CHAPTER ONE

Introduction to Computer Architecture

1.1 What is Computer Architecture?

A simplistic view of computer architecture is that it is the interface between software and a computer's hardware: in other words, the view best seen by the assembly language programmer. Another view of computer architecture emerges from an analogy with building architecture. An architect of buildings takes materials such as brick, wood, stone, and steel and components such as electrical and mechanical systems and combines them so that they function as a physical structure that satisfies a human need. Computer architects take components such as logic gates and memory cells and combine them so that they function as a system that satisfies the needs of its users.

An understanding of a processor's architecture, however, benefits far more than just the compiler or driver designers. Anyone interested in performance, be it speed or functionality, can benefit from an appreciation of the features of a processor. Let's return to our analogy with building architecture. An improperly designed building may not be energy efficient, big enough, or adequate for its users due to a lack of features such as bathrooms or design issues such as poor traffic flow. A better architectural design could, however, produce a building that exceeds the user's expectations. Similarly, advances in computer architecture may give the user better performance be it processing speed, cost, reliability, or capacity.

However, in order to take advantage of a building's features, users may need to be aware of and know how to use characteristics that the architects have included in their creations. For example, a homeowner may never have figured out that the switch in their master bedroom, which seems to do nothing, in reality turns on an attic fan that could lower their energy costs. Similarly, a multiprocessor system can only improve an application's performance if the application is capable of using multiple processors simultaneously. If a software engineer is not aware of how the processor's architecture affects his or her code, he or she may end up creating software that performs worse when run on newer architectures.

1.2 Difference between Organization and Architecture

Typically, computing professionals are not concerned with the low-level details of computer organization such as how a computer's circuits are organized and how the computer performs arithmetic. These concerns are usually transparent to them. It shouldn't matter for example what the gate-level design of the processor's integer divide circuit looks like, only that it returns the correct result of a division.

Architecture, however, is concerned with those attributes of a computer that are visible to its programmers, at least at the assembly language level. These attributes may affect high-level language programmers by forcing them to select specific design methodologies or at least configure a compiler to take advantage of the target architecture. Integrated development environments (IDE), usually support a configuration item in a project's properties that specifies the target platform. This is important as different platforms have different architectural issues that the compiler must address.

In general, there are three main areas of concern when it comes to computer architecture: design of the system, design of the interconnecting mechanisms, and design of the instruction set and addressing modes.

System design is a top-down view of the system that depicts its subsystems and interconnections. This view includes a partitioning of the system into processing components such as the main CPU and/or parallel CPUs, direct memory access controllers, memory controllers, and input/output controllers. It also defines the interconnecting buses that the processing components need in order to exchange information.

Instruction set design pertains to the development of the system's machine language. This includes the opcodes (instructions), permissible operands (the data being operated on), operand addressing modes, processor registers, and the formats of the address and data. In order to implement a machine, all of these elements must be precisely defined.

Between the system design and the design of the instruction set is a detailed definition of how the subsystems are connected and how they interact. Issues such as timing, arbitration, data integrity, and latency are all important to the implementation of this level. This includes, for example, a multiprocessor system's strategy for maintaining data integrity when more than one processor is modifying shared data elements.

1.3 An Introduction to Performance

The concept of performance will be addressed later in this book. For now, it is important to understand that performance is more than measuring the frequency of a processor's system clock. Issues such as data throughput, reliability, or the ability to handle concurrent loads can affect whether a processor can satisfy a user's needs.

An analogy with travel planning may be helpful for illustrating how a range of concerns can come into play when trying to decide on a "best" strategy for achieving a goal. Let's assume that someone wants to travel from Washington, D.C., to New York City, a trip of 230 miles. There are a number of ways to make this trip. A flight from Reagan National Airport to LaGuardia Airport would take an hour and fifteen minutes. It could cost anywhere from $300 to $800 depending on when the reservations are made and the level of service desired.

There are, however, issues that might not be immediately apparent with flight. For example, at least an hour will be needed for check-in and security. More time will be required for travel to and from each airport. When traveling by plane, the traveler must also abide by the airline's schedule. Any traveling done while in New York City would have to be done by taxi or rental car. Parking fees at Reagan National Airport are $20 per day. Airlines also limit the amount of luggage travelers may bring. Airline travel also creates a risk of luggage being lost or damaged by the airline. This could be of particular concern if the traveler was going to New York City to run a booth at a trade show and needed to bring displays and merchandise. Finally, there may be issues with the traveler such as an extreme fear of flying or a physical disability that makes traveling difficult if not impossible.

Another option might be to drive from Washington, D.C., to New York City, a 460-mile roundtrip that without traffic will take about four hours and fifteen minutes each way. The benefits of driving include no scheduling restrictions, lower price of $270 (based on the U.S. Government Reimbursement Rate as of August 1, 2008 of $0.585 per mile), the use of the car while in New York City, significantly lower restrictions on luggage, and two or three passengers would be able to go along with minimal additional cost.

The drawbacks, however, might be prohibitive. Traffic might easily add an hour to the trip in addition to creating stress for the driver. Figuring out what to do with a vehicle while in New York City might

also be a problem. Some hotels offer parking, but in downtown, the rates usually climb well above $50 per night.

To relieve the stress of driving, our traveler could take a bus or a train. Traveling by bus would take approximately four and a half hours at a cost of around $50. Traveling by train from Union Station in Washington, D.C., to Penn Station in New York City would take about three and a half hours at a cost of around $80. Not too bad, but once again, there will be restrictions on scheduling and luggage, and there will most likely be a fee for parking a car at the bus or train station. Physical disabilities may also prohibit these options.

In computer architecture, like travel, there are often are many factors to consider when measuring a computer's performance. Let's assume a remote data collection site uses a cell link to transmit collected data back to a central station. Each time a connection is made, about 250 bytes of data are transmitted. The major concern is connection time over the cellular link. What modem standard should the designers select? Since the duration of the connection is a factor, then the fastest modem should be selected, correct? Not necessarily.

Table 1-1 shows the approximate connection durations for different types of modems transmitting 250 bytes. This includes the time it takes for the receiving end to synchronize to the calling modem's data rate, a period typically referred to as negotiation.

Table 1-1 Approximate Modem Connection Times

Modem Data Rate (bits/second)	Approximate Negotiation Time	Approximate Time to Transmit 250 Bytes	Approximate Connection Time
1,200	3 seconds	2 seconds	5 seconds
2,400	5 seconds	1 seconds	6 seconds
14,400	8 seconds	0.17 seconds	8.17 seconds
28,800	11 seconds	negligible	11 seconds
56,000	20 seconds	negligible	20 seconds

Notice how the slower modems connect and transmit the data much quicker than the higher speed modems. This is due to the quicker negotiation times. Some low speed modems can have their negotiation times reduced even further if the system is set up where both ends of

the connection are predefined to expect a specific type of modem. This is called Quick Connect.

If a system such as this makes 20 connections each day for a year, the 28,800 bps modem would require 20 connections/day × 365 days/year × 11 seconds/connection = 80,300 seconds/year while the 1,200 bps modem would require 20 connections/day × 365 days/year × 5 seconds/connection = 36,500 seconds/year. That's a difference of 43,800 seconds/year or around 12 hours of connection time.

Later in this text, we will discuss how performance parameters affect decisions about computer architecture. This simple example illustrates how there are tradeoffs between different methods of improving performance. In addition, there is no single best way to measure performance.

Exercise

Waiting in the drive through lane of a fast food restaurant can be frustrating. Many fast food companies are obsessed with developing ways to shave seconds from the time a customer arrives in the drive-thru lane to when they receive their order. Competition is fierce for the annual "The Best in Drive-thru" report put together by QSR magazine. (www.qsrmagazine.com)

From your experience, identify some of the methods for optimizing a fast food drive thru lane. Be sure to address issues such as the menus (both content and presentation), the steps patrons must go through from the moment they arrive to the moment they leave, methods of payment, and how the food is cooked.

Although this exercise may seem ridiculous, especially when presented in a textbook about computer architecture, a number of the methods for improving the performance of a fast food drive-thru have theoretical equivalents when it comes to improving processor performance. The following discussion presents some of the drive-thru concepts you might have addressed in the previous exercise and compares them to the execution of an assembly language instruction.

Multiple lanes: Adding a second, parallel drive-thru lane has the potential to double the number of customers serviced at the restaurant. The time it takes between a customer placing an order and receiving it may not be reduced, but it could increase the number of customers a

restaurant could serve thereby reducing the amount of time the customers would have to wait in line before placing their order. There are potential drawbacks though. If the restaurant does not have the capacity to service the two lines, one line may end up waiting for the other to finish using those resources. In addition, the second lane may require physical space not available beside the store forcing a reduction in available parking or causing additional congestion as people contend for which line looks the shortest or quickest.

Servicing customers in parallel is equivalent to executing instructions in parallel, a common practice in computer architecture. There are a number of ways that instructions can be executed in parallel: multiple processors executing different applications (symmetric multiprocessors and clusters), a single application being executed on multiple data streams on multiple processors (vector computing), and even non-dependent instructions from a single application being executed in parallel across multiple resources of a single processor (super-scalar). As with the restaurant, however, limited resources, a lack of space (transistors) on the silicon, or not enough instructions capable of being executed in parallel may prohibit realizing scalable performance improvements.

Combo meals: People *tend* to eat fast food meals with a sandwich, a side order, and a drink, so restaurants create packages or combos which are priced slightly lower than if the three items were purchased individually. This helps to speed things up in a drive-thru since ordering a, "number 1 combo with a Sprite," is a great deal smoother than asking for, "a double cheeseburger with a medium order of fries and a medium Sprite."

In computer architecture, CPU designers add flexibility to the design of an instruction set by placing a hardwired interpreter called microcode between the assembly language instructions and rudimentary processes of the CPU. When the designer wishes to add another assembly language instruction, he or she simply needs to create the set of steps the processor must execute and store them as microcode. It is equivalent to having each assembly language instruction act as a tiny program of its own.

Redesigning menuboards: There are a number of ways to improve drive-thru menuboards including using visual representations, creating more combo offerings, and reducing the number of individual items available to the drive-thru customer. A full menu in the drive-thru

might be overwhelming to people who have difficulty making decisions. Simplifying menus or limiting menus to combos may help people make their selection quicker. In addition, by having fewer types of meals to prepare, the kitchen staff might be able to create meals quicker. Problems occur, however, when people become frustrated because they cannot get exactly what they want which may cause them to select a different restaurant in the future.

Simplifying the number of available assembly language instructions is also one way of improving processor performance. Reduced instruction set computing (RISC) takes advantage of the fact that simpler instructions and addressing modes allow processor designers to create faster architectures. Just like the simpler menuboards, however, this may aggravate programmers by forcing them to use non-optimal instruction sequences.

Sub-dividing the process: Imagine that each of the three steps in the drive-thru process (placing an order, paying for it, and receiving it) took 40 seconds for a total of 120 seconds or two minutes. If the drive-thru had only a single window to handle all three of these steps, then the throughput of the drive-thru would be at most one customer every two minutes.

Most drive-thrus dedicate a "window" to each stage of the process. While one customer is receiving their order, the one behind them is paying for theirs while the one behind them is placing an order. It still takes two minutes for a single customer to be helped, but with three customers being helped in parallel, one customer is sent away from the pick-up window every 40 seconds or two-thirds of a minute.

It is important to note that like the parallel lanes, this system doesn't speed up the customer's experience – it improves the overall throughput. In other words, instead of servicing up to 30 customers every hour (60 minutes/hour ÷ 2 minutes/customer), a drive-thru can service up to 90 customers (60 minutes/hour ÷ 0.67 minutes/customer) for a speed up of three. If the drive-thru process could be divided into more stages, the realized speed up might be even greater.

There are, however, a few problems. Depending on the customer, different stages can take different amounts of time thereby holding up customers who have completed their stage and are waiting for the next. In addition, a non-negligible time is required to drive from one stage to the next.

Processors use a comparable strategy for increasing throughput. This strategy, referred to as pipelining, divides the work of executing an instruction into an assembly-line-like series of stages. These stages are then executed in parallel with other instructions in different stages. In general, it still takes as long to execute a single instruction, but more instructions can be executed during a specific period of time.

The drawbacks of pipelining are much like those of the drive-thru. Delays are encountered when an instruction moves from one stage to the next. In addition, not all instructions are created equal. Some instructions will stall the pipeline because they require a longer delay in some stages while others might monopolize processor resources. Still other instructions are "conditional" and cannot be processed until the outcome of the previous instruction has been determined.

Pre-ordering: Made-to-order restaurants such as pizza places usually allow pre-ordering over the phone or Internet, and although this doesn't apply to fast food per se, it does have an equivalent in computer architecture.

For an instruction to be executed, it must be retrieved from memory and decoded to determine which of a processor’s circuits to activate. A cache can be used to store pre-fetched instructions along with the instruction's decoding information. This allows some instructions to skip stages of execution. It is also beneficial in some cases to maintain a history of how the instruction behaved during previous executions so a prediction can be made regarding its current behavior.

Remote call centers: Typically, on-site employees who take drive-thru orders use a head set so that they can simultaneously do other tasks such as fill drinks or take money. This might force a customer to wait for an employee to become available to take their order. It also makes it more difficult for the employee to be polite when taking the order. Off-location call centers provide fast-food restaurants with trained "order takers" who then send the order back to the restaurant for completion.

In an effort to improve the performance of processors, CPU designers make every effort to take tasks that were once the responsibility of the computer's main CPU and off-load them to other processors. Tasks might include handling block transfers of data, maintaining data coherence between multiple caches, processing video output, or performing complex mathematical instructions. By distributing mundane tasks to other processors, the main CPU is better able to focus on the execution needs of an application.

1.4 About this Book

Now we come to the purpose of this book. The original system design of the modern-day processor has not changed much since the early pioneers of computing first made their electromechanical machines in the 1930's and 1940's. What has happened is that like the restaurant owners who have been fiddling with the details of a drive-thru lane, computer architects have been obsessed with shaving nanoseconds from the execution of a stream of instructions. This book begins by presenting the general foundation of early architectures. It then examines ways in which these architectures have been tweaked in order to gain better performance and how some of those advances place special requirements on their applications. Along the way, there will be opportunities to demonstrate through both simulation and performance measurement how different architectural concepts affect the performance of a computer.

The reader should have a clear understanding of digital logic including the design of logical circuits, the use of Boolean algebra, and the concept of state machines and state machine design before beginning this study. An understanding of these topics will offer insight as to how architectural components are implemented in addition to revealing difficulties encountered when trying to realize certain architectures. The reader should also have an understanding of assembly language programming. This will provide a foundation for discussions on instruction set design, instruction execution, and the requirements of the architecture.

1.5 What's Next?

The purpose of this chapter was to define computer architecture and in general terms describe its importance and its aims. Chapter 2 will provide a foundation for a study of computer architecture by examining early architectures and how they affected modern processor design. This will be important when we begin addressing performance to see where bottlenecks exist and how to remove them or lessen their effects.

1.6 Problems

1. Brainstorm on the performance issues as they pertain to the applications of a banking database (customer accounts, on-line access, etc).

    - What items should we measure?
    - What are the units of these measures?
    - What applications rely on these measures?
    - What are the implications if these performance goals are not met?
    - What laboratory methods can we use to take these measures?

2. Repeat the previous problem for the application of an on-line multimedia server.
3. Explain why increasing a processor's throughput does not always require a reduction in the time it takes to execute a single instruction.
4. Describe the difference between a processor's response time and a processor's throughput.
5. Identify three reasons why the overlapping of instructions in a pipelined processor might not work for some instructions.
6. Identify three "tasks" that a processor might off-load to peripheral processors in an effort to improve its performance.