Computer Architecture

Chapter 1. Fundamentals of Computer Design

Performance: ① Latency (Response time) event从开始到结束
② Throughput (bandwidth) 给定周期内total work
※ performance ≡ 1 / Execution Time
CPU Execution Time = CPU Clock Cycles × Clock Period
= CPU Clock Cycles / Clock Rate (frequency)
CPI = CPU Clock Cycles / Inst Count (cycles per inst)
CPU Clock Cycles = Inst Count × CPI
CPU Time = Inst Count × CPI × Clock Period

Amdahl's Law: $T_{improved} = \frac{T_{affected}}{\text{improve factor}} + T_{unaffected}$

Speedup = $\frac{\text{Execution Time}_{无改进}}{\text{Execution Time}_{有改进}}$, $T_{new} = T_{old} \cdot (1-f+\frac{f}{S_p})$
整体加速比 = $\frac{1}{1-f+f/S_p}$　$S_p \to$ 无穷时 $speedup_{overall} < \frac{1}{1-f}$　f: 改进部分占比　S_p: 改进部分的加速比

Great Architecture ideas:
① Moore's Law (18~24月) ② Use abstraction to simplify design
③ Make the common case fast ④ Use a hierarchy of memories
⑤⑥ Improve performance via parallelism / pipelining / prediction
⑦ Improve dependability via redundancy

ISA: ① memory addressing ② addressing modes
③ types, sizes of operands ④ operations ⑤ control flow inst
⑥ encoding an ISA (opcode)

ISA Classes: ① stack arch ② Accumulator ③ GPR (General-purpose reg) arch → {reg-mem / load-store}

Chapter 2: Pipelining

single function / multi function: 流水线中不同段组合可以不同功能
static: 同一时刻流水线只做同一个功能, e.g. FP Add
dynamic: 多 ...
以不同粒度分类: ① component level~: 将不同类型操作分段 ② processor level ~ (inst ~) ③ Inter processor ~ (macro~) 一个处理器完成一部分data.
还可以分为 ① Linear ~: 串行, no loop ② nonlinear: 有feedback loop. ① ordered / disordered ② scalar / vector

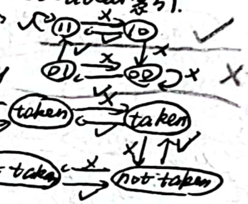
Throughput (TP) = $\frac{n}{n+m-1} TP_{max}$　n: inst数　m: pipeline段数
→ TP_{max} ($n \to \infty$)
最长的一段: bottleneck segment. sol: ① subdivision ② repetition
$TP_{max} = \frac{1}{\max \Delta t}$　speedup = $\frac{n}{n+m-1} \to m$ ($n \gg m$)
EX 部件利用率 efficiency $\eta = \frac{n}{n+m-1}$, 百分比

Hazards:
① Structure ~: 对Resource争用. sol: bubble / 加hw
② Data ~: (i) RAW: 除load-use外, 可由forwarding解决
(ii) WAR (iii) WAW → output-dependence
anti-dependence ↳ name dependence, 乱序中可能有

Forwarding conditions:
EX hazard: if (EX/MEM.RegWrite and E/M.Rd ≠ 0 and E/M.Rd = ID/EX.Rs) ForwardA = 10
------ ID/EX.Rt ForwardB = 10
MEM hazard: if (MEM/WB.RegWrite and M/W.Rd ≠ 0 and M/W.Rd = I/E.Rs and not (EX...)) ForwardA = 01
------ I/E.Rt ------ ForwardB = 01
Load-use hazard: if I/E.MemR and ((I/E.Rt = IF/ID.Rs) or (I/E.Rt = IF/ID.Rt))
stall: ID/EX → 0, EX.MEM.WB do nop, PC, IF/ID 不更新
ld 后跟 Rt Rs
别名] ld R5 bypass → sd R5

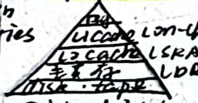
Control Hazard

static branch prediction: predict taken, not-taken, delayed branch (延时槽)
dynamic branch prediction:
BHT (history table). 由branch指令的addr索引.
1-bit predictor; 2-bit predictor



Branch-Target Buffer (BTB): cache of target addr, indexed by PC if
(m, n): 根据最近m次分支结果, 每种2^m组合对应一个n bit predictor
$2^m \times n$: 一个branch需要

Chapter 3 Memory Hierarchy



bigger ↓　↑ faster
everything is a cache. (上层是下层的缓存)
e.g. regs, L1-cache, L2-cache, memory, TLB, BTB

① block放在上层的哪里 (Block placement): direct-mapped; fully-associative; set-associative n-way表明一个组有n块
② block 找? (Block Identification): tag
index 用来 select set

block Addr		offset
tag	index	

③ miss了替换谁? (Block replacement): Random; LRU; FIFO
LRU是stack replacement alg, 即N小命中率小 (FIFO不是)
④ write strategy: write hit时 (i) write-through 也写回到内存 (ii) write-back 只写cache (多一个dirty bit)
write miss时 (i) write allocate 把块先读到cache中 (ii) write around: 直接写回内存
可以在mem前加上write buffer, 可以减少stall
allocate + back
around + through

AMAT = Hit Time + MissRate × MissPenalty
CPU Time = IC × ($\frac{ALU ops}{Inst} \cdot CPI_{ALU} + \frac{MemAccess}{Inst} \times AMAT$) × Cycle Time
= IC × ($CPI_{execution} + \frac{Miss}{Inst} \times Penalty$) × Cycle Time
(分为 inst 和 data 两部分)

优化cache: ① ↓ miss penalty: multilevel cache; (a) critical word first; (b) read miss before write miss; (c) merging write buffers; (d) victim caches
② ↓ miss rate: larger block size; larger cache size; higher associativity; (e) way prediction and (f) pseudo associativity; (g) compiler optimization
③ ↓ hit time: small & simple cache; avoid addr translation; (h) pipelined cache access; (i) trace caches
④ 并行, ↓ miss rate & penalty: (j) non-blocking caches; (k) hw prefetching (l) compiler prefetching
(a) 优先拿需要的字, 这样处理器可以提前工作, + early restart
(b) 给read更高优先级 (c)
(d) cache of cache, 存被踢出的block, ↓ miss rate, penalty
(e) 预测访问组中哪一行有需要数据. 如果√, 类似于direct map
(f) 逻辑上分为两个区 (一个组内), 先按直接映射在上半区找.
(h) (i) 下一个store在比较tag时, 上个指令就可以写入buffer.
(i) 缓存logic上的指令流, 而不是物理上
(j) miss后依然可执行后续指令
(k) 预取, 多拿几个block
(l) compiler 加入预取指令

Virtual Memory = main memory + secondary storage

全相联 (↓ miss rate); 用VPN找PPN, LRU; write-back
pg table存在内存里 ⇒ 多了一次memory访问 1 get pa 2 get data in pa
⇒ TLB (Translation Lookaside Buffer). 有tag和data
VPN PPN

Chapter 4 ILP (Instruction-Level Parallelism)

idea: dynamic scheduling. method: out-of-order execu
Scoreboard algorithm: IS → RO → EX → WB
IS阶段会检查structural hazards (in-order)
RO阶段会等到没有data hazards时去读operands (out-of-order)
3 tables: Instruction Status; Function Component Status; Register Status:
功能部件表中: Fi Fj Fk表示Rd, Rs1, Rs2. Qj, Qk表示源操作数来自哪个部件; Rj, Rk表示源操作数状态
Rj, Rk = yes (operand is ready but no read), no & Qj = null has been read
no & Qj ≠ null not ready (有指令要修改reg)
有的指令只有Fj Qj 无Fk Qk
寄存器状态表中, Qi (F1 F2 ...) 表示哪个指令 (name) 会写回到这个reg.
Scoreboard算法只检测, 没有解决冲突.

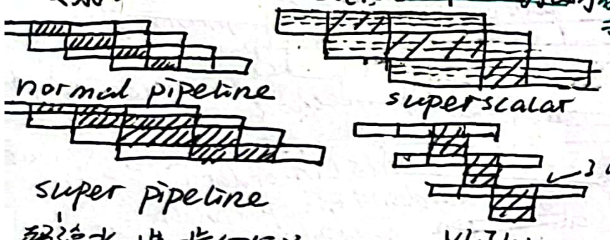
Tomasulo's algorithm: Issue → EX → WB
通过保留站实现register renaming, 解决WAW, WAR
① Issue: 从Instruction Queue中流出. 若对应功能保留站有空闲, 发射. 若op就绪后, 否则跟踪对应的功能单元.
② Execute: op就绪时, 在功能部件执行对应运算 (load / store 两步: 先计算addr, 再放入load/store buffer)
③ Write Result: 结果计算好后, 通过CDB (Common Data Bus) 写到寄存器中, 并广播到其他保留站.
ADD/SUB
指令状态表同前, 保留站 (功能部件) 状态表不同: Op保留站的操作; Qj Qk表示对应寄存器的值由哪个保留站得出; Vj Vk表示已经得出的寄存器值; A表示内存地址 (load/store); Busy表示当前此保留站是否被使用. 寄存器状态表记录保留站结果往哪里写.

Hardware-Based Speculation

ROB (Reorder Buffer). 加上一个commit状态要求按顺序提交. ROB记录指令类型, 目标地址和值. 提交时才能写寄存器和内存;
保留站状态表 Qj, Vk写ROB号 多了Dest记录写回的寄存器; ROB状态表记录 指令发射就要记录
指令编号; busy (提交就no); 指令内容; 当前处于哪个阶段; 要写回的寄存器 object, 以及对应的值 (WB了才有, 之前为null)
寄存器状态表记录 F0寄存器对应ROB编号 (指令编号) 和 busy状态

uperscalar：2条及以上指令并行工作（n-issue）引CPI<1

LIW（Very Long Instruction Word）一次发射长指令 or 指令包（由compiler静态调度）也是n个 零信加 硬件资源可以静态调



超流水，进一步细分流水线，f↑ ⇒ 重叠执行↑

hapter 5 DLP and TLP

uniprocessor，可利用ILP（pipeline）

- lynn
  - SISD – von Neumann Arichitecture
  - SIMD（DLP）← Vector Processor / Array Processor
  - MIMD
    - Shared Memory ← NUMA（DSP）← CC-NUMA / NC-NUMA（distributed share-mem Processor）; COMA; UMA（Symmetric MultiProcessor）
    - Message Passing – Massive Parallel Processor ← Cluster / Grid
  - MISD（unknown）

IMD：单指令多数据流，e.g. vector processor（向量数据）

PU是多线程SIMD处理器：Single Instruction Multiple Thread.

thread ⇒ blocks ⇒ grids

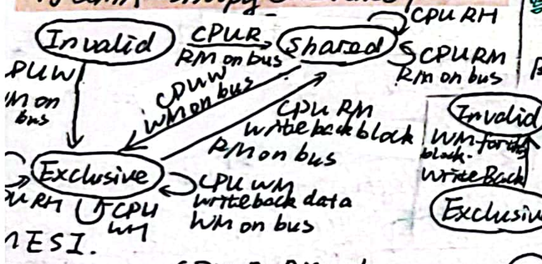
私有内存　本地内存　全局内存

Cache Coherence.

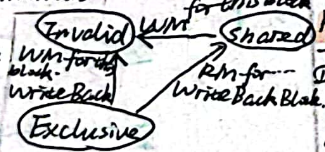
对于UMA：Snoopy coherence protocols（监听）

（Valid, Dirty）0x, 10, 11 Invalid, Shared, Exclusive

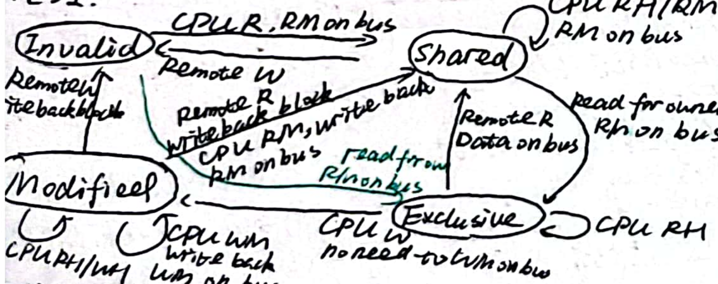
read miss 引要先写回，再 read hit

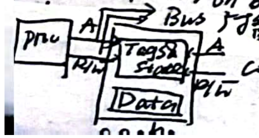


Bus对cache：



MESI.





basic

① write invalidate protocol：把其他位置数据设为无效，下次去内存取

② write broadcast protocol：将数据也广播

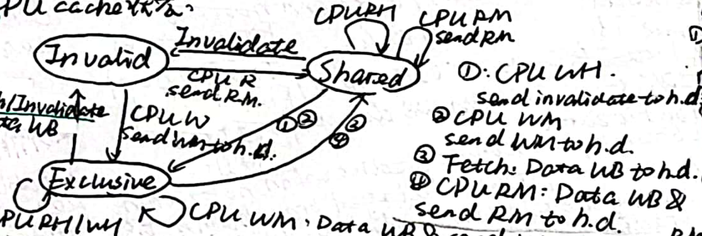
② For NUMA：Directory protocol

每个结点有一个目录，记录自己内存里block状态（shared：≥1 最新数据；uncached：没有cache，或均invalid；exclusive：一个有owner）

local node：request发出者；home node：访问地址所在结点。

remote node：拥有cache block拷贝的node

CPU cache状态：



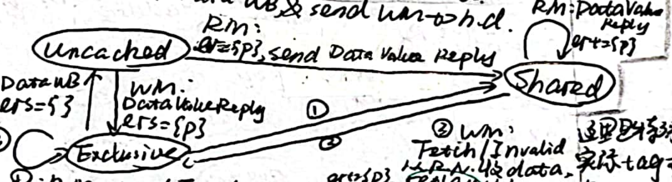
①：CPU WH. send invalidate to h.d.

② CPU WM send WM to h.d.

③ Fetch：Data WB to h.d.

④ CPU RM：Data WB & send RM to h.d.

Directory：



① RM：send Fetch to R.N.，get data from R.N. reply back to local node. ers+={P}

② WM：Fetch/Invalid 从R.N.收data，ers={P}，replay给l.n. 要ack

② WM/Invalid：Invalidate, ers={P}，Data Value Reply（由h.n.发给l.n.）给r.n.（含自己）无需从r.n.获得

Memory Consistency 横向

不同处理器向不同存储单元进行R/W，必须按什么规则

① sequential consistency：所有内存访问串行化，不同处理器之间可以interleave；同一处理器 in order. performance低，限制太多

② releaxed consistency：允许R.W乱序，但使用同步操作来保证正确.

rules：x→y 表示 x 须在 y 前执行，①：W→R，W→W，R→R，R→W

(i) total store order：放松 W→R　(ii) partial store order 放松 W→R，W→W

(iii) weak order / release consistency（release 前 acquire 后全部放松）

Hardware primitive：

不可中断的 fetch/upd 内存的指令或一条引来检索并改变一个值的指令（原子的）

① Atomic exchange：0表示free，1表示被占用　② TestAndSet　③ FetchAndIncrement

RISC-V里有 lr（load reservation）和 sc（store conditional）

lr x2, 0(x1) then sc x3, 0(x1) 这里lr给值，在地址上设置标记。

sc会判断对应位置(i)有标记，正常写入，返回0；(ii) 无标记，不写入，返回1. 清标记

spinlock：EXCH x2, 0(x1)

```
① addi x2,x0,#1.  lockit: EXCH x2,0(x1)
                          bnez x2, lockit
② try: li   x2,#1
   lockit: lw   x3,0(x1)
           bnez x3,lockit
           EXCH x2,0(x1)
           bnez x2,try
③ lockit: lr   x2,0(x1)
          bnez x2,lockit
          addi x2,x0,#1
          sc   x2,0(x1)
          beqz x2,lockit
```

exch要write，会无效其他拷贝，且增加bus traffic.

mtvec：发生异常时跳转入口

mepc：发生异常的指令；发中断时指向中断处理后恢复的位置

mcause：异常原因（硬件断点/访问存储器…）

mtval：异常地址或异常指令内容

csrrw rd, csr, rs1　csrrwi rd, csr, imm

csrrs rd, csr, rs1　csrrsi rd, csr, imm

须有 or csrrc.

Precise Exception.

epc就是异常指令，把以前 inst 提交的

Imprecise：epc是异常后的某条（流水线）

epc之前的指令若此 ……

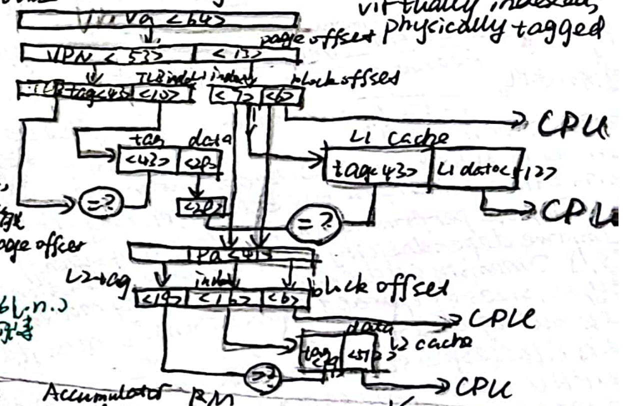
Example	IS	RO	EX	WB	IS	EX	WB	Commit
FLD F6, 34(R2)	1	2	3	4	1	3	4	5
FLD F2, 45(R3)	5	6	7	8	2	4	5	6
FMUL.D F0, F2, F4	6	9	10~19	20	3	6~15	16	17
FSUB.D F8, F2, F6	7	9	10~11	12	4	6~7	8	18
FDIV.D F10, F0, F6	8	21	22~61	62	5	17~56	57	58
FADD.D F6, F8, F2	13	14	15~16	22	6	9~10	11	59

① Scoreboard中，若结构冲突要在前一条执行后（WB后）再发射；若数据冲突，要在WB后再RO；EX结束后，若有WAR, WAW要等待，如最后一条，在DIV RO的下一个周期才写回.

② Tomasulo中，Load在IS后要用额外一拍算地址再EX；有数据冒险，要在WB下一拍才能EX（如MUL）；不用担心WAR, WAW

③ 带ROB就加上了commit一列

virtually indexed; physically tagged



这里因为特殊，实际tag可能有PPN + page offset 的高位

Loop-unrolling：注意边界情况

MTTF：Mean Time To Failure

MTTR：------ Repair

MTBF ------ Between ------ = MTTF + MTTR

$$\text{availability} = \frac{MTTF}{MTBF}$$