



中国科学技术大学

University of Science and Technology of China

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-性能

- **性能度量**
 - 响应时间 (response time)
 - 吞吐率 (Throughput)
- **CPU 执行时间 = IC × CPI × T**
 - CPI (Cycles per Instruction)
- **MIPS = Millions of Instructions Per Second**
- **Latency versus Bandwidth**
 - Latency指单个任务的执行时间, Bandwidth指单位时间完成的任务量 (rate)
 - Latency的提升滞后于带宽的提升 (在过去的30年)
- **Amdahl's Law 用来度量加速比 (speedup)**
 - 性能提升受限于任务中可加速部分所占的比例
 - 应用于多处理器系统的基本假设: 在给定的问题规模下, 研究随着处理器数目的增加性能的变化
- **Benchmarks: 指一组用于测试的程序**
 - 比较计算机系统的性能
 - SPEC benchmark: 针对一组应用综合性能值采用SPEC ratios 的几何平均



review-能耗/功耗

- **给定负载情况下能耗越少，能效越高，特别是对电池供电的移动设备。**
- **功耗已经成为系统设计的重要约束条件之一**
 - A chip might be limited to 120 watts (cooling + power supply)
- **Power Consumed = Dynamic Power + Static Power**
 - 晶体管开和关的切换导致的功耗为动态功耗
 - 由于晶体管静态漏电流导致的功耗称为静态功耗
- **通过降低频率可节省功耗**
- **降低电压可降低功耗和能耗**



第2章 ISA

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



2.1 ISA的基本概念

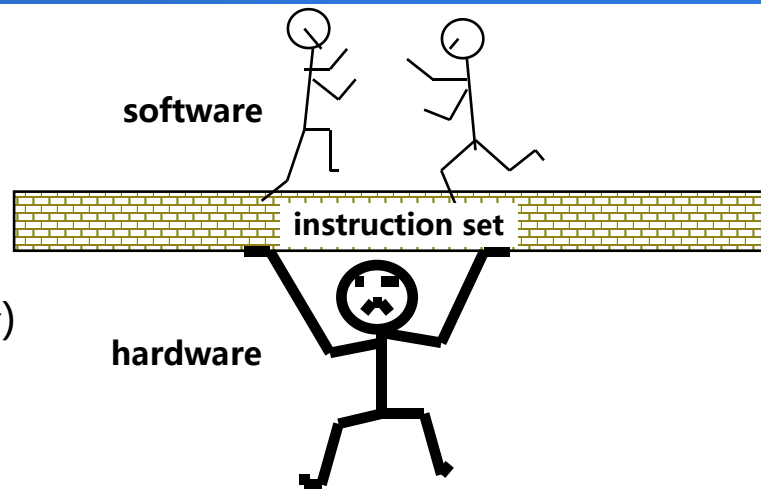
ISA
定义及演进

ISA
基本构成



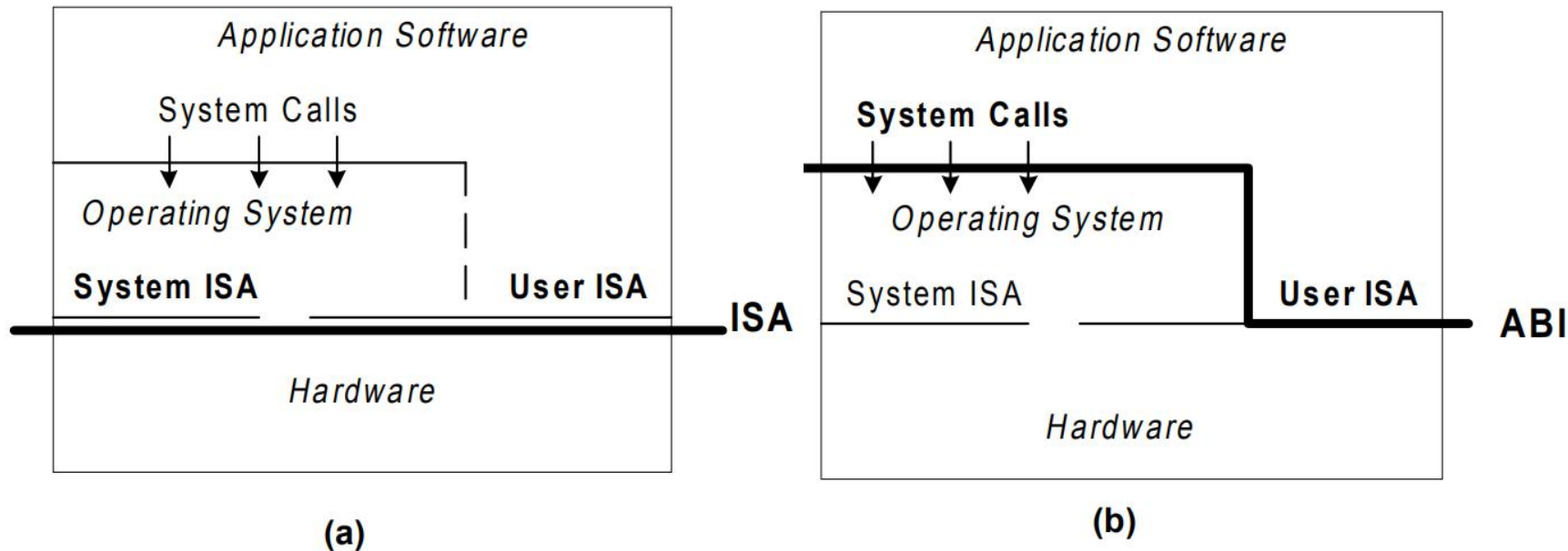
Recap: 指令集架构

- **软件子系统与硬件子系统的关键界面**
- **一组直接由硬件执行的指令，包括**
 - 程序员可见的机器状态
 - 程序员可见的指令集合(操作机器状态的指令)
- **应具备的特性**
 - 成本
 - 简洁性
 - 架构和具体实现分离：可持续多代，以保持向后 (backward) 兼容
 - 可扩展空间：可用于不同应用领域 (desktops, servers, embedded applications)
 - 程序大小：所生成的代码占用空间小
 - 易于编程/编译/链接：为高层软件的设计与开发提供方便的功能
 - 性能：方便低层硬件子系统高效实现
- **IBM 360 是第一个将ISA与其实实现分离的系列机**
 - 给定一个ISA，可以有不同的实现方式；例如AMD/Intel CPU 都是X86-64指令集。ARM ISA 也有不同的实现方式





用户级ISA和特权级ISA



- **重要的系统界面 (System Interface)**
 - ISA界面 (Instruction Set Architecture)
 - ABI界面 (Application Binary Interface)
- **ISA: 用户级ISA+特权级ISA**
 - 用户级ISA 适用于操作系统和应用程序
 - 特权级ISA 适用于硬件资源的管理 (操作系统)

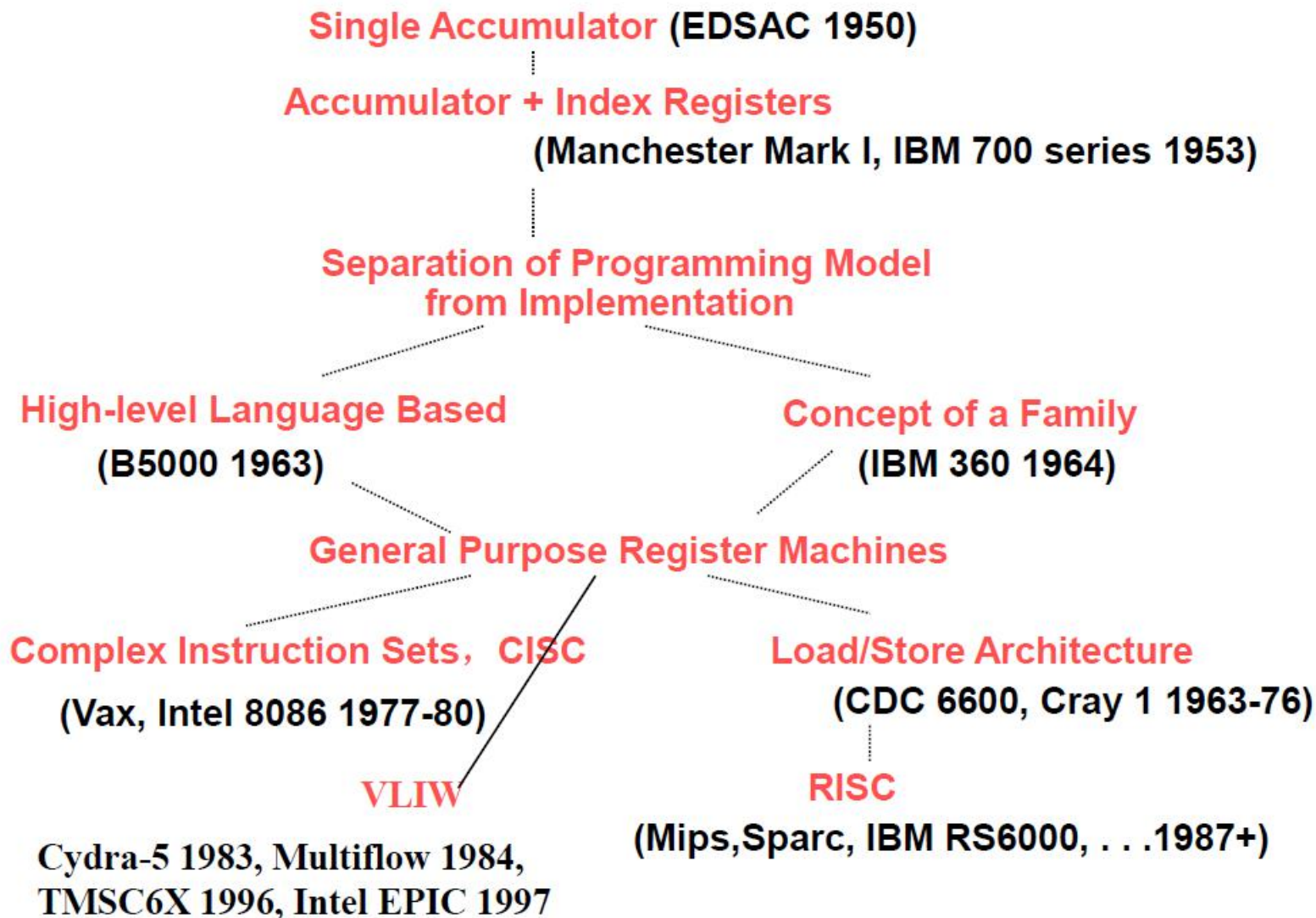


ISA的实现

- **ISA 通常设计时会考虑特定的微体系结构（实现）方式。**
 - Accumulator \Rightarrow hardwired, unpipelined (硬布线、非流水)
 - CISC \Rightarrow microcoded (微程序)
 - RISC \Rightarrow hardwired, pipelined (硬布线、流水线)
 - VLIW \Rightarrow fixed-latency in-order parallel pipelines (固定延时、顺序执行、多条流水线并行)
 - JVM \Rightarrow software interpretation (软件解释)
- **ISA 理论上可以用任何微体系结构（实现）方式**
 - Intel Ivy Bridge: hardwired pipelined CISC (x86) machine (with some microcode support) (硬布线流水化 (部分微程序支持))
 - Spike: Software-interpreted RISC-V machine (模拟器)
 - ARM Jazelle: A hardware JVM processor

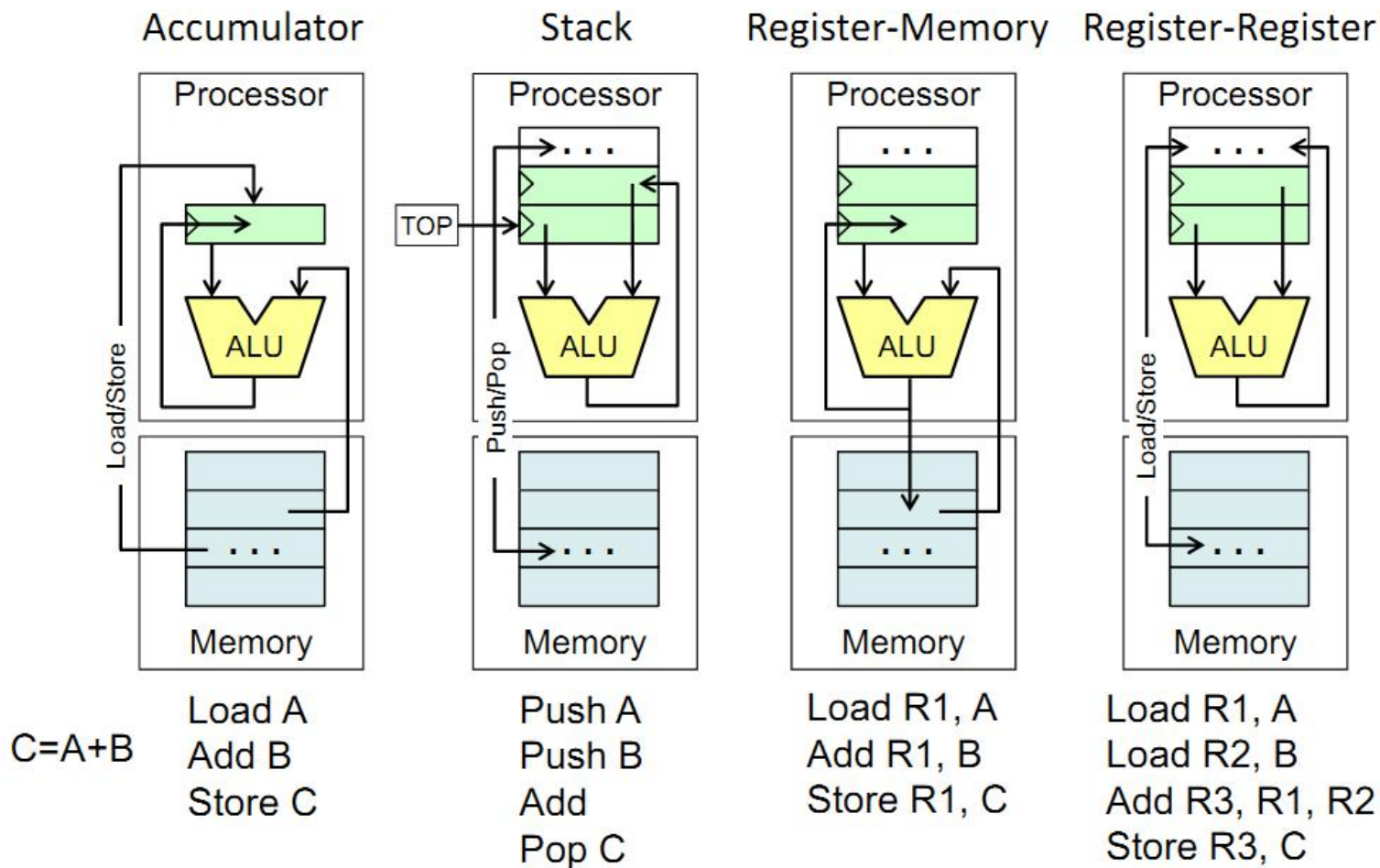


ReCap: ISA的演进





Recap: ISA 的演进





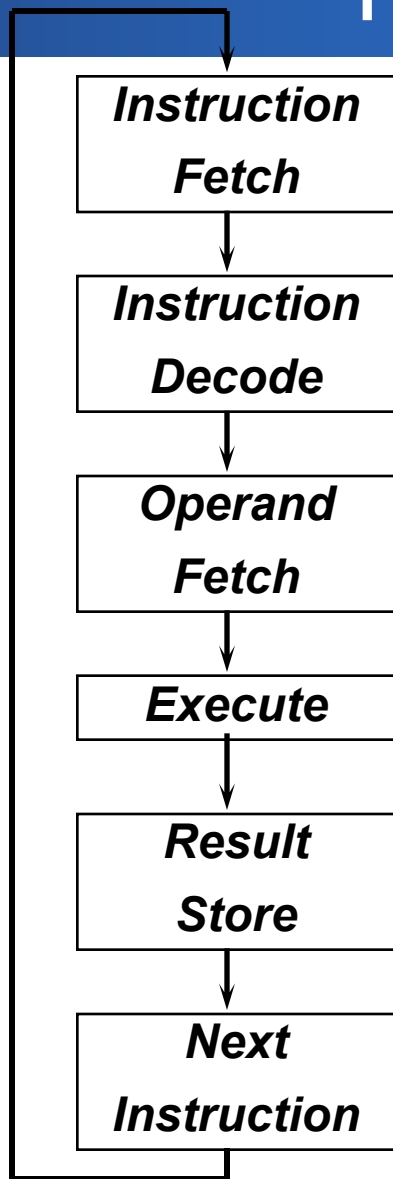
2.1 ISA的基本概念

ISA
定义及演进

ISA
基本构成



Recap: ISA必须说明哪些东西?



- **指令格式或编码方式。即如何编码?**
- **操作数和操作结果的存放位置**
 - 存放位置?
 - 多少个显式操作数?
 - 存储器操作数如何定位?
 - 哪些操作数可以或不可以放到存储器中?
 - 寻址方式
- **数据类型和大小**
- **支持哪些操作**
- **下一条指令地址**
 - jumps, conditions, branches
 - fetch-decode-execute is implicit!



Recap: 有关ISA的若干问题

- 存储器寻址
- 操作数的类型与大小
- 所支持的操作
- 控制转移类指令
- 指令格式

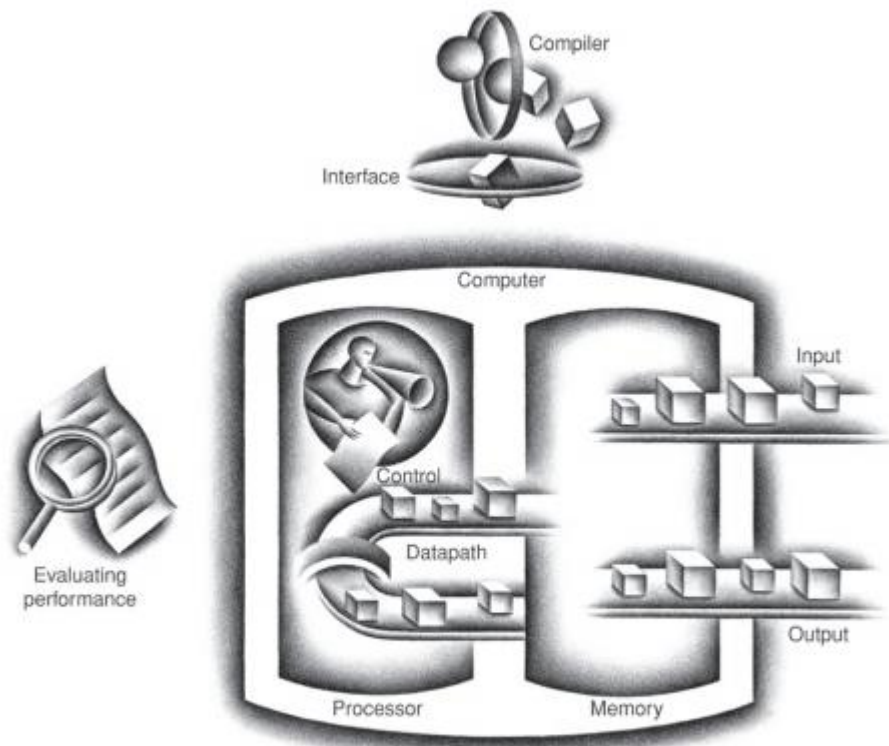


FIGURE 1.5 The organization of a computer, showing the five classic components. The processor gets instructions and data from memory. Input writes data to memory, and output reads data from memory. Control sends the signals that determine the operations of the datapath, memory, input, and output.

操作码 (OPCODE)

若干操作数 (OPRANDS)



Recap: 存储器寻址

- **80年以来几乎所有机器的存储器都是按字节编址**
- **一个存储器地址可以访问：**
 - 一个字节、2个字节、4个字节、更多字节.....
- **不同体系结构对字的定义是不同的**
 - 16位字 (Intel X86) 32位字 (MIPS)
- **如何读32位字，两种方案**
 - 每次一个字节，四次完成；每次一个字，一次完成
- **问题：**
 - (1) 如何将字节地址映射到字地址 (**尾端问题**)
 - (2) 一个字是否可以存放在任何字节边界上 (**对齐问题**)



Recap: 尾端问题

- **little endian, big endian, 在一个字内部的字节顺序问题**

1. **Little Endian** byte ordering

x+3	x+2	x+1	x
Byte 3	Byte 2	Byte 1	Byte 0

 32-bit Register

✧ Memory address X = address of **least-significant** byte (Intel x86)

2. **Big Endian** byte ordering

x	x+1	x+2	x+3
Byte 0	Byte 1	Byte 2	Byte 3

 32-bit Register

✧ Memory address X = address of **most-significant** byte (SPARC)

- **如地址xxx00指定了一个字 (int) , 存储器中从xxx00处连续存放ffff0000, 则有两种方式:**

- Little endian 方式下xxx00位置是字的最低字节, 整数值为0000ffff, Intel 80x86, DEC Vax, DEC Alpha (Windows NT)
- Big endian 方式下xxx00位置是字的最高字节, 整数值为ffff0000, IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA



Recap: 对齐问题

- 对s字节的对象访问地址为A，如果 $A \bmod s = 0$ 称为边界对齐。
- 边界对齐的原因是存储器本身读写的要求，存储器本身读写通常就是边界对齐的，对于不是边界对齐的对象的访问可能要导致存储器的两次访问，然后再拼接出所需要的数。（或发生异常）

Address mod 8	0	1	2	3	4	5	6	7
Byte	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned
2 Bytes	Aligned		Aligned		Aligned		Aligned	
2 Bytes		Misaligned		Misaligned		Misaligned		Misalign
4 Bytes	Aligned				Aligned			
4 Bytes		Misaligned				Misaligned		
4 Bytes				Misaligned			Misaligned	
4 Bytes					Misaligned			Misalign
8 Bytes	Aligned							
8 Bytes		Misaligned						



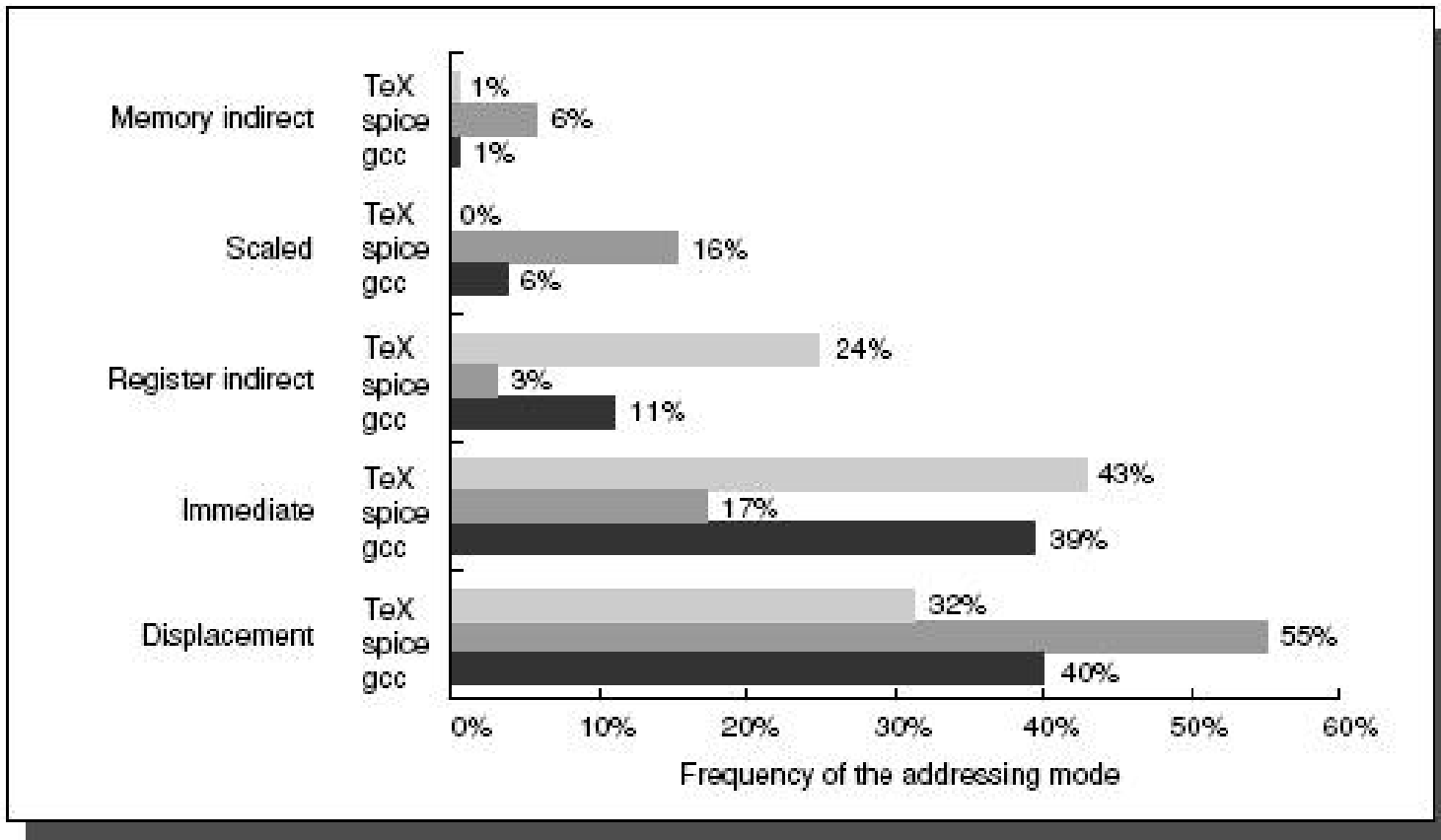
Recap: 寻址方式

- 寻址方式：如何说明要访问的对象地址
- 有效地址：由寻址方式说明的某一存储单元的实际存储器地址。有效地址 vs. 物理地址

Mode	Example	Meaning	When used
Register	Add R1, R2	$R1 \leftarrow R1 + R2$	Values in registers
Immediate	Add R1, 100	$R1 \leftarrow R1 + 100$	For constants
Register Indirect	Add R1, (R2)	$R1 \leftarrow R1 + \text{Mem}(R2)$	R2 contains address
Displacement	Add R1, (R2+16)	$R1 \leftarrow R1 + \text{Mem}(R2+16)$	Address local variables
Absolute	Add R1, (1000)	$R1 \leftarrow R1 + \text{Mem}(1000)$	Address static data
Indexed	Add R1, (R2+R3)	$R1 \leftarrow R1 + \text{Mem}(R2+R3)$	R2=base, R3=index
Scaled Index	Add R1, (R2+s*R3)	$R1 \leftarrow R1 + \text{Mem}(R2 + s*R3)$	s = scale factor = 2, 4, or 8
Post-increment	Add R1, (R2)+	$R1 \leftarrow R1 + \text{Mem}(R2)$ $R2 \leftarrow R2 + s$	Stepping through array s = element size
Pre-decrement	Add R1, -(R2)	$R2 \leftarrow R2 - s$ $R1 \leftarrow R1 + \text{Mem}(R2)$	Stepping through array s = element size



各种寻址方式的使用情况?

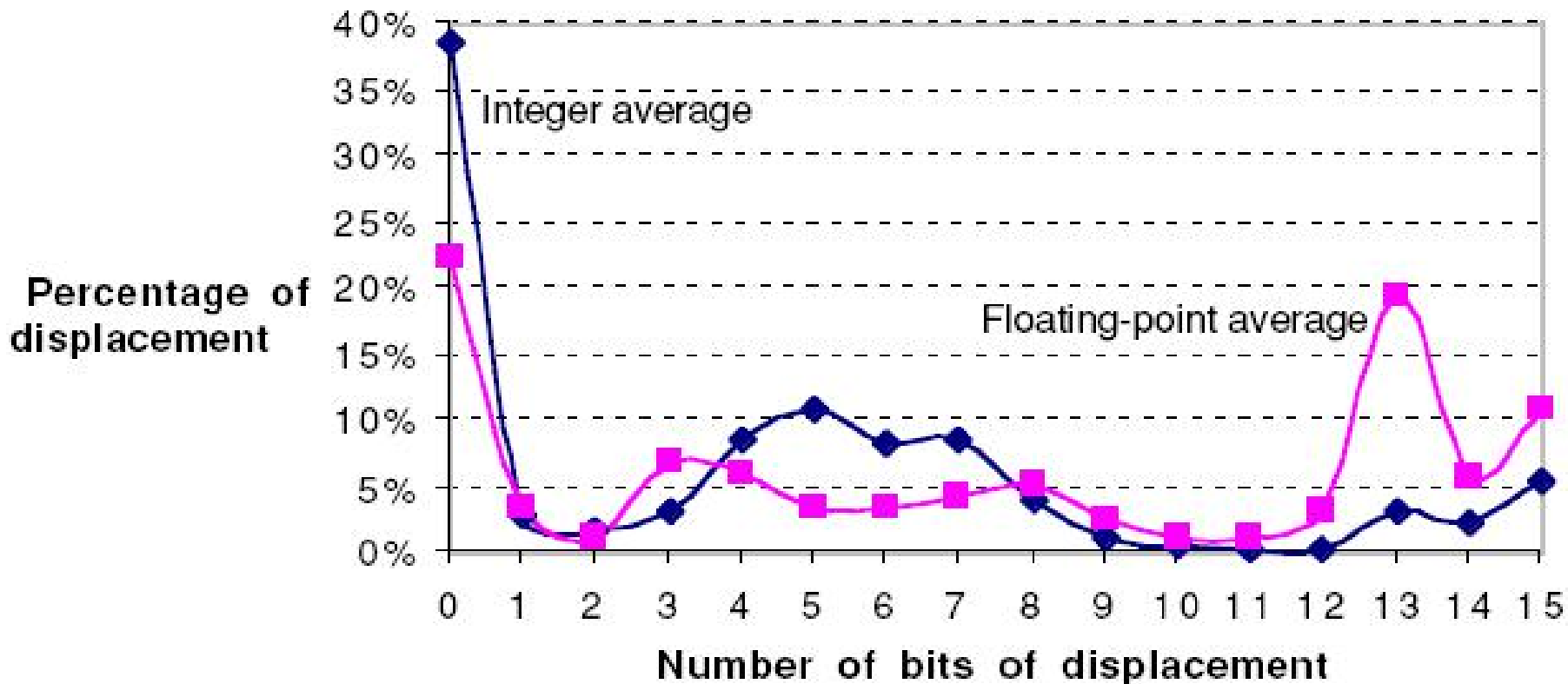


三个SPEC89程序在VAX结构上的测试结果：
立即寻址，偏移寻址使用较多



偏移寻址

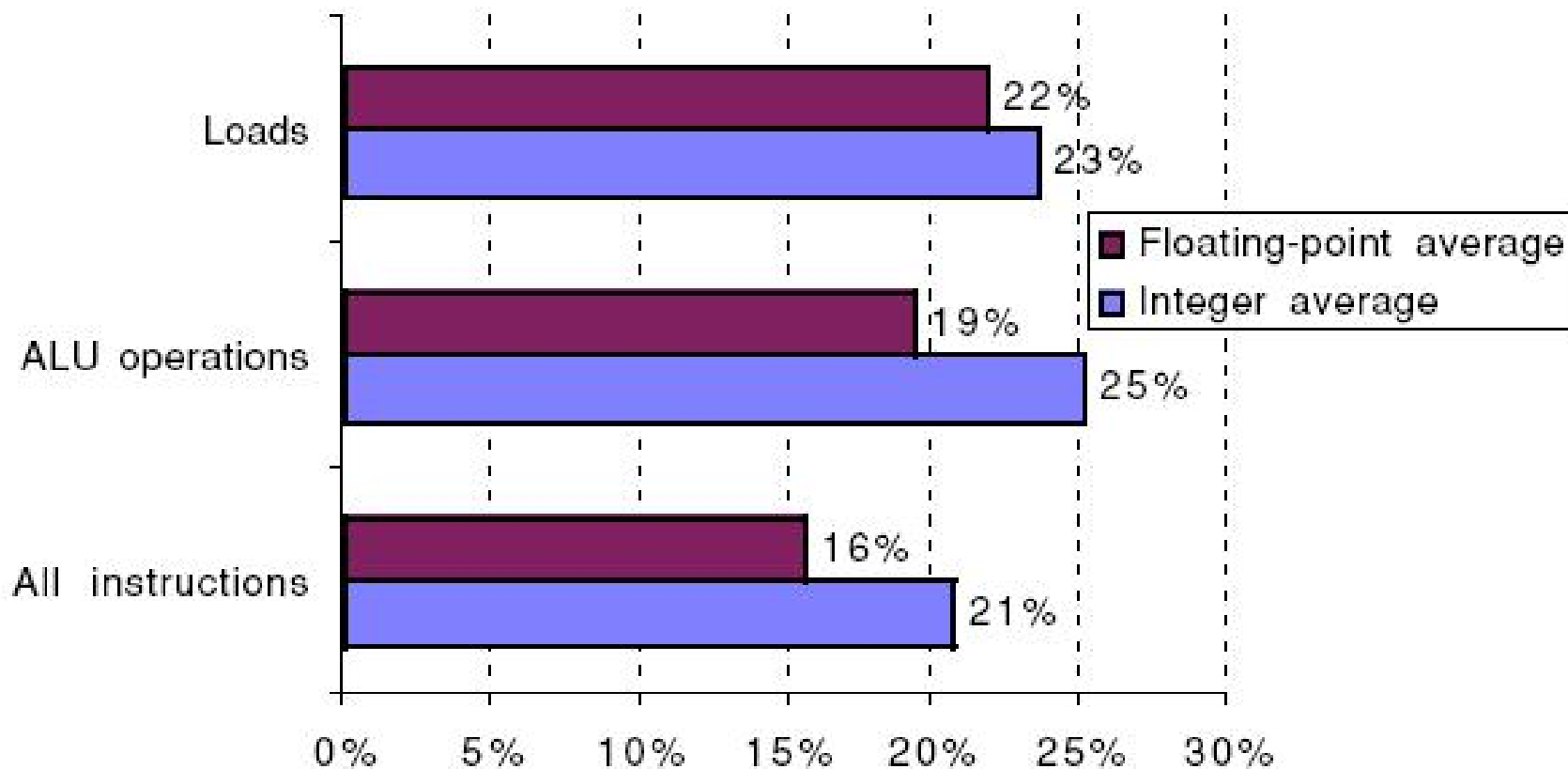
- **主要问题：偏移的范围（偏移量的大小）**



Alpha Architecture with full optimization for Spec CPU2000, showing the average of integer programs(CINT2000) and the average of floating-point programs (CFP2000)



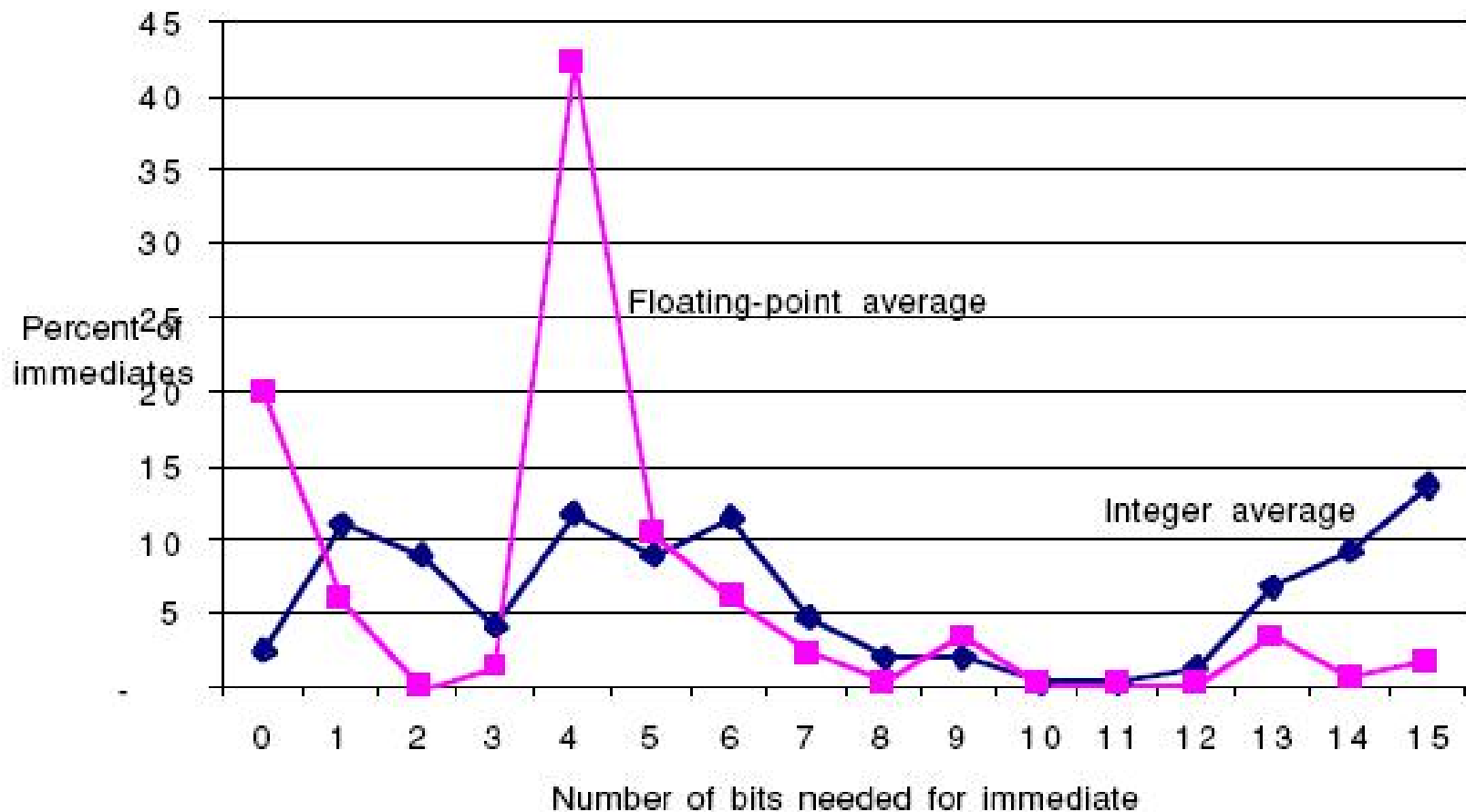
立即数寻址



Alpha Architecture with full optimization for Spec CPU2000, showing the average of integer programs(CINT2000) and the average of floating-point programs (CFP2000)



立即数的大小



The distribution of immediate values. About 20% were negative for CINT2000 and about 30% were negative for CFP2000. These measurements were taken on a Alpha, where the maximum immediate is 16 bits, for the spec cpu2000 programs. A similar measurement on the VAX, which supported 32-bit immediates, showed that about 20% to 25% of immediates were longer than 16 bits.



寻址方式小结

- **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、表示和大小

- **操作数类型和操作数表示是软硬件的主要界面之一。**
- **操作数类型：是面向应用、面向软件系统所处理的各种数据类型。**
 - 整型、浮点型、字符、字符串、向量类型等
 - 类型由操作码确定或数据附加硬件解释的标记，一般采用由操作码确定
 - 数据附加硬件解释的标记，现在已经不采用
- **操作数的表示：操作数在机器中的表示，硬件结构能够识别，指令系统可以直接使用的表示格式**
 - 整型：原码、反码、补码
 - 浮点：IEEE 754标准
 - 十进制：BCD码/二进制十进制表示

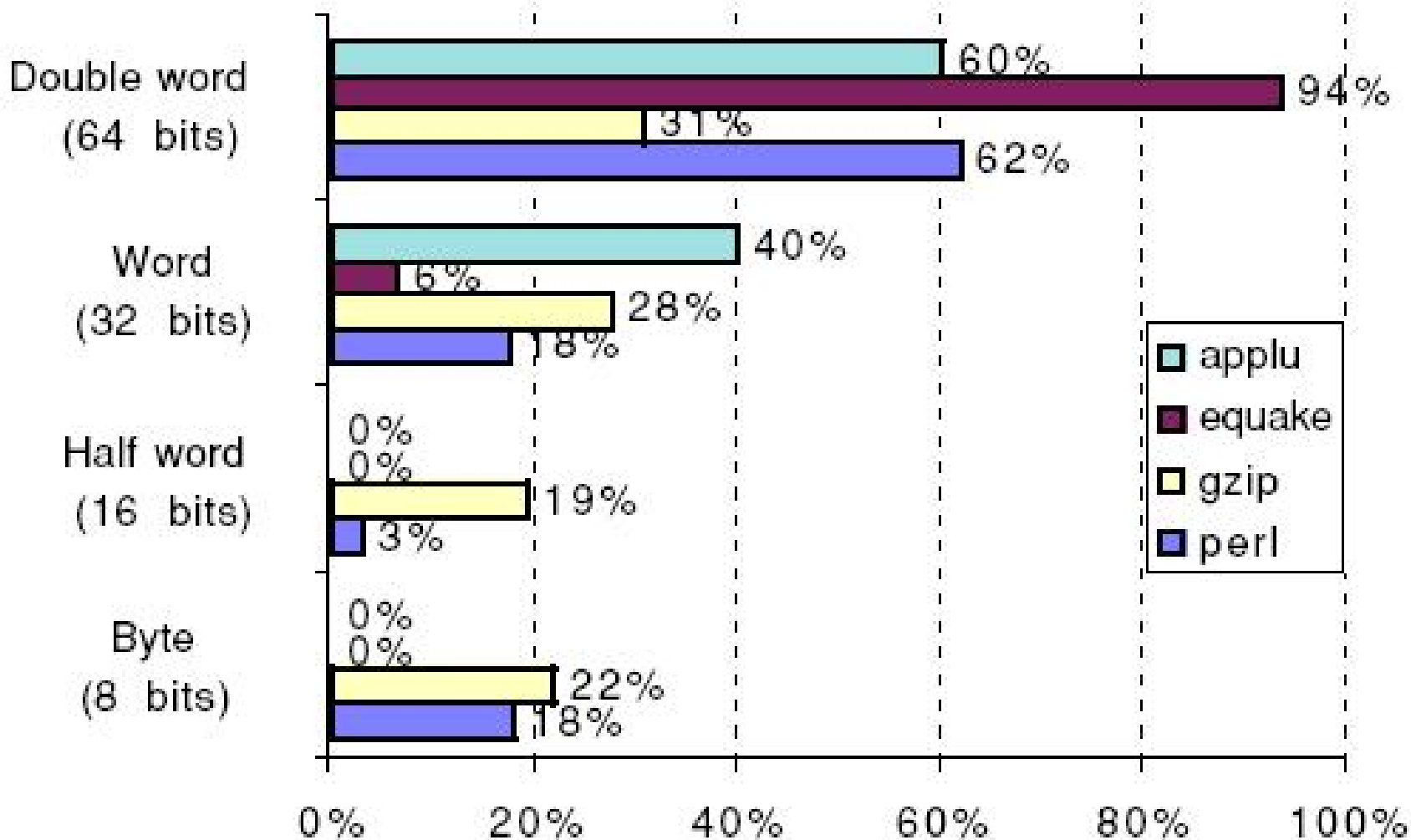


常用操作数类型

- **ASCII character = 1 byte (64-bit register can store 8 characters)**
- **Unicode character or Short integer = 2 bytes = 16 bits (half word)**
- **Integer = 4 bytes = 32 bits (word size on many RISC Processors)**
- **Single-precision float = 4 bytes = 32 bits (word size)**
- **Long integer = 8 bytes = 64 bits (double word)**
- **Double-precision float = 8 bytes = 64 bits (double word)**
- **Extended-precision float = 10 bytes = 80 bits (Intel architecture)**
- **Quad-precision float = 16 bytes = 128 bits**



操作数的大小

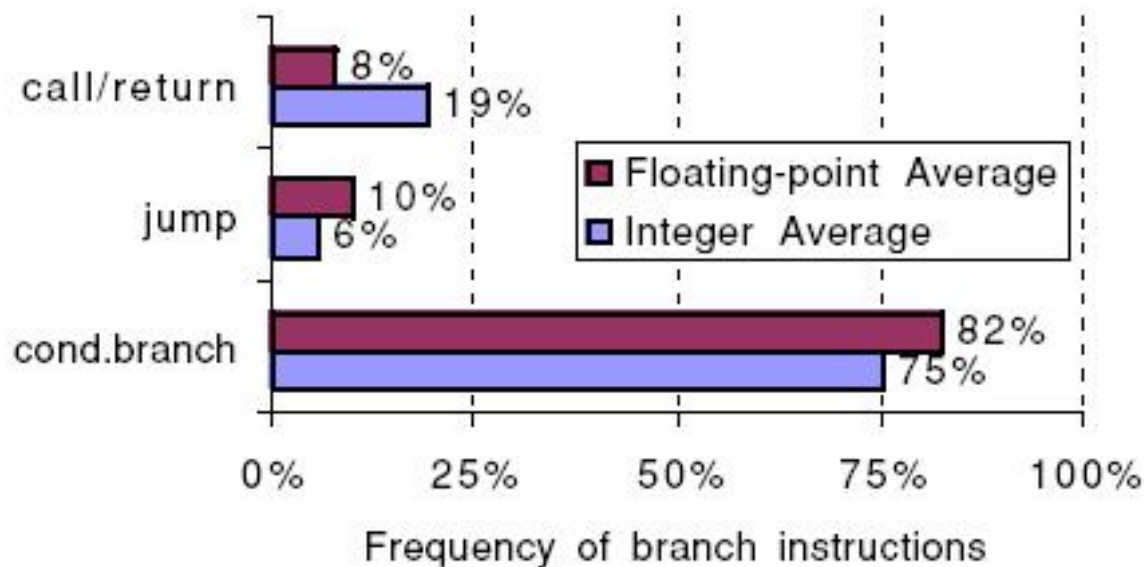


基准测试的结论： (1) 对单字、双字的数据访问具有较高的频率
(2) 支持64位双字操作，更具有—般性



控制类指令

- 四种类型的控制流改变：
 - 条件分支(Conditional branch)、跳转(Jump)、过程调用(Procedure calls)、过程返回(Procedure returns)



Alpha Architecture with full optimization for Spec CPU2000, showing the average of integer programs(CINT2000) and the average of floating-point programs (CFP2000)

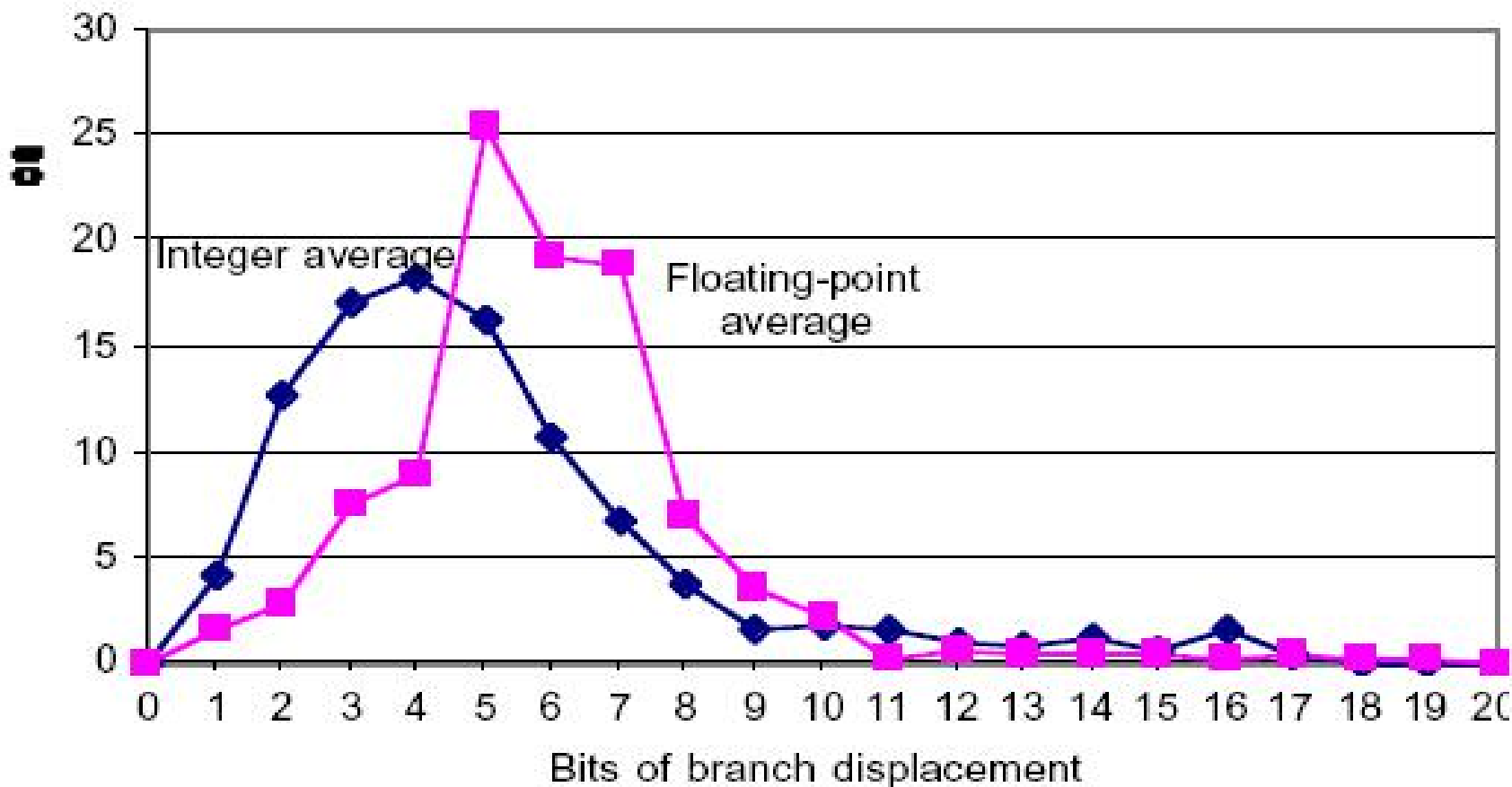


控制流类指令中的寻址方式

- **PC-relative 方式（相对寻址）**
 - 例如：条件转移
- **说明动态的转移地址方式(通过寄存器间接跳转)：**
 - 编译时不知道目标地址，程序执行时动态确定
 - Case or switch statements
 - Virtual function or methods
 - High-order functions or function pointers
 - Dynamically shared libraries
 - 转移地址放到某一寄存器中，通过寄存器间接跳转



转移目标地址与当前指令的距离

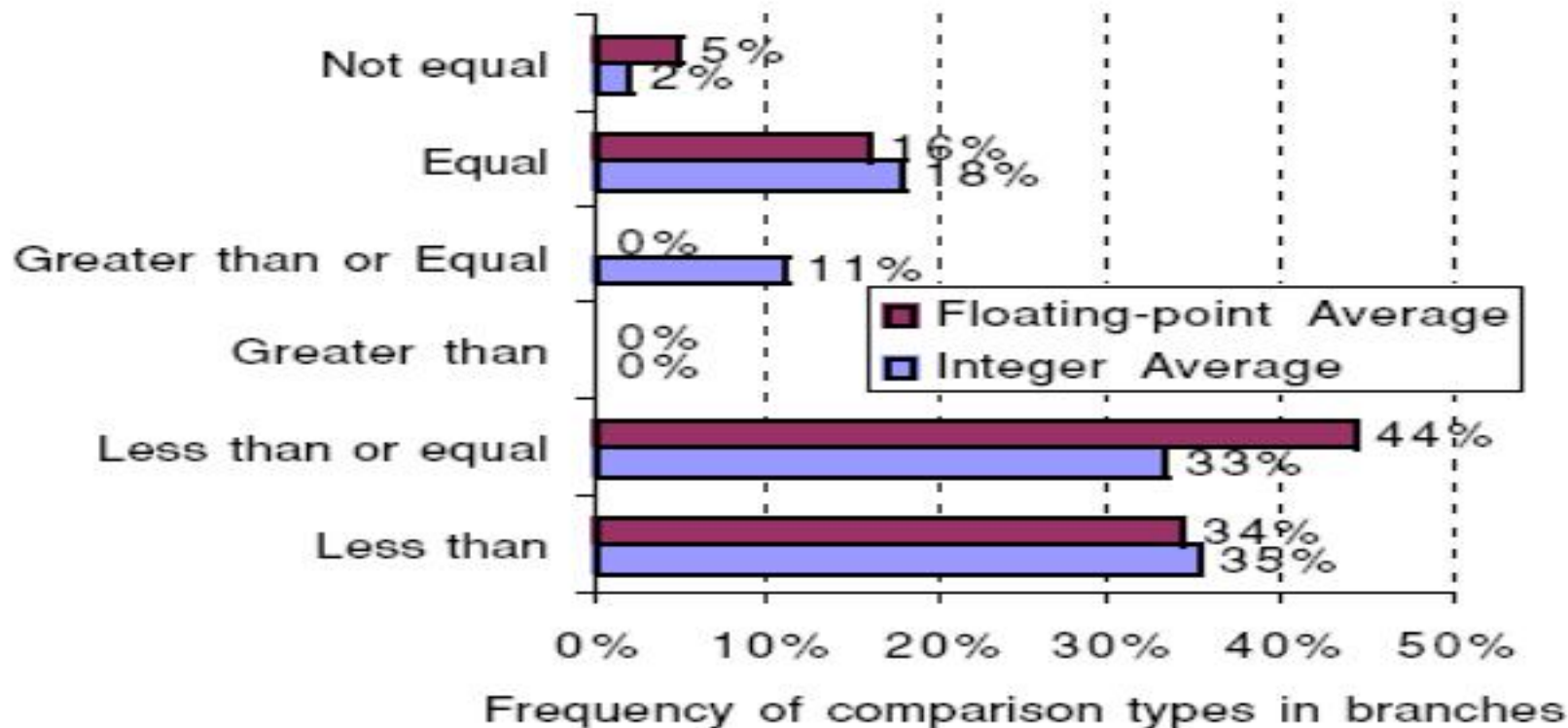


Alpha Architecture with full optimization for Spec CPU2000, showing the average of integer programs(CINT2000) and the average of floating-point programs (CFP2000)

建议：PC-relative 寻址，偏移地址至少8位



分支比较类型比较

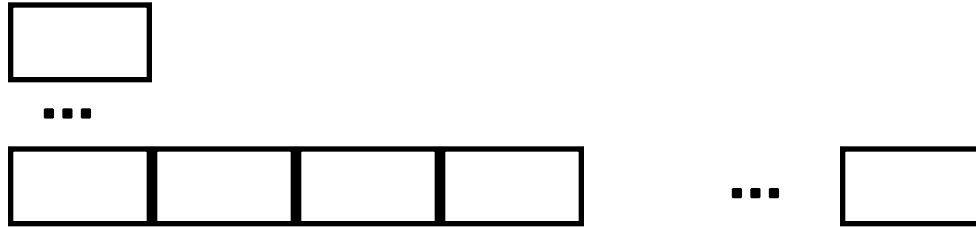


Alpha Architecture with full optimization for Spec CPU2000, showing the average of integer programs(CINT2000) and the average of floating-point programs (CFP2000)



指令编码

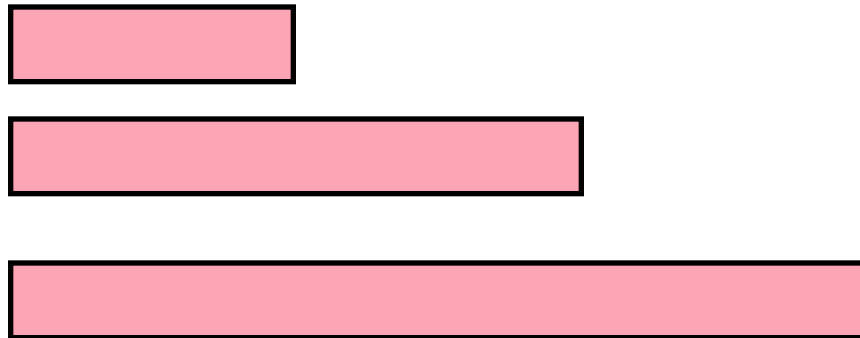
Variable:



Fixed:



Hybrid:



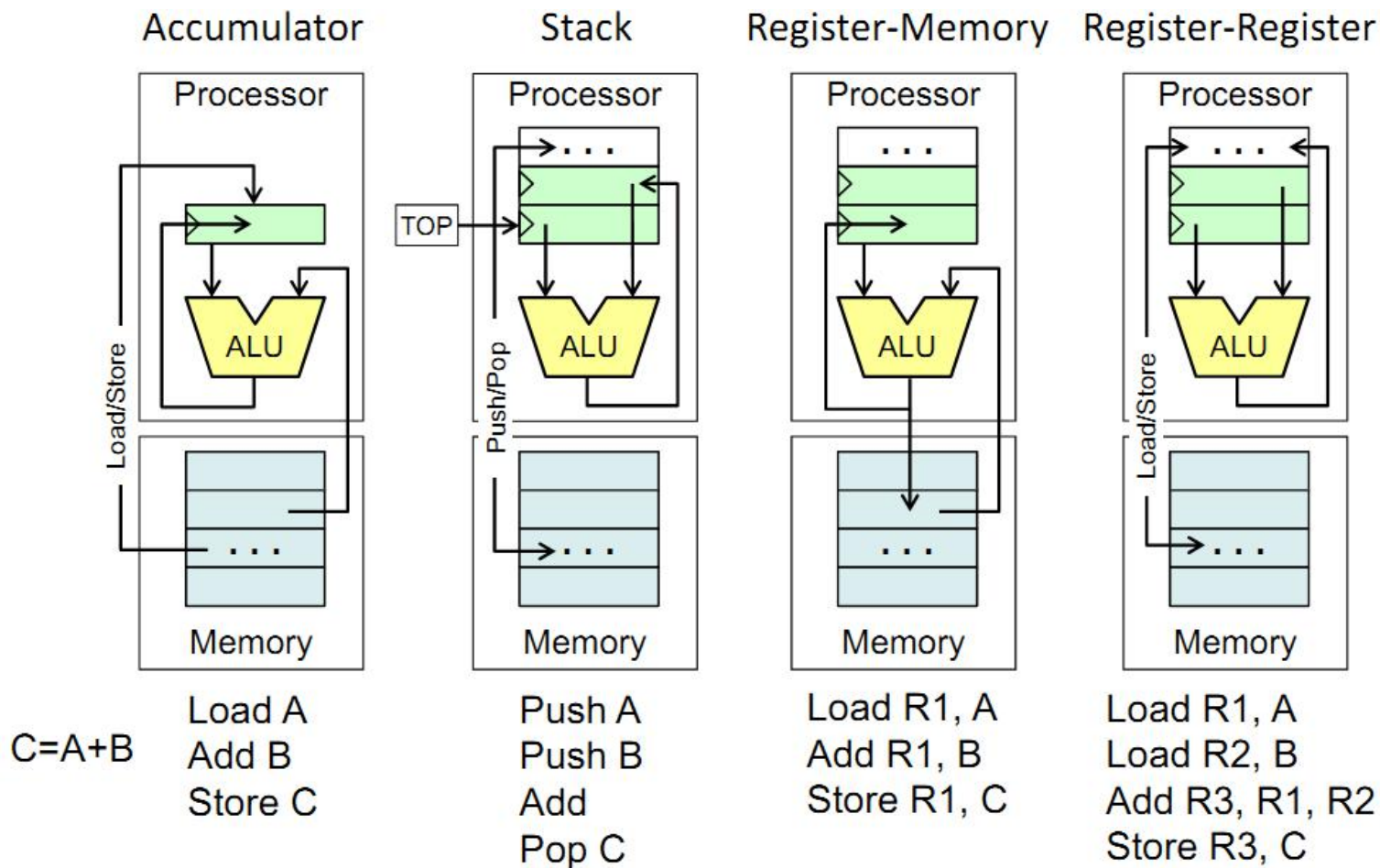


指令格式选择策略

- **如果代码规模最重要，那么使用变长指令格式**
- **如果性能至关重要，使用固定长度指令**
- **有些嵌入式CPU附加可选模式，由每一应用自己选择性能还是代码量**
 - 窄指令支持更少的操作、更短的地址和立即数字段、更少的寄存器以及双地址格式，而不是传统的RISC计算机的三地址格式
 - 例如：RISC-V 的 RV32IC，C表示压缩表示
 - ARM Thumb，microMIPS
- **有些机器使用边执行边解压的方式**
 - 例如 IBM 的CodePack PowerPC



小结：ISA 的演进





小结：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—般性
- **控制转移类指令**
- **指令编码 (指令格式)**



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**