COMPUTER ORGANISATION AND ARCHITECTURE

Designing a Simple CPU

MAY 2, 2020

SUBMITTED BY
Syed Tafreed Numan Scholar ID 1815043

Table of Contents:

I. Introduction
II. Arithmetic and Logical Unit
III. Data Storage and Registers
IV. Input Output Architecture
V. Control Unit

Introduction to Central Processing Unit (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.

The form, design, and implementation of CPUs have changed over the course of their history, but their fundamental operation remains almost unchanged. Principal components of a CPU include the arithmetic logic unit (ALU) that performs arithmetic and logic operations, processor registers that supply operands to the ALU and store the results of ALU operations, and a control unit that orchestrates the fetching (from memory) and execution of instructions by directing the coordinated operations of the ALU, registers and other components.

Most modern CPUs are microprocessors, where the CPU is contained on a single metal-oxide-semiconductor (MOS) integrated circuit (IC) chip. An IC that contains a CPU may also contain memory, peripheral interfaces, and other components of a computer; such integrated devices are variously called microcontrollers or systems on a chip (SoC). Some computers employ a multi-core processor, which is a single chip or "socket" containing two or more CPUs called "cores".

Array processors or vector processors have multiple processors that operate in parallel, with no unit considered central. Virtual CPUs are an abstraction of dynamical aggregated computational resources.

A Basic CPU can be divided to 4 parts

I. Arithmetic and Logical Unit
II. Data Storage and Registers
III. Input Output Architecture
IV. Control Unit
V. Energy and Power of a CPU

Arithmetic and Logical Unit (ALU)

Short for **arithmetic logic unit**, the **ALU** is a complex digital circuit; one of many components within a computer's central processing unit. It performs both bitwise and mathematical operations on binary numbers and is the last component to perform calculations in the processor. The ALU uses operands and code that tells it which operations to perform for input data. After the information is processed by the ALU, it's sent to the computer's memory.

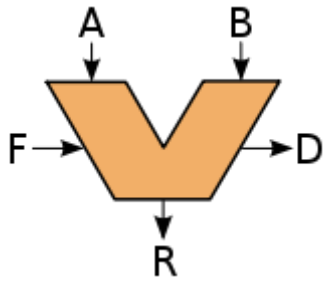


Figure 1: Typical representation of an ALU

A and B are the data words that are going to be used in the next calculation. F represents the input flags which are used for several things. For example, it lets the ALU know what the next instruction is and how to calculate the result (R).

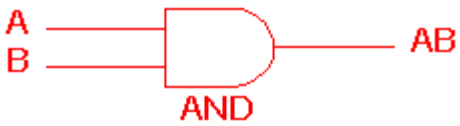
The output-flags (D) can be used by the application programmer. The ALU might have a line that gets high, when the last result was 0, or when it was negative. There might be more flags present and I'll use these for jumps in applications in my CPU. The ALU can be further divided into two parts AU-Arithmetic Unit and LU-Logical Unit.

1)Logical Unit (LU) and Logical Instructions

These include AND, OR, NOT, XOR, NOR, NAND, etc.

These Can be implemented by various gates for example

AND gate



2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

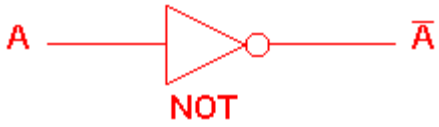
The AND gate is an electronic circuit that gives a **high** output (1) only if **all** its inputs are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is sometimes omitted i.e. AB

OR gate

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

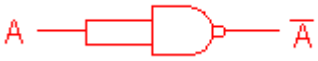
The OR gate is an electronic circuit that gives a high output (1) if **one or more** of its inputs are high. A plus (+) is used to show the OR operation.

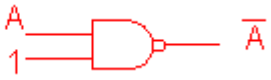
NOT gate



NOT gate	
A	$\overline{A}$
0	1
1	0

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an *inverter*. If the input variable is A, the inverted output is known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the outputs. The diagrams below show two ways that the NAND logic gate can be configured to produce a NOT gate. It can also be done using NOR logic gates in the same way.





NAND gate

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if **any** of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

NOR gate

2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

2)Arithmetic Unit (AU) and Arithmetic Operations

In this section, we will only look at different types of binary addition circuits and multiplication circuit for a simple ALU. Also, all the other simple arithmetic operations can be derived from them.

Adder

An adder is a kind of calculator that is used to add two binary numbers. When I say, calculator, I don't mean one with buttons, this one is a circuit that can be integrated with many other circuits for a wide range of applications. There are two kinds of adders;

Half Adder

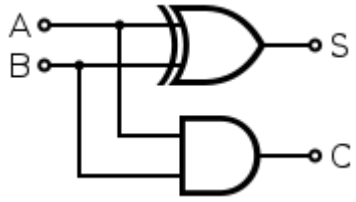
The Half Adder is a digital device used to add two binary bits 0 and 1 The half adder outputs a sum of the two inputs and a carry value.

0 + 0 = Sum 0 Carry 0
0 + 1 = Sum 1 Carry 0
1 + 0 = Sum 1 Carry 0
1 + 1 = Sum 0 Carry 1

TRUTH TABLE

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder can be constructed from AND gate and XOR gate as shown below

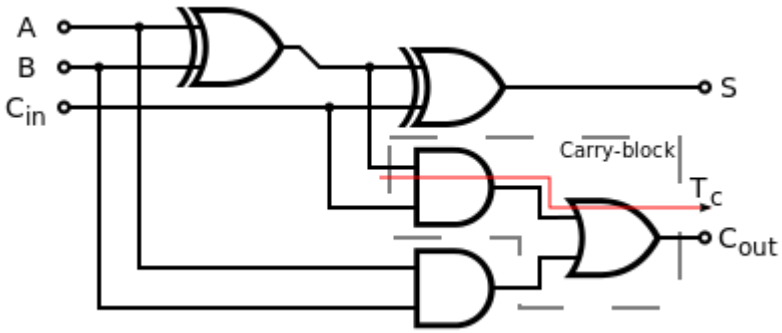


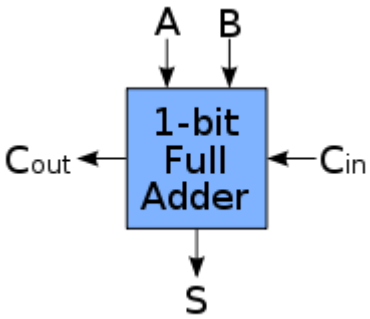
FULL ADDER

A **full adder** is a logical circuit that performs an addition operation on three one-bit binary numbers. The full adder produces a sum of the three inputs and carry value. It can be combined with other full adders (see below) or work on its own.

Truth Table:

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





Note that the final OR gate before the carry-out output may be replaced by an XOR gate without altering the resulting logic. This is because the only difference between OR and XOR gates occurs only when both inputs are 1; for the adder shown here, this is never possible. Using only two types of gates is convenient if one desires to implement the adder directly using common IC chips.

A full adder can be constructed from two half adders by connecting *A* and *B* to the input of one half adder, connecting the sum from that to an input to the second adder, connecting C_i to the other input and OR the two carry outputs. Equivalently, *S* could be made the three bit XOR of *A*, *B*, and C_i, and C_o could be made the three-bit majority function of *A*, *B*, and C_i.

Multiple Bit Adders

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 C_{in}, which is the C_{out} 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.

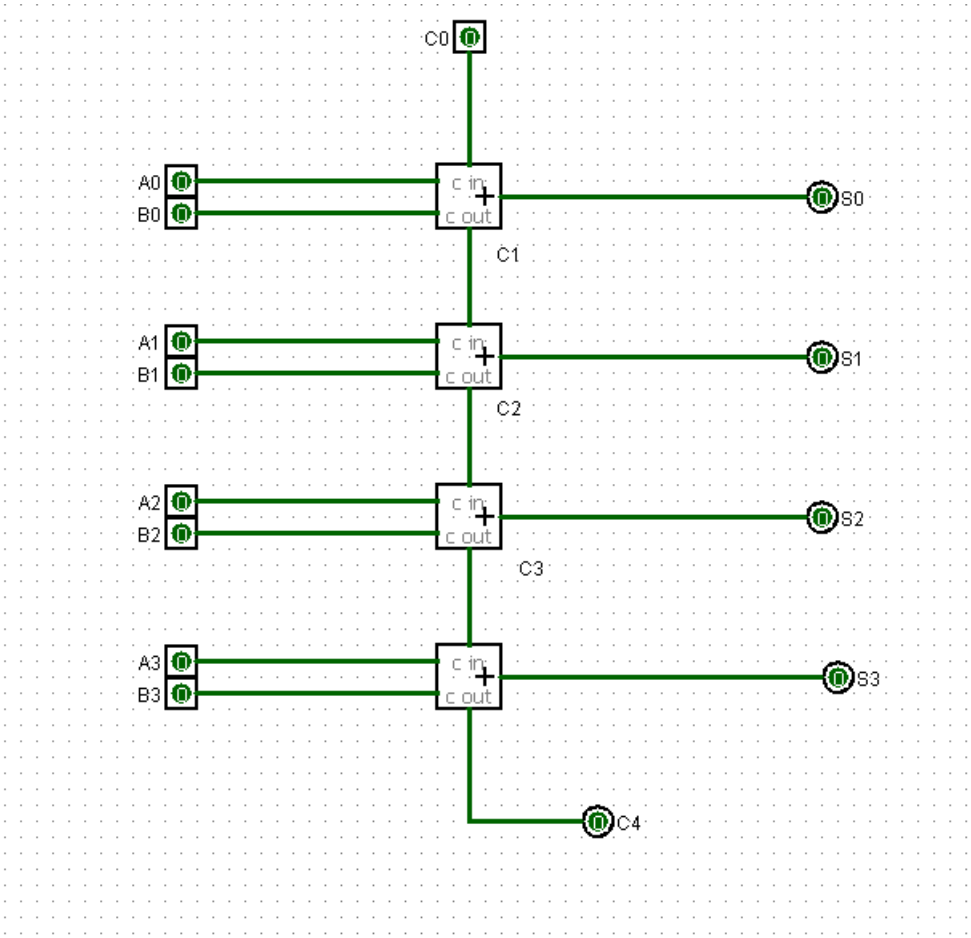


Figure 4-bit Ripple Carry Adder

Carry Look Ahead Adders

To reduce the computation time, engineers devised faster ways to add two binary numbers by using carry lookahead adders. They work by creating two signals (*P* and *G*) for each bit position, based on whether a carry is propagated through from a less significant bit position (at least one input is a '1'), a carry is generated in that bit position (both inputs are '1'), or if a carry is killed in that bit position (both inputs are '0'). In most cases, *P* is simply the sum output of a half-adder and *G* is the carry output of the same adder. After *P* and *G* are generated the carries for every bit position are created.

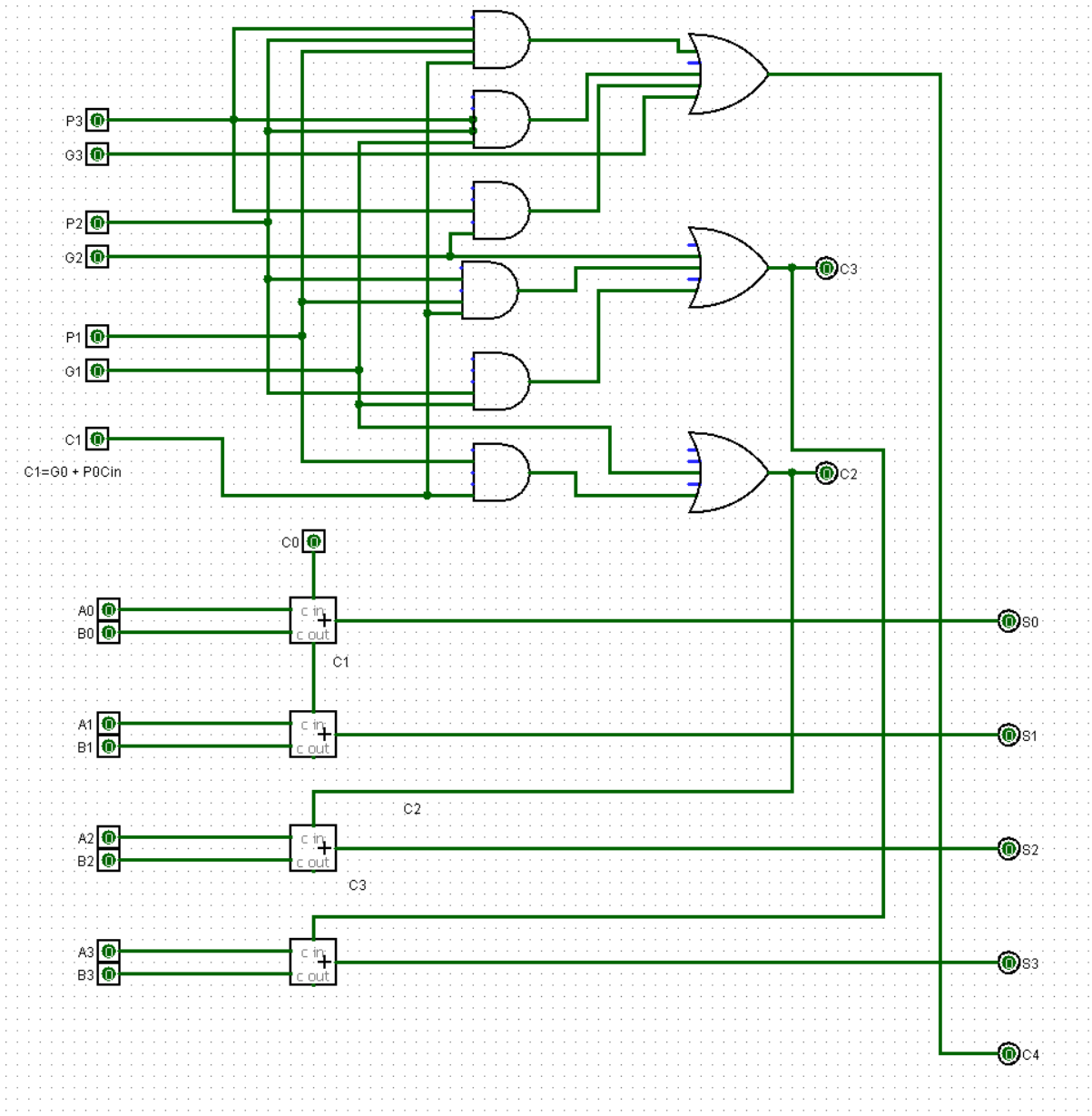


Figure – 4-bit Carry Look Ahead Adder

Some other multi-bit adder architectures break the adder into blocks. It is possible to vary the length of these blocks based on the propagation delay of the circuits to optimize computation time. These block based adders include the carry bypass adder which will determine *P* and *G* values for each block rather than each bit, and the carry select adder which pre-generates sum and carry values for either possible carry input to the block.

MULTIPLIERS

Array Multipliers

An **array multiplier** is a digital combinational circuit 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.

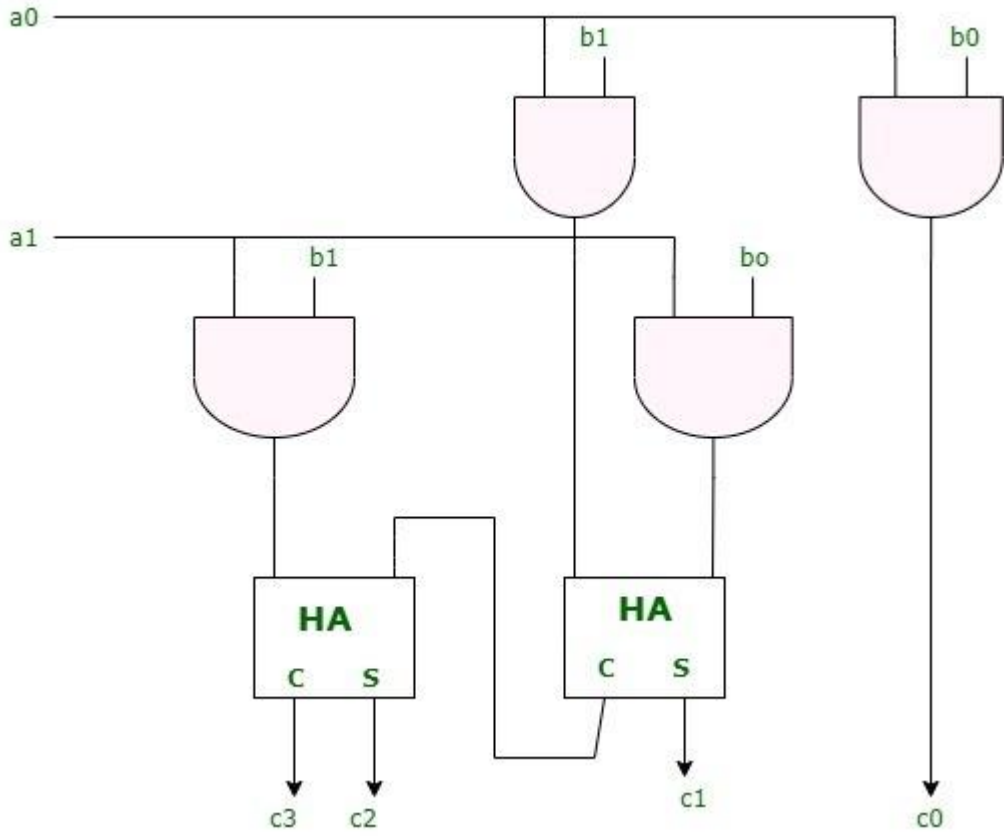


Figure – 2-bit Array Multiplier

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.

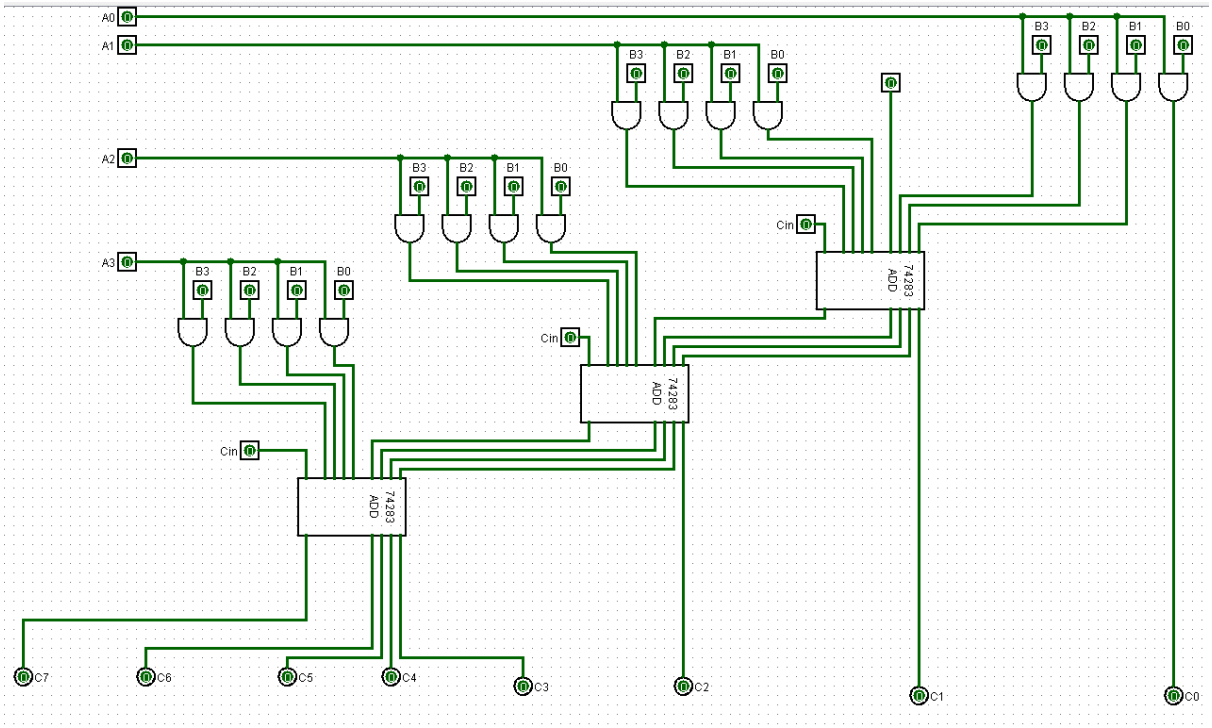
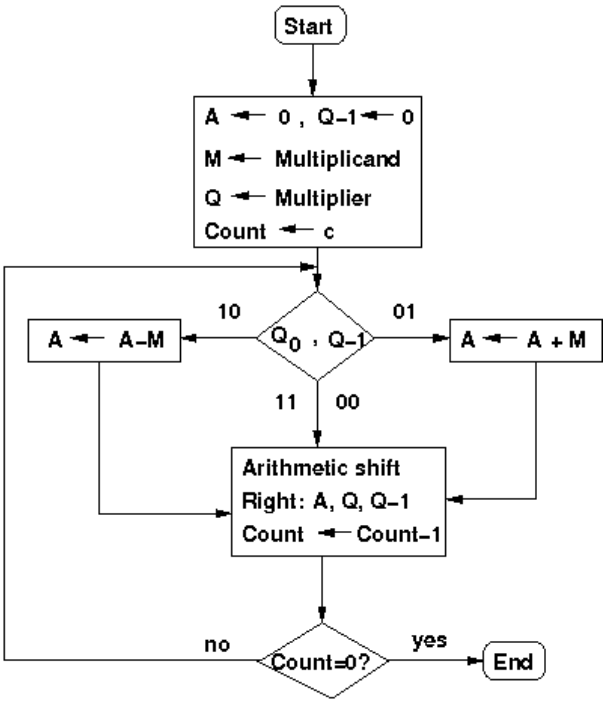


Figure – 4-Bit Combinational Multiplier.

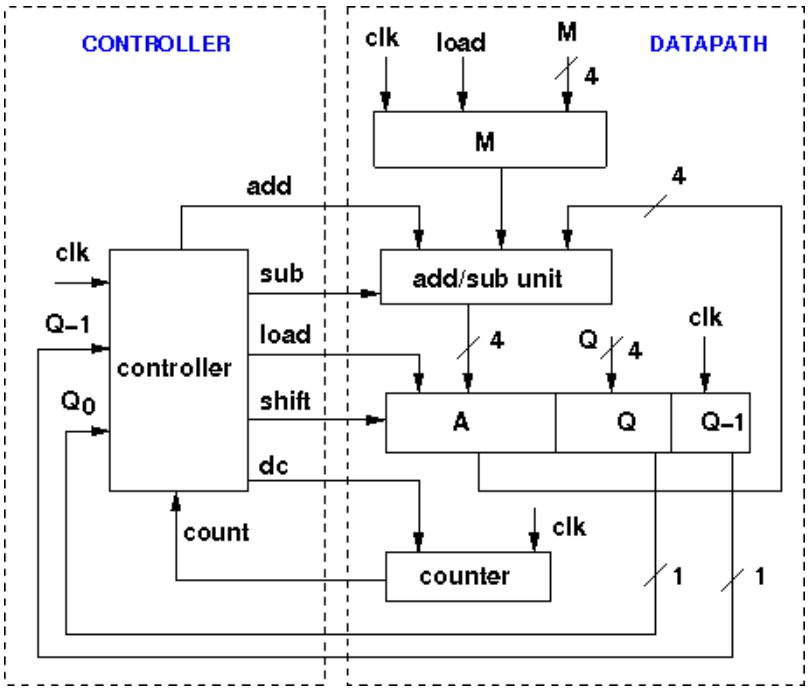
Booth's Multiplier:

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.



Construction:

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.



Data Storage and CPU Registers:

A processor register (CPU register) is one of a small set of data holding places that are part of the computer processor.

A register may hold an instruction, a storage address, or 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:

Register	Symbol	Function
Data register	DR	Holds memory operand
Address register	AR	Holds address for the memory
Accumulator	AC	Processor register
Instruction register	IR	Holds instruction code
Program counter	PC	Holds address of the instruction
Temporary register	TR	Holds temporary data
Input register	INPR	Carries input character
Output register	OUTR	Carries output character

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as Processor registers.

A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).

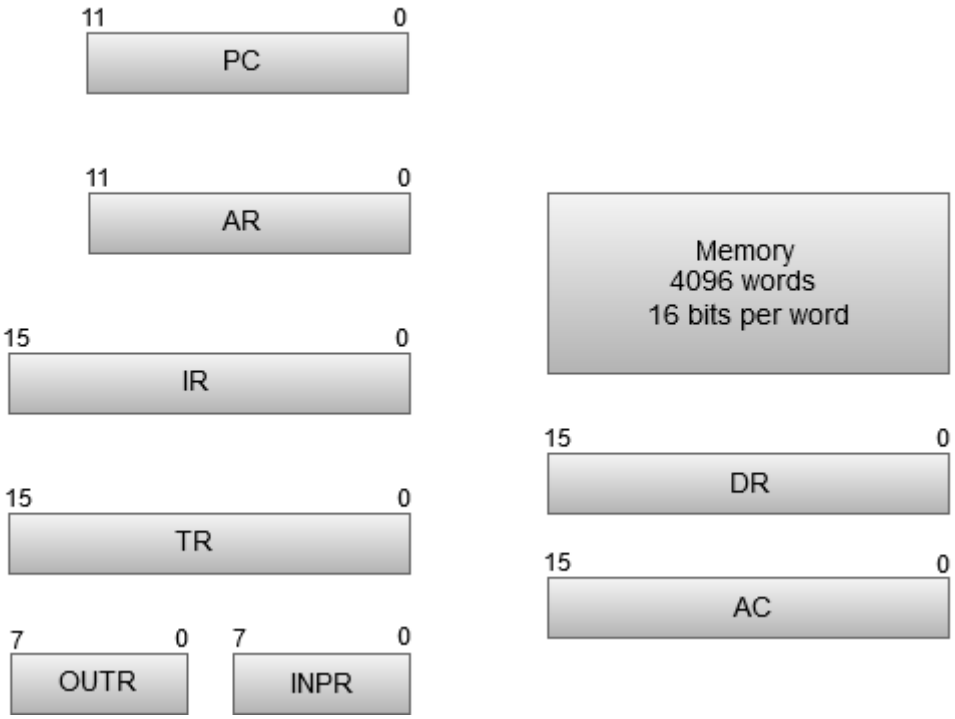
The computer needs processor registers for manipulating data and a register for holding a memory address. The register holding the memory location is used to

calculate the address of the next instruction after the execution of the current instruction is completed.

Following is the list of some of the most common registers used in a basic computer:

The following image shows the register and memory configuration for a basic computer.

Register and Memory Configuration of a basic computer:



- 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.

CPU Input Output (I/O) Architecture

An input-output part of processor (IOP) is a part with direct memory access capability. In this, the computer system is divided into a memory unit and number of processors.

Each IOP controls and manage the input-output tasks. The IOP is similar to CPU except that it handles only the details of I/O processing. The IOP can fetch and execute its own instructions. These IOP instructions are designed to manage I/O transfers only.

Various I/O (Peripheral) Devices are

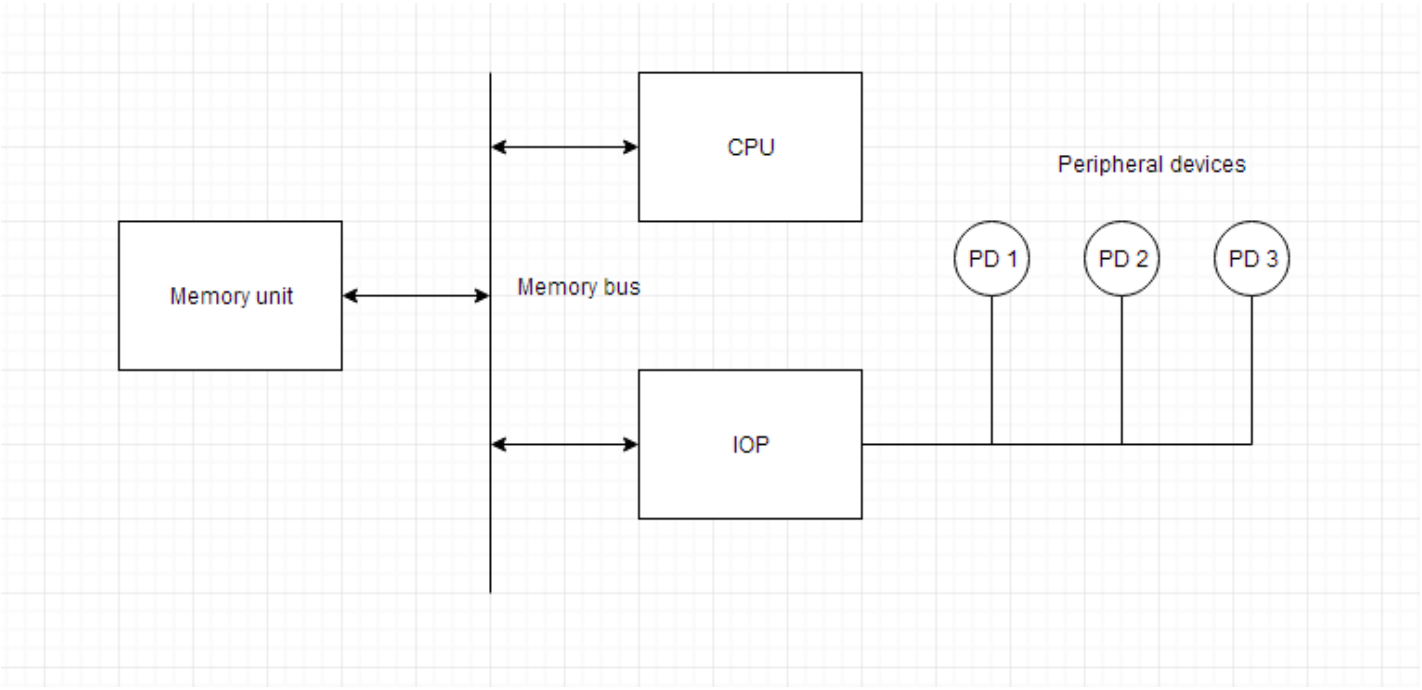
- Mouse
- Keyboard
- Monitor
- Touchpad
- CD/DVD Drive
- Printer

Block Diagram Of I/O Processor

Below is a block diagram of a computer along with various I/O Processors. The memory unit occupies the central position and can communicate with each processor.

The CPU processes the data required for solving the computational tasks. The IOP provides a path for transfer of data between peripherals and memory. The CPU assigns the task of initiating the I/O program.

The IOP operates independent from CPU and transfer data between peripherals and memory.



The communication between the IOP and the devices is similar to the program control method of transfer. And the communication with the memory is similar to the direct memory access method.

In large scale computers, each processor is independent of other processors and any processor can initiate the operation.

The CPU can act as master and the IOP act as slave processor. The CPU assigns the task of initiating operations but it is the IOP, who executes the instructions, and not the CPU. CPU instructions provide operations to start an I/O transfer. The IOP asks for CPU through interrupt.

Instructions that are read from memory by an IOP are also called commands to distinguish them from instructions that are read by CPU. Commands are prepared by programmers and are stored in memory. Command words make the program for IOP. CPU informs the IOP where to find the commands in memory.

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.

The Input Output Architecture is driven by Interrupts coming from the Input/Output(I/O) Peripheral Devices. Eg:

Click of a Mouse,

Pressing of a Keyboard Button,

Insertion of a USB Drive etc

Interrupt

An 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**.

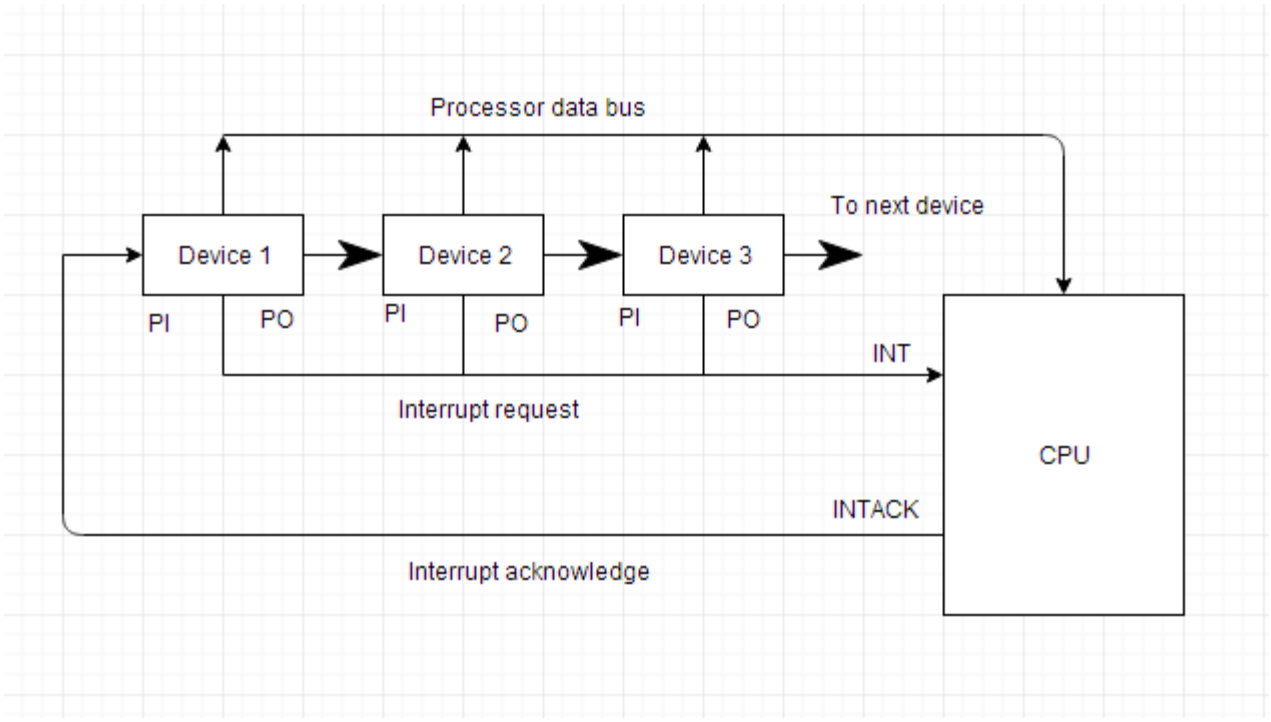
- **Exception**
  Unplanned interrupts which are produced during the execution of some program are called **exceptions**, such as division by zero.

Daisy Chaining Interrupts

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.


Processor data bus
To next device
Device 1
Device 2
Device 3
PI
PO
PI
PO
PI
PO
INT
Interrupt request
CPU
INTACK
Interrupt acknowledge

Control Unit Architecture

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. Examples of devices that require a CU are:

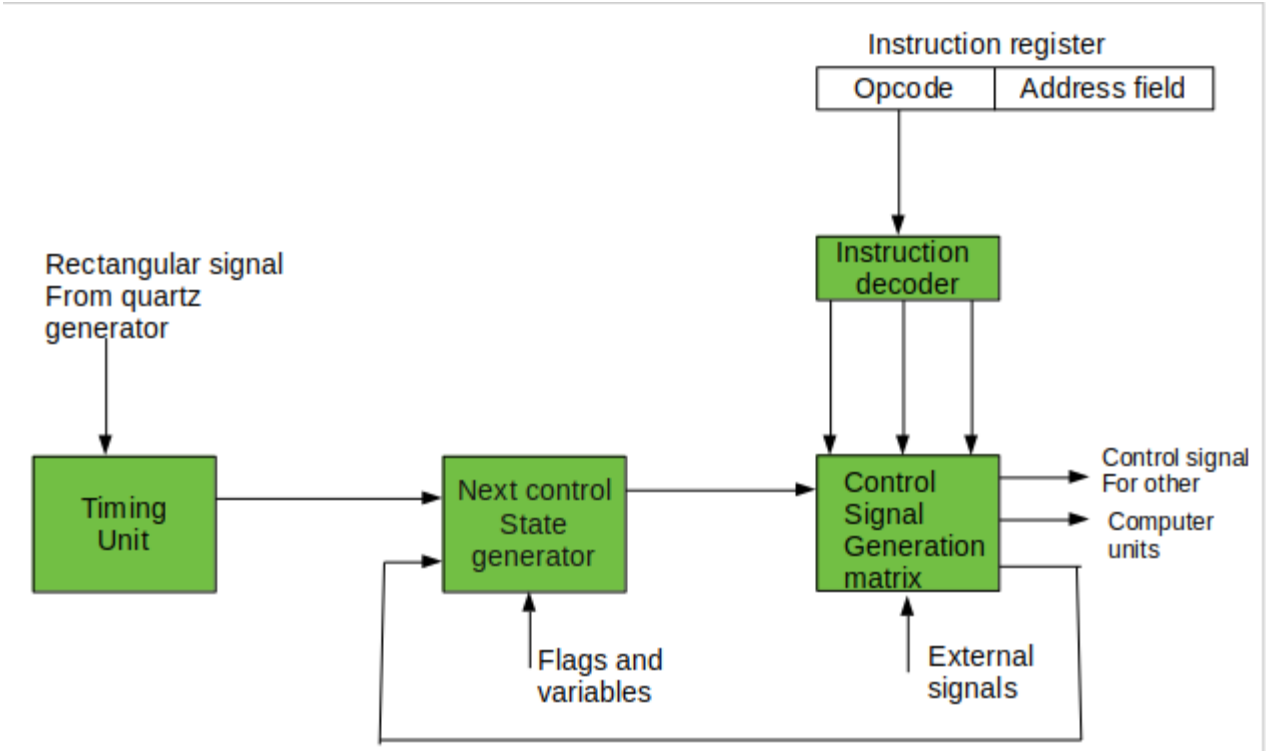
Functions of the Control Unit –

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

Types of Control Unit –

There are two types of control units: Hardwired control unit and Microprogrammable control unit.

1. **Hardwired Control Unit –**
   In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we can not modify the signal generation method without physical change of the circuit structure. 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. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.



Block diagram of a hardwired control unit of a computer

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit.

A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. This matrix combines these signals with the timing signals, which are generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered. A change of any of the earlier mentioned signals stimulates the change of the control unit state.

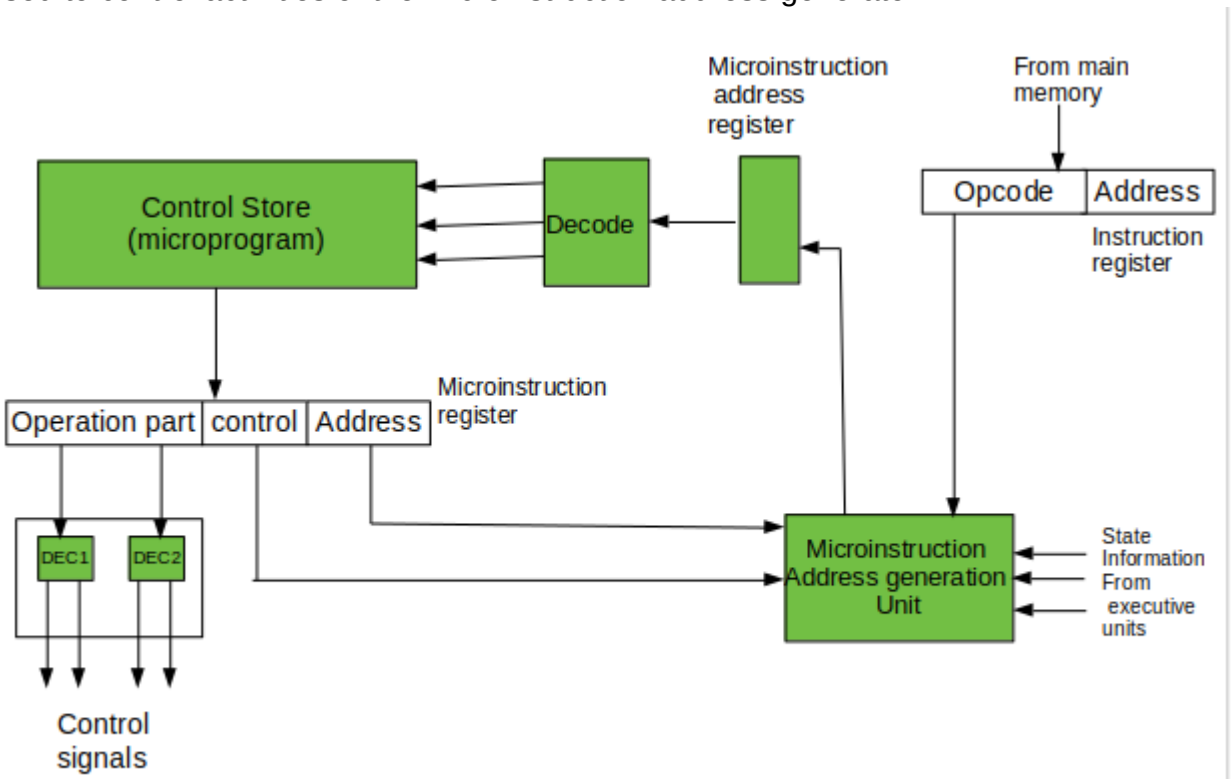
This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing). The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle.

The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.

2. **Microprogrammable control unit –**
The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals mandatory for instruction execution.
In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not

directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.

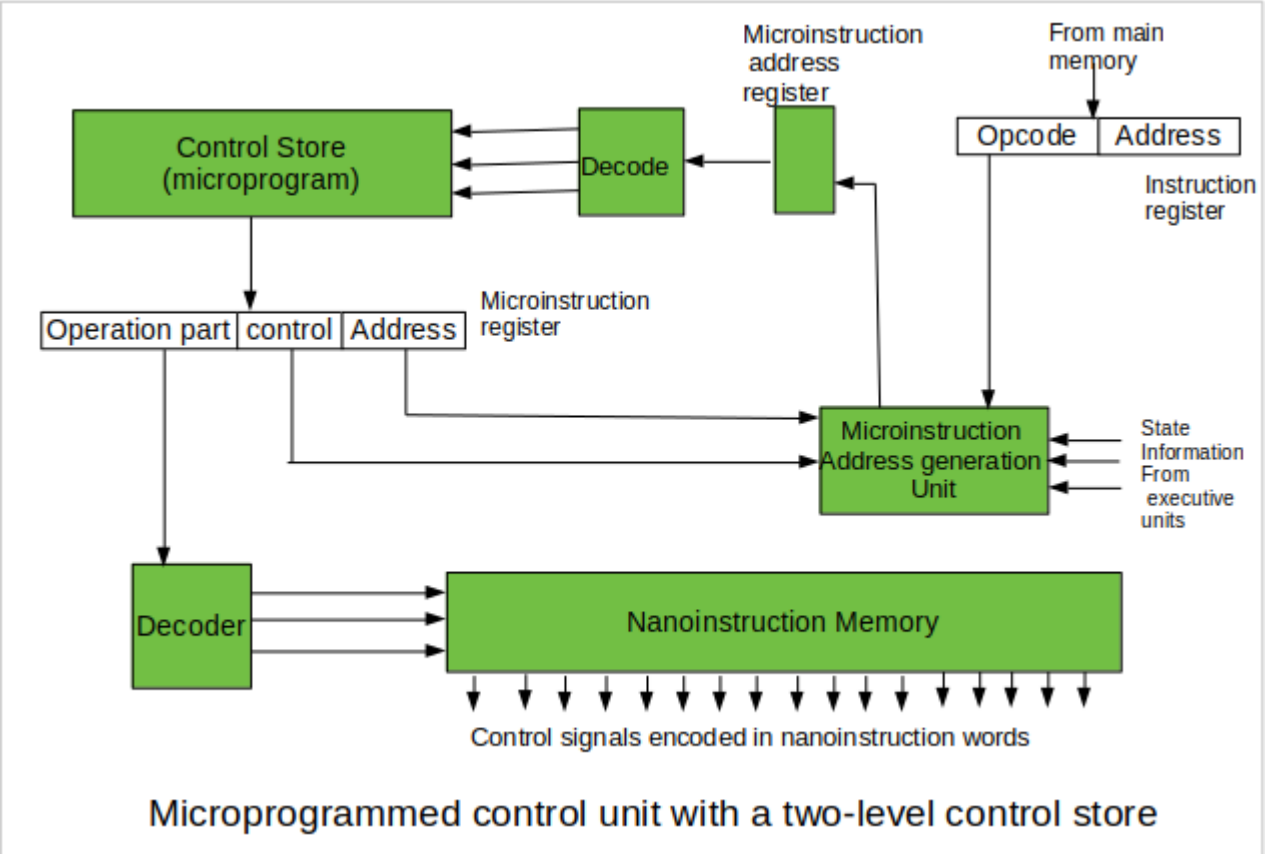
- **With a single-level control store:**
  In this, the instruction opcode from the instruction register is sent to the control store address register. Based on this address, the first microinstruction of a microprogram that interprets execution of this instruction is read to the microinstruction register. This microinstruction contains in its operation part encoded control signals, normally as few bit fields. In a set microinstruction field decoder, the fields are decoded. The microinstruction also contains the address of the next microinstruction of the given instruction microprogram and a control field used to control activities of the microinstruction address generator.



Microprogrammed control unit with a single level control store

The last-mentioned field decides the addressing mode (addressing operation) to be applied to the address embedded in the ongoing microinstruction. In microinstructions along with conditional addressing mode, this address is refined by using the processor condition flags that represent the status of computations in the current program. The last microinstruction in the instruction of the given microprogram is the microinstruction that fetches the next instruction from the main memory to the instruction register.

- **With a two-level control store:**
  In this, in a control unit with a two-level control store, besides the control memory for microinstructions, a nano-instruction memory is included. In such a control unit, microinstructions do not contain encoded control signals. The operation part of microinstructions contains the address of the word in the nano-instruction memory, which contains encoded control signals. The nano-instruction memory contains all combinations of control signals that appear in microprograms that interpret the complete instruction set of a given computer, written once in the form of nano-instructions.



Microprogrammed control unit with a two-level control store

In this way, unnecessary storing of the same operation parts of microinstructions is avoided. In this case, microinstruction word can be much shorter than with the single level control store. It gives a much smaller size in bits of the microinstruction memory and, as a result, a much smaller size of the entire control memory. The microinstruction memory contains the control for selection of consecutive microinstructions, while those control signals are generated at the basis of nano-instructions. In nano-instructions, control signals are frequently encoded using 1 bit/ 1 signal method that eliminates decoding.

Clock Signal:

All of the key components in a processor are connected to a clock signal. This alternates between high and low at a predefined interval known as the frequency. The logic inside processor typically switches values and performs calculations when the clock goes from low to high. The forceful increase in the clock to a processor, known as overclocking, is done to increase its performance. This performance gain comes from switching the transistors and logic inside a processor faster than it was designed for. Since there are more cycles per second, more work can get done and the processor will have a higher performance. This is true up to a certain point though. Modern processors typically run between 3.0GHz and 4.5GHz and that hasn't seemed to change in the past decade. Just like a metal chain is only as strong as the weakest link, a processor can only run as fast as the slowest part. By the end of each clock cycle, every single component in a processor needs to have finished its operation. If any parts aren't done yet, the clock is too fast and the processor won't work. Designers call this slowest part the Critical Path and it is what sets the maximum frequency a processor can run at. Above a certain frequency, the transistors simply cannot switch fast enough and will start glitching or producing incorrect outputs.

By increasing the supply voltage to a processor, we can speed up the switching of the transistors, but that only works up to a certain point as well. If we apply too much voltage, we risk burning up the processor. When we increase the frequency or voltage of a processor, it will always generate more heat and consume more power. This is because processor

power is directly proportional to frequency and proportional to the square of voltage. To determine the power consumption of a processor, we usually think of each transistor as a small capacitor that must be charged or discharged whenever it changes value.

Power delivery is such an important part of a processor that in some cases, half of the physical pins on a chip may be just for power or ground. Some chips may pull more than 150 amps at full load and all that current has to be managed extremely carefully. To put this amount of power into perspective, a CPU generates more heat per unit area than a nuclear reactor.

The clock in modern processors accounts for roughly 30-40% of its total power since it is so complex and must drive so many different devices. To conserve energy, most lower-power designs will turn off portions of the chip when they are not in use. This can be done by turning off the clock, known as Clock Gating, or turning off the power, known as Power Gating.

Clocks present another challenge to designing a processor since as their frequencies keep increasing, the laws of physics start getting in the way. Even though the speed of light is extremely fast, it is not fast enough for high-performance processors. If we were to connect the clock to one end of the chip, by the time the signal reached the other end, it would be out of sync by a considerable amount. To keep all portions of the chip in time, the clock is distributed using what is called an H-Tree. This is a structure that ensures all endpoints are the exact same distance from the centre.

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.

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	I_1				I_2			
S2		I_1				I_2		

STAGE / CYCLE	1	2	3	4	5	6	7	8
S3			I_1				I_2	
S4				I_1				I_2

Total time = 8 Cycle

Overlapped execution:

STAGE / CYCLE	1	2	3	4	5
S1	I_1	I_2			
S2		I_1	I_2		
S3			I_1	I_2	
S4				I_1	I_2

Total time = 5 Cycle

Pipeline Stages

- **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.

Energy and Power of a CPU:

Now that the construction of a CPU has been talked about, next is how the CPU actually works with electricity.

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

Some concerns of power ICs are:-

- Power must be brought in and distributed throughout the chip
- Power is dissipated as heat and that needs to be removed

A standard CPU consumes 65-85 W of Power.
A quad core processor consumes 95-140 W of Power.
e.g. Pentium IV : 84 W, Core i7 920 : 85W, Core i7 940 : 92W, Core i7 965 Extreme : 100W etc.

Thermal Design Power:- It refers to the power consumption under max theoretical load.

Energy is of two kinds:
Dynamic: Energy during ON stage
Static: Energy during OFF stage
Estatic : is proportional to off-stage current and leakage current of reverse bias.

----------XX----------