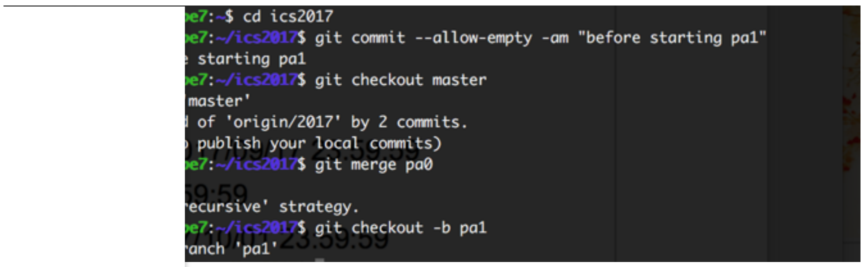
1 PA1 – 开天辟地的篇章：最简单的计算机

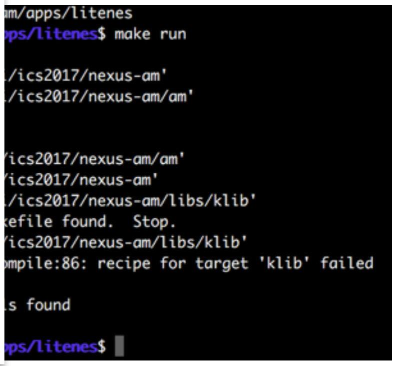
1.1 在开始愉快的 PA 之旅之前

1） 建一个 commit，恢复 master 暂存区文件到工作区，将 pa0 合并到当前分支，新建一个分支 pa1。



```
oe7:~$ cd ics2017
oe7:~/ics2017$ git commit --allow-empty -am "before starting pa1"
e starting pa1
oe7:~/ics2017$ git checkout master
'master'
d of 'origin/2017' by 2 commits.
o publish your local commits)
oe7:~/ics2017$ git merge pa0

recursive' strategy.
oe7:~/ics2017$ git checkout -b pa1
ranch 'pa1'
```

2） 配置 X Servier。安装结束后为 SSH 打开 X11 转发功能（每次）;



```
am/apps/litenes
ps/litenes$ make run

./ics2017/nexus-am'
./ics2017/nexus-am/am'


/ics2017/nexus-am/am'
/ics2017/nexus-am'
./ics2017/nexus-am/libs/klib'
kefile found.  Stop.
/ics2017/nexus-am/libs/klib'
ompile:86: recipe for target 'klib' failed

.s found

ps/litenes$
```

虽然显示有 error，但是马里奥可以玩。

3） NEMU 是什么？

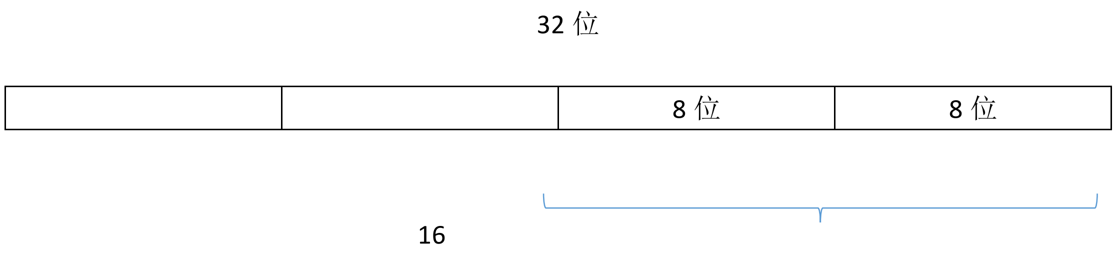
NEMU 是一个虚拟出来的计算机系统，通过程序实现物理计算机的基本功能。

4） 初识虚拟化。在Windows中使用Docker安装了一个GNU/Linux container，然后在container中完成PA，通过NEMU运行Hello World程序的层次图：

“Hello World”program

```
-------------------------------
Simulated x86 hardware
--------------------------------
NEMU
--------------------------------
GNU/Linux
---------------------------------
Docker
----------------------------------
Computer hardware
```

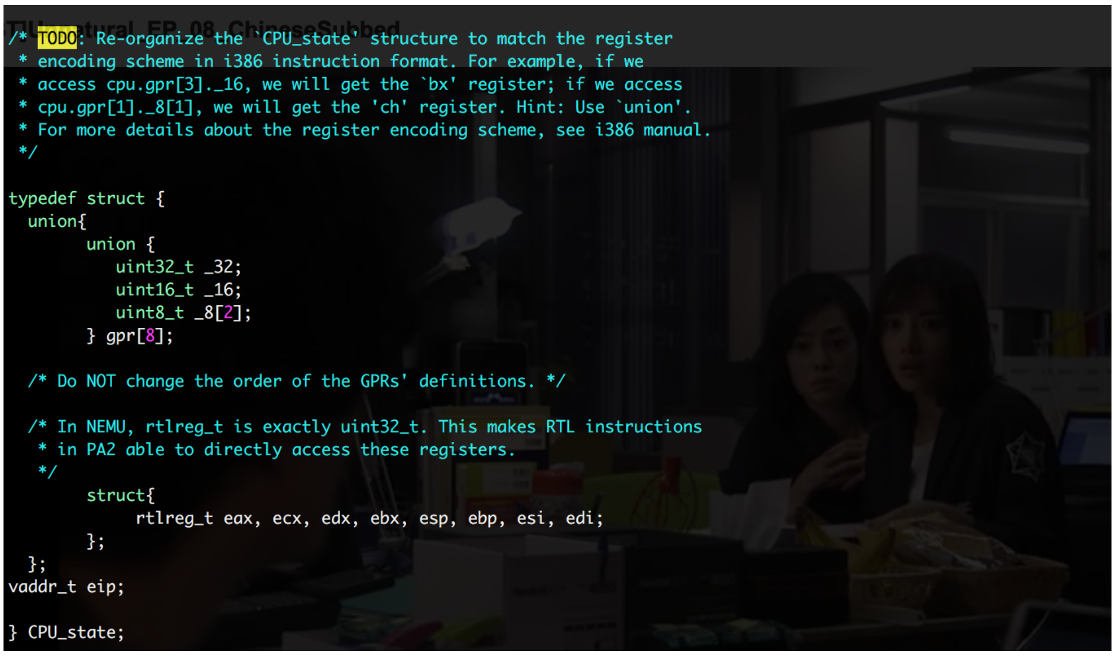
1.2 开天辟地的故事



1.3 RTFSC

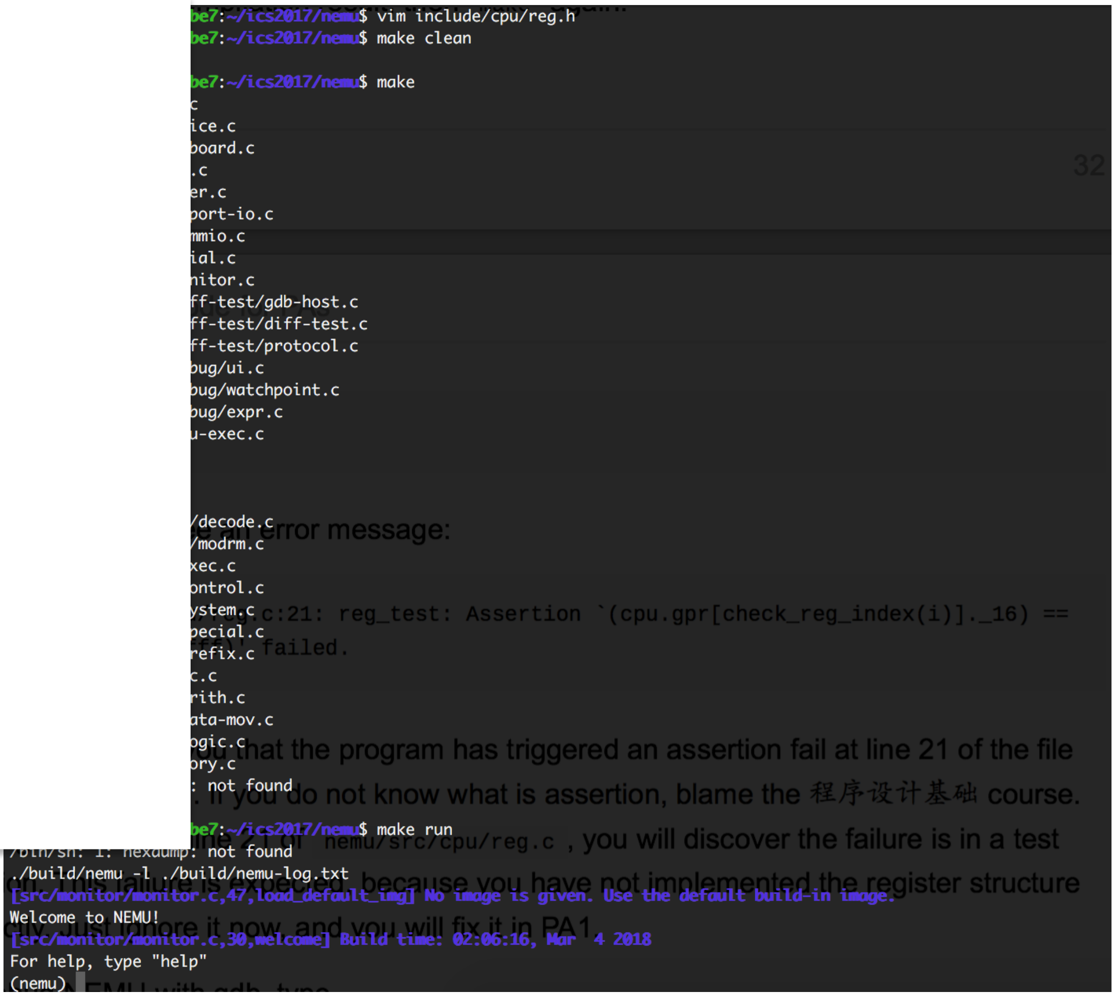
1） 思考题：一个程序是从 main 函数开始执行的。
2） 实现正确的寄存器结构体。
讲义中提到使用 union，前面知道 CPU 的寄存器是公用内存的，所以把 gpr[8]结构体改成了 union，但是实验报错，然后继续改，看到下面的 eax,ecx 之类的寄存器，于是在外面又套了一个 union，但是还是报错，后来在网上搜，看到简书上面有一个解答，就是在 eax，ecx 之类的寄存器外面套一个 struct，尝试之后成功了，但是原理不知道。

修改后的结构体：

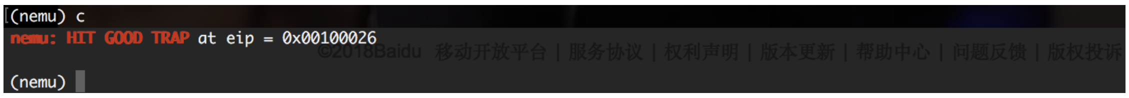


```
/* TODO: Re-organize the `CPU_state' structure to match the register
 * encoding scheme in i386 instruction format. For example, if we
 * access cpu.gpr[3]._16, we will get the `bx' register; if we access
 * cpu.gpr[1]._8[1], we will get the 'ch' register. Hint: Use `union'.
 * For more details about the register encoding scheme, see i386 manual.
 */

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

  /* 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.
   */
        struct{
             rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
        };
  };
vaddr_t eip;

} CPU_state;
```

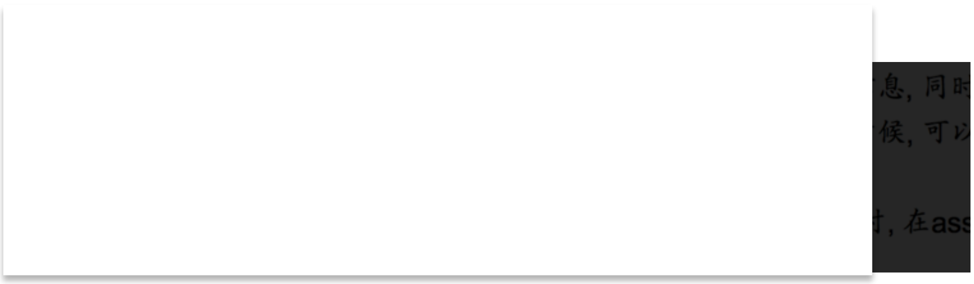
进入 nemu：



```
be7:~/ics2017/nemu$ vim include/cpu/reg.h
be7:~/ics2017/nemu$ make clean

be7:~/ics2017/nemu$ make
c
ice.c
board.c
.c
er.c
port-io.c
mmio.c
ial.c
nitor.c
ff-test/gdb-host.c
ff-test/diff-test.c
ff-test/protocol.c
bug/ui.c
bug/watchpoint.c
bug/expr.c
u-exec.c



/decode.c
/modrm.c
xec.c
ontrol.c
ystem.c
pecial.c
refix.c
c.c
rith.c
ata-mov.c
ogic.c
ory.c
: not found
be7:~/ics2017/nemu$ make run
/bin/sh: 1: hexdump: not found
./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: 02:06:16, Mar  4 2018
For help, type "help"
(nemu)
```

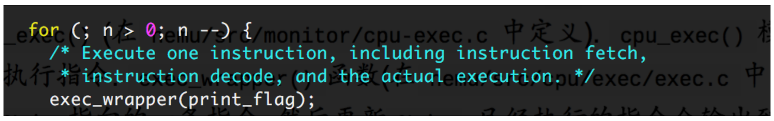


```
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x00100026

(nemu)
```


息，同时
候，可以
时，在ass

3） 究竟要执行多久？

在 cmd_c()函数中调用 cpu_exec()传入参数-1，根据讲义，我在 cpu_exec()看到如下：



```
  for (; n > 0; n --) {
    /* Execute one instruction, including instruction fetch,
     * instruction decode, and the actual execution. */
    exec_wrapper(print_flag);
```

n 是无符号整型，所以-1 就是无符号最大的数字，那么 for 循环可以执行最大次数的循环，而 ecex_wrapper()函数就是执行%eip 指向的当前指令并更新%eip。最终就可以执行完所有指令。

4） 温故而知新

opcode_table 数组是一个函数指针数组（helper 函数），对应某条指令的某种形式。

5） 谁来指示程序的结束？

return 语句是返回结果，并终止当前函数。全局对象的析构函数会在 main 函数之后执行，用 atexit 注册的函数也会在 main 之后执行。main 函数执行完之后还要去执行一些诸如释放空间之类的操作。Main 函数结束时会隐式的调用 exit()函数，运行时会执行由 atexit()函数登记的函数，做一些清理，刷新所有输出流，关闭所有打开的流。所以应该是 exit()函数指示吧。

证明在 main 函数返回后可以再执行的代码：

```
#include <stdlib.h>
#include <stdio.h>

void fn1(void)
{
    printf("next.\n");
}

int main(void)
{
    atexit(fn1);

    puts("This is executed first.");
    return 0;
}
```

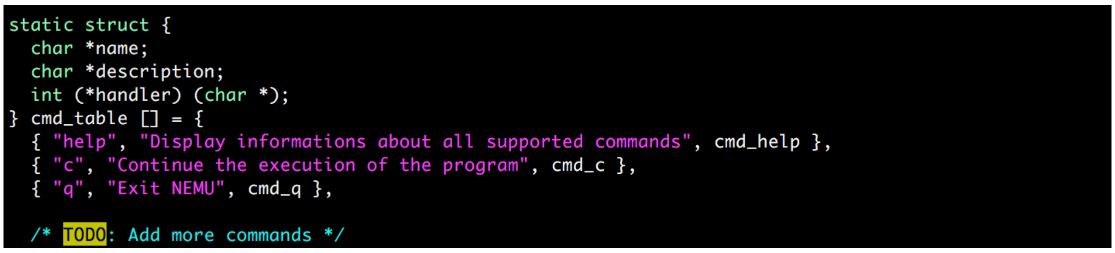
1.4 基础设施

解析命令

1） readline()，读取一行文本。
2） strtok()，根据给定的字符结点，将字符串分解成一段段的字符串。
3） sscanf()，根据字符串读入相符的数据

单步执行

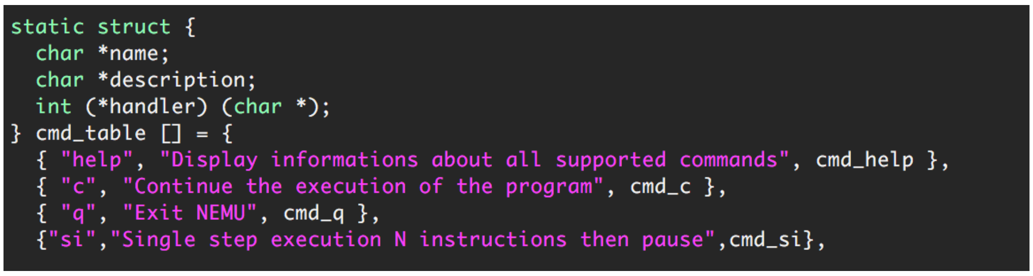
1）在 ui.c 中看到：



```
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 },

  /* TODO: Add more commands */
```

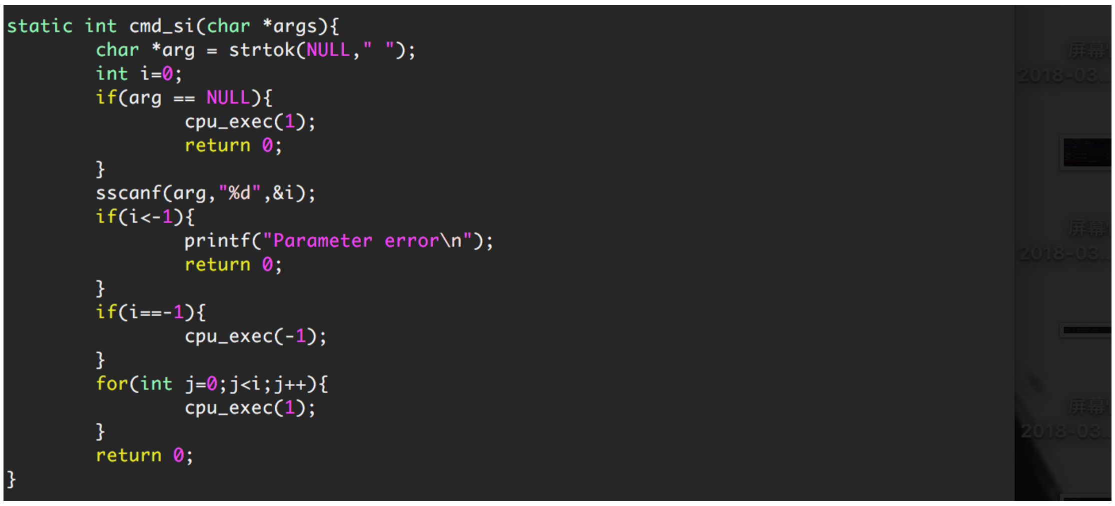
看了代码之后，就明白了是用 readline 读取文本行之后，用 strtok()分解第一个字符串，与 cmd_table[]中的 name 比较，执行对应的操作。

于是做如下变动：



```
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 N instructions then pause",cmd_si},
```

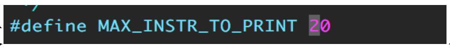
cmd_si 函数读取字符串参数，用 sscanf 转化为相符的数据，执行对应指令。



```
static int cmd_si(char *args){
        char *arg = strtok(NULL," ");
        int i=0;
        if(arg == NULL){
                cpu_exec(1);
                return 0;
        }
        sscanf(arg,"%d",&i);
        if(i<-1){
                printf("Parameter error\n");
                return 0;
        }
        if(i==-1){
                cpu_exec(-1);
        }
        for(int j=0;j<i;j++){
                cpu_exec(1);
        }
        return 0;
}
```

代码如下：

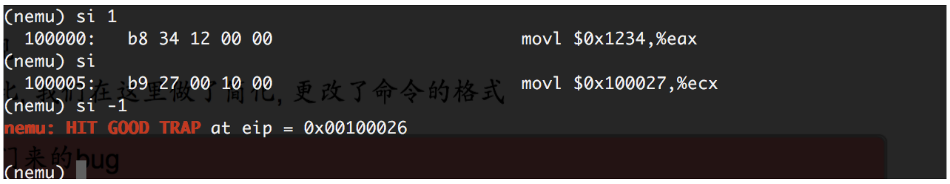
```
static int cmd_si(char *args){
    char *arg = strtok(NULL," ");
    int i=0;
    if(arg == NULL){
        cpu_exec(1);
        return 0;
    }
    sscanf(arg,"%d",&i);
    if(i<-1){
        printf("Parameter error\n");
        return 0;
    }
    if(i==-1){
        cpu_exec(-1);
    }
    for(int j=0;j<i;j++){
        cpu_exec(1);
    }
    return 0;
}
```

尝试做加分项，我修改

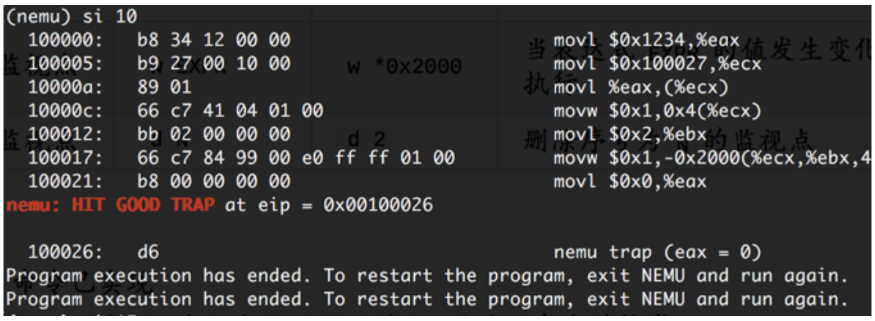
```
#define MAX_INSTR_TO_PRINT 20
```

从 10 到 20，但是修改后还是没有太大差别，其他地方也尝试过了，但是会出现错误需要查看手册。反正修改之后依然无法成功。

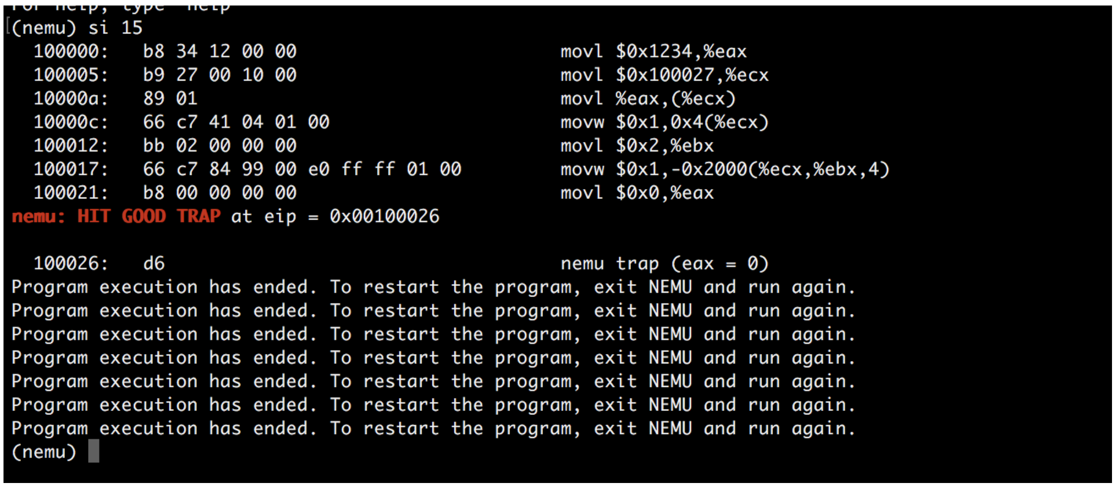
2） 结果截图：



```
(nemu) si 1
 100000:   b8 34 12 00 00                          movl $0x1234,%eax
(nemu) si
 100005:   b9 27 00 10 00                          movl $0x100027,%ecx
(nemu) si -1
nemu: HIT GOOD TRAP at eip = 0x00100026
(nemu)
```



```
(nemu) si 10
  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)
Program execution has ended. To restart the program, exit NEMU and run again.
Program execution has ended. To restart the program, exit NEMU and run again.
```

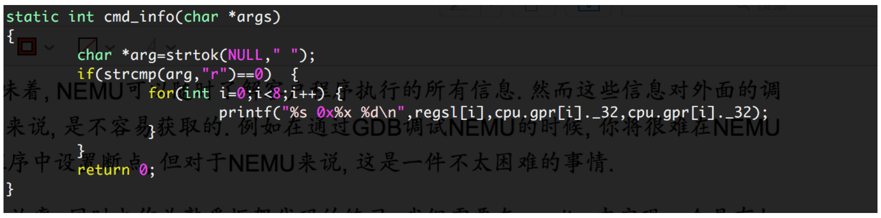


```
For help, type "help"
(nemu) si 15
  100000:   b8 34 12 00 00                          movl $0x1234,%eax
  100005:   b9 27 00 10 00                          movl $0x100027,%ecx
  10000a:   89 01                                   movl %eax,(%ecx)
  10000c:   66 c7 41 04 01 00                       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)
Program execution has ended. To restart the program, exit NEMU and run again.
Program execution has ended. To restart the program, exit NEMU and run again.
Program execution has ended. To restart the program, exit NEMU and run again.
Program execution has ended. To restart the program, exit NEMU and run again.
Program execution has ended. To restart the program, exit NEMU and run again.
Program execution has ended. To restart the program, exit NEMU and run again.
Program execution has ended. To restart the program, exit NEMU and run again.
(nemu)
```

3） log

打印寄存器

1）regsl[]，32 位寄存器，regsw[]，16 位寄存器，regsb[]，8 位寄存器。在前面实现寄存器的正确结构时知道 cpu.gpr[i]._32 是 uint32_t 地址。判断参数为 r 后输出 32 位寄存器的地址。

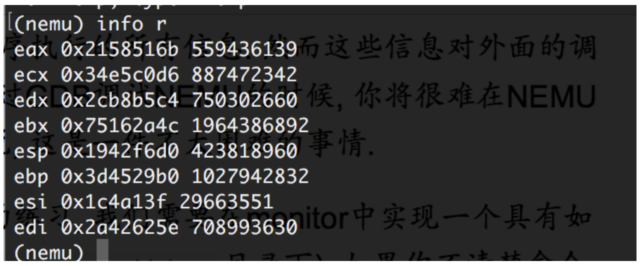


```
static int cmd_info(char *args)
{
        char *arg=strtok(NULL," ");
        if(strcmp(arg,"r")==0)  {
                for(int i=0;i<8;i++) {
                        printf("%s 0x%x %d\n",regsl[i],cpu.gpr[i]._32,cpu.gpr[i]._32);
                }
        }
        return 0;
}
```

代码如下：

```c
static int cmd_info(char *args)
{
    char *arg=strtok(NULL," ");
    if(strcmp(arg,"r")==0)  {
        for(int i=0;i<8;i++) {
            printf("%s %x %d\n",regsl[i],cpu.gpr[i]._32,cpu.gpr[i]._32);
        }
    }
    return 0;
}
```

2）结果

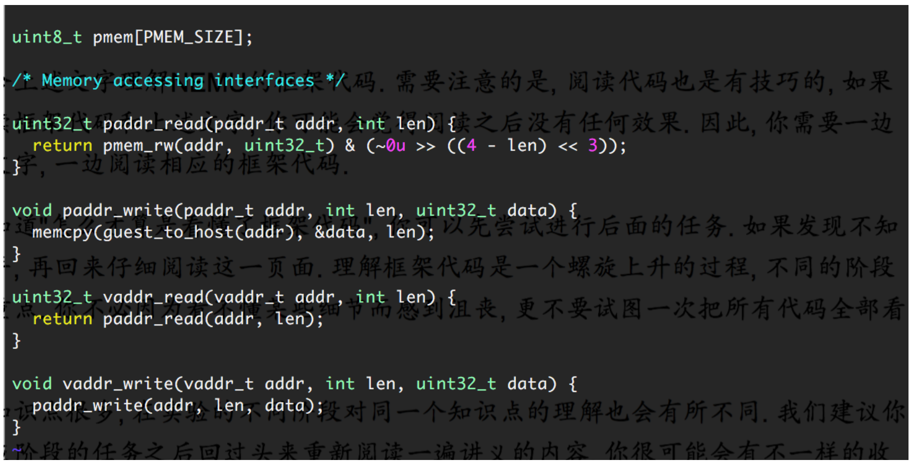


```
(nemu) info r
eax 0x2158516b 559436139
ecx 0x34e5c0d6 887472342
edx 0x2cb8b5c4 750302660
ebx 0x75162a4c 1964386892
esp 0x1942f6d0 423818960
ebp 0x3d4529b0 1027942832
esi 0x1c4a13f 29663551
edi 0x2a42625e 708993630
(nemu)
```

3）log

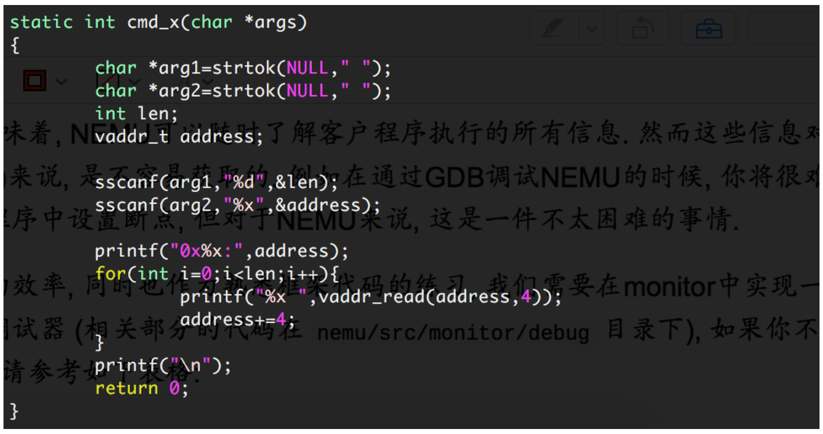
扫描内存

1）讲义中提到过 vaddr_read()在 memory.c 中，查看如下：



```c
uint8_t pmem[PMEM_SIZE];

/* Memory accessing interfaces */

uint32_t paddr_read(paddr_t addr, int len) {
  return pmem_rw(addr, uint32_t) & (~0u >> ((4 - len) << 3));
}

void paddr_write(paddr_t addr, int len, uint32_t data) {
  memcpy(guest_to_host(addr), &data, len);
}

uint32_t vaddr_read(vaddr_t addr, int len) {
  return paddr_read(addr, len);
}

void vaddr_write(vaddr_t addr, int len, uint32_t data) {
  paddr_write(addr, len, data);
}
```

其实就是 vaddr_read 函数调用 paddr_read，传入两个参数：起始地址，扫描长度。
所以我们通过 strtok 分别获得字符串型的地址和扫描长度，用 sscanf 转换为要求的形式，而后循环调用 vaddr_read 函数扫描内存。

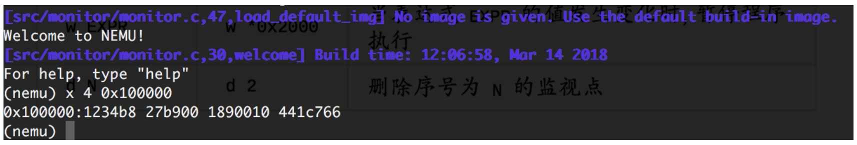


```c
static int cmd_x(char *args)
{
	char *arg1=strtok(NULL," ");
	char *arg2=strtok(NULL," ");
	int len;
	vaddr_t address;

	sscanf(arg1,"%d",&len);
	sscanf(arg2,"%x",&address);

	printf("0x%x:",address);
	for(int i=0;i<len;i++){
		printf("%x ",vaddr_read(address,4));
		address+=4;
	}
	printf("\n");
	return 0;
}
```

代码如下：

```c
static int cmd_x(char *args)
{
    char *arg1=strtok(NULL," ");
    char *arg2=strtok(NULL," ");
    int len;
    vaddr_t address;

    sscanf(arg1,"%d",&len);
    sscanf(arg2,"%x",&address);

    printf("0x%x:",address);
    for(int i=0;i<len;i++){
        printf("%x ",vaddr_read(address,4));
        address+=4;
    }
    printf("\n");
    return 0;
}
```

2） 结果



```
[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: 12:06:58, Mar 14 2018
For help, type "help"
(nemu) x 4 0x100000
0x100000:1234b8 27b900 1890010 441c766
(nemu) 
```

3）log

ty
表达式
的值，将
形式输出