Cache

注意word大小是4B还是8B，1个block = 几个word

① mapped cache：位数计算

#index = $\log_2$(#block num)　#byte_ofs = $\log_2$(block size)

都是bit为单位　（注意有题用word_offset）

#tag = #Address − #index − #byte_ofs

#data = ? bits　#blocks = 2^n个

总大小 2^n(#tag + #data + #valid + dirty)

hit：
- write through：立即把更改写入内存和cache（慢）
- write back：不立即写入，有dirty bit。总带宽更小（要被覆盖时才写入）replace (v.)

write through可引入write buffer，但需CPU产生write速率 < 内存写入速率

miss时：
- write allocate：把block从内存先写到cache
- write around (no allocate)：直接写入内存

② set associative：1个set多个block（可任意选1个）

#index = $\log_2$(set#) = $\log_2\left(\frac{\text{cache blocks}}{\text{associativity}}\right)$

index代表set编号

byte_ofs 还是看1个block大小

block size增加，可减少miss，但miss时访存次数增加（解决：增大内存宽度）

假设内存宽1word　miss：1 + 4(1 + t_{mem})

cache 4 word　62

set associativity ↑，miss ↓，但比较时间↑，tag长度↑

direct map 相当于asso=1，full asso=n

CPU计算：多设N条指令，无miss：N×CPI

有miss：N×CPI + N×Ins cache miss rate × Penalty + N × Load/store% × Data cache miss% × Penalty.

读写带宽（RAM到Cache）

write allocate：write miss时写到内存和cache

read：(Ins miss + Read miss + Write miss) / Cycles

write：Write miss / Cycle

write back：(Read + Write) × miss rate × dirty% / Cycles.

虚拟内存：1个page大小

virtual page num | page offset → physical | page off

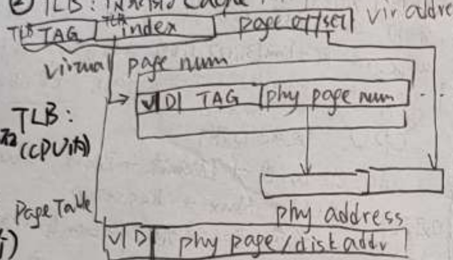
#page offset = $\log_2$(page size)

#virtual page num = 虚拟地址位数 − #page offset

① 页表：每个虚拟page num对应页表中1项

- 页表entry数 = $2^{\text{vir page num}} = 2^n$ / pagesize
- 每个entry大小 = 物理地址位数m − #offset + 2（2是valid和dirty）

② TLB：页表的Cache



TLB	PT	cache
H	H	M
M	H	M
M	H	H
M	M	M (PF).

virtual page num 对应PT的index.

TLB如果是full associative，tag = vir page num

否则还要把拆成tag和index

I/O

Amdahl's law　$T_{new} = T_{par}/100 + T_{seq}$

Speedup = 1/(1 − F_{par} + F_{par}/100)　F_{par}是并行部分比例

time = $t_s + \frac{0.5}{\text{xxx RPM}} + t_T$ + control　(track, sector, platter)

MTTF 无故障时间　A：修故　To：容错　Fore：预见

MTTR 修复　可用性 = MTTF/(MTTF + MTTR)

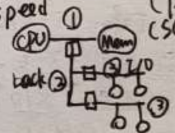
RAID：Redundant Arrays of Inexp Disk

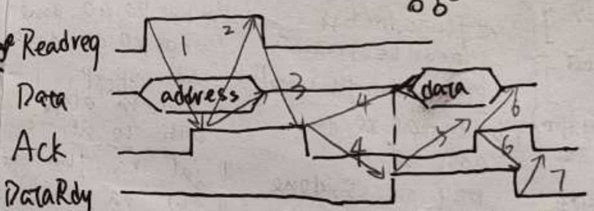
RAID	允许坏的	Data disk	Check disk（额外的盘）	
0	0	8 (很快)	0 (不能恢复)	用的最多，无冗余，并行
1	1	8 (最贵)	8	mirror，每个盘有一个备份，写慢
2	1	8	4	ECC码
3	1	8	1	Bit interleaved：奇偶校验.
4	1	8	1	每个block有自己的校验位
5	1	8	1	
P+Q	2	8	2	

bandwidth = x bytes/时间

Bus:
1. processor_memory：short，high speed，custom
2. backplane：high speed (PCI)
3. I/O：lengthy (SCSI)（最多7 disk）

同步（有clock skew）异步





1. Mem看到Readreq，读address，同时Ack=1
2. I/O看到Ack=1，Readreq=0
3. Mem看到Readreq=0，Ack=0
4. Mem准备data，DataRdy=1
5. I/O看到DataRdy，Ack=1
6. Mem看到Ack，DataRdy=0 释放data
7. IO Ack=0.

handshake 40ns 内存200ns：1:40　2,3,4与内存并行

max(40×3, 200) = 200　5,6,7：40×3

增大bus带宽：① 增大data ② addr与data分开 ③ 多word

IO：
- Polling：处理器定期check status bit。缺点：浪费CPU时间
- Interrupt：I/O给中断（数据准备好），CPU可以在数据准备好时才到达
- DMA：direct memory access，IO设备直接和内存交互

多级页表：outer page (P1) | inner page (P2) | byte offset

外层可以不满的，二级页表个数 $\frac{2^{32}}{2^{\text{inner}+\text{offset}}} \times 2^{13}$

R型：
- mul rd, rs1, rs2：rd = rs1 * rs2低6…
- mulh rd, rs1, rs2：rd = rs1 * rs2 >> 6…
- div rd, rs1, rs2. rd = rs1 / rs2
- divu
- rem　rd = rs1 % rs2
- remu

mulhu：mulh，无符号×无符号

mulhsu：mulh，有符号×无…

by Setsuna@cc98

Prelude time = CPI × n_{ins} / fclock f: 1GHz = $10^9 s^{-1}$

MIPS = (f / CPI) × 10^{-6}

Cost per die = Cost per wafer / (Dies per wafer × yield)

yield = (1 + (Defect per area × Die area / 2))$^{-2}$

Dies per wafer = Wafer area / Die area

速度 SRAM(cache) > DRAM > Flash > Hard disk （SRAM、DRAM: volatile）

risc-v 原子指令 lr.d
sc.d

小数转二进制: 0.6 = 0.1001 1001 …
（0.6×2 = 1.2 →1，0.2×2 = 0.4 →0，0.4×2 = 0.8 →0，0.8×2 = 1.6 →1，0.6×2 = 1.2 →1）

A:10 B:11 C:12 D:13 E:14 F:15

Arithmetic IEEE754

bias = $2^{E-1} - 1$ E为指数长 F为尾数长

exp (0 ~ 2^E-1)	fra	数
0	0	0
0	≠0	$0.fra \times 2^{1-bias}$
$1 \sim 2^E-2$	—	$(-1)^S (1.fra) \cdot 2^{exp-bias}$
2^E-1	0	±inf
2^E-1	≠0	NaN

范围 $[2^{1-bias}, (2-2^{-F}) 2^{2^E-2-bias}]$，精度 2^{-F}

总	S	E	F	bias	范围
32	1	8	23	127	$\pm[2^{-126}, (2-2^{-23}) \cdot 2^{127}]$
64	1	11	52	1023	$\pm[2^{-1022}, (2-2^{-52}) 2^{1023}]$

加法: ① 小指数对齐到大指数（fra 前补 0） ② 相加减 ③ 规格化 $1.xxx \times 2^y$ ④ 舍入

乘法: ① 指数相加，再减 bias ② 1.frac 相乘并规格化 ③ 符号

（舍入）guard round sticky
… 0/1 0/1 （round 后面如有非零，就 sticky=1）
1.0000100 10 00011 guard 1, round 0, sticky 1

round to nearest even 23.5→24 −23.5→−24
up (inf) down (−inf) 0

… 300000 (+inf)

…: $C_{out} = a \cdot b + (a+b) \cdot C_{in}$ $g = a \cdot b$ $p = a+b$

$g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$

Instruction ① little endian

a	0A
a+1	0B
a+2	0C
a+3	0D

存的是 0D0C0B0A

② 指令的位宽是固定的

③ dword: 64 bit word: 32 bit

ld、addi、jalr 是 I 型，sd 是 S 型，beq 是 SB 型，lui、auipc 是 U 型，jal 是 UJ 型。SB 和 UJ 2 字节对齐

sll 左移 srl 右移补 0 sra 右移补 1

slt rd, rs1, rs2 rd = rs1 < rs2 ? 1 : 0（有符号比）

sltu 无符号 lw, sw 高位补 0

beq rs1, rs2 PC += offset

jal rd, imm rd = PC+4, PC += offset

jalr rd, imm(rs1), rd = PC+4, PC = imm + rs1

lw 是 12 位 imm，sw 是 7 位

lui: rd, imm rd = {imm[31:12], 12'b0}

jal 立即数 i[20,1]，21 位，$[-2^{20}, 2^{20}-2]$
[PC − 0x100000, PC + 0xFFFFE]

beq i[12:1] $[-2^{12}, 2^{12}-2]$ [−0x1000, 0xFFE]

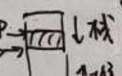
x3: gp 大立即数 lui + addi 如果 imm[11] = 1，就把 imm+1
x4: tp
x8: fp

C 转汇编: x5−x7 (t0−t2) 和 x28−x31 (t3−t6) 不保证调用后不变。x1 (ra); x2 (sp)。x10−x17 (a0−a7) 是 8 个参数，x10 是返回值，x8−x9, x18−x27 是 Saved.

```
int f(int a, int b){
  if(a<=0||b<=0) return 0
  else return f(b-2,a-b)}
```

```
fun: bge x0, a0, done
     bge x0, a1, done
     addi, sp, sp, -4      int: 4 用 lw
     sw  ra, 0(sp)         ll: 8 用 ld
     addi t0, a1, -2
     jal  ra, fun
     lw   ra, 0(sp)
     addi sp, sp, 4
     jalr x0, 0(ra)
done: addi a0, a0, 0
     jalr x0, 0(ra)
```

```
fib: beq a0, x0, done
     addi t0, x0, 1
     beq x10, x5, done
     addi sp, sp, -8
     sw  ra, 0(sp)
     sw  x10, 4(sp)   //n
     addi x10, x10, -1   //n-1
     jal ra, fib
     lw  t0, 4(sp)    //n
     sw  x10, 4(sp)   //存 fib(n-1)
     addi x10, x5, -2  //n-2
     jal ra, fib
     lw  t0, 4(sp)
     addi x10, x10, t0
     lw  ra, 0(sp)
     addi sp, sp, 8
done: jalr x0, x1.
```



ra, sp 也要保证前后不变

ASCII 8 位 lbu

while((x[i] = y[i]) != 0)) i++

长距离跳转，beq 12 位不够
```
bne x1, rs2, Exit.
jal Label
```

x < 0 || x > n bgeu x20, x11, L1

交换
```
xor x3, x4, x3
xor x4, x4, x3
xor x3, x3, x4
```

取反 xori x3, x3, -1
移动 addi x3, x2, 0

auipc: rd = PC + {imm[31,12], 12'b0}
lui x6, imm[31:12] + imm[11]
addi x6, x6, imm[11:0]

```
strcpy: addi, sp, sp, 8
        sd, s3, 0(sp)
        addi s3, x0, 0
L1:     add t0, s3, a1
        lbu t1, 0(t0)
        add t2, s3, a0
        sb  t1, 0(t2)
        beq t1, x0, done
        addi s3, s3, 1   //i++
        jal x0, L1       //for 循环
done:   ld  s3, 0(sp)
        addi sp, sp, 8
        jalr x0, 0(sp).
```

CPU 延时分析

add: Clk-to-Q + IMem 读 + Reg 读 + Mux + ALU + Mux + Reg setup

addi 考虑 Reg 读和 ImmGen 取 max

lw: 在 addi 基础上 (imm-gen) + DMem 读

sw: Clk-to-Q + IMem 读 + ImmGen + Mux + ALU + DMem 写（不同 Reg setup）

beq: Clk-to-Q + IMem 读 + Reg 读 + Mux + ALU + AND + 2 选 Mux + reg setup

jal: Clk-to-Q + IMem 读 + ImmGen + Adder + Mux + Reg Setup（少 1 个 mux，无条件）

fadd.s（单精）
fadd.d（双精）
flw fsw

PC-relative beq
立即数 lui
间接 jalr (register addressing)
base ld x4, 8(x2)

流水线周期: 5 阶段取 max

单流水线时间: max(t_i) × 5

指令非流水线 = $\sum_{i=1}^{5} t_i$ = 时钟周期:

流水线延时分析: 每个阶段都要加 clk-to-q 和 setup

EX: clk-to-q + Mux + ALU + setup

WB: clk-to-q + Mux + setup

单个指令，流水线并没有缩短时间。也没有缩小 CPI。但 clock 周期↓，吞吐量↑，总时间↓

Pipline:

① Structure Hazard: 加寄存器

Double Bump: 寄存器堆上升沿写，PC 和 pip regs 下降沿写，这样 WB 和 ID 可同时（Data hazard 只需 2 个 stall）无 bump 就 3

② Data hazard:
Foward A 和 Foward B 分别对应 rs1, rs2
fowarding 无法避免 load-use（插 1 个 nop）
10: 从 ALU 01: 从 Mem 或更早的 ALU

③ Control hazard: 后续指令会执行下去 2 条指令
stall
分支预测

跳转地址计算在 MEM

Data hazard（无前递）

冲突①
F D E M W
F D D D E M W
F F F D D D E M W
冲突②
F F F D E M W
F D D D E M W

data③ stall，预测在 ID

Cache 续

层级越高，耗时越少

regs CPU High
L1-Cache Mem (chip)
L2-Cache Mem (SRAM)
(DMem) Low
hierachy

temporal: 时间 如循环指令
spatial: 空间 如数组顺序访问

Miss 分类
- compulsory miss: 第一次访问
- capacity miss: 内存大小有限，如 32K Cache 放 128K 数据 block size↑ capa↓
- conflict miss: 非 full-assoc 导致，对齐方式 如两个小数据，映射到同一个地方（cache 中 4B 存一 block） assoc↑ conflict↓

Exception: sepc: 当前遇到异常的指令
scause: 异常原因
status 里面有 MIE，标志异常使…
direct: 跳到固定地址
vector: 不同地址: base + vec