前记

练习题：非课后习题，为书中小测题，答案在每章习题后
习题：课后习题。
本篇基本涵盖书中知识点但不包括习题和书上的练习题，老师课上喜欢出练习题，不过听说以往的小测题目都是一样的，可参考往年智云，历年题改变较大，选择题基本没有参考价值，建议做做大题，无聊时做做选择题也不是不行，ppt一定要看，以下内容可以直接当作书来看（因为你看书也记不住多少还浪费时间），时间多就在看练习题和课后习题，时间少看ppt，基本平时课听懂了，这门课补天还是好补的。

考试回忆

大题目多了一道信号如何添加，然后有一道反码补码的计算和一个写机器码的题目，一道写代码，选择题来自ppt，有一道奇怪的下列选项是否需要硬件和软件支持（为书中练习题）。

性能

$性能 = \frac{1}{执行时间}$
X是Y的n倍快：$\frac{性能x}{性能y} = n$
我们用CPU(执行)时间来反映在CPU花费时间，而非CPU工作时间
$程序CPU执行时间 = \frac{程序的CPU时钟周期数}{时钟频率}$
提升到原来多少是指原来的多少倍
CPI每条指令所需平均时钟周期
$CPU时间 = \frac{指令数*CPI}{时钟频率}$

指令：计算机的语言

32个寄存器
2 ** 30的存储字

数据传送指令

lw $s1,20
sw $1,20($ s2) 值得注意的是，字是32位，不同于x86
lh 半字以及lhu 半字无符号 ， 一个进行符号扩充，一个进行无符号扩充
sh
lb 下载一个字节 对应 lbu
sb

ll 说是取数，不知道是啥，原子交换第一部分,load linked word
sc store condition word是上一个的反过来。想起来了，这个是锁的问题。。。

逻辑运算

and
or 注意这都是R型指令
andi
sll shift left logical
srl

分支语句

beq .1,.2,50 这个pc=50`*`4+4+pc
bne
slt .1,.2,.3 set less than 如果.2小于.3那么.1是1，反之为0
sltu
slti
sltiu

跳转语句

j 2500
jr register
jal 2500 jump and link ra寄存器存储pc+4地址
`#`表示一行的注释

关于寄存器

大量寄存器可能使时钟周期变长
字起始地址应该是4的倍数，这是对齐限制
将不常使用的变量存回寄存器叫做寄存器换出
相比存储器，寄存器吞吐率高，访问时间短
`$s0~$s7`映射到寄存器16~23
`$t0~$t7`映射到8~15

指令

op	rs	rt	rd	shamt	funct
6位	5位	5位	5位	5位	6位

R型指令如上rs或rt可以等于rd

op	rs	rt	constant/address
6位	5位	5位	16位

常数绝对值不得超过2`**`15
对应的op不知道要不要考，暂且不写

mips里面没有not，但有nor（或非）not（a or b）`nor a，0`就是对a取反，xor（异或）
做运算时，如果是加法（包括无符号加法)这类，16位立即数会符号扩充，如果是异或这类，会零扩充

决策指令

beq r1，r2，L1
bne 这两个叫条件分支
没有分支目标/分支标签的指令序列叫做基本块。编译最初阶段任务是分解基本块出来
注意没有小于则分支，大于则分支这类说法，因为slt可以满足需求bne

过程

过程约定：
a0~a3:传递参数的四个寄存器
v0~v1:返回结果的两个值寄存器
ra:返回地址寄存器
jal会让下一条指令链接到ra上
PC全名program counter程序计数器
mips里的push是自己先向下移动sp，再存入数据在sp指向块
在过程中，t0~t9随便使用，s0~s7则需要复原。
不调用其他过程的过程叫叶过程
由于再次调用原因，我们每个程序一般都要push ra和a0的，具体如下

```
addi $sp,-8
sw $ra,4($sp)
sw $a0,0($sp)
之后复原
lw $a0,4($sp)
lw $a0,0(sp)
addi $sp,8
jr $ra
```

不过过程不要求保留a0~a3，要求保留ra，sp，s0~s7还有sp之上的栈，这些要求不变
栈中包含过程中所需寄存器和局部变量的片段叫做过程帧或活动记录。帧指针指向一开始sp还未保存时的地方

加锁

交换原语，寄存器把1与存储器的0交换，视为加锁，如果换出来的是1，是为解锁失败

```
addi $t0,$zero,1
ll $t1,0($s1)
sc $t0,0($t1)//存成返回1，失败返回0，重复循环
```

运算

溢出

add、addi、sub溢出时会产生异常（也叫中断）（EPC）
无符号不会溢出，c语言忽略溢出采用的是无符号
ALU 算术逻辑单元
饱和操作指溢出后直接变成最大值
检测溢出时有符号检测两个数符号和和的符号，无符号检测一个数的取反和和的大小
$k0 $k1 是用来溢出处理后返回指令地址的

加快乘除法

这章注意143和138的图，可能要画的

乘法指令

mult multu
mips通过Hi和Lo这两个寄存器存储积，mflo和mfhi 可以将积取出，可以通过检查hi来判断是否溢出
可通过并行加快

除法指令

div
divu
除数开始时置于左32位，右移变小，检测是否可以减去，然后判断该位为0或1（通过减后大于0小于0判断小于则加回去），商左移，末置位不断赋1或0
除法的改进版相比于乘法余数向左移（最终存储值），这是因为控制单元在右边，通过右边产生0

或1来出现商。
有符号数余数的设置要让商的绝对值不变化，余数应该与被除数的符号相同(笔者猜测)。

> 忽略了除数位0，忽略溢出，通过hi和lo来判断吧（mflo和mfhi）注意余数是hi

浮点运算

科学计数法，没有前导零且小数点左边只有一位有效位叫做规格化数。
二进制数科学计数法$1.0 * 2^{-1}$ 浮点数表示
浮点表示：
注意这个尾数第一位表示的是0.5，也就是前导位1是自带的

> 前导位0的表示是通过把指数位设为0来表示的，0这个数是所有位都为0表示
> 无穷大数设置指数位最大，尾数为0，尾数不为0则为NaN（非数（由0/0或无穷减无穷产生））

s	exponent	fraction
1位	8位	23位

还是可能会出现上溢，在指数级容纳不下，还有下溢，就是指数级10^(-38)达不到的地方
double(两个字大小)表示,

s	exponent	fraction
1位	11位	52位

精度到达$2 * 10^{308}$
采用带偏阶（或移码）记数法（biased notation).意思是指数的真实值是减去这个bias，单精度bias是127，双精度是1023，为啥向下取整我不知道（可能是255表示无穷吧，1-254的指数才是正常指数，但是你看这张图多奇怪

找不到“Pasted image 20241220195351.png”。

)
-0.75_{10} 的单精度格式为 1 01111110 10000000000000000000000

浮点加法

会出现保留位的说法，先将指数小的向指数大的对齐，然后舍去到大数所能包容的有效位，有效位此时是规格化科学计数法的有效位，规格化后检查上溢和下溢(-126~127其实就是有效指数位)，然后舍成有效位，再规格化，具体计算看151和152页，注意写的时候注意写清下标是$_2$ 还是

浮点乘法

较为简单，自行看 p155，当然尾数乘法就是乘出来的最高位肯定是前导1

浮点数指令

add.s（单精度加) add.d（双精度加)
sub.s sub.d
mul.s mul.d莫名没了个t也是坑人
div.s div.d
c.x.s c.x.s 就是比较 x可以是eq、neq、lt、le、gt、ge 感觉不会考，不用rd来接受返回值的操作，值存储在浮点标志中
bclt和bclf真分支跳转和假分支跳转 浮点标志是c.x.s这些给出的
有$f0, $f1, $f2这类浮点寄存器，两个寄存器构成一个双精度0和1构成一个（偶-奇搭配)还有lwcl和swcl ，写到这我就忘了汇编的是啥了。。。。
注意没有浮点常数，一般直接常数放在内存取出来

浮点数算术精度

保护位(guard)舍入位(round)保护位在前舍入位在后
保护位原来在这里,有几种舍入策略,round up ,round down ,truncation,向最近偶数舍入

- 保护位舍入，就是看有效位后面两位，0-49舍去，51-99入位。
  浮点数精确性由尾数最低位的单位(ulp)(unit in the last place)给出，这个方法保证误差在半个ulp以内
  粘贴位就是在舍入位后多一位，表示舍入位后是否全为0，如果不全为0，则置1，保护位和舍入位
  运算时这些位都会用到，最后一步才会真正舍去，详情见p163

处理器

存储访问指令和算术逻辑指令分支指令通过ALU实现
图得看p183
选择不同来源数据使用多路选择器（multiplexor)。例如决定 pc+4还是分支目的地址
控制单元(control unit)，给出multiplexor的选择信号
mips数据通路包含两种单元:处理数据值的单元和存储状态的单元
处理数据值的单元是组合逻辑单元(conbinational element),输出只取决于当前输入。ALU就是这类。
有内部存储功能的单元是状态单元(state element)也叫时序(sequential)部件，指令存储器、数据存储器、寄存器都是状态单元。一个状态单元至少有两个输入和一个输出。两个输入时待写入数据值和决定何时写入的时钟信号。

时钟策略

采用边沿触发时钟(edge-triggered clocking)方法,书上假定上升沿发生变化（也可以下降沿）。 `|-|_` 这个横线是上划线。。。,若某状态单元在时钟边沿进行写入，那可以忽略控制信号。如果不是每个周期都进行修改，则要显式的写控制信号。写控制信号和时钟信号都是输入信号，必须稳定等到时钟沿到来才改变状态。
有效(asserted)表示信号为逻辑高或真
无效(deasserted)表示信号为逻辑低或假

建立数据通路

指令存储器，程序计数器，ALU，加法器（计算pc，可以用ALU）
寄存器堆存放32个传统用寄存器。地址偏移还要有个sign-extend单元来符号拓展
注意寄存器堆的输入信号和输出信号同时变换，所以读出同一时钟周期是不可能相互影响到的，所以读出的只会是写入之前的数据，不过可以在末尾读入，也就是一个时钟周期读写嘛

计算分支指令的基地址是下一条指令的地址。
分支发生 branch taken 分支未发生 branch not taken
跳转指令将26位左移2位后，代替PC的低28位。
ALU的控制信号是4位，控制单元的输入是func字段和2位的ALUop，op决定操作是由00(加法)、01(beq的减法)、10（指令的func字段）（R型指令的选择）来决定操作，输出的就是上面讲的4位控制信号。6种组合看p193。

七个1位控制信号

对着书上的图p196看看位置

信号名	置无效时（0）效果	置有效时（1）的效果
RegDst	写入寄存器时，目标寄存器编号来自rt字段	写入寄存器时，目标寄存器编号是rd字段(15:11)
RegWrite	无	数据写入由写入寄存器输入端口指定的寄存器
ALUSrc	第二个ALU操作数来自寄存器堆第二个输出	第二个ALU操作数是指令低16位的符号拓展
PCSrc	PC使用PC+4更新	PC使用分支目标地址更新
MemRead	无	输入地址对应的数据输出到读数据输出端口
MemWrite	无	将写入数据输入端的数据写入地址输入端指定的存储单元
MemtoReg	写入寄存器数据来自于ALU	写入寄存器数据来自数据存储器

控制信号的输入和输出看p200-201

流水线

$指令执行时间_{流水线} = \frac{指令执行时间_{非流水线}}{流水线级数}$ 这个公式要在程序指令多时成立

面向流水线的设计

为什么mips适合流水线:

- 所有指令长度相同
- 指令格式少，并且每条指令寄存器字段位置相同(对称性），确定取指类型同时开始读取寄存器堆
- 存储器操作数仅出现在load和store，不可以直接访问内存
- 所有操作数都必须在存储器对齐

数据冒险

有三种冒险类型

1. 结构冒险:就是两条指令同时访问一个硬件
2. 数据冒险:取数指令取的数是之前的目标，通过增加硬件来提前得到缺少运算项，这个方法叫做前推(forwarding)或者叫做旁路(bypassing)
   取数-使用型数据冒险(load-use data hazard)上一个指令lw，下一个指令就用，这会导致流水线阻塞（pipeline stall)或叫做空泡(bubble)
3. 控制冒险：决策依赖于一条指令的结果，而其他指令正在执行。就是要看第一条指令结果再决定是否进行下一步，其实就是beq是否跳转
   解决方法是:阻塞（stall)或者预测（predict)(通常预测分支不发生如果真发生的话会产生阻塞)或延迟决定（就是用不相关的指令先搪塞着（调节指令顺序））

   > 阻塞对性能影响可以看p210

流水线小结

流水线不能减少单条指令执行时间（也叫延迟（latency）），流水线指示提高了吞吐率（throughput)。

流水线数据通路与控制

数据通路有五个部分：

- IF：取指令

- ID：指令译码，读寄存器堆
- EX:执行或计算地址
- MEM：访问数据存储器
- WB：写回
  每两个阶段中间有一个流水线寄存器，IF/ID这种，store指令最后WB啥也不做，但是也没有优化空间。
  每一个逻辑单元都只能在一个流水线中使用，否则会产生结构冒险，因此这些单元及其控制可以和一个流水级相关联
  流水线可以看p216-220
  多时钟周期图在p223，不过不知道要不要画。。。，单时钟周期就是电路图上面加了个表，在p224

流水线数据冒险：旁路与阻塞

sub指令的结果被下一条指令用到，旁路条件:
1a. EX/MEM.RigisterRd=ID/EX.RegisterRs（Rd不等于0&®write=1）
1b.EX/MEM.RigisterRd=ID/EX.RegisterRt
2a. MEM/WB.RigisterRd=ID/EX.RegisterRs
2b. MEM/WB.RigisterRd=ID/EX.RegisterRd
这个表明流水线下sub这类指令会影响下面两条指令1和2的区别在于下一条还是下面第二条指令
为了这个旁路，我们亲爱的ID/EX流水线寄存器还多了个5位的rs字段，之前是直接保存rs的值的，现在key也要保存了。
这个具体操作还是得看p231，信号看p233

EX冒险	MEM冒险
1.a:forwardA=10	2.a:ForwardA=01
1.bfowardB=10	2.b:ForwardB=01

A和B其实是第一（二）个ALU操作数来源于哪里的问题，00代表来自寄存器堆。
如果出现add s1，s1，1；add s1，s1，1；add s1，s1，1时，从MEM/WB去取，详情看P233，因为MEM级是最新的，EX/MEM最新的还没写进去。
#问了老师可能书上是错的，因为EX是最新的

> sw和lw的冒险看ppt，不过只能避免一次阻塞,也就是说有冲突必有一空泡

冒险与阻塞

load指令必须阻塞一个时钟周期，这个检测叫冒险检测，检测如下：

```
if(ID/EX.MemRead and ((ID/EX.Rt=IF/IF.Rs)or(ID/EX.Rt=IF/ID.Rt)))
stall the pipeline
```

人话就是如果读的话，然后ID/EX的存储寄存器等于下一指令的运算寄存器就停。
阻塞的办法就是保持PC和IF/ID 流水线寄存器不变。
EX开始的部分都是nop，将EX、MEM和WB级的9个控制信号清除（置0）。方法是将ID/EX流水线寄存器的这三个控制信号置为0。
这导致这个指令IF级在第三个时钟周期，ID级在第五个时钟周期

控制冒险或分支冒险

看p238，

假定分支不发生

丢弃指令就是把控制信号置0，要改变IF、ID和EX级的三条控制信号，而不是单单ID级的

缩短分支延迟

将分支执行到ID级，通过逻辑门的异或来判断。
为了在IF级清除指令，我们加入IF.Flush的控制信号，将IF/ID流水线寄存器的指令字段置为0

动态分支预测(dynamic branch prediction)

采用分支预测缓存(cache)或叫做分支历史记录表(history table),一位数据表示最近是否发生，但是一个总是发生分支第一次和最后一次会错误。
因而采用两位预测位，具体实现看p241-242

异常

控制最难部分之一是实现异常和中断。很多操作系统不区分异常和中断，但mips中异常是任何意外的改变，无论内外，中断是由外部引起的事件。具体看p245的分类
I/O设备请求错误就是中断，像算法溢出之类就是异常（不过感觉不是很清楚）

异常发生时在EPC(exception pc)保存出错指令的地址，将控制权转交给操作系统的特定地址。两种方法表示异常原因，一个是Cause寄存器，一个是向量中断（vectored interrupt），在向量中断中，控制权转移到由异常原因决定的地址处。
多个异常在同一时钟周期同时发生先处理最早发生的异常，注意此处EX的异常与下一指令的ID异常为同一时钟周期，而不是IF

指令级并行

多发射（multiple issue）

每一阶段有多个状态单元和组合逻辑单元
分为静态多发射（static）：决策在编译阶段做出 和动态多发射：决策在执行阶段做出

推测（speculation）

交换指令顺序，可能会导致原本不异常的地方异常。

静态多发射处理器

使用编译器帮助打包多条指令并处理冒险。在给定时钟周期发射多条指令，也称为发射包（issue packet).可将发社保看作允许同时执行多个操作的一条指令：超长指令字（Very Long Instruction Word，VLIW)
一般是双发射。（两个指令一次打包。）组合逻辑单元基本都要加倍
详细过程看p252
这个编译器又能处理一个包的各种冒险(不包括包与包之间的)，又要减少load的使用延迟对打包后的ALU的影响，真强大。
答题看p253-254

另一种更高性能编译技术是循环展开(loop unrolling)
此时需要寄存器重命名来消除一些虚假的数据相关，也叫反相关（antidependence)，或者名字相关

动态多发射处理器

也叫超标量处理器（superscalar)
采用动态流水线调度(dynamic pipeline scheduling)

能耗效率

每块芯片集成多个处理器，要比复杂处理器功耗要好。

存储器存储结构

局部性原理：

- 时间局部性(temporal locality):某个数据时间上访问间隔短
- 空间局部性（spatial locality）：空间上间距短
  用局部性原理组织成存储器层次结构
  高级存储器靠近处理器较贵。
  我们将两级层次结构中存储信息的最小单元称为块（block)或行（line）
  如果高层存储器没有找到所需数据那么这次数据请求称为一次缺失。

命中率（hit rate）
缺失率（1-命中率）
hit time
miss penalty
cache由SRAM(静态随机存取存储器)实现
memory由DRAM实现

SRAM

空闲模式下需要最小功率保持电荷，不需要刷新，对任何数据的访问时间是固定的，供电状态下数值不变

DRAM

使用电容保存电荷，因为电容保存，所以不能长久地保持数据，需要周期性地刷新，这也是动态的原因

有DDR（双数据速率）这个型号。这个是时钟上升下降沿都输入输出信息
DDR4 这种可对4个bank发送一个地址同时访问，用轮转方式对这四个bank访问可以提供四倍带宽，也称地址交叉
书上讲的不明所以，建议看ppt

闪存

写操作会产生损耗，所以会有控制器将已经很多次地块重映射到写入次数较少的块中。俗称损耗均衡(wear leveling)技术

磁盘存储器

一个磁盘具有一组瓷盘片，绕轴每分钟转动近10000圈。每层表面有一个包含小型电磁线圈的读写磁头。

每个磁盘表面划分为同心圆盘，称为磁道，每个磁道被划分为存储信息的扇区。扇区容量0.5KiB~4Kib.
访问数据三步骤：

- 寻道。找到适当磁道
- 等待要访问扇区转动到读写头下面，等待时间称为旋转延迟
- 传输时间
  传输时间计算得看ppt

cache基本原理

cache大小不含标记位和有效位，具体算大小位数看p292.标记字段大小是32-n（块的个数）-m（一块中字的个数）-2（默认字的字节偏移量）
cache的有效大小位数$2^n * 2^m * 32$
总位数等于$2^n * (\text{块大小} + \text{标记字段大小} + \text{有效位字段大小})$ 别忘了这个有效位1位
地址映射看p293，是向下取整。

如果仅仅增加块大小，会导致miss penalty的增加。
解决方法就是加快传输速率，减少这个延迟可以隐藏一些传输时间，最简单方法就是提前重启（early restart），即块中所需的字一旦返回就马上继续执行。不过这个问题在于字的分布不确定，同时若下一条指令请求另一块中的字，那么处理器就无法访问cache。还有一种方法是关键字有限，在p295；

cache缺失处理（cache miss）

cache缺失处理由两部分共同完成：处理器控制单元，以及一个进行初始化主存访问和重新填充cache的独立控制器。cache缺失引起流水线阻塞，阻塞整个处理器，冻结所有内容。我们假定顺序执行处理器。

> 乱序执行（out-of-order)处理器在等待缺失处理同时仍能执行部分指令

指令cache缺失处理步骤：

1. PC原始值(PC-4)送到存储器中
2. 通知主存执行一次读操作，并等待主存访问完成
3. 写cache项，将从主存读回数据写入cache存放数据的部分。并将地址高位写入标记字段，设置有效位。
4. 重启执行执行第一步，重新取指，这次该指令在cache中.
   数据缺失同理。

写操作处理

为保持cache和memory的一致性，我们采用两种办法:

- write-through（写直达）直接将更改的字写入cache和memory中
  - 可以使用(write buffer),如果存储器完成写操作速度慢于产生写操作速度，那么没有作用。
- write-back(写回)，被替换时才要写道较低层存储结构。下面各种时cache缺失策略
- 写分配，在cache中分配一块
- 写不分配，只更新主存，不写入cache中。
  写直达比写回安全，因为写直达有内存备份，而写回没有，所以写回必须得判断cache是否命中再完成写操作，而写直达不需要，因为缺失就再次更改cache。

小结

cache性能可通过增加主存带宽：增加存储器位宽和交叉存取。
计算性能看p301
平均存储器访问时间(Average Memory Access Time,AMAT)：
AMAT=命中时间+缺失率*缺失代价

灵活放置块

全相联：块可以放在cache任何位置，适用块数较少的cache
组相联:介于直接映射和全相联之间。每个块有n个可放位置叫做n路组相联cache。（n也称为相联度）
降低缺失率但是增加了命中时间。
采用更换最少被访问的块的规则更新组。
相同容量和相同块大小下，相联度其实也就一路到二路好点，缺失率降低了15%，其余基本改善空间不大。

在组相联中找cache块

标记	索引	块偏移

这个是查找的信息。每组采用并行检索标记号。需要n个比较器和一个n选1的多路选择器。

替换块

采用LRU（Least recently used）法，最久没有被使用的块。
组相联n越大，标记字段大小越大，因为索引其实不占cache大小。

使用多级cache减少损失代价

采用L1和L2cache，多级cache性能看p307
主存缺失代价为$\frac{100ns}{0.25\frac{ns}{时钟周期}}=400个时钟周期$ 这种形式
L1cache致力于减少命中时间，L2cache致力于改善缺失率
快排比基数排序要快因为cache缺失率低。

L2cache的局部缺失率挺高的，但是全局缺失率低。

可信性问题

冗余技术。
失效定义：两种状态：

- 服务完成：交付的服务于需求不符。
- 服务中断：交付的服务与需求不符。
  失效导致状态1到状态2的转换。状态2到状态1称为恢复。
  平均无故障时间（Mean Time To Failure，MTTF）是一个可靠性度量方法。年失效率（Annual Failure Rate，AFR）指在给定MTTF下，一年内预期的器件失效比例。
  服务中断用平均修复时间（Mean Time To Repair，MTTR)来度量。平均失效间隔（Mean time Between Failure，MTBF）=MTTF+MTTR。
  可用性指系统正常工作时间连续两次服务中断间隔时间中所占的比例：
  可用性=$\frac{MTTF}{MTTF+MTTR}$
  用术语故障（fault）来表示一个器件的失效，用三种方式可以提高系统的MTTF：
- 故障避免技术：合理构建系统
- 故障容忍技术：荣誉措施
- 故障预测技术：在器件失效前进行替换

纠正一位错、检测两位错的汉明编码（SEC/DED)

parity校验码检测1位错误，称为1位错误检测编码（1表示奇数，0表示偶数）。汉明纠错码（Hamming Error Correcting Code，ECC）。
我们采用额外的校验位确定单个错误的位置。具体看p315
校验位1、2、4、8（编号二进制第logn+1位为1）。此外，我们可以通过增加1位来导致码字中的最小汉明距离变到4。可以纠正1位错检测2位错。这个增加的校验位取决于前面的奇偶纠错校验位和原数据位的奇偶产生的偶校验
以下是出现的四种情况（H位纠错码组的奇偶性，全局奇偶校验位为p）：

1. H为偶，p为偶，无错误发生
2. H为奇，P为奇，出现一个可纠正错误。
3. H为偶，P为奇，p错了
4. H为奇，p为偶，常出现两位错
   纠错位$2^p >= p + d + 1$

虚拟机

操作系统虚拟机能与硬件匹配。
支持虚拟机的软件称为虚拟机监视器(Virtual Machine Monitor,VMM)或者管理程序(hypervisor):VMM是虚拟机技术核心。底层硬件平台叫主机（host），其资源被客户端（guest）虚拟机共享。
虚拟机保护功能不错，同时，商业意义上，它也有其他两个重要优势：

- 软件管理：提供一个可以运行完整软件栈的抽象
- 硬件管理：允许独立软件栈在共享硬件的同时独立运行，合并了服务器的数量。
  每个客户端只能达到用户级，VMM能达到更高级，从而保证特权指令执行

虚拟存储器（virtual memory）

虚拟存储器中，块称为页，访问缺失叫做缺页。虚拟地址，由软件和硬件结合转化为物理地址。在虚拟存储器中，地址被划分为虚拟页号（virtual page number）和页偏移（page offset），页偏移与物理地址一致，在低位。缺页改进方法看p321

页的存放和查找

页表（page table），存放在主存中，用虚拟地址中的页号作为索引，这个页表位置由页表寄存器给出。每个虚拟机由一块页表

页表每一entry都有一个有效位。由于页表表示了所有虚拟页映射，所以不用标志位。看p324这张图。

如果虚拟也有效位无效，会出现缺页故障，那么内存要去硬盘找到该页。这个交换空间是在进程开始时在闪存或磁盘上创建的交换区(swap space),不管存在主存还是磁盘上，页大小相等。

LRU替换策略

写回机制，替换时写回磁盘。页表中加个脏页。

加快地址转换：TLB

cache快表(Translation-Lookaside Buffer,TLB),TLB中由脏位和引用位等状态位。每次访问先在TLB查找虚拟页号。命中的话物理页号形成地址，引用位被置位，写操作还是设置脏位。TLB缺失要判断时页缺失还是TLB缺失。引用位其实就是用来LRU的。
TLB换项时将引用位和脏位换到页表里（写回操作在TLB缺失时再写，会很高效）
TLB采用全相联，同时随机选择替换表项。
看p330和ppt
还有p331

虚拟地址索引cache块，标记TLB，可以直接知道cache块中有没有所需内容。

TLB的写访问位能让一个恶意进程不能写另一个用户进程的地址空间，写访问位由VMM控制。
进程切换时，TLB一定要切换成另一个表的表项。或者用进程标识符和任务标识符

处理TLB缺失和缺页

- 页在主存，创建表项
- 不在主存，控制权交给操作系统处理缺页
  出现TLB缺失和缺页要使用异常机制，EPC记录程序计数器的值
  异常必须在同一时钟周期的末尾被判定。
  使主存写控制线无效。
  异常处理时禁止其他异常。

页缺失时被引用的页号被保存在BADVaddr的特殊寄存器里。
TLB缺失表项插入过程看p336，有点麻烦 处理时并不会检查表项是否有效，直接会插入。

小结

虚拟存储器允许单个程序地址空间拓展到主存界限之外。增大memory的表面大小（apparent size)
采用技术降低缺失率：

- 增大页容量
- 全相联
- LRU和访问位之类技术决定选择替换哪一页

存储器框架

cache容量增加，相连度提高对性能改进作用很小，其余看p339吧，感觉没啥。
页表采用的是全相联全映射
cache替换策略：相联度较高的采用随机法，较低的采用LRU（近似实现）
写直达比写回好实现。缺失代价小：不用把整块写回更低级存储系统

3C模型

- 强制缺失（compulsory miss）也称冷启动(cold-start miss)：
  从未出现过的块的第一次访问缺失
- 容量缺失(capacity miss)： cache某个块被替换后再次访问的缺失
- 冲突缺失（conflict miss）也称碰撞缺失(collision miss)：
  多个块竞争同一组，全相联不存在这个问题。
  强制缺失可以增大块来解决
  容量缺失增大容量
  看p343的表

有限状态机控制简单的cache

有限状态机通常假定那些没有明确置为有效的信号设置为无效信号。有限状态机实现看p345，由一个组合逻辑和一个保持当前状态的寄存器实现。称为阻塞性（blocking）cache。

一个简单cache控制器的有限状态机

cache控制器的4个状态:

- 空闲：
- 标记比较：

- 写回：将128位的块写回存储器（DRAM控制器通常是128位）
- 分配：
  懒得写了看p346吧，都要等待准备好信号。标记比较状态可以和cache访问分离，以及价格
  写缓冲可以改进时钟周期

cache一致性

多核多处理器，多个处理器共享一个公共的物理地址空间。但是cache各自拥有
一致性和连贯性：

- 单个CPU写后读的就是写过的
- 一个写操作后其他CPU在一定间隔后可以得到写过的
- 写操作是串行执行的，不会出现同时写入
  实现一致性的基本方案：
- 迁移（migration）：数据项移入本地cache，这样可以不用访问共享寄存器
- 复制（replication）：共享数据被同时读取时，cache在本地对数据项做备份，减少读取竞争
  cache一致性协议：最常用的是监听(snooping)协议。cache保留数据块共享状态的副本，不集中保存状态。所有cache控制器对广播介质（总线或者网络）进行监视或者监听，来确保是否由总线或交换机上的数据块副本。

监听协议

写无效协议(write invalidate protocol)处理器写操作前令其他副本无效。此时其他处理器读时需要重新请求新的数据副本，这时仍有效的CPU数据副本就会响应。
不过写操作处理器要独占访问，采用random rule。

I/O

打印机采用轮询（polling）方式，其他基本采用中断，较慢的情况下