计算机组成原理


# 五、指令系统

# 本章主要内容

- **5.1 指令系统概述**
- 5.2 指令格式
- 5.3 寻址方式
- 5.4 指令类型
- 5.5 指令格式设计
- 5.6 CISC与RISC
- 5.7 指令系统举例

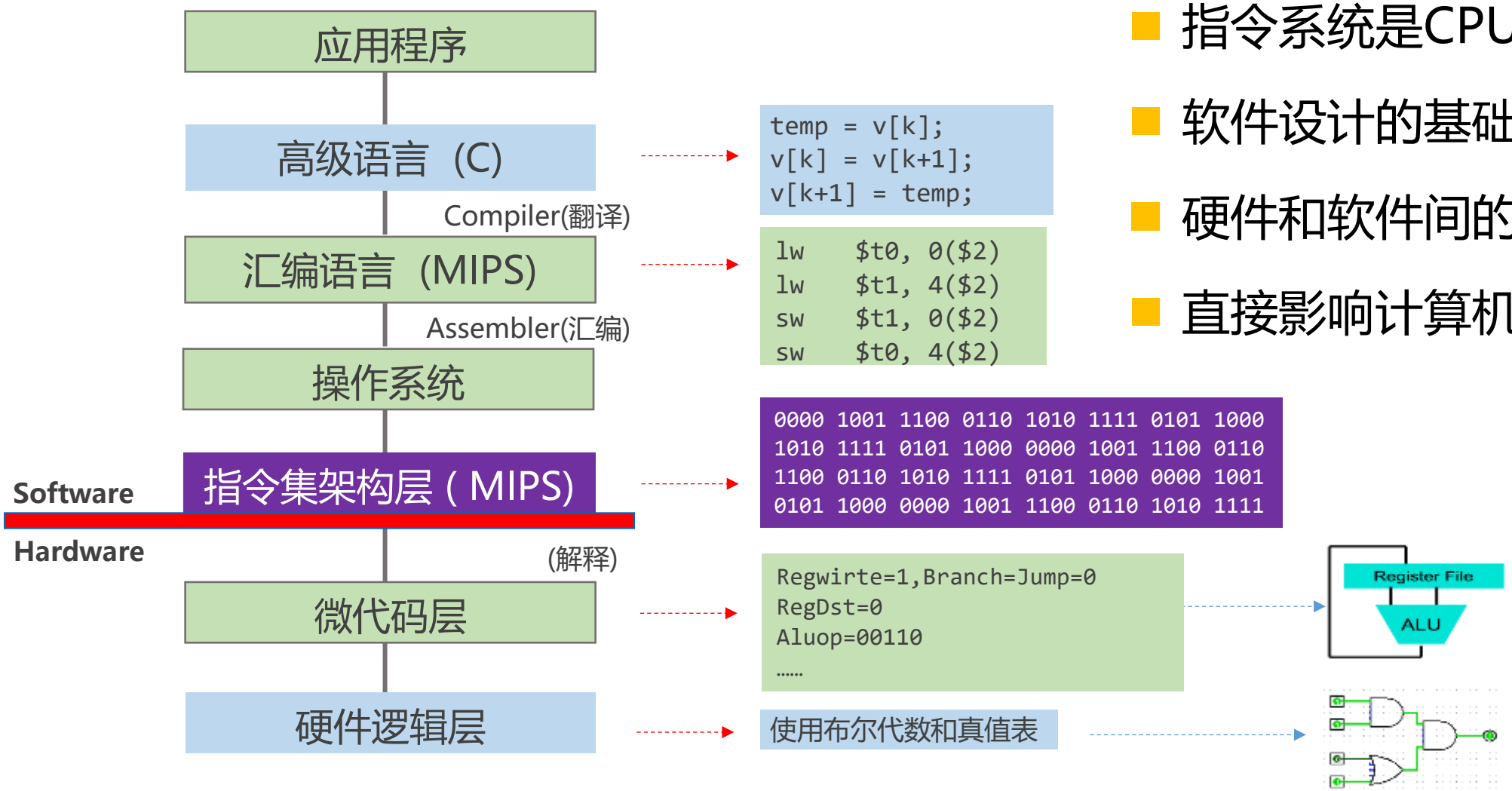
# 指令

- 指令：控制计算机执行某种操作，如加、减、传送、转移等，的命令
- 指令系统：计算机中所有指令的集合
- 指令提供的信息包括指令执行的操作、操作数的来源、操作结果的存放地等

# 指令系统基本概念

- 软件层次的指令
  - 高级语言指令
  - 汇编语言指令
- 硬件层次的指令
  - 机器语言指令
  - 微指令
- 软件层次的指令需要“翻译”成机器语言指令后才能被计算机硬件识别并执行

# 计算机指令系统层次




- 指令系统是CPU设计的依据
- 软件设计的基础
- 硬件和软件间的界面
- 直接影响计算机系统性能

# 指令系统基本概念

- 完备性：指令丰富，功能齐全，使用方便
- 有效性：程序占空间小，执行速度快
- 规整性：
  - 对称性　(对不同寻址方式的支持)
  - 匀齐性　(对不同数据类型的支持)
  - 一致性　(指令长度和数据长度的一致性)
- 兼容性：系列机软件向上兼容

# 本章主要内容

- 5.1 指令系统概述
- 5.2 指令格式
- 5.3 寻址方式
- 5.4 指令类型
- 5.5 指令格式设计
- 5.6 CISC与RISC
- 5.7 指令系统举例

# 指令格式

- **指令**是计算机中传输控制信息的载体
- **指令格式**：用二进制代码表示指令的结构形式
  - 指令要求计算机处理什么数据？
  - 指令要求计算机对数据做什么处理？
  - 计算机怎样才能得到要处理的数据？

指令的操作数需要解决的问题

指令的操作码需要解决的问题

指令的寻址方式需要解决的问题

# 操作码(OP)与地址码(AC)

- **操作码字段**
  - 解决进行何种操作的问题
- **地址码字段**
  - 解决处理什么操作数的问题，地址码可以包括多个操作数
  - 而通过何种方式获取操作数通常由寻址方式字段决定
  - 寻址方式字段决定地址码中操作数存放的位置和访问方式，寻址方式字段可以包含在地址码字段中

# 指令字长度

- **指令字长度**：指令中包含二进制代码的位数
- 字长与机器字的长度有关：**单字长，双字长，半字长**
  - 指令字越长，地址码长度越长，可直接寻址空间越大
  - 指令字越长，占用空间越大，取指令越慢
- **定长指令**：结构简单，控制线路简单，MIPS指令
- **变长指令**：结构灵活，充分利用指令长度，控制复杂，X86指令

# 指令地址码

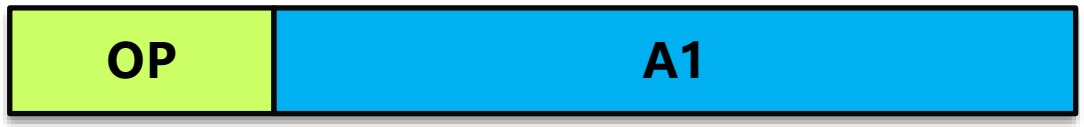
三地址指令


(A1) OP (A2) → A3

二地址指令


(A1) OP (A2) → A1

单地址指令




(AC) OP (A1) → AC

零地址指令

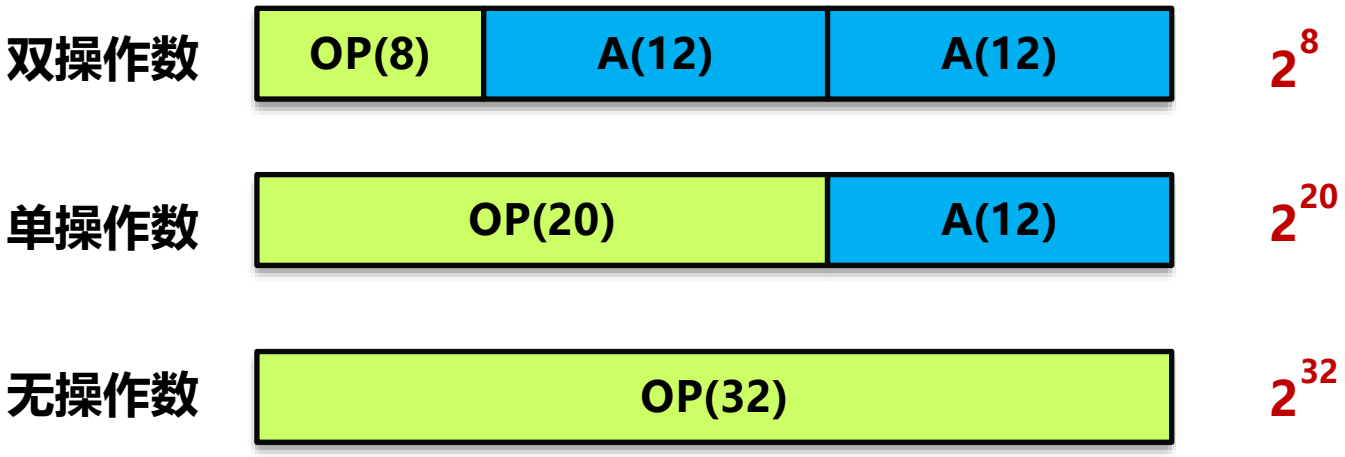

如停机、空操作、开关中断等

**一条指令中所含操作数地址的数量划分**

# 指令操作码

- 操作码字段
  - 具体进行什么运算操作
  - 不同功能的指令其操作码的编码不同
- 定长操作码
  - 操作码的长度固定
  - 在指令中的位置也是固定的
  - 假设指令系统包含 m 条指令，则操作码的位数 n 应该满足$n \geq \log.m$
- 变长操作码
  - 变长操作码中操作码的长度可变
  - 操作码的位置也不固定
  - 可以有效压缩指令操作码的平均长度，便于用较短的指令字长表示更多的操作类型

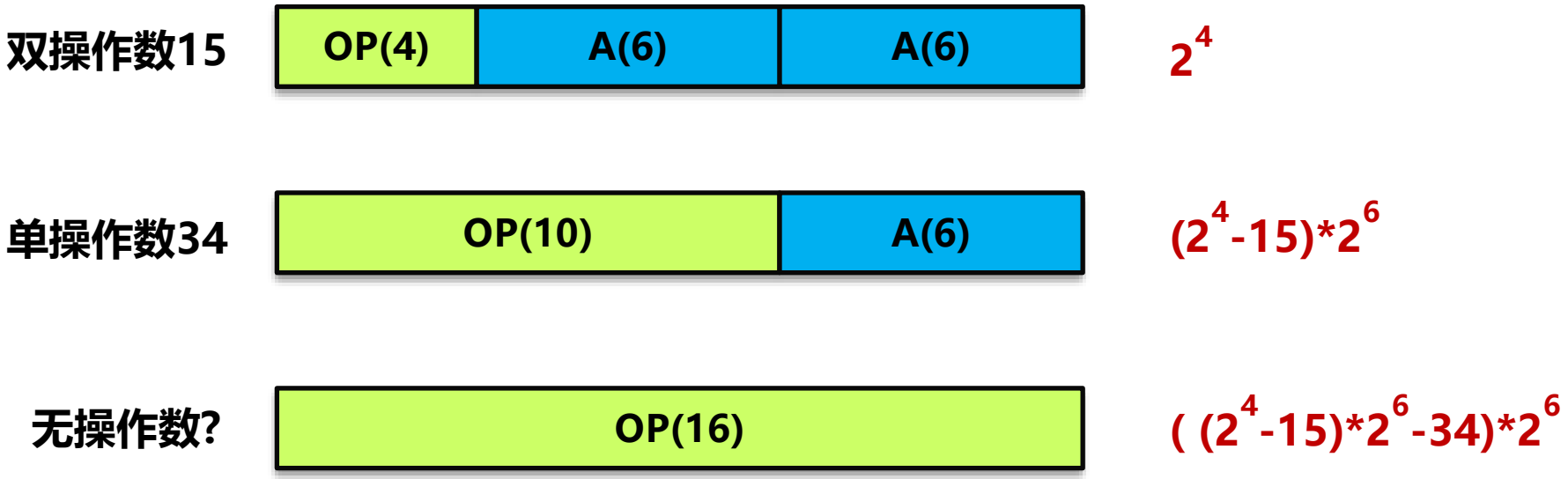
# 扩展操作码




- 采用扩展操作码技术来实现变长操作码，其基本思想是操作码的长度随地址码数目减少而增加
- 3种指令操作码部分不得重叠，否则无法区分，译码
- 设双操作数指令数为k，显然$k<2^8$
- $2^8$-k为多余状态，可用于表示其他类型指令
- 可用于单操作数指令的条数=（$2^8$-k）*$2^{12}$，$2^{12}$ 是多余12位组合

# 扩展指令举例

- 设某指令系统指令字长16位，每个地址码为6位。若要求设计二地址指令15条、一地址指令34条，问最多还可设计多少条零地址指令?



- 例题 5-1

# 本章主要内容

- 5.1 指令系统概述
- 5.2 指令格式
- 5.3 寻址方式
- 5.4 指令类型
- 5.5 指令格式设计
- 5.6 CISC与RISC
- 5.7 指令系统举例

# 寻址方式

## 寻址方式

- 寻找指令或操作数有效地址的方法
- 计算机在运行程序之前必须把指令和数据存放在主存的相应地址单元
- 运行程序时，不断地从主存取指令和数据
- 由于主存是基于地址访问的存储器，只有获得指令和操作数在主存中的地址(称为有效地址EA)后，CPU才能访问所需的指令和数据

# 寻址方式

- 寻址方式分类
  - **指令寻址**
    - 顺序寻址
    - 跳跃寻址
  - **操作数寻址**
    - 立即寻址、直接寻址
    - 间接寻址、寄存器寻址
    - 寄存器间接寻址、相对寻址
    - 基址\变址寻址、复合寻址

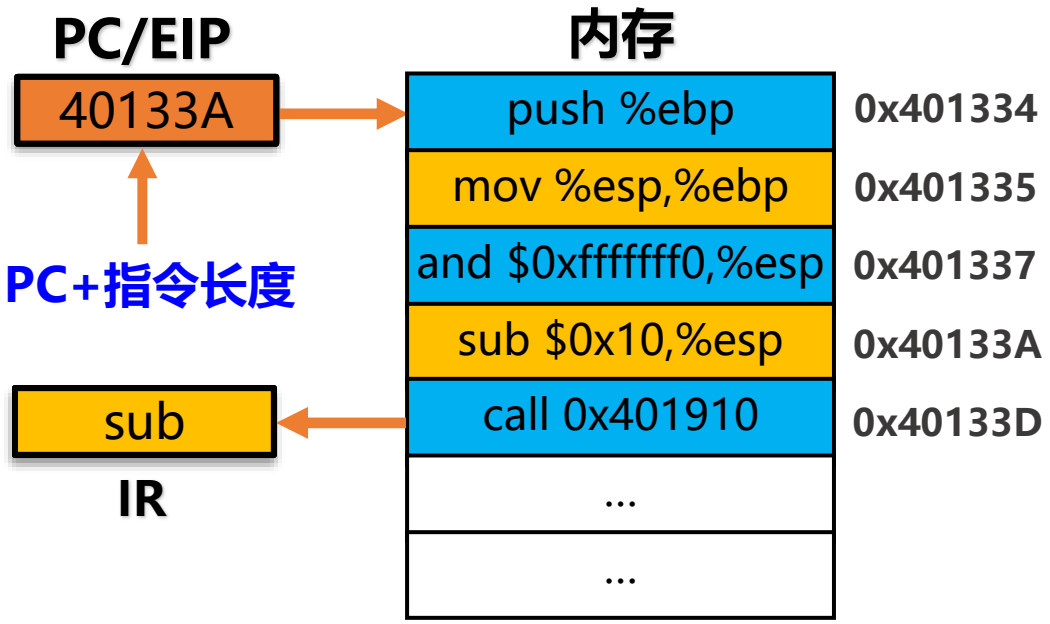
# 顺序寻址

## 顺序寻址方式

- 程序对应的机器指令序列在主存顺序存放
- 执行时从第一条指令开始，逐条取出并执行

## 实现方式

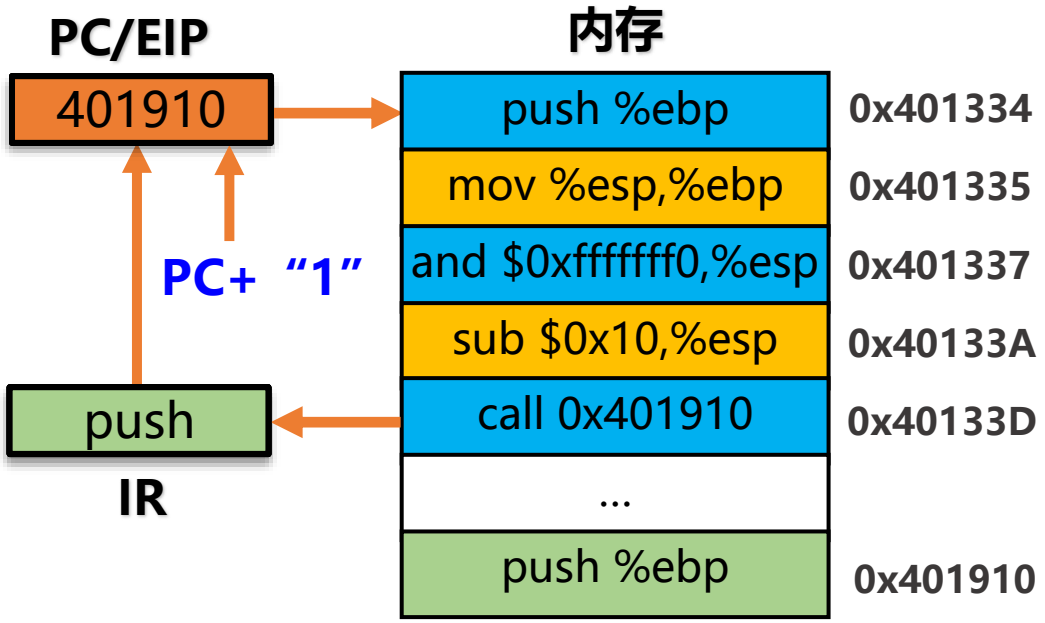
- **程序计数器**（PC）对指令序号进行计数
- PC存放下条指令地址，初始值为程序首址
- 执行一条指令，PC=PC+当前指令字节长度




**Mem[pc++] → IR**

# 跳跃寻址

- **跳跃寻址方式**：当程序中出现分支或循环时，就会改变程序的执行顺序
  - 下条指令地址不是PC++得到，而是由指令本身给出
  - 跳跃的处理方式是重新修改PC的内容，然后进入取指令阶段




**IR(A) → PC**

# 操作数的寻址方式

- 形成操作数有效地址的方法
  - 单地址指令地址码的构成: mode ，D
  - 实际有效地址为 E, 实际操作数 S
  - S= (E)

# 立即寻址

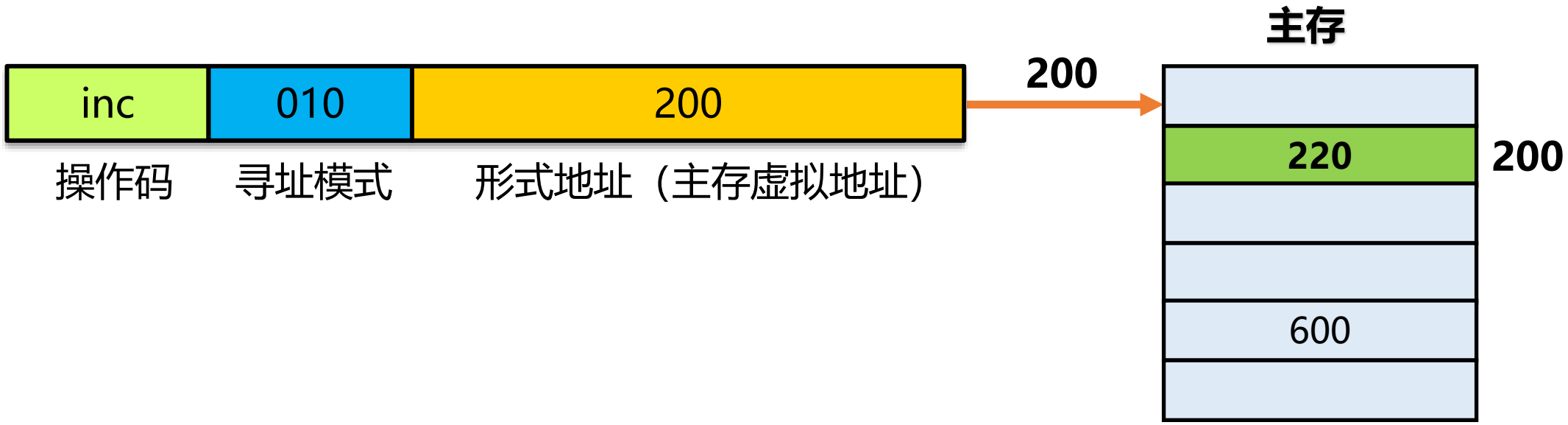
- 地址码字段是操作数本身
  - S=D
  - 例:　MOV　AX,38H　　(38H → AX)

# 寄存器寻址(Register Addressing)

- 操作数在CPU的内部寄存器中.
  - AX,BX,CX,DX
  - PUSH　AX　　E=R

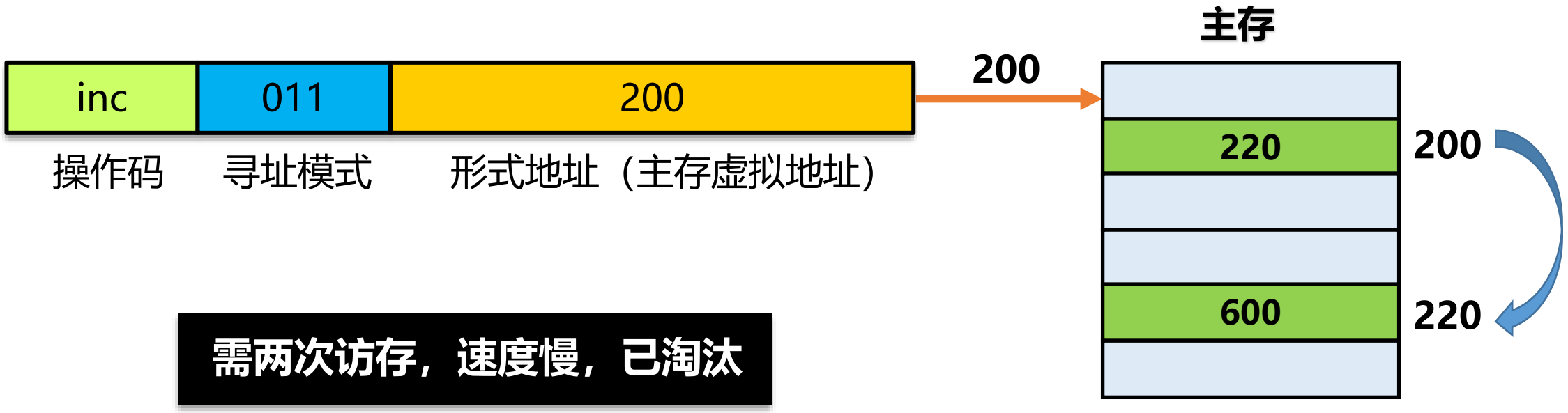
# 直接寻址(Direct Addressing)

- 地址码字段直接给出操作数在内存的地址. E=D
- inc [200]



# 间接寻址(Indirect Addressing)

- D单元的内容是操作数地址，D是操作数地址的地址
- E=(D)　S=((D))

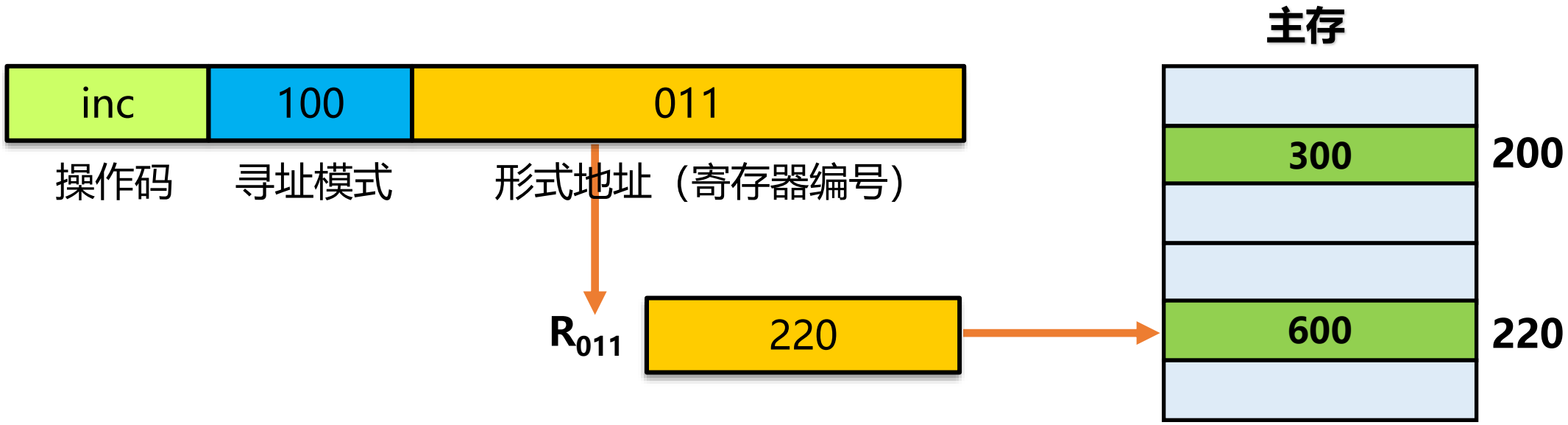



**需两次访存，速度慢，已淘汰**

- 例题5-2

# 寄存器间接寻址 (Register Indirect Addressing)

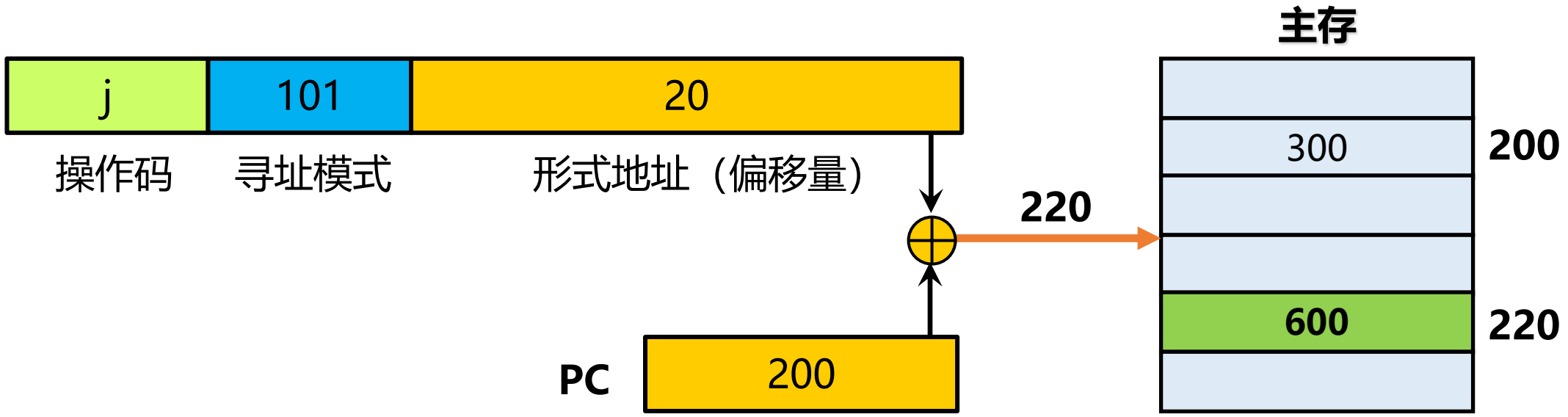
- D单元的内容是操作数的地址,D是操作数地址的地址
- E=(R)　　inc [BX]



# 相对寻址 (Relative Addressing)

- 指令中的D加上PC的内容作为操作数的地址.
- E=D+(PC)

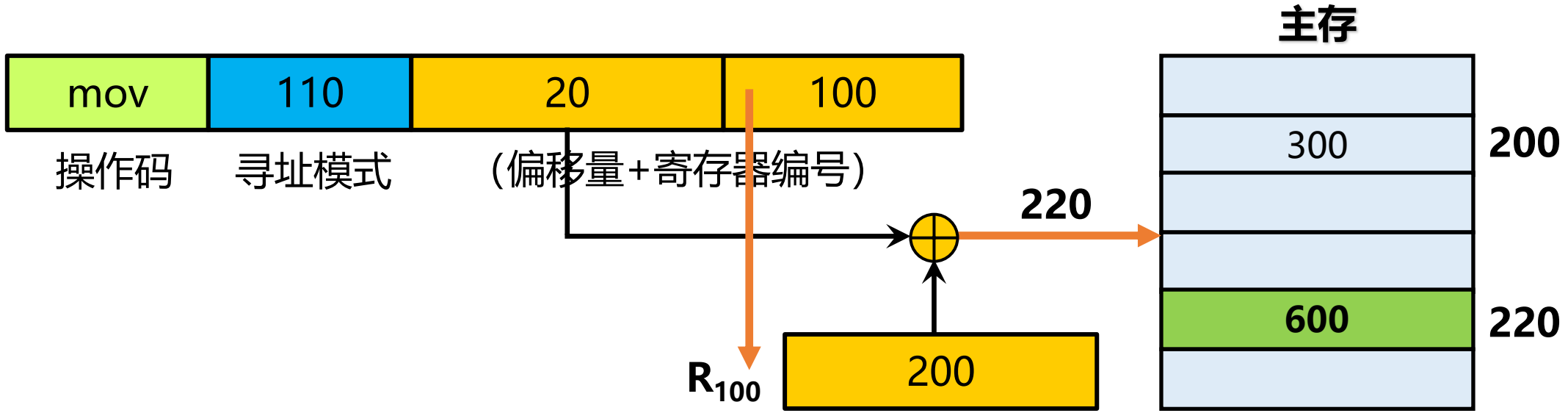
**(PC)+D 还是(PC)+1+D?**



- 例题5-3

# 基址/变址寻址

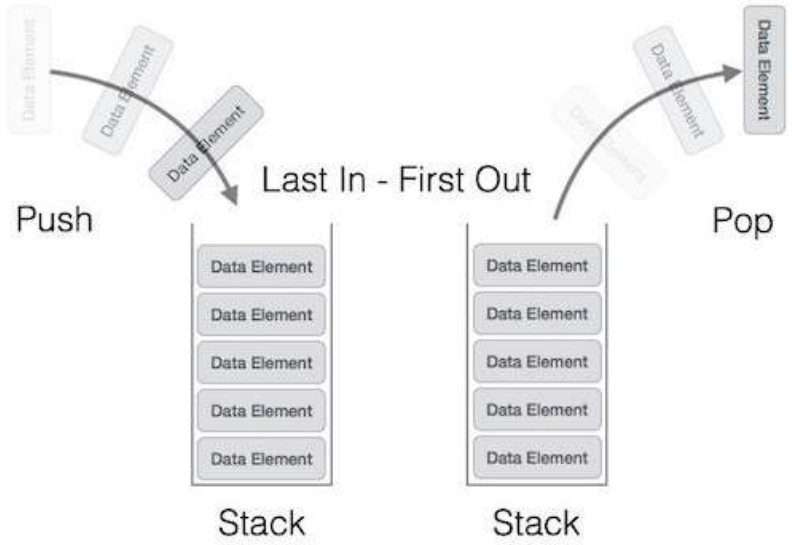
- 操作数地址为基址/变址寄存器+偏移量　　基址寄存器一般不修改
- E=D+(R)
- MOV AX, 32[SI]　　SI,DI 都称为变址寄存器



# 堆栈寻址方式

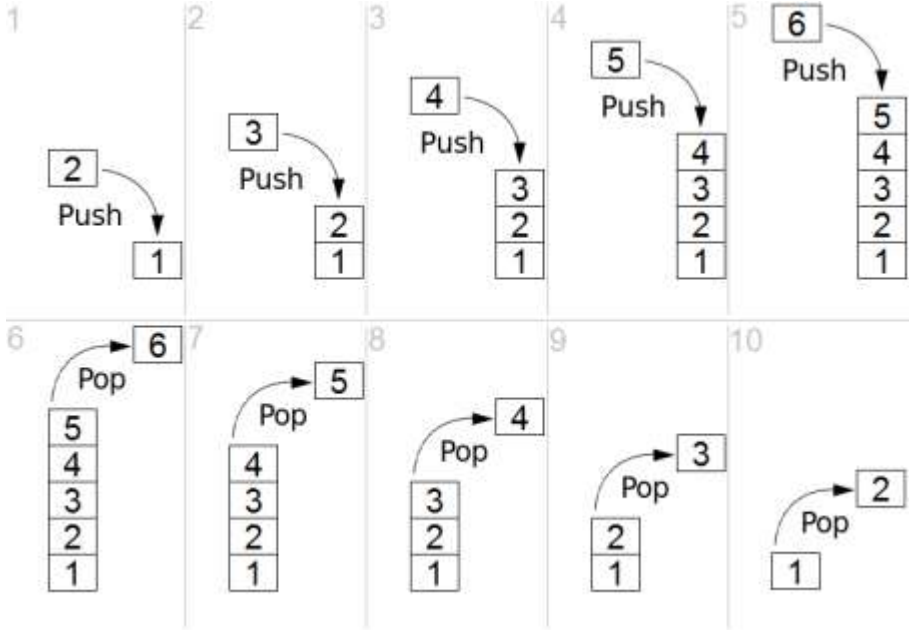
## 硬件堆栈（寄存器串联堆栈）

- CPU内部一组串联的寄存器
- 数据的传送在栈顶和通用寄存器之间进行
- 栈顶不动，数据移动，进出栈所有数据都需移动
- 栈容量有限

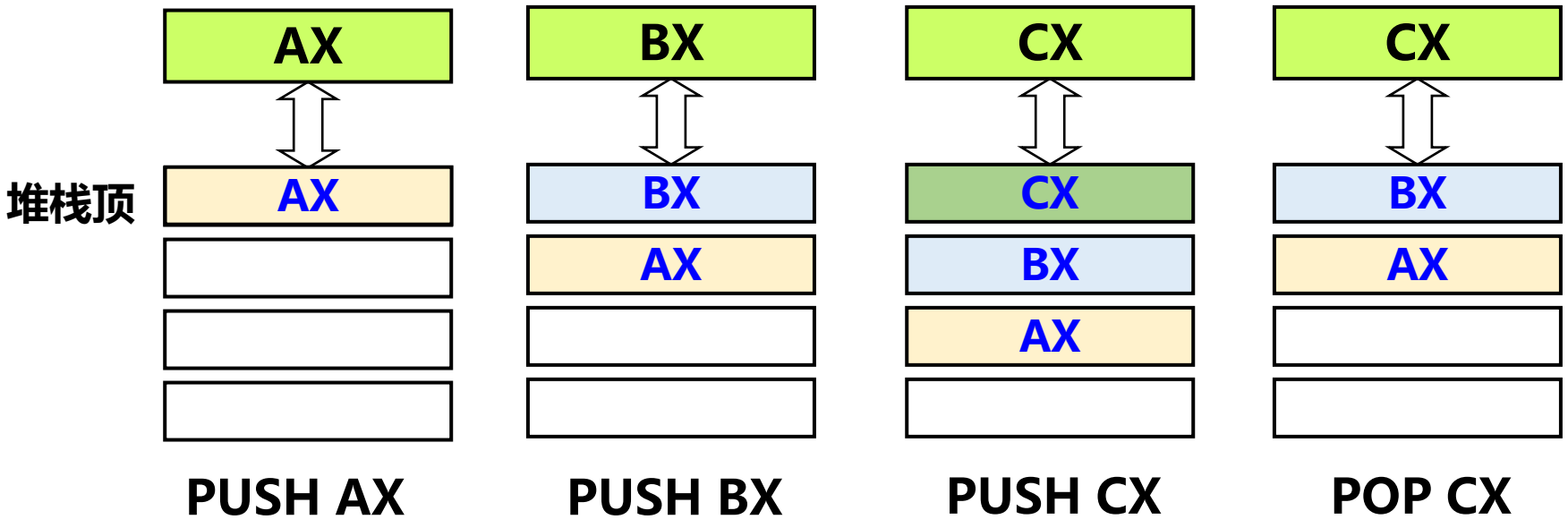



## 软件堆栈（内存堆栈）

- 内存区间做堆栈
- SP---堆栈指示器(栈指针),改变SP即可移动栈顶位置。
- 栈顶移动，数据不动，非破坏性读出
- 栈容量大，栈数目容量均可自定义



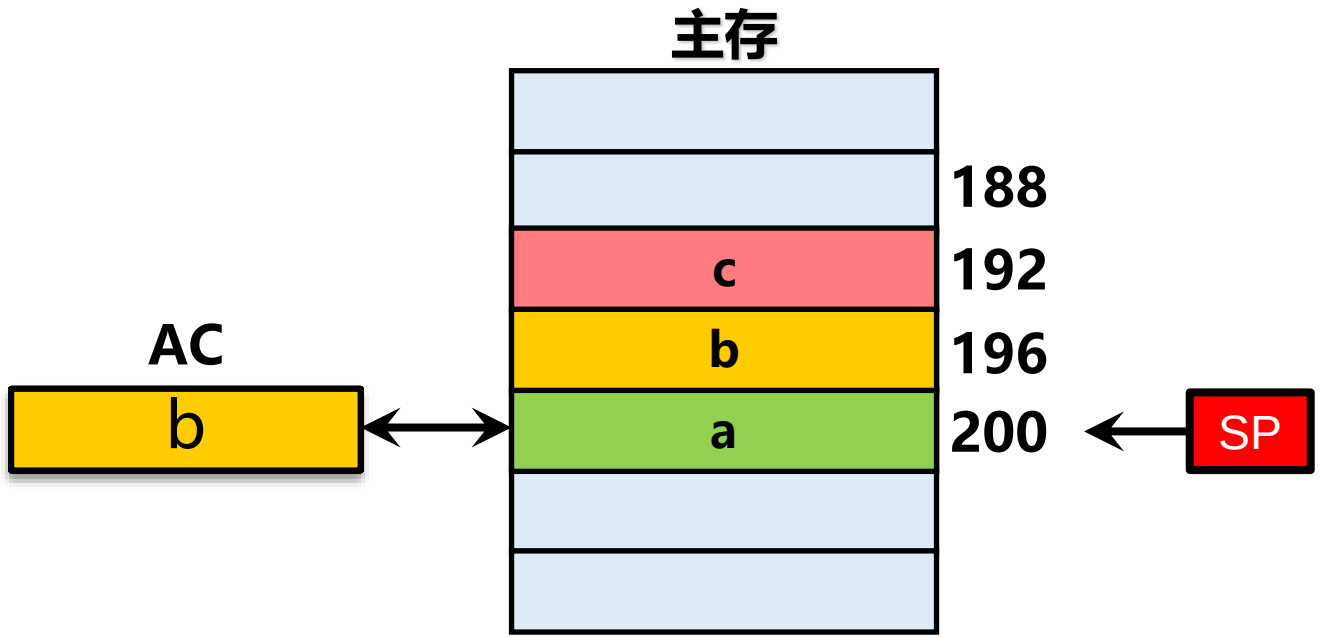
# 硬件堆栈




**栈顶不动，数据移动**

# 内存堆栈

- **进栈：** (AC) → Mem[sp--]　　**出栈：** Mem[++sp]→AC
- Push a 、Push b、Push c 、Pop 、Pop



# 不同寻址方式对比

**不同寻址方式的区别?**

| | 5bits | 3bits | 8bits | | |
|---|---|---|---|---|---|
| | 操作码 | 寻址模式 | 形式地址D | 实地址E | 寻址范围 |
| 立即寻址 | MOV | 000 | 38H | S=D | 0~255 -128~127 |
| 寄存器寻址 | MOV | 001 | 00 | E=R | 0~255# Reg |
| 直接寻址 | MOV | 010 | 200 | E=D | 0~255 RAM Cell |
| 间接寻址 | MOV | 011 | 200 | E=(D) | $0\sim2^{16}-1$ RAM Cell |
| 寄存器间接 | MOV | 100 | 01 | E=(R) | $0\sim2^{16}-1$ RAM Cell |
| 相对寻址 | JMP | 101 | 20 | E=(PC)+D | PC-128~PC+127 |
| 变址寻址 | MOV | 110 | 20 100 | E=(R)+D | $0\sim2^{16}-1$ RAM Cell |

# 寻址方式举例

设某机的指令字长16位，格式、有关寄存器和主存内容如下，X为寻址方式，D为形式地址，请在下表中填入有效地址E及操作数的值。？


PC=1000

$R_{基}$=2000

| | |
|---|---|
| 100 | 200 |
| | |
| 200 | 500 |
| | |
| 500 | 800 |
| | |
| 1100 | 100 |
| 1102 | 350 |
| | |
| 2100 | 200 |

| 寻址方式 | X | 有效地址E | 操作数 |
|---|---|---|---|
| 立即 | 0 | S=D | 100 |
| 直接 | 1 | E=D=100 | 200 |
| 间接 | 2 | E=(D)=200 | 500 |
| 相对 | 3 | E=(PC)+D=1100 | 100 |
| 变址 | 4 | E=(R)+D=2100 | 200 |
| 变址间址 | 5 | E=((R)+D)=200 | 500 |

# 本章主要内容

- 5.1 指令系统概述
- 5.2 指令格式
- 5.3 寻址方式
- **5.4 指令类型**
- 5.5 指令格式设计
- 5.6 CISC与RISC
- 5.7 指令系统举例（MIPS指令系统）

# 指令类型

- 算术逻辑运算指令
- 移位操作指令
- 数据传送指令
- 堆栈操作指令
- 字符串操作指令
- 程序控制指令
- 输入输出指令
- 其它指令

# 算术逻辑运算指令

- 进行各类数据信息处理，包括各种算术及逻辑运算指令
  - 与、或、非、异或等逻辑运算指令
  - 定点、浮点数的加、减、乘、除等算术运算指令

# 移位操作指令

- 包括算术移位、逻辑移位和循环移位指令，可以实现对操作数进行一位或多位的移位操作
  - 算术移位和逻辑移位指令分别控制实现带符号数和无符号数的移位
  - 循环移位按是否与进位位一起循环分为带进位循环(大循环)和不带进位循环(小循环)

# 数据传送指令

- 完成两个部件之间的数据传送操作,如寄存器与寄存器、 寄存器与存储器之间的数据传送。
  - MOV指令，并支持寄存器之间以及寄存器与存储器之间的数据传输
  - LOAD、STORE指令访存，其中LOAD为存储器读数指令，STORE为存储器写数指令

# 堆栈操作指令

- 一种特殊的数据传送指令，主要包括压栈或出栈两种
  - 压栈指令是把指定的操作数送入栈顶
  - 出栈指令是从栈顶弹出数据，并送到指令指定的目标地址中

# 字符串操作指令

- 非数值处理指令
- 字符串处理指令一般包括
  - 字符传送
  - 字符串比较
  - 字符串查找
  - 字符串抽取
  - 字符串转换等指令

# 程序控制指令

- 控制程序运行的顺序和选择程序的运行方向
  - 转移指令
  - 循环控制指令
  - 子程序调用与返回指令

# 输入输出指令

- I/O指令，用于实现主机与外部设备之间的信息传送
  - 主机可以向外部设备发出各种控制命令，从而控制外部设备的工作，也可以从外部设备端口寄存器中读取外部设备的各种工作状态等
  - 当外部设备与主存采用统一编址模式时，不需要设置专用的I/O指令，可以使用访存指令直接访问外部设备

# x86指令分类

- **数据传送类**
  - 取数　MOV AX，TEMP
  - 存数　MOV TEMP，AX
  - 传送　MOV AX，CX
- **算术运算类**
  - 定点+，-，×，÷　等
  - 浮点+，-，×，÷　求反，求补等
- **逻辑运算类**
  - NOT，AND，OR，XOR，TEST
- **程序控制类**
  - 无条件转移 JMP　条件转移 C，Z，N，P，V
  - 转子程序 JSR　子程序返回RET　中断返回IRET
- **输入/输出类**
  - IN AX，n　　OUT n, AX
- **其他类**
  - 标志操作：CLC（clear carry flag)
  - CLI (clear interrupt enable flag)
  - HLT，WAIT

# MIPS指令分类

- **运算指令**
  - 算术：add,addi,addu,addiu, sub,subu, mult,multu,div,divu, slt,slti,sltiu
  - 逻辑：and,andi,or, ori,xor,xori,nor
  - 移位指令：sll,sllv,srl, srlv,sra,srav
- **分支指令**
  - beq，bne，blez(<=0), bgez(>=0), bltz(<0)，bgtz(>0)，jal，j，jr
- **访存指令**
  - lw,lh,lb,sw,sh,sb
- **系统指令**
  - syscall，break，sync, cache

# 本章主要内容

- 5.1 指令系统概述
- 5.2 指令格式
- 5.3 寻址方式
- 5.4 指令类型
- 5.5 指令格式设计
- 5.6 CISC与RISC
- 5.7 指令系统举例

# 指令格式设计

- 指令系统
  - 考虑指令的完备性、规整性、有效性、兼容性和可扩展性
  - 考虑系统支持哪些指令、哪些数据类型和寻址方式
  - 最重要的是设计合理的指令格式
- 方便程序员进行程序设计，也有利于编译系统的设计，还有利于简化硬件实现，而且能够节省大量的程序存储空间
  - 确定指令的编码格式
  - 确定操作码字段和地址码字段的长度及它们的组合形式
  - 各种寻址方式的编码方法

# 指令格式设计

- 指令编码格式的设计
  - 定长指令格式
  - 变长指令格式
  - 混合编码指令格式
- 操作码的设计
  - 满足完备性
  - 确定操作码是采用定长结构还是采用变长结构
- 地址码的设计
- 寻址方式的设计
  - 寻址方式与操作码一起编码
  - 设置专门的寻址方式字段来指示对应的操作数采用的寻址方式

# 指令格式设计举例

- **例1.** 字长16位，主存64K，指令单字长单地址，80条指令。寻址方式有直接、间接、相对、变址。请设计指令格式。

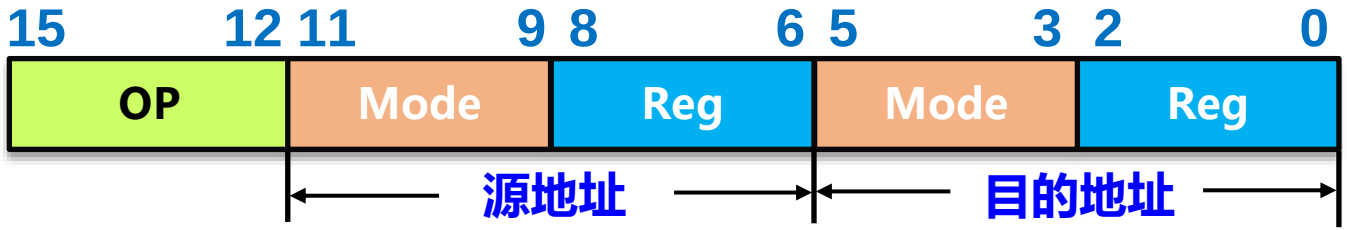
# 指令格式设计举例

- **例1.** 字长16位，主存64K，指令单字长单地址，80条指令。寻址方式有直接、间接、相对、变址。请设计指令格式。

- 80条指令 ⇒ OP字段需7位（$2^7$=128）
- 4种寻址方式 ⇒ 寻址方式位需2位
- 形式地址长度=16-7-2 =7位

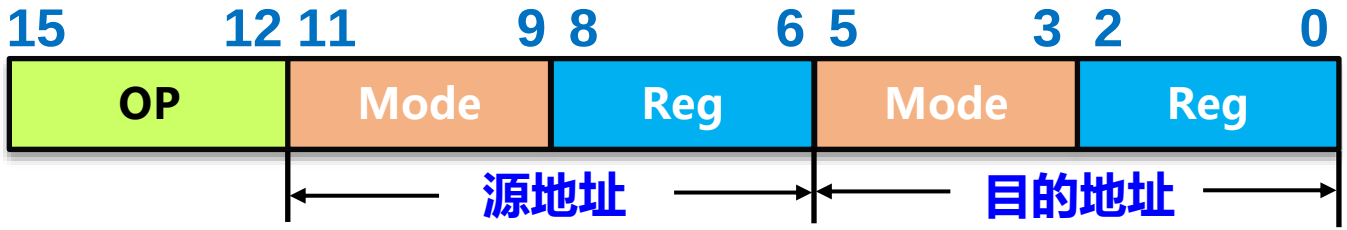
# 指令格式设计举例

- **例2.** 分析以下指令格式及寻址方式特点?



# 指令格式设计举例

**例2.** 分析以下指令格式及寻址方式特点?




- 1）二地址指令；
- 2）操作码可指定16条指令；
- 3）源和目的均有8种寻址方式；
- 4）源地址寄存器和目的地址寄存器均有8个；
- 5）可寻址范围为1~64K（与机器字长有关）

# 本章主要内容

- 5.1 指令系统概述
- 5.2 指令格式
- 5.3 寻址方式
- 5.4 指令类型
- 5.5 指令格式设计
- 5.6 CISC与RISC
- 5.7 指令系统举例

# 指令系统发展方向

- CISC---复杂指令系统计算机
  - Complex Instruction System Computer
  - 指令数量多，指令功能，复杂的计算机。
  - **Intel X86**
- RISC---精简指令系统计算机
  - Reduced  Instruction System Computer
  - 指令数量少，指令功能单一的计算机。
  - 1982年后的指令系统基本都是RISC
  - **MIPS、RISC-V**
- **CISC、 RISC互相融合**

# 复杂指令系统(CISC)

- 指令系统复杂庞大，指令数目一般多达二三百条。
- 寻址方式多。
- 指令格式多。
- 指令字长不固定。
- 对访存指令不加限制。
- 各种指令使用频率相差大。
- 各种指令执行时间相差大。
- 大多数采用微程序控制器。

# 精减指令系统(RISC)

- 指令条数少，只保留使用频率最高的简单指令，指令定长
  - 便于硬件实现，用软件实现复杂指令功能
- Load/Store架构
  - 只有存/取数指令才能访问存储器，其余指令的操作都在寄存器之间进行
  - 便于硬件实现
- 指令长度固定，指令格式简单、寻址方式简单
  - 便于硬件实现
- CPU设置大量寄存器（32~192）
  - 便于编译器实现
- 一个机器周期完成一条机器指令
- RISC CPU采用硬布线控制，CISC采用微程序

# 本章主要内容

- 5.1 指令系统概述
- 5.2 指令格式
- 5.3 寻址方式
- 5.4 指令类型
- 5.5 指令格式设计
- 5.6 CISC与RISC
- 5.7 指令系统举例

# PDP-11指令格式

- 1957年**DEC**公司成立，生产16位微型计算机
- 1970年PDP-11诞生
  - 70~80年代红极一时，后被苹果II，IBM-PC超越
- 1984年VAX8600 扳回一局
- 1998年被Compaq 96亿美金收购，2002并入惠普
- 机器字长16位
- 单字长，双字长，三字长指令


# PDP-11指令集特点


- 双地址指令和
  - 双地址指令操作码字段为4位
  - 地址码部分包括两个操作数，每个操作数各6位
  - 8种寻址方式
  - 寄存器字段为3位，可寻址8个寄存器单元，其中R0~R5为通用寄存器，R6为栈指针SP寄存器，R7为程序计数器 PC。
- 单地址指令
  - 单地址指令采用扩展操作码形式，操作码部分向地址码部分扩展6位，位[illegible]分为6位。

# PDP-11寻址方式

| mode | 寻址方式 | 汇编语法 | 功能 |
|---|---|---|---|
| 0 | 寄存器 | $R_i$ | 寄存器值就是操作数 |
| 1 | 寄存器**间接** | $(R_i)$ | 寄存器的值是操作数地址 |
| 2 | 自增寻址 | $(R_i)+$ | 寄存器的值是操作数地址，取数后寄存器自增 (byte +1,word+2) |
| 3 | 自增**间接** | $@(R_i)+$ | 寄存器的值是操作数地址的地址，取数后寄存器加2 |
| 4 | 自减寻址 | $-(R_i)$ | 先将寄存器自减，运算结果是操作数地址 (byte -1,word -2) |
| 5 | 自减**间接** | $@-(R_i)$ | 先将寄存器减2，运算结果是操作数地址的地址 |
| 6 | 变址寻址 | $index(R_i)$ | 操作数地址=寄存器的值+16位index |
| 7 | 变址**间址** | $@index(R_i)$ | 操作数地址的地址=寄存器的值+16位index |

**C语言风格，适合堆栈指令**

# 扩展寻址方式

| mode | 寻址方式 | 汇编语法 | 例子 | 功能 |
| --- | --- | --- | --- | --- |
| 2 | 立即数寻址 | #n | `Add #10,R0` | 操作数是指令字后续一个机器字 |
| 3 | 直接寻址 | @#n | `CLR @#1100` | 操作数地址是指令字后续一个机器字 |
| 6 | 相对寻址 | A / A(PC) | `Inc 10 / Inc 10(PC)` | 操作数地址=PC+A　A是指令字后续一个机器字 |
| 7 | 相对间接 | @A　@A(PC) | `CLR @10` | 操作数地址的地址=PC+A　**PC是下条指令地址** |

# MIPS指令概述

- **MIPS** (Microprocessor without Interlocked Pipeline Stages)
  - 1981年斯坦福大学Hennessy教授研究小组研制并商用
  - RISC架构
  - 易于流水线CPU设计
  - 易于编译器开发
  - 寻址方式，指令操作非常简单
  - MIPS I，MIPS II，MIPS III，MIPS IV，MIPS V，**MIPS32**，和MIPS64多个版本


- 广泛用于嵌入式系统，在PC机、服务器中也有应用
- 更适合于教学，相比X86更加简洁雅致，不会陷入繁琐的细节

# 32个MIPS寄存器

| 寄存器# | 助记符 | 释义 |
| --- | --- | --- |
| 0 | $zero | 固定值为0　硬件置位 |
| 1 | $at | 汇编器保留，临时变量 |
| 2~3 | $v0~$v1 | 函数调用返回值 |
| 4~7 | $a0~$a3 | 4个函数调用参数 |
| 8~15 | $t0~$t7 | 暂存寄存器，被调用者按需保存 |
| 16~23 | $s0~$s7 | save寄存器，调用者按需保存 |
| 24~25 | $t8~$t9 | 暂存寄存器，同上 |
| 26~27 | $k0~$k1 | 操作系统保留，中断异常处理 |
| 28 | $gp | 全局指针 (Global Pointer) |
| 29 | $sp | 堆栈指针 (Stack Pointer) |
| 30 | $fp | 帧指针　(Frame Pointer) |
| 31 | $ra | 函数返回地址（Return Address) |

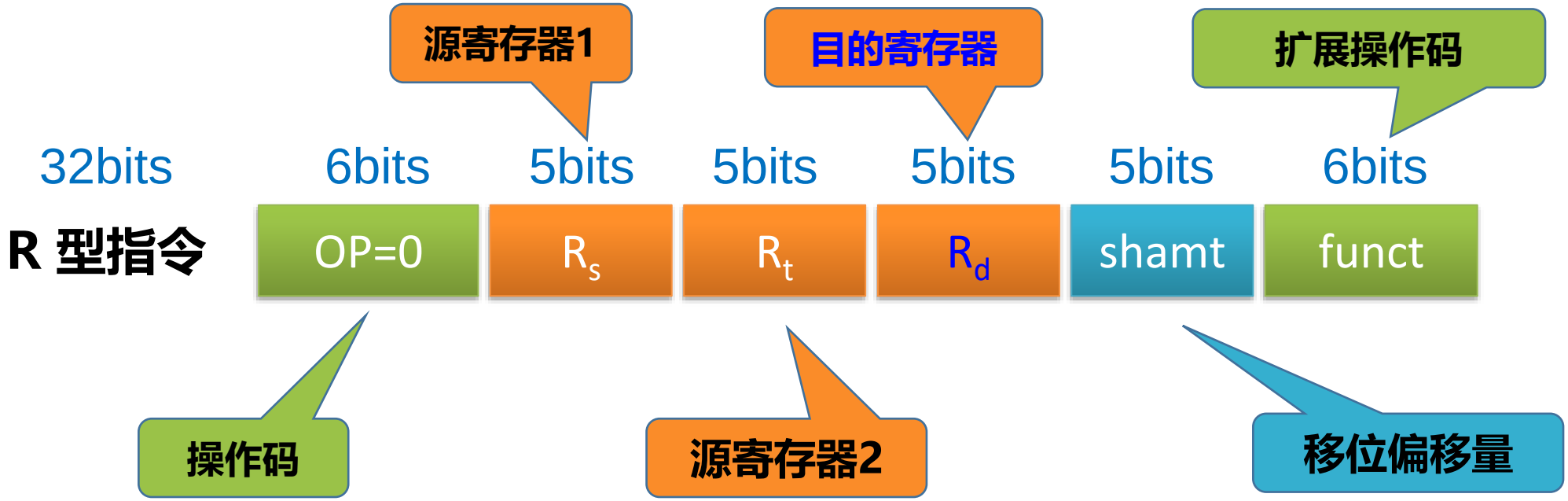
- **32个32位通用寄存器 $0~$31**
- **32个32位单精度浮点寄存器$f_0$-$f_{31}$**

# MIPS指令格式


- OP字段固定为6位,funct字段就是扩展操作码、 shamt字段是移位变量，用于移位指令、 Imm字段为16位立即数字段. rs、rt、rd为寄存器操作数字段，用5位表示，可以访问32个通用寄存器，R型指令最多可以有3个寄存器操作数，I型指令最多可以有2个寄存器操作数

# MIPS 32指令格式（R型指令）

- R型指令的操作数只能来自寄存器，运算结果也只能存入寄存器中



# MIPS指令格式 (R型指令)

| | | 6bits | 5bits | 5bits | 5bits | 5bits | 6bits |
|---|---|---|---|---|---|---|---|
| 指令 | 格式 | OP | rs | rt | rd | shamt | funct |
| add | R | 0 | Reg | Reg | Reg | 0 | $32_{10}$ |
| sub | R | 0 | Reg | Reg | Reg | 0 | $34_{10}$ |
| and | R | 0 | Reg | Reg | Reg | 0 | $36_{10}$ |
| or | R | 0 | Reg | Reg | Reg | 0 | $37_{10}$ |
| nor | R | 0 | Reg | Reg | Reg | 0 | $39_{10}$ |
| sll | R | 0 | 0 | Reg | Reg | x | $0_{10}$ |
| srl | R | 0 | 0 | Reg | Reg | x | $2_{10}$ |
| jr | R | 0 | Reg | | | 0 | $8_{10}$ |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **add** | **R** | **0** | 18 | 19 | 17 | **0** | **32** |

- `add $s1,$s2,$s3` # 机器码 0x2538820

# MIPS指令格式 (I型指令)

| | | 6bits | 5bits | 5bits | 5bits | 5bits | 6bits |
|---|---|---|---|---|---|---|---|
| 指令 | 格式 | OP | rs | rt | rd | shamt | funct |
| add | R | 0 | Reg | Reg | Reg | 0 | $32_{10}$ |
| addi | I | 8 | Reg | Reg | 16bits 立即数 | | |
| lw | I | 35 | Reg | Reg | 16bits 立即数 | | |
| sw | I | 43 | Reg | Reg | 16bits 立即数 | | |
| andi | I | 12 | Reg | Reg | 16bits 立即数 | | |
| ori | I | 13 | Reg | Reg | 16bits 立即数 | | |
| beq | I | 4 | Reg | Reg | 16bits 立即数（相对寻址） | | |
| bne | I | 5 | Reg | Reg | 16bits 立即数（相对寻址） | | |
| **j** | **J** | **2** | 26bits 立即数(伪直接寻址) | | | | |
| **jal** | **J** | **3** | 26bits 立即数(伪直接寻址) | | | | |

# MIPS指令格式 (J型指令)

| OP | 指令助记符 | 指令功能描述 | 备注 |
| --- | --- | --- | --- |
| 02 | j address | PC ← $\{(PC+4)_{31:28}$,address,00$\}$ | 无条件分支 |
| 03 | jal address | R[31] ← PC+8（无延迟槽 +4）<br>PC ← $\{(PC+4)_{31:28}$,address,00$\}$ | 子程序调用指令 |

# MIPS寻址方式总结

- 寄存器寻址
- 变址寻址
- 立即数寻址
- PC相对寻址 `beq reg1,reg2,offset`
  - PC + 4 + 16位偏移地址左移两位
  - 字地址变字节地址
- 伪直接寻址 `J label`
- R型指令的寻址方式只有寄存器寻址，I型指令的寻址方式有寄存器寻址、立即数寻址、基址寻址(偏移寻址)、相对寻址，J型指令只有伪直接寻址

# MIPS X86 差异

| # | X86 | MIPS |
|---|---|---|
| 1 | 变长（1-15bytes） | 定长指令 |
| 2 | 指令数多 CISC | 指令数少 RISC |
| 3 | 8个通用寄存器 | 32个通用寄存器 |
| 4 | 寻址方式复杂 | 寻址方式简单 |
| 5 | 有标志寄存器 | 无标志寄存器 |
| 6 | 最多两地址指令 | 三地址指令 |
| 7 | 无限制 | 只有Load/store能访问存储器 |
| 8 | 有堆栈指令 push，pop | 无堆栈指令（访存指令代替） |
| 9 | 有I/O指令 | 无I/O指令(设备统一编址) |
| 10 | 参数传递：栈帧 | 参数传递（4寄存器+栈帧） |

# MIPS 32 & ARMv8-32

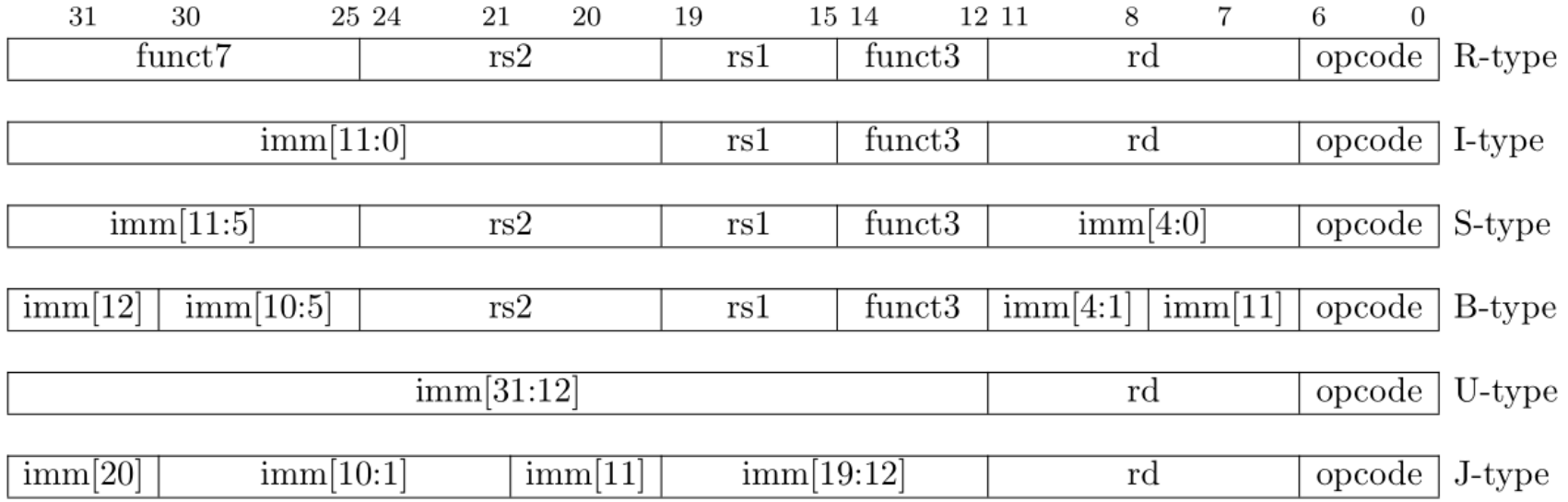
- 相同之处
  - 32位定长指令
  - 32个通用寄存器，一个恒零寄存器
  - load/store架构
  - 都不能并行存取多个寄存器（方便批量保存寄存器，恢复寄存器，硬件实现更复杂）
  - 都有分支指令，能根据寄存器的值为0转移或不为零转移
- 区别
  - 条件分支指令，arm依赖于条件码，mips无状态标志寄存器
  - ARMv7指令还有**条件执行指令**，不满足条件不执行
  - ARMv8指令集规模更大，寻址方式更多

# RISC-V

- **完全开放**的 ISA
- **大道至简，简单就是美**
  - 包含一个最小的**核心冻结的ISA** （可支撑OS，方便教学）
  - 适合硬件实现，而不仅仅是适用于模拟或者二进制翻译
- **无病一生轻的后发优势**
  - 模块化的可扩展指令集
  - 方便简化硬件实现，提升性能
    - 更规整的指令编码、更简洁的运算指令、更简洁的访存模式：Load/Store架构
    - 高效分支跳转指令（减少指令数目）、简洁的子程序调用
    - 无条件码执行、无分支延迟槽

# MIPS 32 & RISC-V

- 指令助记符及语法格式大同小异
- RISC-V 分支预测，MIPS延迟槽
- RISC-V支持变长指令扩展
- RISC-V 将源寄存器rs1，rs2和目标寄存器（rd）固定在同样位置，以简化指令译码
- 立即数分散在不同位置，但符号位固定在第31位，可加速符号扩展电路，与译码并行



# 习题

5.8 计算机的指令格式包括操作码OP、寻址方式特征位I和形式地址D等3个字段，其中OP字段为6位，寻址方式特征位字段I为2位，形式地址字段D为8位。I的取值与寻址方式的对应关系如下。
I=00:直接寻址。
I=01:用变址寄存器X1进行变址。
I=10:用变址寄存器X2进行变址。
I=11:相对寻址。

设(PC)=1234H，(X1)=0037H，(X2)=1122H，以下4条指令均采用上述格式，请确定这些指令的有效地址。

(1)4420H; (2)2244H; (3)1322H; (4)3521H。

# 习题

5.8 计算机的指令格式包括操作码OP、寻址方式特征位I和形式地址D等3个字段，其中OP字段为6位，寻址方式特征位字段I为2位，形式地址字段D为8位。I的取值与寻址方式的对应关系如下。
I=00:直接寻址。
I=01:用变址寄存器X1进行变址。
I=10:用变址寄存器X2进行变址。
I=11:相对寻址。

设(PC)=1234H，(X1)=0037H，(X2)=1122H，以下4条指令均采用上述格式，请确定这些指令的有效地址。

(1)4420H; (2)2244H; (3)1322H; (4)3521H。

解:
(1) 4420H=(010001 00 0010 0000)2，直接寻址，EA=D=20H。
(2) 2244H=(001000 10 0100 0100)2，X2变址寻址，EA=(X2)+D=1122+44=1166H。
(3) 1322H=(000100 11 0010 0010)2，相对寻址，EA=(PC)+2+D=1234+2+22=1258H。
(4) 3521H=(001101 01 0010 0001)2，X1变址寻址.EA=(X1)+D=0037+21=58H。