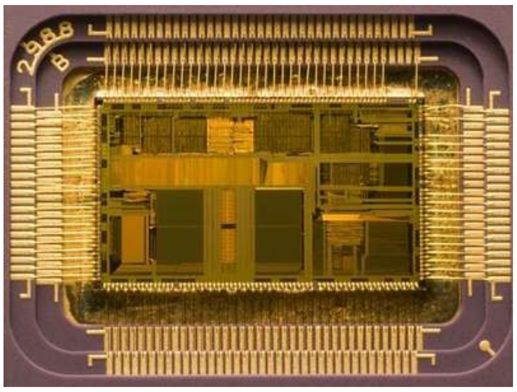
DAR ES SALAAM INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER STUDIES
COU07302 MICROPROCESSOR AND COMPUTER ARCHITECTURE

Lecture 1

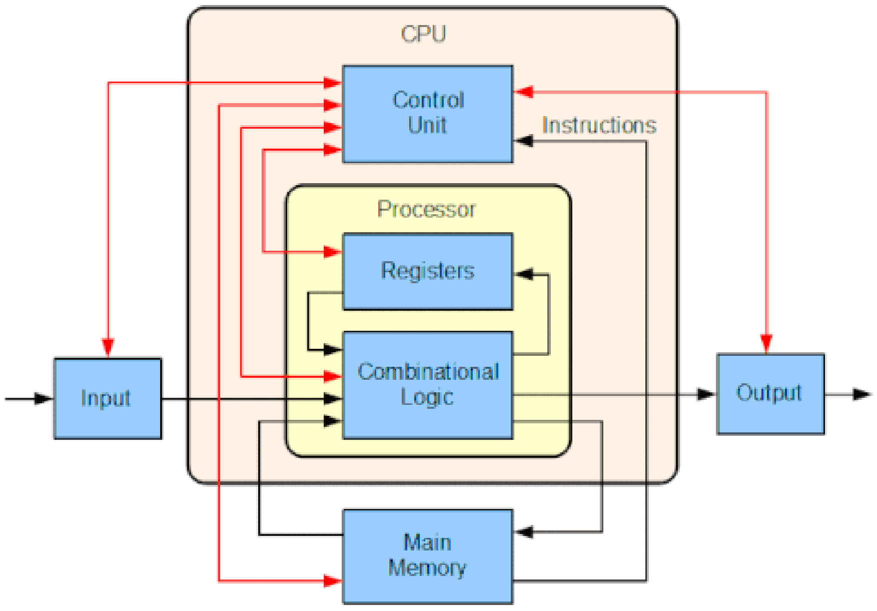
by
E. Kondela

1. INTRODUCTION:

Microprocessor an integrated circuit that contains all the functions of a central processing unit of a computer.
Microprocessor is any of a type of miniature electronic device that contains the arithmetic, logic and control circuitry necessary to perform the functions of a digital computer's central processing unit. In effect, this kind of integrated circuit can interpret and execute program instructions as well as handle arithmetic operations.



Computer architecture is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. The architecture of a system refers to its structure in terms of separately specified components of that system and their interrelationships.



2. EVOLUTION OF COMPUTERS

2.1 First Generation: Vacuum Tubes

2.1.1 ENIAC: The ENIAC (Electronic Numerical Integrator And Computer), designed and constructed at
the University of Pennsylvania, was the world's first general-purpose electronic digital computer.
The project was a response to U.S needs during World War II. John Mauchly, a professor of electrical engineering at the University of Pennsylvania, and John Eckert, one of his graduate students, proposed to build a general-purpose computer using vacuum tubes for the BRL's application. In 1943, the Army accepted this proposal, and work began on the ENIAC. The resulting machine was enormous, weighing 30 tons, occupying 1500 square feet of floor space, and containing more than 18,000 vacuum tubes. When operating, it consumed 140 kilowatts of power. It was also substantially faster than any electromechanical computer, capable of 5000 additions per second. The ENIAC was completed in 1946, too late to be used in the war effort. The use of the ENIAC for a purpose other than that for which it was built demonstrated its general-purpose nature. The ENIAC continued to operate under BRL management until 1955, when it was disassembled.

2.1.2 THE VON NEUMANN MACHINE The task of entering and altering programs for the ENIAC was extremely tedious. The programming process can be easy if the program could be represented in a form suitable for storing in memory alongside the data. Then, a computer could get its instructions by reading them from memory, and a program could be set or altered by setting the values of a portion of memory.
In 1946, von Neumann and his colleagues began the design of a new stored-program computer, referred to as the IAS computer, at the Princeton Institute for Advanced Studies. The IAS computer,although not completed until 1952,is the prototype of all subsequent general-purpose computers.

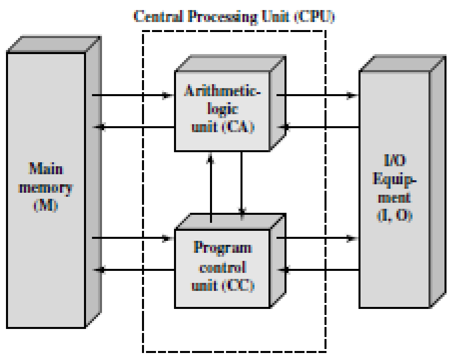

Central Processing Unit (CPU)
Main memory (M)
Arithmetic-logic unit (CA)
Program control unit (CC)
I/O Equip-ment (I, O)

Figure 1 shows the general structure of the IAS computer).

The IAS computer consists of
1. A main memory, which stores both data and instruction
2. An arithmetic and logic unit (ALU) capable of operating on binary data
3. A control unit, which interprets the instructions in memory and causes them to be executed
4. Input and output (I/O) equipment operated by the control unit

This structure was outlined in von Neumann's earlier proposal, which is worth quoting at this point:
1. Because the device is primarily a computer, it will have to perform the elementary operations of arithmetic most frequently. At any rate a central arithmetical part of the device will probably have to exist and this constitutes the first specific part: CA.
2. The logical control of the device, that is, the proper sequencing of its operations, can be most efficiently carried out by a central control organ. By the central control and the organs which perform it form the second specific part: CC

3. Any device which is to carry out long and complicated sequences of operations (specifically of calculations) must have a considerable memory . . . At any rate, the total memory constitutes the third specific part of the device: M.
4. The device must have organs to transfer . . . information from R into its specific parts C and M. These organs form its input, the fourth specific part: I
5. The device must have organs to transfer . . . from its specific parts C and M into R. These organs form its output, the fifth specific part: O.

The control unit operates the IAS by fetching instructions from memory and executing them one at a time. A more detailed structure diagram is shown in Figure 2. This figure reveals that both the control unit and the ALU contain storage locations, called registers, defined as follows:

1. Memory buffer register (MBR): Contains a word to be stored in memory or sent to the I/O unit, or is used to receive a word from memory or from the I/O unit.
2. Memory address register (MAR): Specifies the address in memory of the word to be written from or read into the MBR.
3. Instruction register (IR): Contains the 8-bit opcode instruction being executed.
4. Instruction buffer register (IBR): Employed to hold temporarily the right-hand instruction from a word in memory.
5. Program counter (PC): Contains the address of the next instruction-pair to be fetched from memory.
6. Accumulator (AC) and multiplier quotient (MQ): Employed to hold temporarily operands and results of ALU operations.

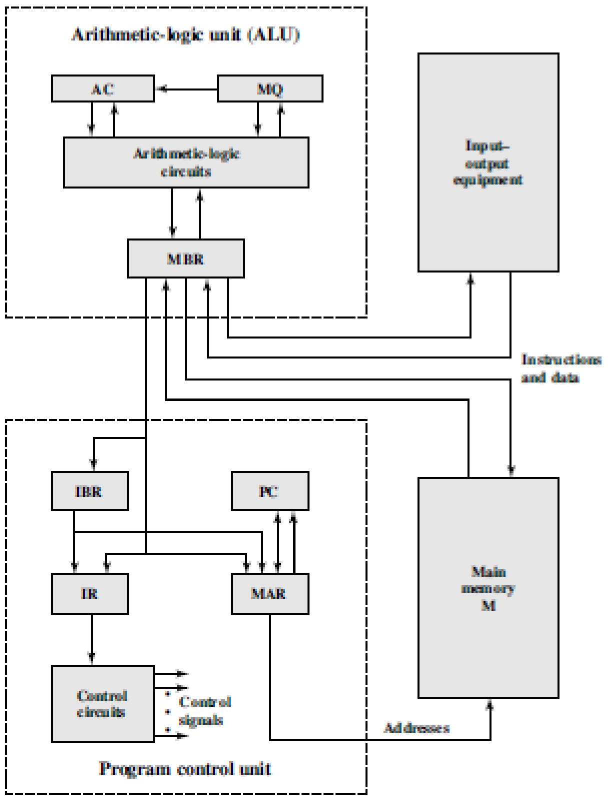


Figure 2 Expanded Structure of IAS Computer

COMMERCIAL COMPUTERS The 1950s saw the birth of the computer industry with two companies, Sperry and IBM, dominating the marketplace. In 1947, Eckert and Mauchly formed the

Eckert-Mauchly Computer Corporation to manufacture computers commercially. Their first successful machine was the UNIVAC I (Universal Automatic Computer), which was commissioned by the Bureau of the Census for the 1950 calculations.The Eckert-Mauchly Computer Corporation became part of the UNIVAC division of Sperry-Rand Corporation, which went on to build a series of successor machines.

1. The UNIVAC I was the first successful commercial computer. It was intended for both scientific and commercial applications.
2. The UNIVAC II, which had greater memory capacity and higher performance than the UNIVAC I, was delivered in the late 1950s and illustrates several trends that have remained characteristic of the computer industry.

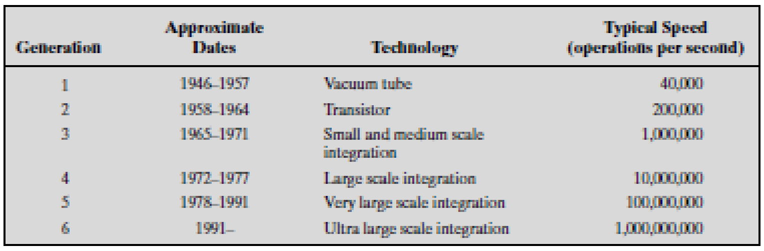
The UNIVAC division also began development of the 1100 series of computers, which was to be its major source of revenue. This series illustrates a distinction that existed at one time. The first model, the UNIVAC 1103, and its successors for many years were primarily intended for scientific applications, involving long and complex calculations.

2.2 The Second Generation: Transistors

The first major change in the electronic computer came with the replacement of the vacuum tube by the transistor. The transistor is smaller, cheaper, and dissipates less heat than a vacuum tube but can be used in the same way as a vacuum tube to construct computers. Unlike the vacuum tube, which requires wires, metal plates, a glass capsule, and a vacuum, the transistor is a solid- state device, made from silicon.

The transistor was invented at Bell Labs in 1947 and by the 1950s had launched an electronic revolution. It was not until the late 1950s, however, that fully transistorized computers were commercially available.

The use of the transistor defines the second generation of computers. It has become widely accepted to classify computers into generations based on the fundamental hardware technology employed (Table 1)



Generation	Approximate Dates	Technology	Typical Speed (operations per second)
1	1946–1957	Vacuum tube	40,000
2	1958–1964	Transistor	200,000
3	1965–1971	Small and medium scale integration	1,000,000
4	1972–1977	Large scale integration	10,000,000
5	1978–1991	Very large scale integration	100,000,000
6	1991–	Ultra large scale integration	1,000,000,000

Table 1 Computer Generations

THE IBM 7094 From the introduction of the 700 series in 1952 to the introduction of the last member of the 7000 series in 1964, this IBM product line underwent an evolution that is typical of computer products. Successive members of the product line show increased performance, increased capacity, and/or lower cost.

2.3 The Third Generation: Integrated Circuits

In 1958 came the achievement that revolutionized electronics and started the era of microelectronics: the invention of the integrated circuit. It is the integrated circuit that defines the third generation of computers.

MICROELECTRONICS: Microelectronics means, literally, “small electronics.” Since the beginnings of digital electronics and the computer industry, there has been a persistent and consistent trend toward the reduction in size of digital electronic circuits.

IBM SYSTEM/360: By 1964, IBM had a firm grip on the computer market with its 7000 series of machines. In that year, IBM announced the System/360, a new family of computer products. DEC

PDP-8: In the same year that IBM shipped its first System/360, another momentous first shipment occurred: PDP-8 from Digital Equipment Corporation (DEC).At a time when the average computer required an air-conditioned room,the PDP-8 (dubbed a minicomputer by the industry, after the miniskirt of the day) was small enough that it could be placed on top of a lab bench or be built into other equipment. It could not do everything the mainframe could, but at $16,000, it was cheap enough for each lab technician to have one. In contrast, the System/360 series of mainframe computers introduced just a few months before cost hundreds of thousands of dollars.

2.4 Later Generations

Table 1 suggests that there have been a number of later generations, based on advances in integrated circuit technology. With the introduction of large-scale integration (LSI), more than 1000 components can be placed on a single integrated circuit chip. Very-large-scale integration (VLSI) achieved more than 10,000 components per chip, while current ultra-large-scale integration (ULSI) chips can contain more than one million components.

SEMICONDUCTOR MEMORY: The first application of integrated circuit technology to computers was construction of the processor (the control unit and the arithmetic and logic unit) out of integrated circuit chips. But it was also found that this same technology could be used to construct memories.

MICROPROCESSORS: Just as the density of elements on memory chips has continued to rise,so has the density of elements on processor chips. As time went on,more and more elements were placed on each chip, so that fewer and fewer chips were needed to construct a single computer processor.
A breakthrough was achieved in 1971,when Intel developed its 4004.The 4004 was the first chip to contain all of the components of a CPU on a single chip.
The next major step in the evolution of the microprocessor was the introduction in 1972 of the Intel 8008. This was the first 8-bit microprocessor and was almost twice as complex as the 4004.
Neither of these steps was to have the impact of the next major event: the introduction in 1974 of the Intel 8080.This was the first general-purpose microprocessor. Whereas the 4004 and the 8008 had been designed for specific applications, the 8080 was designed to be the CPU of a general-purpose microcomputer
About the same time, 16-bit microprocessors began to be developed. However, it was not until the end of the 1970s that powerful, general-purpose 16-bit microprocessors appeared. One of these was the 8086.

3. EVOLUTION OF INTEL X86 ARCHITECTURE:

We have Two computer families: the Intel x86 and the ARM architecture. The current x86 offerings represent the results of decades of design effort on complex instruction set computers

(CISCs).

The x86 incorporates the sophisticated design principles once found only on mainframes and supercomputers and serves as an excellent example of CISC design. An alternative approach to processor design in the reduced instruction set computer (RISC). The ARM architecture is used in a wide variety of embedded systems and is one of the most powerful and best-designed RISC-based systems on the market.

In terms of market share, Intel has ranked as the number one maker of microprocessors for non-embedded systems for decades, a position it seems unlikely to yield. Interestingly, as microprocessors have grown faster and much more complex, Intel has actually picked up the pace.

Intel used to develop microprocessors one after another, every four years.

It is worthwhile to list some of the highlights of the evolution of the Intel product line:

1. 8080: The world's first general-purpose microprocessor. This was an 8-bit machine, with an 8-bit data path to memory. The 8080 was used in the first personal computer, the Altair.
2. 8086: A far more powerful, 16-bit machine. In addition to a wider data path and larger registers, the 8086 sported an instruction cache, or queue, that prefetches a few instructions before they are executed. A variant of this processor, the 8088, was used in IBM's first personal computer, securing the success of Intel. The 8086 is the first appearance of the x86 architecture.
3. 80286: This extension of the 8086 enabled addressing a 16-MByte memory instead of just 1 Mbyte.
4. 80386: Intel's first 32-bit machine, and a major overhaul of the product. With a 32-bit architecture, the 80386 rivaled the complexity and power of minicomputers and mainframes introduced just a few years earlier. This was the first Intel processor to support multitasking, meaning it could run multiple programs at the same time.
5. 80486: The 80486 introduced the use of much more sophisticated and powerful cache technology and sophisticated instruction pipelining. The 80486 also offered a built-in math coprocessor, offloading complex math operations from the main CPU.
6. Pentium: With the Pentium, Intel introduced the use of superscalar techniques, which allow multiple instructions to execute in parallel.
7. Pentium Pro: The Pentium Pro continued the move into superscalar organization begun with the Pentium, with aggressive use of register renaming, branch prediction, data flow analysis, and speculative execution.
8. Pentium II: The Pentium II incorporated Intel MMX technology, which is designed specifically to process video, audio, and graphics data efficiently.
9. Pentium III: The Pentium III incorporates additional floating-point instructions to support 3D graphics software.
10. Pentium 4: The Pentium 4 includes additional floating-point and other enhancements for 8 multimedia.
11. Core: This is the first Intel x86 microprocessor with a dual core, referring to the implementation of two processors on a single chip.
12. Core 2: The Core 2 extends the architecture to 64 bits. The Core 2 Quad provides four processors on a single chip.

    Over 30 years after its introduction in 1978, the x86 architecture continues to dominate the processor market outside of embedded systems. Although the organization and technology of the x86 machines has changed dramatically over the decades, the instruction set architecture has evolved to remain backward compatible with earlier versions. Thus, any program written on an older version of the x86 architecture can execute on newer versions. All changes to the instruction set architecture have involved additions to the instruction set, with no subtractions. The rate of change has been the addition of roughly one instruction per month added to the architecture over the 30 years. so that there are now over 500 instructions in the instruction set.
13. The x86 provides an excellent illustration of the advances in computer hardware over the

past 30 years. The 1978 8086 was introduced with a clock speed of 5 MHz and had 29,000 transistors. A quad-core Intel Core 2 introduced in 2008 operates at 3 GHz, a speedup of a factor of 600, and has 820 million transistors, about 28,000 times as many as the 8086. Yet the Core 2 is in only a slightly larger package than the 8086 and has a comparable cost.

4. DESIGNING FOR PERFORMANCE

Year by year, the cost of computer systems continues to drop dramatically, while the performance and capacity of those systems continue to rise equally dramatically. Desktop applications that require the great power of today's microprocessor-based systems include

1. Image processing
2. Speech recognition
3. Videoconferencing
4. Multimedia authoring
5. Voice and video annotation of files
6. Simulation modeling

Microprocessor Speed

The evolution of Microprocessors continues to bear out Moore's law. So long as this law holds, chipmakers can unleash a new generation of chips every three years—with four times as many transistors. In microprocessors, the addition of new circuits, and the speed boost that comes from reducing the distances between them, has improved performance four- or fivefold every three years or so since Intel launched its x86 family in 1978. The more elaborate techniques for feeding
the monster into contemporary processors are the following:

1. Branch prediction: The processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next
2. Data flow analysis: The processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions
3. Speculative execution: Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations.

Performance Balance

While processor power has raced ahead at breakneck speed, other critical components of the computer have not kept up.The result is a need to look for performance balance: an adjusting of the organization and architecture to compensate for the mismatch among the capabilities of the various components.

The interface between processor and main memory is the most crucial pathway in the entire computer because it is responsible for carrying a constant flow of program instructions and data between memory chips and the processor.

There are a number of ways that a system architect can attack this problem, all of which are reflected in contemporary computer designs. Consider the following examples:

1. Increase the number of bits that are retrieved at one time by making DRAMs "wider" rather than "deeper" and by using wide bus data paths.
2. Change the DRAM interface to make it more efficient by including a cache7 or other buffering scheme on the DRAM chip.
3. Reduce the frequency of memory access by incorporating increasingly complex and efficient cache structures between the processor and main memory.
4. Increase the interconnect bandwidth between processors and memory by using higher- speed buses and by using a hierarchy of buses to buffer and structure data flow.

Improvements in Chip Organization and Architecture

There are three approaches to achieving increased processor speed:

1. Increase the hardware speed of the processor.
2. Increase the size and speed of caches that are interposed between the processor and main memory. In particular, by dedicating a portion of the processor chip itself to the cache, cache access times drop significantly.
3. Make changes to the processor organization and architecture that increase the effective speed of instruction execution.

However, as clock speed and logic density increase, a number of obstacles become more significant:

1. Power: As the density of logic and the clock speed on a chip increase, so does the power density.
2. RC delay: The speed at which electrons can flow on a chip between transistors is limited by the resistance and capacitance of the metal wires connecting them; specifically, delay increases as the RC product increases. As components on the chip decrease in size, the wire interconnects become thinner, increasing resistance. Also, the wires are closer together, increasing capacitance.
3. Memory latency: Memory speeds lag processor speeds.

   Beginning in the late 1980s, and continuing for about 15 years, two main strategies have been used to increase performance beyond what can be achieved simply by increasing clock speed. First, there has been an increase in cache capacity. Second, the instruction execution logic within a processor has become increasingly complex to enable parallel execution of instructions within the processor.

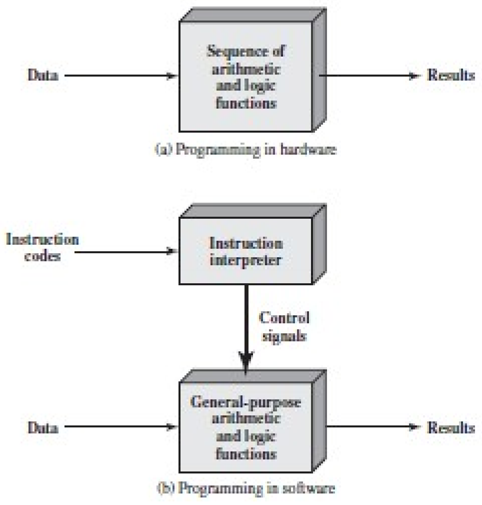
   Two noteworthy design approaches have been pipelining and superscalar. A pipeline works much as an assembly line in a manufacturing plant enabling different stages of execution of different instructions to occur at the same time along the pipeline. A superscalar approach in essence allows multiple pipelines within a single processor so that instructions that do not depend on one another can be executed in parallel.

5. COMPUTER COMPONENTS

Virtually all contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton. Such a design is referred to as the von Neumann architecture and is based on three key concepts:

1. Data and instructions are stored in a single read–write memory.
2. The contents of this memory are addressable by location, without regard to the type of data contained there.
3. Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next.

If there is a particular computation to be performed, a configuration of logic components designed specifically for that computation could be constructed. The resulting “program” is in the form of hardware and is termed a hardwired program.
Now consider this alternative. Suppose we construct a general-purpose configuration of arithmetic and logic functions. This set of hardware will perform various functions on data depending on control signals applied to the hardware. In the original case of customized hardware, the system accepts data and produces results (Figure 3a). With general-purpose hardware, the system accepts data and control signals and produces results.
Thus, instead of rewiring the hardware for each new program, the programmer merely needs to supply a new set of control signals by providing a unique code for each possible set of control signals, and let us add to the general-purpose hardware a segment that can accept a code and generate control signals (Figure 3b). To distinguish this new method of programming, a sequence of codes or instructions is called software.


Data
Sequence of arithmetic and logic functions
Results
(a) Programming in hardware
Instruction codes
Instruction interpreter
Control signals
Data
General-purpose arithmetic and logic functions
Results
(b) Programming in software

3 Hardware and Software approaches

Figure 3b indicates two major components of the system: an instruction interpreter and a module of general-purpose arithmetic and logic functions. These two constitute the CPU. Data and instructions must be put into the system. For this we need some sort of input module. A means of reporting results is needed, and this is in the form of an output module. Taken together, these are referred to as I/O components.
There must be a place to store temporarily both instructions and data. That module is called memory, or main memory to distinguish it from external storage or peripheral devices. Von Neumann pointed out that the same memory could be used to store both instructions and data.
Figure 4 illustrates these top-level components and suggests the interactions among them. The CPU exchanges data with memory. For this purpose, it typically makes use of two internal (to the CPU) registers: a memory address register (MAR), which specifies the address in memory for the next read or write, and a memory buffer register (MBR), which contains the data to be written into memory or receives the data read from memory. Similarly, an I/O address register (I/OAR) specifies a particular I/O device. An I/O buffer (I/OBR) register is used for the exchange of data between an I/O module and the CPU.

A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data. An I/O module transfers data from external devices to CPU and memory, and vice versa. It contains internal buffers for temporarily holding these data until they can be sent on.

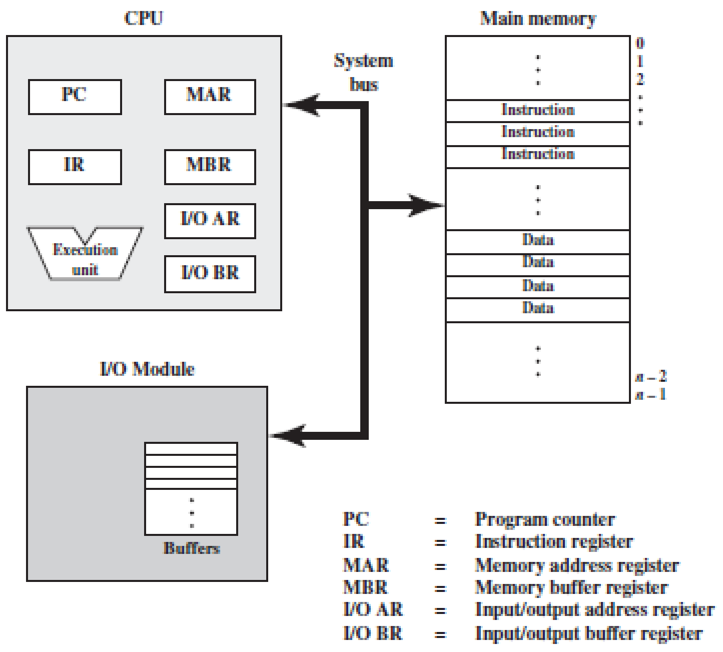


Figure 4. Computer Components

6. COMPUTER FUNCTIONS

The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory. Instruction processing consists of two steps: The processor reads

(fetches) instructions from memory one at a time and executes each instruction. Program execution consists of repeating the process of instruction fetch and instruction execution.
The processing required for a single instruction is called an instruction cycle. Using the simplified two-step description given previously, the instruction cycle is depicted in Figure 5. The two steps are referred to as the fetch cycle and the execute cycle. Program execution halts only if the machine is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the computer is encountered.

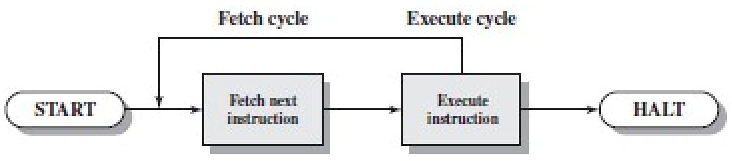


Figure 5 Basic Instruction Cycle

a) Instruction Fetch and Execute
At the beginning of each instruction cycle, the processor fetches an instruction from memory. The program counter (PC) holds the address of the instruction to be fetched next, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence.
For example, consider a computer in which each instruction occupies one 16-bit word of memory. If the program counter is set to location 300. The processor will next fetch the instruction at location 300. On next instruction cycles, it will fetch instructions from locations 301,302,303,and so on.
The fetched instruction is loaded into a register in the processor known as the instruction register (IR). The processor interprets the instruction and performs the required action. In general, these actions fall into four categories:

1. Processor-memory: Data may be transferred from processor to memory or from memory to processor.
2. Processor-I/O: Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
3. Data processing: The processor may perform some arithmetic or logic operation on data.
4. Control: An instruction may specify that the sequence of execution be altered. For example, the processor may fetch an instruction from location 149, which specifies that the next instruction be from location 182. The processor will remember this fact by setting the program counter to 182.Thus,on the next fetch cycle, the instruction will be fetched from location 182 rather than 150.

An instruction's execution may involve a combination of these actions. Consider a simple example using a hypothetical machine that includes the characteristics listed in Figure 6. The processor contains a single data register, called an accumulator (AC). Both instructions and data are 16 bits long. Thus, it is convenient to organize memory using 16-bit words. The instruction format provides 4 bits for the opcode, so that there can be as many as 2^4 = 16 different opcodes, and up to 2^12 = 4096 (4K) words of memory can be directly addressed.

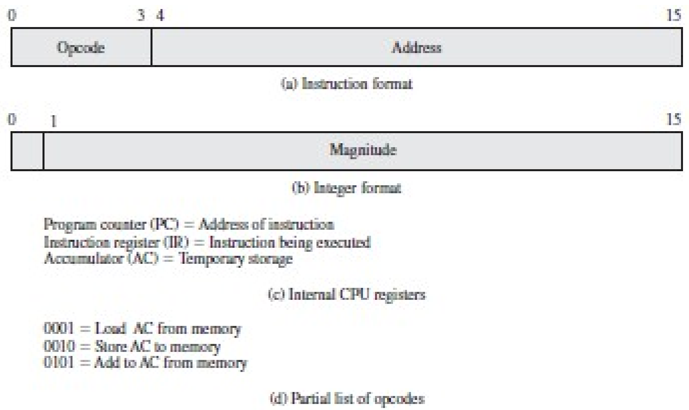

0
3 4
15
Opcode
Address
(a) Instruction format
0
1
15
Magnitude
(b) Integer format
Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage
(c) Internal CPU registers
0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory
(d) Partial list of opcodes

Figure 6 Characteristics of a Hypothetical machine

Figure 7 illustrates a partial program execution, showing the relevant portions of memory and processor registers.1 The program fragment shown adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in later location.

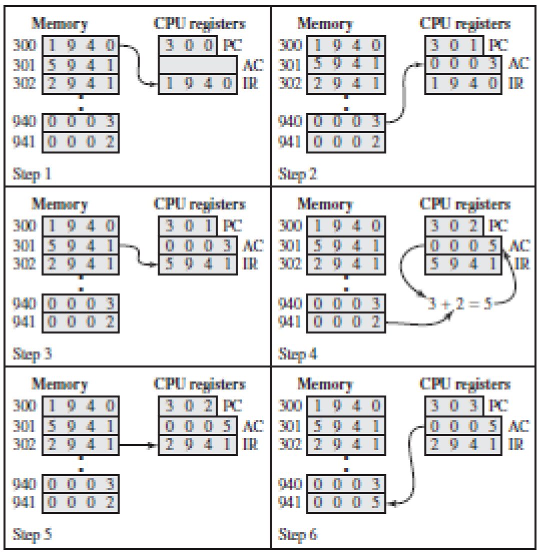


Figure 7 Example of Program Execution

Three instructions are required:

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the instruction register IR and the PC is incremented.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded. The remaining 12 bits (three hexadecimal digits) specify the address (940) from which data are to be loaded.
3. The next instruction (5941) is fetched from location 301 and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
5. The next instruction (2941) is fetched from location 302 and the PC is incremented.
6. The contents of the AC are stored in location 941.

For example, the PDP-11 processor includes an instruction, expressed symbolically as ADD B,A, that stores the sum of the contents of memory locations B and A into memory location A. A single instruction cycle with the following steps occurs:

1. Fetch the ADD instruction.
2. Read the contents of memory location A into the processor.

3. Read the contents of memory location B into the processor. In order that the contents of A are not lost, the processor must have at least two registers for storing memory values, rather than a single accumulator.
4. Add the two values.
5. Write the result from the processor to memory location A.

Figure 8 provides a more detailed look at the basic instruction cycle of Figure 1.5.The figure is in the form of a state diagram The states can be described as follows:

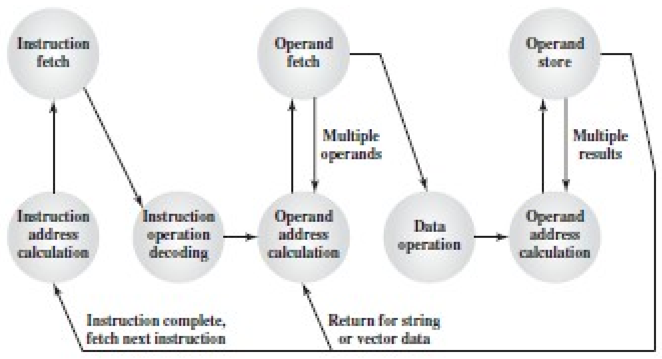
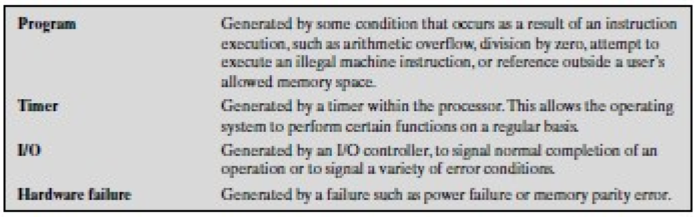

Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

Figure 8 Instruction Cycle State Diagram

1. Instruction address calculation (iac): Determine the address of the next instruction to be executed.
2. Instruction fetch (if): Read instruction from its memory location into the processor.
3. Instruction operation decoding (iod): Analyze instruction to determine type of operation to be performed and operand(s) to be used.
4. Operand address calculation (oac): If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
5. Operand fetch (of): Fetch the operand from memory or read it in from I/O.
6. Data operation (do): Perform the operation indicated in the instruction.
7. Operand store (os): Write the result into memory or out to I/O.

b) Interrupts

Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal processing of the processor. Interrupts are provided primarily as a way to improve processing efficiency. Table 1.2 lists the most common classes of interrupts.



Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure such as power failure or memory parity error.

Table 2 Classes of Interrupts

Figure 1.9a illustrates this state of affairs. The user program performs a series of WRITE calls interleaved with processing. Code segments 1, 2, and 3 refer to sequences of instructions that do not involve I/O. The WRITE calls are to an I/O program that is a system utility and that will perform the actual I/O operation. The I/O program consists of three sections:

1. A sequence of instructions, labeled 4 in the figure, to prepare for the actual I/O operation. This may include copying the data to be output into a special buffer and preparing the parameters for a device command.
2. The actual I/O command. Without the use of interrupts, once this command is issued, the program must wait for the I/O device to perform the requested function (or periodically poll the device). The program might wait by simply repeatedly performing a test operation to determine if the I/O operation is done.
3. A sequence of instructions, labeled 5 in the figure, to complete the operation. This may include setting a flag indicating the success or failure of the operation.

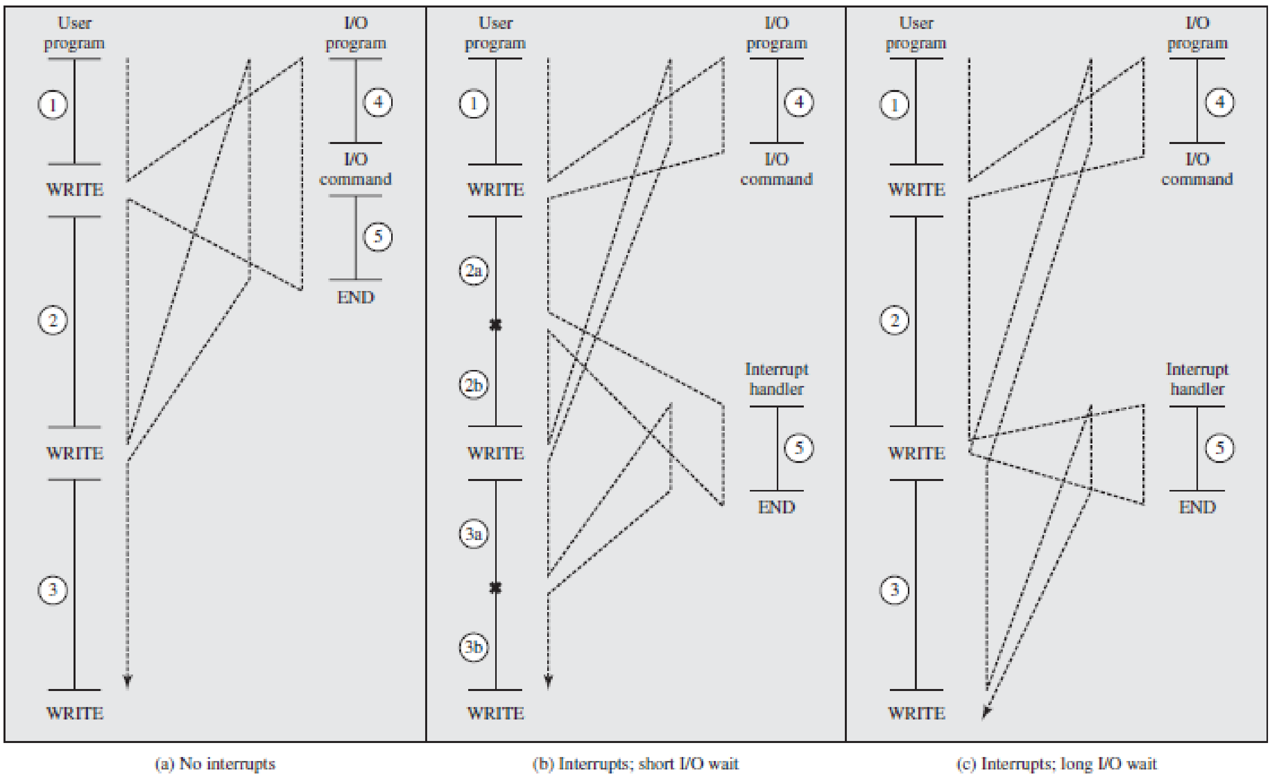


Figure 9 Program Flow of Control Without and With Interrupts

INTERRUPTS AND THE INSTRUCTION CYCLE With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress.
Consider the flow of control in Figure 9b. As before, the user program reaches a point at which it makes a system call in the form of a WRITE call. After these few instructions have been executed, control returns to the user program. Meanwhile, the external device is busy accepting data from computer memory and printing it. This I/O operation is conducted concurrently with the execution of instructions in the user program.
When the external device becomes ready to accept more data from the processor,—the I/O module for that external device sends an interrupt request signal to the processor. The processor responds by suspending operation of the current program, branching off to a program to service that particular I/O device, known as an interrupt handler, and resuming the original execution after the device is serviced. The points at which such interrupts occur are indicated by an asterisk in Figure 9b.
From the point of view of the user program, an interrupt is just that: an interruption of the normal sequence of execution. When the interrupt processing is completed, execution resumes (Figure 1.10).

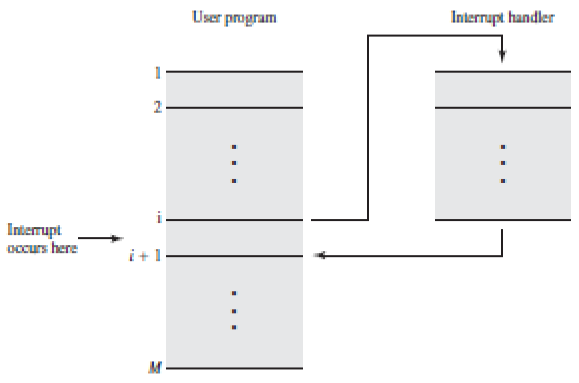


Figure 10 Transfer of Control via Interrupts

To accommodate interrupts, an interrupt cycle is added to the instruction cycle, as shown in Figure 11.

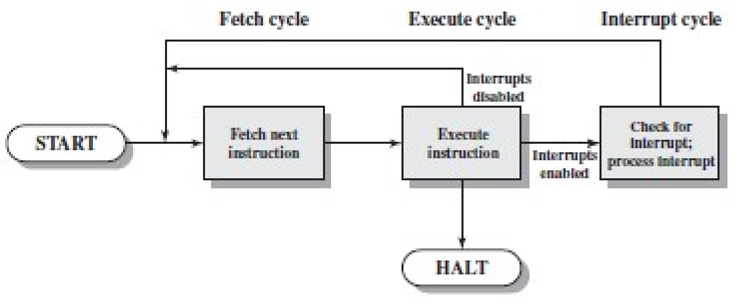


Figure 11Instruction Cycle with Interrupts

In the interrupt cycle, the processor checks to see if any interrupts have occurred. If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction of the current program. If an interrupt is pending, the processor does the following:

1. It suspends execution of the current program being executed and saves its context
2. It sets the program counter to the starting address of an interrupt handler routine.

The processor now proceeds to the fetch cycle and fetches the first instruction in the interrupt handler program, which will service the interrupt. When the interrupt handler routine is completed, the processor can resume execution of the user program at the point of interruption.

Consider Figure 12, which is a timing diagram based on the flow of control in Figures 9a and 9b. Figure 9c indicates this state of affairs. In this case, the user program reaches the second WRITE call before the I/O operation spawned by the first call is complete. The result is that the user program is hung up at that point. When the preceding I/O operation is completed, this new WRITE call may be processed, and a new I/O operation may be started. Figure 1.13 shows the timing for this situation with and without the use of interrupts. We can see that there is still a gain in efficiency because part of the time during which the I/O operation is underway overlaps with the execution of user instructions.

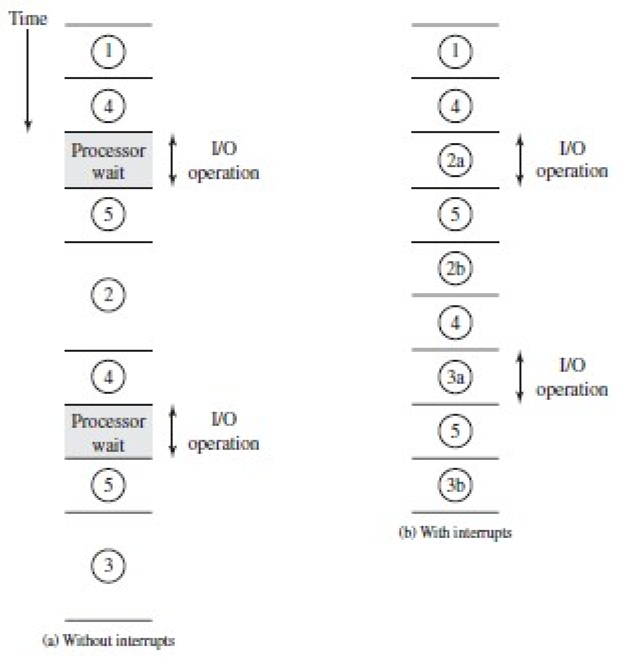


Figure 12 Program Timing: Short I/O Wait

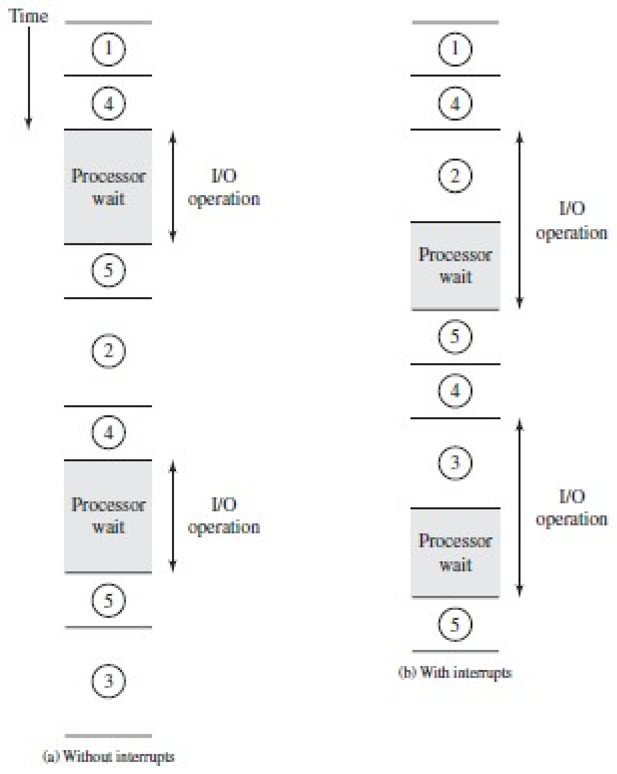


Figure 13 Program Timing: Long I/O Wait

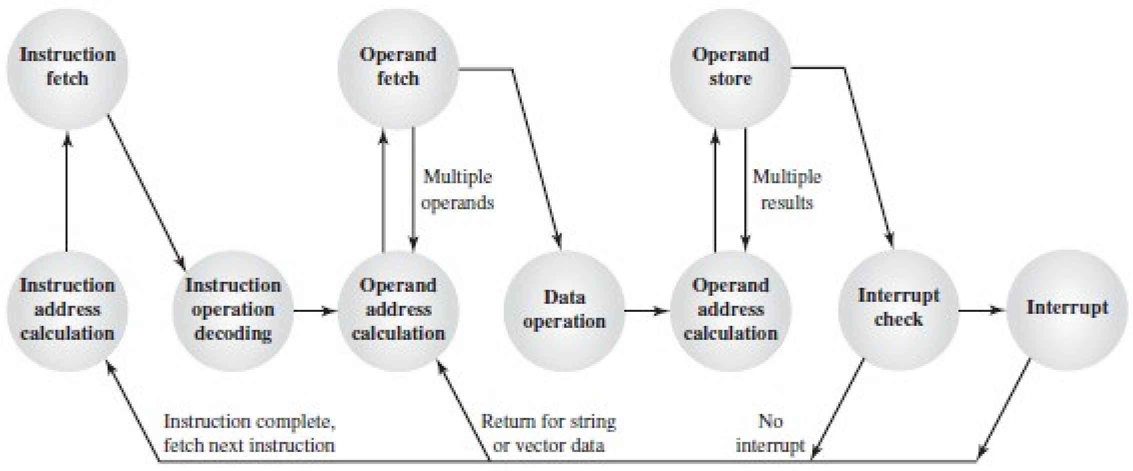

Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt

Figure 14 shows a revised instruction cycle state diagram that includes interrupt cycle processing

MULTIPLE INTERRUPTS Multiple interrupts can occur. Two approaches can be taken to dealing with multiple interrupts.
The first is to disable interrupts while an interrupt is being processed. A disabled interrupt simply means that the processor can and will ignore that interrupt request signal. Thus, when a user program is executing and an interrupt occurs, interrupts are disabled immediately. After the interrupt handler routine completes, interrupts are enabled before resuming the user program and the processor checks to see if additional interrupts have occurred. This approach is nice and simple, as interrupts are handled in strict sequential order (Figure 15a).
The drawback to the preceding approach is that it does not take into account relative priority or time-critical needs
A second approach is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to be itself interrupted (Figure 15b). As an example of this second approach, consider a system with three I/O devices: a printer, a disk, and a communications line, with increasing priorities of 2, 4, and 5, respectively.

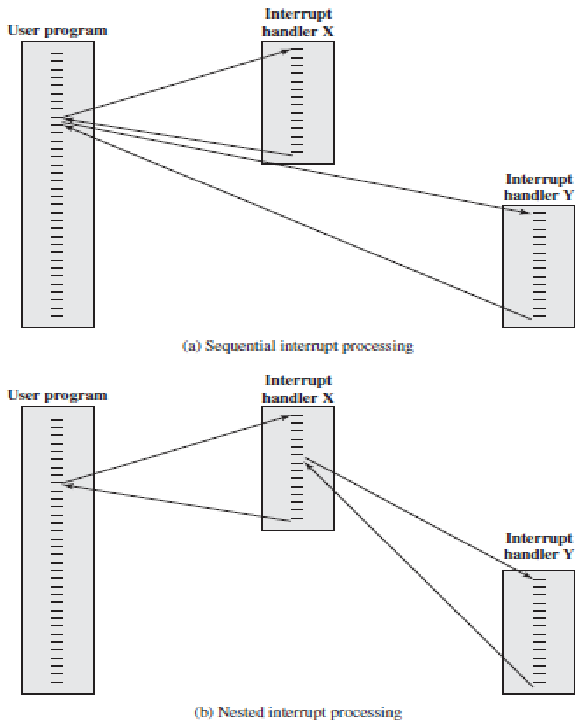


Figure 15 Transfer of Control with Multiple Interrupts

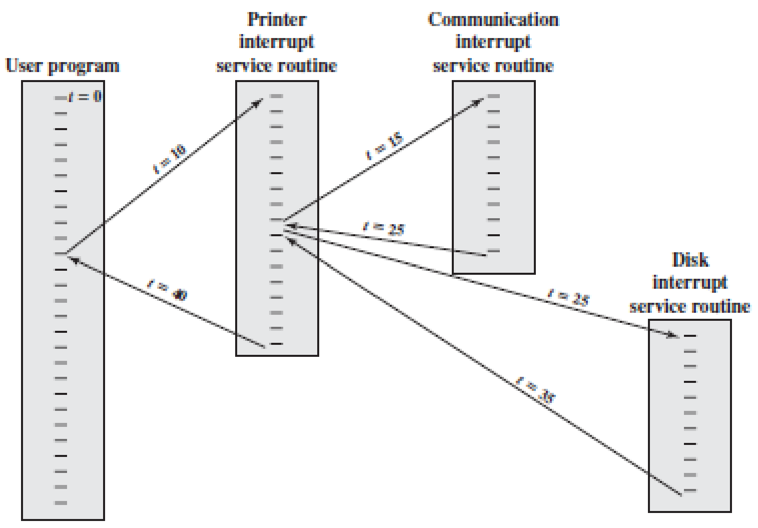

User program
Printer interrupt service routine
Communication interrupt service routine
Disk interrupt service routine
t = 0
t = 10
t = 15
t = 25
t = 25
t = 35
t = 40

Figure 16 Example Time Sequence of Multiple Interrupts

A user program begins at t = 0. At t = 10, a printer interrupt occurs; user information is placed on the system stack and execution continues at the printer interrupt service routine (ISR). While this routine is still executing, at t = 15, a communications interrupt occurs. Because the communications line has higher priority than the printer, the interrupt is honored. The printer ISR is interrupted, its state is pushed onto the stack,and execution continues at the communications ISR.While this routine is executing, a disk interrupt occurs (t = 20). Because this interrupt is of lower priority, it is simply held, and the communications ISR runs to completion.
When the communications ISR is complete (t = 25), the previous processor state is restored, which is the execution of the printer ISR. However, before even a single instruction in that routine can be executed, the processor honors the higher-priority disk interrupt and control transfers to the disk ISR. Only when that routine is complete (t = 35) is the printer ISR resumed. When that routine completes (t = 40), control finally returns to the user program.

I/O Function
An I/O module (e.g., a disk controller) can exchange data directly with the processor. Just as the processor can initiate a read or write with memory, designating the address of a specific location, the processor can also read data from or write data to an I/O module
In some cases, it is desirable to allow I/O exchanges to occur directly with memory. In such a case, the processor grants to an I/O module the authority to read from or write to memory, so that the I/O-memory transfer can occur without tying up the processor. During such a transfer, the I/O module issues read or write commands to memory, relieving the processor of responsibility for the exchange. This operation is known as direct memory access (DMA).

7. BUS INTERCONNECTION

A bus is a communication pathway connecting two or more devices. A key characteristic of a bus is that it is a shared transmission medium. Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus. If two devices transmit during the same time period, their signals will overlap and become garbled. Thus, only one device at a time can successfully transmit.

Typically, a bus consists of multiple communication pathways, or lines. Each line is capable of transmitting signals representing binary 1 and binary 0. An 8-bit unit of data can be transmitted over eight bus lines. A bus that connects major computer components (processor, memory, I/O) is called a system bus.

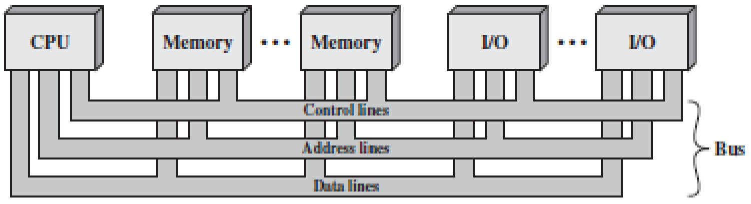


Figure 17Bus Interconnection Schemes

On any bus the lines can be classified into three functional groups (Figure 1.17): data, address, and control lines. In addition, there may be power distribution lines that supply power to the attached modules.

The **data lines** provide a path for moving data among system modules. These lines, collectively, are called the data bus.

The **address lines** are used to designate the source or destination of the data on the data bus. For example, on an 8-bit address bus, address 01111111 and below might reference locations in a memory module (module 0) with 128 words of memory, and address 10000000 and above refer to devices attached to an I/O module (module 1).

The **control lines** are used to control the access to and the use of the data and address lines. Control signals transmit both command and timing information among system modules. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed. Typical control lines include

1. Memory write: Causes data on the bus to be written into the addressed location
2. Memory read: Causes data from the addressed location to be placed on the bus
3. I/O write: Causes data on the bus to be output to the addressed I/O port
4. I/O read: Causes data from the addressed I/O port to be placed on the bus
5. Transfer ACK: Indicates that data have been accepted from or placed on the bus
6. Bus request: Indicates that a module needs to gain control of the bus
7. Bus grant: Indicates that a requesting module has been granted control of the bus
8. Interrupt request: Indicates that an interrupt is pending
9. Interrupt ACK: Acknowledges that the pending interrupt has been recognized
10. Clock: Is used to synchronize operations
11. Reset: Initializes all modules

The operation of the bus is as follows. If one module wishes to send data to another, it must do two things: (1) obtain the use of the bus, and (2) transfer data via the bus. If one module wishes to request data from another module, it must (1) obtain the use of the bus, and (2) transfer a request to the other module over the appropriate control and address lines. It must then wait for that second module to send the data.

The classic physical arrangement of a bus is depicted in Figure 18.

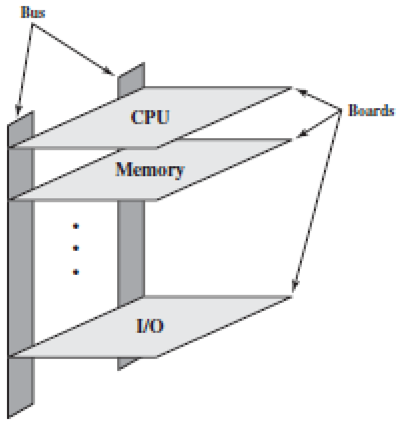


Figure 18 Typical Physical Realization of a Bus Architecture

In this example, the bus consists of two vertical columns of conductors. Each of the major system components occupies one or more boards and plugs into the bus at these slots. Thus, an on-chip bus may connect the processor and cache memory, whereas an on-board bus may connect the processor to main memory and other components.
This arrangement is most convenient. A small computer system may be acquired and then expanded later (more memory, more I/O) by adding more boards. If a component on a board fails, that board can easily be removed and replaced.
Multiple-Bus Hierarchies
If a great number of devices are connected to the bus, performance will suffer. There are two main causes:

1. In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay.
2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.

Most computer systems use multiple buses, generally laid out in a hierarchy. A typical traditional structure is shown in Figure 19a. There is a local bus that connects the processor to a cache memory and that may support one or more local devices The cache memory is connected to a system bus to which all of the main memory modules are attached. It is possible to connect I/O controllers directly onto the system bus. A more efficient solution is to make use of one or more expansion buses for this purpose. This arrangement allows the system to support a wide variety of I/O devices and at the same time insulate memory-to-processor traffic from I/O traffic.
Figure 19a shows some typical examples of I/O devices that might be attached to the expansion bus. Network connections include local area networks (LANs), wide area networks (WANs), SCSI (small computer system interface), serial port.. This traditional bus architecture is reasonably efficient but begins to break down as higher and higher performance is seen in the I/O devices. In response to these growing demands, a common approach taken by industry is to build a high-speed bus that is closely integrated with the rest of the system, requiring only a bridge between the processor's bus and the high-speed bus. This arrangement is sometimes known as a mezzanine architecture.
Figure 19b shows a typical realization of this approach. Again,there is a local bus that connects the processor to a cache controller, which is in turn connected to a system bus that supports main memory. The cache controller is integrated into a bridge, or buffering device, that connects to the high-speed bus. This bus supports connections to high-speed LANs, video and graphics workstation controllers, SCSI and FireWireLower-speed devices are still supported off an expansion bus, with

an interface buffering traffic between the expansion bus and the high-speed bus. The advantage of this arrangement is that the high-speed bus brings high-demand devices into closer integration with the processor and at the same time is independent of the processor.

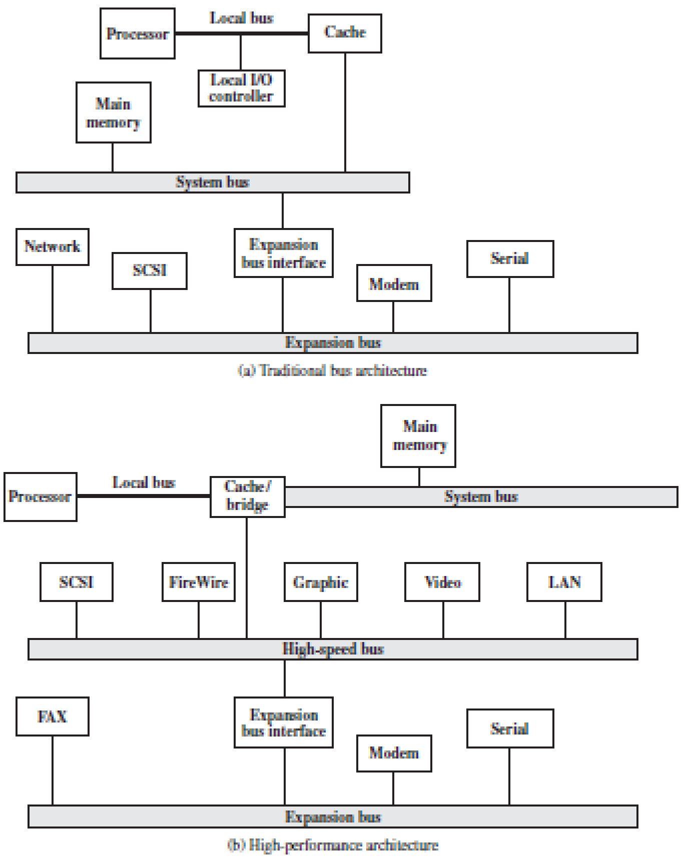


Figure 19 Example Bus Configuration

Elements of Bus Design

Figure 19 Example Bus Confihuration

There are a few design elements that serve to classify and differentiate buses. Table 1.3 lists key elements.

1. BUS TYPES Bus lines can be separated into two generic types: dedicated and multi- plexed. A dedicated bus line is permanently assigned either to one function or to a physical subset of computer components. Physical dedication refers to the use of multiple buses, each of which connects only a subset of modules. The potential advantage of physical dedication is high throughput, because there is less bus contention.A disadvantage is the increased size and cost of the system.
   Address and data information may be transmitted over the same set of lines using an Address Valid control line. At the beginning of a data transfer, the address is placed on the bus and the Address Valid line is activated. The address is then removed from the bus, and the same bus connections are used for the subsequent read or write data transfer. This method of using the same lines for multiple purposes is known as time multiplexing.The advantage of time multiplexing is the use of fewer lines, which saves space and, usually, cost. The disadvantage is that more complex circuitry is needed within each module.
2. METHOD OF ABITRATION. The various methods can be roughly classified as being either centralized or distributed. In a centralized scheme, a single hardware device, referred to as a bus controller or arbiter, is responsible for allocating time on the bus. In a distributed scheme, there is no central controller. Rather, each module contains access control logic and the modules act together to share the bus. With both methods of arbitration, the purpose is to

designate either the processor or an I/O module, as master. The master may then initiate a data transfer (e.g., read or write) with some other device, which acts as slave for this particular exchange.

3. TIMING Buses use either synchronous timing or asynchronous timing. With synchronous timing, the occurrence of events on the bus is determined by a clock. A single 1–0 transmission is referred to as a clock cycle or bus cycle and defines a time slot.

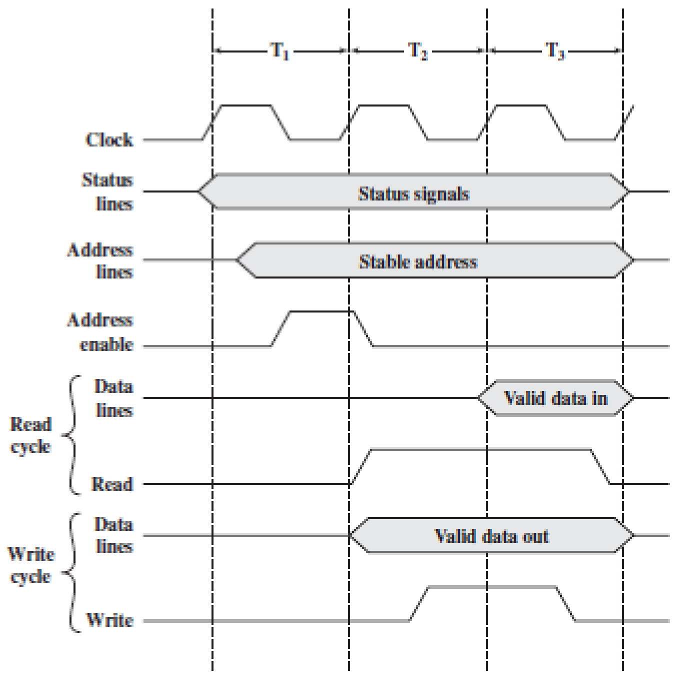


Figure 20 Timing of Synchronous Bus Operations

Figure 19 shows a typical, but simplified, timing diagram for synchronous read and write. In this simple example, the processor places a memory address on the address lines during the first clock cycle and may assert various status lines. Once the address lines have stabilized, the processor issues an address enable signal. For a read operation, the processor issues a read command at the start of the second cycle. A memory module recognizes the address and, after a delay of one cycle, places the data on the data lines. The processor reads the data from the data lines and drops the read signal. For a write operation, the processor puts the data on the data lines at the start of the second cycle, and issues a write command after the data lines have stabilized. The memory module copies the information from the data lines during the third clock cycle. With asynchronous timing, the occurrence of one event on a bus follows and depends on the occurrence of a previous event.

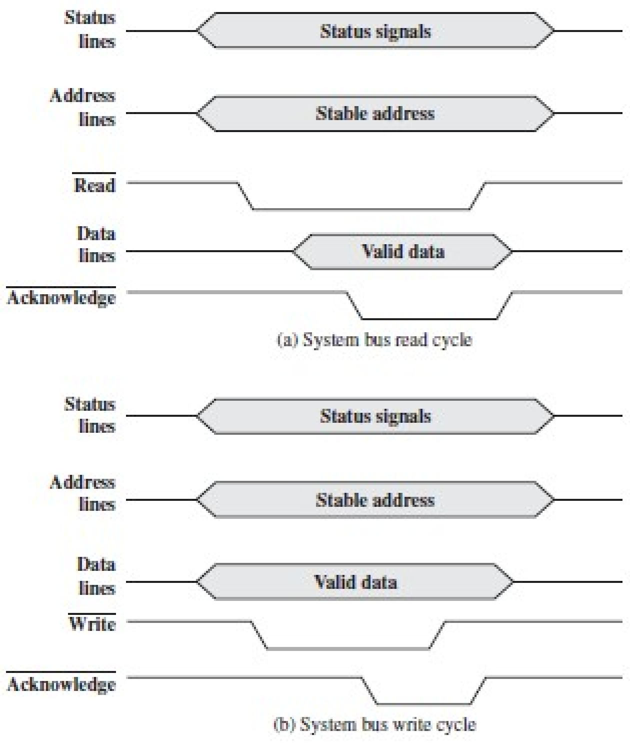


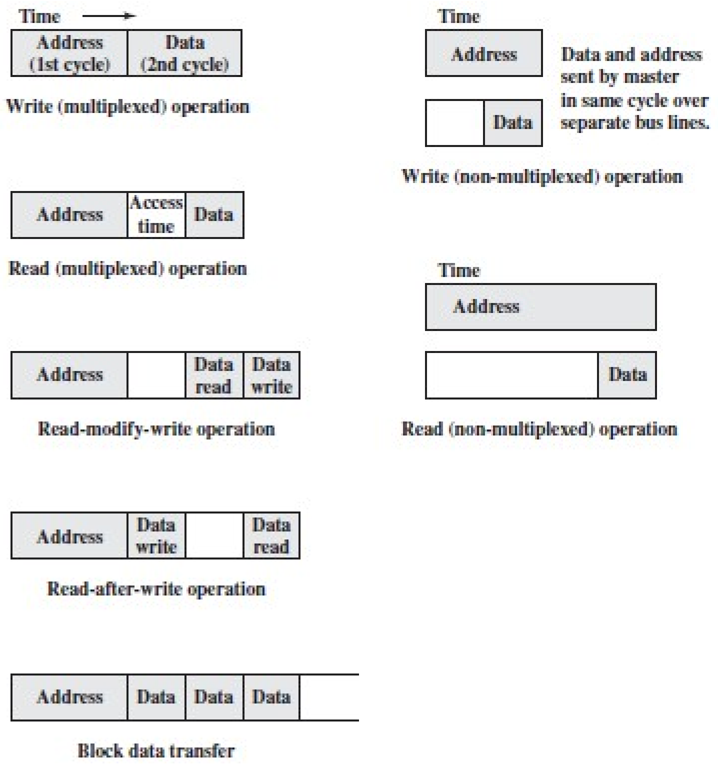
Figure 21 Timing of Asynchronous Bus Operations

In the simple read example of Figure 21a, the processor places address and status signals on the bus. After pausing for these signals to stabilize, it issues a read command, indicating the presence of valid address and control signals. The appropriate memory decodes the address and responds by placing the data on the data line. Once the data lines have stabilized, the memory module asserts the acknowledged line to signal the processor that the data are available. Once the master has read the data from the data lines, it deasserts the read signal. This causes the memory module to drop the data and acknowledge lines. Finally, once the acknowledge line is dropped, the master removes the address information.

Figure 21b shows a simple asynchronous write operation. In this case, the master places the data on the data line at the same time that is puts signals on the status and address lines. The memory module responds to the write command by copying the data from the data lines and then asserting the acknowledge line. The master then drops the write signal and the memory module drops the acknowledge signal

Synchronous timing is simpler to implement and test. However, it is less flexible than asynchronous timing. With asynchronous timing, a mixture of slow and fast devices, using older and newer technology, can share a bus.

- BUS WIDTH The width of the data bus has an impact on system performance: The wider the data bus, the greater the number of bits transferred at one time. The width of the address bus has an impact on system capacity: the wider the address bus, the greater the range of locations that can be referenced.
- DATA TRANSFER TYPE Finally, a bus supports various data transfer types, as illustrated in Figure 22.



Time
Address (1st cycle)
Data (2nd cycle)
Write (multiplexed) operation
Time
Address
Data
Data and address sent by master in same cycle over separate bus lines.
Write (non-multiplexed) operation
Address
Access time
Data
Read (multiplexed) operation
Time
Address
Data
Read (non-multiplexed) operation
Address
Data read
Data write
Read-modify-write operation
Address
Data write
Data read
Read-after-write operation
Address
Data
Data
Data
Block data transfer

Figure 22 Bus Data Transfer Types

In the case of a multiplexed address/data bus, the bus is first used for specifying the address and then for transferring the data. For a read operation, there is typically a wait while the data are being fetched from the slave to be put on the bus. For either a read or a write, there may also be a delay if it is necessary to go through arbitration to gain control of the bus for the remainder of the operation.

In the case of dedicated address and data buses, the address is put on the address bus and remains there while the data are put on the data bus. For a write operation, the master puts the data onto the data bus as soon as the address has stabilized and the slave has had the opportunity to recognize its address. For a read operation, the slave puts the data onto the data bus as soon as it has recognized its address and has fetched the data.

A read–modify–write operation is simply a read followed immediately by a write to the same address

Read-after-write is an indivisible operation consisting of a write followed immediately by a read from the same address.

Some bus systems also support a block data transfer. The first data item is transferred to or from the specified address; the remaining data items are transferred to or from subsequent addresses

8. PCI (PERIPHERAL COMPONENT INTERCONNECT)

The peripheral component interconnect (PCI) is a popular high-bandwidth, processor-independent bus that can function as a peripheral bus. The current standard allows the use of up to 64 data lines at 66 MHz, for a raw transfer rate of 528 MByte/s, or 4.224 Gbps. It requires very few chips to implement and supports other buses attached to the PCI bus.

Intel began work on PCI in 1990 for its Pentium-based systems. The industry association, the PCI Special Interest Group (SIG), developed and further and maintained the compatibility of the PCI specifications. PCI is designed to support a variety of microprocessor-based configurations, including both single- and multiple-processor systems. It makes use of synchronous timing and a centralized arbitration scheme.

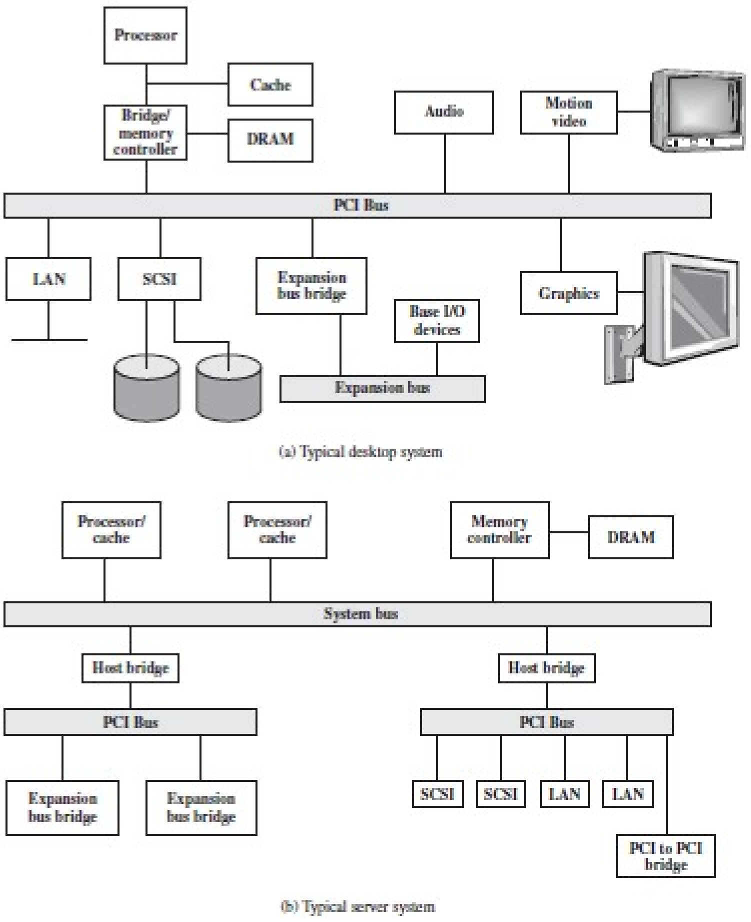


Figure 23 Example PCI Configurations

Figure 23a shows a typical use of PCI in a single-processor system. The bridge acts as a data buffer so that the speed of the PCI bus may differ from that of the processor's I/O capability. In a multiprocessor system (Figure 1.23b), one or more PCI configurations may be connected by bridges to the processor's system bus. Again, the use of bridges keeps the PCI independent of the processor speed yet provides the ability to receive and deliver data rapidly.

Bus Structure

PCI may be configured as a 32- or 64-bit bus. There are 49 mandatory signal lines for PCI which are divided into the following functional groups:

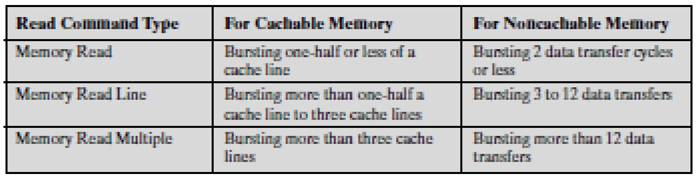
• System pins: Include the clock and reset pins.
• Address and data pins: Include 32 lines that are time multiplexed for addresses and data.
The other lines in this group are used to interpret and validate the signal lines that carry the addresses and data.

• Interface control pins: Control the timing of transactions and provide coordination among initiators and targets.
• Arbitration pins: Unlike the other PCI signal lines, these are not shared lines. Rather, each PCI master has its own pair of arbitration lines that connect it directly to the PCI bus arbiter.
• Error reporting pins: Used to report parity and other errors.
In addition, the PCI specification defines 51 optional signal lines, divided into the following functional groups:
• Interrupt pins: These are provided for PCI devices that must generate requests for service. As with the arbitration pins, these are not shared lines. Rather, each PCI device has its own interrupt line or lines to an interrupt controller.
• Cache support pins: These pins are needed to support a memory on PCI that can be cached in the processor or another device.
• 64-bit bus extension pins: Include 32 lines that are time multiplexed for ad dresses and data and that are combined with the mandatory address/data lines to form a 64-bit address/data bus.
• JTAG/boundary scan pins: These signal lines support testing procedures.

PCI Commands
Bus activity occurs in the form of transactions between an initiator, or master, and a target. When a bus master acquires control of the bus, it determines the type of transaction that will occur next The commands are as follows:

- Interrupt Acknowledge
- Special Cycle
- I/O Read
- I/O Write
- Memory Read
- Memory Read Line
- Memory Read Multiple
- Memory Write
- Memory Write and Invalidate
- Configuration Read
- Configuration Write
- Dual address Cycle

1. Interrupt Acknowledge is a read command intended for the device that functions as an interrupt controller on the PCI bus.
2. The Special Cycle command is used by the initiator to broadcast a message to one or more targets.
3. The I/O Read and Write commands are used to transfer data between the initiator and an I/O controller.
4. The memory read and write commands are used to specify the transfer of a burst of data, occupying one or more clock cycles. The three memory read commands have the uses outlined in Table 1.4



Read Command Type	For Cachable Memory	For Noncachable Memory
Memory Read	Bursting one-half or less of a cache line	Bursting 2 data transfer cycles or less
Memory Read Line	Bursting more than one-half a cache line to three cache lines	Bursting 3 to 12 data transfers
Memory Read Multiple	Bursting more than three cache lines	Bursting more than 12 data transfers

Table 1.4
Interpretations of PCI Read Commands

5. The Memory Write command is used to transfer data in one or more data cycles to memory.
6. The Memory Write and Invalidate command transfers data in one or more cycles to memory. In addition, it guarantees that at least one cache line is written.
The two configuration commands enable a master to read and update configuration parameters in a device connected to the PCI.
The Dual Address Cycle command is used by an initiator to indicate that it is using 64-bit addressing.

Data Transfers
Every data transfer on the PCI bus is a single transaction consisting of one address phase and one or more data phases.
Figure 24 shows the timing of the read transaction. All events are synchronized to the falling transitions of the clock, which occur in the middle of each clock cycle. Bus devices sample the bus lines on the rising edge at the beginning of a bus cycle. The following are the significant events, labeled on the diagram:

a) Once a bus master has gained control of the bus, it may begin the transaction by asserting FRAME. This line remains asserted until the initiator is ready to complete the last data phase. The initiator also puts the start address on the address bus, and the read command on the C/BE lines.
b) At the start of clock 2, the target device will recognize its address on the AD lines.
c) The initiator ceases driving the AD bus. A turnaround cycle (indicated by the two circular arrows) is required on all signal lines that may be driven by more than one device, so that the dropping of the address signal will prepare the bus for use by the target device. The initiator changes the information on the C/BE lines to designate which AD lines are to be used for transfer for the currently addressed data (from 1 to 4 bytes). The initiator also asserts IRDY to indicate that it is ready for the first data item.
d) The selected target asserts DEVSEL to indicate that it has recognized its address and will respond. It places the requested data on the AD lines and asserts TRDY to indicate that valid data are present on the bus.
e) The initiator reads the data at the beginning of clock 4 and changes the byte enable lines as needed in preparation for the next read.
f) In this example, the target needs some time to prepare the second block of data for transmission. Therefore, it deasserts TRDY to signal the initiator that there will not be new data during the coming cycle.Accordingly,the initiator does not read the data lines at the beginning of the fifth clock cycle and does not change byte enable during that cycle. The block of data is read at beginning of clock 6.
g) During clock 6, the target places the third data item on the bus. However, in this example, the initiator is not yet ready to read the data item (e.g., it has a temporary buffer full condition). It therefore deasserts IRDY. This will cause the target to maintain the third data item on the bus for an extra clock cycle.
h) The initiator knows that the third data transfer is the last, and so it deasserts FRAME to signal the target that this is the last data transfer. It also asserts IRDY to signal that it is ready to complete that transfer.
i) The initiator deasserts IRDY, returning the bus to the idle state, and the target deasserts TRDY and DEVSEL.

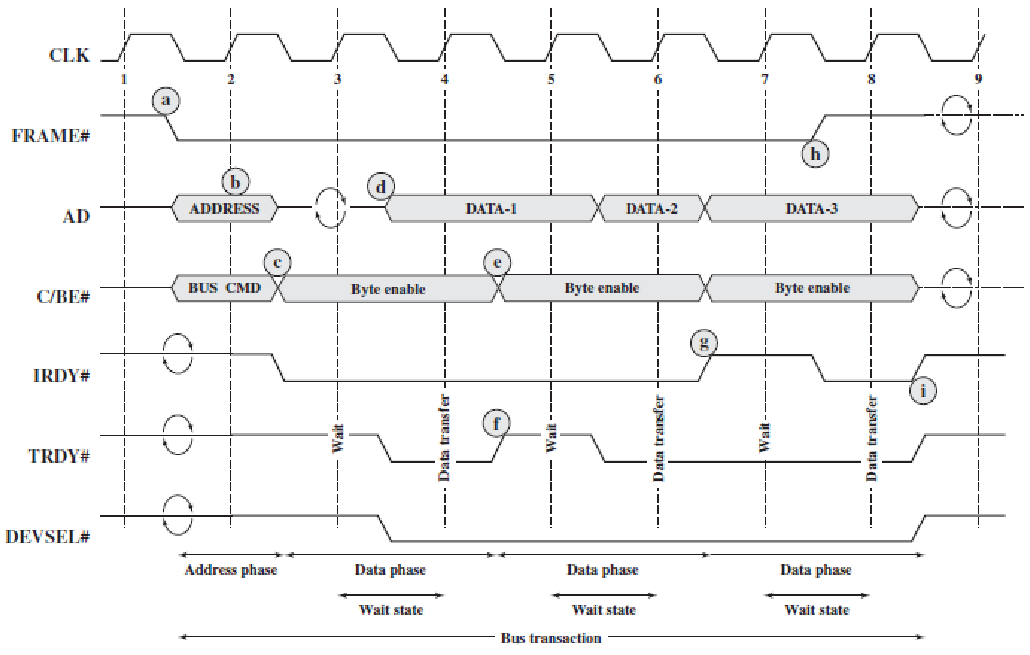


Figure 1.24 PCI Read Operation

Arbitration

PCI makes use of a centralized, synchronous arbitration scheme in which each master has a unique request (REQ) and grant (GNT) signal. These signal lines are attached to a central arbiter (Figure 25) and a simple request–grant handshake is used to grant access to the bus.

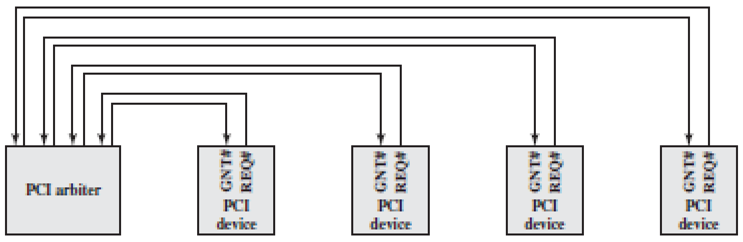


Figure 25 PCI Bus Arbiter

Figure 26 is an example in which devices A and B are arbitrating for the bus. The following sequence occurs:

a. At some point before the start of clock 1, A has asserted its REQ signal.

b. During clock cycle 1, B requests use of the bus by asserting its REQ signal.

c. At the same time, the arbiter asserts GNT-A to grant bus access to A.

d. Bus master A samples GNT-A at the beginning of clock 2 and learns that it has been granted bus access. It also finds IRDY and TRDY deasserted, which indicates that the bus is idle. It also continues to assert REQ-A, because it has a second transaction to perform after this one.

e. The bus arbiter samples all REQ lines at the beginning of clock 3 and makes an arbitration decision to grant the bus to B for the next transaction. It then asserts GNT-B and deasserts GNT-A. B will not be able to use the bus until it returns to an idle state.

f. A deasserts FRAME to indicate that the last data transfer is in progress.It puts the data on the data bus and signals the target with IRDY.The target reads the data at the beginning of the next clock cycle.

g. At the beginning of clock 5, B finds IRDY and FRAME deasserted and so is able to take control of the bus by asserting FRAME. It also deasserts its REQ line, because it only wants to perform one transaction.Subsequently, master A is granted access to the bus for its next transaction.

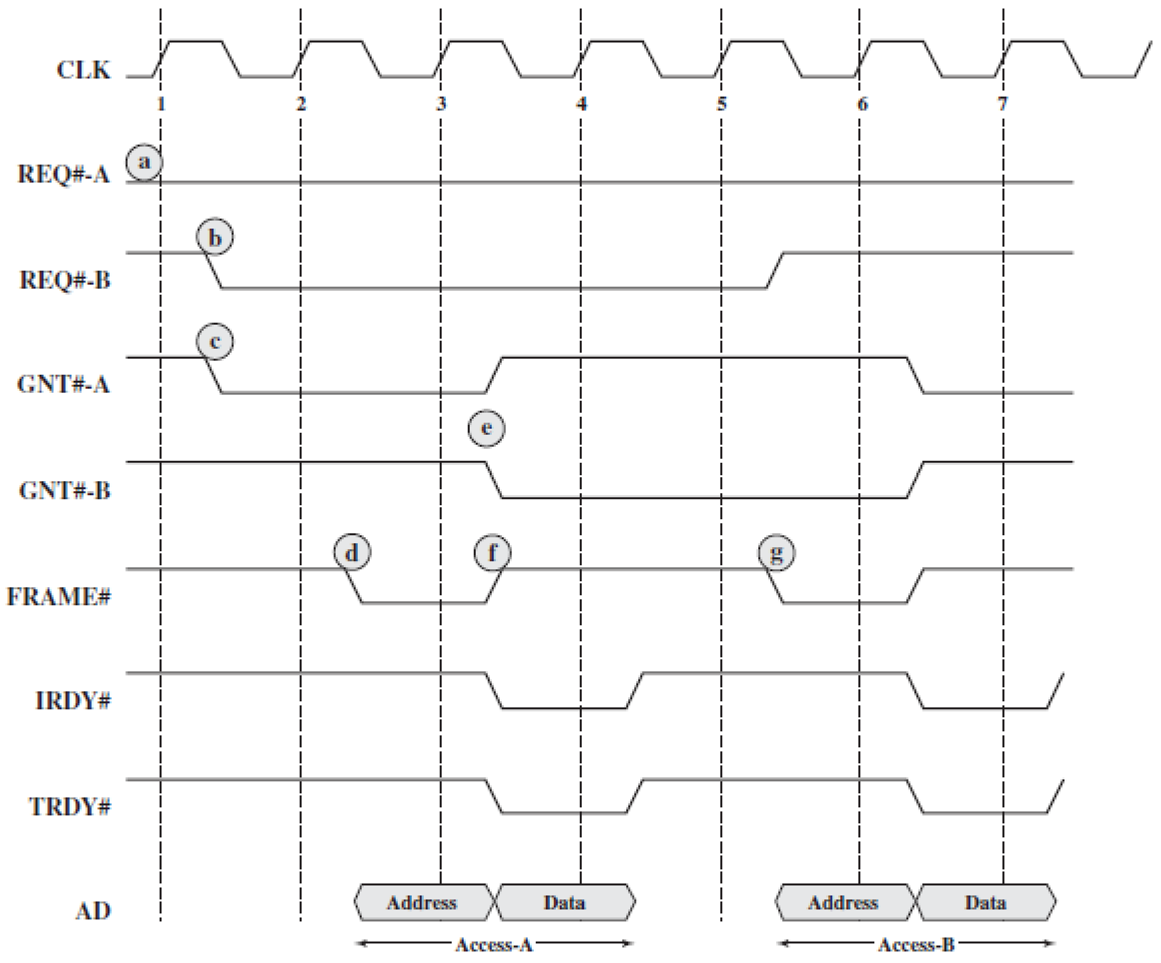


Figure 26 PCI Bus Arbitration Between Two Masters