Chapter 1：

一、性能衡量

1. Latency/Responce Time：同一项任务从开始到结束的时间 2. Bandwidth/Throughput：一定时间内进行的工作总量 3. Performance = 1/Exe Time

4. CPU Time：CPU的实际执行时间，不含等待I/O等时间，与之对应的是Elapsed Time

CPU Time = Clock Cycles × Clock Cycle Time

Clock Cycles = Inst. Count × CPI

5. Amdahl's Law

① $T_{improved} = \frac{T_{affected}}{\text{加速比}} + T_{unaffected}$

② $\text{Speedup} = \frac{1}{(1-F) + F/\text{局部加速比}} < \frac{1}{1-F}$

二、八条设计思想

1. Design for Moore's Law 2. Use abtraction(抽象) to simplify design 3. Make common case fast 4. Improve performance via parallelism 5. … via pipelining 6. … via prediction 7. Use Memory Hierarchy 8. Improve dependability via redundancy

三、ISA

1. Class of ISA

① Stack Architecture：隐式操作数在栈顶(TOS)，操作数在栈中弹出，结果压入栈中

② Accumulator Architecture：1个隐式操作数(累加器)，1个显式操作数(内存地址)，累加器同时也是结果 ③ General-Purpose Register Arch：只支持显式操作数 I. Reg-Mem Arch：任何指令都能访问内存 II. Load-Store Arch：仅load、store可访(80x86是Reg-Mem，ARM、RISC-V是Load-store)

2. 地址对齐(Addr. Alignment)：设对象大小s字节，其内存地址为A，若A%s=0则称地址对齐

四、Flynn分类

1. SISD：单处理器，ILP 2. SIMD：多处理器，DLP，ILP 3. MIMD：ILP、DLP、TLP

五、功耗公式

1. 单位：Power (W，1W=1J/s)，Energy (J)

2. CPU耗能 { Dynamic Energy (主要，来自开关晶体管) / Static Energy (来自泄露电流)

0→1或1→0：$\text{Energy-Dynamic} \propto \frac{1}{2}\text{Capacitive Load} \times \text{电压}^2$，$\text{Power\_Dynamic} \propto \frac{1}{2}\text{Capacitive Load} \times \text{电压}^2 \times \text{切换频率}$ (对同一任务，降低时钟频率可减小Power，但无法减小Energy)，$\text{Power\_Static} = I_{static} \times V$ (与器件数目成正比)

六、Dependability

1. Fault是设计的瑕疵，Fault存在时，会产生潜在的Error，Error影响实际交付的服务时，就发生Failure

2. Reliability ① MTTF，MTTR ② MTBF (Mean Time Between Failure) = MTTF + MTTR

3. Availability = MTTF/(MTTF+MTTR)

4. MTTF的倒数表示故障率(FIT)

e.g. 10个MTTF为10^6h的Disk，和1个MTTF为10^6h的ATA，求总MTTF：先求总故障率 $10 \times \frac{1}{10^6} + \frac{1}{10^6}$ 再倒数

七、集成电路成本

1. 晶片(Dies)由晶圆(Wafer)切割而成

2. $\text{Dies per wafer} = \frac{\pi \times (\text{Wafer Diameter}/2)^2}{\text{Die Area}} - \frac{\pi \times \text{Wafer Diameter}}{\sqrt{2 \times \text{Die Area}}}$

3. $\text{Cost of Die} = \frac{\text{Cost of Wafer}}{\text{Dies per wafer} \times \text{Die Yield (良率)}}$

4. $\text{Die Yield} = \text{Wafer Yield} \times \frac{1}{(1 + \text{Defects per unit area} \times \text{Die area})^N}$

八、补充

Normalized Geometric Mean具有统一结果，不论机器

Chapter 2：Memory Hierarchy

一、定量计算

1. Latency：从内存块读出第一个块的时间(难优化)

Bandwidth：读出内存块剩余部分的时间

2. AMAT = Hit Rate × Hit Time + Miss Rate × Miss Time
= (1 − Miss Rate) × Hit Time + Miss Rate × (Hit Time + Miss Penalty) = Hit Time + Miss Rate × Miss Penalty
= $\text{Hit Time}_{L_1} + \text{Miss Rate}_{L_1} \times (\text{Hit Time}_{L_2} + \text{Miss Rate}_{L_2} \times \text{Miss Penalty}_{L_2})$

3. Local Miss Rate = 该级Cache Miss次数/该级Cache访问次数 Global Miss Rate = …

4. CPU Time = (CPU Clock Cycles + Mem. Stall Cycles) × Clock Cycle Time，其中 Mem. Stall Cycles = Miss数 × Miss Penalty = IC × Mem指令占比 × Miss Rate × Miss Penalty

e.g. 假设理想CPI 1.1，30%指令为MEM，有10%的MEM指令发生Miss，Miss代价为50 cycles；有1%的指令发生Miss，Miss代价为50 cycles，求CPI、AMAT

解：CPI = 1.1 + 30% × 10% × 50 + 1% × 50 = 3.1

$\text{AMAT} = \frac{1}{1.3} + [1 + 1\% \times 50] + \frac{0.3}{1.3} + [1 + 10\% \times 50] = 2.56$

二、设计思想

1. 局部性原理：Temporal / Spatial Locality

2. Unified Cache：命中率低

Split Cache：指令与数据一级Cache分开(二级不分)

3. Miss的种类 ① Compulsory：冷启动/数据第一次被访问的Miss (可通过预取降低) ② Capacity：Cache已满时，一些数据被替换出去，再访问这些数据的Miss ③ Conflict：在组相联/直接映射中，多个块被映射到同一个Cache组中，导致频繁替换

4. VIPT：使用Page Offset (虚拟地址和物理地址的共同部分)索引L1 Cache，从而可在访问TLB的完成对L1 Cache的访问

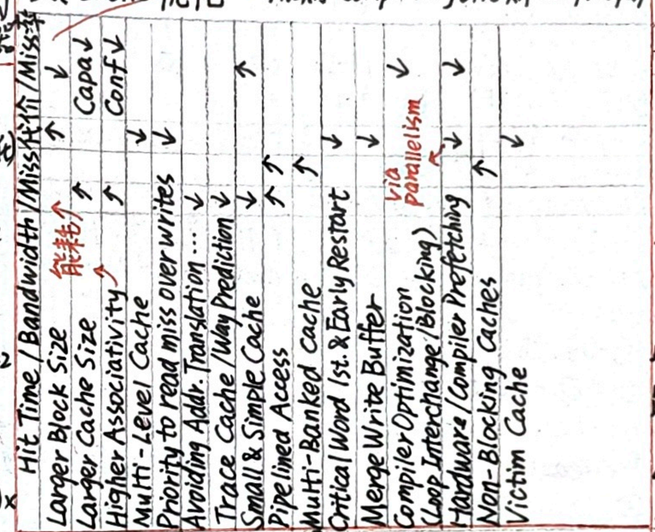
5. 多级Cache：L1足够小，以减小Hit Time，L2足够大，以减小L1 Miss Penalty

6. 写策略 ① Write-Through：同时写进Cache和Mem，Mem总是最新的，Cache无dirty位 ② Write-Back：只写Cache，该块要被替换时才写回Mem

处理Write Miss：① Write Allocate：Miss的块先拿到Cache中，再正常写入 (Write-Back只能用此方法) ② No Write Allocate (Write Around)：不经过Cache，直接写入Memory (常搭配Write-Through)

7. Interleave：在读写大量/分散性数据时，需对不同bank连续操作，通过合理控制，保证各bank传输周期相连贯

三、Cache优化 (U-Shape) 初期：Compulsory↓，后期：Conf/Capa↑



Chapter 3：Pipelining & ILP

一、为流水线添加功能单元

1. 延迟(Latency)：使用数据与得出结果的时钟周期数，常为流水线级数−1；启动间隔(Initiation/Repeat Interval)：向同一FU发出2个操作之间必须间隔的周期数 e.g. FP Add：L(3)，I(1)，DIV：L(24)，I(25)

二、冒险情形与处理

1. Structural Hazard：多条指令竞争FU、数据读写口

2. Data Hazard

① Data dependence：RAW，真依赖

② Name dependence：WAR，WAW，假依赖

1. Antidependence：WAR，指令A读取数据后，指令B向同一寄存器写入数据，A可能会错误读取到B写入的值 II. Output Dependence：WAW，两指令都向同一寄存器写入数据，可能前一条指令覆盖后一指令写的值

3. 寄存器重命名：用于解决Name Dependence，WAR、WAW分别对后面/前面的寄存器重命名

e.g.
```
mul  x2,x2,x2        mul p7,p2,p2
add  x1,x1,x2   ⇒    add p8,p1,p7
mul  x2,x4,x4        mul p9,p4,p4
```

4. Control Dependence

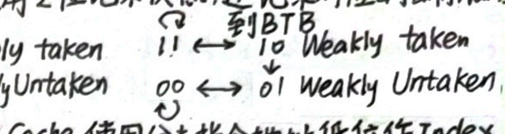
5. 实现前递时，若EX Hazard与MEM Hazard同时条件满足，只执行EX前递

6. 静态 — Compiler，动态 — Hardware

三、控制冒险与分支预测(动态解决)

1. 2-bit predictor

① 使用一个分支预测缓冲区(BHT)，在BHT中，每条分支指令使用2位记录状态，还记录对应的目标Addr



② BHT形似Cache，使用分支指令地址低位作Index，高位作tag. ③ 引申N-bit predictor：若BHT记录值 ≥ $(2^N-1)/2$，则预测为Taken

Is Inst. in BTB ?	Predict	Reality	Delay Cycle
√	Taken	Taken	0
√	Taken	Not Taken	2
×	Not Taken	Taken	2
×	Not Taken	Not Taken	0

2. Local Predictor

分支指令跳转结果可能与该条分支指令近几次跳转结果为其进行预测，为每个指令安排一个n位寄存器，记录其最近n次的跳转结果，再安排2^n个2-bit Predictor进行学习

3. Global Predictor (Correlate Branch Predictor)

安排一个m位移位寄存器，记录所有分支指令近m次跳转结果，再安排2^m个n-bit predictor，称为(m,n) Predictor，其位数为$2^m \times n$

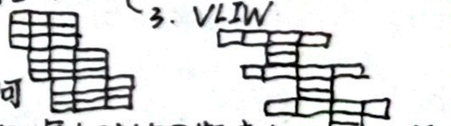
4. Tournament Predictor竞争预测器同时含Local、Global预测器，使用2-bit predictor从中选择结果

四、多发射

1. 多发射可使CPI<1

2. Superscalar 静态、动态都可

3. VLIW



3. Super Pipeline：每个时钟周期流出n条指令，每一级流水线用时$\frac{1}{n}$

五、精确异常

1. 确保异常指令前的所有指令完成执行，异常指令及之后的指令尚未执行

2. mepc：异常指令的地址，mtvec：异常处理程序的入口地址，mtval：存储异常相关信息

六、ScoreBoard 若不允许发射，则它及它的后续指令都无法进入IS，Tomasulo也这样

1. 算法流程

① IS：允许发射的条件：I. FU空闲(避免结构冲突) II. 正在执行的其它指令与该指令的目标寄存器都不相同(避免WAW) ② RO：两个源寄存器都准备好了才读数，进入EX (避免RAW真依赖) ③ WB：写回时看目标寄存器是否在某个FU的Ready List中为yes，若是，说明有指令在读寄存器，等读完再写回(避

免WAR).
2. 由于ScoreBoard通过停顿的方式处理WAR/RAW, 所以说并未解决这些冒险; 升级版可显式Reg.重命名
3. 按序发射, 乱序完成, 不能实现精确中断
4. 三张状态表
① Inst. Status Table: 记录指令所处阶段 IS/RO/EX/WB
② FU Status: 每个FU对应如下记录:
Busy, Op: 正在执行的指令类型
Fi, Fj, Fk: 目标(i)、源(j,k)寄存器号
Qj, Qk: Fj, Fk若还没准备好, 应该从哪个FU的目标寄存器读取, 添写FU的名字
Rj, Rk: 指示源寄存器的读取情况, 刚准备就绪且尚未读取时为Yes, 若还未准备就绪或已读取则为No. 要求两个操作数都就绪后一起读取.
③ Register Status Table: 若某个FU以该寄存器为目标, 则填FU的名字, 否则留空

e.g. (MUL:10, ADD:2, SUB:2, LD:1, DIV:40)

		IS	RO	EX	WB
FLD	F6, 34(R2)	1	2	3	4
FLD	F2, 45(R3)	5	6	7	8
FMUL	F0, F2, F4	6	9	10-19	20
FSUB	F8, F6, F2	7	9	10-11	12
FDIV	F10, F0, F6	8	21	22-61	62
FADD	F6, F8, F2	13	14	15-16	22

七、Tomasulo's Algorithm
1. 算法流程 (分为IS, EX, WB)
① IS: 保留站有空余时, 从指令序列头部取指令发送到保留站(Reservation Center) ② EX: 全部操作数可用时, 将指令送入功能单元执行; 若遇分支指令, 在分支指令执行完前, 不准执行任何指令.
③ WB: 将结果放到CDB上广播
2. 三张状态表 (仅第二张不同):
ROB Status: 每个ROB对应如下记录:
Busy、Op、Vi, Vk: 操作数的真实值
Qi, Qk: 将产生Vi, Vk结果的保留站, 若操作数已产生并送入保留站, 则置空 (即: Qi、Vi只能有一个具有有效值)
A: 对于load, store指令, 存放Mem. Addr.计算的信息. 初始时存放立即数, 地址计算完后存放实际的Effective Addr. (使用Vk存放计算地址所用的寄存器值)
3. 实现了隐式的Reg.重命名, 不能实现精确中断

e.g

		IS	EX	WB
FLD	F6, 34(R2)	1	3	4
FLD	F2, 45(R3)	2	4	5
FMUL	F0, F2, F4	3	6-15	16
FSUB	F8, F6, F2	4	6-7	8
FDIV	F10, F0, F6	5	17-56	57
FADD	F6, F8, F2	6	9-10	11

在Tomasulo中, Load指令的IS和EX之间要停一拍.

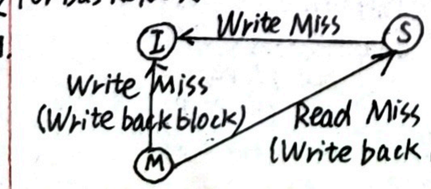
八、Hardware Speculation (Tomasulo with ROB)
1. ROB的条目字段
① Inst. Type: 存储指令类型: I. 分支指令(无目的地) II. Store指令(目的地为存储器地址) III. 寄存器指令(Load/ALU)(目的地是指令的目标寄存器号)
② Destination (目的地) ③ Value: 结果值
④ Ready: 表明指令完成执行, 结果值就绪但还未提交
2. 算法流程
Commit: 提交预测错误的分支指令时, 清空ROB, 执行过程从后续正常指令处重新开始
3. 顺序提交, 实现了精确中断

e.g.

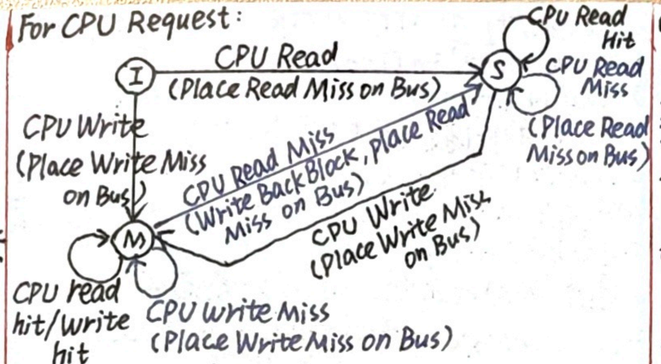
	IS	EX	WB	COMMIT
FLD F6, 34(R2)	1	3	4	5
FLD F2, 45(R3)	2	4	5	6
FMUL F0, F2, F4	3	6-15	16	17
FSUB F8, F6, F2	4	6-7	8	18
FDIV F10, F0, F6	5	17-56	57	58
FADD F6, F8, F2	6	9-10	11	59

Chapter 4: TLP
一、两类多处理器架构 每个核有独立的Cache
① Centralized Shared Memory (SMP/UMA): 所有处理器统一共享物理内存, 所有处理器访问任何内存字的时间相等 ② Distributed Shared Memory (DSM/NUMA): 每个核有独立的内存, 但地址空间是统一的, 访问远程内存较慢.
二、缓存一致性问题 Cache Coherence Problem (同一内存位置)
1. Coherence: 规定读操作应当返回何值
① P. read(x) → P. P. write(x), 总返回P写入的值
② P1. read(x) → P2. write(x), 返回P2写入的值
③ Write Serialization
2. Consistency: 针对内存(不同内存位置), 决定写操作的值何时能被读操作返回. 内存一致性模型:
① Sequential Consistency: 读、写顺序都与程序一致
② Total Store Order: 读乱序, 写顺序
③ Weak Order: 读、写都乱序, 使用Mem. Barrier/Fence来强制顺序.
3. UMA使用Snoopy Protocol: 所有处理器连总线, 写Cache时在总线上发送Invalid信号; NUMA使用Directory Protocol, 使用Directory记录哪些处理器在Cache中有特定block的拷贝, 修改时, 通过Directory以点对点方式通知
4. Write-Invalidate / Write-Broadcast

三、Write Invalidate Protocol
1. MSI: Invalid / Shared / Modified
For Bus Request:



For CPU Request:



2. MESI
Exclusive: 该块只存在于一个Cache中, 且是clean的, 该状态设置于某个块刚被读进来时
E —(Read by Others)→ S
E —(Local Write)→ M, 且无需在总线上发送 invalidate信号
3. MOESI
Owned: 表明该块被Cache所有且为dirty

四、True/False Sharing Miss
1. True Sharing Miss
① 处理器首次写入某个Share状态的块, 致使其它Cache中该块失效 ② 另一处理器读该块中的一个被修改的字
2. False Sharing Miss: 一个块为Invalidated, 此时读该块中的某个字, 然而Invalidate是由于对块中另一个字的修改造成的; 或者由于写操作需要对块Invalidate, 然而写入的字和另一处理器在用的字不是同一个
五、Directory Protocol
三个状态: ① Shared: 一或多个处理器Cache了该块且内存中为最新值 ② Uncached: 无处理器Cache了该块 ③ Modified: 恰一个处理器Cache了该块, 内存中不是最新值
2. 概念: ① Local Node: 发起请求的处理器 ② Home Node: 对应内存地址所在的处理器 ③ Remote Node: 拥有这个副本的处理器
3. 消息 (P: Processor, A: Address)

Type	Source	Destination	Content
Read Miss	L	H	P, A
Write Miss	L	H	P, A
Invalidate	L	H	A
Invalidate	H	R	A
Fetch	H	R	A
Fetch & Invalidate	H	R	A
Data Value Reply	H	L	Data
Data Write Back	R	H	A, Data

Chapter 5: DLP
1. Loop Carried Dependence
主要看循环迭代之间是否有数据依赖, 通过改写消除循环间依赖以实现向量化
2. GPU
① Single Inst. Multiple Threads (SIMD)
② Thread: 线程, Block: 线程块, Grid: 线程块的集合 ③ GPU Memory为所有Grid共用, Local Memory为一个Block所用, Private Memory为一个CUDA Thread所用
3. Horizontal Method/ Vertical Method:
Mem-Mem Structure
Grouping Method: Reg-Reg Structure
4. Vector Chaining
① 对于前后依赖的指令: 若无Chaining, 则前一指令最后一个分量完成后才可开始下一指令; 若有Chaining, 第一个分量完成就可开始
② Convey: 一组可在一起执行的指令, 可以有数据依赖, 但不能有 structual conflict.
Chimes: 执行单个 convey的时间
④ 链接时, 功能单元的数据进出各需1拍
⑤ 不同 convey之间是顺序执行的.