LC-3 addressing modes compared to C++ pointers

.orig x3000

br main   ; don't try to execute the data

my\_string: .stringz "testing!"

;....

main:

lea r0, my\_string

trap x22

will display "testing!" (w/o the quotes)

Since we start at x3000, the instruction 'br main' takes up the memory at address x3000. The string starts at x3001, and it's just like c-strings in C++:

the 't' is @ x3001, 'e' @ x3002, 's' @ x3003, etc. etc.

So when you say 'lea r0, my\_string', it means "find the address of my\_string and put that in r0".

After the lea executes, r0 will have a value of x3001.

This is exactly the same as a pointer.

Roughly equivalent C++ code:

const char \* my\_string = "testing!";

//my\_string is a pointer -- its value is the address of the first character in the string

std::cout << my\_string;

LEA in the LC-3 ISA is much akin to the address-of operator in C++ (&), used in the following way:

int my\_number = 45;    //my\_number: .fill 45

int \* r0 = &my\_number; //lea r0, my\_number -- implies r0 is now a pointer

LDI is a method of de-referencing a pointer.

.orig x3000

br main

some\_number: .fill 33

;....

pointer\_to\_some\_number: .fill x3001

;....

main:

ldi r1, pointer\_to\_some\_number

;r1 now contains the value 33

this is equivalent to:

const int some\_number = 33;

const int \*some\_ptr = &some\_number;

//storing 'address of some\_number' - just like storing x3001 in the asm code above

//...

int r1 = \*some\_ptr;  //here's the LDI

LDR is another method of de-referencing a pointer, but this one expects a register and offset from that register as a pointer, not a value at a label as a pointer:

.orig x3000

br main

my\_string: .stringz "testing!"

;...

lea r3, my\_string   ; put the beginning address of my\_string in r3, i.e. r3 is now a pointer

ldr r0, r3, 0   ; r0 = \*(r3 + 0) i.e. r0 now has a value of 't'

trap x21

ldr r0, r3, 1   ; r0 = \*(r3 + 1) i.e. r0 now has a value of 'e'

trap x21

this will display "te" (w/o the quotes)

it's equivalent to

const char \*my\_string = "testing!";

const char \*r3 = my\_string;

//no address-of operator because of semantics; we specified my\_string is a pointer type

char r0;

r0 = r3[0];

std::cout << r0;

r0 = r3[1];

std::cout << r0;