Arrays:

How to use Arrays in Assembly -----------------------------------

Arrays:

Lowercase v has to be used in front of all operations because v stand for vector. A vector is like a pointer, so you have to use this special syntax when working with arrays from C++ to Assembly.

The operations are the same, but a v had to be put in front of everything. For example ldr would become vldr.

Here are all the standard ARM commands you need to know: <https://docs.google.com/document/d/1pPmu2TG8pLwFLbxz8q_Nk3vSzAvG4NHwFrpHpa3nu5k/edit?usp=sharing>

Note: For double you have to put .f64 in front of numerical operations (addition, subtraction, and multiplication). For example, vadd.f64 will only accept and return doubles. Also, for double you have to use d registers instead of r registers. For example, you would vpush {d1,d2,d3}.

In order to access different parts of an array, you must move the pointer over by a set number of bits. For example, vldr d1[r0] will return the first number in an array, the number at 0 bits. In the example shown below, this would return 1.0 since it’s the first number in the array. In order to get the second number in this array, you would need to do vldr d1[r0,#8] , since it moves the pointer over by 8 bits.

For a double, the pointer needs to be moved by 8 bits each time.

[r0] = first number , [r0,#8] = second number , [r0,#16] = third number …

For a float, the pointer needs to be moved by 4 bits each time.

[r0] = first number , [r0,#4] = second number , [r0,#8] = third number ...

Example:

Given that double a[] = {1.0, 2.0, 3.0}; double b[] = {2.5, 3.2,-1.0};

vpush {d1,d2,d3}

vldr d1, [r0]

vldr d2, [r1]

vmul.f64 d0, d1, d2

vldr d1, [r0, #8]

vldr d2, [r1, #8]

vmul.f64 d3, d1, d2

vadd.f64 d0,d0,d3

vldr d1, [r0, #16]

vldr d2, [r1, #16]

vmul.f64 d3, d1, d2

vadd.f64 d0,d0,d3

vpop {d1,d2,d3}

bx lr

Example of how to add the sum of thing from a array:

R0 points to {3,4,1,2}

R1 is amount of numbers in array

R0 should be the sum of numbers in the array

x = {3,4,1,2}

MOV R2,R0 @ R2 points to the x

MOV R0, #0

.loop:

LDR R3, [R2] @ R1 = current element in Array

ADD R0, R0, R3 @R0=R0+R3

ADD R2,R2,#4 @advance to the next thing in the array

SUB R1,R1,#1 @counts down value of things in array

BNT .loop

bx lr

If this still does not make any sense here is a link: <https://docs.google.com/document/d/19GPmWg8UVCU-5QYBC3V4vQNqUkFoXwK2YJCdejMjenw/edit?usp=sharing>