

2023 计算机系统 2 期中考试答案 (2-3 章)

姓名: _____ 学号: _____ 分数: _____

第二章相关内容:

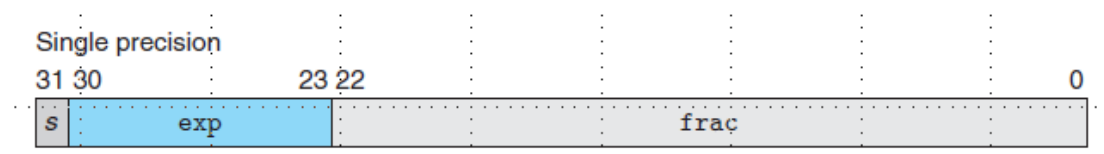
1. 请写出 int 类型最大值、最小值、-1 和 0 值的十六进制表示, unsigned short 类型的最大值、最小值的二进制表示。(5 分)

答:

- Int 类型:
 - 最大值: 0x7FFFFFFF
 - 最小值: 0x80000000
 - -1: 0xFFFFFFFF
 - 0: 0x00000000
- Unsigned short 类型:
 - 最大值: 0xFFFF
 - 最小值: 0x0000

2. 请写出单精度浮点数的“非负值最小规格化数”的小数表示和“最小非规格化数”的二进制表示(单精度浮点数的阶码字段占用 8 位)。(5 分)

答:



- 单精度浮点数的“非负值最小规格化数”的小数表示: 1.0×2^{-126}
 - 因为是非负值, 所以符号位 $s=0$;
 - 规格化数要求阶码 exp 不为 0 和 255, 所以最小规格化数的阶码 exp 为 0x01;
 - 尾数 frac 最小可为 0;
 - 则小数表示为 $1.0 \times 2^{1-127} = 1.0 \times 2^{-126}$ 。
- 单精度浮点数的“最小非规格化数”的二进制表示:
 - 因为没有要求为非负值, 则符号位 s 可为 1;
 - 非规格化数, 所以 exp 为 0x00000000;
 - 由于符号位为 1, 即此浮点数为负数, 因此尾数要最大, 才能使得该浮点数最大, 则 frac 有 23 个 1 构成;
 - 因此二进制表示为: 1 00000000 111111111111111111111111。

3. 写出 8 位浮点数 (阶码采用 4 位, 小数位采用 3 位) “0 0110 110”所表示的数值。(5 分)

答:

- 符号位为 0, 则该浮点数为非负值;
- 阶码为 0110, 偏置值是 0111, 则阶数为 $0110 - 0111 = 1111$, 即 -1
- 尾数为 110, 则“0 0110 110”所表示的数值为 $1.110 \times 2^{-1} = 0.1110$ (二进制表示), 其 10 进制表示为 0.875

4. 现有代码：`int i=0xab cd ef 01;`
`short si=i;`
请问代码执行后，变量 `si` 的数值表示为（十六进制）？（5 分）

答：直接截断取低 16 位，因此 `si` 的数值表示为 `0xef 01`。

5. 如果 `int i=0x86 23 11 32`，`i` 的地址为 `0x400320`，请问地址 `0x400322` 所对应内存上的那个字节存储的数值是？

答：因为 `int` 类型是多字节序列，需要讨论大端、小端的情况：

● 大端：

地址	0x400320	0x400321	0x400322	0x400323
值	0x86	0x23	0x11	0x32

地址 `0x400322` 所对应内存上的那个字节存储的数值是为 `0x11`。

● 小端：

地址	0x400320	0x400321	0x400322	0x400323
值	0x32	0x11	0x23	0x86

地址 `0x400322` 所对应内存上的那个字节存储的数值是为 `0x23`。

6. 在 x86 机器上有如下代码 “`int a=15;`”且已知 `a` 的地址为 `0x40030`，请说明存储变量 `a` 需要的字节数 `n`，以及从 `0x40030` 开始的 `n` 个字节上存储什么内容？如果在 IBM 的 `power8` 处理器这样的大端机器上，各个字节又是什么内容？

`n=4`

在 X86(小端)机器上，各字节存储 `0x0F,0x00,0x00,0x00` 内容。

在 IBM 的 `power8` 处理器这样的大端机器上，各字节存储 `0x00,0x00,0x00,0x0F`

7. 请判定 C 语言表达式 `-2147484647 - 1U < 2147484647` 取值。

`True`

// 解释

因为 `int` 类型的最小值是 `-2147483647-1`，而题目给的数值超过 `int` 类型表示范围，因此是 `long` 类型，`1U` 表示该常数按照 `unsigned int` 存储，按照 C 语言隐式转换规则，此表达式两边转换成 `long` 类型比较。因此答案是 `true`。

8. 有如下代码：`char x= - 8; unsigned int y=x;`请问 `y` 的二进制位模式是什么？数值为多少？

`1111 1111 1111 1111 1111 1111 1000`

10 进制数值是 `4294967288`

// 解释

16 进制是 `0Xff ff ff f8`

根据书本 56 页内容，本题的隐式转换可以写为 `y=(unsigned int)((int) x);`

9. 有以下代码 unsigned int x=15; int y=- 4; x>>2;y>>2; 请问最后 x 和 y 的数值为多少?

x = 3 y = -1

// 解释

没有特别说明, 有符号数默认做算术右移, 无符号数做逻辑右移

10. 有以下代码 unsigned char x=128; x=x+x; 请问 x 最后的二进制位模式是什么?

0000 0000

// 解释

此运算发生溢出, 得到的结果需要%2⁸, 因此是 0

11. 将 int x 变量完成 x*30 中的乘法, 用移位运算来加速, 请给出具体表达式。

(x<<4) + (x<<3) + (x<<2)+ (x<<1) 或者 (x<<5)-(x<<1)

// 解释

需要添加括号, +优先级比<<高。

对于第二个答案, 有些 bug。假设如果类型是 m 位, 需要上述操作, 但是 4 改成 m 位, 也就是说后面移 m+1 会大于最大可移位的数值(m), 这个时候实际移位(m+1)%m, 与想象的不符。书本 70-73 页有相关内容。

12. 请用移位方式实现有符号数 x 除以 4 的计算过程, 写出等效的 C 代码。

x>>2

13. 请写出单精度规格化小数中, 负的绝对值最小值的二进制位模式, 写出数值 1 的位模式。

负的绝对值最小值 1 00000001 000 0000 0000 0000 0000 0000

数值 1 的位模式 0 01111111 000 0000 0000 0000 0000 0000

14. 完成以下两个单精度浮点数的求和运算

0 01111111 000000000000000000000000+0 01111100 011000000000000000000000。

以及

0 01111111 0000000000000000000001111+0 01111111 011000000000000000010101

0 01111111 000000000000000000000000+0 01111100
011000000000000000000000=1+**1.71875E-1=1.171875**
0 01111111 0000000000000000000001111+0 01111111
011000000000000000010101=**1.00000178813934326171875+**

1.37500250339508056640625=2.375004291534424

$0\ 01111111\ 000\ 0000\ 0000\ 0000\ 0000\ 0000$
 $0\ 01111100\ 011\ 0000\ 0000\ 0000\ 0000\ 0000$
 $\swarrow 2^0 \times 1.0$
 $\swarrow 2^{-3} \times 1.011$
 对齐 $2^0 \times 1.0 + 2^0 \times 0.001011$
 尾数相加 $\begin{array}{r} 1.0 \\ + 0.001011 \\ \hline 1.001011 \end{array}$
 规格化处理 $2^0 \times 1.001011$
 无舍入, 即答案是 $0\ 01111111\ 001\ 0110\ 0000\ 0000\ 0000\ 0000$

$0\ 01111111\ 000\ 0000\ 0000\ 0000\ 0000\ 1111$
 $0\ 01111111\ 011\ 0000\ 0000\ 0000\ 0001\ 0101$
 $\swarrow 2^0 \times 1.000\ 0000\ 0000\ 0000\ 0000\ 1111$
 $\swarrow 2^0 \times 1.011\ 0000\ 0000\ 0000\ 0001\ 0101$
 不需要对齐, 尾数相加
 $\begin{array}{r} 1.000\ 0000\ 0000\ 0000\ 0000\ 1111 \\ + 1.011\ 0000\ 0000\ 0000\ 0001\ 0101 \\ \hline 10.011\ 0000\ 0000\ 0000\ 0010\ 0100 \end{array}$
 规格化 $2^1 \times 1.0011\ 0000\ 0000\ 0000\ 0010\ 0100$
 舍入 $2^1 \times 1.001\ 1000\ 0000\ 0000\ 0001\ 0010$
 答案是 $0\ 1000\ 0000\ 001\ 1000\ 0000\ 0001\ 0010$

// 解释

转换成同阶 (小的往大的转), 再计算尾数 (注意小数点对齐), 可能出现舍入, (舍去的数等于 0.5 使用偶数舍入, 其它情况使用向上舍入或者向下舍入)

15. 对于 int b[10][5], 有如下代码

```

for(i=0;i<10;i++)
    for(j=0;j<5;j++)
        sum+=b[i][j]
    
```

假设执行到 sum+=... 时, sum 在 eax 中, b[i][0] 所在的地址在 rdx, j 在 esi, 则 sum+=b[i][j] 对应的指令可以是 ()

- (a) addl 0(%rdx,%esi,4),%eax (b) addl 0(%esi,%rdx,4),%eax
 (c) addl 0(%rdx,%esi,2),%eax (d) addl 0(%esi,%rdx,2),%eax

答案: a

16. IA-32 指令, pushl %ebp 对应的指令 ():

- (a) $R[esp] \leftarrow R[esp]-4, M(R[esp]) \leftarrow R[ebp]$
 (b) $R[esp] \leftarrow R[esp]+4, M(R[esp]) \leftarrow R[ebp]$
 (c) $M(R[esp]) \leftarrow R[ebp], R[esp] \leftarrow R[esp]-4$
 (d) $M(R[esp]) \leftarrow R[ebp], R[esp] \leftarrow R[esp]+4$

答案: a

17. 写出下面 fun_ifelse()的汇编代码, a 变量地址为 addr_a

```
int a;
int fun_ifelse(short b)
{
    int c;
    a=rand();
    b=rand();
    if (a>b)
    {
        c=a+b;
        if(a>0)          c=c+3;
        else             c=c+5;
    }else c=a*4;
    return 0;
}
```

```
fun_ifelse:
    pushq    %rdx          # int c
    pushq    %rsi
    xorq     %rdx,%rdx
    callq    rand
    movl     %eax,%esi      # a = rand()
    callq    rand
    xorq     %rdi,%rdi      # b = rand()
    movw     %ax,%di        # C语言隐式转换成int类型比较
    cmpl     %esi,%edi
    jge      L1
    movl     %esi,%edx
    addl     %edi,%edx
    cmpl     $0,%esi
    jle      L2
    addl     $3,%edx
    jmp      L3
.L2:
    addl     $5,%edx
    jmp      L3
.L1:
    shll     $2,%esi        # a=a*4
    movl     %esi,%edx      # c=a
.L3:
    xorq     %rax,%rax
    popq     %rsi
    popq     %rdx
    ret
```

// 解释

图片给出的答案有 bug, a 和 b 比较的时候需要考虑隐式类型转换, movswl...

下面是在 64 位 Linux 机器给出的代码 (objdump -d funifelse.o)

```
0000000000000000 <fun_else>:
 0:  48 83 ec 08          sub    $0x8,%rsp
 4:  b8 00 00 00 00      mov    $0x0,%eax
 9:  e8 00 00 00 00      callq e <fun_else+0xe>    需要重定位到rand函数
e:  89 05 00 00 00 00    mov    %eax,0x0(%rip)      # 14 <fun_else+0x14>
14: b8 00 00 00 00      mov    $0x0,%eax
19: e8 00 00 00 00      callq 1e <fun_else+0x1e>
1e: b8 00 00 00 00      mov    $0x0,%eax
23: 48 83 c4 08          add    $0x8,%rsp
27:  c3                  retq
```

可以发现, 对于 c 的操作被省略了, 因为函数不返回 (这是开了 -Og 优化选项的结果,)

18. 写出下面 for_fun ()的汇编代码

// 这里原本是 int loop,&sum, 改为下面的参数形式

```
int for_fun(int loop, int sum)
```

```
{
    int k;
    for(k=10;k<100;k++)
    {
        sum+=k;
    }
    return sum;
}
```

```
for_fun:
    pushq    %rdx
    movq     %rsi,%rax # rsi是int &sum
    movl     $10,%edx
    jmp      L2
.L1:
    addl     %edx,%eax
    addl     $1,%edx
.L2:
    cmpl     $100,%edx
    jl       L1
    popq     %rdx
    ret
```

//解释

下面是在 64 位机器下 (objdump -d forfun.o) 的结果

```
0000000000000000 <for_fun>:
   0:  b8 0a 00 00 00      mov     $0xa,%eax
   5:  eb 05              jmp     c <for_fun+0xc>
   7:  01 c6              add     %eax,%esi
   9:  83 c0 01           add     $0x1,%eax
  c:  83 f8 63           cmp     $0x63,%eax
  f:  7e f6             jle     7 <for_fun+0x7>
 11:  89 f0             mov     %esi,%eax
 13:  c3               retq
```

19. (控制) 写出下面函数 Func1 汇编代码对应的 C 程序, 其中参数 1 为 x, 参数 2 为 y:

Func1:

```
    cmpq    %rsi, %rdi
    jge .L2
    leaq    3(%rsi), %rdi
    jmp .L3
.L2:
    leaq    (%rdi,%rdi,4), %rsi
    addq    %rsi, %rsi
.L3:
```

```
leaq    (%rdi,%rsi), %rax
ret
```

答:

```
long func1(long x, long y)
{
    if (x < y) // compq %rsi, %rdi
    {
        x = y + 3; // leaq 3(%rsi), %rdi
    }
    else
    {
        y = 5 * x; // leaq (%rdi,%rdi,4), %rsi; %rdi + %rdi * 4
        y += y;    // addq
    }
    return x + y; // leaq (%rdi, %rsi), %rax
}
```

20. (多重数组+lea 指令) 对于数组 int B[8][5], 需要将 B[i][j] 保存到 eax 中, 数组起始地址保存在 rdi, i 保存在 rsi, j 保存在 rdx 中, 请完成以下代码中的空缺
- ```
leaq (,%rsi,), %rax
leaq (, ,), %rax
movl (, ,), %eax
```

答:

B[i][j] 等价于  $*(B + 5*i + j)$

```
leaq (%rsi ,%rsi, 4), %rax // %rax = 5*i
leaq (%rax ,%rdx, 1), %rax // %rax += j
movl (%rdi , %rax , 4), %eax // %eax = *(B + %rax)
```

21. (数组+函数+乘法的移位实现) 已知 int P[M][N] 和 int Q[N][M], 有以下函数:

```
int addfun(int i,int j){
 return P[i][j]+Q[j][i];
}
```

对应有汇编代码如下, 请问 M 和 N 分别是多少?

addfun:

```
movl %edi, %edx
shl $2,%edx
addl %esi,%edx
movl %esi,%eax
```

```

shll $2,%eax
addl %eax,%edi
movl Q(,%rdi,4),%eax
addl P(,%rdx,4), %eax
ret

```

答:

addfun:

```

movl %edi, %edx # %edx = i
shl $2,%edx # %edx = 4*i
addl %esi,%edx # %edx = 4*i + j
movl %esi,%eax # %eax = j
shll $2,%eax # %eax = 4*j
addl %eax,%edi # %edi = 4*j + i
movl Q(,%rdi,4),%eax # %eax = *(Q + 4*j + i)
addl P(,%rdx,4), %eax # %eax += *(P + 4*i + j)
ret

```

由上可知 M 和 N 均为 4

22. (union+结构体)

```

union a1{
 struct {int * b1; char c1; long d1 } str1;
 double data[3];
}

```

请问按照默认的对齐方式，上述 a1.str1 占用多少字节空间？ a1 占用多少字节空间？

a1.str1:



共 24 字节。

a1.data:



同样是共 24 字节。



即 a1.str1 占用 24 个字节。union 中需要最大的空间为 24 字节，则 a1 占用 24 字节。

23. （结构体+函数+控制）已知 node 结构体定义如下

```
struct node{
 long a;
 struct node *next;
}
```

请对以下 init 函数进行逆向分析，写出其 C 代码

Init:

```
 movl $12,%eax
 jmp .TestExprStat
.Loop:
 addq (%rdi),%rax
 movq 8(%rdi),%rdi
.TestExprStat:
 testq %rdi,%rdi
 jne .Loop
 ret
```

答:

```
long init(struct node *p) {
 long res = 12;
 while(p) {
 res += p->a;
 p = p->next;
 }
 return res;
}
```

24. （结构体）已知结构体定义如下

```
struct{
 char a;
 char *b;
 short c;
 int d;
}
```

请问在紧凑布局和对齐布局中 a/b/c/d 字段的偏移量各是多少？

答：

(这里的答案是指 64 位环境下的，因此指针大小为 64bit，即 8 字节)

| 字段名 | 类型     | 类型大小(字节) | 紧凑布局偏移 | 对齐布局偏移 |
|-----|--------|----------|--------|--------|
| a   | char   | 1        | 0      | 0      |
| b   | char * | 8        | 1      | 8      |
| c   | short  | 2        | 9      | 16     |
| d   | int    | 4        | 11     | 20     |

### 32 位环境

| 字段名 | 类型     | 类型大小(字节) | 紧凑布局偏移 | 对齐布局偏移 |
|-----|--------|----------|--------|--------|
| a   | char   | 1        | 0      | 0      |
| b   | char * | 4        | 1      | 4      |
| c   | short  | 2        | 5      | 8      |
| d   | int    | 4        | 7      | 12     |

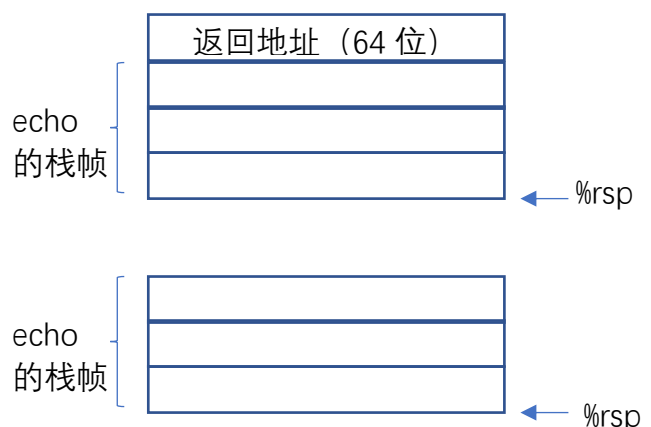
25. (堆栈破坏问题) 函数 echo 定义如下：

```
void echo(){
 char buf[8];
 gets(buf);
 puts(buf);
}
```

对应的汇编代码如下：

echo:

```
subq $24,%rsp
movq %fs:40,%rax
movq %rax,8(%rsp)
xorl %eax,%eax
movq %rsp,%rdi
call gets
movq %rsp,%rdi
call puts
movq 8(%rsp),%rax
xorq %fs:40,%rax
je .L9
call __stack_chk_fail
addq $24,%rsp
ret
```



观察代码，判定该函数是否具有堆栈破坏的检测能力？如果 %fs:38 地址开始存放了 0x00/01/02/03/04/05/06/07/08/09/0a/0b/0c/0d/0e/0f。请问刚进入 echo 函数

时，echo 栈帧中%rsp 位置存放的 8 字节数值是？如果此时输入按键 abcdefg 并回车，程序将如何执行？如果此时输入按键 123456789 并回车，程序能否正常返回？如果不能将执行什么处理？

答：

- 具有堆栈破坏的检测能力，因为在调用 gets 前，在传入的 buf 后面设置了一个金丝雀值，并在 echo 函数退出前，检查了该值是否被修改；
- 刚进入 echo 函数时，echo 栈帧中%rsp 位置存放的 8 字节数值是 echo 的返回地址；
- 因为“abcdefg”只有 7 个字符，加上“\0”刚好 8 个字符，buf 刚好为大小为 8 的字符数组，因此正常执行，输出 abcdefg；
- 因为“123456789”含有 9 个字符，因此在调用 gets 时会出现缓冲区溢出的现象，因此修改了金丝雀值，故会调用“\_\_stack\_chk\_fail”，不能正常返回。

26. （函数参数+浮点）对于一下汇编代码，请写出对应的 C 函数代码（整数参数请使用 a/b，浮点参数请使用 c）

myfun:

```
movsbl %dil, %edi
imull $30, %edi, %edi
addl (%rsi), %edi
movl %edi, (%rsi)
cvtsi2ss %edi, %xmm1
addss %xmm1, %xmm0
ret
```

答：

- movsbl：进行符号扩展，将 1Byte 符号扩展成 4Byte
- cvtsi2ss：将一个有符号整数转换为一个单精度浮点数
- addss：浮点数加法
- %xmm0：为第一个浮点参数和单精度浮点数类型的返回值，同时作为参数和返回值

```
float myfunc(char a, int *b, float c) {
 *b = *b + 30 * a;
 return c + *b;
}
```

或：

```
float myfun(char a, int *b, float c)
{
 int x = a; // movsbl
 int t = a * 30;
 int t = *b + t;
 float res = t;
 return res + c;
}
```