báo cáo bài tập tuần 2

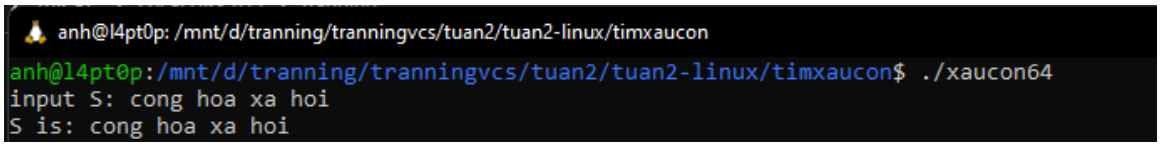
môi trường thực hiện: windows, linux 64 bit

linux: sử dụng nasm để build
- *nasm -f elf64 file.asm*: biên dịch tạo file object
- *ld file.o -o file*: static link file object để tạo file thực thi

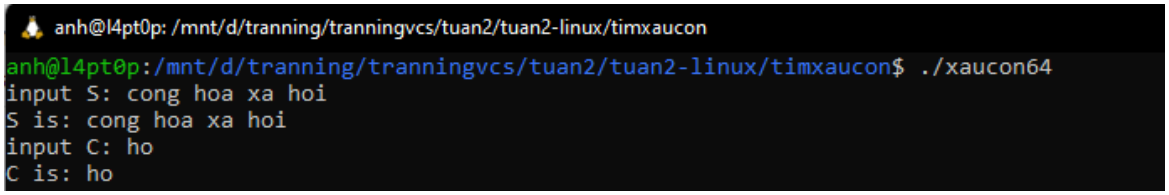
windows: sử dụng masm64 để build http://www.masm32.com/download/install64.zip
- *C:\masm\bin64\ml64 /c / file.asm:* /c để ml không link với file thực thi
- *C:\masm32\bin\link /subsystem:console /entry:main:* tạo chương trình chạy trên console

bài 5: tìm xâu con
- **linux:**
  - section .data tạo 5 trường data bào gồm:
    - chuỗi "input S:" để in ra màn hình l
    - chuỗi "input C:" để in ra màn hình
    - chuỗi "S is: ": để hiển thị xâu S nhập
    - chuỗi "C is: ": để hiển thị xâu C nhập
    - endline để in dấu cách
  - section .bss tạo 7 trường biến bào gồm
    - **s** để lưu xâu s
    - **c** để lưu xâu c
    - **s_len** để lưu độ dài xâu s
    - **c_len** để lưu độ dài xâu c
    - **count** để đếm số lần xuất hiện của xâu c trong xâu s
    - **pos_arr** để lưu trữ những vị trí xuất hiện của xâu c trong xâu s thành 1 mảng
    - **pos** để lưu từng giá trị của mảng pos_arr
  - section .text bao gồm code thực thi entry point là label _start
    - đầu tiên gọi 2 procedure là **ioS_proc** và **ioC_proc** để nhập 2 xâu S và C với sử dụng syscall, link tham khảo: Linux System Call Table for x86 64 · Ryan A. Chapman (rchapman.org)
      - trong đó đầu tiên in ra chuỗi "input S" ra màn hình với
        - rax = 1 lưu trữ opcode sys_write
        - rdi = 1 là stdout file descriptor
        - rsi là địa chỉ của chuỗi in ra màn hình
        - rdx là số lượng ký tự của chuỗi (với "input S" là 6)
        - syscall để ngắt cho phép in ra màn hình
      - tiếp theo cho phép nhập và lưu chuỗi nhập vào biến s
        - rax = 0 lưu trữ opcode sys_read
        - rdi = 0 là stdin file descriptor
        - rsi là địa chỉ chuỗi in ra màn hình

    - rdx là số lượng ký tự tối đa của chuỗi nhập vào
    - syscall để ngắt cho phép đọc ký tự
- cuối cùng cho in ra chuỗi là ký tự nhập vào của biến s tương tự như in ra chuỗi "S is"
- kết quả thu được:



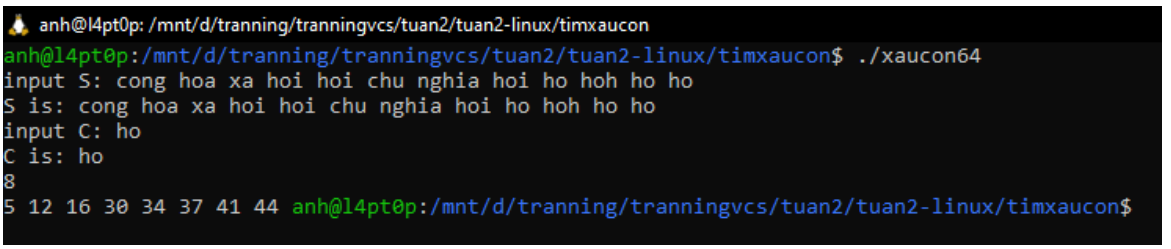
- tương tự với xâu C thu được:



    - tiếp theo đưa offset của 2 xâu s và c làm tham số cho procedure getlen và lưu độ dài 2 xâu vào **s_len** và **c_len**
        - procedure lần lượt check từng ký tự của tham số nhập vào cho đến ký tự 0xA, mỗi lần check thì cộng **rax** lên 1 để lưu trữ độ dài của xâu

    - sau khi thu được độ dài của 2 xâu s và c thực hiện tìm số lần xuất hiện của xâu c trong xâu **s** với lable **compare**
        - lable **compare** thực hiện so sánh từng ký tự của xâu **s** và xâu **c** cho đến hết xâu s và lưu vào **count,** với mỗi lần xuất hiện của xâu **c** trong xâu **s** thêm vào **pos_arr** vị trí tương ứng của xâu **s.**
            - so sánh thực hiện bằng cách so sánh từng ký tự, nếu 2 ký tự đang xét bằng nhau thì cộng offset 2 xâu lên 1 tiếp tục so sanh cho đến khi xâu **c** kết thúc thì reset lại offset của xâu **c** và so sánh tiếp.
            - nếu 2 ký tự đang xét không bằng nhau thì reset offset của xâu **c** luôn và so sánh với ký tự tiếp theo của xâu **s**

    - sau khi thu được số lần xuất hiện của xâu **c** trong xâu **s**, đưa offset của biến **count** vào làm tham số cho procedure **printNumber**
        - procedure **printNumber** thực hiện bằng cách chuyển từng ký tự của tham số và cộng lên 30 để thành chuỗi và sử dụng syscall để in ra màn hình
    - sau khi in ra màn hình biến **count**, in ký tự xuống dòng rồi làm tương tự để in các giá trị trong mảng **pos_arr**
    - cuối cùng thu được kết quả



- **windows:**
  - section .data: tương tự như linux
  - section .data?: tương tự như section .bss trong linux
  - section .code bao gồm code thực thi với entry point là main proc
    - luồng thực thi tương tự linux chỉ khác một số điểm:
      - procedure **ioS_proc** và **ioS_proc** sử dụng winapi **WriteConsole** và **ReadConsole** trong đó
        - **WriteConsole** được sử dụng trong procedure **WriteString** với tham số là địa chỉ của chuỗi cần in, đầu procedure push tất cả các tham số được sử dụng trong hàm vào stack để khôi phục sau khi hàm kết thúc. đầu tiên sử dụng api **lstrlen** để lấy ra độ dài của chuỗi đưa vào r15 và gọi api GetStdHandle vơi tham số rcx là -11(STD_OUTPUT_HANDLE) rồi gọi api **WirteConsole** với 4 tham số:
          - rcx: nStdHandle
          - rdx: offset chuỗi cần ghi
          - r8: số lượng ký tự in(lấy theo độ dài chuỗi trả về từ **lstrlen**)
          - r9: offset cho số lương ký tự in- lấy từ biến của procedure WriteString(rbp - 8)

        - **ReadConsole** được sử dụng trong procedure **ReadString** với 2 tham số là số lượng ký tự đọc và địa chỉ của chuỗi cần đọc, đầu procedure push tất cả các tham số được sử dụng trong hàm vào stack để khôi phục sau khi hàm kết thúc. đầu tiên gọi api GetStdHandle vơi tham số rcx là -10(STD_INPUT_HANDLE) rồi gọi api **WirteConsole** với 4 tham số:
          - rcx: nStdHandle
          - rdx: offset chuỗi cần đọc
          - r8: số lượng ký tự in(lấy theo tham số thứ nhất rbp + 16)
          - r9: offset cho số lương ký tự đọc- lấy từ biến của procedure WriteString(rbp - 8)
      - thay vì tự định nghĩa procedure **getlen** thì sử dụng api **lstrlen**

  - kết quả thu được

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.376]
(c) Microsoft Corporation. All rights reserved.

D:\tranning\tranningvcs\tuan2\tuan2-windows\timxaucon>xaucon64.exe
input S: cong hoa xa hoi co co cooc cococ ococ
S is: cong hoa xa hoi co co cooc cococ ococ
input C: co
C is: co
7
0 16 19 22 27 29 34
D:\tranning\tranningvcs\tuan2\tuan2-windows\timxaucon>
```

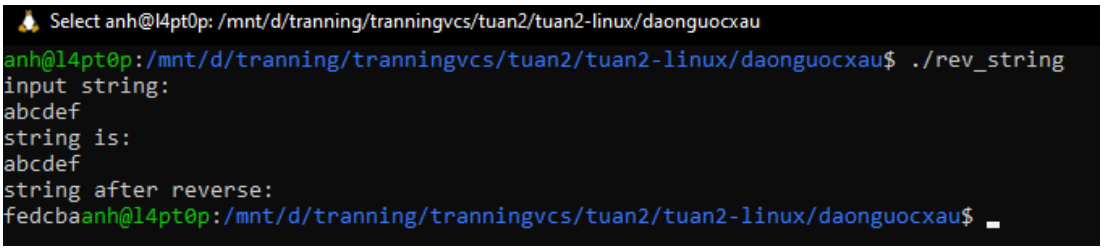
bài 6: đảo ngược xâu

- **linux:**
  - section .data tạo 4 trường data bào gồm:
    - chuỗi input string: ” để in ra màn hình
    - chuỗi “rev string is: ” để in ra màn hình
    - chuỗi “string is: ”: để hiển thị xâu S nhập
    - chuỗi “string after reverse: ”: để hiển thị xâu S sau khi đảo ngược
    - endline để in dấu cách
  - section .bss tạo 2 trường biến bào gồm
    - **s** để lưu xâu s
    - **s_len** để lưu xâu độ dài xâu s
  - section .text bao gồm code thực thi entry point là label _start
    - đầu tiên gọi procedure là **ioS_proc** để nhập xâu S với sử dụng syscall, link tham khảo: [Linux System Call Table for x86 64 · Ryan A. Chapman (rchapman.org)]
      - trong đó đầu tiên in ra chuỗi “input string: ” ra màn hình với
        - rax = 1 lưu trữ opcode sys_write
        - rdi = 1 là stdout file descriptor
        - rsi là địa chỉ của chuỗi in ra màn hình
        - rdx là số lượng ký tự của chuỗi (vơi “input string: ” là 15)
        - syscall để ngắt cho phép in ra màn hình
      - tiếp theo cho phép nhập và lưu chuỗi nhập vào biến s
        - rax = 0 lưu trữ opcode sys_read
        - rdi = 0 là stdin file descriptor
        - rsi là địa chỉ chuỗi in ra màn hình
        - rdx là số lượng ký tự tối đa của chuỗi nhập vào
        - syscall để ngắt cho phép đọc ký tự
      - cuối cùng cho in ra chuỗi là ký tự nhập vào của biến s tương tự như in ra chuỗi “string is: ”

```
anh@l4pt0p: /mnt/d/tranning/tranningvcs/tuan2/tuan2-linux/daonguocxau
anh@l4pt0p:/mnt/d/tranning/tranningvcs/tuan2/tuan2-linux/daonguocxau$ ./rev_string
input string:
abcdef
string is:
abcdef
```

- tiếp theo gọi push offset vào procedure **getlen** và thực thi để thu được độ dài của chuỗi nhập vào và lưu vào biến **s_len,** luồng thực thi của s_len tương tự bài 5.
- cuối cùng push QWORD[**s_len**] và offset **s** vào procedure **rev_string** để đảo ngược xâu
  - procedure **rev_string** thực hiện bằng cách cộng offset của ký tự đầu tiên của chuỗi với độ dài chuỗi và chuyển giá trị của offset này vào trong biến rbp - 0x100, sau mỗi lần chuyển giá trị thì trừ độ dài chuỗi đi 1, khi độ dài chuỗi bằng 0 thì kết thúc và in giá trị trong biến rbp - 0x100 ra màn hình.

- kết quả thu được



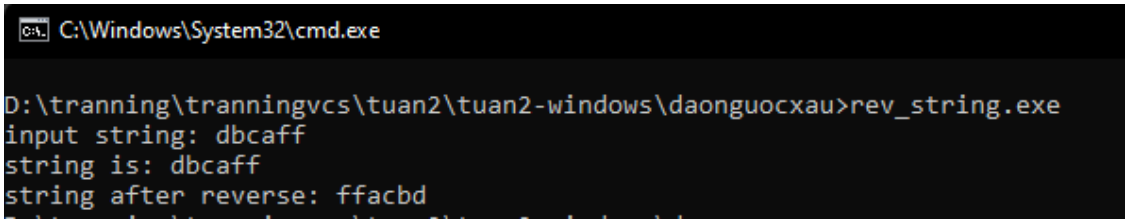
- **windows:**
  - section .data: tương tự như linux
  - section .data?: tương tự như section .bss trong linux
  - section .code bao gồm code thực thi với entry point là main proc
    - luồng thực thi tương tự linux chỉ khác một số điểm:
      - procedure **ioS_proc** sử dụng winapi **WriteConsole** và **ReadConsole** trong đó:
        - **WriteConsole** được sử dụng trong procedure **WriteString** với tham số là địa chỉ của chuỗi cần in, đầu procedure push tất cả các tham số được sử dụng trong hàm vào stack để khôi phục sau khi hàm kết thúc. đầu tiên sử dụng api **lstrlen** để lấy ra độ dài của chuỗi đưa vào r15 và gọi api GetStdHandle vơi tham số rcx là -11(STD_OUTPUT_HANDLE) rồi gọi api **WirteConsole** với 4 tham số:
          - rcx: nStdHandle
          - rdx: offset chuỗi cần ghi
          - r8: số lượng ký tự in(lấy theo độ dài chuỗi trả về từ **lstrlen**)
          - r9: offset cho số lương ký tự in- lấy từ biến của procedure WriteString(rbp - 8)

        - **ReadConsole** được sử dụng trong procedure **ReadString** với 2 tham số là số lượng ký tự đọc và địa chỉ của chuỗi cần đọc, đầu procedure push tất cả các tham số được sử dụng trong hàm vào stack để khôi phục sau khi hàm kết thúc. đầu tiên gọi api

GetStdHandle với tham số rcx là -10(STD_INPUT_HANDLE) rồi gọi api **WirteConsole** với 4 tham số:

- rcx: nStdHandle
- rdx: offset chuỗi cần đọc
- r8: số lượng ký tự in(lấy theo tham số thứ nhất rbp + 16)
- r9: offset cho số lương ký tự đọc- lấy từ biến của procedure WriteString(rbp - 8)

- thay vì tự định nghĩa hàm **getlen** thì sử dụng api **lstrlen**
- kết quả thu được:

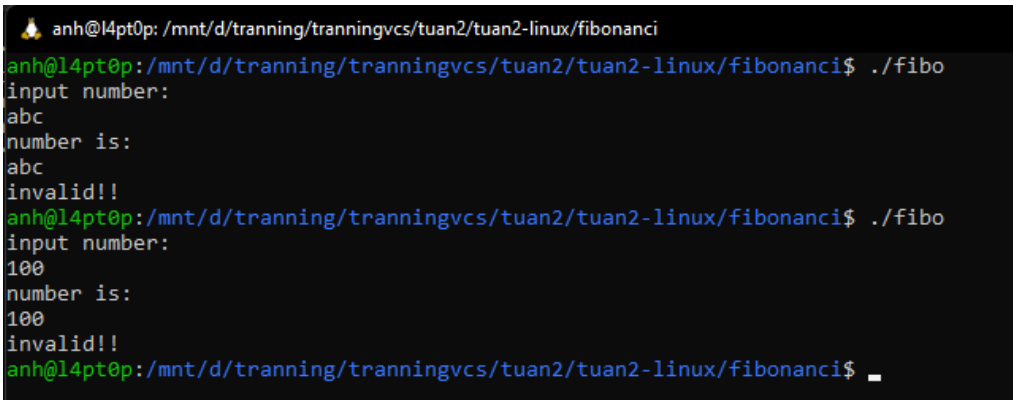


bài 3: fibonanci

- **linux:**
  - section .data tạo 4 trường data bào gồm:
    - chuỗi "input number :" để in ra màn hình
    - chuỗi "number is: " để in ra màn hình
    - chuỗi "invalid!! ": để hiển thị ra màn hình
    - chuỗi " ": để hiển thị dấu cách
    - endline để in dấu cách
  - section .bss tạo 1 trường biến bào gồm
    - n để lưu xâu chứa số n
  - section .text bao gồm code thực thi entry point là label _start
    - đầu tiên gọi procedure là **ioN_proc** để nhập xâu n cho số n sử dụng syscall, link tham khảo: [Linux System Call Table for x86 64 · Ryan A. Chapman (rchapman.org)]
      - trong đó đầu tiên in ra chuỗi "input number:" ra màn hình với
        - rax = 1 lưu trữ opcode sys_write
        - rdi = 1 là stdout file descriptor
        - rsi là địa chỉ của chuỗi in ra màn hình
        - rdx là số lượng ký tự của chuỗi (với "input number" là 15)
        - syscall để ngắt cho phép in ra màn hình
      - tiếp theo cho phép nhập và lưu chuỗi nhập vào biến s
        - rax = 0 lưu trữ opcode sys_read
        - rdi = 0 là stdin file descriptor
        - rsi là địa chỉ chuỗi in ra màn hình
        - rdx là số lượng ký tự tối đa của chuỗi nhập vào
        - syscall để ngắt cho phép đọc ký tự
      - cuối cùng cho in ra chuỗi là ký tự nhập vào của biến n tương tự như in ra chuỗi "number is"

```
anh@l4pt0p:/mnt/d/tranning/tranningvcs/tuan2/tuan2-linux/fibonanci$ ./fibo
input number:
10
number is:
10
```

- tiếp theo đưa offset của xâu **n** vào stack làm tham số cho procedure **checknum** để kiểm tra tính hợp lệ của số **n**
    - procedure lần lượt check từng ký tự của tham số nhập vào cho đến ký tự 0xA, check được thực hiện bằng cách so sánh xem ký tự đang xét có phải là số không bằng cách so sánh trong khoảng 0x30-0x39 và kiểm tra xem số nhập vào không vượt quá 2 chữ s, nếu một trong 2 điều kiện trên không thỏa mãn sẽ in ra "**invalid**" và thoát chương trình



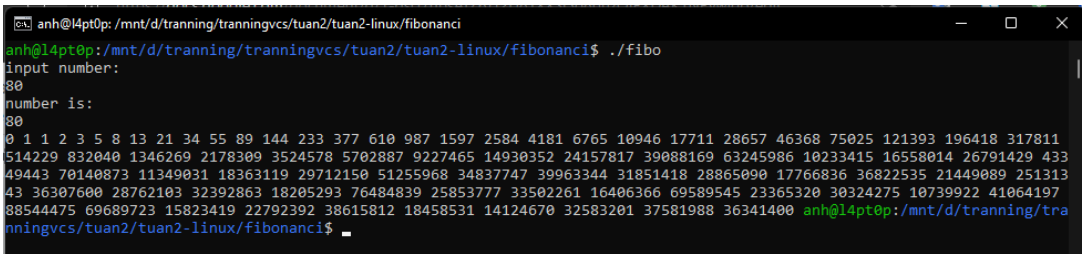
- nếu chuỗi nhập hợp lệ, đưa giá trị của xâu **n** vào làm tham số cho procedure **atoi** để chuyển xâu n thành số
    - procedure atoi thực hiện bằng cách trừ từng ký tự trong n cho 0x30 rồi nhân với 10 và cộng với số của ký tự tiếp theo (VD: 23 = 2 * 10 + 3)

- cuối cùng thực hiện tìm và in dãy số fibonanci của số **n** bằng procedure fibo và tham số là số n trả về từ procedure **atoi**
    - procedure thực hiện kiểm tra:
        - nếu tham số n < 0 thì thoát
        - nếu tham số n = 1 thì in ra 0 rồi thoát
        - nếu tham số n = 2 thì in ra 0 1 rồi tiếp tục thực hiện bằng cách cộng 2 số trước để tiếp tục tạo số mới và in lần lượt ra bằng cách nếu số đang xét có 1 chữ số thì cộng với 0x30 và in ra màn hình sử dụng syscall, nếu số đang xét có nhiều hơn 1 chữ số thì gọi procedure **splitnum** để thực hiện
    - procedure **splitnum** thực hiện bằng cách:
        - chia dư số đang xét cho 10 để lấy ra chữ số cuối cùng của số đang xét(VD: 123 % 10 = 3), với instruction div thì thương số sẽ được lưu trong rax và số dư sẽ được

lưu trong rdx(VD: rax= 123 , rbx = 10, div rbx => rax = 12, rdx = 3), với phương pháp này, làm một cách tuần từ thì sẽ thu được từng ký tự của số **n** trong tham số của **splitnum** thông qua thanh ghi rdx và chuyển lần lượt từng ký tự thu được vào biến rbp - 0x40 rồi đảo ngược lại(do các ký tự lấy bị ngược) bằng phương pháp tương tự bài 6, thu được chuỗi của số tiếp theo, thực hiện in ra màn hình

- thực hiện lặp 2 procedure **fibo** và **spitnum** cho đến khi **bl**(**bl** được khởi tạo ban đầu bằng 2) và số n bằng nhau thì dừng

- kết quả thu được



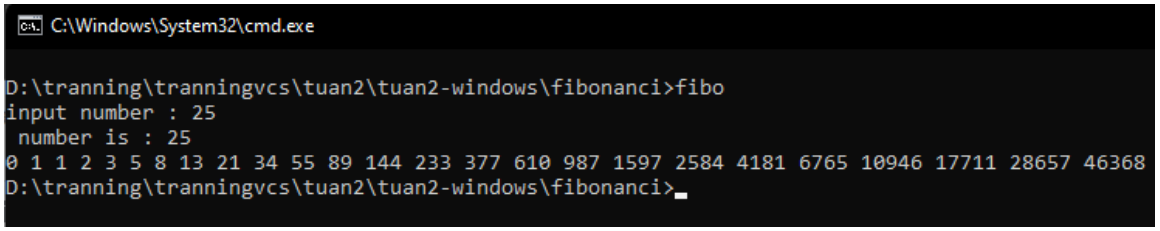
- **windows:**
  - section .data: tương tự như linux
  - section .data?: tương tự như section .bss trong linux
  - section .code bao gồm code thực thi với entry point là main proc
    - luồng thực thi tương tự linux chỉ khác một số điểm:
      - procedure **ioN_proc** sử dụng winapi **WriteConsole** và **ReadConsole** trong đó:
        - **WriteConsole** được sử dụng trong procedure **WriteString** với tham số là địa chỉ của chuỗi cần in, đầu procedure push tất cả các tham số được sử dụng trong hàm vào stack để khôi phục sau khi hàm kết thúc. đầu tiên sử dụng api **lstrlen** để lấy ra độ dài của chuỗi đưa vào r15 và gọi api GetStdHandle vơi tham số rcx là -11(STD_OUTPUT_HANDLE) rồi gọi api **WirteConsole** với 4 tham số:
          - rcx: nStdHandle
          - rdx: offset chuỗi cần ghi
          - r8: số lượng ký tự in(lấy theo độ dài chuỗi trả về từ **lstrlen**)
          - r9: offset cho số lương ký tự in- lấy từ biến của procedure WriteString(rbp - 8)

        - **ReadConsole** được sử dụng trong procedure **ReadString** với 2 tham số là số lượng ký tự đọc và địa chỉ của chuỗi cần đọc, đầu procedure push tất cả các tham số được sử dụng trong hàm vào stack để khôi phục sau khi hàm kết thúc. đầu tiên gọi api

GetStdHandle với tham số rcx là -10(STD_INPUT_HANDLE) rồi gọi api **WirteConsole** với 4 tham số:

- rcx: nStdHandle
- rdx: offset chuỗi cần đọc
- r8: số lượng ký tự in(lấy theo tham số thứ nhất rbp + 16)
- r9: offset cho số lương ký tự đọc- lấy từ biến của procedure WriteString(rbp - 8)

- kết quả thu được:

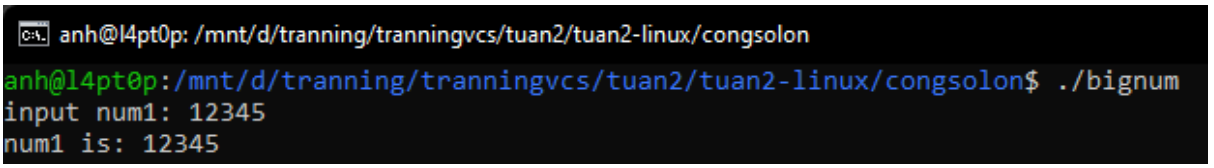


bải 4: simple_addtion

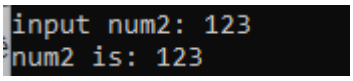
- **linux:**
  - section .data tạo 6 trường data bào gồm:
    - chuỗi "input num1 :" để in ra màn hình
    - chuỗi "num1 is: " để in ra màn hình
    - chuỗi "input num2 :" để in ra màn hình
    - chuỗi "num2 is: " để in ra màn hình
    - chuỗi "invalid!! ": để hiển thị ra màn hình
    - endline để in dấu cách
  - section .bss tạo 9 trường biến bào gồm
    - num1 để lưu xâu chứa số num1
    - num2 để lưu xâu chứa số num2
    - sum để lưu xâu chứa số n
    - num1_len để lưu xâu chứa độ dài num1
    - num2_len để lưu xâu chứa độ dài num2
    - sum_len để lưu xâu chứa dộ dài tổng num1+num2
    - rev_num1 để lưu xâu chứa số num1 đảo ngược
    - rev_num2 để lưu xâu chứa số num2 đảo ngược
    - rev_sum để lưu xâu chứa số tổng num1 + num2 đảo ngược

  - section .text bao gồm code thực thi entry point là label _start
    - đầu tiên gọi 2 procedure là **ion1_proc** và **ion1_proc** để nhập xâu **num1** cho số **n1** và xâu **num2** cho số **n2** sử dụng syscall, link tham khảo: [Linux System Call Table for x86 64 · Ryan A. Chapman (rchapman.org)]
      - trong đó đầu tiên in ra chuỗi "input num1:" ra màn hình với
        - rax = 1 lưu trữ opcode sys_write
        - rdi = 1 là stdout file descriptor
        - rsi là địa chỉ của chuỗi in ra màn hình
        - rdx là số lượng ký tự của chuỗi (với "input num" là
        - syscall để ngắt cho phép in ra màn hình

- tiếp theo cho phép nhập và lưu chuỗi nhập vào biến s
  - rax = 0 lưu trữ opcode sys_read
  - rdi = 0 là stdin file descriptor
  - rsi là địa chỉ chuỗi in ra màn hình
  - rdx là số lượng ký tự tối đa của chuỗi nhập vào
  - syscall để ngắt cho phép đọc ký tự
- cuối cùng cho in ra chuỗi là ký tự nhập vào của biến n tương tự như in ra chuỗi "num1 is"



- tương tự với num2 thu được



  - tiếp theo đưa offset của xâu **num1** và **num2** vào stack làm tham số cho procedure **checknum** để kiểm tra tính hợp lệ của số **num1** và **num2**
    - đầu tiên procedure check số lượng ký tự của tham số truyền vào với procedure getlen(tương tự bài 6) và kiểm tra xem độ dài xâu có vượt quá 21 ký tự không - 21 ký tự do số ký tự nhập không quá 20 + \n là 21 ký tự, nếu không thỏa mãn sẽ in ra "**invalid**" và thoát chương trình.

    - tiếp theo procedure lần lượt check từng ký tự của tham số nhập vào cho đến ký tự 0xA, check được thực hiện bằng cách so sánh xem ký tự đang xét có phải là số không bằng cách so sánh trong khoảng 0x30-0x39 và kiểm tra xem số nhập vào không vượt quá 2 chữ s, nếu một trong 2 điều kiện trên không thỏa mãn sẽ in ra "**invalid**" và thoát chương trình.
  - tiếp theo sẽ đảo ngược 2 xâu nhập vào của **num1** và **num2,** cách thực hiện tương tự như bài số 6, mục đích đảo để khi cộng sẽ cộng các số từ hàng đợn vị với nhau cho đến hang chục, hàng trăm,...

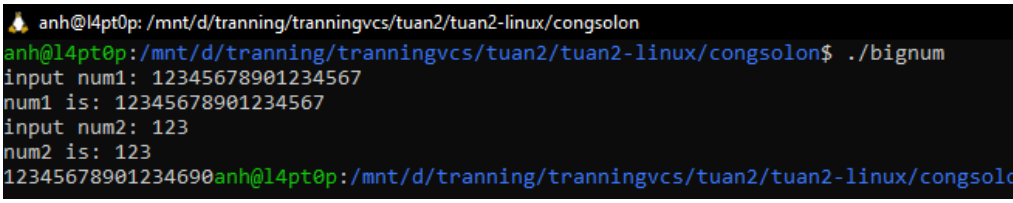
  - sau khi đã đảo ngược 2 xâu **num1** và **num2**, lưu trữ vào 2 xâu **rev_num1** và **rev_num2,** kiểm tra nếu số lượng ký tự của num1 > num2 thì lần lượt push offset của **sum**, **rev_num1**, **rev_num2** làm tham số cho procedure **addNum1_2,** hàm **addNum1_2** thực hiện cộng khi **num1_len > num2_len**

  - thực hiện trừ từng ký tự của **rev_num1** và **rev_num2** cho 0x30 để chuyển thành số và cộng lại với nhau
    - nếu cộng 2 chữ số với nhau < 9 thì cộng tổng 2 chữ số lại với 0x30 rồi chuyển ký tự này vào **sum** và tiếp tục.

- nếu cộng 2 chữ số với nhau > 9 thì cộng với số nhớ là r12b(nếu có) và tách 2 chữ số ra thành chữ số hàng đơn vị sẽ được cộng 0x30 và chuyển vào **sum** còn chữ số hàng chục sẽ được chuyển vào r12b làm số nhớ. Thực hiện bằng cách chia số này cho 10 thì số hàng chục sẽ nằm trong rax(al) còn số hàng đơn vị sẽ nằm trong rdx(dl)

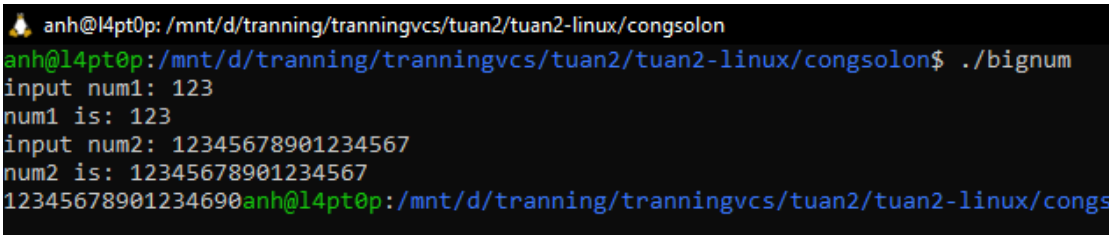
- nếu **rev_num2 =** 0 thì sẽ không trừ với 0x30 mà sẽ cộng tiếp với từng ký tự của **rev_num1** tiếp theo trừ 0x30 do lúc nãy **rev_num2** đã kết thúc chuỗi còn **rev_num2** vẫn còn

- lặp lại cho đến khi kết thúc chuỗi **rev_num1** thì dừng và thoát khỏi procedure, lúc này **sum** đang lưu trữ chuỗi chứa tổng của 2 số **num1** và **num2** đảo ngược. Cuối cùng chỉ cần thực hiện đảo ngược lại xâu chứa trong sum bằng cách tương tự bài 6 và lưu vào **rev_sum** rồi in ra màn hình



  - thực hiện tương tự với trường hợp num1_len < num2_len thì lần lượt push offset của **sum**, **rev_num1**, **rev_num2** làm tham số cho procedure **addNum2_1,** hàm **addNum2_1** thực hiện cộng khi **num1_len < num2_len**

  - kết quả thu được tương tự



- **windows:**
  - section .data: tương tự như linux
  - section .data?: tương tự như section .bss trong linux
  - section .code bao gồm code thực thi với entry point là main proc
    - luồng thực thi tương tự linux chỉ khác một số điểm:
      - procedure **ion1_proc** và **ion2_proc** sử dụng winapi **WriteConsole** và **ReadConsole** trong đó:
        - **WriteConsole** được sử dụng trong procedure **WriteString** với tham số là địa chỉ của chuỗi cần in, đầu procedure push tất cả các

tham số được sử dụng trong hàm vào stack để khôi phục sau khi hàm kết thúc. đầu tiên sử dụng api **lstrlen** để lấy ra độ dài của chuỗi đưa vào r15 và gọi api GetStdHandle vơi tham số rcx là -11(STD_OUTPUT_HANDLE) rồi gọi api **WirteConsole** với 4 tham số:

- rcx: nStdHandle
- rdx: offset chuỗi cần ghi
- r8: số lượng ký tự in(lấy theo độ dài chuỗi trả về từ **lstrlen**)
- r9: offset cho số lương ký tự in- lấy từ biến của procedure WriteString(rbp - 8)

- **ReadConsole** được sử dụng trong procedure **ReadString** với 2 tham số là số lượng ký tự đọc và địa chỉ của chuỗi cần đọc, đầu procedure push tất cả các tham số được sử dụng trong hàm vào stack để khôi phục sau khi hàm kết thúc. đầu tiên gọi api GetStdHandle vơi tham số rcx là -10(STD_INPUT_HANDLE) rồi gọi api **WirteConsole** với 4 tham số:
  - rcx: nStdHandle
  - rdx: offset chuỗi cần đọc
  - r8: số lượng ký tự in(lấy theo tham số thứ nhất rbp + 16)
  - r9: offset cho số lương ký tự đọc- lấy từ biến của procedure WriteString(rbp - 8).

- sử dụng winapi **lstrlen** thay vì procedure getlen tự định nghĩa
- xét xâu 22 ký tự do windows có ký tự kết thúc là \r \n
- kết quả thu được

```
D:\tranning\tranningvcs\tuan2\tuan2-windows\congsolon>bignum.exe
input num1: 129398127379812
num1 is: 129398127379812
input num2: 123434
num2 is: 123434
129398127503246
D:\tranning\tranningvcs\tuan2\tuan2-windows\congsolon>
```

```
D:\tranning\tranningvcs\tuan2\tuan2-windows\congsolon>bignum.exe
input num1: 1276378
num1 is: 1276378
input num2: 123789126378126378
num2 is: 123789126378126378
123789126379402756
D:\tranning\tranningvcs\tuan2\tuan2-windows\congsolon>
```