Chapter 4

Chess Anyone?

Computers, Programs, & Algorithms

Learning Objectives

- *State the purpose and use of a computer*
- *State the primary characteristic that makes the computer a unique device*
- *List and describe the four stages of the program execution cycle*
- *Explain how a computer stores and retrieves programs for execution*
- *State the difference between a computer and a computer system*
- *Define the concept of a program from the human and computer perspective*
- *State the purpose and use of main, auxiliary, and cache memory*
- *Describe how programs are loaded into main memory and executed by a computer.*
- *State the purpose and use of the Java virtual machine*
- *List the similarities between the Java virtual machine and a real computer*
- *Describe the architecture of the Java HotSpot™ Virtual Machine*
- *Explain the purpose of byte code*
- *Define the concept of an algorithm*

Introduction

Computers, programs, and algorithms are three closely related topics that deserve special attention before you start learning about Java proper. Why? Simply put, computers execute programs, and programs implement algorithms. As a programmer, you will live your life in the world of computers, programs, and algorithms.

As you progress through your studies, you will find it extremely helpful to understand what makes a computer a computer, what particular feature makes a computer a truly remarkable device, and how one functions from a programmer's point of view. You will also find it helpful to know how humans view programs, and how human-readable program instructions are translated into a computer-executable form.

Next, it will be imperative for you to thoroughly understand the concept of an algorithm and to understand how good and bad algorithms ultimately affect program performance.

Finally, I will show you how Java programs are transformed into byte code and executed by a Java virtual machine. Armed with a fundamental understanding of computers, programs, and algorithms, you will be better prepared to understand the concepts of the Java virtual machine as well as its execution performance and security ramifications.

What Is A Computer?

A computer is a device whose function, purpose, and behavior is prescribed, controlled, or changed via a set of stored instructions. A computer can also be described as a general-purpose machine. One minute a computer may execute instructions making it function as a word processor or page-layout machine. The next minute it might be functioning as a digital canvas for an artist. Again, this functionality is implemented as a series of instructions. Indeed, in each case the only difference between the computer functioning as a word processor and the same computer functioning as a digital canvas is in the set of instructions the computer is executing.

Computer vs. Computer System

Due to the ever-shrinking size of the modern computer it is often difficult for students to separate the concept of the computer from the computer system in which it resides. As a programmer, you will be concerned with both. You will need to understand issues related to the particular processor that powers a computer system in addition to issues related to the computer system as a whole. Luckily though, as a Java programmer, you can be extremely productive armed with only a high-level understanding of each. Ultimately, I highly recommend spending the time required to get intimately familiar with how your computer operates. For this chapter I will use the Apple Power Mac G4 as an example, but the concepts are the same for any computer or computer system.

Computer System

A typical Power Mac G4 computer system is pictured in figure 4-1.

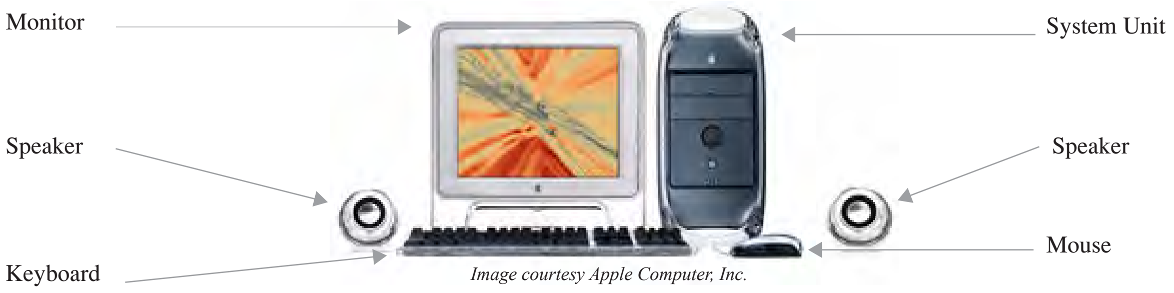


Figure 4-1: Typical Power Mac G4 System

The computer system comprises the system unit, monitor, speakers, keyboard, and mouse. The computer system also includes any operating system or utility software required to make all the components work together.

The system unit houses the processor, power supply, internal hard disk drives, memory, and other system components required to interface the computer to the outside world. These interface components consume the majority of available space within the system unit as shown in figure 4-2.

Image courtesy Apple Computer, Inc.

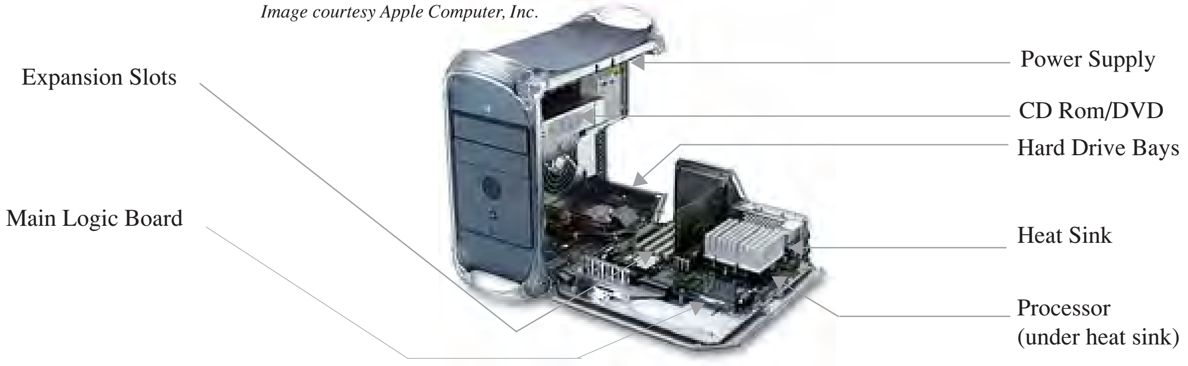


Figure 4-2: System Unit

The processor is connected to the system unit's main logic board. Electronic pathways called buses connect the processor to the various interface components. Other miscellaneous electronic components are located on the main logic board to control the flow of communication between the processor and the outside world. Figure 4-3 is a block diagram of a Power Mac G4 main logic board.

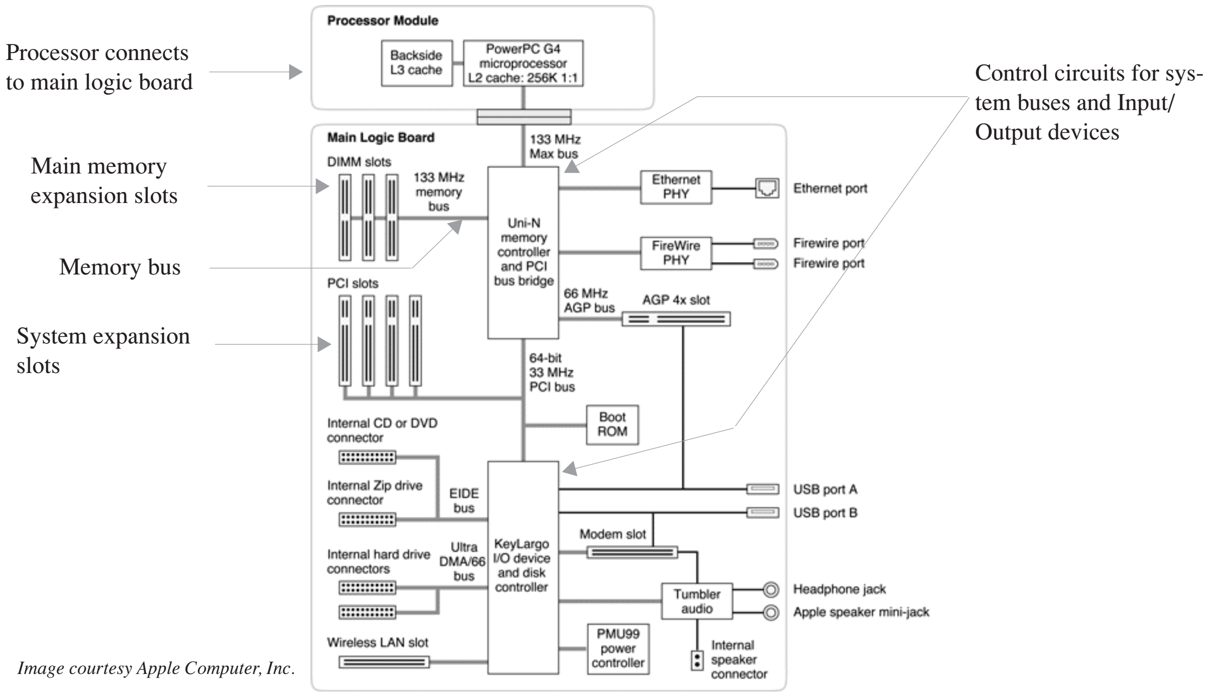


Image courtesy Apple Computer, Inc.

Figure 4-3: Main Logic Board Block Diagram

Figure 4-3 does a good job of highlighting the number of computer system support components required to help the processor do its job. The main logic board supports the addition of main memory, auxiliary storage devices, communication devices such as a modem, a wireless local area network card as well as a standard Ethernet port, keyboard, mouse, speakers, microphones, Firewire devices, and, with the insertion of the appropriate third party system expansion card, just about any other functionality you can imagine. All this system functionality is supported by the main logic board, but the heart of the system is the PowerPC G4 processor. Let's take a closer look.

Processor

If you lift up the heat sink pictured in figure 4-2 you would see a PowerPC G4 processor like the one shown in figure 4-4.

The PowerPC G4 7400 microprocessor pictured here runs at speeds between 350 and 500 megahertz with a Millions of Instructions per Second (MIPS) rating of 917 MIPS at 500 megahertz. The G4 is a powerful processor, yet at the time of this writing there are more powerful processors on the market, namely the G5!

The 7400 processor is a Reduced Instruction Set Computer (RISC) meaning its architecture is optimized to execute instructions in as few clock cycles as possible. The block diagram for the 7400 is shown in figure 4-5 and is even more impressive than that of the main logic board. The G4 is a superscalar, pipelined processor. It is superscalar in that it can pull two instructions from the incoming instruction stream and execute them simultaneously. It is pipelined in that instruction execution is broken down into discrete steps meaning several instructions can be in the pipeline at any given moment at different stages of execution.

Figure 4-4: PowerPC G4 Processor

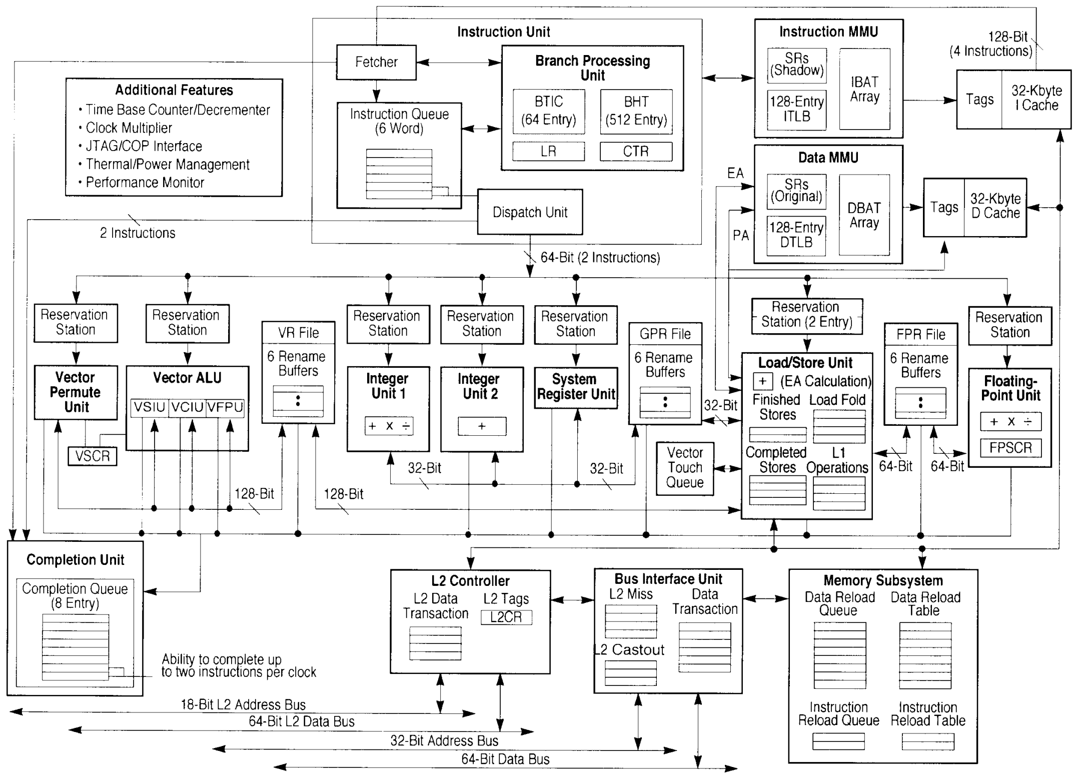


Figure 4-5: Motorola PowerPC 7400 Block Diagram

Three Aspects of Processor Architecture

There are generally three aspects of processor architecture programmers should be aware of: feature set, feature set implementation, and feature set accessibility.

Feature Set

A processor's feature set is derived from its design. Can floating point arithmetic be executed in hardware or must it be emulated in software? Must all data pass through the processor or can input/output be handled off chip while the processor goes about its business? How much memory can the processor access? How fast can it run? How much data can it process per unit time? A processor’s design addresses these and other feature set issues.

Feature Set Implementation

This aspect of computer architecture is concerned primarily with how processor functionality is arranged and executed in hardware. How does the processor implement the feature set? Is it a Reduced Instruction Set Computer (RISC), or Complex Instruction Set Computer (CISC)? Is it superscalar and pipelined? Does it have a vector execution unit? Is the floating-point unit on the chip with the processor or does it sit off to the side? Is the super-fast cache memory part of the processor or is it located on another chip? These questions all deal with how processor functionality is achieved or how its design is executed.

Feature Set Accessibility

Feature set accessibility is the aspect of a processor's architecture you are most concerned with as a programmer. Processor designers make a processor's feature set available to programmers via the processor's instruction set. A valid instruction in a processor's raw instruction set is a set of voltage levels that, when decoded by the processor, have special meaning. A high voltage is usually translated as “on” or “1” and a low voltage is usually translated as “off” or “0”. A set of on and off voltages is conveniently represented to humans as a string of ones and zeros. Instructions in this format are generally referred to as machine instructions or machine code. However, as processor power increases, the size of machine instructions grow as well, making it extremely difficult for programmers to deal directly with machine code.

From Machine Code To Assembly

To make a processor's instruction set easier for humans to understand and work with each machine instruction is represented symbolically in a set of instructions referred to as an assembly language. To the programmer, assembly language represents an abstraction or a layer between programmer and machine intended to make the act of programming more efficient. Programs written in assembly language must be translated into machine instructions before being executed by the processor. A program called an assembler translates assembly language into machine code.

Although assembly language is easier to work with than machine code it requires a lot of effort to crank out a program in assembly code. Assembly language programmers must busy themselves with issues like register usage and stack conventions.

High-level programming languages like Java add yet another layer of abstraction. Java, with its object-oriented language features, lets programmers think in terms of solving the problem at hand, not in terms of the processor or the machine code that it’s ultimately executing.

Memory Organization

Modern computer systems have similar memory organizations and as a programmer you should be aware of how computer memory is organized and accessed. The best way to get a good feel for how your computer works is to poke around in memory and see what’s in there for yourself. This section provides a brief introduction to computer memory concepts to help get you started.

Memory Basics

A computer's memory stores information in the form of electronic voltages. There are two general types of memory: volatile and non-volatile. Volatile memory will lose any information stored there if power is removed for any length of time. Main memory and cache memory, two forms of Random Access Memory (RAM), are examples of volatile memory. Read Only Memory (ROM) and auxiliary storage devices such as CD ROMs, DVDs, hard disk drives, memory sticks, floppy disks, and tapes, are examples of non-volatile memory.

Memory Hierarchy

Computer systems contain several different types of memory. These memory types range from slow and cheap to fast and expensive. The proportion of slow cheap memory to fast expensive memory can be viewed in the shape of a pyramid commonly referred to as the memory hierarchy as is shown in figure 4-6.

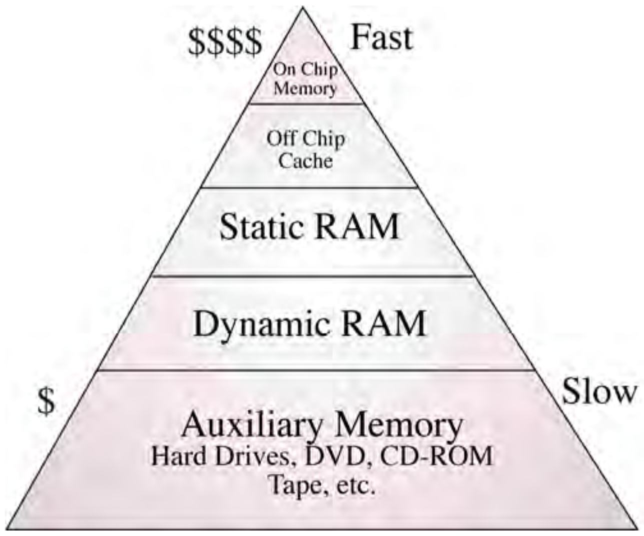


Figure 4-6: Memory Hierarchy

The job of a computer-system designer, with regards to memory subsystems, is to make the whole computer perform as if all the memory were fast and expensive. Thus they utilize cache memory to store frequently used data and instructions and buffer disk reads to memory to give the appearance of faster disk access. Figure 4-7 shows a block diagram of the different types of memory used in a typical computer system.

During program execution, the faster cache memory is checked first for any requested data or instruction. If it's not there, a performance penalty is extracted in the form of longer overall access times required to retrieve the information from a slower memory source.

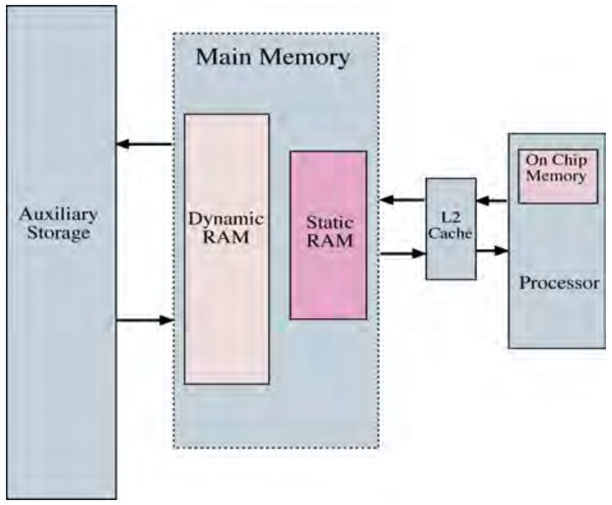


Figure 4-7: Simplified Memory Subsystem Diagram

Bits, Bytes, Words

Program code and data are stored in main memory as electronic voltages. Since I'm talking about digital computers the voltages levels represent two discrete states depending on the level. Usually low voltages represent no value, off, or 0, while a high voltage represents on, or 1.

When program code and data is stored on auxiliary memory devices, electronic voltages are translated into either electromagnetic fields (tape drives, floppy and hard disks) or bumps that can be detected by laser beam (CDs, DVDs, etc.).

Bit

The bit represents one discrete piece of information stored in a computer. On most modern computer systems bits cannot be individually accessed from memory. However, after the byte to which a bit belongs is loaded into the processor the byte can be manipulated to access a particular bit.

Byte

A byte contains 8 bits. Most computer memory is byte addressable, although as processors become increasingly powerful and can manipulate wider memory words, loading bytes by themselves into the processor becomes increasingly inefficient. This is the case with the G4 processor and for that reason the fastest memory reads can be done a word at a time.

Word

A word is a collection of bytes. The number of bytes that comprise a word is computer-system dependent. If a computer's data bus is 32 bits wide and its processor's registers are 32 bits wide, then the word size would be 4 bytes long. (*32 bits / 8 bits = 4 bytes*) Bigger computers will have larger word sizes. This means they can manipulate more information per unit time than a computer with a smaller word size.

Alignment and Addressability

You can expect to find your computer system's memory to be byte addressable and word aligned. Figure 4-8 shows a simplified diagram of a main memory divided into bytes and the different buses connecting it to the processor. In this diagram the word size is 32 bits wide.

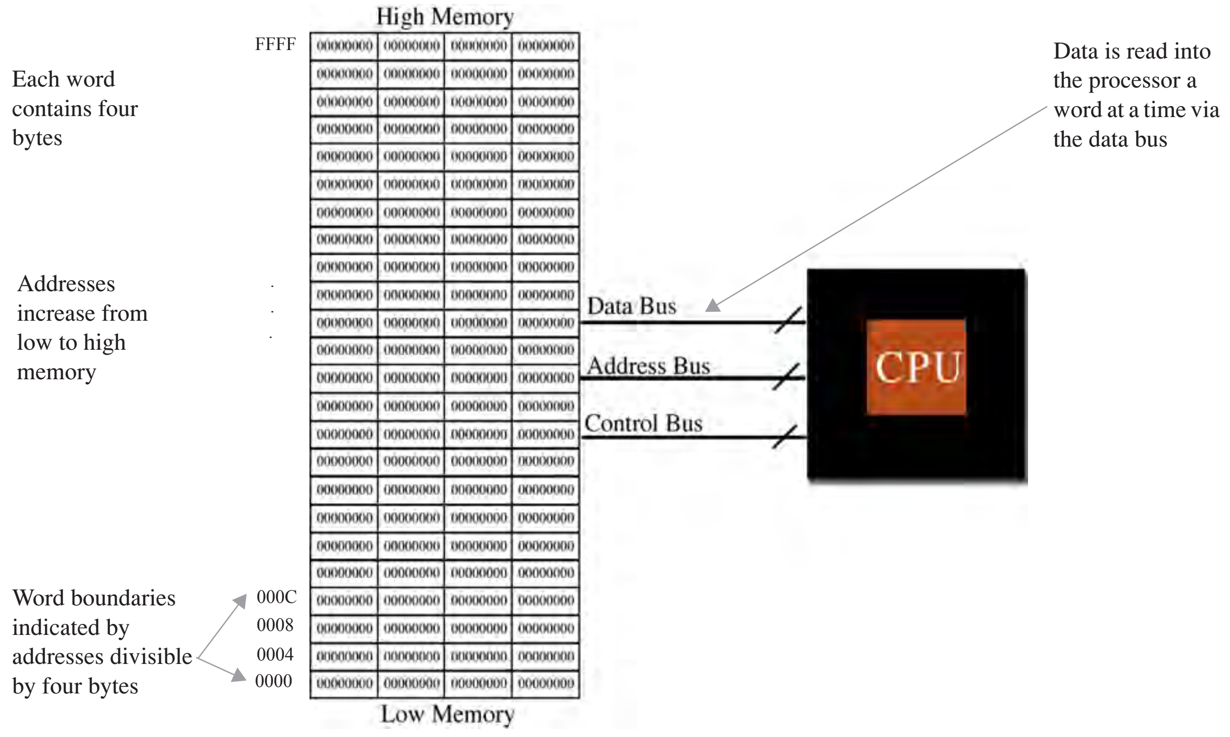

High Memory
FFFF
Each word contains four bytes
Data is read into the processor a word at a time via the data bus
Addresses increase from low to high memory
Data Bus
Address Bus
Control Bus
CPU
Word boundaries indicated by addresses divisible by four bytes
000C
0008
0004
0000
00000000
Low Memory

Figure 4-8: Simplified Main Memory Diagram

The memory is byte addressable in that each byte can be individually accessed although the entire word that contains the byte is read into the processor. Data in memory can be aligned for efficient manipulation. Alignment can be to natural or some other boundary. For example, on a PowerPC system, contents of memory assigned to instances of structures is aligned to natural boundaries meaning a one-byte data element will be aligned to a one-byte boundary. A two-byte element would be aligned to a two-byte boundary. Individual data elements not belonging to structures are usually aligned to four-byte boundaries. (*Note: The term "structure" denotes a C or C++ structure. In Java there is no equivalent and the Java virtual machine prevents direct memory manipulation.*)

What Is A Program?

Intuitively you already know the answer to this question. A program is something that runs on a computer. This simple definition works well enough for most purposes, but as a programmer you will need to arm yourself with a better understanding of exactly what makes a program a program. In this section I will discuss programs from two aspects: the computer and the human. You will find this information extremely helpful and it will tide you over until you take a formal course on computer architecture.

Two Views of a Program

A program is a set of programming language instructions plus any data the instructions act upon or manipulate. This is a reasonable definition and if you are a human it will do. If you are a processor it will just not fly. That's because humans are great abstract thinkers and computers are not, so it is helpful to view the definition of a program from two points of view.

The Human Perspective

Humans are the masters of abstract thought; it is the hallmark of our intelligence. High-level, object-oriented languages like Java give us the ability to analyze a problem abstractly and symbolically express its solution in a form that is both understandable by humans and readable by other programs. By other programs I mean the Java code a programmer writes must be translated from Java into machine instructions recognizable by a particular processor. This translation is effected by running a compiler that converts the Java code into bytecode that is then executed by a Java virtual machine.

To a Java programmer a program is a collection of classes that model the behavior of objects in a particular problem domain. These classes model object behavior by defining object attributes (data) and class methods to manipulate the class attributes. On an even higher level, a program can be viewed as an interaction between objects. This view of a program is convenient for humans.

The Computer Perspective

From the computer's perspective a program is simply machine instructions and data. Usually both the instructions and data reside in the same memory space. This is referred to as a Von Neumann architecture. In order for a program to run it must be loaded into main memory and the address of the first instruction of the program given to the processor. In the early days of computing, programs were coded into computers by hand and then executed. Nowadays all the nasty details of loading programs from auxiliary memory into main memory are handled by an operating system, which, by the way, is a program.

Since both instructions and data reside in main memory how does a computer know when it is dealing with an instruction or with data? The answer to this question will be discussed in detail below but here's a quick answer: It depends on what the computer is expecting. If a computer reads a memory location expecting to find an instruction and it does, everything runs fine. The instruction is decoded and executed. If it reads a memory location expecting to find an instruction but instead finds garbage, then the decode fails and the computer might lock up!

The Processing Cycle

Computers are powerful because they can do repetitive things really fast. When the executable code is loaded into main memory the processor can start executing the machine instructions. When a computer executes or runs a program it constantly repeats a series of processing steps commonly referred to as the processing cycle. The processing cycle consists of four primary steps: instruction fetch, instruction decode, instruction execution, and result store.

The step names can be shortened to simply fetch, decode, execute, and store. Different processors will implement the processing cycle differently but for the most part these four processing steps are being carried out in some form or another. The processing cycle is depicted in figure 4-9.

Fetch

In the fetch step, an instruction is read from memory and presented to the processor's decode section. If cache memory is present it is checked first. If the requested memory address contents resides in the cache the read operation will execute quickly. Otherwise, the processor will have to wait for the data to be accessed from the slower main memory.

Decode

In the decode step, the instruction fetched from memory is decoded. If the computer thinks it is getting an instruction but instead it gets garbage there will be problems. A computer system's ability to recover from such situations is generally a function of a robust operating system.

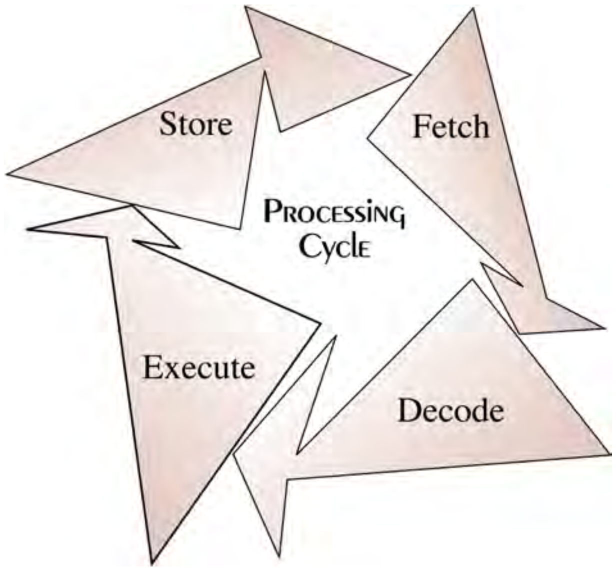


Figure 4-9: Processing Cycle

Execute

If the fetched instruction is successfully decoded, meaning it is a valid instruction in the processor's instruction set, it is executed. A computer is a bunch of electronic switches. Executing an instruction means the computer's electronic switches are turned either on or off to carry out the actions required by a particular instruction. (*Different instructions cause different sets of switches to be turned on or off.*)

Store

After an instruction is executed the results of the execution, if any, must be stored somewhere. Most arithmetic instructions leave the result in one of the processor's onboard registers. Memory write instructions would then be used to transfer the results to main memory. Keep in mind that there is only so much storage space inside a processor. At any given time, almost all data and instructions reside in main memory, and are only loaded into the processor when needed.

Why A Program Crashes

Notwithstanding catastrophic hardware failure, a computer crashes or locks up because what it was told was an instruction was not! The faulty instruction loaded from memory turns out to be an unrecognizable string of ones and zeros and when it fails to decode into a proper instruction the computer halts.

Algorithms

Computers run programs; programs implement algorithms. A good working definition of an algorithm for the purpose of this book is that an algorithm is a recipe for getting something done on a computer. Pretty much every line of source code you write is considered part of an algorithm. What I'd like to do in this brief section is bring to your attention the concept of good vs. bad algorithms.

Good vs. Bad Algorithms

There are good ways to do something in source code and there are bad ways to do the same exact thing. A good example of this can be found in the act of sorting. Suppose you want to sort in ascending order the following list of integers:

1, 10, 7, 3, 9, 2, 4, 6, 5, 8, 0, 11

One algorithm for doing the sort might go something like this:

Step 1: Select the first integer position in the list
Step 2: Compare the selected integer with its immediate neighbor
Step 2.2: If the selected integer is greater than its neighbor swap the two integers
Step 2.3: Else, leave it where it is
Step 3: Continue comparing selected integer position with all other integers repeating steps 2.2 - 2.3
Step 4: Select the second integer position on the list and repeat the procedure beginning at step 2
Continue in this fashion until all integers have been compared to all other integers in the list and have been placed in their proper position.

This algorithm is simple and straightforward. It also runs pretty fast for small lists of integers but it is really slow given large lists of integers to sort. Another sorting algorithm to sort the same list of integers goes as follows:

Step 1: Split the list into two equal sublists
Step 2: Repeat step 1 if any sublist contains more than two integers
Step 3: Sort each sublist of two integers
Step 4: Combine sorted sublists until all sorted sublists have been combined

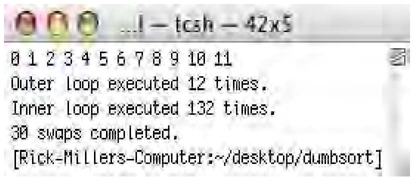
This algorithm runs a little slow on small lists because of all the list splitting going on but sorts large lists of integers way faster than the first algorithm. The first algorithm lists the steps for a routine I call dumb sort. Example 4.1 gives the source code for a short program that implements the dumb sort algorithm.

4.1 DumbSort.java

```
1        public class DumbSort{
2         public static void main(String[] args){
3
4            int a[] = {1,10,7,3,9,2,4,6,5,8,0,11};
5
6            int innerloop = 0;
7            int outerloop = 0;
8            int swaps = 0;
9
10           for(int i = 0; i<12; i++){
11             outerloop++;
12               for(int j = 1; j<12; j++){
13                 innerloop++;
14                   if(a[j-1] > a[j]) {
15                       int temp = a[j-1];
16                       a[j-1] = a[j];
17                       a[j] = temp;
18                       swaps++;}}}
19
20           for(int i = 0; i<12; i++)
21               System.out.print(a[i] + " ");
22
23           System.out.println();
24           System.out.println("Outer loop executed " + outerloop + " times.");
25           System.out.println("Inner loop executed " + innerloop + " times.");
26           System.out.println(swaps + " swaps completed.");
27        }
28       }
```

Included in the dumb sort test source code are a few variables intended to help collect statistics during execution. These are `innerloop`, `outerloop`, and `swaps` declared on lines 6, 7, and 8 respectively. Figure 4-10 gives the results from running the dumb sort test program.

Notice that the inner loop executed 132 times and that 30 swaps were conducted. Can the algorithm run any better? One way to check is to rearrange the order of the integers in the array. What if the list of integers is already sorted? Figure 4-11 gives the results of running dumb sort on an already sorted list of integers:

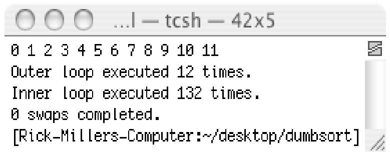


```
0 1 2 3 4 5 6 7 8 9 10 11
Outer loop executed 12 times.
Inner loop executed 132 times.
30 swaps completed.
[Rick-Millers-Computer:~/desktop/dumbsort]
```

Figure 4-10: Dumb Sort Results 1

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

It appears that both the outer loop and inner loop are executed the same number of times in each case, which is of course the way the source code is written, but it did run a little faster because fewer swaps were necessary.



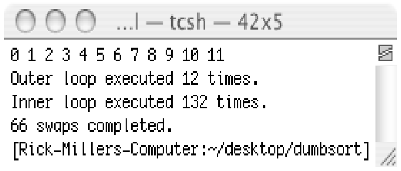
```
0 1 2 3 4 5 6 7 8 9 10 11
Outer loop executed 12 times.
Inner loop executed 132 times.
0 swaps completed.
[Rick-Millers-Computer:~/desktop/dumbsort]
```

Figure 4-11: Dumb Sort Results 2

Can the algorithm run any worse? What if the list of integers is completely unsorted? Figure 4-12 gives the results of running dumb sort on a completely unsorted list:

11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0

The outer loop and inner loop executed the same number of times but 66 swaps were necessary to put everything in ascending order. So it did run a little slower this time.



```
0 1 2 3 4 5 6 7 8 9 10 11
Outer loop executed 12 times.
Inner loop executed 132 times.
66 swaps completed.
[Rick-Millers-Computer:~/desktop/dumbsort]
```

Figure 4-12: Dumb Sort Results 3

In dumb sort, because we're sorting a list of 12 integers, the inner loop executes 12 times for every time the outer loop executes. If dumb sort needed to sort 10,000 integers then the inner loop would need to execute 10,000 times for every time the outer loop executed. To generalize the performance of dumb sort you could say that for some number N integers to sort, dumb sort executes the inner loop roughly N x N times. There is some other stuff going on besides loop iterations but when N gets really large, the loop iteration becomes the overwhelming measure of dumb sort's performance as a sorting algorithm. Computer scientists would say that dumb sort has order N^2 performance. Saying it another way, for a really large list of integers to sort, the time it takes dumb sort to do its job is approximately the square of the number N of integers that need to be sorted.

When an algorithm's running time is a function of the size of its input the term used to describe the growth in time to perform its job vs. the size of the input is called the growth rate. Figure 4-13 shows a plot of algorithms with the following growth rates: *log n, n, n log n,* n^2, n^3, n^n

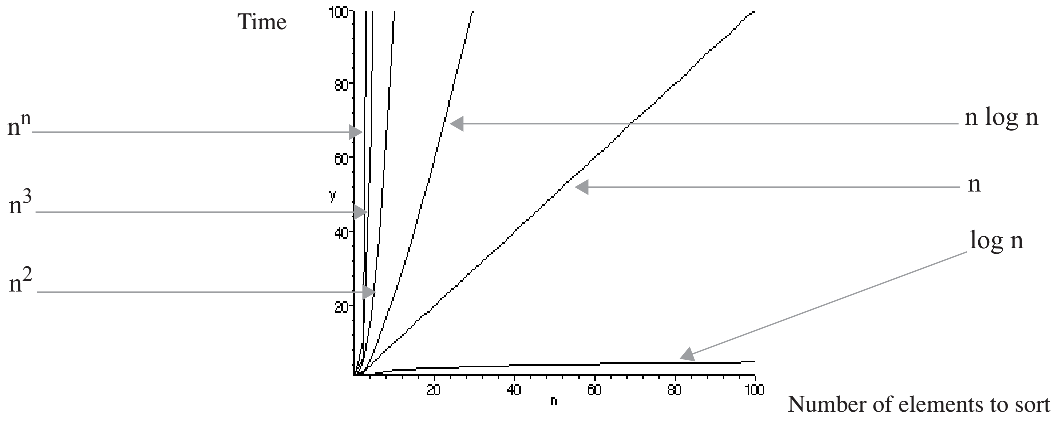


Figure 4-13: Algorithmic Growth Rates

As you can see from the graph, dumb sort, with a growth rate of n^2, is a bad algorithm, but not as bad as some other algorithms. The good thing about dumb sort is that no matter how big its input grows, it will eventually sort all the integers. Sorting problems are easily solved. There are some problems, however, that defy straightforward algorithmic solution.

Don't Reinvent The Wheel!

If you are new to programming the best advice I can offer is for you to seek the knowledge of those who have come before you. There are many good books on algorithms, some of which are listed in the reference section. Studying good algorithms helps you write better code.

The Java HotSpot™ Virtual Machine

The Java HotSpot™ virtual machine (JVM) is a program that runs Java programs. The JVM is targeted to work on specific hardware platforms and host operating systems such as Intel™ processors running the Microsoft Windows™ operating system or the PowerPC™ processor running Mac OS X as is shown in figure 4-14.

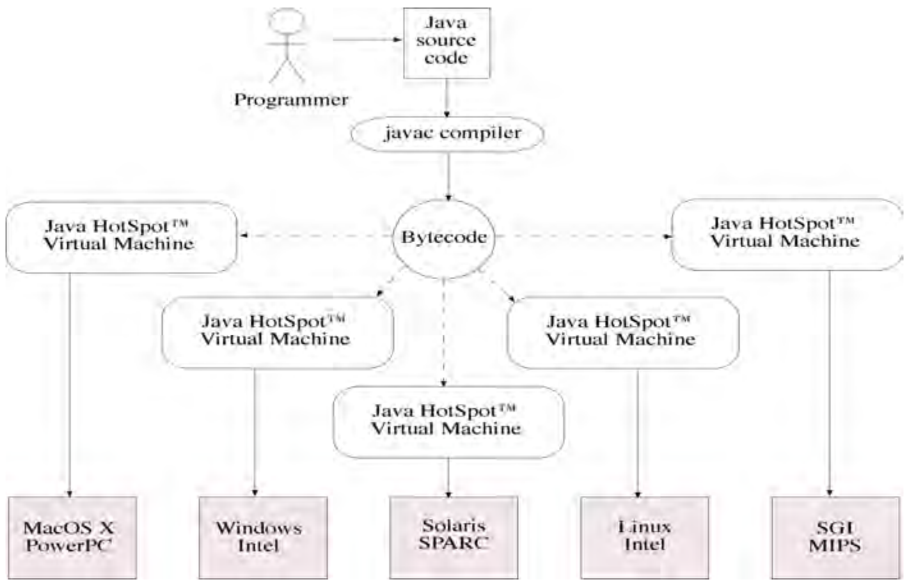

Programmer
Java source code
javac compiler
Bytecode
Java HotSpot™ Virtual Machine
Java HotSpot™ Virtual Machine
Java HotSpot™ Virtual Machine
Java HotSpot™ Virtual Machine
Java HotSpot™ Virtual Machine
MacOS X PowerPC
Windows Intel
Solaris SPARC
Linux Intel
SGI MIPS

Figure 4-14: Java HotSpot™ Virtual Machine Targets Specific Hardware Platforms

As figure 4-14 illustrates, the concept of the virtual machine facilitates the write-once, run-anywhere nature of Java code. Programmers write Java programs that can run on any computing platform that implements a Java virtual machine.

Obtaining The Java HotSpot™ Virtual Machine

When you download the Java Software Developers Kit (SDK) it comes with a Java Runtime Environment (JRE) that includes two versions of the JVM: a client version and a server version.

Client & Server Virtual Machines

When you run Java programs with the java command-line tool you are executing them with the client version of the JVM, which is the default virtual machine. If you specify -server as an argument to the java command-line tool you will execute your programs using the server version of the JVM.

The difference between the two version of the JVM lies in how each version optimizes the java bytecode execution. The client version optimizes execution primarily to support fast program startups and graphical user interfaces (GUIs). The server optimizes execution to support often-repeated code snippets.

Classic VM vs. JIT vs. HotSpot™

The first several releases of the Java runtime environment shipped with what is now referred to as the classic virtual machine. The classic virtual machine ran Java bytecode in an interpreted mode, meaning each bytecode instruction was executed by the virtual machine. The virtual machine was responsible for translating each bytecode instruction into a series of virtual machine instructions that then performed the operation indicated by the bytecode instruction.

Later releases of the Java runtime environment added the services of a just-in-time (JIT) compiler to the virtual machine architecture. The purpose of the JIT is to compile bytecode instructions into processor native code that can be executed directly on the target hardware platform. The JIT provided improvements in execution speed but extracted an upfront penalty due to the compilation process since all bytecodes are compiled prior to execution.

The HotSpot virtual machine provides performance improvements over both the classic VM and the JIT. The HotSpot virtual machine utililizes the services of a bytecode profiler. As bytecode is executed by the virtual machine, an analysis is performed to determine the locations of critical bytecode sequences, or hotspots, and to then compile those bytecode hotspots into native code. Programs startup faster and, as execution progresses, also benefit from native processor execution speeds.

Java HotSpot™ Virtual Machine Architecture

Figure 4-15 shows a high-level Java HotSpot virtual machine architectural diagram Bytecode loaded into and executed by the HotSpot VM is initially interpreted. As execution progresses information about the code's execution behavior is obtained by the profiler. Frequently executed bytecode sequences are compiled into host-computer native code. Bytecode evaluation continues throughout the execution lifetime of the program.

There are two versions of the adaptive compiler: one tuned specifically to compile client programs and one tuned to compile server programs. Use of the client compiler results in smaller program memory footprints and quicker program startup times. Use of the server compiler results in more stringent code optimizations to extract every ounce of performance..

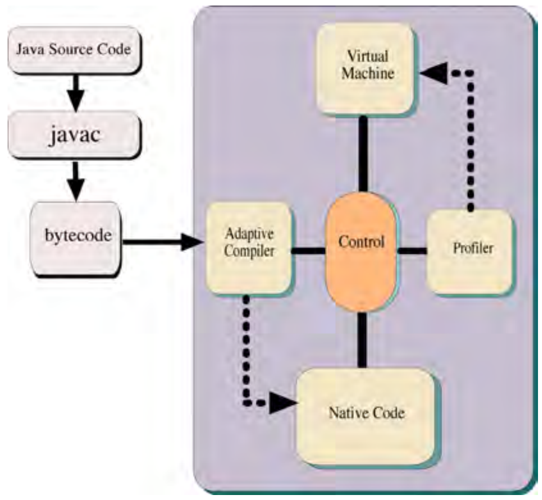


Figure 4-15: Java HotSpot™ Virtual Machine Architecture

Services Provided By The Java HotSpot™ Virtual Machine and the JRE

The HotSpot virtual machine together with the JRE provides several important services to Java programs. These services include class loading, security, memory management and garbage collection, preemptive multithreading and thread synchronization, native method linking, and error and exception management.

Summary

Computers run programs; programs implement algorithms. As a programmer you need to be aware of development issues regarding your computer system and the processor it is based on.

A computer system contains a processor, I/O devices, and supporting operating system software. The processor is the heart of the computer system.

Programs can be viewed from two perspectives: human and computer. From the human perspective, programs, are a high-level solution statement to a particular problem. Object-oriented languages like Java help humans model extremely complex problems algorithmically. Java programs can also be viewed as the interaction between objects in a problem domain.

To a computer, programs are a sequence of machine instructions and data located in main memory. Processors run programs by rapidly executing the processing cycle of fetch, decode, execute, and store. If a processor expects an instruction and gets garbage it is likely to lock up. Robust operating systems can mitigate this problem to a certain degree.

There are bad algorithms and good algorithms. Study from the pros and you will improve your code writing skills.

The Java HotSpot™ virtual machine is a program that runs Java programs. Java HotSpot virtual machines are targeted for specific hardware platforms. There are two versions of the HotSpot virtual machine: one that utilizes the client version of the adaptive compiler and one that utilizes the server version of the adaptive compiler. The differences between the client and server compilers lie in the types of optimizations performed on the bytecode sequences.

Skill-Building Exercises

1. **Research Sorting Algorithms:** The second sorting algorithm listed on page 86 gives the steps for a merge sort. Obtain a book on algorithms, look for some Java code that implements the merge sort algorithm, and compare it to Dumb Sort. What's the growth rate for a merge sort algorithm? How does it compare to Dumb Sort's growth rate?

2. **Research Sorting Algorithms:** Look for an example of a bubble sort algorithm. How does the bubble sort algorithm compare to Dumb Sort? What small changes can be made to Dumb Sort to improve its performance to that of bubble sort? What percentage of improvement is obtained by making the code changes? Will it make a difference for large lists of integers?

3. **Research The Java HotSpot Virtual Machine:** Visit Sun's Java website [java.sun.com] and research the HotSpot virtual machine architecture. Become familiar with all the services the HotSpot virtual machine provides.

Suggested Projects

1. **Research Computer System:** Research your computer system. List all its components including the type of processor. Go to the processor manufacturer's web site and download developer information for your systems processor. Look for a block diagram of the processor and determine how many registers it has and their sizes. How does it get instructions and data from memory? How does it decode the instructions and process data?

2. **Compare Different Processors:** Select two different microprocessors and compare them to each other. List the feature set of each and how the architecture of each implements the feature set.

Self-Test Questions

1. List at least five components of a typical computer system.

2. What device do the peripheral components of a computer system exist to support?

3. From what two perspectives can programs be viewed? How does each perspective differ from the other?

4. List and describe the four steps of the processing cycle?

5. State in your own words the definition of an algorithm.

6. How does a processor's architecture serve to implement its feature set?

7. How can programmers access a processor's feature set?

8. State the purpose of the Java HotSpot virtual machine.

9. What is the difference between the client and server versions of the HotSpot virtual machine compilers?

10. How is the HotSpot virtual machine different from the Java Classic VM and the Just-in-Time (JIT) compiler?

References

Motorola, Inc. *PowerPC 601 RISC Microprocessor User's Manual*

Motorola, Inc. *PowerPC 7400 RISC Microprocessor User's Manual*

Sedgewick, Robert. *Algorithms in C++*, Addison-Wesley Publishing Company, Reading Massachusetts, 1992, ISBN 0-201-51059-6.

Corman, Thomas, et. al. *Introduction To Algorithms*, The MIT Press, Cambridge, Massachusetts, 1990, ISBN 0-262-03141-8

Jon Meyer, et. al. *Java Virtual Machine*, O'Reilly & Associates, Inc. 1997. ISBN: 1-56592-194-1

Java 2 SDK Standard Documentation: http://java.sun.com/j2se/1.4.2/docs/index.html

The Java HotSpot Virtual Machine v1.4.1, Technical White Paper, September 2002

Notes