

BÁO CÁO THỰC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN 4

Họ và tên: Phan Khánh Vũ

MSSV: 20235880

1. Assignment 1:

Nhập chương trình sau: (TH cộng 2 số dương có tràn)

```
lab03_1.asm*
1  # Laboratory Exercise 4, Home Assignment 1
2  .text
3  li s1 1000000000    # Khởi tạo giá trị cho s1
4  li s2 2000000000    # Khởi tạo giá trị cho s2
5  # Thuật toán xác định tràn số
6  li t0, 0            # Mặc định không có tràn số
7  add s3, s1, s2      # s3 = s1 + s2
8  xor t1, s1, s2      # Kiểm tra s1 với s2 có cùng dấu
9  blt t1, zero, EXIT  # Nếu t1 là số âm, s1 và s2 khác dấu
10 blt s1, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm
11 bge s3, s1, EXIT    # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
12 # Nếu s3 >= s1, không tràn số
13 j OVERFLOW
14 NEGATIVE:
15 bge s1, s3, EXIT    # s1 âm, kiểm tra s3 có lớn hơn s1 không
16 # Nếu s1 >= s3, không tràn số
17 OVERFLOW:
18 li t0, 1            # The result is overflow
19 EXIT:
20
```

- Các biến khởi tạo:

- Khởi tạo giá trị của s1 = 1000000000
- Khởi tạo giá trị của s2 = 2000000000

- Các bước thực hiện của chương trình:

- Câu lệnh li s1 1000000000 để gán giá trị cho s1 = 1000000000
- Câu lệnh li s2 2000000000 để gán giá trị cho s2 = 2000000000
- Câu lệnh li t0, 0 để gán giá trị cho t0 = 0. Mặc định không có hiện tượng tràn số.
- Câu lệnh add s3, s1, s2 để tính tổng s3 = s1 + s2;
- Câu lệnh xor t1, s1, s2 để kiểm tra xem s1 và s2 có cùng dấu hay không.
- Nếu s1 và s2 khác dấu thì trả về t1 có kết quả âm.
- Câu lệnh blt t1, zero, EXIT để so sánh t1 với zero. Nếu t1 < 0 thì s1 và s2 khác dấu điều kiện đúng thì nhảy đến thẻ EXIT.
- Câu lệnh blt s1, zero, NEGATIVE để so sánh s1 với zero. Nếu s1 < 0 thì s1 và s2 cùng âm và nhảy đến thẻ NEGATIVE.

- Câu lệnh bge s3, s1, EXIT để kiểm tra xem s3 có nhỏ hơn s1 hay không. Do đã kiểm tra điều kiện 2 số cùng dương nên nếu s3 nhỏ hơn s1 thì xảy ra hiện tượng tràn số.
- Câu lệnh j OVERFLOW dùng để nhảy đến thẻ OVERFLOW. Khi trường hợp tràn số xảy ra sẽ nhảy đến OVERFLOW và thực hiện thay đổi giá trị của t0.
- Câu lệnh bge s1, s3, EXIT dùng để so sánh s1 có nhỏ hơn s3 hay không.
- Do đã kiểm tra điều kiện 2 số cùng âm nên nếu s1 nhỏ hơn s3 thì xảy ra hiện tượng tràn số.
- Câu lệnh li t0, 1 dùng để thay đổi giá trị của t0 = 1. Được đặt trong thẻ OVERFLOW, khi có hiện tượng tràn số xảy ra thì thay đổi giá trị của t0

Quan sát sự thay đổi của thanh ghi:

Name	Number	Value	Name	Number	Value	Name	Number	Value
zero	0	0x00000000	zero	0	0x00000000	zero	0	0x00000000
ra	1	0x00000000	ra	1	0x00000000	ra	1	0x00000000
sp	2	0x7fffffc0	sp	2	0x7fffffc0	sp	2	0x7fffffc0
gp	3	0x10008000	gp	3	0x10008000	gp	3	0x10008000
tp	4	0x00000000	tp	4	0x00000000	tp	4	0x00000000
t0	5	0x00000000	t0	5	0x00000000	t0	5	0x00000000
t1	6	0x00000000	t1	6	0x00000000	t1	6	0x00000000
t2	7	0x00000000	t2	7	0x00000000	t2	7	0x00000000
s0	8	0x00000000	s0	8	0x00000000	s0	8	0x00000000
s1	9	0x3b9aca00	s1	9	0x3b9aca00	s1	9	0x3b9aca00
a0	10	0x00000000	a0	10	saved temporary (preserved across call)	a0	10	0x00000000
a1	11	0x00000000	a1	11	0x00000000	a1	11	0x00000000
a2	12	0x00000000	a2	12	0x00000000	a2	12	0x00000000
a3	13	0x00000000	a3	13	0x00000000	a3	13	0x00000000
a4	14	0x00000000	a4	14	0x00000000	a4	14	0x00000000
a5	15	0x00000000	a5	15	0x00000000	a5	15	0x00000000
a6	16	0x00000000	a6	16	0x00000000	a6	16	0x00000000
a7	17	0x00000000	a7	17	0x00000000	a7	17	0x00000000
a2	18	0x00000000	a2	18	0x00000000	a2	18	0x77359400
s3	19	0x00000000	s3	19	0x00000000	s3	19	0x00000000
s4	20	0x00000000	s4	20	0x00000000	s4	20	0x00000000
s5	21	0x00000000	s5	21	0x00000000	s5	21	0x00000000
s6	22	0x00000000	s6	22	0x00000000	s6	22	0x00000000
s7	23	0x00000000	s7	23	0x00000000	s7	23	0x00000000
s8	24	0x00000000	s8	24	0x00000000	s8	24	0x00000000
s9	25	0x00000000	s9	25	0x00000000	s9	25	0x00000000
s10	26	0x00000000	s10	26	0x00000000	s10	26	0x00000000
s11	27	0x00000000	s11	27	0x00000000	s11	27	0x00000000
t3	28	0x00000000	t3	28	0x00000000	t3	28	0x00000000
t4	29	0x00000000	t4	29	0x00000000	t4	29	0x00000000
t5	30	0x00000000	t5	30	0x00000000	t5	30	0x00000000
t6	31	0x00000000	t6	31	0x00000000	t6	31	0x00000000
pc		0x00400000	pc		0x00400000	pc		0x00400000

Name	Number	Value	Name	Number	Value	Name	Number	Value
zero	0	0x00000000	zero	0	0x00000000	zero	0	0x00000000
ra	1	0x00000000	ra	1	0x00000000	ra	1	0x00000000
sp	2	0x7fffffc0	sp	2	0x7fffffc0	sp	2	0x7fffffc0
gp	3	0x10008000	gp	3	0x10008000	gp	3	0x10008000
tp	4	0x00000000	tp	4	0x00000000	tp	4	0x00000000
t0	5	0x00000000	t0	5	0x00000000	t0	5	0x00000001
t1	6	0x00000000	t1	6	0x4caf5e00	t1	6	0x4caf5e00
t2	7	0x00000000	t2	7	0x00000000	t2	7	0x00000000
s0	8	0x00000000	s0	8	0x00000000	s0	8	0x00000000
s1	9	0x3b9aca00	s1	9	0x3b9aca00	s1	9	0x3b9aca00
a0	10	0x00000000	a0	10	0x00000000	a0	10	0x00000000
a1	11	0x00000000	a1	11	0x00000000	a1	11	0x00000000
a2	12	0x00000000	a2	12	0x00000000	a2	12	0x00000000
a3	13	0x00000000	a3	13	0x00000000	a3	13	0x00000000
a4	14	0x00000000	a4	14	0x00000000	a4	14	0x00000000
a5	15	0x00000000	a5	15	0x00000000	a5	15	0x00000000
a6	16	0x00000000	a6	16	0x00000000	a6	16	0x00000000
a7	17	0x00000000	a7	17	0x00000000	a7	17	0x00000000
a2	18	0x77359400	a2	18	0x77359400	a2	18	0x77359400
s3	19	0xb2d05e00	s3	19	0xb2d05e00	s3	19	0xb2d05e00
s4	20	0x00000000	s4	20	0x00000000	s4	20	0x00000000
s5	21	0x00000000	s5	21	0x00000000	s5	21	0x00000000
s6	22	0x00000000	s6	22	0x00000000	s6	22	0x00000000
s7	23	0x00000000	s7	23	0x00000000	s7	23	0x00000000
s8	24	0x00000000	s8	24	0x00000000	s8	24	0x00000000
s9	25	0x00000000	s9	25	0x00000000	s9	25	0x00000000
s10	26	0x00000000	s10	26	0x00000000	s10	26	0x00000000
s11	27	0x00000000	s11	27	0x00000000	s11	27	0x00000000
t3	28	0x00000000	t3	28	0x00000000	t3	28	0x00000000
t4	29	0x00000000	t4	29	0x00000000	t4	29	0x00000000
t5	30	0x00000000	t5	30	0x00000000	t5	30	0x00000000
t6	31	0x00000000	t6	31	0x00000000	t6	31	0x00000000
pc		0x00400018	pc		0x00400020	pc		0x00400034

Nhận thấy t0 = 1 như kỳ vọng

Các trường hợp khác:

1. $s1 = 0x10000000$ và $s2 = 0x01000000$ (cộng 2 số dương không tràn)

```

1 # Laboratory Exercise 4, Home Assignment 1
2 .text
3 li $t0, 0x10000000 # Khởi tạo giá trị cho x1
4 li $2, 0x10000000 # Khởi tạo giá trị cho x2
5 # Thuật toán xác định tính số
6 li $t0, 0 # Mục đích không có tràn số
7 and $3, $1, $2 # $3 = $1 & $2
8 or $t1, $1, $2 # Kiểm tra $1 với $2 có cùng dấu
9 bit $t1, zero, FMT # Nếu $1 là số âm, $1 và $2 khác dấu
10 bit $1, $2, 0x20, NEGATIVE # Kiểm tra $1 và $2 là số âm hay không âm
11 bne $1, $1, EXIT # $1 không âm, kiểm tra $1 nhỏ hơn $1 không
12 # Nếu $3 >= $1, không tràn số
13 j OVERFLOW
14 NEGATIVE:
15 bge $1, $3, FMT # $1 âm, kiểm tra $1 có lớn hơn $1 không
16 # Nếu $1 >= $3, không tràn số
17 OVERFLOW:
18 li $t0, 1 # The result is overflow
19 EXIT:
20

```

Kết quả: $t_0 = 0$ như kỳ vọng

2. $s1 = 0x80000000$ và $s2 = 0x01000000$ (cộng số âm với số dương)

```

lab03_1.asm
1  # Laboratory Exercise 4, Homework Assignment 1
2  .text
3  li $t0, 0x07000000    # Khởi tạo giá trị cho x1
4  li $t1, 0x01000000    # Khởi tạo giá trị cho x2
5  # Thuật toán xác định trên số
6  li $t2, 0             # Mục đích không rõ trên số
7  and $t3, $t1, $t2     # $t3 = $t1 & $t2
8  xor $t1, $t1, $t2     # Kiểm tra $t1 với $t2 có cùng dấu
9  blt $t1, $zero, EXIT   # Nếu $t1 là số âm, $t1 và $t2 khác dấu
10 blt $t1, $zero, NEGATIVE # Kiểm tra $t1 và $t2 là số âm hay không âm
11 bge $t3, $t1, EXIT     # $t3 không âm, kiểm tra $t2 nhỏ hơn $t1 không
12 # Nếu $t3 >= $t1, không trên số
13 # OVERFLOW
14 HALT($t3);
15 bne $t1, $t3, EXIT     # $t1 âm, kiểm tra $t2 có lớn hơn $t1 không
16 # Nếu $t3 >= $t3, không trên số
17 OVERFLOW:
18 li $t0, 1             # The result is overflow
19 EXIT:
20

```

Name	Number	Value
\$t0	0	0x00000000
\$t1	1	0x00000000
\$t2	2	0x7fffff00
\$t3	3	0x10000000
\$t4	4	0x00000000
\$t5	5	0x00000000
\$t6	6	0x81000000
\$t7	7	0x00000000
\$t8	8	0x00000000
\$t9	9	0x00000000
\$t10	10	0x00000000
\$t11	11	0x00000000
\$t12	12	0x00000000
\$t13	13	0x00000000
\$t14	14	0x00000000
\$t15	15	0x00000000
\$t16	16	0x00000000
\$t17	17	0x00000000
\$t18	18	0x00000000
\$t19	19	0x00000000
\$t20	20	0x00000000
\$t21	21	0x00000000
\$t22	22	0x00000000
\$t23	23	0x00000000
\$t24	24	0x00000000
\$t25	25	0x00000000
\$t26	26	0x00000000

Kết quả : $t_0 = 0$ như kỳ vọng

3. $s1 = 0xF0000000$ và $s2 = 0xA0000000$ (cộng 2 số âm không tràn)

	lab03_01.asm		Name	Number	Value
1	# Laboratory Exercise 4, Homework Assignment 1		ZERO	0	0x00000000
2	.text		ra	1	0x00000000
3	li \$t, 0x20000000 # Khởi tạo giá trị cho s1		\$E	2	0x00000000
4	li \$t2, 0xA2000000 # Khởi tạo giá trị cho s2		\$Z	3	0x10000000
5	# Thuật toán xác định trên số		tf	4	0x00000000
6	li t0, 0 # Mạo diện không có trên số		t0	5	0x00000000
7	add \$t, \$t, \$t2 # \$t = \$t + \$t2		L1	6	0x50000000
8	sbc \$t, \$t, \$t2 # Kiểm tra s1 với s2 có cùng dấu		L2	7	0x00000000
9	bne \$t, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác nhau		\$0	8	0x00000000
10	blt \$t, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm		a1	9	0xf0000000
11	bge \$t, \$t, EXIT # s1 không âm, kiểm tra s2 nhỏ hơn s1 không		a0	10	0x00000000
12	# Nếu s3 >= s1, không trên số		a1	11	0x00000000
13	OVERFLOW		a2	12	0x00000000
14	NEGATIVE:		a3	13	0x00000000
15	bge \$t, \$t, \$t, EXIT # s1 âm, kiểm tra s3 có lớn hơn s1 không		a4	14	0x00000000
16	# Nếu s3 >= s2, không trên số		a5	15	0x00000000
17	OVERFLOW:		a6	16	0x00000000
18	li t0, 1 # The result is overflow		a7	17	0x00000000
19	EXIT:		a2	18	0xa0000000
20			a3	19	0x90000000
			a4	20	0x00000000
			a5	21	0x00000000
			a6	22	0x00000000
			a7	23	0x00000000
			a8	24	0x00000000
			a9	25	0x00000000
			a10	26	0x00000000

Kết quả: $t_0 = 0$ như kỳ vọng

4. $s1 = 0x90000000$ và $s2 = 0x99999999$ (cộng 2 số âm tràn)

lab03_1.asm	Name	Number	Value
1 # Laboratory Exercise 3, Homework Assignment 1	KMEM	0	0x00000000
2 .lword	ED	1	0x00000000
3 li s0, 0x90000000 # Khởi tạo giá trị cho s0	BP	2	0x7ffffc00
4 li s2, 0x99999999 # Khởi tạo giá trị cho s2	GP	3	0x10000000
5 # Thuật toán ước định tràn số	FP	4	0x00000000
6 li t0, 0 # Mặc định không có tràn số	TD	5	0x00000001
7 add s3, s1, s2 # s3 = s1 + s2	L1	6	0x09999999
8 xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu	L2	7	0x00000000
9 bit t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu	s0	8	0x00000000
10 bll s1, <zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm	s1	9	0x90000000
11 bge s3, s1, EXIT # s1 không âm, kiểm tra s3 nhỏ hơn s1 không	a0	10	0x00000000
12 # Nếu s3 >= s1, không tràn số	a1	11	0x00000000
13 OVFLOW	a2	12	0x00000000
14 NRIADJUST:	a3	13	0x00000000
15 bge s1, s3, EXIT # s1 âm, kiểm tra s3 có lớn hơn s1 không	a4	14	0x00000000
16 # Nếu s1 >= s3, không tràn số	a5	15	0x00000000
17 OVFLOW:	a6	16	0x00000000
18 li t0, 1 # Cho result is overflow	a7	17	0x00000000
19 EXIT:	a8	18	0x99999999
20	a9	19	0x79999999
	a4	20	0x00000000
	a5	21	0x00000000
	a6	22	0x00000000
	a7	23	0x00000000
	a8	24	0x00000000
	a9	25	0x00000000
	a10	26	0x00000000

Kết quả: t0 = 1 như kỳ vọng

2. Assignment 2:

Nhập chương trình sau:

lab03_1.asm	lab03_2.asm*
1 .text	
2 li s0, 0x20235658 # Khởi tạo giá trị cho s0 = 0x20235658	
3	
4 # 1. Trích xuất MSB (Most Significant Byte)	
5 srli t0, s0, 24 # Dịch phải 24 bit để lấy byte cao nhất (MSB)	
6	
7 # 2. Xóa LSB (Least Significant Byte)	
8 li t1, 0xFFFFF00 # Khởi tạo giá trị để xóa LSB	
9 and s0, s0, t1 # Xóa LSB của s0	
10	
11 # 3. Thiết lập LSB thành 0xFF (tất cả bit của byte thấp nhất là 1)	
12 ori s0, s0, 0xFF # OR với 0xFF để thiết lập byte thấp nhất	
13	
14 # 4. Xóa thanh ghi s0 bằng cách sử dụng các lệnh logic	
15 xor s0, s0, s0 # Xóa toàn bộ nội dung của s0 (s0 = 0)	
16	
17 END:	

- **Khởi tạo giá trị:**
 - Giá trị của s0 = 0x20235658
 - Giá trị của t1 = 0xFFFFF00
- **Các bước thực hiện của chương trình:**
 - Câu lệnh li s0, 0x20235658 để khởi tạo giá trị cho s0 = 0x20235658
 - Câu lệnh srli t0, s0, 24 dùng để dịch phải 24 bit để lấy byte cao nhất. Do dịch phải 24 bit để giữ lại 2 bit cao nhất lưu vào t0;
 - Câu lệnh li t1, 0xFFFFF00 để khởi tạo giá trị cho t1 = 0xFFFFF00
 - Câu lệnh and s0, s0, t1 để xóa LSB của s0
 - Câu lệnh ori s0, s0, 0xFF để thiết lập byte bé nhất
 - Câu lệnh xor s0, s0, s0 dùng để xóa toàn bộ nội dung của s0 (s0 = 0) thông qua phép logic xor.

Kết quả chạy của chương trình của mỗi phần

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0	0x00000000	
ra	1	0x00000000	
sp	2	0x7ffffefc	
gp	3	0x10008000	
tp	4	0x00000000	
t0	5	0x00000020	
t1	6	0x00000000	
t2	7	0x00000000	
s0	8	0x20235658	
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	

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0	0x00000000	
ra	1	0x00000000	
sp	2	0x7ffffefc	
gp	3	0x10008000	
tp	4	0x00000000	
t0	5	0x00000020	
t1	6	0xffffffff00	
t2	7	0x00000000	
s0	8	0x20235600	
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	

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0	0x00000000	
ra	1	0x00000000	
sp	2	0x7ffffefc	
gp	3	0x10008000	
tp	4	0x00000000	
t0	5	0x00000020	
t1	6	0xffffffff00	
t2	7	0x00000000	
s0	8	0x202356ff	
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		0x00400018	

Registers	Floating Point	Control and Status	
Name	Number	Value	
zero	0	0x00000000	
ra	1	0x00000000	
sp	2	0x7ffffefc	
gp	3	0x10008000	
tp	4	0x00000000	
t0	5	0x00000020	
t1	6	0xffffffff00	
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	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		0x00400020	

3. Assignment 3:

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

- Nhập chương trình:

```
.text
li s1 0x20235658
sub s0, zero, s1
```

- Các bước thực hiện

- Câu lệnh li s0 0x20235658 để khởi tạo giá trị cho s0
- Câu lệnh sub s0, zero, s1 để thực hiện phép trừ $s0 = 0 - s1$

- Kết quả thực hiện

s0	8	0x20235658
s1	9	0xdfdca9a8

lab03_1.asm	lab03_2.asm*	lab03_3.asm
1	.text	
2	li s1 0x20235658	
3	sub s0, zero, s1	
4		

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0xffffffff
fp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
a0	8	0xdfdca9a8

b. mv s0, s1 (s0 = s1)

- Nhập chương trình:

```
.text
li s1 0x20235658
add s0, zero, s1
```

- Các bước thực hiện

- + Câu lệnh li s0 0x20235658 để khởi tạo giá trị cho s0
- + Câu lệnh add s0, zero, s1 để thực hiện phép trừ $s0 = 0 + s1$

- Kết quả thực hiện:

s0	8	0x20235658
s1	9	0x20235658

lab03_1.asm	lab03_2.asm*	lab03_3.asm
1	.text	
2	li s1 0x20235658	
3	add s0, zero, s1	
4		

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0xffffffff
fp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
a0	8	0x20235658
s1	9	0x20235658

c. not s0 (s0 = bit_invert(s0))

- Nhập chương trình:

```
.text
li s0 0x20235658
xori s1, s0, 0xFFFFFFFF
```

- Các bước thực hiện:

- Câu lệnh li s0 0x20235658 để khởi tạo giá trị cho s0

- Câu lệnh xori s1, s0, 0xFFFFFFFF để xor s0 với 0xFFFFFFFF

- **Kết quả thực hiện:**

s0	8	0x20235658
s1	9	0xdfdca9a7

lab03_1.asm	lab03_2.asm*	lab03_3.asm
1 .text		
2 li s0 0x20235658		
3 xori s1, s0, 0xFFFFFFFF		
4		

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffffc
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x20235658
s1	9	0xdfdca9a7

d. ble s1, s2, label (if s1 <= s2 j label)

- **Nhập chương trình:**

```
.text
    li s1 100
    li s2 200
    li s0 0
    bge s2, s1, else
then:
    j end
else:
    li s0 1
end:
```

- **Các bước thực hiện:**

- Câu lệnh li s1 100 để gán giá trị s1 = 100
- Câu lệnh li s2 200 để gán giá trị s2 = 200
- Câu lệnh li s0 0 để gán giá trị s0 = 0
- Câu lệnh bge s2, s1, else để so sánh nếu s2 >= s1 thì nhảy đến thẻ else
- Câu lệnh j end để nhảy đến thẻ end
- Câu lệnh li s0 1 để gán giá trị của s0 = 1 nhằm xác định xem nếu s2 >= s1 thì đổi giá trị của s0 = 1;

- **Kết quả thực hiện của chương trình:**

c2	7	0x00000000
s0	8	0x00000001
s1	9	0x00000064
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	0x000000c8

lab03_1.asm	lab03_2.asm*	lab03_3.asm	Name	Number	Value
1	.text		zero	0	0x00000000
2	li s1 100		ea	1	0x00000000
3	li s2 200		fp	2	0xffffffff
4	li s3 0		gp	3	0x10000000
5	bge s2, s1, else		tp	4	0x00000000
6	thont		t0	5	0x00000000
7	li s0 1		t1	6	0x00000000
8	endz		t2	7	0x00000000
9			a0	8	0x00000001
10			a1	9	0x00000064
11			a2	10	0x00000000
			a3	11	0x00000000
			a4	12	0x00000000
			a5	13	0x00000000
			a6	14	0x00000000
			a7	15	0x00000000
			a8	16	0x00000000
			a9	17	0x00000000
			a10	18	0x000000c8
			a11	19	0x00000000

4. Assignment 4:

- Ý tưởng thuật toán:

- o Cộng hai số s1 và s2.
- o Nếu s1 và s2 khác dấu, không thể tràn số → thoát.
- o Nếu s1 và s3 khác dấu, tràn số xảy ra.
- o Gán t0 = 1 nếu có tràn số, ngược lại t0 = 0.

Mã code: (TH cộng 2 số dương tràn số)

Laboratory Exercise 4, Home Assignment 4

.text

TODO: Thiết lập giá trị cho s1 và s2

li s1 0x70000000

li s2 0x78888888

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

xor t2 s1, s3 # Kiểm tra s1 với s3 có cùng dấu không

blt t2, zero, OVERFLOW # khác dấu thì tràn số

j EXIT

OVERFLOW:

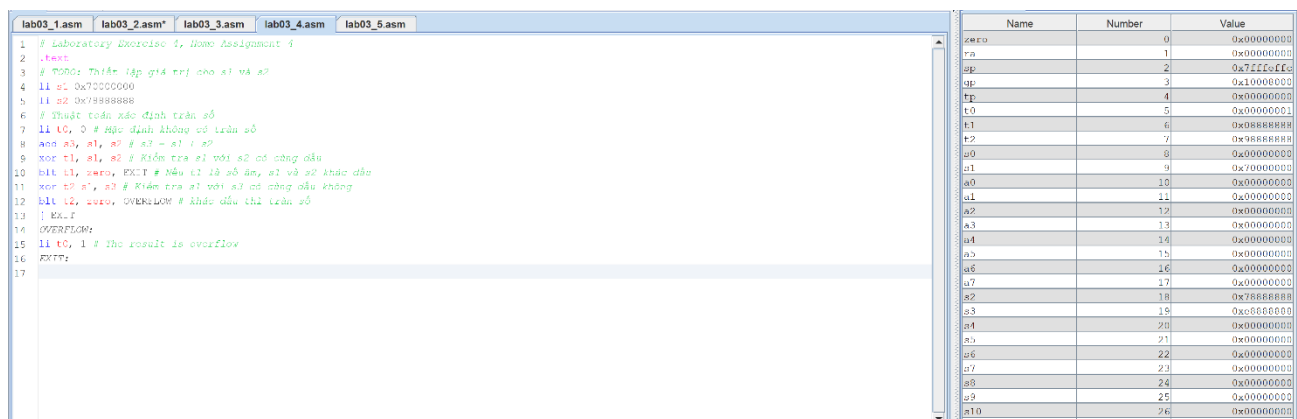
li t0, 1 # The result is overflow

EXIT:

Chú thích:

Chương trình RISC-V này kiểm tra tràn số khi cộng hai số nguyên có dấu. Đầu tiên, nó nạp hai số s1 và s2, sau đó thực hiện phép cộng $s3 = s1 + s2$. Nếu s1 và s2 khác dấu, tràn số không thể xảy ra, nên chương trình thoát ngay. Nếu chúng cùng dấu, chương trình kiểm tra xem s3 có cùng dấu với s1 không. Nếu khác dấu, nghĩa là phép cộng đã gây tràn số, và t0 được gán giá trị 1 để đánh dấu.

Nhập chương trình trên vào và chạy:



The screenshot shows a RISC-V assembly editor with the following code:

```
1 # Laboratory Exercise 1, Homework Assignment 1
2 .text
3 # TODO: Thiết lập giá trị cho s1 và s2
4 li s1, 0x70000000
5 li s2, 0x88888888
6 # Thuật toán xác định tràn số
7 li t0, 0 # Hạng định không có tràn số
8 add s3, s1, s2 # s3 = s1 + s2
9 xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu
10 blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu
11 xor t2, s1, s3 # Kiểm tra s1 với s3 có cùng dấu không
12 blt t2, zero, OVERFLOW # khác dấu thì tràn số
13 j EXIT
14 OVERFLOW:
15 li t0, 1 # The result is overflow
16 EXIT:
17
```

The register window on the right shows the following values:

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffff000
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000001
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x70000000
s2	10	0x88888888
s3	11	0x00000000
a0	12	0x00000000
a1	13	0x00000000
a2	14	0x00000000
a3	15	0x00000000
a4	16	0x00000000
a5	17	0x00000000
a6	18	0x00000000
a7	19	0x00000000
s2	20	0x00000000
s3	21	0x00000000
a1	22	0x00000000
a2	23	0x00000000
a3	24	0x00000000
a4	25	0x00000000
a5	26	0x00000000

Kết quả: t0 = 1 như kỳ vọng.

Các trường hợp khác:

1.s1 = 0x70000000 và s2 = 0x88888888(cộng số âm với số dương)

lab03_1.asm	lab03_2.asm*	lab03_3.asm	lab03_4.asm	lab03_5.asm
<pre> 1 # Laboratory Exercise 4, Home Assignment 4 2 .text 3 # TODO: Thiết lập giá trị cho s1 và s2 4 li s1 0x00000000 5 li s2 0x88888888 6 # Thuật toán xác định trên số 7 li t0, 0 # Mặc định không có tràn số 8 add s3, s1, s2 # s3 = s1 + s2 9 xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu 10 blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu 11 xor t2 s1, s2 # Kiểm tra s1 với s2 có cùng dấu không 12 bne t2, zero, OVERFLOW # khác dấu thì tràn số 13 EXIT: 14 OVERFLOW: 15 li t0, 1 # The result is overflow 16 EXIT: 17 </pre>				

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffffe0
gp	3	0x10000000
lp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x70000000
s2	10	0x00000000
s3	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x88888888
s3	19	0x78888888
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000

Kết quả: t0 = 0 như kỳ vọng.

2.s1 = 0x80000000 và s2 = 0x88888888 (cộng 2 số âm tràn)

lab03_1.asm	lab03_2.asm*	lab03_3.asm	lab03_4.asm	lab03_5.asm
<pre> 1 # Laboratory Exercise 4, Home Assignment 4 2 .text 3 # TODO: Thiết lập giá trị cho s1 và s2 4 li s1 0x80000000 5 li s2 0x88888888 6 # Thuật toán xác định trên số 7 li t0, 0 # Mặc định không có tràn số 8 add s3, s1, s2 # s3 = s1 + s2 9 xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu 10 blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu 11 xor t2 s1, s2 # Kiểm tra s1 với s2 có cùng dấu không 12 bne t2, zero, OVERFLOW # khác dấu thì tràn số 13 EXIT: 14 OVERFLOW: 15 li t0, 1 # The result is overflow 16 EXIT: 17 </pre>				

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffffe0
gp	3	0x10000000
lp	4	0x00000000
t0	5	0x00000001
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x80000000
s2	10	0x00000000
s3	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x88888888
s3	19	0x08888888
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000

Kết quả: t0 = 1 như kỳ vọng.

5. Assignment 5:

Chạy trường hợp 6 x 8 để tìm cách tổng quát:

.text

addi t1,zero,0x6

addi t2,zero,0x8

slli t1,t1,3 #khi dịch 3 bit sang trái, giá trị tăng lên 2^3 lần

KẾT QUẢ:

lab03_1.asm	lab03_2.asm*	lab03_3.asm	lab03_4.asm	lab03_5.asm
<pre> 1 .text 2 addi t1,zero,0x6 3 addi t2,zero,0x8 4 slli t1,t1,3 #khi dịch 3 bit sang trái, giá trị tăng lên 2^3 lần 5 </pre>				

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7fffffe0
gp	3	0x10000000
lp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000030
t2	7	0x00000008
s0	8	0x00000000

t0	5	0x00000000
t1	6	0x00000030
t2	7	0x00000008
s0	8	0x00000000

- $t1 = 0x30$ chính là 48 kết quả của 6×8
- Khi dịch 1 số n bit sang trái thì giá trị của nó sẽ tăng lên 2^n
- dịch 3 bit thì giá trị sẽ tăng 8 lần tương đương với việc nhân 8.
- ta nhận thấy lũy thừa của 2 thì sau khi dịch phải n bit thì sẽ bằng 1

Tổng quát:

.text

li t1 , 6

li t2, 16 # ví dụ 16 sẽ bằng 10000 là có 4bit 0

addi s0,s0,1 # gán giá trị $s0 = 1$ để dễ so sánh

loop:

beq t2,s0,break # so sánh giá trị t2 sau khi dịch bit

srli t2,t2,1 # dịch bit sang phải 1 bit lần lượt

slli t1,t1,1 # dịch bit sang trái 1 bit lần lượt

j loop

break:

Chú thích:

Chương trình RISC-V này đếm số bit 0 liên tiếp ở cuối của t2 và lưu kết quả theo dạng lũy thừa của 2 trong t1. Ban đầu, t1 được gán giá trị 6, còn $t2 = 16$ (0b10000), tức là có 4 bit 0 ở cuối. Thanh ghi s0 được gán giá trị 1 để làm điều kiện dừng vòng lặp. Trong vòng lặp, chương trình dịch phải t2 (srli t2, t2, 1), giúp loại bỏ từng bit 0 ở cuối cho đến khi $t2 == 1$, đồng thời dịch trái t1 (slli t1, t1, 1), tức là nhân đôi giá trị của nó mỗi lần lặp. Khi t2 bằng 1, vòng lặp dừng và giá trị cuối của t1 sẽ là $6 \times 2^{(\text{số bit 0 của t2})}$. Ví dụ với $t2 = 16$ (4 bit 0 cuối), t1 cuối cùng sẽ là $6 \times 2^4 = 96$.

KẾT QUẢ:

```

1  .text
2  li t1, 6
3  li t2, 16 # ví dụ 16 số bằng 10000 là số 4bit 0
4  andi s0, s0, 1 # gán giá trị s0 = 1 để để s0 nhân
5  loop:
6  beq t2, s0, break # so sánh giá trị t2 sau khi dịch bit
7  sll t2, t2, 1 # dịch bit sang phải 1 bit lần tiếp
8  sll t1, t1, 1 # dịch bit sang trái 1 bit lần tiếp
9  | loop
10 break:
11

```

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0xffffffff
tp	3	0x10000000
fp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000060
t2	7	0x00000001
s0	8	0x00000001
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
a8	18	0x00000000
a9	19	0x00000000
a10	20	0x00000000
a11	21	0x00000000
a12	22	0x00000000
a13	23	0x00000000
a14	24	0x00000000
a15	25	0x00000000
a16	26	0x00000000
a17	27	0x00000000

t0		5	0x00000000
t1		6	0x00000060
t2		7	0x00000001
s0	temporary (not preserved across call)	8	0x00000001
s1		9	0x00000000

- Nhận xét giá trị t1 = 0x60 đúng bằng 6*16
- ⇒ đúng như dự kiến

Câu hỏi kết luận: Ứng dụng phép dịch bit để thực hiện phép nhân có lợi gì so với sử dụng các lệnh nhân trong extension M (RV32M: RISC-V 32-bit Multiplier/Divider)

Lợi ích của việc sử dụng phép dịch bit để thực hiện phép nhân thay vì lệnh MUL (RV32M):

1. Tương thích với hệ thống không có extension M (Ví dụ: RV32I, RV32E).
2. Tiết kiệm tài nguyên phần cứng (Không cần mạch nhân phần cứng).
3. Tiêu thụ ít năng lượng hơn so với lệnh MUL.
4. Có thể nhanh hơn trên vi xử lý không có nhân phần cứng, tránh phải giả lập phần mềm.
5. Tận dụng tối ưu cho một số phép nhân đặc biệt (Nhân với lũy thừa của 2, nhân với số cố định).
6. Giảm độ phức tạp của vi xử lý, phù hợp với hệ thống nhúng hoặc IoT.
7. Linh hoạt hơn trong việc tối ưu hóa thuật toán, có thể thay thế phép nhân bằng các phép toán đơn giản hơn.

