计算机与网络技术

第2讲　计算机工作流程、指令系统

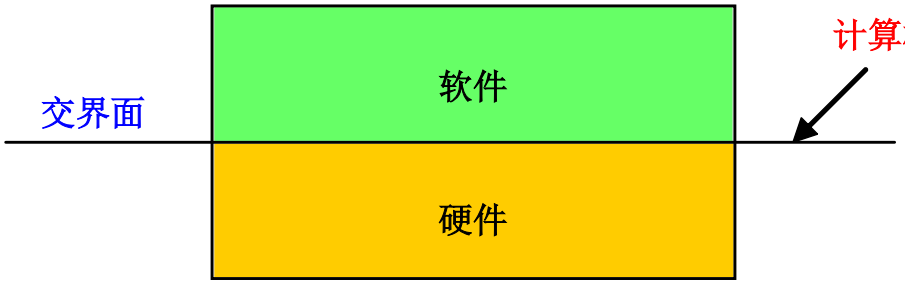
群聊: 2025春季计网课程

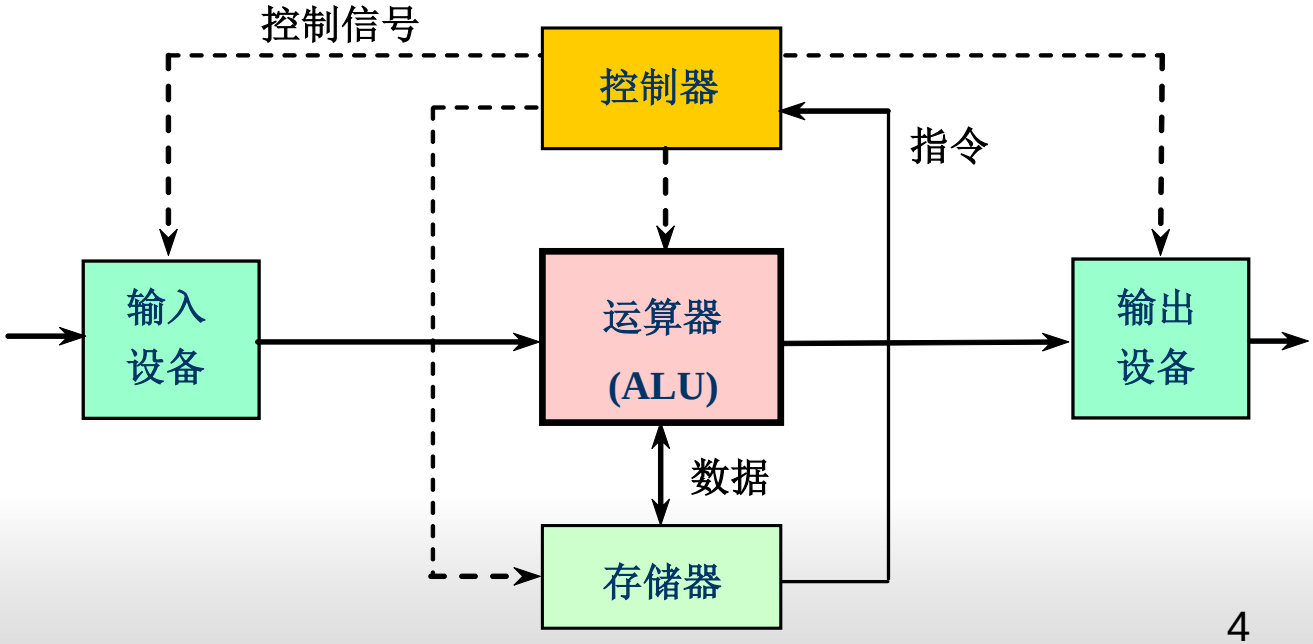
该二维码7天内(3月3日前)有效，重新进入将更新

提纲

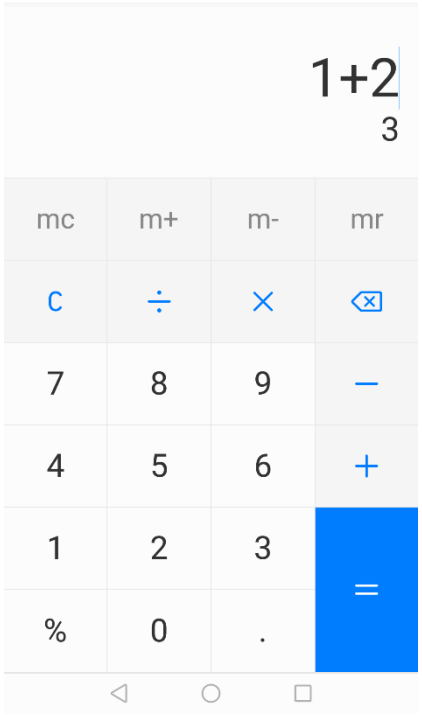
- 计算机工作流程
  - 程序创建执行过程
  - CPU微观工作流程
  - 计算机宏观工作流程
- 计算机指令系统（MIPS）
  - 指令系统
  - 寻址方式

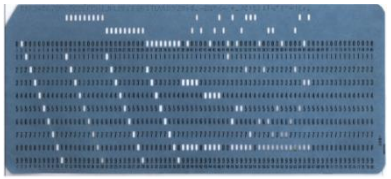
计算机工作流程





计算机工作流程





1937年，图灵提出以“存储程序、程序控制”为核心的“通用”计算机的概念：1）“可计算性”：可以执行任何一个描述好的程序（算法），在有限的时间内实现需要的功能；2）“存储程序”：问题的求解由程序或过程给出，程序和过程可以通过语言描述

计算机工作流程

George Gordon Byron
1788年1月22日 - 1824年4月19日

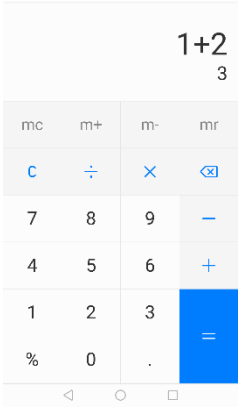
Augusta Ada King-Noel, Countess of Lovelace
1815年12月10日－1852年11月27日

在1842年与1843年期间，埃达花了9个月的时间翻译意大利数学家路易吉·米那比亚讲述查尔斯·巴贝奇分析机的论文。在译文后面，她增加了许多注记，详细说明用该机器计算伯努利数的方法，被认为是世界上第一个计算机程序；因此，埃达也被认为是世界上第一位程序员。

伦敦科学馆复制的巴贝奇分析机

计算机工作流程

例： 针对“a+b=?”的运算的过程



C等高级编程语言

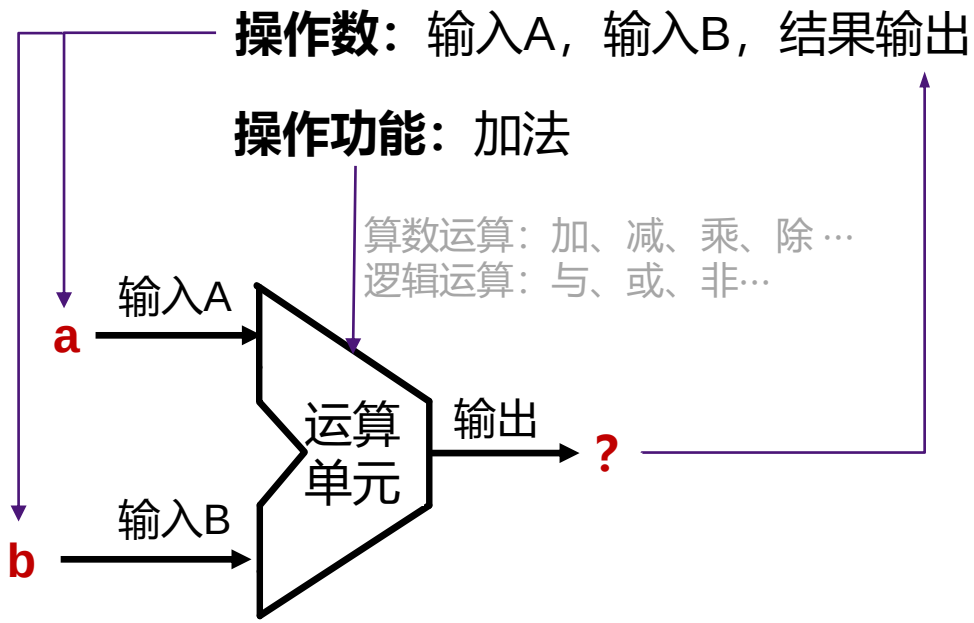
```
int a, b, c;
a = 1;
b = 2;
c = a + b;
```

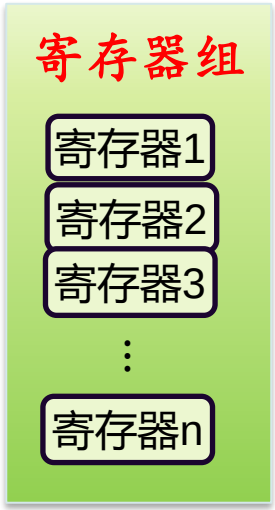
CPU执行了哪些操作（指令）？

计算机工作流程

例：“a+b=?”的运算

假设已经有一个设计好的运算单元



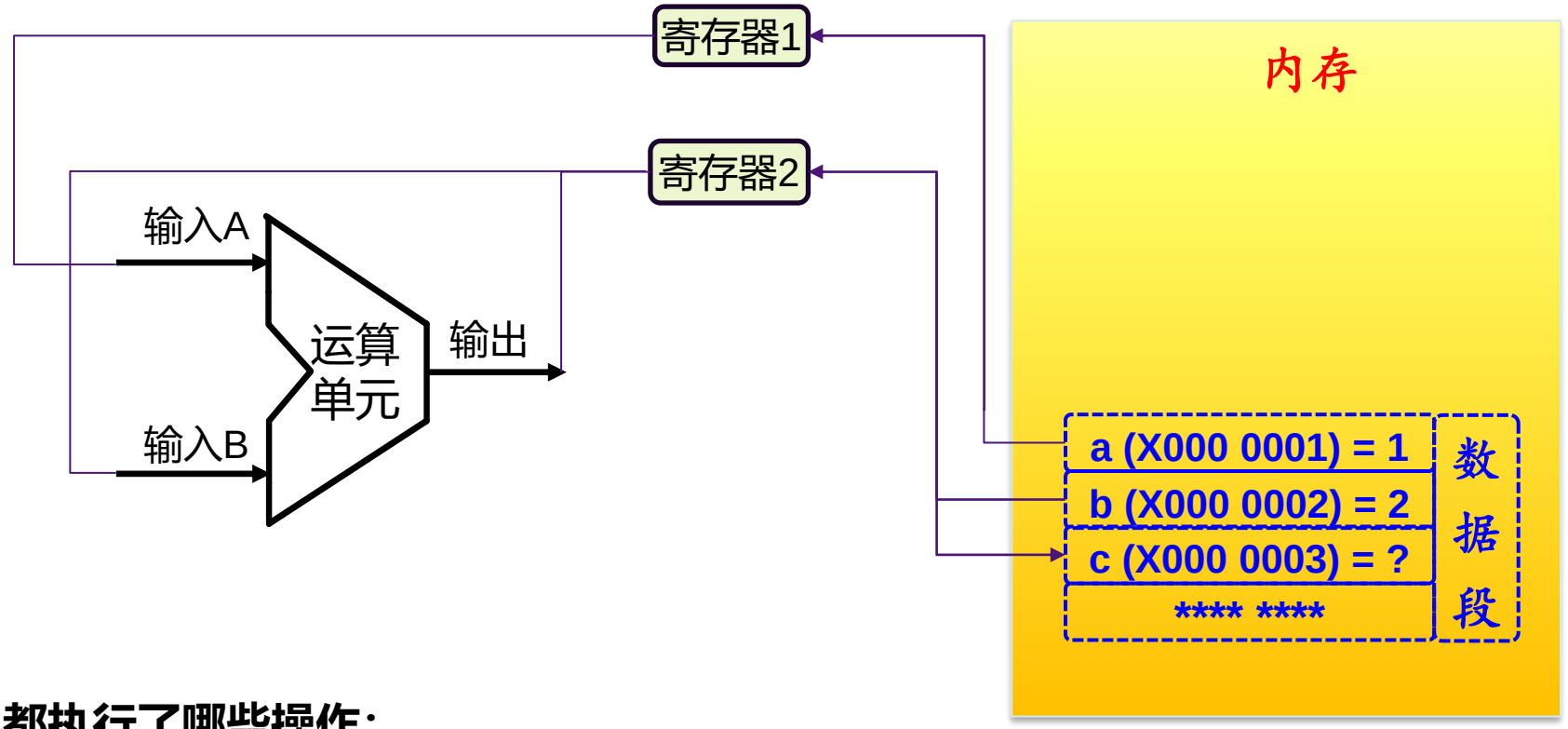


操作数放在哪里？（操作数寻址方式）

- 立即数：在操作指令中直接给定
- 寄存器：操作数放在寄存器中
- 内存：操作数放在存储单元中

计算机工作流程

例：计算 a+b=?　　输入的数放在内存中，计算结果也放在内存中



都执行了哪些操作：

1. 从内存中取数a到寄存器1
2. 从内存中取数b到寄存器2
3. 运算单元计算a+b（从寄存器1和寄存器2取操作数），计算结果放到寄存器2
4. 将寄存器2中的数放到内存c

计算机工作流程

C等高级编程语言

{计算机指令系统}

汇编语言

机器语言：二进制操作码
（Exe等可执行文件）

c = a + b;

编译
（Compiler）

Mov a, R1
Mov b, R2
Add R1, R2
Mov R2, c
指令（汇编）

汇编
（Assembler）

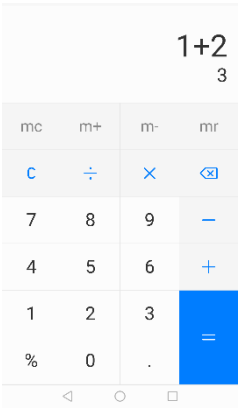
0000 0001
0000 0010
0010 0010
0000 1010
指令（二进制码）

存储

宏观
运行

微观
执行

执行
输出



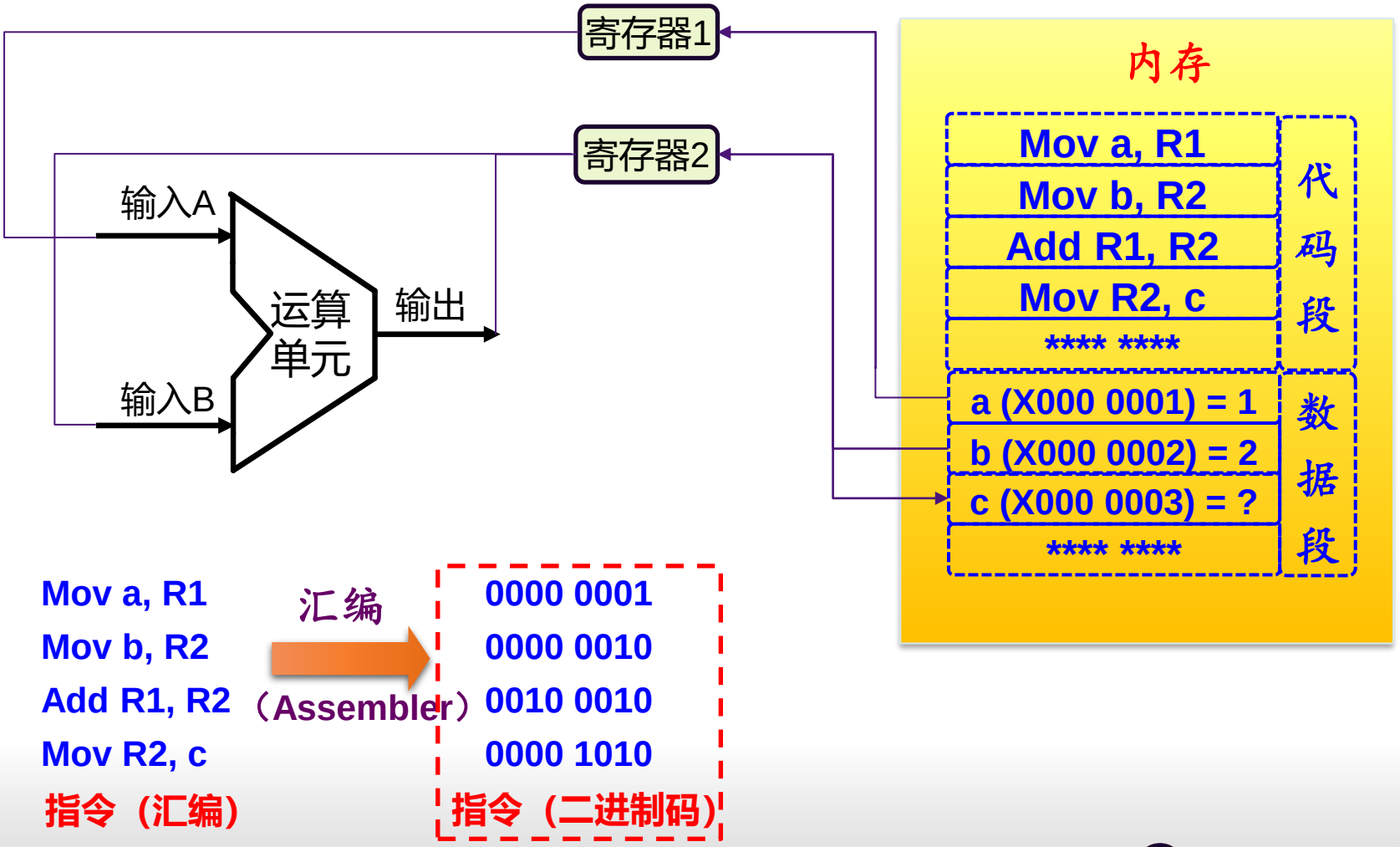
存储器

CPU
运算器
控制器

输出
设备

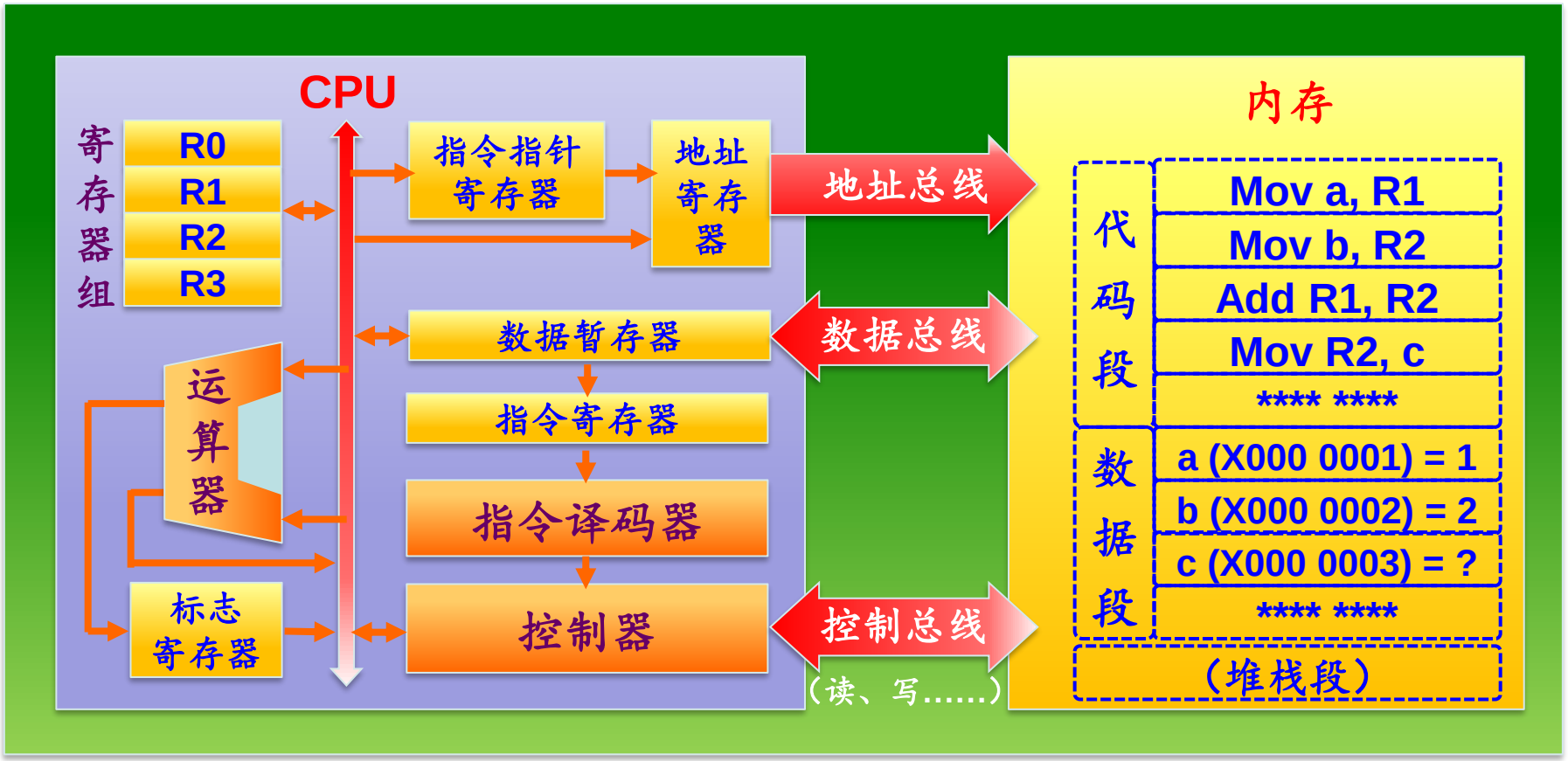
计算机工作流程

多条代码指令的连续执行



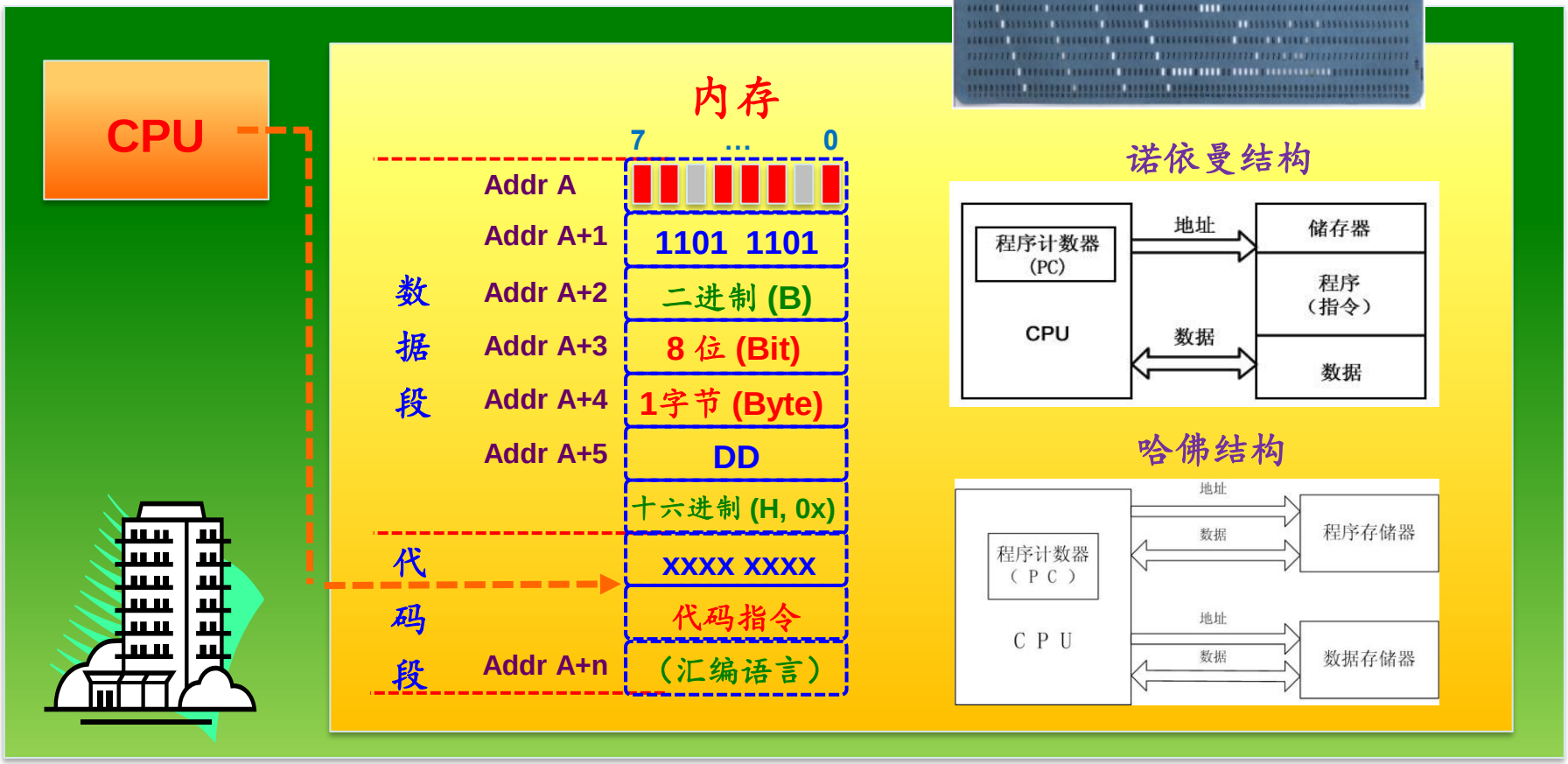
CPU微观工作流程

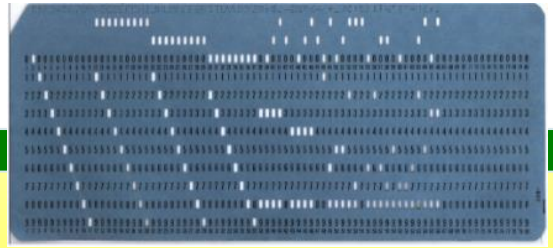
多条代码指令的连续执行



程序创建执行过程

程序在内存中的存储



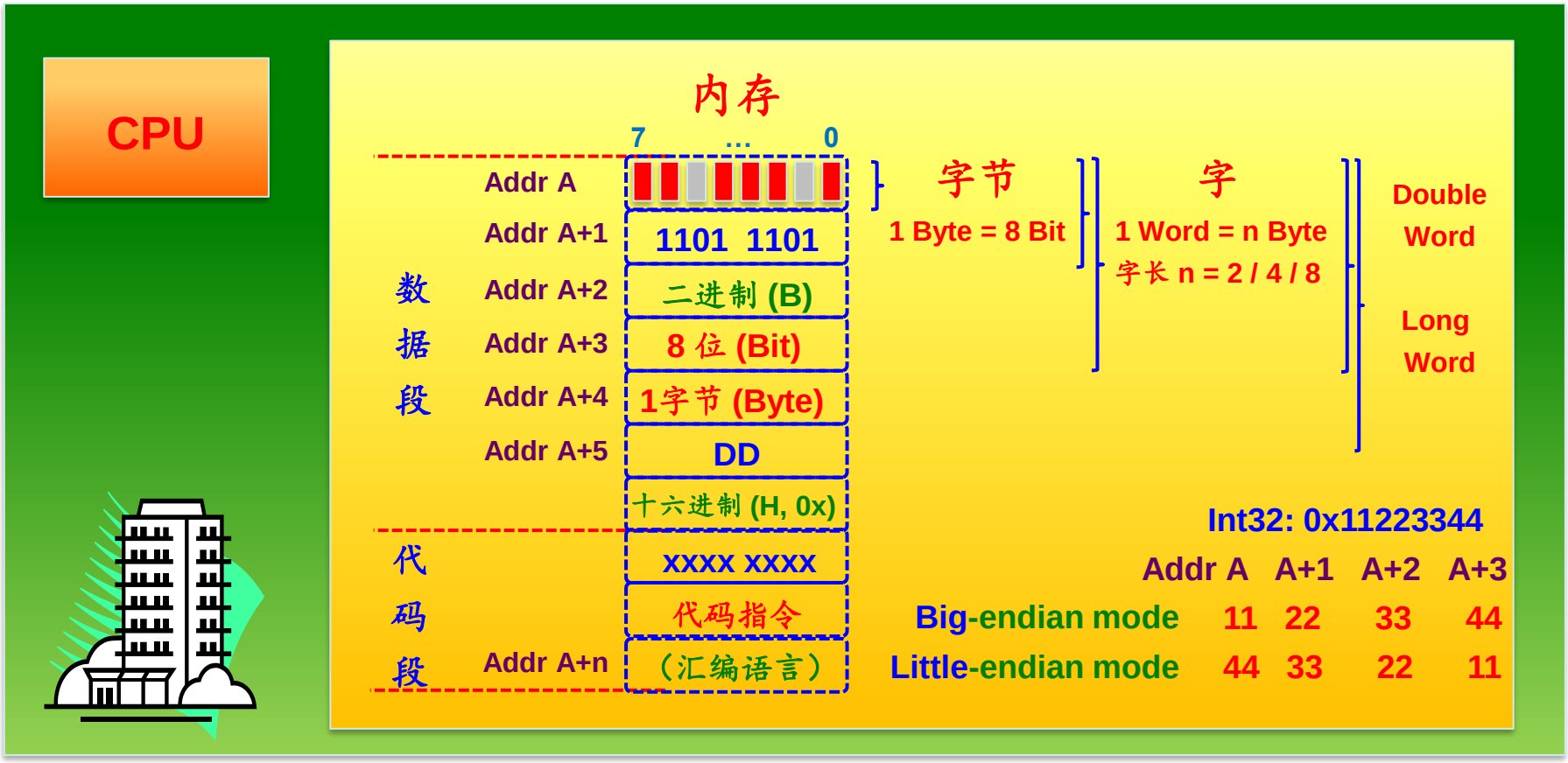


诺依曼结构

哈佛结构

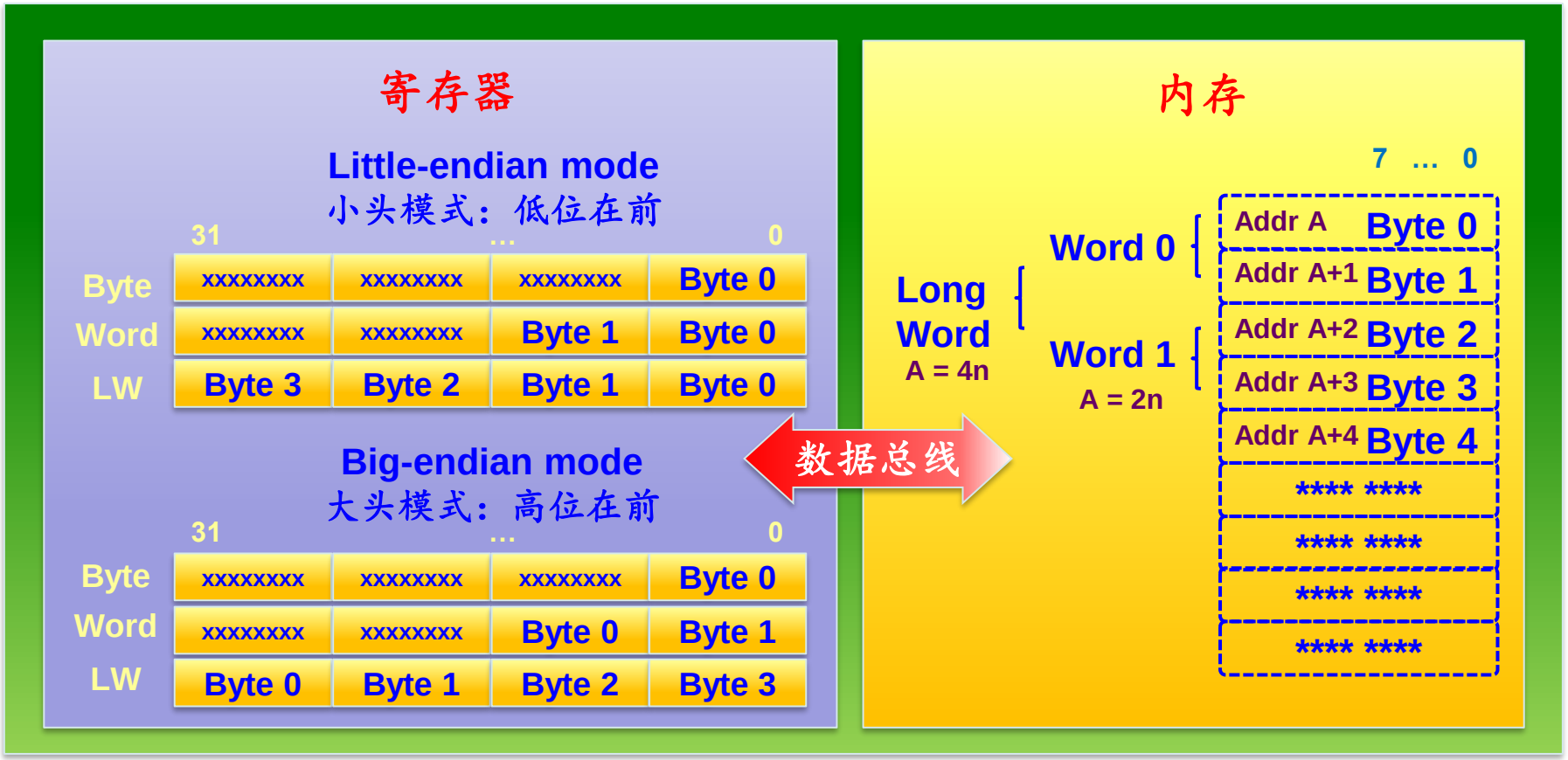
程序创建执行过程

数据在内存中的存储



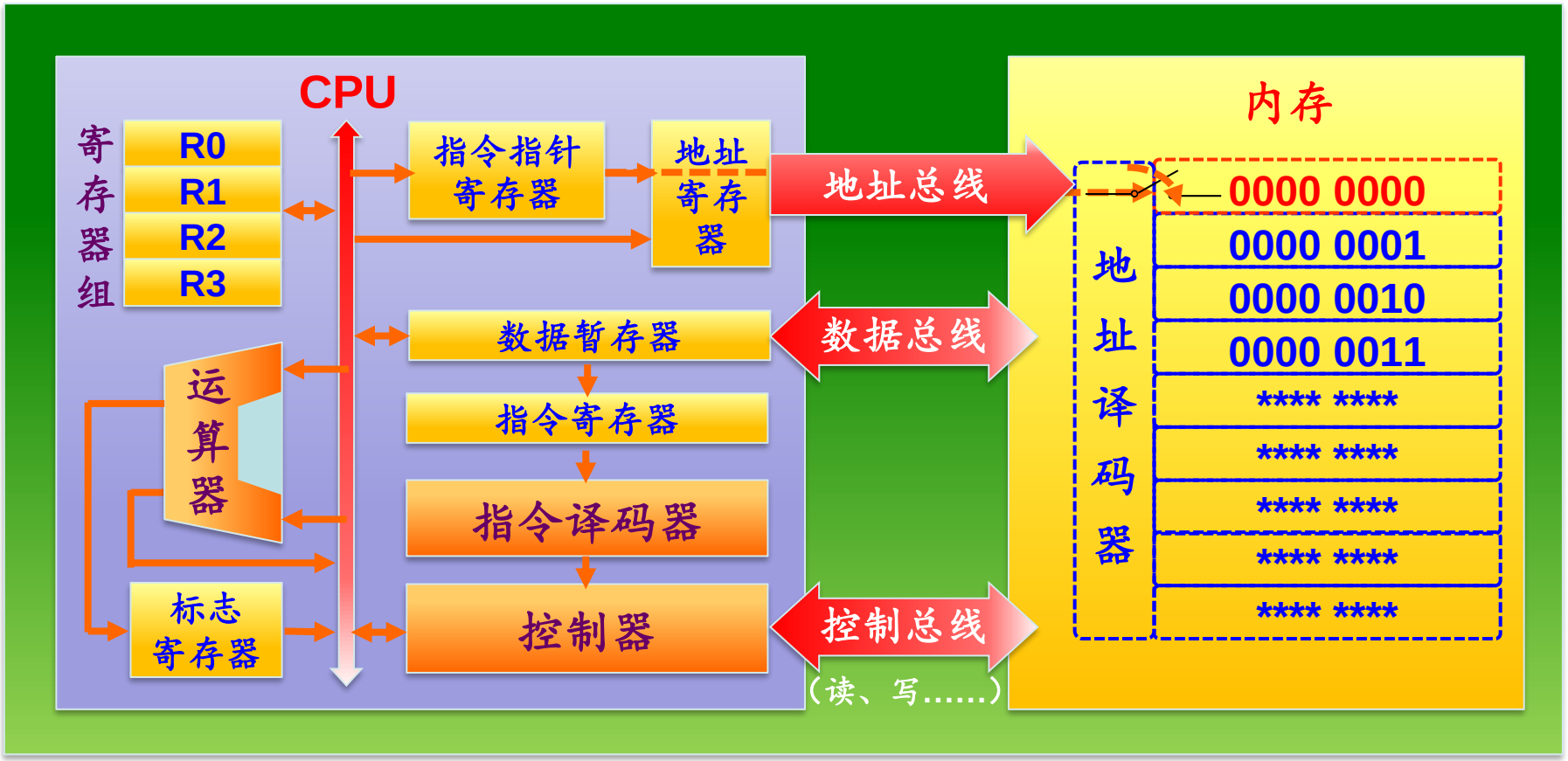
程序创建执行过程

数据在内存中的存储



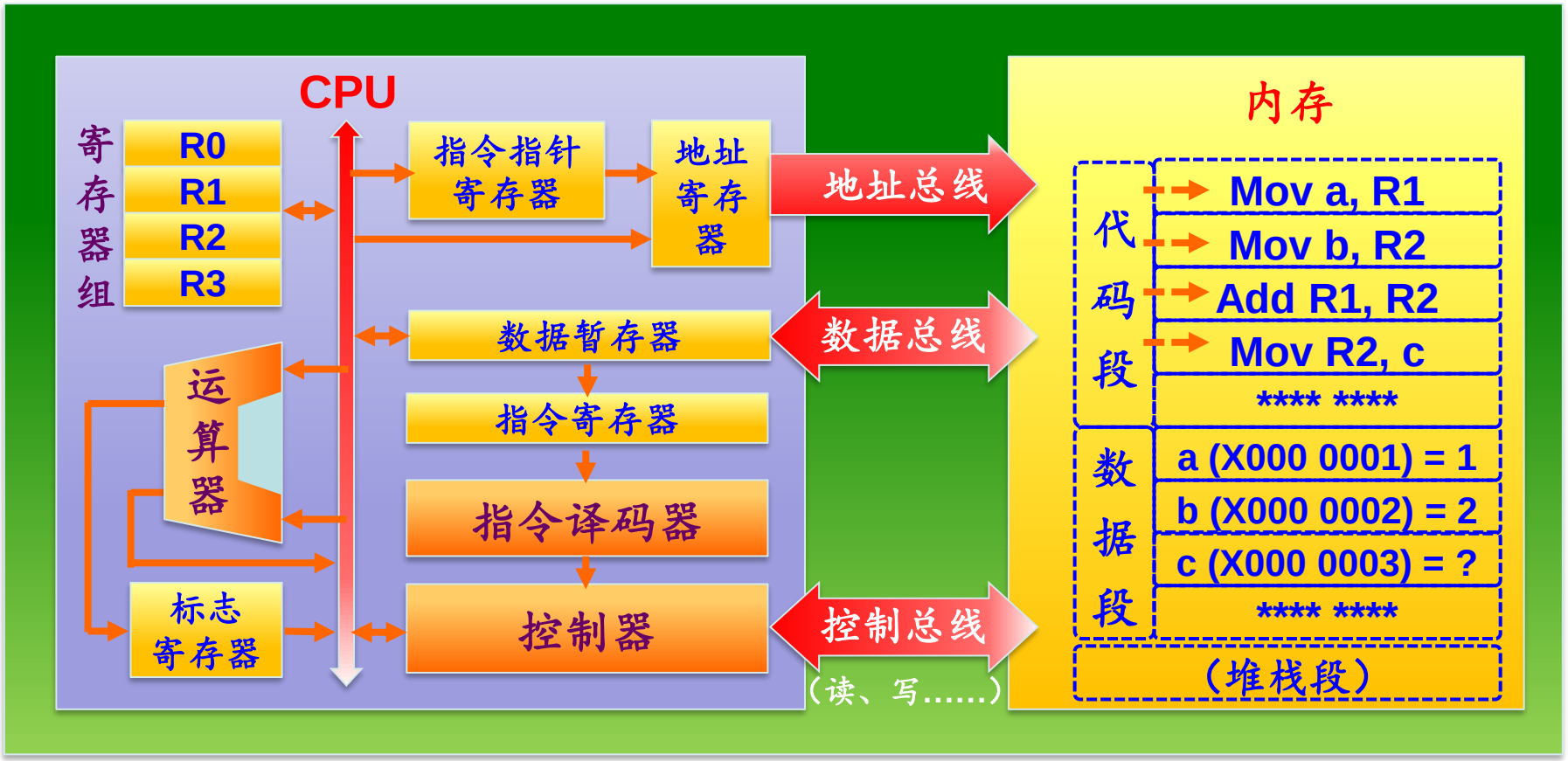
CPU微观工作流程

CPU与内存的总线连接



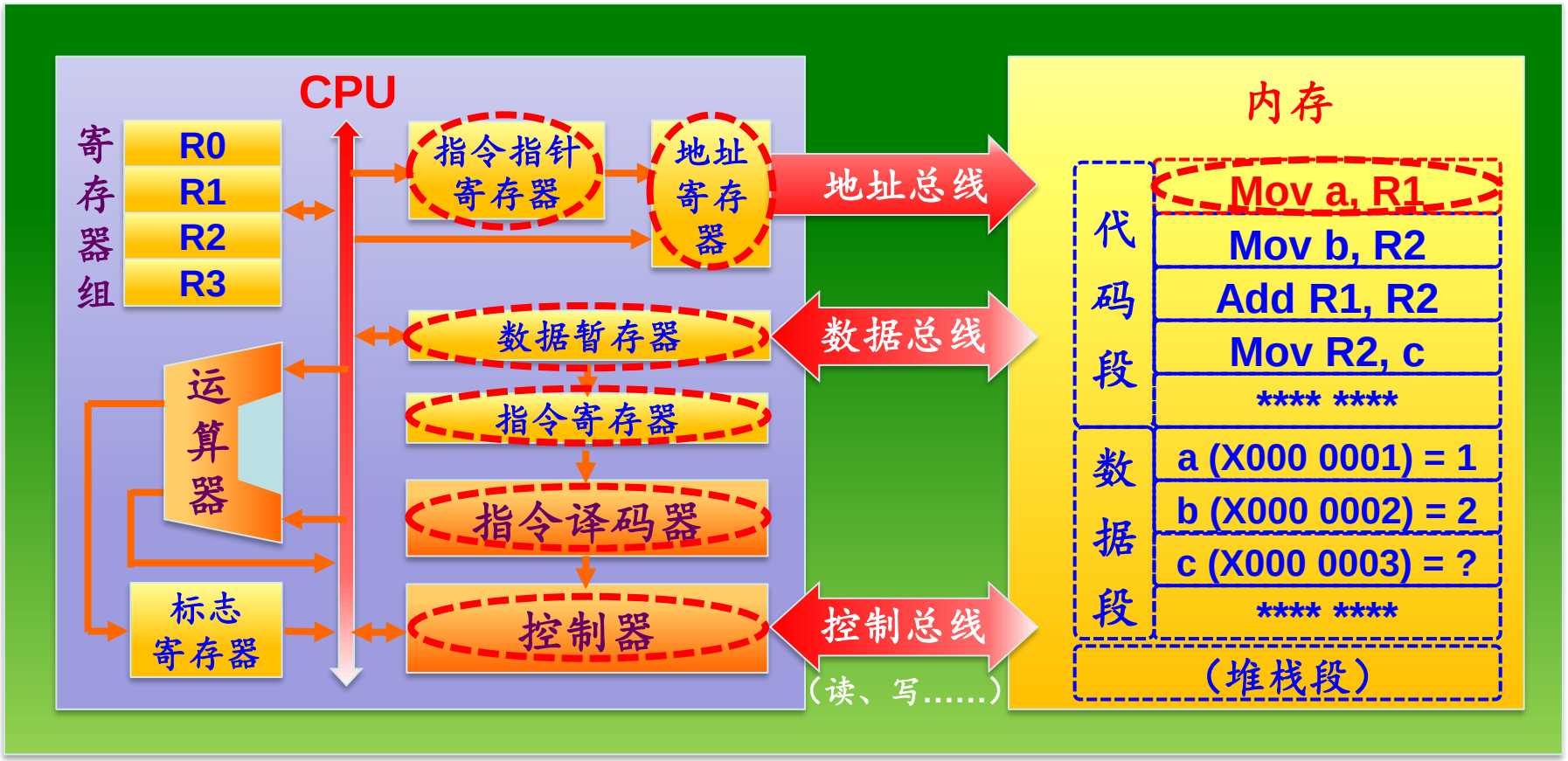
CPU微观工作流程

多条代码指令的连续执行



CPU微观工作流程

单条指令执行过程分解：取指令 → 译码 → 执行

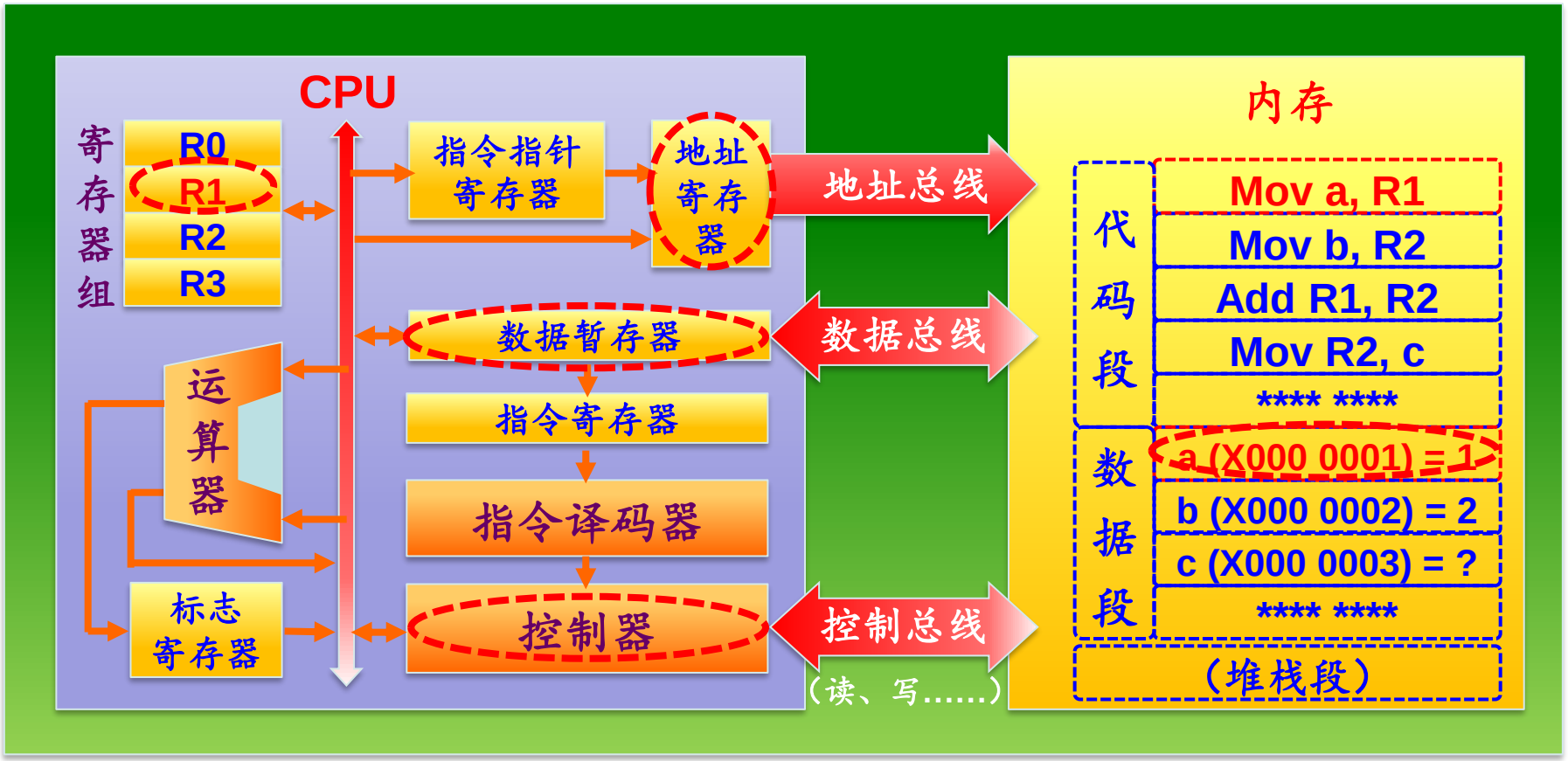


数据通路（Datapath）：指令执行过程中，数据所经过的路径，包括路径中的指令执行部件

指令控制部件（Control）：对执行部件发出控制信号，包括对指令进行译码，生成指令对应的控制信号，控制数据通路的动作

CPU微观工作流程

单条指令执行过程分解：取指令 → 译码 → 执行

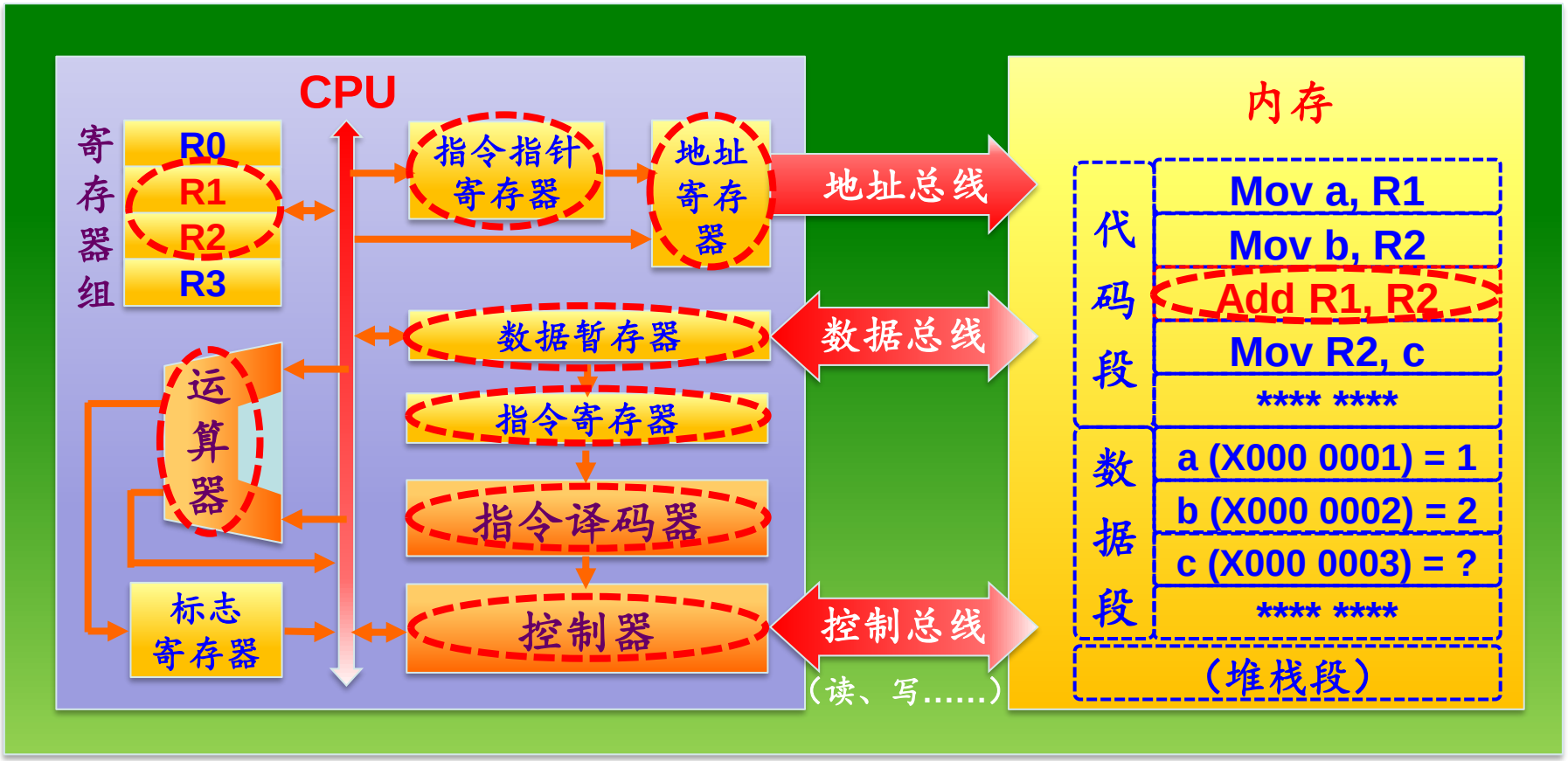


数据通路（Datapath）：指令执行过程中，数据所经过的路径，包括路径中的指令执行部件

指令控制部件（Control）：对执行部件发出控制信号，包括对指令进行译码，生成指令对应的控制信号，控制数据通路的动作

CPU微观工作流程

单条指令执行过程分解：取指令 → 译码 → 执行

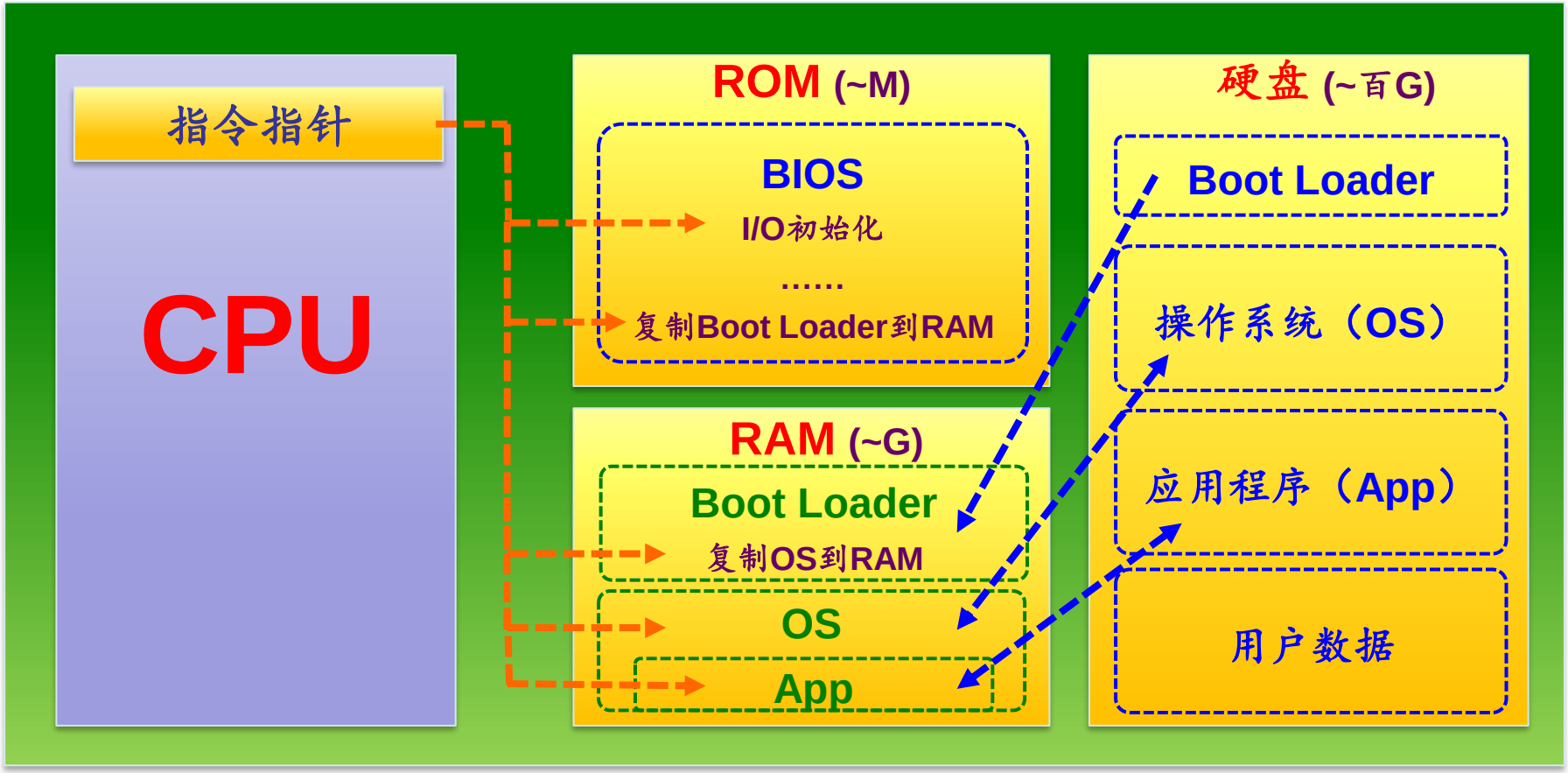


数据通路（Datapath）：指令执行过程中，数据所经过的路径，包括路径中的指令执行部件

指令控制部件（Control）：对执行部件发出控制信号，包括对指令进行译码，生成指令对应的控制信号，控制数据通路的动作

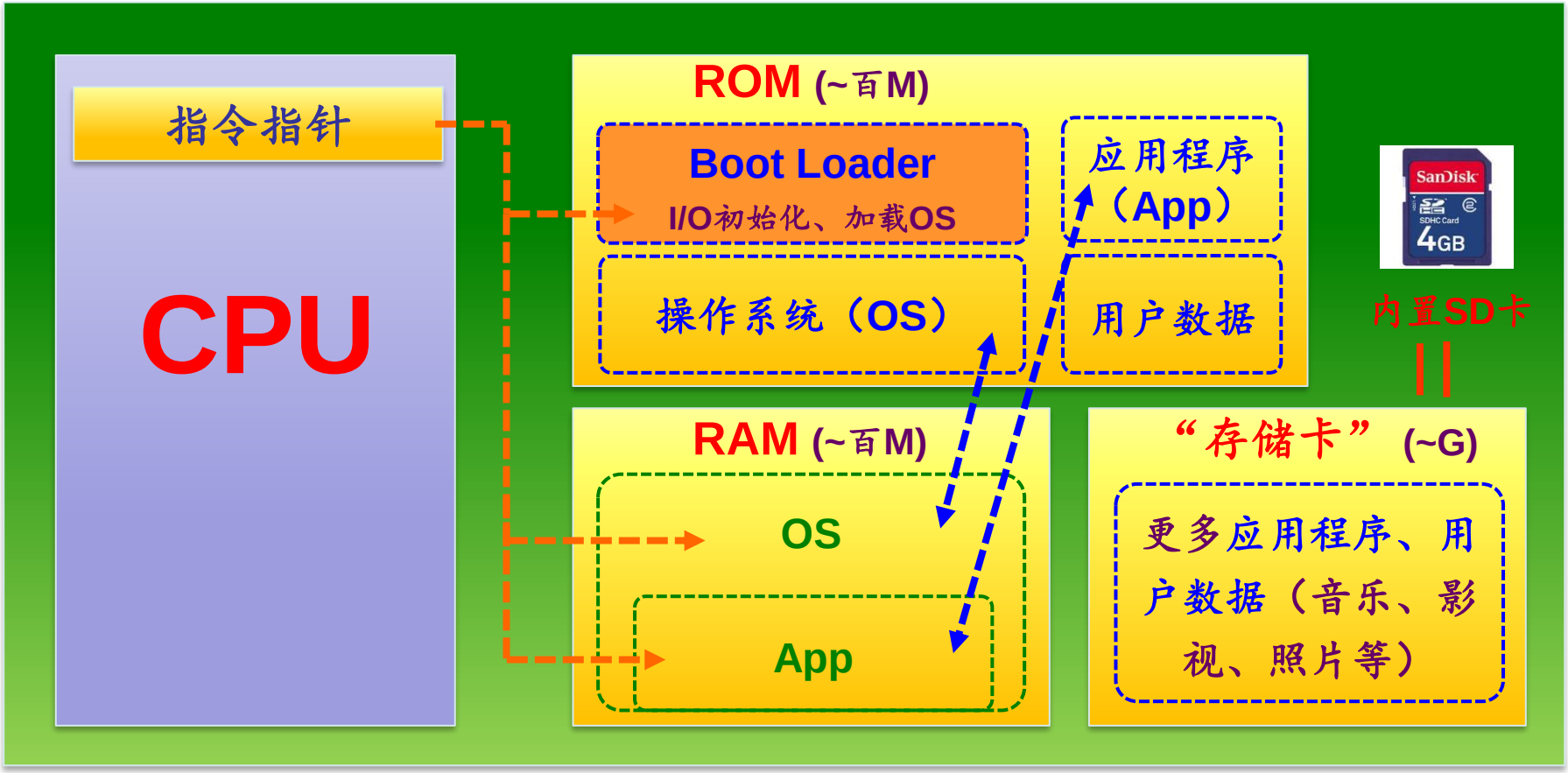
计算机宏观工作流程

个人电脑



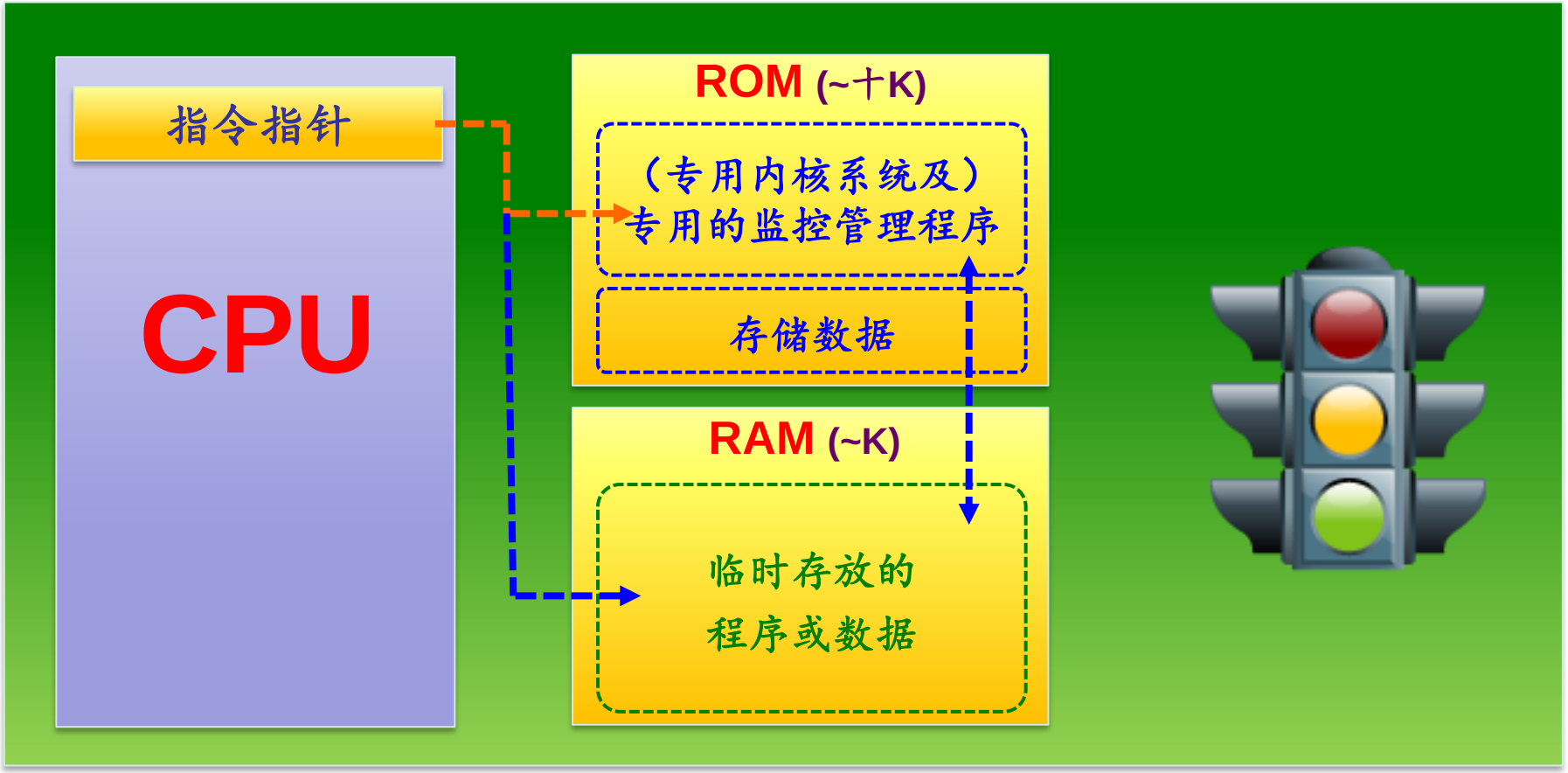
计算机宏观工作流程

智能手机



计算机宏观工作流程

嵌入式设备



提纲

- 计算机工作流程
  - 程序创建执行过程
  - CPU微观工作流程
  - 计算机宏观工作流程

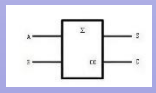
- **计算机指令系统（以MIPS为例）**
  - 指令系统
  - 寻址方式

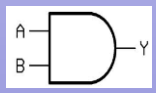
计算机系统抽象层次

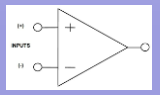
计算机系统的抽象层次

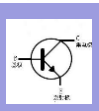
计算机系统

- 应用软件（程序）
- 操作系统（设备驱动、程序）
- 体系结构（指令、寄存器、寻址方式）
- 微体系结构（数据通路、控制器）
- 逻辑（加法器、存储器）
- 数字电路（与门、或门）
- 模拟电路（放大器、滤波器）
- 器件（晶体管）
- 物理（电子）









指令系统

指令系统的作用/目的：

C等高级编程语言

{计算机指令系统}

汇编语言

机器语言：二进制操作码（Exe等可执行文件）

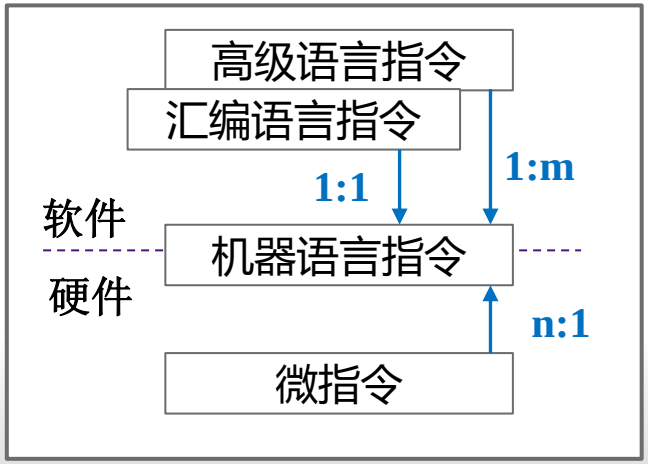
c = a + b;

编译（Compiler）

Mov a, R1
Mov b, R2
Add R1, R2
Mov R2, c

汇编（Assembler）

0000 0001
0000 0010
0010 0010
0000 1010



STM32程序实例


C语言
存储地址
指令代码
汇编语言
int a,b,c;
a=1;
b=2;
c=a+b;
0800021e: 0x00000123 movs r3, #1
08000220: 0x0000fb60 str r3, [r7, #12]
08000222: 0x00000223 movs r3, #2
08000224: 0x0000bb60 str r3, [r7, #8]
08000226: 0x0000fa68 ldr r2, [r7, #12]
08000228: 0x0000bb68 ldr r3, [r7, #8]
0800022a: 0x00001344 add r3, r2
0800022c: 0x00007b60 str r3, [r7, #4]

MSP430程序实例

C语言

```
WDTCTL = WDTPW | WDTHOLD;// stop watchdog timer
int a,b,c;
a=1;
b=2;
c=a+b;
P1OUT=c;
```

存储地址	指令代码	汇编语言
c014:	40B2 5A80 0120	MOV.W #0x5a80,&Watchdog_Timer_WDTCTL
c01a:	431E	MOV.W #1,R14
c01c:	532E	INCD.W R14
c01e:	4EC2 0021	MOV.B R14,&Port_1_2_P1OUT

指令系统

STM32程序实例

C语言

```
int a,b,c;
a=1;
b=2;
if (a>b)
    c=a-b;
else
    c=b-a;
```

存储地址	指令代码	汇编语言
28		a=1;
0800021e:	0x00000123	movs r3, #1
08000220:	0x0000fb60	str r3, [r7, #12]
29		b=2;
08000222:	0x00000223	movs r3, #2
08000224:	0x0000bb60	str r3, [r7, #8]
30		if (a>b)
08000226:	0x0000fa68	ldr r2, [r7, #12]
08000228:	0x0000bb68	ldr r3, [r7, #8]
0800022a:	0x00009a42	cmp r2, r3
0800022c:	0x000004dd	ble.n 0x8000238 <main+32>
31		c=a-b;
0800022e:	0x0000fa68	ldr r2, [r7, #12]
08000230:	0x0000bb68	ldr r3, [r7, #8]
08000232:	0x0000d31a	subs r3, r2, r3
08000234:	0x00007b60	str r3, [r7, #4]
08000236:	0x000003e0	b.n 0x8000240 <main+40>
33		c=b-a;
08000238:	0x0000ba68	ldr r2, [r7, #8]
0800023a:	0x0000fb68	ldr r3, [r7, #12]
0800023c:	0x0000d31a	subs r3, r2, r3
0800023e:	0x00007b60	str r3, [r7, #4]
08000240:	0x00000023	movs r3, #0

指令系统

MSP430程序实例

C语言

```
WDTCTL = WDTPW | WDTHOLD;// stop watchdog timer
int a,b,c;
a=1;
b=2;
if (a>b)
   c=a-b;
else
   c=b-a;
P1OUT=c;
```

存储地址	指令代码	汇编语言
c000:	40B2 5A80 0120	MOV.W #0x5a80,&Watchdog_Timer_WDTCTL
c006:	431D	MOV.W #1,R13
c008:	432E	MOV.W #2,R14
c00a:	9D0E	CMP.W R13,R14
c00c:	3403	JGE (CL1)
c00e:	4D0F	MOV.W R13,R15
c010:	8E0F	SUB.W R14,R15
c012:	3C02	JMP (CL2)
c014:	4E0F	MOV.W R14,R15
c016:	8D0F	SUB.W R13,R15
c018:	4FC2 0021	MOV.B R15,&Port_1_2_P1OUT

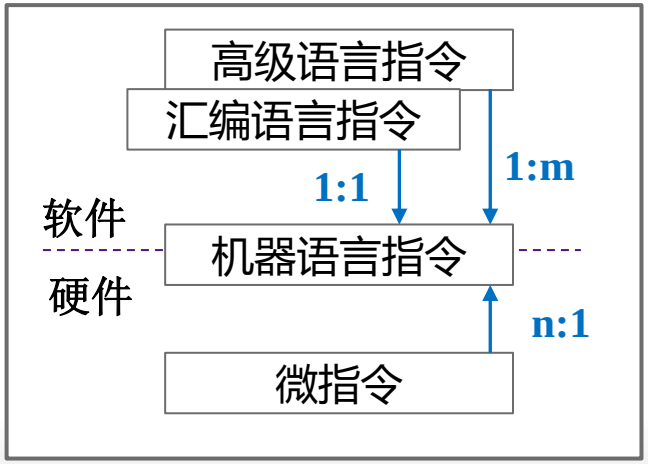
指令系统

机器指令

- CPU能直接执行的命令，简称指令
- 用户控制计算机的工具
- 用户能够控制计算机的最小功能单位
- 计算机本身运行的最小功能单位
- 汇编指令是机器指令的“马甲”，两者一一对应

指令系统

- CPU能执行的机器指令的集合
- 硬件工程师和软件工程师协商的结果
- 硬软件接口



指令系统

1. 指令：要求计算机进行基本操作的命令。
2. 一台计算机所能执行的全部指令的集合称为该计算机的指令系统或指令集。
3. 计算机硬件能够直接识别并执行的程序是机器语言程序，而机器语言中的每一个语句就是一条指令。
4. 指令系统决定了计算机硬件所能完成的全部功能，是计算机系统结构设计者、系统软件设计者和硬件设计者所共同关心的问题。

指令系统分类

指令集分类

CISC：复杂指令集计算机
(Complex Instruction Set Computing)

- 指令条数多、功能齐全
- 硬件实现复杂
- 并行度差

RISC：精简指令集计算机
(reduced instruction set computing)

- 指令数少，寻址方式少，大部分指令单周期执行
- 指令功能简单，指令格式规整，实现简单
- 并行度高，指令适合流水线操作
- 以提高系统性能为目标
  - 80%操作用简单指令（8:2原则）
  - 有较多的通用寄存器和大容量的缓存（Cache），只有存取指令可以访问内存，提高运行速度

两者日趋融合：指令格式复杂，内部实现精简

指令系统

指令格式

- **操作码 + 操作数**
  - **操作码：规定本指令要完成的功能**
  - **操作数：指明本指令要处理数据的来源**

指令系统

指令格式

指令字：完整的一条指令的二进制表示

指令字长：每条指令（转换为二进制位后）占用的存储字节数目

- 指令长度主要取决于操作码的长度、操作数地址的长度、操作数地址的个数。
- 与机器字长有简单的倍数关系
  - 单字长指令：指令长度等于机器字长的指令
  - 半字长指令：指令长度等于半个机器字长的指令
  - 双字长指令：指令长度等于机器字长的两倍的指令
- 指令长度一般应是字节的整数倍
- 多字长指令：
  - 解决寻址较大存储空间的问题
  - 取指令多次访问内存，影响速度，占用空间大

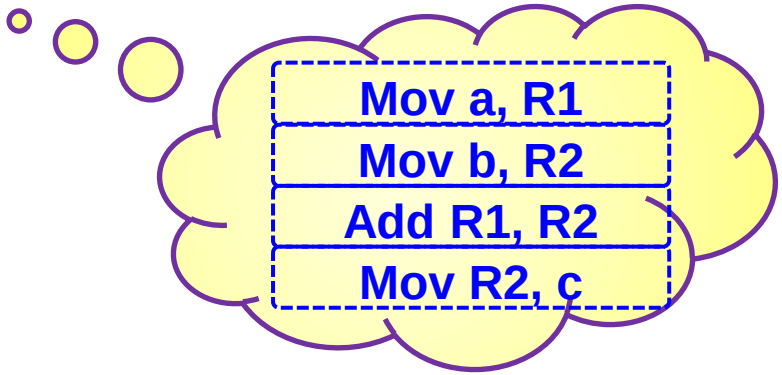
指令系统

指令格式

OP：操作码

SRC：源操作数

DEST：目的操作数



.sz：操作数类型（例如：缺省为Word，.B为Byte，.L为LongWord）

三地址指令

- 指令中给出三个地址，其中两个指明参与操作的源操作数，另一个表示操作结果的存放地址。

指令格式：

操作码 OP	A3	A1	A2

A3 ← (A1) OP (A2)

二地址指令

- 指令中给出两个地址，分别指出参与操作的两个源操作数，而其中有一个又表示操作结果的存放地址。

指令格式：

操作码 OP	A1	A2

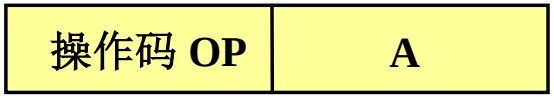
A1 ← (A1) OP (A2)

指令系统

一地址指令

- 指令中给出一个地址

指令格式：



- 在两种情况下可能采用一地址指令
  - 指令本身只需要一个操作数

    A ← OP (A)
  - 指令操作需要两个操作数，指令中指明一个操作数，而另外一个操作数在默认的某个地方。

  如：累加器AC中，操作结果存放到累加器AC中。

  AC ← (AC) OP (A)

指令系统

零地址指令

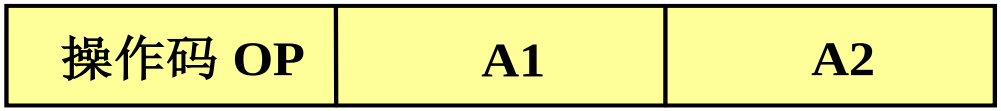
- 零地址指令中只有操作码，没有地址码。

  指令格式：

- 通常在两种情况下可能采用零地址指令
  - 指令本身不需要任何操作数
  - 指令中所需的操作数是隐含指定的

根据指令中操作数的物理位置分类



A1 ← (A1) OP (A2)

- 根据存放操作数的位置不同，分为3种。
  - **寄存器-寄存器型（R-R型）指令**
  - **寄存器-存储器型（R-M型）指令**
  - **存储器-存储器型（M-M型）指令**

指令系统

指令格式

指令字：完整的一条指令的二进制表示

指令字长：每条指令（转换为二进制位后）占用的存储字节数目

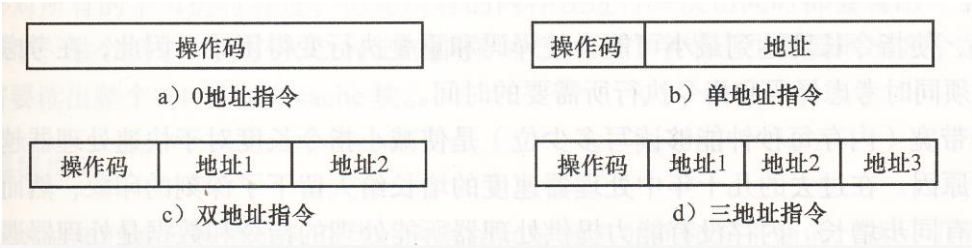
- 在一个指令系统中，如果每条指令的长度都相同，则称为固定长度编码格式。
- 如果不同指令的长度随指令功能而不同，则称为可变长度编码格式。
  - **如奔腾系列机的指令系统中，指令长度可以从一个字节到12个字节。**

指令系统

指令格式： 指令中操作码和操作数的二进制位的分配方案

指令字长决定因素：

- 操作码长度：定长？变长？
- 操作数个数
  - 无操作数：停机、空操作、关中断等
  - 单操作数： INC；IN
  - 双操作数
  - 多操作数
- 操作数来源：寻址方式？



指令系统

指令操作码：

1. 指令系统中的每一条指令都有唯一确定的操作码，不同指令的操作码是不相同的。
   - 操作码的长度决定了指令系统的最大规模
   - 若操作码的位数为n位，则该指令系统最多能有2^n条指令。
2. 固定长度操作码
   - 所有指令操作码的长度都是固定的，且集中放在指令的一个字段内 。
   - 有利于简化硬件设计，减少指令译码时间。
   - 很多现代计算机都采用了固定长度操作码。

指令系统

2. 可变长度操作码

- 指令系统中操作码的长度有多种，不同指令的操作码的长度不完全相同。
  - 使用频率高的指令使用短的操作码
  - 使用频率低的指令使用较长的操作码
- 可以缩短操作码的平均长度，但会使硬件设计复杂化，增加指令译码的时间和难度。

指令系统

3. 扩展操作码技术

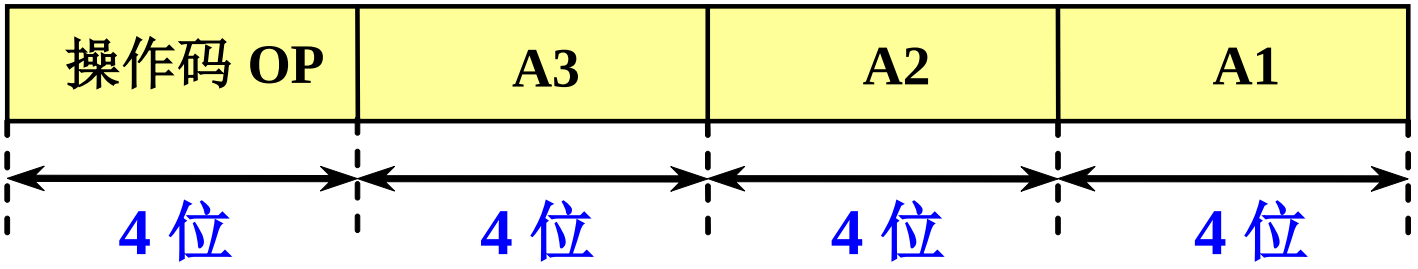
- 将操作码设计为几种不同的固定长度，且相互之间按某种规则进行扩展。
- 优点
  - 可以简化硬件设计；
  - 当指令总长度一定时，可以使操作码的长度随地址数的增加而减少，不同地址数的指令的操作码的长度也不同，从而有效地缩短指令总长度。
- 扩展操作码的方法

  等长扩展、不等长扩展

例：固定长度操作码

指令字长16位

通用寄存器16个，固定3个操作数



采用固定长度操作码，该指令系统最多只能有16条指令

指令系统

例：等长扩展操作码

指令字长16位

通用寄存器16个，最多3个操作数

采用等长扩展，操作码分别为4位、8位、12位、16位

有61条指令

指令系统

例：不等长扩展操作码

指令字长16位

通用寄存器16个，最多3个操作数

操作码 操作数

4位	12位

可安排？条3操作数指令：0000-1110

操作码 操作数

1111	4位	8位

可安排？条双操作数指令：1111 0000
1111 1101

操作码 操作数

1111	111X	4位	4位

可安排 ？条单操作数指令

操作码

1111	1111	1111	4位

可安排 ？条无操作数指令

指令系统

指令字长16位

通用寄存器16个，最多3个操作数

操作码 操作数

4位	12位

可安排15条3操作数指令：0000-1110

操作码 操作数

1111	4位	8位

可安排14条双操作数指令：1111 0000

⋮

1101

操作码 操作数

1111	111X	4位	4位

可安排31条单操作数指令 1111 1110 0000

⋮

1111

1111 1111 0000

⋮

1110

操作码

1111	1111	1111	4位

可安排16条无操作数指令 1111 1111 1111 0000

⋮

1101

采用不等长扩展，可支持76条指令

指令系统

指令格式设计原则

- 操作码字段要能满足指令功能完备性的要求
- 操作数字段要能满足不同寻址方式和寻址空间（寄存器、存储器）的要求
- 在满足上述要求的情况下，尽量采用规则、短小的指令格式，以达到实现简单、运行高效的目的

指令系统

指令系统设计原则

- 完备性

  —— 指令功能齐全，方便程序员使用

- 兼容性

  —— 包容旧指令系统，老程序能在新的CPU上运行

- 规则性

  —— 指令格式规则简单统一，易于实现

- 高效性

  —— 实现复杂度低，运行速度高

指令系统

- 规则性：简单性来自规则性
- 高效性：高效性来自简单性（越小越快）

  - 指令格式规则简单统一，易于实现
  - **d=a+b**
  - **Add d,a,b**
  - **d=a+b+c**
    - 操作数不固定，指令复杂化：**Add d,a,b,c**
      - 增加加法器、寄存器、控制器、代码的复杂度
    - 规格化：**Add d,a,b; Add d,d,c**
      - 硬件设计、指令系统设计规整、简单
  - 复杂运算（多媒体）：专用指令、专用硬件

指令系统

常用指令类型

序号	分类	基本指令
1	算术运算、关系运算	加、减、乘、除、比较、扩展
2	逻辑操作	与、或、非、异或、测试
3	移位指令	逻辑移位、算术移位、循环移位
4	数据传送	将存储器中的数据读到寄存器，将寄存器中的数据写入存储器
5	跳转、分支转移	条件转移、条件延迟、分支、子程序
6	系统控制	各种系统寄存器的存取、设置

MIPS指令概览

MIPS（Microprocessor without interlocked piped stages）

- 是80年代初由斯坦福大学教授约翰·轩尼诗与他的团队创立；
- 属于精简指令集计算机RISC
- MIPS指令集有MIPS I, MIPS II, MIPS III, MIPS IV, MIPS V, MIPS32, MPIS64多个版本；
- 早期主要用于嵌入式系统，如Windows CE设备、路由器、家用网关和视频游戏机，现在已经在PC机、服务器中得到广泛应用
- 2007年8月16日，“龙芯”获得MIPS处理器IP的全部专利和总线、指令集授权

MIPS指令系统（32位）

MIPS基本指令

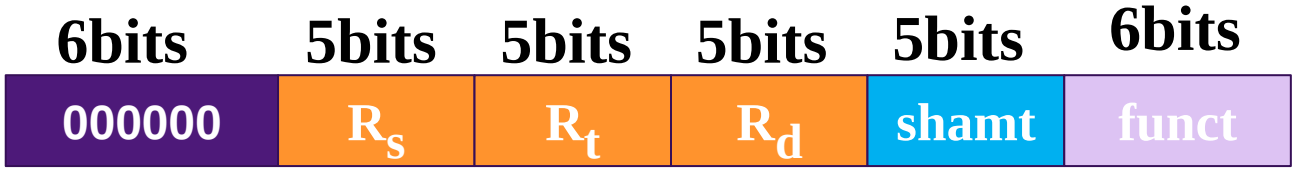
序号	分类	基本指令
1	算术运算	add, sub, addi, addu, mul, mulu, div, divu
2	逻辑、移位操作	and, or, andi, ori, sll, srl, lui
3	数据传送	lw, sw, lb, lbu, sb
4	分支转移	beq, bne, bnez, slt, slti, sltu,sltiu
5	无条件跳转	j, jr, jal

指令后缀： i 表示立即数；u 表示无符号；w 表示字(Word)操作；b 表示字节(Byte)操作

MIPS指令系统（32位）

- 只有三种指令格式

R型指令



寄存器型

- Rs， Rt 分别为第一、二源操作数；Rd为目标操作数

I型指令

立即数型

- 双目、Load/Store: Rs和立即数是源操作数， Rt为目标操作数
- 条件转移： Rs, Rt均为源操作数

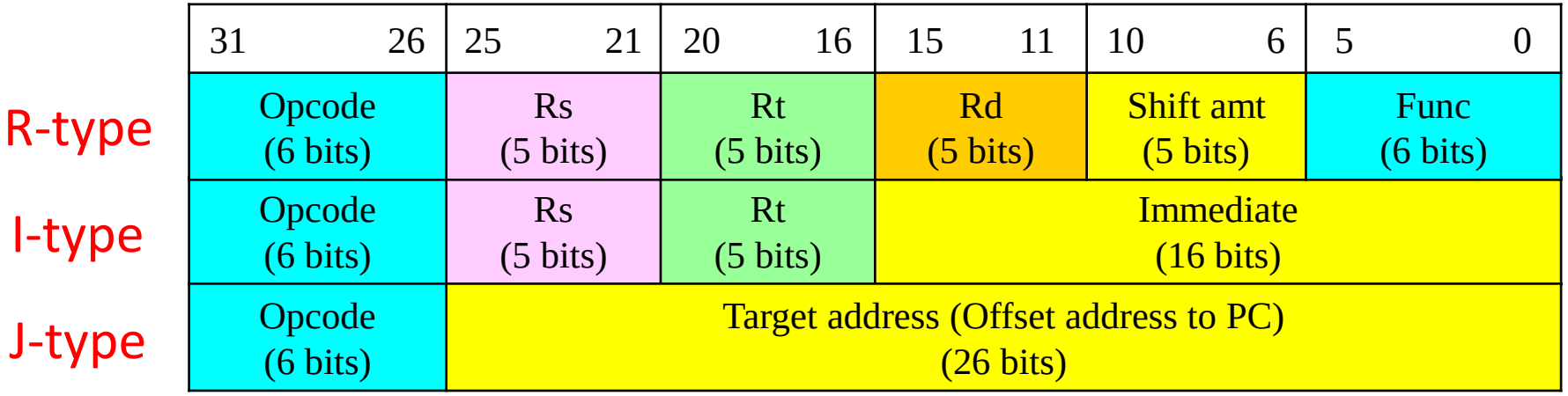
J型指令

跳转

- 26位立即数作为跳转目标地址的部分地址

MIPS指令系统（32位）

32位MIPS系统指令格式



- Opcode: partially specifies what instruction it is; **Equal to 0 for all R-Format instructions**
- Funct: combined with opcode, this number exactly specifies the instruction
- Rs (Source Register): generally used to specify register containing first operand
- Rt (Target Register): generally used to specify register containing second operand
- Rd (Destination Register): generally used to specify register which will receive result of Opcode
- Shamt: This field contains the amount a shift instruction will shift by.
- Immediate: address offset or immediate value
- Target address (Offset address to PC): target address of the jump instruction

MIPS指令系统（32位）

32位MIPS系统指令特点

1. 指令定长（32 bits）
   - 操作码长度固定（6位），在指令格式中位置固定
   - 寄存器编码长度固定（5位），可寻址32个寄存器，在指令格式中位置固定
   - 立即数长度固定（16位），在指令格式中位置固定
2. 由操作码决定寻址方式
3. 运算操作主要基于寄存器/立即数

MIPS指令系统（32位）

MIPS系统寄存器

寄存器名称	编码	用途
$zero	0	Constant 0
$at	1	Reserved for assembler
$v0 ~ $v1	2~3	Expression evaluation and results of a function
$a0 ~ $a3	4~7	Arguments 13
$t0 ~ $t7	8~15	Temporary (not preserved across call)
$s0 ~ $s7	16~23	Saved temporary (preserved across call)
$t8 ~ $t9	24~25	Temporary (not preserved across call)
$k0 ~ $k1	26~27	Reserved for OS kernel
$gp	28	Pointer to global area
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address (used by function call)

MIPS指令系统（32位）

MIPS基本指令

- R-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	32
0000 00	10 010	1 0011	1000 1	000 00	10 0000
add $1, $2, $3 # $1=$2+$3, R-type					

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	34
0000 00	10 010	1 0011	1000 1	000 00	10 0010
sub $s1, $s2, $s3 # $s1=$s2-$s3, R-type					

MIPS指令系统（32位）

MIPS基本指令

- R-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	36
0000 00	10 010	1 0011	1000 1	000 00	10 0100
and $s1, $s2, $s3 # $s1=$s2 AND $s3, R-type					

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	37
0000 00	10 010	1 0011	1000 1	000 00	10 0101
or $s1, $s2, $s3 # $s1=$s2 OR $s3, R-type					

MIPS指令系统（32位）

MIPS基本指令

- R-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	0	18	17	10	2
0000 00	00 000	1 0010	1000 1	010 10	00 0010
srl $s1, $s2, 10	#$s1=shift($s2) right logical 10 bits, R-type				

0	0	18	17	10	0
0000 00	00 000	1 0010	1000 1	010 10	00 0000
sll $s1, $s2, 10	# $s1=shift($s2) left logical 10 bits, R-type				

MIPS指令系统（32位）

MIPS基本指令

- R-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shift amt (5 bits)	Func (6 bits)
0	18	19	17	0	42
0000 00	10 010	1 0011	1000 1	000 00	10 1010
slt $s1, $s2, $s3 # set if less than $s2<$s3, $s1=1; else $s1=0, R-type					

- R-type 指令

0	31	0	0	0	8
0000 00	11 111	0 0000	0000 0	000 00	00 1000
jr $ra # jump register $ra, R-type					

MIPS指令系统（32位）

MIPS基本指令 ■ I-type指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Address/Immediate (16 bits)		
8	18	17	100		
0010 00	10 010	1 0001	0000 0000 0110 0100		
addi $s1, $s2, 100 # $s1=$s2+100, I-type					

12	18	17	100
0011 00	10 010	1 0001	0000 0000 0110 0100
andi $s1, $s2, 100 # $s1=$s2 AND 100, I-type			

13	18	17	100
0011 01	10 010	1 0001	0000 0000 0110 0100
ori $s1, $s2, 100 # $s1=$s2 OR 100, I-type			

MIPS指令系统（32位）

MIPS基本指令

- I-type 指令

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Address/Immediate (16 bits)		
35	18	17	100		
1000 11	10 010	1 0001	0000 0000 0110 0100		
lw $1, 100($2) # load memory word $1=Memory[$2+100], I-type					

43	18	17	100
1010 11	10 010	1 0001	0000 0000 0110 0100
sw $s1, 100($s2) # save memory word $Memory[$s2+100]=$s1, I-type			

MIPS指令系统（32位）

MIPS基本指令 ■ I-type指令

R：通用寄存器
PC：指令指针寄存器

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Address/Immediate (16 bits)		
15	0	17	100		
0011 11	00 000	1 0001	0000 0000 0110 0100		
lui $s1, 100 # load & shift left unsigned immediate $s1=100«16, I-type					

MIPS指令系统（32位）

MIPS基本指令 ■ I-type指令

R：通用寄存器
PC：指令指针寄存器

4	17	18	25
0001 00	10 001	1 0010	0000 0000 0001 1001
beq $s1, $s2, 25 # branch if $s1=$s2, goto PC+4+25*4, I-type			

5	17	18	25
0001 01	10 001	1 0010	0000 0000 0001 1001
bne $s1, $s2, 25 # branch if $s1≠$s2, goto PC+4+25*4, I-type			

MIPS指令系统（32位）

MIPS基本指令

- J-type 指令

New PC = { PC[31..28], target address, 00 }
= { 1010, 00000000000000100111000100, 00 }
32 bit address = { 4 bits , 26 bits , 2 bits }

31 26	25 21	20 16	15 11	10 6	5 0
Opcode (6 bits)	Target address (26 bits)				
2	2500				
0000 10	00 0000 0000 0000 1001 1100 0100				
j 2500 # jump to 2500, I-type					

指令系统

指令系统设计实验

- 指令字长：16位
- 数据字长：16位
- 通用寄存器数量：4个
- 寻址空间：256字节
- 功能：15个

功能			操作码
算术运算	不带进位加减	不带进位加	0010
		不带借位减	0011
		不带进位立即数加	1010
	带进位加减	带进位加	0110
		带借位减	0101
关系运算	无符号数比较	无符号数比较	0100
逻辑运算	寄存器之间的与或运算	与	0000
		或	0001
	寄存器与立即数与或运算	立即数与	1000
		立即数或	1001
数据传输	将存储器中的数据读到寄存器	读存储器	1011
	将寄存器中的数据写入存储器	写存储器	1100
跳转	有条件跳转（根据比较结果）	相等时跳转	1101
		不等时跳转	1110
	无条件跳转	无条件跳转	0111

指令系统

指令系统设计实验

R：通用寄存器
PC：指令指针寄存器

	操作名称	操作码	汇编语言格式指令	执行操作
R型指令	与	0000	AND Rd, Rs, Rt	Rd←Rs and Rt; PC ← PC + 1
	或	0001	OR Rd, Rs, Rt	Rd←Rs or Rt; PC ← PC +1
	不带进位加	0010	ADD Rd, Rs, Rt	Rd←Rs+ Rt; PC ← PC + 1
	不带借位减	0011	SUB Rd, Rs, Rt	Rd←Rs– Rt; PC ← PC + 1
	带进位加	0110	ADDC Rd, Rs, Rt	Rd←Rs+ Rt+C; PC ← PC + 1
	带借位减	0101	SUBC Rd, Rs, Rt	Rd←Rs– Rt-(1-C); PC ← PC + 1
	无符号数比较	0100	SLT Rd,Rs,Rt	If Rs<Rt, Rd=1 else Rd=0; PC ← PC + 1
I型指令	立即数与	1000	ANDI Rt, Rs, imm	Rt←Rs and imm; PC ← PC +1
	立即数或	1001	ORI Rt, Rs, imm	Rt←Rs or imm; PC ←PC +1
	立即数加	1010	ADDI Rt, Rs, imm	Rt←Rs+ imm; PC ← PC +1
	读存储器	1011	LW Rt, Rs, imm	Rt←MEM[Rs+imm]; PC ←PC +1
	写存储器	1100	SW Rt, Rs, imm	MEM[Rs+imm] ←Rt; PC ← PC +1
	相等时跳转	1101	BEQ Rs, Rt, imm	If Rt=Rs, PC←PC+imm+1 else PC←PC+1
	不等时跳转	1110	BNE Rs, Rt, imm	If Rt!=Rs, PC←PC+imm+1 else PC←PC+1
J型指令	无条件跳转	0111	JMP imm	PC ← imm

指令系统设计实验

根据功能与性能要求，仿照MIPS指令格式进行设计

R型指令

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op				Rs		Rt		Rd		---					

I型指令

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op				Rs		Rt		Imm							

J型指令

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Op				----				Imm							

Op：指令操作码
Rs：第1个操作数的寄存器号
Rt：第2个操作数的寄存器号
Imm：立即数
Rd：第3个操作数的寄存器号
---：表示任意值，编码时通常取0

指令系统

指令系统设计实验

1. 指令定长（16 bits）
   - 操作码长度固定（4位），在指令格式中位置固定
   - 寄存器编码长度固定（2位），可寻址4个寄存器，在指令格式中位置固定
   - 立即数长度固定（8位），在指令格式中位置固定
2. 由操作码决定寻址方式
3. 运算操作主要基于寄存器/立即数

课后作业：

复习：

《计算机组成原理》第6章《计算机执行程序的过程》、第7章《指令系统》

预习：

《计算机组成原理》第4章《组合逻辑电路》

思考题：

《计算机组成原理》 P188页习题 7.1、7.2、7.5、7.7、7.11、7.12题

谢 谢

请不要将课件上传到公共网络平台上~~