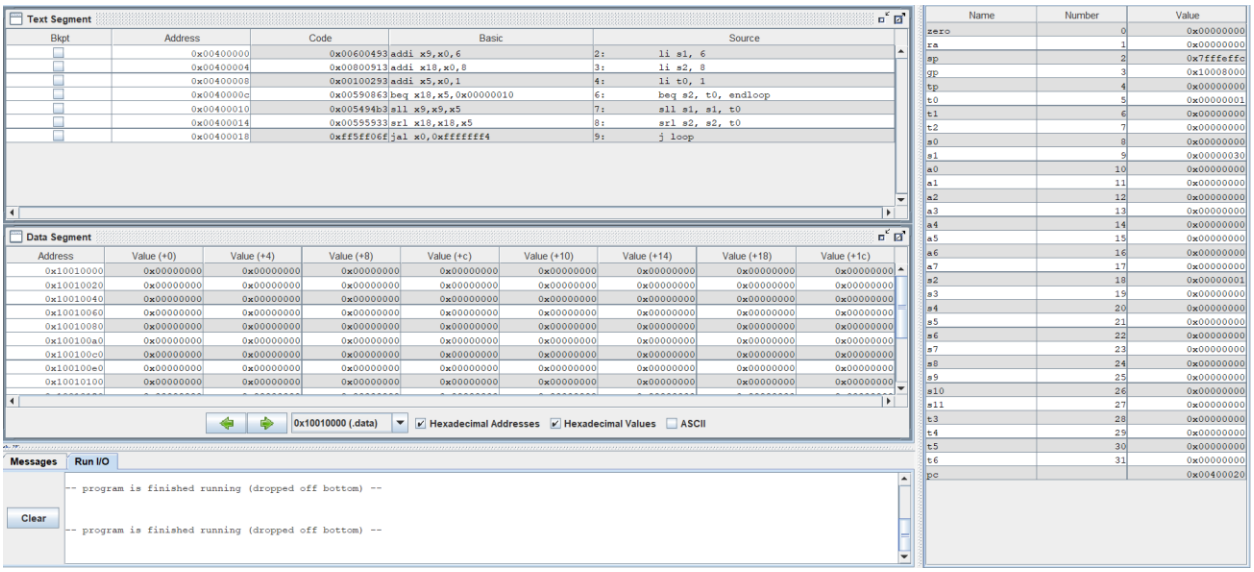
```
.text
        li s1, 0x80000000
        li s2, -1
        li t0, 0

        add s3, s1, s2
        xor t1, s1, s2
        blt t1, zero, EXIT
        xor t2, s1, s3
        bge t2, zero, EXIT
        li t0, 1
EXIT:
```



Kết quả: t0 = 1 -> Xảy ra hiện tượng tràn số

Assignment 5

```
.text
        li s1, 6
        li s2, 8
        li t0, 1
loop:
        beq s2, t0, endloop
        sll s1, s1, t0
        srl s2, s2, t0
        j loop
endloop:
```



Kết quả: s1 = s1*s2 = 48

KẾT LUẬN: Lợi ích của phép dịch bit so với lệnh nhân (mul)

- Nhanh hơn trên phần cứng không hỗ trợ M extension
- Tiêu tốn ít tài nguyên hơn
- Thích hợp cho các bội số của 2