

**ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY**

*College of Electrical and Mechanical Engineering*

*Department of Electrical and Computer Engineering*

*Computer Architecture*

*Group Members*

|  |  |
| --- | --- |
| **NAME** | **ID** |
| MUSSIE LEAKE | ETS0496/12 |
| NAOD MULUGETA | ETS0511/12 |
| NATHAN SHIBIRU | ETS0517/12 |
| NATNAEL ENGLIZ | ETS0528/12 |
| NAZRAWI MEDHIN | ETS0541/12 |
| OBSI ELIYAS | ETS0553/12 |
|  |  |

Submitted to: Mr. Netsanet G

JANUARY 2023

**1. Implement the following using the assembly programming on lmc:**

**a. Factorial of a number, n**

inp

sta final

brz oneval

sub one

sta iteration

sta counter

lda final

sta num

mult lda iteration

brz end

sub one

brz end

lda final

add num

sta final

lda counter

sub one

sta counter

sub one

brz next

bra mult

next lda final

sta num

lda iteration

sub one

sta iteration

sta counter

sub one

brz end

bra mult

end lda final

out

hlt

oneval lda one

out

hlt

final dat 0

counter dat 0

one dat 1

iteration dat 0

num dat 0

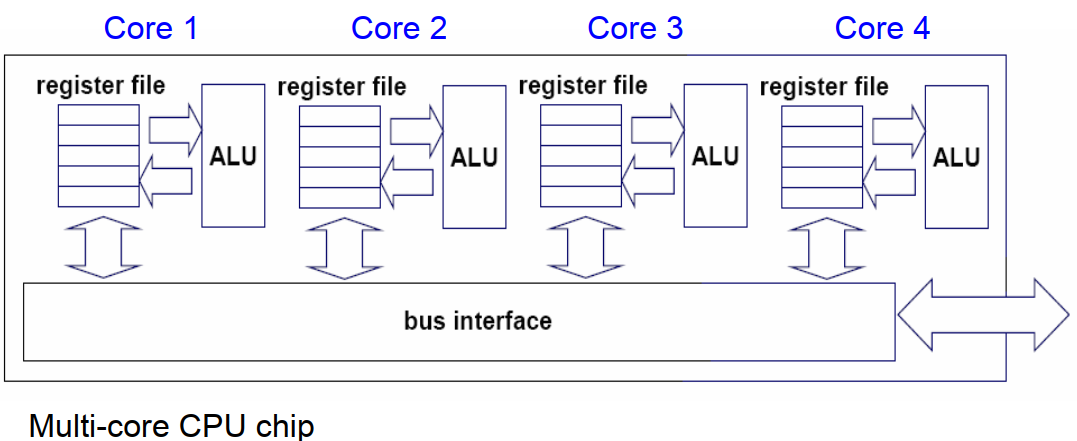
**b. Prime numbers from 0 to n, where n is the input to the program.**

**2.Report**

**Multi-core processors**

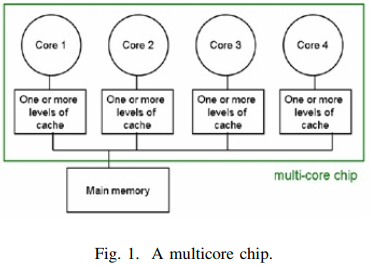
A single processor with multiple cores on a single chip is called a multi-core processor. The cores are functional groups made up of caches and compute units. This single chip's numerous cores work together to mimic the performance of a single faster CPU. Although a multi-core processor's individual cores may not always operate as quickly as the best single-core processors, they do enhance overall performance by handling more jobs concurrently. By knowing how single core and multi-core processors run applications, the speed improvement can be witnessed. When using a single core processor to run many applications, each program would be given its own time slice, with various time slices being given to the other programs. All of the other processes begin to lag behind if one of them takes longer than the others to finish. However, with multi-core processors, if you have several jobs that may be carried out concurrently in parallel, each of them will be carried out by a distinct core in parallel, enhancing performance. Instead of being clocked at a higher frequency, the chip's numerous cores' capacity to run programs simultaneously is what ultimately contributes to overall performance, making them more power-efficient and energy-efficient cores. Generally speaking, multi-core processors are partitioned so that the unused cores can be turned on or off as needed by the program, resulting in overall power dissipation savings.

Depending on the needs of the application, there are numerous approaches to deploy multi-core CPUs. It could be implemented as a collection of homogeneous cores, a collection of heterogeneous cores, or a collection of both. In a homogeneous core design, all of the CPU's cores are the same and use a divide-and-conquer strategy to boost total processor performance by splitting up computationally heavy tasks into less intensive ones and running them concurrently. Another important advantage of employing a homogeneous multi-core processor is that it is easier to meet time-to-market requirements due to its reduced design complexity, reusability, and verification work. Contrarily, heterogeneous cores address the problem of running a range of programs on a computer by utilizing specialized application-specific processor cores. A DSP core that handles multimedia applications requiring intense mathematical computations, a complicated core that handles computationally demanding applications, and a remedial core that handles less computationally difficult applications are a few examples.



In a single physical package, a multi-core CPU implements multiprocessing. Cores of a multi-core device may be coupled firmly or loosely by the designers. For instance, cores may or may not implement message forwarding or shared memory inter-core communication techniques, and they may or may not share caches. Network topologies including bus, ring, two-dimensional mesh, and crossbar are frequently used to link cores. Only identical cores are present in homogeneous multi-core systems; in contrast, non-identical cores (such as large. AMD Accelerated Processing Units have cores that don't even share the same instruction set, while LITTLE have heterogeneous cores that share the same instruction set. Multi-core systems' cores can implement architectures like VLIW, superscalar, vector, or multithreading, just like single-processor systems can. In various application fields, including general-purpose, embedded, network, digital signal processing (DSP), and graphics, multi-core processors are frequently employed (GPU)

**Internal Structure:** Multicore processors are systems with a central processing unit separated into several logical cores, each of which may have one or more private caches, as seen in figure 1.



In the diagram, there are four cores, each of which have one or more caches and is linked to the system's memory through an interconnection network.

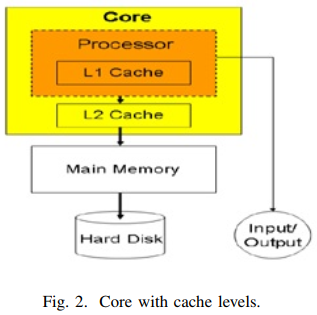


Figure 2 depicts an inside view of a single core. Figure 2 depicts a single-core CPU with a private L1 cache and a shared L2 cache [1, 2]. The distance from the main memory or the number of cycles required to access the main memory determines how many cache layers are needed. Homogeneous and heterogeneous multicore processors are the two different types of multicore CPUs. All of the cores of homogeneous multicore processors are identical, but heterogeneous multicore processors only have non-identical cores.

***Architecture of any multicore processor***

The multicore processor architecture may be roughly categorized into the following categories: application class, power/performance, processing elements, memory system, and accelerators/integrated peripherals.

1. **Application Class:** This multicore processor's architecture is primarily focused on meeting the needs of a certain application domain. This has a number of beneficial impacts, but frequently the multicore for a specific application domain cannot be used or has negative implications when used in other domains. Applications can be classified as either data processing- or control processing-dominated during their execution phase. Data-dominated procedures involve little to no data reuse and are carried out on a collection of data. This fact enables parallel processing with excellent throughput and performance for huge data sets. Examples of applications include wireless base band processing, audio processing, and picture processing. The application deals with a high level of conditional branching, parallelism, and data reuse in the control processing-dominated class. For instance, query processing, network processing, and data compression and decompression.
2. **Power/Performance:** Certain applications demand high power and performance standards. For instance, on Play-Stations, the game offers a real-time setting. Only applications that leverage multicore architecture with first-class performance design limitations may deliver this feeling. However, as many games require a long battery life for use on mobile devices, power is also a concern for these applications. Therefore, for such applications, the processor's architecture should be such that less power is consumed while still producing excellent performance.
3. **Processing Element:** The type of instruction set architecture, such as Reduced Instruction Set Computer (RISC) or Complex Instruction Set Computer, determines the multicore architecture (CISC). The architecture of a multicore processor is also defined by the processing element used in the core. There are two different sorts of cores based on the processing element: in-order cores and out-order cores. The device size and power consumption of in-order cores are modest, and they can operate with large applications that have more sensitive serial sections and higher levels of parallelism with ease. Out order cores require additional die space and are not appropriate for systems that use less power.
4. **Memory System:** The intrachip interface, consistency model, cache coherence support, and caches and their levels are all part of the memory system architecture. These decide how the cores will communicate, giving the system a high degree of parallelism and programmability. In essence, the consistency model specifies the order in which the instructions should be carried out. While weak consistency models are less complex and simpler to construct memory systems, strong consistency models include stringent ordering limitations. The cache setup essentially controls how many, how many tiers, and how much of a cache the system needs. The application determines how much cache is needed from an architectural perspective. The size of the cache will increase as data reuse increases. Better performance is a result of larger caches, but it also affects the die area and power budget. How close each processing element is to the main memory determines how many cache levels are necessary. Cache coherence and general communication between CPU components are handled through the intrachip connection. Bus, ring, crossbar, and network on chip comprise the connection for intercore communication. Cache coherence is a requirement for the programming paradigms that the architecture supports. The consistency of data that is visible to all other processor cores is defined by cache coherence.

**Issues in designing Multicore processor**

1. **Cache related issue**
   1. **Amount of cache**: The amount of cache space used by a multicore CPU depends on the application. Large caches are desirable for applications where data reuse is high. Greater performance will result from larger cache sizes, but the price will be higher. Larger cache sizes also result in faster access times.
   2. **Number of cache level**: Choosing how many cache levels a multicore cache can have is another caching issue. It is not necessary for each cache level in the core's cache to be equal. In essence, the distance from the main memory or the number of cycles required to access the main memory determines how many cache levels are needed. The cache levels will be higher and access times will be quicker as the number of cycles increases.
2. **Selection of core used:** a program divides up its duties among the processor's cores in order for the tasks to compute effectively. Regarding this, two concerns must be resolved: choosing the type of core that is compatible with the duties that are allocated to them and the number of cores required for a certain application.
3. **Consistency among cores:** The copy of a piece of data in each cache might not be the same because each core has its own private cache. Cache coherence, or consistency among cache, ensures that the other CPU cores see the same image of data stored in memory when performing computations. Both broadcast coherence and directory-based coherence can be used to implement cache coherence [3]. Only one processor may conduct an operation in broadcast coherence.

**Speed vs.** **Power:** Applications today need great performance, which necessitates quick computing speeds. If additional cores are employed or the cache capacity is larger, a multicore processor's speed can be boosted. The system will need more static and dynamic power if the number of processing elements is raised. If all the cores operate at their full potential, overheating may also be an issue. Therefore, methods or plans should be used to obtain excellent performance at a lower rate of power consumption.

# Parallel Processing

Traditionally, the computer has been viewed as a sequential machine. Most computer programming languages require the programmer to specify algorithms as sequences of instructions. Processors execute programs by executing machine instructions in a sequence and one at a time. Each instruction is executed in a sequence of operations (fetch instruction, fetch operands, perform operation, store results).

This view of the computer has never been entirely true. At the micro-operation level, multiple control signals are generated at the same time. Instruction pipelining, at least to the extent of overlapping fetch and execute operations, has been around for a long time. Both of these are examples of performing functions in parallel. This approach is taken further with superscalar organization, which exploits instruction-level parallelism. With a superscalar machine, there are multiple execution units within a single processor, and these may execute multiple instructions from the same program in parallel.

For the purpose of increasing the computational speed of computer system, the term ‘parallel-processing employed to give simultaneous data-processing operations is used to represent a large class. In addition, a parallel processing system is capable of concurrent data processing to achieve faster execution times.

## Types of Parallel Processer Systems

A taxonomy first introduced by Flynn is still the most common way of categorizing systems with parallel capability. Flynn proposed the following categories of computer systems:

1. **Single instruction, single data (SISD) stream:** A single processor executes a single instruction stream to operate on data stored in a single memory. Uniprocessors fall into this category.
2. **Single instruction, multiple data (SIMD) stream:** A single machine instruction controls the simultaneous execution of a number of processing elements on a lockstep basis. Each processing element has an associated data memory, so that each instruction is executed on a different set of data by the different processors. Vector and array processors fall into this category.
3. **Multiple instruction, single data (MISD) stream:** A sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. This structure is not commercially implemented.
4. **Multiple instruction, multiple data (MIMD) stream:** A sequence of processors simultaneously execute different instruction sequences on different data sets. Symmetric multiprocessors (SMPs), clusters, and non-uniform memory access(NUMA) systems fit into this category.

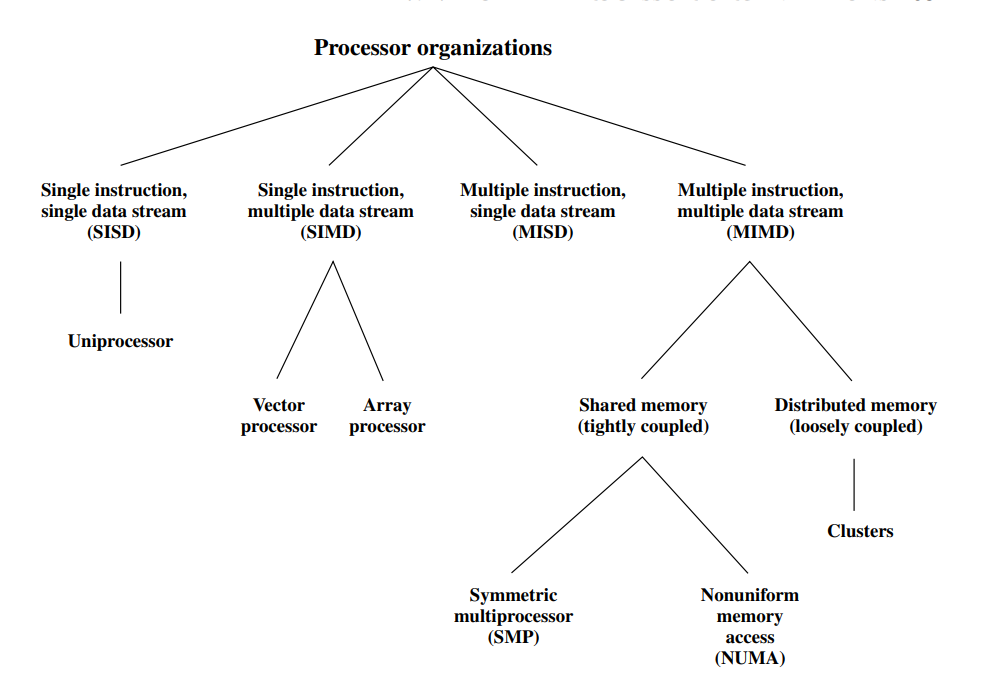


Figure: A Taxonomy of Parallel Processor Architectures

With the MIMD organization, the processors are general purpose; each is able to process all of the instructions necessary to perform the appropriate data transformation. MIMDs can further subdivided by the means in which the processors communicate in the figure above.

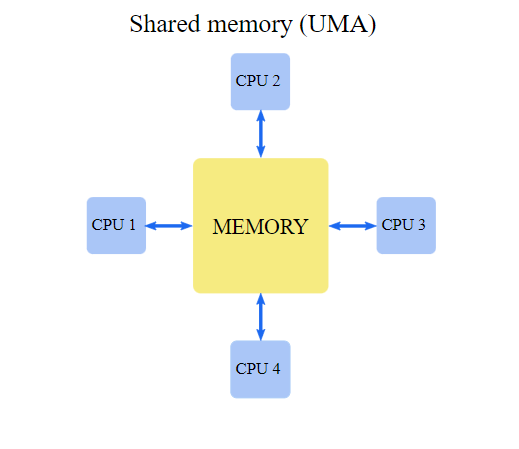
## Memory Organizations of Parallel Computers

### Shared memory

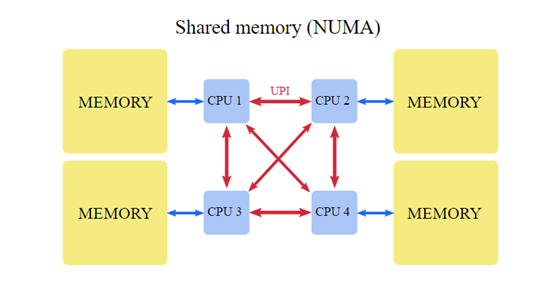
* Multiple processors can operate independently but share the same memory resources.
* All processors have equal access to data and instructions in this memory
* Changes in a memory done by one processor are visible to all other processors.

Based upon memory access times shared memory computers can be divided into two categories:

* Uniform Memory Access (UMA)
* Non-Uniform Memory Access (NUMA)



* All processors share memory uniformly, i.e., access time to a memory location is independent on which of the processors.



* Used in multiprocessor systems.
* When many CPUs are trying to access the same memory, they can be “starved” of data because only one processor can access the computer’s memory at a time.
* NUMA attempts to address this problem by providing separate memory for each processor.
* CPUs are physically linked using a fast interconnect.
* A CPU can access its local memory faster than non-local memory (memory local to another processor or memory shared between processors).

In a shared memory system, it is only necessary to build a data structure in memory and pass references to the data structure to parallel subroutines. For example, a matrix multiplication routine that breaks matrices into a set of blocks only needs to pass the indices of each block to the parallel subroutines.

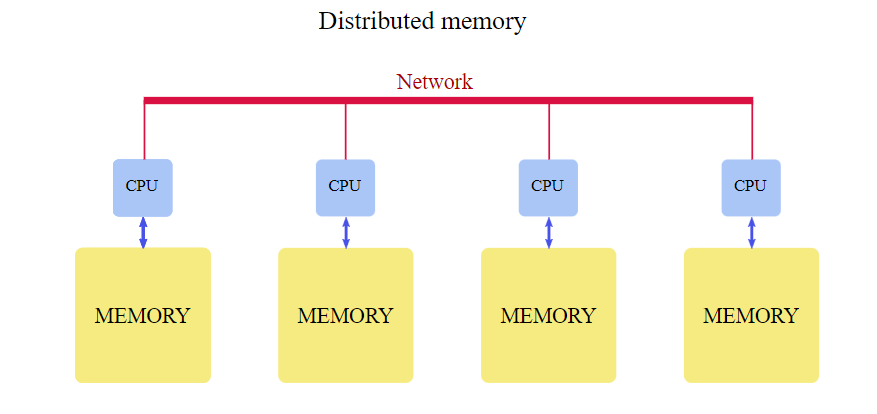
Advantages

* User-friendly programming.
* Fast data sharing due to a close connection between CPUs and memory

Disadvantages

* Not highly scalable. Adding more CPUs increases traffic on the shared memory-CPU path,
* Lack of data coherence. The change in the memory of one CPU needs to be reflected to the other processors, otherwise, the different processors will be working with incoherent data.
* The programmer is responsible for synchronization.

### Distributed memory



* In a distributed memory system, each processor has a local memory that is not accessible from any other processor
* Programs on each processor interact with each other using some form of a network interconnect (Ethernet, Infiniband, Quadrics, etc.).
* A distributed memory program must create copies of shared data in each local memory. These copies are created by sending a message containing the data to another processor.

In the matrix multiplication example, the controlling process would have to send messages to other processors. Each message would contain all submatrices required to compute one part of the result. A drawback to this memory organization is that these messages might have to be quite large.

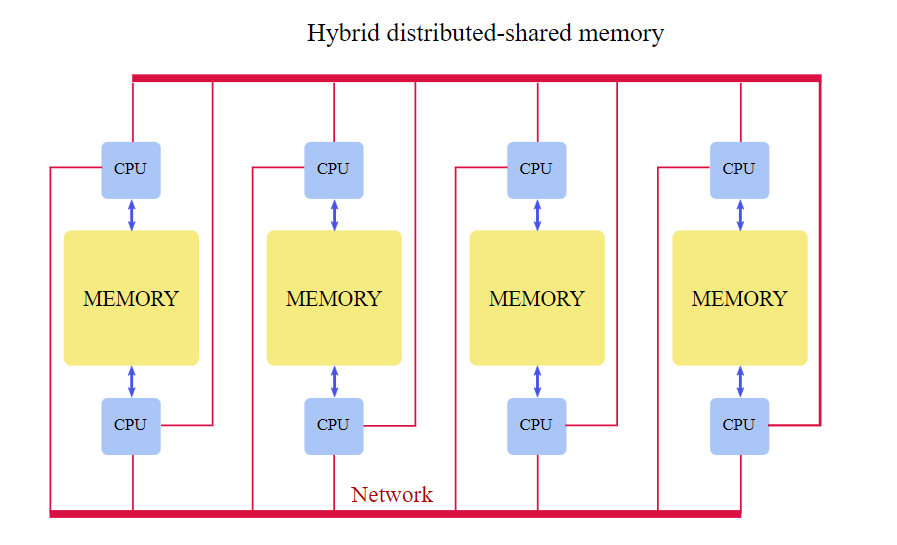
Advantages

* Each processor can use its local memory without interference from other processors.
* There is no inherent limit to the number of processors; the size of the system is constrained only by the network used to connect processors.

Disadvantages

* The complexity of programming: the programmer is responsible for many of the details associated with data communication between processors.
* It may be difficult to map complex data structures from global memory, to distributed memory organization.
* Longer memory access times.

### Hybrid Distributed-Shared Memory



Practically all HPC computer systems today employ both shared and distributed memory architectures.

* The shared memory component can be a shared memory machine and/or graphics processing units (GPU).
* The distributed memory component is the networking of multiple shared memory/GPU machines.

The important advantage is increased scalability. Increased complexity of programming is an important disadvantage.

# 8086 Microprocessor

8086 Microprocessor is an enhanced version of 8085Microprocessor that was designed by Intel in 1976. It is a 16-bit Microprocessor having 20address lines and16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily. It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

Features of 8086

The most prominent features of a 8086 microprocessor are as follows –

* It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.
* It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
* It is available in 3 versions based on the frequency of operation −

1. 8086 → 5MHz
2. 8086-2 → 8MHz
3. (c)8086-1 → 10 MHz

* It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.
* Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue.
* Execute stage executes these instructions.
* It has 256 vectored interrupts.
* It consists of 29,000 transistors.

Architecture of 8086

Microprocessor is divided into two functional units, i.e., EU (Execution Unit) and BIU (Bus Interface Unit).

**EU (Execution Unit)** Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions. Its function is to control operations on data using the instruction decoder & ALU. EU has no direct connection with system buses as shown in the above figure, it performs operations over data through BIU.

It has the following functional parts –

**ALU** It handles all arithmetic and logical operations, like +, −, ×, /, OR, AND, NOT operations.

**Flag Register** It is a 16-bit register that behaves like a flip-flop, i.e. it changes its status according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups Conditional Flags and Control Flags.

**Conditional Flags** It represents the result of the last arithmetic or logical instruction executed. Following is the list of conditional flags

* + - * 1. Carry flag − This flag indicates an overflow condition for arithmetic operations.
        2. Auxiliary flag − When an operation is performed at ALU, it results in a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.
        3. Parity flag − This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1’s, then the Parity Flag is set. For odd number of 1’s, the Parity Flag is reset.
        4. Zero flag − This flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.
        5. Sign flag − This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.
        6. Overflow flag − This flag represents the result when the system capacity is exceeded.

**Control Flags** controls the operations of the execution unit. Following is the list of control flags

1. Trap flag − It is used for single step control and allows the user to execute one instruction at a time for debugging. If it is set, then the program can be run in a single step mode.
2. Interrupt flag − It is an interrupt enable/disable flag, i.e. used to allow/prohibit the interruption of a program. It is set to 1 for interrupt enabled condition and set to 0 for interrupt disabled condition.
3. Direction flag − It is used in string operation. As the name suggests when it is set then string bytes are accessed from the higher memory address to the lower memory address and vice-a-versa.

**General purpose register**: There are 8 general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be used individually to store 8-bit data and can be used in pairs to store 16bit data. The valid register pairs are AH and AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively

* AX register − It is also known as accumulator register. It is used to store operands for arithmetic operations.
* BX register − It is used as a base register. It is used to store the starting base address of the memory area within the data segment.
* CX register − It is referred to as counter. It is used in loop instruction to store the loop counter.
* DX register − This register is used to hold I/O port address for I/O instruction.

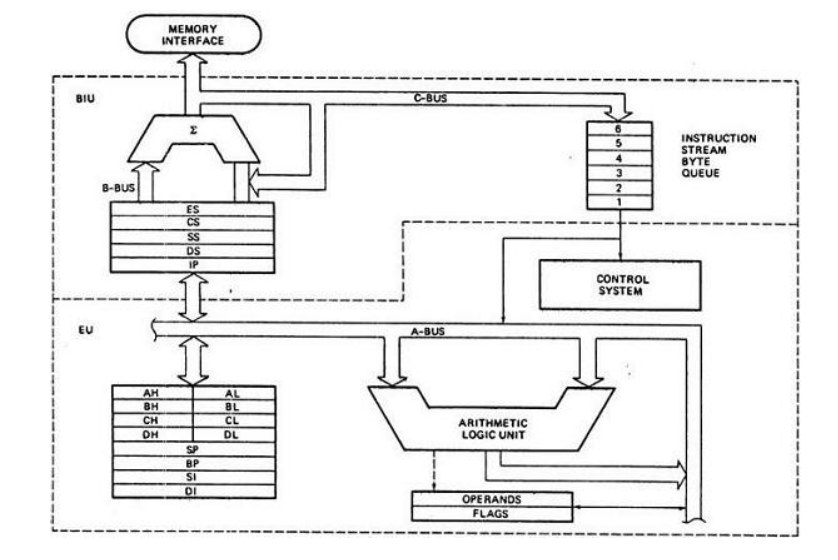
**Stack pointer register** It is a 16-bit register, which holds the address from the start of the segment to the memory location, where a word was most recently stored on the stack.

**BIU (Bus Interface Unit)** BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory. EU has no direction connection with System Buses so this is possible with the BIU. EU and BIU are connected with the Internal Bus.

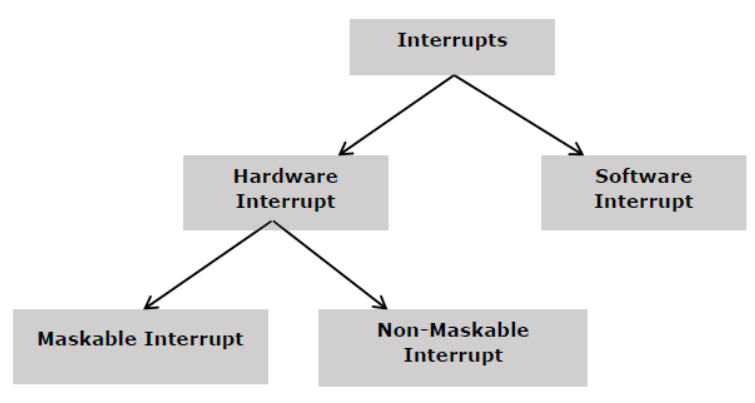
The Bus Interface Unit (BIU) generates the 20-bit physical memory address and provides the interface with external memory (ROM/RAM). 8086 has a single memory interface. To speed up the execution, 6- bytes of instruction are fetched in advance and kept in a 6-byte Instruction Queue while other instructions are being executed in the Execution Unit (EU). Hence after the execution of an instruction, the next instruction is directly fetched from the instruction queue without having to wait for the external memory to send the instruction. This is called pipe-lining and is helpful for speeding up the overall execution process.

It has the following functional parts –

* Instruction queue − BIU contains the instruction queue. BIU gets upto 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed.
* Fetching the next instruction while the current instruction executes is called pipelining.
* Segment register − BIU has 4 segment buses, i.e. CS, DS, SS& ES. It holds the addresses of instructions and data in memory, which are used by the processor to access memory locations. It also contains 1 pointer register IP, which holds the address of the next instruction to executed by the EU.
  + CS − It stands for Code Segment. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.
  + DS − It stands for Data Segment. It consists of data used by the program andis accessed in the data segment by an offset address or the content of other register that holds the offset address.
  + SS − It stands for Stack Segment. It handles memory to store data and addresses during execution.
  + ES − It stands for Extra Segment. ES is additional data segment, which is used by the string to hold the extra destination data.
* Instruction pointer − It is a 16-bit register used to hold the address of the next instruction to be executed.



The following image shows the types of interrupts we have in a 8086 microprocessor



**Hardware Interrupts**

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor. The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

**NMI**

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR)and it is of type 2 interrupt.

When this interrupt is activated, these actions take place –

• Completes the current instruction that is in progress.

• Pushes the Flag register values on to the stack.

• Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.

• IP is loaded from the contents of the word location 00008H.

• CS is loaded from the contents of the next word location 0000AH.

• Interrupt flag and trap flag are reset to 0.

**INTR**

The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.

The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends ‘0’ on INTA pin twice. The first ‘0’ means INTA informs the external device to get ready and during the second ‘0’ the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.

These actions are taken by the microprocessor –

* First completes the current instruction.
* Activates INTA output and receives the interrupt type, say X.
* Flag register value, CS value of the return address and IP value of the return address are pushed on to the stack.
* IP value is loaded from the contents of word location X × 4
* CS is loaded from the contents of the next word location.
* Interrupt flag and trap flag is reset to 0

**Software Interrupts**

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes “INT- Interrupt instruction with type number” It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

**Addressing modes**

There are 8 different addressing modes

* **Immediate addressing mode** The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode.
* **Register addressing mode** It means that the register is the source of an operand for an instruction.
* **Direct addressing mode** The addressing mode in which the effective address of the memory location is written directly in the instruction.
* **Register indirect addressing mode** This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI & SI.
* **Based addressing mode In this addressing mode**, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement.
* **Indexed addressing mode** In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements.
* **Based-index addressing mode** In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register.
* **Based indexed with displacement mode** In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement.

**3.Detailed discussion on architectures**

**CISC**

CISC stands for Complex Instruction Set Computer chips, which are memory-efficient and simple to program. The CISC philosophy was frequently used in large computers like the PDP-11 and the DEC system 10 and 20 machines because the initial machines were written in assembly language and memory was slow and expensive.

To make compiler development easier, CISC was created. It transfers the majority of the responsibility for creating machine instructions to the processor. For instance, a CISC processor would already be able to calculate a square root without the need for a compiler to construct complicated machine instructions.

The CISC technique aims to cut down on the number of cycles per instruction by lowering the number of instructions per program. The CISC architecture is used to build computers that have lower memory costs. Huge programs require more storage, which raises the cost of memory as well as the price of large memory. In order to address these problems, it is possible to reduce the number of instructions needed for each system by consolidating the number of operations into a single instruction, increasing the complexity of the instructions in the process.

CISC instruction sets share the following traits:

• A two-operand format where the source and destination of the instructions are different. Commands can be registered to memory, registered to register, and registered to register.

• Instructions of variable lengths, whose lengths frequently change depending on the addressing method.

• Instructions that must be executed across a number of clock cycles.

Due to the requirement for a single instruction to support several addressing modes, complex instruction-decoding circuitry is required.

• A few registers that are general-purpose. This is due to the restricted amount of chip area not devoted to instruction decoding, execution, and microcode storage as well as the existence of instructions that can be executed directly on memory.

• Several registries for various purposes. For the stack pointer, interrupt management, and other purposes, specific registers are often set in CISC architectures.

• A register called "Condition code" that is set by most instructions. This register keeps track of specific error circumstances and indicates whether the outcome of the most recent operation is less than, equal to, or higher than zero.

When they were first developed, CISC machines made use of current technologies to boost computer performance. Microprogramming is substantially less expensive than hardwiring a control unit and is just as simple to implement as assembly language.

• Because it was simple to micro-code new instructions, designers were able to create CISC machines that were upwardly compatible, meaning that a new computer could execute the same programs as older ones because it would have a superset of the older computers' instructions.

• Fewer instructions might be used to complete a task as each instruction gained in capability. This utilized the relatively slow main memory more effectively.

• The compiler does not need to be as complex because microprogram instruction sets can be created to correspond to the features of high-level languages.

Principal features of a CISC design »1) A large number of instructions, often between 100 and 250

2) A few infrequently used instructions that carry out particular duties

3) A wide range of addressing options

4) Formats for instruction with variable length

5) Memory-based operand manipulation instructions

**Addressing Modes in CISC**

2. Variable-length instructions are a result of CISC processor designers' choice to offer a range of addressing options. For instance, if an operand is in memory rather than a register, the length of the instruction rises.

This is due to the fact that memory addresses must be specified as part of instruction encoding, which requires many more bits.

b. This makes scheduling and decoding of instructions more difficult. Because there are so many different instruction types available, different amounts of clocks are needed to perform each instruction.

c. This once more causes issues with lesson planning and scheduling.

**Advantages of CISC Processors**

1. High-level programs or statement languages are easily converted into assembly or machine language in CISC processors by the compiler.

2. The code is rather small, which reduces the amount of RAM needed.

3. There is a very small RAM need for each CISC to store an instruction.

4. A single instruction needs the completion of numerous low-level operations.

5. To control power consumption, CISC develops a procedure that modifies voltage and clock speed.

6. It needs fewer instructions than the RISC to carry out the identical task.

**Disadvantages of CISC Processors**

1. CISC chips are slower than RISC chips to execute per instruction cycle on each program.

2. The performance of the machine decreases due to the slowness of the clock speed.

3. Executing the pipeline in the CISC processor makes it complicated to use.

4. The CISC chips require more transistors as compared to RISC design.

5. In CISC it uses only 20% of existing instructions in a programming event.

**Examples of CISC:**

**System/360**

The Variant 30, the slowest System/360 model introduced in 1964, could execute up to 34,500 instructions per second and had memory ranging from 8 to 64 KB. Later came high-performance models. Up to 16.6 million instructions may be processed every second by the 1967 IBM System/360 Model 91 computer. A big installation could have as little as 256 KB of main storage, but 512 KB, 768 KB, or 1024 KB was more typical. The larger 360 models could have up to 8 MB of main memory, however that much main memory was exceptional. For some models, slower (8 microsecond) Large Capacity Storage (LCS) of up to 8 megabytes was also an option.

**Motorola 6800**.

The Motorola 6800 is an 8-bit microprocessor that was created and released in 1974. The M6800 Microcomputer System, which also comprised RAM, ROM, parallel and serial interface ICs, and other support chips, had the MC6800 CPU. The M6800 family of ICs' single five-volt power supply need, at a time when the majority of other microprocessors required three voltages, was an important design element. The 6800 includes an 8-bit bi-directional data bus and a 16-bit address bus that can directly access 64 KB of memory. It includes 197 opcodes altogether and 72 instructions with seven addressing modes. Clock frequency for the original MC6800 was up to 1 MHz. Later models have a 2 MHz maximum clock frequency.

Motorola offered an entire assembly language development system in addition to the ICs. The customer might utilize the software on a local minicomputer system or a remote timeshare computer. The M6800 ICs were used to construct the Motorola Exorciser, a desktop computer that could be used for developing and debugging new designs. Datasheets for every IC were supplied in a comprehensive documentation package, together with two guides on programming in assembly language and a 700-page application guide that explained how to construct a point-of-sale terminal—a computerized cash register—based on the 6800.

**RISC**

Reduced instruction set computer, or RISC, is a type of microprocessor created to carry out fewer different kinds of computer instructions so that it can run at a faster rate (perform more millions of instructions per second, or MIPS). A larger list or set of computer instructions tends to make the microprocessor more complex and slower to operate since each instruction type that a computer must do requires additional transistors and circuitry.

The RISC concept was developed in 1974 by John Cocke of IBM Research in Yorktown, New York, who demonstrated that around 20% of a computer's instructions performed 80% of the work. The IBM PC/XT was the first computer to use this breakthrough in 1980. The RISC idea has encouraged a more careful approach to microprocessor design. Design factors include how successfully an instruction can be mapped to the microprocessor's clock speed (ideally, an instruction should be completed in one clock cycle), how basic the needed architecture is, and how much work the microchip itself can complete without the aid of software.

There are some design elements that set it apart from the competition:

• **Execution time for one cycle**. CPI (clock per instruction) for RISC processors is one cycle. This is caused by the CPU's optimization of every instruction and a process known as pipelining.

• **Pipelining**. a method for processing instructions more quickly that enables the simultaneous execution of different parts or stages of an instruction.

• There are a lot of registers. A greater number of registers are typically used in RISC design in order to reduce the quantity of memory interactions.

**Typical Characteristics of RISC Architecture**

The key findings that encouraged designers to think about CISC design possibilities were:

a. **Simple guidelines** because they bridge the semantic gap, sophisticated instructions were heavily anticipated by CISC architectural designers. It turns out that compilers largely disregard these directives in practice. This has been demonstrated by a number of empirical research. The usage of various meanings by high-level languages is one explanation for this. For instance, the C for loop's semantics differ slightly from those of other languages. As a result, compilers frequently use simpler instructions to create the code.

b. **A Few Data Types**: the CISC ISA typically supports a wide range of data structures, from straightforward data types like characters and integers to more intricate ones like records and structures. According to empirical data, sophisticated data structures are utilized only sometimes.

Therefore, it is advantageous to build a system that supports a few simple data types effectively and allows for the synthesis of the complicated data kinds that are missing.

c**. Simple Addressing Modes**: many addressing modes are offered by CISC designs. The two main drivers are to:

* support complicated data structures and
* offer operand access flexibility.

1. Issues Raised. This offers flexibility, but it also creates issues. First, depending on where the operands are located, it results in varied instruction execution times.
2. It results in variable-length instructions, secondly. The IA-32 instruction length, for instance, can vary from 1 to 12 bytes. Variable instruction lengths result in ineffective scheduling and decoding of instructions.

d. Exactly the same general-purpose registers. the ability to use any register in any situation, simplifying compiler design (even if there are typically separate floating-point registers).

e. Harvard-based architecture. Because the CPU has a separate instruction and data cache, changing the memory where code is stored might not have any impact on the instructions that are executed by the processor—at least not until a special synchronization instruction is issued. RISC designs are also more likely to use a Harvard memory model, where the instruction stream and the data stream are conceptually separated. On the plus side, this enables simultaneous access to both caches, which frequently enhances performance.

**Advantages of RISC Processor**

• If making a new microprocessor simpler is one of its goals, this can speed up the development and testing process.

• Operating system and application programmers will find it simpler to write code with a reduced instruction set if they employ the microprocessor's instructions.

• Higher-level language compilers now write more efficient code than in the past since they have always tended to use the smaller set of instructions to be found on a RISC machine, which is made possible by the simplicity of RISC.

• The cost of RAM has dropped significantly. 1MB of DRAM cost roughly $5,000 in 1977.

• In 1994, the identical amount of RAM only cost $6. (When adjusted for inflation). The RISC usage of RAM and concentration on software has become perfect due to the advancement of compiler technology.

**Disadvantages of RISC Processor**

1. The performance of the RISC processor might vary depending on the code that is run because a cycle's worth of instructions may depend on one another.

2. Complex instructions are frequently used by programmers and compilers.

3. RISC processors need highly quick memory to store a variety of instructions that need a lot of cache memory to react to the command quickly.

**RISC Processors (Examples)**

**Digital Equipment Corporation (DEC)** - Alpha, formerly known as Alpha AXP, is a 64-bit RISC instruction set architecture (ISA) that was created by DEC to take the role of the 32-bit VAX complex instruction set computer (CISC) ISA and its implementations.

a. DEC was the company that first designed and produced the microprocessors that used Alpha.

b. These microprocessors were primarily utilized in a number of DEC workstations and servers, which eventually served as the foundation for nearly all of their mid-to-high-end product lines.

c. A number of independent suppliers also created Alpha systems, including motherboards in the PC form factor.

**SuperH**

Hitachi created SuperH (SH), a RISC instruction set architecture (ISA) for 32-bit computers. Microprocessors and microcontrollers used in embedded systems implement it. It falls into the following main categories:

a. SH-1, which is used in microcontrollers for highly embedded applications (CD- ROM drives, major appliances, etc.)

b. SH-2. Used in microcontrollers with higher performance requirements, as well as in networking applications or engine control units in automobiles, as well as in gaming consoles like the Sega Saturn. The SH-2 has also been used in numerous applications involving motor control.

c. SH-DSP, originally created for the market for mobile phones, afterwards utilized in numerous consumer applications needing DSP performance for JPEG compression, etc.

d. SH-3, which has a long history in the market for automotive navigation systems and is used for mobile and handheld devices like the Jornada.

e. SH-3 DSP. used mostly in networking applications and multimedia interfaces, as well as in printers and fax machines.

f. SH-4. Used in applications where high performance is required, such as set-top boxes, video game consoles, and automobile multimedia terminals.

g. SH-5 is used in high-end multimedia applications.

**RISC vs. CISC**

Comparison of the two using an example:

Memory Multiplication of Two Numbers Locations in the main memory are numbered from (row) 1: (column) 1 to (row) 6: (column) 4. All computations must be performed by the execution unit. But only data that has been loaded into one of the six registers can be used by the execution unit (A, B, C, D, E, or F). Let's imagine we want to determine the product of two numbers that are stored at locations 2:3 and 5:2, then store the result back at position 2:3.

**The CISC Approach**: CISC architecture's main objective is to use the least number of assembly lines necessary to execute a task. This is accomplished by designing processing hardware that can comprehend and carry out a number of functions. A CISC processor would be equipped with a special instruction for this task (say "MUL").

a. This instruction multiplies the operands in the execution unit before storing the result in the appropriate register after loading the two values into separate registers.

b. As a result, just one instruction is required to multiply two numbers:

MUL 2:3, 5:2

c. MUL falls under the category of "complex instruction."

d. It doesn't require the programmer to explicitly call any loading or saving functions; instead, it operates directly on the memory banks of the computer.

e. It resembles a command in a higher-level language in a lot of ways. This command is equivalent to the C phrase "a = an x b," for example, if we let "a" represent the value of 2:3 and "b" represent the value of 5:2.

**The RISC Approach**: only straightforward commands that can be completed in a single clock cycle are used by RISC processors. As a result, the command "MUL" mentioned above could be broken into three different commands:

A "LOAD" command loads data into a register; a "PROD" command calculates the product of two operands inside a register; and a "STORE" command transfers data from a register to the memory banks.

d. A programmer would need to write four lines of assembly code in order to carry out the precise succession of steps outlined in the CISC approach:

2:3 LOAD A 5:2 LOAD B

A, B, PROD, STORE 2:3, A