Chương 2

Máy tính và biên dịch

Mục đích cuối cùng của việc tận dụng lỗi phần mềm là thực thi các tác vụ mong muốn. Để làm được điều đó, trước hết chúng ta phải biết rõ cấu trúc của máy tính, cách thức hoạt động của bộ vi xử lý, những lệnh mà bộ vi xử lý có thể thực hiện, làm sao truyền lệnh tới bộ vi xử lý. Việc này cũng tương tự như học chạy xe máy vậy. Chúng ta phải biết nhấn vào nút nào để khởi động máy, nút nào để bật đèn xin đường, làm sao để rẽ trái, làm sao để dừng xe.

Trong chương này, chúng ta sẽ xem xét cấu trúc máy tính mà đặc biệt là bộ vi xử lý (Central Processing Unit, CPU), các thanh ghi (register), và bộ lệnh (instruction) của nó, cách đánh địa chỉ bộ nhớ tuyến tính (linear addressing). Kế tiếp chúng ta sẽ bàn tới mã máy (machine code), rồi hợp ngữ (assembly language) để có thể chuyển qua trao đổi về cách chương trình biên dịch (compiler) chuyển một hàm từ ngôn ngữ C sang hợp ngữ. Kết thúc chương chúng ta sẽ đưa ra một mô hình vị trí ngăn xếp (stack layout, stack diagram) của một hàm mẫu với các đối số và biến nội bộ.

Trong suốt tài liệu này, chúng ta sẽ chỉ nói đến cấu trúc của bộ vi xử lý Intel 32 bit.

2.1 Hệ cơ số

Trước khi đi vào cấu trúc máy tính, chúng ta cần nắm rõ một kiến thức nền tảng là hệ cơ số. Có ba hệ có số thông dụng mà chúng ta sẽ sử dụng trong tài liệu này:

Hệ nhị phân (binary) là hệ cơ số hai, được máy tính sử dụng. Mỗi một chữ số có thể có giá trị là 0, hoặc 1. Mỗi chữ số này được gọi là một bit. Tám (8) bit lập thành một byte (có ký hiệu là B). Một kilobyte (KB) là 1024 (2^{10}) byte. Một megabyte (MB) là 1024 KB.

Hệ thập phân (decimal) là hệ cơ số mười mà chúng ta, con người, sử dụng hàng ngày. Mỗi một chữ số có thể có giá trị là 0, 1, 2, 3, 4, 5, 6, 7, 8, hoặc 9.

Hệ thập lục phân (hexadecimal) là hệ cơ số mười sáu, được sử dụng để tính toán thay cho hệ nhị phân vì nó ngắn gọn và dễ chuyển đổi hơn. Mỗi một chữ số có thể có giá trị 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, và F trong đó A có giá trị là 10 (thập phân), B có giá trị là 11 và tương tự với C, D, E, F.

2.1.1 Chuyển đổi từ hệ cơ số bất kỳ sang hệ cơ số mười

Gọi cơ số đó là R, số chữ số là n, chữ số ở vị trí mang ít ý nghĩa nhất (least significant digit) là x_0 (thường là số tận cùng bên phải), chữ số tại vị trí mang nhiều ý nghĩa nhất (most significant digit) là x_{n-1} (thường là số tận cùng bên trái), và các chữ số còn lại từ x_1 cho tới x_{n-2}. Giá trị thập phân của con số này sẽ được tính theo công thức sau:

$$\textit{Giá trị thập phân} = x_0 \times R^0 + x_1 \times R^1 + ... + x_{n-2} \times R^{n-2} + x_{n-1} \times R^{n-1}$$

Ví dụ giá trị thập phân của số nhị phân 00111001 ($R = 2$, $n = 8$) là $1\times2^0+0\times2^1+0\times2^2+1\times2^3+1\times2^4+1\times2^5+0\times2^6+0\times2^7 = 57$, giá trị thập phân của số thập lục phân 7F ($R = 16$, $n = 2$) là $15 \times 16^0 + 7 \times 16^1 = 127$.

2.1.2 Chuyển đổi qua lại giữa hệ nhị phân và hệ thập lục phân

Mỗi một chữ số trong hệ thập lục phân tương ứng với bốn chữ số ở hệ nhị phân vì $16 = 2^4$. Do đó, để chuyển đổi qua lại giữa hai hệ này, chúng ta chỉ cần chuyển đổi từng bốn bit theo Bảng 2.1.

Ví dụ giá trị nhị phân của số thập lục phân AF là 10101111 vì A tương ứng với 1010 và F tương ứng với 1111, giá trị thập lục phân của số nhị phân 01010000 là 50.

Bảng 2.1: Chuyển đổi giữa hệ thập lục phân và nhị phân

Thập phân	**Thập lục phân**	**Nhị phân**
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Bảng 2.2: Một vài giá trị phổ thông trong bảng mã ASCII

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0																
1																
2																
3	0	1	2	3	4	5	6	7	8	9						
4		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z					
6		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z					

2.1.3 Bảng mã ASCII

Vì máy tính chỉ hiểu các bit 0 và 1 nên chúng ta cần có một quy định chung về cách biểu diễn những ký tự chữ như A, B, C, X, Y, Z. Bảng mã ASCII là một trong những quy định đó. Bảng mã này ánh xạ các giá trị thập phân nhỏ hơn 128 (từ 00 tới 7F trong hệ thập lục phân) thành những ký tự chữ thông thường. Bảng mã này được sử dụng phổ biến nên các hệ điều hành hiện đại đều tuân theo chuẩn ASCII.

Ngày nay chúng ta thường nghe nói về bảng mã Unicode vì nó thể hiện được hầu hết các ngôn ngữ trên thế giới và đặc biệt là tiếng Việt được giành riêng một vùng trong bảng mã. Bản thân Unicode cũng sử dụng cách ánh xạ ASCII cho các ký tự nhỏ hơn 128.

Bảng 2.2 liệt kê một số giá trị phổ thông trong bảng mã ASCII. Theo đó, ký tự chữ *A* hoa có mã 41 ở hệ thập lục phân, và mã thập lục 61 tương ứng với ký tự chữ *a* thường, mã thập lục 35 tương ứng với chữ số *5*.

Ngoài ra, một vài ký tự đặc biệt như ký tự kết thúc chuỗi NUL có mã thập lục 00, ký tự xuống dòng, tạo dòng mới (line feed, new line) có mã thập lục 0A, ký tự dời con trỏ về đầu dòng (carriage return) có mã thập lục 0D, ký tự khoảng trắng có mã thập lục 20.

Chúng ta đã xem xét qua kiến thức căn bản về các hệ cơ số và bảng mã ASCII. Ở phần kế tiếp chúng ta sẽ bàn về bộ vi xử lý của máy tính.

2.2 Kiến trúc máy tính

Máy tính gồm ba bộ phận chính là bộ xử lý (CPU), bộ nhập chuẩn (bàn phím) và bộ xuất chuẩn (màn hình). Chúng ta sẽ chỉ quan tâm đến bộ xử lý vì đây chính là trung tâm điều khiển mọi hoạt động của máy tính.

2.2.1 Bộ vi xử lý (Central Processing Unit, CPU)

Bộ vi xử lý đọc lệnh từ bộ nhớ và thực hiện các lệnh này một cách liên tục, không nghỉ. Lệnh sắp được thực thi được quyết định bởi con trỏ lệnh (instruction pointer). Con trỏ lệnh là một thanh ghi của CPU, có nhiệm vụ lưu trữ địa chỉ của lệnh kế tiếp trên bộ nhớ. Sau khi CPU thực hiện xong lệnh hiện tại, CPU sẽ thực hiện tiếp lệnh tại vị trí do con trỏ lệnh chỉ tới.

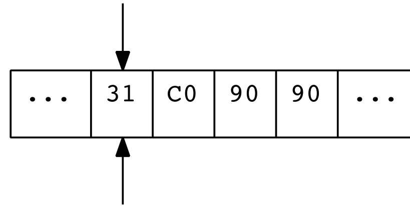
Hình 2.1a giả sử con trỏ lệnh đang mang giá trị 12345678. Điều này có nghĩa là CPU sẽ thực hiện lệnh tại địa chỉ 12345678. Tại địa chỉ này, chúng ta có lệnh 31 C0 (`xor eax, eax`). Vì lệnh này chiếm hai byte trên bộ nhớ nên sau khi thực hiện lệnh, con trỏ lệnh sẽ có giá trị là $12345678 + 2 = 1234567A$ như trong Hình 2.1b.

Tại địa chỉ 1234567A là lệnh 90 (`nop`). Do lệnh `nop` chỉ chiếm một byte bộ nhớ nên con trỏ lệnh sẽ trỏ tới ô nhớ kế nó tại địa chỉ 1234567B. Hình 2.1c minh họa giá trị của con trỏ lệnh sau khi CPU thực hiện lệnh `nop` ở Hình 2.1b.

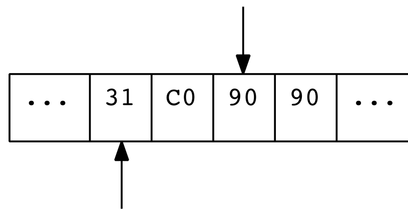
Để bạn đọc dễ nắm bắt, chúng ta có các quy ước sau:

- Những giá trị số đề cập đến trong tài liệu này sẽ được biểu diễn ở dạng thập lục phân trừ khi có giải thích khác.
- Ô nhớ sẽ có địa chỉ thấp hơn ở bên tay trái, địa chỉ cao hơn ở bên tay phải.
- Ô nhớ sẽ có địa chỉ thấp hơn ở bên dưới, địa chỉ cao hơn ở bên trên.
- Đôi khi chúng ta sẽ biểu diễn bộ nhớ bằng một dải dài từ trái sang phải như đã minh họa ở Hình 2.1; đôi khi chúng ta sẽ biểu diễn bằng một hộp các ngăn nhớ, mỗi ngăn nhớ dài 04 byte tương ứng với 32 bit như trong Hình 2.2.

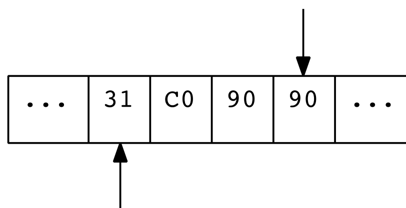
Từ ví dụ về con trỏ lệnh chúng ta nhận thấy rằng nếu muốn CPU thực hiện một tác vụ nào đó, chúng ta cần thỏa mãn hai điều kiện:



(a) Đang chỉ đến lệnh thứ nhất

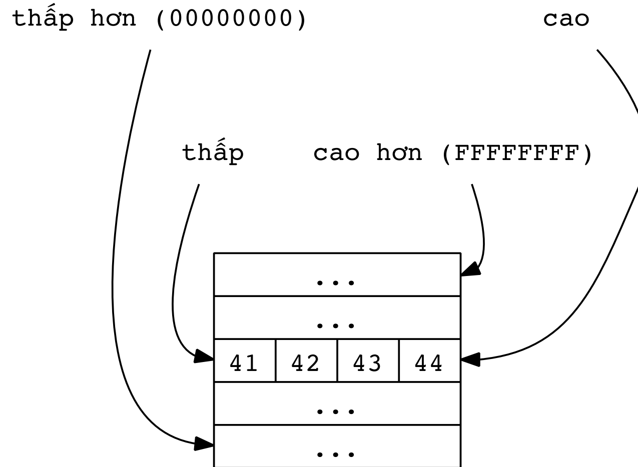


(b) Đang chỉ đến lệnh thứ hai



(c) Sau khi thực hiện lệnh `nop`

Hình 2.1: Con trỏ lệnh



Hình 2.2: Quy ước biểu diễn

1. Các lệnh thực thi cần được đưa vào bộ nhớ.
2. Con trỏ lệnh phải có giá trị là địa chỉ của vùng nhớ chứa các lệnh trên.

Vì mã lệnh thực thi và dữ liệu chương trình đều nằm trên bộ nhớ nên ta có thể tải mã lệnh vào chương trình thông qua việc truyền dữ liệu thông thường. Đây cũng chính là mô hình cấu trúc máy tính von Neumann với bộ xử lý và bộ phận chứa dữ liệu lẫn mã lệnh được tách rời.

Việc chọn và sử dụng mã lệnh (shellcode) phù hợp với mục đích tận dụng lỗi nằm ngoài phạm vi của tài liệu này. Chúng ta sẽ không bàn tới cách tạo các mã lệnh mà thay vào đó chúng ta sẽ giả sử rằng mã lệnh phù hợp đã được nạp vào bộ nhớ. Nói như vậy không có nghĩa là việc tạo mã lệnh quá đơn giản nên bị bỏ qua. Ngược lại, việc tạo mã lệnh là một vấn đề rất phức tạp, với nhiều kỹ thuật riêng biệt cho từng cấu trúc máy, từng hệ điều hành khác nhau, thậm chí cho từng trường hợp tận dụng riêng biệt. Hơn nữa, phần lớn các mã lệnh phổ thông đều có thể được sử dụng lại trong các ví dụ chúng ta sẽ bàn tới ở những phần sau nên bạn đọc có thể tự áp dụng như là một bài tập thực hành nhỏ.

Với giả thiết điều kiện thứ nhất đã hoàn thành, tài liệu này sẽ tập trung

vào việc giải quyết vấn đề thứ hai, tức là điều khiển luồng thực thi của máy tính. Theo ý kiến cá nhân của tác giả, đây thường là vấn đề mấu chốt của việc tận dụng lỗi, và cũng là lý do chính khiến chúng ta gặp nhiều khó khăn trong việc đọc hiểu các tin tức báo chí. Thực tế cho thấy (và sẽ được dẫn chứng qua các ví dụ) trong phần lớn các trường hợp tận dụng lỗi chúng ta chỉ cần điều khiển được luồng thực thi của chương trình là đã thành công 80% rồi.

Trong phần này, chúng ta đề cập đến con trỏ lệnh, và chấp nhận rằng con trỏ lệnh chứa địa chỉ ô nhớ của lệnh kế tiếp mà CPU sẽ thực hiện. Vậy thì con trỏ lệnh thật ra là gì?

2.2.2 Thanh ghi

Con trỏ lệnh ở 2.2.1 thật ra là một trong số các thanh ghi có sẵn trong CPU. Thanh ghi là một dạng bộ nhớ tốc độ cao, nằm ngay bên trong CPU. Thông thường, thanh ghi sẽ có độ dài bằng với độ dài của cấu trúc CPU.

Đối với cấu trúc Intel 32 bit, chúng ta có các nhóm thanh ghi chính được liệt kê bên dưới, và mỗi thanh ghi dài 32 bit.

Thanh ghi chung là những thanh ghi được CPU sử dụng như bộ nhớ siêu tốc trong các công việc tính toán, đặt biến tạm, hay giữ giá trị tham số. Các thanh ghi này thường có vai trò như nhau. Chúng ta hay gặp bốn thanh ghi chính là *EAX*, *EBX*, *ECX*, và *EDX*.

Thanh ghi xử lý chuỗi là các thanh ghi chuyên dùng trong việc xử lý chuỗi ví dụ như sao chép chuỗi, tính độ dài chuỗi. Hai thanh ghi thường gặp gồm có *EDI*, và *ESI*.

Thanh ghi ngăn xếp là các thanh ghi được sử dụng trong việc quản lý cấu trúc bộ nhớ ngăn xếp. Cấu trúc này sẽ được bàn đến trong Tiểu mục 2.2.4.3. Hai thanh ghi chính là *EBP* và *ESP*.

Thanh ghi đặc biệt là những thanh ghi có nhiệm vụ đặc biệt, thường không thể được gán giá trị một cách trực tiếp. Chúng ta thường gặp các thanh ghi như *EIP* và *EFLAGS*. EIP chính là con trỏ lệnh chúng ta đã biết. EFLAGS là thanh ghi chứa các cờ (mỗi cờ một bit) như cờ dấu (sign flag), cờ nhớ (carry flag), cờ không (zero flag). Các cờ này được thay đổi như là một hiệu ứng phụ của các lệnh chính. Ví

dụ như khi thực hiện lệnh lấy hiệu của 0 và 1 thì cờ nhớ và cờ dấu sẽ được bật. Chúng ta dùng giá trị của các cờ này để thực hiện các lệnh nhảy có điều kiện ví dụ như nhảy nếu cờ không được bật, nhảy nếu cờ nhớ không bật.

Thanh ghi phân vùng là các thanh ghi góp phần vào việc đánh địa chỉ bộ nhớ. Chúng ta hay gặp những thanh ghi *DS*, *ES*, *CS*. Trong những thế hệ 16 bit, các thanh ghi chỉ có thể định địa chỉ trong phạm vi từ 0 đến $2^{16} - 1$. Để vượt qua giới hạn này, các thanh ghi phân vùng được sử dụng để hỗ trợ việc đánh địa chỉ bộ nhớ, mở rộng nó lên 2^{20} địa chỉ ô nhớ. Cho đến thế hệ 32 bit thì hệ điều hành hiện đại đã không cần dùng đến các thanh ghi phân vùng này trong việc định vị bộ nhớ nữa vì một thanh ghi thông thường đã có thể định vị được tới 2^{32} ô nhớ tức là 4 GB bộ nhớ.

2.2.3 Bộ nhớ và địa chỉ tuyến tính

Thanh ghi là bộ nhớ siêu tốc nhưng đáng tiếc dung lượng của chúng quá ít nên chúng không phải là bộ nhớ chính. Bộ nhớ chính mà chúng ta nói đến là RAM với dung lượng thường thấy đến 1 hoặc 2 GB.

RAM là viết tắt của Random Access Memory (bộ nhớ truy cập ngẫu nhiên). Đặt tên như vậy vì để truy xuất vào bộ nhớ thì ta cần truyền địa chỉ ô nhớ trước khi truy cập nó, và tốc độ truy xuất vào địa chỉ nào cũng là như nhau. Vì thế việc xác định địa chỉ ô nhớ là quan trọng.

2.2.3.1 Định địa chỉ ô nhớ

Đến thế hệ 32 bit, các hệ điều hành đã chuyển sang dùng địa chỉ tuyến tính (linear addressing) thay cho địa chỉ phân vùng (segmented addressing). Cách đánh địa chỉ tuyến tính làm đơn giản hóa việc truy xuất bộ nhớ. Cụ thể là ta chỉ cần xử lý một giá trị 32 bit đơn giản, thay vì phải dùng công thức tính toán địa chỉ ô nhớ từ hai thanh ghi khác nhau. Ví dụ để truy xuất ô nhớ đầu tiên, ta sẽ dùng địa chỉ 00000000, để truy xuất ô nhớ kế tiếp ta dùng địa chỉ 00000001 và cứ thế. Ô nhớ sau nằm ở địa chỉ cao hơn ô nhớ trước 1 đơn vị.

Khi ta nói đến địa chỉ bộ nhớ, chúng ta đang nói đến địa chỉ tuyến tính của RAM. Địa chỉ tuyến tính này không nhất thiết là địa chỉ thật của ô

nhớ trong RAM mà sẽ phải được hệ điều hành ánh xạ lại. Công việc ánh xạ địa chỉ bộ nhớ được thực hiện qua phần quản lý bộ nhớ ảo (virtual memory management) của hệ điều hành.

Kiểu đánh địa chỉ tuyến tính ảo như vậy cho phép hệ điều hành mở rộng bộ nhớ thật có bằng cách sử dụng thêm phân vùng trao đổi (swap partition). Chúng ta thường thấy máy tính chỉ có 1 GB RAM nhưng địa chỉ bộ nhớ có thể có giá trị BFFFF6E4 tức là khoảng hơn 3 GB.

Trong 3 GB này, ngoài dữ liệu còn có các mã lệnh của chương trình. Chúng ta sẽ bàn tới các lệnh đó ở Tiểu mục 2.2.4.

2.2.3.2 Truy xuất bộ nhớ và tính kết thúc nhỏ

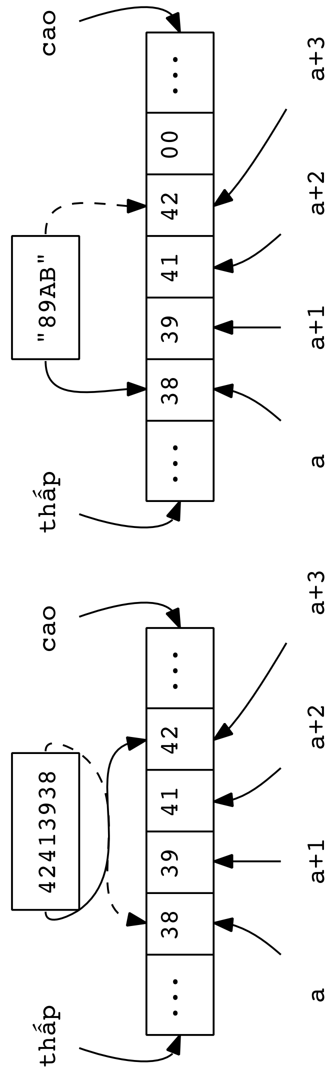
Như đã nói sơ qua, bộ vi xử lý cần xác định địa chỉ ô nhớ, và sẵn sàng nhận dữ liệu từ hoặc truyền dữ liệu vào bộ nhớ. Do đó để kết nối CPU với bộ nhớ chúng ta có hai đường truyền là đường truyền dữ liệu (data bus) và đường truyền địa chỉ (address bus). Khi cần đọc dữ liệu từ bộ nhớ, CPU sẽ thông báo rằng địa chỉ ô nhớ đã sẵn sàng trên đường truyền địa chỉ, và yêu cầu bộ nhớ truyền dữ liệu qua đường truyền dữ liệu. Khi ghi vào thì CPU sẽ yêu cầu bộ nhớ lấy dữ liệu từ đường truyền dữ liệu và ghi vào các ô nhớ.

Các đường truyền dữ liệu và địa chỉ đều có độ rộng 32 bit cho nên mỗi lần truy cập vào bộ nhớ thì CPU sẽ truyền hoặc nhận cả 32 bit để tối ưu việc sử dụng đường truyền. Điều này dẫn đến câu hỏi về kích thước các kiểu dữ liệu nhỏ hơn 32 bit.

Câu hỏi đầu tiên là làm sao để CPU nhận được 1 byte thay vì 4 byte (32 bit) nếu mọi dữ liệu từ bộ nhớ truyền về CPU đều là 32 bit? Câu trả lời là CPU nhận tất cả 4 byte từ bộ nhớ, nhưng sẽ chỉ xử lý 1 byte theo như yêu cầu của chương trình. Việc này cũng giống như ta có một thùng hàng to nhưng bên trong chỉ để một vật nhỏ.

Câu hỏi thứ hai liên quan tới vị trí của 8 bit dữ liệu sẽ được xử lý trong số 32 bit dữ liệu nhận được. Làm sao CPU biết lấy 8 bit nào? Các nhà thiết kế vi xử lý Intel x86 32 bit đã quyết định tuân theo tính kết thúc nhỏ (little endian). Kết thúc nhỏ là quy ước về trật tự và ý nghĩa các byte trong một kiểu trình bày dữ liệu mà byte ở vị trí cuối (vị trí thấp nhất) có ý nghĩa nhỏ hơn byte ở vị trí kế.

Ví dụ trong Hình 2.3a, bốn ô nhớ bắt đầu từ địa chỉ *a* biểu diễn giá trị thập lục *42413938*. Chúng ta thấy rằng byte ở vị trí thấp nhất có ý nghĩa



(a) Đối với giá trị 42413938

(b) Đối với chuỗi “89AB”

Hình 2.3: Tính kết thúc nhỏ

nhỏ nhất, và byte ở vị trí cao nhất có ý nghĩa lớn nhất đối với giá trị này. Thay đổi 1 đơn vị của byte thấp chỉ làm giá trị thay đổi $256^0 = 1$ đơn vị, trong khi thay đổi 1 đơn vị ở byte cao làm giá trị thay đổi $256^3 = 16777216$ đơn vị.

Cùng lúc đó, Hình 2.3b minh họa cách biểu diễn một chuỗi *"89AB"* kết thúc bằng ký tự NUL trong bộ nhớ. Chúng ta thấy từng byte của chuỗi (trong hình là giá trị ASCII của các ký tự tương ứng) được đưa vào bộ nhớ theo đúng thứ tự đó. Tính kết thúc nhỏ không có ý nghĩa với một chuỗi vì các byte trong một chuỗi có vai trò như nhau; không có sự phân biệt về mức quan trọng của từng byte đối với dữ liệu.

Thông qua hai hình minh họa, bạn đọc cũng chú ý rằng các ô nhớ có thể chứa cùng một dữ liệu (các byte 38, 39, 41, 42) nhưng ý nghĩa của dữ liệu chứa trong các ô nhớ đó có thể được hiểu theo các cách khác nhau bởi chương trình (là giá trị thập lục 42413938 hay là chuỗi "89AB").

Vì tuân theo tính kết thúc nhỏ nên CPU sẽ lấy giá trị tại địa chỉ thấp, thay vì tại địa chỉ cao. Xét cùng ví dụ đã đưa, nếu ta lấy 1 byte từ 32 bit dữ liệu bắt đầu từ địa chỉ a thì nó sẽ có giá trị thập lục *38*; 2 byte sẽ có giá trị *3938*; và 4 byte sẽ có giá trị *42413938*.

2.2.4 Tập lệnh, mã máy, và hợp ngữ

Tập lệnh (instruction set) là tất cả những lệnh mà CPU có thể thực hiện. Đây có thể được coi như kho từ vựng của một máy tính. Các chương trình là những tác phẩm văn học; chúng chọn lọc, kết nối các từ vựng riêng rẽ lại với nhau thành một thể thống nhất diễn đạt một ý nghĩa riêng.

Cũng như các từ vựng trong ngôn ngữ tự nhiên, các lệnh riêng lẻ có độ dài khác nhau (như đã nêu ra trong ví dụ ở Hình 2.1). Chúng có thể chiếm 1 hoặc 2 byte, và đôi khi có thể tới 9 byte, thậm chí đạt độ dài tối đa 15 byte trong các trường hợp đặc biệt. Những giá trị chúng ta đã thấy như 90, 31 C0 là những lệnh được CPU hiểu và thực hiện được. Các giá trị này được gọi là mã máy (machine code, opcode). Mã máy còn được biết đến như là ngôn ngữ lập trình thế hệ thứ nhất.

Tuy nhiên, con người sẽ gặp nhiều khó khăn nếu buộc phải điều khiển máy tính bằng cách sử dụng mã máy trực tiếp. Do đó, chúng ta đã sáng chế ra một bộ từ vựng khác gần với ngôn ngữ tự nhiên hơn, nhưng vẫn giữ được tính cấp thấp của mã máy. Thay vì chúng ta sử dụng giá trị 90 thì chúng ta dùng từ vựng *NOP*, tức là No Operation. Thay cho 31 C0 sẽ có

XOR EAX, EAX, tức là thực hiện phép toán luận tử XOR giữa hai giá trị thanh ghi EAX với nhau và lưu kết quả vào lại thanh ghi EAX, hay nói cách khác là thiết lập giá trị của EAX bằng 0. Rõ ràng bộ từ vựng này dễ hiểu hơn các giá trị khó nhớ kia. Chúng được gọi là hợp ngữ.

Hợp ngữ được xem là ngôn ngữ lập trình thế hệ thứ hai. Các ngôn ngữ khác như C, Pascal được xem là ngôn ngữ lập trình thế hệ thứ ba vì chúng gần với ngôn ngữ tự nhiên hơn hợp ngữ.

2.2.4.1 Các nhóm lệnh

Hợp ngữ có nhiều nhóm lệnh khác nhau. Chúng ta sẽ chỉ điểm qua các nhóm và những lệnh sau.

Nhóm lệnh gán là những lệnh dùng để gán giá trị vào ô nhớ, hoặc thanh ghi ví dụ như LEA, MOV, SETZ.

Nhóm lệnh số học là những lệnh dùng để tính toán biểu thức số học ví dụ như INC, DEC, ADD, SUB, MUL, DIV.

Nhóm lệnh luận lý là những lệnh dùng để tính toán biểu thức luận lý ví dụ như AND, OR, XOR, NEG.

Nhóm lệnh so sánh là những lệnh dùng để so sánh giá trị của hai đối số và thay đổi thanh ghi EFLAGS ví dụ như TEST, CMP.

Nhóm lệnh nhảy là những lệnh dùng để thay đổi luồng thực thi của CPU bao gồm lệnh nhảy không điều kiện JMP, và các lệnh nhảy có điều kiện như JNZ, JZ, JA, JB.

Nhóm lệnh ngăn xếp là những lệnh dùng để đẩy giá trị vào ngăn xếp, và lấy giá trị từ ngăn xếp ra ví dụ như PUSH, POP, PUSHA, POPA.

Nhóm lệnh hàm là những lệnh dùng trong việc gọi hàm và trả kết quả từ một hàm ví dụ như CALL và RET.

2.2.4.2 Cú pháp

Mỗi lệnh hợp ngữ có thể nhận 0, 1, 2, hoặc nhiều nhất là 3 đối số. Đa số các trường hợp chúng ta sẽ gặp lệnh có hai đối số theo dạng tương tự như *ADD dst, src*. Với dạng này, lệnh số học *ADD* sẽ được thực hiện với hai

đối số *dst* và *src*, rồi kết quả cuối cùng sẽ được lưu lại trong *dst*, thể hiện công thức $dst = dst + src$.

Tùy vào mỗi lệnh riêng biệt mà *dst* và *src* có thể có các dạng khác nhau. Nhìn chung, chúng ta có các dạng sau đây cho *dst* và *src*.

Giá trị trực tiếp là một giá trị cụ thể như 6789ABCD. Ví dụ `MOV EAX, 6789ABCD` sẽ gán giá trị 6789ABCD vào thanh ghi EAX. Giá trị trực tiếp không thể đóng vai trò của *dst*.

Thanh ghi là các thanh ghi như EAX, EBX, ECX, EDX. Xem ví dụ trên.

Bộ nhớ là giá trị tại ô nhớ có địa chỉ được chỉ định. Để tránh nhầm lẫn với giá trị trực tiếp, địa chỉ này được đặt trong hai ngoặc vuông. Ví dụ `MOV EAX, [6789ABCD]` sẽ gán giá trị 32 bit bắt đầu từ ô nhớ 6789ABCD vào thanh ghi EAX. Chúng ta cũng sẽ gặp các thanh ghi trong địa chỉ ô nhớ ví dụ như lệnh `MOV EAX, [ECX + EBX]` sẽ gán giá trị 32 bit bắt đầu từ ô nhớ tại địa chỉ là tổng giá trị của hai thanh ghi EBX và ECX. Bạn đọc cũng nên lưu ý rằng lệnh LEA (Load Effective Address, gán địa chỉ) với cùng đối số như trên sẽ gán giá trị là tổng của ECX và EBX vào thanh ghi EAX vì địa chỉ ô nhớ của *src* chính là ECX + EBX.

2.2.4.3 Ngăn xếp

Chúng ta nhắc đến ngăn xếp (stack) trong khi bàn về các nhóm lệnh ở Tiểu mục 2.2.4.1. Ngăn xếp là một vùng bộ nhớ được hệ điều hành cấp phát sẵn cho chương trình khi nạp. Chương trình sẽ sử dụng vùng nhớ này để chứa các biến nội bộ bộ (local variable), và lưu lại quá trình gọi hàm, thực thi của chương trình. Trong phần này chúng ta sẽ bàn tới các lệnh và thanh ghi đặc biệt có ảnh hưởng đến ngăn xếp.

Ngăn xếp hoạt động theo nguyên tắc vào sau ra trước (Last In, First Out). Các đối tượng được đưa vào ngăn xếp sau cùng sẽ được lấy ra đầu tiên. Khái niệm này tương tự như việc chúng ta chồng các thùng hàng lên trên nhau. Thùng hàng được chồng lên cuối cùng sẽ ở trên cùng, và sẽ được dỡ ra đầu tiên. Như vậy, trong suốt quá trình sử dụng ngăn xếp, chúng ta luôn cần biết vị trí đỉnh của ngăn xếp. Thanh ghi ESP lưu giữ vị trí đỉnh ngăn xếp, tức địa chỉ ô nhớ của đối tượng được đưa vào ngăn xếp sau cùng, nên còn được gọi là con trỏ ngăn xếp (stack pointer).