

# BÁO CÁO THỰC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN 4

Họ và tên: Hoàng Văn Thắng

MSSV: 20235828

## Assignment 1

Tạo project để thực hiện chương trình ở Home Assignment. Dịch và chạy mô phỏng với RARS. Khởi tạo các toán hạng cần thiết, chạy từng lệnh của chương trình, quan sát bộ nhớ và giá trị thanh ghi.

### 1. Home Assignment 1

**Trường hợp 1:** Tràn hai số nguyên dương

**Nhập chương trình:**

```
1  # Laboratory Exercise 4, Home Assignment 1
2  .text
3      # TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
4      # Case 1: Tràn hai số dương
5      li s1, 2147483647
6      li s2, 1
7
8      # Thuật toán xác định tràn số
9      li t0, 0          # Mặc định không có tràn số
10     add s3, s1, s2     # s3 = s1 + s2
11     xor t1, s1, s2     # Kiểm tra s1 với s2 có cùng dấu
12     blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu
13     blt s1, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm
14     bge s3, s1, EXIT   # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
15     # Nếu s3 >= s1
16     j OVERFLOW
17 NEGATIVE:
18     bge s1, s3, EXIT   # s1 âm, kiểm tra s3 có lớn hơn s1 không
19     # Nếu s1 >= s3, không tràn số
20 OVERFLOW:
21     li t0, 1          # The result is overflow
22 EXIT:
```

```
# Laboratory Exercise 4, Home Assignment 1
```

```
.text
```

```
    # TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
```

```
    # Case 1: Tràn hai số dương
```

```
    li s1, 2147483647
```

```
    li s2, 1
```

```

# Thuật toán xác định tràn số
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
    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 chạy

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000001
t1	6	0x7ffffffe
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x7fffffff
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000001
s3	19	0x80000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400034

## Quan sát bộ nhớ và thanh ghi:

Trước khi thực hiện phép cộng  $s3 = s1 + s2$  ta thấy:

- $s1 = 2147483647$
- $s2 = 1$
- $t0 = 0$

Khi thực hiện phép cộng ta thấy hiện tượng overflow và được lưu vào  $s3$  giá trị:  $0x80000000$  là số âm trong kiểu số nguyên có dấu 32-bit, cho thấy một hiện tượng tràn số đã xảy ra.

Kết quả:

- $s1$  chứa giá trị  $2147483647$  ( $0x7FFFFFFF$ ), là số nguyên dương lớn nhất có dấu trên 32-bit.
- $s2$  chứa giá trị  $1$
- $s3$  chứa giá trị của phép cộng  $s1 + s2$ , có giá trị  $0x80000000$ , là số âm trong hệ thống có dấu 32-bit.
- $t0$  được đặt thành  $1$ , cho biết rằng đã xảy ra hiện tượng tràn số

## Trường hợp 2: 2 số nguyên âm không tràn

### Nhập chương trình:

```
1  # Laboratory Exercise 4, Home Assignment 1
2  .text
3      # TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
4      # Case 2: Hai số nguyên âm
5      li s1, -2147483647
6      li s2, -1
7
8      # Thuật toán xác định tràn số
9      li t0, 0                # Mặc định không có tràn số
10     add s3, s1, s2          # s3 = s1 + s2
11     xor t1, s1, s2          # Kiểm tra s1 với s2 có cùng dấu
12     blt t1, zero, EXIT      # Nếu t1 là số âm, s1 và s2 khác dấu
13     blt s1, zero, NEGATIVE   # Kiểm tra s1 và s2 là số âm hay không âm
14     bge s3, s1, EXIT         # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
15     # Nếu s3 >= s1
16     j OVERFLOW
17 NEGATIVE:
18     li t0, 2                # t0 = 2 (nếu s1 và s2 là số âm)
19     bge s1, s3, EXIT         # s1 âm, kiểm tra s3 có lớn hơn s1 không
20     # Nếu s1 >= s3, không tràn số
21 OVERFLOW:
22     li t0, 1                # The result is overflow
23 EXIT:
```

# # Laboratory Exercise 4, Home Assignment 1

.text

# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau

# Case 2: Hai số nguyên âm

li s1, -2147483647

li s2, -1

# Thuật toán xác định tràn số

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  $s3 \geq s1$

j OVERFLOW

NEGATIVE:

li t0, 2 #  $t0 = 2$  (nếu s1 và s2 là số âm)

bge s1, s3, EXIT # s1 âm, kiểm tra s3 có lớn hơn s1 không

# Nếu  $s1 \geq s3$ , không tràn số

OVERFLOW:

li t0, 1 # The result is overflow

EXIT:

## Kết quả chạy

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffefc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x7fffffff
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x80000001
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0xffffffff
s3	19	0x80000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400038

### Quan sát bộ nhớ và thanh ghi:

- Kết quả của phép cộng:  $s1 + s2 = -2147483648$
- Không có tràn số ( $t0 = 0$ )
- $t0: 0x00000002$  (có nghĩa là  $s1, s2$  là hai số nguyên âm)

**Trường hợp 3:** Hai số có tổng bằng 0 (một số nguyên dương, một số nguyên âm)

### Nhập chương trình

```
1  # Laboratory Exercise 4, Home Assignment 1
2  .text
3      # TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
4      # Case 3: Tổng hai số bằng 0
5      li s1, 69
6      li s2, -69
7
8      # Thuật toán xác định tràn số
9      li    t0, 0                # Mặc định không có tràn số
10     add   s3, s1, s2           # s3 = s1 + s2
11     xor   t1, s1, s2           # Kiểm tra s1 với s2 có cùng dấu
12     blt   t1, zero, EXIT       # Nếu t1 là số âm, s1 và s2 khác dấu
13     blt   s1, zero, NEGATIVE   # Kiểm tra s1 và s2 là số âm hay không âm
14     bge   s3, s1, EXIT         # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
15     # Nếu s3 >= s1
16     j     OVERFLOW
17 NEGATIVE:
18     bge   s1, s3, EXIT         # s1 âm, kiểm tra s3 có lớn hơn s1 không
19     # Nếu s1 >= s3, không tràn số
20 OVERFLOW:
21     li    t0, 1                # The result is overflow
22 EXIT:|
```

# Laboratory Exercise 4, Home Assignment 1

.text

# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau

# Case 3: Tổng hai số bằng 0

li s1, 69

li s2, -69

# Thuật toán xác định tràn số

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

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 chạy

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0xfffffffffe
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000045
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0xffffffffbb
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400030

## Quan sát bộ nhớ và thanh ghi

Kết quả là chương trình thoát khi gặp câu lệnh: *blt t1, zero, EXIT*

**Trường hợp 4:** Có một số có giá trị bằng 0 (Tổng quát: Hai số dương không tràn)

## Nhập chương trình

```

1  # Laboratory Exercise 4, Home Assignment 1
2  .text
3      # TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
4      # Case 4: Một số có giá trị bằng 0
5      li s1, 0
6      li s2, 69
7
8      # Thuật toán xác định tràn số
9      li    t0, 0                # Mặc định không có tràn số
10     add   s3, s1, s2           # s3 = s1 + s2
11     xor   t1, s1, s2           # Kiểm tra s1 với s2 có cùng dấu
12     blt   t1, zero, EXIT       # Nếu t1 là số âm, s1 và s2 khác dấu
13     blt   s1, zero, NEGATIVE   # Kiểm tra s1 và s2 là số âm hay không âm
14     bge   s3, s1, EXIT         # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
15     # Nếu s3 >= s1
16     j     OVERFLOW
17 NEGATIVE:
18     bge   s1, s3, EXIT         # s1 âm, kiểm tra s3 có lớn hơn s1 không
19     # Nếu s1 >= s3, không tràn số
20 OVERFLOW:
21     li    t0, 1                # The result is overflow
22 EXIT:|

```

# Laboratory Exercise 4, Home Assignment 1

.text

# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau

# Case 4: Một số có giá trị bằng 0

li s1, 0

li s2, 69

# Thuật toán xác định tràn số

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

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 chạy

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000045
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000045
s3	19	0x00000045
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400030

## Quan sát bộ nhớ và thanh ghi

- Chương trình không bị tràn số và thoát khi gặp lệnh:

*bge s3, s1, EXIT* # s1 và s2 dương. Nếu  $s3 \geq s1$  thì kết quả không bị tràn số

## Trường hợp 5: Tràn hai số nguyên âm

## Nhập chương trình

```

1  # Laboratory Exercise 4, Home Assignment 1
2  .text
3      # TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
4      # Case 5: Tràn hai số nguyên âm
5      li s1, -2147483648
6      li s2, -1
7
8      # Thuật toán xác định tràn số
9      li    t0, 0                # Mặc định không có tràn số
10     add   s3, s1, s2           # s3 = s1 + s2
11     xor   t1, s1, s2           # Kiểm tra s1 với s2 có cùng dấu
12     blt   t1, zero, EXIT       # Nếu t1 là số âm, s1 và s2 khác dấu
13     blt   s1, zero, NEGATIVE   # Kiểm tra s1 và s2 là số âm hay không âm
14     bge   s3, s1, EXIT         # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
15     # Nếu s3 >= s1
16     j     OVERFLOW
17 NEGATIVE:
18     bge   s1, s3, EXIT         # s1 âm, kiểm tra s3 có lớn hơn s1 không
19     # Nếu s1 >= s3, không tràn số
20 OVERFLOW:
21     li    t0, 1                # The result is overflow
22 EXIT:|

```

# Laboratory Exercise 4, Home Assignment 1

.text

# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau

# Case 5: Tràn hai số nguyên âm

li s1, -2147483648

li s2, -1

# Thuật toán xác định tràn số

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

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 chạy

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000001
t1	6	0x7fffffff
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x80000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0xffffffff
s3	19	0x7fffffff
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400034

### Quan sát bộ nhớ và thanh ghi

Ta thấy t0 có giá trị 0x00000001 → đã rơi vào trường hợp tràn số

Viết một chương trình thực hiện các công việc sau:

- MSB: Most Significant Byte (Byte có trọng số cao)

s0 = 0x 1 2 3 4 5 6 7 8  
                   ↓                   ↓  
                   MSB                LSB

```

1 .text
2     li      s0, 0x12345678      # s0 = 0x12345678
3     srli    t0, s0, 24          # Trích xuất MSB (dịch phải 24 bit)
4     andi    t1, s0, 0xFFFFF00  # Xóa LSB (AND với 0xFFFFF00 để xóa LSB)
5     ori     t2, s0, 0x000000FF  # Thiết lập LSB (OR với 0xFF để đặt các bit từ 7 đến 0)
6     li      s0, 0               # Xóa thành ghi s0 (s0 = 0)

```

```
.text
    li    s0, 0x12345678          # s0 = 0x12345678
    srli  t0, s0, 24              # Trích xuất MSB (dịch phải 24 bit)
    andi  t1, s0, 0xFFFFFFFF00   # Xóa LSB (AND với 0xFFFFFFFF00 để
xóa LSB)
    ori   t2, s0, 0x000000FF      # Thiết lập LSB (OR với 0xFF để đặt các
bit từ 7 đến 0)
    li    s0, 0                  # Xóa thanh ghi s0 (s0 = 0)
```

## Kết quả chạy chương trình

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000012
t1	6	0x12345600
t2	7	0x123456ff
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x0040001c

## Quan sát các thanh ghi và bộ nhớ

### - Thanh ghi s0

- + Ban đầu, s0 được khởi tạo với giá trị 0x12345678
- + Ở cuối chương trình, lệnh *li s0, 0* sẽ đặt giá trị của s0 thành 0.
- + **Giá trị cuối cùng của s0:** 0x00000000

### - Thanh ghi t0 (trích xuất MSB của s0)

- + Lệnh *srli t0, s0, 24* dịch phải giá trị trong s0 24-bit, để lại 8 bit cao nhất (MSB).

+ Giá trị ban đầu của s0 là 0x12345678, khi dịch phải 24-bit, chỉ còn lại giá trị 0x12.

+ **Giá trị cuối cùng của t0:** 0x00000012

- **Thanh ghi t1** (xóa LSB của s0)

+ Lệnh *andi t1, s0, 0xFFFFF00* sẽ giữ nguyên tất cả các bit ngoại trừ 8 bit thấp nhất (LSB) bằng cách AND với 0xFFFFF00.

+ Giá trị ban đầu của s0 là 0x12345678, khi AND với 0xFFFFF00, kết quả sẽ là 0x12345600

+ **Giá trị cuối cùng của t1:** 0x12345600

- **Thanh ghi t2** (thiết lập LSB của s0)

+ Lệnh *ori t2, s0, 0x000000FF* sẽ thiết lập 8 bit thấp nhất của s0 thành 1 (hoặc 0xFF).

+ Giá trị ban đầu của s0 là 0x12345678, khi OR với 0x000000FF, kết quả sẽ là 0x123456FF.

+ **Giá trị cuối cùng của t2:** 0x123456FF

### Assignment 3

Như đã đề cập, giả lệnh không phải lệnh chính thống của RISC-V, khi biên dịch assembler sẽ chuyển chúng thành các lệnh chính thống. Viết chương trình thực thi các giả lệnh dưới đây sử dụng các lệnh chính thống mà RISC-V định nghĩa:

a. *neg s0, s1* ( $s0 = -s1$ )

#### Nhập chương trình

```
1 .text
2     li s1, 10          # s1 = 10
3     sub s0, zero, s1   # s0 = 0 - s1
```

<pre>.text     li s1, 10          # s1 = 10     sub s0, zero, s1   # s0 = 0 - s1</pre>
--

## Kết quả chạy

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0xffffffff6
s1	9	0x0000000a
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x0040000c

**Giải thích:** Lệnh *neg s0, s1* có chức năng lấy giá trị âm của thanh ghi s1 và lưu kết quả vào thanh ghi s0. Trong RISC-V, lệnh chính thống để thực hiện lấy giá trị âm là sử dụng lệnh *sub* với toán hạng zero (thanh ghi không) để trừ giá trị của s1 từ 0.

b. *mv s0, s1* ( $s0 = s1$ )

## Nhập chương trình

```
1 .text
2     li    s1, 10        # s1 = 10
3     addi  s0, s1, 0      # s0 = s1 + 0
```

```
.text
```

```
li    s1, 10    # s1 = 10  
addi  s0, s1, 0    # s0 = s1 + 0
```

Kết quả chạy

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffefc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x0000000a
s1	9	0x0000000a
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x0040000c

**Giải thích:** Lệnh *mv s0, s1* có chức năng sao chép giá trị từ thanh ghi s1 sang thanh ghi s0. Trong RISC-V, có thể thực hiện việc này bằng cách dùng lệnh *addi* với giá trị cộng thêm là 0.



c. not s0 (s0 = bit\_invert(s0))

### Nhập chương trình

```
1 .text
2     li s0, 1          # s0 = 1
3     xori s0, s0, -1    # s0 = s0 XOR (-1) (đảo tất cả các bit của s0)
```

.text	
li s0, 1	# s0 = 1
xori s0, s0, -1	# s0 = s0 XOR (-1) (đảo tất cả các bit của s0)

### Kết quả chạy

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0xffffffff
s1	9	0x00000000
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x0040000c

**Giải thích:** Lệnh này có chức năng đảo tất cả các bit của thanh ghi s0. Trong RISC-V, có thể thực hiện phép này bằng cách sử dụng lệnh xori với giá trị -1 (hay tất cả các bit là 1 trong số nguyên có dấu).

d. ble s1, s2, label

if (s1 < s2)

j label

## Nhập chương trình

```
1 .text
2     li s1, 5           # s1 = 5
3     li s2, 10          # s2 = 10
4     li t1, 1           # x = 1
5     li t2, 2           # y = 2
6     li t3, 3           # z = 3
7     bge s2, s1, else   # if s2 >= s1, jump else
8 then:
9     add t3, t1, t2      # z = x + y
10    j endif
11 else:
12    sub t3, t2, t1       # z = y - x
13 endif:
```

```
.text
    li s1, 5           # s1 = 5
    li s2, 10          # s2 = 10
    li t1, 1           # x = 1
    li t2, 2           # y = 2
    li t3, 3           # z = 3
    bge s2, s1, else   # if s2 >= s1, jump else
then:
    add t3, t1, t2      # z = x + y
    j endif
else:
    sub t3, t2, t1       # z = y - x
endif:
```

## Kết quả chạy

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000001
t2	7	0x00000002
s0	8	0x00000000
s1	9	0x00000005
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x0000000a
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000001
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400028

$$s2 \geq s1 \Rightarrow z = y - x = 2 - 1 = 1$$

**Giải thích:** Lệnh này thực hiện phép so sánh hai thanh ghi s1 và s2, và nếu s1 nhỏ hơn hoặc bằng s2, chương trình sẽ nhảy tới nhãn label. Trong RISC-V, không có lệnh ble (branch if less than or equal), nhưng có thể sử dụng lệnh bge (branch if greater than or equal) để kiểm tra  $s2 \geq s1$ , tương đương  $s1 \leq s2$ .

## Assignment 4

Để xác định tràn số xảy ra khi thực hiện phép cộng, có một cách đơn giản hơn so với cách được mô tả trong Home Assignment 1. Giải thuật được mô tả như sau: Khi cộng hai toán hạng cùng dấu, tràn số xảy ra nếu tổng của chúng không cùng dấu với hai toán hạng nguồn. Hãy viết chương trình xác định tràn số theo giải thuật trên.

### Nhập chương trình

```
1  .data
2      message_no_overflow:  .asciz "No overflow detected.\n"
3      message_overflow:     .asciz "Overflow detected.\n"
4  .text
5      .globl _start
6  _start:
7      # Giả sử chúng ta cần cộng hai số nguyên dương hoặc hai số nguyên âm, ở đây sử dụng hai số nguyên dương.
8      li s0, 0x7FFFFFFF      # Toán hạng thứ nhất (số nguyên dương lớn nhất 32-bit)
9      li s1, 1               # Toán hạng thứ hai (giá trị cần cộng)
10
11     add t0, s0, s1          # t0 = s0 + s1
12
13     # Kiểm tra tràn số
14     # 1. Kiểm tra nếu s0 và s1 đều là số dương
15     # 2. Nếu tổng t0 là số âm, điều đó có nghĩa là có tràn số dương.
16     bltz s0, check_negative # If s0 < 0, jump check_negative
17     bltz s1, check_negative # If s1 < 0, jump check_negative
18     bltz t0, overflow       # If t0 < 0, jump overflow
19     j no_overflow           # jump no_overflow
20
21 check_negative:
22     # Kiểm tra nếu cả s0 và s1 đều là số âm
23     # Nếu tổng là số dương, điều đó có nghĩa là có tràn số âm.
24     bgez s0, no_overflow    # If s0 >= 0, jump no_overflow
25     bgez s1, no_overflow    # If s1 >= 0, jump no_overflow
26     bgez t0, overflow       # If t0 >= 0, jump overflow
27
28 no_overflow:
29     # In ra thông báo không có tràn số
30     la a0, message_no_overflow # Đưa địa chỉ chuỗi "No overflow detected!" vào a0
31     li a7, 4                  # Sử dụng syscall 4 (print string)
32     ecall                     # Thực hiện syscall để in chuỗi
33     j end                     # Nhảy tới kết thúc chương trình
34
35 overflow:
36     # In ra thông báo có tràn số
37     la a0, message_overflow    # Đưa địa chỉ chuỗi "Overflow detected!" vào a0
38     li a7, 4                  # Sử dụng syscall 4 (print string)
39     ecall                     # Thực hiện syscall để in chuỗi
40 end:
41     li a7, 10                 # Sử dụng syscall 10 (exit)
42     ecall                     # Thoát chương trình
```

```
.data
    message_no_overflow:  .asciz "No overflow detected.\n"
    message_overflow:     .asciz "Overflow detected.\n"
.text
    .globl _start
```

```

_start:
    # Giả sử chúng ta cần cộng hai số nguyên dương hoặc hai số nguyên âm, ở
    # đây sử dụng hai số nguyên dương.
    li s0, 0x7FFFFFFF # Toán hạng thứ nhất (số nguyên dương lớn nhất 32-
    bit)
    li s1, 1          # Toán hạng thứ hai (giá trị cần cộng)

    add t0, s0, s1      # t0 = s0 + s1

    # Kiểm tra tràn số
    # 1. Kiểm tra nếu s0 và s1 đều là số dương
    # 2. Nếu tổng t0 là số âm, điều đó có nghĩa là có tràn số dương.
    bltz s0, check_negative # If s0 < 0, jump check_negative
    bltz s1, check_negative # If s1 < 0, jump check_negative
    bltz t0, overflow       # If t0 < 0, jump overflow
    j no_overflow          # jump no_overflow

check_negative:
    # Kiểm tra nếu cả s0 và s1 đều là số âm
    # Nếu tổng là số dương, điều đó có nghĩa là có tràn số âm.
    bgez s0, no_overflow    # If s0 >= 0, jump no_overflow
    bgez s1, no_overflow    # If s1 >= 0, jump no_overflow
    bgez t0, overflow       # If t0 >= 0, jump overflow

no_overflow:
    # In ra thông báo không có tràn số
    la a0, message_no_overflow # Đưa địa chỉ chuỗi "No overflow
    detected!" vào a0
    li a7, 4                  # Sử dụng syscall 4 (print string)
    ecall                     # Thực hiện syscall để in chuỗi
    j end                     # Nhảy tới kết thúc chương trình

overflow:
    # In ra thông báo có tràn số
    la a0, message_overflow   # Đưa địa chỉ chuỗi "Overflow detected!"
    vào a0
    li a7, 4                  # Sử dụng syscall 4 (print string)
    ecall                     # Thực hiện syscall để in chuỗi
end:
    li a7, 10                 # Sử dụng syscall 10 (exit)

```

**Giải thích mã lệnh:**

- Khởi tạo hai toán hạng
  - *li s0, 0x7FFFFFFF*: Khởi tạo thanh ghi s0 với giá trị lớn nhất dương của số nguyên 32-bit (0x7FFFFFFF).
  - *li s1, 1*: Khởi tạo thanh ghi s1 với giá trị 1.
- Cộng hai toán hạng
  - Lệnh *add t0, s0, s1* thực hiện phép cộng hai số trong thanh ghi s0 và s1, kết quả được lưu vào thanh ghi t0.
- Kiểm tra tràn số dương
  - Nếu cả hai số trong s0 và s1 đều dương, ta dùng lệnh *bltz* để kiểm tra dấu của t0. Nếu t0 âm, thì tràn dương xảy ra, và chương trình sẽ nhảy tới nhãn *overflow* để in ra thông báo tràn.
- Kiểm tra tràn số âm
  - Nếu cả hai số trong s0 và s1 đều âm, ta kiểm tra tổng trong t0 là số dương. Nếu đúng, tràn số âm xảy ra, và chương trình sẽ nhảy tới nhãn *overflow*
- In thông báo
  - Nếu không có tràn số, chương trình sẽ nhảy tới nhãn *no\_overflow* và in ra thông báo “No overflow detected.”
  - Nếu có tràn số, chương trình sẽ nhảy tới nhãn *overflow* và in ra thông báo “Overflow detected.”
- Thoát chương trình
  - Chương trình kết thúc bằng lệnh *li a7, 10* và thực hiện *syscall* để thoát chương trình

**Kiểm tra tràn số với các trường hợp:**

- **Trường hợp cộng hai số dương lớn:** Khi  $s0 = 0x7FFFFFFF$  và  $s1 = 1$ , kết quả sẽ là tràn số dương và chương trình sẽ in ra thông báo “Overflow detected.”
- **Trường hợp cộng hai số âm lớn:** Nếu  $s0 = -2$  (ví dụ) và  $s1 = -2147483647$ , kết quả sẽ không có tràn vì tổng vẫn nhỏ hơn giá trị nhỏ nhất của số nguyên âm 32-bit.
- **Trường hợp cộng hai số trái dấu:** Nếu  $s0 = -10$  (ví dụ) và  $s1 = 9$ , kết quả sẽ không có tràn vì tổng vẫn nhỏ hơn giá trị nhỏ nhất của số nguyên âm 32-bit.

## Kết quả chạy chương trình

Hai số trái dấu			Cùng dấu tràn số			Cùng dấu không tràn số		
Name	Number	Value	Name	Number	Value	Name	Number	Value
zero	0	0x0000	zero	0	0x00000000	zero	0	0x00000000
ra	1	0x0000	ra	1	0x00000000	ra	1	0x00000000
sp	2	0x7fff	sp	2	0x7ffffeff	sp	2	0x7ffffeff
gp	3	0x1000	gp	3	0x10008000	gp	3	0x10008000
tp	4	0x0000	tp	4	0x00000000	tp	4	0x00000000
t0	5	0xffff	t0	5	0x00000000	t0	5	0x00000000
t1	6	0x0000	t1	6	0x00000000	t1	6	0x00000000
t2	7	0x0000	t2	7	0x00000000	t2	7	0x00000000
a0	8	0xffff	a0	8	0x7ffffeff	a0	8	0x00000000
a1	9	0x0000	a1	9	0x00000000	a1	9	0x00000001
a0	10	0x1001	a0	10	0x10010011	a0	10	0x10010000
a1	11	0x0000	a1	11	0x00000000	a1	11	0x00000000
a2	12	0x0000	a2	12	0x00000000	a2	12	0x00000000
a3	13	0x0000	a3	13	0x00000000	a3	13	0x00000000
a4	14	0x0000	a4	14	0x00000000	a4	14	0x00000000
a5	15	0x0000	a5	15	0x00000000	a5	15	0x00000000
a6	16	0x0000	a6	16	0x00000000	a6	16	0x00000000
a7	17	0x0000	a7	17	0x00000000	a7	17	0x00000000
a2	18	0x0000	a2	18	0x00000000	a2	18	0x00000000
a3	19	0x0000	a3	19	0x00000000	a3	19	0x00000000
a4	20	0x0000	a4	20	0x00000000	a4	20	0x00000000
a5	21	0x0000	a5	21	0x00000000	a5	21	0x00000000
a6	22	0x0000	a6	22	0x00000000	a6	22	0x00000000
a7	23	0x0000	a7	23	0x00000000	a7	23	0x00000000
a8	24	0x0000	a8	24	0x00000000	a8	24	0x00000000
a9	25	0x0000	a9	25	0x00000000	a9	25	0x00000000
s10	26	0x0000	s10	26	0x00000000	s10	26	0x00000000
s11	27	0x0000	s11	27	0x00000000	s11	27	0x00000000
t3	28	0x0000	t3	28	0x00000000	t3	28	0x00000000
t4	29	0x0000	t4	29	0x00000000	t4	29	0x00000000
t5	30	0x0000	t5	30	0x00000000	t5	30	0x00000000
t6	31	0x0000	t6	31	0x00000000	t6	31	0x00000000
pc		0x0040	pc		0x00400058	pc		0x00400054
No overflow detected.			Overflow detected.			No overflow detected.		
-- program is finished running (0) --			-- program is finished running (0) --			-- program is finished running (0) --		

## Assignment 5

Viết chương trình thực hiện nhân một số nguyên bất kỳ với một lũy thừa của 2 (2, 4, 8, 16, ...) mà không sử dụng lệnh nhân.

Ví dụ: Cho 2 thanh ghi  $t1 = 6$ ,  $t2 = 8$ . Yêu cầu viết chương trình tính tích của 2 thanh ghi này mà không sử dụng lệnh nhân.

## Nhập chương trình

```

1  .text
2      li t1, 6                # Số nguyên bất kì
3      li t0, 8                # Bậc của lũy thừa 2
4      li t2, 0                # t2 sẽ lưu số bit cần dịch
5  loop:
6      srli t0, t0, 1          # Dịch phải t0 một bit
7      beqz t0, endloop        # Nếu t0 == 0, thoát vòng lặp
8      addi t2, t2, 1          # Tăng số bit cần dịch
9      j loop
10 endloop:
11      sll s1, t1, t2          # Dịch trái t1 với số bit trong t2
12      # Kết quả của phép nhân sẽ được lưu trong thanh ghi s1

```

```

.text
    li t1, 6                # Số nguyên bất kì
    li t0, 8                # Bậc của lũy thừa 2
    li t2, 0                # t2 sẽ lưu số bit cần dịch

```

loop:

```
    srli t0, t0, 1      # Dịch phải t0 một bit
    beqz t0, endloop    # Nếu t0 == 0, thoát vòng lặp
    addi t2, t2, 1      # Tăng số bit cần dịch
    j loop
```

endloop:

```
    sll s1, t1, t2      # Dịch trái t1 với số bit trong t2
    # Kết quả của phép nhân sẽ được lưu trong thanh ghi s1
```

## Kết quả chạy

Name	Number	Value
zero	0	0
ra	1	0
sp	2	2147479548
gp	3	268468224
tp	4	0
t0	5	0
t1	6	6
t2	7	3
s0	8	0
s1	9	48
a0	10	0
a1	11	0
a2	12	0
a3	13	0
a4	14	0
a5	15	0
a6	16	0
a7	17	0
s2	18	0
s3	19	0
s4	20	0
s5	21	0
s6	22	0
s7	23	0
s8	24	0
s9	25	0
s10	26	0
s11	27	0
t3	28	0
t4	29	0
t5	30	0
t6	31	0
pc		4194340



## Giải thích kết quả chạy

- $t_0$  chứa số là lũy thừa của 2.
- Dùng vòng lặp để đếm số bit 1 trong  $t_0$ , tức là tính  $\log_2(t_0)$ .
- Dịch trái  $t_1$  với số bit tương ứng ( $t_2$ ).
- Ví dụ: Nếu  $t_1 = 6$  và  $t_0 = 8$  ( $2^3$ ), chương trình sẽ tính  $\log_2(8) = 3$ , rồi thực hiện  $6 \ll 3 = 48_{10}$ .
- Như vậy nếu muốn nhân với lũy thừa của 2, thì ra chỉ cần dịch trái  $n$  bit, thì sẽ trả về kết quả là nhân với  $2^n$

## Kết luận

Ứng dụng phép dịch bit để thực hiện phép nhân trong RISC-V có một số lợi ích so với sử dụng các lệnh nhân trong extension M (RV32M), bao gồm:

1. **Tiết kiệm tài nguyên phần cứng** – Không phải tất cả các vi xử lý RISC-V đều hỗ trợ extension M. Thay vì cần phần cứng chuyên dụng cho phép nhân, ta có thể sử dụng các phép dịch bit và cộng để thực hiện phép nhân trên các vi xử lý tối giản.
2. **Tối ưu hiệu năng trên vi xử lý không có extension M** – Nếu một vi xử lý không hỗ trợ RV32M, phép nhân sẽ phải được mô phỏng bằng phần mềm. Dùng phép dịch bit kết hợp với phép cộng có thể giúp tối ưu tốc độ so với việc sử dụng các thuật toán nhân tổng quát.
3. **Giảm chi phí năng lượng** – Các phép dịch bit thường tiêu tốn ít năng lượng hơn so với các lệnh nhân phần cứng. Điều này đặc biệt quan trọng với các thiết bị nhúng hoặc IoT, nơi tiết kiệm năng lượng là ưu tiên.
4. **Linh hoạt hơn trong tối ưu hóa thuật toán** – Khi sử dụng phép dịch bit, ta có thể tận dụng các kỹ thuật tối ưu như dịch trái thay vì nhân với 2, hoặc dịch phải thay vì chia cho 2, giúp tăng hiệu suất trong các thuật toán xử lý tín hiệu và đồ họa.

Tuy nhiên, nếu phần cứng có hỗ trợ extension M, việc sử dụng các lệnh nhân chuyên dụng sẽ nhanh và hiệu quả hơn so với phép dịch bit.