## HARDWIRED APPROACH

- Control logic is a clocked sequential ckt.
- So conventional sequential ckt design procedure can be applied to build CU.
- Final circuit is obtained by physically connecting gates and flip flops.
- Cost of control logic increases with system complexity.

# 10 steps for hardwired control

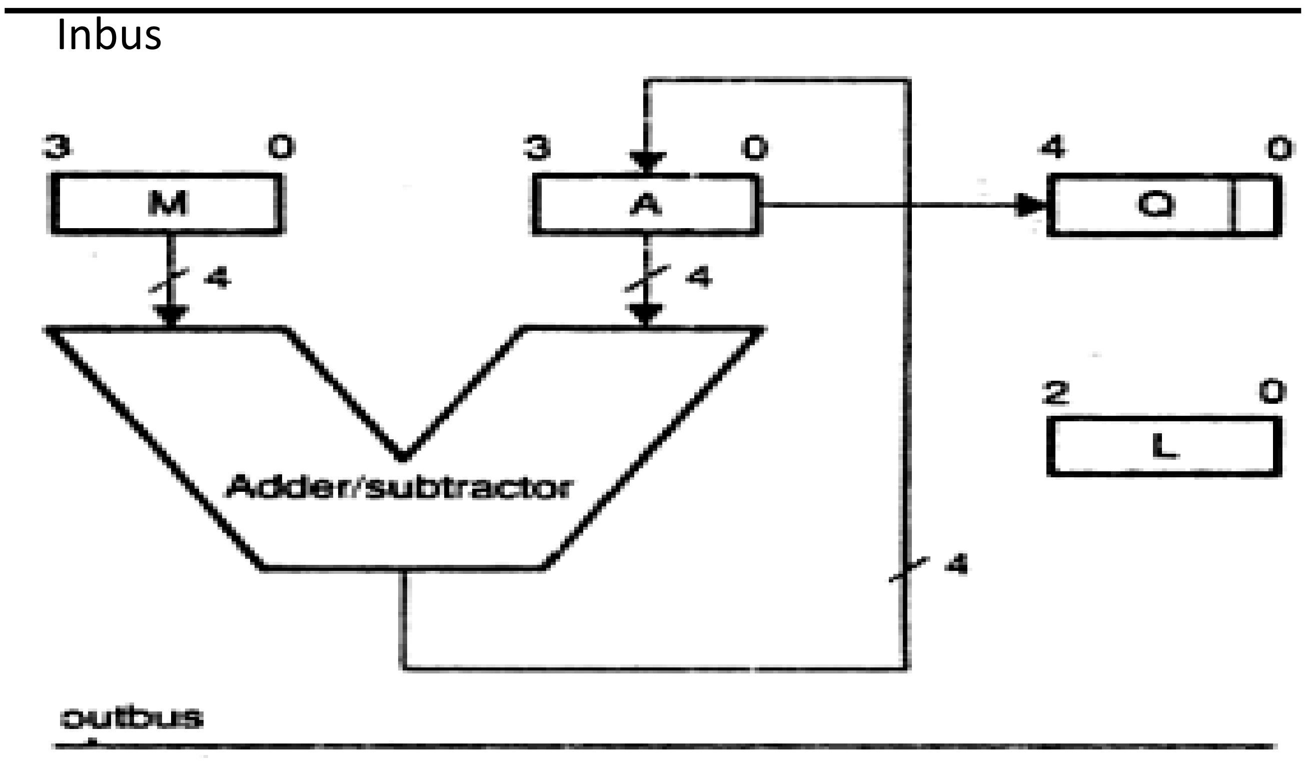
1) Define task to be performed.
2) Propose a trial processing section.
3) Provide a reg tx descr algo based on processing section outlined.
4) Validate the algo by using trial data.
5) Describe the basic char of the HW elements to be used in the processing section.
6) Complete the design of the processing section by establishing necessary control points.
7) Propose the block diagram of the controller.
8) Specify state diagram of controller.
9) Specify the char of the HW elements to be used in the controller.
10) Complete the controller design and draw a logic diagram of final circuit.

**Step 1:** Task definition.

Design a Booth's multiplier to multiply two 4-bit signed numbers.

**Step 2:** trial processing section.

| $q_1$ | $q_0$ |
|---|---|
| 0 | 0→none |
| 0 | 1→ add M |
| 1 | 0→sub M |
| 1 | 1→ None |



**Step 3: register transfer description of Booth’s multiplier procedure based on the processing section outlined in the previous step.**

Declare registers A[4], M[4], Q[5], L[3]

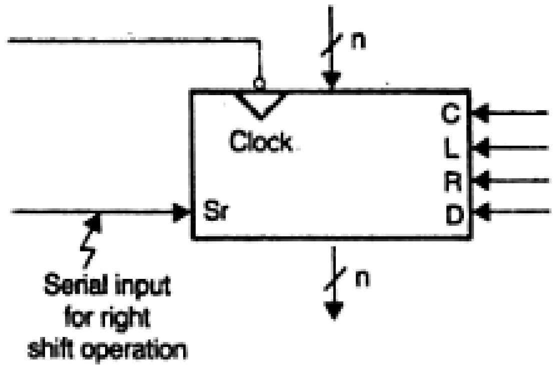
Declare buses Inbus[4], outbus[4]

```
Start:   A←0, M←inbus, L←4;                 clear A and transfer M
         Q[4:1]←inbus, Q[0]←0;              transfer Q
Loop:    if Q[1:0]= 01, then go to ADD
         if Q[1:0]=10, then  go to SUB;
         go to Rshift;
ADD:     A←A+M;
         goto Rshift;
SUB:     A←A-M;
Rshift:  ASR(AQ), L←L-1;
          if L>0, then go to loop
          outbus =A;
          outbus=Q[4:1];
          go to halt
```

# Step 4: Validate the algo by using trial data.

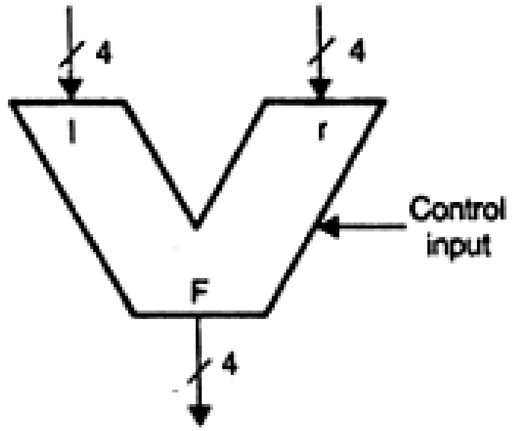
| A | Q | $Q_{-1}$ | M | | |
|---|---|---|---|---|---|
| 0000 | 0011 | 0 | 0111 | Initial Values | |
| 1001 | 0011 | 0 | 0111 | A A - M | First Cycle |
| 1100 | 1001 | 1 | 0111 | Shift | |
| 1110 | 0100 | 1 | 0111 | Shift | Second Cycle |
| 0101 | 0100 | 1 | 0111 | A A + M | Third Cycle |
| 0010 | 1010 | 0 | 0111 | Shift | |
| 0001 | 0101 | 0 | 0111 | Shift | Fourth Cycle |

**Step 5:** Processing section includes GPRs, 4-bit adder / subtractor, Tristate buffers



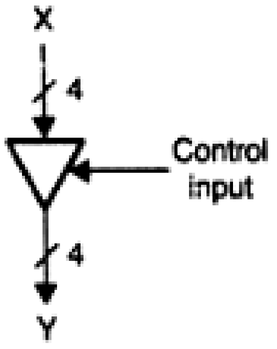
| C | L | R | D | Clock | Action |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | ↓ | Clear |
| 0 | 1 | 0 | 0 | ↓ | Load external data |
| 0 | 0 | 1 | 0 | ↓ | Right shift |
| 0 | 0 | 0 | 1 | ↓ | Decrement by one |
| 0 | 0 | 0 | 0 | ↓ | No change |

a. Storage Register



| Control input | F |
|---|---|
| 1 | l + r |
| 0 | l − r |

b. Adder subtractor



| Control input | Y |
|---|---|
| 1 | X |
| 0 | High Z |

c. Tri-state Buffer

**Figure 4.17** Characteristics of the Component Parts Used in the Processing Section of the Booth's Multiplier

**Step 6:** The complete design of processing section establishing control points.

| | |
|---|---|
| $C_0$: | $A \leftarrow 0$ |
| $C_1$: | $M \leftarrow$ Inbus |
| $C_2$: | $L \leftarrow 4$ |
| $C_3$: | $Q[4:1] \leftarrow$ Inbus |
| | $Q[0] \leftarrow 0$ |
| $C_4$: | $F = l + r$ |
| $C_4'$: | $F = l - r$ |
| $C_5$: | $A \leftarrow F$ |
| $C_6$: | ASR (A \$ Q) |
| $C_7$: | $L \leftarrow L - 1$ |
| $C_8$: | Outbus = A |
| $C_9$: | Outbus = Q[4:1] |

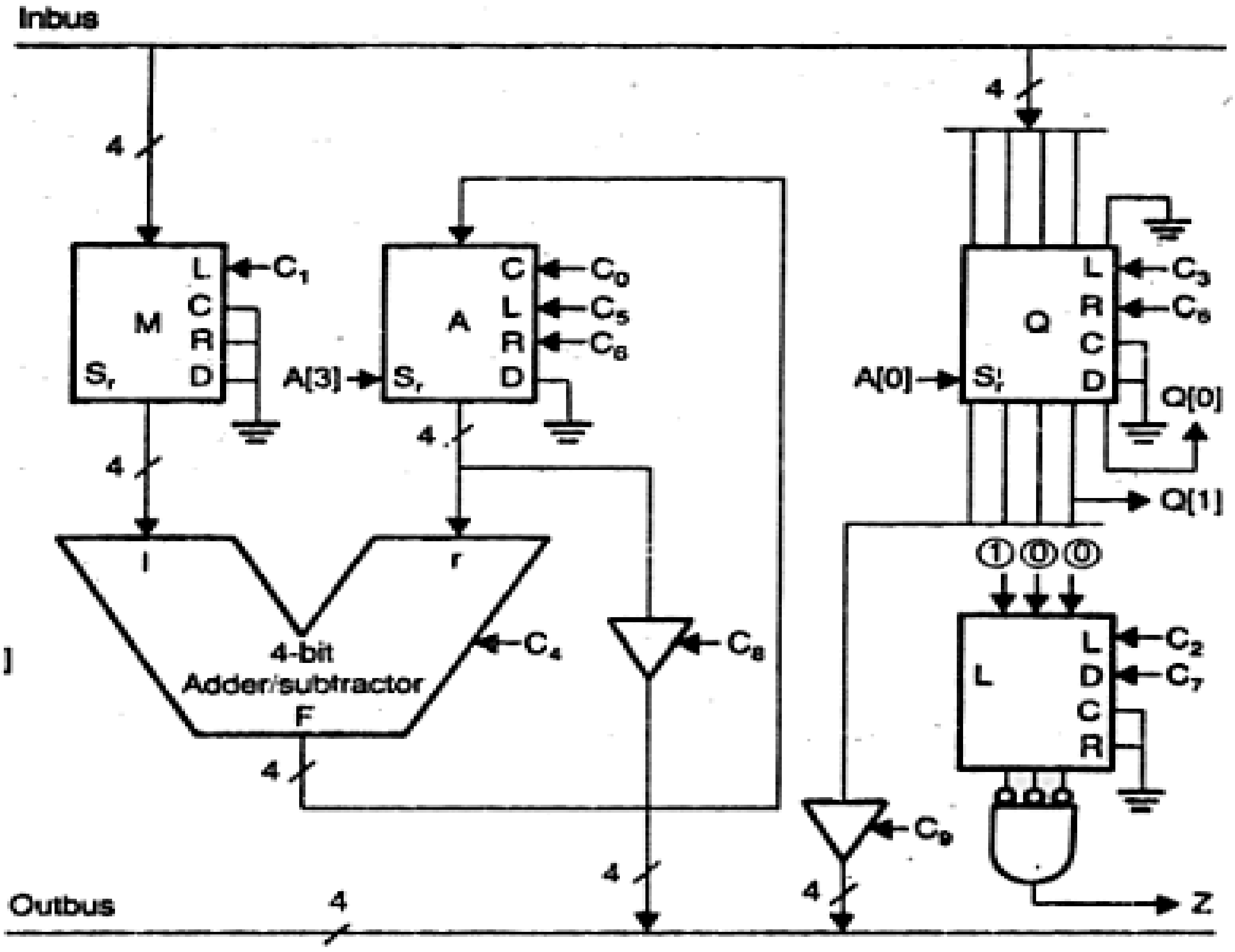


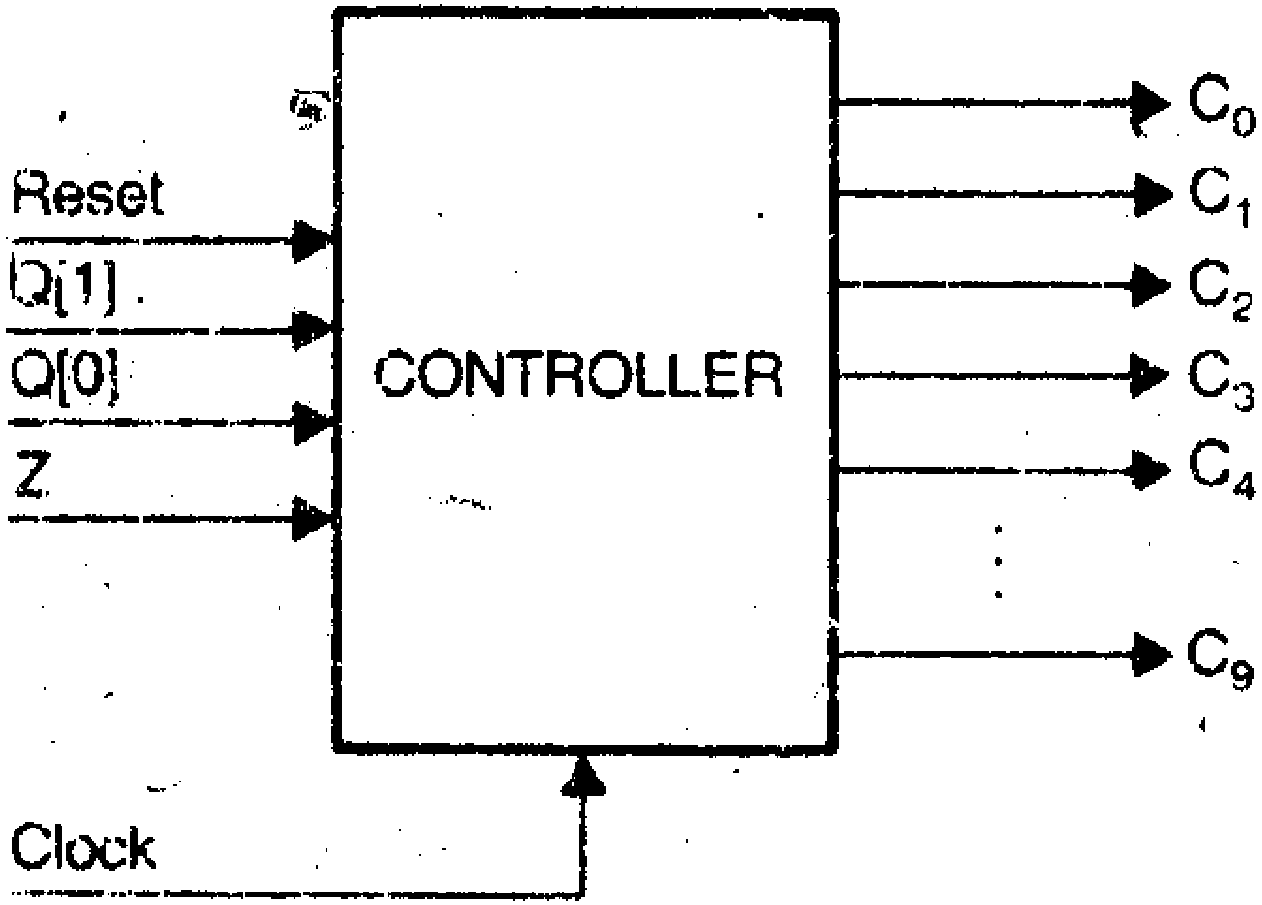
Figure 4.18 Processing Section of the Booth's Multiplier

## **Step 7:** Block diagram of controller:

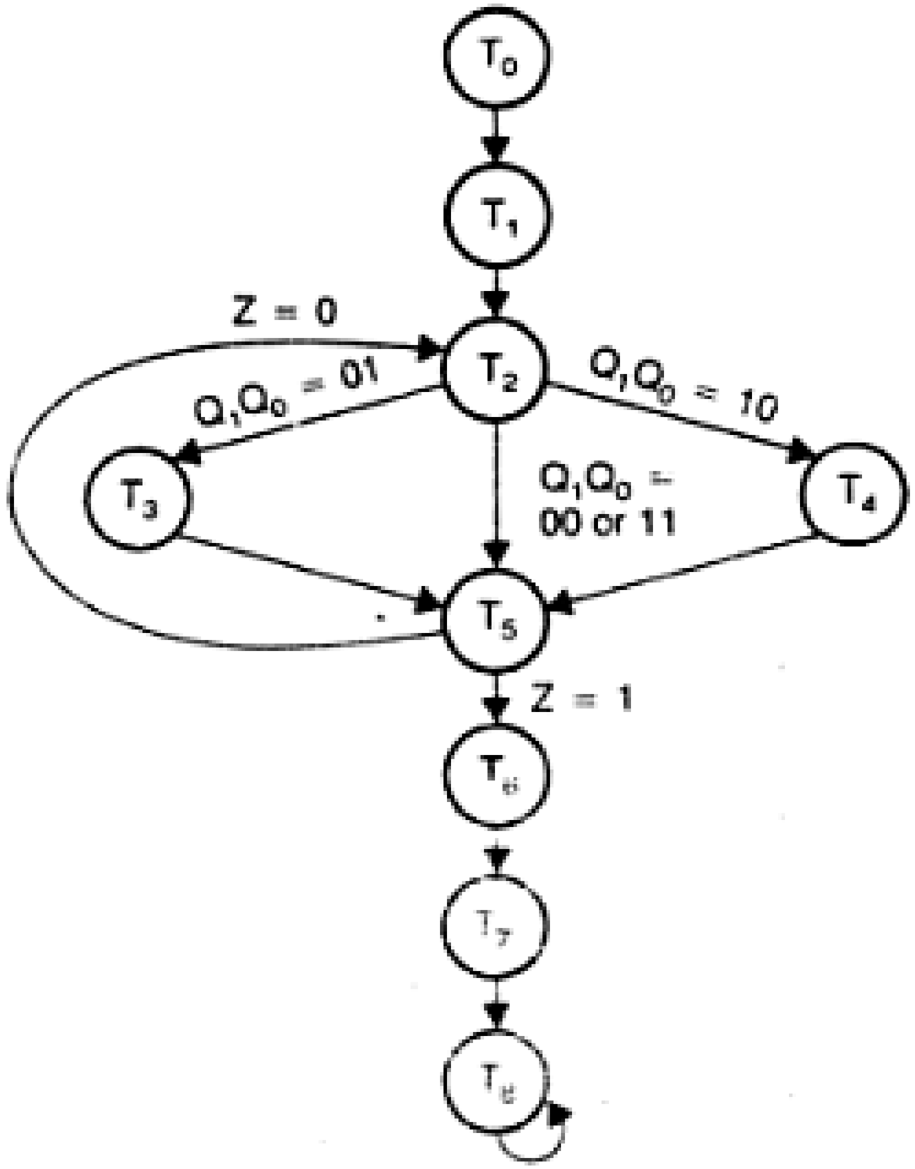
will have 5 i/ps and 10 o/ps.

RESET i/p is used to reset the controller so a new computation can begin.

CLK is used to synch the controller action for trailing edge of clock pulse.



## **Step 8:** The state diagram of Booth's multiplier controller



a. State Diagram

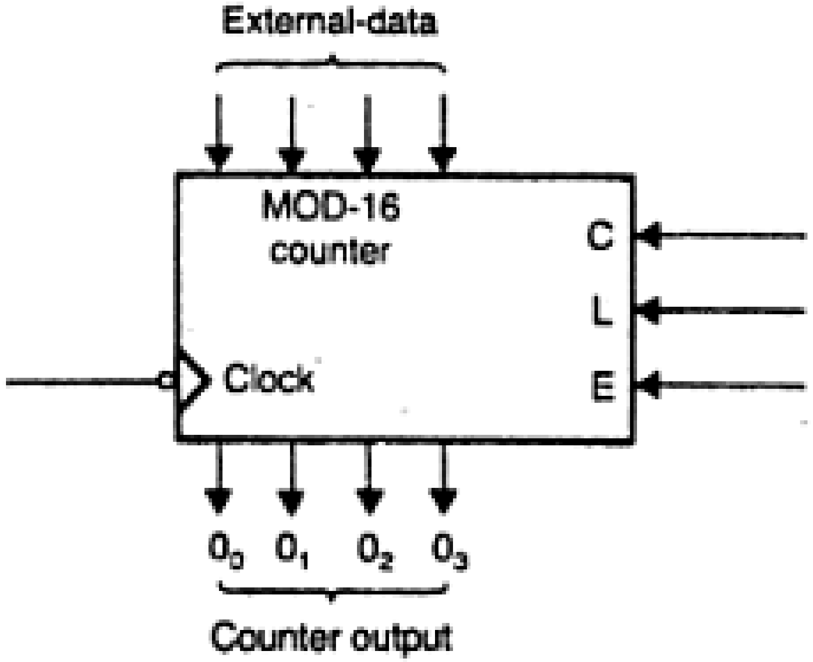
| CONTROL STATE | OPERATION PERFORMED | CONTROL SIGNALS TO BE ACTIVATED |
|---|---|---|
| $T_0$ | $A \leftarrow 0$, $L \leftarrow 4$, $M \leftarrow$ Inbus | $C_0$, $C_1$, $C_2$ |
| $T_1$ | $Q[4:1] \leftarrow$ Inbus, $Q[0] \leftarrow 0$ | $C_3$ |
| $T_2$ | None | None |
| $T_3$ | $A \leftarrow A + M$ | $C_4$, $C_5$ |
| $T_4$ | $A \leftarrow A - M$ | $C_5$ $(C_4 = 0)$ |
| $T_5$ | ASR (ASQ), $L \leftarrow L - 1$ | $C_6$, $C_7$ |
| $T_6$ | Outbus = A | $C_8$ |
| $T_7$ | Outbus = $Q[4:1]$ | $C_9$ |
| $T_8$ | None | None |

b. Controller Action

Figure 4.20 Controller Description

**Step 9:** The controller includes a mod -16 counter, a 4: 16 decoder, a sequence controller (SC).

**Step 9:** The controller includes a mod -16 counter, a 4: 16 decoder, a sequence controller (SC).



a. Block Diagram

| C | L | E | Clock | Action |
|---|---|---|---|---|
| 1 | X | X | X | Clear |
| 0 | 1 | X | ↓ | Load external data |
| 0 | 0 | 1 | ↓ | Count up |
| 0 | 0 | 0 | ↓ | No operation |

**Figure 4.22** Characteristics of the Counter Used in the Controller Design

SC HW, which sequences the controller according to state diagram.

Hence TT for SC must be derived from the controller's state diagram.

**Step 10:** The multiplier controller with its logic diagram

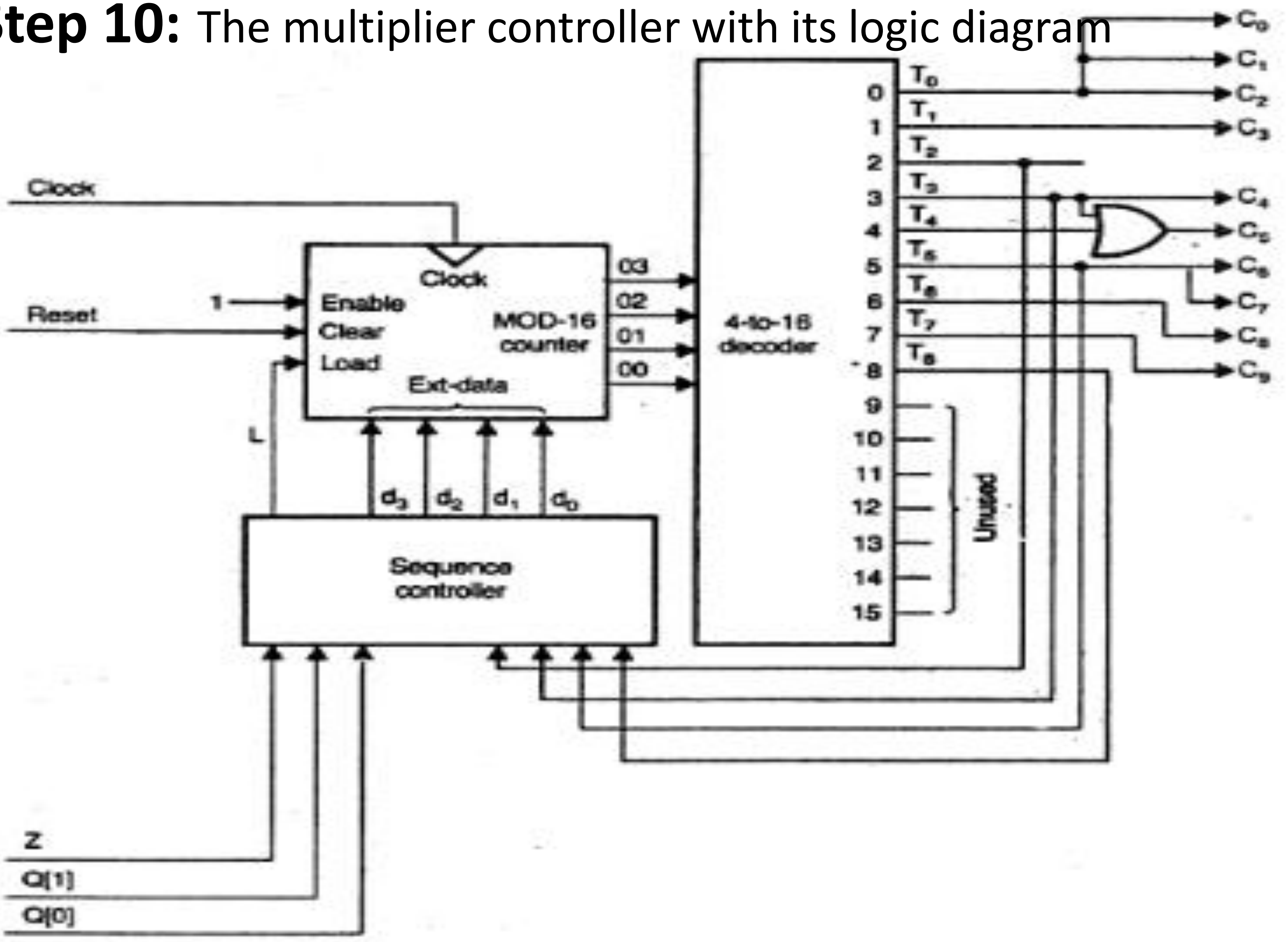


**Figure 4.23** Logic Diagram of the Booth's Multiplier Controller

|  |  |  |  |  |  |  |  | External-data |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Z | Q [1] | Q [0] | $T_2$ | $T_1$ | $T_3$ | $T_0$ | L | d3 | d2 | d1 | d0 |
| X | 0 | 0 | 1 | X | X | X | 1 | 0 | 1 | 0 | 1 |
| X | 1 | 1 | 1 | X | X | X | 1 | 0 | 1 | 0 | 1 |
| X | 1 | 0 | 1 | X | X | X | 1 | 0 | 1 | 0 | 0 |
| X | X | X | X | 1 | X | X | 1 | 0 | 1 | 0 | 1 |
| 0 | X | X | X | X | 1 | X | 1 | 0 | 0 | 1 | 0 |
| X | X | X | X | X | X | 1 | 1 | 1 | 0 | 0 | 0 |

## Implementing SC using PLA:

$P_0 = Q[1]' Q[0]' T_2$

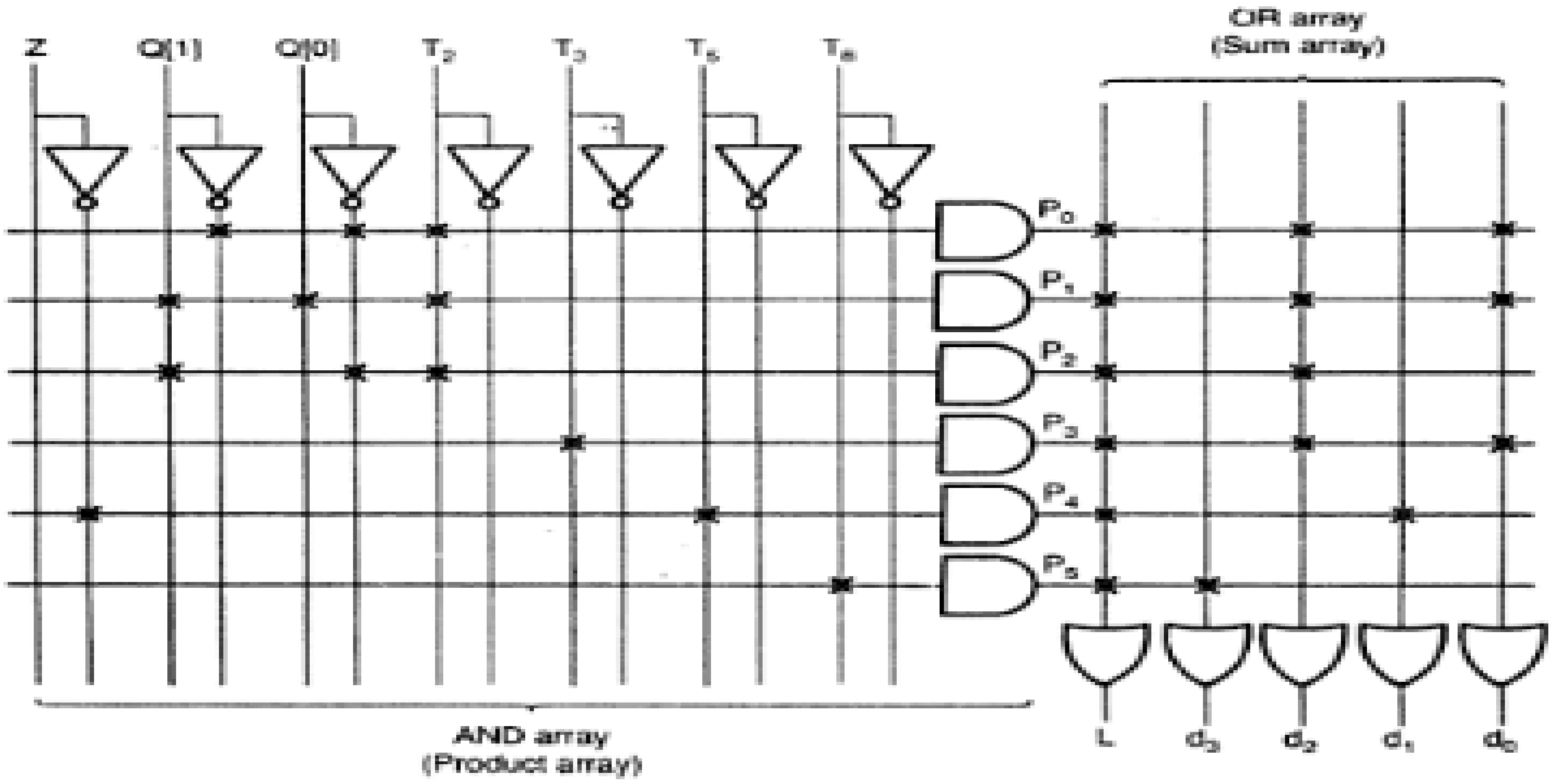
$P_1 = Q[1] Q[0] T_2$

$P_2 = Q[1] Q[0]' T_2$

$P_3 = T_3$

$P_4 = Z' T_5$

$P_5 = T_8$



b. PLA Implementation

**Figure 4.24** Sequence Controller Design

The PLA o/ps are summerized as

L = P0 + P1 + P2 + P3 + P4 + P5

d3 = P5

d2 = P0 + P1 + P2 + P3

d1 = P4

d0 =  P0 + P1 + P3

The controller design is completed by relating the control unit (T0-T8) with control i/ps C0-C9 as below:

C0 = C1 = C2 = T0

C3 = T1

C4 = T3

C5  = T3 + T4

C6 = C7 = T5

C8 =T6

C9 = T7

# CU design using a PLA and D F/Fs

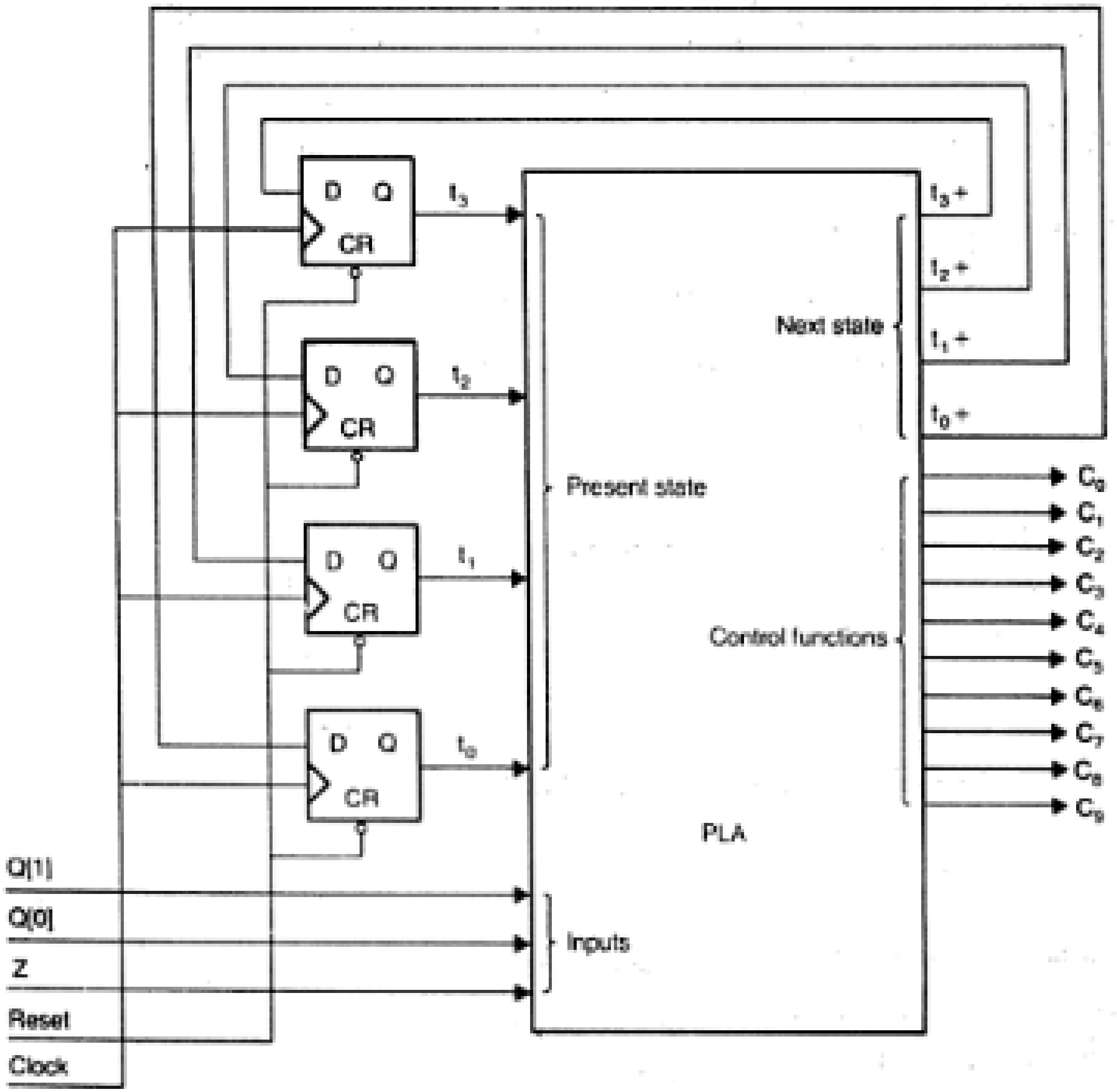


**Figure 4.25** Organization of the PLA Control Unit for the Booth's Multiplier

| Present state in | PLA Input | | | | | | | PLA Output | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Present state in binary | | | | Inputs | | | Next state (in binary) | | | | Control functions | | | | | | | | | |
| | $t_3$ | $t_2$ | $t_1$ | $t_0$ | Q [1] | Q [0] | Z | $t_3^*$ | $t_2^*$ | $t_1^*$ | $t_0^*$ | $C_0$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ | $C_9$ |
| $T_0$ | 0 | 0 | 0 | 0 | X | X | X | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $T_1$ | 0 | 0 | 0 | 1 | X | X | X | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $T_2$ | 0 | 0 | 1 | 0 | 0 | 1 | X | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $T_2$ | 0 | 0 | 1 | 0 | 0 | 0 | X | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $T_2$ | 0 | 0 | 1 | 0 | 1 | 1 | X | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $T_2$ | 0 | 0 | 1 | 0 | 1 | 0 | X | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $T_3$ | 0 | 0 | 1 | 1 | X | X | X | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| $T_4$ | 0 | 1 | 0 | 0 | X | X | X | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| $T_5$ | 0 | 1 | 0 | 1 | X | X | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| $T_5$ | 0 | 1 | 0 | 1 | X | X | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| $T_6$ | 0 | 1 | 1 | 0 | X | X | X | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $T_7$ | 0 | 1 | 1 | 1 | X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $T_8$ | 1 | 0 | 0 | 0 | X | X | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Controller's state table

**Figure 4.38** PLA Table

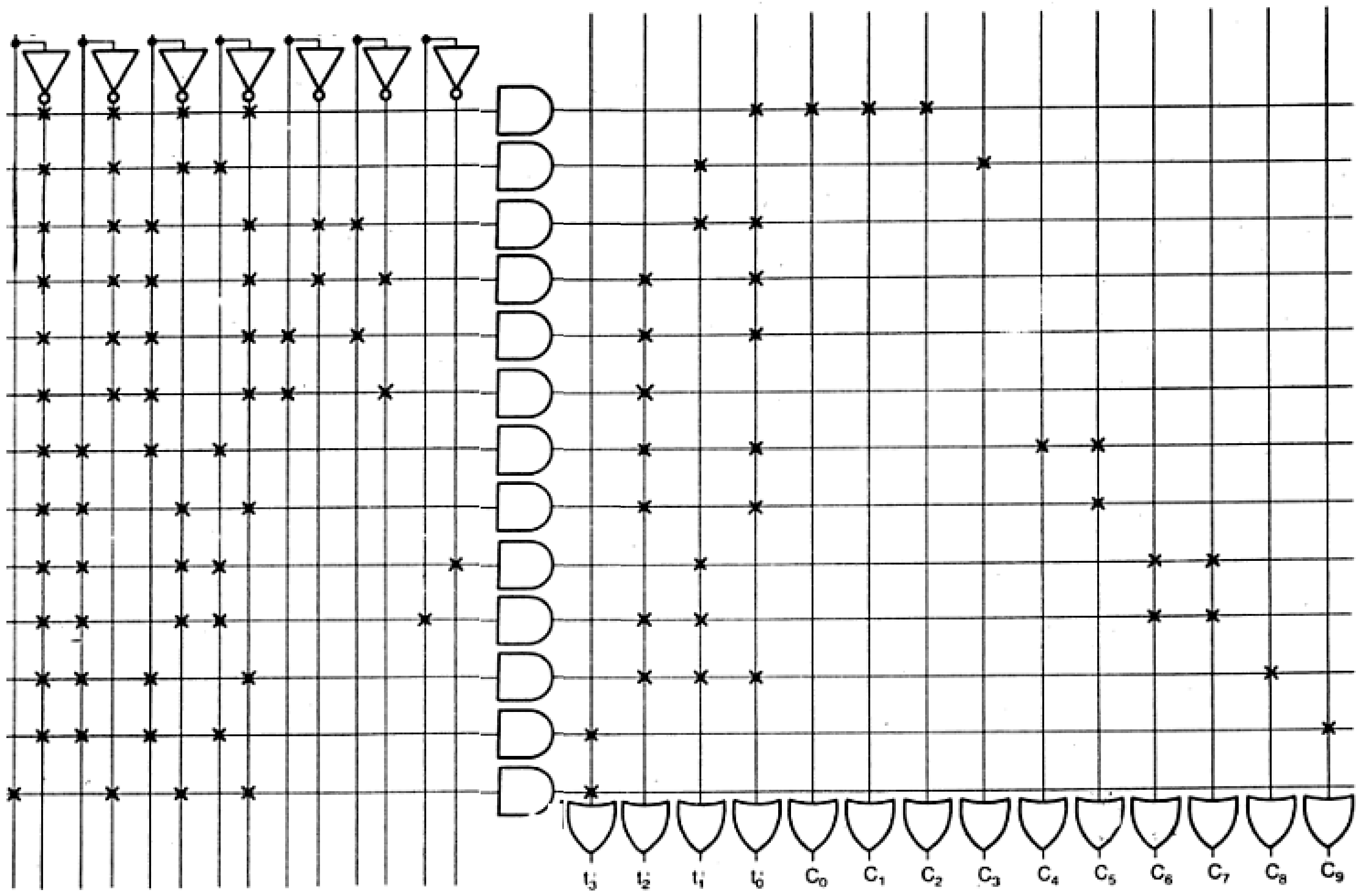


**Figure 4.27** PLA Contents

# Microprogrammed control unit:

Control word: all control signals that can be simultaneously activated are grouped to form the CW.

Micro operation: $A \leftarrow 0$,  outbus=A,  etc..
Each CW contains signals to activate one or more micro operations.

Control memory: -Control words are held in separate memory called control memory (CM).
-Control words are fetched from CM and individual control fields are routed to various functional units to achieve desired task.

All microinstructions have 2 important fields:

- -Control field
- -Next address field

Purpose of control field is to indicate which control lines are to be activated.

Purpose of Next address field is to specify the address of the next microinstruction to be executed.

# Wilke’s design of microprogrammed control unit

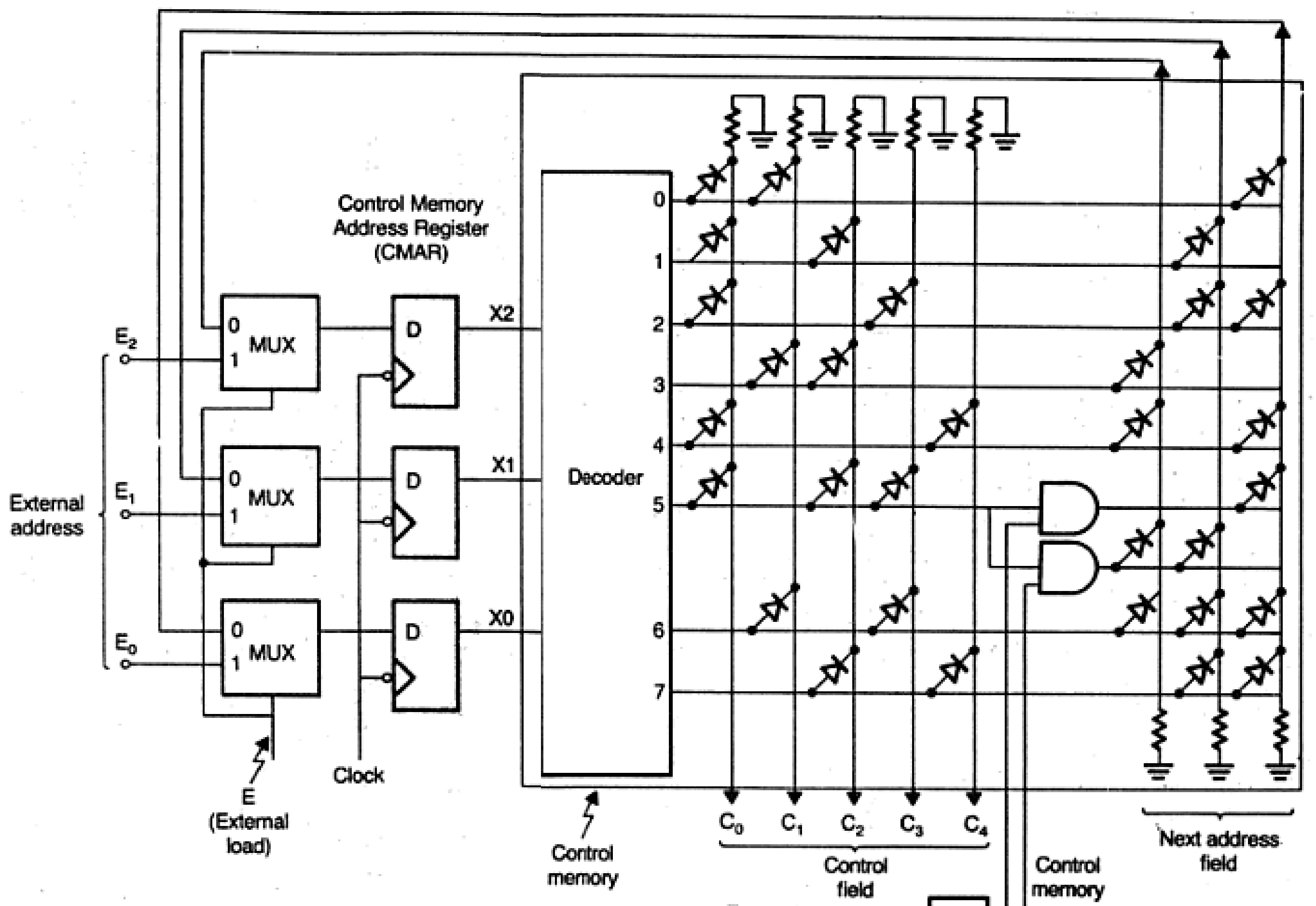


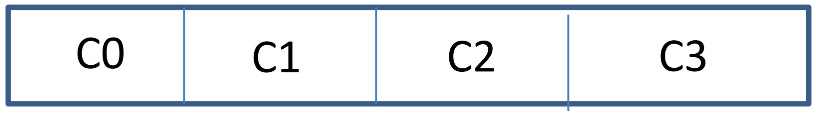
**Figure 4.28** Wilke's Design

The length of the µinstruction is dependent on factors:

- How many µoperations will have to be activated simultaneously.
- The method by which the address of next µinstruction is specified.

All µoperations executed in parallel can be specified in a single µinstruction.
This allows short µprograms to be written.

If the degree of parallelism increases, then the length of μinstruction increases.
IIIrly short μinstructions have limited capability in expressing parallelism. The overall length of μprogram will increase.
Various ways of organizing the control information:
Consider A, B, C, D each communicates with the outbus
C0: outbus=A;
C1: outbus=B;
C2: outbus=C;
C3: outbus=D;

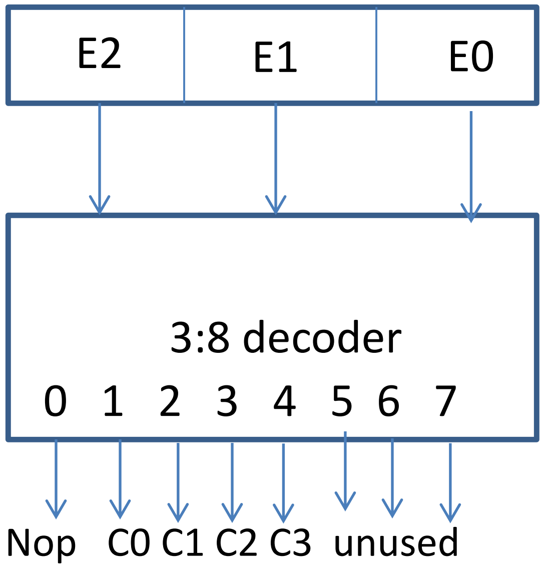


| C0 | C1 | C2 | C3 | |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | Outbus=A |
| 0 | 1 | 0 | 0 | Outbus=B |
| 0 | 0 | 1 | 0 | Outbus=C |
| 0 | 0 | 0 | 1 | Outbus=D |
| 0 | 0 | 0 | 0 | NO operation |

Here there is no need of decoding the control field. This method is known as unencoded format.

The above valid 5 binary patterns also can be represented as

| E2 | E1 | E0 | |
|---|---|---|---|
| 0 | 0 | 0 | No operation |
| 0 | 0 | 1 | Outbus=A |
| 0 | 1 | 0 | Outbus=B |
| 0 | 1 | 1 | Outbus=C |
| 1 | 0 | 0 | Outbus=D |



HW: How a 15 control lines can be specified in unencoded and encoded format? What variations you can have?

What is Horizontal and vertical µinstr? Features?

## Hardwired approach v/s microprogrammed CU:

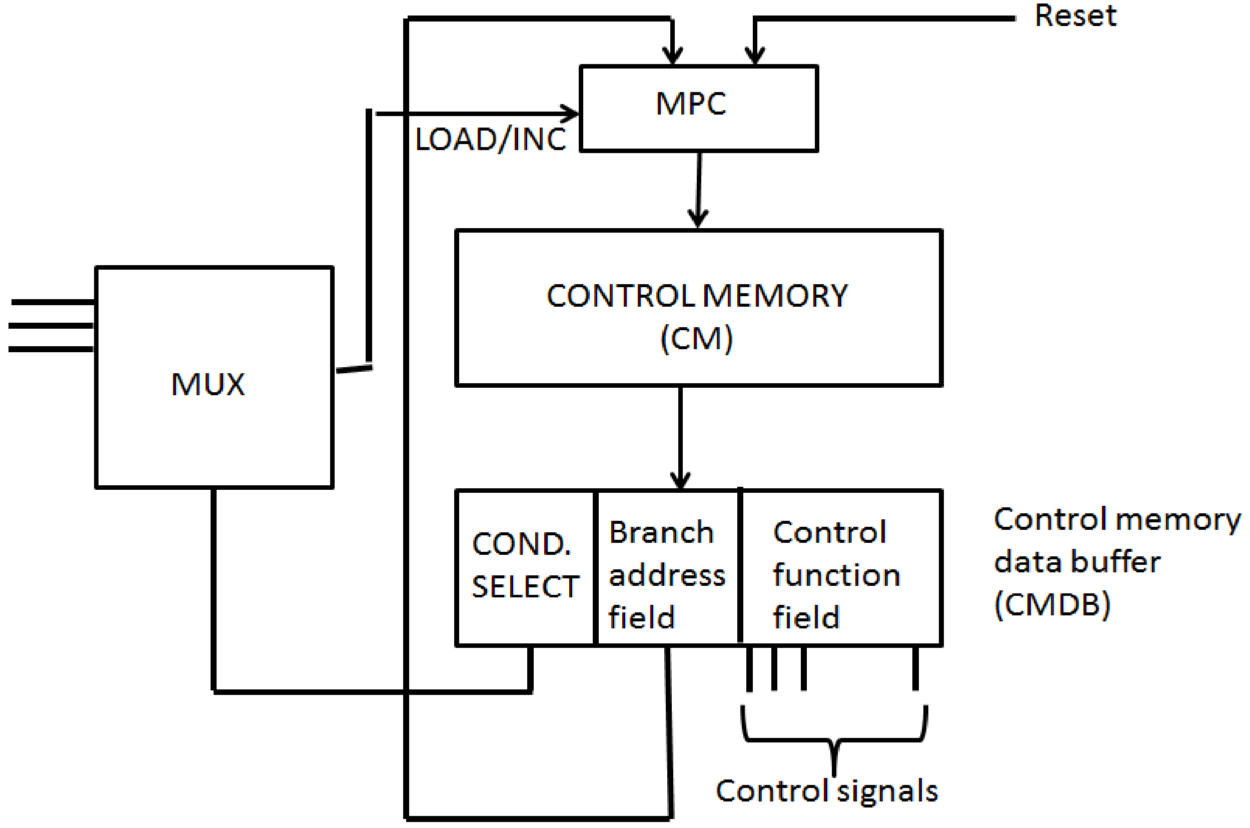
1. Microprogrammed approach is more expensive.
2. Control memory may reduce the overall speed of the machine, since microinsructions retrieval process takes significant amount of time.
3. Microprogramming provides a well structured control organization.
4. With Microprogramming, many additions and changes are made by simply changing the microprogram in Control memory, where as a small change in hardwired approach may lead to redesign the entire system.
5. Microprogrammed approach offers greater flexibility than hardwired since it provides an easy means for altering the contents of CM.

## Architecture of modern microprogrammed control unit:

Next address field from the μinstruction can be eliminated. This pointer is referred as microprogram counter (MPC).

MPC is functionally identical to PC.

It points to the μinstruction to be executed next and incremented after each μinstruction fetch.

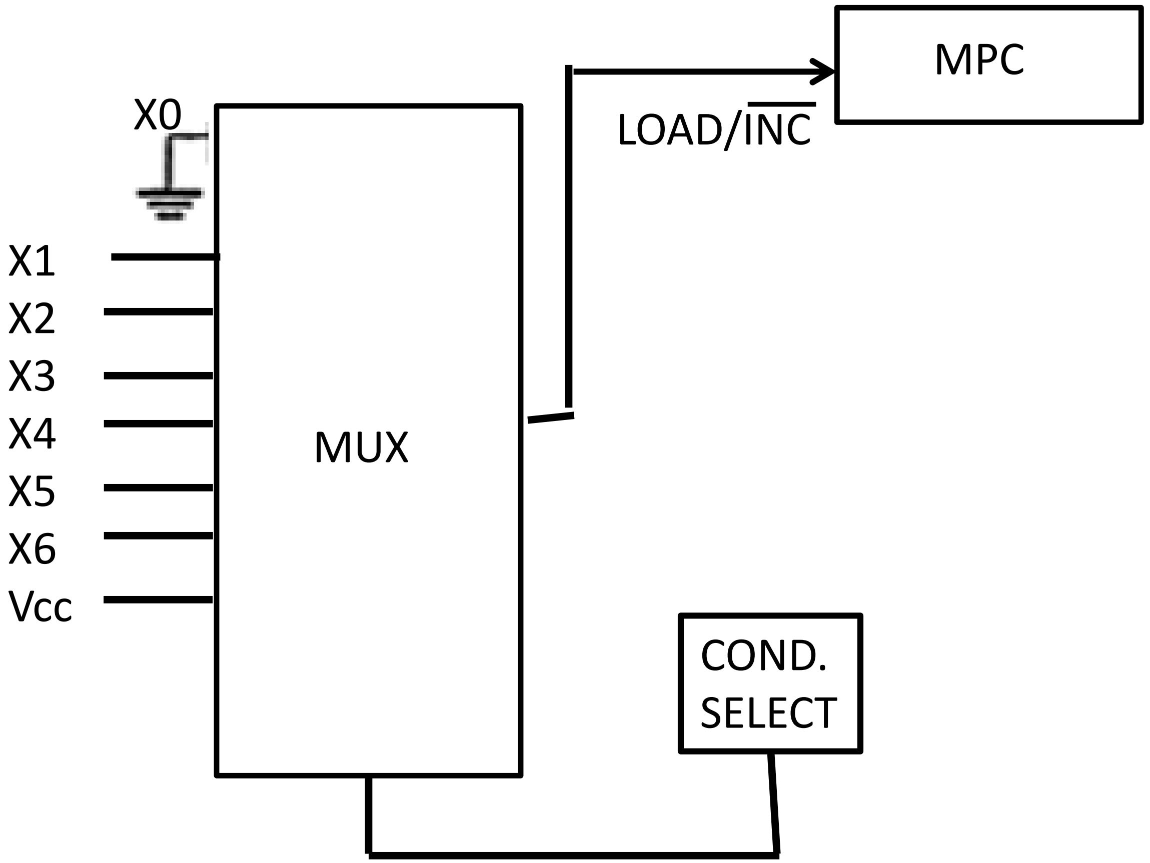

Reset
MPC
LOAD/INC
CONTROL MEMORY (CM)
MUX
COND. SELECT
Branch address field
Control function field
Control memory data buffer (CMDB)
Control signals

Suppose 6 external conditions X1, X2, X3, ..X6 are to be tested. Cond select field and MUX can be organized as:

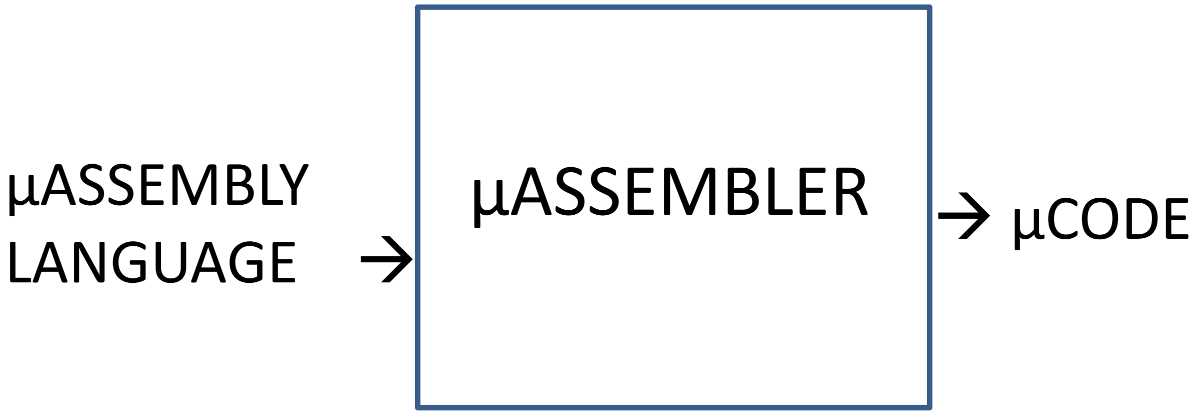
If cond sel 000 then MUX o/p 0. MPC incremented. No branch.

If cond sel 111 MUX o/p 1. Unconditional branching.

If cond sel 001 MUX o/p is same as value of X1. Now MPC will be loaded with branch addr when X1=1 else it is incremented.



Writing µprogram is IIIr to writing ALP.
µprogrammer must have more thorough knowledge about system archi
To speedup the development of µcode, →µassembly language.



These µcodes are held in CM.

# Design of µprogrammed CU for 4 X 4 Booth's Multiplier:

STEP1: Write µprogram in a symbolic form.

| Control Mem Addr | | Control Word |
|---|---|---|
| 0 | START: | A←0, M←Inbus, L←4; |
| 1 | | Q[4:1]←Inbus, Q[0]←0; |
| 2 | LOOP: | If Q[1:0]=01 then goto ADD; |
| 3 | | If Q[1:0]=10 then goto SUB; |
| 4 | | Goto RSHIFT; |
| 5 | ADD: | A← A+M; |
| 6 | | Goto RSHIFT; |
| 7 | SUB: | A←A-M; |
| 8 | RSHIFT: | ASR(A$Q), L←L-1; |
| 9 | | If Z=0 then goto LOOP |
| 10 | | outbus=A; |
| 11 | | outbus=Q[4:1]; |
| 12 | HALT: | Goto HALT |

CM holds 13 words, requiring a 4-bit branch address field.

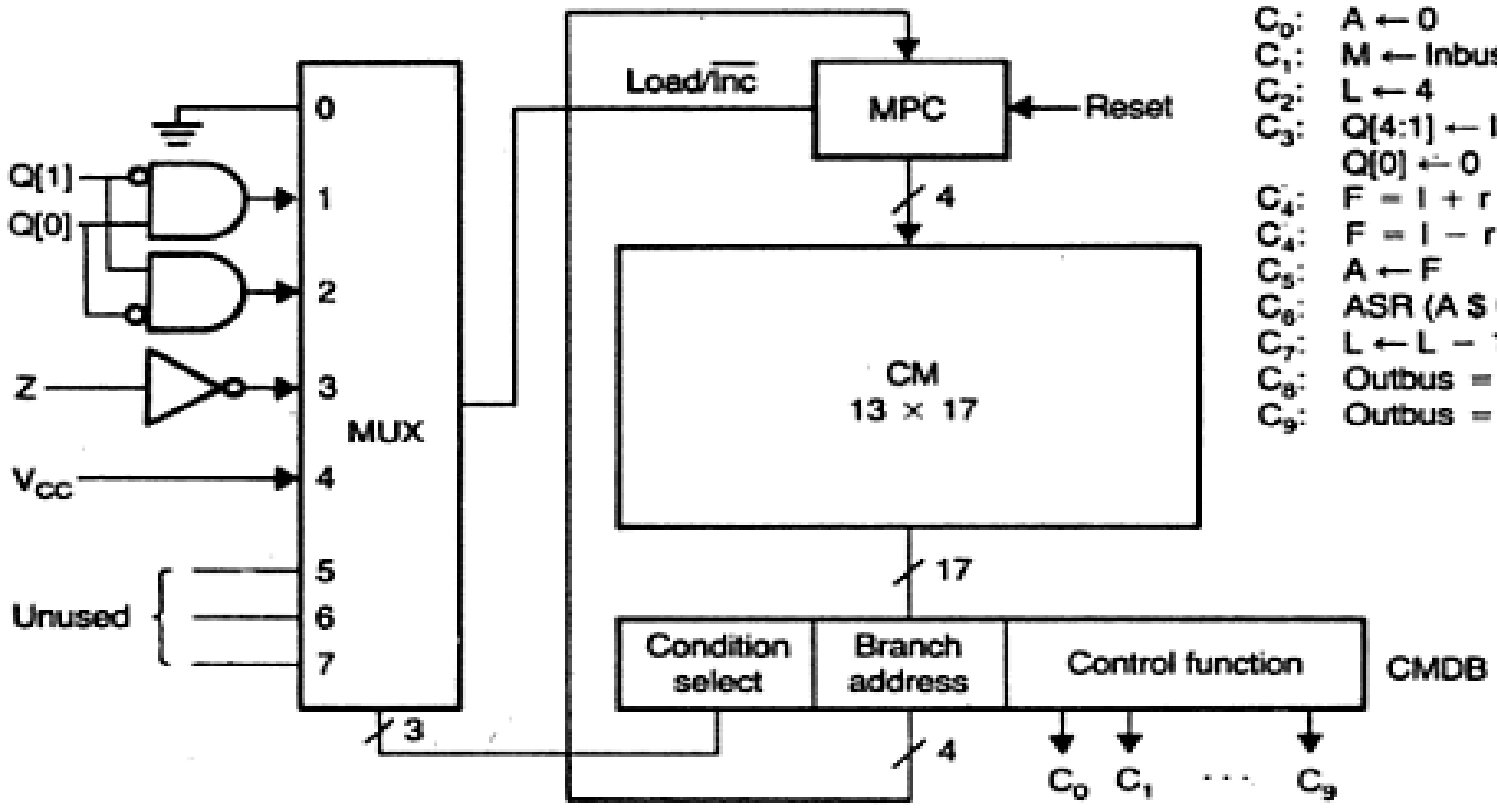
STEP2: Q[1]Q[0]=01
Q[1]Q[0]=10 and Z=0 are checked. These cond are applied as i/ps to cond sel MUX.
MUX must have atleast 5 data i/ps and 8:1 data selector.
3-bit cond sel field is used to encode 5 diff cond:

| **Cond Select Field** | **Action Taken** |
|---|---|
| 0 0 0 | No branching |
| 0 0 1 | Branch if Q[1]Q[0]=01 |
| 0 1 0 | Branch if Q[1]Q[0]=10 |
| 0 1 1 | Branch if Z=0 |
| 1 0 0 | Unconditional branch |

Size of CW is

| | | | | | |
|---|---|---|---|---|---|
| = | Size of Cond sel field | + | Size of branch field | + | No of functions |
| = | 3 | + | 4 | + | 10 |
| = | 17 bits | | | | |

Size of CMDB is 17 bits and CM is 13 x 17 =221bits



| | |
|---|---|
| $C_0$: | $A \leftarrow 0$ |
| $C_1$: | $M \leftarrow$ Inbus |
| $C_2$: | $L \leftarrow 4$ |
| $C_3$: | $Q[4:1] \leftarrow$ Inbus |
| | $Q[0] \leftarrow 0$ |
| $C_4'$: | $F = l + r$ |
| $C_4''$: | $F = l - r$ |
| $C_5$: | $A \leftarrow F$ |
| $C_6$: | ASR (A $\$$ Q) |
| $C_7$: | $L \leftarrow L - 1$ |
| $C_8$: | Outbus = A |
| $C_9$: | Outbus = Q[4:1] |

| ROM ADDRESS | | CONTROL WORD | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| In Dec | In Binary | COND SELECT | BRANCH ADDRESS | CONTROL FUNCTION | | | | | | | | | | |
| | | | | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 |
| 0 | 0 0 0 0 | 0 0 0 | 0 0 0 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 0 0 1 | 0 0 0 | 0 0 0 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 0 1 0 | 0 0 1 | 0 1 0 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 0 1 1 | 0 1 0 | 0 1 1 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 1 0 0 | 1 0 0 | 1 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 1 0 1 | 0 0 0 | 0 0 0 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 1 1 0 | 1 0 0 | 1 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 1 1 1 | 0 0 0 | 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 0 0 0 | 0 0 0 | 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 9 | 1 0 0 1 | 0 1 1 | 0 0 1 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1 0 1 0 | 0 0 0 | 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 11 | 1 0 1 1 | 0 0 0 | 0 0 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 12 | 1 1 0 0 | 1 0 0 | 1 1 0 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Binary listing of µprogram for 4 X 4 Booth's Multipier