ICS 2021 QUIZ 1

学号_____ 姓名 _____

Problem 1 (Process Control)

What is the output of following code?

```
int main() {
    if (fork() && (!fork())) {
        if (fork() || fork()) {
            fork();
        }
    }
    printf("3 ");
    while(waitpid(-1, NULL, 0) > 0);
    return 0;
}
```

Output is "3 3 3 3 3 3 3 " (7x"3 ")

Problem 2 (Signals)

Please read the following code:

```
1. volatile sig_atomic_t child_exit_num = 0;
2. volatile sit_atomic_t sig_user1_cnt = 0;
3. void handle_sigchld(int sig) {
4.     if (waitpid(-1, NULL, 0) > 0) {
5.         child_exit_num++;
6.         Sio_puts("handle sigchld");
7.     }
8. }
9. void handle_sig_usr1(int sig) { sig_user1_cnt++; }
10.void handle_sig_alarm (int sig) { Sio_puts("handle alarm"); }
11.int main() {
12.    sigset_t mask_all, prev_one;
13.    Sigfillset(&mask_all);
14.    Sigprocmask(SIG_BLOCK, &mask_all, &prev_one);
15.    Signal(SIGCHLD, handle_sigchld);
16.    Signal(SIGUSER1, handle_sig_usr1);
```

```
17.    Signal(SIGALRM, handle_sig_alarm);
18.    for (int i = 0; i < 3; i++) {
19.        if ((!i) && (Fork() == 0)) {
20.            Sigprocmask(SIG_SETMASK, &prev_one);
21.            for (int j = 0; j < 3; j++)
22.                kill(getppid(), SIGUSER1);
23.            alarm(0.0001 /*0.0001s*/);
24.            if (sig_user1_cnt)
25.                 printf("sig_user1_cnt != 0\n");
26.            Pause();
27.            exit(0);
28.        } else if (i && (Fork() == 0)) {
29.            printf("I'm child %d\n", i);
30.            exit(0);
31.        }
32.    }
33.    Sigprocmask(SIG_SETMASK, &prev_one);
34.    printf("sig_user1_cnt: %d\n", sig_user1_cnt);
35.    while (child_exit_num != 3)
36.        Pause();
37.    printf("End\n");
38.    return 0;
39.}
```

Note: (1)The kernel checks the pending bits when it switches from kernel to user (e.g. returning from system call). The return of signal handler will trigger **sigreturn** system call; (2) We assume that **alarm()** can arrange for a **SIGALRM** signal to be delivered to the calling process with a finer granularity than second.

1. Please fill the following table:

For visibility, please write **T** if the line will always be printed, **F** if the line will never be printed, **M** if the line might be printed.

Line	Visibility
6	M
10	M
25	F
29	T
34	T
37	M

2. Please show all possible values of sig_user1_cnt in **Line 34**. 0/1/2/3

3. There are several bugs in the program, which could make the program cannot terminate. Please figure out all the bugs, in which execution order they cannot terminate and how to fix them.

Bug 1: If SIGCHLD is received between 54 and 55, parent cannot terminate

Fix 1: change Line 35~36 to:

```
While (child_exit != 3)
    sigsuspend(&prev_one);
```

Bug 2: If all signal sent to child is receive between 41 and 42, child cannot terminate

Fix 2: change Line 26 to:

```
sigsuspend(&prev_one);
```

Bug 3: If multiple children exit when parent is handling the SIGCHLD signal, the child_exit_num cannot be 3 finally.

Fix 3: change Line 4 to:

```
while (waitpid(-1, NULL, 0) > 0)
```

Problem 3 (Pipeline-1)

Suppose we are using the final pipeline implementation from Figure 4.52 in CSAPP book. Now, we want to add a new instruction:indirect jump with register, **`jmpr (%rA)`**, to the original Y86-64 instruction set, which jumps to the address stored in the memory address pointed by register rA, using the following encoding:

	Byte	0		1	
jmpr		E	0	rA	F

For example, if **%rax=0x100** and **0x100=0x200** in memory, then **`jmpr (%rax)`** will jump to address 0x200.

1. Please fill in the generic function of each stage for **`jmpr`** like Figure 4.21 in CSAPP book.

Field	jmpr
Fetch	icode:ifun ← M1[PC] rA:rB ← M1[PC+1]=rA:F
Decode	valA ← R[rA]
Execute	valE ← valA + 0
Memory	valM ← M8[valE]
Write Back	-
PC Update	PC ← valM

2. There will be some hazards due to this new instruction **jmpr**. Please list new detection conditions like Figure 4.64 and new control actions like Figure 4.66. Use "--" for no additional control action("normal"). **Note**: Suppose icode of **jmpr** is **IJMPR**.

Condition	Trigger				
Jmpr_hazard	IJMPR in {D_icode,E_icode,M_icode}				
	Pipeline register				
	F	D	E	M	W
	S	B	-	-	-

3. Does **jmpr** cause any new combinations of hazard? If there is, select one and fill in the pipeline states figure (see Figure 4.67) and list pipeline control action (see the table right next to Practice Problem 4.37).

M	-
E	load
D	use

M	-
E	-
D	jmpr

Condition	Pipeline register				
	F	D	E	M	W
jmpr	S	B	-	-	-
[12]	S	S	B	-	-
combination	S	S	B	-	-

Another one is jxx and jmpr.

4. Given the frequency of instructions and the frequency these hazards arise. Please fill the table and calculate the CPI (cycles per instruction).

Cause	Instruction Frequency	Condition frequency	Bubbles	Product
Load/Use	0.20	0.2	1	0.04
Mispredict jump	0.10	0.4	2	0.08
Return	0.01	1.0	3	0.03
Jmpr_hazard	0.05	1.0	3	0.15

CPI= __1.30__

Problem 4 (Pipeline-2)

Suppose we wanted to create a lower-cost pipelined processor based on the structure we devised for PIPE– (Figure 4.41), without any bypassing. This design would handle all data dependencies by **stalling** until the instruction generating a needed value **has passed through** the write-back stage.

The file contains a modified version of the HCL code for PIPE in which the bypassing logic has been disabled. That is, the signals **d_valA** and **d_valB** are simply declared as follows:

```
word d_valA = [
    D_icode in { ICALL, IJXX } : D_valP; # Use incremented PC
    1 : d_rvalA; # Use value read from register file
]
## No forwarding. valB is value from register file
word d_valB = d_rvalB;
```

Please analyze possible data hazards and answering the following questions.

1. Please describe the **Detection Condition** of the data hazard like Figure 4.64.

```
bool data_hazard = (d_srcA != RNONE &&
      d_srcA in { e_dstE, E_dstM, M_dstM, M_dstE, W_dstM, W_dstE })
       || (d_srcB != RNONE &&
       d_srcB in { e_dstE, E_dstM, M_dstM, M_dstE, W_dstM,
W_dstE });
```

2. Please describe the **Control Action** of the data hazard like Figure 4.66.

Data Hazard	Trigger				
	data_hazard is True				
	Pipeline register				
	F	D	E	M	W
	S	S	B	-	-

3. Below we give the detection conditions of several hazards just same as Figure 4.64. Analyze different combinations of control cases, and use **data_hazard, jxx_error, ret and load_use** to implement the PIPE- control logic. (Hint: **load_use** is just a

kind of **data_hazard)**.

```
   ## Detection conditions in PIPE-. data_hazard is not provided
here
   bool jxx_error = (E_icode = IJXX && !e_Cnd)
   bool ret = IRET in { D_icode, E_icode, M_icode };
   bool load_use = (E_icode in {IMRMOVQ, IPOPQ} &&
                       E_dstM in {d_srcA, d_srcB})
   ## You implementation of PIPE- control logic
   bool F_stall = (data_hazard || ret);
   bool F_bubble = 0;
   bool D_stall = data_hazard & !jxx_error;
   bool D_bubble = jxx_error ||(!data_hazard&& ret);
   bool E_stall = 0;
   bool E_bubble = jxx_error || data_hazard;
   ## The control logical of M and W does not change.
```