第七章 指令系统

第一次课--课后思考题：10 月 7 日

1、什么是计算机的机器指令？（注意和汇编语言、高级语言等编程语言之间的关系）一般高级语言的一条语句对应若干条机器指令（01 串），一条机器指令对应一条汇编指令。

2、什么是计算机的指令系统（指令集）？它是由哪些因素决定的？指令集设计关系到哪些方面（三个）？

3、CPU 的机器指令中主要包含哪几个部分？分别是什么作用？

4、指令的地址码（操作数）部分在设计时要考虑哪些问题？

5、指令的操作码设计中主要有哪些方案？它们的优缺点？（重点和考点：**等长扩展操作码**技术）

6、可变长度操作码设计中为什么高频指令使用短操作码？

课后作业: 7.11、7.12、7.14、7.15、7.16

第二次课--课后思考题：10 月 12 日

1、 理解**指令长度**与**操作码**和**操作数**等都相关，现实中指令长度一般是字节的整数倍。

2、 理解硬件的**数据表示**关注的重点什么？（注意和软件的数据类型及数据结构相结合）

3、 计算机中数据分类：**数值型**和**非数值型**，数值型要掌握十进制和二进制数值间的转换，二进制和十六进制之间的转换，掌握常用 BCD 码和十进制 0-9 之间的关系；非数值型数据又分逻辑数据和字符数据等。这部分是基础，需要自行复习掌握。

4、 整数表示：n 位无符号整数的数据表示范围？有符号数表示范围？掌握有符号整数的补码表示，给出真值能得到对应补码，给了补码能得到对应的真值。

5、 有符号数采用补码表示有哪 3 个好处？说明一下：书上有小数补码的例题，这部分可以不用看，因为小数在计算机里是以浮点数形式存放的，补码都是针对有符号整数而言，不讨论小数的补码。

6、 什么是零扩展，什么是符号位扩展？

7、 为什么引入移码（增码）？它和补码的关系？

例 2.29 以下是一个 C 语言程序，用来计算一个数组 a 中每个元素的和。当参数 len 为 0 时，返回值应该是 0，但是在机器上执行时，却发生了存储器访问异常。请问这是什么原因造成的，并说明程序应该如何修改。

```
1  float sum_array(int a[], unsigned len)
2  {
3      int i, sum = 0;
4
5      for(i = 0; i <= len-1; i++)
6          sum += a[i];
7
8      return sum;
9  }
```

第三次课--课后思考题：10 月 19 日

这部分涉及一些规范或标准的定义，学习的思路是首先理解该知识点是为了解决什么问题？这个问题让你解决你有什么方案？再看看目前业界是怎么解决的？补码表示和 IEEE754 都是这个思路。

1、 实数表示中，什么是定点表示？什么是浮点表示？体会引入浮点表示的优点，将**数据表示范围（阶码）**和**精度（尾数）**分开表示的好处。阶码和尾数常用补码或移码表示，具体看上下文约定，这里没有对错，只要统一即可。

   例如现在大多数计算机采用的 **IEEE754 标准**中，尾数就采用的是**原码**，阶码采用的是**移码（偏置是 127 或 1023（单双精度的区别））**，一个浮点数其实是由两个定点数来描述的。

2、 掌握 IEEE754 浮点表示标准（大多数计算机浮点表示都采用该标准），其中尾数部分巧妙的省了一位（怎么省的？），单精度偏置为什么选择 127，选择其他值可以吗？

（假定 int 为 32 位）。

① 从 int 转换为 float 时，不会发生溢出，但有效数字可能被舍去。

② 从 int 或 float 转换为 double 时，因为 double 的有效位数更多，故能保留精确值。

③ 从 double 转换为 float 时，因为 float 表示范围更小，故可能发生溢出，此外，由于有效位数变少，故数据可能被舍入。

④ 从 float 或 double 转换为 int 时，因为 int 没有小数部分，所以数据可能会向 0 方向被截断。例如，1.9999 被转换为 1，-1.9999 被转换为 -1。此外，因为 int 的表示范围更小，故可能发生溢出。将大的浮点数转换为整数可能会导致程序错误，这在历史上曾经有过惨痛的教训。

1996 年 6 月 4 日，Ariane 5 火箭初次航行，在发射仅仅 37 秒后，偏离了飞行路线，然后解体爆炸，火箭上载有价值 5 亿美元的通信卫星。调查发现，原因是控制惯性导航系统的计算机向控制引擎喷嘴的计算机发送了一个无效数据。它没有发送飞行控制信息，而是发送了一个异常诊断位模式数据，表明在将一个 64 位浮点数转换为 16 位带符号整数时，产生了溢出异常。溢出的值是火箭的水平速率，这比原来的 Ariane 4 火箭所能达到的速率高出了 5 倍。在设计 Ariane 4 火箭软件时，设计者确认水平速率决不会超出一个 16 位的整数，但在设计Ariane 5 时，他们没有重新检查这部分，而是直接使用了原来的设计。

在不同数据类型之间转换时，往往隐藏着一些不容易被察觉的错误，这种错误有时会带来重大损失，因此，编程时要非常小心。

例 2.25 假定变量 i、f、d 的类型分别是 int、float 和 double，它们可以取除 $+\infty$、$-\infty$ 和 NaN 以外的任意值。请判断下列每个 C 语言关系表达式在 32 位机器上运行时是否永真。

① i == (int)(float) i

② f == (float)(int) f

③ i == (int)(double) i

④ f == (float)(double) f

⑤ d == (float) d

⑥ f == -(-f)

⑦ (d + f) - d == f

解 ① 不是，int 有效位数比 float 多，i 从 int 型转换为 float 型时有效位数可能丢失。

② 不是，float 有小数部分，f 从 float 型转换为 int 型时小数部分可能会丢失。

③ 是，double 比 int 有更大的精度和范围，i 从 int 型转换为 double 型时数值不变。

④ 是，double 比 float 有更大的精度和范围，f 从 float 型转换为 double 型时数值不变。

⑤ 不是，double 比 float 有更大的精度和范围，当 d 从 double 型转换为 float 型时可能丢失有效数字或发生溢出。

⑥ 是，浮点数取负就是简单地将数符取反。

⑦ 不是，例如，当 $d = 1.79 \times 10^{308}$、$f = 1.0$ 时，左边为 0（因为 $d + f$ 时 f 需向 d 对阶，对阶后 f 的尾数有效数位被舍去而变为 0，故 $d + f$ 仍然等于 d，再减去 d 后结果为 0），而右边为 1。 ■

第二章的考点：

1) 会用基数乘除法或降幂法完成十进制与 2（8，16）进制间的转换（注意小数部分可能存在非完全转换问题，如没办法用二进制精确表述十进制 0.32）；
2) 会用直接替换法完成 2 进制与 8，16 进制间的转换（注意首、尾补零位置）；
3) 掌握压缩 BCD 码（主要是 8421 和余三）的数据表示；
4) 掌握原码、反码、补码和移码的表示方式，有符号数的补码表示有什么优点（0 的问题；数据扩展；

加法的一次运行不同解释等）；

5) 会使用 IEEE754 浮点表示方式；

6) 可靠性编码中只要求掌握奇偶校验码。

3、 操作数类型的说明可放在操作码字段或直接以标记符方式跟着操作数，但工业界常用第一种方式（为什么？），书上及课后题目为了好理解，用的是第二种方式。

4、 **寻址方式**是第七章的重点，可分为**指令寻址**（PC 如何更新）和**操作数寻址**（操作数从哪来到哪去）。体会寻址方式越多，用户获取指令和操作数的方式就越灵活，但是会增加 CPU 设计和实现的复杂度（学了第 8,9 章体会会更深刻），所以现在精简指令集 RISC 往往提供的寻址方式很少，如本章最后介绍的 MIPS 只有 4 种寻址方式（其实是 3 种，一种是变形）。

第四次课--课后思考题：10 月 26 日

1. 寻址时为什么大多数要进行**地址变换**（形式地址和有效地址 EA 间的转换方法）？目前学习的寻址方式中哪些不需要进行地址变化？哪些需要变换？需要变换时，他们的具体变换方式是什么（能结合图示理解变换过程）？
2. 指令寻址中书上仅介绍了**相对寻址**（是基址寻址的一个例子），但大多数 CPU 还会提供**绝对寻址**（对应数据寻址中的直接寻址），就如课堂上介绍的中断处理程序的调用往往采用的就是绝对寻址，还有如第六章 jump 1000；
3. 体会**立即数寻址**和**寄存器（直接）寻址**在获取操作数时为什么快？
4. 对照图示描述目前学习的**存储器寻址**中**直接寻址**、**间接寻址**、**寄存器间接寻址**的形式地址到 EA 地址转换过程。如果从快到慢排序，应该怎么排？加上上面立即数寻址和寄存器寻址，又怎么排？
5. 从**寻址范围**看，上面不同的寻址方式又如何排序？
6. 为什么**变址寻址**是站在“用户”角度？**基址寻址**是站在“系统”角度（这里强调系统,就是对”上层用户是不可操作的”）？比较两者的不同，教材里的**基址寻址**和**相对寻址**的关系是什么？
7. 在介绍的寻址方式中，指令取指后，哪些寻址方式不需要再访问内存了？哪些需要访问寄存器？哪些需要访问一次内存？哪些需要访问不止一次内存？（访问内存次数会影响操作数获取速度）
8. 寻址方式在操作码或操作数中加标注这两种不同的确定方法有什么优缺点？以后看到指令，思考下该指令中有哪些操作数？哪些是**目的操作数**？哪些是**源操作数**？他们分别采用什么寻址方式？

第七章总作业：7.10 –7.17（提交时间待定）

答疑：例题中相对寻址为什么采用符号位扩展？

答：因为采用**符号位扩展**相当于把偏移量做有符号数，这样基于当前 PC，指令可以向前或向后跳转，如果采用**零扩展**相当于把偏移量看成无符号数，指令只能单向跳转。

第五次课--课后思考题：11 月 2 日

1. 指令集设计中一般会包含哪些功能分类？为什么 IO 指令不是必需的（取决于 IO 地址空间的两种处理方式：**IO 映像**与**存储器映像**）？以后看到指令，想想它的功能是什么（数据传送&运算&程序控制类）？
2. 指令集设计中依据什么来确定该功能是由硬件完成还是软件完成（既他们的优缺点）？
3. 指令集设计的标准有哪些？（了解实际中往往没办法全兼顾，要依据设计需求有所取舍）
4. 等长指令封装的优点是什么？CISC 为什么不采用等长指令封装？
5. 了解**复杂指令集 CISC** 和**精简指令集 RISC** 两种不同指令集设计思路，比较两者主要区别，他们分别适合于什么应用？
6. 什么是 **Load-store 结构**？RISC 为什么一般采用该结构？

7. 本课程不需要了解流水线细节，只要知道它是为了提高资源利用率和减少指令平均执行周期的就可以。该技术细节会在后续课程学习。
8. 什么是**硬连逻辑**实现？为什么大多数 RISC 采用该方法实现部件间连接？

学习“7.7 指令系统实例---MIPS 指令系统”时思考：

1、 MIPS 处理器指令是定长的吗？如果是定长，是多长？操作码部分是定长的吗？理论上 MIPS 最多有多少条指令？
2、 MIPS 指令集有 IO 指令（输入/输出指令）吗？为什么？那些是数据传送类指令？哪些是运算类指令？哪些是控制类指令？
3、 熟悉 MIPS 的 3 类指令封装，R 类、I 类与 J 类，尤其是前两种，后面章节在介绍 CPU 设计时，我们用到了这两种。
4、 MIPS 的指令功能和指令封装是一一对应的吗？数据传送类指令都有哪些封装格式？运算类指令都有哪些封装格式？控制类指令都有哪些封装格式？
5、 MIPS 的指令寻址方式有哪些？看表 7.5 中哪些是指令的相对寻址？哪些是绝对寻址？
6、 MIPS 的数据寻址方式有哪些？对照表 7.3-7.5，能说出源操作数与目的操作数各自的寻址方式。
7、 MIPS 的数据寻址方式是单独编码还是在操作码中体现的？

7.7 的 MIPS 指令实例是我们后面模型机实现的基础（我们要设计一个只有 8 条指令的 MIPS 子集），该部分在后面学实习要经常返回头看看；

1） 了解 MIPS64 的寄存器资源（64 位）、数据表示（重点记住：字是 32 位）。
2） 什么是**零扩展**和**符号位扩展**？
3） MIPS 的寻址方式有哪几种？了解其寻址方式编码在操作码中，了解 MIPS 的内存是**按字节编址**（及每个字节分配一个地址，地址是 64 位）的。
4） 什么是存储访问的**边界对齐**，其优缺点是什么？（体会**用空间换时间**）
5） 熟知 MIPS 的三种**指令封装格式**：I 类、R 类、J 类，重点是前两种（因为后面原型机设计会用到）；
6） 理解指令功能和指令封装是两个概念!!! 两者不是一一对应的（例如运算指令可以采用 R 类封装，也可以采用 I 类封装）
7） 表 7.3-7.5 的例子看明白。（注意 MIPS 采用的是**大端字节顺序**）
说明：关于表 7.3 中装载半字的例子，注意它隐含了**大端字节顺序**（big endian），其实只要是多字节数据都存在字节顺序问题，所以表中“装入双字”应该也按大端顺序存放，但作者应该是考虑内容过于冗余，所以省略了，大端存放时采用符号位扩展，就要找对“符号位”的位置，就如我们黑板中实例，内存地址从[R3]+20 开始由低到高如果存放 0xA1、0x22、0x33、0x44、0x55、0x66、0x77、0x88，如果取半字（16 位）最后 R2（64 位寄存器）中应该是 0xFFFFFFFFFFFFA122，注意这里**符号位**取得是 A 的高有效位“1”，其他的类似。（大多数 RISC 采用大端字节顺序，如 MIPS、ARM（可选）、PowerPC 等，但 x86 采用的是**小端字节顺序**，所以如果是 x86 对多字节的访问，如内存地址开始由低到高存放 0x11、0x22，进行双字节操作时，得到的数据是 0x2211）。

指令举例	指令名称	含 义
LD R2，20(R3)	装入双字	Regs[R2] ←64 Mem[20+Regs[R3]]
LW R2，40(R3)	装入字 符号位扩展	Regs[R2] ←64 (Mem[40+Regs[R3]]31)32 ## Mem[40+Regs[R3]] ## Mem[41+Regs[R3]] ## Mem[42+Regs[R3]] ## Mem[43+Regs[R3]] 大端存放
LB R2，30(R3)	装入字节	Regs[R2] ←64 (Mem[30+Regs[R3]]7)56 ## Mem[30+Regs[R3]] 符号位扩展
LBU R2，40(R3)	装入无符号字节	Regs[R2] ←64 056 ## Mem[40+Regs[R3]] 零扩展
LH R2，30(R3)	装入半字	Regs[R2] ←64 (Mem[30+Regs[R3]]7)48 ## Mem[30+Regs[R3]]## Mem[31+Regs[R3]]

思考题：MIPS 有没有一条指令完成将一个立即数直接赋值给内存某个存储器单元的指令？为什么？

关于 MIPS 程序控制类指令的解释：（书上表 7.5 的修正，修正原因见后面解释部分）

指令举例	指令名称	含义
J name	跳转	$PC_{27..0}$← name<<2
JAL name	跳转并链接	Regs[R31]←PC+8；$PC_{27..0}$←name<<2；
JALR R3	寄存器跳转并链接	Regs[R31]←PC+8；PC← Regs[R3]
JR R5	寄存器跳转	PC← Regs[R5]
BEQZ R4，name	等于零时分支	if(Regs[R4]== 0) PC←PC+4+name<<2； ((PC+4)－2^{17})≤name＜((PC+4)+2^{17})
BNE R3，R4，name	不相等时分支	if(Regs[R3]!= Regs[R4]) PC←PC+4+name<<2； ((PC+4)－2^{17})≤name＜((PC+4)+2^{17})

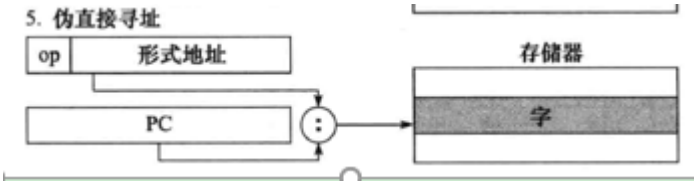
查 MIPS 手册:

Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		$0\ /\ 08_{hex}$

- J name 中位段 name 是 26 位，对这 26 位硬件做了以下处理：

1） 先左移 2 位（地址变成 4B 边界对齐），得到 28 位（末尾两位为 00）；

2） 然后直接替换 PC 的后 28 位吗，如下图，有些资料里叫“伪直接寻址”；



- JAL name 中 PC 地址更新一样，唯一不同是会将 PC+8 保存在 R31 寄存器中，为什么不是 PC+4，注意是 MIPS 是支持流水执行的 CPU，PC+4 处的指令已经被提前放到“分支延迟槽”里了，下学期学习流水线机制时会介绍。
- JALR 指令，手册中没看到该指令
- JR R5 指令，直接用 R5 寄存器的值替换 PC
- BEQZ R4,name（条件转移）

1） 判断 R4==0；如果条件满足执行 2），否则执行下一条指令。

2） 该指令采用 I 类封装，所以 name 位 16 位，先左移 2 位（因为是指令寻址，地址需要 4B 边界对齐），得到 18 位（末尾两位为 00）；

3） 然后对移位后的 18 位做“符号位”扩展变成 64 位；（跳转可前和后，所以偏移量可正可负）

4） 用运算器进行 64 位加运算（因为是 64 位处理器，运算都是 64 位的）：将下一条指令地址 PC+4 与第 3 步移位和符号位扩展后的数相加，和的结果更新 PC 寄存器，完成程序跳转。

- BNE 指令执行类似 BEQZ，只是条件判断取反，这里不赘述。

关于 PC+4 的解释：因为 PC 更新是在取指后就更新的，所以当前指令（PC 所在位置）一完成取指，PC 寄存器的内容就更新为 PC+4 了。

关于左移两位的解释：如果一条指令一个字节，就不存在左移的问题，但是现在一条指令是 4B，存放的时候按照边界对齐放的，所以取的时候就像上面处理的那样，得到的也一定是末尾两位是 00 的一条新指令的起始地址。

关于偏移量（偏移地址）的解释：偏移量本身不是地址，它反映的是两个**内存地址间的距离**。

关于边界对齐的解释：存放是边界对齐的，取也按边界对齐，这样访存速度快。