

Lecture IV: Computer Structure and Basic Programming

Charles B. Moss¹

¹University of Florida

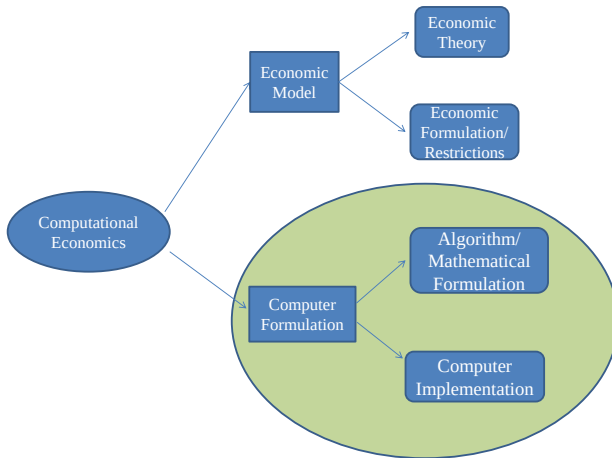
August 28, 2012

- 1 Computer Structure and Basic Programming
- 2 Computer Architecture
 - Turning the Computer On
- 3 Compilers Versus Application Packages
- 4 Binary Numbers and the Computer

Computer Structure and Basic Programming

In the last couple of lecture, we covered three research efforts that emphasize in one way or another computational economics. In fact, the field is fairly broad and even can include computational efforts in econometrics. The final point was a division of computational economics into four components. As depicted in Figure 4, we are going to start at the bottom of the diagram (farthest away from economic theory).

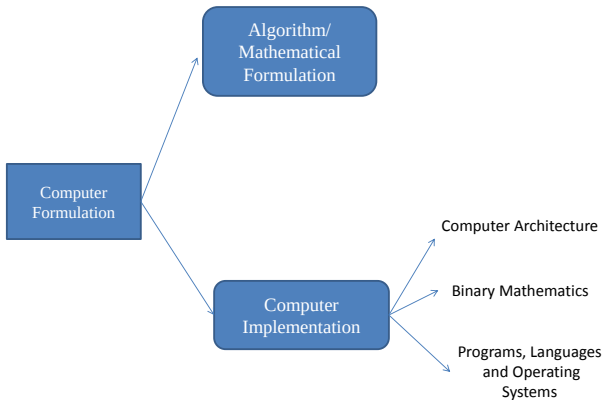
Overview of Computational Economics



Computer Architecture

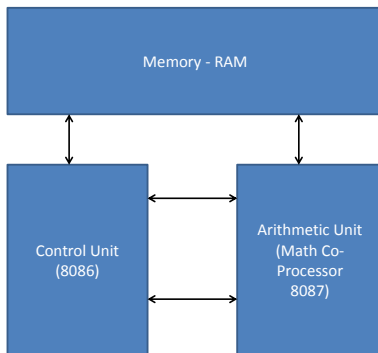
This lecture starts at the very bottom with a discussion of computer architecture, binary mathematics, and programs, languages, and operating systems (the diodes and switches of the computer).

Basic Computer Knowledge

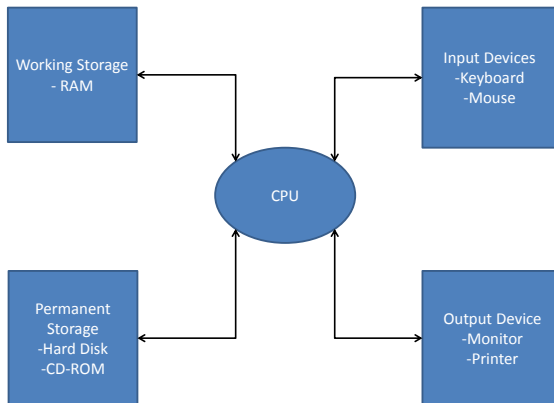


Much of this discussion can be found in Pearson Certification Team. 2011. *Computer Structure and Logic* Pearson Education. The first personal computers followed the general framework of "The von Neumann Computer Model" as depicted in Figure 8. Over time, this simple structure has evolved into the "Five Main Groups within the von Neumann Computer" as presented in Figure 9.

von Neumann Computer Model



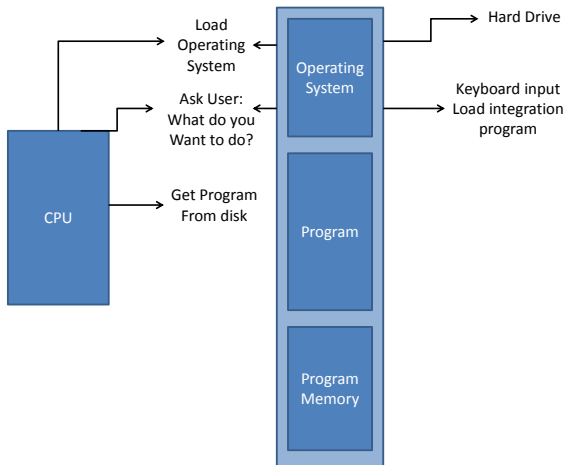
Five Groups within the von Neuman Computer



Turning the Computer On

- When a computer is turned on, the standard first operation is for the control unit to go to the first memory location and execute the stored command.
- Typically, the core memory is divided into the "operating system" which contains the general set of instructions as to how the computer interacts with the user -
 - Unix
 - IBM TSO (Time Sharing)
 - Microsoft DOS (Disk Operating System)
 - Windows
- The operating system controls how the computer executes the user code.

Executing Program within the Computer



Compilers Versus Application Packages

Our interest in computer architecture involves writing the program that the operating system calls. In general, we are interested in two types of computer code: Compiled Languages such as Fortran and Application Packages such as GAMS, Gauss, MatLab, etc. To make an economist proud, the choice between compiled languages and application packages involves tradeoff speed, efficiency, and simplicity.

As a starting point, consider a compiled language such as Fortran. Remember that computers execute machine language (a form of binary instructions unique to a particular processor). Consider the simple program to ask a computer compute the value of a Cobb-Douglas utility function in Fortran.

Fortran Code

```
program util
implicit real*8(a-h,o-z)
double precision x(3),a(3),g(3),dutil
data x /5.00,5.00,7.50/
data a /0.35,0.25,0.40/
open(unit=6,file=" Example01-01.out")
```

C_____

C

C Compute the utility function

C

C_____

```
dutil=(x(1)**a(1))*(x(2)**a(2))*(x(3)**a(3))
```

C_____

C

C Compute the gradients of the utility function

C

C_____

```
do 50 i=1,3
```

$$g(i) = a(i) * dutil / x(i)$$

50 continue

```
write(6,100) (x(i),i=1,3),dutil,(g(i),i=1,3)
```

```
100 format(' ',10f10.4)
```

end

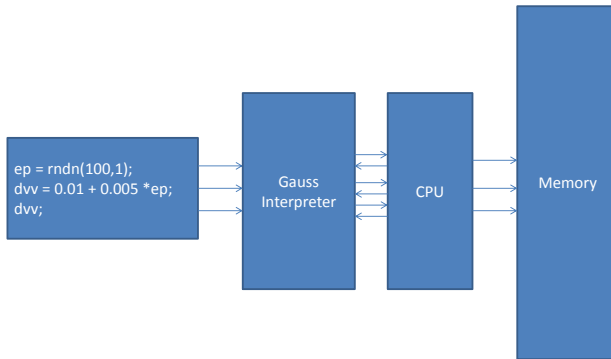
This code is compiled into what is referred to as an object file. The object file is then linked to become an executable file say utility.exe which can be executed by the program. This is the set of instructions that are actually loaded into memory in Figure 2. It is important to note that while the Fortran code is portable (can be used on any computer with a Fortran compiler), the object and executable codes are specific to the computer (CPU instruction set). Hence, differences in compilers may make it difficult to "port" applications between computers.

Each language has its own particularities. Fortran requires us to specify things like the size of the memory required, but executes very quickly. The efficiency of the code (both in terms of memory use and speed of application) is dependent on the compiler used. The object code step is important because it allows us to build libraries of routines that we can use on different programs. In fact several Numeric Recipe books for various languages exist.

The difference between a compiled language and an application package may appear subtle, but could have significant implications for your computations. An application program is basically a program between your program and the computer executable file. For example, consider the simply Gauss code to generate a random walk

```
ep = rndn(100,1);  
dvv = 0.01 + 0.005*ep;  
dvv;
```

Interpreted Language



One way to conceptualize the difference is that the interpreter compiles the system line-by-line. Because the applications packages are written around certain uses they are typically easier to use (i.e., there is already an object file for the inverse of a matrix or drawing random normal variables). The difficulty involves the fact that you are limited to either the functions written by the designers of the code or the "fundamentals" of the application package.

Binary Numbers and the Computer

Number theory is always abstract. As a starting point, consider the construction of a number such as 27 in decimal form. The number can be written as

$$2 \times 10^1 + 7 \times 10^0 \quad (1)$$

Modern computers are digital in that they only deal with two numbers (a base two system) – 0 and 1. Thus, the number 27 is written as

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \quad (2)$$

Thus, $11011 \Rightarrow 27$.

Technically, this set of numbers is known as "unsigned integers". In the case of signed integer, the first bit (0 or 1) would be the sign bit (0 - positive and 1 - negative). An important point in computer programming is how big the number could be. Typically, a certain number of bits or memory locations are assigned to a number. Single precision involves 16 bits where double precision numbers are assigned 32 bits.

It is important to note that all the numbers defined in Equation 2 are whole numbers – integers. Another set of numbers are referred to as real numbers. As depicted in scientific notation, real numbers are a combination of a mantissa (or significand) and an exponent (typically a power of 10). Hence, 27 could be written as

$$2.700000 \times 10^1 \quad (3)$$

In this case the memory location for 27 is a linked set of binary locations. For example, the first number is a sign bit, the next twelve numbers are binary powers for the mantissa and the last three bits are a sign bit for the exponent and two number bits for the power of ten – sixteen bits.

The important thing is that references an integer as a real number of visa versa could yield really bad results. Some compilers and interpreters are picky about the type of number.