Question 2. Thread Switching [5 MARKS]

You know that implementing thread switching is a little tricky. To simplify matters, you start by implementing thread switching between just two threads, A and B, as shown below. You have added A and B to the run queue, and initialized their thread context so that they will start executing the thread_A() and thread_B() functions when they are run for the first time. Your scheduler runs Thread A first.

```c
int i = 0;
ucpcontext_t uA, uB;

thread_A() {
    int d = 0;
    while (i < 3) {
        i++;
        printf("A:%d\n", i);
        d = 0;
        getcontext(&uA);
        if (d == 0) {
            d = 1;
            setcontext(&uB);
        }
    }
}

thread_B() {
    int d = 1;
    while (i < 3) {
        i++;
        printf("B:%d\n", i);
        d = 1;
        getcontext(&uB);
        if (d == 1) {
            d = 0;
            setcontext(&uA);
        }
    }
}
```

Part (a) [3 MARKS] Circle the output you expect to see? Hint: Think carefully about what is saved by the getcontext() function.

A) A:1 A:2 A:3
B) A:1 B:2 B:3
C) A:1 B:2 A:3 B:4 A:5 B:6
D) A:1 B:2
E) A:1 B:1
F) A:1 B:2 A:3

Part (b) [2 MARKS] Briefly explain why you have chosen the answer above.
Question 2. Thread Switching [5 MARKS]

You know that implementing thread switching is a little tricky. To simplify matters, you start by implementing thread switching between just two threads, A and B, as shown below. You have added A and B to the run queue, and initialized their thread context so that they will start executing the thread_A() and thread_B() functions when they are run for the first time. Your scheduler runs Thread A first.

```
int i = 0;
ucontext_t uA, uB;

thread_A() {
    int d = 0;
    while (i < 3) {
        i++;
        printf("A:%d\n", i);
        d = 0;
        getcontext(&uA);
        if (d == 0) {
            d = 1;
            setcontext(&uB);
        }
    }
}

thread_B() {
    int d = 1;
    while (i < 3) {
        i++;
        printf("B:%d\n", i);
        d = 1;
        getcontext(&uB);
        if (d == 1) {
            d = 0;
            setcontext(&uA);
        }
    }
}
```

Part (a) [3 MARKS] Circle the output you expect to see? Hint: Think carefully about what is saved by the getcontext() function.

A) A:1 A:2 A:3
B) A:1 B:2 B:3
C) A:1 B:2 A:3 B:4 A:5 B:6
D) A:1 B:2
E) A:1 B:1
F) A:1 B:2 A:3

We gave 3 marks for a correct answer.

For an incorrect answer, we gave 1 partial mark for Part(a) if Part(b) gave a reasonable explanation for the choice here.

Part (b) [2 MARKS] Briefly explain why you have chosen the answer above.

The main idea that we expected students to understand is that the getcontext call saves the stack pointer and not the values of the stack variables. Execution happens as follows:

print A:1, a:d=0, a:getcontext, a:d=1, a:setcontext, print B:2, b:d=1,b:getcontext,b:d=0,b:setcontext
now we start running a at the point its getcontext was called. The value of a:d is 1 (note that it was set after the getcontext call), so we go to the next iteration of a's loop.

print A:3, a:d=0, a:getcontext, a:d=1,a:setcontext
now we start running b at the point its getcontext was called. The value of b:d is 0, so we go to the next iteration of b's loop. However, i is 3 at this point, so b stops. Thread a nevers runs again.