Define complex instruction set computer architecture

Stands for Complex Instruction Set Computing. This is a kind of microprocessor design. The CISC architecture contains a large number of computer instructions, ranging from very simple to very complex and specialized. Although the design was designed to calculate complex statements in the most efficient way, it was later determined that many small, short statements could calculate complex statements more efficiently. This led to a design called Reduced Instruction Set Computing (RISC), which is now the other important type of microprocessor architecture. Intel Pentium processors are mainly CISC-based, with some RISC devices built into them, while the PowerPC processors are fully RISC-based. Quote this definition: This page contains a technical definition of CISC. It explains in the calculation terminology what CISC means and is one of many hardware terms in the TechTerms dictionary. All definitions on the TechTerms website are technically correct, but also easy to understand. If you find this CISC definition helpful, you can reference it using the citation links above. If you think a term should be updated or added to the TechTerms dictionary, please send an email to TechTerms! Eric Conrad, ... Joshua Feldman, in CISSP Study Guide (Third Edition), 2016CISC (Complex Instruction Set Computer) and RISC (Reduced Instruction Set Computer) are two forms of CPU design. CISC uses a large set of complex machine language statements, while RISC uses a reduced set of simpler statements. The best way to design a CPU has been the subject of discussion: should the low-level commands be longer and more powerful, use fewer individual statements to perform a complex task (CISC), or should the commands be shorter and simpler, requiring more individual instructions to perform a complex task (RISC), but allowing fewer cycles per statement and more efficient code? There is no right answer: both approaches have advantages and disadvantages. ×86 CPUs (among many others) are CISC; ARM (used in many mobile phones and PDAs), PowerPC, Sparc and others are RISC. Marilyn Wolf, in Computers as Components (Third Edition), 2012In this section, we will consider some general concepts in computer architecture and programming, including some different styles of computer architecture and the type of assembly language. Before we look at the details of the microprocessor instruction sets, it is helpful to develop some basic terminology. We do this by checking a taxonomy of the basic ways in which we can connect a computer to can.von Neumann architecturesA block diagram for a computer type is shown in Figure 2.1. The computing system consists of a central processing unit (CPU) and a memory. The store contains both data and instructions and can be read or written when an address is specified. A computer whose memory contains both data and instructions is referred to as von Neumann. Referred. 2.1. An architectural computer by Neumann. The CPU has several internal registers that store internally used values. One of these registers is the program counter (PC), which holds the address of a statement in memory. The CPU retrieves the statement from memory, decodes the statement, and executes it. The program counter does not directly determine what the computer does next, but only indirectly by pointing to a statement in memory. By only changing the instructions, we can change what the CPU does. It is this separation of instruction memory from the CPU that distinguishes a stored program computer from a general finite-state computer. An alternative to the Neumann-style computer organization is Harvard architecture, which is almost as old as Neumann's architecture. As shown in Figure 2.2, a Harvard machine has separate storage for data and program. The program counter refers to the program memory, not to the data store. Therefore, it is more difficult to write self-modifying programs (programs that write data values and then use these values as instructions) on Harvard machines. Figure 2.2. A Harvard Architecture. Harvard architectures are commontoday for a very simple reason—separating program and data stores provides higher performance for digital signal processing. The processing of signals in real time burdens the data access system in two ways: first, large amounts of data flow through the CPU; and secondly, that data must be processed at exact intervals, not only when the CPU comes to it. Records that arrive continuously and periodically are called streaming data. Two storage sacans with separate ports provide higher memory bandwidth. Not being able to compete with the same port makes it easier to move the data at the right time. DSPs make up a large proportion of all microprocessors sold today, and most of them are Harvard architectures. A single example shows the importance of DSP: Most phone calls in the world go through at least two DSPs, one at each end of the phone call. Another axis on which we can organize computer architectures relates to their instructions and their execution. Many early computer architectures were what is now known as a complex command set computer (CISC). These machines provided a variety of instructions that can perform very complex tasks, such as string search. they also used a number of different instruction formats of different lengths. One of the advances in the development of high-performance microprocessors the concept of reduced teaching set computers (RISC). These computers tended to provide slightly fewer and simpler instructions. RISC computers typically use load/memory instruction sets—operations cannot be performed directly in storage locations, but only in registers. The instructions were also chosen, chosen, they could run efficiently in pipeline processors. Early RISC designs significantly outperformed CISC designs of that time. As it turns out, we can use RISC techniques to efficiently execute at least one common subset of CISC statement sets efficiently, narrowing the performance gap between RISC-like and CISC-like statement sets. Instruction set properties Beyond basic RISC/CISC characterization, we can classify computers according to multiple characteristics of their instruction sets. The computer's instruction set defines the interface between software modules and the underlying hardware. define the instructions for what the hardware will do under certain circumstances. Statements can have a variety of characteristics, including:•fixed versus variable length;•addressing modes;•numbers of operands;•types of operations supported. We often characterize architectures by their word length: 4-bit, 8-bit, 16-bit, 32-bit, and so on. In some cases, the length of a data word, statement, and address are the same. Especially for computers designed to work with smaller words, instructions and addresses can be longer than the basic data word. Little-endian vs. big-endianA subtle but important characterization of architectures is the way they number bits, bytes, and words. Cohen [Coh81] introduced the terms Little-Endian mode (with the byte of the lowest order, which is in the low-order bits of the word) and the big-endian mode (the byte of the lowest order stored in the highest bits of the word). We can also characterize processors by executing them, a separate request from the command set. A processor with a problem executes a statement at a time. Although it can have multiple statements at different stages of execution, only one can be in a particular phase of execution. Several other processor types allow instructions for multiple problems. A superscalar processor uses special logic to identify run-time statements that can run concurrently. A VLIW processor relies on the compiler to determine which combinations of statements can be executed lawfully together. Superscalar processors often consume too much energy and are too expensive for widespread use in embedded systems. VLIW processors are often used in high-performance embedded computing. The set of registers that are available for use by programs is called a programming model, also known as a programming model. (The CPU has many other registers that are used for internal operations and for are not available.) Architectures and implementationsThere can be several different implementations of an architecture. In fact, the architecture definition is used to define the characteristics that must apply to all implementations and which can vary from implementation to implementation. Different CPUs can have different clock rates, different cache cache Changes to bus or interrupt lines and many other changes that can make one CPU model more attractive to one application than another. The CPU is only part of a complete computer system. In addition to storage, we also need I/O devices to create a useful system. We can build a computer from several different chips, but many useful computer systems come on a single chip. A microcontroller is a form of a single-chip computer that contains a processor, memory, and I/O devices. The term microcontroller is typically used to obtain a computer system chip with a relatively small CPU and one that contains read-only memory for program memory. A system-on-chip usually refers to a larger processor that contains on-chip RAM, which is usually supplemented by an off-chip memory. Figure 2.3 shows a fragment of the ARM assembly code to remind us of the basic functions of assembly languages. Assembly languages typically have the same basic functions:•a statement appears per row;•Labels that give memory locations names start in the first column;•Statements must start in the second column or after distinguishing labels;•Comments are executed from a specific comment character (; in the case of ARM) to the end of the line. Figure 2.3. An example of the ARM assembly language. The assembly language follows this relatively structured form to make it easier for the assembler to analyze the program and consider most aspects of the program line by line. (It should be remembered that early assemblers were written in assembly language to fit into a very small amount of memory. These early restrictions have traditionally translated into modern assembly languages.) Figure 2.4 shows the format of an ARM data processing instruction, such as an ADD. For the instructionFigure 2.4. Format of an ARM data processing instruction. ADDGT r0,r3,#5the cond field would be set according to the GT condition (1100), the opcode field would be set to the binary code for the ADD statement (0100), the first operand register Rn would be set to 3 to represent r3, the target register Rd would be set to 0 for r0 and the Operand 2 field to the immediate value of 5. Assemblers also need to provide some pseudo-ops to help programmers create complete assembly language programs. An example of a pseudo-op is an example that allows data values to be loaded into storage locations. This allows, for example, constants to be set into memory. An example of a memory allocation pseudo-op for ARM is:BIGBLOCK % 10The ARM % pseudo-op block of memory to the size specified by the operand and initializes these locations to zero. CPUs can run programs faster if they can execute more than one statement at a time. if the operands of a statement depend on the results of a previous statement the CPU cannot start the new statement until the previous statement statement However, neighboring statements cannot be directly dependent on each other. In this case, the CPU can run multiple at the same time. Various techniques have been developed to parallelize execution. Desktop and laptop computers often use super-scalar execution. A superscalar processor scans the program during execution to find sets of instructions that can run together. Digital signal processing systems tend to use very long VLIW (Instruction Word) processors. These processors rely on the compiler to identify statements that can run in parallel. Superscalar processors can end up concurrency that VLIW processors can't—some statements may be independent in some situations and not in others. However, super-scalar processors are more expensive in terms of both cost and energy consumption. Because it is relatively easy to extract concurrency from many DSP applications, the efficiency of VLIW processors can be more easily utilized by digital signal processing software. In modern terminology, a set of instructions is bundled into a VLIW package, which is a set of statements that can be executed together. Execution of the next package will not start until all statements in the current package have been executed. The compiler identifies packages by parsing the program to determine statements that can always be executed together. Inter-instruction dependenciesTo understand the parallel execution, let's first understand what limits statements from parallel execution. A data dependency is a relationship between the data that is operated by statements. In the example of Figure 2.5, the first statement writes to r0, while the second statement reads from it. Therefore, the first statement must be completed before the second statement can add it. The data dependency diagram shows the order in which these operations must be performed. Figure 2.5. Data dependencies and order of instruction execution. Branches can also introduce control dependencies. Consider this simple branch bnz r3,foo add r0,r1,r2foo: ... The add statement is executed only if the branch statement that precedes it does not accept its branch. Concurrency possibilities arise because many combinations of statements do not introduce data or control dependencies. The natural grouping of mappings in the source code indicates some possibilities of concurrency that can also be influenced by the use of registers by the object code. Take the example of Figure 2.6. Although these instructions input registers, the result of one statement does not affect the result of the other statements. Figure 2.6. Instructions without data dependencies. VLIW processors examine inter-statement dependencies only within a command package. You rely on the compiler to provide the necessary dependencies and into a package to avoid combinations of statements that cannot run correctly in a package. Super-scalar processors, on the other hand, use hardware to analyze the instruction stream and determine dependencies that need to be followed. VLIW and Embedded ComputingA number of different processors have implemented VLIW execution modes and these processors have been used in many embedded computing systems. Because the processor does not need to analyze data dependencies at run time, VLIW processors are smaller and consume less power than super-scalar processors. VLIW is very suitable for many signal processing and multimedia applications. For example, cellular base stations must perform the same processing on many parallel data streams. Channel processing can be easily mapped to vLIW processors because there are no data dependencies between the different signal channels. Enrico Perla, Massimiliano Oldani, in A Guide to Kernel Exploitation, 2011The role of the CPU is extremely simple: execute instructions. All statements that a CPU can execute include the statement set of the architecture. At least a typical instruction set provides instructions for arithmetic and logical operations (add, sub, or, and, etc.), control flow (jump/branch, call, int, etc.) and memory manipulation (load, store, push, pop, etc.). Because access to memory is usually a slow process (compared to the speed at which the CPU can crank up instructions), the CPU has a number of local, faster registers. These registers can be used to store temporary values (general registers) or to retain relevant control over information and data structures (special registers). CPU statements typically work on registers.Computer architectures are divided into two main families: RISC (Reduced Instruction Set Computer), which focuses on simple, fixed size statements that can be executed in a clock cycle; and CISC (Complex Instruction Set Computer), which contains instructions of different sizes that perform multiple operations and can perform more than one cycle. We can further distinguish the two depending on how they access memory: RISC architectures require that memory access be performed using either a load statement (copy from memory) or a memory statement, while CISC architectures may have a single statement to access the memory, such as performing an arithmetic operation on its contents. For this reason, RISC architectures are typically referred to as load store architectures. RISC architectures, in addition to load, storage, and some execute all instructions exclusively for registers. Today, the distinction between RISC and CISC is blurred, and many of the problems of the past have less impact (for example, binary size). For example, all current x86 processors decode complex statements in micro-ops, which are then which is pretty much an internal RISC core. The CPU retrieves the statements to be executed from memory, reads a stream of bytes, and decodes it according to the instruction set. A special register, typically referred to as an instruction pointer (IP) or program counter (PC), tracks which statement is executed. As we explained in Chapter 2, a system can be equipped with a single CPU, in which case it is called a Uniprocessor (UP), or with multiple CPUs, in which it is called a symmetric Multiprocessing (SMP) system. B-SMP systems are more complex to handle for an operating system itself, as real concurrent execution is now taking place. From the attacker's point of view, however, SMP systems open up more possibilities, especially when it comes to winning race conditions, as we will discuss later in this chapter. In Power and Performance, 2015 processor architectures are classified as either RISC (Reduced Instruction Set Computer) or Complex Instruction Set Computer (CISC). The difference between the two classifications is that RISC architectures have a small number of simple general statements, each performing a single operation and essentially providing the basic building blocks for calculations. CISC architectures, on the other hand, have a large number of more complex statements, each of which can perform multiple internal operations. For example, consider performing an arithmetic operation on a value in memory. In a RISC architecture, the corresponding arithmetic statement would only be able to work in a register. Therefore, a load instruction is issued before the process begins to retrieve the value from memory and store it in a register. Once this is complete, the operation is performed, with the result stored in a register. Finally, a memory statement is output to transfer the result back to memory. On the other hand, the statement of the arithmetic operation for a CISC architecture would accept a memory operand. Assuming that the memory operand is the target operand of the statement, this form of the statement would automatically retrieve the value from memory, perform the operation, and then pass the result back to the store in a statement. As a result, CISC architectures are often able to execute an algorithm in fewer statements than a RISC architecture because a CISC statement can perform the corresponding work of multiple RISC statements. On the other hand, RISC architectures are often less complex due to the simplified nature of their instructions and therefore require less silicon. In addition, due to the logical separation of different instructions for specific tasks, it is able to plan and execute statements with a finer granularity than CISC architectures. The x86 processor family is as CISC, because x86 statements can perform multiple internal operations. Starting with the Pentium Pro, Intel Architecture is actually a hybrid approach between the two. The command set is not changed so that x86 statements are still CISC, but the front end of the processor translates each statement into one or more micro-ops, typically referred to as 'ops or sometimes just uops. These ops are very similar to RISC statements, each specialized for a particular task. Consider the previous example of how CISC and RISC architectures process an arithmetic operation. The x86 statement set continues to support memory operands for this arithmetic statement so that it is displayed to the als CISC programmer. However, the front end can decode this single statement into three ops. The first, a load op, may be responsible for loading the content described by the storage operator. The second op would then be responsible for carrying out the actual operation. The third op would then be responsible for transferring the result back to memory. This hybrid approach gives Intel Architectures the benefits of both approaches. Because memory access can be expensive, fewer statements are retrieved. The CISC nature of the x86 statement set can be considered opcode compression, which improves the bandwidth of the command retrieval. At the same time, the execution pipeline can be more agile and flexible by splitting these complex statements into smaller 'ops, as described in Section 2.2.3. The cost of this approach is a more complicated front end that requires logic for decoding statements in 'ops. In general, these costs are negligible compared to the performance improvement achieved. Caesar Wu, Rajkumar Buyya, in Cloud Data Centers and Cost Modeling, 2015In this chapter we focused on the server. We not only discussed x86 (or CISC) servers, but also examined the details of RISC servers, with a special focus on the Oracle/Sun SPARC server. From a computer development perspective, we can see why the client/server architecture has become the mainstream computing architecture in the data center:1.The Internet has become widespread and is integrated into our daily lives. The number of hosts has increased exponentially or by 900% over the past 20 years, or so.2.In unlike a mainframe, the client/server architecture is very flexible and much easier to deploy.3.The cost of server deployment is only a fraction of the cost of a mainframe. We also presented brief information about the major vendors that provide these servers:•X86 (CISC) These are often produced by two major vendors, Intel and AMD.•RISC (SPARC) processors: these are mainly provided by Oracle/Sun and Fujitsu. We wanted to create the basis for our cost modelling because Units and terms such as sockets, cores, methods, domains, and multithreading are the physical basis for measuring the cost of a cloud infrastructure. As Extremetech has pointed out, the Intel x86 chip has taken over not only the PC/workstation market over the past 30 years, but also the server market in data centers. Compared to RISC servers, the x86 or CISC server has gradually become the dominant computer in the server market, and RISC servers have lost ground. The landscape of the server market has changed considerably since 2005. Cisco has been gaining momentum in the market share of x86 Server (Blade Server) since 2009. Based on IDC data from the second quarter of 2013, traditional server providers are losing x86 server market shares, and others (including some Chinese vendors such as Huawei and Lenovo) have gained a significant portion of the server market (40% in volume and 21% in revenue). This has shown that x86 servers have become a standard product as many servers are OEM products made in China. This could be one of the factors influencing many capex investment decision-makers. We will continue this discussion in later chapters. Although large traditional server vendors (such as HP, IBM, and Dell) are losing market share, they still hold a market share of more than 50% in volume and 70% of revenue (for all types of servers, including mainframe, x86, RISC, and EPIC). After discussing the details of x86 servers or processors, we went to the physical installation of servers. Traditionally, there were only two types of server styles: tower and rack servers (or pizza boxes). During the dotcom boom era between the late 1990s and early 2000s, people began developing blade servers to deploy servers on a large scale while saving physical space and cabling. Incrementally, the rack server would be a better solution for TCO/ROI. In later chapters, we will explain further details. In the last part of this chapter, we presented the details of the RISC servers. We explained what a RISC server is. And what are the differences between x86 (CISC) and RISC servers. We also looked at why the RISC server is losing market share to x86? In particular, we focused on Oracle/Sun SPARC servers. We explained the difference between M-Series and T-Series SPARC servers. We also listed both The M-Series server configurations and the T-Series SPARC server configurations, as well as the prices for outdated and current models. At the end of this chapter, we briefly looked at logical SPARC domains (LDoMs) or the VM Manager for SPARC to explain how the number of VMs per SPARC server can be accepted. James D. Broesch, in Digital Signal Processing, 2009A conventional microprocessor often uses one of Neumann's architecture, which means that there is only one common system bus that can be used to transmit instructions and between the external memory chips and the processor (see Figure 8.2). The system bus consists of the three subbuses: the data bus, the address bus and the control bus. In many cases, the same system bus is also used for I/O operations. In signal processing applications, this single bus is a bottleneck. For example, the execution of the 10-tap FIR filter (equation 8.2) requires at least 60 bus cycles for instruction callers and 40 bus cycles for data and coefficient transmissions, a total of about 100 bus cycles. Therefore, even with a fast processor, the speed of the bus cycle will be a limiting factor. Figure 8.2. of Neumann Share Architecture, Program Code and Data SharingA way to solve this problem is the introduction of pipelining techniques, which means that an execution unit (EU) and a bus unit (BU) work on the processor chip at the same time. While a statement is executed in the EU, the next statement is retrieved from memory by the BU and placed in a statement queue, which enters the statement decoder. This eliminates idle bus cycles. However, if a Jump statement occurs in the program, the statement queue must be restarted, resulting in a delay. Another improvement is to add cache memory on the processor chip. A limited block (a few thousand words) of the program code is read into the fast internal cache memory. This allows statements to be retrieved from the internal cache while transferring data through the external system bus. This approach can be very efficient in signal processing applications because in many cases the entire program can fit into the cache and no reloading is required. The execution unit in a conventional microprocessor can consist of an arithmetic logic unit (ALU), a multiplier, a shift lever, a floating-point unit (FPU), and some data and flag registers. The ALU often processes the complement arithmetic of 2, and the FPU uses some standard floating point formats from the Institute of Electrical and Electronics Engineers (IEEE). The formatted binary fraction format discussed later in this chapter is commonly used in signal processing applications, but is not supported by all-purpose microprocessors. In addition to program counters (PC) and stack pointer (SP), the address inheit (AU) of a conventional microprocessor can contain a number of address and segment registers. There may also be an ALU to calculate addresses that are used in complicated addressing modes and/or when handling virtual memory functions. The instructional repertoire of many supports quite exotic addressing modes that are rarely used in signal processing algorithms. On the other hand, instructions for the efficient handling of such things as delay lines or circular buffers are rare. MAC operation often requires a set of computer instructions, and loop counters must be Use of general data registers. In addition, there are instructions for operating systems and multitasking processing under higher processors. These instructions are often of very limited interest to signal processing applications. Most common processors today are of the Type Complex Instruction Set Computer (CISC), i.e. instructions can occupy more than one memory word and therefore require more than 1 bus cycle to retrieve. In addition, these instructions often require more than 1 machine cycle to perform. In many cases, reduced instruction set processors (RLSCs) can work better in signal processing applications. In a RISC processor, no statement occupies more than one memory word. it can be retrieved in 1 bus cycle and is executed in 1 machine cycle. On the other hand, many RISC statements may be required to perform the same function as a CISC statement, but in the RISC case, you can only get the required complexity when needed. Retrieving analog signals in and out of an all-purpose microprocessor often requires a lot of external hardware. Some microcontrollers have built-in A/D and D/A converters, but in most cases these converters have only 8 or 12 bit resolutions, which is not enough in many applications. Sometimes these converters are also quite slow. Even if there are good built-in converters, external sample-and-hold circuits (S/H) as well as (analog) anti-aliasing and reconstruction filters are always required. Some microprocessors have built-in high-speed serial communication circuits, serial peripheral interfaces (SPI) or I2C ™. In such cases, we still need external converters, but the interface will be easier than using the conventional approach of connecting the converters parallel to the system bus. Parallel communication will of course be faster, but the required circuits will be more complicated and we will steal capacity from a common single system bus. The interrupt devices found on many general-purpose processors are in many cases overkill for signal processing systems. In this type of real-time application, timing is crucial and synchronous programming is preferred. The number of asynchronous events, such as interrupts, is kept to a minimum. Digital signal processing systems that use more than a few interrupt sources are rare. A single interrupt source (such as timing or sampling rate) or none is common. ANDREW N. SLOSS, ... CHRIS WRIGHT, in ARM System Developer's Guide, 2004The ARM core uses a RISC architecture. RISC is a design philosophy that aims to provide simple but powerful instructions that can be cycle at high clock speed. The RISC philosophy focuses on reducing the complexity of the instructions executed by the hardware because it is easier to provide more flexibility and intelligence in the software than in hardware. As a result, a RISC RISC is makes higher demands on the compiler. In contrast, the traditional complex Instruction Set Computer (CISC) relies more on the hardware for instruction alscription functionality, and therefore the CISC statements are more complicated. Figure 1.1 shows these major differences. Figure 1.1. CISC vs. RISC. CISC emphasizes hardware complexity. RISC emphasizes the complexity of the compiler. The RISC philosophy is implemented with four main design rules:1.Instructions– RISC processors have a reduced number of classrooms. These classes provide simple operations that can be performed in a cycle at a time. The compiler or programmer synthesizes complicated operations (such as a split operation) by combining several simple statements. Each statement is a fixed length so that the pipeline can retrieve future statements before decoding the current statement. In contrast, the instructions for CISC processors are often variable-sized and require many cycles to run.2.Pipelines — The processing of statements is divided into smaller units that can be executed in parallel by pipelines. Ideally, the pipeline advances one step in each cycle to achieve maximum throughput. Statements can be decoded in a pipeline phase. There is no need for a statement to be executed by a mini-program called Microcode, as with CISC processors.3.Registers – RISC machines have a large general set of registers. Each register can contain either data or an address. Registers act as a fast local storage for all data processing operations. In contrast, CISC processors have dedicated registers for specific purposes.4.Load store architecture—The processor works with data stored in registers. Separate loading and storage instructions transfer data between the registry and the external storage. Storage access is expensive, so separating storage access from data processing is an advantage because you can use data items that reside in the register bank multiple times without requiring multiple storage accesses. In contrast, the data processing operations of a CISC design can respond directly to the memory. These design rules allow a RISC processor to be simpler so that the core can work at higher clock frequencies. In contrast, traditional CISC processors are more complex and work at lower clock frequencies. Over the course of two decades, however, the distinction between RISC and CISC has become blurred as CISC processors have implemented more RISC concepts. Max Robert, Bruce A. Fette, in Cognitive Radio (Second Edition), 2009General Purpose processors are the target processors that probably come to mind first for anyone who writes a computer program. GPPs are the processors that power desktop computers and are at the center of the computer revolution that began in the 1970s. The landscape of microprocessor design is characterized by a large number of devices from a variety of Although these different processors are unique, they have some similarities: a generic instruction set, a statement sequencer, and a memory management unit (MMU). There are two common types of command sets: (1) machines with fairly broad instruction sets called complex command set computers (CISCs); and (2) machines with a narrow set of commands, the so-called reduced command set computers (RISCs). In general, the CISC statements give the assembly programmer powerful instructions that address an efficient implementation of certain common software functions. RISC statement sets are narrower, but they are designed to generate efficient compiler code. The differences between DEM CISC and RISC are arbitrary, and both processor styles converge to a single instruction set type. Regardless of whether the machine is CISC or RISC, both share a generic character of their instructions. This includes statements that perform multiplication, addition, or storage, but these statement sets are not tailored to a specific application type. In the context of CR, the application we are most interested in is signal processing. The other key aspect of the GPP is the use of an MMU. Because GPPs are designed for generic applications, they are typically paired with an operating system. This operating system creates a layer of abstraction over hardware, allowing the development of applications with little or no knowledge of the underlying hardware. Memory management is a tedious and error-prone process, and in a system that runs multiple applications, memory management includes paging memory, distributed programming, and data storage in different blocks of memory. An MMU allows the developer to see a contiguous storage set, even if the underlying storage structure is fragmented or otherwise too difficult to control (especially in a multitasking system that runs continuously over a long period of time). Given the generic nature of the applications running on a GPP, an MMU is critical because it allows for easy mixing of different applications without any special care on the part of the developer.