**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

**---oOo---**

Logo

Description automatically generated

**TIỂU LUẬN CUỐI KỲ**

**HỢP NGỮ X86**

Họ và tên sinh viên: Huỳnh Tấn Thọ

Mã số sinh viên: 19120383

Lớp: 19\_1

Môn học: Hệ thống máy tính

Giảng viên: Lê Quốc Hòa

Thành phố Hồ Chí Minh, tháng 9 năm 2021

**MỤC LỤC**

[1. Mở đầu 3](#_Toc83374621)

[2. Giới thiệu chung về hợp ngữ x86 3](#_Toc83374622)

[3. Điểm giống và khác so với MIPS 6](#_Toc83374623)

[4. Một số trình hợp dịch x86 phổ biến 7](#_Toc83374624)

[5. Lập trình hợp ngữ x86 trên Visual Studio 2019 8](#_Toc83374625)

[TÀI LIỆU THAM KHẢO 13](#_Toc83374626)

# **1. Mở đầu**

- Trên lớp lý thuyết của môn học này, ta đã học và thực hành khá nhiều về một hợp ngữ được sử dụng phổ biến với tên gọi là MIPS. MIPS bản thân nó khá đơn giản, do đó rất phù hợp dành cho người mới bắt đầu, đặc biệt là người mới tìm hiểu về kiến trúc của máy tính, cách CPU (bộ xử lý trung tâm) xử lý thông tin, từ đó có cái nhìn sâu sắc hơn về cách máy tính hoạt động.

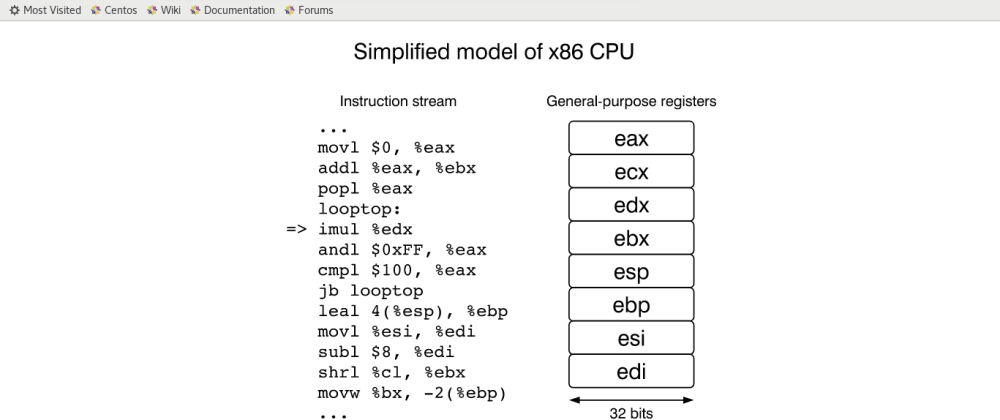
- Tuy nhiên, trên thực tế, có rất nhiều loại vi xử lý dành cho từng thiết bị khác nhau, mỗi hệ máy lại có kiến trúc khác nhau, do đó hợp ngữ cũng không thể chỉ có một. Ngoài MIPS như ta đã được học thì còn có ARM, x86… Trong phạm vi bài tiểu luận này, ta sẽ tìm hiểu kĩ hơn về x86, cách x86 hoạt động trên một số loại máy phổ biến, cũng như sự giống và khác nhau cơ bản giữa x86 và MIPS.

# **2. Giới thiệu chung về hợp ngữ x86**

- Trước hết, ta phải hiểu x86 là gì. Kiến trúc x86 ban đầu được Intel phát triển dành cho các vi xử lý của họ, bao gồm Intel 8086 và 8088 cùng các thế hệ sau của nó (80186, 80286, 80386,…)

- Kiến trúc này về sau được sử dụng rộng rãi bởi các hãng sản xuất vi xử lý. Người ta ngày càng mở rộng và tối ưu hóa thêm cho nó, dù vậy, khả năng tương thích ngược vẫn rất cao, có thể chạy được trên các dòng CPU những năm 70 80 của Intel.

- Hợp ngữ x86 là một ngôn ngữ trung gian được dùng để giao tiếp giữa các ngôn ngữ lập trình bậc cao và ngôn ngữ máy, sử dụng chủ yếu cho các vi xử lý có kiến trúc x86. Ban đầu, có tổng cộng 81 lệnh dành cho dòng CPU Intel 8086/8088 khi được ra mắt. Sau này, với sự ra đời của nhiều vi xử lý đến từ nhiều hãng khác nhau, người ta thêm vào rất nhiều lệnh cho hợp ngữ này. Các lệnh của kiến trúc x86 thường được biểu diễn bởi các opcode. Ví dụ, lệnh HLT (halt – dừng chương trình) có opcode là 0xF4.



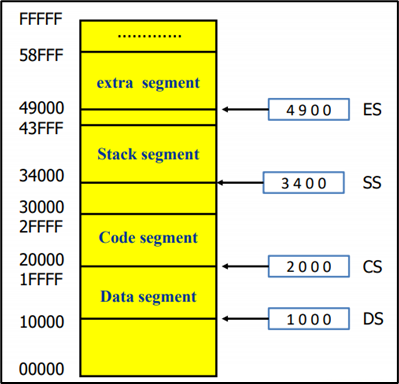
- Cú pháp của hợp ngữ x86 được chia làm 2 nhánh chính: Intel và AT&T. Chúng có những điểm khác nhau chủ yếu sau đây:

|  |  |  |
| --- | --- | --- |
|  | Intel | AT&T |
| Thứ tự tham số | Đích đứng sau, nguồn đứng trước. Ví dụ: mov eax, 10 | Đích đứng trước, nguồn đứng sau. Ví dụ: movl $10, %eax |
| Kích thước tham số | Do kích thước thanh ghi lưu trữ quy định. Ví dụ 16-bit, 32-bit… | Do toán hạng quy định. Ví dụ: addl là cộng số kiểu long, addw là cộng số kiểu word |
| Tiền tố của giá trị | Hệ thống có khả năng tự nhận biết. | Giá trị cố định bắt đầu với $, giá trị thanh ghi bắt đầu với %. |
| Truy cập bộ nhớ | Sử dụng ngoặc vuông. Ví dụ:  mov eax, [ebx + ecx\*4 + mem\_loc] | Sử dụng ngoặc tròn. Ví dụ: movl mem\_loc(%ebx, %ecx, 4), %eax |

- Các thanh ghi đa năng:

* AX (accumulator): lưu trữ kết quả
* BX (base): lưu địa chỉ/index cơ bản
* CX (count): lưu số lần lặp. Thanh ghi CL giữ số bit khi thực hiện shift, rotate
* DX (data): kết hợp với thanh ghi AX để giữ kết quả của phép nhân/chia.
* Các thanh ghi này nếu có dạng \*X (ví dụ: AX, BX…) thì là dành cho vi xử lý 16-bit, nếu có dạng E\*X (ví dụ: EAX, EBX…) thì là dành cho 32-bit, còn nếu có dạng R\*X (ví dụ như RAX, RBX…) là dành cho 64-bit.

- Các thanh ghi phân vùng: có dạng \*S (ví dụ: ES, DS,…)



- Các thanh ghi phân vùng này không nằm rời nhau, không chiếm 1 vùng nhớ riêng, mà chúng chồng lên nhau.

**Chart

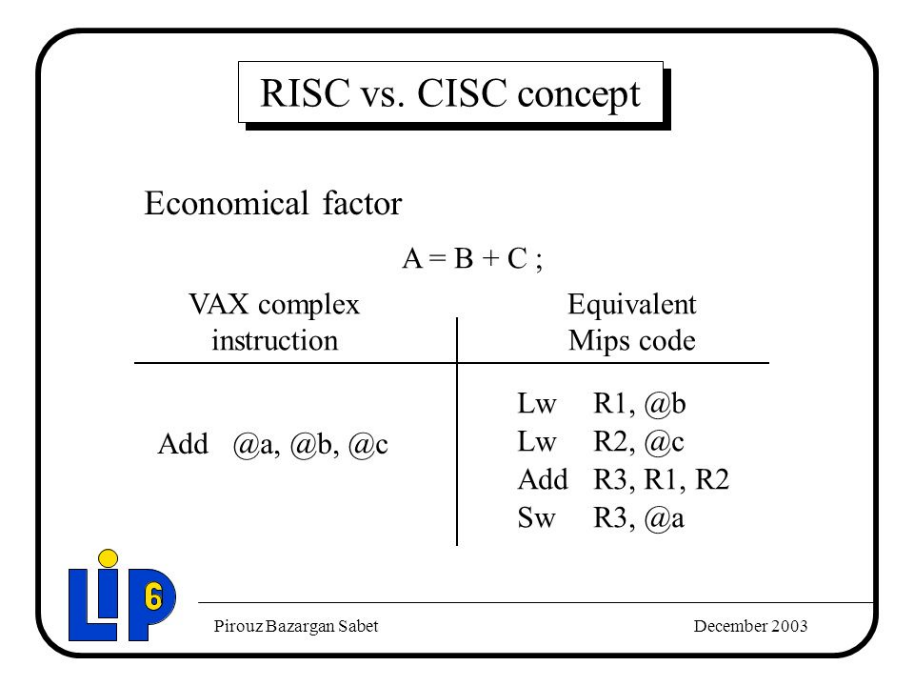
Description automatically generated with medium confidence**

- Ngoài ra còn có một số loại thanh ghi khác như thanh ghi con trỏ, thanh ghi chỉ mục, thanh ghi cờ…

# **3. Điểm giống và khác so với MIPS**

- MIPS và x86 đều có những lệnh cũng như cách thực thi tương tự nhau. Chúng đều có các lệnh giúp thực hiện phép toán số học (add, sub, mul, div…), các phép toán bitwise (and, or, xor…), các lệnh giúp truy cập thanh ghi, truy cập bộ nhớ, lệnh di chuyển dữ liệu (mov), các lệnh rẽ nhánh, sử dụng các nhãn (label) để thực hiện vòng lặp, các kiểu dữ liệu như chuỗi, số nguyên có dấu, số thực,… Về cơ bản, nguyên tắc hoạt động của chúng gần như tương đương.

- Sự khác nhau cơ bản giữa MIPS và x86 chính là: MIPS là một kiến trúc dựa trên Tập Lệnh Giản Lược (Reduced Instruction Set Computer – RISC). Trong khi đó, x86 là kiến trúc dựa trên Tập Lệnh Hoàn Thiện (Complete Instruction Set Computer – CISC). Điều này có nghĩa là, các lệnh trên vi xử lý có kiến trúc MIPS gồm nhiều lệnh đơn giản thay vì một lệnh phức tạp như trên x86. Ví dụ như ta cần cộng hai số sau đó lưu kết quả vào biến nào đó, thì MIPS thực hiện 2 lệnh riêng biệt nhau (mỗi lệnh 1 chu kỳ) trong khi đó x86 thực thi công việc này như một lệnh.



- Sự khác nhau về độ phức tạp đó cũng tạo ra sự khác biệt lớn giữa chúng. Các lệnh đơn giản sẽ giúp vi xử lý tiết kiệm năng lượng hơn, do chúng thực hiện ít công việc hơn trong một đơn vị thời gian, hiển nhiên, việc lập trình cũng sẽ cần nhiều dòng code hơn. Vì nguyên nhân đó, các vi xử lý x86 thường dành cho các hệ thống yêu cầu hiệu năng cao (như máy tính để bàn, máy chủ), còn các vi xử lý MIPS sẽ dành cho các hệ thống yêu cầu mức tiêu thụ điện năng thấp (như các thiết bị di động, các hệ thống nhúng).

- Các lệnh trong hợp ngữ x86 cũng có khả năng thao tác trực tiếp với bộ nhớ máy mà không cần qua thanh ghi, giúp nhiều lệnh khác nhau truy cập trực tiếp vào bộ nhớ hệ thống. Trong khi đó, MIPS chỉ có thể thông qua thanh ghi cho phép một lệnh cụ thể nào đó truy cập tại một thời điểm.

- Việc chúng có cách truy cập bộ nhớ và thực thi lệnh khác nhau cũng ảnh hưởng đến quy trình sản xuất. Các vi xử lý x86, hoặc nói rộng ra là kiến trúc CISC thì dùng nhiều bóng bán dẫn (transistor) hơn để lưu những lệnh phức tạp khi hoạt động. Trong khi đó, các vi xử lý theo kiến trúc RISC dùng nhiều bóng bán dẫn cho các thanh ghi địa chỉ ô nhớ hơn.

- Ngoài ra, các vi xử lý x86 ngày càng được sử dụng rộng rãi, chúng được tích hợp thêm rất nhiều lệnh mới và ngày càng có khả năng tính toán phức tạp hơn.

# **4. Một số trình hợp dịch x86 phổ biến**

- Trình hợp dịch (assembler) là một chương trình dịch hợp ngữ thành mã máy, từ đó máy tính có thể hiểu và thực thi.

- Mỗi kiến trúc vi xử lý khác nhau lại có cú pháp lập trình hợp ngữ khác nhau, đặc biệt kiến trúc x86 được sử dụng rất rộng rãi, do đó, trình hợp dịch cho hợp ngữ x86 cũng rất đa dạng. Một số loại phổ kiến có thể kể đến như:

* MASM: do Microsoft phát triển, sử dụng trên Windows và DOS.
* NASM: sử dụng trên Linux, macOS,…
* FASM: sử dụng trên Windows, DOS, Linux,…
* TASM: sử dụng trên Windows và DOS.

- Các trình hợp dịch khác nhau trên mỗi hệ điều hành khác nhau cũng có sự khác biệt nhất định về cú pháp khi sử dụng. Ví dụ, trình hợp dịch MASM chạy trên Windows phải cần khai báo một số thư viện khi sử dụng, còn trên DOS thì không; trình hợp dịch NASM dùng trên Linux cũng khác so với trên Windows, thậm chí NASM cho Linux có hỗ trợ cả thư viện ngôn ngữ C.

- Ngoài ra, hợp ngữ x86 sử dụng trên hệ máy 32-bit và hệ máy 64-bit cũng có sự khác nhau nhất định về cú pháp và nguyên lý hoạt động.

# **5. Lập trình hợp ngữ x86 trên Visual Studio 2019**

- Ta có thể sử dụng Visual Studio 2019 để chạy một số chương trình hợp ngữ x86. Cụ thể, Visual Studio 2019 sử dụng trình hợp dịch MASM.

- Trước tiên, ta tạo một Empty Project trên Visual Studio 2019.

A screenshot of a computer

Description automatically generated with medium confidence

- Click phải vào tên project, chọn Build Dependencies, chọn Build Customizations

A screenshot of a computer

Description automatically generated with medium confidence

- Chọn masm (.targets, .props)

A screenshot of a computer

Description automatically generated

- Tạo file mới với đuôi .asm

A screenshot of a computer

Description automatically generated with medium confidence

- Gõ đoạn code ngắn và nhấm F5 để chạy thử. Ở đây, ta thực hiện tính tổng hai số nguyên cho trước và trả về kết quả thông qua Exitcode.

A screenshot of a computer

Description automatically generated with medium confidence

- Kết quả sau khi chạy chương trình.

A screenshot of a computer

Description automatically generated

- Ngoài ra, ta cũng có thể sử dụng một số thư viện có sẵn. Ở đây, ta dùng irvine32.inc để in ra dòng chữ Hello World trên màn hình Console.

Text

Description automatically generated

- Kết quả sau khi chạy chương trình.

Graphical user interface, text, application

Description automatically generated

# **TÀI LIỆU THAM KHẢO**

1. Slide bài giảng Kiến trúc x86-32bit. GV: Phạm Tuấn Sơn. Đại học Khoa học Tự nhiên, ĐHQG TP.HCM
2. Slide bài giảng Kiến trúc x86. GV: Chung Thùy Linh. Đại học Khoa học Tự nhiên, ĐHQG TP.HCM
3. X86 – Wikipedia

<https://en.wikipedia.org/wiki/X86>

1. X86 Assembling Language

<https://en.wikipedia.org/wiki/X86_assembly_language>

1. X86 Instruction Listings

<https://en.wikipedia.org/wiki/X86_instruction_listings>

1. What is x86 assembly?

<https://resources.infosecinstitute.com/topic/what-is-x86-assembly/>

1. What is x86 assembly language?

<https://www.quora.com/What-is-x86-assembly-language>

1. RISC. vs CISC.

<https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/risccisc/>

1. What is the actual difference between x86, ARM and MIPS architectures?

<https://www.quora.com/What-is-the-actual-difference-between-x86-ARM-and-MIPS-architectures>

1. Assembly Language for x86 Processors by Kip Irvine

<http://www.nlpir.org/wordpress/wp-content/uploads/2019/03/Assembly.Language.For_.x86.Processors.Kip_.R..Irvine..6ed.Prentice.Hall_.2011www.xuexi111.com_.pdf>

1. Getting Started with MASM and Visual Studio 2017

<http://asmirvine.com/gettingStartedVS2017/index.htm>