

Chapter 2:

Data Manipulation

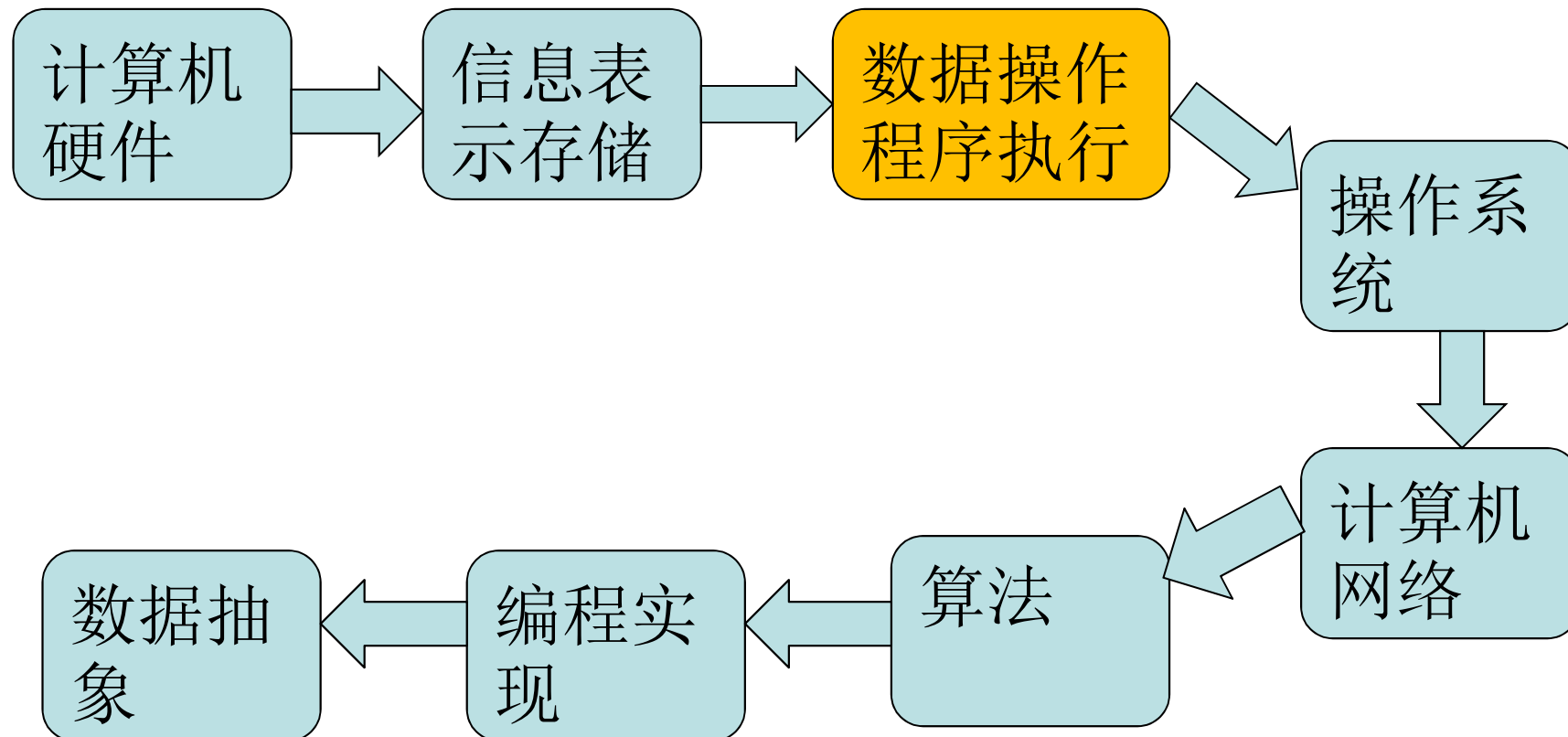
Computer Science: An Overview
Tenth Edition

by
J. Glenn Brookshear



Copyright © 2008 Pearson Education, Inc. Publishing as Pearson Addison-Wesley

从程序的角度了解计算机系统



What is a Computer?

- Monitor, case, keyboard, mouse, speaker, scanner, webcam, printer, ...



What's inside?

Inside the Case

- CPU, motherboard, adaptors, hard disk, memory, CDROM, ...



We are going to talk about those.

Central Processing Unit (CPU)

- An electronic circuit that can execute **computer programs**

- Intel i7
- AMD K10
- IBM Cell
- ARM Acorn
- Sun SPARC



- To understand CPU, we need to know what **computer programs** are.

Stored Program Concept

*"The final major step in the development of
the general purpose electronic computer
was the idea of a stored program..."*

Brian Randell

What're the Differences?



TV: you can watch different channels.

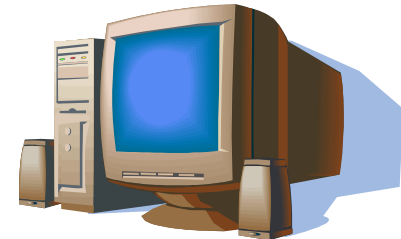


面包机: you can make different food.

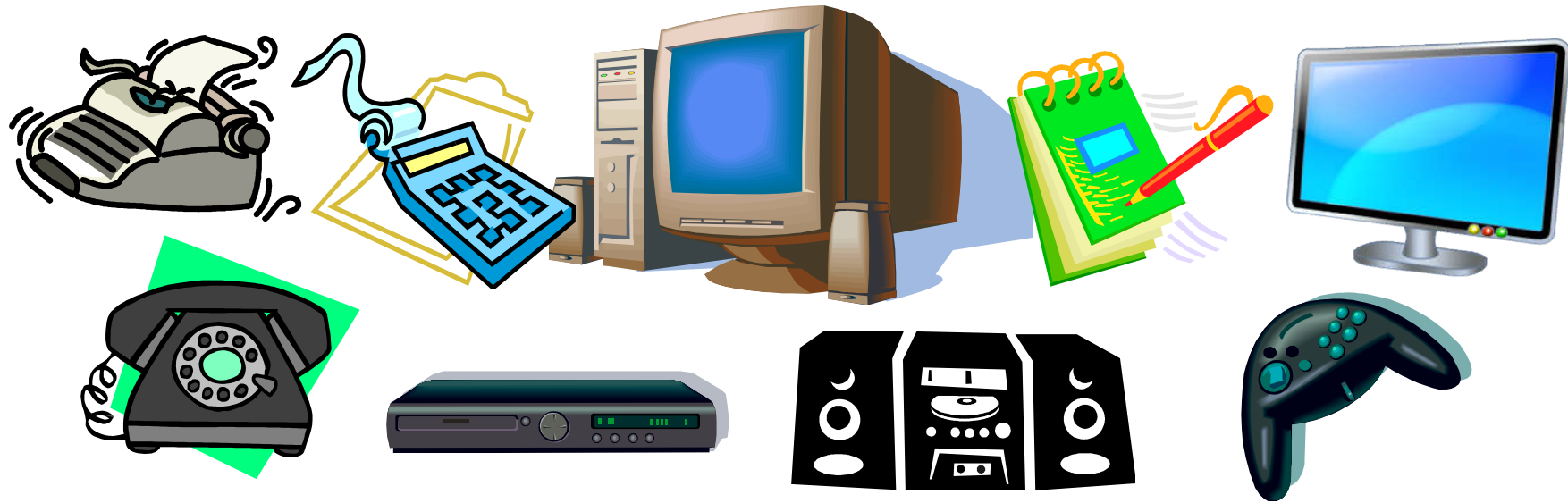


Swiss knife: you can use different tools

Computer: you can ...

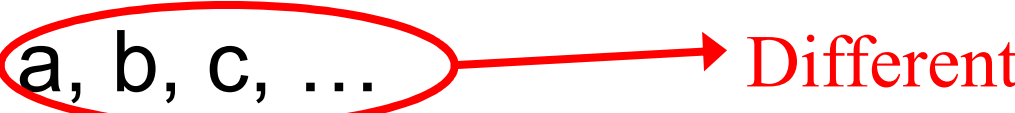


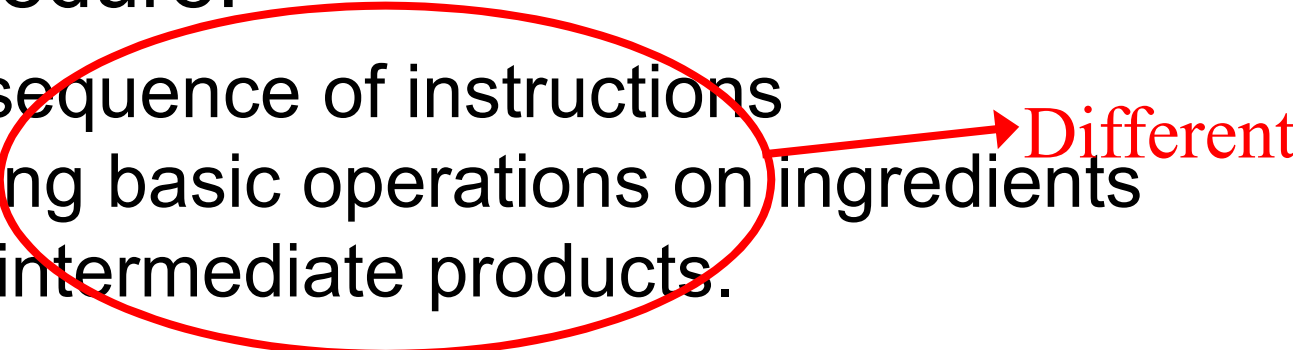


Magic box



- You can add more functions to it. How?
 - Program is like data to be input to computers.
- It can perform multiple functions at a time
 - We will talk about this in the OS lesson.

A Generic Recipe

- Ingredient: a, b, c, ...  Different
- Tools: 锅、炉、刀...  The same
- Basic operations: 切、洗、炒、煮... 
- Procedure:
 - A sequence of instructions using basic operations on ingredients or intermediate products.  Different
- Output: dish x

First Try

- e.g. 面包机

Fixed procedures
to choose from



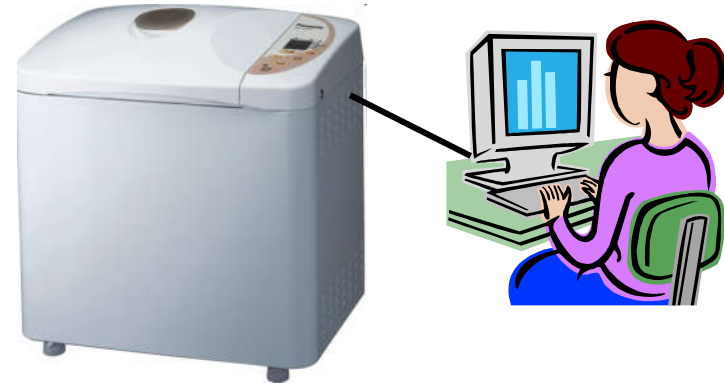
How about Programmable?

- How to tell the machine to do the procedure that we invent?



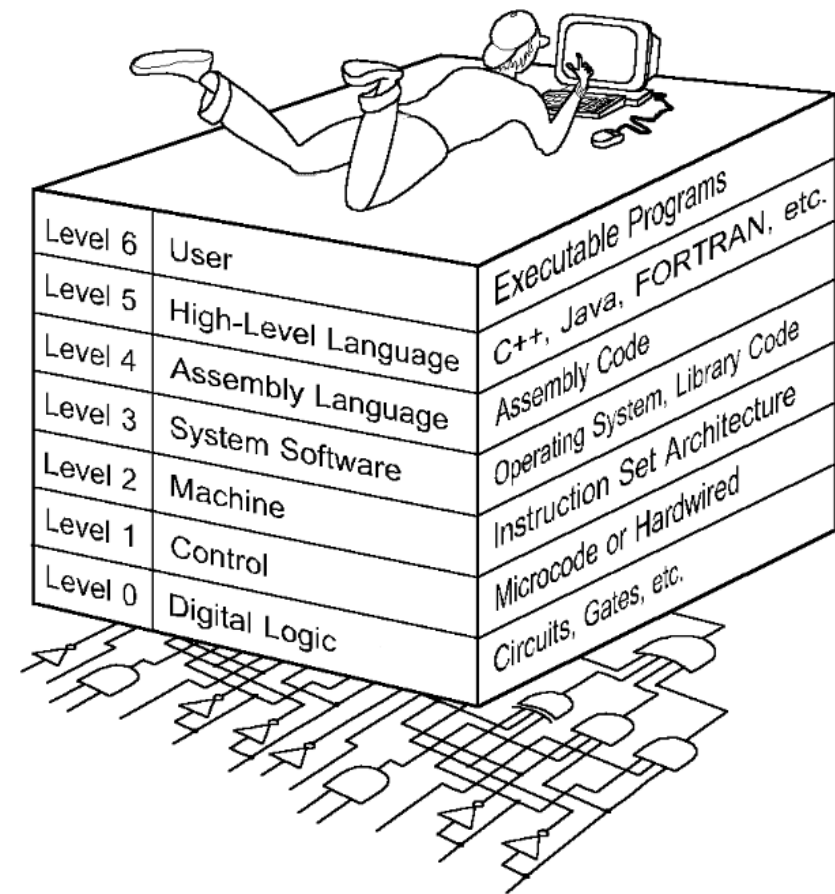
Ideal: A Universal Cooking Machine

- Input:
 - Ingredient a, b, c, ...
 - Instruction 1, 2, 3, ...
- Tool: universal cooking machine
 - Can **read** instructions and **execute** them step by step → *programmable*.
 - Have all **tools** and **ability** to perform basic operations.
- Output: dish x



For Computers

- Data are stored as 0 and 1
- Instructions are also expressed and stored as 0 and 1
→ in memory



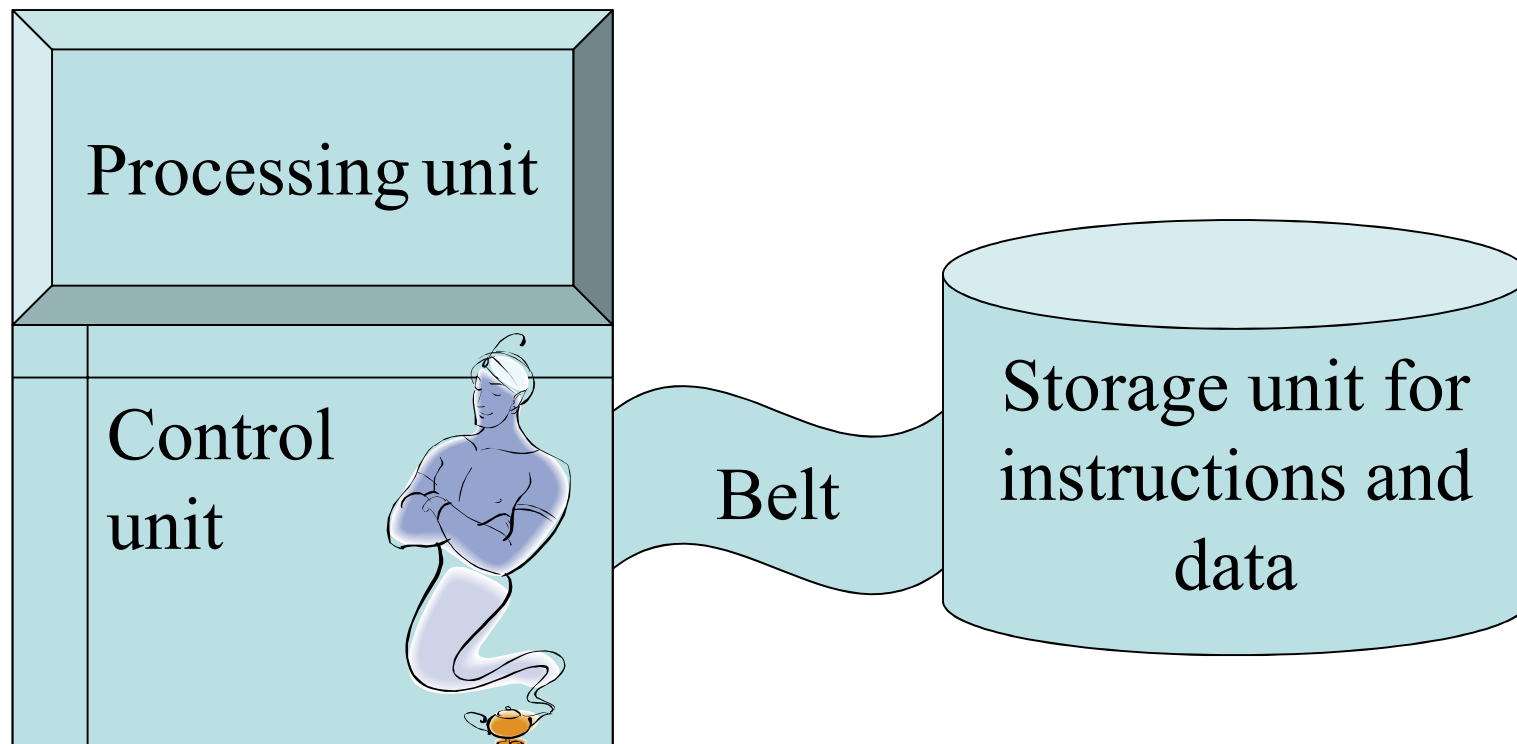
Stored-Program Concept

- Program: a sequence of instructions
- *Stored-program concept:*
 - A program can be encoded as **bit patterns** and stored in main memory, just like data.
 - From there, the CPU can fetch the instructions and execute them.
- Advantage: programmable
 - We can use a **single** machine to perform different **functions** by loading different prog.

Problems

- How to convert instructions to operations?
 - This is like Harry Porter's spell.
- There should be a control unit.
 - To control which function to perform.
 - To control which data to be operated.
 - How can the control unit understand the instructions?
- What function units should be included?
 - CD players, game console, calculators, ...?

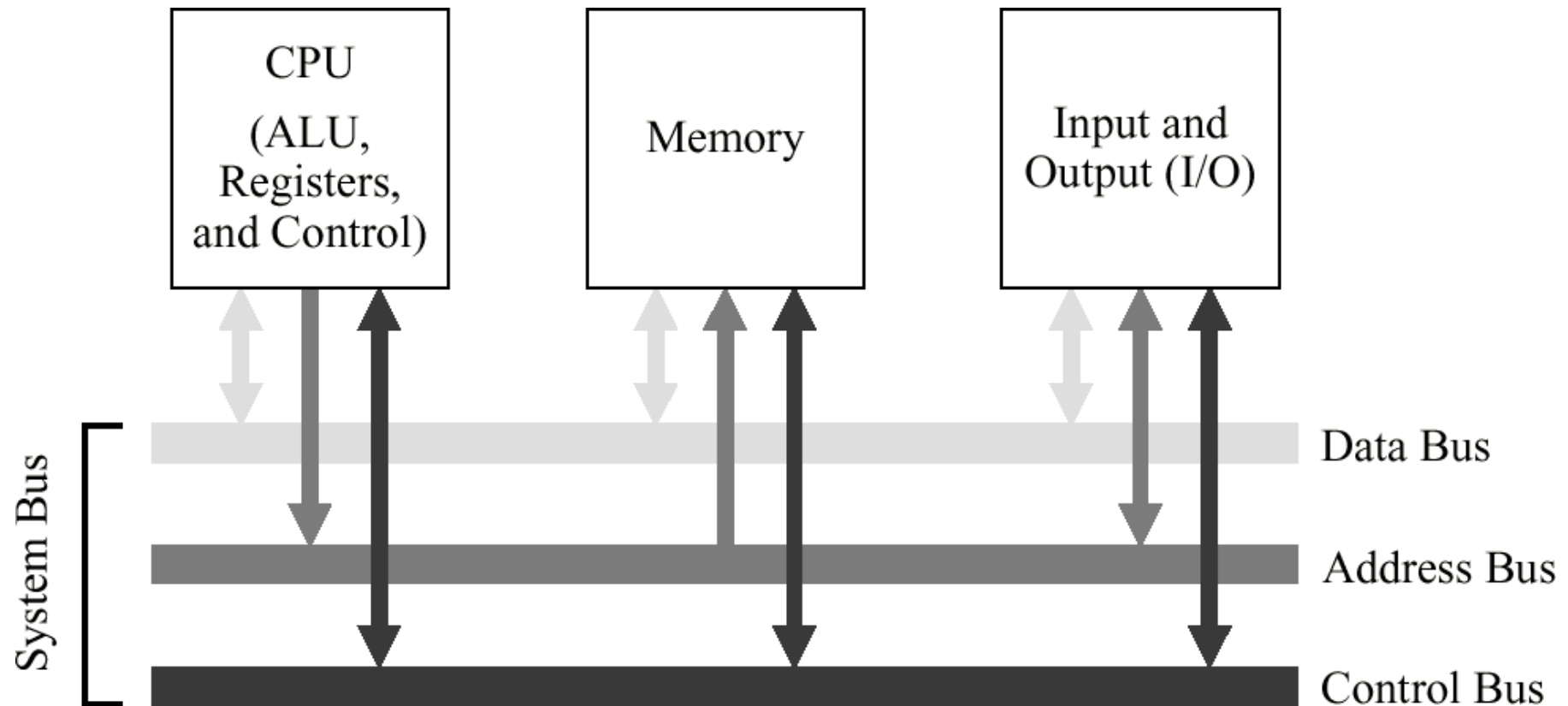
Outline of the Magic Box



Chapter 2: Data Manipulation

- 2.1 Computer Architecture
- 2.2 Machine Language
- 2.3 Program Execution
- 2.4 Arithmetic/Logic Instructions
- 2.5 Communicating with Other Devices
- 2.6 Other Architectures

Von-Neumann Model



Computer Architecture

- Central Processing Unit (CPU) or processor
 - Arithmetic/Logic unit versus Control unit
 - Registers（寄存器）
 - General purpose（通用）
 - Special purpose（专用）
- Bus
- Motherboard

Von Neumann Architecture

- General purpose electronic computer

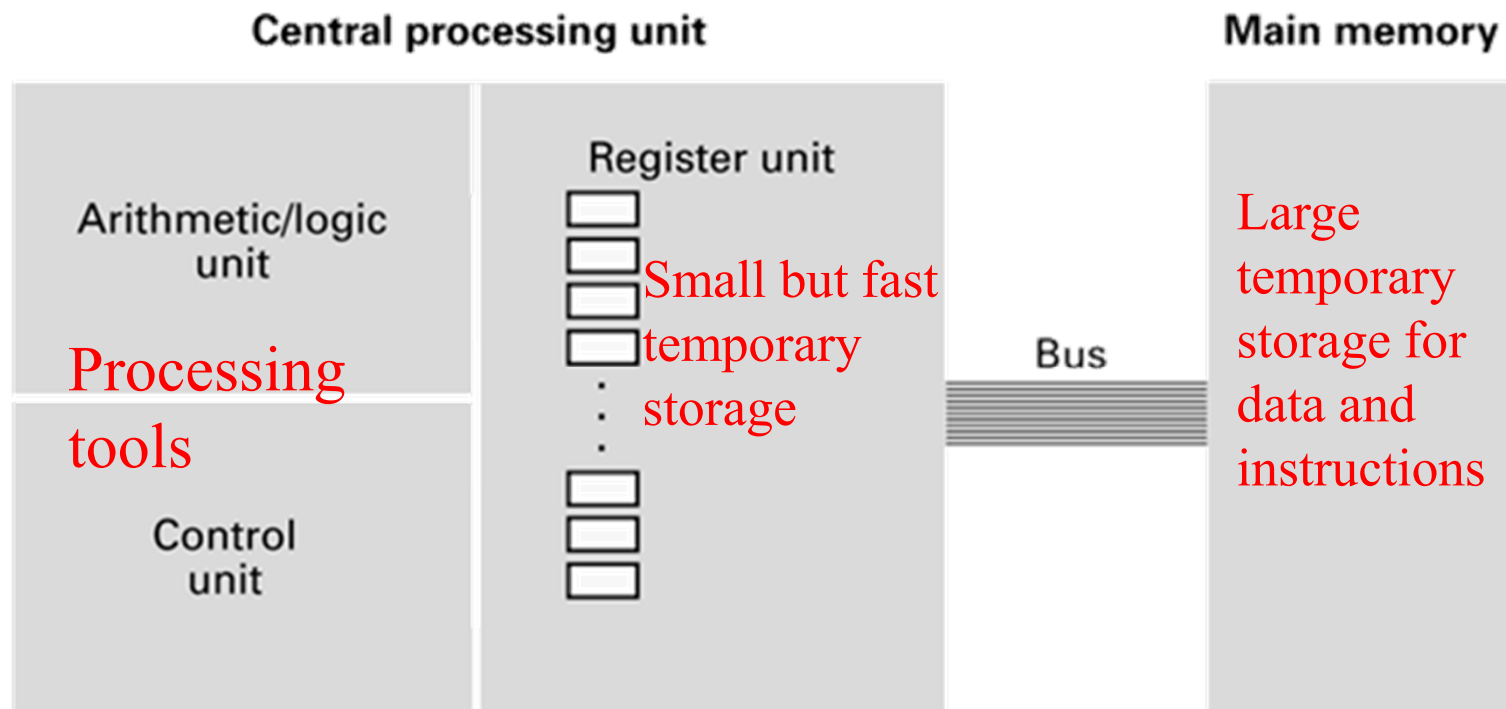
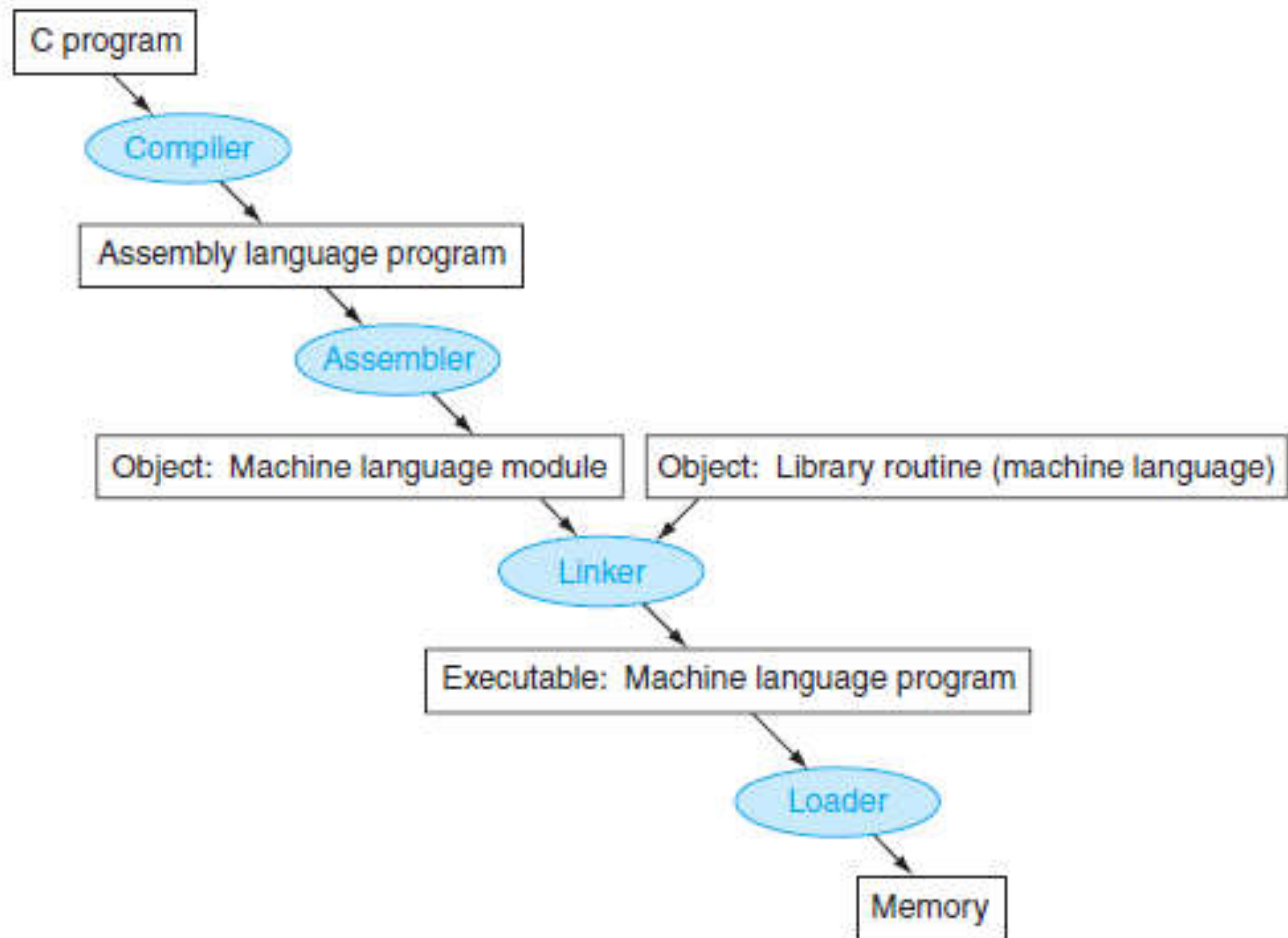


Fig. 2.1

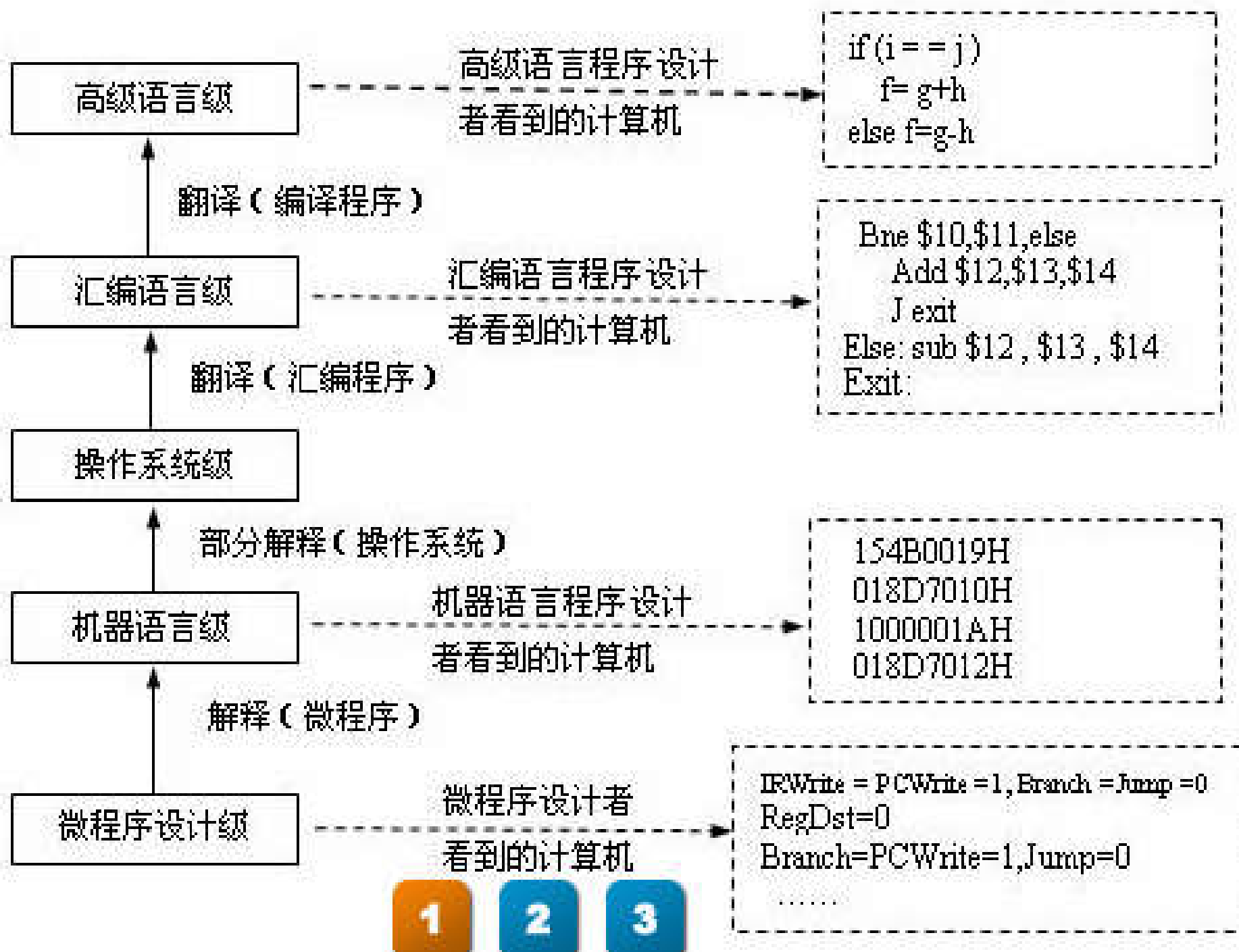
Stored Program Concept

A program can be encoded as bit patterns and stored in main memory. From there, the CPU can then extract the instructions and execute them. In turn, the program to be executed can be altered easily.



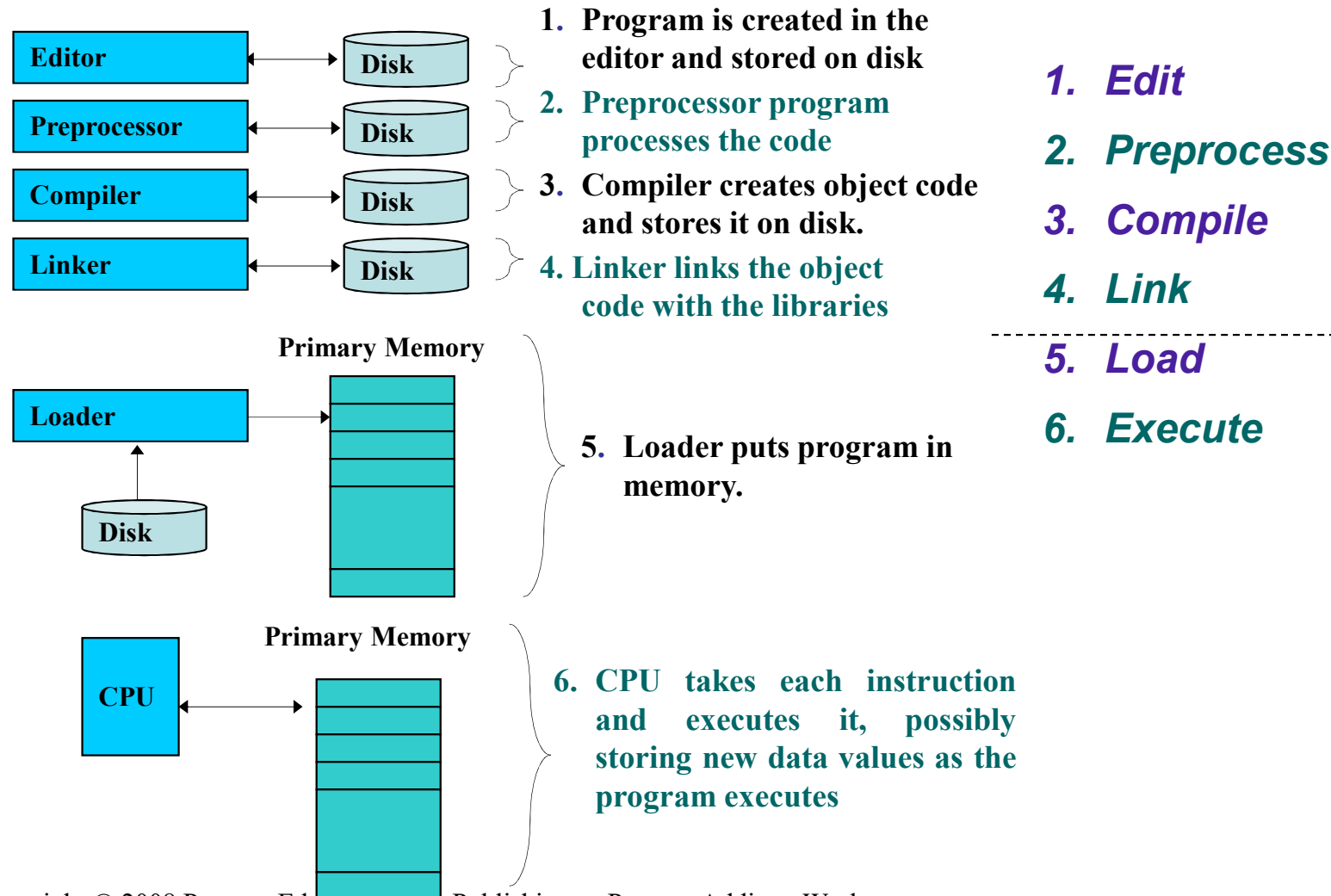
Some concepts

- ❖ **Instruction set (指令集)**: collection of instructions that a processor can execute
- ❖ A **compiler (编译器)** translates a high level language, which is architecture independent, into assembly language, which is architecture dependent.
- ❖ An **assembler (汇编器)** translates assembly language programs into executable binary codes.
- ❖ For fully compiled languages like C and Fortran, the binary codes are executed directly by the target machine.



A Typical C Program Development Environment

• Phases of C Programs:



```

;CLEAR SCREEN USING BIOS
CLR: MOV AX,0600H      ;SCROLL SCREEN
    MOV BH,30          ;COLOUR
    MOV CX,0000        ;FROM
    MOV DX,184FH       ;TO 24,79
    INT 10H            ;CALL BIOS;
;INPUTTING OF A STRING
KEY: MOV AH,0AH        ;INPUT REQUEST
    LEA DX,BUFFER      ;POINT TO BUFFER WHERE STRING STORED
    INT 21H            ;CALL DOS
    RET                ;RETURN FROM SUBROUTINE TO MAIN PROGRAM;
; DISPLAY STRING TO SCREEN
SCR: MOV AH,09         ;DISPLAY REQUEST
    LEA DX,STRING      ;POINT TO STRING
    INT 21H            ;CALL DOS
    RET                ;RETURN FROM THIS SUBROUTINE;

```

Assembly code

Assembler

```

0001010010110101010101010101010100010
1110110101010101010101010101010100010
001010010101001011101011101011101010
100101001011010101010101010101010110
011010010011001011101011101010100010
000100010101110101010100010101011010
10101001010100101010101101011101011
00010100101101010101010101010100010

```

Object code

- Different for various processor type (size, kind of operation, type of operands, type of results)
- Incompatibility: in contrast to high-level language (re-compile)
- One exception: Java bytecodes for virtual machine , run on any processor
- Computer systems are often identified by CPU
- Instruction set decides the program that can be executed and affect the performance
- Instruction sets: 80x86 CPU (e.g. IBM PC) ~ Motorola PowerPC CPU (e.g. Apple Macintosh)

Terminology

- **Machine instruction:** An instruction (or command) encoded as a bit pattern recognizable by the CPU
- **Machine language:** The set of all instructions recognized by a machine

What to do

+

Specified information

Machine Language Philosophies

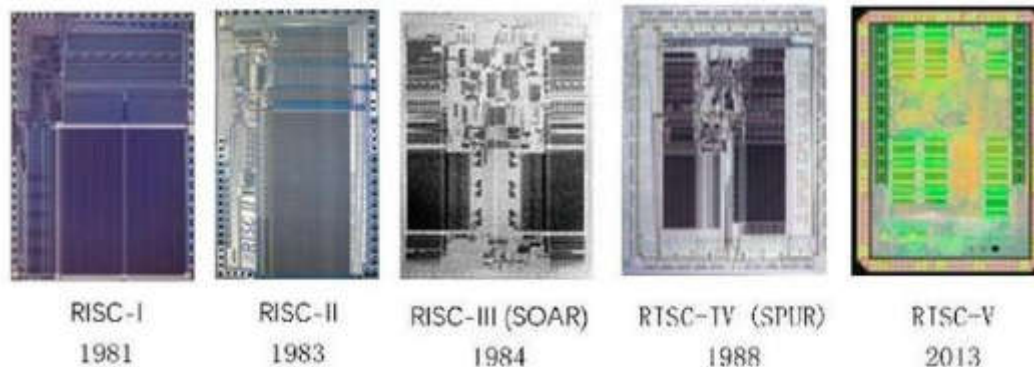
精简指令集

- Reduced Instruction Set Computing (RISC)
 - Few, simple, efficient, and fast instructions
 - Examples: PowerPC from Apple/IBM/Motorola and SPARK from Sun Microsystems
- Complex Instruction Set Computing (CISC)
 - Many, convenient, and powerful instructions
 - Example: Pentium from Intel

复杂指令集

开源指令集RISC-V

- 2010年，加州大学伯克利分校的一个研究团队为新项目选架构
- Arm、MIPS、SPARC和x86指令集越来越复杂，还有很多法律问题
- RISC-V指令集架构简单、完全开源并且免费，将基准指令和扩展指令分开，可以通过扩展指令做定制化的模块和扩展
- RISC-V已经有多个单位加入，包括谷歌、华为、英伟达、高通、麻省理工学院、普林斯顿大学、印度理工学院、中科院计算所等。
- RISC-V不仅为我们提供了一个庞大且快速发展的社区，而且还给予了我们架构进行创新以达到极致能效的机会，无需购买昂贵的架构许可证
- 三星研发中的这个CPU内核基于开源的RISC-V指令集架构，而不是我们常见的ARM架构
- NVIDIA和高通已经在使用RISC-V架构开发自己的物联网处理器和GPU内存控制器



阿里系芯片公司中天微发布中国自研CPU架构RISC-V处理器

- 2018杭州中天微系统有限公司宣布，正式推出支持RISC-V第三代指令系统架构处理器CK902，可灵活配置TEE引擎，支持物联网安全功能
- GreenWaves推出基于RISC-V处理器的开发板，旨在边缘端（传感器端）依赖电池做长期的复杂运算，如行人监测，人脸检测，语音关键字识别，震动识别等等



Machine Instruction Types

- **Data Transfer**: copy data from one location to another
- **Arithmetic/Logic**: use existing bit patterns to compute a new bit patterns
- **Control**: direct the execution of the program

MIPS Instructions

- http://en.wikipedia.org/wiki/MIPS_architecture

Instruction class	MIPS examples	HLL correspondence
Arithmetic	add, sub, addi	operations in assignment statements
Data transfer	lw, sw, lb, sb, lui	references to data structures, such as arrays
Logical	and, or, nor, andi, ori, sll, srl	operations in assignment statements
Conditional branch	beq, bne, slt, slti	<i>if</i> statements and loops
Jump	j, jr, jal	procedure calls, returns, and <i>case/switch</i> statements

Arithmetic instructions

- add:
 - Adds two registers and stores the result in a register, executes a trap on overflow

– add \$d, \$s, \$t

$\$d = \$s + \$t$

Comments

- sub
 - subtracts two registers, executes a trap on overflow
 - sub \$d,\$s,\$t

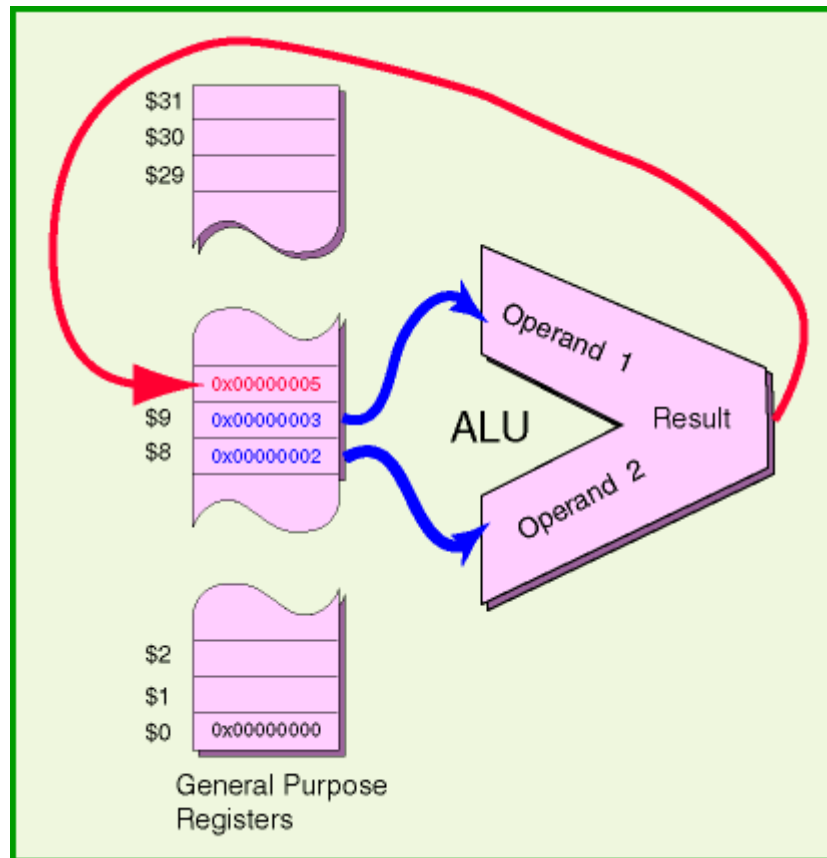
$\$d = \$s - \$t$

- Example: $f = (g + h) - (i + j)$
 - $f, g, h, i, j \longrightarrow \$s0, \$s1, \$s2, \$s3, \$s4$
 - `add $t0, $s1, $s2` `# g+h`
 - `add $t1, $s3, $s4` `# i+j`
 - `sub $s0, $t0, $t1` `# (g+h)-(i+j)`

Figure 2.2 Adding values stored in memory

MIPS(RISC)

- add \$10,\$8,\$9



Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Figure 2.3 Dividing values stored in memory

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

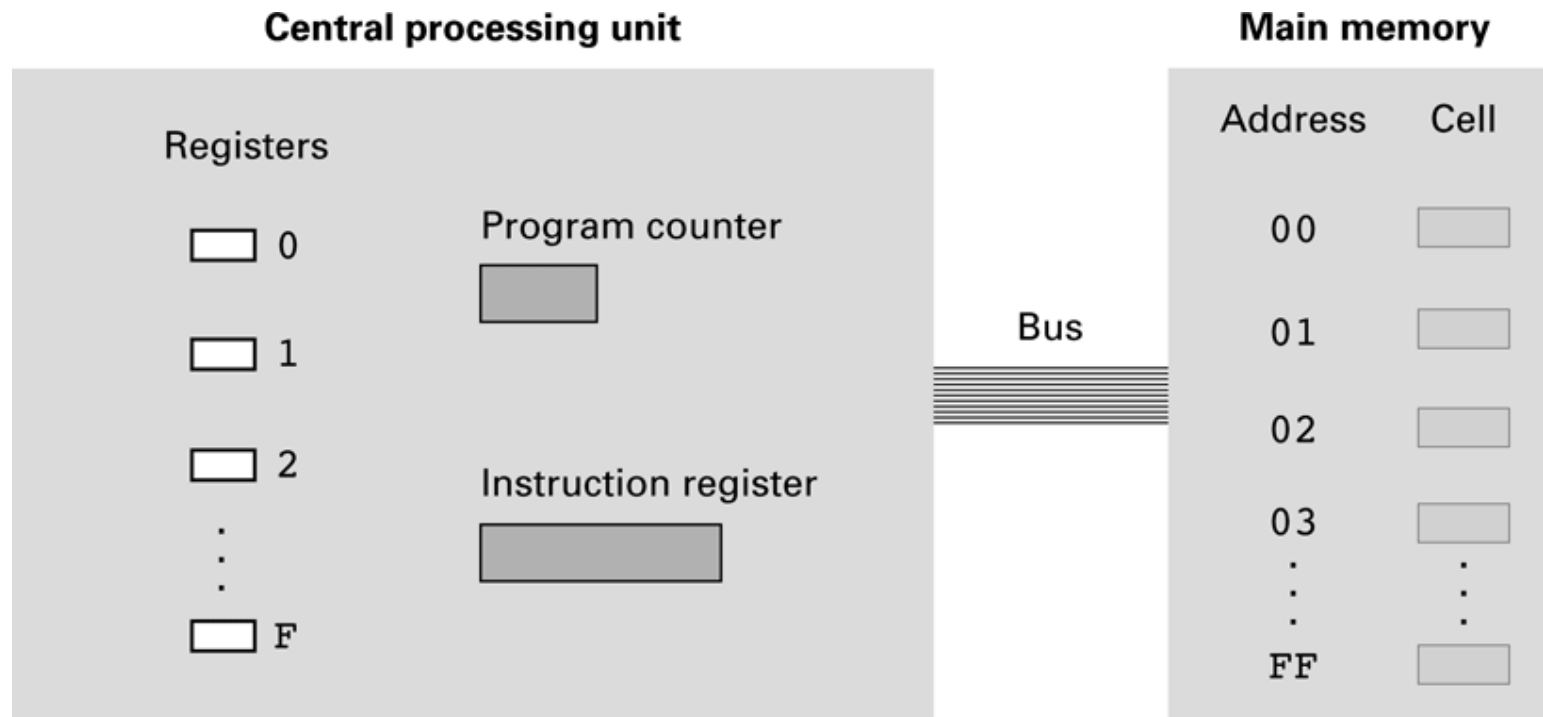
Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

Figure 2.4 The architecture of the machine described in Appendix C



Parts of a Machine Instruction

- **Op-code**（操作码）：Specifies which operation to execute
- **Operand**（操作数）：Gives more detailed information about the operation
 - Interpretation of operand varies depending on op-code

Figure 2.5 and Figure 2.6

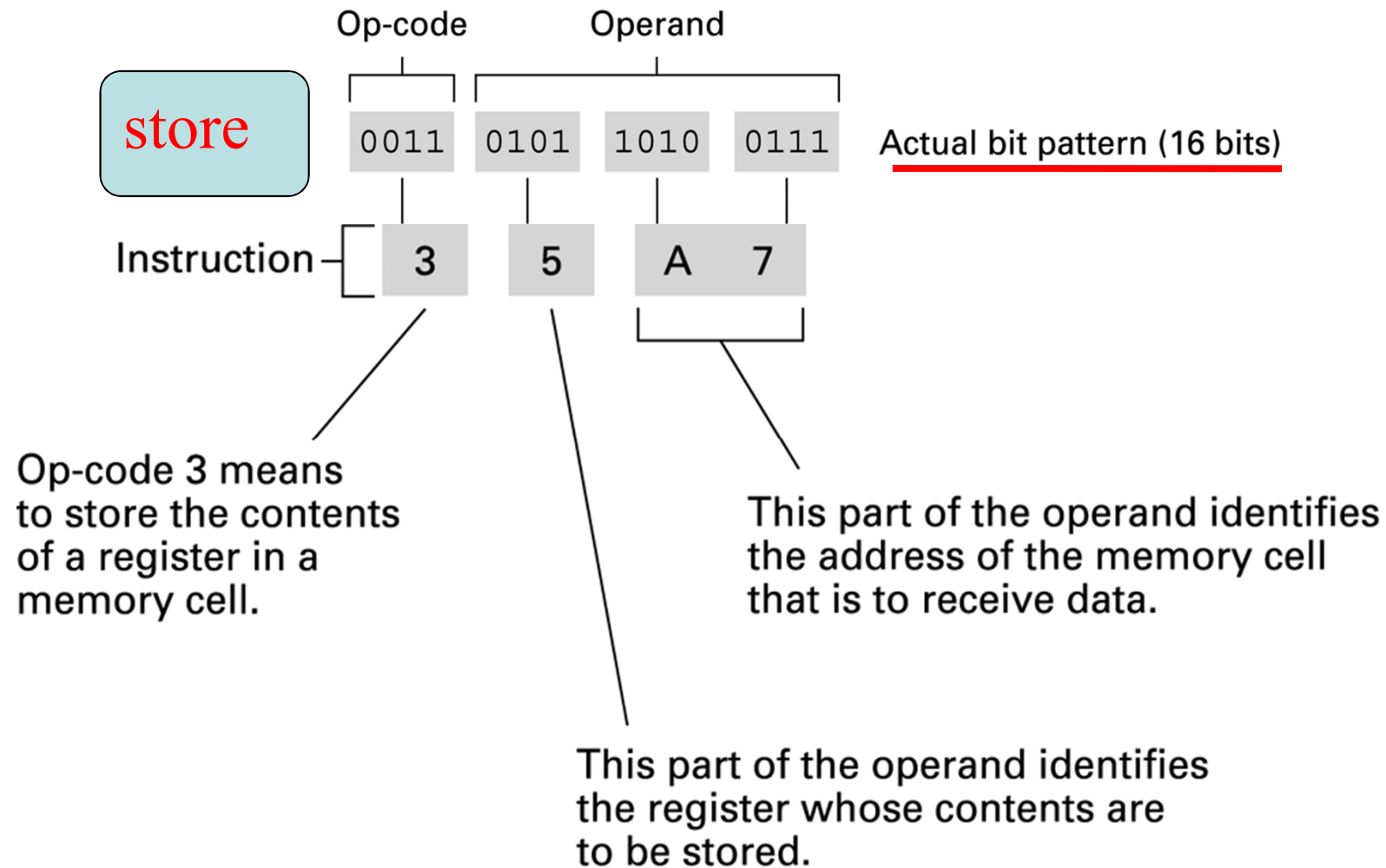
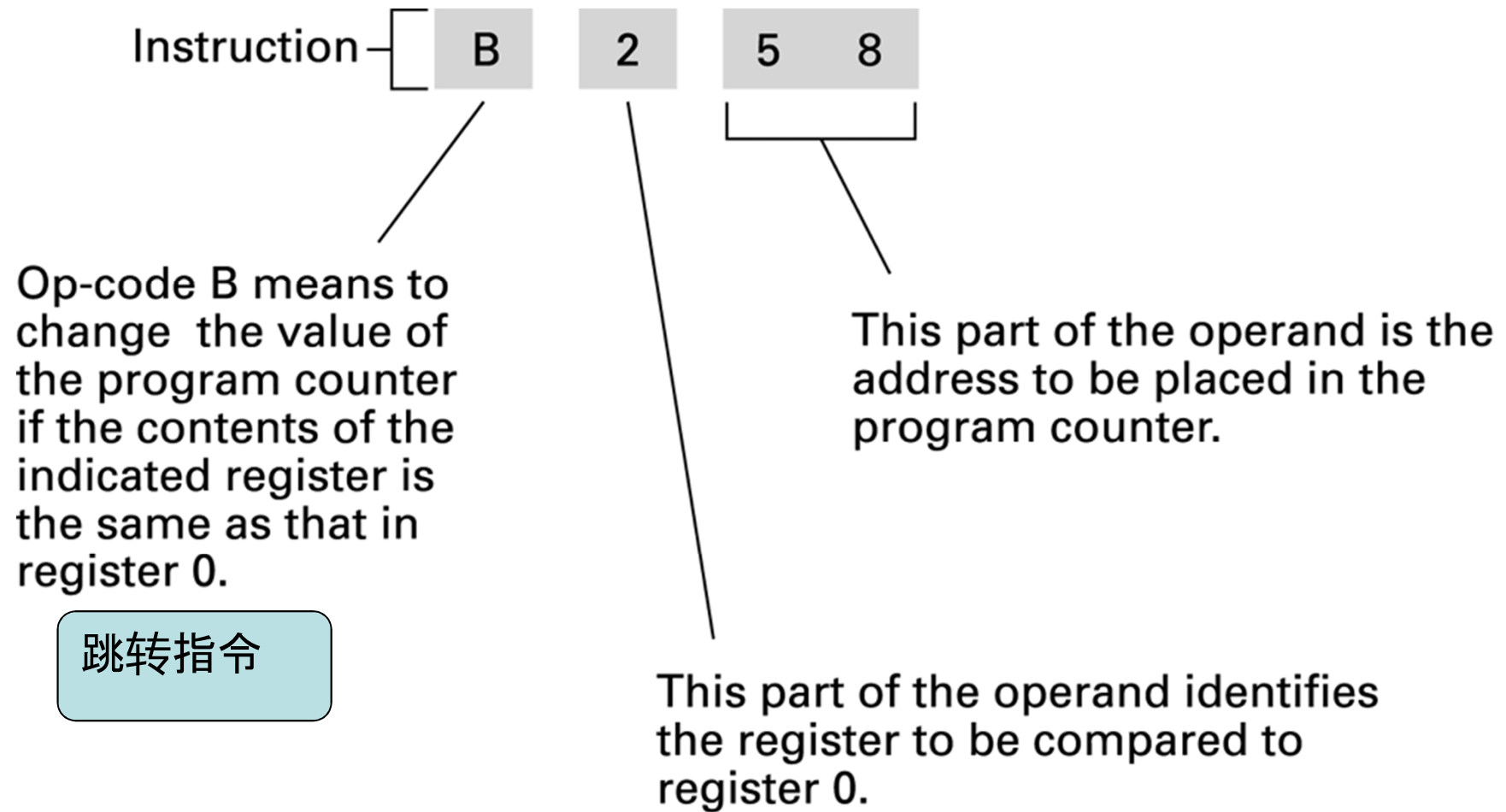


Figure 2.9 Decoding the instruction B258



Program Execution

- Controlled by two special-purpose registers
 - Program counter(程序计数器): address of next instruction
 - Instruction register (指令寄存器) : current instruction
- Machine Cycle
 - Fetch (取指)
 - Decode (译码)
 - Execute (执行)

Figure 2.8 The machine cycle

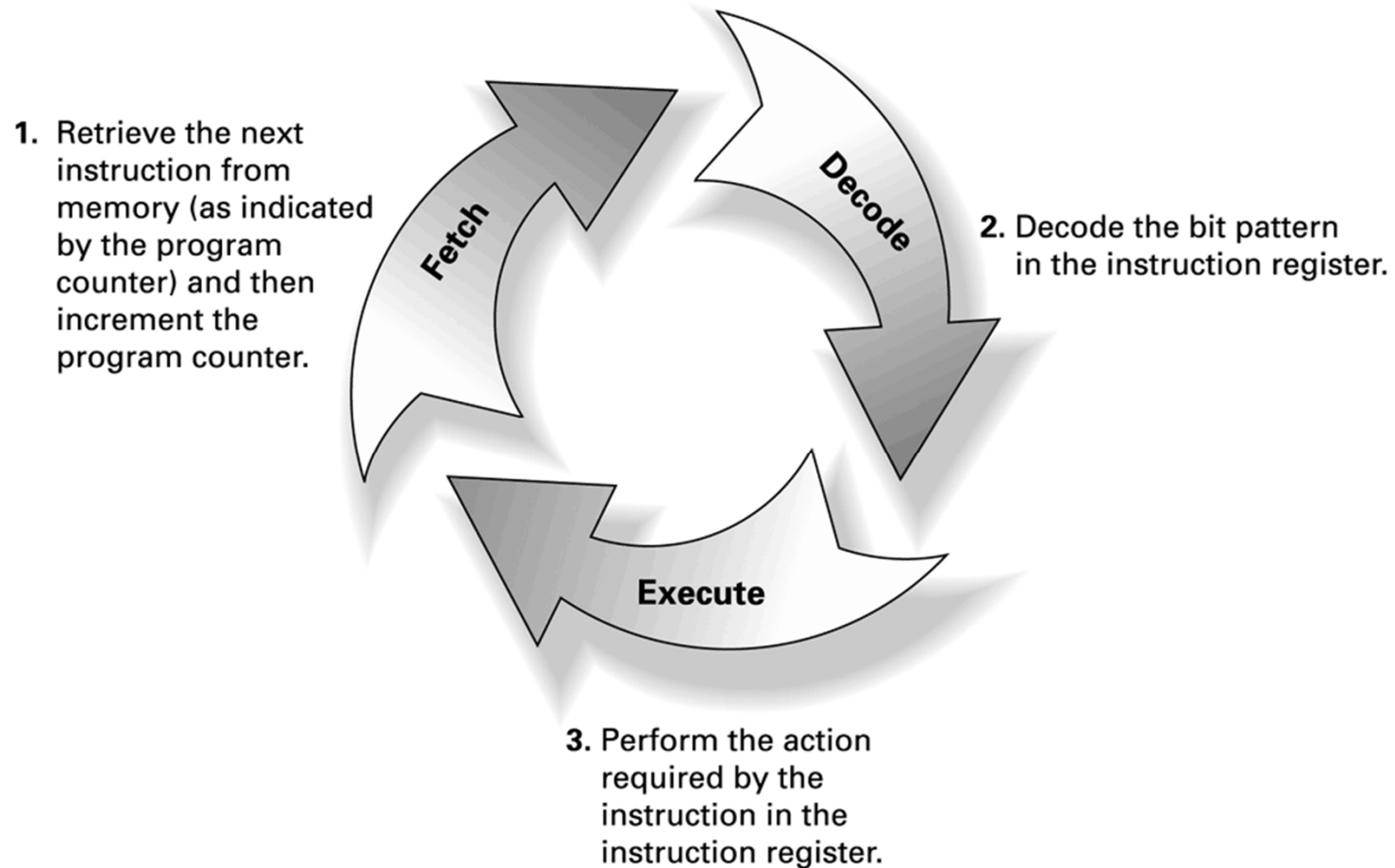


Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution

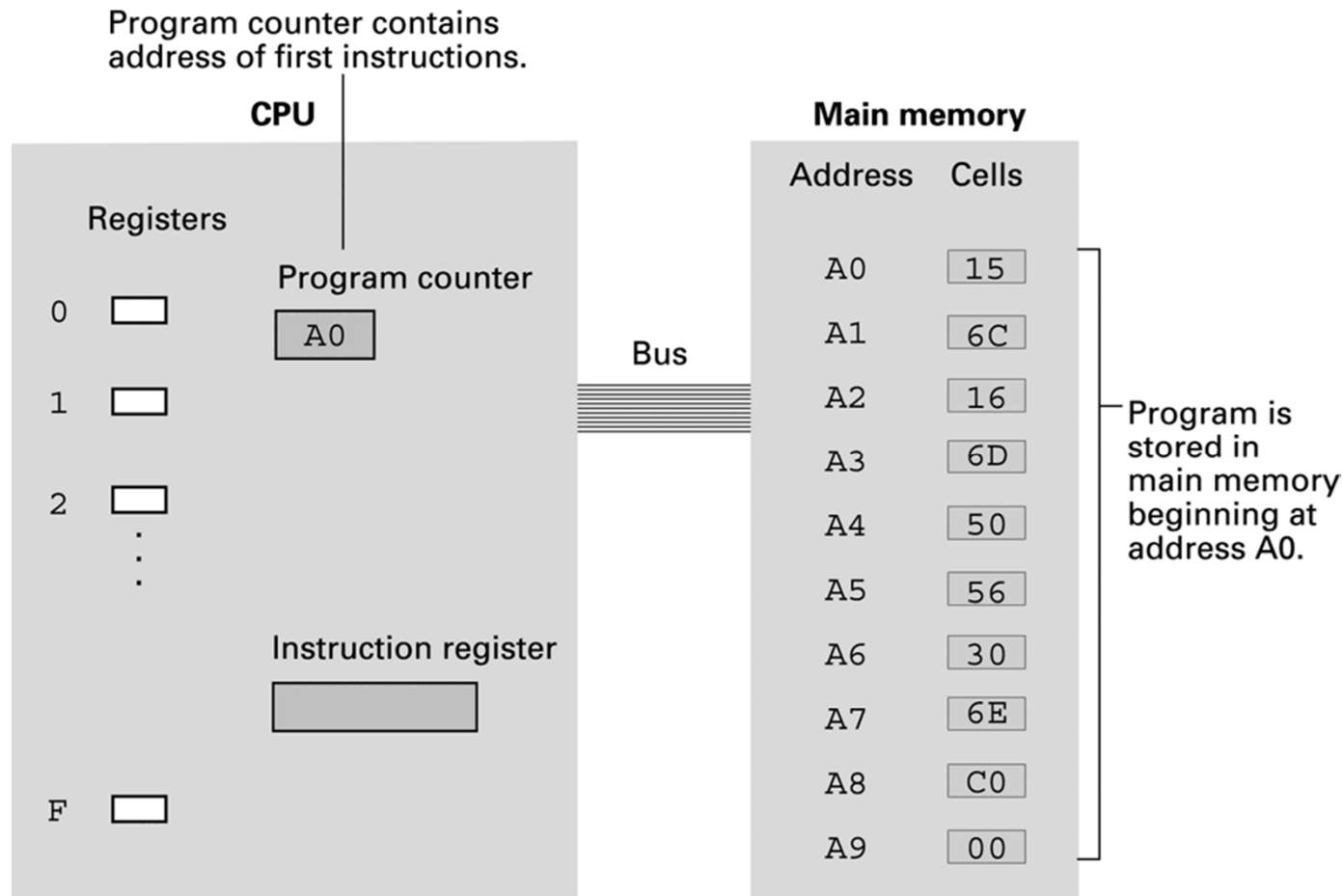
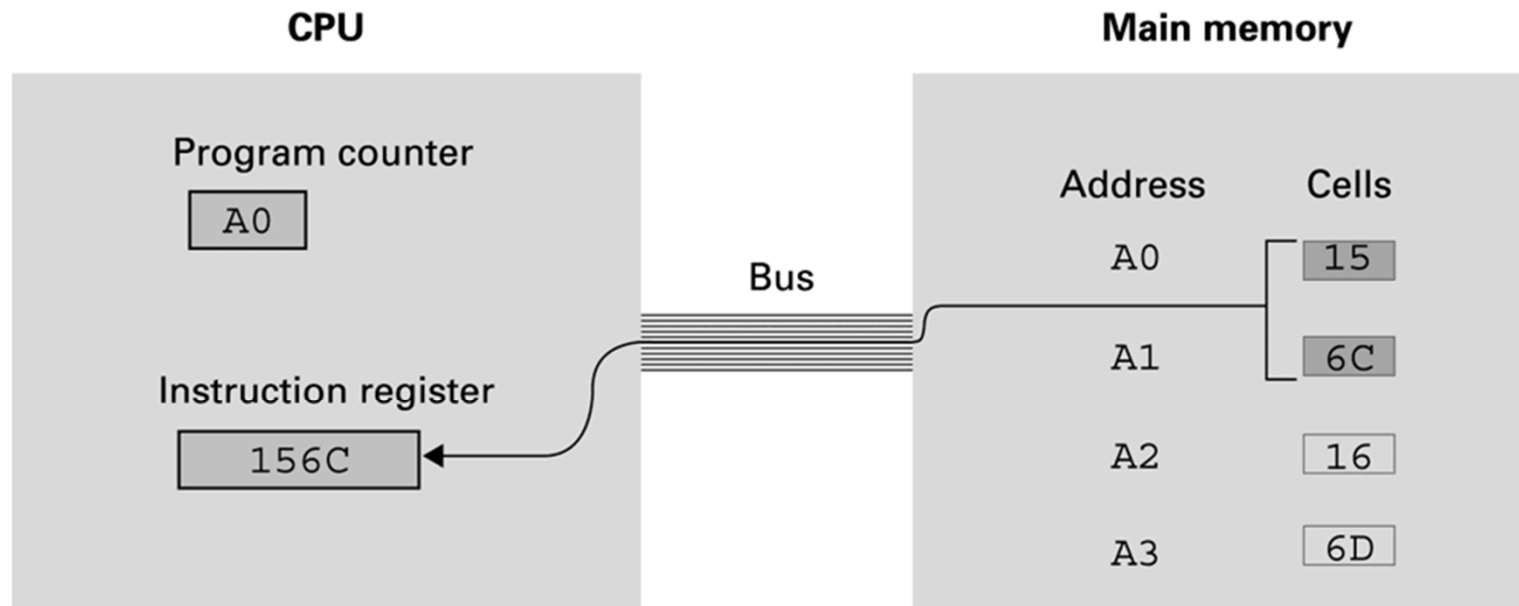
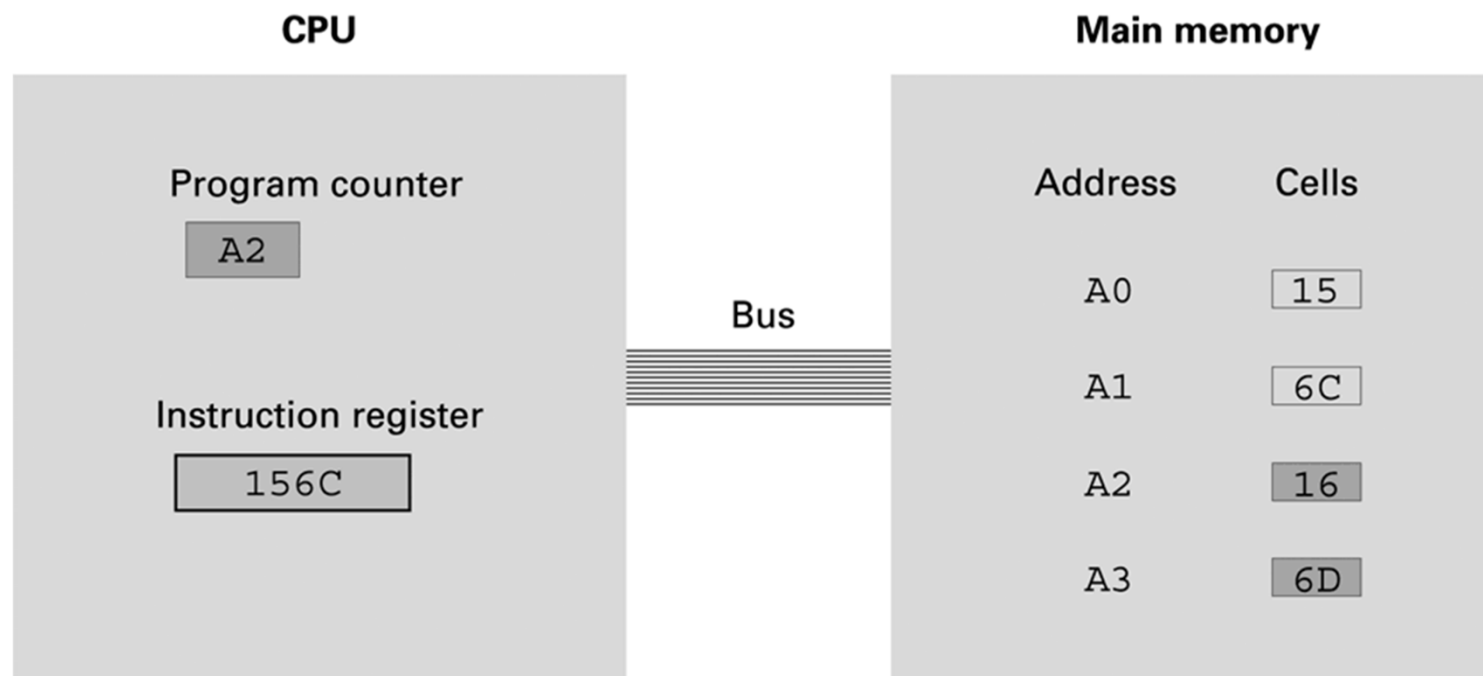


Figure 2.11 Performing the fetch step of the machine cycle



- a. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.

Figure 2.11 Performing the fetch step of the machine cycle (cont'd)



b. Then the program counter is incremented so that it points to the next instruction.

Figure 2.7 An encoded version of the instructions in Figure 2.2

operation	register	address
-----------	----------	---------

Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Encoded instructions

Translation

156C

Load register 5 with the bit pattern found in the memory cell at address 6C.

166D

Load register 6 with the bit pattern found in the memory cell at address 6D.

5056

Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.

306E

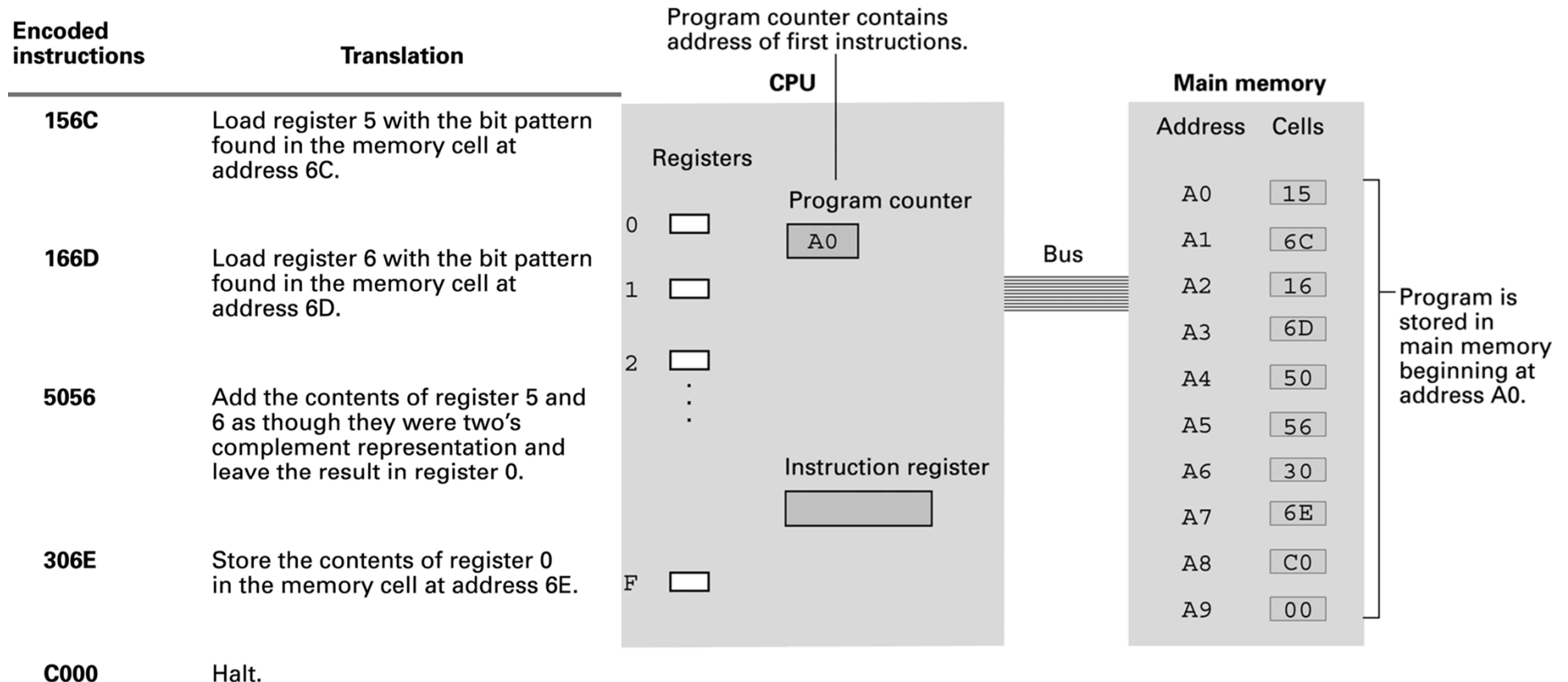
Store the contents of register 0 in the memory cell at address 6E.

C000

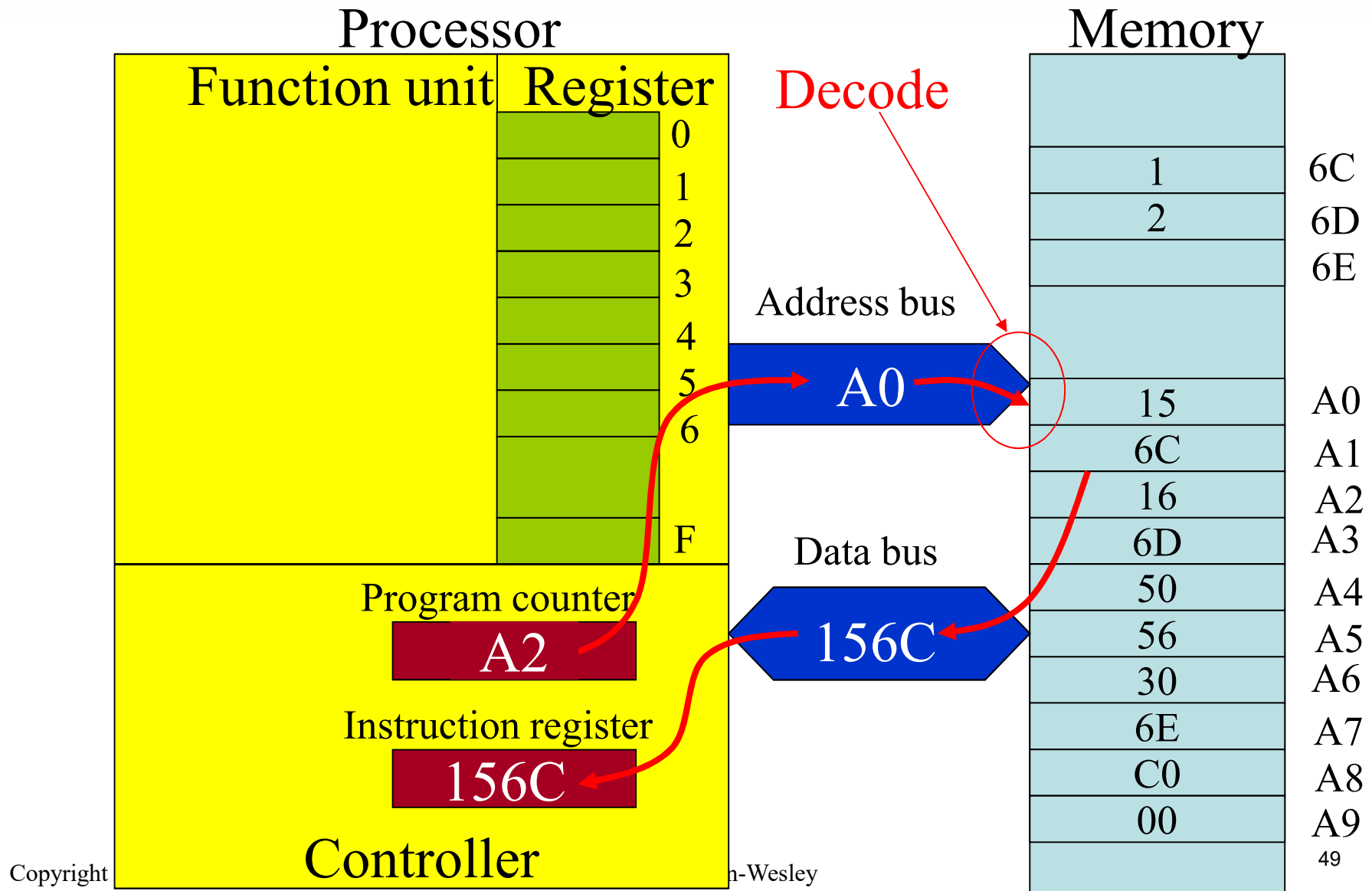
Halt.

Operation: 1-load 5-add 3-store

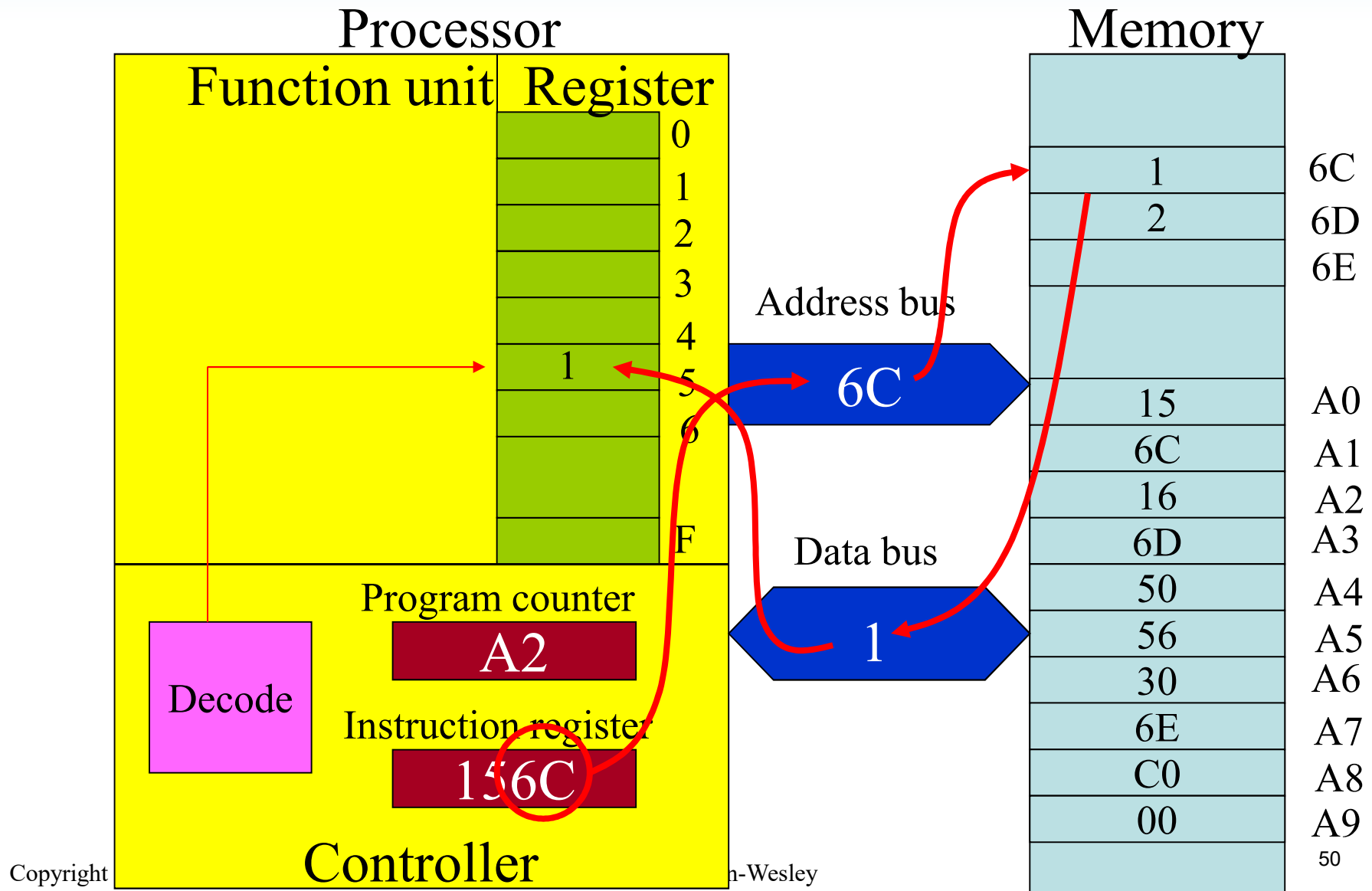
Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution



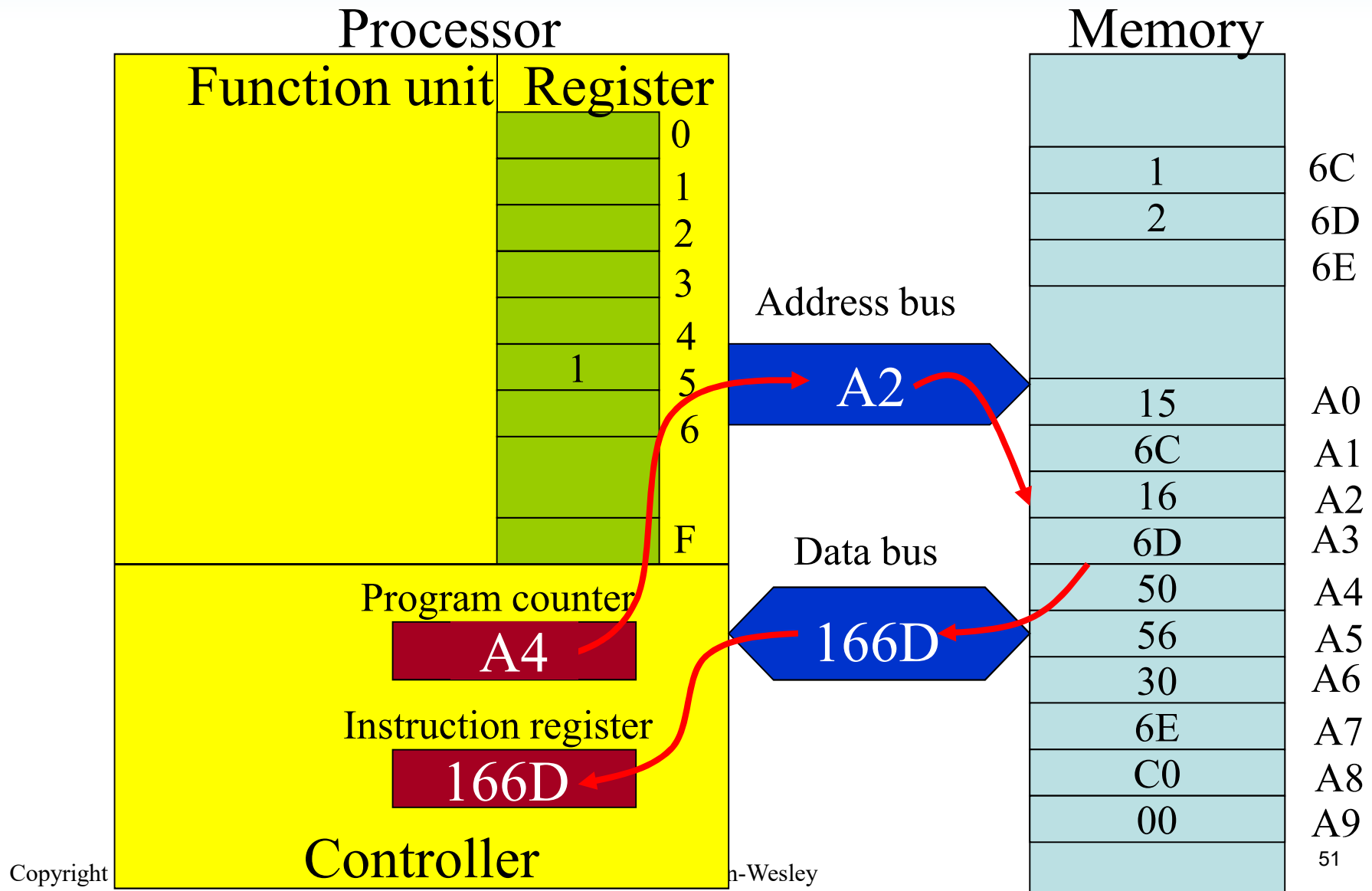
Fetch Instruction 1



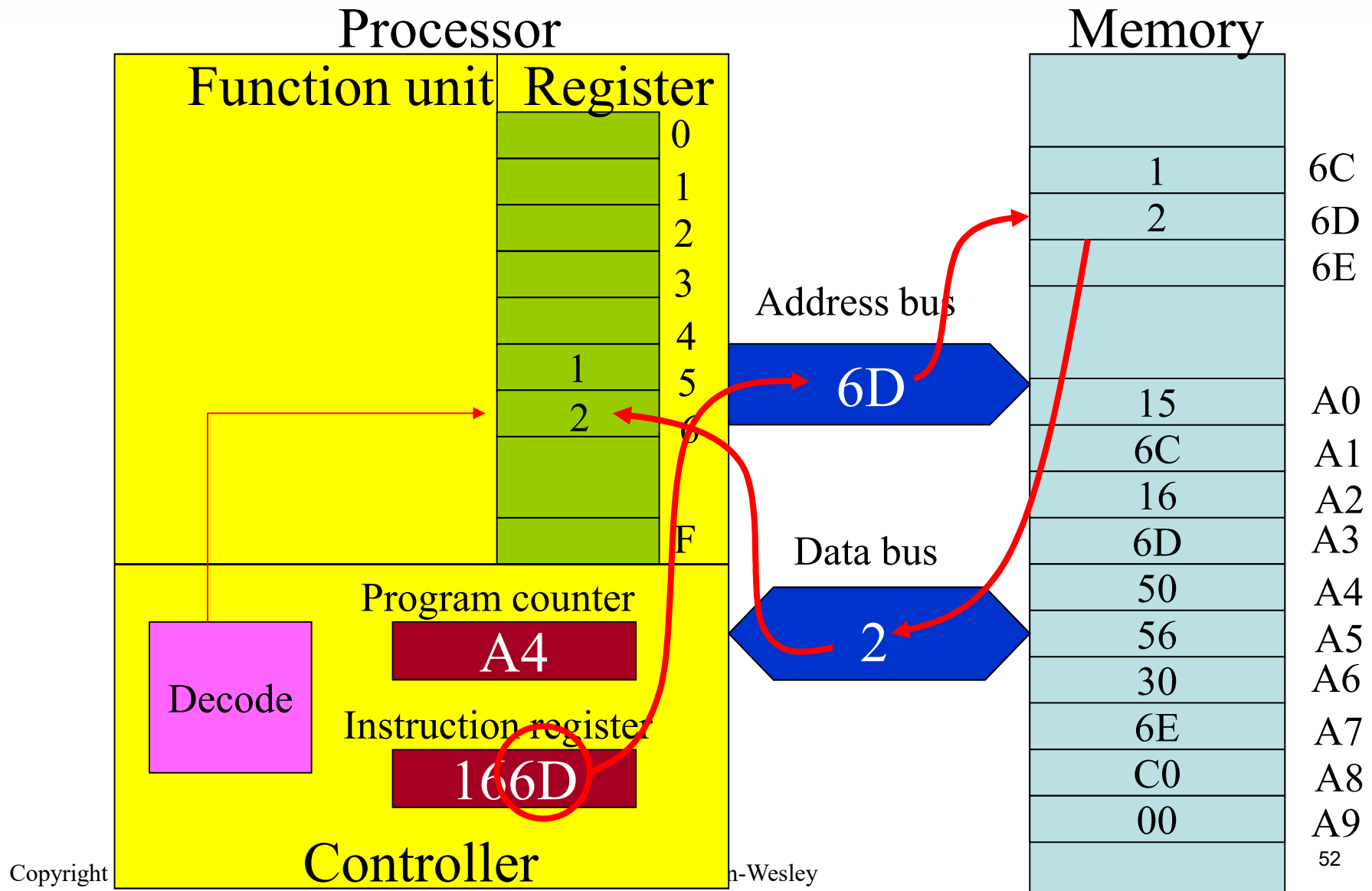
Execute Instruction 1



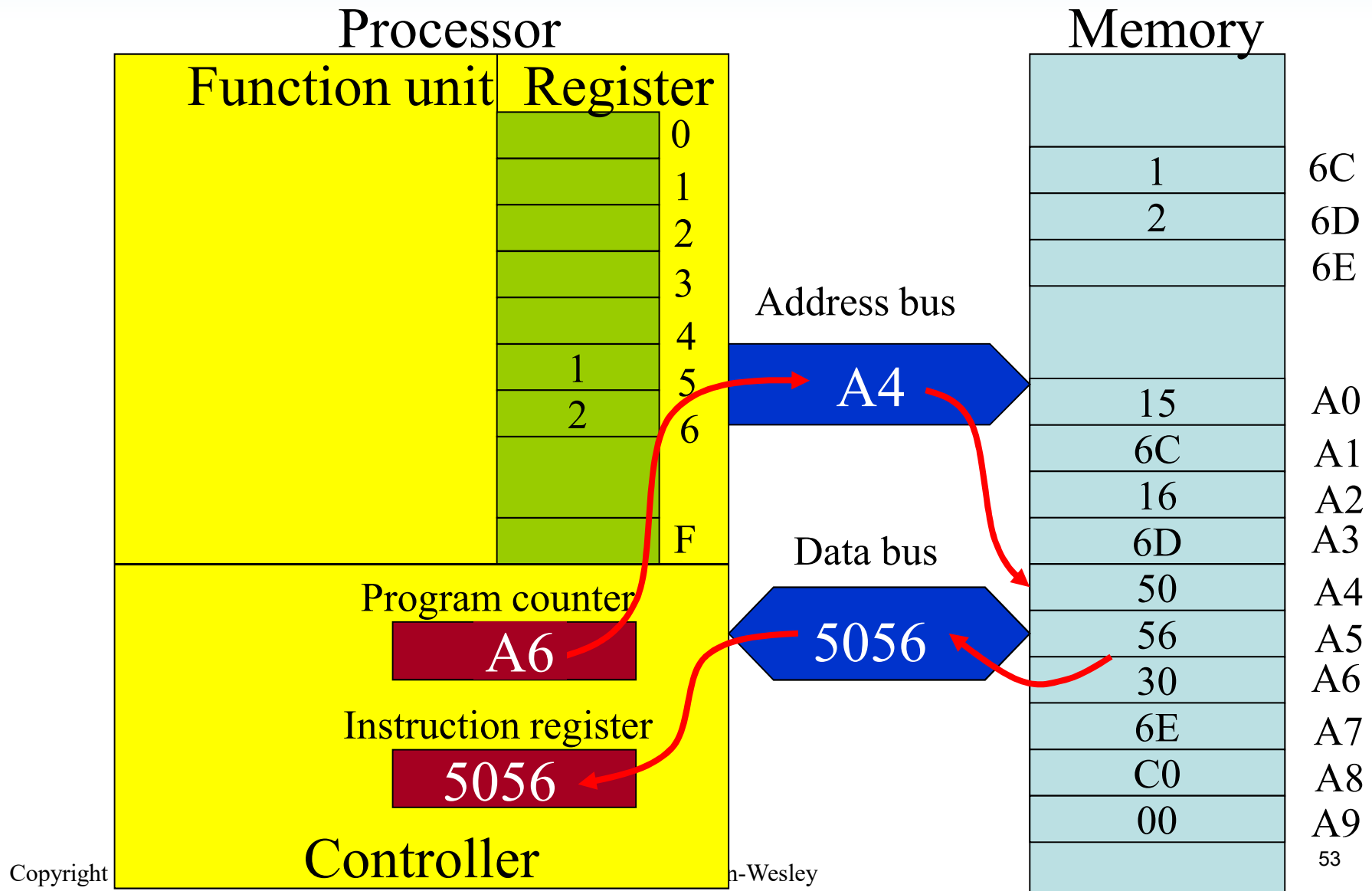
Fetch Instruction 2



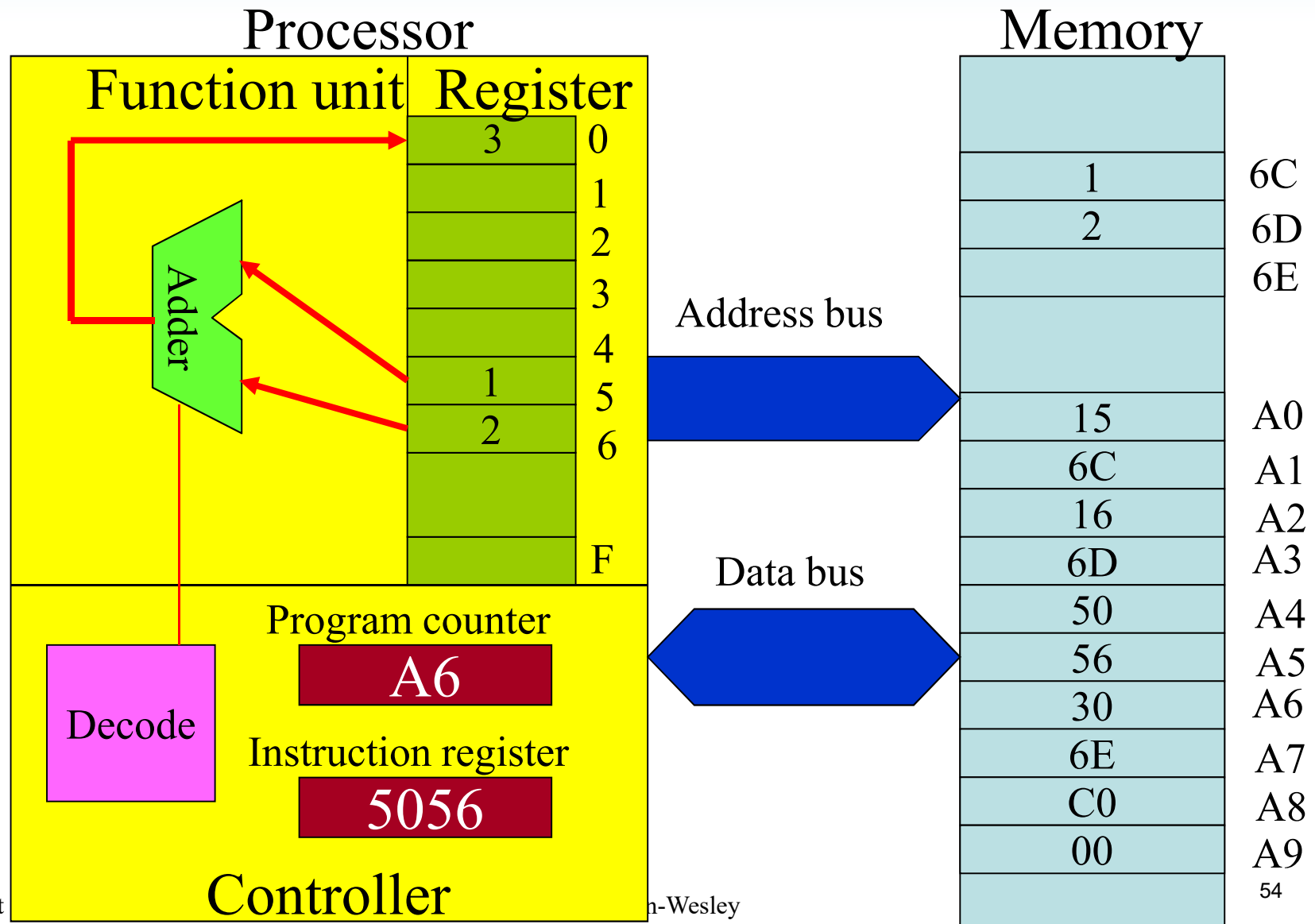
Execute Instruction 2



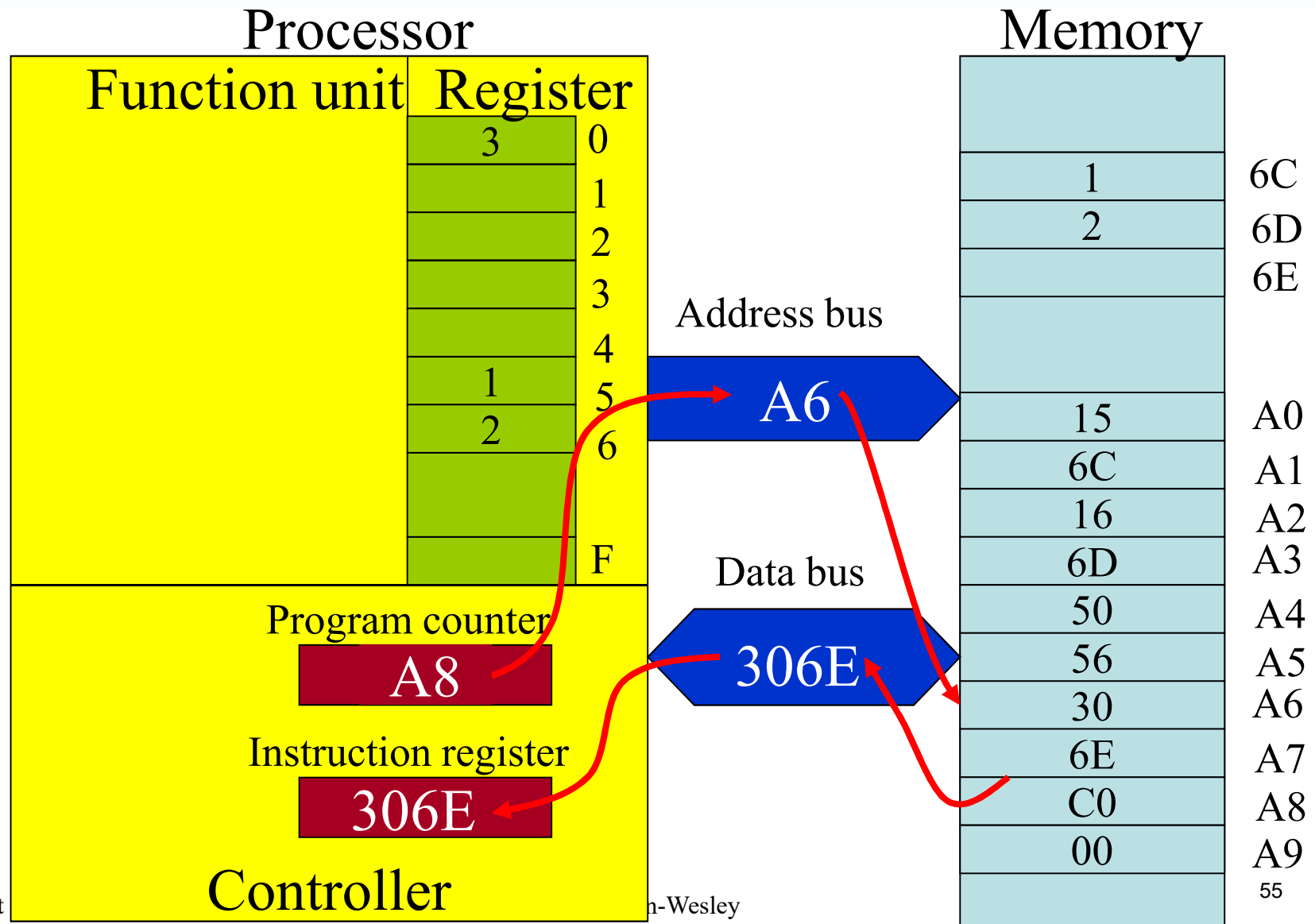
Fetch Instruction 3



Execute Instruction 3



Fetch Instruction 4



Execute Instruction 4

