计算机系统结构
Computer Architecture

Ch.02 – Instruction System
指令系统

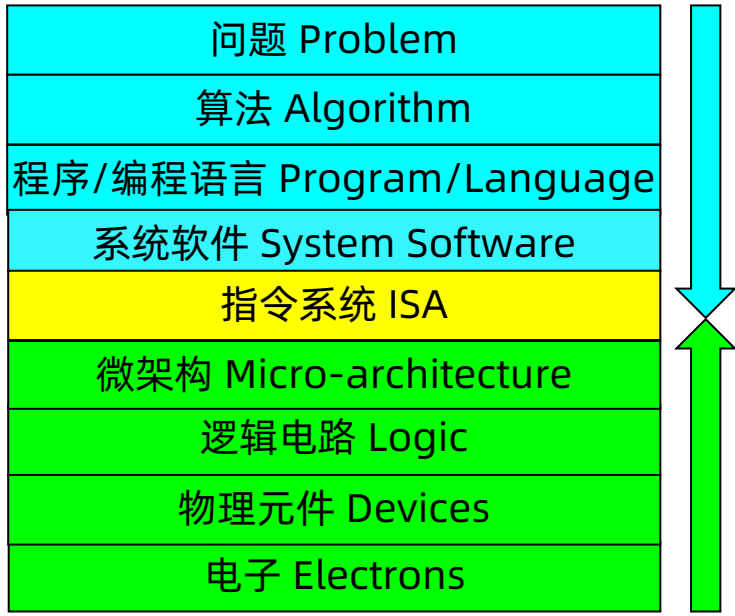
Dr. Borui Li
李博睿 博士

2026年秋季·东南大学

Part I. 为什么要学习指令系统?

回顾计算机系统结构的定义

- **Amdahl经典定义**
  - 传统级机器(机器语言)程序员所看到的计算机属性，包括**概念性结构和功能特性**
- **Hennessy and Patterson广义定义**
  - 系统结构包含**指令集结构**、组成及硬件
- **Onur Mutlu定义**
  - 是在考虑各种独特目标（性能、功耗、成本等）的情况下
  - 硬件组件设计、选择、连接和**软硬接口设计**的科学和艺术



Part I. 为什么要学习指令系统?

Part II. 什么是指令系统?

为“图片变亮”程序设计一套指令系统

编写的高级语言代码，机器无法理解

```
for (int i = 0; i < N; i++) {
  int x = src[i] + delta;
  if (x > 255) x = 255;
  dst[i] = (uint8_t)x;
}
```

原始像素	增加亮度	写回结果
250	+ 20	255

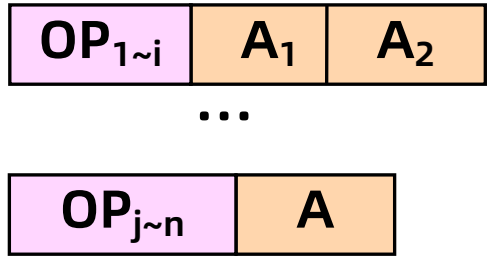
指令系统的概念

- **机器指令 Instruction**
  - 定义：指硬件可直接识别和执行(实现功能)的命令
  - 表示：一定格式(指令格式)的二进制编码

格式1：

…

格式k：



- **指令系统 Instruction System**
  - 定义：指所有机器指令的集合 → 反映硬件所有功能
  - 实质：指令功能–指令格式(硬件–软件)的约定

多条指令共用一种格式
不同指令通过OP码标识

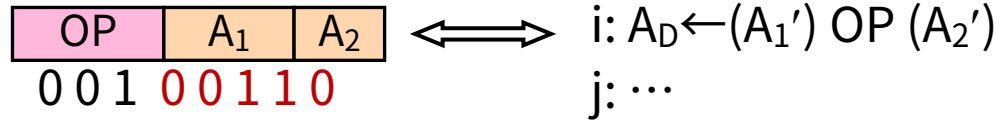
指令系统=指令功能+指令格式

- **指令功能**
  - 所含信息：操作类型、源/目的地址形式、下条指令地址

表示

- **指令格式**
  - **操作码**表示的信息 - 操作类型、格式信息(地址码个数+目的OPD位置等)

标识不同指令的功能



Operand
操作数

  - **地址码**表示的信息 - 源/目的OPD(地址形式)、下条指令地址

Part I. 为什么要学习指令系统?

Part II. 什么是指令系统?

Part III. 指令系统具体设计

指令功能设计→数据表示设计→数据地址设计→ISA结构设计→指令格式设计与优化

指令功能设计

- **常见指令功能分类**
  - 操作OP:

分类	操作类型	数据类型相关性
数据传送	Load/Store，In/Out	×
转移控制	分支/跳转/调用/返回/自陷等	×
系统控制	OS调用、虚存管理等	×
算逻运算	整数加/减/乘/除等，逻辑数与/或/非等	√
浮点运算	浮点加/乘等	√
十进制	十进制加、乘等	√
字符串	字符比较，字符串拷贝、移动、比较等	√
图形	像素操作、压缩/解压操作等	√

~硬件成本

  - 操作数数据类型：指**运算时的**数据类型，OPD集合≤数据表示集合
    - 如：数据表示支持8/16/32b整数， ALU仅支持32b运算

指令功能设计

- **设计基础：**需求程序
- **设计内容：**拟支持的操作类型(OP功能+OPD类型)
- **设计目标：**CISC思路与RISC思路有所不同
  - CISC：侧重**强化**指令功能，减少目标代码长度
  - RISC：侧重**简化**指令功能，以提高执行速度、适应并行处理及VLSI

CISC v.s. RISC

	RISC简单指令集（ARM）	CISC复杂指令集（X86）
吃饭的例子	指令：拿勺子->舀起一勺饭->送到嘴里->咽下去	指令: 吃饭
优劣势	• **指令多**，需要好几个指令实现一个复杂功能 • 针对常用指令优化，处理效率高 • 不常用指令则需要指令组合方式，可以通过流水 • 技术和超标量技术弥补。指令执行周期<=1cycles	• **指令简单**，不需要下达太多指令 • 指令动作复杂，速度慢；设计复杂，且流水中很多操作浪费 • 指令执行周期大都在5~8 cycles

- **典型CISC架构处理器**
  - Intel x86 (8086/Pentium/Core系列)、AMD x86 (Ryzen/RPYC系列)
- **典型RISC架构处理器**
  - ARM (移动端：苹果A系列/高通骁龙系列、桌面服务器端：华为鲲鹏920)
  - RISC-V、MIPS

指令功能设计——CISC思路

- **设计方法**
  - ① 需求程序**分解**为若干基本操作及数据表示
  - ② **统计**各基本操作类型的**使用频率**
  - ③ **选择**拟支持的基本操作

} **频带分析法**

- **主要思想**
  - 支持**频率较高**的基本操作

指令功能设计——CISC思路 (Cont'd)

- **例1：**计算机系统需求包含BCD数的支持，需求程序中的各基本操作及使用频率如下：
  - (忽略OPD设计)
  - BCD数
    - Binary Coded Decimal
    - 二进制编码的十进制
    - 0011 0001 <-> 31

操作类型		频率	操作类型		频率	操作类型		频率	操作类型		频率
算术	加减	0.150	传送	取	0.060	浮点	加减	0.060	十进制	加减	0.008
	±1	0.050		存	0.050		乘除	0.039		乘除	0.007
	乘除	0.100	控制	分支	0.100		±1	0.001		调整	0.005
逻辑	与	0.040		跳转	0.050	图形	叠加	0.015	字符串	拷贝	0.004
	或	0.040		调用	0.020		反色	0.003		比较	0.004
	非	0.005		返回	0.020		压缩	0.001		移动	0.002
	异或	0.020		自陷	0.015		解压	0.001	系统		0.140

指令功能设计结果：支持频率≥1%的操作、特殊操作(e.g., 调整)

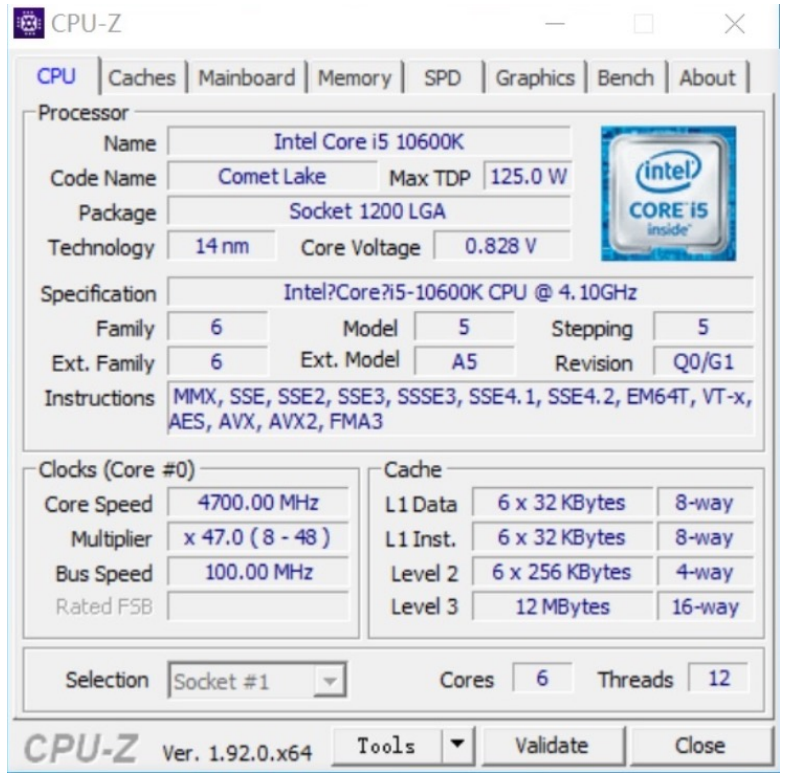
说明： ①低频率操作可用软件实现，如逻辑非、浮点±1

②可分解的操作无需统计，如图形叠加 ③特殊操作必须支持，如调整运算(二进制运算←→BCD码)

(保持完整性) (15H+06H=1BH→21H)

CISC指令集的功能优化

- **优化方法：**
  - ①统计各指令操作在代码中及执行时的使用频率，
  - ②根据指令(串)的使用频率，增强指令功能、增设新指令



Intel AVX指令集 (e.g., VADDPS指令)

AVX：256-bit 向量加法

YMM1	1	2	3	4	5	6	7	8
+								
YMM2	10	20	30	40	50	60	70	80
=	8 组加法同时进行							
YMM0	11	22	33	44	55	66	77	88

```
VADDPS ymm0, ymm1, ymm2
```

YMM 寄存器宽度为 256 bit，可容纳 8 个 FP32 数。

指令功能设计——RISC思路

- **CISC指令集的主要问题**
  - 为了**提高代码效率**→指令格式、编译、体系结构的复杂化
  - 因为**指令功能不平衡**→不利于并行处理(如流水线)的实现
  - 因为**指令格式复杂**→控制器复杂，不利于大规模硬件设计
- **RISC目标：**侧重简化指令功能，以提高执行速度、适应并行处理及大规模集成电路设计
- **设计与优化方法：**同CISC的功能设计与优化方法
- **设计结果：**支持频率很高的基本操作
  - 例2：对例1中的应用，RISC支持使用频率$\geq 2\%$的操作、特殊操作

CISC与RISC的比较

- **比较方法：**CPU性能公式

$$\text{程序CPU时间} = \text{IC} \times \text{CPI} \times \text{时钟周期时间}$$

- **特征比较**
  - CISC：强化指令功能，减少IC、增加CPI
  - RISC：简化指令功能，减小CPI、增加IC

- **结论：**RISC占优（利于并行处理、VLSI）

Part I. 为什么要学习指令系统?

Part II. 什么是指令系统?

Part III. 指令系统具体设计

指令功能设计→**数据表示设计**→数据地址设计→ISA结构设计→指令格式设计与优化

- CISC
- RISC

数据结构与数据表示

- **数据结构：** 指**软件**能够直接识别和引用的数据类型
- **数据表示：** 指**硬件**能够直接识别和引用的数据类型

- **两者关系：** 数据表示是数据结构的子集

数据结构 —直接使用法(x%)→ **数据表示**

数据结构 —软件映像法(1-x%)→ **数据表示**

软/硬件间语义差别

- **数据表示所含内容：** 表示方法（格式/进制/编码）+数据长度

数据表示的设计

- **数据表示常见类型**
  - 基本数据表示：定点数、浮点数、逻辑数、字符等
  - 高级数据表示：向量、树、链表、指针等

- **设计基础：**程序需求，性能及性价比需求

- **设计内容：**拟支持的数据类型及其参数

- **设计目标：**性价比较高，性能关注计算速度、存储效率

存储效率是什么？

数据表示的设计

- **设计方法**
  - 需求程序分解为若干基本操作及基本数据类型
  - 统计各种基本数据类型的使用频率
  - 选择拟支持的数据类型及其参数

称为**频带分析法**

- **主要思想：**支持频率较高的数据类型及其参数

关注经常性事件

- **影响因素：**性能、性价比需求

Amdahl定律

例：若标量部件占硬件成本的5%，向量部件的速度、成本分别为标量部件的10倍、3倍，测试程序中向量操作占10%，则支持向量表示后的性/价是否有提高？

解：设测试程序执行时间 = t，硬件成本 = s，则性/价 = $(1/t)/s=1/(t*s)$ **；**

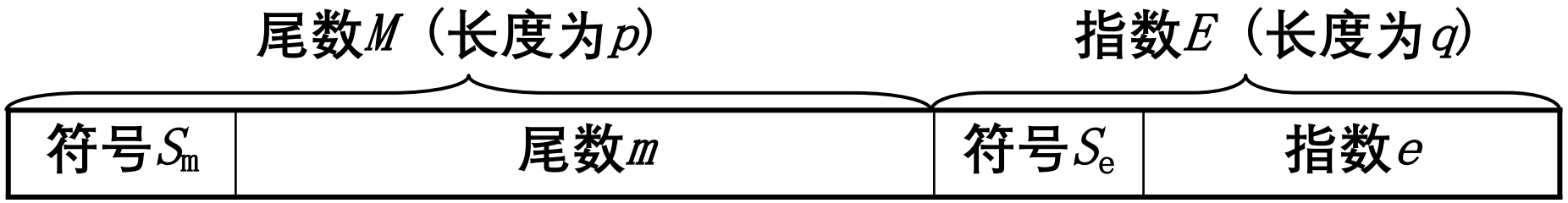
新的执行时间 = $0.9t+0.1t/10 = 0.91t,$

成本 = $s+0.05*3s = 1.15t,$ ←标量部件还需要存在

性/价 = $(1/0.91t)/1.15s = 0.96/(t*s)$

数据表示设计举例——浮点数

- **浮点数的表示**
  - 表示方法：纯小数尾数+纯整数指数（阶）



  - 格式参数：尾数和指数的码制、数制 、长度
    - 原/反/补；2/4/8/10/16进制、FP32/FP16/等
  - 设计内容：格式参数设计、操作处理设计
  - 设计基础：需求程序，表示范围及精度需求

Part I. 为什么要学习指令系统?
Part II. 什么是指令系统?
Part III. 如何设计指令系统?
Part IV. 指令系统具体设计

指令功能设计→数据表示设计→**数据地址设计**→ISA结构设计→指令格式设计与优化

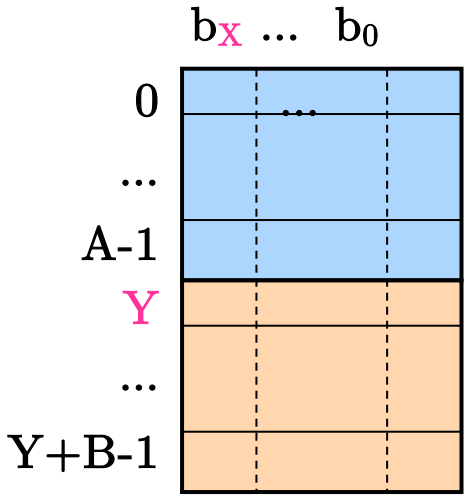
- CISC
- RISC
- 格式参数
- 操作处理

数据地址设计=编址方式设计+寻址方式设计

- **设计基础：**已确定的存储部件类型、需求程序

- **设计内容**
  - 编址单位：部件单元存放的信息长度（x）
  - 地址空间：部件单元的个数（A、B）
  - 编址方式：不同部件地址间的关系（Y）

影响部件的识别方法



- **选择依据**
  - 存储效率（数据长度/占单元数）和访问效率（访问次数/OPD）均好

内存的编址单位设计

- **数据长度的特征：**逻辑数（1位）、其他数据
- **编址单位的类型：**二进制位、机器字长、折中长度

地址 数据(1位)

0000H	0
0001H	1
…	…

地址 数据(设字长为16位)

0000H	0000100011110000
0001H	0000000010001111
…	…

地址 数据(8位)

0000H	00001000
0001H	10001111
…	…

- **编址单位设计**
  - 类型选择：常为折中长度 ←重存储效率、轻访问效率

↓可改进 [如并行访问]

↓ASCII码最常用、逻辑数可组合

  - 设计结果：$=\min\{\text{所有数据表示的长度}\}=2^{m}$位，通常为字节
  - 数据存放：占k个连续存储单元$(k\geq 1)$ ←如何放?

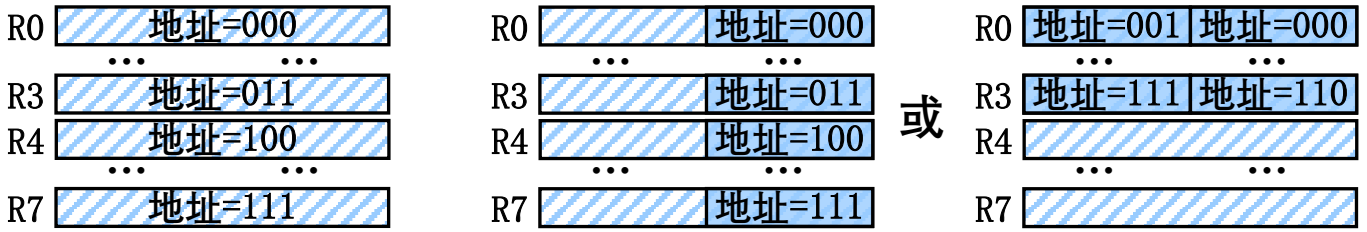
其他部件的编址单位设计

- **外设的编址单位：** 与内存一致（含数据存放方式：端序/对齐） ←面向传输（均通过总线访问）

- **REG的编址单位：** 与OPD长度对应（=ALU运算位数） ←面向运算（暂存运算结果）

思考–数据表示有8/16/32位，CPU的OPD支持16/32位，REG编址单位？依据?

- REG组织：设置一个REG，通过重命名构成多个编址空间
- 使用方法：多个地址空间可同时使用



- **堆栈的编址单位：** 常与REG一致 ←操作速度快（无需长度转换）

地址空间设计

- **内存地址空间设计**
  - 所有需求程序(考虑可扩展性)所需内存总和的最大值　如：ARMv8为48位
- **I/O地址空间设计**
  - 所有需求程序(考虑可扩展性)需要存储总和的最大值　如：IA32为16位
- **REG地址空间设计**
  - 专用REG数量：取决于OS、组成，无需编址
  - 通用REG数量：取决于编译程序、需求程序，显式编址

编址方式设计

- **编址方式：指不同部件地址之间的关系**

- **编址方式类型：**独立编址、统一编址 **←堆栈无需编址**

- **编址方式的设计：**
  - REG <-> MEM及I/O：独立编址方式 **←访问路径不同、地址短**
  - MEM <-> I/O：任选一种
    - 如：x86为独立编址，ARM为统一编址

数据地址设计=编址方式设计+寻址方式设计

- **设计基础：**需求程序、部件编址方式（单元/空间/相互关系）
- **设计内容：**拟支持的寻址方式及其参数

- **寻址方式种类的确定**
  - 常见：立即，REG，直接、REG间接、偏移(基址/变址)、相对等
  - 其他：索引[(RB)+(RI)]、比例变址[(RI)*S+A]等

- **选择依据：**能/否减少指令长度或指令条数，对指令CPI影响小

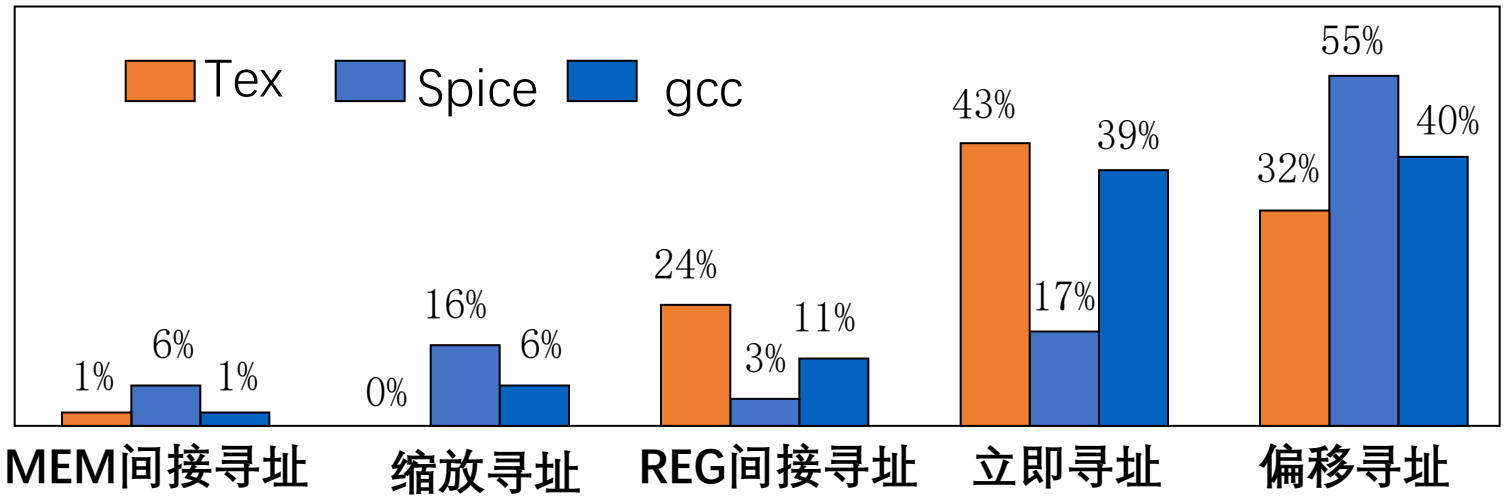
寻址方式种类的确定

- **设计方法**
  - 形成需求程序中各基本操作的数据&指令寻址方式
    ↳←种类尽量多
  - 统计各种寻址方式的使用频率，
  - 选择高频率的寻址方式

- **设计结果**（高频率的定义不同）
  - CISC：尽量多的寻址方式，增加灵活性 ←减少指令长度/条数
  - RISC：尽量少的寻址方式，简化硬件 ←对CPI影响小

寻址方式种类的确定 (Cont'd)

- **例1**：VAX计算机中，运行gcc、Spice和Tex基准程序，各种寻址方式的分布如下，该支持哪些寻址方式?

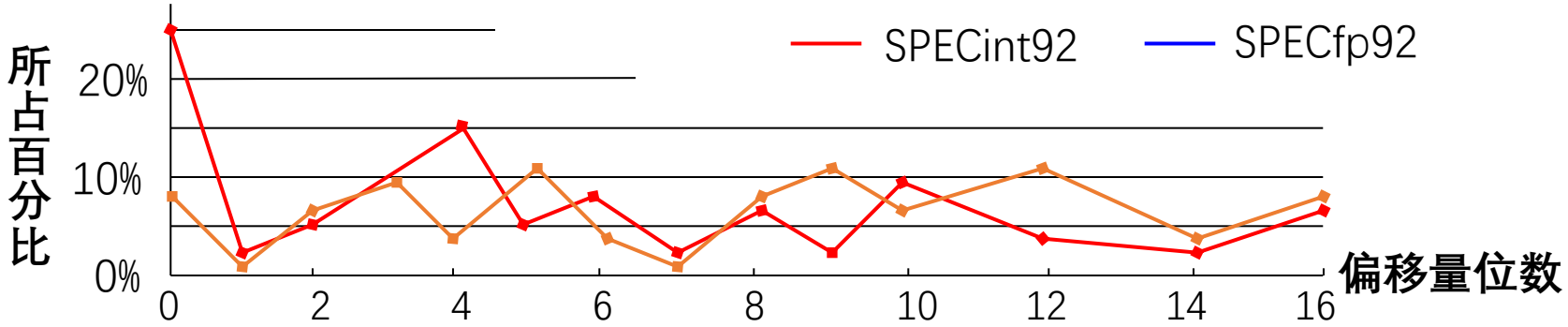


注：偏移寻址—带偏移量的寻址方式，如基址、变址、相对寻址

- **选择结果**
  - 偏移、立即、REG间接寻址方式，需支持 (高频率)
  - 其他寻址方式，依据CISC/RISC风格，再选择

寻址方式参数的确定

- **设计方法** **(对每种寻址方式分别设计)**
  - 统计每个参数不同取值的使用频率　　←即频带分析法
  - 选择满足需求$(\geq m\%)$的参数值(可有几种)
- **例2：**某R-R型机器上运行测试程序时，偏移寻址结果分布如下，要求长度应满足≥85%的访问，偏移长度该如何选择?



选择结果： 长度仅1种时，为12~16位(可结合指令字长再确定)；

长度有2种时，为6位、14位　　←利于缩短平均码长

Part I. 为什么要学习指令系统?

Part II. 什么是指令系统?

Part III. 指令系统具体设计

指令功能设计→数据表示设计→数据地址设计→**ISA结构设计**→指令格式设计与优化

- CISC
- RISC

- 格式参数
- 操作处理

- 编址方式
  - 单位、空间、方式
- 寻址方式

ISA结构分类

- **分类方法**
  - 按操作数(OPD)的存放部件分类　←指运算类指令的OPD
- **结构分类**
  - 堆栈型：OPD放在栈(栈在MEM中)，指令条数多/OPD访问慢
  - 累加器型：OPD放在AC及MEM中，指令字长长/OPD访问较慢
  - 通用寄存器型：OPD放在GPR中，指令字长短/执行速度快/…，
    - 风格有R-M型(CISC)、R-R型(RISC)
- **趋势:** R-R风格的通用寄存器型
- **指令格式相关参数：**指令字结构、显式OPD个数

ISA结构分类

Operations that perform i++

```
PUSH    &i ;
PUSH    0x01;
ADD;
```

堆栈型 Stack

Operations that perform i++

```
LDA    &i ;
ADDA    0x01;
STA     &i;
```

累加型 Accumulator

Operations that perform i++

```
MOVE    d0, &i ;
INC     d0;
MOVE    &i, d0;
```

通用寄存器型R-M风格【CISC】（Register-Memory）

Operations that perform i++

```
LD    r7, &i ;
ADD    r7, 0x01, r7;
ST    r7, &i;
```

通用寄存器型R-R风格【RISC】（Register-Register）

ISA结构设计

- **设计目标：**应用需求(程序)，性能、性价比需求
- **设计内容：**ISA结构类型、指令字结构、显式OPD个数、寄存器数量

- **ISA结构类型的确定**
  - 类型：堆栈型、累加器型、通用寄存器型(R-M或R-R)
  - 设计：趋势为R-R风格的通用寄存器型，如ARM、MIPS、RISC-V

↑硬件成本下降　↑利于并行处理(指令功能简单)

- **指令字结构的确定**
  - 类型：定长指令字、变长指令字 (←指令字长=k*存储单元长度)
  - 设计：取决于ISA结构类型(R-R型为定长结构，其余常为变长结构)

↑目标是提高对软/硬件的支持程度

思考–为何R-M型指令集不宜采用定长指令字结构?

ISA结构设计 (Cont'd)

- **显式OPD个数的确定** **(主要指运算型指令[其余指令OPD较少])**
  - 类型：堆栈型0个，累加器型1个，通用寄存器型≥2个

Operations that perform *i*++

```
PUSH   &i ;
PUSH   0x01;
ADD;
```

Operations that perform *i*++

```
LDA    &i ;
ADDA   0x01;
STA    &i;
```

Operations that perform *i*++

```
MOVE   d0, &i ;
INC    d0;
MOVE   &i, d0;
```

Operations that perform *i*++

```
LD     r7, &i ;
ADD    r7, 0x01, r7;
ST     r7, &i;
```

  - 设计：基于ISA类型(通常RISC＞2、CISC＝2) **←提高指令效率**

结构类型	显式OPD个数	MEM型OPD数	示例
R-R	3	0	MIPS、PowerPC、ARM
R-M	2	1	80x86、Motorola 68000
M-M	2、3	2、3	PDP-11、VAX

ISA结构设计 (Cont'd)

- **寄存器数量的确定**
  - 专用寄存器数量设计：取决于OS、组成，用户程序不可见

    如：控制寄存器、状态寄存器等，仅特权指令可以使用

  - 通用寄存器数量设计：取决于编译程序、应用程序

    寄存器分配算法　变量使用特征

ARM Calling Convention

设计内容：
ISA结构类型、指令字结构
显式OPD个数、寄存器数量

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

Part I. 为什么要学习指令系统?

Part II. 什么是指令系统?

Part III. 指令系统具体设计

指令功能设计→数据表示设计→数据地址设计→ISA结构设计→**指令格式设计与优化**

- CISC
- RISC

- 格式参数
- 操作处理

- 编址方式
  - 单位、空间、方式
- 寻址方式

- 堆栈型
- 累加器型
- 通用寄存器型

指令格式设计 = 操作码设计+指令格式设计

- **设计基础：** 已确定的操作类型、使用频率、指令字结构
- **设计内容：** 各条指令的操作码编码
- **设计步骤：** 确定需编码信息种类，确定编码方法，进行编码
- **需编码信息种类：** $=\sum$(指令i操作类型数$\times$指令i格式信息数)

  = 操作功能×OPD类型种类→

  = 地址个数种类×目的OPD位置种类等→

思考①： 整数加法指令支持8/16位OPD、单/双地址(目的OPD靠右)，编码个数?
栈操作指令支持16位的压栈/出栈操作，编码个数?

4种，2种

- **编码方法：** 常采用扩展编码的方式 **←规整性较好、平均码长较短**

思考②： 3种频率操作码信息种数为4、15、56，编码时如何扩展? 扩展目标?

按频率分布，目标是平均码长最短

操作码的编码

- **扩展编码分类：**方向有横向、纵向、混合，长度有等长、不等长

横向扩展		纵向扩展		混合扩展	
OP个数	扩展编码	OP个数	扩展编码	OP个数	扩展编码
15种	0000 … 1110	4种	0xx	10种	0000 … 1001
15种	1111 0000 … … 1111 1110	16种	1xx 0xx	20种	1010 00 … … 1110 11
15种	1111 1111 0000 … … … 1111 1111 1110	64种	1xx 1xx 0xx	64种	1111 00 0000 … … … 1111 11 1111
(OP频率分布1:1:1)		(OP频率分布1:4:16)		(OP频率分布1:2:5)	

思考①—OP分布为4、15、64时，不等长横向扩展时如何编码？

思考②—两个操作码分别为0100及01001001，编码正确吗？

- **扩展方法：**根据所有编码信息的频带进行扩展

思考③—各操作频率为：0.12有4种，0.02有15种，0.004有55种，如何扩展？平均码长？

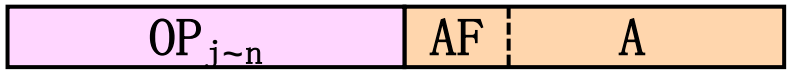
指令格式设计 = 操作码设计+指令格式设计

- **设计基础：**已确定的指令集功能、操作码、寻址方式
- **设计内容：**各条指令的指令字格式

- **指令字格式设计**
  - 方法：组织各条指令的地址码，合并操作码、地址码

第p种指令格式：

第k种指令格式：



注：①各指令的地址码中寻址方式位可单独编码(如2种时占1位，结合OP指明)

②各指令的操作码可有多个编码(如OPD长度/目的OPD位置/寻址方式位)

③各指令的信息仅一种取值时可缺省编码(如OPD长度/AF，OP可指明)

指令格式设计 (Cont'd)

- **要求**
  - ①操作码在前、地址码在后 ←利于正交性(如取指时取首字)
  - ②地址码个数≤结构规定 (否则用隐式寻址/简化指令功能)
  - ③指令字长＝k*MEM编址单位 ←利于指令寻址
  - ④预留部分OP编码 ←利于兼容性

Part I. 为什么要学习指令系统?

Part II. 什么是指令系统?

Part III. 指令系统具体设计

指令功能设计→数据表示设计→数据地址设计→ISA结构设计→**指令格式设计与优化**

- CISC
- RISC

- 格式参数
- 操作处理

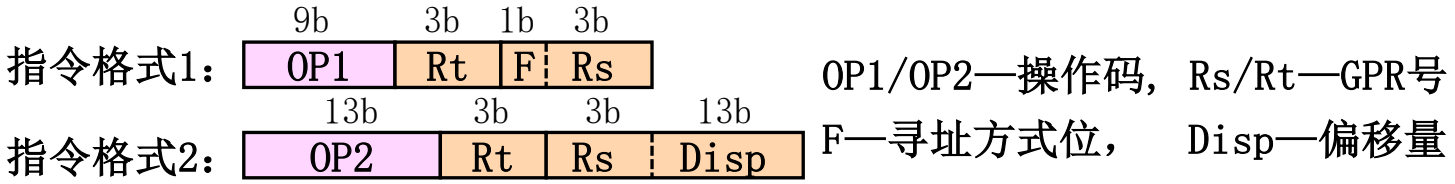
- 编址方式
  - 单位、空间、方式
- 寻址方式

- 堆栈型
- 累加器型
- 通用寄存器型

- 操作码设计
  - 扩展编码
- 指令格式设计

指令格式设计 (Cont'd)

例1：某ISA的MEM按字节编址、地址空间为32位，OPD可放在GPR、MEM中，指令中M_OPD个数≤1个，指令格式如图：



OP1/OP2—操作码，Rs/Rt—GPR号
F—寻址方式位，Disp—偏移量

(1)采用的是哪种指令集结构？显式OPD个数最多是多少？

(2)格式1中<F,Rs>支持哪些寻址方式？

(3)GPR个数、长度分别是多少？

(4)格式2中，某指令支持8/16位OPD，<Rs,Disp>支持基址寻址和变址寻址方式，目的OPD为Rt，该指令需占用几个操作码？

(5)格式1中，若去除F、将OP1改为10b，好处是什么？

答：(1)R-M型GPR结构，≤2个；

(2)REG寻址、REG间接寻址

(3)8个，32位；

(4)2种$_{OPD}$×2种$_{寻址方式}$ = 4个

(5)增加了格式1指令数(部分指令仅REG寻址)

指令格式设计 (Cont'd)

例2：某CISC中，寄存器有8个32位GPR (可用作16位GPR)和8个32位FPR，存储器按字节编址、地址空间为32位，数据表示、寻址方式、指令集功能如下表所示，指令中显式OPD≤2个，请设计指令格式。

数据表示
定点(8/16/32b)
浮点(S/D)
寻址方式
①立即(8b)
②寄存器
③寄存器间接
④基址(8b偏移)

指令名	功能	OPD寻址	op频率
ADD/SUB	Rd←(Rd) op OPD，　OPD=16/32b	-,①②	2种op*0.05
INC等	Rd←(Rd) op 常数，　OPD=32b	-,①常数	4种op*0.05
AND/OR等	Rd←(Rd) op OPD，　OPD=16/32b	-,①②	6种op*0.002
LSL等移位	Rd←(Rd) op (R20)，OPD=16/32b	-,②固定	4种op*0.002
FADD/FSUB等	Fd←(Fd) op (Fs)，　OPD=S	-,②	13种op*0.01
LD	Rd←OPD，　　OPD=8/16/32b	-,③④	1种op*0.05
ST	OPD←(Rd)，　　OPD=8/16/32b	③④,-	1种op*0.05

操作码设计：

需编码种类数：　(操作类型+格式信息)

格式信息隐含表示(个数/位置)，寻址方式位放在地址码中；

0.05频率有2*2+4*1+1*3+1*3=14种，其余频率为13种、20种

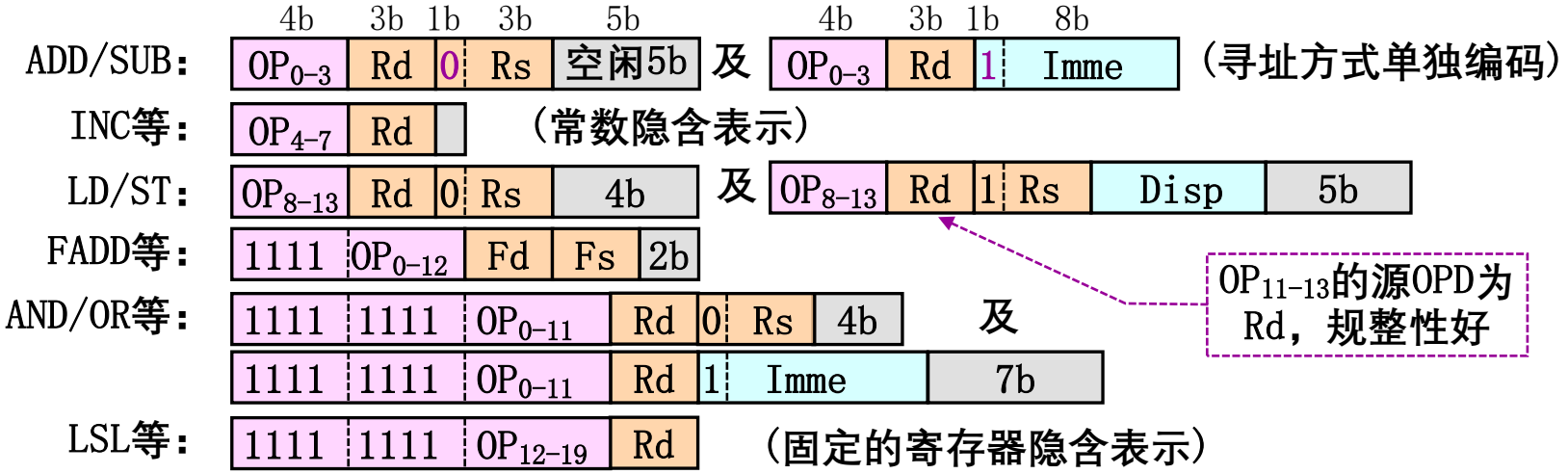
编码方法：

采用扩展编码(不等长横向扩展)，分别占4位、8位、13位

数据表示
定点(8/16/32b)
浮点(S/D)
寻址方式
①立即(8b)
②寄存器
③寄存器间接
④基址(8b偏移)

指令名	功能	OPD寻址	op频率
ADD/SUB	Rd←(Rd) op OPD, OPD=16/32b	-,①②	2种op*0.05
INC等	Rd←(Rd) op 常数, OPD=32b	-,①常数	4种op*0.05
AND/OR等	Rd←(Rd) op OPD, OPD=16/32b	-,①②	6种op*0.002
LSL等移位	Rd←(Rd) op (R20), OPD=16/32b	-,②固定	4种op*0.002
FADD/FSUB等	Fd←(Fd) op (Fs), OPD=S	-,②	13种op*0.01
LD	Rd←OPD, OPD=8/16/32b	-,③④	1种op*0.05
ST	OPD←(Rd), OPD=8/16/32b	③④,-	1种op*0.05

指令格式设计： **字长=k*8b，寻址方式位∈地址码，目的OPD靠左**



指令格式优化： **INC、FADD的空闲位存放部分OP码；** ←减少主OP码

AND/OR的寻址方式位移入$OP_{0\text{-}11}$ ←提高存储和译码效率

Part I. 为什么要学习指令系统?

Part II. 什么是指令系统?

Part III. 指令系统具体设计

ISA结构设计→数据表示设计→指令功能设计→寻址方式设计→**指令格式设计与优化**

- 堆栈型
- 累加器型
- 通用寄存器型

- 格式参数
- 操作处理

- CISC
- RISC

- 编址方式
  - 单位、空间、方式
- 寻址方式

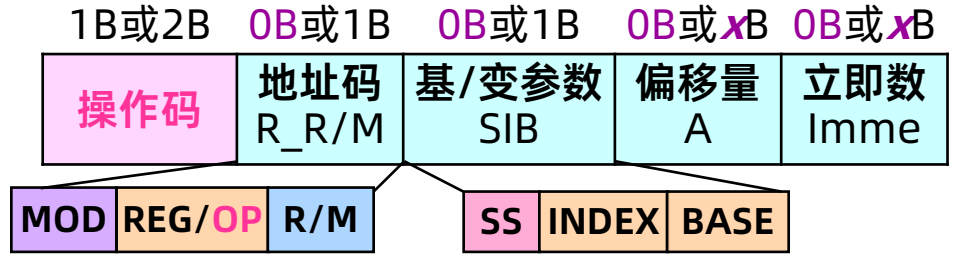
- 操作码设计
  - 扩展编码
- 指令格式设计

Part I. 为什么要学习指令系统?
Part II. 什么是指令系统?
Part III. 指令系统具体设计
Part IV. 如何设计指令系统?

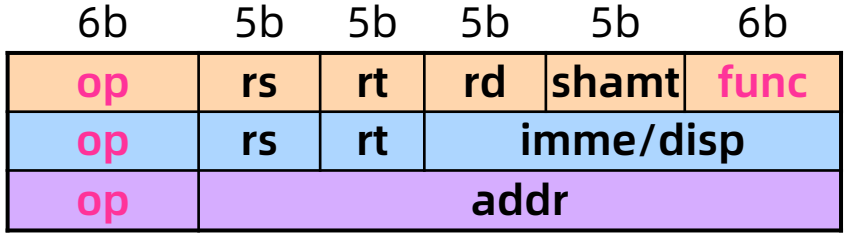
基本原则→性能指标→基本过程

指令系统所含内容

- **指令系统结构（也是ISA结构）**
  - 类型(OPD存放部件)，指令字结构(长度/地址码个数)
- **数据表示**
  - 支持的数据类型(格式/参数)
- **指令功能**
  - 支持的操作类型(操作功能+数据类型)
- **寻址方式**
  - 支持的寻址单元及范围（编址方式/地址空间）、地址形成规则（寻址方式）
- **指令格式**
  - 指令信息的表示(格式/编码)



IA32的指令格式



MIPS32的指令格式

ARM Instruction Set (Cont'd)

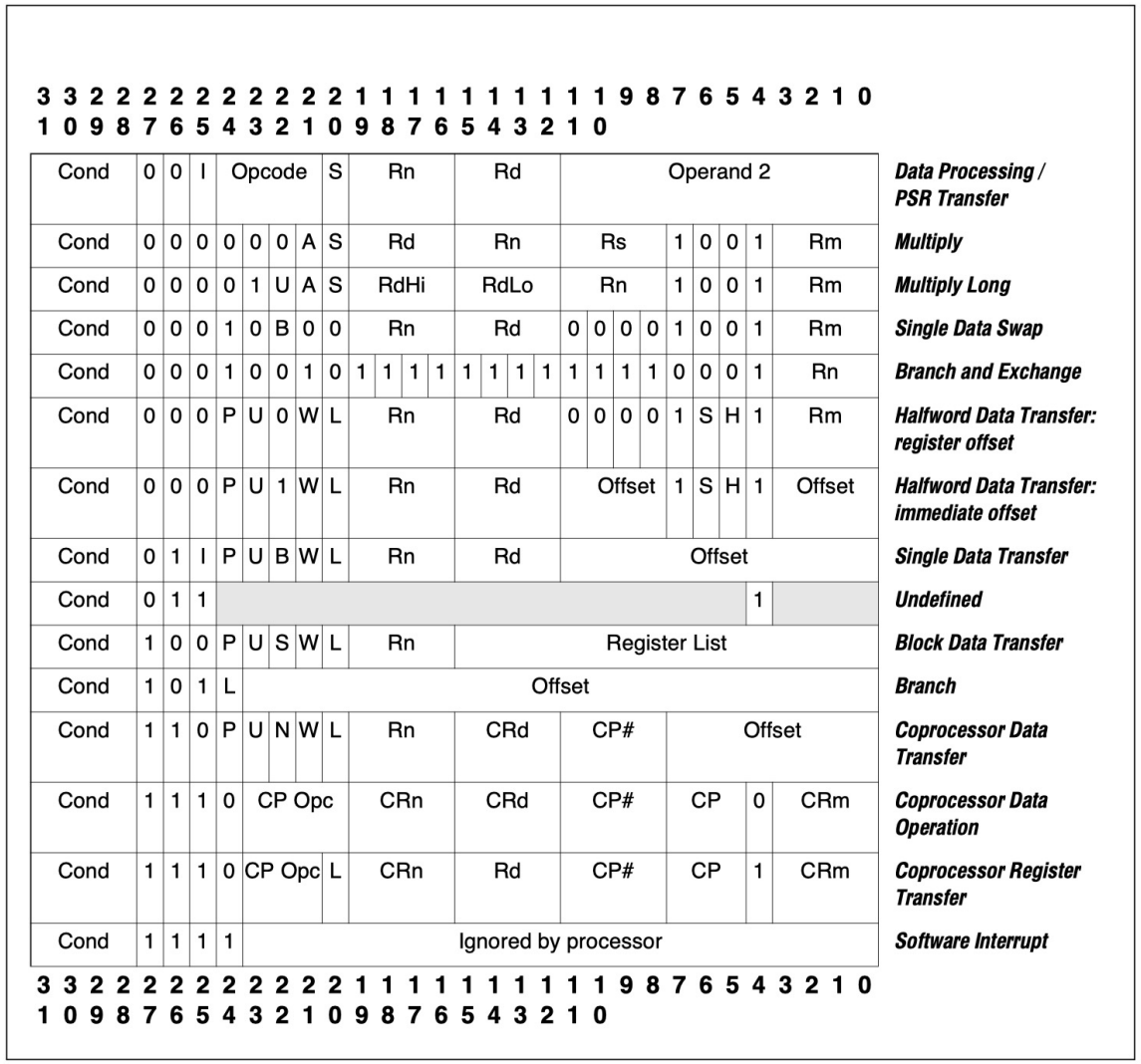


Figure 4-1: ARM instruction set formats

指令系统的设计基本原则

- **完整性**
  - 指令系统的功能齐全、使用方便
    - 如不支持NOT指令[频率较低]时，需支持NOR、恒0寄存器
- **规整性**
  - 相似的OP、OPD等有相同的约定 (如OP编码、REG使用)
- **正交性**
  - 指令字中不同字段相互独立 (如OP/Addr字段)
- **兼容性**
  - 预留部分操作码，便于IS扩充 (如系列机)

ARM Instruction Set

https://blog.csdn.net/Mculover666/article/details/117323244

Part I. 为什么要学习指令系统?
Part II. 什么是指令系统?
Part III. 指令系统具体设计
Part IV. 如何设计指令系统?

基本原则→**性能指标**→基本过程

指令系统的性能指标

- **性能指标：**指令系统对软件&硬件的支持程度
  - 包括程序的代码效率、执行效率

- **提高代码效率：**减少指令条数及指令字长
  - 减少指令条数：指令功能强、地址码个数多，寻址方式多、寻址范围大、对称性好等
  - 缩短指令字长：地址码个数少、长度短，冗余空间小等

例1：32位运算频率较高时，若指令系统仅支持8/16位运算，则1个32位运算→2个16位运算指令，代码效率不高(指令条数多)

例2：数组操作频率较高时，若指令系统仅支持基址寻址，不支持基址+变址寻址(如(RB)+(RI))，代码效率不高(地址码较长)

指令系统的性能指标 (Cont'd)

- **执行效率：包括译码时间及执行时间，应尽量少**
  - 减少译码时间—指令格式规整、寻址方式并行识别等
  - 减少执行时间—指令功能弱、OPD访问速度快、地址形成简单

例3：指令格式不规整时，译码速度较慢(需多级译码)

第i种指令格式：

OP_i(7位)	IMME(8位)

第j种指令格式：

OP_j(8位)	IMME(8位)

OP_i(8位)	IMME(8位)
OP_j(8位)	IMME(8位)

例4：寻址方式位前移/部分缺省，可缩短译码时间；

指令字中MEM_OPD较多时，执行时间长(需多次访问MEM)

指令格式：

OP_i	DF_M	A_1	DF_R	A_2

OP_i	DF_M	DF_R	A_1	A_2

$T_{译码} = T_{OP}+T_{DFM}+T_{DFR}$

$T_{译码} = T_{OP}+\max(T_{DFM},T_{DFR})$

$T_{执行} = T_M+T_R+T_{OP}+T_R$或$T_M$，$A_1$为目的OPD时$T_{执行}$较长

- **代码效率与执行效率的矛盾处理：**
  - 采用折中方法，用CPU时间($=I_N*CPI*T_C$)评价

Part I. 为什么要学习指令系统?
Part II. 什么是指令系统?
Part III. 指令系统具体设计
Part IV. 如何设计指令系统?

基本原则→性能指标→**基本过程**

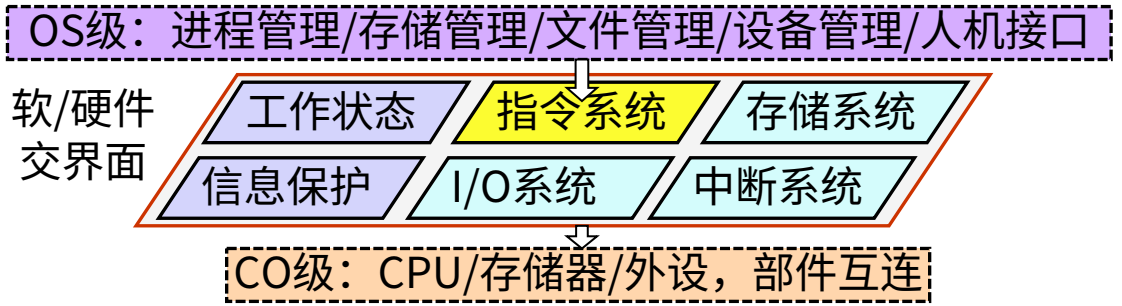
指令系统的设计基本过程

- **指令功能设计**
  - 确定支持的OP类型
- **数据表示设计**
  - 确定支持的数据类型
- **寻址方式设计**
  - 确定部件的编址方式及空间、寻址方式
- **ISA结构设计**
  - 确定ISA结构，包括指令字结构、显式OPD个数
- **指令格式设计**
  - 确定各条指令的格式及编码
- **设计优化**

传统系统结构（即ISA）包含的内容

***概念性结构：**主要为指令系统，涉及硬件子系统、软件管理等

└→程序(指令序列)←┴←结构&原理　└←面向OS



***功能特性：**

指令系统

- **数据表示**— 硬件能直接识别和处理的数据类型
- **寻址方式**— 数据的最小寻址单元、寻址种类及地址计算规则等
- **指令功能**— 指令的数据操作类型、下条指令地址计算方法等
- **指令格式**— 表示指令功能的指令字格式及各信息的编码方式

硬件子系统

- **存储系统**— 主存的编址单位、地址空间，层次结构等
- **I/O系统**— 外设的数据传送控制，I/O操作及状态表示等
- **中断系统**— 中断&异常的类型、响应机制等

软件管理

- **机器工作状态**系统态/用户态的定义与切换
- **信息保护**— 保护方式、保护机构

恭喜大家通关指令系统设计！

我们流水线设计再见！

Copyright: Nintendo, Source: http://news.yzz.cn/domestic/201312-731311.shtml

课后复习思考题与作业

1. 指令系统包含哪些内容？设计目标是什么？设计过程是什么？
2. ISA结构有哪些类型？与结构相关的指令格式参数有哪些？CISC/RISC风格对其有哪些影响？
3. 若指令操作码为8位，存储器按字节编址、地址为64位，寄存器地址为6位，A、B、C、D为内存变量，针对堆栈型、累加器型、R-M风格GPR型、R-R风格GPR型指令系统，写出计算D=A+B-C的指令序列、代码大小。
4. 数据表示与数据结构的关系？数据表示设计的内容、目标、方法？
5. 指令系统功能设计的方法？CISC/RISC风格对功能设计及优化的影响？
6. 数据存储部件可有哪些？确定编址单位的方法？寻址方式设计的内容、目标、方法？
7. x86及MIPS指令系统的基本特征、指令格式的优化方法？

P51-5、思考题-3