1.) Ideally, switching a single-cycle datapath that has a clock cycle time $T$ to be a pipelined datapath with $n$ stages would mean that the pipelined datapath's clock cycle time would be $T/n$ and an instruction would complete execution on the pipelined datapath every $T/n$ units of time. We've talked about 4 issues that prevent the pipelined datapath from actually completing an instruction every $T/n$ units of time. What are those 4 issues that prevent us from achieving the ideal benefits of pipelining?

- uneven stages
- pipelined register delay
- data hazards
- control hazards

2.) Suppose you have a 5-stage pipelined datapath. If an exception occurs while executing one of the instructions in the pipelined datapath, the datapath will invoke the exception handler to figure out what to do in response to the exception. When will the datapath invoke the exception handler in order to deal with the instruction that raised the exception?

*We handle an exception when an instruction reaches the writeback stage*

3.) When an instruction causes an exception in a pipelined datapath, we want to make sure that instruction causing the exception and any instructions in the pipeline that come after the instruction causing the exception do not modify the state of the processor. What are the 3 different pieces of state in the datapath that we want to prevent being modified by the instruction that caused the exception and the instructions following it in the pipeline?

- register file
- memory
- condition codes

4.) Consider the following code:

```c
int arr[100];
int x;

for(int i = 0; i < 100; i++){
    if( i % 25 == 0){
        x+= arr[i];
    }
    else{
        x++;
    }
}
```
Consider the variables x and arr. Does the code's use of arr exhibit good or bad spatial locality? Does the code's use of arr exhibit good or bad temporal locality? Does the code's use of x exhibit good or bad temporal locality?

- *arr has poor spatial locality because we only access every 25th element.*
- *arr has poor temporal locality because we never access an element more than once, so there is no reuse over time*
- *x has fantastic temporal locality as it is accessed on every loop iteration*