计算机系统结构

第二章 指令系统

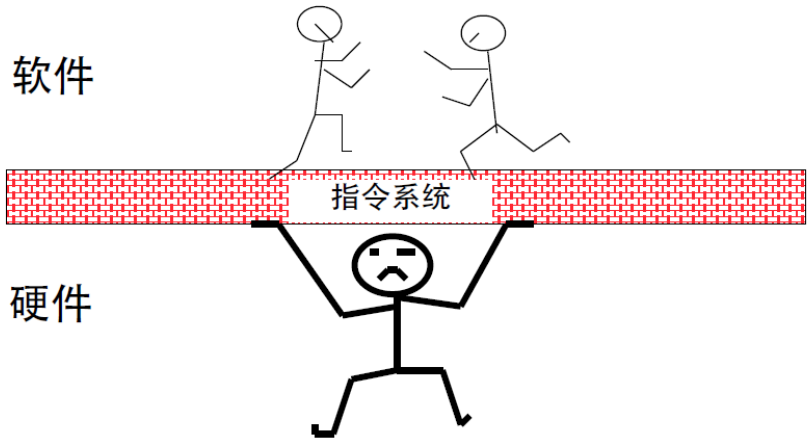
冯 丹

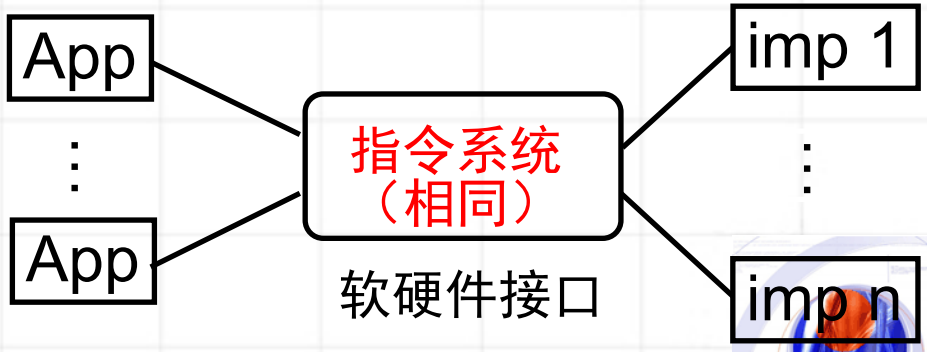
武汉光电国家研究中心

华中科技大学计算机科学与技术学院

2.1 指令系统的基本概念与分类

- 定义：
  - 一台计算机能够直接识别并执行的机器指令的集合
- 软、硬件的分界面
  - 定义了软、硬件交互的“**协约**”
  - 提供了一种软件告诉硬件该执行什么操作的机制
- 不关心硬件实现
  - 可以有不同的实现
    - Intel X86
    - AMD X86
  - 相同指令系统，软件兼容





2.1 指令系统的基本概念与分类

- 指令系统与人类语言
  - 交流工具
    - 语言：人与人
    - ISA：软件与硬件
  - 同一种语言/ISA才能交流
  - 共同点
    - 构成：verbs, nouns, adjectives, adverbs, etc
    - 共同操作：calculation, control/branch, memory
  - 不同的语言/ISA
    - 有很多不同，也很多相似之处
    - 结构不同
  - 随时间不断发展
- 与人类语言最大不同：不能模糊不清
  - ISA设计、扩展必须严格的工程化的方法进行

2.1 指令系统的基本概念与分类

指令系统设计原则

- 可编程性(Programmability) ——软件
  - 简单、高效的编程
    - 程序员、编译器
- 可实现性(Implementability) ——硬件
  - 容易设计出高性能计算机
    - 低功耗
    - 高可靠
    - 低成本
- 兼容性(Compatibility) ——用户
  - 向上兼容与向下兼容
  - x86 (IA32) generations: 8086, 286, 386, 486, Pentium, PentiumII, PentiumIII, Pentium4(向上兼容)

所有ISA都可以实现，但不是所有ISA都能高效地被实现

2.1 指令系统的基本概念与分类

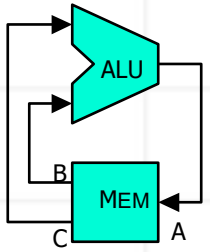
指令系统设计要素

- 指令格式
  - 指令长度、编码
    - 定长、变长
- 操作数存储位置、类型、长度、个数
  - 寄存器、主存、累加器、堆栈……
  - 整型、浮点
  - 字节、字，双字……
  - 1, 2, 3，多操作数……
- 寻址方式
  - 寄存器, 立即数，间接寻址……
- 操作类型
  - add, sub, mul, move, compare……

根据操作数存储单元分类

仅主存型(Memory Only)

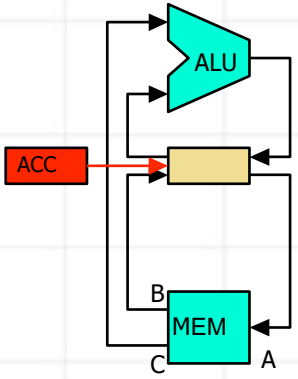
- 操作数仅在主存：
  add B,C,A mem[A] = mem[B] + mem[C]



累加器型(Accumulator)

- 隐式给出一个操作数为累加器：

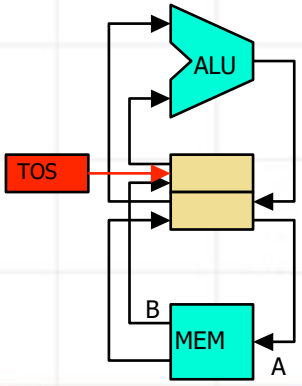
```
load  B        ACC = mem[B]
add   C        ACC = ACC + mem[C]
store A        mem[A] = ACC
```



堆栈型(Stack)

- 隐式给出一个操作数为累加器：

```
push  B     stk[TOS++] = mem[B]
add         stk[TOS++] = stk[--TOS] + stk[--TOS]
pop   A      mem[A] = stk[--TOS]
```

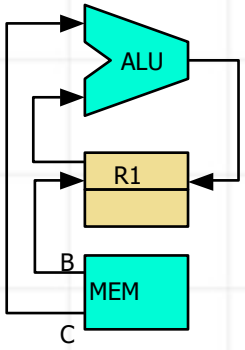


根据操作数存储单元分类

- 通用寄存器型(Registers)
  - 寄存器-存储器型结构（RM）：

    load B, R1　　R1 = mem[B]

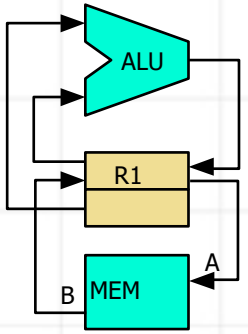
    add C, R1　　R1 = R1 + mem[C]



  - 寄存器-寄存器结构（RR）
    - 仅load-store指令可访问存储器

      store R1, A　　mem[A] = R1

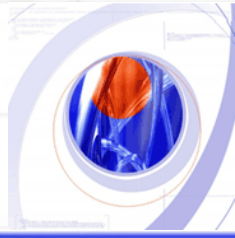
      load B, R1　　R1 = mem[B]



2.2 指令系统发展

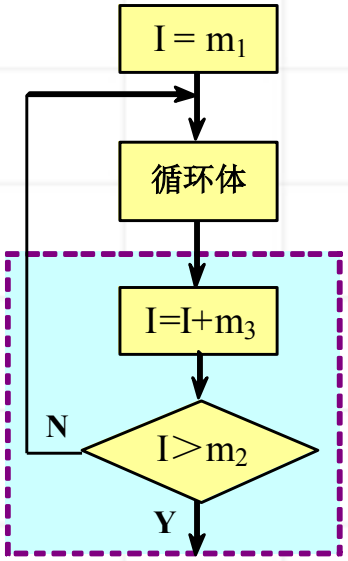
指令系统发展

- 性能公式
  - CPU时间=指令条数IC * CPI * 周期时间
- 复杂指令系统CISC
  - 减少指令条数，使用复杂的指令
  - 易编程（汇编语言级），程序代码量少
  - 商业上很成功
- 精简指令系统RISC
  - 减少CPI，使用大量单周期指令
  - 增加了指令条数，但并不太多
  - 可能减少时钟周期时间
  - 技术上胜利者



复杂指令系统CISC

- 早期计算机指令数较少（50-60年代）
  - 程序简单，需求少，几十条指令，寻址方式简单
- 60年代中后期出现系列机、兼容机
  - IBM370系列，IBM PC系列（向下兼容）
  - VLSI技术进步，硬件越来越复杂，所支持的指令系统也趋于多用途、强功能化
  - 围绕缩小指令与高级语言语义差异以及有利于操作系统优化进行改进
  - 对于使用频度高的指令串，用一条新的指令来替代



循环控制部分通常用3条指令（加、比较、分支）完成

设置循环控制指令，用一条指令完成上述3条指令的功能

复杂指令系统存在的问题

- 设计周期长，准确性难以保证
- 需要大量硬件支持
- 很多复杂指令使用频率很低，造成资源浪费
  - 统计显示，使用最频繁的前10条指令在程序中占比达96%
- 许多指令由于操作繁杂，其CPI值比较大，执行速度慢
- 规整性不好，不利于采用流水技术来提高性能

80x86 Instruction Frequency

Rank	*Instruction*	*Frequency*
1	load	22%
2	branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	register move	4%
9	call	1%
10	return	1%
Total		96%

精简指令系统 RISC

- RISC遵循的原则
  - 指令条数少、功能简单
    - 只选取使用频度很高的指令，补充一些最有用的指令
  - 指令格式简单、规整，并减少寻址方式
    - 如：指令字长都为32位或64位
  - 指令的执行在单个周期内完成（采用流水线机制）
  - 只有load和store指令才能访问存储器，其它指令的操作都是在寄存器之间进行；（即采用load-store结构）
  - 大多数指令都采用硬连逻辑来实现；
  - 强调优化编译器的作用，为高级语言程序生成优化的代码；
  - 充分利用流水技术来提高性能。

$$S_n = \frac{1}{(1-Fe)+\frac{Fe}{Se}}$$

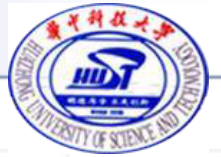
精简指令系统 RISC

- 早期的RISC微处理器
  - 1981年，Berkeley分校的Patterson 等人的32位微处理器RISC Ⅰ：
    - 31条指令，指令字长都是32位，78个通用寄存器，时钟频率为8MHz；
    - 控制部分所占的芯片面积只有约6%。商品化微处理器MC68000和Z8000分别为50%和53%；
    - 性能比MC68000和Z8000快3～4倍。
  - 1983年的RISC Ⅱ：
    - 指令条数为39，通用寄存器个数为138，时钟频率为12MHz。
    - 后来发展成了Sun公司的SPARC系列微处理器。
  - 1981年，Stanford大学Hennessy等人的MIPS
    - 后来发展成了MIPS Rxxx系列微处理器。
  - IBM的801

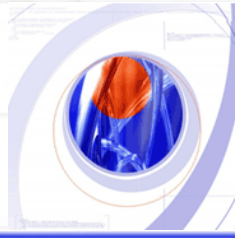
共同特点：采用load-store结构、指令字长为32位、采用高效的流水技术

基于精简指令集的RISC-V

- 2010年始于加州大学柏克莱分校
- 完全开源
  - RISC-V指令集可以自由地用于任何目的，允许任何人设计、制造和销售RISC-V芯片和软件
- 架构简单
  - 基础指令集则只有40多条，加上其他的模块化扩展指令总共几十条指令
- 适用于现代计算设备，小型、快速、低功耗
  - 如仓库规模云计算机、高端移动电话和微小嵌入式系统
- 具有众多支持的软件
  - 开发者能非常方便的移植linux和unix系统到RISC-V平台
- 完整的工具链
  - RISC-V社区已经提供了完整的工具链，并且RISC-V基金会持续维护该工具链。当前RISC-V的支持已经合并到主要的工具中，比如编译工具链gcc, 仿真工具qemu等



3.3 MIPS指令系统



国产高性能处理器

- 飞腾
  - 2014年10月，飞腾第一款**兼容ARM指令集**的CPU——FT-1500A面世，最新推出的FT-2000+实测性能达到了2014年Intel志强E5主流服务器CPU的水平。应用在天河超级计算机上。

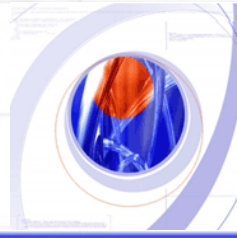
- 龙芯
  - 龙芯是中国科学院计算所主导研发的通用CPU，主要产品有龙芯1号、龙芯2号和龙芯3号。采用的是RISC架构，**兼容MIPS指令**。龙芯3B是首款国产商用8核处理器，支持向量运算加速，峰值计算能力达到128GFLOPS

- 申威
  - 申威处理器使用的是**Alpha指令**，主要用于超算处理器中，国内最强的超算神威·太湖之光就使用了申威的处理器。

MIPS指令系统结构

MIPS的寄存器

- 32个64位通用寄存器（GPRs）
  - R0，R1，...，R31，也称为整数寄存器
  - R0的值永远是0
- 32个64位浮点数寄存器（FPRs）
  - F0，F1，...，F31
  - 用来存放32个单精度浮点数（32位），也可以用来存放32个双精度浮点数（64位）。
  - 存储单精度浮点数（32位）时，只用到FPR的一半，其另一半没用。
- 一些特殊寄存器
  - 它们可以与通用寄存器交换数据。
  - 例如浮点状态寄存器：用来保存有关浮点操作结果的信息



MIPS指令系统结构

MIPS的数据表示

- MIPS的数据表示
  - 整数

    字节（8位）　　半字（16位）

    字（32位）　　双字（64位）
  - 浮点数

    单精度浮点数（32位）　双精度浮点数（64位）
- 字节、半字或者字在装入64位寄存器时，用零扩展或者用符号位扩展来填充该寄存器的剩余部分。装入以后，对它们将按照64位整数的方式进行运算。

MIPS指令系统结构

MIPS的数据寻址方式

- 立即数寻址与偏移量寻址；

  立即数字段和偏移量字段都是16位的。

- 寄存器间接寻址是通过把0作为偏移量来实现的；
- 16位绝对寻址是通过把R0（其值永远为0）作为基址寄存器来完成的；
- MIPS的存储器是按字节寻址的，地址为64位；
- 所有存储器访问都必须是边界对齐的。

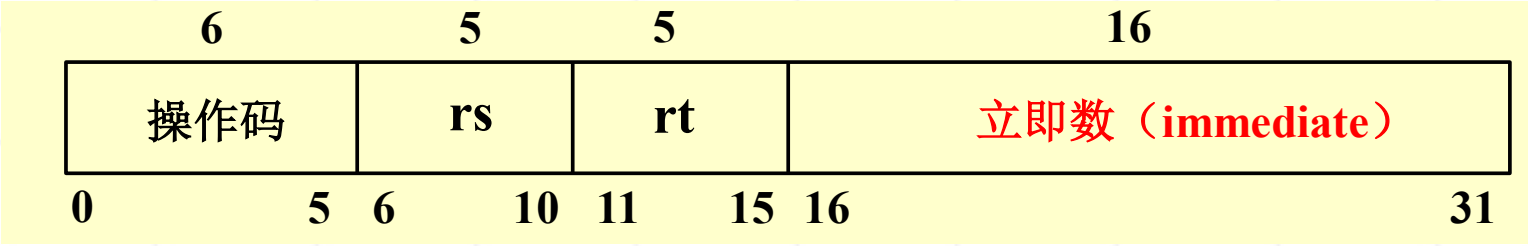
MIPS指令系统结构

MIPS的指令格式

- 寻址方式编码到操作码中
- 所有的指令都是32位的
- 操作码占6位
- 3种指令格式：R、I、J型指令
  - 3种格式中，同名字段的位置固定不变

MIPS指令系统结构

- I类指令
  - 包括所有的load和store指令，立即数指令，分支指令，寄存器跳转指令，寄存器链接跳转指令。
  - 立即数字段为16位，用于提供立即数或偏移量。



MIPS指令系统结构

典型I类指令

- load指令

  访存有效地址：Regs[rs]＋immediate

  从存储器取来的数据放入寄存器rt

- store指令

  访存有效地址：Regs[rs]＋immediate

  要存入存储器的数据放在寄存器rt中

- 立即数指令

  Regs[rt] ← Regs[rs] op immediate

- 分支指令

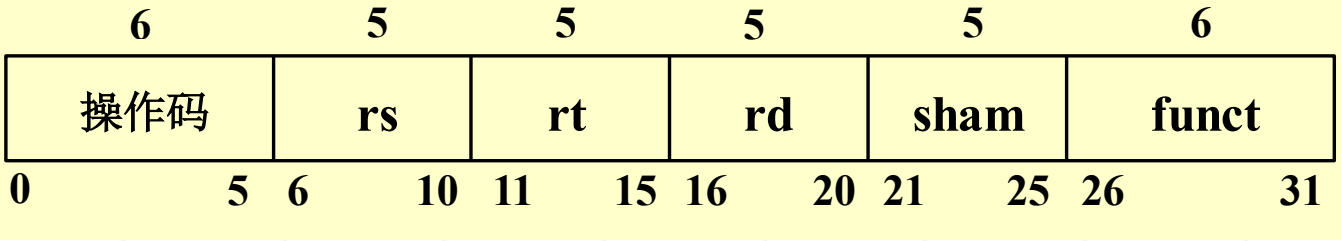
  转移目标地址：Regs[rs]＋immediate，rt无用

- 寄存器跳转、寄存器跳转并链接

  转移目标地址为Regs[rs]

MIPS指令系统结构

- R类指令
  - 包括ALU指令，专用寄存器读/写指令，move指令等。
  - ALU指令
    - Regs[rd]← Regs[rs] funct Regs[rt]
    - funct为具体的运算操作编码



MIPS指令系统结构

- J类指令
  - 包括跳转指令，跳转并链接指令，自陷指令，异常返回指令
  - 在这类指令中，指令字的低26位是偏移量，它与PC值相加形成跳转的地址。

龙芯基于MIPS指令系统设计，正在构建系统生态链，形成规模产业