# ICS EXE 2

March 10, 2021

## 1 SEQ Y86-64

`call` and `ret` instructions access stack to save and load return addresses. Please consider a different set of these instructions:

- `callr Dest`, which saves the return address in `%rsi` and jumps to the instruction at `Dest`;
- `retr`, which jumps to the instruction with the address saved in `%rsi`.

### 1.1

Please fill in the blanks below to describe these two instructions which are executed in sequential Y86-64 implementation. You only need to write the generic logic for each instruction. (Note that you are allowed to add new wires in the base SEQ Y86-64 hardware architecture.)

| Stage | Generic | Generic |
|---|---|---|
| | `callr Dest` | `retr` |
| Fetch | | |
| Decode | | |
| Execute | | |
| Memory | | |
| Write back | | |
| PC update | | |

### 1.2

For any correctly implemented Y86-64 program, if we simply replace all `call` instructions with `callr` and all `ret` with `retr`, will the program run correctly? Why or why not?

## 2 Process

Assume we have the following code:

```
1  #include <sys/types.h>
2  #include <unistd.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int x = 10;
7  int main() {
8    int y = 1, i;
9    for (i = 0; i < 2; ++i) {
10     pid_t pid = fork();
11     if (!pid) {
12       x += y;
13       printf("x in child: %d\n", x);
14       exit(0);
15     }
16     y++;
17     x <<= 1;
18     printf("x in parent: %d\n", x);
19   }
20   return 0;
21 }
```

### 2.1

Give a possible output of the program

### 2.2

Based on Q2.1, how to use waitpid to make sure the "x in child" and the "x in parent" are interleaved? Will this influence the value of x in output?

### 2.3

Based on Q2.1, please list all possible output order. (or show the constraints on the output order).

### 2.4

Based on Q2.1, if we accidentally delete the line 14, please give a possible output.

# 3 Waitpid and Zombie

Assume we have the following code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <sys/wait.h>
6
7  int main(void) {
8    if (fork() == 0) {
9      if (fork() == 0) { // Proc 1
10       exit(0);
11     } else { // Proc 2
12       waitpid(-1, NULL, 0);
13       exit(0);
14     }
15   } else { // Proc 3
16     while (waitpid(-1, NULL,
17                    WNOHANG) > 0) ;
18     exit(0);
19   }
20   return 0;
21 }
```

## 3.1

How many zombie processes could the program leave for init to reap? Please show all possible number and show one execution order for each. (Assume that shell will not reap the main process)

## 3.2

What if we remove the line 12?