OPERATING SYSTEM: THE BASICS

Patrick McClanahan
San Joaquin Delta College

LibreTexts™

TABLE OF CONTENTS

CHAPTER OVERVIEW

1: The Basics - An Overview

This page titled 1: The Basics - An Overview is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.1: Introduction to Operating Systems

Introduction to Operating System

An operating system acts as an intermediary between the user of a computer and computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.

An operating system is a software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

Operating System – Definition:

- An operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.
- A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all else being application programs.
- An operating system is concerned with the allocation of resources and services, such as memory, processors, devices, and information. The operating system correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

Functions of Operating system – Operating system performs three functions:

1. **Convenience:** An OS makes a computer more convenient to use.
2. **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
3. **Ability to Evolve:** An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions at the same time without interfering with service.

Operating system as User Interface –

1. User
2. System and application programs
3. Operating system
4. Hardware

Every general-purpose computer consists of the hardware, operating system, system programs, and application programs. The hardware consists of memory, CPU, ALU, and I/O devices, peripheral device, and storage device. System program consists of compilers, loaders, editors, OS, etc. The application program consists of business programs, database programs.

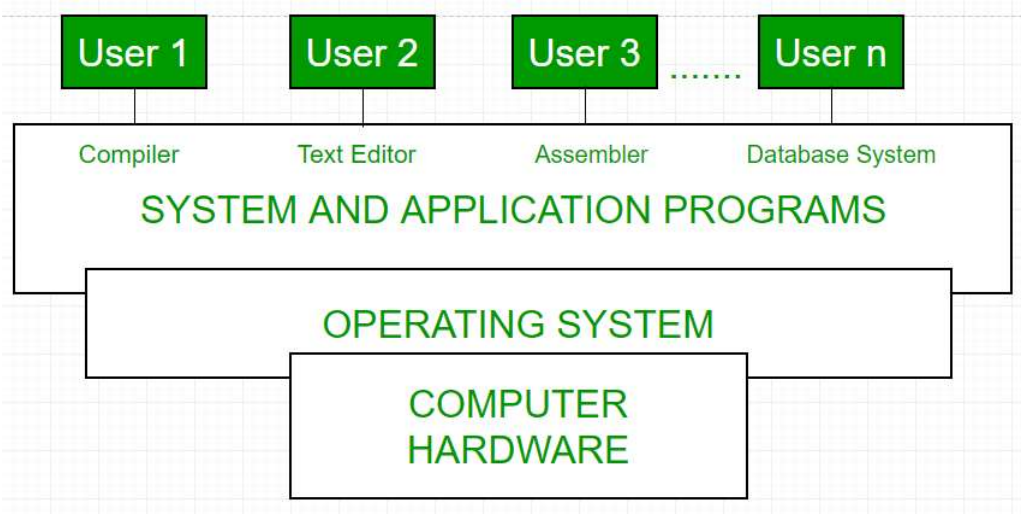


Figure 1.1.1: Conceptual view of a computer system ("Conceptual view of a computer system" by Unknown, Geeks for Geeks is licensed under CC BY-SA 4.0)

Every computer must have an operating system to run other programs. The operating system coordinates the use of the hardware among the various system programs and application programs for various users. It simply provides an environment within which other programs can do useful work.

The operating system is a set of special programs that run on a computer system that allows it to work properly. It performs basic tasks such as recognizing input from the keyboard, keeping track of files and directories on the disk, sending output to the display screen and controlling peripheral devices.
OS is designed to serve two basic purposes:

1. It controls the allocation and use of the computing System's resources among the various user and tasks.
2. It provides an interface between the computer hardware and the programmer that simplifies and makes feasible for coding, creation, debugging of application programs.

The Operating system must support the following tasks. The task are:

1. Provides the facilities to create, modification of programs and data files using an editor.
2. Access to the compiler for translating the user program from high level language to machine language.
3. Provide a loader program to move the compiled program code to the computer's memory for execution.
4. Provide routines that handle the details of I/O programming.

I/O System Management –
The module that keeps track of the status of devices is called the I/O traffic controller. Each I/O device has a device handler that resides in a separate process associated with that device.
The I/O subsystem consists of

- A memory Management component that includes buffering caching and spooling.
- A general device driver interface.

Drivers for specific hardware devices.

Assembler – The input to an assembler is an assembly language program. The output is an object program plus information that enables the loader to prepare the object program for execution. At one time, the computer programmer had at their disposal a basic machine that interpreted, through hardware, certain fundamental instructions. The programmer would program the computer by writing a series of ones and zeros (machine language), and place them into the memory of the machine.

Compiler/Interpreter – The high-level languages - for example C/C++, are processed by compilers and interpreters. A compiler is a program that accepts source code written in a "high-level language "and produces a corresponding object program. An interpreter is a program that directly executes a source program as if it was machine language.

Loader – A loader is a routine that loads an object program and prepares it for execution. There are various loading schemes: absolute, relocating and direct-linking. In general, the loader must load, relocate and link the object program. The loader is a program that places programs into memory and prepares them for execution. In a simple loading scheme, the assembler outputs the machine language translation of a program on a secondary device and a loader places it in the core. The loader places into memory the machine language version of the user's program and transfers control to it. Since the loader program is much smaller than the assembler, those make more core available to the user's program.

History of Operating system –
Operating system has been evolving through the years. Following Table shows the history of OS.

Generation	Year	Electronic device used	Types of OS Device
First	1945-55	Vacuum Tubes	Plug Boards
Second	1955-65	Transistors	Batch Systems
Third	1965-80	Integrated Circuits(IC)	Multiprogramming
Fourth	Since 1980	Large Scale Integration	PC

Adapted from:
"Introduction of Operating System – Set 1" by Unknown, Geeks for Geeks is licensed under CC BY-SA 4.0

This page titled 1.1: Introduction to Operating Systems is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.2 Starting with the Basics

Processor

The processor is an important part of a computer architecture, without it nothing would happen. It is a programmable device that takes input, perform some arithmetic and logical operations and produce some output. In simple words, a processor is a digital device on a chip which can fetch instruction from memory, decode and execute them and provide results.

Basics of a Processor –
A processor takes a bunch of instructions in machine language and executes them, telling the processor what it has to do. Processors performs three basic operations while executing the instruction:

1. It performs some basic operations like addition, subtraction, multiplication, division and some logical operations using its Arithmetic and Logical Unit (ALU).
2. Data in the processor can move from one location to another.
3. It has a Program Counter (PC) register that stores the address of next instruction based on the value of PC.

A typical processor structure looks like this.

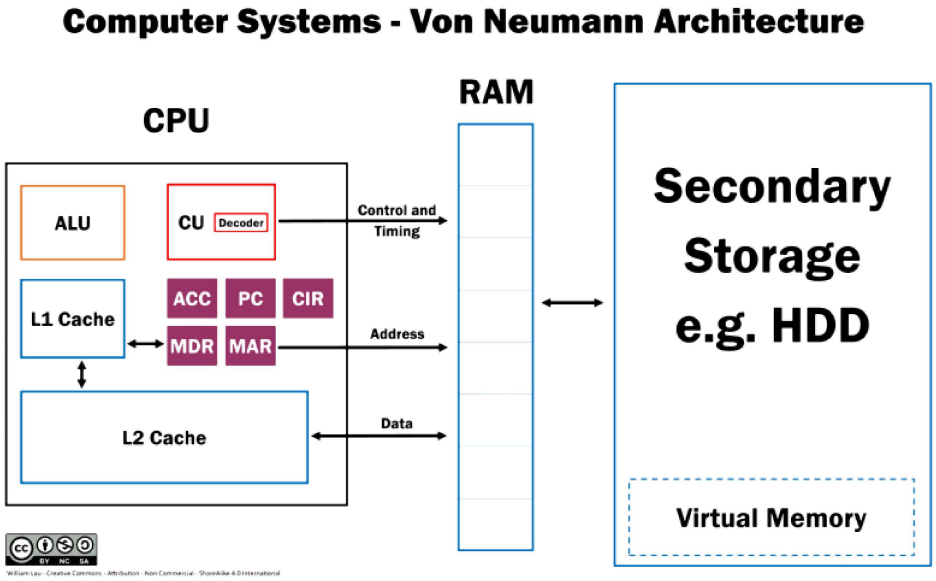


Figure 1: Von Neumann Architecture. ("File:Computer Systems - Von Neumann Architecture Large poster anchor chart.svg" by BotMultichillT, Wikimedia Commons is licensed under CC BY-NC-SA 4.0)

Basic Processor Terminology

- **Control Unit (CU)**
  A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches the code for instructions and controlling how data moves around the system.
- **Arithmetic and Logic Unit (ALU)**
  The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need, e.g. Addition, Subtraction, Comparisons. It performs Logical Operations, Bit Shifting Operations, and Arithmetic Operation.
- **Main Memory Unit (Registers)**
  1. **Accumulator (ACC):** Stores the results of calculations made by ALU.
  2. **Program Counter (PC):** Keeps track of the memory location of the next instructions to be dealt with. The PC then passes this next address to Memory Address Register (MAR).
  3. **Memory Address Register (MAR):** It stores the memory locations of instructions that need to be fetched from memory or stored into memory.

4. **Memory Data Register (MDR):** It stores instructions fetched from memory or any data that is to be transferred to, and stored in, memory.
5. **Current Instruction Register (CIR):** It stores the most recently fetched instructions while it is waiting to be coded and executed.
6. **Instruction Buffer Register (IBR):** The instruction that is not to be executed immediately is placed in the instruction buffer register IBR.

- **Input/Output Devices –** Program or data is read into main memory from the *input device* or secondary storage under the control of CPU input instruction. *Output devices* are used to output the information from a computer.
- **Buses –** Data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory, by the means of Buses. Types:
  1. **Data Bus (Data):** It carries data among the memory unit, the I/O devices, and the processor.
  2. **Address Bus (Address):** It carries the address of data (not the actual data) between memory and processor.
  3. **Control Bus (Control and Timing):** It carries control commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer.

Memory

Memory attached to the CPU is used for storage of data and instructions and is called internal memory The internal memory is divided into many storage locations, each of which can store data or instructions. Each memory location is of the same size and has an address. With the help of the address, the computer can read any memory location easily without having to search the entire memory. when a program is executed, it's data is copied to the internal memory and is stored in the memory till the end of the execution. The internal memory is also called the Primary memory or Main memory. This memory is also called as RAM, i.e. Random Access Memory. The time of access of data is independent of its location in memory, therefore this memory is also called Random Access memory (RAM).

I/O Modules

The method that is used to transfer information between main memoryand external I/O devices is known as the I/O interface, or I/O modules. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. **Programmed I/O:** is the result of the I/O instructions written in the program's code. Each data transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and/or memory. In this case it requires constant monitoring by the CPU of the peripheral devices.
2. **Interrupt- initiated I/O:** using an interrupt facility and special commands to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed processing other programs. The interface meanwhile keeps monitoring the device. When it is determined that the device is ready for a data transfer it initiates an interrupt request signal to the CPU. Upon detection of an external interrupt signal the CPU momentarily stops the task it was processing, and services program that was waiting on the interrupt to process the I/O transfer> Once the interrupt is satisfied, the CPU then return to the task it was originally processing.
3. **Direct memory access(DMA)**: The data transfer between a fast storage media such as magnetic disk and main memory is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as direct memory access, or DMA. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

Adapted from:
"Introduction of Microprocessor" by DikshaTewari, **Geeks for Geeks** is licensed under CC BY-SA 4.0
"Last Minute Notes Computer Organization" by Geeks for Geeks is licensed under CC BY-SA 4.0
"Functional Components of a Computer" by aishwaryaagarwal2, Geeks for Geeks is licensed under CC BY-SA 4.0
"System Bus Design" by deepak, Geeks for Geeks is licensed under CC BY-SA 4.0
"I/O Interface (Interrupt and DMA Mode)" by Unknown, Geeks for Geeks is licensed under CC BY-SA 4.0

This page titled 1.2 Starting with the Basics is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.3 The Processor - History

Microprocessor Evolution

Generations of microprocessor:

1. **First generation:** From 1971 to 1972 the era of the first generation came which brought microprocessors like INTEL 4004 Rockwell international PPS-4 INTEL 8008 etc.
2. **Second generation:** The second generation marked the development of 8 bit microprocessors from 1973 to 1978. Processors like INTEL 8085 Motorola 6800 and 6801 etc came into existence.
3. **Third generation:** The third generation brought forward the 16 bit processors like INTEL 8086/80186/80286 Motorola 68000 68010 etc. From 1979 to 1980 this generation used the HMOS technology.
4. **Fourth generation:** The fourth generation came into existence from 1981 to 1995. The 32 bit processors using HMOS fabrication came into existence. INTEL 80386 and Motorola 68020 are some of the popular processors of this generation.
5. **Fifth generation:** From 1995 till now we are in the fifth generation. 64 bit processors like Pentium, Celeron, dual, quad and octa core processors came into existence.

Types of microprocessors:

- **Complex instruction set microprocessor:** The processors are designed to minimize the number of instructions per program and ignore the number of cycles per instructions. The compiler is used to translate a high level language to assembly level language because the length of code is relatively short and an extra RAM is used to store the instructions. These processors can do tasks like downloading, uploading and recalling data from memory. Apart from these tasks these microprocessor can perform complex mathematical calculation in a single command.
  Example: IBM 370/168, VAX 11/780
- **Reduced instruction set microprocessor:** These processor are made according to function. They are designed to reduce the execution time by using the simplified instruction set. They can carry out small things in specific commands. These processors complete commands at faster rate. They require only one clock cycle to implement a result at uniform execution time. There are number of registers and less number of transistors. To access the memory location LOAD and STORE instructions are used.
  Example: Power PC 601, 604, 615, 620
- **Super scalar microprocessor:** These processors can perform many tasks at a time. They can be used for ALUs and multiplier like array. They have multiple operation unit and perform fast by executing multiple commands.
- **Application specific integrated circuit:** These processors are application specific like for personal digital assistant computers. They are designed according to proper specification.
- **Digital signal multiprocessor:** These processors are used to convert signals like analog to digital or digital to analog. The chips of these processors are used in many devices such as RADAR SONAR home theaters etc.

Advantages of microprocessor

1. High processing speed
2. Compact size
3. Easy maintenance
4. Can perform complex mathematics
5. Flexible
6. Can be improved according to requirement

Disadvantages of microprocessors

1. Overheating occurs due to overuse
2. Performance depends on size of data
3. Large board size than microcontrollers
4. Most microprocessors do not support floating point operations

Adapted from:
"Evolution of Microprocessors" by Ayusharma0698, Geeks for Geeks is licensed under CC BY-SA 4.0

This page titled 1.3 The Processor - History is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.3.1: The Processor - Components

Accumulator

In a computer's central processing unit (CPU), the accumulator (ACC in the image below) is a register in which intermediate arithmetic and logic results are stored.

Without a register like an accumulator, it would be necessary to write the result of each calculation (addition, multiplication, shift, etc.) to main memory, perhaps only to be read right back again for use in the next operation.

Access to main memory is slower than access to a register like an accumulator because the technology used for the large main memory is slower (but cheaper) than that used for a register. Early electronic computer systems were often split into two groups, those with accumulators and those without.

Modern computer systems often have multiple general-purpose registers that can operate as accumulators, and the term is no longer as common as it once was. However, to simplify their design, a number of special-purpose processors still use a single accumulator.

Arithmetic logic unit

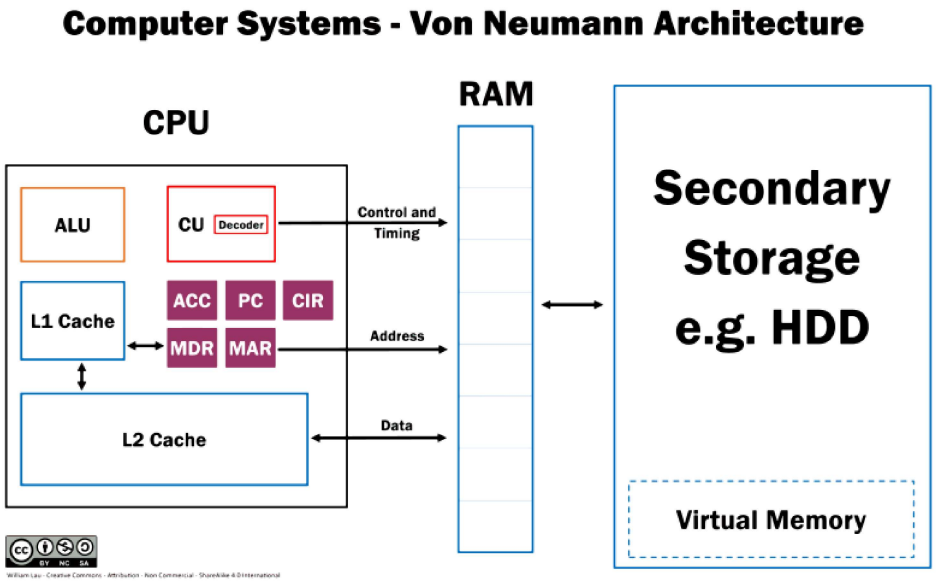
The arithmetic logic unit (ALU) performs the arithmetic and logical functions that are the work of the computer. There are other general purpose registers that hold the input data, and the accumulator receives the result of the operation. The instruction register contains the instruction that the ALU is to perform.

For example, when adding two numbers, one number is placed in one of the general purpose registers register and the other in another general purpose register. The ALU performs the addition and puts the result in the accumulator. If the operation is a logical one, the data to be compared is placed into the one of the general purpose registers. The result of the comparison, a 1 or 0, is put in the accumulator. Whether this is a logical or arithmetic operation, the accumulator content is then placed into the cache location reserved by the program for the result.

Instruction register and pointer

The instruction pointer (CIR in the image below) specifies the location in memory containing the next instruction to be executed by the CPU. When the CPU completes the execution of the current instruction, the next instruction is loaded into the instruction register from the memory location pointed to by the instruction pointer.

After the instruction is loaded into the instruction register, the instruction register pointer is incremented by one instruction address. Incrementing allows it to be ready to move the next instruction into the instruction register.



Memory Address Register

In a computer, the memory address register (MAR) is the CPU register that either stores the memory address from which data will be fetched to the CPU, or the address to which data will be sent and stored.

In other words, This register is used to access data and instructions from memory during the execution phase of instruction. MAR holds the memory location of data that needs to be accessed. When reading from memory, data addressed by MAR is fed into the MDR (memory data register) and then used by the CPU. When writing to memory, the CPU writes data from MDR to the memory location whose address is stored in MAR. MAR, which is found inside the CPU, goes either to the RAM (random access memory) or cache.

Memory Data Register

The memory data register (MDR) is the register that stores the data being transferred to and from the immediate access storage. It contains the copy of designated memory locations specified by the memory address register. It acts as a buffer allowing the processor and memory units to act independently without being affected by minor differences in operation. A data item will be copied to the MDR ready for use at the next clock cycle, when it can be either used by the processor for reading or writing or stored in main memory after being written.

This register holds the contents of the memory which are to be transferred from memory to other components or vice versa. A word to be stored must be transferred to the MDR, from where it goes to the specific memory location, and the arithmetic data to be processed in the ALU first goes to MDR and then to accumulated register, and then it is processed in the ALU.

The MDR is a two-way register. When data is fetched from memory and placed into the MDR, it is written to go in one direction. When there is a write instruction, the data to be written is placed into the MDR from another CPU register, which then puts the data into memory.

Cache

The CPU never directly accesses RAM. Modern CPUs have one or more layers of cache. The CPU's ability to perform calculations is much faster than the RAM's ability to feed data to the CPU.

Cache memory is faster than the system RAM, and it is closer to the CPU because it is physically located on the processor chip. The cache provides data storage and instructions to prevent the CPU from waiting for data to be retrieved from RAM. When the CPU needs data - program instructions are also considered to be data - the cache determines whether the data is already in residence and provides it to the CPU.

If the requested data is not in the cache, it's retrieved from RAM and uses predictive algorithms to move more data from RAM into the cache. The cache controller analyzes the requested data and tries to predict what additional data will be needed from RAM. It loads the anticipated data into the cache. By keeping some data closer to the CPU in a cache that is faster than RAM, the CPU can remain busy and not waste cycles waiting for data.

The simple example CPU has two levels of cache. Levels 2 is designed to predict what data and program instructions will be needed next, move that data from RAM, and move it ever closer to the CPU to be ready when needed. These cache sizes typically range from 1 MB to 32 MB, depending upon the speed and intended use of the processor.

CPU clock and control unit (CU in the image)

All of the CPU components must be synchronized to work together smoothly. The control unit performs this function at a rate determined by the clock speed and is responsible for directing the operations of the other units by using timing signals that extend throughout the CPU.

Random access memory (RAM)

Although the RAM, or main storage, is shown in this diagram and the next, it is not truly a part of the CPU. Its function is to store programs and data so that they are ready for use when the CPU needs them.

Adapted from:
"Accumulator (computing)" by Multiple Contributors, Wikipedia is licensed under CC BY-SA 3.0
"Memory buffer register" by Multiple Contributors, Wikipedia is licensed under CC BY-SA 3.0
"Memory address register" by Multiple Contributors, Wikipedia is licensed under CC BY-SA 3.0
William Lau, CC BY-SA 4.0, via Wikimedia Commons

1.3.1: The Processor - Components is shared under a not declared license and was authored, remixed, and/or curated by LibreTexts.

1.3.2: The Processor - Bus

Control Bus

In computer architecture, a control bus is part of the system bus, used by CPUs for communicating with other devices within the computer. While the address bus carries the information about the device with which the CPU is communicating and the data bus carries the actual data being processed, the control bus carries commands from the CPU and returns status signals from the devices. For example, if the data is being read or written to the device the appropriate line (read or write) will be active (logic one).

Address bus

An address bus is a bus that is used to specify a physical address. When a processor or DMA-enabled device needs to read or write to a memory location, it specifies that memory location on the address bus (the value to be read or written is sent on the data bus). The width of the address bus determines the amount of memory a system can address. For example, a system with a 32-bit address bus can address 232 (4,294,967,296) memory locations. If each memory location holds one byte, the addressable memory space is 4 GiB.

Data / Memory Bus

The memory bus is the computer bus which connects the main memory to the memory controller in computer systems. Originally, general-purpose buses like VMEbus and the S-100 bus were used, but to reduce latency, modern memory buses are designed to connect directly to DRAM chips, and thus are designed by chip standards bodies such as JEDEC. Examples are the various generations of SDRAM, and serial point-to-point buses like SLDRAM and RDRAM. An exception is the Fully Buffered DIMM which, despite being carefully designed to minimize the effect, has been criticized for its higher latency.

Adapted from:
"Bus (computing)" by Multiple Contributors, Wikipedia is licensed under CC BY-SA 3.0
"Memory bus" by Multiple Contributors, Wikipedia is licensed under CC BY-SA 3.0
"Control bus" by Multiple Contributors, Wikipedia is licensed under CC BY-SA 3.0

1.3.2: The Processor - Bus is shared under a not declared license and was authored, remixed, and/or curated by LibreTexts.

1.4 Instruction Cycles

Instruction Cycles

The **instruction cycle** (also known as the **fetch–decode–execute cycle**, or simply the **fetch-execute cycle**) is the cycle that the central processing unit (CPU) follows from boot-up until the computer has shut down in order to process instructions. It is composed of three main stages: the fetch stage, the decode stage, and the execute stage.

Role of components

The program counter (PC) is a special register that holds the memory address of the next instruction to be executed. During the fetch stage, the address stored in the PC is copied into the memory address register (MAR) and then the PC is incremented in order to "point" to the memory address of the next instruction to be executed. The CPU then takes the instruction at the memory address described by the MAR and copies it into the memory data register (MDR). The MDR also acts as a two-way register that holds data fetched from memory or data waiting to be stored in memory (it is also known as the memory buffer register (MBR) because of this). Eventually, the instruction in the MDR is copied into the current instruction register (CIR) which acts as a temporary holding ground for the instruction that has just been fetched from memory.

During the decode stage, the control unit (CU) will decode the instruction in the CIR. The CU then sends signals to other components within the CPU, such as the arithmetic logic unit (ALU) and the floating point unit (FPU). The ALU performs arithmetic operations such as addition and subtraction and also multiplication via repeated addition and division via repeated subtraction. It also performs logic operations such as AND, OR, NOT, and binary shifts as well. The FPU is reserved for performing floating-point operations.

Summary of stages

Each computer's CPU can have different cycles based on different instruction sets, but will be similar to the following cycle:

1. **Fetch Stage**: The next instruction is fetched from the memory address that is currently stored in the program counter and stored into the instruction register. At the end of the fetch operation, the PC points to the next instruction that will be read at the next cycle.
2. **Decode Stage**: During this stage, the encoded instruction presented in the instruction register is interpreted by the decoder.
    - **Read the effective address**: In the case of a memory instruction (direct or indirect), the execution phase will be during the next clock pulse. If the instruction has an indirect address, the effective address is read from main memory, and any required data is fetched from main memory to be processed and then placed into data registers (clock pulse: T_3). If the instruction is direct, nothing is done during this clock pulse. If this is an I/O instruction or a register instruction, the operation is performed during the clock pulse.
3. **Execute Stage**: The control unit of the CPU passes the decoded information as a sequence of control signals to the relevant functional units of the CPU to perform the actions required by the instruction, such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them, and writing the result back to a register. If the ALU is involved, it sends a condition signal back to the CU. The result generated by the operation is stored in the main memory or sent to an output device. Based on the feedback from the ALU, the PC may be updated to a different address from which the next instruction will be fetched.
4. **Repeat Cycle**

Registers Involved In Each Instruction Cycle:

- **Memory address registers(MAR)** : It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
- **Memory Buffer Register(MBR)** : It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.
- **Program Counter(PC)** : Holds the address of the next instruction to be fetched.
- **Instruction Register(IR)** : Holds the last instruction fetched.

Adapted from:
"Instruction cycle" by Multiple Contributors, Wikipedia is licensed under CC BY-NC 3.0

This page titled 1.4 Instruction Cycles is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.4.1 Instruction Cycles - Fetch

The Fetch Cycle

At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter*(PC).

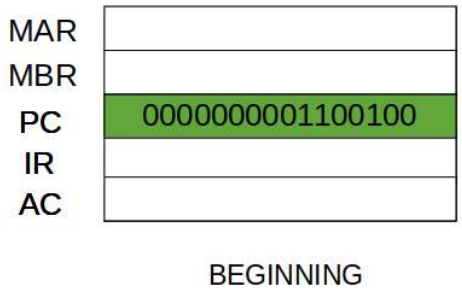


Figure 1: Beginning of the Fetch Cycle. ("Beginning of the Fetch Cycle" by Astha_Singh, Geeks for Geeks is licensed under CC BY-SA 4.0)

Step 1: The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.

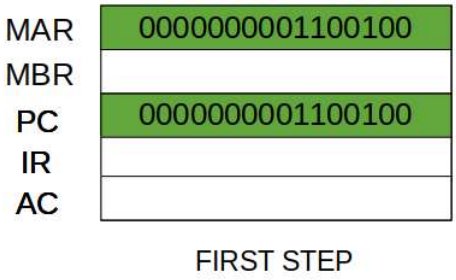


Figure 1: Step #1 of Fetch Cycle. ("Beginning of the Fetch Cycle" by Astha_Singh, Geeks for Geeks is licensed under CC BY-SA 4.0)

Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction.(These two action can be performed simultaneously to save time)

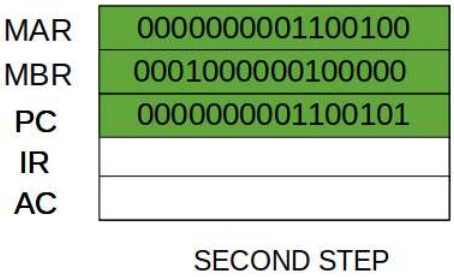


Figure 1: Step #2 of Fetch Cycle. ("Step #2 of the Fetch Cycle" by Astha_Singh, Geeks for Geeks is licensed under CC BY-SA 4.0)

Step 3: The content of the MBR is moved to the instruction register(IR).

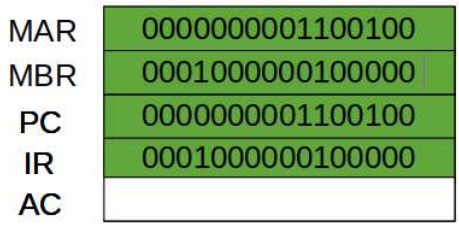


Figure 1: Step #3 of Fetch Cycle. ("Step #3 of the Fetch Cycle" by Astha_Singh, Geeks for Geeks is licensed under CC BY-SA 4.0)

Thus, a simple *Fetch Cycle* consist of three steps and four micro-operation. Symbolically, we can write these sequence of events as follows:-

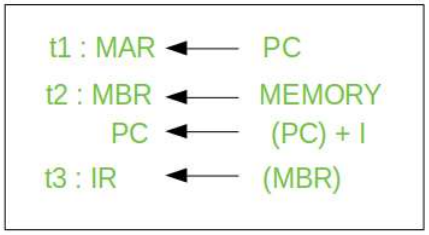


Figure 1: Fetch Cycle Steps. ("Fetch Cycle Steps" by Astha_Singh, Geeks for Geeks is licensed under CC BY-SA 4.0)

Here 'I' is the instruction length. The notations (t1, t2, t3) represents successive time units. We assume that a clock is available for timing purposes and it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit.

First time unit: Move the contents of the PC to MAR. The contents of the Program Counter contains the address (location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence. When the computer restarts or is reset, the program counter normally reverts to 0. The MAR stores this address.

Second time unit: Move contents of memory location specified by MAR to MBR. Remember - the MDB contains the value to be stored in memory or the last value read from the memory, in this example it is an instruction to be executed. Also in this time unit the PC content gets incremented by 1.

Third time unit: Move contents of MBR to IR. Now the instructin register contains the instruction we need to execute.
Note: Second and third micro-operations both take place during the second time unit.

Adapted from:
"Computer Organization | Different Instruction Cycles" by Astha_Singh, Geeks for Geeks is licensed under CC BY-SA 4.0

This page titled 1.4.1 Instruction Cycles - Fetch is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.4.2 Instruction Cycles - Instruction Primer

How Instructions Work

On the previous page we showed how the fetched instruction is loaded into the instruction register (IR). The instruction contains bits that specify the action the processor is to take. The processor will read the instruction and performs the necessary action. In general, these actions fall into four categories:

- **Processor - memory:** Data may be transferred from processor to memory, or from memory to processor.
- **Processor - I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and the system's I/O module.
- **Data processing:** The processor may perform some arithmetic or logic operation on data, such as addition or comparisons.
- **Control:** An instruction may specify that the sequence of execution be altered. For example, the processor may fetch an instruction from location 149, which specifies that the next instruction be from location 182. The processor sets the program counter to 182. Thus, on the next fetch stage, the instruction will be fetched from location 182 rather than 150.

It is not the intent of this course to cover assembly language programming, or the concepts taught there. However, in order to provide an example the following details are provided.

In this simplistic example, both the instructions and data are 16 bits long. The instructions have to following format:

Figure 1: Instruction Layout. ("Instruction Layout" by Patrick McClanahan is licensed under CC BY-SA 4.0)

The opcode is a numerical representation of the instruction to be performed, instructions such as mathematical or logic operations. The opcode tells the processor what it needs to do. The second part of the instructions tells the processor WHERE to find the data that is being operated on. (we will see more specifically how this works in a moment).

Figure 2: Data Layout. ("Data Layout" by Patrick McClanahan is licensed under CC BY-SA 4.0)

When reading data from a memory location the first bit is a sign bit, and the other 15 bits are for the actual data.

In our example, we include an Accumulator (AC) which will be used as a temporary storage location for the data.

There will be 3 opcodes - PLEASE understand these are sample opcode for this example...do NOT confuse these with actual processor opcodes.

1. 0001 - this opcode tells the processor to load the accumulator (AC) from the given memory address.
2. 0011 - this opcode tells the processor to add to the value currently stored in the AC from the specified memory address.
3. 0111 - this opcode tells the processor to move the value in the AC to the specified memory address.

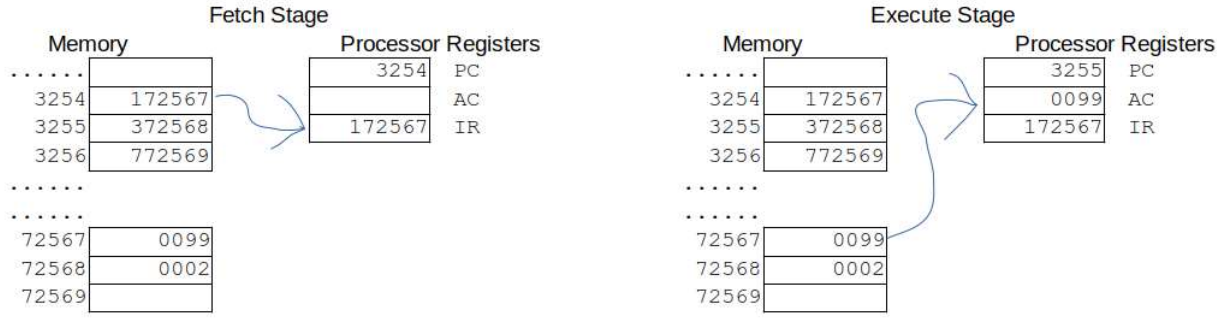


Figure 3: First Cycle. ("First Cycle" by Patrick McClanahan is licensed under CC BY-SA 4.0)

1. At the beginning the PC contains 3254, which is the memory address of the next instruction to be executed. In this example we have skipped the micro-steps, showing the IR receiving the value at the specified address.

2. The instruction at address 3254 is 172567. Remember from above - the first 4 bits are the opcode, in this case it is the number 1 (0001 in binary).
3. This opcode tells the processor to load the AC from the memory address located in the last 12 bits of the instruction - 72567.
4. Go to address 72567 and load that value, 0099, into the accumulator.

ALSO NOTICE - the PC has been incremented by 1, so it now points to the next instruction in memory.

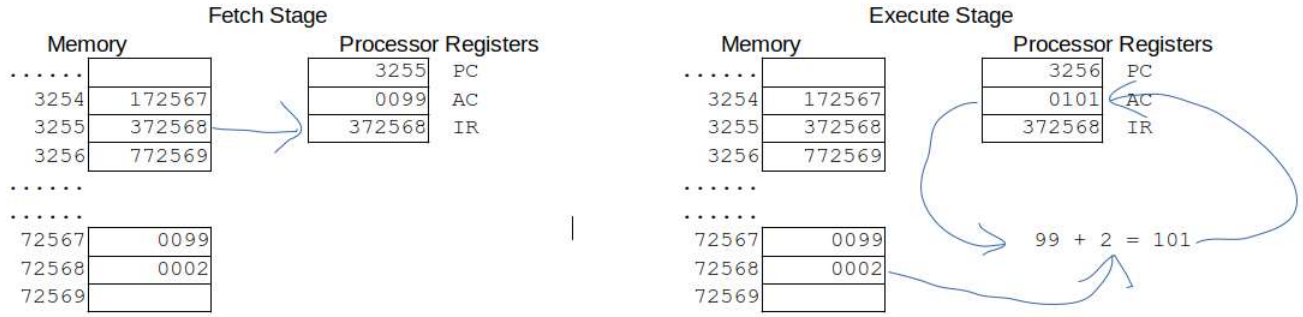


Figure 4: Second Cylce ("Second Cycle" by Patrick McClanahan is licensed under CC BY-SA 4.0)

1. Again - we start with the PC - and move the contents found at the memory address, 3255, into the IR.
2. The instruction in the IR has an opcode of 3 (0011 in binary).
3. This opcode in our example tells the processor to add the value currently stored in the AC, 0099, to the value stored at the given memory address, 72568, which is the value 2.
4. This value, 101, is stored back into the AC.

AGAIN, the PC has been incremented by one as well.

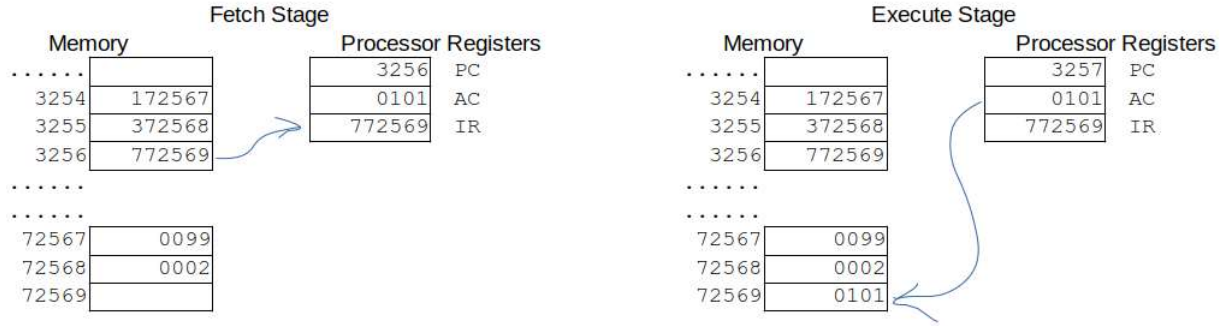


Figure 5: Third Cycle. ("Third Cycle" by Patrick McClanahan is licensed under CC BY-SA 4.0)

1. The PC points to 3256, the value 772569 is moved to the IR.
2. This instruction has an opcode of 7 (0111 in binary)
3. This opcode tells the processor to move the value in the AC, 101, to the specified memory address, 72569.

The PC has again been incremented by one - and when our simple 3 instructions are completed, whatever instruction was at that address would be executed.

This page titled 1.4.2 Instruction Cycles - Instruction Primer is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.5 Interrupts

What is an Interrupt

An interrupt is a signal emitted by hardware or software when a process or an event needs immediate attention. It alerts the processor to a high priority process requiring interruption of the current working process. In I/O devices one of the bus control lines is dedicated for this purpose and is called the *Interrupt Service Routine (ISR)*.

When a device raises an interrupt at lets say process i, the processor first completes the execution of instruction i. Then it loads the Program Counter (PC) with the address of the first instruction of the ISR. Before loading the Program Counter with the address, the address of the interrupted instruction is moved to a temporary location. Therefore, after handling the interrupt the processor can continue with process i+1.

While the processor is handling the interrupts, it must inform the device that its request has been recognized so that it stops sending the interrupt request signal. Also, saving the registers so that the interrupted process can be restored in the future, increases the delay between the time an interrupt is received and the start of the execution of the ISR. This is called Interrupt Latency.

Hardware Interrupts:
In a hardware interrupt, all the devices are connected to the Interrupt Request Line. A single request line is used for all the n devices. To request an interrupt, a device closes its associated switch. When a device requests an interrupts, the value of INTR is the logical OR of the requests from individual devices.

Sequence of events involved in handling an IRQ:

1. Devices raise an IRQ.
2. Processor interrupts the program currently being executed.
3. Device is informed that its request has been recognized and the device deactivates the request signal.
4. The requested action is performed.
5. Interrupt is enabled and the interrupted program is resumed.

Conceptually an interrupt causes the following to happen:

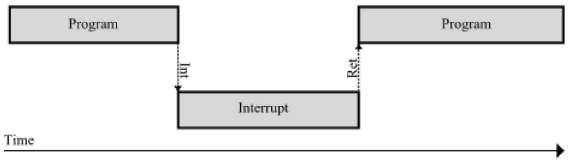


Figure 1: Concept of an interrupt. ("Concept of an Interrupt" by lemilxavier, WikiBooks is licensed under CC BY-SA 3.0)

The grey bars represent the control flow. The top line is the program that is currently running, and the bottom bar is the interrupt service routine (ISR). Notice that when the interrupt (**Int**) occurs, the program stops executing and the microcontroller begins to execute the ISR. Once the ISR is complete, the microcontroller returns to processing the program where it left off.

Handling Multiple Devices:
When more than one device raises an interrupt request signal, then additional information is needed to decide which which device to be considered first. The following methods are used to decide which device to select: Polling, Vectored Interrupts, and Interrupt Nesting. These are explained as following below.

1. **Polling:**
   In polling, the first device encountered with with IRQ bit set is the device that is to be serviced first. Appropriate ISR is called to service the same. It is easy to implement but a lot of time is wasted by interrogating the IRQ bit of all devices.
2. **Vectored Interrupts:**
   In vectored interrupts, a device requesting an interrupt identifies itself directly by sending a special code to the processor over the bus. This enables the processor to identify the device that generated the interrupt. The special code can be the starting address of the ISR or where the ISR is located in memory, and is called the interrupt vector.
3. **Interrupt Nesting:**
   In this method, I/O device is organized in a priority structure. Therefore, interrupt request from a higher priority device is

recognized where as request from a lower priority device is not. To implement this each process/device (even the processor). Processor accepts interrupts only from devices/processes having priority more than it.

What happens when external hardware requests another interrupt while the processor is already in the middle of executing the ISR for a previous interrupt request?

When the first interrupt was requested, hardware in the processor causes it to finish the current instruction, disable further interrupts, and jump to the interrupt handler.

The processor ignores further interrupts until it gets to the part of the interrupt handler that has the "return from interrupt" instruction, which re-enables interrupts.

If an interrupt request occurs while interrupts were turned off, some processors will immediately jump to that interrupt handler as soon as interrupts are turned back on. With this sort of processor, an interrupt storm "starves" the main loop background task. Other processors execute at least one instruction of the main loop before handling the interrupt, so the main loop may execute extremely slowly, but at least it never "starves".

A few processors have an interrupt controller that supports "round robin scheduling", which can be used to prevent a different kind of "starvation" of low-priority interrupt handlers.

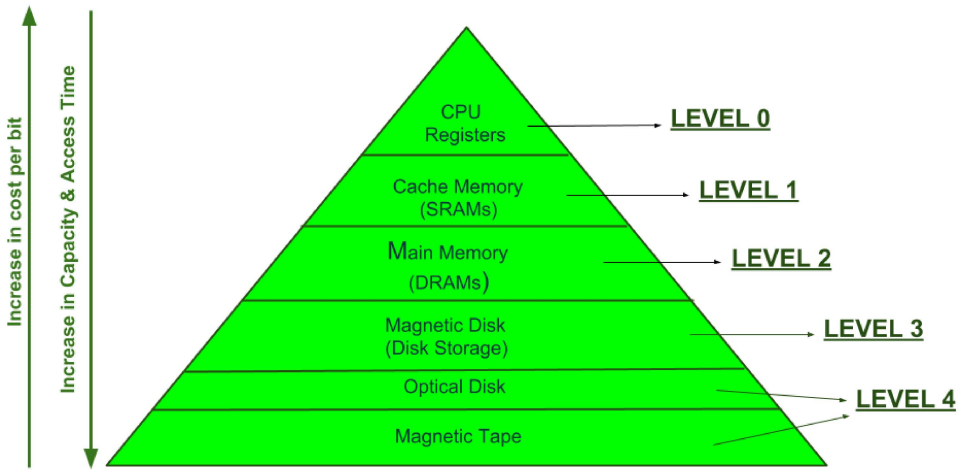
Processors priority is encoded in a few bits of PS (Process Status register). It can be changed by program instructions that write into the PS. Processor is in supervised mode only while executing OS routines. It switches to user mode before executing application programs

Adapted from:
"Interrupts" by lemilxavier, Geeks for Geeks is licensed under CC BY-SA 4.0
"Microprocessor Design/Interrupts" by lemilxavier, WikiBooks is licensed under CC BY-SA 3.0

This page titled 1.5 Interrupts is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.6 Memory Hierarchy

Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references.The figure below clearly demonstrates the different levels of memory hierarchy :



MEMORY HIERARCHY DESIGN

This Memory Hierarchy Design is divided into 2 main types:

1. **External Memory or Secondary Memory –**
   Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
2. **Internal Memory or Primary Memory –**
   Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

We can infer the following characteristics of Memory Hierarchy Design from above figure:

1. **Capacity:**
   It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
2. **Access Time:**
   It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
3. **Performance:**
   Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.
4. **Cost per bit:**
   As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

Adapted from:
"Memory Hierarchy Design and its Characteristics" by RishabhJain12, Geeks for Geeks is licensed under CC BY-SA 4.0

This page titled 1.6 Memory Hierarchy is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.7 Cache Memory

Cache Memory

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.

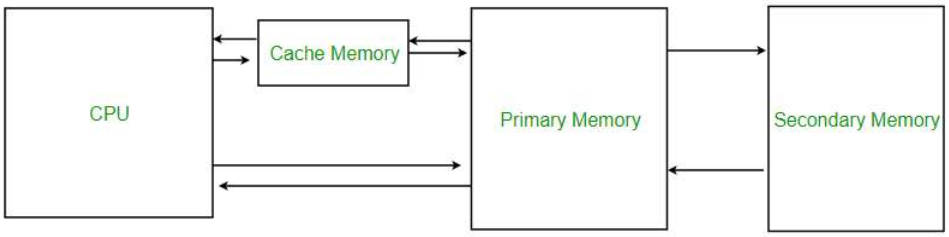


Figure 1: Cahce Memory. ("Cache Memory" by VaibhavRai3, Geeks for Geeks is licensed under CC BY-SA 4.0)

Levels of memory:

- **Level 1 or Register**
  It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory**
  It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory (Primary Memory in the image above)**
  It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory**
  It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance:
When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

```
Hit ratio = hit / (hit + miss) =  no. of hits/total accesses
```

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Application of Cache Memory

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Types of Cache

- **Primary Cache**
  A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
- **Secondary Cache**
  Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

Locality of reference
Since size of cache memory is less as compared to main memory. So to check which part of main memory should be given priority and loaded in cache is decided based on locality of reference. Locality will be discussed in greater detail later on.

Adapted from:
"Cache Memory in Computer Organization" by VaibhavRai3, Geeks for Geeks is licensed under CC BY-SA 4.0

This page titled 1.7 Cache Memory is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.7.1 Cache Memory - Multilevel Cache

Multilevel Cache

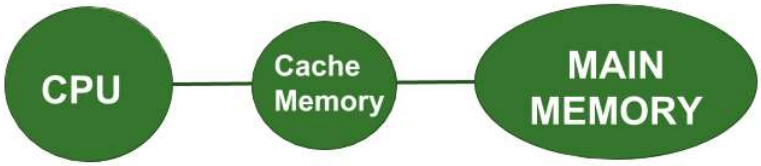
Multilevel cache is one of the techniques to improve cache performance by reducing the “miss penalty”. The term miss penalty refers to the extra time required to bring the data into cache from the main memory whenever there is a “miss” in cache .

For clear understanding let us consider an example where CPU requires 10 memory references for accessing the desired information and consider this scenario in the following 3 cases of System design :

System Design without cache memory

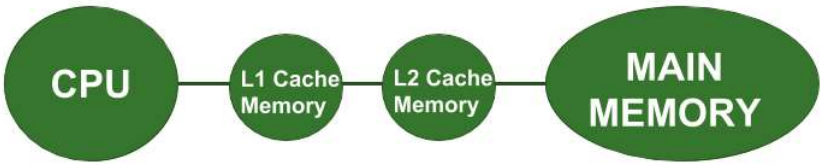
Here the CPU directly communicates with the main memory and no caches are involved.
In this case, the CPU needs to access the main memory 10 times to access the desired information.

System Design with cache memory



Here the CPU at first checks whether the desired data is present in the cache memory or not i.e. whether there is a “hit” in cache or “miss” in cache. Suppose there are 3 miss in cache memory then the main memory will be accessed only 3 times. We can see that here the miss penalty is reduced because the main memory is accessed a lesser number of times than that in the previous case.

System Design with Multilevel cache memory



Here the cache performance is optimized further by introducing multilevel Caches. As shown in the above figure, we are considering 2 level cache Design. Suppose there are *3 miss* in the L1 cache memory and out of these 3 misses there are *2 miss* in the L2 cache memory then the Main Memory will be accessed only 2 times. It is clear that here the miss penalty is reduced considerably than that in the previous case thereby improving the performance of cache memory.

NOTE :
We can observe from the above 3 cases that we are trying to decrease the number of main memory references and thus decreasing the miss penalty in order to improve the overall system performance. Also, it is important to note that in the multilevel cache design, L1 cache is attached to the CPU and it is small in size but fast. Although, L2 cache is attached to the primary cache i.e. L1 cache and it is larger in size and slower but still faster than the main memory.

```
Effective Access Time = Hit rate * Cache access time
                      + Miss rate * Lower level access time
```

Average access Time For Multilevel Cache:(T_{avg})

$T_{avg} = H_1 * C_1 + (1 - H_1) * (H_2 * C_2 + (1 - H_2) * M)$

where
H1 is the Hit rate in the L1 caches.
H2 is the Hit rate in the L2 cache.
C1 is the Time to access information in the L1 caches.
C2 is the Miss penalty to transfer information from the L2 cache to an L1 cache.
M is the Miss penalty to transfer information from the main memory to the L2 cache.

Adapted from:
"Multilevel Cache Organisation" by shreya garg 4, Geeks for Geeks is licensed under CC BY-SA 4.0

This page titled 1.7.1 Cache Memory - Multilevel Cache is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.7.2 Cache Memory - Locality of reference

Locality of Reference

Locality of reference refers to a phenomenon in which a computer program tends to access same set of memory locations for a particular time period. In other words, Locality of Reference refers to the tendency of the computer program to access instructions whose addresses are near one another. The property of locality of reference is mainly shown by:

1. Loops in program cause the CPU to repeatedly execute a set of instructions that constitute the loop.
2. Subroutine calls, cause the set of instructions are fetched from memory each time the subroutine gets called.
3. References to data items also get localized, meaning the same data item is referenced again and again.

Even though accessing memory is quite fast, it is possible for repeated calls for data from main memory can become a bottleneck. By using faster cache memory, it is possible to speed up the retrieval of frequently used instructions or data.

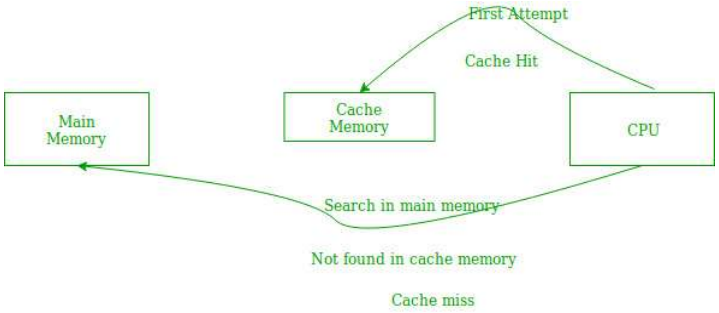


Figure 1: Cache Hit / Cache Miss. ("Cache Hit / Miss" by balwant_singh, Geeks for Geeks is licensed under CC BY-SA 4.0)

In the above figure, you can see that the CPU wants to read or fetch the data or instruction. First, it will access the cache memory as it is near to it and provides very fast access. If the required data or instruction is found, it will be fetched. This situation is known as a cache hit. But if the required data or instruction is not found in the cache memory then this situation is known as a cache miss. Now the main memory will be searched for the required data or instruction that was being searched and if found will go through one of the two ways:

1. The inefficent method is to have the CPU fetch the required data or instruction from main memory and use it. When the same data or instruction is required again the CPU again has to access the main memory to retrieve it again .
2. A much more efficient method is to store the data or instruction in the cache memory so that if it is needed soon again in the near future it could be fetched in a much faster manner.

Cache Operation:
This concept is based on the idea of locality of reference. There are two ways in which data or instruction are fetched from main memory then get stored in cache memory:

1. **Temporal Locality**
   Temporal locality means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data.

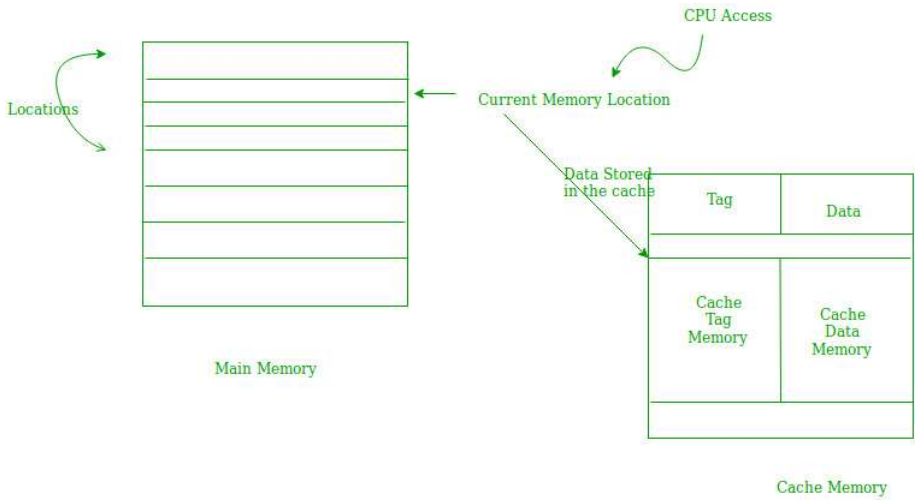


Figure 1: Temporal Locality. ("Temporal Locality" by balwant_singh, Geeks for Geeks is licensed under CC BY-SA 4.0)

2. When CPU accesses the current main memory location for reading required data or instruction, it also gets stored in the cache memory which is based on the fact that same data or instruction may be needed in near future. This is known as temporal locality. If some data is referenced, then there is a high probability that it will be referenced again in the near future.

3. **Spatial Locality**
   Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed by the processor soon. This is different from the temporal locality in that we are making a guess that the data/instructions will be needed soon. With temporal locality we were talking about the actual memory location that was being fetched.

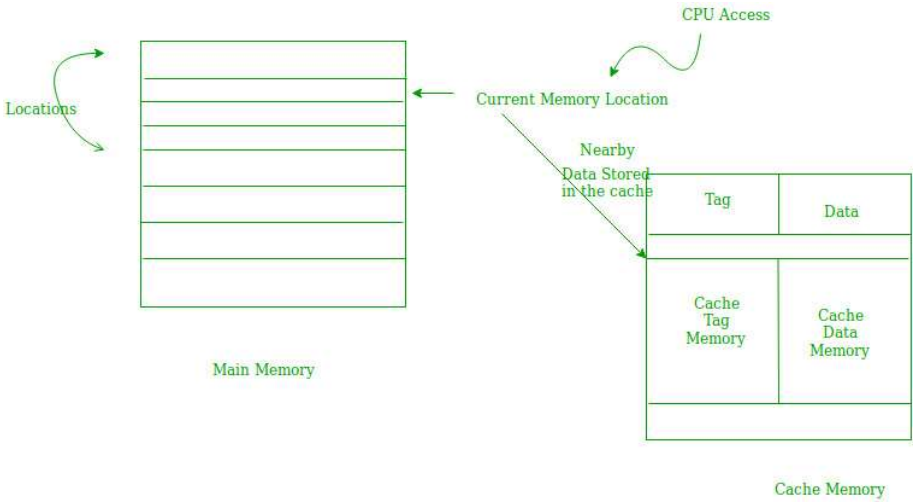


Figure 1: Spatial Locality. ("Spatial Locality" by **balwant_singh,** Geeks for Geeks is licensed under CC BY-SA 4.0)

Adapted From:
"Locality of Reference and Cache Operation in Cache Memory" by balwant_singh, Geeks for Geeks is licensed under CC BY-SA 4.0

This page titled 1.7.2 Cache Memory - Locality of reference is shared under a CC BY-SA license and was authored, remixed, and/or curated by Patrick McClanahan.

1.8 Direct Memory Access

DMA

There are three techniques used for I/O operations: programmed I/O, interrupt-driven I/O, and direct memory access (DMA). As long as we are discussing DMA, we will also discuss the other two techniques.

The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface. The CPU is interfaced using special communication links by the peripherals connected to any computer system. These communication links are used to resolve the differences between CPU and peripheral. There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.
3. Direct memory access(DMA).

Now let's discuss each mode one by one.

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

   Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.

2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing. \

Note: Both the methods programmed I/O and Interrupt-driven I/O require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.
- The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.

3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

Bus Request : It is used by the DMA controller to request the CPU to relinquish the control of the buses.