CS211 Lab3 Report

Name: Feng Xiao
ID Number: 2023234296

2023-12-2

1 Implementation of system call in 5 stages pipeline

1.1 Design and implementation details

- When the simulator first starts running, the prepared system call code needs to be loaded to the specified location, and the first addresses of different system calls are recorded in the system call processing vector.
- When the pipeline executes a system call instruction, the pipeline will no longer execute instructions after the system call instruction, and will complete and commit the instructions before the system call.
- The current register state will be saved at this time, and the PC of the system call instruction will be saved in the SPEC register and clear the pipeline.
- According to the system call function vector, the PC becomes the first address of the system call resolution function.
- At the end of the system call, use the sret instruction to push out the system call status, which is the same as the context switch mentioned above. It should be noted here that when restoring the register, the a0 register cannot be overwritten. The return value of the system call is saved here.
- Depending on the system call to be processed, we may set the PC to the position of the next instruction in the SPEC register or the current position. In this implementation, all system calls are the former.
- I implemented two system calls, namely No. 7 and No. 8. System call No. 7 has three parameters. The first is the array size, the second is the first address of the initial array, and the third is the first address of the destination array. The function to be implemented is to move the values in the initial array to the destination array. . System call No. 8 has two parameters. The first is the number of arrays, and the second is the first address of the array. Its function is to obtain the maximum value in the given array and return it.

1.2 Test

My test code is as follows,it is riscv-elf/test_mysyscall.riscv

```
#include "lib.h"
int main() {
    int a=11;
    int b[11]={12,323,4535,4567,576,456,3434,324,346345,0,5234};
    int c[11];
    set_array(a,b,c);
    for(int i=0;i<11;i++){
        print_d(c[i]);
        print_c('\n');
    }
    a=get_arraymax(a,c);
```

```
12      print_d(a);
13      print_c('\n');
14      exit_proc();
15  }
16
17  void set_array(int num,int*orignal,int*final)
18  {
19      asm("li a7,7;"
20          "scall"
21      );
22  }
23
24  int get_arraymax(int num,int* orignal)
25  {
26      int result;
27      asm("li a7,8;"
28          "scall"
29      );
30      asm("addi %0, a0, 0" : "=r" (result));
31  }
```

The running results are as follows,In line with expectations,

```
# fxiao @ fxiao-XiaoXinPro-16-ARH7 in ~/CS211/CS211/lab3/build on git:master x [
21:37:26]
$ ./Simulator ../riscv-elf/test_mysyscall.riscv
12
323
4535
4567
576
456
3434
324
346345
0
5234
346345
Program exit from an exit() system call
------------ STATISTICS -----------
Number of Instructions: 793
Number of Cycles: 1227
Avg Cycles per Instrcution: 1.5473
Branch Perdiction Accuacy: 0.5556 (Strategy: Always Not Taken)
Number of Control Hazards: 117
Number of Data Hazards: 416
Number of Memory Hazards: 70
-----------------------------------
```

Figuur 1: Test result