姓名：　　班级：　　学号：

X86 部分

一、填空题（18 分）

1） 已知某 32 位整数 X，其值为-101，则其 16 进制补码为 0XFFFFFF9B，另一 32 位整数 Y 的补码为 0xFFFFFF63，则 X+Y 的 16 进制补码(32 位)为 0XFFFFFEFE，X-Y 的 16 进制补码为 0X38。

2） X86 的 32 位浮点数的 exp 位数是 8，frac 位数是 23；64 位浮点数的 exp 位数是 11，frac 位数是 52。

3） 给出 11/4 这一数字的 32 位浮点数表示，即 exp= 10000000；frac= 01100…000。

4） 假设存在一种 16 位的浮点数表示，exp 位数是 7，frac 位数是 8，符号位数为 1，其所能表示的最大的非规格化数的 exp 是 0000000，frac 是 11111111；255 的 exp 是 1000110，frac 是 11111110。

5） 在 X86-32 位体系结构中，当前运行函数的帧（Frame）基址寄存器所指向的栈地址的“上方”（高地址）由低到高存放的是函数返回地址、传入参数；“下方”存放的是 局部变量、寄存器保留值（或子过程参数）（此处无需考虑顺序）。

二、简答题 （42 分）

1）X86 32 位体系结构中的条件跳转指令 jg 是用于符号数比较还是无符号数比较的？其产生跳转的成立条件是~(SF^OF)&~ZF 为真，请解释为何是这一条件。（4 分）

符号数比较

SF：结果正负，SF=1,表示运算结果为负数

OF：溢出标志位，OF=1，表示溢出

ZF：结果是否为零，ZF=1，表示结果为 0

即当 SF=OF 且 ZF=0 时，进行跳转

如果前后两数都为正，不发生溢出，则 SF=0，OF=0

如果前后两数都为负，不发生溢出，则 SF=0，OF=0

如果前正后负，则可发生溢出，不发生溢出则为 SF=0，OF=0；否则为 SF=1，OF=1

所以为 SF 和 OF 需要同或，最后需要保证前后不相等，ZF=0

2）下图给出了一个 C 函数，并由 gcc 编译成相应的汇编代码（AT&T 语法格式），请补全这段代码里头被省去的部分。（32 位 X86 代码，10 分）

```
int arith(int x, int y, int z)
{
    int t1= x+y;
    int t2 =z+t1;
    int t3=x+4;
    int t4=y*48;
    int t5=t3+t4;
    int rval=t2*t5;
    return rval;
}
```

编译出的代码：

```
…
movl    8(%ebp),%eax
movl    12 (%ebp),%edx
leal    (% edx , %eax),%ecx
leal    (%edx,%edx,2),%edx
sall    $4 ,%edx
addl    16 (%ebp), %ecx
leal    4(%edx,%eax),%eax
imull   %ecx,%eax
…
```

3）C语言中过程的参数个数可以是不固定的。比如定义了如下能够产生格式化输出的过程：

```
void my_printf(const char *fmt, ...);
```

其参数个数大于等于 1,第一个参数是一个格式字符串,可以接受形如“input string %d %d”之类的字符串作为输入，其中%d 指定输出 32 位带符号整数，具体的输出整数值则由后续的参数指定（为简化起见，这个函数只能接受%d 作为格式转换）。这个函数的汇编代码如下所示，请分析这些代码并回答如下问题：（11 分）

- 这类不定参数的过程是如何传入参数的？
  通过栈传递，从右至左压栈；8(%ebp)是字符串地址，12(%ebp)为变参的起始地址

- my_printf是如何确定不定参数个数的？
  从字符串起始地址开始依次扫描每个字符，统计出现的%d个数。

```
    .section .rdata,"dr"
LC0:
    .ascii "%d\0"
    .text
.globl _my_printf
_my_printf:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $16, %esp

    pushl   %esi
    leal    12(%ebp), %esi
    pushl   %ebx
    movl    8(%ebp), %ebx
    movzbl  (%ebx), %eax
    testb   %al, %al
    je      L11
L14:
    cmpb    $37, %al #'%'的ascii码值是37
    je     L5
    movsbl  %al,%eax
L8:
    movl    %eax, (%esp)
    incl       %ebx
    call      _putchar
```

```
L15:
    movzbl  (%ebx), %eax
    testb   %al, %al
    jne L14
L11:
    addl    $16, %esp
    popl    %ebx
    popl    %esi
    popl    %ebp
    ret
L5:
    incl       %ebx
    movsbl  (%ebx),%eax
    cmpl    $100, %eax
    jne L8
    movl    %esi, %eax
    incl       %ebx
    movl    (%eax), %eax
    addl    $4, %esi
    movl    $LC0, (%esp)
    movl    %eax, 4(%esp)
    call    _printf
    jmp     L15
```

4）在 X86-32 位编程中有一种简单的获得所运行的指令地址的方法（X86-32 位结构下 eip 寄存器是无法直接访问的）。比如说我们要获得下面程序中 XXXX 这条指令的地址(并将其置于 eax 寄存器中)，那么可以采用如下代码段。请补充完函数 GetAddress 的第一条语句(AT&T 语法)，以及说明这么实现的理由（5 分）。

```
call GetAddress
XXXX
```

```
GetAddress:
    movl    (%esp)    ,    %eax
    ret
```

5）已知三个二维矩阵的定义如下，并已初始化。

```
#define n 10
int a[n][n] ;
int b[n][n] ;
int c[n][n] ;
```

需要进行矩阵乘，即矩阵 a x b 结果置于 c。下面这段 C 代码是一个矩阵乘函数。

```
void column()
{
    int j, k , i;
    int r;

    for (j=0; j<n; j++) {
```

```
        for (k=0; k<n; k++) {
            r = b[k][j];
            for (i=0; i<n; i++)
                c[i][j] += a[i][k] * r;
        }
    }
}
```

编译完的汇编代码如下——

```
_matrix:
    pushl   %ebp
    movl    %esp , %ebp
    pushl   %edi
    pushl   %esi
    pushl   %ebx
    subl    $8, %esp
    movl    $0, -16(%_ebp_)
L13:
    movl    -16(%ebp), %eax
    xorl    %edi,  %edi
    leal    b_start_addr(, %eax ,4),%eax
    movl    %eax , -20(%ebp)
L12:
    movl    -20(%ebp), %edx
    xorl    %ecx,  %ecx
    movl    $9, %ebx
    movl    (%edx), %esi
```

```
L11:
    movl    -16(%ebp), %edx
    leal    (%ecx,%edx), %eax
    leal    (%ecx,%edi), %edx
    movl    a_start_addr (, %edx ,4), %edx
    addl    $10, %ecx
    imull   %esi , %edx
    addl    %edx, c_start_addr (,%eax,4)
    decl    %ebx
    jns L11
    addl    $40, -20(%ebp)
    incl    %edi
    cmpl    $9 , %edi
    jle L12
    incl    -16(%ebp)
    cmpl    $9 , -16(%ebp)
    jle L13
    addl    $8 , %esp
    popl    %ebx
    popl    %esi
    popl    %edi
    popl    %ebp
    ret
```

请填出代码中的空缺部分（12 分）。

MIPS 部分（10 分）

一、 简答题

1） 请说出 MIPS 指令集不同于 X86 指令集的四处地方。（4 分）

MIPS 寄存器数量远多于 X86

MIPS 指令是等长的，而 X86 是变长的

MIPS 只支持单一寻址方式

MIPS（load/store 指令和）跳转指令时有延迟槽

MIPS 的读取存储指令要求地址 4 个 byte 对齐……

2）补全下列代码：（6 分）

2.1）lw $t6, 65536($sp)经过MIPS汇编器处理后，产生的代码如下，请补全。

```
lui $1,    1
addu $1, $1,    $sp
lw $14, 0($1)     #t6即为14号寄存器;
```

2.2）li $6, 0x345678 经过MIPS汇编器处理后，产生的代码如下，请补全

```
lui   $1,    0x34
 addiu      $6, $1,   0x5678
```

（还有其他写法。。。）