Chắc chắn rồi, đây là giải thích chi tiết về các phần trong đoạn mã Assembly RISC-V bạn đã cung cấp, được chia theo chức năng:

**📁 Phần Khai Báo Dữ Liệu (**.data**)**

Đây là nơi bạn định nghĩa các biến, chuỗi ký tự, và vùng nhớ đệm sẽ được chương trình sử dụng.

Đoạn mã

.data

# Vùng nhớ lưu trữ số đọc từ file và các thông tin liên quan

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

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

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

# Mảng bitmask để đánh dấu số âm (hiện tại ít có tác dụng rõ ràng)

neg\_bitmask: .space 2500 # 2500 bytes = 20000 bits

# Buffer và chuỗi cho việc nhập tên file

input\_filename: .space 256

file\_read\_buffer: .space 1024 # Buffer đọc từng byte từ file trong hàm read\_numbers

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

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

# Chuỗi cho menu lựa chọn thuật toán

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

# Biến và chuỗi liên quan đến file và thời gian

fd: .word 0 # File descriptor cho file input

newline: .asciz "\n"

space: .string " " # Dấu cách (lưu ý .string có null terminator)

start\_time: .word 0 # Thời gian bắt đầu sắp xếp

end\_time: .word 0 # Thời gian kết thúc sắp xếp

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

# Dữ liệu cho file output

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 khi ghi file

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

char\_minus: .asciz "-" # Chuỗi dấu trừ (ít cần thiết)

* **Giải thích:** Phần này khai báo tất cả các dữ liệu tĩnh mà chương trình cần, từ các thông báo hiển thị cho người dùng, tên file, đến các vùng nhớ lớn để chứa số liệu và các buffer tạm.

**🚀 Phần Mã Lệnh (**.text**)**

Đây là nơi chứa các chỉ thị, logic thực thi của chương trình.

**🏁 Chương Trình Chính (**main**) và Xử Lý Menu**

Đoạn mã

.text

.globl main

main:

# In ra msg\_prompt\_input (Yêu cầu nhập tên file)

# ... (code in lời nhắc) ...

# Doc input\_filename (Đọc tên file từ người dùng)

# ... (code đọc tên file) ...

# Loai bo newline khoi input\_filename

# ... (code loại bỏ '\n' khỏi tên file) ...

replace\_null:

# ... (thay '\n' bằng '\0') ...

open\_input\_file:

# Mo file input (Mở file người dùng đã nhập)

# ... (code mở file, syscall 1024) ...

# Kiem tra loi mo file

# ... (xử lý nếu mở file lỗi) ...

# Luu file descriptor

# ... (lưu file descriptor vào biến fd) ...

# Goi ham doc so tu file

jal read\_numbers

menu\_loop:

# Hien thi menu

# ... (code in menu lựa chọn thuật toán, syscall 4) ...

# Doc lua chon cua nguoi dung

# ... (code đọc lựa chọn là số nguyên, syscall 5) ...

# Xu ly lua chon (Phân nhánh dựa trên lựa chọn)

li t0, 1

beq a0, t0, bubble\_sort\_array # Nếu chọn 1 -> Bubble Sort

li t0, 2

beq a0, t0, insertion\_sort\_array # Nếu chọn 2 -> Insertion Sort

li t0, 3

beq a0, t0, selection\_sort\_array # Nếu chọn 3 -> Selection Sort

li t0, 4

beq a0, t0, quick\_sort\_array # Nếu chọn 4 -> Quick Sort

li t0,5

beq a0, t0, exit # Nếu chọn 5 -> Thoát

# Lua chon khong hop le, thoat

j exit

file\_error\_open:

# In ra thong bao loi mo file

# ... (code in thông báo lỗi) ...

j exit

exit:

# Ket thuc chuong trinh (syscall 10)

li a7, 10

ecall

* **Giải thích:**
  + main: Điểm bắt đầu của chương trình.
  + Yêu cầu người dùng nhập tên file, đọc tên file, và loại bỏ ký tự xuống dòng thừa.
  + open\_input\_file: Mở file được chỉ định. Nếu thành công, gọi read\_numbers để đọc nội dung.
  + menu\_loop: Hiển thị menu các thuật toán, đọc lựa chọn của người dùng. Dựa vào lựa chọn (1-5), chương trình sẽ nhảy đến hàm sắp xếp tương ứng hoặc thoát.

**🔢 Đọc và Phân Tích Số Từ File (**read\_numbers**)**

Đoạn mã

read\_numbers:

# ... (Lưu các thanh ghi cần dùng vào stack) ...

# Reset count ve 0

# ... (code reset biến count) ...

# Khoi tao bien tam de parse so

li t0, 0 # Số hiện tại đang được parse

li t1, 0 # Cờ báo hiệu đang trong một số (1=true, 0=false)

li t6, 0 # Cờ báo hiệu số âm (1=âm, 0=dương)

read\_loop:

# Doc mot ky tu tu file (syscall 63 - Read, đọc 1 byte)

# ... (code đọc 1 byte từ file vào file\_read\_buffer) ...

# Kiem tra cuoi file (nếu syscall trả về 0 byte đã đọc)

beqz a0, read\_done

# Tai ky tu da doc

lb t2, 0(a1) # Lấy ký tự từ buffer đọc

# Kiem tra dau tru '-'

# ... (code kiểm tra nếu là '-' và đặt cờ t1, t6) ...

set\_negative:

# ... (đặt cờ số âm và cờ đang trong số) ...

not\_char\_minus:

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

# ... (code kiểm tra nếu là ' ' hoặc '\n' thì nhảy tới save\_number) ...

# Chuyen doi ky tu ASCII so sang gia tri so '0'-'9'

addi t2, t2, -48 # Chuyển ASCII '0'..'9' thành giá trị 0..9

# ... (code nhân số hiện tại với 10 rồi cộng với chữ số mới, đặt cờ t1) ...

j read\_loop

save\_number:

# Chi luu neu dang trong mot so (cờ t1 = 1)

# ... (code kiểm tra cờ t1) ...

# Ap dung dau (neu la so am, cờ t6 = 1)

# ... (code nếu t6=1 thì đổi dấu số t0) ...

save\_positive:

# Luu so vao mang numbers

# ... (code tính địa chỉ numbers[count] và lưu t0 vào đó) ...

# Tang bien dem count

# ... (code tăng biến count) ...

# Reset bien tam cho so tiep theo

# ... (reset t0, t1, t6 về 0) ...

j read\_loop

read\_done:

# Luu so cuoi cung neu dang parse do khi het file (tức là file kết thúc bằng một số)

# ... (logic tương tự save\_number cho số cuối cùng nếu có) ...

save\_last\_positive:

# ... (lưu số cuối cùng) ...

close\_file:

# Dong file input (syscall 57)

# ... (code đóng file) ...

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

* **Giải thích:** Hàm này có nhiệm vụ đọc file đầu vào ký tự theo ký tự, nhận diện các chữ số để tạo thành số nguyên hoàn chỉnh. Nó cũng xử lý dấu trừ (-) để đọc số âm. Khi gặp ký tự phân cách (dấu cách, xuống dòng) hoặc kết thúc file, số vừa được phân tích sẽ được lưu vào mảng numbers.

**⏳ Các Hàm Tiện Ích Thời Gian (**get\_time**,** print\_time**)**

Đoạn mã

get\_time:

# Syscall lay thoi gian (syscall 30, thường trả về ms trong MARS/RARS)

li a7, 30

ecall

ret # Thời gian trả về trong a0

print\_time:

# Tinh toan thoi gian thuc thi

la t0, start\_time

lw t1, 0(t0) # Lấy start\_time

la t0, end\_time

lw t2, 0(t0) # Lấy end\_time

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

# In thong bao "Execution time (ms): "

# ... (code in chuỗi msg\_execution\_time) ...

# In gia tri thoi gian (số nguyên t3)

# ... (code in số t3, syscall 1) ...

ret

* **Giải thích:**
  + get\_time: Gọi system call để lấy thời gian hiện tại của hệ thống.
  + print\_time: Tính toán sự chênh lệch giữa end\_time và start\_time rồi hiển thị ra màn hình.

**🔄 Các Hàm Sắp Xếp**

Mỗi hàm bao ngoài (xxx\_sort\_array) sẽ thực hiện:

1. Gọi get\_time để lấy thời gian bắt đầu.
2. Gọi hàm lõi thực hiện thuật toán sắp xếp (xxx\_sort\_core hoặc tương tự).
3. (Gọi flag\_negative\_numbers - hiện tại không có tác dụng rõ ràng đến kết quả).
4. Gọi get\_time để lấy thời gian kết thúc.
5. Gọi print\_time để hiển thị thời gian chạy.
6. Gọi write\_results để ghi mảng đã sắp xếp ra file.
7. Quay lại menu\_loop.

**🛁 Bubble Sort (**bubble\_sort\_array**,** bubble\_sort\_core**)**

Đoạn mã

bubble\_sort\_array:

# ... (lấy thời gian bắt đầu) ...

# Goi Bubble Sort core

la a0, numbers

lw a1, count

jal bubble\_sort\_core

# ... (đánh dấu số âm, lấy thời gian kết thúc, in thời gian, ghi file, về menu) ...

bubble\_sort\_core:

# Luu thanh ghi (ra, s0: array\_base, s1: size, s2: i)

# ...

# Khoi tao

mv s0, a0 # Địa chỉ đầu mảng

mv s1, a1 # Số lượng phần tử

li s2, 0 # i = 0 (biến đếm vòng lặp ngoài)

outer\_loop\_bubble\_sort:

# Vong lap ngoai: i from 0 to size-1

bge s2, s1, bubble\_done # if i >= size, done

li t0, 0 # j = 0 (biến đếm vòng lặp trong)

inner\_loop\_bubble\_sort:

# Vong lap trong: j from 0 to size-i-2

# ... (tính giới hạn cho j) ...

bge t0, t1, inner\_done\_bubble\_sort # if j >= size-i-1, done inner loop

# So sanh arr[j] va arr[j+1]

# ... (tính địa chỉ arr[j] và arr[j+1], load giá trị vào t3, t4) ...

# Neu arr[j] <= arr[j+1], khong doi cho

ble t3, t4, no\_swap\_bubble\_sort

# Hoan doi arr[j] va arr[j+1]

# ... (swap t3 và t4 tại các địa chỉ đã tính) ...

no\_swap\_bubble\_sort:

# Tang j

addi t0, t0, 1

j inner\_loop\_bubble\_sort

inner\_done\_bubble\_sort:

# Tang i

addi s2, s2, 1

j outer\_loop\_bubble\_sort

bubble\_done:

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

* **Giải thích:** bubble\_sort\_core triển khai thuật toán nổi bọt. Vòng lặp ngoài chạy size lần, vòng lặp trong đẩy phần tử lớn nhất/nhỏ nhất về cuối đoạn chưa sắp xếp bằng cách so sánh và hoán vị các cặp liền kề.

**📥 Insertion Sort (**insertion\_sort\_array**,** insertion\_sort\_array\_impl**)**

Đoạn mã

insertion\_sort\_array:

# ... (tương tự bubble\_sort\_array: lấy thời gian, gọi core, ...) ...

insertion\_sort\_array\_impl:

# Luu thanh ghi (ra, s0: array\_base, s1: size, s2: i)

# ...

# Khoi tao

mv s0, a0

mv s1, a1

li s2, 1 # i = 1 (bắt đầu từ phần tử thứ hai)

outer\_loop\_insertion:

# Vong lap ngoai: i from 1 to size-1

bge s2, s1, insertion\_done # if i >= size, done

# Lay key = arr[i]

# ... (tính địa chỉ arr[i], load giá trị vào t1 = key) ...

# Khoi tao j = i-1

addi t2, s2, -1 # t2 = j

inner\_loop\_insertion:

# Vong lap trong: j from i-1 down to 0 AND arr[j] > key

bltz t2, inner\_done\_insertion # Neu j < 0, done inner loop

# So sanh arr[j] voi key

# ... (tính địa chỉ arr[j], load giá trị vào t4 = arr[j]) ...

ble t4, t1, inner\_done\_insertion # Neu arr[j] <= key, done inner loop (tìm được vị trí chèn)

# Dich chuyen arr[j] sang arr[j+1]

# ... (lưu t4 (arr[j]) vào vị trí arr[j+1]) ...

# Giam j

addi t2, t2, -1

j inner\_loop\_insertion

inner\_done\_insertion:

# Chen key vao vi tri arr[j+1]

# ... (tính địa chỉ arr[j+1] (sau khi j đã giảm), lưu t1 (key) vào đó) ...

# Tang i

addi s2, s2, 1

j outer\_loop\_insertion

insertion\_done:

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

* **Giải thích:** insertion\_sort\_array\_impl triển khai thuật toán chèn. Nó duyệt qua mảng, với mỗi phần tử key = arr[i], nó tìm vị trí đúng trong đoạn arr[0...i-1] đã sắp xếp và chèn key vào đó, dời các phần tử lớn hơn key lên một vị trí.

**🎯 Selection Sort (**selection\_sort\_array**,** selection\_sort\_array\_impl**)**

Đoạn mã

selection\_sort\_array:

# ... (tương tự bubble\_sort\_array: lấy thời gian, gọi core, ...) ...

selection\_sort\_array\_impl:

# Luu thanh ghi (ra, s0: array\_base, s1: size, s2: i)

# ...

# Khoi tao

mv s0, a0

mv s1, a1

li s2, 0 # i = 0

outer\_loop\_selection:

# Vong lap ngoai: i from 0 to size-2

# ... (tính giới hạn cho i là size-1) ...

bge s2, t0, selection\_done # if i >= size-1, done

# Khoi tao min\_idx = i

mv t1, s2 # t1 = min\_idx

# Khoi tao j = i+1

addi t2, s2, 1 # t2 = j

inner\_loop\_selection:

# Vong lap trong: j from i+1 to size-1

bge t2, s1, inner\_done\_selection # if j >= size, done inner loop

# So sanh arr[j] voi arr[min\_idx]

# ... (load arr[j] vào t4, arr[min\_idx] vào t6) ...

# Neu arr[j] < arr[min\_idx], cap nhat min\_idx

bge t4, t6, no\_update\_min

mv t1, t2 # min\_idx = j

no\_update\_min:

# Tang j

addi t2, t2, 1

j inner\_loop\_selection

inner\_done\_selection:

# Hoan doi arr[i] voi arr[min\_idx] (neu min\_idx != i)

# ... (code hoán đổi giá trị tại địa chỉ arr[i] và arr[min\_idx]) ...

no\_swap\_selection:

# Tang i

addi s2, s2, 1

j outer\_loop\_selection

selection\_done:

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

* **Giải thích:** selection\_sort\_array\_impl triển khai thuật toán lựa chọn. Vòng lặp ngoài duyệt từ đầu mảng. Trong mỗi lượt, vòng lặp trong tìm phần tử nhỏ nhất trong đoạn chưa sắp xếp và đổi chỗ nó với phần tử đầu của đoạn đó (arr[i]).

**⚡ Quick Sort (**quick\_sort\_array**,** quick\_sort\_logic**,** partition\_elements**)**

Đoạn mã

quick\_sort\_array:

# ... (tương tự bubble\_sort\_array: lấy thời gian, gọi core, ...) ...

# Tham số cho quick\_sort\_logic: a0=numbers, a1=0 (left), a2=count-1 (right)

jal quick\_sort\_logic

# ...

quick\_sort\_logic: # (a0: array\_base, a1: left\_index, a2: right\_index)

# ... (Lưu thanh ghi vào stack, bao gồm ra, s0-s3) ...

# Luu cac tham so vao thanh ghi s (s0=array, s1=left, s2=right)

# ...

# Dieu kien dung de quy: if left >= right, return

bge s1, s2, quick\_sort\_end

# Goi ham phan hoach (partition)

# ... (copy s0, s1, s2 vào a0, a1, a2) ...

jal partition\_elements

mv s3, a0 # Luu chi so pivot (trả về từ partition\_elements)

# De quy Quick Sort cho phan ben trai pivot

# ... (gọi quick\_sort\_logic với right = pivot\_index - 1) ...

# De quy Quick Sort cho phan ben phai pivot

# ... (gọi quick\_sort\_logic với left = pivot\_index + 1) ...

quick\_sort\_end:

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

partition\_elements: # (a0: array\_base, a1: left, a2: right)

# ... (Lưu thanh ghi vào stack, bao gồm ra, s0-s4) ...

# Luu tham so (s0=array, s1=left, s2=right)

# ...

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

# ... (load arr[right] vào s3 = pivot\_value) ...

# Khoi tao i = left - 1 (s4 = i, chỉ số của phần tử nhỏ hơn cuối cùng)

# Khoi tao j = left (t1 = j, biến chạy)

# ...

partition\_loop\_elements:

# Vong lap j tu left den right-1

bge t1, s2, partition\_elements\_done # if j >= right, done loop

# Tai phan tu arr[j]

# ... (load arr[j] vào t2) ...

# Neu arr[j] <= pivot\_value

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

# Tang i

addi s4, s4, 1

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

# ... (code hoán đổi giá trị tại arr[i] và arr[j]) ...

skip\_swap:

# Tang j

addi t1, t1, 1

j partition\_loop\_elements

partition\_elements\_done:

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

# ... (tính i+1, hoán đổi giá trị tại arr[i+1] và arr[right]) ...

# Tra ve chi so cua pivot (i+1)

mv a0, s4 # s4 lúc này là i+1, vị trí mới của pivot

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

* **Giải thích:**
  + quick\_sort\_logic: Hàm đệ quy chính của Quick Sort. Nếu left < right, nó gọi partition\_elements. Sau đó, nó gọi đệ quy cho hai mảng con bên trái và bên phải của pivot.
  + partition\_elements: Chọn phần tử cuối cùng làm pivot. Sắp xếp lại đoạn mảng sao cho các phần tử nhỏ hơn hoặc bằng pivot nằm bên trái pivot, và các phần tử lớn hơn nằm bên phải. Trả về chỉ số cuối cùng của pivot.

**📝 Chuyển Đổi Số Sang Chuỗi và Ghi File**

**↔️ Chuyển Số Sang Chuỗi (**number\_to\_string**,** str\_reverse**)**

Đoạn mã

number\_to\_string: # (a0: buffer\_address, a1: number\_to\_convert) -> a0: length

# ... (Lưu thanh ghi) ...

# Khoi tao (s0=buffer, s1=number, s2=length, s3=negative\_flag, s4=temp\_pointer\_in\_buffer)

# ...

# Xu ly so 0 (trường hợp đặc biệt)

# ...

check\_sign:

# Kiem tra so am

# ... (nếu âm, đặt s3=1, đổi s1 thành dương) ...

convert\_digits:

# Chuyen doi cac chu so (luu nguoc vao buffer s4)

digit\_loop:

# ... (lấy last\_digit = number % 10, number = number / 10, chuyển digit sang ASCII, lưu vào buffer, tăng s4, s2) ...

finalize\_string:

# Them dau '-' neu am (dựa vào s3, lưu '-' vào buffer tại s4)

# ...

reverse\_string:

# Dao nguoc chuoi trong buffer (buffer bắt đầu từ s0, kết thúc trước s4)

# ... (gọi str\_reverse(s0, s4-1)) ...

# Tra ve do dai trong a0 (là s2)

num\_to\_str\_done:

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

str\_reverse: # (a0: start\_ptr, a1: end\_ptr)

# ... (logic đảo ngược chuỗi bằng cách hoán vị ký tự từ hai đầu vào giữa) ...

* **Giải thích:**
  + number\_to\_string: Chuyển một số nguyên (a1) thành chuỗi ASCII và lưu vào buffer (a0). Nó xử lý số 0, số âm. Các chữ số được tạo ra theo thứ tự ngược, sau đó str\_reverse được gọi. Hàm trả về độ dài của chuỗi số trong a0.
  + str\_reverse: Đảo ngược một chuỗi ký tự tại chỗ.

**💾 Ghi Kết Quả Ra File (**write\_results**)**

Đoạn mã

write\_results:

# ... (Lưu thanh ghi) ...

# Mo file output (syscall 1024, tên file "C:\\RISCV\\output5.txt", cờ ghi a1=1)

# ... (kiểm tra lỗi, lưu file descriptor vào out\_fd) ...

# Khoi tao vong lap ghi file (s0=i, s1=count, s2=numbers\_base)

# ...

write\_loop:

# Kiem tra ket thuc vong lap (i >= count)

# ...

# Tai so hien tai numbers[i]

# ... (load numbers[i] vào t2) ...

# Chuyen so sang chuoi

# ... (gọi number\_to\_string(buffer\_number, t2), kết quả độ dài trong t3) ...

# Ghi chuoi so vao file (syscall 64 - WriteFile, fd=out\_fd, buffer=buffer\_number, length=t3)

# ...

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

# ... (kiểm tra i < count-1, ghi " " (dài 1 byte) vào file) ...

skip\_space:

# Tang bien dem i

addi s0, s0, 1

j write\_loop

write\_done:

# Ghi newline cuoi file

# ... (ghi "\n" (dài 1 byte) vào file) ...

# Dong file output (syscall 57)

# ...

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

msg\_file\_error\_openor: # Lỗi ghi file

# ...

* **Giải thích:** Mở file output5.txt ở chế độ ghi. Duyệt qua mảng numbers đã sắp xếp. Với mỗi số, chuyển nó thành chuỗi bằng number\_to\_string rồi ghi chuỗi đó (cùng với dấu cách) vào file output bằng syscall 64. Cuối cùng, ghi một ký tự xuống dòng và đóng file.

**👻 Phần Mã Ít Rõ Ràng hoặc Có Thể Không Dùng Đến**

* flag\_negative\_numbers và neg\_bitmask:

Như đã phân tích, hàm flag\_negative\_numbers được gọi sau mỗi lần sắp xếp để đặt các bit trong neg\_bitmask nếu một số là âm. Tuy nhiên, neg\_bitmask này không được sử dụng ở bất kỳ đâu khác, đặc biệt là không dùng để khôi phục dấu hay ảnh hưởng đến việc ghi file. Các thuật toán sắp xếp trong mã này (ví dụ, Quick Sort thông qua so sánh trực tiếp) đã xử lý đúng các số nguyên có dấu. Do đó, phần này hiện tại có vẻ như không có tác dụng cụ thể đến kết quả cuối cùng của chương trình. Nó có thể là một phần còn lại từ một ý tưởng thiết kế khác hoặc chưa được hoàn thiện.

* **Các nhãn** parse\_loop**,** next\_char **(sau** selection\_sort\_array\_impl**):**

Đoạn mã

parse\_loop:

# (Day la doan code khong duoc su dung, s0, s1, s2 khong ro ngu canh)

# ...

next\_char:

# ...

Đoạn code này có vẻ như là một phần của một logic phân tích cú pháp khác, không được gọi từ luồng chính của chương trình và có thể là mã thừa hoặc đang phát triển dang dở. Logic phân tích cú pháp chính nằm trong hàm read\_numbers.

* Các nhãn check\_negative, positive\_conversion, reverse\_digits (gần cuối file):

Đây thực chất là các phần logic bên trong hàm number\_to\_string. Việc đặt chúng thành các nhãn toàn cục có thể gây nhầm lẫn nhưng không ảnh hưởng đến hoạt động nếu chúng chỉ được nhảy tới từ bên trong number\_to\_string.

* Hàm write\_positive\_numbers:

Hàm này không được gọi trong luồng chính và có vẻ như là một phiên bản khác hoặc chưa hoàn chỉnh của logic ghi file, tập trung vào việc ghi số dương (bằng cách lấy giá trị tuyệt đối). Logic ghi file chính thức là write\_results.

Hy vọng giải thích chi tiết này giúp bạn hiểu rõ hơn về cấu trúc và hoạt động của từng phần trong đoạn mã Assembly của mình! 🎢