



南开大学  
Nankai University

南 开 大 学

计 算 机 学 院  
计算机系统设计作业报告

---

PA 实验一报告

---

朱浩泽 1911530

年级：2019 级

专业：计算机科学与技术

指导教师：卢冶

2022 年 3 月 16 日

## 目录

一、 概述	1
(一) 实验目的 . . . . .	1
(二) 实验内容 . . . . .	1
二、 阶段一	1
(一) 实现正确的寄存器结构体 . . . . .	1
(二) 实现解析命令 . . . . .	2
(三) 实现单步执行 . . . . .	2
(四) 实现打印寄存器 . . . . .	3
(五) 实现内存扫描 . . . . .	4
三、 阶段二	5
(一) 词法分析 . . . . .	5
(二) 递归求值 . . . . .	7
(三) 实现调试中的表达式求值 . . . . .	10
四、 阶段三	11
五、 必答题	15
(一) 查阅 i386 手册回答问题 . . . . .	15
1. EFLAGS 寄存器中的 CF 位是什么意思? . . . . .	15
2. ModR/M 字节是什么? . . . . .	15
3. mov 指令的具体格式是怎么样的? . . . . .	15

## 一、 概述

### (一) 实验目的

熟悉基础设施的各种工具和手段；熟悉寄存器间的存储关系；学习实现简易调试器补充指令；学习表达式求值，监视点的实现方法

### (二) 实验内容

- 阶段一
  - 实现正确的寄存器结构体
  - 实现解析命令
  - 实现单步执行
  - 实现打印寄存器
  - 实现内存扫描
- 阶段二
  - 实现词法分析
  - 实现递归求值
  - 实现调试中的表达式求值
- 阶段三
  - 实现断点
  - 实现监视点

## 二、 阶段一

### (一) 实现正确的寄存器结构体

- 为了实现 32 位、16 位、8 位寄存器各 8 个和一个程序计数器 eip，我们利用匿名 Union 各个变量互斥并共享同一内存首地址这一特点，来实现这一结构体
- 修改 nemu/include/cpu/reg.h 中的代码如下

```
1 typedef struct {
2     union{
3         /* data */
4         union {
5             uint32_t _32;
6             uint16_t _16;
7             uint8_t _8[2];
8         } gpr[8];
9         struct
10        {
11            rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
12        };
13    };
14    vaddr_t eip;
15 } CPU_state;
```

- 在上面的代码中，grp 中的的 \_32 代表的是 eax 中的 32 位，\_16 代表的是 ax 中的 16 位，\_8[2] 分别代表的是 ah 中的 8 到 16 位和 al 中的 0 到 8 位

## (二) 实现解析命令

- 在 nemu/src/monitor/debug/ui.c 中实现命令表，其代码如下

```

1 static struct {
2     char *name;
3     char *description;
4     int (*handler) (char *);
5 } cmd_table [] = {
6     { "help", "Display informations about all supported commands", cmd_help },
7     { "c", "Continue the execution of the program", cmd_c },
8     { "q", "Exit NEMU", cmd_q },
9
10    /* TODO: Add more commands */
11
12    { "si", "args:[N]; exectue [N] instructions step by step", cmd_si}, //让程序单步执
13    { "info", "args:r/w;print information about register or watch point ", cmd_info //行 N 条指令后暂停执行，当N没有给出时，缺省为1
14    }, //打印寄存器状态
15    { "x", "x [N] [EXPR];sacn the memory", cmd_x }, //内存扫描
16    { "p", "expr", cmd_p}, //表达式
17    { "w", "set the watchpoint", cmd_w}, //添加监视点
18    { "d", "delete the watchpoint", cmd_d} //删除监视点
19 };

```

- 其执行效果如下图所示

```

(nemu) help
help - Display informations about all supported commands
c - Continue the execution of the program
q - Exit NEMU
si - args:[N]; exectue [N] instructions step by step
info - args:r/w;print information about register or watch point
x - x [N] [EXPR];sacn the memory
p - expr
w - set the watchpoint
d - delete the watchpoint

```

## (三) 实现单步执行

- 在 nemu/src/monitor/debug/ui.c 添加 cmd\_si 函数，其代码如下

```

1 static int cmd_si(char *args) {
2     uint64_t N = 0;
3     if(args == NULL) {
4         N = 1;
5     }
6     else {
7         int temp = sscanf(args, "%lu", &N);
8         if(temp <= 0) {
9             printf("args error in cmd_si\n");
10            return 0;
11        }
12    }
13 }

```

```

13  cpu_exec(N);
14  return 0;
15  }

```

- 执行效果如下图所示，输入 si 和要执行的步骤 N，进行打印

```

lighthouse@VM-24-4-ubuntu:~/main/PA/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 image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 10:56:22, Mar 16 2022
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) █

```

#### (四) 实现打印寄存器

- 在 nemu/src/monitor/debug/ui.c 添加 cmd\_info 函数，其代码如下，输入参数 r 对寄存器直接进行打印，输入参数 w 对监视点进行打印

```

1  static int cmd_info(char *args) {
2      char s;
3      if(args == NULL) {
4          printf("args error in cmd_info (miss args)\n");
5          return 0;
6      }
7      int temp = sscanf(args, "%c", &s);
8      if(temp <= 0) {
9          //解析失败
10         printf("args error in cmd_info\n");
11         return 0;
12     }
13     if(s == 'w') {
14         //打印监视点信息
15         print_wp();
16         return 0;
17     }
18     if(s == 'r') {
19         //打印寄存器
20         //32bit
21         for(int i = 0; i < 8; i++) {
22             printf("%s 0x%x\n", regsl[i], reg_l(i));
23         }
24         printf("eip 0x%x\n", cpu.eip);
25         //16bit
26         for(int i = 0; i < 8; i++) {
27             printf("%s 0x%x\n", regsw[i], reg_w(i));
28         }
29         //8bit
30         for(int i = 0; i < 8; i++)
31         {
32             printf("%s 0x%x\n", regsb[i], reg_b(i));
33         }
34         return 0;

```

```

35 }
36 //如果产生错误
37 printf("args error in cmd_info\n");
38 return 0;
39 }

```

- 执行效果如下图所示

```

lighthouse@VM-24-4-ubuntu:~/main/PA/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 image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 10:56:22, Mar 16 2022
For help, type "help"
(nemu) info r
eax 0x1642ffb1
ecx 0x5af09203
edx 0x48c6d29
ebx 0x24b2f146
esp 0x11c84186
ebp 0x5ce6fb09
esi 0x3416ceb5
edi 0x578efbd0
eip 0x100000
ax 0xffb1
cx 0x9203
dx 0x6d29
bx 0xf146
sp 0x4186
bp 0xfb09
si 0xceb5
di 0xfbd0
al 0xb1
cl 0x3
dl 0x29
bl 0x46
ah 0xff
ch 0x92
dh 0x6d
bh 0xf1
(nemu) 

```

## (五) 实现内存扫描

- 在 nemu/src/monitor/debug/ui.c 添加 cmd\_x 函数，其代码如下

```

1 static int cmd_x(char *args) {
2     int nLen = 0;
3     vaddr_t addr;
4     int temp = sscanf(args, "%d 0x%x", &nLen, &addr);
5     if(temp <= 0) {
6         //解析失败
7         printf("args error in cmd_si\n");
8         return 0;
9     }
10    printf("Memory:");
11    for(int i = 0; i < nLen; i++) {
12        if(i % 4 == 0) {
13            printf("\n0x%x: 0x%02x", addr + i, vaddr_read(addr + i, 1));
14        }
15        else {
16            printf(" 0x%02x", vaddr_read(addr + i, 1));
17        }
18    }
19    printf("\n");
20    return 0;
21 }

```

- 该函数主要是调用了 nemu/src/memory/memory.c 中的 vaddr\_read 函数进行内存扫描
- 执行效果如下图所示

```
lighthouse@VM-24-4-ubuntu:~/main/PA/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 image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 10:56:22, Mar 16 2022
For help, type "help"
(nemu) x 39 0x100000
Memory:
0x100000: 0xb8 0x34 0x12 0x00
0x100004: 0x00 0xb9 0x27 0x00
0x100008: 0x10 0x00 0x89 0x01
0x10000c: 0x66 0xc7 0x41 0x04
0x100010: 0x01 0x00 0xbb 0x02
0x100014: 0x00 0x00 0x00 0x66
0x100018: 0xc7 0x84 0x99 0x00
0x10001c: 0xe0 0xff 0xff 0x01
0x100020: 0x00 0xb8 0x00 0x00
0x100024: 0x00 0x00 0xd6
(nemu)
```

### 三、 阶段二

在本次实验中，要求实现包括  $+$   $-$   $*$   $/$   $==$   $!=$   $>=$   $<=$   $>$   $<$   $\&\&$   $||$   $!$  指针和正负运算等多种运算，以及十进制数、八进制数、十六进制数等运算。

在有了编译原理课程的基础后，我们可以知道，表达式求值的过程是先将整个句子切分为各个 token，然后对各个 token 按照赋予的优先级进行分割后求返回值，最终通过这种递归运算求出整个表达式的值。

在这个求值过程中，我们需要注意寄存器的值是可以当作操作数进行运算的，且对于括号的操作跟编译原理中直接利用 lex 工具，本次实验中需要手写函数进行动态判定，这本质上是一个脱括号的过程，需要判断脱括号后其匹配是否合法。

本节实验主要代码在 nemu/src/monitor/debug/exp.c 文件中实现

#### (一) 词法分析

- 首先我们定义 tokens 类型和正则表达式来识别 tokens，其代码如下

```
1 enum {
2     TK_NOTYPE = 256,
3     TK_NUMBER,
4     TK_HEX,
5     TK_REG,
6     TK_EQ,
7     TK_NEQ,
8     TK_AND,
9     TK_OR,
10    TK_NEGATIVE,
11    TK_DEREF,
12 };
13
14 static struct rule {
15     char *regex;
16     int token_type;
17 } rules[] = {
18
19     /* TODO: Add more rules.
20      * Pay attention to the precedence level of different rules.
```

```

21  */
22
23  {"+", TK_NOTYPE}, // spaces
24  {"0x[0-9A-Fa-f][0-9A-Fa-f]*", TK_HEX},
25  {"0|[1-9][0-9]*", TK_NUMBER}, // 数字
26  {"\\$(eax|ecx|edx|ebx|esp|ebp|esi|edi|eip|ax|cx|dx|bx|sp|bp|si|di|al|cl|dl|bl|ah
    |ch|dh|bh)", TK_REG},
27  {"==", TK_EQ},
28  {"!=", TK_NEQ},
29  {"&&", TK_AND},
30  {"\\|\\|", TK_OR},
31  {"!", '!'},
32  {"\\+", '+'},
33  {"-", '-'},
34  {"\\*", '*'},
35  {"\\/", '/'},
36  {"\\(", '('},
37  {"\\)", ')'},
38  };

```

- 修改代码框架中的 make\_token 函数，对各种类型的输入进行处理，其中八进制数和十六进制数将会被提前处理，而寄存器则将会跳过开头的美元符号，其代码如下

```

1  static bool make_token(char *e) {
2  int position = 0;
3  int i;
4  regmatch_t pmatch;
5
6  nr_token = 0;
7
8  while (e[position] != '\0') {
9      /* Try all rules one by one. */
10     for (i = 0; i < NR_REGEX; i++) {
11         if (regexexec(&re[i], e + position, 1, &pmatch, 0) == 0 && pmatch.rm_so == 0) {
12             char *substr_start = e + position;
13             int substr_len = pmatch.rm_eo;
14
15             Log("match rules[%d] = \"%s\" at position %d with len %d: %.s",
16                 i, rules[i].regex, position, substr_len, substr_start);
17             position += substr_len;
18
19             /* TODO: Now a new token is recognized with rules[i]. Add codes
20              * to record the token in the array `tokens'. For certain types
21              * of tokens, some extra actions should be performed.
22              */
23             if (substr_len > 32) {
24                 assert(0);
25             }
26             if (rules[i].token_type == TK_NOTYPE) {
27                 break;
28             }
29             else {
30                 tokens[nr_token].type = rules[i].token_type;
31                 switch (rules[i].token_type) {
32                     case TK_NUMBER:

```



```

33     strncpy(tokens[nr_token].str, substr_start, substr_len);
34     *(tokens[nr_token].str + substr_len) = '\0';
35     break;
36 case TK_HEX:
37     strncpy(tokens[nr_token].str, substr_start + 2, substr_len - 2); //跳过开头的0x
38     *(tokens[nr_token].str + substr_len - 2) = '\0';
39     break;
40 case TK_REG:
41     strncpy(tokens[nr_token].str, substr_start + 1, substr_len - 1); //跳过开头的$
42     *(tokens[nr_token].str + substr_len - 1) = '\0';
43 }
44 printf("Success record : nr_token = %d, dtype = %d, str = %s\n", nr_token,
45        tokens[nr_token].type, tokens[nr_token].str);
46 nr_token += 1;
47 break;
48 }
49 if (i == NR_REGEX) {
50     return false;
51 }
52 }
53
54 return true;
55 }

```

## (二) 递归求值

- 首先我们要进行的工作是检查括号匹配，其主要想法是利用一个计数器来计算匹配的数目，如果最终计数器的值不为 0 则说明括号不匹配，则表达式出错。其代码如下

```

1 //判断括号的匹配
2 bool check_parentheses(int p, int q) {
3     if(p >= q) {
4         //右括号少于左括号
5         printf("error:p>=q in check_parntheses\n");
6         return false;
7     }
8     if(tokens[p].type != '(' || tokens[q].type != ')'){
9         //括号不匹配
10        return false;
11    }
12    int cnt = 0; //记录当前未匹配的左括号的数目
13    for(int curr = p + 1; curr < q; curr++) {
14        if(tokens[curr].type == '(') {
15            cnt++;
16        }
17        if(tokens[curr].type == ')') {
18            if(cnt != 0) {
19                cnt--;
20            }
21        }
22        else {
23            //左右括号不匹配
24            return false;
25        }
26    }
27    return cnt == 0;
28 }

```

```

24     }
25 }
26 }
27 if(cnt == 0) {
28     return true;
29 }
30 else {
31     return false;
32 }
33 }

```

- 利用一个函数来计算表达式优先级最低的运算符的位置，如果有同为最低优先级的运算符，返回最后被结合的运算符的索引位置，其代码如下

```

1 int findDominantOp(int p, int q) {
2     int level=0;
3     int pos[5]={-1, -1, -1, -1, -1};
4     for(int i = p; i < q; i++){
5         if(level == 0) {
6             if(tokens[i].type == TK_AND || tokens[i].type == TK_OR) {
7                 pos[0] = i;
8             }
9             if(tokens[i].type == TK_EQ || tokens[i].type == TK_NEQ) {
10                pos[1] = i;
11            }
12            if(tokens[i].type == '+' || tokens[i].type == '-') {
13                pos[2] = i;
14            }
15            if(tokens[i].type == '*' || tokens[i].type == '/') {
16                pos[3] = i;
17            }
18            if(tokens[i].type == TK_NEGATIVE || tokens[i].type == TK_DEREF || tokens[i].
                type == '!') {
19                pos[4] = i;
20            }
21        }
22        if(tokens[i].type=='(') {
23            level++;
24        }
25        if(tokens[i].type==')') {
26            level--;
27        }
28    }
29    for(int i = 0; i < 5; i++) {
30        if(pos[i] != -1) {
31            return pos[i];
32        }
33    }
34    printf("error in findDominantOp\n");
35    printf("[p=%d,q=%d]\n",p,q);
36    assert(0);
37 }

```

- 根据实验指导书编写 eval 函数，用于值的计算，其中的参数 p 和 q 分别代表这个子表达式的开始位置和结束为止，其代码如下

```

1 uint32_t eval(int p, int q) {
2     if(p > q) {
3         printf("error:p>q in eval, p = %d, q = %d\n", p, q);
4         assert(0);
5     }
6     if(p == q) {
7         int num;
8         switch (tokens[p].type){
9             case TK_NUMBER:
10                sscanf(tokens[p].str, "%d", &num);
11                return num;
12             case TK_HEX:
13                sscanf(tokens[p].str, "%x", &num);
14                return num;
15             case TK_REG:
16                for(int i = 0; i < 8; i++) {
17                    if(strcmp(tokens[p].str, regsl[i]) == 0) {
18                        return reg_l(i);
19                    }
20                    if(strcmp(tokens[p].str, regsw[i]) == 0) {
21                        return reg_w(i);
22                    }
23                    if(strcmp(tokens[p].str, regsb[i]) == 0) {
24                        return reg_b(i);
25                    }
26                }
27                if(strcmp(tokens[p].str, "eip") == 0) {
28                    return cpu.eip;
29                }
30                else {
31                    printf("error in TK_REG in eval()\n");
32                    assert(0);
33                }
34            }
35        }
36        if(check_parentheses(p, q) == true) {
37            return eval(p + 1, q - 1);
38        }
39        else {
40            int op = findDominantOp(p, q);
41            vaddr_t addr;
42            int result;
43            switch (tokens[op].type) {
44                case TK_NEGATIVE: //负号
45                    return -eval(p + 1, q);
46                case TK_DEREF: //指针求值
47                    addr = eval(p + 1, q);
48                    result = vaddr_read(addr, 4);
49                    printf("addr=%u(0x%x)---->value=%d(0x%08x)\n", addr, addr, result, result);
50                    ;
51                    return result;
52                case '!':
53                    result = eval(p + 1, q);
54                    if(result != 0) {
55                        return 0;
56                    }
57            }
58        }
59    }
60 }

```

```

55     }
56     else {
57         return 1;
58     }
59 }
60 uint32_t val1 = eval(p, op - 1);
61 uint32_t val2 = eval(op + 1, q);
62 switch(tokens[op].type) {
63     case '+':
64         return val1 + val2;
65     case '-':
66         return val1 - val2;
67     case '/':
68         return val1 / val2;
69     case '*':
70         return val1 * val2;
71     case TK_EQ:
72         return val1 == val2;
73     case TK_NEQ:
74         return val1 != val2;
75     case TK_AND:
76         return val1 && val2;
77     case TK_OR:
78         return val1 || val2;
79     default:
80         assert(0);
81 }
82 }
83 return 1;
84 }

```

### (三) 实现调试中的表达式求值

- 修改框架代码中的 `expr` 函数进行表达式求值的工作,其中主要分为两步:先调用 `make_token` 进行词法分析,再调用 `eval` 函数进行求值操作。为了实现这一功能,我们要对 `token` 进行选择保存信息:对于寄存器和数字,我们保存其类型,并利用字符串保存寄存器的名字和数值;对于一般运算符保存 `token` 类型;对于加减法需要区分是单目运算符还是双目运算符,其代码如下

```

1  uint32_t expr(char *e, bool *success) {
2      if (!make_token(e)) {
3          *success = false;
4          return 0;
5      }
6      /* TODO: Insert codes to evaluate the expression. */
7      // TODO();
8
9      // return 0;
10     if(tokens[0].type == '-') {
11         tokens[0].type = TK_NEGATIVE;
12     }
13     if(tokens[0].type == '*') {
14         tokens[0].type = TK_DEREF;
15     }

```

```

16 for(int i = 1; i < nr_token; i++) {
17     if(tokens[i].type == '-') {
18         if(tokens[i - 1].type != TK_NUMBER && tokens[i - 1].type != ')') {
19             tokens[i].type = TK_NEGATIVE;
20         }
21     }
22     if(tokens[i].type == '*') {
23         if(tokens[i - 1].type != TK_NUMBER && tokens[i - 1].type != ')') {
24             tokens[i].type = TK_DEREF;
25         }
26     }
27 }
28 *success = true;
29 return eval(0, nr_token - 1);
30 }

```

- 实验效果如下图所示

```

lighthouse@VN-24-4-ubuntu:~/main/PA/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 image.
Welcome to NEMU!
[src/monitor/monitor.c:30,welcome] Build time: 10:56:22, Mar 16 2022
For help, type "help"
(nemu) p 1+1
[src/monitor/debug/expr.c:91,make_token] match rules[2] = "0|[1-9][0-9]*" at position 0 with len 1: 1
Success record : nr_token = 0, dtype = 257, str = 1
[src/monitor/debug/expr.c:91,make_token] match rules[9] = "+" at position 1 with len 1: +
Success record : nr_token = 1, dtype = 43, str =
[src/monitor/debug/expr.c:91,make_token] match rules[2] = "0|[1-9][0-9]*" at position 2 with len 1: 1
Success record : nr_token = 2, dtype = 257, str = 1
the value of expr is:2
(nemu) p 1+(-1)
[src/monitor/debug/expr.c:91,make_token] match rules[2] = "0|[1-9][0-9]*" at position 0 with len 1: 1
Success record : nr_token = 0, dtype = 257, str = 1
[src/monitor/debug/expr.c:91,make_token] match rules[9] = "+" at position 1 with len 1: +
Success record : nr_token = 1, dtype = 43, str =
[src/monitor/debug/expr.c:91,make_token] match rules[13] = "(" at position 2 with len 1: (
Success record : nr_token = 2, dtype = 40, str = 1
[src/monitor/debug/expr.c:91,make_token] match rules[10] = "-" at position 3 with len 1: -
Success record : nr_token = 3, dtype = 45, str =
[src/monitor/debug/expr.c:91,make_token] match rules[2] = "0|[1-9][0-9]*" at position 4 with len 1: 1
Success record : nr_token = 4, dtype = 257, str = 1
[src/monitor/debug/expr.c:91,make_token] match rules[14] = ")" at position 5 with len 1: )
Success record : nr_token = 5, dtype = 41, str =
the value of expr is:0
(nemu) p $eax*30/2+1
[src/monitor/debug/expr.c:91,make_token] match rules[3] = "\$(eax|ecx|edx|ebx|esp|ebp|esi|edi|eip|ax|cx|dx|bx|sp|bp|si|di|al|cl|dl|bl|ah|ch|dh|bh)" at position
0 with len 4: $eax
Success record : nr_token = 0, dtype = 259, str = eax
[src/monitor/debug/expr.c:91,make_token] match rules[11] = "*" at position 4 with len 1: *
Success record : nr_token = 1, dtype = 42, str =
[src/monitor/debug/expr.c:91,make_token] match rules[2] = "0|[1-9][0-9]*" at position 5 with len 2: 30
Success record : nr_token = 2, dtype = 257, str = 30
[src/monitor/debug/expr.c:91,make_token] match rules[12] = "/" at position 7 with len 1: /
Success record : nr_token = 3, dtype = 47, str =
[src/monitor/debug/expr.c:91,make_token] match rules[2] = "0|[1-9][0-9]*" at position 8 with len 1: 2
Success record : nr_token = 4, dtype = 257, str = 2
[src/monitor/debug/expr.c:91,make_token] match rules[9] = "+" at position 9 with len 1: +
Success record : nr_token = 5, dtype = 43, str =
[src/monitor/debug/expr.c:91,make_token] match rules[2] = "0|[1-9][0-9]*" at position 10 with len 1: 1
Success record : nr_token = 6, dtype = 257, str = 1
addr=30(0x1e)---->value=0(0x00000000)
addr=0(0x0)---->value=0(0x00000000)
the value of expr is:1
(nemu)

```

## 四、 阶段三

### 监视点和断点

这一部分的主要工作是通过实现两个链表（free 和 head）进行维护的，其代表空闲的和占用的监视点。主要工作和代码在 nemu/include/monitor/watchpoint.h 和 nemu/src/monitor/debug/watchpoint.c 文件中实现。

首先是关于实现方式的，如果删除某个监视点，其余监视点不会递补该监视点的编号（这操作实际上减少了部分工作量），新增加的监视点则会在最大的监视点编号上继续增加。监视点结构及其初始化的代码如下：

```

1 typedef struct watchpoint {
2     int NO;

```

```

3 struct watchpoint *next;
4
5 /* TODO: Add more members if necessary */
6 int old; //旧的值
7 char e[32]; //表达式
8 int hitNum; //记录触发次数
9
10 } WP;
11
12 //结构初始化
13 void init_wp_pool() {
14     int i;
15     for (i = 0; i < NR_WP; i++) {
16         wp_pool[i].NO = i;
17         wp_pool[i].next = &wp_pool[i + 1];
18         wp_pool[i].old = 0;
19         wp_pool[i].hitNum = 0;
20     }
21     wp_pool[NR_WP - 1].next = NULL;
22
23     head = NULL;
24     free_ = wp_pool;
25     used_next = 0;
26 }

```

接下来，我们创建 `new_wp` 函数和 `free_wp` 函数来创建和删除监视点。主要思路是在添加监视点时，首先查找是否有足够的位置，如果有的话从链表中返回一个空闲的监视点结构，并更新 `head` 链表，将新添加的监视点添加到其尾部；在删除监视点的时候，先遍历 `head` 链表找到要被删除的监视点进行释放，然后更新 `free` 链表对空闲的监视点进行更新。其主要代码如下：

```

1 bool new_wp(char *args) {
2     //从free链表中返回一个空闲监视点结构
3     if(free_ == NULL) {
4         assert(0);
5     }
6     WP* result = free_;
7     free_ = free_ -> next;
8
9     result -> NO = used_next;
10    used_next++;
11    result -> next = NULL;
12    strcpy(result -> e, args);
13    result -> hitNum = 0;
14    bool is_success;
15    result -> old = expr(result -> e, &is_success); //计算旧的值
16    if(is_success == false) {
17        printf("error in new_wp; expression fault!\n");
18        return false;
19    }
20
21    wptemp = head;
22    if(wptemp == NULL) {
23        head = result;
24    }
25    else {

```

```

26     while (wptemp -> next != NULL)
27     {
28         wptemp = wptemp -> next;
29     }
30     wptemp -> next = result;
31 }
32 printf("Success: set watchpoint %d, oldvalue = %d\n", result -> NO, result -> old);
33 return true;
34 }
35
36 //删除监视点
37 bool free_wp(int num) {
38     WP *chosen = NULL;
39     if(head == NULL) {
40         printf("no watch point now\n");
41         return false;
42     }
43     if(head -> NO == num) {
44         chosen = head;
45         head = head -> next;
46     }
47     else {
48         wptemp = head;
49         while (wptemp != NULL && wptemp -> next != NULL)
50         {
51             /* code */
52             if(wptemp -> next -> NO == num) {
53                 chosen = wptemp -> next;
54                 wptemp -> next = wptemp -> next -> next;
55                 break;
56             }
57             wptemp = wptemp -> next;
58         }
59     }
60     if(chosen != NULL) {
61         chosen -> next = free_;
62         free_ = chosen;
63         return true;
64     }
65     return false;
66 }

```

同时,我们还对监视点进行实时监测,其主要调用了 nemu/include/cpu/exe.c 中的 cpu\_exec() 函数。通过在 cpu 每执行一次命令是都检查一下所有的监视点的表达式的值是否发生了变化,如果发生了则打印变化信息来实现。其主要代码如下

```

1 void print_wp() {
2     if(head == NULL) {
3         printf("no watchpoint now\n");
4         return;
5     }
6     printf("watchpoint:\n");
7     printf("NO.  expr  hitTimes\n");
8     wptemp = head;
9     while (wptemp != NULL)
10    {

```

```

11     printf("%d %s %d\n", wptemp -> NO, wptemp -> e, wptemp -> hitNum);
12     wptemp = wptemp -> next;
13 }
14 }
15
16 bool watch_wp() {
17     bool is_success;
18     int result;
19     if(head == NULL) {
20         return true;
21     }
22     wptemp = head;
23     while (wptemp != NULL)
24     {
25         /* code */
26         result = expr(wptemp -> e, &is_success);
27         if(result != wptemp -> old)
28         {
29             wptemp -> hitNum += 1;
30             printf("Hardware watchpoint %d:%s\n", wptemp -> NO, wptemp -> e);
31             printf("Old value:%d\nNew value:%d\n\n", wptemp -> old, result);
32             wptemp -> old = result;
33             return false;
34         }
35         wptemp = wptemp -> next;
36     }
37     return true;
38 }

```

实验效果如下图所示

```

Lighthouse@VM-24-4-ubuntu:~/main/PA/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 image.
Welcome to NEMU!
[src/monitor/monitor.c:30,welcome] Build time: 10:56:22, Mar 16 2022
For help, type "help"
(nemu) w $ecx
[src/monitor/debug/expr.c:91,make_token] match rules[3] = "\\$(eax|ecx|edx|ebx|esp|ebp|esi|edi|eip|ax|cx|dx|bx|sp|bp|si|di|al|cl|dl|bl|ah|ch|dh|bh)" at position 0 with len 4: $ecx
Success record : nr_token = 0, dtype = 259, str = ecx
Success: set watchpoint 0, oldvalue = 1944013018
(nemu) si 4
100000: b8 34 12 00 00 movl $0x1234,%eax
[src/monitor/debug/expr.c:91,make_token] match rules[3] = "\\$(eax|ecx|edx|ebx|esp|ebp|esi|edi|eip|ax|cx|dx|bx|sp|bp|si|di|al|cl|dl|bl|ah|ch|dh|bh)" at position 0 with len 4: $ecx
Success record : nr_token = 0, dtype = 259, str = ecx
100005: b9 27 00 10 00 movl $0x100027,%ecx
[src/monitor/debug/expr.c:91,make_token] match rules[3] = "\\$(eax|ecx|edx|ebx|esp|ebp|esi|edi|eip|ax|cx|dx|bx|sp|bp|si|di|al|cl|dl|bl|ah|ch|dh|bh)" at position 0 with len 4: $ecx
Success record : nr_token = 0, dtype = 259, str = ecx
Hardware watchpoint 0:$ecx
Old value:1944013018
New value:1048615
(nemu) info w
watchpoint:
NO. expr hitTimes
0 $ecx 1
(nemu) delete watchpoint 0
Unknown command 'delete'
(nemu) d 0
Success delete watchpoint 0
(nemu) info w
no watchpoint now
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x00100026
(nemu)

```



## 五、 必答题

### (一) 查阅 i386 手册回答问题

#### 1. EFLAGS 寄存器中的 CF 位是什么意思？

CF 是 Carry Flag 的缩写，位于 EFLAGS 的第 0 位，是标志进位的意思。用来允许一条指令的结果影响后面的指令，即一些指令用于设置高位进位或借位，否则清 0。该结果是位于 i386-2.3Register(P33) 和 i386-附录 C(P419)

#### 2.3.4 Flags Register

The flags register is a 32-bit register named EFLAGS. Figure 2-8 defines the bits within this register. The flags control certain operations and indicate the status of the 80386.

The low-order 16 bits of EFLAGS is named FLAGS and can be treated as a unit. This feature is useful when executing 8086 and 80286 code, because this part of EFLAGS is identical to the FLAGS register of the 8086 and the 80286.

The flags may be considered in three groups: the status flags, the control flags, and the systems flags. Discussion of the systems flags is delayed until Part II.

Figure 2-8. EFLAGS Register

