前言及目录

要学计算机组成原理，你得知道它是干什么的，抓住总线：如何改进计算机的性能。

Chapter 1 计算机概要与技术

带你了解了解计算机发展史，告诉你计算机系统结构的 8 个伟大思想，计算机的设计以及性能的改进离不开这 8 个伟大思想，而后告诉你性能的概念以及量度标准；

——01——

Chapter 2 指令：计算机的语言

带你学习计算机的语言，也就是指令，这为学习 Chapter 4 中的流水线指令集打下基础；

——11——

Chapter 3 算术运算

计算机的算术运算，这大家不会陌生，在计算机科学导论这门课程中已经讲到过许多，如果你导论学得通透，那么这一章对你会特别轻松；

——21——

Chapter 4 处理器

流水线，是对计算机指令执行速度上的改进，这一章的各种流水线图以及数据通路图一定要多画画，画几遍就自然地记住了，注意理解记忆各部件在什么时候是怎么运行的，徒手画数据通路图是必备本领；

——28——

Chapter 5 大容量和高速度：开发存储器层次结构

存储器容量和速度的改进，速度上利用 cache，容量上利用虚拟存储器技术。

——39——

Chapter 6 往年真题×2

18-19 年的真题，附带卷王手写答案

——50——

这些精彩内容正等待着聪明的你去探索！祝各位早日走出计组的迷雾！

Chapter 1 计算机概要与技术

一、计算机的发展

1. 计算机的发展是伴随着微电子器件的发展而发展的。

电子管 ⟶ 晶体管 ⟶ 半导体 ⟶ 微处理器

2. 计算机的分类：

个人计算机（PC）：性能良好，价格低廉；个人使用，键盘，鼠标

服务器：基于网络访问，强调可靠性

嵌入式计算机：运行单一应用程序，和硬件集成在一起，严格限制成本功耗

3. 后 PC 时代：

个人移动设备（PMD）：智能手机、平板、电池供电、触摸屏输入

云计算：在网络上提供服务的大服务器集，可供租用

软件即服务（SaaS）：在网络上以服务的方式提供软件、数据；eg. web 搜索、社交网络（通过浏览器登录到远程服务器执行）

二、计算机系统结构的 8 个伟大思想

1. 面向摩尔定律的设计

单芯片集成度每 18~24 个月翻一番

注意：摩尔定律是对硬件而言，这意味着如果有一个判断题说：系统软件的发展符合摩尔定律。那么答案是错误。

2. 使用抽象简化设计

高层次看不到低层次的细节

3. 加速大概率事件

设计者要知道什么事件经常发生

4. 通过并行提高性能

并行执行

5. 通过流水线提高性能

流水线执行(Chapter 4 具体讲流水线)

6. 通过预测提高性能

“求人准许不如求人原谅”

7. 存储器层次

速度最快、容量最小、每单位价格最贵的存储器

注意：是每位的价格，这意味着如果有一个判断题说：速度最快、容量最小、价格最贵的在顶层，那么答案是错误。因为单价和总价之间还有数量的制约，每位的价格是单价，价格是总价。

8. 通过冗余提高可靠性

冗余部件替代失效部件

eg. 牵引式挂车后轴双轮胎

注意：仔细体会这 8 个伟大思想，在第二、四、五章会经常提到。

三、理解程序性能

软件 = 程序 + 文档 = 数据结构 + 算法 + 文档

数据结构 + 算法

1. 对性能的影响：

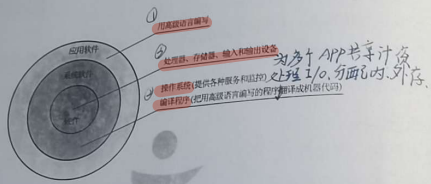
算法：决定源代码数量，决定执行 I/O 操作的数量

程序语言，编译器，体系结构：决定每行源代码对应机器指令的数量

处理器和存储系统：决定指令执行的速度快慢

I/O 系统：决定 I/O 操作执行的速度快慢

2.



3. 编程语言的发展

手编程序：机器语言程序，手工编译

汇编程序：符号语言程序，汇编程序汇编

高级程序：算法语言，机器编译/解释程序

4. 程序语言级别

高级语言 —编译→ 汇编语言 —汇编→ 机器语言 —链接→ 可执行文件

（最接近问题领域的抽象级别，高效&可移植）　（指令的文本表示）　（二进制）

四、硬件概念

1. 所有种类的计算机都有相同的组成部件。

冯·诺伊曼：

（1）五大部件：

① 存储器（Memory）

② ALU

③ 控制单元

④ I/O （算作两个）

注意：如果是说三个部件，则把 ALU 与控制单元合称为 CPU。

2. 运算器

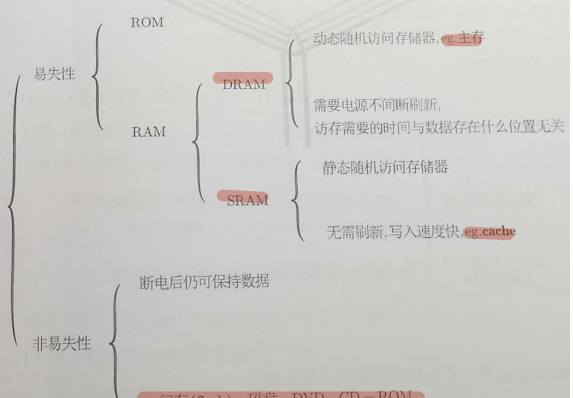
算术运算 & 逻辑运算，参与的数都是二进制的。

3. 存储器

（1）存储程序和数据。

（2）根据提供的地址和读写命令存/取数据

（3）分类：



4. 控制器

指令格式：指令码+地址码

按照指令一条一条：取指令 → 指令译码 → 执行指令

指令和数据如何区分：

指令：取指发生在取指周期，送向控制器

数据：取数发生在执行周期，送向运算器

5. 适配器

使被连接的外围设备通过总线与主机联系，并行工作。

6. I/O 设备

人或其他机器设备能接收识别的信息形式（二进制信息形式）

7. CPU 内部构成

CPU内部构成：
- 数据通路(算术运算)
- 控制器(指导其他部件按指令正确执行，“总指挥官”)
- cache（快速访问存储器，小而快的SRAM）

8. 抽象

指令集-体系结构（硬件和底层软件之间的接口）

应用二进制接口（基本指令集 & 操作系统接口）

五、

- 联网的计算机can do what：通信 + 资源共享 + 远距离访问
- 局域网 LAN（以太网：将一个建筑物同一层的计算机连接起来）
- 广域网 WAN（万维网、Internet）
- 无线网（eg. Wifi、蓝牙）

PS：芯片制造：投入高、风险大、回报慢的高壁垒行业

六、性能

1. 性能

$$\text{性能} = \frac{1}{\text{执行时间}}$$

$$\frac{\text{性能x}}{\text{性能y}} = \frac{\text{执行时间y}}{\text{执行时间x}} = n$$

那么计算机 x 是计算机 y 的 n 倍快。

2. CPU 执行时间

CPU 执行时间：执行某一任务在 CPU 上所花的时间。

include：用户CPU时间（程序本身费时） & 系统CPU时间（为执行程序在操作系统上的费时）

$$\text{一个程序的CPU时间} = \text{CPU时钟周期数} \times \text{时钟周期} = \frac{\text{CPU时钟周期数}}{\text{时钟频率}f}$$

由此公式，可以知道性能改进可以通过以下方法：

（1）减少程序的时钟周期数量

（2）增加时钟频率

（3）硬件设计者需要权衡时钟频率、时钟周期数量

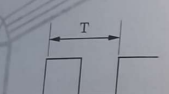
注意：用户感受到的是响应时间，而非 CPU 时间。

3. 响应时间和吞吐率

- 响应时间：完成某任务需要的总时间
  - include硬盘访问、内存访问、I/O、CPU执行
- 吞吐率：带宽，单位时间内完成的任务数量
- 联系：二者互相影响
  - 更换更高型号的处理器：响应时间变快，导致吞吐率变大
  - 处理多任务时，系统要求后续请求排队：吞吐率变大，使得响应时间更快

注意：增加多个处理器分别处理任务，只增大了吞吐率，与响应时间无关。

4. 时钟周期与时钟频率



- 时钟周期T：一个时钟持续的时间(以秒为单位)
- 时钟频率f：每秒包含的时钟周期数(以Hz为单位，Hz：周期数/s)

$$f=\frac{1}{T}$$

$1Hz=10^{-3}KHz=10^{-6}MHz=10^{-9}GHz$　　Hz KHz MHz GHz

$1s=10^{3}ms=10^{6}\mu s=10^{9}ns=10^{12}ps$(皮秒)　　m μ n p

5. 每条指令的平均时钟周期数：CPI

时钟周期数 = 指令数(IC) × CPI

$$\text{CPU时间}=\text{程序指令数}\times\text{CPI}\times\text{时钟周期}=\frac{\text{程序指令数}\times\text{CPI}}{\text{时钟频率}}$$

一台计算机的不同程序中，CPI 不同。

有些处理器在每个时钟周期可以对多条指令取指并执行，故 CPI 可能小于 1。

不同类型指令的时钟周期数不同，则

$$\text{CPU时钟周期数}=\sum_{i=1}^{n}(\text{CPI}_i\times\text{I}_i)$$

$$\text{加权平均CPI(Avg.CPI)}=\frac{\text{CPU时钟周期数}}{\text{指令数}}$$

6. 性能影响

- 算法：影响指令数（IC）、CPI（可能影响 CPI）（这是因为指令有快有慢）
- 编程语言：影响 IC 和 CPI（调用方式影响 CPI）
- 编译程序：影响 IC 和 CPI（翻译过程的效率）
- 指令集体系结构：影响 IC 和 CPI（时钟频率）

注意：CPU 执行时间才是可靠的性能度量标准，任何一个独立的因素都不行。

思考：功耗墙不能再降低电压，也不能散去更多热量，那怎么改进性能呢？

七、例题

假设某一程序，指令条数 IC，其中 70% 是算术运算指令，10% 是存取指令，20% 是分支指令，执行上述指令分别需要 2、6、3 个周期。

（1）求 Avg.CPI

$$Avg.CPI=\frac{70\%\times IC\times 2+10\%\times IC\times 6+20\%\times IC\times 3}{IC}$$

$$=1.4+0.6+0.6$$

$$=2.6$$

（2）存取指令和分支指令的执行时间不变，使性能提升25%，则算术运算指令的CPI为？

（3）求算术运算指令的平均时间。

例题参考答案：

（1）$\text{Avg.CPI}=\dfrac{(0.7IC*2+0.1IC*6+0.2IC*3)}{IC}=2.6$

（2）$CPI_{算}+0.1*6+0.2*3=2.6*0.75$

可以得到$CPI_{算}=0.75$

（3）$0.7*n+0.1*6+0.2*3=2.6*\dfrac{1}{1.25}$

得到n＝1.26（个时钟周期）

思考题参考答案：单处理器到多处理器

Chapter 2 指令：计算机的语言

一、常见的 MIPS 汇编语言指令

1. 寄存器

（1）算术运算指令中，计算机硬件的操作数是寄存器操作数。

（2）MIPS 操作数可以来自 32 个 32bit 的寄存器和 2^{30} 个存储器字，寄存器用于数据的快速存取，存储器只能通过数据传输指令访问。

理解：可以理解存储器（Memory）中的数据只能存或者取，不能直接进行算术运算，而算术运算中如果要用到存储器中的数，可以先通过数据传输指令把存储器中的数放到寄存器中，再用算术运算指令对该寄存器进行操作。访问寄存器比访问存储器快得多，编译器必须尽量使用寄存器来访问变量。

注意：MIPS 字节编址寻址，所以连续的字相差 4，这在 MIPS 汇编语言常见指令中有所体现。一个字节是 8bit，一个字是 32bit（MIPS 指令中）。

2. 常见寄存器

（1）$\$t_0 \sim \$t_9$：临时寄存器

（2）$\$s_0 \sim \$s_7$：用于存储变量（因此指令中，这些寄存器很常见）

（3）$zero：恒置零（这是伟大思想中加速大概率事件的体现，0 经常用到）

（4）$at：用于处理大的常数

3. MIPS 汇编语言常见指令（详见教材 41、42 页）

（1）算术运算指令

① 加法：add $\$s_1,\$s_2,\$s_3$

$\$s_1 = \$s_2 + \$s_3$

理解：$\$s_1$ 是目标寄存器，$\$s_2,\$s_3$ 是源操作数寄存器，是把两个源操作数进行算术运算，得到的结果存放到目标寄存器中，因此不要受习惯影响写成 $\$s_3 = \$s_1 + \$s_2$

② 减法：sub $\$s_1,\$s_2,\$s_3$

$\$s_1 = \$s_2 - \$s_3$

③ 立即数加法：addi $\$s_1,\$s_2,20$

$\$s_1 = \$s_2 + 20$，用于加常数数据

注意：MIPS 中没有 subi 指令。

（2）数据传输指令

① 取字：lw $\$s_1,20(\$s_2)$

$\$s_1 = Memory[\$s_2 + 20]$，将一个字从内存中取到寄存器中

理解：lw = load word 取字，寄存器 $\$s_2$ 的值是基址，20 是偏移量，有效地址是基址加上偏移量。

② 存字：sw $\$s_1,20(\$s_2)$

$Memory[\$s_2 + 20] = \$s_1$，将一个字从寄存器中取到内存中

理解：sw = store word 存字，20 是偏移量

③ 取立即数的高位：lui $\$s_1,20$

$\$s_1 = 20 * 2^{16}$，取立即数并放到高 16 位

（3）逻辑运算指令

① 与：and $\$s_1,\$s_2,\$s_3$

$\$s_1 = \$s_2 \& \$s_3$，按位与

② 立即数或：ori $\$s_1,\$s_2,20$

$\$s_1 = \$s_2 | 20$，和常数按位或

③ 逻辑左移：sll $\$s_1,\$s_2,10$

$\$s_1 = \$s_2 \cdot 2^{10}$（相当于左移 10 位）

$Ss_1=Ss_2<<10$，根据常数左移相应位，相当于乘 2^{10}

（4）条件分支指令

① 相等时跳转：beq $Ss_1,Ss_2,25$

if($Ss_1==Ss_2$) go to PC+4+100

理解：为什么 25 变成 100 了？这是因为上文提到，MIPS 字节编址，字寻址，即编址的时候是一个字节一个字节编号的，如果编了 4 个号，这连续的 4 个号构成一个字（MIPS 中一个字 32bit，是 4 个字节），寻址的时候是按照一个字一个字去寻找，因此，25 在寻址的时候要乘 4 得 100。

那为什么多了个 4？这在后面将会学到，因为取的是下一条指令，而每一条指令差 4 个字节。

注意：由于 MIPS 的字节寻址字编址，明确的地址是要乘 4 的，偏移量是不需要乘 4 的。

② 小于时置位：slt Ss_1,Ss_2,Ss_3

if($Ss_2<Ss_3$) $Ss_1=1$ else $Ss_1=0$

理解：这里还是注意，进行比较操作的时两个源操作数，结果存到目标寄存器中。

（5）无条件分支指令

j 2500

go to 10000

理解：这里还是明确的地址，因此要乘 4。

二、计算机硬件的设计原则

1. 简单源于规整

所有的 MIPS 指令都是 32bit

2. 越小越快

大量的寄存器会使时钟周期变长，因此使用 32 个寄存器。

3. 优秀的设计需要适宜的折中方案

如虽然简单源于规整，但是既希望指令长度相同，又希望有统一的指令格式，会产生冲突，因此折中方案就是保持指令长度相同，但是不同指令类型采用不同的指令格式。

三、大端寻址和小端寻址

1. 大端寻址：高位放在低地址（MIPS 采用大端编址）

2. 小端寻址：高位放在高地址

举个栗子：用大端模式存 12345678H 这个数

地址偏移	大端模式
0x00	12
0x01	34
0x02	56
0x03	78

理解：12345678H 这个数的高 8bit 为 12，按照大端模式高位放在低地址，所以放在 0x00 低地址，其他位依次存放。

四、MIPS 指令的类型（详见教材 55 页，76 页）

1. R 型（Register：寄存器）

op	rs	rt	rd	shamt	funct
操作码（6bit）	一源 Reg（5bit）	二源 Reg（5bit）	目的 Reg（5bit）	偏移量（5bit）	功能码（6bit）

举个栗子：

add s_1,s_2,s_3，其中，s_1 是 17 号寄存器，s_2 是 18 号，s_3 是 19 号。

32D	0D	17D	19D	18D	000000

理解：add 的操作码是 000000，一源 Reg 是 s_2，故 rs 为 18，这里是用十进制表示，比较直观。add 指令的功能码是 32D。指令的 op 码和功能码见教材 56 页图 2-5，更多栗子见教材 57 页图 2-6。

2. I 型（立即数）

op	rs	rt	常数或地址
操作码（6bit）	一源 Reg（5bit）	目的 Reg（5bit）	（16bit）

注意：这里的 rt 不是一源 Reg，而是目的 Reg，区别于 R 型指令的 rt 和 rd。常数或地址有 16bit，这意味着可相对基址寄存器的地址偏移 $\pm 2^{15}$ 个字节。

3. J 型（跳转指令）

op	1000
操作码（6bit）	（26bit）

4. 分支

op	rs	rt	Exit
操作码（6bit）	一源 Reg（5bit）	二源 Reg（5bit）	分支地址（16bit）

五、过程

过程：根据提供的参数执行一定任务的存储的子程序。

理解：可以类比 C 语言程序设计里面的函数调用帮助理解。

a_0 ~ a_3 存放参数，使得过程可以访问这些参数

↓

jal X

跳转到过程X，并将下一条指令的地址（即PC + 4）放到$ra中

↓

控制权交给过程，过程执行得到结果，把结果存在v_0 ~ v_1中

↓

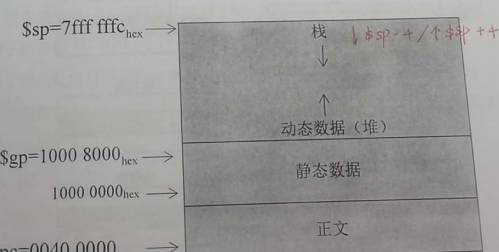
jr $ra

返回到$ra，过程可以访问$v_0$ ~ v_1的结果

总结一下，就是调用者调用程序，把参数保存到被调用者可以访问的寄存器里，被调用者来执行运算并将运算结果保存到调用者可以访问的寄存器里，再用 jr $ra 将控制权返回给调用者。

其中，a_0 ~ a_3 就是用来传递参数的 4 个寄存器，v_0 ~ v_1 是用于返回值的两个值寄存器，$ra 是用于返回起始点的返回地址寄存器。MIPS 的这种寄存器约定，也是加速大概率事件的体现。

六、堆、栈数据空间

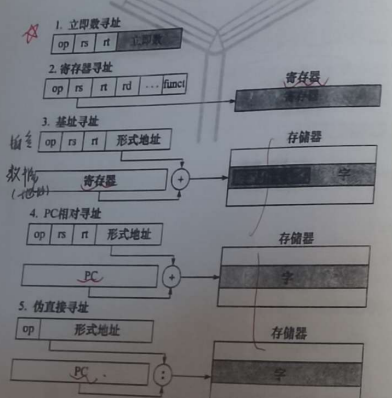


保留

图中，$sp 是栈指针，指向栈顶。从左侧地址大小可以看出，上方是高地址，下方是低地址，而栈是从上往下“长”的。即进栈时，$sp 要减 4，而出栈时，$sp 要加 4。当有数据如过程保存的寄存器和局部变量进栈时，栈向下“生长”，栈顶指针向下移动，这时就需要一个指针来记录栈最初的位置，即帧指针 $fp，它指向该帧的第一个字，$sp 可以通过 $fp 来恢复。

栈指针初始化为 $sp=7fff fffc$_{hex}$，并朝数据段的方向向下增长。静态数据从 1000 0000hex 开始，作为全局指针 $gp，应设置为适当地址以便于访问数据，故初始化为 $gp=1000 8000$_{hex}$，可通过正负 16 位的偏移，访问到 1000 0000H 到 1000 ffffH 之间的数据。程序代码从 pc=0040 0000$_{hex}$ 开始。

七、MIPS 寻址模式（见教材 78 页）



立即数寻址：操作数是立即数（位于指令自身中的常数），可以直接使用。

eg. lui，ori，addi

寄存器寻址：数据存在寄存器当中，访问寄存器即可拿到数据。

eg. R 型指令：add，sub，and

基址寻址：又叫偏移寻址，通过立即数与基址寄存器的值相加得到数据的地址，然后通过数据传输指令进行数据操作。（一般基址寄存器的地址指向内存的某个有固定功能的分区，程序的数据和指令以该地址为起始，进行存取，这种做法可以方便调度内存分配。）

eg. lw，sw

地址 = 基址 Reg 的值 + 常数

PC 相对寻址：把指令中的常数和 PC 中地址相加得到指令地址。这种方式的特点是以 PC 中存储的指令地址为基准，进行一个常数的偏移。

eg. beq，bne

地址 = PC + 常数

伪直接寻址：26 位操作数左移 2 位，与 PC 高 4 位组成新的 32 位地址。（对于跳转指令，由于 32 位里面，高位的 6 位已经被指令操作码占据，那么关于操作数据只能放低位的 26 位，显然不够 32 位地址，只能让这 26 位先左移两位变 28 位，然后将 PC 中的高 4 位充当其高 4 位，以此扩充成 32 位指令地址。）

eg. j，jal

地址 = PC 高 4 位 拼接 指令低 26 位左移两位

立即数寻址：lui，ori，addi

寄存器寻址：add，sub，and

基址（偏移寻址）：lw，sw

PC相对寻址：beq，bne

伪直接寻址：j，jal

再进一步扩展，实际上，寻址分为指令寻址和数据寻址。计算机要执行指令，首先要找到接下来要执行的这条指令，以及它用到的操作数，而寻址的目的就是为了得到指令和操作数。

MIPS 寄存器的 32 位，既可以存储指令地址，又可以存储数据地址，故寄存器寻址是指令寻址和数据寻址共有的方式。此外，数据寻址还有立即数寻址和基址寻址，指令寻址最常用的顺序寻址，还有 PC 相对寻址和伪直接寻址。

【例题 1】

给出产生 32 位常数 0x 2001 4924 并存到$t1 中的 MIPS 代码。

【参考答案】

```
lui $s1,2001H
ori $t1,$s1,4924H
```

【解析】

先用 lui 指令将 32 位常数中的高十六位即 2001H 放到寄存器$s1 中，再 ori 上低十六位 4924H 即可。

【例题 2】

为了缩短指令中某个地址段的位数，有效的方法是：_____

【参考答案】寄存器寻址

【解析】

寄存器寻址中，地址码是寄存器的地址位，由于寄存器数量有限，故寄存器地址位应比较简短，并且访问寄存器的速度很快，因此寄存器寻址很快。

再来分析几个寻址方式，如立即数寻址，地址码是操作数（位于指令自身中的常数），故不必再访问内存去取操作数，速度很快，但是位数固定。

间接寻址：地址码指向主存中的数据，这个数据仍是一个地址，通过这个地址才能找到数据，这可以提高寻址灵活性，扩大寻址范围，但是需要多次访存，速度慢。

八、翻译并执行程序

转换成可执行程序 4 步骤：

Chapter 3 算术运算

本章重点为 3.5 节的浮点表示和浮点加法，要求掌握过程（可能出一个计算题）。理解加减法的溢出并能做出判断，掌握乘除法的硬件执行过程（可能出一个简答题，让画出乘除法器硬件结构图），了解算术精确性（可能出一个判断题）。以下分为五大部分对本章主要内容进行归纳总结。

一、加减法

1. 溢出的概念

二进制数的加减法我们并不陌生，它是计算机科学导论这门课程的重点，相信大家都已掌握，该部分内容主要考察溢出。先明白溢出的定义，类比水缸里的水溢出，是水的体积超过了水缸的体积导致。计算机中的溢出是指运算结果超过了硬件规模的限制，比如字宽只有 32 位时，如果结果超过了 32 位所能表示的数的范围，就会发生溢出（overflow）。

2. 加减法的溢出发生在：

（1）正数 + 正数

（2）负数 + 负数

（3）正数 - 负数

（4）负数 - 正数

举个栗子：5＋4＝9 四位的操作数，最高位为符号位，均为正数，结果却为负数，结果应为0,1001为五位超出了四位所能表示的范围。


```
  0,101
+ 0,100
-------
  1,001
```

3. 溢出的判断

（1）用一位符号位来判断

结果的符号与原操作数符号不同则溢出。

比如两正数相加结果为负，两负数相加结果为正则溢出。

（2）用两位符号位来判断

用两位符号位表示的补码称为变形补码，其两个符号位均参与运算，00 表示正，11 表示负，01 表示正向溢出（结果为正数，但超过了机器所能表示的最大正数），10 表示负向溢出（结果为负数，但是超过了机器所能表示的最小负数）。

举个栗子：两个负数-0.1011 和-0.0111 相加，用双符号位表示如下：

```
  11.0101
+ 11.1001
---------
  10.1110
```

结果的符号位为 10，则产生负向溢出。

注意：MIPS 检测到溢出时会产生异常，即打断正常过程的系统调用，将溢出的指令地址保存到 EPC 寄存器，计算机跳转到一个预定好的地址执行相应异常处理程序，完成后再返回原程序继续执行。当无符号数表示内存地址时，该情况下溢出可忽略。

乘法

见教材 122 页乘法器硬件结构及 123 页乘法器硬件的改进版。

能画出结构图，会做 123 页例题即可。

三、除法

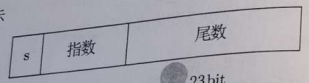
见教材 126 页除法器硬件结构及 127 页改进版。

能画出结构图，会做 127 页例题即可。

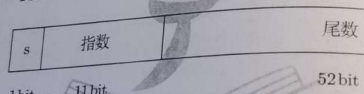
四、浮点运算

1. 浮点表示

单精度：



双精度：



举个栗子：用 IEEE754 单精度表示 -3.3333…

① 规格化：整数部分 3 表示为二进制：0011

小数部分表示为010101…，这个过程看下面的图：

$$
\begin{array}{r}
0.3333\cdots \\
\times \quad 2 \\
\hline
0.6666\cdots \\
\times \quad 2 \\
\hline
1.333\cdots 2 \\
\times \quad 2 \\
\hline
0.666\cdots 4
\end{array}
\qquad 010\cdots 1
$$

② 科学计数

$$-1.1010101\cdots\times 2^{1}(\text{左移1bit})$$

③ 确定符号位，指数和尾数

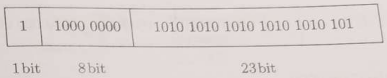
符号位：负数为 1（0 正 1 负）

指数：1＋127＝128，表示为二进制为1000 0000B

尾数：小数点后 23bit，即1010 1010 1010 1010 1010 101

④ 结果

单精度：



即0xC0555555。

2. 浮点运算（用 IEEE754 计算）

$A=-5.25, B=7.125$，求 $A-B$

（1）规格化

A: -5.25 ⟶ -101.01B ⟶（规格化）-1.0101×2^{2}

B:7.125 ⟶ +111.001B ⟶（规格化）$+1.11001\times 2^{2}$

（2）对阶

小阶对大阶，同为二次，阶差为 0，无需对阶

（3）尾数运算

$$
\begin{array}{r}
(-1.0101\text{B})-(+1.11001\text{B})=-11.00011\text{B} \\
1.01010 \\
+\quad 1.11001 \\
\hline
11.00011
\end{array}
$$

（4）规格化

$$-11.00011\text{B}\times 2^{2}=-1.100011\text{B}\times 2^{3}$$

（5）溢出判断

无溢出

（6）舍入判断

无舍入

（7）结果：

单精度：

1	1000 0010	1000 1100 0000 0000 0000 000

即0xC1460000。

五、算术精确性

1. 有关算术精确性的 3 组概念：

（1）保护位：在浮点数中间计算中，在右边多保留的两位中的首位。（用于提高舍入精度）

（2）舍入位：在浮点数中间计算中，在右边多保留的两位中的第二位。（使浮点中间结果满足浮点格式，得到最接近的数）

（3）粘贴位：同保护位和舍入位一样用于舍入的位，当舍入位右边有非零的数据时将其置 1。

不明白的话看这里吧，举个栗子助于理解：

$$2.56_{10}\times10^{0}+2.34_{10}\times10^{2}$$

先对阶，小阶对大阶，故而$2.56_{10}\times10^{0}$转变为$0.0256_{10}\times10^{2}$。

① 不用保护位和舍入位：

$$\begin{array}{r} 0.02_{10} \\ +\ 2.34_{10} \\ \hline 2.36_{10} \end{array}$$

无保护位和舍入位情况下，$0.0256_{10}\times10^{2}$变成$0.02_{10}\times10^{2}$，最后结果为$2.36_{10}\times10^{2}$。

② 使用保护位和舍入位：

$$\begin{array}{r} 2.3400_{10} \\ +\ 0.0256_{10} \\ \hline 2.3656_{10} \end{array}$$

6：舍入位

5：保护位

使用保护位和舍入位情况下，这两位的值若在 0-49 则舍掉，若在 51-99 则向上舍入，如本题为 56，则向上舍入，结果为$2.37_{10}\times10^{2}$

再举个栗子理解粘贴位：

$$5.01_{10}\times10^{-1}+2.34_{10}\times10^{2}$$

在有保护位和舍入位的情况下，先对阶，小阶对大阶，$5.01_{10}\times10^{-1}$变成$0.0050_{10}\times10^{2}$，实则

$$0.00501_{10}\times10^{2}$$

舍入位

由于舍入位之后还有 1，为非零的数，故粘贴位置 1。

$$\begin{array}{r} 0.0050_{10} \\ +\ 2.3400_{10} \\ \hline 2.3450_{10} \end{array}$$

③ 没有粘贴位：

则假设这个数为，然后向最靠近的偶数$2.34500\cdots0$舍入得到 2.34。

④ 使用粘贴位：

粘贴位置 1，即记住这个数是大于$2.34500\cdots0$的，最后一位视为奇数，则加 1，舍入后得到 2.35。

2. IEEE754 的 4 种舍入模式：（详见教材 146 页，感兴趣者可阅读）

（1）总是向上舍入

（2）总是向下舍入

（3）截断舍入

（4）向最靠近的偶数舍入

3. 另外两组概念：

（1）尾数最低位：在实际数和能表达的数之间的有效数最低位上的误差位数。

如果一个数在最低位上少 2，则称少了两个 ulp。

（2）混合乘加：一条浮点指令，其执行一次乘法和一次加法，但只在加法后执行一次舍入。

Chapter 4 处理器

一、单指令的数据通路

1. 三种指令

（1）R 型指令

如 add，sub，and，or，slt。

opcode (6bit)	rs (5bit)	rt (5bit)	rd (5bit)	shamt (5bit)	funct (6bit)

（2）存取指令

有 lw，sw。

opcode (6bit)	rs (5bit)	rt (5bit)	地址 (低 15bit)

lw 与 sw 的理解：lw 是 load word，取；sw 是 store word，是存。

这个存取是以主存为中心来看，存就是把寄存器中的数存到主存中某一地址处，取就是从主存某一地址处取出数据放到寄存器中。

举个栗子：

- lw $s1, 20($s2)　　$s1 = Mem[20 + $s2]
  从Mem地址20 + $s2处读取数据写入$s1
- sw $s1, 20($s2)　　Mem[20 + $s2] = $s1
  $s1寄存器中的数存入Mem的20 + $s2地址处

rs 寄存器作为基址寄存器，取出其中的数与地址相加，得到访存地址。

lw (取) op = $(35)_{10}$ → rt (从Mem取出的数据要存入这个寄存器)

sw (存) op = $(43)_{10}$ → rt (这个寄存器中的数据要存入Mem)

(3) 分支指令

如 beq。

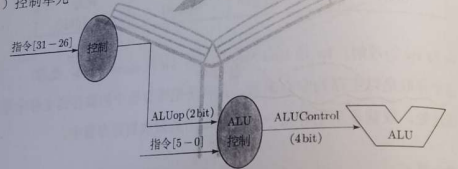
opcode (6bit)	rs (5bit)	rt (5bit)	地址 (低 15bit)

if(rs = rt)

16bit地址 → sign − extend → 左移2bit → add PC + 4 → get 分支目标地址

2. 数据通路图

(1) 控制单元

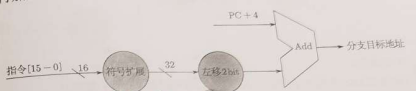


优化之处表现在：通过多级译码减少主控制单元的规模，通过使用多个小控制单元来提高控制单元的速度。

(2) 分支（有条件 eg. beq，bne）

① 比较 rs 与 rt，得到 ALU zero && Branch，即零标志和分支控制信号的与值。与值为 1，则分支成功，下一条指令跳转到分支目标地址；否则分支失败，下一条指令地址仍为 PC+4。

② 分支指令 16bit 立即数经符号扩展后得 32bit，再左移两位变成字寻址，再加上 PC+4 即可得到分支目标地址。

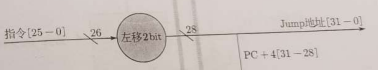


(3) 跳转（无条件 eg. J）

① 取 PC+4 的高 4bit；

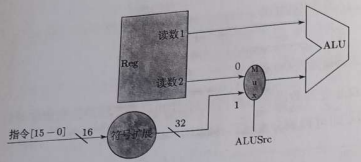
② J 指令的低 26bit 左移 2bit 作为低 28bit；

③ 将步骤①与步骤②所得的高 4bit 与低 28bit 拼接成 32bit 地址，即跳转目标地址。



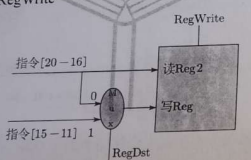
(4) 控制信号

① ALUsrc



ALUSrc
- 0:读数据2
  (即当ALUSrc控制信号为0时，ALU的第二个操作数来自读的数据2)
- 1:低16bit的符号扩展
  (即当ALUSrc控制信号为1时，ALU的第二个操作数来自低16bit的符号扩展)

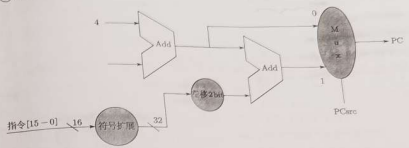
② RegDst && RegWrite



RegDst
- 0:写Reg来自Rs
- 1:写Reg来自Rt

RegWrite
- 0:无
- 1:Reg写使能有效

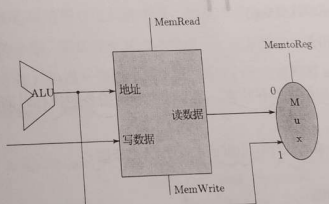
③ PCsrc



PCsrc
- 0:PC = PC + 4
- 1:PC = 分支目标地址 beq

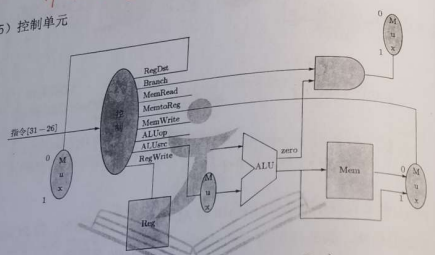
PCsrc 由来自控制单元的 Branch 和 ALU 的 zero 相与得到，除 PCsrc 是间接信号外，其他控制信号只根据指令的 opcode 来确定。

④ MemRead && MemWrite && MemtoReg



MemtoReg
(写入Reg的数据来源)
0:ALU的结果
1:Mem

(5) 控制单元



- 3个控制多选器的1bit信号(RegDst、ALUsrc、MemtoReg)
- 1个决定是否可以转移的1bit信号(Branch)
- 1个ALU的2bit控制信号(ALUop)

二、流水线

性能的提高是通过增加指令的吞吐率，而非减少单条指令的执行时间。

- 指令执行时间(流水线)＝指令执行时间(非流水线)/流水线级数
- 吞吐率＝指令数/时钟周期数
- 加速比＝周期数(非流水线)/周期数(流水线)

1. 流水线图

(1) 无转发

	1	2	3	4	5	6	7	8	9	10	11
lw $s2,20($s1)	IF	ID	EX	MEM	WB						
		stall	stall	stall	stall	stall					
			stall	stall	stall	stall	stall				
sub $s1,$s2,$s3				IF	ID	EX	MEM	WB			
					stall	stall	stall	stall	stall		
							stall	stall	stall	stall	stall
lw $s3,20($s1)								IF	ID	EX	MEM	WB

在无转发的情况下，lw $s2,20($s1)在第 5 个时钟周期才写回，即 $s2 的值此时才更新，由于 sub $s1,$s2,$s3 指令的第一源操作数为$s2，即其在 EX 周期时要用到$s2 的值，因此该指令的 EX 周期最早在第 6 个时钟周期。

同理，lw $s3,20($s1)在 EX 周期时要用到 sub $s1,$s2,$s3 指令的结果，而$s1 的值在第 8 个时钟周期写回，因此该指令的 EX 周期最早在第 9 个时钟周期。

如上图，便形成了含 stall 的流水线图，判断在哪个时钟周期 stall，要看前后指令之间的关系，如后一条指令要在 EX 周期用到前一条指令的结果，那么在无转发的情况下，必须等前一条指令的写回周期执行完，才能开始后一条指令的 EX，那么在此之前都要 stall。

(2) 有转发

	1	2	3	4	5	6	7	8
lw $s2,20($s1)	IF	ID	EX	MEM	WB			
		stall	stall	stall	stall	stall		
sub $s1,$s2,$s3			IF	ID	EX	MEM	WB	
lw $s3,20($s1)				IF	ID	EX	MEM	WB

转发，即旁路，某一指令的计算结果一出来即可送到需要这个结果

的指令继续执行，而不用等这个结果写回之后。

比如，当 lw $s2,20($s1)的结果$s2 是下一条指令 sub $s1,$s2,$s3

EX 周期时要用到的源操作数，有转发后，就可以在 lw 指令 MEM 周

后将更新后的$s2 值送到 sub 指令的 EX 周期。

类似的，sub $s1,$s2,$s3 的结果$s1 是下一条指令 lw $s3,20($s1)

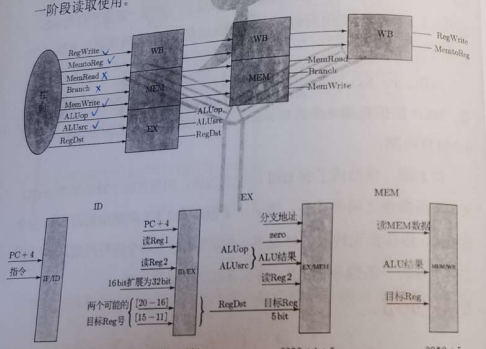
源操作数，这个结果在 EX 周期之后即可得到。

2. 流水线寄存器

为保证流水线正常运转，需要增加部件，即流水线寄存器，其作用为

沿流水线传递所有可能接下来还会用到的信息，并且保存上一阶段数据供

一阶段读取使用。



流水线寄存器需要存的是下一级可能用到或者必定用到的信息，已经用过的下一级用不到就不需要向下一级传递了。

比如，EX 之后才能确定分支，因此 branch & zero 要传递到 EX/MEM。

三、冒险

1. 结构冒险

即硬件不支持。

如：一条指令在访存的时候，另一条指令同时在该存储器预取指令。

2. 数据冒险

即无法提供指令执行需要的数据，而导致指令不能在预定的时钟周期内执行。（一条指令依赖于更早的一条在流水线中的指令）

解决方法：

（1）前推（旁路）

从内部 Reg 或 Mem 中提前取出所需数据，将其直接送到需要它的地方。

（2）流水线阻塞（气泡）

即使旁路，遇到 lw 取数-使用型数据冒险，也不得不阻塞一个时钟周期，除阻塞之外，还可以采用硬件上检测阻塞后，软件上重新安排代码顺序来避免。

通过冒险检测单元来检测 lw 装载指令，从而可以在其后插入阻塞，保持 PC 寄存器和 IF/ID 流水线寄存器内容不变，即 nop（空指令，不产生任何效果），清除所有控制条件（其实只要清除 RegWrite 和 MemWrite 即可）。

3. 控制冒险（分支冒险）

决策依赖于一条指令的执行结果，而指令正在执行中。

解决方法：

（1）假定分支不发生

继续执行顺序的指令流，若分支发生，就丢弃已经读取的指令，按

分支目标执行。（此丢弃必须将 IF、ID、EX 级指令都 flush）

（2）缩短分支的执行时间，提前分支决策。

确定分支目标地址越早，要清除的指令越少。

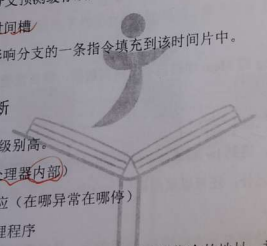
要想提前分支决策，就要提前计算分支目标地址（从 EX 级移到 ID 级），还要提前判断分支条件，即比较 ID 级取的两个数是否相等。着需要额外的旁路和冒险检测单元。

（3）动态分支预测

采用分支预测缓存或分支历史记录表。

（4）分支延迟时间槽

用不影响分支的一条指令填充到该时间片中。



四、异常和中断

异常比中断级别高。

1. 异常（来源：处理器内部）

（1）随时终止响应（在哪异常在哪停）

（2）有固定的处理程序

在异常程序计数器 EPC 中保存出错指令的地址，并把控制权转交给操作系统的特定地址。无符号数表地址的溢出，无需处理。

（3）返回到原来指令去

操作系统处理异常，先明确哪条指令引起异常，即出错指令的地址并保存，以便处理完异常再回到这条指令。设置一个状态寄存器（Cause），记录异常产生的原因，以便控制权转移到由异常原因决定的地址处。

2. 中断（来源：外部）

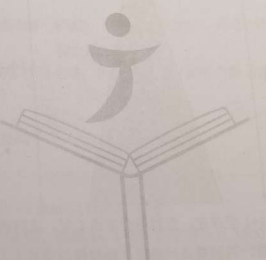
对于多个级别不同的中断请求，优先级高的可以打断优先级低的。中断只能一个个处理，处理该中断时，关中断，但无法关闭高级中断。

（1）先执行完本周期才能响应

（2）要找相应的处理程序

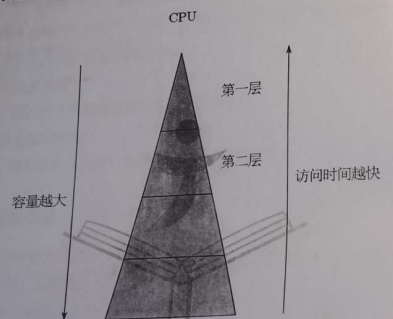
（3）返回到它进入的地址

例如：校长来了，中断讲课，校长走了，继续讲课。



Chapter 5 大容量和高速度：开发存储器层次结构

一、存储器层次结构



除非数据在第 i+1 层存在，否则不会在第 i 层。访问高层，若命中，则可以很快访问到数据，否则要访问底层（速度慢但容量大）。

二、局部性原理

局部性原理源于存储器的层次性结构，分为时间上和空间上。

局部性原理
- 空间局部性：某个数据项被访问，与它地址相邻的数据项也可能将被访问
  如：数组
- 时间局部性：某个数据项被访问，不久的将来这个数据项可能再次被访问
  如：循环结构

番外补充小知识：

- 主存 → DRAM → 成本较低，容量较大，但是速度比SRAM慢
- Cache → SRAM
- 闪存 → flash → 非易失性存储器，损耗均衡
- 磁盘 → 服务器中容量最大但是最慢的一层

RAM
- SRAM：无需刷新，写入快，每位价格高，不易于集成
- DRAM：需要电源不断刷新，写入慢，集成度高，每位价格低

此处电源刷新用到了仲裁电路，用于刷新定时器的刷新请求以及 CPU 访问存储器的请求。

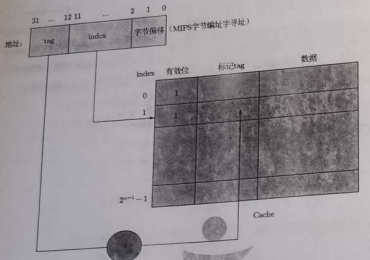
三、Cache 直接映射

直接映射，一个内存块只能映射到 Cache 中唯一一个明确位置，即（块地址） mod （Cache 块数）。

如 Cache 有 64 块，每块 16 字节，则字节地址 1200 映射到：

块地址 = 1200 / 16 = 75

75 mod 64 = 11



从 Cache 左侧 index 的值，从 0 到 2^{n-1}-1，则对于大小为 2^n 块的 Cache，其 index 需要 n 位，index 在 Cache 中不需要存储空间，就像数组的下标一样不需要存储。

如果一个块大小为 2^m 个字，则块内寻址为 m+2，其中 m 位用于索引 Cache 块中 2^m 个字中的哪一个字，2 位用于字节偏移。

get 了 Cache 的 index 和块内偏移的位数，MIPS32 位余下的位数即 tag 位数。

举个栗子：

Cache 有 1024 个字，一个块大小为 1 个字。

【分析】

1024 块 $\log_2 1024 = 10$ => index

$\log_2 4 = 2$ => offset

$32 - 10 - 2 = 20$

Cache有1024个字 = 2^{10}个字 = $2^{10} * 4B$ = 4KB（一个字4Byte）

一个块大小为1个字 -> 块内偏移 = 0 + 2 = 2bit

Cache有1024个字，一个块大小为1个字 -> 2^{10}块 -> index：10bit

余下32 − 10 − 2 = 20bit用作tag标记。

若有效位为1，tag一致，则hit，相应的字提供给处理器；

否则miss，去下一层中寻找数据

补充小知识：

Cache所需总位数 = 每一块的倍数 * 块数

= 2^n * (块大小 + tag + 1) bit

Cache有效率 = $\frac{\text{数据容量}}{\text{总容量}} * 100\%$

四、Cache 缺失

（1）指令 Cache 缺失

① 当前 PC-4 送到存储器中

② 通知主存执行一次读操作，等待主存访问完

③ 写 Cache，把从主存取回的数写入 Cache 中，将地址高位写入 tag，置有效位为 1

④ 重新取指

（2）数据 Cache 缺失

处理器阻塞，直到从 Mem 取回数才响应。

五、写操作

1. 写直达

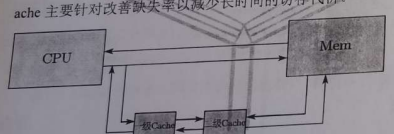
同时更新 Cache 和下一存储层次，保持一致性，但是写主存太慢。其解决方法是写缓冲，即把数据写入 Cache 和写缓冲，处理器继续执行，写缓冲再写主存，写主存完成后再 free buffer。

2. 写回法

新值只写入 Cache 中，只有当修改过的块被替换时才需要写到较低层存储器结构。

六、多级 Cache

一级 Cache 更关注命中时间，二级 Cache 更关注命中率。更确切地说，一级 Cache 致力于减少命中时间来获得较短时钟周期或较少流水级，二级 Cache 主要针对改善缺失率以减少长时间的访存代价。



对两级 Cache：

总的CPI＝基本CPI
＋一级Cache中每条指令阻塞的时钟周期
＋二级Cache中每条指令阻塞的时钟周期

如果一级 Cache 缺失，就去访问二级 Cache，二级 Cache 再缺失，再去访问主存。

因此，

一级Cache阻塞的时钟周期＝访问二级Cache需要的时钟周期

二级Cache阻塞的时钟周期＝访问主存需要的时钟周期

如：

时钟频率为4GHz，则时钟周期为 $\frac{1}{4\text{GHz}} = 0.25\text{ns}$

访问二级Cache时间为5ns，则访问二级Cache需要 $\frac{5\text{ns}}{0.25\text{ns}} = 20$个时钟周期

则：20×一级Cache的缺失率＝一级Cache阻塞的时钟周期

七、映射

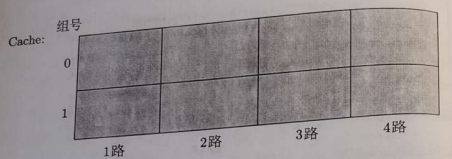
直接映射：位置固定，冲突率高，地址变换速度快，
适于Cache容量大

全相联：tag太长，Cache利用率高，电路复杂成本高
适于Cache容量小

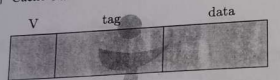
n路组相联：组间直接映射，组内全相联
（直接映射与全相联的折中）
相联度n，即每组 n 块，总块数 / n即组数

如：8 块的 Cache，4 路组相联，则可以得到组数为 8 / 4 = 2 组，每组 4 块。

画图如下：



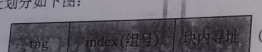
其中每一个 Cache 块如下：



(块地址) mod (组数) = 哪一组

找到组后，在组内的几个块中随意放

地址划分如下图：

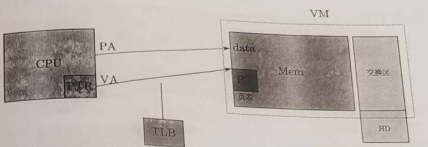


(类似直接映射)

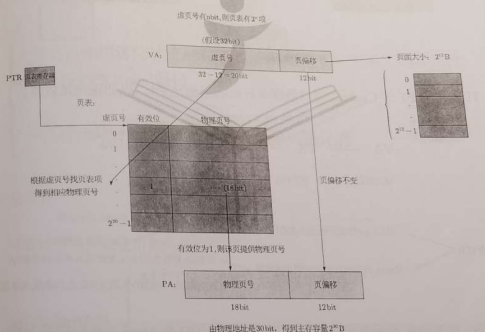
替换时有替换规则：最常见的是 LRU 替换规则，即最近最少使用原则，替换最长时间没用过的块。

八、虚拟内存和页表

虚拟内存是为了解决容量问题，同时相对提高了速度，原理是将主存作为辅助存储器的 Cache，把外存的一部分（交换区）拿出来与 Mem 统一编址成虚拟地址 VA，成为一个更大的虚拟内存 VM。



CPU 里有 PTR（页表寄存器），指向 VM 里的 PT（页表）的首地址，可以理解为指针。

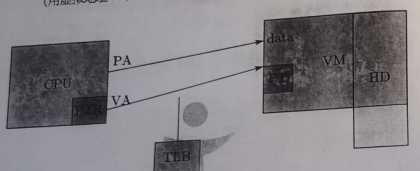


九、快表 TLB

一次访问数据要访问两次Mem(慢) -> 快表TLB来解决这个问题

一次由PTR找页表
(用虚拟地址VA)

一次由PA找数据
(用物理地址PA)



TLB 作为 PT 的 Cache，存放最近访问的物理地址 PA，以加速地址转换。

相联映射方式划分PA找Cache

VA → PA → Cache → 主存Mem

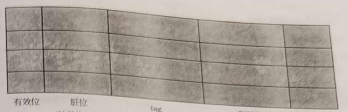
通过查表(先查TLB，再查PT)

先查TLB
- 一旦hit,get物理页号,就不再查PT,直接拿PA再访问Cache
  - if miss,再去主存
  - if hit,直接拿数据
- 一旦miss,再去查页表,同时流水线阻塞
  - 页表hit,则数据在Mem,更新TLB,再去TLB拿PA
  - 若页表miss,则在HD里,发生页缺失,即异常,而非仅阻

注：这里页缺失是异常而不仅是阻塞，因为从 HD 到 Mem 交换数据是上百万周期。

TLB 命中与 Cache 命中无关。

TLB:



十、可靠性可信性

1. AMAT

- 平均存储器访问时间AMAT = 命中时间 + 缺失率 × 确实代价
  (用于检测Cache设计)
- 可信存储器（冗余技术），可信是不可量化的

2. 可靠性

可靠性是从开始使用到失效的时间间隔。

可靠性度量
- MTTF：平均无故障时间
- AFR：年失效率 = $\frac{\text{一年时间}}{\text{MTTF}}$

3. MTTR 与 MTBF

- MTTR(维修平均时间)
- 失效间隔平均时间MTBF = MTTF + MTTR
- 可用性 = $\frac{\text{MTTF}}{\text{MTTF}+\text{MTTR}}$，每年中可用性可以用9的数量衡量，9越多，维修时间越短

4. 提高可用性

- 增加MTTF
  - 故障避免技术：合理构建系统来避免故障
  - 故障容忍技术：冗余措施，保证系统故障时仍能正常工作
  - 故障预报技术：预测故障，在器件失效前替换
- 减少MTTR
  - 故障检测
  - 诊断
  - 修复

2018-2019 学年第 1 学期

《计算机组成原理 A/B》期末考试试题(A 卷)

- 请将答案写在答题纸上，写明题号，不必抄题，字迹工整、清晰；
- 请在答题纸和试题纸上都写上你的班级，学号和姓名，交卷时请将试题纸、答题纸和草纸一并交上来。

一、选择题（10 分，每小题 2 分）

1、下面不属于计算机系统结构中的 8 个伟大思想的是（ ）：
A．通过流水线提高性能　　B．采用多核程序设计
C．采用二进制表示　　D．高级程序语言不必关心底层硬件细节

2．关于硬盘与主存，一般情况下错误的描述是[ ]。
A．主存速度快　　B．磁盘容量更大
C．硬盘价格更低　　D．都是存储介质

3．下面属于非易失性存储器的是（）：
A．SRAM　　B．DRAM
C．CD-ROM　　D．Cache

4．现有 32 位单精度 IEEE754 格式的浮点数 7F800000H（H 表示 16 进制数），请问其含义是[ ]。
A．正无穷　　B．0
C．1×2^{127}　　D．1×2^{128}

5．CPU 能够理解的命令是[ ]。
A．C 语言　　B．高级语言
C．汇编语言　　D．指令

二、简答题（15 分，每小题 3 分）

1、解释保护位、舍入位、粘贴位的作用。