

南京航空航天大学《计算机组成原理II课程设计》报告

- 姓名：邵震哲
- 班级：1620204
- 学号：162020130
- 报告阶段：PA2.1
- 完成日期：2022.4.24
- 本次实验，我完成了所有内容。

目录

南京航空航天大学《计算机组成原理II课程设计》报告

目录

思考题

实验内容

- 1.实现标志寄存器（10分）
- 2.实现所有 RTL 指令（30分）
- 3.实现 6 条 x86 指令（30分）
- 4.成功运行 dummy（10分）
- 5.实现 Diff-test（20分）

遇到的问题及解决办法

实验心得

其他备注

思考题

1. 增加了多少（10分）

包括了操作码

源操作数（立即数、寄存器编号、存储地址）

目的操作数地址（寄存器编号、存储地址）

2. 是什么类型（10分）

表中每个表项是 `opcode_entry` 类型

`opcode_entry.width` 记录了操作数长度信息

`opcode_entry.decode` 是一个函数指针，指向译码函数 `make_DHelper`

`opcode_entry.execute` 是一个函数指针，指向执行函数 `make_EHelper`

3. 操作数结构体的实现（10分）

```
typedef struct {
    uint32_t type;    //类型
    int width;        //宽度
    union {
        uint32_t reg;    //寄存器
        rtlreg_t addr;    //操作数的地址
        uint32_t imm;    //立即数
        int32_t simm;    //带符号立即数
    };
    rtlreg_t val;    //操作数的值
    char str[OP_STR_SIZE]; //指令的解释
} Operand;
```

先根据操作码的类型标记操作数的类型type，再根据操作数类型对共同体内的对应变量赋值。val保存该操作数的值，str保存了要输出的指令信息

4. 复现宏定义 (30分)

- make_EHelper(mov) //mov 指令的执行函数

根据宏定义

```
#define make_EHelper(name) void concat(exec_, name) (vaddr_t *eip)
```

得到 void exec_mov (vaddr_t *eip)

- make_EHelper(push) //push 指令的执行函数

根据宏定义

```
#define make_EHelper(name) void concat(exec_, name) (vaddr_t *eip)
```

得到 void exec_push (vaddr_t *eip)

- make_DHelper(I2r) //I2r 类型操作数的译码函数

根据宏定义

```
#define make_DHelper(name) void concat(decode_, name) (vaddr_t *eip)
```

得到 void decode_I2r (vaddr_t *eip)

- IDEX(I2a, cmp) //cmp 指令的 opcode_table 表项

首先根据

```
#define IDEX(id, ex)      IDEXW(id, ex, 0)
```

得到 IDEXW(I2a, cmp, 0)，再根据

```
#define IDEXW(id, ex, w)  {concat(decode_, id), concat(exec_, ex), w}
```

得到 {decode_I2a, exec_cmp, 0}

- EX(nop) //nop 指令的 opcode_table 表项

```
#define EX(ex) EXW(ex, 0)
```

得到 `EXW(nop,0)`，再根据

```
#define EXW(ex, w) {NULL, concat(exec_, ex), w}
```

得到 `{NULL, exec_nop, 0}`

- `make_rtl_arith_logic(and)` //and 运算的 RTL 指令

根据宏定义

```
#define make_rtl_arith_logic(name) \
    static inline void concat(rtl_, name) (rtlreg_t* dest, const rtlreg_t* \
    src1, const rtlreg_t* src2) { \
        *dest = concat(c_, name) (*src1, *src2); \
    } \
    static inline void concat3(rtl_, name, i) (rtlreg_t* dest, const \
    rtlreg_t* src1, int imm) { \
        *dest = concat(c_, name) (*src1, imm); \
    }
```

得到

```
static inline void rtl_and (rtlreg_t* dest, const rtlreg_t* src1, const \
rtlreg_t* src2) {      *dest = ((*src1) & (*src2)); \
} \
static inline void rtl_andi (rtlreg_t* dest, const rtlreg_t* src1, int \
imm) { \
    *dest = ((*src1) & (imm)); \
}
```

5. 立即数背后的故事 (10分)

需要注意机器是大端架构还是小端架构，

因为大小端架构字节存储和访问的顺序不同。

解决的方法是在存储和访问之前先判断大小端架构，根据类型使用不同的字节读取方式。

6. 神奇的 eflags (20分)

当超出表示范围时产生溢出。

不能替换，进位不一定有溢出。

$OF = C_n \oplus C_{n-1}$

7. git branch 和 git log 截图 (最新的, 一张即可) (10分)

git branch

```
shaozhenzhe@Debian:~/ics2022/nexus-am/tests/cputest$ git branch
master
pa0
pa1
* pa2
shaozhenzhe@Debian:~/ics2022/nexus-am/tests/cputest$
```

git log

```
shaozhenzhe@Debian: ~/ics2022/nexus-am/tests/cputest
3488bb9 (HEAD -> pa2) finish pa2.1
0fcf55e > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 17
:01:41 up 2:45, 2 users, load average: 0.00, 0.00, 0.00 d35fc6fb7029e16b8d0e8becb7981e5e829530c4
f985dd2 > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux
17:01:41 up 2:45, 2 users, load average: 0.00, 0.00, 0.00 dfea34bd26a74dc410c3b6739dff4f8222a7e1d
baa2f26 finished all the tasks and run correctly
2d51f0a > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 16
:54:36 up 2:38, 2 users, load average: 0.08, 0.02, 0.01 ba3bd5d2393894eca06bfb820c99e3a8dc0639
2ed2074 > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux
16:54:36 up 2:38, 2 users, load average: 0.08, 0.02, 0.01 598a3b1dc46a9589625df809a99380f8cdce7ff
812b517 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 16
:53:39 up 2:37, 2 users, load average: 0.00, 0.00, 0.00 6d536a69781fe67a229723405b42f9136fba83bb
489110f > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux
16:53:39 up 2:37, 2 users, load average: 0.00, 0.00, 0.00 4521f815796c02bda25a351c6c6f69acd8ec86d6
607d939 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 16
:51:51 up 2:35, 2 users, load average: 0.00, 0.00, 0.00 792d60379423d869c027907471e0dd485ca3ea57
99f308f > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux
16:51:51 up 2:35, 2 users, load average: 0.00, 0.00, 0.00 e940017649a13bfb40ff1cd1f4e746bd48f96d22
adb8db7 fix an error
b5584d6 try all the code
5b3a8e0 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 16
:44:02 up 2:27, 2 users, load average: 0.00, 0.00, 0.00 979c3aa81b5fad7baa8664ed6faf7048f2ea17a0
fe0728e > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux
16:44:02 up 2:27, 2 users, load average: 0.00, 0.00, 0.00 c9c3ebcfff7bf894c22f1f8calc23709d0547a7d
cd48512 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 16
:33:50 up 2:17, 2 users, load average: 0.00, 0.00, 0.00 69ea2382a067dd5f62d9873cc0b8ae1740a4cf84
1fa6ab5 > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux
16:33:50 up 2:17, 2 users, load average: 0.00, 0.00, 0.00 26112f51ef9fe9dc8e427336f5363a30da80c96a
1d42df7 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 16
:33:27 up 2:17, 2 users, load average: 0.00, 0.00, 0.00 4d1ac1c2d0f089b8e4d4e784f6c3ae39c83a8dd
3657550 > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux
16:33:27 up 2:16, 2 users, load average: 0.00, 0.00, 0.00 48b1a5943859c3d386d85833d62c7234f9fa2140
4ac4893 > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 16
:29:36 up 2:13, 2 users, load average: 0.00, 0.00, 0.00 d2d2784adf947267f811d30e2265969f5855bf71
6a12a44 > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux
16:29:36 up 2:13, 2 users, load average: 0.00, 0.00, 0.00 90c7070fd728d8f6b2710001f46144517a20c8c6
e7b4e4b > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 16
:16:29 up 2:00, 2 users, load average: 0.00, 0.00, 0.00 a4f239d22c6c03b9538715bb725ac3b9c799938a
7a8cdeb4 > compile 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux
16:16:29 up 2:00, 2 users, load average: 0.00, 0.00, 0.00 9f75300df86a31a0565a4e8f72d961fb2b154db3
94ef93d > run 162020130 shaozhenzhe Linux Debian 4.19.0-18-686 #1 SMP Debian 4.19.208-1 (2021-09-29) i686 GNU/Linux 16
```

实验内容

1.实现标志寄存器（10分）

- 实现标志寄存器 `eflags` 并设置初值（均需要有代码和思路）；

根据i386手册可知eflags寄存器结构如下，这里只要求实现 `CF`, `ZF`, `SF`, `IF`, `OF`，可以利用 `union` 和位域完成

`nemu/include/cpu/reg.h`

```
typedef struct {
    //.....省略

    union{
        uint32_t val;    //设初值用的
        struct{
            uint32_t CF:1;
            uint32_t :5; //中间空出5个无名位域
            uint32_t ZF:1;
            uint32_t SF:1;
            uint32_t :1;
            uint32_t IF:1;
            uint32_t :1;
            uint32_t OF:1;
        };
    }eflags;

} CPU_state;
```

设置初值，在i386手册第10章，提到初始化需要把eflags设为0x2

在 `nemu/src/monitor/monitor.c` 的 `restart()` 函数完成设置

```
static inline void restart() {
    /* Set the initial instruction pointer. */
    cpu.eip = ENTRY_START;
    cpu.eflags.val=0x2;    //赋初值位0x2

#ifdef DIFF_TEST
    init_qemu_reg();
#endif
}
```

- 实现所有指令对标志位的设置（如果该指令有设置标志行为）
在第三题中实现指令时会完成。

2.实现所有 RTL 指令（30 分）

- 你需要实现所有 RTL 指令中未实现的部分（框架 `rtl.h` 中已用 `TODO()` 告知）；

`rtl_mv`

按照所给注释，把 `src1` 赋给 `dest` 即可

```
static inline void rtl_mv(rtlreg_t* dest, const rtlreg_t *src1) {
    // dest <- src1
    //TODO();
    *dest=*src1;
}
```

`rtl_not`

把 `dest` 取反赋值即可

```
static inline void rtl_not(rtlreg_t* dest) {
    // dest <- ~dest
    //TODO();
    *dest = ~(*dest);
}
```

`rtl_sext`

实现符号扩展，可以先把无符号数转换为有符号数，先左移让符号位到达最高位，再右移回到原来的位置，这样就实现了符号扩展

```
static inline void rtl_sext(rtlreg_t* dest, const rtlreg_t* src1, int width)
{
    // dest <- signext(src1[(width * 8 - 1) .. 0])
    //TODO();
    int32_t t = *src1;    //转为带符号整数
    width=32-(width*8);
    *dest = (t<<width)>>width;
}
```

`rtl_push`

根据注释，先取出 `ESP`，`ESP-4` 赋给新的 `ESP`，再把 `src1` 赋给 `ESP` 指向的位置

```
static inline void rtl_push(const rtlreg_t* src1) {
    // esp <- esp - 4
    // M[esp] <- src1
    //TODO();
    rtl_lrl(&t0,R_ESP); //获取ESP的值
    rtl_subi(&t0,&t0,4); //ESP-4
    rtl_srl(R_ESP,&t0); //存储新的ESP
    rtl_sm(&t0,4,src1); //存入PUSH的内容
}
```

rtl_pop

根据注释，先取出ESP，把ESP指向位置的内容赋给 `dest`，再ESP+4赋给新的ESP，

```
static inline void rtl_pop(rtlreg_t* dest) {
    // dest <- M[esp]
    // esp <- esp + 4
    //TODO();
    rtl_lrl(&t0,R_ESP); //获取ESP的值
    rtl_lm(dest,&t0,4); //获取ESP指向的内存
    rtl_addi(&t0,&t0,4); //ESP+4
    rtl_srl(R_ESP,&t0); //存储新的ESP
}
```

rtl_eq0

判断 `src1` 是否为0，如果为0，则dest=1，反之，dest=0

```
static inline void rtl_eq0(rtlreg_t* dest, const rtlreg_t* src1) {
    // dest <- (src1 == 0 ? 1 : 0)
    //TODO();
    if(*src1==0){
        *dest=1;
    }
    else{
        *dest=0;
    }
}
```

rtl_eqi

判断 `src1` 是否为立即数 `imm`，如果相等，则dest=1，反之，dest=0

```
static inline void rtl_eqi(rtlreg_t* dest, const rtlreg_t* src1, int imm) {
    // dest <- (src1 == imm ? 1 : 0)
    //TODO();
    if(*src1==imm){
        *dest=1;
    }
    else{
        *dest=0;
    }
}
```

rtl_neq0

判断 `src1` 是否不为0, 如果不为0, 则`dest=1`, 反之, `dest=0`

```
static inline void rtl_neq0(rtlreg_t* dest, const rtlreg_t* src1) {
    // dest <- (src1 != 0 ? 1 : 0)
    //TODO();
    if(*src1!=0){
        *dest=1;
    }
    else{
        *dest=0;
    }
}
```

`rtl_msb`

根据注释, 该函数获取 `src1` 的最高位, 因为是无符号数, 那么右移即可

```
static inline void rtl_msb(rtlreg_t* dest, const rtlreg_t* src1, int width)
{
    // dest <- src1[width * 8 - 1]
    //TODO();
    rtl_shri(dest,src1,width*8-1);
}
```

`make_rtl_setget_eflags`

这是一个宏定义, 实现了 `rtl_set_f()` 与 `rtl_get_f()` 的功能, 对 `eflags` 对应的标志位进行存取即可

```
#define make_rtl_setget_eflags(f) \
    static inline void concat(rtl_set_, f) (const rtlreg_t* src) { \
        cpu.eflags.f = *src; \
    } \
    static inline void concat(rtl_get_, f) (rtlreg_t* dest) { \
        *dest = cpu.eflags.f; \
    }
```

`rtl_update_ZF`

更新 `ZF`, 根据注释, 需要判断 `result` 是否全为0。

可以通过左移把原来 `width*8-1` 移到最高位, 末尾补0。若移位后的数值为0, 则 `ZF=1` 否则 `ZF=0`

```
static inline void rtl_update_ZF(const rtlreg_t* result, int width) {
    // eflags.ZF <- is_zero(result[width * 8 - 1 .. 0])
    //TODO();
    rtl_shli(&t0,result,32-width*8);
    if(t0==0){
        t1=1;
        rtl_set_ZF(&t1);
    }
    else{
        t1=0;
        rtl_set_ZF(&t1);
    }
}
```

```
rtl_update_SF
```

更新 SF，通过右移获取result符号位即可

```
static inline void rtl_update_SF(const rtlreg_t* result, int width) {  
    // eflags.SF <- is_sign(result[width * 8 - 1 .. 0])  
    //TODO();  
    rtl_shri(&t0,result,width*8-1);  
    rtl_set_SF(&t0);  
}
```

- 由于部分 RTL 暂时无法验证其正确性，只写出代码和思路即可。

3.实现 6 条 x86 指令（30 分）

以下这些指令的执行函数 make_EHelper 都需要在 /nemu/src/cpu/exec/all-instr.h 里声明

```
make_EHelper(call);  
make_EHelper(push);  
make_EHelper(sub);  
make_EHelper(nop);  
make_EHelper(pop);  
make_EHelper(ret);  
make_EHelper(xor);
```

- call

运行 dummy 发现 e8 指令没有实现，查询 i386 手册可知需要实现 call

A 表明操作数是一个立即数，v 表明操作数大小可能为2或4字节

根据视频的提示，先填表 opcode_table，这里只是译码一个立即数，因此填 IDEX(I, call)

```
/* 0xe8 */ IDEX(I, call), EMPTY, EMPTY, EMPTY,
```


在 /nemu/src/cpu/exec/control.c 完成对应的 make_EHelper(call) 函数，这里是eip加上立即数为新的跳转地址

```
make_EHelper(call) {  
    // the target address is calculated at the decode stage  
    //TODO();  
    decoding.is_jump = 1; //设置跳转标记  
    rtl_push(eip);        //压栈  
    rtl_add(&decoding.jump_eip,eip,&id_dest->val); //得到要跳转的地址  
    print_asm("call %x", decoding.jump_eip);  
}
```

运行结果


```
shaozhenzhe@Debian: ~/ics2022/nexus-am/tests/cputest
For help, type "help"
(nemu) si 5
100000: bd 00 00 00 00      movl $0x0,%ebp
100005: bc 00 7c 00 00      movl $0x7c00,%esp
10000a: e8 0f 00 00 00      call 10001e
invalid opcode(eip = 0x0010001e): 55 89 e5 83 ec 18 e8 e6 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x0010001e is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x0010001e) in the disassembling result to distinguish which case
it is.

If it is the first case, see

for more details.

If it is the second case, remember:
* The machine is always right!
* Every line of untested code is always wrong!

10001e: 55 55 89 e5 83 ec 18 e8 e6      invalid opcode
(nemu) █
```

- push

55 指令没有实现，查阅 i386 手册可知需要实现 `push`

选取译码函数 `make_DHelper(r)`，其功能是读取操作码中的寄存器信息，选取执行函数 `make_EHelper(push)`

因此填入 `IDEX(r, push)`，`/nemu/src/cpu/exec/exec.c`，这里从 `0x50` 到 `0x57` 都是 `push` 寄存器，因此一起填完了

```
/* 0x50 */    IDEX(r, push), IDEX(r, push), IDEX(r, push), IDEX(r, push),
/* 0x54 */    IDEX(r, push), IDEX(r, push), IDEX(r, push), IDEX(r, push),
```

`/nemu/src/cpu/exec/data-mov.c`，完善 `make_EHelper(push)`，调用 `rtl_push()` 即可

```
make_EHelper(push) {
    //TODO();
    rtl_push(&id_dest->val); //把内容压入栈
    print_asm_template1(push);
}
```

运行结果

```

shaozhenzhe@Debian:~/ics2022/nexus-am/tests/cputest$ make ARCH=x86-nemu ALL=dummy run
Building dummy [x86-nemu]
Building am [x86-nemu]
make[2]: *** No targets specified and no makefile found. Stop.
[src/monitor/monitor.c,65,load_img] The image is /home/shaozhenzhe/ics2022/nexus-am/tests/cputest/build/dummy-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 21:29:56, Apr 22 2022
For help, type "help"
(nemu) si 10
100000: bd 00 00 00 00          movl $0x0,%ebp
100005: bc 00 7c 00 00          movl $0x7c00,%esp
10000a: e8 0f 00 00 00          call 10001e
10001e: 55                      pushl %ebp
10001f: 89 e5                   movl %esp,%ebp
please implement me
nemu: src/cpu/decode/decode.c:41: decode_op_SI: Assertion `0' failed.
make[2]: *** [Makefile:47: run] Aborted
make[1]: *** [/home/shaozhenzhe/ics2022/nexus-am/Makefile.app:35: run] Error 2
make: [Makefile:13: Makefile.dummy] Error 2 (ignored)
dummy
shaozhenzhe@Debian:~/ics2022/nexus-am/tests/cputest$

```

- sub

上面的运行结果没有给出是哪个指令，猜测是 `opcode_table` 里有这个指令，但是执行过程中有 `TODO()`

查询反汇编结果可知需要实现 83 指令，也就是 sub。首先查表，得到 `IDEX(SI2E, gp1)`

先看译码函数，`make_DopHelper(SI2E)` 里调用了 `decode_op_SI`，因此需要先完善 `make_DopHelper(SI)`

/nemu/src/cpu/decode/decode.c，根据注释的内容用 `instr_fetch()` 读取赋值即可

```

static inline make_DopHelper(SI) {
    assert(op->width == 1 || op->width == 4);

    op->type = OP_TYPE_IMM;

    /* TODO: Use instr_fetch() to read `op->width' bytes of memory
     * pointed by `eip'. Interpret the result as a signed immediate,
     * and assign it to op->simm.
     */
    op->simm = ???
    /*
    //TODO();
    op->simm = instr_fetch(eip, op->width);

    rtl_li(&op->val, op->simm);

#ifdef DEBUG
    snprintf(op->str, OP_STR_SIZE, "$0x%x", op->simm);
#endif
}

```

再看执行函数。这里比较特殊，用了 `grp1`，查阅 i386 手册得知需要在 `grp1[5]` 填写 `make_EHelper(sub)`

```

make_group(gp1,
    EMPTY, EMPTY, EMPTY, EMPTY,
    EMPTY, EX(sub), EMPTY, EMPTY)

```

/nemu/src/cpu/exec/arith.c，完善 `make_EHelper(sub)`

这里参考了框架给出的 `make_EHelper(sbb)` 函数, `sbb` 是在 `sub` 的基础上再减去 `CF` 带借位的减法

因此只要把 `sbb` 里获取 `CF` 和减去 `CF` 的部分去掉就是 `sub`

```
make_EHelper(sub) {
    //TODO();
    rtl_sub(&t2, &id_dest->val, &id_src->val);
    rtl_sltu(&t3, &id_dest->val, &t2);
    //rtl_get_CF(&t1); //注释掉这两句就是sub
    //rtl_sub(&t2, &t2, &t1);
    operand_write(id_dest, &t2);

    rtl_update_ZFSF(&t2, id_dest->width);

    rtl_sltu(&t0, &id_dest->val, &t2);
    rtl_or(&t0, &t3, &t0);
    rtl_set_CF(&t0);

    rtl_xor(&t0, &id_dest->val, &id_src->val);
    rtl_xor(&t1, &id_dest->val, &t2);
    rtl_and(&t0, &t0, &t1);
    rtl_msb(&t0, &t0, id_dest->width);
    rtl_set_OF(&t0);
    print_asm_template2(sub);
}
```

运行结果

```
shaozhenzhe@Debian: ~/ics2022/nexus-am/tests/cputest
--am/tests/cputest/build/dummy-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 21:29:56, Apr 22 2022
For help, type "help"
(nemu) si 10
100000: bd 00 00 00 00      movl $0x0,%ebp
100005: bc 00 7c 00 00      movl $0x7c00,%esp
10000a: e8 0f 00 00 00      call 10001e
10001e: 55                  pushl %ebp
10001f: 89 e5              movl %esp,%ebp
100021: 83 ec 18          subl $0x18,%esp
100024: e8 e6 ff ff ff      call 10000f
10002f: 55                  pushl %ebp
100030: 89 e5              movl %esp,%ebp
invalid opcode(eip = 0x00100012): 90 5d c3 55 89 e5 8b 45 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00100012 is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x00100012) in the disassembling result to distinguish which case it is.

If it is the first case, see
0x00100012: 90 5d c3 55 89 e5 8b 45
for more details.

If it is the second case, remember:
* The machine is always right!
* Every line of untested code is always wrong!

100012: 90 90 5d c3 55 89 e5 8b 45      invalid opcode
(nemu) 
```

- `nop`


`90` 指令没有实现, 查阅 `i386` 手册得知需要实现 `nop`

```
/* 0x90 */ EX(nop), EMPTY, EMPTY, EMPTY,
```

```
shaozhazhe@Debian: ~/ics2022/nexus-am/tests/cputest
```

```
build/dummy-x86-nemu.bin  
Welcome to NEMU!  
[src/monitor/monitor.c,30,welcome] Build time: 21:29:56, Apr 22 2022  
For help, type "help"  
(nemu) si 20  
100000: bd 00 00 00 00 movl $0x0,%ebp  
100005: bc 00 7c 00 00 movl $0x7c00,%esp  
10000a: e8 0f 00 00 00 call 10001e  
10001e: 55 pushl %ebp  
10001f: 89 e5 movl %esp,%ebp  
100021: 83 ec 18 subl $0x18,%esp  
100024: e8 e6 ff ff ff call 10000f  
10000f: 55 pushl %ebp  
100010: 89 e5 movl %esp,%ebp  
100012: 90 nop  
invalid opcode(eip = 0x00100013): 5d c3 55 89 e5 8b 45 08 ...  
  
There are two cases which will trigger this unexpected exception:  
1. The instruction at eip = 0x00100013 is not implemented.  
2. Something is implemented incorrectly.  
Find this eip(0x00100013) in the disassembling result to distinguish which case it is.
```

If it is the first case, see



for more details.

If it is the second case, remember:

- * The machine is always right!
- * Every line of untested code is always wrong!

```
100013: 5d 5d c3 55 89 e5 8b 45 08 invalid opcode  
(nemu)
```


```
/* 0x58 */ IDEX(r, pop), IDEX(r, pop), IDEX(r, pop), IDEX(r, pop),
/* 0x5c */ IDEX(r, pop), IDEX(r, pop), IDEX(r, pop), IDEX(r, pop),
```

```
make_EHelper(pop) {
    //TODO();
    rtl_pop(&id_dest->val);    //rtl_pop把栈顶内容存入id_dest->val
    operand_write(id_dest, &id_dest->val); //模仿mov写入id_dest
    print_asm_template1(pop);
}
```

运行结果

```
shaozhenzhe@Debian: ~/ics2022/nexus-am/tests/cputest
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 21:29:56, Apr 22 2022
For help, type "help"
(nemu) si 20
100000: bd 00 00 00 00      movl $0x0,%ebp
100005: bc 00 7c 00 00      movl $0x7c00,%esp
10000a: e8 0f 00 00 00      call 10001e
10001e: 55                  pushl %ebp
10001f: 89 e5              movl %esp,%ebp
100021: 83 ec 18          subl $0x18,%esp
100024: e8 e6 ff ff ff      call 10000f
10000f: 55                  pushl %ebp
100010: 89 e5              movl %esp,%ebp
100012: 90                  nop
100013: 5d                  popl %ebp
invalid opcode(eip = 0x00100014): c3 55 89 e5 8b 45 08 d6 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00100014 is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x00100014) in the disassembling result to distinguish which case it is.

If it is the first case, see

for more details.

If it is the second case, remember:
* The machine is always right!
* Every line of untested code is always wrong!

100014: c3 c3 55 89 e5 8b 45 08 d6      invalid opcode
(nemu) 
```

- ret

c3 指令没有实现，查阅 i386 手册可知需要实现 `ret`

`ret` 也不需要译码函数，直接填表

```
/* 0xc0 */      IDEXW(gp2_Ib2E, gp2, 1), IDEX(gp2_Ib2E, gp2), EMPTY,
EX(ret),
```

/nemu/src/cpu/exec/control.c

```
make_EHelper(ret) {
    //TODO();
    decoding.is_jump = 1; //设置跳转标记
    rtl_pop(&decoding.jump_eip); //获取栈顶跳转的位置
    print_asm("ret");
}
```

运行结果

```

shaozhenghe@Debian: ~/ics2022/nexus-am/tests/cputest
100014: c3 ret
100029: e8 14 00 00 00 call 100042
100042: 55 pushl %ebp
100043: 89 e5 movl %esp,%ebp
100045: b8 00 00 00 00 movl $0x0,%eax
10004a: 5d popl %ebp
10004b: c3 ret
10002e: 89 45 f4 movl %eax,-0xc(%ebp)
100031: 83 ec 0c subl $0xc,%esp
invalid opcode(eip = 0x00100034): ff 75 f4 e8 d9 ff ff ff ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00100034 is not implemented.
2. Something is implemented incorrectly.
Find this eip(0x00100034) in the disassembling result to distinguish which case it is.

If it is the first case, see


```

- push (ff 指令)

运行过程中出现了 ff 指令，通过查阅得知是 push，查表得到 IDEX(E, qp5)

根据手册得知需要在 `qp5[6]` 的位置填上 `make_EHelper(push)`

```
make_group(gp5,
           EMPTY, EMPTY, EMPTY, EMPTY,
           EMPTY, EMPTY, EX(push), EMPTY)
```

运行结果

```
[src/monitor/monitor.c,65,load_img] The image is /home/shaozhenzhe/ics2022/nexus-am/tests/cputest/build/dummy
~x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 16:53:39, Apr 23 2022
For help, type "help"
(nemu) si 30
100000: bd 00 00 00 00      movl $0x0,%ebp
100005: bc 00 7c 00 00      movl $0x7c00,%esp
10000a: e8 0f 00 00 00      call 10001e
10001e: 55                  pushl %ebp
10001f: 89 e5              movl %esp,%ebp
100021: 83 ec 18          subl $0x18,%esp
100024: e8 e6 ff ff ff      call 10000f
10000f: 55                  pushl %ebp
100010: 89 e5              movl %esp,%ebp
100012: 90                  nop
100013: 5d                  popl %ebp
100014: c3                  ret
100029: e8 14 00 00 00      call 100042
100042: 55                  pushl %ebp
100043: 89 e5              movl %esp,%ebp
100045: b8 00 00 00 00      movl $0x0,%eax
10004a: 5d                  popl %ebp
10004b: c3                  ret
10002e: 89 45 f4           movl %eax,-0xc(%ebp)
100031: 83 ec 0c          subl $0xc,%esp
100034: ff 75 f4          pushl -0xc(%ebp)
100037: e8 d9 ff ff ff      call 100015
100015: 55                  pushl %ebp
100016: 89 e5              movl %esp,%ebp
100018: 8b 45 08           movl 0x8(%ebp),%eax
nemu: HIT GOOD TRAP at eip = 0x0010001b
10001b: d6                  nemu trap (eax = 0)
(nemu)
```

- xor

虽然运行dummy的时候没有出现这条指令，但是题目要求实现。

查阅 i386 手册得到 xor 命令是指令 30-35，根据手册填表

```

/* 0x30 */    IDEXW(G2E, xor, 1), IDEX(G2E, xor), IDEXW(E2G, xor, 1),
IDEX(E2G, xor),
/* 0x34 */    IDEXW(I2a, xor, 1), IDEX(I2r, xor), EMPTY, EMPTY,

```

/nemu/src/cpu/exec/logic.c, 调用rtl_xor, CF和OF都赋为0, 更新ZF、SF即可

```

make_EHelper(xor) {
    //TODO();
    rtl_xor(&id_dest->val, &id_src->val, &id_src2->val); //异或
    operand_write(id_dest, &id_dest->val); //赋值

    rtl_set_CF(&tzero); //CF=0
    rtl_set_OF(&tzero); //OF=0
    rtl_update_ZFSF(&id_dest->val, id_dest->width); //更新ZF、SF

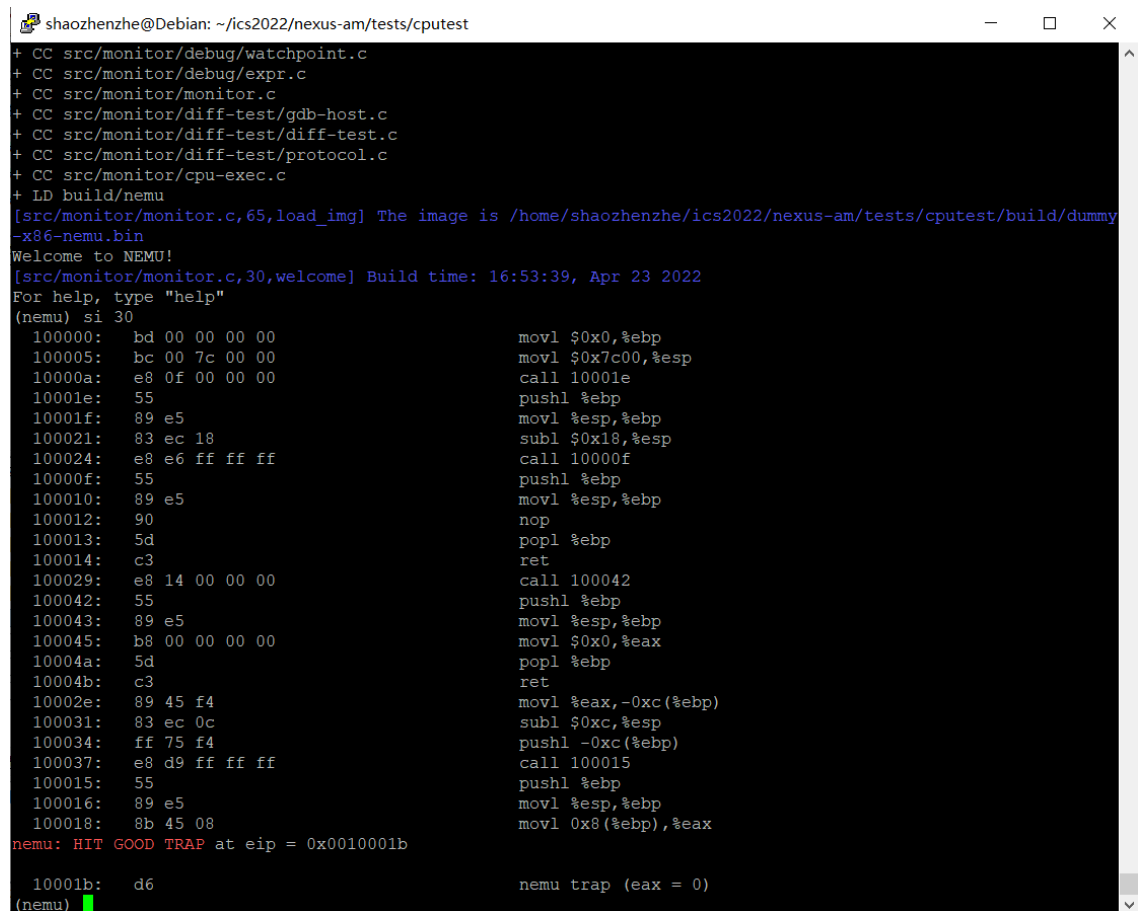
    print_asm_template2(xor);
}

```

4.成功运行 dummy (10 分)

- 截图

输入 si 30 后出现了 HIT GOOD TRAP



```

shaozhenzhe@Debian: ~/ics2022/nexus-am/tests/cputest
+ CC src/monitor/debug/watchpoint.c
+ CC src/monitor/debug/expr.c
+ CC src/monitor/monitor.c
+ CC src/monitor/diff-test/gdb-host.c
+ CC src/monitor/diff-test/diff-test.c
+ CC src/monitor/diff-test/protocol.c
+ CC src/monitor/cpu-exec.c
+ LD build/nemu
[src/monitor/monitor.c,65,load_img] The image is /home/shaozhenzhe/ics2022/nexus-am/tests/cputest/build/dummy
-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 16:53:39, Apr 23 2022
For help, type "help"
(nemu) si 30
100000: bd 00 00 00 00      movl $0x0,%ebp
100005: bc 00 7c 00 00      movl $0x7c00,%esp
10000a: e8 0f 00 00 00      call 10001e
10001e: 55                  pushl %ebp
10001f: 89 e5              movl %esp,%ebp
100021: 83 ec 18          subl $0x18,%esp
100024: e8 e6 ff ff ff      call 10000f
10000f: 55                  pushl %ebp
100010: 89 e5              movl %esp,%ebp
100012: 90                  nop
100013: 5d                  popl %ebp
100014: c3                  ret
100029: e8 14 00 00 00      call 100042
100042: 55                  pushl %ebp
100043: 89 e5              movl %esp,%ebp
100045: b8 00 00 00 00      movl $0x0,%eax
10004a: 5d                  popl %ebp
10004b: c3                  ret
10002e: 89 45 f4          movl %eax,-0xc(%ebp)
100031: 83 ec 0c          subl $0xc,%esp
100034: ff 75 f4          pushl -0xc(%ebp)
100037: e8 d9 ff ff ff      call 100015
100015: 55                  pushl %ebp
100016: 89 e5              movl %esp,%ebp
100018: 8b 45 08          movl 0x8(%ebp),%eax
nemu: HIT GOOD TRAP at eip = 0x0010001b
10001b: d6                  nemu trap (eax = 0)
(nemu)

```

5.实现 Diff-test (20 分)

- 代码、代码思路和截图

在 `/nemu/include/common.h` 中取消注释

```
#define DIFF_TEST
```

在 `nemu/src/monitor/difftest/diff-test.c` 中完善 `difftest_step()` 函数

对8个通用寄存器以及 `eip` 进行比较, 若不同设为 `true`, 相同的话设为 `false`

这里我用了宏, 这样就不用写很多的判断条件了

```
#define diff_reg(reg) \
    if (r.reg != cpu.reg) { \
        diff = true; \
        Log("reg error NEMU.reg=0x%08x QEMU.reg=0x%08x\n",cpu.reg,r.reg); \
    }

void difftest_step(uint32_t eip) {
    union gdb_regs r;
    bool diff = false;

    if (is_skip_nemu) {
        is_skip_nemu = false;
        return;
    }

    if (is_skip_qemu) {
        // to skip the checking of an instruction, just copy the reg state to
        qemu
        gdb_getregs(&r);
        regcpy_from_nemu(r);
        gdb_setregs(&r);
        is_skip_qemu = false;
        return;
    }

    gdb_si();
    gdb_getregs(&r);

    // TODO: Check the registers state with QEMU.
    // Set `diff` as `true` if they are not the same.
    //TODO();
    diff_reg(eax);
    diff_reg(ecx);
    diff_reg(edx);
    diff_reg(ebx);
    diff_reg(ebp);
    diff_reg(esp);
    diff_reg(edi);
    diff_reg(esi);
    diff_reg(eip);

    if (diff) {
        nemu_state = NEMU_END;
    }
}
```



```
}
```

测试结果

出现 Connect to QEMU successfully, 并且出现 HIT GOOD TRAP

```
shaozhenzhe@Debian: ~/ics2022/nexus-am/tests/cptest
1 file changed, 1 insertion(+), 1 deletion(-)
shaozhenzhe@Debian:~/ics2022/nexus-am/tests/cptest$ make ARCH=x86-nemu ALL=dummy run
Building dummy [x86-nemu]
Building am [x86-nemu]
make[2]: *** No targets specified and no makefile found. Stop.
+ CC src/monitor/diff-test/diff-test.c
+ LD build/nemu
[src/monitor/diff-test/diff-test.c,96,init_difftest] Connect to QEMU successfully
[src/monitor/monitor.c,65,load_img] The image is /home/shaozhenzhe/ics2022/nexus-am/tests/cptest/b
uild/dummy-x86-nemu.bin
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 16:08:45, Apr 23 2022
For help, type "help"
(nemu) si 30
100000: bd 00 00 00 00      movl $0x0,%ebp
100005: bc 00 7c 00 00      movl $0x7c00,%esp
10000a: e8 0f 00 00 00      call 10001e
10001e: 55                  pushl %ebp
10001f: 89 e5               movl %esp,%ebp
100021: 83 ec 18            subl $0x18,%esp
100024: e8 e6 ff ff ff      call 10000f
10000f: 55                  pushl %ebp
100010: 89 e5               movl %esp,%ebp
100012: 90                  nop
100013: 5d                  popl %ebp
100014: c3                  ret
100029: e8 14 00 00 00      call 100042
100042: 55                  pushl %ebp
100043: 89 e5               movl %esp,%ebp
100045: b8 00 00 00 00      movl $0x0,%eax
10004a: 5d                  popl %ebp
10004b: c3                  ret
10002e: 89 45 f4            movl %eax,-0xc(%ebp)
100031: 83 ec 0c            subl $0xc,%esp
100034: ff 75 f4            pushl -0xc(%ebp)
100037: e8 d9 ff ff ff      call 100015
100015: 55                  pushl %ebp
100016: 89 e5               movl %esp,%ebp
100018: 8b 45 08            movl 0x8(%ebp),%eax
nemu: HIT GOOD TRAP at eip = 0x0010001b
10001b: d6                  nemu trap (eax = 0)
(nemu)
```

遇到的问题及解决办法

1.问题: 在实现55(push)指令时, 补全了 opcode_table 和 make_EHelper(push), 但是仍然无法执行

解决方法: 重新观看了视频, 观看在实现call时的步骤, 发现自己遗漏了在 all-instr.h 中声明

make_EHelper(push) 函数这一步, 补上之后成功解决。并且在之后的实现指令的过程中, 提前声明函数, 因此不再遇到该问题。

2.实现了所有指令后运行dummy, 程序运行错误, 没有出现 HIT GOOD TRAP

解决方法: 这里不知道错在哪一步。因此考虑 diff-test, 完成 diff-test 后运行, 在pop运行后报了两边寄存器不同的错误。因此查看修改 make_EHelper(pop) 后解决。

实验心得

PA2.1的难度相较于之前要大很多, 主要难点在于涉及了大量的文件, 指令运行过程比较复杂, 需要我们去反复阅读理解代码, 理解程序运行的逻辑才能完成实验。最大的收获是知道了各种精妙的宏定义, 宏定义代替完成了很多繁琐的工作, 如果能够熟练运用宏定义, 能够写出很优美的代码。同时我一直对指令的执行感到神奇, 本次实验让我深入了解了取指—译码—执行的过程, 对冯诺依曼计算机系统的了解更加深刻。

其他备注

无

