

Chapter 1: 一、性能衡量

1. Latency/Response 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 = $1/\text{Exe Time}$
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

$$\text{CPU Time} = \text{Clock Cycles} \times \text{Clock Cycle Time}$$

$$\text{Clock Cycles} = \text{Inst. Count} \times \text{CPI}$$

5. Amdahl's Law

$$\textcircled{1} \quad T_{\text{improved}} = T_{\text{affected}} / \text{加速比} + T_{\text{unaffected}}$$

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

二、八条设计思想

1. Design for Moore's Law
2. Use abstraction (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, ILP, DLP
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)}

$O \rightarrow 1$ 或 $1 \rightarrow 0$: Energy-Dynamic $\propto \frac{1}{2} \text{Capacitive Load} \times$
电压², Power-Dynamic $\propto \frac{1}{2} \text{Capacitive Load} \times \text{电压}^2 \times t$
换频率 (对同一任务，降低时钟频率可减小 Power,
但无法减小 Energy), Power-Static = $I_{\text{static}} \times V$
(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 10^6 h的Disk, 和1个MTTF为 10^4 h的
5. ATA. 求总MTTF: 先求总故障率 $10 \times \frac{1}{10^6} + \frac{1}{10^4}$ 再倒数
6. 集成电路成本
1. 晶片 (Dies) 由晶圆 (Wafer) 切割而成
 2. Dies per wafer = $\frac{\pi \times (\text{Wafer Diameter}/2)^2}{\text{Die Area}} - \sqrt{2 \times \text{Die Area}}$
 3. Cost of Die = $\frac{\text{Cost of Wafer}}{\text{Dies per wafer} \times \text{Die Yield}}$ (良率)
 4. Die Yield = Wafer Yield $\times \frac{1}{(1 + \text{Defects per unit area} \times \frac{\text{Die area}}{\text{Wafer area}})}$
7. 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 (难以量化)
- Bandwidth: 读出内存块剩余部分的时间
2. AMAT = Hit Rate \times Hit Time + Miss Rate \times Miss Time
 $= (1 - \text{Miss Rate}) \times \text{Hit Time} + \text{Miss Rate} \times (\text{Hit Time} + \text{Miss Penalty}) = \text{Hit Time} + \text{Miss Rate} \times \text{Miss Penalty}$
 $= \text{Hit Time}_{L_1} + \text{Miss Rate}_{L_1} \times (\text{Hit Time}_{L_2} + \text{Miss Rate}_{L_2} \times \text{Miss Penalty}_{L_2})$
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) \times Clock Cycle Time. 其中 Mem. Stall Cycles = Miss 数 \times Miss Penalty = IC \times Mem 指令占比 \times Miss Rate \times 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% \times 10% \times 50 + 1% \times 50 = 3.1

$$\text{AMAT} = \frac{1}{3.1} + [1 + 1\% \times 50] + \frac{0.3}{1.1} + [1 + 10\% \times 50] = 2.54$$

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

Hit Time / Bandwidth / Miss 代价 / Miss 率		
Larger Block Size	↑ 能耗 ↑	↓
Larger Cache Size	↑	Capa ↓
Higher Associativity	↑	Conf ↓
Multi-Level Cache		↓
Priority to read miss over writes		↓
Avoiding Addr. Translation ...		↓
Trace Cache / Way Prediction		↓
Small & Simple Cache	↓	↑
Pipelined Access	↑ ↑	
Multi-Banked Cache	↑	
Critical Word 1st. & Early Restart		↓
Merge Write Buffer		↓
Compiler Optimization (Loop Interchange / Blocking)	via parallelism ↗	↓
Hardware / Compiler Prefetching		↓
Non-Blocking Caches	↑	
Victim Cache		↓

前完成对 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 传输周期相连接 (U-Shape)

三、Cache 优化 → 初期: Compulsory ↓, 后期: Conf/Cap ↑

Chapter3 : 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

I. 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条指令写的值

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

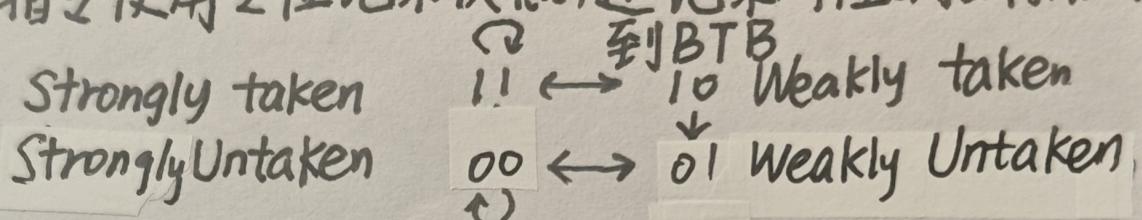
5. 实现前递时，若 EX Hazard 与 MEM Hazard 同时条件满足，只执行 EX 前递

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 $\geq (2^n - 1)/2$ ，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

免WAR).

2. 由于ScoreBoard通过停顿的方式处理WAR/RRAW,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

F_i, F_j, F_k: 目标(i). 源(j,k)寄存器号

Q_j, Q_k: F_j, F_k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

R_j, R_k: 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.

FLD F6, 34(R2)

	IS	RO	EX	WB	MUL:10
FLD F6, 34(R2)	1	2	3	4	ADD:2
FLD F2, 45(R3)	5	6	7	8	SUB:2
FMUL F0, F2, F4	6	9	10-19	20	LD:1
FSUB F8, F6, F2	7	9	10-11	12	DIV:40
FDIV F10, F0, <u>F6</u>	8	21	22-61	62	
FADD <u>F6</u> , F8, F2	13	14	15-16	22	

→ 采用流水线Algoithm

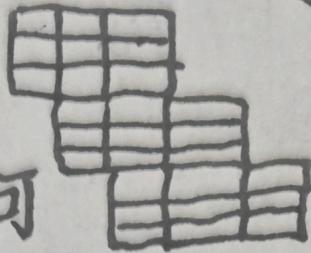
四、多发射

1. 多发射可使 $CPI < 1$

2. Superscalar

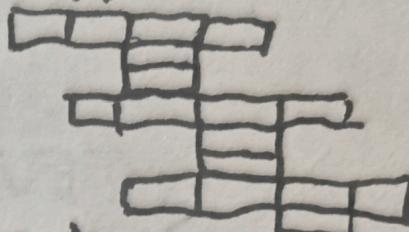


静态、动态都可

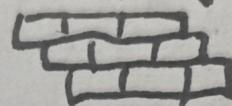


静态发射、调度、冒险检测

3. VLIW



25) 3. Super Pipeline: 每个时钟周期流出n条指令, 每一级流水线用时为



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

1. 算法流程 都無法进入IS, Tomasulo也这样

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

t. 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, V_i, V_k : 操作数的真实值

Q_i, Q_k : 将产生 V_i, V_k 结果的保留站, 若操作数已产生并送入保留站, 则置空 (即: Q_i, V_i 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 V_k 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

FLD F6, 34(R2)

IS EX WB 在Tomasulo

1 3 4 中, Load指令

FLD F2, 45(R3)

2 4 5 的IS和EX之

FMUL F0, F2, F4

3 6-15 16 间要停一拍.

FSUB F8, F6, F2

4 6-7 8

FDIV F10, F0, F6

5 17-56 57

FADD F6, F8, F2

6 9-10 11

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

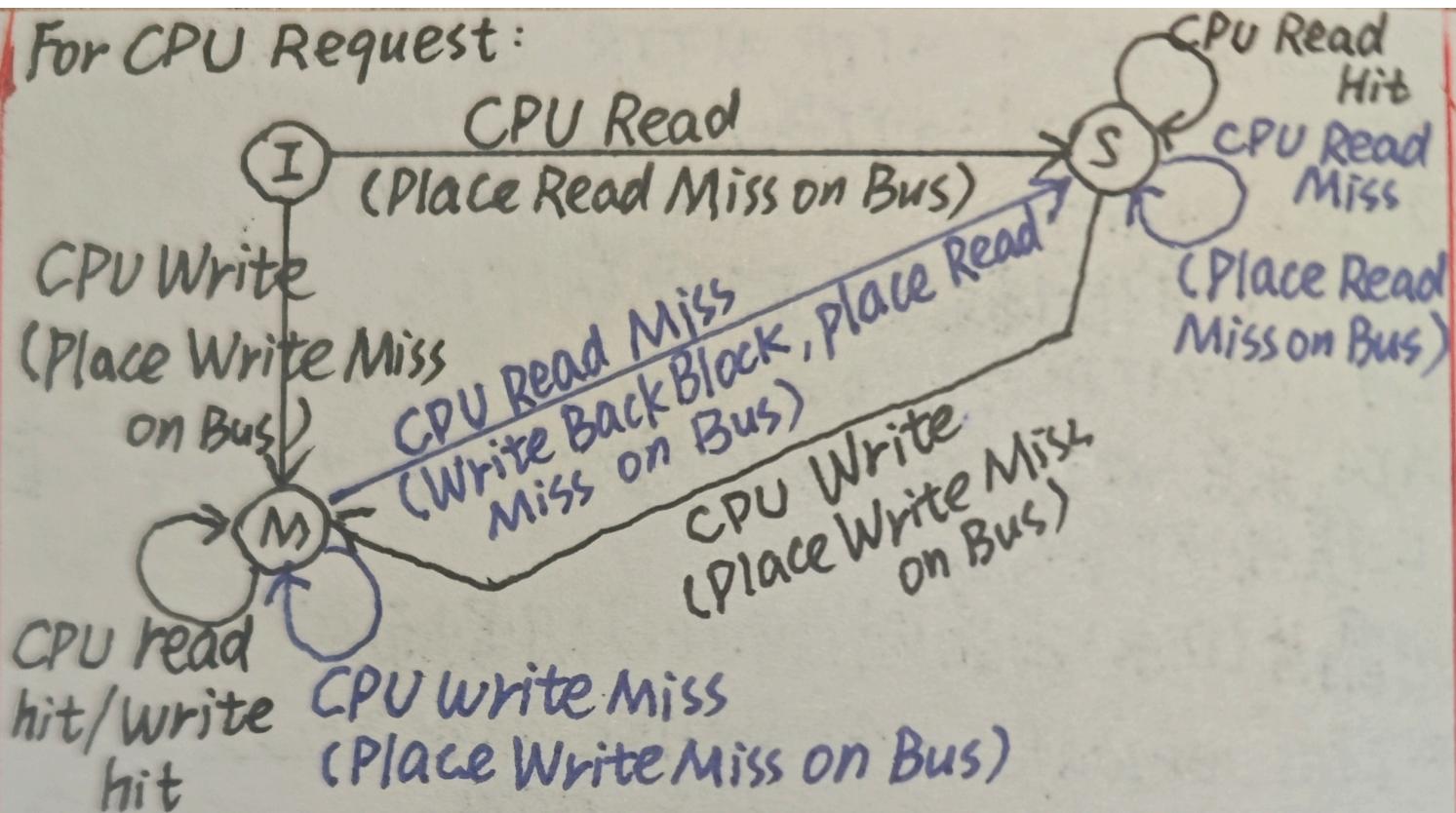
2. 算法流程

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

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 invalidation 信号

3. MOESI

Chapter 4: TLP

一、两类多处理器架构

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
③ 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。

每个核有独立的 Cache

同一内存位置

二、缓存一致性问题 Cache Coherence Problem

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。

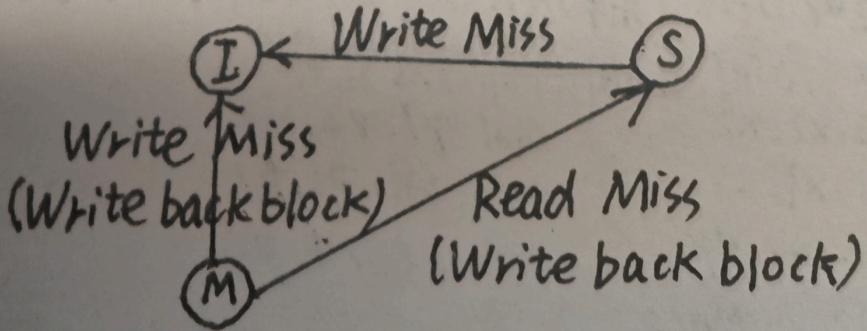
3. UMA 使用 Snoopy Protocol: 所有处理器连总线，写 Cache 时在总线上发送 Invalid 信号；NUMA 使用 Directory Protocol，使用 Directory 记录哪些处理器在 Cache 中有特定 block 的拷贝，修改时，通过 Directory 以点对点方式通知

4. Write-Invalidate / Write-Broadcast

三、Write Invalidate Protocol

1. MSI: Invalid / Shared / Modified

For Bus Request:



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

1. 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 structural conflict.

Chimes: 执行单个 convey 的时间

④ 连接时，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。