PROCESSOR

KHTN
TP. HO CHI MINH
TRUONG DAI HOC KHOA HOC TU NHIEN

cdio
4.0

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

fit@hcmus

REMIND

☐ Inside a CPU

☐ Abstraction layer

What will you learn?

☐ How programs are translated into the machine language

☐ How hardware executes a program

☐ How CPU process an instruction

☐ Measuring execution time

☐ Uniprocessor vs Multiprocessor

Instruction

☐ The sequence bit that contains the request that the processor must make.

☐ An instruction consists of 2 part:

Opcode: the operation ALU must take

Operand: objects affected by the action contained in the code
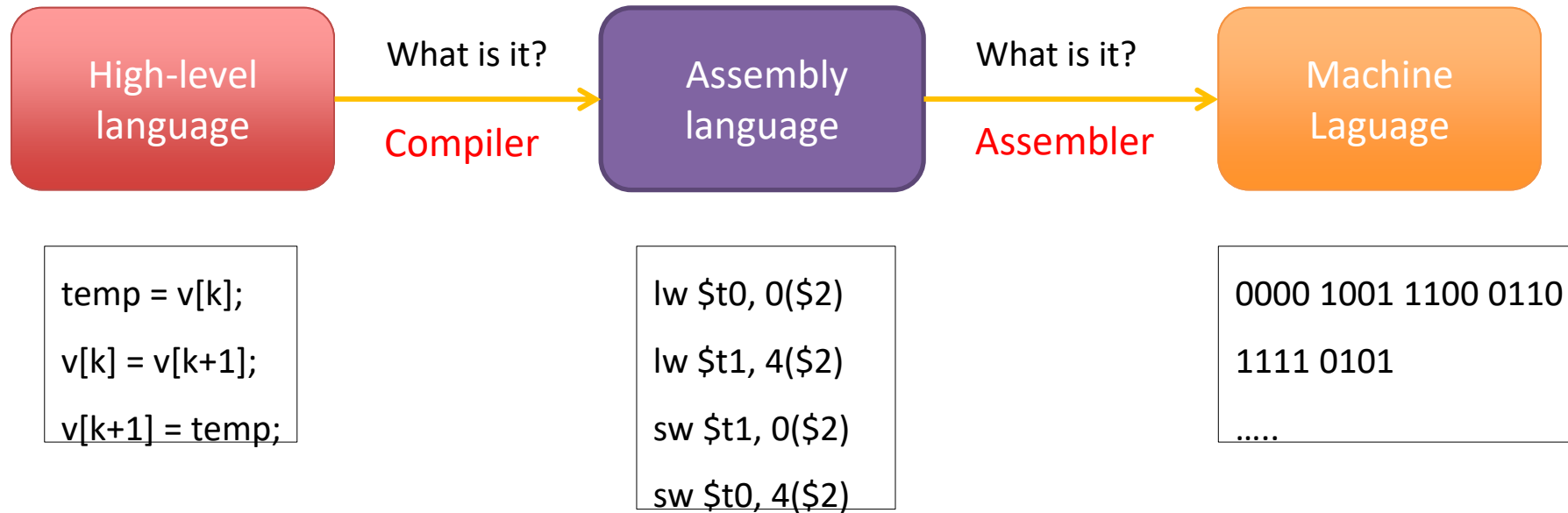
Instruction Set Architecture (ISA)

☐ The format and behavior of a machine-level program is defined by the instruction set architecture

☐ Different computers have different instruction sets but with many aspects in common

☐ Commonly ISA:

- MIPS: used in embedded system
- ARM: A64, A32, T32
- Power-PC

- IA-16: 16-bits processor (Intel 8086, 80186, 80286)
- IA-32: 32-bits processor (Intel 80368 – i386, 80486 – i486, Pentium II, Pentium III …)
- IA-64: 64-bits processor (Intel x86-64 - Pentium D...)

ISA design: CISC & RISC

☐ Complete Instruction Set Computer (CISC): includes many instructions, from simple to complex

☐ Reduced Instruction Set Computer (RISC): consists of only simple instructions

➔ *Which one is better?*

Discussion

High-level language	→ What is it? Compiler →	Assembly language	→ What is it? Assembler →	Machine Laguage

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

.....

Assembly Language

☐ A symbolic representation of machine code, clearer than in machine code

☐ Each assembly instruction represents exactly one machine instruction

☐ Ex: Save the value 5 decimal in the register $4

Machine language: 00110100 0000100 00000000 0000101

Assembly: ori $4, $0,   5

                   opcode dest reg   src reg
                   immediate

Assembly Language

☐ Since each processor has its own register structure and instruction set when setting the assembly, it must be clear which processor is set, or the family of the processor.

☐ Ex:

☐ Assembly for MIPS

☐ Assembly for the line of Intel 8086 processors

Compiler

☐ A program that translates high-level language statements into assembly language statements

☐ Belong to:

   ☐ The system hardware architecture below which it is running

   ☐ The high-level language which it compiles

☐ Ex:

Compiler for C <> Compiler cho Java

Compiler for "C on Windows" <> "C on Linux"

Assembler

☐ A program that translates a symbolic version of instructions into the machine code

☐ A single processor (1 set of definitions) can have multiple assemblers from different vendors running on different operating systems.

☐ Ex: list of assembler for x86 architecture

 A86, GAS, TASM, MASM, NASM

☐ The Assembly program depends on the assembler it uses

Discussions

☐ Who will compile the compiler? (It's also a program)

➔ Assembler

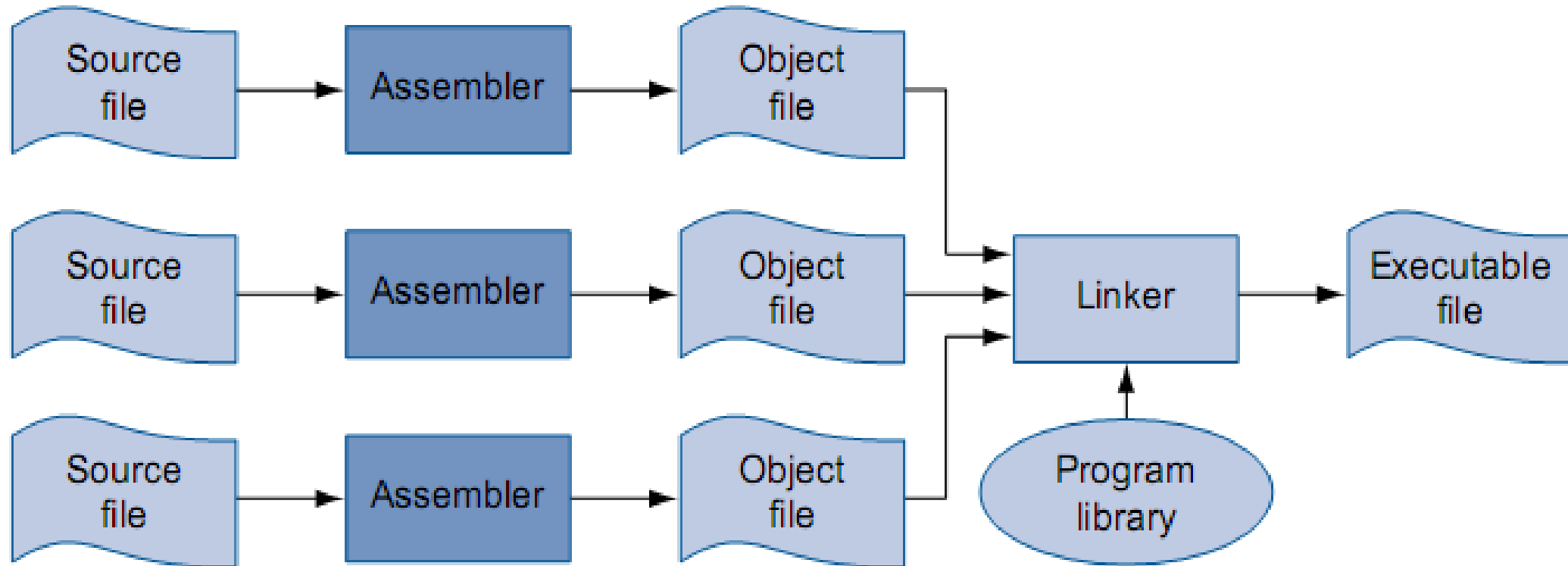☐ How the hardware execute a program?

➔ Loader & Linker

Linker

☐ A systems program that combines independently assembled machine language programs (object file) and resolves all undefined labels into an executable file.
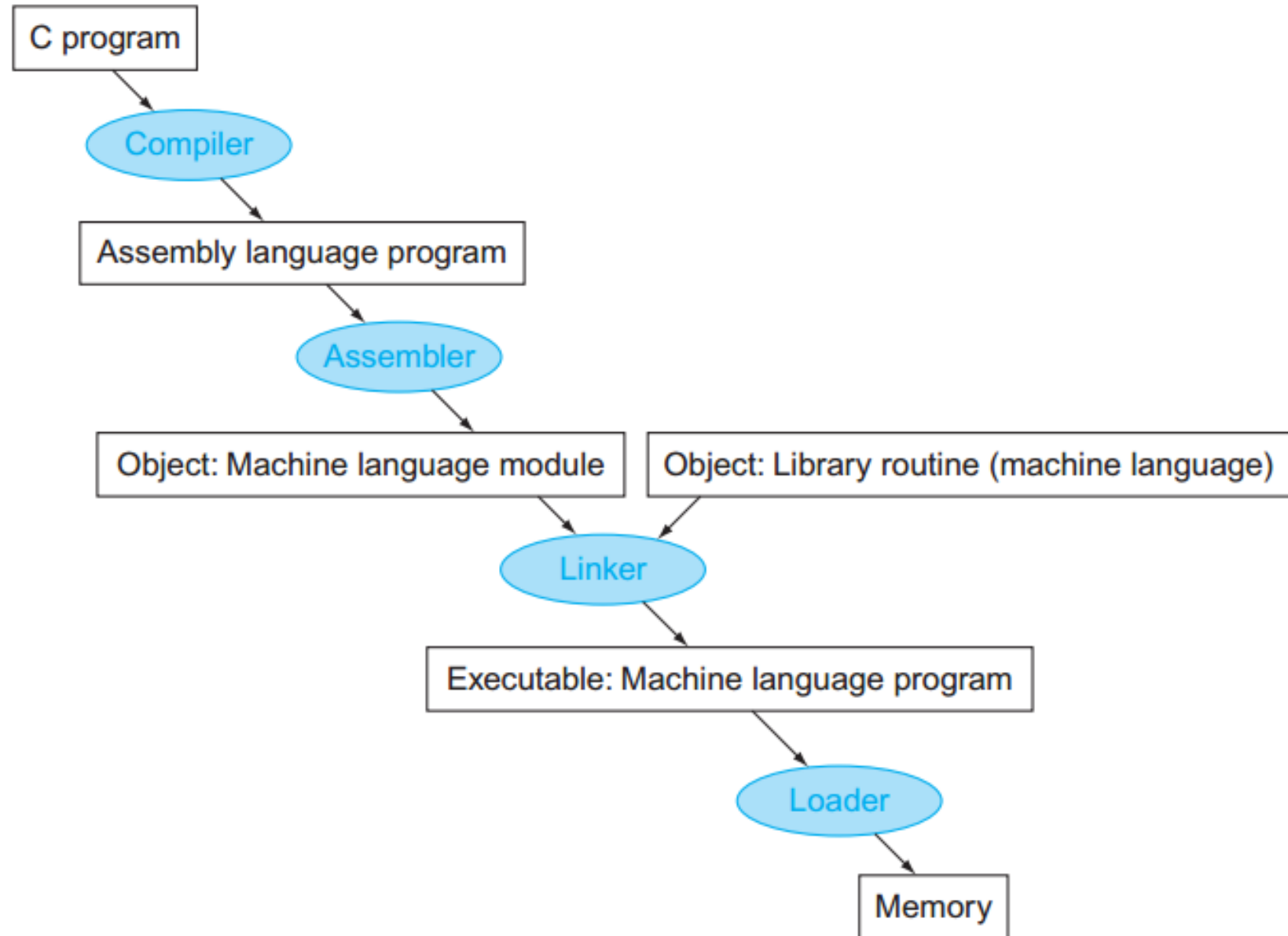
Loader

☐ A systems program that places an object program in main memory so that it is ready to execute.
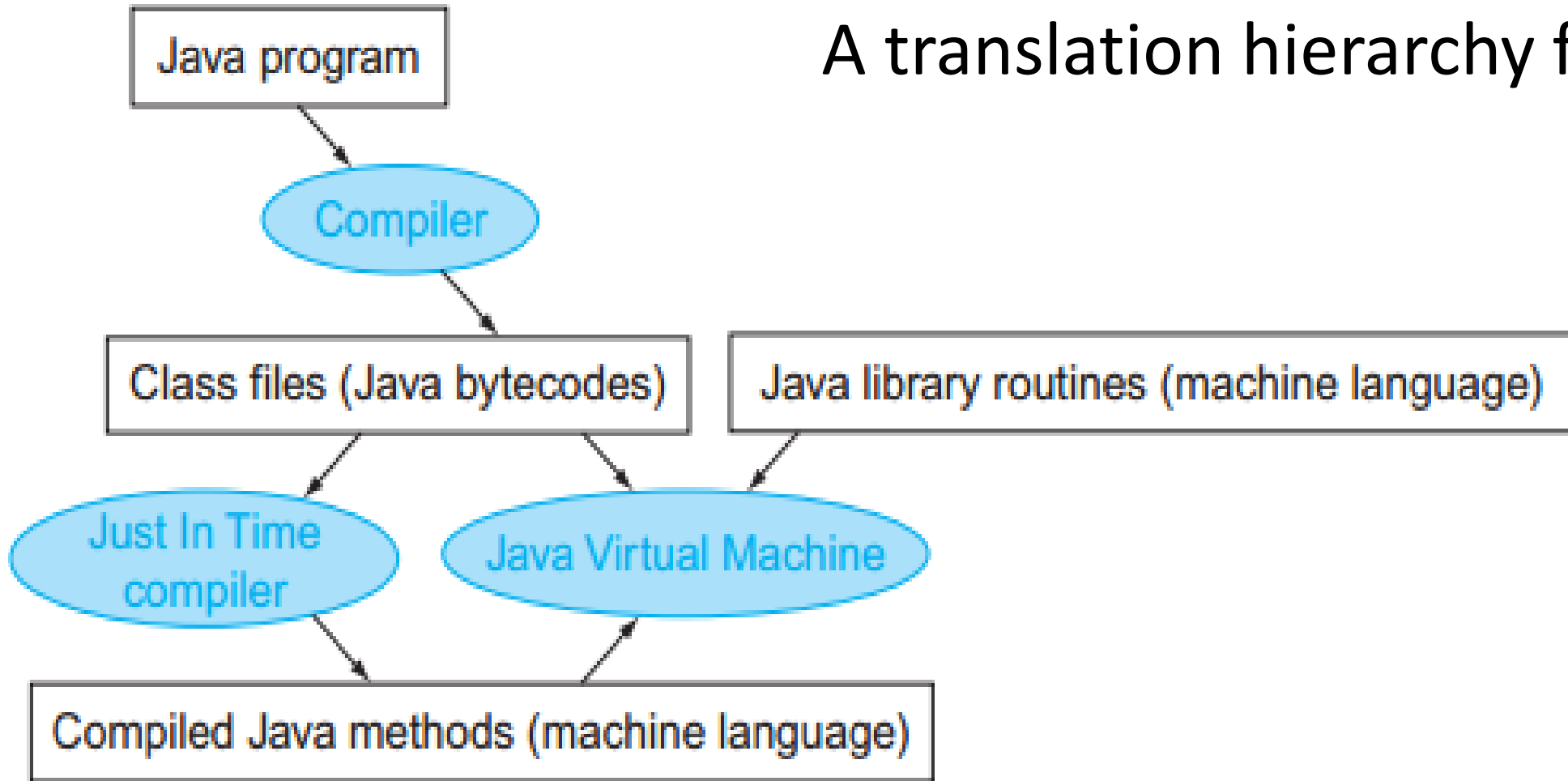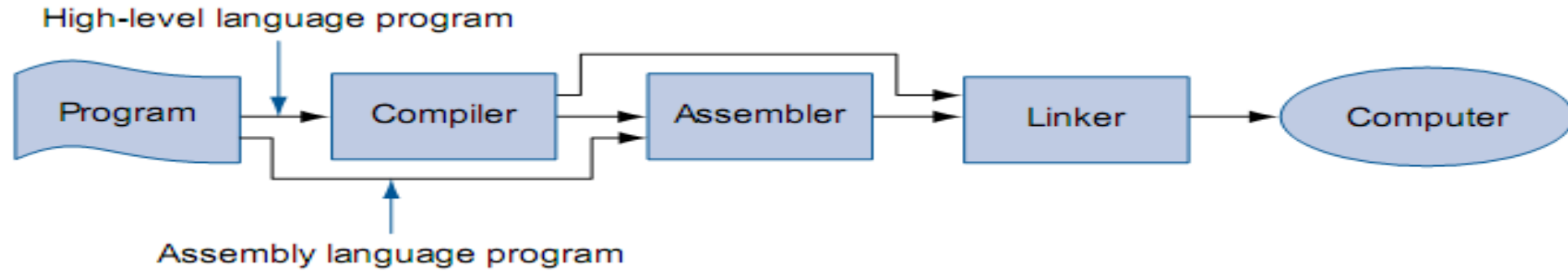
Create executable file

A translation hierarchy for C
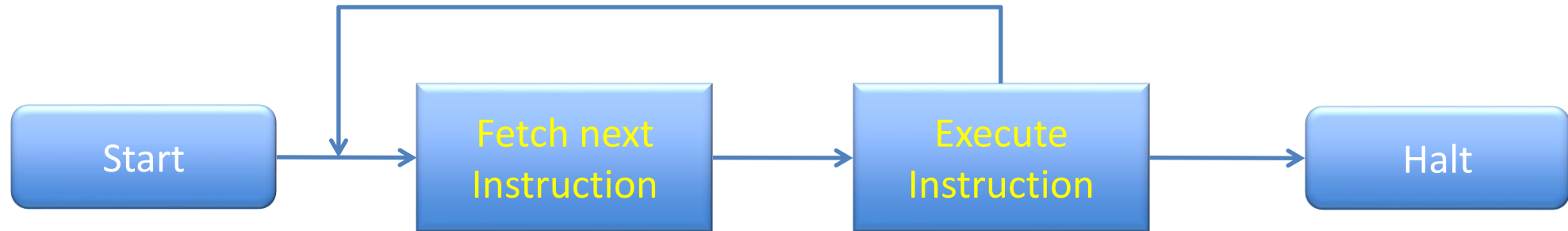
A translation hierarchy for Java

Realistic Model

High-level language program

Program → Compiler → Assembler → Linker → Computer

Assembly language program

- Compiler and assembler can be skipped in the certain cases
- In fact, there are several compilers that can create executables on a variety of underlying architecture platforms (cross-platform compiler)
- Ex: Compiler for Java, Cygwin, Code::Block Studio

Instruction processing

```
            ┌──────────────────────────────────────┐
            │                                      │
            ↓                                      │
Start  →  Fetch next     →   Execute      →   Halt
          Instruction          Instruction
```

Instruction Cycle: consists of 2 phases

- Fetch cycle: Transfer data from memory to registers

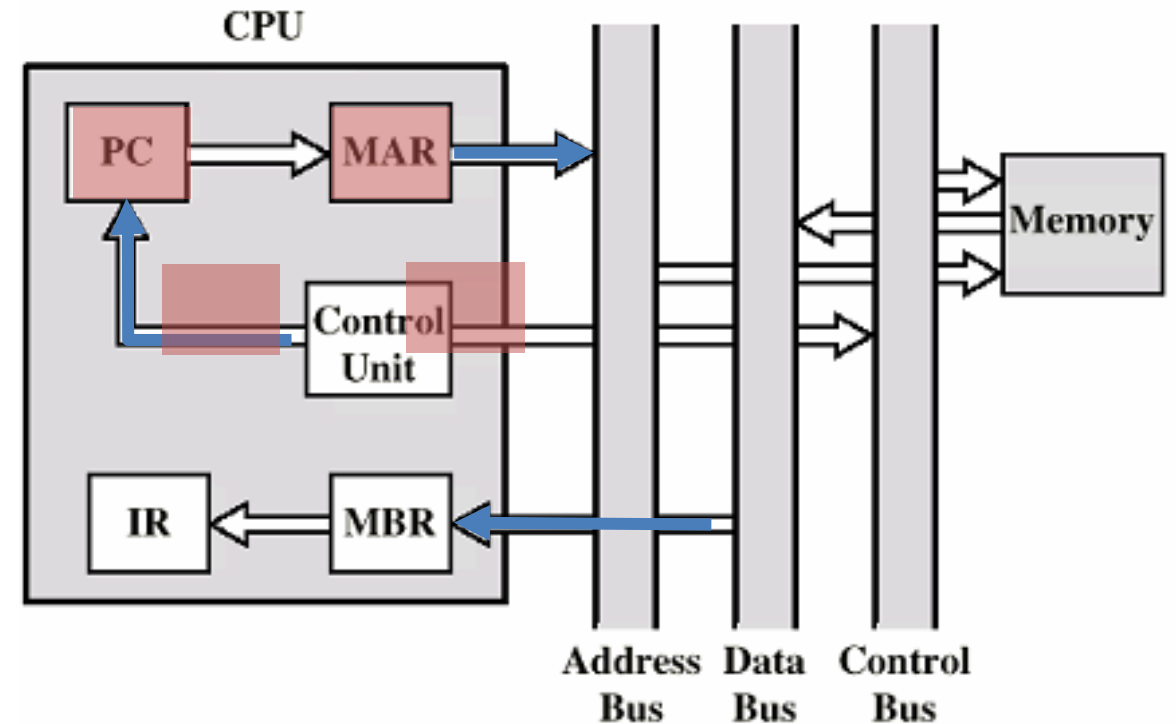- Execute cycle: Decode the instruction and execute the requirements of it

Fetch cycle

- **PC (Program Counter)**
  - ☐ Store the next instruction's address
- **MAR (Memory Address Register)**
  - ☐ Store the address of a location in memory (output to address bus)
- **MBR (Memory Buffer Register)**
  - ☐ A word of data to be written to memory or the word most recently read (output to data bus)
- **IR (Instruction Register)**
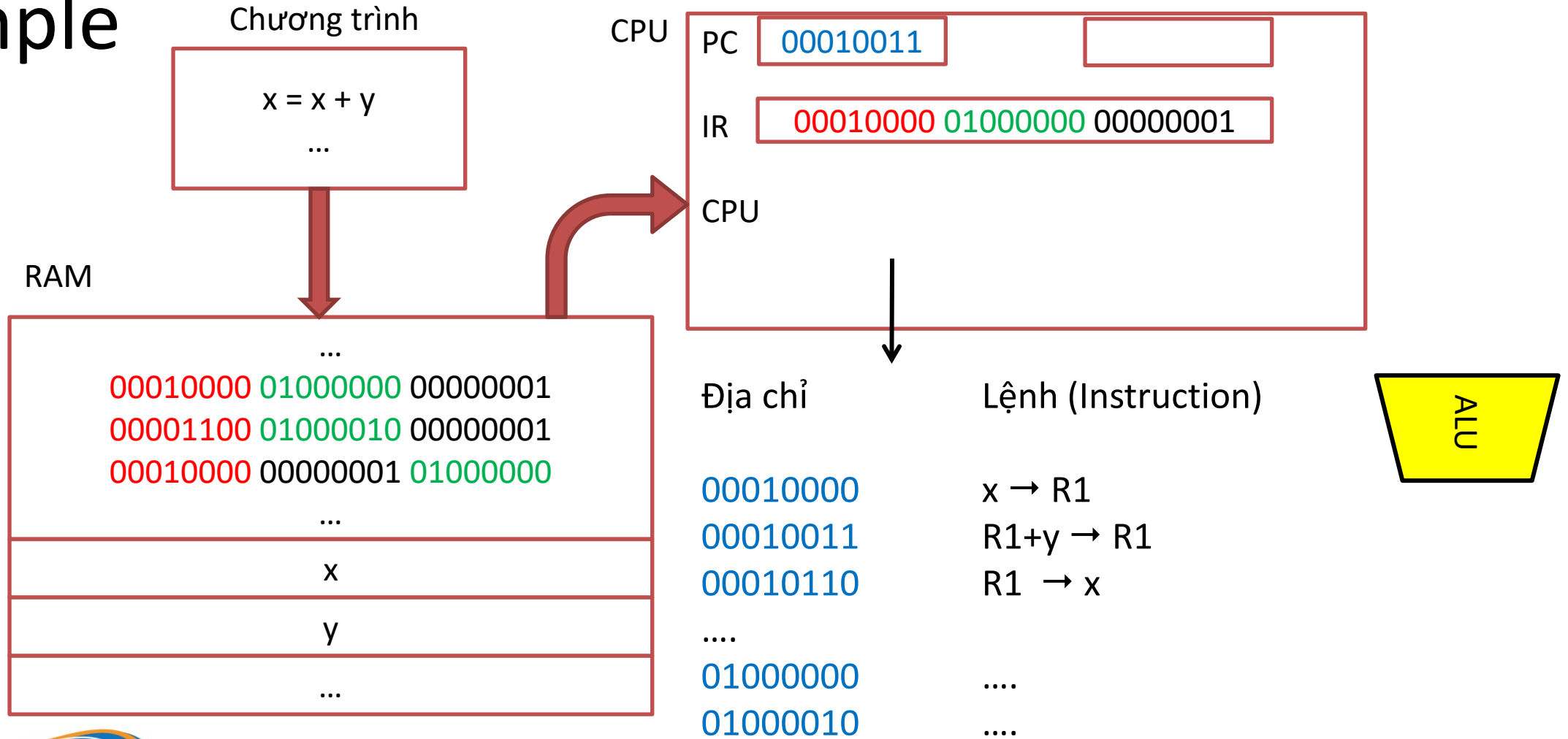 - ☐ Contain the most recently fetched instruction
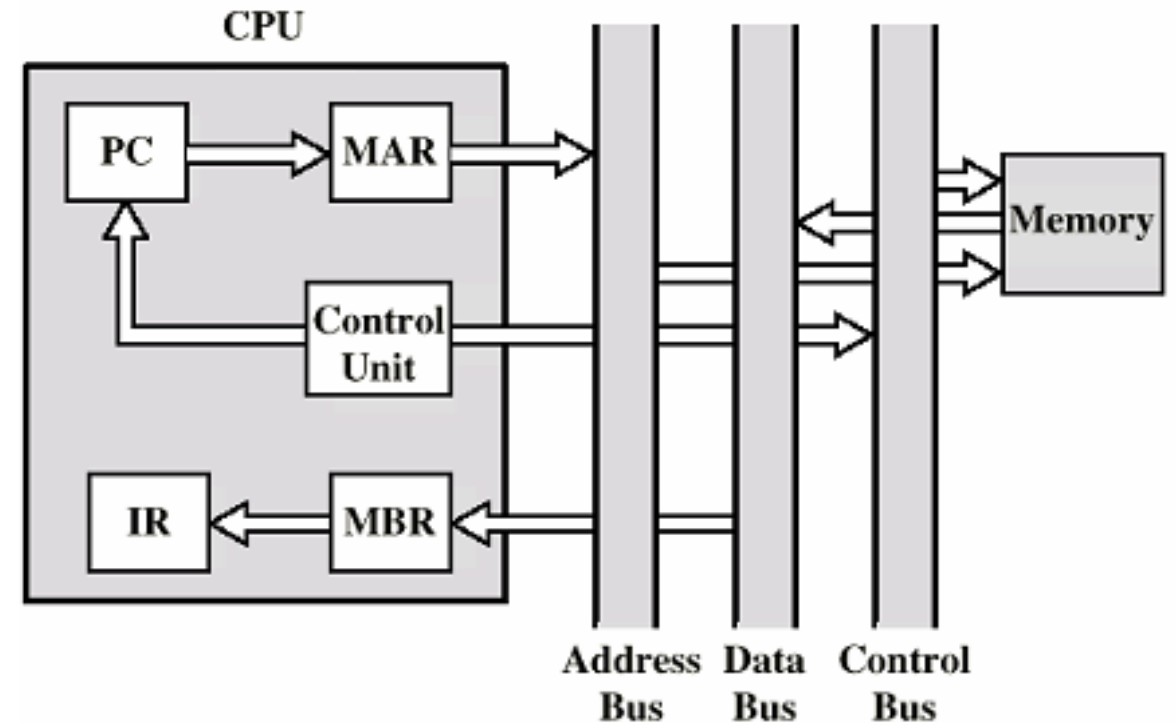
Fetch cycle
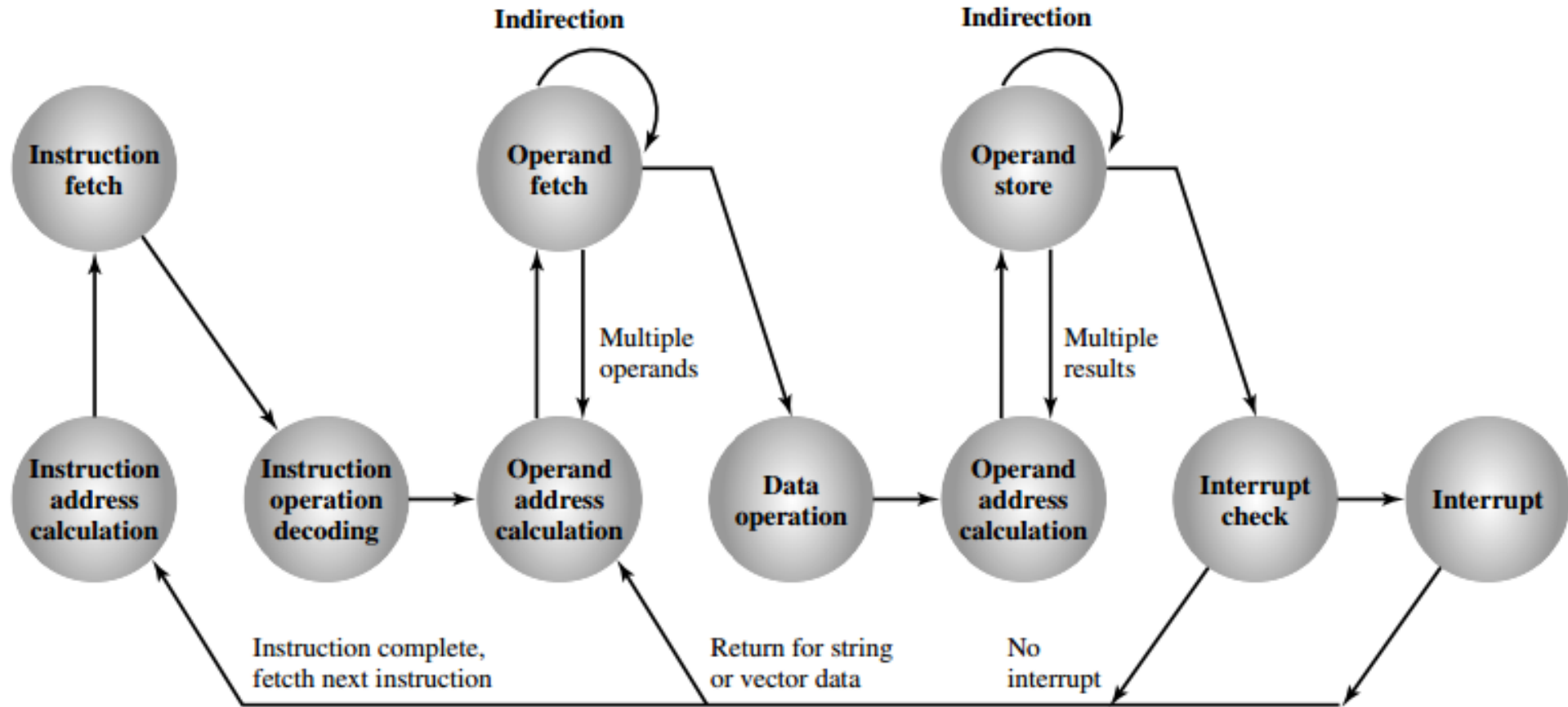
- The control unit move the instruction which has address in PC regs to IR

→Default update PC reg:

PC += size of the fetched instruction

- The fetched instruction is loaded into an IR, where the opcode and operand are analyzed

- Data are exchanged with memory using the MAR and MBR

Example

Chương trình

x = x + y
…

RAM

… 00010000 01000000 00000001 00001100 01000010 00000001 00010000 00000001 01000000
…
x
y
…

CPU

PC 00010011

IR 00010000 01000000 00000001

CPU

ALU

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

Execute cycle

Chap3, William Stalling, **Computer Organization and Architecture, 8th ed, 2014** Figure 3.6

Elapsed time

- Total response time, including all aspects: processing, i/o, idle time, OS overhead

CPU time

- Time spent processing a given task

- Comprise user CPU time and system CPU time

- Different program are affected differently by CPU and system performance

fit@hcmus

Instead of reporting execution time in seconds, we often use *cycles*. In modern computers hardware events progress cycle by cycle: in other words, each event, e.g., multiplication, addition, etc., is a sequence of cycles

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

cycle time = time between ticks = seconds per cycle
clock rate (*frequency*) = cycles per second (1 Hz. = 1 cycle/sec, 1 MHz. = 10^6 cycles/sec)

CPU Time

$$\text{CPU time} = \frac{Instructions}{Program} \times \frac{CPU\ clock\ cycles}{Instruction} \times \frac{seconds}{CPU\ clock\ cycles}$$

Performance depends on:
- Algorithm
- Programming language
- Compiler
- ISA

Uniprocessor vs Multiprocessor

Constrained by:

    Power

    Instruction-level parallelism

    Memory latency

Multicore microprocessors (>1 processor/ chip)

Requires explicitly parallel programming

- Compare with instruction level parallelism
  - o Hardware executes multiple instructions at once
  - o Hidden from the programmer
- Hard to do
  - o Programming for performance
  - o Load balancing
  - o Optimizing communication and synchronization