UNIT IV

Microprocessor

Computer's Central Processing Unit (CPU) built on a **single Integrated Circuit (IC)** is called a **microprocessor**.

A digital computer with one microprocessor which acts as a CPU is called microcomputer.

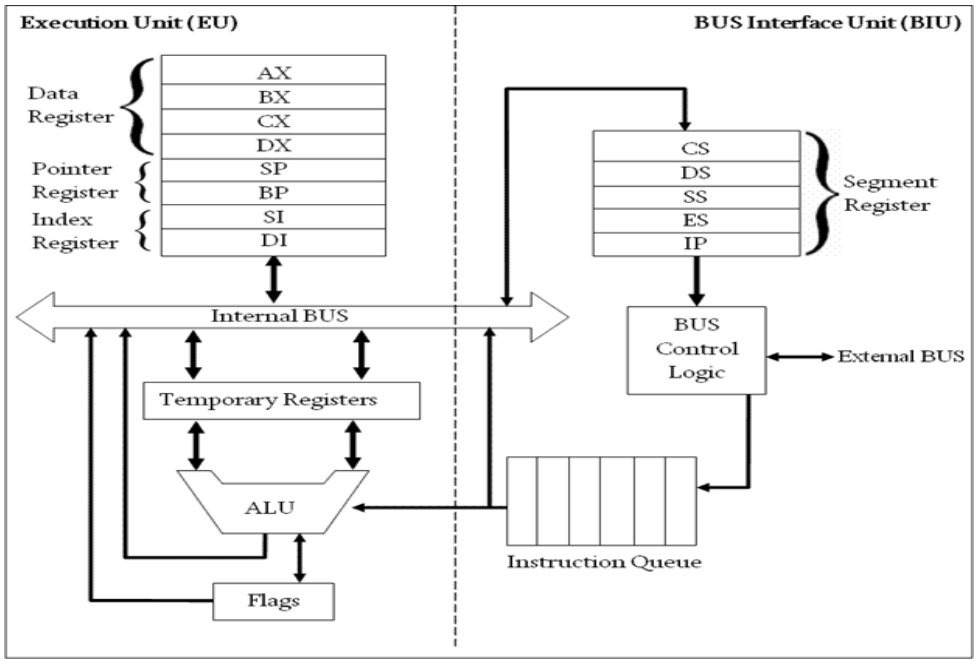
It is a programmable, multipurpose, clock -driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as output.

The microprocessor contains millions of tiny components like transistors, registers, and diodes that work together.

Types of Microprocessors

- Vector Processors
- Array Processors or SIMD Processors
- Scalar and Superscalar Processors
- RISC and CISC Processors
- Digital Signal Processors (DSP)

Block diagram of 8086-registers



Segment Registers

A segment register contains the addresses of instructions and data in memory which are used by the processor to access memory locations.

It points to the starting address of a memory segment currently being used. There are 4 segment registers in 8086 as given below:

Code Segment Register (CS): Code segment of the memory holds instruction codes of a program.

Data Segment Register (DS): The data, variables and constants given in the program are held in the data segment of the memory.

Stack Segment Register (SS): Stack segment holds addresses and data of subroutines. It also holds the contents of registers and memory locations given in PUSH instruction.

Extra Segment Register (ES): Extra segment holds the destination addresses of some data of certain string instructions.

Address

An address typically refers to a location in memory or storage where data or instructions can be stored or retrieved.

Effective address

The effective address is the location of an operand of the instruction, since the operand is the data to be accessed. Immediate instructions use their operand to hold the data needed to complete the instruction.

Flag register

The flag register is used to store the results of arithmetic and logic operations. The flags in the flag register provide information about the outcome of these operations, such as whether a result is negative or zero, or whether there was an overflow or carry.

Applications of microprocessor.

1. Microprocessors used in handheld devices:

Today, handheld devices - this includes smartphones, tablets, portable media players - have rather powerful microprocessors, so that they can compete with a desktop computer. And processors these days are coming up with more cores. Initially, processors were single core, followed by two cores, Quad-core, Hexa-core, Octa-core and now even Deca cores. Most processors nowadays are 64bit. With Graphics Processing Unit (GPU) being included inside mobile processors, these devices can now offer high quality graphics, Virtual Reality capability, 3D capability and 4k recording, plus the improved processor technology means greater power efficiency.

2. Microprocessors used in general-purpose computing (regular desktops and laptops):

In this segment, Intel and AMD are the market leaders. Intel chips are said to be the best for gaming and other predominantly single-threaded tasks. Particularly deserving a mention is the Core i5-9400F, which is both affordable and powerful. It comes with six cores, which is more than enough for most programs and a decent frequency of 4.1GHz when in turbo model. It is performs better than the Core i5-7600K from a couple of generations ago and is cheaper. Meanwhile, Core i3-8100 is the best budget offering from Intel. It’s a four-core chip with frequency of 3.6GHz that doesn’t offer additional boost speeds, but is fast enough for most PC users.

3. Microprocessors used in high-performance computing (supercomputers):

These computers, or supercomputers as they are called, are computers with a higher level of performance compared to a general-purpose computer. Supercomputers play an important role in computational science, and are used for many computationally intensive tasks in areas such as quantum mechanics, weather forecasting, climate research, oil and gas exploration, molecular modeling (computing the structures and properties of chemical compounds, macromolecules, polymers and crystals), and physical simulations (simulations of the early moments of the universe, airplane and spacecraft aerodynamics, the detonation of nuclear weapons, and nuclear fusion). They have also played a major

role in the field of cryptanalysis (analysing information systems in order to study the hidden aspects of the systems).

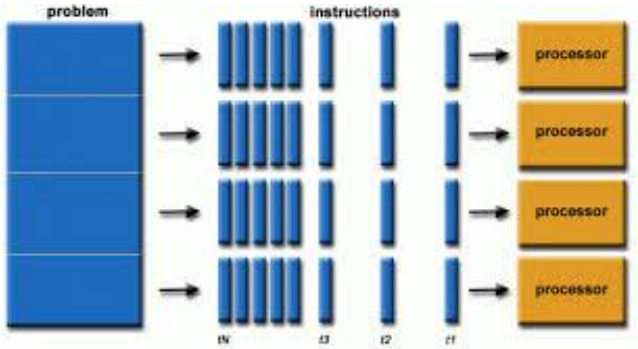
4. Microprocessors used in appliances (smart home devices, gaming, cars):

MediaTek, Intel and AMD are major players in this category. MediaTek makes chips for wireless communications, high-definition television, navigation systems, consumer multimedia products, digital subscriber line services and optical disc drives. In 2018, the company announced its plans to enhance its home entertainment platform by building solutions powering artificial intelligence (AI) voice and vision capabilities in several smart home devices. The announcement was accompanied by the launch of the MT8695, a 4K dongle SoC, the MT8516 system on module (SoM) and the smart display solution. MediaTek is already a leading name when it comes to offering voice assistant device (VAD) SoC solutions for the smart home ecosystem; MediaTek chips support multiple mainstream AI voice services including Amazon Alexa; Google Assistant, Alibaba and Baidu.

Parallel processing

Parallel processing in computer architecture is the simultaneous execution of multiple computing tasks or instructions to solve a single problem. By breaking a large workload into smaller sub-tasks, systems drastically increase computational speed, efficiency, and throughput compared to traditional sequential processing.

A parallel processing system can carry out simultaneous data-processing to achieve faster execution time. For instance, while an instruction is being processed in the ALU component of the CPU, the next instruction can be read from memory.

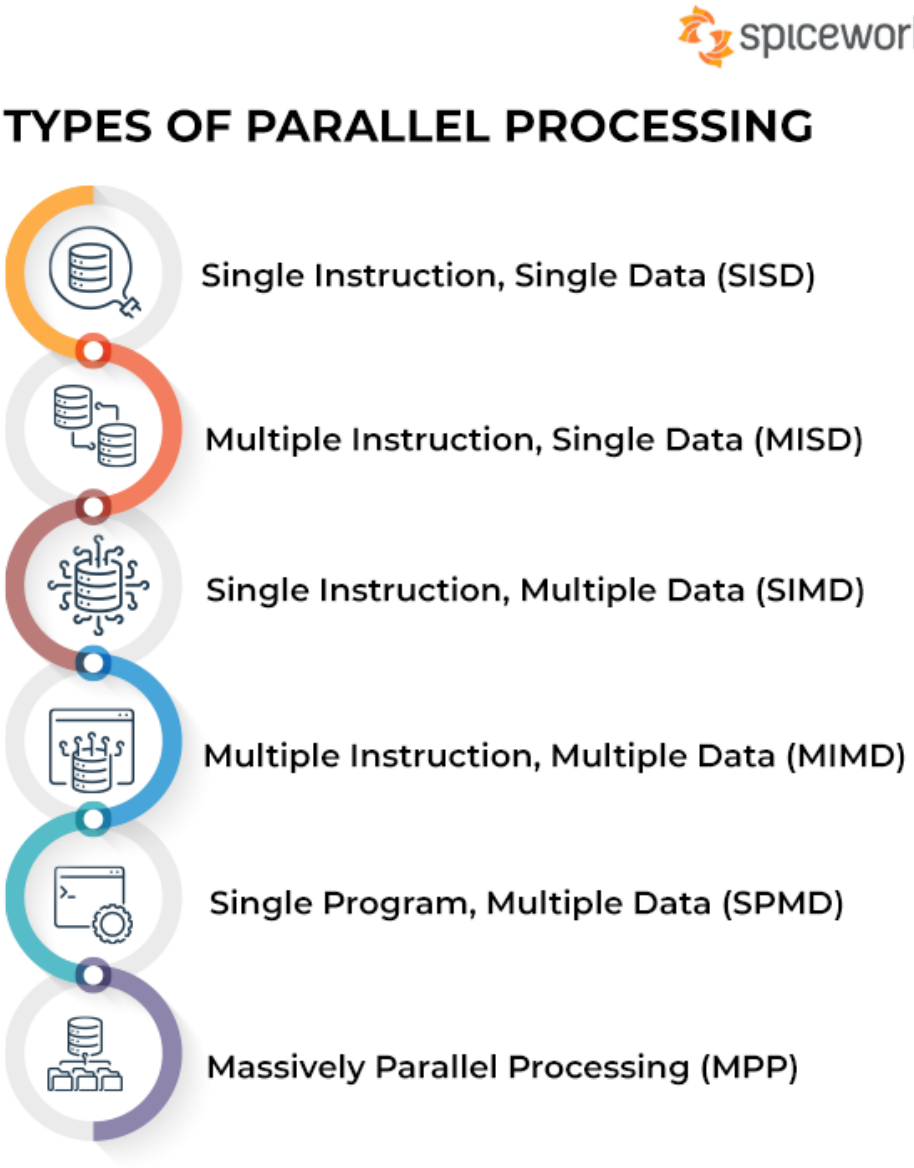


The primary purpose of parallel processing is to enhance the computer processing capability and increase its throughput, i.e. the amount of processing that can be accomplished during a given interval of time.

A parallel processing system can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously. The data can be distributed among various multiple functional units.

Types of parallel processing systems.

There are various varieties of parallel processing, such as MMP, SIMD, MISD, SISD, and MIMD, of which SIMD is probably the most popular. Single instruction multiple data, or SIMD, is a parallel processing type where a computer has two or more processors that all follow the same instruction set but handle distinct data types. Let us now take a look at the various kinds of parallel processing and how they work:



1. Single Instruction, Single Data (SISD)

In the type of computing called Single Instruction, Single Data (SISD), a single processor is responsible for simultaneously managing a single algorithm as a single data source. A computer organization having a control unit, a processing unit, and a memory unit is represented by SISD. It is similar to the current serial computer. Instructions are carried out sequentially by SISD, which may or may not be capable of parallel processing, depending on its configuration.

Sequentially carried-out instructions may cross over throughout their execution phases. There may be more than one functional unit inside a SISD computer. However, one control unit is in charge of all functional units. Such systems allow for pipeline processing or using numerous functional units to achieve parallel processing.

2. Multiple Instruction, Single Data (MISD)

Multiple processors are standard in computers that use the Multiple Instruction, Single Data (MISD) instruction set. While using several algorithms, all processors share the same input data. MISD computers can simultaneously perform many operations on the same batch of data. As expected, the number of operations is impacted by the number of processors available.

The MISD structure consists of many processing units, each operating under its instructions and over a comparable data flow. One processor's output becomes the input for the following processor. This organization's debut garnered little notice and wasn't used in architecture.

3. Single Instruction, Multiple Data (SIMD)

Computers that use the Single Instruction, Multiple Data (SIMD) architecture have multiple processors that carry out identical instructions. However, each processor supplies the instructions with its unique collection of data. SIMD computers apply the same algorithm to several data sets. The SIMD architecture has numerous processing components.

All of these components fall under the supervision of a single control unit. While processing numerous pieces of data, each processor receives the same instruction from the control unit. Multiple modules included in the shared subsystem aid in simultaneous communication with every CPU. This is further separated into organizations that use bit-slice and word-slice modes.

4. Multiple Instruction, Multiple Data (MIMD)

Multiple Instruction, Multiple Data, or MIMD, computers are characterized by the presence of multiple processors, each capable of independently accepting its instruction stream. These kinds of computers have many processors. Additionally, each CPU draws data from a different data stream. A MIMD computer is capable of running many tasks simultaneously.

Although MIMD computers are more adaptable than SIMD or MIMD computers, developing the sophisticated algorithms that power these machines is more challenging. Since all memory flows are changed from the shared data area transmitted by all processors, a MIMD computer organization incorporates interactions between the multiprocessors.

The multiple SISD operation is equivalent to a collection of separate SISD systems if the many data streams come from various shared memories.

5. Single Program, Multiple Data (SPMD)

SPMD systems, which stand for Single Program, Multiple Data, are a subset of MIMD. Although an SPMD computer is constructed similarly to a MIMD, each of its processors is responsible for carrying out the same instructions. SPMD is a message passing programming used in distributed memory computer systems. A group of separate computers, collectively called nodes, make up a distributed memory computer.

Parallel organisation.

Parallel organisation refers to organising multiple processors in a computer system to perform simultaneous operations, enhancing the computational speed and performance. The premise is 'Divide and Conquer' strategy which involves dividing a large problem into smaller sub-problems.

Benefits of Parallel organisation

Increased speed:

By performing tasks concurrently, parallel computing can significantly reduce processing time.

Improved performance:

Parallel architectures can handle larger datasets and more complex problems efficiently.

Enhanced reliability:

In some cases, parallel systems can be made more robust by having redundant processors.

Applications

Scientific computing: Modeling, simulations, and data analysis in fields like physics, chemistry, and engineering.

Database management: Handling large datasets and high-volume transactions.

Image and video processing: Rendering, manipulation, and analysis of images and videos.

Artificial intelligence: Training and inference of complex machine learning models.

These parallel organizations play a crucial role in designing high-performance computing systems capable of meeting the computational demands of modern applications across various domains.

Pipelining

Pipelining is an implementation technique whereby multiple instructions are overlapped in execution; it takes advantage of parallelism that exists among the actions needed to execute an instruction. Today, pipelining is the key implementation technique used to make fast CPUs. A pipeline is like an assembly line.

In computer architecture, pipelining is a technique used for improving processor throughput. It allows multiple instructions to be in different stages of execution simultaneously, thereby increasing the overall speed of instruction execution.

Here's how it works:

1. **Instruction Fetch (IF)**: The processor fetches the next instruction from memory.

2. **Instruction Decode (ID)**: The fetched instruction is decoded to determine what operation it represents and what operands are involved.

3. **Instruction Execute (EX)**: The instruction is executed, which involves performing the necessary arithmetic or logic operation.

4. **Memory Access (MEM)**: If the instruction involves accessing memory (e.g., loading or storing data), this stage retrieves or stores data from/to memory.

5. **Write Back (WB)**: The results of the instruction are written back to the appropriate register.

Each stage of the pipeline works concurrently, and while one instruction is being executed, the next instruction can be fetched, and so on. This overlap of instructions allows for better utilization of the processor's resources.

Instruction pipeline

An instruction pipeline is a technique used in the design of modern microprocessors, where multiple instructions are overlapped in execution.

The pipeline is divided into stages, with each stage responsible for a particular aspect of instruction execution, such as fetching, decoding, executing, and writing back results.

Each stage operates concurrently with the other stages, allowing multiple instructions to be processed simultaneously.

The goal of an instruction pipeline is to improve processor throughput by reducing the time wasted between instructions. While one instruction is being executed, other instructions can be fetched and decoded, improving overall efficiency.

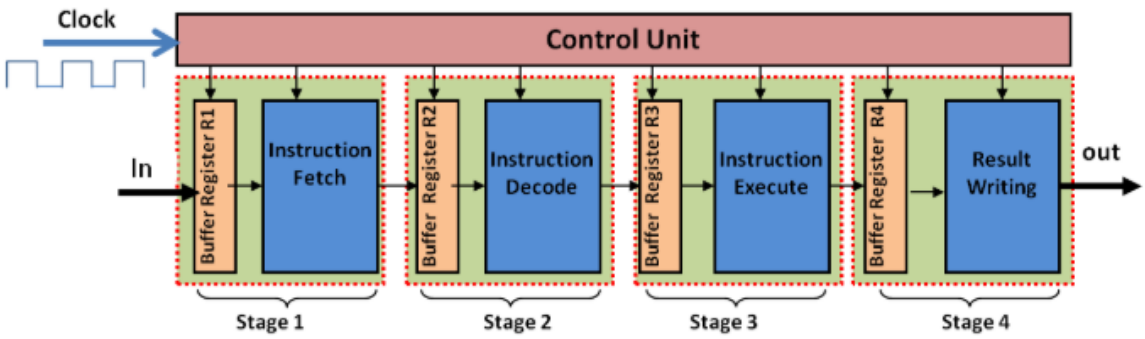


Figure 15.2 Four Stage Instruction Pipeline – an Example

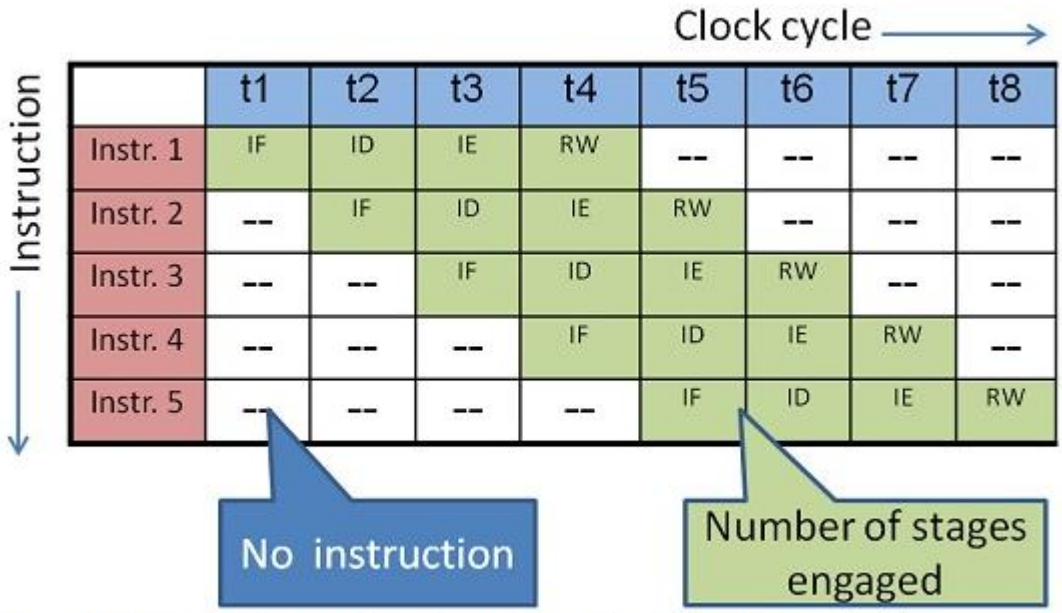


Figure 15.3 Phase diagram for a four-stage pipelined CPU

Arithmetic pipeline

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations, multiplication and various other computations.

An arithmetic pipeline is a specialized form of a pipeline used for performing arithmetic operations, such as addition, subtraction, multiplication, and division, in a sequential and efficient manner. Similar to an instruction pipeline, an arithmetic pipeline divides the arithmetic operation into multiple stages, each handling a specific aspect of the operation.

For example, in a multiplication pipeline, the stages might include operand fetch, partial product generation, partial product accumulation, and result generation.

By breaking down the arithmetic operation into smaller stages and processing them concurrently, arithmetic pipelines can significantly improve the speed of arithmetic computations.

RISC pipeline

RISC stands for Reduced Instruction Set Computers. It was introduced to execute as fast as one instruction per clock cycle. This RISC pipeline helps to simplify the computer architecture's design.

RISC (Reduced Instruction Set Computing): RISC is a CPU design strategy that emphasizes a simple and streamlined instruction set, typically with a smaller number of instructions and a more regular instruction encoding format. The idea behind RISC is to simplify the CPU's instruction decoding process, allowing for faster execution and more efficient pipelining. RISC processors often use a load-store architecture, where arithmetic and logic operations only operate on data loaded from memory into registers. Examples of RISC architectures include ARM, MIPS, and PowerPC.

RISC Pipeline: RISC pipeline refers to a technique used in CPU design to improve throughput by allowing multiple instructions to be processed simultaneously at different stages of execution. Instructions are divided into smaller, discrete stages (such as fetch, decode, execute, memory access, and writeback), and each stage operates concurrently. This allows for better utilization of the CPU's resources and can significantly improve performance. RISC architectures often make heavy use of pipelining due to their simpler instruction sets and regular instruction formats.

Super pipelining

Super pipelining simply allows a processor to improve its performance by running the pipeline at a higher clock rate. The higher clock rate is achieved by identifying the pipeline stages that determine the clock cycle (in other words, the most time consuming stages) and decomposing them into smaller stages.

Superpipelining is a technique that extends the concept of pipelining by breaking down the stages of instruction execution into even smaller, finer-grained stages. This allows for even higher clock frequencies and potentially greater performance, as each stage can be completed more quickly. However, superpipelining also introduces challenges such as

increased pipeline stalls and complexity in managing dependencies between instructions.

Superscalar processors

Superscalar Architecture is a microprocessor design type that allows processing of multiple instructions during a single clock cycle. This is achieved via functional units that independently operate, extracting and reordering instructions for efficient parallel execution.

Superscalar processors take advantage of instruction-level parallelism by executing multiple instructions simultaneously within the same pipeline stage. Unlike traditional pipelining, where each instruction progresses through the pipeline stages sequentially, superscalar processors can issue multiple instructions per clock cycle if there are enough resources available and if the instructions are independent of each other. This allows for greater instruction throughput and improved performance by exploiting parallelism at the instruction level.

Vector processing

Vector processing is a computer method that can process numerous data components at once. It operates on every element of the entire vector in one operation, or in parallel, to avoid the overhead of the processing loop.

Vector processing, also known as vector computing, is a type of processing paradigm that involves performing operations on multiple data elements simultaneously. In contrast to scalar processing, where operations are performed on individual data elements, vector processing enables the execution of operations on entire arrays or vectors of data in parallel

Vector processors are specialized hardware units designed to efficiently execute vector operations. These processors typically include specialized instruction sets optimized for performing operations on vectors, as well as hardware support for parallel execution.

One of the key advantages of vector processing is its ability to exploit data-level parallelism, which can lead to significant performance improvements for certain types of computations, such as numerical simulations, scientific computing, and multimedia processing. By

processing multiple data elements in parallel, vector processors can achieve higher throughput compared to scalar processors.

Vector processing has been used in various fields, including high-performance computing, graphics processing, and signal processing. Modern CPU architectures often include support for vector processing through technologies like SIMD (Single Instruction, Multiple Data) instructions, which enable parallel execution of operations on multiple data elements. Additionally, specialized accelerators like GPUs (Graphics Processing Units) are highly optimized for vector processing tasks and are commonly used in applications that require massive parallelism.

Array processing

Array processing is a wide area of research in the field of signal processing that extends from the simplest form of 1 dimensional line arrays to 2 and 3 dimensional array geometries. Array structure can be defined as a set of sensors that are spatially separated, e.g. radio antenna and seismic arrays.

Array processing in computer architecture typically refers to the use of specialized hardware or techniques to efficiently process data stored in arrays. This concept is particularly relevant in parallel computing and high-performance computing environments where large datasets need to be processed quickly.

Here are some key aspects of array processing in computer architecture:

1. **Vector Processors**: Vector processors are specialized CPUs that are optimized for performing operations on arrays of data. They can execute the same operation on multiple data elements simultaneously, often referred to as SIMD (Single Instruction, Multiple Data) processing. Vector processors excel at tasks like numerical simulations, image processing, and scientific computing.

2. **GPU Computing**: Graphics Processing Units (GPUs) are massively parallel processors originally designed for rendering graphics but now widely used for general-purpose computation, including array processing. GPUs consist of thousands of small processing cores that can

execute instructions in parallel, making them well-suited for tasks that involve processing large arrays of data, such as machine learning, computer vision, and simulations.

3. **Array Processors**: Array processors are specialized hardware units designed specifically for performing operations on arrays efficiently. They often include features like dedicated memory banks optimized for array access patterns and specialized instruction sets tailored for array processing tasks.

4. **Parallel Processing**: In modern computer architectures, parallel processing techniques are commonly employed to perform array processing tasks efficiently. This includes techniques like parallel loops, parallel computing frameworks (e.g., OpenMP, MPI), and distributed computing systems that distribute array processing tasks across multiple processors or nodes.

5. **Memory Hierarchy Optimization**: Efficient array processing often relies on optimizing data access patterns to take advantage of the memory hierarchy in computer systems. Techniques such as data prefetching, cache optimization, and memory locality are crucial for minimizing data transfer overhead and maximizing processing throughput.

6. **Instruction Set Architecture (ISA) Extensions**: Many modern CPU architectures include specialized instructions or extensions tailored for array processing tasks. For example, Intel's Advanced Vector Extensions (AVX) and ARM's NEON instruction set provide SIMD instructions to accelerate array processing operations.

Overall, array processing in computer architecture involves a combination of specialized hardware, parallel processing techniques, and software optimization to achieve high performance when working with arrays of data.

Example of SIMD array-processor.

Let's consider a simple example of a SIMD (Single Instruction, Multiple Data) array processor. In this example, we'll implement a basic vector addition operation using SIMD instructions.

Let's assume we have two arrays `A` and `B` of integers, and we want to add corresponding elements of these arrays to produce a result array `C`.

Here's how you might implement it using SIMD instructions in C/C++ with the help of libraries like Intel's SSE (Streaming SIMD Extensions) or ARM NEON:

```
#include <iostream>
#include <immintrin.h> // For SIMD instructions

const int ARRAY_SIZE = 8; // Assuming arrays have 8 elements

void vectorAdd(int* A, int* B, int* C) {
    // Load arrays A and B into SIMD registers
    __m256i vecA = _mm256_loadu_si256((__m256i*)A);
    __m256i vecB = _mm256_loadu_si256((__m256i*)B);

    // Perform SIMD addition
    __m256i vecC = _mm256_add_epi32(vecA, vecB);

    // Store result back to array C
    _mm256_storeu_si256((__m256i*)C, vecC);
}

int main() {
```

```
    int A[ARRAY_SIZE] = {1, 2, 3, 4, 5, 6, 7, 8};
    int B[ARRAY_SIZE] = {8, 7, 6, 5, 4, 3, 2, 1};
    int C[ARRAY_SIZE];

    vectorAdd(A, B, C);

    // Print the result
    std::cout << "Result: ";
    for (int i = 0; i < ARRAY_SIZE; ++i) {
        std::cout << C[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

In this code:

- We use `_mm256_loadu_si256()` to load arrays `A` and `B` into SIMD registers.
- `_mm256_add_epi32()` performs SIMD addition on the elements of `vecA` and `vecB`.
- Finally, `_mm256_storeu_si256()` stores the result back into array `C`.

This example is written for systems supporting AVX2 instructions (256-bit SIMD), but similar concepts can be applied to other SIMD instruction sets like SSE or ARM NEON.