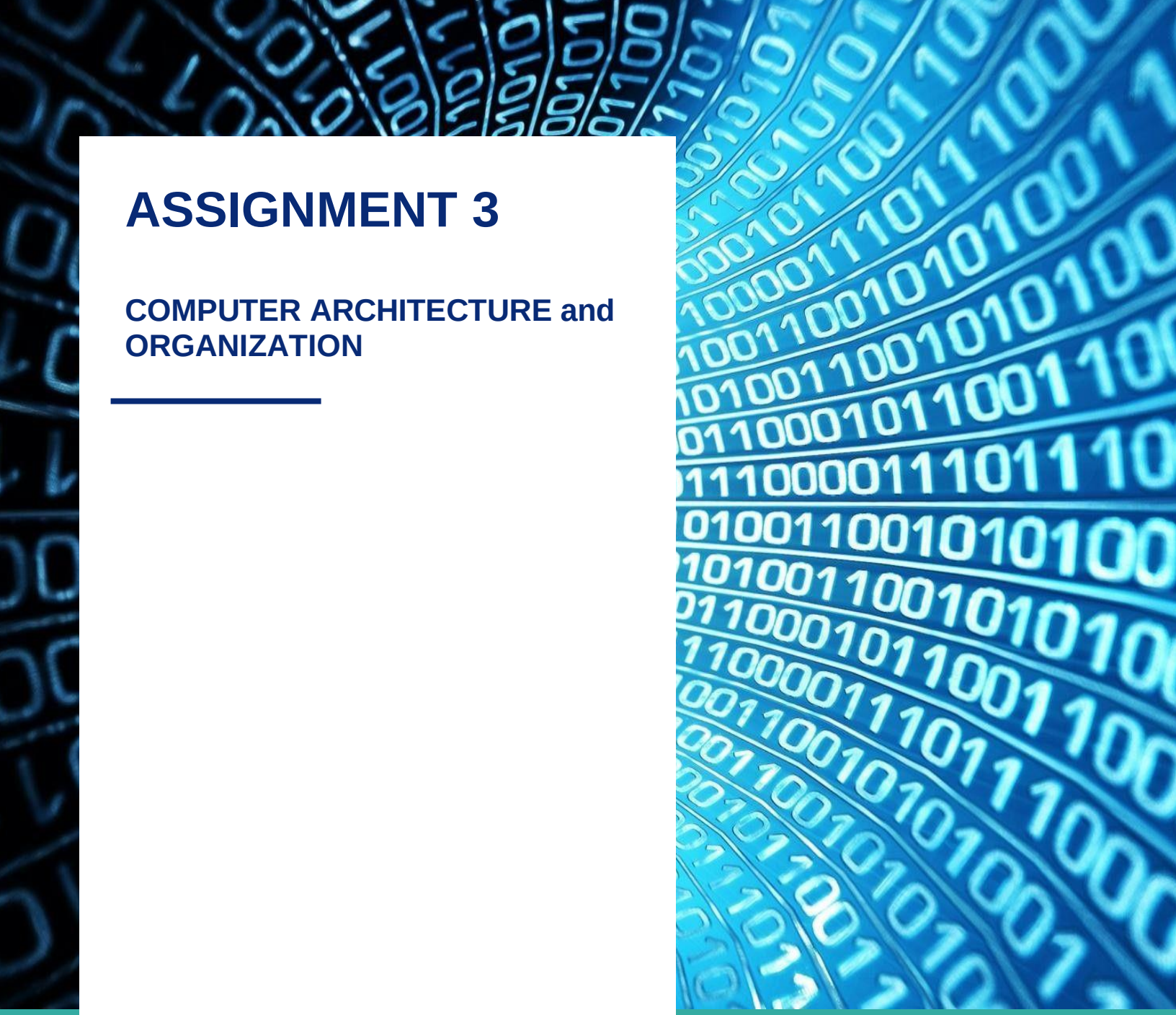


ASSIGNMENT 3

COMPUTER ARCHITECTURE and ORGANIZATION

MAY 2

SWAGAT SHUBHAM BHUYAN
18-1-5-059
Sec: A SEM: IV

SUBMITTED TO:
Dr. Malaya Dutta Borah

Q.:

How do you design a CPU? Explain it with your own words by giving technical justifications with appropriate figures/ circuits.

Sol.:

Computers have come a long way from mechanical devices capable of maybe one calculation per second, to CPUs running at kilohertz and megahertz speeds. In the early days of electronic computing, processors were typically made faster by improving the switching time of the transistors inside the chip - the ones that make up all the logic gates, ALUs, Control Units, etc. But just making transistors faster and more efficient only went so far, so processor designers have developed various techniques to boost performance allowing not only simple instructions to run fast, but also performing much more sophisticated operations using circuits instead of using simple ALU instructions. This is what is covered in making an advanced CPU – the architecture and organization of various aspects that make up a CPU.

WHAT IS A CPU

A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions.

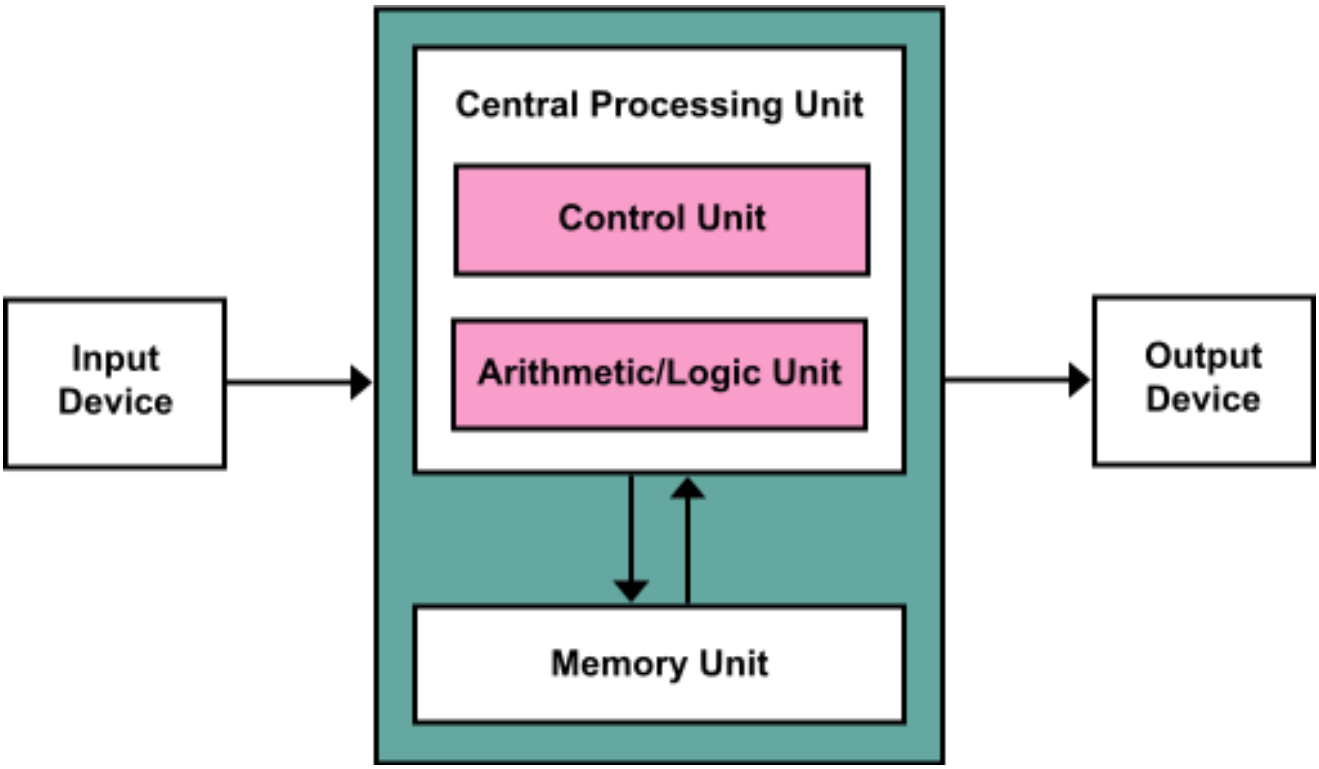


FIG: Basic block diagram of a CPU

The Design of a CPU can be basically broken down to the following components:

- ALU (Arithmetic and Logical Unit)
- Registers
- Memory and I/O interfaces
- Control Unit
- Clock
- Buses

ORGANIZATION OF A CPU

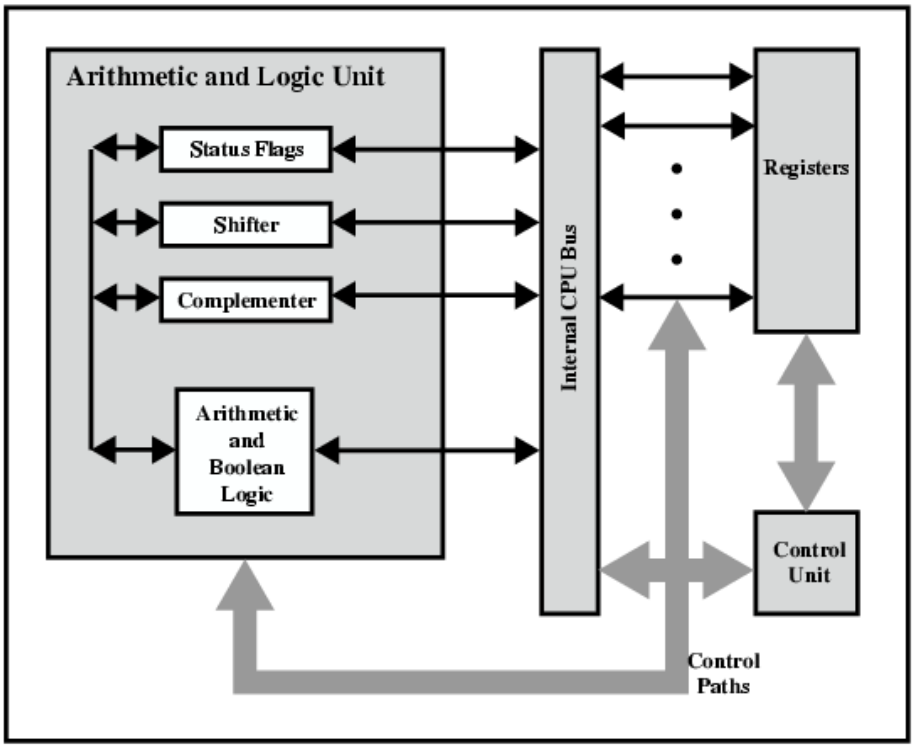
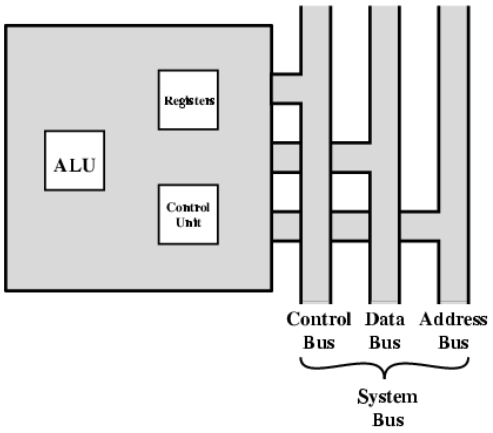


FIG: Internal Organization of a CPU

BASIC CPU OPERATIONS:

- **Fetch instruction:** The CPU reads an instruction from memory.
- **Interpret instruction:** The instruction is decoded to determine what action is required.
- **Fetch data:** The execution of an instruction may require reading data from memory or an I/O module.
- **Process data:** The execution of an instruction may require performing some arithmetic or logical operation on data.
- **Write data:** The results of an execution may require writing data to memory or an I/O module.

For the various data required and manipulated to achieve the above functions to be available, **system buses** are used to carry data around inside the architecture between various components.



TO DESIGN A CPU:

For a productive and efficient design of CPU, one must keep in mind the following aspects of a CPU and develop them proficiently:

- ALU (Arithmetic and Logical Unit)
  - Logic Unit
  - Arithmetic Unit
- Register Organization
  - User Registers
  - Control and Status Registers
- Cache memory organization
- I/O architecture
- Control Unit
  - Hardwired CU
  - Micro programmable CU
- Clock
- Buses
  - Control Bus
  - Data Bus
  - Address Bus
- Pipelining
- ISA (Instruction Set Architecture)
- Energy and Power

ARITHMETIC AND LOGICAL UNIT (ALU)

The ALU is the mathematical brain of a computer. It is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. It is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

ALUs in its early stages had fewer instructions, hence fewer unique circuits. This meant that to perform complex calculations, the CPU relied on multiple cycles of basic ALU instruction circuits to get the job done. However with higher generations, calculational complexities sky-rocketed, hence the need for an ALU which could handle bigger calculations without compromising on clock cycles. This was achieved by implementing different Adder and Multiplier circuits along with the basic logic gates into the ALU of modern CPUs.

An ALU, like it sounds, consists of the Logical unit and the Arithmetic unit:

LOGICAL UNIT

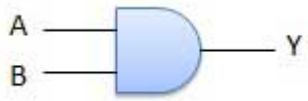
This section of the ALU consists of all the logic gates which deal with logical operations, which although carry out bit manipulations, also help out in basic calculations as well, thanks to the versatility in logic gates when designed into a circuit. The basic Logic gates are:

AND Gate

A circuit which performs an AND operation is shown in figure. It has n input (n >= 2) and one output.

Y = A AND B AND C ..... N
Y = A.B.C ..... N
Y = ABC ..... N

Logic diagram



Truth Table

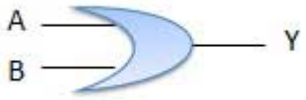
Inputs		Output
A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

A circuit which performs an OR operation is shown in figure. It has n input (n >= 2) and one output.

Y = A OR B OR C ..... N
Y = A + B + C ..... N

Logic diagram



Truth Table

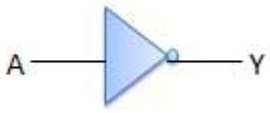
Inputs		Output
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gate

NOT gate is also known as **Inverter**. It has one input A and one output Y.

Y = NOT A
Y = $\overline{A}$

Logic diagram



Truth Table

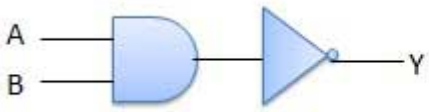
Inputs	Output
A	B
0	1
1	0

NAND Gate

A NOT-AND operation is known as NAND operation. It has n input (n >= 2) and one output.

Y = A NOT AND B NOT AND C ..... N
Y = A NAND B NAND C ..... N

Logic diagram



Truth Table

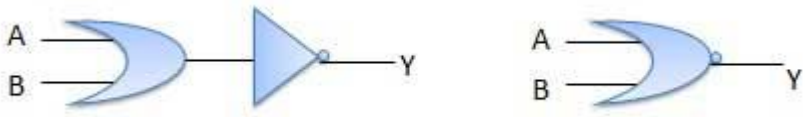
Inputs		Output
A	B	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

A NOT-OR operation is known as NOR operation. It has n input (n >= 2) and one output.

Y = A NOT OR B NOT OR C ..... N
Y = A NOR B NOR C ..... N

Logic diagram



Truth Table

Inputs		Output
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

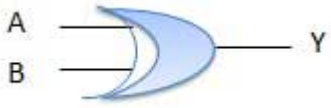
XOR Gate

XOR or Ex-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-OR gate is abbreviated as EX-OR gate or sometime as X-OR gate. It has n input (n >= 2) and one output.

Y = A XOR B XOR C ..... N
Y = $A \oplus B \oplus C$ ..... N

Y = $\overline{A}B + A\overline{B}$

Logic diagram



Truth Table

Inputs		Output
A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

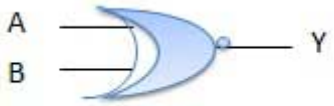
XNOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. The exclusive-NOR gate is abbreviated as EX-NOR gate or sometime as X-NOR gate. It has n input (n >= 2) and one output.

$$Y = A \text{ XOR } B \text{ XOR } C \ldots\ldots N$$

$$Y = A \odot B \odot C \ldots\ldots N$$

$$Y = \overline{A}\,B + A\,\overline{B}$$

Logic diagram



Truth Table

Inputs		Output
A	B	A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1

ARITHMETIC UNIT

Deals with the arithmetic aspect of CPU calculations, using simple adder circuits with simpler cycles, and complex multipliers to reduce clock cycles for instructions.

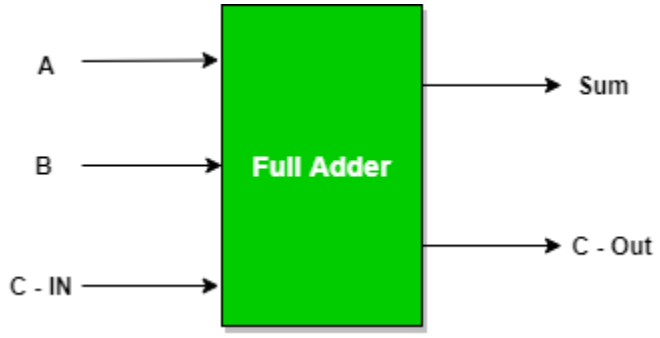
HALF ADDER:

A half adder just adds two bits together and gives a two-bit output.

FULL ADDER:

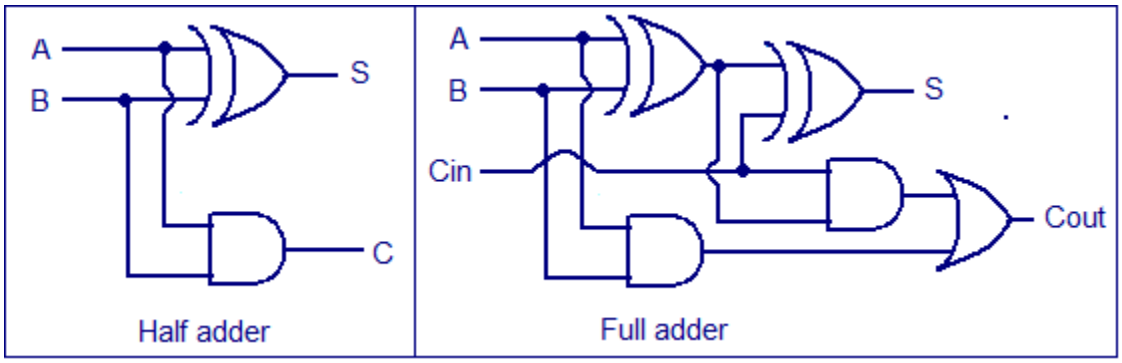
Full Adder is the adder which adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.

A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to the another.



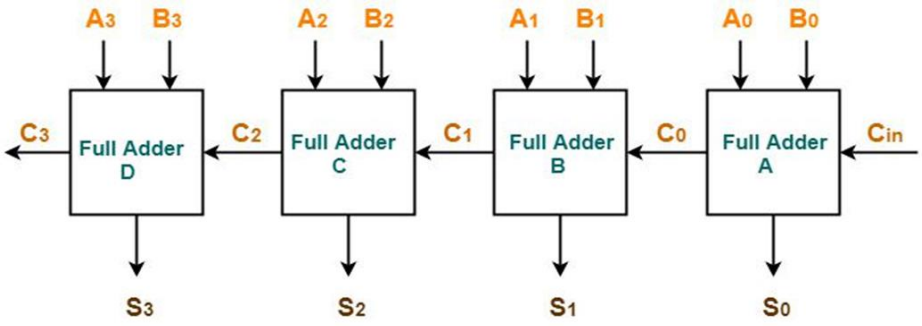
Inputs			Outputs	
A	B	C – IN	Sum	C - Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A Full adder can be made by combining two half adder circuits together (a half adder is a circuit that adds two input bits and outputs a sum bit and a carry bit).



RIPPLE CARRY ADDER:

Ripple Carry Adder is a combinational logic circuit. It is used for the purpose of adding two n-bit binary numbers. It requires n full adders in its circuit for adding two n-bit binary numbers. It is also known as n-bit parallel adder.

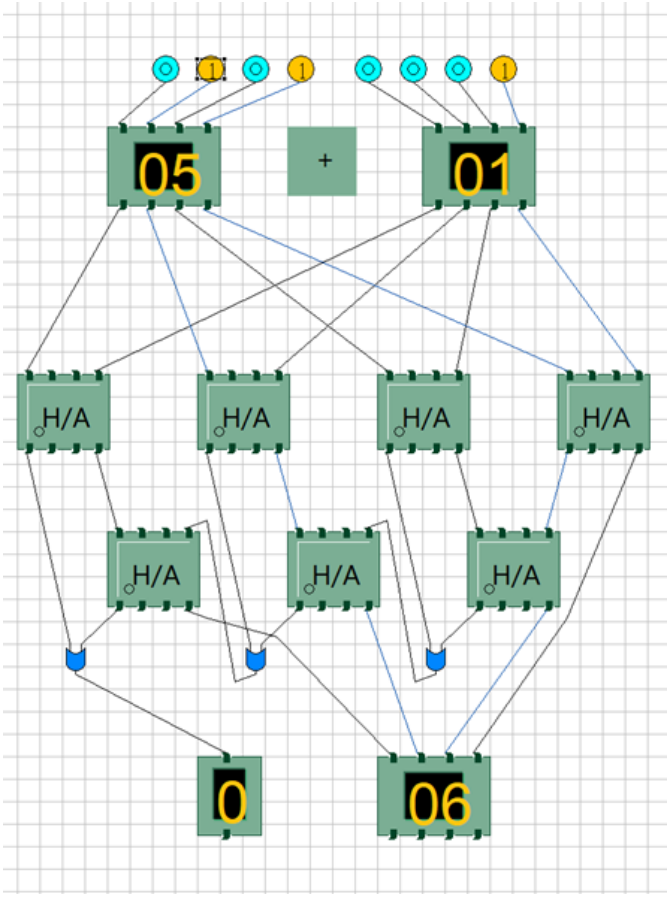


4-bit Ripple Carry Adder

CARRY LOOK-AHEAD ADDER:

A carry look-ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic.

Circuit Diagram (using Simulator)



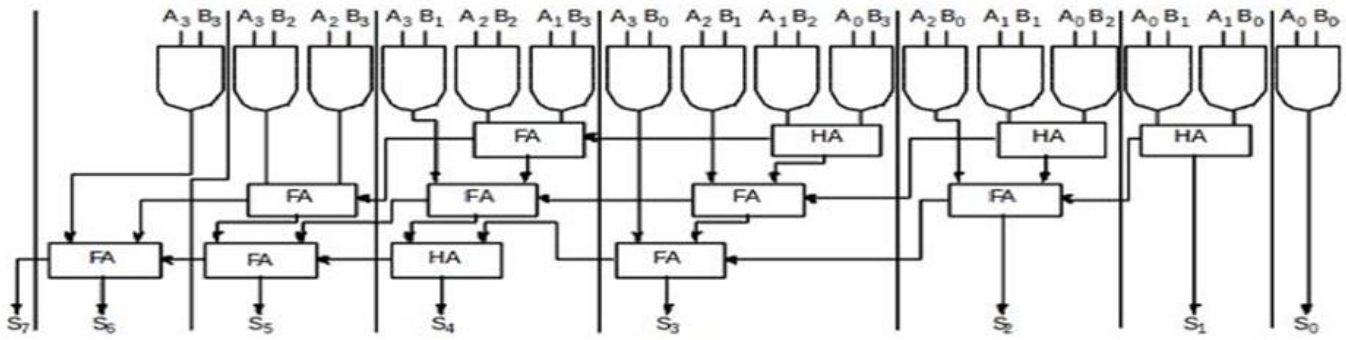
H/A: Half Adder

COMBINATIONAL MULTIPLIER:

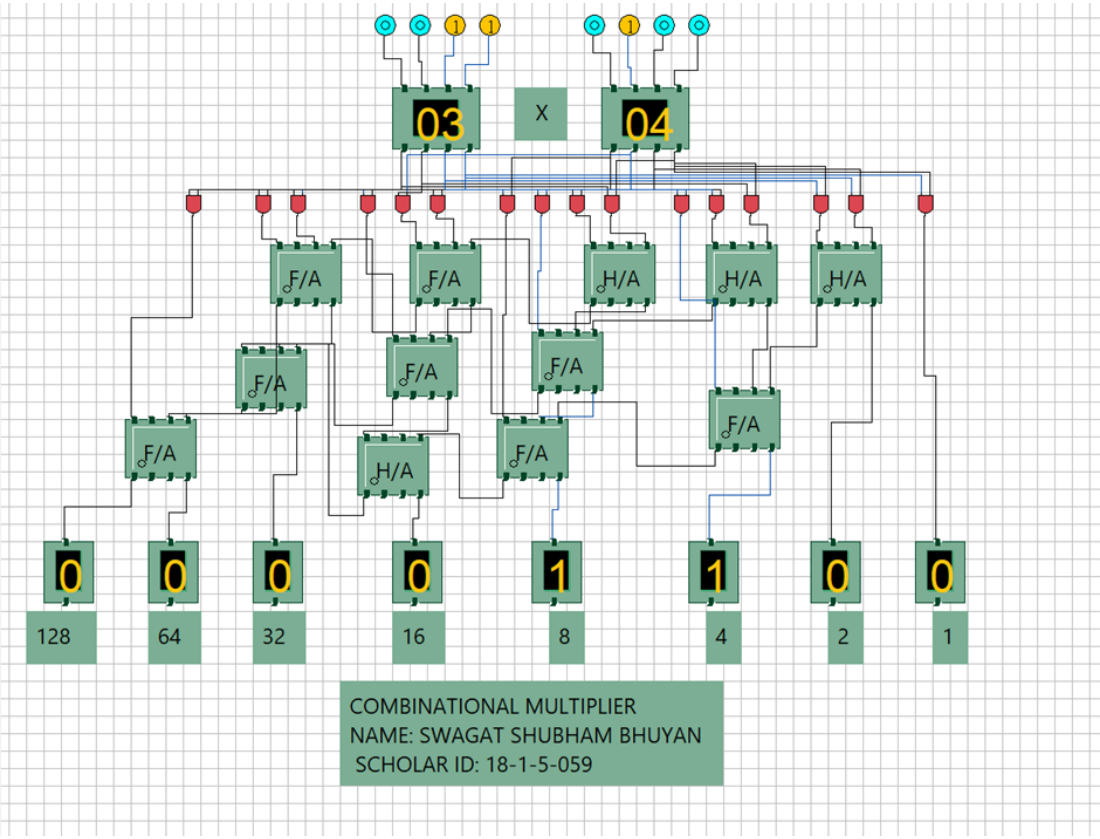
Combinational Multipliers do multiplication of two unsigned binary numbers. Each bit of the multiplier is multiplied against the multiplicand, the product is aligned according to the position of the bit within the multiplier and the resulting products are then summed to form the final result. Main gain of binary multiplication is that the generation of intermediate

products are simple: if the multiplier bit is a 1, the product is an appropriately shifted copy of the multiplicand; if the multiplier bit is a 0, the product is simply 0.

Circuit Diagram



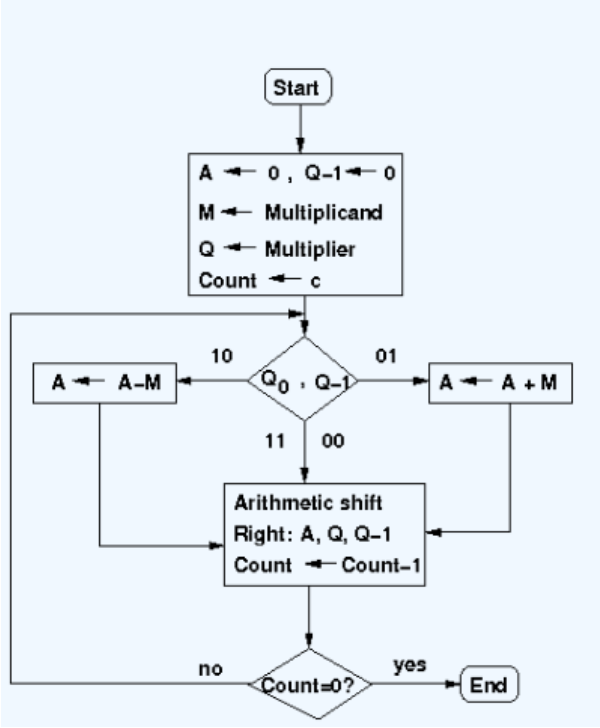
Circuit Diagram using Simulator



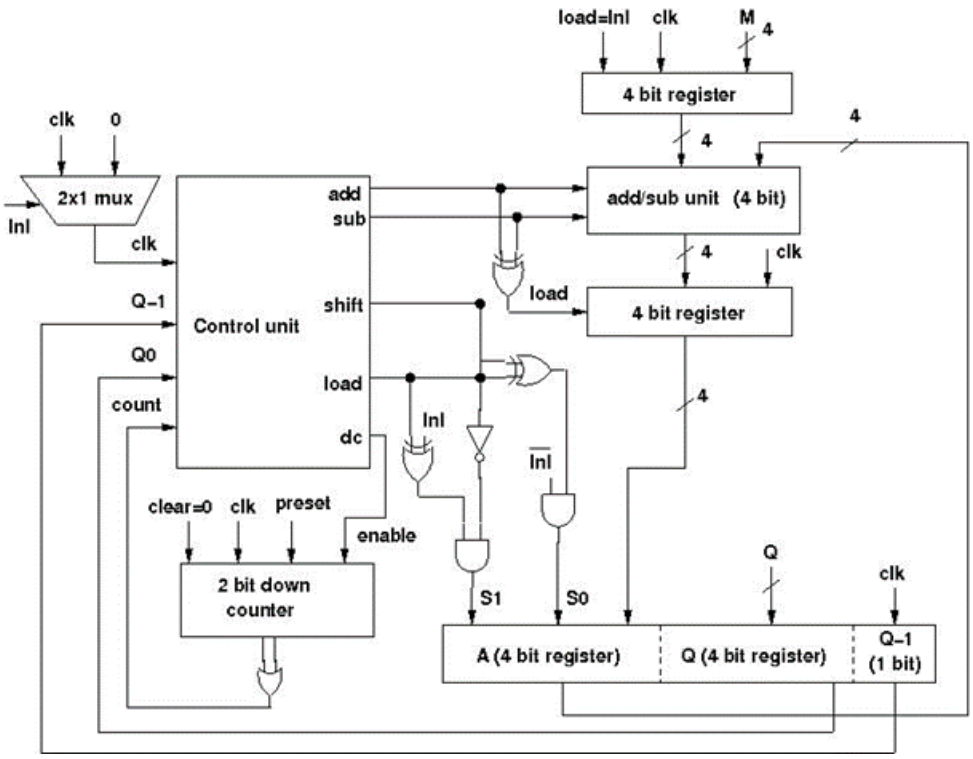
BOOTH's MULTIPLIER:

Booth's multiplication algorithm is an algorithm which multiplies 2 signed integers in 2's complement. The algorithm is depicted in the following figure with a brief description. This approach uses fewer additions and subtractions than more straightforward algorithms.

The Following Flow chart clearly explains the algorithm:



Circuit Diagram:



The Logic gates along with simple Adders and complex Multipliers help make the ALU a powerful tool for an efficient design paradigm of a CPU.

NOTE: ALU does not have Counters, but can INCREMENT, DECREMENT by 1 using pre-existing circuits.

REGISTER ORGANIZATION

In Computer Architecture, the Registers are very fast computer memory which are used to execute programs and operations efficiently. Within the CPU, registers function as a level of memory above main memory and cache in the hierarchy. This is done by giving access to commonly used values, i.e., the values which are in the point of operation/execution at that time.

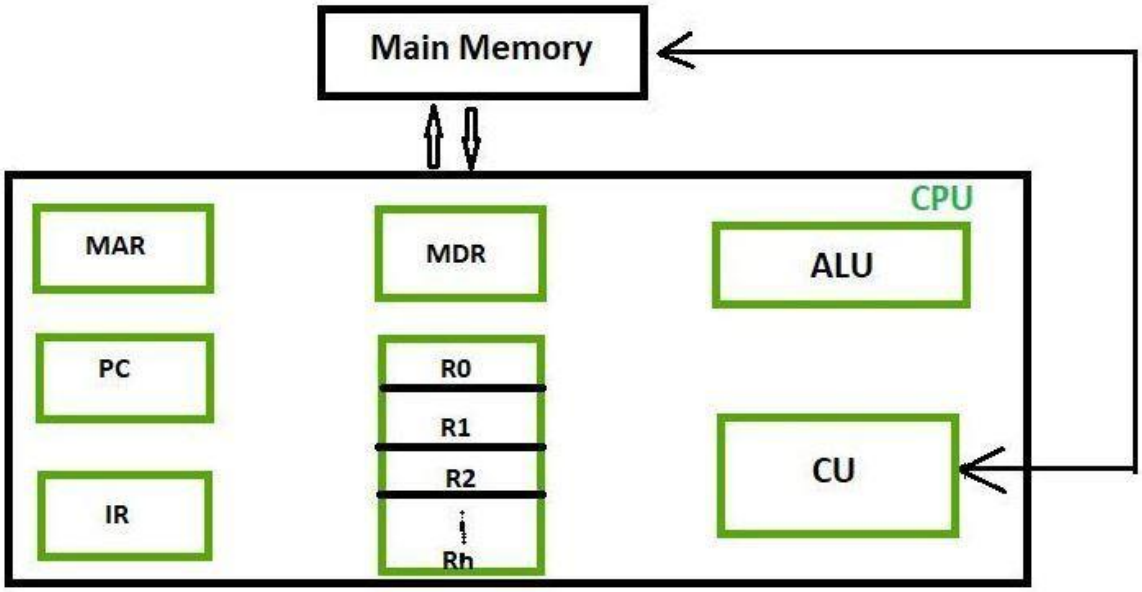


FIG: Register Block Diagram

User-visible registers:

These enable the machine or assembly language programmer to minimize main memory references by optimizing use of registers.

- **Memory Address Registers (MAR):**
  - It holds the address of the location to be accessed from memory. MAR and MDR (Memory Data Register) together facilitate the communication of the CPU and the main memory.
- **Memory Data Registers (MDR):**
  - It contains data to be written into or to be read out from the addressed location.
- **Program Counter (PC):**
  - Program Counter (PC) is used to keep the track of execution of the program. It contains the memory address of the next instruction to be fetched. PC points to the address of the next instruction to be fetched from the main memory when the previous instruction has been successfully completed. Program Counter (PC) also functions to count the number of instructions.
- **Instruction Register (IR):**
  - It is the register which holds the instruction which is currently been executed.

Control and Status registers:

These enable the machine or assembly language programmer to minimize main memory references by optimizing use of registers.

- **General Purpose Registers:**
  - These are numbered as R0, R1, R2….Rn, and used to store temporary data during any ongoing operation. Its content can be accessed by assembly programming.

- **Accumulator:**
    - This is the most frequently used register used to store data taken from memory. It is in different numbers in different microprocessors.
- **Segment pointers:**
    - In a machine with segmented addressing, a segment register holds the address of the base of the segment. There may be multiple registers: for example, one for the operating system and one for the current process.
- **Index registers:**
    - These are used for indexed addressing and may be auto-indexed.
- **Stack pointer:**
    - If there is user-visible stack addressing, then typically the stack is in memory and there is a dedicated register that points to the top of the slack. This allows implicit addressing; that is, push, pop, and other slack instructions need not contain an explicit stack operand.
- **Condition codes register (also referred to as flags):**
    - Condition codes are bits set by the CPU hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero, or overflow result. In addition to the result itself being stored in a register or memory, a condition code is also set. The code may subsequently be tested as part of a conditional branch operation.

WORKING

Typically, the CPU updates the PC after each instruction fetch so that the PC always points to the next instruction to be executed. A branch or skip instruction will also modify the contents of the PC. The fetched instruction is loaded into an IR, where the opcode and operand specifiers are analyzed. Data are exchanged with memory using the MAR and MBR. In a bus-organized system, the MAR connects directly to the address bus, and the MBR connects directly to the data bus. User-visible registers, in turn, exchange data with the MBR.

All CPU designs include a register or set of registers, often known as the program status word (PSW), that contain status information. The PSW typically con-tains condition codes plus other stains information. Common fields or flags include the following:

- **Sign:** Contains the sign bit of the result of the last arithmetic operation.
- **Zero:** Set when the result is 0.
- **Carry:** Set if an operation resulted in a carry (addition) into or borrow (sub-traction) out of a high-order hit. Used for multiword arithmetic operations.

- **Equal:** Set if a logical compare result is equality.
- **Overflow:** Used to indicate arithmetic overflow
- **Interrupt enable/disable:** Used to enable or disable interrupts.
- **Supervisor:** Indicates whether the CPU is executing in supervisor or user mode. Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

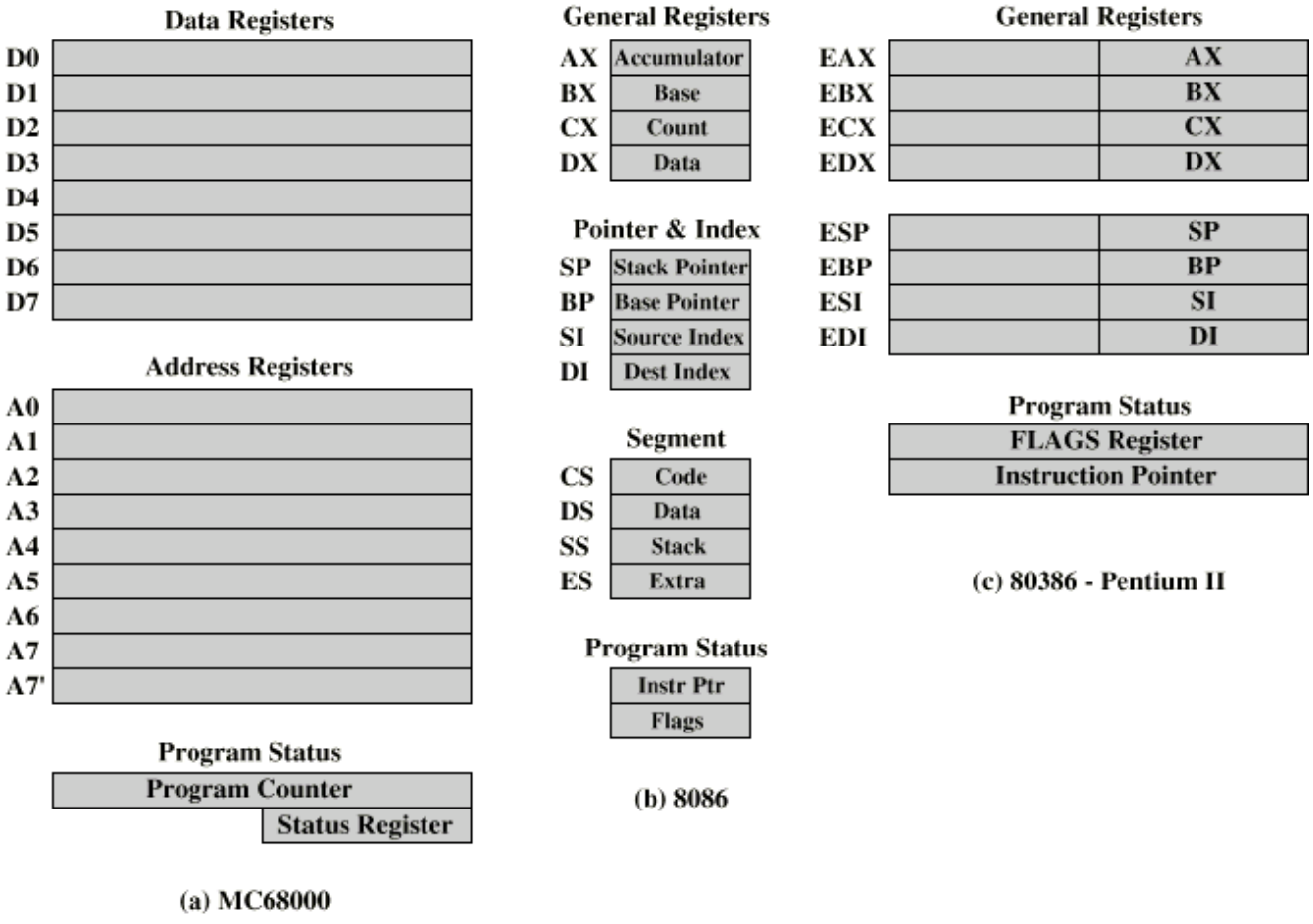


FIG: Register Organization map

CACHE MEMORY ORGANIZATION

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.

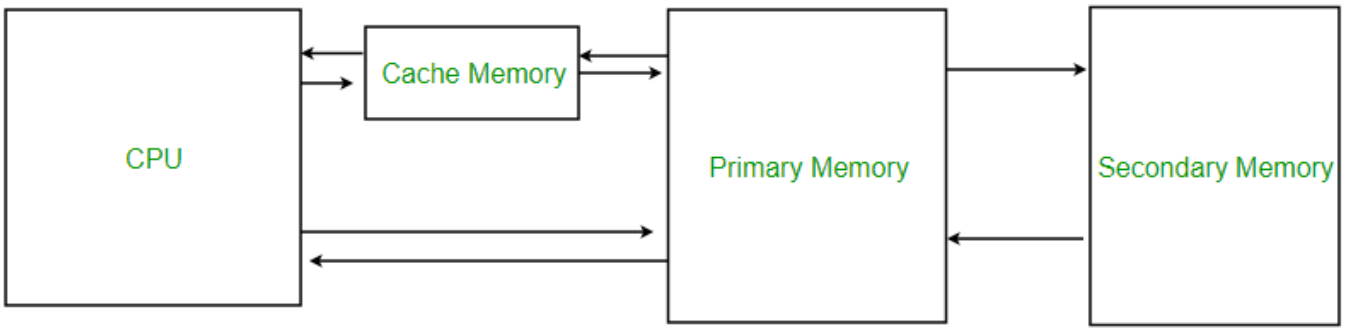


FIG: Cache memory organization

WORKING:

- 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.

MAPPING:

1. **Direct Mapping**

   The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line.

2. **Associative mapping**

   In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the

block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.

3. Set-associative Mapping

This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a set. Then a block in memory can map to any one of the lines of a specific set. Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques.

LEVELS

- **L1 (Level 1)** cache is the fastest memory that is present in a computer system. In terms of priority of access, L1 cache has the data the CPU is most likely to need while completing a certain task.
- **L2 (Level 2)** cache is slower than L1 cache, but bigger in size. Its size typically varies between 256KB to 8MB, although the newer, powerful CPUs tend to go past that. L2 cache holds data that is likely to be accessed by the CPU next. In most modern CPUs, the L1 and L2 caches are present on the CPU cores themselves, with each core getting its own cache.
- **L3 (Level 3)** cache is the largest cache memory unit, and also the slowest one. It can range between 4MB to upwards of 50MB. Modern CPUs have dedicated space on the CPU die for the L3 cache, and it takes up a large chunk of the space.

CACHE HIT OR MISS

The data flows from the RAM to the L3 cache, then the L2, and finally L1. When the processor is looking for data to carry out an operation, it first tries to find it in the L1 cache. If the CPU is able to find it, the condition is called a cache hit. It then proceeds to find it in L2, and then L3. If it doesn't find the data, it tries to access it from the main memory. This is called a cache miss.

INPUT OUTPUT ARCHITECTURE

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

PERIPHERAL DEVICES

A Peripheral Device is defined as the device which provides input/output functions for a computer and serves as an auxiliary computer device without computing-intensive functionality. Generally peripheral devices, however, are not essential for the computer to perform its basic tasks, they can be thought of as an enhancement to the user's experience. A peripheral device is a device that is connected to a computer system but is not part of the core computer system architecture.

- **Input Devices:**
  - The input devices is defined as it converts incoming data and instructions into a pattern of electrical signals in binary code that are comprehensible to a digital computer.
  - Keyboard, mouse, scanner, microphone etc.

- **Output Devices:**
  - An output device is generally reverse of the input process and generally translating the digitized signals into a form intelligible to the user. The output device is also performed for sending data from one computer system to another. For some time punched-card and paper tape readers were extensively used for input, but these have now been supplanted by more efficient devices.
  - Monitors, headphones, printers etc.

- **Storage Devices:**
  - Storage devices are used to store data in the system which is required for performing any operation in the system. The storage device is one of the most requirement devices and also provide better compatibility.
  - Hard disk, magnetic tape, Flash memory etc.

INTERFACES

Interface is a shared boundary between two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

Peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These components are called input-output interface units because they provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

MODES OF I/O TRANSFER

- **Programmed I/O:**
  - Programmed I/O instructions are the result of I/O instructions written in computer program. Each data item transfer is initiated by the instruction in the program.
  - Usually the program controls data transfer to and from CPU and peripheral. Transferring data under programmed I/O requires constant monitoring of the peripherals by the CPU.

- **Interrupt Initiated I/O**
  - In the programmed I/O method the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is time consuming process because it keeps the processor busy needlessly.
  - This problem can be overcome by using interrupt initiated I/O. In this when the interface determines that the peripheral is ready for data transfer, it generates an interrupt. After receiving the interrupt signal, the CPU stops the task which it is processing and service the I/O transfer and then returns back to its previous processing task.

- **Direct Memory Access**
  - Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This technique is known as DMA.
  - In this, the interface transfer data to and from the memory through memory bus. A DMA controller manages to transfer data between peripherals and memory unit.
  - Many hardware systems use DMA such as disk drive controllers, graphic cards, network cards and sound cards etc. It is also used for intra chip data transfer in multicore processors. In DMA, CPU would initiate the transfer, do other operations while the transfer is in progress and receive an interrupt from the DMA controller when the transfer has been completed.

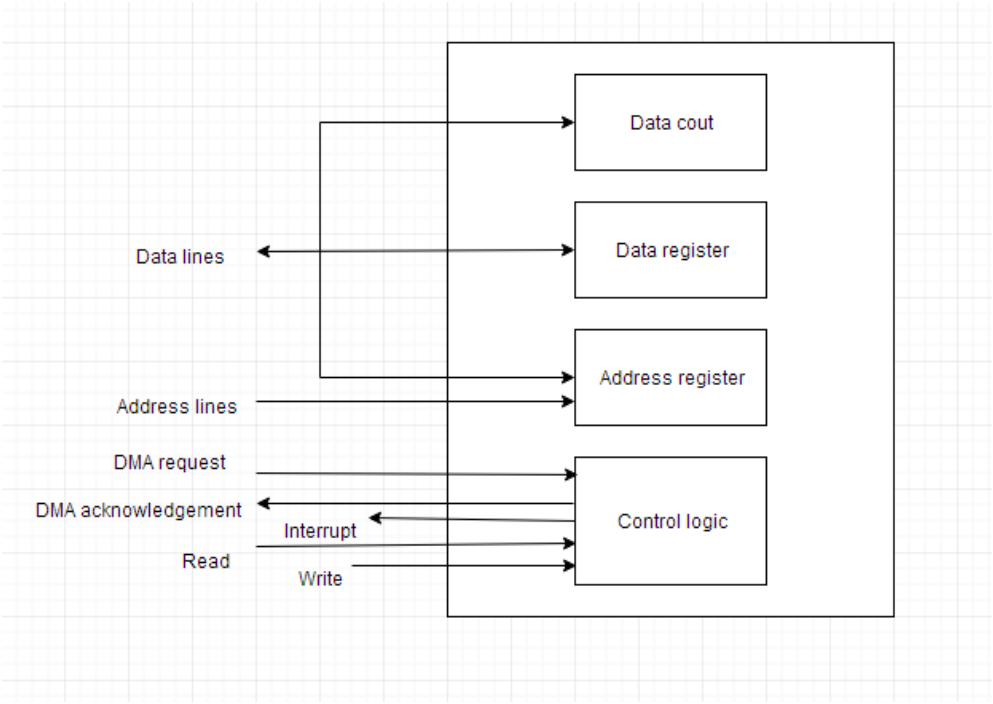


FIG: DMA block diagram

INTERRUPTS

There are many situations where other tasks can be performed while waiting for an I/O device to become ready. A hardware signal called an Interrupt will alert the processor when an I/O device becomes ready. Interrupt-request line is usually dedicated for this purpose. For example, consider, COMPUTE and PRINT routines.

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 interrupt, the value of INTR is the logical OR of the requests from individual devices. E.g.- a keypress.

- **A maskable interrupt** is a one that can be suppressed by software/code. It may be ignored. Usually there are standard interrupt masking techniques for every processor, so that it may not be interrupted while performing some crucial task. Maskable interrupts can be ignored using these methods.

- **Non-maskable interrupts** are, likewise, those which can (and should) not be ignored. So, events like critical hardware failure and system resets are attached to non-maskable interrupts to ensure that there's a "way out".

SOFTWARE INTERRUPTs:

A software interrupt is a type of interrupt that is caused either by a special instruction in the instruction set or by an exceptional condition in the processor itself. A software interrupt is invoked by software, unlike a hardware interrupt, and is considered one of the ways to communicate with the kernel or to invoke system calls, especially during error or exception handling. This includes exceptions such as division by 0.

MULTIPLE INTERRUPT HANDLING:

When more than one device raises an interrupt request signal, then additional information is needed to decide 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.

- **Polling:**
    - In polling, the first device encountered 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.

- **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.

- **Interrupt Nesting:**
    - In this method, I/O device is organized in a priority structure. Therefore, interrupt request from a higher priority device is recognized whereas 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.

CONTROL UNIT

Control Unit is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.

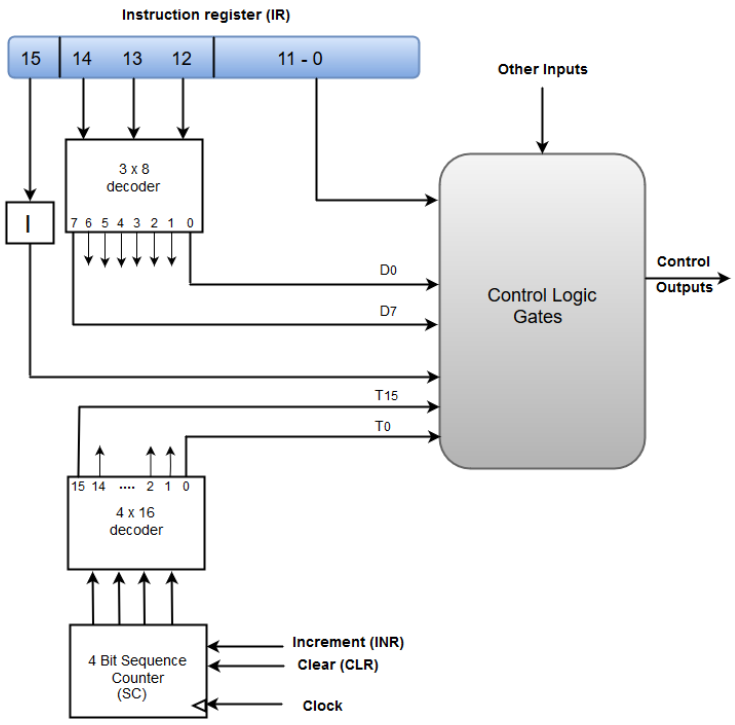
A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer.

- It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
- It interprets instructions.
- It controls data flow inside the processor.
- It receives external instructions or commands to which it converts to sequence of control signals.
- It controls many execution units (i.e. ALU, data buffers and registers) contained within a CPU.
- It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.

HARDWIRED CONTROL UNIT

The Hardwired Control organization involves the control logic to be implemented with gates, flip-flops, decoders, and other digital circuits. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode.

As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for executive units of the computer.

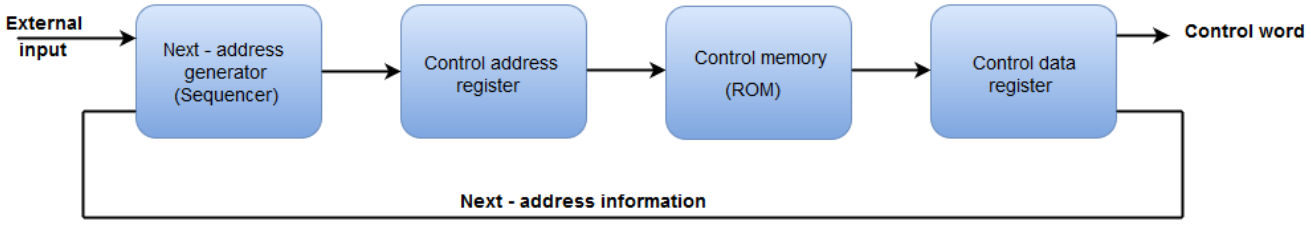


- A Hard-wired Control consists of two decoders, a sequence counter, and a number of logic gates.
- An instruction fetched from the memory unit is placed in the instruction register (IR).
- The component of an instruction register includes; I bit, the operation code, and bits 0 through 11.
- The operation code in bits 12 through 14 are coded with a 3 x 8 decoder.
- The outputs of the decoder are designated by the symbols D0 through D7.
- The operation code at bit 15 is transferred to a flip-flop designated by the symbol I.
- The operation codes from Bits 0 through 11 are applied to the control logic gates.
- The Sequence counter (SC) can count in binary from 0 through 15.
-

MICRO-PROGRAMMED CONTROL UNIT

The Microprogrammed Control organization is implemented by using the programming approach. In Microprogrammed Control, the micro-operations are performed by executing a program consisting of micro-instructions.

Microprogrammed Control Unit of a Basic Computer:

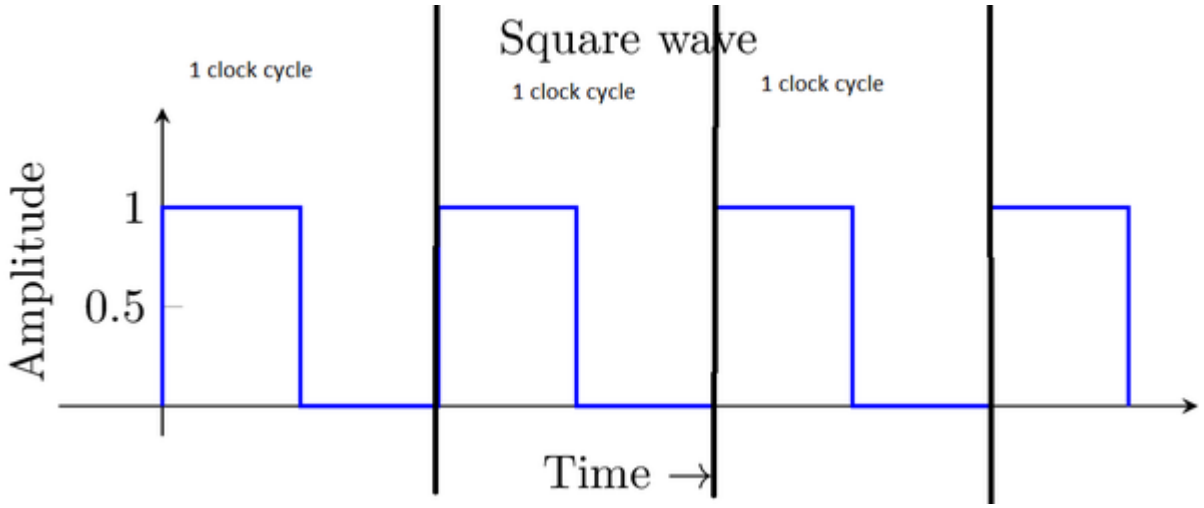


- The Control memory address register specifies the address of the micro-instruction.
- The Control memory is assumed to be a ROM, within which all control information is permanently stored.
- The control register holds the microinstruction fetched from the memory.
- The micro-instruction contains a control word that specifies one or more micro-operations for the data processor.
- While the micro-operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.
- The next address generator is often referred to as a micro-program sequencer, as it determines the address sequence that is read from control memory.

CLOCK

The clock is the timing device that regulates the movements of data throughout the machine, making it a crucial part in designing a CPU. The cycle is a complete action of the clock. It produces a signal by switching on a voltage and then switching it off again, very quickly. One switch on and one switch off is a complete cycle.

Virtually every CPU ever designed is driven by what is called a clock. Basically, a clock is a small circuit that generates a square wave signal: i.e. it generates a fixed length pulse at fixed intervals.



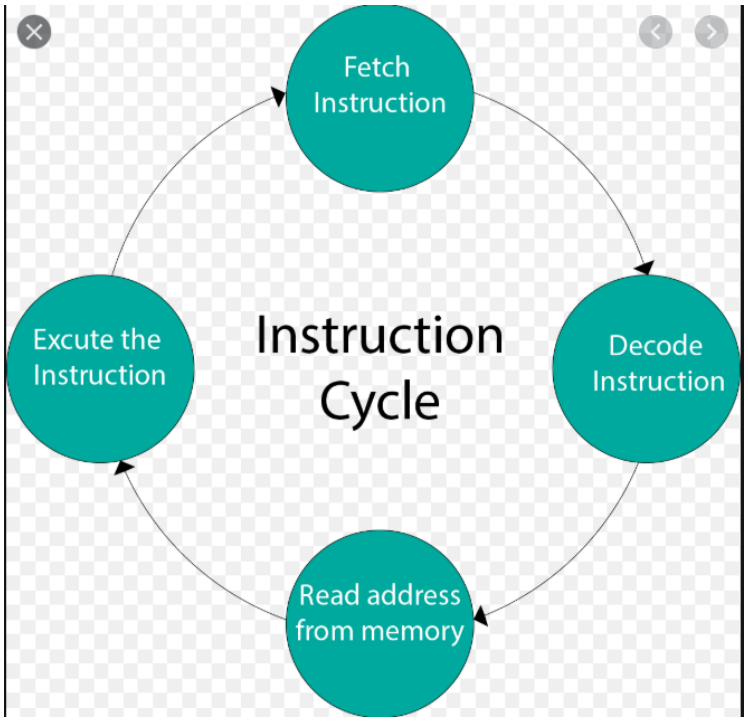
Getting the Clock right is key to determining the speed at which instructions are processed and overall speed of the CPU.

INSTRUCTION CYCLE

A program residing in the memory unit of a computer consists of a sequence of instructions. These instructions are executed by the processor by going through a cycle for each instruction.

In a basic computer, each instruction cycle consists of the following phases:

- Fetch instruction from memory.
- Decode the instruction.
- Read the effective address from memory.
- Execute the instruction.



SYSTEM BUS

A bus is a subsystem that is used to connect computer components and transfer data between them. A bus may be parallel or serial. Parallel buses transmit data across multiple wires. Serial buses transmit data in bit-serial format.

Bus is a communication channel. Characteristic of bus is shared transmission media. Limitation of a bus is only one transmission at a time. A bus which is used to provide the communication between the major components of a computer is called as System bus.

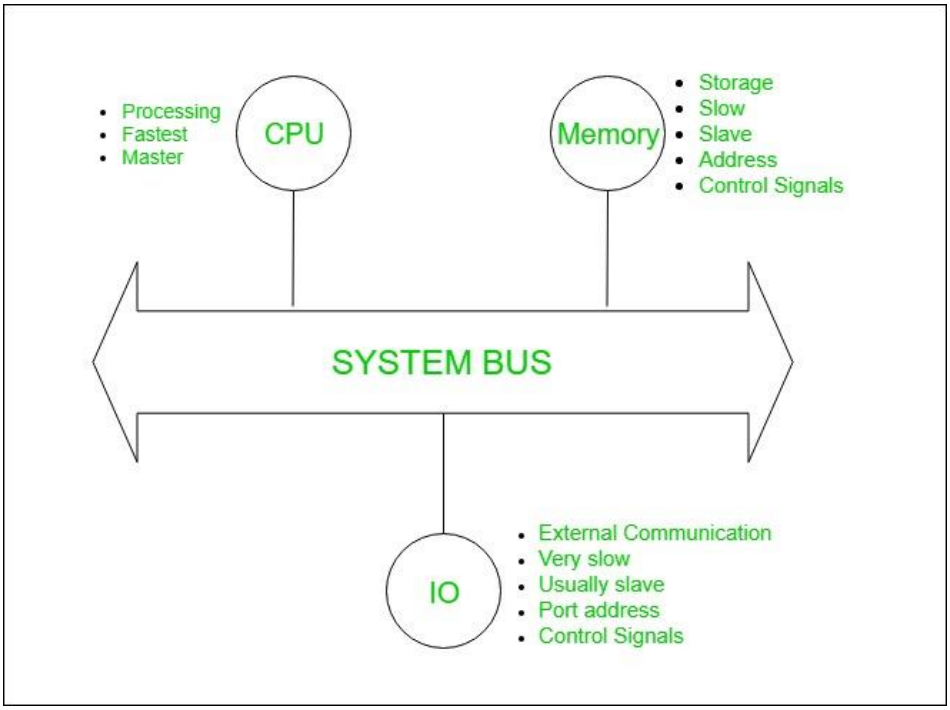


FIG: System Bus

- **Address bus**
  - Used to carry the address to memory ad IO.
  - Unidirectional.
  - Based on width of a address bus we can determine the capacity of a main memory.

- **Data bus**
  - Used to carry the binary data between the CPU, memory and IO.
  - Bidirectional.
  - Based on the width of a data bus we can determine the word length of a CPU.
  - Based on the word length we can determine the performance of a CPU.

- **Control bus**
  - Used to carry the control signals and timing signals
  - Control signals indicates type of operation.
  - Timing Signals used to synchronize the memory and IO operations with a CPU clock.

PIPELINING

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end. Pipelining increases the overall instruction throughput.

In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.

Let us see a real-life example that works on the concept of pipelined operation. Consider a water bottle packaging plant. Let there be 3 stages that a bottle should pass through, Inserting the bottle(I), Filling water in the bottle(F), and Sealing the bottle(S). Let us consider these stages as stage 1, stage 2 and stage 3 respectively. Let each stage take 1 minute to complete its operation.

Now, in a non-pipelined operation, a bottle is first inserted in the plant, after 1 minute it is moved to stage 2 where water is filled. Now, in stage 1 nothing is happening. Similarly, when

the bottle moves to stage 3, both stage 1 and stage 2 are idle. But in pipelined operation, when the bottle is in stage 2, another bottle can be loaded at stage 1. Similarly, when the bottle is in stage 3, there can be one bottle each in stage 1 and stage 2. So, after each minute, we get a new bottle at the end of stage 3. Hence, the average time taken to manufacture 1 bottle is:

Without pipelining = 9/3 minutes = 3m

```
I F S | | | | | |
| | | I F S | | |
| | | | | | I F S (9 minutes)
```

With pipelining = 5/3 minutes = 1.67m

```
I F S | |
| I F S |
| | I F S (5 minutes)
```

Thus, pipelined operation increases the efficiency of a system.

Design of a basic pipeline

- In a pipelined processor, a pipeline has two ends, the input end and the output end. Between these ends, there are multiple stages/segments such that output of one stage is connected to input of next stage and each stage performs a specific operation.
- Interface registers are used to hold the intermediate output between two stages. These interface registers are also called latch or buffer.
- All the stages in the pipeline along with the interface registers are controlled by a common clock.

Pipeline Stages

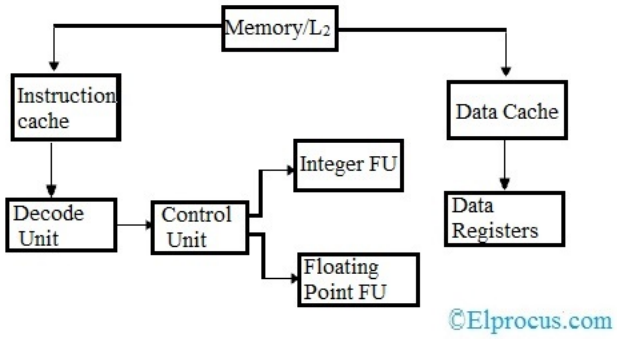
RISC processor has 5 stage instruction pipeline to execute all the instructions in the RISC instruction set. Following are the 5 stages of RISC pipeline with their respective operations:

- **Stage 1 (Instruction Fetch)**
  - In this stage the CPU reads instructions from the address in the memory whose value is present in the program counter.
- **Stage 2 (Instruction Decode)**
  - In this stage, instruction is decoded and the register file is accessed to get the values from the registers used in the instruction.

- **Stage 3 (Instruction Execute)**
  - In this stage, ALU operations are performed.
- **Stage 4 (Memory Access)**
  - In this stage, memory operands are read and written from/to the memory that is present in the instruction.
- **Stage 5 (Write Back)**
  - In this stage, computed/fetched value is written back to the register present in the instructions.

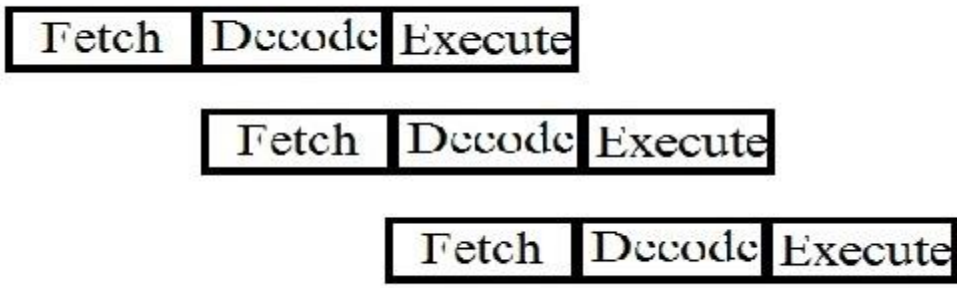
PIPELINING ARCHITECTURE

Parallelism can be achieved with Hardware, Compiler, and software techniques. To exploit the concept of pipelining in computer architecture many processor units are interconnected and are functioned concurrently. In pipelined processor architecture, there are separated processing units provided for integers and floating point instructions. Whereas in sequential architecture, a single functional unit is provided.



Pipelining in RISC Processors

The most popular RISC architecture ARM processor follows 3-stage and 5-stage pipelining. In 3-stage pipelining the stages are: Fetch, Decode, and Execute. This pipelining has 3 cycles latency, as an individual instruction takes 3 clock cycles to complete.



This model is used to design efficient CPUs to reduce instruction cycles required to get tasks done. Hence Pipelining proves to be an efficient and productive feature that is crucial in the design of a good CPU.

Instruction Set Architecture

When designing a CPU, the complexity of instruction set determines the type of processor and what tasks it can handle with level of efficiency. is part of a computer that pertains to programming, which is basically machine language. The instruction set provides commands to the processor, to tell it what it needs to do. The instruction set consists of addressing modes, instructions, native data types, registers, memory architecture, interrupt, and exception handling, and external I/O.

ISA determines the programmer's overview of designing the final CPU, rather than the hardware part we have already mentioned above in bulk. ISA determines whether a processor is RISC (Reduced Instruction Set Computer) or CISC (Complex ISC). RISCs have smaller and simpler instruction set whereas CISCs have more complex instruction sets.

MIPS (Microprocessor without Interlocked Pipelined Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA). Computer architecture courses in universities and technical schools often study the MIPS architecture. The architecture greatly influenced later RISC architectures such as Alpha.

ENERGY and POWER CONSUMPTION

When designing a CPU, energy and power consumed is also an important factor to determining efficiency in design. Bottlenecks in this area can cause long term damages to the core system itself, and can cause a nuisance when tested to the clock.

The following factors must be kept in mind while ensuring perfection in energy and power consumption:

- Maximum power requirement
  - If system draws more power than what the power supply can provide, voltage drop is imminent, leading to system malfunction
  - Modern CPUs hence use Voltage Indexing methods that allow processors to slow down and regulate voltage in case of high peak currents.
- Sustained power consumption
  - A cooling system is usually in place when TDP (Thermal Design Power) is exceeded.
  - Failure to cool adequately will allow junction temperature in processor to exceed max safe value, resulting in device failure.

- Hence for the design of an efficient CPU, Thermal Overload Traps must be set (various levels), so that CPU cock rate is reduced to facilitate reduced power dissipation.

CONCLUSION

CPU design has evolved though many phases of trials and research and these design paradigms brought in by decades of technological evolution can now finally ensure the making of a CPU efficient and productive enough to cater to the demands of the complex 21st century problems, be it social, economic, scientific, technological, ethical, etc.

This is how the modern day firms develop a well-built CPU considering all the above mentioned features and design mechanics.