Họ và tên	Nguyễn Đình Dương
MSSV	20225966

BÁO CÁO COMPUTER ARCHITECHTURE TUẦN 4

Assignment 1

1. **Case 1:**

s1= 2147483647 → số dương lớn

s2 = 1 → số dương nhỏ

```
.text
        # Khởi tạo s1 và s2
        li s1, 2147483647
        li s2, 1

        # Thuật toán xác định điều kiện tràn số
        li t0, 0 # Mặc định không có tràn số
        add s3, s1, s2 # s3 = s1 + s2
        xor t1, s1, s2 # Kiểm tra xem s1 và s2 có cùng dấu hay không
        blt t1, zero, EXIT # Nếu không, thoát
        blt s1, zero, NEGATIVE # Kiểm tra xem s1 và s2 có âm không?
        bge s3, s1, EXIT # s1 và s2 dương
        # nếu s3 >= s1 thì kết quả không bị tràn số
        j OVERFLOW

NEGATIVE:
        bge s1, s3, EXIT # s1 và s2 âm
        # nếu s1 >= s3 thì kết quả không bị tràn số

OVERFLOW:
        li t0, 1 # Kết quả bị tràn số
```

t0	5	0x00000000
t1	6	0x00000000
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

s1= 2147483647

s2= 1

t0 = 0

+) s3 = s1 + s2 = 2147483647 + 1 =0x7fffffff

⇨ overflows và được lưu vào s3: 0x80000000 là số âm trong kiểu số nguyên có dấu 32-bit, cho thấy một hiện tượng tràn số đã xảy ra.

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
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

⇨

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, kết quả của phép cộng s1 + s2, có giá trị 0x80000000, là số âm trong hệ thống số nguyên có dấu 32-bit.
- t0 được đặt thành 1, cho biết rằng đã xảy ra hiện tượng tràn số.

2. Case 2: 2 negative number

s1= -2147483647

s2 = -1

```
.text
        # Khởi tạo s1 và s2
        li s1, -2147483647
        li s2, -1

        # Thuật toán xác định điều kiện tràn số
        li t0, 0 # Mặc định không có tràn số
        add s3, s1, s2 # s3 = s1 + s2
        xor t1, s1, s2 # Kiểm tra xem s1 và s2 có cùng dấu hay không
        blt t1, zero, EXIT # Nếu không, thoát
        blt s1, zero, NEGATIVE # Kiểm tra xem s1 và s2 có âm không?
        bge s3, s1, EXIT # s1 và s2 dương
        # nếu s3 >= s1 thì kết quả không bị tràn số
        j OVERFLOW

NEGATIVE:
        li, t0, 2
        bge s1, s3, EXIT # s1 và s2 âm
        # nếu s1 >= s3 thì kết quả không bị tràn số

OVERFLOW:
        li t0, 1 # Kết quả bị tràn số
EXIT:
```

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000002
t1	6	0x7ffffffe
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

Kết quả:

- Kết quả của phép cộng: s1 + s2 = -2147483648.
- Không có tràn số (giá trị t0=0).
- t2: 0x000002: nghĩa là 2 số âm

3. Case 3: Tổng bằng 0
   S1 = 69
   S2 = -69

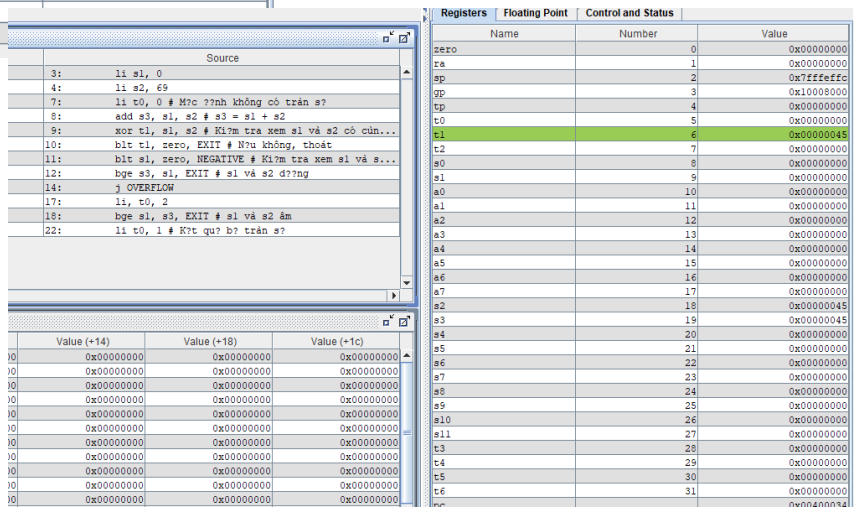
Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0xfffffffe
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000045
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0xffffffbb
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		
pc		

Kết quả là chương trình thoát khi gặp câu lệnh:

blt t1, zero, EXIT # Nếu không, thoát

4. Một số là số 0

s1= 0
s2 = 69



Chương trình không bị tràn số và thoát khi gặp lệnh:
bge s3, s1, EXIT # s1 và s2 dương
 # nếu s3 >= s1 thì kết quả không bị tràn số

Assignment 2

Viết chương trình thực hiện các công việc sau:

- Trích xuất MSB (byte có trọng số cao) của thanh ghi s0
- Xóa LSB (byte có trọng số thấp) của thanh ghi s0
- Thiết lập LSB của thanh ghi s0 (đặt các bit từ 7 đến 0 thành 1)
- Xóa thanh ghi s0 bằng các lệnh logic (đặt s0 = 0)

```
.text
        li    s0, 0x12345678      # Giá trị mẫu cho s0
        srli  t0, s0, 24          # Trích xuất MSB (dịch phải 24 bit)
        andi  t1, s0, 0xFFFFFF00  # Xóa LSB (AND với 0xFFFFFF00 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 (đặt s0 = 0)
```

- **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):

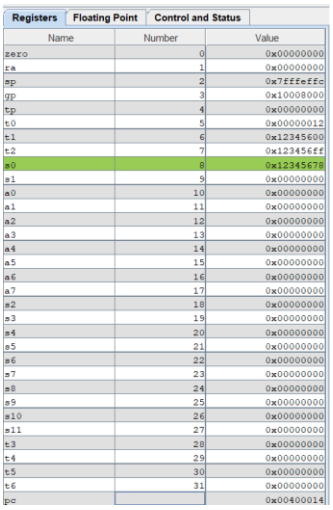
  - Lệnh andi t1, s0, 0xFFFFFF00 sẽ giữ nguyên tất cả các bit ngoại trừ 8 bit thấp nhất (LSB) bằng cách AND với 0xFFFFFF00.
  - Giá trị ban đầu của s0 là 0x12345678, khi AND với 0xFFFFFF00, kết quả sẽ là 0x12345600.

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



Registers | Floating Point | Control and Status

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	0x12345678
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		0x00400014

Assignment 3

1. **Giả lệnh neg s0, s1 (s0 = -s1):**
   Lệnh này 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 phép lấy giá trị âm là sử dụng lệnh sub với toán hạng x0 (thanh ghi không) để trừ giá trị của s1 từ 0.

```
sub s0, x0, s1  # s0 = 0 - s1
```

2. **Giả lệnh mv s0, s1 (s0 = s1):**
   Lệnh này 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.

```
addi s0, s1, 0  # s0 = s1 + 0 (sao chép giá trị từ s1 sang s0)
```

3. **Giả lệnh not s0 (s0 = bit_invert(s0))**:
   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).

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

4. **Giả lệnh ble s1, s2, label (nhảy tới label nếu s1 <= s2)**:
   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 >= s1, tương đương với s1 <= s2.

```
bge s2, s1, label  # Nếu s2 >= s1 (tương đương s1 <= s2), nhảy tới label
```

Assignment 4

```
    .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 dương hoặc hai số âm, ở đây sử dụng hai số dương.
    li    s0, 0x7FFFFFFF   # Toán hạng thứ nhất (số lớn nhất dương)
    li    s1, 1            # Toán hạng thứ hai (giá trị cần cộng)

    # Cộng hai số
    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  # Nếu s0 là số âm, chuyển tới kiểm tra số âm
    bltz  s1, check_negative  # Nếu s1 là số âm, chuyển tới kiểm tra số âm
```

```
    bltz  t0, overflow       # Nếu tổng là âm mà các số hạng đều dương, có tràn dương
    j     no_overflow        # Nếu không, không có tràn

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    # Nếu s0 là số dương, không có tràn số
    bgez  s1, no_overflow    # Nếu s1 là số dương, không có tràn số
    bgez  t0, overflow       # Nếu tổng là dương mà các số hạng đều âm, có tràn âm

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)
    ecall                    # Thoát chương trình
```

Giải thích mã lệnh:

1. **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.
2. **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 t0.
3. **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.
4. **Kiểm tra tràn số âm**:
    - Nếu cả hai số trong s0 và s1 đều âm, ta kiểm tra nếu tổng trong t0 là số dương. Nếu đúng, tràn âm xảy ra, và chương trình sẽ nhảy tới nhãn overflow.

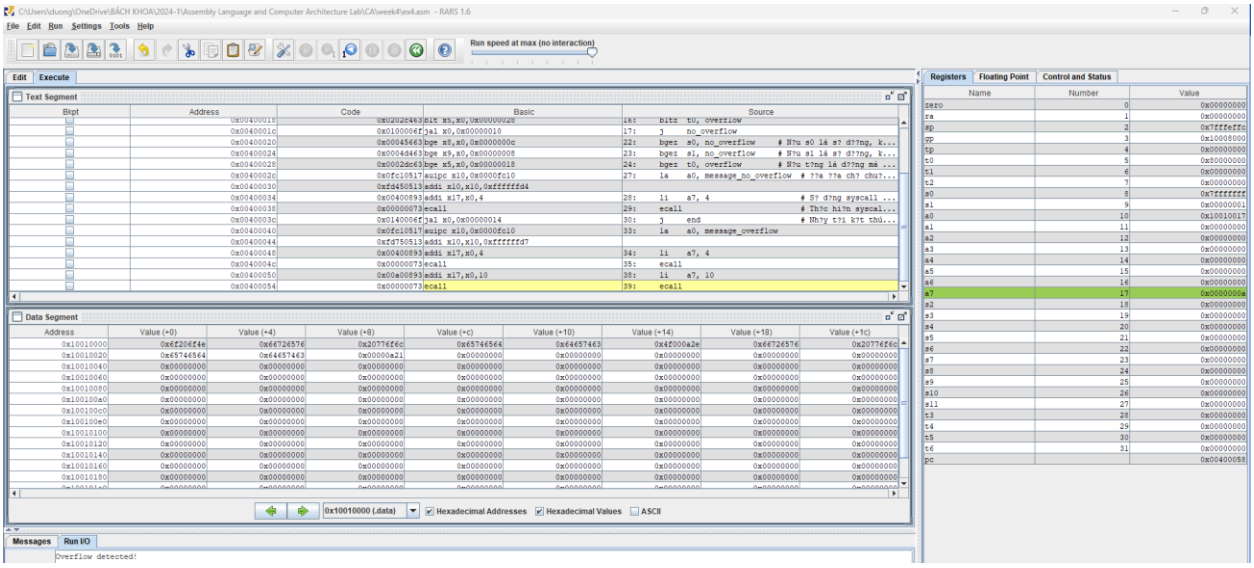
5. **In thông báo**:
    - Nếu không có tràn số, chương trình nhảy tới nhãn no_overflow và in ra thông báo "No overflow detected."
    - Nếu có tràn số, chương trình nhảy tới nhãn overflow và in ra thông báo "Overflow detected!"
6. **Thoát chương trình**:
    - Chương trình kết thúc bằng lệnh li a7, 10 và thực hiện syscall ecall để thoát chương trình.

Kiểm tra tràn số với các trường hợp:

- **Trường hợp cộng hai số dương lớn**: Khi s0 = 0x7FFFFFFF (giá trị lớn nhất dương) và s1 = 1, kết quả sẽ là tràn dương và chương trình sẽ in 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.

Kết quả khi chạy chương trình:

- Nếu có tràn số: **"Overflow detected!"**.



-

```
Overflow detected!


-- program is finished running (0) --
```

Assignment 5

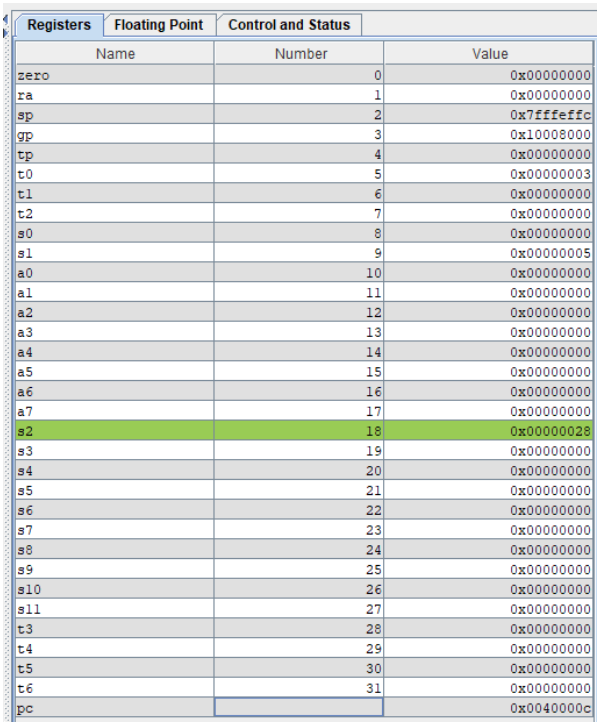
```
# RISC-V assembly code để nhân một số với lũy thừa của 2 bằng cách sử dụng dịch trái

    li x1, 5          # x1 = 5
    li t0, 3          # t0 = 3 (để thực hiện phép nhân với 2^3 = 8)

    sll x2, x1, t0    # Dịch trái giá trị trong thanh ghi x1 với số bit bằng giá trị trong t0 (3 bit)
                      # Việc này tương đương với phép nhân x1 với 2^3 (5 * 8 = 40)

    # Sau khi thực hiện lệnh trên, thanh ghi x2 sẽ chứa kết quả của phép nhân x1 * 8
    # (kết quả là 40 vì 5 * 8 = 40)
```

Kết quả:



Registers | Floating Point | Control and Status

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffeffc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000003
t1	6	0x00000000
t2	7	0x00000000
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	0x00000028
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

s2: 0x00000028: đổi sang decimal là 40, đúng với dự tính từ đầu

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