C 语言的秘密

姓名： 杨尚斌

学号： 1130310128

学院： 计算机科学与技术学院

指导老师： 郑贵斌

日期： 2014-11-21

哈尔滨工业大学

一、 导读

经过大一学年度的洗礼，前后比较浅显地学习了 Python，C，Java，C++等高级语言。但由于高级语言的高封装性与库函数的丰富，极易让我们走上一个不注意细节的道路；高级 IDE 的检测，错误抛出的提示，优雅的代码风格，极易让我们不为底层涉想，不能写出最优最快的代码。

学习汇编，能让我们更加细心的进行 code，注意更多的细节。并且能够更加细心的对自己的代码进行思考，把代码降低到底层，从与“硬件的交互”中彻底的了解代码的工作原理，让我们对编程能有一个更好的认识。

二、 作业目的

1. 复习高级编程相关知识。
2. 结合反汇编生成的代码对以前学过的高级语言进行更好的认识。
3. 对 c 进行反汇编，认识 c 语言与汇编语言的对应关系。
4. 比较深层次的认识汇编，比较 c 与 asm 的相关优缺点，能对 c 写过的程序进行相关优化。

三、 C++ 与 asm 的对比

1. 数据类型

C++： 在 c++中定义了丰富的数据类型，大致上可以分为四类：

>基本类型：

>>数值类型：

>>>整型：

>>>短整型：short

>>>整型：int

>>>长整型：long

>>>浮点型：

>>>单精度型：float

>>>双精度型：double

>>字符类型：char

>构造类型

>>数组

>>结构体：struct

>>共用体：unoin

>>枚举类型

>指针类型

>空类型

注：上述数据位 c++定义的基本数据类型，当我们定义一个变量的时候，数据会自动给他分配一个对应空间大小的地址。 在一般的情况下，int 只能用来储存整数，char 只能用来储存字符，指针用来储存一个地址（操作系统不同，分配大小不同。一般默认 short 2 个字节，int 4 个字节，long 8 个字节， char 1 个字

节……具体可以在电脑上进行测试得出结果）。

C++对字符串没有定义类型，一般我们在字符串的操作中，利用字符数组来储存字符串。

C++是强类型的语言，对不同数据的结构做出了比较严格的规范，在一定的程度上限制了我们的自由，但从编码风格上说很好的帮助我们改善了自己的代码。同时也是最大限度的利用编译器在编译阶段优化自己的代码，方便内存的申请与使用，避免出现不可知的错误。

C++中的指针在无时无刻中展现着自己的魅力，由于 c++中有着指针的概念，也就让我们能够更加自由的操作程序，但在开放的同时也容易造成比较严重的错误。

汇编：汇编中的按照内存的大小可以把数据类型分为四个：BYTE(SBYTE)、WORD(SWORD)、DWORD(SDWORD)、QWORD(SQWORD)。分别代表 8 位、16 位、32 位、64 位的数据位。

汇编语言对进制的区分是通过后缀来区分的，有 B(二进制)、O(八进制)、D(十进制)、H(十进制)。在汇编语言中，默认的进制是十进制，当添加进制后缀的时候，将按照后缀进行表示。

汇编语言在它的四大数据类型中，没有对储存量的类型（整数，浮点数，字符）做出限制，可以用来储存数字，也可以用来储存字符，字符串，只要长度允许即可。在定义的时候一般会在 data 段申明。

2. 算术运算与符号表达式

C++: c++中拥有丰富的运算符，包括基本的+、-、*、/、%、++、--等。在编译器的编译阶段，将按照事先定义好的优先顺序对运算表达式进行运算。

c++同时有宏定义，可进行宏定义与宏编程。

汇编：有 add, sub, div, mul 等，值得区别的就是汇编中的 MOD，取两个数想出的余数。并且支持 AND、OR、NOT、XOR 等逻辑运算。关系运算操作符：EQ、NE、LT、GT、LE、GE.

汇编中的语句表达式求值不是在语句执行的时候就完成的，而是在对源程序进行汇编链接时完成的，换句话说，表达式中个标识符的值在汇编或者链接的时候就是确定的。

汇编中也可以进行宏编程。一旦被定义，就可以在程序中调用任意多次

3. 结构控制

1） 选择结构

C++: C++中的选择一般由 if{}else{}控制，多选在利用 else if{}的同时还可以用 switch 进行控制。

汇编：汇编中没有单独的选择结构，一般由 je、jne 等跳转语句跳转至相应的标识符控制。

2） 循环结构

C++: c++中的循环结构大致可以分为三类：

while{}、do{}while、for。在做循环的过程中可以有 continue 和 break 进行相应的辅助使其在特定时刻跳出循环。保证代码的自由性。

汇编：汇编没有独立的循环操作，一般可用 ecx 寄存器做计数器，loop 控制循环。在中途可以用跳转指令跳出循环。

4. 函数与过程

C++:c++中提供了两张函数的定义机制：一种是在 main 函数之前进行定义，在 main 函数之后对他进行实体化的定义；另一种是放在 main 函数之后进行定义，可以省略 main 函数之前的定义。

C++中函数的类型也是不同的，有在代码开始阶段 include 的库里面的库函数，也有自己在代码中定义的自定义函数。

汇编：在汇编中，我们不妨把类似 c++中函数的东西称为是“过程”。

类似于 c++，汇编的过程中都会出现一个 main 的启动过程，如下：

```
main PROC
；Statement
exit
main ENDP
```

该过程以 exit 结束，而其他过程都是以 ret 结束，如

下：

Other PROC

;Statement

Ret

Other ENDP

汇编中过程的调用是以 call 进行调用的，比较高级的还有 invoke 等指令。在调用的过程中，该指令指示处理器在新的内存地址执行相关指令，在过程中使用 ret 返回到过程被调用的地方继续执行。一般在过程的调用，涉及到了比较多的堆栈操作。

5. 局部变量与全局变量

C++: C++中全局变量需要定义在调用的最前面，局部变量单独分配空间，不影响全局变量。

6. 堆栈储存

C++: c++中的栈可以分为数组栈与指针栈，不是c++固有的一种数据结构，是认为定义，拥有“先进后出，后进先出”思想的一种人为定义的数据结构。由于库文件 stack.h 的引入也在很大意义上方便了 c++关于栈的操作。

汇编：汇编中的栈是一种必要的结构，在过程调用中起着极其重要的作用。

四、 buffer.cpp 与 buffer.asm 的相关比较

1. 定义整形变量

```
37:       int i = 0;
01235028 C7 45 E8 00 00 00 00 mov          dword ptr [i],0
```

注：定义一个 dword 的标识符

2. 调用函数

```
39:       makeNull(S);
0123502F 8D 45 CC          lea          eax,[S]
01235032 50                push         eax
01235033 E8 3D C3 FF FF    call         makeNull
(01231375h)
01235038 83 C4 04          add          esp,4
```

注：lea 操作：[s]取出 s 地址的值，lea 操作取[s]的地址，并赋值给 eax

3. makeNull 函数

```
void makeNull(Stack &S)
      13: {
01235FF0 55                push         ebp
01235FF1 8B EC             mov          ebp,esp
01235FF3 81 EC C0 00 00 00 sub          esp,0C0h
01235FF9 53                push         ebx
01235FFA 56                push         esi
01235FFB 57                push         edi
```

```
01235FFC 8D BD 40 FF FF FF      lea             edi,[ebp-0C0h]
01236002 B9 30 00 00 00         mov             ecx,30h
01236007 B8 CC CC CC CC         mov             eax,0CCCCCCCCh
0123600C F3 AB                  rep stos        dword ptr es:[edi]
      14:       S.top = -1;
0123600E 8B 45 08               mov             eax,dword ptr [S]
01236011 C7 00 FF FF FF FF      mov             dword ptr [eax],0FFFFFFFFh
      15: }
01236017 5F                     pop             edi
01236018 5E                     pop             esi
01236019 5B                     pop             ebx
```

注：push ebp; mov ebp, esp; sub esp, 0c0h,减 32 个地址，指向栈的空间，接着保护数据，push ebx esi edi，然后通过 lea edi, [ebp-0c0h]把栈的地址给 edi

mov ecx, 30h; mov eax, 0cccccccch; rep stos dword ptr es:[edi] rep 设置循环，进行初始化；stos 指令作用是将 eax 中的值拷贝到 es:edi 指向的地址；rep 可以是任何字符传指令的前缀，rep 能够引发其后的字符串指令被重复，只要 ecx 的值不为 0，重复就会继续，每一次字符串指令执行后，ecx 的值都会减小.

5. getche 实现

```
40:        buffer = _getche();
0123503B 8B F4                mov
esi,esp
0123503D FF 15 F4 11 24 01    call          dword
ptr ds:[12411F4h]
01235043 3B F4                cmp
esi,esp
01235045 E8 03 C3 FF FF       call
__RTC_CheckEsp (0123134Dh)
0123504A 88 45 F7             mov           byte
ptr [buffer],al
```

注：通过 al 把值给 buffer

6. ESC 的判断

```
41:        while (buffer != ESC)
0123504D 0F BE 45 F7          movsx         eax,byte ptr
[buffer]
01235051 83 F8 1B             cmp           eax,1Bh
01235054 0F 84 84 00 00 00    je            main+0DEh
(012350DEh)
```

注：movsx eax, byte ptr [buffer]; cmp eax,1BH; je main+0Deh(012350Deh),把 buffer 强转为 byte，然后扩展给 eax,然后让 eax 与 1Bh 进行比较，如果相等，则 je 到 main+0Deh

7. push 操作

```
45:       push(S, buffer);
01235063 0F B6 45 F7          movzx          eax,byte
ptr [buffer]
01235067 50                    push           eax
01235068 8D 4D CC              lea            ecx,[S]
0123506B 51                    push           ecx
0123506C E8 A2 C0 FF FF        call           push
(01231113h)
01235071 83 C4 08              add            esp,8
```

注：movzx eax, byte ptr [buffer]，把 buffer 强转为为 byte，然后传值给 eax,接着 push eax ，然后把 S 给 ecx,接着压栈 ecx,然后 call push 过程并且把 esp+8(buffer 和 ip 占用 8 个字节)

8. push 过程

```
18:             S.top++;
0123669E 8B 45 08              mov
eax,dword ptr [S]
012366A1 8B 08                 mov
ecx,dword ptr [eax]
012366A3 83 C1 01              add            ecx,1
012366A6 8B 55 08              mov
edx,dword ptr [S]
012366A9 89 0A                 mov            dword
ptr [edx],ecx
if (S.top >= 16)
012366AB 8B 45 08              mov
eax,dword ptr [S]
```

注：实现 S.top++的操作，把 S.top 的值给 eax

```
        19:             if (S.top >= 16)
012366AE 83 38 10                   cmp             dword
ptr [eax],10h
012366B1 7C 0D                      jl
push+40h (012366C0h)
        20:             {
        21:                 S.top -= 16;
012366B3 8B 45 08                   mov
eax,dword ptr [S]
012366B6 8B 08                      mov
ecx,dword ptr [eax]
012366B8 83 E9 10                   sub             ecx,10h
012366BB 8B 55 08                   mov
edx,dword ptr [S]
012366BE 89 0A                      mov             dword
ptr [edx],ecx
        22:             }
        23:             S.a[S.top] = c;
012366C0 8B 45 08                   mov
eax,dword ptr [S]
        22:             }
        23:             S.a[S.top] = c;
012366C3 8B 08                      mov
ecx,dword ptr [eax]
012366C5 8B 55 08                   mov
edx,dword ptr [S]
012366C8 8A 45 0C                   mov             al,byte
ptr [c]
012366CB 88 44 0A 04                mov             byte ptr
[edx+ecx+4],al
        24: }
012366CF 5F                         pop             edi
012366D0 5E                         pop             esi
012366D1 5B                         pop             ebx
012366D2 8B E5                      mov
esp,ebp
012366D4 5D                         pop             ebp
012366D5 C3                         ret
```

注：dword 型的 eax 与 10H 比较，如果大于等于 10H，执行相应的操作。eax 是 S.top，然后把 eax 的值给 ecx，后面用到[edx+ecx+4]，其中 edx 是 S，寻址，为到[edx+ecx+4]。

五、 CPP 与 asm 优缺点对比

cpp 与 asm 的优缺点，都是站在不同的角度说的，从总体上来说，很难分出谁高谁低。

CPP:属于高级语言，封装了大量的库函数，并且现有的编译器有很好的优化能力，容易在短时间被开发出大型的程序，语言更加接近自然语言，容易让人理解。但可能会产生机器编译的盲点，造成不必要的时间空间损失。

汇编：属于底层的语言，代码复杂，容易出错，基本上是建立在于硬件打交道的基础上，如果能够较好的操作，则能简化加速代码，是程序在运行上占有较大的优势；汇编程序由于和硬件底层打交道，可以编一些驱动程序，BOIS 级别的小程序。

如何更好的使用它们：如果进行大型程序的开发，高级语言的选择是必须的，现有的编译器在代码的编译上已经足够优化，完全可以满足平时的使用。但是如果由于运行环境的问题，直接用高级语言写到的程序肯能占用内存太大，这时候就需要利用汇编知识来优化，消除机器编译产生的盲点，让程序以更高的效率进行运行。

六、 学习收获

汇编语言比较注重细节，能在汇编程序的编写过程中及时的发现自己的思考漏洞，并且能从底层上解释 cpp 等高级语言中

参数的调用问题，能帮助我们更好的理解问题。汇编让原本神秘的编程不再变得那么神秘，对以后的学习有良好的启发作用。

七、 源代码

buffer.cpp 和 buffer.asm 代码

buffer.cpp(code)

```
#include <iostream>
#include <conio.h>
#define MaxSize 16
#define ESC 27
#define print 45
using namespace std;
struct Stack
{
   int top;
   char a[MaxSize];
};
void makeNull(Stack &S)
{
   S.top = -1;
}
void push(Stack &S, char c)
```

```
{
      S.top++;
      if (S.top >= 16)
      {
         S.top -= 16;
      }
      S.a[S.top] = c;
}
char pop(Stack &S)
{
      S.top--;
      if (S.top < -1)
      {
         S.top += 16;
      }
      return S.a[S.top+1];
}
int main()
{
   char buffer;
   int i = 0;
   Stack S;
```

```
    makeNull(S);
    buffer = _getche();
    while (buffer != ESC)
    {
       if (buffer != print)
       {
          push(S, buffer);
          buffer = _getche();
       }
       while (buffer == print)
       {
          cout << pop(S) << " ";
          i++;
          buffer = _getche();
       }

    }
    cout << "Thank you!" << endl;
    return 0;
}
```

buffer.asm(code)——反汇编代码（部分代码）

```
    34: int main()
    35: {
01235004 EC                in         al,dx
01235005 F8                clc
    34: int main()
    35: {
01235006 00 00              add        byte ptr
[eax],al
01235008 00 53 56           add        byte ptr
[ebx+56h],dl
0123500B 57                push       edi
0123500C 8D BD 08 FF FF FF    lea
edi,[ebp-0F8h]
01235012 B9 3E 00 00 00      mov        ecx,3Eh
01235017 B8 CC CC CC CC      mov
eax,0CCCCCCCCh
0123501C F3 AB              rep stos    dword
ptr es:[edi]
0123501E A1 00 00 24 01      mov
eax,dword ptr ds:[01240000h]
01235023 33 C5              xor        eax,ebp
01235025 89 45 FC           mov        dword
```

```
ptr [ebp-4],eax
    36:    char buffer;
    37:    int i = 0;
01235028 C7 45 E8 00 00 00 00 mov        dword
ptr [i],0
    38:    Stack S;
    39:    makeNull(S);
0123502F 8D 45 CC           lea        eax,[S]
01235032 50                push       eax
01235033 E8 3D C3 FF FF      call
makeNull (01231375h)
01235038 83 C4 04           add        esp,4
    40:    buffer = _getche();
0123503B 8B F4              mov        esi,esp
0123503D FF 15 F4 11 24 01    call       dword
ptr ds:[12411F4h]
01235043 3B F4              cmp        esi,esp
01235045 E8 03 C3 FF FF      call
__RTC_CheckEsp (0123134Dh)
0123504A 88 45 F7           mov        byte ptr
[buffer],al
    41:    while (buffer != ESC)
```

```
0123504D 0F BE 45 F7        movsx
eax,byte ptr [buffer]
01235051 83 F8 1B           cmp        eax,1Bh
01235054 0F 84 84 00 00 00    je
main+0DEh (012350DEh)
   42:   {
   43:       if (buffer != print)
0123505A 0F BE 45 F7        movsx
eax,byte ptr [buffer]
0123505E 83 F8 2D           cmp        eax,2Dh
01235061 74 23              je         main+86h
(01235086h)
   44:       {
   45:           push(S, buffer);
01235063 0F B6 45 F7        movzx
eax,byte ptr [buffer]
01235067 50                 push       eax
01235068 8D 4D CC           lea        ecx,[S]
0123506B 51                 push       ecx
0123506C E8 A2 C0 FF FF      call       push
(01231113h)
01235071 83 C4 08           add        esp,8
```

```
   44:      {
   45:          push(S, buffer);
0123506C E8 A2 C0 FF FF      call       push
(01231113h)
01235071 83 C4 08           add        esp,8
   46:          buffer = _getche();
01235074 8B F4              mov        esi,esp
01235076 FF 15 F4 11 24 01    call       dword
ptr ds:[12411F4h]
0123507C 3B F4              cmp        esi,esp
0123507E E8 CA C2 FF FF      call
__RTC_CheckEsp (0123134Dh)
01235083 88 45 F7           mov        byte ptr
[buffer],al
   47:      }
   48:      while (buffer == print)
01235086 0F BE 45 F7         movsx
eax,byte ptr [buffer]
   47:      }
   48:      while (buffer == print)
0123508A 83 F8 2D           cmp        eax,2Dh
0123508D 75 4A              jne
```

```
main+0D9h (012350D9h)
   49:      {
   50:          cout << pop(S) << " ";
0123508F 68 70 DC 23 01      push
123DC70h
01235094 8D 45 CC           lea        eax,[S]
01235097 50                 push       eax
01235098 E8 23 C3 FF FF      call       pop
(012313C0h)
0123509D 83 C4 04           add        esp,4
012350A0 0F B6 C8           movzx      ecx,al
012350A3 51                 push       ecx
012350A4 8B 15 A8 10 24 01    mov
edx,dword ptr ds:[12410A8h]
012350AA 52                 push       edx
012350AB E8 28 C4 FF FF      call
std::operator<<<std::char_traits<char> >
(012314D8h)
012350B0 83 C4 08           add        esp,8
012350B3 50                 push       eax
012350B4 E8 08 C2 FF FF      call
std::operator<<<std::char_traits<char> >
```

```
(012312C1h)
012350B9 83 C4 08           add        esp,8
   51:         i++;
012350BC 8B 45 E8           mov
eax,dword ptr [i]
012350BF 83 C0 01           add        eax,1
012350C2 89 45 E8           mov        dword
ptr [i],eax
   52:         buffer = _getche();
012350C5 8B F4              mov        esi,esp
012350C7 FF 15 F4 11 24 01    call       dword
ptr ds:[12411F4h]
012350CD 3B F4              cmp        esi,esp
012350CF E8 79 C2 FF FF      call
__RTC_CheckEsp (0123134Dh)
012350D4 88 45 F7           mov        byte ptr
[buffer],al
   53:      }
012350D7 EB AD              jmp        main+86h
(01235086h)
   54:
   55:   }
```

```
012350D9 E9 6F FF FF FF      jmp
main+4Dh (0123504Dh)
   56:   cout << "Thank you!" << endl;
012350DE 8B F4              mov        esi,esp
012350E0 68 0B 14 23 01      push
123140Bh
012350E5 68 74 DC 23 01      push
123DC74h
012350EA A1 A8 10 24 01      mov
eax,dword ptr ds:[012410A8h]
012350EF 50                 push       eax
012350F0 E8 CC C1 FF FF      call
std::operator<<<std::char_traits<char> >
(012312C1h)
012350F5 83 C4 08            add        esp,8
012350F8 8B C8              mov        ecx,eax
012350FA FF 15 94 10 24 01    call       dword
ptr ds:[1241094h]
01235100 3B F4              cmp        esi,esp
01235102 E8 46 C2 FF FF      call
__RTC_CheckEsp (0123134Dh)
   57:   return 0;
```

```
01235107 33 C0              xor         eax,eax
   58: }
01235109 52                 push        edx
0123510A 8B CD              mov         ecx,ebp
0123510C 50                 push        eax
0123510D 8D 15 38 51 23 01    lea
edx,ds:[1235138h]
01235113 E8 23 C0 FF FF       call
@_RTC_CheckStackVars@8 (0123113Bh)
01235118 58                 pop         eax
01235119 5A                 pop         edx
0123511A 5F                 pop         edi
0123511B 5E                 pop         esi
0123511C 5B                 pop         ebx
0123511D 8B 4D FC            mov
ecx,dword ptr [ebp-4]
01235120 33 CD              xor         ecx,ebp
01235122 E8 1A BF FF FF       call
@__security_check_cookie@4 (01231041h)
01235127 81 C4 F8 00 00 00    add
esp,0F8h
0123512D 3B EC              cmp         ebp,esp
```

```
0123512F E8 19 C2 FF FF      call
__RTC_CheckEsp (0123134Dh)
01235134 8B E5              mov         esp,ebp
01235136 5D                 pop         ebp
01235137 C3                 ret
01235138 01 00              add         dword
ptr [eax],eax
0123513A 00 00              add         byte ptr
[eax],al
0123513C 40                 inc         eax
0123513D 51                 push        ecx
0123513E 23 01              and
eax,dword ptr [ecx]
01235140 CC                 int         3
01235141 ??                 ?? ??
01235142 ??                 ?? ??
01235143 FF 14 00           call        dword
ptr [eax+eax]
01235146 00 00              add         byte ptr
[eax],al
01235148 4C                 dec         esp
01235149 51                 push        ecx
```

```
0123514A 23 01              and
eax,dword ptr [ecx]
0123514C 53                 push        ebx
0123514D 00 CC              add         ah,cl
```

```
**************
void makeNull(Stack &S)
13: {
01235FF0 55                 push        ebp
01235FF1 8B EC              mov         ebp,esp
01235FF3 81 EC C0 00 00 00    sub         esp,0C0h
01235FF9 53                 push        ebx
01235FFA 56                 push        esi
01235FFB 57                 push        edi
01235FFC 8D BD 40 FF FF FF    lea         edi,[ebp-
0C0h]
01236002 B9 30 00 00 00       mov         ecx,30h
01236007 B8 CC CC CC CC       mov
```

```
eax,0CCCCCCCCh
0123600C F3 AB              rep stos    dword ptr
es:[edi]
   14: S.top = -1;
0123600E 8B 45 08           mov         eax,dword
ptr [S]
01236011 C7 00 FF FF FF FF    mov         dword ptr
[eax],0FFFFFFFFh
   15: }
01236017 5F                 pop         edi
01236018 5E                 pop         esi
01236019 5B                 pop         ebx
0123601A 8B E5              mov         esp,ebp
0123601C 5D                 pop         ebp
0123601D C3                 ret



void push(Stack &S, char c)
   17: {
01236680 55                 push        ebp
```

```
01236681 8B EC              mov        ebp,esp
01236683 81 EC C0 00 00 00    sub        esp,0C0h
01236689 53                push       ebx
0123668A 56                push       esi
0123668B 57                push       edi
0123668C 8D BD 40 FF FF FF    lea        edi,[ebp-
0C0h]
01236692 B9 30 00 00 00       mov        ecx,30h
01236697 B8 CC CC CC CC       mov
eax,0CCCCCCCCh
0123669C F3 AB              rep stos    dword ptr
es:[edi]
   18:    S.top++;
0123669E 8B 45 08            mov        eax,dword
ptr [S]
012366A1 8B 08              mov        ecx,dword
ptr [eax]
012366A3 83 C1 01            add        ecx,1
012366A6 8B 55 08            mov        edx,dword
ptr [S]
012366A9 89 0A              mov        dword ptr
[edx],ecx
```

```
   19:    if (S.top >= 16)
012366AB 8B 45 08           mov         eax,dword
ptr [S]
   19:    if (S.top >= 16)
012366AE 83 38 10           cmp         dword ptr
[eax],10h
012366B1 7C 0D              jl          push+40h
(012366C0h)
   20:    {
   21:        S.top -= 16;
012366B3 8B 45 08           mov         eax,dword
ptr [S]
012366B6 8B 08              mov         ecx,dword
ptr [eax]
012366B8 83 E9 10           sub         ecx,10h
012366BB 8B 55 08           mov         edx,dword
ptr [S]
012366BE 89 0A              mov         dword ptr
[edx],ecx
   22:    }
   23:    S.a[S.top] = c;
012366C0 8B 45 08           mov         eax,dword
```

```
ptr [S]
   22:    }
   23:    S.a[S.top] = c;
012366C3 8B 08             mov        ecx,dword
ptr [eax]
012366C5 8B 55 08           mov        edx,dword
ptr [S]
012366C8 8A 45 0C           mov        al,byte ptr
[c]
012366CB 88 44 0A 04         mov        byte ptr
[edx+ecx+4],al
   24: }
012366CF 5F               pop        edi
012366D0 5E               pop        esi
012366D1 5B               pop        ebx
012366D2 8B E5             mov        esp,ebp
012366D4 5D               pop        ebp
012366D5 C3               ret
```