控制器概述
指令与指令系统

2022年秋

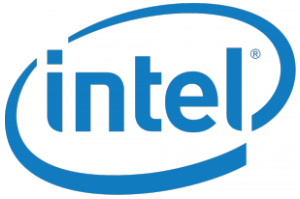
主要内容和教学安排

- 第一讲 指令系统概述，指令格式，寻址方式
- 第二讲 RISC-V指令功能及实现
- （总线及存储访问实验）
- 第三讲 指令格式，数据通路
- 第四讲 单周期**CPU**设计
- 第五讲 多周期**CPU**设计
- 第六讲 指令流水基本概念
- 第七讲 流水中的结构冲突、数据冲突
- 第八讲 控制冲突、中断的解决方案
- （存储器单元）
- 第九讲 **THINPAD**介绍大实验
- **第十讲大实验辅导及检查(1)**
- **第十一讲大实验辅导及检查(2)**
- **第十二讲大实验辅导及检查(3)**

重点和难点

- 单条指令功能的实现
  - 如何设计指令的数据通路？
  - 如何划分指令的执行步骤?
  - 如何根据指令得到控制信号？
- 机器语言程序的自动执行
  - 指令之间如何衔接？
- 提高性能
  - 在不增加太多硬件的情况下如何提高性能？
- 实现途径
  - 控制信号生成：组合逻辑或微程序
  - 程序自动执行：PC 、节拍和下地址
  - 指令系统：RISC和CISC
  - 提高性能：指令流水

主要的指令集体系结构



x86

Designer	Intel, AMD
Bits	16-bit, 32-bit and 64-bit
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
Design	CISC
Type	Register-memory
Encoding	Variable (1 to 15 bytes)
Endianness	Little

笔记本电脑，台式机，服务器
(Core i3, i5, i7, M)
x86 Instruction Set

ARM architectures

Designer	ARM Holdings
Bits	32-bit, 64-bit
Introduced	1985;
Design	RISC
Type	Register-Register
Encoding	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions. ARMv7 user-space compatibility[1]
Endianness	Bi (little as default)

智能手机
(iPhone, iPad, Raspberry Pi)
ARM Instruction Set

MIPS

Designer	MIPS Technologies, Inc.
Bits	64-bit (32→64)
Introduced	1981;
Design	RISC
Type	Register-Register
Encoding	Fixed
Endianness	Bi

Digital home & networking equipment
(Blu-ray, PlayStation 2)
MIPS Instruction Set

RISC-V指令集体系结构

- RISC-V是一个基于精简指令集（RISC）原则的开源指令集架构（ISA）。该项目2010年始于Berkeley大学，许多贡献者是该大学以外的志愿者和工业界工作者
- 其设计使其适用于各种现代计算设备（云计算机中的服务器、台式机、智能手机和嵌入式系统等）
- RISC-V 指令使用模块化的设计, 包括几个可以互相替换的基本指令集, 以及额外可以选择的扩展指令集。所有基本跟扩展的指令集都是由科技产业，研究机构跟学术界合作开发的。基本指令集规范了指令跟他们的编码，控制流程，寄存器数目（以及它们的长度），存储器跟定址方式，逻辑（整数）运算以及其他。只要有软件以及一个通用的编译器的支持，只用基本指令集就可以用来制作一个简单的通用型的电脑。
- 标准的扩展指令集可以搭配所有的基本指令集，以及其他扩展指令集，而不会冲突。

RISC-V模块化设计（基本指令集）

指令集名称	描述	版本	状态
基本指令集			
RV32I	基本整数指令集, 32位	2.0	冻结
RV32E	基本整数指令集 (嵌入式系统), 32位, 16 寄存器	1.9	开放
RV64I	基本整数指令集, 64位	2.0	冻结
RV128I	基本整数指令集, 128位	1.7	开放

RISC-V模块化设计（扩展指令集）

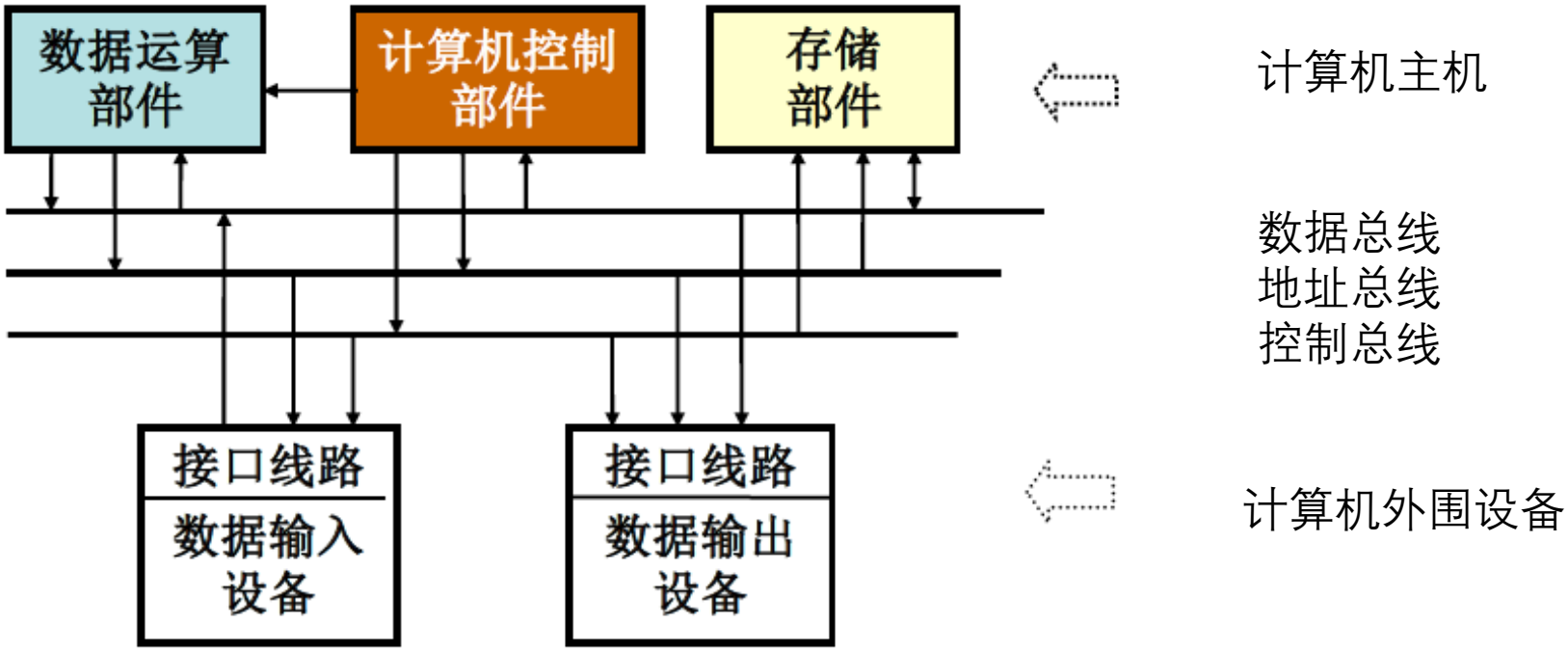
标准扩展指令集			
M	整数乘除法标准扩展	2.0	冻结
A	不可中断指令(Atomic)标准扩展	2.0	冻结
F	单精确度浮点运算标准扩展	2.0	冻结
D	双倍精确度浮点运算标准扩展	2.0	冻结
G	所有以上的扩展指令集以及基本指令集的总和的简称	不适用	不适用
Q	四倍精确度浮点运算标准扩展	2.0	冻结
L	十进制浮点运算标准扩展	0.0	开放
C	压缩指令标准扩展	2.0	冻结
B	位操作标准扩展	0.36	开放
J	动态指令翻译标准扩展	0.0	开放
T	顺序存储器访问标准扩展	0.0	开放
P	单指令多资料流（SIMD）运算标准扩展	0.1	开放
V	向量运算标准扩展	0.2	开放
N	用户中断标准扩展	1.1	开放

复杂指令集（CISC）/精简指令集（RISC）

- 早期的趋势：为了能够做复杂的操作，增加越来越多的指令
  - 导致学习和理解上的困难
  - 导致硬件复杂（可能会变慢）
- 改变设计思路：Reduced Instruction Set Computing (RISC)
  - 使用更简单，更小的指令集，会更加容易获得快速的硬件
  - 如果需要做复杂操作的话，让软件来做，组合几条简单的指令来完成工作

计算机硬件系统功能部件

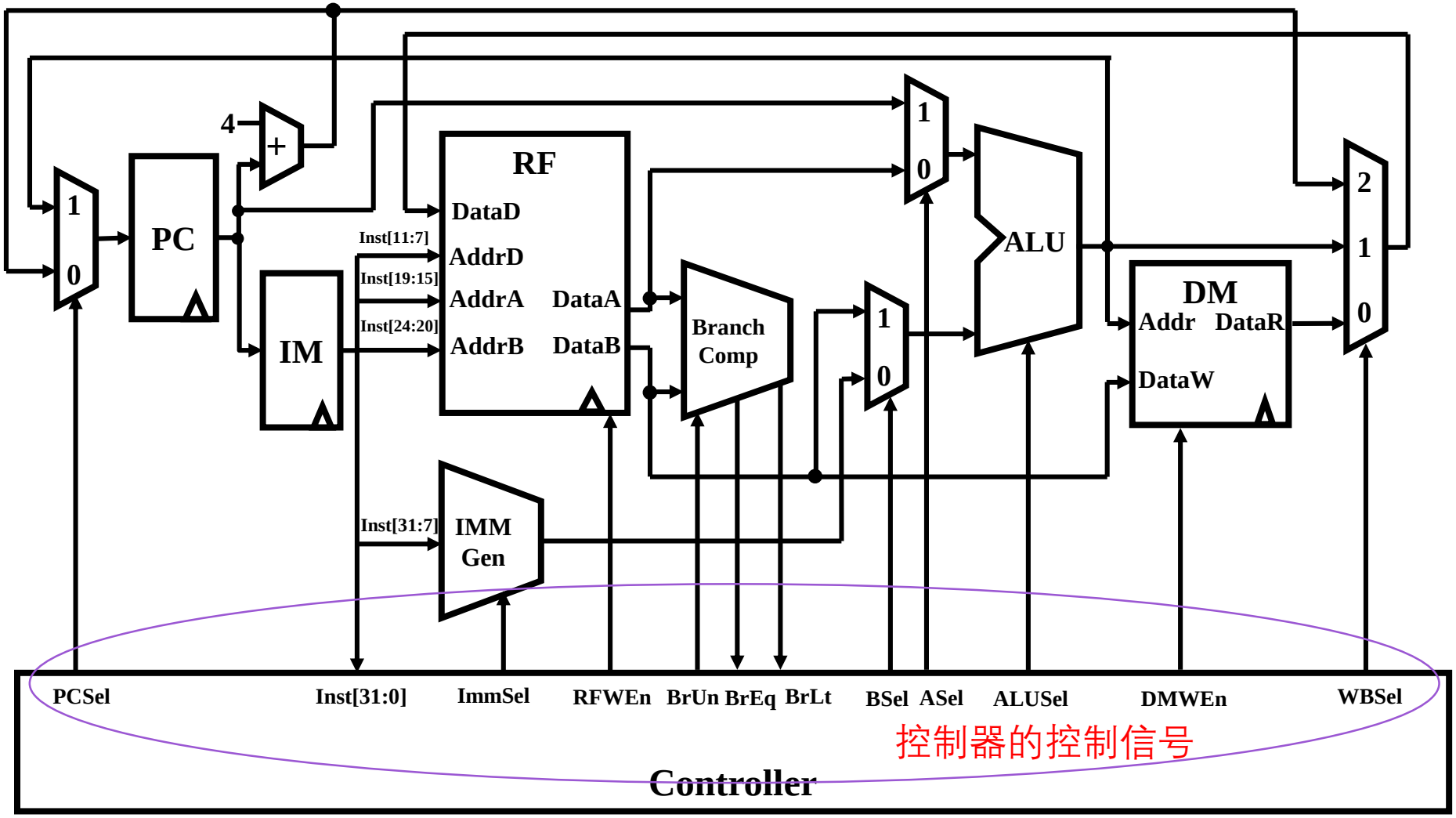
计算机的CPU



控制器的作用

- 计算机的基本功能
  - 执行程序
- 程序的构成
  - 指令序列
- 控制器的作用
  - 根据指令的要求，提供给各部件相应的控制信号，指挥、协调各部件共同完成指令规定的功能
  - 自动执行下一条指令
- 指令执行
  - 取指令
  - 分析指令
  - 执行指令

单周期CPU


4
+
1
0
PC
IM
RF
DataD
AddrD
AddrA
DataA
AddrB
DataB
Inst[11:7]
Inst[19:15]
Inst[24:20]
Inst[31:7]
IMM
Gen
Branch
Comp
1
0
1
0
ALU
DM
Addr
DataR
DataW
2
1
0
PCSel
Inst[31:0]
ImmSel
RFWEn
BrUn
BrEq
BrLt
BSel
ASel
ALUSel
DMWEn
WBSel
控制器的控制信号
Controller

指令与指令系统

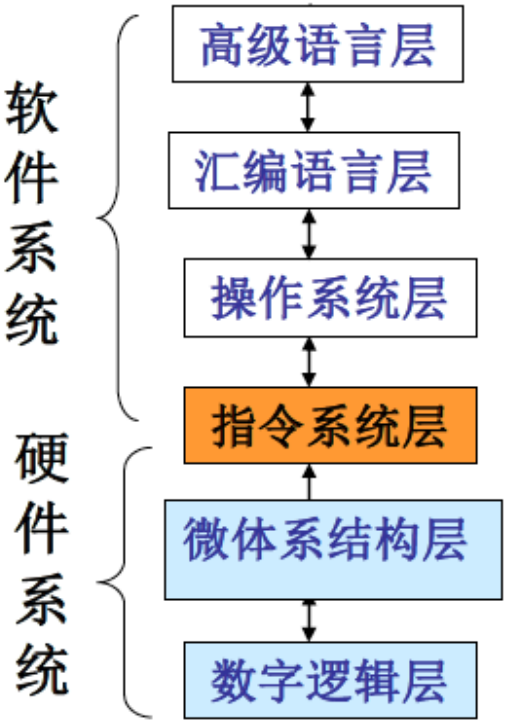
- 指令与指令系统的概念
- 指令系统设计要求
- 指令功能和分类
- 指令格式
  - 变长指令字/定长指令字
  - 操作码扩展
- 寻址方式

指令与指令系统

- 计算机系统由硬件和软件两大部分组成。硬件指由中央处理器、存储器以及外围设备等组成的实际装置。软件是为了使用计算机而编写的各种系统的和用户的程序，程序由一个序列的计算机指令组成。
- 指令是计算机运行的最小的功能单元，是指挥计算机硬件运行的命令，是由多个二进制位组成的位串，是计算机硬件可以直接识别和执行的一个信息体。
- 一台计算机提供的全部指令构成该计算机的指令系统。指令用于程序设计人员告知计算机执行一个最基本运算、处理功能，多条指令可以组成一个程序，完成一项预期的任务。

指令系统的地位

- 可以从6 个层次分析和看待计算机系统的基本组成。
- 指令系统层处在硬件系统和软件系统之间，是硬、软件之间的接口部分，对两部分都有重要影响。
- 硬件系统用于实现每条指令的功能，解决指令之间的衔接关系；
- 软件由按一定规则组织起来的许多条指令组成，完成一定的数据运算或者事务处理功能。
- 指令系统优劣是一个计算机系统是否成功的关键因素。



计算机系统的层次结构

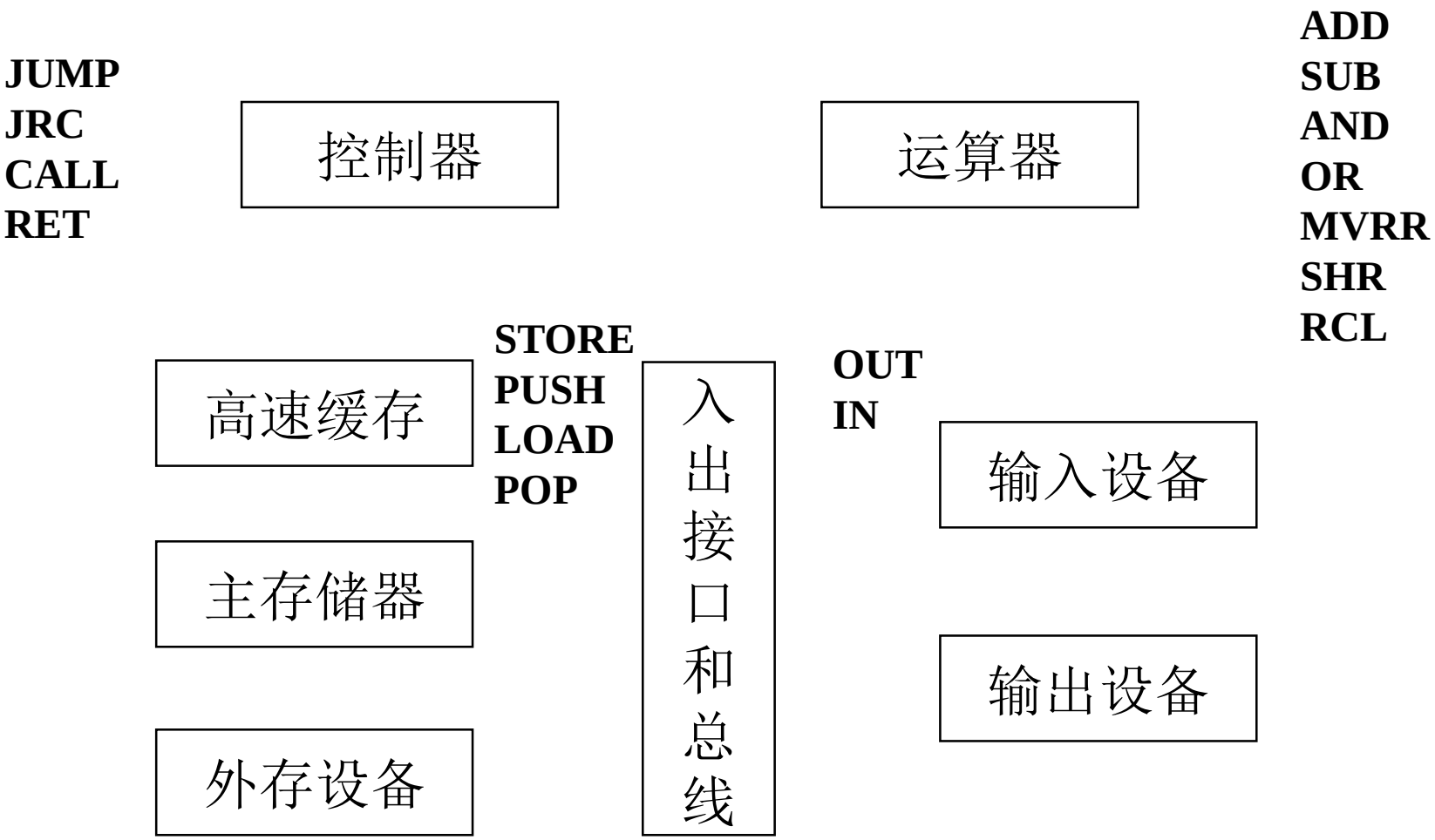
指令系统设计要求

- 完备性
  - 指令功能齐全、编程方便
- 规整性
  - 指令格式简单、统一
- 高效性
  - 占内存少，运行高效
- 兼容性
  - 同一系列软件兼容

计算机中需配备的指令

- 指令是用户使用计算机和计算机本身运行的最小的功能单元：①指令是由多个二进制位组成的数串，②用于设计程序，③计算机硬件可直接识别和执行。通常情况下一台计算机需要提供哪些指令呢？
- 计算机用于计算和处理数据，为此，要在计算机硬件系统中设置5 种类型的部件：运算器部件、控制器部件、存储器部件、输入设备、输出设备，各自承担数据运算、系统指挥控制、保存当前程序和数据、执行输入和执行输出的功能。需要在计算机中设置为使用和控制这几个部件运行的相应指令。

使用硬件系统的基本指令



指令的功能和分类

- 指令用于设计程序，指令系统构成最低级别的程序设计语言，程序设计人员通过指令直接指挥计算机的硬件完成某一个基本的运算、处理功能，例如：
  - 对数值数据的算术运算，对逻辑数据的逻辑运算，
  - 在计算机部件之间传送、保存数据，
  - 从外部向计算机内输入数据，
  - 把计算机内部计算结果输出出来，
  - 按照某种条件控制计算机选择执行某段程序，
  - 当然还有另外一些方面的更深层次的要求等；
  - 可以按照指令执行的功能对它们进行分类。

指令的功能和分类

- 算术与逻辑运算指令
  - 加、减、乘、除、变符号等算术运算
  - 与、或、非、异或等逻辑运算
- 移位操作指令
  - 算术移位（一般只右移）、逻辑移位、循环移位
- 数据传送指令
  - 通用寄存器之间传送
  - 通用寄存器与主存储器存储单元之间传送
  - 主存储器不同存储单元之间传送
- 输入输出指令
  - 通用寄存器与输入输出设备（接口）之间传送

指令的功能和分类

- 转移指令
  - 变动程序中指令执行次序的指令，分为无条件转移指令和条件转移指令
- 子程序调用与返回指令
  - 调用指令与返回指令二者要配合使用，子程序的最后一条指令一定是返回指令，执行结束后返回主程序断点
- 堆栈操作指令
  - 堆栈（**stack**）是由若干个连续存储单元组成的先进后出的存储区，有压入（即进栈）和弹出（即退栈）操作
- 其他指令
  - 置条件码指令、开中断指令、关中断指令
  - 停机指令、空操作指令、特权指令

指令表示

- 指令中的内容，包括指令操作码（指出指令完成的运算处理功能和数据类型）和操作数或指令的地址（指明用到的数据或地址）两部分。例如：
  - 算逻运算中的运算功能，数据来源或结果去向
  - 数据传送指令中的数据原来位置和新的存储位置
  - 输入输出指令中用到的设备和数据来、去的位置
  - 转移指令的转移类别、转移条件和转移地址等
- 每一条指令必须指明它需要完成的功能，通常用几位指令操作码表示；还需要指明用到的数据、地址或设备，通常在地址字段给出，可能是：
  - **(1)**寄存器编号，**(2)**设备端口地址，
  - **(3)**存储器的单元地址**(4)** 数值等几种信息。

指令格式与指令字长

- 指令字长是指组成一条指令的二进制数的位数，例如8 bits、16 bits、32 bits、64 bits等，指令格式与指令字长密切相关，指令字越长可以给出的信息越多。一个指令字通常由指令操作码和操作数地址两部分组成，如何把一个指令字划分成多个字段并分配各字段所表示的内容大有学问。

R 型	funct7	rs2	rs1	funct3	rd	opcode
I 型	imm[11:0]		rs1	funct3	rd	opcode
S 型	Imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB / B 型	Imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
UJ / J 型	Imm[20,10:1,11,19:12]				rd	opcode
U 型	Imm[31:12]				rd	opcode

RISC-V 指令格式 (32位)

<table>
<tr><td rowspan="2">op</td><td>dr</td><td>sr</td></tr>
<tr><td colspan="2">io port/offset</td></tr>
<tr><td colspan="3">Immediate/address/offset</td></tr>
</table>

TEC-2000 指令格式 (16位)

指令格式

- 指令字：完整的一条指令的二进制表示
- 指令字长：指令字中二进制代码的位数
  - 机器字长：计算机能直接处理的二进制数据的位数
  - 指令字长（字节倍数）＝**0.5**、**1**、**2**…个机器字长
  - 定长指令字结构**vs.** 变长指令字结构
- 指令格式：指令字中操作码和操作数地址的二进制位的分配方案

  - 操作码：指明本条指令的操作功能，
  - 每条指令有一个确定的操作码
  - 操作数地址：说明操作数存放的地址，有时是操作数本身

操作码组织与编码

- 定长的操作码的组织方案
  - 在指令字最高位部分分配固定若干位用于表示操作码，有利于简化计算机硬件设计，提高指令译码和识别速度
  - 例如：**IBM360**机、**THINPAD**教学机
- 变长的操作码的组织方案
  - 在指令字最高位部分用一固定长度的字段来表示基本操作码，而对于部分操作数地址位数可以少的指令，则把另外多位辅助操作码扩充到该操作数地址字段，即操作码位数可变。
  - 这种方法在不增加指令字长的情况下，可表示更多的指令，但增加了译码和分析难度，要求更多的硬件支持
  - 例如：**PDP-11**计算机、**TEC-2000**的**8**位机

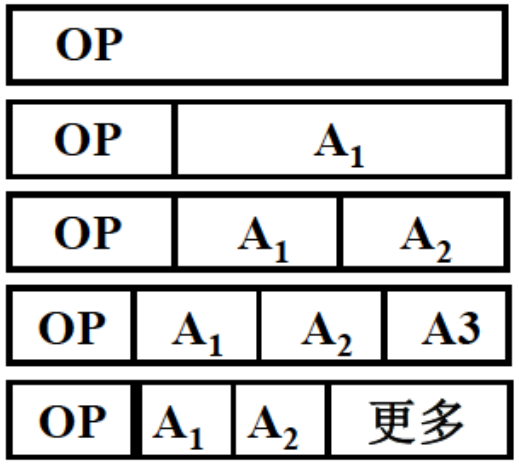
操作码组织与编码

- 操作码字段与操作数地址字段有所交叉的方案
- 不同指令的操作码长度可以不同，表示操作码所用到的一些二进制位不再集中在指令字的最高位部分，而是与用于表示操作数地址的一些字段有所交叉，操作码还被区分为主操作码和辅助操作码这样不同的两部分，这是一种比较特殊、不很常用的方案。
- 例如：**NOVA (DJS-130)** 计算机就采用这种方案

指令操作码的位数限制指令系统中的指令条数!

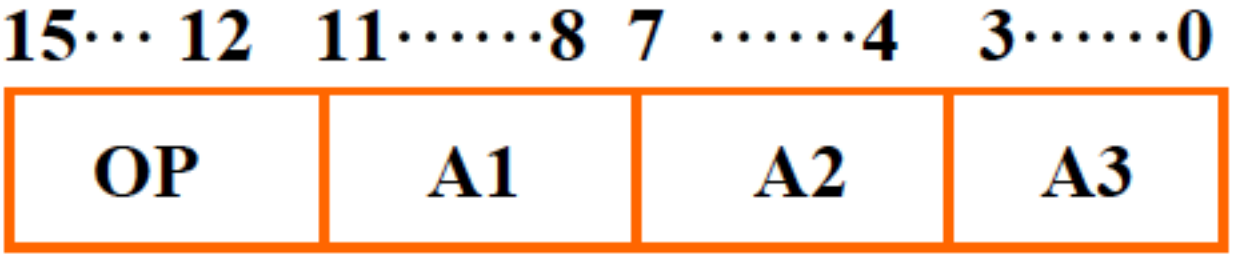
操作数个数与来源

- 指令操作数个数
  - 无操作数指令（零地址指令）
  - 单操作数指令（一地址指令）
  - 双操作数指令（二地址指令）
  - 三操作数指令（三地址指令）
  - 多操作数指令（多地址指令）
- 指令操作数来源和去向
  - CPU内部的通用寄存器
  - 输入输出设备（接口）的一个寄存器
  - 主存储器的一个存储单元



指令操作码的扩展技术

- 假设某机器的指令长度为16位，包括4位基本操作码和三个4位地址码段。



- 4 位基本操作码可表示16个状态，
- 如用4 位操作码，则能表示16 条三地址指令，
- 若用8 位操作码，则可表示256 条二地址指令，
- 而用12位操作码，则可表示4096条一地址指令，
- 若16位全用作操作码，则可表示65536条零地址指令

指令操作码的扩展技术

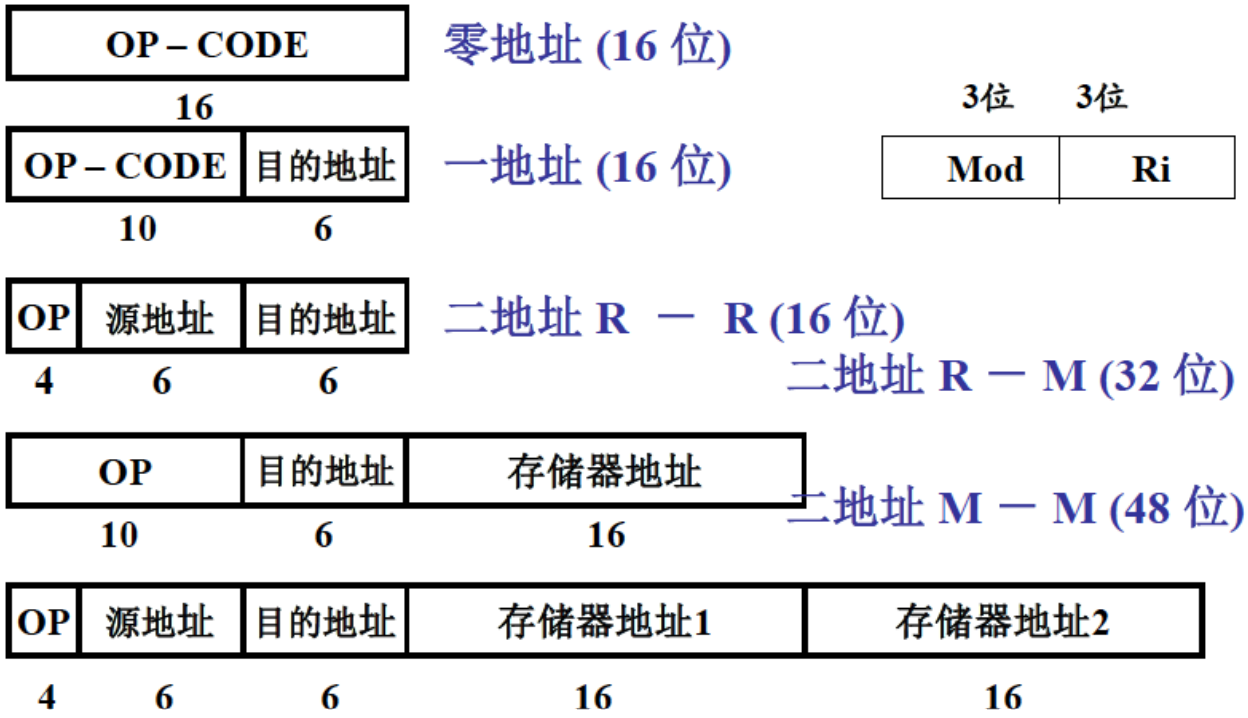
- 若需要在**16**位字长的指令中能够同时支持三地址、二地址、一地址指令各**15**条，零地址指令**16**条，则可以选用如下方案的变长操作码实现：
- 15条三地址指令的操作码为：0000 ~ 1110
- 15条二地址指令的操作码的高4 位选用1111，低4 位用0000 ~ 1110, 即得到: 11110000 ~ 11111110
- 15条一地址指令的操作码的高8位选用11111111, 低4位用0000 ~ 1110, 即: 111111110000 ~111111111110
- 16条零地址指令的操作码的高12位每位均用1，低4位随意, 即: 1111111111110000~1111111111111111

指令操作码扩展技术

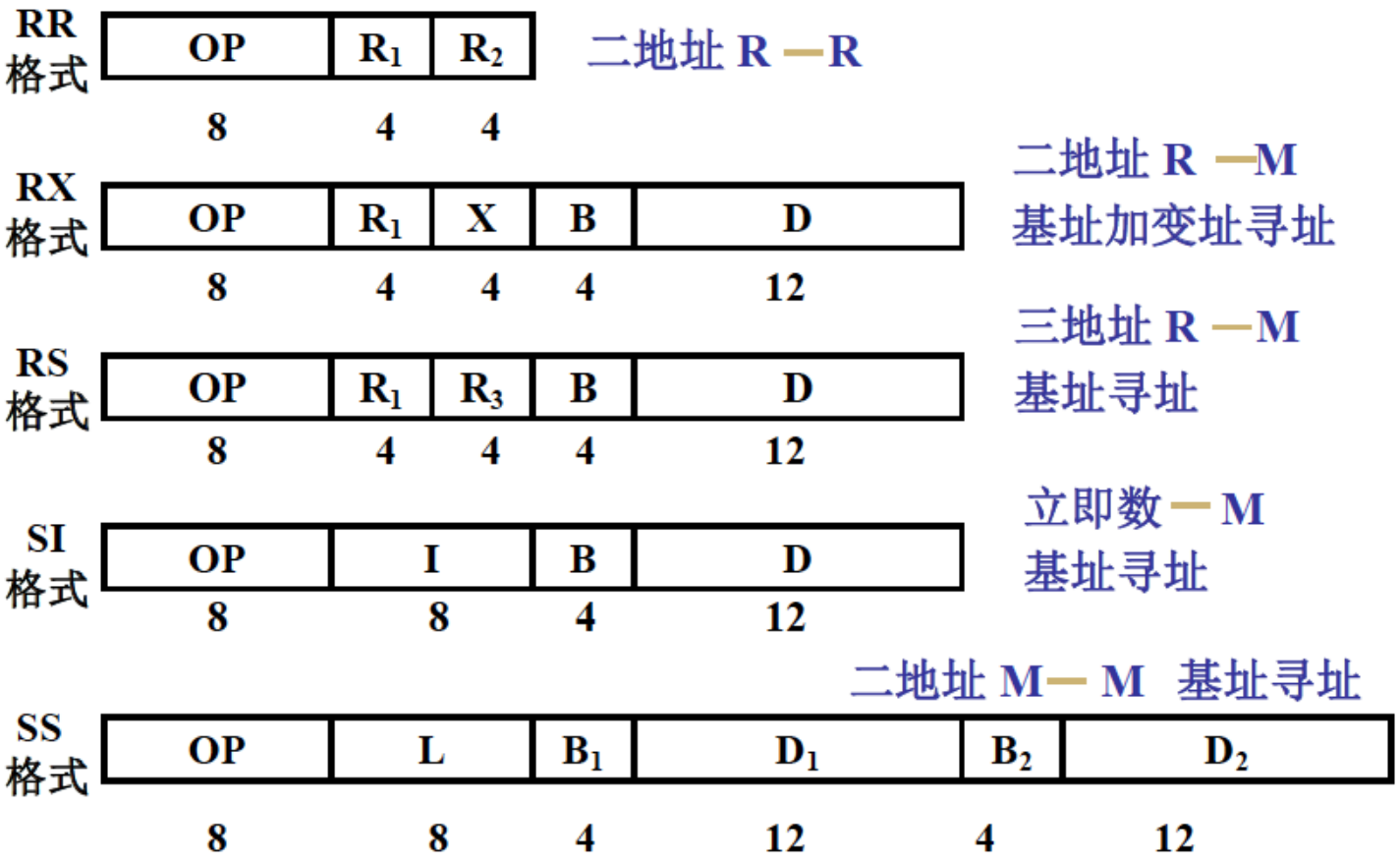
- 前面介绍的操作码扩展方案中，每次扩展4位并仅保留了一个编码用于接下来的扩展过程，当每次扩展的位数和保留的位数变化时，后面可扩展的指令条数就可以变化。例如在16位字中的指令字中，可以选用如下方案支持三地址指令、二地址指令、一地址指令和零地址指令14、30、31、16条：
  - **14**条三地址为：**0000 ~ 1101 (**保留**1110**、**1111** 两个码**)**
  - 30条二地址为：11100000 ~11111101 （保留2个码）
  - 31条一地址为：111111100000 ~111111111110 (保留1个码)
  - 16条零地址为：1111111111110000~1111111111111111

指令操作码扩展技术

- (PDP–11 指令为例)
- 指令字长有16 位、32 位、48 位三种(1字、2字、3字)



IBM 360指令格式

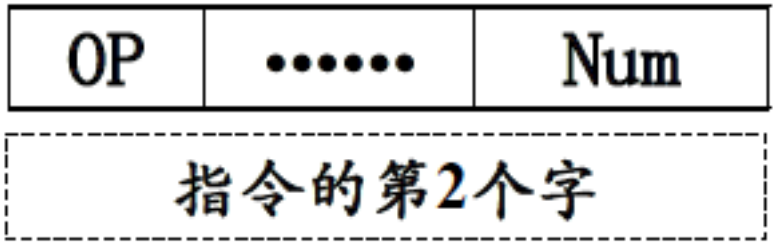

RR
格式
OP
R_1
R_2
二地址 R —R
8
4
4
RX
格式
OP
R_1
X
B
D
二地址 R —M
基址加变址寻址
8
4
4
4
12
RS
格式
OP
R_1
R_3
B
D
三地址 R —M
基址寻址
8
4
4
4
12
SI
格式
OP
I
B
D
立即数 — M
基址寻址
8
8
4
12
二地址 M— M 基址寻址
SS
格式
OP
L
B_1
D_1
B_2
D_2
8
8
4
12
4
12

寻址方式

- 寻址方式（又称编址方式）指的是确定本条指令的操作数地址及下一条要执行的指令地址的方法。
- 不同的计算机系统,使用数目和功能不同的寻址方式，其实现的复杂程度和运行性能各不相同。有的计算机寻址方式较少，而有些计算机采用多种寻址方式。
- 通常需要在指令中为每一个操作数专设一个地址字段，用来表示数据的来源或去向的地址。在指令中给出的操作数（或指令）的地址被称为形式地址，使用形式地址信息并按一定规则计算出来或读操作得到的一个数值才是数据（或指令）的实际地址。在指令的操作数地址字段，可能要指出：
- ①运算器中的累加器的编号或专用寄存器名称（编号）
- ②输入/输出指令中用到的I/O 设备的入出端口地址
- ③内存储器的一个存储单元（或一I/O设备）的地址
- 有多种基本寻址方式和某些复合寻址方式,简介如下。

立即数寻址

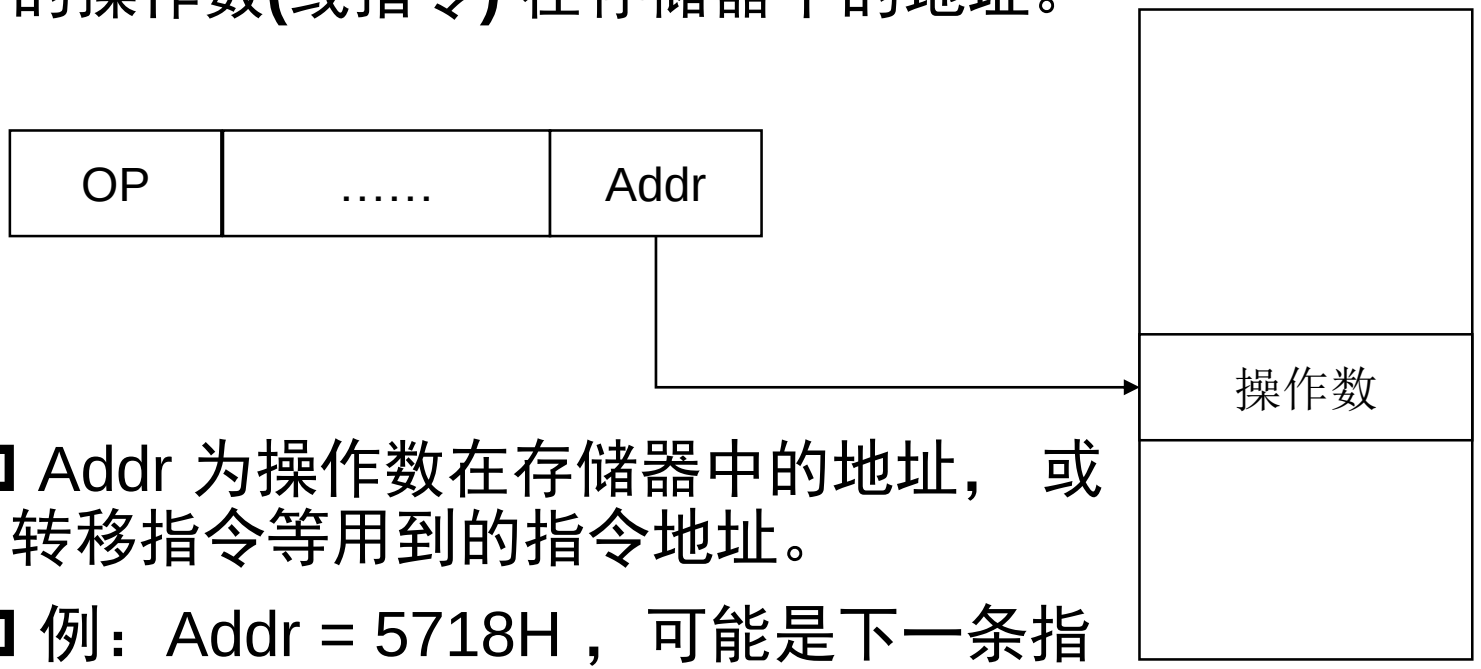
- 所需的一个操作数在指令的地址字段部分直接给出。
- Num 即为操作数的值
-



  - 适用于操作数固定的情况，取指同时取得操作数，指令执行阶段不必到存储器中取操作数，提高了指令执行速度；
  - 当该立即数的值较小(占用位数少)时，可在指令字第一个字中直接给出，否则需要用指令的第二个字提供。
- 例：Num ＝ 1234H，指令的一个操作数就是1234H
- 这里的H 表示1234 是16 进制的值

直接寻址

- 在指令的地址码字段，直接给出所需的操作数(或指令) 在存储器中的地址。



- Addr 为操作数在存储器中的地址， 或转移指令等用到的指令地址。
- 例：Addr = 5718H ，可能是下一条指令的地址或一个操作数的地址，若[5718H] = 3，则用5718H 作地址，从内存储器单元中读出的操作数就是3。

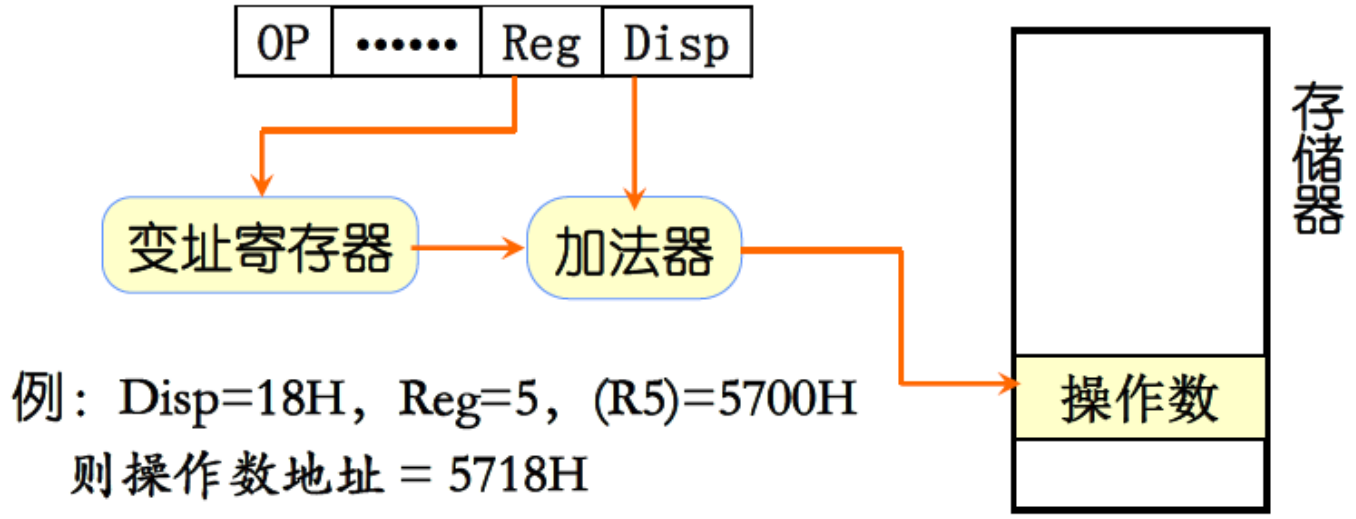
寄存器寻址/寄存器间址

- 计算机的CPU中设置有一定数量的通用寄存器，用于存放操作数、操作数地址或中间结果。假如指令地址码字段给出某一通用寄存器的编号(地址)，且所需的操作数就在这一寄存器中，这就是寄存器寻址方式；若该寄存器中存放的是操作数在内存储器中所在单元的地址，这就是寄存器间接寻址方式。可通过指令的操作码或另设一个字段，来区分这两种不同的寻址方式。

- 例：RegNo.=5，使用5# 累加器，
- 此时若5# 累加器中的内容为7，可记为(R5)=7，
- 对寄存器寻址，操作数就是寄存器中的数值7
- 对寄存器间接寻址，从内存7# 单元读出来的数才是操作数

变址寻址

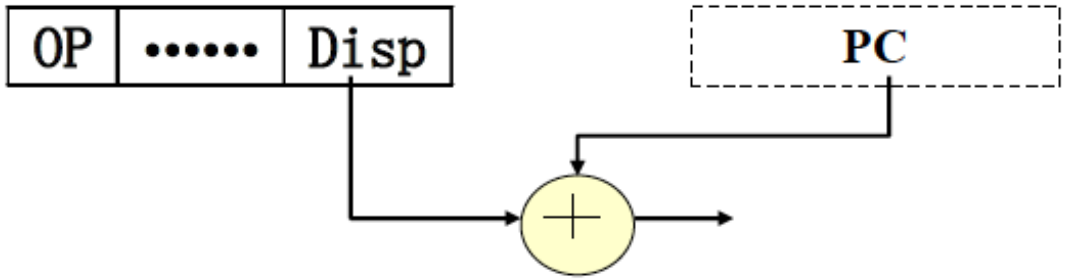
- 操作数的地址由指定的变址寄存器（由Reg指定）的内容和指令中的变址偏移量（Disp）相加得到。



- 变址寄存器内容变化，变址偏移量不变，便于读写数组中的元素，是计算机中常用的一种寻址方式。

相对寻址

- 指令的地址由程序计数器PC 的内容（即当前执行指令的地址）和指令的相对寻址偏移量相加得到。



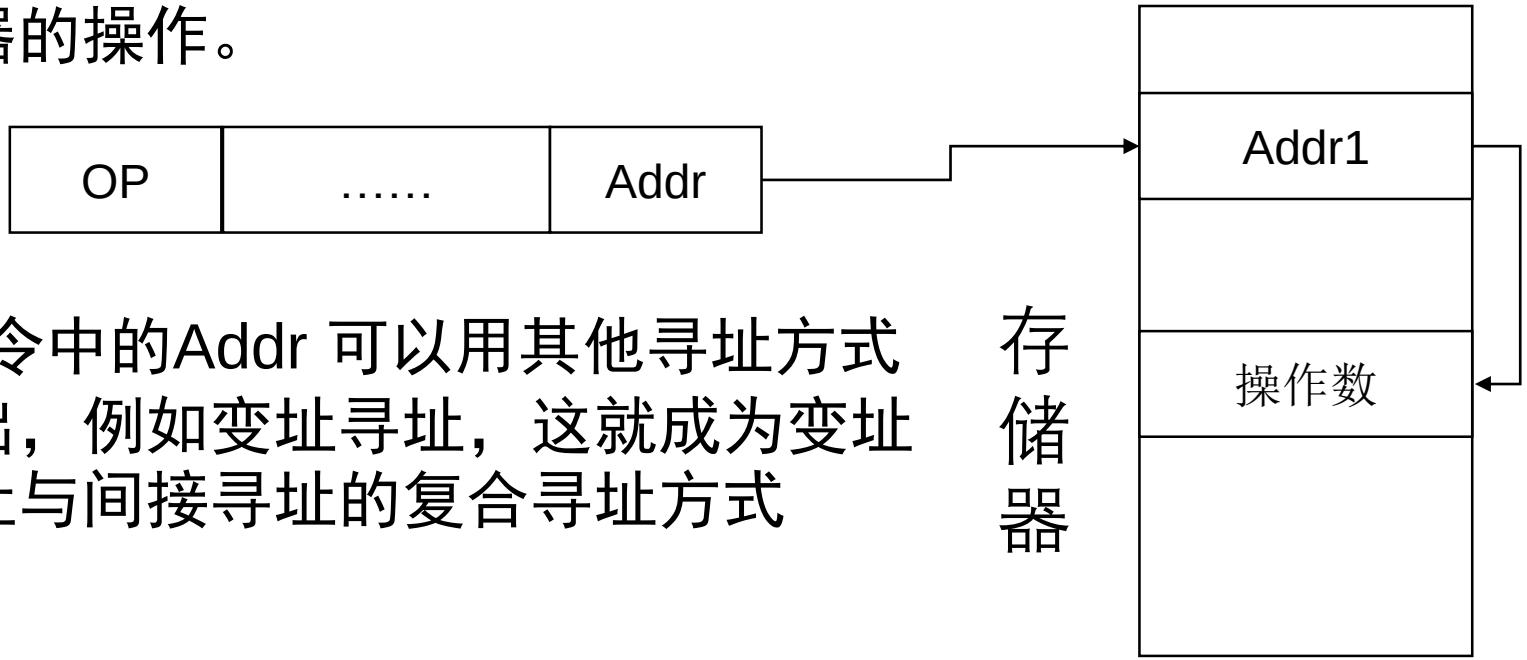
例：Disp = 48H　　(PC) = 5600H

则实际地址 = 5648H

- （1）主要用于转移指令，对浮动程序很有用。
- （2）偏移量可正可负，通常用补码表示。

间接寻址

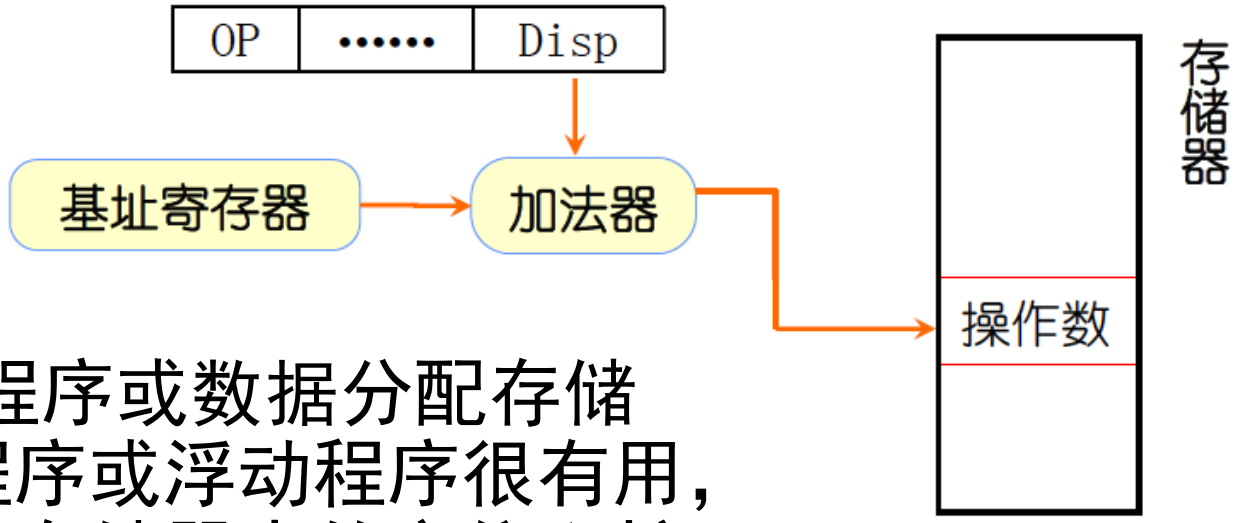
- 指令的地址码字段给出的内容既不是操作数，也不是操作数的地址，而是操作数（或指令）地址的地址， 这被称为间接寻址方式，多一次读内存储器的操作。



- 指令中的Addr 可以用其他寻址方式给出，例如变址寻址，这就成为变址寻址与间接寻址的复合寻址方式

基址寻址

- 在计算机中设置一个专用的基址寄存器，操作数（或指令）的地址通过基址寄存器的内容和指令中的地址码相加得到。

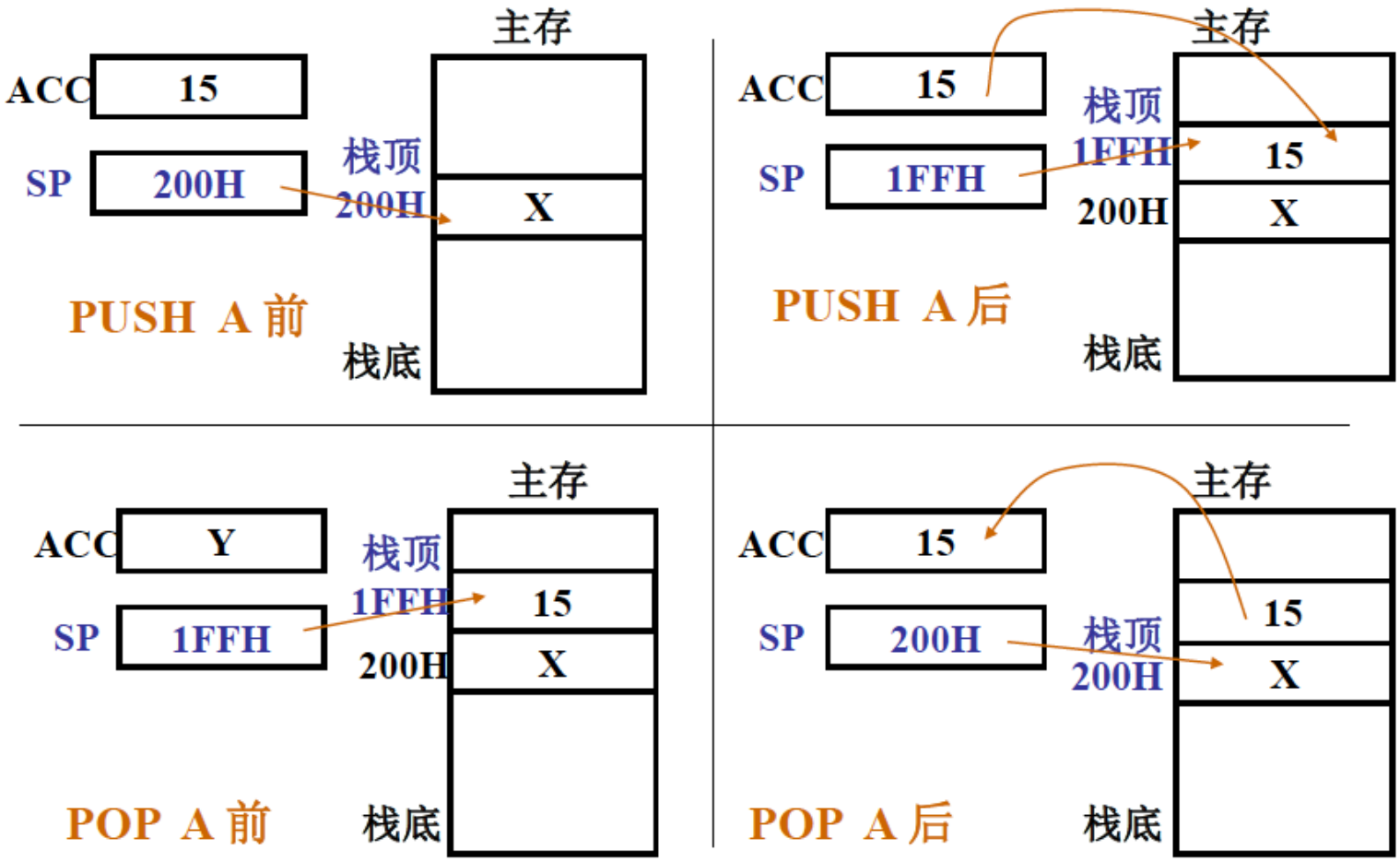


- 主要用于为程序或数据分配存储区，对多道程序或浮动程序很有用，解决了程序在存储器中的定位和扩大寻址空间等问题。

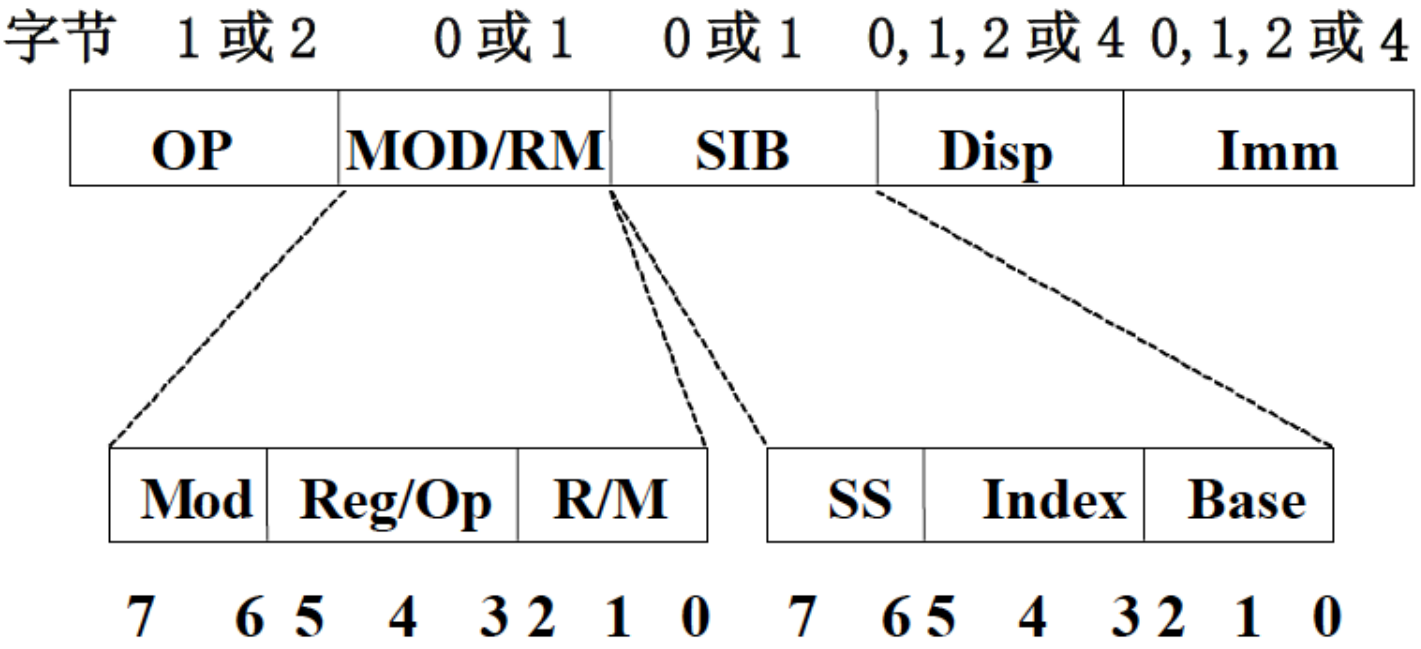
堆栈寻址

- 堆栈是内存储器中一块按“后进先出”原则进行读写的存储区，并通过一个专用的寄存器(称为堆栈指针SP) 给出堆栈的栈顶地址，执行读写堆栈操作通常总在栈顶进行，故不必在指令中给出堆栈地址，而且在读写操作的前后伴随有自动修改SP 内容的动作，确保使SP 总是指向堆栈的栈顶。例如，按字寻址时：
- 入栈操作:SP -1→SP 和AR，即SP 的内容减1 存回SP,并送入内存地址寄存器，接下来才可以把数据写到堆栈中，这是因为需要把数据写到新开辟出来的栈顶单元中。
- 出栈操作: SP →AR，完成一次读堆栈操作后，还要执行一次SP +1→SP 的操作，用于修改SP 内容，这是因为数据读出后原来它的下一个相邻单元变成为栈顶。

堆栈寻址举例


主存
ACC
15
SP
200H
栈顶
200H
X
栈底
PUSH A 前
主存
ACC
15
SP
1FFH
栈顶
1FFH
15
200H
X
栈底
PUSH A 后
主存
ACC
Y
SP
1FFH
栈顶
1FFH
15
200H
X
栈底
POP A 前
主存
ACC
15
SP
200H
15
栈顶
200H
X
栈底
POP A 后

x86 指令系统


字节 1或2 0或1 0或1 0, 1, 2或4 0, 1, 2或4
OP
MOD/RM
SIB
Disp
Imm
Mod
Reg/Op
R/M
SS
Index
Base
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0

x86指令的格式

- OP：指令操作码，某些指令在操作码中还包含有操作数长度或立即数是否需扩充符号位(S) 等信息；
- MOD / RM：与下一字节提供寻址信息。MOD / RM 指出操作数在寄存器中还是在存储器中。该字节分成3个字段：Mod字段与R / M字段可以产生32个编码，分别表示8个寄存器和24种变址方法；Reg / op字段可以是寄存器号，或者作为3位附加的操作码; R / M字段是1个操作数所在寄存器或者与Mod字段一起指出寻址方式。
- SIB：当MOD / RM为某些值时，需SIB 参与决定寻址方式。SIB分成3个字段，SS 指出变址寄存器的放大因子；Index 指出变址寄存器；Base 指出基址寄存器。
- Disp：当寻址方式指示用到disp(位移量)时，存在8位，16位或32位的位移量3种情况。
- Imm：对寻址方式是立即数，分为8位, 16位或32位3 种。
- 前缀：指令前缀、段前缀、操作数前缀、地址长度前缀

x86指令寻址方式

- 1. 立即数
- 2. 寄存器
- 3. 直接　　E = Disp
- 4. 基址　　E = (B)　　B基址寄存器
- 5. 基址 + 偏移量　　E = (B) + Disp
- 6. 比例变址 + 偏移量　　E = (I) * S + Disp
- 　　I 变址寄存器，S为比例因子（1,2,4,8)
- 7. 基址 + 变址 + 偏移　　E = (B) + (I) + Disp
- 8. 基址 + 比例变址 + 偏移　E = (B) + (I)*S + Disp
- 9. 相对　　指令地址 = (PC) +Disp

数据移动指令

指令	说明
MOV DST, SRC	数据从SRC复制到DST
PUSH SRC	把SRC 压入堆栈
POP DST	从堆栈弹出一字存入DST
XCHG DS1, DS2	交换 DS1 和 DS2
LEA DST, SRC	把SRC的有效地址存储DST
CMOV DST, SRC	条件复制

算术指令

指令	说明
ADD DST, SRC	把SRC加到DST中
SUB DST, SRC	从SRC减去DST
MUL SRC	EAX乘以SRC（无符号）
IMUL SRC	EAX乘以SRC（带符号）
DIV SRC	EDX : EAX 除以 SRC（无符号）
IDIV SRC	EDA : EAX 除以 SRC（带符号）
ADC DST, SRC	把SRC加到DST中，并加进位位
SBB DST, SRC	从SRC中减去DST和进位位
INC DST	DST 加 1
DEC DST	DST 减 1
NEG DST	DST取反（也就是0-DST）

逻辑指令

指令	说明
AND DST, SRC	SRC 和 DST进行逻辑与，结果存入 DST
OR DST, SRC	SRC 和 DST进行逻辑或，结果存入 DST
XOR DST, SRC	SRC 和 DST进行逻辑异或，结果存储 DST
NOT DST	把DST替换成二进制反码

二 —— 十进制数指令

指令	说明
DAA	十进制调整
DAS	为减法进行十进制调整
AAA	为加法进行ASCII调整
AAS	为减法进行ASCII调整
AAM	为乘法进行ASCII调整
AAD	为除法进行ASCII调整

移位/循环移位指令

指令	说明
SAL/SAR DST, #	DST左移或者右移#位
SHL/SHR DST, #	DST逻辑左移或者右移#位
ROL/ROR DST, #	DST循环左移或者右移#位
RCL/RCR DST, #	通过进位位对DST移位#位

测试/比较和转移指令

指令	说明
TST SRC1, SRC2	逻辑与，根据结果设置标志位
CMP SRC1, SRC2	依SRC1-SRC2的结果设置标志位
JMP ADDR	跳转到ADDR
Jxx ADDR	基于标志执行条件转移
CALL ADDR	调用ADDR处的过程
RET	从过程返回
IRET	从中断返回
LOOP xx	循环直到条件满足
INT ADDR	初始化一个软件中断
INT0	如果溢出位被设置则发生中断

字符串指令

指令	说明
LODS	读取一个串
STOS	保存串
MOVS	复制串
CMPS	比较两个串
SCAS	扫描一个串

条件码指令

指令	说明
STC	设置EFLAGS寄存器中的进位位
CLC	清除EFLAGS寄存器中的进位位
CMC	EFLAGS中的补码进位位
STD	设置EFLAGS寄存器中的方向位
CLD	清除EFLAGS寄存器中的方向位
STI	设置EFLAGS寄存器中的中断位
CLI	清除EFLAGS寄存器中的中断位
PUSHFD	EFLAGS寄存器入栈
POPFD	EFLAGS寄存器出栈
LAHF	从EFLAGS寄存器中读取AH
SAHF	把AH保存到EFLAGS寄存器中

其它指令

指令	说明
SWAP DST	改变DST的字节顺序
CWQ	为了进行除法，把EAX扩展成EDX：EAX
CWDE	把AX中的16位数扩展成EAX
ENTER SIZE, LV	创建SIZE个字节的堆栈段
LEAVE	撤销ENTER创建的堆栈段
NOP	空操作
HLT	停机指令
IN AL, PORT	从PORT端口向AL输入一个字节
OUT PORT, AL	从AL向PORT端口输出一个字节
WAIT	等待中断

x86指令系统特点

- 指令格式复杂
- 寻址方式多样
- 通用寄存器较少
- 兼容直到8086
- 编译系统复杂
- 指令流水实现复杂
- 典型CISC 指令集

现代x86指令系统

- 系统的运行模式
  - 实模式，兼容16位的物理地址模式
  - 32位保护模式，分页分段模式（地址扩展模式）
  - 64位平坦模式，分段模式禁止
  - SMM模式（用以处理紧急情况）
  - 虚拟机模式
  - SGX兼容运行环境
- 系统的指令
  - 应用程序指令，SIMD指令
  - 模式相关指令
  - 内存相关指令
  - IO相关指令

系统的运行模式

- 决定当前处理器的运行的系统上下文，对相同的指令可能有不同的解释，使用不同的物理资源
  - 运行在实模式：直接访问物理地址，只能用16位地址模式
  - 运行在保护模式：必须通过段页式转换，才能访问物理地址
  - 运行在虚拟机模式：经过两层的地址转换，才能访问物理地址

不同的x86指令类别

- General Purpose Instructions: 算术指令，IO指令，跳转分支指令
- 系统级指令
- x87 FPU指令
- SIMD: MMX, SSE, AVX
- 加解密指令，密码学哈希指令
- TSX硬件事务处理指令
- VMX 虚拟机扩展指令
- SMX安全执行扩展指令
- SGX用户级安全执行环境指令

x86指令举例

- 短的指令：int 3，指令码为0xcc
- The maximum length of an Intel 64 and IA-32 instruction remains 15 bytes.(from the manual)
- AVX指令可以到这个长度，超过这个长度会抛出非法指令异常
- 按照不同的方法，统计结果不一样，粗的有1000条左右，特别细致的话会有3000条左右指令
- 最新的手册有4922页（指令手册加在一起）
- 除了指令本身外，指令所处的运行模式，寻址方式也造成了指令执行的复杂度
- 作为对比：MIPS32指令有150条左右

小结

- 控制器的主要功能
  - 正确执行指令规定的功能
  - 自动、连续执行指令
- 指令系统
  - 是硬件和软件的接口
  - CISC和RISC
- 指令格式
  - 指令如何用二进制编码表示，主要是操作码和操作数地址的安排方案
- 寻址方式
  - 操作数寻址方式
  - 对操作系统、编译程序提供支持，方便程序员使用计算机
  - 寻址方式的选择

阅读和思考

- 阅读
- 思考
  - 计算机指令系统中哪些是必备指令?为什么?
  - 指令寻址方式有哪些?这些寻址方式可以在高级语言程序中找到哪些影子?
  - 根据ThinPAD RISC-V指令系统要求，确定ALU应具备的功能。

谢谢