**第1周作业 计算机基本工作原理**

1、简要描述计算机工作的基本原理。

**答：计算机的工作原理是基于冯·诺依曼体系结构，其核心思想是“存储程序控制”：将程序和数据存储在内存中，CPU按顺序读取并执行指令。通过输入、处理、存储、输出和控制五个步骤，计算机能够完成复杂的任务。**

**输入：计算机通过输入设备（如键盘、鼠标、扫描仪等）接收用户提供的数据和指令。输入的数据被转换为计算机能够理解的二进制形式（0和1），并存储在内存中，等待进一步处理。**

**处理：中央处理器（CPU）是计算机的核心部件，负责执行指令和处理数据。CPU从内存中读取指令和数据，通过其内部的算术逻辑单元（ALU）进行算术运算（如加减乘除）和逻辑运算（如比较、判断）。CPU的工作分为以下几个阶段：取指令、解码、执行、写回。**

**存储：存储系统采用层次化结构（存储层次结构），从高速缓存（Cache）到内存再到外存，逐层降低速度和成本，提高整体效率。**

**输出：处理完成后，计算机通过输出设备（如显示器、打印机、扬声器等）将结果呈现给用户。输出数据可以是文本、图像、声音等形式。**

**控制：操作系统和软件负责管理和协调计算机的硬件资源，确保各个部件高效协作。操作系统通过调度程序、内存管理、文件系统等功能，为用户和应用程序提供统一的接口。**

2、程序（或者指令）执行的基本过程是什么？（特别要写清楚程序计数器在什么时候改变、如何改变）？

**答：程序执行的基本过程是取指令、译码、取操作数并执行、结果写回，而程序计数器（PC）在取指令阶段指向当前指令的地址，并在指令译码完成后自动增加该指令的长度，然后执行指令。如果当前指令是跳转指令（如条件分支的转移条件成立时、无条件转移指令、函数调用、从函数返回等），执行该指令，会再次修改PC的值，使得PC的值被修改为跳转目标地址。若是一般的数据传送、算术运算、位运算指令，则执行指令时不会再次修改PC。在一条指令执行结束后，会重复上述步骤，直到程序结束（返回操作系统）。**

3、在一个程序的调试中看到如下信息

6: x = 20;

00851853 C7 45 FC 14 00 00 00 mov dword ptr [ebp-4],14h

7: y = 21;

0085185A C7 45 F8 15 00 00 00 mov dword ptr [ebp-8],15h

8: z = 22;

00851861 C7 45 F4 16 00 00 00 mov dword ptr [ebp-0Ch],16h

1. 从地址 0x00851853 开始的连续的 20个字节内容是什么？【即内存窗口观察到的20个字节的内容，字节字节要有空格，以便阅读】

**C7 45 FC 14 00 00 00 C7 45 F8 15 00 00 00 C7 45 F4 16 00 00**

1. 在监视窗口看到 &x 为 0x010ff954， 此时，在寄存器窗口看到的 ebp 是多少（用16进制表示）？ y 的地址是多少？ z的地址又是多少？

**ebp = x地址 + 4 = 0x010ff954 + 4 = 0x010ff958**

**y 的地址 = ebp - 8 = 0x010ff958 - 8 = 0x010ff950**

**z 的地址 = ebp - 0Ch = 0x010ff958 - 0Ch = 0x010ff94C**

1. 在上述语句执行完后，从地址 0x010ff94c 开始的连续的 12个字节内容是什么？（每个字节都用两位16进制数表示）

**16 00 00 00 15 00 00 00 14 00 00 00**

1. 假设 再次调用上面语句所在的函数时，EBP = 0x0075FE24，则 x，y, z的地址又是多少？

**x 的地址 = 0x0075FE24 - 4 = 0x0075FE20**

**y 的地址 = 0x0075FE24 - 8 = 0x0075FE1C**

**z 的地址 = 0x0075FE24 - 0Ch = 0x0075FE18**

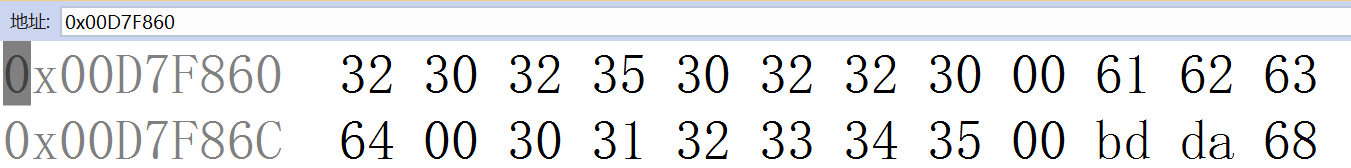
1. 虽然两次调用函数，看到的 x的地址发生变化，但x的地址表达形式 [ebp-4] 是没有变化的。这种表达形式是在什么时候就已确定的（是编译生成程序时，还是执行程序时）？ x的地址具体值又是在什么时候确定的？

**这种表达形式是在编译生成程序时确定的。编译器在编译阶段会根据函数的局部变量分配栈帧，并确定每个变量相对于 ebp 的偏移量。**

**x 的地址具体值是在程序运行时确定的。每次调用函数时，ebp 的值会根据当前的栈帧位置动态变化，因此 x 的地址（ebp-4）也会随之变化**

4、设有 char buf[20]; buf 的首地址为 0x00D7F860

（即数组的起始地址为0x00D7F860，后面元素的地址是顺序加1）。



1. 执行 printf("%s\n", buf); 即以字符串的形式显示buf中的内容，显示结果是什么？

**对应ASCII字符：  
'2' '0' '2' '5' '0' '2' '2' '0' '\0' 'a' 'b' 'b'**

**第一个\0停止，结果：20250220**

1. 执行 printf("%x %x\n", \*(int \*)buf, \*((int \*)buf+1); 显示的结果是什么？

**第一个读取4字节为：32 30 32 35，转为16进制：0x35323032 【Intel CPU,数据采用小端存放方式，数据的最低字节存在在地址最小的单元中】 显示：35323032  
 第二个读取4字节为：30 32 32 30，转为16进制：30323230**

1. 执行 printf("%x \n", \*(int \*)(buf+1)); 显示的结果是什么？

**从第二字节读取4字节，结果为：30323530 显示：30353230**

1. 写出 能给buf赋值成如上图所示内容的语句段。（即可用多条语句完成该功能）

**1）char buf[20] = {0x32, 0x30, 0x32, 0x35, 0x30, 0x32, 0x32, 0x30, 0x00, 0x61, 0x62, 0x63, 0x64, 0x00, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35};**

**2）char buf[20] = "20250220\0abcd\0012345";**

**3）char buf[20];**

**strcpy(buf, "20250220");**

**strcat(buf, "\0abcd"); strcpy(buf+9, "abcd");**

**strcat(buf, "\0012345"); strcpy(buf+14, "012345");**

**4）char buf[20];**

**char data[] = {0x32, 0x30, 0x32, 0x35, 0x30, 0x32, 0x32, 0x30, 0x00, 0x61, 0x62, 0x63, 0x64, 0x00, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35};**

**memcpy(buf, data, sizeof(data));**

5、对于一个二维数组所有元素求和，为什么按行序优先的原则（第0行元素都访问后，再访问第1行的元素，依次类推）比按列序优先的原则访问，速度要快？

**（1）二维数组是按行优先顺序存储的；**

**（2）当CPU要访问一个数据时，若数据不在 CPU的cache中时，则会从内存中将一定大小的数据块(如 4K)调入cache 。按行序访问时，要访问的下一个数据紧挨着前一个数据，前面调入数据块时，该数据就会在cache 中，避免了从内容中再去读数据。若按列序访问时，一个元素访问后，再访问下一个元素时，该元素与刚被访问的前一个元素的存放位置相距远，其所在的数据块不在cache中，会导新的内存的内存数据块被读入cache。简单的看，读入一个cache块，一次只能使用其中的一个数据，cache的命中率低。同时，存在一个数据块被多次调入cache,又多次被调出的问题，产生cache抖动，因而效率低下。根据数据访问的局部性原理，缓存（cache）可以高效地预取和存储连续的内存块；**

**（3）缓存命中率高意味着更少的主存访问，从而显著提升程序性能。**

6、机器指令与汇编语言语句有何关系？什么是汇编？什么是反汇编？

**（1）汇编语言是机器指令的符号化表示，通过汇编器将汇编语言转换为机器指令，机器指令是汇编语言的二进制形式，通过反汇编器将机器指令转换为汇编语言；**

**（2）汇编是指将汇编语言代码转换为机器指令的过程；**

**（3）反汇编是指将机器指令（二进制代码）转换为汇编语言的过程。**