Jay Chen

CS 2640

Project 2

Iterative Fibonacci in MIPS

The goal of this project was to program an iterative version of the Fibonacci sequence in MIPS assembly language. The code accepts a user input for the n value, input validates this n value, then calculates the Fibonacci sequence up to the nth value and stores all the results sequentially in an array.

For input validation, we first had to find the maximum value of n the program can accept. The range of a 32-bit unsigned integer in MIPS is $(2^{32} - 1)$ or (0 - 4294967295). Using the following list of Fibonacci numbers up to n=300 published online by Dr. Ron Knott of the University of Surrey, (http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibtable.html), I was able to calculate the last n value that would fall within the range of unsigned integers. This would be n=47, or F(n) = 2971215073. This can be confirmed by the fact that the next Fibonacci number in the sequence is greater than the maximum range of 32-bit unsigned integers calculated before, as F(48) = 4807526976 > 4294967295. Therefore the program contains in the data section a MAX_INT integer value for the number 47, the largest possible n value the program can accept. The lower bound for input validation is 0, as the Fibonacci sequence starts at positive integers. The program prompts the user for an unsigned integer value within the range [0-47], and prints an error message and asks for input again if the value entered is less than 0 or greater than 47. The size of the array also takes into account this max value of n, as the array is 192 bits = 4*48, for the possibility of the max array size containing a word space for n values (0-47).

For the iterative Fibonacci sequence calculation, the easiest way to ensure that the calculations were only conducted up to n=the user input value was to copy the user input value into a temporary register and use that variable as a counter for each iteration of the loop, decrementing down to 0 to make use of comparison to the $zero register. Due to the fact that the calculations were supposed to be done up to and including n, the actual number of Fibonacci values printed is always n+1 to account for F(0) = 0 at the start, such that for n = 10 the Fibonacci series list will actually contain 11 numbers starting at 0. The program sets F(0) = 0 and checks if n = 0 before the loop runs, because if so there is no calculation to be done and a branch statement skips straight to the output loop. Otherwise the program iteratively calculates the next Fibonacci number using the pseudocode given.

The pseudocode essentially updates the current value of the array to the previously saved value of the Fibonacci sequence, then adds the previous value with the current value that was stored. So for example, n = 3, where previously F(2) = 1 and F(1) = 1, therefore F(3) = F(2) + F(1) = 2. The array address value stored in register $t3 is incremented by a word space of 4 each time so that its 0 offset value is able to be easily accessed. One weakness of following the pseudocode exactly is the requirement of 5 different registers to solve the problem (or 6 if counting the fact that the user input was copied to another register to avoid erasing the number entirely). This is due to the need for a temporary register to store the a and b values as they are incremented and

added to one another. In this sense, the recursive Fibonacci implementation, which I tried programming as a test, actually uses less registers because the arithmetic operations can be conducted directly on the values stored in the array at previous addresses (n-1) and (n-2) rather than with iterative registers.

For printing the resulting unsigned integer, a different syscall code had to be utilized from the syscall code of 1 used to print normal integers. Due to the fact that unsigned integers are much larger than the signed int size, if the regular syscall code was used to print integer 47 a negative number was output to the console instead due to integer overflow, as the Fibonacci number for F(47) = 2971215073 is greater than the maximum range for signed 32-bit integer values: (-2^{32-1} to $2^{32-1}-1$), or -2147483648 to 2147483647. To fix this issue, the unsigned int syscall code 36 is used to print the list of Fibonacci values instead.