Computer architecture

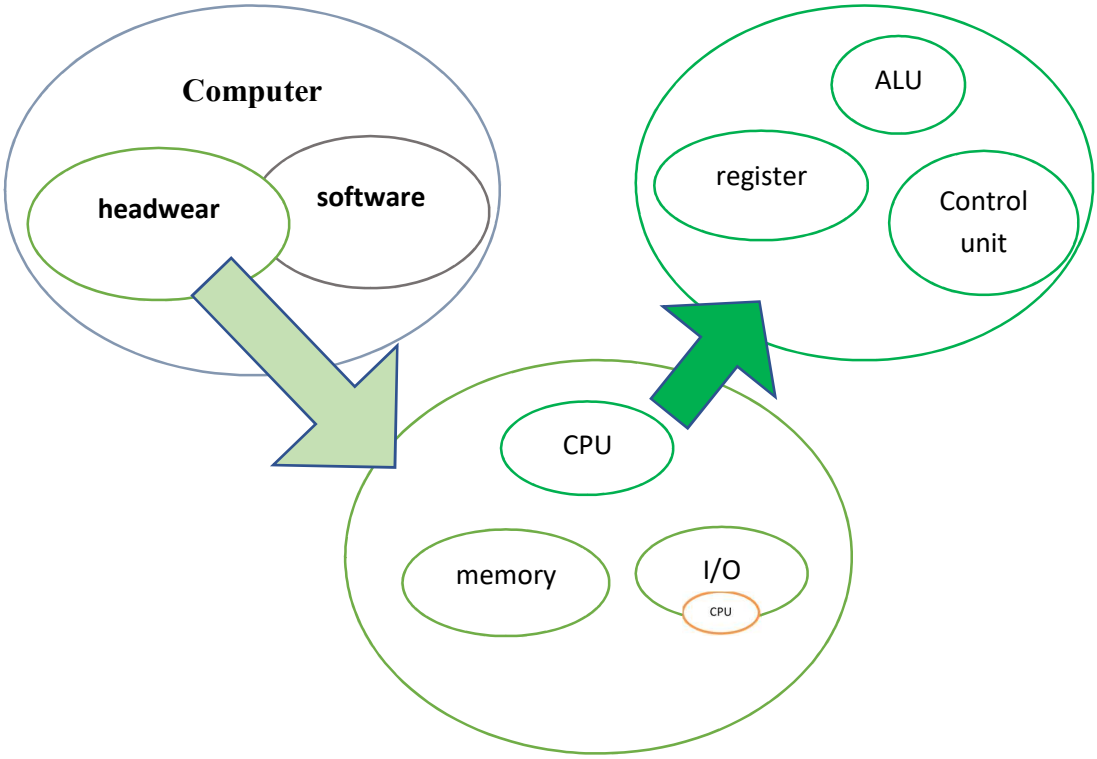
Computer architecture: is the design of computers including their instruction sets, hardware components and system organization.

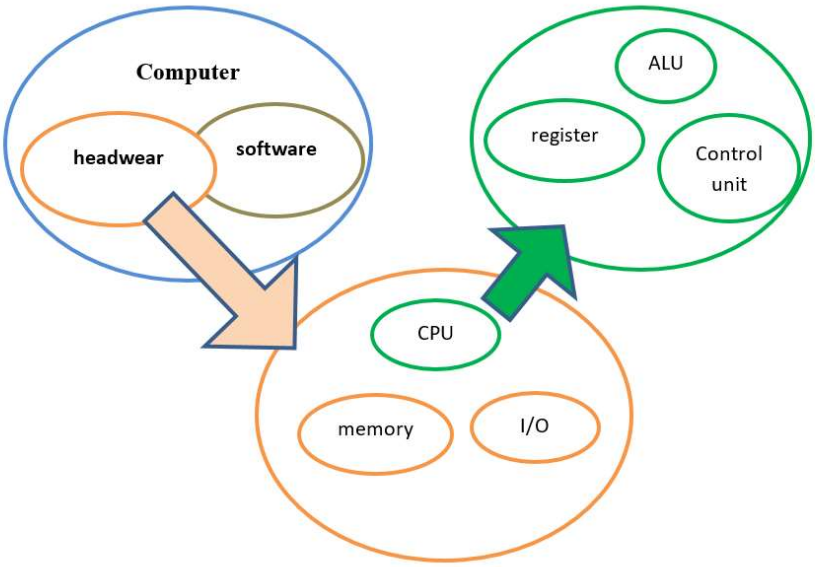
There are two parts to computer architecture:

1. Instruction set architecture(ISA).
2. Hardware system architecture (HSA).

ISA includes the specification that determine how machine language programmers will interact with the computer.
A computer is generally viewed in terms of its ISA, which determines the computational characteristics of the computer.

HAS deals with the computer's major hardware subsystems, including its central processing unit(CPU), I/O system and storage system. HAS determines how efficiently the machine will operate.





Classification of computer architectures:

1. Von Neumann architecture

also known as the **von Neumann model** is a computer architecture based on a 1945 description by the mathematician and physicist John von Neumann and others.

Von Neumann machine meet the following criteria:

1. It has three basic hardware subsystems
   - CPU
   - The main memory system
   - I/O system
2. It is a stored program computer.
3. It carries out instruction sequentially. The CPU executes one program at a time.

The word has evolved to mean any stored-program computer in which an instruction fetch and a data operation cannot occur at the same time because they share a common bus. This is referred to as the von Neumann often limits the performance of the system.[3]

The design of a von Neumann architecture machine is simpler than a Harvard architecture machine—which is also a stored-program system but has one dedicated set of address and data buses for reading and writing to memory, and another set of address and data buses to fetch instructions.

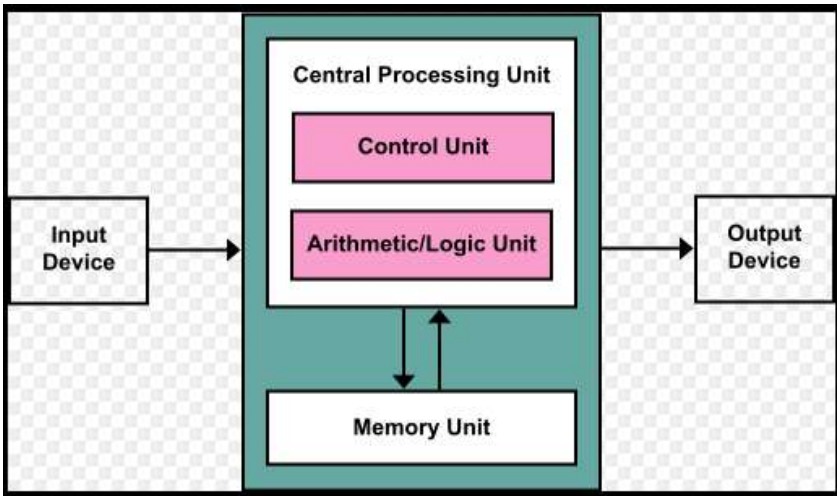


Fig. 1: Von Neumann architecture

2. Non-Von Newman Machine:

The classification includes the following categories:
1.Single Instruction stream, Single data stream (SISD)
2. Single Instruction stream, Multiple data stream (SIMD)
3. Multiple Instruction stream, Single data stream (MISD)
4. Multiple Instruction stream, Multiple data stream (MIMD)

A. Single Instruction Stream, Single Data Stream (SISD)

The Von Neumann Machines belong to this classification. SISD computers have one CPU that execute one instruction at a time & fetch or store one item of data at a time.

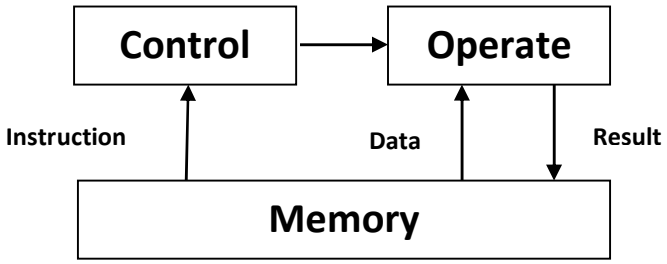


Figure 2. SISD Architecture

B. Single Instruction Stream, Multiple Data Stream (SIMD)

SIMD machine have a control unit (CU) that operate like VNM, it executes a single instruction stream. But have more than on Processing Element (PE). The CU generates the control signals for all PEs, which execute the same operation on different data items.

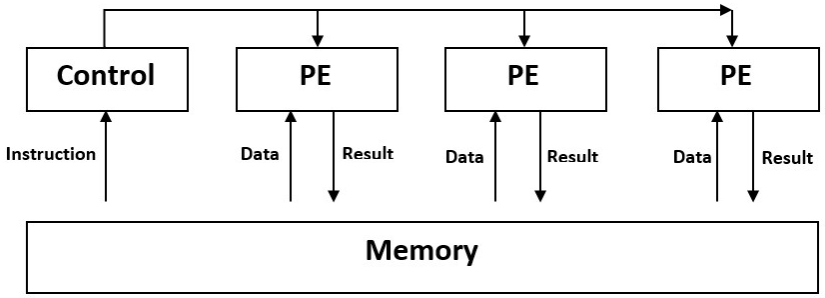


Figure 3. SIMD Architecture

C. Multiple Instruction Stream, Single Data Stream (MISD)

Logically machines in this class would execute several different programs on the same data item. There are currently no such machines.

D. Multiple Instruction Stream, Multiple Data Stream (MIMD)

MMD machines also called multiprocessors. They are more than one independent processor, and each processor can execute a different program on its own data.

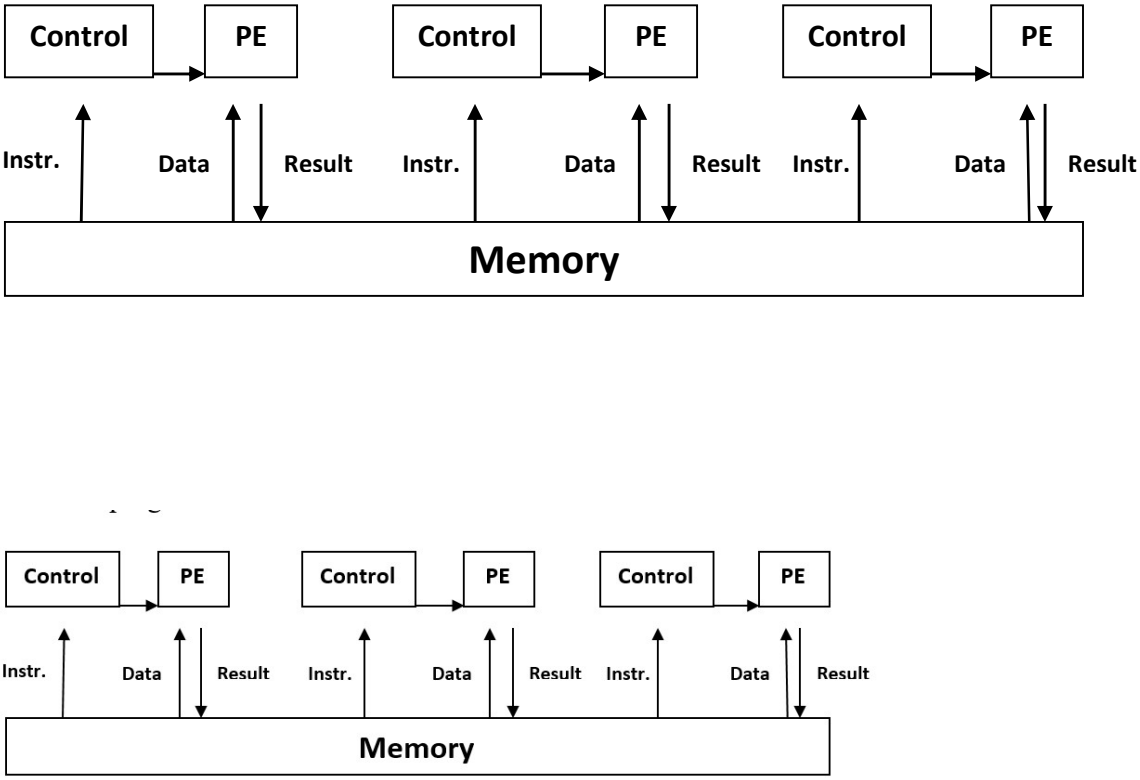


Figure 4. MIMD architecture

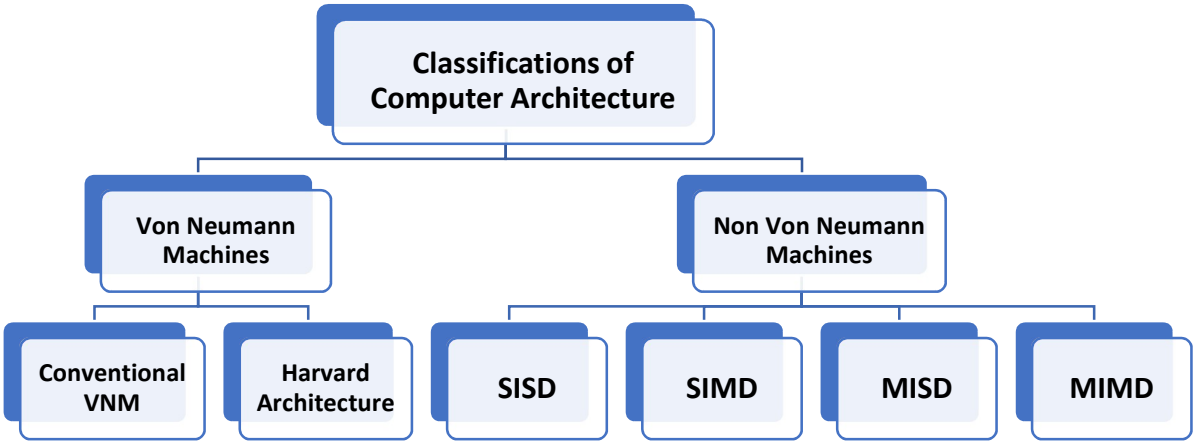


Figure 5. Classifications of Computer Architecture

MEMORY LOCATIONS AND OPERATIONS

The memory can be modeled as an array of millions of adjacent cells, each capable of storing a binary digit (bit), having value of 1 or 0. These cells are organized in the form of groups of fixed number of cells or bits. 8 bits is called a byte. size of a word ranges from 16 to 64 bits. It is, however, customary to express the size of the memory in terms of bytes. For example, the size of a typical memory of a personal computer is 256 Mbytes. In order to be able to move a word in and out of the memory, a distinct address has to be assigned to each word. This address will be used to determine the location in the memory in which a given word is to be stored. This is called a memory write operation. Similarly, the address will be used to determine the memory location from which a word is to be retrieved from the memory. This is called a memory read operation.

INSTRUCTION SET ARCHITECTURE AND DESIGN

Three basic steps are needed in order for the CPU to perform a write operation into a specified memory location:

1. The word to be stored into the memory location is first loaded by the CPU into a specified register, called the memory data register (MDR).

2. The address of the location into which the word is to be stored is loaded by the CPU into a specified register, called the memory address register (MAR).
3. A signal, called write, is issued by the CPU indicating that the word stored in the MDR is to be stored in the memory location whose address in loaded in the MAR.
Figure 2.2 illustrates the operation of writing the word given by 7E (in hex) into the memory location whose address is 2005. Part a of the figure shows the status of the registers and memory locations involved in the write operation before the execution of the operation. Part b of the figure shows the status after the execution of the operation.
It is worth mentioning that the MDR and the MAR are registers used exclusively by the CPU and are not accessible to the programmer.
Similar to the write operation, three basic steps are needed in order to perform a memory read operation:

1. The address of the location from which the word is to be read is loaded into the MAR.
2. A signal, called read, is issued by the CPU indicating that the word whose address is in the MAR is to be read into the MDR.
3. After some time, corresponding to the memory delay in reading the specified word, the required word will be loaded by the memory into the MDR ready for use by the CPU.

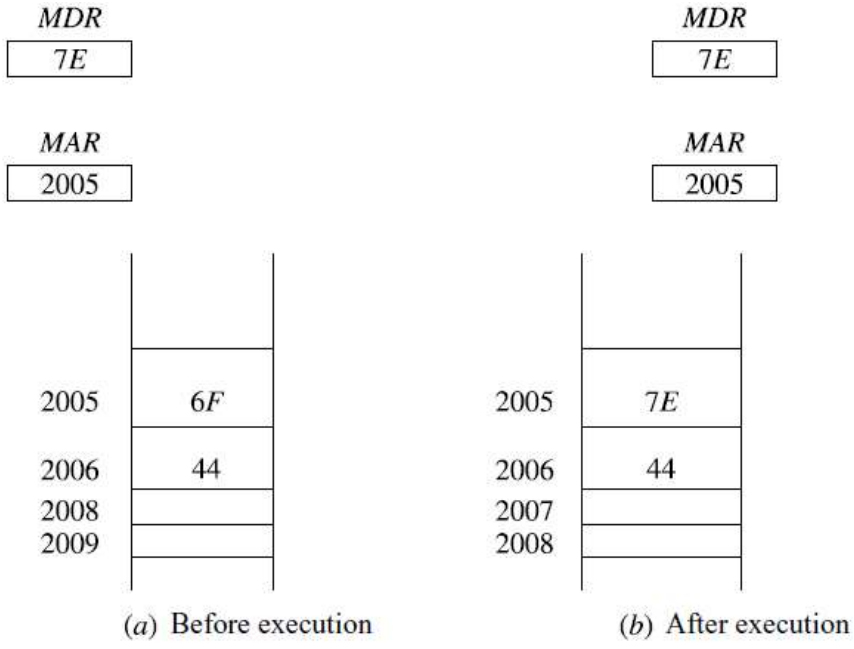


Figure 2.2 Illustration of the memory write operation

Figure 2.3 illustrates the operation of reading the word stored in the memory location whose address is 2010. Part a of the figure shows the status of the registers and memory locations involved in the read operation before the execution of the operation. Part b of the figure shows the status after the read operation.

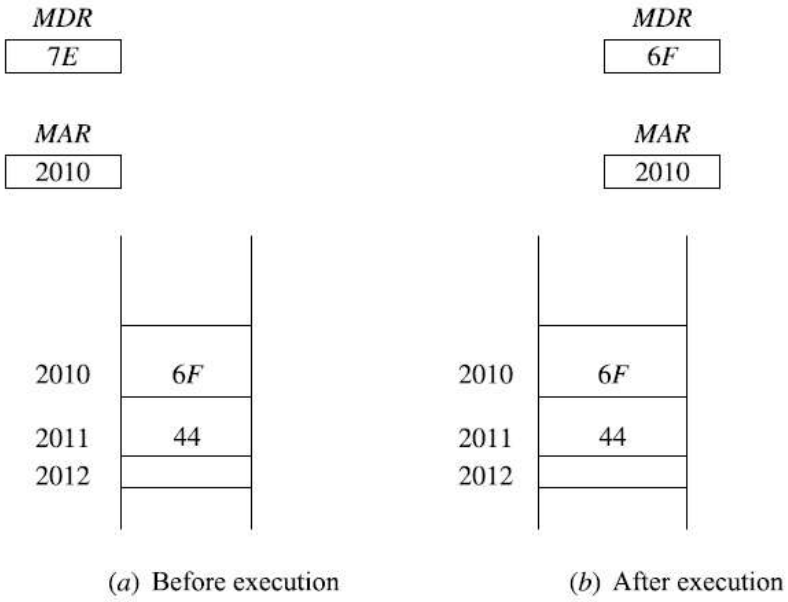


Figure 2.3 Illustration of the memory read operation

ADDRESSING MODES

Information involved in any operation performed by the CPU needs to be addressed. In computer terminology, such information is called the operand. Therefore, any instruction issued by the processor must carry at least two types of information.

These are the operation to be performed, encoded in what is called the op-code field, and the address information of the operand on which the operation is to be performed, encoded in what is called the address field.

Instructions can be classified based on the number of operands as: three-address, two-address, one-and-half-address, one-address, and zero-address. We explain these classes together with simple examples in the following paragraphs.

It should be noted that in presenting these examples, we would use the convention operation, source, destination to express any instruction. In that convention, operation represents the operation to be performed, for example, add, subtract, write, or read.

The source field represents the source operand(s). The source operand can be a constant, a value stored in a register, or a value stored in the memory. The destination field represents the place where the result of the operation is to be stored, for example, a register or a memory location.
for example a three-address instruction the instruction

ADD A,B,C

The instruction adds the contents of memory location A to the contents of memory location B and stores the result in memory location C.

A two-address instruction example:

ADD A,B

In this case, the contents of memory location A are added to the contents of memory location B and the result put in contents of memory location B.
A one-address instruction takes the form

ADD B

the instruction adds the content of the accumulator reg. to the content of memory location B and stores the result back into the accumulator .
Between the two- and the one-address instruction, there can be a one-and-half address instruction. Consider, for example, the instruction

ADD B,R1

In this case, the instruction adds the contents of register R1 to the contents of memory location B and stores the result in register R1.
It is interesting to indicate that there exist zero-address instructions. These are the instructions that use stack operation.

These are the push and the pop operations. Figure 2.4 illustrates these two operations. As can be seen, a specific register, called the stack pointer (SP), is used to indicate the stack location that can be addressed.

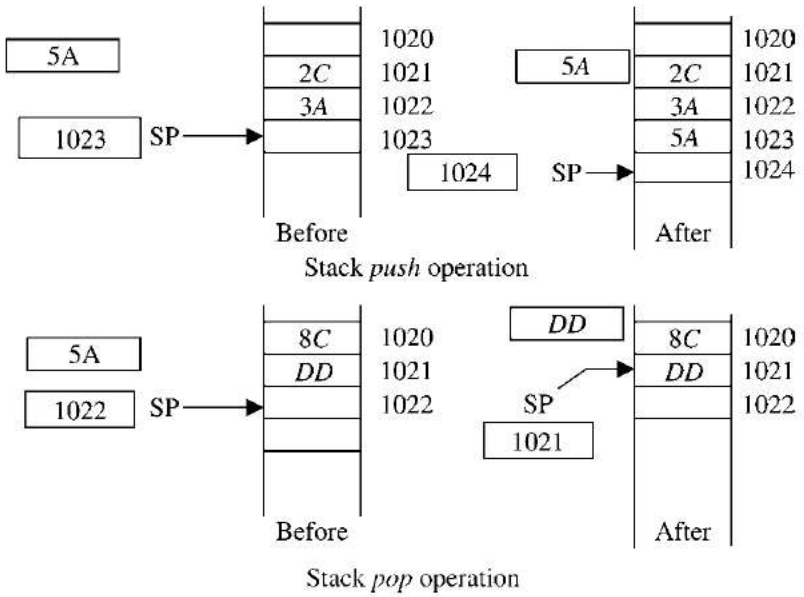


Figure 2.4 The stack push and pop operations

Processing Unit Design

we focus our attention on the main component of any computer system, the central processing unit (CPU). The primary function of the CPU is to execute a set of instructions stored in the computer's memory. A simple CPU consists of a set of registers, an arithmetic logic unit (ALU), and a control unit (CU). In what follows, the reader will be introduced to the organization and main operations of the CPU.

1. CPU BASICS

A typical CPU has three major components: (1) register set, (2) arithmetic logic unit (ALU), and (3) control unit (CU). The register set differs from one computer architecture to another. It is usually a combination of general-purpose and special purpose registers. General-purpose registers are used for any purpose, hence the name general purpose. Special-purpose registers have specific functions within the CPU. For example, the program counter (PC) is a special-purpose register that is used to hold the address of the instruction to be executed next. Another example of special-purpose registers is the instruction register (IR), which is used to hold the instruction that is currently executed. The ALU provides the circuitry needed to perform the arithmetic, logical and shift operations demanded of the instruction set. we have covered a number of arithmetic operations and circuits used to support computation in an ALU. The control unit is the entity responsible for fetching the instruction to be executed from the main memory and decoding and then executing it. Figure 5.1 shows the main components of the CPU and its interactions with the memory system and the input/ output devices.

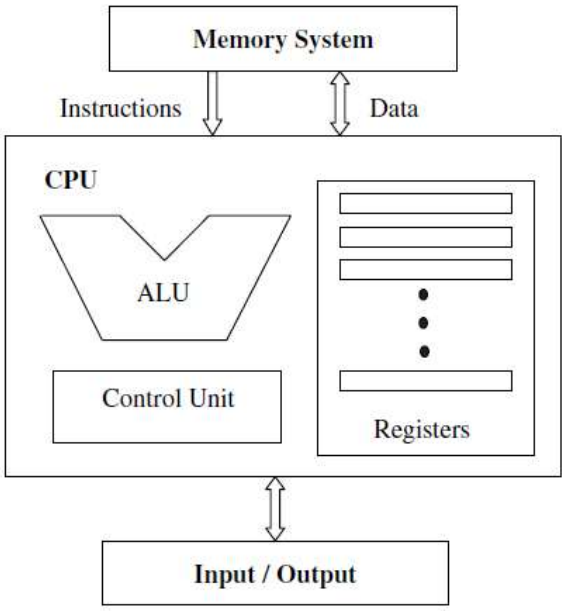


Figure 5.1 Central processing unit main components and interactions with the memory and I/O

The CPU fetches instructions from memory, reads and writes data from and to memory, and transfers data from and to input/output devices. A typical and simple execution cycle can be summarized as follows:

1. The next instruction to be executed, whose address is obtained from the PC, is fetched from the memory and stored in the IR.
2. The instruction is decoded.
3. Operands are fetched from the memory and stored in CPU registers, if needed.
4. The instruction is executed.
5. Results are transferred from CPU registers to the memory, if needed.

The execution cycle is repeated as long as there are more instructions to execute. A check for pending interrupts is usually included in the cycle. Examples of interrupts include I/O device request, arithmetic overflow, or a page fault.
When an interrupt request is encountered, a transfer to an interrupt handling routine takes place. Interrupt handling routines are programs that are invoked to collect the state of the currently executing program, correct the cause of the interrupt, and restore the state of the program.

The actions of the CPU during an execution cycle are defined by micro-orders issued by the control unit. These micro-orders are individual control signals sent over dedicated control lines. For example, let us assume that we want to execute an instruction that moves the contents of register X to register Y. Let us also assume that both registers are connected to the data bus, D. The control unit will issue a control signal to tell register X to place its contents on the data bus D. After some delay,
another control signal will be sent to tell register Y to read from data bus D. The activation of the control signals is determined using either hardwired control or microprogramming.

2. REGISTER SET

Registers are essentially extremely fast memory locations within the CPU that are used to create and store the results of CPU operations and other calculations. Different computers have different register sets. They differ in the number of registers, register types, and the length of each register. They also differ in the usage of each register. General-purpose registers can be used for multiple purposes and assigned to a variety of functions by the programmer. Special-purpose registers are restricted to only specific functions. In some cases, some registers are used only to hold data and cannot be used in the calculations of operand addresses. The length of a data register must be long enough to hold values of most data types. Some machines allow two contiguous registers to hold double-length values. Address registers may be dedicated to a particular addressing mode or may be used as address general purpose. Address registers must be long enough to hold the largest address. The number of registers in a particular architecture affects the instruction set design. A very small number of registers may result in an increase in memory references. Another type of registers is used to hold processor status bits, or flags. These bits are set by the CPU as the result of the execution of an operation. The status bits can be tested at a later time as part of another operation.

Memory Access Registers

Two registers are essential in memory write and read operations: the memory data register (MDR) and memory address register (MAR). The MDR and MAR are used exclusively by the CPU and are not directly accessible to programmers.
In order to perform a write operation into a specified memory location, the MDR and MAR are used as follows:

1. The word to be stored into the memory location is first loaded by the CPU into MDR.

2. The address of the location into which the word is to be stored is loaded by the CPU into a MAR.
3. A write signal is issued by the CPU.

Similarly, to perform a memory read operation, the MDR and MAR are used as follows:
1. The address of the location from which the word is to be read is loaded into the MAR.
2. A read signal is issued by the CPU.
3. The required word will be loaded by the memory into the MDR ready for use by the CPU.

DIRECT MEMORY ACCESS (DMA)

The main idea of direct memory access (DMA) is to enable peripheral devices to cut out the "middle man" role of the CPU in data transfer. It allows peripheral devices to transfer data directly from and to memory without the intervention of the CPU. Having peripheral devices access memory directly would allow the CPU to do other work, which would lead to improved performance, especially in the cases of large transfers.

The DMA controller is a piece of hardware that controls one or more peripheral devices. It allows devices to transfer data to or from the system's memory without the help of the processor. In a typical DMA transfer, some event notifies the DMA controller that data needs to be transferred to or from memory. Both the DMA and CPU use memory bus and only one or the other can use the memory at the same time. The DMA controller then sends a request to the CPU asking its permission to use the bus. The CPU returns an acknowledgment to the DMA controller granting it bus access. The DMA can now take control of the bus to independently conduct memory transfer. When the transfer is complete the DMA relinquishes its control of the bus to the CPU. Processors that support DMA provide one or more input signals that the bus requester can assert to gain control of the bus and one or more output signals that the CPU asserts to indicate it has relinquished the bus. Figure 8.10 shows how the DMA controller shares the CPU's memory bus.

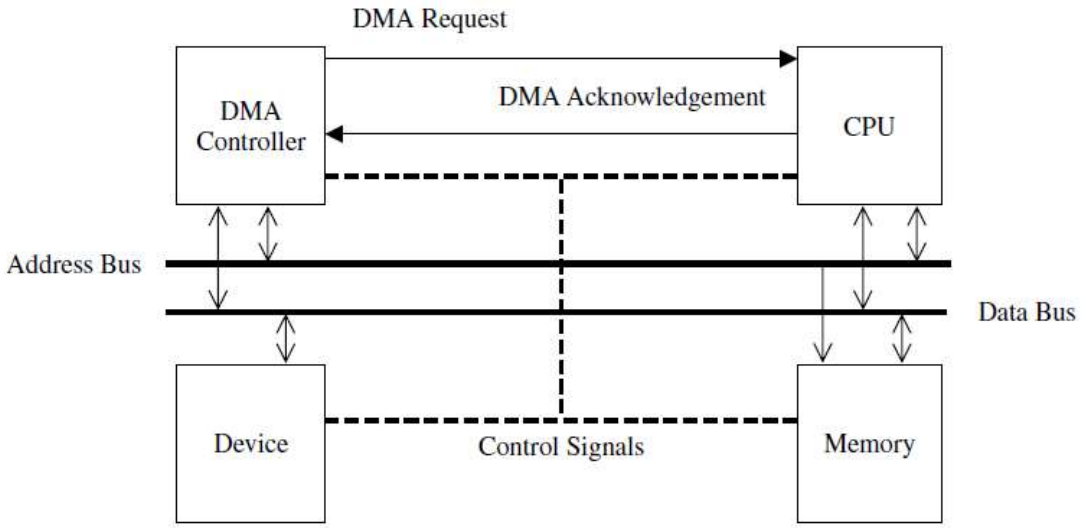


Figure 8.10 DMA controller shares the CPU's memory bus

Direct memory access controllers require initialization by the CPU. Typical setup parameters include the address of the source area, the address of the destination area, the length of the block, and whether the DMA controller should generate a processor interrupt once the block transfer is complete. A DMA controller has an address register, a word count register, and a control register. The address register contains an address that specifies the memory location of the data to be transferred. It is typically possible to have the DMA controller automatically increment the address register after each word transfer, so that the next transfer will be from the next memory location. The word count register holds the number of words to be transferred.

The word count is decremented by one after each word transfer. The control register specifies the transfer mode.

Direct memory access data transfer can be performed in burst mode or single cycle mode. In burst mode, the DMA controller keeps control of the bus until all the data has been transferred to (from) memory from (to) the peripheral device.

This mode of transfer is needed for fast devices where data transfer cannot be stopped until the entire transfer is done. In single-cycle mode (cycle stealing), the DMA controller relinquishes the bus after each transfer of one data word. This minimizes the amount of time that the DMA controller keeps the CPU from controlling the bus, but it requires that the bus request/acknowledge sequence be performed for every single transfer. This overhead can result in a degradation of the performance.

The single-cycle mode is preferred if the system cannot tolerate more than a few cycles of added interrupt latency or if the peripheral devices can buffer very large

amounts of data, causing the DMA controller to tie up the bus for an excessive amount of time.

The following steps summarize the DMA operations:
1. DMA controller initiates data transfer.
2. Data is moved (increasing the address in memory, and reducing the count of words to be moved).
3. When word count reaches zero, the DMA informs the CPU of the termination by means of an interrupt.
4. The CPU regains access to the memory bus.

A DMA controller may have multiple channels. Each channel has associated with it an address register and a count register. To initiate a data transfer the device driver sets up the DMA channel's address and count registers together with the direction of the data transfer, read or write. While the transfer is taking place, the CPU is free to do other things. When the transfer is complete, the CPU is interrupted.

Direct memory access channels cannot be shared between device drivers. A device driver must be able to determine which DMA channel to use. Some devices have a fixed DMA channel, while others are more flexible, where the device driver can simply pick a free DMA channel to use.

Linux tracks the usage of the DMA channels using a vector of dma_chan data structures (one per DMA channel). The dma_chan data structure contains just two fields, a pointer to a string describing the owner of the DMA channel and a flag indicating if the DMA channel is allocated or not.