CA Formulas

Moore's law: 晶体管数量每两年翻一番
Dennard Scaling: SRAM→cache, DRAM→mem

1. CPU time = inst cnt × CPI × cc time (or $\frac{1}{\text{cc rate}}$)
2. $\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{exec time}_Y}{\text{exec time}_X}$
3. Amdahl's Law: $\text{speedup} = \frac{\text{exectime}_{old}}{\text{exectime}_{new}} = \frac{1}{(1-f_e)+\frac{f_e}{sp_e}}$ (speedup enhanced)
4. $\text{energy}_{dynamic} \propto \frac{1}{2} \cdot \text{capacitive load} \cdot \text{Voltage}^2$
5. $\text{power}_{dynamic} \propto \frac{1}{2} \cdot \text{capacitive load} \cdot \text{voltage}^2 \cdot \text{frequency}$ (switched)
6. $\text{power}_{static} \propto \text{current}_{static} \cdot \text{voltage}$
7. $\text{availability} = \frac{MTTF}{MTTF+MTTR}$ (MTTF → 倒数为故障率; MTTF+MTTR → MTBF)
8. $\text{Die yield} = \text{wafer yield} \cdot \frac{1}{(1+\text{defects/unit area} \cdot \text{die area})^N}$, $N \in [11.5, 15.5]$
9. $AM = \frac{1}{n}\sum_{i=1}^{n} Time_i$; $WAM = \sum_{i=1}^{n} w_i \cdot t_i$; $GM = \sqrt[n]{\prod_{i=1}^{n} t_i}$
10. AMAT = hit time + miss rate · miss penalty
11. misses/inst = miss rate · mem access/inst
12. cache index size: $2^{index} = \frac{\text{cache size}}{\text{block size} \cdot \text{set associativity}}$

Rules of Thumb

(non-blocking cache: very complex, miss penalty↑; for out-of-order CPU, miss ↛ stall)

1. Amdahl/Case Rule: A balanced computer sys need ≈ 1MB main mem capacity and 1Mb/s I/O bandwidth per MIPS of CPU performance
2. 90/10 Locality Rule: program —exec→ inst 90% / code 10%
3. Bandwidth Rule: bandwidth grows by ≥ (improvement in latency)^2
4. 2:1 Cache Rule: (miss rate) dm cache (size N) = 2-way sa cache ($\frac{N}{2}$)
5. Dependability Rule: Design with no single point of failure

Chap 1: Foundamentals of Quantitative Design and Analysis

3 Walls: ILP、mem. power

Flynn's taxonomy: SISD. SIMD. MISD. MIMD (SISD → ILP, SIMD → DLP, MIMD → TLP)

CA — ISA / microarch/org / hw/sys

ISA — stack: push.pop; accumulator: load.add.store; general-purpose register — reg-mem: 都能访存 / ld-st: 只有ld.st

performance — factors: arch. hw. compiler. os
measurement — exec time / throughput

quantative principles: parallelism. locality (temporal / spatial). common cases (Amdahl's law)

Pipelining

classes — static / dynamic (same / diff); component / processor / inte processor (op / inst / macro) (serial); linear / non-linear (feedback loop); order / disorder; scalar / vector

throughput: $tp = \frac{n}{T_k} = \frac{n}{(m+n-1)\Delta t_0}$ (n: # of inst, m: # of stage)

$(n \gg m) = tp_{max} = \frac{1}{\Delta t_0}$

$\text{speedup} = \frac{nm}{m+n-1}$

solution: subdivision. repetition

bottleneck segment: $\max(\Delta t_i)$

efficiency $\eta = \frac{n}{m+n-1}$

Chap 2: Memory Hierarchy Design

higher bandwidth: interleaved mem (multibank) → independent (avoid conflicts)

4 Questions:

① block placement — direct-mapped (unique block): pos = block addr % # of blocks; fully-associative (any); set associative (unique set): pos = block addr % # of sets; n-way: n blocks in one set (usually n≤4)

② block identification

block address		block offset
tag	index	

index: # bits = $\log_2$(# sets); block offset: # bits = $\log_2$(size of block)

valid bit, reference bit, dirty bit

③ block replacement (for set/full associative): random、LRU、FIFO (LRU ← approx)

④ write strategy — write through (write to mem as well) → write around (only mem) (miss); → write buffer (stall↓); write back (write dirty block when replaced) → write allocate (mem → cache) (miss)

unified v.s. split cache ([I] [D])

exec time = (CPU cc + mem stall cycle) · cc time; mem stall cycle = IC · $\frac{\text{mem access}}{\text{inst}}$ · miss rate · miss penalty

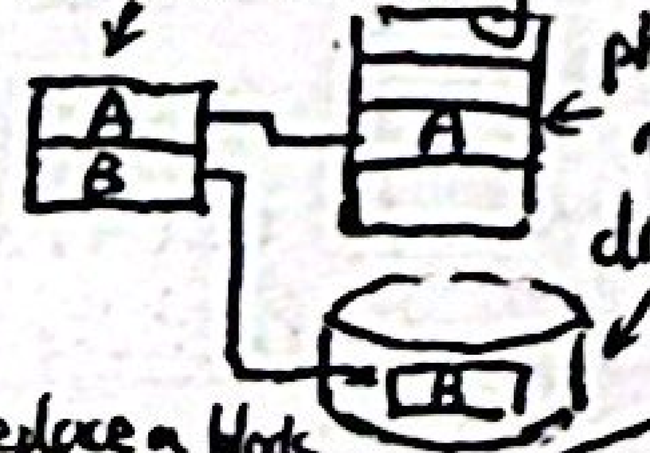
misses → 3C model: compulsory. capacity. conflict (最开始访问 / 替换被换掉的块 / 多块在同组)

(△ widely used)

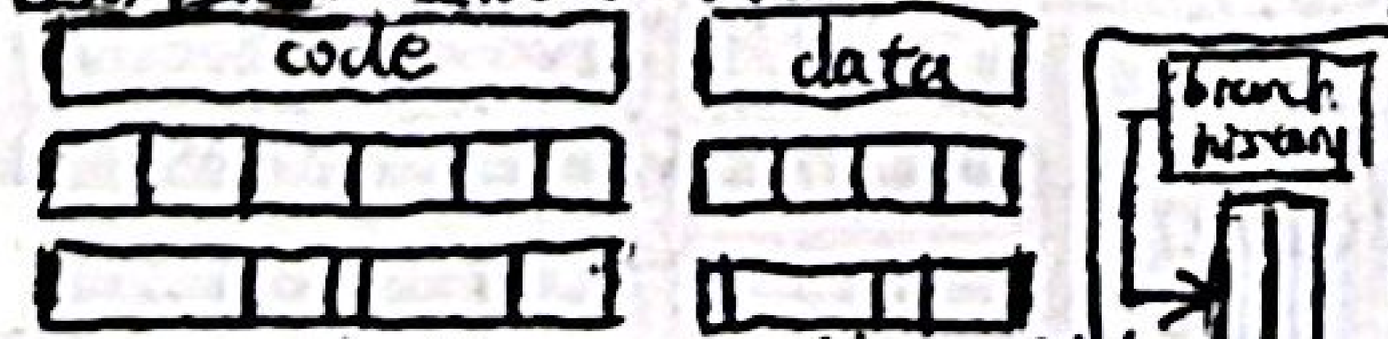
☆ Optimizations

1. hit time↓
   - △ small and simple L1 cache (also power↓)
   - way prediction (预测下次被访问的块，准确率高)
   - △ addr translation ✕ → virtual mem → virtual indexed physically tagged (VIPT)
   - trace cache (VIVT的问题：歧义 + 别名) (same V, diff P / diff V, same P)
2. cache bandwidth↑
   - △ pipelined cache → cc↑, (hit) time↑
   - △ multibanked cache (benefit to larger cache)
   - non blocking cache
3. miss penalty↓ (向mem请求missed data 得到后立马给processor 同时继续执行inst)
   - △ critical word first & early start (一旦获得data mem 就……(同前))
   - merge write buffer (合并相邻addr的write 高效利用mem, stall(↓)) (OK! exist)
   - multilevel cache: L1、L2 (bigger); AMAT = $ht_{L1} + mr_{L1} \times (ht_{L2} + mr_{L2} \cdot mp_{L2})$ (mp_{L1}); miss rate — local: mr_{L1}, mr_{L2}; global: $mr_{L1}, mr_{L1} \cdot mr_{L2}$; multilevel — inclusion → consistency; exclusion → space waste↓
   - △ read > write (for wt. wb): when read miss, check write buffer, get data from mem otherwise
   - victim cache (读起来的block不会直接写到mem而是先在这里)
4. miss rate↓
   - size of block↑: compulsory↓ (L2块略大于L1) but conflict. capacity↑, miss penalty↑
   - △ capacity of cache↑: hit time↑, cost & power↑
   - △ associativity↑: hit time↑
   - compiler optimization — inst: reorder. analyse; data: merge array. loop interchange. loop fusion. blocking (spatial locality↑ / temporal locality)
5. miss penalty↓ / miss rate↓ by parallelism (power↑)
   - prefetch — hw: (以inst cache为例) when miss, 除了请求missed inst, 还会 prefetch next inst, 放在流buffer; compiler — register / cache (nonbinding prefetch), use loops

Virtual memory



作用: 共享受保护的mem space. 自动管理mem层次、简化load、start time↓

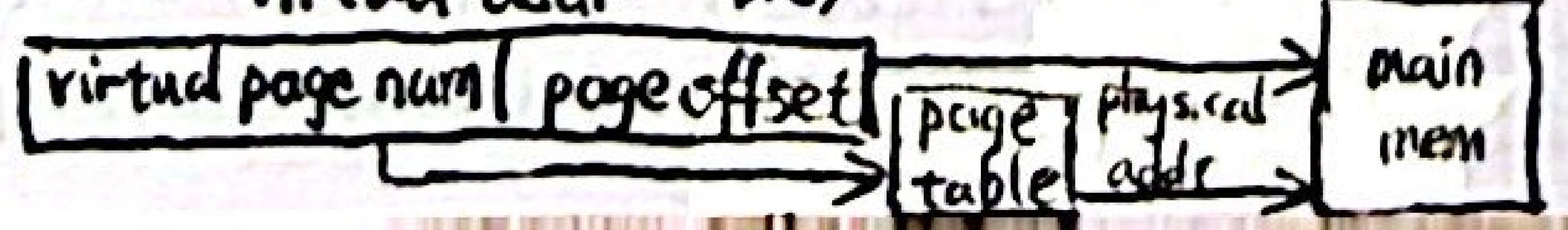


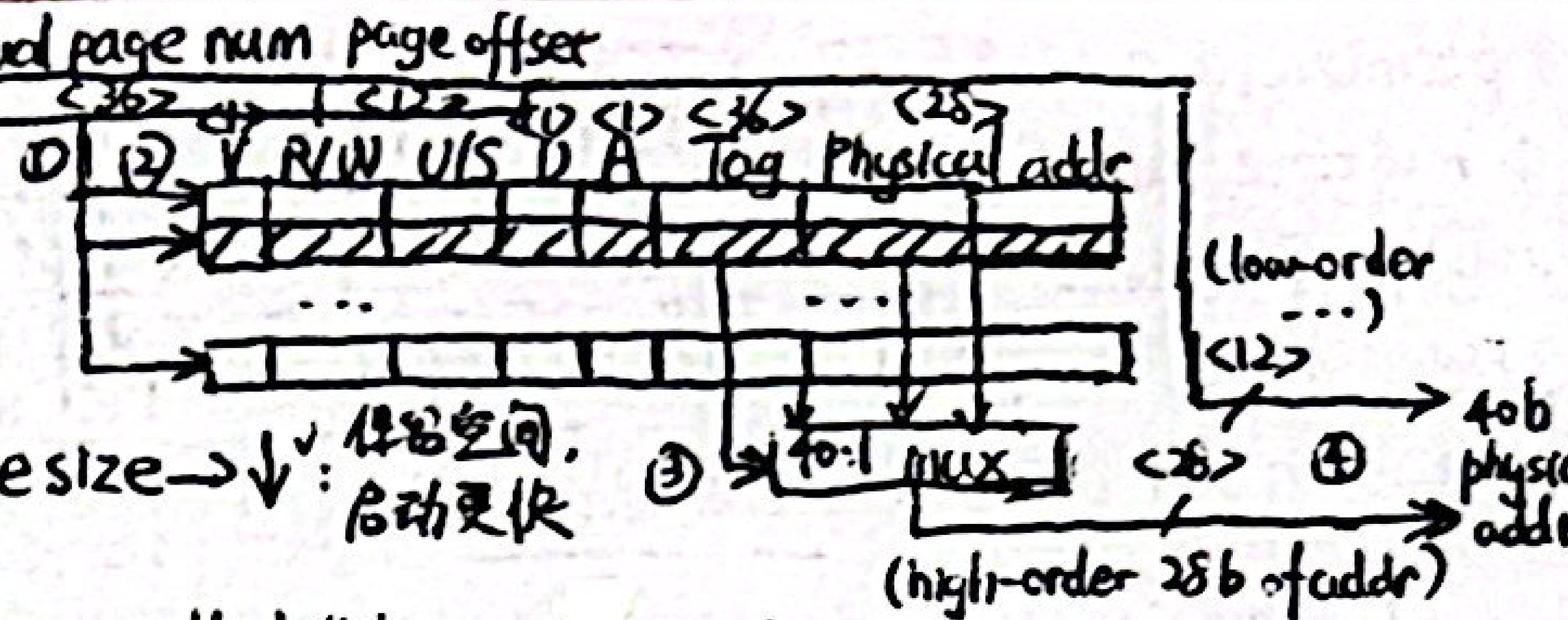
paging: 1 word addr、invisible. trivial. internal fragmentation. efficient

segmentation: 2 word addr、may visible. difficult. external ~. not always ~

(replace a block)

4 questions: fully-associative. page table/TLB; LRU (reference bit); write-back (dirty bit)

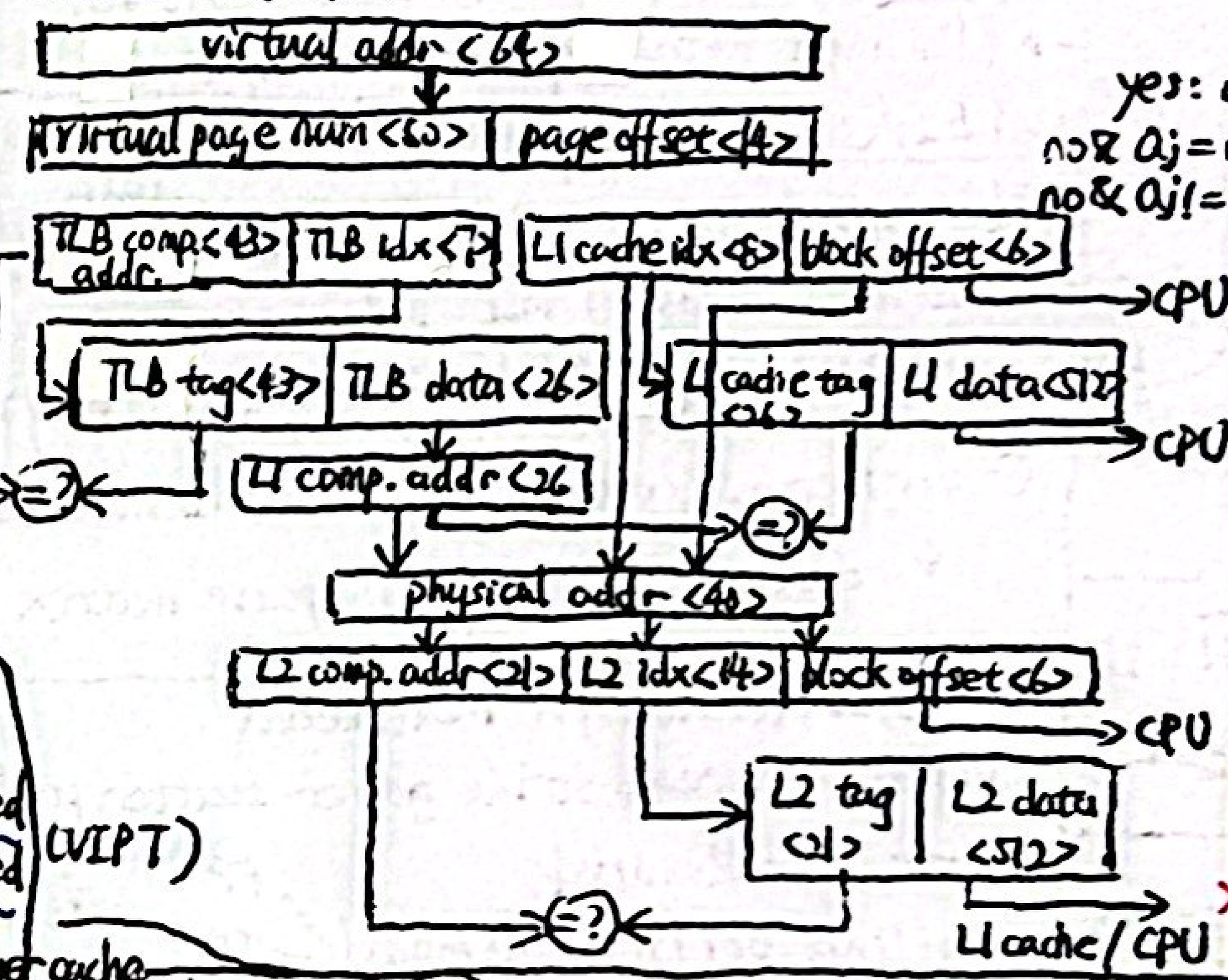




page size → ↓: 保留空间, 启动更快; ↑: page table↓ 节省mem. hit time↓. TLB miss↓

保护: 多程序、进程、时间共享、上下文切换、环级权限 (ring) (user. kernel) (变体)

addr translation



Chap 3: ILP

basic block: 既无分支入, 亦无分支出

dependence (ILP ✕, 不能变顺序) → hazard

- data (true) ~ → RAW ~
- name ~ — anti ~ → WAR ~; output ~ → WAW ~
- control ~ → control ~

(data ~: 调整inst顺序)

relax → 保留 — 异常行为 / 数据流

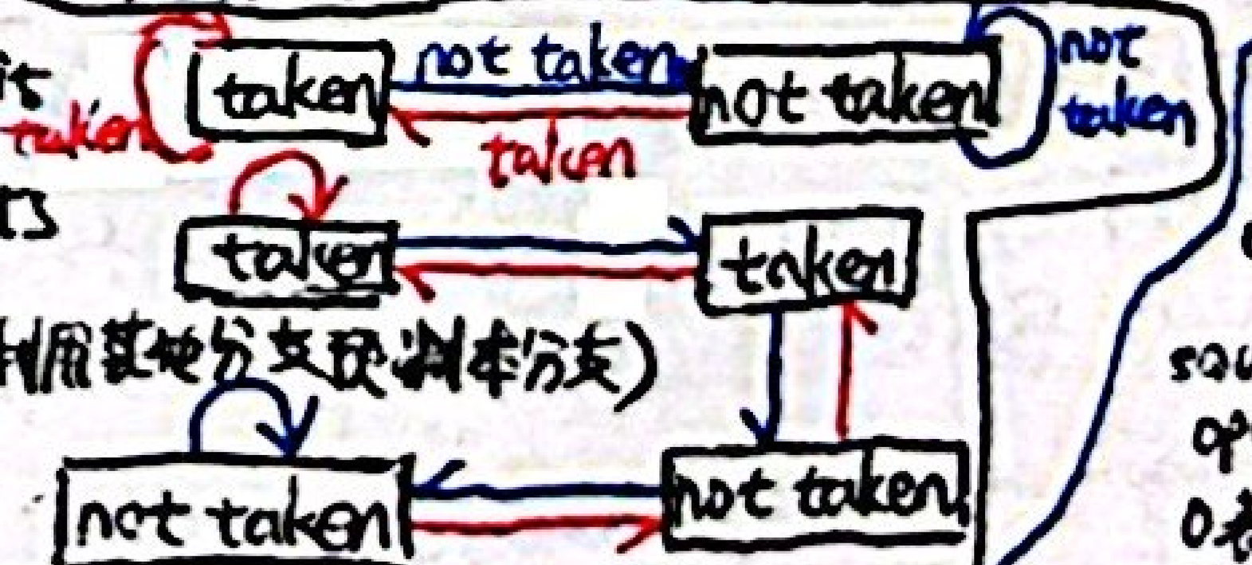
simple compiler technique — scheduling; loop unrolling (→ code scale↑) (↓没有依赖时更有效)

3 tables: ROB. reservation station. FP reg

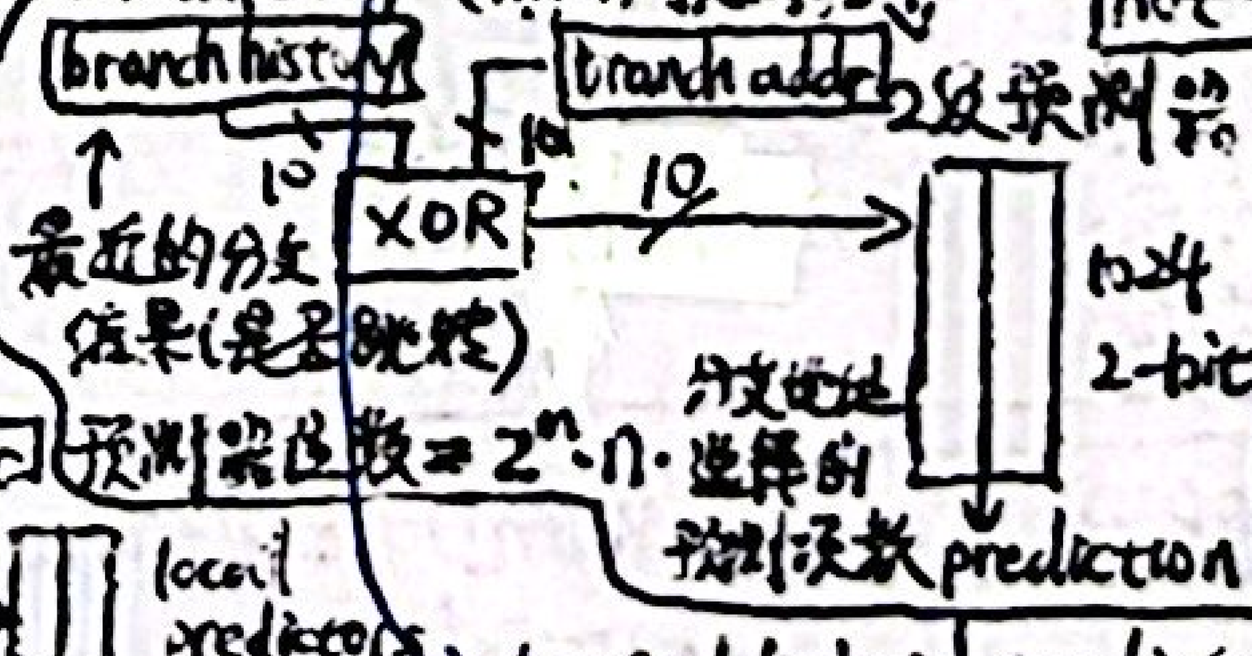
Branch predictions

static: 提前 — 计算分支地址 / 进行分支跳转决策

dynamic → 分支历史表: 1 bit / 2 bits; → 相关预测器 (利用其他分支预测本分支); (m, n) 预测器 (branches, n bit)



2 bit: 1st bit: last NT → ✓; 2nd bit: last T → ✓

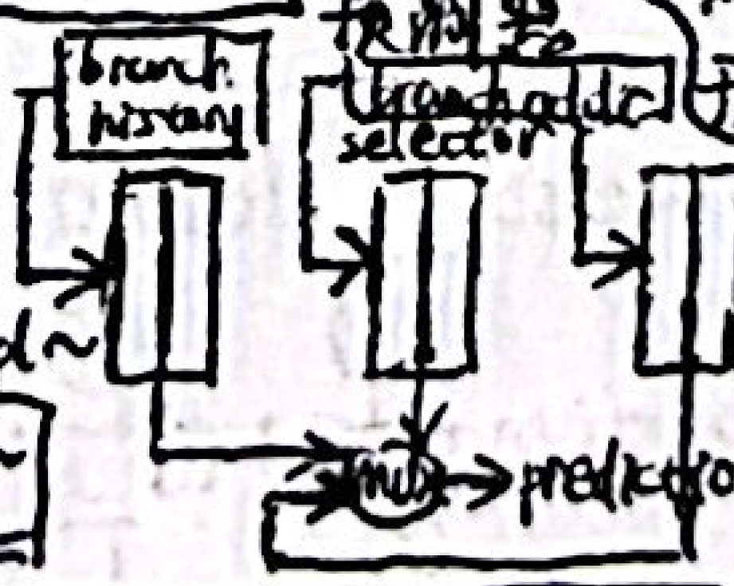


最近的分支结果 (是否跳转)

预测器总数 = 2^n · n. 选择的预测次数

2级预测器

锦标赛预测器



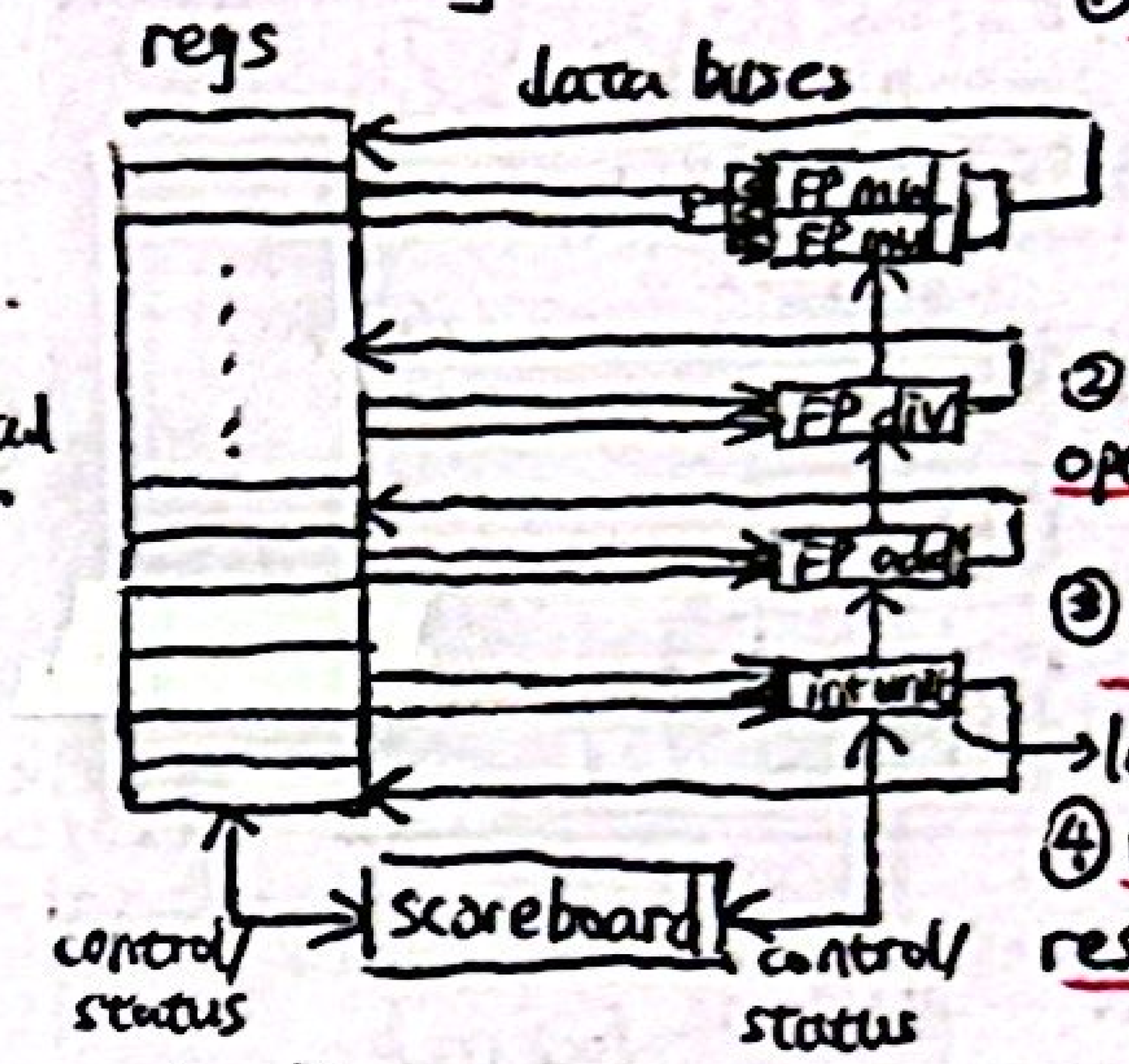
tagged hybrid predictor: 使用一些不同长度历史idx的global ~; use field (2 bits)

Dynamic scheduling

Dynamic scheduling: 顺序发射、乱序执行 (程序顺序) 问题 — WAR. WAW↑ → reg rename; imprecise异常 → ROB (无法确保异常点前所有指令完成执行, ~之后已部分执行的状态)

ID stage — issue (IS): 译码, 检查结构冒险; read operand (RO): 确保没有数据冒险后读operand

Scoreboarding



(无结构~和WAW~)

① issue: function unit和regs空闲 → 发射指令到fu上 (IF和IS间有buffer)

② read operand: operand可用 (无RAW~) → fu读operand from reg

③ exec: fu执行 → 算好后告诉记分板

④ write result: 记分板检查WAR冒险, 若有则stall, 没有则将结果写到reg

缺点.

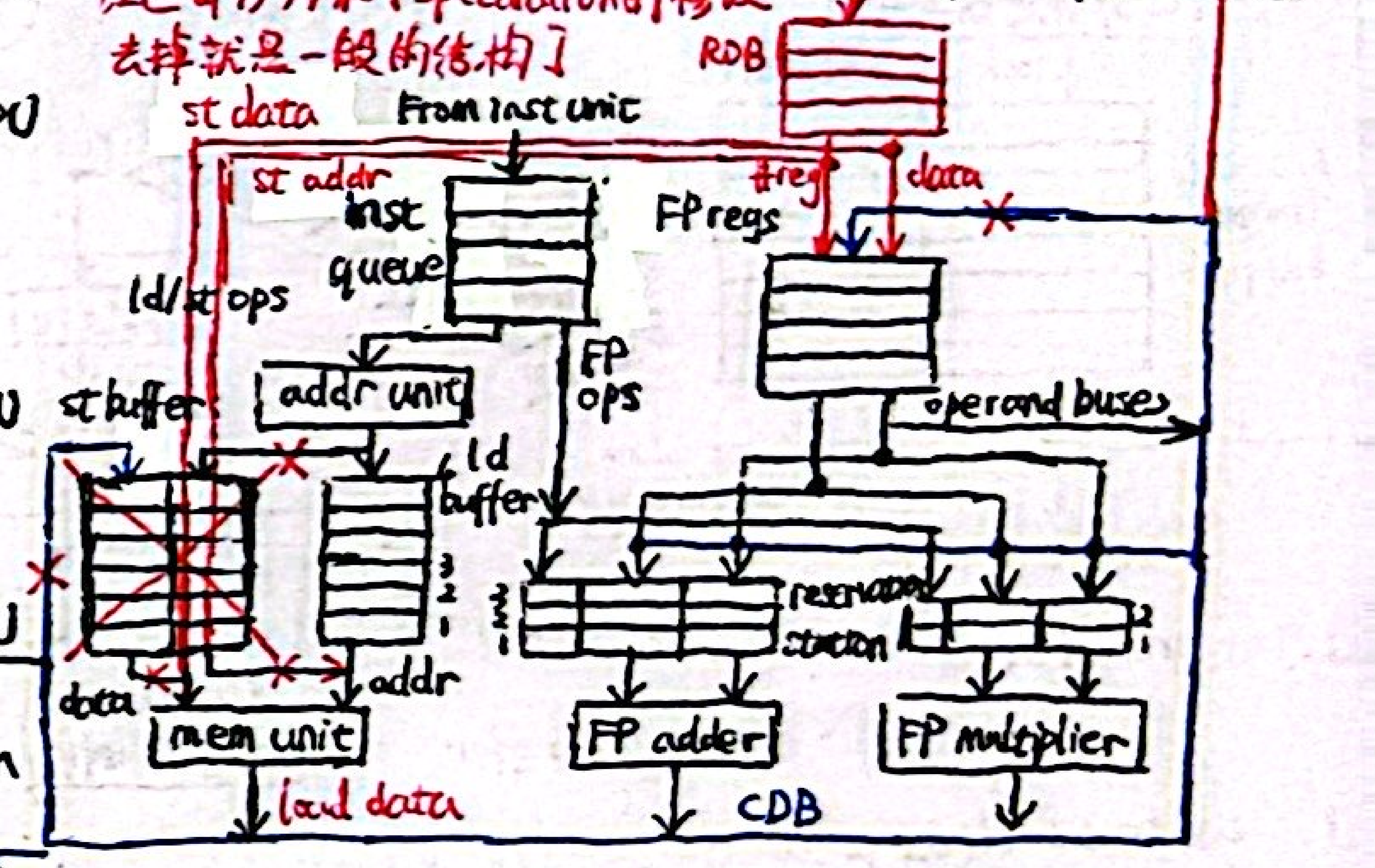
3 tables: inst status ~. fu status ~. reg status ~

fu status: busy. op. Fi.Fj.Fk (operand对应的reg). Qj.Qk (source of operand (fu)). Rj. Rk (operand是否ready)

yes: operand ready but not read; no & Qj = null: operand is read; no & Qj != null: operand is not ready

Tomasulo's Algorithm

红色部分为加了speculation的修改, 去掉就是一般的结构了



reservation station: buffer, reg renaming

CDB: 公共数据总线

① IS: 从queue获取inst, 若有空的保留站, 将inst和operand (from reg) 发射到保留站内, 实现reg renaming (消除WAR和WAW~) (没有表示结构~, IS停止)

② EX: 若存在operand不可用, 监控CDB, 等到可用后放到FU执行 (消除RAW); load/store额外考虑

③ WR: 将结果写到CDB (广播)

3 tables: 将记分板的fu → reservation station ~ field

op. Qj.Qk (source of operand (rs), 0表示已经有val了). Vj.Vk (val of operand). A (mem addr). busy (保留站&FU是否被占)

缺点: 实现复杂、受限于CDB、ld/st相同addr、不精确的中断 (↳1个cc内只能出1个结果)

dynamic scheduling + speculation: 乱序执行 → 顺序提交

(为减少control依赖的影响)

ROB: 重排缓冲区 (合并了st buffer, 并负责reg rename) (FIFO)

(fields) inst type (branch/st/reg). dest (reg/mem). val. ready

① IS: 保留站和ROB都可用再发射, 更新这2者
② EX: 同上
③ WR: CDB与ROB而不是reg连接 (其他同上)
④ commit — 正常 (包括st): ROB → reg/mem (st); 错误预测的分支被清除ROB项, 重开下一条inst (实现精确中断 (异常指令不会提交))

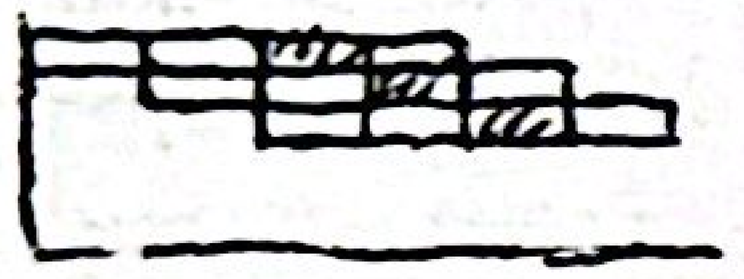
multiple issue

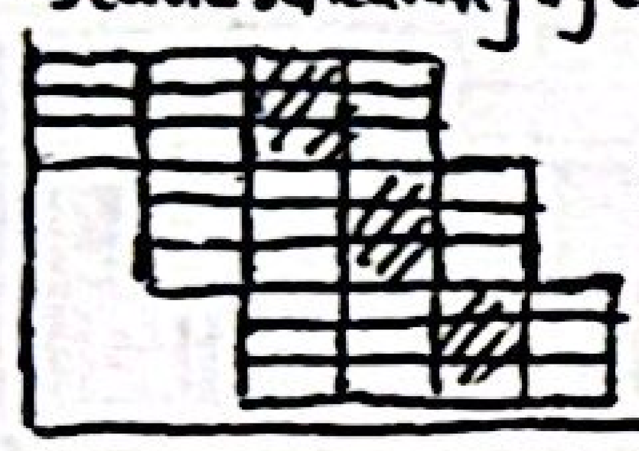
superscalar
- 每cc内发射inst数可变
- upper limit n → n-issue
- static → compiler
- dynamic → Tomasulo

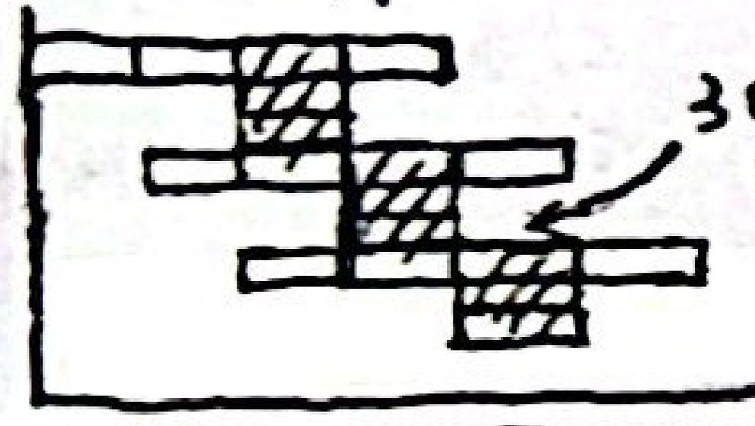
IS是瓶颈

VLIW (超长指令字)
- 每cc内发射固定量inst (inst packet)
- static scheduling by compiler

op slot



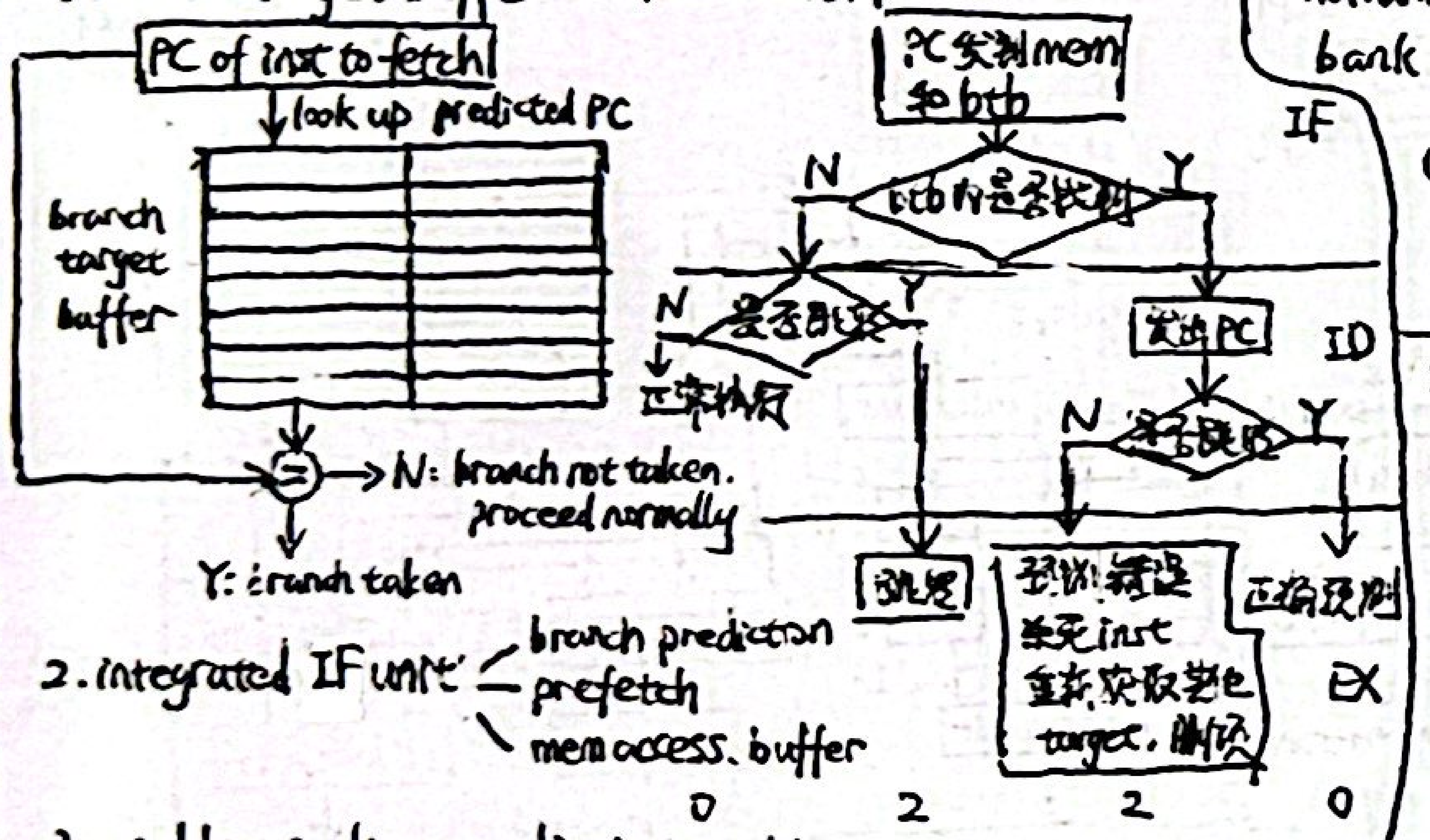




VLIW → problem
- code scale ↑ — solution → loop unroll / smarter encode / compress
- lock step
- logic compatibility ×

advanced techniques for IF and speculation

1. branch-target buffer → IF bandwidth ↑



2. integrated IF unit
- branch prediction
- prefetch
- mem access buffer

3. address alias prediction: ld/st 是否访问相同addr

multithreading in uniprocessor

- fine-grained ↓ switch in threads, round-robin (轮询)
- coarse-grained ↓ switch only for high-cost stall
- simultaneous ☆ (SMT) ↓ reg renaming + dynamic scheduling

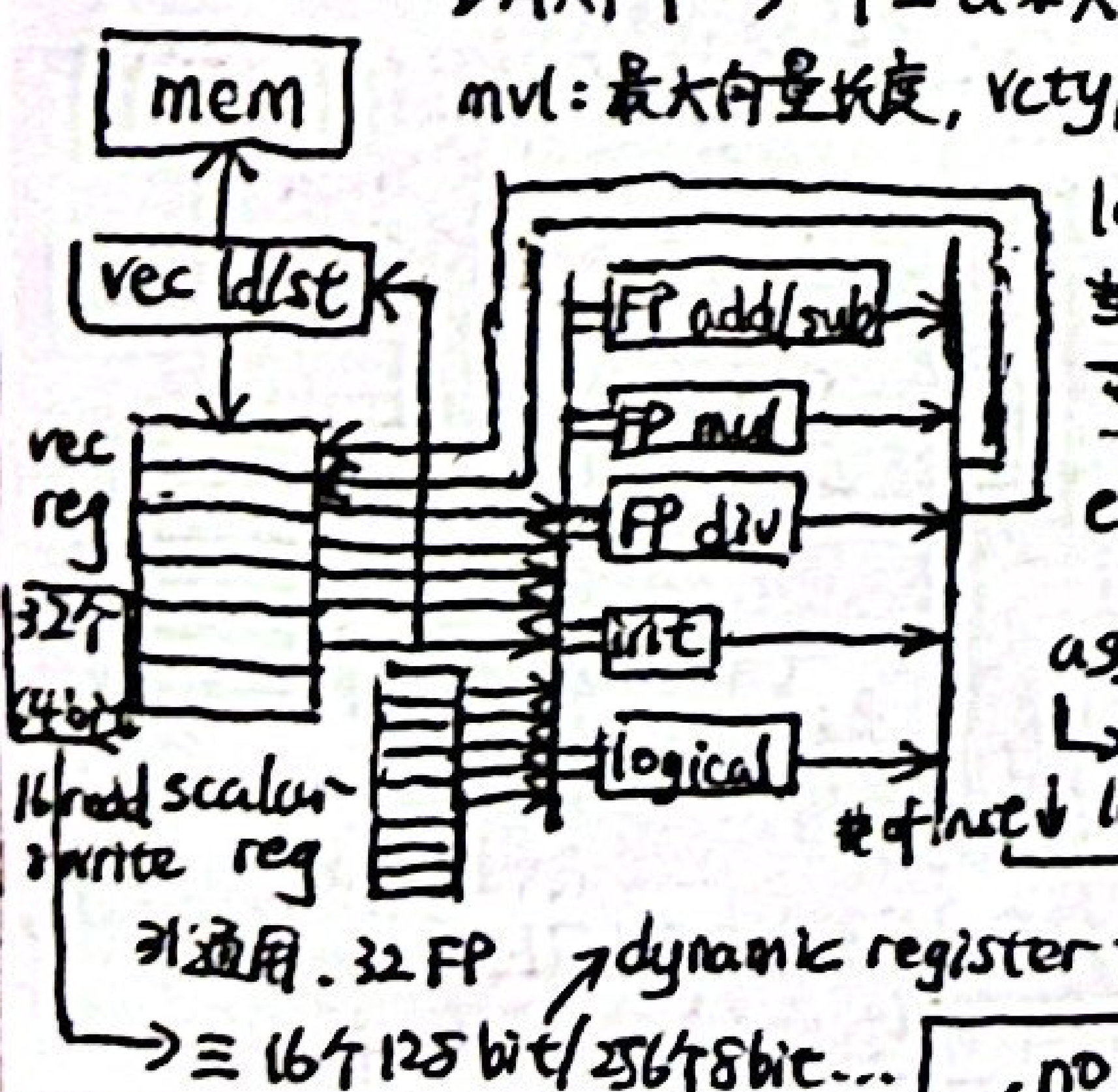
Chap 4: DLP → SIMD

vector arch
- mem-mem
- vec-reg ✓

RV64V

DAXPY → Y = a * X + Y (X, Y: vec, a: scalar)

mvl: 最大向量长度, vctype: 寄存器类型, pi: 谓词寄存器



引通用, 32 FP

→ 三 16个 128 bit / 32个 64bit ...

dynamic register typing

convoy: 一组可同时执行的 vec insts → chaining — allow RAW 结果一旦出来就可用 无需等到了reg再写

no structure hazard

chime: exec time for 1 convoy → ignore start-up time

correlation problem

frequent function switch
↓ solution
horizontal / vertical / group

loop-carried dependence: 当某代数据依赖前代时, 可向量化循环代码 (语句间的依赖 × / 语句内的依赖 ✓ → recurrence) (不同迭代)

exec time
- vec len
- structure
- data

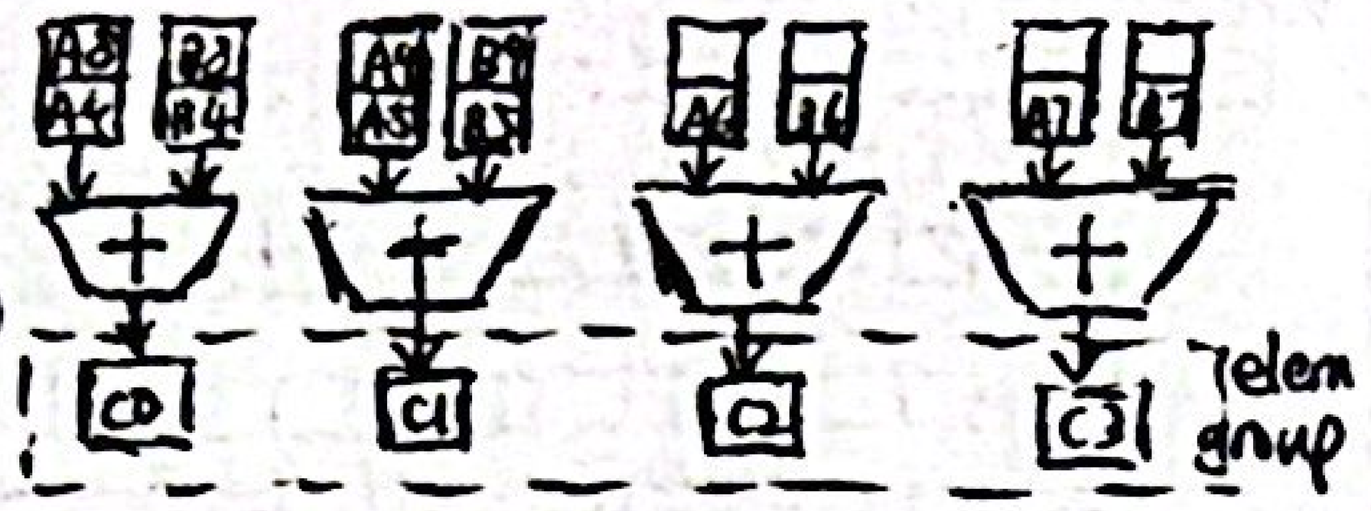
assume 1 lane
↳ initiation rate: 1 elem/cc

# of inst ↓ long vec implicit conversion — benefit — exec time ↓ — for 1 inst = vec len

optimization

① multiple lanes

throughput ↑ (4 lanes) by parallelism



② vec-len reg (当len > mvl 或len未知) inst: setvl
↓ vl strip mining vl = min(mvl, n) : n -= vl loop

③ predicate reg (p0, boolean) inst: vsetpcfgi, vpdisable
vector-mask control / IF-conversion
↳ eliminate control dependence

④ mem banks → throughput ↑ → maintain initiation rate

⑤ stride: 向量内相邻2个元素的物理距离 (byte)
nonunit stride (stride ≠ 1) inst: vlds, vsts
bank conflict: $\frac{\# \text{ of banks}}{LCM(\text{stride}, \# \text{ of banks})}$ < bank busy time

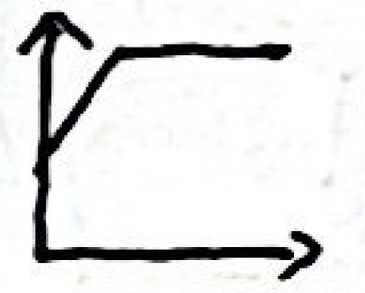
⑥ gather-scatter inst: vldi/vldx, vsti/vstx
↓ index vector A[K[i]] sparse matrix

SIMD Extension for Multimedia

no vec-len reg, stride, gather-scatter, pred reg

cost ↓ bandwidth ↓

roofline performance model
- FP / mem ↑ FLOP / B mem
- arithmetic intensity



attainable GFLOPs/s = min (peak mem BW × arithmetic in~, 平衡 peak fp performance)

GPU

CUDA: compute unified device arch

CUDA thread → SIMT, multithread SIMD processor

Thread Block (body of vec loop)

Grid (vectorized loop) (sequence of SIMD lane op)

local mem (private mem)

global ~ (GPU ~)

loop-level parallelism

(1) find dependence
- m ≤ j, k ≤ n, a·j + b = c·k + d — affined index
- compile time ×
- NPC problem

GCD test: GCD(c, a) % (d - b) = 0

(2) eliminate dependence
- scalar expansion
- reduction

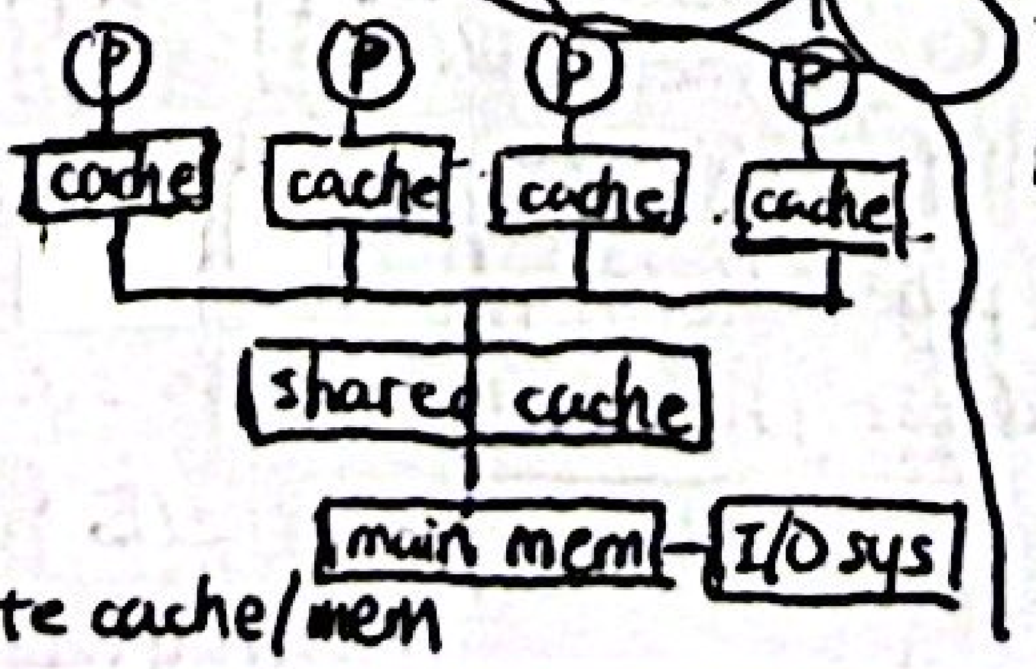
Chap 5 TLP

MIMD multiprocessor (multicore)

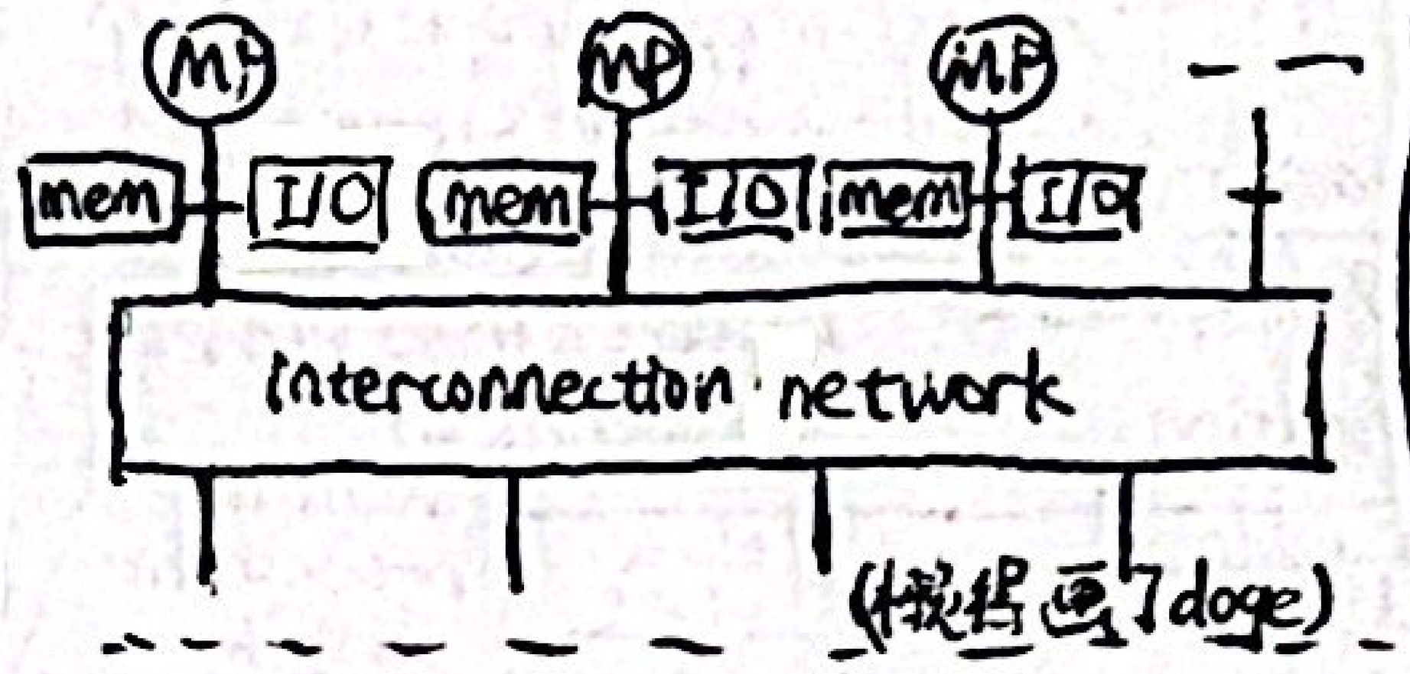
software model
- parallel processing
- request-level parallelism

memory orgnization

① SMP (centralized shared-mem arch)
↓ UMA < 访问时间相等, 每个processor有1个private cache / mem



② DSM: local mem: fast, remote mem: slow
↓ NUMA



SMP, DSM 都有 shared mem → addr space

challenge
- 独立task → limited parallelism
- 处理间协作 → high communication cost
- 完成的并行任务

cache coherence (满足以下条件)

① 处理器P写入X后立马读X, 中间无其X写操作
② P2在P1写入X后读X, 中间无对X写操作
③ write serialization: 2次write结果符合顺序

规定同一mem addr的R/W

shared data → local ~ < migration / replication

protocol
- directory based: 目录存有block的shared status
- ☆ snooping: cache可trace block的shared status, using bus (broadcast media)

☆ write invalidate: 处理器写入数据时对其有exclusive access, 其他processor读该块(的copy)会miss, 需fetch updated one 从mem

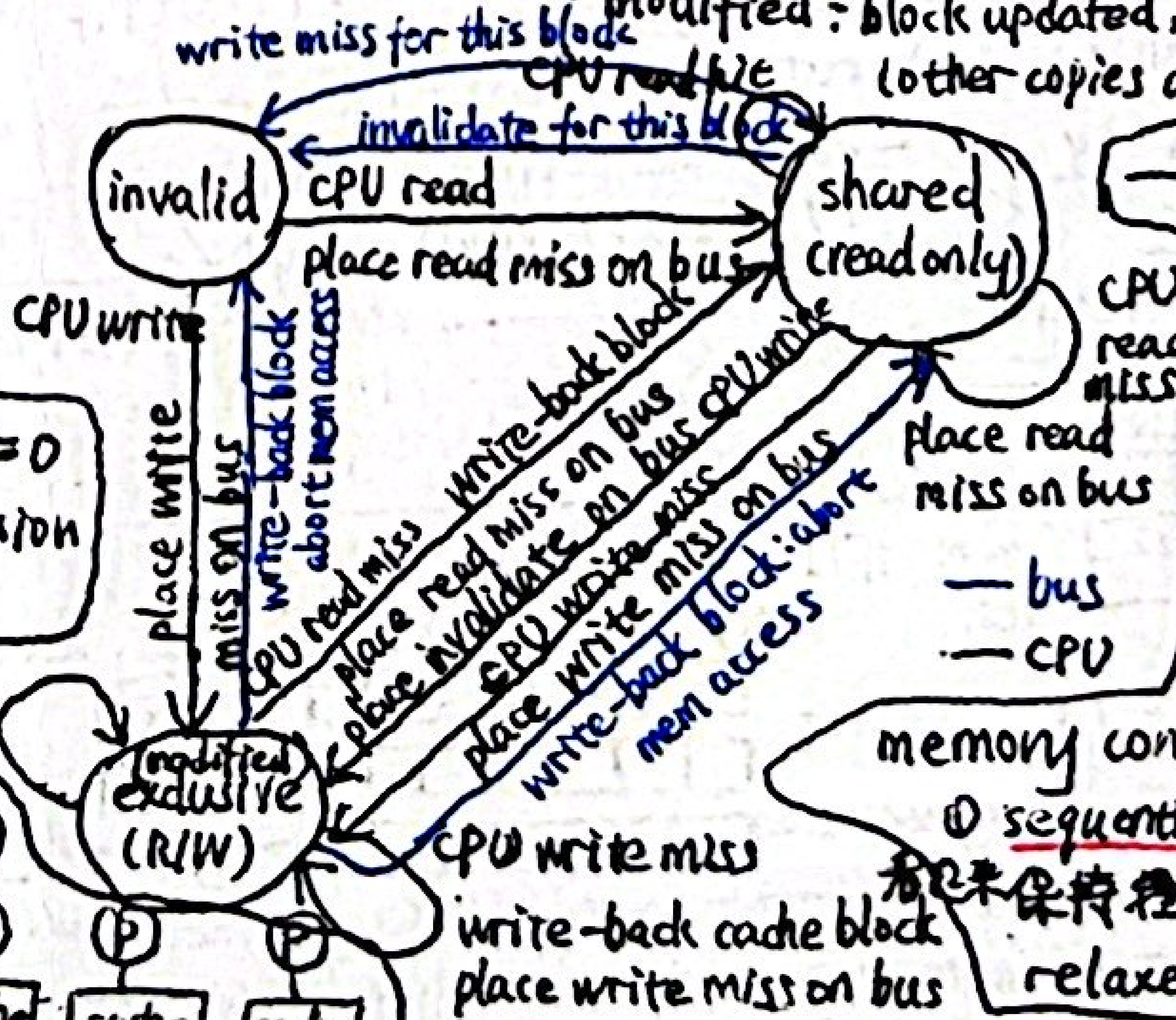
write update / broadcast: 要同时更新其他processor对应的copy → using more bandwidth

☆ write back or write through (easy, bandwidth ↑)

在replace时 bandwidth ↓, 才将dirty copy写到mem

owner: 拥有cache block唯一copy的core

MSI protocol
- invalid
- shared: block 被多核共享
- modified: block updated, exclusive (other copies are invalid)



MESI: 加了E (exclusive): 仅在某一cache且unmodified, 好处是对exclusive block写入时无需write miss, 只要改成modified状态

MOESI: 加了O (owned): cache持有在mem中过期的block

modified → owned, 无需写入内存

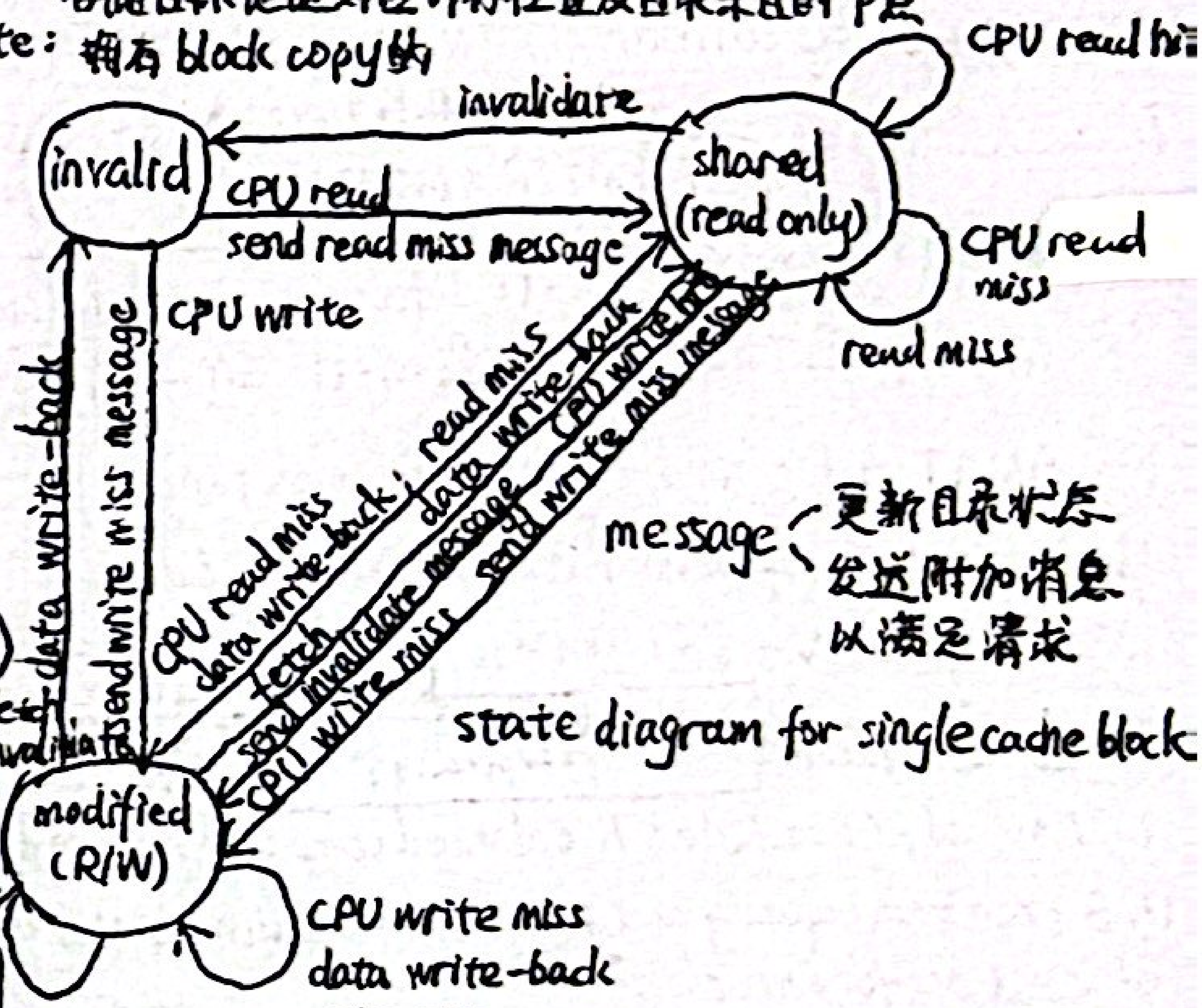
DSM → directory protocol

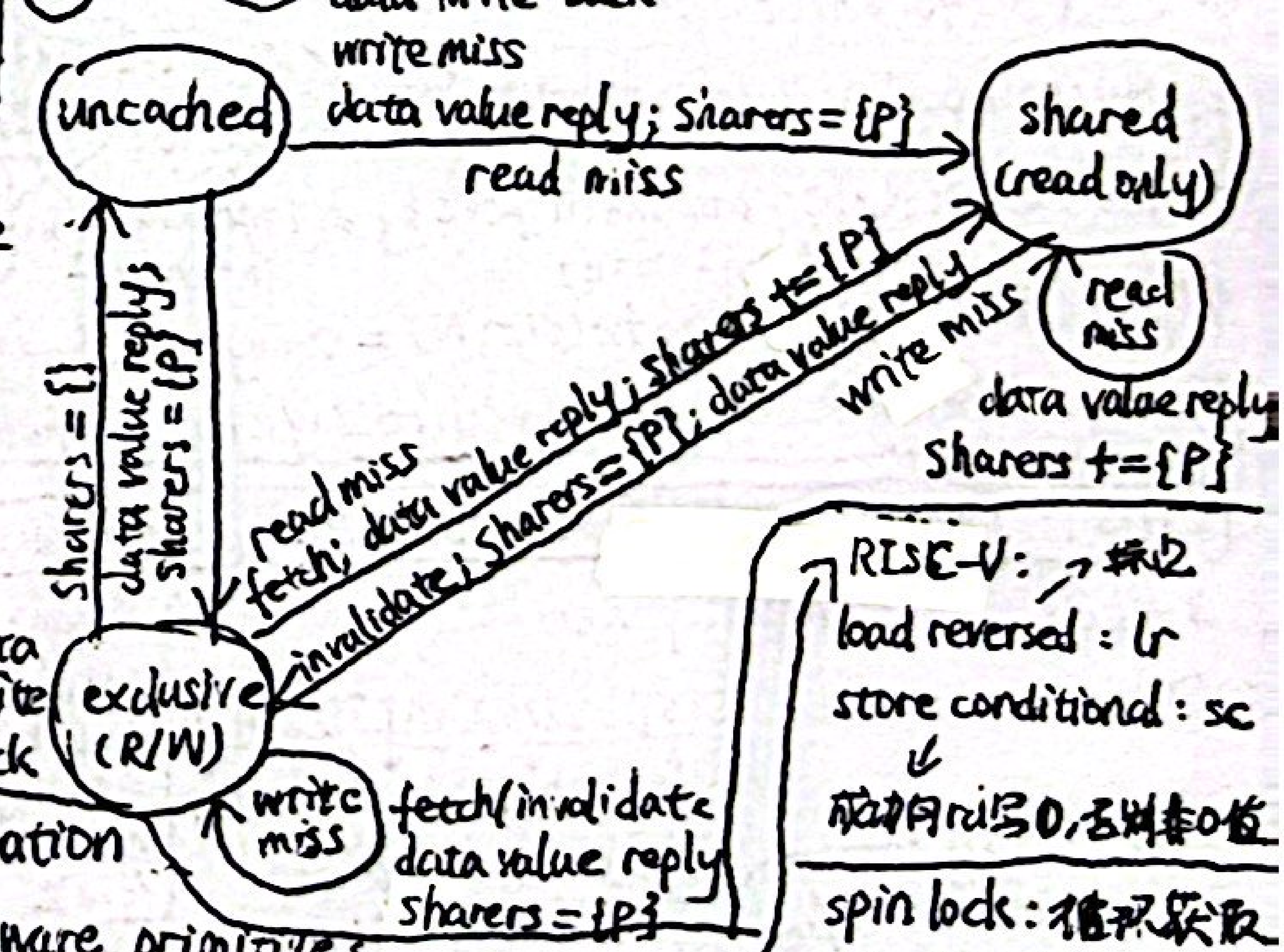
(directory) state
- shared
- uncached: 没有任何node有block的copy
- modified: 只有该node有block的copy (owner)

sharers 用(bit vector)来记录哪个node有block的copy

node
- local: 发起 request
- home: 存储目标地址对应内存位置及目录条目的节点
- remote: 拥有block copy的

message
- 更新目录状态
- 发送附加消息以满足请求





synchronization

basic hardware primitives

① atomic exchange: 交换reg和mem的值
lock: 0表示free, 1表示 occupied

② test-and-set ③ fetch-and-increment
通过检测置新值 返回mem val, val++

RISC-V: 挑战
- load reversed: lr
- store conditional: sc
↓
成功时rd写0, 否则非0值

spin lock: 循环获取锁, 直至该锁可用

```
lockit: addi x2, R0, #1
        EXCH x2, 0(x1)
        bnez x2, lockit
```

caching lock

```
lockit: lr   x2, 0(x1)
        bnez x2, lockit
        addi x2, R0, #1
        sc   x2, 0(x1)
        bnez x2, lockit
```

memory consistency (涉及不同mem addr的R/W)

① sequential consistency: 每个processor的内存访问都要保持程序顺序 → 延迟, performance ↓

relaxed consistency model: (R → W, R → R, W → R, W → W)

② processor consistency: (total store) W → R

③ partial store order: W → R, W → W — out-of-order

④ weak ordering, release consistency: 都放松 — acquire S_A, release S_R