



南開大學
Nankai University

计算机学院
计算机系统设计

PA1-RISCV32 实验报告

姓名：李彦泽

学号：2012009

专业：计算机科学与技术

2023 年 3 月 21 日

目录

1 实验目的	2
2 实验内容	2
3 第一阶段	2
3.1 增加 nemu 的单步执行命令	2
3.2 增加 nemu 的打印寄存器状态命令	2
3.3 增加 nemu 的打印监视点信息命令	3
3.4 增加 nemu 的内存扫描命令	3
3.5 增加 nemu 的表达式求值命令	3
3.6 增加 nemu 的设置监视点命令	4
3.7 增加 nemu 的删除监视点命令	4
3.8 第一阶段完成	4
4 第二阶段	5
4.1 表达式求值	5
4.2 表达式求值功能测试	6
4.3 第二阶段完成	7
5 第三阶段	7
5.1 监视点结构体	7
5.2 监视点初始化及维护的全局变量	7
5.3 监视点申请	8
5.4 监视点删除	9
5.5 监视点打印	9
5.6 监视点检查	10
5.7 第三阶段完成	10
6 遇到的问题和解决方法	11
7 问答题	11
7.1 第一题【必答题】	11
7.2 第二题【必答题】	11
7.3 第三题【必答题】	12
7.4 第四题【必答题】	12
7.5 第五题	13
7.6 第六题	13
7.7 第七题	13
7.8 第八题	13

1 实验目的

完成 PA1 所有内容

2 实验内容

1. 增加 nemu 的单步执行命令
2. 增加 nemu 的打印寄存器状态命令
3. 增加 nemu 的打印监视点信息命令
4. 增加 nemu 的内存扫描命令
5. 增加 nemu 的表达式求值命令
6. 增加 nemu 的设置监视点命令
7. 增加 nemu 的删除监视点命令
8. 完善 nemu 表达式求值功能
9. 完善 nemu 监视点管理功能

3 第一阶段

3.1 增加 nemu 的单步执行命令

需要在 `nemu/src/monitor/sdb/sdb.c` 中增加如下代码

```
1 static int cmd_si(char *args){
2     if(args != NULL) cpu_exec(atoi(args));
3     else cpu_exec(1);
4     return 0;
5 }
```

3.2 增加 nemu 的打印寄存器状态命令

该指令的添加依赖于 `nemu/src/isa/RISCV32/reg.c` 中 `isa_reg_display()` 的函数, 该函数主体如下:

```
1 void isa_reg_display() {
2     printf("%-15s%-17s%-15s\n", "Registers", "Hexadecimal", "Decimal");
3     for(int i = 0; i < NR_REGS; ++i){
4         printf("%-15s0x%-15x%-15d\n", regs[i], cpu.gpr[i], cpu.gpr[i]);
5     }
```

```

6     printf("%-15s0x%-15x%-15d\n", "pc", cpu.pc, cpu.pc);
7 }

```

3.3 增加 nemu 的打印监视点信息命令

该指令需要在完成 nemu 监视点管理功能后添加 `print_wp()` 函数，之后在 `sdb.c` 中对其进行调用，该函数实现如下：

```

1  void print_wp(){
2      WP *itr = head;
3      if(head==NULL) {
4          printf("there is no watchpoint in the pool");
5          return;
6      }
7      printf("%-15s%-7s%s\n", "NO", "Exp", "Val");
8      while(itr != NULL){
9          printf("%02d\t%10s\t%-10u\n", itr->NO, itr->exp, itr->last);
10         itr = itr->next;
11     }
12 }

```

3.4 增加 nemu 的内存扫描命令

该指令的实现依赖于 `src/memory/vaddr.c` 中的 `vaddr_read()` 函数，`vaddr_read()` 函数已经在框架中实现此处不再赘述，该指令输出部分如下：

```

1  for(int i = 0; i < cnt; ++i){
2      printf("\t %04d %04d %04d %04d\n",
3          vaddr_read(addr,1), vaddr_read(addr+1,1), vaddr_read(addr+2,1), vaddr_read(addr+3,1));
4      addr += 4;
5  }

```

3.5 增加 nemu 的表达式求值命令

该指令的实现依赖于 `nemu/src/monitor/sdb/expr.c` 中的 `expr()` 函数，`expr()` 函数需要在第二阶段进行完善，该指令的实现如下：

```

1  static int cmd_p(char *args){
2      if(args == NULL) return 0;
3      bool success=false;
4      int res = expr(args,&success);
5      if(success == false)

```

```
6     printf("Invalid Expression\n");
7     else
8         printf("> %u\n", res);
9     return 0;
10 }
```

3.6 增加 nemu 的设置监视点命令

该指令的实现依赖于 `nemu/src/monitor/sdb/watchpoint.c` 中的 `new_wp()` 函数, `new_wp()` 函数需要在第三阶段进行完善, 该部分的实现如下:

```
1 static int cmd_w(char *args){
2     if(args == NULL) return 0;
3     if(new_wp(args)==NULL)
4         printf("the watch_point_pool is full!\n");
5     return 0;
6 }
```

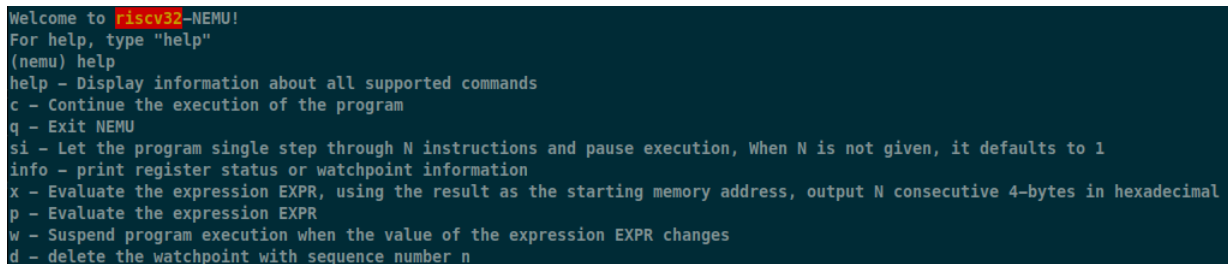
3.7 增加 nemu 的删除监视点命令

该指令的实现依赖于 `nemu/src/monitor/sdb/watchpoint.c` 中的 `free_wp()` 函数, `free_wp()` 函数需要在第三阶段进行完善, 该部分的实现如下:

```
1 static int cmd_d(char *args){
2     if(args == NULL) return 0;
3     free_wp(atoi(args));
4     return 0;
5 }
```

3.8 第一阶段完成

第一阶段完成后, 在 `nemu` 运行客户端镜像时支持如下 `DBG` 指令



```
Welcome to riscv32-NEMU!
For help, type "help"
(nemu) help
help - Display information about all supported commands
c - Continue the execution of the program
q - Exit NEMU
si - Let the program single step through N instructions and pause execution, When N is not given, it defaults to 1
info - print register status or watchpoint information
x - Evaluate the expression EXPR, using the result as the starting memory address, output N consecutive 4-bytes in hexadecimal
p - Evaluate the expression EXPR
w - Suspend program execution when the value of the expression EXPR changes
d - delete the watchpoint with sequence number n
```

图 3.1: help 指令输出

命令	格式	样例	说明
帮助	help	help	打印命令的帮助信息
继续运行	c	c	继续运行被暂停的程序
退出	q	q	退出 NEMU
单步执行	si [N]	si 10	单步执行 N 条指令后暂停
打印程序状态	info r	info r	打印寄存器状态
打印监视点信息	info w	info w	打印监视点信息
扫描内存	x N EXPR	x 10 \$esp	扫描特定内存位置
表达式求值	p EXPR	p \$eax + 1	求出表达式 EXPR 的值
设置监视点	w EXPR	w *0x2000	设置监视点
删除监视点	d N	d 2	删除序号为 N 的监视点

表 1: 指令列表

4 第二阶段

4.1 表达式求值

该部分代码较为庞大，具体细节请移步至`expr`中查看，该函数设计思路【分治】如下：

```

1  eval(p, q) {
2      if (p > q) {
3          /* Bad expression */
4      }
5      else if (p == q) {
6          /* Single token.
7           * For now this token should be a number.
8           * Return the value of the number.
9           */
10     }
11     else if (check_parentheses(p, q) == true) {
12         /* The expression is surrounded by a matched pair of parentheses.
13          * If that is the case, just throw away the parentheses.
14          */
15         return eval(p + 1, q - 1);
16     }
17     else {
18         op = the position of 主运算符 in the token expression;
19         val1 = eval(p, op - 1);
20         val2 = eval(op + 1, q);
21         switch (op_type) {
22             case '+': return val1 + val2;
23             case '-': /* ... */
24             case '*': /* ... */

```

```

25         case '/': /* ... */
26         default: assert(0);
27     }
28 }
29 }

```

4.2 表达式求值功能测试

该部分通过 `tools/gen-expr/gen-expr.c` 中的 `gen_rand_expr()` 函数生成随机的表达式，并通过格式化输出的方式获得对应答案，将表达式和答案存入 `input` 文件中，在 `monitor` 启动时对文件进行读取，调用表达式求值函数将结果与文件中的答案进行对比。

其中 `gen_rand_expr()` 函数思路如下：

```

1 void gen_rand_expr() {
2     switch (choose(3)) {
3         case 0:
4             gen_num();
5             break;
6         case 1:
7             gen('(');
8             gen_rand_expr();
9             gen(')'); break;
10        default:
11            gen_rand_expr();
12            gen_rand_op();
13            gen_rand_expr();
14            break;
15    }
16 }

```

随机表达式写入文件大致代码指导书中已给出，本文不再赘述。

函数调用及答案对比需要在 `nemu/src/nemu-main.c` 中添加如下代码，该代码会在 `nemu` 启动前对 `input` 文件中前 100 个表达式进行测试：

```

1 #ifdef CALCULATION_TEST
2     FILE * fp=fopen("tools/gen-expr/input","r");
3     char expression[70000];
4     char *str;char *unused;
5     unsigned answer=0;unsigned ans;
6     for(int i = 0;i < 100;i++){
7         bool flag=true;
8         expression[0]='\0';

```

```

9      unused=fgets(expression,70000,fp);
10     str = strtok(expression, " ");
11     sscanf(expression,"%u",&answer);
12     str=expression+strlen(expression)+1;
13     ans=expr(str,&flag);
14     if(answer!=ans)
15         printf("[%d] is wrong\n",i);
16     else
17         printf("[%d] is correct\n",i);
18 }
19 #endif

```

4.3 第二阶段完成

我们使用 gcc 对 *gen - expr.c* 进行编译后, 使用 *./gen - expr100 > input* 命令向 *input* 文件中写入了 100 个随机的测试样例, 之后对 *nemu* 进行编译, 使用 *make run* 命令对表达式求值功能进行测试, 终端显示如下:

```

1  [0] is correct
2  [1] is correct
3  ...
4  [99] is correct

```

表达式求值功能工作正常

5 第三阶段

5.1 监视点结构体

监视点结构体定义及解释如下:

```

1  typedef struct watchpoint {
2      int NO;                //监视点序号
3      struct watchpoint *next; //监视点后继
4      char exp[128];         //监视点表达式
5      uint32_t last;         //监视点上次访问时的值
6  } WP;

```

5.2 监视点初始化及维护的全局变量

主要代码如下:

```
1  static WP wp_pool[NR_WP] = {};
2  static WP *head = NULL, //正在使用的监视点链表头
3      *free_ = NULL, //空闲监视点链表头
4      *curr = NULL; //正在使用的监视点链表尾
5  void init_wp_pool() {
6      int i;
7      for (i = 0; i < NR_WP; i++) {
8          wp_pool[i].NO = i;
9          wp_pool[i].next = (i == NR_WP - 1 ? NULL : &wp_pool[i + 1]);
10         strcpy(wp_pool[i].exp, "\0");
11         wp_pool[i].last = 0;
12     }
13     head = NULL;
14     free_ = wp_pool;
15 }
```

5.3 监视点申请

监视点的申请实现如下：

```
1  WP* new_wp(char *expr){
2      if(head==NULL){
3          head=free_;
4          free_=free_->next;
5          head->next=NULL;
6          strcpy(head->exp,expr);
7          curr=head;
8      }
9      else{
10         if(free_==NULL)
11             return NULL;
12         else{
13             curr->next=free_;
14             free_=free_->next;
15             curr=curr->next;
16             curr->next=NULL;
17             strcpy(curr->exp,expr);
18         }
19     }
20     return curr;
21 }
```

5.4 监视点删除

监视点删除的主要代码如下：

```
1  WP* search_(WP *wp){//用于寻找前节点
2      WP* temp=head;
3      while(temp){
4          if(temp->next==wp) break;
5          temp=temp->next;
6      }
7      return temp;
8  }
9  void free_wp(int des){
10     WP *wp=head;
11     while(wp!=NULL){
12         if(wp->NO==des) break;
13         else wp=wp->next;
14     }
15     if(wp==head){
16         head=head->next;
17         wp->next=free_;
18         free_=wp;
19     }
20     else if(wp==curr){
21         wp->next=free_;
22         free_=wp;
23         curr=search_(wp);
24         curr->next=NULL;
25     }
26     else{
27         search_(wp)->next=wp->next;
28         wp->next=free_;
29         free_=wp;
30     }
31 }
```

5.5 监视点打印

监视点打印主要代码如下：

```
1  void print_wp(){
2      WP *itr = head;
```

```

3     if(head==NULL) {
4         printf("there is no watchpoint in the pool");
5         return;
6     }
7     printf("%-15s%-7s%s\n", "NO", "Exp", "Val");
8     while(itr != NULL){
9         printf("%02d\t%10s\t%-10u\n", itr->NO, itr->exp, itr->last);
10        itr = itr->next;
11    }
12 }
```

5.6 监视点检查

为了在 cpu 每执行过一条指令后都对全部的监视点进行检查，我们需要在 *nemu/src/cpu/cpu - exec.c* 中调用对监视点的检查，其中监视点检查代码如下：

```

1  void check_wp(){
2      bool success;
3      WP *itr = head;
4      while(itr != NULL){
5          uint32_t res = expr(itr->exp, &success);
6          if(!success){
7              printf("The expression of watch point %d is invalid!\n", itr->NO);
8          }
9          else if(res != itr->last){
10             printf("Num:%-6d\t Expr:%-20s\t New Val:%-14u\t Old Val:%-14u\n",
11                 itr->NO,
12                 itr->exp,
13                 res,
14                 itr->last);
15             itr->last = res;
16             nemu_state.state = NEMU_STOP;
17         }
18         itr=itr->next;
19     }
20 }
```

5.7 第三阶段完成

我们在终端中对 *w EXPR info w d N* 进行测试，结果如下：

```
1 //w 指令
2 w 2/2
3 w 1--1
4 //info 指令
5 0    2/2    1
6 1    1--1    2
7 //d 指令 +info 指令
8 d 0
9 1    1--1    2
```

监视点功能运行正常

6 遇到的问题和解决方法

暂时没有遇到什么有趣的问题，代码编写过程中遇到的问题都通过 google 解决了。

7 问答题

7.1 第一题【必答题】

问题：

理解基础设施

解答：

假设每次编译仅出现 1 个 bug：

- 用于 debug 的编译次数：450
- 使用 GDB 排除一个 bug 的时间： $30 * 20 = 600s$
- 使用简易调试器排除一个 bug 的时间： $10 * 20 = 200s$
- 节约的总时间： $(600 - 200) * 450 = 50h$

7.2 第二题【必答题】

问题：

查阅手册，回答下列 RISCV32 相关问题

解答：

- riscv32 的指令格式共有 6 种：R-型 (寄存器操作数指令)，I-型 (短立即数操作或取数指令)，S-型 (存数指令)，B-型 (条件跳转指令)，U-型 (长立即数操作指令)，J-型 (无条件跳转指令)。
- LUI 指令可以将 32 位无符号立即数放入指定的寄存器并将低 12 位设为 0。
- mstatus 结构见图 7.2：
- 手册中的具体位置见表 2

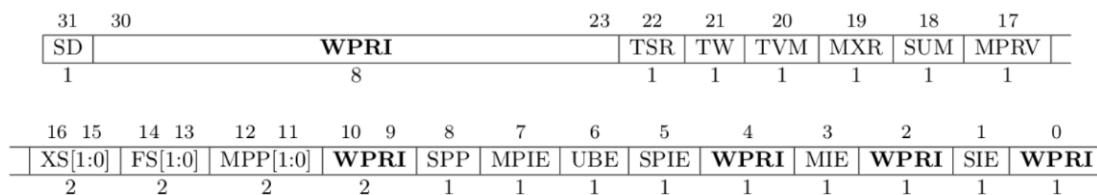


图 7.2: mstatus 结构

问题	手册中的位置
RISCV32 有哪几种指令格式	1.2 RISC-V ISA Overview 1.5 Base Instruction-Length Encoding 2.2 Base Instruction Formats
LUI 指令行为	2.4 Interger Computational INstructions
mstatus 寄存器结构	18 "N" Standard Extension for User-Level Interrupts

表 2: 手册阅读

7.3 第三题【必答题】

问题:

代码行数统计

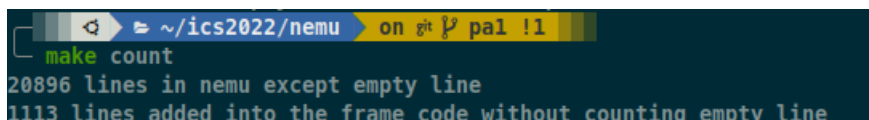
解答:

```

1 //pa1
2 find . -name "*.c" -or -name "*.h" | xargs cat | wc -l
3 24028
4 find . -name "*.c" -or -name "*.h" | xargs grep -Ev "^$" | wc -l
5 20896
6 //pa0
7 find . -name "*.c" -or -name "*.h" | xargs cat | wc -l
8 22933
9 find . -name "*.c" -or -name "*.h" | xargs grep -Ev "^$" | wc -l
10 19783

```

将 `makecount` 加入 `makefile` 后:



```

~/ics2022/nemu on p1 pa1 !1
$ make count
20896 lines in nemu except empty line
1113 lines added into the frame code without counting empty line

```

图 7.3: make count

7.4 第四题【必答题】

问题:

RTFM 参数问题

解答:

- -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.

打开 GCC 的警告。

- -Werror

Make all warnings into errors.

让 GCC 把警告视为错误进行处理。

7.5 第五题

问题:

程序从哪里开始执行

解答:

从 *main* 函数开始执行

7.6 第六题

问题:

cmd_c() 为什么传入了 -1

解答:

-1 是无符号整型中最大的数, 该参数使得 cpu 可以执行最多的指令数。

7.7 第七题

问题:

printf() 输出问题

解答:

unix 上标准输入输出都是带有缓存的, 一般是行缓存。对于标准输出, 需要输出的数据并不是直接输出到终端上, 而是首先缓存到某个地方, 当遇到行刷新标志或者该缓存已满的情况下, 才会把缓存的数据显示到终端设备上。换行符 '\n' 可以认为是行刷新标志。所以, *printf* 函数没有带 '\n' 是不会自动刷新输出流, 直至缓存被填满。

7.8 第八题

问题:

为什么要将变量设置为 *static*

解答:

将变量设为全局变量, 方便全局访问和修改。