- **x86 data types**: Integer (signed/unsigned/pointer), Floating point, arrays and structs are handled one element at a time.

- **x86 instruction types**
  - *load and store* move data back and forth to memory and registers
  - *math* performs basic add/subtract/shift ops on data in registers
  - *control* jumps into and out of functions, loops, etc

- **Registers** are on-chip data storage for instructions
  - 32 bit versions: eax, ecx, edx, ebx, esi, edi, esp, ebp
    - can get at a two-byte register that only uses the lower half or a single-byte register (legacy)
  - 64 bit versions are rd, rcx, ... etc (plus an additional 8)
    - can still get at the 32-bit versions, only uses half of register

- **Move instructions** `movl src, dest` (moves 4 bytes) `movq src, dest` (moves 8 bytes)
  - brings data in from memory and stores it in registers (to later be used) or stores from registers into memory (when computation is done)
  - may move *immediate* ($1, $0x43) *registers* (%eax, %edx), or *memory references*

- **Memory Addressing Modes**
  - D(Rb, Ri, S)  =>  Load Mem[Reg[Rb] + S*Reg[Ri] + D]
  - Rb is a base register (e.g. address of array)
  - Ri is an index registers (e.g. which element of the array)
  - S is a scale factor (e.g. size of element in array)
  - D is a displacement factor (size to add)
    - e.g. (%edx, %ecx, 4)  =>  0xf000 + 4*0x100
    - e.g. 0x80(, %edx, 2)  =>  0 + 2*0xf000 + 0x80

- **Load Effective Address** `leal src, dest`
  - does the address computation for src and stores the address (without fetching any values) into the destination registers
  - Approximately equivalent to something like &x[i]
  - Can be overloaded to compute alternate math, but we don’t usually do this (why?)