南京航空航天大学《计算机组成原理Ⅱ课程设计》报告

- 姓名：唐希
- 班级：1619201
- 学号：161920122
- 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

思考题

1.存放的是什么？

存放的是指令的地址，PC指向下条执行指令地址从而执行下一条指令，用地址取指的方式便于执行跳转指令。

2.贵圈真乱

```
+---------------------------------+
|      "Hello World" program      |
+---------------------------------+
|      Simulated x86 hardware     |
+---------------------------------+
|            GNU/Linux            |
+---------------------------------+
|              Nemu               |
+---------------------------------+
|            Dockers              |
+---------------------------------+
|        Computer hardware        |
+---------------------------------+
```

3..虚拟机和模拟器的差别

两者都是通过在pc上分配一部分硬盘空间来实现一台全新的机器，没有本质的区别。虚拟机提供了一个接口来对真实的机器情况进行模拟，抽象化的表示机器的运作。模拟器则更加表面一点，只是用于模拟机器使用方面的过程，对如何实现和更加深层的结构不加讨论。

4.从哪开始阅读代码呢?

一个c语言程序从main函数开始执行。

5.究竟要执行多久?

n是无符号整型，-1表示最大的数。所以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。

6.谁来指示程序的结束?

一个C程序总是从main()函数开始执行的，但是不一定在main函数最后结束。在 main() 之后的最后一条语句结束后，程序还要运行一些代码，以正常返回操作系统。main函数执行完后还执行其他语句，有时候需要有一种与程序退出方式无关的方法来进行程序退出时的必要处理，方法就是用atexit()函数来注册程序正常终止时要被调用的函数，atexit()函数的参数是一个函数指针，函数指针指向一个没有参数也没有返回值的函数。

7.为什么会这样?

数据的储存有大端与小端两种方式，小端字节序，低地址低字节，高地址高字节；大端字节序，低地址低字节，高地址低字节，所以单字节和4字节打印出来的顺序不一样。

8.Git Log截图

忘记创建pa1了，这是pa0下的截图。

```
tangxi@debian: ~/ics2021
tangxi@debian:~/ics2021$ git log --oneline
5dc4ca3 (HEAD -> pa0) >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  08:43:24 up 11 min,  1 user,  l
oad average: 0.00, 0.00, 0.00 8b939943e4aae59e36e0a74e39cb1df42aa1a16
9b9e07a >  compile 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  08:43:24 up 11 min,  1 user,  load averag
e: 0.00, 0.00, 0.00 e23763541f49ea816431e257462eae69ab4453ea
30e4c3b >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  08:36:12 up 4 min,  1 user,  load average: 0.
00, 0.02, 0.00 eed4327c70ce5a211153bbbf58e04c433eae9660
715522c >  compile 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  08:36:12 up 4 min,  1 user,  load average
: 0.00, 0.02, 0.00 cee08988ecf07ebed3c58f378b6b6cbf43041054
3466989 >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  08:27:06 up 48 min,  1 user,  load average: 0
.00, 0.00, 0.00 7b528565502071124eff9d34798ca3d1ce3a5805
e4ed776 >  compile 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  08:27:05 up 48 min,  1 user,  load averag
e: 0.00, 0.00, 0.00 7f3f4cfc7f2b019111a746f7f70797f69adf0
04a4dc0 (pa1) >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  01:49:16 up  4:02,  1 user,  load avera
ge: 0.00, 0.00, 0.00 b9c2e9044152e701839162212f9c19918a5b2863
d1caf15 >  compile 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  01:49:16 up  4:02,  1 user,  load average
: 0.00, 0.00, 0.00 807522566bdf4d81f251007df445eda4e74d0923
0271dfd >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  01:46:12 up  3:59,  1 user,  load average: 0.
00, 0.00, 0.00 eb7db33de3b64015860416cc73f6194e23f35504
f8b205e >  compile 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  01:46:12 up  3:59,  1 user,  load average
: 0.00, 0.00, 0.00 21f4d3567fedba459977539a3d3a82635434b5d
ecfbd6f >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  01:30:31 up  3:43,  1 user,  load average: 0.
00, 0.00, 0.00 59dcd774c3de4904a7cb5a3bef7857a7e7e1355
57321a8 >  compile 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  01:30:30 up  3:43,  1 user,  load average
: 0.00, 0.00, 0.00 7770eebd1f0d19f0e20a1a531839caa22a3e673a
cb5eb38 >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:49:08 up  3:02,  1 user,  load average: 0.
01, 0.01, 0.00 ad40a02c233383b5856e13d650b24872338335e4
0d522a1 >  compile 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:49:07 up  3:02,  1 user,  load average
: 0.01, 0.01, 0.00 c6a13cbedc68167ed6737a00e32d01a5295d708
f05a492 >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:27:29 up  2:40,  1 user,  load average: 0.
00, 0.00, 0.00 df341ae5cf1bccb49476ea59828abfbd358b7b98
a07406c >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:27:20 up  2:40,  1 user,  load average: 0.
00, 0.00, 0.00 6115cf1985920ad78de5edcab669af6d658a0f9a
ac41f71 >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:25:46 up  2:38,  1 user,  load average: 0.
00, 0.00, 0.00 8424f78ba2a024638c052c250ade58713599090
440b11e >  compile 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:25:46 up  2:38,  1 user,  load average
: 0.00, 0.00, 0.00 9959c4f32afbb7d58816222c8b64ac2bf8879912
b3a8598 >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:20:59 up  2:34,  1 user,  load average: 0.
00, 0.00, 0.00 11fc4aadf4802fefc181fd9f2476ad7db49509d3
dd5070c >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:20:36 up  2:33,  1 user,  load average: 0.
00, 0.00, 0.00 ccf95c79bf20a1d66b0ec90564ae4c1350ee5c
66a3859 >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:19:00 up  2:32,  1 user,  load average: 0.
00, 0.00, 0.00 a69d0e6ed92f059fd85a4d30861d10a9aa408d0e
6b4a4a5 >  run 161920122 tangxi Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux  00:04:39 up  2:17,  1 user,  load average: 0.
00, 0.00, 0.00 c817db5a3096282a3794a3171d330fb3d649ab72
```

9.Git Branch截图

```
tangxi@debian:~/ics2021$ git checkout pa1
Switched to branch 'pa1'
tangxi@debian:~/ics2021$ git branch
  master
  pa0
* pa1
tangxi@debian:~/ics2021$
```

10.远程git仓库提交截图

```
tangxi@debian:~/ics2021$ git push myrepo pa0
Username for 'https://e.coding.net': 15913121302
Password for 'https://15913121302@e.coding.net':
Enumerating objects: 103, done.
Counting objects: 100% (103/103), done.
Compressing objects: 100% (93/93), done.
Writing objects: 100% (93/93), 10.69 KiB | 405.00 KiB/s, done.
Total 93 (delta 70), reused 0 (delta 0)
To https://e.coding.net/tangxi1/tangxi/PA.git
   0c18d6c..5dc4ca3  pa0 -> pa0
tangxi@debian:~/ics2021$
```

tracer-ics2017 > compile ···　　最后提交 9b9e07ac04 于 28 分钟前

```
161920122
tangxi
Linux debian 4.19.0-14-686 #1 SMP Debian 4.19.171-2 (2021-01-30) i686 GNU/Linux
 084324 up 11 min,  1 user,  load average: 0.00, 0.00, 0.00
e23763541f49ea816431e257462eae69ab4453ea
```

..

expr.c	Jin Hang	ics2017 initialized	3 年前
ui.c	tracer-ics2...	> compile	28 分钟前
watchpoint.c	Jin Hang	ics2017 initialized	3 年前

实验内容

PA1.1 简易调试器

任务1：实现正确的寄存器结构体

struct(结构体)与union(共用体)有以下的区别：

1.共用体和结构体都是由多个不同的数据类型成员组成， 但在任何同一时刻，共用体只存放一个被选中的成员， 而结构体则存放所有的成员变量。
2.对于共用体的不同成员赋值，将会对其他成员重写， 原来成员的值就不存在了， 而对于结构体的不同成员赋值是互不影响的。
3.二者的内存分配不同。共用体的大小为其内部所有变量的最大值。

寄存器在物理结构上是相互嵌套的，32位寄存器中有16位的寄存器，16位中又有8位的。符合union的结构类型。

32位寄存器eax,ebx,ecx,edx,esp,esi,edp,edi之间相互独立，所以包含在同一结构体中
gpr与eax,ebx,ecx,edx,esp,esi,edp,edi都表示寄存器，所以指向同一内存地址，包含在一个共用体中
eip与寄存器与通用寄存器相互独立，所以最终包含在同一结构体.

```
typedef struct {
  struct {
    uint32_t _32;
    uint16_t _16;
    uint8_t _8[2];
  } gpr[8];
  rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;

  vaddr_t eip;

}
```

改成：

```
typedef struct{
   union{
      union{
         uint32_t _32;
         uint16_t _16;
         uint8_t _8[2];
      }gpr[8];
    struct{
        uint32_t eax,ecx,edx,ebx,esp,edp,esi,edi;
    };
   };
   vaddr_t eip;
}CPU_state;
```

然后运行nemu，发现可以成功运行

tangxi@debian: ~/ics2021/nemu

```
tangxi@debian:~/ics2021/nemu/include/cpu$ cd ..
tangxi@debian:~/ics2021/nemu/include$ cd ..
tangxi@debian:~/ics2021/nemu$ make run
+ CC src/memory/memory.c
In file included from ./include/nemu.h:6,
                 from src/memory/memory.c:1:
./include/cpu/reg.h:25:6: error: unknown type name 'unit32_t'
      unit32_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
      ^~~~~~~~
make: *** [Makefile:25: build/obj/memory/memory.o] Error 1
tangxi@debian:~/ics2021/nemu$ cd include/
tangxi@debian:~/ics2021/nemu/include$ cd cpu/
tangxi@debian:~/ics2021/nemu/include/cpu$ vim reg.h
Error detected while processing /home/tangxi/.vimrc:
line   39:
E488: Trailing characters: end if
line  139:
E171: Missing :endif
Press ENTER or type command to continue
tangxi@debian:~/ics2021/nemu/include/cpu$ cd../..
-bash: cd../..: No such file or directory
tangxi@debian:~/ics2021/nemu/include/cpu$ cd ../..
tangxi@debian:~/ics2021/nemu$ make run
+ CC src/memory/memory.c
+ CC src/cpu/intr.c
+ CC src/cpu/reg.c
+ CC src/cpu/decode/modrm.c
+ CC src/cpu/decode/decode.c
+ CC src/cpu/exec/logic.c
+ CC src/cpu/exec/system.c
+ CC src/cpu/exec/special.c
+ CC src/cpu/exec/cc.c
+ CC src/cpu/exec/arith.c
+ CC src/cpu/exec/control.c
+ CC src/cpu/exec/exec.c
+ CC src/cpu/exec/prefix.c
+ CC src/cpu/exec/data-mov.c
+ CC src/monitor/cpu-exec.c
+ CC src/monitor/debug/expr.c
+ CC src/monitor/debug/ui.c
+ CC src/monitor/diff-test/diff-test.c
+ CC src/monitor/monitor.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
uild-in image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 22:54:18, Mar 31 2021
For help, type "help"
(nemu)
```

输入c测试指令的执行

```
tangxi@debian: ~/ics2021/nemu
+ CC src/cpu/exec/arith.c
+ CC src/cpu/exec/control.c
+ CC src/cpu/exec/exec.c
+ CC src/cpu/exec/prefix.c
+ CC src/cpu/exec/data-mov.c
+ CC src/monitor/cpu-exec.c
+ CC src/monitor/debug/expr.c
+ CC src/monitor/debug/ui.c
+ CC src/monitor/diff-test/diff-test.c
+ CC src/monitor/monitor.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 b
uild-in image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 22:54:18, Mar 31 2021
For help, type "help"
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x00100026

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

任务2.1实现单步/指定步数执行功能

进入nemu/src/monitor/debug/ui.c中，找到结构体。

```
tangxi@debian: ~/ics2021/nemu/src/monitor/debug

static int cmd_help(char *args);

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 */

};
```

通过讲义可以了解到 readline 读取我们输入的命令之后，用 exec_wrapper()接收字符串来解析命令，然后与cmd_table[]中的 name 比较，执行对应的函数。

在cmd_table中添加命令si；在单步执行的函数中，为此需要在文件中加入cmd_si函数，用aoti进制转化位相应的格式。

源代码：

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

运行情况

```
tangxi@debian:~/ics2021/nemu$ make run
+ CC src/monitor/cpu-exec.c
+ 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 b
uild-in image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 22:54:18, Mar 31 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 10
  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) si -1
Program execution has ended. To restart the program, exit NEMU and run again.
(nemu) q
tangxi@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 b
uild-in image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 22:54:18, Mar 31 2021
For help, type "help"
(nemu) si -1
nemu: HIT GOOD TRAP at eip = 0x00100026

(nemu)
```

任务2.2：修改一次打印步数上限

运行两次si 5和运行si 10的输出不一样，因为第二次输出5行要从头开始。还是只输出了5行，而一次性输出了10行会多出后5行的内容所以不一样。修改打印步数的上限可以使得一次性输出的指令行数更多。 由cmd_c可知循环函数为cpu_exec，所以，最大上限应该在cpu_exec函数所在的文件中有定义。通过讲义目录找到文件cpu-exec.c，打开该文件后发现最大上限为10。

```
 */
#define MAX_INSTR_TO_PRINT 2147483647
```

改成整型上限。

tangxi@debian: ~/ics2021/nemu

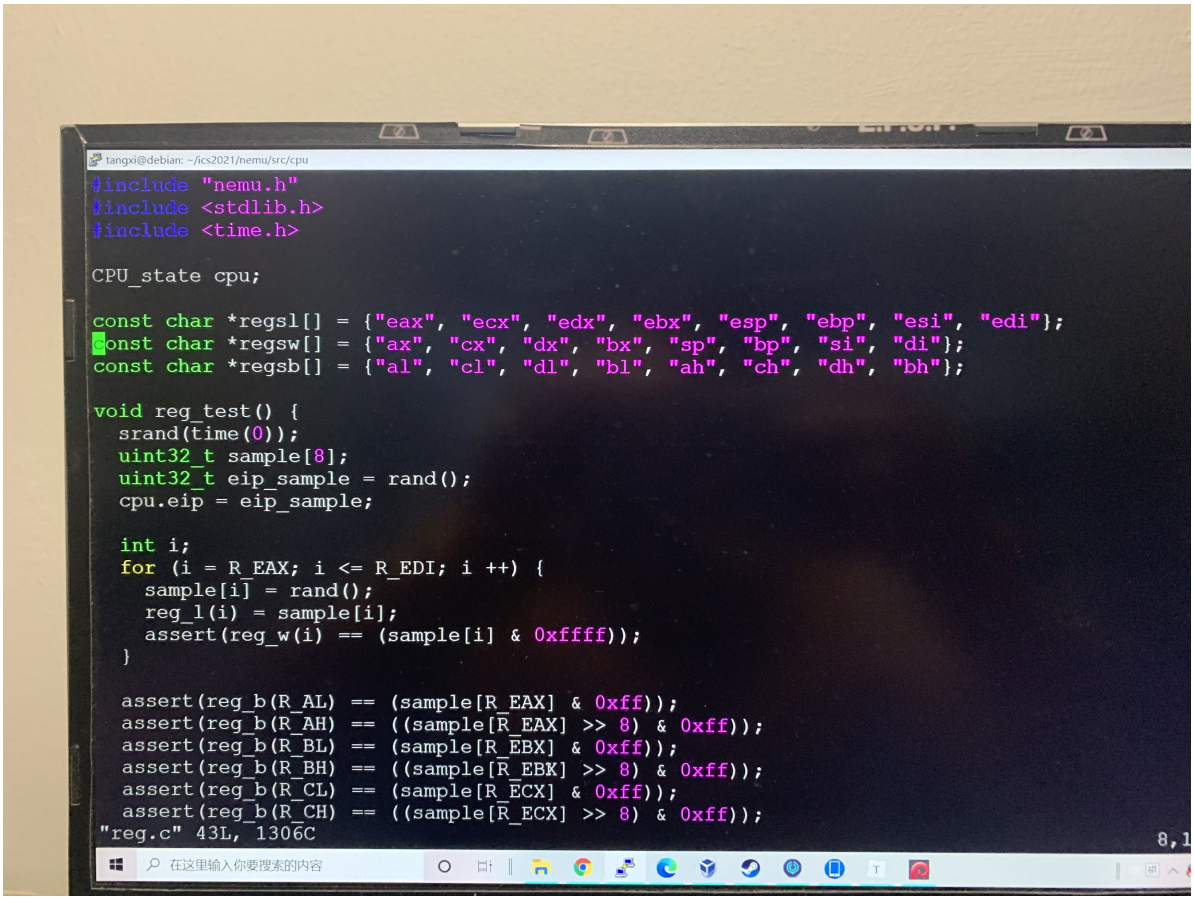
```
For help, type "help"
(nemu) si 5
  100000:   b8 34 12 00 00                      movl $0x1234,%eax
  100005:   b9 27 00 10 00                      movl $0x100027,%ecx
  10000a:   89 01                               movl %eax,(%ecx)
  10000c:   66 c7 41 04 01 00                   movw $0x1,0x4(%ecx)
  100012:   bb 02 00 00 00                      movl $0x2,%ebx
(nemu) si 10
  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) q
tangxi@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: 22:54:18, Mar 31 2021
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)
(nemu) q
tangxi@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: 22:54:18, Mar 31 2021
For help, type "help"
(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)
(nemu)
```

虽然只有7条指令，但应该是可以输出多于10指令的，测试结果正确。

任务3：实现打印寄存器功能

往cmd_table[]中添加info命令，参考i386手册和源代码注释。

```
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.
/
```



源代码：

```
static void dum_regs(){
    int i;
    for(i=R_EAX; i<=R_EDI; i++){
        printf("%s:0x%08x\n",regsl[i],cpu.gpr[i]._32, cpu.gpr[i]._32);
    }
    for(i=R_EAX; i<=R_EAX; i++){
        printf("%s:0x%08x\n",regsw[i],cpu.gpr[i]._16cpu.gpr[i]._16);
    }
    for(i=R_EAX; i<=R_EAX; i++){
        printf("%s:0x%08x\n",regsw[i],cpu.gpr[i]._8[0]cpu.gpr[i]._8[0]);
    }
    printf("eip :0x%08x\n",cpu.eip);
}

static int cmd_info(char *args){
    switch(*args){
        case 'r':dum_regs();
        return 0;
        default:
            return 1;
    }
}
```

执行info r，运行si 5后再执行info r

tangxi@debian: ~/ics2021/nemu

```
+ 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 bui
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 22:54:18, Mar 31 2021
For help, type "help"
(nemu)
(nemu) info r
eax:0x74afff4b 1957691211
ecx:0x76a4fd60 1990524256
edx:0x463db41d 1178448925
ebx:0x16ff54aa 385832106
esp:0x03b59bed 62233581
ebp:0x52cc482a 1389119530
esi:0x0943fdf3 155450867
edi:0x5d7907cc 1568212940
ax:0x0000ff4b 65355
cx:0x0000fd60 64864
dx:0x0000b41d 46109
bx:0x000054aa 21674
sp:0x00009bed 39917
bp:0x0000482a 18474
si:0x0000fdf3 65011
di:0x000007cc 1996
al:0x0000004b 75
cl:0x00000060 96
dl:0x0000001d 29
bl:0x000000aa 170
ah:0x000000ed 237
ch:0x0000002a 42
dh:0x000000f3 243
bh:0x000000cc 204
eip :0x00100000
(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:0x00001234 4660
ecx:0x00100027 1048615
edx:0x463db41d 1178448925
ebx:0x00000002 2
esp:0x03b59bed 62233581
ebp:0x52cc482a 1389119530
esi:0x0943fdf3 155450867
edi:0x5d7907cc 1568212940
ax:0x00001234 4660
cx:0x00000027 39
dx:0x0000b41d 46109
bx:0x00000002 2
sp:0x00009bed 39917
bp:0x0000482a 18474
si:0x0000fdf3 65011
di:0x000007cc 1996
al:0x00000034 52
cl:0x00000027 39
dl:0x0000001d 29
bl:0x00000002 2
ah:0x000000ed 237
ch:0x0000002a 42
dh:0x000000f3 243
bh:0x000000cc 204
eip :0x00100017
(nemu)
```

任务4.1：实现扫描内存功能

对命令进行解析，找出要扫描内存的起始地址后，通过输入要循环的次数来打印出相应的十六进制数据.

```
static int cmd_x(char *args){
    if(args == NULL){
        printf("too few parameter! \n");
        return 1;
    }

    char *arg = strtok(args," ");
    if(arg == NULL){
        printf("too few parameter! \n");
        return 1;
    }
    int  n = atoi(arg);
    char *EXPR = strtok(NULL," ");
    if(EXPR == NULL){
        printf("too few parameter! \n");
        return 1;
    }
    if(strtok(NULL," ")!=NULL){
        printf("too many parameter! \n");
        return 1;
    }
    bool success = true;
    if (success!=true){
        printf("ERRO!!\n");
        return 1;
    }
    char *str;
    vaddr_t addr =  strtol( EXPR,&str,16 );
    for(int i = 0 ; i < n ; i++){
       printf("0x%x ",addr);
       printf("%x ",vaddr_read(addr,4));
       addr+=4;
       printf("\n");
    }

    return 0;
}
```

```
For help, type "help"
(nemu) x 4 0x100000
0x100000 1234b8
0x100004 27b900
0x100008 1890010
0x10000c 441c766
(nemu)
```

任务4.2：转换为字节显示

以4字节为例，用循环语句来使每4个数据打印一次。

修改后的代码：

```c
static int cmd_x(char *args){
    if(args == NULL){
        printf("too few parameter! \n");
        return 1;
    }

    char *arg = strtok(args," ");
    if(arg == NULL){
        printf("too few parameter! \n");
        return 1;
    }
    int  n = atoi(arg);
    char *EXPR = strtok(NULL," ");
    if(EXPR == NULL){
        printf("too few parameter! \n");
        return 1;
    }
    if(strtok(NULL," ")!=NULL){
        printf("too many parameter! \n");
        return 1;
    }
    bool success = true;
    if (success!=true){
        printf("ERRO!!\n");
        return 1;
    }
    char *str;
    vaddr_t addr =  strtol( EXPR,&str,16 );
    for(int i = 0 ; i < n ; i++){
        uint32_t data = vaddr_read(addr + i * 4,4);
        printf("0x%08x  " , addr + i * 4 );
        for(int j =0 ; j < 4 ; j++){
            printf("0x%02x " , data & 0xff);
            data = data >> 8 ;
        }
        printf("\n");
    }

    return 0;
}
```

```
For help, type "help"
(nemu) x 4 0x100000
0x00100000  0xb8 0x34 0x12 0x00
0x00100004  0x00 0xb9 0x27 0x00
0x00100008  0x10 0x00 0x89 0x01
0x0010000c  0x66 0xc7 0x41 0x04
(nemu)
```

测试结果正确。

遇到的问题及解决办法

1.多步执行时修改了输出上限还是只有7行指令，看不出来是否能输出1000000行。

解决办法：询问了助教，暂时默认可以输出。

2.用info r打印寄存器时用整型i做循环变量时一直报错。

解决办法：后来用vim打开reg.c文件，找到了带有寄存器名称的数组，用正确的数组元素名称代入解决。

实验心得

经过本次的实验，我学会了在 Linux环境下用vim等工具来编写，调试程序。通过讲义与实践相结合的方式巩固了寄存器和计算机数值存储的相关知识。虽然在新的环境下还有些不适应，请教了许多同学和老师，但是完成任务的时候能切实感受到自己从刚开始看的时候的一无所知到现在能独立完成的成长。

其他备注

无