

南京航空航天大学《计算机组成原理II课程设计》报告

- 姓名：郑伟林
- 班级：1619303
- 学号：061920125
- 报告阶段：PA1.1
- 完成日期：2021.3.28
- 本次实验，我完成了所有内容。

目录

南京航空航天大学《计算机组成原理II课程设计》报告

目录

思考题

实验内容

实现寄存器结构体(30分)

实现单步执行(15 分)

修改一次打印步数上限(10 分)

实现打印寄存器功能(15 分)

实现扫描内存功能(15 分)

实现扫描内存字节单位显示(15 分)

遇到的问题及解决办法

实验心得

其他备注

思考题

1. 存放的是什么？

PC中存放的是下一条指令的地址，这样在一条指令执行完后下一条指令可以继续被运行，PC再+1到下一条指令，程序就可以自动执行一段指令序列。而当前指令是存放在IR中。

2. 贵圈真乱

```
+-----+
| "Hello world" program |
+-----+
+ Micro operating system +
+-----+
| NEMU |
+-----+
| GNU/Linux |
+-----+
| VirtualBox (Simulated Hardware) |
+-----+
| Host Operating System |
+-----+
| Computer hardware |
+-----+
```

3. 虚拟机和模拟器的区别

我觉得虚拟机实际上是直接使用Host主机的硬件系统来运行操作系统，而NEMU是用程序构建了一系列硬件系统从而再运行操作系统，是从组成层开始构建。

4. 从哪开始阅读代码呢？

应该找到目录下的 `main.c` 文件开始执行其主函数 `main`。

5. 究竟要执行多久？

`cpu_exec()` 函数的形参是 `uint64_t` 其本质是 `unsigned long long` 类型，是无符号的整型，如果传给它-1，则以-1的补码形式是全1，截去符号位后为 `unsigned long long` 的最大值，因此函数可以执行最大指令数，即将指令全部运行。

6. 谁来指示程序的结束？

`main()` 函数的返回语句只是将函数本身结束并返回一个值，该值会供操作系统判断程序是否正常结束，而真正让程序结束的是 `exit()` 函数，其在 `main()` 结束后会隐式调用结束程序。

7. 为什么会这样？

数据是按字节存储的，在读取时，x86是从低位到高位的，所以一次性打印4字节和以1字节打印4次时，顺序看起来是不一样的。如 `0x00ab1234`，以1字节打印时，先取低位 `34`，再 `12`，再 `ab`，再 `00`。

8. Git Log截图

```
zhengweilin@debian: ~/ics2021/nemu

commit 28d06ba6fcd12eaa18e9854ceb35837510c834ce
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 19:56:18 2021 +0800

    add ui.c/cmd_x

commit 050d3047d01013e7955264066dcc9ffdeec923ba
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 19:53:51 2021 +0800

    add ui.c/cmd_x

commit 677c0aa9dd07bda4d7b9fae65fd22055a7fbae2a
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 19:47:10 2021 +0800

    modified ui.c/cmd_info

commit 1c917670c7fd4ecdaa26ff95d77d7011e7a975c6
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 18:02:45 2021 +0800

    modified cpu-exec.c/MAX_INSTR_TO_PRINT to 1000000

commit 0c8f810131146ca2a57451abaf490b3f32aff3d7
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 17:20:59 2021 +0800

    modified ui.c/cmd_info

commit 35378afcc444c8e6dcd358a7379c52ffebb17679
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 17:12:49 2021 +0800

    modified: ui.c/cmd_info

commit 1b14c867588f6f9f4394e91eeb9843df81a8eb62
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 17:07:35 2021 +0800

    add cmd_info funtion

commit 4e00bc3d7afe90c212ef51eb46ed984d61b622c4 (pa0)
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sun Mar 21 15:01:33 2021 +0800

    before starting pal

:
```

9. Git Branch截图

```
zhengweilin@debian: ~/ics2021/nemu

    add ui.c/cmd_x

commit 050d3047d01013e7955264066dcc9ffdeec923ba
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 19:53:51 2021 +0800

    add ui.c/cmd_x

commit 677c0aa9dd07bda4d7b9fae65fd22055a7fbae2a
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 19:47:10 2021 +0800

    modified ui.c/cmd_info

commit 1c917670c7fd4ecdaa26ff95d77d7011e7a975c6
Author: 061920125-Zheng Weilin <2529039819@qq.com>
Date: Sat Mar 27 18:02:45 2021 +0800

zhengweilin@debian:~/ics2021/nemu$ git branch
  master
* pa0
* pa1
zhengweilin@debian:~/ics2021/nemu$
```

10. 远程git仓库提交截图

吃 吃花椒的热心

个人主页

仓库

1

Pull Requests

0

任务

0

代码片段

0

我 Star 的仓库

0

企业

暂未加入任何企业

[了解 Gitee 企业版](#) 或 [创建免费企业版](#)

动态

所有动态

今天

2021-03-28

吃花椒的热心

推送了新的分支 pa1 到 吃花椒的热心/ics2021

3 小时前

2021-03-07

吃花椒的热心

推送了新的分支 pa0 到 吃花椒的热心/ics2021

21 天前

推送了新的分支 master 到 吃花椒的热心/ics2021

21 天前

创建了 吃花椒的热心/ics2021

21 天前

加载更多

实验内容

实现寄存器结构体(30分)

根据寄存器结构，我们应该可以通过 `cpu.gpr[0]._32` 访问 `eax`，而 `cpu.gpr[0]._16` 访问 `eax` 的低16位，`cpu.gpr[0]._8[0]` 访问 `eax` 的低8位，因为三者类型不同，因此我们可以考虑使用union结构来包含这些结构。

```
typedef union {
    union {
        uint32_t _32;
        uint16_t _16;
        uint8_t _8[2];
    } gpr[8];

    struct{
        rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;

        vaddr_t eip;
    };
} CPU_state;
```

```
zhengweilin@debian: ~/ics2021/nemu
zhengweilin@debian:~/ics2021/nemu$ make run
+ CC src/memory/memory.c
+ CC src/cpu/exec/arith.c
+ CC src/cpu/exec/logic.c
+ CC src/cpu/exec/exec.c
+ CC src/cpu/exec/control.c
+ CC src/cpu/exec/special.c
+ CC src/cpu/exec/prefix.c
+ CC src/cpu/exec/cc.c
+ CC src/cpu/exec/data-mov.c
+ CC src/cpu/exec/system.c
+ CC src/cpu/decode/modrm.c
+ CC src/cpu/decode/decode.c
+ CC src/cpu/intr.c
+ CC src/cpu/reg.c
+ CC src/main.c
+ CC src/misc/logo.c
+ CC src/monitor/cpu-exec.c
+ CC src/monitor/monitor.c
+ CC src/monitor/diff-test/diff-test.c
+ CC src/monitor/diff-test/protocol.c
+ CC src/monitor/diff-test/gdb-host.c
+ CC src/monitor/debug/watchpoint.c
+ CC src/monitor/debug/ui.c
+ CC src/monitor/debug/expr.c
+ CC src/device/serial.c
+ CC src/device/vga.c
+ CC src/device/device.c
+ CC src/device/keyboard.c
+ CC src/device/timer.c
+ CC src/device/io/port-io.c
+ CC src/device/io/mmio.c
+ LD build/nemu
./build/nemu -l ./build/nemu-log.txt
[src/monitor/monitor.c,47,load_default_img] No image is given. Use the default build-in image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 17:22:59, Mar 21 2021
For help, type "help"
(nemu) █
```

实现单步执行(15 分)

实现函数为 `static int cmd_si(char *args)`，首先对缺省时进行判断，执行一步，再对负数进行判断，如果是-1的话执行 `cpu_exec(-1)`，接下来才是正数时，通过分解字符串 `args` 从低位到高位将其转为整数 `n`，执行 `cpu_exec(n)`。

最后记得再 `cmd_table` 中添加相关字符串及函数名。

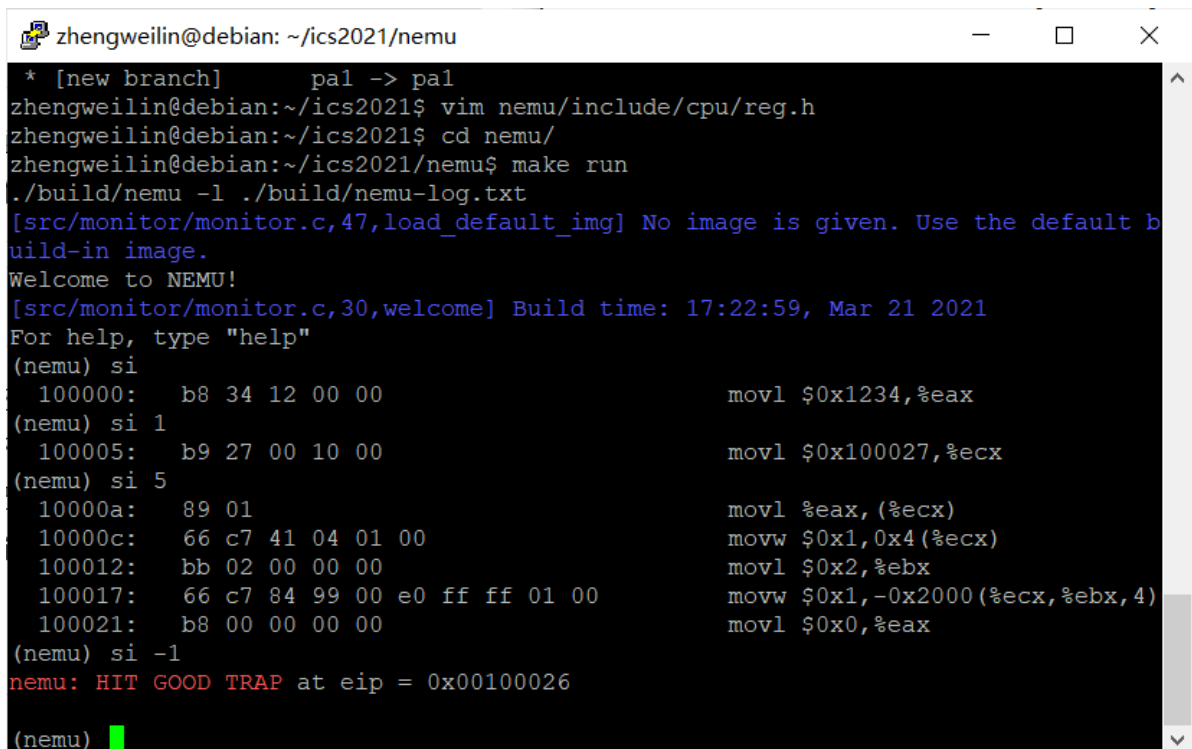
```
static int cmd_si(char *args){
    if (args == NULL )
    {
        cpu_exec(1);
        return 0;
    }

    if (args[0] == '-')
    {
        if (args[1] == '1' && strlen(args)==2)
        {
            cpu_exec(-1);
        }
        else
            printf("Unknown comand si '%s'\n",args);
    }
}
```

```

}
else
{
    for(int i=0;i<strlen(args);i++)
    {
        if (args[i]<'0' || args[i]>'9')
        {
            printf("Unknown command si '%s'\n",args);
            return 0;
        }
    }
    int c = 1;
    int n = 0;
    for (int i = strlen(args) - 1; i >= 0; i--)
    {
        n += (args[i] - '0') * c;
        c *= 10;
    }
    cpu_exec(n);
}
return 0;
}
static int cmd_q(char *args) {
    return -1;
}
}

```



```

zhengweilin@debian: ~/ics2021/nemu
* [new branch]    pal -> pal
zhengweilin@debian:~/ics2021$ vim nemu/include/cpu/reg.h
zhengweilin@debian:~/ics2021$ cd nemu/
zhengweilin@debian:~/ics2021/nemu$ make run
./build/nemu -l ./build/nemu-log.txt
[src/monitor/monitor.c,47,load_default_img] No image is given. Use the default build-in image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 17:22:59, Mar 21 2021
For help, type "help"
(nemu) si
100000:      b8 34 12 00 00      movl $0x1234,%eax
(nemu) si 1
100005:      b9 27 00 10 00      movl $0x100027,%ecx
(nemu) si 5
10000a:      89 01      movl %eax, (%ecx)
10000c:      66 c7 41 04 01 00      movw $0x1,0x4(%ecx)
100012:      bb 02 00 00 00      movl $0x2,%ebx
100017:      66 c7 84 99 00 e0 ff ff 01 00      movw $0x1,-0x2000(%ecx,%ebx,4)
100021:      b8 00 00 00 00      movl $0x0,%eax
(nemu) si -1
nemu: HIT GOOD TRAP at eip = 0x00100026
(nemu)

```

修改一次打印步数上限(10 分)

该问题是在 `/nemu/src/monitor/cpu-exe.c/cpu_exec(unit64_t n)` 函数中，会对 `n` 进行判断，如果 `n > MAX_INSERT_TO_PRINT` 则将打印标记记为 `false` 即不打印，所以我们可以通过修改 `MAX_INSERT_TO_PRINT`，将其改为很大的数或者 -1 来扩大其打印范围。

```
#define MAX_INSTR_TO_PRINT 1000000
```

```
zhengweilin@debian: ~/ics2021/nemu
100021:  b8 00 00 00 00          movl $0x0,%eax
(nemu) si -1
nemu: HIT GOOD TRAP at eip = 0x00100026

(nemu) q
zhengweilin@debian:~/ics2021/nemu$ make run
./build/nemu -l ./build/nemu-log.txt
[src/monitor/monitor.c,47,load_default_img] No image is given. Use the default build-in image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 17:22:59, Mar 21 2021
For help, type "help"
(nemu) si 100
100000:  b8 34 12 00 00          movl $0x1234,%eax
100005:  b9 27 00 10 00          movl $0x100027,%ecx
10000a:  89 01                   movl %eax, (%ecx)
10000c:  66 c7 41 04 01 00       movw $0x1,0x4(%ecx)
100012:  bb 02 00 00 00          movl $0x2,%ebx
100017:  66 c7 84 99 00 e0 ff ff 01 00  movw $0x1,-0x2000(%ecx,%ebx,4)
100021:  b8 00 00 00 00          movl $0x0,%eax
nemu: HIT GOOD TRAP at eip = 0x00100026

100026:  d6                     nemu trap (eax = 0)
(nemu) █
```

实现打印寄存器功能(15 分)

实现函数为 `static int cmd_info(char *args)`，打印寄存器的核心代码在于三个循环，每个循环中都打印相应寄存器的值，分别是整个寄存器的值、低十六位的值和低八位的值。直接访问 `cpu.gpr[i]._32`、`cpu.gpr[i]._16`、`cpu.gpr[i]._8[0]` 和 `cpu.gpr[i]._8[1]`。

最后记得在 `cmd_table` 中添加相关代码。

```
static int cmd_info(char *args)
{
    // 分割字符
    if (strlen(args) != 1 || ((args[0] != 'r') && (args[0] != 'w')))
    {
        printf("Unknown command info '%s'\n", args);
        return 0;
    }
    // 判断子命令是否是r
    if (args[0] == 'r')
    {
        // 依次打印所有寄存器
        // 这里给个例子：打印出 eax 寄存器的值
        for (int i = 0; i < 8; i++)
            printf("%s:\t0x%08x\t%d\n", regsl[i], cpu.gpr[i]._32, cpu.gpr[i]._32);
        for (int i = 0; i < 8; i++)
            printf("%s:\t0x%04x\t%d\n", regsw[i], cpu.gpr[i]._16, cpu.gpr[i]._16);
        for (int i = 0; i < 4; i++)
            printf("%s:\t0x%02x\t%d\t%s:\t0x%02x\t%d\n",
                regsb[i],
                cpu.gpr[i]._8[0], cpu.gpr[i]._8[0], regsb[i+4], cpu.gpr[i]._8[1], cpu.gpr[i]._8[1]);

        return 0;
    }
    else if (args[0] == 'w')
    {
        // 这里我们会在 PA1.3 中实现
    }
}
```

```

}
return 0;
}

```

```

zhengweilin@debian: ~/ics2021/nemu
100026:  d6                                nemu trap (eax = 0)
(nemu) info r
eax:  0x00000000      0
ecx:  0x00100027     1048615
edx:  0x6625fbf7     1713765367
ebx:  0x00000002      2
esp:  0x04d00844     80742468
ebp:  0x34f60a59     888539737
esi:  0x16235df8     371416568
edi:  0x0ea10678     245433976
ax:   0x0000      0
cx:   0x0027     39
dx:   0xfbf7    64503
bx:   0x0002      2
sp:   0x0844    2116
bp:   0x0a59    2649
si:   0x5df8    24056
di:   0x0678    1656
al:   0x00      0      ah:   0x00      0
cl:   0x27     39      ch:   0x00      0
dl:   0xf7     247     dh:   0xfb     251
bl:   0x02      2      bh:   0x00      0
(nemu)

```

实现扫描内存功能(15 分)

实现函数为 `static int cmd_x(char *args)`，首先分割字符串，利用 `strtok` 函数分解到 `num` 和 `addr_s` 中，接着利用字符串操作从低位将字符转为数值到 `n` 和 `addr` 中，接着循环 `n` 次调用 `vaddr_read(addr,4)` 输出相应内存里的信息。

最后记得在 `cmd_table` 中添加相关代码。

```

static int cmd_x(char *args){
    //分割字符串，得到起始位置 and 要读取的次数
    char *num = strtok(NULL, " ");
    char *addr_s = strtok(NULL, " ");
    int n = 0, c = 1;
    for (int i = strlen(num) - 1; i >= 0; i--)
    {
        n += (num[i] - '0') * c;
        c *= 10;
    }

    uint32_t dword, temp, addr = 0;
    c = 1;
    for (int i = strlen(addr_s) - 1; addr_s[i] != 'x'; i--)
    {
        if (addr_s[i] >= '0' && addr_s[i] <= '9')
        {
            addr += (addr_s[i] - '0') * c;
        }
        else if (addr_s[i] >= 'a' && addr_s[i] <= 'f')
        {
            addr += (addr_s[i] - 'a' + 10) * c;
        }
    }
}

```



```

    c*=16;
}

//循环使用 vaddr_read 函数来读取内存
printf("Address      Dword block ... Byte sequence\n");
for(int i=0;i<n;i++){
    uint16_t byte_seq[4]={0};
    dword = vaddr_read(addr,4);    //如何调用，怎么传递参数，请阅读代码
    temp=dword;
    for (int j = 0; j < 4; j++){
        byte_seq[j]=temp % 256;
        temp/=256;
    }

    //每次循环将读取到的数据用 printf 打印出来
    printf("0x%08x  0x%08x ... %02x %02x %02x %02x\n",
        addr,dword,byte_seq[0],byte_seq[1],byte_seq[2],byte_seq[3]);    //如果你不
    //知道应该打印什么，可以参考参考输出形式
    addr+=4;
}
return 0;
}

```

```

zhengweilin@debian: ~/ics2021/nemu
di:      0x0678  1656
al:      0x00   0      ah:      0x00   0
cl:      0x27   39      ch:      0x00   0
dl:      0xf7   247     dh:      0xfb   251
bl:      0x02   2       bh:      0x00   0
(nemu) x 4 0x100000
Address   Dword block ... Byte sequence
0x00100000 0x001234b8 ... b8 34 12 00
0x00100004 0x0027b900 ... 00 b9 27 00
0x00100008 0x01890010 ... 10 00 89 01
0x0010000c 0x0441c766 ... 66 c7 41 04
(nemu) x 10 0x100000
Address   Dword block ... Byte sequence
0x00100000 0x001234b8 ... b8 34 12 00
0x00100004 0x0027b900 ... 00 b9 27 00
0x00100008 0x01890010 ... 10 00 89 01
0x0010000c 0x0441c766 ... 66 c7 41 04
0x00100010 0x02bb0001 ... 01 00 bb 02
0x00100014 0x66000000 ... 00 00 00 66
0x00100018 0x009984c7 ... c7 84 99 00
0x0010001c 0x01ffffe0 ... e0 ff ff 01
0x00100020 0x0000b800 ... 00 b8 00 00
0x00100024 0x34d60000 ... 00 00 d6 34
(nemu)

```

实现扫描内存字节单位显示(15 分)

在 `static int cmd_x(char *args)` 中我们可以先将 `dword` 复制一份到 `temp` 中，再对 `temp` 进行分解，要按一字节一字节分解只需将 `temp` 不断 `%256`，再 `/256`，存到一个数组里，最后打印出来即可。

```
temp=dword;
for (int j = 0; j < 4; j++)
{
    byte_seq[j]=temp % 256;
    temp/=256;
}
printf("0x%08x 0x%08x ... %02x %02x %02x %02x\n",
    addr,dword,byte_seq[0],byte_seq[1],byte_seq[2],byte_seq[3]);
```

.....

遇到的问题及解决办法

1. 遇到问题：在完成 `cmd_x` 命令时输出和预想的不一樣。

解决方案：通过对各个变量分析定位，最后确定时 `strtok` 函数截取字符串时有问题，再去查阅相关资料后，才明白 `strtok` 的正确用法，在第一次只需 `strtok` 后，之后再用 `strtok` 不需再传原字符串，只需传 `NULL`。

2. 遇到问题：git操作时没有可提交的东西。

解决方案：通过网络搜寻和询问同学之后，才知道在对代码修改过后，需要立马 `git add .` 和 `git commit`，否则退回上一层文件夹或运行操作后，这次修改的信息会被丢弃掉。因此在改完代码后需马上进行git操作进行记录。

3. 无

实验心得

做完了PA1.1，本阶段主要是完善寄存器结构和完成 `nemu` 的调试方法的函数。通过这次课设内容，我对寄存器的结构有了进一步了解，学会了利用结构体和联合嵌套来模拟其结构。同时，阅读了一部分的 `nemu` 源码，见识到了一些C语言的特别的用法。在编写代码时，对 `strtok` 函数的用法有了进一步了解。在整个实验过程中，不断利用git来提交保存修改，对git的使用更加熟练。

其他备注

无