Notes on Computer Project #9

Comments about the assignment and responses to frequently asked questions will be added to this file as necessary.

***** comments added on 10/29/18 *****

1) As stated on the assignment handout, you are required to create a makefile which controls the translation of your program, and the name of your executable files must be "proj09".

2) As stated on the assignment handout, your program will be assembled and linked using "gcc". Be sure to use the "-march=native" option if you wish to use "sdiv" instructions.

3) Be sure to use the register conventions described in the project specifications.

4) For your convenience, I have supplied the following file:

```
  /user/cse320/Projects/project09.support.h
```

That file contains declarations of the functions that are required in your support module.

Since that file may not be modified, there is no reason to copy it into your account. Instead, use its absolute pathname in the appropriate preprocessor directive. For example:

```
  #include "/user/cse320/Projects/project09.support.h"
```

Any C source code which invokes those functions should contain the appropriate preprocessor directive.

5) Please note the following statements from the assignment handout:

> Those eight functions (and any associated "helper" functions which you develop) will constitute a module named "proj09.support.s". The functions in that module will not call any C library functions.

You may wish to use function "printf" to display intermediate results as you develop your support module, but be sure to remove those calls (or turn them into comments) before submitting your work.

6) I recommend that you develop your solution in a top-down manner. For example, you might concentrate on developing a simple version of the driver module first, with the other functions being implemented as stubs which return a constant value.

At that point, you can begin incrementally developing the support module (and extending your driver module to test the new capabilities of those functions).

7) Think carefully about the error conditions which can occur. Some errors involve invalid arguments, and some involve invalid results. Regardless of the cause, the functions should return the integer value 0x80000000 when an error is detected. That value is the smallest integer value that can be represented as a 32-bit two's complement value.

8) Overflow is a possibility in most of the functions and should be considered an error condition. To check for overflow with addition and subtraction, you may be able to use the V bit in the NZCV condition code bits.

9) Some additional commentary about multiplication and division: the multiplication instructions (MUL and SMULL) and the division instruction (SDIV) are available on the ARM microprocessors in the CSE Pi array.

The description of MUL in the ARM manual:

> Multiply multiplies two register values. The least significant 32 bits of the result are written to the destination register. These 32 bits do not depend on whether the source register values are considered to be signed values or unsigned values.
>
> Assembler syntax: `mul RD, RN, RM`

The description of SMULL in the ARM manual:

> Signed Multiply Long multiplies two 32-bit signed values to produce a 64-bit result.
>
> Assembler syntax: `smull RDlo, RDhi, RN, RM`

The description of SDIV in the ARM manual:

> Signed Divide divides a 32-bit signed integer register value by a 32-bit signed integer register value, and writes the result to the destination register. The condition flags are not affected.
>
> Assembler syntax: `sdiv RD, RN, RM`

Note that the multiply instructions do not update the V bit of the integer condition code bits (NZCV), but overflow can still occur.

--M. McCullen