HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



BÁO CÁO CUỐI KỲ

**THỰC HÀNH KIẾN TRÚC MÁY TÍNH**

**NHÓM 14**

**Giảng viên hướng dẫn**

Ngô Lam Trung

**Trợ Giảng**

Đỗ Quốc Minh

**Thành viên nhóm**

Phan Khánh Vũ

Hoàng Văn Thắng

***Mục lục***

[1. Assignment 6: Mô phỏng ổ đĩa RAID 5 3](#_Toc199366320)

[**1.1.** **Mô tả bài toán:** 3](#_Toc199366321)

[**1.2.** **Thuật toán sử dụng:** 4](#_Toc199366322)

[1.2.1. Khởi tạo 4](#_Toc199366323)

[1.2.2. Nhập liệu và kiểm tra độ dài : 4](#_Toc199366324)

[1.2.3. Xử lý và phân phối dữ liệu (RAID 5 Logic): 4](#_Toc199366325)

[1.2.4. Thủ tục HEX (Chuyển đổi sang Hexa): 5](#_Toc199366326)

[**1.3.** **Đoạn mã lệnh và giải thích** 6](#_Toc199366327)

[1.3.1. Khai báo dữ liệu (.data) 6](#_Toc199366328)

[1.3.2. Main và input 6](#_Toc199366329)

[1.3.3. Kiểm tra độ dài chuỗi (length, check\_char, test\_length, error) 7](#_Toc199366330)

[1.3.4. Thủ tục HEX (Chuyển byte sang 2 ký tự Hexa ASCII) 9](#_Toc199366331)

[1.3.5. Mô phỏng RAID 5 (block1, block2, block3) 10](#_Toc199366332)

[1.4. Chạy thử mã lệnh và kết quả: 24](#_Toc199366333)

[1.4.1. Kết quả khi chạy chương trình với đầu vào “*DCE.\*\*\*\*ABCD1234HUSTHUST*”: 24](#_Toc199366334)

[1.4.2. Kết quả khi chạy với đầu vào rỗng: 24](#_Toc199366335)

[1.4.3. Kết quả khi chạy đầu vào ABCXYZ: 24](#_Toc199366336)

[1.4.4. Kết quả khi chạy với đầu vào HUSTABCD: 25](#_Toc199366337)

[1.4.5. Kết quả khi chạy với đầu vào ABCDXYZT1234567890125487DCE.\*\*\*\*: 25](#_Toc199366338)

[2. Assignment 9: Kiểm thử các thuật toán sắp xếp 26](#_Toc199366339)

[**2.1.** **Mô tả bài toán:** 26](#_Toc199366340)

[**2.2.** **Thuật toán sử dụng:** 26](#_Toc199366341)

[2.2.1. Luồng điều khiển chính (main): 26](#_Toc199366342)

[2.2.2. Đọc và phân tích cú pháp số (read\_numbers): 26](#_Toc199366343)

[2.2.3. Các thuật toán sắp xếp: 27](#_Toc199366344)

[2.2.4. Đo thời gian (get\_time, print\_time): 27](#_Toc199366345)

[2.2.5. Chuyển đổi số sang Chuỗi (number\_to\_string, str\_reverse): 27](#_Toc199366346)

[2.2.6. Ghi File kết quả (write\_results): 27](#_Toc199366347)

[2.2.7. Xử lý số âm (Bitmask – flag\_negative\_number, neg\_bitmask): 28](#_Toc199366348)

[**2.3.** **Đoạn mã lệnh và giải thích:** 28](#_Toc199366349)

[2.3.1. Khai báo dữ liệu(.data) 28](#_Toc199366350)

[2.3.2. Hàm main và vòng lặp menu: 29](#_Toc199366351)

[2.3.3. Hàm read\_numbers (Đọc và phân tích cú pháp số từ File) 32](#_Toc199366352)

[2.3.4. Các hàm sắp xếp 36](#_Toc199366353)

[2.3.4.1. Quick Sort (quick\_sort\_array, quick\_sort\_logic, partition\_elements) 36](#_Toc199366354)

[2.3.4.2. Bubble Sort (bubble\_sort\_array, bubble\_sort\_core) 41](#_Toc199366355)

[2.3.4.3. Insertion Sort (insertion\_sort\_array, insertion\_sort\_arrayimpl) 43](#_Toc199366356)

[2.3.4.4. Selecttion Sort (selection\_sort\_array, selection\_sort\_array\_impl) 44](#_Toc199366357)

[2.3.5. Hàm flag\_negative\_numbers 45](#_Toc199366358)

[2.3.6. Hàm get\_time và print\_time 47](#_Toc199366359)

[2.3.7. Hàm number\_to\_string và str\_reverse 48](#_Toc199366360)

[2.3.8. Hàm write\_results (Ghi kết quả đã sắp xếp ra File ) 51](#_Toc199366361)

[**2.4.** **Kết quả chạy mẫu:** 54](#_Toc199366362)

# **Assignment 6: Mô phỏng ổ đĩa RAID 5**

## **Mô tả bài toán:**

Hệ thống ổ đĩa RAID5 cần tối thiểu 3 ổ đĩa cứng, trong đó phần dữ liệu parity sẽ được chứa lần lượt lên 3 ổ đĩa như trong hình bên. Hãy viết chương trình mô phỏng hoạt động của RAID 5 với 3 ổ đĩa, với giả định rằng, mỗi block dữ liệu có 4 kí tự. Giao diện như trong minh họa dưới. Giới hạn chuỗi kí tự nhập vào có độ dài là bội của 8.

Trong ví dụ sau, chuỗi kí tự nhập vào từ bàn phím (DCE.\*\*\*\*ABCD1234HUSTHUST) sẽ được chia thành các block 4 byte. Block 4 byte đầu tiên “DCE.” sẽ được lưu trên Disk 1, Block 4 byte tiếp theo “\*\*\*\*” sẽ lưu trên Disk 2, dữ liệu trên Disk 3 sẽ là 4 byte parity được tính từ 2 block đầu tiên với mã ASCII là 6e=’D’ xor ‘\*’ ; 69=’C’ xor ‘\*’; 6f=’E’ xor ‘\*’ ; 04=’.’ xor ‘\*’

A close-up of a computer code

AI-generated content may be incorrect.

Sơ đồ lưu trữ:

|  |  |  |  |
| --- | --- | --- | --- |
| **Stripe** | **Disk 1** | **Disk 2** | **Disk 3** |
| 1 | Data Block 1 | Data Block 2 | Parity (1,2) |
| 2 | Data Block 3 | Parity (3,4) | Data Block 4 |
| 3 | Parity (5,6) | Data Block 5 | Data Block 6 |
| ... | ... | ... | ... |

## **Thuật toán sử dụng:**

### Khởi tạo

Định nghĩa các chuỗi thông báo, vùng nhớ cho đĩa (disk1, disk2, disk3), mảng lưu trữ parity (array), và buffer cho chuỗi nhập liệu (string).

Thiết lập các con trỏ thanh ghi ban đầu cho các vùng nhớ đĩa và mảng parity.

### Nhập liệu và kiểm tra độ dài :

Hiển thị lời nhắc "Nhap chuoi ki tu : ".

Đọc chuỗi ký tự từ người dùng.

Kiểm tra độ dài chuỗi:

* Đếm số ký tự trong chuỗi (không bao gồm ký tự xuống dòng \n).
  + Nếu chuỗi rỗng (chỉ có \n) hoặc độ dài không phải là bội số của 8, hiển thị thông báo lỗi "Do dai chuoi khong hop le! Chieu dai cua chuoi phai chia het cho 8. Hay nhap lai." và quay lại bước nhập liệu.
  + Nếu độ dài hợp lệ, tiếp tục.

### Xử lý và phân phối dữ liệu (RAID 5 Logic):

Chương trình xử lý dữ liệu theo từng cụm 24 byte (tương ứng với 3 stripe, mỗi stripe xử lý 8 byte dữ liệu đầu vào).

Vòng lặp chính (nextloop) sẽ lặp qua các cụm 24 byte này. Bên trong mỗi cụm, có 3 giai đoạn (block1, block2, block3) tương ứng với 3 cách phân phối parity:

* Giai đoạn 1 (block1): Parity trên Disk 3
  + Đọc 4 ký tự đầu tiên của 8 byte hiện tại vào disk1.
  + Đọc 4 ký tự tiếp theo của 8 byte hiện tại vào disk2.
  + Tính toán parity: Với mỗi cặp ký tự tương ứng từ disk1 và disk2, + thực hiện phép XOR. Lưu 4 byte kết quả vào mảng array.
  + Hiển thị nội dung disk1 (dạng ký tự), disk2 (dạng ký tự), và parity từ array (dạng hexa, sử dụng thủ tục HEX).
* Giai đoạn 2 (block2): Parity trên Disk 2
  + Con trỏ chuỗi đầu vào được dịch chuyển 8 byte so với đầu cụm 24 byte.
  + Đọc 4 ký tự đầu tiên của 8 byte tiếp theo vào disk1.
  + Đọc 4 ký tự tiếp theo của 8 byte này vào disk3.
  + Tính toán parity giữa dữ liệu trên disk1 và disk3. Lưu 4 byte kết quả vào mảng array.
  + Hiển thị nội dung disk1 (dạng ký tự), parity từ array (dạng hexa) cho vị trí Disk 2, và disk3 (dạng ký tự).
* Giai đoạn 3 (block3): Parity trên Disk 1
  + Con trỏ chuỗi đầu vào được dịch chuyển 16 byte so với đầu cụm 24 byte.
  + Đọc 4 ký tự đầu tiên của 8 byte cuối cùng trong cụm 24 byte vào disk2.
  + Đọc 4 ký tự tiếp theo của 8 byte này vào disk3.
  + Tính toán parity giữa dữ liệu trên disk2 và disk3. Lưu 4 byte kết quả vào mảng array.
  + Hiển thị parity từ array (dạng hexa) cho vị trí Disk 1, nội dung disk2 (dạng ký tự), và disk3 (dạng ký tự).
  + Sau khi xử lý một cụm 24 byte (3 stripe), con trỏ chuỗi đầu vào (s0) được cập nhật để trỏ đến cụm 8 byte tiếp theo cho block1 của chu kỳ mới (nếu còn dữ liệu).

### Thủ tục HEX (Chuyển đổi sang Hexa):

Nhận một byte dữ liệu trong thanh ghi s8.

Chuyển đổi byte này thành 2 ký tự ASCII biểu diễn dạng hexa.

Ví dụ: byte 0x6E sẽ được chuyển thành hai ký tự '6' và 'e'.

In hai ký tự này ra màn hình.

* + 1. Lặp lại hoặc kết thúc:

Sau khi xử lý toàn bộ chuỗi đầu vào, chương trình hiển thị một dòng gạch ngang phân cách.

Hỏi người dùng "Try another string?".

Nếu người dùng chọn "Yes", xóa nội dung chuỗi cũ và quay lại bước Nhập liệu.

Nếu người dùng chọn "No" hoặc "Cancel", kết thúc chương trình.

## **Đoạn mã lệnh và giải thích**

### Khai báo dữ liệu (.data)

|  |
| --- |
| .data  prompt: .asciz "Nhap chuoi ki tu : "  # ASCII into hexa  hex: .byte '0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'  disk1: .space 4  disk2: .space 4  disk3: .space 4  array: .space 32 # Store parities (results for data XOR)  string: .space 5000 # Input string  newline: .asciz "\n" # Ký tự xuống dòng  error\_message: .asciz "Do dai chuoi khong hop le! Chieu dai cua chuoi phai chia het cho 8. Hay nhap lai.\n"  disk: .asciz " Disk 1 Disk 2 Disk 3\n"  msg1: .asciz " -------------- -------------- --------------\n"  msg2: .asciz "| "  msg3: .asciz " | "  msg4: .asciz "[[ "  msg5: .asciz "]] "  comma: .asciz ","  message: .asciz "Try another string?" |

- Giải thích: Đoạn này khai báo các chuỗi hằng, bảng chuyển đổi hexa, và các vùng nhớ sẽ được sử dụng để lưu trữ dữ liệu tạm thời cho các đĩa, parity, và chuỗi đầu vào.

### Main và input

|  |
| --- |
| .text  main: # Bắt đầu của chương trình chính  la s1, disk1 # s1 = address of disk 1  la s2, disk2 # s2 = address of disk 2  la s3, disk3 # s3 = address of disk 3  la a2, array # Address of parities    j input  nop    input: # Bắt đầu phần nhập liệu  li a7, 4 # Print " Nhap chuoi ky tu"  la a0, prompt  ecall    li a7, 8 # Get string (Syscall num 8)  la a0, string  li a1, 1000 # maximum number of characters to read  ecall    mv s0, a0 # s0 = address of input string |

* Giải thích:

main khởi tạo các thanh ghi s1, s2, s3 trỏ tới vùng nhớ của disk1, disk2, disk3 và a2 trỏ tới array (dùng để lưu các byte parity).

input hiển thị lời nhắc và đọc chuỗi ký tự từ người dùng, lưu địa chỉ bắt đầu của chuỗi vào s0.

### Kiểm tra độ dài chuỗi (length, check\_char, test\_length, error)

|  |
| --- |
| # ---------------- Check whether input string's length is multiple of 8 ----------------  length: # Kiểm tra độ dài chuỗi  addi t3, zero, 0 # t3 = length  addi t0, zero, 0 # t0 = index  check\_char:  # Check \n?  add t1, s0, t0 # t1 = address of string[i]  lb t2, 0(t1) # t2 = string[i]  li s4, 10 # '\n' = 10 ASCII  beq t2, s4, test\_length # if string[i] = '\n', then jump test\_length  nop    addi t3, t3, 1 # length++  addi t0, t0, 1 # index++  j check\_char  nop    test\_length: # Kiểm tra tính hợp lệ của độ dài chuỗi  mv t5, t3 # t5 = string length  beq t0, zero, error # If only '\n' -> error    andi t1, t3, 0x0000000f # t1 = (t3 & 0xF) last byte  bne t1, zero, test1 # if (t3 & 0xF) != 0, then jump 'test1'  j input\_prompt # else jump 'block1'  nop  test1:  li s11, 8 # s11 = 8  beq t1, s11, input\_prompt # if (t3 & 0xF) == 8, then jump 'block1'  j error # else jump 'error'  nop    error: # Xử lý lỗi độ dài không hợp lệ  li a7, 4 # Print error\_message  la a0, error\_message  ecall    j input  nop    input\_prompt:  li a7, 4  la a0, disk  ecall  li a7, 4  la a0, msg1  ecall  j block1 |

- Giải thích:

length và check\_char lặp qua chuỗi đầu vào để tính độ dài thực tế (t3) cho đến khi gặp ký tự \n.

test\_length kiểm tra:

* Nếu độ dài là 0 (chỉ có \n), báo lỗi.
* Sử dụng phép toán bitwise andi t1, t3, 0x0000000f để lấy 4 bit cuối của độ dài. Nếu kết quả này là 0 (ví dụ: độ dài 16, 32,...) hoặc 8 (ví dụ: độ dài 8, 24,...), thì độ dài được coi là hợp lệ (là bội của 8). Nếu không, nhảy đến error.
* Nếu độ dài hợp lệ, in ra tiêu đề bảng Disk và nhảy đến block1 để bắt đầu xử lý.
* error in thông báo lỗi và quay lại input.

### Thủ tục HEX (Chuyển byte sang 2 ký tự Hexa ASCII)

|  |
| --- |
| HEX: # Nhãn: Thủ tục chuyển đổi một byte sang 2 ký tự hexa ASCII  # -------------------- Get parities --------------------  # Đầu vào s8 chứa byte parity, chuyển từ số sang hexa (ASCII)  li t4, 7 # Khởi tạo biến đếm t4 = 7 (cho 8 nibble, nhưng chỉ in 2 cuối)    loopH:  blt t4, zero, endloopH # t4 < 0 -> endloop  slli s6, t4, 2 # s6 = t4\*4  srl a0, s8, s6 # a0 = s8 >> s6  andi a0, a0, 0x0000000f # Get the last byte of a0  la s7, hex # s7 = adrress of hex  add s7, s7, a0  li a4, 1  bgt t4, a4, nextc # if t4 > 1 , jump to nextC  lb a0, 0(s7) # Print hex[a0]  li a7, 11  ecall  nextc:  addi t4, t4, -1 # t4 --  j loopH  nop  endloopH:  jr ra  nop |

* Giải thích:

Thủ tục này nhận một byte trong s8.

Nó tách byte đó thành hai nibble (4 bit).

Nibble cao (4 bit đầu): Dịch phải s8 4 vị trí, sau đó dùng andi với 0x0f để lấy giá trị.

Nibble thấp (4 bit cuối): Dùng andi với 0x0f trực tiếp trên s8.

Mỗi giá trị nibble (0-15) được dùng làm chỉ số để tra cứu trong bảng hex và lấy ký tự ASCII tương ứng ('0'-'9', 'a'-'f').

In ra 2 ký tự hexa này.

jr ra quay lại nơi gọi.

*Lưu ý:* Phần mã gốc của HEX có vẻ được thiết kế để xử lý một word 32-bit và in ra 8 ký tự hexa, nhưng trong ngữ cảnh này nó chỉ được gọi để in 2 ký tự hexa cuối cùng của byte trong s8 (do bgt t4, a4, nextc với a4=1). Đoạn mã giải thích ở trên đã được đơn giản hóa cho mục đích chuyển đổi 1 byte thành 2 ký tự hexa, điều này phù hợp hơn với cách nó được sử dụng. Nếu giữ nguyên mã gốc, t4 nên được khởi tạo là 1 để nó thực hiện 2 lần lặp cho 2 nibble.

### Mô phỏng RAID 5 (block1, block2, block3)

|  |
| --- |
| #------------------------------ RAID5 SIMULATION------------------------------------  RAID5: # Đánh dấu bắt đầu phần mô phỏng RAID5 (không được nhảy trực tiếp đến)  # Block 1 : byte parity is stored in disk 3  # Block 2 : byte parity is stored in disk 2  # Block 3 : byte parity is stored in disk 1  block1:  # Function block1: First 2 4-byte blocks are stored in disk1, disk2; parity is stored in disk3  addi t0, zero, 0  addi s9, zero, 0  addi s8, zero, 0  la s1, disk1  la s2, disk2  la a2, array    print11: # In phần mở đầu cho Disk 1  li a7, 4  la a0, msg2  ecall    b11: # Vòng lặp xử lý byte cho block1  # Store into disk1  lb t1, 0(s0) # t1 = first value of input string  addi t3, t3, -1 # t3 = length - 1  sb t1, (s1) # store t1 into disk1  b12: # Xử lý byte thứ hai cho cặp  # Store ịnto disk2  addi s5, s0, 4 # s5 = s0 + 4  lb t2, 0(s5) # t2 = string[5]  addi t3, t3, -1 # t3 = t3 - 1  sb t2, 0(s2) # store t2 into disk2  b13: # Tính và lưu parity  # Store XOR result into disk3  xor a3, t1, t2 # a3 = t1 xor t2  sw a3, 0(a2) # Store a3 into a2  addi a2, a2, 4 # Parity string  addi t0, t0, 1 # Next char  addi s0, s0, 1 # Eliminate considered char, eg : "D"  addi s1, s1, 1 # Address of disk 1 + 1  addi s2, s2, 1 # Address of disk 2 + 1  li a6, 3 # a6 = 3  bgt t0, a6, reset # 4 byte are considered --> reset disk  j b11  nop  reset: # Reset con trỏ buffer disk để chuẩn bị in  la s1, disk1  la s2, disk2    print12: # Vòng lặp in nội dung buffer disk1  lb a0, 0(s1) # Print each char in disk1  li a7, 11 # syscall 11 (print char)  ecall  addi s9, s9, 1 # Tăng biến đếm s9 (số ký tự đã in của disk1)  addi s1, s1, 1 # Tăng con trỏ buffer disk1  bgt s9, a6, next11 # Print 4 times --> end priting disk1  j print12  nop    next11: # Chuẩn bị in disk2  li a7, 4  la a0, msg3  ecall  li a7, 4  la a0, msg2  ecall    print13: # Vòng lặp in nội dung buffer disk2  lb a0, 0(s2) # Nạp byte từ buffer disk2 (s2) vào a0  li a7, 11 # syscall 11 (print char)  ecall  addi s8, s8, 1 # Tăng biến đếm s8 (số ký tự đã in của disk2)  addi s2, s2, 1 # Tăng con trỏ buffer disk2 (s2)  bgt s8, a6, next12 # Print 4 times --> end printing disk2  j print13  nop    next12: # Chuẩn bị in parity (disk3)  li a7, 4  la a0, msg3  ecall  li a7, 4  la a0, msg4  ecall  la a2, array # a2 = address of parity string[i]  addi s9, zero, 0 # Reset biến đếm s9 = 0 (cho việc in parity)    print14: # Convert parity string --> ASCII and print  lb s8, 0(a2) # s8 = adress of parity string[i]  jal HEX  nop  li a7, 4  la a0, comma  ecall    addi s9, s9, 1 # Parity string's index + 1  addi a2, a2, 4 # Tăng con trỏ mảng parity (a2) lên 4 byte  li a5, 2 # Nạp giá trị 2 vào a5 (để in 3 dấu phẩy cho 4 parity)  bgt s9, a5, endisk1 # Print first 3 parities with ','  j print14  endisk1: # In byte parity cuối cùng (không có dấu phẩy sau)  lb s8, 0(a2) # Nạp byte parity cuối cùng vào s8  jal HEX  nop  li a7, 4  la a0, msg5  ecall    li a7, 4  la a0, newline  ecall  beq t3, zero, exit1 # If string length = 0 --> exit  j block2 # else --> block2  nop |

Giải thích block1:

Vòng lặp b11 đến b13:

* Đọc 1 byte từ s0 (chuỗi đầu vào) vào disk1.
* Đọc 1 byte từ s0+4 (chuỗi đầu vào) vào disk2.
* Tính XOR của 2 byte này, lưu kết quả vào array.
* Tăng con trỏ s0, s1 (disk1), s2 (disk2), a2 (array).
* Biến t3 (độ dài còn lại của chuỗi) được giảm 2 lần cho mỗi cặp byte xử lý (1 lần ở b11, 1 lần ở b12 trong mã gốc). Điều này có nghĩa là sau khi xử lý 4 cặp byte (1 stripe), t3 sẽ giảm đi 8.

Sau khi lặp 4 lần (xử lý 4 byte cho mỗi block), các con trỏ s1, s2 được reset.

In nội dung disk1 (4 ký tự).

In nội dung disk2 (4 ký tự).

In 4 byte parity từ array, mỗi byte được chuyển sang hexa bằng HEX và cách nhau bằng dấu phẩy (trừ byte cuối).

Con trỏ s0 đã tự động dịch chuyển 4 byte trong quá trình đọc vào disk1. *Quan trọng:* Để block2 bắt đầu từ dữ liệu mới, s0 cần được điều chỉnh thêm ở đầu block2 để bỏ qua 4 byte đã được đọc vào disk2 trong block1. Mã gốc thực hiện điều này.

|  |
| --- |
| #----------------------------------------  block2: # Funtion block2: Next 2 4-byte blocks are stored in disk1, disk3; parity is stored in disk2  la a2, array # Nạp địa chỉ mảng parity 'array' vào a2 (parity cho Disk2)  la s1, disk1 # Nạp địa chỉ buffer disk1 vào s1  la s3, disk3 # Nạp địa chỉ buffer disk3 vào s3  addi s0, s0, 4 # Tăng con trỏ chuỗi đầu vào (s0) lên 4 (bỏ qua 4 byte đã xử lý ở block1 cho disk1/disk2)  addi t0, zero, 0 # Reset biến đếm vòng lặp t0 = 0    print21: # print "| "  li a7, 4  la a0, msg2  ecall  b21: # Store 4 bytes into disk1  lb t1, 0(s0) # Nạp byte từ chuỗi đầu vào (s0) vào t1 (cho Disk1)  addi t3, t3, -1 # string\_length --  sb t1, 0(s1) # Lưu byte t1 vào buffer disk1  b23: # Store next 4 bytes into disk3  addi s5, s0, 4 # string addr + 4  lb t2, 0(s5)  addi t3, t3, -1 # length --  sb t2, 0(s3)    b22: # Store XOR result into disk2  xor a3, t1, t2 # Tính XOR của t1 và t2 -> a3 (parity)  sw a3, 0(a2) # Lưu byte parity a3 (như word) vào mảng 'array' (cho Disk2)  addi a2, a2, 4 # Tăng con trỏ mảng parity (a2)  addi t0, t0, 1 # Tăng biến đếm vòng lặp t0  addi s0, s0, 1 # Tăng con trỏ chuỗi đầu vào (s0)  addi s1, s1, 1 # Tăng con trỏ buffer disk1 (s1)  addi s3, s3, 1 # Tăng con trỏ buffer disk3 (s3)  bgt t0, a6, reset2 # Nếu t0 > 3, nhảy đến 'reset2'  j b21  nop  reset2: # Reset disks  la s1, disk1 # Nạp lại địa chỉ buffer disk1 vào s1  la s3, disk3 # Nạp lại địa chỉ buffer disk3 vào s3  addi s9, zero, 0 # Reset biến đếm s9 = 0 (cho việc in)    print22: # In nội dung buffer disk1  lb a0, 0(s1) # Nạp byte từ buffer disk1 (s1) vào a0  li a7, 11 # syscall 11 (print char)  ecall  addi s9, s9, 1 # Tăng biến đếm s9  addi s1, s1, 1 # Tăng con trỏ buffer disk1 (s1)  bgt s9, a6, next21 # Nếu s9 > 3, nhảy đến 'next21'  j print22  nop    next21: # Chuẩn bị in parity (Disk2)  li a7, 4  la a0, msg3  ecall  la a2, array # Nạp lại địa chỉ mảng parity 'array' vào a2  addi s9, zero, 0 # Reset biến đếm s9 = 0  li a7, 4  la a0, msg4  ecall    print23: # Vòng lặp in các byte parity (Disk2)  lb s8, 0(a2) # Nạp byte parity từ mảng (a2) vào s8  jal HEX  nop  li a7, 4  la a0, comma  ecall  addi s9, s9, 1 # Tăng biến đếm s9  addi a2, a2, 4 # Tăng con trỏ mảng parity (a2)  bgt s9, a5, next22 # Nếu s9 > 2, nhảy đến 'next22'  j print23  nop    next22: # In byte parity cuối cùng (Disk2)  lb s8, (a2) # Nạp byte parity cuối cùng vào s8  jal HEX  nop    li a7, 4  la a0, msg5  ecall    li a7, 4  la a0, msg2  ecall  addi s8, zero, 0 # Reset biến đếm s8 = 0 (cho việc in Disk3)    print24: # Vòng lặp in nội dung buffer disk3  lb a0, 0(s3) # Nạp byte từ buffer disk3 (s3) vào a0  li a7, 11  ecall  addi s8, s8, 1 # Tăng biến đếm s8  addi s3, s3, 1 # Tăng con trỏ buffer disk3 (s3)  bgt s8, a6, endisk2 # Nếu s8 > 3, nhảy đến 'endisk2'  j print24  nop  endisk2: # Kết thúc in block2  li a7, 4  la a0, msg3  ecall  li a7, 4  la a0, newline  ecall  beq t3, zero, exit1  j block3  nop |

* Giải thích block2:

addi s0, s0, 4: Con trỏ s0 được dịch 4 byte để bỏ qua phần dữ liệu đã được đọc vào disk2 ở block1. Bây giờ s0 trỏ đến đầu của segment 8 byte tiếp theo trong chuỗi đầu vào.

Vòng lặp b21 đến b22:

* Đọc 1 byte từ s0 vào disk1.
* Đọc 1 byte từ s0+4 vào disk3.
* Tính XOR, lưu vào array (đây sẽ là parity cho Disk2).
* t3 lại giảm 2 cho mỗi cặp byte, tổng cộng 8 cho stripe này.

In disk1 (dữ liệu), array (parity cho vị trí Disk2), disk3 (dữ liệu).

Con trỏ s0 lại dịch thêm 4 byte.

|  |
| --- |
| #--------------------------------  block3: # Funtion block3: Next 2 4-byte blocks are stored in disk2, disk3; parity is stored in disk1  la a2, array  la s2, disk2  la s3, disk3  addi s0, s0, 4 # Tăng con trỏ chuỗi đầu vào (s0) lên 4  addi t0, zero, 0 # Reset biến đếm vòng lặp t0 = 0  print31: # Print '[['  li a7, 4  la a0, msg4  ecall  b32: # Byte stored in Disk 2  lb t1, 0(s0) # Nạp byte từ chuỗi đầu vào (s0) vào t1 (cho Disk2)  addi t3, t3, -1 # string\_length --  sb t1, 0(s2) # Lưu byte t1 vào buffer disk2  b33: # Store in Disk 3  addi s5, s0, 4 # Tính địa chỉ byte tương ứng (s0+4)  lb t2, 0(s5) # Nạp byte từ (s0+4) vào t2 (cho Disk3)  addi t3, t3, -1 # Giảm độ dài còn lại t3  sb t2, 0(s3) # Lưu byte t2 vào buffer disk3    b31: # Store XOR result into disk1  xor a3, t1, t2 # Tính XOR của t1 và t2 -> a3 (parity)  sw a3, 0(a2) # Lưu byte parity a3 (như word) vào mảng 'array' (cho Disk1)  addi a2, a2, 4 # Tăng con trỏ mảng parity (a2)  addi t0, t0, 1 # Tăng biến đếm vòng lặp t0  addi s0, s0, 1 # Tăng con trỏ chuỗi đầu vào (s0)  addi s2, s2, 1 # Tăng con trỏ buffer disk2 (s2)  addi s3, s3, 1 # Tăng con trỏ buffer disk3 (s3)  bgt t0, a6, reset3 # Nếu t0 > 3, nhảy đến 'reset3'  j b32  nop  reset3: # Reset con trỏ buffer disk để chuẩn bị in  la s2, disk2  la s3, disk3  la a2, array  addi s9, zero, 0 # Index - Reset biến đếm s9 = 0 (cho việc in)    print32: # Vòng lặp in các byte parity (Disk1)  lb s8, 0(a2) # Nạp byte parity từ mảng (a2) vào s8  jal HEX  nop  li a7, 4  la a0, comma  ecall    addi s9, s9, 1 # Tăng biến đếm s9  addi a2, a2, 4 # Tăng con trỏ mảng parity (a2)  bgt s9, a5, next31 # Nếu s9 > 2, nhảy đến 'next31'  j print32  nop    next31: # In byte parity cuối cùng (Disk1)  lb s8, 0(a2) # Nạp byte parity cuối cùng vào s8  jal HEX  nop  li a7, 4  la a0, msg5  ecall  li a7, 4  la a0, msg2  ecall  addi s9, zero, 0    print33: # Vòng lặp in nội dung buffer disk2  lb a0, 0(s2) # Nạp byte từ buffer disk2 (s2) vào a0  li a7, 11 # syscall 11 (print char)  ecall  addi s9, s9, 1 # Tăng biến đếm s9  addi s2, s2, 1 # Tăng con trỏ buffer disk2 (s2)  bgt s9, a6, next32 # Nếu s9 > 3, nhảy đến 'next32'  j print33  nop    next32: # Chuẩn bị in Disk3  addi s9, zero, 0 # Reset biến đếm s9 (không thực sự dùng ngay sau)  addi s8, zero, 0 # Reset biến đếm s8 (cho việc in Disk3)  li a7, 4  la a0, msg3  ecall  li a7, 4  la a0, msg2  ecall  print34: # Vòng lặp in nội dung buffer disk3  lb a0, (s3) # Nạp byte từ buffer disk3 (s3) vào a0  li a7, 11 # syscall 11 (print char)  ecall  addi s8, s8, 1 # Tăng biến đếm s8  addi s3, s3, 1 # Tăng con trỏ buffer disk3 (s3)  bgt s8, a6, endisk3 # a6 is still 3 - Nếu s8 > 3, nhảy đến 'endisk3'  j print34  nop  endisk3: # Kết thúc in block3  li a7, 4  la a0, msg3  ecall    li a7, 4  la a0, newline  ecall  beq t3, zero, exit1 # Nếu độ dài còn lại t3 = 0, nhảy đến 'exit1'  j nextloop # Nhảy đến 'nextloop' để xử lý cụm 3 block tiếp theo  nop |

* Giải thích block3:

addi s0, s0, 4: Con trỏ s0 lại được dịch 4 byte, bỏ qua dữ liệu đã đọc vào disk3 ở block2. s0 trỏ đến đầu segment 8 byte thứ ba trong cụm 24 byte.

Vòng lặp b32 đến b31:

* Đọc 1 byte từ s0 vào disk2.
* Đọc 1 byte từ s0+4 vào disk3.
* Tính XOR, lưu vào array (parity cho Disk1).
* t3 giảm 8 cho stripe này.

In array (parity cho vị trí Disk1), disk2 (dữ liệu), disk3 (dữ liệu).

Con trỏ s0 lại dịch thêm 4 byte.

* + 1. Vòng lặp và Kết thúc (nextloop, exit1, ask, clear, exit)

|  |
| --- |
| #-----------End first 6 4-byte blocks----------------------------- (Thực ra là 3 block 8 byte = 24 byte)  #-----------Next 6 4-byte blocks---------------------------------- (Tương tự, là các cụm 24 byte tiếp theo)  nextloop: # Bắt đầu một chu kỳ mới của 3 block RAID  addi s0, s0, 4 # Tăng con trỏ chuỗi đầu vào (s0) lên 4 (để block1 tiếp theo bắt đầu từ đoạn 8 byte mới)  j block1  nop    exit1: # Print ------ and end RAID simulation  li a7, 4  la a0, msg1  ecall  j ask  nop    #--------------------END RAID 5 SIMULATION-------------------------  #--------------------TRY ANOTHER STRING----------------------------  ask: # Hỏi người dùng có muốn thử chuỗi khác không  li a7, 50 # syscall 50 (message dialog yes/no/cancel)  la a0, message  ecall  beq a0, zero, clear # a0: 0 = YES; 1 = NO; 2 = CANCEL - Nếu a0 = 0 (Yes), nhảy đến 'clear'  nop  j exit # Nếu không phải Yes (No hoặc Cancel), nhảy đến 'exit'  nop    # clear function: Return string to original state  clear: # Xóa nội dung chuỗi đầu vào cũ để chuẩn bị nhập mới  la s0, string # Nạp địa chỉ của buffer 'string' vào s0  add s3, s0, t5 # Tính địa chỉ cuối của chuỗi cũ (string + length\_old\_string)  li t1, 0 # Set t1 = 0  goAgain: # Return string to empty state to start again  sb t1, (s0) # Ghi byte null (t1) vào địa chỉ s0 trong buffer 'string'  nop  addi s0, s0, 1 # Tăng con trỏ s0 lên byte tiếp theo  bge s0, s3, input # Nếu s0 >= s3 (đã xóa hết), nhảy đến 'input'  nop  j goAgain  nop  #-----Exit program----------  exit:  li a7, 10  ecall |

* Giải thích:

nextloop: Sau khi hoàn thành 3 block (tức là 3 stripes, xử lý 24 byte dữ liệu đầu vào), s0 được dịch thêm 4 byte. Điều này là để s0 trỏ đến điểm bắt đầu của dữ liệu cho disk1 trong block1 của chu kỳ tiếp theo. Chu trình 3-block (block1, block2, block3) lặp lại.

exit1: Khi t3 (độ dài còn lại) bằng 0, in dòng gạch ngang cuối cùng và nhảy đến ask.

ask: Sử dụng syscall 50 để hiển thị hộp thoại Yes/No. Nếu Yes (a0=0), nhảy đến clear. Ngược lại, thoát.

clear và goAgain: Xóa nội dung của string bằng cách ghi các byte null vào đó, dựa trên độ dài t5 đã lưu trước đó. Sau đó quay lại input.

exit: Kết thúc chương trình.

## Chạy thử mã lệnh và kết quả:

### Kết quả khi chạy chương trình với đầu vào “*DCE.\*\*\*\*ABCD1234HUSTHUST*”:

A screenshot of a computer

AI-generated content may be incorrect.

### Kết quả khi chạy với đầu vào rỗng:

A screenshot of a computer

AI-generated content may be incorrect.

### Kết quả khi chạy đầu vào ABCXYZ:

A screenshot of a computer

AI-generated content may be incorrect.

### Kết quả khi chạy với đầu vào HUSTABCD:

A screen shot of a computer

AI-generated content may be incorrect.

### Kết quả khi chạy với đầu vào ABCDXYZT1234567890125487DCE.\*\*\*\*:

A screenshot of a computer program

AI-generated content may be incorrect.

# Assignment 9: Kiểm thử các thuật toán sắp xếp

## **Mô tả bài toán:**

- Tìm hiểu hàm hệ thống để đọc, ghi file văn bản.

- Cho trước file văn bản chứa các số nguyên ngẫu nhiên, phân cách bởi dấu cách. Số lượng phần tử có thể lớn, tối đa 10000 phần tử.

- Tạo giao diện cho phép người dùng nhập tên file để mở, các số trong file được đọc và lưu vào bộ nhớ.

- Người dùng chọn thuật toán sắp xếp cần thực hiện (Nổi bọt, Chèn, Lựa chọn). Được điểm cộng nếu thực hiện thêm các thuật toán khác.

- Chương trình chạy thuật toán và in ra thời gian thực hiện.

- Chương trình ghi kết quả sắp xếp vào file kết quả.

## **Thuật toán sử dụng:**

### 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.

### 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.

### Các 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).

### 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.

### 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).

### 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.

### Xử lý số âm (Bitmask – flag\_negative\_number, 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.

## **Đoạn mã lệnh và giải thích:**

### Khai báo dữ liệu(.data)

|  |
| --- |
| .data  # Phan du lieu hien co  numbers: .space 80000 # Vùng nhớ 80000 byte để lưu trữ các số đã đọc từ file  input\_buffer\_size: .word 80000 # Kích thước của buffer đọc file  count: .word 0 # Biến đếm số lượng các số nguyên đã đọc được    # Them mang bitmask cho so am (1 bit moi so)  # Voi 80000 byte so (20000 so nguyen), chung ta can 20000 bit = 2500 byte  neg\_bitmask: .space 2500 # Mảng bitmask để đánh dấu các số âm, mỗi bit tương ứng một số    input\_filename: .space 256  file\_read\_buffer: .space 1024 # Buffer tạm để đọc từng phần của file  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 # Biến lưu trữ file descriptor của file input  newline: .asciz "\n"  space: .string " "  start\_time: .word 0  end\_time: .word 0    msg\_execution\_time: .asciz "\nExecution time (ms): "    # Du lieu moi cho file output  output\_filename: .asciz "C:\\RISCV\\output5.txt"  out\_fd: .word 0 # Biến lưu trữ file descriptor của file output  buffer\_number: .space 12 # Buffer tạm để chuyển đổi số sang chuỗi khi ghi file  msg\_file\_error\_open: .asciz "\nError writing to output file\n"  char\_minus: .asciz "-" |

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.

### Hàm main và vòng lặp menu:

|  |
| --- |
| .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  la t0, input\_filename  remove\_newline\_from\_filename:  # Kiem tra tung ky tu de tim newline  lb t1, 0(t0)  beqz t1, open\_input\_file  li t2, 10 # 10 = "\n"  beq t1, t2, replace\_null  addi t0, t0, 1  j remove\_newline\_from\_filename  replace\_null:  # Thay the newline bang null terminator  sb zero, 0(t0)    open\_input\_file:  # Mo file input  li a7, 1024  la a0, input\_filename  li a1, 0  ecall  # Kiem tra loi mo file  bltz a0, file\_error\_open  # Luu file descriptor  la t1, fd  sw a0, 0(t1)  # Goi ham doc so tu file  jal read\_numbers    menu\_loop:  # Hien thi menu  li a7, 4  la a0, menu  ecall    # Doc lua chon cua nguoi dung  li a7, 5  ecall    # Xu ly lua chon  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  # Lua chon khong hop le, thoat  j exit  file\_error\_open:  # In ra thong bao loi mo file  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.

### Hàm read\_numbers (Đọc và phân tích cú pháp số từ File)

|  |
| --- |
| read\_numbers:  # Luu thanh ghi vao stack  addi sp, sp, -16  sw ra, 12(sp)  sw s0, 8(sp)  sw s1, 4(sp)  sw s2, 0(sp)  # Reset count ve 0  la t1, count  sw zero, 0(t1)  # Khoi tao bien tam de parse so  li t0, 0 # So hien tai dang duoc parse  li t1, 0 # Co bao hieu dang trong mot so (1=true)  li t6, 0 # Co bao hieu so am (1=am)  read\_loop:  # Doc mot ky tu tu file  li a7, 63 # Syscall: 63 - Read  lw a0, fd  la a1, file\_read\_buffer  li a2, 1  ecall  # Kiem tra cuoi file  beqz a0, read\_done    # Tai ky tu da doc  lb t2, 0(a1)    # Kiem tra dau tru '-'  li t3, 45  bne t2, t3, not\_char\_minus  beqz t1, set\_negative  j read\_loop    set\_negative:  # Dat co so am va co dang trong so  li t6, 1  li t1, 1  j read\_loop    not\_char\_minus:  # Kiem tra ky tu phan cach (space hoac newline)  li t3, 32  beq t2, t3, save\_number  li t3, 10  beq t2, t3, save\_number    # Chuyen doi ky tu ASCII so sang gia tri so  addi t2, t2, -48  li t3, 10  mul t0, t0, t3  add t0, t0, t2  li t1, 1 # Dat co dang trong so  j read\_loop    save\_number:  # Chi luu neu dang trong mot so  beqz t1, read\_loop    # Ap dung dau (neu la so am)  beqz t6, save\_positive  neg t0, t0    save\_positive:  # Luu so vao mang numbers  la t3, count  lw t3, 0(t3)  slli t4, t3, 2  la t5, numbers  add t5, t5, t4  sw t0, 0(t5)    # Tang bien dem count  addi t3, t3, 1  la t4, count  sw t3, 0(t4)    # 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 dang parse do khi het file  beqz t1, close\_file    # Ap dung dau cho so cuoi  beqz t6, save\_last\_positive  neg t0, t0  save\_last\_positive:  # Luu so cuoi vao mang  la t3, count  lw t3, 0(t3)  slli t4, t3, 2  la t5, numbers  add t5, t5, t4  sw t0, 0(t5)  # Tang count cho so cuoi  addi t3, t3, 1  la t4, count  sw t3, 0(t4)  close\_file:  # Dong file input  li a7, 57  lw a0, fd  ecall    # Khoi phuc thanh ghi tu stack  lw ra, 12(sp)  lw s0, 8(sp)  lw s1, 4(sp)  lw s2, 0(sp)  addi sp, sp, 16  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.

### 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.

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

|  |
| --- |
| quick\_sort\_array:  # Lay thoi gian bat dau  jal get\_time  sw a0, start\_time, t0 # t0 la thanh ghi tam, khong can thiet    # Goi Quick Sort  la a0, numbers  li a1, 0  lw a2, count  addi a2, a2, -1  jal quick\_sort\_logic    # (Logic danh dau so am co the can xem lai sau khi sap xep gia tri tuyet doi)  la a0, numbers  lw a1, count  jal flag\_negative\_numbers    # Lay thoi gian ket thuc va in  jal get\_time  sw a0, end\_time, t0  jal print\_time    # Ghi ket qua ra file  jal write\_results  j menu\_loop # Quay lai menu chinh |

Giải thích: Thiết lập tham số và gọi quick\_sort\_logic. Đo thời gian, in thời gian, ghi kết quả.

|  |
| --- |
| quick\_sort\_logic:  # Luu thanh ghi vao stack  addi sp, sp, -24  sw ra, 20(sp)  sw s0, 16(sp) # array\_base  sw s1, 12(sp) # left\_index  sw s2, 8(sp) # right\_index  sw s3, 4(sp) # pivot\_index  sw s4, 0(sp) # unused    # Luu cac tham so vao thanh ghi s  mv s0, a0  mv s1, a1  mv s2, a2    # Dieu kien dung de quy  bge s1, s2, quick\_sort\_end    # Goi ham phan hoach (partition)  mv a0, s0  mv a1, s1  mv a2, s2  jal partition\_elements  mv s3, a0 # Luu chi so pivot    # De quy Quick Sort cho phan ben trai pivot  mv a0, s0  mv a1, s1  addi a2, s3, -1  jal quick\_sort\_logic    # De quy Quick Sort cho phan ben phai pivot  mv a0, s0  addi a1, s3, 1  mv a2, s2  jal quick\_sort\_logic    quick\_sort\_end:  # Khoi phuc thanh ghi tu stack  lw ra, 20(sp)  lw s0, 16(sp)  lw s1, 12(sp)  lw s2, 8(sp)  lw s3, 4(sp)  lw s4, 0(sp)  addi sp, sp, 24  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:  # Luu thanh ghi  addi sp, sp, -24  sw ra, 20(sp)  sw s0, 16(sp) # array\_base  sw s1, 12(sp) # left  sw s2, 8(sp) # right  sw s3, 4(sp) # pivot\_value  sw s4, 0(sp) # i (index of smaller element)    # Luu tham so  mv s0, a0  mv s1, a1  mv s2, a2  # Chon pivot la phan tu cuoi cung arr[right]  slli t0, s2, 2  add t0, s0, t0  lw s3, 0(t0) # s3 = pivot\_value    # Khoi tao i = left - 1  addi s4, s1, -1  # Khoi tao j = left  mv t1, s1    partition\_loop\_elements:  # Vong lap j tu left den right-1  bge t1, s2, partition\_elements\_done    # Tai phan tu arr[j]  slli t0, t1, 2  add t0, s0, t0  lw t2, 0(t0) # t2 = arr[j]    # Neu arr[j] <= pivot\_value  bgt t2, s3, skip\_swap    # Tang i  addi s4, s4, 1    # Hoan doi arr[i] va arr[j]  slli t0, s4, 2 # dia chi arr[i]  add t0, s0, t0  slli t3, t1, 2 # dia chi arr[j]  add t3, s0, t3    lw t4, 0(t0) # temp = arr[i]  lw t5, 0(t3) # arr[j]  sw t5, 0(t0) # arr[i] = arr[j]  sw t4, 0(t3) # arr[j] = temp    skip\_swap:  # Tang j  addi t1, t1, 1  j partition\_loop\_elements    partition\_elements\_done:  # Hoan doi arr[i+1] voi arr[right] (pivot)  addi s4, s4, 1 # i+1    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)  sw t3, 0(t0) # arr[i+1] = pivot\_value  sw t2, 0(t1) # arr[right] = temp    # Tra ve chi so cua pivot (i+1)  mv a0, s4  # Khoi phuc thanh ghi  lw ra, 12(sp)  lw s0, 8(sp)  lw s1, 4(sp)  lw s2, 0(sp)  addi sp, sp, 16  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.

### Bubble Sort (bubble\_sort\_array, bubble\_sort\_core)

|  |
| --- |
| 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\_arrayimpl)

|  |
| --- |
| 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.

### Selecttion Sort (selection\_sort\_array, selection\_sort\_array\_impl)

|  |
| --- |
| 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]).

### Hàm flag\_negative\_numbers

|  |
| --- |
| flag\_negative\_numbers:  # Luu thanh ghi  addi sp, sp, -16  sw ra, 12(sp)  sw s0, 8(sp)  sw s1, 4(sp)  sw s2, 0(sp)    # Khoi tao s0 (dia chi mang), s1 (size), s2 (index)  mv s0, a0  mv s1, a1  li s2, 0    flag\_loop:  # Kiem tra ket thuc vong lap  bge s2, s1, flag\_done    # Tai so hien tai numbers[s2]  slli t0, s2, 2  add t0, s0, t0  lw t1, 0(t0)    # Neu so duong, bo qua  bgez t1, skip\_flag    # Tinh toan offset byte va vi tri bit trong bitmask  mv t0, s2  srai t1, t0, 3 # byte\_offset = index / 8  andi t2, t0, 0x7 # bit\_position = index % 8  li t3, 1  sll t3, t3, t2 # tao bit mask: 1 << bit\_position    # Dat bit trong neg\_bitmask  la t4, neg\_bitmask  add t4, t4, t1 # dia chi byte trong bitmask  lb t5, 0(t4) # load byte hien tai  or t5, t5, t3 # set bit  sb t5, 0(t4) # luu lai byte    skip\_flag:  # Tang index  addi s2, s2, 1  j flag\_loop    flag\_done:  # Khoi phuc thanh ghi  lw ra, 12(sp)  lw s0, 8(sp)  lw s1, 4(sp)  lw s2, 0(sp)  addi sp, sp, 16  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.

### Hàm get\_time và print\_time

|  |
| --- |
| get\_time:  # Syscall lay thoi gian  li a7, 30  ecall  ret  print\_time:  # Tinh toan thoi gian thuc thi  la t0, start\_time  lw t1, 0(t0)  la t0, end\_time  lw t2, 0(t0)  sub t3, t2, t1 # execution\_time = end - start    # In thong bao  li a7, 4  la a0, msg\_execution\_time  ecall    # In gia tri thoi gian  li a7, 1  mv a0, t3  ecall  ret |

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

### Hàm number\_to\_string và str\_reverse

|  |
| --- |
| number\_to\_string:  # Luu thanh ghi  addi sp, sp, -24  sw ra, 20(sp)  sw s0, 16(sp) # buffer\_address  sw s1, 12(sp) # number\_to\_convert  sw s2, 8(sp) # string\_length  sw s3, 4(sp) # negative\_flag  sw s4, 0(sp) # temp\_pointer\_in\_buffer    # Khoi tao  mv s0, a0  mv s1, a1  li s2, 0  li s3, 0    # 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:  # Kiem tra so am  bgez s1, convert\_digits  li s3, 1 # set negative\_flag  neg s1, s1 # make number positive    convert\_digits:  # Chuyen doi cac chu so (luu nguoc vao buffer)  mv s4, s0 # s4 la con tro tam trong buffer  digit\_loop:  beqz s1, finalize\_string # Neu so = 0, da xong phan chu so  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) # store digit in buffer  addi s4, s4, 1 # advance buffer pointer  addi s2, s2, 1 # increment length  j digit\_loop    finalize\_string:  # Them dau '-' neu am  beqz s3, reverse\_string  li t1, 45 # '-'  sb t1, 0(s4)  addi s4, s4, 1  addi s2, s2, 1  reverse\_string:  # Dao nguoc chuoi trong buffer  mv a0, s0 # start\_address  addi a1, s4, -1 # end\_address (s4 dang o sau ky tu cuoi)  jal str\_reverse    # Tra ve do dai  mv a0, s2    num\_to\_str\_done:  # Khoi phuc thanh ghi  lw ra, 20(sp)  lw s0, 16(sp)  lw s1, 12(sp)  lw s2, 8(sp)  lw s3, 4(sp)  lw s4, 0(sp)  addi sp, sp, 24  ret  str\_reverse:  # Kiem tra dieu kien dung (start\_ptr >= end\_ptr)  bge a0, a1, str\_rev\_done    # Hoan doi ky tu  lb t0, 0(a0)  lb t1, 0(a1)  sb t1, 0(a0)  sb t0, 0(a1)    # Di chuyen con tro  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.

### Hàm write\_results (Ghi kết quả đã sắp xếp ra File )

|  |
| --- |
| write\_results:  # Luu thanh ghi  addi sp, sp, -16  sw ra, 12(sp)  sw s0, 8(sp) # i (loop counter)  sw s1, 4(sp) # count (total numbers)  sw s2, 0(sp) # numbers\_array\_base    # Mo file output  li a7, 1024  la a0, output\_filename  li a1, 1 # Write-only, create, truncate  li a2, 0x1ff # Permissions rwxrwxrwx  ecall    # Kiem tra loi mo file  bltz a0, msg\_file\_error\_openor  sw a0, out\_fd, t0 # Luu output file descriptor    # Khoi tao vong lap ghi file  li s0, 0 # i = 0  lw s1, count  la s2, numbers    write\_loop:  # Kiem tra ket thuc vong lap  bge s0, s1, write\_done    # Tai so hien tai numbers[i]  slli t0, s0, 2  add t1, s2, t0  lw t2, 0(t1)    # Chuyen so sang chuoi  la a0, buffer\_number  mv a1, t2  jal number\_to\_string  mv t3, a0 # t3 = length of stringified number    # Ghi chuoi so vao file  li a7, 64 # WriteFile syscall  lw a0, out\_fd  la a1, buffer\_number  mv a2, t3  ecall    # Ghi dau cach (neu khong phai so cuoi cung)  addi t0, s1, -1 # last\_index = count - 1  bge s0, t0, skip\_space    li a7, 64  lw a0, out\_fd  la a1, space  li a2, 1  ecall    skip\_space:  # Tang bien dem i  addi s0, s0, 1  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  li a7, 57 # CloseFile syscall  lw a0, out\_fd  ecall    # Khoi phuc thanh ghi  lw ra, 12(sp)  lw s0, 8(sp)  lw s1, 4(sp)  lw s2, 0(sp)  addi sp, sp, 16  ret  msg\_file\_error\_openor:  # In thong bao loi ghi file  li a7, 4  la a0, msg\_file\_error\_open  ecall    # Khoi phuc thanh ghi (neu can)  lw ra, 12(sp)  lw s0, 8(sp)  lw s1, 4(sp)  lw s2, 0(sp)  addi sp, sp, 16  ret |

**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.

## **Kết quả chạy mẫu:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.