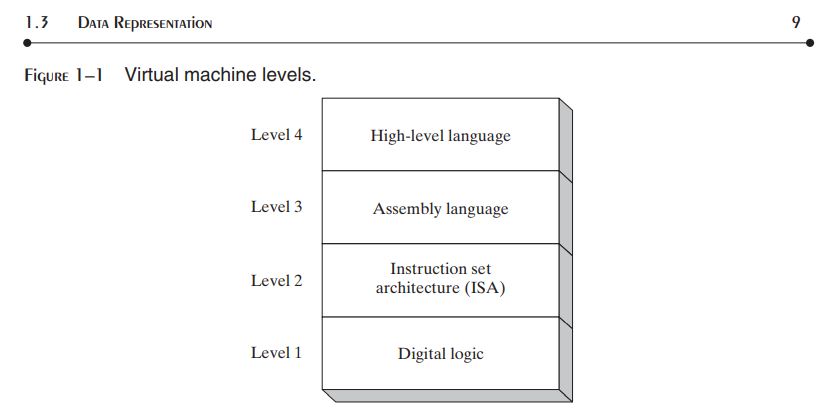
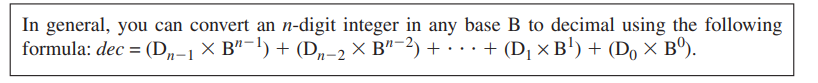
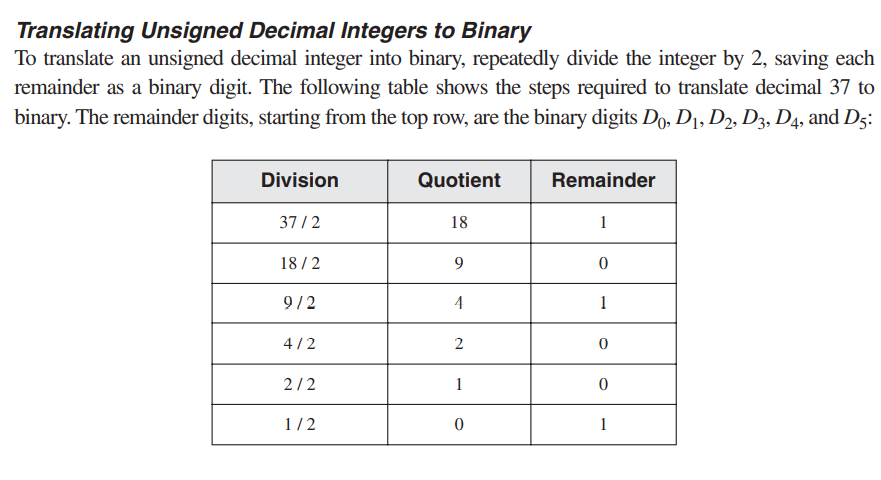
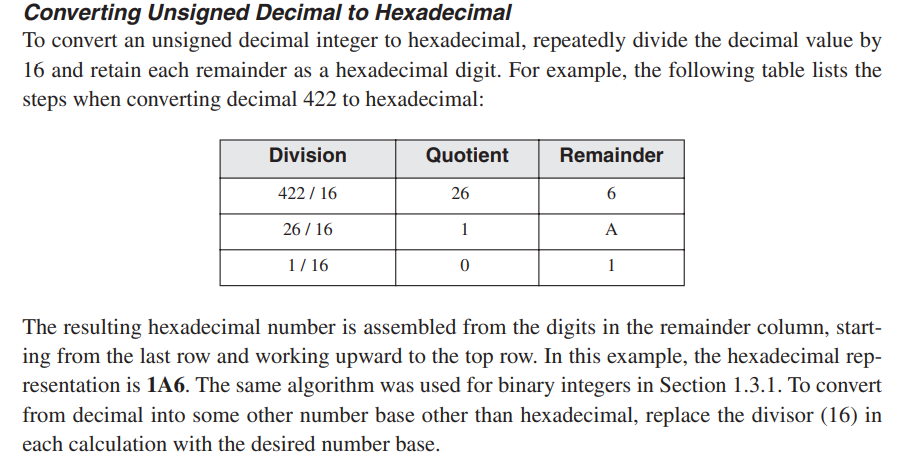
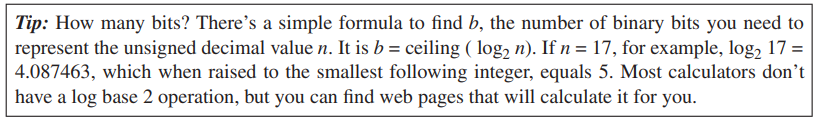
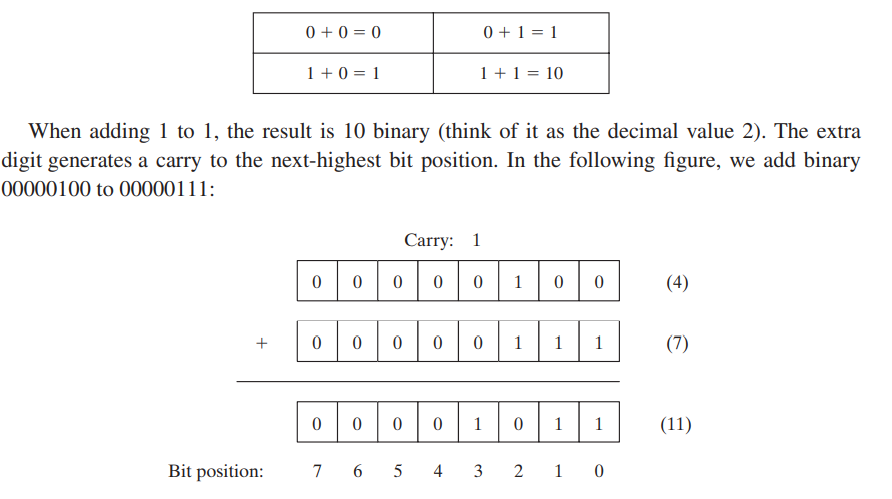
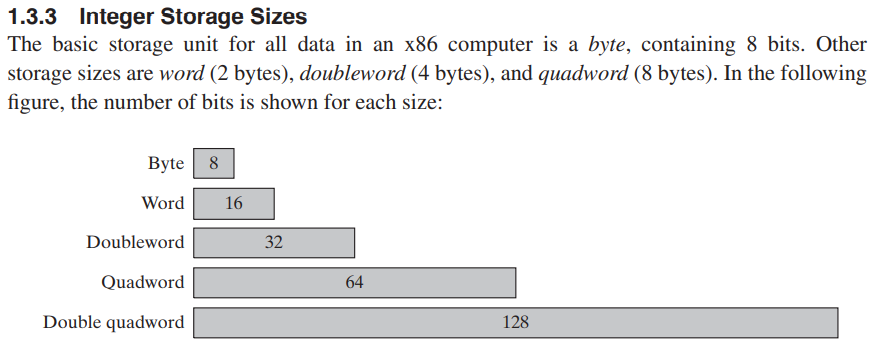
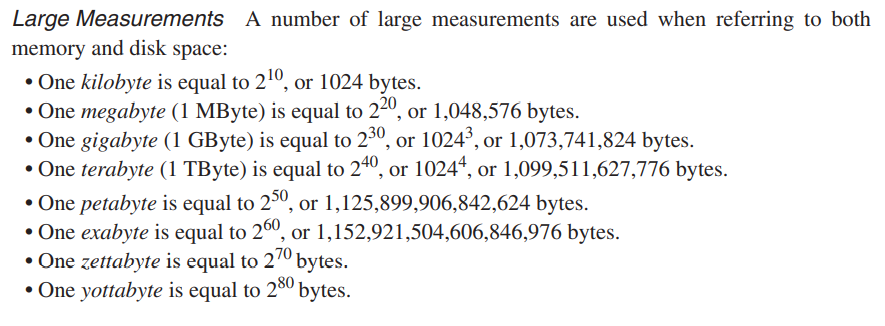
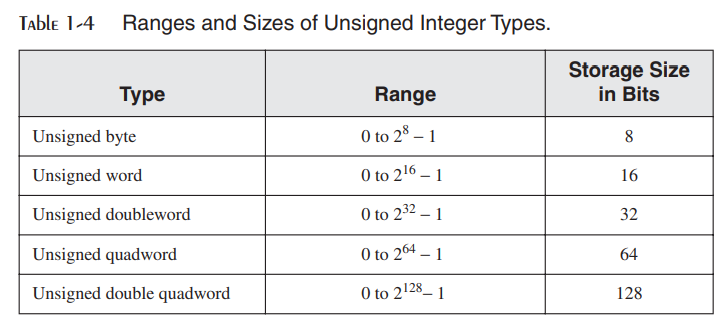
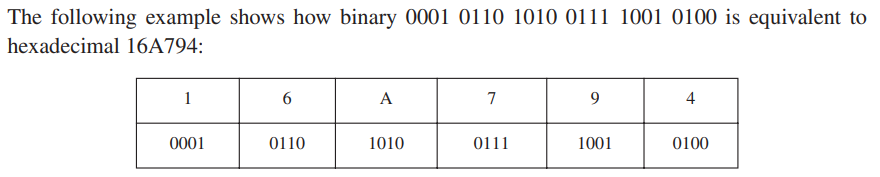
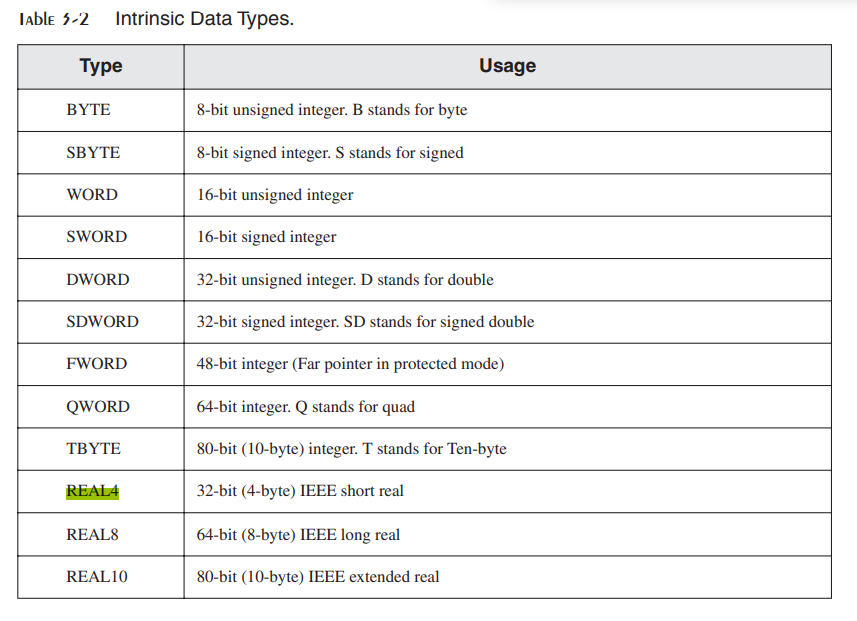
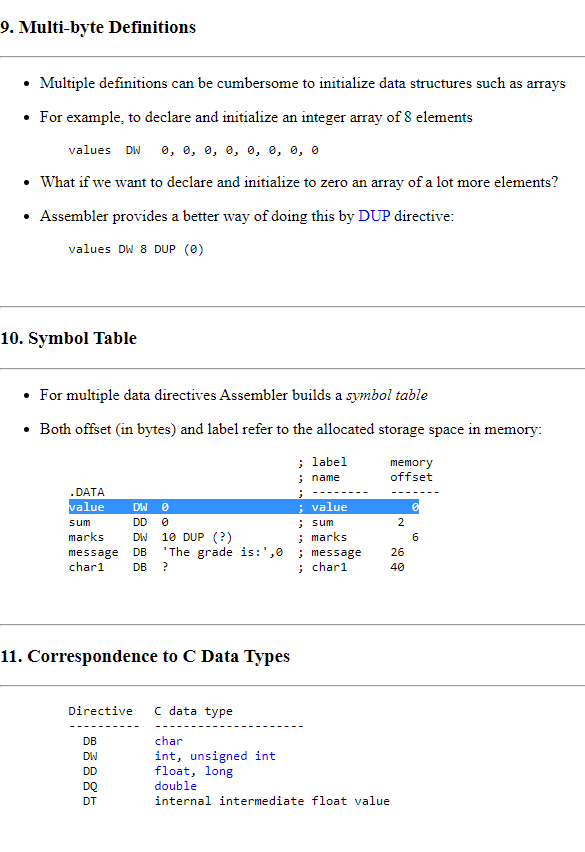
# Chapter 1 Notes

* 
  + L1 = DIGITAL BOOLEAN CIRCUITRY LOGIC (ON’S AND OFF’S)
  + L2 = BINARY numeric version of the digital Boolean circuitry logic (1001010111010101)🡪 TRANSLATES TO ON’S AND OFF’S ON A CIRCUIT
  + L3 = MNEMONICS VERSION OF THE L2 BINARY VALUE; EASIER FOR PROGRAMMER TO WORK WITH NAME VERSIONS OF THE BOOLEAN BINARY OPERATIONS
    - Example: MOV operation in assembly lang translates to some binary value 10111011; MOVE is a mnemonic representation of it’s binary equivalent/expansion; MOVE = some binary value (1101011101 for ex).
  + L4 = upper-level software logic; C++, JAVA, C#, ETC… a further *abstraction* of a set/series/sequence of assembly language operations to make it even easier to write a larger/more complex program with larger more complex operations.
    - For example: int X = 4 (C++; L4) 🡪 MOV X,4 (assembly; L3) 🡪 1010110(L2; ISA)🡪 sequence of ON’S and OFF’S executed/ran on the computer hardware (circuitry, such as the processor; L1 Digital Logic)
* EX2: L4 (A combined, complex Series of MOV/ADD/MULTIPL assembly language operations abstracted into one/fewer high-level language statements) 🡪 L3 (A large, simple stream of 0’s and 1’s that are abstracted into the few, simple Assembly Lang operations) 🡪 L2 (Many combinations of ON’S AND OFF’s abstracted into a binary number system as a swath of 0’s and 1’s to represent the two simple states of circuitry on and off) 🡪 L1: A Circuit Board unintelligently “executing” a large swath of electrical signals being pulsated through its advanced circuitry, by which it is wired in such a way so as to handle and manipulate those electrical signals through simple digital/circuit Boolean logic but at the speed of electricity (nearly light speed) so as to have the capability to execute simple but extraordinarily large and wide-ranging combinations of ON and OFF electrical signals, which allow for the many layers of abstraction above itself in order to represent these simple signals into highly complex, manipulative representations that can be used to solve a problem. Computer circuitry is so small, that is why the processors are often referred to as micro-processors, that there can be millions of circuits carefully and meticulously manufactured into a single micro processing chip. The large swath of circuitry in conjunction with the speed of electricity allows for the speed and the variety of operations with simple ON’s and OFF’s electrical signals.
* **Weighted positional notation:**
  + 
  + D indicates a binary digit. For example, binary 00001001 is equal to 9
  + 
* Translating Unsigned Decimal Integers to Binary and unsigned decimal integers to hexadecimal
  + 
    - Note: answer is to take final remainder and proceed to first remainder from the first division performed (so go from bottom 🡪 up in the table for the remainders)
      * Then solution is: 100101 🡪 using 8 bits: 00100101
    - Also note; you divide by the base you are converting to (ex: binary 🡪 /2; hexadecimal 🡪 /16) since each place value is a power of the base. For example; Hex 1A 🡪 1\*16^1 for the 1 digit, and the A digit 🡪 A\*16^0 ; so you see each place value is a power of the base, in this case base 16 for Hexadecimal.
  + 
  + 
* Bit addition
  + 
* Integer storage sizes
  + 
* Byte prefixes
  + 
  + 
* 8 bits in 1 byte; 2 hexadecimal digits together represent 1 byte; 1 hex digit = represent 4 bits
  + note: because of this fact, you can convert a hex number into binary by converting every 4 bits into a place value of the hex number:
  + 
    - The vice versa works too; you can take every place value of a hex digit and convert it to 4 bits, then combine all bits into one string of bits to get the binary of the hex number
* 
* 
  + See: <http://www.c-jump.com/CIS77/ASM/DataTypes/lecture.html#T77_0020_integers>