TỔ CHỨC HOẠT ĐỘNG BỘ VI XỬ LÝ

Môn học: Kiến trúc máy tính & Hợp ngữ

Ngôn ngữ lập trình

- Là loại ngôn ngữ nhân tạo (Ví dụ: C/C++) được cấu thành bởi 2 yếu tố chính:
  - Từ vựng: là các keyword (struct, enum, if, int…)
  - Ngữ pháp: syntax (if(…){} else{}, do{} while()…)
- Ngôn ngữ lập trình giúp cho người sử dụng nó (gọi là lập trình viên) có thể diễn đạt và mô tả các hướng dẫn cho máy tính hoạt động theo ý muốn của mình.
- Độ phức tạp (trừu tượng) của các hướng dẫn này quyết định thứ bậc của ngôn ngữ
  - Độ phức tạp càng cao thì bậc càng thấp
  - Ví dụ: C Sharp (C#) là ngôn ngữ bậc cao hơn C

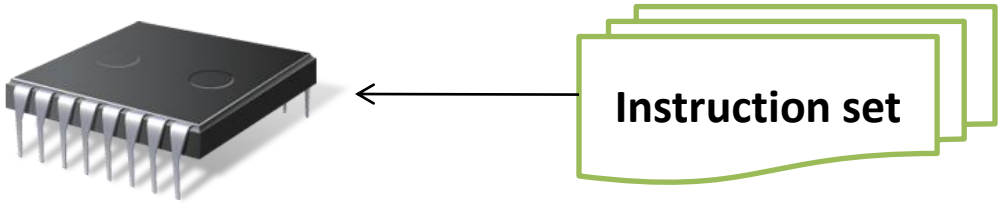
Nhận xét

- Ngôn ngữ nào mà con người dễ hiểu nhất lại là ngôn ngữ máy tính "khó hiểu" nhất
  - Ngôn ngữ bậc càng cao thì con người càng dễ hiểu nhưng máy tính lại càng "khó hiểu"
- Nhưng máy tính lại là nơi chúng ta cần nó hiểu đúng và nhanh nhất để có thể thực thi những gì chúng ta muốn

→ Ngôn ngữ máy (Machine language)

Ngôn ngữ máy (Machine Laguage)

- Ngôn ngữ máy cho phép người lập trình đưa ra các hướng dẫn đơn giản mà bộ vi xử lý (CPU) có thể thực hiện được ngay
- Các hướng dẫn này được gọi là chỉ thị / lệnh (instruction) hoặc mã máy (machine code)
- Mỗi bộ vi xử lý (CPU) có 1 ngôn ngữ riêng, gọi là bộ lệnh (instruction set)
- Trong cùng 1 dòng vi xử lý (processor family) bộ lệnh gần giống nhau



Instruction

- Là dãy bit chứa yêu cầu mà bộ xử lý trong CPU (ALU) phải thực hiện
- Instruction gồm 2 thành phần:
  - Mã lệnh (opcode): thao tác cần thực hiện
  - Thông tin về toán hạng (operand): các đối tượng bị tác động bởi thao tác chứa trong mã lệnh

ISA (Instruction Set Architecture)

- Tập lệnh dành cho những bộ vi xử lý có kiến trúc tương tự nhau
- Một số ISA thông dụng:
  - Dòng vi xử lý 80x86 (gọi tắt x86) của Intel
    - IA-16: Dòng xử lý 16 bit (Intel 8086, 80186, 80286)
    - IA-32: Dòng xử lý 32 bit (Intel 80368 – i386, 80486 – i486, Pentium II, Pentium III …)
    - IA-64: Dòng xử lý 64 bit (Intel x86-64 như Pentium D…)
  - MIPS: Dùng rất nhiều trong hệ thống nhúng (embedded system)
  - PowerPC của IBM

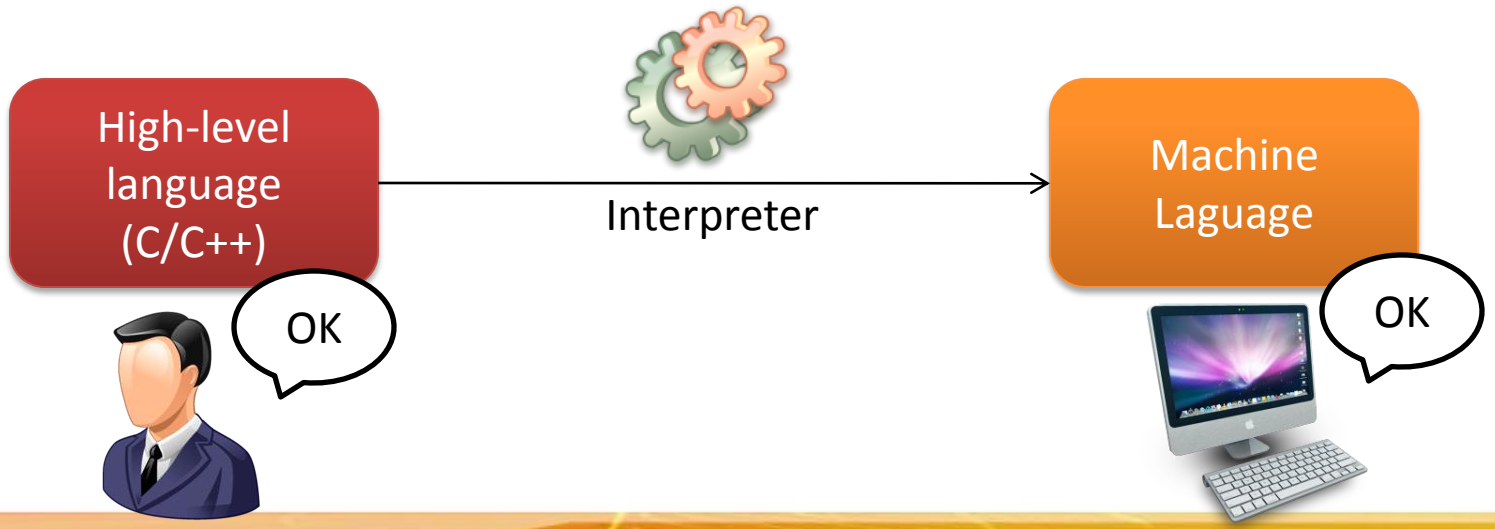
Thiết kế ISA: CISC & RISC

- Có 2 trường phái thiết kế bộ lệnh:
  - Complete Instruction Set Computer (CISC): bộ lệnh gồm rất nhiều lệnh, từ đơn giản đến phức tạp
  - Reduced Instruction Set Computer (RISC): bộ lệnh chỉ gồm các lệnh đơn giản
- Nên chọn kiểu nào?

Tuy nhiên

- Không phải ai cũng muốn / có thể lập trình ngôn ngữ máy vì quá khó hiểu so với ngôn ngữ bình thường của con người

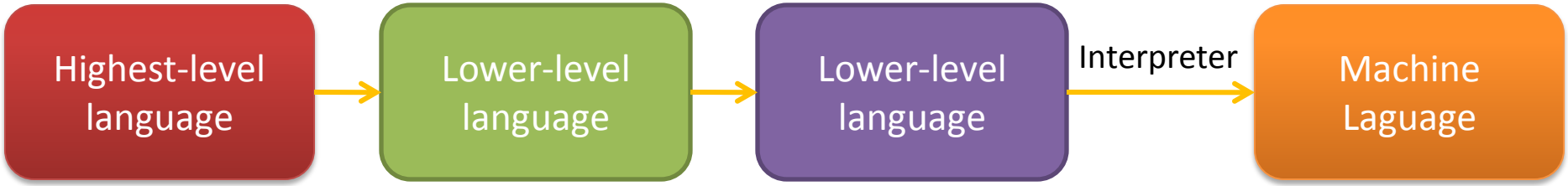
→ Nhu cầu cần có bộ phận phiên dịch (interpreter)



Nhận xét

- Trong 1 số trường hợp, việc viết bằng ngôn ngữ cấp "quá cao" trở nên chạy khá chậm vì phải phiên dịch nhiều lần để trở thành ngôn ngữ máy

→ Hợp ngữ (Assembly language)



Hợp ngữ

- Các mã máy chỉ là các con số (0 / 1)
- Trong ngôn ngữ máy không có khái niệm biến → thay vào đó là địa chỉ ô nhớ, thanh ghi (lưu trữ mã lệnh, dữ liệu)
- Để dễ dàng lập trình hơn → dùng ký hiệu mã giả thay cho các số biểu diễn địa chỉ ô nhớ, các tên (label, tên biến, tên chương trình)
- Hợp ngữ rất gần với ngôn ngữ máy nhưng lại đủ để con người hiểu và sử dụng tốt hơn ngôn ngữ máy
  - Ví dụ: Ghi giá trị 5 vào thanh ghi $4

    Ngôn ngữ máy: 00110100 0000100 00000000 00000101

    Hợp ngữ : ori $4, $0, 5

Lưu ý

- Vì mỗi bộ vi xử lý có 1 cấu trúc thanh ghi và tập lệnh (ngôn ngữ) riêng nên khi lập trình hợp ngữ phải nói rõ là lập trình cho bộ vi xử lý nào, hay dòng (family) vi xử lý nào
  - Ví dụ:
    - Hợp ngữ cho MIPS → RISC
    - Hợp ngữ cho dòng vi xử lý Intel 80x86 → CISC

Ví dụ: hợp ngữ MIPS-32bit

```
.data                # data segment
  str:
     .asciiz ”hello MIPS”
.text                # text segment
  .globl main
  main:
     addi $v0, $0, 4 # 4 = print str syscall
     la $a0, str     # load address of string
     syscall         # execute the system call
```

Ví dụ: hợp ngữ X86-32bit

```
global _WinMain@16
extern _MessageBoxA@16

[section .data]
  title db "Message",0
  message db "Hello asm!",0
[section .code]
_WinMain@16:
   push 0
   push title
   push message
   push 0
   call _MessageBoxA@16
   ret 16
```

Thảo luận

- Ta có thể hình dung như sau:


```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
sw $t0, 4($2)
```

```
0000 1001 1100 0110
1111 0101
.....
```

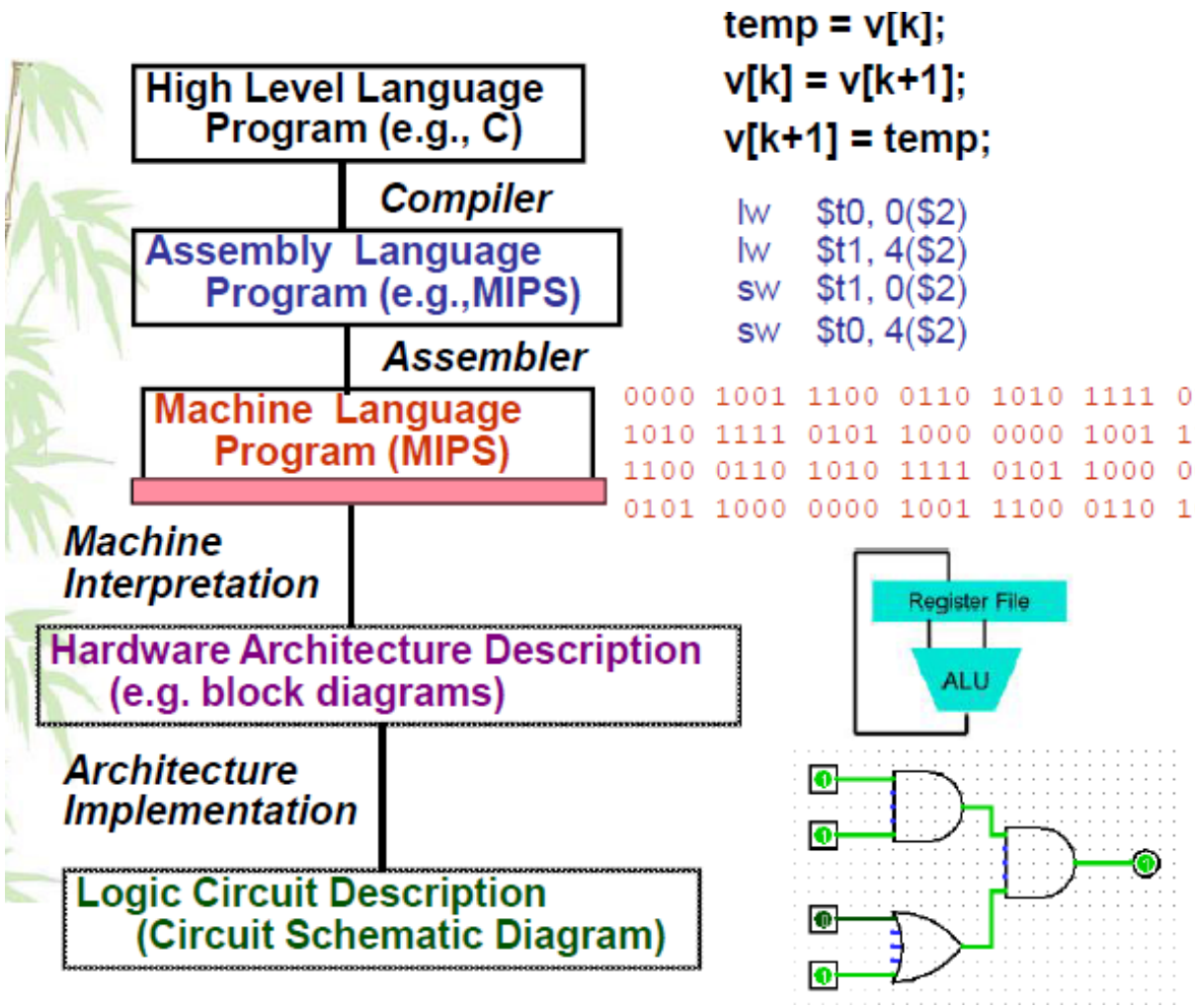
Compiler

- Trình biên dịch ngôn ngữ cấp cao → hợp ngữ
- Compiler phụ thuộc vào:
  - Ngôn ngữ cấp cao được biên dịch
  - Kiến trúc hệ thống phần cứng bên dưới mà nó đang chạy
  - Ví dụ:
    - Compiler cho C <> Compiler cho Java
    - Compiler cho "C on Windows" <> "C on Linux"

Assembler

- Trình biên dịch hợp ngữ → ngôn ngữ máy
- Một bộ vi xử lý (đi kèm 1 bộ lệnh xác định) có thể có nhiều Assembler của nhiều nhà cung cấp khác nhau chạy trên các OS khác nhau
  - *Ví dụ: Cùng là kiến trúc x86, nhưng có thể dùng A86, GAS, TASM, MASM, NASM*
- Assembly program phụ thuộc vào Assembler mà nó sử dụng (do các mở rộng, đặc điểm khác nhau giữa các Assembler)

Mô hình phân tầng các ngôn ngữ trên máy tính



Thảo luận

- Bản thân Compiler cũng là chương trình, vậy nó được biên dịch bằng gì?

→ Assembler

- Sau khi đã biên dịch tập tin mã nguồn ngôn ngữ cấp cao thành tập tin mã máy (machine language), làm sao để chạy những tập tin này trên máy tính?

→ Linker & Loader

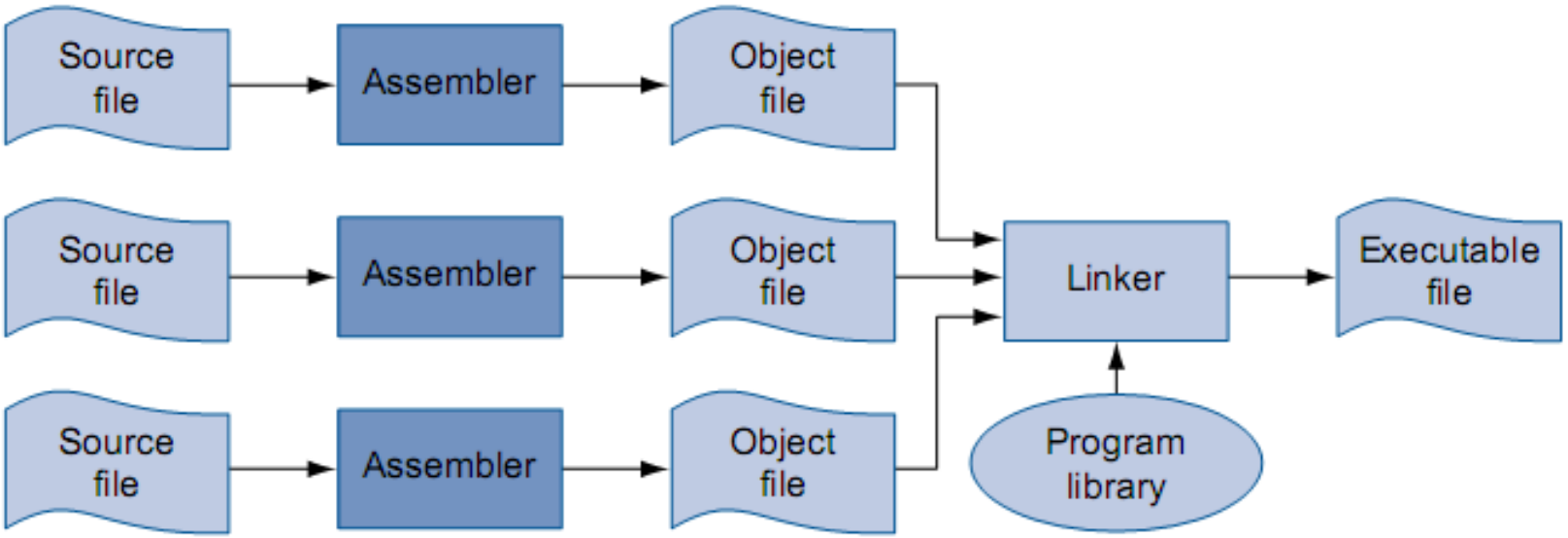
Linker

- Thực tế khi lập trình, ta sẽ dùng nhiều file (header / source) liên kết và kèm theo các thư viện có sẵn

→ Cần chương trình Linker để liên kết các file sau khi đã biên dịch thành mã máy này (Object file)

→ *Tập tin thực thi (ví dụ: .exe, .bat, .sh)*

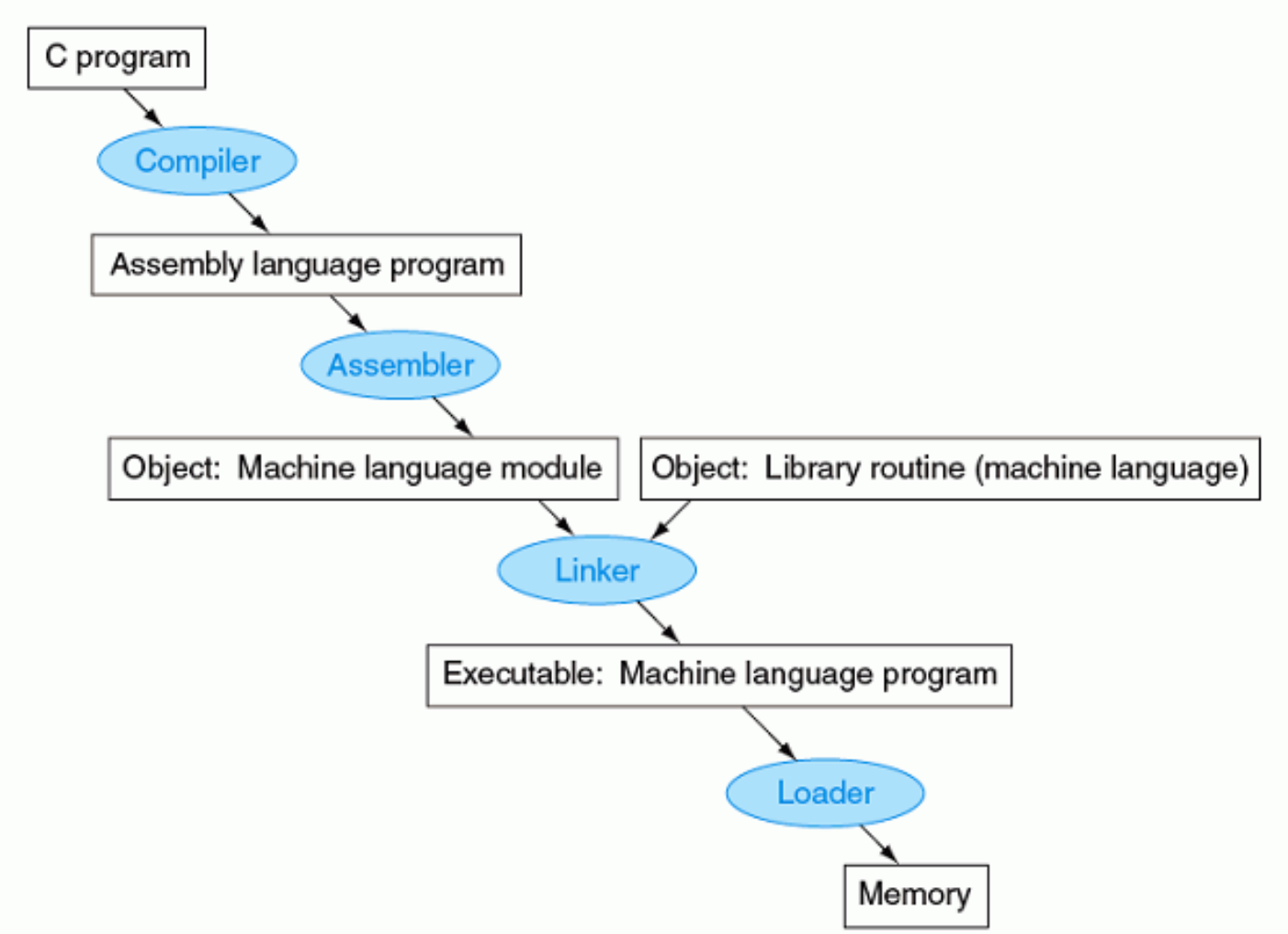
Quá trình tạo file thực thi



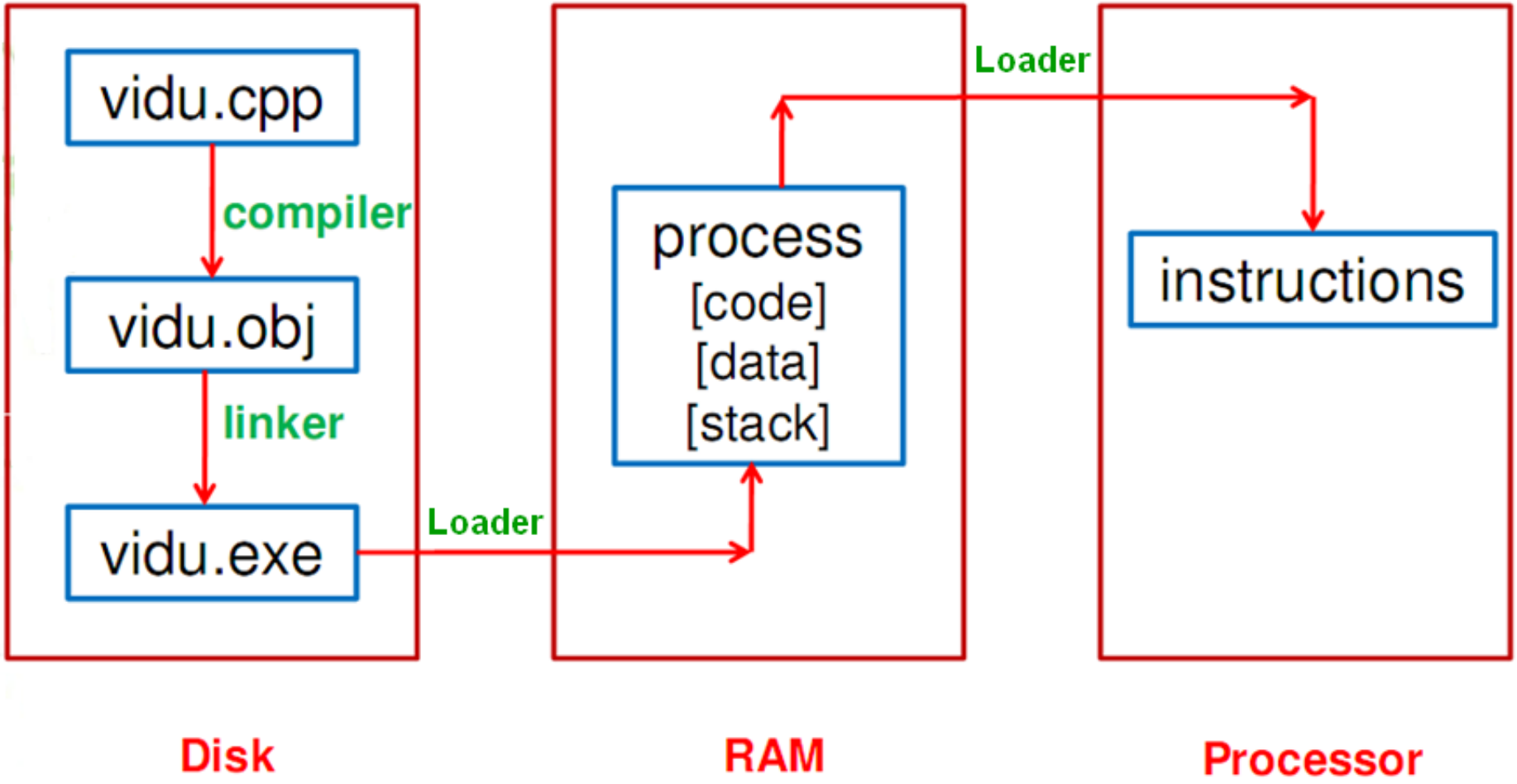
- Khi double click vào những tập tin thực thi, cần chương trình tính toán và tải vào memory để CPU xử lý

→ Loader

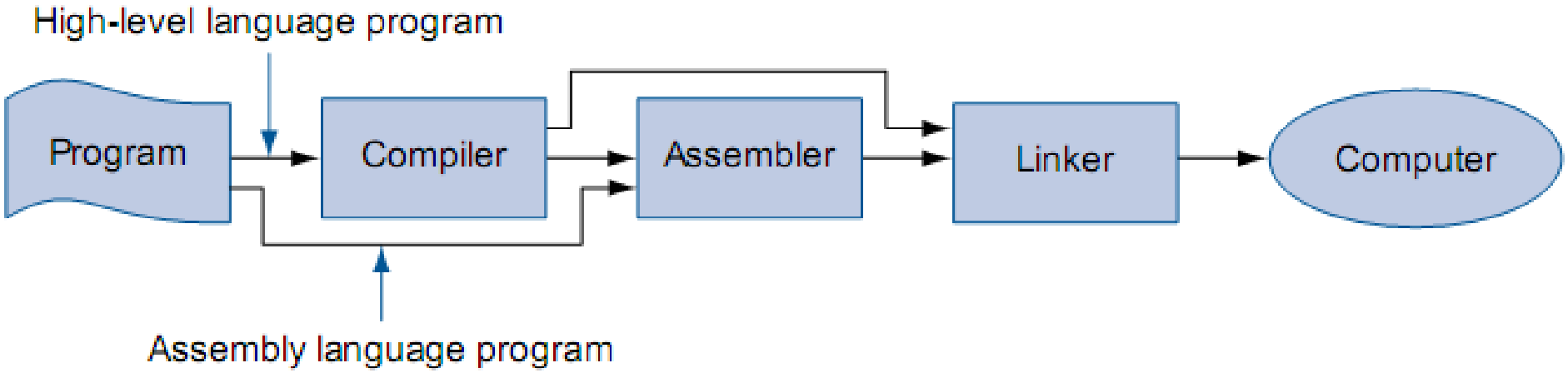
Quá trình thực thi file trên máy



Ví dụ

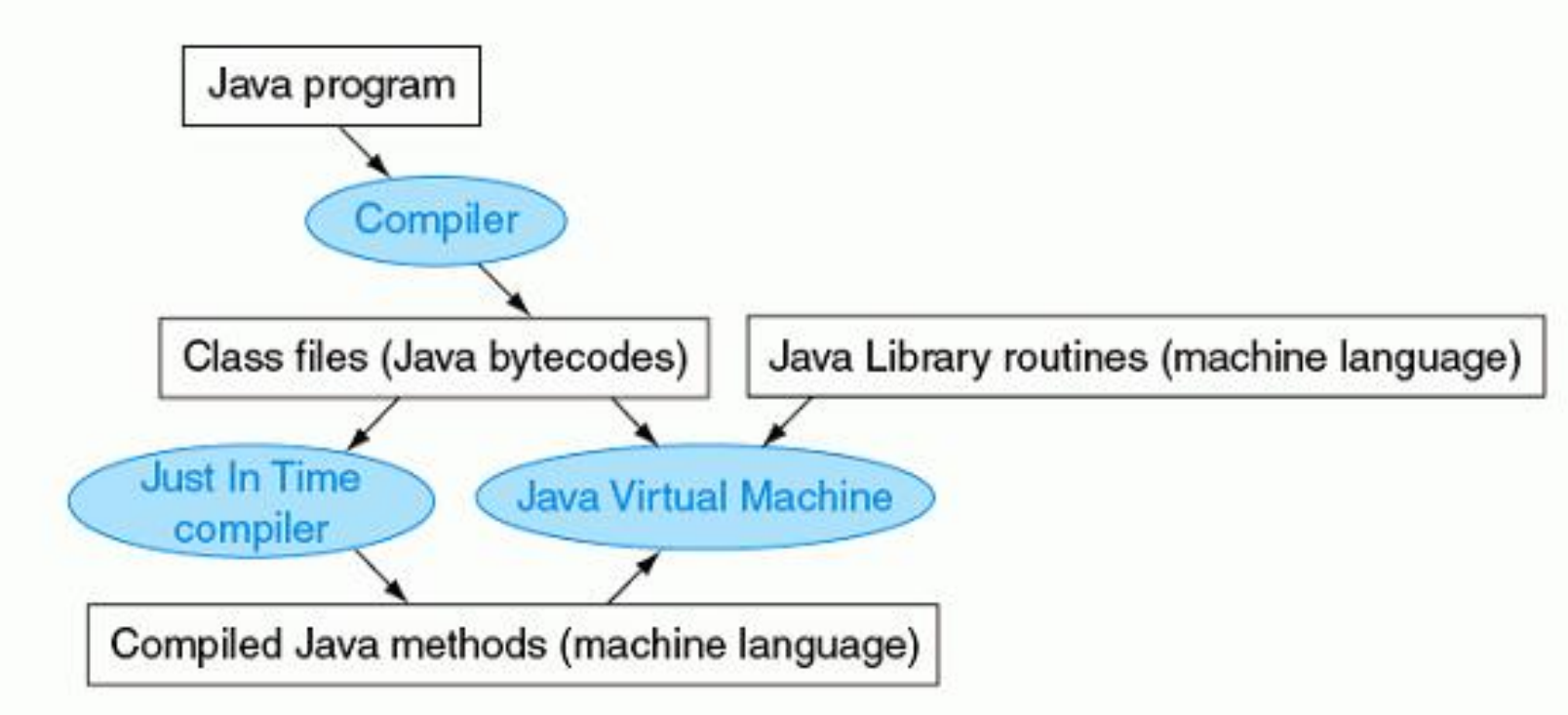

vidu.cpp
compiler
vidu.obj
linker
vidu.exe
Loader
process
[code]
[data]
[stack]
Loader
instructions
Disk
RAM
Processor

Mô hình thực tế



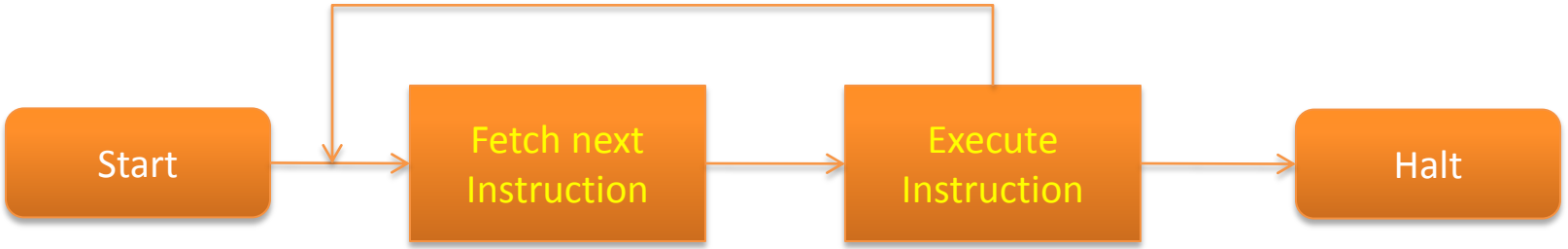
- Compiler và Assembler có thể được bỏ qua trong 1 số trường hợp cụ thể…
- Trong thực tế, có 1 số compiler có thể tạo file thực thi ở nhiều nền tảng kiến trúc bên dưới khác nhau, được gọi là cross-platform compiler
  - Compiler cho Java
  - Cygwin
  - Code::Block Studio

Quá trình thực thi file trên máy Java program



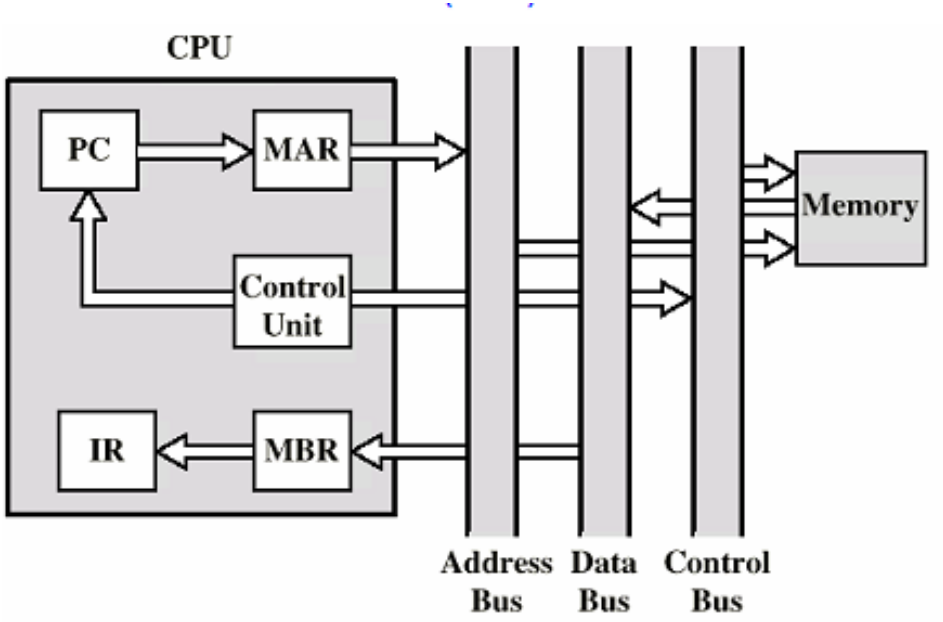
Hoạt động của CPU khi xử lý lệnh

- CPU xử lý lệnh qua 2 bước, gọi là chu kỳ lệnh:
  - Nạp lệnh (Fetch): Di chuyển lệnh từ memory vào thanh ghi (register) trong CPU
  - Thực thi lệnh (Excute): Giải mã lệnh và thực thi thao tác yêu cầu



Quá trình nạp lệnh (Fetch cycle)

- MAR ← PC
- MBR ← Memory
- IR ← MBR
- PC ← PC + 1

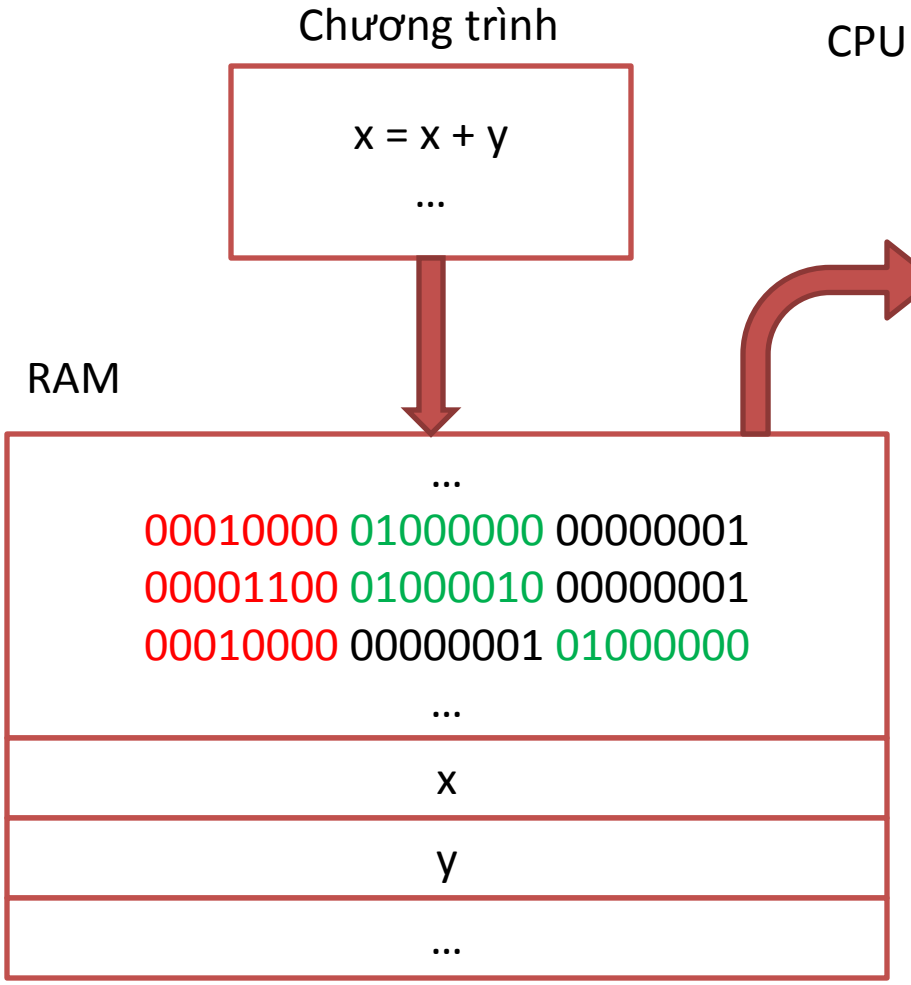


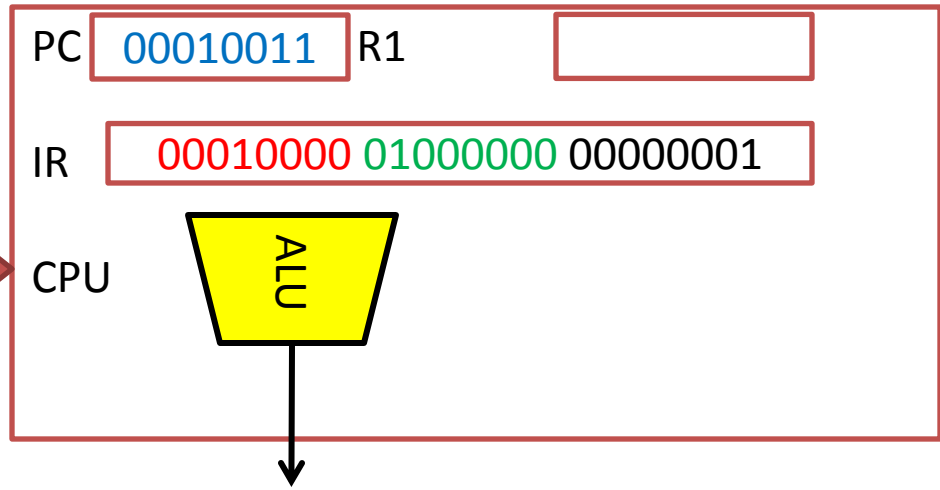
- Thanh ghi PC (Program Counter)
  - Lưu địa chỉ (address) của lệnh sắp được nạp
- Thanh ghi MAR (Memory Address Register)
  - Lưu địa chỉ (address) sẽ được output ra Address bus
- Thanh ghi MBR (Memory Buffer Register)
  - Lưu giá trị (value) sẽ được input / output từ Data bus
- Thanh ghi IR (Instruction Register)
  - Lưu mã lệnh sẽ được xử lý tiếp

• Control Unit di chuyển mã lệnh, *có địa chỉ trong PC*, vào thanh ghi IR

• Mặc định, giá trị thanh ghi PC sẽ tăng 1 lượng = chiều dài của lệnh vừa được nạp

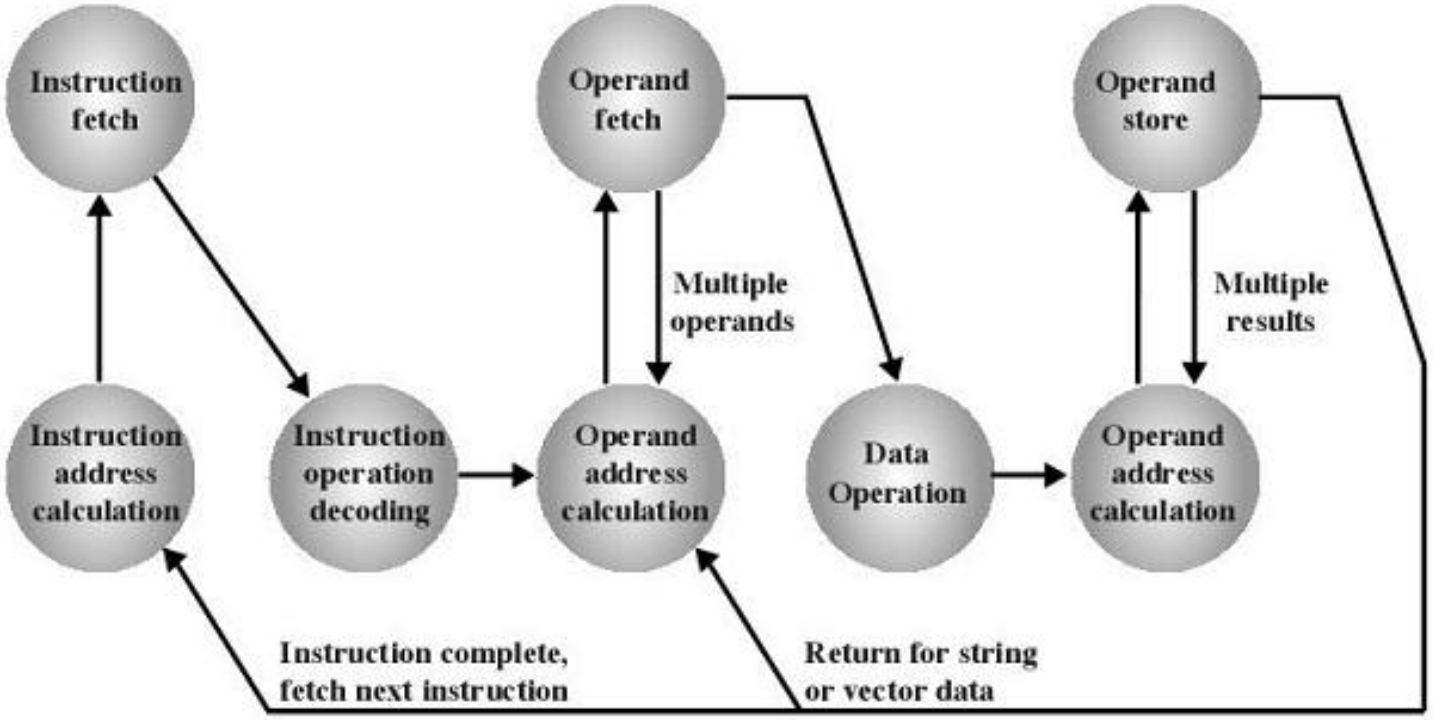
Ví dụ





Địa chỉ	Lệnh (Instruction)
00010000	x → R1
00010011	R1+y → R1
00010110	R1 → x
....	
01000000	....
01000010	....

Quy trình thực thi lệnh (Execute Cycle)



- Tính địa chỉ lệnh
- Nạp lệnh
- Giải mã lệnh
- Tính địa chỉ của toán hạng
- Nạp toán hạng
- Thực hiện lệnh
- Tính địa chỉ của toán hạng chứa kết quả
- Ghi kết quả

- Các bước này được lặp đi lặp lại cho tất cả các lệnh tiếp theo
- Quy trình này gọi là Instruction cycle – vòng lặp xử lý lệnh

Một số câu hỏi

- Ngôn ngữ lập trình giống và khác ngôn ngữ tự nhiên của con người ở những điểm nào?
- Tại sao cần nhiều loại ngôn ngữ lập trình: C, C++, C#, VB, Java…?
- Một chương trình không khai báo biến nào cả thì có sử dụng bộ nhớ không?
- Chương trình được thực thi trong RAM hay CPU?
- Tại sao file .exe chỉ chạy được trên Windows mà không thể chạy trên Linux?

Homework

- Sách Petterson & Hennessy: Đọc chương 2 (đọc kỹ 2.12 và 2.13)
- Tài liệu tham khảo: Đọc "08_HP_AppA.pdf"