3

Lab

Lập trình Assembly cơ bản (tiếp theo)

Basic Assembly Programming (cont)

Thực hành Lập trình Hệ thống

Lưu hành nội bộ

A. TỔNG QUAN

A.1 Mục tiêu

- Tìm hiểu và làm quen với Hợp ngữ (Assembly Language - ASM)
- Tìm hiểu quá trình dịch từ một mã nguồn assembly thành tập tin thực thi
- Thực hành viết một số chương trình mẫu dưới dạng hợp ngữ (theo AT&T)

A.2 Môi trường

- Máy cài Hệ điều hành Linux (máy ảo).
- Các công cụ biên dịch, hợp dịch, liên kết gồm **gcc**, **as**, **ld**.

A.3 Liên quan

- Sinh viên cần vận dụng kiến thức trong Buổi 3 – 4 – 5 (Lý thuyết).
- Các kiến thức này đã được giới thiệu trong nội dung lý thuyết đã học do đó sẽ không được trình bày lại trong nội dung thực hành này.
- Tham khảo tài liệu (Mục F).

B. KIẾN THỨC NỀN TẢNG

B.1 Cấu trúc cơ bản và các thành phần của một chương trình hợp ngữ

B.1.1 Cấu trúc của chương trình hợp ngữ

Trong Linux, các chương trình viết bằng mã assembly sẽ có đuôi **.s**, gồm các section:

B.1.1.1 Section .data

Là section khai báo những **vùng nhớ** hoặc **hằng số**, nơi chứa các dữ liệu dùng cho chương trình. Thường sử dụng để khai báo biến đã được khởi tạo giá trị hoặc các hằng số sử dụng cho chương trình. Định dạng khai báo:

```
.section .data
<name1>:  .<type> <data1>     # vùng nhớ chứa dữ liệu
<name2> = <data2>             # hằng số
...
```

Trong đó:

+ **name** là tên gợi nhớ, có thể được dùng ở các lệnh để truy xuất vùng nhớ hoặc hằng số sau này.
+ **type** là kiểu dữ liệu, ví dụ: int, string, byte...
+ **data** là giá trị cụ thể cần khởi tạo cho vùng nhớ trong vùng nhớ.

Ví dụ: đoạn mã khai báo vùng nhớ **a** lưu số nguyên **10**, vùng nhớ **my_str** lưu chuỗi **"Hello"** và 1 hằng số **b** có giá trị 9.

```
.section .data
a:      .int 10
my_str: .string "Hello"
b = 9
```

B.1.1.2 Section .bss

Phần .bss khai báo các dữ liệu chưa được gán giá trị (hoặc null), thường được dùng khi cần chuẩn bị sẵn các **vùng nhớ** trống để lưu dữ liệu sau này. Đây là thành phần không bắt buộc, nếu không sử dụng có thể bỏ qua. Định dạng khai báo:

```
.section .bss
      .lcomm <name1>, <length>
      .lcomm <name2>, <length>
      ...
```

Trong đó:

+ **name** là tên gợi nhớ, có thể được dùng ở các lệnh để truy xuất vùng nhớ này.
+ **length** là kích thước vùng nhớ cần được cấp, tính theo byte.

Ví dụ: đoạn mã bên dưới khai báo vùng nhớ tên **mem** gồm 2 bytes và vùng nhớ **result** có kích thước 4 bytes.

```
.section .bss
      .lcomm mem, 2
      .lcomm result, 4
```

B.1.1.3 Section .text

Là section **bắt buộc** phải có trong tất cả các chương trình hợp ngữ. Đây là nơi các câu lệnh được khai báo. Định dạng khai báo:

```
.section .text
      .globl _start
_start:
      // các lệnh assembly
```

Sau nhãn _start, các câu lệnh assembly sẽ được viết tùy thuộc vào chức năng cần lập trình. Trong đó có thể sử dụng các vùng nhớ đã được khai báo ở 2 section .data hoặc .bss để lấy hoặc lưu trữ dữ liệu vào các vùng nhớ nhất định, hoặc tương tác với các thanh ghi.

Với 1 vùng nhớ tên **a** được khai báo trong section .data (hoặc .bss):

- Ký hiệu **$a** tương ứng với việc lấy *địa chỉ* của vùng nhớ **a,** được xem là 1 hằng số.
- Ký hiệu **(a)** hoặc **a** tương ứng với việc truy xuất *giá trị* đang lưu tại vùng nhớ **a**.

B.1.2 Định dạng lệnh assembly

Trong phạm vi bài thực hành này, các lệnh assembly được viết theo định dạng AT&T và tập lệnh của kiến trúc 32bit.

B.2 Linux System Call (Hàm gọi hệ thống Linux)

Các system call yêu cầu đầu vào sao cho các giá trị được đặt trong thanh ghi. Có một thứ tự cụ thể trong đó mỗi giá trị đầu vào được đặt trong các thanh ghi tương ứng. Đặt giá trị đầu vào trong một thanh ghi không đúng chuẩn có thể tạo ra kết quả sai.

Cụ thể, các giá trị đầu vào sẽ được gán cho các thanh ghi sau:

- %eax: Giá trị của hàm gọi hệ thống (tham khảo bảng bên dưới)
- %ebx: Bộ mô tả tệp (số nguyên)
- %ecx: Con trỏ (địa chỉ bộ nhớ) của chuỗi cần hiển thị
- %edx: Kích thước của chuỗi cần hiển thị

%eax	Name	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	int	-	-	-	-
2	sys_fork	struct pt_regs	-	-	-	-
3	sys_read	unsigned int	char *	size_t	-	-
4	sys_write	unsigned int	const char *	size_t	-	-
5	sys_open	const char *	int	int	-	-
6	sys_close	unsigned int	-	-	-	-

Giá trị mô tả tệp cho vị trí đầu ra (hoặc vào) để chỉ định vị trí nhận hoặc xuất dữ liệu được đặt trong %ebx. Các hệ thống Linux chứa ba mô tả tệp đặc biệt:

- 0 (STDIN): Đầu vào tiêu chuẩn cho thiết bị đầu cuối (thường là bàn phím)
- 1 (STDOUT): Đầu ra tiêu chuẩn cho thiết bị đầu cuối (thường là màn hình đầu cuối)
- 2 (STDERR): Đầu ra lỗi tiêu chuẩn cho thiết bị đầu cuối (thường là màn hình đầu cuối)

Ví dụ để gọi hàm in ra màn hình chuỗi "Hello, World".

- **%eax = 4** tương ứng với chỉ thị in chuỗi.
- **%ebx = 1** tương ứng với vị trí in ra là stdout (terminal).
- **%ecx = địa chỉ ô nhớ** đang chứa chuỗi cần in.
- **%edx = độ dài** (tính theo byte) của chuỗi cần in.

Sau khi khai báo các giá trị tham số cần thiết của system call trong các thanh ghi, lệnh **int $0x80** được dùng để thực thi system call đó.

B.3 Các câu lệnh rẽ nhánh: nhảy không điều kiện và có điều kiện

Việc thực thi một số đoạn mã của chương trình dựa trên điều kiện có thể được thực hiện thông qua các câu lệnh rẽ nhánh. Những câu lệnh này sẽ thay đổi luồng hoạt động của chương trình để thực thi những đoạn mã mong muốn. Trong bài thực hành này, chúng ta xem xét các câu lệnh nhảy không điều kiện và có điều kiện.

- **Câu lệnh jump không có điều kiện**

Câu lệnh thực hiện: ***JMP label***

Câu lệnh JMP kèm theo một label trỏ đến vị trí đoạn mã cần nhảy đến. Khi gặp câu lệnh này, chương trình lập tức chuyển hướng đến vị trí cần nhảy.

Ví dụ: Đoạn mã khi thực thi câu lệnh **jmp** (dòng 5) sẽ nhảy đến label ***overhere*** ở dòng 7, bỏ qua lệnh **movl** ở dòng 6.

```
1 .section .text
2     .globl _start
3
4 _start:
5     jmp overhere
6     movl $10, %ebx
7 overhere:
8     movl $20, %ebx
```

- **Câu lệnh jump có điều kiện**

Việc nhảy dựa trên một điều kiện nhất định có thể thực hiện với các lệnh nhảy có điều kiện. Các lệnh này chỉ thực hiện nhảy khi một điều kiện đánh giá nào đó được thỏa mãn. Chúng thường được kết hợp với các câu lệnh so sánh (cmp) các giá trị để nhảy khi kết quả so sánh thỏa mãn điều kiện nào đó.

```
cmpl src2, src1
jX label
```

Một số câu lệnh jump thông dụng:

jX	Điều kiện nhảy
je	src1 = src2
jne	src1 != src2

jg	src1 > src2
jge	src1 ≥ src2
jl	src1 < src2
jle	src1 ≤ src2

Ví dụ: Thực hiện so sánh 2 số num1 và num2 với lệnh **cmpl**. Với lệnh **je** ngay sau đó, chương trình sẽ sử dụng kết quả so sánh của lệnh cmpl làm điều kiện nhảy, và sẽ nhảy đến label **_is_equal** nếu 2 số bằng nhau.

```
1  .section .data
2  num1:
3      .int 1
4  num2:
5      .int 2
6
7  .section .text
8      .globl _start
9  _start:
10     movl (num1), %eax
11     movl (num2), %ebx
12     cmpl %ebx, %eax
13     je _is_equal
14
15 _is_equal:
16     ...
```

C. THỰC HÀNH

Hướng dẫn chung: Sinh viên thực hiện viết các chương trình dưới dạng hợp ngữ (assembly) trong các file **.s**, sau đó thực hiện các bước hợp dịch và liên kết như sau:

```
as -o <file .o> <file .s đầu vào>
ld -o <file thực thi> <file .o đầu vào>
./<file thực thi đầu ra>
```

Ví dụ:

```
ubuntu@ubuntu: ~/LTHT/Lab2
ubuntu@ubuntu:~/LTHT/Lab2$ as -o example.o example.s
ubuntu@ubuntu:~/LTHT/Lab2$ ld -o example example.o
ubuntu@ubuntu:~/LTHT/Lab2$ ./example
Hello, World
ubuntu@ubuntu:~/LTHT/Lab2$
```

C.1 Chương trình kiểm tra tính đối xứng của các chữ số trong số có 5 chữ số

Input: Một số nguyên a có 5 chữ số (10000 ≤ a ≤ 99999)

Output: Xuất ra màn hình nhận định: "**Doi xung**" nếu các chữ số trong a đối xứng, ngược lại xuất ra "**Khong doi xung**"

Ví dụ:

```
ubuntu@ubuntu: ~/LTHT/Lab3/2024/V22/C1
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C1$ as -o C1.o C1.s
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C1$ ld -o C1 C1.o
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C1$ ./C1
Enter a number (5-digits): 12345
Khong doi xung
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C1$ ./C1
Enter a number (5-digits): 12321
Doi xung
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C1$ ./C1
Enter a number (5-digits): 11611
Doi xung
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C1$
```

Gợi ý 1.1: Dựa vào địa chỉ của chuỗi số để truy xuất các ký tự số dựa trên chỉ số của chúng. Ví dụ bên dưới đoạn mã lấy ký tự thứ 2 của chuỗi **input**.

```
    movl $input, %eax
    mov  1(%eax), %bl
```

C.2 Đếm số từ trong một chuỗi 10 chữ cái

Input: Nhập 1 chuỗi có 10 ký tự

Output: Xuất ra số từ có trong chuỗi.

Nâng cao: Xử lý chuỗi có độ dài không quá 255 ký tự và số từ <10, trong đó có thể có nhiều dấu cách bị lặp.

Optional (cộng điểm): Giới hạn 255 ký tự nhưng không giới hạn số từ trong câu.

Ví dụ: Xử lý chuỗi 10 ký tự

```
ubuntu@ubuntu: ~/LTHT/Lab3/2024/V22/C2
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ as -o C2.o C2.s
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ ld -o C2 C2.o
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ ./C2
Enter a string (10 chars): I am Thien
Number of words: 3
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ ./C2
Enter a string (10 chars): Class ANTT
Number of words: 2
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$
```

Phần nâng cao:

```
ubuntu@ubuntu: ~/LTHT/Lab3/2024/V22/C2
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ as -o C2.o C2.s
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ ld -o C2 C2.o
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ ./C2
Enter a string (<255 chars): Hello   antt,  check  this sentence.
Number of words: 5
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$
```

Optional (cộng điểm):

```
ubuntu@ubuntu: ~/LTHT/Lab3/2024/V22/C2
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ as -o C2.o C2.s
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ ld -o C2 C2.o
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ ./C2
Enter a string (<255 chars): hello antt
Number of words: 2
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$ ./C2
Enter a string (<255 chars): This sentence is for the optional part, can  you ch
eck?
Number of words: 10
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C2$
```

Gợi ý 2.1: Từ thường cách nhau bằng dấu cách. Xem bảng mã ASCII để xem cách kiểm tra một ký tự có phải dấu cách hay không.

Gợi ý 2.2 (cho phần nâng cao): Khi nhập chuỗi, ta thường gõ xuống dòng để kết thúc chuỗi và hệ thống tương ứng sẽ nhận và lưu 1 ký tự ***newline***. Do vậy, ký tự này có thể được dùng làm dấu hiệu để nhận biết vị trí kết thúc của chuỗi đã nhập.

C.3 Chương trình tìm số lớn nhất trong 5 số có 1 chữ số

Input: Nhập 5 số nguyên có 1 chữ số ($0 \leq a \leq 9$)

Output: Xuất ra màn hình số lớn nhất trong 5 số trên.

Ví dụ:

```
ubuntu@ubuntu: ~/LTHT/Lab3/2024/V22/C3
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C3$ as -o C3.o C3.s
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C3$ ld -o C3 C3.o
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C3$ ./C3
Enter a number (1-digit): 5
Enter a number (1-digit): 2
Enter a number (1-digit): 6
Enter a number (1-digit): 7
Enter a number (1-digit): 4
Max number: 7
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C3$
```

Gợi ý 3.1: Khi viết nhiều system call read liên tục để đọc nhiều input, cần khai báo độ dài của dữ liệu nhận vào (giá trị %edx) lớn hơn 1 để tránh ký tự xuống dòng của lần nhập trước ảnh hưởng đến lần nhập sau. Ví dụ: để nhập số có 1 chữ số, cần khai báo %edx = 2.

Gợi ý 3.2: Dữ liệu nhập vào là chuỗi ký tự số, nên chuyển từ ký tự số sang số nguyên để kiểm tra chính xác (ví dụ: chuyển từ '1' sang 1, chuyển '20' sang số 20, *xem phần **D.2***).

C.4 Kiểm tra năm nhuận

Input: Nhập một năm a có 4 chữ số ($1000 \leq a \leq 9999$)

Output: Xuất ra màn hình nhận định: "Nam nhuan" nếu **a là năm nhuận**, ngược lại thì xuất "Khong phai nam nhuan".

Nâng cao: In ra thêm năm nhuận kế tiếp.

Ví dụ:

```
ubuntu@ubuntu: ~/LTHT/Lab3/2024/V22/C4
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$ as -o C4.o C4.s
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$ ld -o C4 C4.o
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$ ./C4
Enter a year (4-digits): 1996
Nam nhuan
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$ ./C4
Enter a year (4-digits): 2005
Khong phai nam nhuan
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$
```

Phần nâng cao: In ra thêm năm nhuận kế tiếp

```
ubuntu@ubuntu: ~/LTHT/Lab3/2024/V22/C4
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$ as -o C4.o C4.s
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$ ld -o C4 C4.o
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$ ./C4
Enter a year (4-digits): 1996
Nam nhuan
Nam nhuan ke tiep: 2000
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$ ./C4
Enter a year (4-digits): 2099
Khong phai nam nhuan
Nam nhuan ke tiep: 2104
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$ ./C4
Enter a year (4-digits): 9999
Khong phai nam nhuan
Nam nhuan ke tiep: 10000
ubuntu@ubuntu:~/LTHT/Lab3/2024/V22/C4$
```

Gợi ý 4.1: Phép chia

Với phép chia **div**, ví dụ bên dưới tính phép chia 15/10:

```
xor %edx, %edx      // clear giá trị thanh ghi
movl $15, %eax      // Thanh ghi eax chứa giá trị số bị chia (15)
movl $10, %ecx      // Thanh ghi ecx chứa giá trị số chia (10)
div %ecx            // Tính phép chia %eax/%ecx (15/10)
                    // Kết quả: %eax chứa kết quả (1), %edx chứa số dư phép chia (15 % 10) = 5
```

Gợi ý 4.2: Xét năm nhuận

Năm nhuận là năm chia hết cho 4, không chia hết cho 100 hoặc chia hết cho 400

Ví dụ:

Đoạn mã bên dưới in ra số có 2 chữ số, sử dụng lệnh **div**. Trong đó:

- Thanh ghi ***%eax*** sẽ chứa giá trị số bị chia, là 1 số có 2 chữ số (dòng màu xanh).
- Chuỗi được tạo từ số sẽ lưu trong ô nhớ **number_str** (khai báo trong .bss với kích thước 2 byte) (màu đỏ) và sau đó được in ra

Sinh viên có điều chỉnh giá trị được truyền vào %eax và tên các biến cho phù hợp.

```
_print_2_digit_number:
    // remove current value of %edx
    xor %edx, %edx

    // pass dividend to %eax
    movl (sum), %eax
```

```
    // pass divisor to %ebx (= 10)
    movl $10, %ebx
    div %ebx

    // remainder in %edx, result in %eax
    addl $'0, %edx
    addl $'0, %eax

    // save number to memory for printing
    movl $number_str, %ebx
    movl %eax, (%ebx)
    movl %edx, 1(%ebx)

    // print number
    movl $4, %eax
    movl $1, %ebx
    movl $number_str, %ecx
    movl $2, %edx
    int $0x80
```

D. THAM KHẢO

D.1 Một số lưu ý khi lập trình assembly định dạng AT&T

- **Cần phân biệt được các toán hạng:**

Khác biệt trong ký hiệu của các dạng toán dạng khác nhau:

  - Hằng số: **$1**
  - Thanh ghi: **%eax**
  - Địa chỉ ô nhớ: **$output** với output là nhãn của ô nhớ trong .data hoặc .bss
  - Giá trị trong ô nhớ: **(output)** hoặc **output**

- **Trong các câu lệnh assembly, cần đảm bảo:**
  - Có **tối đa 1 toán hạng là hằng số**, nếu có hằng số cần đứng ở vị trí src.
    Ví dụ: addl $1, %eax
  - Có **tối đa 1 toán hạng liên quan đến truy xuất ô nhớ**.

- **Cách viết biểu thức địa chỉ để truy xuất ô nhớ**

Giả sử với ô nhớ có nhãn tên **output**

  - Truy xuất giá trị ô nhớ từ địa chỉ bắt đầu: viết dưới dạng **(output)** hoặc **output**.
  - Truy xuất giá trị ô nhớ từ địa chỉ cách n bytes: cần đưa địa chỉ vào thanh ghi và dùng biểu thức tính toán địa chỉ

```
    movl $output, %eax
    movl 1(%eax), %ebx    // truy xuất từ địa chỉ cách 1 byte
```

D.2 Bảng mã ASCII

- **Chuyển đổi số có 1 chữ số**

Sử dụng bảng mã ASCII để chuyển đổi từ số (decimal) sang chữ (dạng ascii) và ngược lại. Tham khảo tại https://www.ascii-code.com/. Ví dụ: Có thể chuyển đổi từ số 5 sang ký tự '5' bằng cách cộng **$48** (ký tự '0'). Ngược lại, để chuyển từ ký tự sang số thì trừ ký tự '0'.

- **Chuyển đổi số có nhiều chữ số**

Tách riêng từng số (hay ký tự số) sau đó áp dụng cách chuyển đổi trên số có 1 chữ số để chuyển qua lại giữa số và ký tự số tương ứng của nó.

Ví dụ: Chuyển chuỗi ký tự số '123' sang số nguyên: 1x100 + 2x10 + 3 = 123.

D.3 Lấy độ dài chuỗi trong section .data

Như đã trình bày ở trên, trong section .data có thể dùng để khai báo một số biến đã gán trước giá trị hoặc hằng số được dùng trong chương trình, ví dụ chuỗi output sẽ in ra màn hình. Tuy nhiên, thay vì gán cứng độ dài các chuỗi này khi dùng trong các system call, đoạn khai báo bên dưới cho phép lấy giá trị độ dài của chuỗi **rs**, lưu ý dù nằm trong .data nhưng **rs_len** là hằng số.

```
.section .data:
rs:  .string "Max is "
rs_len = . -rs
```

E. YÊU CẦU & ĐÁNH GIÁ

Sinh viên thực hành theo **nhóm tối đa 3 sinh viên**, có thể nộp bài theo 2 hình thức:

- Nộp trực tiếp trên lớp: báo cáo và demo kết quả với GVTH.
- Nộp file code tại website môn học theo thời gian quy định, *có chú thích chức năng của các đoạn code.*

Tạo **thư mục** tên **Lab3-NhomX-MSSV1-MSSV2-MSSV3,** chứa **các file .s** với định dạng:

Lab3-NhomX-<yêu cầu>.s

Ví dụ*: Lab3-Nhom2-C11.s*

F. THAM KHẢO

[1] Linux assemblers: A comparison of GAS and NASM [Online] Available at:
https://www.ibm.com/developerworks/library/l-gas-nasm/index.html

[2] Randal E. Bryant, David R. O'Hallaron (2011). *Computer System: A Programmer's Perspective (CSAPP)*

[3] Richard Blum (2005). *Professional Assembly Language*

HẾT