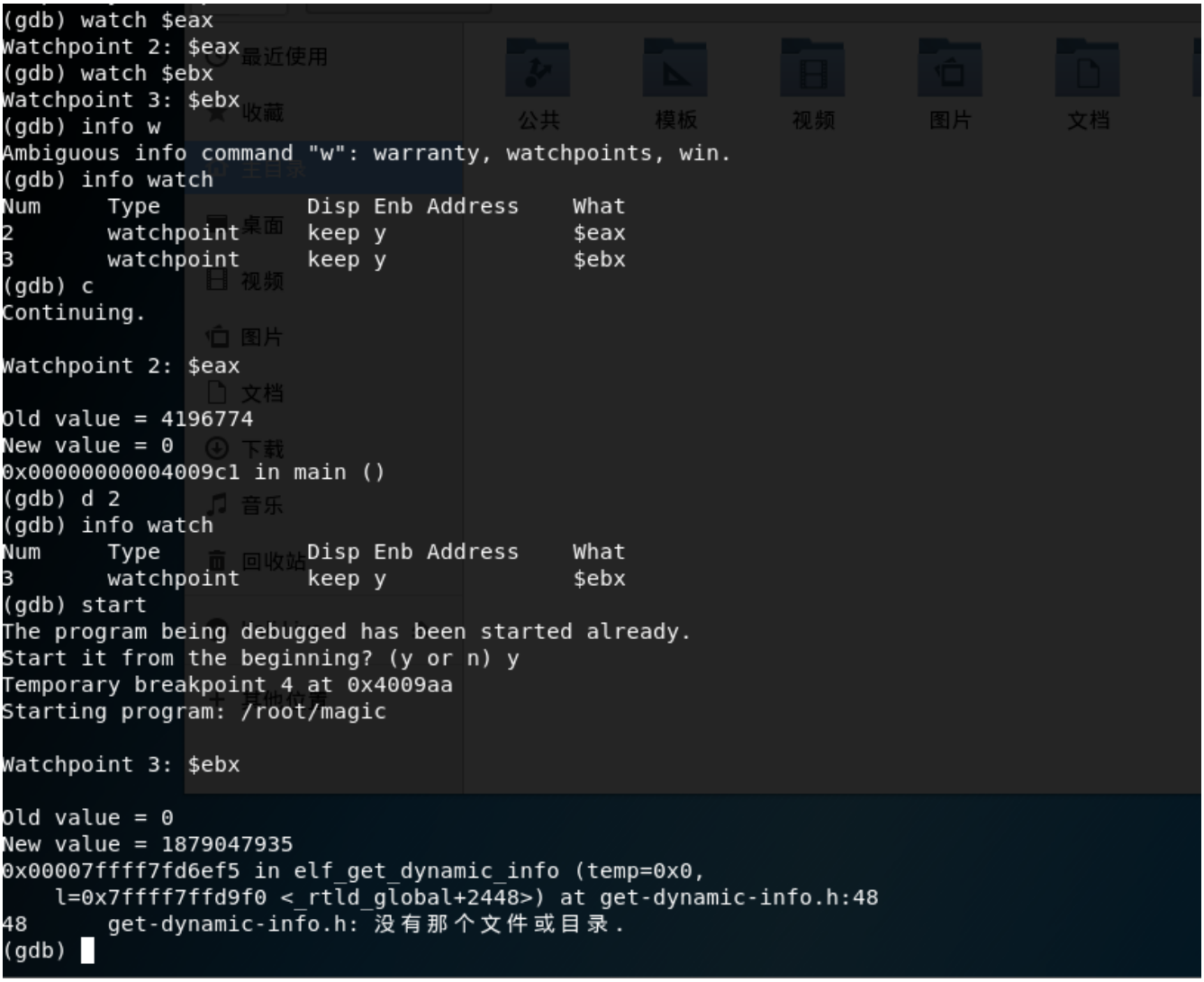
南京航空航天大学《计算机组成原理II课程设计》报告

- 姓名：曹伟思
- 班级：1617302
- 学号：161730213
- 报告阶段：PA1.3
- 完成日期：2019.4.18
- 本次实验，我完成了所有内容并实现软件断点。

目录

思考题

体验监视点



科学起名

C语言stdlib.h中定义了free函数,所以全局变量名不能叫free.

温故而知新

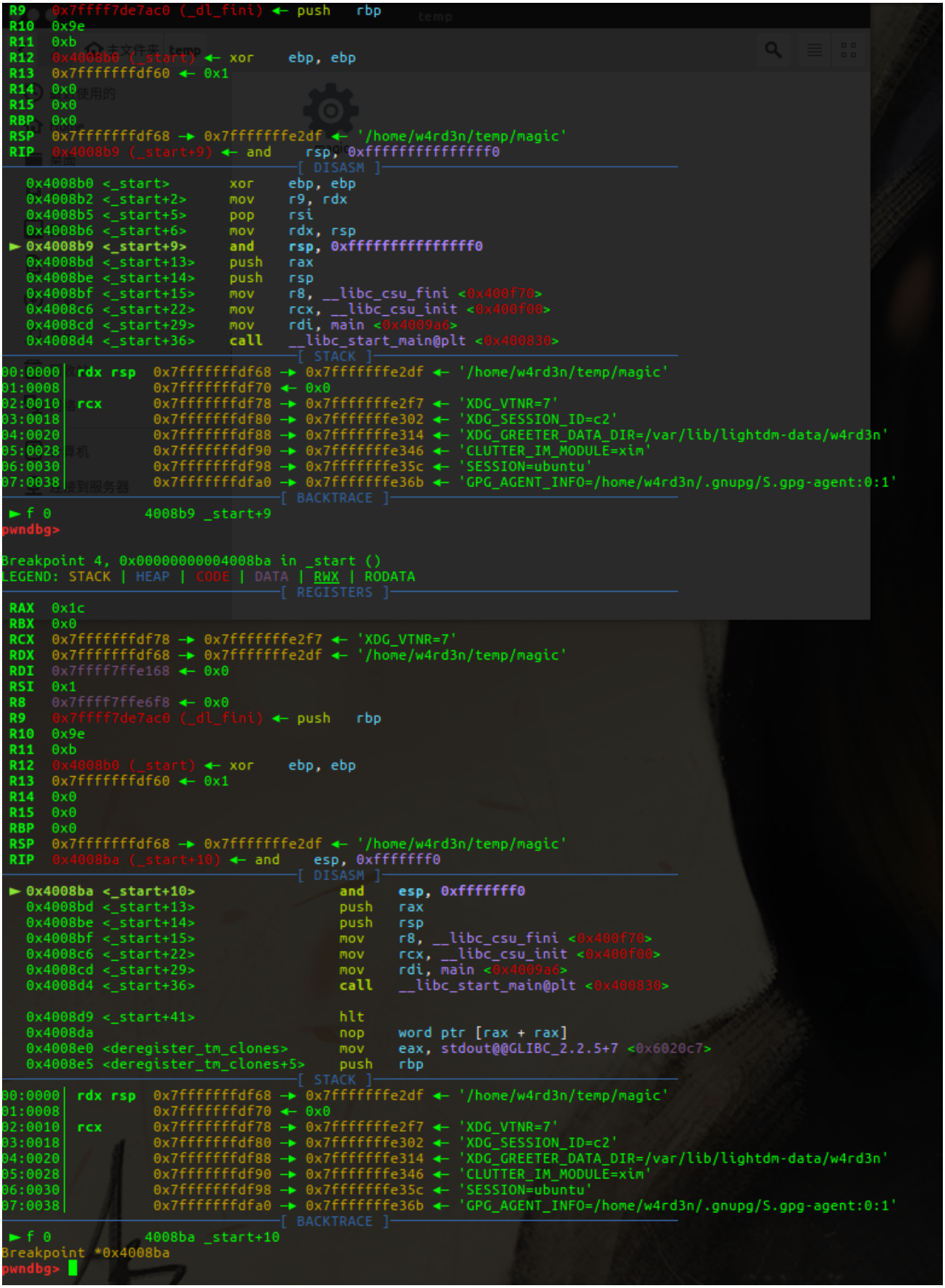
这里的static主要声明了文件作用域,使全局变量变成静态全局变量,全局变量和静态全局变量在存储方式上并无不同.区别在于非静态全局变量的作用域是整个源程序,当一个源程序由多个源文件组成时,非静态的全局变量在各个源文件中都是有效的.而静态全局变量则限制了其作用域,即只在定义该变量的源文件内有效,在同一源程序的其它源文件中不能使用它.

一点也不能长?

这种单字节形式很有价值,因为它可用于用断点替换任何指令的第一个字节,包括其他一个字节指令,而不会覆盖其他代码.

"随心所欲"的断点

如果不是指令第一个字节的地址可能会产生新的指令.


R9 0x7ffff7de7ac0 (_dl_fini) ← push rbp
R10 0x9e
R11 0xb
R12 0x4008b0 (_start) ← xor ebp, ebp
R13 0x7fffffffdf60 ← 0x1
R14 0x0
R15 0x0
RBP 0x0
RSP 0x7fffffffdf68 → 0x7fffffffe2df ← '/home/w4rd3n/temp/magic'
RIP 0x4008b9 (_start+9) ← and rsp, 0xfffffffffffffff0
[DISASM]
0x4008b0 <_start> xor ebp, ebp
0x4008b2 <_start+2> mov r9, rdx
0x4008b5 <_start+5> pop rsi
0x4008b6 <_start+6> mov rdx, rsp
► 0x4008b9 <_start+9> and rsp, 0xfffffffffffffff0
0x4008bd <_start+13> push rax
0x4008be <_start+14> push rsp
0x4008bf <_start+15> mov r8, __libc_csu_fini <0x400f70>
0x4008c6 <_start+22> mov rcx, __libc_csu_init <0x400f00>
0x4008cd <_start+29> mov rdi, main <0x4009a6>
0x4008d4 <_start+36> call __libc_start_main@plt <0x400830>
[STACK]
00:0000 rdx rsp 0x7fffffffdf68 → 0x7fffffffe2df ← '/home/w4rd3n/temp/magic'
01:0008 0x7fffffffdf70 ← 0x0
02:0010 rcx 0x7fffffffdf78 → 0x7fffffffe2f7 ← 'XDG_VTNR=7'
03:0018 0x7fffffffdf80 → 0x7fffffffe302 ← 'XDG_SESSION_ID=c2'
04:0020 0x7fffffffdf88 → 0x7fffffffe314 ← 'XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/w4rd3n'
05:0028 0x7fffffffdf90 → 0x7fffffffe346 ← 'CLUTTER_IM_MODULE=xim'
06:0030 0x7fffffffdf98 → 0x7fffffffe35c ← 'SESSION=ubuntu'
07:0038 0x7fffffffdfa0 → 0x7fffffffe36b ← 'GPG_AGENT_INFO=/home/w4rd3n/.gnupg/S.gpg-agent:0:1'
[BACKTRACE]
► f 0 4008b9 _start+9
pwndbg>
Breakpoint 4, 0x00000000004008ba in _start ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
[REGISTERS]
RAX 0x1c
RBX 0x0
RCX 0x7fffffffdf78 → 0x7fffffffe2f7 ← 'XDG_VTNR=7'
RDX 0x7fffffffdf68 → 0x7fffffffe2df ← '/home/w4rd3n/temp/magic'
RDI 0x7ffff7ffe168 ← 0x0
RSI 0x1
R8 0x7ffff7ffe6f8 ← 0x0
R9 0x7ffff7de7ac0 (_dl_fini) ← push rbp
R10 0x9e
R11 0xb
R12 0x4008b0 (_start) ← xor ebp, ebp
R13 0x7fffffffdf60 ← 0x1
R14 0x0
R15 0x0
RBP 0x0
RSP 0x7fffffffdf68 → 0x7fffffffe2df ← '/home/w4rd3n/temp/magic'
RIP 0x4008ba (_start+10) ← and esp, 0xfffffff0
[DISASM]
► 0x4008ba <_start+10> and esp, 0xfffffff0
0x4008bd <_start+13> push rax
0x4008be <_start+14> push rsp
0x4008bf <_start+15> mov r8, __libc_csu_fini <0x400f70>
0x4008c6 <_start+22> mov rcx, __libc_csu_init <0x400f00>
0x4008cd <_start+29> mov rdi, main <0x4009a6>
0x4008d4 <_start+36> call __libc_start_main@plt <0x400830>
0x4008d9 <_start+41> hlt
0x4008da nop word ptr [rax + rax]
0x4008e0 <deregister_tm_clones> mov eax, stdout@@GLIBC_2.2.5+7 <0x6020c7>
0x4008e5 <deregister_tm_clones+5> push rbp
[STACK]
00:0000 rdx rsp 0x7fffffffdf68 → 0x7fffffffe2df ← '/home/w4rd3n/temp/magic'
01:0008 0x7fffffffdf70 ← 0x0
02:0010 rcx 0x7fffffffdf78 → 0x7fffffffe2f7 ← 'XDG_VTNR=7'
03:0018 0x7fffffffdf80 → 0x7fffffffe302 ← 'XDG_SESSION_ID=c2'
04:0020 0x7fffffffdf88 → 0x7fffffffe314 ← 'XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/w4rd3n'
05:0028 0x7fffffffdf90 → 0x7fffffffe346 ← 'CLUTTER_IM_MODULE=xim'
06:0030 0x7fffffffdf98 → 0x7fffffffe35c ← 'SESSION=ubuntu'
07:0038 0x7fffffffdfa0 → 0x7fffffffe36b ← 'GPG_AGENT_INFO=/home/w4rd3n/.gnupg/S.gpg-agent:0:1'
[BACKTRACE]
► f 0 4008ba _start+10
Breakpoint *0x4008ba
pwndbg>

NEMU的前世今生

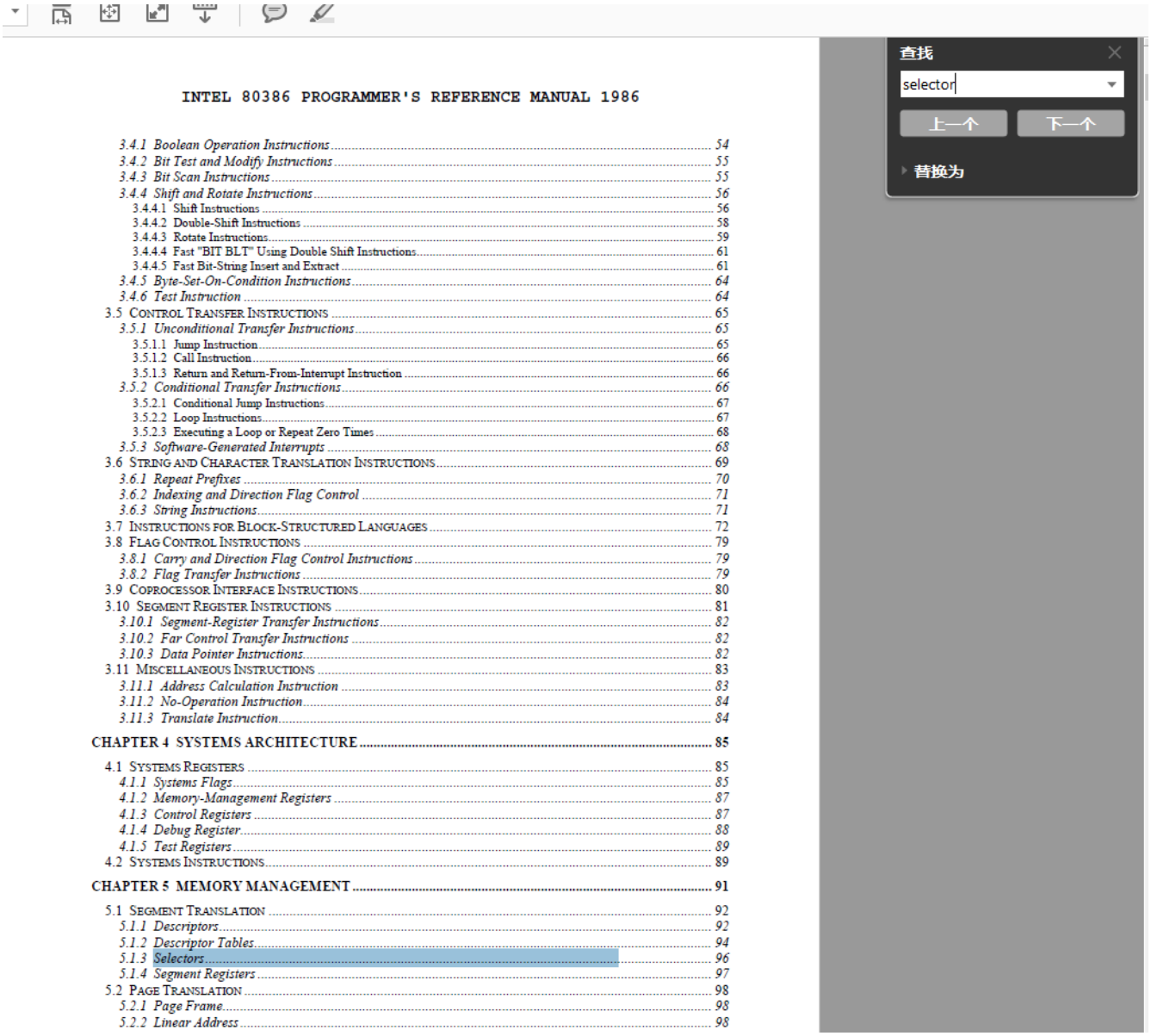
模拟器(Emulator)是用于模拟一个系统内部并实现其功能的软件,而调试器(Debugger)只是单纯的使用操作系统包括硬件本身提供的接口.

调试器能够随心所欲的停止程序的执行,主要通过软件断点和硬件断点两种方式.

软件断点在X86系统中就是指令INT 3,它的二进制代码opcode是0xCC.当程序执行到INT 3指令时,会引发软件中断.操作系统的INT 3中断处理器会寻找注册在该进程上的调试处理程序. 硬件断点,X86系统提供8个调试寄存器(DR0~DR7)和2个MSR用于硬件调试.其中前四个DR0~DR3是硬件断点寄存器,可以放入内存地址或者IO地址,还可以设置为执行,修改等条件.CPU在执行的到这里并满足条件会自动停下来.

硬件断点十分强大,但缺点是只有四个,这也是为什么所有调试器的硬件断点只能设置4个原因.我们在调试不能修改的ROM时,只能选择这个,所以要省着点用,在一般情况下还是尽量选择软件断点.

尝试通过目录定位关注的问题



理解基础设施

调试是找出bug最有效也是最轻松的方式,信息是最重要的武器.

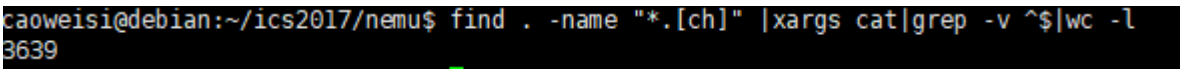
查阅i386手册

答案:

1. 首先从目录定位到章节2.3.4:Flags Register,在子部分Status Flags中找到定义:There are instructions to set, clear, and complement CF before execution of an arithmetic instruction. Refer to Appendix C for definition of each status flag.(p34)
2. 首先从目录定位到章节17.2.1:ModR/M and SIB Bytes,在子部分The ModR/M byte contains three fields of information中找到定义.(p241-p242)
3. 关键词指令和mov定位到章节17.2.2.11:Instruction Set Detail的子部分下.(p345-p351)

shell命令

```
find . -name "*.[ch]" |xargs cat|grep -v ^$|wc -l
#先获取需要统计的文件名再使用xargs过滤器当做参数传给cat,将输出内容传给grep过滤最后使用wc -l统计行数.
```



使用man

-Werror:Make all warnings into errors.-Wall:This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros. This also enables some language-specific warnings described in C++ Dialect Options and Objective-C and Objective-C++ Dialect Options.

使用man的/string搜索功能检索.

实验内容

任务1：实现监视点结构体

修改watchpoint.h即可,后面实现软件断点时添加了type变量,实现方法时增加了引用声明.

watchpoint.h.

```
#ifndef __WATCHPOINT_H__
#define __WATCHPOINT_H__

#include "common.h"

typedef struct watchpoint
{
  int NO;
  struct watchpoint *next;
  char expr[32];
  uint32_t new_val;
  uint32_t old_val;
  uint8_t type;
  //0 is watchpoint, 1 is breakpoint.
} WP;

WP *new_wp();
void free_wp(WP *wp);
int set_watchpoint(char *e);
```

```
bool delete_watchpoint(int NO);
void list_watchpoint(void);
WP *scan_watchpoint(void);
int set_breakpoint(char *e);
void recover_int3();

#endif
```

任务2：实现监视点池的管理

基础的链表插入,删除操作.

```
WP *new_wp()
{
  if (!free_)
    assert(0);
  WP *last = NULL;
  WP *now = free_;
  while (now->next)
  {
    last = now;
    now = now->next;
  }
  if (last)
    last->next = NULL;
  else
    free_ = NULL;

  last = head;
  if (last)
  {
    while (last->next)
    {
      last = last->next;
    }
    last->next = now;
  }
  else
  {
    head = now;
  }

  return now;
}

void free_wp(WP *wp)
{
  WP *now = head;
  if (head == wp)
  {
    head = head->next;
    wp->next = NULL;
    now = free_;
    if (!free_)
    {
      free_ = wp;
    }
    else
    {
      while (now->next)
```

```
      {
        now = now->next;
      }
      now->next = wp;
    }
  }
  else
  {
    while (now && now->next != wp)
    {
      now = now->next;
    }
    if (now)
    {
      now->next = wp->next;
      wp->next = NULL;
      now = free_;
      if (!free_)
      {
        free_ = wp;
      }
      else
      {
        while (now->next)
        {
          now = now->next;
        }
        now->next = wp;
      }
    }
  }
}
```

任务3：将监视点加入调试器功能

修改ui.c.

```
static int cmd_w(char *args);

static int cmd_b(char *args);

static int cmd_d(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},
    {"si", "si N:Single-step execution N, no N mean 1", cmd_si},
    {"info", "info r:print register status, info w:list watchpoint information", cmd_info},
    {"x", "x N EXPR:Find the value of EXPR as the starting memory address, output N
consecutive 4 bytes in hexadecimal form", cmd_x},
    {"p", "p EXPR:output EXPR's value", cmd_p},
    {"w", "w EXPR:set a watchpoint for EXPR", cmd_w},
    {"b", "b EXPR:set a breakpoint for eip as the value of EXPR", cmd_b},
    {"d", "d NO:delete a watchpoint for NO", cmd_d},
```

```
};

...

static int cmd_info(char *args)
{
  char *arg = strtok(NULL, " ");

  if (arg == NULL)
  {
    //code
  }
  else if (strcmp(arg, "r") == 0)
  {
    //code
  }
  else if (strcmp(arg, "w") == 0)
  {
    list_watchpoint();
  }
  else
  {
    //code
  }

  return 0;
}

static int cmd_w(char *args)
{
  if (strlen(args) > 31)
  {
    printf("too long EXPR\n");
    return 0;
  }
  int NO = set_watchpoint(args);
  printf("NO %d\nExpr %s\n", NO, args);

  return 0;
}

static int cmd_b(char *args)
{
  bool flag;
  int NO = set_breakpoint(args);
  printf("NO %d\naddress  %#010x\n", NO, expr(args, &flag));

  return 0;
}

static int cmd_d(char *args)
{
  int NO = atoi(args);
  delete_watchpoint(NO);

  return 0;
}
```

具体功能实现与截图在任务4.

任务4：实现监视点

基础的遍历链表操作,scan_watchpoint多个监视点一起触发只报警第一个.

```
int set_watchpoint(char *e)
{
  bool flag;
  WP *wp = new_wp();
  wp->old_val = expr(e, &flag);
  wp->type = 0;
  strcpy(wp->expr, e);

  return wp->NO;
}

bool delete_watchpoint(int NO)
{
  WP *wp = head;
  while (wp && wp->NO != NO)
  {
    wp = wp->next;
  }
  if (wp->NO == NO)
  {
    free_wp(wp);
    return true;
  }
  else
  {
    printf("NO %d watchpoint/breakpoint not found\n", NO);
    return false;
  }
}

void list_watchpoint(void)
{
  printf("NO  Expr            Old Value\n");
  WP *now = head;
  while (now)
  {
    if (now->type == 0)
    {
      printf("%2d  %-16s%#010x\n", now->NO, now->expr, now->old_val);
      break;
    }
    now = now->next;
  }
}

WP *scan_watchpoint(void)
{
  bool flag;
  WP *now = head;
  while (now)
  {
    if (now->type)
    {
      if (now->new_val == 1)
      {
        cpu.eip -= 1;
```

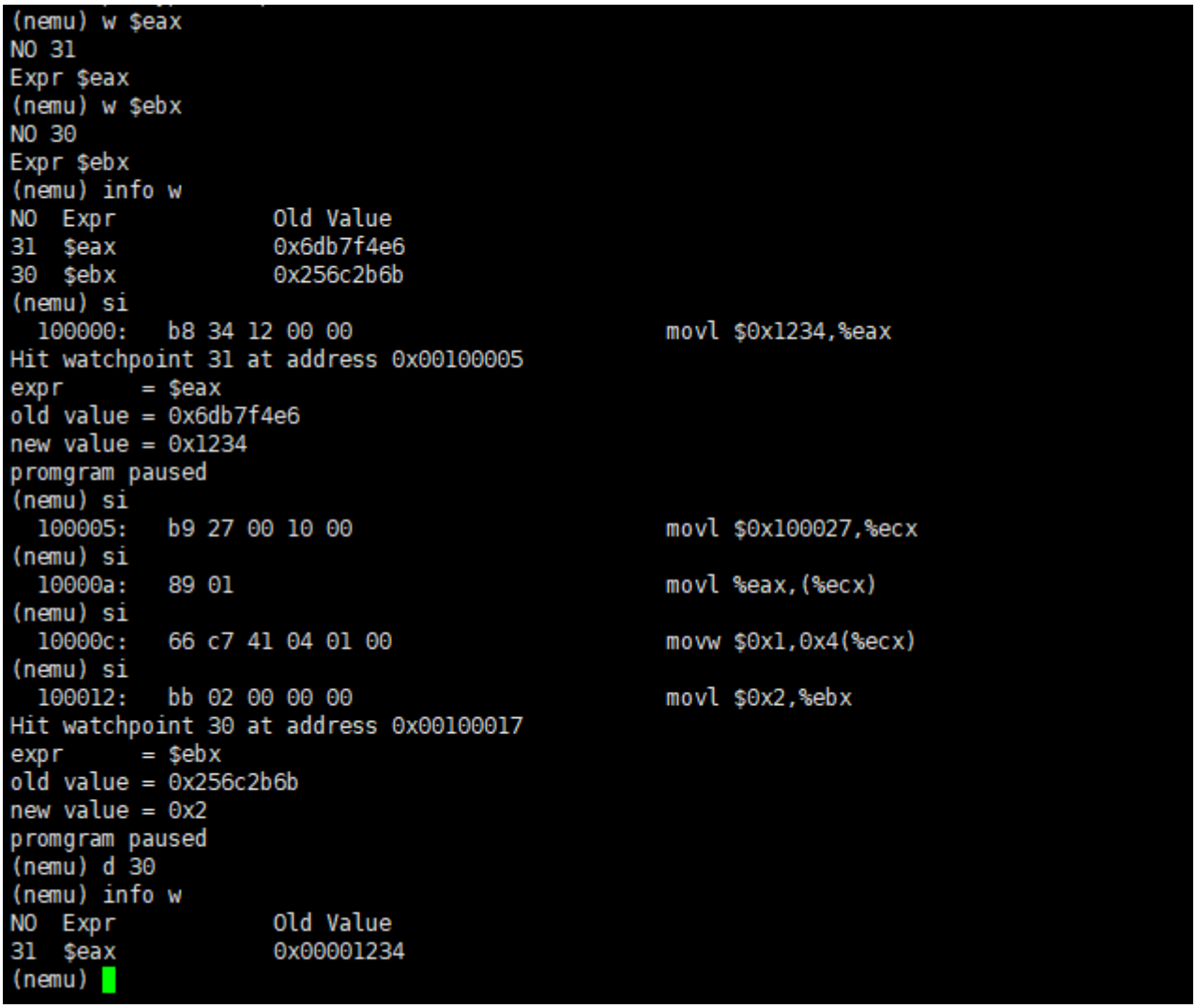
```
        now->new_val = 2;
      }
      else if (now->new_val == 2)
      {
        *now->expr = *(char *)guest_to_host(now ->old_val);
        *(char *)guest_to_host(now->old_val) = 0xcc;
      }
      //恢复断点.
      now = now->next;
      continue;
    }
    else if (now->old_val != expr(now->expr, &flag))
    {
      printf("Hit watchpoint %d at address %#010x\n", now->NO, cpu.eip);
      printf("expr      = %s\n", now->expr);
      printf("old value = %#x\n", now->old_val);
      printf("new value = %#x\n", expr(now->expr, &flag));
      printf("promgram paused\n");
      nemu_state = NEMU_STOP;
      now->old_val = expr(now->expr, &flag);
      return now;
    }
    now = now->next;
  }

  return NULL;
}
```

在cpu-exec.c的cpu_exec函数调用scan_watchpoint.

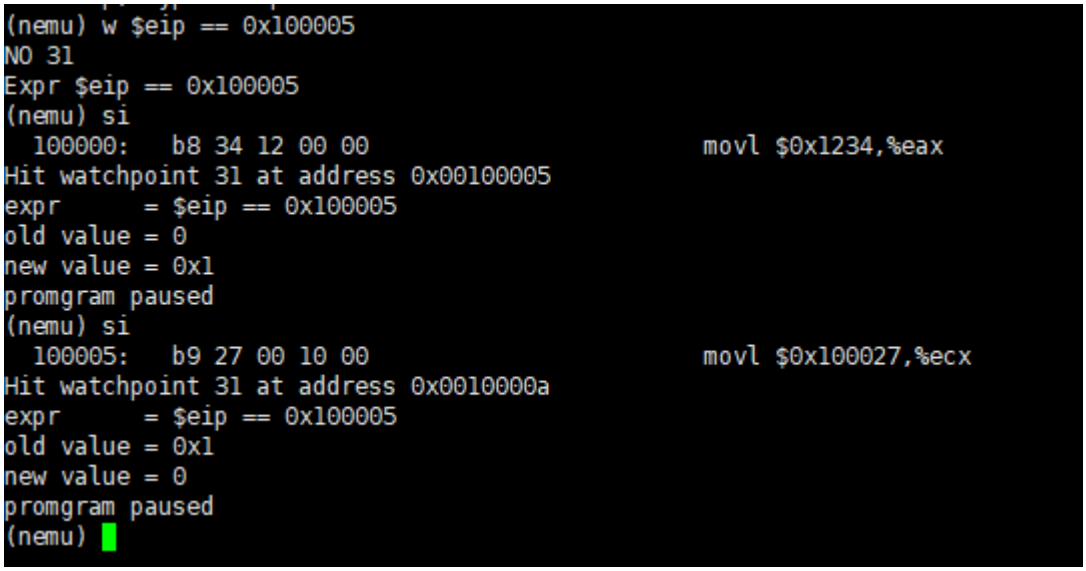
```
void cpu_exec(uint64_t n)
{
  if (nemu_state == NEMU_END)
  {
    printf("Program execution has ended. To restart the program, exit NEMU and run
again.\n");
    return;
  }
  nemu_state = NEMU_RUNNING;

  bool print_flag = n < MAX_INSTR_TO_PRINT;

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

#ifdef DEBUG
    if (scan_watchpoint())
      return;
#endif

#ifdef HAS_IOE
    extern void device_update();
    device_update();
#endif

    if (nemu_state != NEMU_RUNNING)
    {
```

```
      return;
    }
  }

  if (nemu_state == NEMU_RUNNING)
  {
    nemu_state = NEMU_STOP;
  }
}
```

贴一下图.



任务5：使用模拟断点



这种实现方式会在eip到指定值之时和之后都触发一次,十分不方便.

选做任务：实现软件断点

首先为了实现这个单字节操作码,先找到模拟cpu运行的逻辑,也就是cpu_exec,往下找一层找到exec_wrapper.这个函数输出调试信息和运行.

其中运行的语句为:

```
decoding.seq_eip = cpu.eip;
exec_real(&decoding.seq_eip);
```

exec_real这个函数一开始没找到,仔细RTFSC后发现这些exec_开头的函数的函数名都是用C语言宏的#字符组合出来的,而具体的宏如下:

```
#define make_EHelper(name) void concat(exec_, name) (vaddr_t *eip)
```

所以exec_real的定义如下:

```
make_EHelper(real) {
  uint32_t opcode = instr_fetch(eip, 1);
  decoding.opcode = opcode;
  set_width(opcode_table[opcode].width);
  idex(eip, &opcode_table[opcode]);
}
```

执行逻辑在idex里面.

```
static inline void idex(vaddr_t *eip, opcode_entry *e) {
  /* eip is pointing to the byte next to opcode */
  if (e->decode)
    e->decode(eip);
  e->execute(eip);
}
```

opcode_entry的定义.

```
typedef struct {
  DHelper decode;
  EHelper execute;
  int width;
} opcode_entry;
//decode.h: typedef void (*DHelper) (vaddr_t *);
//exec.h: typedef void (*EHelper) (vaddr_t *);
```

然后就找到了对应的数组opcode_table.

找到单字节0xcc的位置,按照宏定义使用EX()宏设置0xcc操作码.(取名为int3)

```
{
    ...
/* 0xcc */  EX(int3), EMPTY, EMPTY, EMPTY,
    ...
}
```

在system.c中定义.

```
make_EHelper(int3) {
  print_asm("breakpoint eip = %0#10x", cpu.eip);
  printf("\33[1;31mnemu: HIT BREAKPOINT\33[0m at eip = %0#10x\n\n", cpu.eip);
  recover_int3();
  nemu_state = NEMU_STOP;
}
```

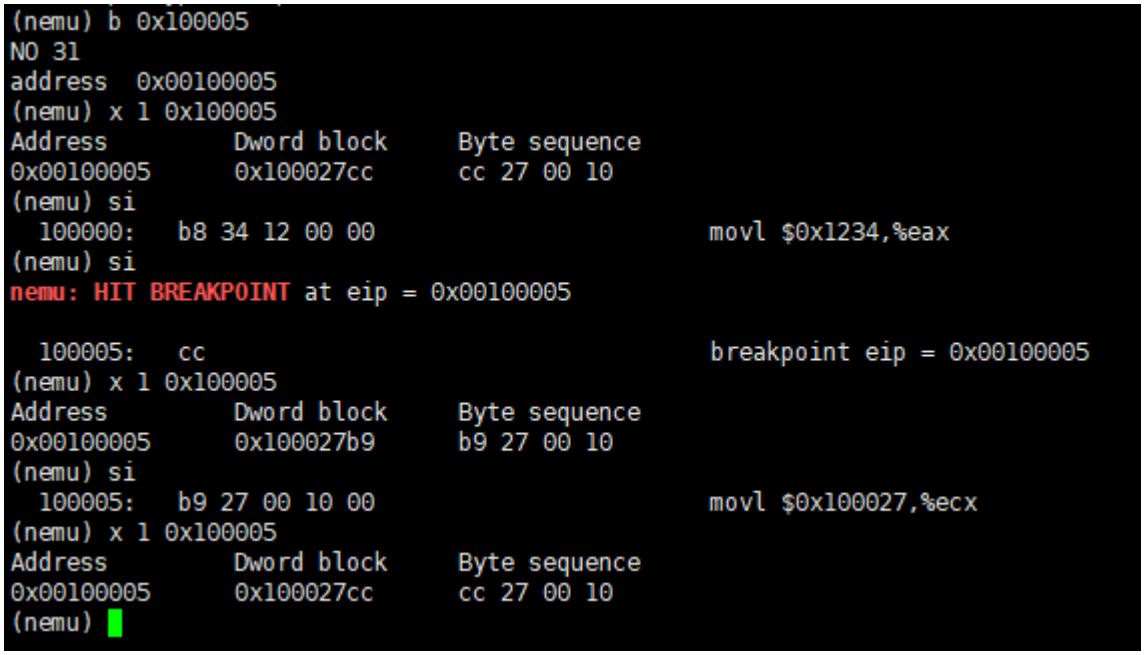
recover_int3()定义于watchpoint.c(因为要操作head指针).

```
void recover_int3()
{
  WP *now = head;
  while (now)
  {
    if (now->type == 1 && now->old_val == cpu.eip)
    {
      *(char *)guest_to_host(now->old_val) = *now->expr;
      now->new_val = 1;
      break;
    }
    now = now->next;
  }
}
```

修改ui.c增加b指令(代码在任务3),增加set_breakpoint于watchpoint.c.

```
int set_breakpoint(char *e)
{
  bool flag;
  WP *wp = new_wp();
  wp->old_val = expr(e, &flag);
  wp->new_val = 0;
  wp->type = 1;
  *wp->expr = *(char *)guest_to_host(wp->old_val);
  *(char *)guest_to_host(wp->old_val) = 0xcc;

  return wp->NO;
}
```

贴一下图.



遇到的问题及解决办法

无

实验心得

深入理解监视点,软件断点和C语言宏.

其他备注

无.