Assignment 3: Recursive Integer Vector Sum

Due: Wednesday 9/30/20 (before class)

On a15, change into a directory you create for this assignment, and issue the following command to get the file you need:

```
cp ~whaley/classes/ARMF20/p3_irsum/tst_irsum.c .
```

Read this file to understand how the tester works. You can compile this along with your own file as needed, for instance:

```
gcc -O3 -marm -o xtst tst_irsum.c irsum.S
```

In this assignment, you will write an ARM assembly divide-by-half recursive vector summation to find the summation of all integer elements in a 1-D array, using the API:

```
int irsum(int N, int *pX);
```

It should return zero if $N \leq 0$, without accessing the pointer.

Note that you must recursively divide N into as close to halves as possible in integers (eg, if you call with N=7, the first step of recursion will call one side with N=3 and other with N=4).

If you are unclear at all on recursion or caller's side of ABI, I suggest you do this in stages, skipping those you think you already know. Note that you should vary optimization, and use valgrind to get confidence you aren't just getting lucky with wrong ABI stuff that works!

Recommended steps:

1. First, write a loop-based vector sum
2. Test your loop-based vector sum using the provided tester, making sure to handle error cases
3. Now change name of loop-based code, and write your assembler routine where all it does is make a single function call your loop-based code
   → This should help clarify the caller's job in the ABI
4. Change your assembly to make two calls to loop-based code, split as if you were doing recursion (but you are calling a different routine twice and summing up the separate calls, so its not yet recursion).
   → This will ensure you understand how to keep things live accross calls, and to do the required arithmetic
5. Write the above recursion in C, and test with the provided tester, making sure you handle error and basis cases, use `ddd` to ensure your recursion is working as expected
   → This makes sure recursion you want to do is concrete in head before writing in assembly
6. Make your assembly routine fully recursive so it calls only itself, while still handling all cases
7. Optimize your code as much as you can for code size

For this assignment, use of CPP names for registers is optional (I don't find it aids readability much when most of code is just following ABI). However, you should continue commenting every single line, with no tabs and no line having more than 80 characters.