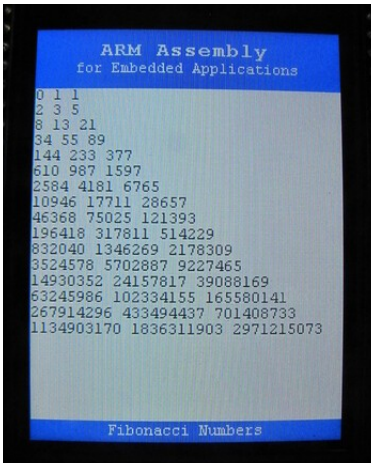
CS2240 Fall 2022
Programming Project #1 – The Fibonacci Sequence
Due: Friday, October 14, 2022, 11:59pm

Hint: Remember the helpful 'SVC #0' debugging instruction available with your Discover Board library! That can be inserted anywhere in your assembly code any time you need to clarify exactly what is in the registers!

First study the basics of the Fibonacci Sequence in the first several paragraphs at https://en.wikipedia.org/wiki/Fibonacci_number. Download the C code file Project1- FibonacciMain.c and the template assembly source file Project1-fibonacci.s. Your project code will resemble the lab exercises with two source code modules, a main program in C and an assembly module. For this project the assembly module will provide two functions called by the C code.

a) Note in the main C program a first call to a function 'init_fibonacci' in the assembly language module to initialize the state variables required by your assembly language code. Then the main program iteratively calls a second function 'fibonacci' written so that the sequence of unsigned integers returned from each function call are the integers representing the Fibonacci sequence. That is, the first two calls to 'fibonacci' should return the first two fixed initialization numbers that begin the Fibonacci sequence, namely 0 and 1. Then subsequent calls to 'fibonccci' should return the sum of the previous two return values. However, your assembly code must also monitor the result flags from the addition operation, and if the *unsigned* arithmetic overflows, the function should return an error code of 0xFFFFFFFF instead of the (meaningless) arithmetic result. The loop in the main C program tests all the function return values for this special value and terminates the program, as the sequence numbers have grown too large to be correctly handled with 32-bit arithmetic. But before this overflow indication is found, the C program uses the printf() formatted print function to display the returned Fibonacci number with an *unsigned* '%lu' format conversion, three numbers per line, separated by a space character, on the the Discovery Board display.



b) Now write the assembly directives and code in the assembly language module to both allocate variable words in the .bss linker section needed for the state variables, and the instructions needed to implement your two functions. See the example declarations and functions on slide 27 of the Memory Access class slides for an example of how a state can be preserved and processed by assembly code. (Hint: I used three variable words in RAM in my solution to this project to allow the assembly function to preserve its state between successive calls. The first word simply holds one of three state values, 0, 1, or 2, to keep track of whether the function expects either its first call, its second call, or all subsequent calls. Then the other two word variables hold the previous two return values from prior calls to be summed in the next call if the state is 2. But your solution may use a different number of variables or some other scheme to preserve the state between calls.) You'll need some conditional code in your function to check the state each time it is called and decide on the appropriate action and return value for each possible state. Also review slide 16 of the Conditional Execution class slides for testing for arithmetic overflow.

c) Run your code and verify that it produces a display similar to that shown above. Upload your code to Canvas and a screen photo. Make sure, however, that any debugging breakpoints you inserted for debugging your code have been removed in the version you submit.