一、 选择题（每小题 1 分，共 10 分）

1. **hello.i** 生成程序 **hello.s**，使用如下指令 （ **D** ）

   A．cpp **B. as** **C. cc** **D. cc1**

2. 浮点数的舍入采用（ **B** ）

   A． 四舍五入 **B.**向偶数舍入 **C.** 向 **0** 舍入 **D.** 向无穷大/小舍入

3. 现代 **Intel I7** 计算机的多级 **Cache** 采用（ **C** ）结构

   A. 全相联 **B.** 直接映射 **C.**组相联 **D.** 不同层级采用不同结构

4. 普通的局部变量是（ **D** ）。

   A.局部/本地符号 **B.**强符号 **C.**弱符号 **D.**以上都不是

5. **Intel I7 CPU** 访问 **L3 Cache** 采用 **(** **B** **)** 地址

   A. 虚拟地址 **B.**物理地址 **C.** 标记 **Tag** 地址 **D.**组索引

6. 普通内存 **RAM** 的一个 **bit** 单元采用（ **B** ）存储数据

   A.寄存器 **B.**电容 **C.** 二极管 **D.**触发器

7. **fork** 函数调用后返回（ **2** ）次

   A.1 **B. 2** **C.0** **D.**任意

8. 缓冲器溢出漏洞不能通过（ **B** ）防范

   A. 金丝雀技术 **B.**局部变量数组用 **nop** 填充

   C. 随机栈地址 **D.**页表引入 **NX** 不可执行位

9. 关于异常，描述错误的是（ **C** ）

   A. 发生时会执行对应内核子程序 **B.**系统调用如 **write** 是异常

   C. 信号是异常的一种

   D. 缺页故障是一种异常，完成磁盘块的加载及虚拟内存到物理内存的映射

10. 页表中的元素存放内容错误的是（ **D** ）

    A. 物理内存的页号 **B.**页面的许可位

    C. 页面的有效位（是否已缓存） **D.**磁盘块号

二、 填空题（10 分，每空 2 分）

11. int x=-10，则&x 开始的内存四个字节值为__F6 FF FF FF（或 FF FF FF F6）__(16 进制形式)
12. 函数 f (int a, short b)在 64 位环境下，b 采用 ___%SI____寄存器传递参数
13. 某 Cache 物理地址 20 位，64 组，每块 64 字节，则 tag 标记为___8_____位
14. %AL = -128 执行 NEGB %AL 后，%AL 的值为_____80_____(16 进制形式)
15. 对执行程序 hello 反汇编生成 hello.s 用___objdump -D hello > hello.s__命令

三、 判断对错（共 10 分，每题 2 分，正确打√、错误打×）

16. （ √ ）**for** 循环中的循环变量满足程序的时间局部性
17. （ × ）按键会产生异步异常，但不会给进程发送信号
18. （ √ ）**gcc hello.c -o hello** 生成的执行程序中 **printf** 采用动态链接
19. （ × ）程序中的 **int x=1； x** 一定是强符号

20. （ × ）IN/OUT 指令对外设进行输入输出的地址是物理内存地址

四、 简答分析题（共 40 分，每题 10 分）

21. 写出 float 大于 0 且非无穷大的最小数、最大数，根据其分布特点说明怎么编程进行两个浮点数的比较。

答：最小数：阶码为-126，尾数 0.0000 0000 0000 0000 0000 001，即 2^{-149} （2 分）

最大数：阶码为 127，尾数 1.1111 1111 1111 1111 1111 111 即（$2-2^{-23}$）x 2^{127}（2 分）

分布特点：越靠近数轴原点 0，浮点数分布越稠密，精度越高；

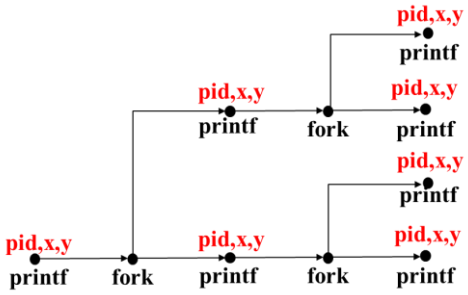
越远离数轴原点 0，浮点数分布越稀疏，精度越低。（2 分）

所以在两个浮点数进行比较时，不能仅仅看其数值是否完全一致，就判断其相等；也不能看起来相差极小就认为其相等，或者看起来相差极大就不认为其等。

根据浮点数比较的具体应用场景，如果两个浮点数的距离在此场景规定的范围内，就认为相等，否则就认为不等；根据差值是正或者负，可以确定谁大谁小。（4 分）

22. forkx 当前进程编号 100，画出其进程图，写出一个可能的执行结果？

```
int x=100;
void forkx(){
  int y=200;
  printf("%d %d %d\n",getpid(),++x,++y);
  fork();
  printf("%d %d %d\n",getpid(),++x,++y);
  fork();
  printf("%d %d %d\n",getpid(),++x,++y);
}
```



答：　图 5 分，错一处扣 1 分，直到为 0

```
100   101   201
100   102   202
100   103   203
101   102   202
101   103   203
102   103   203
103   103   203
```

执行结果 5 分，错一处扣 1 分，直到为 0。 注意要遵循父子孙进程的流程，随机调度

23. 请写出对如下程序进行优化的原理与结果程序。

```
void smooth(long img[5120][8192],long val[5120][8192]){
      int i,j;
      for(i=1;i<5119;i++)
         for(j=1;j<8191;j++)
            val[i][j] = ( img[i-1][j] + img[i+1][j] +img[i][j-1] + img[i][j+1] ) /4;
}
```

答：采用一般有用的优化方法，共享共用子表达式，采用精简指令，代码移动方式。4 级分离的循环展开。

```
void  void smooth(long img[5120][8192],long val[5120][8192]){
         long inj=
         for(i=1;i<5119;i++){
            long imgj=img+i*n;   //代码移动，共享共用子表达式
            long valj =val+i*n;     //代码移动，共享共用子表达式
            for(j=1;j<8191;j+=4){          //4 级分离的循环展开
               imgj +=j;
```

```
                valj +=j;
                *(valj) =( *( imgj-n) + *( imgj+n) + *( imgj-1) + *( imgj+1) ) >> 2
                                                //   >>2 代替 /4   精简指令
                imgj ++;
                valj++;
                *(valj) =( *( imgj-n) + *( imgj+n) + *( imgj-1) + *( imgj+1) ) >> 2
                imgj ++;
                valj++;
                *(valj) =( *( imgj-n) + *( imgj+n) + *( imgj-1) + *( imgj+1) ) >> 2
                imgj ++;
                valj++;
                *(valj) =( *( imgj-n) + *( imgj+n) + *( imgj-1) + *( imgj+1) ) >> 2
            }
        }
    }
```

注：也可以采用其他优化方法，特别是面向存储器的优化方法。2 种以上

24. 阅读如下程序，在线右边写出划线指令的功能，在下面写出完整的 C 语言程序

```
40128e:   mov      $0x0,%eax
401293:   mov      $0x0,%edx              累加和寄存器 rdx 初值赋值为 0
401298:   cmp      %esi,%eax
40129a:   jge      4012a8                 for 循环结束，跳出循环
40129c:   movslq   %eax,%rcx
40129f:   add      (%rdi,%rcx,8),%rdx     rdi 数组的 rcx 元素与 rdx 累加
4012a3:   add      $0x1,%eax
4012a6:   jmp      401298
4012a8:   mov      %rdx,%rax              累加和作为返回值给 rax 寄存器
4012ab:   retq
```

答：　功能 4 分，程序 6 分

```
long fp(long *arr,int n)                    //long arr[]     1 分
{
    long res=0;                                                1 分
    int i;

    for(int i=0;i<n;i++)                                       1 分
            res=res+ *(arr+i);              //res+=arr[i]      2 分

    return res;                                                1 分
}
```

注：while 循环等也算对，各种名字改动随意

五、线综合分析与设计题（30 分） 若空间不足可写在背面

25. 请分析指令 0X4018d3：C3　RETQ 在 CPU 内的执行过程（取指、访存、修改寄存器等），并结合代码或各种数据的不同地址空间（如虚拟地址、物理地址等）、不同存储区域（如代码、数据、堆栈等）、存储器的多层级结构（寄存器、L1……、RAM、硬盘等）、页表及其加速机制、缺页异常等，简要说明在最乐观以及最悲观的情况下，该指令访问各种存储的次数，标出每次访问存储的类型及其作用（20 分）。

答：各位老师，根据自己讲解的课程范围，灵活确定答案与分数范围

如第 9 章的 TLB、多级页表、内存映射若没有讲，可以去掉这部分评分。

1. RETQ 指令执行过程：取指令（从指令缓冲区中），译码 RSP—>MAR，执行-修改 RSP（+8）访存（MAR）→MDR，更新 PC 即 MDR->RIP(PC)　　（3 分）
2. RETQ 取指令访存过程：从 CS 段的 RIP 处即用 CS:RIP 逻辑地址取指令，　（10 分，此 1 分）
   1）访问描述符表：段式管理实现逻辑地址到线性地址（虚拟地址）VA 的变换。（可无）
   2）访问 L1-iTLB：用 VA 访问 L1 级的指令 TLB，若命中则返回物理页号 PPN（最乐观）。
   3）访问 L2-TLB：用 VA 访问 L2 统一 TLB，若命中则返回物理页号 PPN。
   4）访问页表：用 VA 依次访问 4 级页表，若在 L1Cache，则返回 PPN，否则同 Cache 步骤

5）缺页异常处理 1：若 PPN 未缓存，会产生缺页异常，执行异常处理子程序（取指令、数据等等，访存次数 X 与子程序实现相关）
6）缺页异常处理 2：若物理内存已满，则会驱逐一页到硬盘页面交换文件（访存次数 Y）
7）缺页异常处理 3：从磁盘文件加载一页到物理内存（访存次数 Z），其物理页号 PPN
8）缺页异常处理 4：PPN 更新 4 级页表、L2TLB 、L1-iTLB。可能含有 TLB 块的驱逐
9）缺页异常处理 5：返回步骤 1）处重新指令本指令。
10）　访问 L1-iCache：用 PPN 与 PPO 形成物理地址 PA 访问 L1 指令 Cache，命中则读
11）　访问 L2-Cache：不命中，用 PA 访问 L2 统一 Cache，命中读取指令内容（1 字节）
12）　访问 L3-Cache：不命中，用 PA 访问 L3 统一 Cache，命中读取指令内容（1 字节）
13）　访问主存：用 PA 访问物理内存，读取内存内容，考虑到都是对齐的，只访存一次。
14）　更新 Cache 与驱逐：L1 到主存访问期间，下级命中会更新、驱逐上级的块。

3. RETQ 访问存储器过程：从 SS：RSP 读 8 个字节，同 2 只不过是 L1-dTLB、L1-dCache（3 分）

最乐观：2*3=6 次，如不考虑段式管理，4 次（2 分）　。最悲观：至少 20 次以上（2 分）。

26. 结合进程、加载、动态链接、shell、信号等机制，简要论述“Hello World!”的执行程序 hello 在从键盘输入./hello 到被回收的整个执行过程（10 分）。

答：各位老师，根据自己讲解的课程范围，灵活确定答案与分数范围

hello 程序的执行是在 shell 下由 OS 进行管理和执行的。

1. shell 读取当前的命令行(fgets)　　“ ./hello”
2. shell 解析(eval)命令行，判断是否为内置命令(builtin_cmd)，hello 不是则执行 3
3. shell 使用 fork 创建子进程 hello,继承了父进程 shell 的页表等进程控制信息，但拥有完全独立的私有虚拟地址空间，且进程 pid 不同.
4. shell 创建新的作业 job，并将 hello 进程加入此作业，等待前台作业终止。
5. hello 子进程 execve 执行：删除旧的内存区域，通过 loader 模块将磁盘文件./hello 加载到内存,即调用 mmap 创建新的区域结构，建立文件与进程私有虚拟地址空间的映射。对应的虚拟页面为“未缓冲”
6. 由于 hello 调用了 printf 函数,shell 执行动态链接，从 C 语言标准库 libc.so 加载 printf.o 到内存,即调用 mmap 建立 printf.o 与进程 hello 的共享虚拟地址区域的映射。如有其他进程已经加载过了，则对应虚拟页面为“已缓冲”
7. 设置 PC 指向代码区域的入口点_start
8. 操作系统的进程调度到 hello，执行_start 处指令，在第一次执行指令或访问数据时，或产生缺页中断，实现虚拟内存到物理内存的映射（加载），然后重新执行指令。
9. hello 进程执行，调用 printf，进而调用 write 产生同步异常-陷阱，由 OS 完成显示输出。
10. hello 执行 return,进而通过 exit 终止进程的执行，向父进程 shell 发送 SIGCHLD 信号
11. 父进程 shell 的 sigchld 信号处理子程序，通过 waitpid 完成对 hello 僵尸子进程的回收,并删除作业 job。
12. 父进程 shell 的 waitfg 检查 job 状态非前台，则结束对此进程和作业的处理。