PA2-简单复杂的机器: 冯诺依曼计算机系统-实验报告

1 概述

1.1 实验目的

1. 熟悉CPU执行指令的步骤
2. 探究程序运行时的环境
3. 探究`diff-test`
4. 探究模拟输入输出

1.2 实验内容

1. 在NEMU实现部分指令，运行第一个C程序`dummy`
2. 实现更多的指令，能运行所有的`cputest`
3. 实现输入输出，运行打字小游戏

2 阶段一

2.1 指令执行过程

在进行阶段一之前，我们已经了解了TRM的工作方式

```
while (1) {
  从PC指示的存储器位置取出指令;
  执行指令;
  更新PC;
}
```

冯诺依曼体系结构的核心思想就是"存储程序，程序控制"，其实就是取出PC所指向的指令从内存读入到CPU中；CPU通过查表的方式得知这条指令的操作数与操作码；CPU往计算部件里输入操作数就能得到执行结果并写回目的操作数；更新PC指向下一个指令的位置。对应在NEMU中就是调用`instr_fetch()`函数取指；在`opcode_table`里面通过`IDEX()`与`IDEXW()`寻找对应的译码辅助函数`make_DHelper()`与执行辅助函数`make_EHelper`以及操作数宽度，然后进行译码；执行`make_EHelper`里的函数执行相应的指令；调用`update_pc`更新PC。

> 问题一：请整理一条指令在NEMU中的执行过程.
>
> 答：
>
> 1. 在NEMU中主要通过`exec_once()`来执行PC所指向的指令。
> 2. 将当前的PC保存在全局译码信息`decinfo`的成员`seq_pc`中
> 3. 然后把`decinfo.seq_pc`的地址传送给`isa_exec()`
> 4. 然后调用`instr_fetch()`去内存取出指令，并获得`opcode`，放入`decinfo.opcode`中
> 5. 根据`opcode`在`opcode_table`中进行索引，获取的元素将被作为参数调用`idex()`函数来进行译码与执行。
> 6. `idex()`调用译码辅助函数`make_DHelper(name)`来获取指令中的操作数信息
> 7. `idex()`调用执行辅助函数`make_EHelper(name)`使用RTL指令来执行该指令本身的功能，并修改相应的寄存器的值。
> 8. 回到`exec_once()`执行`update_pc()`来让PC指向下一条动态指令的地址

9. 循环。

2.2 运行第一个C程序

在`nexus-am/tests/cputest/`目录下键入

```
make ARCH=$ISA-nemu ALL=dummy run
```

可以看到提示`invaild opcode`，

1. 去`nexus-am/tests/cputest/build/dummy-$ISA-nemu.txt`中查看对应PC对应的指令
2. 然后去查看i386手册找打对应的指令
3. 去`all-instr.h`中填充指令
4. 到`exec.c`中填充`opcode_table`里对应的`idex()`函数，选择对应的译码与执行函数
5. 到对应的执行函数文件里修改执行函数
6. 如果需要RTL指令，还需要到`rtl.h`中实现

2.2.1 `call`

`call`在指令中的位置为E8，

	LOOPNE	LOOPE	LOOP	JCXZ	IN		OUT		CALL
E	Jb	Jb	Jb	Jb	AL,Ib	eAX,Ib	Ib,AL	Ib,eAX	Av

指令详情如下：

```
E8  cd    CALL rel32       7+m            Call near, displacement relative
                                          to next instruction
```

执行操作如下：

Operation

```
IF rel16 or rel32 type of call
THEN (* near relative call *)
   IF OperandSize = 16
   THEN
      Push(IP);
      EIP := (EIP + rel16) AND 0000FFFFH;
   ELSE (* OperandSize = 32 *)
      Push(EIP);
      EIP := EIP + rel32;
   FI;
FI;
```

1. `all-instr.h`中添加

```
make_EHelper(call);
```

2. 修改`opcode_table[]`:

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

3. 因为`call`为跳转指令，所以在`control.c`中实现

```
make_EHelper(call)
{
  // the target address is calculated at the decode stage
  decinfo.is_jmp = 1;
  rtl_push(&decinfo.seq_pc);
  rtl_j(decinfo.jmp_pc);
  print_asm("call %x", decinfo.jmp_pc);
}
```

4. 需要用到`rtl_push()`，到`rtl.h`中实现

```
static inline void rtl_push(const rtlreg_t *src1)
{
  // esp <- esp - 4
  cpu.esp-=4;
  // M[esp] <- src1
  rtl_sm(&cpu.esp, src1, 4);
}
```

2.2.2 push

重复上面的步骤

1. `all-instr.h`中添加

```
make_EHelper(push);
```

2. 修改`opcode_table[]`

```
        /* 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),
        /* 0x68 */ IDEX(push_SI, push), IDEX(I_E2G, imul2),
IDEXW(push_SI, push, 1), EMPTY,
```

3. 因为`push`是数据移动指令，在`data-mov.c`中实现

```
make_EHelper(push) {
  rtl_push(&id_dest->val);


  print_asm_template1(push);
}
```

2.2.3 pusha

重复上面的步骤

1. `all-instr.h`中添加

```
make_EHelper(pusha);
```

2. 修改`opcode_table[]`

```
        /* 0x60 */ EX(pusha), EX(popa), EMPTY, EMPTY,
```

3. 因为`pusha`是数据移动指令，在`data-mov.c`中实现

```
make_EHelper(pusha) {
  // TODO();
  s0=cpu.pc;
  rtl_push(&cpu.eax);
  rtl_push(&cpu.ecx);
  rtl_push(&cpu.edx);
  rtl_push(&cpu.ebx);
  rtl_push(&s0);
  rtl_push(&cpu.ebp);
  rtl_push(&cpu.esi);
  rtl_push(&cpu.edi);
  print_asm("pusha");
}
```

2.2.4 sub

重复步骤

1. `all-instr.h`中添加

```
make_EHelper(sub);
```

2. 修改`opcode_table[]`，手册上`opcode`为83是gp1，所以填充gp1中的指令

```
make_group(gp1,
           EX(add), EX(or), EX(adc), EX(sbb),
           EX(and), EX(sub), EX(xor), EX(cmp))
        /* 0x80 */ IDEXW(I2E, gp1, 1), IDEX(I2E, gp1), EMPTY,
IDEX(SI2E, gp1),
```

3. 修改`arith.c`中的函数

```
make_EHelper(sub)
{
  rtl_sub(&s0, &id_dest->val, &id_src->val);
  operand_write(id_dest, &s0);
  if (id_dest->width != 4)
  {
    rtl_andi(&s0, &s0, 0xffffffffu >> ((4 - id_dest->width) *
8));
  }
```

```
  rtl_update_ZFSF(&s0, id_dest->width);
  rtl_is_sub_carry(&s1, &s0, &id_dest->val);
  rtl_set_CF(&s1);
  rtl_is_sub_overflow(&s1, &s0, &id_dest->val, &id_src->val,
id_dest->width);
  rtl_set_OF(&s1);

  print_asm_template2(sub);
}
```

4. 在`reg.h`中增加`EFLAGS`寄存器

```
    // EFLAGS
    union {
        struct {
            uint32_t CF :1;
            uint32_t dummy0 :1;
            uint32_t PF :1;
            uint32_t dummy1 :1;
            uint32_t AF :1;
            uint32_t dummy2: 1;
            uint32_t ZF :1;
            uint32_t SF :1;
            uint32_t TF :1;
            uint32_t IF :1;
            uint32_t DF :1;
            uint32_t OF :1;
            uint32_t OLIP :2;
            uint32_t NT :1;
            uint32_t dummy3 :1;
            uint32_t RF :1;
            uint32_t VM :1;
            uint32_t dummy4 :14;
        };
        uint32_t val;
    } eflags;
```

5. 修改`rtl.h`中的修改标志位的函数

```
tatic inline void rtl_update_ZF(const rtlreg_t *result, int
width)
{
  // eflags.ZF <- is_zero(result[width * 8 - 1 .. 0])
  switch (width)
  {
  case 1:
    cpu.eflags.ZF = *((int8_t *)result) ? 0 : 1;
    break;
  case 2:
    cpu.eflags.ZF = *((int16_t *)result) ? 0 : 1;
    break;
  case 4:
```

```
    cpu.eflags.ZF = *((int32_t *)result) ? 0 : 1;
    break;
  default:
    assert(0);
  }
}

static inline void rtl_update_SF(const rtlreg_t *result, int
width)
{
  // eflags.SF <- is_sign(result[width * 8 - 1 .. 0])
  switch (width)
  {
  case 1:
    cpu.eflags.SF = ((*((int8_t *)result)) >> 7) ? 1 : 0;
    break;
  case 2:
    cpu.eflags.SF = ((*((int16_t *)result)) >> 15) ? 1 : 0;
    break;
  case 4:
    cpu.eflags.SF = ((*((int32_t *)result)) >> 31) ? 1 : 0;
    break;
  default:
    assert(0);
  }
}

static inline void rtl_is_sub_overflow(rtlreg_t *dest,
                                       const rtlreg_t *res,
const rtlreg_t *src1, const rtlreg_t *src2, int width)
{
   // dest <- is_overflow(src1 - src2)
  // TODO();
  //src1-src2=res
  //减法有溢出是src1与src2的符号不一致，且res的符号与src1相反
    int result = 0;
  if (width == 1) {
    int8_t src1_ = *((int8_t *)src1);
    int8_t src2_ = *((int8_t *)src2);
    int8_t res_ = *((int8_t *)res);
    result = (((src1_ ^ src2_) >> 7) && ((res_ ^ src1_) >>
7)) ? 1 : 0;
  } else if (width == 2) {
    int16_t src1_ = *((int16_t *)src1);
    int16_t src2_ = *((int16_t *)src2);
    int16_t res_ = *((int16_t *)res);
    result = (((src1_ ^ src2_) >> 15) && ((res_ ^ src1_) >>
15)) ? 1 : 0;
  } else if (width == 4) {
    int32_t src1_ = *((int32_t *)src1);
    int32_t src2_ = *((int32_t *)src2);
    int32_t res_ = *((int32_t *)res);
```

```
    result = (((src1_ ^ src2_) >> 31) && ((res_ ^ src1_) >>
31)) ? 1 : 0;
  } else {
    assert(0);
  }
  rtl_li(dest, result);
}

// carry 是在无符号数运算时产生的
static inline void rtl_is_sub_carry(rtlreg_t *dest,
                                    const rtlreg_t *res,
const rtlreg_t *src1)
{
  // dest <- is_carry(src1 - src2)
  //借位直接比大小即可，src1比res小，则肯定借位
  rtl_setrelop(RELOP_LTU,dest,src1,res);

}
```

2.2.5 xor

重复上面的步骤

1. `all-instr.h`中添加

```
make_EHelper(xor);
```

2. 修改`opcode_table[]`

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

3. 因为`xor`是逻辑运算指令，在`logic.c`中实现

```
make_EHelper(xor)
{
  // TODO();
  rtl_xor(&s1, &id_dest->val, &id_src->val);
  s0 = 0;
  rtl_set_OF(&s0);
  rtl_set_CF(&s0);
  rtl_update_ZFSF(&s1, id_dest->width);
  operand_write(id_dest, &s1);
  print_asm_template2(xor);
}
```

2.2.6 ret

重复上面的步骤

1. `all-instr.h`中添加

```
make_EHelper(ret);
```

2. 修改`opcode_table[]`

```
        /* 0xc0 */ IDEXW(gp2_Ib2E, gp2, 1), IDEX(gp2_Ib2E,
gp2), IDEXW(I, ret, 2), EX(ret),
```

3. 因为`ret`是控制指令指令，在`control.c`中实现

```
make_EHelper(ret)
{
  // TODO();
  rtl_pop(&decinfo.jmp_pc);
  rtl_j(decinfo.jmp_pc);

  print_asm("ret");
}
```

4. 实现`rtl.h`中的`rtl_pop`指令

```

static inline void rtl_pop(rtlreg_t *dest)
{
  // dest <- M[esp]
  rtl_lm(dest, &cpu.esp, 4);
  // esp <- esp + 4
  rtl_addi(&cpu.esp, &cpu.esp, 4);
  // TODO();
}
```

2.2.7 特别说明`endbr32`

该指令为Intel新添加的，该指令什么也不做只是验证跳转指令，而该指令由3条指令组成，所以让pc+=3

重复上面的步骤

1. `all-instr.h`中添加

```
make_EHelper(endbr32);
```

2. 修改`opcode_table[]`

```
        /* 0xf0 */ EMPTY, EMPTY, EMPTY, EX(endbr32),
```

3. 因为`endbr32`是特殊指令，在`special.c`中实现

```
make_EHelper(endbr32)
{
  decinfo.seq_pc += 3;
  // rtl_push(&cpu.ebp);
  // rtl_mv(&cpu.eax,&cpu.esp);
  // s0=4;
  // rtl_sub(&cpu.esp,&cpu.esp, &s0);
  print_asm("endbr32");
}
```

2.2.8 运行结果

```
Welcome to x86-NEMU!
For help, type "help"
nemu: HIT GOOD TRAP at pc = 0x0010002c
```

3 阶段二

3.1 Differential Testing

指令会变得更多与复杂，所以先实现`Differential Testing`很重要，

1. 打开`common.h`中的`#define DIFF_TEST`
2. 修改`x86/diff-test.c`中`isa_difftest_checkregs()`：

```
bool isa_difftest_checkregs(CPU_state *ref_r, vaddr_t pc)
{
  if (cpu.pc != ref_r->pc)
  {
    printf("input pc %x\n", cpu.pc);
    printf("ref pc %x\n", ref_r->pc);
    printf("wrong in pc\n");
    return false;
  }
  for (int index = 0; index < 8; index++)
  {
    if (cpu.gpr[index]._32 != ref_r->gpr[index]._32)
    {
      printf("wrong in gpr[%d], value is %x\n", index, ref_r-
>gpr[index]._32);
      return false;
    }
  }
    return true;
}
```

3.2 运行更多的程序

修改`nexus-am`中的`Makefile.check`里的`ARCH?=native`为`ARCH?=x86`，然后在`nexus-am/tests/cputest`中执行

```
make ALL=xxx run
```

xxx为测试程序的名字，如`dummy`

3.3 实现更多指令

根据阶段一的步骤实现所有需要的指令。

3.3.1 特别说明`notrack`

`notrack`也是一个在i386手册上不存在的指令，没有查到相关资料，到github上发现有人这样解决。

1. `all-instr.h`中添加

```
make_EHelper(notrack);
```

2. 修改`opcode_table[]`

```
        /* 0x3c */ IDEXW(I2a, cmp, 1), IDEX(I2a, cmp),
EX(notrack), EMPTY,
```

3. 因为`notrack`是特殊指令，在`special.c`中实现

```
make_EHelper(notrack)
{
  isa_exec(pc);
}
```

3.3.2 `all-instr.h`

```
#include "cpu/exec.h"

make_EHelper(mov);

make_EHelper(operand_size);

make_EHelper(inv);
make_EHelper(nemu_trap);

//control.c
make_EHelper(call);
make_EHelper(call_rm);
make_EHelper(ret);
make_EHelper(jmp);
make_EHelper(jmp_rm);
make_EHelper(jcc);
//data-mov.c
make_EHelper(push);
make_EHelper(pop);
make_EHelper(pusha);
make_EHelper(popa);
```

```
make_EHelper(lea);
make_EHelper(movzx);
make_EHelper(cltd);
make_EHelper(movsx);
make_EHelper(cwtl);
make_EHelper(movsb);


//arith.c
make_EHelper(sub);
make_EHelper(add);
make_EHelper(adc);
make_EHelper(sbb);
make_EHelper(cmp);
make_EHelper(inc);
make_EHelper(dec);
make_EHelper(leave);
make_EHelper(imul);
make_EHelper(imul1);
make_EHelper(imul2);
make_EHelper(mul);
make_EHelper(idiv);
make_EHelper(div);
make_EHelper(neg);
//cc.c
make_EHelper(setcc);

//logic.c
make_EHelper(xor);
make_EHelper(or);
make_EHelper(and);
make_EHelper(test);
make_EHelper(sar);
make_EHelper(shl);
make_EHelper(not);
make_EHelper(shr);
make_EHelper(rol);

//special.c
make_EHelper(endbr32);
make_EHelper(nop);
make_EHelper(notrack);
//system.c
make_EHelper(in);
make_EHelper(out);
make_EHelper(lidt);
make_EHelper(int);
```

3.3.3 exec.c

```
#include "cpu/exec.h"
```

```
#include "all-instr.h"

static inline void set_width(int width)
{
  if (width == 0)
  {
    width = decinfo.isa.is_operand_size_16 ? 2 : 4;
  }
  decinfo.src.width = decinfo.dest.width = decinfo.src2.width =
width;
}

static make_EHelper(2byte_esc);

#define make_group(name, item0, item1, item2, item3, item4,
item5, item6, item7) \
  static OpcodeEntry concat(opcode_table_, name)[8] = {
                \
      /* 0x00 */ item0, item1, item2, item3,
                 \
      /* 0x04 */ item4, item5, item6, item7};
                \
  static make_EHelper(name)
                \
  {
                \
    idex(pc, &concat(opcode_table_, name)
[decinfo.isa.ext_opcode]);              \
  }

/* 0x80, 0x81, 0x83 */
make_group(gp1,
           EX(add), EX(or), EX(adc), EX(sbb),
           EX(and), EX(sub), EX(xor), EX(cmp))

    /* 0xc0, 0xc1, 0xd0, 0xd1, 0xd2, 0xd3 */
    make_group(gp2,
               EX(rol), EMPTY, EMPTY, EMPTY,
               EX(shl), EX(shr), EMPTY, EX(sar))

    /* 0xf6, 0xf7 */
    make_group(gp3,
               IDEX(test_I, test), EMPTY, EX(not), EX(neg),
               EX(mul), EX(imul1), EX(div), EX(idiv))

    /* 0xfe */
    make_group(gp4,
               EX(inc), EX(dec), EMPTY, EMPTY,
               EMPTY, EMPTY, EMPTY, EMPTY)

    /* 0xff */
    make_group(gp5,
```

```
                EX(inc), EX(dec), EX(call_rm), EX(call),
                EX(jmp_rm), EMPTY, EX(push), EMPTY)

    /* 0x0f 0x01*/
    make_group(gp7,
                EMPTY, EMPTY, EMPTY, EX(lidt),
                EMPTY, EMPTY, EMPTY, EMPTY)

    /* TODO: Add more instructions!!! */

    static OpcodeEntry opcode_table[512] = {
        /* 0x00 */ IDEXW(G2E, add, 1), IDEX(G2E, add), IDEXW(E2G,
add, 1), IDEX(E2G, add),
        /* 0x04 */ IDEXW(I2a, add, 1), IDEX(I2a, add), EMPTY,
EMPTY,
        /* 0x08 */ IDEXW(G2E, or, 1), IDEX(G2E, or), IDEXW(E2G,
or, 1), IDEX(E2G, or),
        /* 0x0c */ IDEXW(I2a, or, 1), IDEX(I2a, or), EMPTY,
EX(2byte_esc),
        /* 0x10 */ IDEXW(G2E, adc, 1), IDEX(G2E, adc), IDEXW(E2G,
adc, 1), IDEX(E2G, adc),
        /* 0x14 */ IDEXW(I2a, adc, 1), IDEX(I2a, adc), EMPTY,
EMPTY,
        /* 0x18 */ IDEXW(G2E, sbb, 1), IDEX(G2E, sbb), IDEXW(E2G,
sbb, 1), IDEX(E2G, sbb),
        /* 0x1c */ IDEXW(I2a, sbb, 1), IDEX(I2a, sbb), EMPTY,
EMPTY,
        /* 0x20 */ IDEXW(G2E, and, 1), IDEX(G2E, and), IDEXW(E2G,
and, 1), IDEX(E2G, and),
        /* 0x24 */ IDEXW(I2a, and, 1), IDEX(I2a, and), EMPTY,
EMPTY,
        /* 0x28 */ IDEXW(G2E, sub, 1), IDEX(G2E, sub), IDEXW(E2G,
sub, 1), IDEX(E2G, sub),
        /* 0x2c */ IDEXW(I2a, sub, 1), IDEX(I2a, sub), EMPTY,
EMPTY,
        /* 0x30 */ IDEXW(G2E, xor, 1), IDEX(G2E, xor), IDEXW(E2G,
xor, 1), IDEX(E2G, xor),
        /* 0x34 */ IDEXW(I2a, xor, 1), IDEX(I2a, xor), EMPTY,
EMPTY,
        /* 0x38 */ IDEXW(G2E, cmp, 1), IDEX(G2E, cmp), IDEXW(E2G,
cmp, 1), IDEX(E2G, cmp),
        /* 0x3c */ IDEXW(I2a, cmp, 1), IDEX(I2a, cmp),
EX(notrack), EMPTY,
        /* 0x40 */ IDEX(r, inc), IDEX(r, inc), IDEX(r, inc),
IDEX(r, inc),
        /* 0x44 */ IDEX(r, inc), IDEX(r, inc), IDEX(r, inc),
IDEX(r, inc),
        /* 0x48 */ IDEX(r, dec), IDEX(r, dec), IDEX(r, dec),
IDEX(r, dec),
        /* 0x4c */ IDEX(r, dec), IDEX(r, dec), IDEX(r, dec),
IDEX(r, dec),
```

```
        /* 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),
        /* 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),
        /* 0x60 */ EX(pusha), EX(popa), EMPTY, EMPTY,
        /* 0x64 */ EMPTY, EMPTY, EX(operand_size), EMPTY,
        /* 0x68 */ IDEX(push_SI, push), IDEX(I_E2G, imul2),
IDEXW(push_SI, push, 1), EMPTY,
        /* 0x6c */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x70 */ IDEXW(J, jcc, 1), IDEXW(J, jcc, 1), IDEXW(J,
jcc, 1), IDEXW(J, jcc, 1),
        /* 0x74 */ IDEXW(J, jcc, 1), IDEXW(J, jcc, 1), IDEXW(J,
jcc, 1), IDEXW(J, jcc, 1),
        /* 0x78 */ IDEXW(J, jcc, 1), IDEXW(J, jcc, 1), IDEXW(J,
jcc, 1), IDEXW(J, jcc, 1),
        /* 0x7c */ IDEXW(J, jcc, 1), IDEXW(J, jcc, 1), IDEXW(J,
jcc, 1), IDEXW(J, jcc, 1),
        /* 0x80 */ IDEXW(I2E, gp1, 1), IDEX(I2E, gp1), EMPTY,
IDEX(SI2E, gp1),
        /* 0x84 */ IDEXW(G2E, test, 1), IDEX(G2E, test), EMPTY,
EMPTY,
        /* 0x88 */ IDEXW(mov_G2E, mov, 1), IDEX(mov_G2E, mov),
IDEXW(mov_E2G, mov, 1), IDEX(mov_E2G, mov),
        /* 0x8c */ EMPTY, IDEX(lea_M2G, lea), EMPTY, EMPTY,
        /* 0x90 */ EX(nop), EMPTY, EMPTY, EMPTY,
        /* 0x94 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x98 */ EX(cwtl), EX(cltd), EMPTY, EMPTY,
        /* 0x9c */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xa0 */ IDEXW(O2a, mov, 1), IDEX(O2a, mov), IDEXW(a2O,
mov, 1), IDEX(a2O, mov),
        /* 0xa4 */ EX(movsb), EX(movsb), EMPTY, EMPTY,
        /* 0xa8 */ IDEXW(I2a,test,1),IDEX(I2a,test), EMPTY,
EMPTY,
        /* 0xac */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xb0 */ IDEXW(mov_I2r, mov, 1), IDEXW(mov_I2r, mov,
1), IDEXW(mov_I2r, mov, 1), IDEXW(mov_I2r, mov, 1),
        /* 0xb4 */ IDEXW(mov_I2r, mov, 1), IDEXW(mov_I2r, mov,
1), IDEXW(mov_I2r, mov, 1), IDEXW(mov_I2r, mov, 1),
        /* 0xb8 */ IDEX(mov_I2r, mov), IDEX(mov_I2r, mov),
IDEX(mov_I2r, mov), IDEX(mov_I2r, mov),
        /* 0xbc */ IDEX(mov_I2r, mov), IDEX(mov_I2r, mov),
IDEX(mov_I2r, mov), IDEX(mov_I2r, mov),
        /* 0xc0 */ IDEXW(gp2_Ib2E, gp2, 1), IDEX(gp2_Ib2E, gp2),
IDEXW(I, ret, 2), EX(ret),
        /* 0xc4 */ EMPTY, EMPTY, IDEXW(mov_I2E, mov, 1),
IDEX(mov_I2E, mov),
        /* 0xc8 */ EMPTY, EX(leave), EMPTY, EMPTY,
        /* 0xcc */ EMPTY, IDEXW(I,int,1), EMPTY, EMPTY,
```

```
        /* 0xd0 */ IDEXW(gp2_1_E, gp2, 1), IDEX(gp2_1_E, gp2),
IDEXW(gp2_cl2E, gp2, 1), IDEX(gp2_cl2E, gp2),
        /* 0xd4 */ EMPTY, EMPTY, EX(nemu_trap), EMPTY,
        /* 0xd8 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xdc */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xe0 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xe4 */ IDEXW(in_I2a, in, 1), IDEX(in_I2a, in),
IDEXW(out_a2I,out,1), IDEX(out_a2I,out),
        /* 0xe8 */ IDEX(J, call), IDEX(J, jmp), EMPTY, IDEXW(J,
jmp, 1),
        /* 0xec */ IDEXW(in_dx2a,in,1),
IDEX(in_dx2a,in),IDEXW(out_a2dx,out,1), IDEX(out_a2dx,out),
        /* 0xf0 */ EMPTY, EMPTY, EMPTY, EX(endbr32),
        /* 0xf4 */ EMPTY, EMPTY, IDEXW(E, gp3, 1), IDEX(E, gp3),
        /* 0xf8 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xfc */ EMPTY, EMPTY, IDEXW(E, gp4, 1), IDEX(E, gp5),

        /*2 byte_opcode_table */

        /* 0x00 */ EMPTY, IDEX(gp7_E, gp7), EMPTY, EMPTY,
        /* 0x04 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x08 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x0c */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x10 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x14 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x18 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x1c */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x20 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x24 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x28 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x2c */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x30 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x34 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x38 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x3c */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x40 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x44 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x48 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x4c */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x50 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x54 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x58 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x5c */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x60 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x64 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x68 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x6c */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x70 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x74 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x78 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0x7c */ EMPTY, EMPTY, EMPTY, EMPTY,
```

```
        /* 0x80 */ IDEX(J, jcc), IDEX(J, jcc), IDEX(J, jcc),
IDEX(J, jcc),
        /* 0x84 */ IDEX(J, jcc), IDEX(J, jcc), IDEX(J, jcc),
IDEX(J, jcc),
        /* 0x88 */ IDEX(J, jcc), IDEX(J, jcc), IDEX(J, jcc),
IDEX(J, jcc),
        /* 0x8c */ IDEX(J, jcc), IDEX(J, jcc), IDEX(J, jcc),
IDEX(J, jcc),
        /* 0x90 */ IDEXW(setcc_E, setcc, 1), IDEXW(setcc_E,
setcc, 1), IDEXW(setcc_E, setcc, 1), IDEXW(setcc_E, setcc, 1),
        /* 0x94 */ IDEXW(setcc_E, setcc, 1), IDEXW(setcc_E,
setcc, 1), IDEXW(setcc_E, setcc, 1), IDEXW(setcc_E, setcc, 1),
        /* 0x98 */ IDEXW(setcc_E, setcc, 1), IDEXW(setcc_E,
setcc, 1), IDEXW(setcc_E, setcc, 1), IDEXW(setcc_E, setcc, 1),
        /* 0x9c */ IDEXW(setcc_E, setcc, 1), IDEXW(setcc_E,
setcc, 1), IDEXW(setcc_E, setcc, 1), IDEXW(setcc_E, setcc, 1),
        /* 0xa0 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xa4 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xa8 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xac */ EMPTY, EMPTY, EMPTY, IDEX(E2G, imul2),
        /* 0xb0 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xb4 */ EMPTY, EMPTY, IDEXW(mov_E2G, movzx, 1),
IDEXW(mov_E2G, movzx, 2),
        /* 0xb8 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xbc */ EMPTY, EMPTY, IDEXW(mov_E2G,movsx,1),
IDEXW(mov_E2G,movsx,2),
        /* 0xc0 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xc4 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xc8 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xcc */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xd0 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xd4 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xd8 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xdc */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xe0 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xe4 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xe8 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xec */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xf0 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xf4 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xf8 */ EMPTY, EMPTY, EMPTY, EMPTY,
        /* 0xfc */ EMPTY, EMPTY, EMPTY, EMPTY};

static make_EHelper(2byte_esc)
{
  uint32_t opcode = instr_fetch(pc, 1) | 0x100;
  decinfo.opcode = opcode;
  set_width(opcode_table[opcode].width);
  idex(pc, &opcode_table[opcode]);
}

void isa_exec(vaddr_t *pc)
```

```
{
  uint32_t opcode = instr_fetch(pc, 1);
  decinfo.opcode = opcode;
  set_width(opcode_table[opcode].width);
  idex(pc, &opcode_table[opcode]);
}
```

3.3.4 arith.c

```
#include "cpu/exec.h"

make_EHelper(add)
{
  // TODO();
  rtl_add(&s0, &id_dest->val, &id_src->val);
  operand_write(id_dest, &s0);
  if (id_dest->width != 4)
  {
    rtl_andi(&s0, &s0, 0xffffffffu >> ((4 - id_dest->width) *
8));
  }
  rtl_update_ZFSF(&s0, id_dest->width);
  rtl_is_add_carry(&s1, &s0, &id_dest->val);
  rtl_set_CF(&s1);

  rtl_is_add_overflow(&s1, &s0, &id_dest->val, &id_src->val,
id_dest->width);
  rtl_set_OF(&s1);

  print_asm_template2(add);
}

make_EHelper(sub)
{
  rtl_sub(&s0, &id_dest->val, &id_src->val);
  operand_write(id_dest, &s0);
  if (id_dest->width != 4)
  {
    rtl_andi(&s0, &s0, 0xffffffffu >> ((4 - id_dest->width) *
8));
  }
  rtl_update_ZFSF(&s0, id_dest->width);
  rtl_is_sub_carry(&s1, &s0, &id_dest->val);
  rtl_set_CF(&s1);
  rtl_is_sub_overflow(&s1, &s0, &id_dest->val, &id_src->val,
id_dest->width);
  rtl_set_OF(&s1);

  print_asm_template2(sub);
}
```

```
make_EHelper(cmp)
{
  // TODO();
  // printf("left%x\n",id_dest->val);
  // printf("right%x\n",id_src->val);

  rtl_sub(&s0, &id_dest->val, &id_src->val);
  // operand_write(id_dest, &s0);
  if (id_dest->width != 4)
  {
    rtl_andi(&s0, &s0, 0xffffffffu >> ((4 - id_dest->width) *
8));
  }
  rtl_update_ZFSF(&s0, id_dest->width);
  rtl_is_sub_carry(&s1, &s0, &id_dest->val);
  // printf("CF:s1,%d",s1);
  rtl_set_CF(&s1);
  rtl_is_sub_overflow(&s1, &s0, &id_dest->val, &id_src->val,
id_dest->width);
  rtl_set_OF(&s1);

  print_asm_template2(cmp);
}

make_EHelper(inc)
{
  // TODO();
  rtl_addi(&s0, &id_dest->val, 1);
  operand_write(id_dest, &s0);
  if (id_dest->width != 4)
  {
    rtl_andi(&s0, &s0, 0xffffffffu >> ((4 - id_dest->width) *
8));
  }
  rtl_update_ZFSF(&s0, id_dest->width);
  rtl_is_add_carry(&s1, &s0, &id_dest->val);
  rtl_set_CF(&s1);

  rtl_is_add_overflow(&s1, &s0, &id_dest->val, &id_src->val,
id_dest->width);
  rtl_set_OF(&s1);

  print_asm_template1(inc);
}

make_EHelper(dec)
{
  // TODO();
  rtl_subi(&s0, &id_dest->val, 1);
  operand_write(id_dest, &s0);
  if (id_dest->width != 4)
```

```
  {
    rtl_andi(&s0, &s0, 0xffffffffu >> ((4 - id_dest->width) *
8));
  }
  rtl_update_ZFSF(&s0, id_dest->width);
  rtl_is_sub_carry(&s1, &s0, &id_dest->val);
  rtl_set_CF(&s1);
  rtl_is_sub_overflow(&s1, &s0, &id_dest->val, &id_src->val,
id_dest->width);
  rtl_set_OF(&s1);

  print_asm_template1(dec);
}

make_EHelper(neg)
{
  // TODO();

  rtl_mv(&s0, &id_dest->val);
  rtl_not(&s0, &s0);
  rtl_addi(&s0, &s0, 1);
  operand_write(id_dest, &s0);

  s1 = (id_dest->val != 0);
  rtl_set_CF(&s1);

  rtl_update_ZFSF(&s0, id_dest->width);
  rtl_xor(&s1, &s0, &id_dest->val);
  rtl_not(&s1, &s1);
  rtl_msb(&s1, &s1, id_dest->width);
  rtl_set_OF(&s1);
  print_asm_template1(neg);
}

make_EHelper(adc)
{
  // s0 = dest + src
  rtl_add(&s0, &id_dest->val, &id_src->val);
  // s1 = s0 + CF
  rtl_get_CF(&s1);
  rtl_add(&s1, &s0, &s1);

  operand_write(id_dest, &s1);

  if (id_dest->width != 4)
  {
    rtl_andi(&s1, &s1, 0xffffffffu >> ((4 - id_dest->width) *
8));
  }

  rtl_update_ZFSF(&s1, id_dest->width);
```

```
  // update CF
  rtl_is_add_carry(&s1, &s1, &s0);
  rtl_is_add_carry(&s0, &s0, &id_dest->val);
  rtl_or(&s0, &s0, &s1);
  rtl_set_CF(&s0);

  // update OF
  rtl_is_add_overflow(&s0, &s1, &id_dest->val, &id_src->val,
id_dest->width);
  rtl_set_OF(&s0);

  print_asm_template2(adc);
}

make_EHelper(sbb)
{
  // s0 = dest - src
  rtl_sub(&s0, &id_dest->val, &id_src->val);
  // s1 = s0 - CF
  rtl_get_CF(&s1);
  rtl_sub(&s1, &s0, &s1);

  operand_write(id_dest, &s1);

  if (id_dest->width != 4)
  {
    rtl_andi(&s1, &s1, 0xffffffffu >> ((4 - id_dest->width) *
8));
  }

  rtl_update_ZFSF(&s1, id_dest->width);

  // update CF
  rtl_is_sub_carry(&s1, &s1, &s0);
  rtl_is_sub_carry(&s0, &s0, &id_dest->val);
  rtl_or(&s0, &s0, &s1);
  rtl_set_CF(&s0);

  // update OF
  rtl_is_sub_overflow(&s0, &s1, &id_dest->val, &id_src->val,
id_dest->width);
  rtl_set_OF(&s0);

  print_asm_template2(sbb);
}

make_EHelper(mul)
{
  rtl_lr(&s0, R_EAX, id_dest->width);
  rtl_mul_lo(&s1, &id_dest->val, &s0);

  switch (id_dest->width)
```

```
  {
  case 1:
    rtl_sr(R_AX, &s1, 2);
    break;
  case 2:
    rtl_sr(R_AX, &s1, 2);
    rtl_shri(&s1, &s1, 16);
    rtl_sr(R_DX, &s1, 2);
    break;
  case 4:
    rtl_mul_hi(&s0, &id_dest->val, &s0);
    rtl_sr(R_EDX, &s0, 4);
    rtl_sr(R_EAX, &s1, 4);
    break;
  default:
    assert(0);
  }

  print_asm_template1(mul);
}

// imul with one operand
make_EHelper(imul1)
{
  rtl_lr(&s0, R_EAX, id_dest->width);
  rtl_imul_lo(&s1, &id_dest->val, &s0);

  switch (id_dest->width)
  {
  case 1:
    rtl_sr(R_AX, &s1, 2);
    break;
  case 2:
    rtl_sr(R_AX, &s1, 2);
    rtl_shri(&s1, &s1, 16);
    rtl_sr(R_DX, &s1, 2);
    break;
  case 4:
    rtl_imul_hi(&s0, &id_dest->val, &s0);
    rtl_sr(R_EDX, &s0, 4);
    rtl_sr(R_EAX, &s1, 4);
    break;
  default:
    assert(0);
  }

  print_asm_template1(imul);
}

// imul with two operands
make_EHelper(imul2)
{
```

```
  rtl_sext(&s0, &id_src->val, id_src->width);
  rtl_sext(&s1, &id_dest->val, id_dest->width);

  rtl_imul_lo(&s0, &s1, &s0);
  operand_write(id_dest, &s0);

  print_asm_template2(imul);
}

// imul with three operands
make_EHelper(imul3)
{
  rtl_sext(&s0, &id_src->val, id_src->width);
  rtl_sext(&s1, &id_src2->val, id_src->width);

  rtl_imul_lo(&s0, &s1, &s0);
  operand_write(id_dest, &s0);

  print_asm_template3(imul);
}

make_EHelper(div)
{
  switch (id_dest->width)
  {
  case 1:
    rtl_lr(&s0, R_AX, 2);
    rtl_div_q(&s1, &s0, &id_dest->val);
    rtl_sr(R_AL, &s1, 1);
    rtl_div_r(&s1, &s0, &id_dest->val);
    rtl_sr(R_AH, &s1, 1);
    break;
  case 2:
    rtl_lr(&s0, R_AX, 2);
    rtl_lr(&s1, R_DX, 2);
    rtl_shli(&s1, &s1, 16);
    rtl_or(&s0, &s0, &s1);
    rtl_div_q(&s1, &s0, &id_dest->val);
    rtl_sr(R_AX, &s1, 2);
    rtl_div_r(&s1, &s0, &id_dest->val);
    rtl_sr(R_DX, &s1, 2);
    break;
  case 4:
    rtl_lr(&s0, R_EAX, 4);
    rtl_lr(&s1, R_EDX, 4);
    rtl_div64_q(&cpu.eax, &s1, &s0, &id_dest->val);
    rtl_div64_r(&cpu.edx, &s1, &s0, &id_dest->val);
    break;
  default:
    assert(0);
  }
```

```
  print_asm_template1(div);
}

make_EHelper(idiv)
{
  switch (id_dest->width)
  {
  case 1:
    rtl_lr(&s0, R_AX, 2);
    rtl_idiv_q(&s1, &s0, &id_dest->val);
    rtl_sr(R_AL, &s1, 1);
    rtl_idiv_r(&s1, &s0, &id_dest->val);
    rtl_sr(R_AH, &s1, 1);
    break;
  case 2:
    rtl_lr(&s0, R_AX, 2);
    rtl_lr(&s1, R_DX, 2);
    rtl_shli(&s1, &s1, 16);
    rtl_or(&s0, &s0, &s1);
    rtl_idiv_q(&s1, &s0, &id_dest->val);
    rtl_sr(R_AX, &s1, 2);
    rtl_idiv_r(&s1, &s0, &id_dest->val);
    rtl_sr(R_DX, &s1, 2);
    break;
  case 4:
    rtl_lr(&s0, R_EAX, 4);
    rtl_lr(&s1, R_EDX, 4);
    rtl_idiv64_q(&cpu.eax, &s1, &s0, &id_dest->val);
    rtl_idiv64_r(&cpu.edx, &s1, &s0, &id_dest->val);
    break;
  default:
    assert(0);
  }

  print_asm_template1(idiv);
}
```

3.3.5 control.c

```
#include "cpu/exec.h"
#include "cc.h"

make_EHelper(jmp)
{
  // the target address is calculated at the decode stage
  rtl_j(decinfo.jmp_pc);

  print_asm("jmp %x", decinfo.jmp_pc);
}
```

```
make_EHelper(jcc)
{
  // the target address is calculated at the decode stage
  uint32_t cc = decinfo.opcode & 0xf;
  rtl_setcc(&s0, cc);
  rtl_li(&s1, 0);
  // printf("jmp_pc:%x\n",decinfo.jmp_pc);
  rtl_jrelop(RELOP_NE, &s0, &s1, decinfo.jmp_pc);

  print_asm("j%s %x", get_cc_name(cc), decinfo.jmp_pc);
}

make_EHelper(jmp_rm)
{
  rtl_jr(&id_dest->val);

  print_asm("jmp *%s", id_dest->str);
}

make_EHelper(call)
{
  // the target address is calculated at the decode stage
  decinfo.is_jmp = 1;
  rtl_push(&decinfo.seq_pc);
  rtl_j(decinfo.jmp_pc);
  print_asm("call %x", decinfo.jmp_pc);
}

make_EHelper(ret)
{
  // TODO();
  rtl_pop(&decinfo.jmp_pc);
  rtl_j(decinfo.jmp_pc);

  print_asm("ret");
}

make_EHelper(ret_imm)
{
  TODO();

  print_asm("ret %s", id_dest->str);
}

make_EHelper(call_rm)
{
  // TODO();
  rtl_push(&decinfo.seq_pc);
  rtl_jr(&id_dest->val);
  print_asm("call *%s", id_dest->str);
}
```

3.3.6 data-mov.c

```
#include "cpu/exec.h"

make_EHelper(mov) {
  operand_write(id_dest, &id_src->val);
  print_asm_template2(mov);
}

make_EHelper(push) {
  rtl_push(&id_dest->val);


  print_asm_template1(push);
}

make_EHelper(pop) {
rtl_pop(&id_src->val);
operand_write(id_dest,&id_src->val);
  print_asm_template1(pop);
}

make_EHelper(pusha) {
  // TODO();
  s0=cpu.pc;
  rtl_push(&cpu.eax);
  rtl_push(&cpu.ecx);
  rtl_push(&cpu.edx);
  rtl_push(&cpu.ebx);
  rtl_push(&s0);
  rtl_push(&cpu.ebp);
  rtl_push(&cpu.esi);
  rtl_push(&cpu.edi);
  print_asm("pusha");
}

make_EHelper(popa) {
  // TODO();
rtl_pop(&cpu.edi);
rtl_pop(&cpu.esi);
rtl_pop(&cpu.ebp);
rtl_pop(&s0);
rtl_pop(&cpu.ebx);
rtl_pop(&cpu.edx);
rtl_pop(&cpu.ecx);
rtl_pop(&cpu.eax);
  print_asm("popa");
}

make_EHelper(leave) {
  // TODO();
```

```
  rtl_mv(&cpu.esp,&cpu.ebp);
  rtl_pop(&cpu.ebp);

  print_asm("leave");
}

make_EHelper(cltd) {
  if (decinfo.isa.is_operand_size_16) {
    // TODO();
    rtl_sext(&s0, &cpu.eax, 2);
    rtl_shri(&cpu.edx,&s0,16);
  }
  else {
    // TODO();
    rtl_sari(&cpu.edx,&cpu.eax,31);
  }

  print_asm(decinfo.isa.is_operand_size_16 ? "cwtl" : "cltd");
}

make_EHelper(cwtl) {
  if (decinfo.isa.is_operand_size_16) {
    // TODO();

    rtl_shli(&reg_l(R_EAX),&reg_l(R_EAX),24);
    rtl_sari(&reg_l(R_EAX),&reg_l(R_EAX),8);
    rtl_shri(&reg_l(R_EAX),&reg_l(R_EAX),16);
  }
  else {
    // TODO();
    rtl_sext(&reg_l(R_EAX),&reg_l(R_EAX),2);

  }

  print_asm(decinfo.isa.is_operand_size_16 ? "cbtw" : "cwtl");
}

make_EHelper(movsx) {
  id_dest->width = decinfo.isa.is_operand_size_16 ? 2 : 4;
  rtl_sext(&s0, &id_src->val, id_src->width);
  operand_write(id_dest, &s0);
  print_asm_template2(movsx);
}

make_EHelper(movzx) {
  id_dest->width = decinfo.isa.is_operand_size_16 ? 2 : 4;
  operand_write(id_dest, &id_src->val);
  print_asm_template2(movzx);
}

make_EHelper(lea) {
  operand_write(id_dest, &id_src->addr);
```

```
  print_asm_template2(lea);
}
make_EHelper(movsb){
 int in=1;
  rtl_lr(&s0,R_ESI,4);
  rtl_lm(&s1,&s0,1);
  s0+=in;
  rtl_sr(R_ESI,&s0,4);
  rtl_lr(&s0,R_EDI,4);
  rtl_sm(&s0,&s1,1);
  s0+=in;
  rtl_sr(R_EDI,&s0,4);
  print_asm_template2(movsb);

}
```

3.3.7 logic.c

```
#include "cpu/exec.h"
#include "cc.h"

make_EHelper(test)
{
  // TODO();
  rtl_and(&s1, &id_dest->val, &id_src->val);
  s0 = 0;
  rtl_set_CF(&s0);
  rtl_set_OF(&s0);
  rtl_update_ZFSF(&s1, id_dest->width);
  print_asm_template2(test);
}

make_EHelper(and)
{
  rtl_and(&s1, &id_dest->val, &id_src->val);
  s0 = 0;
  rtl_set_OF(&s0);
  rtl_set_CF(&s0);
  rtl_update_ZFSF(&s1, id_dest->width);
  operand_write(id_dest, &s1);
  print_asm_template2(and);
}

make_EHelper(xor)
{
  // TODO();
  rtl_xor(&s1, &id_dest->val, &id_src->val);
  s0 = 0;
  rtl_set_OF(&s0);
  rtl_set_CF(&s0);
  rtl_update_ZFSF(&s1, id_dest->width);
```

```
  operand_write(id_dest, &s1);
  print_asm_template2(xor);
}

make_EHelper(or)
{
  // TODO();
  rtl_or(&s1, &id_dest->val, &id_src->val);
  s0 = 0;
  rtl_set_OF(&s0);
  rtl_set_CF(&s0);
  operand_write(id_dest, &s1);
  rtl_update_ZFSF(&s1, id_dest->width);

  print_asm_template2(or);
}

make_EHelper(sar)
{
  // TODO();
  // unnecessary to update CF and OF in NEMU
  rtl_sar(&s0, &id_dest->val, &id_src->val);
  operand_write(id_dest, &s0);
  rtl_update_ZFSF(&s0, id_dest->width);
  print_asm_template2(sar);
}

make_EHelper(shl)
{
  // TODO();
  // unnecessary to update CF and OF in NEMU
  rtl_shl(&s0, &id_dest->val, &id_src->val);
  operand_write(id_dest, &s0);
  rtl_update_ZFSF(&s0, id_dest->width);
  print_asm_template2(shl);
}

make_EHelper(shr)
{
  // TODO();
  // unnecessary to update CF and OF in NEMU
  rtl_shr(&s0, &id_dest->val, &id_src->val);
  operand_write(id_dest, &s0);
  rtl_update_ZFSF(&s0, id_dest->width);
  print_asm_template2(shr);
}

make_EHelper(setcc)
{
  uint32_t cc = decinfo.opcode & 0xf;

  rtl_setcc(&s0, cc);
```

```
  operand_write(id_dest, &s0);

  print_asm("set%s %s", get_cc_name(cc), id_dest->str);
}

make_EHelper(not)
{
  // TODO();
  rtl_not(&id_dest->val, &id_dest->val);
  operand_write(id_dest, &id_dest->val);

  print_asm_template1(not);
}
make_EHelper(rol)
{
  rtl_li(&s0,id_dest->val);
  rtl_shri(&s1,&s0,id_dest->width*8-id_src->val);
  rtl_shli(&s0,&s0,id_src->val);
  rtl_or(&s0,&s1,&s0);
  operand_write(id_dest,&s0);
  rtl_update_ZFSF(&s0,id_dest->width);

  print_asm_template1(rol);
}
```

3.3.8 rtl.h

```
#ifndef __X86_RTL_H__
#define __X86_RTL_H__

#include "rtl/rtl.h"


/* RTL pseudo instructions */

static inline void rtl_lr(rtlreg_t *dest, int r, int width)
{
  switch (width)
  {
  case 4:
    rtl_mv(dest, &reg_l(r));
    return;
  case 1:
    rtl_host_lm(dest, &reg_b(r), 1);
    return;
  case 2:
    rtl_host_lm(dest, &reg_w(r), 2);
    return;
  default:
    assert(0);
  }
```

```
}

static inline void rtl_sr(int r, const rtlreg_t *src1, int width)
{
  switch (width)
  {
  case 4:
    rtl_mv(&reg_l(r), src1);
    return;
  case 1:
    rtl_host_sm(&reg_b(r), src1, 1);
    return;
  case 2:
    rtl_host_sm(&reg_w(r), src1, 2);
    return;
  default:
    assert(0);
  }
}

static inline void rtl_push(const rtlreg_t *src1)
{
  // esp <- esp - 4
  cpu.esp-=4;
  // M[esp] <- src1
  rtl_sm(&cpu.esp, src1, 4);

}

static inline void rtl_pop(rtlreg_t *dest)
{
  // dest <- M[esp]
  rtl_lm(dest, &cpu.esp, 4);
  // esp <- esp + 4
  rtl_addi(&cpu.esp, &cpu.esp, 4);
  // TODO();
}

// @res: 传入的运算结果
// overflow 是在有符号数运算时产生的，当两个操作数符号不同且第一个操作数与
结果
// 符号不同时产生溢出
static inline void rtl_is_sub_overflow(rtlreg_t *dest,
                                       const rtlreg_t *res, const
rtlreg_t *src1, const rtlreg_t *src2, int width)
{
   // dest <- is_overflow(src1 - src2)
  // TODO();
  //src1-src2=res
  //减法有溢出是src1与src2的符号不一致，且res的符号与src1相反
    int result = 0;
  if (width == 1) {
```

```
    int8_t src1_ = *((int8_t *)src1);
    int8_t src2_ = *((int8_t *)src2);
    int8_t res_ = *((int8_t *)res);
    result = (((src1_ ^ src2_) >> 7) && ((res_ ^ src1_) >> 7)) ?
1 : 0;
  } else if (width == 2) {
    int16_t src1_ = *((int16_t *)src1);
    int16_t src2_ = *((int16_t *)src2);
    int16_t res_ = *((int16_t *)res);
    result = (((src1_ ^ src2_) >> 15) && ((res_ ^ src1_) >> 15))
? 1 : 0;
  } else if (width == 4) {
    int32_t src1_ = *((int32_t *)src1);
    int32_t src2_ = *((int32_t *)src2);
    int32_t res_ = *((int32_t *)res);
    result = (((src1_ ^ src2_) >> 31) && ((res_ ^ src1_) >> 31))
? 1 : 0;
  } else {
    assert(0);
  }
  rtl_li(dest, result);
}

// carry 是在无符号数运算时产生的
static inline void rtl_is_sub_carry(rtlreg_t *dest,
                                    const rtlreg_t *res, const
rtlreg_t *src1)
{
  // dest <- is_carry(src1 - src2)
  //借位直接比大小即可，src1比res小，则肯定借位
  rtl_setrelop(RELOP_LTU,dest,src1,res);

}

// overflow 是在有符号数运算时产生的，当两个操作数符号相同且第一个操作数与
结果
// 符号不同时产生溢出
static inline void rtl_is_add_overflow(rtlreg_t *dest,
                                       const rtlreg_t *res, const
rtlreg_t *src1, const rtlreg_t *src2, int width)
{
  // dest <- is_overflow(src1 + src2)
    //加法溢出与减法相反，src1与src2同号溢出
  int result = 0;
  if (width == 1)
  {
    int8_t src1_ = *((int8_t *)src1);
    int8_t src2_ = *((int8_t *)src2);
    int8_t res_ = *((int8_t *)res);
    result = (!((src1_ ^ src2_) >> 7) && ((res_ ^ src1_) >> 7)) ?
1 : 0;
  }
```

```
  else if (width == 2)
  {
    int16_t src1_ = *((int16_t *)src1);
    int16_t src2_ = *((int16_t *)src2);
    int16_t res_ = *((int16_t *)res);
    result = (!((src1_ ^ src2_) >> 15) && ((res_ ^ src1_) >> 15))
? 1 : 0;
  }
  else if (width == 4)
  {
    int32_t src1_ = *((int32_t *)src1);
    int32_t src2_ = *((int32_t *)src2);
    int32_t res_ = *((int32_t *)res);
    result = (!((src1_ ^ src2_) >> 31) && ((res_ ^ src1_) >> 31))
? 1 : 0;
  }
  else
  {
    assert(0);
  }
  rtl_li(dest, result);
}

static inline void rtl_is_add_carry(rtlreg_t *dest,
                                    const rtlreg_t *res, const
rtlreg_t *src1)
{
  // dest <- is_carry(src1 + src2)
  rtl_setrelop(RELOP_LTU, dest, res, src1);

}

#define make_rtl_setget_eflags(f)                                \
  static inline void concat(rtl_set_, f)(const rtlreg_t *src) \
  {                                                              \
    cpu.eflags.f = (*src) ? 1 : 0;                               \
  }                                                              \
  static inline void concat(rtl_get_, f)(rtlreg_t * dest)     \
  {                                                              \
    *dest = cpu.eflags.f;                                        \
  }

make_rtl_setget_eflags(CF)
    make_rtl_setget_eflags(OF)
        make_rtl_setget_eflags(ZF)
            make_rtl_setget_eflags(SF)

                static inline void rtl_update_ZF(const rtlreg_t
*result, int width)
{
  // eflags.ZF <- is_zero(result[width * 8 - 1 .. 0])
  switch (width)
```

```
  {
  case 1:
    cpu.eflags.ZF = *((int8_t *)result) ? 0 : 1;
    break;
  case 2:
    cpu.eflags.ZF = *((int16_t *)result) ? 0 : 1;
    break;
  case 4:
    cpu.eflags.ZF = *((int32_t *)result) ? 0 : 1;
    break;
  default:
    assert(0);
  }
}

static inline void rtl_update_SF(const rtlreg_t *result, int
width)
{
  // eflags.SF <- is_sign(result[width * 8 - 1 .. 0])
  switch (width)
  {
  case 1:
    cpu.eflags.SF = ((*((int8_t *)result)) >> 7) ? 1 : 0;
    break;
  case 2:
    cpu.eflags.SF = ((*((int16_t *)result)) >> 15) ? 1 : 0;
    break;
  case 4:
    cpu.eflags.SF = ((*((int32_t *)result)) >> 31) ? 1 : 0;
    break;
  default:
    assert(0);
  }
}

static inline void rtl_update_ZFSF(const rtlreg_t *result, int
width)
{
  rtl_update_ZF(result, width);
  rtl_update_SF(result, width);
}

#endif
```

3.4 实现常用的库函数

3.4.1 `string.c`

为了让`string`能够正常运行，需要实现`nexus-am/am/libs/klib/src/string.c`中的库函数，这些库函数都是字符串处理的函数，上C++课的时候也实现过。

```c
#include "klib.h"

#if !defined(__ISA_NATIVE__) || defined(__NATIVE_USE_KLIB__)

size_t strlen(const char *s)
{
  size_t len = 0;
  while ((*s++) != '\0')
  {
    len++;
  }
  return len;
}

char *strcpy(char *dst, const char *src)
{
  return strncpy(dst, src, strlen(src));
}

char *strncpy(char *dst, const char *src, size_t n)
{
  if (n > strlen(src))
  {
    n = strlen(src);
  }
  char *res = dst;
  while (n && (*dst++ = *src++))
  {
    n--;
  }
  return res;
}

char *strcat(char *dst, const char *src)
{
  char *temp = dst;
  while (*dst)
  {
    dst++;
  }
  while ((*dst++ = *src++) != 0)
    ;
  return temp;
}

int strcmp(const char *s1, const char *s2)
{
  int t = 0;
  while ((t = (*s1 - *s2)) == 0 && *s1 && *s2)
  {
    s1++;
    s2++;
```

```
  }
  return t;
}

int strncmp(const char *s1, const char *s2, size_t n)
{
  assert(s1 != NULL && s2 != NULL);
  while (n--)
  {
    if (*s1 == 0 || *s1 != *s2)
    {
      return *s1 - *s2;
    }
    s1++;
    s2++;
  }
  return 0;
}

void *memset(void *v, int c, size_t n)
{
  //const unsigned char temp=c;
  //unsigned char *s;
  //for(s=v;n>0;++s,--n){
  //  *s=temp;
  //}
  for (size_t i = 0; i < n; i++)
  {
    ((int8_t *)v)[i] = c;
  }
  return v;
}

void *memcpy(void *out, const void *in, size_t n)
{
  char *pout = (char *)(out);
  const char *pin = (const char *)(in);
  if (pout > pin && pout < pin + n)
  {
    pout = pout + n - 1;
    pin = pin + n - 1;
    while (n--)
    {
      *pout-- = *pin--;
    }
  }
  else
  {
    while (n--)
    {
      *pout++ = *pin++;
    }
```

```
  }
  return pout;
}

int memcmp(const void *s1, const void *s2, size_t n)
{
  assert(s1 || s2);
  const unsigned char *s11, *s22;
  int count = 0;
  for (s11 = s1, s22 = s2; n > 0; ++s11, ++s22, n--)
  {
    if ((count = *s11 - *s22) != 0)
    {
      break;
    }
  }
  return count;
}

#endif
```

3.4.2 stdio.c

这个库里面都是打印的函数，难点在于对于参数的处理

```
#include "klib.h"
#include <stdarg.h>

#if !defined(__ISA_NATIVE__) || defined(__NATIVE_USE_KLIB__)

int printf(const char *fmt, ...) {
  va_list args;
  va_start(args,fmt);
  char out[1000];
  int out_len=vsprintf(out,fmt,args);
  va_end(args);
  for(int i=0;i<out_len;i++){
    _putc(out[i]);
  }
  return 0;
}

int vsprintf(char *out, const char *fmt, va_list ap) {
  char *outp;int out_len=0;
  int width;
  int flags;
  char nums[1000];
  char *ss=nums;

  for(outp=out;*fmt;fmt++){
    if(*fmt!='%'){
```

```
      *outp++=*fmt;
      out_len++;
      continue;
    }
    int temp=1;
    flags=0;
    while(temp==1){
      fmt++;
      switch(*fmt){
        case '0':flags|=1;break;
        case '+':flags|=4;break;
        case ' ':flags|=8;break;
        case '-':flags|=16;break;
        case '#':flags|=32;break;
        default:temp=0;
      }
    }
    width=0;
    if(*fmt>='0'&&*fmt<='9'){
      for(;*fmt>='0'&&*fmt<='9';fmt++){
        width=width*10+*fmt-'0';
      }
    }
    else if(*fmt=='*'){
      fmt++;
      width=va_arg(ap,int);
      if(width<0){
        width=-width;
        flags|=16;
      }
    }
    switch(*fmt){
      case 'd':break;
      case 's':{
        char *s=va_arg(ap,char *);
        int len_s=strlen(s);
        if(!(flags&16)){
          while(len_s<width--){
            *outp++=' ';out_len++;
          }
        }
        for(int i=0;i<len_s;i++){
          *outp++=*s++;out_len++;
        }
        while(len_s<width--){
          *outp++=' ';out_len++;
        }
        continue;
      }
    }

    int num=va_arg(ap,int);
```

```
    int count=0;
    if(num==0){
      *ss++='0';
      count+=1;
    }
    else{
      if(num<0){
        *outp++='-';out_len++;
        num=-num;
      }
      while(num){
        *ss++=num%10+'0';
        num=num/10;
        count+=1;
      }
    }
    if(count<width){
      num=width-count;
      if(flags&1){
        while(num--){
          *outp++='0';out_len++;
        }
      }
      else if(flags&8){
        while(num--){
          *outp++=' ';out_len++;
        }
      }
    }
    while(count--){
      *outp++=*--ss;out_len++;
    }
  }
  *outp='\0';
  return out_len;
}

int sprintf(char *out, const char *fmt, ...) {
  va_list args;
  va_start(args,fmt);
  int out_len=vsprintf(out,fmt,args);
  va_end(args);
  return out_len;
}

int snprintf(char *out, size_t n, const char *fmt, ...) {
  return 0;
}

#endif
```

3.5 一键回归测试

在`nemu/`目录下运行

```
bash runall.sh ISA=$ISA
```

> 问题二：你觉得该如何捕捉死循环
>
> 答：对程序设置最大运行时间，如果超过该时间则判断程序进入死循环

3.6 实验结果

```
testcases compile OK
[  add-longlong] PASS!
[           add] PASS!
[           bit] PASS!
[   bubble-sort] PASS!
[           div] PASS!
[         dummy] PASS!
[          fact] PASS!
[           fib] PASS!
[      goldbach] PASS!
[     hello-str] PASS!
[       if-else] PASS!
[     leap-year] PASS!
[    load-store] PASS!
[    matrix-mul] PASS!
[           max] PASS!
[          min3] PASS!
[         mov-c] PASS!
[         movsx] PASS!
[  mul-longlong] PASS!
[        pascal] PASS!
[         prime] PASS!
[    quick-sort] PASS!
[     recursion] PASS!
[   select-sort] PASS!
[         shift] PASS!
[   shuixianhua] PASS!
[        string] PASS!
[  sub-longlong] PASS!
[           sum] PASS!
[        switch] PASS!
[ to-lower-case] PASS!
[       unalign] PASS!
[        wanshu] PASS!
kelee@kelee-virtual-machine:~/ics2019/nemu$
```

4 阶段三

> 问题三：如果代码中`p`指向的地址最终被映射到一个设备寄存器，去掉`volatile`可能会带来什么问题?
>
> 答：变量如果加了 `volatile` 修饰，则会从内存重新装载内容，而不是直接从寄存器拷贝内容，去掉`volatile`会导致错误发生

4.1 串口

实现串口要实现`in` `out`指令

1. 在`all-instr.h`中增加

```
//system.c
make_EHelper(in);
make_EHelper(out);
```

2. 修改`opcode_table`

```
        /* 0xe4 */ IDEXW(in_I2a, in, 1), IDEX(in_I2a, in),
IDEXW(out_a2I,out,1), IDEX(out_a2I,out),
        /* 0xec */ IDEXW(in_dx2a,in,1),
IDEX(in_dx2a,in),IDEXW(out_a2dx,out,1), IDEX(out_a2dx,out),
```

3. 修改`system.c`

   根据想写入和读取的寄存器的宽度进行操作不同的寄存器

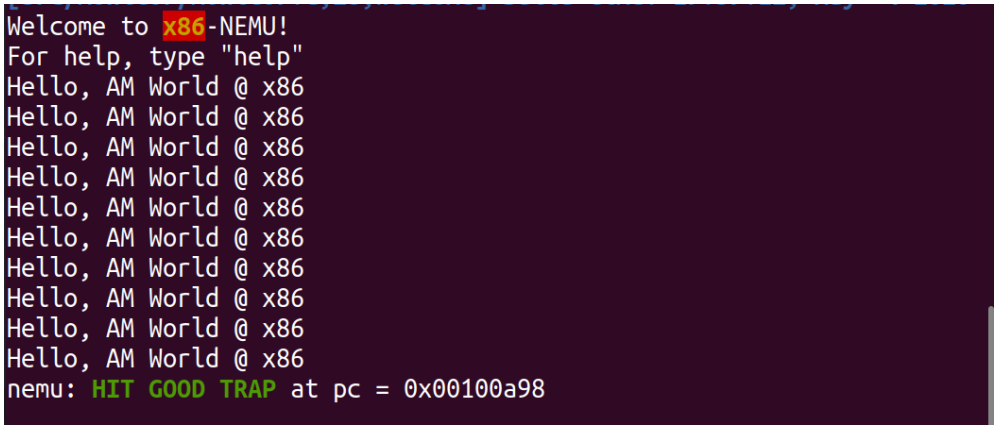
```
make_EHelper(in)
{
  // TODO();
  // difftest_skip_ref();
  switch (id_src->width)
  {
  case 1:
    s0 = pio_read_b(id_src->val);
    break;
  case 2:
    s0 = pio_read_w(id_src->val);
    break;
  case 4:
    pio_read_l(id_src->val);
    break;
  default:
    assert(0);
    break;
  }
  operand_write(id_dest, &s0);
  difftest_skip_ref();

  print_asm_template2(in);
}

make_EHelper(out)
{
  // TODO();
  switch (id_src->width)
  {
  case 1:
    pio_write_b(id_dest->val, id_src->val);
    break;
  case 2:
```

```
    pio_write_w(id_dest->val, id_src->val);
    break;
  case 4:
    pio_write_l(id_dest->val, id_src->val);
    break;
  default:
    assert(0);
  }
  difftest_skip_ref();

  print_asm_template2(out);
}
```

4. 在`nexus-am/tests/amtest/`目录下键入

```
make mainargs=h run
```

5. 实验结果



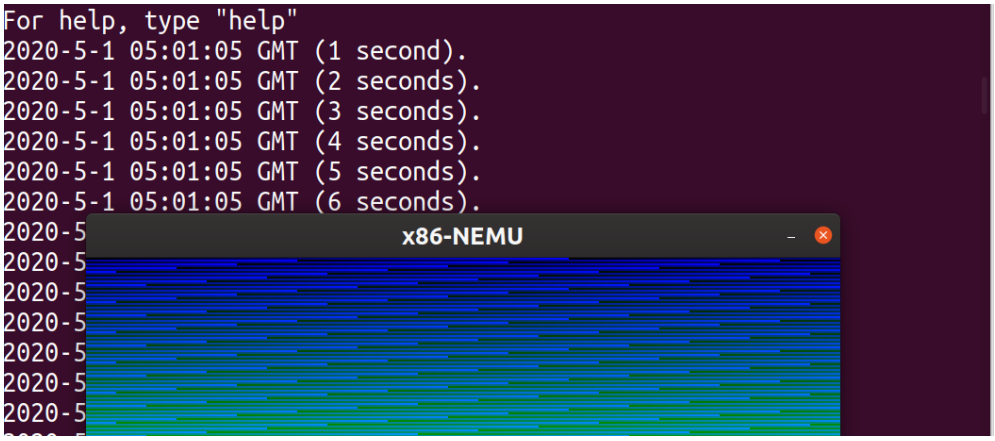
4.2 时钟

1. 在`nexus-am/am/src/nemu-common/nemu-timer.c`中实现`_DEVREG_TIMER_UPTIME`的功能，就是读取端口地址减去开机时间。

```
static _DEV_TIMER_UPTIME_t boot_time;
size_t __am_timer_read(uintptr_t reg, void *buf, size_t size)
{
  switch (reg) {
    case _DEVREG_TIMER_UPTIME: {
      _DEV_TIMER_UPTIME_t *uptime = (_DEV_TIMER_UPTIME_t
*)buf;
      uptime->hi = 0;
      uptime->lo = inl(RTC_ADDR)-boot_time.lo;
      return sizeof(_DEV_TIMER_UPTIME_t);
    }
    case _DEVREG_TIMER_DATE: {
      _DEV_TIMER_DATE_t *rtc = (_DEV_TIMER_DATE_t *)buf;
      rtc->second = 5;
      rtc->minute = 1;
      rtc->hour   = 5;
```

```
      rtc->day    = 1;
      rtc->month  = 5;
      rtc->year   = 2020;
      return sizeof(_DEV_TIMER_DATE_t);
    }
  }
  return 0;
}

void __am_timer_init() {
  boot_time.hi=0;
  boot_time.lo=inl(RTC_ADDR);
}
```

2. 实验结果是失败的，间隔时间实际不为1s

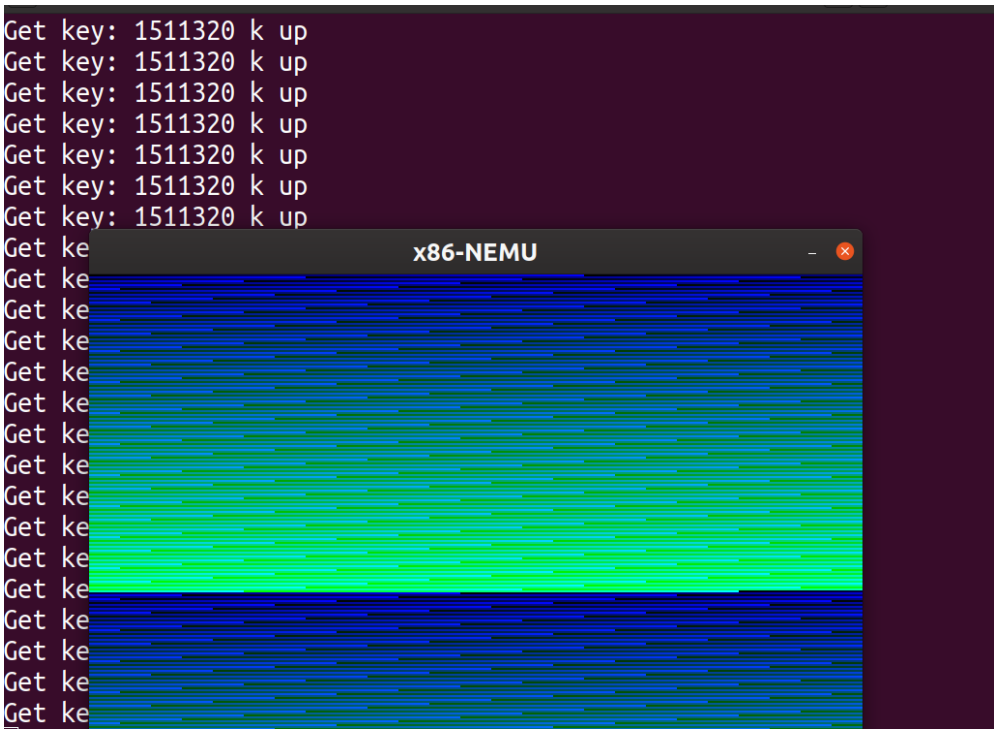


4.3 键盘

1. 在`nexus-am/am/src/nemu-common/nemu-input.c`中实现`_DEVREG_INPUT_KBD`的功能

```
#define KBD_PORT 0x60
size_t __am_input_read(uintptr_t reg, void *buf, size_t size)
{
  switch (reg) {
    case _DEVREG_INPUT_KBD: {
      _DEV_INPUT_KBD_t *kbd = (_DEV_INPUT_KBD_t *)buf;
      kbd->keydown = 0;
      kbd->keycode = inl(KBD_PORT);
      return sizeof(_DEV_INPUT_KBD_t);
    }
  }
  return 0;
}
```

2. 实验结果为失败的一直都是显示k



> 问题四：如何检测多个键被同时按下?
>
> 答：当检测到一个键被按下的时候，去检测此时其他是否有按键被按下

4.4 VGA

> 问题五：在一些90年代的游戏中, 很多渐出渐入效果都是通过调色板实现的, 聪明的你知道其中的玄机吗?
>
> 答：将颜色模拟成类似透明的白色盖在其他颜色上面，模拟出渐变的颜色

1. 向`__am_vga_init()`中添加如下测试代码:

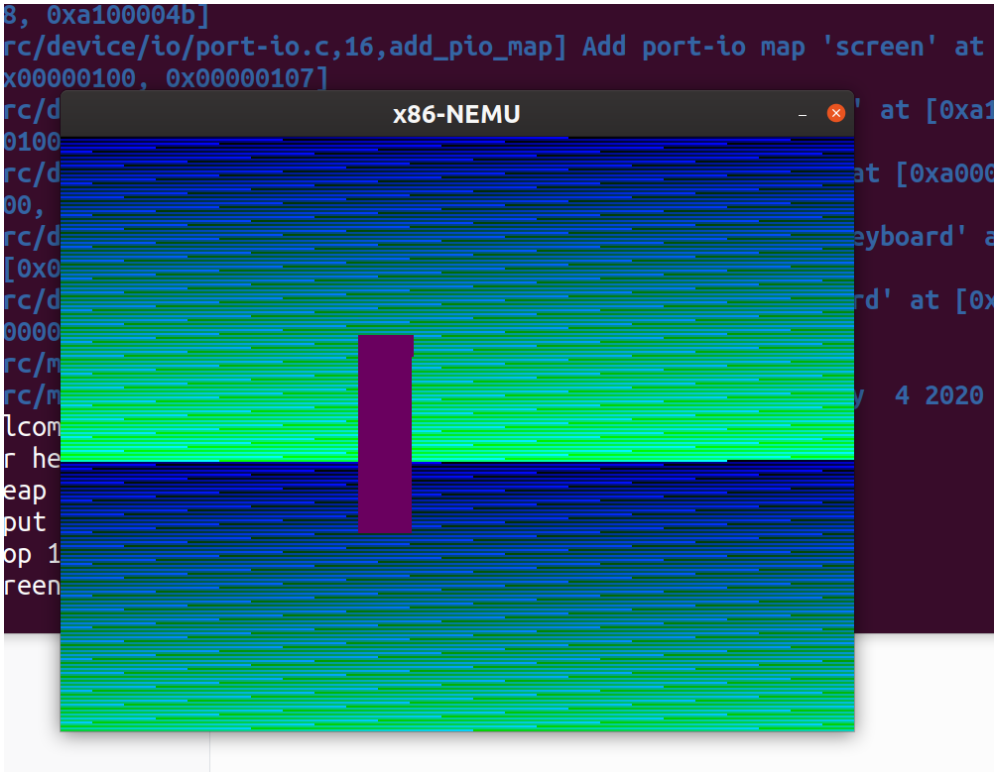
```
--- nexus-am/am/src/nemu-common/nemu-video.c
+++ nexus-am/am/src/nemu-common/nemu-video.c
@@ -31,2 +31,7 @@
 void __am_vga_init() {
+  int i;
+  int size = screen_width() * screen_height();
+  uint32_t *fb = (uint32_t *)(uintptr_t)FB_ADDR;
+  for (i = 0; i < size; i ++) fb[i] = i;
+  draw_sync();
 }
```

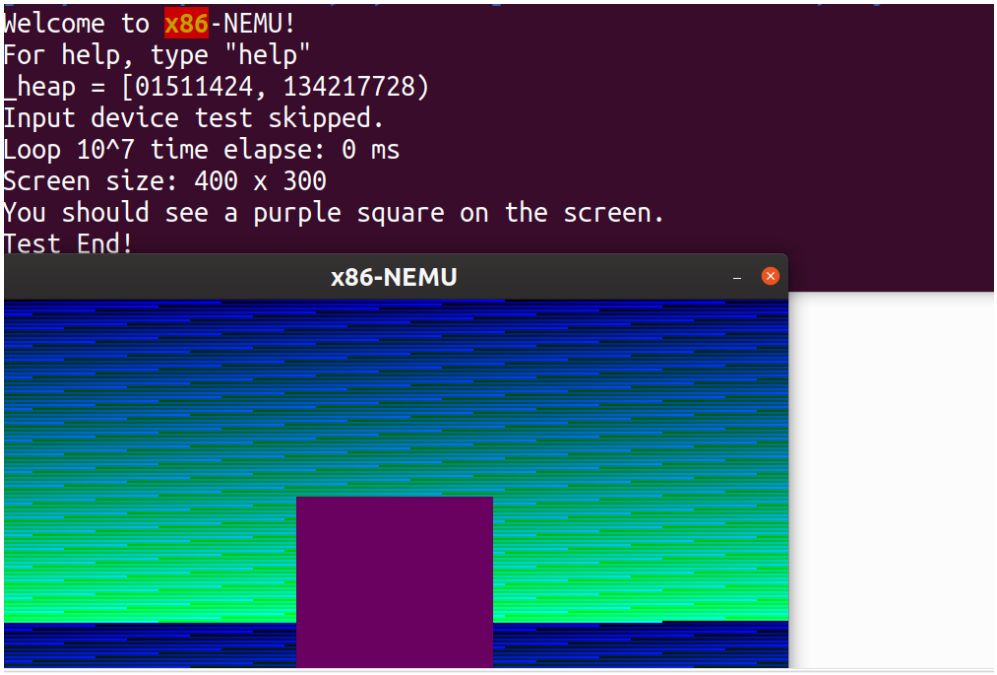
2. 然后在`$ISA-nemu`中运行`amtest`中的`display test`测试. 如果你的实现正确, 你会看到新窗口中输出了全屏的颜色信息.
3. 实现`_DEVREG_VIDEO_FBCTL`的功能

```
{
    _DEV_VIDEO_FBCTL_t *ctl = (_DEV_VIDEO_FBCTL_t *)buf;
    uint32_t *fb = (uint32_t *)(uintptr_t)FB_ADDR;

    int x = ctl->x, y = ctl->y, w = ctl->w, h = ctl->h;
```

```
    uint32_t *pixels = ctl->pixels;
    int cp_bytes = sizeof(uint32_t) * (w < W - x ? w : W -
x);
    for (int j = 0; j < h && y + j < H; j++)
    {
      memcpy(&fb[(y + j) * W + x], pixels, cp_bytes);
      pixels += w;
    }

    if (ctl->sync)
    {
      outl(SYNC_ADDR, 0);
    }
    return size;
  }
  }
```

4. 实验结果





5 必答题

5.1 编译与链接

问题六：去掉`static`，去掉`inline`或去掉两者，然后重新进行编译，你会发现错误，请分别解释为什么会发生这些错误？你有办法证明你的想法么

答：当函数被声明`static`后，它只在定义它的源文件内有效，其他源文件无法访问，所以用来解决不同文件函数重名问题，如果去掉进行编译的话，若不同文件有相同函数名则会报错，证明可以将不同文件中的函数名里添加文件名进行区分，若不报错则想法正确。`inline`修饰的函数变为内联函数,同时和`static`类似，只有本地文件可见，允许多个文件内重复定义相同名的函数，错误与`static`类似，可能会报重复定义的错误，证明可以将不同文件中的函数名里添加文件名进行区分，若不报错则想法正确

问题七：在`nemu/include/common.h`中添加一行`volatile static int dummy;`然后重新编译NEMU. 请问重新编译后的NEMU含有多少个`dummy`变量的实体? 你是如何得到这个结果的?

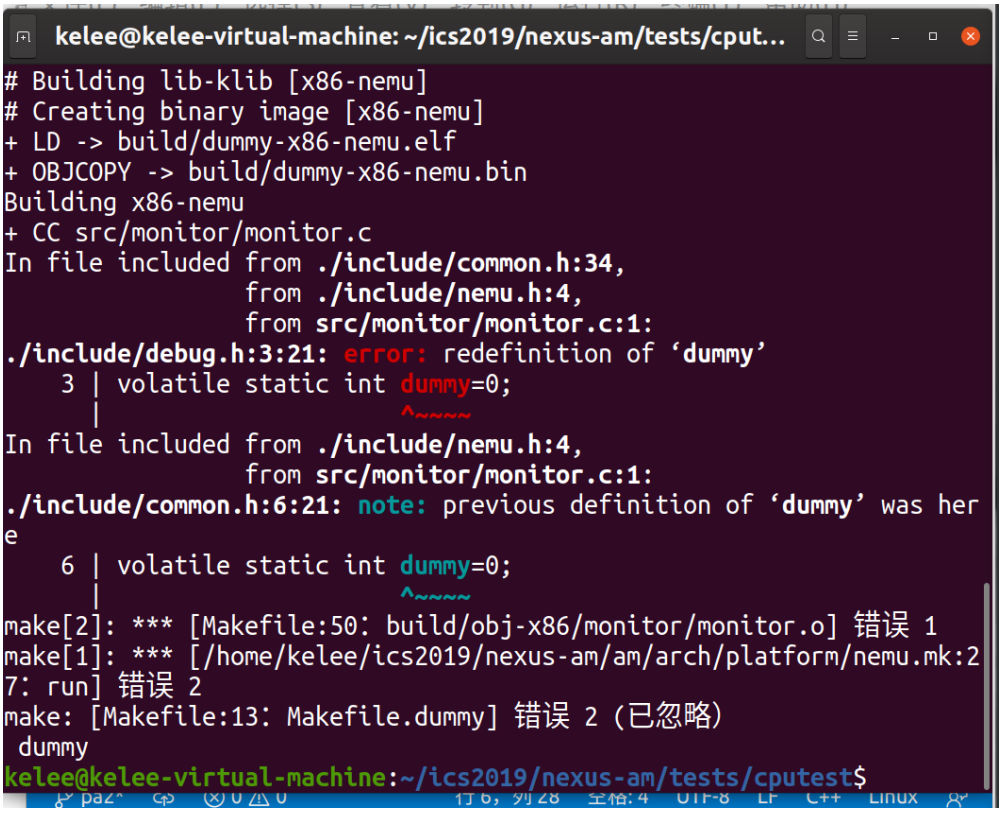
答：有1个。因为在这里用`volatile`定义了一个`dummy`

问题八：添加上题中的代码后, 再在`nemu/include/debug.h`中添加一行`volatile static int dummy;`然后重新编译NEMU. 请问此时的NEMU含有多少个`dummy`变量的实体? 与上题中`dummy`变量实体数目进行比较, 并解释本题的结果。

答：有一个，因为这里虽然在两个文件中定义了，但这两个都使用了 `static` 修饰符，而且 `debug.h` 包含了 `common.h`，这样就相当于在 `debug.h`中定义了两个包含`static`修 饰符的`dummy`，而在同一个文件中使用 `static`修饰符同一个变量可以写多次定义，但 只定义了一次。

问题九：修改添加的代码, 为两处`dummy`变量进行初始化:`volatile static int dummy = 0;`然后重新编译NEMU. 你发现了什么问题? 为什么之前没有出现这样的问题? (回答完本题后可以删除添加的代码.)

答：会出现错误，因为在 `debug.h` 文件中使用 `static` 定义了两次 `dummy` 且对这两个 都进行了赋值。



5.2 了解`Makefile`

问题十：请描述你在`nemu/`目录下敲入`make`后，`make`程序如何组织.c和.h文件，最终生成可执行文件`nemu/build/$ISA-nemu`。(这个问题包括两个方面:`Makefile`的工作方式和编译链接的过程.)

答：`Makefile`文件主要由一个个规则构成，这些规则包含了目标文件、需要的源文件以及生成目标文件所需要的命令。在敲入`make`命令之后，由于在`Makefile`中设置了`DEFAULT_GOAL`为`app`，所以会去运行`app`规则，而`app`需要的文件为`BINARY`，这时又 会去运行`BINARY`所对应的规则，就这样一直进行下去直到可以运行一条规则，之后 再递归上来最终运行`app`规则。 在`Makefile`中，`.c`和`.h`文件主要是由以下三行组织：

```
SRCS = $(shell find src/ -name "*.c" | grep -v "isa")
SRCS += $(shell find src/isa/$(ISA) -name "*.c")
INC_DIR += ./include ./src/isa/$(ISA)/include
```

其中前两行是通过 shell 命令来寻找该文件夹下包含的所有.c 文件。

6 问题

1. 在运行dummy的时候报错：

   解决方法，更改`nexus-am/Makefile.compile`里的CFLAGS中去掉`-Werror`即可

2. `jle`指令错误，difftest后除了pc其他寄存器都是正确的

   解决方法：检查出是`rtl.h`中的`rtl_sub_overflow`与`rtl_sub_carry`出错

3. 时钟实现时间隔时间不为1s，键盘不能读取键盘输入，在`native`中能正确运行

   未解决