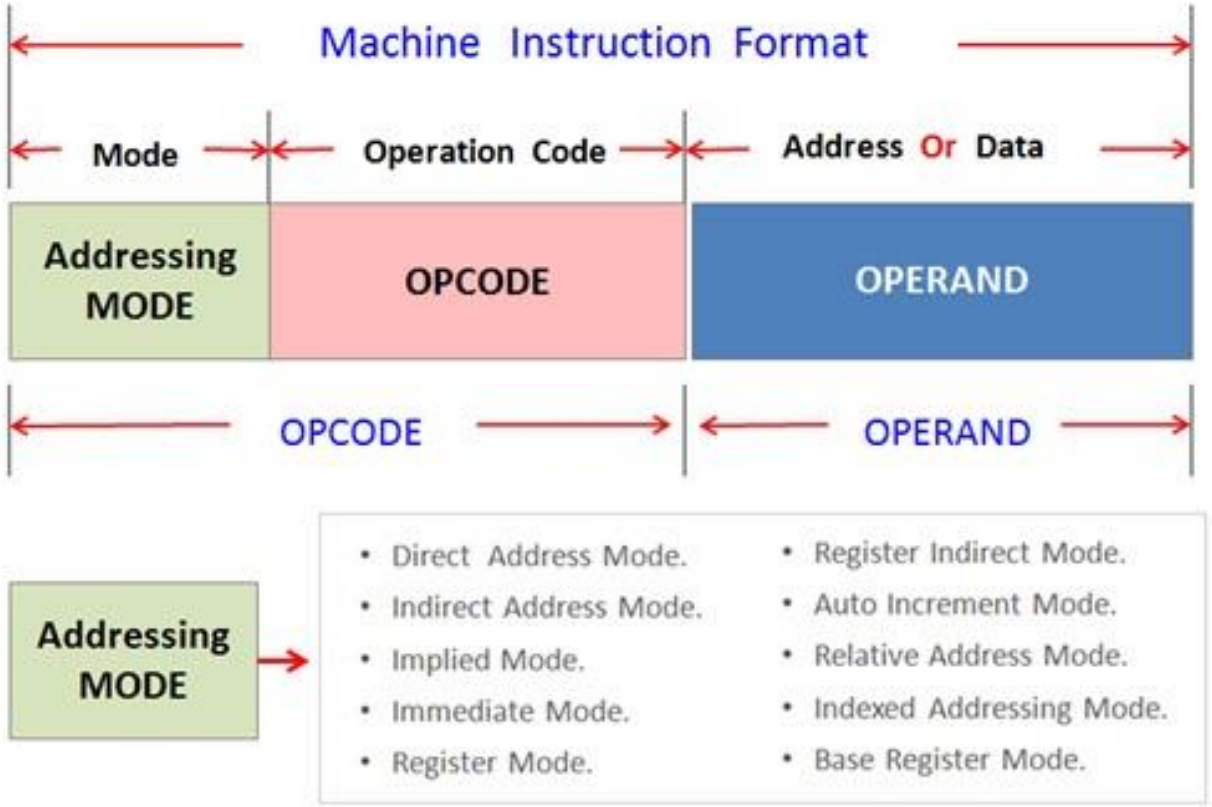
- **Arithmetic Logic Unit (ALU) is the main part the Central Processing Unit (CPU) in processing the instructions.**
- **The other parts of the CPU are the Control Unit (CU) and register unit.**
- **ALU simply executes the instructions in the order as dictated by the CU.**
- **The ALU performs the instruction execution on the operand data stored in registers**



In assembly language mnemonic form an opcode is a command such as MOV or ADD or JMP.

For example

MOV AL, 34h

The opcode is the MOV instruction. The other parts are called the 'operands'.

In this example, the operands are the register named AL and the value 34 hex.

"An operation performed on the data stored in registers is called micro-operation."

e.g.: The result of the micro-operation may replace the previous binary information of a register or may be transferred to another register. Other examples of micro-operations are add, subtract, shift, load and clear, etc.

The internal hardware structure of a computer is characterized by the following attributes:

1. **The types of micro-operations performed on the binary information stored in registers.**
2. **The control signals that initiate the sequence of micro-operations.**
3. **The set of registers it contains and their functions.**

The frequently used micro-operations in digital computers are classified into four categories:

1. **Register Transfer Micro-operations:** Transfer of binary information from one register to another.

2. **Arithmetic Micro-operations:** Arithmetic operations performed on the data stored in registers.

3. **Logical Micro-operations:** Bit manipulation operations on non-numeric data stored in registers.

4. **Shift Micro-operations:** Shift operations on data stored in registers.

REGISTERS

Registers are available in the following forms:

(a) Accumulator (AC).

(b) General-purpose registers,

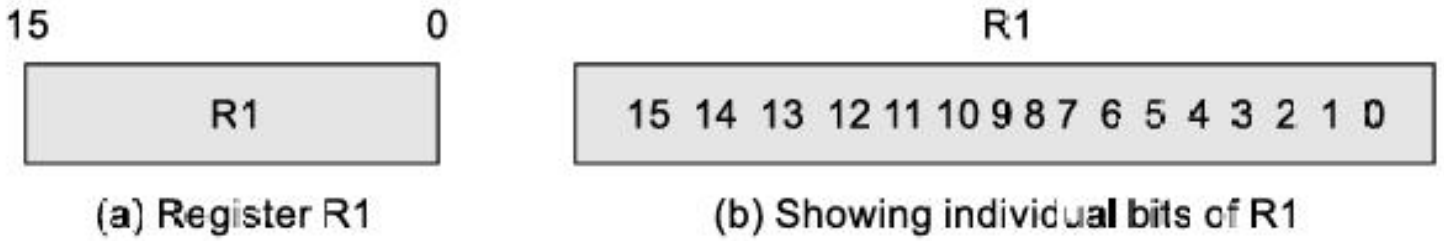
(c) Special-purpose registers.

Accumulator (AC)

The accumulator is a register which holds one of the operands before the execution of an instruction, and receives the result of most of the arithmetic and logical micro-operations. Thus, an accumulator is the most frequently used register. Some CPUs have a single accumulator and some have several accumulators.

General-purpose Registers

General-purpose registers or processor registers are used for storing data and intermediate results during the execution of a program. These registers are donated by capital letter R followed by some number. The individual flip-flops in an n-bit register are numbered in sequence from 0 to n-1, starting from 0 in the rightmost position and increasing the numbers towards the left. The Fig. shows the representation of 16-bit register in block diagram.



(a) Register R1 (b) Showing individual bits of R1

Special-purpose Registers

A CPU contains a number of special purpose registers for various purposes. Commonly used special-purpose registers and their functions are summarized below:

Register	*Function*
PC (Program Counter)	Holds the address of the next instruction to be executed.
IR (Instruction register)	Holds the instruction code (operation code) currently being executed.
SP (Stack Pointer)	Holds the address of the top element of the memory stack.
BR (Base Register)	Holds the starting address of the memory stack.
MAR (Memory Address Register)	Holds the address of the data item to be retrieved from the main memory.
MBR or DR (Memory Buffer Register or Data Register)	Holds the data item retrieved from the main memory.
SR or PSW (Status Register or Program Status Word)	Holds the condition code flags and other information that describe the status of the currently executing program.

Register Transfer Language (RTL)

Register Transfer Language (RTL) is the symbolic notation used to describe the micro-operation transfer between registers. The symbolic code R1 ←R2 indicates a transfer of the content of register R2 into R1. The transfer micro-operation means the content of source register R2 is copied into the destination register R1, but the content of R2 remains same.

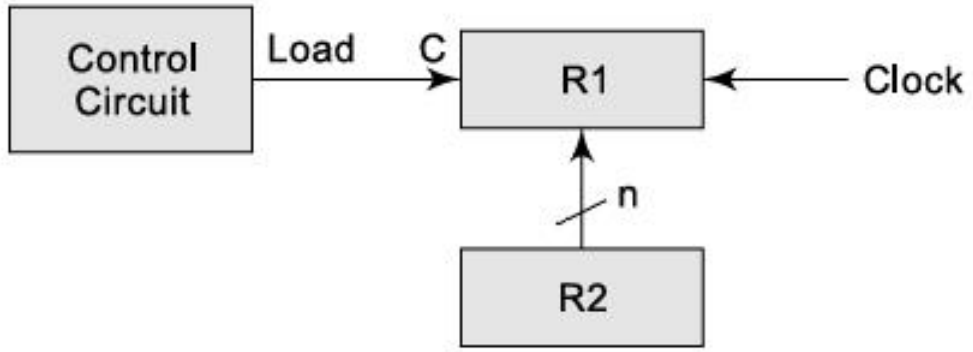
As far as internal hardware connectivity is concerned, Normally, the register transfer occurs under a predetermined control condition. This can be illustrated by means of an if-then symbolic code:

$$\text{If } (C = 1) \text{ then } (R1 \leftarrow R2)$$

A control function is nothing, but a Boolean variable that is equal to 1 or 0.

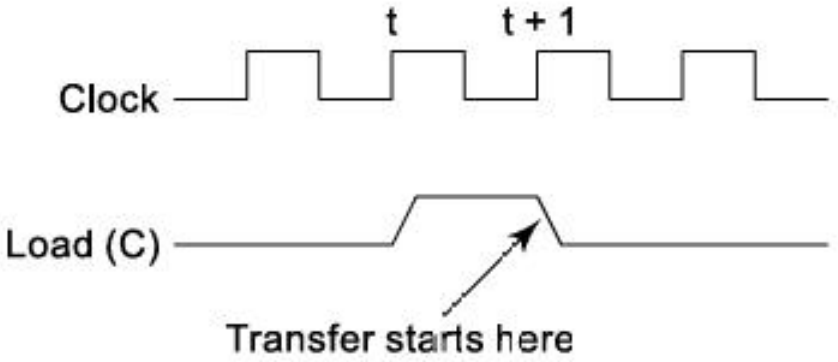
Thus the above symbolic code is equivalent to: $C: R1 \leftarrow R2$

A colon is used to terminate the control condition. This code means that the transfer micro-operation be performed only if C = 1. Fig.(a) shows the block diagram that illustrates the transfer from R2 to R1. Register R1 has a load control input C that is controlled by the control circuit. A common clock is used to synchronize all the activities in the total circuit for transfer operation.



(a) Block diagram
(n indicates the number of parallel lines in the bus)

As shown in the timing diagram Fig.(b) in the rising edge of a clock pulse at time t, the control section activates C. The next positive transition of the clock at time t + 1 finds the load input enabled and the data inputs of R1 are then loaded with the data outputs of register R2 in parallel. At time t + 1, C should be disabled; otherwise if C remains active, the transfer will occur with every clock pulse transition.



(b) Timing diagram

BUS TRANSFER

- **Many registers are provided in the CPU of a computer for fast execution. Therefore several paths must be provided to transfer information from one register to another. If a separate communication line is used between each register pair in the system, the number of lines will be excessive and thus cost of communication will be huge. Thus it is economical to have a common bus system for transferring information between registers in a multiple-register configuration.**

- **A bus system consists of a group of common communication lines, where each line is used to transfer one bit of a register at a time. Thus, a shared communication path consisting of one or more connection lines is known as a bus and the transfer of data through this bus is known as bus transfer. Sometimes, it is said that n-bit bus or n-line bus, the meaning of which is that the bus consists of n parallel lines to transfer n-bit of data all at a time.**
- ***We will present two ways to construct common bus system. One way is using multiplexers (simply known as MUXs), and another way is using tri-state buffers.***

Construction of a Common Bus Using MUXs

- The Fig. shows an n-line common bus system using multiplexers for register transfer, where four registers are used each of n-bit. This common bus is used to transfer a register's content to other register or memory at a single time. Two multiplexers are shown in the figure one for the low-order significant bit and another for the high order significant bit. The bus consists of n 4 X 1 multiplexers each having four data inputs, 0 through 3 and two common selection lines for all multiplexers.

S_1	S_0	*Register selected*
0	0	A
0	1	B
1	0	C
1	1	D

- General case Suppose an n-line bus system is to be constructed for k registers of n bits each. The number of MUXs needed to construct the bus is equal to n, the number of bits in each register. The size of each multiplexer must be k X 1, since it multiplexes k data lines.

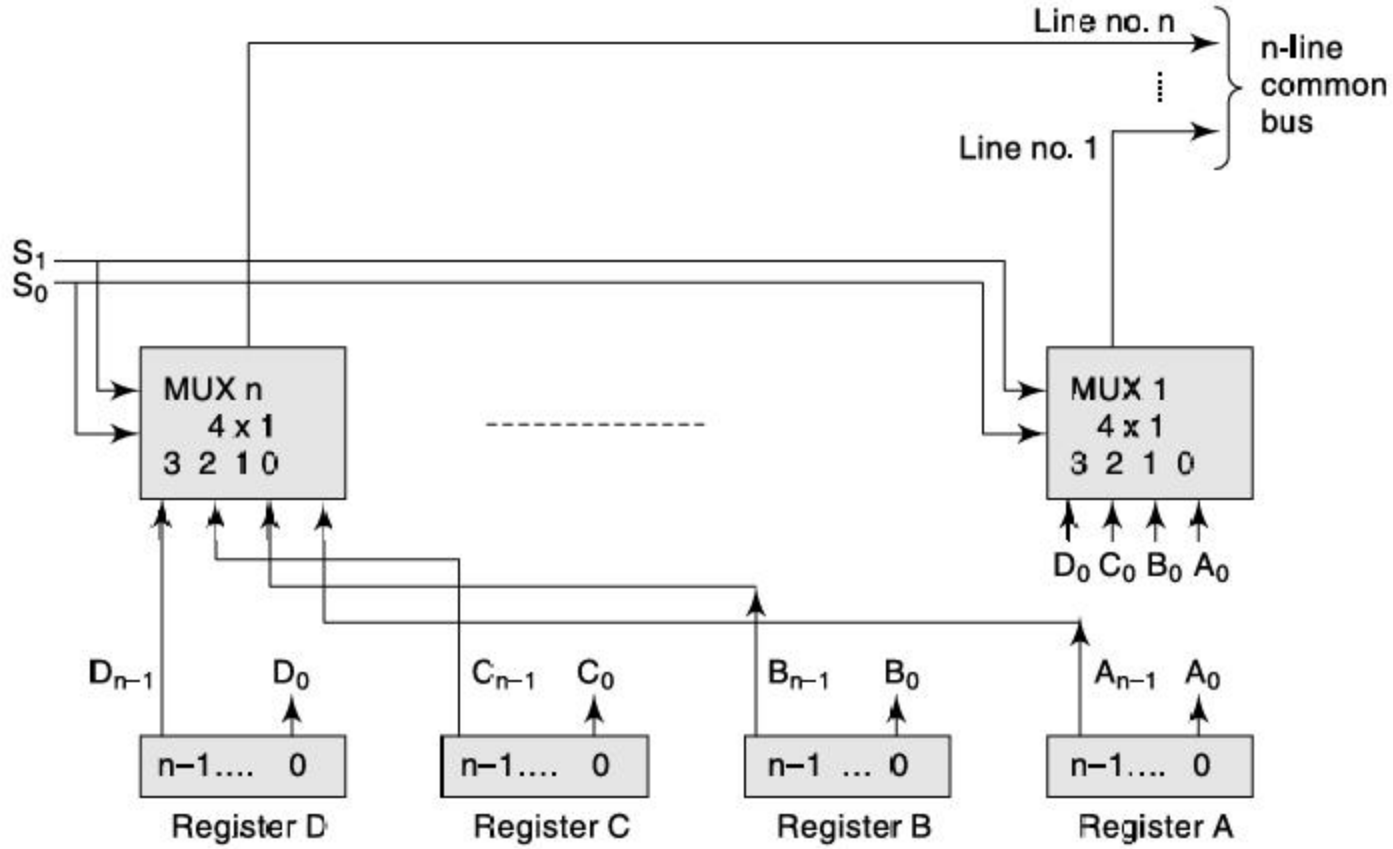
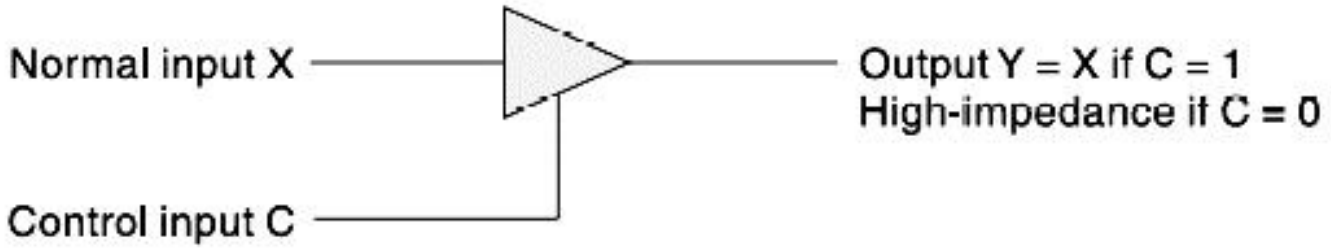


Figure *Bus system for four registers*

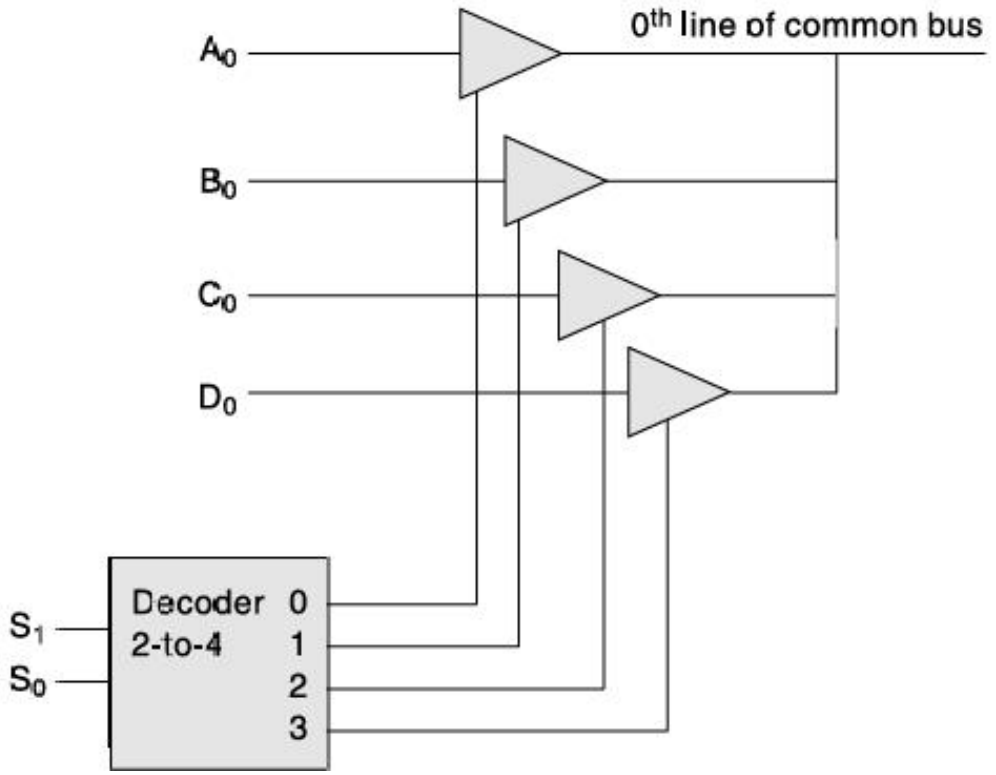
Construction of a Common Bus Using Tri-state Buffers

A tri-state gate is a digital circuit that exhibits three states out of which two states are normal signals equivalent to logic 1 and logic 0 similar to a conventional gate. The third state is a high-impedance state. The high-impedance state behaves like an open circuit, which means that no output is produced though there is an input signal and does not have logic significance. The gate is controlled by one separate control input C. If C is high the gate behaves like a normal logic gate having output 1 or 0. When C is low the gate does not product any output irrespective of the input values.



- The outputs of four buffers are connected together to form a single line of the bus. The control inputs to the buffers, which are generated by a common decoder, determine which of the four normal inputs will communicate with the common line of the bus.

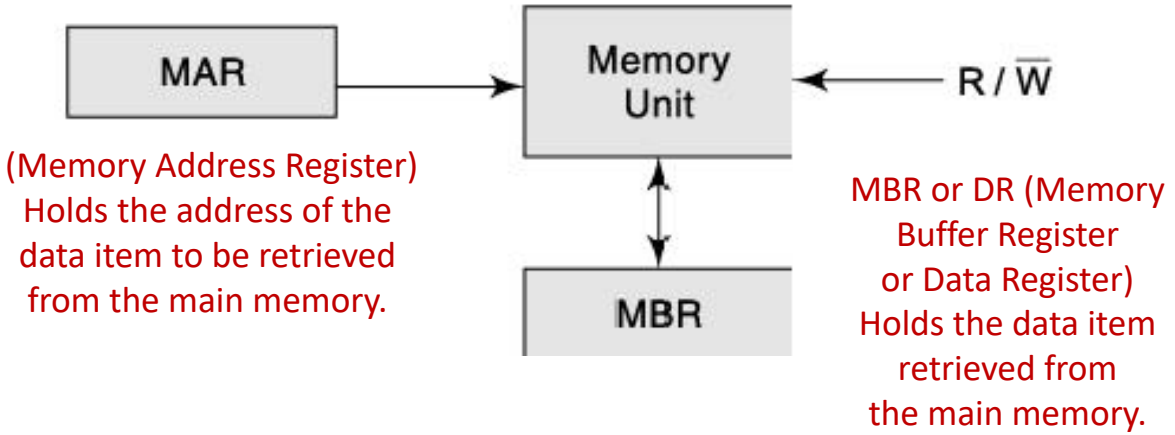
- For example, if select combination S1S0 is equal to 00,then 0^{th} output of the decoder will be activated, which then activates the top-most tri-state buffer and thus the bus line content will be currently A0, 0^{th} bit of A resistor.



General case Suppose an n-line common bus for k registers of n bits each using tri-state buffers needs to be constructed. We need n circuits with k buffers in each as shown in Fig. Therefore, total number of buffers needed is k * n. Only one decoder is required to select among the k registers. Size of the decoder should be $\log_2$k-to- k.

MEMORY TRANSFER

- **When the information is transferred from a memory word it is called a read operation and when the information is stored into a memory it is called write operation.**
- In both cases the memory word is specified by an address. This memory word is designated by the symbol M.
- A memory address is specified to select a particular memory word among many available words dur ing the transfer.
- Consider a memory unit that receives the address from a register, called the memory address register (MAR). The data from memory is transferred to another register, called memory buffer register (MBR) or data register (DR).



(Memory Address Register) Holds the address of the data item to be retrieved from the main memory.

MBR or DR (Memory Buffer Register or Data Register) Holds the data item retrieved from the main memory.

The read operation can be stated as:

$$\text{Read:} \quad \text{MBR} \leftarrow \text{M[MAR]}$$

This symbolic instruction causes a transfer of data into MBR from the memory word M selected by the address information in MAR.

The write operation can be stated as:

$$\text{Write:} \quad \text{M[MAR]} \leftarrow \text{R1}$$

By this symbolic instruction, a data word is transferred from R1 register to the memory word M selected by the register MAR.

ARITHMETIC MICROOPERATION

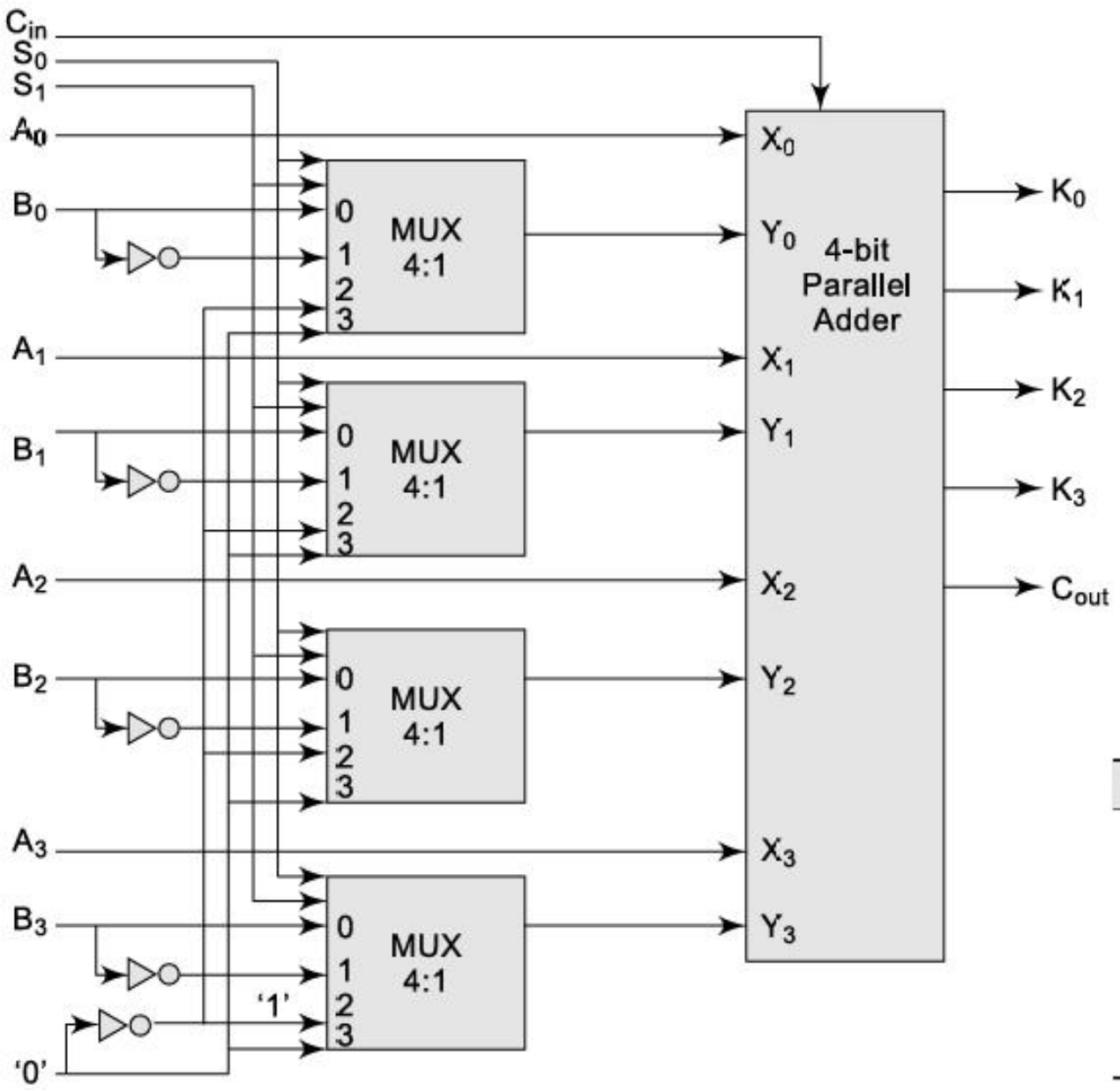
The basic arithmetic microoperations are addition, subtraction, increment, decrement and shift. The arithmetic addition microoperation is defined by the statement

$$R1 \leftarrow R2 + R3$$

This microoperation states that the content of R2 register is added to the content of R3 register and the result is stored into R1 register. In order to implement this operation with hardware we need three registers and a digital circuit to perform addition.

Symbolic notation	*Description*
$R1 \leftarrow R2 + R3$	Added contents of R2 and R3 transferred to R1
$R1 \leftarrow R2 - R3$	Contents of R2 minus R3 transferred to R1
$R1 \leftarrow R1 + 1$	Incrementing the content of R1 by 1
$R1 \leftarrow R1 - 1$	Decrementing the content of R1 by 1
$R1 \leftarrow \overline{R1}$	Complementing the content of R1 (1's complement)
$R1 \leftarrow \overline{R1} + 1$	2's complement the content of R1 (negate)
$R1 \leftarrow R2 + \overline{R3} + 1$	Content of R2 added with 2's complement of R3 (subtraction) and transferred to R1.

Arithmetic Unit



S_1	S_0	C_{in}	Y	$K = A + Y + C_{in}$	*Operation*
0	0	0	B	K=A + B	Addition
0	0	1	B	K=A + B + 1	Addition with carry
0	1	0	$\overline{B}$	K=A+ $\overline{B}$	Subtraction with borrow
0	1	1	$\overline{B}$	K=A + $\overline{B}$ +1	Subtraction
1	0	0	1	K=A – 1	Decrement
1	0	1	1	K=A	Transfer
1	1	0	0	K=A	Transfer
1	1	1	0	K=A + 1	Increment

Case 1: When $S_1\ S_0 = 00$.

In this case, the values of B are selected to the Y inputs of the adder. If $C_{in} = 0$, output K = A + B. If $C_{in} = 1$, output K = A + B + **1**. In both cases the microoperation addition is performed without carry or without adding the carry input.

Case 2: When $S_1\ S_0 = 01$.

The complement of B is selected to the Y inputs of the adder. If $C_{in} = 0$, output $K = A + \overline{B}$. This means the operation is subtraction with borrow. If $C_{in} = 1$, output $K = A + \overline{B} + 1$, which is equivalent to A + 2's complement of B. Thus this gives the subtraction A – B.

Case 3: When $S_1\ S_0 = 10$.

Here, all 1_s are selected to the Y inputs of the adder. This means Y = (1111), which is equivalent to 2's complement of decimal 1, that means, Y = –1. If $C_{in} = 0$, the output K = A –1, which is a decrement operation. If $C_{in} = 1$, the output K = A –1 + 1 = A. This causes the direct transfer of A to K.

Case 4: When $S_1\ S_0 = 11$.

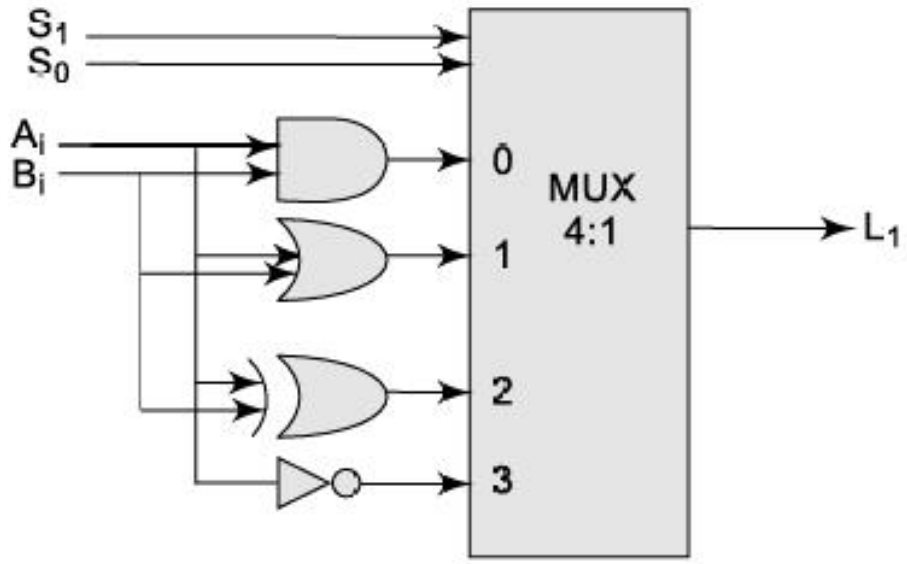
In this case, all 0_s are selected to the Y inputs of the adder. If $C_{in} = 0$, the output K = A, which is a transfer operation. If $C_{in} = 1$, output K = A + 1. This means the value of A is incremented by 1.

Observe that only seven different arithmetic microoperations are deduced, because the transfer operation is generated twice.

LOGIC UNIT

Now, we will design a logic unit that can perform the four basic logic microoperations: OR, AND, XOR and complement. Because from these four microoperations, all other logic microoperations can be derived.

S_1	S_0	*Output (L)*	*Operation*
0	0	$L = A \wedge B$	AND
0	1	$L = A \vee B$	OR
1	0	$L = A \oplus B$	XOR
1	1	$L = \overline{A}$	Complement of A



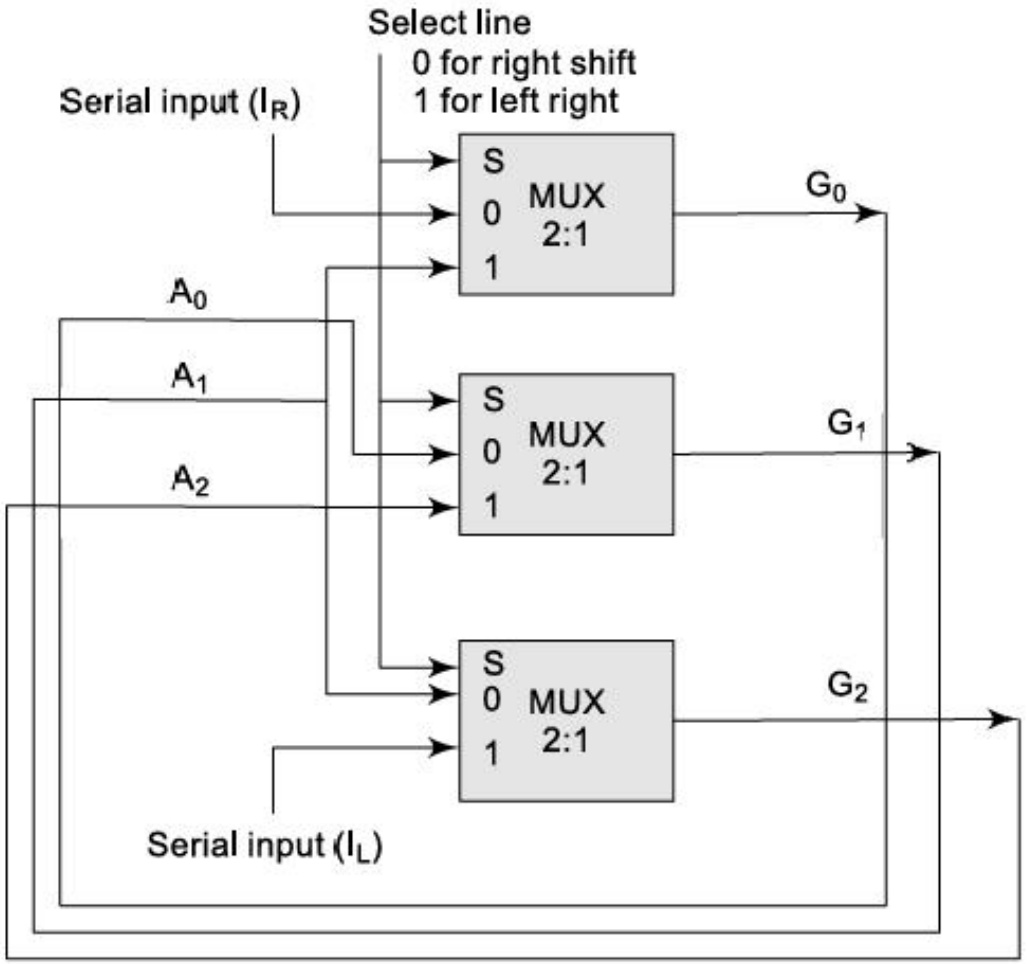
SHIFTER UNIT

Shifter unit is used to perform shift microoperations. Shift microoperations are used to transfer stored data serially. The shifting of bits of a register can be in either direction, left or right. Shift micro operations can be classified into three categories: (a) Logical , (b) Circular, (c) Arithmetic

In logical shift, all bits including sign bit take part in the shift operation. A bit 0 is entered in the vacant extreme bit position (left most or right most). As a result, the left-most bit is lost, if it is the left shift operation. Similarly, the right-most bit is lost, if it is the right shift operation.

In circular shift (also known as rotation operation), one bit shifted out from one extreme bit position enters the other extreme side's vacant bit position. No bit is lost or added

In arithmetic shift, sign bit remains unaffected and other bits (magnitude bits) take part in shift micro-operation As a result of the arithmetic left shift operation, the left-most bit of Circular shift microoperation magnitude part is lost and extreme right vacant bit is filled in with 0.



Selection line	*Output*		
S	G_0	G_1	G_2
0	I_R	A_0	A_1
1	A_1	A_2	I_L

A shifter unit for n data inputs and outputs requires n multiplexers.

ARITHMETIC LOGIC UNIT (ALU)

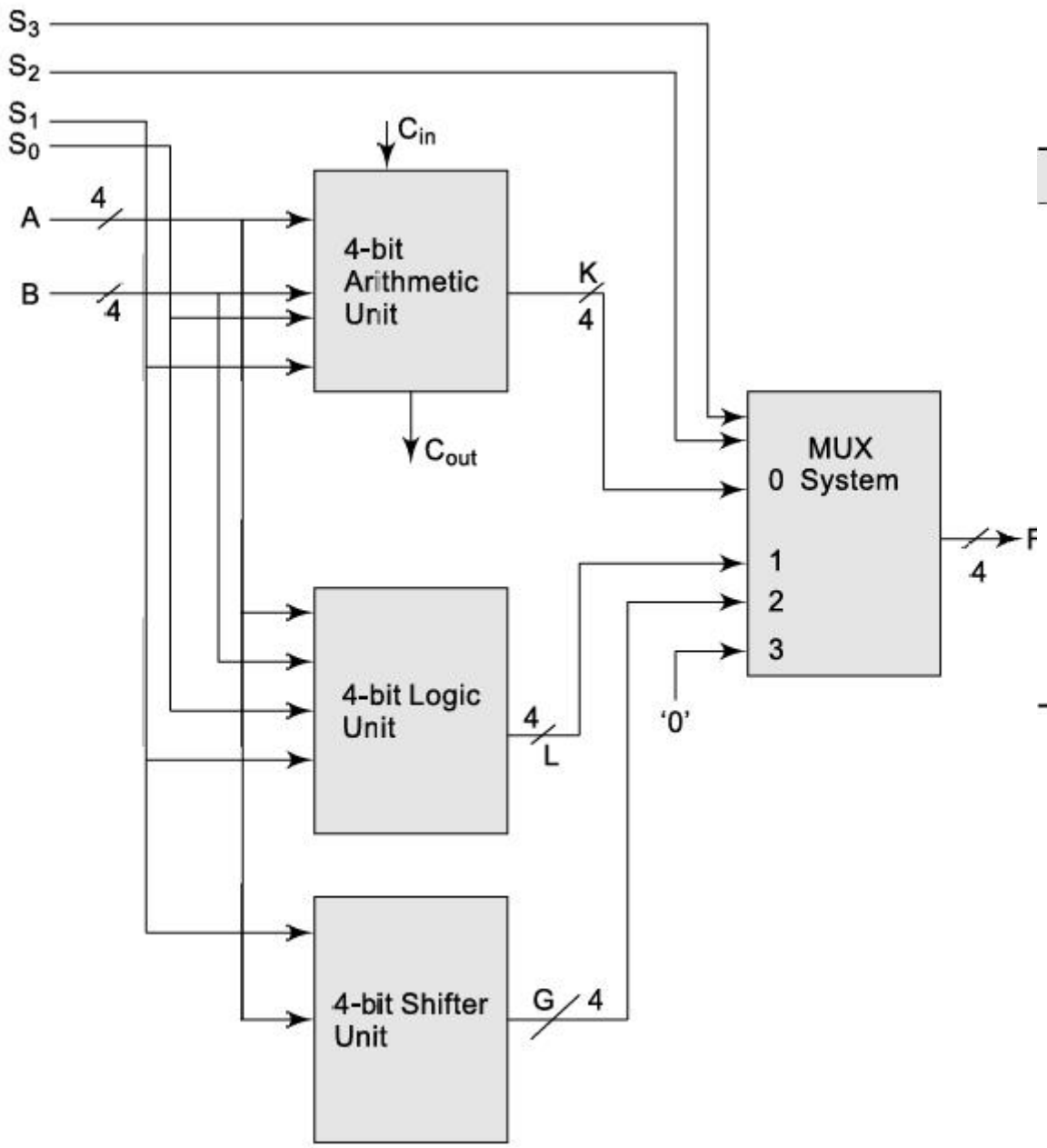


Figure *4-bit 14-function ALU*

S_3	S_2	S_1	S_0	C_{in}	*Output (F)*	*Operation*
0	0	0	0	0	F = A + B	Addition
0	0	0	0	1	F = A + B + 1	Addition with carry
0	0	0	1	0	F = A + B′	Subtraction with borrow
0	0	0	1	1	F = A + B′ + 1	Subtraction
0	0	1	0	0	F = A − 1	Decrement A
0	0	1	0	1	F = A	Transfer A
0	0	1	1	0	F = A	Transfer A
0	0	1	1	1	F = A + 1	Increment A
0	1	0	0	x	F = A ∧ B	AND
0	1	0	1	x	F = A ∨ B	OR
0	1	1	0	x	F = A ⊕ B	XOR
0	1	1	1	x	F = $\bar{A}$	Complement of A
1	0	0	x	x	F = lsr A	Shift right A into F
1	0	1	x	x	F = lsl A	Shift left A into F

- The set of four multiplexers each of 4:1 at output end chooses among arithmetic output in K, logic output in L and shift output in G. A particular arithmetic or logic microoperation is selected with selection inputs S_1 and S_0. The final output of the ALU is determined by the set of multiplexers with selection lines S_3 and S_2.
- The table lists 14 microoperations, 8 for arithmetic, 4 for logic and 2 for shifter unit.

Design of Control Unit

Control Unit performs the following responsibilities:

- **Instruction interpretation**

During the interpretation phase, the control unit reads instructions from the memory (using the PC register as a pointer). It then resolves the instruction type and addressing mode, gets the necessary operands and routes them to the appropriate functional units of the execution unit. Required signals are then issued to the different units of ALU to perform the desired operation and the results are routed to the specific destination. Thus, this phase is done in “instruction decoding” step of the instruction cycle.

- **Instruction sequencing**

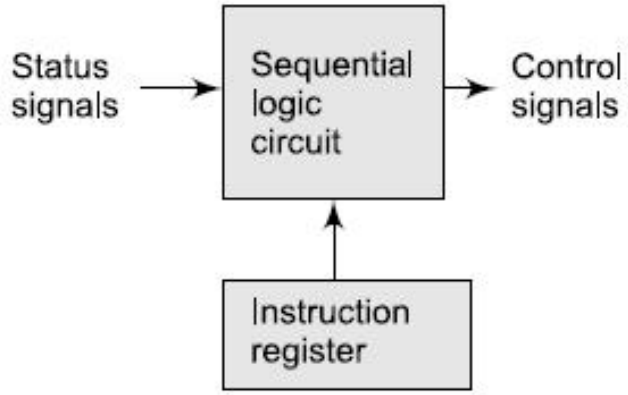
During the sequencing phase, the control unit finds the address of the next instruction to be executed and loads it into the PC. Thus, this phase is done in “instruction fetch” step of the instruction cycle.

DESIGN METHODS

Control units are designed in two different ways:

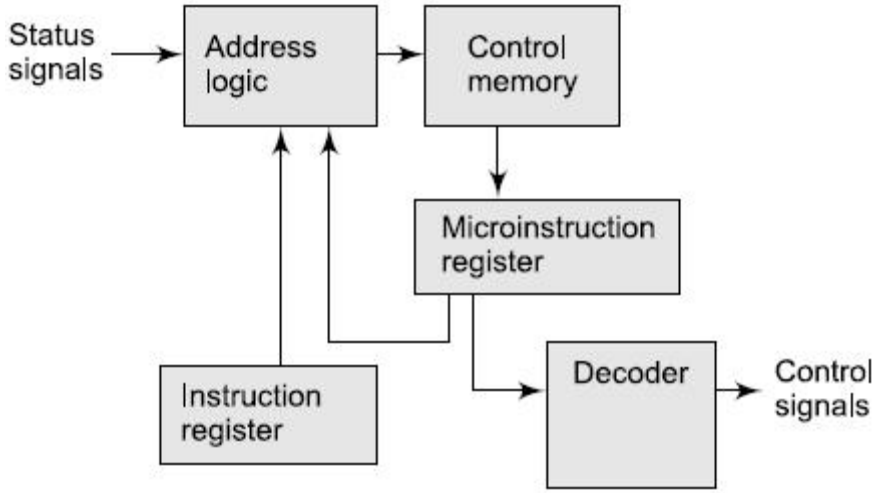
- **Hardwired approach**

When the control signals are generated using conventional sequential logic design techniques, the control unit is said to be hardwired. Logic gates, flip flops, decoders and other digital circuits are used to implement hardwired control organization.



- **Microprogramming approach**

In the microprogrammed approach, all control functions that can be simultaneously activated are grouped to form control words stored in a separate ROM memory called the control memory. From the control memory, the control words are fetched one at a time and the individual control fields are routed to various functional units to activate their appropriate circuits



Comparison between Two Methods

- The microprogramming approach is more expensive than hardwired approach.
- In microprogramming approach, a control ROM memory is needed.
- The main advantage of microprogramming is it provides a well-structured control organization. Control signals are systematically transformed into formatted words (microinstructions).With microprogramming, many additions and changes are made by simply changing the microprogram in the control memory, as the control signals are embedded in a kind of two-level software called firmware.
- A small change in the hardwired approach may lead to redesigning the entire system.
- Now-a-days microprogramming is accepted as a standard tool to design the control unit of a computer. For example, processors such as IBM 370, PDP-11 and Intel 80 x 86 family have a microprogrammed control unit. However, some olden day computers like Zilog's 16-bit microprocessor Z8000 still use a hardwired control unit.

Designing methods of hardwired control unit

There are three types of Hardwired Control Units.

1. State table method :

T - States	Instructions			
	I_1	I_2	...	I_N
T_1	$C_{1,1}$	$C_{1,2}$	...	$C_{1,N}$
T_2	$C_{2,1}$	$C_{2,2}$	...	$C_{2,N}$
...	...	...	...	...
T_M	$C_{M,1}$	$C_{M,2}$	...	$C_{M,N}$

$C_{1,1}$ means control signal to be produced in T-state(T_1) of Instruction (I_1)

- Here, each row represents the T-states and the columns represent the instructions.
- Every intersection of the specific column to each row indicates which control signal will be produced in the corresponding T- state of an instruction.
- Here the hardware circuitry is designed for each column(i.e. for a specific instruction) for producing control signals in different T-states.

Advantage –

It is the simplest method. This method is mainly used for small instruction set processors(i.e. in RISC processors).

Drawback –

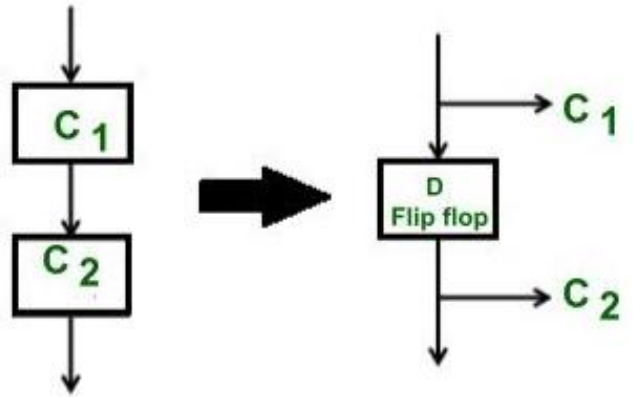
- In modern processors ,there is a very large number of instruction set. Therefore, the circuit becomes complicated to design, difficult to debug, and if we make any modifications to the state table then the large parts of the circuit need to be changed.

- Therefore ,this is not widely used for these kinds of processors.

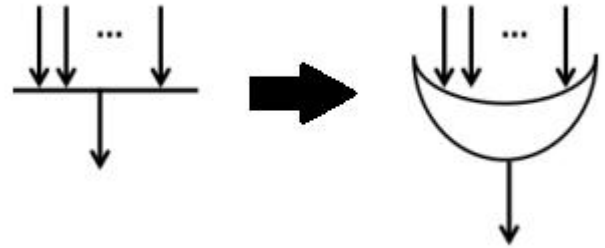
- There are many redundancies in circuit design like the control signals are required for fetching the instruction is common and which is repeated for N number of instruction. So the cost of circuitry design may increase.

2. Delay element method :

- Here the control unit behavior is represented in the form of a flowchart.
- Each step in the flowchart represents a control signal that needs to be produced for processing the instructions.
- If all the steps of the instructions are performed, this means the instruction is executed completely.
- Control signals perform micro-operations and each micro-operation requires one T-state.
- For the micro-operations which are independent, they are required to be performed in different T-state. Therefore, for every consecutive control signal an exactly 1-state delay is required, which can be produced with the help of D FF.
- Therefore. D Flip-Flops are inserted between every two consecutive control signals.

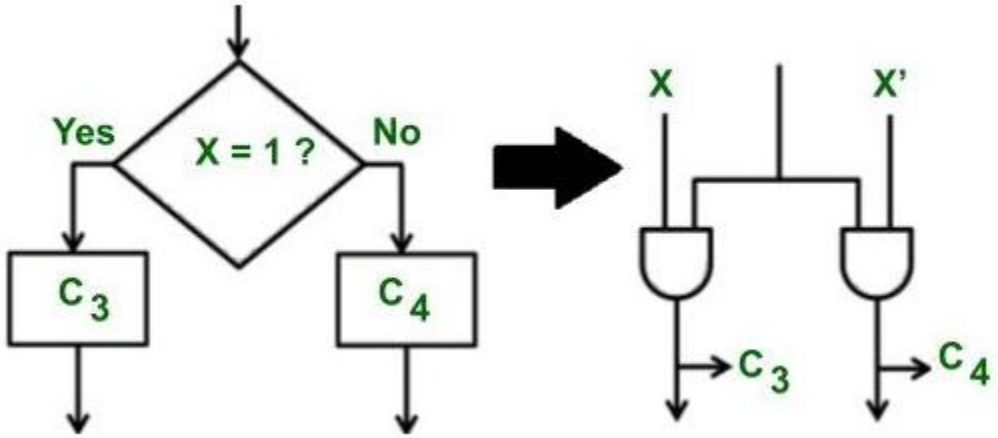


- As we can observe, the D FF is introduced between each pair of control signals .Therefore, after a control signal is generated, then the delay element before that control signal is not in use until before the next instruction required that control signal. Therefore, of all D Flip-Flops, only one will be active at a time. Therefore, this method is also known as **one hot method**.



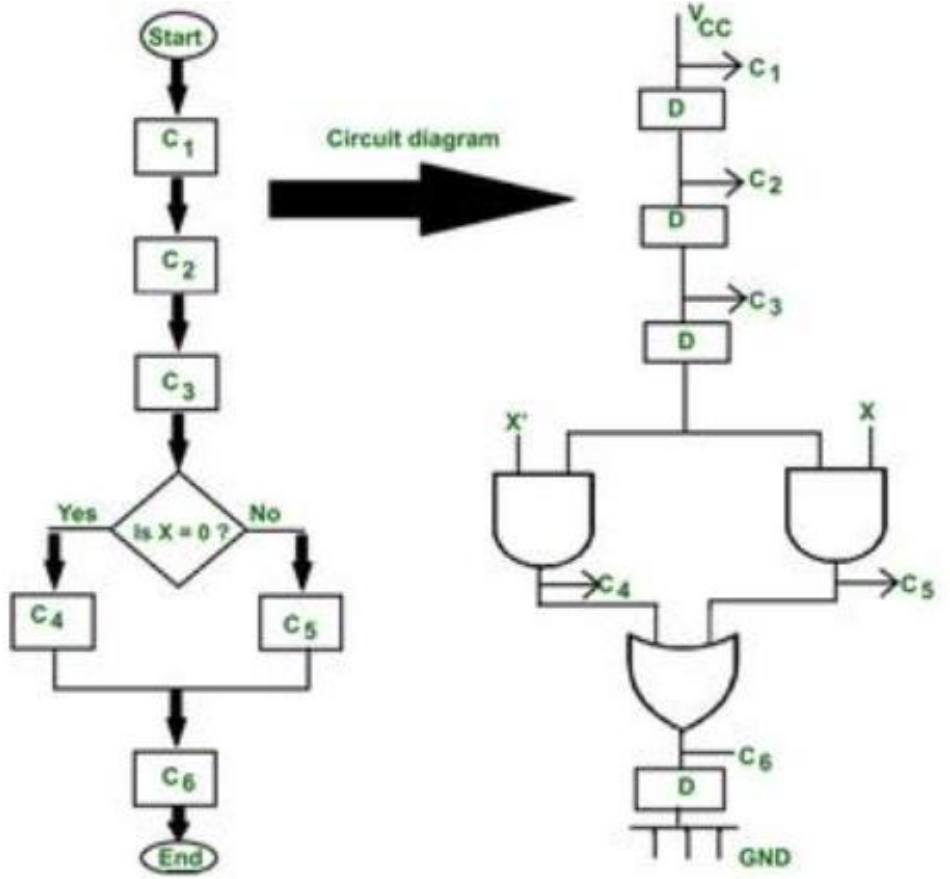
- In a flowchart, if there is a multiple entry point for control signal then to combine two or more paths, we use an OR gate.

- **A decision box is converted into a set of two complementing AND gates.**



Example –

Suppose the processor has two instructions add or subtract (Therefore an opcode of 1 bit is needed in which 0 opcode for add instruction and 1 for subtract is used.



Delay element method for generating control signals.

Flowchart design –

Say C_1,C_2,C_3 is the control signals for fetching the instruction. When X= 0, then C_4 control signal is produced (i.e. decoding) which is used for performing add operation, and when x=1, then control signal C_5 will be produced for performing the subtract operation. And c_6 control signal is used for storing the result and the process ends.

Circuit design –

Between two consecutive control signals which are independent, one delay element is introduced between them to produce a delay of 1 T state. The decision box is converted into and complemented AND gate circuit(i.e. if x= 0 then x'= 1, so, a c_4 control signal is generated.

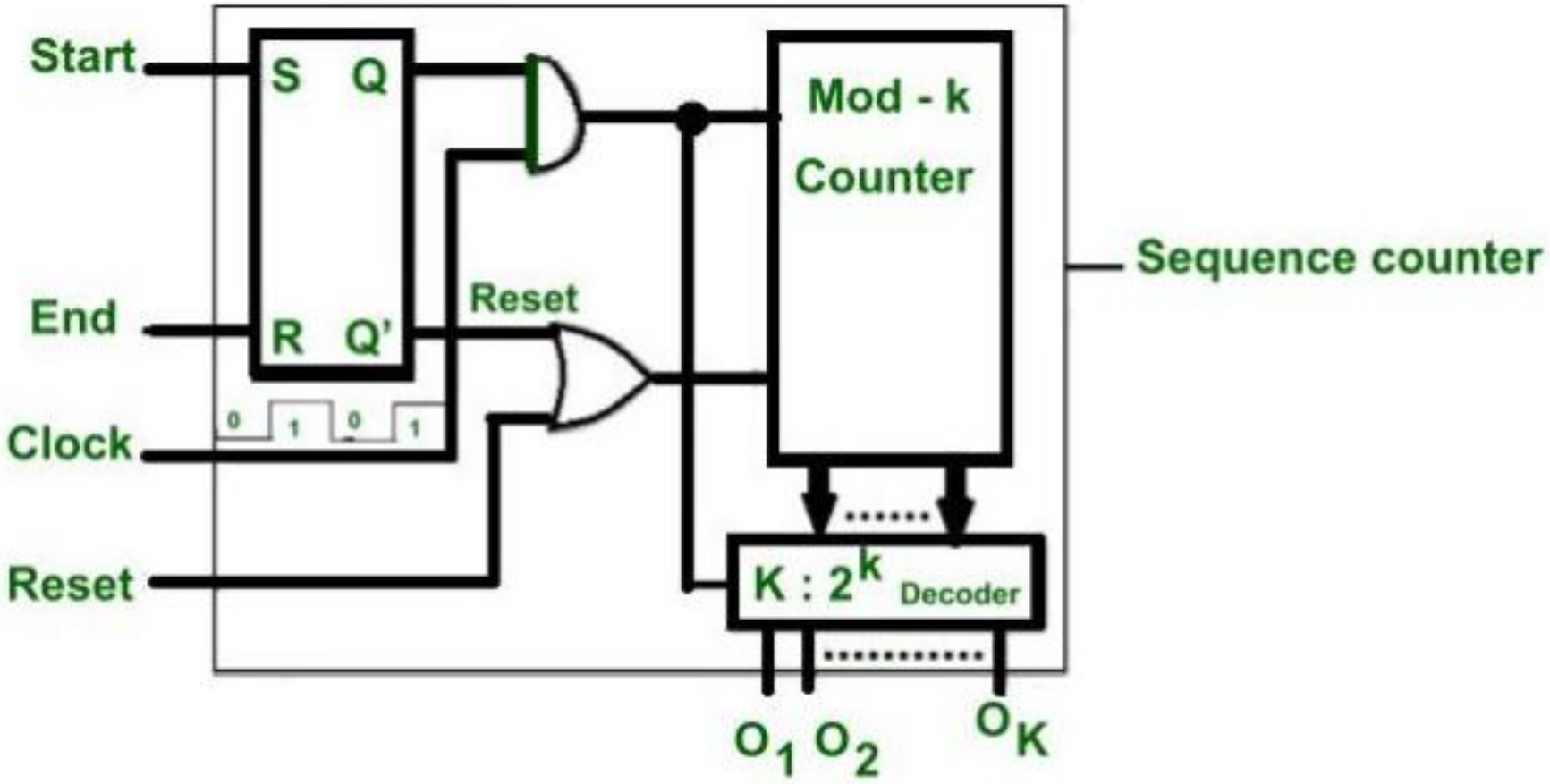
Advantage –

- This method has a logical approach ,therefore it helps to reduce the circuit complexity.
- For the common control signals which need to be generated in every instruction, for them only one circuitry can be designed .

Drawback –

- As the number of instructions increases , the number of D FF for generating delay is increased, so overall circuit complexity and cost increases.

3. Sequence counter method :

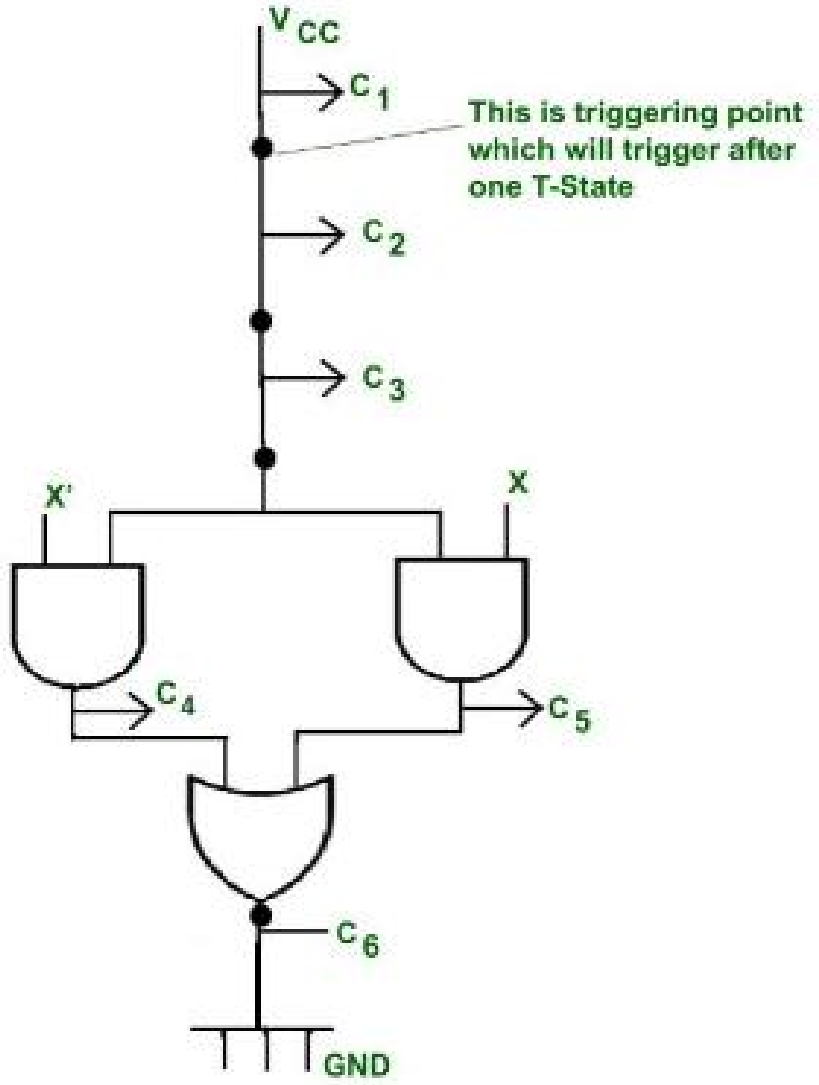


Sequence Counter method

- This is the most popular and most commonly used method for generating delays between every consecutive control signal. It's main advantage is that it uses the logical approach of flowchart and doesn't use the unnecessary number of D FF.
- First, a flowchart is designed to represent the behavior of a control unit.
- It is then converted into a circuit using the same method of AND & OR gates (as seen above in the Delay element method).

Example of 2 instructions used in the delay element method

- It is similar to the delay element method, but the only difference is that instead of unnecessary D Flip flops there are triggering points in the circuit. They are activated after a gap of one-one T-state.



Example of 2 instructions used in the delay element method

Working of Sequence counter circuit –

- Here one SR FF , one decode and one counter is used.

Clock	S	R	Q	Q'
0	X	X	Memory	
1	0	0	Memory	
1	0	1	0	1
1	1	0	1	0
1	1	1	Not used	

Truth table of SR FF

- When the instruction cycle starts, then start = 1**.**
- As we know, when Start = 1, because S is connected to Start, therefore Q becomes 1 and and Q' becomes 0.
- Here the **level triggering clock** is used. Therefore, when clock = 1 or high and Start=1, as both outputs are connected to AND gate, so if the resultant of both is 1 that will enable the counter and counter starts counting from 0 0 0 state. So the 0 0 0 state is decoded by a decoder and produces output O_1 , which will trigger the triggering point in the control circuit.

- **As the clock becomes high again after 1 T-state**. Therefore, when clock = 0, then the counter state is preserved(Q and Q') remains the same until the clock becomes high again. This makes sure that the counter changes its states after a gap of one-one T state.
- **Suppose the counter is 3 bits, it generates 2^3 = 8 states(000 001 ….. 111)** . The first count 0 0 0 is given to 3:8 decoder. It will active output number 1. This output is not a control signal but this will trigger the triggering point in the control unit circuit.
- As the clock becomes high again after a gap of one T state, therefore clock =1 and start = 1 ,then the counter is enabled and changes it state to 001 and the counter decodes the count and makes O_2 output high . And this will trigger a second triggering point in the circuit.
- All counting states are decoded in the same manner.

If the counter is of K bits then K:2^K decoder is required this can produce 2K outputs and that will trigger the 2K triggering points after a gap of 1-1 T-states in the circuit.

- When the instruction ends, the control signal is generated to make End pin = 1, and the counter is reset, so the next time, it begins from the first count(0 0 0).
- If reset pin =1, then the counter will reset and after that it will again start counting from 000 states.

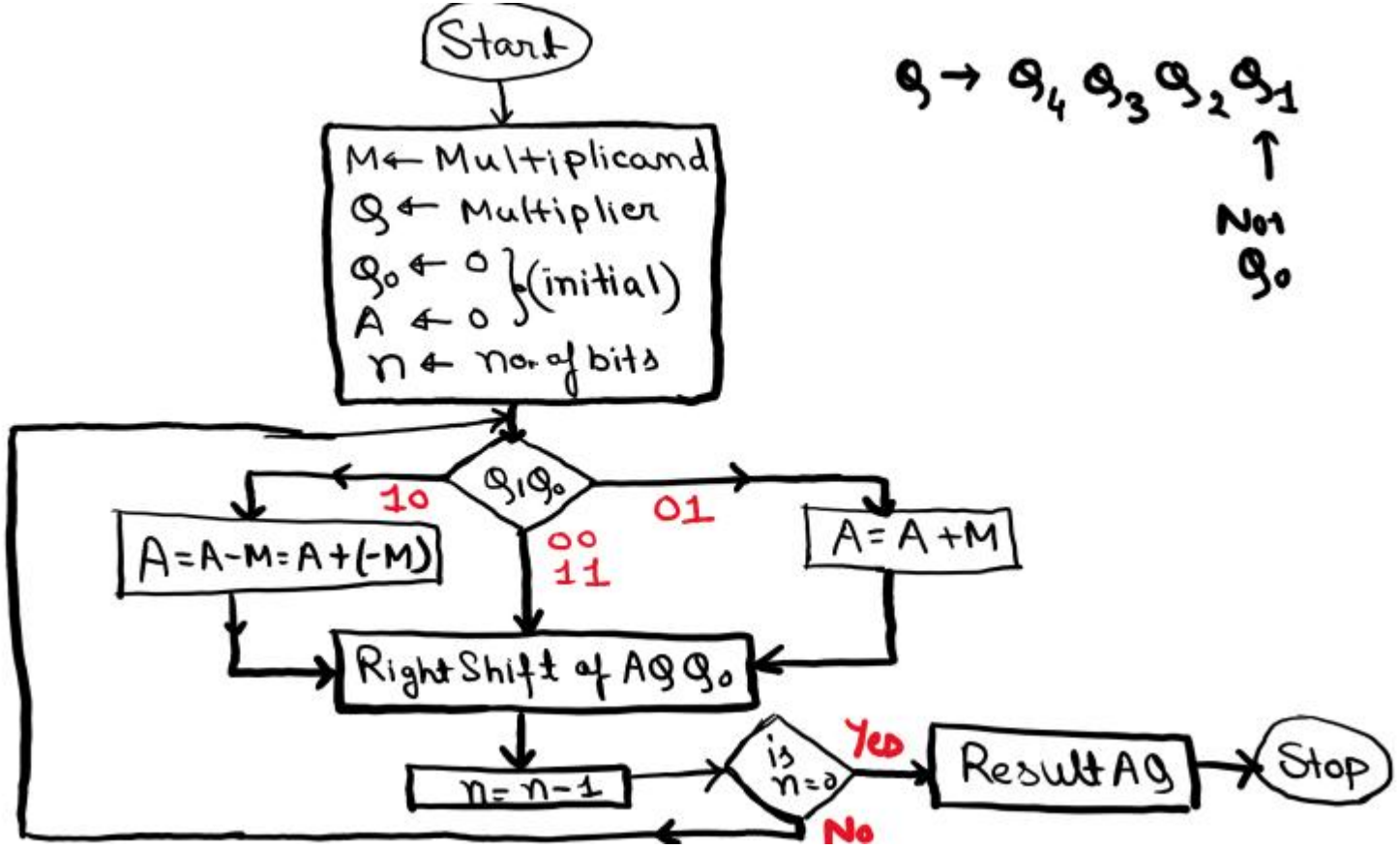
Advantages :

- Less number of flip-flops are used.

Disadvantages of hardwired control unit :

- In modern processors ,there is a very large number of instruction set. Therefore, the circuit becomes complicated to design, difficult to debug, and if we make any modifications then a large part of the circuit needs to be changed. Therefore, it is suited for RISC processors.

Multiplier Control Unit Let us take one example to illustrate the design procedure.



Step 1 : We know that Booth's procedure inspects a pair of multiplier bits and performs one of the following actions:

Multiplier bits inspected Q[i]	Q[i-1]	Action (in ith position)
0	0	None
0	1	Add M
1	0	Subtract M
1	1	None

Step 2 : the 4-bit register M will hold the multiplicand. The multiplier Q register is 5-bit wide. Initially, the high-order 4-bit of this register will hold the 4-bit multiplier. The least-significant bit of this register is initialized with the fictitious 0.

- **The 4–bit adder/subtractor unit is used to perform the operations A + M or A − M. The result produced by this** hardware **unit is always stored to the 4–bit accumulator A. Here, the accumulated partial product stored in (AQ) register pair is shifted right.**
- The L register is used to keep track of the iteration count.
- The final product are found in the registers A and Q, respectively. The 4-bit data buses - Inbus and Outbus, are used to transfer data into and out of the processing section respectively.

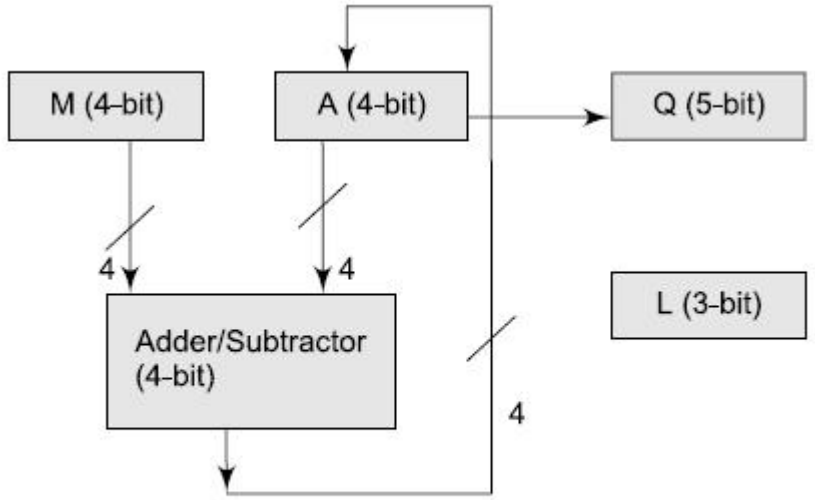


Figure 6.3 *Processing section for 4 × 4 Booth's multiplication*

Step 3: For 4 X 4 Booth's multiplication algorithm, a register transfer description is given

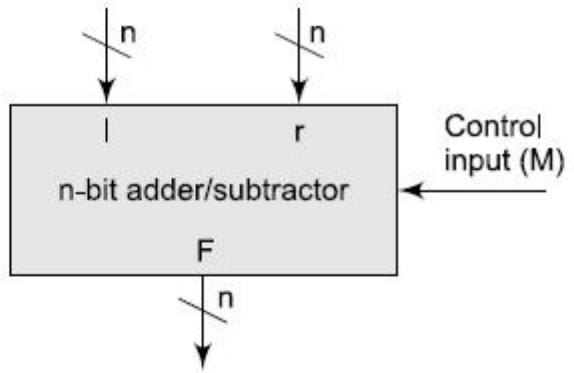
```
Registers: M[4], A[4], Q[5], L[3];
Buses: Inbus[4], Outbus[4];
START       A ← 0, M ← Inbus, L← 4
            Q[3:0] ← Inbus, Q[–1] ← 0;
LOOP        If Q[0:–1] = 01 then go to ADD
            If Q[0:–1] = 10 then go to SUB

               Go to RSHIFT;
ADD         A ← A + M;
            Go to RSHIFT;
SUB         A ← A – M;
RSHIFT      ASR (AQ), L ← L–1;
            If L ≠ 0 then go to LOOP
            Output = A;
            Output = Q[3:0];
HALT        Go to HALT;
```

Q [0: –1] is used to indicate the low-order 2 bits of the Q register (Initially Q[0] indicates the lsb of Q register and Q[–1] indicates a fictitious 0). Similarly, Q[3:0] indicates the high-order 4 bits of the Q register. The last step, Go to HALT, introduces an infinite loop after the algorithm is completed.

Step 4: The processing section contains three main elements:

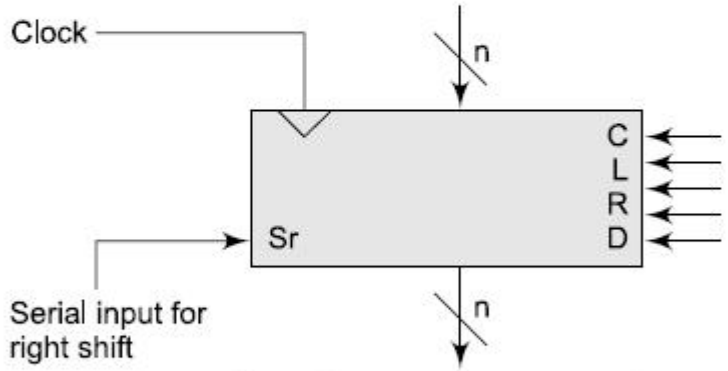
- 4-bit adder/subtarctor.
- General-purpose registers.
- Tri-state buffers.

The operational characteristics of these three elements are provided in Fig. By introducing the proper values to control inputs C, L, R and D, four operations (clear, parallel load, right shift and decrement) can be performed. A clock circuit synchronizes all these operations. The 4-bit adder/subtractor can be implemented using a 4-bit parallel adder chip and four XOR gates. To build a general-purpose register, standard flip-flops and gates can be used. The tri-sate buffers are used to control the data transfer to the outbus.



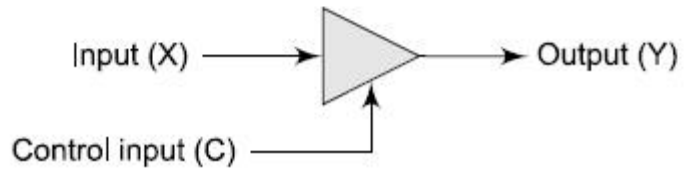
M	F
0	1 + r
1	1 – r

(a) : n-bit adder/subtractor



C L R D	Action
1 0 0 0	Clear
0 1 0 0	Load input data
0 0 1 0	Right shift
0 0 0 1	Decrement by one
0 0 0 0	No change

(b) : n-bit general purpose register



C	Y
1	X
0	No output

(c) : Tristate buffer

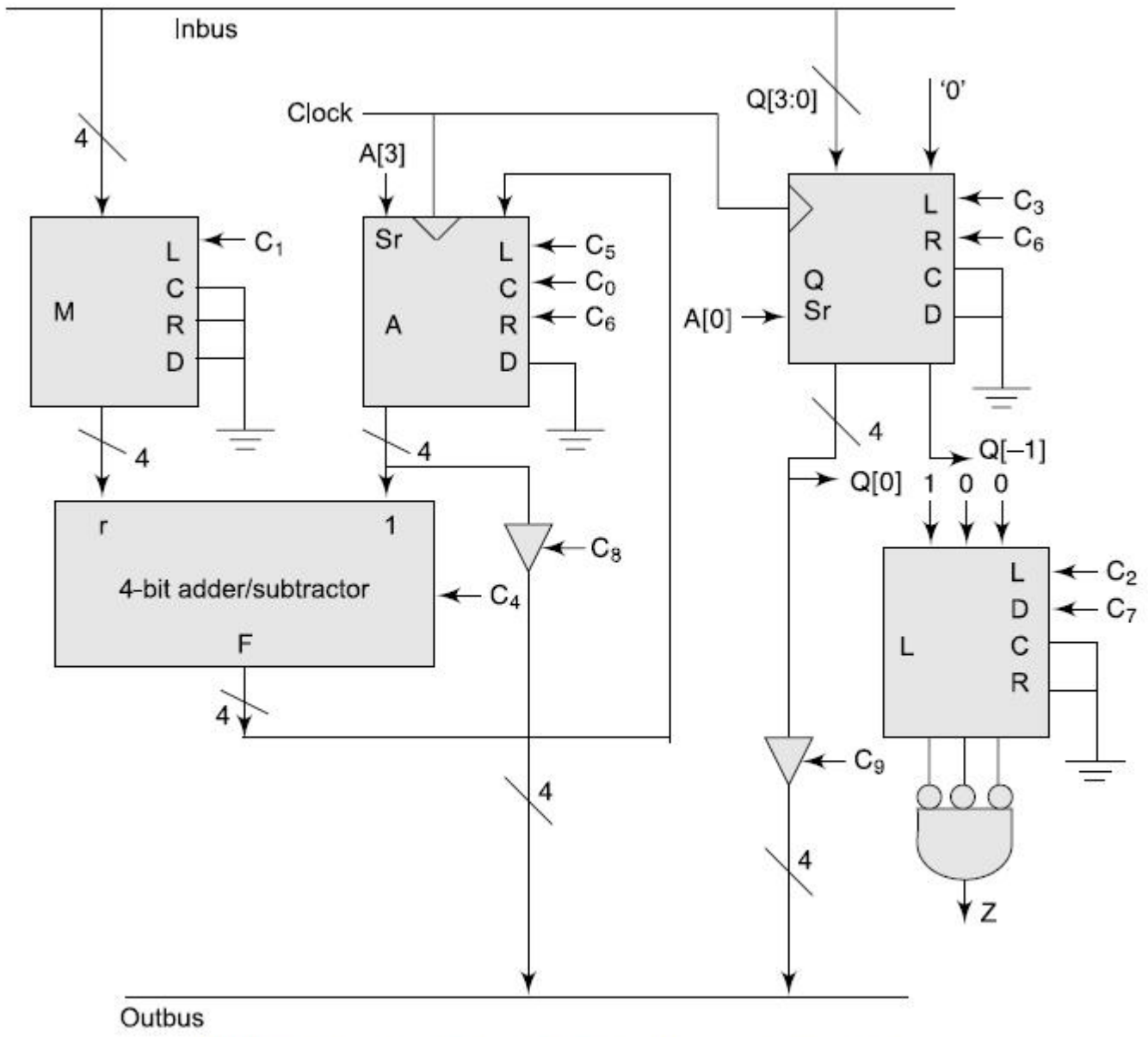
Step 5: There are 10 control signals required: C0, C1, C2, C3, C4, C5, C6, C7, C8, C9 and their tasks are provided next. Though, the signals' tasks are self-explanatory.

C_0: A ← 0
C_1: M ← Inbus
C_2: L ← 4

C_3: Q[3:0] ← Inbus
Q[–1] ← 0
C_4: F = l + r
C_4': F = l – r

C_5: A ← F
C_6: ASR (AQ)
C_7: L ← L – 1

C_8: Outbus = A
C_9: Outbus = Q[3:0]



Figrue 6.5 *4 × 4 Booth's multiplier processing section*

Step 6: The processing section intermediately generates three outputs Q[0], Q[–1] and Z. When the content of the L register becomes 0, then Z register is set to 1. These outputs are status outputs and are used as inputs to the controller to allow the controller to decide the next step of the algorithm.

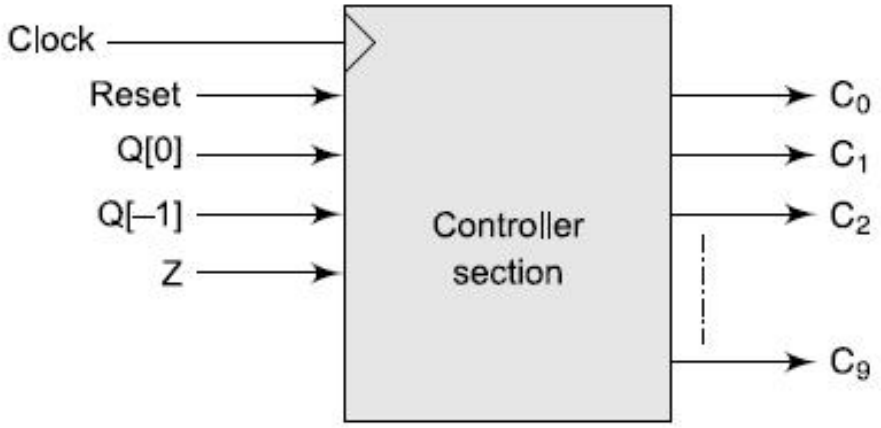
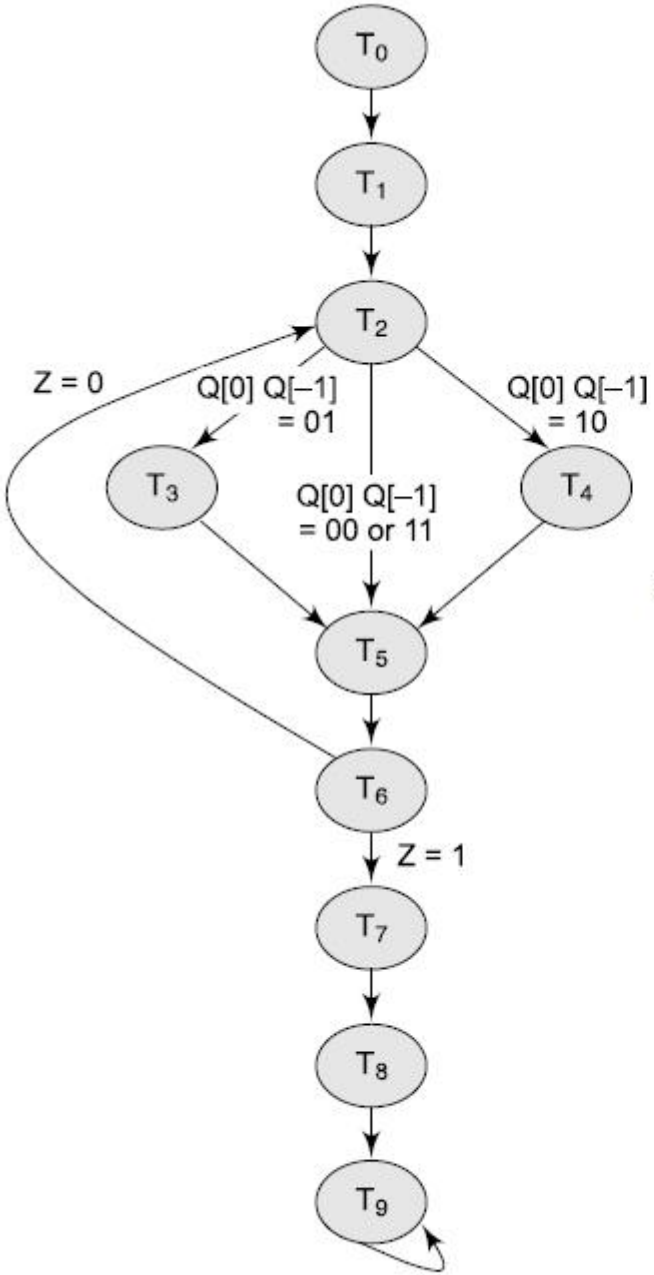


Figure 6.6 *Block diagram of the booth's multiplier controller*

Step 7 : The controller has 5 inputs and 10 control outputs. The clock input is used to synchronize the controller's activities. The Reset input is asynchronous input used to reset the controller so that a new computation can start. The state diagram for theBooth's multiplier controller is shown in the Fig.

There are 10 states in the controller state diagram. Ten non-overlapping timing signals (T_0 to T_9) must be generated for the controller to perform the Booth's algorithm. But, only one will be high for a clock pulse.



Control State	*Operation performed*	*Signals to be activated*
T_0	$A \leftarrow 0$, $M \leftarrow$ Inbus $L \leftarrow 4$	C_0, C_1, C_2
T_1	$Q[3:0] \leftarrow$ Inbus $Q[-1] \leftarrow 0$	C_3
T_2	None	None
T_3	$A \leftarrow A + M$	C_4, C_5
T_4	$A \leftarrow A - M$	C_5
T_5	ASR (AQ) $L \leftarrow L - 1$	C_6, C_7
T_6	None	None
T_7	Outbus = A	C_8
T_8	Outbus = Q[3:0]	C_9
T_9	None	None

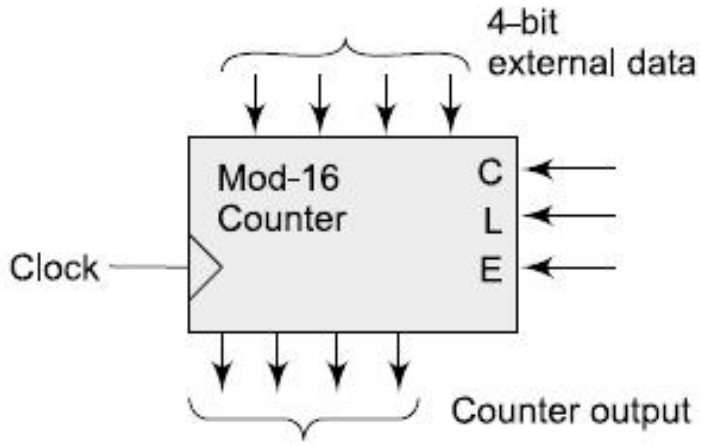
Figure 6.7 *Controller's state diagram and description*

Initially, the controller is in the state T_0. At this time the control signals C_0, C_1 and C_2 are generated at high state. Thus, the operations A $\leftarrow$ 0, M $\leftarrow$ Inbus and L $\leftarrow$ 4 are performed. The controller then moves to the state T_1 in the next clock cycle to perform the operation Q[3:0] $\leftarrow$ Inbus and Q[–1] $\leftarrow$ 0. The controller moves to the state T_9 only when a computation is completed and the controller stays in that state infinitely until a Reset input forces the controller to switch to the state T_0 and a new computation step starts.

The states are generated in the state diagram according to the following rules:

- If the two or more micro-operations are independent of each other and can be completed within one clock cycle, they are grouped into one state. For example, micro-operations A $\leftarrow$ 0, M $\leftarrow$ Inbus and L $\leftarrow$ 4 are independent to each other. That is why they are executed in one clock period. If these micro-operations cannot be performed within the selected T_0 clock period, then either clock period duration needs to be increased or the micro-operations have to be divided into a sequence of micro-operations.
- Generally a new state is introduced for conditional testing. For example, the conditional testing of the bit pair Q[0] Q[–1] introduces the new state T_2 in Fig. 6.7.

Step 8: Since minimum 10 clock cycles (periods) are needed for 10 states in the controller, a mod-16 counter and a 4-to-16 decoder are used to generate the clock periods and to select one of the control signals at the appropriate state respectively. The characteristic of the mod-16 counter is discussed in the Fig.



C	L	E	Clock	Action
1	x	x	x	Clear
0	1	x	⎍	Load ext. data
0	0	1	⎍	Count up
0	0	0	⎍	No operation

Function table of the counter

Figure 6.8 *Characteristics of the counter used in the controller design*

Step 9: The controller and its logic diagram are shown in the Fig

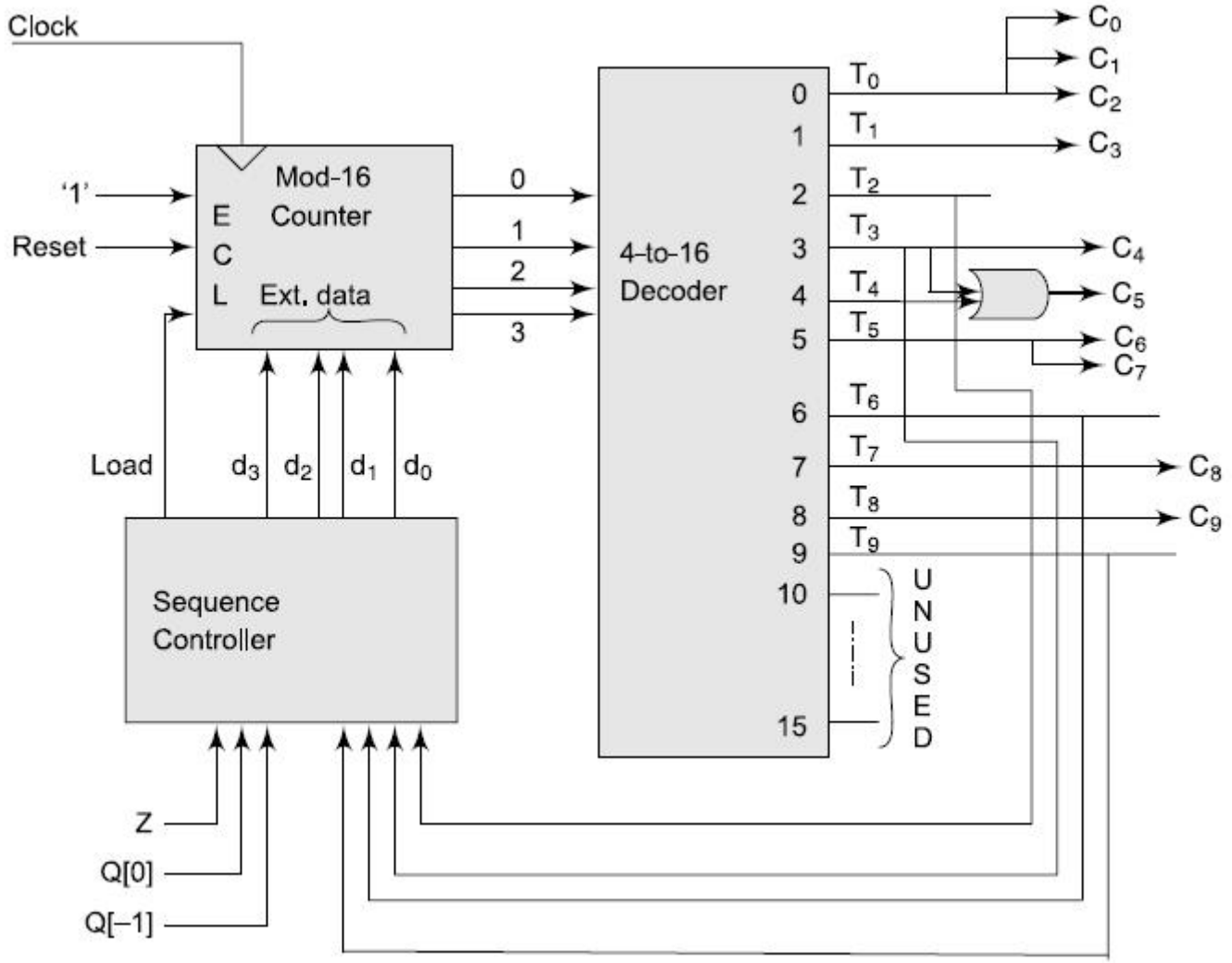
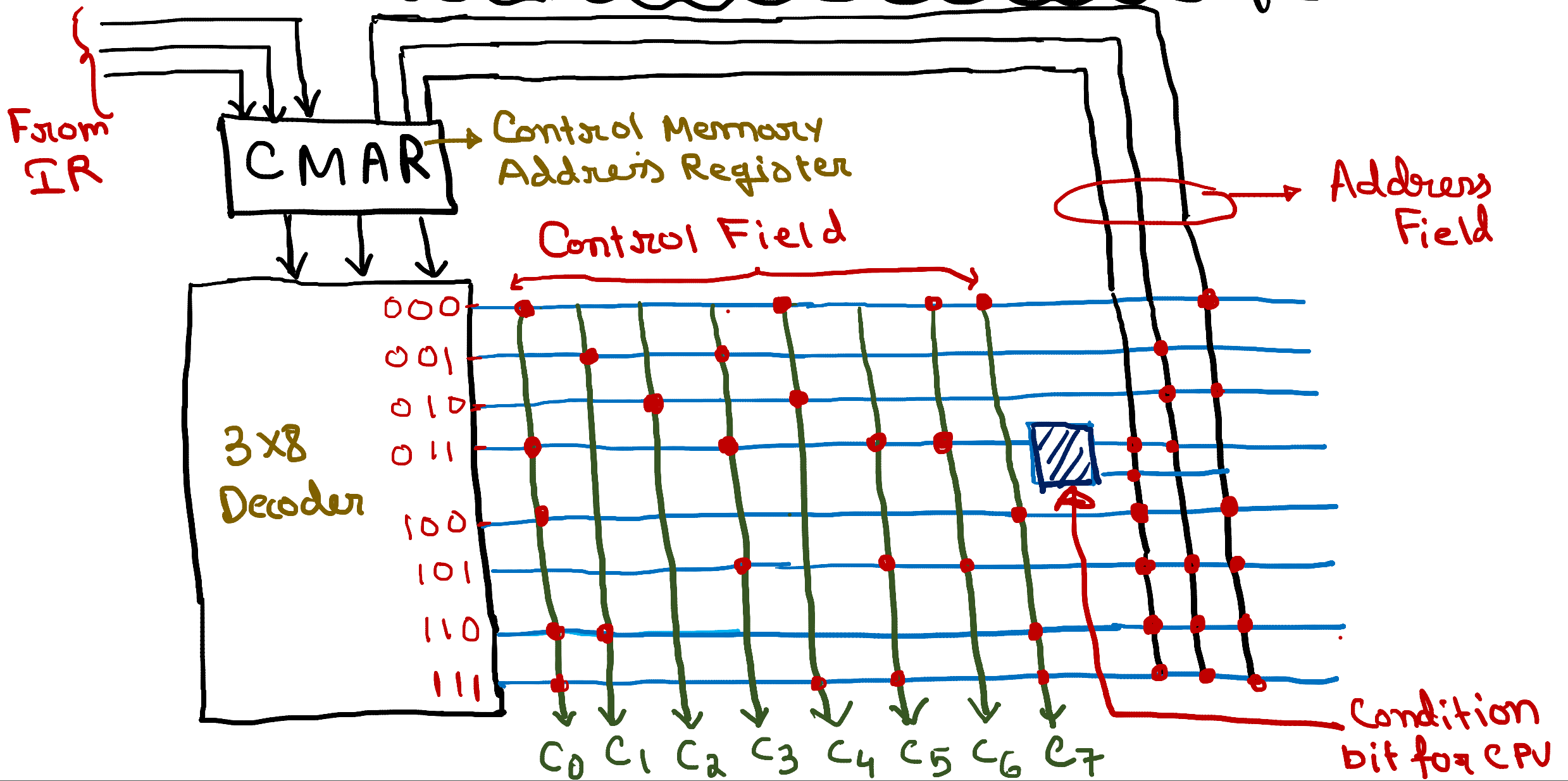


Figure 6.9 *Logic diagram of the Booth's multiplier controller*

Wilkes Control Circuit Design



Microprogrammed Control

Decode Line Activated	Control Signal Generated	Address of Next micro-instruction
000	C_0, C_4, C_6, C_7	000
001	C_1, C_3	010
010	C_2, C_4	011
011	C_0, C_3, C_5, C_6	?
Check Condition bit Either True: 011	C_0, C_3, C_5, C_6	If true then go to 110
Either True: 110	C_0, C_1, C_7	111
Either True: 111	C_0, C_4, C_5, C_7	Load Next IR
OR False. 011	C_0, C_3, C_5, C_6	If false then go (100)
OR False. 100	C_0, C_5	101
OR False. 101	C_3, C_5, C_6	111
OR False. 111	C_0, C_4, C_5, C_7	Load Next IR