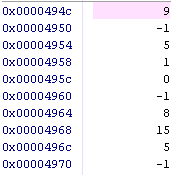
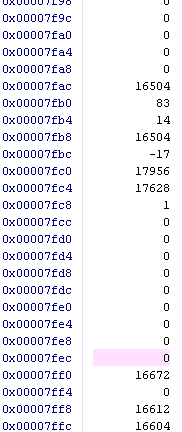
6. Memory Tab Showing Search Results



7. First 25 Words in Stack Memory



The stack contains ra (the return address), r16 (the middle index value), and r17 (the middle index value). The reason why the stack contains these values is because main.c assumes r16 and r17 to be unchanged when calling the BinarySearch function, which means this function is responsible for storing (pushing to the stack at the beginning of the function) and restoring (popping from the stack at the end). The reason ra is kept in the stack is because whenever the function recursively calls itself, it needs to remember where to return the execution to.

Answers:

* Numbers that are at the extremes of being very large or very small relative to the list of numbers causes more recursive calls.
* Binary search often yields better performance than linear search since binary search at most takes Log(n) time, since the code divides the search in half iteratively until the value is found. This is faster than linear especially when the value is at the end of the list, since linear in that case takes almost n operations instead of Log(n)
* The Complexity of the binary search is O(Log(n)), since it splits the problem in half until the number is either found or determined to not exist in the sorted list.

Program

.global BinarySearch

BinarySearch:

/\*Push to stack\*/

subi sp, sp, 12

stw ra, 0(sp)

stw r16, 4(sp)

stw r17, 8(sp)

/\*a. find middle index (r17) 0000494c\*/

sub r17, r7, r6 /\*set r17 to difference of r7 and r6\*/

srli r17, r17, 1 /\*divide r17 by 2 to get middle index offset from r6 start\*/

add r17, r17, r6 /\*add to start index (r6)\*/

/\*b. find middle index address (r16) \*/

muli r16, r17, 0x4 /\*set r16 to byte address OFFSET by multiplying the index by 4 bytes\*/

add r16, r16, r4 /\*add array pointer (r8) address to OFFSET (r16)\*/

/\*c. find middle index value (r16) \*/

ldw r16, 0(r16) /\*load number at that address (r16)\*/

/\*branch if equal\*/

beq r16, r5, Case\_EQUAL /\*if middle index value (r16) is equal to r5, go to Case\_A\*/

beq r6, r7, Case\_Not\_Found:

/\*if middle is not same as searched, continue to check if greater than or less than\*/

/\*if middle index value (r16) is less than searched value (r5), search right\*/

bgt r5, r16, Skip\_LESS:

/\*IF Searched < Middle Value\*/

Case\_LESS:

subi r7, r17, 0x1 /\*subtract 1 from middle index and store in end index (r7)\*/

call BinarySearch /\*call BinarySearch with end index being one less than middle index\*/

br Start\_Return

Skip\_LESS:

/\*IF Searched > Middle Value\*/

Case\_MORE:

addi r6, r17, 0x1 /\*set startIndex to middle index plus 1\*/

call BinarySearch /\*Call BinarySearch with startIndex at middle index plus 1\*/

br Start\_Return

Skip\_MORE:

/\*IF Middle Value == Searched\*/

Case\_EQUAL:

add r2, r0, r17 /\*set return value to r17 (middle index)\*/

br Start\_Return

Case\_Not\_Found:

addi r2, r0, -1

Start\_Return:

ldw ra, 0(sp)

ldw r16, 4(sp)

ldw r17, 8(sp)

addi sp, sp, 12

ret

/\*return\*/