1a) Prove XORing an operand with itself changes the operand to zero.

XOR Truth table

a	b	result
0	0	0
0	1	1
1	0	1
1	1	0

Therefore: if a bit is zero and you execute XOR bitwise with another bit that is zero, the result is zero. Likewise, if the two bits are one, the result is zero. The result is only one if the two bits are different. That is to say if the first bit is one, the second must be zero for the XOR to result in one, and vice versa.

ex: bitwise XOR on Hex: 0x6 ⇒ binary: 0110

Start bit	0	1	1	0
Second bit	0	1	1	0
Result	0	0	0	0

1b) we can show this in gdb by observing value saved in the eax register after using XOR eax, eax (assuming a non-zero value has already been placed in eax).

2a) After observing a simple:

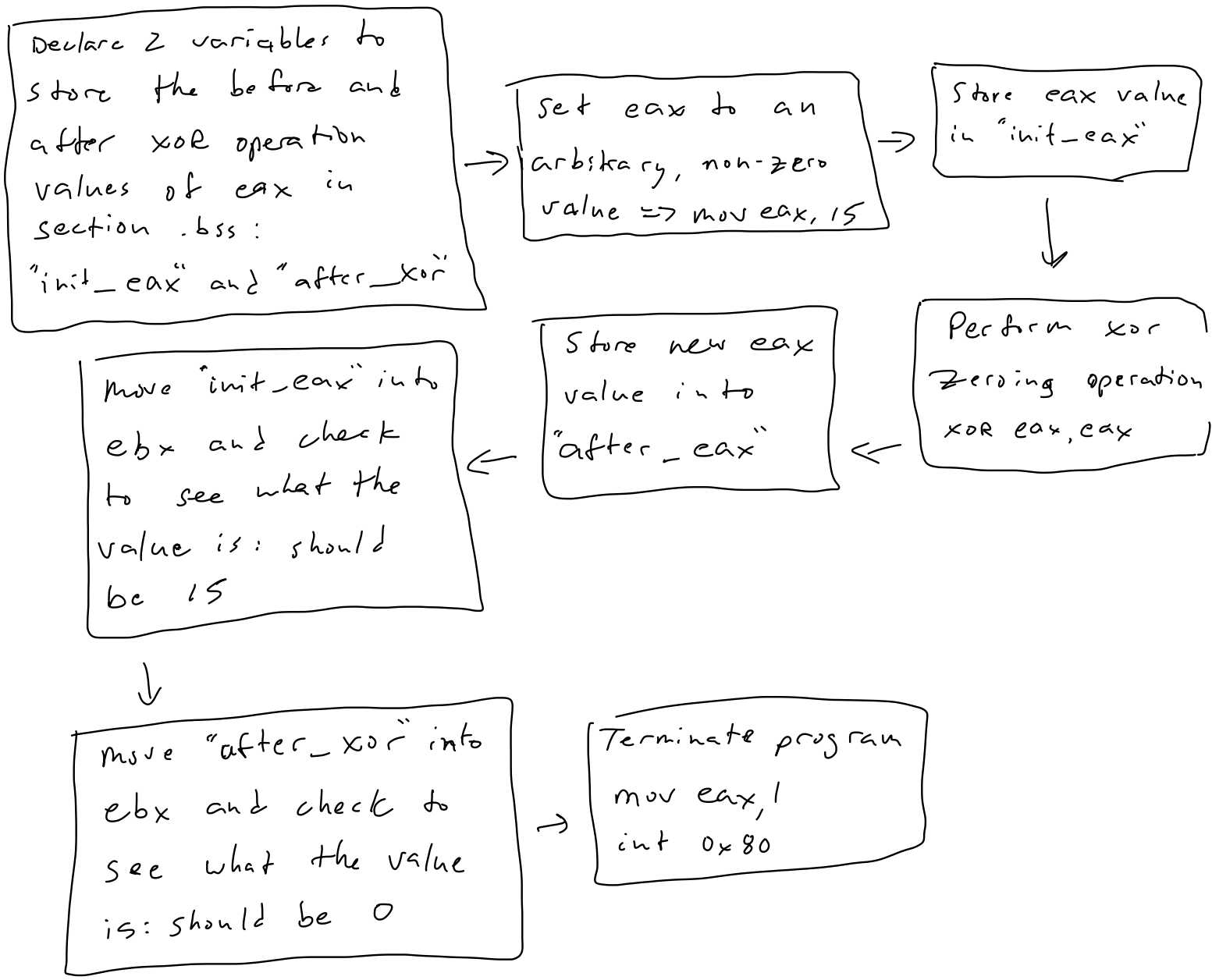
```
mov eax, 11
test eax, 0
```

in GDB, I noticed the only real change was to the flags section in GDB's register watching panel. After some research I found you can use code to trigger off flags; I would use those triggers with a jump to indicate even or odds to the user. Could simulate a 50/50 probability/coin flip.

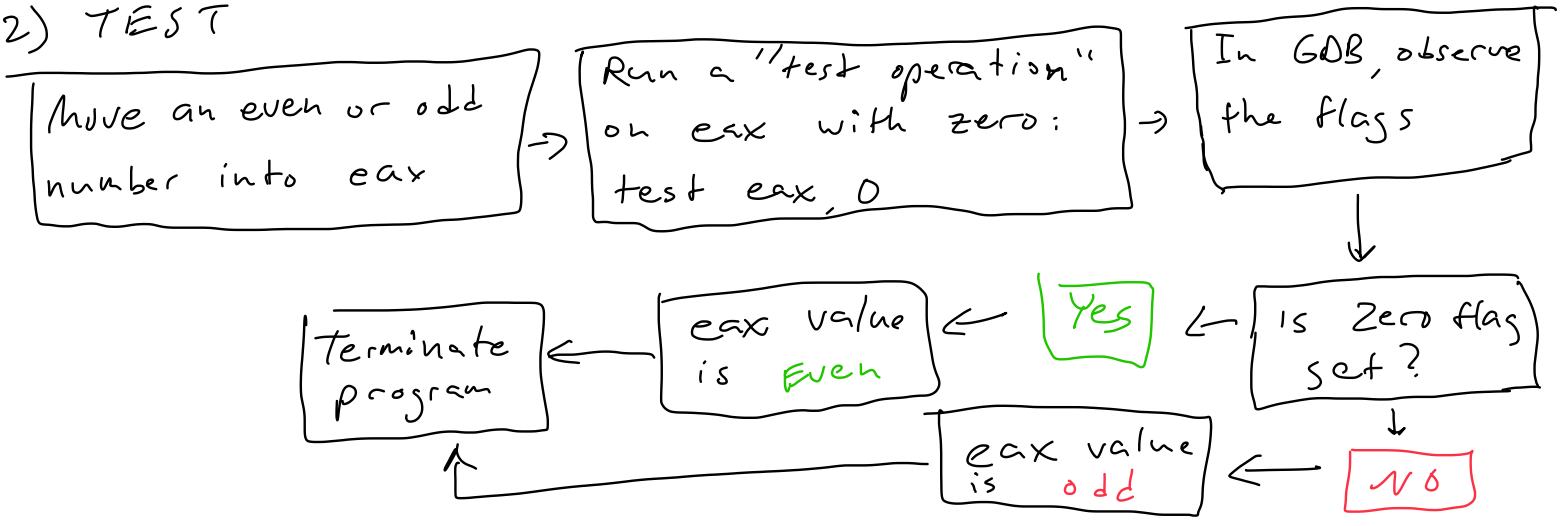
Flow charts

Both of these codes were sequential, with no decision making, resulting in a linear flow chart.

1) XOR Zeroing



2) TEST



Challenges:

The xor zeroing was straight forward and simple as I was exposed to the concept during the w6 activity as a possible solution to a problem I was having with mul/imul. But I was confused as to what "test" was actually doing. I noticed the flags section was changing and looked it up. After some reading on stack overflow I found out that we can interrogate the flag status, in this case the zero flag (ZF), to use "test" in a meaningful way. For the case of determining even/odd, I'd check if the zero flag got set after "test eax, 0". If the ZF is set I would conclude that eax held an even number.