Chap04 (Con.)

流水线 Pipeline

Hazards: Structure ~: IF.MEM 同时访问 / Regs / 未完全流水线
$\Downarrow$
要分开放 (否则必要stall)

double bump (双重触发): Reg 上升沿写, 下降沿读

stall

有时允许结构冒险, 为了降低成本或延迟, 在冒险不多的情况下.

Data Hazards.

ForwardA (B) = 00 没有hazard; 10. 从EX/MEM中; 01. 从MEM/WB中

detect:

① EX/MEM.RW (RegWrite = 1), Rd ≠ 0, Rd = ID/EX.Rs1 (2) => 10

② not ① and. MEM/WB.RW = 1, Rd ≠ 0, Rd = ID/EX.Rs1 (2) $\Downarrow$ 01

③ load-use: ID/EX.MR = 1, Rd = IF/ID.Rs1 (2) => stall ← nop. pc保持 flush ... 提前detect

④ ld x1, (-) add / sd x1, (-). 加一条forwarding. 由WB → MEM

Control Hazards:

① ID阶段 stall × 3 ② predit-not taken / taken

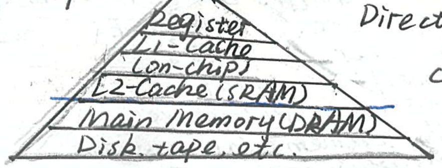
③ 提前在ID阶段计算出是否跳转, 以及跳转目标

处理data hazard (ID)

```
IF ID  EX       add x1    ld  x1
   IF  ID ○ ○   stall     stall
       ID       beq x1,   stall
                          beq x1
```

④ Dynamic predition: 1-bit, 2-bit

Chap05 Memory



CPU给出addr

Direct-Mapped: tag | index | byte offset

cache: V | Tag | Data

Read Miss: 数据/指令, 从内存中取到cache. 再read.

Write Hits: ① write-back 写回. 只写cache, dirty bit ② write-through 直写 同时写cache和内存 ↳ write buffer, buf满了也要stall

Write Misses: ① write-allocation, 把块从内存中读出再写. 常与w-back搭配 ② write-around 只往内存写, 常与w-through搭配

CPU Time = (CPU execution + $\underbrace{\text{Memory stall cycles}}_{\text{R/W Rate} \times \text{miss rate} \times \text{penalty} = R + W}$) × Clock cycle time

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

① set-associative index: 组号. n-way表示n路, 即一个组有n个块 (不是有n个组!)
fully 全. 只有一个组; direct <=> 1-way
n↑, miss rate↓

replacement. LRU/FIFO

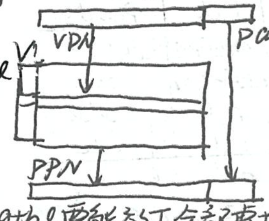
② multilevel caches
L1: hit time↓
L2: miss rate↓ 大

e.g. CPU: CPI = 1.0 5GHz. 2% miss rate, 100ns penalty => 原来: CPI: $1.0 + 2\% \times \frac{100}{0.2} = 11.0$
改进后. L2-cache 5ns访问, miss rate 0.5%
改后: CPI: $1.0 + 2\% \times \frac{5ns}{0.2} + 0.5\% \times \frac{100}{0.2} = 4.0$

Chap05 (Con.)

Virtual Memory

把主存当作磁盘的cache (全相联, 写回, dirty)
page table 在memory中.



页表只存页号!

page fault: 页不在内存中

pgtbl要能放下全部虚拟页号!

Translation Lookaside Buffer (TLB) 相联度高

Chap06 Storage, I/O

I/O三特性. ① Behavior 输入/输出/存储; ② Partner; ③ Data rate

性能衡量: ① Throughput ② Response Time ③: ①+② 不同app关心不同

Disk: access time = seek + rotational latency + transfer + controller

Dependability:

MTTF Mean Time To Failure 平均无故障时间

MTTR Mean Time To Repair 平均修复时间

MTBF Mean Time Between Failure 平均故障间隔时间 = MTTF + MTTR

Availability = $\frac{MTTF}{MTTF + MTTR}$

↑MTTF: ① fault avoid (提前换) / ② tolerance (备份) / ③ forecasting

RAID: Redundant Arrays of Inexpensive Disks

level	最少故障存活数	example disks	check disks
0	0	8	0
1	1	8	8
2	1	8	4
3	1	8	1
4	1	8	1
5	1	8	1
6	2	8	2

RAID1 镜像备份, 每次写两个盘.

RAID3. ▯ ▯ ▯ ▯P bit-interleave parity 按位添加奇偶校验盘, 用于恢复

RAID4 每个盘有自己的detection field (在P中) 但small write要访问P盘, 容易坏; 因此操作不能并行.

RAID5: P盘不定, 不易坏, 可以并行写.

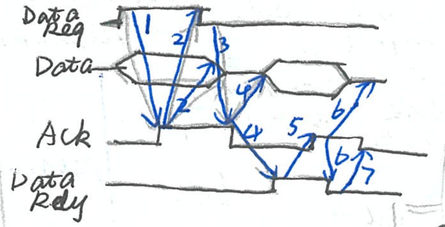
RAID6. 有P. Q两位, 可以恢复出两个盘的内容

Bus

有两种线, control + data lines, 有input/output两种op.

types of buses: processor-memory, backplane, I/O

两种方式. synchronous (使用时钟和同步协议, bus短)
asynchronous: handshaking 握手协议



I/O 向CPU

三大芯片: 处理器, 南桥, 北桥 (合称芯片组)

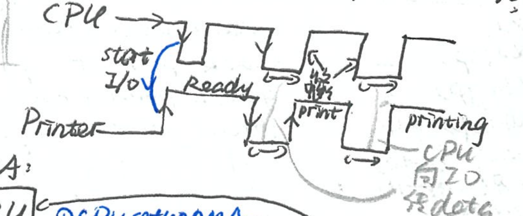
访问I/O设备

① memory-mapped I/O

② 特殊I/O指令

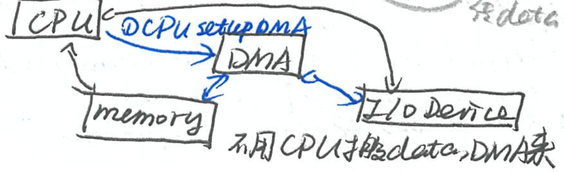
③ 端口. Status/Data reg

Polling. Interrupt-Driven I/O



总线上有若干个devices, 也有很多个master (竞争) 只有master可以操作bus. (DMA. CPU是, 但DMA优先级高)

DMA:



不用CPU搬data, DMA来

秦嘉俊 3210106182

Chap03 算术运算

signed magnitude

1's / 2's complement

↳ 2's bias 移码 (取反符号位)

同样的位码表示，移码值 = 补码值 $- 2^{n-1}$

Overflow: $OF = C_i \oplus C_{i+1}$
符号位 的 溢出去位

Multiplier: 64bit multiplicand (A) x 64 bit multiplier (B) = 128 bit product C

B放C低64bit, A单独放. 每次根据B最低位决定是否将A加到C高64位上去, 随后右移C.

Booth:		
1 0	begin, 减A	令最开始 bit₋₁ = 0, 每轮 +/-/nop A到C低位, 并右移, 直到B被完全移出.
0 0	nop	
0 1	end, 加A	
1 1	nop	

Division: 64bit dividend (A) ÷ 64bit Divisor (B) = 64bit quotient (C) … 64bit reminder (D)

用128bit Reminder, A放低64位, 每次用Reminder高64位减B, 若能减则商1, 随后左移D并加入商到D末位.
⚠ 最后一次相减后, 要把商放回D, 要多移一位, 因此最后再把Reminder高位右移回来

Float: $X = (-1)^S \times M \times 2^E$

S	exp	frac
1	8	23
1	11	52

规格: $M = 1 + frac$
非规格: exp=0 时 $M = 0 + frac$
exp=全1 时 frac=0 ⇒ ±inf; frac≠0 ⇒ "NaN"

$E = exp - bias$ 其中 $bias = 2^{n-1} - 1$ 127 / 1023

Addition: ①对齐 (小往大) ②加 ③规范化 ④Rounding

multiplication: $(S_1 \cdot 2^{e_1}) \cdot (S_2 \cdot 2^{e_2}) = (S_1 \cdot S_2) \cdot 2^{e_1+e_2}$

⚠ $e_1 + e_2 = exp_1 + exp_2 - 2 \cdot bias = \underbrace{exp'}_{} - bias$ ☆ $= exp_1 + exp_2 - bias$

guard, round bit: 运算时多保留两位
sticky bit: 后面的位若全0则为0, 否则为1

RISC-V Registers Convention

			call preserved
x0	0	const val 0	
x1(ra)	1	return addr	✓
x2(sp)	2	stack pointer	✓
x3(gp)	3	global pointer	✓
x4(tp)	4	thread pointer	✓
x5-x7(t0-t2)	5~7	temporaries	✗
x8(s0/fp)	8	saved/frame pointer	✓
x9(s1)	9	saved	✓
x10-x17(a0-a7)	10~17	argument/result	✗
x18-x27(s2-s11)	18~27	saved	✓
x28-x31(t3-t6)	28~31	temporaries	✗

Chap02 指令

	31 … 25	24 … 20	19 … 15	14 … 12	11 … 7	6 … 0
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	imm[11:0]		rs1	funct3	rd	opcode
S-type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB-type	imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
UJ-type	imm[20,10:1,11,19:12]				rd	opcode
U-type	imm[31:12]				rd	opcode

imm指写在指令里要用的数

R: add, sub, xor, or, and, sll, srl, sra, slt, sltu
>>imm[4:0] 算右移 (rs1<rs2)? 逻辑右移

要求imm<32
I: addi, xori, ori, andi, slli, srli, srai, slti, sltui . jalr rd=PC+4 PC=rs1+imm
lb. lh. lw. ld. lbu. lhu. lwu rd=M[rs1+imm]

S: sb, sh, sw, sd M[rs1+imm]=rs2[:]
写的时候: ld rd, offset(rs1)
sd rs2, offset(rs1)
Load 与 alui 的 opcode 不同

SB: beq. bne. bge. blt, bgeu. bltu
beq x1, x2, 2000 ← 放进指令里只放1000 (末位默认为0)
PC+=imm

UJ: jal

U: lui, auipc. rd=PC+(imm<<12)

stack ①push: sp←sp-8
M[sp]←data
②pop: data←M[sp]
sp←sp+8

⚠ 当调用 对于nonleaf的程序, 临时变量可能被子程序修改!

Chap04 处理器

latency 时延: 完成指令的时间
throughput 吞吐量: 1s完成的指令条数
CPI: Clock Cycle Per Instruction = 1 + stall cycles per inst
$speedup = \frac{CPI\_unpip}{CPI\_pip} \times \frac{Clock\ cycle\_un}{Clock\ Cycle\_pip} = \frac{depth(n)}{1 + stall}$
$Clock Rate = \frac{Cycles}{CPU Time}$ 时钟频率

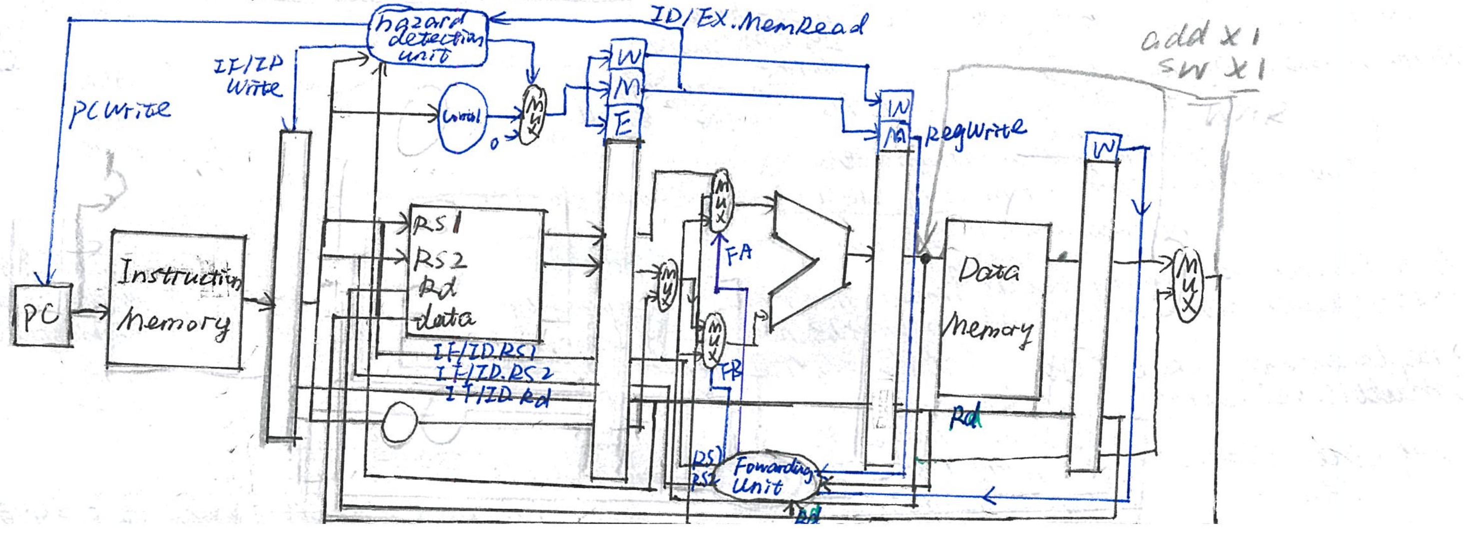
单周期CPU信号

	ALUop (2bits)	ALUSrc	M2R	RW (regWrite)	MW	MR	Branch	Jump
R	funct3+7	0	0	1	0	0	0	0
I(alui)	funct3+imm[5:11]	1	0	1	0	0	0	0
ld	+	1	1	1	0	1	0	0
sd	+	1	X	0	1	0	0	0
branch	cmp	0	X	0	0	0	1	0
jal	X	X	2	1	0	0	0	1
jalr	+	1	2	1	0	0	0	2
lui	X	X	3	1	0	0	0	0

ALU operation:
对于R-type, funct3区分不同类型, funct7用来区分 add/sub, srl/sra (funct3相同)
对于I-type, 仅有srli/srai的funct3相同,
这里 slli/srli: imm[5:11]=0, srai: imm[5:11] = 0x20.

Exception.

Level 0 User/Application; 1 Supervisor; 2 Reserved; 3 Machine
至少要有 M-mode, 其最重要的任务就是处理中断/异常.
4096 CSR (Control. Status)
相关指令: I型, 但 imm[11:0] 放的是csr编号
① mstatus 0x300. 当前控制状态
② mie/mip: 0x304/344, 中断使能/挂起
③ mtvec: 0x305. bit[1:0]: 0. Direct模式, 所有trap跳到BASE
1: Vectored模式, jmp BASE+4×cause
(BASE低2位为0, 高30位与mtvec高位对齐)
④ mepc. 0x341 返回PC
⑤ mcause 0x342 异常号
优先级: 外部中断 > 软件中断 > timer
① 进入中断: mstatus.mie = mie[i] = mip[i] = 1
② 准备: mepc ← PC(中断) / PC+4(异常), PC ← mtvec, mcause, mtval (非法指令或mem addr), mstatus
③ 返回: mret: PC ← mepc mstatus[3] ← mstatus[7]



hazard detection unit
ID/EX.MemRead
IF/ID Write
PCWrite
Control
0
W
M
E
regWrite
PC
Instruction memory
RS1
RS2
Rd
data
IF/ID.RS1
IF/ID.RS2
IF/ID.Rd
FA
FB
Data memory
Rd
RS1
RS2
Forwarding Unit
add x1
sw x1