计算机系统基础答案（仅供参考!）

1、进制转换

167　10100111　0xA7

243　11110011　0xF3

170　10101010　0xAA

（二进制每四位对应十六进制一位，从右往左开始，若最后不足四位，则用零补齐四位）

2、计算表达式结果

char　a = 0x96, b = 0xAA;

~a　01101001	**a|b　10111110**	（按位取反、按位或运算）
~b　01010101	**a||b　1**	（按位取反取反、逻辑或运算）
a&b　10000010	**a^b　00111100**	（按位与运算、按位异或运算）
a&&b 1	**!a　0**	（逻辑与运算、逻辑非运算）
-1 < 0u　0(即 false)	**a/b　1**	（**0** 之后加上 **u** 表示这是一个无符号数 **0**，则整个表达式两边的数都会先转换成无符号数再比较，**-1** 对应的无符号数是有符号数最大值加一； **a/b**，先将 **a** 和 **b** 按照补码写出，再按照有符号数的计算规则得到对应的十进制数，除法之后，向零取整）

附对应规则示意图：

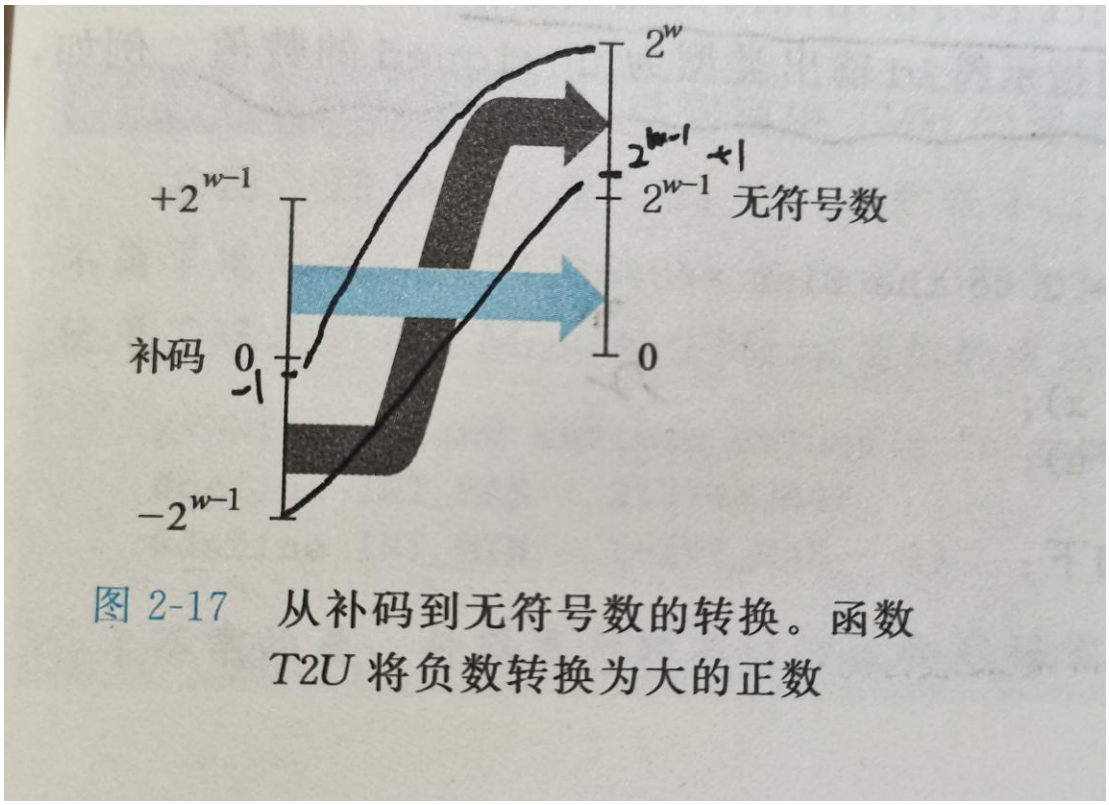


图 2-17　从补码到无符号数的转换。函数 *T2U* 将负数转换为大的正数

3、从左到右依次是　18　20　56　43　（题目已经写出十六进制，无需加上 **0x**）

大端序人看着舒服，小端序人看着不舒服，就是这样。每一个字节填写两位 **16** 进制的位

4、从左到右依次是　36　20　19　56　（每一个字节内顺序不要反着填）

5、从上到下依次是　0x100　0x306　0x80C　0x80C　0x4　0xCCCCCCCC
（mov 指令系列源操作数加上括号时，表示寻址并且从内存中该地址处取得对应的内容，而 lea 指令加上括号时，只是计算出地址的值（有时候计算的甚至不是地址，只是计算一个值），并不访问内存！）

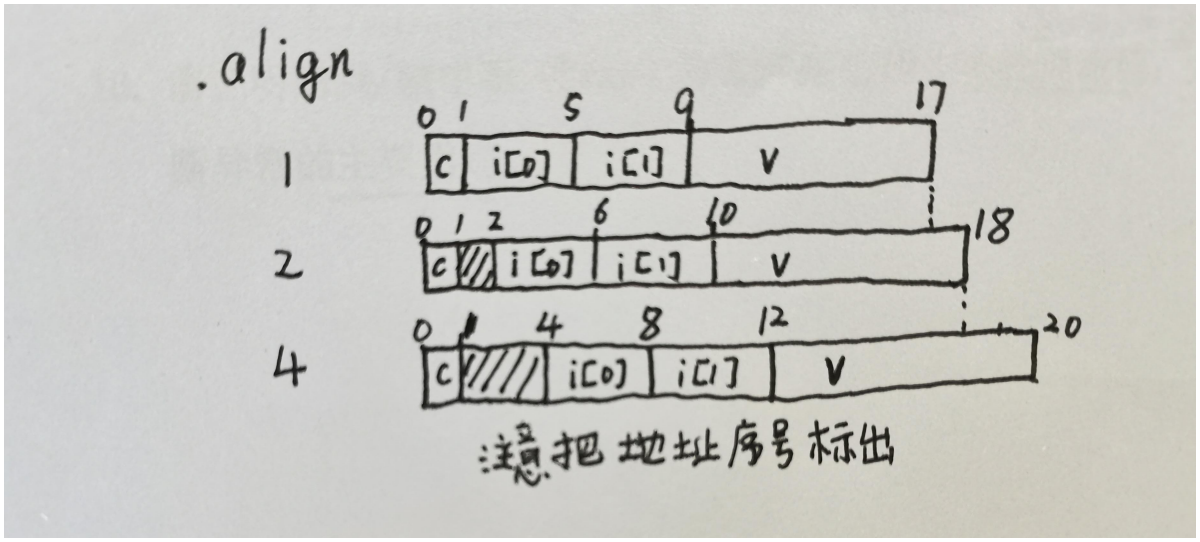
6、第一空：　*xp = t1　第二空：　*yp = t0
（读题，然后把寄存器画出来）

7、第一空：result = x - y　　第二空：result = y - x
（push 指令相当于两条指令：先将%rsp 减 8 留出空间，然后将操作的寄存器的内容填入；cmp　S　D 指令，计算 D-S 的值，然后根据后面 jx 系列指令跳转条件实现跳转，这里 jle 意思是：当 cmp 结果小于等于 0 的时候，跳转到后面的标签.L6 处，即对应 C 语言中的 else 语句；否则继续执行指令直到无条件跳转.L7 处）

8、全局符号：bufp0 , swap　　外部符号：buf[]　　内部符号：bufp1
（全局符号是本模块的非 static 的函数和全局变量；外部符号是本模块中引用来自其他模块的全局符号；内部符号是带有 static 属性的函数和全局变量。这里的指针*符号和函数的（）不用写出，extern 显示地标出对外部符号的引用。注意，模块中的函数中定义的局部变量不参与连接时的符号解析，不能称之为符号）

9、地址从小到大可以表示为　2 0 6 2 1 3 2 1 7 2 2 1
（嵌套数组，参考 P178 页，题目中的表示相当于二维数组 pgh[4][3]，按照行优先原则分配内存地址）

10、大概是这样



11、switch 语句在汇编语言水平上的主要实现思路：C 语言中 switch 语句的实现依赖于跳转表这种数据结构，根据一个整数的索引值从而实现多重分支。汇编

代码中利用之前得到的索引值来执行一个跳转表内的数组引用从而确定跳转指令的目标代码位置。对于重复的情况，跳转表使用相同的代码标号；对于缺失的情况，则是使用默认情况的标号。

12、什么是中断？中断的工作过程？

中断是一种一部类型的异常控制流，是来自处理器外部的 I/O 设备的信号的结果，而不是由执行任意一条专门的指令造成的结果。

工作过程：发生中断时，CPU 执行完当前语句后，注意到中断引脚的电压变高，就从系统总线读取异常号，然后调用适当的中断处理程序（这里包括保护现场、目标跳转、异常处理、恢复现场（如果异常没有被正常处理的话））。在处理程序返回时，将控制返回给下一条指令（即若没有发生中断，控制流在当前指令之后的那条指令）

13、说明 call（过程调用）指令和 ret（过程返回）指令的使用方法和具体工作过程，并说明相关寄存器的变化。

call 指令之后会跟着一个地址(直接或者间接的)，执行时会先将当前 PC 的值(call 指令之后一条指令的地址）压入栈中，然后将 PC 的值更新为 call 之后的地址，接着处理器会从 PC 的地址处的指令开始执行。ret 指令会先从栈中弹出 call 原先压入栈中的地址，然后将该地址赋给 PC，接着 PC 将会从该地址处执行指令（紧跟在原先 call 指令之后的指令）。

寄存器的变化：call 会让%rsp 寄存器减 8 方便返回地址入栈，同时用调用地址更新 PC 寄存器%rip 的值；ret 指令会让%rsp 寄存器加 8 并将弹出的地址返回给 PC 寄存器%rip。

14、什么是程序的局部性原理？

倾向于引用临近与其他最近引用过的数据项的数据项，或者最近最近引用的数据项本身，这种倾向性被称为局部性原理。

局部性原理通常有两种形式，时间局部性和空间局部性。时间局部性良好意味着被引用过一次的内存位置很可能在不久后再次被多次引用；空间局部性良好意味着如果一个内存位置被引用了，其附近的数据很可能不久后被引用。

15、存储器层次的特点：①存储器分层，图中层次更高的存储结构其存储容量更小、速度更快、每字节成本更高；②存储器层次的本质是每一层存储设备都是低一层的缓存，即层次中每一层都缓存来自较低一层的数据。

16、攻击缓冲区溢出漏洞采取的主要手段、从程序设计的角度和编译器的角度讨论防御手段。

①主要攻击手段：利用向缓冲区填充超过其本身容量的数据位数，使得溢出的数据覆盖原先合法数据上。借此可以进一步实现让程序执行原先不愿意执行的函数或者代码。

②程序设计角度：程序员应避免使用可能会造成缓冲区溢出的有风险的函数；

编译器角度：（1）栈随机化：让栈的位置在程序每次运行时都有变化；（2）栈破坏检测：加入栈保护者机制，在栈帧中的任何局部缓冲区与栈状态之间存储一个随机产生的特殊的金丝雀值，在每一次恢复寄存器状态之前和从函数返回

之前，检查金丝雀值，若被改变则程序立即异常中断。（3）限制可执行代码区域：限制哪些内存区域能够存放可执行代码。

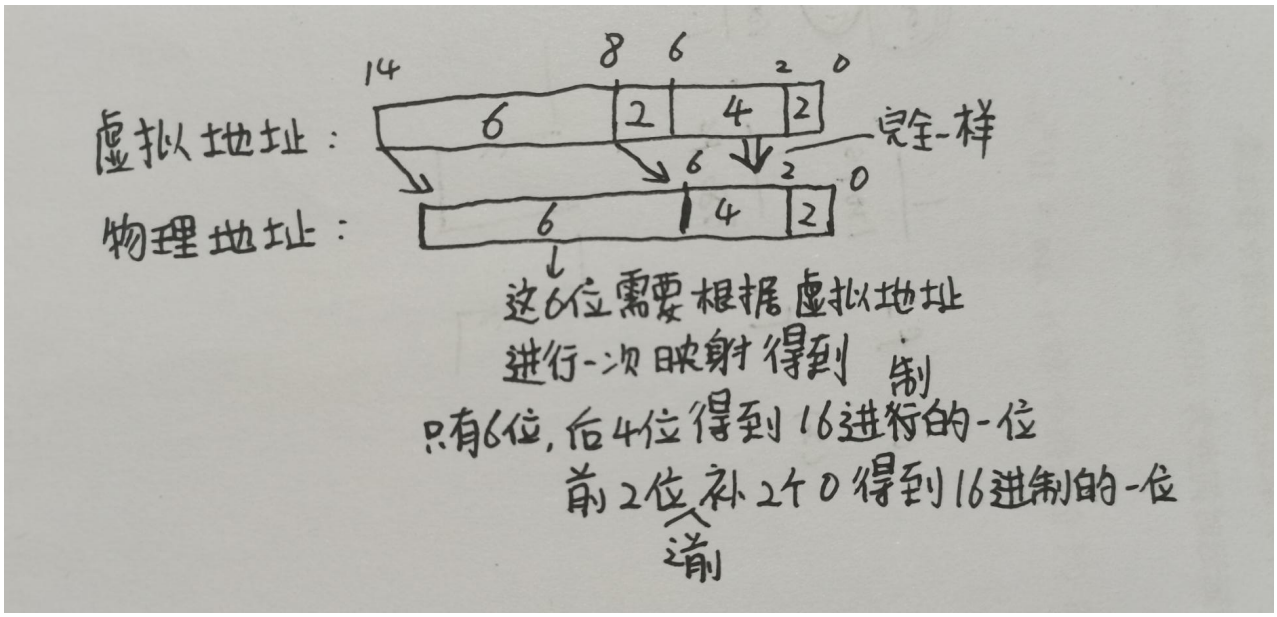
17、（注：首先强弱符号都是全局符号！）
（a）强符号：函数以及已经初始化的全局变量是强符号；未初始化的全局变量是弱符号。
规则：（1）不允许有多个同名的强符号（2）若有一个强符号和多个弱符号重名，则选择强符号（3）若只有多个弱符号，则从中任意选择一个。
（b）第一组：两个强符号　p1　重名！
第二组：两个弱符号重名，两个模块都可以更改 x 的值，会有意想不到的风险
第三组：两个弱符号重名，且数据类型（大小）不同，若选择前者链接，则会导致第二个模块的 x 出现莫名其妙的错误；若选择后者链接，由于 x 和 y 内存地址连续（认为是这样的），更改第二个模块 x 的值会更改八个字节的内容，导致 y 的值被覆盖。
第四组：选择第一个模块中的强符号 x，则在第二个模块中更改 x 的值一定会覆盖 y 的值。

18、说明陷阱/软中断（Trap）异常后的主要处理流程，并说明陷阱/软中断异常的主要用途
①流程：当用户需要产生这一有意的异常时，程序会执行 syscall 指令实现系统调用、将控制传递给异常（陷阱）处理程序，此时处理器的工作模式由原先的用户模式切换到具有更高权限的系统模式，进而可以获得调用执行特权指令、访问定义在内核中的栈等权限。执行完陷阱处理程序后，控制返回到 syscall 之后的指令。
②用途：当用户需要更高的权限向内核请求服务时，如读一个文件、创建一个新的进程、加载一个新的程序、终止当前进程等。

19、流水线：这里的计算最重要的是确定时钟周期，即流水线所有阶段（一个（或多个）逻辑组合加上其后一个寄存器视为一个阶段）中，最长的那一个最为时钟周期，之后就是套用公式（实际上只要把时钟周期取倒数就得到结果，其余的操作都只是单位的换算）
（1）负载均衡！时钟周期确定为 120ps（三个阶段），时钟周期（以 ps 为单位）取倒数再乘以 1000（）即得到结果（以 GIPS　10^9 条/每秒为单位）：8.33GIPS
（2）确定时钟周期为 360+20=380ps，取倒乘以一千得到 2.63GIPS

20、注：这一题应该是必考.....首先注意，研究对象是两个：虚拟地址、物理地址，所有的计算都在这里。这里由两种缓存（高速），虚拟地址的 TLB、物理地址的 Cache（高速缓存）。这里是对实际的简化模拟，按题目的简化规定：虚拟地址有 14 位，物理地址 12 位，前者多两位是因为前者多两位用来进行 TLB 中的组索引（题目条件有四组，所以需要两个比特位索引）。
注意，VPO（虚拟地址偏移量）和 PPO（物理地址偏移量）是相同的，且都是从右往左看，现在确定其位数：从高速缓存来看（从 TLB 看是一样的），有 16 个组，所以需要 4 位（s）确定，每一组四个字节，所以需要 2 位（b）确定真正的

字节偏移量。所以 PPO 有 4+2=6 位，从右往左看，前两位是字节偏移（b），之后四位是组索引（s），则物理地址还剩前六位作为 tag 位，虚拟地址还剩 8 位，这八位后两位作为组索引，前六位作为 tag 位
大概是这样子：



（1）VPN　6+2 = 8 位，VPO = PPO = 6 位；PPN　6 位，PPO6 位
（2）0x036a　先写出位 (00)00 0011 0110 1010
先看橙色——TLB 的组索引，是 1，红色的是 tag 位，其值位 0x03，查表，发现这一组有 tag 为 0x03 的！再看 valid 位为 1！则 TLB 命中，直接读取对应的 PPN:0x2D（六位，直接对应到物理地址的六位），于是得到物理地址的位表达：(00)10 110110 1010，接下来组索引（橙色）得到 A(10)，先看 tag 位为 2D，再看 valid 有效位为 1，命中！于是最后用到物理地址的最后两位字节偏移 10，即偏移两位，取到 Cache 第 A 组的第三个字节即 DA！

交互过程：开始时，MMU 抽取虚拟地址的 VPN（0x0D）,然后检查 TLB，即 TLB 抽取 TLB 索引（0x01），和 TLBtag（0x03）,和第一组的第一个条目有效匹配，将缓存的 PPN（0x2D）返还给 MMU，接下来 MMU 发送物理地址给缓存，缓存从物理地址中抽取缓存偏移 CO（0x2）、缓存组索引 CI（0xA）、缓存标记 CT（0x2D）由于高速缓存 Cache 组 A 中的 tag 标记与 CT 相同，且 valid 为 1，缓存命中！则读取再偏移量 CO 处的数据字节（0xDA），并将其返回给 MMU，随即 MMU（亦或者是 Cache 直接？书上说的模糊~）将其传递给 CPU。

（3）同样的过程，这一次发现 TLB 不命中，那么 MMU 需要从主存（内存）中读取相应的 PTE 并将其放在 TLB 中，这可能会覆盖一个已经存在的条目。

注：个人能力有限，可能会有勘误，如有疑问，请和各自的小伙伴讨论~