

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

----- oOo -----



BÁO CÁO BÀI TẬP GIỮA KỲ 2024.2

Học phần: Thực hành kiến trúc máy tính
Mã học phần: IT3280

Giảng viên hướng dẫn:	TS. Nguyễn Đình Thuận
Sinh viên thực hiện:	Nguyễn Quý Đức - 20235682
Mã lớp:	156788

A. Bài 4

1. Phần dữ liệu (.data):

- ask: Chuỗi yêu cầu người dùng nhập số ("Enter n:").
- nl: Ký tự xuống dòng ("\n").
- end_msg: Chuỗi thông báo kết quả ("Total number of perfect nums: ")
- err_msg: Chuỗi thông báo lỗi (" is not a possitive num, please enter again: ").

2. Phần mã (.text):

- main: bắt đầu của chương trình.
- Vòng lặp nhập liệu (enter và error_handler):
 - Yêu cầu người dùng nhập số (ask, ecall với a7=4).
 - Đọc số nguyên nhập vào (ecall với a7=5).
 - Kiểm tra xem đầu vào có dương không (bgtz a0, begin). Nếu không, nhảy đến error_handler, in ra thông báo lỗi (err_msg), và lặp lại vòng lặp enter.
- begin: Chương trình chỉ bắt đầu xử lý sau khi nhận được đầu vào dương hợp lệ. s0 là số nguyên dương N
- Khởi tạo: s1 (biến đếm vòng lặp i) được khởi tạo là 1, và s2 (số lượng số hoàn hảo) là 0.
- Vòng lặp chính (main_loop):
 - Lặp từ $i = 1$ đến N (bge s1, s0, exit).
 - Với mỗi i , gọi chương trình con isperfect để kiểm tra xem i có phải là số hoàn hảo không.
 - Nếu isperfect trả về a0 với giá trị 1 (true), in ra i và tăng biến đếm s2.
- exit: Sau vòng lặp, in ra tổng số lượng số hoàn hảo tìm thấy (end_msg, s2) và kết thúc chương trình.

3. Chương trình con:

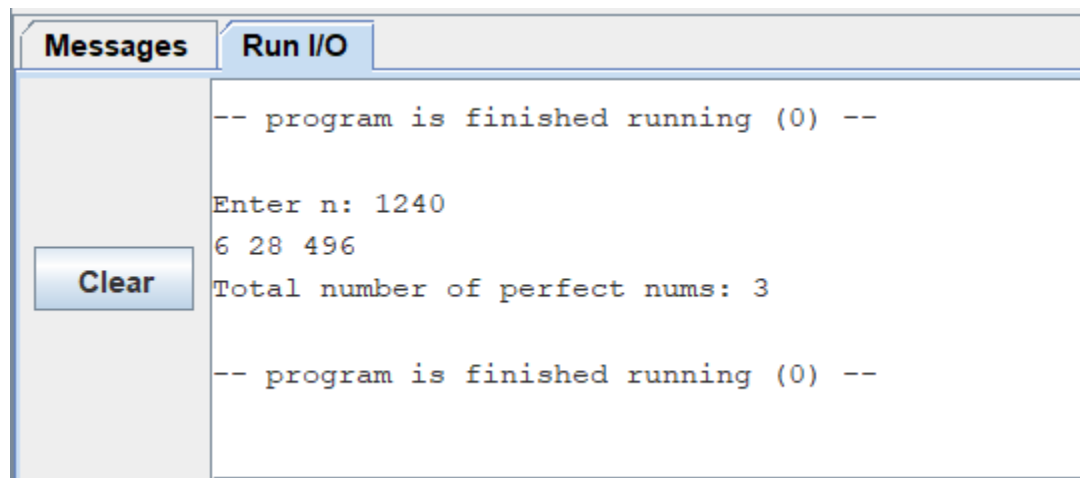
isperfect kiểm tra xem một số đã cho có phải là số hoàn hảo hay không.

If $(i == s(i))$, $isperfect(i) = \text{true}$.

- Input: a0 chứa số cần kiểm tra.
- Output: a0 chứa 1 nếu số đó là số hoàn hảo, 0 nếu không.
- Thuật toán:
 - Khởi tạo $t0 = i$.
 - t1 lưu trữ tổng các ước số (khởi tạo là 0).
 - t2 là hằng số = 2
 - $t3 = i/t2$ vì các ước số lớn hơn $i/2$ sẽ không tồn tại.
 - Vòng lặp (loop_check) lặp từ $t4 = 1$ đến $t3$.
 - Kiểm tra xem $t4$ có phải là ước của $t0$ không bằng phép chia lấy dư (kết quả rem lưu vào $t5$).
 - Nếu là ước ($t5 == 0$), cộng $t4$ vào tổng ($t1$).
 - Kiểm tra xem tổng các ước số ($t1$) có bằng số ban đầu ($t0$) không.
 - Nếu bằng nhau, n là số hoàn hảo, ret_yes: là a0, 1
 - Nếu không thì không phải (là a0, 0).
 - Nếu n hoàn hảo, in ra n

4. Thực nghiệm:

- Trường hợp nhập số nguyên dương (không vượt quá `int_max`):

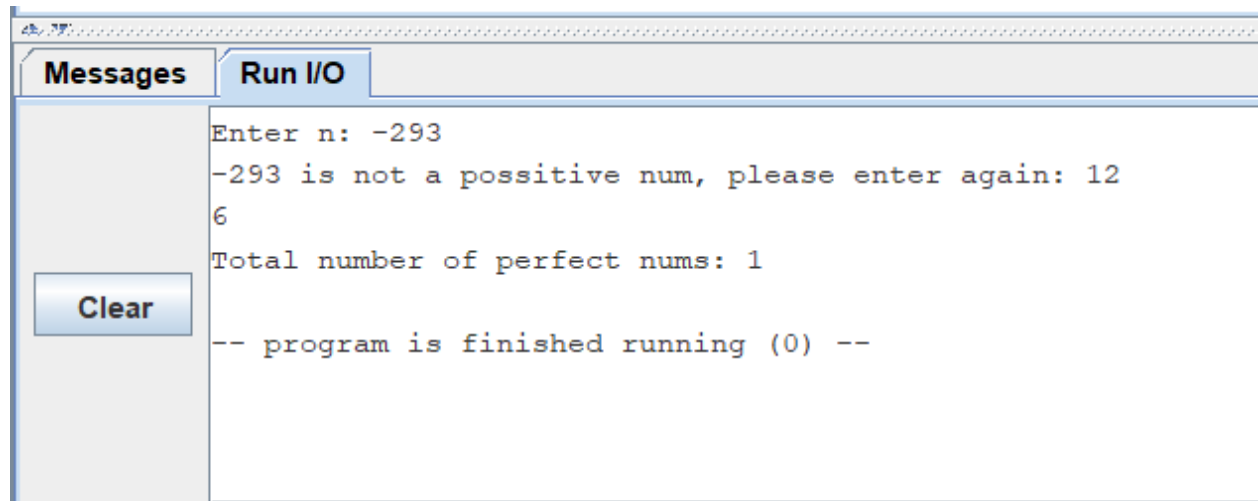


```
Messages Run I/O
-- program is finished running (0) --

Enter n: 1240
6 28 496
Total number of perfect nums: 3

-- program is finished running (0) --
```


- Trường hợp nhập số nguyên không dương sẽ bị bắt buộc nhập lại:



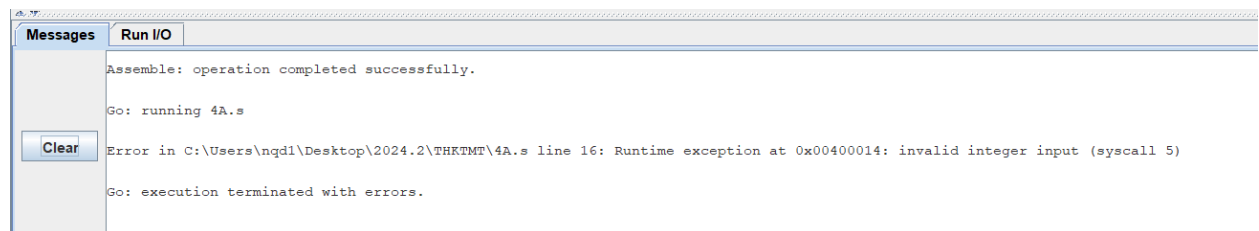
```

Enter n: -293
-293 is not a possitive num, please enter again: 12
6
Total number of perfect nums: 1
-- program is finished running (0) --

```

- Trường hợp dữ liệu nhập vào không phải kiểu int32 (fp, int64, ...):

Chương trình sẽ báo lỗi nhập sai định dạng số do syscall5 chỉ cho phép nhập số kiểu int32



```

Assemble: operation completed successfully.
Go: running 4A.s
Error in C:\Users\nqdl\Desktop\2024.2\THHTMT\4A.s line 16: Runtime exception at 0x00400014: invalid integer input (syscall 5)
Go: execution terminated with errors.

```

B. Bài 15

1. Phần dữ liệu (.data)

- ask_n: .asciz "enter array size: " yêu cầu người dùng nhập kích thước của mảng.
- ask_el: .asciz "enter element: " yêu cầu người dùng nhập một phần tử của mảng.

- ans: .asciz "least freq element: “ thông báo kết quả trước khi in ra phần tử có tần suất thấp nhất.
- nl: .asciz "\n" ký tự xuống dòng để định dạng output.
- err_msg: .asciz "an array cant have less than 0 elements!\n" khi nhập n là số nguyên âm.
- err_n0: .asciz "array is empty!\n" khi nhập n = 0

2. Phần mã (.text)

- main: bắt đầu của chương trình chính.
- Nhập kích thước mảng (n)
- la a0, ask_n; li a7, 4; ecall: In chuỗi ask_n ra màn hình.
- li a7, 5; ecall: Đọc số nguyên người dùng nhập (kích thước mảng) vào thanh ghi a0.
- bgtz a0, valid_n: nếu n nguyên dương thì hợp lệ.
- beqz a0, err_0: nếu n = 0 nhảy đến err_0 để xử lý.
- la a0, err_msg; li a7, 4; ecall : in ra err_msg do người dùng nhập số âm.
- j input: bắt người dùng nhập lại.
- err_0: la a0, err_n0; li a7, 4; ecall: in ra lỗi người dùng nhập n = 0
- j input: bắt người dùng nhập lại.
- valid_n: mv s0, a0: Lưu kích thước mảng n (thỏa mãn điều kiện n nguyên dương) vào thanh ghi s0.
- Cấp phát bộ nhớ trên Stack:
 - slli t0, s0, 3: Tính tổng kích thước cần cấp phát: 8n bytes với 4n bytes cho mảng array và 4n bytes cho mảng count.
 - sub sp, sp, t0: Di chuyển con trỏ ngăn xếp (sp) xuống để cấp phát không gian.
 - mv s1, sp: Lưu địa chỉ bắt đầu của vùng nhớ cấp phát (địa chỉ mảng array) vào thanh ghi s1.
 - slli t1, s0, 2: Tính kích thước của mảng array ($t1 = 4n$).

- add s2, s1, t1: Tính địa chỉ bắt đầu của mảng count ($s2 = s1 + t1$) và lưu vào s2.
- Vòng lặp nhập các phần tử mảng (read_loop)
- li t2, 0: Khởi tạo biến đếm vòng lặp i (thanh ghi t2) bằng 0.
- read_loop: bắt đầu vòng lặp.
 - bge t2, s0, read_done: Nếu $i \geq n$, nhảy đến read_done.
 - la a0, ask_el; li a7, 4; ecall: In chuỗi ask_el.
 - li a7, 5; ecall: Đọc phần tử người dùng nhập vào a0.
 - slli t3, t2, 2: Tính offset = $4i$.
 - add t4, s1, t3: Tính $\&array[i] = \&array + 4i$.
 - sw a0, 0(t4): Lưu phần tử vừa nhập vào array[i].
 - addi t2, t2, 1: Tăng i.
 - j read_loop: Quay lại đầu vòng lặp.
 - read_done: Kết thúc vòng lặp nhập.
- Vòng lặp đếm tần suất (outer_loop, inner_loop):
 - li t2, 0: Khởi tạo biến đếm vòng lặp ngoài i (t2).
 - outer_loop: Bắt đầu vòng lặp ngoài (duyệt qua từng phần tử để đếm).
 - bge t2, s0, outer_done: Nếu $i \geq n$, kết thúc.
 - Lấy giá trị array[i] lưu vào t5.
 - li t6, 0: Khởi tạo biến đếm vòng lặp trong j (t6).
 - li t1, 0: Khởi tạo biến đếm tần suất freq (t1) bằng 0 cho array[i].
 - inner_loop: Bắt đầu vòng lặp trong (so sánh array[i] với tất cả array[j]).
 - + bge t6, s0, store_freq: Nếu $j \geq n$, kết thúc vòng lặp trong.
 - + Lấy giá trị array[j] lưu vào t0.
 - + bne t0, t5, skip_add: Nếu array[j] không bằng array[i], bỏ qua việc tăng freq.
 - + addi t1, t1, 1: Nếu bằng, tăng freq.
 - + skip_add: Nhấn bỏ qua.

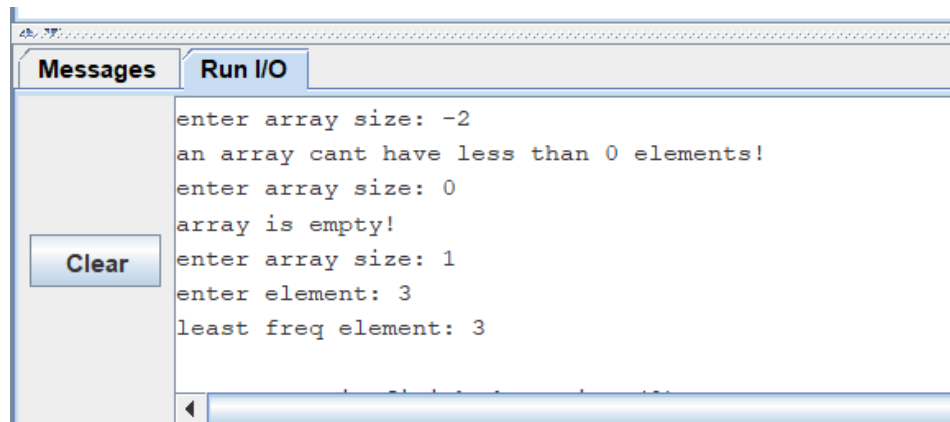
- + addi t6, t6, 1: Tăng j.
 - + j inner_loop: Quay lại đầu vòng lặp trong.
 - store_freq: Lưu tần suất vừa đếm được.
 - + Tính địa chỉ count[i] (add t4, s2, t3, với $t3 = i * 4$).
 - + sw t1, 0(t4): Lưu freq (t1) vào count[i].
 - addi t2, t2, 1: Tăng i.
 - j outer_loop: Quay lại đầu vòng lặp ngoài.
 - outer_done: Kết thúc đếm tần suất.
- Vòng lặp tìm tần suất nhỏ nhất (find_loop):
 - li t2, 0: Khởi tạo i (t2).
 - li t3, 2147483647: Khởi tạo min_freq (t3) = int_max
 - li t4, 0: Khởi tạo min_idx (t4) bằng 0 (chỉ số của phần tử có tần suất nhỏ nhất).
 - find_loop: Bắt đầu vòng lặp tìm kiếm.
 - bge t2, s0, print_res: Nếu $i \geq n$, kết thúc.
 - Lấy giá trị count[i] lưu vào t0.
 - blt t0, t3, update_min: Nếu $\text{count}[i] < \text{min_freq}$, nhảy đến update_min.
 - j skip_min: Nếu không, bỏ qua cập nhật.
 - update_min: Cập nhật giá trị nhỏ nhất.
 - + mv t3, t0: min_freq = count[i].
 - + mv t4, t2: min_idx = i.
 - skip_min: Nhấn bỏ qua.
 - addi t2, t2, 1: Tăng i.
 - j find_loop: Quay lại đầu vòng lặp tìm kiếm.
- In kết quả (print_res):
 - slli t5, t4, 2: Tính offset min_idx * 4.
 - add t6, s1, t5: Tính $\&\text{array}[\text{min_idx}] = \&\text{array} + t5$.
 - lw a1, 0(t6): Load giá trị array[min_idx] (phần tử cần tìm) vào a1.
 - la a0, ans; li a7, 4; ecall: In chuỗi ans.
 - mv a0, a1: Di chuyển phần tử cần in vào a0.

- li a7, 1; ecall: In giá trị phần tử (số nguyên) bằng syscall 1.
- la a0, nl; li a7, 4; ecall: In ký tự xuống dòng.
- Giải phóng bộ nhớ Stack và Kết thúc:
 - slli t0, s0, 3: Tính lại kích thước đã cấp phát ($n * 8$).
 - add sp, sp, t0: Di chuyển con trỏ ngăn xếp lên để giải phóng bộ nhớ.
 - li a7, 10; ecall: Kết thúc chương trình.

3. Thực nghiệm

- Trường hợp nhập n nguyên âm hoặc $n = 0$

Chương trình sẽ bắt nhập lại cho đến khi người dùng nhập n nguyên dương.

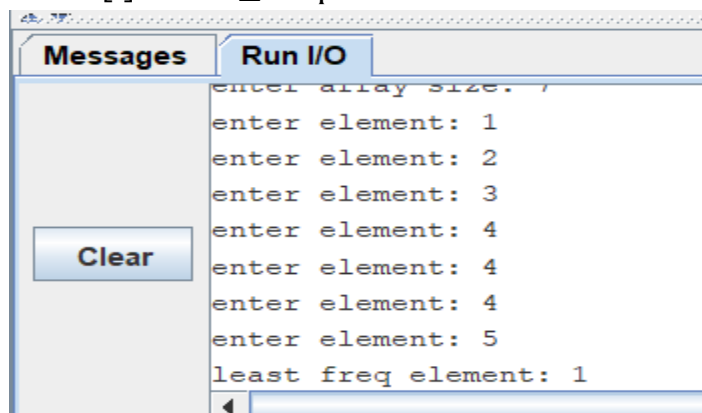


The screenshot shows a program window with two tabs: 'Messages' and 'Run I/O'. The 'Messages' tab is active, displaying the following text:

```
enter array size: -2
an array cant have less than 0 elements!
enter array size: 0
array is empty!
enter array size: 1
enter element: 3
least freq element: 3
```

There is a 'Clear' button on the left side of the message area.

- Trường hợp người dùng nhập mảng có nhiều phần tử đều xuất hiện ít nhất, VD: $arr = [1,2,3,4,4,4,5]$: Chương trình sẽ trả về phần tử đầu tiên có tần suất ít nhất do điều kiện `update_min` chỉ thay đổi khi `count[i] < min_freq`.



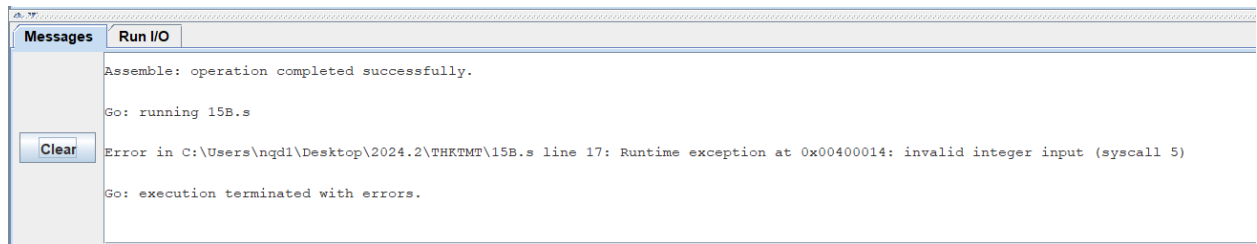
The screenshot shows the same program window as before, but with the following text in the 'Messages' tab:

```
enter array size: 7
enter element: 1
enter element: 2
enter element: 3
enter element: 4
enter element: 4
enter element: 4
enter element: 5
least freq element: 1
```

The 'Clear' button is still visible on the left.

- Trường hợp dữ liệu nhập vào không phải kiểu int32 (fp, int64, ...):

Chương trình sẽ báo lỗi nhập sai định dạng số do syscall5 chỉ cho phép nhập số kiểu int32



C. Bài 4

1. Phần dữ liệu (.data)

- ask: .asciz "enter string: " yêu cầu người dùng nhập chuỗi.
- input: .space 4096, dành ra một vùng nhớ 4096 byte để lưu chuỗi do người dùng nhập vào.
- nl: .asciz "\n", ký tự newline.
- output_msg: .asciz "longest word: " thông báo kết quả trước khi in ra từ dài nhất.
- error_msg: .asciz "error: empty input" thông báo lỗi nếu người dùng không nhập gì hoặc chuỗi không chứa chữ cái.

2. Phần mã (.text)

- main: Điểm bắt đầu thực thi của chương trình.
- In lời nhắc:
 - la a0, ask: Load địa chỉ của chuỗi ask vào thanh ghi a0.
 - li a7, 4: syscall 4 (in chuỗi) vào a7.
 - ecall: Thực hiện syscall để in lời nhắc.
- Đọc chuỗi (read str):
 - la a0, input: Load địa chỉ của buffer input vào a0.
 - li a1, 4096: Đặt kích thước tối đa của buffer vào a1.

- li a7, 8: Đặt mã syscall 8 (đọc chuỗi) vào a7.
- ecall: Thực hiện syscall để đọc chuỗi từ người dùng vào buffer input. Chuỗi đọc được sẽ kết thúc bằng ký tự null (\0) hoặc khi buffer đầy, bao gồm cả ký tự xuống dòng (\n) nếu người dùng nhấn Enter để kết thúc nhập.
- Kiểm tra chuỗi rỗng (chk empty input):
 - la s0, input: Load địa chỉ bắt đầu của chuỗi input vào s0 (dùng làm con trỏ duyệt chuỗi).
 - lb t0, 0(s0): Đọc ký tự đầu tiên của chuỗi vào t0.
 - li t1, 10: Load mã ASCII của ký tự xuống dòng (\n) vào t1.
 - beq t0, zero, print_error: Nếu ký tự đầu tiên là null (\0), nhảy đến print_error.
 - beq t0, t1, print_error: Nếu ký tự đầu tiên là \n, nhảy đến print_error.
- Khởi tạo con trỏ và độ dài
 - mv s1, s0: s1 lưu địa chỉ bắt đầu của từ hiện tại đang xét. Ban đầu trỏ vào đầu chuỗi.
 - mv s2, s0: s2 lưu địa chỉ bắt đầu của từ dài nhất tìm được cho đến nay. Ban đầu trỏ vào đầu chuỗi.
 - li s3, 0: s3 lưu độ dài của từ hiện tại đang xét. Ban đầu là 0.
 - li s4, 0: s4 lưu độ dài của từ dài nhất tìm được cho đến nay. Ban đầu là 0.
- Vòng lặp xử lý chuỗi:
 - lb t0, 0(s0): Đọc ký tự tại vị trí con trỏ s0 vào t0.
 - beq t0, zero, done: Nếu là ký tự null (\0), kết thúc vòng lặp, nhảy đến done.
 - Nếu là chữ cái (is_alph):
 - addi s3, s3, 1: Tăng độ dài từ hiện tại (s3).
 - addi s0, s0, 1: Di chuyển con trỏ s0 đến ký tự tiếp theo.
 - j loop: Quay lại đầu vòng lặp.
 - Nếu không là chữ cái (not_alph): check_max

- Kiểm tra và cập nhật từ dài nhất (check_max, update_max, reset_word):
 - check_max: kiểm tra độ dài từ chữ
 - bge s3, s4, update_max: So sánh độ dài từ hiện tại (s3) với độ dài lớn nhất đã lưu (s4). Nếu s3 >= s4, nhảy đến update_max.
 - j reset_word: Nếu s3 < s4, nhảy đến reset_word.
 - update_max: Cập nhật nếu từ hiện tại dài hơn hoặc bằng từ dài nhất trước đó.
 - mv s4, s3: Cập nhật độ dài lớn nhất: max_len = cur_len.
 - mv s2, s1: Cập nhật con trỏ bắt đầu từ dài nhất: max_ptr = cur_ptr.
 - reset_word: Chuẩn bị cho từ tiếp theo.
 - addi s0, s0, 1: Di chuyển con trỏ s0 sang kí tự tiếp theo. Nếu có nhiều kí tự liên tiếp không phải chữ cái, logic này vẫn xử lý đúng vì nó sẽ vào check_max với word_length s3 = 0 cho các kí tự thừa.
 - mv s1, s0: Đặt con trỏ bắt đầu từ tiếp theo (s1) là vị trí hiện tại của s0.
 - li s3, 0: Reset độ dài từ hiện tại (s3) về 0.
 - j loop: Quay lại vòng lặp chính để xử lý ký tự tiếp theo.
- Xử lý cuối chuỗi (done, update_final):
 - done: gọi khi gặp ký tự null (\0). Cần kiểm tra từ cuối cùng trong chuỗi (từ ký tự sau dấu cách cuối cùng đến \0).
 - bge s3, s4, update_final: So sánh độ dài từ cuối cùng (s3) với độ dài lớn nhất (s4). Nếu dài hơn hoặc bằng, nhảy đến update_final.
 - j print_result: Nếu ngắn hơn, nhảy thẳng đến in kết quả.
 - update_final: Cập nhật nếu từ cuối cùng là từ dài nhất.
 - mv s4, s3: max_len = cur_len.
 - mv s2, s1: max_ptr = cur_ptr.
- In kết quả (print_result, print_loop)

- print_result:
 - beqz s4, print_error: Kiểm tra nếu max_len (s4) bằng 0 (nghĩa là không tìm thấy từ nào, có thể do input rỗng hoặc không chứa chữ cái). Nếu bằng 0, nhảy đến print_error.
 - In thông báo output_msg bằng syscall 4.
 - mv t0, s4: Copy độ dài từ dài nhất (s4) vào t0 (dùng làm biến đếm cho vòng lặp in).
 - mv t1, s2: Copy con trỏ bắt đầu từ dài nhất (s2) vào t1 (dùng làm con trỏ đọc ký tự để in).
- print_loop: Vòng lặp in từng ký tự của từ dài nhất.
 - beqz t0, exit: Nếu biến đếm t0 bằng 0 (đã in đủ ký tự), nhảy đến exit.
 - lb a0, 0(t1): Đọc ký tự tại địa chỉ t1 vào a0.
 - li a7, 11: Đặt mã syscall 11 (in ký tự) vào a7.
 - ecall: Thực hiện in ký tự.
 - addi t1, t1, 1: Di chuyển con trỏ đọc t1 đến ký tự tiếp theo.
 - addi t0, t0, -1: Giảm biến đếm t0.
 - j print_loop: Quay lại đầu vòng lặp in.
- In lỗi (print_error):
 - In thông báo lỗi error_msg bằng syscall 4.
 - Nhảy đến exit.
- Kết thúc chương trình (exit):
 - In ký tự xuống dòng nl bằng syscall 4.
 - li a7, 10: Đặt mã syscall 10 (thoát chương trình) vào a7.
 - ecall: Thực hiện thoát chương trình.

3. Chương trình con check_alph:

Kiểm tra xem ký tự có phải chữ cái alphabet không:

- Input t0 là ký tự cần kiểm tra
- Output: void, nhảy đến is_alph hoặc not_alph

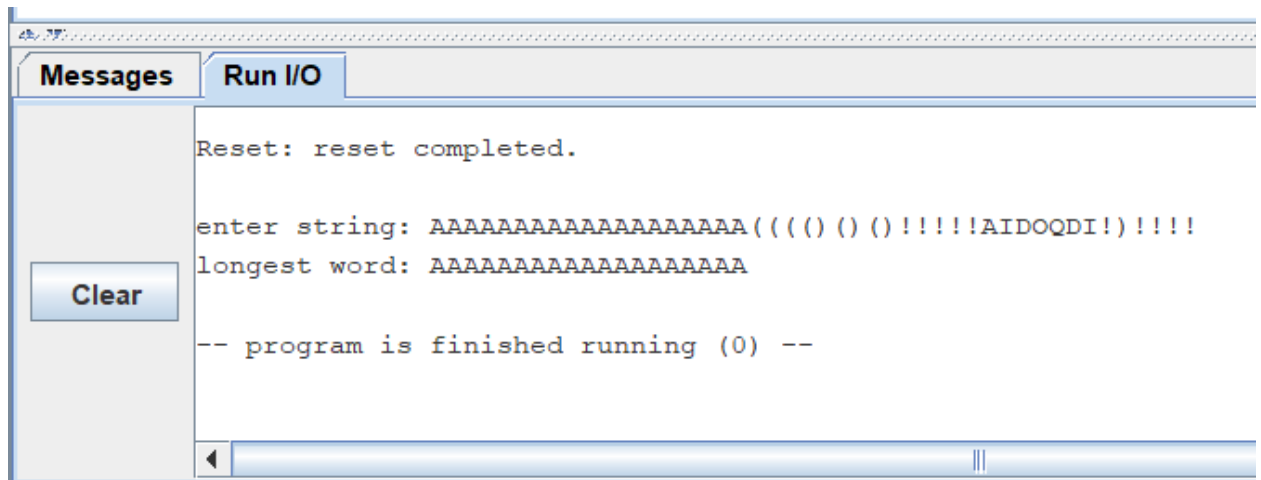
Thuật toán: kiểm tra xem t0 có trong [A,Z] hoặc [a,z] không:

- li t1, 0x41, mã ascii của ký tự A

- li t2, 0x5A, mã ascii của kí tự Z
- li t3, 0x61, mã ascii của kí tự a
- li t4, 0x7A, mã ascii của kí tự z
- Kiểm tra t0:
 - Nếu $t0 < A$ hoặc $t0 > z \rightarrow \text{not_alph}$
 - Nếu $t0 \text{ in range } [A, z]$
 - + Nếu $t0 \leq Z \rightarrow \text{is_alph}$
 - + Nếu $t0 > Z$, kiểm tra nếu $t0 < a$ thì $\rightarrow \text{not_alph}$, không thì is_alph

4. Thực nghiệm

- Trường hợp bình thường:



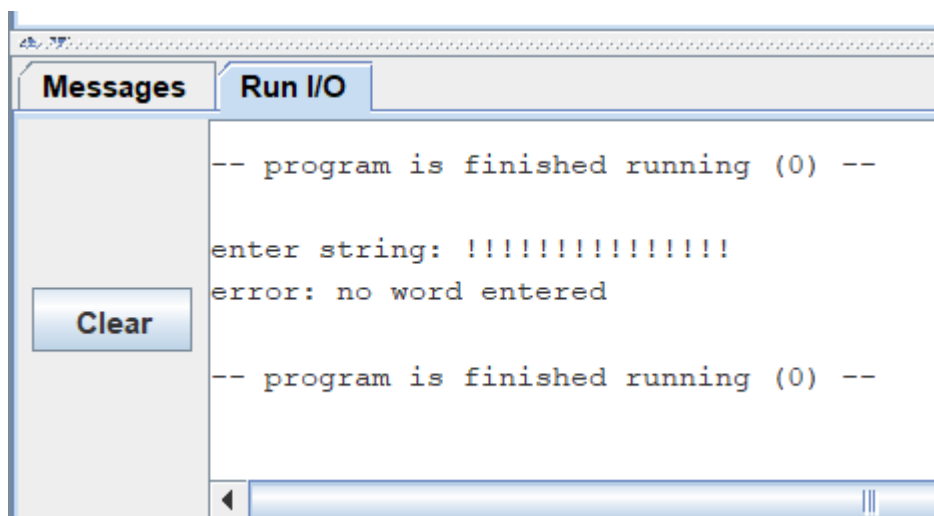
```

Reset: reset completed.

enter string: AAAAAAAAAAAAAAAAAA((( () () !!!!!AIDOQDI!)!!!!
longest word: AAAAAAAAAAAAAAAAAA

-- program is finished running (0) --
  
```

- Trường hợp không chữ cái nào:



```

-- program is finished running (0) --

enter string: !!!!!!!!!!!!!!!
error: no word entered

-- program is finished running (0) --
  
```

- Trường hợp không nhập gì:

