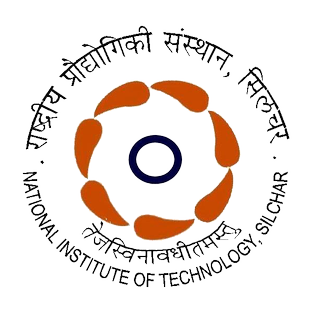
**ASSIGNMENT III**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

ASSAM-788010, INDIA

COMPUTER ARCHITECTURE AND ORGANISATION



**SUBMITTED BY:**  **SUBMITTED TO:**

NAME: Tanveer ahmed Dr. Malaya Dutta Borah

SCHOLAR ID: 1815010

sec:a

* CPU DESIGNING

CPU design is divided into design of the following components:

1. Arithmetic and Logical Unit(data paths)
2. Data Storage and Registers
3. Input and output Architecture
4. Control unit

**I.ARITHMETIC AND LOGICAL UNIT**

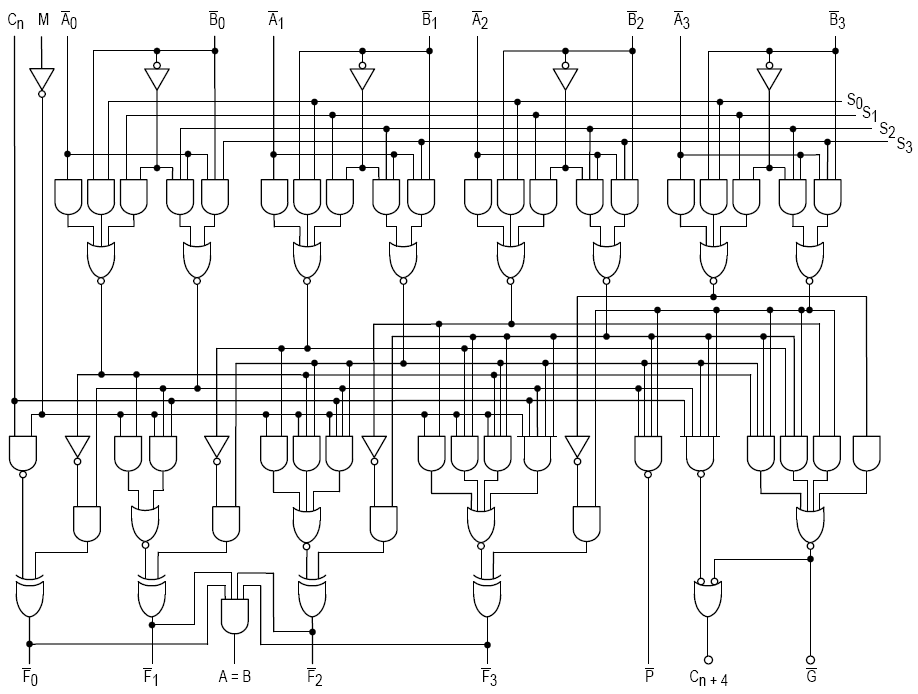
**a.ALU(Arithmetic Logic Unit)**

An ALU is a combinatonal circuit, meaning that its outputs will change asynchronously in response to input changes. In normal operation, stable signals are applied to all of the ALU inputs and, when enough time (known as the "propagation delay") has passed for the signals to propagate through the ALU circuitry, the result of the ALU operation appears at the ALU outputs. The external circuitry connected to the ALU is responsible for ensuring the stability of ALU input signals throughout the operation, and for allowing sufficient time for the signals to propagate through the ALU before sampling the ALU result.

External circuitry controls an ALU by applying signals to its inputs. Typically, the external circuitry employs sequential logic to control the ALU operation, which is paced by a clock signal of a sufficiently low frequency to ensure enough time for the ALU outputs to settle under worst-case conditions.

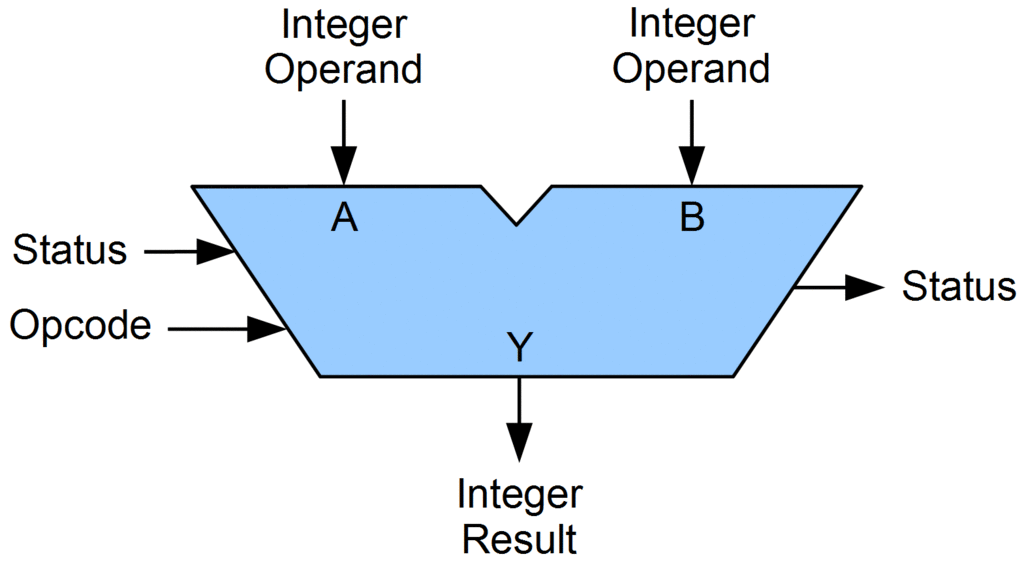
CPU begins an ALU addition operation by routing operands from their sources (which are usually registers) to the ALU's operand inputs, while the [control unit](https://en.wikipedia.org/wiki/Control_unit) simultaneously applies a value to the ALU's opcode input, configuring it to perform addition. At the same time, the CPU also routes the ALU result output to a destination register that will receive the sum. The ALU's input signals, which are held stable until the next clock, are allowed to propagate through the ALU and to the destination register while the CPU waits for the next clock. When the next clock arrives, the destination register stores the ALU result and, since the ALU operation has completed, the ALU inputs may be set up for the next ALU operation.

The following diagram gives the combinational logic circuitry of the 74181 integrated circuit,which is a simple 4- bit ALU



**Symbolic representation** of an ALU and its input and output signals indicated by arrows pointing into or out

of the ALU.



**Different functions of the ALU:**

**1**.**Arithmetic operations**

* ***Add***: A and B are summed and the sum appears at Y and carry-out.
* ***Add with carry*:** A, B and carry-in are summed and the sum appears at Y and carry-out.
* ***Subtract***: B is subtracted from A (or vice versa) and the difference appears at Y and carry-out. For this function, carry-out is effectively a "borrow" indicator. This operation may also be used to compare the magnitudes of A and B; in such cases the Y output may be ignored by the processor, which is only interested in the status bits (particularly zero and negative) that result from the operation.
* ***Subtract with borrow*:** B is subtracted from A (or vice versa) with borrow (carry-in) and the difference appears at Y and carry-out (borrow out).
* ***Two's complement (negate)*:** A (or B) is subtracted from zero and the difference appears at Y.
* ***Increment*:** A (or B) is increased by one and the resulting value appears at Y.
* ***Decrement***: A (or B) is decreased by one and the resulting value appears at Y.
* ***Pass through***: all bits of A (or B) appear unmodified at Y. This operation is typically used to determine the parity of the operand or whether it is zero or negative, or to load the operand into a processor register

**2.Bitwise logical operations**

LEFT

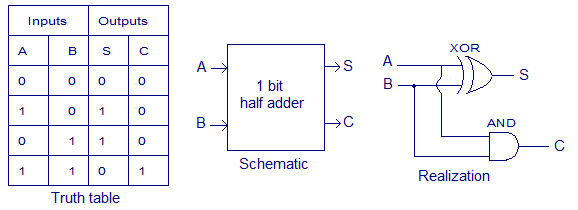
|  |  |  |
| --- | --- | --- |
| **TYPE** | **LEFT** | **RIGHT** |
| **Arithmetic shift** | [Rotate left logically.svg](https://en.wikipedia.org/wiki/File:Rotate_left_logically.svg) | [Rotate right arithmetically.svg](https://en.wikipedia.org/wiki/File:Rotate_right_arithmetically.svg) |
| **Logical shift** | [Rotate left logically.svg](https://en.wikipedia.org/wiki/File:Rotate_left_logically.svg) | [Rotate right logically.svg](https://en.wikipedia.org/wiki/File:Rotate_right_logically.svg) |
| **Rotate** | [Rotate left.svg](https://en.wikipedia.org/wiki/File:Rotate_left.svg) | [Rotate right.svg](https://en.wikipedia.org/wiki/File:Rotate_right.svg) |
| **Rotate through carry** | [Rotate left through carry.svg](https://en.wikipedia.org/wiki/File:Rotate_left_through_carry.svg) | [Rotate right through carry.svg](https://en.wikipedia.org/wiki/File:Rotate_right_through_carry.svg) |

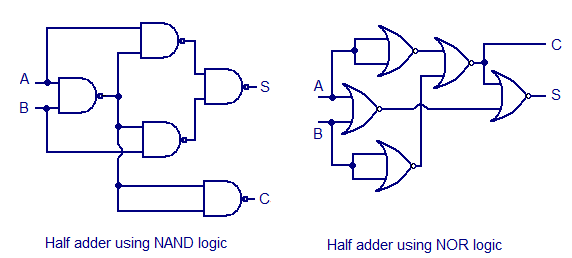
**THE ARITHMETIC UNIT OF ALU**

* **ADDER: An adder is a circuit that is used to add two binary numbers.**

**a. Half adder:**

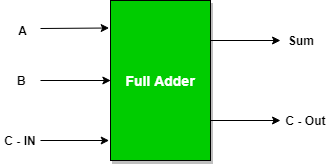
Half adder is a combinational arithmetic circuit that adds two numbers and produces a sum bit (S) and carry bit (C) as the output. If A and B are the input bits, then sum bit (S) is the X-OR of A and B and the carry bit (C) will be the AND of A and B. From this it is clear that a half adder circuit can be easily constructed using one X-OR gate and one AND gate. Half adder is the simplest of all adder circuit, but it has a major disadvantage. The half adder can add only two input bits (A and B) and has nothing to do with the carry if there is any in the input. So, if the input to a half adder have a carry, then it will be neglected it and adds only the A and B bits. That means the binary addition process is not complete and that’s why it is called a half adder.



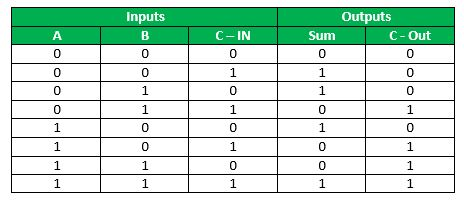


**b.Full Adder:**

Full-adder has three inputs and 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. When a full adder logic is designed, we will be able to string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.



Full adder truth table :



Logical Expression for SUM:

= A’ B’ C-IN + A’ B C-IN’ + A B’ C-IN’ + A B C-IN

= C-IN (A’ B’ + A B) + C-IN’ (A’ B + A B’)

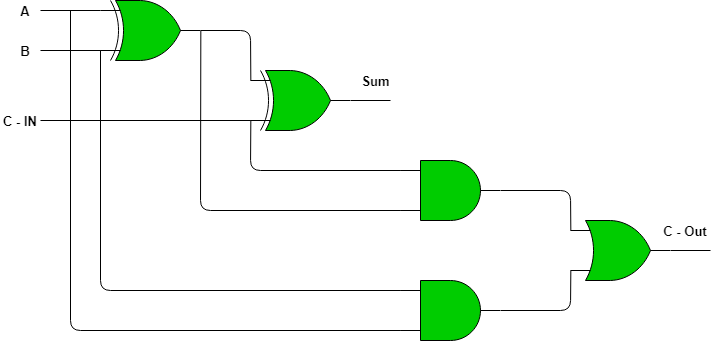
= C-IN XOR (A XOR B)

Logical Expression for C-OUT:

= A’ B C-IN + A B’ C-IN + A B C-IN’ + A B C-IN

= A B + B C-IN + A C-IN

Its logic circuit is given below:



* **MULTIPLE BIT ADDERS**

**a.Ripple carry adder**

It is possible to create a logical circuit using multiple full adders to add N-bit numbers. Each full adder inputs a Cin, which is the Cout of the previous adder. This kind of adder is a ripple carry adder, since each carry bit "ripples" to the next full adder. Note that the first (and only the first) full adder may be replaced by a half adder.

The layout of ripple carry adder is simple, which allows for fast design time; however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder.

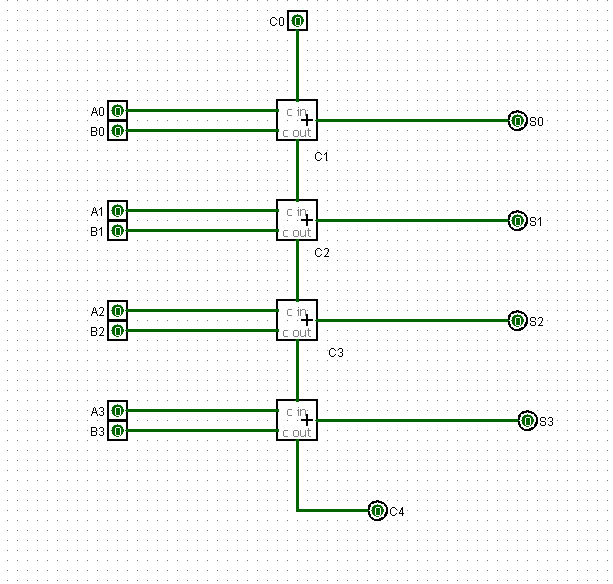
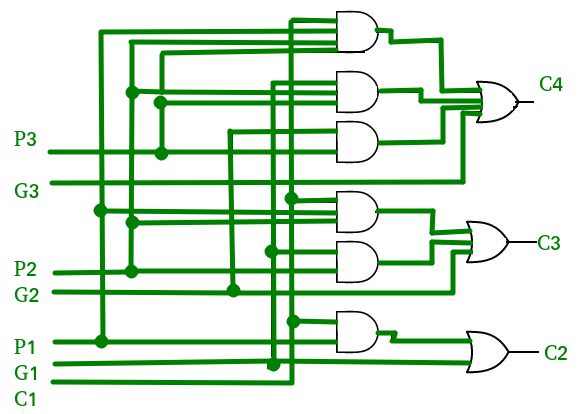


Fig: 4 bit ripple carry adder

**b.carry look ahead adder**

A carry-lookahead adder (CLA) or fast adder is a type of electronics adder used in digital logic. A carry-lookahead adder improves speed by reducing the amount of time required to determine carry bits. It can be contrasted with the simpler, but usually slower, ripple-carry adder (RCA), for which the carry bit is calculated alongside the sum bit, and each stage must wait until the previous carry bit has been calculated to begin calculating its own sum bit and carry bit. The carry-lookahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger-value bits of the adder. The Kogge–Stone adder (KSA) and Brent–Kung adder (BKA) are examples of this type of adder.



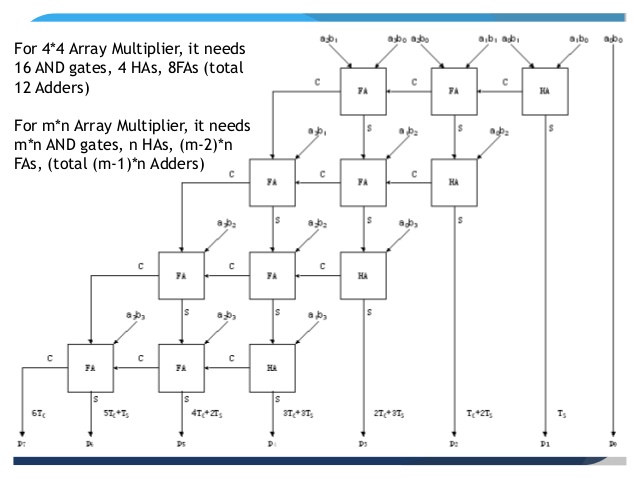
­­Here Pi=Ai xor Bi , Gi=AiBi ;

Ci+1=Gi+PICi,Si=Pi xor Ci where Si is the sum

* **MULTIPLIERS**

**1. Array Multipliers**

* An array multiplier is a digital [combinational circuit](https://www.geeksforgeeks.org/construction-of-combinational-circuits/) used for multiplying two binary numbers by employing an array of full adders and half adders. This array is used for the nearly simultaneous addition of the various product terms involved. To form the various product terms, an array of AND gates is used before the Adder array.
* Checking the bits of the multiplier one at a time and forming partial products is a sequential operation that requires a sequence of add and shift micro-operations. The multiplication of two binary numbers can be done with one micro-operation by means of a combinational circuit that forms the product bits all at once. This is a fast way of multiplying two numbers since all it takes is the time for the signals to propagate through the gates that form the multiplication array. However, an array multiplier requires a large number of gates, and for this reason it was not economical until the development of integrated circuits.
* For implementation of array multiplier with a combinational circuit, consider the multiplication of two 2-bit numbers as shown in figure. The multiplicand bits are b1 and b0, the multiplier bits are a1 and a0, and the product is c3c2c1c0.



**2.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 advantage 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.

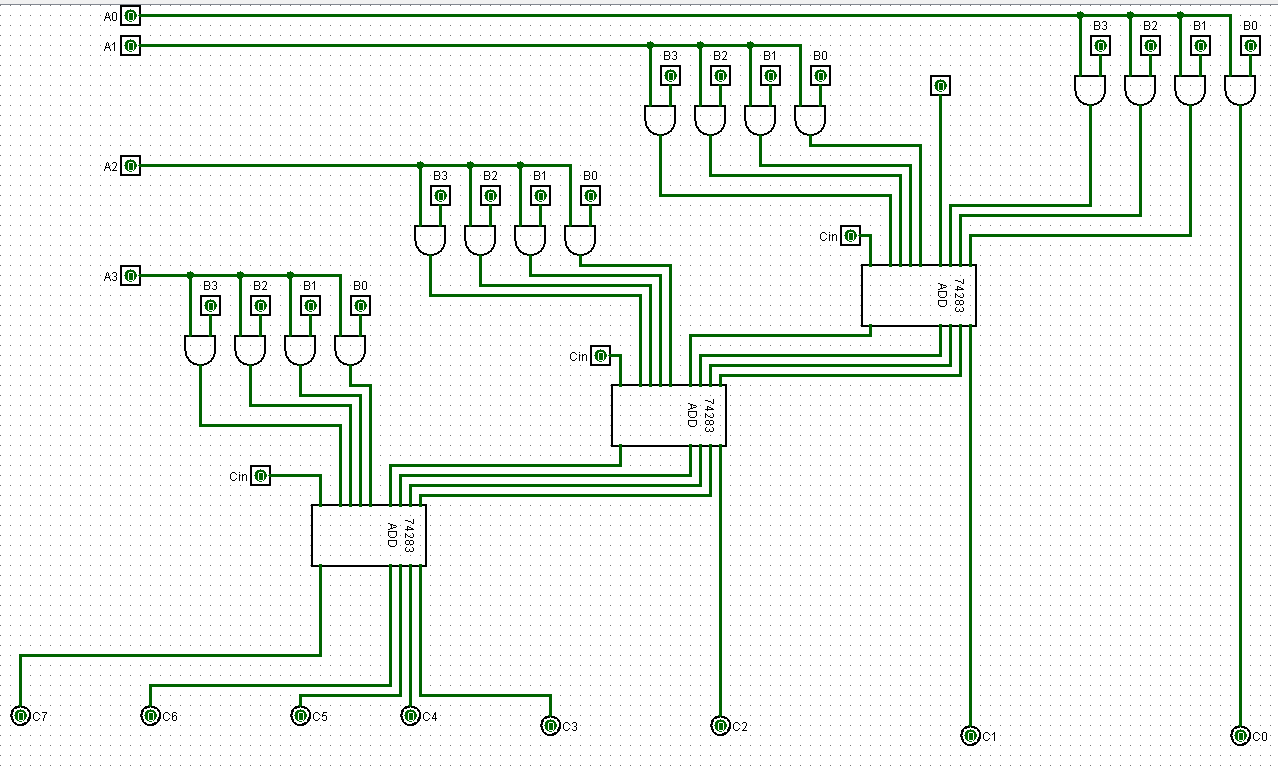


FIG:4-BIT combinational multiplier

**3.Booth’s multiplier**

Booth's algorithm can be implemented in many ways. This experiment is designed using a controller and a datapath. The operations on the data in the datapath is controlled by the control signal received from the controller. The datapath contains registers to hold multiplier, multiplicand, intermediate results, data processing units like ALU, adder/subtractor etc., counter and other combinational units. Following is the schematic diagram of the Booth's multiplier which multiplies two 4-bit numbers in 2's complement of this experiment. Here the adder/subtractor unit is used as data processing unit. M, Q, A are 4-bit and Q-1 is a 1-bit register. M holds the multiplicand, Q holds the multiplier, A holds the results of adder/subtractor unit. The counter is a down counter which counts the number of operations needed for the multiplication. The data flow in the data path is controlled by the five control signals generated from the controller. these signals are load (to load data in registers), add (to initiate addition operation), sub (to initiate subtraction operation), shift (to initiate arithmetic right shift operation), dc (this is to decrement counter). The controller generates the control signals according to the input received from the datapath. Here the inputs are the least significant Q0 bit of Q register, Q-1 bit and count bit from the down counter.

**PIPELINING**

**Pipelining :** Pipelining is a process of arrangement of hardware elements of the CPU such that its overall performance is increased. Simultaneous execution of more than one instruction takes place in a pipelined processor

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.
* **Execution in a pipelined processor**

Execution sequence of instructions in a pipelined processor can be visualized using a space-time diagram. For example, consider a processor having 4 stages and let there be 2 instructions to be executed. We can visualize the execution sequence through the following space-time diagrams:

* **Non overlapped execution:**

| **STAGE / CYCLE** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S1 | I1 |  |  |  | I2 |  |  |  |
| S2 |  | I1 |  |  |  | I2 |  |  |
| S3 |  |  | I1 |  |  |  | I2 |  |
| S4 |  |  |  | I1 |  |  |  | I2 |

* Total time = 8 Cycle
* **Overlapped execution:**

| **STAGE / CYCLE** | **1** | **2** | **3** | **4** | **5** |
| --- | --- | --- | --- | --- | --- |
| S1 | I1 | I2 |  |  |  |
| S2 |  | I1 | I2 |  |  |
| S3 |  |  | I1 | I2 |  |
| S4 |  |  |  | I1 | I2 |

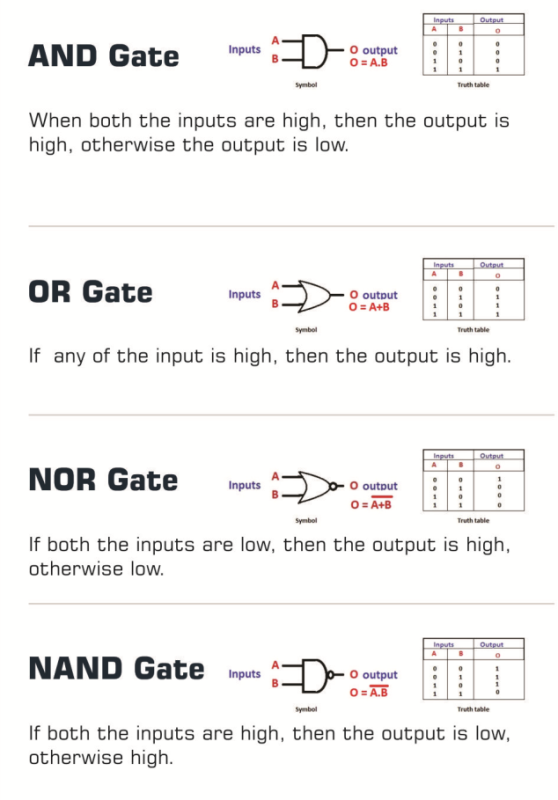
* Total time = 5 Cycle

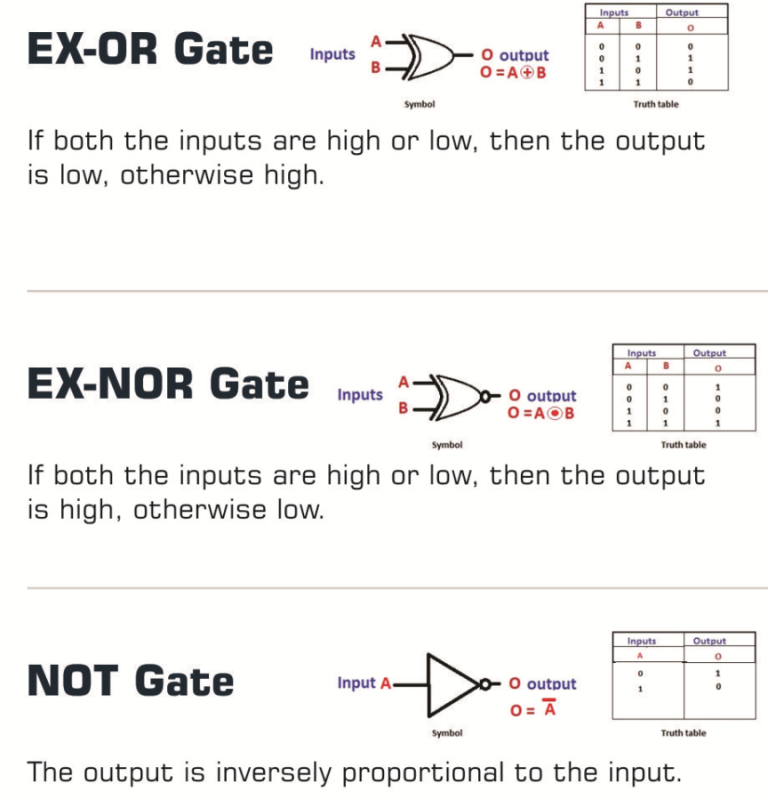
**Pipeline Stages**

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.

**II.LOGICAL UNIT OF THE ALU**





**II.Data Storage and CPU Registers**

A processor register is one of the small set of data holding places that are part of the computer processor.A register may hold an instruction ,a storage address pr any kind of data(such as a bit sequence or individual characters).some instructions specify registers as part of the instruction.For example an instruction may specify that the contents of two defined registers be added together and then placed in a specified register.

There are Different Types of Registers:



* The Memory unit has a capacity of 4096 words, and each word contains 16 bits.
* The Data Register (DR) contains 16 bits which hold the operand read from the memory location.
* The Memory Address Register (MAR) contains 12 bits which hold the address for the memory location.
* The Program Counter (PC) also contains 12 bits which hold the address of the next instruction to be read from memory after the current instruction is executed.
* The Accumulator (AC) register is a general purpose processing register.
* The instruction read from memory is placed in the Instruction register (IR).
* The Temporary Register (TR) is used for holding the temporary data during the processing.
* The Input Registers (IR) holds the input characters given by the user.
* The Output Registers (OR) holds the output after processing the input data.

**III.CPU INPUT OUTPUT(I/O) 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:**

Input or output devices that are connected to computer are called **peripheral devices**. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called **peripherals**.

For example: *Keyboards*, *display units* and *printers* are common peripheral devices.

There are three types of peripherals:

1. **Input peripherals** : Allows user input, from the outside world to the computer. Example: Keyboard, Mouse etc.
2. **Output peripherals**: Allows information output, from the computer to the outside world. Example: Printer, Monitor etc
3. **Input-Output peripherals**: Allows both input(from outised world to computer) as well as, output(from computer to the outside world). Example: Touch screen etc.

**Interfaces**

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

There are two types of interface:

1. CPU Inteface
2. I/O Interface

Let's understand the I/O Interface in details,

**Input-Output Interface**

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 Data Transfer**

Data transfer between the central unit and I/O devices can be handled in generally three types of modes which are given below:

1. Programmed I/O
2. Interrupt Initiated I/O
3. Direct Memory Access

**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.

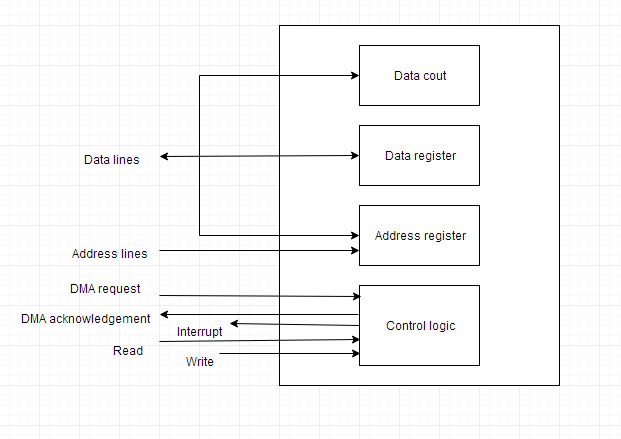


Fig:block diagram of DMA

INTERRUPTS IN CPU

Data transfer between the CPU and the peripherals is initiated by the CPU. But the CPU cannot start the transfer unless the peripheral is ready to communicate with the CPU. When a device is ready to communicate with the CPU, it generates an interrupt signal. A number of input-output devices are attached to the computer and each device is able to generate an interrupt request.

The main job of the interrupt system is to identify the source of the interrupt. There is also a possibility that several devices will request simultaneously for CPU communication. Then, the interrupt system has to decide which device is to be serviced first.

**Priority Interrupt**

A priority interrupt is a system which decides the priority at which various devices, which generates the interrupt signal at the same time, will be serviced by the CPU. The system has authority to decide which conditions are allowed to interrupt the CPU, while some other interrupt is being serviced. Generally, devices with high speed transfer such as *magnetic disks* are given high priority and slow devices such as *keyboards* are given low priority.

When two or more devices interrupt the computer simultaneously, the computer services the device with the higher priority first.

**Types of Interrupts:**

Following are some different types of interrupts:

Hardware Interrupts

When the signal for the processor is from an external device or hardware then this interrupts is known as **hardware interrupt**.

Let us consider an example: when we press any key on our keyboard to do some action, then this pressing of the key will generate an interrupt signal for the processor to perform certain action. Such an interrupt can be of two types:

* **Maskable Interrupt**

The hardware interrupts which can be delayed when a much high priority interrupt has occurred at the same time.

* **Non Maskable Interrupt**

The hardware interrupts which cannot be delayed and should be processed by the processor immediately.

**Software Interrupts**

The interrupt that is caused by any internal system of the computer system is known as a **software interrupt**. It can also be of two types:

* **Normal Interrupt**

The interrupts that are caused by software instructions are called **normal software interrupts**.

* **Exception**

Unplanned interrupts which are produced during the execution of some program are called **exceptions**, such as division by zero.

**Daisy Chaining Priority**

This way of deciding the interrupt priority consists of serial connection of all the devices which generates an interrupt signal. The device with the highest priority is placed at the first position followed by lower priority devices and the device which has lowest priority among all is placed at the last in the chain.

In daisy chaining system all the devices are connected in a serial form. The interrupt line request is common to all devices. If any device has interrupt signal in low level state then interrupt line goes to low level state and enables the interrupt input in the CPU. When there is no interrupt the interrupt line stays in high level state. The CPU respond to the interrupt by enabling the interrupt acknowledge line. This signal is received by the device 1 at its PI input. The acknowledge signal passes to next device through PO output only if device 1 is not requesting an interrupt.

The following figure shows the block diagram for daisy chaining priority system.

Priority Interrupts | (S/W Polling and Daisy Chaining) - GeeksforGeeks

Fig:Daisy chain priority Interrupt

**IV. CONTROL UNIT ARCHITECTURE**

**Design of Control Unit**

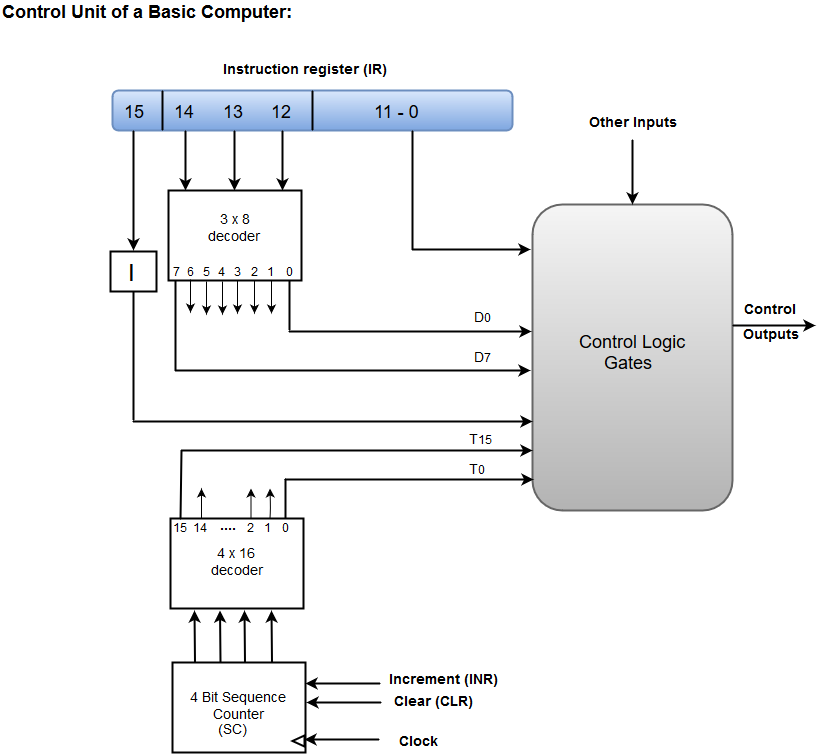
The Control Unit is classified into two major categories:

1. Hardwired Control
2. Microprogrammed Control

**Hardwired Control**

The Hardwired Control organization involves the control logic to be implemented with gates, flip-flops, decoders, and other digital circuits.

The following image shows the block diagram of a Hardwired Control organization.



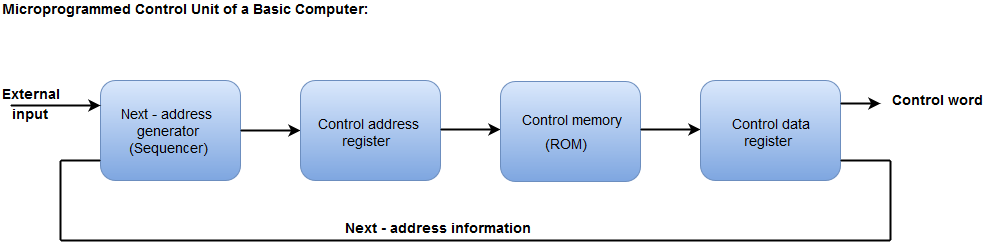
* A Hard-wired Control consists of two decoders, a sequence counter, and a number of logic gates.
* An instruction fetched from the memory unit is placed in the instruction register (IR).
* The component of an instruction register includes; I bit, the operation code, and bits 0 through 11.
* The operation 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**

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.

The following image shows the block diagram of a Microprogrammed Control organization.



* 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.

**ENERGY AND POWER OF A CPU**

To design a cpu it is important to understanf the specifications of the computer from various levels of abstraction.this is because we need to understand the cost,performance,power consumption etc that is essential for a CPU to work.

Some concerns are:

1.power must be brought in and distributed throughout the chip

2.power is dissipated as head and that need to be removed

A standard CPU consumes about 65-85 W of power.

A quad core processor consumes 95-140 W of power

e.g pentium4,core i7 20 core i7 950,core i7 965 etc

Thermal design power: it refers to the power consumption under max theoretical load

Energy is of two kinds:

Dynamic and static.

Dynamic:energy during ON state

Static:Energy during OFF state.

---------------------------------------x---------------------------------x------------------------------------------x----------------------------