Unit-2 :

Control Unit : Data path and control path design, microprogramming v/s hardwired control, pipelining in CPU design, RISC v/s CISC, superscalar processors.

Introduction:

Control unit generates timing and control signals for the operations of the computer. The control unit communicates with ALU and main memory. It also controls the transmission between processor, memory and the various peripherals. It also instructs the ALU which operation has to be performed on data.

The control unit provides instructions to the other CPU devices in a way that causes them to operate coherently to achieve some goal. Basically, there is one control unit, because two control units may cause conflict. The control unit of a simple CPU performs the FETCH / DECODE / EXECUTE operations.

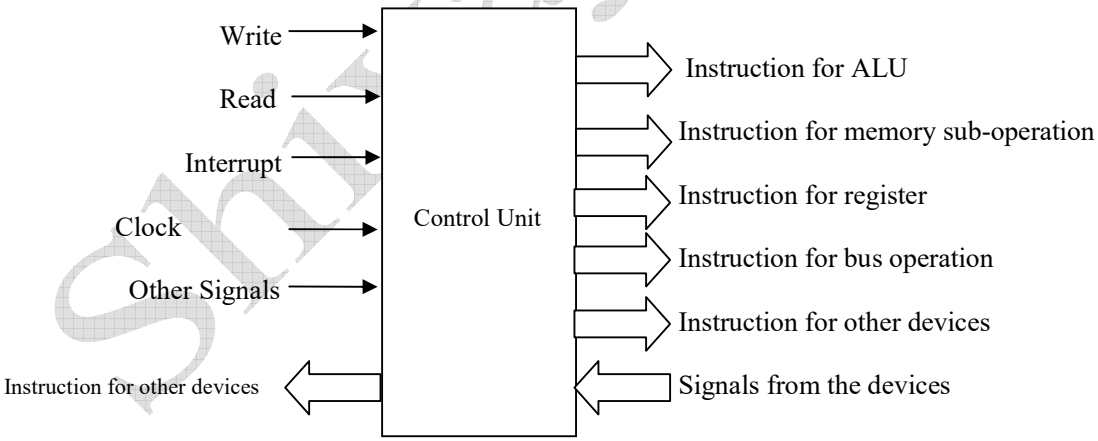


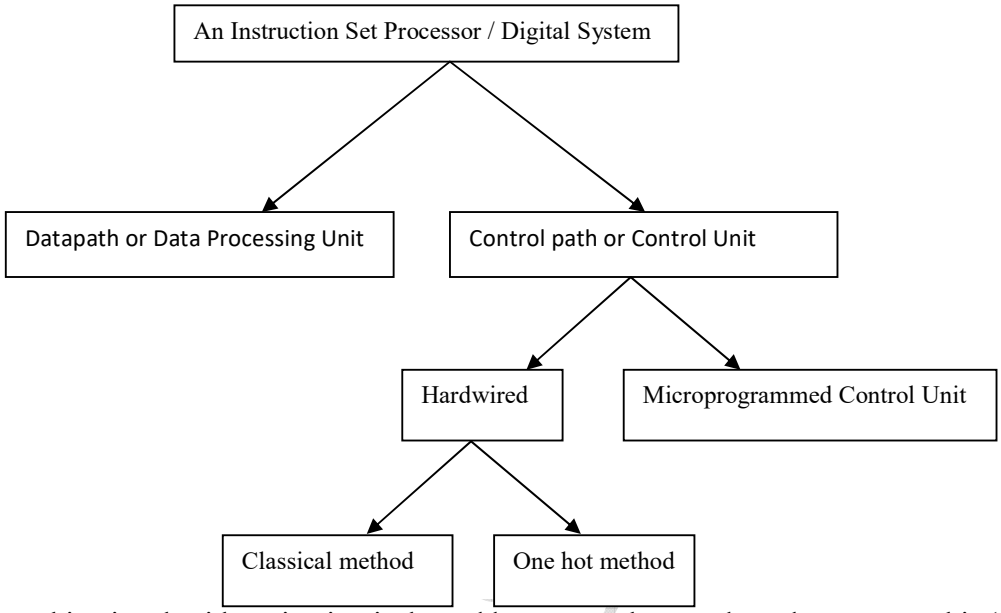
Fig a. A general control unit

An instruction set processor consists of datapath (data processing) and control units. The topic addresses the register-level design of the datapath unit and control unit.

(I) The **data path** is a network of functional and storage units capable of performing certain micro-operations on the data words. A CPU's data path contains circuits to perform arithmetic and logical operations on words such as

fixed-point or floating point numbers. The data path contains circuits to implement four basic arithmetic instructions for fixed-point numbers –addition, subtraction, multiplication and division. It also contains circuits for implementation of logic instructions and ALU design.

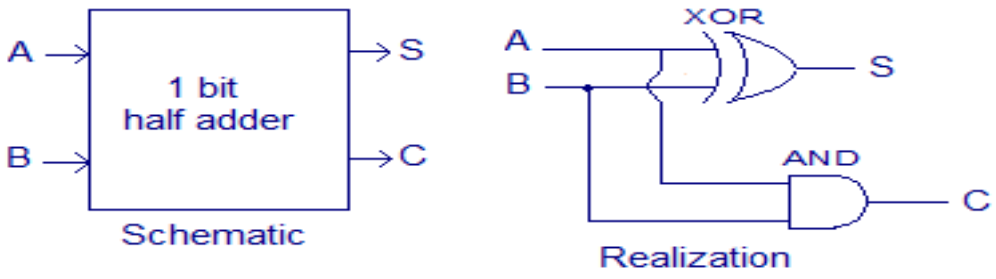
(II) The purpose of the **control unit** is to issue control signals to the data path. The control signals enter the datapath at "control points" where they select the functions to be performed at specific times and route the data through the appropriate parts of the datapath unit. In other words the control unit logically reconfigures the data path to implement some specified instruction or program.



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.

The truth table, schematic representation and XOR//AND realization of a half adder are shown in the figure below.



Inputs		Outputs	
A	B	Sum (S)	Carry(C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth Table

The sum (S) is the XOR of A and B. Therefore S = $A\bar{B} + \bar{A}B$

$= A \oplus B$

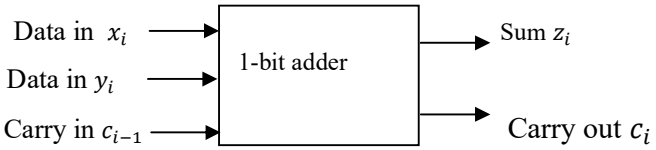
The carry(C) is the AND of A and B. Therefore C=A.B

FULL ADDER

A full adder is an arithmetic ckt that adds two bits and a carry and output a sum bit and a carry bit. When we want to add two binary numbers each having two or more bits, the LSB can be added by using a half adder.

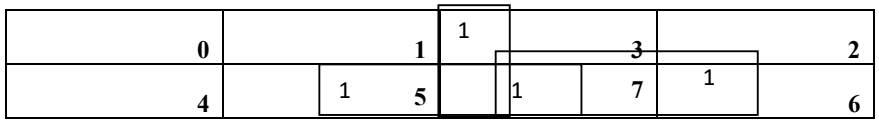
The carry resulted from the addition of the LSB's is carried over to the next significant column and added to the two bits in that column. So, in the second and higher column, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full adder adds the bits x_i and y_i and the carry from the previous column called the carry-in (c_{i-1}) and output the sum bit S and the Carry bit called the Carry-out (Cout).



Inputs			Outputs	
x_i	y_i	c_{i-1}	z_i	c_i
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

$$z_i = \bar{x}_i\ \bar{y}_i\ c_{i-1} + \bar{x}_i\ y_i\ \overline{c_{i-1}} + x_i\ \bar{y}_i\ \overline{c_{i-1}} + x_i\ y_i\ c_{i-1}$$



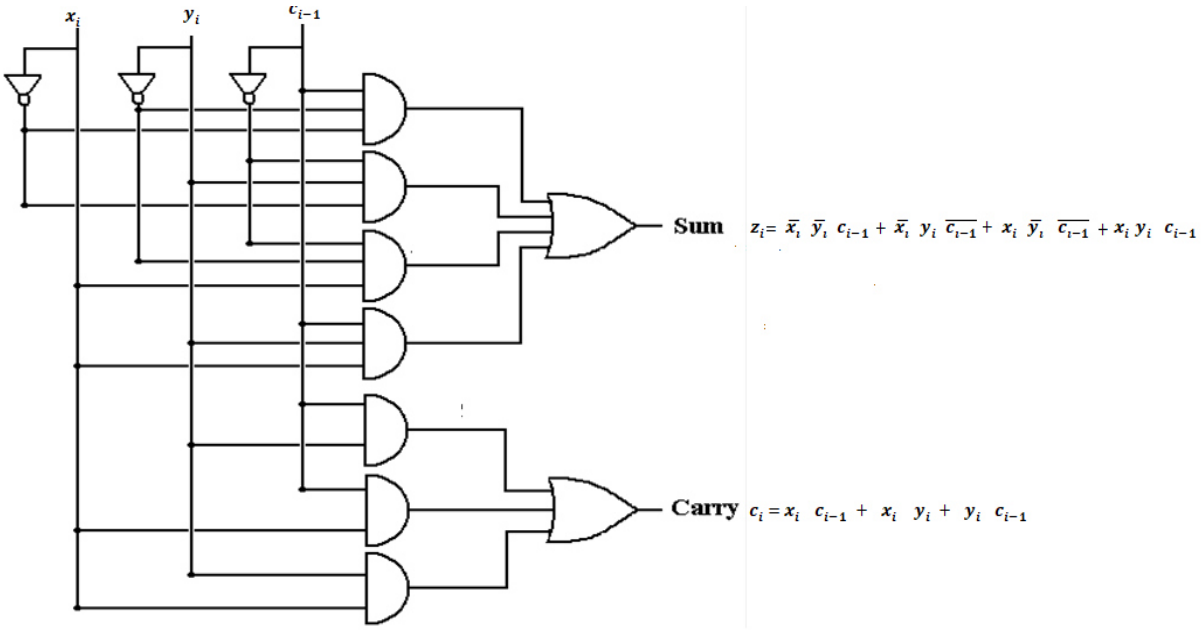
$$c_i = \bar{x}_i\ y_i\ c_{i-1} + x_i\bar{y}_i\ c_{i-1} + x_i y_i\ \overline{c_{i-1}} + x_i y_i\ c_{i-1}$$

$$= 0\,1\,1 \quad + 1\,0\,1 \quad + 1\,1\,0 \quad + 1\,1\,1$$

$$= 3 + 5 + 6 + 7$$

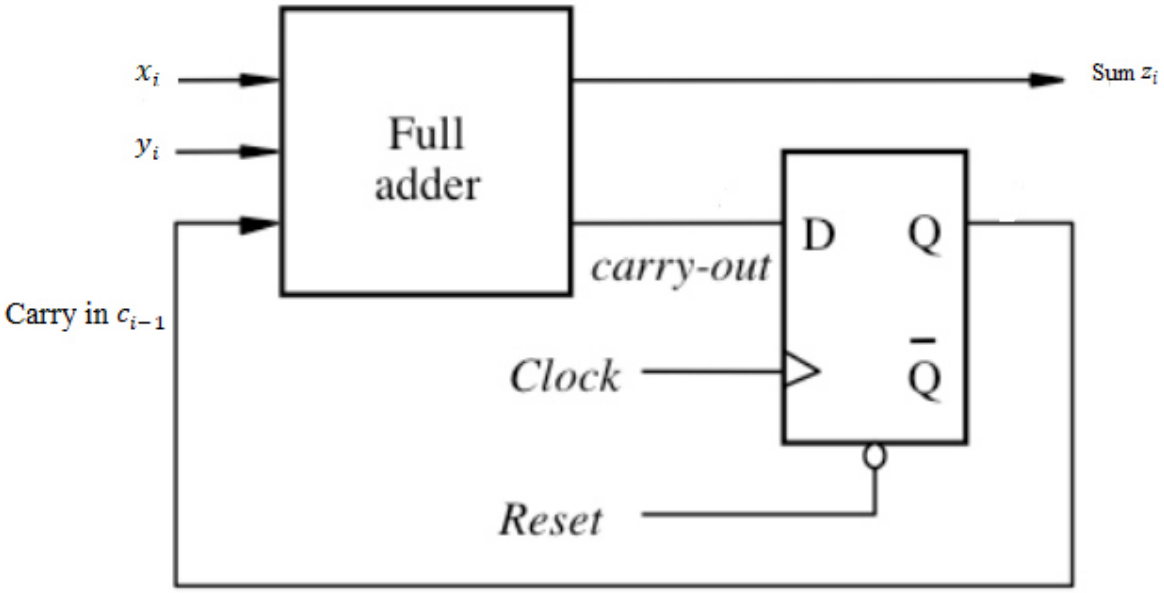
$$= \sum m\,(3, 5, 6, 7)$$

$$c_i = x_i \ \ c_{i-1} + x_i \ \ y_i + y_i \ \ c_{i-1}$$



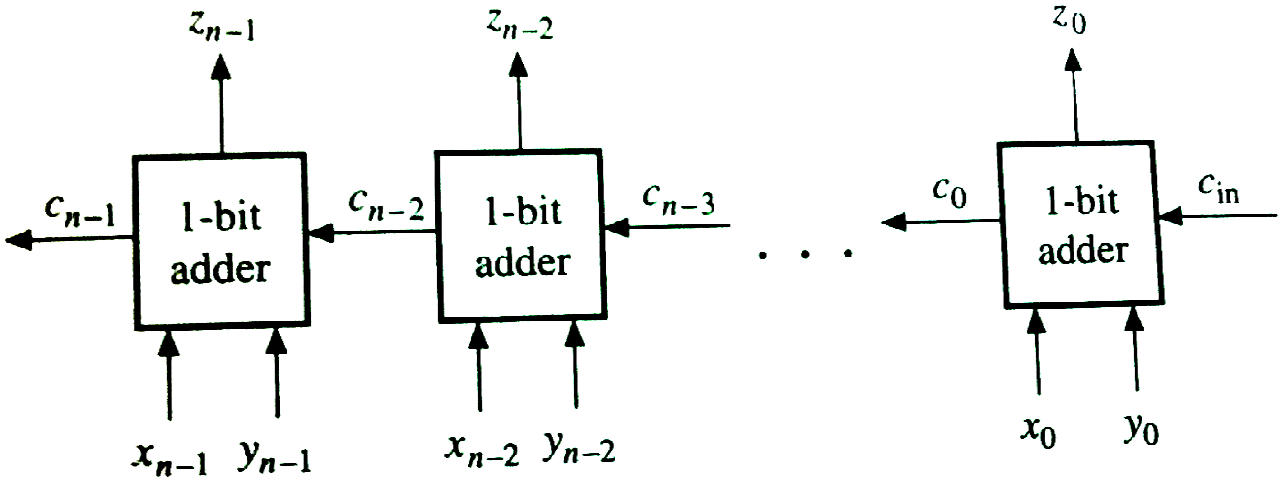
Above figure shows a fast AND-OR realization of a 1-bit adder, along with an appropriate circuit symbol for use in register level design.

A serial binary adder



The least expensive circuit in terms of hardware cost for adding two n-bit binary numbers is a serial adder. A serial adder adds the numbers bit by bit and so requires n clock cycles to compute the complete sum of two n-bit numbers. A above figure indicates a serial adder consist of a full adder and a flip flop to store $\boldsymbol{c_i}$**.** One sum bit is generated in each clock cycle, a carry is also computed and stored for use during the next clock cycle. Above fig presents a high level view of a serial adder that has a D flip flop as the carry store. Although this adder is slow its circuit size is very small and is independent of n.

An n-bit ripple carry adder



Circuit that, in one clock cycle, add all bits of two n bit numbers as well as an external carry in signal c_{in} are called n bit parallel adder or simply n bit adders .The simplest such adder is formed by connecting n full adders as in fig. Each 1 bit adder stage supplies a carry bit to the stage on its left. A 1 appearing on the carry in line of a1 bit adder can cause it to generate a 1 on its carry out line.

Hence carry signals propagate though the adder from right to left giving rise to the name ripple carry adder carry adder. In the worst case a carry signal can ripple through all n stages of the adder. The input carry signal c_{in} is normally set to 0 through all n stages of the adder. The input carry signal c_{in} is normally set to 0 for addition. The maximum signal propagation delay of an n bit ripple carry adder which is synchronous circuit design determines the operating speed, is nd, where d is the delay of a full adder stage. Unlike a serial adder, the amount of hardware in a ripple carry adder increases linearly with n the work size of the numbers being added.

Drawback

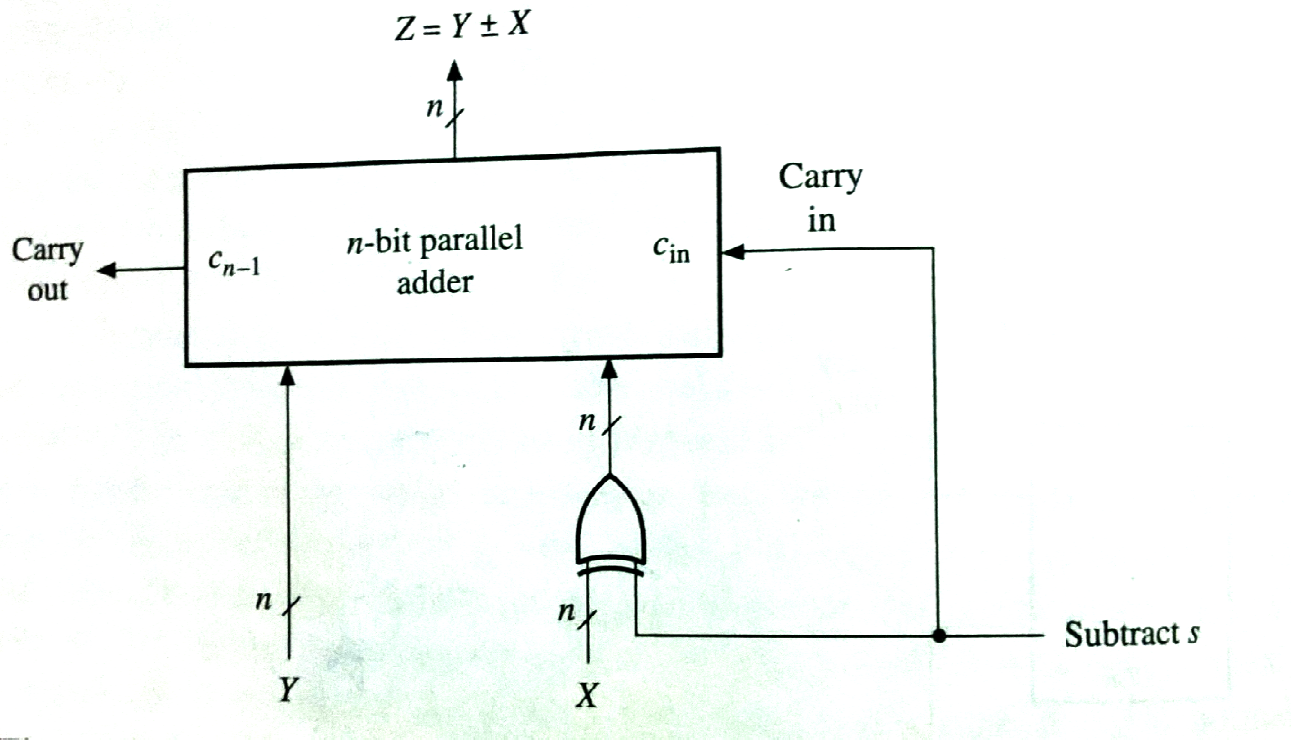
In Ripple Carry Ahead Adder, we cannot perform addition of two bits for any particular block unless the output from the previous block is generated. This delays the overall addition process and performance of the system.

The propagation time is equal to the propagation delay of the typical gate times the number of gate levels in the circuit.

For example, if each full adder stage has a propagation delay of 20n seconds, then FA 4 will reach its final correct value after 80n (20 × 4) seconds. If we extend the number of stages for adding more number of bits then this situation becomes much worse

This particular drawback of Ripple Carry Ahead Adder is overcomed by using Look Ahead Carry Adder.

An n-bit twos complement adder-subtracter



An n bit 2's Complement adder-subtracter is shown in fig above. The above circuit is capable of performing both addition as well as subtraction.

Addition: When S=0, Cin = 0, one input to the XOR gate is 0, the X bits pass through it without inversion. Therefore the n bit parallel adder will add the X and Y bits and produce the Sum (Z) =Y+X and Carry out.

Subtraction: When S=1, Cin=1, one input to the XOR gate is 1, now XOR gate works as a controlled inverter, the X bits passes through the gate with inversion.
Adding –X to Y is equivalent to subtracting X from Y so the ability to add negative numbers implies the ability to do subtraction. Subtraction is relatively simple with twos complement code because negation (changing X to –X) is very easy to implement.

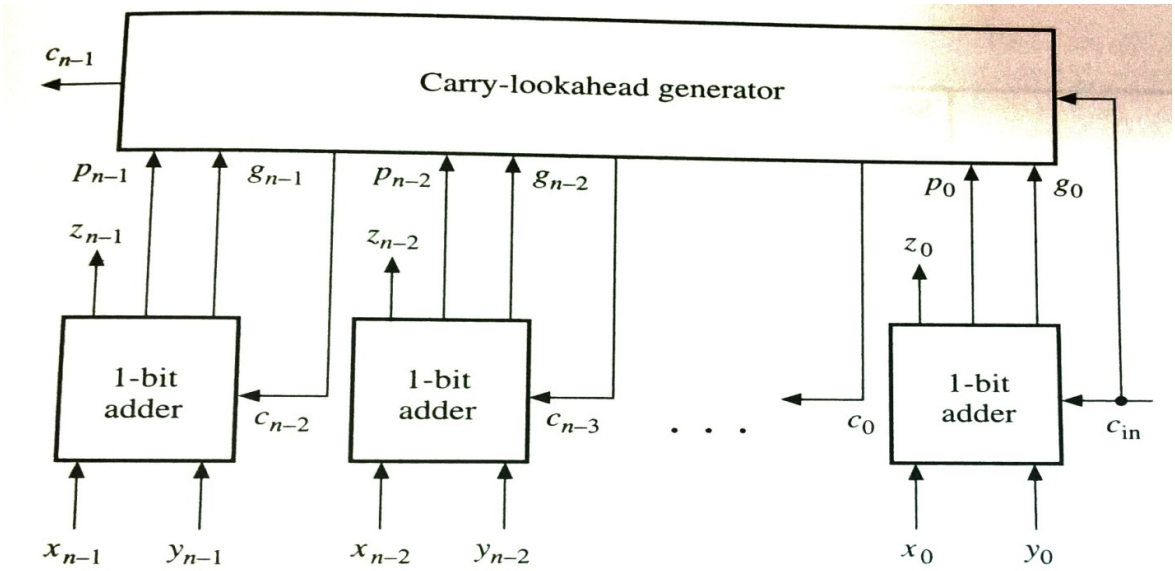
If X = $x_{n-1}\ x_{n-2}\ x_{n-3}$ ……. x_0
then 1's Complement of X is $\bar{X} = \bar{x}_{n-1}\ \bar{x}_{n-2}\ \bar{x}_{n-3}$ …… $\bar{x}_0$
and 2's Complement of X is –X = $\bar{x}_{n-1}\ \bar{x}_{n-2}\ \bar{x}_{n-3}$ …… $\bar{x}_0$ +1
An efficient way to obtain the ones complement is to use XOR function. X $\oplus$ s with a control variable s.
When s=0, X $\oplus$ s = X, but when s=1, X $\oplus$ s = $\bar{X}$.
Suppose that Y and X $\oplus$ s are now applied to the inputs of an n-bit adder. The addition 1 required to change $\bar{X}$ to –X can be realized by applying s to the carry input line of the adder. In the resulting circuit the control line s selects the addition operation Y+X when s=0 and the subtraction operation Y-X =Y+$\bar{X}$ + 1 when s=1. Thus extending a parallel adder to perform twos complement subtraction as well as addition merely requires connecting n two input XOR gates to the adder's inputs these gates are represented by a single n-bit word gate.
Example:

Drawback of Ripple Carry Adder:

In Ripple Carry Ahead Adder, we cannot perform addition of two bits for any particular block unless the output from the previous block is generated. This delays the overall Addition process and performance of the system. This particular drawback of Ripple Carry Ahead Adder is overcomed by using Look Ahead Carry Adder.

LOOK AHEAD CARRY ADDER / High speed Adder/ Carry Look ahead adder/ CLA Adder

If we know the output of previous block in advance using any logical or hardware arrangement then we can perform addition quickly by using look Ahead Carry Adder. In look Ahead Carry Adder, the computation for all the blocks is done simultaneously and when the output is generated from the previous block, it is feeded to the next consecutive block and the process executes.



x_0	y_0	c_{in}	z_0	c_0
0	0	0	-	0
0	0	1	-	0
0	1	0	-	0
0	1	1	-	1
1	0	0	-	0
1	0	1	-	1
1	1	0	-	1
1	1	1	-	1

When x_0 $y_0$are different
$x_0 \oplus y_0$=1 & $c_{in} = 1$ then only c_0= 1

When both x_0 & y_0 are 1 then $c_0 = 1$ it does not depend upon c_{in} only. i.e $x_0 \cdot y_0$

Therefore $c_0 = \overbrace{x_0\, y_0}^{G} + \overbrace{(x_0 \oplus y_0)}^{P} c_{in}$

G=Carry Generator because there is no dependency on c_{in}

P= Carry Propagator because carry is generated ($c_0 = 1$) only when ($c_{in} = 1$) i.e carry is propagated as 1.

$c_0 = G + P\ c_{in}$ $\{c_{in} = c_{i-1}\}$

Now generalized the above equation

$$c_i = G_i + P_i c_{i-1}$$

If i=0

$c_0 = G_0 + P_0 c_{-1}$ ---------**(1)**

i.e $c_0 = G_0 + P_0 c_{in}$

If i=1

$c_1 = G_1 + P_1 c_0$

$c_1 = G_1 + P_1(G_0 + P_0 c_{-1})$

$c_1 = G_1 + P_1 G_0 + P_1 P_0\ c_{-1}$ ----------**(2)**

$i.e\ c_1 = G_1 + P_1 G_0 + P_1 P_0\ c_{in}$

If i=2

$c_2 = G_2 + P_2 c_1$

$c_2 = G_2 + P_2(G_1 + P_1 G_0 + P_1 P_0\ c_{-1})$

$c_2 = G_2 + P_2 G_1 + P_2\ P_1 G_0 + P_2\ P_1 P_0\ c_{-1}$ ----------**(3)**

i.e $c_2 = G_2 + P_2 G_1 + P_2\ P_1 G_0 + P_2\ P_1 P_0\ c_{in}$

If i=3

$c_3 = G_3 + P_3 c_2$

$c_3 = G_3 + P_3(G_2 + P_2 G_1 + P_2\ P_1 G_0 + P_2\ P_1 P_0\ c_{-1})$

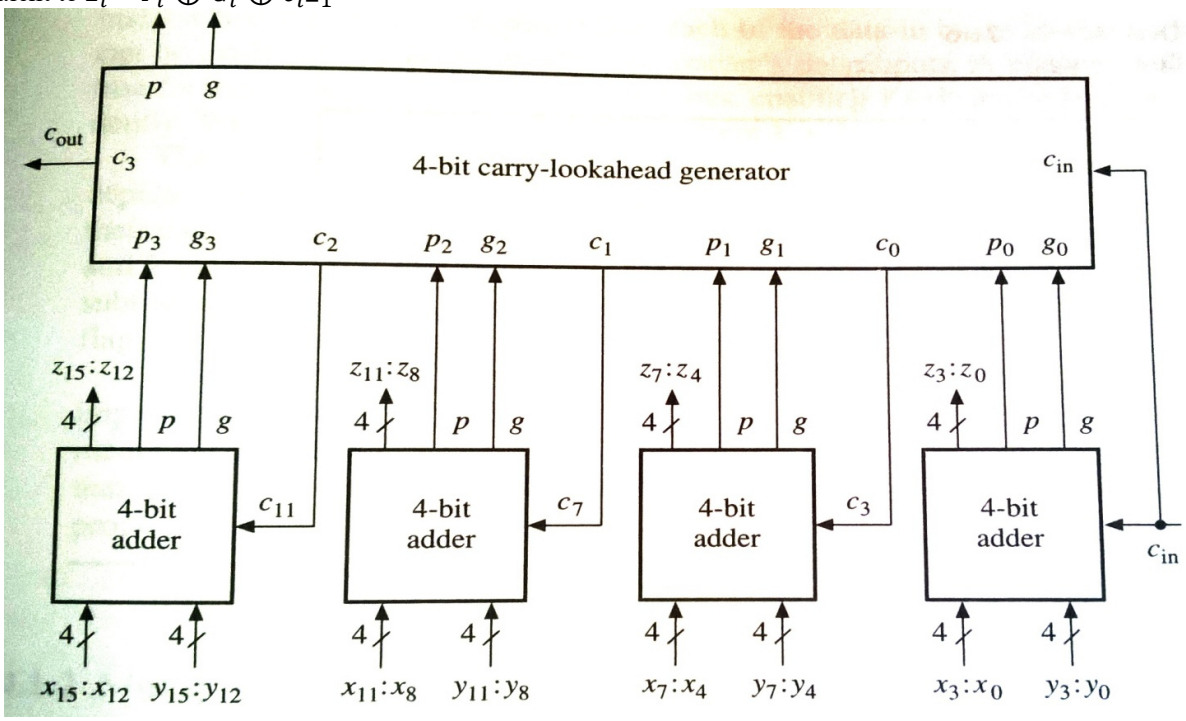
$c_3 = G_3 + P_3 G_2 + P_3\ P_2 G_1 + P_3 P_2\ P_1 G_0 + P_3 P_2\ P_1 P_0\ c_{-1}$ ----------**(4)**

$c_3 = G_3 + P_3 G_2 + P_3\ P_2 G_1 + P_3 P_2\ P_1 G_0 + P_3 P_2\ P_1 P_0\ c_{in}$

We can further simplify the design by noting that the sum equation for stage i

$z_i = x_i \oplus y_i \oplus c_{i-1}$

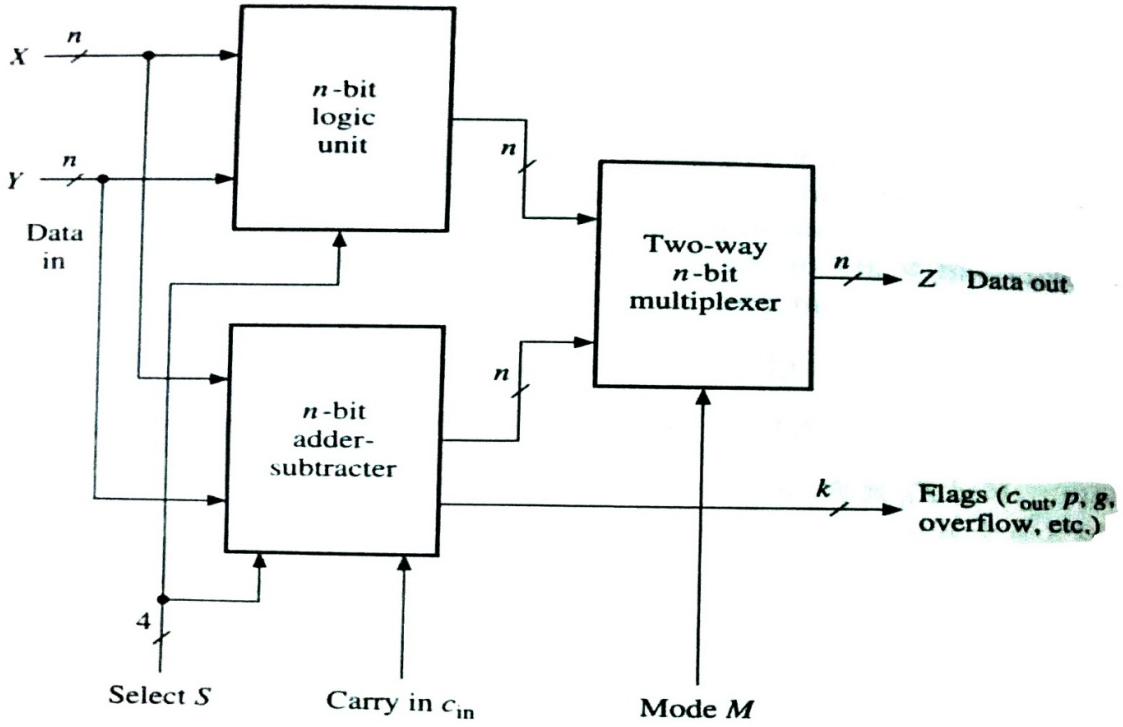
is equivalent to $z_i = P_i \oplus G_i \oplus c_{i-1}$



A 16 bit adder composed of 4-bit adders linked by carry lookahead.

Booth's Algorithm

n-bit Arithmetic logic unit or Combinational ALU



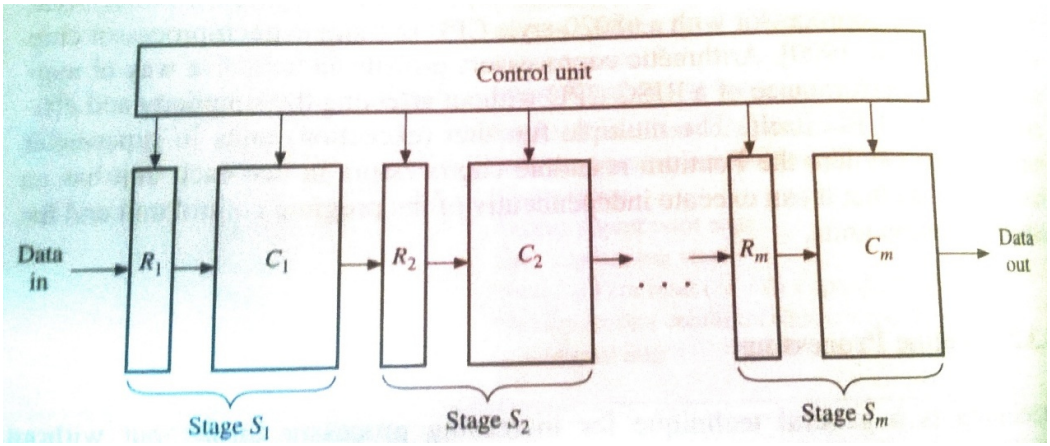
Various circuits used to execute data processing instructions are usually combined in a single circuit called an arithmetic logic unit (ALU). Simple ALU that perform fixed-point addition and subtraction as well as word based logical operation can be realized by combinational circuits. ALUs that also perform multiplication and division can be constructed around the same circuits.

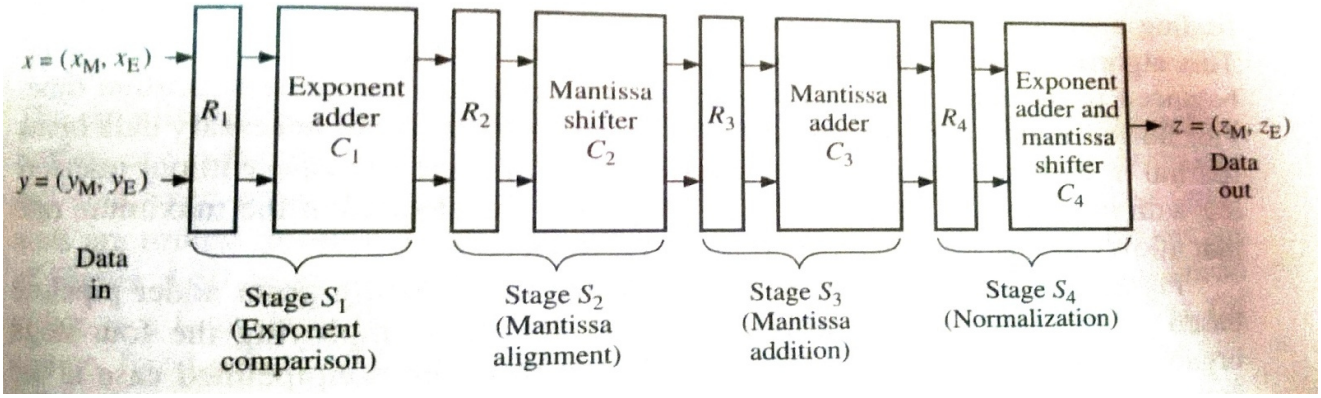
The simplest ALUs combine the function of a two's complement adder-subtractor with those of a circuit that generates word-based logic functions of the form $f(X, Y)$, for example, AND, XOR, and NOT. They can thus implement most of a CPU's fixed-point data processing instructions. Above fig outlines an ALU that has separate subunits for logical and arithmetic operations. The particular class of operation logical and arithmetic to be performed is determined by a "mode" control line M attached to a two way multiplexer that channels the required result to the output bus Z. The specific operation performed by the desired subunit is determined by a "select" control line S. The ALU's logical operations are performed bitwise that is the same operation f is applied to every pair of data lines x_i y_i . The maximum number of distinct logical operations of the form $f(x_i, y_i)$ is 16, which is the number of distinct truth tables of two Boolean variables. Hence the select bus S needs to be of size 4 at most. S can also be used to select up to 16 different arithmetic operations such as X+Y, X-Y, Y-X, X+1,X-1 and so on as needed.

The logical operations in above fig can be obtained by generating all four minterms of $f(x_i, y_i)$, namely

$m_3 = x_i y_i$, $m_2 = x_i \overline{y_i}$, $m_2 = \overline{x_i} y_i$ $m_0 = \overline{x_i}\,\overline{y_i}$, for every pair $x_i y_i$ of data bits and by using the control lines S=S3S2S1S0 to select desired subsets of the minterms to be ORed together.

STRUCTURE OF A PIPELINE PROCESSOR





Four Stage floating-point adder pipeline

Pipelining is a general technique for increasing processor performance without requiring large amount of extra hardware this technique is applied to the design of the complex DPU such as multiplier and floating point adders.

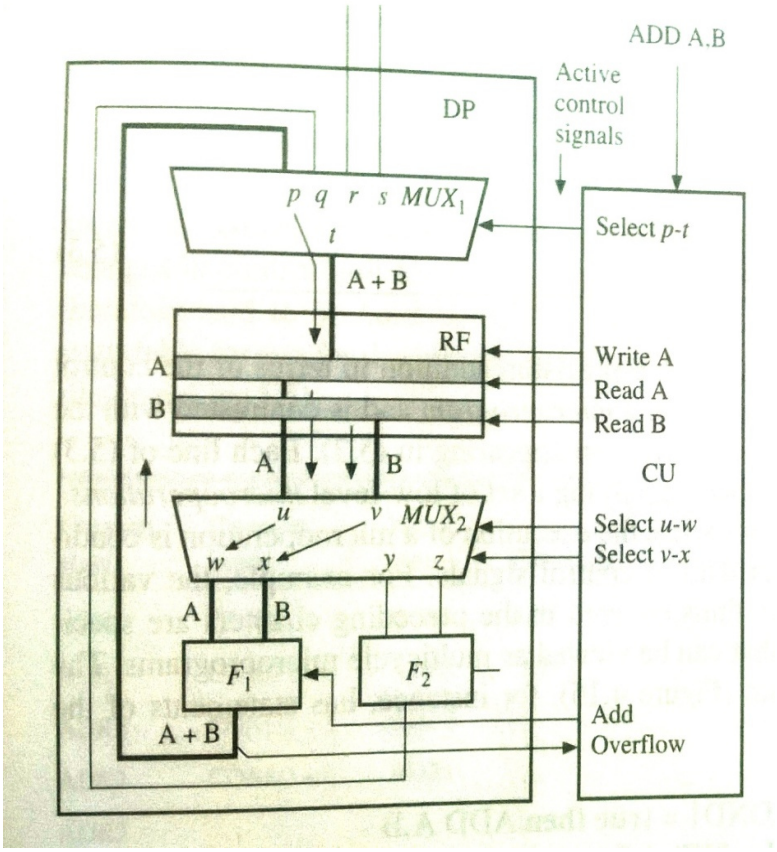
A pipeline processor consists of a sequence of 'm' data processing circuit called stages or segments which collectively perform a single operation on a stream of data operands passing through them. Some processing takes place in each stage but a final result is obtained only after an operand set has passed through an entire pipeline. As shown in fig above a stage Si contains a multiword input register or latch Ri and data path circuit Ci that is usually combinational. The Ri's hold partially process result as they move through the pipeline, they also serve as buffers that prevent neighboring stages from interfering with one another. A common clock signal causes Ri's to change stage synchronously. Each Ri receives a new set of input data Di-1 from the preceding stage Si-1 except R1 whose data is supplied from an external source.

Di-1 represents the result generated by Ci-1 once Di-1 is loaded into Ri, Ci proceeds to use Di-1 to compute a new data set Di. Thus in each clock period every stage transfers its previous result to next stage and computes a new set of result.

Advantages:

(1) In pipeline processor m stage pipeline can simultaneously process upto 'm' independent set of data operands.
(2) These data sets move though the pipeline stage by stage so that when the pipeline is full 'm' separate operations are being executed simultaneously each in different stage, and a new final result emerges from the pipeline every clock cycle.
(3) Effective time utilization or increasing processor performance.
Suppose that each stage of 'm' stage pipeline takes T seconds to perform its local suboperations and stores its results. T is the pipeline clock period. The delay or latency of the pipeline i.e the time to complete a single operation is mT. However the output of the pipeline i.e the maximum no of operations completed per second is 1/T. When performing a long sequence of operations in the pipeline its performance is determined by the delay T of a single stage rather than by the delay of the entire pipeline. Hence an m stage pipeline provides a speed up factor of 'm' compared to a non pipelined implementation of the same target operation.

Processor composed of a Datapath Unit DP and a Control Unit CU configured to implement the add operation A: =A+B



A CPU's datapath contains circuits to perform arithmetic and logical operations on words such as fixed point or floating point numbers. The internal structure of datapath circuit DP of a small microprocessor is depicted in fig above. It contains RF for temporary storage of operands, two functional units F1 and F2 responsible for data processing and multiplexer to allow the data to be steered through DP. Typical functional units are an ALU performing addition, subtraction and logical operations, shifter or a multiplier. The CU receives instructions or commands which it converts into a sequence of control signals that the CPU applies to DP to implement a sequence of register transfer operations.
The control signals that implement an addition instruction of the form ADD A,B which we write as A:=A+B
The input variables A and B are obtained from registers of the same name in RF, and the result is stored back into register A. RF is configured with two input and two output ports to support operations like with two or three addresses. Besides

selecting the data registers to be used, the control unit CU must also select the operation to be performed on the data in this case functional unit F1's ADD operation. Finally the necessary logical connections for the data to flow through DP must be established by applying appropriate control signals to the multiplexers.

Thus we see that CU must activate the following three types of control signals during the clock cycle in which the ADD A, B instruction is executed.

1. Function select: ADD
2. Storage Control: Read A, Read B, Write A.
3. Data routing: Select p-t, Select u-w, Select v-x

There is usually some feedback of control information from DP to CU to indicate exceptional conditions encountered during execution.

The functional unit F1 performing the addition sends an overflow signal to CU whenever the sum A+B exceeds the normal word size.

Control Unit Design

Control unit generates timing and control signals for the operations of the computer. The control unit communicates with ALU and main memory. It also controls the transmission between processor, memory and the various peripherals. It also instructs the ALU which operation has to be performed on data.

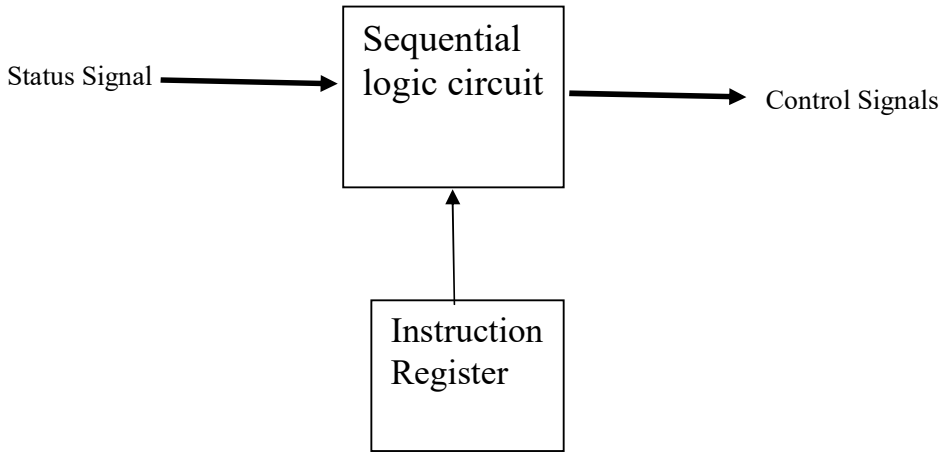
Control unit can be designed by two methods which are given below.

(I) Hardwired Control Unit

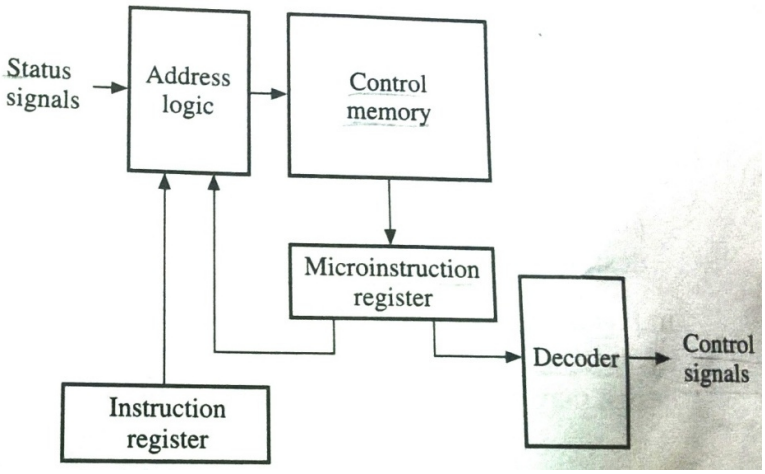
One approach views the controller as a sequential logic circuit or finite state machine that generates specific sequences of control signals in response to externally supplied instructions. It is designed with the usual goal of minimizing the number of components used and maximizing the speed of operation. Once the unit is constructed the only way to implement changes in control unit behavior is by redesigning the entire unit. Such a circuit is called as hardwired.

It is implemented with the help of gates, flip flops, decoders etc. in the hardware. The inputs to control unit are the instruction register, flags, timing signals etc. This organization can be very complicated if we have to make the control unit large. If the design has to be modified or changed, all the combinational circuits have to be modified which is a very difficult task.

RISC processor with their emphasis on small, fast instruction set favor the use of hardwired control units.

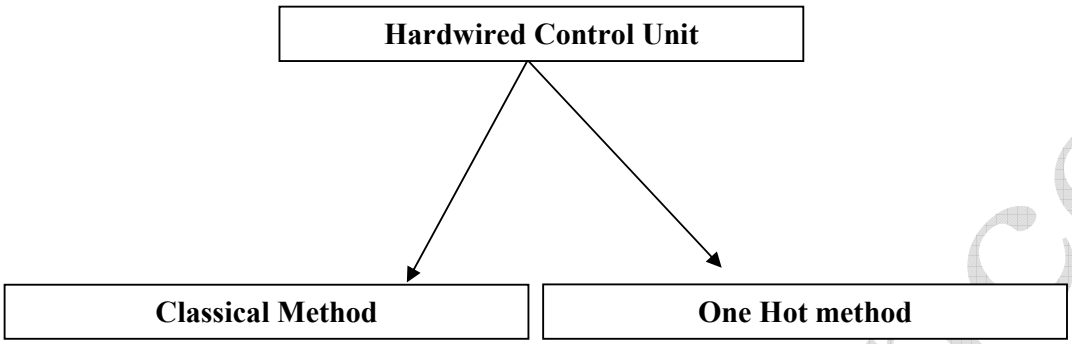


(II) Microprogrammed Control Unit



Microprogramming provides an alternative method of designing program control unit. It is built around a storage unit called a control memory, where all the control signals are stored in a program like format. The control memory stores a set of microprograms designed to implement or emulate the behavior of the given instruction set. Each instruction causes the corresponding microprogram to be fetched and its control information extracted in a manner that resembles the fetching and execution of a program from the computer's main memory. Microprogramming makes control unit design more systematic by organizing control signals into formatted words. Since the control signals are embedded in a kind of low level software this is referred to as firmware. Design changes can be easily made by altering the contents of the control memory and its access circuitry. Microprogrammed units tend to be slower because of extra time required to fetch microinstructions from the control memory.

Hardwired Control	Microprogrammed Control
Technology is circuit based.	Technology is software based.
It is implemented through flip-flops, gates, decoders etc.	Microinstructions generate signals to control the execution of instructions.
Fixed instruction format.	Variable instruction format (16-64 bits per instruction).
Instructions are register based.	Instructions are not register based.
ROM is not used.	ROM is used.
It is used in RISC.	It is used in CISC.
Faster decoding.	Slower decoding.
Difficult to modify.	Easily modified.
Chip area is less.	Chip area is large.



(1) Classical Method: The classical method of sequential circuit design, attempts to minimize the amount of hardware, in particular, by using only [log_2P] flip flop to realize a P– state circuit.
(2) One hot- Method: The One hot Method uses one flip-flop per state. This method simplifies CU design and debugging.

Concept of State Table

	Inputs			
State	I_1	I_2		I_m
S1	$S_{1,1}, O_{1,1}$	$S_{1,2}, O_{1,2}$	. . .	$S_{1,m}, O_{1,m}$
S2	$S_{2,1}, O_{2,1}$	$S_{2,2}, O_{2,2}$	. . .	$S_{2,m}, O_{2,m}$
			. . .	
Sn	$S_{n,1}, O_{n,1}$	$S_{n,2}, O_{n,2}$	. . .	$S_{n,m}, O_{n,m}$

(a)

	Inputs				
State	I_1	I_2		I_m	Outputs
S1	$S_{1,1}$	$S_{1,2}$	. . .	$S_{1,m}$	O_1
S2	$S_{2,1}$	$S_{2,2}$	. . .	$S_{2,m}$	O_2
			. . .		
Sn	$S_{n,1}$	$S_{n,2}$	. . .	$S_{n,m}$	O_n

State tables for a finite state machine (a) Mealy type and (b) Moore type

The behavior required of a control unit like that of any finite state machine can be represented by a state table of the general type shown in fig (a). The rows of the state table correspond to the set of internal states {Si}.These states are determined by the information stored in the machine at discrete points of time.Let X and Z denote the input and output variables.

*A **finite state machine** is a computation model that can be implemented with hardware or software and can be used to simulate sequential logic and some computer programs*

The columns correspond to the combinations of the X signals that can be applied to the machine and are denoted here by {Ij}. The entry in row Si and column Ij has the form Sij, Oij, where Sij is the next state of the machine that results from the application of input combination Ij, and Oi,j denotes the output signals that appear on Z whenever the machine is in State Si, with input Ij applied. In general, an entry in the state table defines a specific, one cycle transition between two states.

Control units have a feature that favors a slightly different style of state table: Their output signal values often depend on the current state Si only and so are independent of the input combination Ij. If all outputs are of this type, the circuit is called a *Moore machine (fig b).*
The state table of **fig a** becomes a Moore machine if for every row i, we have Oi,j =Oi,k=Oi for all j,k=1,2,…..m. In that case we can represent the machine's behavior in the more compact format of **fig b,** where the output signals associated with each row are placed in a separate column.

GCD Processor

To illustrate the classical and one hot approaches to control unit design we will can apply them to a special purpose processor that computes the greatest common divisor gcd(X,Y) of two integers X andY; gcd (X,Y) is defined as the largest integer that divides exactly into both X and Y. For example gcd (12,18) = 6, and gcd (12,17)=1. It is customary to assume that gcd(0,0)=0. We use a variant of **Euclid's algorithm** to calculate gcd(X,Y).

```
gcd(in: X,Y; out: Z);
   register XR, YR, TEMPR;
   XR := X;                          {Input the data}
   YR := Y;
   while XR > 0 do begin
      if XR ≤ YR then begin          {Swap XR and YR}
         TEMPR := YR;
         YR := XR;
         XR := TEMPR; end
         XR := XR – YR;              {Subtract YR from XR}
   end
   Z := YR;                          {Output the result}
end gcd;
```

Fig a: Procedure to compute the greatest common divisor (gcd) of two numbers or HDL description

The basic idea is to subtract the smaller of the two numbers from the other repeatedly (division corresponds to repeated subtraction)
For example, with X=20 and Y=12, our gcd algorithm proceeds as follows.

Conditions		Actions	
		$XR := 20; YR := 12;$	
$XR > 0$:	$XR > YR$:	$XR := XR - YR = 8;$	
$XR > 0$:	$XR \leq YR$:	$YR := 8; XR := 12;$	$XR := XR - YR = 4;$
$XR > 0$:	$XR \leq YR$:	$YR := 4; XR := 8;$	$XR := XR - YR = 4;$
$XR > 0$	$XR \leq YR$:	$YR := 4; XR := 4;$	$XR := XR - YR = 0;$
$XR \leq 0$:		$Z := 4;$	

Fig b: gcd Algorithm

Hence we conclude that gcd (20,12) = 4

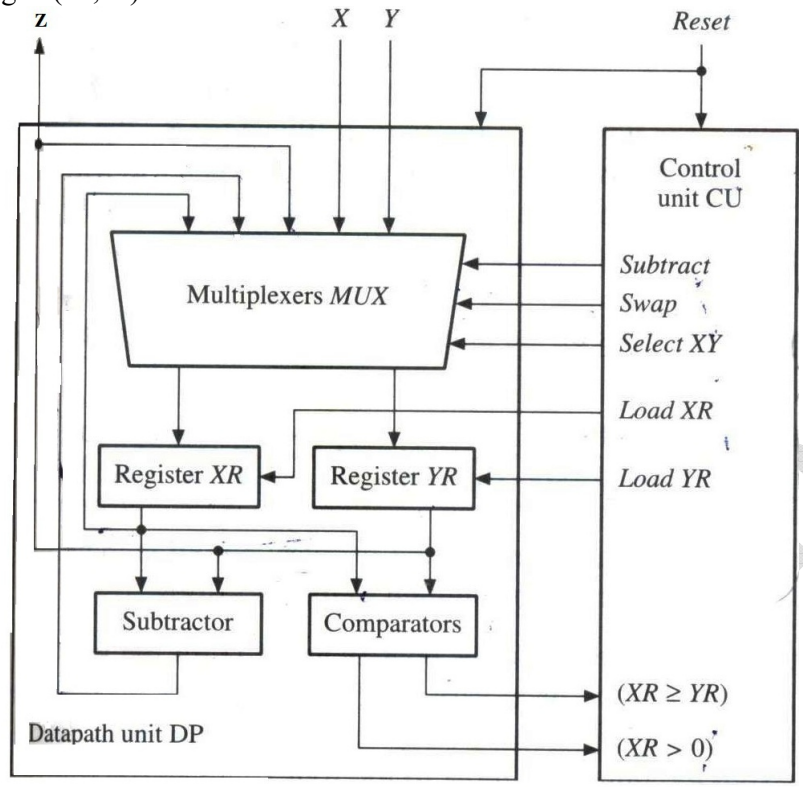


Fig c: Hardware needed to implement the gcd processor

Analysis of the gcd procedure (fig a) suggests that its datapath unit DP should contain a pair of register XR and YR to store the corresponding variables, one or more functional units to perform subtraction and magnitude comparison, and multiplexer for data routing, as shown in fig (c). Temporary Variable (TEMPR) is not required in Datapath Unit as **read from** and **write to** a register is done in the same clock cycle. The swap operation can therefore be done without conflict in one cycle thus: X:=Y, Y:=X.

The control unit CU generates control signals *Load XR* and *Load YR* to load each register independently with the input data X and Y. A control signal *Select XY* routes X and Y to XR and YR, respectively. Another signal *Swap* controls the swap operation, which requires routing the outputs of the XR and YR registers to each other's inputs. A final signal *Subtract* is assumed to control the subtraction XR:=XR-YR by routing the output of the subtracter to XR. The input signals to CU are an asynchronous *Reset* signal, two comparison signals (XR≥YR) and (XR>0) generated by DP, and the usual, implicit clock signal.

	Inputs (XR > 0) (XR >= YR)			Outputs				
State	0-	10	11	Subtract	Swap	Select XY	Load XR	Load YR
S_0 (Begin)	S_3	S_1	S_2	0	0	1	1	1
S_1 (Swap)	S_2	S_2	S_2	0	1	0	1	1
S_2 (Subtract)	S_3	S_1	S_2	1	0	0	1	0
S_3 (End)	S_3	S_3	S_3	0	0	0	0	0

Fig d: State table defining the control unit of the gcd processor

We can identify a set of states of CU by examining the behavior defined in fig (a). A start state So is entered when Reset becomes 1, this state also loads X and Y into the DP registers. The subsequent actions of gcd processor are either a swap or a subtraction, for which we define the states S1 and S2, respectively. A final state S3 is entered when gcd(X,Y) has been computed.

Fig (d) gives a Moore type state table defining the CU's behavior. Each state transition is deduced directly from the gcd procedure (fig a). If the input control signal (XR>0) = 0, indicating that the **while** loop should be skipped, a transition is made from So to S3, this yields the first next-state entry in the top row of fig d. If, on the other hand, (XR>0)=1, the while loop is entered and a transition is made to S1 to perform a swap if (XR≥YR) =0; otherwise the transition is to S2 to perform a subtraction. The latter case defines to the third entry of the state table, whose input combination is (XR>0) (XR≥YR)=11. Since a subtraction always follows a swap, all next state entries in the second row are S2. The corresponding active outputs are the two register-load signals Load XR and Load YR, along with Swap, which route the outputs of XR and YR to YR and XR respectively. The next states for state S2 are the same as those for So; the active outputs are Subtract, which the output XR-YR of the subtracter to XR, and Load XR. The final state S3 is assumed to be a "dead" state that is unaffected by all inputs (except Reset) and produces no active outputs.

CLASSICAL METHOD / DESIGN FOR THE CONTROL UNIT OF THE GCD PROCESSOR

The major steps of the classical design method are as follows:

1. Construct a P-row state table that defines the desired input-output behavior.
2. Select the minimum number p of D-type flip flop and assign a p-bit binary code to each state.
3. Design a combinational circuit C that generates the primary output signals $\{Zi\}$ and the secondary output $\{Di\}$ that must be applied to the flip flop.

We now apply this method to the design of the control unit CU for the gcd processor. We have already constructed the state table (fig a)

	Inputs (XR > 0) (XR >= YR)			Outputs				
State	0-	10	11	Subtract	Swap	Select XY	Load XR	Load YR
S_0 (Begin)	S_3	S_1	S_2	0	0	1	1	1
S_1 (Swap)	S_2	S_2	S_2	0	1	0	1	1
S_2 (Subtract)	S_3	S_1	S_2	1	0	0	1	0
S_3 (End)	S_3	S_3	S_3	0	0	0	0	0

Fig a: State table defining the control unit of the gcd processor

Since there are four states, we require two flip flops, whose outputs D1 Do = $y1y0$ define CU's internal states. We assign the binary patterns to the four states in the following way:

$$S0 = 00$$
$$S1 = 01$$
$$S2 = 10$$
$$S3 = 11$$

At this point we can construct a binary version of the state table, the excitation table as shown in (fig b)

Inputs		Present state		Next state		Outputs				
$(XR > 0)$	$(XR \geq YR)$	D_1	D_0	D_1^+	D_0^+	*Subtract*	*Swap*	*Select XY*	*Load XR*	*Load YR*
0	*d*	0	0	1	1	0	0	1	1	1
0	*d*	0	1	1	0	0	1	0	1	1
0	*d*	1	0	1	1	1	0	0	1	0
0	*d*	1	1	1	1	0	0	0	0	0
1	0	0	0	0	1	0	0	1	1	1
1	0	0	1	1	0	0	1	0	1	1
1	0	1	0	0	1	1	0	0	1	0
1	0	1	1	1	1	0	0	0	0	0
1	1	0	0	1	0	0	0	1	1	1
1	1	0	1	1	0	0	1	0	1	1
1	1	1	0	1	0	1	0	0	1	0
1	1	1	1	1	1	0	0	0	0	0

fig (b) Excitation table for the control unit of the gcd processor.

The DFF's characteristics equation $Di^+(t+1)=Di(t)$ defines the inputs $D1^+$ and Do^+ to the flip flops. CU's combinational logic C can now be derived from the excitation table. Suppose for instance that we use two level sum of product (SOP) equations which lead directly to the design of (fig c) . Note that all gates in an AND-OR SOP circuit can be changed to NAND's to produce a NAND-NAND realization of the original function.

$$D_1^+ = \overline{(XR > 0)} + (XR \geq YR) + D_0$$

$$D_0^+ = D_1.D_0 + \overline{(XR \geq YR)} \cdot \overline{D_0} + \overline{(XR > 0)} \cdot D_0$$

$$Subtract = D_1 \cdot \bar{D}_0 \qquad \overline{Subtract} = \overline{D_1 \cdot \bar{D}_0}$$

$$Swap = \bar{D}_1 \cdot D_0 \qquad \overline{Swap} = \overline{\bar{D}_1 \cdot D_0}$$

$$Select\ XY = \bar{D}_1 \cdot \bar{D}_0 \qquad \overline{Select\ XY} = \overline{\bar{D}_1 \cdot \bar{D}_0}$$

$$Load\ XR = \bar{D}_0 + \bar{D}_1 \qquad Load\ XR = \overline{D_0 \cdot D_1}$$

$$Load\ YR = \bar{D}_1$$

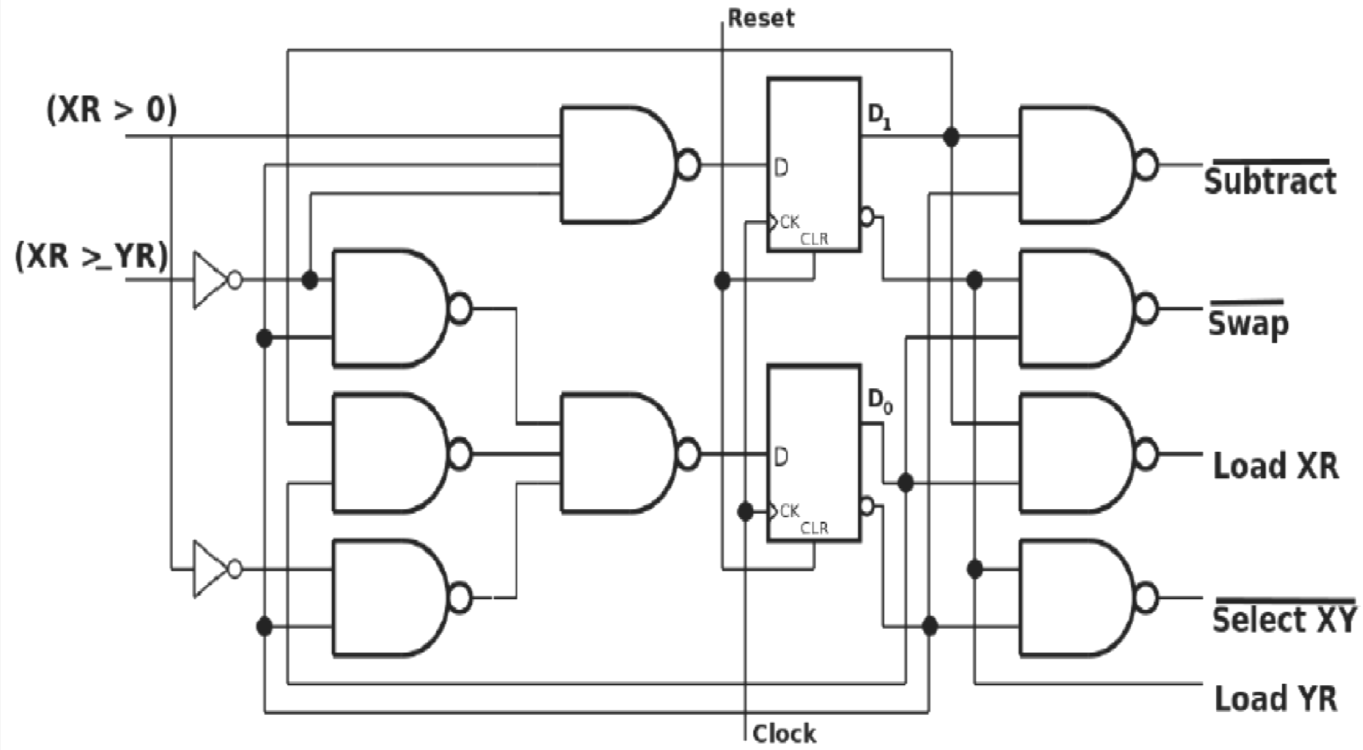


Fig c: All NAND Classical design for the control unit of the gcd processor

ONE HOT METHOD

Drawback of Classical Method: While the classical design method minimizes a control unit's memory elements, its effect on the amount of combinational logic C is less obvious. Futhermore, control unit designed by this technique tend to have a complicated random structure which makes a design debugging and subsequent maintenance of the circuit difficult.

One Hot Method: An alternative approach that simplifies the design process and gives combinational circuit (C) a regular and predictable structure is the one-hot method, so called because its binary state assignment always contains a single 1- the hot bit while all the remaining bits are 0. Thus the state assignment for a four state machine like the gcd processor takes the following form:

$$S0 = 0001$$
$$S1 = 0010$$
$$S2 = 0100 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$
$$S3 = 1000$$

In general, P flip flops are needed to represent P states, so the one hot method is restricted to fairly small values of P (here P=4 states).

A key feature of this technique is that the next state and output equations have a simple, systematic form and can be written down directly from the control unit's original symbolic state table. Because the binary pattern assigned to each state is in effect fully decoded, we can find out whether the machine is in state Si merely by inspecting the corresponding hot state variable Di. The classical method requires us to check all state variables to get this information.

Suppose that state Si in one hot design has the hot variable Di. Further suppose that $I_{j,1}$, $I_{j,2}$,……. $I_{j,nj}$ denote all input combinations that cause a state transition from S_j to S_i. Then each AND combination of the form $D_j I_{j,k}$ must make D_i=1.

Hence considering all such combinations that cause transitions to S_i, we can write

$D_i^+ = D_1(I_{1,1} + I_{1,2} + \ldots..+I_{1,n1}) + D_2 (I_{2,1} + I_{2,2}+\ldots\ldots+I_{2,n2}) +\ldots\ldots\ldots$ …….(2)

This immediately yields the SOP form

$D_i^+ = D_1 I_{1,1} + D_1 I_{1,2} + \ldots\ldots..+D_1 I_{1,n_1} + D_2 I_{2,1} + D_2 I_{2,2} + \ldots\ldots+D_2 I_{2,n_2} +\ldots\ldots\ldots$

which is practical to implement by an AND-OR or NAND-NAND circuit, provided that each state transition is determined by relatively few states and input variables, as is common in control unit behavior.

	Inputs (XR > 0) (XR >= YR)			Outputs				
State	0-	10	11	Subtract	Swap	Select XY	Load XR	Load YR
S_0 (Begin)	S_3	S_1	S_2	0	0	1	1	1
S_1 (Swap)	S_2	S_2	S_2	0	1	0	1	1
S_2 (Subtract)	S_3	S_1	S_2	1	0	0	1	0
S_3 (End)	S_3	S_3	S_3	0	0	0	0	0

Fig a: State table defining the control unit of the gcd processor

Consider the state table of (fig a) for the gcd processor's CU. State S_1 appears as a next state only for S_0 and S_2 in each case with the input combination (XR>0) ($\overline{XR \geq YR}$). Hence equation(2) becomes

$D_1+ = D_0(XR>0)\ (\overline{XR \geq YR}) + D_2\ (XR>0)\ (\overline{XR \geq YR})$

$=(D_0+D_2)\ (XR>0)\ (\overline{XR \geq YR})$

The primary output equations are even easier to derive for one hot design. If output signal Z_k is 1 (active) only in rows k, h for h=1,2,……m_k, then we have

$Z_k = D_{k,1} + D_{k,2} + \ldots\ldots\ldots + D_{k,mk}$

De Mogan's law of Boolean algebra allows us to rewrite this OR equation as

$$Z_k = \overline{\overline{D}_{k,1}\ \overline{D}_{k,2} \ldots\ldots \overline{D}_{k,mk}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

in which form it can be generated by a single NAND whose inputs are the complemented outputs of the flip flops. In the gcd processor case, output Load YR=1 in states S_0 and S_1 only; therefore

Load YR = $D_0 + D_1 = \overline{\overline{D}_0 \overline{D}_1}$

The entire set of next state and output equations obtained by applying (eqn 2) and (eqn 3) to the gcd processor's CU follows.

$$D_0^+ = 0$$

$$D_1^+ = D_0 \cdot (XR > 0) \cdot (\overline{XR \geq YR}) + D_2 \cdot (XR > 0) \cdot (\overline{XR \geq YR})$$

$$D_2^+ = D_0 \cdot (XR > 0) \cdot (XR \geq YR) + D_1 + D_2 \cdot (XR > 0) \cdot (XR \geq YR)$$

$$D_3^+ = D_0 \cdot (\overline{XR > 0}) + D_2 \cdot (\overline{XR > 0}) + D_3$$

$$Subtract = D_2 \qquad Load\ XR = D_0 + D_1 + D_2 = \overline{\overline{D_0} \cdot \overline{D_1} \cdot \overline{D_2}}$$

$$Swap = D_1$$

$$Select\ XY = D_0 \qquad Load\ YR = D_0 + D_1 = \overline{\overline{D_0} \cdot \overline{D_1}}$$

A NAND implementation of these equations appears in (fig b). Note that the asynchronous Reset line must set D_0 to 1 and all other state variables to 0.

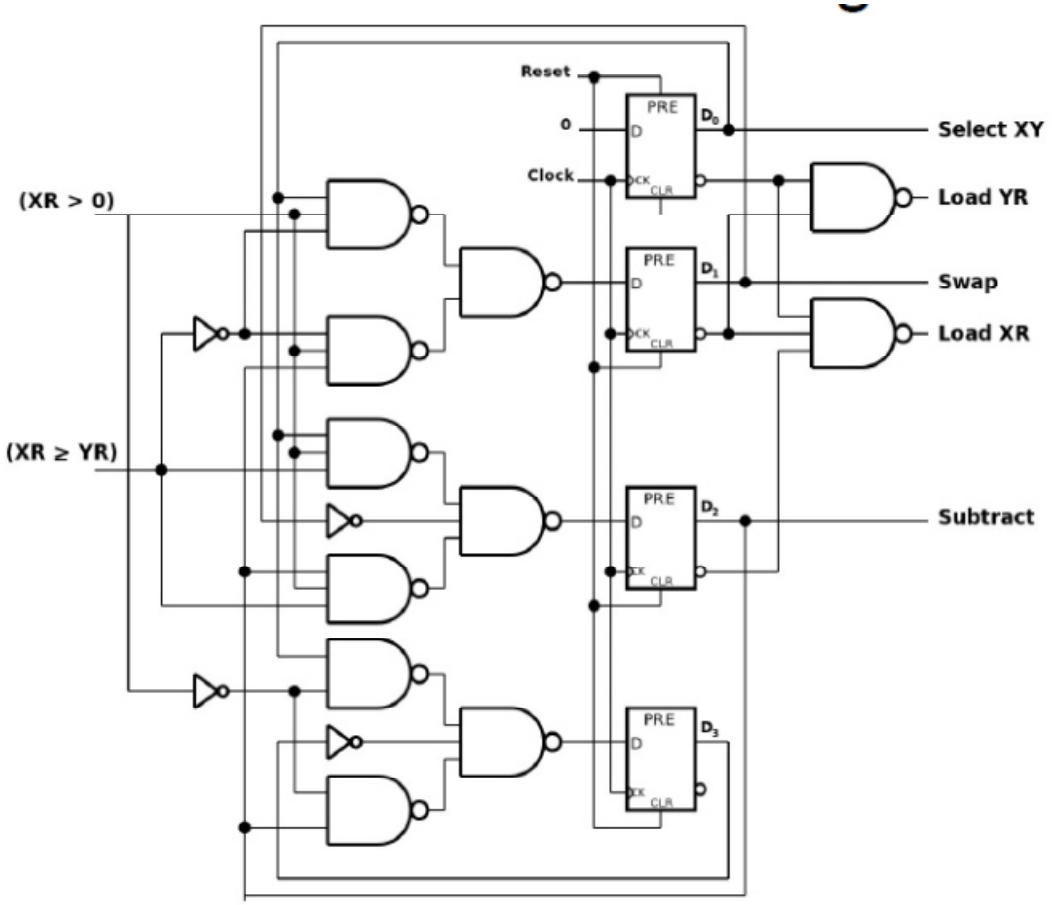


Fig b: All NAND one hot design for the control unit of the gcd processor

The steps of the one hot design method for a Moore machine can be summarized as follows

1. Construct a P-row state table that defines the desired input-output behavior
2. Associate as separate D type Flip Flop Di with each state S_i and assign the P-bit one hot binary code D1, D2,……$D_{i-1,}$ Di,$D_{i+1,}$ ….D_P = 0,0,……0,1,0…….0 to Si.
3. Design a combinational circuit C that generates the primary and secondary output signal $\{Di\}$ and $\{Z_k\}$ respectively. Di^+ is defined by the logic equation
   $Di^+ = \sum_{i=1}^{P} Di(Ij, 1 + Ij, 2 + Ij, nj)$
   where Ij,1, Ij,2,…..Ij,nj denote all input combinations that cause a transition from Sj to Si. If Zk=1 (active) only in rows k,h for j=1,2,…..mk then zk is defined by
   zk=Dk,1+Dk,2+….Dk,mk=

Microprogrammed Control Unit

Introduction:

Microprogramming is a method of control unit design in which the control signal selection and sequencing information is stored in a ROM or RAM called as *control memory* CM. The control signals to be activated at any time are specified by a microinstruction which is fetched from CM in much the same way an instruction is fetched from main memory. Each microinstruction implicitly or explicitly specifies the next microinstruction to be used, thereby providing the necessary information for microoperation sequencing.

A set of related microinstruction forms microprogram. Microprogram can be changed relatively easily by changing the contents of CM, hence microprogramming gives control unit more flexibility than hardwired technique.

Disadvantages:

Hardware cost is increased due to Control memory and its access circuitry.

Time required is more as microinstructions are accessed from CM.

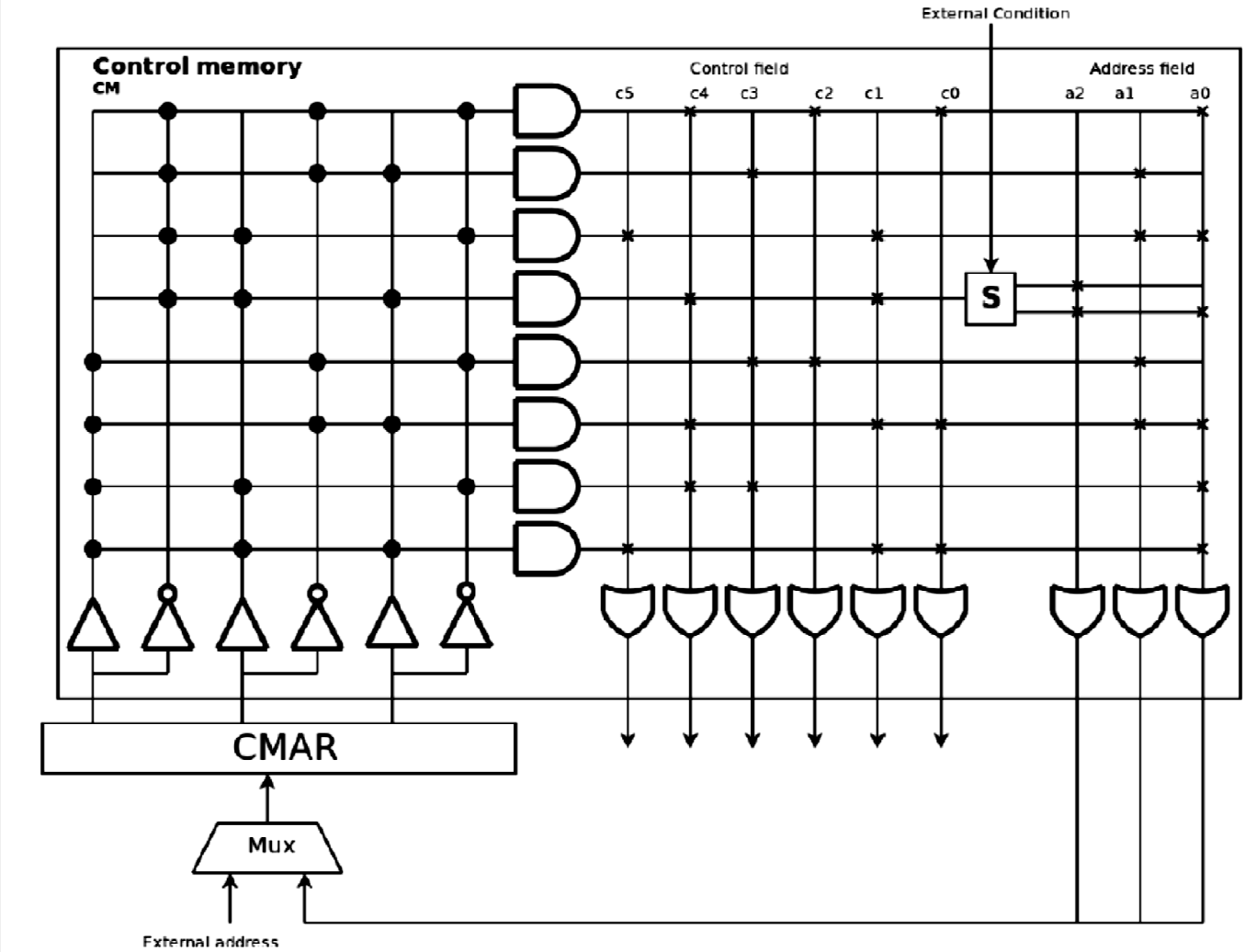
Due to this Microprogrammed CU is not used in RISC and high speed processor; it is used in CISC processor.

Micro programmed Control Unit Organization:

A Microinstruction has two parts

(1) A set of control fields that specify the control signals to be activated

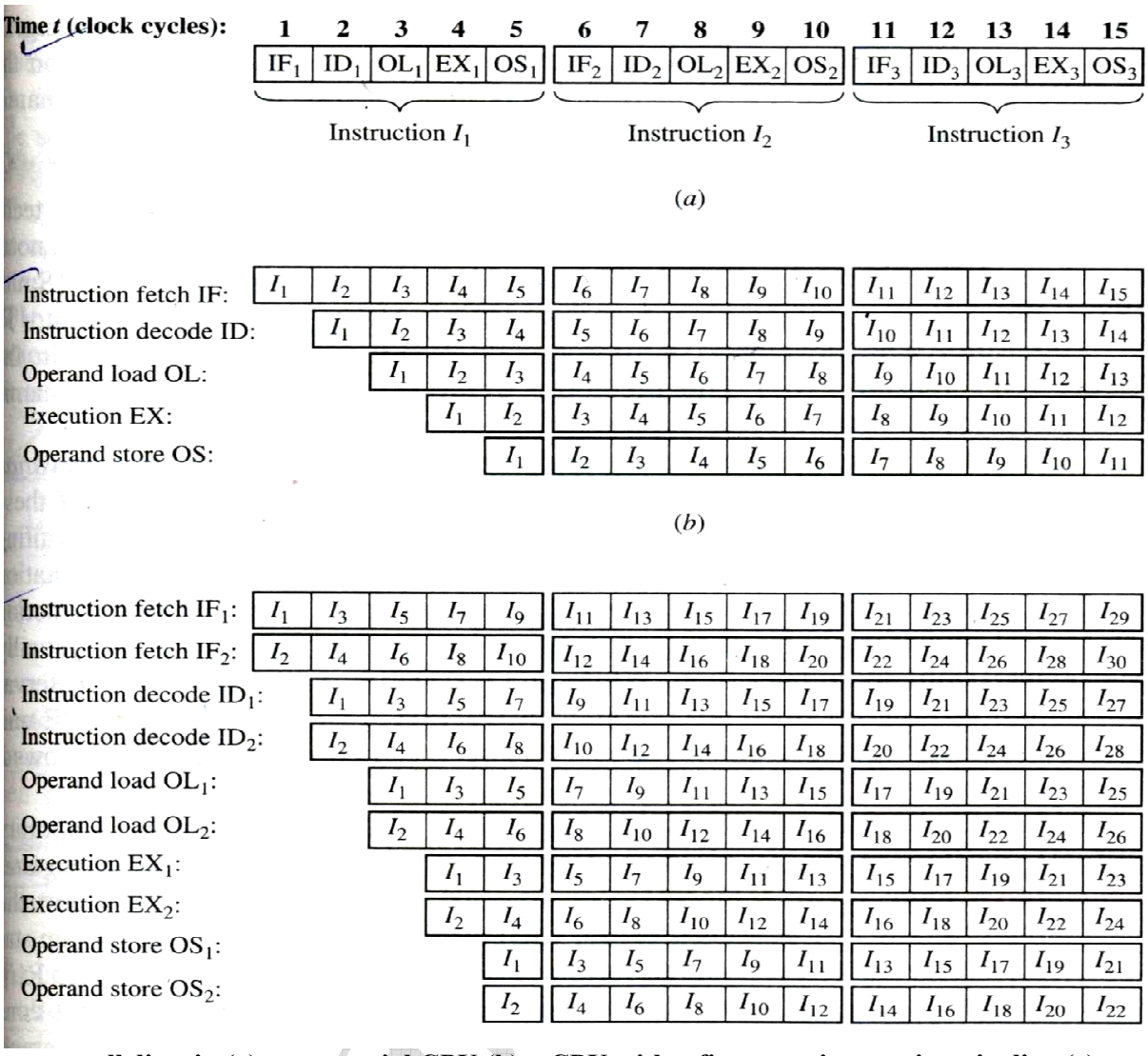
(2) An address field that contains the address in CM of the next microinstruction to be executed.



In the original scheme proposed by Maurice V. Wilkes, the inventor of microprogramming, each bit k_i of a control corresponds to a distinct control line $c_{i..}$ When k_i=1 in the current microinstruction c_i is activated, otherwise c_i remains inactive. Above fig shown microprogrammed control unit designed in this style. The control memory CM is implemented by a ROM. The left part (AND plane) of the ROM decodes an address obtained from the control memory address register (CMAR). Each address selects a particular row in the right part (OR plane) of the ROM that contains a microinstruction composed of a 6 bit control field and a 3-bit address field. When the top most row which represents the microinstruction with address 000, is selected, the control signal c_o, c_2 and c_4 are activated, as indicated by the Xs in the control field. At the same time, the contents of the address field $a_2a_1a_0$ =001 are sent to the CMAR, where they are stored and used to address the next microinstruction to be executed.

The CMAR can be loaded from an external source as well as from the address field of a microinstruction in the CM. The external source typically provides the starting address of a microprogram in the CM. A specific microprogram prestored in CM executes each instruction of a microprogrammed CPU. The instruction's opcode after suitable encoding, provides the starting address for its microprogram.

Superscalar Processor or Computer



Maximum parallelism in (a) a sequential CPU (b) a CPU with a five stage instruction pipeline (c) a superscalar CPU with two five stage instruction pipelines.

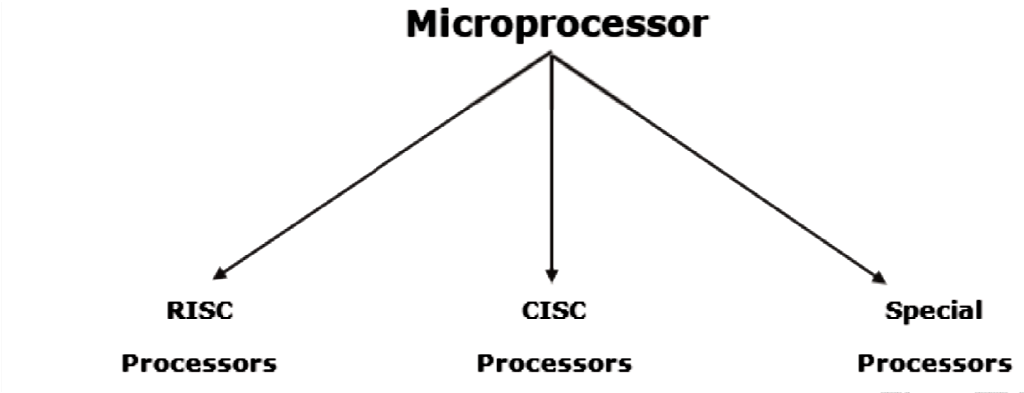
A superscalar processor is a CPU that implements a form of parallelism called instruction-level parallelism within a single processor. In contrast to a scalar processor that can execute at most one single instruction per clock cycle, a superscalar processor can execute more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to different execution units on the processor. It therefore allows for more throughput (the number of instructions that can be executed in a unit of time) than would otherwise be possible at a given clock rate. Each execution unit is not a separate processor, but an execution resource within a single CPU such as an arithmetic logic unit.

Super scalar processor has multiple execution units (E-Units), each of which is usually pipelined, so that they constitute a set of independent instruction pipelines. The CPU's program control unit PCU is designed to fetch and decode several instructions concurrently. It can *issue or dispatch* up to k instructions simultaneously to the various E-units where k, the *instructions issue degree* can be six or more using current technology. Above figure shows instruction processing abilities between three CPU organizations: a sequential processor, a basic pipelined processor, and a super scalar processor, all of which are executing the same instruction stream I1, I2, I3…. Assuming that each instruction requires a total of five cycles, we see that a single five stage instruction (k=1) offers a speedup of 5, while the two issue (k=2) super scalar design has a potential speedup of 10. Observe that at the start of cycle 15, the sequential CPU has completed only two instructions, whereas the pipelined and superscalar machine has completed 10 and 20 instructions, respectively; moreover, the superscalar CPU has already started processing instructions I_{21} through I_{30}

The PCU of a superscalar machine is responsible for determining when each instruction can be executed and for providing it with access to the resources it needs, such as memory operands, E-Units and CPU registers, in a prompt and efficient manner. To do so it must take following factors into account

Instruction type: For example, a floating-point add instruction has to be issued to a floating point E-unit and not to an integer E-unit.
E-unit availability: An instruction can be issued to a pipelined E-unit only if no collisions will result, as determined by pipeline's reservation table.
Data dependencies: To avoid conflicting use of registers, data dependency constraints among the operands of the active instructions must be satisfied.
Control dependencies: To maintain high performance levels, techniques are needed to reduce the impact of branch instructions on pipeline efficiency.
Program order: Instruction**s** must eventually produce results in the order specified by the program being executed. The results may however be computed out of order internally to improve the CPU's performance.

A microprocessor can be classified into three categories:



RISC Processor

RISC stands for **Reduced Instruction Set Computer**. It is designed to reduce the execution time by simplifying the instruction set of the computer. Using RISC processors, each instruction requires only one clock cycle to execute results in uniform execution time.

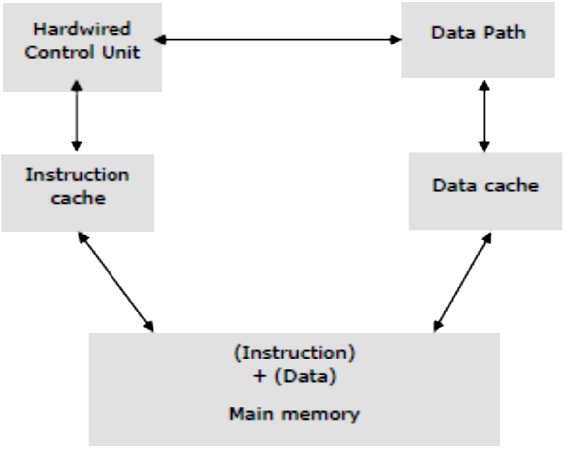
This reduces the efficiency as there are more lines of code, hence more RAM is needed to store the instructions. The compiler also has to work more to convert high-level language instructions into machine code.

Some of the RISC processors are:

- Power PC: 601, 604, 615, 620
- DEC Alpha: 210642, 211066, 21068, 21164
- MIPS: TS (R10000) RISC Processor
- PA-RISC: HP 7100LC

Architecture of RISC

RISC microprocessor architecture uses highly-optimized set of instructions. It is used in portable devices like Apple iPod due to its power efficiency.



Characteristics of RISC

The major characteristics of a RISC processor are as follows:

- It consists of simple instructions.
- It supports various data-type formats.
- It utilizes simple addressing modes and fixed length instructions for pipelining.
- It supports register to use in any context.
- One cycle execution time.
- "LOAD" and "STORE" instructions are used to access the memory location.
- It consists of larger number of registers.
- It consists of less number of transistors.

CISC Processor

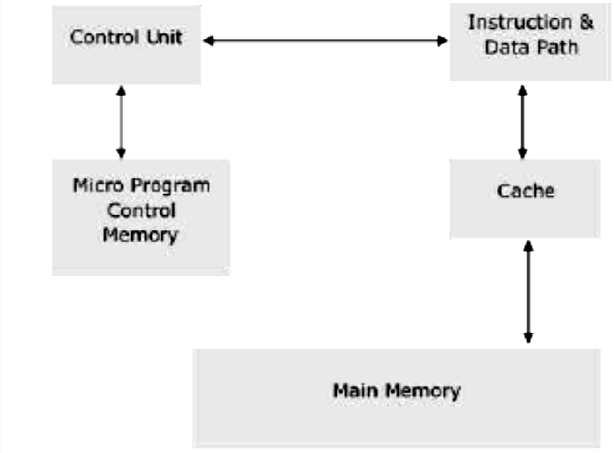
CISC stands for **Complex Instruction Set Computer**. It is designed to minimize the number of instructions per program, ignoring the number of cycles per instruction. The emphasis is on building complex instructions directly into the hardware. The compiler has to do very little work to translate a high-level language into assembly level language/machine code because the length of the code is relatively short, so very little RAM is required to store the instructions.

Some of the CISC Processors are:

- IBM 370/168
- VAX 11/780
- Intel 80486

Architecture of CISC

Its architecture is designed to decrease the memory cost because more storage is needed in larger programs resulting in higher memory cost. To resolve this, the number of instructions per program can be reduced by embedding the number of operations in a single instruction.



Characteristics of CISC

- Variety of addressing modes.
- Larger number of instructions.
- Variable length of instruction formats.
- Several cycles may be required to execute one instruction.
- Instruction-decoding logic is complex.
- One instruction is required to support multiple addressing modes.

Comparison of RISC and CISC

SN	RISC	CISC
1	Reduced Instruction Set Computer	Complex Instruction Set Computer
2	Emphasis on Software	Emphasis on hardware
3	Single Clock, reduced instruction only	Includes multiclock complex Instructions
4	Register to register: "LOAD" and "STORE" are independent instructions	Memory to Memory: "LOAD" and "STORE" incorporated in instructions
5	Low cycles per second, large code sizes	Small code size, high cycles per second
6	Spends more transistors on memory registers	Transistors used for storing complex instructions.
7	It reduces the cycles per instruction at the cost of the number of instructions per program	It attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction.
8	Simple instruction Format	Complex instruction to support high level language
9	Easy to construct a superscalar processor based on RISC architecture.	Difficult to design a superscalar processor based on CISC architecture.
10	Less number of addressing modes	More number of addressing modes
11	Large set of CPU registers	Small set of CPU registers

12	Maximum number of uses of the memory management unit (MMU) for a data address in an instruction.	The instruction are usually implemented by microcode.