第七章 指令系统（ISA---软件调用硬件的接口）

& 第二章 数制与编码

第五周--课后思考：第七章 指令系统（1）+ 第二章 数制与编码（1）

第七章 指令系统（1）：

1、 什么是计算机的**机器指令**（01 串）？（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，一条机器指令对应一条汇编指令（例如课堂上的例子）。

2、 什么是计算机的指令系统（**指令集**）？这里要理解指令系统是计算机软件和硬件最底层的**接口（interface）**。指令集设计中考虑哪些因素？--这一章的**核心**

3、 CISC 和 RISC 指令集的特点是什么？各自的优缺点。

4、 CPU 的机器指令中主要包含哪几个部分？分别是什么作用？

5、 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。

6、 指令的地址码（操作数）部分在设计时要考虑哪些问题？

7、 指令的操作码主要对应指令功能，不同功能的操作码是**唯一**确定的，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）

8、 可变长度操作码设计中为什么高频指令使用短操作码？

9、 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，软件层面是对底层数据表示的各种封装和抽象）

10、 操作数类型的说明（也就是数据的寻址方式）可放在操作码字段（方案一），也可以直接以标记符方式跟着操作数（方案二），但 RISC 常用第一种方式，CISC 采用第二种方案，书上及课后题目用的是第二种方式，两种方案的优缺点是什么？

课后作业: 7.11、7.12、7.14、7.15、7.16

第二章 数制与编码（1）：（主要靠自学）

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、移码和 IEEE754 都是这个思路。

1、 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。

2、 任意进制数如何转换为十进制数？

3、 了解计算机中 2、8、16 进制常用描述方式。

第六周--课后思考：第二章 数制与编码（2）

1、 **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**（重点）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。

2、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。

3、什么是**零扩展**，什么是**符号位扩展**？

4、为什么引入移码（增码）？它和补码的关系？

5、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。在字长一定的情况下如何进行取舍？（难两全，不同的浮点数标准的本质差别就在这里）

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。

6、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（其实可以，只是数据表示范围的取舍）？

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。

第七周--课后思考：第七章 指令系统（2）

1、**寻址方式**是第七章的难点也是重点，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）。（书上这部分没有分，理解上有些感念会容易混淆，所以做了分类）

2、指令寻址中书上仅介绍了**相对寻址（也叫指令相对寻址）**，所谓相对就是新地址与当前地址相关，但大多数 CPU 还会提供**绝对寻址**（也叫指令直接寻址），所谓绝对就是新地址与当前地址无关，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、 数据寻址时为什么大多数要进行从形式地址到有效地址 EA 间的**地址变换？**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）

4、 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？

5、 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？在学习寻址方式中，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，目前寄存器和内存访问速度相差**几十甚至上百倍**!）

6、 从**寻址范围**看，上面不同的寻址方式又如何排序？

7、 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？

8、 关于**相对寻址中偏移量**的说明：指令寻址中的相对寻址，由于新指令的地址可前可后，所以偏移量**必须**是可正可负（往后或往前跳），即**偏移量是有符号数**，所以在进行位对齐时采用**符号位扩展**，但数据寻址中的相对寻址（也就是教材里的变址寻址）一般对应数组等结构体的寻址，偏移量**一般**只为正，即偏移量做无符号处理，所以采用**零扩展**，数组首地址对应基址，偏移量对应下标信息，说明一下，这些细节的设定，对不同的具体实现，要看相应手册是如何处理的。

9、 以后看到指令，除了关心该指令功能外，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）

第八周--课后思考：第七章 指令系统（3）

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时间的就可以。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？（第八章我们自己设计 CPU 时也用的该方案）

9、 通过对本章的学习，要理解 ISA 支持的指令功能越多、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 章会更有体会），所以现在精简指令集 RISC 往往提供的尽可能少的指令功能和寻址方式，如本章最后介绍的 MIPS 只有几十条指令（相较 X86 的 2k 多条指令），4 种数据寻址（其实是 3 种，一种是变形）和两种指令寻址。

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？哪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？结合表 7.3-7.5 关注同功能指令的不同寻址方式和不同的指令封装形式。

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？

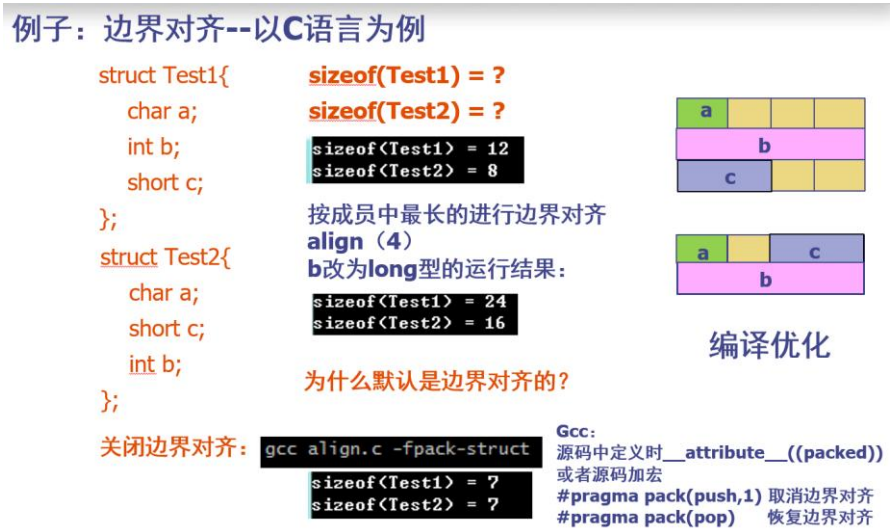
7.7 的 MIPS 指令实例是我们后面模型机实现的基础（我们要设计一个只有 8 条指令的 MIPS 子集），该部分在后面学习时要经常返回头看看；

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 MIPS64 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，如内存地址从[R3]+20 开始由低到高存放 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？

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	$PC_{27\cdots0}$← name<<2
JAL name	跳转并链接	Regs[R31]←PC+8; $PC_{27\cdots0}$←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） Name 在指令列表中其实是一个“符号地址”，例如：

Work： 指令 1

指令 2

……

J work

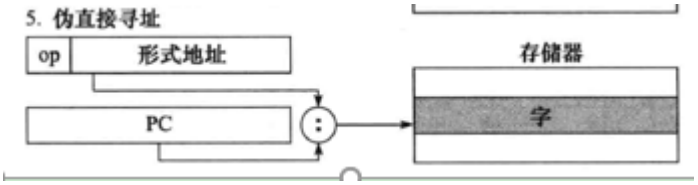
Work 就是符号地址，用过 C 语言 goto 的应该更有体会，汇编时 work 被替换为一个地址了，对应 name 位段。例如下面的反汇编代码：

```
0000000120000060 <main>:
   120000060:   67bdfff0        daddiu  sp,sp,-16
   120000064:   ebbe003f        gssq    ra,s8,0(sp)
   120000068:   03a0f02d        move    s8,sp
   12000006c:   0c000014        jal     120000050 <__export_parasite_head_start>
   120000070:   00000000        nop
```

Jal 采用的就是 J 类封装，后 26 位存放的 0x14 左移 2 位相当于乘 4，0x14*4=0x50 替换 PC 后 28 位，新 PC 地址：0x120000050。

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

3） 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。

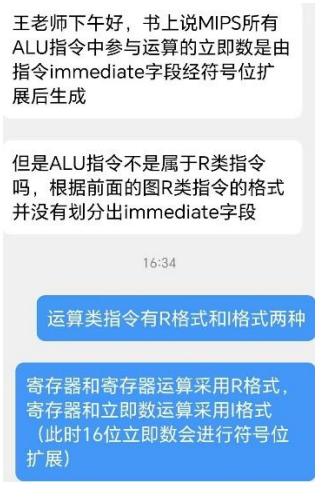
- BNE 指令执行类似 BEQZ，只是条件判断取反，这里不赘述。

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。



提交作业

- 时间：第九周随堂提交
- 内容：

  第一章：1.1（选5个）、1.2、1.3；

  第六章：6.1、6.2；

  第七章：7.10 –7.17