Tuyệt vời! Dựa trên đoạn mã Assembly bạn cung cấp và các yêu cầu của Bài tập 9, đây là báo cáo chi tiết:

**Báo cáo Bài tập 9: Kiểm thử các thuật toán sắp xếp**

**1.1. Mô tả bài toán**

Bài toán yêu cầu xây dựng một chương trình Assembly (RISC-V) để kiểm thử hiệu năng của các thuật toán sắp xếp khác nhau. Chương trình cần thực hiện các chức năng sau:

1. **Đọc dữ liệu từ file:** Người dùng nhập tên một file văn bản. File này chứa các số nguyên (có thể âm hoặc dương) được phân cách bởi dấu cách hoặc ký tự xuống dòng. Số lượng phần tử có thể lên đến 10,000.
2. **Lưu trữ dữ liệu:** Các số nguyên đọc từ file được lưu vào một mảng trong bộ nhớ.
3. **Giao diện lựa chọn thuật toán:** Hiển thị menu cho phép người dùng chọn một trong các thuật toán sắp xếp: Nổi bọt (Bubble Sort), Chèn (Insertion Sort), Lựa chọn (Selection Sort), và Sắp xếp nhanh (Quick Sort).
4. **Thực thi sắp xếp và đo thời gian:**
   * Chạy thuật toán sắp xếp đã chọn trên mảng dữ liệu.
   * Đo và hiển thị thời gian thực hiện của thuật toán (tính bằng mili giây).
5. **Ghi kết quả:** Lưu mảng số nguyên đã được sắp xếp vào một file văn bản kết quả có tên cố định là "C:\RISCV\output5.txt".
6. **Xử lý số âm:** Chương trình có khả năng đọc và lưu trữ số âm. Các thuật toán sắp xếp cần xử lý đúng thứ tự của các số âm và dương.

**1.2. Thuật toán sử dụng**

Chương trình sử dụng một loạt các thuật toán và kỹ thuật để hoàn thành yêu cầu:

1. **Luồng điều khiển chính (**main**):**
   * Nhắc người dùng nhập tên file đầu vào.
   * Xử lý tên file (loại bỏ ký tự xuống dòng).
   * Mở file đầu vào. Nếu lỗi, hiển thị thông báo và thoát.
   * Gọi hàm read\_numbers để đọc và phân tích cú pháp các số từ file vào mảng numbers.
   * Hiển thị menu lựa chọn thuật toán sắp xếp.
   * Dựa trên lựa chọn của người dùng, gọi hàm sắp xếp tương ứng.
   * Trước và sau mỗi thuật toán sắp xếp, gọi hàm get\_time để lấy thời gian hệ thống.
   * Gọi hàm print\_time để tính và hiển thị thời gian thực thi.
   * Gọi hàm write\_results để ghi mảng đã sắp xếp ra file output.
   * Lặp lại việc hiển thị menu cho đến khi người dùng chọn thoát.
2. **Đọc và Phân tích cú pháp số (**read\_numbers**):**
   * Sử dụng syscall 63 (read) để đọc từng ký tự từ file đầu vào.
   * Xây dựng số nguyên từ các ký tự số ('0'-'9').
   * Xử lý dấu trừ ('-') ở đầu để xác định số âm.
   * Khi gặp ký tự phân cách (dấu cách hoặc xuống dòng), số vừa xây dựng được lưu vào mảng numbers.
   * Đếm tổng số lượng số nguyên đọc được (count).
   * Sử dụng syscall 57 (close) để đóng file sau khi đọc xong.
3. **Thuật toán Sắp xếp:**
   * **Bubble Sort (**bubble\_sort\_core**):** So sánh các cặp phần tử liền kề và hoán vị nếu chúng sai thứ tự. Lặp lại cho đến khi mảng được sắp xếp.
   * **Insertion Sort (**insertion\_sort\_array\_impl**):** Xây dựng mảng đã sắp xếp bằng cách chèn lần lượt từng phần tử từ phần chưa sắp xếp vào đúng vị trí trong phần đã sắp xếp.
   * **Selection Sort (**selection\_sort\_array\_impl**):** Lặp lại việc tìm phần tử nhỏ nhất trong phần chưa sắp xếp và hoán vị nó với phần tử đầu tiên của phần đó.
   * **Quick Sort (**quick\_sort\_logic**,** partition\_elements**):**
     + Sử dụng thuật toán chia để trị.
     + Hàm partition\_elements chọn một phần tử làm pivot (ở đây là phần tử cuối cùng của đoạn đang xét) và sắp xếp lại mảng sao cho tất cả các phần tử nhỏ hơn hoặc bằng pivot đứng trước pivot, và tất cả các phần tử lớn hơn đứng sau. Hàm trả về chỉ số của pivot sau khi phân hoạch.
     + Hàm quick\_sort\_logic gọi đệ quy chính nó cho hai nửa mảng (trái và phải của pivot).
4. **Đo thời gian (**get\_time**,** print\_time**):**
   * get\_time: Sử dụng syscall 30 để lấy thời gian hiện tại của hệ thống (thường là số mili giây từ một thời điểm gốc).
   * print\_time: Tính hiệu số giữa thời gian kết thúc và thời gian bắt đầu để có được thời gian thực thi, sau đó in ra console.
5. **Chuyển đổi Số sang Chuỗi (**number\_to\_string**,** str\_reverse**):**
   * number\_to\_string: Chuyển đổi một số nguyên (có thể âm) thành một chuỗi ký tự ASCII.
     + Xử lý trường hợp số 0.
     + Xử lý dấu trừ cho số âm.
     + Chuyển đổi từng chữ số bằng cách lấy phần dư cho 10 và cộng với mã ASCII của '0'. Các chữ số được lưu ngược vào buffer.
   * str\_reverse: Đảo ngược chuỗi ký tự trong buffer (vì number\_to\_string tạo ra chuỗi số bị ngược).
6. **Ghi File Kết quả (**write\_results**):**
   * Sử dụng syscall 1024 (open) để mở/tạo file output (C:\\RISCV\\output5.txt) với cờ ghi.
   * Lặp qua mảng numbers đã sắp xếp.
   * Với mỗi số, gọi number\_to\_string để chuyển nó thành chuỗi.
   * Sử dụng syscall 64 (write) để ghi chuỗi số vào file.
   * Ghi một dấu cách sau mỗi số (trừ số cuối cùng).
   * Ghi một ký tự xuống dòng ở cuối file.
   * Sử dụng syscall 57 (close) để đóng file output.
7. **Xử lý số âm (Bitmask -** flag\_negative\_numbers**,** neg\_bitmask**):**
   * Đoạn mã có khai báo neg\_bitmask và hàm flag\_negative\_numbers. Hàm này được gọi *sau* mỗi thuật toán sắp xếp. Nó duyệt qua mảng numbers và nếu một số là âm, nó sẽ đặt một bit tương ứng trong neg\_bitmask.
   * **Lưu ý quan trọng:** Mảng neg\_bitmask không được sử dụng ở bất kỳ đâu khác trong chương trình (ví dụ, không dùng khi ghi file). Các thuật toán sắp xếp trong mã (Bubble, Insertion, Selection, Quick Sort) khi thực hiện so sánh trực tiếp các giá trị số nguyên (đã được đọc và lưu trữ dưới dạng số có dấu bởi read\_numbers) sẽ tự động sắp xếp chúng theo đúng thứ tự số học (ví dụ: -5 < -2 < 0 < 1 < 3). Do đó, chức năng của flag\_negative\_numbers và neg\_bitmask trong ngữ cảnh hiện tại là không rõ ràng hoặc có thể là một phần còn sót lại của một chiến lược xử lý số âm khác (ví dụ: sắp xếp theo giá trị tuyệt đối rồi khôi phục dấu). Hiện tại, chúng không ảnh hưởng đến kết quả sắp xếp cuối cùng được ghi ra file.

**1.3. Đoạn mã lệnh và giải thích**

Dưới đây là phân tích các đoạn mã chính trong chương trình của bạn:

**1. Khai báo dữ liệu (**.data**)**

Đoạn mã

.data

numbers: .space 80000 # Vùng nhớ 80KB cho tối đa 20,000 số nguyên (4 byte/số)

input\_buffer\_size: .word 80000 # Kích thước tối đa của dữ liệu đọc (không trực tiếp dùng làm buffer đọc)

count: .word 0 # Biến đếm số lượng số nguyên đã đọc

neg\_bitmask: .space 2500 # 2500 bytes = 20000 bits, cho bitmask số âm

input\_filename: .space 256 # Buffer cho tên file input

file\_read\_buffer: .space 1024 # Buffer tạm, trong read\_numbers dùng để đọc 1 byte mỗi lần

msg\_prompt\_input: .asciz "Enter filename: "

error\_msg: .asciz "\nError opening file\n"

menu: .asciz "\nUser select sorting algorithm:\n1. Bubble Sort\n2. Insertion Sort\n3. Selection Sort\n4. Quick Sort\n5. Close\nChoice: "

fd: .word 0 # File descriptor cho file input

newline: .asciz "\n"

space: .string " " # ".string" sẽ bao gồm cả null terminator, ".asciz" thì không.

# Nếu chỉ cần 1 dấu cách, .asciz " " là đủ.

start\_time: .word 0

end\_time: .word 0

msg\_execution\_time: .asciz "\nExecution time (ms): "

output\_filename: .asciz "C:\\RISCV\\output5.txt" # Tên file output cố định

out\_fd: .word 0 # File descriptor cho file output

buffer\_number: .space 12 # Buffer tạm để chuyển số sang chuỗi (đủ cho số 32-bit và dấu)

msg\_file\_error\_open: .asciz "\nError writing to output file\n"

char\_minus: .asciz "-" # Chuỗi dấu trừ (không cần thiết nếu number\_to\_string tự xử lý)

* **Giải thích:** Khai báo các vùng nhớ, chuỗi thông báo, biến toàn cục. numbers là mảng chính lưu dữ liệu. file\_read\_buffer được dùng trong read\_numbers để đọc từng byte một.

**2. Hàm** main **và Vòng lặp Menu**

Đoạn mã

.text

.globl main

main:

# In ra msg\_prompt\_input

li a7, 4

la a0, msg\_prompt\_input

ecall

# Doc input\_filename

li a7, 8

la a0, input\_filename

li a1, 256

ecall

# Loai bo newline khoi input\_filename (quan trọng để tên file đúng)

la t0, input\_filename

remove\_newline\_from\_filename:

lb t1, 0(t0)

beqz t1, open\_input\_file # Kết thúc chuỗi

li t2, 10 # 10 = "\n"

beq t1, t2, replace\_null

addi t0, t0, 1

j remove\_newline\_from\_filename

replace\_null:

sb zero, 0(t0) # Thay '\n' bằng '\0'

open\_input\_file:

# Mo file input (syscall 1024 là syscall không chuẩn, thường là cho openat)

li a7, 1024 # Giả sử đây là syscall "open file"

la a0, input\_filename

li a1, 0 # Cờ 0: Read-only

ecall

bltz a0, file\_error\_open # Kiểm tra lỗi (nếu fd < 0)

la t1, fd

sw a0, 0(t1) # Lưu file descriptor

jal read\_numbers # Gọi hàm đọc số

menu\_loop:

# Hien thi menu

li a7, 4

la a0, menu

ecall

# Doc lua chon cua nguoi dung

li a7, 5 # Syscall for read\_integer

ecall

# Xu ly lua chon (so sánh a0 với 1, 2, 3, 4, 5)

li t0, 1

beq a0, t0, bubble\_sort\_array

li t0, 2

beq a0, t0, insertion\_sort\_array

li t0, 3

beq a0, t0, selection\_sort\_array

li t0, 4

beq a0, t0, quick\_sort\_array

li t0,5

beq a0, t0, exit

j exit # Lựa chọn không hợp lệ cũng thoát

file\_error\_open:

li a7, 4

la a0, error\_msg

ecall

j exit

exit:

li a7, 10 # Syscall for exit program

ecall

* **Giải thích:**
  + main bắt đầu bằng việc yêu cầu người dùng nhập tên file.
  + remove\_newline\_from\_filename: Loại bỏ ký tự \n thường có ở cuối chuỗi nhập từ read\_string.
  + open\_input\_file: Mở file bằng syscall 1024. a0 là tên file, a1 là cờ (0 = read-only). File descriptor trả về được lưu vào biến fd.
  + read\_numbers được gọi để đọc dữ liệu.
  + menu\_loop: Hiển thị menu, đọc lựa chọn của người dùng (syscall 5), và nhảy đến hàm sắp xếp tương ứng hoặc thoát.

**3.** **Hàm** read\_numbers **(Đọc và Phân tích cú pháp số từ File)**

Đoạn mã

read\_numbers:

# ... (lưu thanh ghi vào stack) ...

la t1, count

sw zero, 0(t1) # Reset count = 0

li t0, 0 # current\_number = 0

li t1, 0 # in\_number\_flag = false

li t6, 0 # is\_negative\_flag = false

read\_loop:

# Doc mot ky tu tu file (syscall 63 - Read)

li a7, 63

lw a0, fd # File descriptor

la a1, file\_read\_buffer # Buffer để đọc (chỉ 1 byte)

li a2, 1 # Số byte cần đọc = 1

ecall

beqz a0, read\_done # Nếu số byte đọc được là 0 -> EOF (End Of File)

lb t2, 0(a1) # t2 = ký tự đã đọc

# Kiem tra dau tru '-'

li t3, 45 # ASCII for '-'

bne t2, t3, not\_char\_minus

beqz t1, set\_negative # Nếu chưa ở trong một số, thì đây là dấu âm

j read\_loop # Nếu đã ở trong số mà gặp '-' -> lỗi format, bỏ qua ký tự này

set\_negative:

li t6, 1 # is\_negative\_flag = true

li t1, 1 # in\_number\_flag = true (bắt đầu một số)

j read\_loop

not\_char\_minus:

# Kiem tra ky tu phan cach (space hoac newline)

li t3, 32 # ASCII for space

beq t2, t3, save\_number

li t3, 10 # ASCII for newline

beq t2, t3, save\_number

# Chuyen doi ky tu ASCII so sang gia tri so

addi t2, t2, -48 # Chuyển '0'-'9' thành 0-9

# Kiểm tra nếu t2 không phải là số (0-9) thì nên bỏ qua hoặc xử lý lỗi

# Ví dụ: blt t2, zero, read\_loop; bgt t2, 9, read\_loop (sau khi trừ 48)

li t3, 10

mul t0, t0, t3 # current\_number = current\_number \* 10

add t0, t0, t2 # current\_number = current\_number + digit

li t1, 1 # in\_number\_flag = true

j read\_loop

save\_number:

beqz t1, read\_loop # Nếu không ở trong một số (ví dụ: nhiều dấu cách), bỏ qua

beqz t6, save\_positive

neg t0, t0 # Áp dụng dấu âm: current\_number = -current\_number

save\_positive:

# Luu so vao mang numbers

la t3, count

lw t3, 0(t3) # Lấy giá trị count hiện tại

slli t4, t3, 2 # offset = count \* 4 (vì mỗi số nguyên 4 byte)

la t5, numbers

add t5, t5, t4 # Địa chỉ numbers[count]

sw t0, 0(t5) # Lưu số

# Tang bien dem count

addi t3, t3, 1

la t4, count

sw t3, 0(t4) # Cập nhật count

# Reset bien tam cho so tiep theo

li t0, 0

li t1, 0

li t6, 0

j read\_loop

read\_done:

# Luu so cuoi cung neu file ket thuc khi dang parse (ví dụ: file không có space/newline cuối)

beqz t1, close\_file

beqz t6, save\_last\_positive

neg t0, t0

save\_last\_positive:

# ... (lưu số cuối tương tự như save\_number) ...

close\_file:

# Dong file input (syscall 57 - Close)

li a7, 57

lw a0, fd

ecall

# ... (khôi phục thanh ghi từ stack và ret) ...

* **Giải thích:**
  + Hàm đọc file từng ký tự một.
  + Biến t0 tích lũy giá trị số hiện tại. t1 (in\_number\_flag) cho biết có đang phân tích một số hay không. t6 (is\_negative\_flag) đánh dấu nếu số là âm.
  + Khi gặp dấu cách hoặc xuống dòng (hoặc EOF), số hiện tại (nếu có) được lưu vào mảng numbers, bao gồm cả việc áp dụng dấu âm nếu t6 được đặt.
  + Số lượng phần tử được cập nhật trong count.

**4.** **Các hàm Sắp xếp (Ví dụ:** quick\_sort\_array**,** quick\_sort\_logic**,** partition\_elements**)**

* quick\_sort\_array **(Hàm bao ngoài cho Quick Sort):**

Đoạn mã

quick\_sort\_array:

jal get\_time # Lấy thời gian bắt đầu

sw a0, start\_time, t0 # Lưu thời gian bắt đầu (t0 là thanh ghi tạm không cần thiết ở đây)

la a0, numbers # Tham số 1: địa chỉ mảng

li a1, 0 # Tham số 2: left\_index = 0

lw a2, count # Lấy số lượng phần tử

addi a2, a2, -1 # Tham số 3: right\_index = count - 1

jal quick\_sort\_logic # Gọi hàm Quick Sort chính

# Gọi flag\_negative\_numbers (hiện tại không có tác dụng rõ ràng)

la a0, numbers

lw a1, count

jal flag\_negative\_numbers

jal get\_time # Lấy thời gian kết thúc

sw a0, end\_time, t0 # Lưu thời gian kết thúc

jal print\_time # In thời gian thực thi

jal write\_results # Ghi kết quả ra file

j menu\_loop

* + **Giải thích:** Thiết lập tham số và gọi quick\_sort\_logic. Đo thời gian, in thời gian, ghi kết quả.
* quick\_sort\_logic **(Hàm đệ quy Quick Sort):**

Đoạn mã

quick\_sort\_logic:

# ... (lưu thanh ghi s0-s4, ra vào stack) ...

mv s0, a0 # s0 = array\_base

mv s1, a1 # s1 = left\_index

mv s2, a2 # s2 = right\_index

bge s1, s2, quick\_sort\_end # Điều kiện dừng đệ quy: if left >= right

# Gọi hàm phan hoach (partition)

mv a0, s0 # array

mv a1, s1 # left

mv a2, s2 # right

jal partition\_elements

mv s3, a0 # s3 = pivot\_index (kết quả từ partition)

# De quy Quick Sort cho phan ben trai pivot

mv a0, s0

mv a1, s1

addi a2, s3, -1 # right = pivot\_index - 1

jal quick\_sort\_logic

# De quy Quick Sort cho phan ben phai pivot

mv a0, s0

addi a1, s3, 1 # left = pivot\_index + 1

mv a2, s2 # right (giữ nguyên từ ban đầu của đoạn này)

jal quick\_sort\_logic

quick\_sort\_end:

# ... (khôi phục thanh ghi và ret) ...

* + **Giải thích:** Thực hiện logic đệ quy của Quick Sort. Gọi partition\_elements để tìm vị trí pivot, sau đó gọi đệ quy cho hai phần con.
* partition\_elements **(Hàm phân hoạch của Quick Sort):**

Đoạn mã

partition\_elements:

# ... (lưu thanh ghi s0-s4, ra vào stack) ...

mv s0, a0 # s0 = array\_base

mv s1, a1 # s1 = left

mv s2, a2 # s2 = right

# Chon pivot la phan tu cuoi cung arr[right]

slli t0, s2, 2

add t0, s0, t0

lw s3, 0(t0) # s3 = pivot\_value = arr[right]

addi s4, s1, -1 # s4 = i (index of smaller element) = left - 1

mv t1, s1 # t1 = j (current\_index) = left

partition\_loop\_elements:

bge t1, s2, partition\_elements\_done # Loop j from left to right-1

slli t0, t1, 2

add t0, s0, t0

lw t2, 0(t0) # t2 = arr[j]

bgt t2, s3, skip\_swap # if arr[j] > pivot\_value, skip swap

addi s4, s4, 1 # Increment i

# Hoan doi arr[i] va arr[j]

slli t0, s4, 2 # address of arr[i]

add t0, s0, t0

slli t3, t1, 2 # address of arr[j]

add t3, s0, t3

lw t4, 0(t0) # temp = arr[i]

lw t5, 0(t3) # arr[j] (giá trị, không phải địa chỉ)

sw t5, 0(t0) # arr[i] = arr[j]

sw t4, 0(t3) # arr[j] = temp

skip\_swap:

addi t1, t1, 1 # Increment j

j partition\_loop\_elements

partition\_elements\_done:

# Hoan doi arr[i+1] voi arr[right] (pivot)

addi s4, s4, 1 # pivot\_final\_index = i + 1

# ... (logic hoán đổi arr[pivot\_final\_index] và arr[right]) ...

slli t0, s4, 2 # dia chi arr[i+1]

add t0, s0, t0

slli t1, s2, 2 # dia chi arr[right]

add t1, s0, t1

lw t2, 0(t0) # temp = arr[i+1]

# lw t3, 0(t1) # arr[right] (pivot\_value, đã có trong s3)

sw s3, 0(t0) # arr[i+1] = pivot\_value

sw t2, 0(t1) # arr[right] = temp (giá trị cũ của arr[i+1])

mv a0, s4 # Tra ve chi so cua pivot (i+1)

# ... (khôi phục thanh ghi và ret) ...

* + **Giải thích:** Chọn phần tử arr[right] làm pivot. Biến i theo dõi ranh giới của các phần tử nhỏ hơn pivot. Duyệt j từ left đến right-1. Nếu arr[j] nhỏ hơn hoặc bằng pivot, tăng i và hoán vị arr[i] với arr[j]. Cuối cùng, hoán vị pivot (arr[right]) với arr[i+1]. Trả về i+1 là vị trí mới của pivot. Các so sánh lw và bgt hoạt động chính xác với các số nguyên có dấu.

**5. Hàm** flag\_negative\_numbers

Đoạn mã

flag\_negative\_numbers:

# ... (lưu thanh ghi) ...

mv s0, a0 # s0 = array\_base (numbers)

mv s1, a1 # s1 = size (count)

li s2, 0 # s2 = index

flag\_loop:

bge s2, s1, flag\_done # Loop for index from 0 to size-1

slli t0, s2, 2

add t0, s0, t0

lw t1, 0(t0) # t1 = numbers[index]

bgez t1, skip\_flag # If number is positive or zero, skip

# Tinh toan offset byte va vi tri bit trong neg\_bitmask

mv t0, s2 # t0 = index

srai t1, t0, 3 # byte\_offset = index / 8

andi t2, t0, 0x7 # bit\_position = index % 8 (0-7)

li t3, 1

sll t3, t3, t2 # create bit mask: 1 << bit\_position

# Dat bit trong neg\_bitmask

la t4, neg\_bitmask

add t4, t4, t1 # address of byte in bitmask

lb t5, 0(t4) # load current byte from bitmask

or t5, t5, t3 # set the specific bit

sb t5, 0(t4) # store modified byte back

skip\_flag:

addi s2, s2, 1

j flag\_loop

flag\_done:

# ... (khôi phục thanh ghi và ret) ...

* **Giải thích:** Hàm này duyệt qua mảng numbers. Nếu một số là âm, nó tính toán vị trí bit tương ứng trong neg\_bitmask và đặt bit đó thành 1. Như đã đề cập, neg\_bitmask không được sử dụng ở đâu khác nên chức năng này hiện không có tác động đến đầu ra.

**6. Hàm** get\_time **và** print\_time

Đoạn mã

get\_time:

li a7, 30 # Syscall 30 (thường là get\_time\_ms trong MARS/RARS)

ecall # Thời gian trả về trong a0

ret

print\_time:

la t0, start\_time

lw t1, 0(t0) # t1 = start\_time

la t0, end\_time

lw t2, 0(t0) # t2 = end\_time

sub t3, t2, t1 # t3 = execution\_time = end - start

li a7, 4

la a0, msg\_execution\_time

ecall

li a7, 1 # Syscall for print\_integer

mv a0, t3

ecall

ret

* **Giải thích:** get\_time lấy thời gian hệ thống. print\_time tính toán chênh lệch và in ra.

**7.** **Hàm** number\_to\_string **và** str\_reverse

Đoạn mã

number\_to\_string:

# ... (lưu thanh ghi) ...

mv s0, a0 # s0 = buffer\_address

mv s1, a1 # s1 = number\_to\_convert

li s2, 0 # s2 = string\_length

li s3, 0 # s3 = negative\_flag

# Xu ly so 0

bnez s1, check\_sign

li t0, 48 # '0'

sb t0, 0(s0)

li a0, 1 # length = 1

j num\_to\_str\_done

check\_sign:

bgez s1, convert\_digits

li s3, 1 # set negative\_flag

neg s1, s1 # make number positive for conversion

convert\_digits:

mv s4, s0 # s4 = temp\_pointer\_in\_buffer (để lưu chữ số ngược)

digit\_loop:

beqz s1, finalize\_string # Nếu số = 0, đã xong phần chữ số

li t1, 10

rem t2, s1, t1 # t2 = last\_digit

div s1, s1, t1 # number = number / 10

addi t2, t2, 48 # convert digit to ASCII

sb t2, 0(s4)

addi s4, s4, 1

addi s2, s2, 1 # string\_length++

j digit\_loop

finalize\_string:

beqz s3, reverse\_string # Nếu không âm, nhảy tới đảo chuỗi

li t1, 45 # '-'

sb t1, 0(s4) # Thêm dấu '-' vào cuối chuỗi (trước khi đảo)

addi s4, s4, 1

addi s2, s2, 1

reverse\_string:

mv a0, s0 # start\_address của buffer

addi a1, s4, -1 # end\_address (s4 đang ở vị trí sau ký tự cuối)

jal str\_reverse

mv a0, s2 # Trả về độ dài chuỗi trong a0

num\_to\_str\_done:

# ... (khôi phục thanh ghi và ret) ...

str\_reverse: # a0 = start\_ptr, a1 = end\_ptr

bge a0, a1, str\_rev\_done

lb t0, 0(a0)

lb t1, 0(a1)

sb t1, 0(a0)

sb t0, 0(a1)

addi a0, a0, 1

addi a1, a1, -1

j str\_reverse

str\_rev\_done:

ret

* **Giải thích:** number\_to\_string chuyển đổi số nguyên thành chuỗi ASCII, xử lý số âm và số 0. Các chữ số được tạo ra theo thứ tự ngược, sau đó str\_reverse được gọi để đảo ngược chuỗi về đúng thứ tự.

**8.** **Hàm** write\_results **(Ghi kết quả đã sắp xếp ra File)**

Đoạn mã

write\_results:

# ... (lưu thanh ghi) ...

# Mo file output (syscall 1024)

li a7, 1024

la a0, output\_filename

li a1, 1 # Cờ 1: Write-only (nếu file tồn tại, sẽ bị ghi đè, nếu không sẽ tạo mới)

# Cần thêm cờ O\_CREAT, O\_TRUNC nếu syscall hỗ trợ

li a2, 0x1ff # Permissions rwxrwxrwx (octal 0777)

ecall

bltz a0, msg\_file\_error\_openor # Kiểm tra lỗi

sw a0, out\_fd, t0 # Lưu output file descriptor

li s0, 0 # i = 0 (loop counter)

lw s1, count # s1 = total numbers

la s2, numbers # s2 = numbers\_array\_base

write\_loop:

bge s0, s1, write\_done # if i >= count, done

# Tai so hien tai numbers[i]

slli t0, s0, 2

add t1, s2, t0

lw t2, 0(t1) # t2 = numbers[i]

# Chuyen so sang chuoi

la a0, buffer\_number

mv a1, t2

jal number\_to\_string

mv t3, a0 # t3 = length of stringified number (trả về từ number\_to\_string)

# Ghi chuoi so vao file (syscall 64 - WriteFile)

li a7, 64

lw a0, out\_fd

la a1, buffer\_number

mv a2, t3 # Số byte cần ghi (độ dài chuỗi)

ecall

# Ghi dau cach (neu khong phai so cuoi cung)

addi t0, s1, -1 # last\_index = count - 1

bge s0, t0, skip\_space # Nếu i == last\_index, bỏ qua dấu cách

li a7, 64

lw a0, out\_fd

la a1, space

li a2, 1 # Ghi 1 byte dấu cách

ecall

skip\_space:

addi s0, s0, 1 # i++

j write\_loop

write\_done:

# Ghi newline cuoi file

li a7, 64

lw a0, out\_fd

la a1, newline

li a2, 1

ecall

# Dong file output (syscall 57 - CloseFile)

li a7, 57

lw a0, out\_fd

ecall

# ... (khôi phục thanh ghi và ret) ...

msg\_file\_error\_openor: # Nhãn xử lý lỗi ghi file

li a7, 4

la a0, msg\_file\_error\_open

ecall

# ... (khôi phục thanh ghi nếu cần và ret hoặc thoát) ...

* **Giải thích:** Mở file output. Lặp qua mảng numbers đã sắp xếp, chuyển từng số thành chuỗi và ghi vào file, cách nhau bởi dấu cách. Cuối cùng ghi một ký tự xuống dòng và đóng file.

**1.4. Chạy thử mã lệnh và kết quả (Giả định)**

**1. Chuẩn bị file input:**

Tạo một file văn bản, ví dụ test\_numbers.txt, với nội dung:

10 5 -2 8 0 -5 12 1

**2. Quá trình chạy chương trình:**

* **Nhập tên file:**
* Enter filename: test\_numbers.txt
* **Đọc file:** Chương trình mở test\_numbers.txt, hàm read\_numbers sẽ đọc và parse các số. Mảng numbers sẽ chứa: [10, 5, -2, 8, 0, -5, 12, 1]. Biến count sẽ là 8.
* **Hiển thị Menu và Chọn Thuật Toán:**
* User select sorting algorithm:
* 1. Bubble Sort
* 2. Insertion Sort
* 3. Selection Sort
* 4. Quick Sort
* 5. Close
* Choice: 4

Người dùng chọn 4 (Quick Sort).

* **Thực thi Quick Sort:**
  + get\_time được gọi (ví dụ, trả về T1).
  + quick\_sort\_array gọi quick\_sort\_logic với mảng [10, 5, -2, 8, 0, -5, 12, 1].
  + Quick Sort sẽ sắp xếp mảng. Kết quả sau sắp xếp: [-5, -2, 0, 1, 5, 8, 10, 12].
  + flag\_negative\_numbers được gọi. Nó sẽ duyệt qua mảng đã sắp xếp và đặt các bit trong neg\_bitmask cho -5 và -2. (Như đã nói, việc này không ảnh hưởng đến kết quả cuối).
  + get\_time được gọi lại (ví dụ, trả về T2).
* **In thời gian thực thi:**
  + print\_time tính T2 - T1 (ví dụ, 0 ms hoặc một số nhỏ nếu dữ liệu ít).
* Execution time (ms): 0
* **Ghi kết quả ra file:**
  + write\_results được gọi.
  + File C:\\RISCV\\output5.txt sẽ được tạo/ghi đè với nội dung: -5 -2 0 1 5 8 10 12 (Lưu ý có một ký tự xuống dòng ở cuối file).
* **Quay lại Menu:** Chương trình hiển thị lại menu, người dùng có thể chọn thuật toán khác hoặc thoát.

**Kết luận về mã lệnh:**

* Chương trình có cấu trúc tốt, chia thành các hàm cho từng chức năng cụ thể.
* Xử lý file (đọc, ghi) và phân tích cú pháp số có vẻ hoạt động.
* Các thuật toán sắp xếp cơ bản và Quick Sort được triển khai.
* Việc đo thời gian cung cấp một cách để so sánh hiệu năng.
* Cách xử lý số âm bằng neg\_bitmask và flag\_negative\_numbers hiện tại không cần thiết vì các thuật toán sắp xếp đã xử lý đúng các số nguyên có dấu được đọc từ read\_numbers. Có thể đơn giản hóa bằng cách loại bỏ phần này.
* Một số syscall (như 1024 cho open file) là đặc thù cho môi trường RISC-V cụ thể (ví dụ: RARS hoặc một hệ điều hành tùy chỉnh).
* Tên file output được mã hóa cứng, có thể cải thiện bằng cách cho người dùng nhập hoặc tạo tên file output dựa trên input.
* Cần đảm bảo xử lý lỗi chặt chẽ hơn, ví dụ khi chuyển đổi ký tự sang số trong read\_numbers nếu gặp ký tự không phải số.

Đây là một bài tập khá toàn diện, bao gồm nhiều khía cạnh của lập trình Assembly.