南京航空航天大学 计算机科学与技术系学院 计算机组成原理 课程实验

学号：161630220

姓名：赵维康

1 PA1- 开天辟地的篇章: 最简单的计算机

表达式求值

进入/nemu/src/monitor/debug 目录 在 expr.c 中进行函数编写

词法分析

在 enum 中添加更多 token type,以下是代码截图

```
enum {
  TK_NOTYPE = 256,
TK_EQ,TK_NEQ,TK_AND,TK_OR,TK_MINUS,TK_POINTER,TK_NUMBER,TK_HNUMBER,TK_REGISTER,TK_MARK

  /* TODO: Add more token types */

};
```

在结构体 rules[]添加 more rules，以下是代码截图

```
} rules[] = {

  /* TODO: Add more rules.
   * Pay attention to the precedence level of different rules.
   */

  {" +", TK_NOTYPE},    // spaces
  {"\\+", '+'},         // plus
  {"==", TK_EQ},         // equal
  {"\\b[0-9]+\\b",TK_NUMBER},   //number
  {"\\b0[xX][0-9a-fA-F]+\\b",TK_HNUMBER}, //16 number
  {"\\$[a-zA-Z]+",TK_REGISTER}, //register
  {"\\b[a-zA-Z0-9]+",TK_MARK}, //mark
  {"!=",TK_NEQ}, //not equall
  {"!",'!'}, //not
  {"\\*",'*'}, //mul
  {"/",'/'}, //div
  {"      +",TK_NOTYPE}, //tabs
  {"-",'-'}, //sub
  {"&&",TK_AND}, //and
  {"\\|\\|",TK_OR}, //or
  {"\\(",'('}, //left bracket
  {"\\)",')'}, //right bracket
};
```

完成　　make_token()函数的完整实现，以下是代码截图

```
static bool make_token(char *e) {
  int position = 0;
  int i;
  regmatch_t pmatch;

  nr_token = 0;

  while (e[position] != '\0') {
    /* Try all rules one by one. */
    for (i = 0; i < NR_REGEX; i ++) {
      if (regexec(&re[i], e + position, 1, &pmatch, 0) == 0 && pmatch.rm_so == 0) {
        char *substr_start = e + position;
        int substr_len = pmatch.rm_eo;
        char *tmp = e + position + 1;
        Log("match rules[%d] = \"%s\" at position %d with len %d: %.*s",
            i, rules[i].regex, position, substr_len, substr_len, substr_start);
        position += substr_len;

        /* TODO: Now a new token is recognized with rules[i]. Add codes
         * to record the token in the array `tokens'. For certain types
         * of tokens, some extra actions should be performed.
         */

        switch (rules[i].token_type) {
          case TK_NOTYPE: break;
```

```
        switch (rules[i].token_type) {
          case TK_NOTYPE: break;
          case TK_REGISTER:
                      tokens[nr_token].type=rules[i].token_type;
                      tokens[nr_token].priority=rules[i].priority;
                      strncpy(tokens[nr_token].str,tmp,substr_len-1);
                      tokens[nr_token].str[substr_len-1]='\0';
                      nr_token ++;
                      break;
          default:
                      tokens[nr_token].type=rules[i].token_type;
                      tokens[nr_token].priority=rules[i].priority;
                      strncpy(tokens[nr_token].str,substr_start,substr_len);
                      tokens[nr_token].str[substr_len]='\0';
                      nr_token ++;
        }

        break;
      }
    }

    if (i == NR_REGEX) {
      printf("no match at position %d\n%s\n%*.s^\n", position, e, position, "");
      return false;
    }
  }


  return true;
}
```

switch 语句中，如果是空格或者 **tab**,直接跳出，如果是 **register** 类型，则拷贝到 **tokens[]**数组中

接下来测试词法分析成功与否

执行 **make** 与 **make run** 命令截图

```
root@zhaoweikang:/home/zhaoweikang/ics2017/nemu# make
+ CC src/monitor/debug/expr.c
+ LD build/nemu
root@zhaoweikang:/home/zhaoweikang/ics2017/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 i
mage.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 12:38:57, Mar 14 2018
For help, type "help"
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x00100026
```

从输出来看，说明词法分析成功实现

递归求值

在这一部分，根据讲义，需要实现 **check_parentheses()** 、**dominant_operator()** 、**eval()**

先实现 **check_parentheses()**函数，也就是检查左右括号是否匹配的问题，以下是实现代码截图

```
bool check_parentheses(int p,int q){
     int i;
     if(tokens[p].type=='('&&tokens[q].type==')'){
          int pc = 0,qc = 0;
          for(i = p + 1;i < q;i ++){
                if(tokens[i].type=='(')pc ++;
                if(tokens[i].type==')')qc ++;
                if(qc > pc)return false;
                      }
                if(pc == qc)return true;
                   }
    return false;
}
```

下面实现 **dominant_operator()**函数，即找到表达式中优先级最低的，以下是实现代码截图

```
int dominant_operator(int p,int q)
{
    int i,j;
    int min_priority = 10;
    int oper=p;
    for(i = p;i <= q;i ++)
    {
       if(tokens[i].type==TK_NUMBER||tokens[i].type==TK_HNUMBER||tokens[i].type==TK_REGISTER||
tokens[i].type==TK_MARK)continue;
       int cnt=0;
       bool key=true;
      for(j=i-1;j>=p;j--)
      {
           if(tokens[j].type=='('&&!cnt)
           {
            key=false;break;
           }
           if(tokens[j].type=='(')cnt--;
           if(tokens[j].type==')')cnt++;
       }
       if(!key)continue;
       if(token[i].priority<=min_priority)
       {min_priority=tokens[i].priority;
        oper=i;
       }
   }
   return oper;
}
```

下面完成对求值函数 **eval()**的实现，在此函数中会对指针解应用与乘号、减号与负号的情况进行处理，　　以下是实现代码截图

```
uint32_t eval(int p,int q) {
        if (p> q){Assert (p>q," something unexpected!\n");return 0;}

        if (p == q) {

        uint32_t num = 0;

        if (tokens[p].type == TK_NUMBER)

                sscanf(tokens[p].str,"%d",&num);
        if (tokens[p].type == TK_HNUMBER)

                sscanf(tokens[p].str,"%x",&num);

        if (tokens[p].type == TK_REGISTER)

                {

                 if (strlen (tokens[p].str) == 3) {

                 int i;

                 for (i = R_EAX; i <= R_EDI; i ++)


                for (i = R_EAX; i <= R_EDI; i ++)

                        if (strcmp (tokens[p].str,regsl[i]) == 0)break;

                if (i > R_EDI)

                        if (strcmp (tokens[p].str,"eip") == 0)

                                num = cpu.eip;

                        else Assert (1,"no this register!\n");

                else num = reg_l(i);

                }

                else if (strlen (tokens[p].str) == 2) {

                if (tokens[p].str[1] == 'x' || tokens[p].str[1] == 'p' || tokens[p].str[1]
== 'i') {

                        int i;

                        for (i = R_AX; i <= R_DI; i ++)

                                if (strcmp (tokens[p].str,regsw[i]) == 0)break;
```

```
                                num = reg_w(i);

                        }

                        else if (tokens[p].str[1] == 'l' || tokens[p].str[1] == 'h') {

                                int i;

                                for (i = R_AL; i <= R_BH; i ++)

                                        if (strcmp (tokens[p].str,regsb[i]) == 0)break;

                                num = reg_b(i);

                        }

                        else assert (1);

                        }

                }
        return num;
        }

         else if (check_parentheses (p,q) == true){return eval (p + 1,q - 1);}

         else {
```

```
                int op = dominant_operator (p,q);

                if ((p == op )||( tokens [op].type == TK_POINTOR) || (tokens [op].type ==TK_MINUS)
|| (tokens [op].type == '!'))

                {

                        uint32_t val = eval (p + 1,q);
                        switch (tokens[p].type)

                        {

                                case TK_POINTOR: return vaddr_read (val,4);

                                case TK_MINUS:return -val;

                                case '!':return !val;

                                default :Assert (1,"default\n");

                        }

                }


                uint32_t val1 = eval (p,op - 1);
```

```
                uint32_t val2 = eval (op + 1,q);
                switch (tokens[op].type)

                {

                        case '+':return val1 + val2;

                        case '-':return val1 - val2;

                        case '*':return val1 * val2;

                        case '/':return val1 / val2;

                        case TK_EQ:return val1 == val2;

                        case TK_NEQ:return val1 != val2;

                        case TK_AND:return val1 && val2;

                        case TK_OR:return val1 || val2;

                        default:assert (1);

                        break;

                }

        assert (1);

        return -123456;

}
```

在此函数中，分别对 p>q（异常，使用 assert（））、p==q（这个条件下，对类型为十进制数、十六进制数、寄存器类型进行了讨论）、左右括号匹配（即 check_parentheses()）、dominant oprater 等情况进行了处理，此外，switch 语句也使得代码书写起来更方便。

实现代码之后，执行 make 命令，结果如下

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo make
+ CC src/monitor/debug/expr.c
+ LD build/nemu
```

继续执行 make run 命令，结果如下

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo 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: 12:38:57, Mar 14 2018
For help, type "help"
(nemu)
```

此结果说明 eval()函数实现正确

由于调试中的表达式求值这一部分只有一个函数 expr()需要实现，所以，接下来实现此函数，代码如下

```
uint32_t expr (char *e, bool *success) {

        if(!make_token(e)) {

                *success = false;

                return 0;

        }

        int i;

        for (i = 0;i < nr_token; i ++) {

                if (tokens[i].type == '*' && (i == 0 || (tokens[i - 1].type != TK_NUMBER && tokens
[i - 1].type != TK_HNUMBER && tokens[i - 1].type != TK_REGISTER && tokens[i - 1].type !=TK_MARK &&
tokens[i - 1].type !=')'))) {

                        tokens[i].type = TK_POINTOR;



                        tokens[i].priority = 6;

                }

                if (tokens[i].type == '-' && (i == 0 || (tokens[i - 1].type != TK_NUMBER && tokens
[i - 1].type != TK_HNUMBER && tokens[i - 1].type != TK_REGISTER && tokens[i - 1].type != TK_MARK
&& tokens[i - 1].type !=')'))) {

                        tokens[i].type =TK_MINUS;

                        tokens[i].priority = 6;

                }

        }

        /* TODO: Insert codes to evaluate the expression. */

        *success = true;

        return eval (0,nr_token-1);

}
```

expr()函数，对指针解引用，负号的情况进行了处理，最后返回表达式的值

实现代码之后，执行 make 命令，结果如下

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo make
+ CC src/monitor/debug/expr.c
+ LD build/nemu
```

继续执行 make run 命令，结果如下

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo 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: 12:38:57, Mar 14 2018
For help, type "help"
(nemu)
```

此结果说明 expr()函数实现正确

下面实现用于实现表达式求值的 p 命令

进入/nemu/src/monitor/debug/ui.c 文件

找到 cmd_table[]结构体数组，添加命令 p,如图

```
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 */
  {"si","Single step",cmd_si},
  {"info","dump informations with option:r(register)",cmd_info},
  {"x","dump memory: x length addr",cmd_x},
  {"p","Expression evaluation",cmd_p},
};
```

再编写 cmd_p()函数，调用 expr()函数，返回表达式的值，并用 printf()输出，如图

```
static int cmd_p(char *args){
        uint32_t num;
        bool suc;
        num=expr(args,&suc);
        if(suc)
             printf("0x%08x:\t%d\n",num,num);
        //else assert(0);
    return 0;
}
```

同样，执行 make、make run 命令，结果如图

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo make
+ CC src/monitor/debug/ui.c
+ LD build/nemu
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo 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: 12:38:57, Mar 14 2018
For help, type "help"
(nemu)
```

说明，以上实现正确

下面，先单步执行一下，如图

```
(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)
```

再打印寄存器状态

```
(nemu) info r
eax :0x00000000 0
ecx :0x00100027 1048615
edx :0x3b3ada47 993712711
ebx :0x00000002 2
esp :0x70570763 1884751715
ebp :0x07c9722d 130642477
esi :0x465cad69 1180478825
eip: 0x00100027 1048615
```

表达式求值

```
(nemu) p 0xc0100000+($eax+5)*4-*(
.gitignore    Makefile.git  build/        runall.sh
Makefile      README.md     include/      src/
(nemu) p 0xc0100000+($eax+5)*4-*($ebp+8)
[src/monitor/debug/expr.c,101,make_token] match rules[4] = "\b0[xX][0-9a-fA-F]+\
b" at position 0 with len 10: 0xc0100000
[src/monitor/debug/expr.c,101,make_token] match rules[3] = "\b[0-9]+\b" at posit
ion 20 with len 1: 4
[src/monitor/debug/expr.c,101,make_token] match rules[9] = "\*" at position 22 w
ith len 1: *
[src/monitor/debug/expr.c,101,make_token] match rules[5] = "\$[a-zA-Z]+" at posi
tion 24 with len 4: $ebp
[src/monitor/debug/expr.c,101,make_token] match rules[6] = "\b[a-zA-Z0-9]+" at p
osition 32 with len 2: DM
0x8c751000:     -1938485248
```

扫描内存

```
(nemu) x 10 0x8c751000

0x8c751000: 0x8c751000 0x8c751001 0x8c751002 0x8c751003 0x8c751004 0x8c751005 0x
8c751006 0x8c751007 0x8c751008 0x8c751009
```

表达式求值（负数求值）

```
(nemu) p (1+(-1))
[src/monitor/debug/expr.c,101,make_token] match rules[15] = "\(" at position 0 w
ith len 1: (
[src/monitor/debug/expr.c,101,make_token] match rules[1] = "\+" at position 2 wi
th len 1: +
[src/monitor/debug/expr.c,101,make_token] match rules[12] = "-" at position 4 wi
th len 1: -
[src/monitor/debug/expr.c,101,make_token] match rules[16] = "\)" at position 6 w
ith len 1: )
0x00000000:     0
```

扫描内存

```
(nemu) x 10 0x00000000

0x00000000: 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
```

思考题：温故而知新(2) 框架代码中定义 wp_pool 等变量的时候使用了关键 static ,static 在此处的含义是什么?为什么要在此处使用它?

答：我认为 static 关键字所修饰的静态局部变量，利用了 static 的“记忆性”这一特性，下一次使用时，能保存上一次的值。

git 记录

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo git status
位于分支 pa1
无文件要提交，干净的工作区
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo git add .
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo git commit --allow-empty
[pa1 4366907] fix for pa1.2
zhaoweikang@zhaoweikang:~/ics2017/nemu$ git log
commit 43669078df7e123d1f25cd1eea6f234da38eaf3c
Author: 161630220-Zhao Weikang <2875206963@qq.com>
Date:   Sat Mar 24 16:17:46 2018 +0800

    fix for pa1.2

commit ff1aa9a19748f7aad81f76604996104421470fc5
Author: tracer-ics2017 <tracer@njuics.org>
Date:   Sat Mar 24 16:14:45 2018 +0800

    >  run
    161630220
    root
    Linux zhaoweikang 4.9.0-6-686-pae #1 SMP Debian 4.9.82-1+deb9u3 (2018-03-02)
 i686 GNU/Linux
 i686 GNU/Linux
     16:14:45 up  6:33,  1 user,  load average: 0.08, 0.02, 0.00
    61b78cb776bc6557a067a76321a397b476d049fd

commit c8b77d1557804a33b25fb7247fd1be551f813bc1
Author: tracer-ics2017 <tracer@njuics.org>
Date:   Sat Mar 24 16:11:41 2018 +0800

 _  >  run
```

监视点

在这一部分，首先进入目录　　nemu/include/monitor/watchpoint.h，对监视点的结构体进行补充，以实现监视点的功能，我在此结构体中添加了记录表达式值的变量、保存表达式的字符数组，此外在这个 watchpoint.h 文件中还添加了要在 watchpoint.c 实现的几个函数声明，代码如下

```c
#ifndef __WATCHPOINT_H__
#define __WATCHPOINT_H__

#include "common.h"

typedef struct watchpoint {
  int NO;
  struct watchpoint *next;

  /* TODO: Add more members if necessary */
  uint32_t val;
  char expr[32];
  int b;

} WP;
bool check_wp();
WP* new_wp();
void free_wp(WP *);
void info_wp();
void delete_wp(int );
#endif
```

然后执行 make 及 make run 命令，结果如图

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo make
[sudo] zhaoweikang 的密码：
+ CC src/monitor/debug/watchpoint.c
+ CC src/monitor/debug/ui.c
+ LD build/nemu
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo 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: 12:38:57, Mar 14 2018
For help, type "help"
(nemu)
```

说明上述代码没问题

下面进入 nemu/src/monitor/watchpoint.c 文件，编写函数 new_wp()、free_wp()，new_wp() 从 free_链表中返回一个空闲的监视点结构, free_wp()将 wp 归还到 free_链表中,此外，再添加几个函数，分别是 check_wp()，用于查看监视点、查看表达式的旧值，以及新值，delete_wp()，此函数用来删除监视点，释放相应的监视点结构，info_wp()，此函数用来打印使用中的监视点信息，以下是具体实现代码

```
WP * new_wp()
{     WP *f,*p;
       f=free_;
       free_=free_->next;
       f->next=NULL;
       p=head;
       if(p==NULL)
       {head=f;p=head;}
       else
       {while(p->next!=NULL)p=p->next;
        p->next=f;
       }
       return f;
}
void free_wp(WP *wp)
{ WP *f,*p;
  p=free_;
  if(p==NULL)
  {free_=wp;
   p=free_;
   }
  else
  {while(p->next!=NULL)p=p->next;
   p->next=wp;
   }
  f=head;
  if(head==NULL)assert(0);
```

```
   }
  f=head;
  if(head==NULL)assert(0);
  if(head->NO==wp->NO)
  {head=head->next;}
  else
  {while(f->next!=NULL&&f->next->NO!=wp->NO)f=f->next;
   if(f->next==NULL&&f->NO==wp->NO)printf("Oh,my God!");
   else if(f->next->NO==wp->NO)f->next=f->next->next;
   else assert(0);
   }
  wp->next=NULL;
  wp->val=0;
  wp->b=0;
  wp->expr[0]='\0';
}
bool check_wp()
{WP *f;
 f=head;
 bool key=true;
 bool suc;
 while(f!=NULL)
 {uint32_t tmp_expr=expr(f->expr,&suc);
  if(!suc)assert(1);
  if(tmp_expr!=f->val)
  {key=false;
   if(f->b)
   {printf("Hit BreakPoint %d at 0x%08x\n",f->b,cpu.eip);
```

```
    f=f->next;
    continue;
   }
    printf("WatchPoint %d: %s\n",f->NO,f->expr);
    printf("Old Value=%d\n",f->val);
    printf("New Value=%d\n",tmp_expr);
    f->val=tmp_expr;
   }
    f=f->next;
  }
 return key;
 }
void delete_wp(int num)
{WP *f;
 f=&wp_pool[num];
 free_wp(f);
}
void info_wp()
{WP *f;
 f=head;
 while(f!=NULL)
 {printf("WatchPoint %d: %s=%d\n",f->NO,f->expr,f->val);
  f=f->next;
 }
 }
```

New_wp()函数使用链表结构，返回一个空监视点结构，**free_wp()**函数找到一个 **wp**,并释放它，该函数对 **free_wp** 为空，**head** 为空等情况做了处理，对应于链表中的某些操作，**check_wp()**函数中，如果新值与旧值不同，就相应的设置监视点、断点等，用到了遍历链表操作，**delete_wp()**函数释放相应的监视点。

下面执行 **make** 以及 **make run** 命令，结果如图

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo make
+ CC src/monitor/debug/watchpoint.c
+ LD build/nemu
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo 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: 12:38:57, Mar 14 2018
For help, type "help"
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x00100026

(nemu) q
```

说明上述代码没有问题

下面进入 **nemu/src/monitor/debug/ui.c** 文件，编写函数 **cmd_w()**、**cmd_b()**、同时对 **cmd_info()**函数进行补充，由于 **cmd_b()**函数为对断点的操作，因此在此一并实现，不再设置断点一节，具体代码如图

```
static int cmd_w(char *args)
{        WP *f;
         bool suc;
         f=new_wp();
         printf("WatchPoint %d: %s\n",f->NO,args);
         f->val=expr(args,&suc);
         strcpy(f->expr,args);
         if(!suc)Asseert(1,"something wrong\n");
         printf("Value :%d\n",f->val);
         return 0;
}
```

cmd_w()函数：找到空闲监视点，打印监视点编号、表达式、表达式的值

```
static int cmd_b(char *args)
{         bool suc;
          vaddr_t addr;
          addr=expr(args+1,&suc);
          if(!suc)assert(0);
          sprintf(args,"$eip==0x%x",addr);
          printf("BreakPoint %d at 0x%x\n",breakpoint_cnt,addr);
          WP *f;
          f=new_wp();
          f->val=expr(args,&suc);
          f->b=breakpoint_cnt;
          breakpoint_cnt++;
          strcpy(f->expr,args);
          return 0;
}
```

cmd_b()函数：设置断点，以“b $eip==*addr”的形式读入命令，输出断点信息，调用 free_wp()函数，可以设置多个断点

```
static int cmd_d(char *args)
{          int num;
           sscanf(args,"%d",&num);
           delete_wp(num);
           return 0;
}
```

cmd_d()函数：删除监视点，找到某个想要删除的监视点，调用 delete_wp()函数进行删除。

然后在 cmd_table[]结构体数组中添加用于实现查看监视点、删除监视点的命令 w、d，以及实现断点的命令 b,代码如图

```
} 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 */
  {"si","Single step",cmd_si},
  {"info","dump informations with option:r(register)",cmd_info},
  {"x","dump memory: x length addr",cmd_x},
  {"p","Expression evaluation",cmd_p},
  {"w","If the value of the expression has chaged,stop the execution.",cmd_w},
  {"d","Delete the nth watchpoint",cmd_d},
  {"b","BreakPoint + *ADDR",cmd_b},
};
```

下面执行 make 以及 make run 命令，结果如图

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo make
+ CC src/monitor/debug/ui.c
+ LD build/nemu
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo 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: 12:38:57, Mar 14 2018
For help, type "help"
(nemu)
```

说明上述代码没有问题

单步执行

```
(nemu) si
  100000:   b8 34 12 00 00                      movl $0x1234,%eax
```

打印寄存器

```
(nemu) info r
eax     0x00001234      4660
ecx     0x08178427      135758887
edx     0x2389abfe      596225022
ebx     0x3190c3da      831570906
esp     0x7aea8152      2062188882
ebp     0x011f9e5b      18849371
esi     0x401e50be      1075728574
edi     0x3d0324a2      1023616162
eip     0x00100005      1048581
```

表达式求值

```
(nemu) p 0xc0100000+($eax+5)*4-*($ebp+8)
[src/monitor/debug/expr.c,101,make_token] match rules[4] = "\b0[xX][0-9a-fA-F]+\
b" at position 0 with len 10: 0xc0100000
[src/monitor/debug/expr.c,101,make_token] match rules[3] = "\b[0-9]+\b" at posit
ion 20 with len 1: 4
[src/monitor/debug/expr.c,101,make_token] match rules[9] = "\*" at position 22 w
ith len 1: *
[src/monitor/debug/expr.c,101,make_token] match rules[5] = "\$[a-zA-Z]+" at posi
tion 24 with len 4: $ebp
0x2ff11340:     804328256
```

设置监视点

```
(nemu) w 0xc0100000+($eax+5)*4-*($ebp+8)
WatchPoint 0: 0xc0100000+($eax+5)*4-*($ebp+8)
[src/monitor/debug/expr.c,101,make_token] match rules[4] = "\b0[xX][0-9a-fA-F]+\
b" at position 0 with len 10: 0xc0100000
[src/monitor/debug/expr.c,101,make_token] match rules[3] = "\b[0-9]+\b" at posit
ion 20 with len 1: 4
[src/monitor/debug/expr.c,101,make_token] match rules[9] = "\*" at position 22 w
ith len 1: *
[src/monitor/debug/expr.c,101,make_token] match rules[5] = "\$[a-zA-Z]+" at posi
tion 24 with len 4: $ebp
Value :804328256
(nemu) info w
WatchPoint 0: 0xc0100000+($eax+5)*4-*($ebp+8)=804328256
```

设置断点

```
(nemu) b $eip==0x00100005
[src/monitor/debug/expr.c,101,make_token] match rules[6] = "\b[a-zA-Z0-9]+" at p
osition 0 with len 3: eip
[src/monitor/debug/expr.c,101,make_token] match rules[6] = "\b[a-zA-Z0-9]+" at p
osition 6 with len 9: x00100005
[src/monitor/debug/expr.c,101,make_token] match rules[6] = "\b[a-zA-Z0-9]+" at p
osition 24 with len 3: eip
[src/monitor/debug/expr.c,101,make_token] match rules[6] = "\b[a-zA-Z0-9]+" at p
osition 30 with len 9: x00100005
BreakPoint 1 at 0xfffe1dc0
[src/monitor/debug/expr.c,101,make_token] match rules[5] = "\$[a-zA-Z]+" at posi
tion 0 with len 4: $eip
[src/monitor/debug/expr.c,101,make_token] match rules[6] = "\b[a-zA-Z0-9]+" at p
osition 8 with len 8: fffe1dc0
[src/monitor/debug/expr.c,101,make_token] match rules[5] = "\$[a-zA-Z]+" at posi
tion 24 with len 4: $eip
[src/monitor/debug/expr.c,101,make_token] match rules[3] = "\b[0-9]+\b" at posit
ion 32 with len 8: 00100005
```

删除监视点

```
(nemu) d 0
```

思考题：一点也不能长？我们知道 int3 指令不带任何操作数,操作码为 1 个字节,因此指令的长度是 1 个字节.这是必须的吗? 假设有一种 x86 体系结构的变种 my-x86，除了 int3 指令的长度变成了 2 个字节之外,其余指令和 x86 相同.在 my-x86 中,文章中的断点机制还可以正常工作吗？为什么?

答：指令长度为一个字节是必须的，数据访问断点的指令可以为 1、2、4 字节；不可以正常工作，可能会产生非法行为

"随心所欲"的断点 如果把断点设置在指令的非首字节(中间或末尾)，会发生什么？你可以在 GDB 中尝试一下，然后思考并解释其中的缘由.。

答：断点必须设置在首字节，不然会发生意想不到的错误。

git 记录

```
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo git add .
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo git commit --allow-empty
[master（根提交） 4e0f0e6] fix bug for pa1.2
 53 files changed, 4064 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Makefile
 create mode 100644 Makefile.git
 create mode 100644 README.md
 create mode 100644 include/common.h
 create mode 100644 include/cpu/decode.h
 create mode 100644 include/cpu/exec.h
 create mode 100644 include/cpu/reg.h
 create mode 100644 include/cpu/rtl.h
 create mode 100644 include/debug.h
 create mode 100644 include/device/mmio.h
 create mode 100644 include/device/port-io.h
 create mode 100644 include/macro.h
 create mode 100644 include/memory/memory.h
 create mode 100644 include/memory/mmu.h
```

```
 create mode 100644 include/monitor/expr.h
 create mode 100644 include/monitor/monitor.h
 create mode 100644 include/monitor/watchpoint.h
 create mode 100644 include/nemu.h
 create mode 100644 runall.sh
 create mode 100644 src/cpu/decode/decode.c
 create mode 100644 src/cpu/decode/modrm.c
 create mode 100644 src/cpu/exec/all-instr.h
 create mode 100644 src/cpu/exec/arith.c
 create mode 100644 src/cpu/exec/cc.c
 create mode 100644 src/cpu/exec/control.c
 create mode 100644 src/cpu/exec/data-mov.c
 create mode 100644 src/cpu/exec/exec.c
 create mode 100644 src/cpu/exec/logic.c
 create mode 100644 src/cpu/exec/prefix.c
 create mode 100644 src/cpu/exec/special.c
 create mode 100644 src/cpu/exec/system.c
 create mode 100644 src/cpu/intr.c
 create mode 100644 src/cpu/reg.c
 create mode 100644 src/device/device.c
 create mode 100644 src/device/io/mmio.c
 create mode 100644 src/device/io/port-io.c
 create mode 100644 src/device/keyboard.c
 create mode 100644 src/device/serial.c
 create mode 100644 src/device/timer.c
 create mode 100644 src/device/vga.c
 create mode 100644 src/main.c
 create mode 100644 src/memory/memory.c
 create mode 100644 src/misc/logo.c
 create mode 100644 src/monitor/cpu-exec.c
 create mode 100644 src/monitor/debug/expr.c
 create mode 100644 src/monitor/debug/ui.c
 create mode 100644 src/monitor/debug/watchpoint.c
 create mode 100644 src/monitor/diff-test/diff-test.c
 create mode 100644 src/monitor/diff-test/gdb-host.c
 create mode 100644 src/monitor/diff-test/protocol.c
 create mode 100644 src/monitor/diff-test/protocol.h
 create mode 100644 src/monitor/monitor.c
zhaoweikang@zhaoweikang:~/ics2017/nemu$ sudo git log
commit 4e0f0e6c4813e2b7590a2cb90b5b064a74b0fd9b
Author: 161630220-Zhao Weikang <2875206963@qq.com>
Date:   Tue Mar 27 17:18:18 2018 +0800

    fix bug for pa1.2
```

必答题

理解基础设施 我们通过一些简单的计算来体会简易调试器的作用.首先作以下假设:假设你需要编译 500 次 NEM 才能完成 PA. 假设这 500 次编译当中,有 90%的次数是用于调试. 假设你没有实现简易调试器,只能通过 GDB 对运行在 NEMU 上的客户程序进行调 试.在每一次调试中,由于 GDB 不能直接观测客户程序,你需要花费 30 秒的时间来从 GDB 中获取并分析一个信息.假设你需要获取并分析 20 个信息才能排除一个 bug. 那么这个学期下来, 你将会在调试上花费多少时间?

500*0.9*30*20=270000（秒）

由于简易调试器可以直接观测客户程序, 假设通过简易调试器只需要花费 10 秒的时间 从中获取并分析相同的信息. 那么这个学期下来, 简易调试器可以帮助你节省多少调试的时间?

500*0.9*10*20=90000（秒）

节省时间 270000-90000=180000（秒）

查阅 i386 手册 理解了科学查阅手册的方法之后,请你尝试 i386 手册中查阅以下问题所在的位置,把需要阅读的范围写到你的实验报告里面: EFLAGS 寄存器中的 CF 位是什么意思? ModR/M 字节是什么? mov 指令的具体格式是怎么样的?

要回答第一个问题，需要阅读 chapter 2 的 2.3.4 Flags Register 的内容以及 2.3.4.1 Status Flags 的内容

要回答第二个问题，需要阅读 chapter 3 的 3.1.1 General-Purpose Data Movement Instructions 节的内容

shell 命令 完成 PA1 的内容之后,nemu/目录下的所有.c 和.h 和文件总共有多少行代码? 你是使用什么命令得到这个结果的? 和框架代码相比,你在 PA1 中编写了多少行代码? (Hint: 目前 2017 分支中记录的正好是做 PA1 之前的状态,思考一下应该如何回到"过去"?) 你可以把这条命令写入 Makefile 中,随着实验进度的推进, 你可以很方便地统计 工程的代码行数, 例如敲入 make count 就会自动运行统计代码行数的命令.再来个难一 点的,除去空行之外,nemu/目录下的所有.c 和.h 文件总共有多少行代码?

包括空行的.c 文件行数

```
root@zhaoweikang:/home/zhaoweikang/ics2017/nemu# find . -name *.c |xargs wc -l
   29 ./src/device/serial.c
   38 ./src/device/vga.c
   98 ./src/device/device.c
   70 ./src/device/keyboard.c
   28 ./src/device/timer.c
   70 ./src/device/io/mmio.c
   56 ./src/device/io/port-io.c
   28 ./src/memory/memory.c
   44 ./src/monitor/cpu-exec.c
  143 ./src/monitor/monitor.c
  357 ./src/monitor/debug/expr.c
   99 ./src/monitor/debug/watchpoint.c
  222 ./src/monitor/debug/ui.c
  157 ./src/monitor/diff-test/diff-test.c
  106 ./src/monitor/diff-test/gdb-host.c
  314 ./src/monitor/diff-test/protocol.c
  113 ./src/cpu/decode/modrm.c
  311 ./src/cpu/decode/decode.c
   43 ./src/cpu/reg.c
   32 ./src/cpu/exec/cc.c
  223 ./src/cpu/exec/arith.c
   43 ./src/cpu/exec/control.c
   32 ./src/cpu/exec/cc.c
  223 ./src/cpu/exec/arith.c
   43 ./src/cpu/exec/control.c
    9 ./src/cpu/exec/prefix.c
   60 ./src/cpu/exec/logic.c
   65 ./src/cpu/exec/system.c
   77 ./src/cpu/exec/data-mov.c
  255 ./src/cpu/exec/exec.c
   46 ./src/cpu/exec/special.c
   13 ./src/cpu/intr.c
   12 ./src/main.c
   31 ./src/misc/logo.c
 3192 总用量
```

包括空行的.h 文件行数

```
root@zhaoweikang:/home/zhaoweikang/ics2017/nemu# find . -name *.h |xargs wc -l
    8 ./include/nemu.h
   13 ./include/device/port-io.h
   14 ./include/device/mmio.h
   13 ./include/macro.h
   18 ./include/memory/memory.h
   82 ./include/memory/mmu.h
    8 ./include/monitor/expr.h
   21 ./include/monitor/watchpoint.h
    7 ./include/monitor/monitor.h
   63 ./include/cpu/reg.h
   53 ./include/cpu/exec.h
  115 ./include/cpu/decode.h
  189 ./include/cpu/rtl.h
   45 ./include/debug.h
   29 ./include/common.h
   48 ./src/monitor/diff-test/protocol.h
    8 ./src/cpu/exec/all-instr.h
  734 总用量
```

去掉空行的.c 文件行数

```
root@zhaoweikang:/home/zhaoweikang/ics2017/nemu# find . -name "*.c" |xargs cat|g
rep -v ^$|wc -l
2578
```

去掉空行的.h 文件行数

```
root@zhaoweikang:/home/zhaoweikang/ics2017/nemu# find . -name "*.h" |xargs cat|g
rep -v ^$|wc -l
581
```

使用 man 打开工程目录下的 Makefile 文件,你会在 CFLAGS 变量中看到 gcc 的一些编译 选项. 请解释 gcc 中的-Wall 和-Werror 有什么作用?为什么要使用-Wall 和-Werror?

-Wall 打开 gcc 的所有警告, -Werror，它要求 gcc 将所有的警告当成错误进行处理。