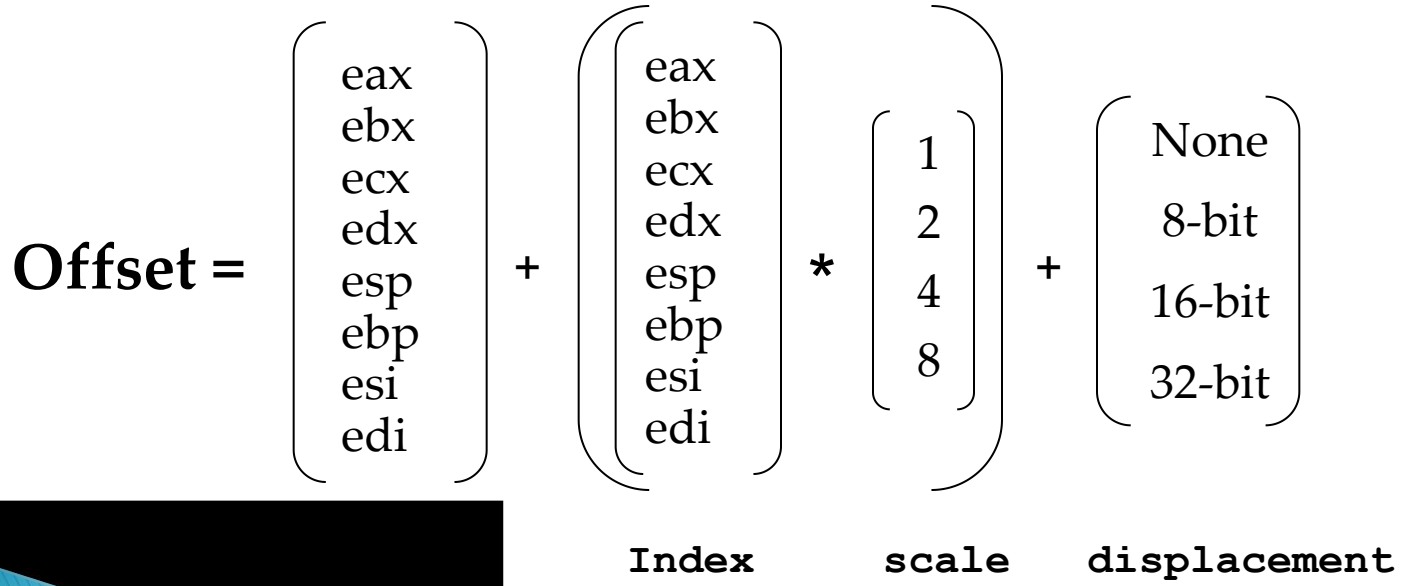
各类指令集简介

X86指令集

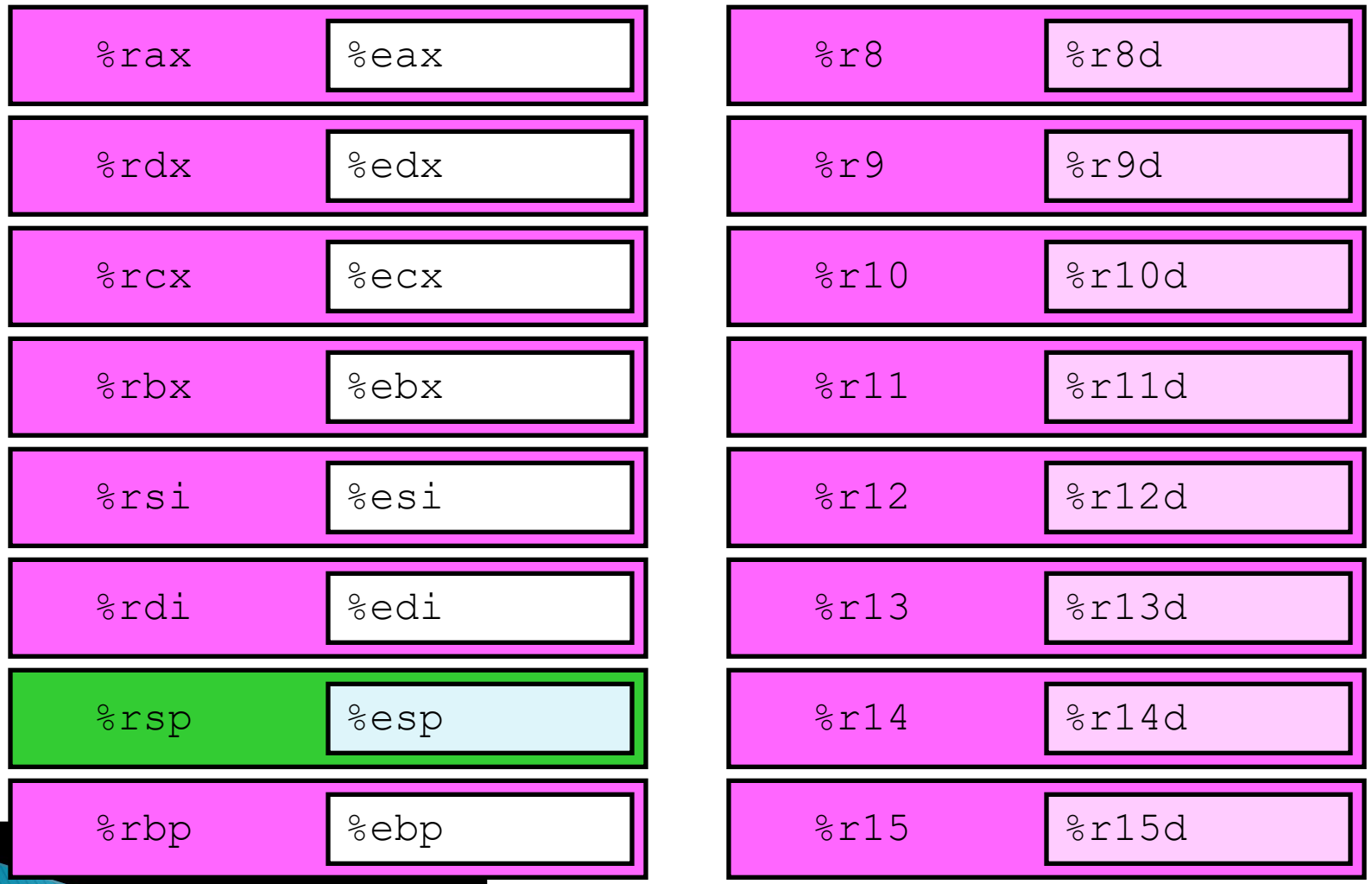
X86指令集的基本特色

- 向下兼容
- 变长指令
  - 1–15 字节，多数为2–3字节长度
- 多种寻址方式（可访问不对齐内存地址）



- 指令集的通用寄存器个数有限
  - X86-32系统下拥有8个通用寄存器（x86-64扩展到16个）
- 至多只有一个操作数在内存中，另一个操作数为立即数或者寄存器

x86-32/64 General Purpose Registers



X86指令类型

- 数据传输指令
  - 在存贮器和寄存器、寄存器和输入输出端口之间传送数据
  - 通用数据传送指令

    MOV、PUSH、POP、XCHG
  - 累加器专用传送指令

    IN、OUT、XLAT
  - 地址传送指令

    LEA、LDS、LES
  - 标志寄存器传送指令

    LAHF、SAHF、PUSHF、POPF
  - 类型转换指令

- 算术指令
  - 加法指令

    ADD、ADC、INC
  - 减法指令

    SUB、SBB、DEC、NEG、CMP
  - 乘法指令

    MUL、IMUL
  - 除法指令

    DIV、IDIV
  - 十进制调整指令

    DAA、DAS、AAA、AAS、AAM、AAD

•逻辑指令

- 逻辑运算指令

  AND、OR、NOT、XOR、TEST

- 移位指令

  SHL、SHR 、 SAL 、SAR、ROL、ROR、RCL、RCR

控制转移指令：

- 无条件转移指令

  JMP

- 条件转移指令

  JZ / JNZ 、 JE / JNE、 JS / JNS、 JO / JNO、

  JP / JNP、 JB / JNB、 JL / JNL、 JBE / JNBE、

  JLE / JNLE、 JCXZ

- **循环指令**

  LOOP、LOOPZ / LOOPE、LOOPNZ / LOOPNE

- 子程序调用和返回指令

  CALL、RET

- 中断与中断返回指令

  RET

处理机控制与杂项操作指令：

- 标志处理指令

  CLC、 STC、 CMC、

  CLD、STD、

  CLI、STI

- 其他处理机控制与杂项操作指令

  NOP、HLT、WAIT、ESC、LOCK

X86浮点指令与SIMD

- 专用的浮点寄存器（最初为80-bit internal registers，后宽度逐步扩展）
  - IEEE Standard 754（Established in 1985 as uniform standard for floating point arithmetic）
    - *Extended precision: 15 `exp` bits, 63 `frac` bits*

  - 早先的x86中浮点寄存器是一个基于栈的结构（而RISC处理器中的浮点寄存器一般可使用寄存器号直接寻址，并且寄存器个数较多）
    - ST0 ~ ST7

SIMD (Single Instruction Multi-DATA)

- 单指令、多数据指令，使得多个运算可以在同一条指令内并发进行（向量运算）
  - 通常其复用浮点寄存器，如128-bit的寄存器可以同时存放2或4个浮点数 (64 or 32 bits wide respectively)；或者是2, 4, 8 或16个整数,同时进行相同运算
  - 充分利用应用的数据并发性
- Intel Sandy Bridge架构拥有了256-bit 的SIMD 指令(including 256-bit memory load and store, AVX).

- **SIMD位宽的加大的一个前提条件是访问内存指令的数据宽度的增大**
  - 但是SIMD的应用效率取决于两个方面，一是处理器的IO带宽；二是应用自身的特性能否提供充分的数据并发性

SIMD一再拓宽…

- **较早的MMX（1997， Intel）**
  - 主要应用于多媒体处理
  - MMX “增加”了8 个寄存器（MM0~MM7 ）
    - 实际上，其复用了现有的浮点寄存器（ST0~ST7），但是寻址方式不同
      - 寄存器号直接寻址
  - 任一个MMX寄存器的宽度是64bit，使用了*packed data types*（two 32-bit integers（doubleword), four 16-bit integers（word) or eight 8-bit integers）

- MMX的缺点
  - 只支持整数运算，浮点数运算仍然要使用传统浮点指令。
  - 与浮点寄存器相互重叠，这限制了MMX指令在需要大量浮点运算的程序，如三维几何变换、裁剪和投影中的应用
  - 栈式暂存器结构，使得硬件上将其流水线化和软件上合理调度指令都很困难，这成为提高x86架构浮点性能的一个瓶颈。

3DNow!（AMD）

- K6-2处理器是第一个能执行浮点SIMD指令的x86处理器
- SIMD多媒体指令集，支持单精度浮点数的向量运算

- SSE （Streaming SIMD Extensions）
  - SSE 彻底抛弃了传统的栈式浮点处理器结构，
  - 8个独立的128位寄存器 （XMM0 ~ XMM7），64位结构下增至16个

SSE-1——同时处理4个32位单精度浮点数

SSE-2——增加了对于2 x 64位浮点或者整数、4x32位整数、8x16位整数、16x8位整数的支持

- Intel SandyBridge架构引入了Advanced Vector Extensions指令集扩展
  - SIMD 寄存器宽度增至256位，16个寄存器(YMM0-YMM15)
    - 向下兼容SSE
    - 每个周期可以进行两个256-bit 的AVX操作
  - 引入了三操作数的指令，
  - Sandy Bridge架构支持256位的内存访问接口

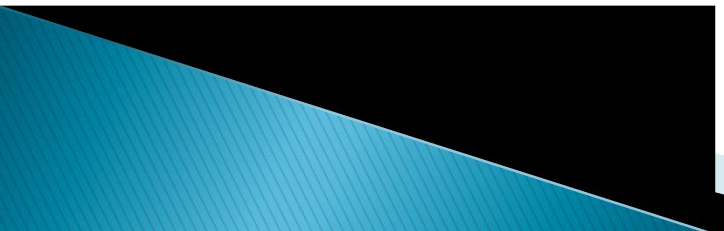
2015年计划推出最新的AVX-512 （512位）

- 为何X86指令集长久不衰？
  - 商业上的成功是其主要原因（生态环境）
  - 技术上注重向下兼容
    - 一个反例是Itanium（IA-64， Explicitly Parallel Instruction Computing ），技术创新但是无法兼容，已基本“死亡”

X86指令集的缺点？

- 向下兼容导致指令集越来越大、越复杂
- 类RISC内核，采用micro-op模式进行翻译，使得功耗相对增大，这导致其在注重低功耗的嵌入式领域不易占优势
- 对很多领域而言，资源利用率低
  - 在高性能计算领域，300余条X86指令中，大致只有80余条是被科学计算所需要的（*美国劳伦斯伯克利国家实验室的研究，2009*）

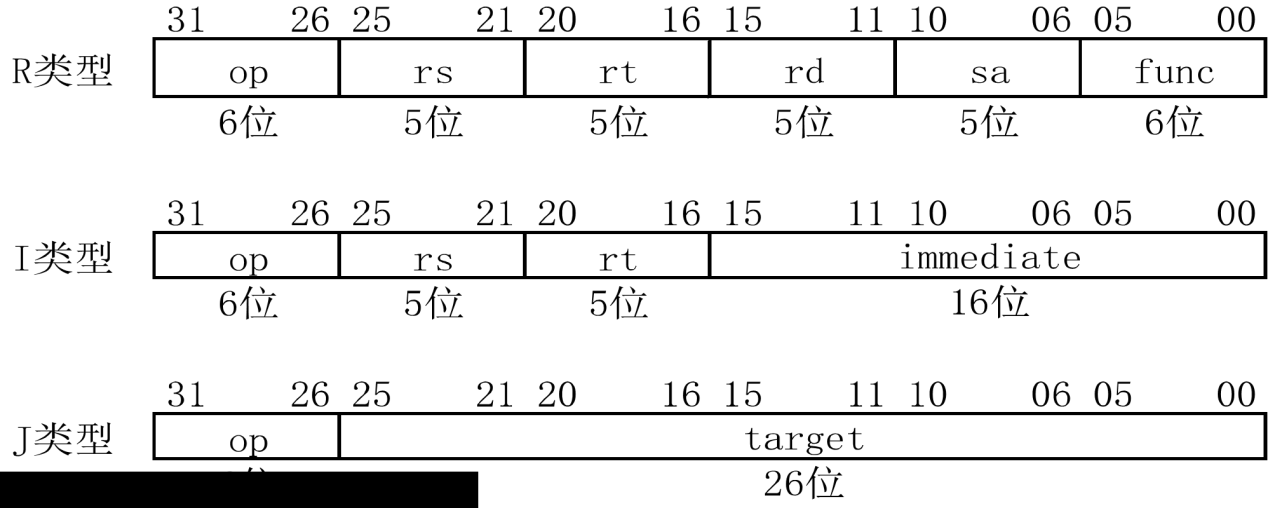
MIPS指令集



- 经典的RISC指令集（主要应用于嵌入式领域）
  - MIPS I、MIPS II、MIPS III、MIPS IV到MIPS V，嵌入式指令体系MIPS16、MIPS32到MIPS64的发展已经十分成熟
  - 为充分利用处理器的流水线结构，其设计思想是使得各个指令的流水线分段较为均匀
    - 分段一致，每段的操作时延相差不多，从而提高主频
    - 尽量使得能够每一周期完成一条指令，控制相对简单
  - 尽量利用软件办法避免流水线中的数据相关/控制相关问题
    - 一个实例：Delay Slot

- 以寄存器为中心（32个），只有Load/Store指令访问内存，所有的计算类型的指令均从寄存器堆中读取数据并把结果写入寄存器堆中。
  - MIPS32还定义了32个浮点寄存器
- MIPS32指令集的指令格式非常规整，所有的指令长度一定，而且指令操作码在固定的位置上。
- MIPS指令的寻址方式非常简单，每条指令的操作也较简单

- MIPS32™的指令格式只有3种
  - R（register）类型的指令从寄存器堆中读取两个源操作数，计算结果写回寄存器堆。
  - I（immediate）类型的指令使用一个16位的立即数作为源操作数。
  - J（jump）类型的指令使用一个26位立即数作为跳转的目标地址（target address）。



- Load和Store指令都为立即数（I-type）类型，用来在存储器和通用寄存器之间的储存和装载数据。MIPS指令集只有该类指令访问内存，而其他指令都在寄存器之间进行，所以指令的执行速度较高。
  - 该类指令只有基址寄存器的值加上扩展的16位有符号立即数一种寻址模式，数据的存取方式可以是字节（byte）、字（word）和双字（Double word）。

- MIPS扩展指令集
  - MIPS-3D， 浮点SIMD 用于三维几何处理（MIPS64架构下）
    - Vertex transformation (matrix multiplication)
    - Clip-check (compare and branch)
    - Transform to screen coordinates (perspective division using reciprocal)
    - Lighting : infinite and local (normalization using reciprocal square root)
  - 采用MIPS64浮点运算单元和双单精度数据类型。
    - PS (paired-single，双单精度)操作可对64位寄存器中的两个32位浮点值进行运算，从而提供2路SIMD (单指令多数据)能力

- MIPS扩展指令集
  - MDMX (MaDMaX， MIPS digital media extensions)
    - 定义了一个“新”的64位寄存器堆（32个）以及一个192位的乘法累加器
      - 重命名了现有的浮点寄存器
    - 新的数据类型： octo byte (8x8 Bit) and quad half (4x16 Bit)
      - 这些数据类型常见于音视频、图像处理应用

Modified Instructions	
ADD, SUB, MUL, MIN, MAX, MSGN	Saturating arithmetic
AND, XOR, OR, NOR, SLL, SRL, SRA	Logicals and shifts
ALNI, ALNV	Align vectors
C.EQ, C.LT, C.LE	Compare bytes
New Instructions	
SHFL.op	Shuffle bytes
PICKF, PICKT	Combine vectors
ADDL, SUBL, MULL, MULSL	Store result in ACC
ADDA, SUBA, MULA, MULS	Operate on ACC
RZU, RNAU, RNEU, RZS, RNAS, RNES	Round ACC
RAC, WAC	Read/write ACC

- MIPS扩展指令集
  - MIPS16e

  - 16位指令，指令集被压缩，代码存储容量要求减小，从而降低系统成本
  - 相比MIPS32，利用MIPS16e编译的应用程序平均减小30%
    - 32个通用寄存器中有8个可用于MIPS16e模式
    - 与MIPS32一起使用时，支持8位、16位和32位数据类型；与MIPS64一起使用时，支持8位、16位、32位和64位数据类型

  - MIPS16e 和 MIPS32/64之间的模式切换支持：通过一条特殊的跳转指令来实现模式切换的软件控制

- 小结
  - MIPS是一个经典的RISC指令集，兼具RISC设计的简洁优雅与不足
    - 代码密度较低
    - 应用于嵌入式领域，32bit指令有些“大材小用”
    - 因此注重扩展，包括提高代码密度（16位指令）以及多媒体、加密领域的指令扩展

ARM指令集

“有些不同”的RISC指令集（32位）

- ARM 指令提供简单的操作，使一个周期就可以执行一条指令。编译器或者程序员通过几条简单指令的组合来实现一个复杂的操作
- ARM 指令集大多数指令采用相同的字节长度，并且在字边界上对齐，字段位置固定，特别是操作码的位置。
- ARM 处理器使用Load/Store 的存储模式，其中只有Load 和Store 指令才能从内存中读取数据到寄存器，所有其他指令只对寄存器中的操作数进行计算。
  - 16 个32位寄存器

与MIPS相比，寻址相对复杂

- **立即数寻址**

  每个立即数都是采用一个8位的常数循环右移偶数位间接得到。

- **寄存器寻址**

  ADD R3，R2，R1，LSR #2

  寄存器R1的内容右移了两位，但是注意本指令执行完毕以后R1的内容并不改变。

- **前变址、自动变址和后变址**

  1、前变址：LDR R0，[R1，#4]

  R1寄存器的内容先加4，然后执行内存操作，但操作完毕以后，R1的内容不变。

  2、自动变址

  R1寄存器的内容先加4，然后执行内存操作，R1变化

  3、后变址：LDR R0，[R1]，#4

  先进行内存操作，然后R1+4->R1（即操作完毕后，变化）

-

- **多寄存器寻址**

采用多寄存器寻址方式，一条指令可以完成多个寄存器值的传送。这种寻址方式可以用一条指令完成传送最多16个通用寄存器的值。

```
LDMIA R0!，{R1，R2，R3，R4}
；R1←[R0]
；R2←[R0＋4]
；R3←[R0＋8]
；R4←[R0＋12]
```

该指令的后缀IA表示在每次执行完加载/存储操作后，R0按字长度增加，因此，指令可将连续存储单元的值传送到R1～R4。

- 多数指令支持条件执行
  - A 4-bit condition code selector
    - 多数处理器只有跳转指令有条件码判断
  - 这一设计的优势在于对于简单的if-else语句，无需产生跳转指令；
    - 劣势在于减少了“立即数”域的位数

```
while(i != j) {
    if (i > j)
        i -= j;
    else
        j -= i;
}
```



```
loop CMP Ri, Rj ; set condition "NE" if (i != j),
                ; "GT" if (i > j),
                ; or "LT" if (i < j)
SUBGT Ri, Ri, Rj ; if "GT" (greater than), i = i-j;
SUBLT Rj, Rj, Ri  ; if "LT" (less than), j = j-i;
BNE loop          ; if "NE" (not equal), then loop
```

- 其他特性（一般RISC指令集所不具备的）
  - PC-relative addressing
  - 融合数据处理功能（数据移动、算术计算）与移位（ shifts 与 rotates ）功能

- **ARM的Thumb指令集**

  - Thumb指令可以看做是ARM指令压缩形式的子集，为提高代码密度而引入的
  - 必须与ARM指令集混用

  - 主要区别
    - 除了跳转指令 有条件执行功能外,其它指令均为无条件执行
    - 没有乘加指令及 64 位乘法指令等，且指令的第二操作数受到限制

    The shorter opcodes give improved code density overall, even though some operations require extra instructions. In situations where the memory port or bus width is constrained to less than 32 bits, the shorter Thumb opcodes allow increased performance compared with 32-bit ARM code, as less program code may need to be loaded into the processor over the constrained memory bandwidth.

- 浮点运算扩展
  - ARM的浮点运算能力不是强项
    - 某些ARM内核只支持软件浮点指令模拟
  - VFP
    - single-precision and double-precision floating-point computation
    - 但是对于SIMD的支持不好
  - NEON
    - 64/128位的 SIMD浮点指令集（相当于Intel的SSE）
    - 采用独立的寄存器堆与硬件流水线
    - 支持8-, 16-, 32- and 位整数与单精度(32-bit) 浮点数据与计算

- 小结
  - ARM指令集介于经典的RISC与CISC之间，相对复杂
  - 注重代码密度，降低功耗
  - 浮点较弱，逐步扩展强化

- 小结
  - CISC与RISC指令集互为借鉴，走向融合
  - 兼容性考虑是指令集发展的关键性因素
  - 为提高数据并行度，SIMD扩展是指令集发展的一个共性
    - 但是取决于处理器访问存储的宽度以及应用的特性
    - 充分利用SIMD是一个非常困难的事！

补　充*

Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures

- In the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA 2013)

- **Question**
  - **The question** of whether ISA plays an intrinsic role in performance or energy efficiency is becoming important. We seek to answer this question through a detailed measurement based study on real hardware running real applications.
- **Answer**
  - We find that ARM and x86 processors are simply engineering design points optimized for different levels of performance, and there is nothing fundamentally more energy efficient in one ISA class or the other. **The ISA being RISC or CISC seems irrelevant.**