计算机组成原理笔记

1 进制

十进制->二进制：整数部分循环->（求余，num/2）；小数部分循环->（num×2看是否大于1）

二进制->十进制：2的n次幂的和

八进制<-->二进制<-->十六进制（很简单）

2 实数编码

原码：第1位表示符号，后面表示数字

补码：正数和原来相同；负数->绝对值原码取反+1（？补码设计的初衷就是让1 + (-1) = 0，因为加法的实现更加简单，这是原码是无法得到的。补码利用溢出，将相加的结果取模。1 + (-1) = 2^n，2^n % 2^n = 0）

移码：移码 = 偏置值 +/- 绝对值（偏置值未规定的情况下，一般为2^{n-1}；负数可以看成除了第1位，其他位取反+1，正数直接在最高位+1）

范围：

- 原码：$-(2^{n-1}-1) \sim 2^{n-1}-1$（1111 1111 ~ 1000 0000/0000 0000 ~ 0111 1111）
- 补码：$-2^{n-1} \sim 2^{n-1}-1$（1000 0000原本是0，但是变成了-128）
- 移码：$-2^{n-1} \sim 2^{n-1}-1$（原本是0 ~ 2^n-1，但是减去了偏置值2^{n-1}）

3 小数编码

> 参考博客：*IEEE754 浮点数：简读+案例=秒懂*

IEEE 754 浮点数表示方法。

符号1位 + 次幂8位 + 信息23位 = 32位

- 符号：1代表是负，0表示正
- 次幂：使用移码表示，偏置值为127（一定要注意是**127**，一般情况下移码应该为2^{n-1}，但是127是$2^{n-1}-1$，具体原因我不知道；在移码->真值时，使用“-128+1”获取值；真值->移码时，使用“+128-1”获取值）
- 信息：规定小数有效部分转化为1.xxxx的形式，所以可以省略开头的1，不会放入数据，称之为“隐藏位”。

4 编码之间的转换

有符号和无符号表示方式都为补码

高精度->低精度，取机器码后一部分

相同精度，机器码不改变

short->int，补符号位（因为这样做不会失真；一定要注意，这个经常考！）

int->float，机器码变化，尽可能地转为浮点数表示

float->int，机器码变化，舍去小数位

5 实数之间的运算

> 参考博客：关于标志信息*ZF、OF、SF、CF*的理解

ZF（Zero Flag，零标志）：使用0和每位求或，结果再取反。表示是否为零。

SF（Sign Flag，符号标志）：表示带符号整数加减运算结果的符号位，因此直接取结果的最高位作为SF。（无符号就不说了）

OF（Overflow Flag，溢出标志）：OF是判断带符号数是否溢出的，对于无符号数OF不能用作溢出的判断，因为如果两个高位为0的数相加，得到的结果高位为1，明显对于无符号数这种情况是可以成立的，但是对于带符号数，证明其得到的结果是负数，由两个正数做和还得到了一个负数结果明显是溢出。所以在OF为溢出的情况下，对于无符号数是不能由此就判断无符号数的结果是否溢出的。

CF（Carry Flag，进/借位标志）：CF只能判断无符号数是否溢出（溢出为1，反之为0），而不能判断带符号数的溢出情况。

下面是具体的判断是否溢出的总结：

- 有符号：$OF = C_n \wedge C_{n-1}$（C_n 表示最高位进位情况，C_{n-1}表示第2位进位情况）
- 无符号：$CF = C_{out} \wedge C_{in}$（$C_{out}$代表最高位进位情况，$C_{in}$代表是否为减法，只有加法进位才会溢出）

补充：异或运算如下：$A \wedge B = (A \& !B) | (!A \& B)$，总结：AB相同就为0，不相同就为1（异或还有“传递性”，这个点很有趣）

6 寻址方式

寻址方式：指令存在值H1，H1地址的值为H2，H2地址的值位H3。（操作数：得到的值；有效地址：操作数所在的地址）

- 立即寻址：操作数H1
- 直接寻址：有效地址H1，操作数H2
- 间接寻址：有效地址H2，操作数H3

- 相对寻址：有效地址：当前PC+1+H1（注意：其中的**“+1”**，代表加一个命令的大小单位，因为PC被设定为“读取这个命令后，会自动移到下一个命令”）
- 基址寻址：有效地址：基址寄存器数值+H1
- 变址寻址：有效地址：变址寄存器数值+H1

（部分没有给出操作数，只要知道了地址，对应的值就是操作数）

“基址寻址”和“变址寻址”的区别：前者寄存器内容不变，形式地址改变，用于操作系统重定位程序；后者形式地址不变，寄存器改变，用于遍历数组。

7 存放模式

小端模式：从低位开始，存放至内存

大端模式：从高位开始，存放至内存

例如12345678H，小端模式，存放地址从小到大：78 56 34 12。

注意：低位就是低位，不要因为数字小或者从左往右读把**1**（左边）当作低位了。

8 J指令

j指令：j target address其中PC寄存器地址为H1，指令中存储了一个26位立即数。

目标地址为：

1. 26位立即数前后分别加上00。
2. 取PC寄存器地址H1的前四位添加到立即数前。

（具体原因我其实也不清楚，我猜是因为操作数需要6位，所以剩余26位只能这样定义了；其实还有说，因为指令大小是4位的，所以指令的地址是4的倍数，后面两位自然没有作用）

9 MIPS汇编

MIPS汇编命令：

命令	意义	**operation**	**function**
add $1, $2, $3	$1 = $2 + $3	000000	10 0000
addi $1, $2, 25	$1 = $2 + 25	000000	不重要
sub $1, $2, $3	$1 = $2 – $3	000000	10 0010
beq $1, $2, 25	if ($1 == $2) goto PC + (1+25)×4	不重要	不重要
beq $1, $2, L	if ($1 == $2) goto L	不重要	不重要
j L	goto L	不重要	不重要

R型的MIPS机器码：000000 00000 00000 00000 00000 000000（add和sub都是R型）

值得区分一下：“add rd, rs, rt”顺序是rd, rs, rt，但是机器码是rs, rt, td。rs是先读取的。其中rs为Register Source；rt为Register Target；rd为Register Destination，顾名思义知道为什么这样设计了：“rd = rs + rt”的本质上就是“rs + rt”先得到结果，然后“rd = the result”赋给对应的地址值，这样设计可以让计算机顺序读取。

值得一提的是：类似“addi $1, $2, 25”“beq $1, $2, 25”这种存放了立即数的命令，立即数都是以“补码”的形式，所以可以进行指令前进/倒退操作。

对于“a = 1”这种语句的MIPS命令则是“add a, $0, 1”（$0在寄存器中指定了必须为0）

对于if-then语句，例如：

```
if (a == b)
    b = 1;
else
    a = 1;
```

其汇编命令为：

```
    beq a, b, lalala
    add b, $0, 1
    j EXIT
lalala:
    add a, $0, 1
EXIT:
```

---

10 缓存

> 参考博客：彻底搞懂*cache*主存映射以及*cache*容量的计算

1. 为什么需要缓存？（这部分我没学）

在计算机系统中，CPU执行指令时需要从主存中读取数据和指令，而主存的访问速度相对较慢。为了弥补这一速度差异，引入了缓存作为一种介于CPU和主存之间的高速存储器。当CPU需要访问数据或指令时，首先检查缓存中是否存在所需的内容。如果缓存中有命中（hit），则直接从缓存中读取，避免了对主存的访问。如果缓存中没有命中（miss），则从主存中加载数据到缓存，并且可能替换缓存中的一部分内容。

2. 为什么缓存要分块？

因为内存也是分块的。内存分块是因为时间局部性原理和空间局部性原理。

时间局部性原理：某一个数据被访问时，未来很有可能被再次访问。

空间局部性原理：某一个数据被访问时，它周围的数据在未来很有可能会被访问。

3. 全相关映射和直接映射有什么区别？

全相联映射（Fully Associative Mapping）和直接映射（Direct Mapping）是两种不同的缓存映射方式，它们在如何将主存中的数据映射到缓存中有很大的区别。

- 在全相联映射中，一个主存块可以映射到任意一个缓存块，减少了冲突，提高了灵活性，但比较复杂的映射算法可能会导致更高的硬件成本。
- 在直接映射中，一个主存块只能映射到一个特定的缓存块，因此可能会出现不同主存块映射到同一组的情况，导致冲突（冲突是指两个不同的主存块映射到同一组的缓存行）。

4. 总结

主存：就是内存，具体的数据存储是按照字节为单位存储的，数据存储的地址称为“主存地址”。

缓存：主要将具体数据缓存到，方便CPU快速访问。缓存数据时，先按照“主存地址”，结合“映射规则”，得到“缓存地址”，然后在缓存中的“缓存地址”缓存数据。缓存的存储单位为“块”。

映射规则：将内存后几位设置为块内偏移（offset），前几位称为“主存块号”。

- 全相关联映射：“主存块号”前几位->“缓存地址”
- 直接映射：“主存块号”后几位->“缓存地址”
- 组相联映射：将缓存地址分组；“主存块号”后几位->“缓存组号”

值得一提：“直接映射”考得最多，这里说一下具体的每部分的意义：Tag（标志，其实没意义）、Cache（对应的Cache的槽号）、Offset（块内位置）。

X 抛弃内容

- 取指令内容（给你一个图，让你分析指令，完全看不懂）
- 中断（其实和操作系统很接近，但是有点复杂，分值很低）