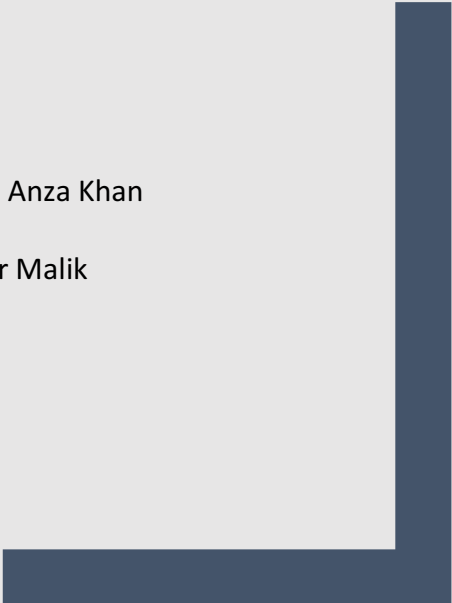




LAB 2

Muhammad Anza Khan
260618490
Ayden Aamir Malik
260627064



Stack

This code is a simple implementation of the push and pop pseudo instructions using ARM ISA.

We subtract the stack pointer by 4 and store 10 onto the stack.

Popping in this context means, we first load the contents of the stack onto a register, say R0 and then we decrease the size of the stack by 4 and then load the next value onto the next register.

Max using subroutines

START points to the location of the characteristics of the list in memory.

LOOP finds the maximum of the list.

The code begins with R4 being loaded with the address of the RESULT label.

R2 holds the size of the list.

R3 points to the first number of the list.

R0 holds the first number of the list.

We call our subroutine LOOP and store the address of the first instruction in the link register.

The loop begins with 1 being subtracted from the size of the list stored in R2, using the SUBS function. R2 acts as our loop counter.

R3 now points to the next element in the list and R1 holds that element. Using the compare function, we compare the second element with the first element, if it is greater than the one in R0, we swap it using the MOV function, otherwise we access our return address using the BX function in the link register i.e. the address of the first instruction of the loop function. This process shall continue to run until the loop counter

hits 0 and we exit the subroutine using the branch & exchange instruction going to the address in the link register which is the address of the end instruction.

Fibonacci

We start our code by initializing R0 with the number 11 which is the number for which we need our Fibonacci sum.

We branch down to the Fibonacci subroutine.

We push the contents of R1, R2 and the link register onto the stack because as we run each pass on the Fibonacci subroutine we know the contents of these registers will change thus we save contents of each register on the stack.

The next 4 instructions check the value of R0 to see if it is equal to 2 we move 1 into R0 and exit the subroutine. (exit case)

We push the values of R1, R2 and LR on the stack until we get to the exit case.

We make sure we hit iteration of n-2, after that we begin iterations for n-1. Then we begin popping the values from the stack and adding them.

Question 2 part 1

In this code we simply sort the array using bubble sort and return the last value, which is the largest value.

We calculate the size of the array using a macro defined function called `ARRAY_SIZE(X)`. The code starts by calculating the size of the array using the number of bits occupied by the array and dividing it by the number of bits of the first element.

In the main function the array is defined. The size is calculated by `ARRAY_SIZE(X)` and stored in an integer called `size`.

The maximum value of the array is calculated by calling the `maxValOfArray` function.

`maxValOfArray` accepts 2 parameters and uses bubble sort to sort the array and returns the last value of the array which is the largest element of the array.

Question 2.2

We calculate the size of the array using a macro defined function called `ARRAY_SIZE(X)`.

The code starts by calculating the size of the array using the number of bits occupied by the array and dividing it by the number of bits of the first element.

The main body of the code begins initializing three integer variables `a`, `b` and `c`. An array of five elements is declared.

A loop iterates through the list and calls upon the `MAX_2` label from the assembly program already provided.

The code provided holds on to a number and compares each element of the array to the it. If it finds one larger than it, the `MAX_2` replaces it with the largest.

`Temp` holds the largest number
`C` holds the size of the array.

It is printed at the end.