(微)体系结构
——如何作用于——
编译器后端

邱吉 qiuji.odyssey@gmail.com

2023年5月20日 南盘江计划

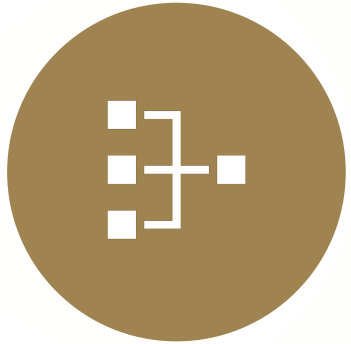
提纲

- 背景概念
- 简明体系结构入门
  - 指令集架构
  - 微体系结构
- 指令集架构对编译器后端的影响
- 微体系结构对编译器后端的影响
- 其他的Tips
- 总结

背景概念——
什么是计算机体系结构（computer architecture）
三个层次的内容

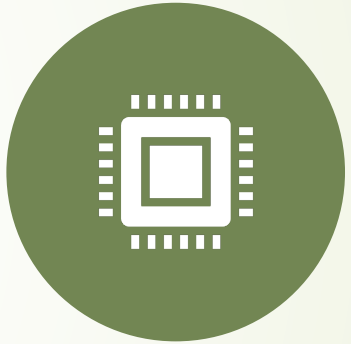
系统架构

SYSTEM ARCHITECTURE



指令集体系结构

INSTRUCTION SET ARCHITECTURE，ISA



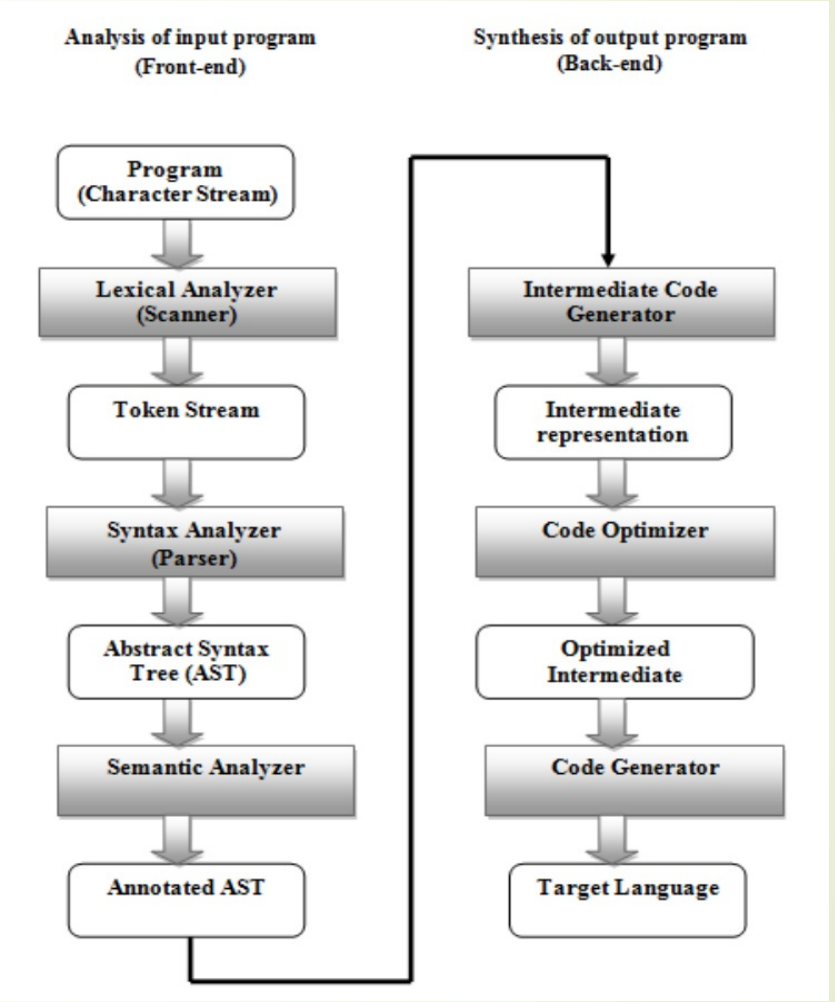
微体系结构

MICRO-ARCHITECTURE

指令集体系结构 （比如：RISC-V，X86，ARM64， PPC）	CISC/RISC 地址宽度（32/64bit） 寄存器宽度（32/64/128/256/512bit） 指令集（语义和编码） 寄存器组（GPR/FPR/VR，控制/状态寄存器，FLAG Bits） 数据类型（Int/FP） 寻址模式 ABI（传参，返回值，内存对齐，数据类型约定）
微体系结构 （比如：每一个具体的CPU IP 或者芯片，都有它的微体系结构参数）	发射宽度 Function Unit类型和数量 Pipeline延迟 存储层次中的Cache组织
系统架构 （比如：每一个开发板、每一台计算机，每一个集群系统都会有这些参数）	内存组织 多核，大小核 多芯片互联 集群 网络互联

背景概念——
什么是编译器后端
（compiler backend)

- 词法分析
- 语法分析
- 语义分析
- IR生成
- IR上的变换和优化
- 指令选择
- 指令调度
- 寄存器分配
- 二进制产生
- 贯穿上述所有过程的架构相关优化



简明体系结构入门-1.指令集体系结构(1)

- ISA的分类
  - 从CPU内部存储类型：堆栈型（指令隐含操作数在栈顶）、累加器型（指令的一个隐含操作数在acc里）、寄存器（指令的操作数必须明确指出，通用寄存器or内存地址）
  - 通用寄存器型的指令集又分成两种，寄存器-存储器型（任何一条指令都可以访问内存）、寄存器-寄存器型（也叫load-store）型
  - load-store型的指令集，也还有更多分类：ALU操作数个数（2or3），操作数位于寄存器还是内存？
  - 从上个世纪80年代后，主流的CPU几乎全都是load-store型
  - why？

简明体系结构入门-1.指令集体系结构(2)

- ISA的分类
  - 从CPU内部存储类型：堆栈型（指令隐含操作数在栈顶）、累加器型（指令的一个隐含操作数在acc里）、寄存器（指令的操作数必须明确指出，通用寄存器or内存地址）
  - 通用寄存器型的指令集又分成两种，寄存器-存储器型（任何一条指令都可以访问内存）、寄存器-寄存器型（也叫load-store）型
  - load-store型的指令集，也还有更多分类：ALU操作数个数（2or3），操作数位于寄存器还是内存？
  - 从上个世纪80年代后，主流的CPU几乎全都是load-store型
  - why？---- 因为它性能更好、编译器更好利用**（实际上编译器也影响了体系结构的发展）**
    - 寄存器访问速度高
    - 计算效率高（相对acc和stack而言）
    - 可以暂存变量，减少数据流量，加快访问速度，改善代码密度

简明体系结构入门-1.指令集体系结构(3)

- 内存地址
  - 字节寻址：地址的单位是字节
  - big endian(高字节在低地址) / little endian（低字节在低地址）
  - 对齐：一个大小为m字节的数据的地址为addr，addr%m == 0，那么addr就是m字节对齐的
    - 地址不对齐的访存操作，会导致例外或者性能损失
  - 寻址模式

寻址模式

寻址模式	样例	语义	使用情景
寄存器寻址	add r1, r2	Regs[r1] ← Regs[r1]+Regs[r2]	数值在寄存器中
立即数寻址	add r1, #3	Regs[r1] ←Regs[r1]+3	数值是常量，编码在指令本身
寄存器间接寻址	mov r1, (r2)	Regs[r1]←Mem[Regs[r2]]	指针
寄存器+偏移量寻址	addr r1, 30(r2)	Regs[r1]←Regs[r1]+Mem[30+Regs[r2]]	指针+偏移、 局部变量
直接寻址	addr r1 , (1024)	Regs[r1]←Mem[1024]	静态数据 但现在几乎没有这种模式了
存储器间接寻址	mov r1,@(r2)	Regs[r1]←Mem[Mem[Regs[r2]]]	指针的指针，地址的产生太复杂，很少见
自增寻址 (post-indexed case)	mov r1,(r2)+	Regs[r1]←Mem[Regs[r2]] Regs[r2] ←Regs[r2] + sizeof_op	循环中的数组递增访问、堆栈push/pop
自减寻址 (pre-indexed case)	mov r1,-(r2)	Regs[r2] ←Regs[r2] - sizeof_op Regs[r1]←Mem[Regs[r2]]	循环中数组的递减访问、 堆栈push/pop
比例寻址	mov r1, 20(r2)[r3]	Regs[r1]←Mem[20+Regs[r2]+Regs[r3]*sizeof_op	数组的基址+index+起始偏移访问

简明体系结构入门-1.指令集体系结构(4)

- 操作数的大小和类型
  - 大小：byte，half word，word，double word
  - 操作数类型：有符号整数、无符号整数、单/双精度浮点
  - 表示方法：二进制补码

简明体系结构入门-1.指令集体系结构(5)

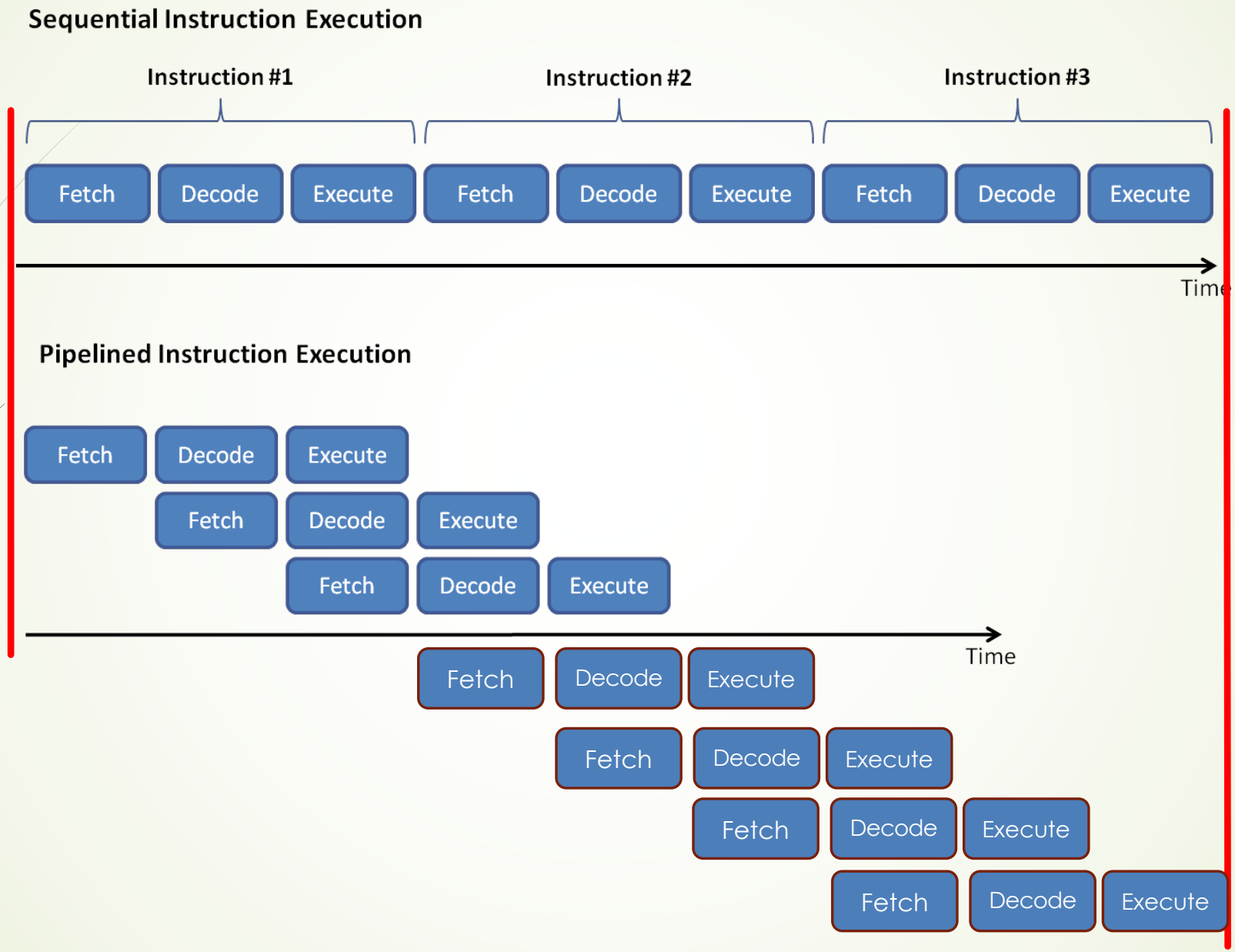
- 指令集的操作
  - 算术和逻辑运算
  - 数据移动：load/store, mov
  - 控制：分支、跳转、函数调用、返回、陷阱
    - 寻址方式：PC相对寻址，寄存器间接寻址
    - 条件转移（分支）的不同实现：基于条件码，基于条件寄存器，cmp+branch一揽子
    - 函数调用的不同实现：遵循ABI规范的caller save和callee save
    - 函数返回的不同实现：基于默认的RA、指定的RA
  - 系统：syscall
  - 浮点：算术、比较、规格化（IEEE754标准），cvt，trunc

简明体系结构入门-1.指令集体系结构(6)

- 指令集的编码
  - 以字节的倍数为单位
  - 变长编码(X86)：一个操作码下，操作数可以有各种寻址模式
  - 定长编码(MIPS/ARM/RISC-V)：一个操作码下，操作数只有固定的寻址模式
  - 混合编码：固定几种格式，由操作码+附加编码域来决定指令语义和操作数寻址模式

简明体系结构入门-2.CPU微体系结构(1)

- 时钟周期：主频
  - 触发器：在每个时钟周期采样输入状态并产生输出状态
- 流水线
  - 一种在连续指令流中开发指令集并行性的技术
  - 将指令的执行过程进行拆分，每个步骤完成一小部分工作，串联这些步骤，在时钟信号的作用下，指令依次“流过”
  - 同一时钟周期，不同的步骤上执行不同的指令
  - 实现了指令的并行



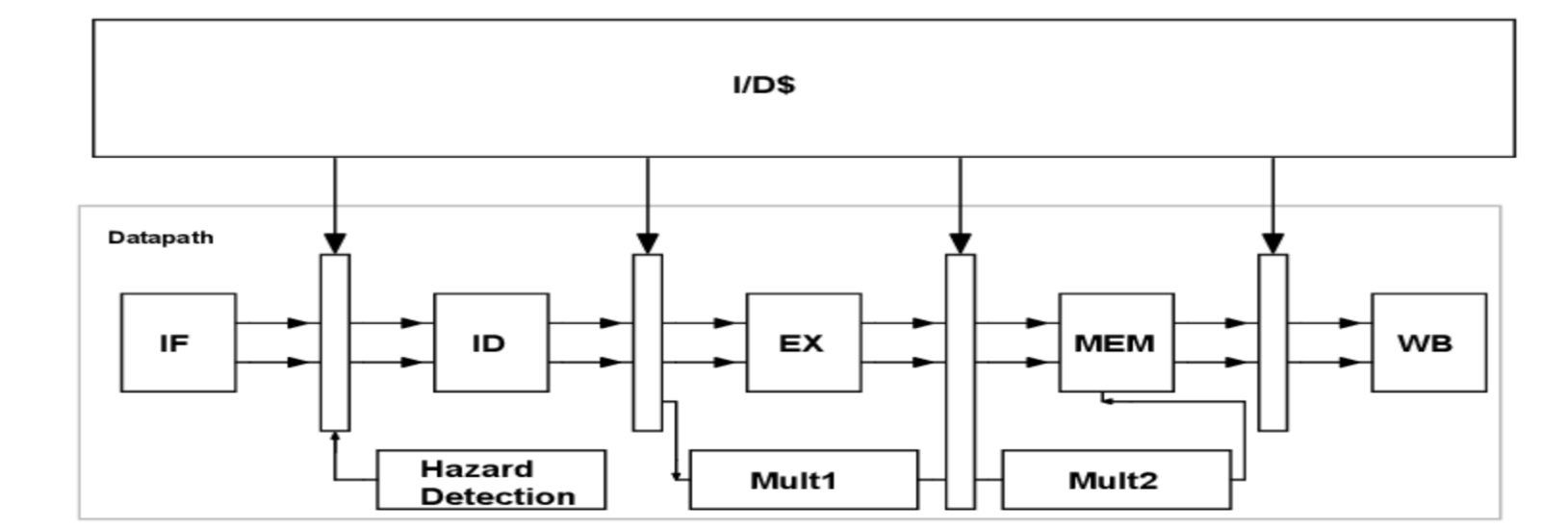
完成3条指令

完成7条指令

https://skills.microchip.com/pic32mx-core-architecture/699292

简明体系结构入门-2.CPU微体系结构(2)

- 流水线的附加开销
  - 流水线寄存器的读写开销导致引入流水线以后，每条指令的实际执行时间增加了

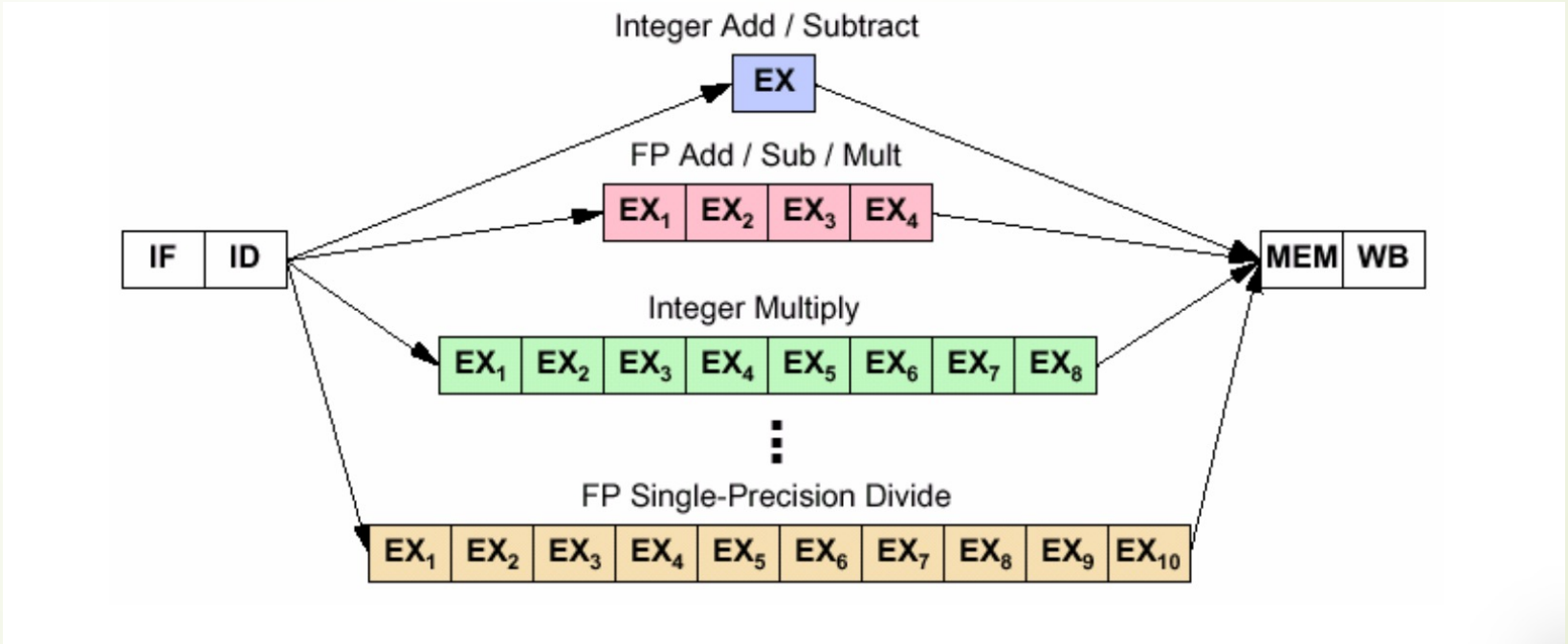


  - 这种开销限制了流水的深度

Jokai, Arpad. “Energy Evaluation of a 5- and a 7-Stage Processor Pipeline Across Nominal and Near-Threshold Supply Voltages Master’s Thesis in Embedded Electronic System Design.” (2015).

简明体系结构入门-2.CPU微体系结构(3)

- 流水线的延迟
  - 多周期操作和不同的功能单元（FU）



https://www.corsi.univr.it/documenti/OccorrenzaIns/matdid/matdid958600.pdf

简明体系结构入门-2.CPU微体系结构(4)

- 流水线停顿
  - 资源冲突：
    - 硬件FU数量不够
    - 多周期操作的FU是非流水的
    - 寄存器读写端口数量限制导致不能同时执行某些指令的组合
  - 数据冲突：流水线中一条指令依赖于它前面指令的结果数据，但却不能及时得到
    - 真相关RAW可以部分通过forwarding来缓解
  - 控制冲突：分支或其他改写指令改写了PC，需要清空流水线
    - 分支跳不跳的计算导致了延迟（Branch History Table等来预测）
    - 分支跳到哪里的计算也导致了延迟（Branch Target Buffer等来预测）
    - 对特殊代码结构用特别的方式来解决：
      - RAS：返回地址栈
      - Loop buffer：对“回跳”的分支进行profile和loop body的暂存
    - 软硬件协同的一种解决方法（已经很少见了）：将不管分支跳不跳都要执行的指令，放进分支后面的delay slot里，“总是做有效的事”

简明体系结构入门-2.CPU微体系结构(5)

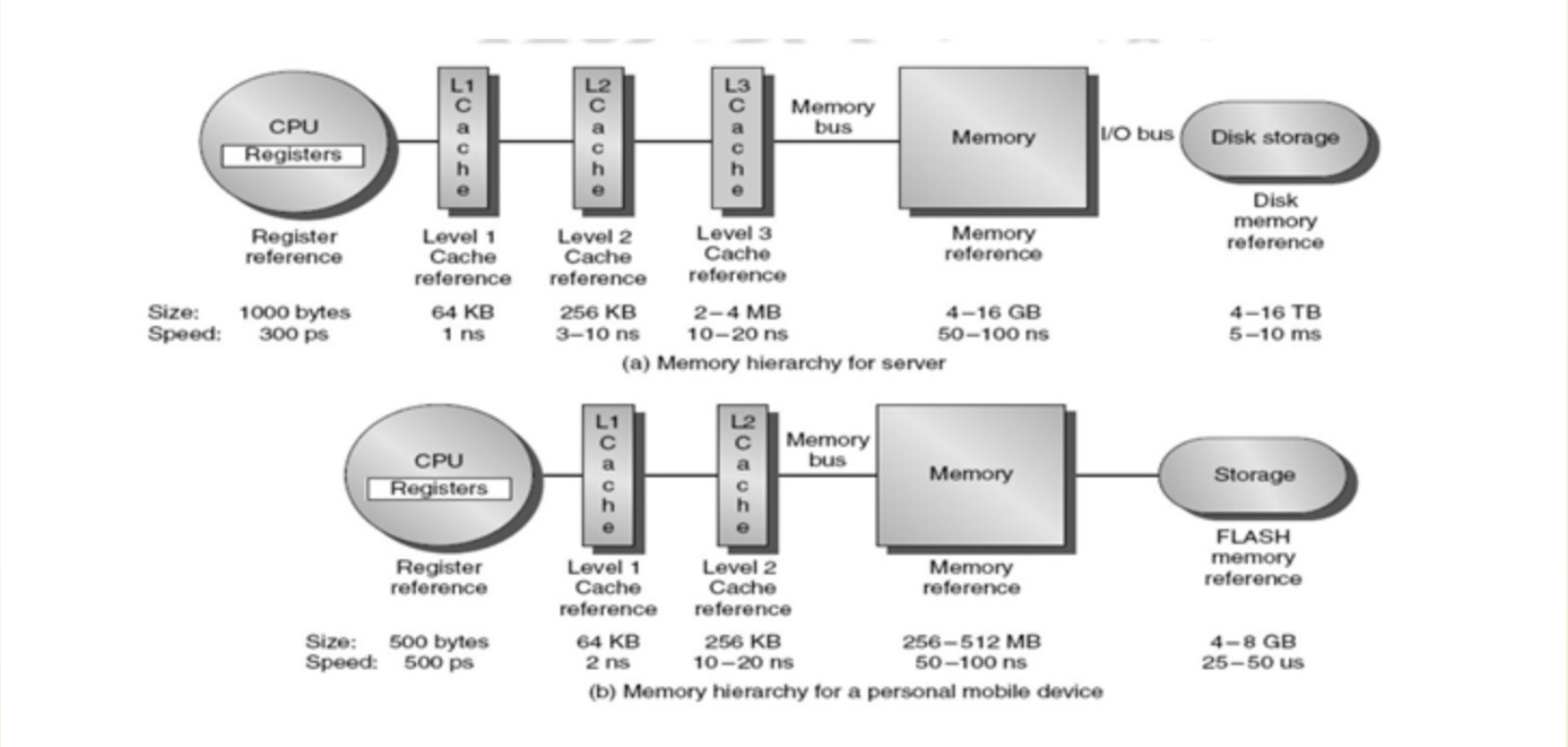
- 按序发射（In Order）
  - 按照程序指令流的逻辑顺序执行
- 乱序发射（Out of Order）
  - 只要操作数和功能单元就绪，就开始执行
  - 乱序发射要解决相关的问题
    - RAW：停顿等待、forwarding
    - WAR/WAW：寄存器重命名
    - 逻辑寄存器/物理寄存器
    - CPU中实现乱序发射执行的硬件算法：记分板/Tomasulo with 寄存器重命名

简明体系结构入门-2.CPU微体系结构(6)

- 多发射
  - 每个周期可以对多条指令取指和执行
  - 硬件要提供足够的pipeline、FU、forwarding网络和冲突检测机制
  - 流水线中同时存在的指令越多，由于分支误预测、跳转地址计算不及时、例外导致的清空流水线造成的损失就越大
  - 发射宽度、乱序程度（乱序的指令窗口大小）、主频（时钟周期的切分）也不是越激进越好

简明体系结构入门-2.CPU微体系结构(7)

- 存储层次



https://www.cs.umd.edu/~meesh/411/CA-online/chapter/memory-hierarchy-design-basics/index.html

简明体系结构入门-2.CPU微体系结构(8)

- cache的组织
  - 指令/数据：Unified or Split, I-D coherent
  - Capacity
  - Line size
  - Way / Association
  - 替换策略：
    - 随机 / LRU / FIFO
  - Write through / Write back
  - Write miss policy：
    - write allocate/no-write allocate
  - Victim Cache
  - 软件是否可控： flush, lock，prefetch level

Index/Tag bit， VIVT/VIPT/PIPT

https://www.cs.umd.edu/~meesh/411/CA-online/chapter/memory-hierarchy-design-basics/index.html

指令集体系结构对编译器后端的影响
1. 从宏观角度看

- 指令集体系结构影响的部分
  - 指令选择
  - 寄存器分配
  - 二进制产生（汇编器/反汇编器）
  - 链接器部分的relaxation（跟寻址模式的offset相关）
  - 部分体系结构相关的优化（Peephole，CSE，Constant Propagation，Strength Reduction,…)
- 微体系结构影响的部分
  - 指令调度
  - 部分体系结构相关的优化（分支的hint，loop内数组的访存优化，loop body的优化）

编译器“100%能”拿到的性能，一般是遵循和实现“指令集体系结构”的规范
但编译器“不一定”拿到的性能，原因大多在微体系结构的部分

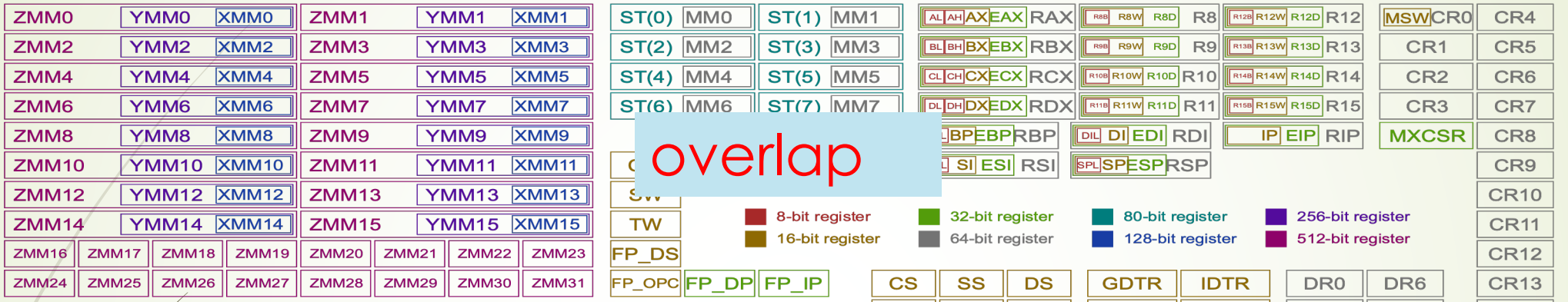
指令集体系结构对编译器后端的影响
2. CISC vs RISC

- CISC指令集：
  - 寄存器-存储器型
  - 变长编码
  - 多种寻址模式
  - 单条指令操作复杂，时钟周期范围较大
  - 指令集包含的指令数较多
  - 代码生成期间的decision比较多
- RISC指令集：
  - load-store类型
  - 定长编码
  - 寻址模式较少
  - 单条指令操作简单，时钟周期短
  - 指令集包含的指令数较少
  - 代码生成策略固定

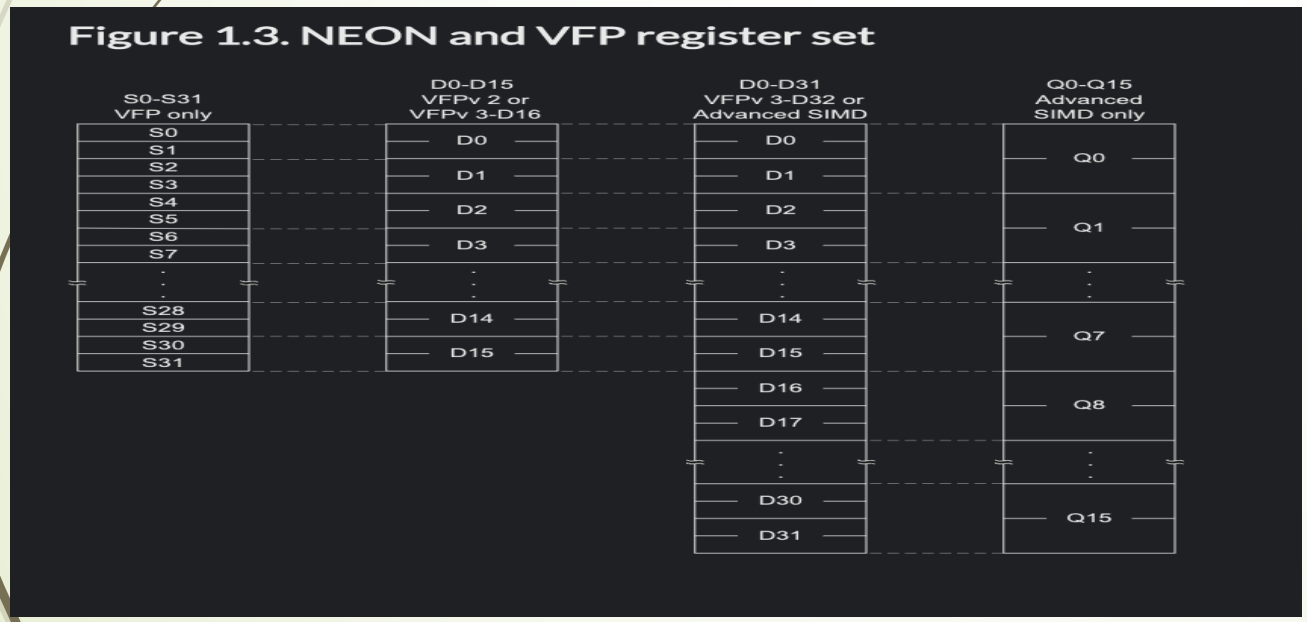
指令集体系结构对编译器后端的影响
3. 寄存器的数量和类型

- 要了解ABI对函数调用和返回时寄存器的使用规定
  - Prologue/Epilogue的产生
- Zero寄存器：hard-wired constant，给编译器提供方便
- 了解寄存器的特殊用法：
  - FP/SIMD Reg的overlap/combine
  - HI/LO寄存器对：早期MIPS使用HI/LO来存储乘法指令产生的翻倍长度的结果，但HI/LO的读写可能会造成资源冲突，导致流水线停顿

寄存器的overlap、combine



overlap



combine

https://en.wikipedia.org/wiki/X86#x86_registers
https://developer.arm.com/documentation/dht0002/a/Introducing-NEON/NEON-architecture-overview/NEON-register s

指令集体系结构对编译器后端的影响
4. 是否存在fused operation指令/操作

- fused multiply-add/ multiply-sub
  - peephole能merge, 但fuse以后精度可能会不同
- 软件不可见的fuse operation
  - 写合并（在连续地址上的写操作会在write buffer中合起来凑够一个cache line之后一起写回低级存储层次）
  - 读关键字优先（读缺失时，优先将缺失地址的内容返回，但位于同一个cache line的其他内容稍后也会就绪）
  - RISC-V会将符合要求乘法指令序列合并，只做一次乘法
    - MULH[[S]U] rdh, rs1, rs2; MUL rdl, rs1, rs2

指令集体系结构对编译器后端的影响
5. 对某些指令子集的特殊支持

- 为了增加指令密度，设计的压缩指令集
  - 压缩指令的编码宽度（16/8bit）一般小于正常指令（32bit）
  - 通过减少offset、imm编码域，或隐含寄存器来实现
- codegen的两种模式
  - 随缘：在正常指令产生完成后，再扫描一遍，把能压缩的都压缩
  - 激进模式：在IR上进行变换、或在寄存器分配阶段增加约束，来产生更多的压缩指令

微体系结构对编译器后端的影响

- 指令的代价描述 制导 指令选择和体系结构相关的优化
  - 总是产生代价较小的代码
- 机器描述 制导 指令调度
  - 面向发射宽度和FU数量进行调度
  - 但在OOO发射机制下，调度的作用不是很大
- 决定优化策略的前提是要对微体系结构的某些细节建立代价模型
  - 例如：将cachehit/缺失的代价，映射到数组不同的访问地址pattern上
- 优化策略在硬件上能发挥作用的前提是代码模型符合实际
  - work on silicon

一些额外需要考虑的Tips

- cache指令
  - flush
  - invalidate
- 在具有温度传感器，能动态调整频率电压的系统
- 在具有任务调度的大小核系统
- 内存控制器的参数调整导致访存性能差异
- 内存条的参数调整导致访存性能差异

https://www.corsi.univr.it/documenti/OccorrenzaIns/matdid/matdid958600.pdf

总结

- ISA（图纸）
  - 是硬件(微架构)和软件（compiler backend)和汇编程序之间的桥梁
  - ISA的设计：既要充分发挥硬件的性能，又要便于软件使用
  - 编译器后端对ISA的支持，跟实际的微架构无关，照着标准文档就可以完成，用静态code size，动态指令count就可以衡量实现的优劣
- 微架构（实现）
  - 真正决定了性能
  - 现代微体系结构的复杂性导致建立精确的、有效的代价模型是很难的
  - 编译器后端调优的难点，强烈依赖于实际的硬件和配置

不想给ISA提意见、给微架构找茬的

编译器后端工程师

不是

好架构师