

1 PA1 – 开天辟地的篇章：最简单的计算机

1.2 表达式求值

数学表达式求值

词法分析

1) 为 token 类型添加规则。

添加了空格 `== () * / + - != && || !` 16 进制 10 进制 寄存器 变量等 16 种规则。根据正则表达式，例如 `+` 需要转义，所以 `+` 需要转义两次，第一次转义 `\`，`\` 再转义 `+`。而其他字符不需要，可直接用 `ascii` 码代替。如 `&&`、十进制等，运用正则表达式，并用 `enum` 给他们赋值。为了避免 `!=` 被识别为 `!` 和 `=` 的情况出现，我们将 `!=` 的顺序放在 `!` 前面。

代码如下：

```
1. enum {
2.     TK_NOTYPE = 256, TK_EQ, TK_UEQ, TK_logical_AND, TK_logical_OR, TK_logical_NO
   T, TK_register, TK_variable, TK_number, TK_hex
3.
4.     /* TODO: Add more token types */
5.
6. };
7.
8. static struct rule {
9.     char *regex;
10.    int token_type;
11. } rules[] = {
12.
13.     /* TODO: Add more rules.
14.      * Pay attention to the precedence level of different rules.
15.      */
16.
```

```

17.  {"+", TK_NOTYPE},    // spaces
18.  {"==", TK_EQ},      // equal
19.  {"\\(", '('},       // left parenthesis
20.  {"\\)", ')'},       // right parenthesis
21.  {"\\*", '*'},       // multiplication
22.  {"/", '/'},         // division
23.  {"\\+", '+'},       // plus
24.  {"-", '-'},         // subtraction
25.  {"!=", TK_UEQ},     // unequal
26.  {"&&", TK_logical_AND}, //logical AND
27.  {"\\|\\|\\|", TK_logical_OR}, //logical OR
28.  {"!", TK_logical_NOT}, //logical NOT
29.  {"0[xX][A-Fa-f0-9]{1,8}", TK_hex}, //hex
30.  {"\\$[a-zA-
    D][h1HL]|\\$[eE]? (ax|dx|cx|bx|bp|si|di|sp)", TK_register}, //register
31.  {"[a-zA-Z_][a-zA-Z0-9]*", TK_variable}, //variable
32.  {"[0-9]{1,10}", TK_number} //number
33.
34. };

```

2) make_token()函数。

识别出表达式中的单元 token。该函数读取每个位置上的字符，通过 for 循环，用 regexec 函数匹配，和前面定义的 rules[i] 中的正则表达式比较，pmatch.rm_so==0 表示匹配串在目标串中的第一个位置，pmatch.rm_eo 表示结束位置，成功识别得到该字符或者字符串的对应规则后，用 switch 语句将表达式中每一个部分用对应的数字表示 type，将 ==、十进制数等复制到 tokens[nr_token].str 中。

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 (regexec(&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_len, sub
               str_start);
17.             position += substr_len;
18.
19.             /* TODO: Now a new token is recognized with rules[i]. Add co
               des
20.             * to record the token in the array `tokens'. For certain ty
               pes
21.             * of tokens, some extra actions should be performed.
22.             */
23.
24.             switch (rules[i].token_type) {
25.                 case 257:
26.                     tokens[nr_token].type=257;
27.                     strcpy(tokens[nr_token].str, "=");
28.                     break;
29.                 case 40:
30.                     tokens[nr_token].type=40;
31.                     break;
32.                 case 41:
33.                     tokens[nr_token].type=41;
34.                     break;
35.                 case 42:
36.                     tokens[nr_token].type=42;
37.                     break;
38.                 case 47:
39.                     tokens[nr_token].type=47;
40.                     break;
41.                 case 43:
42.                     tokens[nr_token].type=43;
43.                     break;
44.                 case 45:
45.                     tokens[nr_token].type=45;
46.                     break;
47.                 case 258:
48.                     tokens[nr_token].type=258;
49.                     strcpy(tokens[nr_token].str, "!=");
50.                     break;
51.                 case 259:
52.                     tokens[nr_token].type=259;
53.                     strcpy(tokens[nr_token].str, "&&");
54.                     break;

```

```

55.             case 260:
56.                 tokens[nr_token].type=260;
57.                 strcpy(tokens[nr_token].str,"||");
58.                 break;
59.             case 261:
60.                 tokens[nr_token].type=261;
61.                 break;
62.             case 262:
63.                 tokens[nr_token].type=262;
64.                 strncpy(tokens[nr_token].str,&e[position-
        substr_len],substr_len);
65.                 break;
66.             case 263:
67.                 tokens[nr_token].type=263;
68.                 strncpy(tokens[nr_token].str,&e[position-
        substr_len],substr_len);
69.                 break;
70.             case 265:
71.                 tokens[nr_token].type=265;
72.                 strncpy(tokens[nr_token].str,&e[position-
        substr_len],substr_len);
73.                 break;
74.             default:
75.                 nr_token--;
76.                 break;
77.         }
78.         nr_token++;
79.         break;
80.     }
81. }
82.
83.
84.     if (i == NR_REGEX) {
85.         printf("no match at position %d\n%s\n%*.s^\n", position, e, posi
        tion, "");
86.         return false;
87.     }
88.
89. }
90. nr_token--;
91. return true;
92. }

```

递归求值

1) 检查左右括号是否匹配。

在该函数中，设置了两个变量 `left=0`，`flag=0`，若 `tokens[p]` 为左括号，则判断从 `tokens[p+1]` 到 `tokens[q]`。

遇到了左括号则 `left++`；

遇到右括号则 `left--`，且判断 `left` 是否等于 0 且当前位置是否到了末尾，若 `left` 为 0 且当前位置不在末位，`flag` 赋值 1，说明两侧括号不匹配，若 `left` 小于 0 则 `assert(0)`，因为说明出现了讲义中提到的类似 $(4+3)*((2+1))$ 的情况。最后若 `(left==0)&&(tokens[q]为右括号)&&flag!=1`，即该表达式仅有一对括号，位于两端，互相匹配，则返回 1；若 `left!=0` 则 `assert(0)`，说明可能出现了不匹配的括号；其他情况都返回 0；

结果截图：

```
for help, type help
(nemu) p (1+2)+(3-1)
[src/monitor/debug/expr.c,89,make_token] match rules[2] = "\" at position 0 with len 1: (
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 1 with len 1: 1
[src/monitor/debug/expr.c,89,make_token] match rules[6] = "+" at position 2 with len 1: +
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 3 with len 1: 2
[src/monitor/debug/expr.c,89,make_token] match rules[3] = "\" at position 4 with len 1: )
[src/monitor/debug/expr.c,89,make_token] match rules[6] = "+" at position 5 with len 1: +
[src/monitor/debug/expr.c,89,make_token] match rules[2] = "\" at position 6 with len 1: (
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 7 with len 1: 3
[src/monitor/debug/expr.c,89,make_token] match rules[7] = "-" at position 8 with len 1: -
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 9 with len 1: 1
[src/monitor/debug/expr.c,89,make_token] match rules[3] = "\" at position 10 with len 1: )
5
(nemu) p ((4-2)+1)-3
[src/monitor/debug/expr.c,89,make_token] match rules[2] = "\" at position 0 with len 1: (
[src/monitor/debug/expr.c,89,make_token] match rules[2] = "\" at position 1 with len 1: (
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 2 with len 1: 4
[src/monitor/debug/expr.c,89,make_token] match rules[7] = "-" at position 3 with len 1: -
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 4 with len 1: 2
[src/monitor/debug/expr.c,89,make_token] match rules[3] = "\" at position 5 with len 1: )
[src/monitor/debug/expr.c,89,make_token] match rules[6] = "+" at position 6 with len 1: +
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 7 with len 1: 1
[src/monitor/debug/expr.c,89,make_token] match rules[3] = "\" at position 8 with len 1: )
[src/monitor/debug/expr.c,89,make_token] match rules[7] = "-" at position 9 with len 1: -
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 10 with len 1: 3
0
(nemu) 
```

代码如下：

```
1. bool check_parentheses(int p,int q)
2. {
3.     int left=0;
4.     int flag=0;
5.     if(tokens[p].type==40)
6.     {
7.         left++;
8.         for(int i=p+1;i<=q;i++)
9.         {
10.             if(tokens[i].type==40)
11.             {
12.                 left++;
13.             }
14.             else if(tokens[i].type==41)
15.             {
16.                 left--;
```

```

17.         if(left==0&&i!=q)
18.             flag=1;
19.         if(left<0)
20.             assert(0);
21.     }
22. }
23. if(left==0&&tokens[q].type==41&&flag!=1)
24.     return 1;
25. else if(left==0)
26.     return 0;
27. else
28.     assert(0);
29. }
30. else
31.     return 0;
32. }

```

2) 找出 dominant operator。

其实做完之后发现自己用的方法太暴力太傻了.....完全可以在 rules 中再设置一个变量用来记录运算符优先级，方便简单，但是已经做完了暂时不改了，等空的时候改过来。

思路就是不考虑十进制十六进制寄存器和非，将 op 值设为起始位置

（若表达式为!2，则返回 op 为! 的位置），然后判断当前位置的运算符，（若遇到左括号，用 r 来记录括号的个数，然后从左右括号包含的范围之后的一位再判断运算符优先级，跳过括号内的运算符），与 op 位置的运算符比较优先级，优先级小于等于 op 位置，则 op=当前位置。其中，-和*需要再判断它是否是第一个位置或者前面一位是否为符号，右括号除外，为了避免将指针和负号错认。

代码如下：

```

1. int find_dominant_operator(int p, int q)
2. {
3.     int theop=p;
4.     int j=p;
5.     int r=0;
6.     for(;j<=q;j++)
7.     {
8.         if(tokens[j].type<262&&tokens[j].type!='!')
9.         {
10.            if(tokens[j].type==40)
11.            {
12.                r++;//判断括号的个数
13.                for(j=j+1;tokens[j].type!=41||r!=1;j++)
14.                {
15.                    if(tokens[j].type==40)

```

```

16.         r++;
17.         if(tokens[j].type==41)
18.             r--;
19.     }
20.     r=0;
21. }
22.     else if(tokens[j].type==260)    // 是||
23.         theop=j;
24.     else if(tokens[j].type==259&&(tokens[theop].type<260||tokens[theop].type>=262))
25.         theop=j;    //是&&
26.     else if(tokens[j].type==258&&(tokens[theop].type<259||tokens[theop].type>=262))
27.         theop=j;    //是!=
28.     else if(tokens[j].type==257&&(tokens[theop].type<258||tokens[theop].type>=262))
29.         theop=j;    //是==
30.     else if(tokens[j].type=='+'&&(tokens[theop].type<256||tokens[theop].type>=261))
31.         theop=j;
32.     else if(tokens[j].type=='-'&&(j==p||tokens[j-1].type>=256||tokens[j-
1].type=='')&&(tokens[theop].type<256||tokens[theop].type>=261))    //区分负号
33.         theop=j;
34.     else if(tokens[j].type=='/'&&(tokens[theop].type>=261||tokens[theop].type=='('||tokens[theo
p].type=='*'||tokens[theop].type=='/'))
35.         theop=j;
36.     else if(tokens[j].type=='*'&&(j==p||(tokens[j-1].type>=256||tokens[j-
1].type==''))&&(tokens[theop].type>=261||tokens[theop].type=='('||tokens[theop].type=='*'||tokens[theop].type=='/'))
37.         theop=j;
38.     }
39. }
40. return theop;
41. }

```

3) 递归求值。

eval(p,q) 函数大体上就是先判断表达式的首尾地址是否合理，不合理 **assert(0)**；再判断 **pq** 是否相等，若相等它最终必然是一个数值并返回它的值；然后运用 **check_parentheses** 函数判断该表达式是否被一对匹配的括号包围着，若是则递归求值括号包围的那个表达式。若以上判断都是否，则用 **find_dominant_operator** 函数找出最后一步运行的运算符所在位置 **op**，而后递归调用 **eval** 函数，求出 **op** 左右两端表达式的值 **val1**，**val2**，再根据 **op** 的运算符进行对应的操作并返回结果。

4) 实现算术表达式的递归求值。

其中提到了要区分出 $(4+3)*((2+1)$ 和 $(4+3)*(2-1)$ ，其实在 **check_parentheses** 函数中已经实现了，上文已经提到思路。

5) 实现带有负数的算术表达式的求值（选做）。

1. 负号和减号如何区分？

负号可以在第一位，或者它前面一位是运算符，)除外。而减号不是如此。

2. 负号是单目运算符，分裂时需注意什么？

不要和双目运算符减号搞混。与前一运算符分裂，与其后的一个数值或变量或寄存器之后的运算符分裂。

调试中的表达式求值

1) 扩展功能

需要实现十进制、十六进制、寄存器、左右括号，+、*、/、==、!=、与或非，指针解引用。其中，+、*、/、左右括号、十进制已经在上文中实现。==、!=、&&、||，同+类似。其余的功能实现的方法在代码贴图中解释了。

结果截图：

加减乘除

```
(nemu) p 1*2-3+4/2
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 0 with len 1: 1
[src/monitor/debug/expr.c,88,make_token] match rules[4] = "*" at position 1 with len 1: *
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 2 with len 1: 2
[src/monitor/debug/expr.c,88,make_token] match rules[7] = "-" at position 3 with len 1: -
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 4 with len 1: 3
[src/monitor/debug/expr.c,88,make_token] match rules[6] = "+" at position 5 with len 1: +
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 6 with len 1: 4
[src/monitor/debug/expr.c,88,make_token] match rules[5] = "/" at position 7 with len 1: /
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 8 with len 1: 2
1
```

负号

```
(nemu) p 2--1
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 0 with len 1: 2
[src/monitor/debug/expr.c,88,make_token] match rules[7] = "-" at position 1 with len 1: -
[src/monitor/debug/expr.c,88,make_token] match rules[7] = "-" at position 2 with len 1: -
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 3 with len 1: 1
3

(nemu) p (-1)*2
[src/monitor/debug/expr.c,88,make_token] match rules[2] = "(" at position 0 with len 1: (
[src/monitor/debug/expr.c,88,make_token] match rules[7] = "-" at position 1 with len 1: -
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 2 with len 1: 1
[src/monitor/debug/expr.c,88,make_token] match rules[3] = ")" at position 3 with len 1: )
[src/monitor/debug/expr.c,88,make_token] match rules[4] = "*" at position 4 with len 1: *
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 5 with len 1: 2
-2
```

寄存器、十六进制、==、!=、&&、||

```
(nemu) p $a1
[src/monitor/debug/expr.c,88,make_token] match rules[13] = "\\$[a-zA-Z][hHL]\\$[eE]?([xldx|c|b|p|s|d|l|sp])" at position 0 with len 3: $a1
5
(nemu) p $ax
[src/monitor/debug/expr.c,88,make_token] match rules[13] = "\\$[a-zA-Z][hHL]\\$[eE]?([xldx|c|b|p|s|d|l|sp])" at position 0 with len 3: $ax
8004
(nemu) █
```



```

-2
(nemu) p $eax
[src/monitor/debug/expr.c,88,make_token] match rules[13] = "$[a-zA-Z][hlHL]|\$[eE]?(axldx|cx|b|bp|s|d|l|sp)" at position 0 with len 4: $eax
971688242
(nemu) p 0xf
[src/monitor/debug/expr.c,88,make_token] match rules[12] = "0[xX][A-Fa-f0-9]{1,8}" at position 0 with len 3: 0xf
15
(nemu) p 0x111f
[src/monitor/debug/expr.c,88,make_token] match rules[12] = "0[xX][A-Fa-f0-9]{1,8}" at position 0 with len 6: 0x111f
4383
(nemu) p 2==1
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 0 with len 1: 2
[src/monitor/debug/expr.c,88,make_token] match rules[1] = "==" at position 1 with len 2: ==
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 3 with len 1: 1
0
(nemu) p 2!=2
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 0 with len 1: 2
[src/monitor/debug/expr.c,88,make_token] match rules[8] = "!=" at position 1 with len 2: !=
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 3 with len 1: 2
0
(nemu) p (1+2)&&0
[src/monitor/debug/expr.c,88,make_token] match rules[2] = "\" at position 0 with len 1: (
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 1 with len 1: 1
[src/monitor/debug/expr.c,88,make_token] match rules[6] = "+" at position 2 with len 1: +
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 3 with len 1: 2
[src/monitor/debug/expr.c,88,make_token] match rules[3] = "\" at position 4 with len 1: )
[src/monitor/debug/expr.c,88,make_token] match rules[9] = "&&" at position 5 with len 2: &&
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 7 with len 1: 0
0
(nemu) p (3-3)||0
[src/monitor/debug/expr.c,88,make_token] match rules[2] = "\" at position 0 with len 1: (
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 1 with len 1: 3
[src/monitor/debug/expr.c,88,make_token] match rules[7] = "-" at position 2 with len 1: -
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 3 with len 1: 3
[src/monitor/debug/expr.c,88,make_token] match rules[3] = "\" at position 4 with len 1: )
[src/monitor/debug/expr.c,88,make_token] match rules[10] = "||" at position 5 with len 2: ||
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 7 with len 1: 0
0

```

!、*指针解引用

```

(nemu) p !1+1
[src/monitor/debug/expr.c,88,make_token] match rules[11] = "!" at position 0 with len 1: !
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 1 with len 1: 1
[src/monitor/debug/expr.c,88,make_token] match rules[6] = "+" at position 2 with len 1: +
[src/monitor/debug/expr.c,88,make_token] match rules[15] = "[0-9]{1,10}" at position 3 with len 1: 1
1
(nemu) p *0x100000
[src/monitor/debug/expr.c,88,make_token] match rules[4] = "*" at position 0 with len 1: *
[src/monitor/debug/expr.c,88,make_token] match rules[12] = "0[xX][A-Fa-f0-9]{1,8}" at position 1 with len 8: 0x100000
1193144

```

括号

```

for netp, type netp
(nemu) p (1+2)+(3-1)
[src/monitor/debug/expr.c,89,make_token] match rules[2] = "\" at position 0 with len 1: (
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 1 with len 1: 1
[src/monitor/debug/expr.c,89,make_token] match rules[6] = "+" at position 2 with len 1: +
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 3 with len 1: 2
[src/monitor/debug/expr.c,89,make_token] match rules[3] = "\" at position 4 with len 1: )
[src/monitor/debug/expr.c,89,make_token] match rules[6] = "+" at position 5 with len 1: +
[src/monitor/debug/expr.c,89,make_token] match rules[2] = "\" at position 6 with len 1: (
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 7 with len 1: 3
[src/monitor/debug/expr.c,89,make_token] match rules[7] = "-" at position 8 with len 1: -
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 9 with len 1: 1
[src/monitor/debug/expr.c,89,make_token] match rules[3] = "\" at position 10 with len 1: )
5
(nemu) p ((4-2)+1)-3
[src/monitor/debug/expr.c,89,make_token] match rules[2] = "\" at position 0 with len 1: (
[src/monitor/debug/expr.c,89,make_token] match rules[2] = "\" at position 1 with len 1: (
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 2 with len 1: 4
[src/monitor/debug/expr.c,89,make_token] match rules[7] = "-" at position 3 with len 1: -
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 4 with len 1: 2
[src/monitor/debug/expr.c,89,make_token] match rules[3] = "\" at position 5 with len 1: )
[src/monitor/debug/expr.c,89,make_token] match rules[6] = "+" at position 6 with len 1: +
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 7 with len 1: 1
[src/monitor/debug/expr.c,89,make_token] match rules[3] = "\" at position 8 with len 1: )
[src/monitor/debug/expr.c,89,make_token] match rules[7] = "-" at position 9 with len 1: -
[src/monitor/debug/expr.c,89,make_token] match rules[16] = "[0-9]{1,10}" at position 10 with len 1: 3
0
(nemu)

```

代码如下：

```

1. int eval(int p,int q)
2. {
3.     int i=0;
4.     if(p>q)
5.         assert(0);
6.     else if(p==q)
7.     {
8.

```

```

9.             if(tokens[p].type==264) //为十进制数
10.            {
11.                sscanf(tokens[p].str,"%d",&i);
12.                return i;
13.            }
14.            else if(tokens[p].type==265) //为 16 进制
15.            {
16.                sscanf(tokens[p].str,"%x",&i);
17.                return i;
18.            }
19.            else if(tokens[p].type==262)
20.            { //寄存器，利用 regsl 等函数判断出它是 32 位或 16 位，并判断出位置
                是第几个，然后求值。8 位寄存器一个个判断。
21.                int j=0,s1=1,sw=1;
22.                for(;j<8&&s1!=0&&sw!=0;j++)
23.                {
24.                    s1=strcmp(tokens[p].str+1,regsl[i]);
25.                    sw=strcmp(tokens[p].str+1,regsw[i]);
26.                }
27.                if(s1==0)
28.                { i=cpu.gpr[j]._32;
29.                  return i;
30.                }
31.                else if(sw==0)
32.                    return cpu.gpr[j]._16;
33.                else
34.                {
35.                    if(strcmp(tokens[p].str,"$a1")==0)
36.                        return reg_b(0);
37.                    if(strcmp(tokens[p].str+1,"c1")==0)
38.                        return reg_b(1);
39.                    if(strcmp(tokens[p].str+1,"d1")==0)
40.                        return reg_b(2);
41.                    if(strcmp(tokens[p].str+1,"b1")==0)
42.                        return reg_b(3);
43.                    if(strcmp(tokens[p].str+1,"ah")==0)
44.                        return reg_b(4);
45.                    if(strcmp(tokens[p].str+1,"ch")==0)
46.                        return reg_b(5);
47.                    if(strcmp(tokens[p].str+1,"dh")==0)
48.                        return reg_b(6);
49.                    if(strcmp(tokens[p].str+1,"bh")==0)
50.                        return reg_b(7);
51.                }

```

```

52.             if(j==8)
53.                 assert(0);
54.         }
55.     else
56.         assert(0);
57. }
58. else if(check_parentheses(p,q)==true)
59.     return eval(p+1,q-1);
60. else
61. {
62.     int op,val1,val2;
63.     if((q-p==1)&&tokens[p].type=='-') //负号是会被视作 op 的, 所以
        它只能最后被分解为最小的表达式, 例如 3--2, 最后会 eval(2,3), 即-2。
64.         return 0-eval(q,q);
65.     if(((q-
        p==1)|| (tokens[p+1].type=='(' && tokens[q].type==')) && tokens[p].type==261)
66.         { //此处判断!, 要么是!2 要么是!(1-1)这两种情况, 进行判断。
67.             i=eval(p+1,q);
68.             return !i;
69.         }
70.     if(((q-
        p==1)|| (tokens[p+1].type=='(' && tokens[q].type==')) && tokens[p].type=='*')
71.         { //此处判断*, 要么是*0x100000 要么是*(0x10000+4)两种情况, 进
        行判断, 然后用 eval 得到整数, 用 vaddr_read() 读取内存。
72.
73.         return vaddr_read(eval(p+1,q),4);
74.     }
75.     op=find_dominant_operator(p,q);
76.
77.     val1=eval(p,op-1);
78.     val2=eval(op+1,q);
79.     switch(tokens[op].type)
80.     {
81.         case '+':return val1+val2;
82.         case '-':return val1-val2;
83.         case '*':return val1*val2;
84.         case '/':return val1/val2;
85.         case 257:
86.             if(val1==val2)
87.                 return 1;
88.             else
89.                 return 0;
90.         case 258:
91.             if(val1!=val2)

```

```

92.                                     return 1;
93.                                     else
94.                                     return 0;
95.                                     case 259:
96.                                     if(val1&&val2)
97.                                     return 1;
98.                                     else
99.                                     return 0;
100.                                    case 260:
101.                                    if(val1||val2)
102.                                    return 1;
103.                                    else
104.                                    return 0;
105.                                    default:assert(0);
106.                                }
107.        }
108.    return 0;
109. }

```

在 ui.c 函数中 cmd_p:

```

static int cmd_p(char *args)
{
    bool success;
    int i;
    i=expr(args,&success);
    printf("%d\n",i);
    return 0;
}

```

log

1.3 监视点

1)监视点的结构体

```

1. typedef struct watchpoint {
2.     int NO; //编号
3.     struct watchpoint *next;
4.     int value; //旧值,

```

```

5.  int newvalue; //新值,
6.  char type; //类型 监视点 w or 断点 b
7.  char Enb; //是否开启
8.  char str[32]; //被监视的表达式
9.  /* TODO: Add more members if necessary */
10.
11.
12. } WP;

```

3) 实现监视点的池的管理。

`new_wp` 是从 `free` 链表中取一个结点给 `head` 链表，且将表达式、值赋给它，修改开关，并输出该节点的编号。运用正则表达式判断是否为断点，若是则 `type` 为 `b`，否则为 `w`，具体见下文断点处。

```

1. WP* new_wp(char *string)
2. {
3.     WP *help;
4.     bool success;
5.     if(free_==NULL)
6.         assert(0);
7.
8.     help=free_;
9.     free_=free_>next;
10.    strcpy(help->str,string);
11.    help->value=expr(string,&success);
12.
13.    int status;
14.    int cflags = REG_EXTENDED;
15.    regmatch_t pmatch[1];
16.    const size_t nmatch =1 ;
17.    regex_t reg;
18.    const char * pattern="\\[eE][iI][pP][=][=]\[0x[0-9a-fA-F]{1,8}";
19.    regcomp(&reg,pattern,cflags);
20.    status=regexec(&reg,string,nmatch,pmatch,0);
21.    if(status == REG_NOMATCH)
22.        help->type='w';
23.    else if(status ==0)
24.        help->type='b';
25.
26.    help->Enb='Y';
27.    help->next=head;
28.    head=help;
29.    printf("The number is %d\n",help->NO);
30.    return help;
31. }

```

free_wp 函数是遍历 head 链表直到找出对应 NO 的结点，从 head 中删除，添加到 free 链表中。同时修改类型、表达式、值、开关。

```
1. void free_wp(int number)
2. {
3.     WP *p,*q;
4.     q=NULL;
5.     p=head;
6.     while(p!=NULL&& p->NO!=number)
7.     {
8.         q=p;
9.         p=p->next;
10.    }
11.    if(p==NULL)
12.        assert(0);
13.    if(q==NULL)
14.    {
15.        head=head->next;
16.        p->value=0;
17.        memset(p->str,0,sizeof(p->str));
18.        p->type=' ';
19.        p->Enb=N;
20.        p->next=free_;
21.        free_=p;
22.    }
23.    else
24.    {
25.        q->next=p->next;
26.        p->value=0;
27.        memset(p->str,0,sizeof(p->str));
28.        p->type=' ';
29.        p->Enb=N;
30.        p->next=free_;
31.        free_=p;
32.    }
33.    printf("Free the %d\n",p->NO);
34.    return;
35. }
```

3) 温故而知新

框架代码中定义 wp_pool 等变量时使用了关键字 static，在此处的含义是静态全局变量，该变量只能被本文件中的函数调用，并且是全局变量，而不能被同一程序其他文件中的函数调用。在此处使用 static 是为了避免它被误修改。

4) 实现以下功能:

- 1、如讲义中所言，每当 `cpu_exec()` 执行完一条指令，调用函数 `judge_wp` 对所有表达式求值判断是否变化，若变化则返回-1，暂停，输出提示并返回。

代码如下:

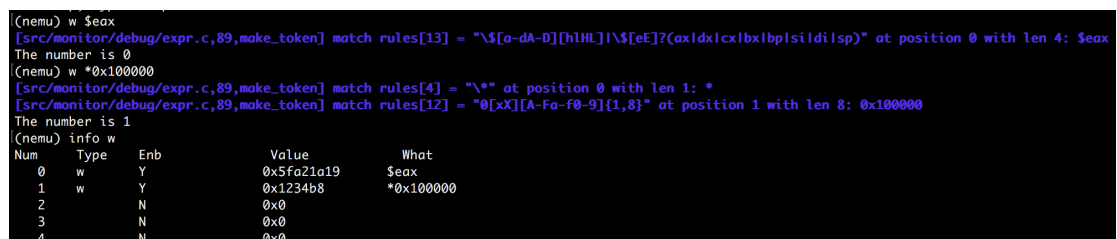
在 `cpu-exec.c` 中:

```
1.  /* TODO: check watchpoints here. */
2.      int judge=judge_wp();
3.      if(judge==-1)
4.      {
5.          nemu_state = NEMU_STOP;
6.          printf("Triggered the monitoring point\n");
7.          return;
8.      }
```

在 `watchpoint.c` 中:

```
1.  int judge_wp()
2.  {
3.      WP* p;
4.      bool success;
5.      p=head;
6.      if(p==NULL)
7.          return 0;
8.      while(p)
9.      {
10.         p->newvalue=expr(p->str,&success);
11.         if(p->newvalue!=p->value)
12.             return -1;
13.         p=p->next;
14.     }
15.     return 0;
16. }
```

结果截图:



The screenshot shows a debugger window with the following content:

```
(nemu) w $eax
[src/monitor/debug/expr.c,89,make_token] match rules[13] = "\\$[a-zA-Z][hlHL]\\$[eE]?([axldxlcxlbx|bpl|dils|p])" at position 0 with len 4: $eax
The number is 0
(nemu) w *0x100000
[src/monitor/debug/expr.c,89,make_token] match rules[4] = "*" at position 0 with len 1: *
[src/monitor/debug/expr.c,89,make_token] match rules[12] = "0[xX][A-Fa-f0-9]{1,8}" at position 1 with len 8: 0x100000
The number is 1
(nemu) info w
```

Num	Type	Enb	Value	What
0	w	Y	0x5fa21a19	\$eax
1	w	Y	0x1234b8	*0x100000
2		N	0x0	
3		N	0x0	
4		N	0x0	

```

31 N 0x0
(nemu) si
100000: b8 34 12 00 00 movl $0x1234,%eax
[src/monitor/debug/expr.c,89,make_token] match rules[4] = "*" at position 0 with len 1: *
[src/monitor/debug/expr.c,89,make_token] match rules[12] = "[0-9a-f]{1,8}" at position 1 with len 8: 0x100000
[src/monitor/debug/expr.c,89,make_token] match rules[13] = "[a-dA-D][hHlL]\[eE]?([x|d|x|b|p|s|i|d|l|s|p])" at position 0 with len 4: $eax
(nemu) si
100005: b9 27 00 10 00 movl $0x100027,%ecx
[src/monitor/debug/expr.c,89,make_token] match rules[4] = "*" at position 0 with len 1: *
[src/monitor/debug/expr.c,89,make_token] match rules[12] = "[0-9a-f]{1,8}" at position 1 with len 8: 0x100000
[src/monitor/debug/expr.c,89,make_token] match rules[13] = "[a-dA-D][hHlL]\[eE]?([x|d|x|b|p|s|i|d|l|s|p])" at position 0 with len 4: $eax
Triggered the monitoring point

```

3)使用 info w 来打印监视点信息。这里在 cmd_info 中调用了函数 print_wp(), 代码如下:

```

1. void print_wp()
2. {
3.     printf("Num\tType\tEnb \t\t Value\t\t What\n");
4.     for(int i=0;i<NR_WP;i++)
5.         printf("%4d\t%c\t%c\t\t\t0x%x\t\t%s\n",wp_pool[i].NO,wp_pool[i].
        type,wp_pool[i].Enb,wp_pool[i].value,wp_pool[i].str);
6. }

```

结果截图:

```

(nemu) w $eax
[src/monitor/debug/expr.c,89,make_token] match rules[13] = "[a-dA-D][hHlL]\[eE]?([x|d|x|b|p|s|i|d|l|s|p])" at position 0 with len 4: $eax
The number is 0
(nemu) w *0x100000
[src/monitor/debug/expr.c,89,make_token] match rules[4] = "*" at position 0 with len 1: *
[src/monitor/debug/expr.c,89,make_token] match rules[12] = "[0-9a-f]{1,8}" at position 1 with len 8: 0x100000
The number is 1
(nemu) info w
Num    Type    Enb      Value      What
0      w      Y      0x5fa21a19 $eax
1      w      Y      0x1234b8   *0x100000
2      N      N      0x0
3      N      N      0x0
4      N      N      0x0

```

3、删除监视点。在 cmd_d 中调用 free_wp 函数即可。

结果截图:

```

(nemu) d 0
Free the 0
(nemu) info w
Num    Type    Enb      Value      What
0      N      N      0x0
1      w      Y      0x1234b8   *0x100000
2      N      N      0x0
3      N      N      0x0
4      N      N      0x0
5      N      N      0x0
6      N      N      0x0

```

1.4 断点

1)断点;

代码在上面已经贴出。运用正则表达式判断是否为断点格式: \$eip==0x16 进制数字, regcomp()函数编译正则表达式, 执行成功返回 0, 则 type 为 b, 否则为 w。

截图:


```

The number is 0
(nemu) w $eip==0x100000
[src/monitor/debug/expr.c,89,make_token] match rules[14] = "\\$[Ee][Ii][Pp]" at position 0 with len 4: $eip
[src/monitor/debug/expr.c,89,make_token] match rules[1] = "==" at position 4 with len 2: ==
[src/monitor/debug/expr.c,89,make_token] match rules[12] = "0[xX][A-Fa-f0-9]{1,8}" at position 6 with len 8: 0x100000
The number is 1
(nemu) info w
Num      Type      Enb      Value      What
0        w          Y      0x133abd68  $eax
1        b          Y      0x1        $eip==0x100000
2        N          N      0x0
31       N          N      0x0
(nemu) si
100000:  b8 34 12 00 00      movl $0x1234,%eax
[src/monitor/debug/expr.c,89,make_token] match rules[14] = "\\$[Ee][Ii][Pp]" at position 0 with len 4: $eip
[src/monitor/debug/expr.c,89,make_token] match rules[1] = "==" at position 4 with len 2: ==
[src/monitor/debug/expr.c,89,make_token] match rules[12] = "0[xX][A-Fa-f0-9]{1,8}" at position 6 with len 8: 0x100000
Triggered the monitoring point

```

2) 一点也不能长?

必须的。因为当我们在调试器中对代码的某一行设置断点时，会把这里本来指令的第一个字节保存起来，然后写入一条 INT 3 指令，机器码为 0xcc，仅有一个字节，设置和取消断点时也需要保存和恢复一个字节。

断点机制不能正常工作。因为 INT3 断点将被调试进程中对地址处的字节替换为 0xcc，当 int3 指令的长度变成 2 个字节，其他指令同 x86，会导致这里本来指令的第一个字节被 2 个字节所代替，空间不够，不正确。

3) 随心所欲的断点。

将断点设置在指令的非首字节，就无法检测到断点。因为会检测函数的地址，读取它的第一个字节，判断是否等于 0xCCH。

4) NEMU 的前世今生。

模拟器 emulator 和调试器 debugger 的不同。我觉得 debugger 是一个命令行调试工具，设置断点，调试程序，测试 bug，然后 emulator 是一个载体，就是虚拟机，是一个独立的系统，不会对电脑本身的系统造成影响，我们可以借助它兼容不同的软件。

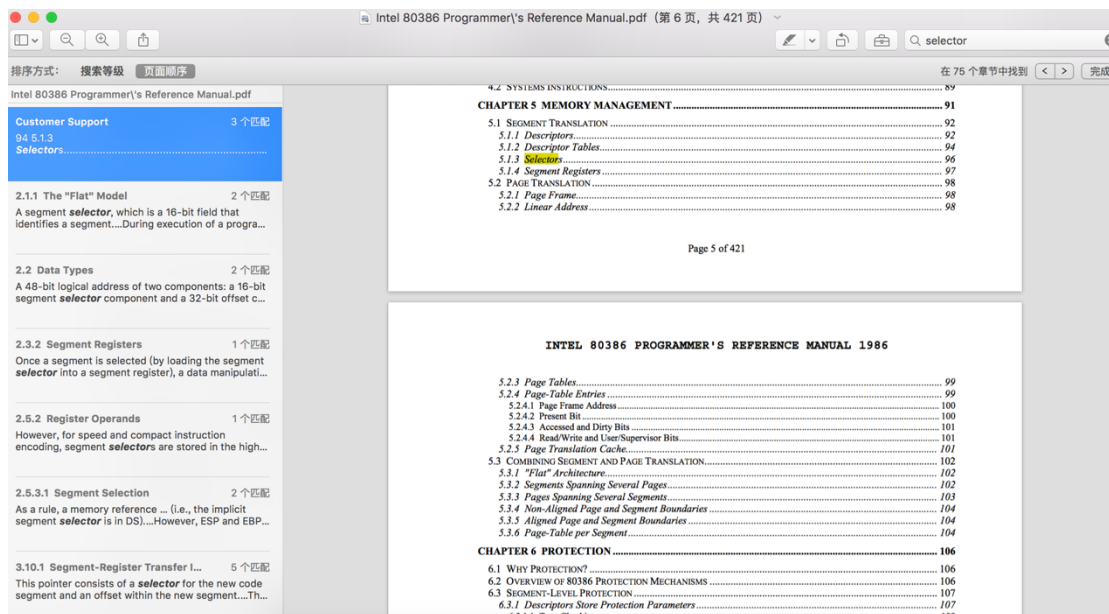
我觉得 gdb 和 nemu 中我们在实现的功能好像差不多，但是 gdb 是可以直接在函数某一行或是函数入口处设置断点，可以随便查看变量当前的值。

Log 截图：



1.5 i386 手册

1) 尝试通过目录定位关注的问题；



1.6 必答题：

1) 理解基础设施；

在调试上花费 75h，简易调试器帮它节省 50h。

2) 查阅 i386 手册；

1、EFLAGS 寄存器中的 CF 位是什么意思？

P34 页中提到参阅附录 c，CF 是进位标志；

P419 页中写到 CF 位：在最高位发生进位或者借位的时候将其置 1，否则清零。

2、ModR/M 字节是什么？

P241-243 页。ModR/M 由 Mod，Reg/Opcode，R/M 三部分组成。Mod 是前两位，提供寄存器寻址和内存寻址，Reg/Opcode 为 345 位，如果是 Reg 表示使用哪个寄存器，Opcode 表示对 group 属性的 Opcode 进行补充；R/M 为 678 位，与 mod 结合起来查图得 8 个寄存器和 24 个内存寻址

3、mov 指令的具体格式是怎么样的？

P347页。格式是 $DEST \leftarrow SRC$ 。

3)shell 命令；

.c 和.h 文件有 3994 行代码，使用命令：

```
1 find -name "[h]l[.]cpp" | xargs wc -l
```

```
nemu$ find . -name "[h]l[.]cpp" | xargs wc -l
wc: ./: Is a directory
0 .
39 ./runall.sh
8 ./include/nemu.h
29 ./include/common.h
45 ./include/debug.h
12 ./include/asm.h
```

```
wc: ./build/obj/misc: Is a directory
    0 ./build/obj/misc
wc: ./build/obj/cpu/exec: Is a directory
    0 ./build/obj/cpu/exec
3994 total
```

和框架代码相比，在 PA1 中写了 1008 行代码；

log 回转代码回到过去；除去空行之外，nemu/ 目录下的所有 .c 和 .h 文件总共有 3308 行代码。

```
017/nemu$ find . -name "*[.c|.h]" | xargs grep "^." | wc -l
grep: .: Is a directory
grep: ./src: Is a directory
grep: ./src/misc: Is a directory
grep: ./src/cpu/exec: Is a directory
grep: ./build/obj/misc: Is a directory
grep: ./build/obj/cpu/exec: Is a directory
3308
~/ics2017/nemu$
```

-Wall 使 GCC 产生尽可能多的警告信息，取消编译操作，打印出编译时所有错误或警告信息。

-Werror 要求 GCC 将所有的警告当成错误进行处理，取消编译操作。

使用-Wall 和-Werror 就是为了找出存在的错误，尽可能地避免程序运行出错，优化程序。