

考试科目名称 计算机系统基础 (A 卷)

2016—2017 学年第 1 学期 教师 袁春风 路通 苏丰 唐杰 汪亮 考试方式: 开卷

系 (专业) 计算机科学与技术 年级 2015 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, 用于计算 $f(n)=2^n+2^{n-1}+\cdots+2^0=11\cdots 1B$ ($n+1$ 个 1)。该程序有两个源文件: **main.c** 和 **test.c**, 它们的内容如下图所示。

```
/* main.c */
1  #include <stdio.h>
2  typedef int ret_type;
3  extern ret_type i_max( unsigned int );
4  int inval;
5
6  void main( ) {
7      ret_type result;
8      scanf( "%d", &inval );
9      result = i_max( inval );
10     printf( "Result = %d\n", result );
11 }
```

```
/* test.c */
1  typedef int ret_type;
2  ret_type i_max(unsigned inval) {
3      ret_type sum, tmp;
4      unsigned i;
5      sum = 1; tmp = 1;
6      for( i=0; i<=inval-1; i++ ) {
7          tmp = tmp* 2;
8          sum = sum + tmp;
9      }
10     return sum;
11 }
```

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

```
0804844b <i_max>:
0804844b: 55                push    %ebp
0804844c: 89 e5             mov     %esp, %ebp
0804844e: 83 ec 10          sub     $0x10, %esp
08048451: c7 45 fc 01 00 00 00 movl    $0x1, -0x4(%ebp)
08048458: c7 45 f8 01 00 00 00 movl    $0x1, -0x8(%ebp)
0804845f: c7 45 f4 00 00 00 00 movl    $0x0, -0xc(%ebp)
08048466: eb 0d             jmp     08048475 <i_max+0x2a>
08048468: d1 65 f8          shl     -0x8(%ebp)
0804846b: 8b 45 f8           mov     -0x8(%ebp), %eax
0804846e: 01 45 fc          add     %eax, -0x4(%ebp)
08048471: 83 45 f4 01       addl    $0x1, -0xc(%ebp)
08048475: 8b 45 08           mov     0x8(%ebp), %eax
08048478: 83 e8 01          sub     $0x1, %eax
0804847b: 3b 45 f4           cmp     -0xc(%ebp), %eax
0804847e: 73 e8             jae     08048468 <i_max+0x1d>
08048480: 8b 45 fc           mov     -0x4(%ebp), %eax
08048483: c9               leave
08048484: c3               ret

08048485 <main>:
08048485: 8d 4c 24 04       lea     0x4(%esp), %ecx
.....
08048499: 68 b0 97 04 08    push    $0x80497b0
0804849e: 68 70 85 04 08    push    $0x8048570
080484a3: e8 98 fe ff ff    call    8048340 <__isoc99_scanf@plt>
```

```

80484a8: 83 c4 10      add    $0x10,%esp
80484ab: a1 b0 97 04 08 mov    0x80497b0,%eax
80484b0: 83 ec 0c      sub    $0xc,%esp
80484b3: 50           push   %eax
80484b4: e8 92 ff ff ff call   804844b <i_max>
.....

```

回答下列问题或完成下列任务。

（提示：IA-32 为小端方式，按字节编址，字长为 32 位，即 `sizeof(int)=4`，页大小为 4KB，虚拟地址空间中的只读数据和代码段、可读写数据段都按 4KB 边界对齐）

一、执行程序时，该生在键盘上输入 0 后程序一直没有反应，为什么？此时，若在键盘上按下“Ctrl+C”即可让程序退出执行，这种导致程序终止的事件属于内部异常还是外部中断？从键盘上按下“Ctrl+C”到程序终止过程中计算机系统进行了哪些处理？（要求用 100~200 字简要描述处理过程）（8 分）

答：因为 `i_max` 函数的入口参数为 0 时，`ival=-1`，其机器数为 `11...1`，因此，for 循环的终止条件“`i>ival-1`”永远不会满足，for 循环为死循环，因而程序没有反应。（2 分）

用按键 Ctrl+C 终止程序属于外部中断。（1 分）

中断过程（略）（5 分）

二、若把 `test.c` 中所有 `unsigned` 改为 `int`，则 `i_max` 函数的反汇编结果中哪条指令会发生改变？应改变成什么指令？（2 分）

答：应该将 `jae` 改成 `jge`。（2 分）

三、在键盘上输入 31 时程序执行结果为 `Result=-1`，为什么？若输入 32，则程序执行结果是什么？（4 分）

答：因为 `ival=31`，函数 `i_max` 返回的机器数为 32 个 1，按照 `%d` 格式解释，为带符号整数 -1。（2 分）
`ival=32` 时，程序执行结果还是 -1。（2 分）

四、为了在更大输入值时也能正确执行，该生把程序中的 `ret_type` 改成 `float` 类型，`i_max` 函数名改为 `f_max`，`printf` 函数中 `%d` 改成 `%f`。结果发现当键盘输入值超过 23 时，`i_max` 的返回值总是比 `f_max` 的返回值小 1，例如，输入 24 时，前者为 33554431，后者为 33554432.000000。为什么？当输入为 127 时，函数 `f_max` 的返回值为 `inf`，为什么？此时返回结果的机器数是什么？（提示：`inf` 为无穷大）（6 分）

答：`ival=24` 时，对应的结果应该为 25 个 1，而 `float` 型有效数字只有 `23+1=24` 位，故函数 `f_max` 返回的值有舍入。舍入位为 1，舍入后尾数加 1，数值变大。（2 分）

`ival=127` 时，对应的精确结果应该为 128 个 1，即 $1.11\cdots 1 \times 2^{127}$ 。但程序在计算过程中会发生舍入，用 `float` 型格式表示时，23 位尾数后面的 1 舍入后会有进位，因而尾数变为 `10.00...0`，进行右规操作，尾数右移，阶码加 1，用 `float` 型格式表示：阶码为 `127+127+1=255`，即阶码全 1、尾数全 0，结果为无穷大。（3 分）

结果的机器数为 `0 1111 1111 000 0000 0000 0000 0000 0000B=7F80 0000H`。（1 分）

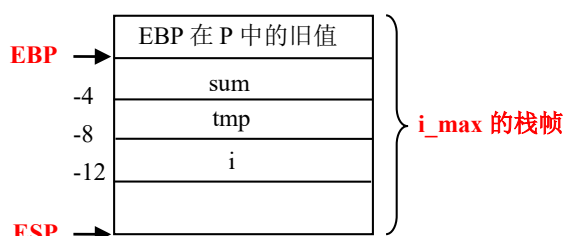
五、`i_max` 函数中哪条指令实现了 `tmp=tmp*2` 的功能？对于第四题中提到的函数 `f_max`，能否也不用乘法指令来实现 `tmp=tmp*2`？为什么？（4 分）

答：`i_max` 的机器级代码中，`shll` 指令是左移指令，左移一位相当于乘 2。（2 分）

对于 `f_max` 函数，可以用加法指令来实现乘 2 功能。（2 分）

六、根据反汇编结果画出 `i_max` 函数（过程）的栈帧，要求分别用 `EBP` 和 `ESP` 标示出 `i_max` 函数的栈帧底部和顶部，并标出 `sum`、`tmp` 和 `i` 的位置。（6 分）

答：（6 分）



七、该进程的只读代码段和可读写数据段的起始地址各是什么？main.c 和 test.c 中各定义了哪几个符号？这些符号分别在哪个段中定义？库函数 scanf 和 printf 中格式符（如"%d"）在哪个段中定义？（8 分）

答：只读代码段和可读写数据段的起始地址分别是 0x8048000 和 0x8049000。（2 分）

main.c 中定义了一个全局变量 inval 和一个函数 main。（2 分）

test.c 中定义了一个函数 i_max。（1 分）

函数 main 和 i_max 定义在只读代码段内；inval 定义在可读可写数据段内。（2 分）

格式符（如"%d"）在只读代码段中定义。（1 分）

八、填写下表中各标识符的情况，说明每个标识符是否出现在 test.o 的符号表（.symtab 节）中，如果是的话，进一步说明定义该符号的模块是 main.o 还是 test.o、该符号的类型是全局、外部还是本地符号、该符号出现在 test.o 中的哪个节（.text、.data 或.bss）。（6 分）

标识符	在 test.o 的符号表中？	定义模块	符号类型	节
i_max				
inval				
sum				

答：

标识符	在 test.o 的符号表中？	定义模块	符号类型	节
i_max	是	test.o	全局	text
inval	否	-	-	-
sum	否	-	-	-

九、地址 0x8048480 处的 mov 指令中，源操作数采用什么寻址方式？若 R[ebp]=0xbfff f000，则源操作数的有效地址是多少？源操作数的访问过程需要经过哪些步骤？（要求从有效地址计算开始进行简要说明，包括何时判断及如何判断 TLB 缺失、缺页和 cache 缺失等，300 字左右）（18 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“基址+位移量”寻址方式。（1 分）

有效地址 EA=R[ebp]-0x4 = 0xbfff f000-4 = 0xbfff effc（2 分）

操作数的访问过程（略）。（15 分）

十、已知页大小为 4KB，主存地址位数为 32 位。假设代码 Cache 的数据区大小为 32KB，采用 8 路组相联映射方式，主存块大小为 64B，采用 LRU 替换算法和回写（Write Back）策略，则主存地址如何划分？代码 Cache 的总容量是多少？虚拟地址页内偏移（VPO）与 Cache 组索引（CI）和 Cache 块内偏移（CO）之间有什么关系？函数 i_max 的代码将会映射到几个 Cache 组？各自映射到的 Cache 组号是什么？试分析执行 i_max 函数过程中可能发生几次缺失？（20 分）

答：32 位主存地址中，块内地址占 $\log_2 64=6$ 位，代码 Cache 共有 $32\text{KB}/64\text{B}=512$ 行，分成 $512/8=64$ 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 8 路，故每行中 LRU 位占 3 位，还有 1 位修改位、1 位有效位、20 位标记、64B 数据。总容量位数为 $512 \times (3+1+1+20+64 \times 8)=274\,944$ 。（4 分）

因为页大小为 4KB，按字节编址，所以页内偏移量占 12 位。即物理地址和虚拟地址的低 12 位完全相同，因此，虚拟地址的低 12 位中，高 6 位为 Cache 组号，低 6 位为块内地址。（2 分）

映射到两个 Cache 组。（1 分）

i_max 代码的虚拟地址中，804844b~804847f 映射到同一个 Cache 组，8048480~8048484 映射到同一组，前者对应组号为 0100 01，后者的对应组号为 0100 10。（2 分）

执行 i_max 函数过程中可能会发生 0 次或 1 次 cache 缺失。（1 分）

虚拟地址 804844b~804847f 范围内的指令中，可能第一条指令会缺失；若在执行 i_max 之前已经有同一个主存块的指令执行过，则不会发生缺失。因为 8048480~8048484 范围内的 3 条指令与 main 中的到 80484b4 处的 call 指令，都在同一个主存块中，对应的 cache 组号为 18，从 call 指令转到 i_max 执行后，i_max 中 804844b~804847f 范围内的指令对应的 cache 组号为 17，不会冲掉已经装入到 cache 第 18 组中的 8048480~8048484 范围内的 3 条指令，因而不会发生缺失。综上所述，执行 i_max 函数过程中可能会发生 0 次或 1 次 cache 缺失。（7 分）

十一、根据反汇编结果回答以下问题并给出理由：在执行 i_max 函数的过程中，是否可能发生缺页异常？是否可能因为溢出而转内核进行相应处理？（6 分）

答：不会发生缺页。（1 分）因为转到 i_max 执行之前，已经执行了 main 函数，它们在同一页，因而 i_max 的代码已经在主存。（2 分）

不会因溢出转内核处理。（1 分）执行 INTO 指令时，如果 OF=1，则触发异常，需转内核处理。而这里反汇编结果中没有 INTO 指令。（2 分）

十二、printf 语句用于实现从屏幕上显示一个字符串信息。计算机系统如何实现 printf 的功能？（要求从调用 printf 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）（12 分）

答：（略）。