BÁO CÁO THỰC HÀNH
IT3280 – 156786 – THỰC HÀNH KIẾN TRÚC MÁY TÍNH

NỘI DUNG
Bài 4. Các lệnh số học và logic

Họ và tên	Nguyễn Minh Quân
Mã số sinh viên	20235816

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.

Home Assigment 1

Lệnh số học đã được giới thiệu ở Bài thực hành 2, bài này sẽ đề cập đến trường hợp đặc biệt khi thực hiện lệnh cộng hai số 32-bit, đó là hiện tượng tràn số.

Chương trình ban đầu khi chưa có dữ liệu:

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

Trường hợp 1: Trường hợp không tràn số (s1 và s2 khác dấu)

```
1  .text
2  # TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
3  li s1, 1000
4  li s2, -50
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:
```

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.

- Dòng 1: li s1, 1000 # gán s1 = 1000

s1	9	1000

- Dòng 2: li s2, -50 # gán s2 = -50

s2	18	-50

- Dòng 3: li t0, 0 # Mặc định không có tràn số

t0	5	0

- Dòng 4: add s3, s1, s2 # s3 = s1 + s2

s3	19	950

- Dòng 5: xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu

t1	6	-986

➔ t1=-986 chứng tỏ rằng s1 và s2 khác dấu

- Dòng 6: blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu
  .t1 = -986 <0 → Kết thúc chương trình với kết quả không có tràn số xảy ra

Trường hợp 2: Trường hợp có tràn số (Cộng hai số dương lớn)

```
2  .text
3  # TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
4  li s1, 2147483640
5  li s2, 200
6   # Thuật toán xác định tràn 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 blt s1, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm
12 bge s3, s1, EXIT # s1 không âm, kiểm tra s3 nhỏ hơn s1 không
13 # Nếu s3 >= s1, không tràn số
14 j OVERFLOW
15 NEGATIVE:
16 bge s1, s3, EXIT # s1 âm, kiểm tra s3 có lớn hơn s1 không
17 # Nếu s1 >= s3, không tràn số
18 OVERFLOW:
19 li t0, 1 # The result is overflow
20 EXIT:
```

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.

- **Dòng 1: li s1, 2147483640 # gán s1 = 2147483640**

s1	9	2147483640

- **Dòng 2: li s2, 200 # gán s2 = 200**

s2	18	200

- **Dòng 3: li t0, 0 # Mặc định không có tràn số**

t0	5	0

- **Dòng 4: add s3, s1, s2 # s3 = s1 + s2**

s3	19	-2147483456

s3 = s1 + s2 = 2147483740, vượt quá 2147483647 (giá trị lớn nhất của số nguyên 32-bit có dấu).

- **Dòng 5: xor t1, s1, s2 # Kiểm tra s1 với s2 có cùng dấu**

t1	6	2147483440

T1 là một số lớn hơn 0

- **Dòng 6: blt t1, zero, EXIT # Nếu t1 là số âm, s1 và s2 khác dấu**

T1>0 → s1 và s2 cùng dấu với nhau → tiếp tục kiểm tra

- **Dòng 7: blt s1, zero, NEGATIVE # Kiểm tra s1 và s2 là số âm hay không âm**

  S1 = 2147483640 > 0 → tiếp tục kiểm tra

- **Dòng 8: bge s3, s1, EXIT # s1 không âm, kiểm tra s3 nhỏ hơn s1 không, nếu s3 >= s1, không tràn số**

S3 = -2147493456 < s1 = 2147483640 → jump tới OVERFLOW

- **Dòng 9: li t0, 1 # The result is overflow t0 =1**

t0	5	1

Trường hợp 3: Trường hợp có tràn số (Cộng hai số âm nhỏ)

```
.text
# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
li s1, -2147483640
li s2, -100
 # 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, 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:
```

s3 = s1 + s2 = -2147483740, nhỏ hơn -2147483648 (giá trị nhỏ nhất của số nguyên 32-bit có dấu).

→Bị tràn số, t0=1

Các dòng lệnh chạy thì như trường hợp 2

Trường hợp 4: Trường hợp không tràn số (Cộng hai số nhỏ cùng dấu)

```
.text
# TODO: Thiết lập giá trị cho s1 và s2 với trường hợp khác nhau
li s1, 50
li s2, 20
 # 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, 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:
```

s3 = s1 + s2 = 70, vẫn nằm trong phạm vi của số nguyên có dấu 32-bit.

→ Không bị tràn số, t0 vẫn là 0.

Các dòng lệnh chạy như trường hợp 1

Home Assignment 2

Chương trình dưới đây minh họa cách trích xuất nội dung thanh ghi sử dụng các lệnh logic. Như đã nói, ta có thể trích xuất 1 bit hoặc nhiều bit có trong thanh ghi. Hãy đọc kỹ ví dụ sau đây để hiểu rõ từng dòng lệnh:

Chương trình thực hiện:

```
1  # Laboratory Exercise 4, Home Assignment 2
2  .text
3  li s0, 0x12345678 # Test value
4  andi t0, s0, 0xff # Extract LSB of s0
5  andi t1, s0, 0x0400 # Extract bit 10 of s0
```

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.

- **Lệnh 1: li s0, 0x12345678 # Load giá trị 0x12345678 vào thanh ghi s0**

s0	8	0x12345678

- **Lệnh 2: andi t0, s0, 0xff # Lấy 8 bit cuối (LSB) của s0 (0x78)**

t0	5	0x00000078

Lệnh này thực hiện phép toán AND giữa giá trị trong thanh ghi s0 và giá trị tức thời 0xff, sau đó lưu kết quả vào thanh ghi t0.
S0: 0x12345678 (hex) = 00010010 00110100 01010110 01111000 (bin)
0xff: 00000000 00000000 00000000 11111111 (nhị phân)
➔ Kết quả : 00000000 00000000 00000000 01111000 (binary)= 0x78 (hệ 8).

- **Lệnh 3: andi t1, s0, 0x0400 # Extract bit 10 of s0**

t1	6	0x00000400

Lệnh này thực hiện phép toán AND giữa giá trị trong thanh ghi s0 và giá trị tức thời 0x0400, sau đó lưu kết quả vào thanh ghi t1
s0: 0x12345678 (hex) = 00010010 00110100 01010110 01111000 (bin)
0x0400: 00000000 00000100 00000000 00000000 (bin)
➔ Kết quả: 00000000 00000100 00000000 00000000 (binary= 0x0400 (hex).

Home Assigment 3

Loại lệnh tiếp theo được xét trong bài này là lệnh dịch bit (shift instruction). Lệnh này dịch toàn bộ 32-bit của thanh ghi sang trái hoặc sang phải. Các lệnh dịch bit bao gồm sll (shift left logical), srl (shift right logical), and sra (shift right arithmetic). Đối với lệnh shift right sẽ có hai phiên bản đó là logical (gán tất cả bit mới bằng 0) và arithmetic (gán tất cả các bit mới bằng bit dấu).

Chương trình thực hiện:

```
1  # Laboratory Exercise 3, Home Assignment 3
2  .text
3  li s0, 1 # s0 = 1
4  sll s1, s0, 2 # s1 = s0 * 4
```

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.

Lệnh 1: li s0, 1 # Gán giá trị 1 vào thanh ghi s0

s0	8	1

Lệnh 2: slli s1, s0, 2 # s1 = s0 * 4

s1	9	4

Lệnh này thực hiện phép dịch trái (shift left) giá trị trong thanh ghi s0 đi 2 bit, sau đó lưu kết quả vào thanh ghi s1

Giá trị s0: 1 (dec) = 00000001 (bin).

Dịch trái 2 bit: 00000001 dịch trái 2 bit thành 00000100 (bin) = 4 (dec).

Assignment 2

Viết một chương trình thực hiện các công việc sau:

- **Trích xuất MSB của thanh ghi s0**
- **Xóa LSB của thanh ghi s0**
- **Thiết lập LSB của thanh ghi s0 (bit 7 đến bit 0 được thiết lập bằng 1)**
- **Xóa thanh ghi s0 bằng cách dùng các lệnh logic (s0 = 0)**

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

LSB: Least Significant Byte (Byte có trọng số thấp)

Chương trình thực hiện:

```
1  # Laboratory Exercise 3, Home Assignment 3
2  .text
3  li s0,0x12345678 #s0 = 0x12345678
4  #Trich xuat MSB cua thanh ghi S0
5  srli t0, s0, 24
6  #Xóa LSB của thanh ghi s0
7  andi t1, s0, 0xffffff00
8  #Thiết lập LSB của thanh ghi s0 (bit 7 đến bit 0 được thiết lập bằng 1)
9  ori t2,s0,0xff
10 #Xóa thanh ghi s0 bằng cách dùng các lệnh logic (s0 = 0)
11 xor t3,s0,s0
```

- **Trích xuất MSB của thanh ghi s0:**

  Dùng lệnh srli (Shift Right Logical Immediate) để dịch phải giá trị của s0 đi 24 bit. Khi đó, byte cao nhất sẽ nằm ở 8 bit thấp nhất của kết quả.

  Kết quả:

t0	5	0x00000012
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x12345678

- **Xóa LSB của thanh ghi s0:**

Dùng lệnh andi để AND giá trị của s0 với 0xffffff00. Việc này sẽ giữ nguyên các bit từ bit 8 trở lên và xóa các bit từ bit 7 đến bit 0.

Kết quả

t1	6	0x12345600
t2	7	0x00000000
s0	8	0x12345678

- **Thiết lập LSB của thanh ghi s0:**

Dùng lệnh ori (OR Immediate) để OR giá trị của s0 với 0xff. Việc này sẽ đặt tất cả các bit từ bit 7 đến bit 0 thành 1.

t2	7	0x123456ff
s0	8	0x12345678

- **Xóa thanh ghi s0 bằng cách dùng các lệnh logic:**

Dùng lệnh xor (XOR) để XOR s0 với chính nó. Kết quả của phép XOR giữa hai giá trị giống nhau luôn là 0.

s0	8	0x12345678
t3	28	0x00000000

<u>Assignment 3</u>

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

b. mv s0, s1 s0 = s1

c. not s0 s0 = bit_invert(s0)

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

a) neg s0,s1 s0=-s1

Chương trình thực hiện

```
1   .text
2   #a)neg s0,s1
3   li s0, 20 #s0 = 20
4   sub s1,zero,s0 #s1 = 0 - s0 = -s0
```

Kết quả:

s0	8	20
s1	9	-20

b)mv s0,s1 #s0=s1

Chương trình thực hiện:

```
1   .text
2   #mv s0,s1
3   li s1, 30 #s1=30
4   add s0,zero,s1 #s0 = 0 + 30 = 30=s1
5
6
```

Kết quả:

s0	8	30
s1	9	30

c)not s0 s0 = bit_invert(s0)

Chương trình thực hiện:

```
.text
li s0, 0x12345 #s0=0x12345
xori s1,s0,0xFFFFFFFF # s0 = s0 XOR 0xFFFFFFFF
```

Trong biểu diện nhị phân, 0xFFFFFFFF là 1 số có tất cả 32 bit bằng 1, khi XOR với nó sẽ đảo toàn bộ bit

- 0 xor 1 = 1
- 1 xor 1 = 0

Kết quả:

s0	8	74565
s1	9	-74566

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

Chương trình thực hiện:

```
.text
li s1,2
li s2,3
# ble s1, s2, label (if (s1 <= s2) j lab )
bge s2, s1, CONDITION  # if (s2 >= s1) j lab
CONDITION:
addi s2,s2,1
```

s1 <= s2 tương đương với s2>=s1

Kết quả:

s2	18	4

<u>Assignment 4</u>

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.

Chương trình thực hiện

```
1   .text
2   li s0,2147483000
3   li s1,1000
4   add s2,s0,s1 #s2 = s0 + s1
5   xor t0, s0, s1  # t0 = s0 XOR s1 (kiểm tra cùng dấu)
6   bltz t0, EXIT # Nếu khác dấu, không tràn số
7   xor t1,s2,s0 #t1= s2 XOR s0
8   blt t1, zero, OVERFLOW #if t1<0 -> tràn số
9   OVERFLOW:
10  li a0, 1
11  EXIT:
```

Giải thuật

- **Bước 1: Kiểm tra xem hai toán hạng nguồn có cùng dấu không.**
  Nếu s1 XOR s2 có bit dấu (31) bằng 0, thì chúng cùng dấu.
- **Bước 2: Nếu hai toán hạng cùng dấu, kiểm tra xem kết quả có cùng dấu với toàn hạng nguồn hay không**
  Nếu s1 XOR s3 có bit dấu (31) bằng 1, thì xảy ra tràn số.

Kết quả:

Sau khi thực hiện câu lệnh add s2,s0,s1 ta thu được kết quả

Sau khi kiểm tra dấu giữa hai toán hạng nguồn ta thu được kết quả

➔ Hai toán hạng cùng dấu, tiếp tục kiểm tra

Sau khi kiểm tra dấu giữa tổng và toán hạng nguồn ta thu được kết quả

t1	6	-1000

Sau khi kiểm tra câu lệnh blt t1,zero, OVERFLOW ta thu được kết quả

a0	10	1

A0=1 chứng tỏ tràn số đã xảy ra, cũng dễ hiểu sau khi kiểm tra dấu giữa tổng và toàn hạng nguồn ta thấy t1=-1000 < 0 nên sẽ jump tới OVERFLOW thực hiện câu lệnh bên trong

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.

Chương trình thực hiện

```
1  .text
2  li t1,6 #t1 = 6
3  li t2,8 #t2 = 8 ( 2^3 )
4  slli s0,t1,3 #Dịch trái 3 bit
5
```

Kết quả:

t1	6	6
t2	7	8
s0	8	48

- Giá trị trong thanh ghi t1 là 6, biểu diễn nhị phân là 0000 0110.

- Dịch trái 3 bit sẽ biến 0000 0110 thành 0011 0000, tương đương với 48 trong hệ thập phân. Hay việc dịch trái 3 bit tương đương với việc nhân giá trị ban đầu với 2^3 nên ta có kết quả 6 x 2^3 = 48

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)

1. **Tiết kiệm tài nguyên phần cứng:**
    - Phép dịch bit là một phép toán đơn giản và không yêu cầu phần cứng phức tạp như bộ nhân (multiplier). Điều này giúp tiết kiệm diện tích chip và năng lượng tiêu thụ, đặc biệt quan trọng trong các hệ thống nhúng hoặc các thiết bị có nguồn tài nguyên hạn chế.
2. **Tốc độ thực hiện nhanh hơn trong một số trường hợp:**
    - Đối với các phép nhân đơn giản, chẳng hạn như nhân với lũy thừa của 2, phép dịch bit có thể được thực hiện nhanh hơn so với việc sử dụng lệnh nhân. Ví dụ, nhân một số với 4 có thể được thực hiện bằng cách dịch trái 2 bit, thay vì sử dụng lệnh nhân.
3. **Giảm độ phức tạp của mã lệnh:**
    - Sử dụng phép dịch bit có thể làm cho mã lệnh đơn giản và dễ hiểu hơn, đặc biệt là khi thực hiện các phép toán liên quan đến lũy thừa của 2. Điều này cũng giúp giảm thiểu lỗi trong quá trình lập trình.
4. **Tối ưu hóa cho các phép toán cụ thể:**
    - Đối với các phép toán cụ thể như nhân với lũy thừa của 2, phép dịch bit là phương pháp tối ưu nhất. Nó không chỉ nhanh hơn mà còn không gây ra lỗi do làm tròn hoặc mất mát dữ liệu.