Lab2 浮点数相关

Lab2目标：熟悉浮点数的表示和运算，并会用gcc生成指定平台的汇编代码

一、工具的使用

1.用gcc编译c代码，生成汇编代码、目标代码和可执行文件。

我们之前接触过：

`gcc example.c -o example`可以直接生成一个c代码的可执行文件。实际上，它内部包含了产生汇编代码和链接的过程。

- 默认情况下，

  ```
  gcc example.c -o example
  ```

  自动完成了所有步骤：

  1. **预处理 (Preprocessing)**（展开宏、头文件）
  2. **编译 (Compilation)**（从C语言直接编译到汇编代码.s）
  3. **汇编 (Assembly)**（汇编为机器指令.o）
  4. **链接 (Linking)**（将生成的机器指令与库文件链接为可执行文件，生成 `example`）

此命令不会在磁盘上产生`.o`或`.s`文件（在内存中就完成了这些步骤），而是**直接生成可执行文件。**

如果我们想在磁盘中看到这些中间的文件，只需要调用gcc的每一步的指令即可。

以下是一个**从 C 源代码到最终可执行文件**的完整过程——

假设你的源文件名为 `example.c`：

```c
// example.c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

🚩 第一步：C代码 → 汇编代码（.s）

命令：

```
gcc -S example.c -o example.s
```

- 参数含义：
  - `-S`：只进行编译（C代码转为汇编代码），**不进行汇编和链接。**
  - `-o example.s`：生成的汇编代码文件名。

输出文件：

- `example.s`（**汇编代码**源文件，可用文本编辑器或vim查看）

🚩 第二步：汇编代码（.s）→ 目标文件（.o）

命令：

```
gcc -c example.s -o example.o
```

- 参数含义：
  - `-c`：将汇编代码进行汇编，生成**目标文件**（二进制，但未链接）。
  - `-o example.o`：目标文件名。

输出文件：

- `example.o`（目标文件，不可直接运行）

🚩 第三步：目标文件（.o）→ 可执行文件

命令：

```
gcc example.o -o example
```

- 参数含义：
  - 此步骤默认进行链接。
  - `-o example`：生成的最终可执行文件名。

输出文件：

- `example`（可执行文件，可以直接运行）

🚩 第四步：运行可执行文件

在终端中（注意目录是否正确）输入：

```
./example
```

- 输出结果为：

```
Hello, World!
```

📌 完整过程总结

```
example.c (C源代码)
    |
    | gcc -S example.c -o example.s
    ↓
example.s (汇编代码)
```

```
    |
    | gcc -c example.s -o example.o
    ↓
example.o (目标文件)
    |
    | gcc example.o -o example
    ↓
example (链接后的可执行文件)
    |
    | ./example
    ↓
程序执行输出: Hello, World!
```

以上即为从C源代码到最终执行文件的**完整编译链接过程。**

2.用objdump来反汇编前面生成的目标文件.o

这一步可以让我们使用目标文件.o来得到它对应的汇编代码.s。使用的工具名为objdump。

下面是在 **Linux系统** 下，从安装到使用 `objdump` 反汇编目标文件（`.o`）的完整步骤说明：

✅ 第一步：安装 objdump 工具

在大多数 Linux 系统上，`objdump` 包含在名为 **binutils** 的软件包中，通常默认安装，大概率不需要额外的安装步骤。

1. 检查是否已安装：

打开终端，输入：

```
objdump --version
```

- 如果显示类似以下版本信息，则表示已安装：

```
GNU objdump (GNU Binutils) 2.34
...
```

- 如果提示找不到命令：

```
objdump: command not found
```

则表示未安装，需要安装它。

---

2. 安装 `objdump` (binutils)，大概率不需要：

- 在 Ubuntu/Debian 系统：

```
sudo apt update
sudo apt install binutils
```

安装完成后再执行确认安装成功。：

```
objdump --version
```

✅ 第二步：生成目标文件（`.o`），这是先前已经完成的步骤

源代码为 `example.c`：

```
#include <stdio.h>

int main() {
    printf("Hello, objdump!\n");
    return 0;
}
```

编译生成目标文件：

```
gcc -c example.c -o example.o
```

- 此命令**只编译不链接**，生成目标文件 `example.o`。

✅ 第三步：使用 `objdump` 反汇编目标文件

- 基础反汇编命令：

```
objdump -d example.o
```

参数说明：

参数	说明
`-d`	反汇编(disassemble)

✅ 第四步：查看 `objdump` 输出结果

执行上述命令后，将显示类似如下的汇编代码片段：

```
example.o:     file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
   0:   55                      push   rbp
   1:   48 89 e5                mov    rbp,rsp
   4:   48 83 ec 10             sub    rsp,0x10
   8:   48 8d 3d 00 00 00 00    lea    rdi,[rip+0x0]
   f:   e8 00 00 00 00          call   14 <main+0x14>
  14:   b8 00 00 00 00          mov    eax,0x0
  19:   c9                      leave
  1a:   c3                      ret
```

> 上面的输出表示：
>
> - 左侧是机器指令在内存中的偏移地址；
> - 中间是二进制的机器指令；
> - 右侧是对应的汇编指令。

✅ 补充技巧（可选）：

- 若想保存输出结果以便后续查看，执行：

```
objdump -d example.o > example_disassembly.txt
```

这个命令会将反汇编的结果输出重定向到txt文件中，便于接下来可用任何文本编辑器查看反汇编生成的汇编代码，可以与先前生成的.s文件进行对比。

📌 总结整个过程：

完整流程：

```
# 检查 objdump 是否安装
objdump --version

# 编译生成目标文件
gcc -c example.c -o example.o

# 反汇编目标文件
objdump -d -M intel example.o
```

二、课堂练习

需要提交代码和文档，文档内容包括原理和数据分析。

1.浮点数的表示

根据浮点数的表示，S.Exp.Frac的表示，用位运算来构造指定的浮点数，可以指定float或double。例如：

a) 20250309和-20250309
b) -0.0625
c) 正无穷大
d) NaN

最后打印数值和其二进制表示。

2.浮点数和整数的转换

给定int/long类型整数x，转换为float/double后，输出x,-x,(float)x,(double)x的二进制表示，并计算|x|和（double/float）x的最大公共子串。

3.gcc编译和objdump反汇编的应用

编写简单的函数， 包含不同类型数值的四则运算，并且涉及到类型转换。用gcc编译生成汇编代码。

- 用gcc编译生成目标文件，用objdump反汇编。
- 观察两种不同途径生成的汇编代码是否有区别?
- 不同数据类型对应的运算和数据类型转换对应的指令有哪些?