**t作业G** 等级：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学号：\_1929409021\_ 姓名:\_\_胡志彬\_\_\_\_

[F1]如果采用一个字节表示地址差，那么无条件转移指令的转移的范围是\_-128-+127\_\_，条件转移指令的转移范围是\_\_\_\_-128-+127\_\_\_\_。

[F2]按照转移目标地址是否涉及代码段寄存器CS，转移可以分为段内转移和段间转移\_两类。

[F3]按照表示转移目标地址的方法，无条件转移指令可以分为\_直接转移\_和\_间接转移\_两类。

[S1]以下关于直接转移和间接转移的叙述，正确的是（多选）？ABCD

（A）直接转移指令自身含有距离目标地址的位移量或者目标地址。

（B）间接转移指令利用存储单元或通用寄存器给出转移目标地址。

（C）直接转移指令，使用方便。

（D）间接转移指令，转移灵活。

[S2]以下关于相对转移方式的叙述，正确的是（多选）？AC

（A）相对转移方式有利于程序的浮动。

（B）无条件转移指令都采用相对转移方式。

（C）条件转移指令都采用相对转移方式。

（D）直接转移指令都采用相对转移方式。

[S3]以下关于无条件转移的叙述，正确的是（多选）？AB

（A）无条件转移指令分为直接和间接两类。

（B）无条件转移指令有段内转移和段间转移的区分。

（C）无条件转移指令都是近转移。

（D）无条件转移指令都不采用相对转移方式。

[R1]请阅读如下程序，说明其功能：

#include <stdio.h>

int main()

{ char cha; int count;

printf("input:"); cha = getchar();

\_asm {

XOR ECX, ECX

MOV AL, cha

LAB:SHL AL, 1

ADC CL, 0

OR AL, AL

JNZ LAB

MOV count, ECX

}

printf("%d\n", count);

return 0;

}

统计用户输入字符 `cha` 的二进制表示中 \*\*1的个数（汉明重量）\*\*，结果存入 `count` 并输出。

[R2]设有如下C函数：

int ftest(int x, int y)

{

unsigned z;

if (y > 2\*x)

z = 1;

else if (y == 2\*x)

z = 0;

else

z = -1;

return z;

}

在VC2010环境中，禁止编译优化情形下，可得到如下的目标代码，请说明每条汇编格式指令的作用（可以利用高级语言语句或表达式来描述）。

push ebp

mov ebp, esp

push ecx

mov eax, DWORD PTR [ebp+8]

shl eax, 1

cmp DWORD PTR [ebp+12], eax

jle SHORT LN4@ftest

mov DWORD PTR [ebp-4], 1

jmp SHORT LN3@ftest

LN4@ftest:

mov ecx, DWORD PTR [ebp+8]

shl ecx, 1

cmp DWORD PTR [ebp+12], ecx

jne SHORT LN2@ftest

mov DWORD PTR [ebp-4], 0

jmp SHORT LN3@ftest

LN2@ftest:

mov DWORD PTR [ebp-4], -1

LN3@ftest:

mov eax, DWORD PTR [ebp-4]

mov esp, ebp

pop ebp

ret

push ebp ; 保存旧栈帧基址

mov ebp, esp ; 建立新栈帧

push ecx ; 保护ECX

mov eax, [ebp+8] ; eax = 参数x

shl eax, 1 ; eax = 2\*x

cmp [ebp+12], eax ; 比较 y 和 2\*x

jle LN4@ftest ; if (y <= 2\*x) 跳转到LN4

mov [ebp-4], 1 ; z = 1

jmp LN3@ftest ; 跳转到返回前

LN4@ftest:

mov ecx, [ebp+8] ; ecx = x

shl ecx, 1 ; ecx = 2\*x

cmp [ebp+12], ecx ; 比较 y 和 2\*x

jne LN2@ftest ; if (y != 2\*x) 跳转到LN2

mov [ebp-4], 0 ; z = 0

jmp LN3@ftest ; 跳转到返回前

LN2@ftest:

mov [ebp-4], -1 ; z = -1

LN3@ftest:

mov eax, [ebp-4] ; 返回值 = z

mov esp, ebp ; 恢复栈指针

pop ebp ; 恢复旧栈帧基址

ret ; 返回

[Q1]参考3.3节的例6，删除函数cf319中case 4及以上的分支情形，然后编译生成目标代码，比较分析之。

删除后代码分配220字节(0xDCh)栈空间，删除前代码分配208字节(0xD0h)

删除后代码额外维护一个初始化标志位[ebp-0D1h]

删除后代码有运行时检查(RTC\_UninitUse)，删除前代码没有

删除后代码是线性条件判断、仅处理2个case、每次判断都需重新加载比较值。删除前代码是跳转表实现，处理更多case。