考试科目名称　计算机系统基础　（　卷）

学年 第___ 学期　教师 袁春风 苏丰 唐杰 汪亮 蒋炎岩　　考试方式：开卷

系（专业）　计算机科学与技术　　年级_____　　班级_____

学号_____　　姓名_____　　成绩_____

题号	一	二	三	四	五	六	七	八	九	十	十一	十二	十三
分数													

某生写了一个 C 语言程序，用于对一个数据索引文件按关键字进行排序。该程序有两个源文件：**main.c** 和 **sort.c**，它们的内容如下图所示（注：取消了写文件部分）。

```
/* main.c */
#include <stdio.h>
#include <stdlib.h>
typedef struct record {
      ......
} RECORD;
typedef struct index {
      unsigned char key;
      RECORD *pdata;
} INDEX;
extern void sort();
INDEX rec_idx[256];
const int rec_num = 256;
void main(int argc, char *argv[])
{   FILE *fp = fopen(argv[1], "rb");
    if (fp) {
        fread(rec_idx, sizeof(INDEX), rec_num, fp);
        fclose(fp);
    } else exit(1);
    sort();
}
```

```
/* sort.c：bubble sort */
extern INDEX rec_idx[];
extern const int rec_num;

void sort()
{
      int i, swapped;
      INDEX temp;
      do { swapped = 0;
           for(i=0; i<rec_num-1; i++)
                if (rec_idx[i].key > rec_idx[i+1].key) {
                     temp.key = rec_idx[i].key;
                     temp.pdata = rec_idx[i].pdata;
                     rec_idx[i].key = rec_idx[i+1].key;
                     rec_idx[i].pdata = rec_idx[i+1].pdata;
                     rec_idx[i+1].key = temp.key;
                     rec_idx[i+1].pdata = temp.pdata;
                     swapped = 1;
                    }
      } while (swapped);
}
```

假设在 IA-32/Linux 平台上用 GCC 编译驱动程序处理，**main.c** 和 **sort.c** 的可重定位目标文件名分别是 **main.o** 和 **sort.o**，生成的可执行文件名为 **bubsort**。使用“**objdump –d sort**”得到反汇编部分结果如下。

```
0   08048530 <sort>:
1   8048530: 55                      push   %ebp
2   8048531: 89 e5                   mov    %esp,%ebp
3   8048533: 83 ec 10                sub    $0x10,%esp
4   8048536: c7 45 f8 00 00 00 00    movl   $0x0,-0x8(%ebp)
5   804853d: c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%ebp)
6   8048544: e9 93 00 00 00          jmp    80485dc <sort+0xac>
7   8048549: 8b 45 fc                mov    -0x4(%ebp),%eax
8   804854c: 0f b6 14 c5 60 a0 04 08 movzbl 0x804a060(,%eax,8),%edx
9   8048554: 8b 45 fc                mov    -0x4(%ebp),%eax
10  8048557: 83 c0 01                add    $0x1,%eax
11  804855a: 0f b6 04 c5 60 a0 04 08 movzbl 0x804a060(,%eax,8),%eax
12  8048562: 38 c2                   cmp    %al,%dl
13  8048564: 76 72                   jbe    80485d8 <sort+0xa8>
14  8048566: 8b 45 fc                mov    -0x4(%ebp),%eax
15  8048569: 0f b6 04 c5 60 a0 04 08 movzbl 0x804a060(,%eax,8),%eax
16  8048571: 88 45 f0                mov    %al,-0x10(%ebp)
17  8048574: 8b 45 fc                mov    -0x4(%ebp),%eax
```

```
18  8048577: 8b 04 c5 64 a0 04 08       mov    0x804a064(,%eax,8),%eax
19  804857e: 89 45 f4                   mov    %eax,-0xc(%ebp)
 .....    ................          ........................
38  80485d1: c7 45 f8 01 00 00 00       movl   $0x1,-0x8(%ebp)
39  80485d8: 83 45 fc 01                addl   $0x1,-0x4(%ebp)
40  80485dc: a1 80 86 04 08             mov    0x8048680,%eax
41  80485e1: 83 e8 01                   sub    $0x1,%eax
42  80485e4: 3b 45 fc                   cmp    -0x4(%ebp),%eax
43  80485e7: 0f 8f 5c ff ff ff          jg     8048549 <sort+0x19>
44  80485ed: 83 7d f8 00                cmpl   $0x0,-0x8(%ebp)
45  80485f1: 0f 85 3f ff ff ff          jne    8048536 <sort+0x6>
46  80485f7: 90                         nop
47  80485f8: c9                         leave
48  80485f9: c3                         ret
```

已知 IA-32 页大小为 4KB，主存地址位数为 32 位。假设代码 Cache 和数据 Cache 的数据区大小皆为 8KB，采用 2 路组相联映射、LRU 替换算法和直写（Write Through）策略，主存块大小为 64B，系统中没有其他用户进程在执行，请回答下列问题或完成下列任务。

一、第 7~13 行指令实现什么功能？对应 sort.c 中哪条语句？第 40~41 行指令实现什么功能？（4 分）

答：第 7~13 行指令实现对 rec_idx[i].key 和 rec_idx[i+1].key 进行比较，对应的语句为“if (rec_idx[i].key > rec_idx[i+1].key) {…}”。（3 分）

第 40~41 行指令实现 rec_num 减 1，结果在 EAX 中。（1 分）

二、访问 Cache 时主存地址应如何划分？代码 Cache 的总容量为多少位？（7 分）

答：32 位主存地址中，块内地址占 $\log_2 64=6$ 位，代码 Cache 共有 8KB/64B=128 行，分成 128/2=64 组，因此组号（组索引）占 6 位，标记（Tag）字段占 32-6-6=20 位。（3 分）

因为每组 2 路，故每行中 LRU 位占 1 位，还有 1 位有效位、20 位标记、64B 数据。代码 Cache 的总容量位数为 128×(1+1+20+64×8)=68 352。（4 分）

（也可以每组一位 LRU 位，总容量为 64×1+128×(1+20+64×8)=68 288。）

三、第 18 行指令的源操作数采用什么寻址方式？第一次执行该指令时，源操作数的有效地址为多少？读取该指令过程中是否发生 TLB 缺失和缺页？为什么？读取该指令过程中是否发生 Cache 缺失？为什么？用 300 字左右简述该指令的执行过程。（25 分，若能结合题目中给出的具体例子清楚描述 IA-32/Linux 中的地址转换过程，则额外加 10 分））

答：采用“比例变址加位移量”的寻址方式。（1 分）第一次执行该指令时，R[eax]=0，因此，有效地址为 EA=R[eax]*8+0x804a064=0x804a064。（2 分）

取指令时不会发生 TLB 缺失，也不会发生缺页。因为该指令和其前面执行的指令在同一个页面，执行到该指令时，所在页已经被装入主存，对应页表已经被调入 TLB，并没有被替换出来。（4 分）

第一次执行该指令时，取指令过程中不会发生 Cache 缺失。因为该指令和第 17 行指令在同一个主存块中，执行第 17 行指令时，主存块已被调入 Cache，所以不会发生缺失。（3 分）

根据指令地址 0x8048577 中的 0x8048 访问 TLB，得到指令所在的物理地址；CPU 根据物理地址访问 Cache 命中，从 Cache 中取指令；对指令进行译码；根据源操作的寻址方式计算有效地址，然后根据有效地址高 20 位访问 TLB，若 TLB 命中，则取出 TLB 中的页框号与低 12 位有效地址串联起来形成物理地址；若 TLB 缺失，则进入 TLB 缺失处理，根据虚页号到主存页表中找到对应的页表项，若其中的存在位（有效位）为 1，则取出页框号与低 12 位有效地址串联起来形成物理地址；否则发生缺页，需要转入内核进行处理。得到物理地址后，则根据物理地址到 Cache 中查询，若 Cache 命中，则取出 Cache 中的相应数据，否则，发生 Cache 缺失，CPU 需要根据主存地址，到主存中取出该主存地址所在的主存块，然后装入 Cache，将对应 Cache 行的有效位置 1，并填入标记字段，并将需要的 32 位数据取出，存到 EAX 中。（15 分）

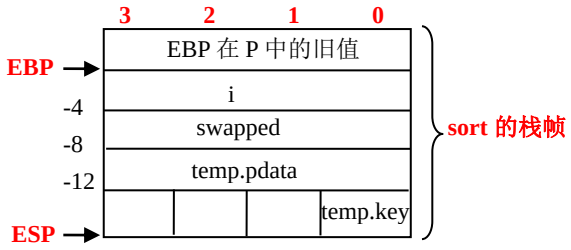
四、从反汇编代码中的哪条指令可看出 INDEX 的长度？sizeof(INDEX)为多少？编译器如何确定 INDEX 所占字节数？（4 分）

答：从 804854c（或 804855a 或 8048569 或 8048577）处的指令可以看出，sizeof(INDEX)=8。（2 分）
编译器根据指针类型按 4 字节边界对齐的原则，在 key 和 pdata 之间插入了 3 个字节的空间，因此 sizeof(INDEX)=1+3+4=8。（2 分）

五、43 行的 jg 指令采用的是什么寻址方式？为什么？从这条指令可以看出 IA-32 采用的是小端还是大端方式？为什么？该指令中偏移量的真值为多少？（7 分）

答：jg 指令采用相对寻址方式。（1 分）因为转移目标地址为 0x8048549，下条指令地址 PC 为 0x80485ed，0x804 8549-0x804 85ed=0x0804 8549+0xf7fb 7a13=0xff ff ff 5c，与指令中的后 4 个字节正好相反，因而可以看出，后 4 字节是偏移量，符合相对寻址公式“转移目标地址=PC+偏移量”。（3 分）
IA-32 采用小端方式。（1 分）因为指令中偏移量的 LSB 在小地址上。（1 分）
偏移量的真值为−1010 0100B=−164。（1 分）

六、根据反汇编结果画出 sort 函数（过程）的栈帧，要求分别用 EBP 和 ESP 标示出 sort 函数的栈帧底部和顶部，并标出 i、swapped 和 temp 中各个成员变量的位置。（6 分）



答：（6 分）

七、数组 rec_idx 和常量 rec_num 的起始虚拟地址分别是多少？数组名 rec_idx 和常量名 rec_num 分别在 main.o 中的哪个节（section）中定义？可执行文件 bubsort 只读代码段的起始地址是多少？可读可写数据段的起始地址可能是多少？main.c 和 sort.c 中各定义了哪几个符号？这些符号分别属于 bubsort 的哪个段？ （10 分）

答：rec_idx 的首地址是 0x804a060；rec_num 的地址是 0x8048680。（2 分）
rec_idx 在 main.o 中.bss 节定义；rec_num 在 main.o 中.rodata 节定义。（2 分）
只读代码段的起始地址是 0x8048000；可读可写数据段的起始地址可能是 0x8049000 或 0x804a000。（2 分）
main.c 中定义了符号 rec_idx、rec_num 和 main；sort.c 定义了符号 sort。（2 分）
rec_num、main 和 sort 属于只读代码段；rec_idx 属于可读可写数据段。（2 分）

八、若数组 rec_idx 定义为 main 函数内部的局部变量，则应如何调用 sort 函数（写出 sort 函数原型声明即可）？与题干中 main.c 的做法相比，程序包含的指令条数会增加还是减少？为什么？在被调用过程 sort 中，如何获得数组 rec_idx 的首地址，相应地，804854c 处的指令如何变化（写出两条汇编指令即可）？在 rec_idx 被定义为非静态局部数组（即 INDEX rec_idx[256];）和静态局部数组（即 static INDEX rec_idx[256];）的情况下，rec_idx 分别存放在进程虚存空间中的何处？（10 分）

答：void sort(INDEX *rec_idx);（或 void sort(INDEX rec_idx[])、void sort(INDEX *);）或
void sort(INDEX *rec_idx, int rec_num);（或 void sort(INDEX rec_idx[], int rec_num)、void sort(INDEX *, int);）（2 分）
程序包含的指令条数会增加。（1 分）因为 sort 函数的参数需要用指令来传递，而原来的做法无需用指令传递参数。（1 分）
可用以下指令“movl 0x8(%ebp), %ecx”获得数组 rec_idx 的首地址，存 ECX 中。（2 分）
804854c 处的指令应变成以下指令“movzbl (%ecx,%eax,8),%edx”。（2 分）

非静态局部数组时，rec_idx 存放在栈区，在 main 过程的栈帧中；静态局部数组时，rec_idx 存放在可读可写数据段。（2 分）

九、执行第 40 行指令时，源操作数所在主存块映射到数据 Cache 哪一组？rec_idx 数组共占多少字节？rec_idx 所在主存块被映射到数据 Cache 的哪些组中？第 40 行指令源操作数所在主存块会不会被 rec_idx 所在主存块替换出来？假定 rec_idx 数组各元素已经按升序排好序，且从 main 跳转到 sort 第一条指令后 R[esp]=0xbffa003c，则 sort 栈帧所在的虚拟地址范围是什么？执行 sort 函数过程中，对 sort 的局部变量的访问是否发生 Cache 缺失？为什么？执行 sort 函数过程中，rec_idx 数组的 Cache 访问命中率为多少？（17 分）

答：第 40 行指令的源操作数地址为 0x8048680，后 12 位地址为 0110 10 00 0000，因而映射到 Cache 第 26 组。（2 分）

rec_idx 数组共有 256×8=2048 字节。（1 分）rec_idx 起始地址为 0x804a060，后 12 位地址为 0000 01 10 0000，因为后 6 位地址不是 0，所以 rec_idx 数组共映射到 2048B/64B+1=32+1=33 个不同的 Cache 组中，组号为 1~33。（2 分）

虽然第 40 行指令的源操作所在主存块映射到在 1~33 之间的第 26 组，但是，因为 Cache 为 2-路组相联，所以，并不会被替换。（2 分）

sort 栈帧范围为底部 R[ebp]=0xbffa003c-4=0xbffa0038，顶部 R[esp]=0xbffa0038-0x10=0xbffa0028。（2 分）

对 sort 的局部变量的访问不会发生 Cache 缺失。（1 分）因为在 main 函数中 call 指令需要将返回地址压栈，已经将地址 0xbffa003c（后 12 位地址为 0000 00 11 1100）所在的主存块装入 Cache 第 0 组，sort 所有局部变量都在其栈帧中，而其栈帧与地址 0xbffa003c 同在一个主存块中，因而对 sort 的局部变量进行访问时，对应数据已在 Cache 第 0 组。（3 分）

因为 rec_idx 数组已经排好序，所以，if 语句的条件表达式不成立，因而无需执行交换操作，而只要执行关系表达式的计算。循环次数为 255，每次要访问两个数组中的数据，因此，总访问次数为 510，每个主存块总是第一次缺失，因而共缺失 33 次。综上，rec_idx 数组命中率为(510-33)/510=93.5%。（4 分）

十、在执行 bubsort 程序过程中，是否会陷入内核执行？计算机系统如何实现 fread 函数的功能（要求从调用 fread 库函数开始简要说明，包括对应哪个系统调用、如何从用户态陷入内核态、内核的大致处理过程等。200 字左右）？（10 分）

答：在执行 bubsort 程序过程中，会陷入内核执行。（1 分）

fread 库函数最终会调用系统调用封装函数 read()，该封装函数由若干 mov 指令和一条 int 0x80 指令以及其他指令序列组成。int 指令前面的若干 mov 指令用于将 read 系统调用的参数送到相应寄存器中，执行到 int 指令时，便从用户态陷入内核态，在内核中，通过系统调用号跳转到 sys_read 服务例程执行，sys_read 服务例程主要有三个部分组成：设备无关软件、设备驱动程序和中断服务程序。最终完成读取磁盘文件的操作。（9 分）