Embedded Systems Programming **Michaelmas 2014**

What have I learnt?

Student: Ankush Gupta

I have taken a similar course in machine architecture (CS61C) at UC Berkeley. Hence, most of the ideas introduced in this course were familiar to me. However, following is a list of things that I did not know before (or had forgotten, same thing) taking this course:

1. I/O pins are expensive, hence, time multiplexing/ giving specific address ranges to memory modules is used to interface with the memory.
2. CAS latency being an important parameter for RAMs : CAS latency is the delay b/w the time a specific memory column is accessed and the time when the first data-bit(s) is(are) received. Your slide on RAM timings has made me a more informed buyer of RAMs.
3. Collisions in cache are extremely bad – which reminds me of this Jeff Dean slide 13 (also see this : this cool visualisation).
4. Cache sizes (especially L1 sizes) have remained fairly constant over the years.
5. Floating point constants in C are by default taken to be `double`, any one needs to append an `f` if they require a `float`.
6. `_Bool` is equivalent to `signed char` in x86 and to `unsigned char` in ARM.
7. `size_t` could be 64-bits but `int` could be actually smaller (32-bits). Hence, assigning `size_t` to `int` could lead to overflow.
8. Namespace of `enum`s and variables is different. Hence, enums and variables with the same name can co-exist — so confusing.
9. `volatile` type qualifier : I thought that it was to protect against compiler optimisation of loads and stores in a multi-threaded scenario. Upon further reading I found that its real use is when interfacing with memory-mapped devices and that it actually does not work for threaded scenarios.
10. **Only** the statically stored variables are zero initialised and not all the variables.
11. Arithmetic promotion is done *always* — even when the types of the variables involved are the same (e.g.: two `char`s are promoted to `int`s before they are added etc.).
12. Realized why "evaluation is done once" in expressions with assignments is important. For example in `arr[i++] += 1`
13. Good to be reminded that order of evaluation of functions is not defined. e.g.: `f() + g()*h()`
14. Arrays and pointers are actually different — because `sizeof` does the right thing for an array but not for a pointer.

15. It is **always** a good idea to declare `virtual` destructors in C++ because of polymorphism — otherwise, we might be left with dangling/ zombie objects.
16. I liked your example of not storing a reference to an element in a `vector` because dynamic re-allocation can make it point to garbage.
17. LabView can actually be fun.

Thank you!