Computer Structure

1

Computer Structure

NLU-FIT

Chapter 02

Top Level View of Computer Function and Interconnection

KEY POINTS

2

Computer Structure

NLU-FIT

- The major computer system components (processor, main memory, I/O modules) need to be interconnected in order to exchange data and control signals.
  - The most popular means of interconnection is the use of a shared system bus consisting of multiple lines.
  - In contemporary systems, there typically is a hierarchy of buses to improve performance.
- Key design elements for buses include
  - Arbitration (whether permission to send signals on bus lines is controlled centrally or in a distributed fashion);
  - Timing (whether signals on the bus are synchronized to a central clock or are sent asynchronously based on the most recent transmission);
  - And width (number of address lines and number of data lines).

3.1. Computer Components

3 Computer Structure NLU-FIT

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

3.1. Computer Components

4 Computer Structure NLU-FIT

- Program Concept
  - There is a small set of basic logic components that can be combined in various ways to store binary data and to perform arithmetic and logical operations on that data.
  - If there is a particular computation to be performed
    - ✓a configuration of logic components designed specifically for that computation could be constructed.
    - ✓We can think of the process of connecting the various components inthe desired configuration as a form of programming.
  - The resulting "program" is in the form of hardware and is termed a *hardwired program.*

3.1. Computer Components

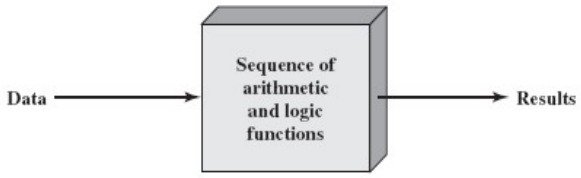


Figure 3.1 a. Programming in hardware

- Hardwired systems are inflexible
- General purpose hardware can do different tasks, given correct control signals
- Instead of re-wiring, supply a new set of control signals

3.1. Computer Components

- What is a program?
  - The entire program is actually a sequence of steps. At each step, some arithmetic or logical operation is performed on some data.
  - Programming is now much easier. Instead of rewiring the hardware for each new program, all we need to do is provide a new sequence of codes. Each code is, in effect, an instruction, and part of the hardware interprets each instruction and generates control signals
  - A sequence of codes or instructions is called *software.*

3.1. Computer Components

7

Computer Structure

NLU-FIT

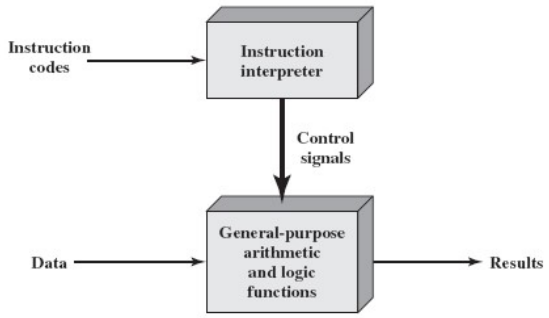


Figure 3.1b. Programming in software

3.1. Computer Components

8

Computer Structure

NLU-FIT

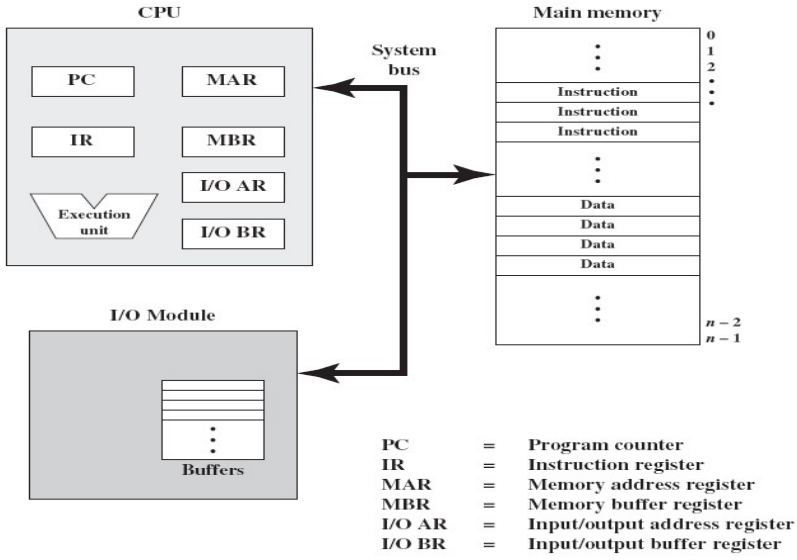


Figure 3.2. Computer Components:Top-Level View

3.1. Computer Components

- The CPU exchanges data with memory. 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.
  - 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/O AR) specifies a particular I/O device. An I/O buffer (I/O BR) register is used for the exchange of data between an I/O module and the CPU.

3.1. Computer Components

- 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.

3.2. Computer Function

11

Computer Structure

NLU-FIT

- 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.
  - Executes each instruction.
- Program execution consists of repeating the process of ***instruction fetch*** and ***instruction execution***.

3.2. Computer Function

12

Computer Structure

NLU-FIT

- The processing required for a single instruction is called an ***instruction cycle***.
- 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.

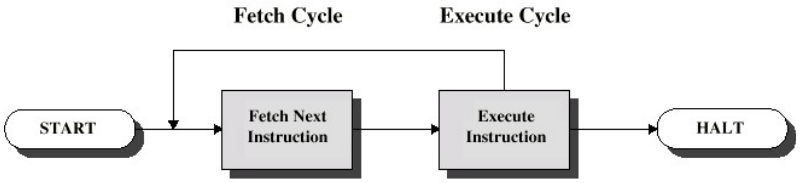


Figure 3.3. Basic Instruction Cycle

3.2.1. Instruction Fetch and Execute

13

Computer Structure

NLU-FIT

- At the beginning of each instruction cycle, the processor fetches an instruction from memory.
- In a typical processor, a register called the program counter (PC) holds the address of the instruction to be fetched next.
- Unless told otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence.

3.2.1. Instruction Fetch and Execute

14

Computer Structure

NLU-FIT

- The fetched instruction is loaded into a register in the processor known as the instruction register (IR).
- The instruction contains bits that specify the action the processor is to take.
- The processor interprets the instruction and performs the required action.
- In general, these actions fall into four categories:

3.2.1. Instruction Fetch and Execute

15

Computer Structure

NLU-FIT

- **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.
- **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
- **Data processing:** The processor may perform some arithmetic or logic operation on data.
- **Control:** An instruction may specify that the sequence of execution be altered. ***E.g jump***
  - ✓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.

3.2.1. Instruction Fetch and Execute

16

Computer Structure

NLU-FIT

- 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 3.4
  - 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.
    - ✓up to 2^{12} =4096 (4K) words of memory can be directly addressed.

3.2.1. Instruction Fetch and Execute

17

Computer Structure

NLU-FIT

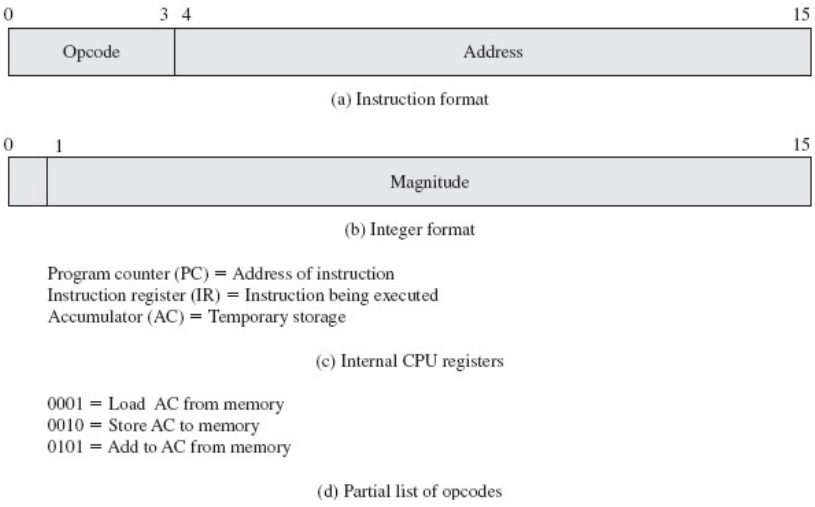


Figure 3.4. Characteristics of a Hypothetical Machine

3.2.1. Instruction Fetch and Execute

18

Computer Structure

NLU-FIT

- Figure 3.5 illustrates a partial program execution, showing the relevant portions of memory and processor registers.
- 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 the latter location.
- Three instructions, which can be described as three fetch and three execute cycles, are required:

3.2.1. Instruction Fetch and Execute

19

Computer Structure

NLU-FIT

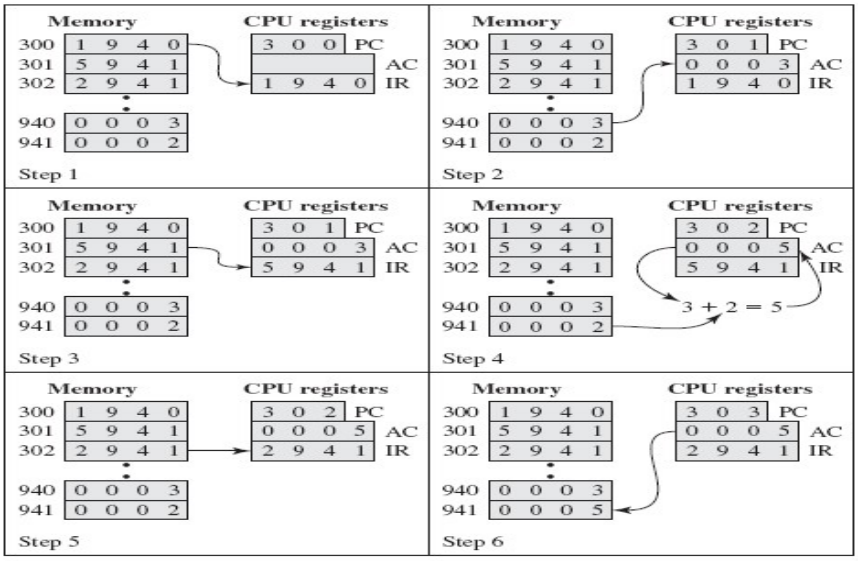


Figure 3.5 Example of Program Execution

3.2.1. Instruction Fetch and Execute

20

Computer Structure

NLU-FIT

- Three instructions, which can be described as three fetch and three execute cycles, are required:
  - 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.
  - 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.

3.2.1. Instruction Fetch and Execute

- The next instruction (5941) is fetched from location 301 and the PC is incremented.
- The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
- The next instruction (2941) is fetched from location 302 and the PC is incremented.
- The contents of the AC are stored in location 941.

- The execution cycle for a particular instruction may involve more than one reference to memory. Also, instead of memory references, an instruction may specify an I/O operation.

3.2.1. Instruction Fetch and Execute

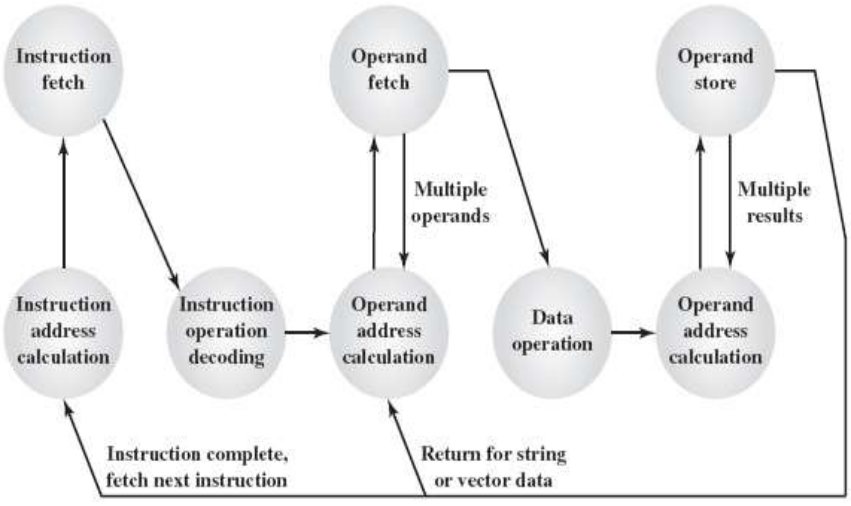


Figure 3.6. Instruction Cycle State Diagram

3.2.1. Instruction Fetch and Execute

- **Instruction fetch (if)**: Read instruction from its memory location into the processor.
- **Instruction operation decoding (iod)**: Analyze instruction to determine type of operation to be performed and operand(s) to be used.
- **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.
- **Operand fetch (of)**: Fetch the operand from memory or read it in from I/O.
- **Data operation (do)**: Perform the operation indicated in the instruction.
- **Operand store (os)**: Write the result into memory or out to I/O.

3.2.1. Instruction Fetch and Execute

- States in the upper part of Figure 3.6 involve an exchange between the processor and either memory or an I/O module.
- States in the lower part of the diagram involve only internal processor operations.
- The **oac** state appears twice, because an instruction may involve a read, a write, or both. However, the action performed during that state is fundamentally the same in both cases.
- A single instruction can specify an operation to be performed on a vector (one-dimensional array) of numbers or a string (one-dimensional array) of characters.

3.2.2. Interrupts

- Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal processing of the processor.

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.

3.2.2. Interrupts

- Interrupts are provided primarily as a way to improve processing efficiency.
- For example:
  - 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:

3.2.2. Interrupts

27

Computer Structure

NLU-FIT

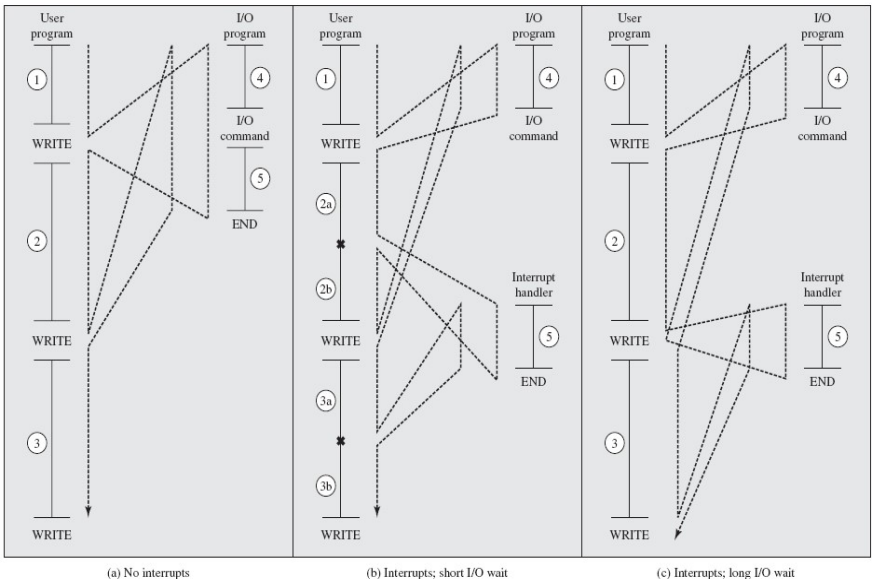


Figure 3.7. Program Flow of Control

3.2.2. Interrupts

28

Computer Structure

NLU-FIT

- ✓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.
- ✓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.
- ✓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.

3.2.2. Interrupts

29

Computer Structure

NLU-FIT

- From the point of view of the user program, an interrupt is just that: an interruption of the normal sequence of execution.
- Thus, the user program does not have to contain any special code to accommodate interrupts; the processor and the operating system are responsible for suspending the user program and then resuming it at the same point.

3.2.2. Interrupts

30

Computer Structure

NLU-FIT

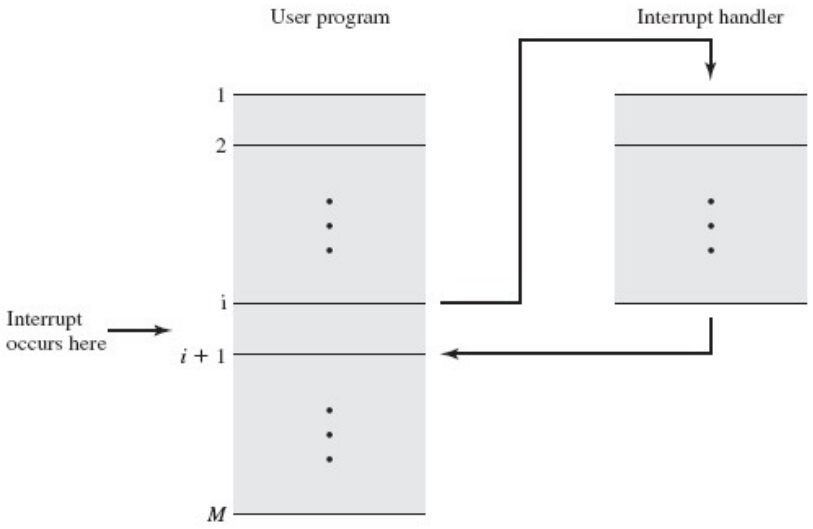


Figure 3.8 Transfer of Control via Interrupts

3.2.2. Interrupts

- To accommodate interrupts, an *interrupt cycle is added to the instruction cycle*.
- In the interrupt cycle, the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.
- 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:

3.2.2. Interrupts

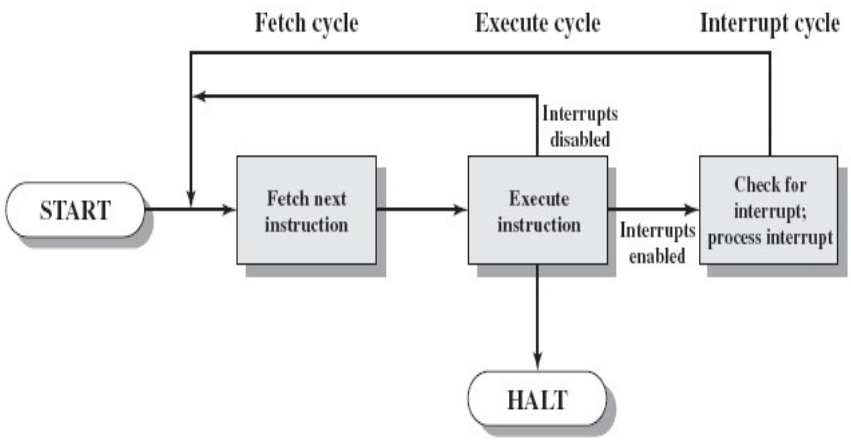


Figure 3.9. Instruction Cycle with Interrupts

3.2.2. Interrupts

- It suspends execution of the current program being executed and saves its context. This means saving the address of the next instruction to be executed (current contents of the program counter) and any other data relevant to the processor's current activity.
- It sets the program counter to the starting address of an *interrupt handler routine*.

- The interrupt handler program is generally part of the operating system.Typically, this program determines the nature of the interrupt and performs whatever actions are needed.

3.2.2. Interrupts

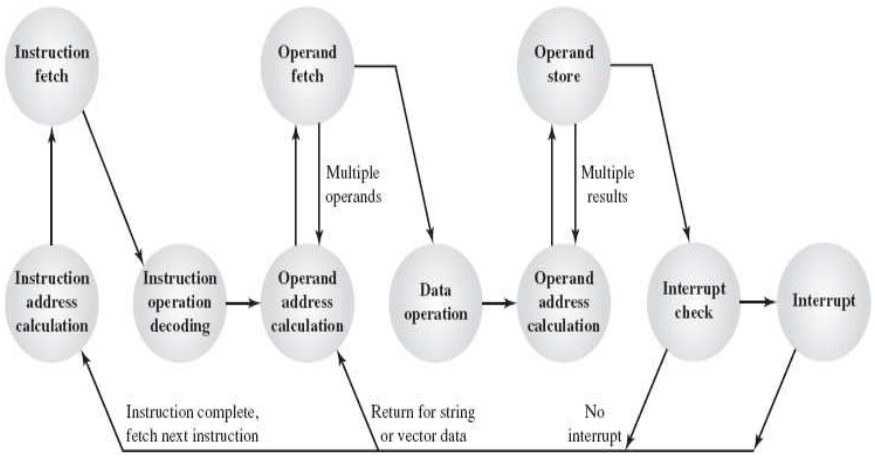


Figure 3.12. Instruction Cycle State Diagram, with Interrupts

3.2.2. Interrupts

35

Computer Structure

NLU-FIT

MULTIPLE INTERRUPTS

- For example, a program may be receiving data from a communications line and printing results.
  - The printer will generate an interrupt every time that it completes a print operation.
  - The communication line controller will generate an interrupt every time a unit of data arrives.
- Two approaches can be taken to dealing with multiple interrupts

3.2.2. Interrupts

36

Computer Structure

NLU-FIT

- 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.
  - If an interrupt occurs during this time, it generally remains pending and will be checked by the processor after the processor has enabled interrupts.
  - Thus, when a user program is executing and an interrupt occurs, interrupts are disabled immediately.

3.2.2. Interrupts

37

Computer Structure

NLU-FIT

- 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

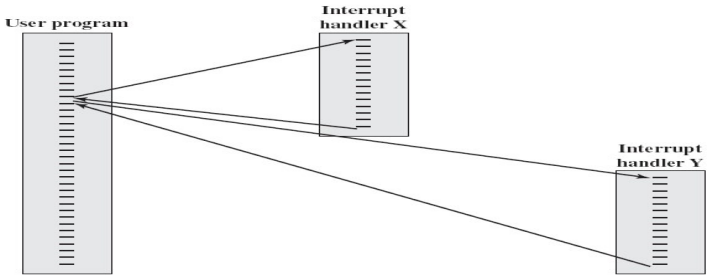


Figure 3.1.3 a. Sequential interrupt processing

3.2.2. Interrupts

38

Computer Structure

NLU-FIT

- A second approach is to define priorities for interrupts.
  - Low priority interrupts can be interrupted by higher priority interrupts
  - When higher priority interrupt has been processed, processor returns to previous interrupt

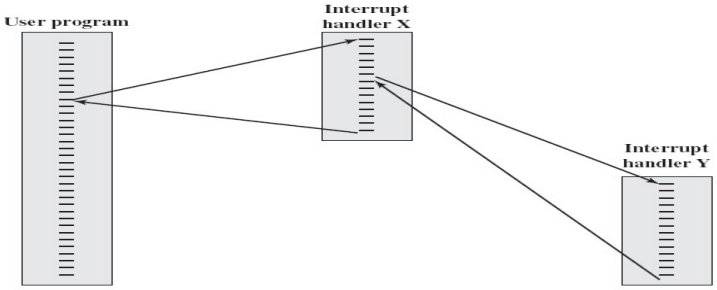


Figure 3.13 b. Nested interrupt processing

3.3. Interconnection Structure

39

Computer Structure

NLU-FIT

- A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other. Thus, there must be paths for connecting the modules
- The collection of paths connecting the various modules is called the interconnection structure.
- The design of this structure will depend on the exchanges that must be made among modules.

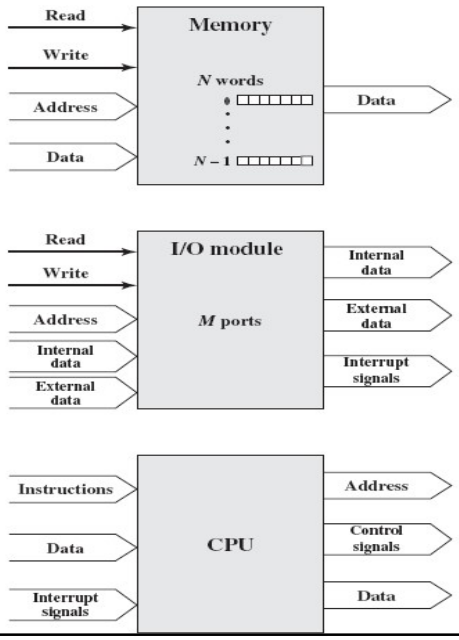
3.3. Interconnection Structure

40

Computer Structure

NLU-FIT

Figure 3.15
Computer Modules



3.3. Interconnection Structure

- **Memory**:
  - Typically, a memory module will consist of N words of equal length.
  - Each word is assigned a unique numerical address (0, 1, . . . ,N – 1).
  - A word of data can be read from or written into the memory.The nature of the operation is indicated by read and write control signals.
  - The location for the operation is specified by an address.

3.3. Interconnection Structure

- **I/O module**:
  - From an internal (to the computer system) point of view, I/O is functionally similar to memory.There are two operations, read and write.
  - Further, an I/O module may control more than one external device.
  - We can refer to each of the interfaces to an external device as a *port and give each a unique* address (e.g., 0, 1, . . . ,M– 1).
  - In addition, there are external data paths for the input and output of data with an external device.
  - Finally, an I/O module may be able to send interrupt signals to the processor.

3.3. Interconnection Structure

- **Processor:**
  - The processor reads in instructions and data, writes out data after processing, and uses control signals to control the overall operation of the system. It also receives interrupt signals.
- The interconnection structure must support the following types of transfers:
  - **Memory to processor**: The processor reads an instruction or a unit of data from memory.
  - **Processor to memory**: The processor writes a unit of data to memory.

3.3. Interconnection Structure

- **I/O to processor**:The processor reads data from an I/O device via an I/O module.
- **Processor to I/O**: The processor sends data to the I/O device.
- **I/O to or from memory**: For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA).

- Over the years, a number of interconnection structures have been tried. By far the most common is the bus and various multiple-bus structures.

3.4. 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.

3.4. Bus InterConnection

- Typically, a bus consists of multiple communication pathways, or lines.
- Each line is capable of transmitting signals representing binary 1 and binary 0.
- Over time, a sequence of binary digits can be transmitted across a single line.
- Taken together, several lines of a bus can be used to transmit binary digits simultaneously (in parallel). For example, an 8-bit unit of data can be transmitted over eight bus lines.

3.4. Bus InterConnection

47

Computer Structure

NLU-FIT

- Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy.
- A bus that connects major computer components (processor, memory, I/O) is called a system bus.
- The most common computer interconnection structures are based on the use of one or more system buses.

3.4.1. Bus Structure

48

Computer Structure

NLU-FIT

- Although there are many different bus designs, on any bus the lines can be classified into three functional groups (Figure 3.16): data, address, and control lines.
- In addition, there may be power distribution lines that supply power to the attached modules
- Power lines may not be shown

3.4.1. Bus Structure

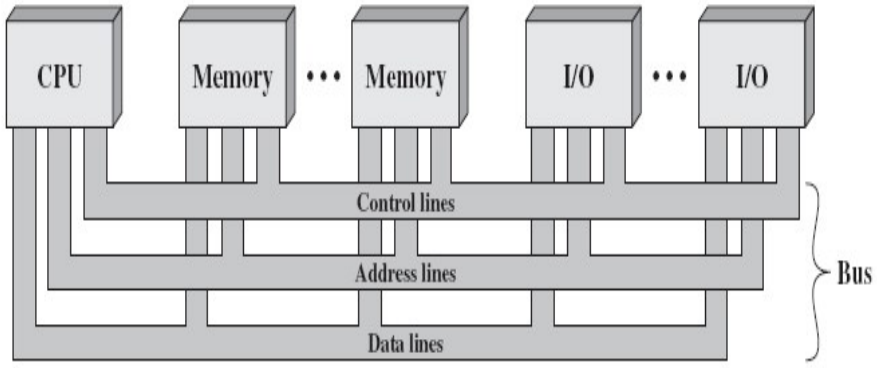


Figure 3.16. Bus Interconnection Scheme

3.4.1. Bus Structure

- The **data lines- Data bus**
  - Provide a path for moving data among system modules.
  - These lines, collectively, are called the data bus.The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the width of the data bus.
  - Because each line can carry only 1 bit at a time, the number of lines determines how many bits can be transferred at a time.
  - The width of the data bus is a key factor in determining overall system performance.
    - ✓For example, if the data bus is 32 bits wide and each instruction is 64 bits long, then the processor must access the memory module twice during each instruction cycle.

3.4.1. Bus Structure

51

Computer Structure

NLU-FIT

- The **Address lines – Address bus**
  - Are used to designate the source or destination of the data on the data bus.
  - For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines.
  - Clearly, the width of the address bus determines the maximum possible memory capacity of the system.

3.4.1. Bus Structure

52

Computer Structure

NLU-FIT

- Furthermore, the address lines are generally also used to address I/O ports.
- Typically, the higher-order bits are used to select a particular module on the bus, and the lower-order bits select a memory location or I/O port within the module.
  - ✓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).

3.4.1. Bus Structure

- The **Control lines – Control bus**
  - Are used to control the access to and the use of the data and address lines.
  - Because the data and address lines are shared by all components, there must be a means of controlling their use.
  - 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.

3.4.1. Bus Structure

- Typical control lines include
  - **Memory write**: Causes data on the bus to be written into the addressed location
  - **Memory read**: Causes data from the addressed location to be placed on the bus
  - **I/O write**: Causes data on the bus to be output to the addressed I/O port
  - **I/O read**: Causes data from the addressed I/O port to be placed on the bus
  - **Transfer ACK**: Indicates that data have been accepted from or placed on the bus

3.4.1. Bus Structure

- **Bus request**: Indicates that a module needs to gain control of the bus
- **Bus grant**: Indicates that a requesting module has been granted control of the bus
- **Interrupt request**: Indicates that an interrupt is pending
- **Interrupt ACK**: Acknowledges that the pending interrupt has been recognized
- **Clock**: Is used to synchronize operations
- **Reset**: Initializes all modules

3.4.1. Bus Structure

- 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.

3.4.1. Bus Structure

57

Computer Structure

NLU-FIT

- Physically, the system bus is actually a number of parallel electrical conductors.
- In the classic bus arrangement, these conductors are metal lines etched in a card or board (printed circuit board).
- The bus extends across all of the system components, each of which taps into some or all of the bus lines-Figure 3.17
  - The bus consists of two vertical columns of conductors.
  - At regular intervals along the columns, there are attachment points in the form of slots that extend out horizontally to support a printed circuit board.

3.4.1. Bus Structure

58

Computer Structure

NLU-FIT

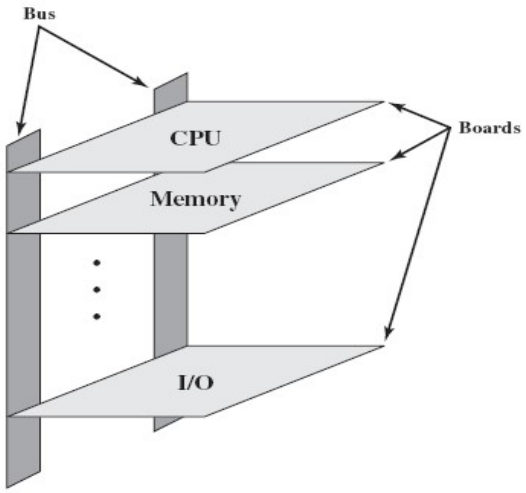


Figure 3.17. Typical Physical Realization of a Bus Architecture

3.4.1. Bus Structure

- However, modern systems tend to have all of the major components on the same board with more elements on the same chip as the processor.
- 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.

3.4.2. 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.
    - ✓This delay determines the time it takes for devices to coordinate the use of the bus.
    - ✓When control of the bus passes from one device to another frequently, these propagation delays can noticeably affect performance.

3.4.2. Multiple-Bus Hierarchies

61

Computer Structure

NLU-FIT

- 2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.
  - ✓This problem can be countered to some extent by increasing the data rate that the bus can carry and by using wider buses (e.g., increasing the data bus from 32 to 64 bits).
  - ✓However, because the data rates generated by attached devices (e.g., graphics and video controllers, network interfaces) are growing rapidly, this is a race that a single bus is ultimately destined to lose.

3.4.2. Multiple-Bus Hierarchies

62

Computer Structure

NLU-FIT

- Accordingly, most computer systems use multiple buses, generally laid out in a hierarchy-Figure 3.18a

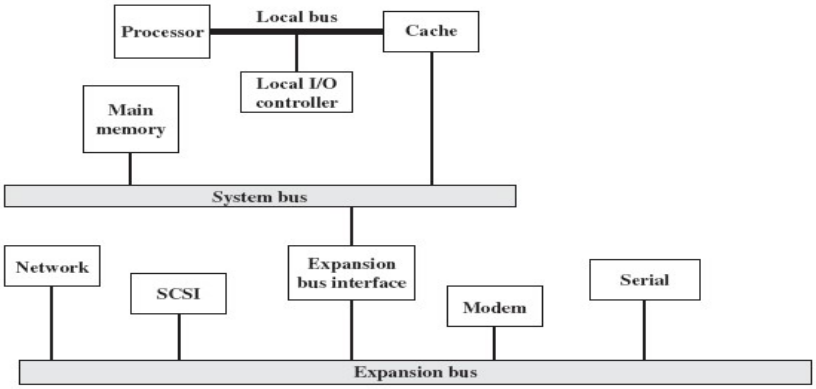


Figure 3.18a. Traditional bus architecture

3.4.2. Multiple-Bus Hierarchies

- 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 controller connects the cache not only to this local bus, but to a system bus to which are attached all of the main memory modules.
- The use of a cache structure insulates the processor from a requirement to access main memory frequently. Hence, main memory can be moved off of the local bus onto a system bus.
- I/O transfers to and from the main memory across the system bus do not interfere with the processor's activity.

3.4.2. Multiple-Bus Hierarchies

- 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 highspeed 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 3.18b

3.4.2. Multiple-Bus Hierarchies

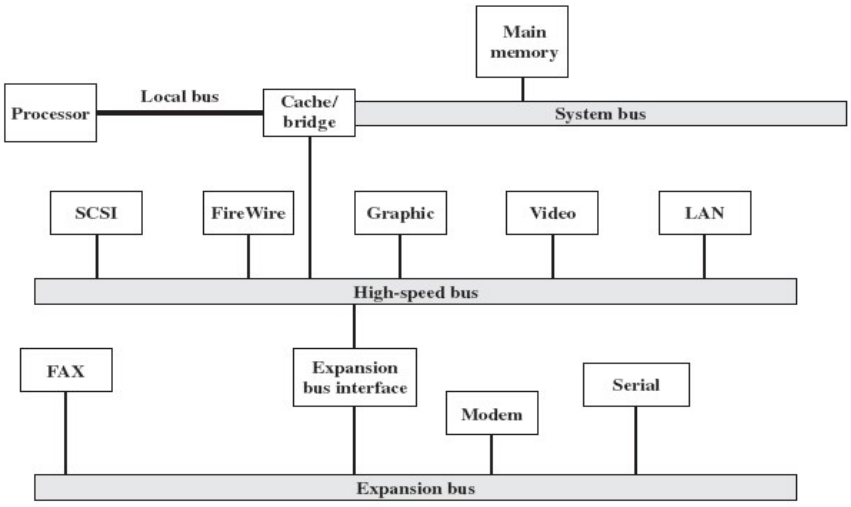


Figure 3.18b. High-performance architecture

3.4.2. Multiple-Bus Hierarchies

- 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.
- Lower-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. Thus, differences in processor and high-speed bus speeds and signal line definitions are tolerated.
  - Changes in processor architecture do not affect the high-speed bus, and vice versa.

3.4.3. Elements of Bus Design

- Although a variety of different bus implementations exist, there are a few basic parameters or design elements that serve to classify and differentiate buses as follow

Type	**Bus Width**
Dedicated	Address
Multiplexed	Data
Method of Arbitration	**Data Transfer Type**
Centralized	Read
Distributed	Write
Timing	Read-modify-write
Synchronous	Read-after-write
Asynchronous	Block

3.4.3. Elements of Bus Design

- **BUS TYPES** Bus lines can be separated into two generic types: dedicated and multiplexed.
  - Dedicated
    - ✓Separate data & address lines
  - Multiplexed
    - ✓Shared lines
    - ✓Address valid or data valid control line
    - ✓Advantage - fewer lines
    - ✓Disadvantages
      - ✓More complex control
      - ✓Ultimate performance

3.4.3. Elements of Bus Design

- **METHOD OF ARBITRATION** 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.*
    - ✓*The device may be* a separate module or part of the processor.
  - In a distributed scheme
    - ✓There is no central controller.
    - ✓Each module contains access control logic and the modules act together to share the bus.

3.4.3. Elements of Bus Design

- **TIMING** Timing refers to the way in which events are coordinated on the bus. Buses use either synchronous timing or asynchronous timing.
  - With synchronous timing, the occurrence of events on the bus is determined by a clock.
    - ✓The bus includes a clock line upon which a clock transmits a regular sequence of alternating 1s and 0s of equal duration.
    - ✓A single 1–0 transmission is referred to as a *clock cycle or bus cycle and defines a time slot.*
    - ✓*All other devices on* the bus can read the clock line, and all events start at the beginning of a clock cycle

3.4.3. Elements of Bus Design

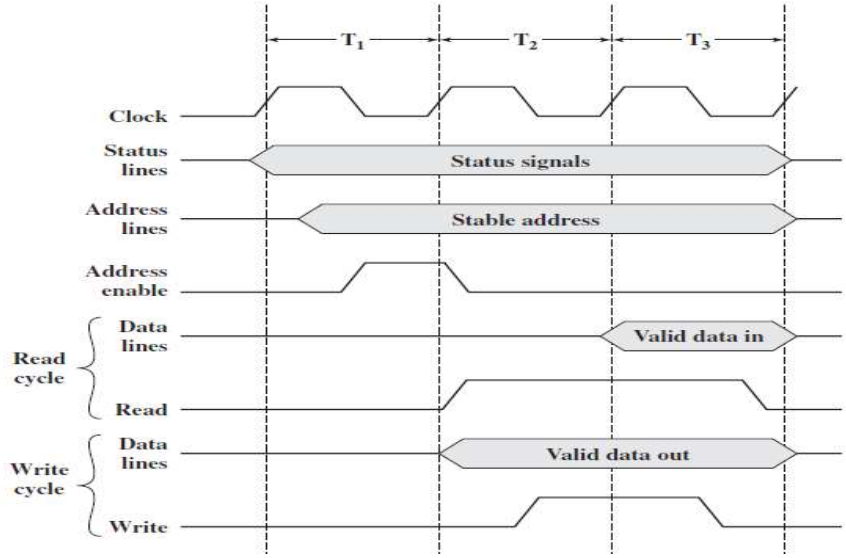


Figure 3.19. Timing of Synchronous Bus Operations

3.4.3. Elements of Bus Design

- With asynchronous timing, the occurrence of one event on a bus follows and depends on the occurrence of a previous event.

- Synchronous timing is simpler to implement and test.
  - However, it is less flexible than asynchronous timing. Because all devices on a synchronous bus are tied to a fixed clock rate, the system cannot take advantage of advances in device performance.
- With asynchronous timing, a mixture of slow and fast devices, using older and newer technology, can share a bus.

3.4.3. Elements of Bus Design

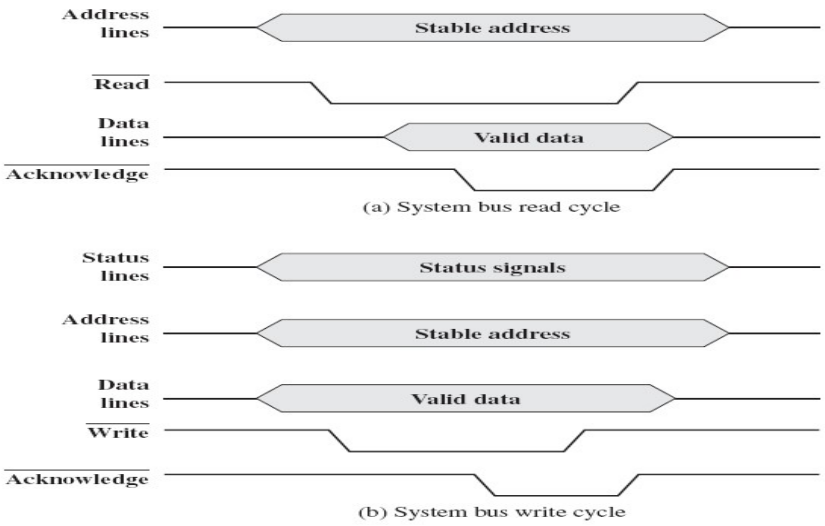


Figure 3.20. Timing of Asynchronous Bus Operations

3.4.3. Elements of Bus Design

- **BUS WIDTH** We have already addressed the concept of 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.

3.4.3. Elements of Bus Design

- **DATA TRANSFER TYPE** A bus supports various 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.
  - 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.

Reference: *Computer Organization and Architecture Designing for Performance (8th Edition), William Stallings, Prentice Hall, Upper Saddle River, NJ 07458.*