2151974 赵明泽

第五次作业

目录

第五次作业
目录
第一题：C语言嵌入汇编
示例1：两数相加
示例2：一个数乘5
第二题：多模块链接
更新：分别是多个文件，分别编译和汇编，然后生成不同的目标文件，再链接
感想

第一题：C语言嵌入汇编

要求：c语言和汇编的混合编程，实现嵌入式

思路：首先，查找资料，参考以下文章：https://blog.csdn.net/u014555106/article/details/124577187

熟悉一下语法，要注意语法的一些规则：

1. 源-目标：在 Intel 语法中，第一个操作数是目标，第二个操作数是源
2. 寄存器命名：在寄存器名字前加e，例如`ax`写作`eax`
3. 在 Intel 语法中，基址寄存器包含在 '[' 和 ']' 中，间接内存引用类似于`section:[base + index*scale + disp]`

以下是一些代码示例：

```
mov     eax,1
mov     ebx,0ffh
int     80h
mov     ebx, eax
mov     eax,[ecx]
mov     eax,[ebx+3]
mov     eax,[ebx+20h]
add     eax,[ebx+ecx*2h]
lea     eax,[ebx+ecx]
sub     eax,[ebx+ecx*4h-20h]
```

下面是C语言基本内联汇编的格式：

```
//单行
asm("mov eax, 1");
__asm__("mov ebx, 0ffh");
//多行
__asm__ ("mov eax, 1\n\t"
         "mov ebx, 0ffh\n\t");
```

上面用到`asm`和`__asm__`，两个都是有效的。当关键字`asm`在程序中有冲突的时候，可以使用`__asm__`。如果有多个指令，每行写一个双引号，并在指令后添加`\n\t`。这是因为gcc将每条指令作为字符串发送给as(GAS)，通过换行符/制表符区分来发送正确格式化字符给汇编器。

如果不希望自己编写的汇编代码被gcc优化或移动，需要使用到`volatile`这个keyword，将其放在`asm`和`()`之间即可。

接下来，又去了解了"扩展汇编"有关知识：

在基本嵌入汇编格式中，只使用了指令。在扩展汇编中，还可以指定更多操作，比如：指定输入寄存器，输出寄存器和变化表（clobber list）。并不一定要指定使用哪些寄存器，可以把这件头痛的事情交给GCC去做。扩展汇编的格式如下：

```
asm ( assembler template
      : output operands                  /* optional */
      : input operands                   /* optional */
      : list of clobbered registers      /* optional */
    );
```

这个模板由若干条汇编指令组成，每个操作数（括号里C语言的变量）都有一个限制符（""中的内容）加以描述。冒号用来分割输入的操作和输出的操作，如果每组内有多个操作数，用逗号分割它们。操作数最多为10个，或者依照具体机器而异。如果没有输出操作，但是又有输入，必须使用连续两个冒号，两个连续冒号中无内容，表示没有输出结果的数据操作。

以下是一些示例：

示例1：两数相加

```
#include <stdio.h>
int main()
{
  int num1 = 10, num2 = 15;
  __asm__ __volatile__("add %%ebx, %%eax"
                       :"=a"(num1)              // output
                       :"a"(num1), "b"(num2)    // input
                       );
  printf("num1 + num2 = %d\n",num1);
  return 0;
}
```

这个程序演示了两个数相加，首先，定义了两个数`num1`和`num2`，两个数作为输入，加和后，`ax`寄存器值作为输出，结果为：

```
num1 + num2 = 25
```

示例2：一个数乘5

```
#include <stdio.h>
int main()
{
  int x = 5, five_times_x = 0;
  asm ("leal (%1,%1,4), %0"
       : "=r" (five_times_x)
       : "r" (x)
      );
  printf("five_times_x = %d\n",five_times_x);
  return 0;
}
```

这个程序演示了一个输入的数乘5，定义的数`x`是要乘5的数，`five_times_x`是乘完的结果：

```
five_times_x = 25
```

由于在很多情况下，C语言无法完全代替汇编语言，比如：操作某些特殊的CPU寄存器，操作主板上的某些IO端口或者性能达不到要求等情况下，我们必须在C语言里面嵌入汇编语言，以达到我们的需求。

第二题：多模块链接

使用一款好用的在线编译器查看上一题示例1的代码，关键两行代码对比如下：

C语言：

```
int num1 = 10, num2 = 15;
__asm__ __volatile__("add     %%ebx, %%eax"
                     :"=a"(num1)              // output
                     :"a"(num1), "b"(num2)    // input
                     );
```

汇编语言：

```
mov     DWORD PTR [rbp-20], 10
mov     DWORD PTR [rbp-24], 15
mov     eax, DWORD PTR [rbp-20]
mov     edx, DWORD PTR [rbp-24]
mov     ebx, edx
add     %ebx, %eax
mov     DWORD PTR [rbp-20], eax
```

发现定义变量是两个双字指针，然后把这两个指针指向的内容，赋值给对应寄存器，然后把嵌入的汇编代码直接复制粘贴过来，结果放回一个双字指针，也就是C语言定义的变量中。

模仿这种样式，参考网上资料，现建立两个文件：`add.h`和`add.c`，在`add.h`中写入如下代码：

```
#ifndef ADD_H
#define ADD_H

int sum(int a, int b);

#endif
```

在`add.c`中写入如下代码：

```
#include <stdio.h>
#include "add.h"
int main()
{
    int num1 = 10, num2 = 15;
    printf("num1 + num2 = %d\n", sum(num1, num2));
    return 0;
}
```

发现头文件提前声明了加和函数`sum()`，但是并未实现。在路径下打开`cmd`，输入`gcc`指令如下：

对C文件进行预处理：

```
gcc -E add.c -o add.i
```

编译：

```
gcc -S add.i -o add.s
```

打开由这一步生成的汇编代码，在其中寻找部分关键代码如下：

```
call    __main
movl    $10, -4(%rbp)
movl    $15, -8(%rbp)
movl    -8(%rbp), %edx
movl    -4(%rbp), %eax
movl    %eax, %ecx
call    sum
movl    %eax, %edx
```

这一段的意思是`main`函数中定义两个变量，两个变量被存入寄存器`DX`和`AX`，随后调用`sum`过程，计算结果存入寄存器`DX`。此时，修改`.s`文件，在后面加入`sum`的实现，如下：

```
.global     sum
sum:
    add     %edx, %eax
    retq
```

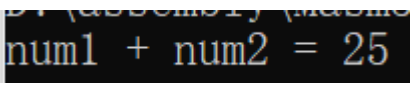
汇编：

```
gcc -c add.s -o add.o
```

生成可执行文件：

```
gcc add.o -o add
```

运行结果如下：



更新：分别是多个文件，分别编译和汇编，然后生成不同的目标文件，再链接

拆分文件，分别放主函数`main`和求和函数`sum`，如下：

在`main.c`中：

```
#include <stdio.h>
extern int sum(int a, int b);
int main()
{
    int c = sum(1, 2);
    printf("1 + 2 = %d", c);
    return 0;
}
```

在`sum.c`中：

```
int sum(int a, int b)
{
    return a + b;
}
```

使用在线编译器查看`sum.c`汇编代码实现，写在新的文件`sum.s`中，如下：

```
    .file   "wlc1.c"
    .text
    .globl  sum
    .def    sum;    .scl    2;  .type   32; .endef
    .seh_proc   sum
sum:
    pushq   %rbp
    .seh_pushreg    %rbp
    movq    %rsp, %rbp
    .seh_setframe   %rbp, 0
    .seh_endprologue
    movl    %ecx, 16(%rbp)
    movl    %edx, 24(%rbp)
    movl    16(%rbp), %edx
    movl    24(%rbp), %eax
    addl    %edx, %eax
    popq    %rbp
    ret
    .seh_endproc
    .ident  "GCC: (x86_64-posix-sjlj-rev0, Built by MinGW-W64 project) 8.1.0"
```

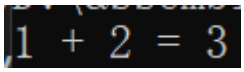
汇编成`.o`文件。接下来，使用`gcc`编译`main.c`，使用如下指令：

```
gcc -E main.c -o main.i
gcc -S main.i -o main.s
gcc -c main.s -o main.o
```

继续使用`gcc`把两个`.o`文件链接到一起，如下：

```
gcc sum.o main.o -o main
```

这是一个汇编文件与一个C语言文件，使用外联汇编形式链接在一起的示例，结果：



感想

学习了C语言内联和外联汇编，感觉到可以利用这个途径修改CPU中寄存器值，进一步提高程序运行速度。不过要小心，随意修改寄存器可能导致产生意想不到的后果。

进一步深入体会了`gcc`的操作，对C语言程序某些情况下报错与否有了新的理解，比如，第一步预处理时，就不在乎函数实现是否缺失，只对文本进行处理。