| 主管领导审核签字 |
| --- |
| |

# 哈尔滨工业大学 2018 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　密　姓名　封　学号　线　院系

## 一、 单项选择题（每小题 1 分，共 20 分）

1. C 语言程序中的整数常量、整数常量表达式是在（　　）阶段变成 2 进制补码的。
   （A）预处理　（B）编译　（C）连接　（D）执行
2. C 语言程序如下，叙述正确的是（　　）

```
#include <stdio.h>
#define DELTA sizeof(int)
int main(){
  int i;
  for (i = 40;  i - DELTA >= 0; i -= DELTA)
    printf("%d ",i);
}
```

   A. 程序有编译错误
   B. 程序输出 10 个数：40 36 32 28 24 20 16 12 8 4 0
   C. 程序死循环，不停地输出数值
   D. 以上都不对
3. 下数值列叙述正确的是（　　）
   A.一条 mov 指令不可以使用两个内存操作数
   B.在一条指令执行期间，CPU 不会两次访问内存
   C.CPU 不总是执行 CS::RIP 所指向的指令，例如遇到 call、ret 指令时
   D.X86-64 指令"mov$1,%eax"不会改变%rax 的高 32 位
4. 条件跳转指令 JE 是依据（　　）做是否跳转的判断
   A. ZF　B. OF　C. SF　D. CF
5. 以下关于程序中链接“符号”的陈述，错误的是（　　）
   A.赋初值的非静态全局变量是全局强符号
   B.赋初值的静态全局变量是全局强符号
   C.未赋初值的非静态全局变量是全局弱符号
   D.未赋初值的静态全局变量是本地符号
6. 在 Y86-64 CPU 中有 15 个从 0 开始编码的通用寄存器，在对指令进行编码时，对于仅使用一个寄存器的指令，简单有效的处理方法是（　　）

A.用特定的指令类型代码
B.用特定的指令功能码
C.用特定编码 0xFF 表示操作数不是寄存器
D.无法实现

7. 采用缓存系统的原因是（ ）
A. 高速存储部件造价高 B.程序往往有比较好的空间局部性
C. 程序往往有比较好的时间局部性 D.以上都对

8. 关于动态库的描述错误的是（ ）
A.可在加载时链接，即当可执行文件首次加载和运行时进行动态链接。
B.更新动态库，即便接口不变，也需要将使用该库的程序重新编译。
C.可在运行时链接，即在程序开始运行后通过程序指令进行动态链接。
D.即便有多个正在运行的程序使用同一动态库，系统也仅在内存中载入一份动态库。

9. 内核为每个进程保存上下文用于进程的调度，不属于进程上下文的是（ ）
A.全局变量值 B.寄存器 C.虚拟内存一级页表指针 D.文件表

10. 不属于同步异常的是( )
A.中断 B.陷阱 C.故障 D.终止

11. 异步信号安全的函数要么是可重入的（如只访问局部变量）要么不能被信号处理程序中断，包括 I/O 函数（ ）
A. printf B. sprintf C. write D. malloc

12. 虚拟内存页面不可能处于（ ）状态
A.未分配、未载入物理内存 B. 未分配但已经载入物理内存
C.已分配、未载入物理内存 D. 已分配、已经载入物理内存

13. 下面叙述错误的是( )
A.虚拟页面的起始地址%页面大小恒为 0；
B.虚拟页面的起始地址%页面大小不一定是 0；
C.虚拟页面大小必须和物理页面大小相同；
D.虚拟页面和物理页面大小是可设定的系统参数；

14. 虚拟内存发生缺页时，正确的叙述是（ ）触发的
A. 缺页异常处理完成后，重新执行引发缺页的指令
B. 缺页异常处理完成后，不需要重新执行引发缺页的指令
C.缺页异常都会导致程序退出
D. 中断由 MMU 触发

15. 进程从用户模式进入内核模式的方法不包括（ ）
A.中断 B.陷阱 C.复位 D.故障

16. 程序语句"execve("a.out",NULL,NULL);"在当前进程中加载并运行可执行文件 a.out 时,错误的叙述是（ ）
A. 为代码、数据、bss 和栈创建新的、私有的、写时复制的区域结构
B. bss 区域是请求二进制零的，映射到匿名文件，初始长度为 0；
C. 堆区域也是请求二进制零的，映射到匿名文件，初始长度为 0；
D. 栈区域也是请求二进制零的，映射到匿名文件，初始长度为 0；

bss区域请求二进制零，但大小包含在a.out中。

17. 若将标准输出重定向到文本文件 file.txt，错误的是（ ）
A.需要先打开重定位的目标文件"file.txt"
B.设"file.tx"t 对应的 fd 为 4,内核调用 dup2(1,4)函数实现描述符表项的复制
C.复制"file.txt"的打开文件表项、并修正 fd 为 1 的描述符
D.修改"file.txt"的打开文件表项的引用计数

18. 关于局部变量，正确的叙述是（　　）
    A.普通（auto）局部变量也是一种编程操作的数据，存放在数据段
    B.非静态局部变量在链接时是本地符号
    C.静态局部变量是全局符号
    D.编译器可将 rsp 减取一个数为局部变量分配空间
19. 关于异常处理后返回的叙述，错误的叙述是（　　）
    A.中断处理结束后，会返回到下一条指令执行
    B.故障处理结束后，会返回到下一条指令执行
    C.陷阱处理结束后，会返回到下一条指令执行
    D.终止异常，不会返回
20. UNIX I/O 的 read、write 函数无法读/写指定字节的数据量，称为“不足值”问题，叙述正确的是(　　)
    A.读磁盘文件时遇到 EOF，会出现“不足值”问题
    B.写磁盘文件也会出现“不足值”问题
    C.读磁盘文件不会有这个问题
    D.以上均不对

## 二、填空题 ( 每空 1 分，共 10 分 )

21. 判断整型变量 n 的位 7 为 1 的 C 语言表达式是_____。
22. C 语言程序定义了结构体 struct noname{char c; int n; short k; char *p;};若该程序编译成 64 位可执行程序，则 sizeof(noname)的值是_____。
23. 整型变量 x=-2,其在内存从低到高依次存放的数是_____(16 进制表示)
24. 将 hello.c 编译生成汇编语言的命令行_____。
25. 程序运行时，指令中的立即操作数存放的内存段是：_____段。
26. 若 p.o->libx.a->liby.a 且 liby.a->libx.a->p.o 则最小链接命令行_____。
27. 在计算机的存储体系中，速度最快的是_____。
28. Cache 命中率分别是 97%和 99%时，访存速度差别_____(很大/很小？)。
29. 子程序运行结束会向父进程发送_____信号。
30. 向指定进程发送信号的 linux 命令是_____。

## 三、判断对错（每小题 1 分，共 10 分，正确打 √、错误打 ×）

31. （　）C 语言程序中，有符号数强制转换成无符号数时，其二进制表示将会做相应调整。
32. （　）在 Y86-64 的顺序结构实现中，寄存器文件写时是作为组合逻辑器件看待。
33. （　）链接时，若有一个强符号和多个弱符号同名，则对弱符号的引用均将被解析成强符号。
34. （　）异常处理程序运行在内核模式下，对所有的系统资源都有完全的访问权限。
35. （　）C 语言中数值从 int 转换成 double 后，数值虽然不会溢出，但有可能是不精确的。

授课教师____ 姓名____ 学号____ 院系____ 密 封 线

36. （ ）子进程即便运行结束，父进程也应该使用 **wait** 或 **waitpid** 对其进行回收。
37. （ ）在动态内存分配中，内部碎片不会降低内存利用率。
38. （ ）如果系统中程序的工作集大小超过物理内存大小，虚拟内存系统会产生抖动:页面不断地换进换出，导致系统性能暴跌。
39. （ ）虚拟内存系统能有效工作的前提是软件系统具有“局部性”。
40. （ ）相比标准 **I/O**，**Unix I/O** 函数是异步信号安全的，可以在信号处理程序中安全地使用。

## 四、简答题（每小题 5 分，共 20 分）

41. 从汇编的角度阐述：函数 **int sum(int x1,int x2,int x3,int x4,int x5,int x6,int x7,int x8)**，调用和返回的过程中，参数、返回值、控制是如何传递的？并画出 **sum** 函数的栈帧（**X86-64** 形式）。

42. 简述缓冲区溢出攻击的原理以及防范方法。

43. 简述 **shell** 的主要原理与过程。

44. 请结合 **ieee754** 编码，说明怎样判断两个浮点数是否相等？

## 五、系统分析题（20 分）

两个 C 语言程序 **main.c**、**test.c** 如下所示：

```
/* main.c */
#include <stdio.h>
int a[4]={-1,-2,2, 3};
extern int val;
int sum();
int main(int argc, char * argv[] )
{
    val=sum();
    printf("sum=%d\n",val);
}
```

```
/* test.c */
extern int a[];
int val=0;
int sum()
{
    int i;
    for (i=0; i<4; i++)
        val += a[i];
    return val;
}
```

用如下两条指令编译、链接，生成可执行程序 **test**：

```
gcc -m64 -no-pie -fno-PIC -c test.c main.c
gcc -m64 -no-pie -fno-PIC -o test test.o main.o
```

运行指令 **objdump -dxs main.o** 输出的部分内容如下：

```
 Contents of section .data:
 0000 ffffffff feffffff 02000000 03000000    ................
 Contents of section .rodata:
 0000 73756d3d 25640a00                      sum=%d..
…
 Disassembly of section .text:
0000000000000000 <main>:
       0:    55                      push   %rbp
       1:    48 89 e5                mov    %rsp,%rbp
       4:    48 83 ec 10             sub    $0x10,%rsp
       8:    89 7d fc                mov    %edi,-0x4(%rbp)
       b:    48 89 75 f0             mov    %rsi,-0x10(%rbp)
```

```
 f:    b8 00 00 00 00          mov    $0x0,%eax
14:    e8 00 00 00 00          callq  19 <main+0x19>
            15: R_X86_64_PC32    sum-0x4
19:    89 05 00 00 00 00       mov    %eax,0x0(%rip)    # 1f <main+0x1f>
            1b: R_X86_64_PC32    val-0x4
1f:    8b 05 00 00 00 00       mov    0x0(%rip),%eax    # 25 <main+0x25>
            21: R_X86_64_PC32    val-0x4
25:    89 c6                   mov    %eax,%esi
27:    bf 00 00 00 00          mov    $0x0,%edi
            28: R_X86_64_32  .rodata
2c:    b8 00 00 00 00          mov    $0x0,%eax
31:    e8 00 00 00 00          callq  36 <main+0x36>
            32: R_X86_64_PC32    printf-0x4
36:    b8 00 00 00 00          mov    $0x0,%eax
3b:    c9                      leaveq
3c:    c3                      retq
```

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

```
SYMBOL TABLE:
0000000000400400 l    d  .text   0000000000000000              .text
00000000004005e0 l    d  .rodata 0000000000000000              .rodata
0000000000601020 l    d  .data   0000000000000000              .data
0000000000601040 l    d  .bss    0000000000000000              .bss
0000000000000000         F *UND*  0000000000000000  printf@@GLIBC_2.2.5
0000000000601044 g     O .bss    0000000000000004              val
0000000000601030 g     O .data   0000000000000010              a
00000000004004e7 g     F .text   0000000000000039              sum
0000000000400400 g     F .text   000000000000002b              _start
0000000000400520 g     F .text   000000000000003d              main

Contents of section .rodata:
4005e0 01000200 73756d3d 25640a00                ....sum=%d..
…
Contents of section .data:
601020 00000000 00000000 00000000 00000000    ................
601030 ffffffff feffffff 02000000 03000000    ................
…
00000000004003f0 <printf@plt>:
 4003f0:ff 25 22 0c 20 00     jmpq   *0x200c22(%rip) # 601018 <printf@GLIBC_2.2.5>
 4003f6:68 00 00 00 00        pushq  $0x0
 4003fb:e9 e0 ff ff ff        jmpq   4003e0 <.plt>

Disassembly of section .text:
0000000000400400 <_start>:
  400400:  31 ed             xor    %ebp,%ebp
....
00000000004004e7 <sum>:
 4004e7:     55                 push   %rbp          #①
 4004e8:     48 89 e5           mov    %rsp,%rbp #②
 4004eb:     c7 45 fc 00 00 00 00 movl   $0x0,-0x4(%rbp) #③
 4004f2:eb 1e                   jmp    400512 <sum+0x2b>
 4004f4:8b 45 fc                mov    -0x4(%rbp),%eax
 4004f7:48 98                   cltq
 4004f9:8b 14 85 30 10 60 00 mov    0x601030(,%rax,4),%edx
 400500:     8b 05 3e 0b 20 00      mov    0x200b3e(%rip),%eax #601044 <val>
 400506:     01 d0                  add    %edx,%eax
 400508:     89 05 36 0b 20 00      mov    %eax,0x200b36(%rip) #601044 <val>
```

```
40050e:     83 45 fc 01          addl   $0x1,-0x4(%rbp)
400512:     83 7d fc 03          cmpl   $0x3,-0x4(%rbp)#④
400516:     7e dc                jle    4004f4 <sum+0xd>#⑤
400518:     8b 05 26 0b 20 00    mov    0x200b26(%rip),%eax # 601044 <val>
40051e:     5d                   pop    %rbp
40051f: c3                   retq

0000000000400520 <main>:
400520:     55                   push   %rbp
400521:     48 89 e5             mov    %rsp,%rbp
400524:     48 83 ec 10          sub    $0x10,%rsp
400528:     89 7d fc             mov    %edi,-0x4(%rbp)
40052b:     48 89 75 f0          mov    %rsi,-0x10(%rbp)
40052f: b8 00 00 00 00       mov    $0x0,%eax
400534:     e8(    ①    )        callq    4004e7 <sum>
400539:     89 05(      ②    )    mov    %eax,  ■■■■■(%rip) #601044<val>
40053f: 8b 05(    ③    )    mov    ■■■■■(%rip),%eax    #601044<val>
400545:     89 c6                mov    %eax,%esi
400547:     bf（   ④   ）    mov       ■■■■■,%edi
40054c:     b8 00 00 00 00   mov       $0x0,%eax
400551:     e8（   ⑤   ）    callq     4003f0 <printf@plt>
400556:     b8 00 00 00 00   mov       $0x0,%eax
40055b:     c9                   leaveq
40055c:     c3                   retq
40055d:     0f 1f 00             nopl    (%rax)
```

45. 阅读的 sum 函数反汇编结果中带下划线的汇编代码（编号①-⑤），解释每行指令的功能和作用（5 分）

46. 根据上述信息，链接程序从目标文件 test.o 和 main.o 生成可执行程序 test，对 main 函数中空格①--⑤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 2 个即可）。（5 分）

47. 在 sum 函数地址 4004f9 处的语句"mov 0x601030(,%rax,4),%edx"中，源操作数是什么类型、有效地址如何计算、对应 C 语言源程序中的什么量(或表达式)？其中，rax 数值对应 C 语言源程序中的哪个量(或表达式)？如何解释数字 4？（5 分）

48. 一个 C 程序的 main()函数如下：

```
int main（）
{
  if(fork()==0){
    printf("a");     fflush(stdout);
    exit(0);
  }
  else{
    printf("b");     fflush(stdout);
    waitpid(-1,NULL,0);
  }
  printf("c");       fflush(stdout);
```

```
    exit(0);
}
```

**48.1** 请画出该程序的进程图

**48.2** 该程序运行后，可能的输出数列是什么？

院系_____ 学号_____ 姓名_____ 授课教师_____

密 封 线

## 六、综合设计题（共 20 分）

49. 为 Y86-64 CPU 增加一指令"iaddq V,rB" ，将常量数值 V 加到寄存器 rB。参考 irmovq、OPq 指令，请设计 iaddq 指令在各阶段的微操作。(10 分)

| 指令 | irmovq V,rB | OPq rA, rB | iaddq V,rB |
|---|---|---|---|
| 取指 | icode:ifun←M1[PC]<br>rA:rB←M1[PC+1]<br>valC←M8[PC+2]<br>valP←PC+10 | icode:ifun←M1[PC]<br>rA:rB←M1[PC+1]<br>valP←PC+2 | |
| 译码 | valB←0 | valA←R[rA]<br>valB←R[rB] | |
| 执行 | valE←valB+valC | valE←valB OP valA<br>Set CC | |
| 访存 | | | |
| 写回 | R[rB]←valE | R[rB]←valE | |
| 更新 PC | PC←valP | PC←valP | |

50. 现代超标量 CPU X86-64 的 Cache 的参数 s=5，E=1，b=5，若 M=N=64，请优化如下程序，并说明优化的方法（至少 CPU 与 Cache 各一种）。

```
void trans(int M, int N, int A[M][N], int B[N][M])
 {
     for (int i = 0; i < M; i++)
          for (int j = 0; j < N; j++)
               B[j][i] =A[i][j];
}
```