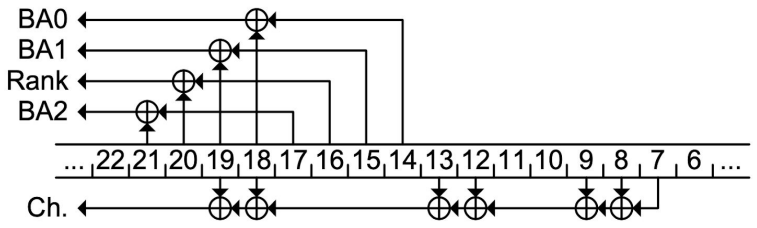
Lab 1 - Clarifications

Since we saw a couple of recurring mistakes in the deliverable we wanted to clarify some of these misunderstandings.

XORing functions

A DRAM address is a hextuple composed of `<channel, DIMM, rank, bank, row, col>`. The memory controller needs to be able to transform the physical address it receives from the CPU cores into such hextuple. The XORing functions we keep on referring to are used to generate the `<channel, DIMM, rank, bank>` sub-tuple. To be completely correct we should look at this whole conversion between physical address → `<channel, DIMM, rank, bank, row, col>` as a single function. We talk about multiple XORing functions because the memory controller applies different *hashing* functions to generate the single addressing bits.



Intel Haswell (desktop system) – DDR3

Let's resurrect the diagram from the DRAMA paper mentioned in the manual.
As you can see here bits `14^18` generates a single bank bit (as it goes for `15^19`, etc.). How many addressing bits do you need? It depends on the memory configuration.
So what are these XORing functions of Task 2 again? Every single bit to identify to which bank the current rows map to.

Let me explain this with an example. Let's pick the HWSec02 node where we have the following configuration <2,2,1,16>. We have 2 channels so we need 1 bit to pick one of the two channels (either ch0 or ch1), once you pick one of the two channels you only have one DIMM so you don't need to address this but you need to pick one of the two ranks inside the dimm (one more addressing bit). And finally you have 16 banks inside the rank. So you need log2(16) = 4 addressing bits for the bank.
In total you need 1ch bit, 1rk bit, 4bk bits = 6 addressing bits; i.e., **6 XORing functions**.

So now I hope this concept is a bit clearer and it will become easier to understand the few common mistakes we saw.
It was not clear to everyone what these bitmasks (or *functions*) you were generating with the `next_bit_perm()` function were and why we were testing only between 1 and 6 bits set. So let's have a look at the figure above once again. As you can see the hashing function that is used to choose the *channel* XORs together 6 bits. These 6 bits will give you an output which is either 0 or 1; i.e., channel 0 or channel 1. Intel systems do not seem to use more than 6 bits for addressing. Hence the upper bound we recommended in the manual.
Furthermore, when talking about XORing functions as `base ⊕ fn`, we did not mean `base ^ fn` . And by now it should be clear why doing the latter doesn't work.

But let me explain this with another example. Let's take the function that picks the channel in the HWSec02 node (`0x4b300`) and a random address (`0x7fac78758780`).
Now if I do the following `addr ^ fn ==> 0x7fac78758780 ^ 0x4b300` the results will be this one `0x7fac78713480` . This is definitely not a single bit of addressing to pick channel 0 or 1. What you wanted to do instead was computing the XOR of the bits set in the `0x4b300` bitmask (`0b10010110011000000000`). So you wanted to XOR bits {8,9,12,13,15,18} of the given address (`0x7fac78713480`) to figure out to which channel it maps to. XORing multiple bits together is equal to doing the following `parity(addr & fn)`. In other words, applying the bitmask to the address and then checking if the number of bits set is even or odd.

Hope this clarifies most of the doubts.

Pietro
On behalf of the HS team