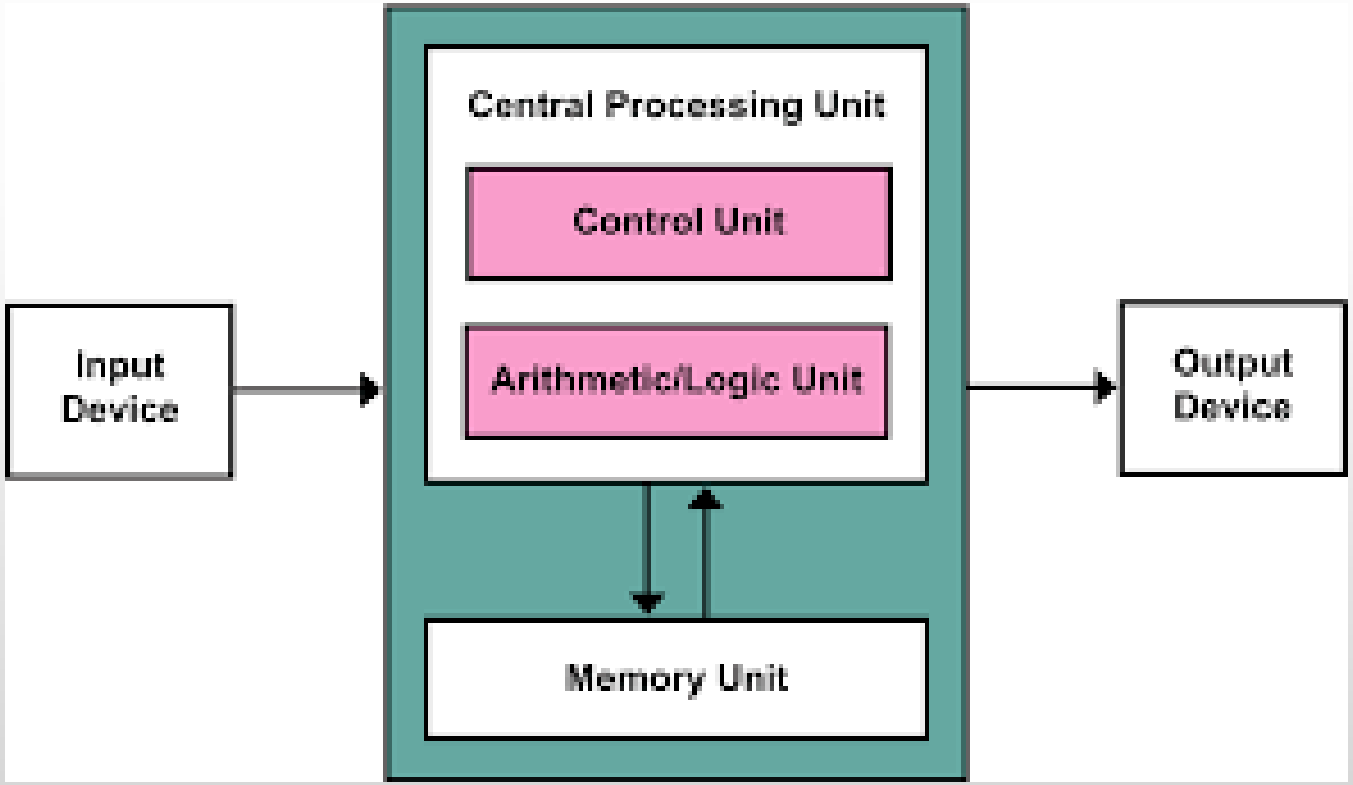
COMPUTER ORGANIZATION

Von Neuman Architecture

- Uses stored programmed concept
- Basic 5 functional units: Input unit, Output unit, Control unit, ALU and memory unit



Execution Unit

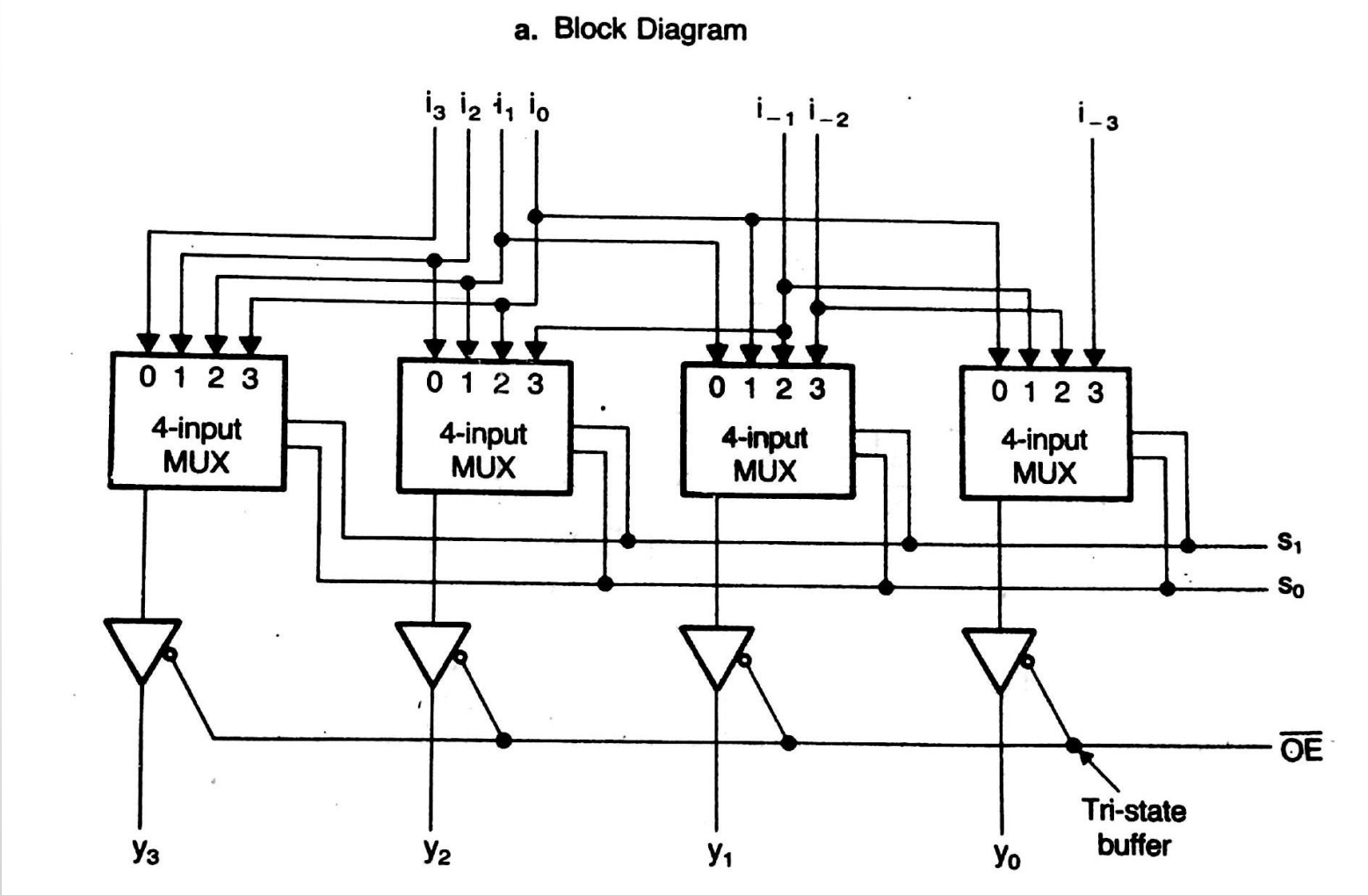
- Combinational shifter design
- Carry save adder (CSA)
- ALU
- Multiplication and Division algorithms

Combinational Shifter

Design a 4 x 4 combinational shifter to shift the 4-bit data according to the function table given below.

	Shift Count		Output				
$\overline{OE}$	s_1	s_0	y_3	y_2	y_1	y_0	Comment
1	X	X	Z	Z	Z	Z	Output lines float
0	0	0	i_3	i_2	i_1	i_0	Pass (no shift)
0	0	1	i_2	i_1	i_0	i_{-1}	Left shift once
0	1	0	i_1	i_0	i_{-1}	i_{-2}	Left shift twice

Combinational Shifter



Carry Save Adder (CSA)

- Applicable/efficient if the number of operands are more than 3
- Consider the example of addition of 4 operands: 34 +28+92+86

- 34 +
- 28
- 92
- 86

_____

2 0 + ←-------- Sum vector (no carry propagation from unit s place to tenths place)

22 ←---------- Carry vector

_____

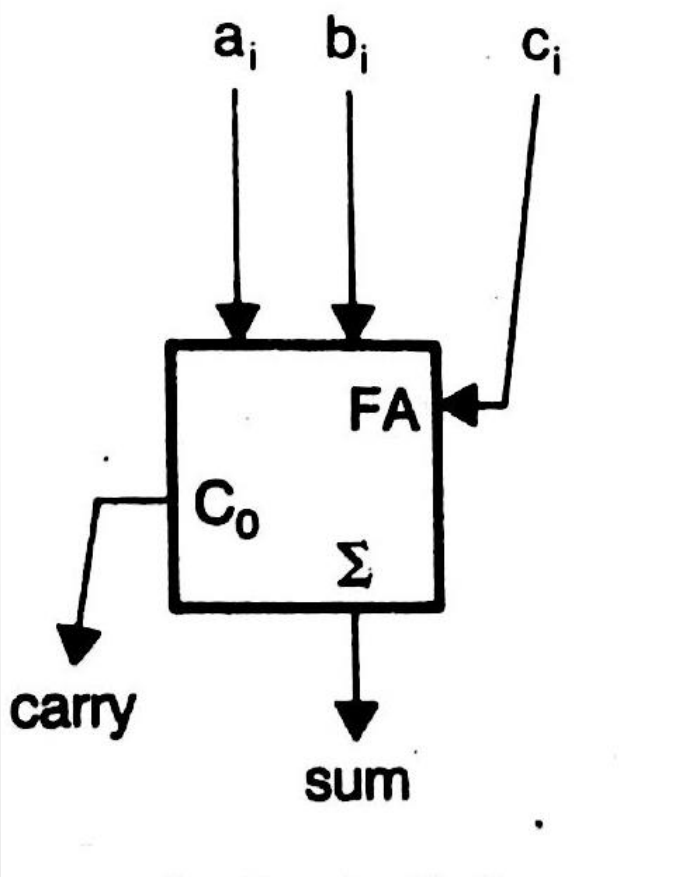
2 4 0 ----------→ Addition with carry propagation

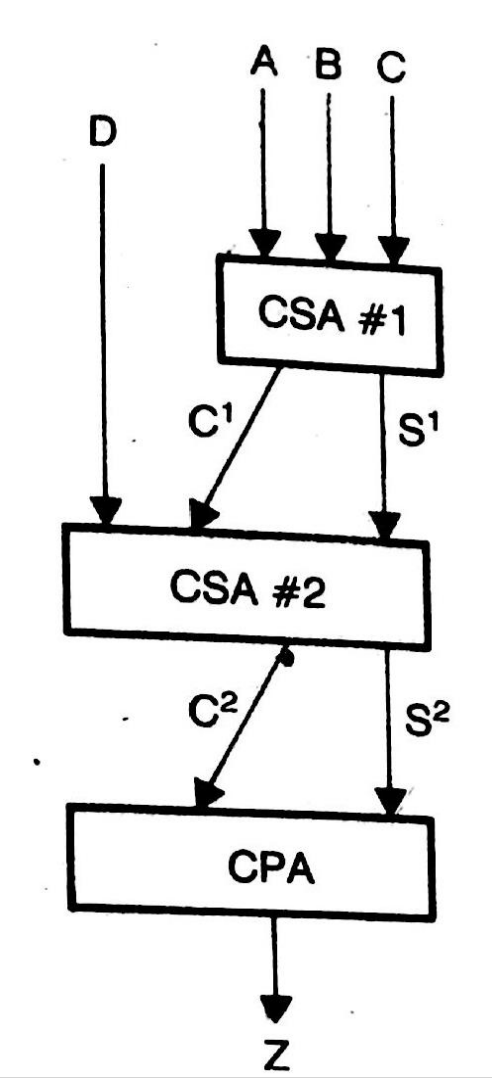
Carry Save Adder (CSA)

- Example with binary numbers: 110, 010, 101, 011
- 110
- 010
- 101
- 011
- _____
- _____

Carry Save Adder (CSA)

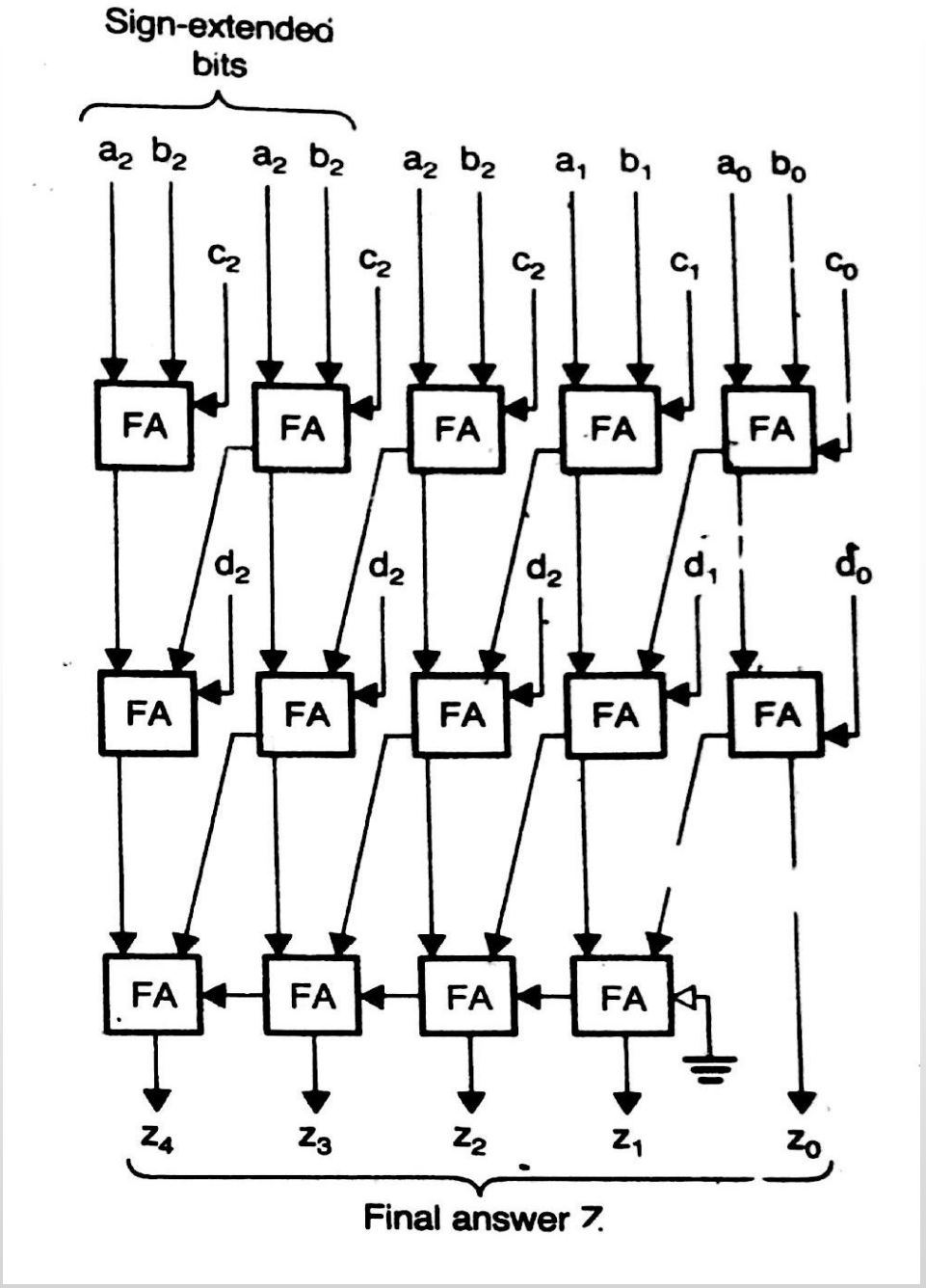
- Design a CSA using Full Adder (FA) blocks to add 4, 3-bit signed numbers..





Carry Save Adder (CSA)

3-bit signed operand range : (-4 to 3)
Sum of 4, 3-bit operands : (-16 to 12)
Hence 5-bits used for sum Z and 2 guard bits
are used for every operand. Guard bits are same as sign bit



Add and Shift multiplication algorithm

0111×0011

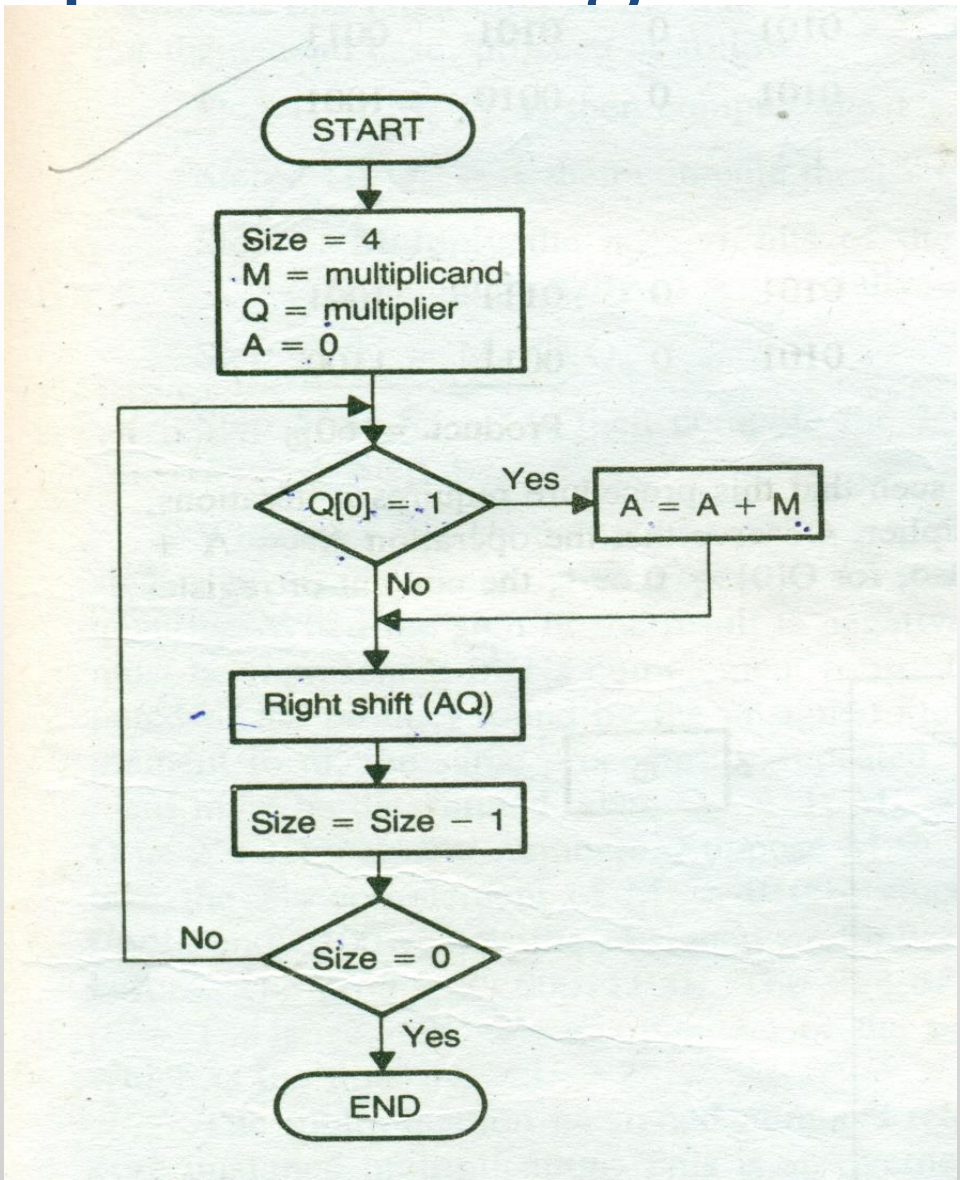
0111

0111

0000

0000

10101



15×15= 225

M	F(carry)	A	Q	Size	Remarks
1111	0	0000	1111	4	Initialization,
1111	0	1111	1111		A=A+M
1111	0	0111	1111	3	RS(FAQ),size=size -1
1111	1	0110	1111		A=A+M
1111	0	1011	0111	2	RS(FAQ),size=size -1
1111	1	1010	0111		A=A+M
1111	0	1101	0011	1	RS(FAQ),size=size -1
1111	1	1100	0011		A=A+M
1111	0	1110	0001	0	RS(FAQ),size=size -1

Solve using Add and Shift method

- ✓ *13 × 5*
- ✓ *18 × 12*

- Applicable only to unsigned numbers and number of additions is equal to --?

18 x 12 using add and shift method

M	A	Q	Q(-1)	Size	Remarks	Rough

Booth’s Algorithm

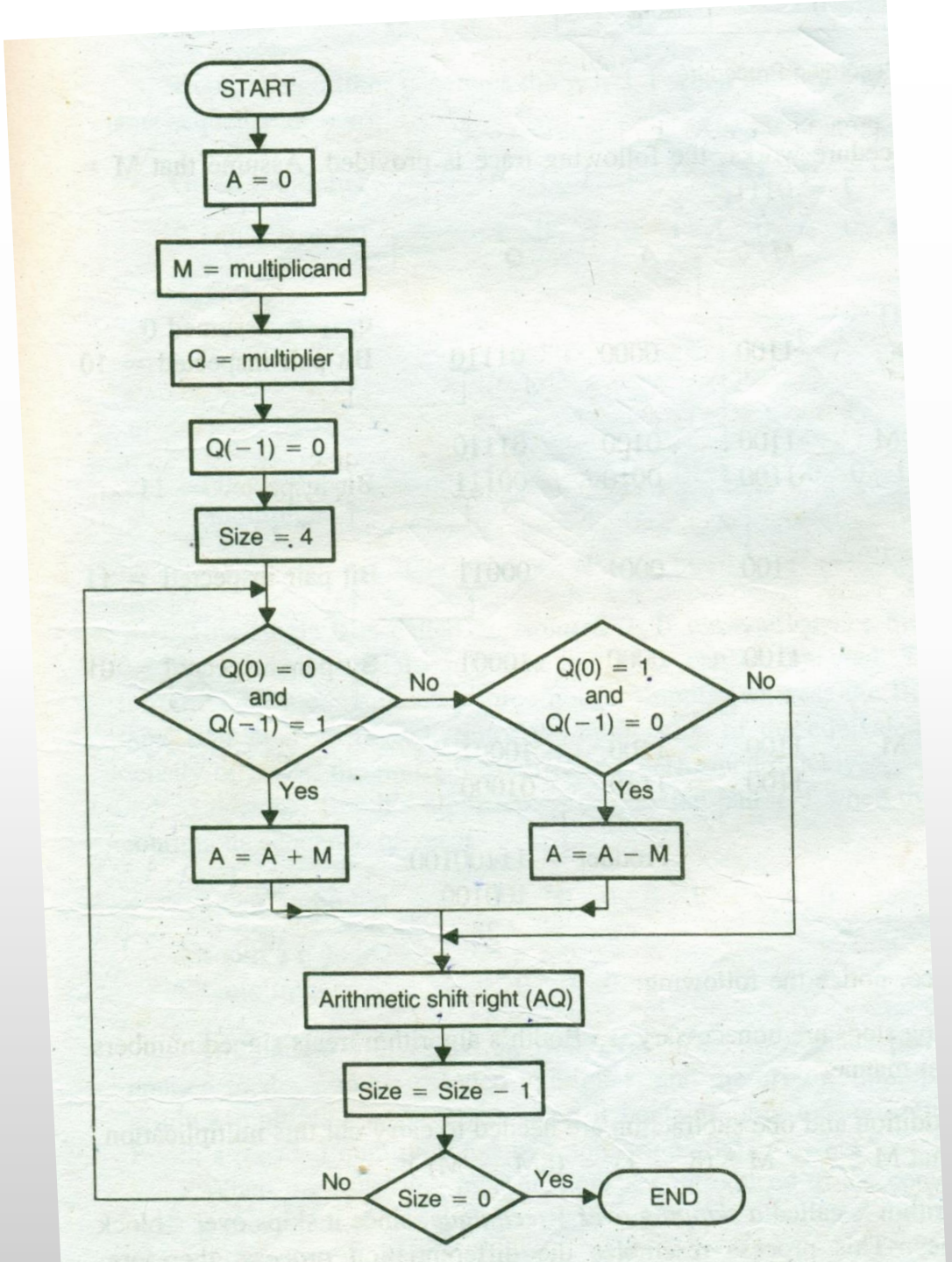
- Used for signed number multiplication
- Number of additions are less compared to add and shift method

- Consider 0110111 (55) as Multiplier which is recoded as shown below.

  01101110

  10 $\bar{1}$100 $\bar{1}$ 0(55)Booth’s multiplier

Booth's Algorithm



15×-15= -225

M=(15)=(01111), -M= (-15)= (10001), Q = (10001)
Note: Ignore the carry generated during A+M addition

M	A	Q	Q(-1)	Size	Remarks	Rough
01111	00000	10001	0	5	Initialization	
01111	10001	10001	0	5	A=A-M	
01111	11000	11000	1	4	ARS(AQ),size -1	
01111	00111	11000	1	4	A=A+M	
01111	00011	11100	0	3	ARS(AQ),size -1	
01111	00001	11110	0	2	ARS(AQ),size -1	
01111	00000	11111	0	1	ARS(AQ),size -1	
01111	10001	11111	0	1	A=A-M	
01111	11000	11111	1	0	ARS(AQ),size -1	

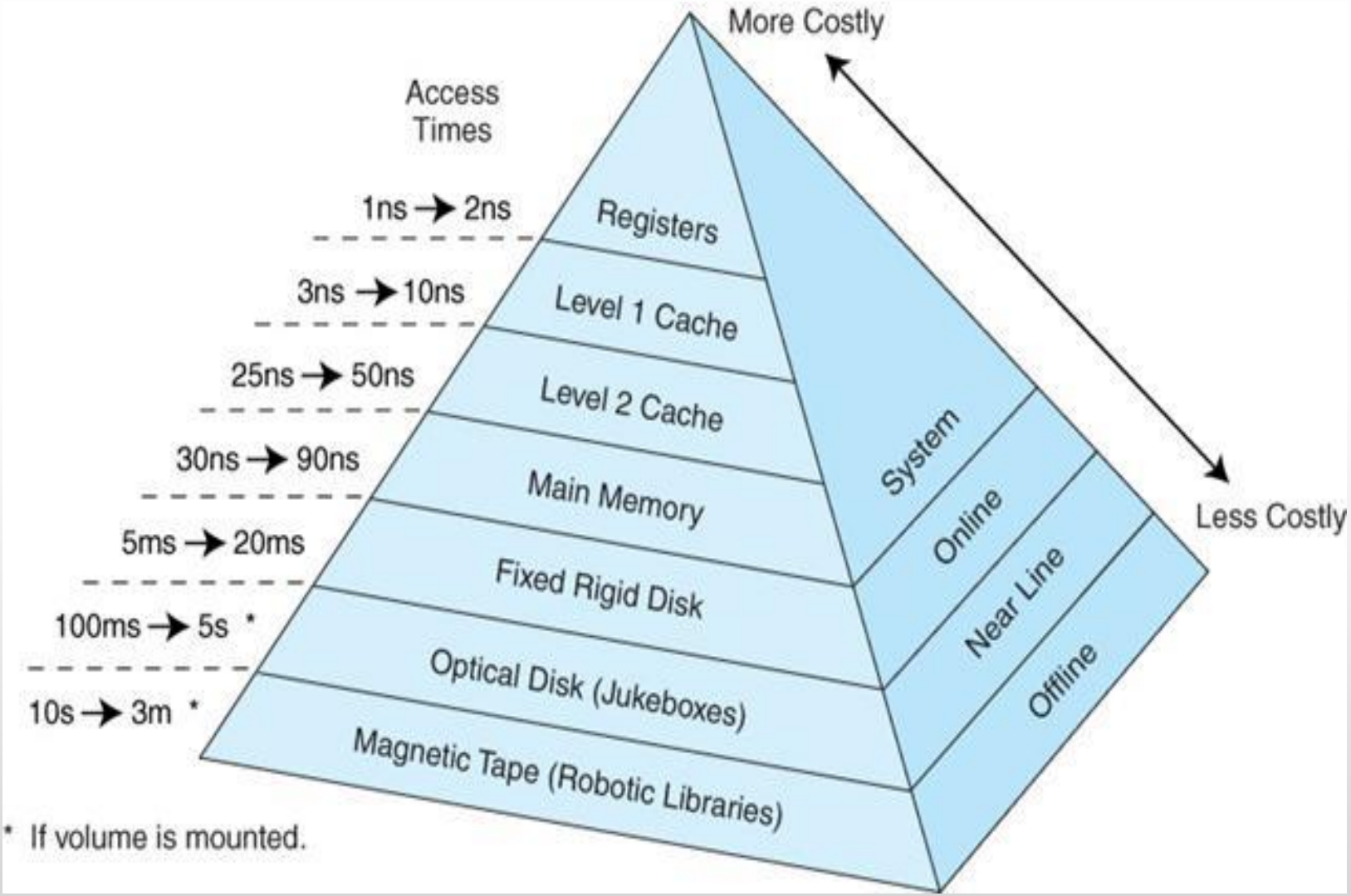
Solve using Booth’s Algorithm

- -13 × 7
- -14 × 11
- 9 × 10

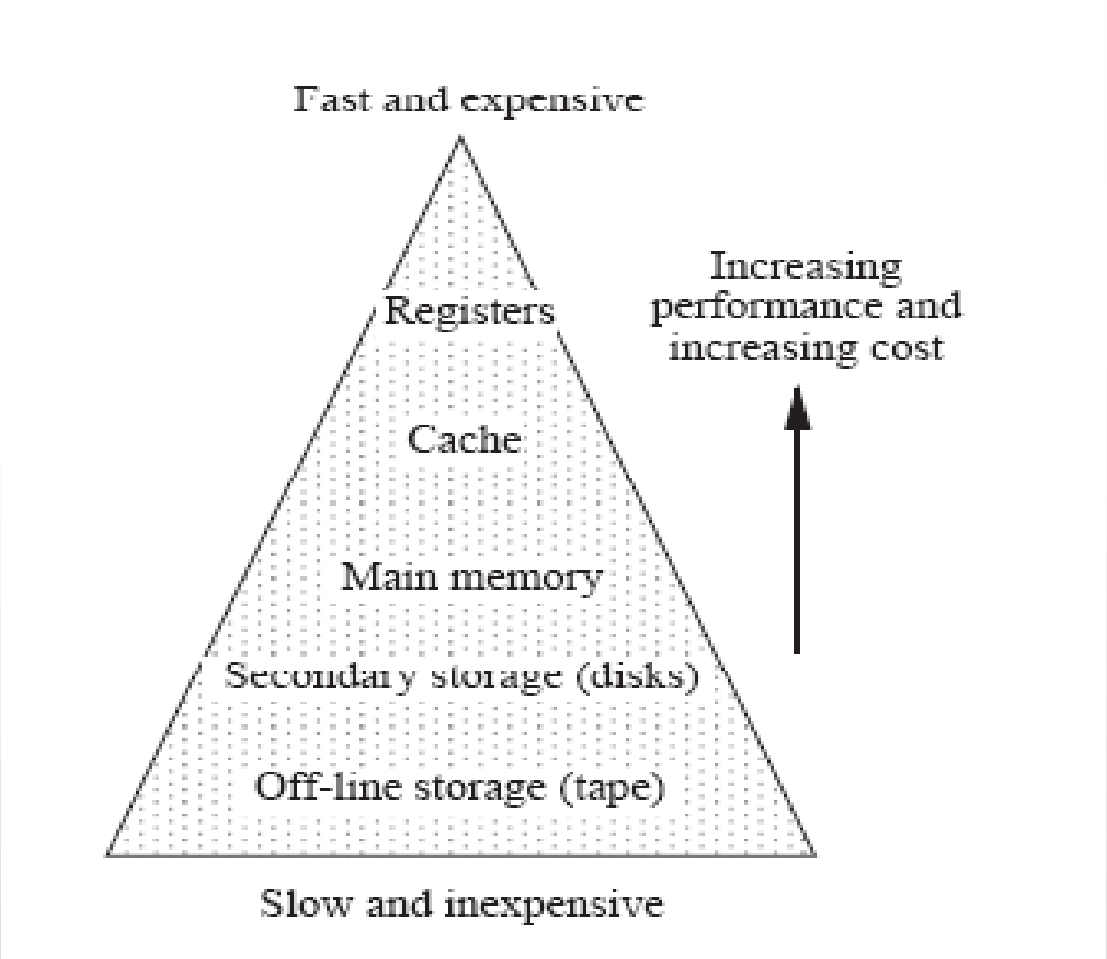
(-13)x (7)

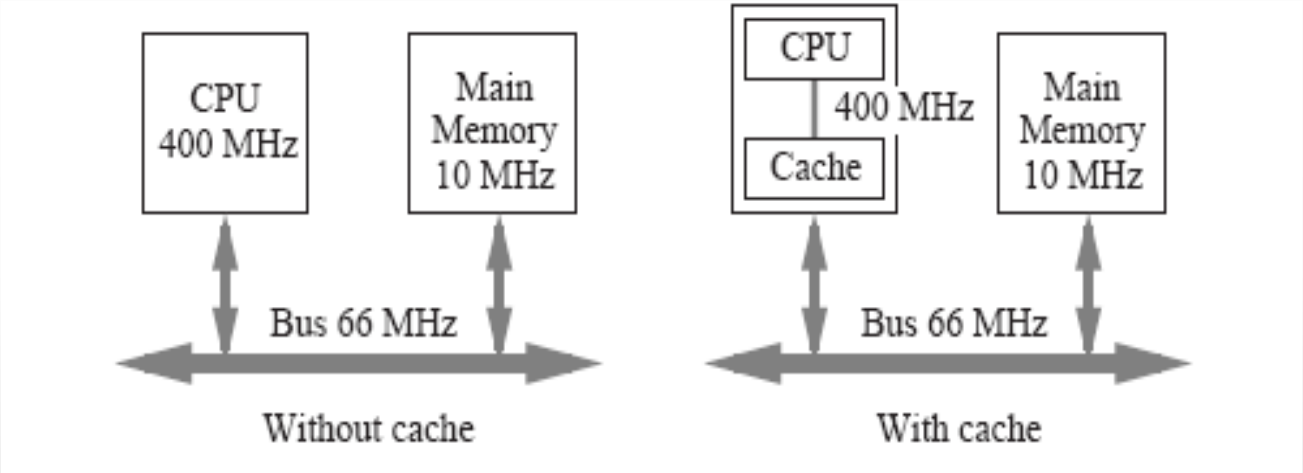
M	A	Q	Q(-1)	Size	Remarks	Rough

Memory Hierarchy



Memory Hierarchy




CPU
400 MHz
Main
Memory
10 MHz
Bus 66 MHz
Without cache
CPU
400 MHz
Cache
Main
Memory
10 MHz
Bus 66 MHz
With cache

Cache mapping

Commonly used methods:

- Associative Mapped Cache
- Direct-Mapped Cache
- Set-Associative Mapped Cache

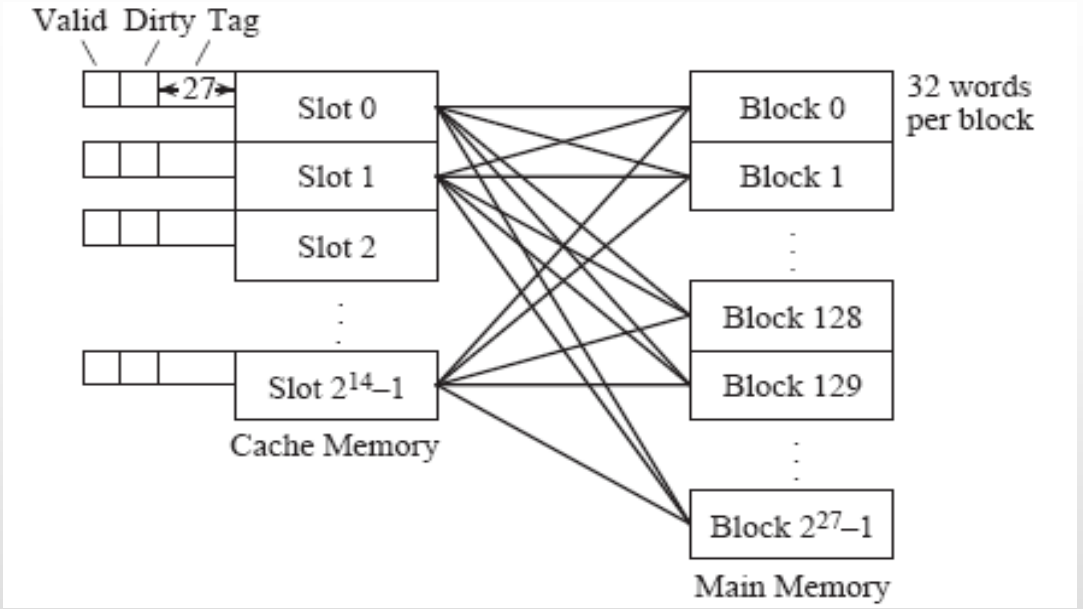
Consider $N=2^{n,}$ no. of cache blocks

$M=2^{m,}$ no. of main memory blocks

$S=2^{s}$ no. of cache blocks per set

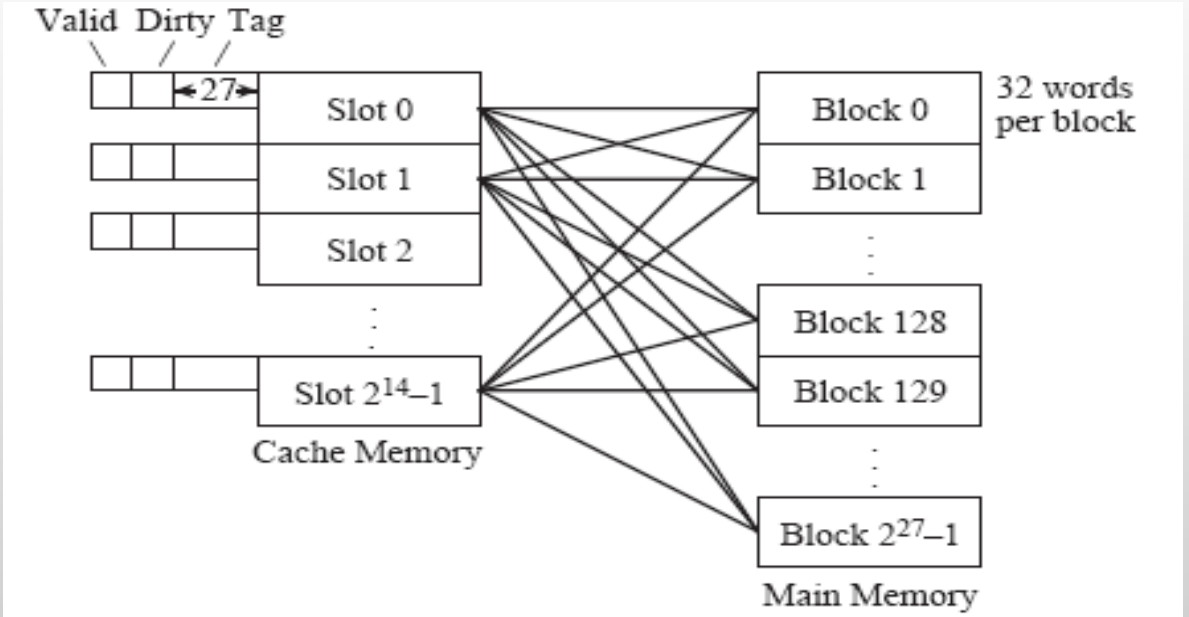
Associative Mapped Cache

- Any main memory blocks can be mapped into each cache slot.
- To keep track of which one of the 2^{27} possible blocks is in each slot, a 27-bit tag field is added to each slot.



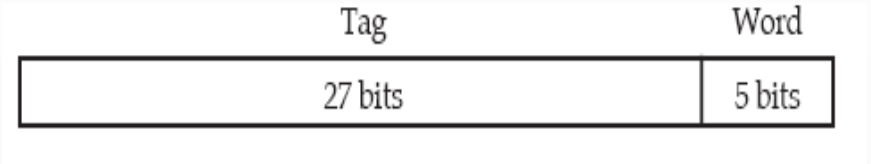
Associative Mapped Cache

- Valid bit is needed to indicate whether or not the slot holds a line that belongs to the program being executed.
- Dirty bit keeps track of whether or not a line has been modified while it is in the cache.



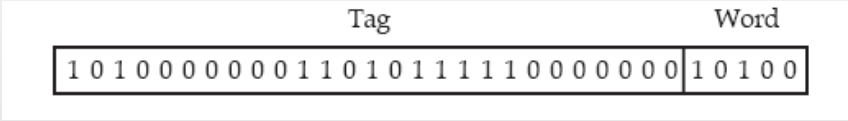
Associative Mapped Cache

- The mapping from main memory blocks to cache slots is performed by partitioning an address into fields.
- For each slot, if the valid bit is 1, then the tag field of the referenced address is compared with the tag field of the slot.



Associative Mapped Cache

- How an access to the memory location $(A035F014)_{16}$ is mapped to the cache.
- If the addressed word is in the cache, it will be found in word $(14)_{16}$ of a slot that has a tag of $(501AF80)_{16}$, which is made up of the 27 most significant bits of the address.



Associative Mapped Cache

Advantages

- Any main memory block can be placed into any cache slot.
- Regardless of how irregular the data and program references are, if a slot is available for the block, it can be stored in the cache.

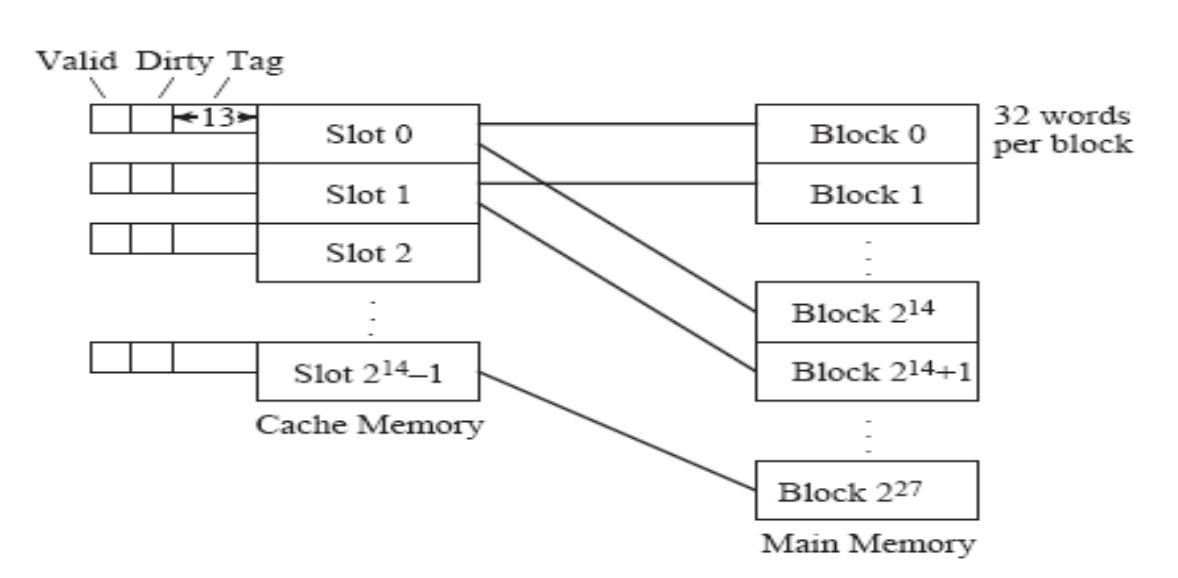
Associative Mapped Cache

Disadvantages

- Considerable hardware overhead needed for cache bookkeeping.
- There must be a mechanism for searching the tag memory in parallel.

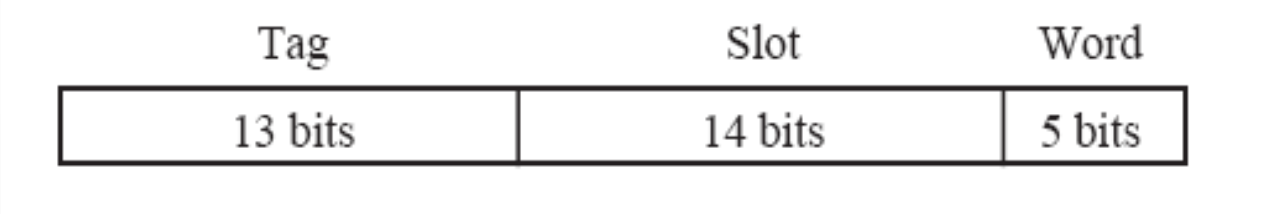
Direct-Mapped Cache

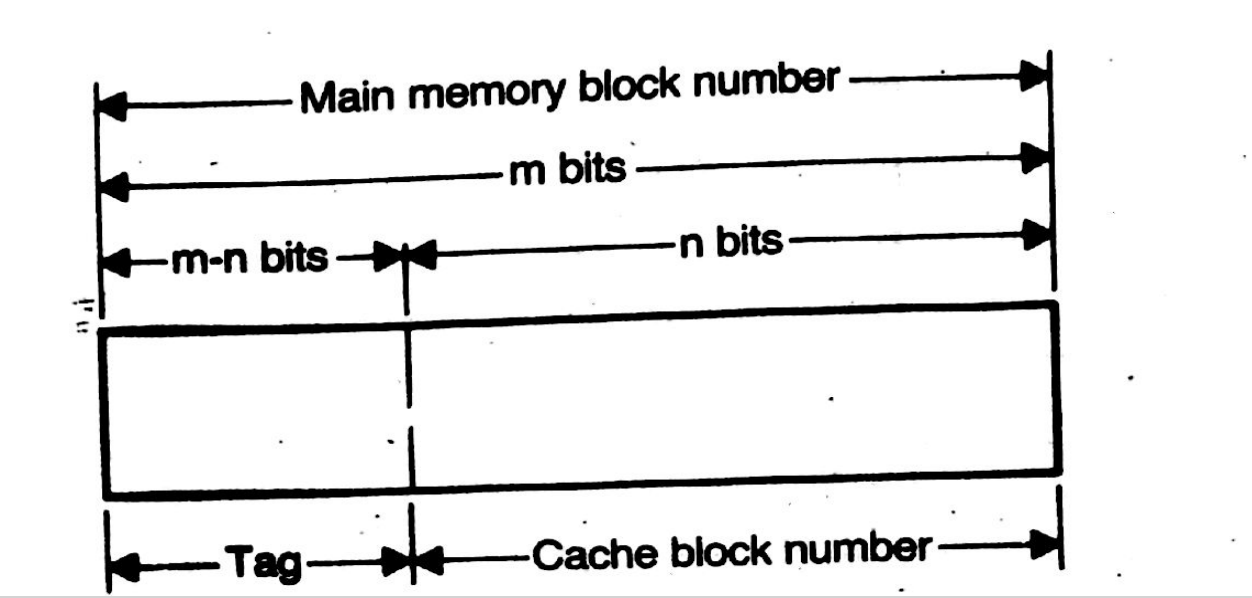
- Each cache slot corresponds to an explicit set of main memory.
- In our example we have 2^{27} memory blocks and 2^{14} cache slots.
- A total of $2^{27} / 2^{14} = 2^{13}$ main memory blocks can be mapped onto each cache slot.



Direct-Mapped Cache

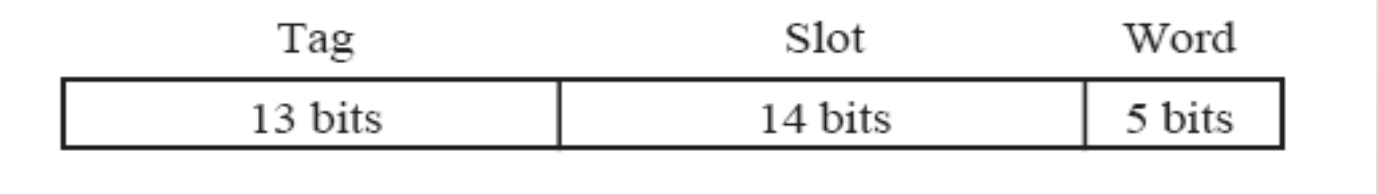
- The 32-bit main memory address is partitioned into a 13-bit tag field, followed by a 14-bit slot field, followed by a five-bit word field.





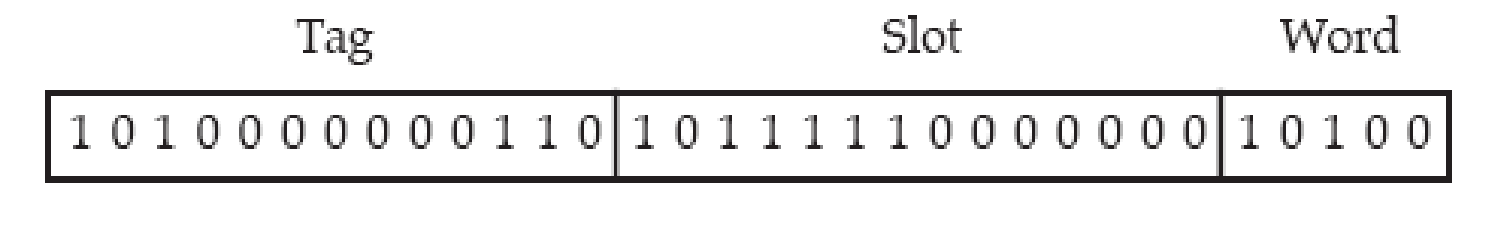
Direct-Mapped Cache

- When a reference is made to the main memory address, the slot field identifies in which of the 2^{14} slots the block will be found.
- If the valid bit is 1, then the tag field of the referenced address is compared with the tag field of the slot.



Direct-Mapped Cache

- How an access to memory location $(A035F014)_{16}$ is mapped to the cache.
- If the addressed word is in the cache, it will be found in word $(14)_{16}$ of slot $(2F80)_{16}$ which will have a tag of $(1406)_{16}$.



Direct-Mapped Cache

Advantages

- The tag memory is much smaller than in associative mapped cache.
- No need for an associative search, since the slot field is used to direct the comparison to a single field.

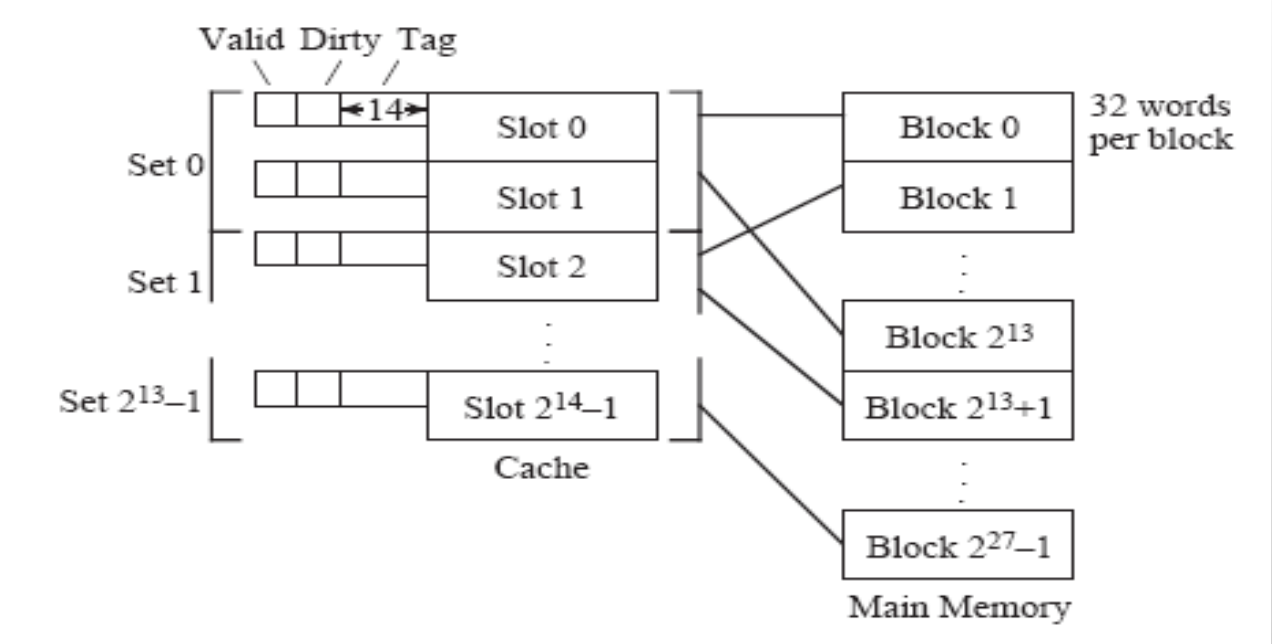
Direct-Mapped Cache

Disadvantages

- Consider what happens when a program references locations that are 2^{19} words apart, which is the size of the cache. Every memory reference will result in a miss, which will cause an entire block to be read into the cache even though only a single word is used.

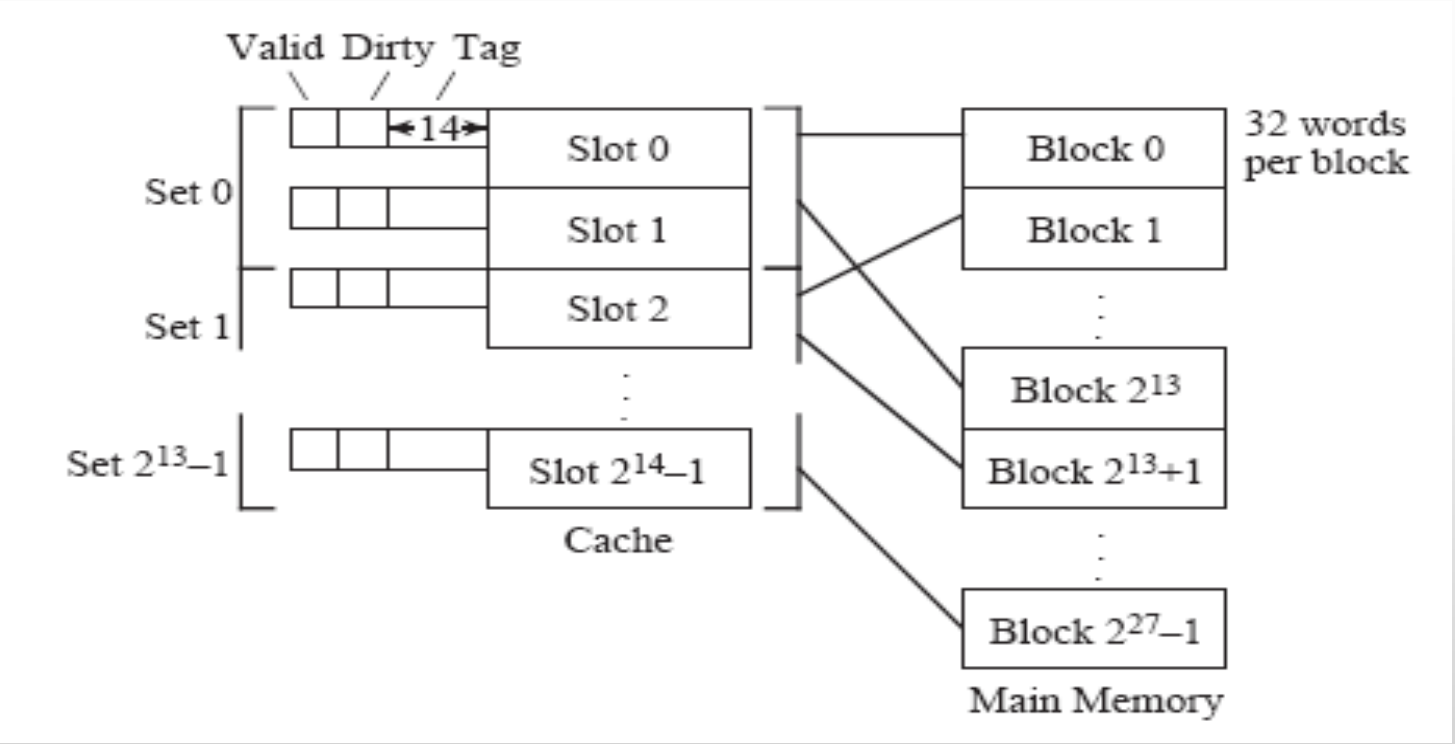
Set-Associative Mapped Cache

- Combines the simplicity of direct mapping with the flexibility of associative mapping
- For this example, two slots make up a set. Since there are 2^{14} slots in the cache, there are $2^{14}/2 = 2^{13}$ sets.



Set-Associative Mapped Cache

- When an address is mapped to a set, the direct mapping scheme is used, and then associative mapping is used within a set.



Set-Associative Mapped Cache

- The format for an address has 13 bits in the set field, which identifies the set in which the addressed word will be found. Five bits are used for the word field and 14-bit tag field.

Tag	Set	Word
14 bits	13 bits	5 bits

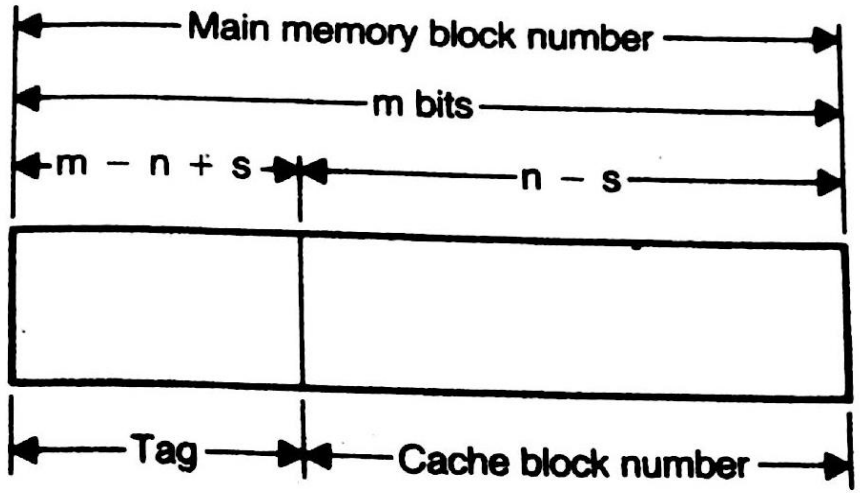
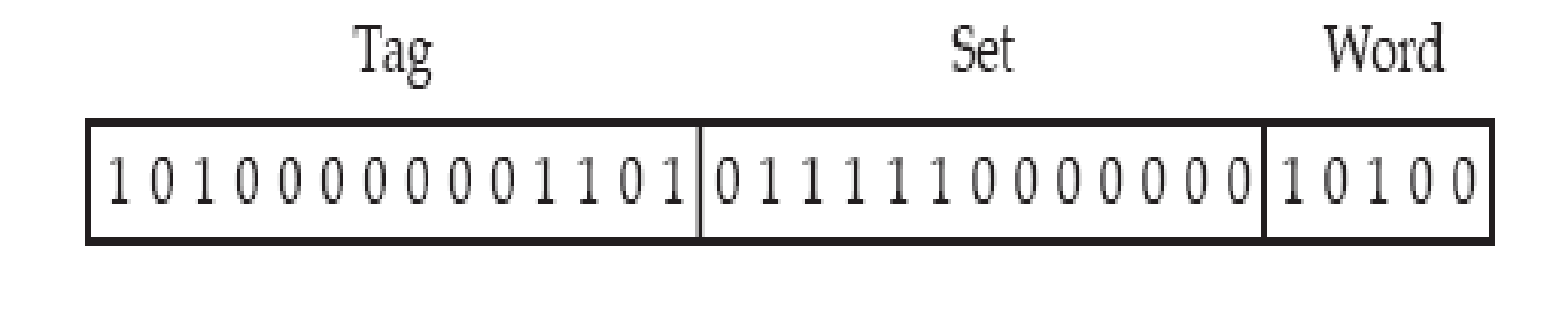


Figure 5.20 Address Format under Set Associative Mapping Scheme

Set-Associative Mapped Cache

- Consider again the address $(A035F014)_{16}$



Set-Associative Mapped Cache

Advantages

- In our example the tag memory increases only slightly from the direct mapping and only two tags need to be searched for each memory reference.

The set-associative cache is widely used in today’s microprocessors.

Exercise:

- A Computer system has a 32K of main memory and 4K of cache memory. The cache block size is 8 words. Calculate the tag field width for fully associative mapping, direct mapping and 4-way set associative mapping schemes.

Solution:

- Number of main memory blocks: M =?
- Bits to represent M => m =?
- Number of cache memory blocks : n = ?
- Bits to represent N =>n= ?
- no. of cache blocks per set = S =

- Associative tag width = m =
- Direct associative tag width = m-n
- Set associative tag width = m-n+s=

Internal bus organisation

Used for data transfer within the processing (i.e. execution)unit.

- Single bus organisation
- Two bus organisation
- Three bus organisation

Single bus architecture for R2←ro+r1

Requires three clock cycles

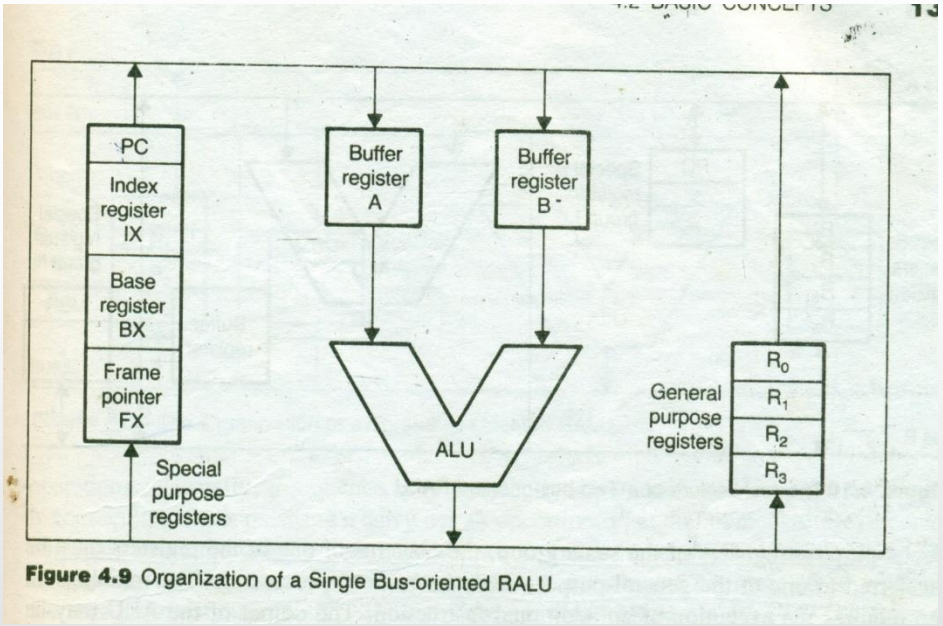


Figure 4.9 Organization of a Single Bus-oriented RALU

Two bus architecture for R2←ro+r1

Requires two clock cycles

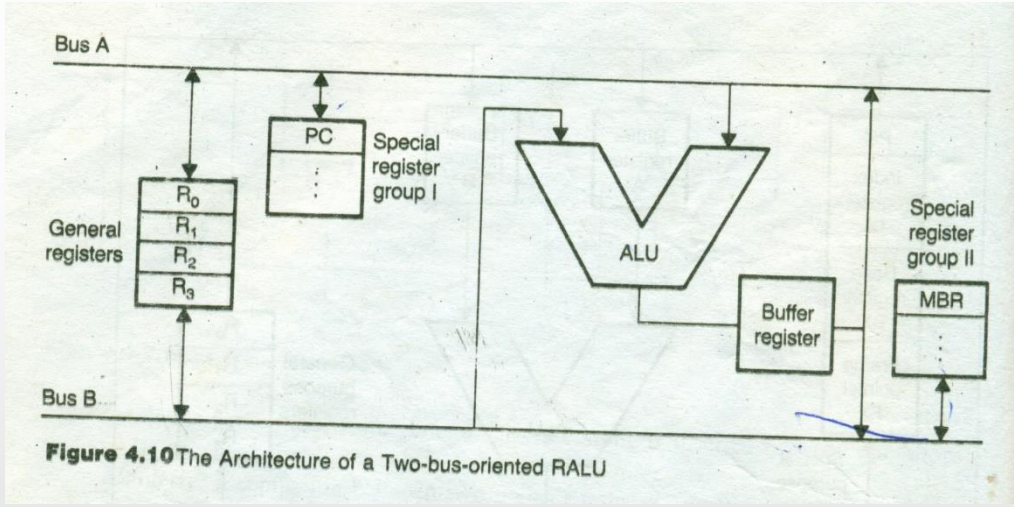


Figure 4.10 The Architecture of a Two-bus-oriented RALU

Three bus architecture for R2←ro+r1

Requires one clock cycle

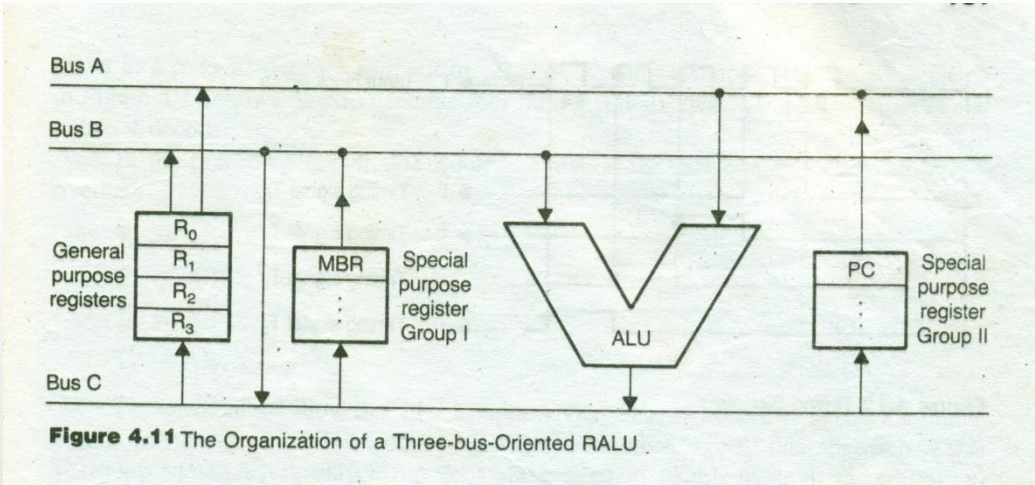


Figure 4.11 The Organization of a Three-bus-Oriented RALU

Control unit design

Two methods

- Hardwired approach
- Microprogramming approach

Hardwired design

- Designed using sequential circuit design procedure
- Final design is the interconnection of registers, gates, FFs, MUXs and other digital components
- Control lines from CU are physically connected to other devices.

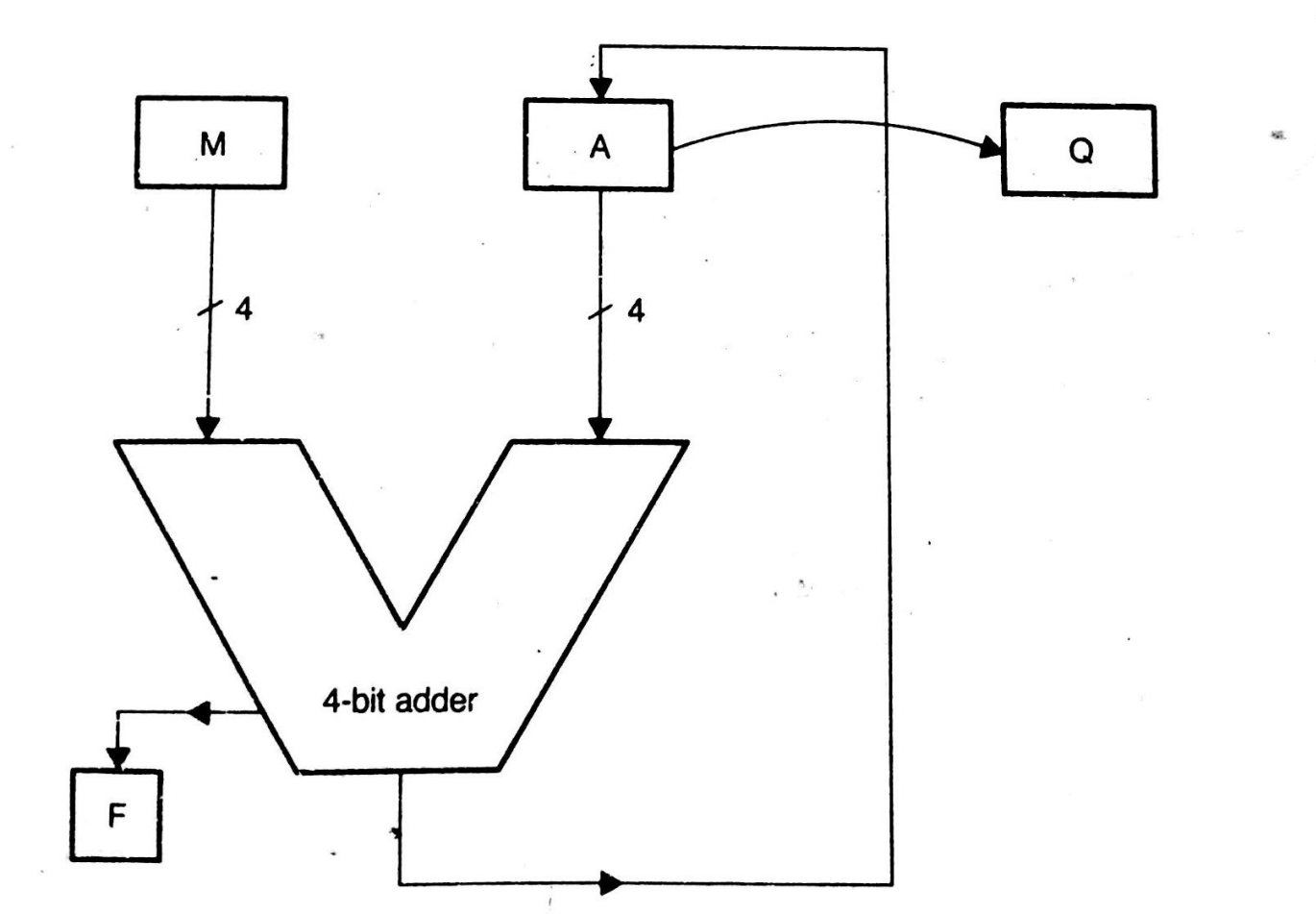
10 Steps for Hardwired Control

- 1. Define the task to be performed
- 2. Propose a trial processing section
- 3. Provide a Register Transfer Description algorithm based on the processing section outlined
- 4. Validate the algorithm using trial data
- 5. Describe the basic characteristics of the hardware 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 the state diagram of the controller
- 9. Specify the characteristics of the hardware elements to be used in the controller
- 10. Complete the controller design and draw a logic diagram of the final circuit

Step 1

- Design processing section and control unit for 4-bit booths multiplication algorithm

Step2 : trial processing section



Step 3

Declare register A[4], M[4],Q[5], L[3];

Declare buses Inbus[4], Outbus[4];

Start: A←0, M←Inbus, L←4; C0,c1,c2

Q[4:1]←Inbus, Q[0]←0; c3

Loop: If Q[1:0]=01 then goto Add;

If Q[1:0]=10 then goto sub;

Go to Rshift

Add: A←A+M; C4,C5

Go to Rshift

Sub: A←A-M; C4’,c5

Rshift: ASR(AQ), L←L-1; c6, c7

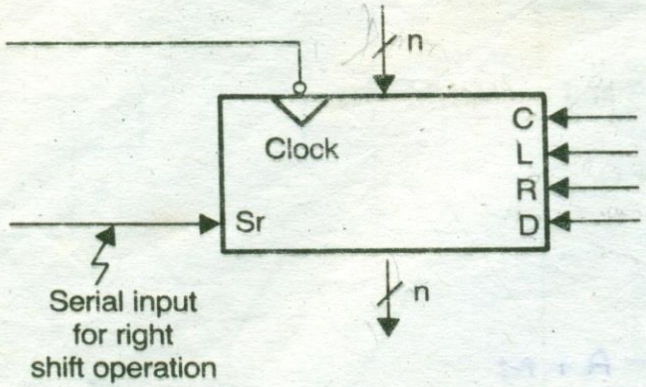
If L<>0 then goto loop

Outbus=A; c8

Outbus=Q[4:1]; c9

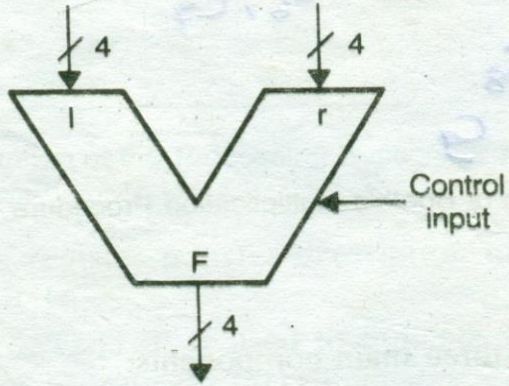
Halt: go to Halt;

Step 5



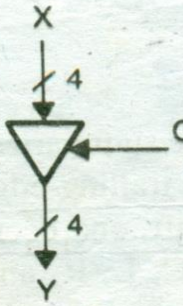
C	L	R	D	Clock	Action
1	0	0	0	↓	Clear
0	1	0	0	↓	Load external data
0	0	1	0	↓	Right shift
0	0	0	1	↓	Decrement by one
0	0	0	0	↓	No change

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

Step 6

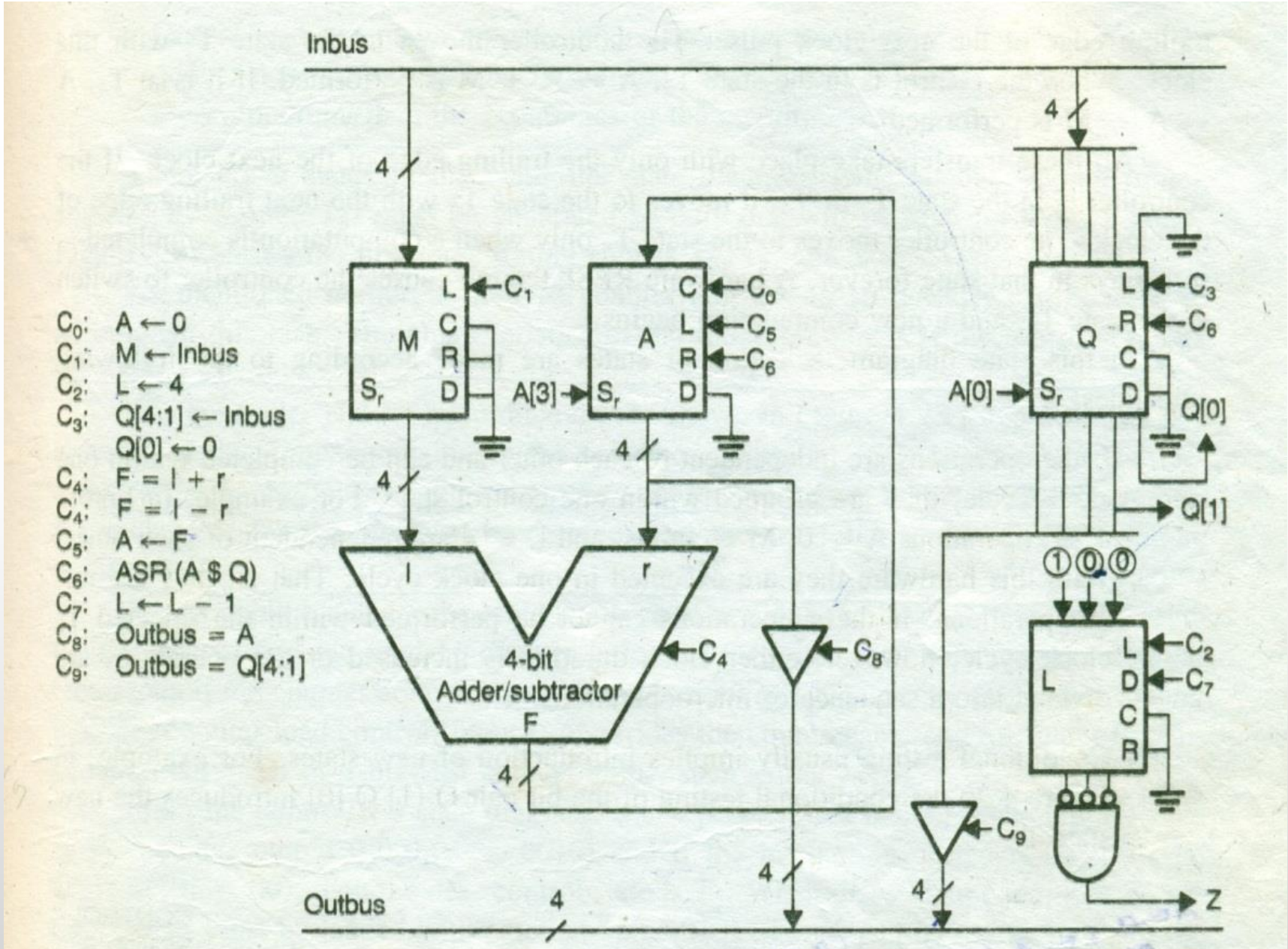


Figure 4.18 Processing Section of the Booth's Multiplier

Step 7

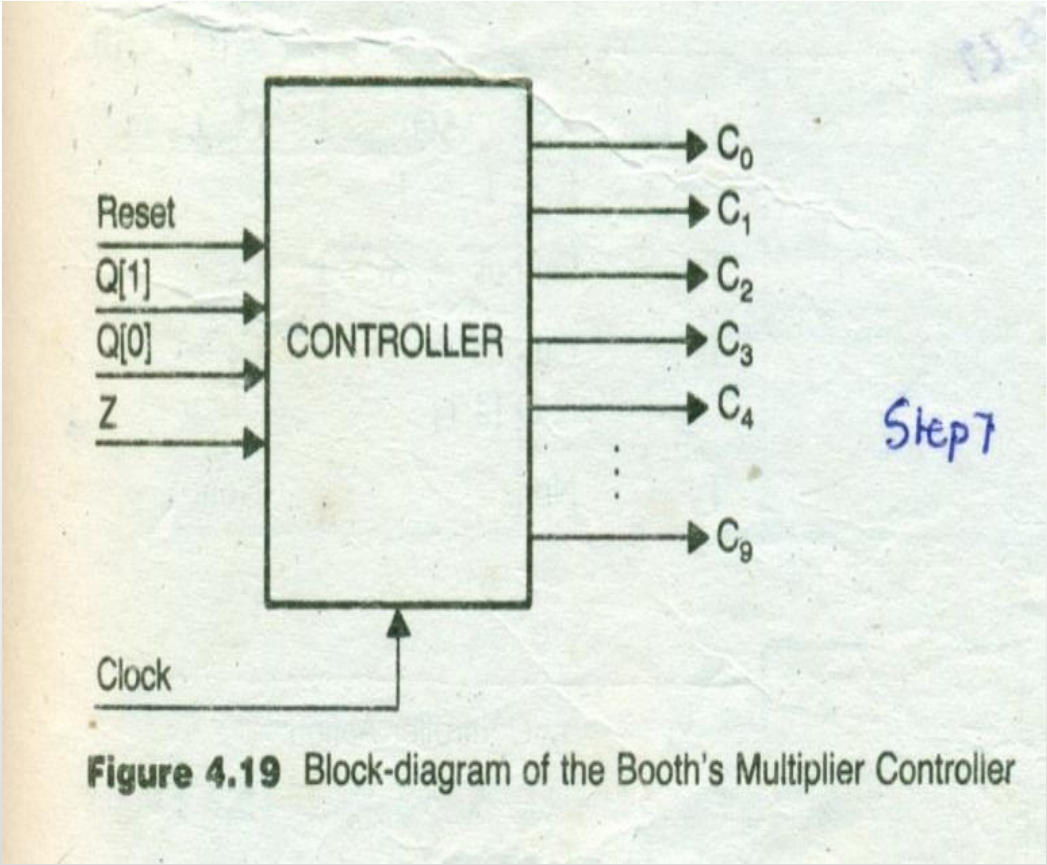
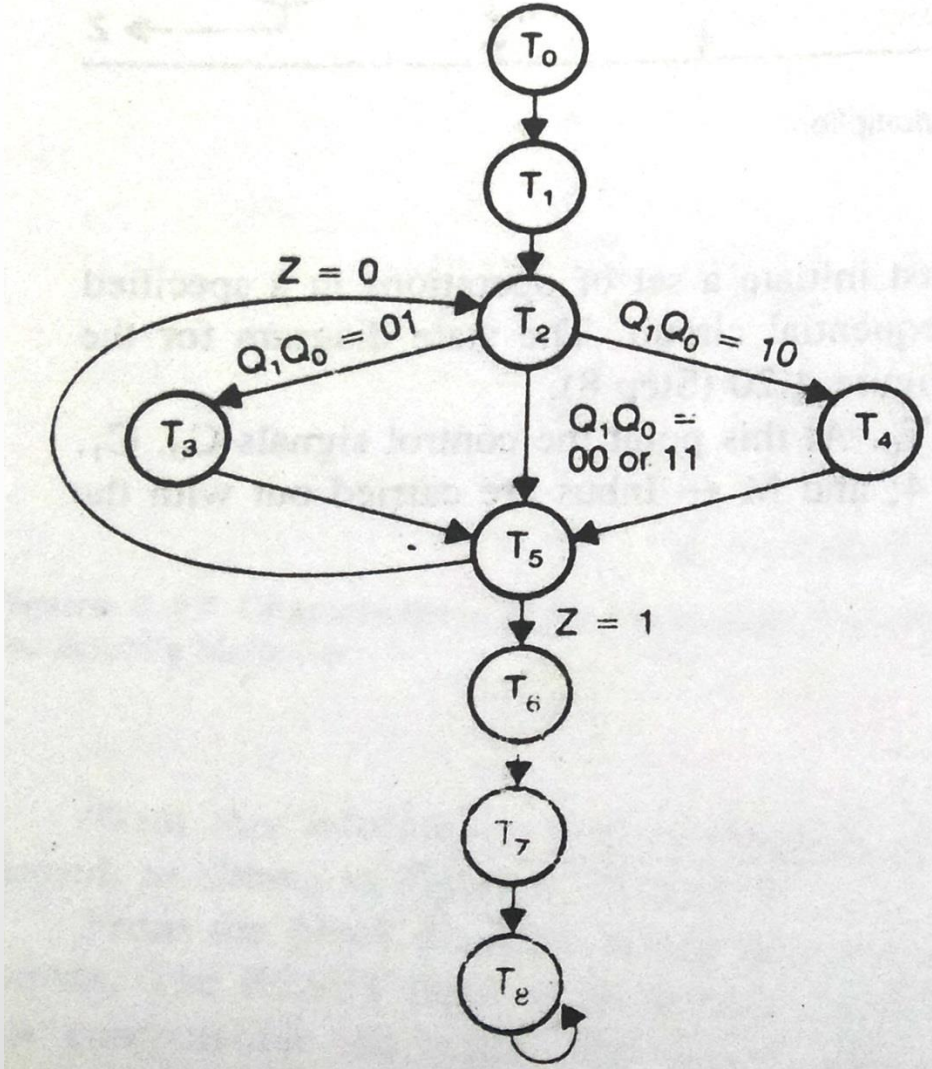


Figure 4.19 Block-diagram of the Booth's Multiplier Controller

Step 8



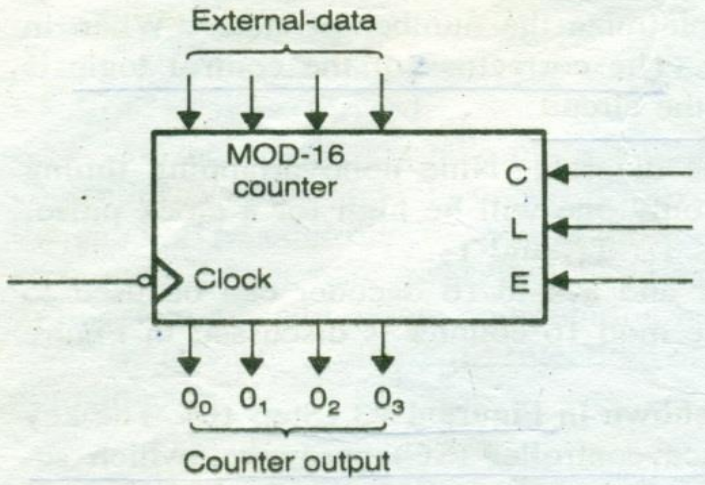
a. State Diagram

CONTROL STATE	OPERATION PERFORMED	CONTROL SIGNALS TO BE ACTIVATED
T_0	A ← 0, L ← 4, M ← Inbus	C_0, C_1, C_2
T_1	Q [4:1] ← Inbus, Q [0] ← 0	C_3
T_2	None	None
T_3	A ← A + M	C_4, C_5
T_4	A ← A − M	C_5 (C_4 = 0)
T_5	ASR (A$Q), L ← 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



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

b. Function Table

Figure 4.22 Characteristics of the Counter Used in the Controller Design

Step 10

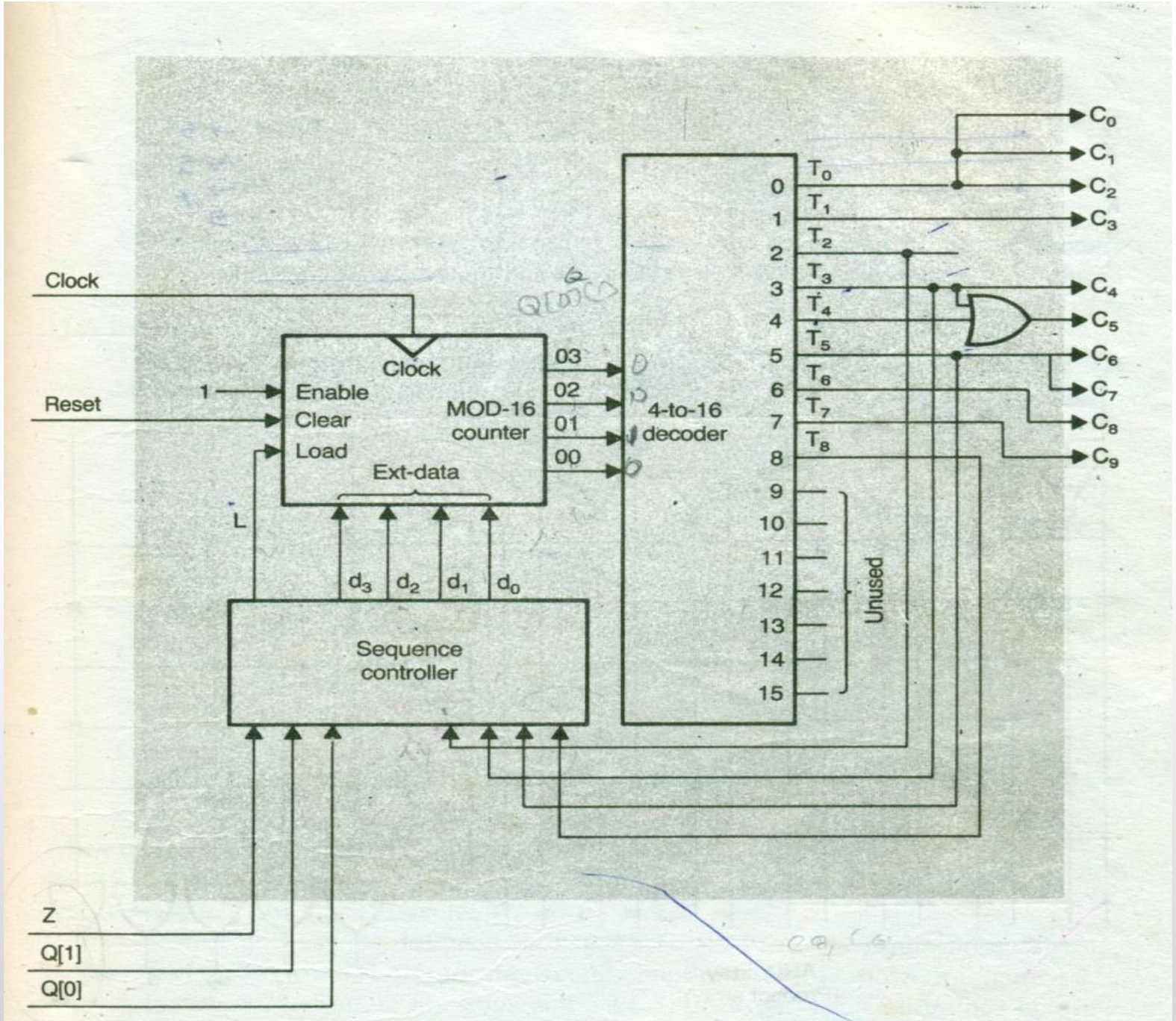
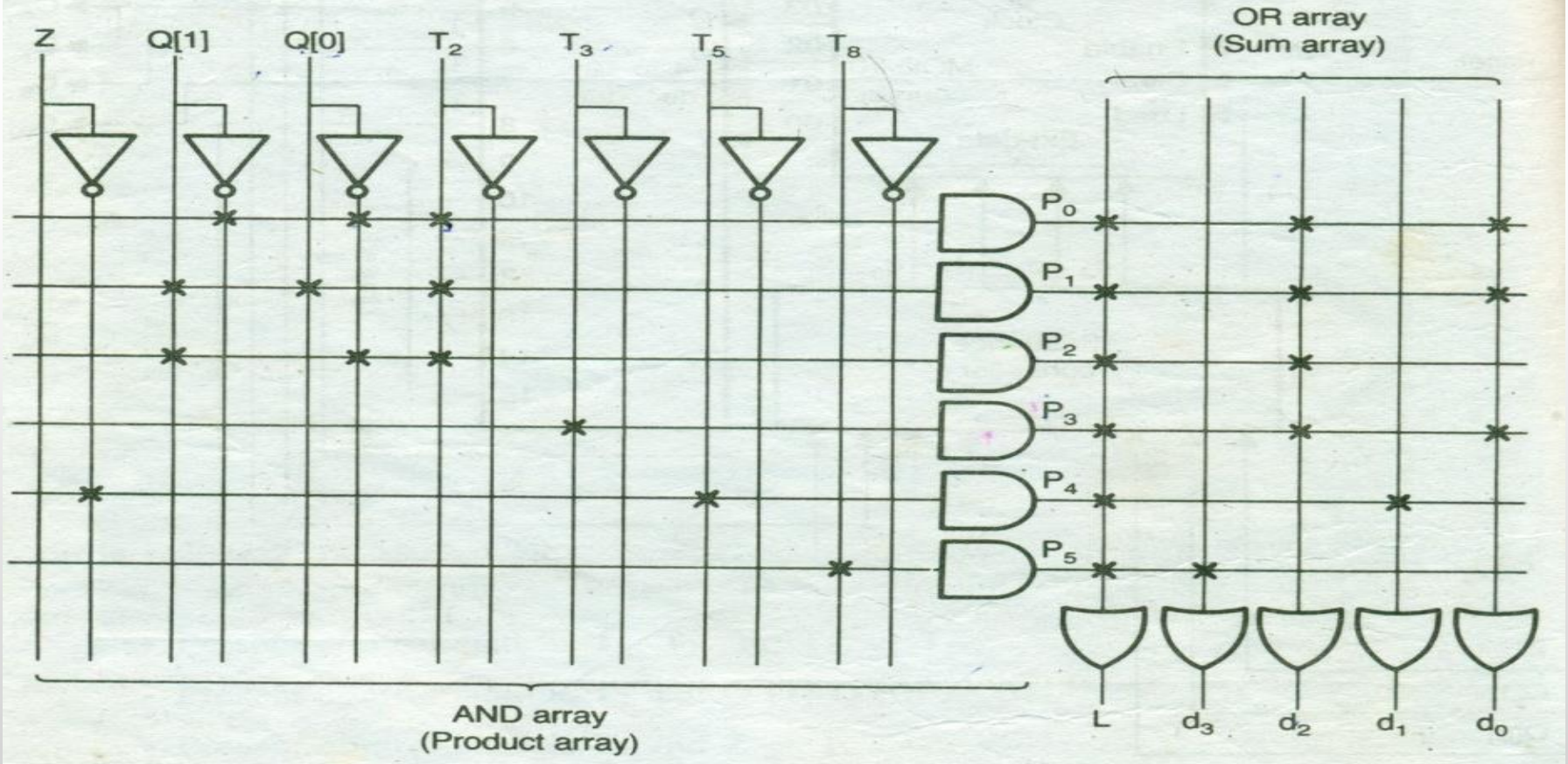


Figure 4.23 Logic Diagram of the Booth's Multiplier Controller

Z	Q [1]	Q [0]	T_2	T_3	T_5	T_8	L	External-data d3	d2	d1	d0	
X	0	0	1	X	X	X	1	0	1	0	1	→ 5
X	1	1	1	X	X	X	1	0	1	0	1	→ 5
X	1	0	1	X	X	X	1	0	1	0	0	→ 4
X	X	X	X	1	X	X	1	0	1	0	1	5
0	X	X	X	X	1	X	1	0	0	1	0	
X	X	X	X	X	X	1	1	1	0	0	0	

a. Truth Table



Step 10 contd..

table, a product term is generated in the PLA. The product terms generated are summarized as follows:

$$P_0 = Q\,[1]^1\,Q\,[0]^1\,T_2$$

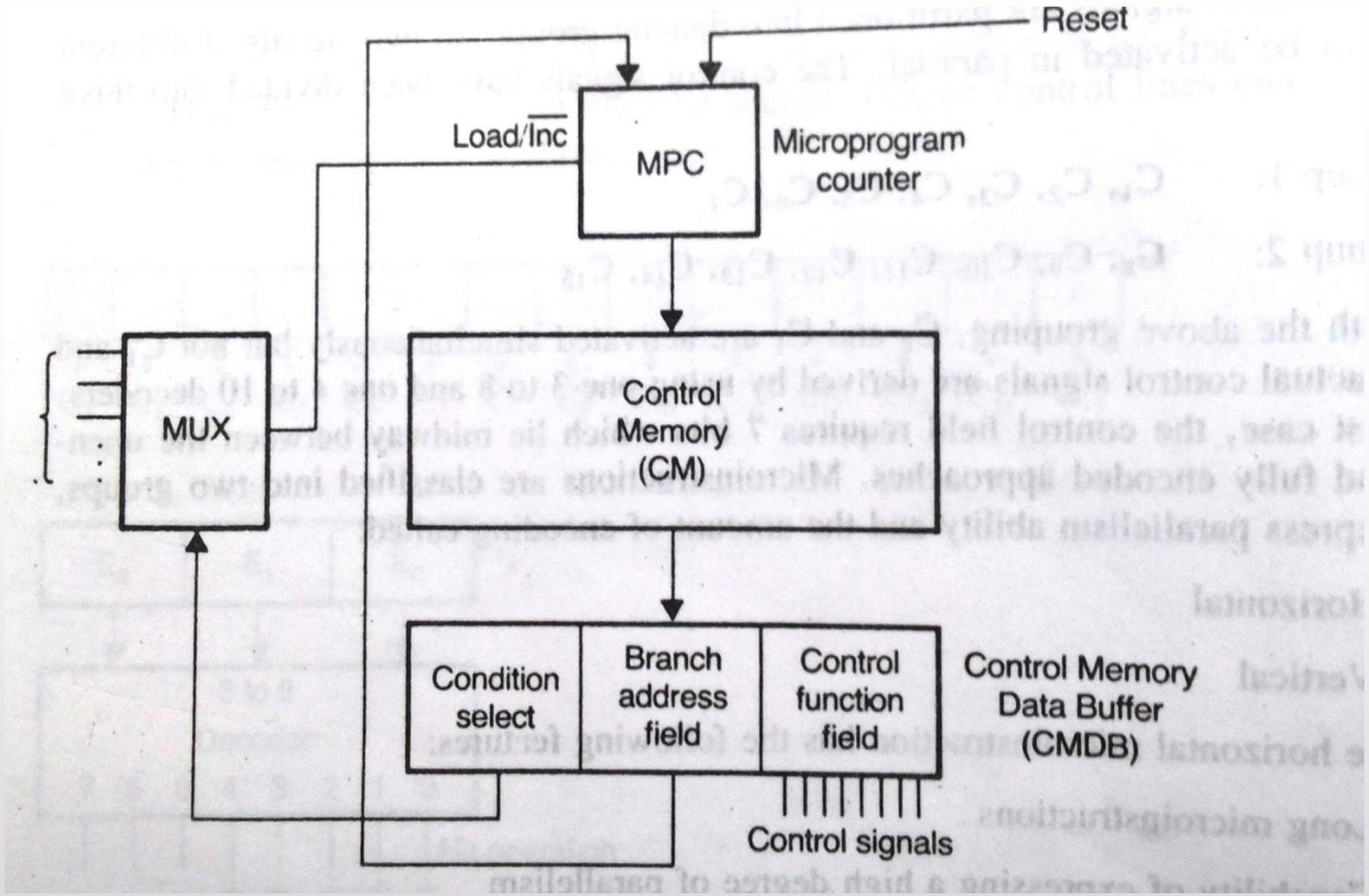
$$P_2 = Q\,[1]\,Q\,[0]\,T_2$$

$$P_2 = Q\,[1]\,Q\,[0]^1\,T_2$$

$$P_3 = T_3$$

$$P_4 = Z'T_5$$

Microprogrammed control unit



```
Control Memory  Declare register A[4], M[4],Q[5], L[3];
Address                 Declare buses Inbus[4], Outbus[4];
        0  0000 Start: A←0, M←Inbus, L←4;
        1  0001 Q[4:1]←Inbus, Q[0]←0;
        2  0010 Loop: If Q[1:0]=01 then Goto Add;
        3  0011 If Q[1:0]=10 then Goto sub;
        4  0100                 Go to Rshift
        5  0101         Add:    A←A+M;
        6  0110                 Go to Rshift
        7  0111         Sub:    A←A-M;
        8  1000 Rshift: ASR(AQ), L←L-1;
        9  1001 If L<>0 then Goto loop
        10 1010         Outbus=A;
        11 1011 Outbus=Q[4:1];
        12 1100 Halt:   Go to Halt;
```

CONDITION-SELECT FIELD	INTERPRETATION
000	No branching
001	Branch if Q [1] = 0 and Q [0] = 1
010	Branch if Q [1] = 1 and Q [0] = 0
011	Branch if Z = 0
100	Unconditional branching

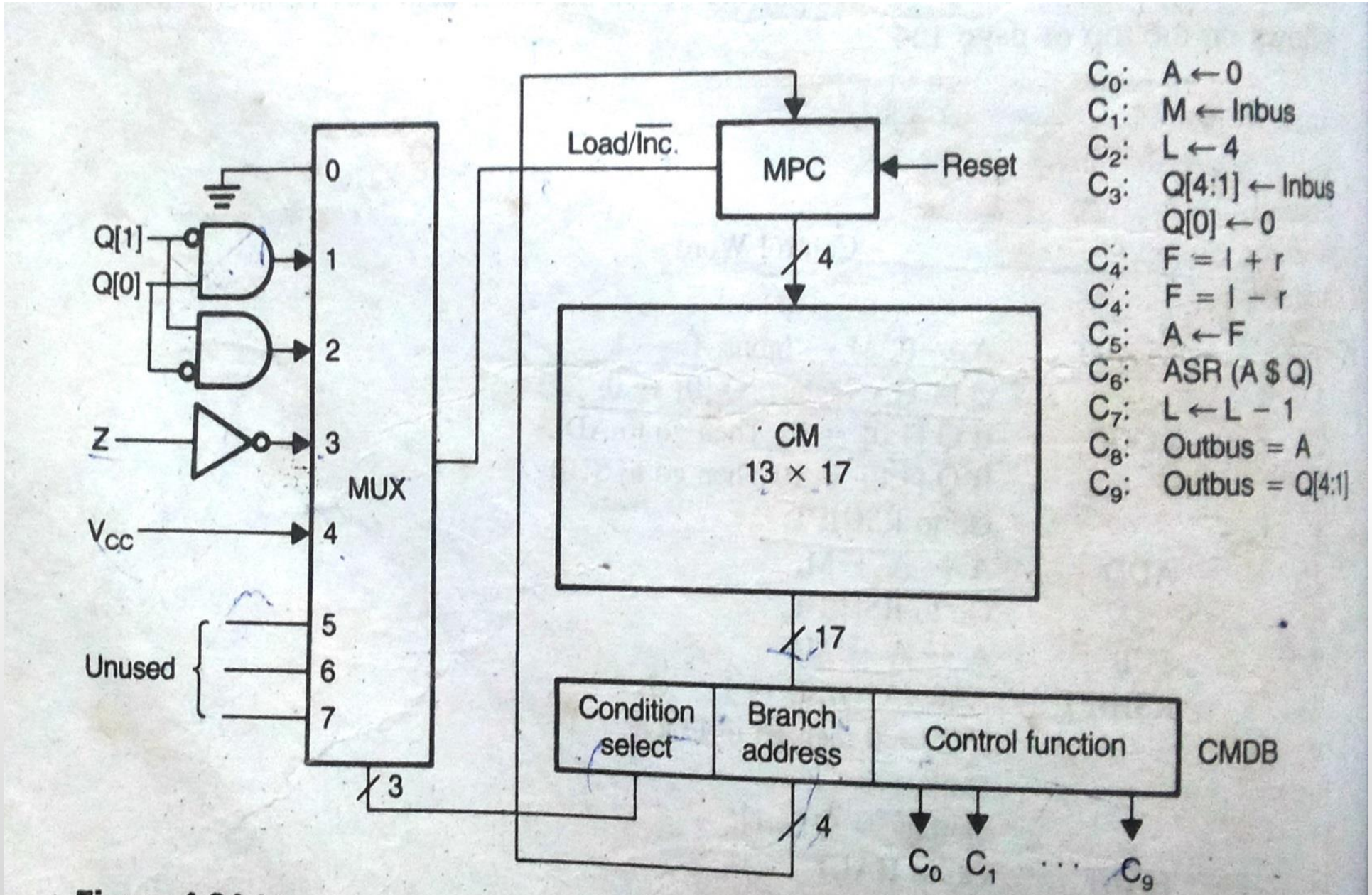


Figure 4.34 Microprogrammed Multiplier Control Unit

ROM Address													Control Word									
In decimal	In binary				Cond. Sel			Branch Addr.					Control Function									Comments
					c_{s2}	c_{s1}	c_{s0}	b_{r3}	b_{r2}	b_{r1}	b_{r0}	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	A ← 0, M ← Inbus, L ← 4
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	Q [4:1] ← Inbus, Q [0] ← 0
2	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	If Q [1:0] = 01 Then go to 5 (ADD)
3	0	0	1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	If Q [1:0] = 10 Then go to 7 (SUB)
4	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	go to 8 (RSHIFT)
5	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	A ← A + M
6	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	go to 8 (RSHIFT)
7	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	A ← A − M
8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	ASR (ASQ), L ← L − 1
9	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	If Z = 0 Then go to 2
10	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Outbus = A
11	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Outbus = Q [4:1]
12	1	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	Go to 12 (HALT)

ROM Address					Cond. Sel			Branch Addr.					Control Word									
In decimal	In binary												Control Function									Comments
					c_{s2}	c_{s1}	c_{s0}	b_{r3}	b_{r2}	b_{r1}	b_{r0}	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	A ← 0, M ← Inbus, L ← 4
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	Q [4:1] ← Inbus, Q [0] ← 0
2	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	If Q [1:0] = 01 Then go to 5 (ADD)
3	0	0	1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	If Q [1:0] = 10 Then go to 7 (SUB)
4	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	go to 8 (RSHIFT)
5	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	A ← A + M
6	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	go to 8 (RSHIFT)
7	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	A ← A − M
8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	ASR (A$Q), L ← L − 1
9	1	0	0	1	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	If Z = 0 Then go to 2
10	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Outbus = A
11	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Outbus = Q [4:1]
12	1	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	Go to 12 (HALT)

Figure 4.35 Binary Listing of the Microprogram For the 4 x 4 Booth's Multiplier

Input / Output(i/o)

- 3 ways of data transfer between i/o and computer
  - *Programmed i/o*
  - *Interrupt i/o*
  - *Direct memory access (DMA)*

- I/O ports are addressed using
  standard i/o or memory mapped i/o.

Interrupt i/0

- Two ways of servicing multiple interrupts
  - *Polled interrupts*
  - *Daisy chain techniques*

Polled interrupt

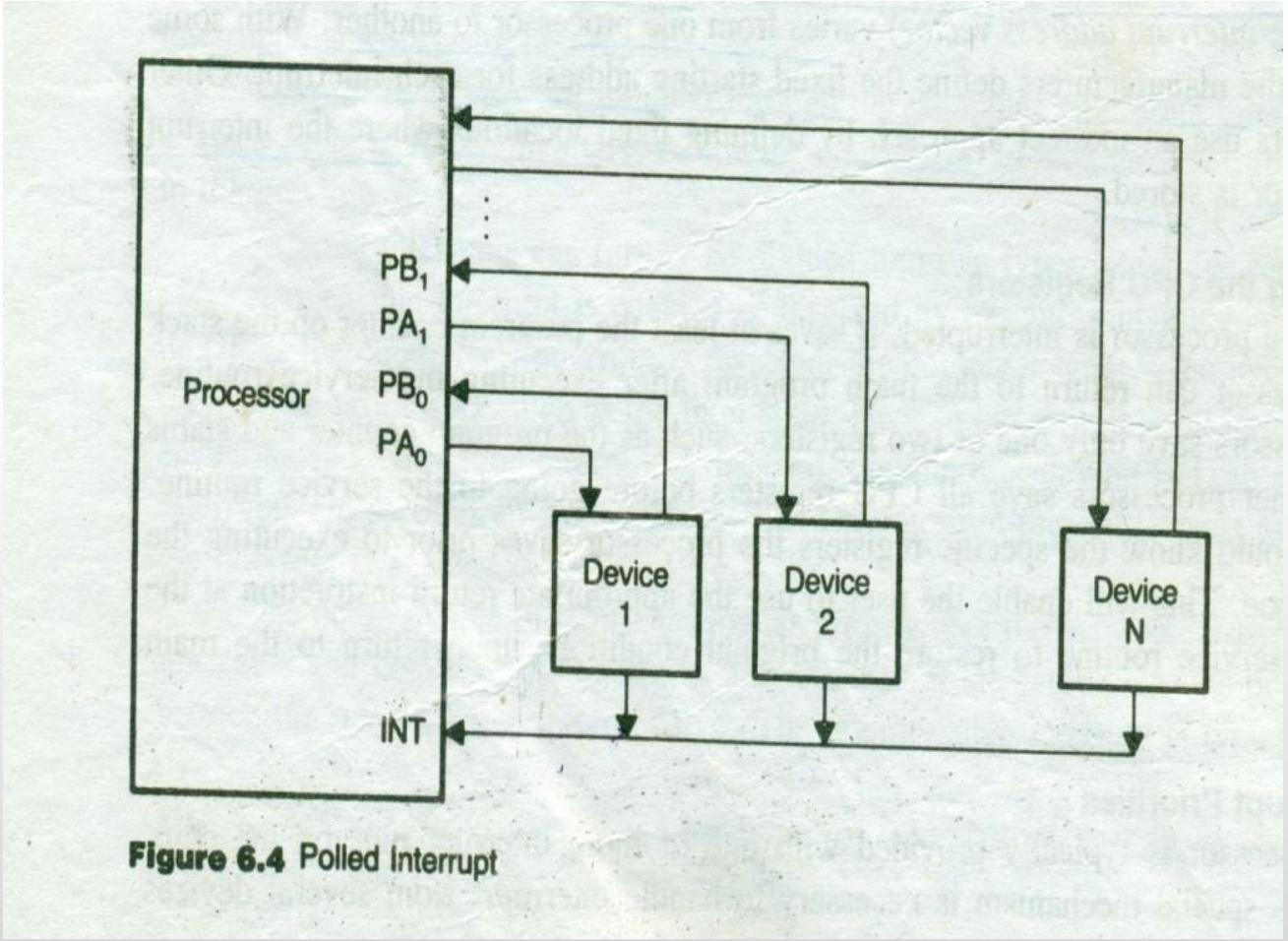


Figure 6.4 Polled Interrupt

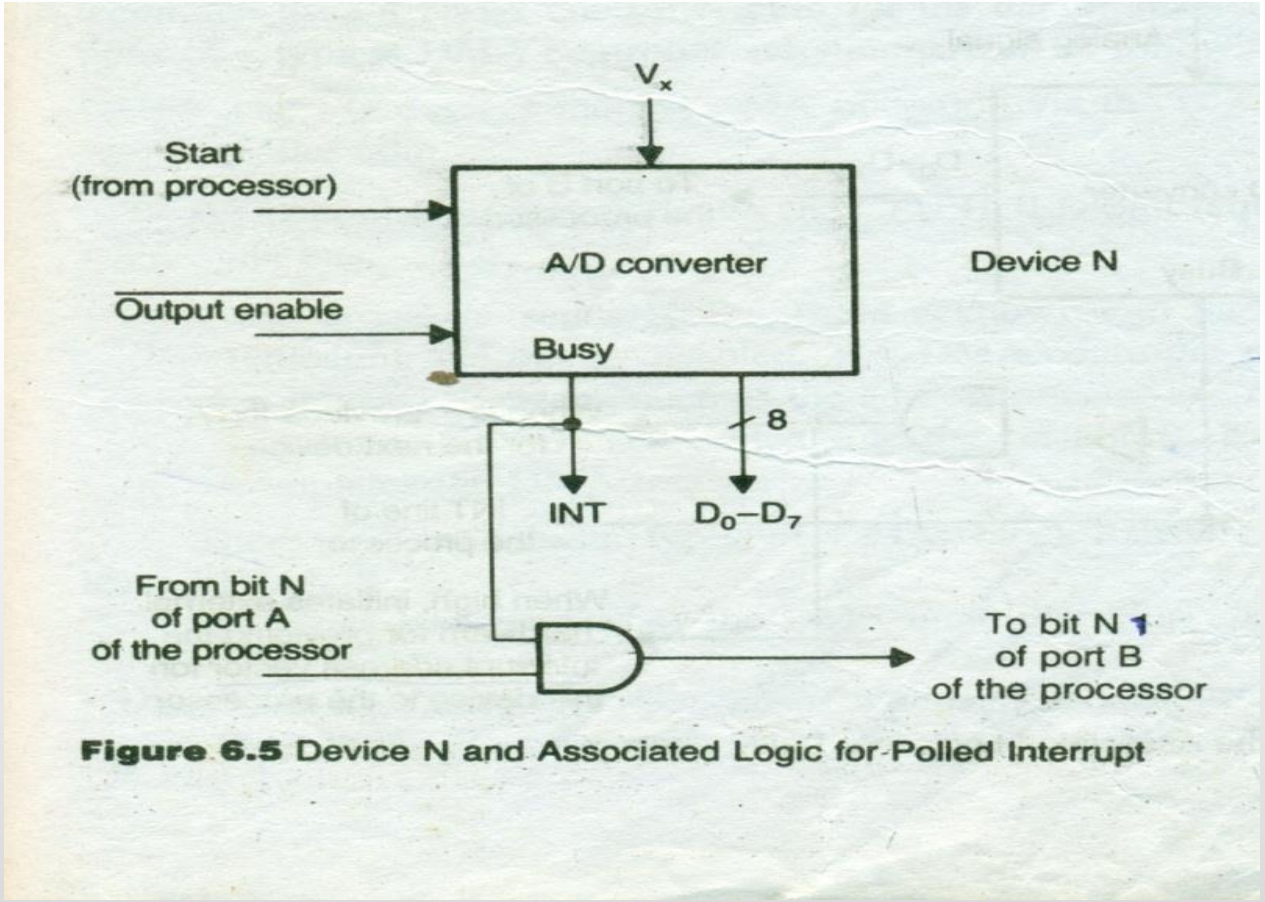


Figure 6.5 Device N and Associated Logic for Polled Interrupt

Daisy chain interrupt

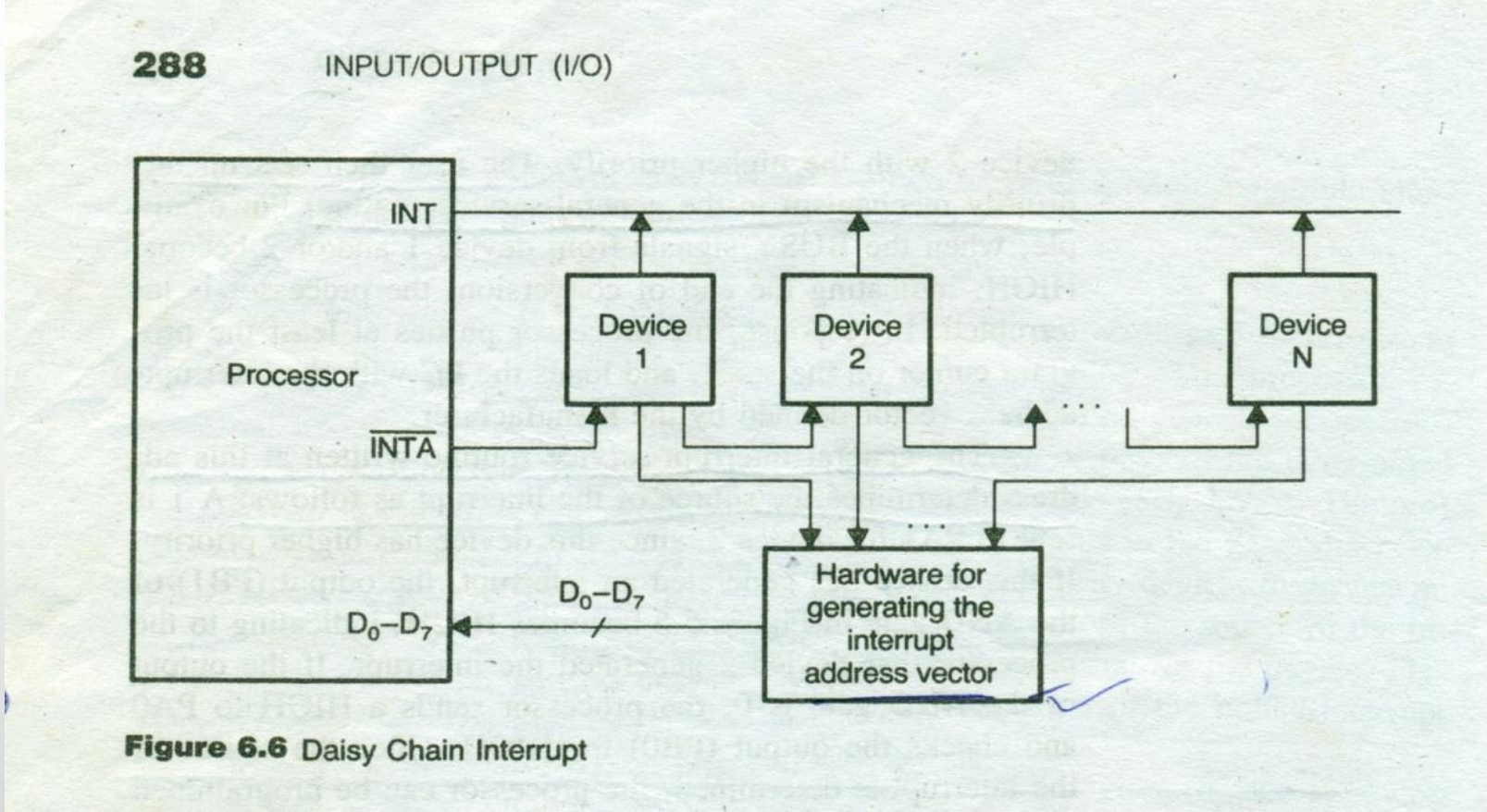


Figure 6.6 Daisy Chain Interrupt

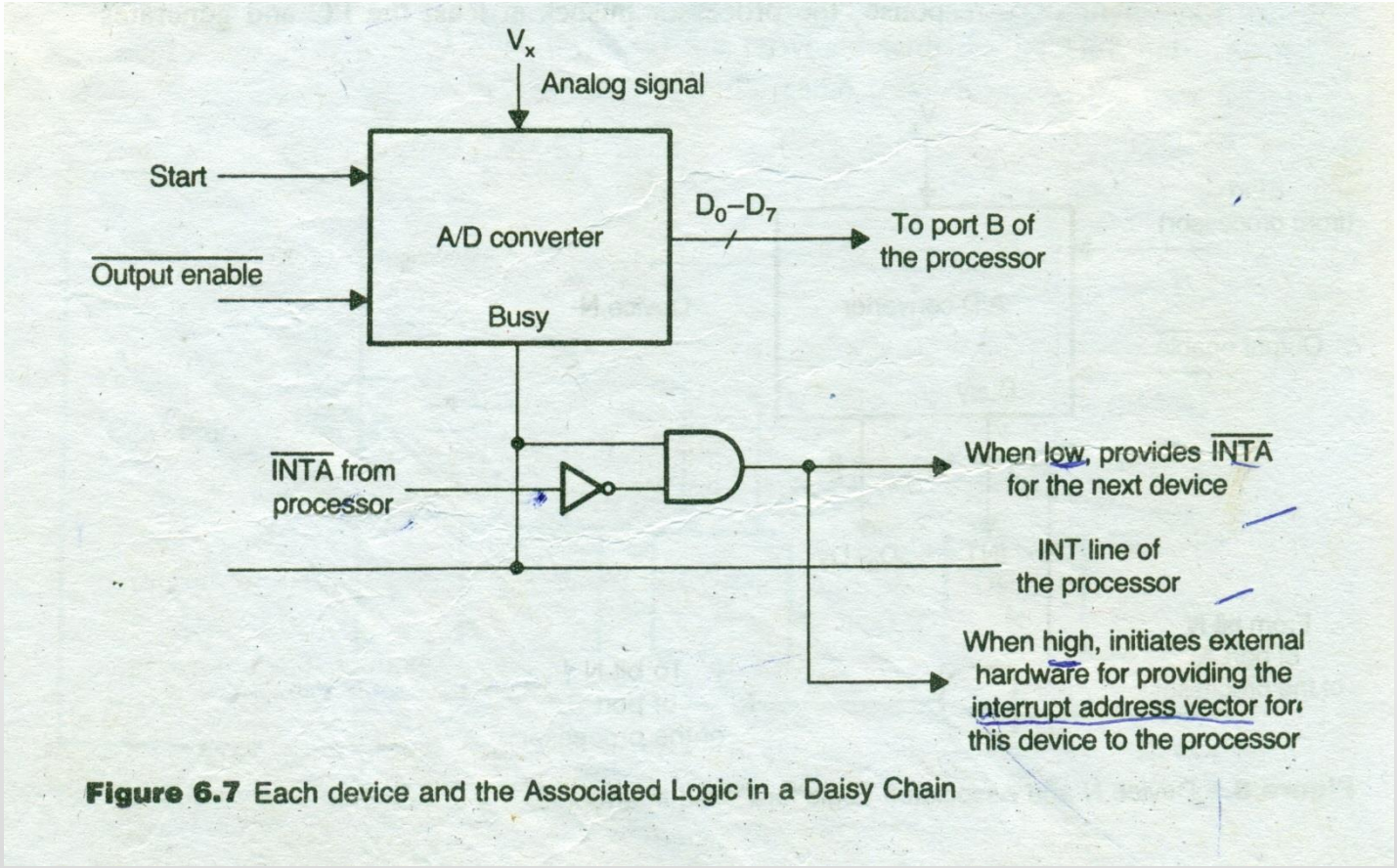


Figure 6.7 Each device and the Associated Logic in a Daisy Chain

3 basic types of DMA transfer

- Block transfer
- Cycle stealing
- Inter leaved DMA

Direct memory access-block transfer method