考试科目名称 计算机系统基础 （A 卷）

2019—2020 学年第 1 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 2018 班级

学号 姓名 成绩

题号	一	二	三	四	五	六	七	八	九	十
分数										

某生在 IA-32+Linux 平台上写了一个如右图所示的 C 语言程序 test.c。已知 IA-32 按字节编址，采用段页式存储管理方式，页大小为 4KB，主存地址位数为 32 位。假设代码 Cache 采用 4 路组相联映射，其数据区大小为 16KB，主存块大小为 64B。在 IA-32+Linux 平台上使用“objdump –d test”得到可执行目标文件 test 的反汇编部分结果如下。

```
/* test.c */
#include <stdio.h>
double fun(int i)
{     volatile double d[1]={ -19.5 };
      volatile int a[2];
      a[i]=-1020;
      return d[0];
}
int main()
{     int j;
      for (j=0; j<5; j++)
          printf("j=%d, d=%lf\n", j, fun(j));
}
```

```
    0804841d <fun>:
1   804841d: 55                          push   %ebp
2   804841e: 89 e5                       mov    %esp,%ebp
3   8048420: 83 ec 10                    sub    $0x10,%esp
4   8048423: dd 05 30 85 04 08           fldl   0x8048530
5   8048429: dd 5d f0                    fstpl  -0x10(%ebp)
6   804842c: 8b 45 08                    mov    0x8(%ebp),%eax
7   804842f: c7 44 85 f8 -- -- -- --     movl   $0x--------,-0x8(%ebp,%eax,4)
8   8048437: dd 45 f0                    fldl   -0x10(%ebp)
9   804843a: c9                          leave
10  804843b: c3                          ret
    0804843c <main>:
11  804843c: 55                          push   %ebp
... ....................................  ..........................
15  8048445: c7 44 24 1c 00 00 00 00     movl   $0x0,0x1c(%esp)
16  804844d: eb 29                       jmp    8048478 <main+0x3c>
17  804844f: 8b 44 24 1c                 mov    0x1c(%esp),%eax
18  8048453: 89 04 24                    mov    %eax,(%esp)
19  8048456: e8 -- -- -- --              call   804841d <fun>
20  804845b: dd 5c 24 08                 fstpl  0x8(%esp)
21  804845f: 8b 44 24 1c                 mov    0x1c(%esp),%eax
22  8048463: 89 44 24 04                 mov    %eax,0x4(%esp)
23  8048467: c7 04 24 20 85 04 08        movl   $0x8048520,(%esp)
24  804846e: e8 7d fe ff ff              call   80482f0 <printf@plt>
25  8048473: 83 44 24 1c 01              addl   $0x1,0x1c(%esp)
26  8048478: 83 7c 24 1c 04              cmpl   $0x4,0x1c(%esp)
27  804847d: 7e d0                       jle    804844f <main+0x13>
... ....................................  ..........................
```

请回答下列问题或完成下列任务。

一、第 4 行指令中地址 0x8048530 处存放的是何信息？该信息属于.text 节还是.rodata 节的内容？（2 分）

答：是-19.5 的机器数。（1 分）属于.rodata 节的内容。（1 分）

二、第 7 行指令的源操作数和目的操作数各采用什么寻址方式？其中 EAX 寄存器中存放的是什么信息？目的操作数存在虚拟地址空间的哪个区域？第 7 行指令实现什么功能？对应 test1.c 中哪条语句？对应汇编指令中立即数是什么？机器指令中最后 4 个字节分别是什么（用十六进制表示）？（10 分）

答：源操作数是立即寻址。（1 分）目的操作数是“基址+比例变址+位移量”寻址。（2 分）

EAX 寄存器中的内容是入口参数 i。（1 分）

目的操作数存在虚拟地址空间的用户栈区。（1 分）

实现将一个立即数存入 a[i]的功能。（1 分）

对应语句“a[i]=-1020;”（1 分）

-1020 的 int 型机器数为：-1020=-0011 1111 1100B，故对应机器数为 1… 1100 0000 0100B=FF FF FC 04H，对应汇编指令中立即数为$0xfffffc04。（2 分）

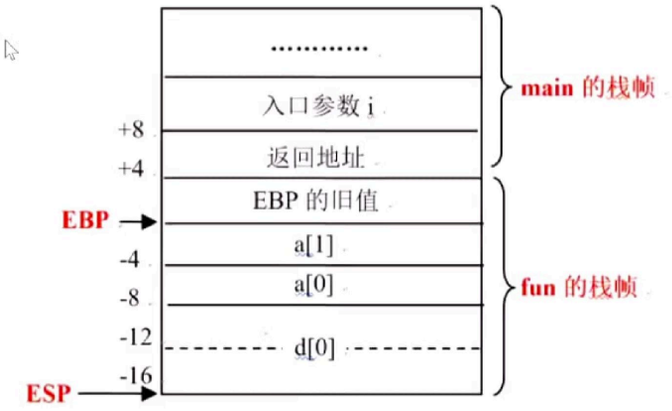
机器指令中后 4 个字节为 04 fc ff ff。（1 分）

三、整数类型返回值存放在哪里？从 fun 函数机器级代码可看出，浮点类型返回值存放在哪里？（2 分）

答：整数类型返回值存放在累加寄存器 EAX（或 AX 或 AH）。（1 分）

浮点类型返回值存放在浮点寄存器栈顶（即 ST0）。（1 分）

四、根据反汇编结果画出函数 main 和 fun 的栈帧，要求分别用 EBP 和 ESP 标示出 fun 的栈帧底部和顶部，并标出数组 d 和 a 中各元素的位置，以及 fun 函数的入口参数 i 和返回地址的位置。（7 分）



五、可执行文件 test1 的执行结果为：

j=0, d=-19.500000

j=1, d=-19.500000

j=2, d=-19.500000

Segmentation fault

这种情况下出现的“段故障”，在 IA-32 中的中断类型号是多少？在 Linux 中的信号名是什么？IA-32+Linux 平台中的硬/软件是如何检测、响应和处理该段故障的？（要求从执行段故障指令开始进行说明，包括地址转换、保护错检测、异常响应及处理等过程，300 字左右。）（16 分）

答：中断类型号是 13，信号名为 SIGSEGV。（2 分）

第一步检测到发生“段故障”。IA-32 采用段页式虚拟地址管理方式，因此，先通过查询段描述符表将 48 位逻辑地址转换为线性地址；然后，通过查询页表将线性地址转换为物理地址，在 MMU 进行地址转换时，若发现用户进程访问了不该访问的页面，或进行了与访问权限不相符的访问，则检测到存储保护错，从而产生 13 号中断（即段故障）。

第二步响应“段故障”异常。CPU 根据中断类型号 13，找到 IDT 中的第 13 个表项，CPU 将用户栈的栈顶信息保存到内核栈中，并将当前的断点（CS:EIP）和机器状态（EFLAGS）保存到内核栈顶，发生故障的线性地址保存到 CR2 中，最后将 IDT 表项中的段选择符和偏移地址分别装入 CS 和 EIP，从而转到段故障处理程序执行。

第三步处理“段故障”异常。Linux 中采用信号机制进行异常处理，在对应的异常处理程序 general_protection 中会发一个“SIGSEGV”信号给用户进程，然后返回到用户进程执行。用户进程接受到一个信号后，会转到“SIGSEGV”对应的信号处理程序去执行，该信号处理程序在屏幕上输出“segmentation fault”后，终止程序的执行。（14 分）

六、第 15~27 行指令（main 函数的部分机器级代码）中，哪些指令中有重定位字段（即其中包含的引用在链接生成可执行文件时进行了重定位）？各自引用的符号（或节名）是什么？各自采用的重定位方式是哪种？（8 分）

答：第 19 行（call 指令）、23 行（movl 指令）、24 行（call 指令）中有重定位字段。（3 分）

各自引用的符号（或节名）为：fun、.rodata、printf。（3 分）

movl 指令采用了绝对地址方式；两条 call 指令都采用了 PC 相对地址方式。（2 分）

七、第 19 行指令的机器代码中，后 4 个字节依次是什么（用十六进制表示）？（4 分）

答：call 指令采用 PC 相对寻址方式，转移目标地址=PC+位移量，因此，位移量=转移目标地址-PC=0x804841d-0x804845b=0xffffffc2，4 个字节依次是 c2、ff、ff 和 ff。（4 分）

八、访问代码 Cache 时主存地址如何划分？虚页号和页内地址各占几位？fun 函数所有代码是否在同一个页面中？为什么？fun 函数所有代码是否在同一个主存块中？为什么？fun 函数代码所在主存块对应的 Cache 的组号是什么？在 j=0 的情况下调用 fun(j)时，执行 fun 函数前，main 函数的逻辑控制流是什么？此时执行 fun 函数过程中是否会发生代码 Cache 缺失？为什么？在 j=1 的情况下调用 fun(j)时，执行 fun 函数过程中是否可能发生代码 Cache 缺失？为什么？（24 分）

答：Cache 共有 16KB/(64B×4)=64 组。因此，32 位主存地址中，块内地址占 6 位，组索引占 6 位，高 20 位是标记。（3 分）

虚页地址 32 位，页大小为 4KB，因此，页内地址占 12 位，虚页号占 32-12=20 位。（2 分）

fun 函数所有代码在同一个页面中。（1 分）

因为 fun 函数的虚拟地址范围为 0x804841d~0x804843b，其中高 20 位虚拟地址相同，因而在同一页中。（1 分）

fun 函数所有代码在同一个主存块中。（1 分）

因为 fun 函数所有代码在同一页面中，装入内存后在同一个页框中，转换后的物理地址的高 20 位也一定相同；虚拟地址中低 12 位为页内地址，其中，高 6 位为 Cache 组号，低 6 位为块内地址；低 12 位地址 0x41d=0100 00 01 1101B，0x43b=0100 00 11 1011B，这两个 12 位页内地址的高 6 位都是 010000，因此，它们的主存块号是相同的。（3 分）

映射到的 Cache 组号为 16。（1 分）

在 j=0 的情况下调用 fun(j)时，执行 fun 前，main 函数的逻辑控制流是 0x804843c~0x804844e、0x8048478~0x804847e、0x804844f~0x804845a，然后跳到 fun 的起始地址 0x804841d 执行。（4 分）

此时，执行 fun 过程中不会发生代码 Cache 缺失。（1 分）

因为 0x804843c 的后 12 位为 0100 00 11 1100，前 6 位与 fun 代码所在主存块对应的 Cache 组号相同。因此，在执行 0x804843c 处的指令时，已经把 fun 代码一起调入代码 Cache 的第 16 组。这种情况下，执行过的 main 代码除了映射到第 16 组的代码外，其余代码的地址的低 12 位中的高 6 位都是 0100 01，也即都映射到第 17 组。因而不会把第 16 组中的 fun 代码替换出来。（3 分）

在 j=1 的情况下调用 fun(j)时，执行 fun 函数过程中可能发生代码 Cache 缺失。（1 分）

因为在执行 fun 函数前，已经调用过 printf 函数，在此过程中，fun 函数的代码很可能从 Cache 中被替换出来，而调用 fun 之前，最靠近 fun 代码的第 17 行指令地址低 12 位中的高 6 位为 0100 01=17，与 fun 代码不在同一个主存块，fun 的代码不能被一起装入 Cache，因而执行 fun 代码过程中可能发生代码 Cache 缺失。（3 分）

九、简述第 1 次执行 printf 函数的整个过程，并说明第 2 次以后执行 printf 函数的过程与第 1 次执行过程有什么不同。（150 字左右）（7 分）

答：第 1 次执行 printf 函数，需要执行动态链接延迟绑定代码，以对 printf 函数进行重定位，将 printf 函数的首地址填到对应的 GOT 表项中，然后转到 printf 函数执行。printf 函数最终要调用 write 系统调用封装函数，其中有一条 int 0x80 指令，执行该指令后就陷入内核执行系统调用处理程序行 system_call，然后，根据系统调用号 4，跳转到 sys_write 执行，在内核经过设备无关层、设备驱动程序层，通过驱动 I/O 外设进行 I/O 处理。第 2 次以后执行 printf 函数无需进行重定位。（7 分）

十、选择题（每小题 2 分，共 20 分）

1. 考虑以下 C 语言代码：

```
int i=-129;
unsigned short usi=i;
```

执行上述程序段后，usi 的值是（ B ）。

A. $2^{15}-129$　　B. $2^{16}-129$　　C. $2^{31}-129$　　D. $2^{32}-129$

2. –3.75 采用 IEEE 754 单精度浮点数格式表示的结果（十六进制形式）是（ C ）。

A. 4070 0000H　　B. 40F0 0000H　　C. C070 0000H　　D. C0F0 0000H

3. 假设 R[ax]= DFF0H，R[bx]=6EF5H，执行指令“subw %bx, %ax”后，各标志为（ A ）。

A. OF=1，SF=0，CF=0　　B. OF=0，SF=1，CF=0

C. OF=1，SF=0，CF=1　　D. OF=0，SF=1，CF=1

4. 假设 R[eax]=0000 01F3H，R[ebx]=EF89 0010H，执行指令“mulw %bx”后，寄存器内容变化为（ A ）。

A. R[eax]=0000 1F30H，R[dx]=0000H　　B. R[eax]=0000 0000H，R[dx]=1F30H

C. R[eax]=0000 1F30H，R[bx]=0000H　　D. R[eax]=0000 1F30H，其余不变

5. 指令序列如下（左边为指令地址，中间为机器代码，右边为汇编指令）：

```
804857c   39 c2     cmpl   %eax, %edx
804857e   7e f8     jbe    xxxxxxxx
```

若执行到上述 cmpl 指令时，R[edx]=80，R[eax]=68，则 jbe 指令执行后将会执行（ D ）处指令。

A. 8048572　　B. 8048576　　C. 8048578　　D. 8048580

6. 假定静态 int 型二维数组 b 和指针数组 pb 的声明如下：

```
static int b[4][4]={ {2, 9, -1, 5}, {3, 1, -6, 2 }, {0, 23, 0, 10},{0, 20, 3, 12}};
static int *pb[4]={b[0], b[1], b[2], b[3]};
```

若 b 的首地址为 0x8049820，则&pb[0]和 pb[2]分别是（ A ）。

A. 0x8049860、0x8049840　　B. 0x8049884、0x8049840

C. 0x8049860、0x8049868　　D. 0x8049884、0x8049868

7. 假定一个磁盘的转速为 7200RPM，磁盘的平均寻道时间为 8ms，内部数据传输率为 4MB/s，不考虑排队等待时间，则读一个 512B 扇区的平均时间大约为（ B ）。

A. 12.16ms　　B. 12.29ms　　C. 16.32ms　　D. 16.46ms

8. 以下选项中，不属于“故障”类的异常是（ C ）。

A. 非法指令操作码　　B. 整数除 0　　C. 断点设置　　D. 缺页

9. 以下事件中，不会引起从用户态陷入内核态的是（ C ）。

A. 发生缺页　　B. DMA 传送完成　　C. 发生 cache 缺失　　D. 按“Ctrl+C”键

10. 以下程序中，不在内核态运行的是（ A ）。

A. 命令行解释程序　　B. 设备驱动程序　　C. 系统调用服务例程　　D. 中断服务程序