Notes on Computer Project #7

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

***** comments added on 10/15/18 *****

1) Please note that you are required to create a makefile which controls the translation of your program, and that the name of your executable file must be "proj07".

2) For your convenience, I have supplied the following interface file, which contains the declaration of function "decode":

  ~cse320/Projects/project07.support.h

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/project07.support.h"
```

Any program which uses that function should contain the appropriate preprocessor directive.

3) Please note that this project focuses on a subset of the ARM instruction set, and that subset only contains 16 instructions.

It does not focus on the entire instruction set for the ARM microprocessor.

4) You might consider using functions from the standard C library to assist you in formatting the character string. For example:

```
    char result[80];

    sprintf( result, "Result: %d (%s)", 16, "flag" );
```

After returning from "sprintf", "result" would contain the character string "Result: 16 (flag)".

Please note that "printf" and "sprintf" are functionally equivalent, with one exception: "printf" builds its character string in a system output buffer which is eventually flushed to the standard output stream, while "sprintf" builds its character string in the array explicitly supplied as its first argument.

5) Function "printf" and its relatives (such as "sprintf") recognize some conversion specifications that may be useful to you. For example, consider the following lines of C source code and the output they produce:

```
  unsigned A = 0xA6;
  printf( "A: %#08x\n", A );         // displays "A: 0x0000a6"
```

To left-justify a character string, use something like "%-10s" when you call "printf" and its relatives.

For more details, see the on-line manual ("man -s 3 printf").

6) The C "string" library may be useful to you. For example, the "strcpy" function can be used to copy C-style character strings (arrays of type "char", terminated by a null byte) and the "strlen" function can be used to determine the length of a C-style character string:

```
  char buffer[80];
  strcpy( buffer, "Hello" );

  printf( "'%s' contains %d characters\n", buffer, strlen( buffer ) );
```

Those lines of C source code produce the following output:

```
  Hello contains 5 characters
```

You may wish to review the source code in:

```
  ~cse320/Projects/project07.sample.c
```

For more details, see the on-line manual ("man -s 3 string").

7) Be sure to test your solution with a variety of different 32-bit ARM instructions. To generate the test cases, you might consider using the "gcc" package to do some of the work of translating the assembly language instructions into machine language.

For example, you could use "objdump" to observe the machine language instructions:

```
  <...> gcc -c example14.s
  <...> objdump -d -Mreg-names-raw example14.o
```

Another approach would be to turn some assembly code into object code, then use "objdump" to match the machine language insructions back to assembly language instructions. Here's a quick example:

```
  <...> cat test.s

        add     r5, r6, r10
        orrs    r7, r8, #0xff

  <...> gcc -c test.s

  <...> objdump -d -Mreg-names-raw test.o

Disassembly of section .text:

00000000 <.text>:
   0:   e086500a        add     r5, r6, r10
   4:   e39870ff        orrs    r7, r8, #255    ; 0xff
```

Notice that you don't have to create complete assembly language programs to use this technique.

--M. McCullen