PA2-1 实验报告

刘时宜 201180078

2022 年 4 月 10 日

目录

1 实现过程

按照指令说的，不断重复跑测试样例、查看无效指令、对应查找反汇编结果确定指令类型、参考i386手册实现函数、填入opcode.c、跑测试样例的过程，直到完全跑完测试样例。

由于指令太多，具体实现过程就不在此报告中一一列举了。可以到以下两个连接中查看。

Instruction-Implementations-1

Instruction-Implementations-2

2 对于一些指令实现讨论

2.1 LEA的实现

这条指令的功能是将计算好的内存地址写到寄存器中。

这条指令在手册中被放在不能使用框架代码的指令列表中，但是若将源操作数作为一内存类型操作数，将目的操作数作为一寄存器类型操作数，在赋值是将源操作数的地址写入目的操作数的值，应当可以使用框架代码实现这个函数？

在实验中，按照上述思路实现代码，并未报错，代码如下：

```
// @ pa_nju/nemu/src/cpu/instr/lea.c
#include "cpu/instr.h"
/*
Put the implementations of 'lea' instructions here.
*/

static void instr_execute_2op()
{
    opr_dest.val = opr_src.addr;

    operand_write(&opr_dest);
}

make_instr_impl_2op(lea, rm, r, v)
```

3 i386手册勘误

3.1 eip 在指令描述中的取值

在指令的伪代码描述中（如下图），eip的取值均为加上当前指令长度之后的取值，而非代码实现当中进入当前指令时的eip的值。这点可能和芯片的电路实现有关，这样描述更加自然，但是在NEMU的代码中会造成一定困扰，或许可以在guide中做一点提醒？

```
ELSE (* OperandSize = 32 *)
   Push(EIP);
   EIP ← EIP + rel32;
```

图 1: call 指令中的伪代码

3.2 Jcc 指令细节描述

JNA指令描述中，CF = 1和ZF = 1之间的逻辑关系（or）没有表达清楚，如下图：

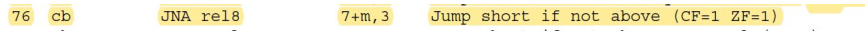

图 2: Jcc (JNA) 指令描述

正确表达式应当如JBE指令的描述。

图 3: Jcc (JBE) 指令描述

3.3 MODr/m 位的解读

MODr/m = 00 100是手册上的描述里没有偏移量，如下图所示：

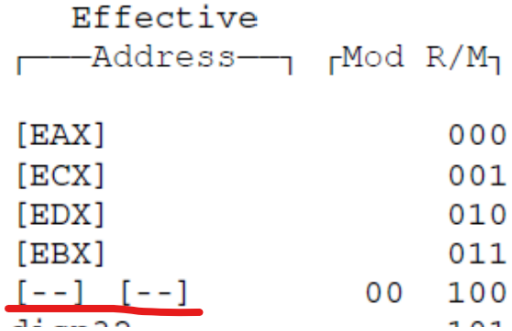

图 4: MODr/m 描述

但是框架代码和反汇编结果均有32位偏移量：

```
9  int case_mod_00(uint32_t eip, MODRM modrm, OPERAND *opr)
10 {
11     int len = 0;
12     switch (modrm.rm)
13     {
14     case 4: // SIB + disp32
```

图 5: 框架代码实现

4 思考题

1. 左侧为反汇编结果，右侧为.img文件，对比可以看到，.img文件的内容即为内存地址30000开始以后的内容。

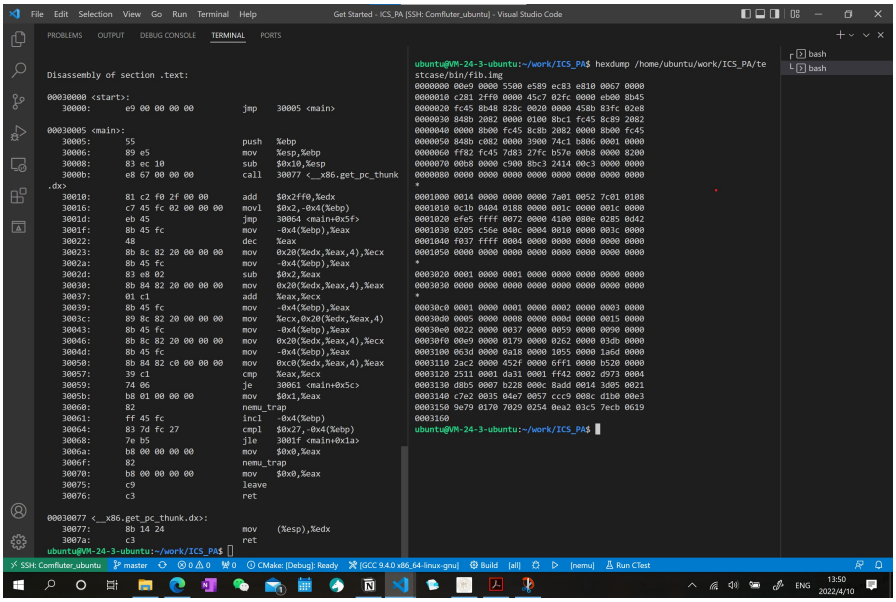

图 6: 框架代码实现

指令在机器中的表示为一串位串。

2. 会使得`instr_execute_2op()`函数全局可见，导致重名或者函数引用关系混乱。

3. 由于这个测试样例中有大量浮点数运算，猜测为浮点数运算导致的精度损失造成的不能通过测试样例。

5 心得体会

1. PA教程设计得很好，让我对于解读机器码的过程以及指令的执行过程有了很切身的体会。

2. 让我学习到构建大型应用程序的时候应当遵循模块化的代码架构设计，能够极大地减少混乱的可能性。

3. 让我认识到代码可复用性的重要性。通过宏定义实现代码极大地减轻了工作量。