cache data size = 32KB, block size = 2word, word size = 64bit = 8B

byte-addressed (如果是word-addressed, ofs=1，共有16byte)

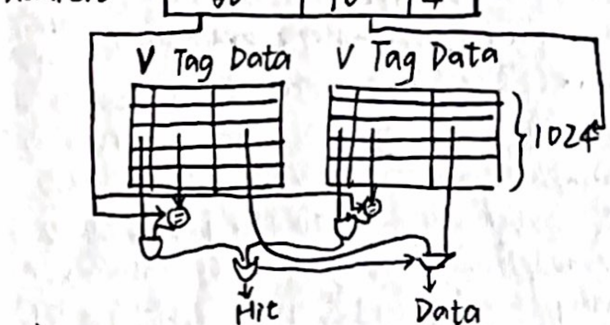
① block size = 16B → block ofs = 4bit (地址中的一个1代表一个byte)

② cache data size = #blocks × block size, $\#blocks = \frac{32KB}{16B} = 2^{11}$

③ 对于direct-mapped, #sets = #blocks = 2^{11} → index = 11 bit
tag = 64 - 4 - 11 = 49 bit

④ 对于2-way组关联, #sets = $\frac{\#blocks}{2(路数)} = 2^{10}$ → index = 10 bit
关联度是to有深
tag = 64 - 4 - 10 = 50 bit 如果是full-way, 不需要index

⑤ cache total size = (valid + tag + block size) × #blocks
直接映象 = $(1+49+16\times8)\times2^{11}$ bit = 44.5KB (÷2^{13}得KB)
2-way = $(1+50+16\times8)\times2^{11}$ bit = 366592bit = 44.75KB



tag	index	ofs
63-10	9-5	4-0

#block = 2^5 blocksize = 2^5B

读操作的命中/失效/替换

byte-address

Hex	400	C1C
Dec	1024	3100

① 方法一, 转成二进制地址

addr(16)	addr(2)	tag	index	ofs	h/m	content
400	010000000000	1	0	0	m	Mem[x400]-Mem[x41F]
~~3100~~ C1C	110000011100	3	0	28	m	Mem[xC00]-Mem[xC1F]

② 方法二, 使用十进制

addr(10)	ofs	index	tag	content
1024	1024%32=0	1024/32=32, 32%32=0	1	32×32=1024, 1024~1055
3100	3100%32=28	3100/32=96, 96%32=0	3	96×32=3072, 3072~3103

mem和disk访问成本高使用write back

Write Hit <
- write back: 只改cache, 需dirty bit, 1个块的buffer
- write through: 同时改cache和mem, buffer最小只要1word

Write Miss <
- write allocate: 先读入cache, 再写
- write around (no write allocate): 不读入cache, 直接改mem

概念知识: memory用DRAM, cache用SRAM。DRAM分成 ① classic 异步 ② SDRAM (synchronous) 同步 ③ DDR SDRAM 2倍速率 ④ DDR2 4倍速
temporal locality 时间局部性, 其本身在之后会访问; spatial ~, 周围数据~, 增大blocksize可以提高。Hierarchy: L1 cache(片上) → L2 cache → Main Memory → Disk. 常用的替换策略: Random, FIFO, LRU (如2-way, 使用Ref bit) Memory Design 1个周期送读信号, 15个周期读内存, 1个周期送回data, block size = 4word, 1word = 4byte ① 1-word-wide memory, bandwidth = bytes/clock cycle. time to transfer 1 block (miss penalty) = 1+4×(15+1) = 65 CLKS, bandwidth = $\frac{4\times4}{65} = \frac{1}{4}$ ② 4-word wide, miss penalty = 1+15+1 = 16, bandwidth = $\frac{16}{17}$ ③ 4-way interleaved 并行从4个memory bank读4个word, miss penalty = 1+15+4×1 = 20

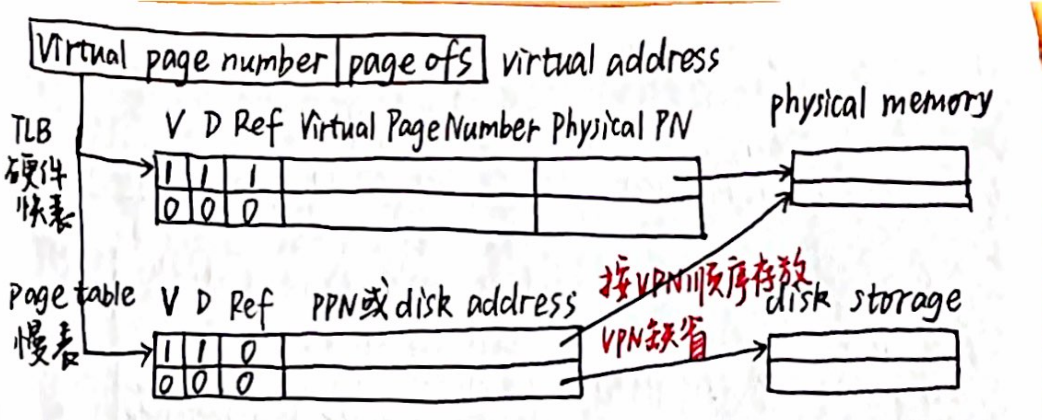
1 2 3 4 miss rate = 100次10失败n次, misses/inst = 100条inst失败n次 (每条inst 10≥1)

AMAT (Average Memory Access Time) = hit time + miss rate × miss penalty

例1, 计算CPI with stalls. inst cache miss rate = 2%, data cache miss rate = 4%, perfect CPI = 2, miss penalty = 100 cycles, LD+SW = 36%
① inst miss cycles = I×1×2%×100 = 2I
② data miss cycles = I×36%(访问DMem次数) × 4% × 100 = 1.44I
③ CPI with stalls = (2I + 2I + 1.44I)/I = 5.44

例2. L1 cache ⊆ L2 cache ⊆ main memory. perfect CPI = 1, clock freq = 5GHz, memory access/inst = 1.3, L1 cache miss rate = 2%, L2 cache miss rate = 0.5%, L2 cache access time = 5ns = 25 CLKS, memory access time = 100ns = 500 CLKS ① 如果没有L2 cache, total CPI = 1 + 1.3×2%×500 = 14
② 有L2, total CPI = $\underset{\text{perfect}}{1} + \underset{\text{IO次数}}{1.3} \times (\underset{\text{L1缺失}}{2\%} \times \underset{\text{L2hit}}{25} + \underset{\text{L2缺失}}{0.5\%} \times \underset{\text{访问mem}}{500}) = 4.9$

source of misses ① compulsory misses, 第一次访问block ② capacity misses, 由于cache size限制 ③ conflict misses, 非全关联cache的index冲突: 不会出现在全关联cache中, 增加关联度无用



PPN		Page ofs
tag	index	block ofs

⇒ 去cache找data

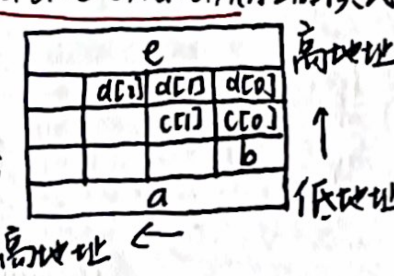
VPN $\xrightarrow{\text{TLB Miss}}$ 找页表 $\xrightarrow{\text{Miss}}$ 在disk中; $\xrightarrow{\text{Hit}}$ 在memory中, ~~但不在cache中~~

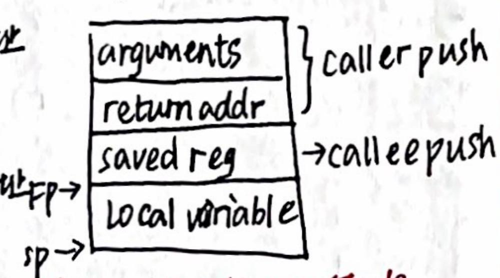
VPN $\xrightarrow{\text{TLB Hit}}$ 去cache读 $\xrightarrow{\text{Miss}}$ 在memory中, 未取至cache; $\xrightarrow{\text{Hit}}$ 找到

TLB	PT	Cache	原因
H	H	M	在内存, 未取入cache
M	H	H	在内存, cache, 未取入TLB
M	H	M	在内存, 不在cache
M	M	M	在disk
H	M	X	X, TLB⊆PT
X	M	H	X, cache⊆mem

memory little endian (小端模式) 低字节存在低地址

int a;
char b;
char c[2];
char d[3];
float e;





RISC-V特点之一: load/store结构 只有load/store指令会访问存储器, 其他指令不会

R add rd, rs1, rs2

I addi rd, rs1, imm12
ld rd, imm12(rs1) rd ← Mem[rs1 + sext32(imm12)]
jalr rd, imm12(rs1) pc ← rs1 + sext32(imm12), rd ← pc+4

S sd rs2, imm12(rs1) Mem[rs1 + sext32(imm12)] ← rs2

SB bne rs1, rs2, label ofs = label地址 - 当前PC $[-2^{12}, 2^{12}-2]$

imm[12] imm[10:5] rs2 rs1 f3 imm[4:1] imm[11] opcode7

sext20	1	6	4	1'b0

如PC=60012, L1=60024, ofs=12, imm[4:1]=0110, 其余全0
PC=60024, L1=60012, ofs=-12, imm[4:1]=1010, 其余全1

UJ jal rd, label rd ← pc+4, pc ← pc + sext32(imm20)

imm[20] imm[10:1] imm[11] imm[19:12] rd opcode7

sext12	8	1	10	1'b0

ofs ∈ $[-2^{20}, 2^{20}-2]$

U lui rd, imm20 rd ← {imm20 + zero12}

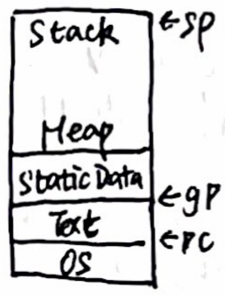
伪指令 ① NOT x10 ⇔ xori x10, x10, -1; ② bgt rs1, rs2, L1 ⇔ 也 blt rs2, rs1, L1
③ ble rs1, rs2, L1 ⇔ bge rs2, rs1, L1 ⑧ 32位常数 lui+addi
④ if (x20<0 or x20≥x11) goto IndexOutofBound
bgeu x20, x11, IndexOutofBound
⑤ mv rd, rs ⇔ addi rd, rs, 0 ⑥ J Label ⇔ jal x0, label ~~jalr x0, 0(label)~~
⑦ snez rd, rs rd = (rs!=0)1:0 ⇔ sltu rd, x0, rs (判断非0)

t寄存器 (临时变量, 一般不保护) x5-x7 (t0-t2), x28-x31 (t3-t6)
s寄存器 (函数的本地变量, 由callee在函数首尾压/出栈) x8-x9 (s0-s1) x18-x27 (s2-s11)
参数与返回值 (参数由caller保护, 返回值无须恢复, 是子程序回传给caller的纽带) x10-x17 (a0-a7)
返回地址 x1(ra), 由caller保护; 栈顶指针 x2(sp)
递归函数: 既是caller又是callee, ra, 参数, 中间结果都要压栈保护

```
int fact(int n)
  if n<1 return 1
  else return n*fact(n-1)
n—x10, fact(n)—x10
fact: addi sp,sp,-16
      sd   x1,8(sp)    保护
      sd   x10,0(sp)
      addi x5,x10,-1
      bge  x5,x0,L1
      addi x10,x0,1    //return 1
      addi sp,sp,16    //无须恢复x1和x10
      jalr x0,0(x1)
L1:   addi x10,x10,-1
      ~~add x6,x10,0~~
      jal  x1,fact
      add  x6,x10,x0   //x6=fact(n-1)
      ld   x10,0(sp)
      ld   x1,8(sp)    反向restore
      addi sp,sp,16
      mul  x10,x10,x6
      jalr x0,0(x1)
```

没变 变成返回值

```
int fib(int i)        i-x10, fib(i)-x10
if(i==0) return 0;
else if (i==1) return 1;
else return fib(i-1)+fib(i-2);
fib: beq x10, x0, done
     addi x5, x0, 1
     beq x10, x5, done
     addi x2, x2, -8      32位寄存器
     sw   x1, 0(x2)
     sw   x10, 4(x2)
     addi x10, x10, -1
     jal  x1, fib
     lw   x5, 4(x2)   // x5=i
     sw   x10, 4(x2)  // push fib(i-1) on stack
     addi x10, x5, -2     中间结果入栈
     jal  x1, fib
     lw   x5, 4(x2)   // x5=fib(i-1)
     add  x10, x10, x5
     lw   x1, 0(x2)
     addi x2, x2, 8
done:
     jalr x0, 0(x1)
```



caller save: x1 (ra), argument, 重要t寄存器; callee: s寄存器压栈

Inst	ALUop	RW	ALUsrc2	MR	M2R	MW	Imm	taken	jal	lui	jalr
ALURR	f3/7	1	0(rs2)	0	0	0	X	0	0	0	0
ALURI	f3	1	1(imm)	0	0	0	I12	0	0	0	0
load	+	1	1	1	1	0	I12	0	0	0	0
jalr	+	1	1	0	X	0	I12	1	0	0	1
store	+	0	1	0	X	1	S12	0	0	0	0
branch	cmp	0	0	0	X	0	SB12	Xcmp	0	0	0
jal	X	1	X	0	X	0	UJ20	1	1	0	0
lui	X	1	X	0	X	0	U20	0	0	1	0

latency = time between inst issued and complete; throughput = 每秒钟完成的指令条数。结构竞争解决方式 ①分离IMem和DMem, 防止IF和MEM撞车 ②对于RegFile, 前边沿写, 随后读 (double bump) ③浮点加法组件不流水, 或增加组件个数 (not common case) 数据竞争成因: 数据依赖关系、RAW, WAR, WAW (乱序流水线中可能会有竞争)

way1. Stall (硬件或软件插入nop)

```
① add a0,_,_    IF ID EX MEM WB
nop add _,a0,_     IF ID ∅   ∅   } 停2拍
nop                   IF ID  ∅
                         IF  ID
                             IF
```

stall时做的事
① pc不变
② IF/ID不更新
③ 传给EX全0信号

```
② add a0,_,_    IF ID EX MEM WB
   other           IF ID EX  MEM
nop add _,a0,_        IF ID  ∅   停1拍
                         IF  ID
                             IF
```

Stall = regWrite(EX/M) && ((opcode(ID) == R|I|S|B && rd(EX/M) != 0 && rd(EX/M) == rs1(ID)) || (opcode(ID) == R|S|B && rd(EX/M) != 0 && rd(EX/M) == rs2(ID)))

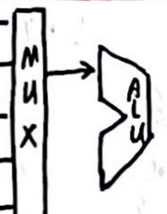
```
③ beq a0,a1,label  IF ID EX MEM WB
                      IF ID ∅   ∅   } 停3拍
                         IF ∅   ∅
                            IF  ∅
                                ID
```

stall = EX branch·cmp-res + jal(EX) + jalr(EX)

① pc可变
② ID级清零
③ EX级清零

way2. forwarding source (产生数据) → sink (使用数据, ALU和DMem)

① ALU sink, 四种来源 (若为第2个来源则有5个入口)
① ID级从RegFile中读出的rs1
② EX/MEM后的ALU-res
③ MEM/WB后的ALU-res
④ MEM/WB后的DMem-out
⑤ ID/EX后的imm32 (rs2入口)




```
② add a0,_,_     IF ID EX MEM
   add _,a0,_       IF ID EX

③ add a0,_,_     IF ID EX MEM WB
   other            IF ID EX  MEM
   add _,a0,_          IF ID  EX

④ lw a0,_        IF ID EX MEM WB
   other            IF ID EX  MEM
   add _,a0,_          IF ID  EX
```

特殊情况 ① double hazard

```
add x1,x1,x2    IF ID EX MEM WB
add x1,x1,x3       IF ID EX  MEM
add x1,x1,x4          IF ID  EX
```

用EM→EX而非ban

② lw a0,_ 停一拍
add _,a0,_

DMem sink,
① EX/MEM后的ALU-res (正常)
② MEM/WB后的ALU-res
③ MEM/WB后的DMem-out

② add x1,_,_
   sw x1,_

③ lw x1,_
   sw x1,_

控制竞争 ① predict-not taken CPI = 1 + (Jmp% + branch% × taken%) × 3 stall
② 提前至ID段计算pc-branch, stall=1, 问题: add a0,_,_; beq a0,_,_ 要stall 1; ld x1,_, beq x1,_ 要stall 2

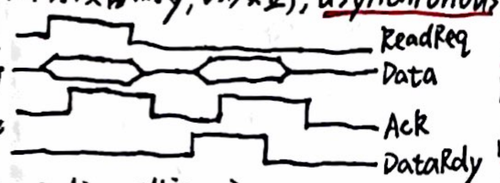
IF ID EX MEM WB clock cycle time (single) = 1250 ld total latency = 1250/1350
250 350 150 300 200 clock cycle time (pipeline) = 350

$$\text{Speedup for pipeline} = \frac{\text{CPI unpipeline}}{\text{CPI pipeline}} \times \frac{\text{clock cycle unpipelined}}{\text{clock cycle pipelined}}$$

CPI pipelined = 1 + stall cycles / Inst

mtvec base[31:2] mode[1:0] 00: PC←base mcause 1中断, 0异常 [30:0] 原因
csrrw rd, csr, rs1 csrrwi rd, csr, zimm5, csrrs rd, csr, rs1 按位或 csrrc 按位与非
① 准备进入中断: mstatus[3] & mip[i] & mie[i] 条件
② 准备: mepc←PC, pc←mtvec, mstatus[7]←mstatus[3] 改mcause
③ 返回: mret: pc←mepc (或+4, 为异常), mstatus[3]←mstatus[7]

I/O三特性: behavior (read once, Input, output (write only, not read), storage (r&w)), partner, data rate. dependability (可信赖性) the quality of delivered service (accomplishment & interruption), 可靠性 reliability MTTF (failure) MTTR (repair) MTBF (between failure) = MTTF + MTTR availability = MTTF/MTBF 增加MTTF — fault avoid / tolerance (增加冗余度) / forecast。bus有control line 和data line, 有input/output两种操作。三种线: processor-memory (short high speed, custom design), backplane (high speed, standardized, PCI), I/O (lengthy, SCSI) 两种方式: synchronous (使用时钟和同步协议, 要求所有设备同步, bus短), asynchronous 用handshaking (ReadReq, DataRdy, Ack)



三大芯片: 处理器、北桥、南桥 (南北合称芯片组) 北桥处理高信号, 南桥负责I/O
两种方式访问设备: memory-mapped I/O, special I/O inst (give device num 和 command word). polling, interrupt, DMA (direct memory access), 处理器发布任务, 设备直接和内存交互通过DMA, 而不用cpu。完成后告诉cpu。interrupt bit mask 可以改变任务的优先级。RAID (Redundant Arrays of Inexpensive Disks)
RAID0 无备份 RAID1 镜像备份, 每次写2个盘。RAID3 Bit-Interleaved Parity, 按位添加一奇偶校验盘, 用于恢复坏盘。RAID4 block 坏处: small write 都要访问P盘, 容易坏; 写操作不能并行 1次写 = 2次读 + 2次写 RAID5 可以并行写, P盘不易坏

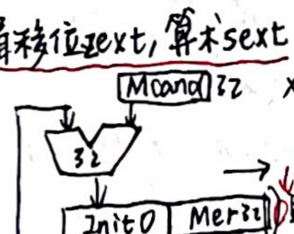
n位 unsigned $[0, 2^n-1]$ signed $[-2^{n-1}, 2^{n-1}-1]$ sign-magnitude & 反码 & 补码
移码 = 补码最高位取反 $[X]_{bias} = X + 2^n$ Float 1 8 23 exp[1, 254] -127 → [-126, 127]

	2'sC	bias
+1011	01011	11011
-1011	10101	00101

255 其余0 → ±∞ double 1 11 52 exp[-1022, 1023]
全0 → 0, 否则非规格化 overflow: exp ≥ 255

浮点加法 ① 获取双符号位的尾数和阶码 (补码表示的[-126, 127])
② 对阶小阶向大阶对齐 (尾数右移) 1.101 → 001101 -1.101 → 110011
③ 规格化
01.XX → 00.1XX 10.XX → 11.0XX 右归, frac右移, exp++
00.0XX → 00.1X 11.1XX → 11.0X } 左归, frac左移, exp--, 循环
#11.00001 → -00.11110 → -1.1110 frac
④ 舍入 ⑤ 溢出 exp > 127 上溢; exp < -126 下溢

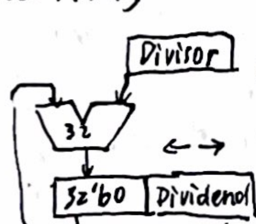
逻辑移位zext, 算术sext 浮点乘法, fracA + fracB, exp1 + exp2 - 127



01: +Mcand本身, product右移
10: +Mcand补码, 右移
1/00: 只右移
n位Mer就移动n次

Booth: 当前位和后位
00 #11 10 01
0111001100
+s-s-s+s-s

① R左移 ② R←R-divisor
③ R>0, R左移, 低位=1
R<0, 先加回, R左移, 低位=0
④ 循环②③ 32次
⑤ R右移



结果: 00010011
余数 商

Virtual Addr

8 (一级页表)	11 (二级页表)	12 (page ofs)

PTE=6B PTE=4B

最大页表空间 ~~2048~~ $2^8 \times 2^{11} \times 4B + 2^8 \times 6B$

熊 3200105278