

BÁO CÁO THỰC HÀNH

LẬP TRÌNH HỆ THỐNG

Tên bài Thực hành: Lab 02 - Lớp: NT209.P12.ANTT

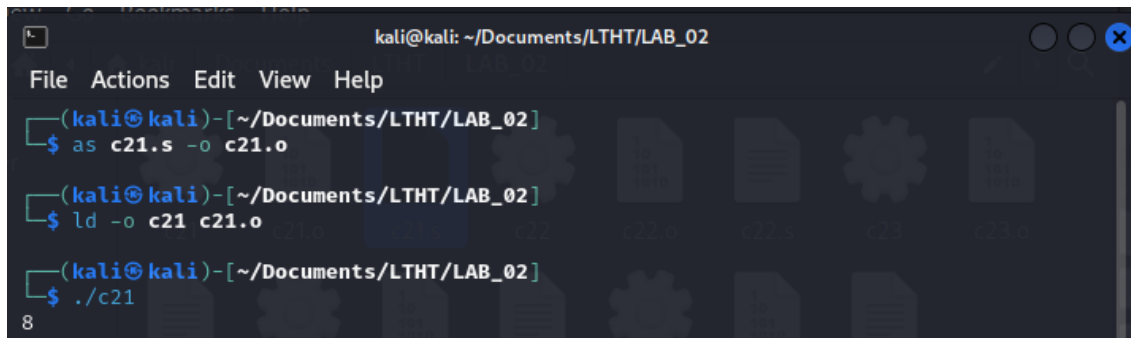
Giáo viên hướng dẫn: Đỗ Thị Thu Hiền

Họ và tên Sinh viên	MSSV
Nguyễn Trần Minh Khôi	23520780
Lê Đăng Khôi	23520766
Vương Thành Đạt	23520281

C 2.1: Chương trình in ra độ dài của một chuỗi ký tự cho trước (tối đa 9 ký tự)

- + Input: Chuỗi msg được khai báo sẵn trong .section .data, tối đa 9 ký tự.
- + Output: Xuất ra màn hình độ dài của chuỗi (số ký tự - không tính ký tự null).
- + Cài đặt chương trình (file .s):

```
1 .section .data
2 msg: .string "Love UIT"
3 msg_len = . -msg
4
5 .section .bss
6     .lcomm output 9
7
8 .section .text
9     .globl _start
10
11 _start:
12
13     movl $msg_len, %eax
14     addl $47, %eax
15
16     movl %eax, output
17
18     movl $4, %eax
19     movl $1, %ebx
20     movl $output, %ecx
21     movl $5, %edx
22     int $0x80
23
24     movl $1, %eax
25     int $0x80
```



```
kali@kali: ~/Documents/LTHT/LAB_02
File Actions Edit View Help
(kali@kali) - [~/Documents/LTHT/LAB_02]
$ as c21.s -o c21.o
(kali@kali) - [~/Documents/LTHT/LAB_02]
$ ld -o c21 c21.o
(kali@kali) - [~/Documents/LTHT/LAB_02]
$ ./c21
8
```

+ Giải thích ý tưởng:

+ Sử dụng “msg_len = . -msg” để lấy ra được độ dài của chuỗi “msg”. Nhưng msg_len đang là hằng số.

+ Đầu tiên, dùng lệnh movl để đưa giá trị của hằng số msg_len lên thanh ghi %eax. Theo lý thuyết, để chuyển từ ký tự ‘1’ sang số 1 thì cần phải cộng 48 giá trị theo bảng mã ascii. Tuy nhiên, chuỗi msg có ký tự ‘\n’ cuối cùng nên hằng số ‘msg_len’ sẽ dư ra một giá trị. Do đó, chỉ cần cộng thêm cho 47 giá trị để có thể in ra được độ dài của chuỗi ‘msg’ trong phần data.

+ Bước tiếp theo, sau khi đã cộng 47 giá trị vào thanh ghi %eax, chuyển giá trị đang có trong thanh ghi này lên vùng nhớ output để tiến hành in ra.

+ Dùng linux system call để in ra giá trị đang chứa trong vùng nhớ output.

+ Dùng thêm một linux system call để kết thúc chương trình (tránh segmentation-fault).

C.2.2. Chương trình in SBD (6 ký tự) từ MSSV (8 ký tự)

Yêu cầu:

Input: MSSV gồm 8 ký tự (định dạng của UIT, giới hạn MSSV từ 1552xxxx trở đi)

Output: Xuất ra màn hình **SBD** (là chuỗi con, 6 ký tự, bỏ 2 ký tự 3 và 4).

Yêu cầu: Sử dụng tối đa **02 lần** in ra màn hình để in kết quả.

```
D: > demo.s > asm c22.s
1  .section .data          #Định nghĩa đoạn dữ liệu nơi lưu trữ dữ liệu đã được khởi tạo.
2  .section .bss           #Định nghĩa đoạn bss nơi lưu trữ dữ liệu chưa được khởi tạo.
3      .lcomm input , 8    #Khai báo biến input có kích thước 8 byte
4  .section .text          #Định nghĩa đoạn text nơi lưu trữ mã máy đã được biên dịch.
5      .globl _start       #Định nghĩa một hàm toàn cục _start
6  _start:                #Hàm main
7      movl $3, %eax        #Gán giá trị 3 vào thanh ghi eax
8      movl $0, %ebx        #Gán giá trị 0 vào thanh ghi ebx
9      movl $input, %ecx    #Gán địa chỉ của biến input vào thanh ghi ecx
10     movl $9, %edx        #Gán giá trị 9 vào thanh ghi edx
11     int $0x80            #Gọi hàm sys_read
12
13     movl $input, %eax     #Gán địa chỉ của biến input vào thanh ghi eax
14     movb $0, 2(%eax)     #Gán giá trị 0 vào vị trí thứ 2 của biến input
15     movb $0, 3(%eax)     #Gán giá trị 0 vào vị trí thứ 3 của biến input
16
17     movl $4, %eax        #Gán giá trị 4 vào thanh ghi eax
18     movl $1, %ebx        #Gán giá trị 1 vào thanh ghi ebx
19     movl $input, %ecx    #Gán địa chỉ của biến input vào thanh ghi ecx
20     movl $9, %edx        #Gán giá trị 9 vào thanh ghi edx
21     int $0x80            #Gọi hàm sys_write
22
23     movl $1, %eax        #Gán giá trị 1 vào thanh ghi eax
24     int $0x80            #Gọi hàm sys_write
```

```
(kali@kali)-[~/NT209]
$ as -o c22.o c22.s

(kali@kali)-[~/NT209]
$ ld -o c22 c22.o

(kali@kali)-[~/NT209]
$ ./c22
23520281
230281

(kali@kali)-[~/NT209]
$ ./c22
20520281
200281
```

Ý tưởng:

- Chèn ký tự null vào bytes thứ 3 và 4 tương ứng với ký tự trong input, trước đó phải khai báo input có 9 bytes để thêm ký tự xuống hàng cho chương trình.

Giải thích:

- Cấp phát vùng nhớ cho input là **8 bytes** tương ứng với 8 ký tự đề yêu cầu trong đoạn bss.
- Ở dòng 7 – 11, ta đặt:
 - Syscall để read \$3 vào thanh ghi %eax.
 - \$0 được xem như chuẩn đầu vào (sdtin) vào thanh ghi %ebx.
 - Tải địa chỉ của input vào thanh ghi %ecx để lưu dữ liệu cần đọc.
 - Ta cần đọc 9 bytes vì tính thêm cả ký tự \n(endline).

- Kích hoạt ngắt phần mềm để thực hiện syscall read.
- Ở dòng 13 –15:
 - Tải địa chỉ input vào %eax.
 - Đặt byte ở vị trí thứ 3 của input thành 0, chèn ký tự 'null'..
 - Đặt byte ở vị trí thứ 4 của input thành 0, chèn ký tự 'null'
- Ở dòng 17 – 21:
 - Syscall để write \$4 vào thanh ghi %eax.
 - \$1 được xem như chuẩn đầu ra(stdout) vào thanh ghi %ebx.
 - Tải địa chỉ của input vào thanh ghi %ecx để chỉ bộ đệm ghi từ.
 - Ta cần ghi 9 bytes.
 - Kích hoạt ngắt phần mềm để thực hiện syscall write.
- Ở dòng 23 – 24:
 - Đặt số syscall để thực hiện exit() 1 vào thanh ghi %eax.
 - Kích hoạt ngắt phần mềm để thực hiện syscall exit và kết thúc chương trình.

C 2.3 Chương trình tính giá trị trung bình cộng của 4 số (1 chữ số)

+ Input: 4 số nguyên (1 chữ số và <10) nhập vào từ bàn phím.

+ Output: Giá trị trung bình cộng (lấy phần nguyên) của 4 số đã nhập.

+ Xây dựng chương trình:

```
.section .data
my_string : .string "Enter a number (0-9): "

.section .bss
.lcomm input1, 2
.lcomm input2, 2
.lcomm input3, 2
.lcomm input4, 2
.lcomm sum, 4      # Sử dụng 4 byte để lưu tổng
.lcomm output, 2   # Chỉ cần 2 byte để lưu một ký tự và kết thúc

.section .text
.globl _start

_start:
# Khởi tạo tổng bằng 0
movl $0, sum

# Nhập số thứ 1
movl $4, %eax      # sys_write
movl $1, %ebx      # stdout
movl $my_string, %ecx # Địa chỉ của my_string
movl $26, %edx     # Độ dài chuỗi "Enter a number (0-9): " là 22 byte
int $0x80

movl $3, %eax      # sys_read
movl $0, %ebx      # stdin
movl $input1, %ecx # Địa chỉ của input1
movl $2, %edx      # Chỉ đọc 2 byte (0-9 chữ số + newline)
int $0x80
```

```

# Chuyển đổi input1 từ ASCII sang số
movb input1, %al
subb '$0', %al
movzx %al, %eax
addl %eax, sum

# Nhập số thứ 2
movl $4, %eax
movl $1, %ebx
movl $my_string, %ecx
movl $26, %edx
int $0x80

movl $3, %eax
movl $0, %ebx
movl $input2, %ecx
movl $2, %edx
int $0x80

# Chuyển đổi input2 từ ASCII sang số
movb input2, %al
subb '$0', %al
movzx %al, %eax
addl %eax, sum

# Nhập số thứ 3
movl $4, %eax
movl $1, %ebx
movl $my_string, %ecx
movl $26, %edx
int $0x80

movl $3, %eax
movl $0, %ebx
movl $input3, %ecx
movl $2, %edx
int $0x80

# Chuyển đổi input3 từ ASCII sang số
movb input3, %al
subb '$0', %al
movzx %al, %eax
addl %eax, sum

```

```

# Nhập số thứ 4
movl $4, %eax
movl $1, %ebx
movl $my_string, %ecx
movl $26, %edx
int $0x80

movl $3, %eax
movl $0, %ebx
movl $input4, %ecx
movl $2, %edx
int $0x80

# Chuyển đổi input4 từ ASCII sang số
movb input4, %al
subb '$0', %al
movzx %al, %eax
addl %eax, sum

# Tính trung bình: sum / 4
movl sum, %eax
sarl $2, %eax      # Chia bằng 4 (dịch phải 2 bit)
movl %eax, sum     # Lưu lại kết quả trung bình

# Chuyển đổi sum từ số sang ASCII
addl '$0', %eax
movb %al, output   # Lưu ký tự ASCII vào output
movb $0x0A, output+1

# Ghi kết quả
movl $4, %eax      # sys_write
movl $1, %ebx      # stdout
movl $output, %ecx # Địa chỉ của output
movl $1, %edx      # Số byte cần ghi (chỉ 1 byte ký tự số)
int $0x80

# Kết thúc chương trình
movl $1, %eax      # sys_exit
xor %ebx, %ebx     # Trả về 0
int $0x80

```

```
kali@kali: ~/Documents/LTHT/LAB_02
File Actions Edit View Help
(kali@kali)-[~/Documents/LTHT/LAB_02]
$ as c23.s -o c23.o
(kali@kali)-[~/Documents/LTHT/LAB_02]
$ ld -o c23 c23.o
(kali@kali)-[~/Documents/LTHT/LAB_02]
$ ./c23
Enter a number (1-digit): 1
Enter a number (1-digit): 2
Enter a number (1-digit): 3
Enter a number (1-digit): 4
2
```

+ Giải thích ý tưởng:

+ Phần `.section .data` khai báo một string để thông báo nhập vào một số (Enter a number (1-digit))

+ Phần `.section .bss` khai báo vùng nhớ để lưu trữ 4 số nhập vào tương ứng input1 đến input4.

+ System call để in ra dòng thông báo 'my_string'

- Syscall để write \$4 vào thanh ghi %eax.
- \$1 được xem như chuẩn đầu ra(sdtout) vào thanh ghi %ebx.
- Tải địa chỉ của input vào thanh ghi %ecx để lưu dữ liệu cần đọc.
- Ta cần đọc 26 byte cho câu thông báo 'my_string'
- Kích hoạt ngắt phần mềm để thực hiện system call write.

+ System call để nhập vào một số (1-digit):

- Syscall để read \$3 vào thanh ghi %eax.
- \$0 được xem như chuẩn đầu vào (sdtin) vào thanh ghi %ebx.
- Tải địa chỉ của input1 vào thanh ghi %ecx để lưu dữ liệu cần đọc.
- Ta cần đọc 2 bytes vì tính thêm cả ký tự \n(endline)
- Kích hoạt ngắt phần mềm để thực hiện syscall read

+ Chuyển đổi giá trị nhập vào sang số:

- Di chuyển giá trị của từng input1 đến input4 lên thanh ghi %al
- Trừ nó cho giá trị ascii '0'.
- Cộng giá trị vào ô nhớ sum.
- `movzx` là viết tắt của Move with Zero Extend. Lệnh này di chuyển dữ liệu từ một thanh ghi nguồn sang một thanh ghi đích và mở rộng giá trị nguồn bằng cách thêm các bit 0 vào phần cao hơn của thanh ghi đích để đảm bảo rằng giá trị không bị thay đổi về mặt số học.

+ Lặp lại quá trình này cho đến khi nhập hết 4 input và tính xong giá trị sum.

+ Shift left giá trị bên trong ô nhớ sum 2 bit, tức là đang chia giá trị trong ô nhớ sum cho 4.

+ Chuyển đổi giá trị bên trong ô nhớ sum từ số sang ASCII lần cuối để in ra kết quả

+ System call để in ra giá trị trong ô nhớ sum

- Syscall để write \$4 vào thanh ghi %eax.
- \$1 được xem như chuẩn đầu ra(sdtout) vào thanh ghi %ebx.
- Tải địa chỉ của input vào thanh ghi %ecx để lưu dữ liệu cần đọc.
- Ta cần đọc 1 byte cho giá trị bên trong ô nhớ sum
- Kích hoạt ngắt phần mềm để thực hiện system call write.

+ Cuối cùng, kích hoạt system call exit để kết thúc chương trình.

C.2.4 Mã hóa Caesar cho chuỗi gồm 4 ký tự

Input: Chuỗi gồm 4 ký tự chữ in thường, số bước dịch n ($0 - 9$), trong đó với 1 ký tự x thì $'a' \leq x + n \leq 'z'$.

Output: Xuất ra chuỗi ký tự đã được mã hóa Caesar với bước dịch n .

Nâng cao: Xử lý được trường hợp xoay vòng khi kết quả sau khi dịch vượt quá ký tự 'z'.
Ví dụ $'z' + 1 = 'a'$, $'z' + 2 = 'b'$.

Ý tưởng:

Do phải xoay vòng ký tự khi mã hóa nên cần xác định chính xác vị trí bắt đầu xoay vòng.
Ta có:

- Vị trí xoay vòng là những vị trí lớn hơn giá trị của ký tự z(122) trong bảng mã Ascii.
- Khi xoay vòng, chúng ta quay ngược về ký tự a và tính bước xoay vòng như 1 bước dịch (từ z-->a), sau đó tiếp tục những bước dịch còn lại.
- Để thực hiện xoay vòng chúng ta có thể trừ giá trị sau khi dịch n bước cho 26 (26 ký tự chữ cái trong Ascii). Ta có ví dụ sau:
 - + Gọi i là vị trí ban đầu, n là bước dịch, j là vị trí sau khi dịch.
 - + Sau khi dịch m bước, giả sử j nằm ở vị trí z, $j = i + m = "z"(122) \rightarrow$ Bước dịch sau sẽ xoay vòng về a.
 - + Để quay về a, ta cộng 1 bước dịch vào $j = (i+m) + 1 = 123$, và quay về a: $j = ((i+m)+1) - 26 = 97$.
 - + Sau khi quay về a, ta còn $n-(m+1)$ bước dịch, giá trị cuối cùng của j là:
$$j = ((i+m)+1) - 26 + n - (m+1)$$
$$= i+m+1-26+n-m-1$$
$$= i+n-26$$
$$= (i+n)-26$$
- Kết luận: để thực hiện mã hóa, ta cộng n bước dịch vào vị trí ban đầu (i+n), sau đó xét xem vị trí đó có vượt qua vị trí cuối (z=122) hay chưa, nếu đã vượt qua, thực hiện xoay vòng bằng cách trừ vị trí đó cho 26 ((i+n)-26) và thu được vị trí cuối cùng.

Ngoài ra cũng cần lưu ý thêm 1 số bước như tách chuỗi thành từng kí tự riêng biệt để xử lí, chuyển từng kí tự đã mã hóa vào chuỗi ban đầu và trừ bước dịch n cho 48, do kí tự ban đầu được nhập vào là kí tự Ascii, không phải giá trị đại số mà ta mong muốn, để chuyển số từ dạng kí tự về dạng giá trị đại số, ta trừ cho 48 và lưu lại để sử dụng sau.

Cách thực hiện và giải thích chi tiết trong ảnh sau:

```
1.section .data
2.section .bss
3    .lcomm output, 5
4    .lcomm n, 1
5    .lcomm a, 1
6
7.section .text
8    .globl _start
9_start:
10   # Nhập chuỗi 4 kí tự từ bàn phím
11   movl $3, %eax
12   movl $0, %ebx
13   movl $output, %ecx
14   movl $5, %edx
15   int $0x80
16
17   # Nhập bước dịch n
18   movl $3, %eax
19   movl $0, %ebx
20   movl $n, %ecx
21   movl $1, %edx
22   int $0x80
23
24   # Tách chuỗi kí tự vào các thanh ghi
25   movl $output, %eax
26   movb 0(%eax), %bl
27   movb 1(%eax), %cl
28   movb 2(%eax), %dl
29   movb 3(%eax), %sil
30
31   # Chuyển n từ mã ascii thành giá trị n
32   movb n, %al
33   subb $48, %al
34
35   # Mã hóa chuỗi với bước dịch n
36   addb %al, %bl
37   addb %al, %cl
38   addb %al, %dl
39   addb %al, %sil
40
41   # Kiểm tra kí tự vượt qua "z" thì xoay vòng
42   cmpb $122, %bl
43   jle check_cl
44   subb $26, %bl
45
46 check_cl:
47   cmpb $122, %cl
48   jle check_dl
49   subb $26, %cl
50
51 check_dl:
52   cmpb $122, %dl
53   jle check_sil
54   subb $26, %dl
55
56 check_sil:
57   cmpb $122, %sil
58   jle continue
59   subb $26, %sil
60
61 continue:
62   # Chuyển kí tự đã mã hóa vào chuỗi ban đầu
63   movl $output, %eax
64   movb %bl, 0(%eax)
65   movb %cl, 1(%eax)
66   movb %dl, 2(%eax)
67   movb %sil, 3(%eax)
68
69   # In chuỗi đã mã hóa
70   movl $4, %eax
71   movl $1, %ebx
72   movl $output, %ecx
73   movl $4, %edx
74   int $0x80
75
76   # Thoát
77   movl $1, %eax
78   int $0x80
79
```

Trong chương trình trên, từng kí tự trong chuỗi được tách ra là lưu trữ trong các thanh ghi có kích thước 1 byte.

- Ở bước kiểm tra điều kiện xoay vòng và tính giá trị xoay vòng, 3 câu lệnh assembly có chức năng tương tự đoạn lệnh c++ sau: `if (j>122) j-=26`
- Do ngôn ngữ assembly chỉ thực hiện tuần tự các câu lệnh từ trên xuống nên ta phải sử dụng lệnh `jump` để làm phát sinh 2 trường hợp. Lệnh `jump` có 2 dạng:
- Vô điều kiện (`jmp`): luôn nhảy đến nhãn được gắn phía sau.
- Có điều kiện (`je, jl, jg, jle, jle, ...`): chỉ nhảy đến nhãn khi những điều kiện của lệnh so sánh gần nhất phía trên thỏa những yêu cầu của lệnh `jump`, nếu không thực hiện lệnh tiếp theo.
- Để cung cấp kết quả so sánh, ta có lệnh `cmp` để so sánh 2 giá trị, lệnh `cmp` sẽ không làm thay đổi giá trị của cả thanh ghi `src` và `dest`, nó chỉ tác động đến giá trị của thanh ghi `flag` bằng cách lật hay không lật các trường bit bên trong thanh ghi này. Sau đó các lệnh `jump` có điều kiện sẽ dựa vào thanh ghi `flag` này để quyết định có nên nhảy hay không.

VD: ở nhãn `check_cl` trong chương trình phía trên, giả sử giá trị được lưu bên trong thanh ghi `%cl` là 125, khi đó, hàm `cmp` sẽ so sánh `%cl` với `$122` và thu được kết quả lớn hơn và ghi vào `flag`. Sau đó lệnh `jle` sẽ kiểm tra `flag`, nếu `flag` nói rằng kết quả là nhỏ hơn hoặc bằng, lệnh nhảy sẽ được kích hoạt và nhảy đến nhãn `check_dl`, tuy nhiên kết quả là lớn hơn, do đó lệnh `jle` sẽ không được thực thi và thực hiện phép trừ bên dưới. Tương tự với 3 thanh ghi chứa kí tự còn lại.

Kết quả thực thi:

```

kali@kali: ~/NT209
File Actions Edit View Help
(kali@kali)-[~/NT209]
$ ./c24
abcd
1
bcde

(kali@kali)-[~/NT209]
$
(kali@kali)-[~/NT209]
$ ./c24
abcd
5
fghi

(kali@kali)-[~/NT209]
$
(kali@kali)-[~/NT209]
$ ./c24
abyz
1
bcza

(kali@kali)-[~/NT209]
$
(kali@kali)-[~/NT209]
$ ./c24
name
9
wjvn

```