计算系统结构

Computer Architecture

计算机与大数据学院
林嘉雯
ljw@fzu.edu.cn

指令是软件要求微处理器执行的一个命令，这里隐含了下列问题：

- 指令要求微处理器处理什么数据？
- 指令要求微处理器对数据进行什么处理？
- 微处理器怎样才能得到要处理的数据？

这三个问题就是指令的操作数、操作码和寻址方式需要解决的，也是确定指令系统的最基本的要素。（三要素前继课程已学习，本课程不详细介绍）

第 2 章 指令系统的发展和改进

主要内容：

- 指令系统的两个发展方向
- **CISC结构**
- **RISC结构**

指令系统设计的三个阶段：

- **CISC：复杂指令系统**

70-80年代，CISC结构的微处理器为市场的主流

- **RISC：精简指令系统**

80-90年代末期，RICS结构的处理器得到发展，大量出现在高性能的计算机中

- **后RISC时期**

RISC和CISC互相借鉴继续发展，出现了新型的结构

指令系统的两大发展方向： CISC、RISC

2.1 CISC结构

2.1.1 CISC的设计思想概述

复杂指令系统计算机

CISC（Complex Instruction Set Computer）

试图用数目少、功能强的指令来完成程序的任务

- 设计方法： 用一条指令代替一串指令

- 增加新的指令
- 增强指令的功能，设置功能复杂的指令
- 增加寻址方式
- 增加数据的表现形式

CISC实现了软件功能的硬化

□特点：

- 指令条数多
- 指令功能强
- 寻址方式多
- 执行时间长
- CPU控制电路复杂

2.1.2 CISC的主要改进方向

1. 面向目标程序的优化实现改进

- 通过统计已有机器语言程序及其执行过程中指令和指令串的使用频度加以分析改进

  遵循哈夫曼压缩原理

- 增设强功能符合指令

2. 面向高级语言和编译的优化实现改进

- 通过统计高级语言语句的使用频度加以改进分析
- 优化代码生成来改进
- 缩小语义差距
- 面向不同高级程序语言自寻优
- 发展高级语言计算机

3. 面向操作系统的优化实现改进

- 统计操作系统中指令和指令串的使用频度加以改进分析
- 增设专用于操作系统的新指令
- 频繁使用的子程序硬化或固化
- 发展专门的处理机

2.1.3 CISC存在的问题

1. 指令系统庞大

- 指令的条数多，功能复杂
- 寻址方式多，指令的格式多，字节数长
- 指令分析器复杂，面积增大，成本提高
- 芯片设计周期长，成功率低，可靠性降低

2. 指令操作繁杂，速度低

功能复杂，寻址方式多，指令的格式多，字节数长都影响 **CPU** 指令执行的速度

3. 指令系统庞大，使高级语言编译程序生成目标程序的难度大

- 编译程序选择的目标范围大
- 很难生成真正高效 的机器语言程序
- 编译生成的目标程序长度长

4. 指令使用的频度差别很大, 许多指令利用率很低

增加设计人员 负担，降低了系统性能价格比

20%-80%规律

仅**20%**的指令被反复使用，在总程序数量中占**80%**

有**80%**的指令较少被使用，在总程序数量中占**20%**

Intel8088 处理机指令系统使用频度和执行时间统计

（C 语言编译程序和 PROLOG 解释程序）

使用频度				执行时间			
序号	指令	%	累计%	序号	指令	%	累%
1	MOV	24.85	24.85	1	IMUL	19.55	19.55
2	PUSH	10.36	35.21	2	MOV	17.44	36.99
3	CMP	10.28	45.49	3	PUSH	11.11	48.10
4	JMPcc	9.03	54.52	4	JMPcc	10.55	58.65
5	ADD	6.80	61.32	5	CMP	7.80	66.45
6	POP	4.14	65.46	6	CALL	7.27	73.72
7	RET	3.92	69.38	7	RET	4.85	78.57
8	CALL	3.89	73.27	8	ADD	3.27	81.84
9	JUMP	2.70	75.97	9	JMP	3.26	85.10
10	SUB	2.43	78.40	10	LES	2.83	87.93
11	INC	2.37	80.77	11	POP	2.61	90.54
12	LES	1.98	82.75	12	DEC	1.49	92.03
13	REPN	1.92	84.67	13	SUB	1.18	93.21
14	IMUL	1.69	86.36	14	XOR	1.04	94.25
15	DEC	1.37	87.73	15	INC	0.99	95.24
16	XOR	1.13	88.86	16	LOOPcc	0.64	95.88
17	REPNZ	0.78	89.64	17	LDS	0.64	96.52
18	CLD	0.54	90.18	18	CMPS	0.44	96.96
19	LOOPcc	0.52	90.70	19	MOVS	0.39	97.35
20	TEST	0.40	91.10	20	JCXZ	0.37	97.72

问题：

花费如此巨大的硬件成本去实现一些极少使用的指令是否合算？

对软硬件的功能分配需要重新思考

——提出RISC思想

2.2 RISC结构

精简指令系统计算机

RISC（Reduced Instruction Set Computer）

通过减少指令种数和简化指令功能来降低硬件设计的复杂度，提高指令的执行速度

- 只保留功能简单的指令，对复杂的不常用指令进行精简
- 功能较复杂的指令改由软件实现
- 设置大量寄存器
- 提高流水线效率

RISC实现了硬件功能的简化

2.2.1 RISC设计的基本原则

① 选择使用频度较高、最有用，及实现简单的指令；

② 每条指令都在一个机器周期内完成的指令；

③ 减少指令寻址方式的种类，简化指令格式，使指令的长度相同；

④ 增加通用寄存器的数量，减少访问存储器操作；

⑤ 大量采用硬联控制，提高执行速度

⑥ 通过优化和精简指令设计支持的编译程序，能有效地为高级语言生成机器语言程序。

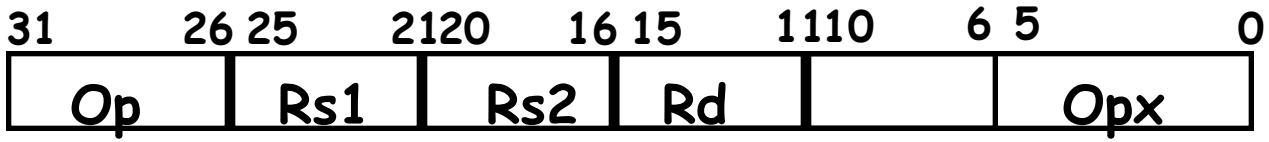
A "Typical" RISC

- 32-bit fixed format instruction (3 formats)
- 32 32-bit GPR (R0 contains zero, DP take pair)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store:
  base + displacement
  - no indirection
- Simple branch conditions
- Delayed branch

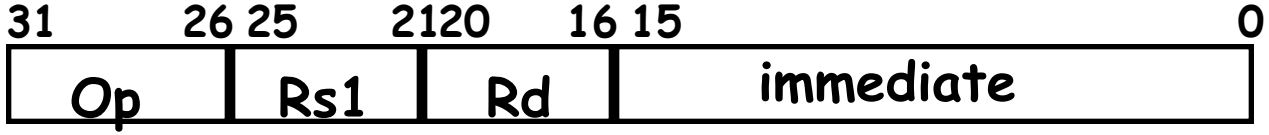
see: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC,
CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

Example: MIPS (- DLX)

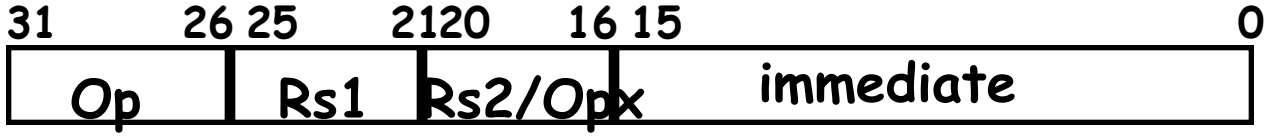
Register-Register



Register-Immediate



Branch



Jump / Call

2.2.2 RISC结构采用的基本技术

1. 遵循按RISC机器一般原则设计的技术。
2. 在逻辑上采用硬联实现和微程序固件实现相结合的技术。

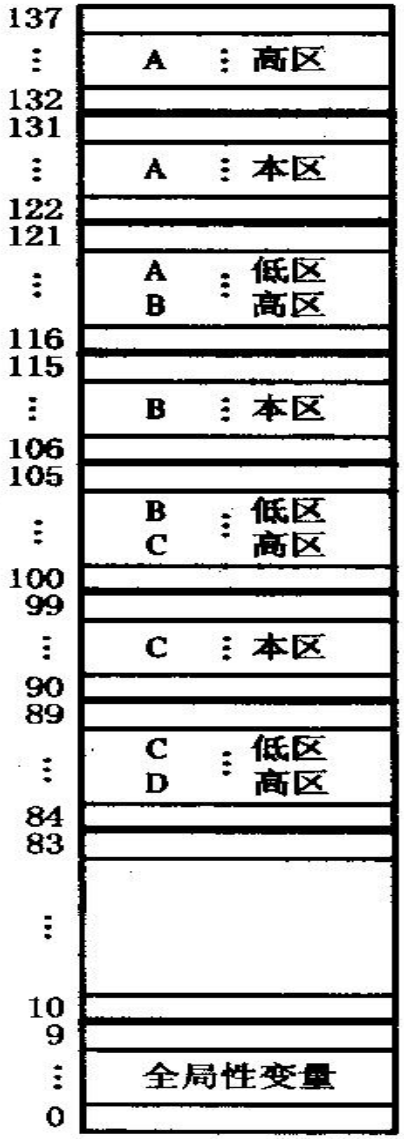
大多数简单指令用硬连线方式实现，功能较复杂的指令用微程序解释实现。

3. 在CPU中设置数量较大的寄存器组， 并采用重叠寄存器窗口的技术。

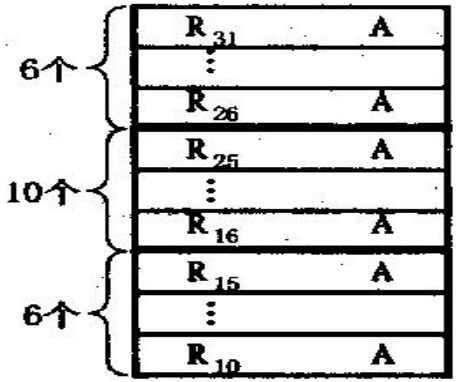
重叠寄存器窗口技术

（Overlapping Register Window）

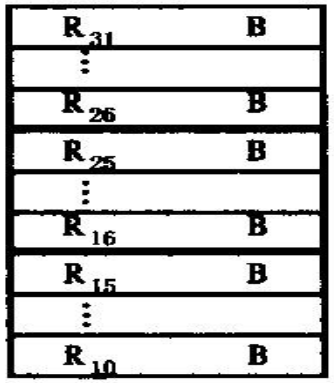
- 美国加洲伯克利分校F Baskett 提出
- 实现方法：设置一个数量比较大的寄存器堆，并把它分成很多个窗口。在每个程序使用的几个窗口中：
  - 有一个窗口是与前一个程序共用
  - 有一个窗口是与后一个程序共用



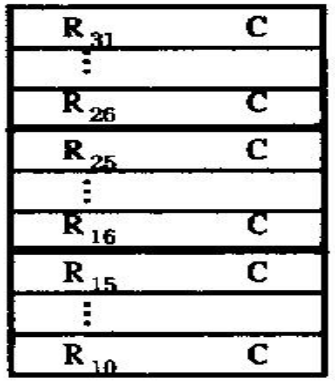
CPU的工作寄存器组



A过程看到的寄存器窗口



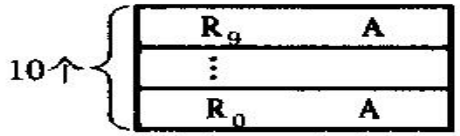
B过程看到的寄存器窗口

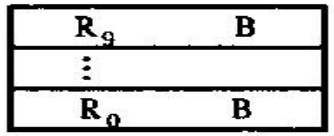


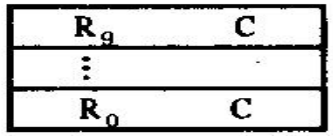
C过程看到的寄存器窗口

每个过程可访问：

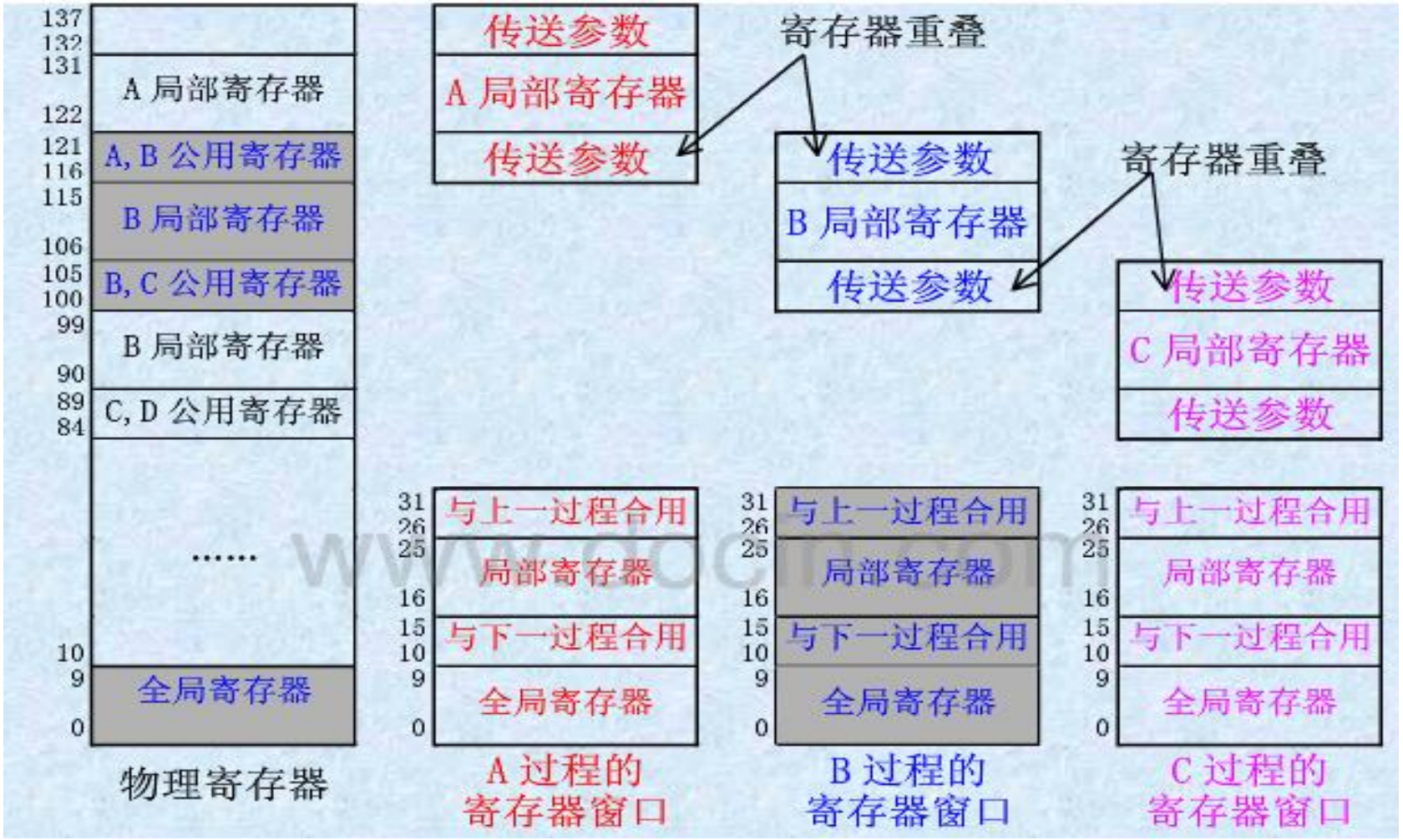
- 1个公共窗口（全局）
- 1个私有窗口（本区）
- 2个重叠窗口（高区与父进程重叠、低区与子进程重叠）







RISC Ⅱ的重叠寄存器窗口



每个过程可利用：

- 公共窗口存放全局变量
- 私有窗口存放局部变量
- 重叠窗口实现与父进程或子进程间的参数传递

- 效果：大量减少了原先由于程序调用引起的访存的次数

- 在主存中开辟堆栈，当调用的进程数（层数）超过规定层数时（寄存器溢出），则将溢出的部分压入堆栈中

- SUN公司的SPARC、SuperSPARC等处理机，将最后一个过程与第一个过程的公用存储器重叠起来，从而形成一个循环圈

4. 指令的执行采用流水和延迟转移技术

- 由于采用固定字长指令，每条指令都在一个机器周期内执行完成，能够充分发挥流水线处理的功能

- 在指令执行中，如果遇到条件转移指令，可能会影响流水线的执行，后一条流入的指令要作废。为了使流水线不断流，在转移指令后插入一条没有数据相关、控制相关的指令，而转移指令被延迟执行，这种技术为延迟转移技术。

  - 可对原指令序列进行调整
  - 可插入一条新的无关指令

无条件转移指令的延迟执行

判断转移需要一个指令周期，则可将不影响最终结果或与转移无关的必须执行指令移动

有条件转移指令的延迟执行

➢调整前的指令序列：

```
1：MOVE R1, R2
2：CMP  R3, R4      ;(R3)与(R4)比较
3：BEQ  EXIT        ;如果(R3)=(R4)则转移
.........
NEXT: MOVE R4, A
```

移动指令实现转移要注意：

●被移动指令在移动中与所经过指令没有数据相关

●被移动指令不破坏条件码，至少不影响后面的指令使用条件

若找不到满足上述两个要求的指令时，则必须在条件转移指令后插入空操作。

例如：设X、Y为主存单元，R_d, R_0, R_b, R_c为寄存器单元，且R_0中放值 0。有一个未采用延迟转移的程序：

指令地址	指　令	功　能
210	取 X，R_d	(X)→R_d
211	加 R_d，#1，R_d	(R_d)+1→R_d
212	条转〈条件〉，215	条件满足转 215，否则执行 213
213	加 R_d，R_0，R_b	(R_d)→R_b
214	减 R_b，R_c，R_b	(R_b)−(R_c)→R_b
215	存 R_d，Y	(R_d)→Y
216	……	

调整后程序如下：

```
210   取 X，Rd
211   加 Rd，#1，Rd
212   条转〈条件〉，216
213   加 R0，R0，R0
214   加 Rd，R0，Rb
215   减 Rb，Rc，Rb
216   存 Rd，Y
```

采用指令延迟技术时，对指令序列的改动和调整，应由编译器自动进行，用户无需干预

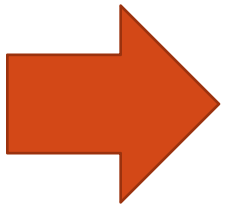
5. 指令流调整技术

- 目标：通过变量重新命名消除数据相关，提高流水线效率

例如

调整后，指令序列的执行速度快了1倍

```
ADD   R1, R2, R3
ADD   R3, R4, R5
MUL   R6, R7, R3
MUL   R3, R8, R9
```



```
ADD   R1, R2, R3
MUL   R6, R7, R0
ADD   R3, R4, R5
MUL   R0, R8, R9
```

调整前　　调整后

例如：设A、 A+1, B, B+1 为主存单元，则程序

取A， R_a ；(A)→R_a

存R_a, B ；(R_a)→B

取A+1, R_a ；(A+1)→R_a

存R_a, B+1 ；(R_a)→B+1

指令之间实际上不能流水

如果通过编译调整其指令的顺序为

取A，　R_a　　；(A)→R_a

取A+1, R_b　　；(A+1)→R_b

存R_a, B　　；(R_a)→B

存R_b, B+1　　；(R_b)→B+1

4条指令均可进行流水操作

采用指令流调整技术时，对指令序列的改动和调整，应由编译器自动进行，用户无需干预

延迟转移技术和指令流调整技术实际上都是优化编译系统的典型例子，设计高质量的编译程序是RISC系统设计的关键之一。

6. 采用高速缓冲存储器Cache

分别设置指令Cache和数据Cache，分别存放指令和数据。保证向指令流水线不间断地输送指令和存取数据，提高流水线效率

2.2.3 RISC技术的性能评价

1. 采用RISC结构的好处

① 简化指令系统设计，适合超大规模电路实现。

② 提高机器的执行速度和效率。

③ 降低设计成本，提高了系统的可靠性。

④ 直接支持高级语言的能力，简化编译程序的设计。

2. RISC结构存在的不足

① 指令的条数减少，使原来在 CISC 上由单一指令完成的功能需要多条 RISC 的指令才能完成，这样增加了汇编语言、机器语言的条数，加大了信息数量。

② 采用 RISC 的指令编写的程序，占用了更大的存储空间。

③ RISC 机器上的编译程序要比 CISC 机器更难写。

为什么采用 RISC 的机器执行程序的速度比 CISC 的快 ？

程序执行时间：

$$P = I * CPI * T$$

P	执行程序段所使用的总的时间
I	程序执行所需的总指令数
CPI	每条指令执行的平均周期数
T	一个周期的时间长度

CISC 与 RISC 的运算速度比较

类 型	指令条数 I	指令平均周期数 CPI	周期时间 T
CISC	1	2～15	33ns～5ns
RISC	1.3～1.4	1.1～1.4	10ns～2ns

- 同类问题，CISC体系下程序比RISC短30%—40%
- CPI，RISC比CISC少2—10倍
- 通常情况下，RISC的时钟周期短于CISC

RISC的速度比CISC要快三倍左右，关键在于CPI

2.3 RISC的发展——后RISC时代

由于RISC也存在不足和问题，使得在设计CPU时，向着RISC和CISC结合，取长补短的方向发展。

在商品化的微处理器中，越来越多的非RISC特性添加到了RISC微处理器上。我们将加入的非RISC特性称为后RISC

随着芯片面积的增大及集成度的提高，多数微型芯片设计开始加入了如下的功能：

(1)更多的寄存器数量

(2)更大更快的片上Cache

(4)更多的功能部件

(5)流水线深度加深

(6)增设不少高速CISC类型的指令；支持媒体和网络的发展，出现了指令集的扩展

后RISC时代出现了一些新的体系结构

1、 VLIW（Very Long Instruction Word）

超长指令字

- 什么是**VLIW**

◆ 一种显示指令级并行指令系统

◆ 在一条VLIW指令中包含有多个相同或者不同的操作字段（每个操作字段的功能相当于一般处理机中得一条指令）

◆ 每个操作字段能独立控制各自的功能部件同时工作

◆ 二维程序结构

◆ 指令并行度高

●VLIW的特点

- ◆ 采用显示并行指令计算方法

二维指令矩阵的每一行的所有操作构成一条超长指令，操作间无数据相关、控制相关和功能部件冲突

- ◆ 指令并行度高

多数VLIW并行度在4—8间，甚至达到几十

- ◆ 硬件结构规整、简单
- ◆ 编译器实现难度大

VLIW的编译器主要依靠指令级并行算法、数据相关性分析算法、寄存器分配算法及并行编译技术来支持显示的指令级并行性。

2、EPIC（Explicitly Parallel Instruction Computing）显式并行指令计算

Intel和HP联手设计了能够运行现存的Intel80x86和HP PA-RISC软件的新型芯片，导致了IA64体系结构的诞生。EPIC即采用的关键技术。

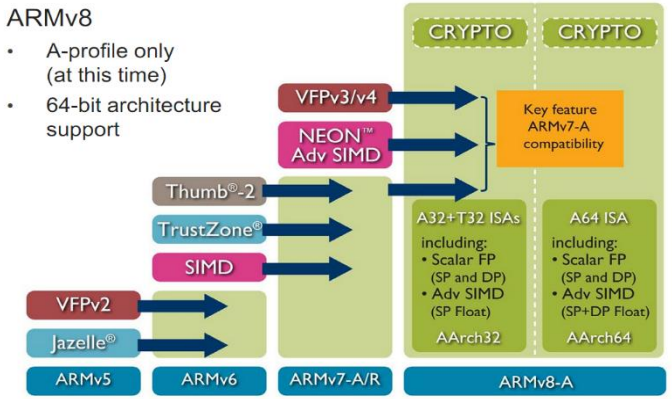
3、单芯片多处理器

在开发指令级并行的同时，有效地开发应用程序中粗粒度的并行性，从而提高芯片的计算性能，达到高效利用VLSI资源的目的。

指令集：RISC vs CISC

- ARM：使用精简指令集（RISC），大幅简化架构，仅保留所需要的指令，可以让整个处理器更为简化，拥有小体积、高效能的特性；ARMv8架构支持64位操作，指令32位，寄存器64位，寻址能力64位；指令集使用NEON扩展结构；

- X86：使用复杂指令集（CISC），以增加处理器本身复杂度为代价，换取更高的性能；X86指令集从MMX，发展到了SSE，AVX；



ARMv8架构的特点：

- 31个64位通用寄存器，原来架构只有15个通用寄存器；
- 新指令集支持64位运算，指令中的寄存器编码由4位扩充到5位；
- 新指令集仍然是32位，减少了条件执行指令，条件执行指令的4位编码释放出来用于寄存器编码；
- 堆栈指针SP和程序指针PC都不再是通用寄存器了，同时推出了零值寄存器（类似PowerPC的r0）；
- A64与A32的高级SIMD和FP相同；
- 高级SIMD与VFP共享浮点寄存器，支持128位宽的vector；
- 新增加解密指令。

- ARM体系结构的指令集（Instruction Set）：
  - **ARM指令集：** 标准arm指令集，32位长度。使用最少指令完成工作。同等条件下，执行速度最快，代价是占用更多存储空间。
  - **Thumb指令集：** 增加了16位长度，指令编码更短，功能更简单。提高编码密度，增加了指令条数，但是存储空间下降
  - **Thumb-2指令集：综合上述两者优势**

ARM指令集

- 指令分类：

指令类型	说明
跳转指令	条件跳转、无条件跳转 (#imm、register)指令
异常产生指令	系统调用类指令 (SVC、HVC、SMC)
系统寄存器指令	读写系统寄存器，如：MRS、MSR指令 可操作PSTATE的位段寄存器
数据处理指令	包括各种算数运算、逻辑运算、位操作、移位 (shift) 指令
load/store内存访问指令	load/store {批量寄存器、单个寄存器、一对寄存器、非-暂存、非特权、独占} 以及load-Acquire、store-Release指令 (A64没有LDM/STM指令)
协处理指令	A64没有协处理器指令

ARM指令集

- A64指令特点：

A64指令编码宽度固定32bit
31个（X0-X30）个64bit通用用途寄存器（用作32bit时是W0-W30），寄存器名使用5bit编码
PC指针不能作为数据处理指或load指令的目的寄存器，X30通常用作LR
移除了批量加载寄存器指令 LDM/STM, PUSH/POP, 使用STP/LDP 一对加载寄存器指令代替
增加支持未对齐的load/store指令立即数偏移寻址，提供非-暂存LDNP/STNP指令，不需要hold数据到cache中
没有提供访问CPSR的单一寄存器，但是提供访问PSTATE的状态域寄存器
相比A32少了很多条件执行指令，只有条件跳转和少数数据处理这类指令才有条件执行.
支持48bit虚拟寻址空间
大部分A64指令都有32/64位两种形式
A64没有协处理器的概念

ARMv8架构有**AArch64、AArch32两种执行状态，前者**支持 A64指令集，后者支持A32和T32