



2

Lab

Tìm hiểu ngôn ngữ Assembly

Introduction to Assembly Language

Thực hành Lập trình Hệ thống

Học kỳ II – Năm học 2020 - 2021

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 (ngôn ngữ bậc cao) 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 **as**, **ld**.

A.3 Liên quan

- Sinh viên cần vận dụng kiến thức trong Chương 3 (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 E).

B. KIẾN THỨC NỀN TẢNG

B.1 Hợp ngữ là gì?

Hợp ngữ - Assembly Language (hay viết tắt là ASM) là ngôn ngữ bậc thấp, và nằm ở vị trí trung gian bên cạnh ngôn ngữ máy và ngôn ngữ bậc cao. ASM sử dụng các từ gợi nhớ (mnemonics) để viết các chỉ thị (instructions) lập trình cho máy tính thay vì bằng những dãy số 0 và 1 (ngôn ngữ máy).

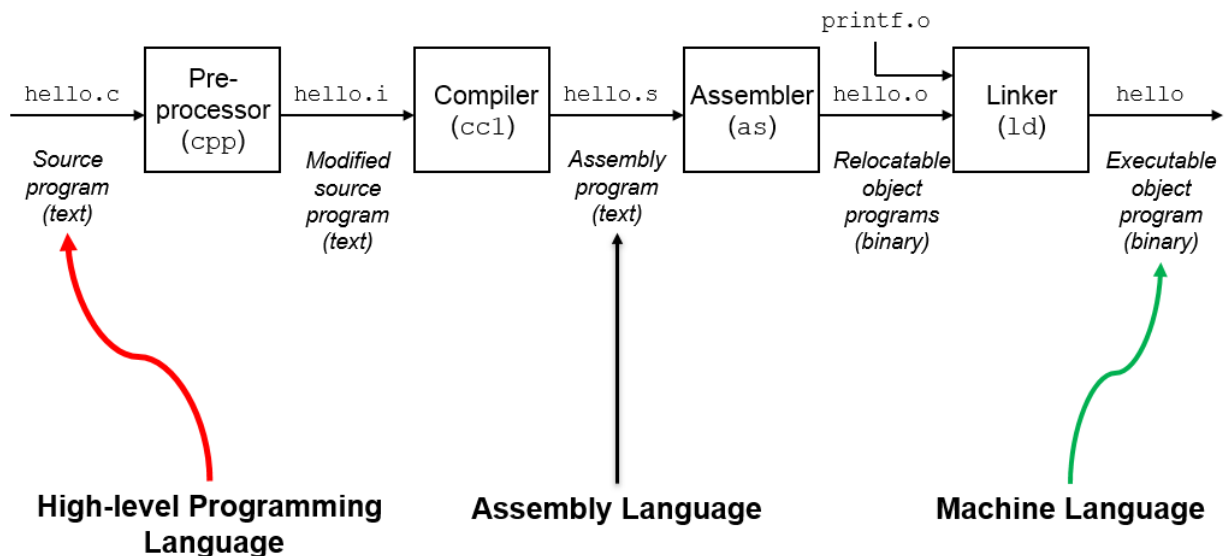
```
push %ebp
mov  %esp, %ebp
sub  $0x8, %esp
movl $0x1, -4(%ebp)
sub  $0xc, %esp
push $0x0
call 8048348
```

Hình 1. Ví dụ về một đoạn hợp ngữ

Về **ưu điểm**, Hợp ngữ thân thiện hơn so với ngôn ngữ máy – vốn rất khó hiểu, dễ gây lỗi và tốn nhiều thời giờ. Hợp ngữ có thể tương tác rất sâu dưới hệ thống, chúng có thể giao tiếp trực tiếp với các phần cứng và bắt chúng hoạt động theo ý người lập trình. Bên cạnh đó, hợp ngữ cũng rất hữu ích trong kỹ thuật dịch ngược (reverse engineering) – được sử dụng rộng rãi trong lĩnh vực an toàn thông tin.

Về **nhược điểm**, so với các ngôn ngữ bậc cao quen thuộc (C, C++, Java, Python ...), hợp ngữ ít thân thiện hơn, và phụ thuộc vào kiến trúc CPU (ARM, x86-32, x86-64), hệ điều hành (Linux, Windows, Mac) và các tập chỉ thị mà nhà sản xuất phần cứng đưa ra. Nói cách khác,

ASM là một ngôn ngữ, nhưng thực tế không giống như các ngôn ngữ lập trình khác, không có một định dạng chuẩn nào cho các trình hợp dịch (Assembler) sử dụng để dịch các chương trình ASM.



Hình 2. Quá trình dịch từ 1 chương trình dạng C sang tập tin thực thi

Một chương trình viết bằng ngôn ngữ bậc cao hay hợp ngữ không thể được thực thi trực tiếp bởi máy tính. Sau khi được viết xong, chương trình này phải trải qua quá trình dịch thành ngôn ngữ máy. Quá trình này được mô tả trong **Hình 2** bao gồm các giai đoạn:

- Giai đoạn tiền xử lý (Pre-processor)
- Giai đoạn dịch từ ngôn ngữ bậc cao sang hợp ngữ (Compiler)
- Giai đoạn dịch hợp ngữ sang ngôn ngữ máy (Assembler)
- Giai đoạn liên kết (Linker)

Vì vậy, bước đầu tiên để học và viết hợp ngữ là xác định môi trường sử dụng của nó.

Trong bài thực hành này, sinh viên được yêu cầu viết hợp ngữ theo cấu trúc của AT&T (sử dụng Hệ điều hành Linux, các công cụ biên dịch, hợp dịch như gcc, as, ld)

B.2 Cấu trúc cơ bản và các thành phần của một chương trình hợp ngữ

Thông thường chương trình hợp ngữ có ba phần (section) được khai báo:

- **Text** section: là **bắt buộc** trong tất cả các chương trình hợp ngữ. Đây là nơi mã lệnh được khai báo.
- **Data** section: được sử dụng để khai báo vùng nhớ nơi các thành phần dữ liệu được lưu trữ cho chương trình. Thường sử dụng để khai báo biến, hằng sử dụng cho chương trình.
- **Bss** section: Phần bss khai báo các phần tử dữ liệu chưa được gán giá trị (hoặc null). Đây là thành phần không bắt buộc, nếu không sử dụng có thể bỏ qua.

```
.section .data
output:
    .string "Hello, World"
.section .text
    .globl _start
_start:
    movl $13, %edx
    movl $output, %ecx
    movl $1, %ebx
    movl $4, %eax
    int $0x80
    movl $1, %eax
    int $0x80
```

Hình 3. Chương trình Hello World dưới dạng ASM

Trong Linux, để tiến hành biên dịch và thực thi một chương trình dưới dạng ASM (file .s), ta sử dụng các công cụ as, ld.

Ví dụ như bên dưới, sau khi trải qua quá trình hợp dịch và liên kết, ta có được file thực thi **example**. Chạy chương trình như hình bên dưới, ta thu được kết quả như sau:

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

B.3 Linux System Call (Hàm gọi hệ thống Linux)

Có nhiều lời gọi hệ thống (system call) được cung cấp bởi kernel Linux và biết cách tìm cũng như sử dụng chúng rất có lợi trong việc lập trình hợp ngữ (assembly language). Những system call này có sẵn cho các lập trình viên sử dụng. Thông thường, với mỗi lần phát hành một kernel mới, các system call mới được thêm vào danh sách.

Các system call thường được định nghĩa trong file sau:

`/usr/include/asm/unistd.h` hoặc `/usr/include/asm-generic/unistd.h`

Không giống như các hàm kiểu C, trong đó các giá trị đầu vào được đặt trên stack, 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.

Thứ tự trong đó các system call mong đợi các giá trị đầu vào như sau:

- %eax

- %ebx (tham số đầu tiên)

- %ecx (tham số thứ hai)
- %edx (tham số thứ ba)
- %esi (tham số thứ tư)
- %edi (tham số thứ năm)

Sử dụng quy ước này, 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) 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 (thông thường là bàn phím)
- 1 (STDOUT): Đầu ra tiêu chuẩn cho thiết bị đầu cuối (thông thường là màn hình đầu cuối)
- 2 (STDERR): Đầu ra lỗi tiêu chuẩn cho thiết bị đầu cuối (thông thường là màn hình đầu cuối)

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

%eax: 4 (system call number)
 %ebx: 1 (file descriptor - STDOUT)
 %ecx: message to write
 %edx: message length

```
.section .data
output:
    .string "Hello, World "
.section .text
.globl _start
_start:
    movl $13, %edx    ;message length
    movl $output, %ecx ;message to write
    movl $1, %ebx     ;file descriptor (stdout)
    movl $4, %eax     ;system call number (sys_write)
    int $0x80         ;call kernel
    movl $1, %eax     ;system call number (sys_exit)
    int $0x80         ;call kernel
```

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

C. THỰC HÀNH

C.1 Yêu cầu 1 – Thiết lập môi trường

Sinh viên cần chuẩn bị môi trường máy ảo Linux có đầy đủ các công cụ cần thiết bao gồm **ld** và **as** để sử dụng trong quá trình biên dịch file mã hợp ngữ sang file thực thi.

C.1.1 Kiểm tra và cài đặt các công cụ

Các công cụ có thể được kiểm tra đã cài đặt chưa bằng việc gõ các lệnh trong terminal:

\$ ld --version

```
ubuntu@ubuntu: ~  
ubuntu@ubuntu:~$ ld --version  
GNU ld (GNU Binutils for Ubuntu) 2.26.1  
Copyright (C) 2015 Free Software Foundation, Inc.  
This program is free software; you may redistribute it under the terms of  
the GNU General Public License version 3 or (at your option) a later version.  
This program has absolutely no warranty.
```

\$ as --version

```
ubuntu@ubuntu: ~  
ubuntu@ubuntu:~$ as --version  
GNU assembler (GNU Binutils for Ubuntu) 2.26.1  
Copyright (C) 2015 Free Software Foundation, Inc.  
This program is free software; you may redistribute it under the terms of  
the GNU General Public License version 3 or later.  
This program has absolutely no warranty.  
This assembler was configured for a target of `x86_64-linux-gnu'.
```

Nếu có kết quả nào trả về không tìm thấy công cụ, sinh viên gõ lệnh sau để cài đặt:

\$ sudo apt-get install binutils

C.1.2 Chạy thử chương trình mẫu trên môi trường Linux

Sinh viên thử chạy lại chương trình mẫu ở **Phần B.2** trên môi trường đã chuẩn bị dựa trên các bước biên dịch và liên kết để kiểm tra thiết lập môi trường cũng như hoạt động của chương trình.

C.2 Yêu cầu 2 – Bài tập lập trình hợp ngữ (assembly)

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:

- Hợp dịch với công cụ **as** để thu được một file nhị phân **.o** từ file **.s**:

```
$ as -o <tên file .o đầu ra> <tên file .s đầu vào>
```

- Liên kết với công cụ **ld** để tạo file thực thi của chương trình

```
$ ld -o <tên file thực thi đầu ra> <tên file .o đầu vào>
```

- Thực thi file

```
$ ./<tên 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.2.1 Chương trình in ra độ dài của một chuỗi cho trước (tối đa 9 ký tự)

Viết chương trình dưới dạng hợp ngữ thỏa với các yêu cầu như sau:

Input: Chuỗi **msg** được khai báo sẵn trong chương trình trong section .data.

Output: Xuất ra màn hình độ dài của chuỗi (số ký tự - không tính ký tự null)

Giới hạn: Chuỗi có độ dài tối đa **9 ký tự**

Ví dụ: Với chuỗi **msg = "Love UIT"**, kết quả đầu ra như hình dưới

```
ubuntu@ubuntu: ~/LTHT/Lab2/C21
ubuntu@ubuntu:~/LTHT/Lab2/C21$ as -o c21.o c21.s
ubuntu@ubuntu:~/LTHT/Lab2/C21$ ld -o c21 c21.o
ubuntu@ubuntu:~/LTHT/Lab2/C21$ ./c21
8
ubuntu@ubuntu:~/LTHT/Lab2/C21$
```

Gợi ý 1.1: Phần **D.3** hướng dẫn cách lấy được độ dài của một chuỗi đã khai báo trong section .data. *Lưu ý độ dài này sẽ có tính luôn ký tự null ở cuối chuỗi.*

Gợi ý 1.2: Để xuất một dữ liệu nào đó ra màn hình, cần đảm bảo:

- Dữ liệu đang **nằm trong 1 ô nhớ** (trong section .bss hoặc .data)
- Ta cần thiết lập giá trị các thanh ghi cho 1 system call xuất dữ liệu như sau:
 - o %eax: SYS_WRITE (4).
 - o %ebx: với STDOUT (1).
 - o %ecx: **địa chỉ ô nhớ** có giá trị cần xuất
 - o %edx: độ dài sẽ xuất ra (tính bằng bytes).

Gợi ý 1.3: Hệ thống xem giá trị nằm trong ô nhớ là mã ASCII của 1 ký tự cần in ra.

C.2.2 Chương trình tính giá trị trung bình cộng của 4 số (1 chữ số)

Viết chương trình dưới dạng hợp ngữ thỏa với các yêu cầu như sau:

Input: 4 số nguyên (1 chữ số) nhập vào từ bàn phím

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

Yêu cầu: Các số a, b, c, d nguyên và < 10.

Ví dụ: Nhập vào 4 số 2, 3, 5, 7 thì thu được kết quả $(2 + 3 + 5 + 7)/4 = 4$


```

ubuntu@ubuntu: ~/LTHT/Lab2/C22
ubuntu@ubuntu:~/LTHT/Lab2/C22$ as -o c22.o c22.s
ubuntu@ubuntu:~/LTHT/Lab2/C22$ ld -o c22 c22.o
ubuntu@ubuntu:~/LTHT/Lab2/C22$ ./c22
Enter a number (1-digit): 2
Enter a number (1-digit): 3
Enter a number (1-digit): 5
Enter a number (1-digit): 7
4
ubuntu@ubuntu:~/LTHT/Lab2/C22$

```

Gợi ý 2.1: Để nhận input từ bàn phím, ta thực hiện các bước:

- Chuẩn bị sẵn 1 vùng nhớ trong section .bss có độ dài phù hợp để lưu input.

Ví dụ bên dưới tạo 1 vùng nhớ có kích thước **2 bytes** (lưu 2 ký tự) có nhãn là **number1**.

```

.section .bss
.lcomm number1, 2

```

Trong đó: **\$number1** sẽ lấy địa chỉ vùng nhớ, (**number1**) sẽ lấy giá trị trong vùng nhớ.

- Ta cần thiết lập giá trị các thanh ghi cho 1 system call nhập dữ liệu như sau:

- %eax: SYS_READ (3).
- %ebx: với STDIN (0).
- %ecx: **địa chỉ ô nhớ** để lưu giá trị nhập vào (đã tạo trong section .bss)
- %edx: độ dài của chuỗi sẽ nhận

Lưu ý: Độ dài nhập vào cần tính cả ký tự xuống dòng. Do đó, với input là 1 ký tự hay số 1 chữ số cần khai báo độ dài của chuỗi nhập vào trong %edx là 2.

Gợi ý 2.2: Input nhập từ bàn phím luôn dưới dạng ký tự/chuỗi (kể cả khi nhập số), cần có bước chuyển từ ký tự số sang giá trị số nguyên tương ứng trước khi tính toán.

Gợi ý 2.3: Sinh viên có thể áp dụng phép tính toán trên bit như lab 1 hoặc phép chia với lệnh **div** để thực hiện tính trung bình cộng 4 số. Tham khảo thêm lệnh div: [Link](#).

C.2.3 Kiểm tra ký tự chữ cái nhập vào là chữ hoa hay chữ thường

Viết chương trình dưới dạng hợp ngữ thỏa với các yêu cầu như sau:

Input: Nhập 1 ký tự chữ cái (a – z hoặc A – Z) từ bàn phím.

Output: Xuất ra màn hình nhận định: “Chu hoa” nếu ký tự nhập vào là chữ in hoa, “Chu thường” nếu ký tự nhập vào là chữ thường.

Ví dụ:

```

ubuntu@ubuntu: ~/LTHT/Lab2/C23
ubuntu@ubuntu:~/LTHT/Lab2/C23$ as -o c23.o c23.s
ubuntu@ubuntu:~/LTHT/Lab2/C23$ ld -o c23 c23.o
ubuntu@ubuntu:~/LTHT/Lab2/C23$ ./c23
Enter a character: h
Chu thường
ubuntu@ubuntu:~/LTHT/Lab2/C23$ ./c23
Enter a character: K
Chu hoa
ubuntu@ubuntu:~/LTHT/Lab2/C23$

```


Gợi ý 3.1: Sử dụng bảng mã ASCII để xem giá trị của 1 ký tự hoa hoặc thường nằm trong khoảng giá trị nào. Ví dụ: 1 ký tự in thường x sẽ có ' $a' \leq x \leq 'z'$ '

Gợi ý 3.2: Sử dụng system call tương tự **C2.2** để nhập dữ liệu.

```
.section .bss
    .lcomm char, 1
...
    movl $3, %eax
    movl $0, %ebx
    movl $input, %ecx
    movl $2, %edx
    int $0x80
```

Gợi ý 3.3: 1 ký tự chỉ ứng với 1 byte, để chỉ lấy 1 byte tương ứng với ký tự đã nhập, ta nên dùng: `mov (char), %c1`. Các xử lý kế tiếp có thể thực hiện trên thanh ghi `%c1`.

C.2.4 Chương trình Kiểm tra tính tăng dần của các chữ số trong số có 3 chữ số

Viết chương trình dưới dạng hợp ngữ thỏa với các yêu cầu như sau:

Input: Số nguyên **a** có 3 chữ số nhập từ bàn phím.

Output: Xuất ra màn hình nhận định: "**Tang dan**" nếu các chữ số của a xếp theo thứ tự tăng dần, ngược lại xuất ra "**Khong tang dan**".

Ví dụ:

```
ubuntu@ubuntu: ~/LTHT/Lab2/C24
ubuntu@ubuntu:~/LTHT/Lab2/C24$ as -o c24.o c24.s
ubuntu@ubuntu:~/LTHT/Lab2/C24$ ld -o c24 c24.o
ubuntu@ubuntu:~/LTHT/Lab2/C24$ ./c24
Enter a number (3-digit): 123
Tang dan
ubuntu@ubuntu:~/LTHT/Lab2/C24$ ./c24
Enter a number (3-digit): 115
Tang dan
ubuntu@ubuntu:~/LTHT/Lab2/C24$ ./c24
Enter a number (3-digit): 362
Khong tang dan
ubuntu@ubuntu:~/LTHT/Lab2/C24$
```

Gợi ý 4.1: 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
```

D. THAM KHẢO

D.1 Một số lưu ý khi lập trình assembly

Cần phân biệt được 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 trong .data hoặc .bss
- Giá trị trong ô nhớ: **(output)** hoặc **output**

D.2 Bảng mã ASCII

Sử dụng bảng mã ASCII để chuyển đổi từ số (demical) 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'.

D.3 Khai báo 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, độ dài các chuỗi này khi dùng trong các system call vẫn được gán cứng giá trị số, đoạn khai báo bên dưới cho phép lấy giá trị độ dài của chuỗi và lưu vào biến.

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

E. YÊU CẦU & ĐÁNH GIÁ

E.1 Yêu cầu

Sinh viên thực hành theo **nhóm tối đa 2 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.

Lưu ý: Cần chú thích chức năng của các đoạn code.

Với mỗi chương trình, sinh viên nộp file .s với định dạng:
Lab2-MSSV1-MSSV2-HọTênSV1-HọTênSV2-<yêu cầu>.s
 Ví dụ: Lab2-MSSV1-MSS2-NguyenVanA-TranVanC-C21.s

E.2 Đánh giá kết quả

Tiêu chí	Chuyên cần (A)	Điểm bài lab (B)	Kết quả
Điểm	Có mặt = 10 Vắng = 7	Tối đa = 10 Không nộp bài = 0	Điểm = (A + B)/2
Lưu ý	Vắng + Không nộp bài : 0		

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