Now let's turn our attention to the second class of instructions: load (LD) and store (ST), which allow the CPU to access values in memory.

Note that since the Beta is a load-store architecture these instructions are the *only* mechanism for accessing memory values.

The LD and ST instructions use the same instruction template as the ALU-with-constant instructions.

To access memory, we'll need a memory address, which is computed by adding the value of the "ra" register to the sign-extended 16-bit constant from the low-order 16 bits of the instruction.

This computation is exactly the one performed by the ADDC instruction - so we'll reuse that hardware - and the sum is sent to main memory as the byte address of the location to be accessed.

For the LD instruction, the data returned by main memory is written to the "rc" register.

The store instruction (ST) performs the same address calculation as LD, then reads the data value from the "rc" register and sends both to main memory.

The ST instruction is special in several ways: it's the only instruction that needs to read the value of the "rc" register, so we'll need to adjust the datapath hardware slightly to accommodate that need.

And since "rc" is serving as a source operand, it appears as the first operand in the symbolic form of the instruction, followed by "const" and "ra" which are specifying the destination address.

ST is the only instruction that does *not* write a result into the register file at end of the instruction.

Here's the example we saw earlier, where we needed to load the value of the variable x from memory, multiply it by 37 and write the result back to the memory location that holds the value of the variable y.

Now that we have actual Beta instructions, we've expressed the computation as a sequence of three instructions.

To access the value of variable x, the LD instruction adds the contents of R31 to the constant 0x1008, which sums to 0x1008, the address we need to access.

The ST instruction specifies a similar address calculation to write into the location for the variable y.

The address calculation performed by LD and ST works well when the locations we need to access have addresses that fit into the 16-bit constant field.

What happens when we need to access locations at addresses higher than 0x7FFF?

Then we need to treat those addresses as we would any large constant, and store those large addresses in main memory so they can be loaded into a register to be used by LD and ST.

Okay, but what if the number of large constants we need to store is greater than will fit in low memory, i.e., the addresses we can access directly?

To solve this problem, the Beta includes a "load relative" (LDR) instruction, which we'll see in the lecture on the Beta implementation.