Name: Manuel Andre Ramiro
ID: UB5967SCN12283
Degree: Bachelor
School of Science and Engineering

Subject Resume

Introduction to Computing Systems Organization: From carpineli,John,ISBN:0201612534.

Course objective

Computer systems organization - General Hardware/software interfaces; Instruction set design; System architectures; System specification methodology

Introduction

This chapter describes the basic components that make up a computer system: the CPU, memory, I/O, and the bus that connects them. Although you can write software that is ignorant of these concepts, high performance software requires a complete understanding of this material. This chapter also discusses the 80x86 memory addressing modes and how you access memory data from your programs.
This chapter begins by discussing bus organization and memory organization. These two hardware components will probably have a bigger performance impact on your software than the CPU's speed. Understanding the organization of the system bus will allow you to design data structures and algorithms that operate at maximum speed. Similarly, knowing about memory performance characteristics, data locality, and cache operation can help you design software that runs as fast as possible. Of course, if you're not interested in writing code that runs as fast as possible, you can skip this

discussion; however, most people do care about speed at one point or another, so learning this information is useful.
With the generic hardware issues out of the way, this chapter then discusses the program-visible components of the memory architecture - specifically the 80x86 addressing modes and how a program can access memory. In addition to the addressing modes, this chapter introduces several new 80x86 instructions that are quite useful for manipulating memory. This chapter also presents several new HLA Standard Library calls you can use to allocate and deallocate memory.

Some might argue that this chapter gets too involved with computer architecture. They feel such material should appear in an architectural book, not an assembly language programming book. This couldn't be farther from the truth! Writing *good* assembly language programs requires a strong knowledge of the architecture. Hence the emphasis on computer architecture in this chapter.

HARDWARE

1. in order for a computer to make calculations and process jobs.it must have a designated location to store,read and write the information that it is being manipulated.the most integral part of a computer is internal memory is called *Primary memory* ,which has generally been in the form of Random Access Memory(commonly known as RAM)since 1968. when a computer is engaged in processing a task,data is stored in the computer´s primary memory.only holds the data for as long as the computer is powered up with a steady flow of electrical current. It is lost when the computer is shut down.

2. Another form of memory that is very important to the modern computer secondary memory.secondary memory is different from primary memory in the speed of access,the duration of the memory,and capacity.it is generally slower than primary memory,which makes it a poor choice for processing jobs.however,secondary memory is usually higher capacity and does not need a constant current running through it to retain its memory storage **hardware** range from hard-disk drives to floppy disk drives to flash drives,as well as to optical disk such as CD-ROM and DVDs.

3. The central Processing Unit (CPU) is also a very necessary component of **Computer hardware.**The CPU,known colloquially as computer´s brain,processor or central processor,conducts all the data calculation and process for the data held in the computer´s memory.

4. the CPU is housed in a silicon chi called a microprocessor.these components understand the basic computer instructions that are provided by the computer´s software,even though CPUs are often physically small and they contain millions of parts.

5. Storage devices (such as disk drives),display devices(such as computer monitor or screen),output devices(printers,speakers),input devices (keyboards,mice,joysticks,scanners) and other peripherals(modems and network cards) are all considered types of **hardware .**that is computer components that can actually be physically touched.

SOFTWARE

1. Software is a general term for all the computer instructions or data(usually in the form of programs or applications)that exist on a computer.programs are stored in memory and generally do not have a tangible aspect.software can range from computer games to word processors(applications software) to the computer´s operating system (OS or system software), which manages and controls all the other software on the system.

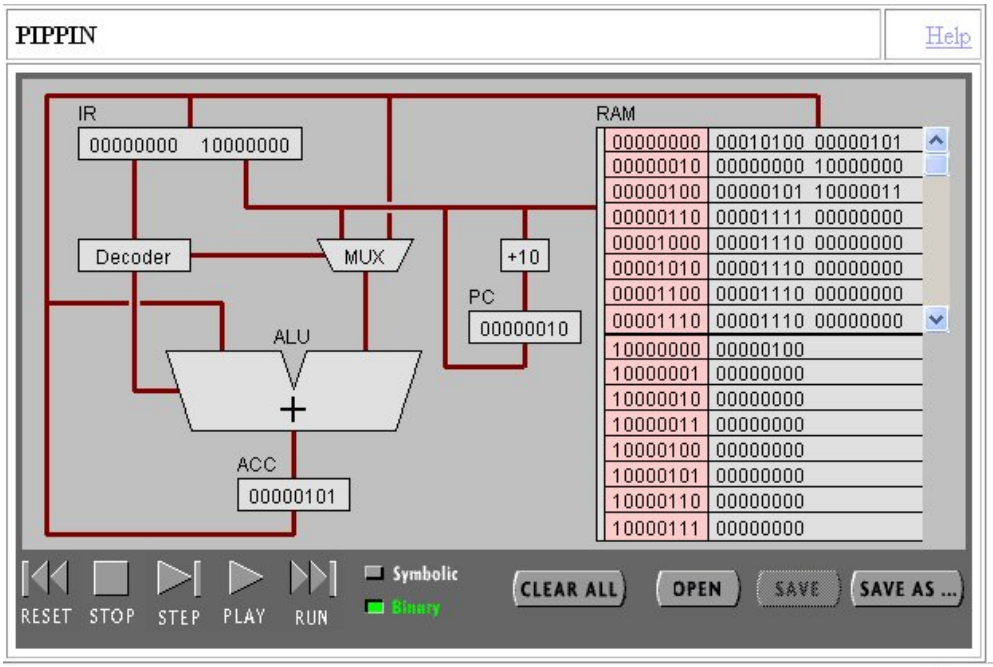
2. Operating systems have been evolving for many years and have become more and more advanced. The major operating systems competing for the personal computing market today ara various versions of Microsoft Windows (2000,NT,XP home,XP Pro) ,Apple Macintosh (OS 8,9,X,Tiger),and Linux that have their own individual strengths and weaknesses)

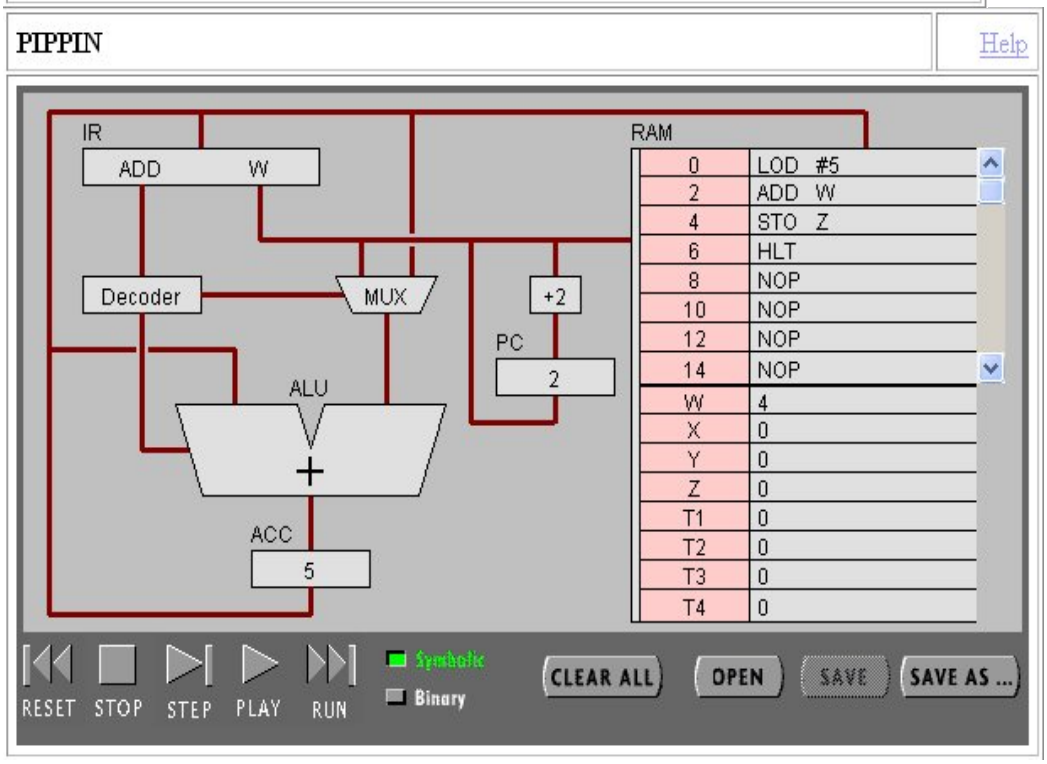
3. the main objective of this curriculum is to teach students how to create their own computer software.when creating computer software programs as students have already been doing, a language translator and compiler are required t convert the java code into software language that the CPU is able to understand.without the aid of these tools,it would be necessario to program computer entirely in binary,with 1´s and 0´s.image debugging code that looked like that!

4. Programs are stored in a binary format that is specific to the processor (or virtual machine, in the case of Java)
5. High-level language programs must be “compiled” into machine language that is targeted toward a specific processor or operating system.

Machine Language Instructions (continued)

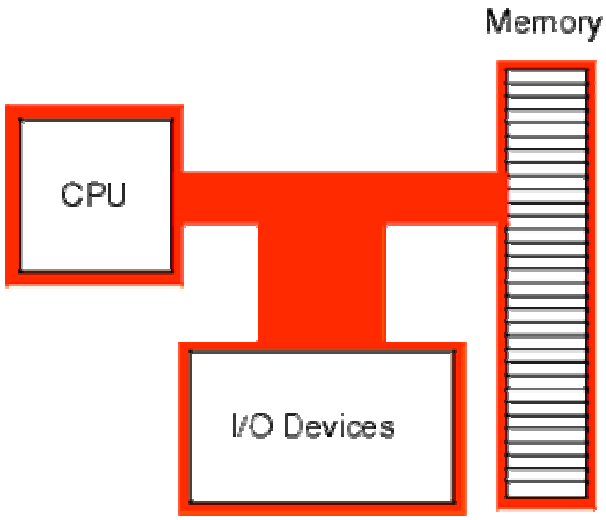
Assembly language-(Closer to Human Language)





1.1 The Basic System Components

The basic operational design of a computer system is called its *architecture*. John Von Neumann, a pioneer in computer design, is given credit for the architecture of most computers in use today. For example, the 80x86 family uses the *Von Neumann architecture* (VNA). A typical Von Neumann system has three major components: the *central processing unit* (or *CPU*), *memory,* and *input/output* (or *I/O*). The way a system designer combines these components impacts system performance .



In VNA machines, like the 80x86 family, the CPU is where all the action takes place. All computations occur inside the CPU. Data and machine instructions reside in memory until required by the CPU. To the CPU, most I/O devices look like memory because the CPU can store data to an output device and read data from an input device. The major difference between memory and I/O locations is the fact that I/O locations are generally associated with external devices in the outside world.

1.2.1 The System Bus

The *system bus* connects the various components of a VNA machine. The 80x86 family has three major busses: the *address* bus, the *data* bus, and the *control* bus. A bus is a collection of wires on which electrical signals pass between components in

the system. These busses vary from processor to processor. However, each bus carries comparable information on all processors; e.g., the data bus may have a different implementation on the 80386 than on the 8088, but both carry data between the processor, I/O, and memory.

A typical 80x86 system component uses *standard TTL logic levels*[1]. This means each wire on a bus uses a standard voltage level to represent zero and one[2]. We will always specify zero and one rather than the electrical levels because these levels vary on different processors (especially laptops).

1.2.1.1 The Data Bus

The 80x86 processors use the *data bus* to shuffle data between the various components in a computer system. The size of this bus varies widely in the 80x86 family. Indeed, this bus defines the "size" of the processor.

Every modern x86 CPU from the Pentium on up employs a 64-bit wide data bus. Some of the earlier processors used 8-bit, 16-bit, or 32-bit data busses, but such machines are sufficiently obsolete that we do not need to consider them here..

You'll often hear a processor called an *eight, 16, 32, or 64 bit processor.* While there is a mild controversy concerning the size of a processor, most people now agree that the minimum of either the number of data lines on the processor or the size of the largest general purpose integer register determines the processor size. The modern x86 CPUs all have 64-bit busses, but only provide 32-bit general purpose integer registers, so most people classify these devices as 32-bit processors.

Although the 80x86 family members with eight, 16, 32, and 64 bit data busses *can* process data up to the width of the bus, they can also access smaller memory units of eight, 16, or 32 bits. Therefore, anything you can do with a small data bus can be done with a larger data bus as well; the larger data bus, however, may access memory faster and can access larger chunks of data in one memory operation. You'll read about the exact nature of these memory accesses a little later.

1.2.1.2 The Address Bus

The data bus on an 80x86 family processor transfers information between a particular memory location or I/O device and the CPU. The only question is, "*Which memory location or I/O device?*" The address bus answers that question. To differentiate

memory locations and I/O devices, the system designer assigns a unique memory address to each memory element and I/O device. When the software wants to access some particular memory location or I/O device, it places the corresponding address on the address bus. Circuitry associated with the memory or I/O device recognizes this address and instructs the memory or I/O device to read the data from or place data on to the data bus. In either case, all other memory locations ignore the request. Only the device whose address matches the value on the address bus responds.

With a single address line, a processor could create exactly two unique addresses: zero and one. With *n* address lines, the processor can provide 2^n unique addresses (since there are 2^n unique values in an *n*-bit binary number). Therefore, the number of bits on the address bus will determine the *maximum* number of addressable memory and I/O locations. Early x86 processors, for example, provided only 20 bit address busses. Therefore, they could only access up to 1,048,576 (or 2^{20}) memory locations. Larger address busses can access more memory.

Table 12: 80x86 Family Address Bus Sizes

Processor	Address Bus Size	Max Addressable Memory	In English!
8088, 8086, 80186, 80188	20	1,048,576	One Megabyte
80286, 80386sx	24	16,777,216	Sixteen Megabytes
80386dx	32	4,294,976,296	Four Gigabytes
80486, Pentium	32	4,294,976,296	Four Gigabytes
Pentium Pro, II, III, IV	36	68,719,476,736	64 Gigabytes

Future 80x86 processors (e.g., the AMD "Hammer") will probably support 40, 48, and 64-bit address busses. The time is coming when most programmers will consider four gigabytes of storage to be too small, much like they consider one megabyte insufficient today. (There was a time when one megabyte was considered far more than anyone would ever need!).

1.2.1.3 The Control Bus

The control bus is an eclectic collection of signals that control how the processor communicates with the rest of the system. Consider for a moment the data bus. The CPU sends data to memory and receives data from memory on the data bus. This prompts the question, "Is it sending or receiving?" There are two lines on the control bus, *read* and *write*, which specify the direction of data flow. Other signals include system clocks, interrupt lines, status lines, and so on. The exact make up of the control bus varies among processors in the 80x86 family. However, some control lines are common to all processors and are worth a brief mention.

The *read* and *write* control lines control the direction of data on the data bus. When both contain a logic one, the CPU and memory-I/O are not communicating with one another. If the read line is low (logic zero), the CPU is reading data from memory (that is, the system is transferring data from memory to the CPU). If the write line is low, the system transfers data from the CPU to memory.

The *byte enable lines* are another set of important control lines. These control lines allow 16, 32, and 64 bit processors to deal with smaller chunks of data. Additional details appear in the next section.

The 80x86 family, unlike many other processors, provides two distinct address spaces: one for memory and one for I/O. While the memory address busses on various 80x86 processors vary in size, the I/O address bus on all 80x86 CPUs is 16 bits wide. This allows the processor to address up to 65,536 different I/O *locations*. As it turns out, most devices (like the keyboard, printer, disk drives, etc.) require more than one I/O location. Nonetheless, 65,536 I/O locations are more than sufficient for most applications. The original IBM PC design only allowed the use of 1,024 of these.

Although the 80x86 family supports two address spaces, it does not have two address busses (for I/O and memory). Instead, the system shares the address bus for both I/O and memory addresses. Additional control lines decide whether the address is intended for memory or I/O. When such signals are active, the I/O devices use the address on the L.O. 16 bits of the address bus. When inactive, the I/O devices ignore the signals on the address bus (the memory subsystem takes over at that point).

[1]Actually, newer members of the family tend to use lower voltage signals, but these remain compatible with TTL signals.

[2]TTL logic represents the value zero with a voltage in the range 0.0-0.8v. It represents a one with a voltage in the range 2.4-5v. If the signal on a bus line is between 0.8v and 2.4v, it's value is indeterminate. Such a condition should only exist when a bus line is changing from one state to the other.

1.2.2 The Memory Subsystem

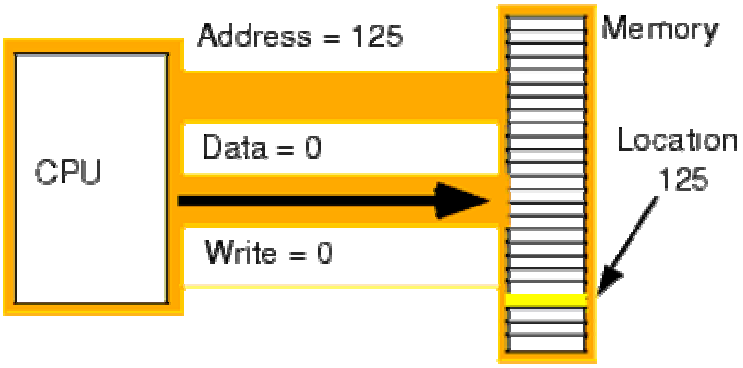
A typical 80x86 processor addresses a maximum of 2^n different memory locations, where *n* is the number of bits on the address bus[1]. As you've seen already, 80x86 processors have 20, 24, 32, and 36 bit address busses (with 64 bits on the way).

Of course, the first question you should ask is, "What exactly is a memory location?" The 80x86 supports *byte addressable memory*. Therefore, the basic memory unit is a byte. So with 20, 24, 32, and 36 address lines, the 80x86 processors can address one megabyte, 16 megabytes, four gigabytes, and 64 gigabytes of memory, respectively.

Think of memory as a linear array of bytes. The address of the first byte is zero and the address of the last byte is 2^n-1. For an 8088 with a 20 bit address bus, the following pseudo-Pascal array declaration is a good approximation of memory:

Memory: array [0..1048575] of byte;

To execute the equivalent of the Pascal statement "Memory [125] := 0;" the CPU places the value zero on the data bus, the address 125 on the address bus, and asserts the write line (since the CPU is writing data to memory).



The term "byte addressable memory array" means that the CPU can address memory in chunks as small as a single byte. It also means that this is the *smallest* unit of memory you can access at once with the processor. That is, if the processor wants to access a four bit value, it must read eight bits and then ignore the extra four bits. Also realize that byte addressability does not imply that the CPU can access eight bits on any arbitrary bit boundary. When you specify address 125 in memory, you get the

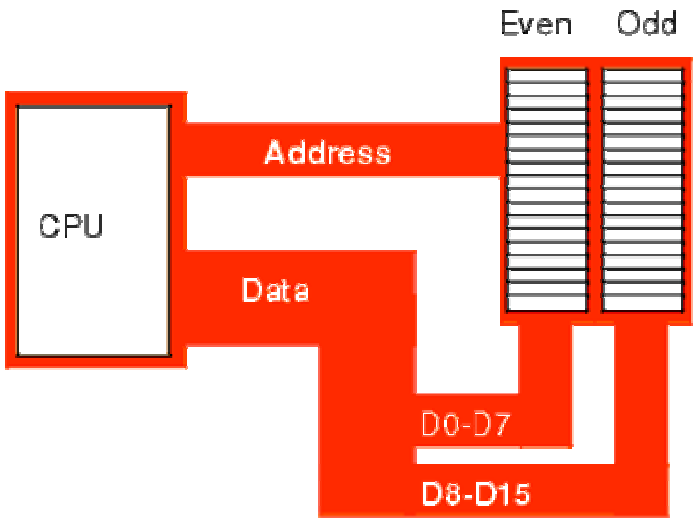
entire eight bits at that address, nothing less, nothing more. Addresses are integers; you cannot, for example, specify address 125.5 to fetch fewer than eight bits.

CPUs with an eight-bit bus can manipulate word and double word values, even through their data bus is only eight bits wide. However, this requires multiple memory operations because these processors can only move eight bits of data at once. To load a word requires two memory operations; to load a double word requires four memory operations.

Some older x86 CPUs (e.g., the 8086 and 80286) have a 16 bit data bus. This allows these processors to access twice as much memory in the same amount of time as their eight bit brethren. These processors organize memory into two *banks*: an "even" bank and an "odd" bank (see Figure 1.6). Figure 1.7 illustrates the connection to the CPU (D0-D7 denotes the L.O. byte of the data bus, D8-D15 denotes the H.O. byte of the data bus):

	Even	Odd
Word 3	6	7
Word 2	4	5
Word 1	2	3
Word 0	0	1

Numbers in cells represent the byte addresses



1.2.3 The I/O Subsystem

Besides the 20, 24, or 32 address lines which access memory, the 80x86 family provides a 16 bit I/O address bus. This gives the 80x86 CPUs two separate address spaces: one for memory and one for I/O operations. Lines on the control bus differentiate between memory and I/O addresses. Other than separate control lines and a smaller bus, I/O addressing behaves exactly like memory addressing. Memory and I/O devices both share the same data bus and the L.O. 16 lines on the address bus.

There are three limitations to the I/O subsystem on the PC: first, the 80x86 CPUs require special instructions to access I/O devices; second, the designers of the PC used the "best" I/O locations for their own purposes, forcing third party developers to use less accessible locations; third, 80x86 systems can address no more than 65,536 (2^{16}) I/O addresses. When you consider that a typical video display card requires over eight megabytes of addressable locations, you can see a problem with the size of I/O bus.

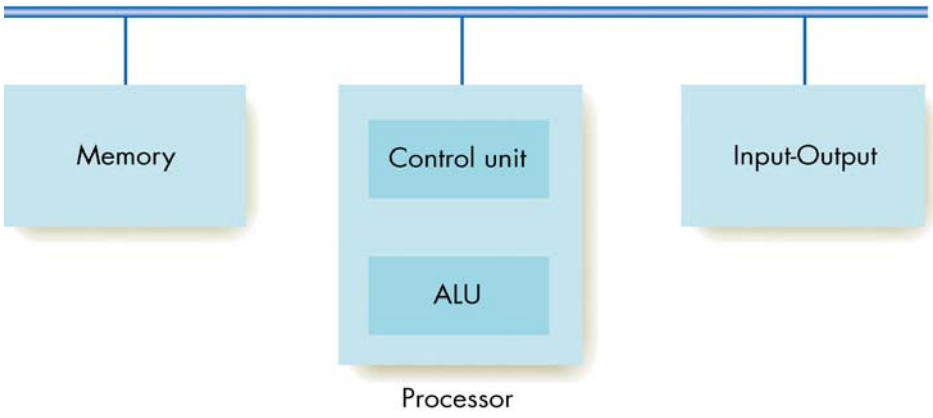
Fortunately, hardware designers can map their I/O devices into the memory address space as easily as they can the I/O address space. So by using the appropriate circuitry, they can make their I/O devices look just like memory. This is how, for example, display adapters on the PC work.

This is the *maximum*. Most computer systems built around 80x86 family do not include the maximum addressable amount of memory.

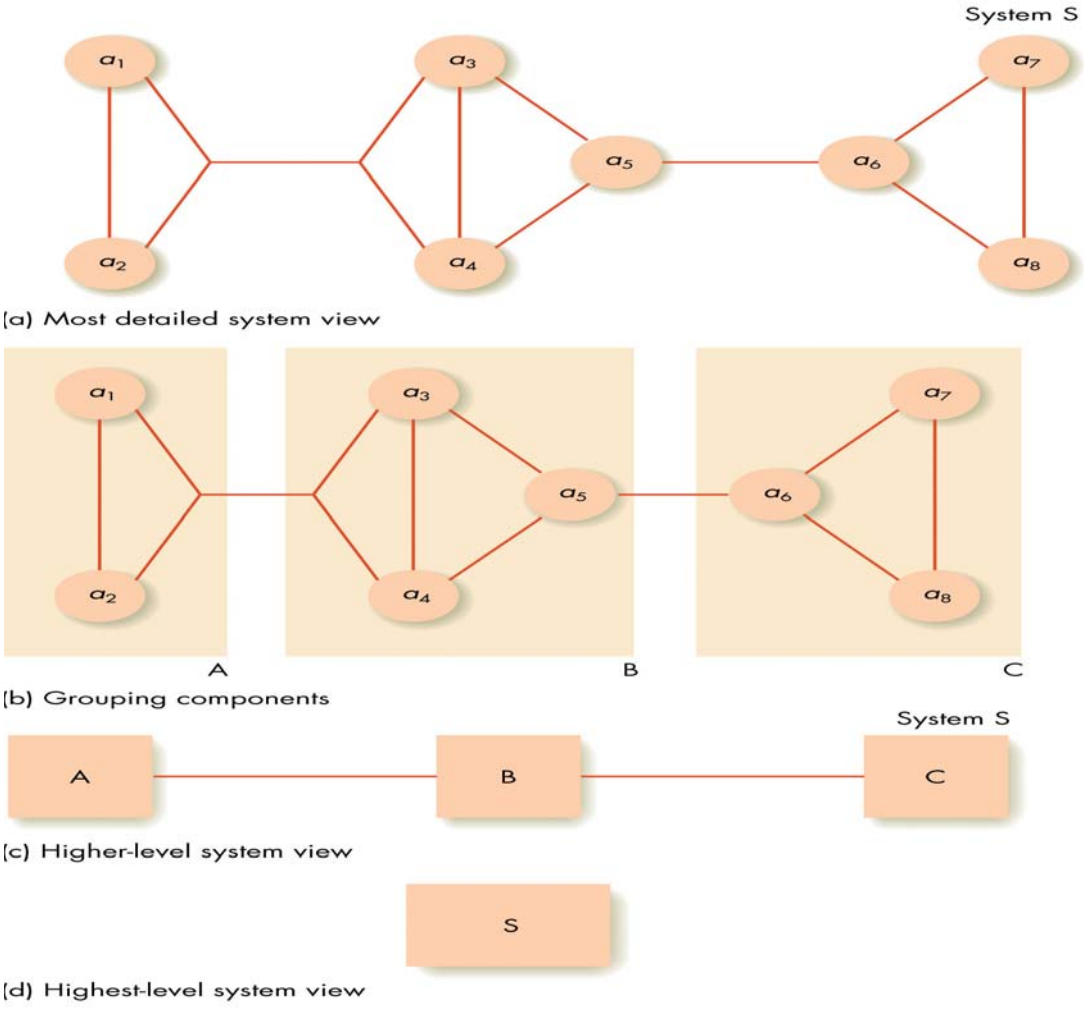
Memory Hierarchies

We are concerned with five types of memory:

1. ***Registers*** are the fastest type of memory, which are located internal to a processor. These elements are primarily used for temporary storage of operands, small partitions of memory, etc., and are assumed to be one word (32 bits) in length in the MIPS architecture.
2. ***Cache*** is a very fast type of memory that can be external or internal to a given processor. Cache is used for temporary storage of blocks of data obtained from main memory (*read* operation) or created by the processor and eventually written to main memory (*write* operation).
3. ***Main Memory*** is modelled as a large, linear array of storage elements that is partitioned into static and dynamic storage, as discussed in Section 2 of these notes. Main memory is used primarily for storage of data that a program produces during its execution, as well as for instruction storage. However, if main memory is too small for such data, then subsets of the data can be *paged* to/from disk storage.
4. ***Disk Storage*** is much slower than main memory, but also has much higher capacity than the preceding three types of memory. Because of the relatively long search times, we prefer not to find data primarily in disk storage, but to *page* the disk data into main memory, where it can be searched much faster.
5. ***Archival Storage*** is offline storage such as a CD-ROM jukebox or (in former years) rooms filled with racks containing magnetic tapes. This type of storage has a very long access time, in comparison with disk storage, and is also designed to be much less volatile than disk data.



The Concept of Abstraction



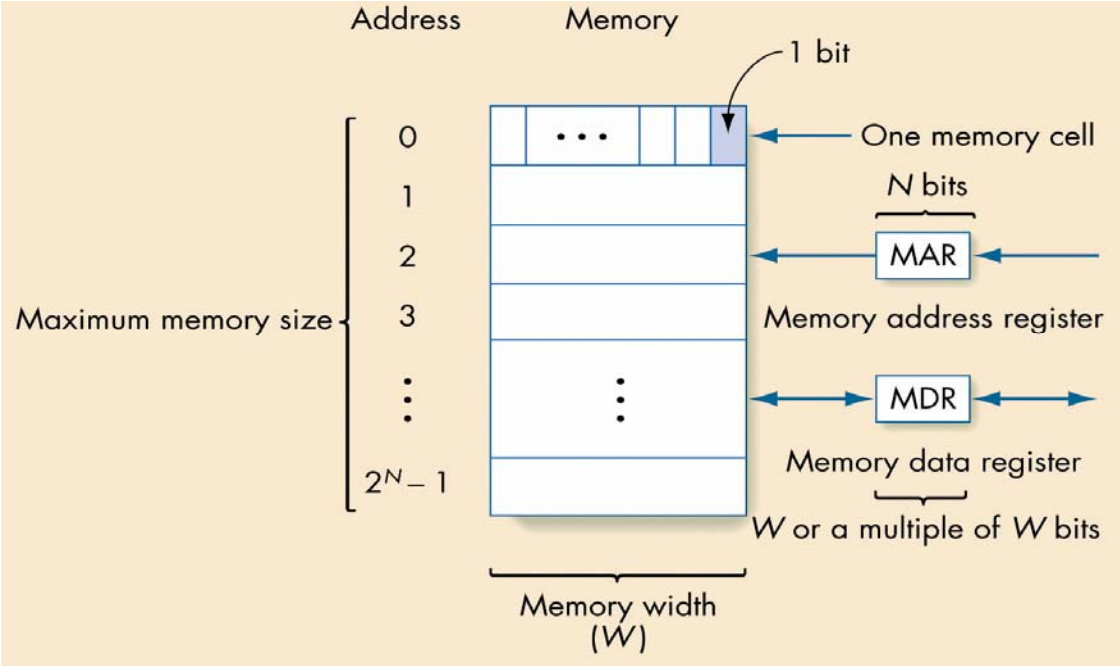
(a) Most detailed system view

(b) Grouping components

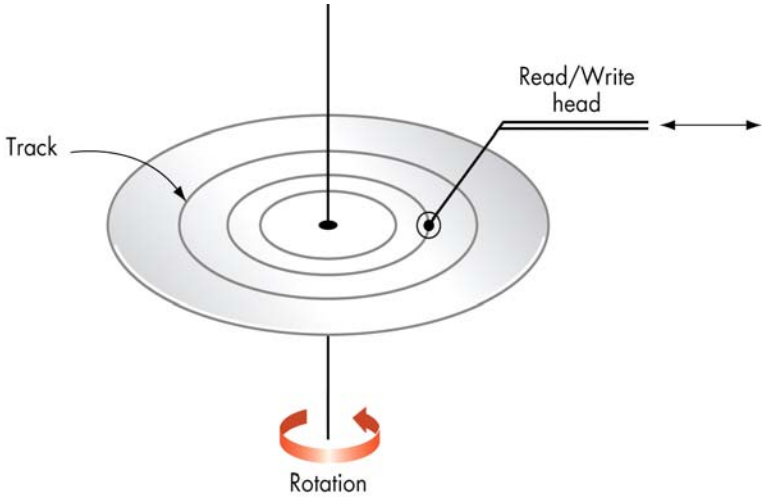
(c) Higher-level system view

(d) Highest-level system view

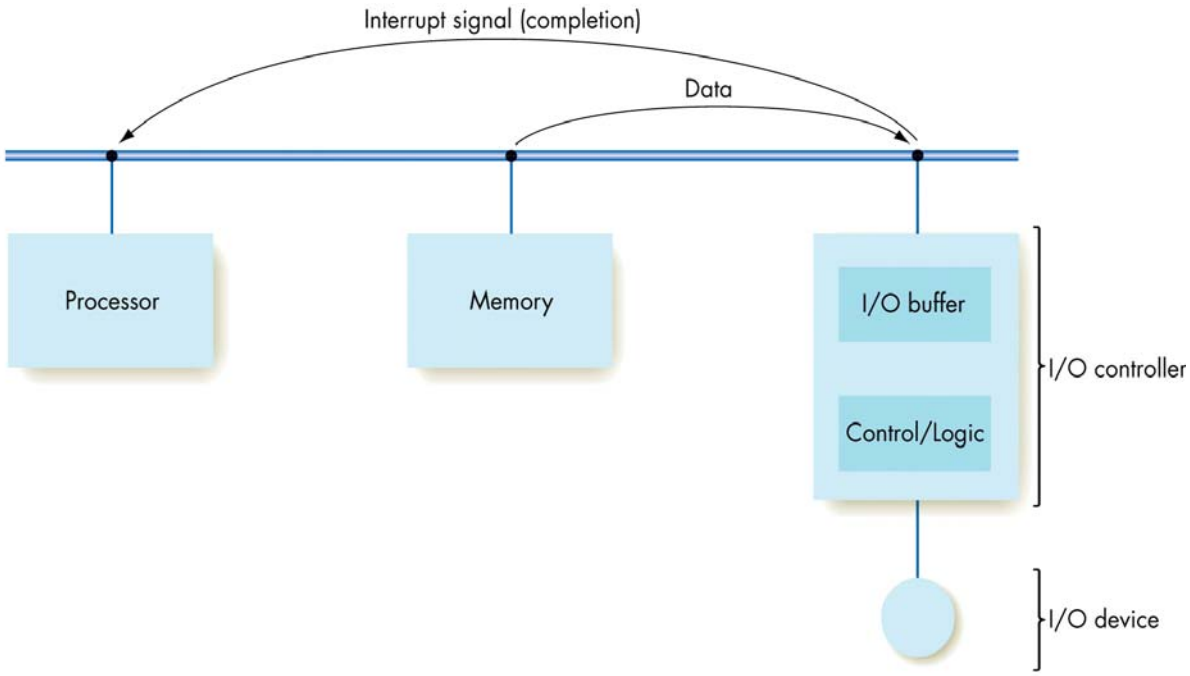
Structure of Random Access Memory



Input/Output and Mass Storage (continued)



Organization of an I/O Controller



Machine Language Instructions

Can be decoded and executed by control unit

- Parts of instructions
- Operation code (op code)
- Unique unsigned-integer code assigned to each machine language operation
- Address fields
- Memory addresses of the values on whixh operation will work.

Conclusion

Computer Systems provides the foundation upon which all other software applications rely. The goal of systems research is to develop the key abstractions and services that enable software to be efficiently and portably run on hardware. Areas of interest to the systems group include operating systems, computer networks, parallel and distributed computation, and computer security.computer systems information security and privacy. In the past, information systems were large, expensive, barely inter-connected, and non-mobile. Today, information systems are small, lightweight, highly connected via wireless technology and mobile. Tomorrow, they will be even smaller and more mobile. The constant evolution in the design and operation of computer systems presents new and increasingly complex challenges for computer hardware management hardware and software that is capable of detecting potential intruders in a high assurance manner and ensures that the intruders activities does not compromise the main system.