南京航空航天大学《计算机组成原理Ⅱ课程设计》报告

- 姓名：傅锦龙
- 班级：1621301
- 学号：162130117
- 报告阶段：PA2.2&2.3
- 完成日期：2023.6.11
- 本次实验，我完成了所有内容。

目录

- 83 c4(add)
- 39（cmp)
- 0f 94(setz)
- 0f b6(movzx)
- ff 45(inc)
- add-longlong.c
  - e9(jmp)
  - 0f 86(jcc)
  - 11（adc)
  - 5b(pop)
  - 33(xor)
  - 09(or)
  - 85(test)
- bits.c
  - 6a(push)
  - c1 f8(sar)
  - d3 e2(shl)
  - 22(and)
  - f7 d0(not)
- bubble-sort.c
  - 2b(sub)
  - 40(inc)
- fact.c
  - 48（dec)
  - 0f af(imul)
- goldbach.c
  - 99(cltd)
  - f7 7d(idiv)
- hello-str.c
  - 0f be(movsx)
  - ff 24(jmp_rm)
  - f7 75(div)
- load-store.c
  - 0f b7(movzx)
  - c1 eb(shr)
- sub-longlong.c
  - 1b(sbb)

通过一键回归测试

IN/OUT 指令

- 实现 IN , OUT 两条指令
- 成功运行 nexus-am/apps/hello 程序

实现时钟设备

- 实现 _uptime() 函数;
- 成功运行 timetest 程序

运行跑分项目

- 成功运行 dhrystone , coremark , microbench 三个跑分项目;
  - dhrystone
  - coremark
  - microbench

实现键盘设备

- 实现 _read_key() 函数
- 成功运行 keytest 程序

添加内存映射 I/O

- 在 paddr_read() 和 paddr_write() 中添加内存映射 I/O 判断
- 成功运行 videotest 程序

运行打字小游戏

捕捉死循环

遇到的问题及解决办法

思考题

什么是 API

API是Application Programming Interface的缩写，指的是应用程序编程接口。API是一组定义了软件组件之间如何互相通信的规则、协议和工具集合。通过API，不同的软件组件可以相互交互，共同完成某个任务或提供某个服务。
API通常包括一组预定义的函数、数据结构、协议和工具，用于在不同的软件组件之间传递信息和执行操作。例如，操作系统提供了一组API，用于访问文件系统、网络、设备驱动程序等系统资源。

AM 属于硬件还是软件?

AM（Abstract Machine）是一个抽象的计算机模型，既不属于硬件也不属于软件 。
操作系统是一种特定的软件，它运行在具体的计算机硬件之上，提供了一系列的服务和接口，用于管理计算机系统的资源和执行应用程序。操作系统通常需要直接访问硬件资源，例如CPU、内存、磁盘等，以实现其功能。
相比之下，AM更加通用化和抽象化，它不依赖于具体的硬件实现，而是提供了一组通用的接口和规范，用于模拟计算机系统的行为。AM可以用于实现各种不同的计算机模拟器，例如NEMU就是基于AM实现的x86模拟器。因此，AM更加灵活和可移植，可以用于模拟不同的计算机系统和架构，而不需要针对每种硬件实现编写不同的模拟器。

堆和栈在哪里?

堆和栈是程序运行时动态分配的内存区域，它们的大小和内容在编译时无法确定，由操作系统根据程序的需要进行分配和管理，因此不能将它们的内容放入可执行文件中。
在程序运行时，操作系统会为程序分配一块内存空间，用于存储程序的代码、数据和静态变量等。在程序运行过程中，如果需要分配更多的内存空间，例如用于存储动态分配的变量或对象，程序就会向操作系统请求更多的内存空间。操作系统会根据程序的请求分配一块新的内存空间，并将其分配给程序使用。这个内存空间就是堆。
栈是用于存储函数调用和局部变量的内存区域。在程序运行时，每当调用一个函数时，程序就会将函数的返回地址、参数和局部变量等信息压入栈中。当函数返回时，程序会从栈中弹出这些信息，恢复函数调用前的状态。
AM提供了一组通用的接口和规范，用于模拟计算机系统的行为。在AM的模型中，堆和栈也是抽象的概念，它们的实现可以根据具体的需求进行选择和实现。因此，我们可以借鉴AM的思想，将堆和栈的实现与具体的硬件和操作系统分离，使得程序更加灵活和可移植。

回忆运行过程

- ARCH=x86-nemu：让AM上面的项目编译到 x86-nemu 的 AM 中
- make ALL=dummy run：用dummy作为测试样例，调用 nexus-am/am/arch/x86-nemu/img/run 来启动 NEMU并载入 dummy 这个用户程序进入运行

神奇的eflags（2）

```
+-----+-----+-----------------------------------+
|  SF |  OF |               实例                |
+-----+-----+-----------------------------------+
|  0  |  0  |               2 - 1               |
+-----+-----+-----------------------------------+
|  0  |  1  |         0x80000000-0x00000001     |
+-----+-----+-----------------------------------+
|  1  |  0  |         0x80000001-0x00000001     |
+-----+-----+-----------------------------------+
|  1  |  1  |         0x7fffffff-0xffffffff     |
+-----+-----+-----------------------------------+
```

这是巧合吗?

- op2>op1,且是无符号数，触发above,也就是ja指令跳转
- op2<op1,且是无符号数，触发below，也就是jb指令跳转
- op2>op1,且是有符号数，触发greater，也就是jg指令跳转
- op2>op1,且是有符号数，触发less，也就是jl指令跳转

NEMU的本质

```
lable1:
    x = x - 1
    a = a + 1
    jne x， lable1

lable2:
    y = y - 1
    a = a + 1
    jne y， lable2
```

还需要输入和输出功能，还需要通过图形界面进行交互

设备是如何工作的?

采用应答式的通信，CPU通过一系列的发送端口向相应的设备发出指令信号，设备接收之后执行自己的工作，执行结束或者遇到错误发送反馈信号给CPU的接收端口。

CPU 需要知道设备是如何工作的吗?

CPU需要知道设备在接收到命令字后所做的事情，以便能够正确地处理设备返回的数据或状态。具体来说，设备在接收到命令字后，需要进行以下操作：

1. 解码命令字：设备需要解码命令字，以便识别CPU要求设备执行的操作。
2. 执行操作：设备根据命令字中的操作码和操作数，执行相应的操作，例如读写数据、启动传感器等。
3. 返回数据或状态：如果设备需要返回数据或状态，它需要将数据或状态写入到数据输出口或状态输出口中，以便CPU能够读取。
4. 发送中断信号：如果设备需要向CPU发送中断信号，它需要将中断信号写入到中断输出口中，以便CPU能够及时处理。

什么是驱动?

驱动（Driver）是一种特殊的软件程序，它用于控制计算机硬件设备的操作。驱动程序通常由设备制造商提供，用于与操作系统进行交互，以便操作系统能够正确地识别和使用硬件设备。
与一般的应用程序或操作系统不同，驱动程序通常需要直接访问硬件设备的寄存器和内存等底层资源，以便能够控制硬件设备的操作。因此，驱动程序需要具有更高的权限和更严格的安全性要求，以避免对系统的稳定性和安全性造成影响。
另外，驱动程序通常需要与操作系统的内核进行交互，以便能够正确地控制硬件设备的操作。因此，驱动程序的编写需要具有更高的技术要求和更深入的系统知识，以便能够正确地实现与操作系统的交互和硬件设备的控制。
总之，驱动程序是一种特殊的软件程序，用于控制计算机硬件设备的操作。它与一般的应用程序或操作系统不同，需要具有更高的权限和更严格的安全性要求，以及更深入的系统知识和技术要求。

CPU 知道吗?

在执行内存映射I/O的指令时，CPU并不需要知道自己要访问的地址是真正处于内存中还是经过重定向的。这是因为内存映射I/O的编程模型和普通编程完全一样，程序员编程时可直接把I/O设备当作内存来访问，而CPU在执行指令时只需要按照指令中给定的地址进行访问即可。

再次理解volatile关键字

添加volatile关键字的反汇编

```
08048410 <fun>:
 8048410:   c6 05 00 80 04 08 00    movb   $0x0,0x8048000
 8048417:   89 f6                   mov    %esi,%esi
 8048419:   8d bc 27 00 00 00 00    lea    0x0(%edi,%eiz,1),%edi
 8048420:   0f b6 05 00 80 04 08    movzbl 0x8048000,%eax
 8048427:   3c ff                   cmp    $0xff,%al
 8048429:   75 f5                   jne    8048420 <fun+0x10>
 804842b:   c6 05 00 80 04 08 33    movb   $0x33,0x8048000
 8048432:   c6 05 00 80 04 08 34    movb   $0x34,0x8048000
 8048439:   c6 05 00 80 04 08 36    movb   $0x36,0x8048000
 8048440:   c3                      ret
 8048441:   66 90                   xchg   %ax,%ax
 8048443:   66 90                   xchg   %ax,%ax
 8048445:   66 90                   xchg   %ax,%ax
 8048447:   66 90                   xchg   %ax,%ax
 8048449:   66 90                   xchg   %ax,%ax
 804844b:   66 90                   xchg   %ax,%ax
 804844d:   66 90                   xchg   %ax,%ax
 804844f:   90                      nop
```

不添加volatile关键字的反汇编

```
080483e0 <fun>:
 80483e0:   c6 05 00 80 04 08 00    movb   $0x0,0x8048000
 80483e7:   eb fe                   jmp    80483e7 <fun+0x7>
 80483e9:   66 90                   xchg   %ax,%ax
 80483eb:   66 90                   xchg   %ax,%ax
 80483ed:   66 90                   xchg   %ax,%ax
 80483ef:   90                      nop
```

若不加volatile，则从反汇编中可看出少了很多指令，第3行会陷入死循环。
在上述代码中，使用了volatile关键字来声明指针p，这意味着编译器不会对指针p所指向的内存进行优化，以避免出现意外的行为。具体来说，指针p所指向的内存可能是一个设备寄存器，而设备寄存器的值可能会被外部设备修改，因此编译器不能对指针p所指向的内存进行优化，以避免出现意外的行为。

如果去掉volatile关键字，编译器可能会对指针p所指向的内存进行优化，例如将多次写入的值合并为一次写入，或者将多次读取的值合并为一次读取。这样可能会导致程序出现意外的行为，例如写入的值被合并后只写入了一次，或者读取的值被合并后只读取了一次，从而导致程序出现错误。
总之，volatile关键字的作用是避免编译器对相应代码进行优化，以避免出现意外的行为。如果代码中的地址最终被映射到一个设备寄存器，去掉volatile可能会导致编译器对指针所指向的内存进行优化，从而导致程序出现错误。

Hello World运行在哪?

不一样。计算机中的Hello World 程序运行在硬件层，而我们这个hello程序运行在AM层

如何检测多个按键同时被按下?

每个键盘对应一个键盘码，当这个键盘被按下去的时候，构成一个通路，发生按键事件，因为每个键盘的按键码是不同的，所以他们之间相互独立，互不影响，当我按下多个键盘的时候，计算机只需要找到对应达成通路的键盘位置来执行相应的操作就行了。

编译与链接 I

去掉static

没有报错

去掉inline

去掉inline关键字时，编译器会认为这些函数不能被内联展开，而是需要在调用处生成函数调用指令。由于这些函数可能只在某些特定的地方被调用，而在其他地方没有被使用，因此编译器会认为这些函数是“defined but not used”，即定义了但未被使用的函数。这会导致编译器在链接时出现警告或错误，提示这些函数未被使用，从而可能会影响程序的正确性和可靠性。
因此，如果我们去掉inline关键字，可能会导致编译器出现“defined but not used”的警告或错误。

去掉static和inline

在所有目标文件中重复定义了相应rtl函数， 又因为没有使用所以报错

编译与链接II（10分）

- 29个， 此时dummy是静态局部变量，可以看到有29个文件重新编译，则有29个变量实体
- 58个， 多了29个，每个包含common.h和debug.h头文件的源文件都会有一个dummy变量
- 报错原因是重复定义变量，此前没报错是因为只声明未初始化为弱符号，初始化了的为强符号，多个强符号会发生重复定义错误

I/O 端口与接口

假如采用端口映射 I/O 的编址方式下, I/O 端口的地址从 0000H 开始, 系统板保留 1K 个 I/O 端口, 那么系统 I/O 地址的范围是多少? 假如总共采用 16 条地址线编址, 用户设计扩展接口时可以使用的端口的地址范围是多少? 请给出答案和解析。

1K个接口=2^10，每个端口是8个地址，那么地址就是0000H->8000H。变成16地址编线以后变成8001H-FFFFH。

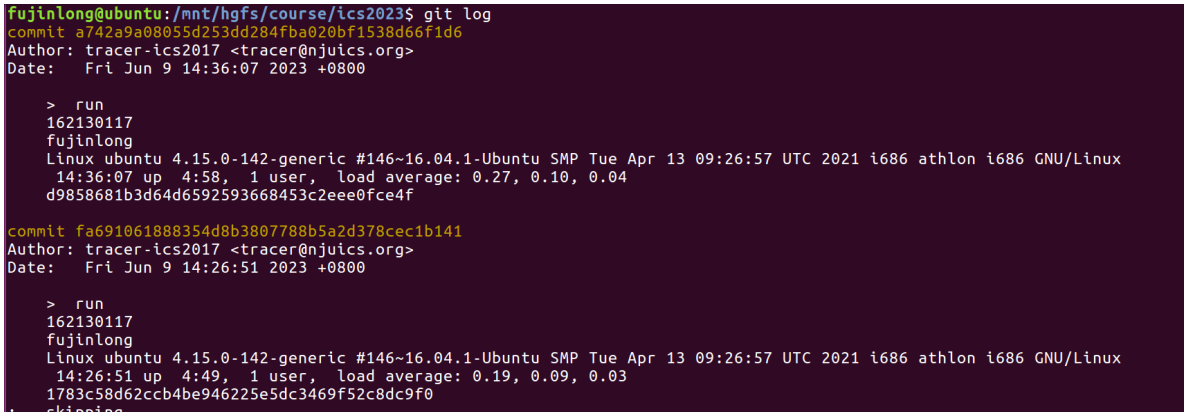
你能想到有什么来自 CPU 的信号参与了设备的选通或控制吗? 在这过程中 CPU 和设备传输了哪几种类型的信息? 请举一个常见的设备例子进行说明.

在计算机系统中，CPU和设备之间的通信需要通过一些信号进行控制和传输。其中，来自CPU的信号可以参与设备的选通或控制，例如：

1. 地址线：CPU通过地址线向设备发送地址信息，以选通需要进行数据传输的设备。
2. 数据线：CPU通过数据线向设备发送数据信息，或从设备接收数据信息。
3. 控制线：CPU通过控制线向设备发送控制信息，例如读写控制、中断请求等。

在这个过程中，CPU和设备之间传输的信息包括地址信息、数据信息和控制信息。其中，地址信息用于选通需要进行数据传输的设备，数据信息用于进行实际的数据传输，控制信息用于控制设备的读写操作或中断请求等。
一个常见的设备例子是硬盘。在计算机系统中，CPU通过地址线向硬盘发送读写操作的地址信息，通过数据线向硬盘发送读写的数据信息，通过控制线向硬盘发送读写控制信息，以进行数据的读写操作。在读写操作完成后，硬盘可以通过中断请求信号向CPU发送中断请求，以通知CPU读写操作已经完成。

git log截图



实验内容

实现剩余所有 x86 指令

add.c

8d(lea)

译码函数是lea_M2G，执行函数lea，填表

```
/* 0x8c */  EMPTY,IDEX(lea_M2G,lea), EMPTY, EMPTY,
```

83 e4(and)

opcode_table中使用了grp1，查表可知grp1[4]应填写执行函数make_EHelper(and)
译码函数是SI2E，需要实现make_DopHelper(SI)

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

  op->type = OP_TYPE_IMM;

  op->simm = instr_fetch(eip, op->width);

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

  rtl_sext(&op->val,&op->val,op->width);

  op->simm=op->val;

#ifdef DEBUG
```

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

实现执行函数make_EHelper(and)

```
make_EHelper(and)
{
    rtl_and(&id_dest->val, &id_dest->val, &id_src->val);
    operand_write(id_dest, &id_dest->val);

    rtl_li(&t0, 0);
    rtl_set_CF(&t0);
    rtl_set_OF(&t0);
    rtl_update_ZFSF(&id_dest->val, id_dest->width);

    print_asm_template2(and);
}
```

填写gp1

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

51(push)

译码函数为r,执行函数为push,都已完成，且50-57均相同，填表如下

```
/* 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),
```

eb(jmp)

译码函数I，执行函数jmp，都已完成，操作数长度为1字节，填表如下

```
/* 0xe8 */  IDEX(I,call), EMPTY, EMPTY, IDEXW(J, jmp, 1),
```

83 f8(cmp)

opcode_table中使用了grp1，查表可知grp1[7]应填写执行函数make_EHelper(cmp)
实现执行函数make_EHelper(cmp)，与sub相似，只是去掉写入结果的步骤

```
make_EHelper(cmp) {
  rtl_sub(&t2, &id_dest->val, &id_src->val);

  rtl_update_ZFSF(&t2, id_dest->width);

  rtl_sltu(&t0, &id_dest->val, &t2);
  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(cmp);
}
```

填表

```
make_group(gp1,
    EMPTY, EMPTY, EMPTY, EMPTY,
    EX(and), EX(sub), EMPTY, EX(cmp))
```

76(jbe)

译码函数，执行函数jcc，需要实现rtl_setcc,获取对应标志寄存器的值并计算结果

```
void rtl_setcc(rtlreg_t *dest, uint8_t subcode)
{
    bool invert = subcode & 0x1;
    enum { CC_O, CC_NO, CC_B, CC_NB, CC_E, CC_NE, CC_BE, CC_NBE, CC_S, CC_NS,
CC_P, CC_NP, CC_L, CC_NL, CC_LE, CC_NLE };

    switch (subcode & 0xe) {
        case CC_O:
            rtl_get_OF(dest);
            break;
        case CC_B:
            rtl_get_CF(dest);
            break;
        case CC_E:
            rtl_get_ZF(dest);
            break;
        case CC_BE:
            rtl_get_CF(&t0);
            rtl_get_ZF(&t1);
            rtl_or(dest, &t0, &t1);
            break;
        case CC_S:
            rtl_get_SF(dest);
            break;
        case CC_L:
            rtl_get_SF(&t0);
            rtl_get_OF(&t1);
            rtl_xor(dest, &t0, &t1);
            break;
        case CC_LE:
            rtl_get_ZF(&t0);
            rtl_get_SF(&t1);
            rtl_get_OF(&t2);
            rtl_xor(&t3, &t1, &t2);
            rtl_or(dest, &t0, &t3);
            break;
        default:
            panic("should not reach here");
        case CC_P:
            panic("n86 does not have PF");
    }
```

```
    if (invert) {
        rtl_xori(dest, dest, 0x1);
    }
}
```

70-7f相同，填表如下

```
/* 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),
```

01(add)

译码函数G2E,实现执行函数add如下

```
make_EHelper(add) {

  rtl_add(&t2, &id_dest->val, &id_src->val);
  rtl_sltu(&t3, &t2, &id_dest->val);
  operand_write(id_dest, &t2);

  rtl_update_ZFSF(&t2, id_dest->width);

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

  rtl_xor(&t0, &id_dest->val, &id_src->val);
  rtl_not(&t0);
  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(add);
}
```

00-05均为add,填表如下

```
/* 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,
```

c9(leave)

无译码函数，执行函数leave实现如下

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

    print_asm("leave");
}
```

填表

```
  /* 0xc8 */ EMPTY, EX(leave), EMPTY, EMPTY,
```

83 c4(add)

83跳转至grp1，大部分执行函数已实现，填表如下

```
make_group(gp1,
    EX(add), EX(or), EX(adc), EX(sbb),
    EX(and), EX(sub), EX(xor), EX(cmp))
```

39（cmp)

译码函数为G2E，执行函数为cmp,已实现，38-3d均为cmp,填表如下

```
/* 0x38 */  IDEXW(G2E,cmp,1), IDEX(G2E,cmp), IDEXW(E2G,cmp,1), IDEX(E2G,cmp),
/* 0x3c */  IDEXW(I2a,cmp,1), IDEX(I2a,cmp), EMPTY, EMPTY,
```

0f 94(setz)

译码函数E,执行函数setcc，已实现,90-9f相似，填表如下

```
/* 0x90 */  IDEXW(E,setcc,1), IDEXW(E,setcc,1), IDEXW(E,setcc,1),
IDEXW(E,setcc,1),
/* 0x94 */  IDEXW(E,setcc,1), IDEXW(E,setcc,1), IDEXW(E,setcc,1),
IDEXW(E,setcc,1),
/* 0x98 */  IDEX(E,setcc), IDEX(E,setcc), IDEX(E,setcc), IDEX(E,setcc),
/* 0x9c */  IDEX(E,setcc), IDEX(E,setcc), IDEX(E,setcc), IDEX(E,setcc),
```

0f b6(movzx)

译码函数mov_E2G,执行函数movxz,已实现,填表如下

```
  /* 0xb4 */ EMPTY, EMPTY, IDEXW(mov_E2G,movzx,1), EMPTY,
```

ff 45(inc)

opcode_table中使用了grp5，查表可知grp5[0]应填写执行函数make_EHelper(inc),实现如下

```
make_EHelper(inc)
{
    rtlreg_t tmp = 1;
    rtl_add(&t2, &id_dest->val, &tmp);
    rtl_sltu(&t3, &t2, &id_dest->val);
    operand_write(id_dest, &t2);

    rtl_update_ZFSF(&t2, id_dest->width);

    rtl_xor(&t0, &id_dest->val, &tmp);
    rtl_not(&t0);
    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_template1(inc);
}
```

填表

```
 /* 0xff */
make_group(gp5,
    EX(inc), EMPTY, EMPTY, EMPTY,
    EMPTY, EMPTY, EX(push), EMPTY)
```

add-longlong.c

e9(jmp)

译码函数是J，执行函数是jmp,填表如下

```
/* 0xe8 */  IDEX(I, call), IDEX(J, jmp), EMPTY, IDEXW(J, jmp,1),
```

0f 86(jcc)

译码函数是J，执行函数是jcc,且80-8f相同，填表如下

```
/* 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),
```

11（adc)

译码函数G2E，执行函数adc，填表如下

```
  /* 0x10 */    EMPTY, IDEX(G2E,adc), EMPTY, EMPTY,
```

5b(pop)

译码函数r，执行函数pop，且58-5f相同，填表如下

```
/* 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),
```

33(xor)

30-33均为xor，填表如下

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

09(or)

08-0d均为or,填表如下

```
/* 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),
```

make_EHelper(or)实现如下， 把CF OF设为0，更新ZF SF

```
make_EHelper(or)
{
    rtl_or(&id_dest->val, &id_dest->val, &id_src->val);
    operand_write(id_dest, &id_dest->val);
    t0 = 0;
    rtl_set_CF(&t0);
    rtl_set_OF(&t0);

    rtl_update_ZFSF(&id_dest->val, id_dest->width);

    print_asm_template2(or);
}
```

85(test)

84，85均为test，填表如下

```
  /* 0x84 */    IDEXW(G2E,test,1), IDEX(G2E,test), EMPTY, EMPTY,
```

make_EHelper(test)具体实现如下， 把CF OF设为0，更新ZF SF

```
make_EHelper(test)
{
    rtl_and(&id_dest->val, &id_dest->val, &id_src->val);
    t0 = 0;
    rtl_set_CF(&t0);
    rtl_set_OF(&t0);
    rtl_update_ZFSF(&id_dest->val, id_dest->width);

    print_asm_template2(test);
}
```

bits.c

6a(push)

译码函数为push_SI,执行函数为push,68和6a相近，填表如下

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

c1 f8(sar)

opcode_table中使用了grp2，查表可知grp2[7]应填写执行函数make_EHelper(sar)，填表如下

```
make_group(gp2,
    EMPTY, EMPTY, EMPTY, EMPTY,
    EMPTY, EMPTY, EMPTY, EX(sar))
```

make_EHelper(sar)实现如下

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

    print_asm_template2(sar);
}
```

d3 e2(shl)

opcode_table中使用了grp2，查表可知grp2[4]应填写执行函数make_EHelper(shl)，填表如下

```
make_group(gp2,
    EMPTY, EMPTY, EMPTY, EMPTY,
    EX(shl), EMPTY, EMPTY, EX(sar))
```

make_EHelper(shl)实现如下

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

    print_asm_template2(shl);
}
```

22(and)

20-25均为and,填表如下

```
/* 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,
```

f7 d0(not)

opcode_table中使用了grp3，查表可知grp3[2]应填写执行函数make_EHelper(not),实现如下

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

    print_asm_template1(not);
}
```

填表如下

```
make_group(gp3,
    EMPTY, EMPTY, EX(not), EMPTY,
    EMPTY, EMPTY, EMPTY, EMPTY)
```

bubble-sort.c

2b(sub)

28-2d均为sub,填表如下

```
 /* 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,
```

40(inc)

译码函数为r,执行函数为inc,40-47相同，填表如下

```
/* 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),
```

fact.c

48（dec)

译码函数r,执行函数dec实现如下

```
make_EHelper(dec)
{
    rtlreg_t tmp = 1;
    rtl_sub(&t2, &id_dest->val, &tmp);
    rtl_sltu(&t3, &t2, &id_dest->val);
    operand_write(id_dest, &t2);

    rtl_update_ZFSF(&t2, id_dest->width);

    rtl_xor(&t0, &id_dest->val, &tmp);
    rtl_not(&t0);
    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_template1(dec);
}
```

48-4f相同，填表如下

```
/* 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),
```

0f af(imul)

译码函数E2G， 执行函数imul2，填表如下

```
  /* 0xac */    EMPTY, EMPTY, EMPTY, IDEX(E2G, imul2),
```

goldbach.c

99(cltd)

查阅可知无译码函数，执行函数为cltd，当操作数为16位时，若ax<0则给dx赋值0xffff否则赋值0；操作数为32位时，若eax<0则给edx赋值0xffffffff否则赋值0，实现如下

```
make_EHelper(cltd)
{
    if (decoding.is_operand_size_16) {
        if ((int16_t)(cpu.eax) < 0) {
            cpu.edx = 0xffff;
        } else {
            cpu.edx = 0;
        }
    } else {
        if ((int32_t)(cpu.eax) < 0) {
            cpu.edx = 0xffffffff;
        } else {
            cpu.edx = 0;
        }
    }

    print_asm(decoding.is_operand_size_16 ? "cwtl" : "cltd");
}
```

填表如下

```
  /* 0x98 */ EMPTY, EX(cltd), EMPTY, EMPTY,
```

f7 7d(idiv)

opcode_table中使用了grp3，查表可知grp3[7]应填写执行函数make_EHelper(idiv),填表如下

```
make_group(gp3,
    EMPTY, EMPTY, EX(not), EMPTY,
    EMPTY, EMPTY, EMPTY, EX(idiv))
```

hello-str.c

0f be(movsx)

be和bf都是movsx,译码函数E2G,执行函数movsx,填表如下

```
  /* 0xbc */ EMPTY, EMPTY, IDEXW(E2G,movsx,1), IDEXW(E2G,movsx,2),
```

ff 24(jmp_rm)

ff跳转至grp5,grp5中的大部分指令的执行函数均已实现，填表如下

```
make_group(gp5,
    EX(inc), EX(dec), EX(call_rm), EX(call),
    EX(jmp_rm), EX(jmp), EX(push), EMPTY)
```

call_rm实现如下

```
make_EHelper(call_rm) {
  rtl_push(eip);
  decoding.is_jmp = 1;
  decoding.jmp_eip = id_dest->val;

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

f7 75(div)

f7跳转至grp3,应在grp3[6]处填写EX(div),grp3中部分指令已实现，填表如下

```
make_group(gp3,
    EMPTY, EMPTY, EX(not), EMPTY,
    EX(mul), EX(imul1), EX(div), EX(idiv))
```

load-store.c

0f b7(movzx)

译码函数mov_E2G,执行函数movzx,填表如下

```
/* 0xb4 */  EMPTY, EMPTY, IDEXW(mov_E2G,movzx,1), IDEXW(mov_E2G,movzx,2),
```

c1 eb(shr)

c1跳转至grp2,应在grp2[5]处填写EX(shr),实现如下

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

    print_asm_template2(shr);
}
```

填表如下

```
make_group(gp2,
    EMPTY, EMPTY, EMPTY, EMPTY,
    EX(shl), EX(shr), EMPTY, EX(sar))
```

sub-longlong.c

1b(sbb)

18-1dj均为sbb,填表如下

```
/* 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,
```

通过一键回归测试

```
fujinlong@ubuntu:/mnt/hgfs/course/ics2023/nemu$ bash runall.sh
NEMU compile OK
compiling testcases...
testcases compile OK
[  add-longlong] PASS!
[           add] PASS!
[           bit] PASS!
[   bubble-sort] 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!
```

IN/OUT 指令

实现 IN , OUT 两条指令

in指令实现如下

```
make_EHelper(in)
{
    id_dest->val = pio_read(id_src->val, id_dest->width);
    operand_write(id_dest, &id_dest->val);

    print_asm_template2(in);

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

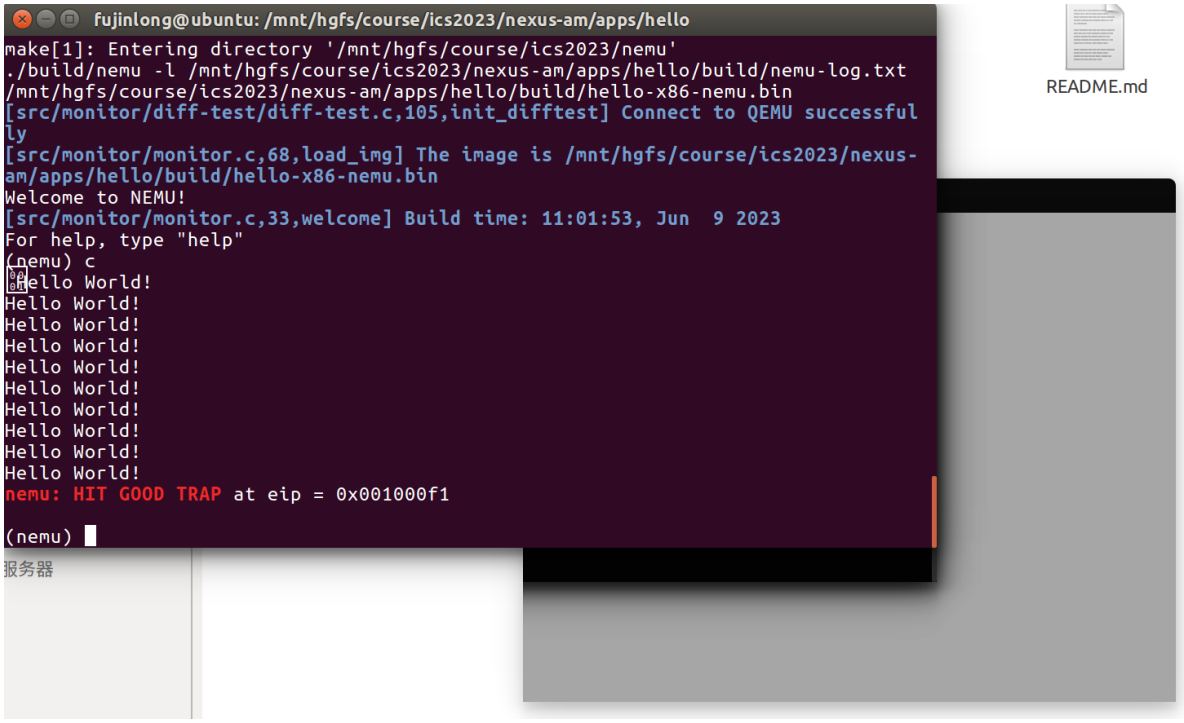
out指令实现如下

```
make_EHelper(out)
{
    pio_write(id_dest->val, id_dest->width, id_src->val);

    print_asm_template2(out);

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

填表如下

```
/* 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),
```

成功运行 nexus-am/apps/hello 程序



实现时钟设备

实现 _uptime() 函数;

```
unsigned long _uptime() {
  return inl(RTC_PORT) - boot_time;
}
```

成功运行 timetest 程序

```
fujinlong@ubuntu:/mnt/hgfs/course/ics2023/nexus-am/tests/timetest$ make run
Building timetest [x86-nemu]
+ CC main.c
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am'
make[2]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/am'
Building am [x86-nemu]
+ CC arch/x86-nemu/src/ioe.c
+ AR /mnt/hgfs/course/ics2023/nexus-am/am/build/am-x86-nemu.a
make[2]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/am'
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am'
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
make[1]: *** 没有指明目标并且找不到 makefile。 停止。
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
/mnt/hgfs/course/ics2023/nexus-am/Makefile.compile:86: recipe for target 'klib' fai
make: [klib] Error 2 (ignored)
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nemu'
./build/nemu -l /mnt/hgfs/course/ics2023/nexus-am/tests/timetest/build/nemu-log.txt
[src/monitor/diff-test/diff-test.c,105,init_difftest] Connect to QEMU successfully
[src/monitor/monitor.c,68,load_img] The image is /mnt/hgfs/course/ics2023/nexus-am/
Welcome to NEMU!
[src/monitor/monitor.c,33,welcome] Build time: 11:01:53, Jun  9 2023
For help, type "help"
(nemu) c
1 second.
2 seconds.
3 seconds.
4 seconds.
5 seconds.
6 seconds.
7 seconds.
8 seconds.
9 seconds.
10 seconds.
11 seconds.
12 seconds.
13 seconds.
14 seconds.
15 seconds.
16 seconds.
17 seconds.
18 seconds.
```

运行跑分项目

成功运行 dhrystone , coremark , microbench 三个跑分项目;

dhrystone

- x86-nemu

```
fujinlong@ubuntu:/mnt/hgfs/course/ics2023/nexus-am/apps/dhrystone$ make ARCH=x86-nemu ALL=dhrystone run
Building dhrystone [x86-nemu]
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am'
make[2]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/am'
Building am [x86-nemu]
make[2]: Nothing to be done for 'archive'.
make[2]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/am'
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am'
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
make[1]: *** 没有指明目标并且找不到 makefile。 停止。
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
/mnt/hgfs/course/ics2023/nexus-am/Makefile.compile:86: recipe for target 'klib' failed
make: [klib] Error 2 (ignored)
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nemu'
+ CC src/monitor/cpu-exec.c
src/monitor/cpu-exec.c: In function ‘cpu_exec’:
src/monitor/cpu-exec.c:29:15: warning: unused variable ‘prev_eip’ [-Wunused-variable]
      uint32_t prev_eip=cpu.eip;
               ^
+ CC src/monitor/debug/expr.c
+ CC src/monitor/debug/ui.c
+ CC src/monitor/debug/watchpoint.c
+ CC src/monitor/diff-test/diff-test.c
+ CC src/monitor/diff-test/gdb-host.c
+ CC src/monitor/diff-test/protocol.c
+ CC src/monitor/monitor.c
+ LD build/nemu
./build/nemu -l /mnt/hgfs/course/ics2023/nexus-am/apps/dhrystone/build/nemu-log.txt /mnt/hgfs/course/ics
[src/monitor/monitor.c,68,load_img] The image is /mnt/hgfs/course/ics2023/nexus-am/apps/dhrystone/build/
Welcome to NEMU!
[src/monitor/monitor.c,33,welcome] Build time: 11:17:48, Jun  9 2023
For help, type "help"
(nemu) c
Dhrystone Benchmark, Version C, Version 2.2
Trying 500000 runs through Dhrystone.
Finished in 12566 ms
------------------------------------------------
Dhrystone PASS         81 Marks
                  vs. 100000 Marks (i7-6700 @ 3.40GHz)
nemu: HIT GOOD TRAP at eip = 0x001000f1
```

- native

```
fujinlong@ubuntu:/mnt/hgfs/course/ics2023/nexus-am/apps/dhrystone$ make ARCH=native ALL=dhrystone run
Building dhrystone [native]
+ CC dry.c
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am'
make[2]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/am'
Building am [native]
+ CC arch/native/src/gui.c
+ CC arch/native/src/ioe.c
+ CC arch/native/src/trm.c
+ AR /mnt/hgfs/course/ics2023/nexus-am/am/build/am-native.a
make[2]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/am'
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am'
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
make[1]: *** 没有指明目标并且找不到 makefile。 停止。
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
/mnt/hgfs/course/ics2023/nexus-am/Makefile.compile:86: recipe for target 'klib' failed
make: [klib] Error 2 (ignored)
Dhrystone Benchmark, Version C, Version 2.2
Trying 500000 runs through Dhrystone.
Finished in 35 ms
------------------------------------------------
Dhrystone PASS         29436 Marks
                  vs. 100000 Marks (i7-6700 @ 3.40GHz)
```

coremark

98(cwtl), 对寄存器的符号扩展 ,无译码函数实现如下

```
make_EHelper(cwtl)
{
    if (decoding.is_operand_size_16) {
        rtl_lr_b(&t0, R_AL);
        rtl_sext(&t0, &t0, 1);
        rtl_sr_w(R_AX, &t0);
    } else {
        rtl_lr_w(&t0, R_AX);
        rtl_sext(&t0, &t0, 2);
        rtl_sr_l(R_EAX, &t0);
    }

    print_asm(decoding.is_operand_size_16 ? "cbtw" : "cwtl");
}
```

f7 d8(neg),跳转至grp3,执行函数实现如下

```
make_EHelper(neg)
{
    if (!id_dest->val) {
        t0 = 0;
        rtl_set_CF(&t0);
    } else {
        t0 = 1;
        rtl_set_CF(&t0);
    }

    id_dest->val = -id_dest->val;
    operand_write(id_dest, & id_dest->val);
    rtl_update_ZFSF(&id_dest->val, id_dest->width);

    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_template1(neg);
}
```

填表如下

```
make_group(gp3,
    EMPTY, EMPTY, EX(not), EX(neg),
    EX(mul), EX(imul1), EX(div), EX(idiv))
```

- x86-nemu

```
fujinlong@ubuntu:/mnt/hgfs/course/ics2023/nexus-am/apps/coremark$ make ARCH=x86-nemu ALL=coremark run
Building coremark [x86-nemu]
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am'
make[2]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/am'
Building am [x86-nemu]
make[2]: Nothing to be done for 'archive'.
make[2]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/am'
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am'
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
make[1]: *** 没有指明目标并且找不到 makefile。 停止。
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
/mnt/hgfs/course/ics2023/nexus-am/Makefile.compile:86: recipe for target 'klib' failed
make: [klib] Error 2 (ignored)
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nemu'
+ CC src/cpu/exec/data-mov.c
+ CC src/cpu/exec/exec.c
+ LD build/nemu
./build/nemu -l /mnt/hgfs/course/ics2023/nexus-am/apps/coremark/build/nemu-log.txt /mnt/hgfs/course/ic
[src/monitor/monitor.c,68,load_img] The image is /mnt/hgfs/course/ics2023/nexus-am/apps/coremark/build
Welcome to NEMU!
[src/monitor/monitor.c,33,welcome] Build time: 16:01:19, Jun 11 2023
For help, type "help"
(nemu) c
Running CoreMark for 1000 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total time (ms)  : 35614
Iterations       : 1000
Compiler version : GCC5.4.0 20160609
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xd340
Finised in 35614 ms.
==================================================
CoreMark PASS       125 Marks
            vs. 100000 Marks (i7-6700 @ 3.40GHz)
nemu: HIT GOOD TRAP at eip = 0x001000f1
```

- native

```
fujinlong@ubuntu:/mnt/hgfs/course/ics2023/nexus-am/apps/coremark$ make ARCH=native ALL=coremark run
Building coremark [native]
+ CC src/core_list_join.c
+ CC src/core_main.c
+ CC src/core_matrix.c
+ CC src/core_portme.c
+ CC src/core_state.c
+ CC src/core_util.c
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am'
make[2]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/am'
Building am [native]
make[2]: Nothing to be done for 'archive'.
make[2]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/am'
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am'
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
make[1]: *** 没有指明目标并且找不到 makefile。 停止。
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
/mnt/hgfs/course/ics2023/nexus-am/Makefile.compile:86: recipe for target 'klib' failed
make: [klib] Error 2 (ignored)
Running CoreMark for 1000 iterations
2K performance run parameters for coremark.
CoreMark Size    : 666
Total time (ms)  : 125
Iterations       : 1000
Compiler version : GCC5.4.0 20160609
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0xd340
Finised in 125 ms.
==================================================
CoreMark PASS       35748 Marks
            vs. 100000 Marks (i7-6700 @ 3.40GHz)
```

microbench

c2(ret),执行函数ret_imm,实现如下

```
make_EHelper(ret_imm) {
  rtl_pop(&decoding.jmp_eip);
  cpu.esp+=id_dest->val;
  decoding.is_jmp = 1;
  print_asm("ret %x", id_dest->val);
}
```

填表如下

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

d3 c2（rol),d3跳转grp2,grp2[0]应填写EX(rol), 循环左移，并最后修改CF，实现如下

```
make_EHelper(rol)
{
    for (t0 = 0; t0 < id_src->val; t0++) {
        rtl_shri(&t1, &id_dest->val, id_dest->width * 8 - 1);
        rtl_shli(&t2, &id_dest->val, 1);
        id_dest->val = t1 + t2;
    }

    rtl_set_CF(&t1);
    operand_write(id_dest, &id_dest->val);

    print_asm_template2(rol);
}
```

填表如下

```
make_group(gp2,
    EX(rol), EMPTY, EMPTY, EMPTY,
    EX(shl), EX(shr), EMPTY, EX(sar))
```

- x86-nemu

```
fujinlong@ubuntu:/mnt/hgfs/course/ics2023/nexus-am/apps/microbench$ make ARCH=x86-nemu ALL=microbench run
Building microbench [x86-nemu]
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am'
make[2]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/am'
Building am [x86-nemu]
make[2]: Nothing to be done for 'archive'.
make[2]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/am'
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am'
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
make[1]: *** 没有指明目标并且找不到 makefile。 停止。
make[1]: Leaving directory '/mnt/hgfs/course/ics2023/nexus-am/libs/klib'
/mnt/hgfs/course/ics2023/nexus-am/Makefile.compile:86: recipe for target 'klib' failed
make: [klib] Error 2 (ignored)
make[1]: Entering directory '/mnt/hgfs/course/ics2023/nemu'
+ CC src/cpu/exec/exec.c
+ CC src/cpu/exec/logic.c
+ LD build/nemu
./build/nemu -l /mnt/hgfs/course/ics2023/nexus-am/apps/microbench/build/nemu-log.txt /mnt/hgfs/course/ics2
[src/monitor/monitor.c,68,load_img] The image is /mnt/hgfs/course/ics2023/nexus-am/apps/microbench/build/m
Welcome to NEMU!
[src/monitor/monitor.c,33,welcome] Build time: 11:17:48, Jun  9 2023
For help, type "help"
(nemu) c
[qsort] Quick sort: * Passed.
  min time: 1699 ms [324]
[queen] Queen placement: * Passed.
  min time: 1104 ms [467]
[bf] Brainf**k interpreter: * Passed.
  min time: 9789 ms [267]
[fib] Fibonacci number: * Passed.
  min time: 100202 ms [28]
[sieve] Eratosthenes sieve: * Passed.
  min time: 32959 ms [128]
[15pz] A* 15-puzzle search: * Passed.
  min time: 6308 ms [91]
[dinic] Dinic's maxflow algorithm: * Passed.
  min time: 3172 ms [426]
[lzip] Lzip compression: * Passed.
  min time: 17986 ms [147]
[ssort] Suffix sort: * Passed.
  min time: 1898 ms [311]
[md5] MD5 digest: * Passed.
  min time: 15919 ms [123]
================================================
MicroBench PASS        231 Marks
                vs. 100000 Marks (i7-6700 @ 3.40GHz)
nemu: HIT GOOD TRAP at eip = 0x001000f1
```

- native

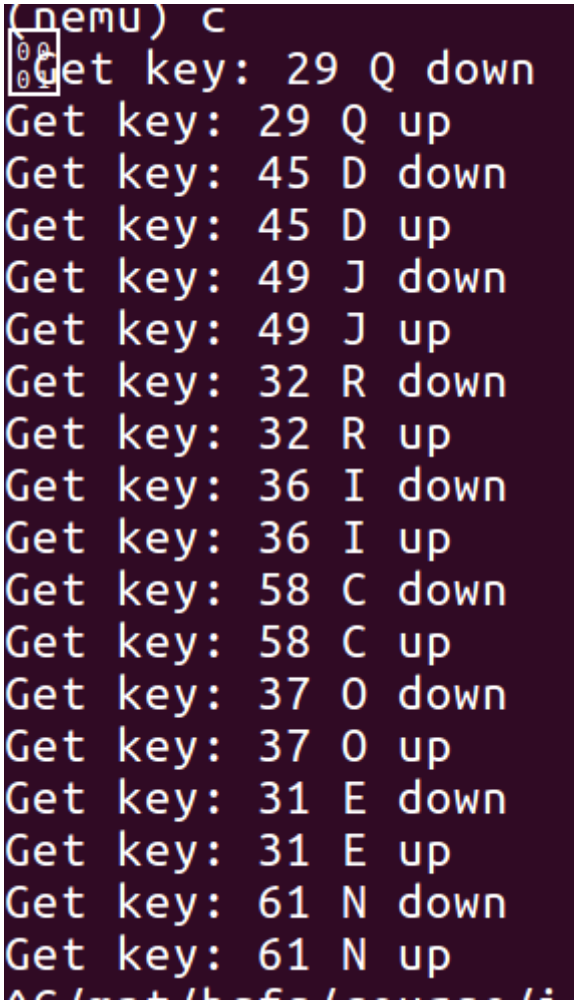
```
[qsort] Quick sort: * Passed.
  min time: 11 ms [50172]
[queen] Queen placement: * Passed.
  min time: 9 ms [57322]
[bf] Brainf**k interpreter: * Passed.
  min time: 43 ms [60951]
[fib] Fibonacci number: * Passed.
  min time: 420 ms [6803]
[sieve] Eratosthenes sieve: * Passed.
  min time: 116 ms [36556]
[15pz] A* 15-puzzle search: * Passed.
  min time: 23 ms [25182]
[dinic] Dinic's maxflow algorithm: * Passed.
  min time: 13 ms [104123]
[lzip] Lzip compression: * Passed.
  min time: 86 ms [30777]
[ssort] Suffix sort: * Passed.
  min time: 11 ms [53772]
[md5] MD5 digest: * Passed.
  min time: 40 ms [48982]
=====================================================
MicroBench PASS         47464 Marks
                    vs. 100000 Marks (i7-6700 @ 3.40GHz)
Exit (0)
```

实现键盘设备

实现 _read_key() 函数

```
int _read_key()
{
    uint32_t key_code = _KEY_NONE;
    if (inb(0x64) & 0x1)
        key_code = inl(0x60);
    return key_code;
}
```

成功运行 keytest 程序



添加内存映射 I/O

在 paddr_read() 和 paddr_write() 中添加内存映射 I/O 判断

引入头文件

```
#include "device/mmio.h"
```

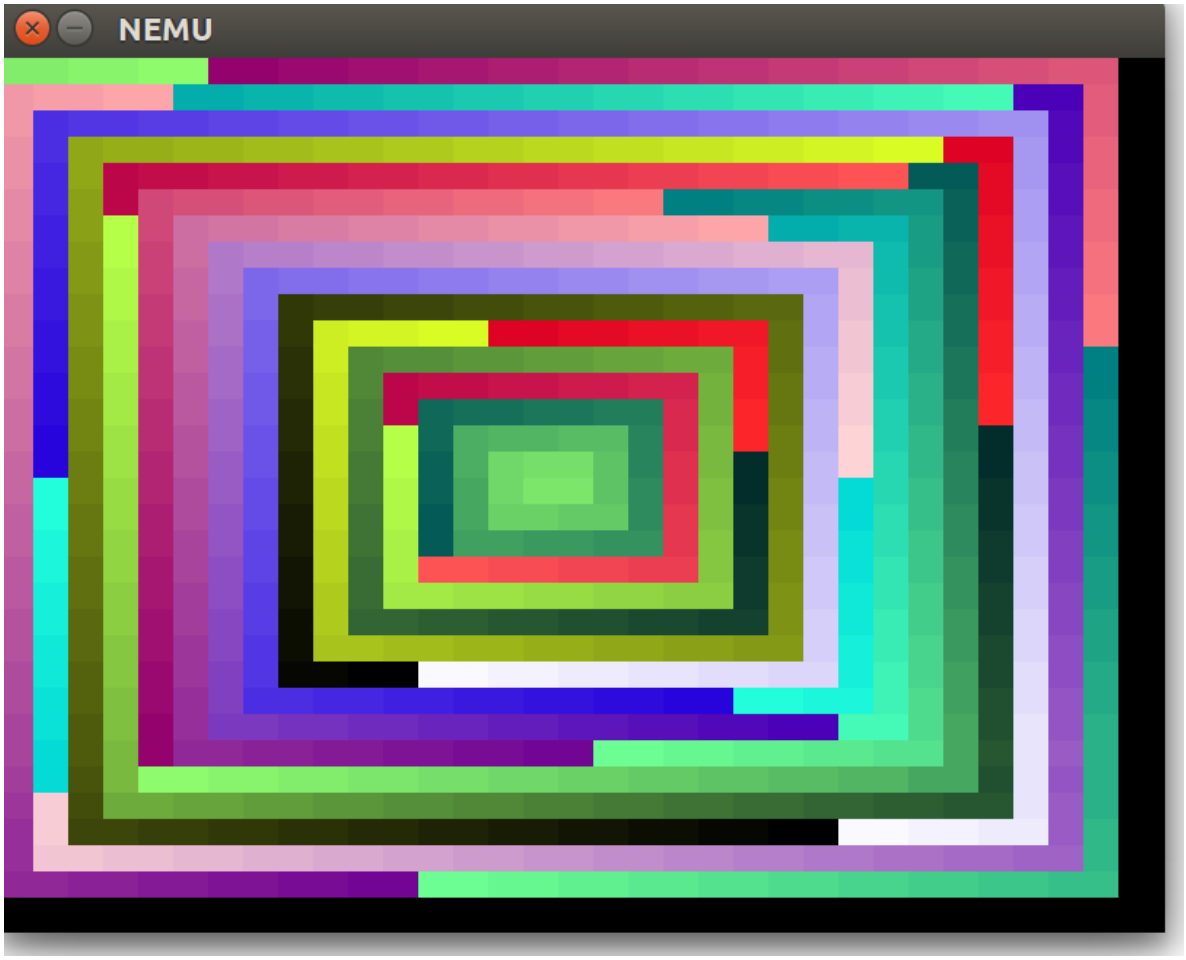
paddr_read

```
uint32_t paddr_read(paddr_t addr, int len)
{
    int mmio_id = is_mmio(addr);
    if (mmio_id != -1)
        return mmio_read(addr, len, mmio_id);
    else
        return pmem_rw(addr, uint32_t) & (~0u >> ((4 - len) << 3));
}
```

paddr_write

```
void paddr_write(paddr_t addr, int len, uint32_t data)
{
    int mmio_id = is_mmio(addr);
    if (mmio_id != -1)
        return mmio_write(addr, len, data, mmio_id);
    else
        memcpy(guest_to_host(addr), &data, len);
}
```

成功运行 videotest 程序



运行打字小游戏

捕捉死循环

为了实现当用户程序陷入死循环时暂停并输出提示信息的功能，可以在NEMU中添加一个计数器，用于记录程序执行的指令数。每当执行一条指令时，将计数器加1。如果计数器的值超过一个阈值（例如1000000），则认为程序陷入了死循环，此时暂停程序并输出提示信息。
具体实现步骤如下：

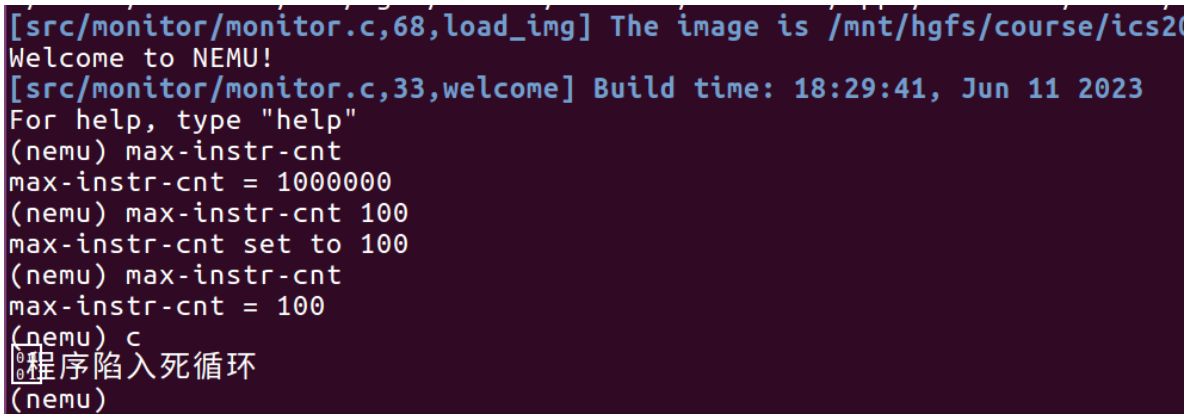
1. 在CPU_state中添加一个计数器的阈值，用于记录程序执行的指令数。
2. 在cpu_exec的执行循环中添加一个计数器变量，每当执行一条指令时，将计数器加1。
3. 在cpu_exec的执行循环中，判断计数器的值是否超过一个阈值（例如1000000），如果超过则认为程序陷入了死循环，此时暂停程序并输出提示信息。
4. 在ui.c中添加一个命令，用于设置计数器的阈值。
5. 在NEMU的命令行中输入max-instr-cnt命令，即可设置计数器的阈值。例如，输入max-instr-cnt 1000000可以将计数器的阈值设置为1000000。如果不带参数直接输入max-instr-cnt，则会显示当前计数器的阈值。

以下是实现代码：

```
struct CPU_state {
    // ...
    uint64_t max_instr_cnt;
};

static inline void restart()
{
    // ...
    cpu.max_instr_cnt = 1000000;//初始化
    // ...
}
```

```c
void cpu_exec(uint64_t n) {
    // ...
    uint64_t instr_cnt = 0;
    for (; n > 0; n--) {
        // ...
        exec_wrapper(print_flag);
        cpu.instr_cnt++; // 每执行一条指令，计数器加1
        // ...
        if (instr_cnt >= cpu.max_instr_cnt) {
            printf("程序陷入死循环\n");
            nemu_state = NEMU_STOP;
        }// 计数器超过阈值，程序陷入死循环
    // ...
}

// 在ui.c中添加一个命令，用于设置计数器的阈值
static int cmd_max_instr_cnt(char *args) {
    if (args == NULL) {
        printf("max-instr-cnt = %llu\n", cpu.max_instr_cnt);
        return 0;
    }
    cpu.max_instr_cnt = strtoull(args, NULL, 0);
    printf("max-instr-cnt set to %llu\n", cpu.max_instr_cnt);
    return 0;
}

//在同一个文件中找到cmd_table数组，将以下代码添加到数组的末尾
static struct {
    char *name;
    char *description;
    int (*handler)(char *);
} cmd_table[] = {
    // ...
    { "max-instr-cnt", "Set the maximum instruction count for detecting infinite
loops", cmd_max_instr_cnt },
};
```

测试结果



遇到的问题及解决办法

1. 遇到问题： 实现and指令的时候nemu打出来的反汇编的立即数是0xf0而反汇编文件里面是0xfffffff0
   解决方案： 在make_DoHelper函数里面没有实现符号扩展，由于默认是0扩展，所以实现失败。

2. 遇到问题： 键盘测试总是无法识别相应的键盘，按下去键盘什么都不会输出
   解决方案： 应该在make run之后弹出来的窗口里面打字，而不是在终端里面的nemu里按键

实验心得

通过实现各种指令，我明白了需要了解CPU指令的执行过程和寄存器的使用，以便正确地实现指令函数，还需要进行充分的测试和调试，以确保指令函数的正确性和稳定性。 除此之外，我还了解了两种 I/O 编址方式: 端口映射 I/O 和内存映射 I/O，以及cpu与不同设备之间的联系，从而怎么使程序能够更好地与用户进行交互的。

其他备注

助教真帅