主管领导审核签字

哈尔滨工业大学 2019 学年 秋 季学期

计算机系统（A） 试 题

题号	一	二	三	四	五	六	总分
得分							
阅卷人							

片纸鉴心 诚信不败

授课教师 姓名 学号 院系 密 封

一、 单项选择题（每小题 1 分，共 20 分）

1. 在 Linux 系统中利用 GCC 作为编译器驱动程序时，能够将汇编程序翻译成可重定位目标程序的程序是（ C ）
   A. cpp B. ccl C. as D. ld
2. 在 x86-64 中，有初始值%rax =0x1122334455667788，执行下述指令后 rax 寄存器的值是（ A ）
   movl $0xaa11, %rax
   A. 0xaa11 B. 0x112233445566aa11
   C. 0x112233440000aa11 D. 0x11223344ffffaa11
3. 下列 Y86-64 硬件结构中，程序员不可见的是（ B ）
   A. 程序寄存器 B. 算逻运算单元（ALU） C. 程序计数器 D. 内存
4. 利用 GCC 生成代码过程中，不属于编译器优化的结果是（ D ）
   A. 用移位操作代替乘法指令 B. 消除循环中的函数调用
   C. 循环展开 D. 使用分块提高时间局部性
5. 记录内存物理页面与虚拟页面映射关系的是（ D ）
   A. 磁盘控制器 B.编译器 C.虚拟内存 D.页表
6. Y86-64CPU 顺序结构设计中，在更新 PC 时与指令 jmp 地址来源相同的指令是（ B ）
   A. pushq B. call C. cmovxx D. ret
7. CPU 寄存器作为计算机缓存层次结构的最高层，决定哪个寄存器存放某个数据的是（ C ）
   A. MMU B.操作系统内核 C.编译器 D.CPU
8. Linux 系统中将可执行目标文件（.out 文件）装入到存储空间时，没有装入到.text 段的是（ D ） 老史：还是应该是代码段，.text 段不准确
   A.ELF 头 B. .init 节 C. .rodata 节 D. .symtab 节
9. 下列异常中经异常处理后能够返回到异常发生时的指令处的是（ C ）
   A. 键盘中断 B. 陷阱 C. 故障 D. 终止
10. 导致进程终止的原因不包括（ B ）
    A. 收到一个信号 B. 执行 wait 函数 C. 从主程序返回 D. 执行 exit 函数

11. 某进程在成功执行函数 malloc(24)后，下列说法正确的是（ C ）
    A. 进程一定获得一个大小 24 字节的块
    B. 进程一定获得一个大于 24 字节的块
    C. 进程一定获得一个不小于 24 字节的块
    D. 进程可能获得一个小于 24 字节的块
12. 下列不属于进程上下文的是（ C ）
    A.页全局目录 pgd　　B.通用寄存器　　C.内核代码　　D.用户栈
13. 下列函数中属于系统调用且在调用成功后，不返回的是(B)
    A.fork　　B.execve　　C.setjmp　　D.longjmp
14. 动态内存分配时产生内部碎片的原因不包括（ D ）
    A. 维护数据结构的开销
    B. 满足对齐约束
    C. 分配策略要求
    D. 超出空闲块大小的分配请求
15. 链接过程中，赋初值的静态全局变量属于（ D ）
    A.强符号　　B.弱符号　　C.可能是强符号也可能是弱符号　D.以上都不是
16. 虚拟页面的状态不可能是（ D ）
    A. 未分配　　B. 已分配未缓存　C. 已分配已缓存　　D. 已缓存未分配
17. C 语言中不同类型的数值进行强制类型转换时，下列说法错误的是（ A ）
    A. 从 int 转换成 float 时，数值可能会溢出
    B. 从 int 转换成 double 后，数值不会溢出
    C. 从 double 转换成 float 时，数值可能会溢出，也可能舍入
    D. 从 double 转换成 int 时，数值可能溢出，可能舍入
18. 三个进程其开始和结束时间如下表所示，则说法正确的是(D)

进程	开始时刻	结束时刻
P1	1	5
P2	2	8
P3	6	7

    A. P1、P2、P3 都是并发执行　　B.只有 P1 和 P2 是并发执行
    C. 只有 P2 和 P3 是并发执行　　D.P1 和 P2、P2 和 P3 都是并发执行
19. x86-64 系统中，函数 int sum (int x, int y)经编译后其返回值保存在（ C ）
    A. %rdi A. %rdi　　B.%rsi　　C.%rax　　D.%rdx
20. x86-64 中，某 C 程序定义了结构体

```
struct SS {
        double v;
        int i;
        short s;
    } aa[10];
```

    则执行 sizeof(aa)的值是（ D ）
    A. 14　　B.80　　C.140　　D. 160

二、填空题（每空 1 分，共 10 分）

21. 若字节变量 x 和 y 分别为 0x10 和 0x01，则 C 表达式 x&&~y 的字节值是__0x01__。
22. 按照“向偶数舍入”的规则，二进制小数 101.110_2 舍入到最接近的 1/2（小数

点右边 1 位）后的二进制为___110.0_2_____。

23. C 程序中定义 int x=-3，则&x 处依次存放（小端模式）的十六进制数据为_ FD FF FF FF__。

24. 某 CPU 主存地址 32 位，高速缓存总大小为 4K 行，块大小 16 字节，采用 4 路组相连，则标记位的总位数（每行标记位数*总行数）是____72K_____。

32-10-4 = 18，18*4K=72K

25. 可重定位目标文件中代码地址从__0____开始。

26. 当工作集的大小超过高速缓存的大小时，会发生__容量_____不命中。

27. 虚拟内存在内存映射时，映射到匿名文件的页面是____请求二进制零____的页。

28. 存储器层次结构中，高速缓存（Cache）是___主存/DRAM____的缓存。

29. TLB 常简称快表，它是____页表_____ 的缓存。

30. 虚拟内存发生缺页时，MMU 将触发__缺页中断/异常/故障____。

密

三、判断对错（每小题 1 分，共 10 分，在题前打 √ X 符号）

31. （ X ）C 语言中对整型指针 p，当 p=null 时，表达式 p&&*p++会间接引用空指针。

32. （ X ）C 语言中，关系表达式：127 > (unsigned char)128U 是成立的。

33. （ X ）x 和 y 是 C 中的整型变量，若 x 大于 0 且 y 大于 0，则 x+y 一定大于 0。

34. （ X ）Cache 的大小对程序运行非常重要，必要的时候可以通过操作系统提高 Cache 的大小。

35. （ √ ）CPU 在同一次访问 Cache L1、L2、L3 时使用的地址是一样的。

36. （ X ）进程是并发执行的，所以能够并发执行的都是进程。

37. （ X ）系统中当前运行进程能够分配的虚拟页面的总数取决于虚拟地址空间的大小。

38. （ √ ）显式空闲链表的优点是在对堆块进行搜索时，搜索时间只与堆中的空闲块数量成正比。

封

39. （ X ）X86-64 CPU 中的寄存器一定都是 64 位的。

40. （ X ）当执行 fork 函数时，内核为新进程创建虚拟内存并标记内存区域为私有的写时复制，意味着新进程此时获得了独立的物理页面。

四、简答题（每小题 5 分，共 20 分）

41. 写出 float f=-1 的 IEEE754 编码。（请按步骤写出转换过程）

-1 =-1.0x2^0　　阶为 0

所以，符号 S=1

阶码 E=127+0=127=01111111_2

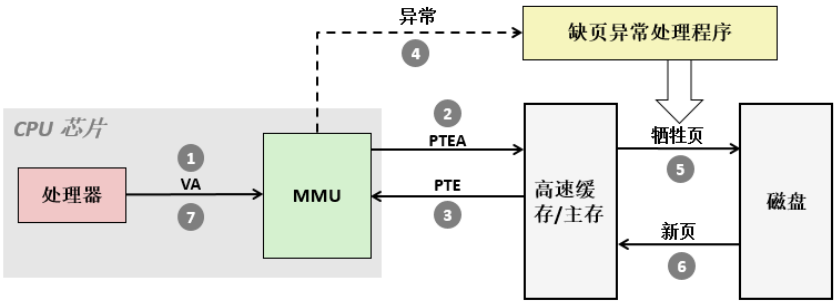
尾数=000 0000 0000 0000 0000 0000_2

f 的 IEEE754 编码为 1 01111111 000 0000 0000 0000 0000 0000

授课教师_____ 姓名_____ 学号_____ 院系_____

其十六进制为 0xBF800000

42. 结合下图，简述虚拟内存地址翻译的过程。



（1） 处理器将虚拟地址发送给 MMU
（2） （3）MMU 利用虚拟地址对应的虚拟页号生成页表项（PTE）地址，并从页表中找到对应的 PTE
（4） PTE 中的有效位为 0，MMU 触发缺页异常
（5） 缺页处理程序选择物理内存中的牺牲页（若页面被修改，则换出到磁盘）
（6） 缺页处理程序调入新的页面到内存，并更新 PTE
（7） 缺页处理程序返回到原来进程，再次执行导致缺页的指令

43. 下列 C 程序存在安全漏洞，请给出攻击方法。如何修复或防范？

```
int getbuf(char *s) {
    char    buf[32];
    strcpy( buf, s );
}
```

攻击方法：采用基于缓冲区溢出攻击，让输入字符串 s 的字符个数大于 32，可导致 getbuf 函数返回到无关的代码处，或返回到指定的攻击代码处。

修复:限制字符串操作的长度可编码缓冲区溢出的攻击，如：

```
int getbuf_s(char *s) {
    char    buf[32];
    if(strlen(s)<sizeof(buf))
        strcpy( buf, s );
}
```

系统级的防范措施：栈空间地址的随机偏移、将栈 stack 标记为不可执行、或在栈中某个位置放入特定的金丝雀值。

44. 结合 fork，execve 函数，简述在 shell 中加载和运行 hello 程序的过程。

① 在 shell 命令行中输入命令：$./hello
② shell 命令行解释器构造 argv 和 envp;
③ 调用 fork()函数创建子进程，其地址空间与 shell 父进程完全相同，包括只读代码段、读写数据段、堆及用户栈等
④ 调用 execve()函数在当前进程（新创建的子进程）的上下文中加载并运行 hello 程序。将 hello 中的.text 节、.data 节、.bss 节等内容加载到当前进程的虚拟地址空间
⑤ 调用 hello 程序的 main()函数，hello 程序开始在一个进程的上下文中运行。

五、系统分析题（每小题 5 分，共 20 分）

两个 C 语言程序 main2.c、addvec.c 如下所示：

```
/* main2.c */
/* $begin main2 */
#include <stdio.h>
#include "vector.h"

int x[2] = {1, 2};
int y[2] = {3, 4};
int z[2];

int main()
{
    addvec(x, y, z, 2);
    printf("z = [%d %d]\n", z[0],
z[1]);
    return 0;
}
/* $end main2 */
```

```
/* addvec.c */
/* $begin addvec */
int addcnt = 0;

void addvec(int *x, int *y,
            int *z, int n)
{
    int i;

    addcnt++;

    for (i = 0; i < n; i++)
        z[i] = x[i] + y[i];
}
/* $end addvec */
```

用如下两条指令编译、链接，生成可执行程序 prog2：

```
gcc -m64 -no-pie -fno-PIC -c addvec.c main2.c
gcc -m64 -no-pie -fno-PIC -o prog2 addvec.o main2.o
```

运行指令 objdump -dxs main2.o 输出的部分内容如下：

```
Disassembly of section .text:

0000000000000000 <main>:
   0:   48 83 ec 08                sub    $0x8,%rsp
   4:   b9 02 00 00 00             mov    $0x2,%ecx
   9:   ba 00 00 00 00             mov    $0x0,%edx
            a: R_X86_64_32  z
   e:   be 00 00 00 00          mov    $0x0,%esi
            f: R_X86_64_32  y
  13:   bf 00 00 00 00          mov    $0x0,%edi
            14: R_X86_64_32 x
  18:   e8 00 00 00 00          callq  1d <main+0x1d>
            19: R_X86_64_PC32   addvec-0x4
  1d:   8b 0d 00 00 00 00       mov    0x0(%rip),%ecx        # 23 <main+0x23>
            1f: R_X86_64_PC32   z
  23:   8b 15 00 00 00 00       mov    0x0(%rip),%edx        # 29 <main+0x29>
            25: R_X86_64_PC32   z-0x4
  29:   be 00 00 00 00          mov    $0x0,%esi
            2a: R_X86_64_32 .rodata.str1.1
  2e:   bf 01 00 00 00          mov    $0x1,%edi
  33:   b8 00 00 00 00              mov    $0x0,%eax
  38:   e8 00 00 00 00          callq  3d <main+0x3d>
```

```
                39: R_X86_64_PC32   __printf_chk-0x4
  3d:   b8 00 00 00 00              mov     $0x0,%eax
  42:   48 83 c4 08                 add     $0x8,%rsp
  46:   c3                          retq
```

objdump -dxs prog2 输出的部分内容如下（■是没有显示的隐藏内容）：

```
SYMBOL TABLE:
0000000000400238 l    d  .interp  0000000000000000              .interp
0000000000400254 l    d  .note.ABI-tag
0000000000000000 l    df *ABS*    0000000000000000              main2.c
0000000000601038 g        *ABS*   0000000000000000              _edata
000000000060103c g     O .bss     0000000000000008              z
0000000000601030 g     O .data    0000000000000008              x
0000000000000000       F *UND*    0000000000000000              addvec
0000000000601018 g       .data    0000000000000000              __data_start
00000000004007e0 g     O .rodata  0000000000000004              _IO_stdin_used
0000000000601028 g     O .data    0000000000000008              y
00000000004006f0 g     F .text0000000000000047                  main

00000000004005c0 <addvec@plt>:
  4005c0:   ff 25 42 0a 20 00       jmpq    *0x200a42(%rip)          # 601008
                                            <_GLOBAL_OFFSET_TABLE_+0x20>
  4005c6:   68 01 00 00 00          pushq   $0x1
  4005cb:   e9 d0 ff ff ff          jmpq    4005a0 <_init+0x18>

00000000004005d0 <__printf_chk@plt>:
  4005d0:   ff 25 3a 0a 20 00       jmpq    *0x200a3a(%rip)          # 601010
                                            <_GLOBAL_OFFSET_TABLE_+0x28>
  ....

00000000004006f0 <main>:
  4006f0:   48 83 ec 08             sub     $0x8,%rsp
  4006f4:   b9 02 00 00 00          mov     $0x2,%ecx
  4006f9:   ba ①_ _ _ _             mov     ■■■■,%edx
  4006fe:   be ②_ _ _ _             mov     ■■■■,%esi
  400703:   bf ③_ _ _ _             mov     ■■■■,%edi
  400708:   e8 ④_ _ _ _             callq   4005c0 <addvec@plt>
  40070d:   8b 0d ⑤_ _ _ _          mov     ■■■■(%rip),%ecx        # 601040 <z+0x4>
  400713:   8b 15 ⑥_ _ _ _          mov     ■■■■(%rip),%edx        # 60103c <z>
  400719:   be e4 07 40 00          mov     $0x4007e4,%esi
  40071e:   bf 01 00 00 00          mov     $0x1,%edi
  400723:   b8 00 00 00 00          mov     $0x0,%eax
  400728:   e8 ⑦ _ _ _ _            callq   4005d0 <__printf_chk@plt>
  40072d:   b8 00 00 00 00          mov     $0x0,%eax
  400732:   48 83 c4 08             add     $0x8,%rsp
  400736:   c3                      retq
```

45. 请指出 addvec.c main2.c 中哪些是全局符号？哪些是强符号？哪些是弱符号？以及这些符号经链接后在哪个节？（5 分）

全局符号：x、y、z、main、addvec、addcnt

强符号：x、y、main、addvec、addcnt

若符号：z（1 分）

x、y 在.data 节，z 在.bss 节，main.text 节。

addvec 在未定义节（UND），addcnt 被优化掉（addvec、addcnt 未说明不扣分）

46. 根据上述信息，**main** 函数中空格①--⑦所在语句所引用符号的重定位结果是什么？以 **16** 进制 **4** 字节数值填写这些空格，将机器指令补充完整（写出任意 **3** 个即可）。（**5** 分）

① 3c 10 60 00　　② 28 10 60 00
③ 30 10 60 00　　④ b3 fe ff ff
⑤ 2d 09 20 00　　⑥ 23 09 20 00
⑦ a3 fe ff ff

47. 某 **CPU** 的 **L1** cache 容量 32kb，64B/块，采用 8 路组相连，物理地址 47 位。试分析其结构参数 B、S、E 分别是多少？地址 0x00007f6635201010 访问该 L1 时，其块偏移 CO、组索引 CI、标记 CT 分别多少？（5 分）

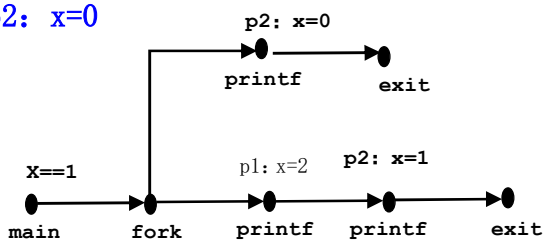
容量 32kb：则 64*8*8 = 4kB
所以：B=64, S=8, E=8; CO = 0x10,
CI=0x0, CT=0x3fb31a9008

若以 32kB 算：B=64，S=64，E=8；CO = 0x10, CI=0x0, CT=0x7f6635201

48. C 程序如下，请画出对应的进程图，并回答父进程和子进程分别输出什么？

```
int main()
{
        int x = 1;
        if(Fork() ! = 0)
                printf( “p1: x=%d\n”, ++x);
        printf( “p2: x=%d\n”, --x);
        exit(0);
}
```

父进程：p1：x=2
　　p2：x=1
子进程：p2：x=0



进程图

六、综合设计题（每小题 10 分，共 20 分）

49. 优化如下程序，给出优化结果并说明理由。（**10** 分）

```
int sum_array(int a[M][N][N])  //M、N 足够大
{
   int i, j, k, sum = 0;
   for (i = 0; i < N; i++)
```

```
        for (j = 0; j < N; j++)
            for (k = 0; k < M; k++)
            sum += a[k][i][j];
    return sum;
}
```

可采用面向 CPU 的优化、面向 Cache 的优化。至少给出一种优化方案，并给出合理细致的解释即可。

50. 写出 Y86-64CPU 顺序结构设计中 addq 指令各阶段的微操作。为 Y86-64 CPU 增加一条指令"mraddq D(rB)，rA"，能够将内存数据加到寄存器 rA。请参考 mrmovq、addq 指令，合理设计 mraddq D(rB)，rA 指令在各阶段的微操作，或给出设计思想。(10 分)

指令 mraddq D(rB)，rA 的编码规则如下

字节 0		字节 1		字节 2...9
C	0	rA	rB	D

指令	mrmovq D(rB),rA	addq rA, rB	mraddq D(rB), rA
取指	icode:ifun←M_1[PC] rA:rB←M_1[PC+1] valC←M_8[PC+2] valP←PC+10	icode:ifun←M_1[PC] rA:rB←M_1[PC+1] valP←PC+2	设计思想： 方案 1：不改变现有指令阶段（机器周期）个数。如可在执行阶段完成地址运算、访存和加法，会增加执行电路的复杂度。 方案 2：增加指令执行阶段
译码	valB←R[rB]	valA←R[rA] valB←R[rB]	
执行	valE←valB+valC	valE←valB + valA Set CC	
访存	ValM←M_8[ValE]		
写回	R[rA]←valM	R[rB]←valE	
更新 PC	PC←valP	PC←valP	