ECSE 324 LAB 1 REPORT

RAFID SAIF

Part 1a: Fibonacci Sequence Using Iteration

Description:

The task for this part was to find the nth Fibonacci number using an iterative approach.

Approach Taken:

We store the first two Fibonacci numbers, F(0) and F(1) in registers R1 and R2. If n is equal to either of these, we simply store the corresponding value into the memory space and exit the loop. If n is greater than these, we store the value of R2 in a temporary register. R2 is then given the value of the sum of R1 and R2, and the value of R2 stored in the temporary register moved into R1. This is one iteration. We complete n-2 iterations, and the final value of R2 is then stored in memory.

Challenges Faced:

None for this part.

Part 2: 2D Convolution

Description:

The task for part 2 was to translate the 2d convolution algorithm given in C into ARM. We start with given arrays fx, gx and kx.

Approach taken:

We start by first loading the constants iw, ih, kw, kh, ksw, and ksh into registers. As some of these share the same value, we store them in one register to save registers. We then initialize counters for our iteration variables x, y, i, and j. We then proceed to create our nested loops. At the beginning of each loop, we check if the counter for that loop has reached its end value. For example, we check if the x-counter is less than iw. If it is, we move into the next loop or if we are in the innermost loop, we complete all the instructions there. If our counter is greater than our maximum value, we first complete any instructions that must be executed within that loop. We then set the counter for that loop to zero, and increment the counter for the loop immediately outside it. We then branch to that outer loop.

At a certain point, we are required to access the memory for a certain element in our arrays. In C, this is done by specifying values in the format “arr[a][b]” where a refers to the row and b refers to the column. As we cannot create 2D arrays in ARM, we must decompose the 2D array into a single array in one dimension. To perform this transformation, we observe that the object at row a and column b would be equal the nth element in a 1D array, where

$$n = a \times numberOfColumns + b$$

We then use this value of n as the offset from the first element of the array, and load or store values as required.

Challenges Faced:

The first challenge faced was to translate the 2D array into a 1D array, the solution to which is provided in the last section. Another was the limited number of registers, as there were several values that needed to be stored. To overcome this, we reused registers and use one register to store constants with the same value.

Possible Improvements:

We could avoid the hassle of storing 4 into a register (used to calculate array offset) and instead use a logical left shift.

Part 3: Bubble Sort

Description:

The task for the last part was to implement the bubble sort algorithm provided in C.

Approach Taken:

We start by loading the address of the first array element into register R0. We then initialize counters for our loops. We start our outer loop, and if the conditions are met to enter our inner loop, we begin by computing the value of size-step-1 to use as the endpoint in our inner loop. We then go through the inner loop, and for each iteration we load two adjacent array elements into registers. We compare, and if the first element is greater than the second, we branch into a label called swap. In swap, we simply swap the two array elements using an intermediate and then branch back to our inner loop. When we complete our inner loop, we set our inner loop counter to zero and increment our outer loop counter before branching back to the outer loop counter.

Challenges Faced:

None. Directly translated from C code.

Possible Improvements:

We used a register to store the value of 4 to calculate offset of an element from the first array element. This could have been avoided using logical shift left.