南京航空航天大学《计算机组成原理Ⅱ课程设计》报告

- 姓名：马睿
- 班级：1619304
- 学号：161930131
- 报告阶段：PA1.1
- 完成日期：2021.3.22
- 本次实验，我完成了所有内容。

目录

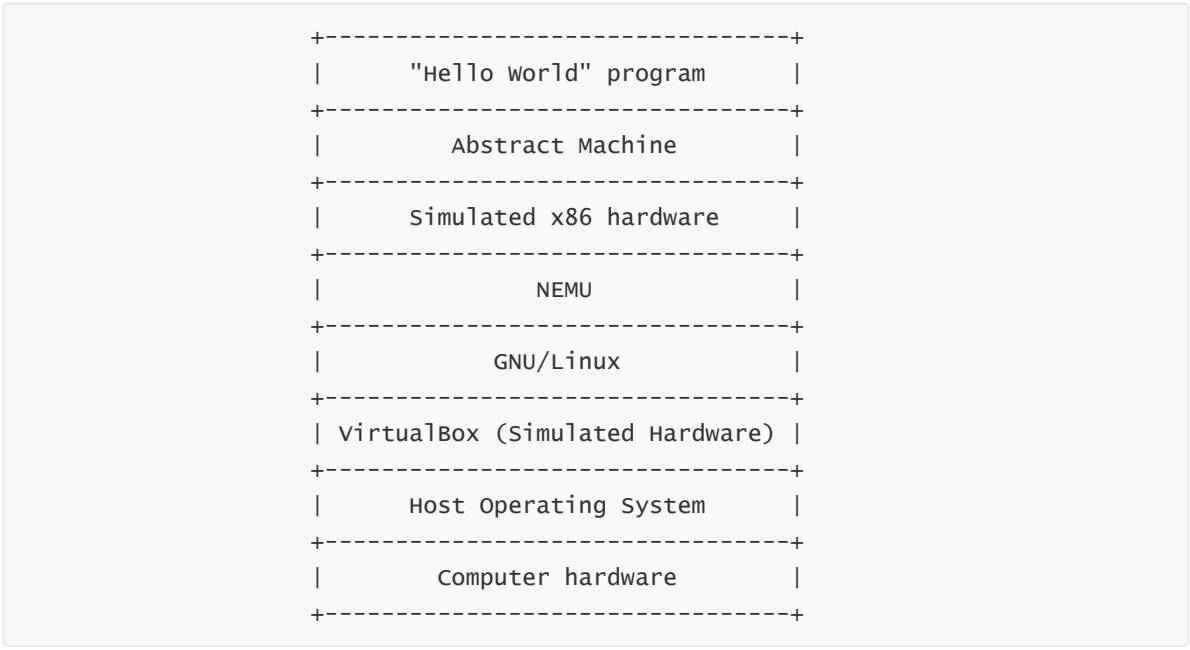
思考题

一、存放的是什么?

为什么是存放指令的存放地址而不是指令本身呢?

存放指令的地址可以通过当前指令地址和当前指令长度得到下一条指令地址，从而一直执行下去。否则不知道下一条指令是哪一条。

二、贵圈真乱



三、虚拟机和模拟器的区别

模拟器其实也是虚拟机的一种，和虚拟机的差别在于，模拟器使用固化的操作系统，不能随意更改操作系统，且结构简单，不能随意添加虚拟硬件设备。

四、从哪开始阅读代码呢

从`main`函数开始执行。

应该从`nemu/src/main.c`开始阅读。

五、究竟要执行多久

`void cpu_exec(uint64_t n)`的参数是 64位无符号数。当传入的参数为`-1`时，它将被解释为64位的无符号整数，64位无符号整数的最大值。

同时，该函数中让`cpu`不断执行指令的是一个for循环，`for (; n>0 ;n--)`，n非常大时可以保证`cpu`持续不断地执行指令，直到执行完所有的指令，而避免指令还没执行完就提前停止执行。

六、谁来指示程序的结束?

`CRT`（显示器）在收到`main`函数返回值后，将这个值传递了`Win32API ExitProcess`，当程序执行这个`API`时，操作系统将终止应用程序并释放整个进程资源，然后进程死亡。

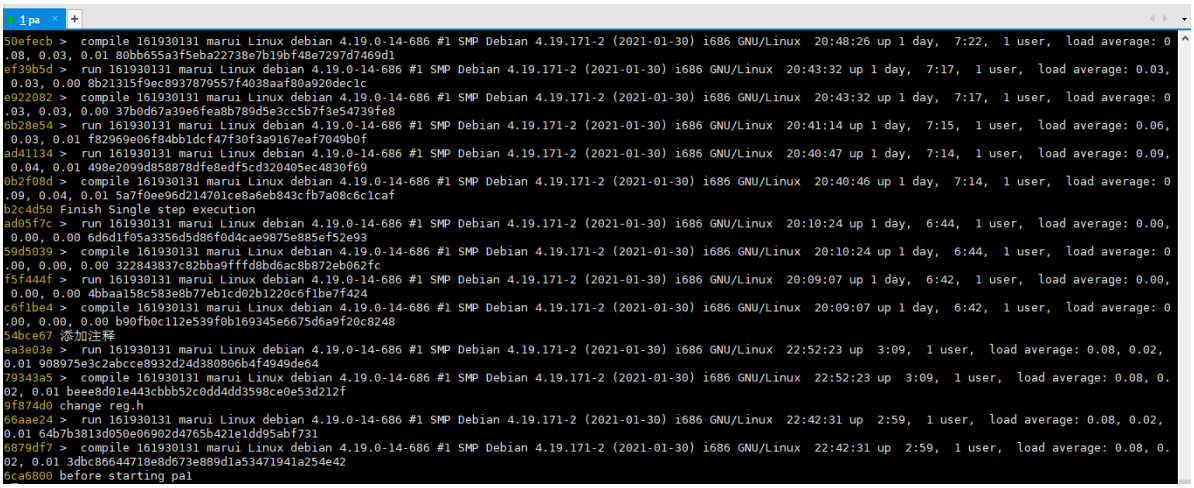
七、为什么会这样?

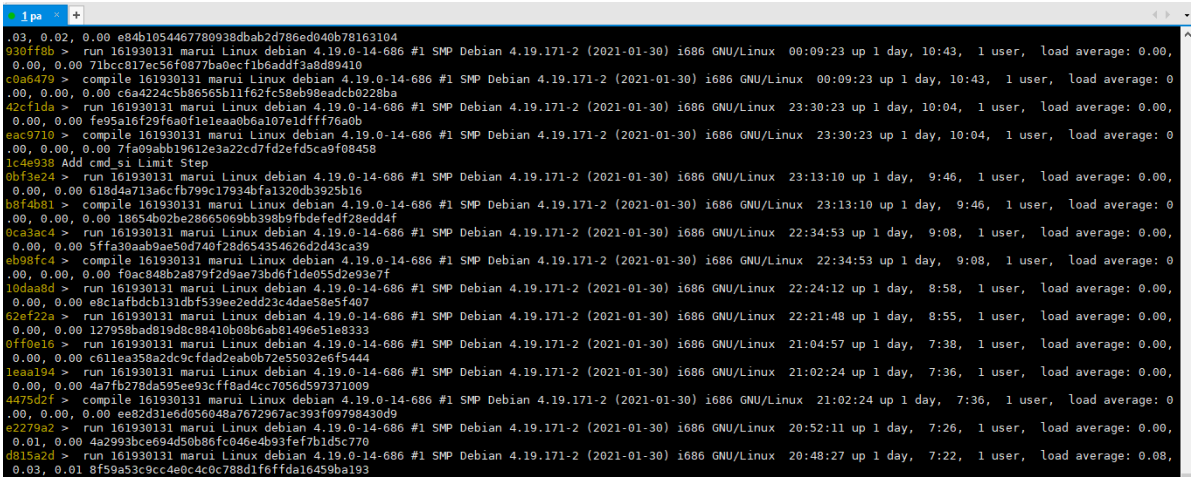
数据的排列顺序分为大端法和小端法。大端法将最高有效字节数据存储在小地址单元中，最低有效字节存放在大地址单元中；小端法相反。

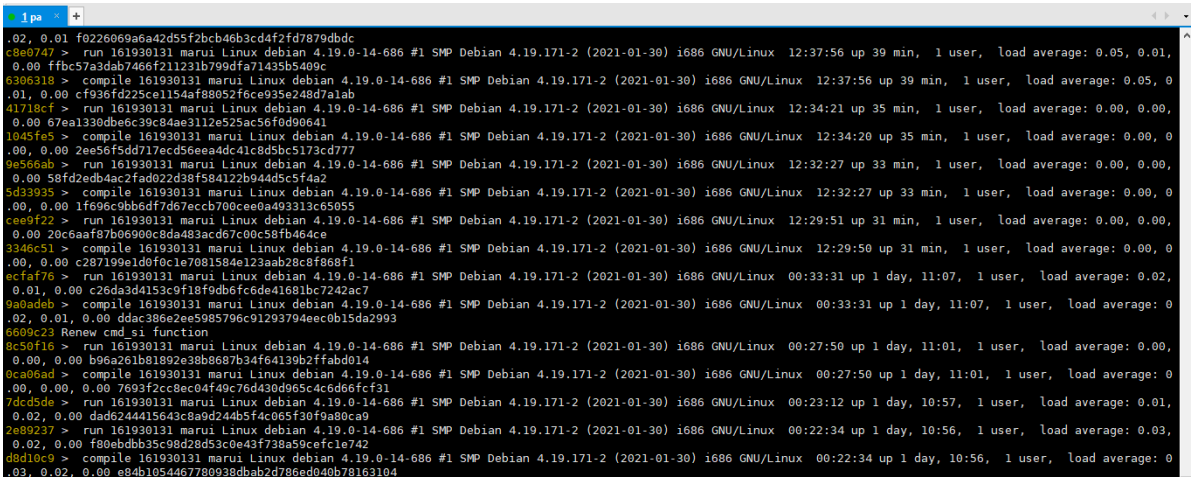
`Linux`采用的是小端法，而将字节进行拆分操作时，是先从低地址位进行获取，所以 4 字节为单位打印和以 1 字节为单位打印时相比，顺序会不一样。

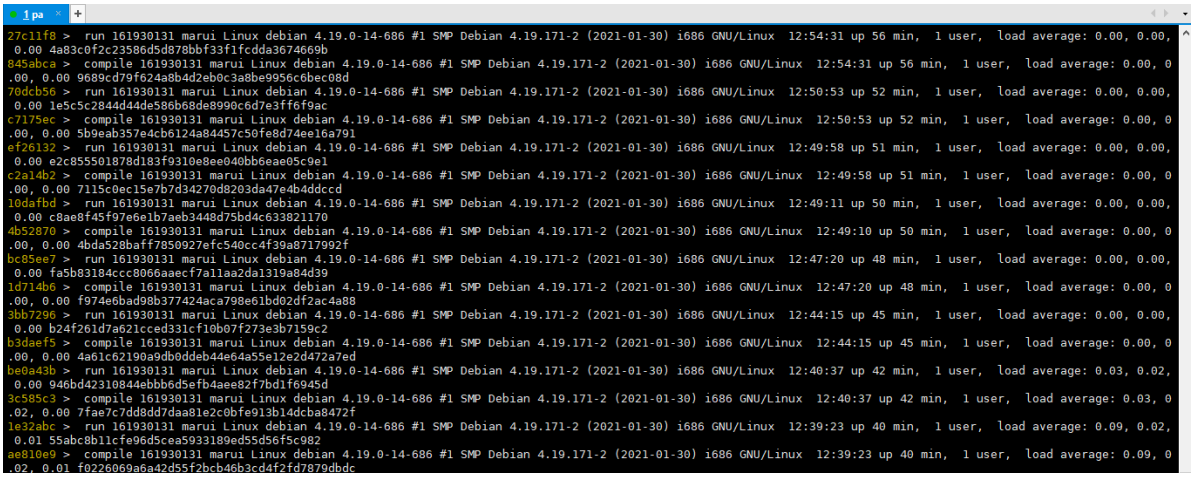
八、Git Log截图

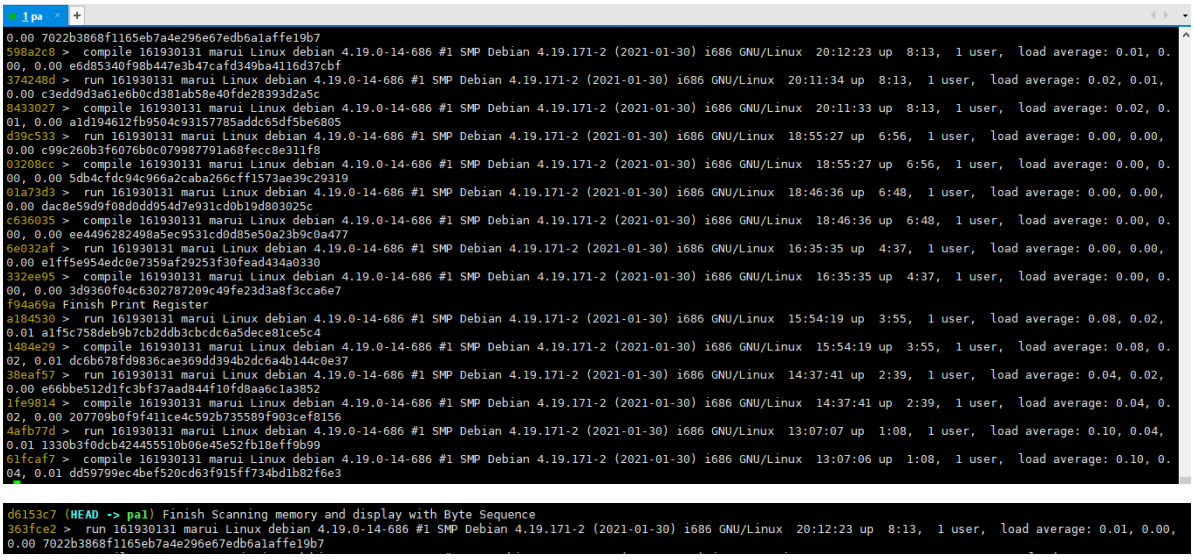
在`pa1`分支下使用命令`git log --oneline`并截图





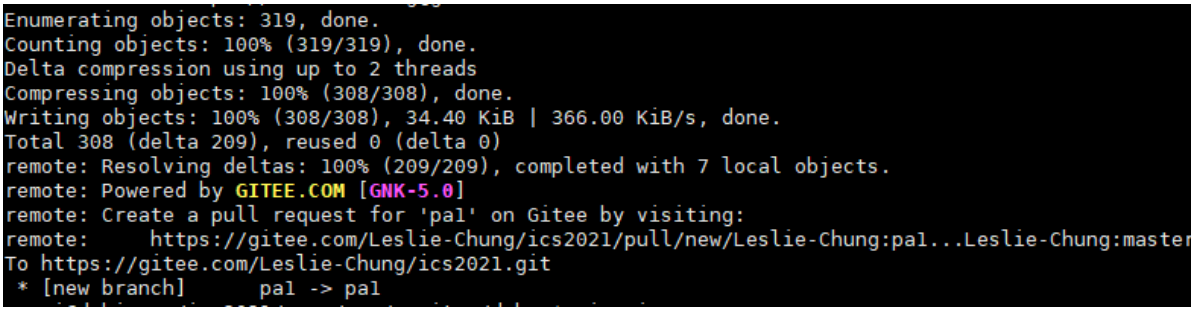






九、Git Branch截图

十、远程git仓库提交截图



实验内容

PA1.1.1 实现寄存器结构体

实现在`nemu/include/cpu/reg.h`中的结构体`CPU_state`：

```
typedef struct {
  union {
    union {
      unsigned int _32;
      unsigned short _16;
      unsigned char _8[2];
    } gpr[8];
    struct {
      rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
    };//这些元素各自使用各自的空间,且每个元素与相应的gpr[i]共用一个地址
  };

  /* Do NOT change the order of the GPRs' definitions. */

  /* In NEMU, rtlreg_t is exactly uint32_t. This makes RTL instructions
   * in PA2 able to directly access these registers.
   */
  union{
    vaddr_t eip;
    unsigned short ip;
  };
  union{
    unsigned int eflags;
    unsigned short flags;
  };
} CPU_state;
```

对于第一个`union`来说，`gpr[i]`中的元素共用一个空间，`gpr[i]`与`gpr[j]`之间使用不同空间（`0<= i,j < 8`）；其次，`struct`中的元素占用不同的空间，而且从左往右来看，每个元素又与其对应的`gpr[i]`共用同一个空间。

剩余的两个`union`同理。

测试样例：

执行`make run`

```
marui@debian:~/ics2021/nemu$ make run
+ CC src/monitor/debug/ui.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: 20:39:54, Mar 21 2021
For help, type "help"
(nemu)
```

PA1.1.2 实现单步执行

声明并定义函数

```
static int cmd_si(char *args){
  char *arg = strtok(NULL, " ");//因为si指令已经被捕获，所以只需读出数字即可
  int n;
  if (arg){
    n = atoi(arg);
    if (n < -1) {
      printf("Input Digit Error!\n");
      return 0;
    }
  }
  else n = 1;//缺省
  cpu_exec(n);
  return 0;
}
```

将`si`命令加入指令列表中：

```
static struct {
  char *name;
  char *description;
  int (*handler) (char *);
} cmd_table [] = {
  { "help", "Display informations about all supported commands", cmd_help },
  { "c", "Continue the execution of the program", cmd_c },
  { "q", "Exit NEMU", cmd_q },
  { "si", "Single-step execution", cmd_si},
  /* TODO: Add more commands */

};
```

测试样例：

`si 1`, `si`, `si -1`, `si 10`

```
(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 10
nemu: HIT GOOD TRAP at eip = 0x00100026
```

```
(nemu) si -1
nemu: HIT GOOD TRAP at eip = 0x00100026
```

PA1.1.3 修改一次打印步数上限

观察`cpu_exec()`函数（`cpu_exec()`模拟了 CPU 的工作方式：不断执行指令。）

```
for (; n > 0; n --) {
    exec_wrapper(print_flag);
```

`exec_wrapper()` 函数让 CPU 执行当前 `%eip` 指向的一条指令，然后更新 `%eip`。参数 `print_flag` 的初始化为 `bool print_flag = n < MAX_INSTR_TO_PRINT;`。

```
if (print_flag) {
    puts(decoding.asm_buf);
}
```

由此可见，如果该值为真，则会执行指令。

所以修改 `MAX_INSTR_TO_PRINT` 为无限大，则输入所有合法值（`n >= -1`）后，就会根据 `n` 一直执行程序，直到 `n == 0`。

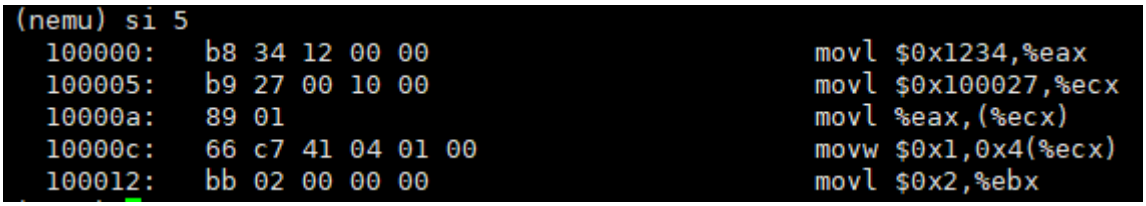
而 `n` 的类型为 `unsigned int`，对应的无限大也就是 `-1`。

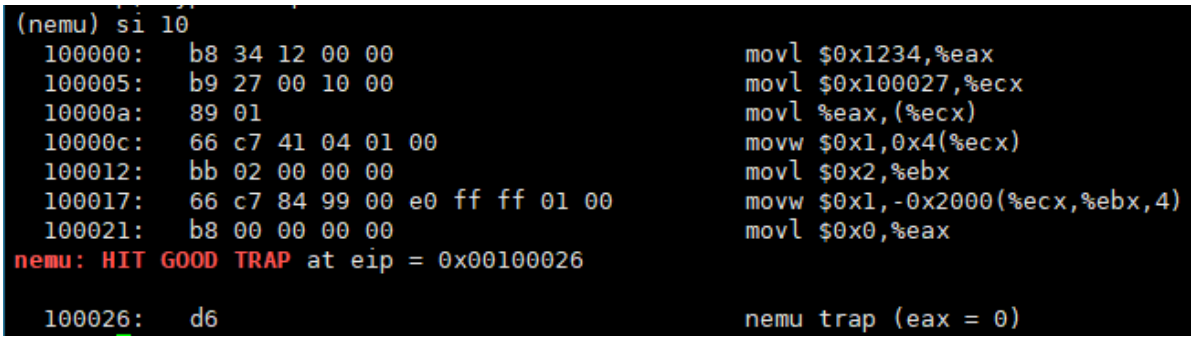
在 `nemu\src\monitor\cpu-exec.c` 中修改：

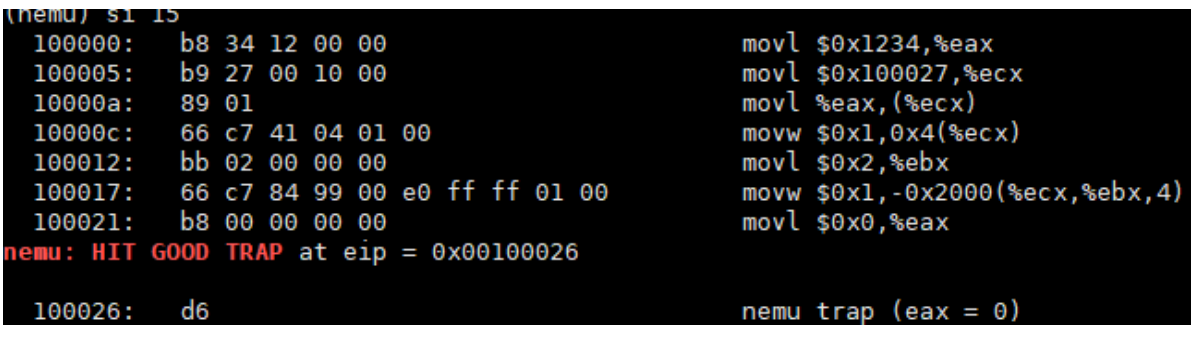
```
#define MAX_INSTR_TO_PRINT -1
```

然后执行命令，`si 5`，`si 10`，`si 15`，`si 1000000`

测试样例：







```
(nemu) si 1000000
  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)
```

PA1.1.4 实现打印寄存器功能

代码实现：

```
void reg_display(){
  int i;
  for (i = R_EAX; i <= R_EBX; i++){
    printf("%s:\t0x%-8x\t%u\n", regsl[i], reg_l(i), reg_l(i));
    printf("%s:\t0x%-8x\t%u\n", regsw[i], reg_w(i), reg_w(i));
    printf("%s:\t0x%-8x\t%u\n", regsb[i], reg_b(i), reg_b(i));
    printf("%s:\t0x%-8x\t%u\n", regsb[i + 4], reg_b((i + 4)), reg_b((i + 4)));
    printf("\n");
  }
  printf("\n");
  for (; i <= R_EDI; ++i) {
    printf("%s:\t0x%-8x\t%u\n", regsl[i], reg_l(i), reg_l(i));
    printf("%s:\t0x%-8x\t%u\n", regsw[i], reg_w(i), reg_w(i));
    printf("\n");
  }
  printf("\n");
  printf("eip:\t0x%-8x\t%u\n", cpu.eip, cpu.eip);
  printf("ip:\t0x%-8x\t%u\n", cpu.ip, cpu.ip);
  printf("\n");
  printf("eflags:\t0x%-8x\t%u\n", cpu.eflags, cpu.eflags);
  printf("flags:\t0x%-8x\t%u\n", cpu.flags, cpu.flags);
}

static int cmd_info(char *args){
  char *arg = strtok(NULL, " ");
  if(arg == NULL){
    printf("A parameter is missing!\n");
    return 0;
  }
  if (strcmp(arg, "r") == 0){
    reg_display();
  }
  else if (strcmp(arg, "w") == 0) {
  }
  else{
    printf("Unknown command '%s'\n", arg);
  }

  return 0;
}
```

将`cmd_info`命令加入指令列表中：

```
static struct {
  char *name;
  char *description;
  int (*handler) (char *);
} cmd_table [] = {
  {"help", "Display informations about all supported commands", cmd_help },
  { "c", "Continue the execution of the program", cmd_c },
  { "q", "Exit NEMU", cmd_q },
  { "si", "Single-step execution", cmd_si},
  { "info", "Show information about registers with argument 'r'", cmd_info},
  /* TODO: Add more commands */

};
```

执行命令`info r`， `si 5`后再次执行`info r`。

```
(nemu) info r
eax:    0x718fd76       119078262
ax:     0xfd76          64886
al:     0x76            118
ah:     0xfd            253

ecx:    0x9c08638       163612216
cx:     0x8638          34360
cl:     0x38            56
ch:     0x86            134

edx:    0x118f213       18412051
dx:     0xf213          61971
dl:     0x13            19
dh:     0xf2            242

ebx:    0xa6457ed       174348269
bx:     0x57ed          22509
bl:     0xed            237
bh:     0x57            87


esp:    0x78382c04      2016947204
sp:     0x2c04          11268

ebp:    0x63e9be52      1676262994
bp:     0xbe52          48722

esi:    0x6acddc7d      1791876221
si:     0xdc7d          56445
```

```
edi:    0x7f232c27      2133011495
di:     0x2c27          11303


eip:    0x100000        1048576
ip:     0x0             0

eflags: 0x0             0
flags:  0x0             0
```

```
(nemu) si 5
  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
(nemu) info r
eax:    0x1234          4660
ax:     0x1234          4660
al:     0x34            52
ah:     0x12            18

ecx:    0x100027        1048615
cx:     0x27            39
cl:     0x27            39
ch:     0x0             0

edx:    0x118f213       18412051
dx:     0xf213          61971
dl:     0x13            19
dh:     0xf2            242

ebx:    0x2             2
bx:     0x2             2
bl:     0x2             2
bh:     0x0             0
```

```
esp:    0x78382c04      2016947204
sp:     0x2c04          11268

ebp:    0x63e9be52      1676262994
bp:     0xbe52          48722

esi:    0x6acddc7d      1791876221
si:     0xdc7d          56445

edi:    0x7f232c27      2133011495
di:     0x2c27          11303


eip:    0x100017        1048599
ip:     0x17            23

eflags: 0x0             0
flags:  0x0             0
```

PA1.1.5 实现扫描内存功能

代码实现：

```c
static int cmd_x(char *args){
  char *arg1 = strtok(NULL, " ");
  char *arg2 = strtok(NULL, " ");
  if (arg1 == NULL || arg2 == NULL) {
    printf("A parameter is missing!\n");
    return 0;
```

```
  }
  int n = atoi(arg1); //读取要读取的次数
  uint32_t addr = 0;
  sscanf(arg2, "%x", &addr); //读取起始位置
  if (n < 1){
    printf("Invalid arguments for x!\n");
    return 0;
  }
  int i;
  uint32_t data;
  printf("Address         Dword block\n");
  //循环使用 vaddr_read 函数来读取内存
  for (i = 1; i <= n; i++, addr += 4){
    data = vaddr_read(addr, 4);
    printf("0x%08x\t", addr);
    printf("0x%08x\t\n", data);
  }
  return 0;
}
```

> `nemu`的内存通过在`nemu/src/memory/memory.c`中定义的大数组`pmem`来模拟。
>
> `vaddr_read(vaddr_t addr, int len)`函数用来访问模拟内存，前一个参数是要访问的虚拟地址，后一个是扫描长度。该函数会返回地址中的数据，类型为`unsigned int`。

加入命令列表：

```
static struct {
  char *name;
  char *description;
  int (*handler) (char *);
} cmd_table [] = {
  {"help", "Display informations about all supported commands", cmd_help },
  { "c", "Continue the execution of the program", cmd_c },
  { "q", "Exit NEMU", cmd_q },
  { "si","Usage: si [N]\n"\
        "     Execute the program with N(default: 1) step", cmd_si },
  { "info", "Show information about registers with argument 'r' and show information about watchpoint with argumen
t 'w'", cmd_info},
  { "x", "Usage: x [N] [EXPR]\n" \
      "    Calculate the value of the expression EXPR, and output N consecutive 4 bytes starting from EXPR in hexa
decimal form", cmd_x },
  /* TODO: Add more commands */

};
```

执行命令`x 4 0x100000`。

```
(nemu) x 4 0x100000
Address         Dword block
0x00100000      0x001234b8
0x00100004      0x0027b900
0x00100008      0x01890010
0x0010000c      0x0441c766
```

PA1.1.6 实现扫描内存字节单位显示

代码实现：

```
void byteSequence_dispaly(uint32_t data){
  printf("   ...  ");
  uint32_t byte[4];
  byte[0] = data & 0x000000ff;
  byte[1] = (data & 0x0000ff00) >> 8;
  byte[2] = (data & 0x00ff0000) >> 16;
  byte[3] = (data & 0xff000000) >> 24;
  int i;
  for(i = 0; i < 4; i++){
    printf("%02x ",byte[i]);
  }
  printf("\n");
}


static int cmd_x(char *args){
  char *arg1 = strtok(NULL, " ");
  char *arg2 = strtok(NULL, " ");
  if (arg1 == NULL || arg2 == NULL) {
    printf("A parameter is missing!\n");
    return 0;
  }
  int n = atoi(arg1); //读取要读取的次数
  uint32_t addr = 0;
  sscanf(arg2, "%x", &addr); //读取起始位置
  if (n < 1){
    printf("Invalid arguments for x!\n");
    return 0;
  }
  int i;
  uint32_t data;
  printf("Address         Dword block   ...  Byte sequence\n");
  //循环使用 vaddr_read 函数来读取内存
  for (i = 1; i <= n; i++, addr += 4){
    data = vaddr_read(addr, 4);
    printf("0x%08x\t", addr);
    printf("0x%08x", data);
    byteSequence_dispaly(data);
  }
  return 0;
}
```

执行命令 `x 4 0x100000`

```
(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
```

遇到的问题及解决办法

在阅读讲义时，看到这个图非常不理解：为什么`EAX`在中间位置，而不是在`31`的地方，以为上方的数字对应其位数。

```
31                              16 15             8 7              0
+-----------------+-----------------+-----------------+----------------+
|                                EAX       AH       AX      AL       |  累加器 EAX
+-----------------+-----------------+-----------------+----------------+
|                                EDX       DH       DX      DL       |  数据寄存器 EDX
+-----------------+-----------------+-----------------+----------------+
|                                ECX       CH       CX      CL       |  计数寄存器 ECX
+-----------------+-----------------+-----------------+----------------+
|                                EBX       BH       BX      BL       |  基址寄存器 EBX
+-----------------+-----------------+-----------------+----------------+
|                                EBP                BP               |  基址指针 EBP
+-----------------+-----------------+-----------------+----------------+
|                                ESI                SI               |  源变址器 ESI
+-----------------+-----------------+-----------------+----------------+
|                                EDI                DI               |  目标变址器 EDI
+-----------------+-----------------+-----------------+----------------+
|                                ESP                SP               |  堆栈指针 ESP
+-----------------+-----------------+-----------------+----------------+
```

在请教学长之后知道了：

> `EAX`在中间是因为，它能表示的数据范围是0~31，它在哪个范围内就将其放置在哪个区间的中心位置；
>
> 再例如AH，它在8~15位之间，则它能表示相应的数据，也就是`AX`的高八位。

实验心得

- 此次实验让我大开眼界，了解到原来可以用C语言来模拟一台计算机，感觉非常神奇；
- 在阅读代码的同时也让我觉得，一个标准的C语言项目也是需要很深的功底去编写的，而且库函数也很丰富，之前自己在写C程序的时候也就是用了简单的几个库函数；
- 无符号数、位运算、匿名`union`与`struct`的搭配使用，这些都是在之前的程序设计课程上接触的比较少的，这次实验也让我更加了解了它们的一些用法。

其他备注

无