

# 华中科技大学

## 2019

### 系统能力综合训练 课程设计报告

题 目:	X86 模拟器设计
专 业:	计算机科学与技术
班 级:	CS1601
学 号:	U201614526
姓 名:	田志伟
电 话:	15827451200
邮 件:	tian_zw@qq.com
完成日期:	2020-1-1



# 目 录

1 课程设计概述.....	1
1.1 课设目的.....	1
1.2 实验环境.....	1
2 实验过程.....	2
2.1 PA0.....	2
2.1.1 环境配置过程.....	2
2.2 PA1.....	2
2.2.1 总体设计.....	2
2.2.2 详细设计.....	2
2.2.3 运行结果.....	5
2.2.4 问题解答.....	7
2.3 PA2.....	8
2.3.1 总体设计.....	8
2.3.2 详细设计.....	8
2.3.3 运行结果.....	11
2.3.4 问题解答.....	13
2.4 PA3.....	13
2.4.1 总体设计.....	13
2.4.2 详细设计.....	14
2.4.3 运行结果.....	16
3 设计总结与心得.....	17
3.1 课设总结.....	17
3.2 课设心得.....	18
参考文献.....	19

# 1 课程设计概述

## 1.1 课设目的

理解"程序如何在计算机上运行"的根本途径是从"零"开始实现一个完整的计算机系统. 南京大学计算机科学与技术系计算机系统基础课程的小型项 (Programming Assignment, PA)将提出 x86 架构的一个教学版子集 n86, 指导学生实现一个功能完备的 n86 模拟器 NEMU(NJU EMUlator), 最终在 NEMU 上运行游戏"仙剑奇侠传", 来让学生探究"程序在计算机上运行"的基本原理. NEMU 受到了 QEMU 的启发, 并去除了大量与课程内容差异较大的部分. PA 包括一个准备实验 (配置实验环境)以及 5 部分连贯的实验内容:

- 简易调试器
- 冯诺依曼计算机系统
- 批处理系统
- 分时多任务
- 程序性能优化

## 1.2 实验环境

- CPU 架构: x64
- 操作系统: GNU/Linux
- 编译器: GCC
- 编程语言: C

## 2 实验过程

### 2.1 PA0

#### 2.1.1 环境配置过程

首先安装 docker，按照 pdf 中所讲，编写 Dockerfile，生成镜像，创建容器。在 dokcer 中安装相关工具包，练习使用 vim。

下载实验框架代码，修改 makefile 中的学号姓名信息，git submit，并在 git origin 添加个人 git hub，方便个人代码管理和分享。

在后续实验中发现，macOS 中的 xquartz 有问题未解决，无法在 docker 中运行 GUI 程序，将实验环境更换为 Ubuntu 虚拟机，将已完成的代码从 github 上 git clone 下来，PA0 环境配置到此结束。

### 2.2 PA1

#### 2.2.1 总体设计

利用 C 语言的联合在 reg.h 中实现寄存器结构体，之后实现简易调试器。

在 nemu/src/monitor/debug 目录中实现如下任务。

Task PA1.1: 实现单步执行 cmd\_si，打印寄存器状态 cmd\_info，扫描内存 cmd\_x

Task PA1.2: 实现算数表达式求值 cmd\_p

Task PA1.3: 实现所有要求 cmd\_w, cmd\_d

#### 2.2.2 详细设计

##### 1) 寄存器

```
union{
    struct {
        rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
    };
    union{
        paddr_t _32;
        ioaddr_t _16;
        bool _8[2];
    } gpr[8];
};
```

使用联合来公用 32 位寄存器的低 16 位和 16 位寄存器和两个 8 位寄存器。

##### 2) 单步执行

```
static int cmd_si(char *args){ //single step
    char *arg = strtok(NULL, " ");
    if(arg == NULL){
        cpu_exec(1);
        return 0;
    }
    char *leftover;
    unsigned long n = strtoul(arg, &leftover, 10);
    if(*leftover != '\0'){
        printf("Please input the right arguement!\n");
    }else{
        cpu_exec(n);
    }

    return 0;
}
```

首先判断输入参数是否为空，空则单步执行 1，不为空则判断参数是否为数字，为数字则执行 n 条指令，不为数字则输出报错信息。

### 3) 打印寄存器状态

```
static void printRegister(int size, int index){
    switch(size) {
        case 1: printf("%s\t%#04X\n", reg_name(index, 1), reg_b(index)); break;
        case 2: printf("%s\t%#06X\n", reg_name(index, 2), reg_w(index)); break;
        case 4: printf("%s\t%#010X\n", reg_name(index, 4), reg_l(index)); break;
        case -1: printf("eip\t%#010X\n", cpu.eip); break;
        default: break;
    }
}
```

完成打印单个寄存器信息的函数，之后遍历调用即可。

### 4) 扫描内存

```
static int cmd_x(char *args){ //scan memory
    if(args == NULL){
        printf("Nothing to scan!\n");
        return 0;
    }

    char *Size = strtok(args, " ");
    int size = -1;
    sscanf(Size, "%d", &size);
    if(size < 0){
        printf("Wrong size to scan!\n");
        return 0;
    }

    char *Addr = Size + strlen(Size) + 1;

    char *leftover;
    uint64_t addr = strtoul(Addr, &leftover, 0);
    if(*leftover != '\0'){
        printf("Wrong addr to scan!\n");
        return 0;
    }

    for(int i = 0; i < size; i++)
        printf("%#010X\t\n", vaddr_read(addr + i * 4, 4));

    return 0;
}
```

首先读取参数，读取初始地址和读取的长度，若不符合标准则输出报错信息，之后利用 `vaddr_read()` 函数遍历输出内存信息。

#### 5) 算数表达式求值

```
{ " +", TK_NOTYPE, 0}, //空格
{ "\\b[0-9]+\\b", TK_NUMBER, 0}, //数字
{ "\\b0[xX][0-9a-fA-F]+\\b", TK_HEX, 0}, //十六进制
{ "\\$(eax|EBX|ecx|ECX|edx|EDX|ebp|EBP|esp|ESP|esi|ESI|edi|EBX|ECX|EDX|EBP|ESP|ESI|EDI)", TK_REGISTER, 0}, //寄存器
{ "\\t+", TK_NOTYPE, 0}, //tab
{ "\\|\\|", TK_OR, 1}, //或
{ "&&", TK_AND, 2}, //与
{ "==", TK_EQ, 3}, //等于
{ "!=", TK_NEQ, 3}, //不等于
{ "<=", TK_LE, 3}, //小于等于
{ ">=", TK_GE, 3}, //大于等于
{ "<", TK_L, 3}, //小于
{ ">", TK_G, 3}, //大于
{ "\\+", TK_PLUS, 4}, //加号
{ "-", TK_SUB, 4}, //减号
{ "\\*", TK_MUL, 5}, //乘号
{ "/", TK_DIV, 5}, //除号
{ "!", TK_NOT, 6}, //取反
{ "\\(", TK_LB, 7}, //左括号
{ "\\)", TK_RB, 7}, //右括号
```

表达式求值过程主要难点在于正则表达式匹配，填写正则表达式中需要匹配的字符，规则以及算数优先级。

并在 `expr` 函数中判断出\*为指针还是乘号，-为负号还是减号。

之后调用 `caculation` 函数进行计算，`caculation` 函数递归求值的主题框架指导手册里已经给出，只要按照需求实现判断括号的 `checkParentheses()` 函数和寻找主运算符的 `findOperator()` 函数即可。

```
bool checkParentheses(int l, int r){
    if(tokens[l].type == TK_LB && tokens[r].type == TK_RB){
        int n = 0; //记录除初始括号外单独的左括号数目，每次配对减一，小于0的时候返回false，以此来保证括号起始位置和结束位置配对
        for(int i = l + 1; i < r; i++){
            if(tokens[i].type == TK_LB) n++;
            else if(tokens[i].type == TK_RB) n--;
            if(n < 0){
                return false;
            }
        }
        if(n == 0)
            return true;
    }
    return false;
}
```

```
int findOperator(int left, int right){
    int min_level = 8;
    int op = left;
    int n = 0;
    for(int i = left; i <= right; i++){
        if(tokens[i].type == TK_NUMBER || tokens[i].type == TK_HEX || tokens[i].type == TK_REGISTER)
            continue;
        else if(tokens[i].type == TK_LB)
            n++;
        else if(tokens[i].type == TK_RB)
            n--;
        if(n != 0) //括号未结束
            continue;
        if(tokens[i].level < min_level){
            min_level = tokens[i].level;
            op = i;
        }
    }
    return op;
}
```

整个计算过程需要注意的是，在判断\*和-时，注意判断寄存器和括号，另外在正则匹配>和>=号时，需要优先匹配>=号，否则会被忽略掉。

#### 6) 监视点设置和删除

监视点设置和删除主要实现的即链表的增删查功能，比较简单。

增加部分如下：

```
WP* res = free_;
free_ = free_ -> next;
res -> next = head;
head = res;
res->value = v;
strcpy(res->expression, args);
return res;
```

删除部分如下：

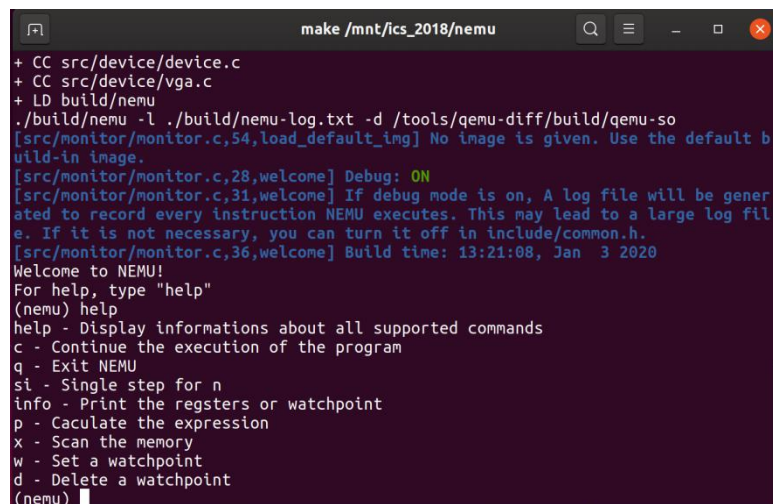
```
WP *wp = &wp_pool[n];

bool success = false;
WP *p = head;
if(p->NO == wp->NO){
    head = head->next;
    wp->next = free_;
    free_ = wp;
    success = true;
}
else{
    while(p != NULL){
        if(p->next->NO == wp->NO){
            p->next = p->next->next;
            wp->next = free_;
            free_ = wp;
            success = true;
            break;
        }
        p = p->next;
    }
}
```

到此为止，PA1 的主体功能实现完毕。

### 2.2.3 运行结果

#### 1) 初始化运行并执行 help



```
make /mnt/ics_2018/nemu
+ CC src/device/device.c
+ CC src/device/vga.c
+ LD build/nemu
./build/nemu -l ./build/nemu-log.txt -d /tools/qemu-diff/build/qemu-so
[src/monitor/monitor.c,54,load_default_img] No image is given. Use the default b
uild-in image.
[src/monitor/monitor.c,28,welcome] Debug: ON
[src/monitor/monitor.c,31,welcome] If debug mode is on, A log file will be gener
ated to record every instruction NEMU executes. This may lead to a large log fil
e. If it is not necessary, you can turn it off in include/common.h.
[src/monitor/monitor.c,36,welcome] Build time: 13:21:08, Jan  3 2020
Welcome to NEMU!
For help, type "help"
(nemu) help
help - Display informations about all supported commands
c - Continue the execution of the program
q - Exit NEMU
si - Single step for n
info - Print the registers or watchpoint
p - Caculate the expression
x - Scan the memory
w - Set a watchpoint
d - Delete a watchpoint
(nemu)
```



## 2) 单步执行

```
(nemu) si
100000: b8 34 12 00 00      movl $0x1234,%eax
(nemu) si 2
100005: b9 27 00 10 00      movl $0x100027,%ecx
10000a: 89 01               movl %eax,(%ecx)
(nemu) █
```

## 3) 打印寄存器信息

```
(nemu) info r
eax    0X00001234
ecx    0X00100027
edx    0X776FA905
ebx    0X09413C71
esp    0X26578C43
ebp    0X446BEB5A
esi    0X2B14BBD7
edi    0X54B86A99
ax     0X1234
cx     0X0027
dx     0XA905
bx     0X3C71
sp     0X8C43
bp     0XEB5A
si     0XBB07
di     0X6A99
al     0X34
cl     0X27
dl     0X05
bl     0X71
ah     0X12
ch     0000
dh     0XA9
bh     0X3C
eip    0X0010000C
(nemu) █
```

## 4) 表达式求值

```
(nemu) p (1+(3*2)) + (($eax-$eax)+ (*$eip - 1**$eip) + (0x5--5+ *0x100005 - *0x100005)*4)
47
(nemu) █
```

## 5) 监视点的设置和删除

```
(nemu) w 1 1
Illegal operator at the front the expression!
Illigal expression to caculate!
(nemu) w 1
(nemu) info w
0      1      1
(nemu) w 2
(nemu) info w
1      2      2
0      1      1
(nemu) w $eax
(nemu) info w
2      $eax   4660
1      2      2
0      1      1
(nemu) █
```

```
(nemu) w $eax
(nemu) info w
2      $eax   4660
1      2      2
0      1      1
(nemu) d 1
Delete successful!
(nemu) info w
2      $eax   4660
0      1      1
(nemu) █
```



所有的功能均实现完毕。

## 2.2.4 问题解答

### 1) 理解基础设施

$500 \times 90\% \times 30 \times 20 = 270000s = 75h$

一学期将要花费 75 个小时的时间在调试上。

$500 \times 90\% \times 20 \times 20 = 180000s = 50h$

一学期可以节省 50 个小时调试的时间。

### 2) 查阅 i386 手册

CF 为进位标志位。

ModR/M 字节对于不定长指令用于指定操作的寄存器号和内存。

mov 指令 eg: 88 01

88 位 opcode 指定指令为 mov 指令, 01 位 ModR/M 字节 00000001, Mod=00

指定 1 字节, Reg=000 即 eax/ax/al, R/M=001 即 ecx, 最后的指令即为

mov ecx, al

### 3) shell 命令

执行 `find . "(" -name "*.h" -or -name "*.c" ")" -print | xargs wc -l` 指令, 即可统计代码总行数, 统计结果如下:

```
306 ./tools/qemu-diff/src/protocol.c
121 ./tools/qemu-diff/src/diff-test.c
48 ./include/debug.h
8 ./include/monitor/expr.h
9 ./include/monitor/monitor.h
20 ./include/monitor/watchpoint.h
82 ./include/memory/mmu.h
18 ./include/memory/memory.h
21 ./include/util/c_op.h
73 ./include/cpu/reg.h
16 ./include/cpu/cc.h
212 ./include/cpu/rtl.h
27 ./include/cpu/relop.h
41 ./include/cpu/exec.h
40 ./include/cpu/rtl-wrapper.h
116 ./include/cpu/decode.h
13 ./include/macro.h
35 ./include/common.h
8 ./include/nemu.h
14 ./include/device/mmio.h
10 ./include/device/port-io.h
31 ./src/misc/logo.c
141 ./src/monitor/monitor.c
66 ./src/monitor/cpu-exec.c
23 ./src/monitor/diff-test/ref.c
6 ./src/monitor/diff-test/diff-test.h
75 ./src/monitor/diff-test/diff-test.c
77 ./src/monitor/debug/watchpoint.c
253 ./src/monitor/debug/ui.c
321 ./src/monitor/debug/expr.c
28 ./src/memory/memory.c
43 ./src/cpu/reg.c
330 ./src/cpu/decode/decode.c
114 ./src/cpu/decode/modrm.c
76 ./src/cpu/exec/data-mov.c
216 ./src/cpu/exec/arith.c
8 ./src/cpu/exec/all-instr.h
254 ./src/cpu/exec/exec.c
62 ./src/cpu/exec/logic.c
9 ./src/cpu/exec/prefix.c
62 ./src/cpu/exec/system.c
49 ./src/cpu/exec/special.c
44 ./src/cpu/exec/control.c
32 ./src/cpu/exec/cc.c
20 ./src/cpu/exec/relop.c
13 ./src/cpu/intr.c
12 ./src/main.c
28 ./src/device/serial.c
28 ./src/device/timer.c
63 ./src/device/keyboard.c
87 ./src/device/io/port-io.c
74 ./src/device/io/mmio.c
102 ./src/device/device.c
40 ./src/device/vga.c
4247 total
```

和统计代码工具包统计结果基本一致

```
ladl0d ~/H/c/i/nemu (pa1)> sloc _
----- Result -----
      Physical : 4248
      Source   : 3227
      Comment   : 525
Single-line comment : 192
      Block comment : 333
      Mixed     : 241
Empty block comment : 0
      Empty     : 737
      To Do     : 62

Number of files read : 58
-----
```

## 2.3 PA2

### 2.3.1 总体设计

实现新指令, 在 `opcode_table` 中填写正确的译码, 执行函数以及操作数宽度, 用 RTL 实现正确的执行函数, 完成以下任务:

Task PA2.1: 在 NEMU 中运行第一个 c 程序 `dummy`

Task PA2.2: 实现更多指令, 在 NEMU 中运行所有 `cputest`

通过内存映射完成 `io` 功能, 完成以下任务:

Task PA2.3: 运行打字游戏, ppt 播放, 超级玛丽

### 2.3.2 详细设计

1) 首先实现未实现的 rtl:

取反

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

变量赋值

```
static inline void rtl_sext(rtlreg_t* dest, const rtlreg_t* src1, int width) { // 变量赋值
    // dest <- signext(src1[(width * 8 - 1) .. 0])
    //TODO();
    switch (width)
    {
    case 4:
        *dest = *src1;
        break;
    case 2:
        *dest = (int16_t)*src1;
        break;
    case 1:
        *dest = (int8_t)*src1;
        break;
    default:
        break;
    }
}
```

入栈

```
static inline void rtl_push(const rtlreg_t* src1) { //源操作数入栈
    // esp <- esp - 4
    // M[esp] <- src1
    cpu.esp -= 4;
    //printf("push 0x%x\n", *src1);
    vaddr_write((cpu.esp), (*src1), 4);
    //TODO();
}
```

出栈

```
static inline void rtl_pop(rtlreg_t* dest) { //出栈
    // dest <- M[esp]
    // esp <- esp + 4
    *dest = vaddr_read(cpu.esp, 4);
    //printf("pop 0x%x\n", *dest);
    cpu.esp += 4;
    //TODO();
}
```

更新标志位

```
static inline void rtl_update_ZF(const rtlreg_t* result, int width) {
    // eflags.ZF <- is_zero(result[width * 8 - 1 .. 0])
    //TODO();
    static uint32_t flags_zf_masks[] = {0, 0xff, 0xffff, 0, 0xffffffff};
    //printf("r:0x%x f:0x%x r&f:0x%x\n", (*result), flags_zf_masks[width], (*result) & flags_zf_masks[width]);
    cpu.EFLAGS.ZF = !((*result) & flags_zf_masks[width]);
}

static inline void rtl_update_SF(const rtlreg_t* result, int width) {
    // eflags.SF <- is_sign(result[width * 8 - 1 .. 0])
    //TODO();
    static uint32_t flags_sf_masks[] = {0, 0x80, 0x8000, 0, 0x80000000};
    cpu.EFLAGS.SF = ((*result) & flags_sf_masks[width]) != 0;
}

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

2) 参照 i386 手册，在 opcode table 中填写需要用到的指令并在 all-instr.h 头中添加需要实现的执行函数，分别在 exec 目录下的各个文件中实现这些执行函数。

3) 完成 klib 中的 printf 函数和 string.c 中的函数

4) 在上述过程中，极易出现译码填写或执行函数编写错误的情况，由于一开始没有完成 difftest 模块，每个 bug 调试起来都需要几个小时的时间，所以完成 difftest 模块，在 common.h 中定义 DIFFTEST，在 diff-test.c 中实现 difftest，代码如下：

```

// TODO: Check the registers state with the reference design.
// Set `nemu_state` to `NEMU_ABORT` if they are not the same.
//printf("test");
Assert(ref_r.eax == cpu.eax, "eax should be 0x%x, not 0x%x", ref_r.eax, cpu.eax);
Assert(ref_r.ebp == cpu.ebp, "ebp should be 0x%x, not 0x%x", ref_r.ebp, cpu.ebp);
Assert(ref_r.ebx == cpu.ebx, "ebx should be 0x%x, not 0x%x", ref_r.ebx, cpu.ebx);
Assert(ref_r.ecx == cpu.ecx, "ecx should be 0x%x, not 0x%x", ref_r.ecx, cpu.ecx);
Assert(ref_r.edi == cpu.edi, "edi should be 0x%x, not 0x%x", ref_r.edi, cpu.edi);
Assert(ref_r.edx == cpu.edx, "edx should be 0x%x, not 0x%x", ref_r.edx, cpu.edx);
Assert(ref_r.eip == cpu.eip, "eip should be 0x%x, not 0x%x", ref_r.eip, cpu.eip);
Assert(ref_r.esi == cpu.esi, "esi should be 0x%x, not 0x%x", ref_r.esi, cpu.esi);
Assert(ref_r.esp == cpu.esp, "esp should be 0x%x, not 0x%x", ref_r.esp, cpu.esp);
/*Assert(ref_r.EFLAGS.CF == cpu.EFLAGS.CF, "CF should be 0x%x, not 0x%x", ref_r.EFLAGS.CF, cpu.EFLAGS.CF);
Assert(ref_r.EFLAGS.OF == cpu.EFLAGS.OF, "OF should be 0x%x, not 0x%x", ref_r.EFLAGS.OF, cpu.EFLAGS.OF);
Assert(ref_r.EFLAGS.SF == cpu.EFLAGS.SF, "SF should be 0x%x, not 0x%x", ref_r.EFLAGS.SF, cpu.EFLAGS.SF);
Assert(ref_r.EFLAGS.ZF == cpu.EFLAGS.ZF, "ZF should be 0x%x, not 0x%x", ref_r.EFLAGS.ZF, cpu.EFLAGS.ZF);
*/

```

5) 完成 src/device 目录下的 input.c, timer.c, video.c 中的功能函数，将设备抽象为 IOE

```

size_t input_read(uintptr_t reg, void *buf, size_t size) {
    unsigned long long key_press;
    switch (reg) {
        case _DEVREG_INPUT_KBD: {
            _KbdReg *kbd = (_KbdReg *)buf;
            key_press = inl(I8042_PORT);
            //kbd->keydown = key_press;
            kbd->keycode = key_press;
            if(key_press != _KEY_NONE){
                kbd->keydown = !kbd->keydown;
            }
            return sizeof(_KbdReg);
        }
    }
    return 0;
}

```

```

        case _DEVREG_TIMER_UPTIME: {
            _UptimeReg *uptime = (_UptimeReg *)buf;
            unsigned long long now = inl(RTC_PORT); //获取当前时间
            uptime->hi = now >> 32;
            uptime->lo = now;
            return sizeof(_UptimeReg);
        }
    }
}

```

```

size_t video_read(uintptr_t reg, void *buf, size_t size) {
    switch (reg) {
        case _DEVREG_VIDEO_INFO: {
            _VideoInfoReg *info = (_VideoInfoReg *)buf;
            uint32_t screen;
            screen = inl(SCREEN_PORT); //0x26214700
            //printf("screen: 0x%x\n", screen);
            info->width = screen >> 16;
            info->height = screen & 0xffff;
            return sizeof(_VideoInfoReg);
        }
    }
    return 0;
}

```

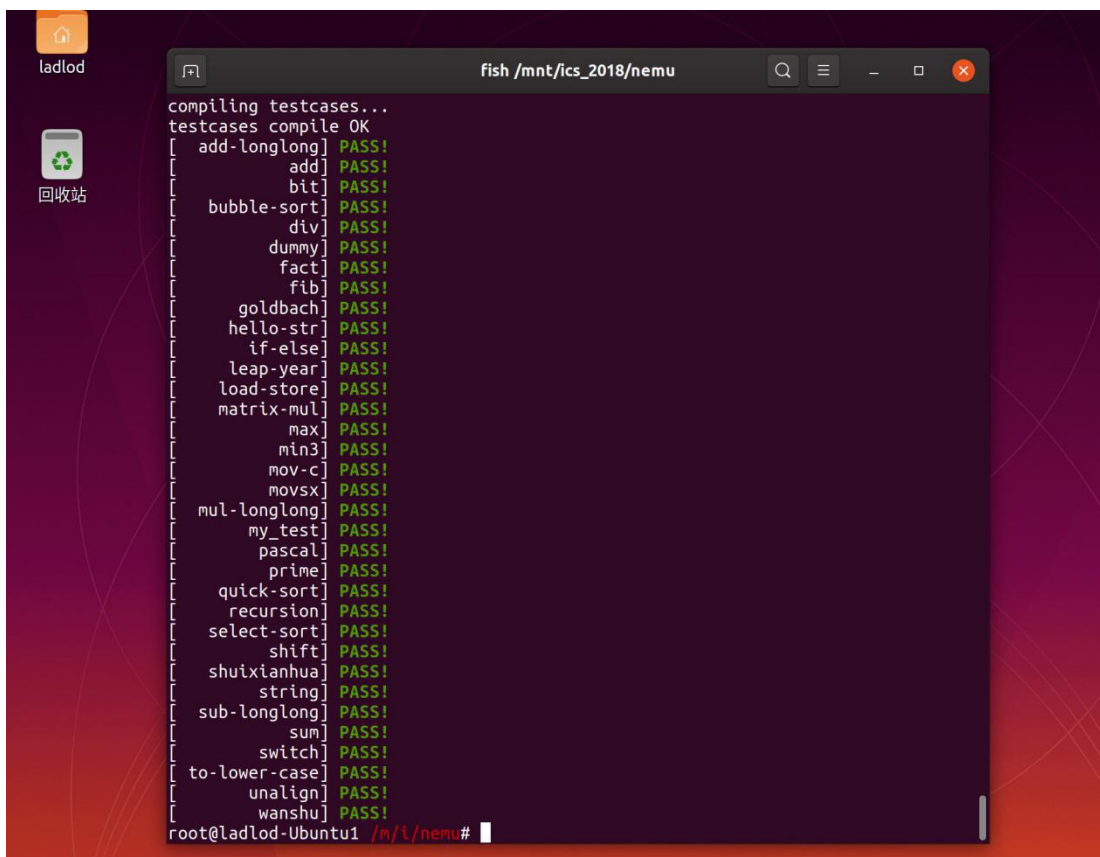
完成以上功能之后，即可调试运行 microbench 以及打字游戏等测试程序。



到此为止，PA2 的主体功能实现完毕。

### 2.3.3 运行结果

#### 1)Runall



```
compiling testcases...
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!
[my_test] 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!
root@ladlod-Ubuntu1 /m/i/nemu#
```

#### 2)Microbench



```
t/ics_2018/nexus-am/apps/microbench/build/microbench-x86-nemu.bin -d /mnt/ics_20
18/nemu/tools/qemu-diff/build/qemu-so
[src/monitor/monitor.c,72,load_img] The image is /mnt/ics_2018/nexus-am/apps/mic
robench/build/microbench-x86-nemu.bin
[src/monitor/monitor.c,33,welcome] Debug: OFF
[src/monitor/monitor.c,36,welcome] Build time: 18:13:31, Dec 17 2019
Welcome to NEMU!
For help, type "help"
[qsrt] Quick sort: * Passed.
min time: 1380 ms [399]
[queen] Queen placement: * Passed.
min time: 1638 ms [314]
[bf] Brainf**k interpreter: * Passed.
min time: 11772 ms [222]
[fib] Fibonacci number: * Passed.
min time: 20037 ms [142]
[sieve] Eratosthenes sieve: * Passed.
min time: 17033 ms [248]
[15pz] A* 15-puzzle search: * Passed.
min time: 3504 ms [165]
[dinic] Dinic's maxflow algorithm: * Passed.
min time: 2967 ms [456]
[lzip] Lzip compression: * Passed.
min time: 8532 ms [310]
[ssort] Suffix sort: * Passed.
min time: 1708 ms [346]
[md5] MD5 digest: * Passed.
min time: 15135 ms [129]
=====
MicroBench PASS 273 Marks
vs. 100000 Marks (i7-6700 @ 3.40GHz)
nemu: HIT GOOD TRAP at eip = 0x0010003a
[src/monitor/cpu-exec.c,22,monitor_statistic] total guest instructions = 1937789
102
make[1]: 离开目录"/mnt/ics_2018/nemu"
root@ladlod-Ubuntu1 /m/i/n/a/microbench#
```

The screenshot shows a terminal window with the following content:

```

make /mnt/ics_2018/nexus-am/apps/litenes
ld/build/typing-x86-nemu.bin
[src/monitor/monitor.c:33, welcome] Debug: OFF
[src/monitor/monitor.c:36, welcome] Build time: 18:13:31, Dec 17 2019
Welcome to NEMU!
For help, type "help"
[src/monitor/cpu-exec.c:22, monitor_statistic] total guest instructions = 2576762
88
make[1]: 离开目录"/mnt/ics_2018/nexus-am"
root@ladod-Ubuntu1:~#
root@ladod-Ubuntu1:~#
Building litenes [x86-nemu]
make[1]: 进入目录"/mnt/ics_2018/nexus-am"
make[2]: 进入目录"/mnt/ics_2018/nexus-am"
Building am [x86-nemu]
make[2]: "/mnt/ics_2018/nexus-am" 已经是最新。
make[2]: 离开目录"/mnt/ics_2018/nexus-am"
make[1]: 离开目录"/mnt/ics_2018/nexus-am"
make[1]: 进入目录"/mnt/ics_2018/nexus-am"
Building klib [x86-nemu]
make[1]: "/mnt/ics_2018/nexus-am" 已经是最新。
make[1]: 离开目录"/mnt/ics_2018/nexus-am"
make[1]: 进入目录"/mnt/ics_2018/nexus-am"
Building compiler-rt [x86-nemu]
make[1]: "/mnt/ics_2018/nexus-am" 已经是最新。
make[1]: 离开目录"/mnt/ics_2018/nexus-am"
make[1]: 进入目录"/mnt/ics_2018/nexus-am"
./build/nexus -b -l /mnt/ics_2018/nexus-am/apps/litenes/build/nemu-log.txt /mnt/ics_2018/nexus-am/apps/litenes/build/litenes-x86-nemu.bin -d /mnt/ics_2018/nexus-am/oools/nemu-diff/build/nemu-so
[src/monitor/monitor.c:72, load_img] The Image is /mnt/ics_2018/nexus-am/apps/litenes/build/litenes-x86-nemu.bin
[src/monitor/monitor.c:33, welcome] Debug: OFF
[src/monitor/monitor.c:36, welcome] Build time: 18:13:31, Dec 17 2019
Welcome to NEMU!
For help, type "help"

```

The emulator window shows the title screen of 'SUPER MARIO BROS.' with the following text:

```

WORLD 00000000 2:00 WORLD TIME 1:1
1. PLAYER GAME
2. PLAYER GAME
TOP- 00000000

```

The emulator window also shows the file path 'x86-nemu.a' in the bottom right corner.

PA2 全部测试样例通过。

### 2.3.4 问题解答

1) 编译与链接: `inline` 关键字表示建议编译器进行函数内联, 但并不强制内联, 非内联函数在多个 `c` 文件中引用会导致重定义, 所以不可以去掉 `inline`, 在 `inline` 前面加上 `static` 表示让该函数只在本文件中可以被识别, 在函数没有被 `inline` 内联时可以防止重定义, 代码健壮性更高。

2) 编译与链接:

1.一个

2.两个

`static` 关键字定义表示只有在本文件中可以被识别, 而 `volatile` 表示此处不被编译器优化, 所以重新声明的变量不会覆盖之前的 `dummy`。

3.出现重定义

```
Welcome to NEMU!
For help, type "help"
(nemu) q
root@ladlod-Ubuntu1 /m/i/nemu# make run
+ CC src/monitor/monitor.c
In file included from ./include/common.h:36,
                  from ./include/nemu.h:4,
                  from src/monitor/monitor.c:1:
./include/debug.h:8:21: error: redefinition of 'dummy'
volatile static int dummy = 0;
                  ^~~~~
In file included from ./include/nemu.h:4,
                  from src/monitor/monitor.c:1:
./include/common.h:15:21: note: previous definition of 'dummy' was here
volatile static int dummy = 0;
                  ^~~~~
make: *** [Makefile:33: build/obj/monitor/monitor.o] 错误 1
root@ladlod-Ubuntu1 /m/i/nemu#
```

3) 了解 Makefile:

- 1.首先依次读取变量“MAKEFILES”定义的 makefile 文件列表
- 2.读取工作目录下的 makefile 文件
- 3.一次读取工作目录下的 makefile 文件指定的 include 文件
- 4.查找重建所有已读的 makefile 文件的规则
- 5.初始化变量值并展开那些需要立即展开的变量和函数并根据预设条件确定执行分支
- 6.建立依赖关系表
- 7.执行 rules

## 2.4 PA3

### 2.4.1 总体设计

继续实现指令, 重新组织 `arch.h` 中的 `_Context` 结构体, 完成 `int` 事件分发, 实现系统调用, 完成以下任务。

PA 3.1: 实现自陷操作 `_yield()` 及其过程

PA 3.2: 实现用户程序的加载和系统调用

实现简易文件系统, 完成以下任务。

PA 3.3: 运行仙剑奇侠传并展示批处理系统 (未完成)



## 2.4.2 详细设计

重新组织 arch.h 中的 Context 结构体使之与 trap.S 中的上下文对应

```
struct _Context {
    uintptr_t esi, ebx, eax, eip, edx, err, eflags, ecx, cs, esp, edi, ebp;
    struct _Protect *prot;
    int      irq;
};
```

```
#----|-----entry-----|-----errorcode---|---irq id---|---handler---|
.globl vecsys;    vecsys:  pushl $0;    pushl $0x80; jmp asm_trap
.globl vectrap;   vectrap: pushl $0;    pushl $0x81; jmp asm_trap
.globl irq0;      irq0:    pushl $0;    pushl $32;  jmp asm_trap
.globl vecnull;   vecnull: pushl $0;    pushl $-1;  jmp asm_trap
```

在 cte.c 中完成 irq.c 实现对 0x80 系统调用和 0x81 自陷的事件分发

```
if (user_handler) {
    _Event ev;
    switch (tf->irq) {
        case 0x80: ev.event = _EVENT_SYSCALL; break;
        case 0x81: ev.event = _EVENT_YIELD; break;
        default: ev.event = _EVENT_ERROR; break;
    }
}
```

在 irq.c 中完成对自陷的处理

```
static _Context* do_event(_Event e, _Context* c) {
    switch (e.event) {
        case _EVENT_SYSCALL: do_syscall(c); break;
        case _EVENT_YIELD: printf("Yield %d\n", e.event); break;
        default: panic("Unhandled event ID = %d", e.event);
    }
    return NULL;
}
```

到此完成 PA3.1

实现 loader 函数，将从 0 开始，size 为 ramsize 的内存读取到 0x4000000 位置即可。

```
static uintptr_t loader(PCB *pcb, const char *filename) {
    ramdisk_read((void *)DEFAULT_ENTRY, 0, get_ramdisk_size());
    //Log("done ramdisk read");
    //TODO();
    //int fd = fs_open(filename, 0, 0);
    //printf("fd=%d\n", fd);
    //fs_read(fd, (void*)DEFAULT_ENTRY, fs_size(fd));
    //fs_close(fd);
    return DEFAULT_ENTRY;
}
```

运行结果即为从 ramdisk 中读取的 dummy 程序。

在 syscall.c 中实现 sys\_write 系统调用, 将从 0 开始 len 长的字符串写入到输出串口即可。

```
case SYS_write:
    c->GPRx = fs_write(a[1], (void*)a[2], a[3]);
    break;
```

```
switch(fd){
    case FD_STDIN:
        break;
    case FD_STDOUT:
    case FD_STDERR:
        file_table[fd].write(buf, 0, len);
        break;
```

修改 makefile, 从 ramdisk 中读取 helloworld 程序进调试。

PA2.2 到此结束。

实现 fs\_open, fs\_read, fs\_write 等函数, 修改 loader 读取文件。

```
static uintptr_t loader(PCB *pcb, const char *filename) {
    //amdisk_read((void *)DEFAULT_ENTRY, 0, get_ramdisk_size())
    //Log("done ramdisk read");
    //TODO();
    int fd = fs_open(filename, 0, 0);
    printf("fd=%d\n", fd);
    fs_read(fd, (void*)DEFAULT_ENTRY, fs_size(fd));
    fs_close(fd);

    return DEFAULT_ENTRY;
}
```

修改 makefile, 从 ramdisk 中读取 text 程序进行调试。

结果在 text 程序中出现以下错误

```
Welcome to NEMU!
For help, type "help"
[src/main.c,15,main] 'Hello World!' from Nanos-lite
[src/main.c,16,main] Build time: 20:47:48, Jan 3 2020
[src/ramdisk.c,29,init_ramdisk] ramdisk info: start = 1056704, end = 3
ze = 2467826 bytes
[src/device.c,36,init_device] Initializing devices...
[src/irq.c,15,init_irq] Initializing interrupt/exception handler...
[src/main.c,27,main] done fs
Open the file /bin/text
fd=9
assertion "fp" failed: file "text.c", line 6, function: main
nemu: HIT GOOD TRAP at eip = 0x0010007c
```

未找到解决方法。

### 2.4.3 运行结果

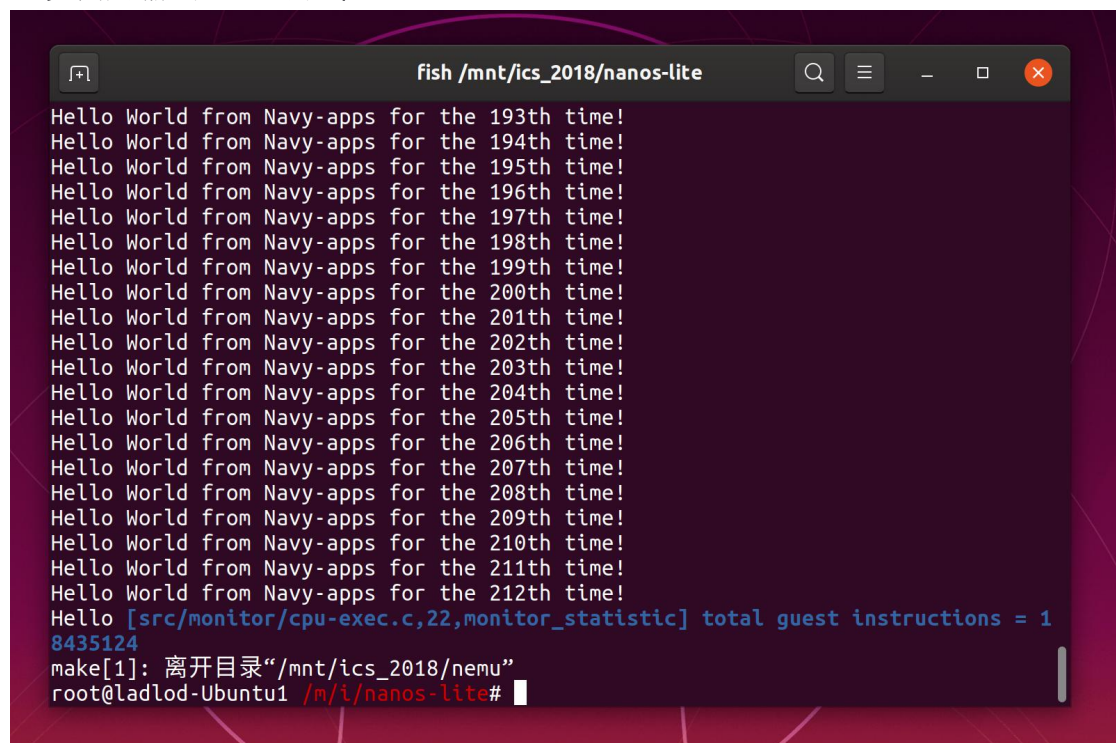
yield 测试:

```
Welcome to NEMU!  
For help, type "help"  
[src/main.c,15,main] 'Hello World!' from Nanos-lite  
[src/main.c,16,main] Build time: 20:41:06, Jan  3 2020  
[src/ramdisk.c,29,init_ramdisk] ramdisk info: start = 1056352, end = 3524178, si  
ze = 2467826 bytes  
[src/device.c,36,init_device] Initializing devices...  
[src/irq.c,15,init_irq] Initializing interrupt/exception handler...  
[src/main.c,27,main] done fs  
[src/main.c,30,main] done proc  
Yield 5  
[src/main.c,36,main] system panic: Should not reach here  
nemu: HIT BAD TRAP at eip = 0x0010007c
```

dummy 测试:

```
[src/monitor/monitor.c,36,welcome] Build time: 20:41:07, Jan  3 2020  
Welcome to NEMU!  
For help, type "help"  
[src/main.c,15,main] 'Hello World!' from Nanos-lite  
[src/main.c,16,main] Build time: 20:47:48, Jan  3 2020  
[src/ramdisk.c,29,init_ramdisk] ramdisk info: start = 1056608, end = 1070000, si  
ze = 13392 bytes  
[src/device.c,36,init_device] Initializing devices...  
[src/irq.c,15,init_irq] Initializing interrupt/exception handler...  
[src/main.c,27,main] done fs  
Yield 5  
nemu: HIT GOOD TRAP at eip = 0x0010007c  
  
[src/monitor/cpu-exec.c,22,monitor_statistic] total guest instructions = 207198  
make[1]: 离开目录“/mnt/ics_2018/nemu”  
root@ladlod-Ubuntu1 /m/i/nanos-lite#
```

系统调用输出 hello 测试:



```
fish /mnt/ics_2018/nanos-lite  
Hello World from Navy-apps for the 193th time!  
Hello World from Navy-apps for the 194th time!  
Hello World from Navy-apps for the 195th time!  
Hello World from Navy-apps for the 196th time!  
Hello World from Navy-apps for the 197th time!  
Hello World from Navy-apps for the 198th time!  
Hello World from Navy-apps for the 199th time!  
Hello World from Navy-apps for the 200th time!  
Hello World from Navy-apps for the 201th time!  
Hello World from Navy-apps for the 202th time!  
Hello World from Navy-apps for the 203th time!  
Hello World from Navy-apps for the 204th time!  
Hello World from Navy-apps for the 205th time!  
Hello World from Navy-apps for the 206th time!  
Hello World from Navy-apps for the 207th time!  
Hello World from Navy-apps for the 208th time!  
Hello World from Navy-apps for the 209th time!  
Hello World from Navy-apps for the 210th time!  
Hello World from Navy-apps for the 211th time!  
Hello World from Navy-apps for the 212th time!  
Hello [src/monitor/cpu-exec.c,22,monitor_statistic] total guest instructions = 1  
8435124  
make[1]: 离开目录“/mnt/ics_2018/nemu”  
root@ladlod-Ubuntu1 /m/i/nanos-lite#
```

后续实验没有完成。



## 3 设计总结与心得

### 3.1 课设总结

本次课程设计总体难度较高，耗时较长，编写过程中遇到过如下问题：

在 PA1 的编写过程中，主要难点在于运算器部分，运算器主要采用的想法是递归，在递归之前需要先对字符串进行正则匹配处理，在递归时需要对运算符进行括号判断，主运算符选取。在正则匹配过程中，需要注意的是\*可以代表乘号也可以代表指针，-可以代表减号也可以代表负号，需要单独处理，处理的想法是如果这两个运算符前面是数字则为乘号或减号。但实际处理过程中需要考虑到，右括号，寄存器，也可以处理为数字，在实验开始时未考虑到这一点，导致实验出错。另外，在正则匹配过程判断>号和>=号时，需要注意优先判断>=号，如果先判断>号则会将>=号判断为一个>号和一个未知类型的=号，导致程序出错。在括号是否匹配判断过程的主要想法是判断左括号个数是否与右括号相等以及过程中是否出现过右括号数量大于左括号的情况，这种想法没有考虑到() + ()的情况，会将第一个括号和最后一个括号判断为匹配，所以将判断条件更改为，处理过程中左括号至少比右括号多一个，直到最后一个再跳出判断。

在 PA2 的编写过程中，主要难点在于 opcode table 的填写和 rtl 中关于符号位寄存器更新的实现。opcode table 的填写比较耗费耐心，尤其注意操作数位数的选择，因为在选错操作数位数时，可能当时的一条指令没有出现问题，在后续过程出现了问题导致很难找出 bug。rtl 中关于负号位更新主要难点在于进位标志位和溢出标志位的判断，溢出主要针对有符号数，进位主要针对无符号数，在有符号数的溢出判断中，尤其需要注意对 0 的判断，在进行 0 - 128 运算时，结果为正数，应该发生了溢出。在调试过程中，一开始没有完成 difftest 部分，出现 opcode table 填写错误的 bug 时很难找出 bug，甚至出现了一个 bug 找一天的情况，后来实现了 difftest，大幅度提高了改代码的效率。在调试死循环过程中，我在 jmp 和 j 指令中将跳转信息打印出来，也对调试过程有很大帮助。

在 PA3 的编写过程中，主要难点在于对代码框架的理解，理解了代码结构之后，编写代码的过程进行的很快。但在读 ramdisk 时因为编译器的原因 disk 过大，导致实验停滞了几天才继续进行。在后续 PA3.3 的文件系统调试 text 的过程中，出现了函数 fopen() 返回 NULL 的问题，经过测试，问题出现在 fopen.c 中，\_fopen\_r() 函数的一个判断条件，\_sfp(ptr) == NULL，而 ptr 为框架代码中的 \_REENT，所以怀疑问题依然出现在 ramdisk 的生成过程中，问题没办法解决，实验停滞在了这一步。

## 3.2 课设心得

本次实验是对我们整个大学四年所学知识的综合性应用考察，考察了我们对汇编语言，编译原理，组成原理，操作系统，还有 c 语言的掌握理解程度，是我们第一次进行综合性的大型实验。实验材料来自南京大学，实验进度安排科学合理，指导手册清晰，老师的指导也十分详细负责，实验过程非常愉快。我在实验的前半阶段进度较快，而在完成 PA2 之后有了些懈怠，导致 PA3 最终没有完成，比较遗憾。

总的来说，本次实验是对我们四年所学的综合性的检验，也是对我们动手能力，学习能力的检验，在实验中使用到的工具比如 vim, git, make 都会对我们未来的工作生活有巨大的帮助。

## 参考文献

- [1] DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第 4 版).北京: 机械工业出版社.
- [2] David Money Harris(美).数字设计和计算机体系结构 (第二版) . 机械工业出版社
- [3] 秦磊华, 吴非, 莫正坤.计算机组成原理. 北京: 清华大学出版社, 2011 年.
- [4] 袁春风编著. 计算机组成与系统结构. 北京: 清华大学出版社, 2011 年.
- [5] 张晨曦, 王志英. 计算机系统结构. 高等教育出版社, 2008 年.