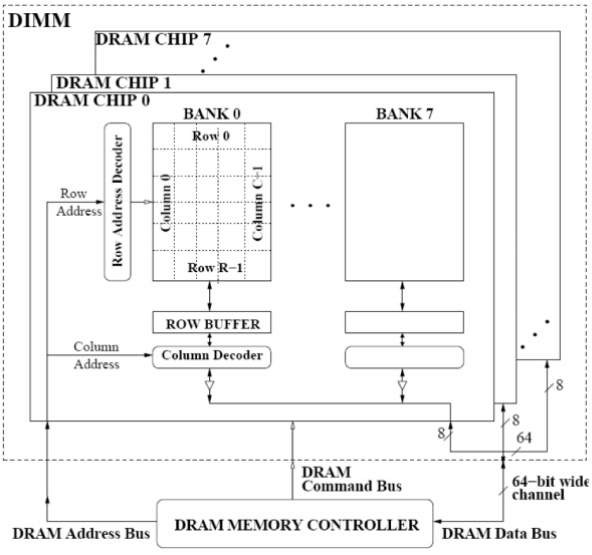
计算机体系结构复习提纲

核心

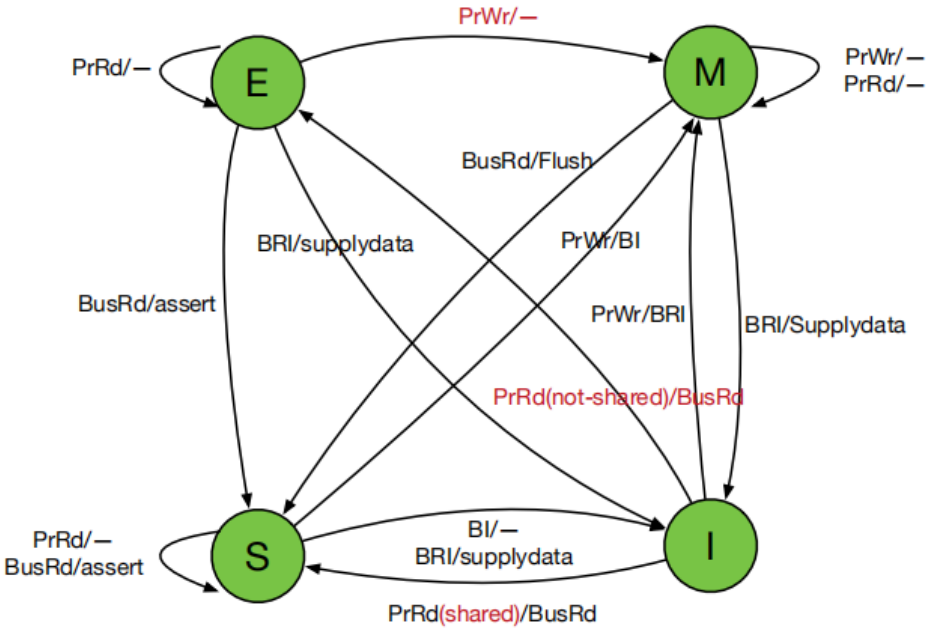
1. $CPI = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$
2. 什么是ISA？什么是微架构？主要的分别标准：程序员能不能看到，如果改变以后程序员写的程序会不会出错？
    1. ISA：指令Instructions、内存Memory、调用Call、中断Interrupt/异常Exception处理访问控制、优先级I/O任务/线程管理功耗多线程、多处理器支持
    2. 微架构：流水线、顺序和乱序指令执行、内存访问调度策略、预测执行（Speculative execution）、超标量（Superscalar）处理（同时发射多条指令）、时钟门（Clock gating）、缓存：如层次、大小、关联、置换策略、预取（Prefetching）等
    3. 特殊情况：指令的执行周期：大多数情况下是微架构的，有特殊情况如VLIW会要求程序员知道，在编译的时候知道执行几个周期进行优化，是ISA
    4. 错题：八个通用寄存器：是ISA不是microarchitecture
3. RISC is motivated by：内存暂停、简化硬件设计、允许编译器更好地优化代码
    1. CISC：Complex Instructions Set Computer
    2. RISC：Reduced Instructions Set Computer
4. 流水线：表（横轴周期数、纵轴指令）、计算
5. 流水线冒险Pipeline Hazards
    1. 结构冒险：硬件不能同时支持所有可能的指令组合
        1. 解决方案：消除争用，例如把数据cache和指令cache分开
    2. 数据冒险：指令取决于仍在流水线中的前一条指令的结果
        1. 基础的5段流水线不会出现WAW和WAR
        2. 解决方案
            1. 检测并等待：直到值出现在寄存器中
            2. 检测并转送：后一段转到前一段
            3. 检测并消除：编译器调整指令顺序来消除依赖等
    3. 控制冒险：由指令获取和控制流更改决定之间的延迟引起的
        1. 解决方案
            1. Stall，拖延流水线
            2. 分支预测：猜测下一个fetch地址
                1. 三个问题：
                    1. 当前取的指令是不是branch——BTB可以做到
                    2. 条件跳转会不会跳——怎么记录跳转方向？
                    3. 跳到哪里——BTB可以做到
                2. BTB：Branch Target Buffer
                3. 怎么记录跳转方向：上次跳到哪里（2bit Counter，又称Saturating Counter）
                4. 改进：
                    1. 全局：记录全局的跳转历史
                        1. Global History Register (GHR) ：记录所有分支的全局T/NT索引

        2. Pattern History Table (table of 2-bit counters)：用GHR所引到PHT，来看最近跳不跳
      2. 本地：每个branch拥有一个自己的分支历史寄存器
      3. 混合：全局+本地，有一个来选择
      4. Call/Return类型的跳转可以在Call时把Return地址记录到栈
    3. 预测执行：消除控制流指令
    4. 分支延迟槽：部署延迟执行的指令，但是缺点是暴露了微架构，所以现代处理器不采用
    5. 细粒度多线程
    6. 多路径执行：从两个路径都取
6. 阿姆达尔定律：$speedup = \frac{1}{1 - f + \frac{f}{N}}$
7. 超标量：
    1. 一个周期取多条指令
    2. 依赖解除
        1. 数据依赖：RAW，WAW，WAR
            1. RAW：使用Scoreborad的Out-of-Order execution解决
            2. WAW、WAR：Tomasulo's algorithm
        2. 控制依赖：
            1. 静态调度：如循环展开
            2. 判定式执行
            3. 分支预测
    3. 多个功能单元
    4. Completion：乱序结束但顺序commit
8. Scoreboard和Tomasulo
    1. Scoreboard
        1. 例子和作业过一遍了吗?
        2. 缺点：会在WAR和WAW上stall
    2. Tomasulo
        1. 本质：OoO with Register Renaming
        2. 例子和作业都过一遍了吗?
        3. 真的都能推出来了吗?
    3. 相同点：顺序issue，乱序执行和结束
    4. 不同点：结构冒险，RS中更多的等待空间；Tomasulo解决WAR和WAW问题
9. ROB
    1. Tomasulo缺点：不支持精确中断，因此Reorder Buffer（ROB），实现IOOI，最后顺序commit
    2. ROB缺点：关键路径、复杂的访问
    3. 继续改进：History Buffer（恢复的时候读寄存器旧值）、Future File（两套寄存器文件）
10. Load/Store Queue相关
    1. LQ+SQ：
        1. 要Store的时候，如果它在ROB头，总是直接Store
        2. 要Load的时候，把地址发送给Store队列，查询是否有一样的地址；查出来以后，比一下Store的年龄是不是比自己更老

        1. 更老：说明是自己要读的数据，需要等待
    3. 需要的数据结构：CAM（Counted Associative Memory），可以同时和Store里所有地址比较
2. 改进：forwarding，让Load抢先执行，而不用先Store存到内存再Load访问内存了
3. 再改进：直接load，如果冲突：
    1. flush重新执行
    2. 选择依赖的指令重新执行——需要指令集支持
11. 同步多线程SMT：允许来自多个线程的指令以相同的时钟周期进入执行
12. DLP
    1. Vector Machine
        1. Vector Chaining：向量版本的forwarding
    2. SIMD
    3. GPU：表面SPMD实际SIMD
        1. GPU的Latency高，但是Throughput高，所以适应于训练，而不一定适应于推理
13. VLIW：编译器打包的多个独立指令
14. static scheduling
    1. Trace Scheduling
        1. Trace：控制流图中经常执行的路径
        2. 保证：为整个trace建立data优先图，模拟执行并分配寄存器，最后加入补偿代码保证语义正确
15. TLP
16. Recursive latency equation：$T_i = t_i + m_i \times T_{i+1}$，即i层cache时间=i层cache命中cycle+miss rate*i+1层cache时间。是一个递归的式子
17. cache
    1. 组织：全相联、组相联
        1. 例如通过访存地址序列和cache命中率来判断cache的属性
    2. victim cache：保存被换出的，防止ping-pong访问
18. Memory
    1. 

19. 并行的瓶颈（一定要背！！！！！！！！）
    1. 同步：操作共享数据的操作不能并行化
        1. 如锁、互斥（临界区）、同步、通信

2. 负载不均衡：并行任务可能有不同的负载
    1. 异构、微架构的影响
3. 资源争用：并行任务可以共享硬件资源，互相延迟
    1. 拷贝全体资源很昂贵，当每个任务单独运行时多余的延迟将不会被呈现

20. Cache coherence
    1. Snoopy Bus：基于抢占总线的，对所有内存请求的全局的序列化点
        1. VI协议：Valid和Invalid状态，是write through的：不停转换且需要不停写回内存，就效率低
        2. MSI协议：Modified、Shared和Invalid状态，问题在如果只有一个processor读块，没必要告诉别人，浪费带宽
            1. Modified：cache块是唯一的cached拷贝，并且它是**脏**的
            2. Shared：cache块是潜在的一份cached拷贝
        3. MESI协议：Modified、Exclusive、Shared和Invalid状态，
            1. Exclusive：cache块是唯一的cached拷贝，并且它是**干净**的
            2. 

            3. Modified状态写要到总线上Invalidate，但Exclusive状态写的时候不需要到总线Invalidate，自己偷偷invalidate即可，节省总线带宽了

4.

当前状态	事件	行为	下一个状态
I	Local Read	如果其它Cache没有这份数据，本Cache从内存中取数据，Cache line状态变成E；如果其它Cache有这份数据，且状态为M，则将数据更新到内存，本Cache再从内存中取数据，2个Cache 的Cache line状态都变成S；如果其它Cache有这份数据，且状态为S或者E，本Cache从内存中取数据，这些Cache 的Cache line状态都变成S	E/S
I	Local Write	从内存中取数据，在Cache中修改，状态变成M；	M
I	Remote Read	如果其它Cache有这份数据，且状态为M，则要先将数据更新到内存；既然是Invalid，别的核的操作与它无关	I
I	Remote Write	如果其它Cache有这份数据，则其它Cache的Cache line状态变成I既然是Invalid，别的核的操作与它无关	I
E	Local Read	从Cache中取数据，状态不变	E
E	Local Write	修改Cache中的数据，状态变成M	M
E	Remote Read	数据和其它核共用，状态变成了S	S
E	Remote Write	数据被修改，本Cache line不能再使用，状态变成I	I
S	Local Read	从Cache中取数据，状态不变	S
S	Local Write	修改Cache中的数据，状态变成M，其它核共享的Cache line状态变成I	M
S	Remote Read	状态不变	S
S	Remote Write	数据被修改，本Cache line不能再使用，状态变成I	I
M	Local Read	从Cache中取数据，状态不变	M
M	Local Write	修改Cache中的数据，状态不变	M

当前状态	事件	行为	下一个状态
M	Remote Read	这行数据被写到内存中，使其它核能使用到最新的数据，状态变成S	S
M	Remote Write	这行数据被写到内存中，使其它核能使用到最新的数据，由于其它核会修改这行数据，状态变成I	I

2. Directory：很大的内存空间划分为block，一个block由Directory分配负责人，Directory决定谁先谁后
3. Snoopy vs Directory
    1. Snoopy Cache
        1. 优点：miss延迟低、全局序列化简单（bus提供了序列化）
        2. 缺点：广播消息，导致可拓展性差
    2. Directory Cache
        1. 优点：不需要对所有cache的广播、可拓展性好
        2. 缺点：
            1. 增加关键路径而增加miss延迟request->dir.->mem
            2. 需要额外的存储空间
            3. 协议和竞争条件更复杂

21. Memory Consistency
    1. 原子操作
    2. RISC-V：
        1. aq：没有后来的内存操作可以在这个指令前执行
        2. rl：所有先前的内存操作都需要全部完成，才能执行这条指令
22. Interconnection
    1. 路由距离：两点之间的跳数
    2. 直径：任意两点之间的最大路由距离
    3. 平均距离
    4. 对分带宽（Bisection Bandwidth）：用一截面将网络划分为对等的两半时（或者两个节点数目都相同的子网时），穿过该截面的最大传输率
    5. 路由器的度
    6. 2D Mesh为例：
        1. 直径：$2\sqrt{N}-2$，对分带宽：$\sqrt{N}$，路由器的度：4
        2. 都会怎么算了吗？

Network	# Nodes	Router Deg.	Diameter	Bisection BW	# Links
1D Torus	N	2	$\lfloor N/2 \rfloor$	2	N
2D Mesh	$\sqrt{N} \times \sqrt{N}$	4	$2\sqrt{N}-2$	$\sqrt{N}$	$2(N-\sqrt{N})$
2D Torus	$\sqrt{N} \times \sqrt{N}$	4	$2\lfloor \sqrt{N}/2 \rfloor$	$2\sqrt{N}$	$2N$
Full Mesh	N	$N-1$	1	$N^2/4$	$N(N-1)/2$
Binary Tree	N	3	$2(\lceil \log_2 N \rceil - 1)$	1	N-1
Hyper Cube	$N=2^n$	n	n	N/2	$nN/2$

23. CAP理论：不支持超过任意两个的以下保证

1. 一致性：每个读取都收到最近的写入或错误
2. 可用性：每个请求都收到一个（非错误）响应——不保证它包含最近的写入
3. 分区容差：尽管节点之间的网络将任意数量的消息丢弃（或延迟），系统仍在继续运行

本课程作业题涉及的内容

1. CPI计算
2. 不同数量address的ISA在一个序列下的code size和memory bandwidth
3. 芯片面积优化
4. MIPS汇编的填充和空间计算、执行指令数量计算
5. BTB（Branch Target Buffer）
    1. FDXMW表格
    2. 循环次数
    3. CPI
6. BHT（Branch History Table）及2-bit counter相关
7. BHR（Branch History Register）、PHT（Patter History Table）
8. Superscalar：表格
9. Tomasulo算法！一定要熟悉
10. Load Store相关
11. 向量汇编语言（SIMD）
12. VLIW
13. static scheduling：基本块优化

CMU课程作业后续涉及的内容

1. cache属性推断（例如block size、映射、写回方式等等）
2. memory：冲突的吞吐量计算、非冲突的吞吐量计算
3. 预取相关的coverage、accuracy计算及预取数量N的推断
4. GPU和SIMD
5. cache一致性
6. Amadahl定律
7. Systolic Array