

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



SOICT

BÁO CÁO CUỐI KỲ
THỰC HÀNH KIẾN TRÚC MÁY TÍNH

NHÓM 14

Giảng viên hướng dẫn

Ngô Lam Trung

Trợ Giảng

Phạm Quốc Minh

Thành viên nhóm

Phan Khánh Vũ

Hoàng Văn Thắng

Mục lục

1. Assignment 6: Mô phỏng ổ đĩa RAID 5	3
1.1. Mô tả bài toán:.....	3
1.2. Thuật toán sử dụng:	4
1.2.1. Khởi tạo.....	4
1.2.2. Nhập liệu và kiểm tra độ dài :	4
1.2.3. Xử lý và phân phối dữ liệu (RAID 5 Logic):.....	4
1.2.4. Thủ tục HEX (Chuyển đổi sang Hexa):.....	5
1.3. Đoạn mã lệnh và giải thích	6
1.3.1. Khai báo dữ liệu (.data).....	6
1.3.2. Main và input	6
1.3.3. Kiểm tra độ dài chuỗi (length, check_char, test_length, error)	7
1.3.4. Thủ tục HEX (Chuyển byte sang 2 ký tự Hexa ASCII).....	9
1.3.5. Mô phỏng RAID 5 (block1, block2, block3)	10
1.4. Chạy thử mã lệnh và kết quả:.....	24
1.4.1. Kết quả khi chạy chương trình với đầu vào “DCE.***ABCD1234HUSTHUST”.....	24
1.4.2. Kết quả khi chạy với đầu vào rỗng:.....	24
1.4.3. Kết quả khi chạy đầu vào ABCXYZ:.....	24
1.4.4. Kết quả khi chạy với đầu vào HUSTABCD:	25
1.4.5. Kết quả khi chạy với đầu vào ABCDXYZT1234567890125487DCE.***:	25
2. Assignment 9: Kiểm thử các thuật toán sắp xếp	26
2.1. Mô tả bài toán:.....	26
2.2. Thuật toán sử dụng:	26
2.2.1. Luồng điều khiển chính (main):	26
2.2.2. Đọc và phân tích cú pháp số (read_numbers):.....	26
2.2.3. Các thuật toán sắp xếp:	27
2.2.4. Đo thời gian (get_time, print_time):	27
2.2.5. Chuyển đổi số sang Chuỗi (number_to_string, str_reverse):.....	27
2.2.6. Ghi File kết quả (write_results):.....	27
2.2.7. Xử lý số âm (Bitmask – flag_negative_number, neg_bitmask):	28
2.3. Đoạn mã lệnh và giải thích:	28
2.3.1. Khai báo dữ liệu(.data)	28
2.3.2. Hàm main và vòng lặp menu:	29
2.3.3. Hàm read_numbers (Đọc và phân tích cú pháp số từ File)	32
2.3.4. Các hàm sắp xếp	36
2.3.4.1. Quick Sort (quick_sort_array, quick_sort_logic, partition_elements).....	36
2.3.4.2. Bubble Sort (bubble_sort_array, bubble_sort_core)	36

2.3.4.3.	Insertion Sort (insertion_sort_array, insertion_sort_arrayimpl).....	38
2.3.4.4.	Selection Sort (selection_sort_array, selection_sort_array_impl).....	39
2.3.5.	Hàm flag_negative_numbers.....	40
2.3.6.	Hàm get_time và print_time	42
2.3.7.	Hàm number_to_string và str_reverse.....	43
2.3.8.	Hàm write_results (Ghi kết quả đã sắp xếp ra File).....	46
2.4.	Kết quả chạy mẫu:	49

1. Assignment 6: Mô phỏng ổ đĩa RAID 5

1.1. 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 $6e = 'D' \text{ xor } '*'$; $69 = 'C' \text{ xor } '*'$; $6f = 'E' \text{ xor } '*'$; $04 = '.' \text{ xor } '*'$

```

Nhập chuỗi kí tự : DCE.***ABCD1234HUSTHUST
      Disk 1          Disk 2          Disk 3
      -----          -----          -----
|      DCE.      |      |      ****      |      | [ 6e, 69, 6f, 04] |
|      ABCD      |      | [ 70, 70, 70, 70] |      | 1234      |
|[ 00, 00, 00, 00]|      |      HUST      |      | HUST      |
      -----          -----          -----

```

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
...

1.2. Thuật toán sử dụng:

1.2.1. 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 thành ghi ban đầu cho các vùng nhớ đĩa và mảng parity.

1.2.2. 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).
 - o 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.
 - o Nếu độ dài hợp lệ, tiếp tục.

1.2.3. 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**
 - o 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 trở đến cụm 8 byte tiếp theo cho block1 của chu kỳ mới (nếu còn dữ liệu).

1.2.4. 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.

1.2.5. 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.

1.3. Đoạn mã lệnh và giải thích

1.3.1. 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.

1.3.2. 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.

1.3.3. 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

```



```

        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.

1.3.5. 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 into 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 printing 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 trở 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.

1.3.6. 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 trở 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.

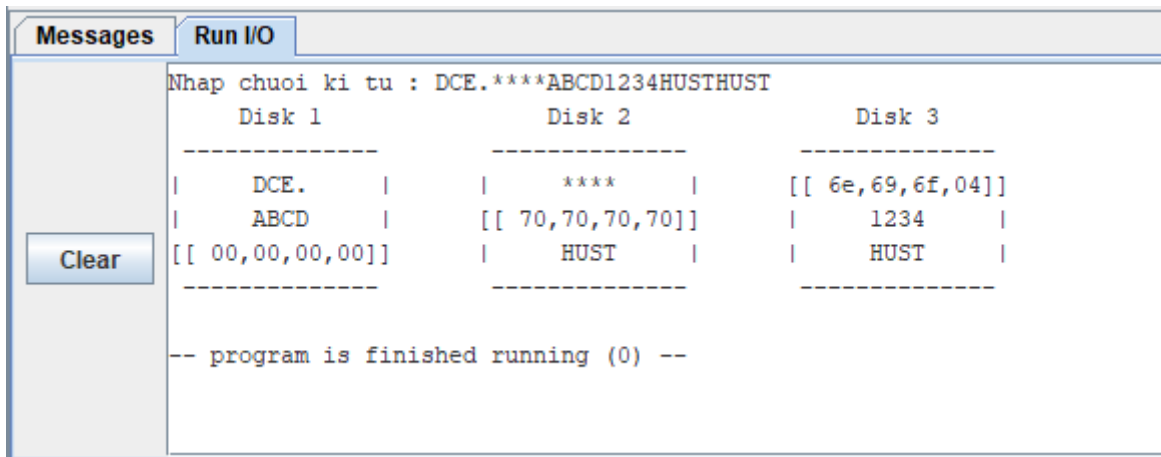
ask: Sử dụng syscall 50 để hiển thị hộp thoại Yes/No. Nếu Yes (a0=0), nhảy đến clear. Ngược lại, thoát.

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.

1.4. Chạy thử mã lệnh và kết quả:

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

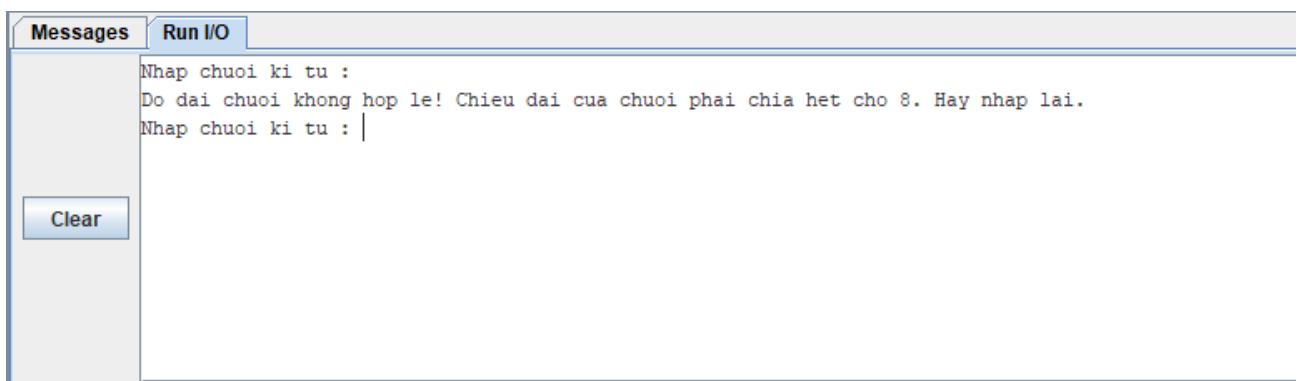


The screenshot shows a window with two tabs: "Messages" and "Run I/O". The "Run I/O" tab is active, displaying the following text:

```
Nhap chuoi ki tu : DCE.***ABCD1234HUSTHUST
      Disk 1          Disk 2          Disk 3
      -----
      | DCE.  |      |  ***  |      | [[ 6e,69,6f,04]]
      | ABCD  |      | [[ 70,70,70,70]] |      | 1234  |
      | [[ 00,00,00,00]] |      | HUST  |      | HUST  |
      -----
-- program is finished running (0) --
```

On the left side of the window, there is a "Clear" button.

1.4.2. Kết quả khi chạy với đầu vào rỗng:

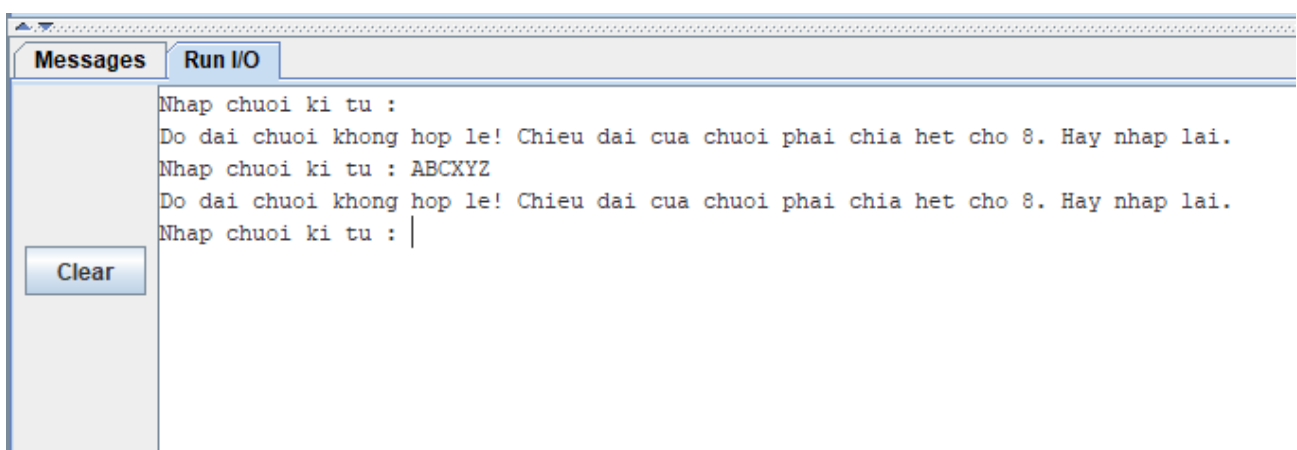


The screenshot shows a window with two tabs: "Messages" and "Run I/O". The "Run I/O" tab is active, displaying the following text:

```
Nhap chuoi ki tu :
Do dai chuoi khong hop le! Chieu dai cua chuoi phai chia het cho 8. Hay nhap lai.
Nhap chuoi ki tu : |
```

On the left side of the window, there is a "Clear" button.

1.4.3. Kết quả khi chạy đầu vào ABCXYZ:



The screenshot shows a window with two tabs: "Messages" and "Run I/O". The "Run I/O" tab is active, displaying the following text:

```
Nhap chuoi ki tu :
Do dai chuoi khong hop le! Chieu dai cua chuoi phai chia het cho 8. Hay nhap lai.
Nhap chuoi ki tu : ABCXYZ
Do dai chuoi khong hop le! Chieu dai cua chuoi phai chia het cho 8. Hay nhap lai.
Nhap chuoi ki tu : |
```

On the left side of the window, there is a "Clear" button.

1.4.4. Kết quả khi chạy với đầu vào HUSTABCD:

Messages	Run I/O												
<div>Clear</div>	<p>Do dai chuoai khong hop le! Chieu dai cua chuoai phai chia het cho 8. Hay nhap lai.</p> <p>Nhap chuoai ki tu : ABCXYZ</p> <p>Do dai chuoai khong hop le! Chieu dai cua chuoai phai chia het cho 8. Hay nhap lai.</p> <p>Nhap chuoai ki tu : HUSTABCD</p> <table> <thead> <tr> <th>Disk 1</th> <th>Disk 2</th> <th>Disk 3</th> </tr> </thead> <tbody> <tr> <td>-----</td> <td>-----</td> <td>-----</td> </tr> <tr> <td> HUST </td> <td> ABCD </td> <td> [[09,17,10,10]] </td> </tr> <tr> <td>-----</td> <td>-----</td> <td>-----</td> </tr> </tbody> </table> <p>-- program is finished running (0) --</p>	Disk 1	Disk 2	Disk 3	-----	-----	-----	HUST	ABCD	[[09,17,10,10]]	-----	-----	-----
	Disk 1	Disk 2	Disk 3										
	-----	-----	-----										
	HUST	ABCD	[[09,17,10,10]]										
	-----	-----	-----										

1.4.5. Kết quả khi chạy với đầu vào
ABCDXYZT1234567890125487DCE.****:

Messages	Run I/O																					
<div>Clear</div>	<p>Nhap chuoai ki tu : ABCDXYZT1234567890125487DCE.****</p> <table> <thead> <tr> <th>Disk 1</th> <th>Disk 2</th> <th>Disk 3</th> </tr> </thead> <tbody> <tr> <td>-----</td> <td>-----</td> <td>-----</td> </tr> <tr> <td> ABCD </td> <td> XYZT </td> <td> [[19,1b,19,10]] </td> </tr> <tr> <td> 1234 </td> <td> [[04,04,04,0c]] </td> <td> 5678 </td> </tr> <tr> <td> [[0c,04,09,05]] </td> <td> 9012 </td> <td> 5487 </td> </tr> <tr> <td> DCE. </td> <td> **** </td> <td> [[6e,69,6f,04]] </td> </tr> <tr> <td>-----</td> <td>-----</td> <td>-----</td> </tr> </tbody> </table> <p>-- program is finished running (0) --</p>	Disk 1	Disk 2	Disk 3	-----	-----	-----	ABCD	XYZT	[[19,1b,19,10]]	1234	[[04,04,04,0c]]	5678	[[0c,04,09,05]]	9012	5487	DCE.	****	[[6e,69,6f,04]]	-----	-----	-----
	Disk 1	Disk 2	Disk 3																			
	-----	-----	-----																			
	ABCD	XYZT	[[19,1b,19,10]]																			
	1234	[[04,04,04,0c]]	5678																			
[[0c,04,09,05]]	9012	5487																				
DCE.	****	[[6e,69,6f,04]]																				
-----	-----	-----																				

2. Assignment 9: Kiểm thử các thuật toán sắp xếp

2.1. 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.

2.2. Thuật toán sử dụng:

2.2.1. 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.

2.2.2. 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.

2.2.3. Các thuật toán sắp xếp:

- *Bubble Sort* (bubble_sort_core): So sánh các cặp phần tử liên 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).

2.2.4. 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.

2.2.5. 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).

2.2.6. 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.

2.2.7. 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.

2.3. Đoạn mã lệnh và giải thích:

2.3.1. 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.

2.3.2. 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: Hiện thị menu, đọc lựa chọn của người dùng (syscall 5), và nhảy đến hàm sắp xếp tương ứng hoặc thoát.

2.3.3. Hàm read_numbers (Đọc và phân tích cú pháp số từ File)

read_numbers:

Lưu thanh ghi vào stack

addi sp, sp, -16

sw ra, 12(sp)

sw s0, 8(sp)

sw s1, 4(sp)

sw s2, 0(sp)

Reset count về 0

la t1, count

sw zero, 0(t1)

Khởi tạo biến tạm để parse số

li t0, 0

Số hiện tại đang được parse

li t1, 0

Có bao nhiêu chữ số trong một số (1=true)

li t6, 0

Có bao nhiêu số âm (1=am)

read_loop:

Đọc một ký tự từ 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 tuphan 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`.

2.3.4. 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`.

2.3.4.1. Quick Sort (`quick_sort_array`, `quick_sort_logic`, `partition_elements`)

2.3.4.2. Bubble Sort (`bubble_sort_array`, `bubble_sort_core`)

```

bubble_sort_array:
    # ... (lấy thời gian bắt đầu) ...

    # Gọi 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:

```

```

# Lưu thành ghi (ra, s0: array_base, s1: size, s2: i)

# ...

# Khởi tạo

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:
    # Vòng lặp ngoài: 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:
    # Vòng lặp 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 sánh arr[j] và arr[j+1]

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

    # Nếu arr[j] <= arr[j+1], không đổi cho

    ble    t3, t4, no_swap_bubble_sort

    # Hoán đổi arr[j] và arr[j+1]

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

no_swap_bubble_sort:
    # Tăng j

    addi    t0, t0, 1

    j      inner_loop_bubble_sort

inner_done_bubble_sort:
    # Tăng 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.

2.3.4.3. 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:
    # Lưu thành ghi (ra, s0: array_base, s1: size, s2: i)
    # ...
    # Khởi tạo
    mv    s0, a0
    mv    s1, a1
    li    s2, 1          # i = 1 (bắt đầu từ phần tử thứ hai)

outer_loop_insertion:
    # Vòng lặp ngoài: i from 1 to size-1
    bge    s2, s1, insertion_done # if i >= size, done
    # Lấy key = arr[i]
    # ... (tính địa chỉ arr[i], load giá trị vào t1 = key) ...
    # Khởi tạo j = i-1
    addi    t2, s2, -1      # t2 = j

inner_loop_insertion:
    # Vòng lặp trong: j from i-1 down to 0 AND arr[j] > key
    bltz    t2, inner_done_insertion      # Nếu j < 0, done inner loop
    # So sánh arr[j] với key
    # ... (tính địa chỉ arr[j], load giá trị vào t4 = arr[j]) ...
    ble     t4, t1, inner_done_insertion   # Nếu arr[j] <= key, done inner loop (tìm được vị trí chèn)
    # Dịch chuyển arr[j] sang arr[j+1]
    # ... (lưu t4 (arr[j]) vào vị trí arr[j+1]) ...
    # Giảm 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.

2.3.4.4. Selection 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:

# Lưu thanh ghi (ra, s0: array_base, s1: size, s2: i)

# ...

# Khởi tạo
mv      s0, a0
mv      s1, a1
li      s2, 0          # i = 0

outer_loop_selection:

# Vòng lặp ngoài: i from 0 to size-2
# ... (tính giới hạn cho i là size-1) ...

bge     s2, t0, selection_done # if i >= size-1, done

# Khởi tạo min_idx = i
mv      t1, s2          # t1 = min_idx

# Khởi tạo j = i+1
addi    t2, s2, 1        # t2 = j

inner_loop_selection:

# Vòng lặp 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]).

2.3.5. Hàm flag_negative_numbers

```

flag_negative_numbers:

# Lưu thanh ghi

addi    sp, sp, -16

sw      ra, 12(sp)

sw      s0, 8(sp)

sw      s1, 4(sp)

sw      s2, 0(sp)

# Khởi tạo s0 (địa chỉ mảng), 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.

2.3.6. 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.

2.3.7. Hàm `number_to_string` và `str_reverse`

```

number_to_string:

# Lưu 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


# Khởi tạo

mv      s0, a0

mv      s1, a1

li      s2, 0

li      s3, 0


# Xử lý số 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.

2.3.8. Hàm `write_results` (Ghi kết quả đã sắp xếp ra File)

```
write_results:
```

```
    # Lưu 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
```

```
    # Mở file output
```

```
    li      a7, 1024
```

```
    la      a0, output_filename
```

```
    li      a1, 1               # Write-only, create, truncate
```

```
    li      a2, 0x1ff           # Permissions rwxrwxrwx
```

```
    ecall
```

```
    # Kiểm tra lỗi mở file
```

```
    bltz    a0, msg_file_error_openor
```

```
    sw      a0, out_fd, t0      # Lưu output file descriptor
```

```
    # Khởi tạo vòng lặp 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)

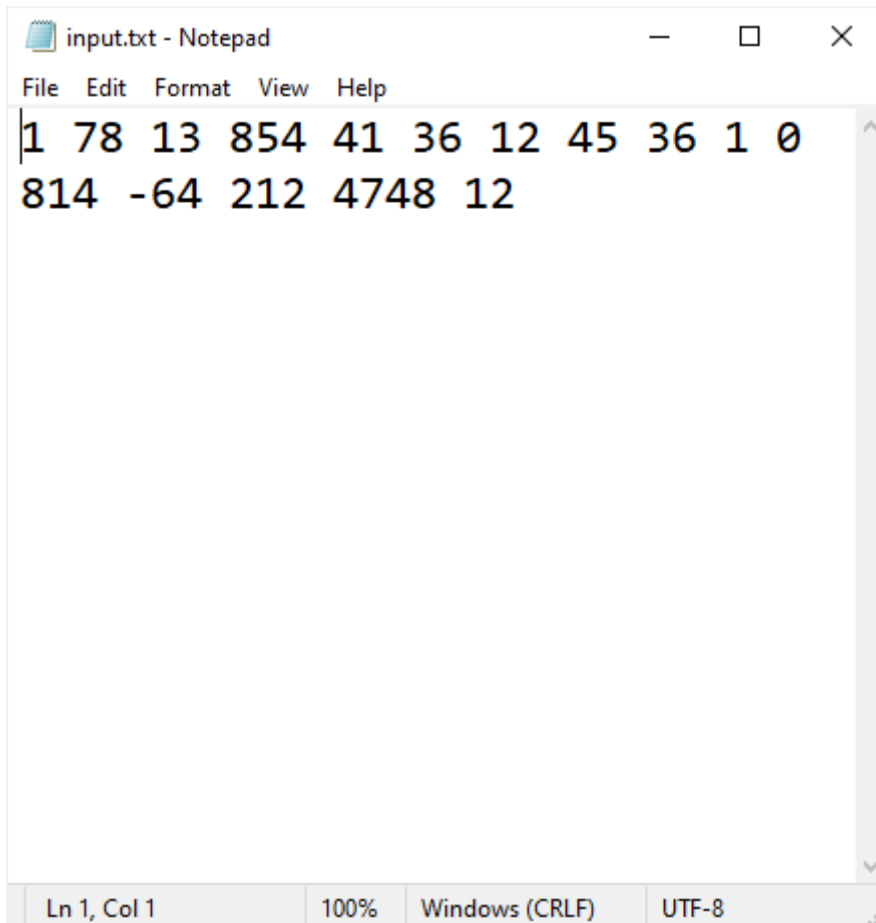
addi  sp, sp, 16

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.

2.4. Kết quả chạy mẫu:

Với đầu vào là file input.txt:



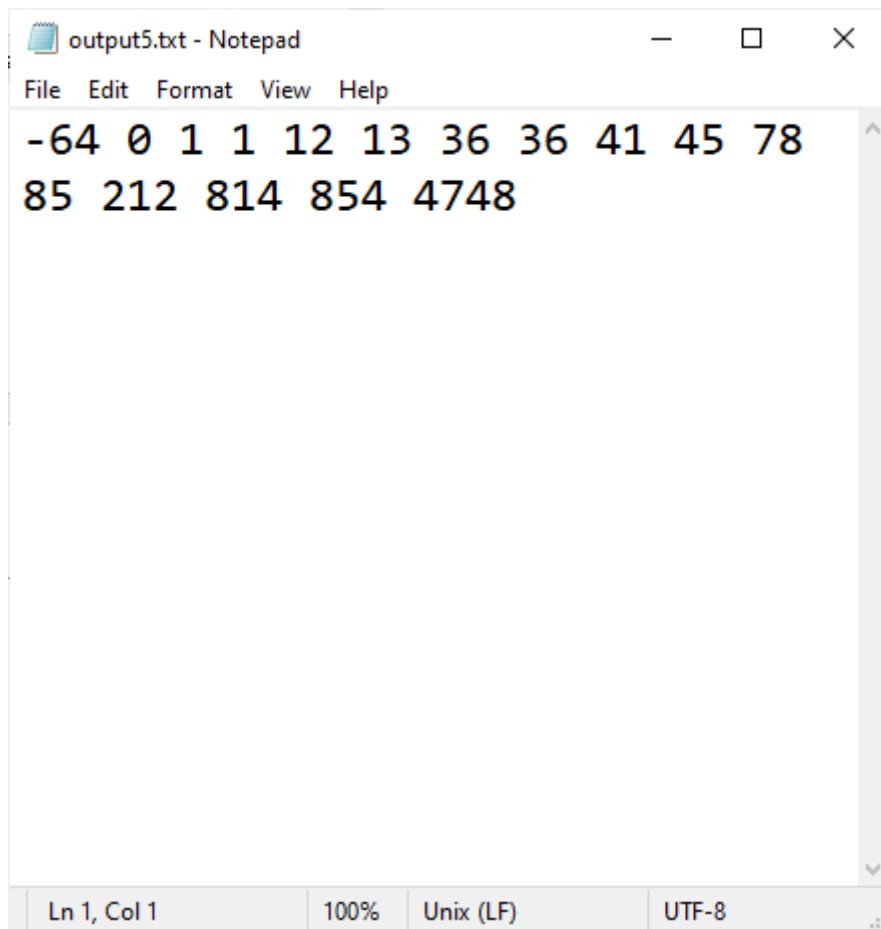
```
input.txt - Notepad
File Edit Format View Help
1 78 13 854 41 36 12 45 36 1 0
814 -64 212 4748 12
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

Messages	Run I/O
<input type="button" value="Clear"/>	User select sorting algorithm: 1. Bubble Sort 2. Insertion Sort 3. Selection Sort 4. Quick Sort 5. Close Choice: 1 Execution time (ms): 7 User select sorting algorithm: 1. Bubble Sort 2. Insertion Sort 3. Selection Sort 4. Quick Sort 5. Close Choice:

Messages	Run I/O
<input type="button" value="Clear"/>	User select sorting algorithm: 1. Bubble Sort 2. Insertion Sort 3. Selection Sort 4. Quick Sort 5. Close Choice: 2 Execution time (ms): 3 User select sorting algorithm: 1. Bubble Sort 2. Insertion Sort 3. Selection Sort 4. Quick Sort 5. Close Choice:

Messages	Run I/O
<div>Clear</div>	User select sorting algorithm: 1. Bubble Sort 2. Insertion Sort 3. Selection Sort 4. Quick Sort 5. Close Choice: 3 Execution time (ms): 3 User select sorting algorithm: 1. Bubble Sort 2. Insertion Sort 3. Selection Sort 4. Quick Sort 5. Close Choice:
	User select sorting algorithm: 1. Bubble Sort 2. Insertion Sort 3. Selection Sort 4. Quick Sort 5. Close Choice: 4 Execution time (ms): 3 User select sorting algorithm: 1. Bubble Sort 2. Insertion Sort 3. Selection Sort 4. Quick Sort 5. Close Choice:

Output:

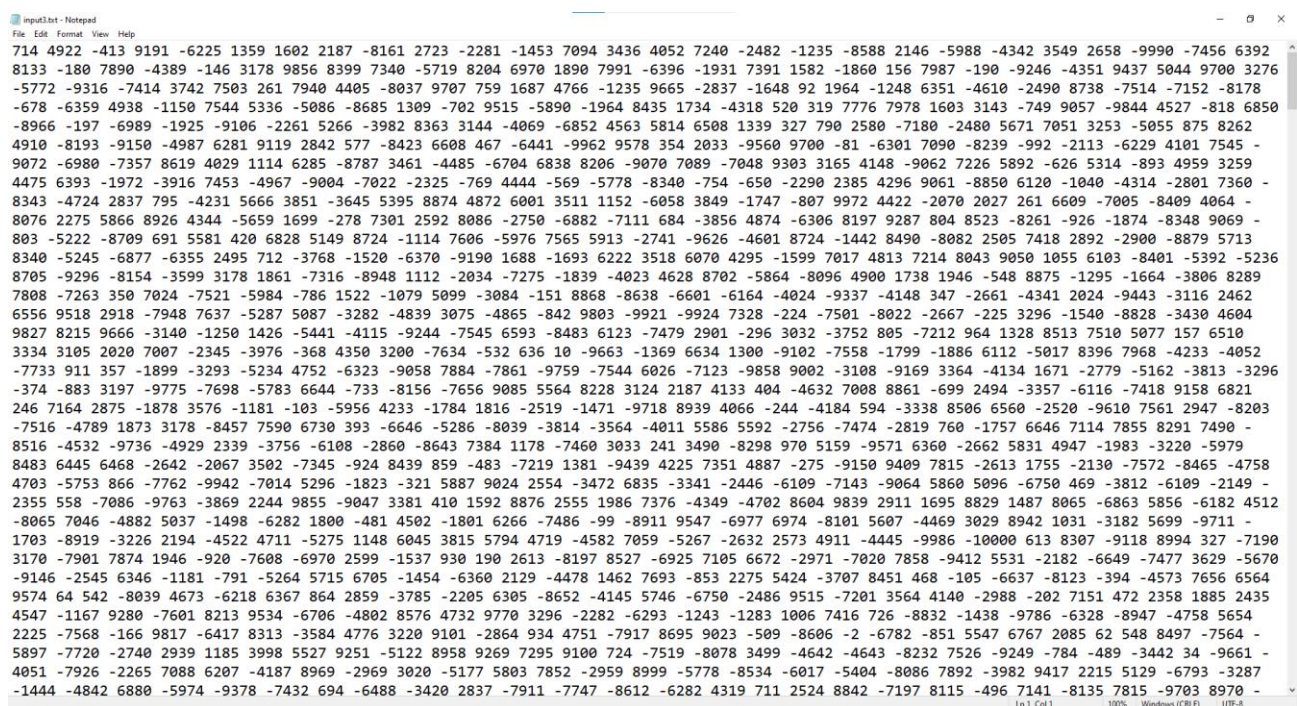


```

-64 0 1 1 12 13 36 36 41 45 78
85 212 814 854 4748

```

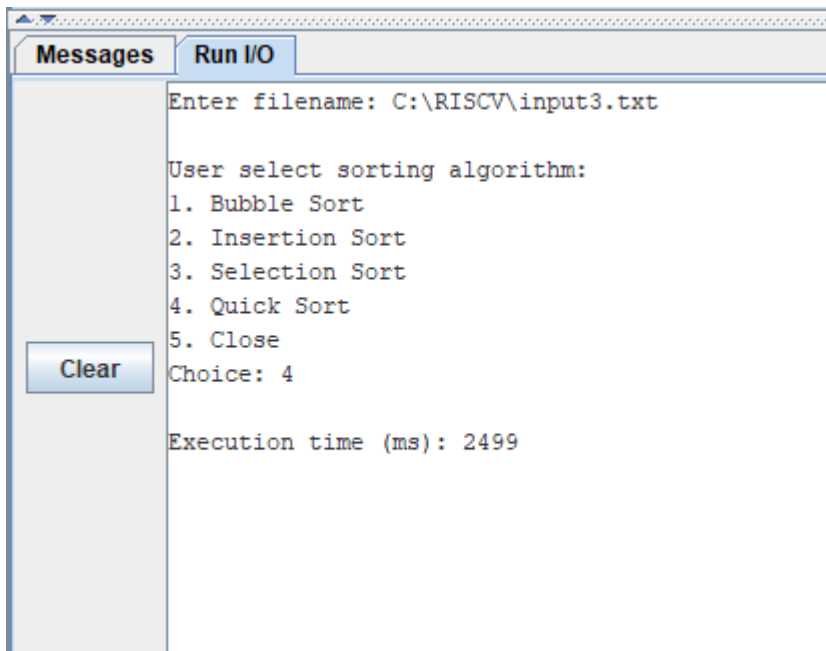
Với file đầu vào là file input3.txt: (Khoảng 10000 số)



```

714 4922 -413 9191 -6225 1359 1602 2187 -8161 2723 -2281 -1453 7094 3436 4052 7240 -2482 -1235 -8588 2146 -5988 -4342 3549 2658 -9990 -7456 6392
8133 -180 7890 -4389 -146 3178 9856 8399 7340 -5719 8204 6970 1890 7991 -6396 -1931 7391 1582 -1860 156 7987 -190 -9246 -4351 9437 5044 9700 3276
-5772 -9316 -7414 3742 7503 261 7940 4405 -8037 9707 759 1687 4766 -1235 9665 -2837 -1648 92 1964 -1248 6351 -4610 -2490 8738 -7514 -7152 -8178
-678 -6359 4938 -1150 7544 5336 -5086 -8685 1309 -702 9515 -5890 -1964 8435 1734 -4318 520 319 7776 7978 1603 3143 -749 9057 -9844 4527 -818 6850
-8966 -197 -6989 -1925 -9106 -2261 5266 -3982 8363 3144 -4069 -6852 4563 5814 6508 1339 327 790 2580 -7180 -2480 5671 7051 3253 -5055 875 8262
4910 -8193 -9150 -4987 6281 9119 2842 577 -8423 6608 467 -6441 -9962 9578 354 2033 -9560 9700 -81 -6301 7090 -8239 -992 -2113 -6229 4101 7545 -
9072 -6980 -7357 8619 4029 1114 6285 -8787 3461 -4485 -6704 6838 8206 -9070 7089 -7048 9303 3165 4148 -9062 7226 5892 -626 5314 -893 4959 3259
4475 6393 -1972 -3916 7453 -4967 -9004 -7022 -2325 -769 4444 -569 -5778 -8340 -754 -650 -2290 2385 4296 9061 -8850 6120 -1040 -4314 -2801 7360 -
8343 -4724 2837 795 -4231 5666 3851 -3645 5395 8874 4872 6001 3511 1152 -6058 3849 -1747 -807 9972 4422 -2070 2027 261 6609 -7005 -8409 4064 -
8076 2275 5866 8926 4344 -5659 1699 -278 7301 2592 8086 -2750 -6882 -7111 684 -3856 4874 -6306 8197 9287 804 8523 -8261 -926 -1874 -8348 9069 -
803 -5222 -8709 691 5581 420 6828 5149 8724 -1114 7606 -5976 7565 5913 -2741 -9626 -4601 8724 -1442 8490 -8082 2505 7418 2892 -2900 -8879 5713
8340 -5245 -6877 -6355 2495 712 -3768 -1520 -6370 -9190 1688 -1693 6222 3518 6070 4295 -1599 7017 4813 7214 8043 9050 1055 6103 -8401 -5392 -5236
8705 -9296 -8154 -3599 3178 1861 -7316 -8948 1112 -2034 -7275 -1839 -4023 4628 8702 -5864 -8096 4900 1738 1946 -548 8875 -1295 -1664 -3806 8289
7808 -7263 350 7024 -7521 -5984 -786 1522 -1079 5099 -3084 -151 8868 -8638 -6601 -6164 -4024 -9337 -4148 347 -2661 -4341 2024 -9443 -3116 2462
6556 9518 2918 -7948 7637 -5287 5087 -3282 -4839 3075 -4865 -842 9803 -9921 -9924 7328 -224 -7501 -8022 -2667 -225 3296 -1540 -8828 -3430 4604
9827 8215 9666 -3140 -1250 1426 -5441 -4115 -9244 -7545 6593 -8483 6123 -7479 2901 -296 3032 -3752 805 -7212 964 1328 8513 7510 5077 157 6510
3334 3105 2020 7007 -2345 -3976 -368 4350 3200 -7634 -532 636 10 -9663 -1369 6634 1300 -9102 -7558 -1799 -1886 6112 -5017 8396 7968 -4233 -4052
-7733 911 357 -1899 -3293 -5234 4752 -6323 -9058 7884 -7861 -9759 -7544 6026 -7123 -9858 9002 -3108 -9169 3364 -4134 1671 -2779 -5162 -3813 -3296
-374 -883 3197 -9775 -7698 -5783 6644 -733 -8156 -7656 9085 5564 8228 3124 2187 4133 404 -4632 7008 8861 -699 2494 -3357 -6116 -7418 9158 6821
246 7164 2875 -1878 3576 -1181 -103 -5956 4233 -1784 1816 -2519 -1471 -9718 8939 4066 -244 -4184 594 -3338 8506 6560 -2520 -9610 7561 2947 -8203
-7516 -4789 1873 3178 -8457 7590 6730 393 -6646 -5286 -8039 -3814 -3564 -4011 5586 5592 -2756 -7474 -2819 760 -1757 6646 7114 7855 8291 7490 -
8516 -4532 -9736 -4929 2339 -3756 -6108 -2860 -8643 7384 1178 -7460 3033 241 3490 -8298 970 5159 -9571 6360 -2662 5831 4947 -1983 -3220 -5979
8483 6445 6468 -2642 -2067 3502 -7345 -924 8439 859 -483 -7219 1381 -9439 4225 7351 4887 -275 -9150 9409 7815 -2613 1755 -2130 -7572 -8465 -4758
4703 -5753 866 -7762 -9942 -7014 5296 -1823 -321 5887 9024 2554 -3472 6835 -3341 -2446 -6109 -7143 -9064 5860 5096 -6750 469 -3812 -6109 -2149 -
2355 558 -7086 -9763 -3869 2244 9855 -9047 3381 410 1592 8876 2555 1986 7376 -4349 -4702 8604 9839 2911 1695 8829 1487 8065 -6863 5856 -6182 4512
-8065 7046 -4882 5037 -1498 -6282 1800 -481 4502 -1801 6266 -7486 -99 -8911 9547 -6977 6974 -8101 5607 -4469 3029 8942 1031 -3182 5699 -9711 -
1703 -8919 -3226 2194 -4522 4711 -5275 1148 6045 3815 5794 4719 -4582 7059 -5267 -2632 2573 4911 -4445 -9986 -10000 613 8307 -9118 8994 327 -7190
3170 -7901 7874 1946 -920 -7608 -6970 2599 -1537 930 190 2613 -8197 8527 -6925 7105 6672 -2971 -7020 7858 -9412 5531 -2182 -6649 -7477 3629 -5670
-9146 -2545 6346 -1181 -791 -5264 5715 6705 -1454 -6360 2129 -4478 1462 7693 -853 2275 5424 -3707 8451 468 -105 -6637 -8123 -394 -4573 7656 6564
9574 64 542 -8039 4673 -6218 6367 864 2859 -3785 -2205 6305 -8652 -4145 5746 -6750 -2486 9515 -7201 3564 4140 -2988 -202 7151 472 2358 1885 2435
4547 -1167 9280 -7601 8213 9534 -6706 -4802 8576 4732 9770 3296 -2282 -6293 -1243 -1283 1006 7416 726 -8832 -1438 -9786 -6328 -8947 -4758 5654
2225 -7568 -166 9817 -6417 8313 -3584 4776 3220 9101 -2864 934 4751 -7917 8695 9023 -509 -8606 -2 -6782 -851 5547 6767 2085 62 548 8497 -7564 -
5897 -7720 -2740 2939 1185 3998 5527 9251 -5122 8958 9269 7295 9100 724 -7519 -8078 3499 -4642 -4643 -8232 7526 -9249 -784 -489 -3442 34 -9661 -
4051 -7926 -2265 7088 6207 -4187 8969 -2969 3020 -5177 5803 7852 -2959 8999 -5778 -8534 -6017 -5404 -8086 7892 -3982 9417 2215 5129 -6793 -3287
-1444 -4842 6880 -5974 -9378 -7432 694 -6488 -3420 2837 -7911 -7747 -8612 -6282 4319 711 2524 8842 -7197 8115 -496 7141 -8135 7815 -9703 8970 -

```

Còn lại các thuật toán sắp xếp khác chạy rất chậm

Output:

```

-10000 -9999 -9998 -9998 -9996 -9996 -9994 -9991 -9990 -9987 -9986 -9985 -9983 -9983 -9982 -9978 -9977 -9975 -9973 -9971 -9971 -9969 -9966 -9962
-9960 -9952 -9952 -9951 -9950 -9949 -9947 -9944 -9942 -9942 -9940 -9939 -9938 -9936 -9936 -9924 -9923 -9922 -9921 -9920 -9916 -9909 -9903 -9899
-9898 -9897 -9895 -9895 -9890 -9889 -9887 -9885 -9884 -9884 -9883 -9882 -9881 -9880 -9878 -9877 -9876 -9873 -9870 -9870 -9869 -9867 -9866 -9864
-9863 -9861 -9858 -9855 -9855 -9852 -9852 -9850 -9849 -9846 -9844 -9841 -9840 -9837 -9835 -9827 -9820 -9819 -9818 -9815 -9815 -9814 -9810 -9810
-9808 -9800 -9797 -9795 -9786 -9786 -9783 -9781 -9775 -9771 -9771 -9768 -9764 -9763 -9761 -9759 -9759 -9755 -9754 -9753 -9751 -9749 -9746 -9736
-9735 -9734 -9733 -9732 -9731 -9728 -9725 -9723 -9721 -9718 -9718 -9714 -9711 -9707 -9706 -9706 -9705 -9704 -9703 -9702 -9696 -9694 -9692 -9692
-9692 -9688 -9684 -9681 -9681 -9681 -9680 -9674 -9667 -9666 -9666 -9666 -9664 -9663 -9662 -9662 -9661 -9660 -9657 -9650 -9649 -9646 -9646 -9642
-9640 -9639 -9636 -9635 -9634 -9631 -9630 -9630 -9628 -9626 -9626 -9623 -9622 -9620 -9620 -9619 -9616 -9610 -9608 -9602 -9602 -9599 -9598 -9598
-9598 -9596 -9593 -9590 -9587 -9585 -9581 -9581 -9580 -9579 -9578 -9576 -9571 -9571 -9571 -9567 -9566 -9565 -9565 -9564 -9562 -9560 -9560 -9559
-9559 -9558 -9553 -9550 -9550 -9547 -9547 -9544 -9544 -9543 -9541 -9537 -9533 -9531 -9524 -9523 -9523 -9516 -9515 -9514 -9512 -9509 -9509 -9507 -9503
-9497 -9492 -9491 -9491 -9490 -9485 -9478 -9467 -9462 -9454 -9451 -9450 -9450 -9449 -9449 -9449 -9443 -9443 -9443 -9439 -9439 -9438 -9437
-9435 -9434 -9433 -9432 -9432 -9429 -9427 -9423 -9422 -9420 -9420 -9418 -9415 -9412 -9412 -9411 -9408 -9407 -9406 -9406 -9404 -9403 -9402 -9401
-9401 -9400 -9397 -9396 -9392 -9390 -9389 -9386 -9383 -9382 -9379 -9378 -9378 -9377 -9374 -9373 -9372 -9372 -9370 -9369 -9369 -9367 -9365 -9361
-9359 -9357 -9356 -9354 -9345 -9344 -9343 -9342 -9339 -9337 -9335 -9335 -9333 -9333 -9324 -9324 -9324 -9323 -9322 -9322 -9319 -9316 -9315 -9313
-9312 -9311 -9307 -9305 -9302 -9299 -9298 -9297 -9296 -9291 -9291 -9288 -9288 -9285 -9283 -9281 -9280 -9279 -9279 -9277 -9271 -9269 -9268 -9267
-9267 -9265 -9260 -9259 -9259 -9255 -9249 -9248 -9246 -9244 -9244 -9244 -9242 -9241 -9239 -9233 -9232 -9227 -9223 -9219 -9217 -9217 -9216 -9216
-9214 -9208 -9201 -9194 -9192 -9190 -9189 -9189 -9188 -9184 -9180 -9179 -9176 -9175 -9174 -9171 -9171 -9169 -9168 -9167 -9166 -9165 -9160 -9153
-9152 -9150 -9150 -9150 -9150 -9147 -9146 -9145 -9144 -9141 -9138 -9131 -9126 -9126 -9125 -9124 -9120 -9120 -9119 -9118 -9118 -9117 -9116 -9111
-9110 -9109 -9107 -9106 -9106 -9105 -9104 -9103 -9102 -9100 -9098 -9096 -9095 -9094 -9093 -9093 -9091 -9089 -9084 -9082 -9081 -9081 -9080 -9075
-9073 -9073 -9072 -9072 -9070 -9070 -9068 -9067 -9064 -9062 -9059 -9058 -9057 -9053 -9049 -9049 -9047 -9045 -9041 -9040 -9037 -9031 -9027 -9027
-9026 -9025 -9016 -9015 -9004 -9003 -9001 -8999 -8998 -8998 -8995 -8993 -8992 -8992 -8989 -8987 -8986 -8984 -8980 -8980 -8979 -8979 -8978
-8976 -8976 -8975 -8972 -8970 -8969 -8969 -8967 -8967 -8966 -8966 -8964 -8963 -8963 -8961 -8957 -8955 -8955 -8953 -8953 -8951 -8951 -8950
-8948 -8948 -8947 -8945 -8940 -8936 -8934 -8933 -8928 -8926 -8924 -8922 -8921 -8920 -8919 -8913 -8911 -8902 -8902 -8902 -8902 -8899 -8897 -8894
-8891 -8887 -8886 -8885 -8881 -8880 -8879 -8879 -8875 -8875 -8874 -8870 -8869 -8859 -8857 -8857 -8850 -8849 -8846 -8844 -8840 -8836 -8835 -8835
-8832 -8832 -8831 -8830 -8828 -8815 -8813 -8813 -8813 -8806 -8805 -8801 -8799 -8794 -8793 -8792 -8790 -8790 -8787 -8786 -8786 -8783 -8781 -8781
-8778 -8776 -8773 -8770 -8767 -8766 -8764 -8762 -8761 -8759 -8758 -8756 -8756 -8755 -8748 -8742 -8735 -8735 -8734 -8730 -8730 -8728 -8727 -8725
-8721 -8720 -8717 -8716 -8714 -8710 -8709 -8708 -8706 -8705 -8705 -8704 -8703 -8702 -8702 -8701 -8697 -8695 -8693 -8686 -8686 -8685 -8685 -8683
-8682 -8682 -8680 -8676 -8674 -8673 -8672 -8671 -8671 -8671 -8668 -8664 -8662 -8660 -8658 -8657 -8657 -8657 -8652 -8652 -8650 -8643 -8643 -8638
-8635 -8634 -8633 -8630 -8628 -8624 -8622 -8620 -8617 -8616 -8612 -8606 -8604 -8602 -8601 -8600 -8598 -8598 -8596 -8590 -8588 -8587 -8585 -8582
-8580 -8578 -8576 -8576 -8574 -8574 -8572 -8571 -8570 -8570 -8568 -8566 -8565 -8565 -8565 -8563 -8560 -8558 -8558 -8557 -8553 -8552 -8544 -8544
-8541 -8540 -8538 -8535 -8534 -8533 -8532 -8530 -8530 -8523 -8521 -8521 -8519 -8519 -8518 -8517 -8516 -8514 -8514 -8513 -8511 -8510 -8509 -8506
-8506 -8506 -8501 -8497 -8495 -8492 -8491 -8491 -8488 -8486 -8485 -8483 -8482 -8482 -8481 -8478 -8477 -8477 -8477 -8474 -8471 -8470 -8469
-8468 -8466 -8465 -8465 -8464 -8462 -8459 -8457 -8457 -8456 -8454 -8448 -8447 -8446 -8444 -8444 -8438 -8434 -8430 -8423 -8420 -8419 -8419 -8418
-8415 -8415 -8414 -8412 -8410 -8409 -8405 -8403 -8402 -8401 -8400 -8398 -8398 -8394 -8394 -8393 -8392 -8390 -8388 -8386 -8385 -8383 -8383
-8377 -8376 -8375 -8374 -8373 -8370 -8369 -8364 -8363 -8362 -8361 -8361 -8361 -8361 -8359 -8359 -8352 -8351 -8351 -8351 -8350 -8348 -8348 -8347

```