计算机体系结构

周学海
xhzhou@ustc.edu.cn
0551-63606864
中国科学技术大学

review：指令集架构

- **ISA需考虑的问题**
  - Class of ISA
  - Memory addressing
  - Types and sizes of operands
  - Operations
  - Control flow instructions
  - Encoding an ISA
  - ……
- **ISA的类型**
  - 通用寄存器型占主导地位
- **寻址方式**
  - 重要的寻址方式: 偏移寻址方式, 立即数寻址方式, 寄存器间址方式
    - SPEC测试表明，使用频度达到 75%--99%
  - 偏移字段的大小应该在 12 - 16 bits, 可满足75%-99%的需求
  - 立即数字段的大小应该在 8 -16 bits, 可满足50%-80%的需求
- **操作数的类型和大小**
  - 对单字、双字的数据访问具有较高的频率
  - 支持64位双字操作，更具有一般性
- **控制转移类指令**
- **指令编码（指令格式）**

2.1 ISA的基本概念

ISA中需要描述的有关问题：

如何访问操作数

需要支持哪些操作

如何控制指令执行的顺序

指令的编码问题

2.2 ISA的功能设计

2.3 ISA的实现

2.2 ISA的功能设计

功能设计

典型ISA

ISA的功能设计

- **功能设计**
  - 任务：确定硬件支持哪些操作
  - 方法：统计的方法
  - 两种类型：CISC和RISC
- **CISC（Complex Instruction Set Computer）**
  - 目标：**强化指令功能**，减少运行的指令条数，**提高系统性能**
  - 方法：面向目标程序的优化，面向高级语言和编译器的优化
- **RISC（Reduced Instruction Set Computer）**
  - 目标：通过**简化指令系统**，用**高效**的方法**实现最常用的指令**
  - 方法：充分发挥流水线的效率，降低（优化）CPI

典型操作类型

Operator type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, subtract, and, or, multiple, divide
Data transfer	Loads-stores (move instructions on computers with memory addressing)
Control	Branch, jump, procedure call and return, traps
System	Operating system call, virtual memory management instructions
Floating point	Floating-point operations: add, multiply, divide, compare
Decimal	Decimal add, decimal multiply, decimal-to-character conversions
String	String move, string compare, string search
Graphics	Pixel and vertex operations, compression/decompression operations

- **一般计算机都支持前三类所有的操作；**
- **不同计算机系统 对系统支持程度不同，但都支持基本的系统功能。**
- **对最后四类操作的支持程度差别也很大，有些机器不支持，有些机器还在此基础上做一些扩展，这些指令有时作为可选的指令。**

Top 10 80x86 Instructions

° Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

° Simple instructions dominate instruction frequency

CISC计算机ISA的功能设计

- **目标：强化指令功能，减少指令条数，以提高系统性能**
- **基本优化方法**

1. 面向目标程序的优化是提高计算机系统性能最直接方法

- 优化目标
  - 缩短程序的长度
  - 缩短程序的执行时间
- 优化方法
  - 统计分析目标程序执行情况，找出使用频度高，执行时间长的指令或指令串
  - 优化使用频度高的指令
  - 用新的指令代替使用频度高的指令串

优化目标程序的主要途径

1）增强运算型指令的功能

如sin(x), cos(x), SQRT(X)，甚至多项式计算

如用一条三地址指令完成

$P(X) = C(0)+C(1)X+C(2)X^2+C(3)X^3+.....$

2) 增强数据传送类指令的功能

主要是指数据块传送指令

R-R, R-M, M-M之间的数据块传送可有效的支持向量和矩阵运算，如IBM370

R-Stack之间设置数据块传送指令，能够在程序调用和程序中断时，快速保存和恢复程序现场，如 VAX-11

3) 增强程序控制指令的功能

在CISC中，一般均设置了多种程序控制指令。

面向高级语言和编译程序的优化（1/3)

2. 面向高级语言和编译程序改进指令系统

优化目标：主要是缩小HL-ML之间的差距

优化方法：

1）增强面向HL和Compiler支持的指令功能

- 统计分析源程序中各种语句的使用频度和执行时间
- 增强相关指令的功能，优化使用频度高、执行时间长的语句
- 增加专门指令，以缩短目标程序长度，减少目标程序执行时间，缩短编译时间

面向高级语言和编译程序的优化（2/3)

FORTRAN语言和COBOL语言中各种主要语句的使用频度

语言	一元赋值	其他赋值	IF	GOTO	I/O	DO	CALL	其他
FORTRAN	31.0	15.0	11.5	10.5	6.5	4.5	6.0	15.0
COBOL	42.1	7.5	19.1	19.1	8.46	0.17	0.17	3.4

观察结果：

（1）一元赋值在其中比例最大，增强数据传送类指令功能，缩短这类指令的执行时间是对高级语言非常有力的支持，

（2）其他赋值语句中，增1操作比例较大，许多机器都有专门的增1指令

（3）条件转移和无条件转移占22%，38.2%,因此增强转移指令的功能，增加转移指令的种类是必要的

面向高级语言和编译程序的优化（3/3)

2）高级语言计算机系统：缩小HL和ML的差别

极端：HL=ML，即所谓的高级语言计算机

高级语言不需要经过编译，直接由机器硬件来执行

如LISP机，PROLOG机

3）支持操作系统的优化实现：特权指令

指令系统对OS的支持主要包括：

- 处理器工作状态和访问方式的转换
- 进程的管理和切换
- 存储管理和信息保护
- 进程同步和互斥，信号量的管理等

RISC指令集结构的功能设计

- **采用RISC设计理念的微处理器**
  - SUN Microsystem: SPARC, SuperSPARC, Ulta SPARC
  - SGI: MIPS R4000, R5000, R10000,
  - IBM: Power PC
  - Intel: 80860, 80960
  - DEC: Alpha
  - Motorola 88100
  - HP HP300/930系列，950系列
  - ARM，MIPS
  - RISC-V

RISC的定义和特点

- **RISC是一种计算机体系结构的设计思想，它不是一种产品。**
- **RISC是近代计算机体系结构发展史中的一个里程碑**
- **早期对RISC特点的描述**
  - 大多数指令在单周期内完成
  - 采用Load/Store结构
  - 硬布线控制逻辑
  - 减少指令和寻址方式的种类
  - 固定的指令格式
  - 注重代码的优化
- **从目前的发展看，RISC体系结构还应具有如下特点：**
  - 面向寄存器结构
  - 十分重视流水线的执行效率 - 尽量减少断流
  - 重视优化编译技术
- **减少指令平均执行周期数是RISC思想的精华**

问题

RISC的指令系统精简了，CISC中的一条指令可能由一串指令才能完成，那么为什么RISC执行程序的速度比CISC还要快?

ExecuteTime = CPI X IC X T

	IC	CPI	T
CISC	1	2~15	33ns~5ns
RISC	1.3~1.4	1.1~1.4	10ns~2ns

IC：实际统计结果，RISC的IC只比CISC 长30%~40%

CPI: CISC CPI一般在4~6之间，RISC 一般CPI =1，
Load/Store 为2

T: RISC采用硬布线逻辑，指令要完成的功能比较简单

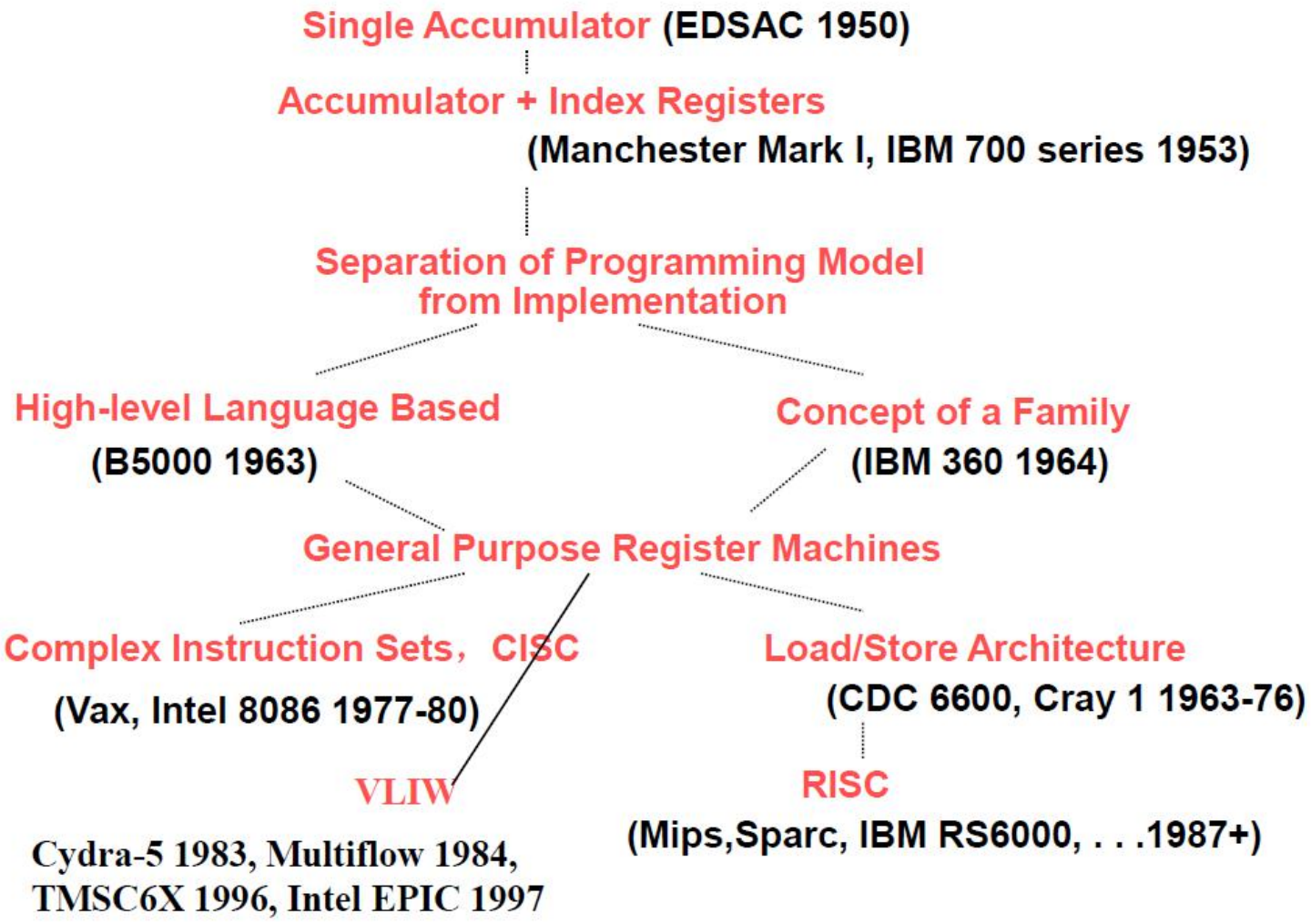
RISC为什么会减少CPI

- **硬件方面：**
  - 硬布线控制逻辑
  - 减少指令和寻址方式的种类
  - 使用固定格式
  - 采用Load/Store
  - 指令执行过程中设置多级流水线。
- **软件方面：十分强调优化编译的作用**

小结

- **ISA的功能设计：**
  - 确定硬件支持哪些操作
  - 设计方法是统计的方法。
- **存在CISC和RISC两种类型**
  - CISC（Complex Instruction Set Computer）
    - 目标：强化指令功能，减少指令的指令条数，以提高系统性能
    - 方法：面向目标程序的优化，面向高级语言和编译器的优化
  - RISC（Reduced Instruction Set Computer）
    - 目标：通过**简化**指令系统，用最**高效**的方法**实现最常用的指令**
    - 手段：充分发挥流水线的效率，降低（优化）CPI

ISA的演进


Single Accumulator (EDSAC 1950)
Accumulator + Index Registers
(Manchester Mark I, IBM 700 series 1953)
Separation of Programming Model
from Implementation
High-level Language Based
(B5000 1963)
Concept of a Family
(IBM 360 1964)
General Purpose Register Machines
Complex Instruction Sets，CISC
(Vax, Intel 8086 1977-80)
Load/Store Architecture
(CDC 6600, Cray 1 1963-76)
VLIW
Cydra-5 1983, Multiflow 1984,
TMSC6X 1996, Intel EPIC 1997
RISC
(Mips,Sparc, IBM RS6000, . . .1987+)

2.2 ISA的功能设计

功能设计

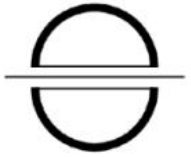
典型ISA

ISA的衡量指标?


成本（美元硬币）


简洁性（轮子）


性能（速度计）


架构和具体实现的分离（分开的两个半圆）


提升空间（手风琴）


程序大小（相对的压迫着一条线的两个箭头）


易于编程/编译/链接（儿童积木“像 ABC 一样简单”）

MIPS

- **MIPS是最典型的RISC 指令集架构**
  - Stanford (1980)提出
  - 第一个商业实现是R2000（1986）
  - 最初的设计中，其整数指令集仅有**58条指令**，直接实现单发射、顺序流水线
  - 30年来，逐步增加到约**400条指令**。
- **主要特征：**
  - Load/Store型结构
  - ALU类指令的操作数来源于寄存器或立即数
  - 降低了ISA和硬件的复杂性，方便了流水线的实现
  - 依赖于优化编译技术

- **主要缺陷：**
  - 针对特定的微体系架构的实现方式（5级流水、单发射、顺序流水线）进行过度的优化设计
    - 延迟转移问题（控制相关）增加了超标量等复杂流水线的实现难度，当无法有效填充延迟槽时会导致代码尺寸变大
    - MIPS-I将其他流水线冲突（数据相关）暴露到ISA中，后来的版本又取消了这种做法，说明硬件采用Interlocking机制，即简单又高效，但为了保持兼容性，仍然保留了延迟转移
  - ISA对位置无关的代码（position-independent code, PIC)支持不足。
    - 直接跳转没有提供PC相对寻址，通过间接跳转方式实现PIC，增加了代码尺寸，降低了性能
    - 2014年MIPS的修订，改进了PC-相对寻址(针对数据)，但仍然要多条指令才能完成
  - 16位位宽立即数消耗了大量编码空间，只有少量的编码空间可供扩展指令
    - 2014修订版，保存有1/64的编码空间供扩展
    - 架构师如果想采用压缩指令编码来降低代码空间，就不得不采用新的指令编码模式
  - 乘除指令使用了特殊的寄存器（HI,LO)，导致上下文切换内容、指令条数、代码尺寸增加，微架构实现复杂

- ISA假设浮点操作部件是一个独立的协处理器，使得单芯片实现无法最优
  - 例如，浮点数转换为整型数结果写到浮点数寄存器，使用结果时，需要额外的mov指令，更糟糕的是浮点数寄存器文件与整型数寄存器文件之间的mov指令，有显式的延迟槽
- 在标准的ABI中，保留两个整型寄存器用于内核程序，减少了用户程序可用的寄存器数
- 使用特殊指令处理未对齐的load和store会消耗大量的操作码空间，并使除了最简单的实现之外的其他实现复杂化。
- 时钟速率/CPI 的权衡使得架构师省略了compare-and-branch指令。随着分支预测和逻辑电路性能的提升，这种权衡在今天已经不太合适了。

- 除了技术方面，MIPS是非开放的专属指令集，不能自由使用

SPARC

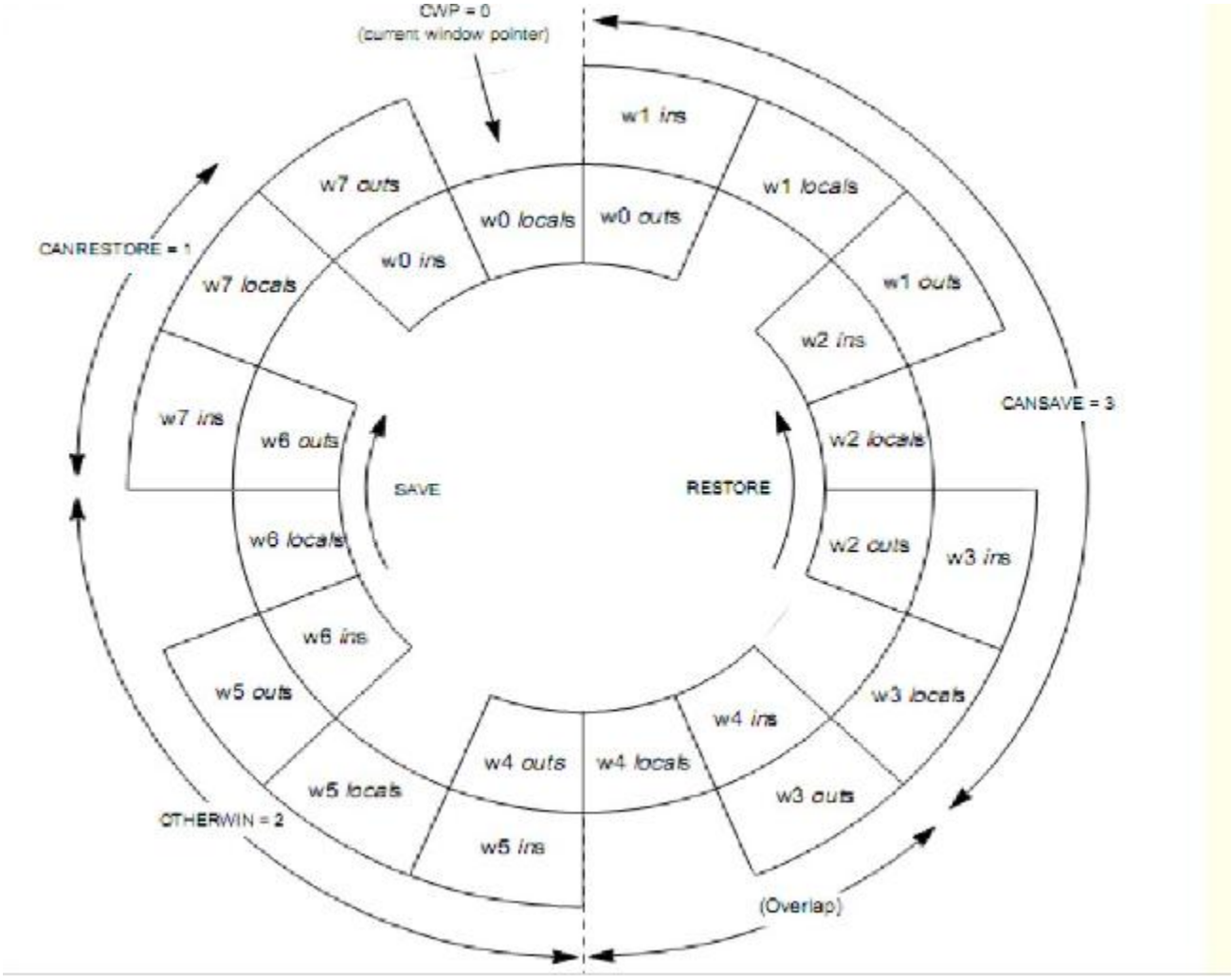
- **Sun Microsystems的专属指令集**
  - 可追溯到Berkeley RISC-I和RISC-II项目； 最近的32位版本的ISA SPARC V8
- **SPARC V8 主要特征**
  - 用户级 整型ISA **90条指令**； 硬件支持IEEE 754-1985标准的浮点数 50条； 特权级指令 **20条**
- **主要问题**
  - SPARC使用了寄存器窗口来加速函数调用
    - 当函数调用所需的栈空间超过了窗口的寄存器数，性能会急剧下降。对于所有的实现来说，寄存器窗口都消耗很大的面积和功耗
  - 分支使用条件码
    - 这些条件码由于在一些指令之间创建了额外的依赖关系，增加了体系结构状态并使实现复杂化
  - load和store**相邻寄存器对**的指令
    - 对于简单的微体系结构很有吸引力，可以在增加很少硬件复杂性的情况下提高吞吐量。
    - 遗憾的是当使用寄存器重命名时使实现复杂化，因为在寄存器文件中数据在物理上可能不再相邻
  - 浮点寄存器文件和整数寄存器文件之间的移动必须使用内存系统作为中介，限制了系统性能

SPARC

- ISA通过体系结构公开的延迟陷阱队列支持非精确浮点异常，该队列向系统监控程序提供信息，以恢复此类异常上的处理器状态
- 唯一的原子内存操作是fetch-and-store，这对于实现许多无等待的数据结构是不够的

- **SPARC与其他80年代RISC结构类似的缺陷**
  - ISA设计面向单发射、顺序、五级流水线的微体系架构；
  - SPARC具有分支延迟插槽和许多显式的数据和控制冲突，这些冲突使代码生成复杂化，无助于更积极的实现；
  - 缺乏位置无关的寻址方式（相对寻址）
  - 由于SPARC缺乏足够的自由编码空间，因此不能方便地对其进行改造以支持压缩ISA扩展

寄存器窗口示例


CWP = 0
(current window pointer)
w1 ins
w1 locals
w1 outs
w2 ins
w2 locals
w2 outs
w3 ins
w3 locals
w3 outs
w4 ins
w4 locals
w4 outs
w5 ins
w5 locals
w5 outs
w6 ins
w6 locals
w6 outs
w7 ins
w7 locals
w7 outs
w0 ins
w0 locals
w0 outs
CANRESTORE = 1
CANSAVE = 3
SAVE
RESTORE
OTHERWIN = 2
(Overlap)

Alpha （DEC）

- **DEC公司的架构师在20世纪90年代初定义了他们的RISC ISA, Alpha**
  - 摒弃了当时非常吸引人的特性，如分支延迟、条件码、寄存器窗口等
  - 创建了64位寻址空间、设计简洁、实现简单、高性能的ISA
  - Alpha架构师仔细地将特权体系结构和硬件平台的大部分细节隔离在抽象接口(特权体系结构库)后面(PALcode)
- **主要问题：DEC对顺序微架构的Alpha进行了过度优化，并添加了一些不太适合现代实现的特性**
  - 为了追求高时钟频率，ISA的原始版本避免了8位和16位的load/store指令，实际上创建了一个字寻址的内存系统。
    - 为了补偿广泛使用这些（非字）操作的应用程序性能，添加了特殊的未对齐的load/store指令以及一些整数指令来加速重新对齐操作。
  - 为了方便长延迟浮点指令的乱序完成，Alpha 有一个非精确的浮点陷阱模型。
    - 这一决定单独来看是可以的。但是ISA还规定：如果使用这种浮点陷阱模型，必须由软件处理例程来给出异常标记和默认值。
  - Alpha缺少整数除法指令，建议使用软件牛顿迭代法实现，导致浮点除法速度高于整数除法

- 与其他前期的RISC一样，没有预先考虑可能的压缩指令集扩展，因此没有足够的操作码空间来进行更新
- ISA包含有条件的mov操作，这使得带有寄存器重命名的微架构复杂化
  - 如果mov条件不满足，指令仍然必须将旧值复制到新的物理目标寄存器中。这实际上使“有条件mov”成为ISA中惟一存取三个源操作数的指令。
  - 实际上，DEC的第一个乱序执行的实现使用了一些技巧来避免该指令的额外数据路径。 Alpha 21264通过将条件mov指令分解为两个微操作来执行，第一个微操作判定mov条件，第二个微操作执行移动。这种方法还要求物理寄存器文件加宽一位以保存中间（判定）结果
- 使用商业Alpha ISAs的一个重要风险:它们可能会被摒弃。
  - 康柏在上世纪90年代末收购了摇摇欲坠的DEC后不久，他们选择逐步淘汰Alpha，转而采用英特尔的安腾架构。
  - 康柏将Alpha的知识产权出售给了英特尔，此后不久，惠普收购了康柏，并在2004年完成了Alpha的最终实现

ARMv7

- **32位 RISC ISA**
  - 使用最广的体系结构
    - 当我们权衡指令集（选择 vs. 重新设计）时，ARMv7是一个自然的选择
    - 大量的软件已经移植到该ISA上，在嵌入式和移动设备中无处不在。
  - 是一个封闭的标准，剪裁或扩充是不允许的，即使是微架构的创新也仅限于那些能够获得ARM所称的架构许可的组织
  - ARMv7十分庞大复杂。整型类指令**600+条**
- **即使知识产权不是问题，它仍然存在一些技术缺陷**
  - 不支持64位地址， ISA缺乏硬件支持IEEE754-2008标准（ ARMv8纠正了这些缺陷）
  - 特权体系结构的细节渗透到用户级体系结构的定义中

- ARMv7附带一个压缩ISA，固定宽度的16位指令，称为Thumb。
  - Thumb虽然提供了有竞争力的代码尺寸，但性能较差
  - 可变长ISA Thumb-2 虽然提供了较高的性能，但32位的Thumb-2编码方式与基本的ISA编码方式不同,16位的Thumb-2的编码方式与Thumb的16位编码方式也不同。导致译码器需要理解三种编码格式，使得能耗、延迟以及设计成本增加
- ISA中包含了许多实现复杂的特性。
  - 程序计数器是可寻址寄存器之一，这意味着几乎任何指令都可以改变控制流。
  - 程序计数器的最低有效位反映ISA当前正在执行(ARM或Thumb)哪个ISA——ADD指令可以更改ISA当前在处理器上执行的指令!
  - 分支使用条件码以及谓词进一步使高性能实现复杂化

ARMv8

- **2011年，ARM发布新的ISA　ARMv8**
  - 64位地址; 扩展了整型寄存器组
  - 摒弃了ARMv7中实现复杂的一些特性
    - PC不再是整数寄存器组的成员；
    - 不再带有有谓词
    - 删除了load-multiple和store-multiple 指令
    - 指令编码归一化
- **主要问题**
  - 使用条件码
  - 存在许多特殊的寄存器
  - 指令集更加厚重：**1070条指令，53种格式，8种寻址方式。说明文档达到了5778页**
  - 以暴露底层实现的方式将用户级ISA和特权级ISA紧密地结合在一起
  - 随着ARMv8的引入，ARM不再支持压缩指令编码
  - **ARMv8也是一个封闭的标准**

OpenRISC

- **OpenRISC项目是一个开放源码处理器设计项目**
  - 来源于Hennessy和Patterson的体系结构教科书 DLX ISA
  - 适用于教学、科研和工业界的实现
- **主要问题**
  - OpenRISC项目主要是开源处理器设计项目，而不是开源的ISA 规格说明，ISA和实现是紧密耦合的
  - 固定的32位编码与16位立即数阻碍了压缩ISA扩展
  - 不支持IEEE 754-2008标准
  - 用于分支和“有条件mov”的条件码使高性能实现复杂化
  - ISA对位置无关的寻址方式支持较弱
  - OpenRISC不利于虚拟化。从异常返回的指令L.RFE，定义为在用户模式下功能，而不是捕获
  - 2010年进展：已经成为可选的，64位版本已经定义(但是，据我们所知，从未实现过)。最终，我们（UCB）认为最好从头开始，而不是相应地修改OpenRISC。

80x86

- Intel 8086架构是过去40年里 笔记本电脑、台式机和服务器市场上最流行的指令集。
  - 除了嵌入式系统领域，几乎所有流行的软件都被移植到x86上，或者是为x86开发的
  - 它受欢迎的原因有很多：该架构在IBM PC诞生之初的偶然可用性; 英特尔专注于二进制兼容性； 它们积极的卓有成效的微结构实现; 以及他们的前沿制造技术
  - **指令集设计质量并不是它流行的原因之一。**
- 主要问题：
  - **1300条指令**，许多寻址方式，很多特殊寄存器，多种地址翻译方式，**从AMD K5微架构开始，所有的Intel支持乱序执行的微结构，都是动态地将x86指令翻译为内部的RISC-风格的指令集。**
  - 不利于虚拟化。因为一些特权指令在用户模式下会**无声地失败**，而不是被捕获。VMware的工程师们用动态二进制翻译软件解决了这一缺陷
  - ISA的指令长度为任意整数字节数，最多为15个字节，数量较少的短操作码已经被随意使用

- ISA有数量极少的寄存器组
- 大多数整数寄存器在ISA中执行特殊功能，加剧了体系结构寄存器的不足
- 更糟糕的是，大多数x86指令只有一种破坏性的指令格式，它会覆盖其中一个源操作数
- 一些ISA特性，包括隐式条件码和带有谓词的mov操作，在微架构中实现复杂

这些ISA决策对静态代码大小有深刻的影响。IA-32仅仅比固定32位的ARM v7编码密度略高，X86-64比ARMv8的编码密度低

尽管存在所有这些缺陷，x86通常比RISC体系结构使用更少的动态指令完成相同的功能，因为x86指令可以完成多个基本操作。

最后，80x86是一个专有指令集

ISA Summary

	MIPS	SPARC	Alpha	ARMv7	ARMv8	OpenRISC	80x86
Free and Open		√				√	
64-bit Address	√	√	√		√	√	√
Compressed Instructions	√			√			Partial
Separate Privileged ISA			√				
Position-Indep. Code	Partial			√	√		√
IEEE 754-2008					√		√
Classically Virtualizable	√	√	√		√		

Acknowledgements

- **These slides contain material developed and copyright by:**
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- **MIT material derived from course 6.823**
- **UCB material derived from course CS252**
- **KFUPM material derived from course COE501、COE502**