# 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:**

**A screenshot of a computer

AI-generated content may be incorrect.**

|  |
| --- |
| # 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

A screenshot of a spreadsheet

AI-generated content may be incorrect.

**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**:

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| # 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 >= 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 >= s3, không tràn số  OVERFLOW:  li t0, 1 # The result is overflow  EXIT: |

**Kết quả chạy**

**A screenshot of a spreadsheet

AI-generated content may be incorrect.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**

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| # 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

A screenshot of a spreadsheet

AI-generated content may be incorrect.

**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**

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| # 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

A screenshot of a spreadsheet

AI-generated content may be incorrect.

**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 >= s1 thì kết quả không bị tràn số

**Trường hợp 5**: Tràn hai số nguyên âm

**Nhập chương trình**

**A screenshot of a computer

AI-generated content may be incorrect.**

|  |
| --- |
| # 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

A screenshot of a spreadsheet

AI-generated content may be incorrect.

**Quan sát bộ nhớ và thanh ghi**

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

## 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 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)

A diagram of a number

AI-generated content may be incorrect.

**Nhập chương trình**

**A green text on a white background

AI-generated content may be incorrect.**

|  |
| --- |
| .text  li s0, 0x12345678 # s0 = 0x12345678  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 (s0 = 0) |

**Kết quả chạy chương trình**

**A screenshot of a spreadsheet

AI-generated content may be incorrect.**

**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, 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

## 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 đấy sử dụng các lệnh chính thống mà RISC-V định nghĩa:

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

**Nhập chương trình**

**A white background with red text

AI-generated content may be incorrect.**

|  |
| --- |
| .text  li s1, 10 # s1 = 10  sub s0, zero, s1 # s0 = 0 - s1 |

**Kết quả chạy**

A table with numbers and lines

AI-generated content may be incorrect.

**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.

1. mv s0, s1 (s0 = s1)

**Nhập chương trình**

A close up of numbers

AI-generated content may be incorrect.

|  |
| --- |
| .text  li s1, 10 # s1 = 10  addi s0, s1, 0 # s0 = s1 + 0 |

Kết quả chạy

A table with numbers and letters

AI-generated content may be incorrect.

**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.

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

**Nhập chương trình** ****

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

Kết quả chạy

A table with numbers and a green line

AI-generated content may be incorrect.

**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).

1. ble s1, s2, label

if (s1 < s2)

j label

**Nhập chương trình**

A screenshot of a computer

AI-generated content may be incorrect.

|  |
| --- |
| .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, junp else  then:  add t3, t1, t2 # z = x + y  j endif  else:  sub t3, t2, t1 # z = y - x  endif: |

Kết quả chạy

A screenshot of a spreadsheet

AI-generated content may be incorrect.

s2 s1 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 s1, tương đương s1 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**

A screenshot of a computer program

AI-generated content may be incorrect.

|  |
| --- |
| .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)  ecall # Thoát chương trình |

**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, trần 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 bảo “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 |
|  |  |  |
|  |  |  |

## 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**

**A screenshot of a computer code

AI-generated content may be incorrect.**

|  |
| --- |
| .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**

**A table with numbers and a green line

AI-generated content may be incorrect.**

**Giải thích kết quả chạy**

* t0 chứa số là lũy thừa của 2.
* Dùng vòng lặp để đếm số bit 1 trong t0, tức là tính log2(t0).
* Dịch trái t1 với số bit tương ứng (t2).
* Ví dụ: Nếu t1 = 6 và t0 = 8 (2³), chương trình sẽ tính log2(8) = 3, rồi thực hiện 6 << 3 = 4810.
* 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:

**+ Tiết kiệm tài nguyên phần cứng**: Không phải tất cả các vi xử lý RISC-V đều hỗ trợ extension M

+ **Hiệu suất cao trong một số trường hợp**: Ví dụ như nhân với lũy thừa của 2, phép dịch bit có thể thực hiện nhanh hơn lệnh *mul*

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.