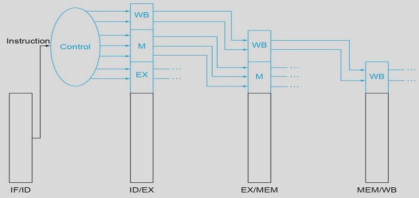
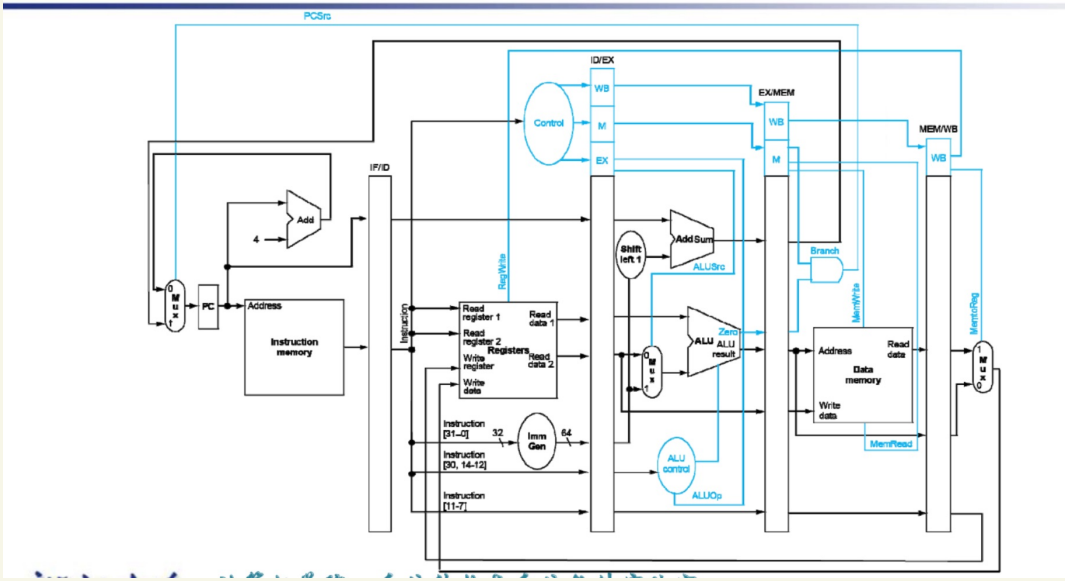
Pipelined Control

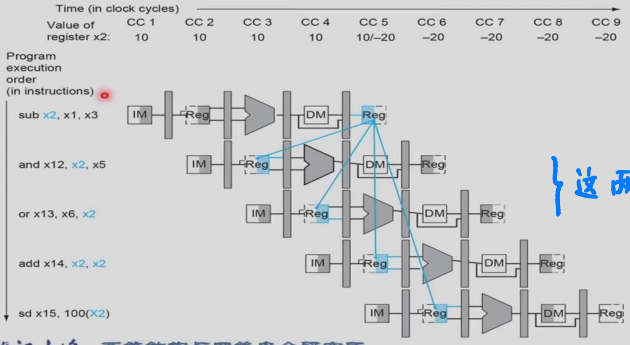
- Control signals derived from instruction
  - As in single-cycle implementation



Pipelined Control



Dependencies & Forwarding



这两条指令都需要 bypass.

Detecting the Need to Forward

- **Pass register numbers along pipeline**
  - e.g., ID/EX.RegisterRs1 = register number for Rs1 sitting in ID/EX pipeline register
- **ALU operand register numbers in EX stage are given by**
  - ID/EX.RegisterRs1, ID/EX.RegisterRs2
- **Data hazards when**
  - 1a. EX/MEM.RegisterRd = ID/EX.RegisterRs1
  - 1b. EX/MEM.RegisterRd = ID/EX.RegisterRs2

  → Fwd from EX/MEM pipeline reg
  - 2a. MEM/WB.RegisterRd = ID/EX.RegisterRs1
  - 2b. MEM/WB.RegisterRd = ID/EX.RegisterRs2

  → Fwd from MEM/WB pipeline reg

Data Hazard 发生的不同条件

Detecting the Need to Forward

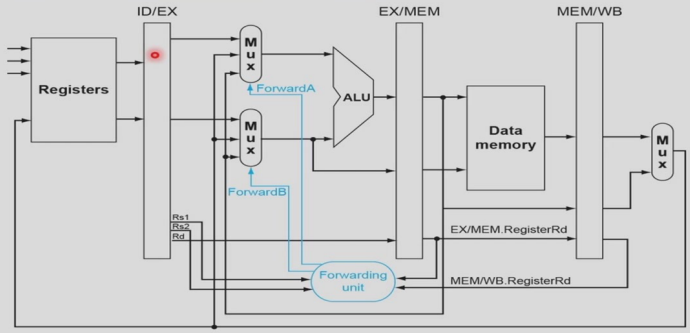
- **But only if forwarding instruction will write to a register!**
  - EX/MEM.RegWrite, MEM/WB.RegWrite
- **And only if Rd for that instruction is not x0**
  - EX/MEM.RegisterRd ≠ 0, MEM/WB.RegisterRd ≠ 0

还需外加条件：

① Write 操作的确发生

② 不是向 X0 内写

Forwarding Paths



往前2条是ALU Logic或前1条是LD

前一条指令是ALU Logic

Forwarding Conditions

Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

标明前递仅标明来源 → 无竞争 而非阶段

不同情况ALU来源不尽相同。

后续还有更新。

仍有问题：rs同时和EX/MEM的rd以及MEM/WB的rd相同，

Double Data Hazard

- Consider the sequence:

```
add x1,x1,x2   MEM/WB
add x1,x1,x3   EX/MEM
add x1,x1,x4
```

- Both hazards occur
  - Want to use the most recent
- Revise MEM hazard condition
  - Only fwd if EX hazard condition isn't true

→ 使用最新的数据！

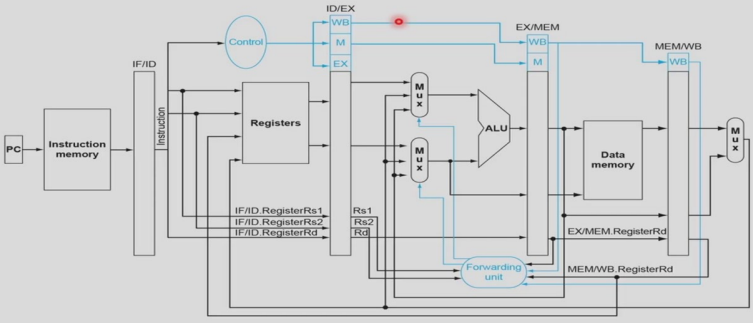
Revised Forwarding Condition

- MEM hazard
  - if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd ≠ 0)
    and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd =ID/EX.RegisterRs1))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs1)) → ForwardA = 01
  - if (MEM/WB.RegWrite
    and (MEM/WB.RegisterRd ≠ 0)
    and not(EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
    and (EX/MEM.RegisterRd = ID/EX.RegisterRs2))
    and (MEM/WB.RegisterRd = ID/EX.RegisterRs2) → ForwardB = 01

浙江大学 Zhejiang University 系统结构与网络安全研究所

最新 Path.

Datapath with Forwarding



浙江大学 Zhejiang University 系统结构与网络安全研究所

对于 Load 类操作：

Load-Use Hazard Detection

- Check when using instruction is decoded in ID stage
- ALU operand register numbers in ID stage are given by
  - IF/ID.RegisterRs1, IF/ID.RegisterRs2
- Load-use hazard when → 是 load 操作
  - ID/EX.MemRead and
    ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or
    (ID/EX.RegisterRd = IF/ID.RegisterRs2)) } 下一条指令的源操作数是 load 指令的目标操作数。
- If detected, stall and insert bubble

浙江大学 Zhejiang University 系统结构与网络安全研究所

暂停，插入 bubble

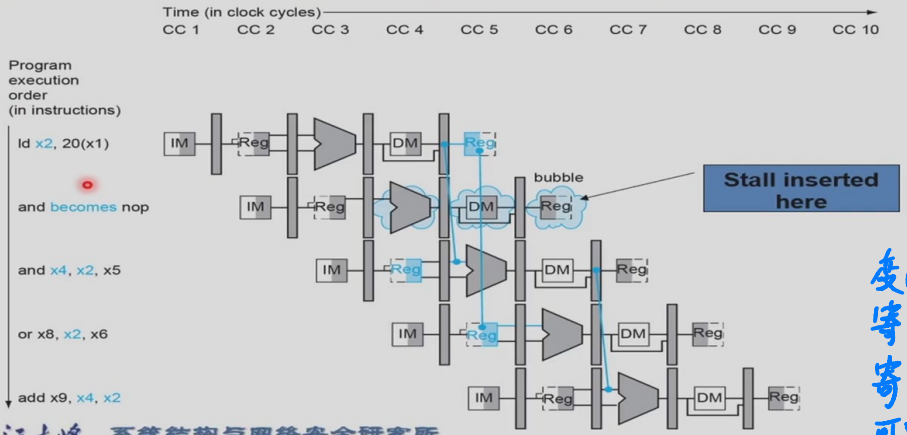
How to Stall the Pipeline

- **Force control values in ID/EX register to 0**
  - EX, MEM and WB do nop (no-operation)
- **Prevent update of PC and IF/ID register**
  - Using instruction is decoded again
  - Following instruction is fetched again
  - 1-cycle stall allows MEM to read data for ld
    - Can subsequently forward to EX stage

把所有控制信号变成0.

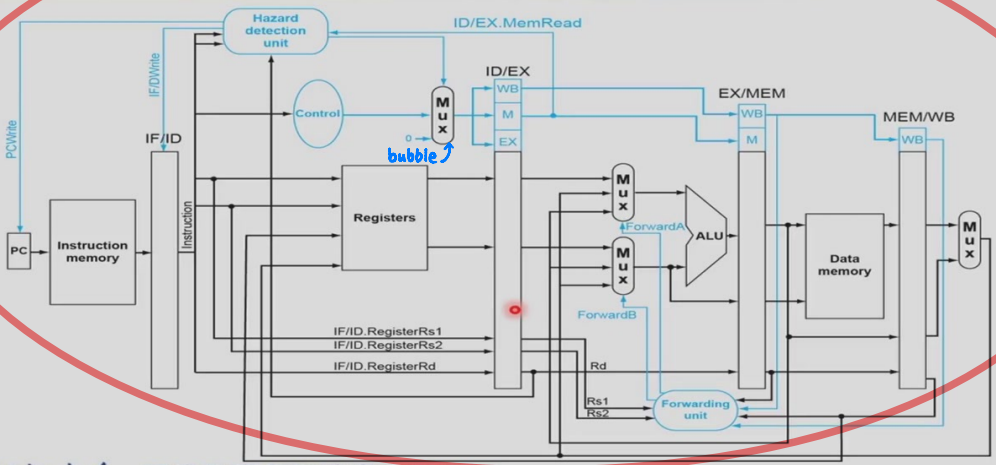
跳过指令的方法：使控制信号均为0.

Load-Use Data Hazard



变成EX/MEM寄存器与ID/EX寄存器相同，故可用bypass解决

Datapath with Hazard Detection



Reducing Branch Delay

- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
36:  sub  x10, x4, x8
40:  beq  x1,  x3, 32   // PC-relative branch
                        // to 40+16*2=72
44:  and  x12, x2, x5
48:  orr  x13, x2, x6
52:  add  x14, x4, x2
56:  sub  x15, x6, x7
     ...
72:  ld   x4, 50(x7)
```

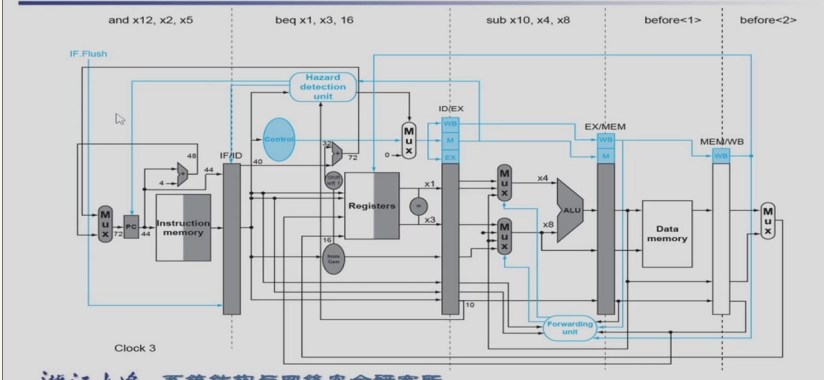
指令里是16，转移到跳32.

舍弃最低位的0去掉.

在ID阶段，就可以算出跳转地址。

可在ID阶段额外增加一个 compare operator.

Example: Branch Taken



beq 条件满足，(在ID即可判断)

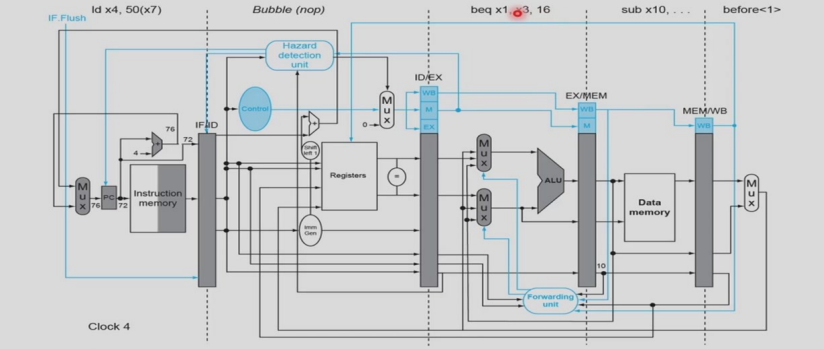
则将IF中的指令置为nop(图中的add)

等效于加一个bubble.

(将控制信号置0)

新指入IF的指令(图中为ld)是beq跳转后的新指令。

Example: Branch Taken



代价仅是1条指令。

Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction

更多级的流水线.

2 表的索引是 branch address.

针对于循环

dynamic prediction: 猜测，本次是否跳转的结果与上次相同.

以上是动态预测。

静态预测：每次预测跳转.

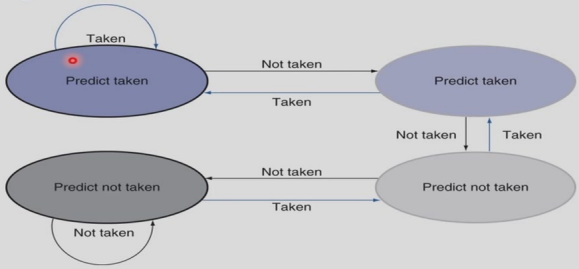
比如：for (i=0; i<100; i++) 100次beq中有99次跳转，如果用dynamic prediction很高效

单循环只会预测错一次，但双层循环就会多。

Solution: 2 Bit Prediction.

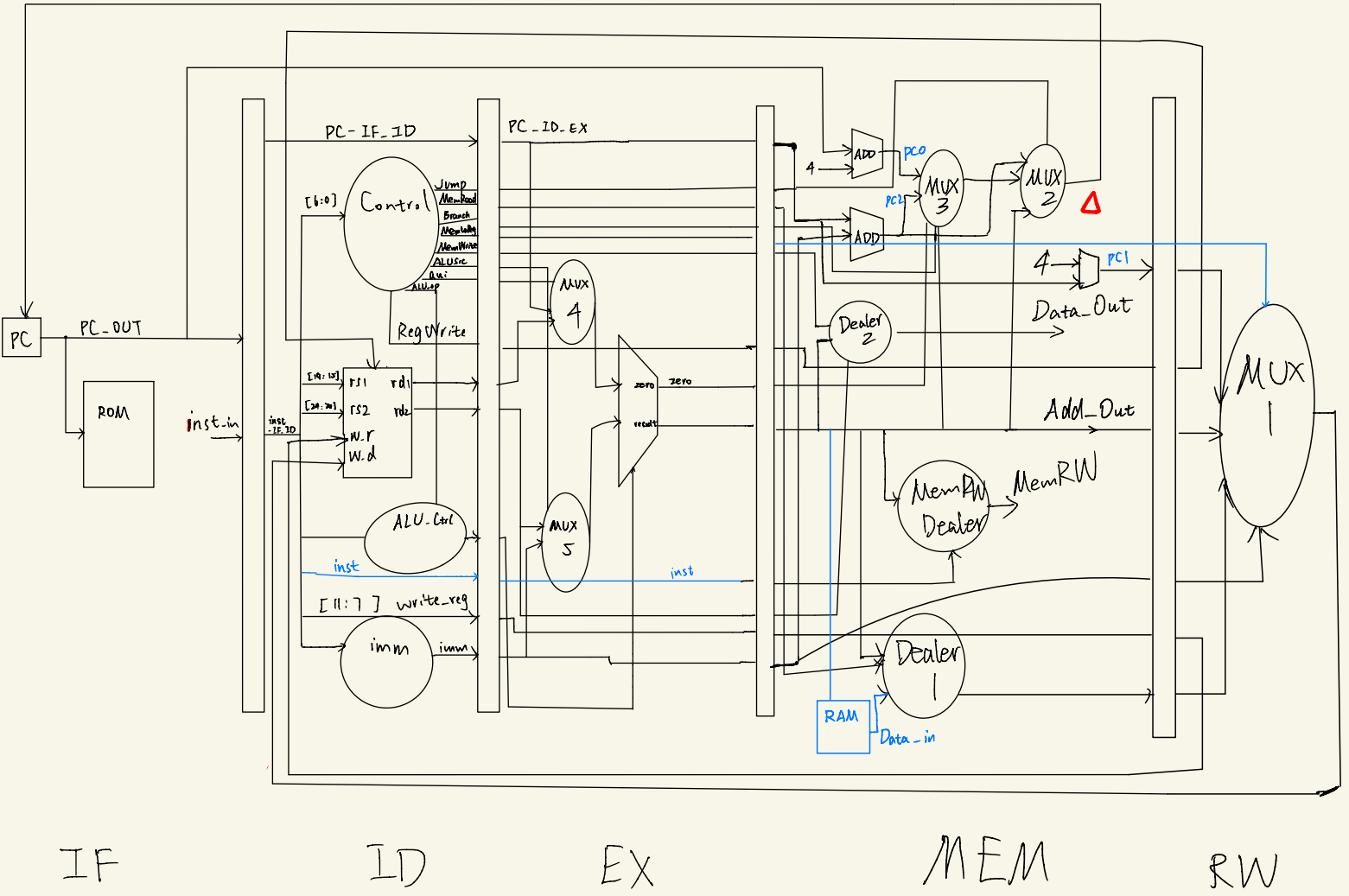
2-Bit Predictor

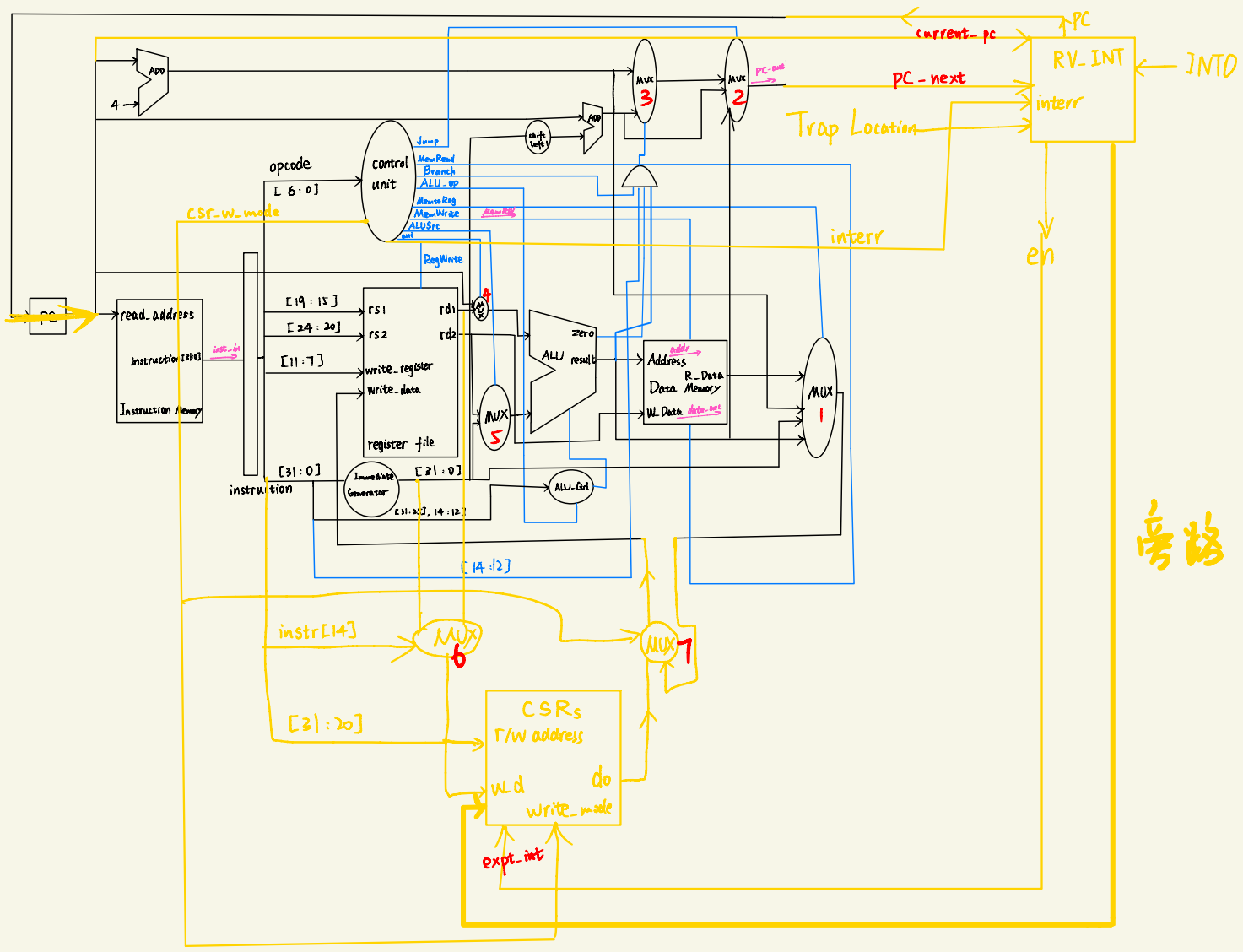
- Only change prediction on two successive mispredictions



有限状态机.

连续2次预测错误，则改变状态.


PC
PC_OUT
ROM
inst_in
inst_IF_ID
PC-IF_ID
PC_ID_EX
Control
[6:0]
Jump
MemRead
Branch
MemtoReg
MemWrite
ALUSrc
RegWrite
[19:15]
[24:20]
rs1
rs2
W.r
W.d
rd1
rd2
ALU_Ctrl
inst
[11:7]
write_reg
imm
imm
MUX 4
MUX 5
zero
zero
result
4
ADD
ADD
pc0
pc2
MUX 3
MUX 2
Dealer 2
4
pc1
Data_Out
Add_Out
MemRW Dealer
MemRW
Dealer 1
RAM
Data_in
MUX 1
inst
IF
ID
EX
MEM
RW


PC
current_pc
RV_INT
INTO
PC_next
interr
Trap Location
ADD
4
MUX
3
2
PC_out
shift left
opcode
[6:0]
control unit
Jump
MemRead
Branch
ALU_op
MemtoReg
MemWrite
ALUSrc
Ctrl
RegWrite
csr_w_mode
en
read_address
instruction [31:0]
Instruction Memory
inst_out
instruction
[19:15]
[24:20]
[11:7]
rs1
rs2
write_register
write_data
rd1
rd2
register file
[31:0]
Immediate Generator
[31:20], [14:12]
ALU
Zero
result
addr
Address
R_Data
Data Memory
W_Data
data_out
MUX
1
5
4
ALU_Ctrl
[14:12]
instr[14]
6
7
[31:20]
CSRs
r/w address
wrd
do
write_mode
expt_int

Calculating the Branch Target

- Even with predictor, still need to calculate the target address
  - 1-cycle penalty for a taken branch
- Branch target buffer
  - Cache of target addresses
  - Indexed by PC when instruction fetched
    - If hit and instruction is branch predicted taken, can fetch target immediately

甚至可以在 table 中储存跳转地址.

异常与中断：

Exception
- 狭义：unexpected event
- 广义：所有改变 CPU 执行顺序的 event.

RISC-V Privileged

All hardware implementations must provide M-mode

- RISC-V Privileged Architecture
  - The machine level has the highest privileges
    - and is the only mandatory privilege level for a RISC-V hardware platform.
    - Code run in machine-mode (M-mode) is usually inherently trusted, as it has low-level access to the machine implementation.
    - M-mode can be used to manage secure execution environments on RISC-V.
  - User-mode (U-mode) and supervisor-mode (S-mode) are intended for conventional application and operating system usage respectively.

Level	Encoding	Name	Abbreviation	
0	00	User/Application	U	用户模式
1	01	Supervisor	S	监督模式
2	10	Reserved		保留
3	11	Machine	M	机器模式

3 种不同模式.

RISC-V Privilege Modes Usage

- Each privilege level has
  - a core set of privileged ISA extensions → 每种模式都有自己的扩展特权指令集
    - with optional extensions and variants.
- Combinations of privilege modes
  - Implementations might provide anywhere from 1 to 3 privilege modes
    - trading off reduced isolation for lower implementation cost

Number of levels	Supported Modes	Intended Usage
1	M	Simple embedded systems
2	M, U	Secure embedded systems
3	M, S, U	Systems running Unix-like operating systems

  - For example, machine-mode supports an optional standard extension for memory protection.

但M是权限最高的

机器模式是最基础的.

所有硬件实现必须支持 M mode.

RISC-V has 4096 Control and Status Registers (CSRs)

↳仅能被特权指令访问.(包括原子性操作)

I类:

CSR Instruction

- All CSR instructions atomically read-modify-write a single CSR
  - whose CSR specifier is encoded in the 12-bit csr field of the instruction held in bits 31–20
    - The immediate forms use a 5-bit zero-extended immediate encoded in the rs1 field.

	31 20	19 15	14 12	11 7	6 0
	csr	rs1	funct3	rd	opcode
	12bits	5bits	3bits	5bits	7bits
I: CSRRW	csr	rs1	001	rd	1110011
I: CSRRS	csr	rs1	010	rd	1110011
I: CSRRC	csr	rs1	011	rd	1110011
I: CSRRWI	csr	zimm	101	rd	1110011
I: CSRRSI	csr	zimm	110	rd	1110011
I: CSRRCI	csr	zimm	111	rd	1110011

浙江大学 Zhejiang University 系统结构与网络安全研究所

CSRRW：把 csr 写入 rd，
把 rs1 读入 csr.

set

CSRRS：把 csr 写入 rd，
把 rs1 与 csr 按位或，读入 csr.

clear

CSRRC：把 csr 写入 rd，
把 rs1 与 csr 按位与，读入 csr.

Interrupts Instruction

- Exception return
  - MRET
    - PC <= MEPC; (MStatus寄存器有变化)

	31 25	24 20	19 15	14 12	11 7	6 0
R: MRET	0011000	00010	00000	000	00000	1110011
R: SRET	0001000	00010	00000	000	00000	1110011
R: wfi	0001000	00101	00000	000	00000	1110011

- Environment call
  - ecall
    - MEPC = ecall指令本身的PC值
- Breakpoint
  - ebreak
    - MEPC = ebreak指令本身的PC值

	31 25	24 20	19 15	14 12	11 7	6 0
I: ecall	0000000	00000	00000	000	00000	1110011
I: ebreak	0000000	00001	00000	000	00000	1110011

浙江大学 Zhejiang University 系统结构与网络安全研究所

MRET：从机器模式返回
SRET：从监督模式返回
wfi：等待中断

ecall：进入机器模式

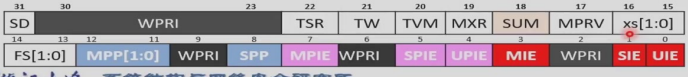
ebreak

Exception & Interrupt related registers

CSR	Privilege	Abbr.	Name	Description
0x300	MRW	mstatus	Machine STATUS register 机器模式状态寄存器	MIE、MPIE域标记中断全局使能
0x304	MRW	mie	Machine Interrupt Enable register 机器模式中断使能寄存器	控制不同类型中断的局部使能
0x305	MRW	mtvec	Machine trap-handler base address 机器模式异常入口基地址寄存器	进入异常服务程序基地址
0x341	MRW	mepc	Machine exception program counter 机器模式异常PC寄存器	异常断点PC地址
0x342	MRW	mcause	Machine trap cause register 机器模式原因寄存器	处理器异常原因
0x343	MRW	mtval	Machine Trap Value register 机器模式异常值寄存器	处理器异常值地址或指令
0x344	MRW	mip	Machine interrupt pending 机器模式中断挂起寄存器	处理器中断等待处理

Interrupt Registers: mstatus(0x300)

- Machine STATUS register
  - These bits are primarily used to guarantee atomicity with respect to interrupt handlers in the current privilege mode.
  - Global interrupt enable bits: MIE, SIE, and UIE 管控中断的使能信号
    - are provided for each privilege mode.
  - xPIE holds the value of the interrupt enable bit active prior to the trap 存放进入异常前各IE变量的值
  - xPP holds the previous privilege mode 存放中断前的 mode.



Interrupt Registers: mstatus(0x300)

- Machine STATUS register
  - These bits are primarily used to guarantee atomicity with respect to interrupt handlers in the current privilege mode.

bit	Name	Attributes	Description
0	UIE	RW	User interrupt enable
1	SIE	RW	Supervisor interrupt enable
3	MIE	RW	Machine interrupt enable
4	UPIE	RW	previous user-level interrupts enable
5	SPIE	RW	Previous Supervisor interrupt-enable
7	MPIE	RW	Previous Machine interrupt enable
8	SPP	RW	Supervisor Previous Privilege mode
12:11	MPP	RW	Machine Previous Privilege mode
.....			
31:23,9,6,2	WPRI		Reserved

一般是支能从上往下:

USER_MODE
SUPERVISOR_MODE
MACHINE_MODE

没有 UPP!

Interrupt Registers: mie/mip (0x304/344)

- **Machine Interrupt Enable register**
  - MIE register controls whether it can respond to interrupts, corresponding different modes
    - MEIE、SEIE and UEIE enable external interrupt
    - MSIE、SSIE & USIE enable software interrupts
    - MTIE、STIE and UTIE enable timer interrupts
- **Machine interrupt-pending register**
  - The mip register is an MXLEN-bit read/write register containing information on pending interrupts

	31–12	11	10	9	8	7	6	5	4	3	2	1	0
mie	WPRI	MEIE	WPRI	SEIE	UEIE	MTIE	WPRI	STIE	UTIE	MSIE	WPRI	SSIE	USIE
mip	WPRI	MEIP	WPRI	SEIP	UEIP	MTIP	WPRI	STIP	UTIP	MSIP	WPRI	SSIP	USIP

M, S and U correspond to M mode, S mode and U mode interrupt respectively

异常：均用查询模式
中断：向量模式。

Interrupt Registers: mtvec (0x305)

- **Machine Trap-Vector Base-Address Register**
  - The mtvec register holds trap vector configuration, consisting of a vector base address (BASE) and a vector mode (MODE)

31	2	1	0
BASE[MXLEN-1:2] (WARL)		MODE (WARL)	

  - The value in the BASE field must always be aligned on a 4-byte boundary, and the MODE setting may impose additional alignment constraints on the value in the BASE field.

MODE Value	Name	Description
0	Direct(查询)	All exceptions set pc to BASE
1	Vectored(向量)	Asynchronous interrupts set pc to BASE+4×cause
≥2	--	Reserved

BASE+4×Exception

查询模式：
直接跳到 Base Address.
查询原因并处理.

向量模式：PC 跳到 Base+4×Exception

Interrupt Registers: mepc (0x341)

- **Machine Exception Program Counter**
  - mepc is a WARL register that must be able to hold all valid physical and virtual addresses.
    - When a trap is taken into M-mode, mepc is written with the address of the instruction that was interrupted or exception.

31	2	1	0
mepc		0	0

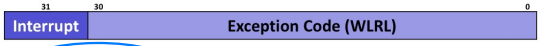
  - The low bit of mepc (mepc[0]) is always zero.
    - On implementations that support only IALIGN=32, the two low bits (mepc[1:0]) are always zero.
  - Exception: `mepc ← pc`
  - Interrupted: `mepc ← pc + 4`

MEPC：
储存当前
(异常的 PC) 或
(中断的 PC+4)

WARL：Write any value; Read only legal value.
可以用 PC 做加法后写入

Interrupt Registers: mcause (0x342)

- **Machine Cause Register (mcause)**
  - When a trap is taken mcause register is written with a code indicating the event that caused the Exception/Interruption.



  - Exception Code与mtvec的向量模式相对应
    - 在异步中断时，不同的模式会跳转到不同的入口
  - The Exception Code is a WLRL
    - Write/Read Only Legal Values
    - 如果写入值不合法可以引发非法指令异常

只有特定合法值可以被正常写入

interrupts　　exceptions

INT	E Code	Description	INT	E Code	Description
1	0	User software interrupt	0	0	Instruction address misaligned
1	1	Supervisor software interrupt	0	1	Instruction access fault
1	2	Reserved for future standard use	**0**	**2**	**Illegal instruction**
1	**3**	**Machine software interrupt**	0	3	Breakpoint
1	4	User timer interrupt	0	4	Load address misaligned
1	5	Supervisor timer interrupt	0	5	Load access fault
1	6	Reserved for future standard use	0	6	Store/AMO address misaligned
1	7	Machine timer interrupt	0	7	Store/AMO access fault
1	8	User external interrupt	0	8	Environment call from U-mode
1	9	Supervisor external interrupt	0	9	Environment call from S-mode
1	10	Reserved for future standard use	0	10	Reserved
1	**11**	**Machine external interrupt**	0	11	**Environment call from M-mode**
1	12-15	Reserved for future standard use	0	12	Instruction page fault
1	≥16	Reserved for platform use	0	13	Load page fault
0	16-23	Reserved for future standard use	0	15	Store/AMO page fault
0	32-47	Reserved for future standard use	0	24-31	Reserved for custom use
0	14/≥64	Reserved for future standard use	0	48-63	Reserved for custom use

实际Table中储存的还是jal指令，跳到其它地方处理中断/异常.

RISV-V Interrupt priority

External interrupt > Software interrupt > Timer interrupt

- **Synchronous exception priority**

Priority	Exception Code	Description
Highest	3	Instruction address breakpoint
	12	Instruction page fault
	1	Instruction access fault
	2	Illegal instruction
	0	Instruction address misaligned
	8，9，11	Environment call
	3	Environment break
	3	Load/Store/AMO address breakpoint
	6	Store/AMO address misaligned
	4	Load address misaligned
	15	Store/AMO page fault
	13	Load page fault
	7	Store/AMO access fault
Lowest	5	Load access fault

中断和异常均有优先级.

RISC-V中断处理--进入异常

- RISC-V处理器检测到异常，开始进行异常处理：
  - 停止执行当前的程序流，转而从CSR寄存器mtvec定义的PC地址开始执行；
  - 更新机器模式异常原因寄存器：mcause
  - 更新机器模式中断使能寄存器：mie
  - 更新机器模式异常PC寄存器：mepc
  - 更新机器模式状态寄存器：mstatus
  - 更新机器模式异常值寄存器：mtval

(以上更新均有硬件完成)

浙江大学 Zhejiang University 系统结构与网络安全研究所

RISC-V中断结构--退出异常

- 异常程序处理完成后，需要从异常服务程序中退出，并返回主程序
  - RISCV中定义了一组退出指令MRET，SRET，和URET
  - 机器模式对应MRET。
- 机器模式下退出异常(MRET)
  - 程序流转而从csr寄存器mepc定义的pc地址开始执行
  - 同时硬件更新csr寄存器机器模式状态寄存器mstatus
    - 寄存器MIE域被更新为当前MPIE的值：mie ← mpie
    - MPIE 域的值则更新为1：MPIE ← 1

不同处理器，流水线处理异常的方式不同.

例如：可以把目前流水线上的指令全部停止，并在SEPC中记下PC.

多个 Exception 同时发生：

Simple Aproach：只处理最早发生 Exception 的阶段，
其它都直接 Flush（停止）→ Precise Exception

Multiple Exceptions

- Pipelining overlaps multiple instructions
  - Could have multiple exceptions at once
- Simple approach: deal with exception from earliest instruction
  - Flush subsequent instructions
  - "Precise" exceptions
- In complex pipelines
  - Multiple instructions issued per cycle
  - Out-of-order completion
  - Maintaining precise exceptions is difficult!

复杂流水线，很难使用 precise exception

Imprecise Exceptions

- Just stop pipeline and save state
  - Including exception cause(s)
- Let the handler work out
  - Which instruction(s) had exceptions
  - Which to complete or flush
    - May require "manual" completion
- Simplifies hardware, but more complex handler software
- Not feasible for complex multiple-issue out-of-order pipelines

保留现状，交给异常处理程序。→硬件简单，但异常程序复杂.

Simplify Interrupt Design

- 简化中断设计
  - 采用ARM中断向量(不兼容RISC-V)
    - 实现非法指令异常和外中断
    - 设计EPC
  - 兼容RISC-V*
    - 仅M-Mode中断寄存器(MCause、Mstatus、MIE、MIP、MEPC和MTVEC)
    - 设计mret、CSRRW (csrw rd, csr, rs1)和ecall指令
- ARM中断向量表

向量地址	ARM异常名称	ARM系统工作模式	本课程定义
0x0000000	复位	超级用户Svc	复位(M-MODE)
0x0000004	未定义指令终止	未定义指令终止Und	非法指令异常
0x0000008	软中断（SWI）	超级用户Svc	ECALL
0x000000c	Prefetch abort	指令预取终止Abt	Int外部中断（硬件）
0x0000010	Data abort	数据访问终止Abt	Reserved自定义
0x0000014	Reserved	Reserved	Reserved自定义
0x0000018	IRQ	外部中断模式IRQ	Reserved自定义
0x000001C	FIQ	快速中断模式FIQ	Reserved自定义

下面都需要了解，高级处理器

4.10 Instruction-Level Parallelism (ILP)

- **Pipelining: executing multiple instructions in parallel** 并行操作
- **To increase ILP**
  - Deeper pipeline 更深的流水线
    - Less work per stage ⇒ shorter clock cycle
  - Multiple issue
    - Replicate pipeline stages ⇒ multiple pipelines
    - Start multiple instructions per clock cycle
    - CPI < 1, so use Instructions Per Cycle (IPC)
    - E.g., 4GHz 4-way multiple-issue
      - 16 BIPS, peak CPI = 0.25, peak IPC = 4
    - But dependencies reduce this in practice

Multiple Issue 分为静态，动态两种

- **Static multiple issue**
  - Compiler groups instructions to be issued together
  - Packages them into "issue slots"
  - Compiler detects and avoids hazards

Compiler 处理

- **Dynamic multiple issue**
  - CPU examines instruction stream and chooses instructions to issue each cycle
  - Compiler can help by reordering instructions
  - CPU resolves hazards using advanced techniques at runtime

CPU 处理

Static Multiple Issue

- **Compiler groups instructions into "issue packets"**
  - Group of instructions that can be issued on a single cycle
  - Determined by pipeline resources required
- **Think of an issue packet as a very long instruction**
  - Specifies multiple concurrent operations
  - ⇒ Very Long Instruction Word (VLIW)

Speculation 投机：预判 → 很难！

- **"Guess" what to do with an instruction**
  - Start operation as soon as possible
  - Check whether guess was right
    - If so, complete the operation
    - If not, roll-back and do the right thing
- **Common to static and dynamic multiple issue**
- **Examples**
  - Speculate on branch outcome
    - Roll back if path taken is different
  - Speculate on load
    - Roll back if location is updated

Scheduling Static Multiple Issue

- **Compiler must remove some/all hazards**
  - Reorder instructions into issue packets
  - No dependencies with a packet
  - Possibly some dependencies between packets
    - Varies between ISAs; compiler must know!
  - Pad with nop if necessary

Compiler/Hardware Speculation

- **Compiler can reorder instructions**
  - e.g., move load before branch
  - Can include "fix-up" instructions to recover from incorrect guess
- **Hardware can look ahead for instructions to execute**
  - Buffer results until it determines they are actually needed
  - Flush buffers on incorrect speculation

RISC-V with Static Dual Issue

- **Two-issue packets**
  - One ALU/branch instruction
  - One load/store instruction
  - 64-bit aligned
    - ALU/branch, then load/store
    - Pad an unused instruction with nop

例如：Branch 用不到 Memory，因此可以与 load 一起。代价是增加一个额外的 ALU。⇒ 没有使用两倍的硬件，但能得到两倍的结果。

Address	Instruction type	Pipeline Stages						
n	ALU/branch	IF	ID	EX	MEM	WB		
n + 4	Load/store	IF	ID	EX	MEM	WB		
n + 8	ALU/branch		IF	ID	EX	MEM	WB	
n + 12	Load/store		IF	ID	EX	MEM	WB	
n + 16	ALU/branch			IF	ID	EX	MEM	WB
n + 20	Load/store			IF	ID	EX	MEM	WB

Speculation and Exceptions

- **What if exception occurs on a speculatively executed instruction?**
  - e.g., speculative load before null-pointer check
- **Static speculation**
  - Can add ISA support for deferring exceptions
- **Dynamic speculation**
  - Can buffer exceptions until instruction completion (which may not occur)

Loop Unrolling

- **Replicate loop body to expose more parallelism**
  - Reduces loop-control overhead
- **Use different registers per replication**
  - Called "register renaming"
  - Avoid loop-carried "anti-dependencies"
    - Store followed by a load of the same register
      - Aka "name dependence"
        - Reuse of a register name

Compiler 做

牺牲程序尺寸，加快程序执行

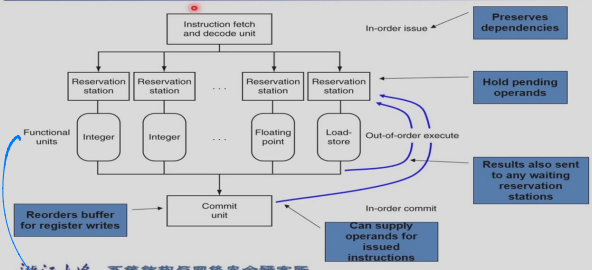
增加每一步 Loop 的步长，减少 loop 的次数，减少循环变量的开销

Dynamic Multiple Issue

超标量处理器

- "Superscalar" processors
- CPU decides whether to issue 0, 1, 2, … each cycle
  - Avoiding structural and data hazards
- Avoids the need for compiler scheduling
  - Though it may still help
  - Code semantics ensured by the CPU

Dynamically Scheduled CPU



不同作用的ALU：整数、浮点，L/S 操作

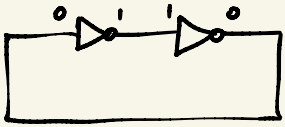
Chapter 5: Memory Hierachy 内存

SRAM：（快而贵）

数据被保存在一对反向器上。

⇒ 稳定！且读写很快。

使用6个摩斯管

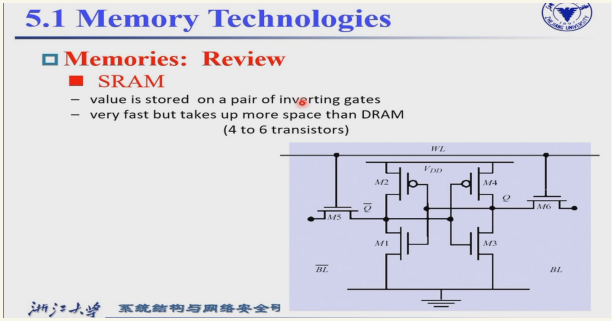


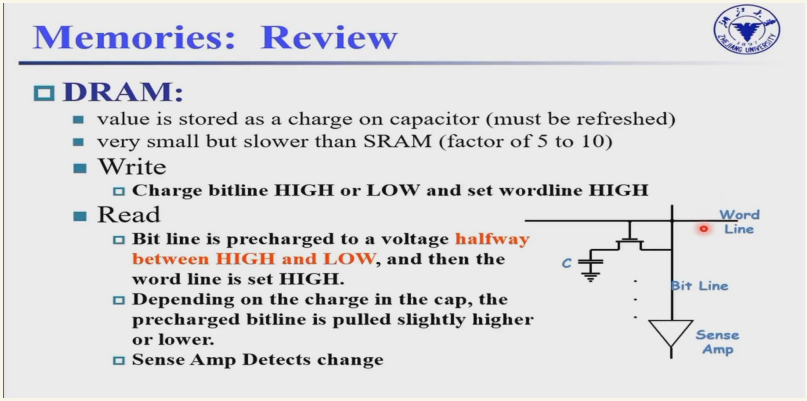
抗干扰性很好

DRAM：（用于存储较大规模的数据）

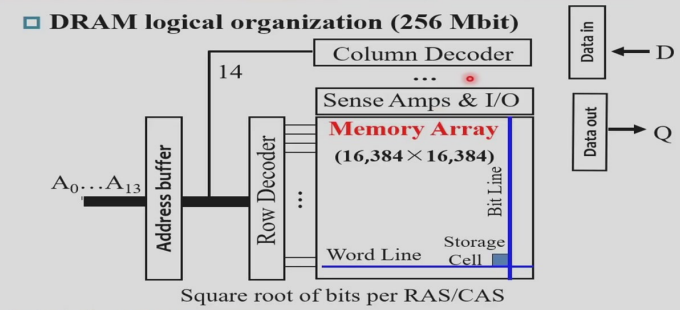
数据被保存在电容上（以电荷的形式）。更小，但速度慢。

↘ SRAM的$\frac{1}{5}$到$\frac{1}{10}$





根据 bit_line 和 word_line 的充放电，来决定 读写.



DRAM 会分 行地址、列地址 ⇒ 可以 有更大的存储空间.

Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs

burst_mode：
读取在同一行中的数据会更快。
（理由：DRAM 在"读"时需要给 word_line 充电，如果读同一行的数据，后面几次就不用再给 word_line 充电了。）

DDR：在 clock 上升沿、下降沿都要读/写数据 ⇒ 速度快一倍

QDR：把读、写端口分开，可以实现同时读写.

Flash：速度是硬盘的千百倍，Flash 且掉电不丢失.

DRAM、SRAM 掉电易失，但硬盘掉电无影响.

Disk：硬盘（磁盘）

Problems in memory designing

- In fact

Memory technology	Typical access time	Cost per GByte (2012)
SRAM	0.5-2.5ns	$2000-$5,000
DRAM	50-70ns	$20-$75
Magnetic disk	5,000,000-20,000,000ns	$0.2-$2

- Users want large and fast memories!

温度越高，越易老化

5.2 Memory Hierarchy Introduction

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

时间局部性：最近被访问的数据大概率还将被访问。

空间局部性：最新被访问内容的附近相关内容将被访问的概率大。

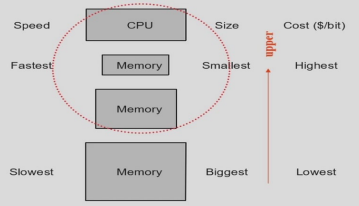
Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory → cashe
  - Cache memory attached to CPU

不断把更常访问的程序放入成本高、容量小、速度快的一部分。

Memory Hierarchy Levels

- Build a memory hierarchy



SRAM (Cache)

DRAM

Disk

miss penalty：从 low level 中的数据搬移 up level 的时间。（不是覆盖，还包括回写！）

Block (aka line)：不同 Memory 间传数据的最小单位。

↘ 可能是 multiple words.

hit_time 包含两部分：1. 判断是否命中，2. 访问数据的时间

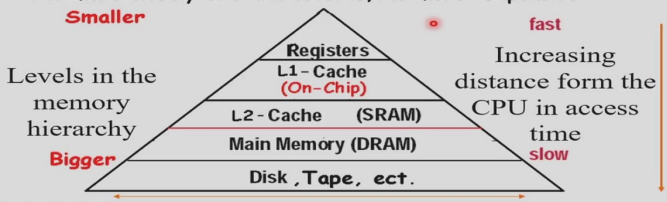
Hit：需要访问的数据可以直接从本级内存中得到，无需从下一级内存中取得。（hit ratio：指令一般在100%，数据四路低：80%-90%）

Miss：需要从 lower level 拷贝 Block）

Exploiting Memory Hierarchy

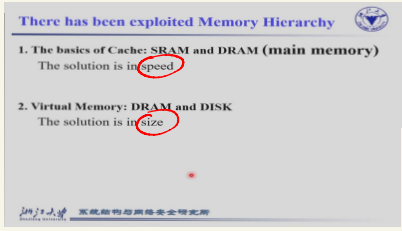
The method

- Hierarchies bases on memories of different speeds and size
- The more closely CPU the level is, the faster the one is.
- The more closely CPU the level is, the smaller the one is.
- The more closely CPU the level is, the more expensive



浙江大学 系统结构与网络安全研究所

L2_Cache 不一定有，且不在 CPU 内部.



Basic Concepts of Cache

Lower Level 中的每一个 data，在 cache 中有唯一确定的存储位置

↳ Low level 中的多个数据 share Cache 中的同一块位置.

2个问题 { 1. 如何知道某个数据，是否在 Cache 中？
2. 如何在 cache 中找到数据？

先以 Cache 一个 word 为 Block 为例子.

Directed-mapping Algorithm：取模

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits (最后几位相同)
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

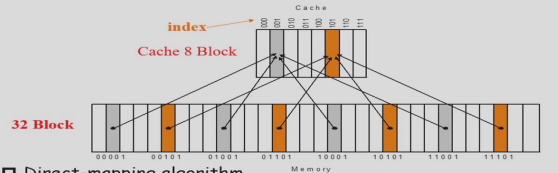
浙江大学 系统结构与网络安全研究所

把 Low Level 的 Block Address 也存在 Cache 中！⇒ 称为 Tag.

另加1位：Valid Bit.

Direct Mapped Cache

- Where can a block be placed in the upper level?



- Direct-mapping algorithm.
  memory address is modulo the number of blocks in the cache
- Fortunately, while the cache has 2^n blocks, the corresponding index is equal to the lowest n bits of memory block address. Here n=3. Let's check

Accessing a cache---how do we find it?

- Memory block address is larger than cache block address

TAG	Index	Byte offset

Block address MOD Numbers of Cache Block

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Valid bit

a. The initial state of the cache after power-on

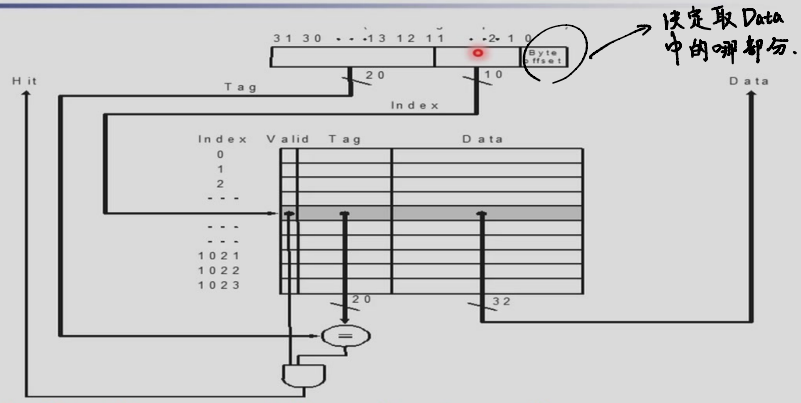
Offset：确定一个 Block 中的哪个 byte。若 1 个 Block 中有 16 Bytes，则需 4 位 Byte Offset。($2^4=16$)

Index：存入 cache 中哪一个 Block 的编号。

Tag：High-Order Bits

Memory Address / Cache 中的 Block 数 = Tag ------ Index

Direct Mapped Cache construction



What kind of locality are we taking advantage of?

temporal

Bits in Cache

Example

- How many total bits are required for a direct-mapped cache 16KB of data and 4-word blocks, assuming a 32-bit address?

Answer

- 16KB=4KWord=2^{12} words
- One block=4 words = 2^2 words
- Number of blocks (index bit) = $2^{12} \div 2^2 = 2^{10}$ blocks
- Data bits of block =4×32=128 bits
- Tag bits = address - index-block size =32-10-2-2 =18 bits
- Valid bit = 1 bit
- Total Cache size = $2^{10} \times (128+18+1) = 2^{10} \times 147 = 147$ Kbits = 18.4KB
- It is about 1.15 times as many as needed just for the data

浙江大学 Zhejiang University 系统结构与网络安全研究所

1. Cache 中 Block 数 = $\frac{16\ KB}{4 \times \frac{32}{8}} = 1K = 2^{10}$

⇒ Index 有 10 bits

2. Offset：一个 Block 中有 $4 \times \frac{32}{8} = 16$ Byte.

⇒ offset 有 4 bits

3. 每个 Tag 有 32 − 10 − 4 = 18 bits

4. 每个 Block 有 4×32 = 128 bits.

5. 一共有 $2^{10} \times (128 + 18 + 1) = 147$ K bits

↑ Data ↑ Tag ↑ Valid

Mapping an Address to Multiword Cache Block

Example

- Consider a cache with 64 blocks and a block size of 16 bytes.
- What block number does byte address 1200 map to?

Answer

(Block address) modulo (Number of cache blocks)

Where the address of the block is First: get BLOCK Address

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor = \left\lfloor \frac{1200}{16} \right\rfloor = 75$$

Notice!!! 75 modulo 64 =11 Then: get INDEX

$$\left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Byte per block} \longleftrightarrow \left\lfloor \frac{\text{Byte address}}{\text{Bytes per block}} \right\rfloor \times \text{Byte per block} + (\text{Byte per block} - 1)$$

Here: 1200 ⟷ 1215

浙江大学 Zhejiang University 系统结构与网络安全研究所

1. Block Address = $\lfloor 1200/16 \rfloor = 75$

2. Map to 75 mod 64 = 11

Miss 分两种：① Instruction Cache Miss ② Data Cache Miss

Handling Cache reads hit and Misses

- **Read hits**
  - this is what we want!
- **Read misses—two kinds of misses**
  - instruction cache miss
  - data cache miss
- **let's see main steps taken on an instruction cache miss**
  - **Stall the CPU**, fetch block from memory, deliver to cache, restart CPU read

1. Send the original PC value (current PC-4) to the memory.
2. Instruct main memory to perform a read and wait for the memory to complete its access.
3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address into the tag field, and turning the valid bit on.
4. Restart the instruction execution at the first step, which will refetch the instruction again, this time finding it in the cache.

浙江大学 Zhejiang University 系统结构与网络安全研究所

Read：

暂停 CPU，从 Memory 中取数据至 cache，重启 CPU

Handling Cache Writes hit and Misses

- **Write hits:** Difference Strategy
  - write-back: Cause Inconsistent
    - Wrote the data into only the data cache
    - Strategy ---- write back data from the cache to memory later
      Fast!
  - write-through: Ensuring Consistent
    - Write the data into both the memory the cache
    - Strategy ---- writes always update both the cache and the memory
    - Slower!----write buffer
- **Write misses:**
  - read the entire block into the cache, then write the word

需要多加一位，记录是否被写回Memory.
↑ 适用于多次写.

两种策略：

1. 只往 cache 里写。不往 Memory 中写。Write Back
   → 快！后续再往Memory中写.
2. 往 cache 和 Memory 中写.
   → Consistent！

* Cache 只能加速 Read，不能加速 Write.

Deep concept in Cache

Four Questions for Memory Hierarchy Designers

Caching is a general concept used in processors, operating systems, file systems, and applications.

There are Four Questions for Memory Hierarchy Designers

- **Q1: Where can a block be placed in the upper level?**
  (Block placement)
  - Fully Associative, Set Associative, Direct Mapped
- **Q2: How is a block found if it is in the upper level?**
  (Block identification)
  - Tag/Block
- **Q3: Which block should be replaced on a miss?**
  (Block replacement)
  - Random, LRU,FIFO
- **Q4: What happens on a write?**
  (Write strategy)
  - Write Back or Write Through (with Write Buffer)

Q1: Block Placement

- **Direct mapped**
  - Block can only go in one place in the cache
    Usually **address MOD Number of blocks in cache**
- **Fully associative**
  Block can go anywhere in cache.
- **Set associative** → 综合前2种
  - Block can go in one of a set of places in the cache.
  - A set is a group of blocks in the cache.
    Block address MOD Number of *sets* in the cache
  - If sets have n blocks, the cache is said to be n-way set associative.

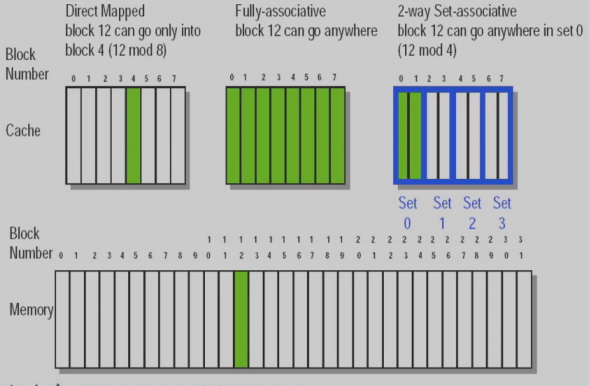
•Note that direct mapped is the same as 1-way set associative, and fully associative is m-way set-associative (for a cache with m blocks).

1路组关联

↑ 哪里空去哪里.

n个 Block 为 1 set.
按 Direct Map 的方式选中 set，进入set后从 Fully associative 选 Block.

Figure 8-32 Block Placement



具体举例.

Q2: Block Identification

- **Tag**
  - Every block has an **address tag** that stores the main memory address of the data stored in the block.
  - When checking the cache, the processor will **compare** the requested **memory address to the cache tag** -- if the two are equal, then there is a cache hit and the data is present in the cache
- **Valid bit**
  - Often, each cache block also has a **valid bit** that tells if the contents of the cache block are valid

Tag: 标志 Cache 中的数据在 Memory 中的地址.

Valid Bit: Cache 中的数据是否有效.

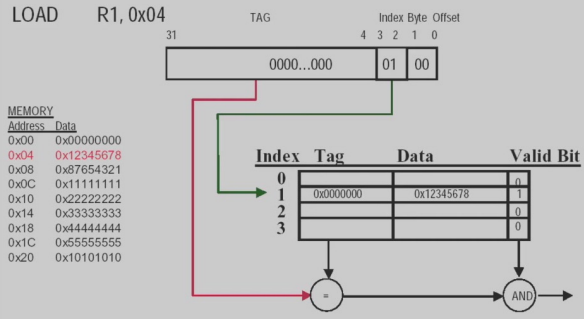
The Format of the Physical Address

- **The Index field selects**
  - The set, in case of a set-associative cache
  - The block, in case of a direct-mapped cache
  - Has as many bits as log2(#sets) for set-associative caches, or log2(#blocks) for direct-mapped caches
- **The Byte Offset field selects**
  - The byte within the block
  - Has as many bits as log2(size of block)
- **The Tag is used to find the matching block within a set or in the cache**
  - Has as many bits as
    Address_size - Index_size - Byte_Offset_Size

Index: 哪个 Block / set ?

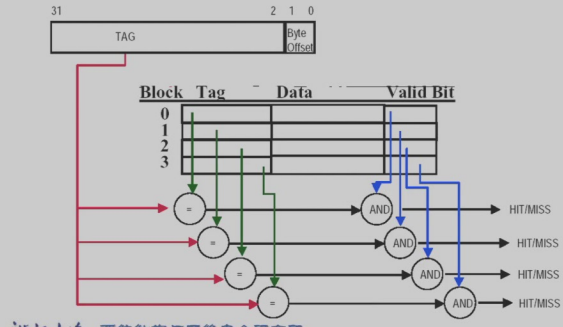
Byte Offset: Block 中的哪个 Byte?

Direct-mapped Cache Example (1-word Blocks)



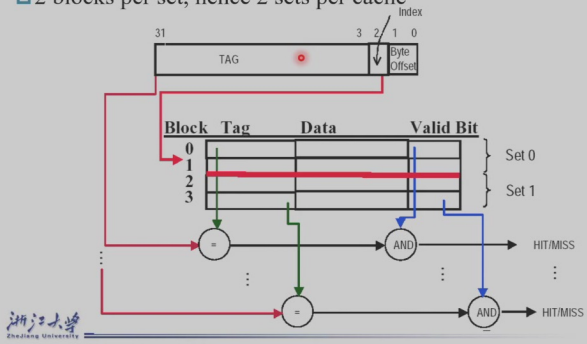
Fully-Associative Cache example (1-word Blocks)

- Assume cache has 4 blocks



2-Way Set-Associative Cache

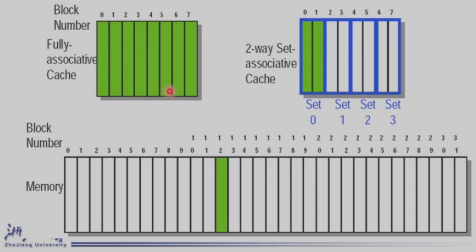
- Assume cache has 4 blocks and each block is 1 word
- 2 blocks per set, hence 2 sets per cache



这张图少了2条线.
Index=0，需要和set中的两个Block的Tag分别比较，Index=1同理.

Q3: Block Replacement

- In a direct-mapped cache, there is **only one block** that can be replaced
- In set-associative and fully-associative caches, there are N blocks (where N is the degree of associativity



具体选择哪个Block?

Strategy of block Replacement

- Several different replacement policies can be used
  - **Random replacement** - *randomly pick any block*
    - Easy to implement in hardware, just requires a random number generator
    - Spreads allocation uniformly across cache
    - May evict a block that is about to be accessed
  - **Least-recently used (LRU)** - *pick the block in the set which was least recently accessed*
    - Assumed more recently accessed blocks more likely to be referenced again
    - This requires extra bits in the cache to keep track of accesses.
  - **First in,first out(FIFO)**-*Choose a block from the set which was first came into the cache*

Q4: Write Strategy

- When data is written into the cache (on a store), is the data also written to main memory?
  - **If the data is written to memory, the cache is called a *write-through cache***
    - Can always discard cached data - most up-to-date data is in memory
    - Cache control bit: only a *valid* bit
    - memory (or other processors) always have latest data
  - **If the data is NOT written to memory, the cache is called a *write-back cache***
    - **Can't** just discard cached data - may have to write it back to memory
    - Cache control bits: both *valid* and *dirty* bits
    - much lower bandwidth, since data often overwritten multiple times
- Write-through adv: Read misses don't result in writes, memory hierarchy is consistent and it is simple to implement.
- Write back adv: Writes occur at speed of cache and main memory bandwidth is smaller when multiple writes occur to the same block.

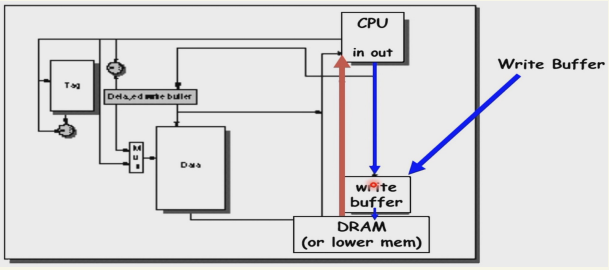
Write stall

- **Write stall** --When the CPU must wait for writes to complete during write through
- **Write buffers**
  - A small cache that can hold a few values waiting to go to main memory.
  - *To avoid stalling on writes, many CPUs use a write buffer.*
  - This buffer helps when writes are clustered.
  - It does not entirely eliminate stalls since it is possible for the buffer to fill if the burst is larger than the buffer.

Write Buffer 可以将 write through 的值缓存，然后写回 Memory. 因此只要不是连续大量 write through, write buffer 就很有效。

→ 速度和 SRAM / Cache 差不多

↳ 随之而来的问题是：Read 时数据还在 Write Buffer 内，还没被写回 Memory?



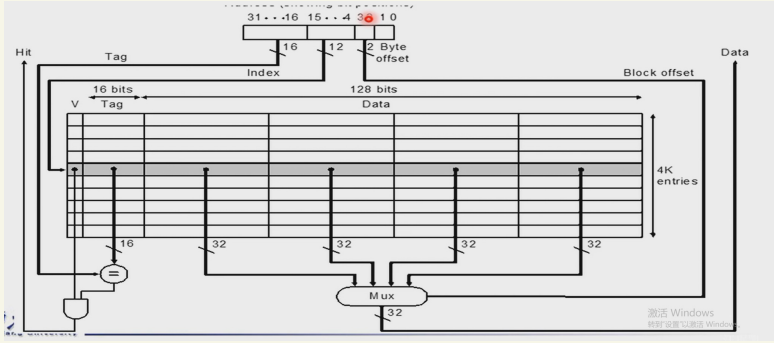
Write misses

- *If a miss occurs on a write (the block is not present), there are two options.*
- *Write allocate*
  - *The block is loaded into the cache on a miss before anything else occurs.*
- *Write around (no write allocate)*
  - *The block is only written to main memory*
  - *It is not stored in the cache.*
- *In general, write-back caches use write-allocate , and write-through caches use write-around .*

这张 PPT 很重要!

↕ Write Through
↕ Write Around

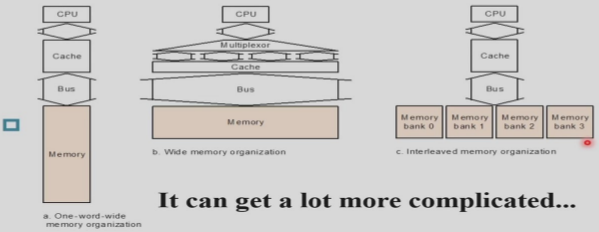
↕ Write Back
↕ Write Allocate



当读入1个Block，如果只用到众多word中的1个，效率就很低。因此最好是每个Block的传输时间和word相近。⇒更新Memory架构!

Designing the Memory system to Support Cache

- Make reading multiple words easier by using banks of memory



It can get a lot more complicated...

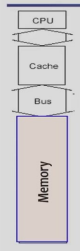
浙江大学 系统结构与网络安全研究所

Bandwidth (带宽)

? Bytes / Clock Cycle.

增加 Memory 宽度，一次传输几个 word.

Performance basic memory organization



Assume

1 clock cycles to send the address
15 memory bus clock cycles for each DRAM access initiated
1 bus clock cycles to send a word of data
Block size is 4 words
Every word is 4 bytes

送地址 找数据 送数据

The time to transfer one word is 1+15+1=17

The miss penalty (The time to transfer one block is):

$$1+4\times(1+15)=65 \text{ CLKs}$$

Bandwidth : $\frac{4\times4}{65} \approx \frac{1}{4}$

Only one word is useful, and three other words may be useless. So, for caches using four-word blocks, this memory system is not viable.

浙江大学 系统结构与网络安全研究所

Performance in Wider Main Memory

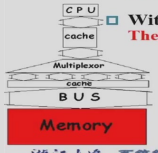
- With a main memory width of 2 words(64bits)

The miss penalty: 4words/Block

$$1+2\times(15+1)=33 \text{ CLKs}$$

only two times that needed to transfer one word.

Bandwidth : $\frac{4\times4}{33} = \frac{16}{33} \approx 0.48$



- With a main memory width of 4 words(128bits)

The miss penalty: 4words/Block

$$1+1\times(15+1)=17 \text{ CLKs}$$

Bandwidth : $\frac{4\times4}{17} = \frac{16}{17} 0.98$

Equal to time to transfer one word.

浙江大学 系统结构与网络安全研究所

缺点：Memory 引脚很多，其成本非常的高。

Performance in Four-way interleaved memory

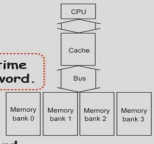
- With 4 banks Interleaved Memory

The miss penalty: 4words/Block

$$1+15+(4\times1)=20$$

Almost equal to time to transfer one word.

Bandwidth : $\frac{4\times4}{20} = 0.8$



Parallel access

Word address	Bank 0	Word address	Bank 1	Word address	Bank 2	Word address	Bank 3
0		1		2		3	
4		5		6		7	
8		9		10		11	
12		13		14		15	

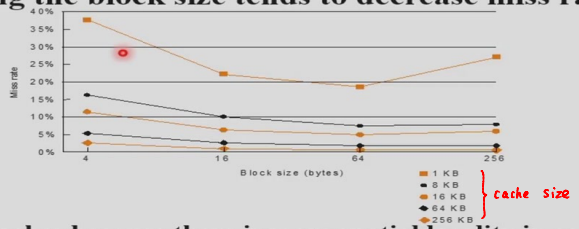
Optimizes sequential address access patterns

浙江大学 系统结构与网络安全研究所

→ 4个 word 并行访问，提高效能。

Performance in different block size

- Increasing the block size tends to decrease miss rate:



- Use split caches because there is more spatial locality in code:

Program	Block size in words	Instruction miss rate	Data miss rate	Effective combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

浙江大学 Zhejiang University 系统结构与网络安全研究所

cache越大，命中率越高.

命中率随Block变化如图.

如何衡量/提高 cache 效率?

Average Memory Assess Time

= Hit-rate × Cache-Time + Miss-rate × Memory-Time

Measuring cache performance

- We use CPU time to measure cache performance.

CPU time= $CPU_{time} = I \times CPI \times Clock\ cycle\ time$

(CPU execution clock cycles + Memory-stall clock cycles) × Clock cycle time

Memory-stall clock cycles = # of instructions × miss ratio × miss penalty
= Read-stall cycles + Write-stall cycles

For Read-stall

$$\text{Read-stall cycles} = \frac{\text{Read}}{\text{Program}} \times \text{Read miss rate} \times \text{Read miss penalty}$$

- For a write-through plus write buffer scheme:

$$\text{Write-stall cycles} = \left(\frac{\text{write}}{\text{Program}} \times \text{Write miss rate} \times \text{Write miss penalty} \right) + \text{Write buffer stalls}$$

- *If the write buffer stalls are small, we can safely ignore them .*
- *If the cache block size is one word, the write miss penalty is 0.*

浙江大学 Zhejiang University 系统结构与网络安全研究所

用CPU Time 衡量.

Read { 取址 / Load 数据

Combine the reads and writes

- In most write-back cache organizations, the read and write miss penalties are the same
  - the time to fetch the block from memory.
- If we neglect the write buffer stalls, we get the following equation:

$$\text{Memory-stall clock cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

We can also write this as:

$$\text{Memory-stall clock cycles} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instructions}} \times \text{Miss penalty}$$

浙江大学 Zhejiang University 系统结构与网络安全研究所

Calculating cache performance

- **Assume:**
  - instruction cache miss rate 2%
  - data cache miss rate 4%
  - CPI without any memory stalls 2
  - miss penalty 100 cycles
  - The frequency of all loads and stores in gcc is 36%

Question: How faster a processor would run with a perfect cache?

- **Answer:**
  - Instruction miss cycles = I×2%×100 =2.00I
  - Data miss cycles = I×36%×4%×100 =1.44I
  - Total memory-stall cycles= 2.00I+ 1.44I =3.44 I
  - CPI with stall = CPI with perfect cache + total memory-stalls
    = (2 + 3.44)I = 5.44I

How faster a processor for ideal

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{I \times CPI_{stall} \times \text{Clock cycle}}{I \times CPI_{perfect} \times \text{Clock cycle}}$$

$$= \frac{CPI_{stal}}{CPI^{'}_{perfect}} = \frac{5.44}{2} = 2.72$$

- **What happens if the processor is made faster?**

Assume CPI reduces from 2 to 1

CPI with stall = CPI with perfect cache + total memory-stalls
=(1+3.44)I = 4.44I

$$\frac{\text{CPU time with stalls}}{\text{CPU time with perfect cache}} = \frac{CPI_{stal}}{CPI^{'}_{perfect}} = \frac{4.44}{1} = 4.44$$

Ratio time for Memory stalls
from $\frac{3.44}{5.44}$ =63% to $\frac{3.44}{4.44}$ =77%

提高CPU性能，反而使 Stall 带来的影响占比更大。

Calculating cache performance with Increased Clock Rate

- *Suppose we increase the performance of the computer in the previous example by doubling its clock rate for same memory system.*
- *Question : How much faster will the computer be with the faster clock to slow clock?*
- *Answer*

Total miss cycles per instruction = (2%×200) + 36%×(4%×200)=6.88
CPI with cache misses = 2 + 6.88 =8.88

$$\frac{\text{Performance with fast clock}}{\text{Performance with slow clock}} = \frac{\text{Execution time with slow clock}}{\text{Execution time with fast clock}}$$

$$= \frac{IC \times CPI_{slow\ clock} \times \text{Clock cycle}}{IC \times CPI_{fast\ clock} \times \text{Clock cycle}/2} = \frac{5.44}{8.88 \times 1/2} = 1.23$$

This, the computer with the faster clock is about 1.2 times faster rather than 2 time faster.

CPU速率提高至2倍

⇒ Miss Penalty 会变成原来的2倍！

（延迟总时间不变，需要更多 cycle）

CPU 提高 2 倍，实际效率提高仅有 1.23 倍.

Solution 1

Reducing cache misses by more flexible placement of blocks

(1) The disadvantage of a direct-mapped cache

(2) The basics of a set-associative cache

(3) Miss rate versus set-associative

(4) Locating a block in the set-associative cache

(5) Size of tags versus set associative

(6) Choosing which block to replace

Miss rate versus set-associativity—8Blocks

Assume: there are three small caches, each consisting of four one-word blocks.
One cache is direct-mapped,
the second is two-way set associative
and the third is fully associative.

Question: Given the following sequence of block addresses:
0, 8, 0, 6, 8, find the number of misses for each cache organization.

Answer: for direct-mapped 5 misses

Memory block	Hit or miss	Contents after each reference			
		Set 0	Set 1	Set 2	Set 3
		Block 0	Block 1	Block 2	Block 3
0	Miss	M[0]			
8	Miss	M[8]			
0	Miss	M[0]			
6	Miss	M[0]		M[6]	
8	Miss	M[8]		M[6]	

浙江大学 Zhejiang University 系统结构与网络安全研究所

Second, for the two-way set associative cache. 4 misses

Memory block	Hit or miss	Contents after each reference			
		Set 0		Set 1	
		Block 0	Block 1	Block 2	Block 3
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[6]		
8	Miss	M[8]	M[6]		

Finally, for the fully associative cache. 3 misses

Memory block	Hit or miss	Contents after each reference			
		Only one set			
		Block 0	Block 1	Block 2	Block 3
0	Miss	M[0]			
8	Miss	M[0]	M[8]		
0	Hit	M[0]	M[8]		
6	Miss	M[0]	M[8]	M[6]	
8	Hit	M[0]	M[8]	M[6]	

Size of tags versus set associativity

Assume
Cache size is 4K Block
Block size is 4 words
Physical address is 32bits

Question
Find the total number of set and total number of tag bits for variety associativity

Answer
Offset size (Byte) = $16 = 2^4$ → 1个Block 4个 word，1个 word 4个 byte，共16 byte.
Number of memory block = $2^{32} \div 2^4 = 2^{28}$ (这里的 Number 是 block 数.)
Number of cache block = 2^{12}

4 bits for address
28 bits for Block address
12 bits for Block address

For direct-mapped
Bits of index = 12 bits
bits of Tag = (28-12) ×4K=16×4K=64 Kbits

浙江大学 Zhejiang University 系统结构与网络安全研究所

For two-way associative
Number of cache set = $2^{12} \div 2 = 2^{11}$
Bits of index = 12-1=11 bits
Bits of Tag = (28-11) ×2×2K=17×2×2K=68 Kbits

For four-way associative
Number of cache set = $2^{12} \div 4 = 2^{10}$
Bits of index = 12-2=10 bits
Bits of Tag = (28-10) ×4×1K=18×4×1K=72 Kbits

For full associative
Number of cache set = $2^{12} \div 2^{12} = 2^0$
Bits of index = 12-12=0 bits
Bits of Tag = (28-0) ×4K×1=112 Kbits

	Direct	2-way	4-way	Fully
Index(bit)	12	11	10	0
Tag(bit)	16	17	18	28

浙江大学 Zhejiang University 系统结构与网络安全研究所

通过在 SRAM (primary cache) 和 DRAM 间增加 secondary level cache，来降低 Miss Penalty.

Decreasing miss penalty with multilevel caches

- **Add a second level cache:**
  - often primary cache is on the same chip as the processor
  - use SRAMs to add another cache above primary memory (DRAM)
  - miss penalty goes down if data is in 2nd level cache
- **Example:**
  - CPI of 1.0 on a 5GHz machine with a 2% miss rate, 100ns DRAM access
  - Adding 2nd level cache with 5ns access time decreases miss rate to 0.5%
- **Miss penalty to main memory is:**

$$\frac{100\text{ns}}{0.2} = 500 \text{ clock cycles}$$

$= \frac{1}{5\text{GHz}} = 0.2$ ns / cycle

- **The CPI with one level of caching**

Total CPI = 1.0 + Memory-stall cycles per instruction
= 1.0 + 2% × 500 = 11.0

Miss penalty with levels of cache without access main memory

$$\frac{5\text{ns}}{0.2} = 25 \text{ clock cycles}$$

浙江大学 Zhejiang University 系统结构与网络安全研究所

- **The CPI with Two level of cache with 0.5% miss rate for main memory**

Total CPI = 1.0 + Primary stalls per instruction + Secondary stalls per instruct
= 1 + 2% ×25 + 0.5% × 500
= 1.0 + 0. 5 +2.5 = 4.0

- **The processor with secondary cache is faster by**

$$\frac{11.0}{4.0} = 2.8$$

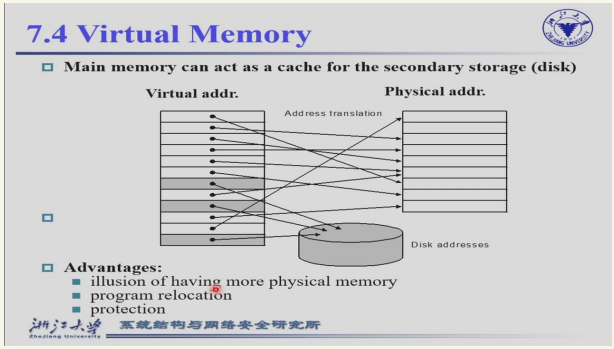
- **Using multilevel caches:**
  - try and optimize the hit time on the 1st level cache
  - try and optimize the miss rate on the 2nd level cache

浙江大学 Zhejiang University 系统结构与网络安全研究所

Virtual Memory

目的 { 不同程序间能安全共享 Memory.
Remove program burden (让每个program认为Memory足够)

作用：把程序的 space address 映射到 physical address



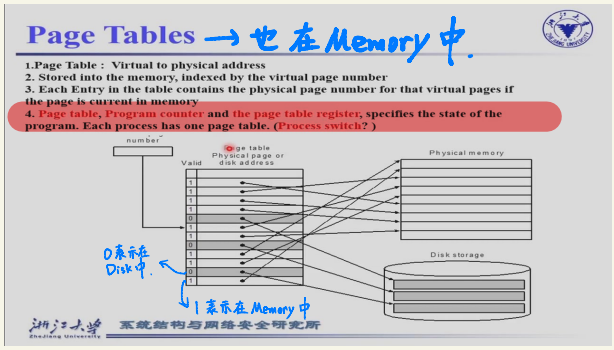
通过把部分 virtual address 直接映射到 Disk，实现了无限内存。当需要 Disk 中的数据时，会将其放入 Memory 中，并改变映射关系。

Page：映射的最小单位。（类似于 Cache 中的 Block）→ 很大

Page Fault：The data is not in Memory.
⇒ Retrieve it from Disk.

LRU 代价最大
可以在软件层面处理而非硬件.

⇒ Page 很大 ⇒ Miss Penalty 很大 ⇒ 使用 Write Back.



每个进程有一张 page table.
通过切换 page table register 的值，切换不同的 page table.

Page Table Number 用来搜索找到 table 中具体哪张 page.

How larger page table?

Assume:

- Virtual address is 32 bits
- page size is 4KB
- Entry size is 4 Bytes

Number of page table entries $= \frac{2^{32}}{2^{12}} = 2^{20}$

Size of pag table $= 2^{20}$ page table entries $\times 2^2 \frac{\text{bytes}}{\text{page table entry}} = 4\text{MB}$

- Five techniques is used to reduce page table size

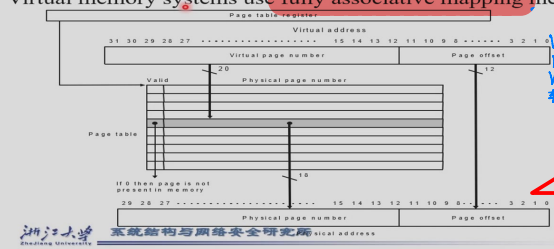
Placing a page and finding it again --Page Tables

Each program has its own page table

Virtual memory systems use fully associative mapping method

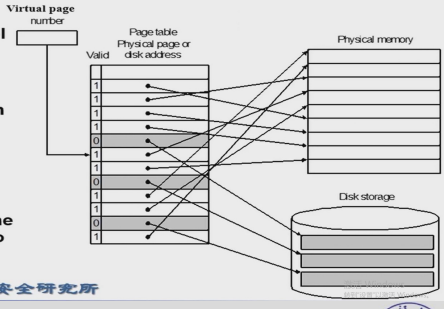
使用全关联

更好地利用 Virtual Memory, 减少从 Disk 向 Virtual Memory 传数据的次数.



Page faults

- When the OS creates a process, it usually creates the space on disk for all the pages of a process(swap space).
  - When a page fault occurs, the OS will be given control through exception mechanism.
  - The OS will find the page in the disk by the page table.
  - Next, the OS will bring the requested page into main memory. If all the pages in main memory are in use, the OS will use LRU strategy to choose a page to replace

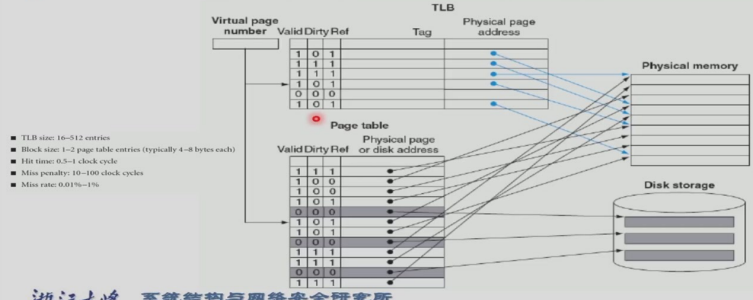


每个程序被创建时, 会在 Disk 上创建一整块空间.

当 page fault 发生, 数据被从 Disk 转入 Main Memory.

Making Address Translation Fast--TLB

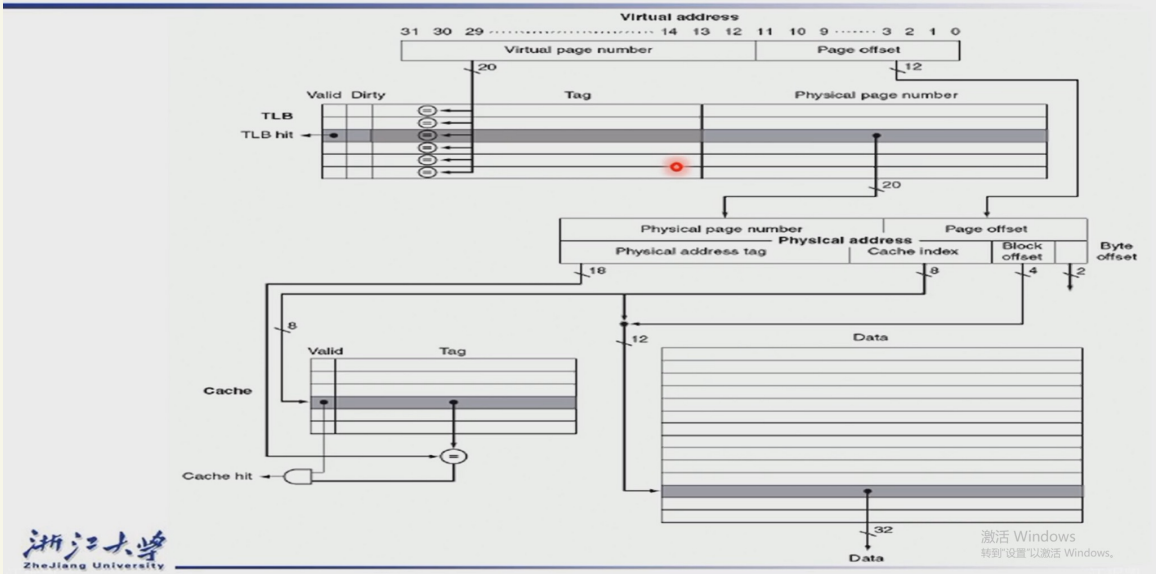
- The TLB (Translation-lookaside Buffer) acts as Cache on the page table
- A cache for address translations: translation look aside buffer



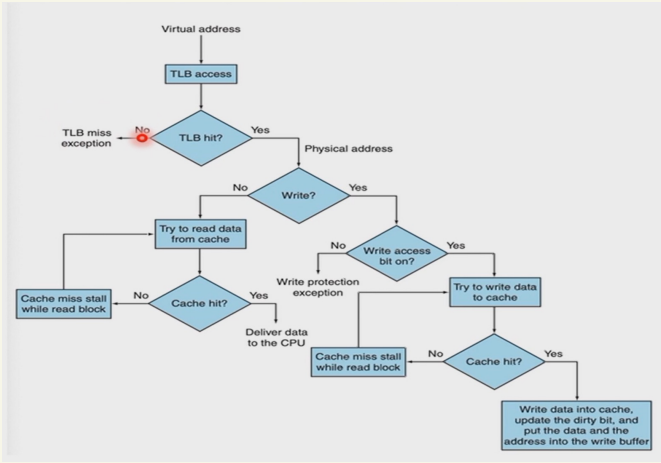
TLB: 用在 page table 上的 cache.

↳ 先在 TLB 找, 找不到再去 page table 找.

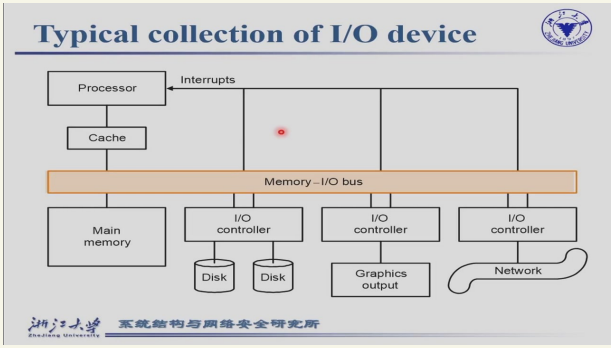
FastMATH Memory Hierarchy



浙江大学 ZheJiang University



Hardware Software Interface



I/O 的衡量非常难进行。

Memory I/O Bus
对 I/O 性能影响非常大。

I/O 的三个特性
- Behavior
  - Input
  - Output
  - Storage
- Partner (和谁对接)
  - 人
  - machine
- Data Rate: I/O Device 和 Memory/Processor 间的最大传输速率

不同 Application 关心 I/O 不同的表现:
1. Throughput
   (1) 单位时间内传输数据量
   (2) 单位时间内操作数
2. Response Time: 返回数据(响应)时间 (Eg: PC)
3. Both (Eg: ATM)

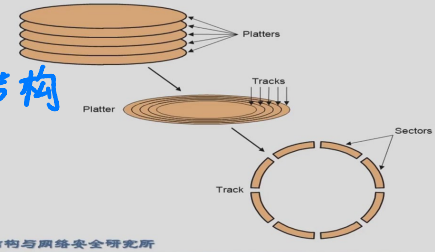
Amdahl's (阿姆达尔)law

- Sequential part can limit speedup
- Example: 100 processors, 90× speedup?
  - $T_{new} = T_{parallelizable}/100 + T_{sequential}$
  - $\text{Speedup} = \frac{1}{(1-F_{parallelizable})+F_{parallelizable}/100} = 90$
  - Solving: $F_{parallelizable} = 0.999$
- Need sequential part to be 0.1% of original time

Amdahl's (阿姆达尔)law

- Remind us that ignoring I/O is dangerous
  - Assume:
    - a bench mark executes in 100 seconds of elapsed time , where 90 seconds is CPU time and the rest is I/O time.
    - CPU time improves by 50% per year, but I/O time doesn't improve.
    - After five years, the improvement in CPU performance is 7.5 times.
  - The elapsed time is reduced to 90/7.5+10=12+10=22 seconds.
  - So, the improvement in elapsed time is only 4.5 times.

- Disks are organized into platters, tracks, and sectors



Disk 结构

Platter

Track

Sector

To access data of disk

- Seek: position read/write head over the proper track
  - minimum seek time
  - maximum seek time
  - average seek time (3 to 14 ms)
- Rotational latency: wait for desired sector

For 5400RPM

$$\text{Average rotational latency} = \frac{0.5 \text{ rotation}}{5400\text{RPM}} = \frac{0.5 \text{ rotation}}{5400\text{RPM} / \left(60 \frac{\text{seconds}}{\text{minute}}\right)} = 0.0056 \text{ seconds} = 5.6 \text{ ms}$$

For 15000RPM

$$\text{Average rotational latency} = \frac{0.5 \text{ rotation}}{15000\text{RPM}} = \frac{0.5 \text{ rotation}}{15000\text{RPM} / \left(60 \frac{\text{seconds}}{\text{minute}}\right)} = 0.0020 \text{ seconds} = 2.0 \text{ ms}$$

计算

To access data of disk

- Transfer: time to transfer a sector (1 KB/sector) function of rotation speed, Transfer rate today's drives - 30 to 80 MBytes/second
- Disk controller: which control the transfer between the disk and the memory

Disk Read Time 512B/sector 50MB/S

Access Time = Seek time + Rotational Latency + Transfer time + Controller Time

$$= 6\text{ms} + \frac{0.5}{10{,}000\text{PRM}} + \frac{0.5\text{KB}}{50\text{MB/sec}} + 0.2\text{ms}$$

$$= 6\text{ms} + 3.0 + 0.01 + 0.2 = 9.2\text{ms}$$

Assuming the measured seek time is 25% of the calculated average

Access Time $= 25\% \times 6\text{ms} + 3.0\text{ ms} + 0.01\text{ms} + 0.2\text{ms} = 4.7\text{ms}$

Flash Types

- NOR flash: bit cell like a NOR gate
  - Random read/write access
  - Used for instruction memory in embedded systems
- NAND flash: bit cell like a NAND gate
  - Denser (bits/area), but block-at-a-time access
  - Cheaper per GB
  - Used for USB keys, media storage, …
- Flash bits wears out after ~~1000's~~ of accesses
  - Not suitable for direct RAM or disk replacement
  - Wear leveling: remap data to less used blocks

随机读写 → 嵌入式

数据块读写 → USB等(便宜)

万次到十万次

有使用次数限制！

⇒有读写算法(使写尽量均匀)

Measure

- *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 Failures) = MTTF+ MTTR* 平均故障间隔时间
- *Availability*

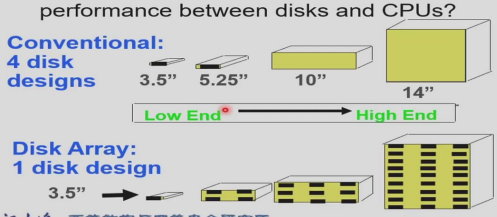
$$Availability = \frac{MTTF}{MTTF+MTTR}$$

Three way to improve MTTF

- ***Fault avoidance:*** *preventing fault occurrence by construction* (及时换硬盘)
- ***Fault tolerance:*** (多备份) *using redundancy to allow the service to comply with the service specification despite faults occurring, which applies primarily to hardware faults*
- ***Fault forecasting:*** (预测) *predicting the presence and creation of faults, which applies to hardware and software faults*

Use Arrays of Small Disks?

- **Katz and Patterson asked in 1987:**
  - Can smaller disks be used to close gap in performance between disks and CPUs?



一个大硬盘用多个小盘代替 ⇒ 接口更多，更快

坏处是：故障概率高.

↳ 解决方法如下：RAID

把文件分块存在不同的Disk中.

Disk 仍会 fail，但数据可根据在冗余中的信息重建.

Redundant Arrays of (Inexpensive) Disks

- **Files are "striped" across multiple disks**
- **Redundancy yields high data availability**
  - Availability: service still provided to user, even if some components failed
- **Disks will still fail**
- **Contents reconstructed from data redundantly stored in the array**

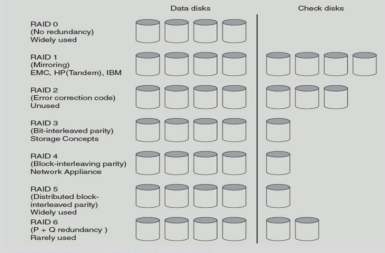
⇒ Capacity penalty to store redundant info

⇒ Bandwidth penalty to update redundant info

RAID: Redundant Arrays of Inexpensive Disks

- **A disk arrays replace larger disk**

RAID level		Minimum number of Disk faults survived	Example Data disks	Corresponding Check disks	Corporations producing RAID products at this level
0	Non-redundant striped	0	8	0	Widely used
1	Mirrored	1	8	8	EMC, Compaq (Tandem), IBM
2	Memory-style ECC	1	8	4	
3	Bit-interleaved parity	1	8	1	Storage Concepts
4	Block-interleaved parity	1	8	1	Network Appliance
5	Block-interleaved distributed parity	1	8	1	Widely used
6	P+Q redundancy	2	8	2	



RAID 0: No Redundancy

- Data is striped across a disk array but there is no redundancy to tolerate disk failure.
- It also improves performance for large accesses, since many disks can operate at once.
- RAID 0 something of a misnomer as there is no Redundancy

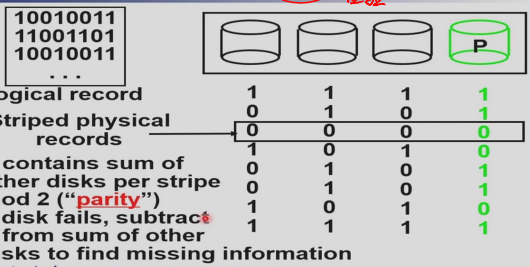
RAID 1: Disk Mirroring/Shadowing

- Each disk is fully duplicated onto its "mirror"
  Very high availability can be achieved
- Bandwidth sacrifice on write:
  Logical write = two physical writes
  - Reads may be optimized
- Most expensive solution: 100% capacity overhead

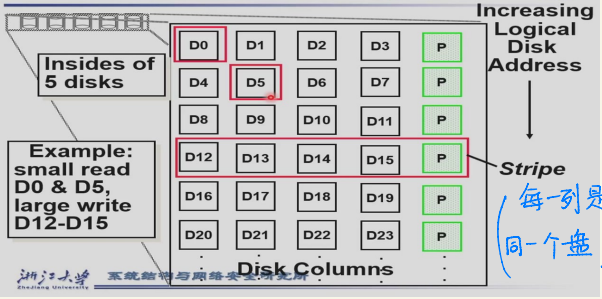
- (RAID 2 not interesting, so skip)

RAID 3: Bit-Interleaved Parity Disk

校验



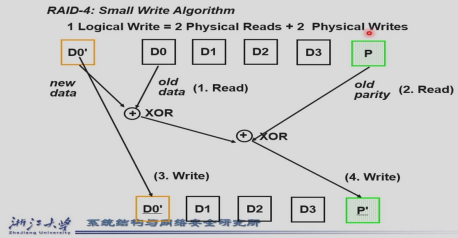
RAID 4:Block-Interleaved Parity



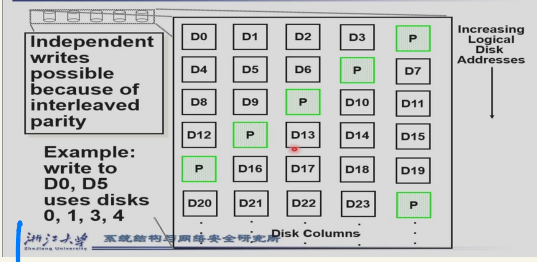
每一列是同一个盘

RAID 4 只对 Small Read 和 Large Write 有效，对 Small Write 效率低。

Problems of Disk Arrays: Small Writes

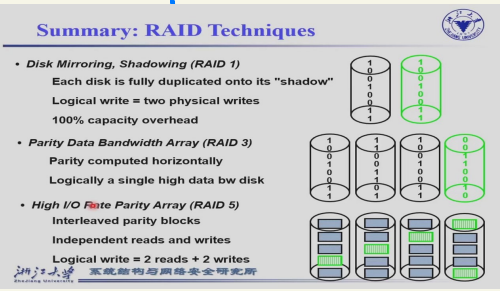


RAID 5: High I/O Rate Interleaved Parity



改进：P盘可变

Summary: RAID Techniques



3的各盘间有依赖关系，4没有。

3：不同盘间校验加存到P.

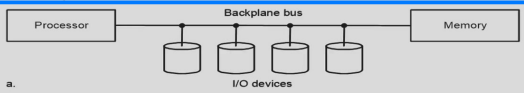
4：同一个盘内的数据校验后存到P.

Bus { Control Line; Data Line }

Bus Transaction { 2 parts: { send address; send / receive data } ; 2 operations: { input data (device → memory); output data (memory → device) } }

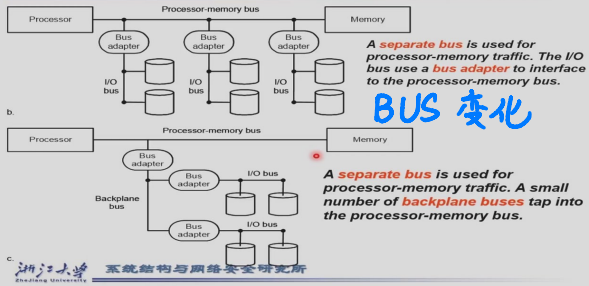
Types of buses

- processor-memory : short high speed, custom design) 短 高速 定制
- backplane : high speed, often standardized, e.g., PCI) 主线，高速，标准化
- I/O : lengthy, different devices, standardized, e.g., SCSI)



Older PCs often use a single bus for processor-to-memory communication, as well as communication between I/O devices and memory.

原先 一根 Bus



A separate bus is used for processor-memory traffic. The I/O bus use a bus adapter to interface to the processor-memory bus.

BUS 变化

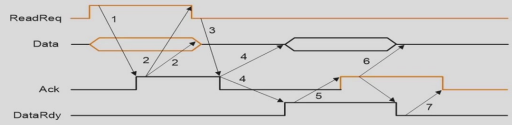
A separate bus is used for processor-memory traffic. A small number of backplane buses tap into the processor-memory bus.

Synchronous vs. Asynchronous

- Synchronous bus: use a clock and a fixed protocol, fast and small but every device must operate at same rate and clock skew requires the bus to be short
- Asynchronous bus : don't use a clock and instead use handshaking
- Handshaking protocol: A serial of steps used to coordinate asynchronous bus transfers.

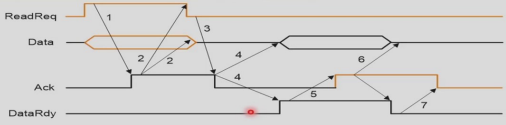
3 解.

Asynchronous example



1. *When memory sees the ReadReq line, it reads the address from the data bus, begin the memory read operation， then raises Ack to tell the device that the ReadReq signal has been seen.*
2. *I/O device sees the Ack line high and releases the ReadReq data lines.*
3. *Memory sees that ReadReq is low and drops the Ack line.*

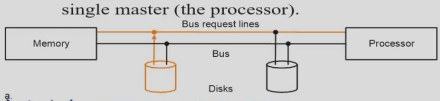
Asynchronous example



4. *When the memory has the data ready, it places the data on the data lines and raises DataRdy.*
5. *The I/O device sees DataRdy, reads the data from the bus , and signals that it has the data by raising ACK.*
6. *The memory sees Ack signals, drops DataRdy, and releases the data lines.*
7. *Finally, the I/O device, seeing DataRdy go low, drops the ACK line, which indicates that the transmission is completed.*

Bus Arbitration

- **Obtaining Access to the Bus**
  - "Without any control, multiple device desiring to communicate could each try to assert the control and data lines for different transfers!"
  - So,a **bus master** is needed. Bus masters initiate and control all bus requests.
    e.g., processor is always a bus master.
  - Example: the initial steps in a bus transaction with a single master (the processor).

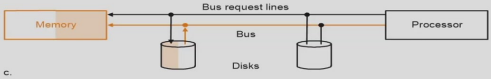


First, the device generates a bus request to indicate to the processor that it wants to use the bus.

Bus Arbitration

b.

The processor responds and generates appropriate bus control signals. For example, if the devices wants to perform output from memory, the processor asserts the read request lines to memory.



c.

The processor also notifies the device that its bus request is being processed; as a result, the device knows it can use the bus and places the address for the request on the bus.

Bus Arbitration

- Deciding which bus master gets to use the bus next In a bus arbitration scheme, a device wanting to use the bus signals a bus request and is later granted the bus.
- Four bus arbitration schemes:
  - *daisy chain arbitration (not very fair)*
  - *centralized, parallel arbitration (requires an arbiter), e.g., PCI*
  - *self selection, e.g., NuBus used in Macintosh*
  - *collision detection, e.g., Ethernet*
- Two factors in choosing which device to grant the bus:
  - bus priority
  - fairness

总线的仲裁：
哪个 Bus Master 拥有
Bus 的控制权

Bus 的衡量：BandWidth

handshake 40 ns：
1、2、3、4 --- 均需 40 ns.

Performance analysis of Synchronous versus Asynchronous buses

Assume: The synchronous bus has a clock cycle time of 50 ns, and each bus transmission takes 1 clock cycle .
The asynchronous bus requires 40 ns per handshake.
The data portion of both buses is 32 bits wide.

Question: Find the bandwidth for each bus when reading one word from a 200-ns memory.

Answer: *synchronous bus:*
the bus cycles is 50 ns. The steps and times required for the synchronous bus are follows:

1. *Send the address to memory : 50ns*
2. *Read the memory : 200ns*
3. *Send the data to the device : 50ns*

Thus, the total time is 300 ns. So,
the bandwidth = 4bytes/300ns = 4MB/0.3seconds
= 13.3MB/second

Performance analysis of Synchronous versus Asynchronous buses

Answer: *asynchronous bus:*
we realize that several of the steps can be overlapped with the memory access time.
In particular, the memory receives the address at the end of step 1 and does not need to put the data on the bus until the beginning of step 5; steps 2, 3, and 4 can overlap with the memory access time.
This leads to the following timing:

step1: 40ns
step2,3,4: maximum(2×40ns+40ns,200ns)=200ns
step5,6,7: 3×40ns=120ns
Thus, the total time is 360ns, so
the maximum bandwidth = 4bytes/360ns = 4MB/0.36seconds
=11.1MB/second
Accordingly, the synchronous bus is only about 20% faster.

Increasing the Bus Bandwidth

Suppose we have a system with the following characteristic:
1. A memory and bus system supporting block access of 4 to 16 32-bit words
2. A 64-bit synchronous bus clocked at 200 MHz, with each 64-bit transfer taking 1 clock cycle, and 1 clock cycle required to send an address to memory.
3. Two clock cycles needed between each bus operation.
4. A memory access time for the first four words of 200ns; each additional set of four words can be read in 20 ns. Assume that a bus transfer of the most recently read data and a read of the next four words can be overlapped.

Find the sustained bandwidth and the latency for a read of 256 words for transfers that use 4-word blocks and for transfers that use 16-word blocks. Also compute effective number of bus transactions per second for each case.

Answer: the 4-word block transfers

Answer: the 4-word block transfers:

each block takes

64-bit synchronous bus

1. 1 clock cycle to send the address to memory
2. 200ns/(5ns/cycle) = 40 clock cycles to read memory
3. 2 clock cycles to send the data from the memory
4. Two clock cycles needed between each bus operation.

This is a total of 45cycles. → 1个 word 4个 cycle.

There are 256/4 = 64 blocks.

So the transfer of 256 words takes

$45 \times 64 = 2880$ clock cycles

The latency for the transfer of 256 words is:

2880 cycles × (5ns/cycle) = 14,400ns.

因此 4 word 需要传 2次 ($\frac{4\times32}{64}=2$)

so the number of bus transactions per second is:

$$64 \text{ transactions} \times \frac{1\text{second}}{14{,}400 \text{ ns}} = 4.44\text{M transactions/second}$$

The bus bandwidth is:

$$(256 \times 4)\text{bytes} \times \frac{1\text{second}}{14{,}400 \text{ ns}} = 71.11 \text{ MB/sec}$$

Answer: the 16-word block transfers:

the first block requires

1. 1 clock cycle to send an address to memory
2. 200ns or 40 cycles to read the first four words in memory.
3. 2 cycles to transfer the data of the set, during which time the read of the next 4-word set is started.
4. It only takes 20ns or 4 cycles to read the next set. After the read is completed, the set will be transferred. Each of the three remaining sets requires repeating only the last two steps.
5. Two clock cycles needed between each bus operation.

Thus, the total number of cycles for each 16- word block is:

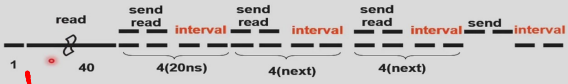
1+40+4×3+2+2=57 cycles.

There are 256/16=16 blocks.

so the transfer of 256 words takes $57 \times 16 = 912$ cycles.

Thus the latency is:

912cycles × 5ns/cycles = 4560ns.



Send read 和 interval 的同时会读下面4个 word (overlapped)

但3就够了

The number of bus transactions per second with 16-word blocks is:

$$16 \text{ transactions} \times \frac{1\text{second}}{4560 \text{ ns}} = 3.51\text{M transactions/second}$$

The bus bandwidth with 16-word blocks is:

$$(256 \times 4)\text{bytes} \times \frac{1\text{second}}{4560 \text{ ns}} = 224.56 \text{ MB/sec}$$

Now,let's put two bus bandwidth together:

4-word blocks: 71.11 MB/sec

16-word blocks:224.56 MB/sec

The bandwidth for the 16-word blocks is 3.16 times higher than for the 4-word blocks.

增加 BandWidth 的方法:

① 提高 Bus Width.

② 分条 Data 线和 Address 线.

③ 一次传输多 words.

A.5 Interfacing I/O Devices to the Memory, Processor, and Operating System

- **Three characteristics of I/O systems**
  - *shared by multiple programs using the processor.*
  - *often use interrupts to communicate information about I/O operations.*
  - *The low-level control of an I/O devices is complex*
- **Three types of communication are required:**
  - *The OS must be able to give commands to the I/O devices.*
  - *The device must be able to notify the OS, when I/O device completed an operation or has encountered an error.*
  - *Data must be transferred between memory and an I/O device*

多程序共享

使用中断与CPU通讯

底层控制复杂(需要驱动)

OS能命令I/O (命令)

I/O能告知OS已完成任务 (状态)

I/O与CPU间能传数据 (数据)

Giving Commands to I/O Devices
Two methods used to address the device

- **memory-mapped I/O:**
  - portions of the memory address space are assigned to I/O devices,and lw and sw instructions can be used to access the I/O port.
- **special I/O instructions**
  - exp: in al,port out port,al
- **command port ,data port**
  - The Status register (a done bit, an error bit……)
  - The Data register, The command register

不可被Cache,因为cache了会使得I/O更新不及时.

在Memory中,I/O有自己的地址空间

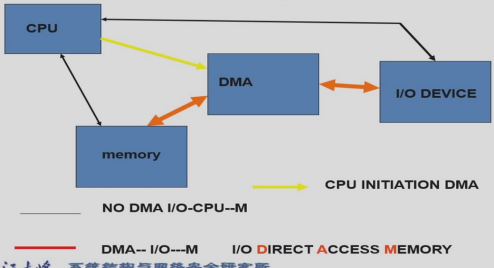
特别的指令

Communication with the Processor

- **Polling: The processor periodically checks status bit to see if it is time for the next I/O operation.**
- **Interrupt: When an I/O device wants to notify processor that it has completed some operation or needs attentions, it causes processor to be interrupted.**
- **DMA *(direct memory access)*: the device controller transfer data directly to or from memory without involving processor.**

直接跳过CPU,CPU配置DMA,从Memory和I/O间直接运输.

DMA transfer mode



DMA transfer mode

- **A DMA transfer need three steps:**
  - The processor sets up the DMA by supplying some information, including the *identity of the device, the operation, the memory address that is the source or destination of the data to be transferred*, and *the number of bytes to transfer*.
  - The DMA starts the operation on the device and arbitrates for the bus. If the request requires more than one transfer on the bus, the DMA unit generates the next memory address and initiates the next transfer.
  - Once the DMA transfer is complete, the controller interrupts the processor, which then examines whether errors occur.

可以利用CPU空闲时间

Compare polling, interrupts, DMA

- The disadvantage of polling is that it wastes a lot of processor time. When the CPU polls the I/O devices periodically, the I/O devices maybe have no request or have not get ready.
- If the I/O operations is interrupt driven, the OS can work on other tasks while data is being read from or written to the device.
- Because DMA doesn't need the control of processor, it will not consume much of processor time.

Overhead of Polling in an I/O System

Assume: that the number of clock cycles for a polling operation is 400 and that processor executes with a 500-Mhz clock.

Determine the fraction of CPU time consumed for the mouse, floppy disk, and hard disk.

We assuming that you poll often enough so that no data is ever lost and that those devices are potentially always busy.

We assume again that:

1. *The mouse must be polled 30 times per second to ensure that we do not miss any movement made by the user.*
2. *The floppy disk transfers data to the processor in 16-bit units and has a data rate of 50 KB/sec. No data transfer can be missed.*
3. *The hard disk transfers data in four-word chunks and can transfer at 4 MB/sec. Again, no transfer can be missed.*

鼠标可用 polling.

如果是硬盘，polling 不可接受.

Answer

the mouse:

clock cycles per second for polling :

$30 \times 400 = 12{,}000$ cycles

Fraction of the processor clock cycles consumed is

$$\frac{12 \times 10^3}{500 \times 10^6} = 0.002\%$$

the floppy disk:

the number of polling access per second:

50KB/2B = 25K

clock cycles per second for polling: $25K \times 400$cycles

Fraction of the processor clock cycles consumed:

$$\frac{10 \times 10^6}{500 \times 10^6} = 2\%$$

Answer

the hard disk:

The number of polling access per second :

4MB/16B = 250K

Clock cycles per second for polling = $250K \times 400$

Fraction of the processor clock cycles consumed:

$$\frac{100 \times 10^6}{500 \times 10^6} = 20\%$$

Now,let's put three fractions together:

Mouse:	0.002%
Floppy disk:	2%
Hard disk:	20%

Clearly, polling can be used for the mouse without much performance impact on the processor, but it is unacceptable for a hard disk on this machine.

Overhead of Interrupt-Driven I/O

Suppose we have the same hard disk and processor we used in the former example, but we used interrupt-driven I/O. The overhead for each transfer, including the interrupt, is 500 clock cycles. Find the fraction of the processor consumed if the hard disk is only transferring data 5% of the time.

Answer: First, we assume the disk is transferring data 100% of the time. So, the interrupt rate is the same as the polling rate.

Cycles per second for disk is:

$250K \times 500 = 125 \times 10^6$ *cycles per second*

Fraction of the processor consumed during a transfer is:

$$\frac{125 \times 10^6}{500 \times 10^6} = 25\%$$

Now, we assume that the disk is only transferring data 5% of the time. The fraction of the processor time consumed on average is:

$25\% \times 5\% = 1.25\%$

As we can see, no CPU time is needed when an interrupt-driven I/O device is not actually transferring. This is the major advantage of an interrupt-driven interface versus polling.

8.7 Designing an I/O system

The general approaches to designing I/O system

- Find the weakest link in the I/O system, which is the component in the I/O path that will constrain the design. Both the workload and configuration limits may dictate where the weakest link is located.
- Configure this component to sustain the required bandwidth.
- Determine the requirements for the rest of the system and configure them to support this bandwidth.

Examples:

Consider the following computer system:

1. A CPU sustains 3 billion instructions per second and it takes average *100,000* instructions in the operating system per I/O operation.
2. A memory backplane bus is capable of sustaining a transfer rate of 1000 MB/sec.
3. SCSI-Ultra320 controllers with a transfer rate of 320 MB/sec and accommodating up to 7 disks.
4. Disk drives with a read/write bandwidth of 75 MB/sec and an average seek plus rotational latency of 6 ms.

If the workload consists of 64-KB reads (assuming the data block is sequential on a track), and the user program need *200,000* instructions per I/O operation, please find the maximum sustainable I/O rate and the number of disks and SCSI controllers required.

Answer:

The two fixed component of the system are the memory bus and the CPU. Let's first find the I/O rate that these two components can sustain and determine which of these is the bottleneck.

$$\text{Maximum I/O rate of CPU} = \frac{\text{Instruction execution rate}}{\text{Instruction per I/O}} = \frac{3 \times 10^9}{(200 + 100) \times 10^3} = 10000 \frac{\text{I/Os}}{\text{seconds}}$$

$$\text{Maximum I/O rate of bus} = \frac{\text{Bus bandwidth}}{\text{Bytes per I/O}} = \frac{1000 \times 10^6}{64 \times 10^3} = 15625 \frac{\text{I/Os}}{\text{seconds}}$$

The CPU is the bottleneck, so we can now configure the rest of the system to perform at the level dictated by the bus, 10000 I/Os per second.

Now, let's determine how many disks we need to be able to Accommodate 10000 I/Os per second. To find the number of disks, we first find the time per I/O operation at the disk:

$$\text{Time per I/O at disk} = \text{Seek/rotational time} + \text{Transfer time} = 6\ ms + \frac{64KB}{75MB/sec} = 6.9\ ms$$

This means each disk can complete 1000ms/6.9ms, or 146 I/Os per second. To saturate the bus,the system need 10000/146≈69 disks.

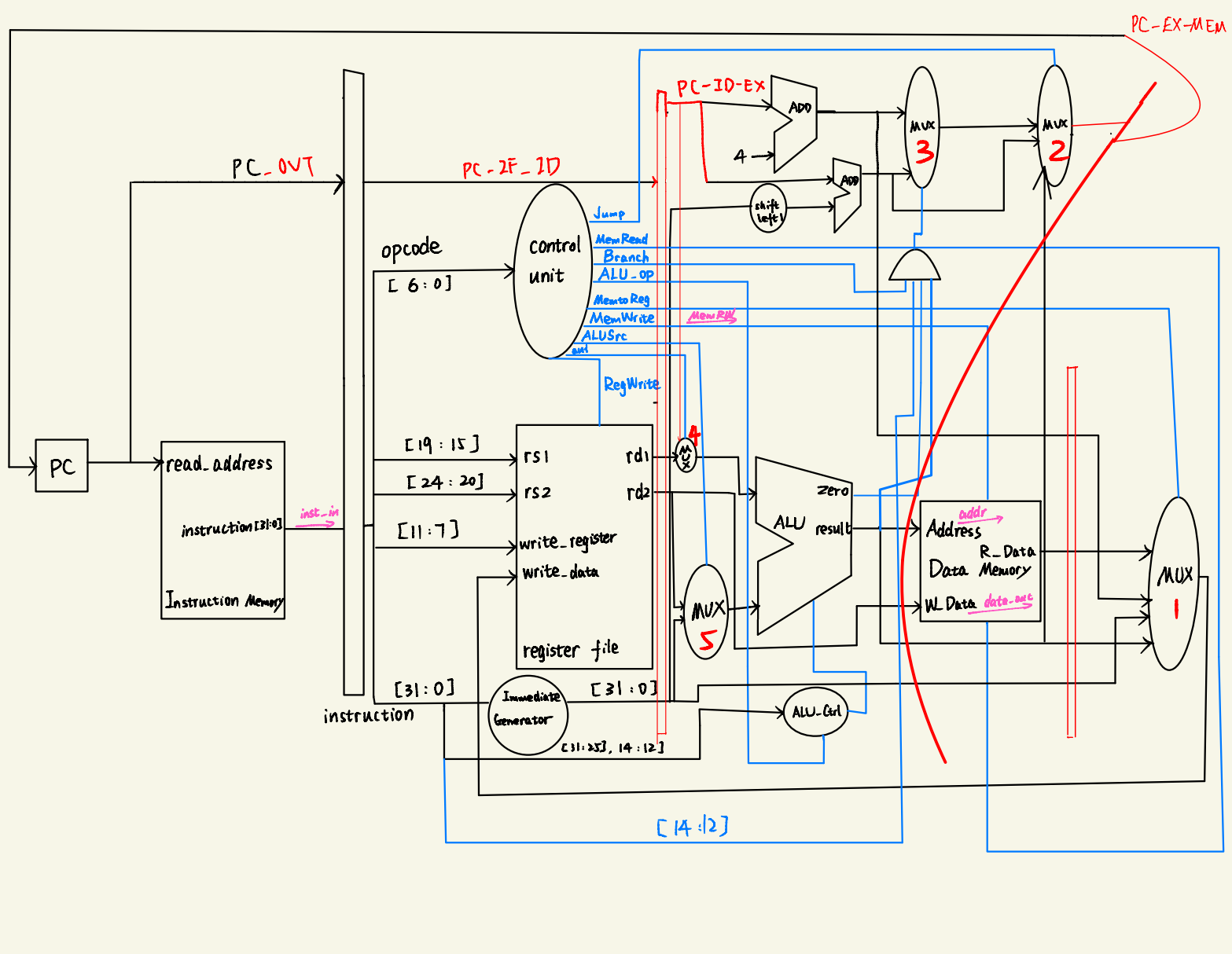
To compute the number of SCSI buses, we need to know the average transfer rate per disk, which is given by:

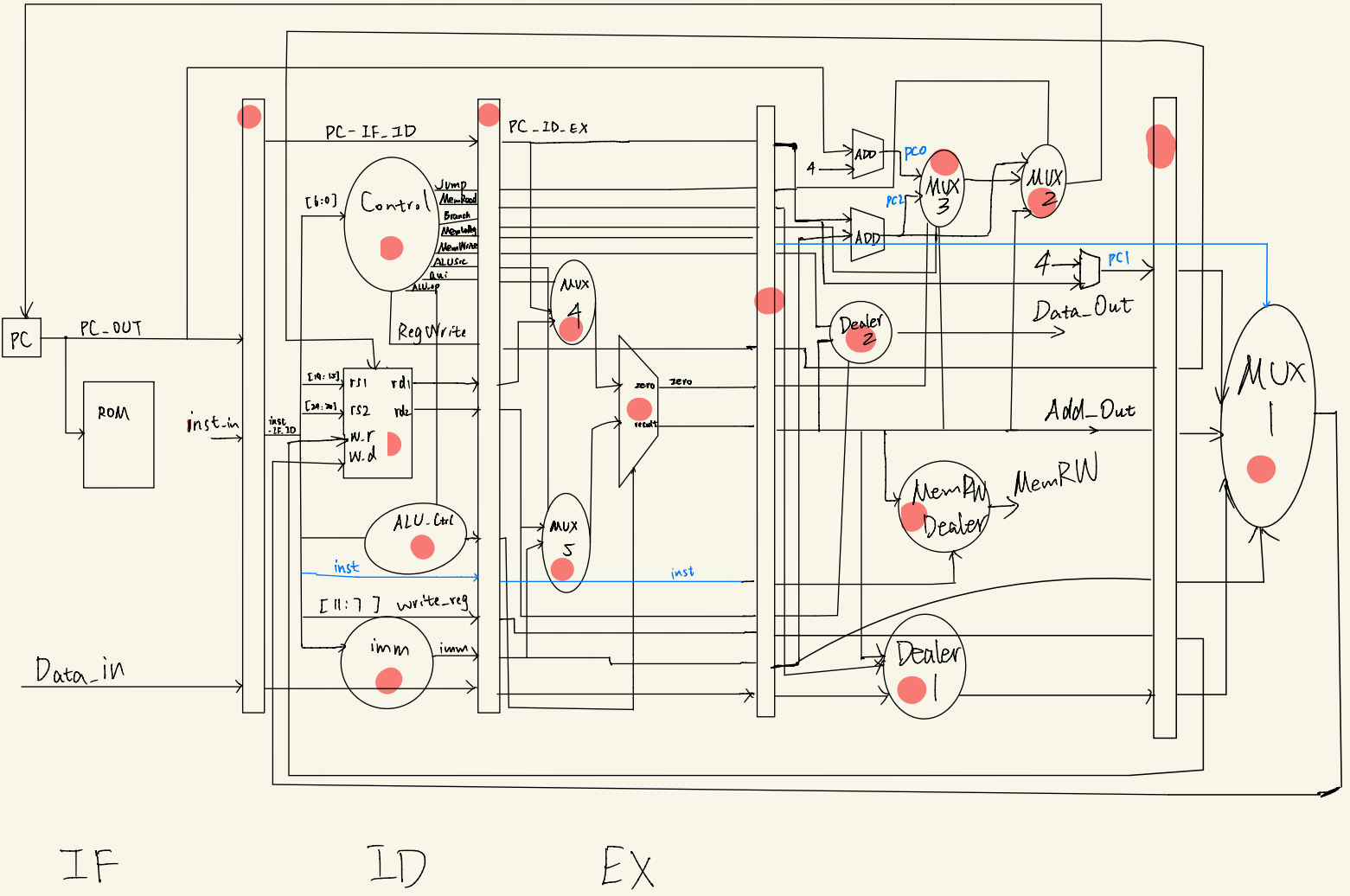
$$\text{Transfer rate} = \frac{\text{Transfer size}}{\text{Transfer time}} = \frac{64KB}{6.9ms} \approx 9.56MB/sec$$

$$69 \times 9.56MB/sec < 320MB/s$$

$$69 \div 7 \approx 10$$

Assuming the disk accesses are not clustered so that we can use all the bus bandwidth, we can place 7 disks per bus and controller. This means we will need 69/7, or 10 SCSI buses and controllers.


PC_EX_MEM
PC_ID_EX
PC_OUT
PC_IF_ID
PC
read_address
instruction[31:0]
Instruction Memory
inst_in
instruction
opcode
[6:0]
control unit
Jump
MemRead
Branch
ALU_op
MemtoReg
MemWrite
ALUSrc
auxi
RegWrite
ADD
4
ADD
shift left 1
MUX 3
MUX 2
[19:15]
[24:20]
[11:7]
rs1
rs2
write_register
write_data
rd1
rd2
register file
MUX 4
MUX 5
[31:0]
Immediate Generator
[31:0]
[31:25], [14:12]
ALU_Ctrl
zero
ALU
result
addr
Address
R_Data
Data Memory
W_Data
data_out
MUX 1
[14:12]


PC
PC_OUT
ROM
inst_in
PC-IF_ID
PC_ID_EX
[6:0]
Control
Jump
MemRead
Branch
MemtoReg
MemWrite
ALUSrc
RegWrite
ALUop
[19:15]
[24:20]
rs1
rs2
W.r
W.d
rd1
rd2
inst_IF_ID
ALU_Ctrl
inst
[11:7]
write_reg
imm
imm
Data_in
MUX 4
MUX 5
zero
zero
result
4
ADD
ADD
PC0
PC2
MUX 3
MUX 2
Dealer 2
4
PC1
Data_Out
Add_Out
MemRW Dealer
MemRW
Dealer 1
MUX 1
inst
IF
ID
EX