[16] "Function template - cppreference.com," en.cppreference.com. https://en.cppreference.com/w/cpp/language/function_template

[17] "Exception Specifications | Exception Handling in C++ | InformIT," www.informit.com. https://www.informit.com/articles/article.aspx?p=31537&seqNum=4

COMPUTER ORGANIZATION AND EMBEDDED SYSTEM (ACtE04)

4.1 CONTROL AND CENTRAL PROCESSING UNITS

Control and central processing units:

The control unit (CU) and central processing unit (CPU) are both essential components of a computer's processor. The control unit is responsible for controlling the flow of data and instructions within the processor, while the central processing unit performs arithmetic and logical operations on the data.

a) The control unit fetches instructions from memory, decodes them, and coordinates the execution of the instruction by sending signals to other parts of the processor. It also manages the flow of data between the CPU and other parts of the computer, such as input/output devices.

b) The central processing unit, on the other hand, performs arithmetic and logical operations on data using its arithmetic logic unit (ALU). It also has registers, small areas of very fast memory, used for holding data and intermediate results.

Control Memory:

a) Control Memory: In a microprocessor, the control memory is a type of memory that stores microcode, which is a set of low-level instructions used to implement the higher-level machine instructions that the processor executes. The microcode in the control memory is used by the control unit to sequence the execution of machine instructions and control the operations of other components in the processor.

b) Addressing Sequencing: The control unit is responsible for fetching instructions from memory and decoding them. It then generates the appropriate control signals to direct the sequencing of data and instructions in the CPU. The control unit uses the program counter (PC) to keep track of the address of the next instruction to be executed.

c) Computer Configuration: The configuration of a computer includes the types of hardware and software components that are used to build the system. This includes the CPU, memory, storage, input/output devices, and other components. The configuration affects the performance, capabilities, and compatibility of the computer.

d) Microinstruction Format: Microinstructions are low-level instructions used to control the operations of the processor's components. The microinstruction format typically includes

fields for the microinstruction operation, address sequencing, control signal generation, and other information needed to execute the microinstruction. The microinstruction format is specific to the microprocessor architecture and is designed by the manufacturer of the microprocessor.

Design of Control Unit:

The control unit (CU) in a processor can be classified into two major categories: hardwired control units and microprogrammed control units.

a) **Hardwired Control Unit**: In a hardwired control unit, the control logic is implemented using digital circuits, such as logic gates and flip-flops. The control signals are generated directly by the combinational logic circuits based on the instruction opcode and addressing modes. Hardwired control units are typically faster than microprogrammed control units, but they are less flexible and harder to modify.

b) **Microprogrammed Control Unit**: In a microprogrammed control unit, the control logic is implemented using microcode, which is a sequence of microinstructions stored in control memory. The microcode defines the control signals for each instruction opcode and addressing mode, and the control unit fetches and executes the microinstructions to generate the control signals. Microprogrammed control units are more flexible than hardwired control units, as the microcode can be easily modified to support new instructions or instruction sets. However, microprogrammed control units are generally slower than hardwired control units, as they require an additional layer of interpretation and execution.

CPU Structure and Function:

The CPU (Central Processing Unit) is the primary component of a computer system responsible for executing instructions and performing arithmetic and logic operations on data. The CPU can be divided into two main components: the control unit (CU) and the arithmetic logic unit (ALU).

Control Unit (CU): The control unit is responsible for fetching instructions from memory, decoding them, and directing the operation of the rest of the CPU. It uses the program counter (PC) to keep track of the address of the next instruction to be executed, and generates control signals that direct the flow of data and instructions between the different components of the CPU.

Arithmetic Logic Unit (ALU): The ALU performs arithmetic and logic operations on data. It consists of circuits that perform basic operations such as addition, subtraction, multiplication, division, logical AND, OR, NOT operations, and more complex operations.

In addition to the CU and ALU, the CPU may also have other components, such as registers, cache memory, and a memory management unit (MMU).

Registers: Registers are small, fast memory locations used to hold data and instructions that the CPU needs to access quickly. There are different types of registers, including general-purpose registers, which can be used for any purpose, and special-purpose registers, which have specific functions, such as the program counter, the instruction register, and the status register.

Cache Memory: Cache memory is a type of high-speed memory that stores frequently accessed data or instructions. It helps to improve the performance of the CPU by reducing the time needed to access data from main memory.

Memory Management Unit (MMU): The MMU is responsible for managing the memory hierarchy of the system. It translates virtual addresses to physical addresses, manages access to the different types of memory (such as cache, RAM, and ROM), and enforces memory protection and security policies.

Overall, the CPU is the "brain" of the computer, responsible for executing instructions and performing computations. Its structure and function are critical to the performance and capabilities of the computer system.

Arithmetic and logical Unit:

The Arithmetic and Logic Unit (ALU) is a component of the CPU (Central Processing Unit) responsible for performing arithmetic and logical operations on binary data. The ALU is typically composed of digital circuits that implement various functions, such as addition, subtraction, multiplication, division, AND, OR, XOR, and NOT operations.

Instruction format:

In computer architecture, the instruction format is a standardized structure used to represent machine instructions. The instruction format defines the various fields and bits used to encode the different components of an instruction, such as the opcode, operand address, and addressing mode.

The instruction format can vary depending on the processor architecture and can include different fields, such as:

a) Opcode: The opcode specifies the operation to be performed by the CPU.
b) Operand address: The operand address specifies the memory location of the data or register to be used in the operation.
c) Addressing mode: The addressing mode specifies how the operand address is calculated, such as immediate, direct, indirect, or indexed addressing.
d) Data size: The data size specifies the number of bits used to represent the data.
e) Condition code: The condition code specifies the condition under which the instruction should be executed, such as a particular flag being set or cleared.

Types of Instruction format:

a) **One-address format:** In this format, the instruction specifies a single memory operand and a single accumulator register. The accumulator register is typically used as the destination register for the instruction, and the result of the operation is stored back in the accumulator. One-address format was commonly used in early computer architectures. Here's an example of a one-address instruction: ADD 100
This instruction adds the contents of memory location 100 to the contents of the accumulator.

b) **Two-address format**: In this format, the instruction specifies two operands, both of which are registers. The result of the operation is stored in one of the registers, which can

be either the source or destination register. Two-address format is commonly used in CISC architectures. Here's an example of a two-address instruction: ADD R1, R2

This instruction adds the contents of register R2 to the contents of register R1 and stores the result back in register R1.

c) **Three-address format:** In this format, the instruction specifies three operands, which can be registers or memory locations. The result of the operation is stored in a specified register. Three-address format is commonly used in RISC architectures. Here's an example of a three-address instruction:

ADD R1, R2, R3

This instruction adds the contents of registers R2 and R3 and stores the result in register R1.

Addressing Mode:

In computer architecture, addressing mode refers to the way in which the processor interprets and uses memory addresses to access data or instructions from memory. Different addressing modes are used to provide flexibility and efficiency in accessing data and instructions in memory.

The following are the different types of addressing modes:

a) Immediate Addressing: The operand is part of the instruction itself. The value of the operand is stored directly in the instruction. For example, in the instruction ADD R1, #10, the operand #10 is added to the contents of register R1.

b) Direct Addressing: The operand is stored in a memory location specified by the instruction. For example, in the instruction ADD R1, [2000], the operand is stored in memory location 2000, and it is added to the contents of register R1.

c) Indirect Addressing: The operand is stored in a memory location whose address is stored in a register or memory location specified by the instruction. For example, in the instruction ADD R1, [R2], the address of the operand is stored in register R2, and the contents of that memory location are added to the contents of register R1.

d) Register Addressing: The operand is stored in a register specified by the instruction. For example, in the instruction ADD R1, R2, the contents of register R2 are added to the contents of register R1.

e) Relative Addressing: The operand is located at a memory address calculated by adding an offset value to the value of the program counter or another register. For example, in the instruction JMP [PC+10], the program counter (PC) is incremented by 10 to find the memory address of the next instruction.

f) Indexed Addressing: The operand is located at a memory address calculated by adding an index value to a base address. For example, in the instruction ADD R1, [R2+10], the value in register R2 is added to 10 to calculate the memory address of the operand.

g) Stack Addressing: The operand is located on the top of a stack. For example, in the instruction PUSH R1, the contents of register R1 are pushed onto the stack

Data Transfer and Manipulation:

Data transfer and manipulation instructions are a type of instruction set used in computer architecture to move data between memory and registers, and perform various operations on the data. These instructions are critical to the operation of the CPU and are used to perform arithmetic, logic, and control operations.

Data transfer instructions are a type of instruction set used in computer architecture to move data between memory and registers, or between different registers. These instructions are critical to the operation of the CPU and are used to move data for processing, storage, or communication.

Some common data transfer instructions are:

a) **Load instructions:** Load instructions transfer data from a memory location to a register. For example, the instruction "LD R1, [2000]" loads the contents of memory location 2000 into register R1.

b) **Store instructions:** Store instructions transfer data from a register to a memory location. For example, the instruction "ST R1, [2000]" stores the contents of register R1 into memory location 2000.

c) **Move instructions**: Move instructions transfer data from one register to another register. For example, the instruction "MOV R1, R2" moves the contents of register R2 to register R1.

d) **Exchange instructions**: Exchange instructions swap the contents of two registers. For example, the instruction "XCHG R1, R2" exchanges the contents of register R1 and R2.

e) **Push and Pop instructions**: Push and Pop instructions are used to push data onto the stack and pop data off the stack, respectively. For example, the instruction "PUSH R1" pushes the contents of register R1 onto the stack.

f) **Input and Output instructions:** Input and Output instructions transfer data between the CPU and input/output devices. For example, the instruction "IN R1, PORT" reads the data from the input port into register R1.

Data manipulation instructions are a type of instruction set used in computer architecture to perform operations on binary data in registers or memory locations. These instructions are used to manipulate data by performing arithmetic, logical, or bitwise operations.

Some common data manipulation instructions are:

- **Arithmetic instructions**: Arithmetic instructions are used to perform mathematical operations on data. Examples include ADD (addition), SUB (subtraction), MUL (multiplication), DIV (division), and INC (increment). For example, the instruction ADD R1, R2 adds the contents of register R2 to the contents of register R1.
- **Logical instructions:** Logical instructions are used to perform logical operations on data, such as AND, OR, and NOT. For example, the instruction AND R1, R2 performs a bitwise AND operation on the contents of registers R1 and R2.
- **Bit manipulation instructions:** Bit manipulation instructions are used to manipulate individual bits within a register or memory location. Examples include SET, CLR (clear),

and TST (test). For example, the instruction SET R1, 5 sets the fifth bit of the contents of register R1 to 1.

- **Shift and Rotate instructions:** Shift and rotate instructions are used to shift or rotate the bits of a register or memory location. Examples include LSL (logical shift left), ASR (arithmetic shift right), and ROR (rotate right). For example, the instruction LSL R1, 2 shifts the bits of the contents of register R1 two bits to the left.
- **Compare and Test instructions:** Compare and test instructions are used to compare the contents of two registers or memory locations, or to test a single bit within a register or memory location. Examples include CMP (compare), TEQ (test equal), and BIT (bit test). For example, the instruction CMP R1, R2 compares the contents of registers R1 and R2.

RISC Pipelining:

In RISC (Reduced Instruction Set Computing) architecture, pipelining is a technique used to improve the performance of the processor. In a pipelined RISC processor, the processor is divided into a series of stages, and each stage is responsible for executing a specific part of the instruction.

The basic stages in a RISC pipeline are:

a) **Instruction Fetch (IF):** The instruction is fetched from memory and loaded into an instruction register.
b) **Instruction Decode (ID):** The instruction is decoded to determine the operation to be performed and the operands.
c) **Execute (EX):** The operation is performed, and the result is stored in a register.
d) **Memory Access (MEM):** The data is accessed from memory, if needed.
e) **Write Back (WB):** The result is written back to the register file.

Once the pipeline is filled, each stage processes one instruction at a time, and the next instruction is fetched as soon as the previous instruction enters the next stage. This results in a faster processor as multiple instructions can be executed in parallel, without waiting for one instruction to complete before starting the next one.

Some of the key features of RISC machines include:

a) **Simple instructions:** RISC instructions are simple and have a uniform format. Each instruction performs only one operation, which makes the processor simpler and faster.
b) **Large number of registers:** RISC machines have a large number of registers, which reduces the number of memory accesses and improves performance.
c) **Load/store architecture**: RISC machines use a load/store architecture, which means that data is loaded from memory into a register, and then manipulated in the register. Results are then stored back to memory. This reduces memory access times and improves performance.
d) **Fixed instruction length:** RISC instructions are fixed in length, which makes them easier to decode and execute.
e) **Pipelining:** RISC machines often use pipelining, which allows multiple instructions to be executed in parallel, improving performance.

f) **Branch prediction:** RISC machines often use branch prediction to avoid pipeline stalls when executing conditional branches.
g) **High-level language support:** RISC machines often have built-in hardware support for high-level programming languages, such as compilers and interpreters. This allows for easier programming and better performance.

RISC machines are designed to be simple, fast, and efficient. The reduced instruction set and load/store architecture minimize memory access times and maximize the use of registers. The use of pipelining and branch prediction further improve performance. RISC architecture is widely used in modern microprocessors, particularly in embedded systems and mobile devices.

CISC:

CISC (Complex Instruction Set Computing) is a type of processor architecture that uses a large instruction set and complex instructions to perform multiple operations in a single instruction. Some of the key features of CISC in computer architecture are:

a) **Large instruction set:** CISC processors have a large number of instructions that can perform a variety of tasks, including arithmetic and logical operations, data movement, and control operations.
b) **Complex instructions:** CISC instructions can perform multiple operations in a single instruction. For example, an instruction might load data from memory, perform an arithmetic operation, and store the result in memory, all in a single instruction.
c) **Addressing** modes: CISC processors support a variety of addressing modes, which allows for more flexible data access. Addressing modes include immediate, direct, indirect, indexed, and stack addressing.
d) **Memory access:** CISC processors can access memory directly, without the need for complex addressing calculations. This allows for faster data access and better memory utilization.
e) **Microcode:** CISC processors use microcode, which is a set of low-level instructions that are used to execute higher-level instructions. Microcode allows for more complex operations and better instruction execution.
f) **Hardware support for high-level languages:** CISC processors often have built-in hardware support for high-level programming languages, such as compilers and interpreters. This allows for easier programming and better performance.

***Table:** Comparison between RISC and CISC architecture*

Feature	RISC	CISC
Instruction Set	Reduced, with few basic instructions	Complex, with many instructions
Instruction Length	Fixed length instructions	Variable length instructions
Clock Cycles per Instruction	1 or 2 cycles per instruction	More than 2 cycles per instruction

Feature	RISC	CISC
Register Set	Large number of general-purpose registers	Fewer general-purpose registers
Memory Access	Load/Store architecture, separate load/store instructions	Memory-to-memory architecture, instructions that can access memory directly
Pipeline	Simple, with few stages	Complex, with many stages
Performance	Faster due to simplified instruction set and pipeline	Slower due to complex instruction set and pipeline
Design Philosophy	Simple and efficient	Complex and versatile

Parallel Processing:

Parallel processing in computer architecture refers to the use of multiple processing units or cores to execute a program or task simultaneously. Parallel processing can significantly improve the performance of a system and is commonly used in high-performance computing, scientific simulations, data analysis, and other applications.

There are several types of parallel processing:

a) **Task parallelism:** In task parallelism, multiple processors work on different tasks simultaneously. For example, in a web server, one processor may handle incoming requests while another processor processes the data for the requests.

b) **Data parallelism:** In data parallelism, the same task is executed on different data sets simultaneously. For example, in a graphics processing unit (GPU), the same instruction is executed on multiple pixels in parallel.

c) **Pipeline parallelism:** In pipeline parallelism, a single task is divided into multiple stages, and each stage is executed by a different processor. For example, in a video decoder, each stage of the decoding process can be executed by a different processor in parallel.

Flynn's taxonomy

Flynn's taxonomy is a classification system for parallel processing systems that was proposed by Michael J. Flynn in 1966. It categorizes parallel processing systems based on the number of instruction streams and data streams that are processed simultaneously. There are four classes of parallel processing systems in Flynn's taxonomy:

a) **Single Instruction, Single Data (SISD):** This class of system has a single processor that executes a single instruction on a single piece of data at a time. This is the most common type of computer system.

b) **Single Instruction, Multiple Data (SIMD):** This class of system has multiple processing elements that execute the same instruction on multiple pieces of data in parallel. SIMD systems are commonly used in applications such as graphics processing and scientific simulations.

c) **Multiple Instruction, Single Data (MISD):** This class of system has multiple processors that execute different instructions on the same piece of data in parallel. MISD systems are not commonly used in practice, as they are difficult to implement and do not provide significant performance benefits.

d) **Multiple Instruction, Multiple Data (MIMD):** This class of system has multiple processors that execute different instructions on different pieces of data in parallel. MIMD systems are commonly used in high-performance computing and scientific simulations, where large amounts of data must be processed in parallel.

4.2 COMPUTER ARITHMETIC AND MEMORY SYSTEM

Arithmetic and Logical operation

Arithmetic and logical operations are two types of operations that can be performed by the arithmetic logic unit (ALU) in a central processing unit (CPU) in a computer architecture.

a) Arithmetic operations involve manipulating numerical values, such as adding, subtracting, multiplying, or dividing two numbers. The ALU can perform arithmetic operations on binary data, which is represented by a sequence of 1s and 0s. The ALU can also perform arithmetic operations on floating-point data, which is used to represent non-integer values.

b) Logical operations, on the other hand, involve manipulating Boolean values, which can be either true or false. The ALU can perform logical operations such as AND, OR, NOT, and XOR on binary data. These logical operations are used to compare and manipulate binary data, such as comparing two binary values to determine if they are equal or not.

Both arithmetic and logical operations are essential for performing calculations and making decisions in a computer system. The ALU can perform a wide variety of arithmetic and logical operations, which can be combined to perform complex calculations and decision-making tasks. The ALU can also perform bitwise operations, which allow the manipulation of individual bits within binary data.

The Memory Hierarchy:

The memory hierarchy is a way of organizing different types of memory in a computer system. The block diagram of the memory hierarchy typically includes several levels of memory, with each level having different characteristics and performance.

The main features of the memory hierarchy are:

Capacity: The capacity of each level of the memory hierarchy is an important factor, as it determines how much data can be stored at each level.

Access Time: The access time of each level of the memory hierarchy refers to the time it takes to access a data item stored in that level.

Access time, speed, and data storage capacity are key characteristics of different types of memory in the memory hierarchy. Here's a brief overview of how these characteristics differ at each level of the memory hierarchy:

a) **Registers:** Registers have the fastest access time and highest speed of any type of memory. They can be accessed within a single clock cycle and can hold only a small amount of data, typically a few bytes or words.

b) **Cache Memory:** Cache memory has a faster access time and higher speed than RAM. Cache memory is typically accessed in a few clock cycles and can hold a few megabytes of data.

c) **Random Access Memory (RAM):** RAM has a slower access time and lower speed than cache memory but is faster than secondary storage. RAM can typically be accessed in tens of nanoseconds and can hold several gigabytes of data.

d) **Read-Only Memory (ROM):** ROM has a very fast access time but a very limited capacity. It is typically used to store the firmware, such as the BIOS, which is used to boot up the computer system.

e) **Secondary Storage:** Secondary storage devices, such as hard disk drives and solid-state drives, have a slow access time and low speed compared to RAM. The access time for secondary storage devices is typically measured in milliseconds. However, secondary storage devices can store a very large amount of data, often measured in terabytes.

f) **Magnetic Tape:** Magnetic tape is typically considered to be at the bottom of the memory hierarchy because it has a very slow access time compared to other types of memory, such as RAM and hard disk drives. Magnetic tape is a sequential access storage device, which means that data can only be read or written sequentially.

The memory hierarchy is organized in such a way that the fastest and most expensive memory is used for storing the most frequently accessed data, while slower and less expensive memory is used for storing less frequently accessed data. By using this organization, the computer system can take advantage of the principle of locality and improve performance by accessing data and instructions in the fastest available memory.

At the top of the memory hierarchy is the smallest and fastest type of memory, such as the cache. Below the cache is the main memory, which is larger and slower than the cache. At the bottom of the memory hierarchy is the non-volatile storage, such as a hard disk drive, which is the largest and slowest type of memory but retains its contents even when the power is turned off.

Internal and External Memory:

Internal memory, also known as primary memory or main memory, is the memory that is directly accessible to the central processing unit (CPU) and is used for storing data and instructions that are currently being processed

Internal memory, also known as primary memory or main memory, is a key component of a computer system. Some of the features of internal memory include:

a) **Speed**: Internal memory is designed to be accessed quickly by the central processing unit (CPU). It is typically made up of volatile memory, such as random-access memory (RAM), which can be read and written to quickly. The speed of internal memory is measured in nanoseconds, and it is typically much faster than external memory, such as hard disk drives.

b) **Capacity**: The capacity of internal memory can impact the performance of a computer system. Having more internal memory can improve the speed at which programs can be executed, as it can reduce the need to transfer data between the CPU and external memory.

c) **Cost**: Internal memory can be more expensive than external memory, such as hard disk drives. This is because internal memory is designed to be accessed quickly and is made from high-quality components.

d) **Volatility**: Internal memory is typically volatile, which means that data stored in internal memory is lost when the power is turned off. This is in contrast to external memory, such as hard disk drives, which are non-volatile and can retain data even when the power is turned off.

External Memory:

External memory, on the other hand, is a type of secondary memory that is used for long-term storage of data and applications. External memory is typically made up of non-volatile memory, such as hard disk drives, solid-state drives, and removable storage media, such as USB drives or SD cards. External memory is not directly accessible to the CPU and requires data to be transferred to internal memory before it can be processed. External memory is used for storing files, media, and other data that does not need to be accessed frequently or quickly.

Some of the key features of external memory include:

a) **Capacity**: External memory is designed to store large amounts of data, applications, and other digital files. The capacity of external memory can range from gigabytes to terabytes, depending on the type of storage device.

b) **Non-Volatility**: External memory is typically non-volatile, which means that data stored in external memory is not lost when the power is turned off. This is in contrast to internal memory, which is typically volatile.

c) **Speed**: External memory is generally slower than internal memory in terms of access time and data transfer rates. However, the speed of external memory has improved significantly in recent years, with solid-state drives (SSDs) providing faster access times than traditional hard disk drives.

d) **Cost**: External memory can be less expensive than internal memory in terms of cost per gigabyte. However, the cost of external memory can vary depending on the type of storage device, capacity, and other factors.

e) **Portability**: External memory is often designed to be portable, which means that it can be easily moved from one device to another. Examples of portable external memory devices include USB drives, SD cards, and external hard disk drives.

f) **Durability**: External memory is often designed to be more durable than internal memory. For example, some external storage devices are designed to be resistant to shock, vibration, and temperature extremes.

Cache Memory Principles:

Cache memory is a small, high-speed memory used to temporarily store frequently accessed data and instructions.

The principle of cache memory is based on the idea of exploiting the principle of locality.

The principle of locality states that programs tend to access a small portion of their memory at any given time.

By storing frequently accessed data and instructions in cache memory, the computer system can reduce the average time it takes to access data and instructions, leading to improved system performance. The basic principles of cache memory are as follows:

a) **Cache Hit:** When the CPU requests data or instructions that are already stored in cache memory, this is known as a cache hit. The data or instructions can be quickly accessed from the cache, which improves system performance.

b) **Cache Miss:** When the CPU requests data or instructions that are not stored in cache memory, this is known as a cache miss. The data or instructions must be retrieved from the main memory, which is slower than accessing data from cache memory.

c) **Cache Replacement:** When cache memory is full and a new piece of data or instruction must be stored, the cache controller must decide which item to remove from cache memory to make room for the new item. This process is known as cache replacement.

d) **Cache Coherency:** Cache coherency refers to the need to ensure that data stored in cache memory is consistent with data stored in main memory. When data is updated in main memory, the cache must be updated as well to ensure that the CPU is accessing the most up-to-date data.

Elements of Cache Design:

The design of a cache memory involves several elements, which can vary depending on the specific architecture and intended use case. Here's an overview of some of the common elements of cache design:

a) **Cache Size:** The cache size refers to the amount of data that can be stored in the cache. Larger caches can improve performance by reducing the number of cache misses, but they can also be more expensive.

b) **Mapping Function:** The mapping function determines how data is mapped to specific locations in the cache. Common mapping functions include direct mapping, set-associative mapping, and fully associative mapping.

c) **Replacement Algorithm:** When the cache is full and new data must be stored, the replacement algorithm determines which data is removed from the cache to make room for the new data. Common replacement algorithms include least recently used (LRU), first in, first out (FIFO), and random replacement.

d) **Write Policy:** The write policy determines how updates to data are handled in the cache. Write-through policies update both the cache and the main memory simultaneously, while write-back policies update only the cache and defer updating the main memory until later.

e) **Number of Caches:** Some systems may use multiple levels of cache, each with a different size and speed. This can help to improve performance by reducing the average time it takes to access data.

f) **Memory Write Ability and Storage Permanence:** Cache memory is typically volatile, which means that data is lost when the power is turned off. Some types of memory, such as flash memory or non-volatile RAM, may provide different write abilities and storage permanence.

g) **Composing Memory:** The design of a cache memory may also depend on the composition of the larger memory system. For example, the cache memory may be designed to work in conjunction with other types of memory, such as RAM or virtual memory.

4.3 INPUT-OUTPUT ORGANIZATION AND MULTIPROCESSOR

Peripheral devices

Peripheral devices, also known as peripherals, are hardware components that are connected to a computer system to provide additional functionality or capabilities. These devices can be either internal or external to the computer system and can include a wide range of devices, such as input devices, output devices, and storage devices. Some examples of peripheral devices include:

a) **Keyboard**: An input device that allows users to enter text and commands into a computer system.

b) **Mouse**: An input device that allows users to move a cursor on a computer screen and select items.

c) **Printer**: An output device that allows users to produce hard copies of documents and images.

d) **Monitor**: An output device that displays visual information, such as text and images.

I/O Module:

An I/O module, short for Input/Output module, serves as a linkage between a computer system and an I/O or peripheral device, like a scanner, printer, or webcam. Its purpose is to enable the computer system to fulfill its intended purpose of communicating with the external world in the required manner.

The I/O module performs several significant functions which are given by:

a) **Processor Communication:** The communication between the processor and I/O module comprises transferring data, decoding commands, reporting the current status, and enabling the I/O module to identify its unique address.

b) **Device communication:** The I/O module should be capable of conducting standard device communication, including reporting status.

c) **Control and timing:** The I/O module must manage the data flow between the computer's internal resources and any connected external devices.

d) **Data Buffering:** A critical function of the I/O module is managing the speed discrepancy between data transfer rates of the processor and memory and peripheral devices.

e) **Error Detection:** Another essential function of the I/O module is to detect errors, whether mechanical (e.g., a paper jam in a printer) or data-based, and report them to the processor.

Input-Output Interface:

The Input-Output (I/O) interface facilitates the transfer of information between internal storage devices, such as memory, and external peripheral devices, which provide input and output to the computer. Input devices, such as keyboards and mice, provide input to the computer, while

output devices, such as monitors and printers, provide output. Some peripheral devices, like external hard drives, offer both input and output capabilities.

The Input/output (I/O) interfaces have several major functions, including:

a) **Data transfer:** The I/O interface is responsible for transferring data between the CPU and external devices, including input and output devices, memory, and other peripherals.

b) **Addressing:** The I/O interface enables the CPU to access specific locations in memory or external devices by providing an addressing scheme.

c) **Timing and control:** The I/O interface manages the timing and control of data transfer between the CPU and external devices.

d) **Interrupt handling:** The I/O interface allows for interrupts to occur, which are signals sent to the CPU to pause the execution of the current program and handle an event or request from an external device.

e) **Error handling:** The I/O interface detects and handles errors that may occur during data transfer between the CPU and external devices.

Modes of transfer:

In computer architecture, there are three modes of data transfer between the CPU and peripherals: programmed I/O, interrupt-driven I/O, and direct memory access (DMA).

a) **Programmed I/O:** In this mode, the CPU performs the entire data transfer operation by continuously checking the status of the device. It is simple but inefficient as it requires a lot of CPU overhead.

b) **Interrupt-driven I/O:** In this mode, the device sends an interrupt signal to the CPU when it is ready, and the CPU performs the data transfer. It is more efficient than programmed I/O but still has high CPU overhead.

c) **Direct Memory Access (DMA):** In DMA mode, a separate DMA controller handles the data transfer operation between the CPU and peripherals. The DMA controller takes over the data transfer operation independently of the CPU, making it the most efficient mode of data transfer. DMA is used for high-speed data transfer operations such as video and audio streaming, disk I/O, and network I/O.

Characteristics of multiprocessors:

A multiprocessor is a single computer with multiple processors that communicate and cooperate at various levels to solve a problem, either by sending messages or sharing a common memory.

The major characteristics of multiprocessors include:

a) **Parallel computing:** Multiple identical processors work together to execute a common task simultaneously.

b) **Distributed computing:** A network of heterogeneous processors work together, with each processor allocated to a single task.

c) **Supercomputing:** The fastest machines are used to resolve big and computationally complex problems.

d) **Pipelining:** A specific task is divided into several subtasks, and functional units perform each subtask simultaneously.

e) **Vector computing:** Vector processors divide operations into many steps and apply them to a stream of operands.

f) **Systolic:** Small steps are performed in a lockstep manner, with units not arranged in a linear order. This is frequently used in special-purpose hardware such as image or signal processors.

Interconnection Structure of Multiprocessor:

Inter connection structures are essential in a multiprocessor system to enable processors to share main memory modules and I/O devices. Some of the commonly used interconnection structures are:

a) **Time-shared/common bus:** Multiple processors are connected to a single bus, and each processor takes turns accessing the bus.

b) **Crossbar switch:** A network of switches is used to connect multiple processors and memory modules, enabling direct communication between any two processors.

c) **Multiport memory:** Multiple memory modules are used, and each processor has direct access to a specific memory module.

d) **Multistage switching network:** A combination of crossbar switches and other network elements, such as routers, is used to connect multiple processors and memory modules.

e) **Hypercube system:** A network of nodes arranged in a hypercube shape is used to connect multiple processors; with each node connected to other nodes in a specific pattern.

Inter-processor communication and synchronization

Inter-processor communication and synchronization are important aspects of a multiprocessor system. Here are some major points about them:

a) Inter-processor communication refers to the exchange of data between processors in a multiprocessor system.

b) Synchronization refers to the coordination of activities between processors in a multiprocessor system.

c) Inter-processor communication and synchronization are critical for ensuring the proper execution of shared resources and for coordinating the actions of multiple processors.

d) Various techniques are used for inter-processor communication and synchronization, including shared memory, message passing, and semaphores.

e) Shared memory is a common technique that allows multiple processors to access the same memory location and share data between them.

f) Message passing involves sending messages between processors to communicate information or request a specific action.

g) Semaphores are used to control access to shared resources and ensure that multiple processors do not access them simultaneously.

4.4 HARDWARE-SOFTWARE DESIGN ISSUES ON EMBEDDED SYSTEM

Here is an overview of embedded systems in summary:

a) Embedded systems are computer systems designed for a specific purpose or application, often with dedicated hardware and software designed to perform a single or limited set of functions.

b) Embedded systems are used in a variety of applications, including consumer electronics, automotive systems, medical devices, and industrial automation.

c) Embedded systems are often designed to operate in real-time and to interact with the physical environment through sensors and actuators.

d) The design of embedded systems requires a deep understanding of the hardware and software components, as well as the specific requirements of the application.

e) The development of embedded systems involves a range of activities, including system specification, hardware and software design, testing, and deployment.

f) The use of embedded systems is increasing rapidly, driven by advances in technology, the growth of the Internet of Things, and the demand for more intelligent and autonomous systems.

Classification of Embedded Systems:

Here is a classification of embedded systems with examples:

Simple embedded systems: These are used in applications with limited functionality and low processing power, such as:

- Digital watches
- Electronic toys
- Home appliances like microwave ovens, washing machines, and air conditioners
- Remote controls

Real-time embedded systems: These are designed to respond to external events within a specific time frame and are used in applications such as:

- Automotive systems like engine control units and anti-lock braking systems
- Avionics systems like flight control and navigation systems
- Medical devices like pacemakers and insulin pumps

Networked embedded systems: These are designed to interact with other systems and can be connected to local or wide area networks. They are used in applications such as:

- Smart homes and buildings
- Industrial automation and control systems
- Transportation systems like traffic control and railway signaling systems

Standalone embedded systems: These are designed to operate independently and are used in applications such as:

- Electronic locks and security systems
- Digital cameras and camcorders
- Wearable devices like fitness trackers and smartwatches

Mobile embedded systems: These are designed to be portable and can be used in applications such as:

- Smartphones and tablets
- Portable media players and game consoles

Custom single-purpose processor design

A processor is a digital circuit that performs computational tasks, and it must have at least two main components: a controller and a data path. The different processor technologies are as follows:

General Purpose Processor (GPP): It is a programmable circuit that can perform a variety of tasks, and it typically includes a program memory and a data path with a large register array and one or more general purpose Arithmetic Logic Units (ALUs).

Single Purpose Processor (SPP): A Single-Purpose Processor (SPP) is a digital circuit designed to perform one specific program. A digital camera is an example of an SPP, which only has data memory and no program memory.

Application Specific Processor (ASP) is a programmable circuit optimized for specific applications. It includes program memory, data memory, a custom-designed ALU, and an optimized data path.

Custom single-purpose processor design is a process of creating a processor that is specifically designed to perform a particular task or set of tasks. It is a type of application-specific integrated circuit (ASIC) that is tailored to the specific needs of the system it is intended to be used in.

The design of a custom single-purpose processor typically involves the following steps:

a) **System specification:** This involves identifying the requirements of the system and the specific tasks that the processor needs to perform.

b) **Architecture design:** This involves selecting the appropriate processor architecture, instruction set, and pipeline structure that will enable the processor to perform the required tasks efficiently.

c) **Hardware design:** This involves designing the various components of the processor, such as the arithmetic logic unit (ALU), register file, memory interface, and input/output interface.

d) **Verification and testing:** This involves testing the processor design through simulation, emulation, and prototyping to ensure that it meets the required specifications.

e) **Fabrication and production:** Once the processor design has been verified and tested, it can be manufactured using semiconductor fabrication technology.

Optimizing Custom Single-Purpose Processors:

Optimization involves the process of improving the values of design metrics to their best possible levels. Optimization opportunities in the context of custom single-purpose processors can include:

a) **Original program:** Optimizing the original program by reducing its complexity, improving its algorithm, or minimizing the number of instructions required to execute it.

b) **Finite state machine with data path (FSMD):** Optimizing the FSMD by minimizing the number of states required and reducing the amount of computation that needs to be performed at each state.

c) **Datapath:** Optimizing the data path by minimizing the number of resources required, such as registers, arithmetic logic units (ALUs), and multiplexers. This can involve optimizing the size and configuration of the register file, minimizing the number of ALUs required, and optimizing the routing of data between resources.

d) **Finite state machine (FSM):** Optimizing the FSM by reducing the number of states required, reducing the amount of computation that needs to be performed at each state, and optimizing the routing of data between states.

Basic Architecture, Operation, and Programmer's View:

The basic architecture of a custom single-purpose processor typically includes the following components:

a) **Instruction register (IR):** This register stores the current instruction being executed by the processor.

b) **Control unit (CU):** The control unit fetches instructions from memory, decodes them, and generates control signals to direct the operation of the processor.

c) **Arithmetic and Logical Unit (ALU):** The ALU performs arithmetic and logical operations on data stored in the register file.

d) **Register file:** This is a set of registers used to store data and intermediate results during the execution of instructions.

e) **Memory interface:** The memory interface is responsible for accessing main memory to fetch instructions and data.

f) **Input/output (I/O) interface:** The I/O interface allows the processor to communicate with external devices such as sensors, actuators, and peripherals.

From a programmer's view, a custom single-purpose processor is similar to any other microprocessor. The programmer writes code in a high-level programming language such as C or Assembly, and the code is then compiled and executed on the processor.

Development Environment:

The development environment for a custom single-purpose processor typically includes the following components:

a) ***Hardware description language (HDL) compiler:*** This is used to translate the hardware design into a format that can be implemented in silicon.

b) ***Simulator:*** This is used to test and verify the design before implementation.

c) ***Debugger:*** This is used to debug the design and find and fix errors.

d) ***Programmer's tools:*** These are tools such as compilers, linkers, and libraries that are used to write and compile code for the processor.

Application-Specific Instruction-Set Processors:

An Application-Specific Instruction-Set Processor (ASIP) is a custom single-purpose processor that includes a tailored instruction set to perform a specific set of tasks. The instruction set is optimized for the target application, providing improved performance and efficiency compared to a general-purpose processor.

ASIPs are commonly used in embedded systems, where performance and power consumption are critical. Examples of applications that use ASIPs include digital signal processing, image processing, and cryptography.

4.5 REAL-TIME OPERATING AND CONTROL SYSTEM

A real-time operating and control system is a type of operating system that is designed to handle time-critical tasks in real-time. Real-time systems are used in a variety of applications, such as aircraft control systems, industrial control systems, and medical equipment, where precise and timely responses are essential.

The key characteristics of a real-time operating and control system include:

a) **Predictable response time:** A real-time system must provide a guaranteed response time for time-critical tasks, so that the system can meet the timing requirements of the application.

b) **Consistency:** The system must consistently meet its real-time deadlines, even in the presence of system failures or other unforeseen events.

c) **Determinism:** The system must operate in a deterministic manner, so that its behavior can be predicted with a high degree of accuracy.

d) **Robustness:** The system must be robust and able to handle failures and other unexpected events without affecting its ability to meet real-time requirements.

A real-time operating and control system typically include a real-time operating system (RTOS), which provides the underlying system services, and a control system, which implements the control logic for the application. The control system may be implemented using a programmable logic controller (PLC), a microcontroller, or a custom single-purpose processor, depending on the requirements of the application.

Operating System Basics:

An operating system (OS) is a software system that provides the basic functionality required to manage a computer's hardware and software resources, and to provide a platform for running applications. The operating system acts as the interface between the user and the computer's hardware, providing services such as memory management, process management, file system management, and input/output (I/O) management.

Tasks, Processes, and Threads:

In the context of an operating system, a task is a general term used to refer to a unit of work that a computer performs. A process is a more specific term that refers to a program in execution, with its own memory space and system resources. A process can be thought of as a container that holds the resources required to run a program.

A thread is a smaller unit of work that can be executed within a process. A process can contain multiple threads, which can run concurrently and share the resources of the process. The use of threads can improve the performance and responsiveness of an application by allowing multiple tasks to be executed in parallel.

Multiprocessing and Multitasking:

Multiprocessing is a technique used by an operating system to run multiple processes simultaneously, allowing multiple programs to run in parallel on a computer. This improves the overall performance of the system by allowing multiple tasks to be executed at the same time.

Multitasking is the ability of an operating system to run multiple tasks simultaneously, either by running multiple processes or by running multiple threads within a process. This allows the user to perform multiple tasks at the same time, such as running multiple applications or performing multiple operations within a single application.

Task Scheduling:

Task scheduling is the process by which an operating system decides which task to run next. The operating system uses a scheduling algorithm to determine the order in which tasks are executed, based on factors such as the priority of the task, the amount of CPU time required, and the amount of available memory.

Task Synchronization:

Task synchronization is the process of coordinating the execution of multiple tasks, so that they work together in a predictable manner. This is particularly important in multithreaded applications, where multiple threads may access shared data structures and need to coordinate their access to avoid conflicts.

The operating system provides synchronization mechanisms, such as semaphores and locks, to help ensure that multiple threads can work together in a coordinated manner.

In conclusion, an operating system provides the basic functionality required to manage a computer's hardware and software resources, and to provide a platform for running applications. The operating system manages tasks, processes, and threads, and provides mechanisms for task scheduling and task synchronization, to ensure that multiple tasks can work together in a coordinated manner.

Device Drivers:

A device driver is a software component that allows an operating system to interact with a specific hardware device. The device driver provides a standard interface between the operating system and the hardware device, allowing the operating system to control the device and access its data.

Device drivers are typically specific to a particular type of hardware device, such as a printer, a network adapter, or a storage device. They are responsible for performing tasks such as initializing the device, sending and receiving data, and controlling the device's behavior.

Open-loop and Closed-Loop Control System Overview:

Open-loop control systems are control systems in which the output of the system does not affect the control signal. In an open-loop control system, the control signal is predetermined and does not change based on the output of the system.

Closed-loop control systems, on the other hand, are control systems in which the output of the system affects the control signal. In a closed-loop control system, the control signal is continuously updated based on the output of the system, allowing the system to maintain a desired state.

Control:

Control is the process of regulating the behavior of a system to achieve a desired outcome. In the context of control systems, control is achieved by adjusting the input signal to the system based on its output.

4.6 HARDWARE DESCRIPTS LANGUAGE AND IC TECHNOLOGY

Hardware Description Language (HDL):

A Hardware Description Language (HDL) is a specialized programming language used to describe digital circuits and systems, including microprocessors, memory, and other digital devices. HDLs are used to design, verify, and implement digital circuits and systems.

The most common HDLs are VHDL (VHSIC Hardware Description Language) and Verilog. These languages allow designers to describe the behavior of a digital system using a high-level, abstract representation, making it easier to design, verify, and test digital systems.

IC Technology:

Integrated Circuit (IC) technology is the process of creating integrated circuits, also known as microchips, by combining multiple transistors, diodes, and other components on a single piece of semiconductor material. IC technology has revolutionized the electronics industry, allowing the creation of smaller, faster, and more efficient electronic devices.

ICs are used in a wide range of electronic devices, including computers, smartphones, and other consumer electronics. They are also used in medical devices, automotive systems, and industrial control systems, among other applications.

VHDL Overview:

VHDL (VHSIC Hardware Description Language) is a hardware description language used to describe and simulate digital systems, including microprocessors, memory, and other digital devices. VHDL is a powerful tool for digital system design, allowing designers to describe the behavior of a digital system using a high-level, abstract representation.

VHDL provides a set of constructs and statements that allow designers to describe the behavior of a digital system, including the inputs, outputs, and internal signals of the system. VHDL

also provides a simulation environment, allowing designers to test and verify the behavior of their design before it is implemented in hardware.

Overflow in VHDL:

Overflow is a condition that occurs when the result of a mathematical operation exceeds the range of the data type used to represent the result. In VHDL, overflow can occur in arithmetic operations, such as addition and multiplication, and can cause incorrect results.

To handle overflow in VHDL, designers can use overflow flags or overflow exceptions, which are triggered when an overflow occurs. The designer can then write code to handle the overflow condition, such as by reducing the precision of the result or by saturating the result to the maximum or minimum value of the data type.

Data Representation in VHDL:

VHDL provides several data types for representing data, including integer, real, and Boolean types. The choice of data type depends on the requirements of the application, including the range of values, the precision, and the size of the data.

In VHDL, data can be represented using fixed-point or floating-point representations, depending on the requirements of the application. Fixed-point representations are used for integer and fractional values, while floating-point representations are used for real values.

Design of Combinational Logic using VHDL:

Combinational logic is a type of digital logic that implements a function that depends only on the current inputs. In VHDL, combinational logic can be designed using concurrent statements, which describe the behavior of the system as a function of the inputs.

For example, to design a 2-to-1 multiplexer in VHDL, the following concurrent statements can be used:

VHDL Code:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux2to1 is
  port (
    sel: in std_logic;
    a: in std_logic;
    b: in std_logic;
    c: out std_logic
  );
end entity mux2to1;

architecture behavioral of mux2to1 is
begin
  c <= a when sel = '0' else b;
end architecture behavioral;
```


This code defines an entity named "mux2to1" that implements a 2-to-1 multiplexer, with an input "sel" that selects between inputs "a" and "b", and an output "c". The concurrent statement in the architecture section describes the behavior of the multiplexer, which outputs "a" when "sel" is 0, and outputs "b" when "sel" is 1.

Design of Sequential Logic using VHDL:

Sequential logic is a type of digital logic that implements a function that depends on both the current inputs and the previous state of the system. In VHDL, sequential logic can be designed using process statements, which describe the behavior of the system as a function of the inputs and the previous state.

For example, to design a D flip-flop in VHDL, the following process statement can be used:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity dff is
  port (
    clk: in std_logic;
    d: in std_logic;
    q: out std_logic
  );
end entity dff;
architecture behavioral of dff is
begin
  process (clk)
  begin
    if rising_edge(clk) then
      q <= d;
    end if;
  end process;
end architecture behavioral;
```

This code defines an entity named "dff" that implements a D flip-flop, with an input "clk" that is used to control the update of the flip-flop, an input "d" that provides the data to be stored, and an output "q" that provides the stored data. The process statement describes the behavior of the flip-flop, which updates the output "q" on the rising edge of the clock input "clk".