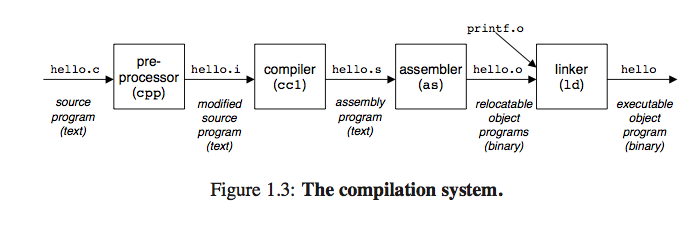
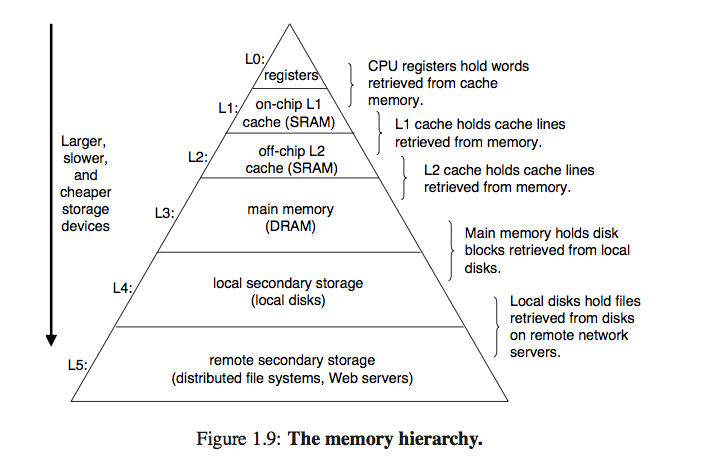
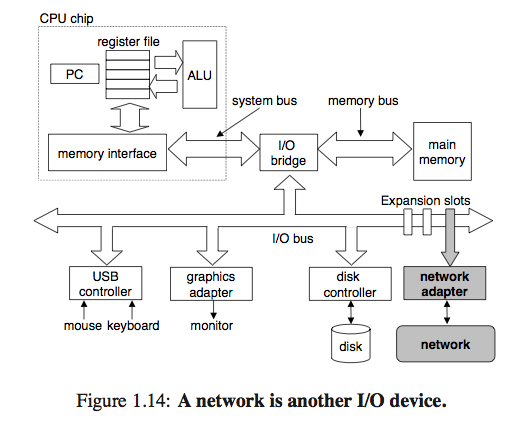
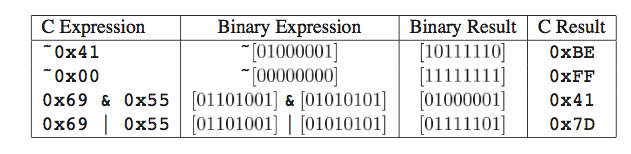
* The hello.c program is stored in a file as a sequence of bytes. Each byte has an integer value that corresponds to some character. For example, the first byte has the integer value 35, which corresponds to the character ’#’. The second byte has the integer value 105, which corresponds to the character ’i’, and so on. Notice that each text line is terminated by the invisible *newline* character ’\n’, which is represented by the integer value 10. Files such as hello.c that consist exclusively of ASCII characters are known as *text files*. All other files are known as *binary files*.
* All information is represented as a bunch of bits.
* 
* The *central processing unit* (CPU), or simply *processor*, is the engine that interprets (or *executes*) instruc- tions stored in main memory. At its core is a word-sized storage device (or *register*) called the *program counter* (PC). At any point in time, the PC points at (contains the address of) some machine-language instruction in main memory.
* (DMA)-direct memory access: data travels directly from disk to main memory.
* For example, the disk drive on a typical system might be 100 times larger than the main memory, but it might take the processor 10,000,000 times longer to read a word from disk than from memory.
* **To deal with the processor-memory gap, system designers include smaller faster storage devices called *caches* that serve as temporary staging areas for information that the processor is likely to need.**
* The L1 and L2 caches are implemented with a hardware technology known as *Static Random Access Memory* (SRAM).
* 
* The main idea of a memory hierarchy is that storage at one level serves as a cache for storage at the next lower level.
* **in modern system a process can actually consist of multiple execution units, called *threads*, each running in the context of the process and sharing the same code and global data.**
* In Linux, the topmost 1/4 of the address space is reserved for code and data in the operating system that is common to all processes. The bottommost 3/4 of the address space holds the code and data defined by the user’s process.
* In particular, each time we call a function, the stack grows. Each time we return from a function, it contracts.
* 
* *virtual memory*. Every byte of memory is identified by a unique number, known as its *address*, and the set of all possible addresses is known as the *virtual address space*. As indicated by its name, this virtual address space is just a conceptual image presented to the machine-level program. The actual implementation (presented in Chapter 10) uses a combination of random-access memory (RAM), disk storage, special hardware, and operating system software to provide the program with what appears to be a monolithic byte array.
* Instruction codings are different, except for the NT and Linux machines. Different machine types use different and incompatible instructions and encodings. The NT and Linux machines both have Intel processors and hence support the same machine-level instructions.
* 
* Logical operations:
* 