ICS Homework 5

March 15, 2022

1 Organization

1.1 Hazard

```
1  # demo.ys
2  0x000:    irmovq stack,%rsp
3  0x00a:    call p
4  0x013:    irmovq $5,%rsi
5  0x01d:    halt
6
7  0x020:.pos 0x20
8  0x020:p:  irmovq $-1,%rdi
9  0x02a:    ret  #below will not be executed
10 0x02b:    irmovq $1,%rax
11 0x035:    irmovq $2,%rcx
12 0x03f:    irmovq $3,%rdx
13 0x049:    irmovq $4,%rbx
14
15 0x100:.pos 0x100
16 0x100:stack:
```

1. During executing the above example, how many hazards will happen? Please point them out.
   SOLUTION:
   There are 3 hazards.
   Data hazard between "irmovq stack,%rsp" and "call p". %rsp is written by the previous instruction and is read by the latter instruction.
   Data hazard between "call p" and "ret". %rsp is written by the previous instruction and is read by the latter instruction.
   Control hazard due to the "ret" instruction.

2. How could the above data hazards be handled? Please describe in detail.
   SOLUTION:
   Data hazards are handled by forwarding.
   For the first data hazard, the call instruction at stage **Decode** will get the value of %rsp through **e_valE** in the **Execute** stage of irmovq instruction.
   For the second data hazard, the ret instruction at the **Decode** stage will get the value of %rsp through **M_valE** in the **Memory** stage of the call instruction.

3. What is the difference between **stall** and **bubble**?
   SOLUTION:
   A pipeline stall means input memory of a stage remains the same as the previous cycle. A pipeline bubble means input memory of a stage is the same as a nop instruction. When an instruction stalls in a stage, a bubble is injected into its subsequent stage.

2 System Software

2.1 Signal

```
1  int counter = 2;
2
3  void handler1(int sig) {
4      counter = counter + 1;
5      printf("%d\n", counter);
6      exit(0);
7  }
8
9  int main() {
10     signal(SIGINT, handler1);
11     printf("%d\n", counter);
12     if ((pid = fork()) == 0) {
13         while(1) {};
14     }
15     kill(pid, SIGINT);
16
17     counter = counter -1;
18     printf("%d\n", counter);
19     waitpid(-1, NULL, 0);
20     counter = counter + 1;
21     printf("%d\n", counter);
22     exit(0);
23 }
```

1. Please rewrite the *handler* according to the guidelines in section 8.5.5 (HINT: you can use *Sio_puts* as thread safe *printf* if needed).

```
1  volatile sig_atomic_t counter = 2;
2
3  void handler1(int sig) {
4      int olderrorno = errorno;
5      sigset_t mask, prev_mask;
6      Sigfillset(&mask);
7      Sigprocmask(SIG_BLOCK, &mask, &prev_mask);
8      counter = counter + 1;
9      Sio_putf("%d\n", counter);
10     Sigprocmask(SIG_BLOCK, &prev_mask, NULL);
11     errorno = olderrorno;
12     _exit(0);
13 }
14
15 int main {
16     .........
```

2. Please write down all the possible outputs of the original programs.
SOLUTION:
2\n3\n1\n2\n or 2\n1\n3\n2\n