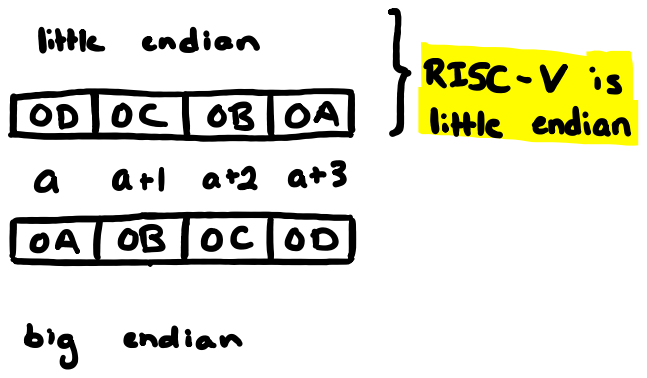
How Memory Works In Project 3

Wednesday, October 21, 2020 6:00 PM

① Endianness

Consider some value 0x0A0B0C0D stored at address a...
There are two ways to store this 4 byte value



② Addressing

Consider some address 0x AABBCCDD that we wish to access in memory. The memory unit will only fetch the top 30 bits of this address. This means that to memory our effective address is the 30 bit value 0x2AAEF337.

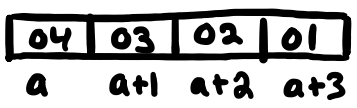
Note that memory always returns the 4 byte word from a referenced effective address.

③ Write Masks

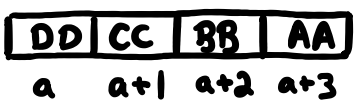
The provided memory is byte level write enabled. This means that our write enable is a 4-bit signal where each bit corresponds to one byte in the addressed word.

Say we have some address a which contains 0x01020304 and we wish to write some new value 0xAABBCCDD. The following examples show the effects of some different write enables

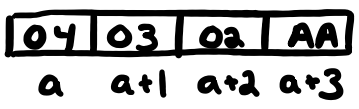
Original:



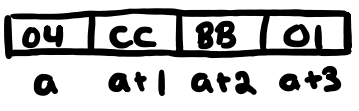
Wen = 4'b1111:



Wen = 4'b1000:



Wen = 4'b0110:



This may seem confusing at first glance, but can be simplified to: if bit i of Wen is 1, write byte i of Din into the ith byte of the word in memory at address a. Remember that regardless of your original address, the effective address will be word aligned!

④ Valid Loads/Stores

The rule for loads and stores is as follows:
Any request where the requested section does not break the boundary of a contiguous 4-byte word in memory is valid.

But what does that mean? Consider some address a, and some register S0 containing a. We can represent bytes in memory as boxes, and requests as overlays. We draw a red line to mark the boundary between contiguous words. Any request where the overlay box does not break the red line is valid.

Below are some examples of this, you are encouraged to build on these examples to determine what is valid for each load/store type.