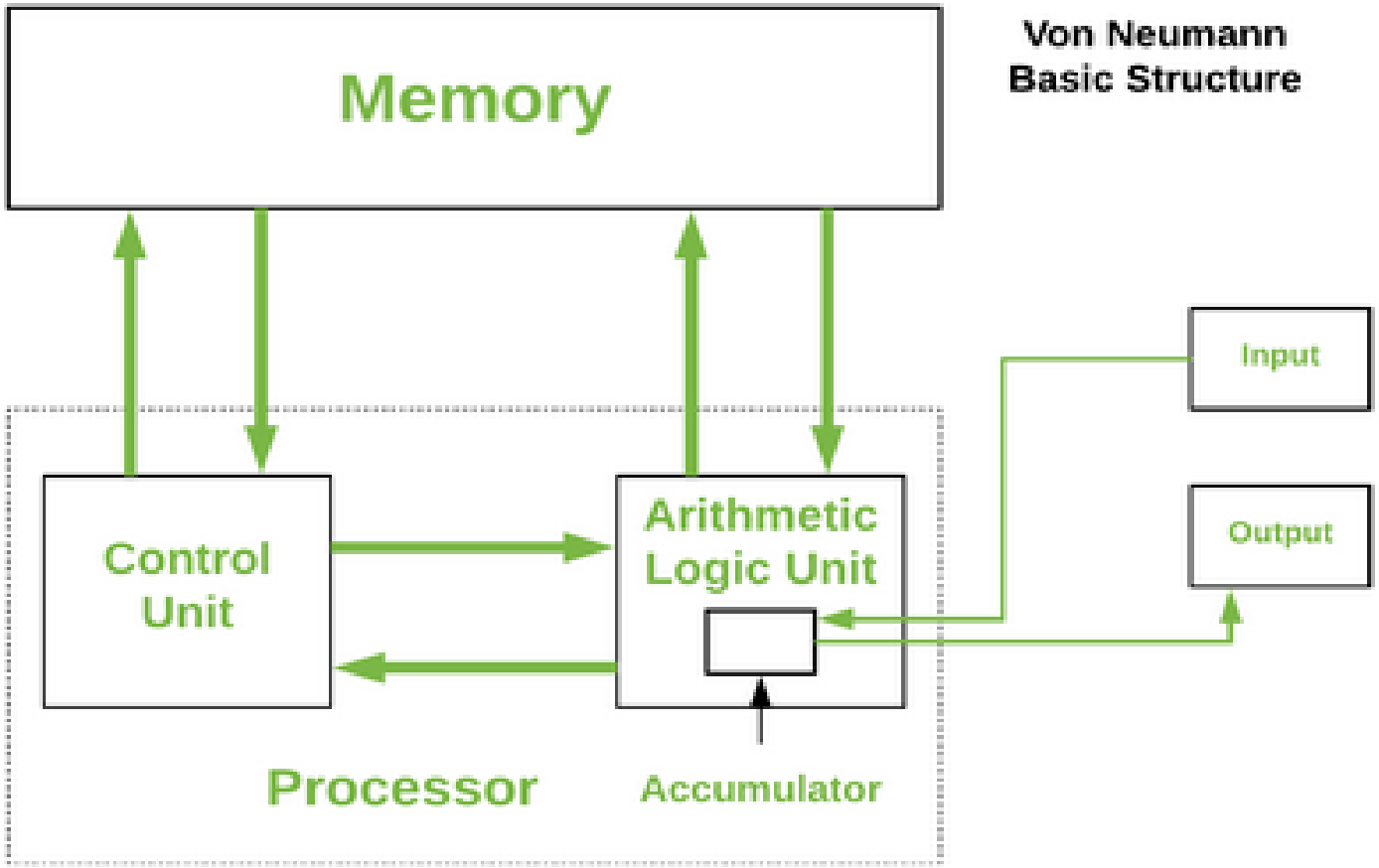
COMPUTER ORGANIZATION

MODERN COMPUTER ARCHITECTURE

BY

MOHAMED RAFIQUZZAMAN AND RAJAN CHANDRA


Memory
Von Neumann
Basic Structure
Input
Output
Control
Unit
Arithmetic
Logic Unit
Processor
Accumulator

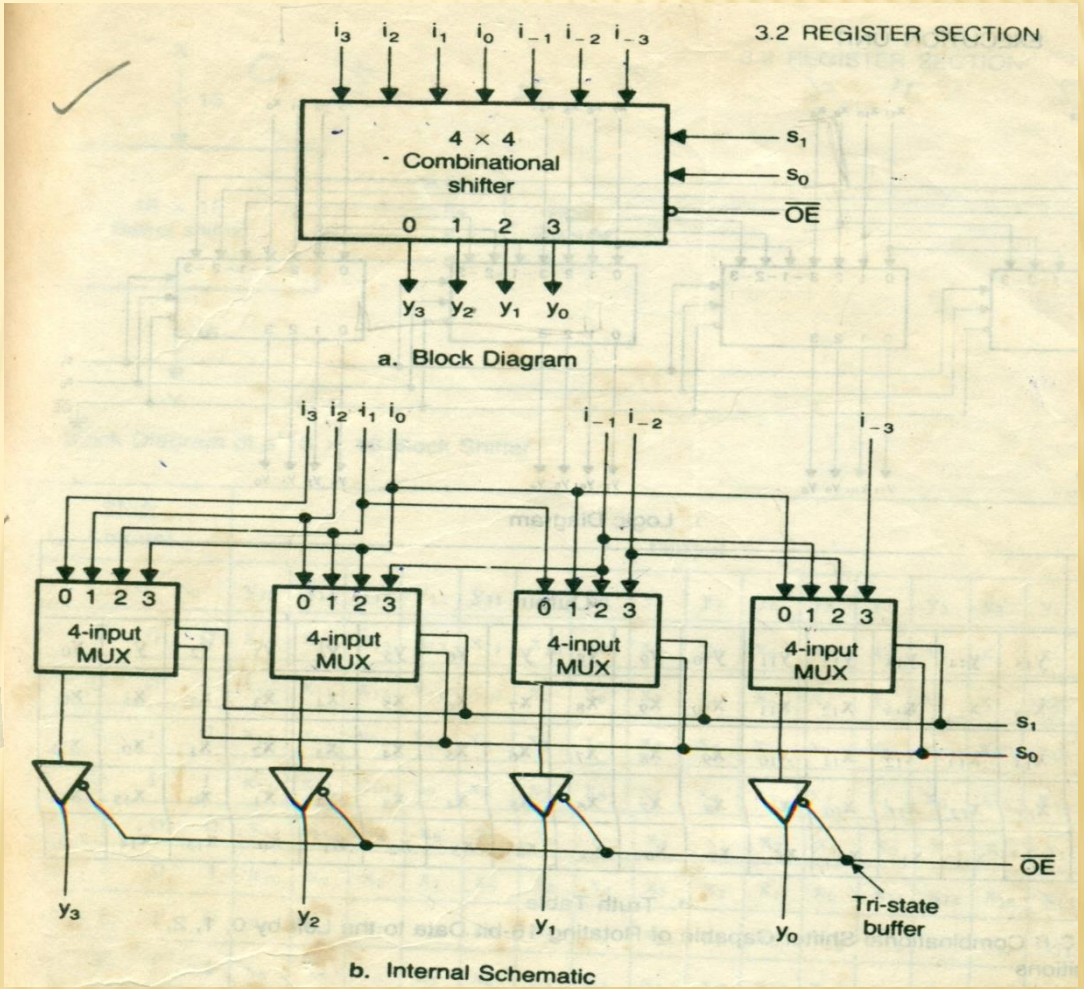
COMBINATIONAL SHIFTER DESIGN

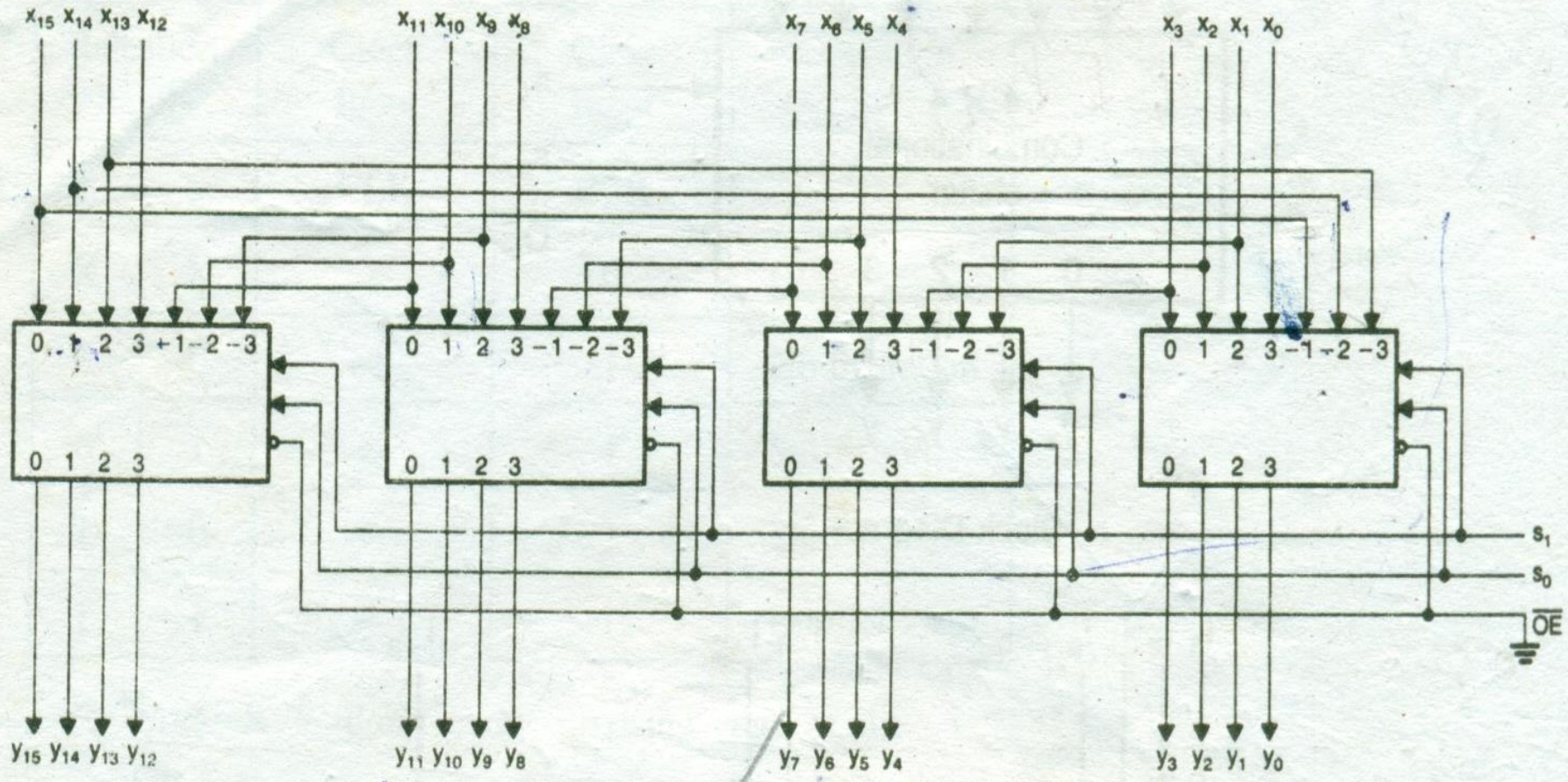
	Shift Count		Output				
$\overline{OE}$	s_1	s_0	y_3	y_2	y_1	y_0	Comment
1	X	X	Z	Z	Z	Z	Output lines float
0	0	0	i_3	i_2	i_1	i_0	Pass (no shift)
0	0	1	i_2	i_1	i_0	i_{-1}	Left shift once
0	1	0	i_1	i_0	i_{-1}	i_{-2}	Left shift twice
0	1	1	i_0	i_{-1}	i_{-2}	i_{-3}	Left shift three times

Note: Z—High-impedance state.
X—Don't care.

c. Truth Table

Figure 3.7 4 × 4 Combinational Shifter





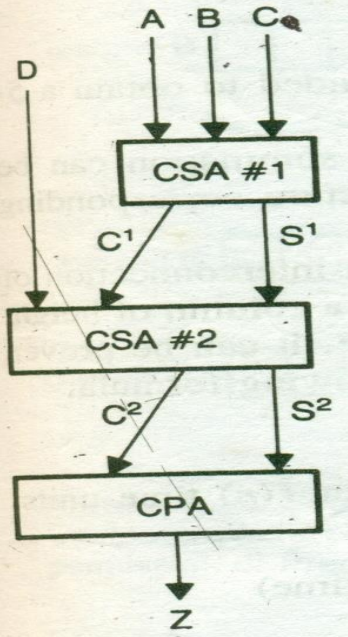
a. Logic Diagram

Shift Count		Output															
s_1	s_0	y_{15}	y_{14}	y_{13}	y_{12}	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	0	x_{15}	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
0	1	x_{14}	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	x_{15}
1	0	x_{13}	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	x_{15}	x_{14}
1	1	x_{12}	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	x_{15}	x_{14}	x_{13}

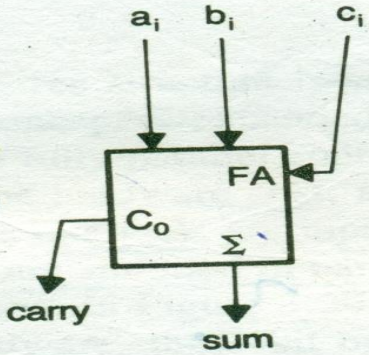
b. Truth Table

Figure 3.8 Combinational Shifter Capable of Rotating 16-bit Data to the Left by 0, 1, 2, or 3 Positions

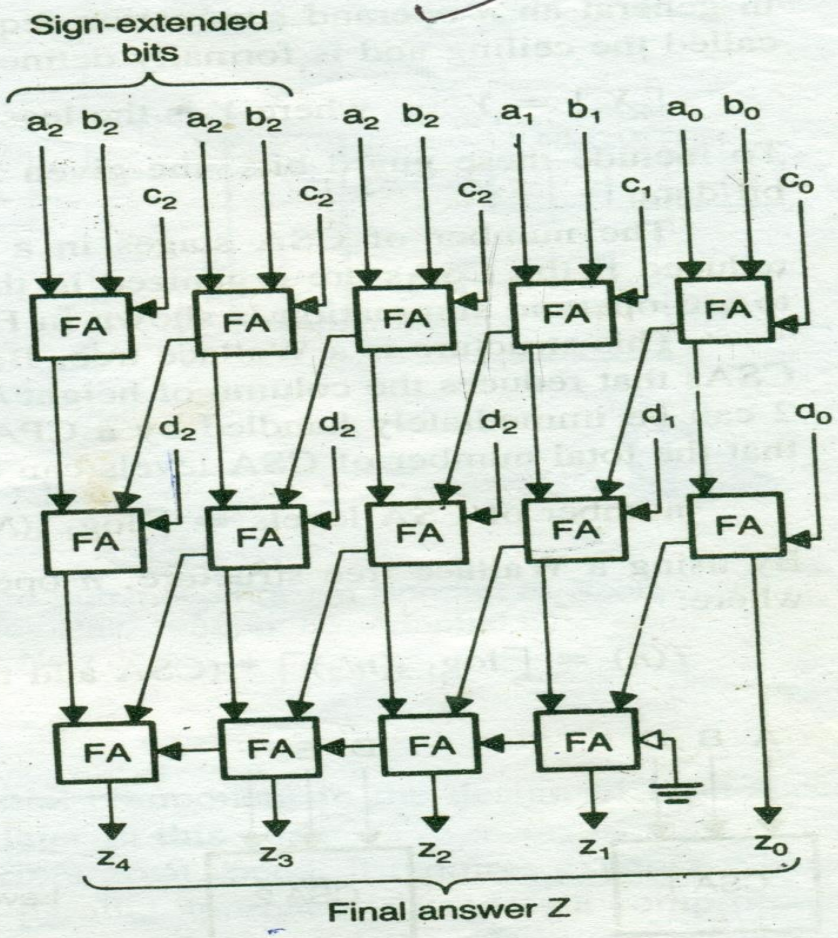
Carry Save Adder



a. Block Diagram



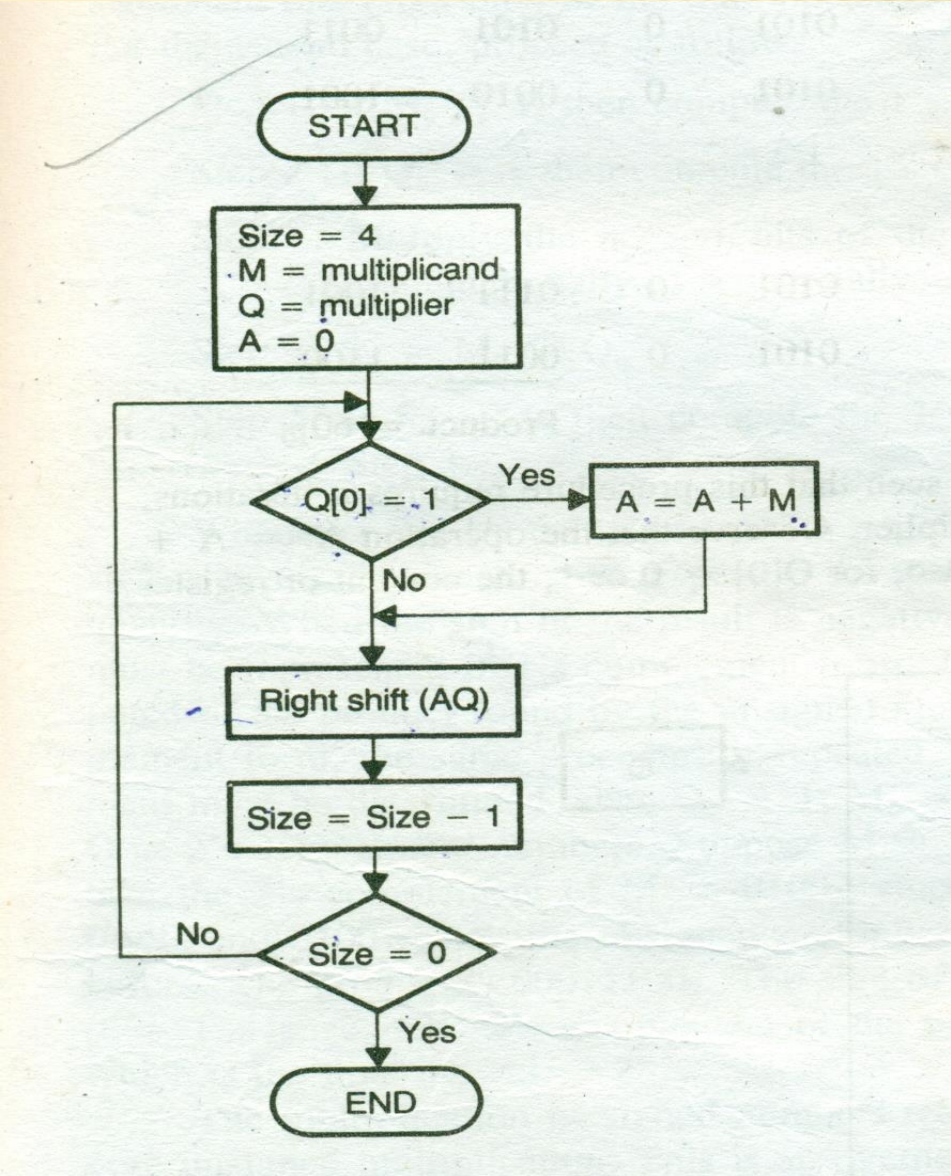
b. Basic Cell



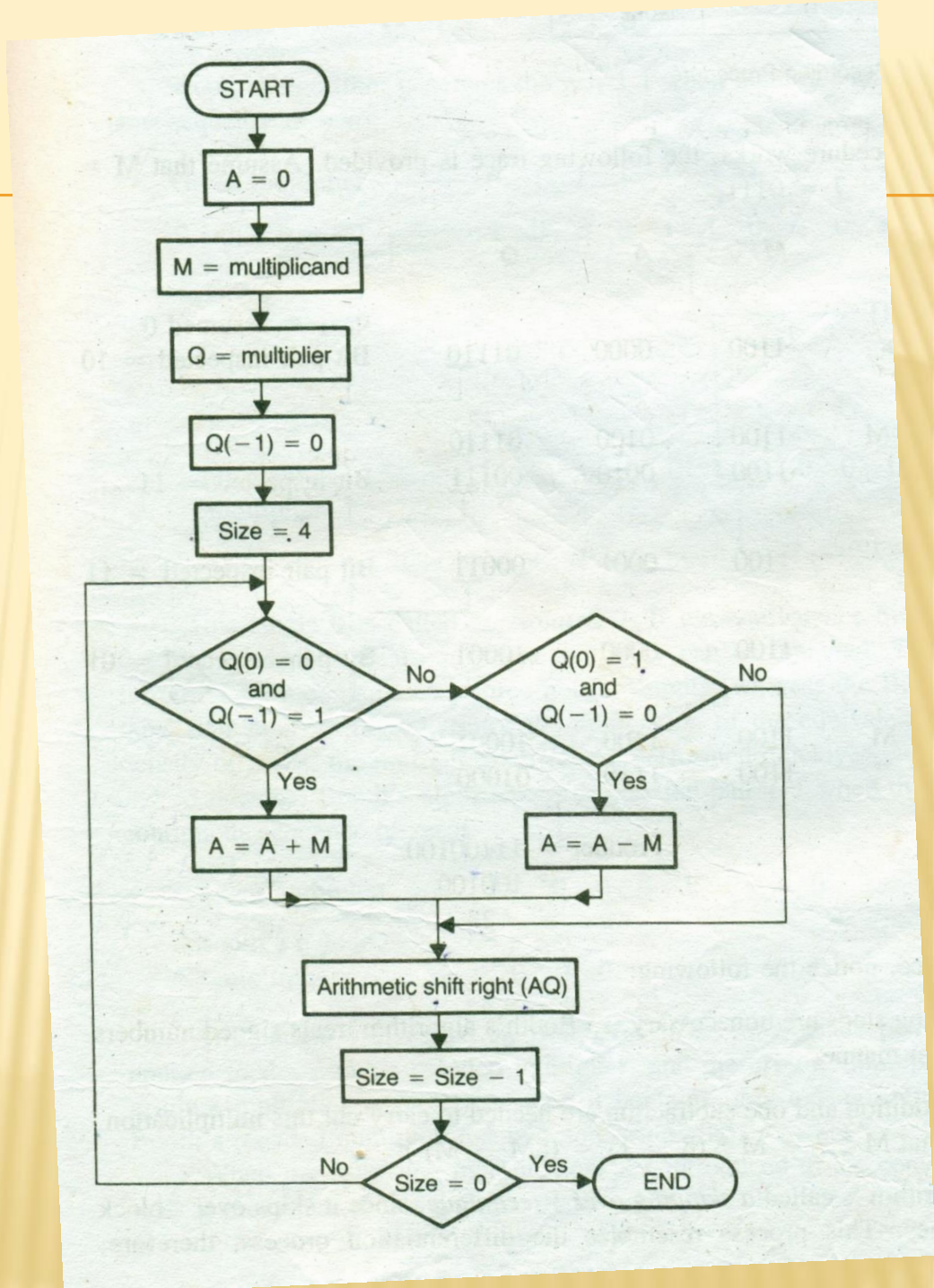
c. Hardware Schematic

Figure 3.18 5-operand Summation

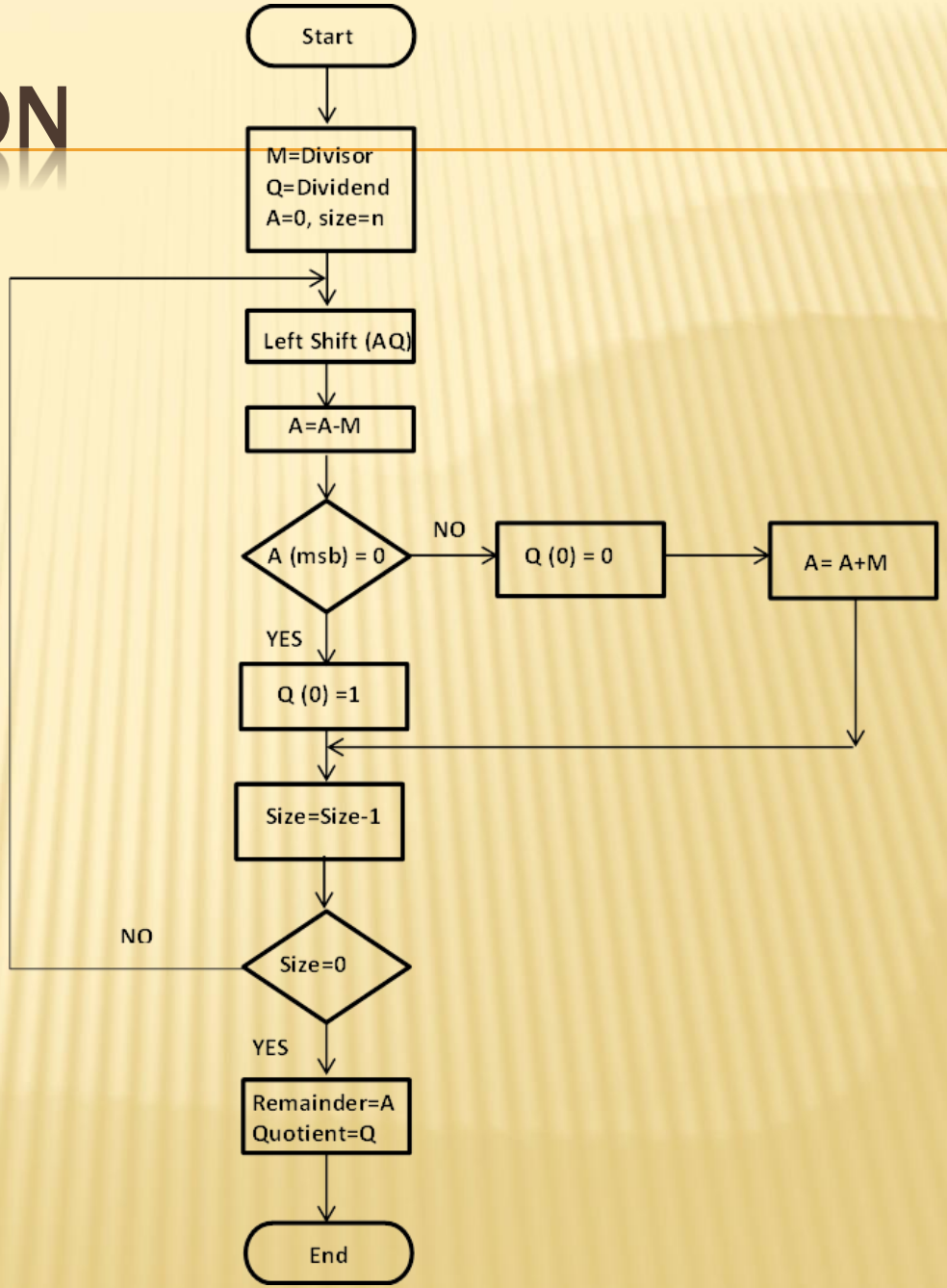
ADD AND SHIFT METHOD



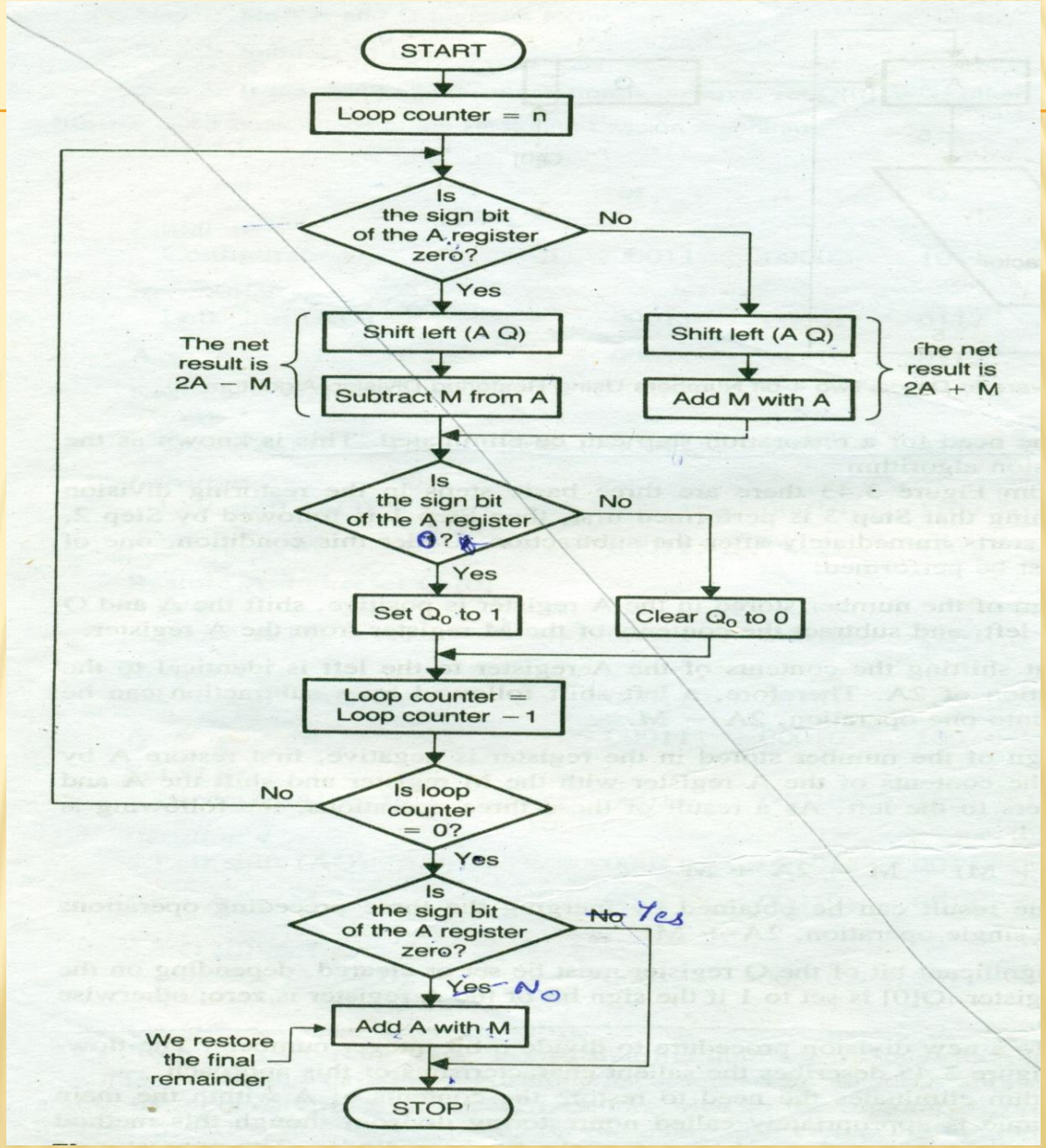
BOOTH'S ALGORITHM



RESTORING DIVISION



NONRESTORING DIVISION



CONTROL UNIT

- Bus Structure:
  - Single bus oriented RALU
  - Two bus oriented RALU
  - Three bus oriented RALU

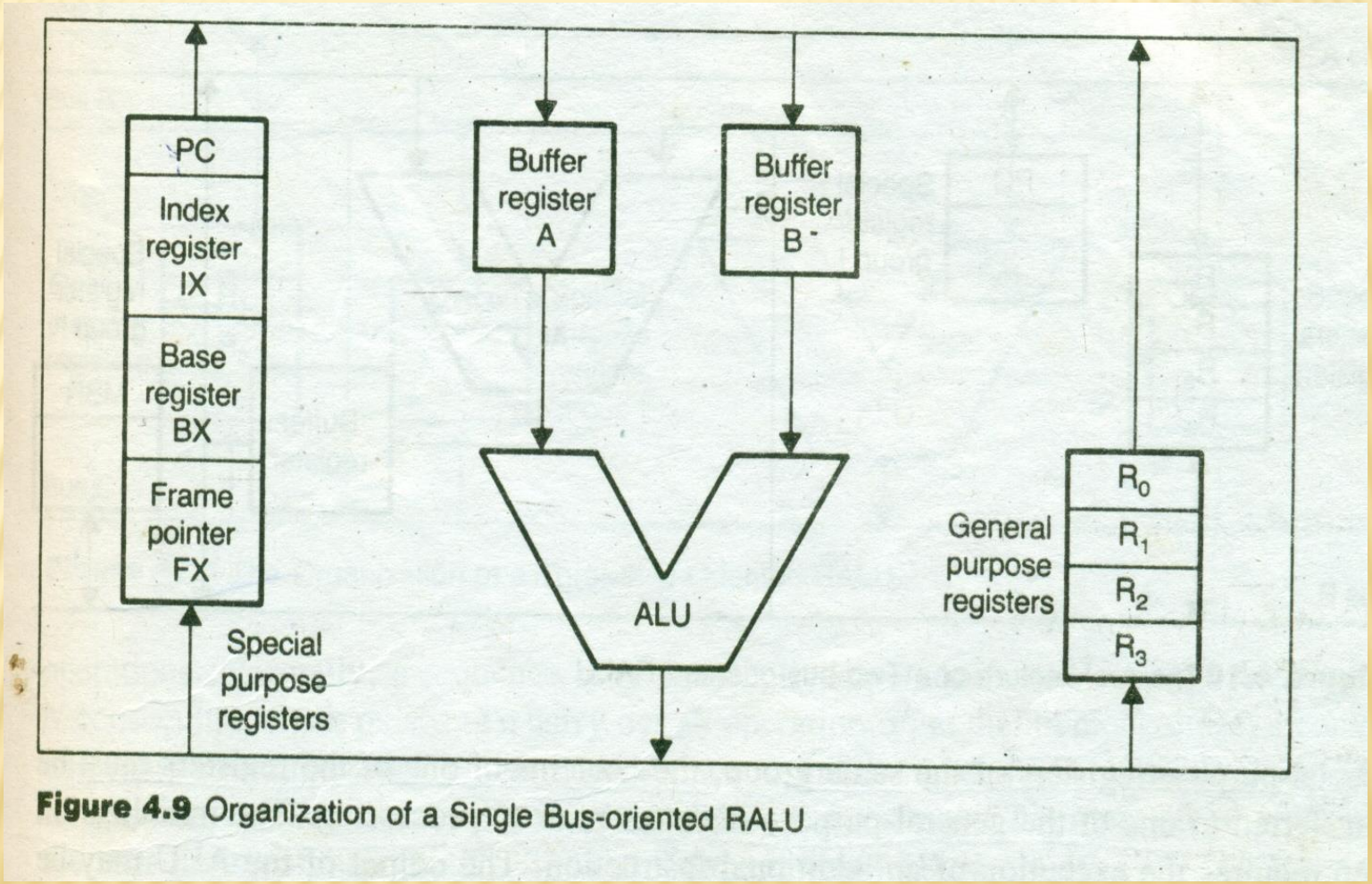


Figure 4.9 Organization of a Single Bus-oriented RALU

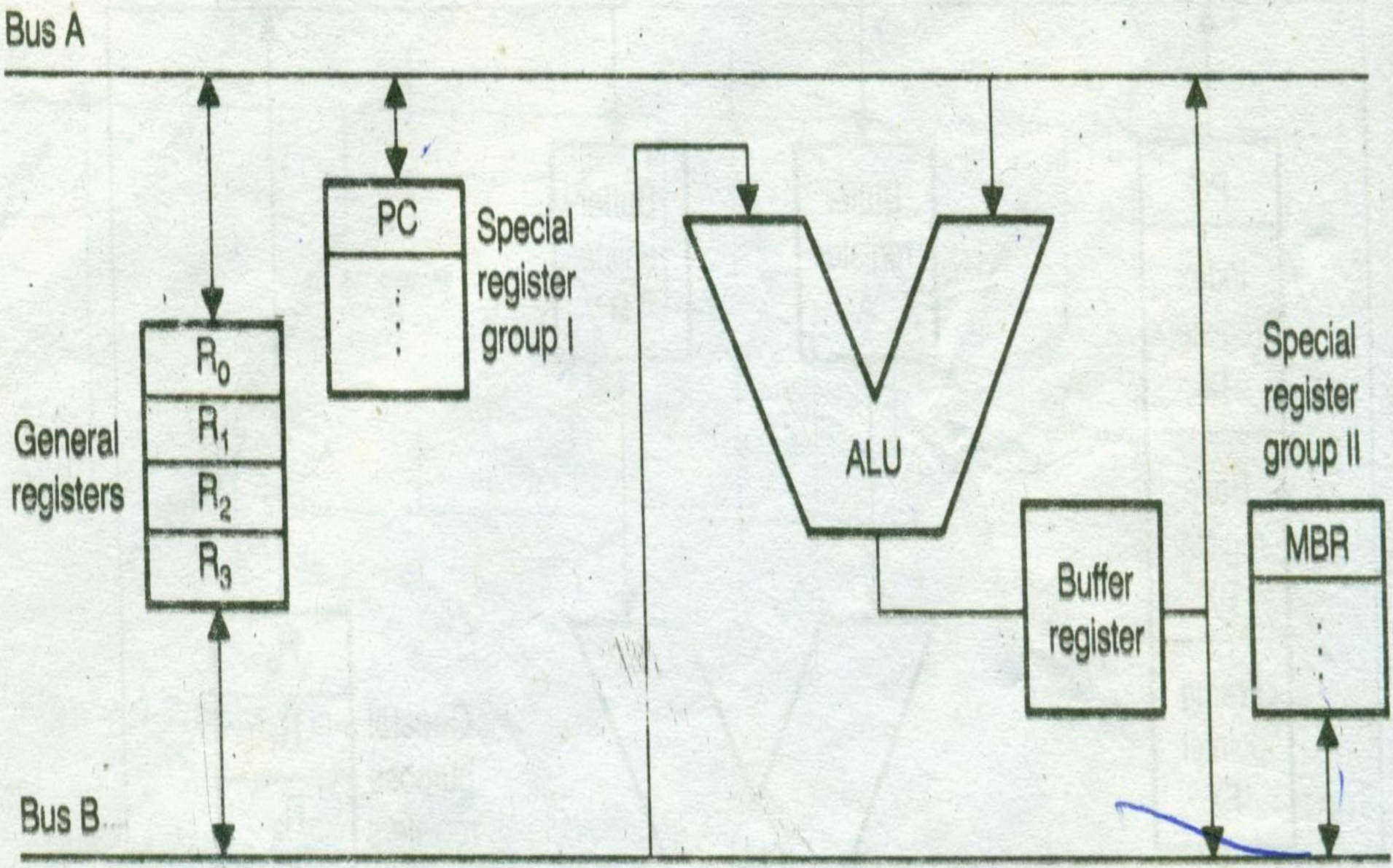


Figure 4.10 The Architecture of a Two-bus-oriented RALU

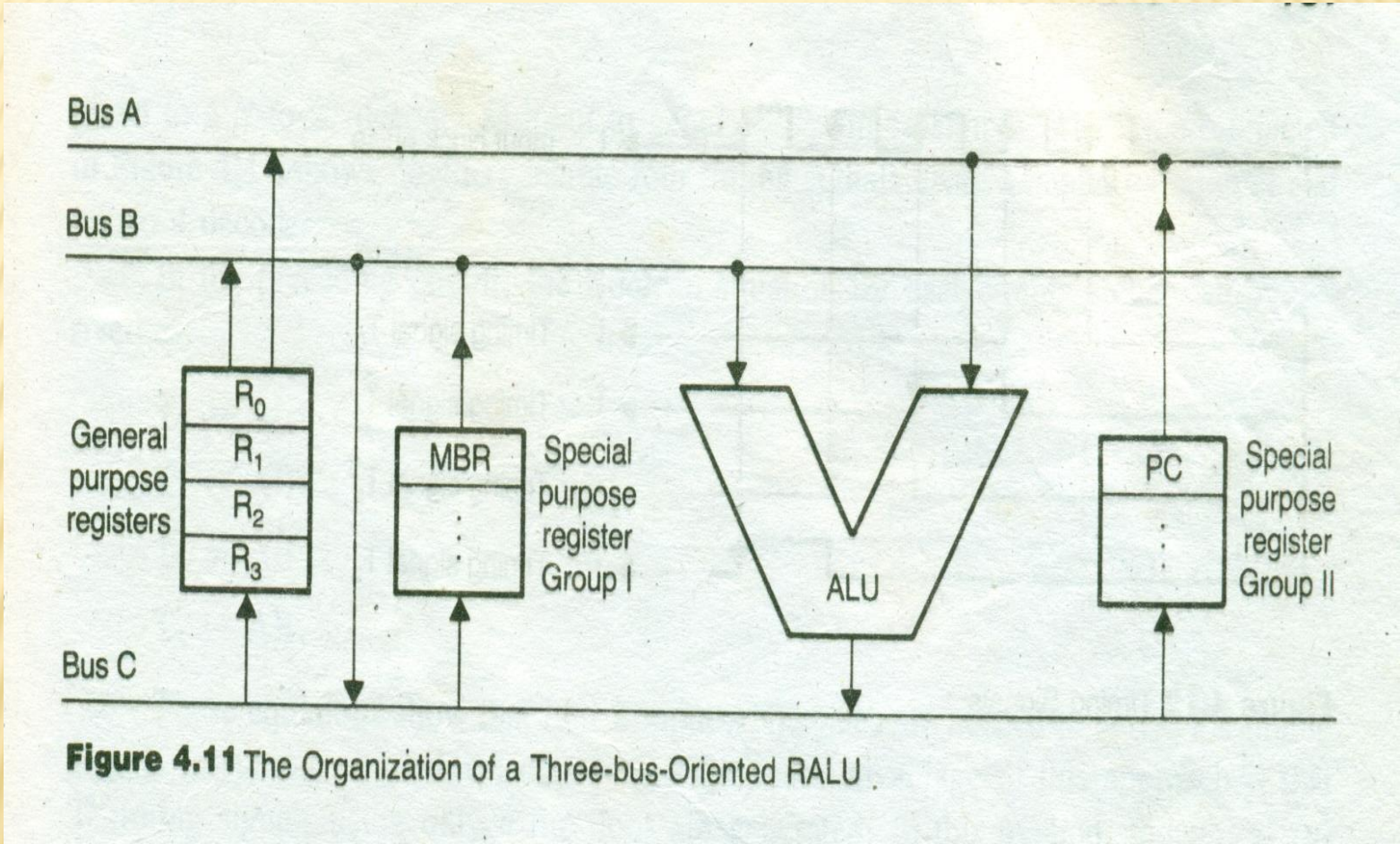


Figure 4.11 The Organization of a Three-bus-Oriented RALU

Design Methods

Control units are designed in two different ways:

- Hardwired approach
- Microprogramming

The first approach exists because control logic is a clocked sequential circuit and, therefore, conventional sequential circuit design procedures can be applied to build a control unit. This technique earns the name *hardwired approuch*, because the final circuit is obtained by physically connecting typical components such as gates and flip-flops. In the microprogrammed approach, ail control functions that can be simultaneously activated are grouped to form control words stored in a separate memory called the control memory. The control words are fetched from the control memory, and the individual control fields are routed to various functional units to enable appropriate gates. When these gates are activated sequentially, the desired task is performed.

Hardwired Approach

1. Define the task to be performed.
2. Propose a trial processing section.
3. Provide a register-transfer description of the algorithm based on the processing section outlined in the previous step.
4. Validate the algorithm by 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 a 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 a Booth's multiplier to multiply two 4 – bit 2's complement numbers.

STEP 2

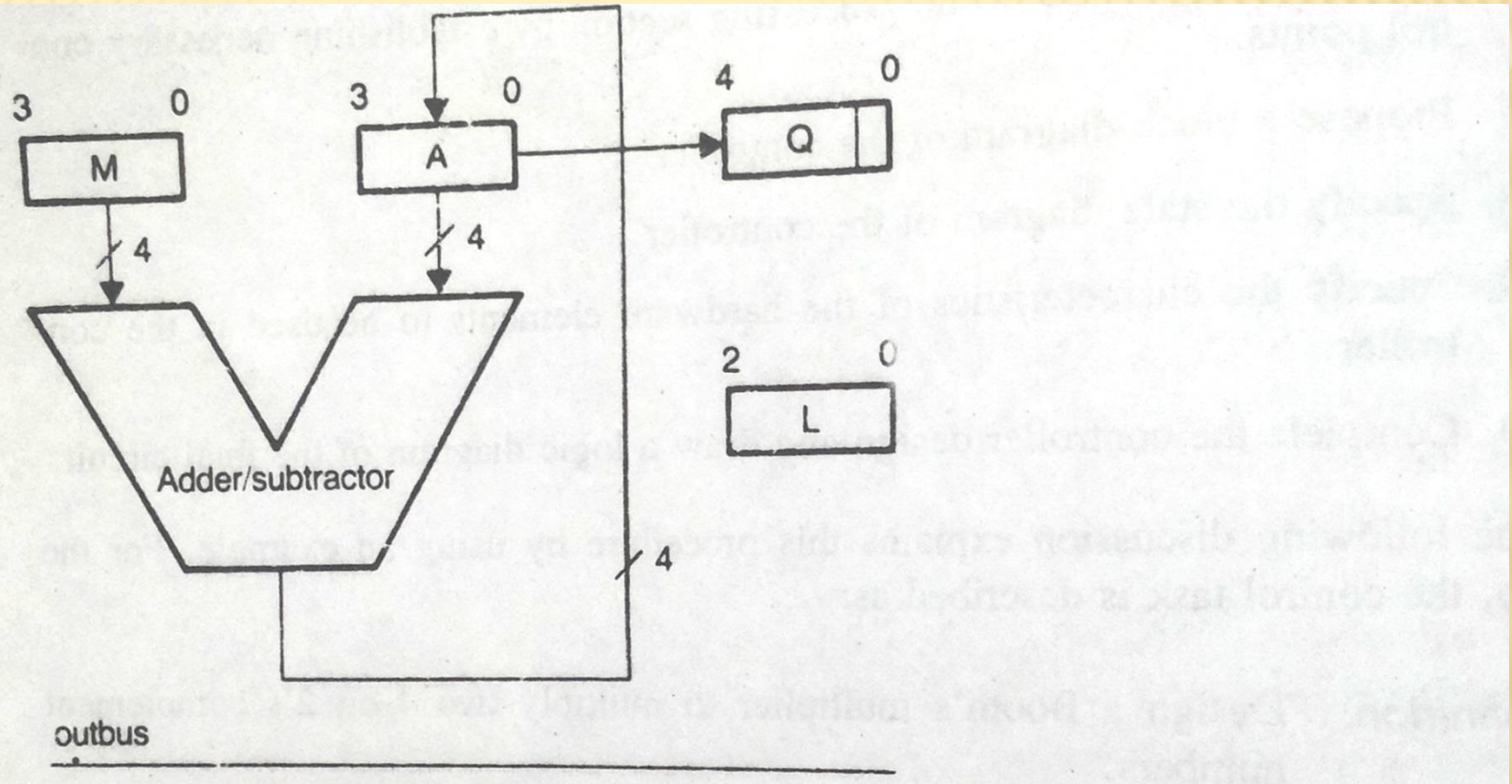


Figure 4.15 Organization of the Processing Section for Performing 4 × 4 Multiplication Using Booth's Algorithm

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

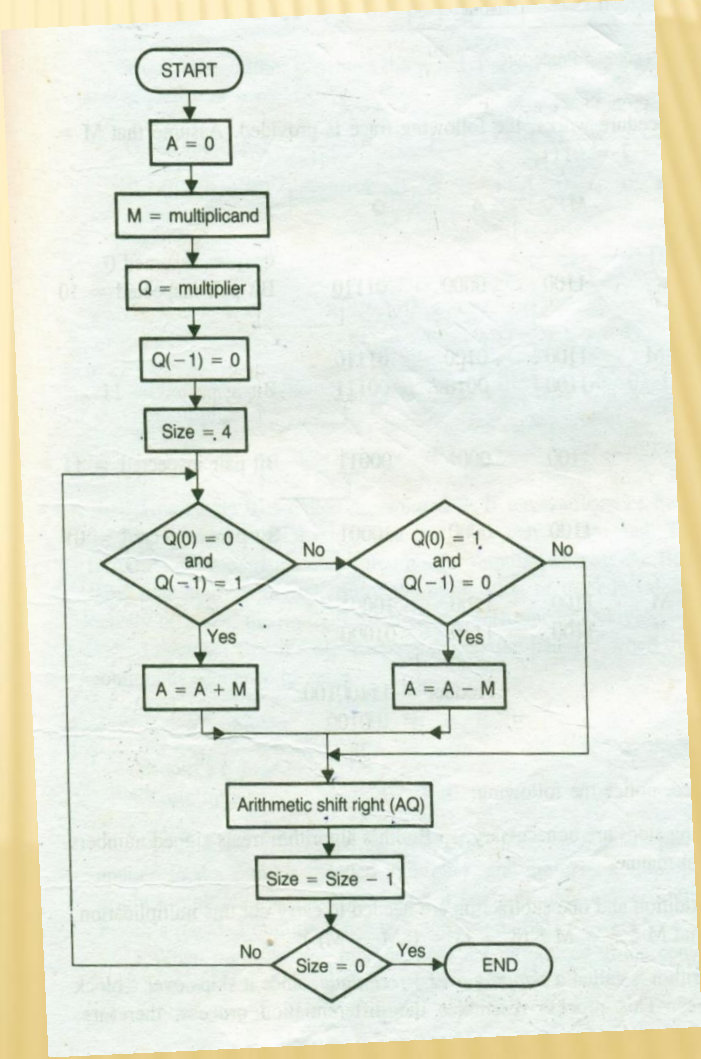
Rshift: ASR(AQ), L←L-1; C6, C7

If L<>0 then goto loop

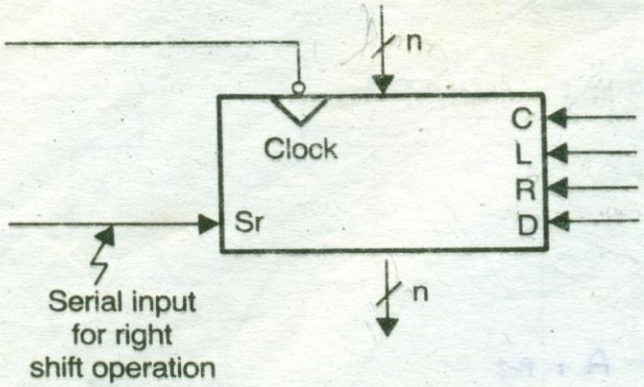
Outbus=A; C8

Outbus=Q[4:1]; C9

Halt: go to Halt;

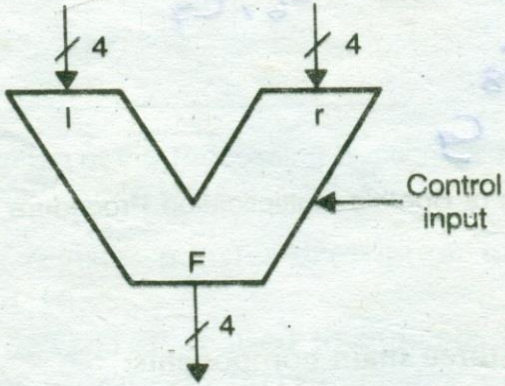


STEP 5



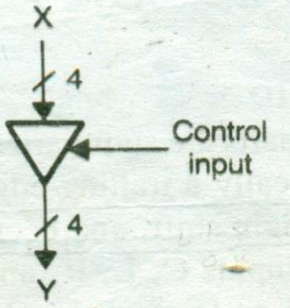
C	L	R	D	Clock	Action
1	0	0	0	↓	Clear
0	1	0	0	↓	Load external data
0	0	1	0	↓	Right shift
0	0	0	1	↓	Decrement by one
0	0	0	0	↓	No change

a. Storage Register



Control input	F
1	I + r
0	I − r

b. Adder/subtractor



Control input	Y
1	X
0	High Z

c. Tri-state Buffer

STEP 6

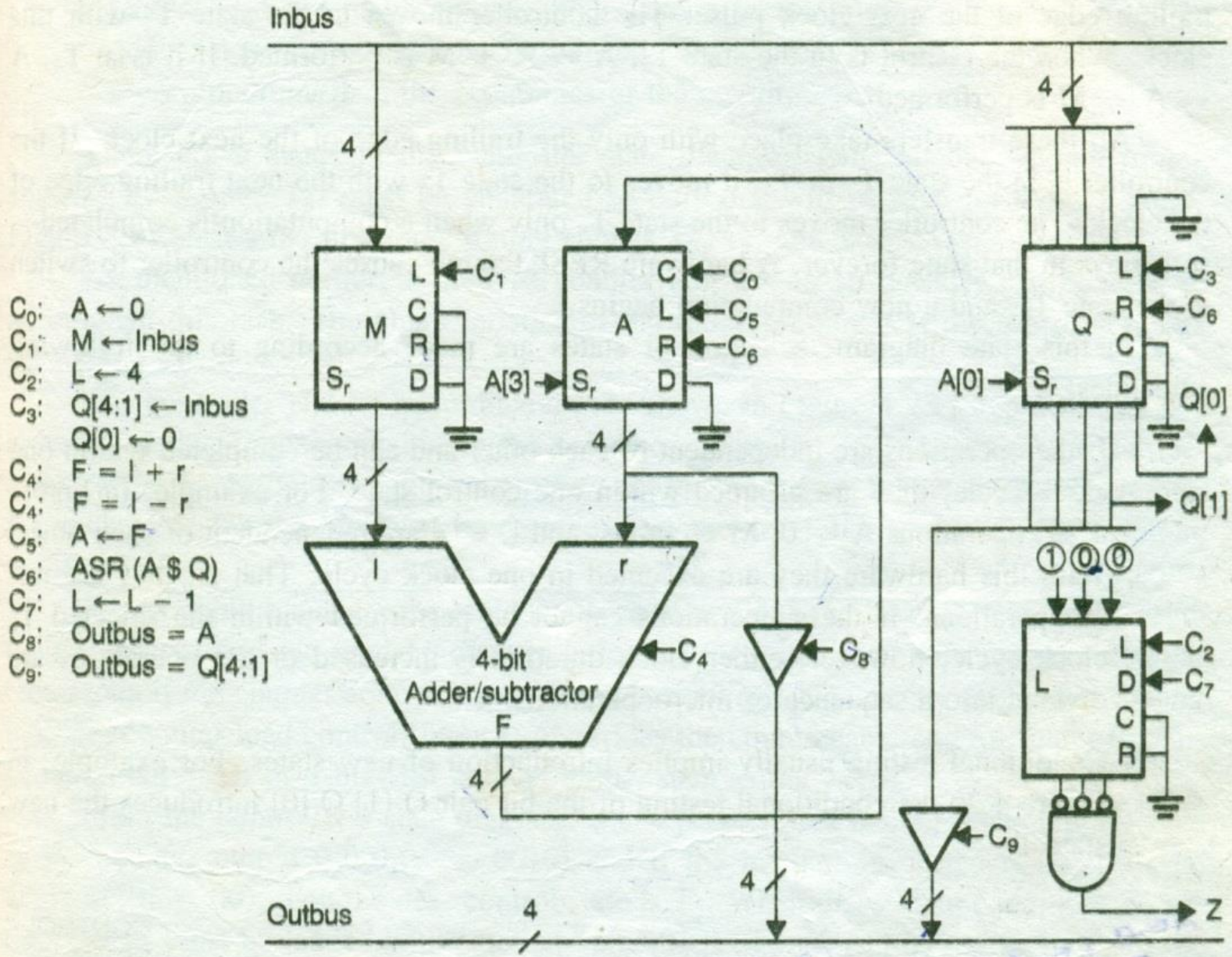


Figure 4.18 Processing Section of the Booth's Multiplier

STEP 7

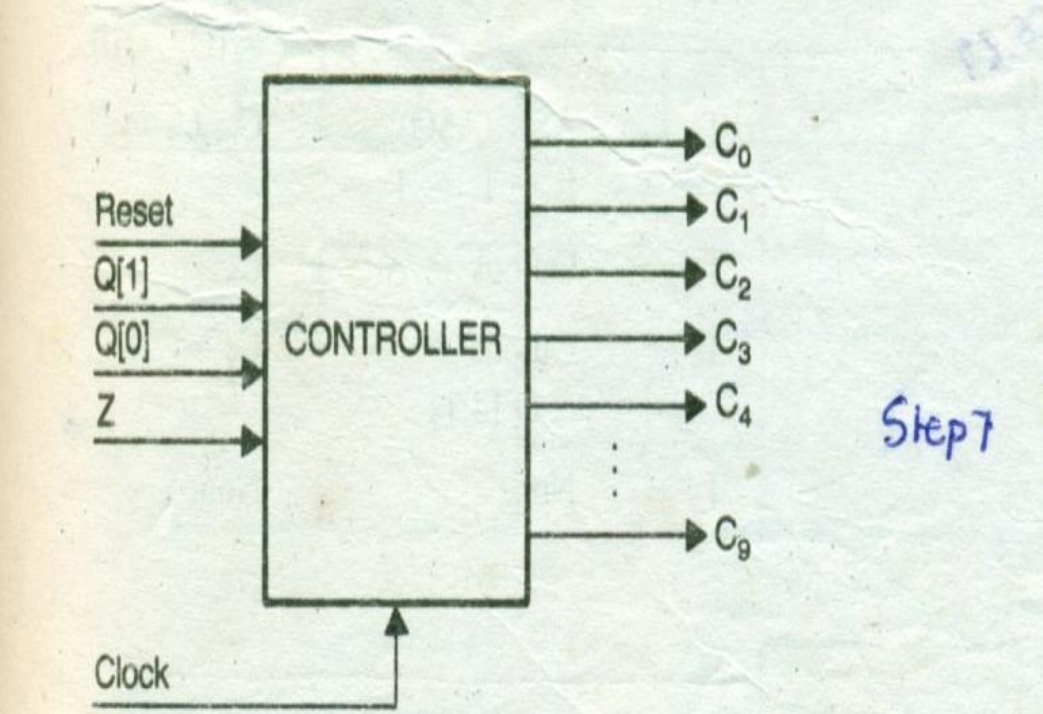
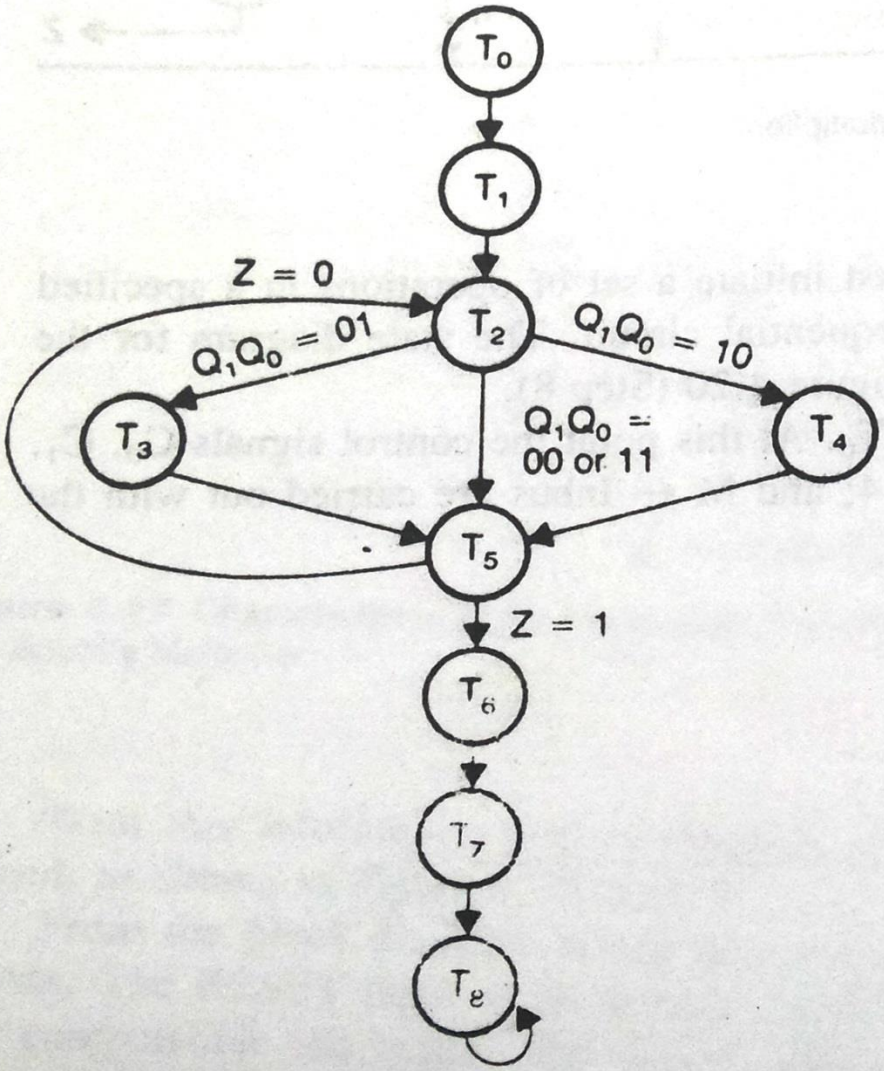


Figure 4.19 Block-diagram of the Booth's Multiplier Controller

STEP 8



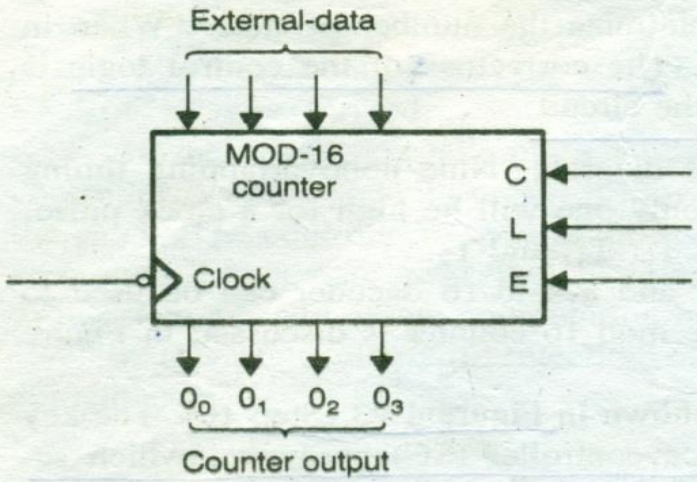
a. State Diagram

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

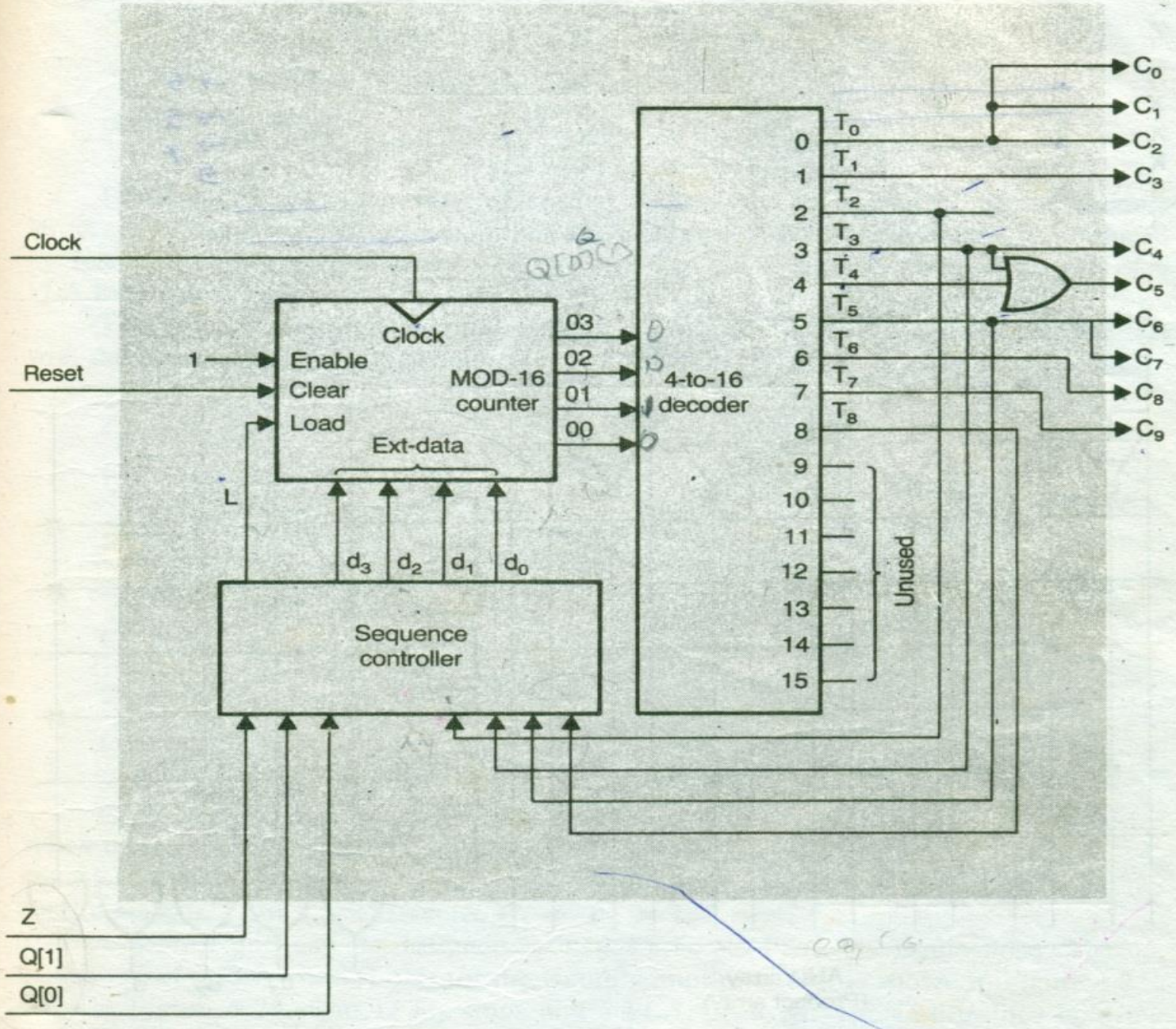
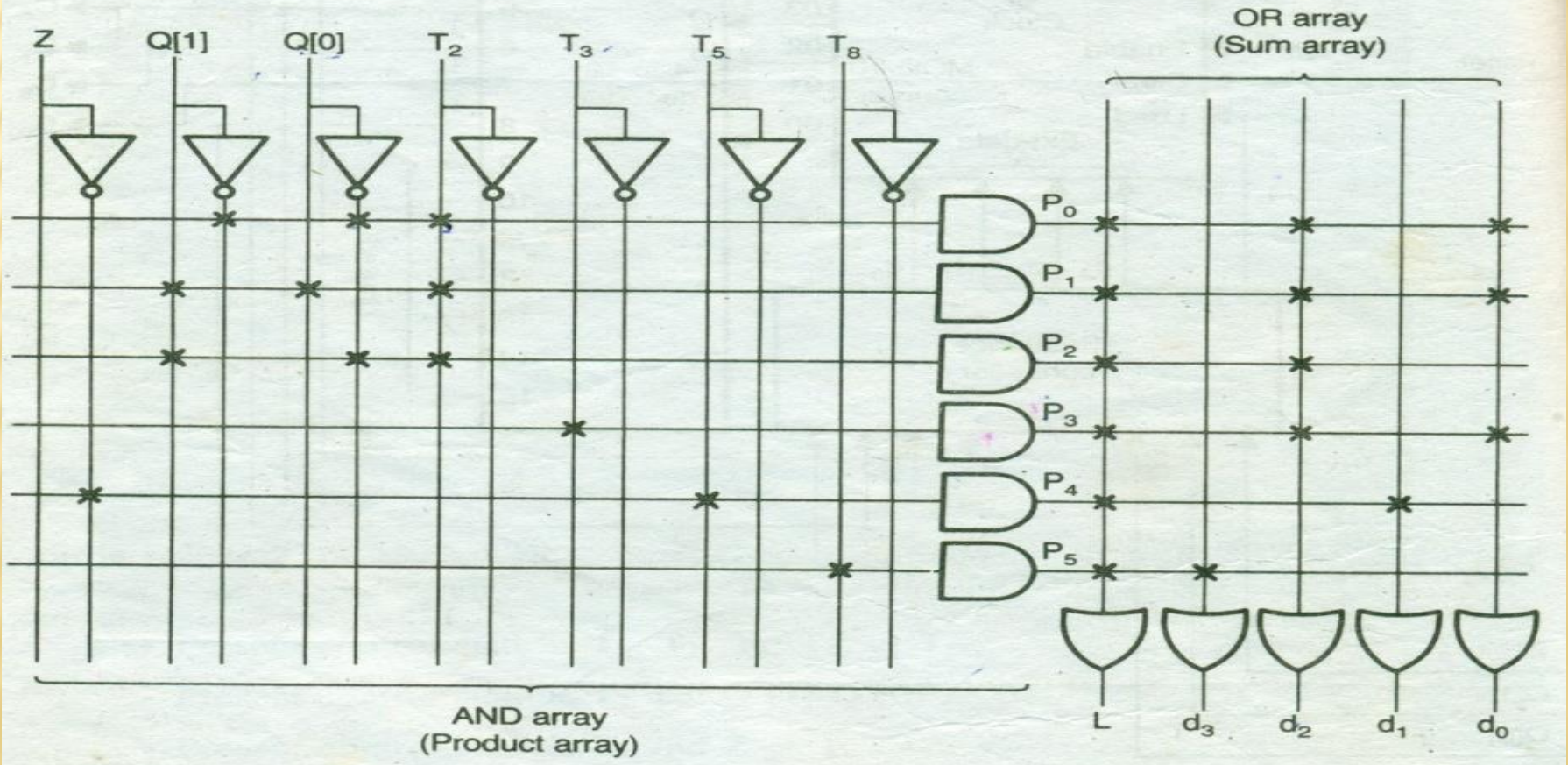


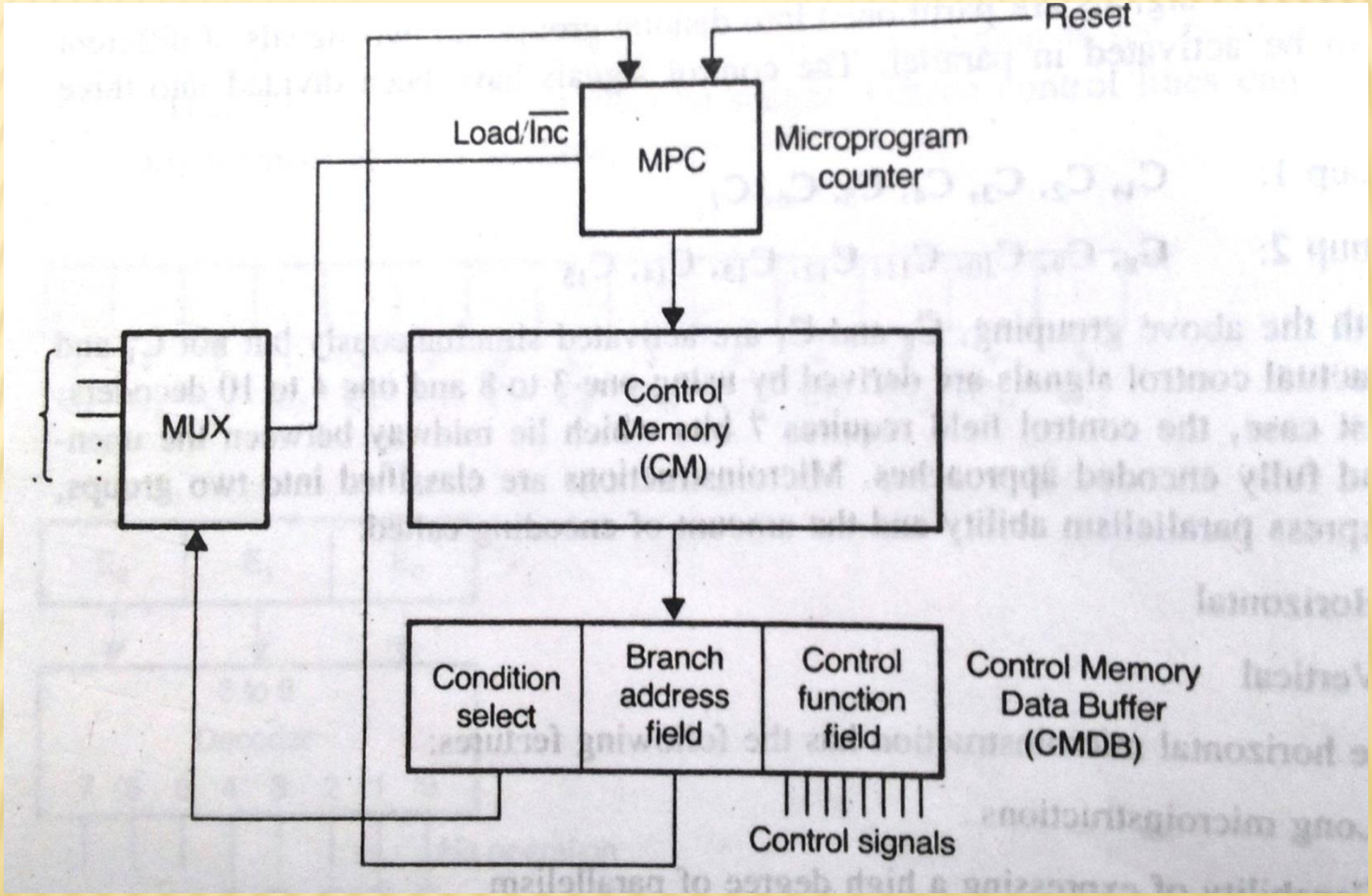
Figure 4.23 Logic Diagram of the Booth's Multiplier Controller

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



MICROPROGRAMMED CONTROL UNIT



Control Memory Address		Declare register A[4], M[4],Q[5], L[3]; Declare buses Inbus[4], Outbus[4];
0 0000	co,c1,c2	Start: A←0, M←Inbus, L←4;
1 0001	c3	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	c4,c5	Add: A←A+M;
6 0110		Go to Rshift
7 0111	c4',c5	Sub: A←A-M;
8 1000	c6,c7	Rshift: ASR(AQ), L←L-1;
9 1001		If L<>0 then Goto loop
101010	c8	Outbus=A;
111011	c9	Outbus=Q[4:1];
121100		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

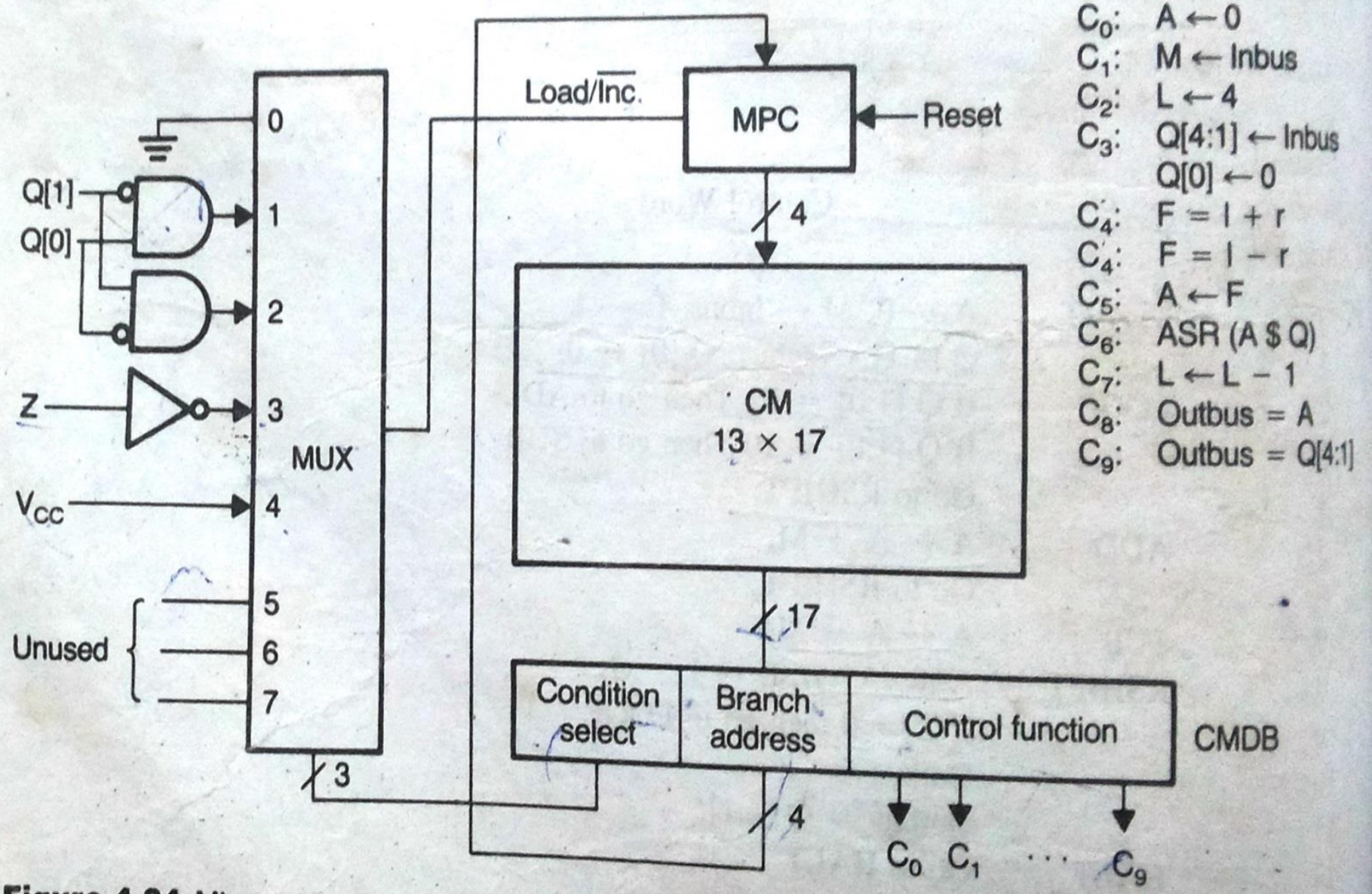


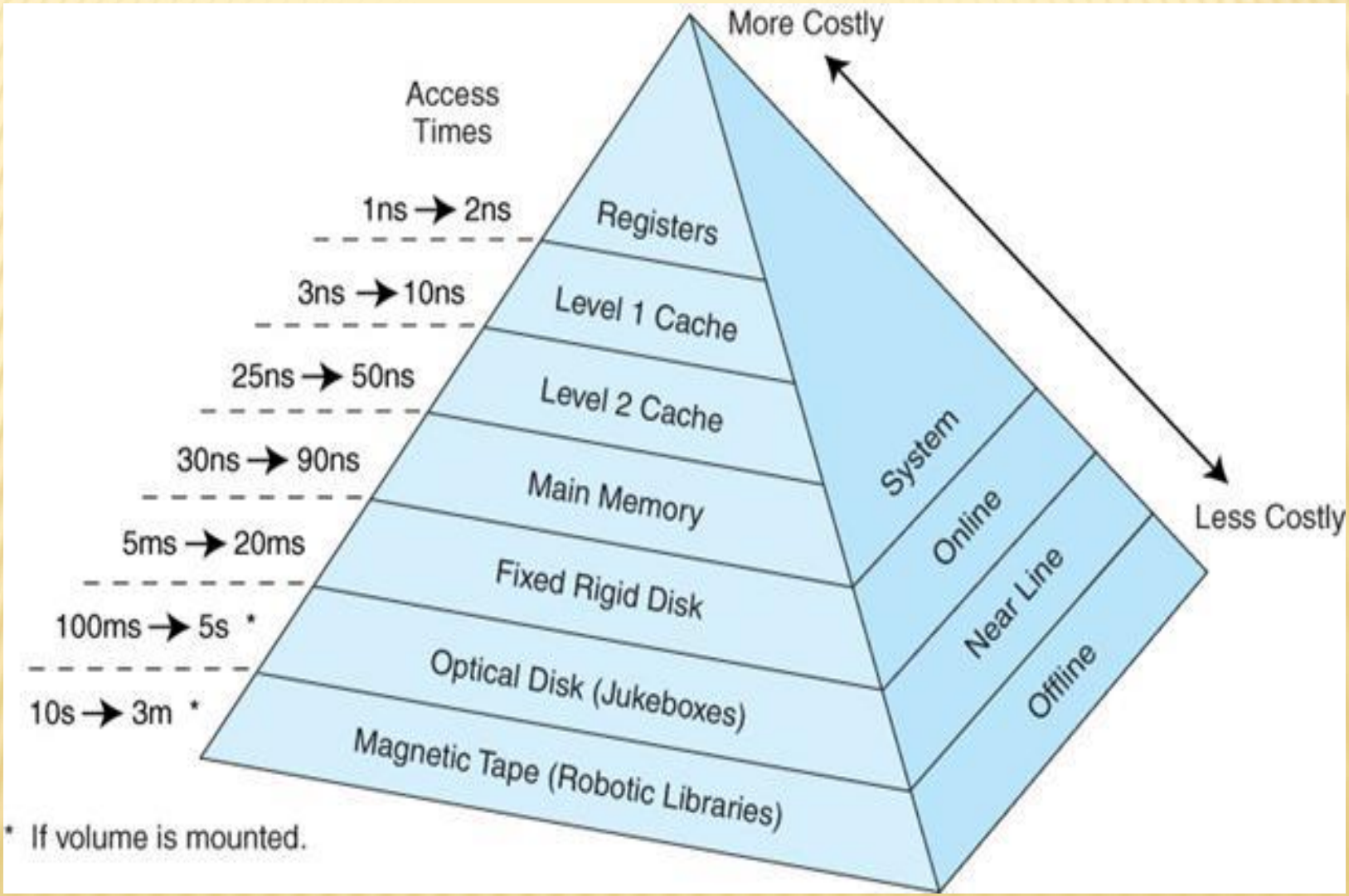
Figure 4.34 Microprogrammed Multiplier Control Unit

ROM Address																					
In decimal	In binary				Cond. Sel			Branch Addr.				Control Word									
												Control Function									
					c_{s2}	c_{s1}	c_{s0}	b_{r3}	b_{r2}	b_{r1}	b_{r0}	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	Comments
0	0	0	0	0	0	0	0	0	0	0	0	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	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	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	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	go to 8 (RSHIFT)
5	0	1	0	1	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	go to 8 (RSHIFT)
7	0	1	1	1	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	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	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	1	0	Outbus = A
11	1	0	1	1	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	1	0	0	0	0	0	0	Go to 12 (HALT)

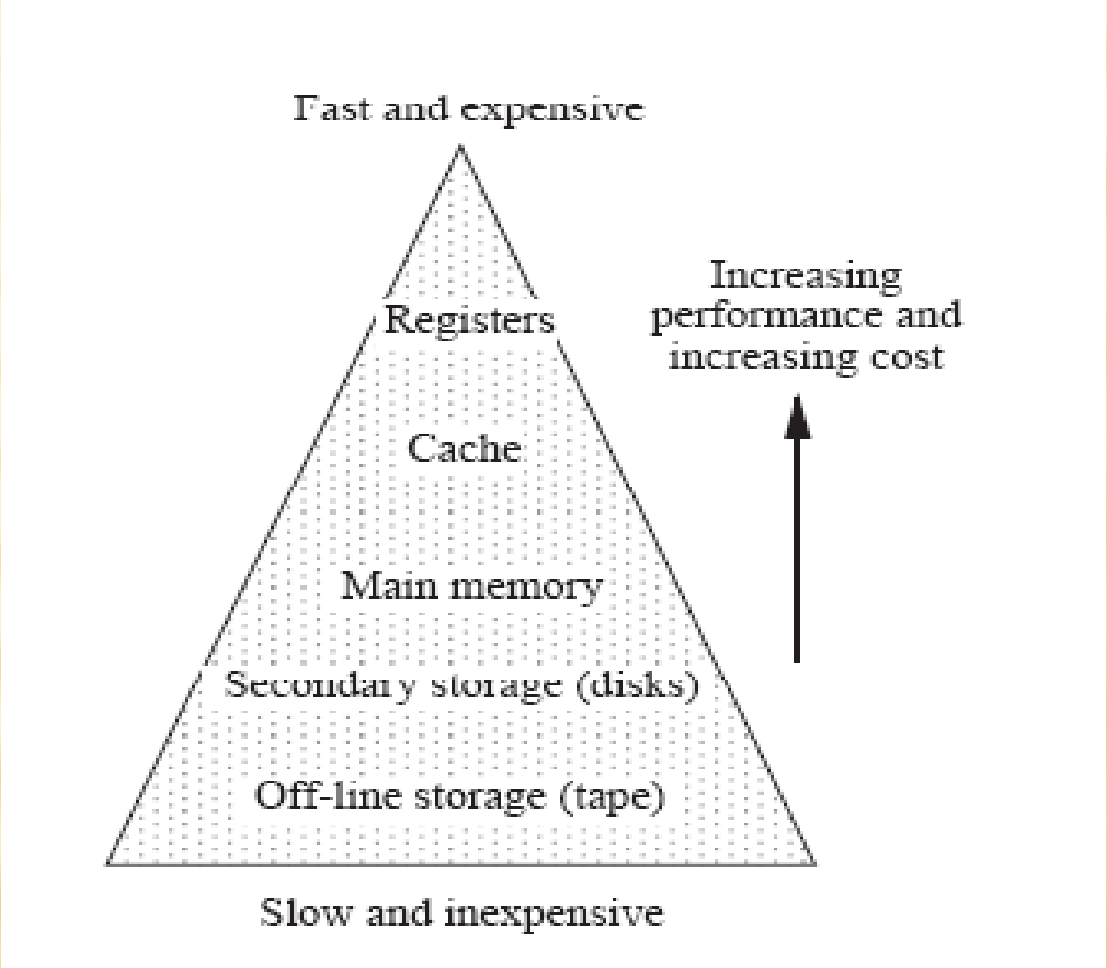
ROM Address					Control Word																				
In decimal	In binary				Cond. Sel			Branch Addr.				Control Function											Comments
					c_{s2}	c_{s1}	c_{s0}	b_{r3}	b_{r2}	b_{r1}	b_{r0}	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	A ← 0, M ← Inbus, L ← 4
1	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 (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)

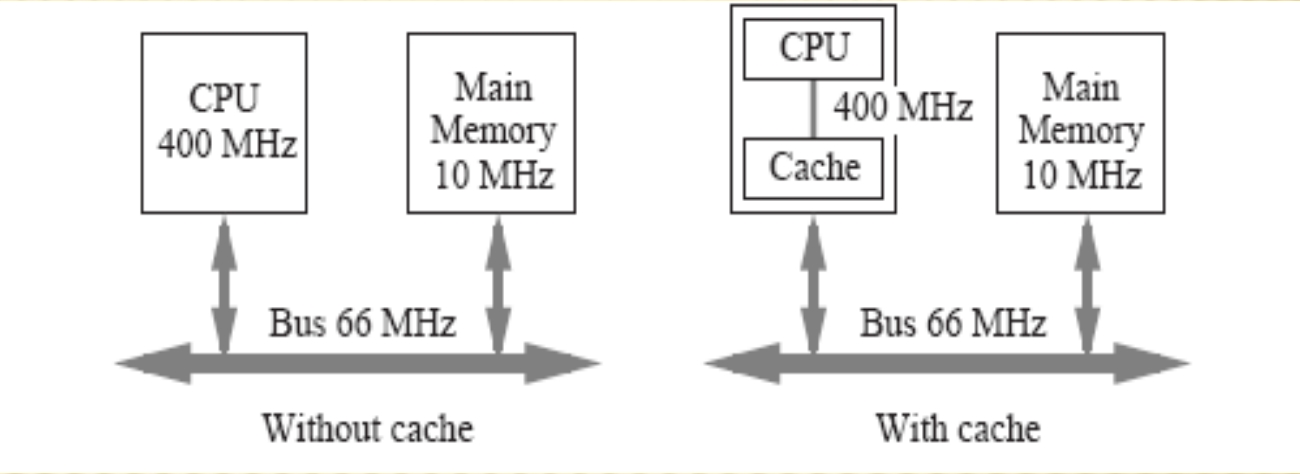
Figure 4.35 Binary Listing of the Microprogram For the 4 x 4 Booth's Multiplier

MEMORY HIERARCHY

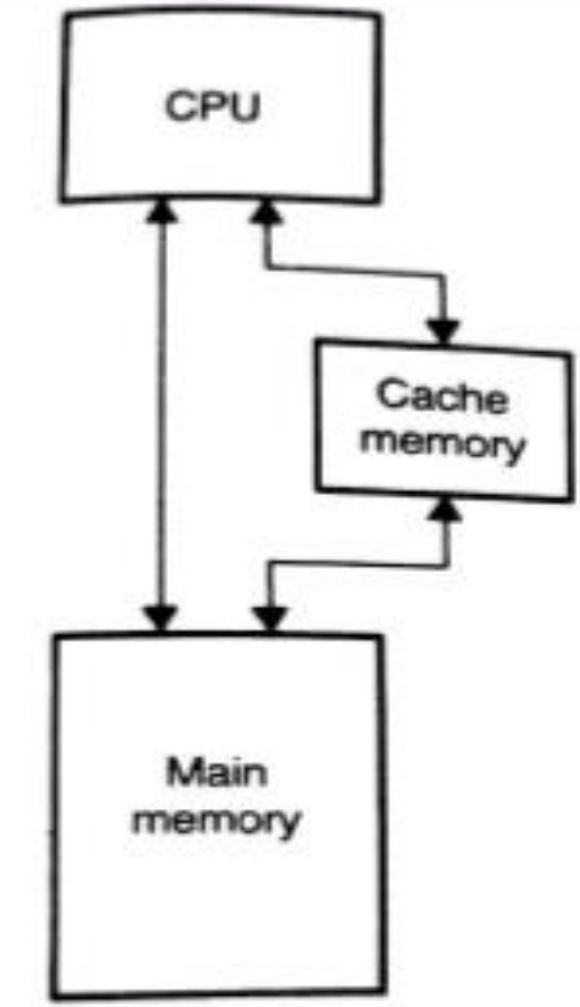


MEMORY HIERARCHY




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 MEMORY

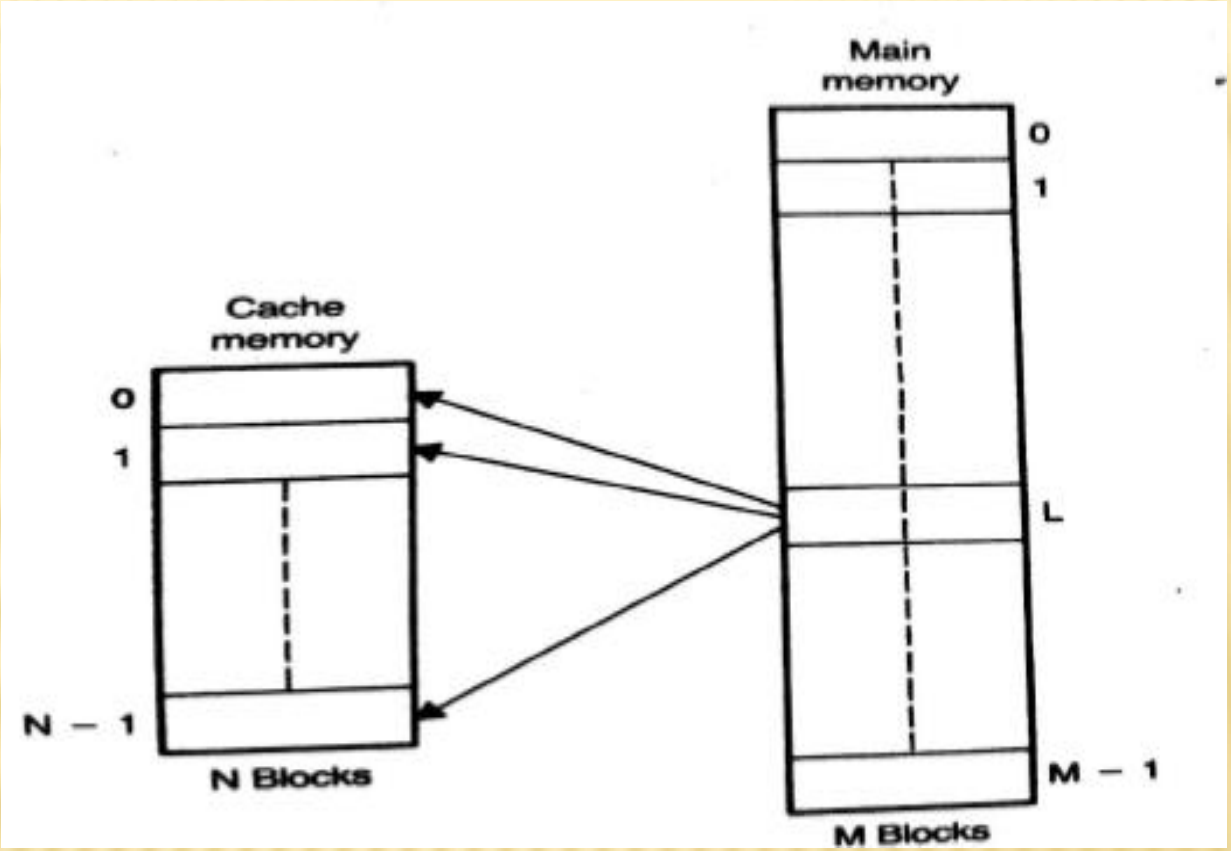


CACHE MAPPING

Commonly used methods:

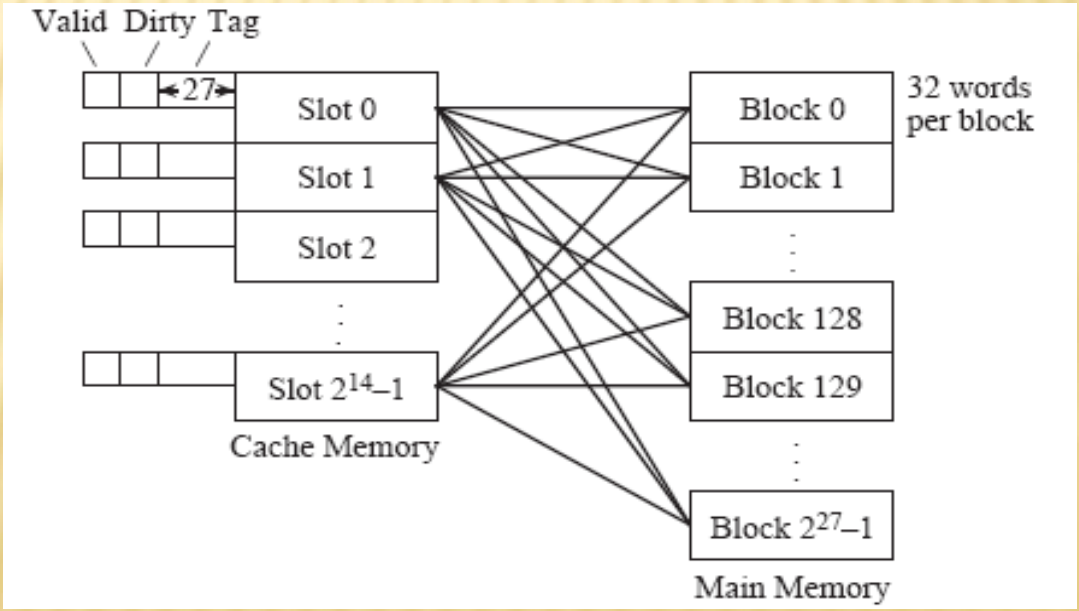
- Fully Associative Mapping
- Direct-Mapping
- Set-Associative Mapping

FULLY ASSOCIATIVE MAPPING



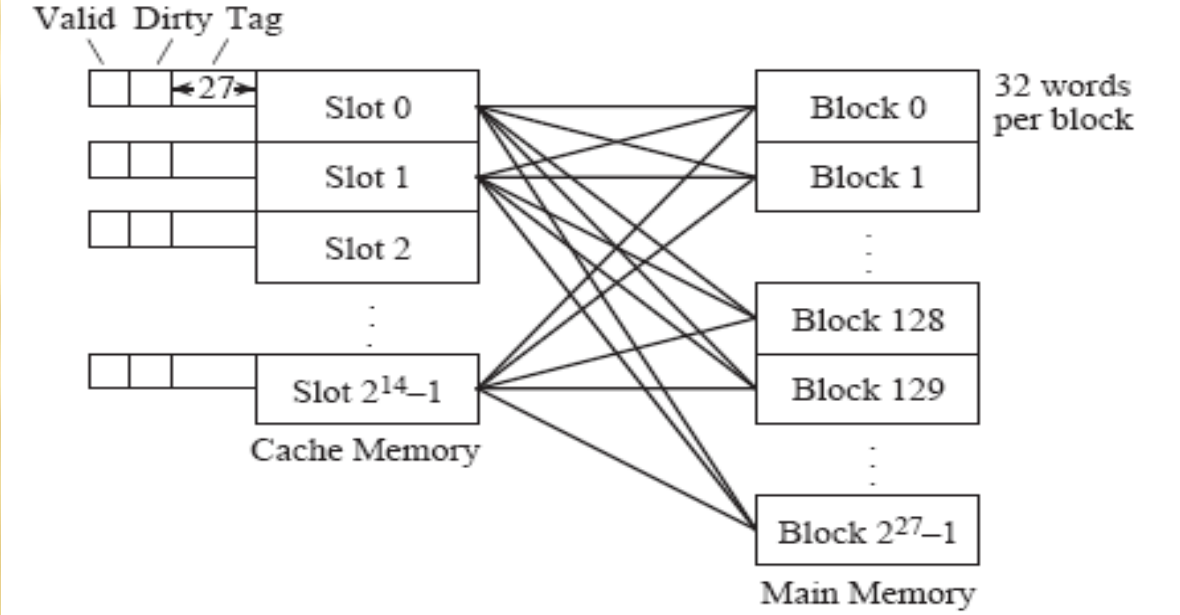
FULLY ASSOCIATIVE MAPPING

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



FULLY ASSOCIATIVE MAPPING

- 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.



FULLY ASSOCIATIVE MAPPING

- 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.

FULLY ASSOCIATIVE MAPPING

Advantages

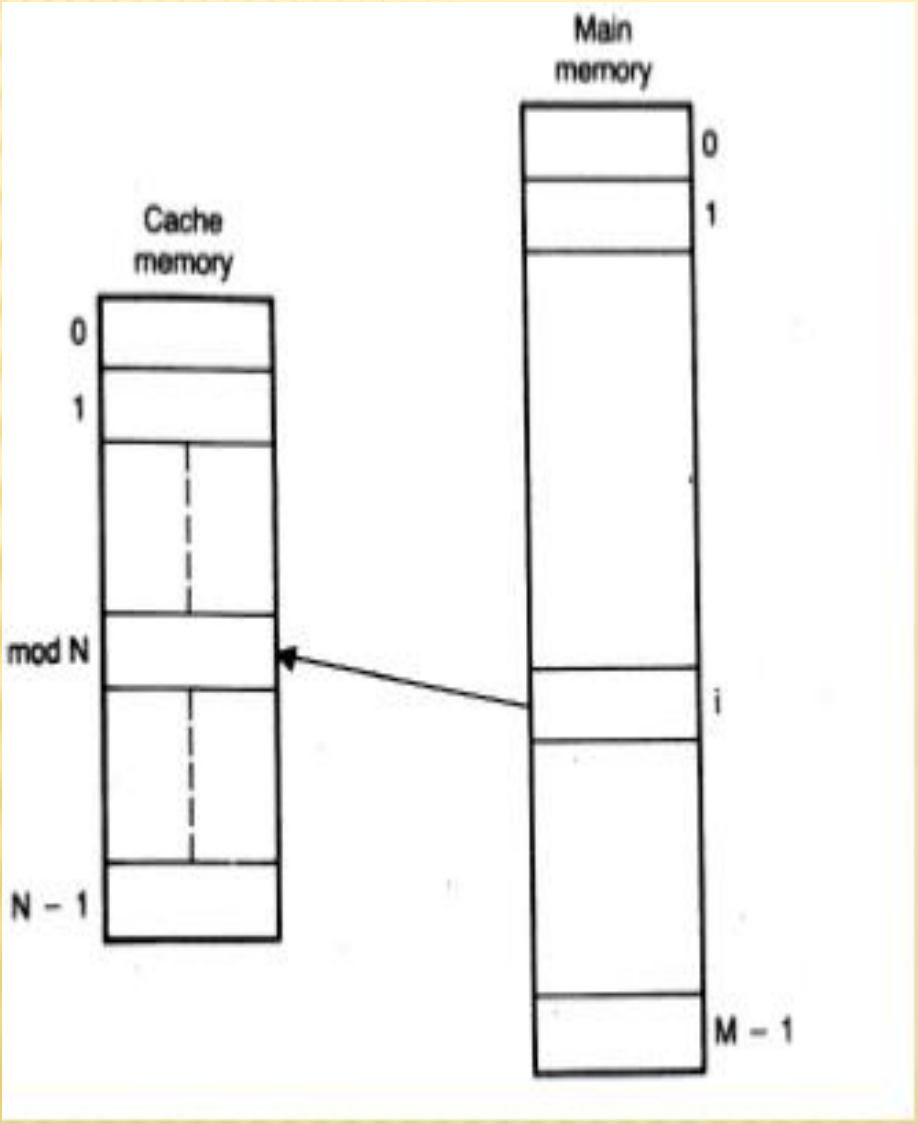
- Any main memory block can be placed into any cache slot.
- Regardless of how irregular the data and program references are, if a slot is available for the block, it can be stored in the cache.

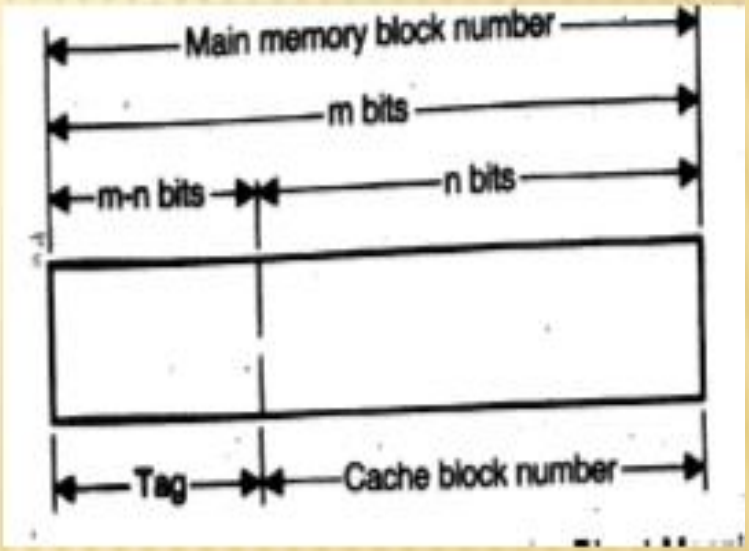
ASSOCIATIVE MAPPED CACHE

Disadvantages

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

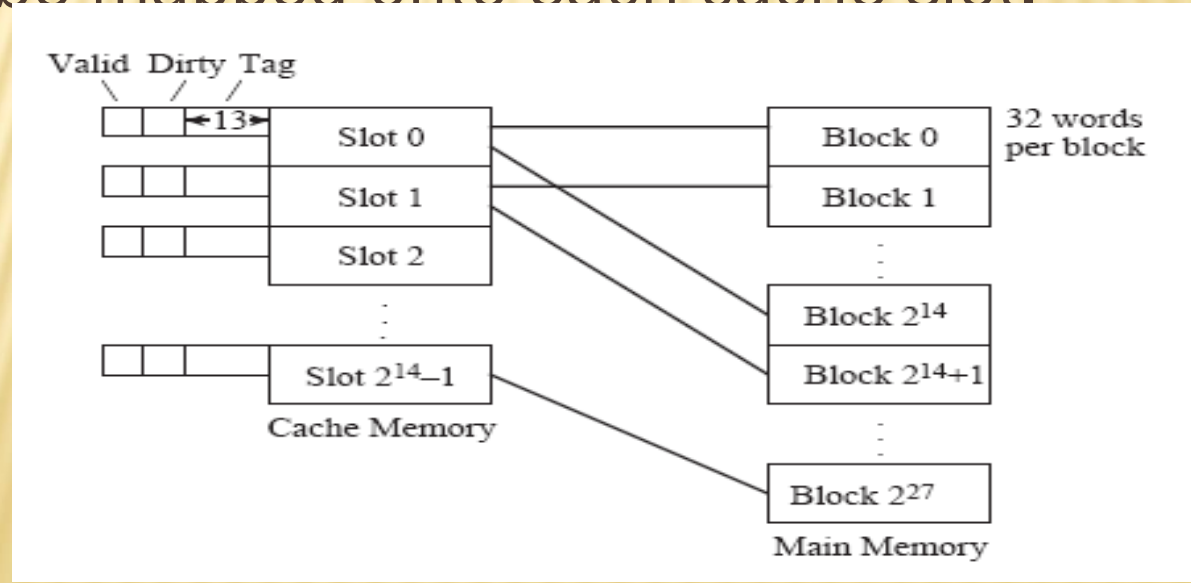
DIRECT MAPPING





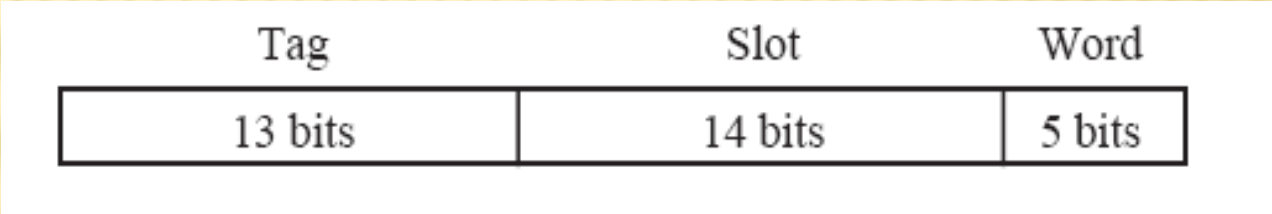
DIRECT-MAPPED CACHE

- 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.



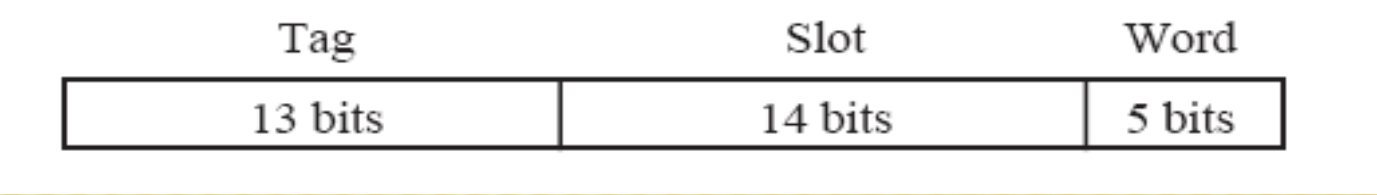
DIRECT-MAPPED CACHE

- 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.



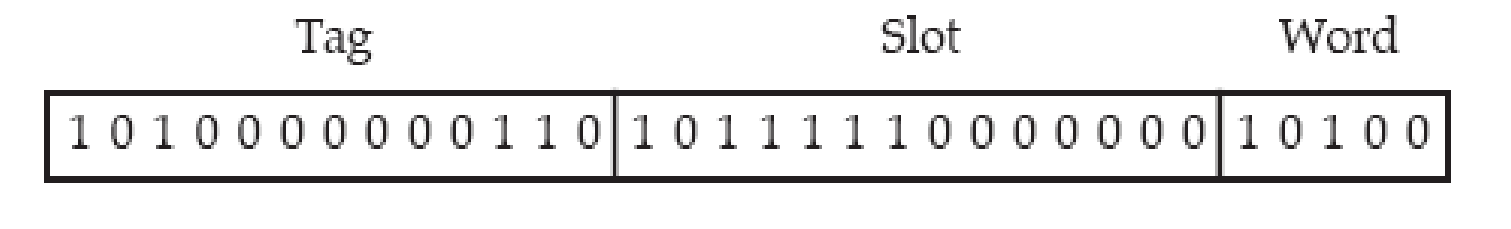
DIRECT-MAPPED CACHE

- 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

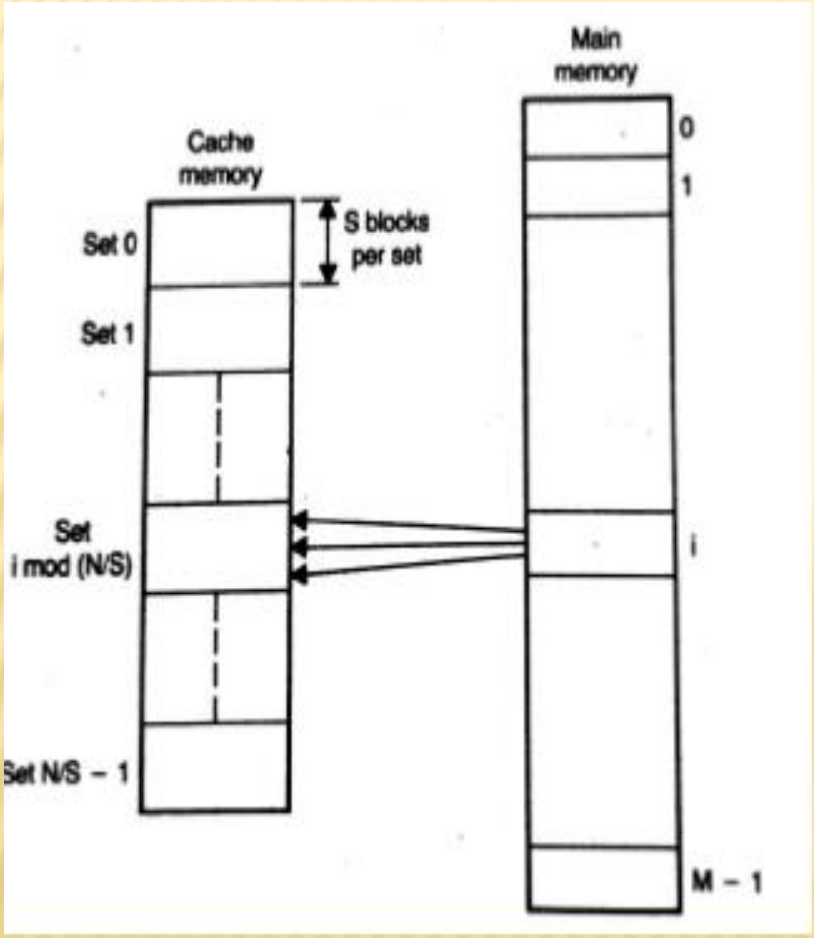
- The tag memory is much smaller than in associative mapped cache.
- No need for an associative search, since the slot field is used to direct the comparison to a single field.

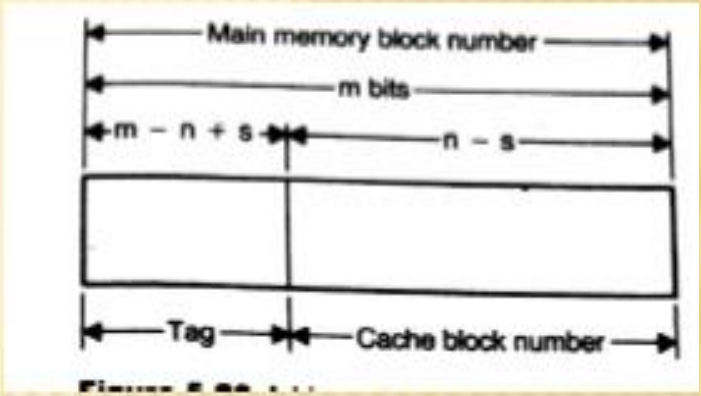
DIRECT-MAPPED CACHE

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.

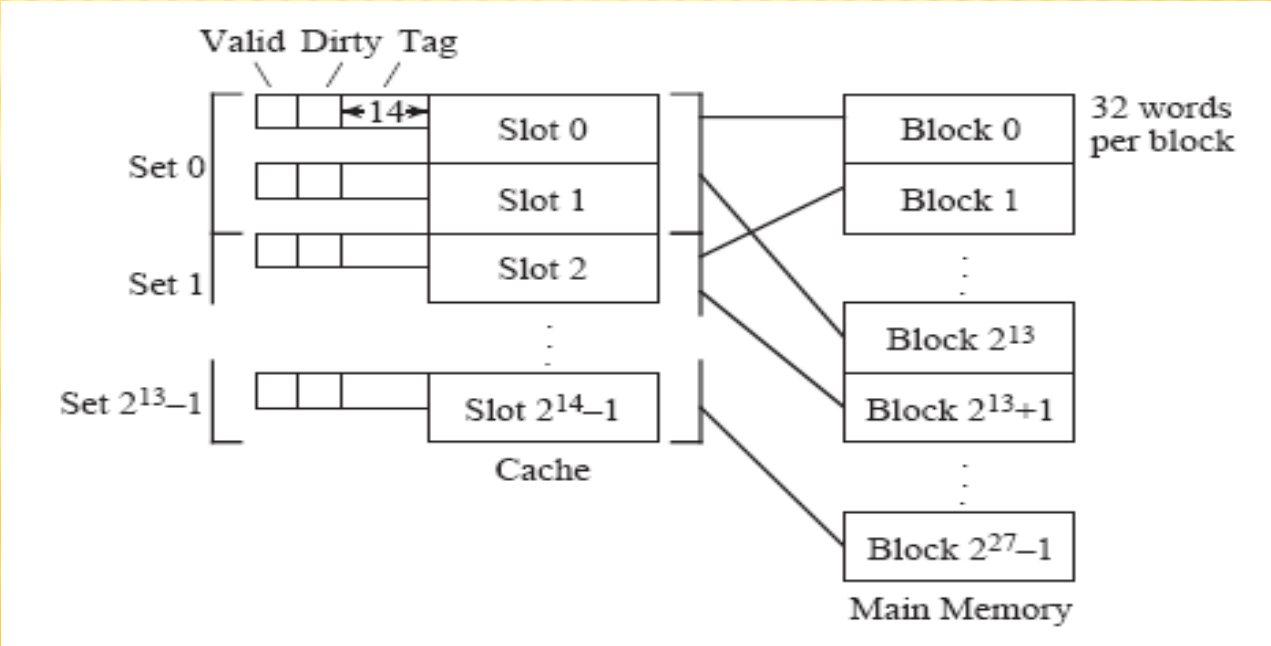
SET ASSOCIATIVE MAPPING





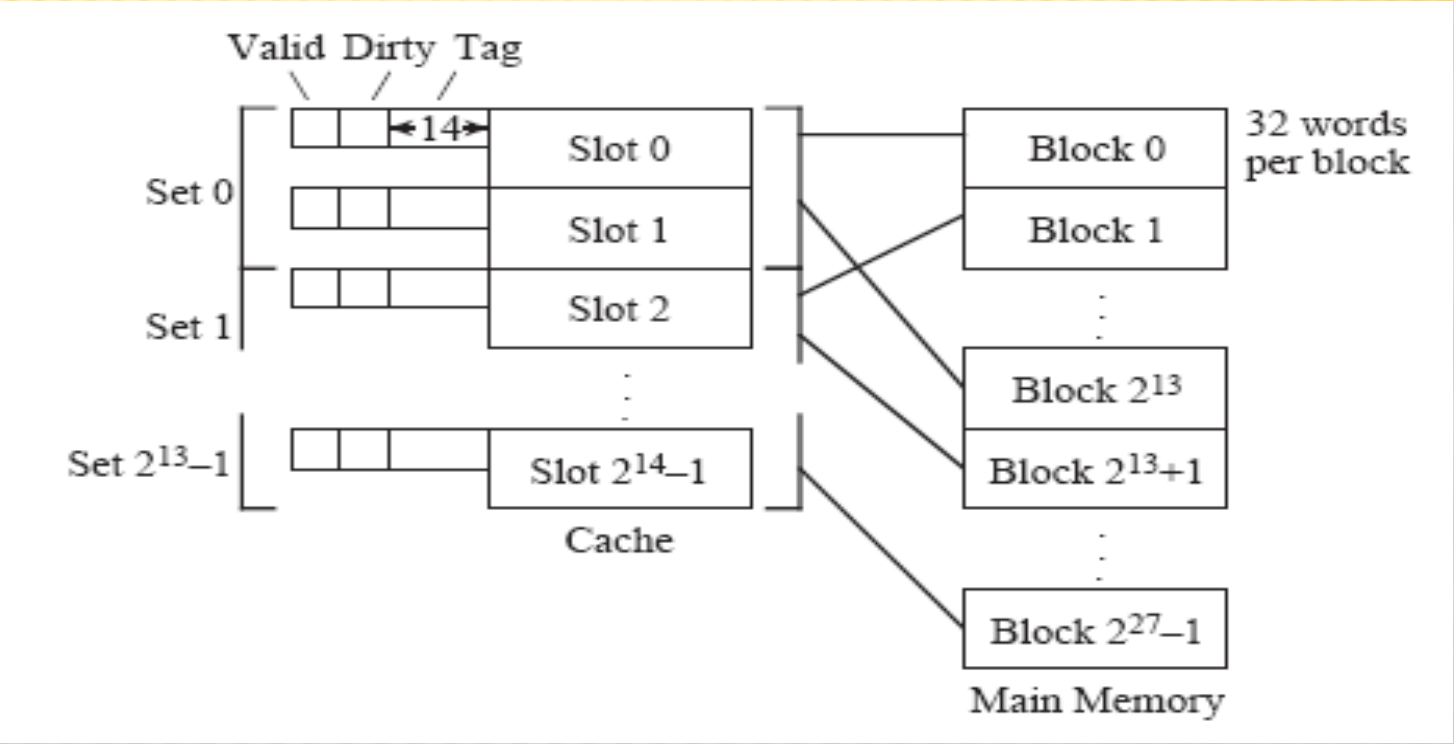
SET-ASSOCIATIVE MAPPED CACHE

- 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.



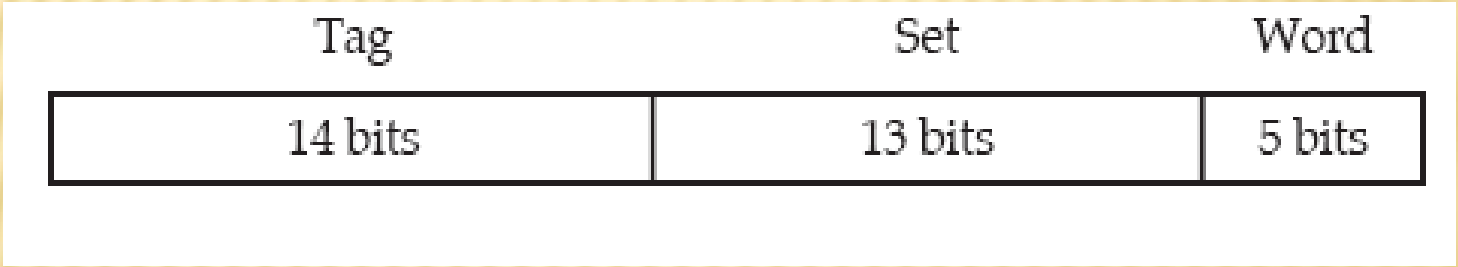
SET-ASSOCIATIVE MAPPED CACHE

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



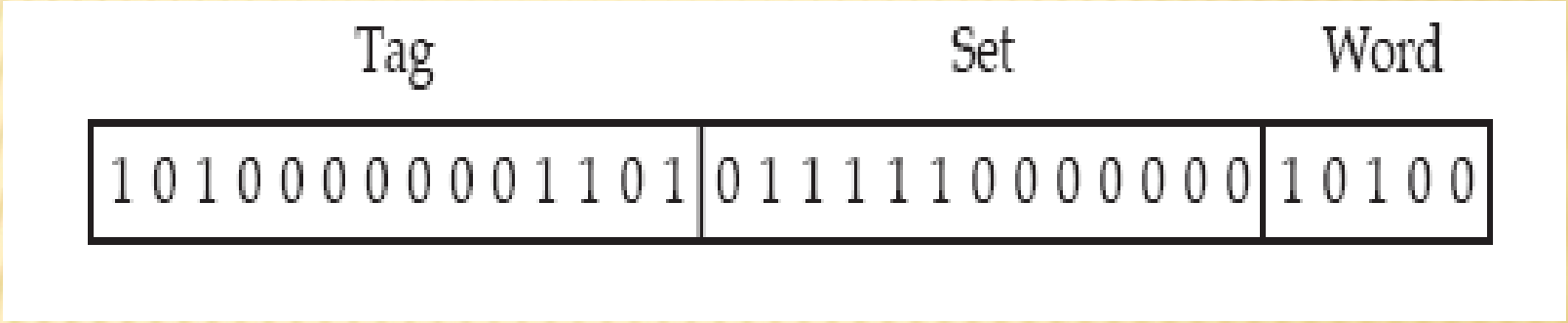
SET-ASSOCIATIVE MAPPED CACHE

- 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.



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.

The parameters of a computer memory system are specified as follows:

- Main memory size = 8K blocks
- Cache memory size = 512 blocks
- Block size = 8 words

Determine the size of the tag field of the main memory address under the following conditions:

a) Fully associative mapping

b) Direct mapping

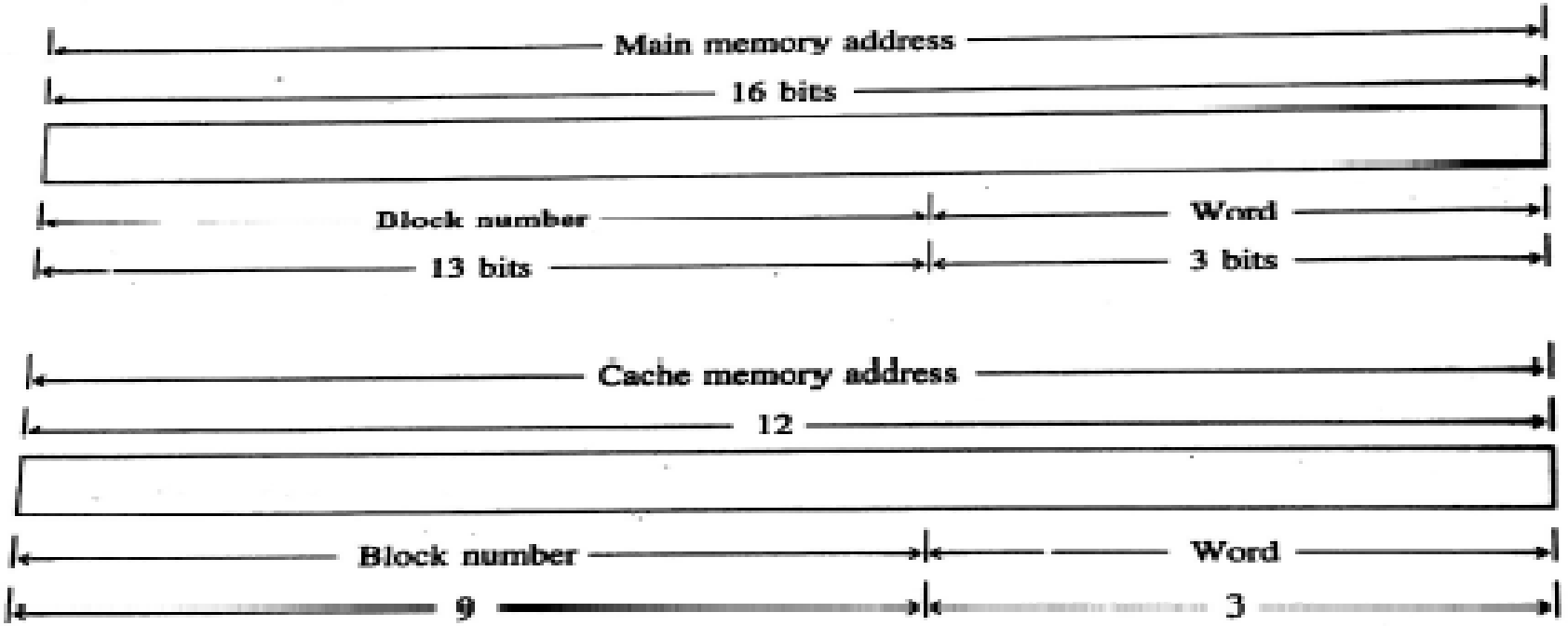
c) Set associative mapping with 16 blocks/set

SOLUTION

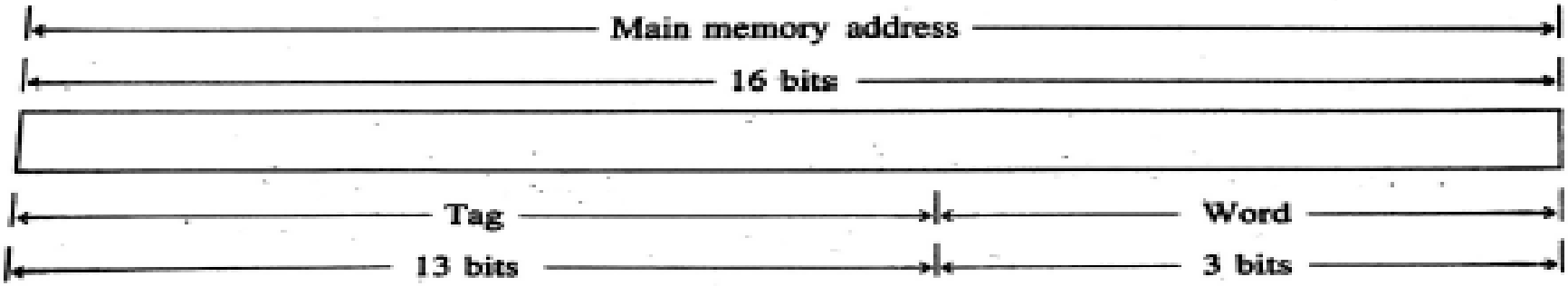
With the given data, compute the following:

- $M = 8K = 8192 = 2^{13}$, and thus $m = 13$.
- $N = 512 = 2^9$, and thus $n = 9$.
- Block size = 8 words = 2^3 words, and thus we require 3 bits to specify a word within a block.

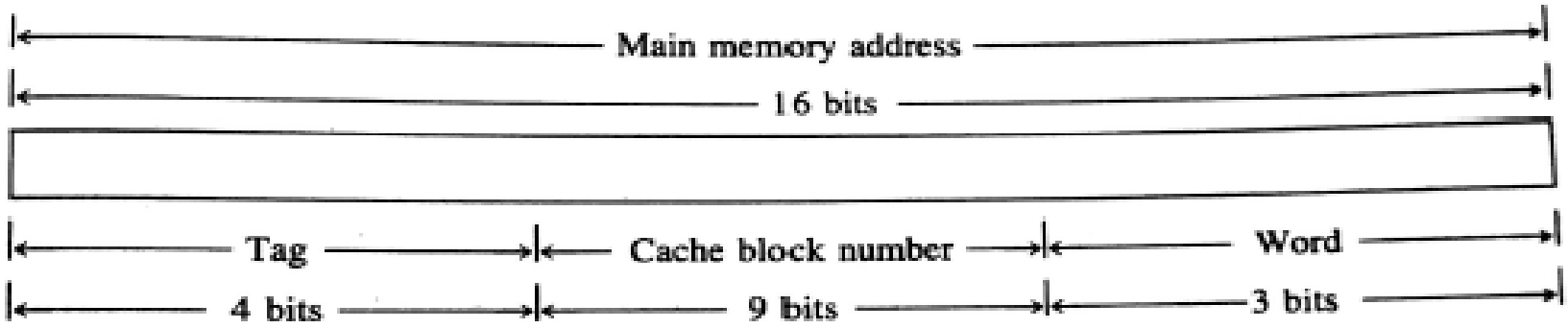
Using this information, we can determine the main and cache memory address formats as shown next:



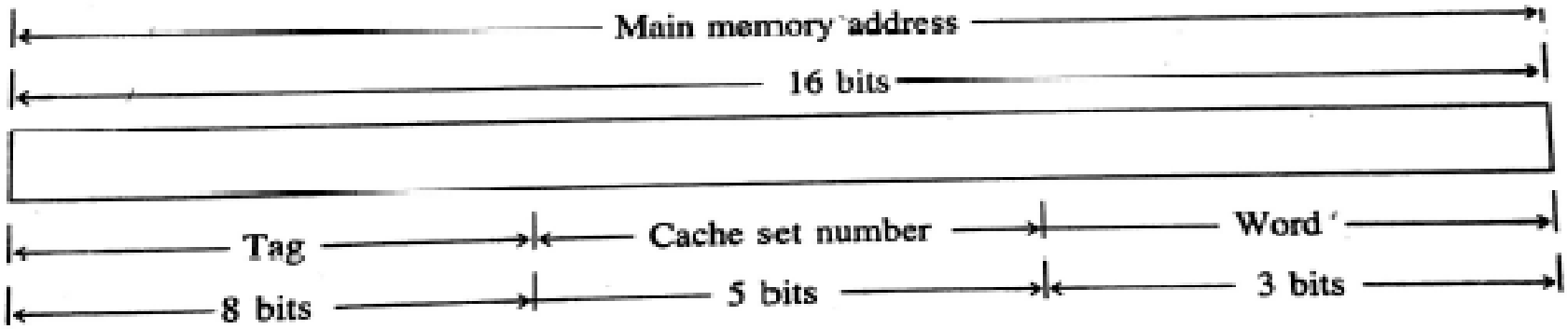
a) In this case, the size of the tag field is $m = 13 =$ bits:



b) In this case, the size of the tag field is $m - n = 13 - 9 = 4$ bits:



c) $S = 16 = 2^4$, and thus $S = 4$. Therefore. the size of the tag field is $m - n + s = 13 - 9 + 4 = 8$ bits:



INPUT/OUTPUT

There are three ways of transferring data between the computer and a physical I/O device:

- Programmed I/O: The computer executes a program to communicate with an external device through a register called the I/O port.
- Interrupt I/O: The computer executes a program called ISR to carry out the function requested by the external device.
- Direct memory access (DMA): Data transfer between the memory and the external device takes place without the involvement of the CPU.

PROGRAMMED I/O

Different techniques of addressing I/O ports:

- Standard I/O (isolated I/O)
  - Uses I/O/ M' pin.
  - I/O ports are separate.
  - Uses IN and OUT instructions to access I/O Ports.
  - The I/O port data must be moved into accumulator for processing.
- Memory mapped I/O
  - Does not use I/O/ M' pin
  - I/O ports are mapped to the processor's memory
  - MSB of the address indicates I/O or memory operation. If MSB=1, I/O port is selected and if MSB=0, Memory is accessed.
  - Processor can perform operation on port data without moving into the CPU

INTERRUPT I/O

- There are two ways of servicing multiple interrupts:
  - Polled technique
  - Daisy chain technique

POLLED TECHNIQUE

- Handled by software
- One general service routine is used for all devices.
- Priority is with respect to how the ISR poll each device.
- Once the source of interrupt is identified, it branches to the service routine for the device.

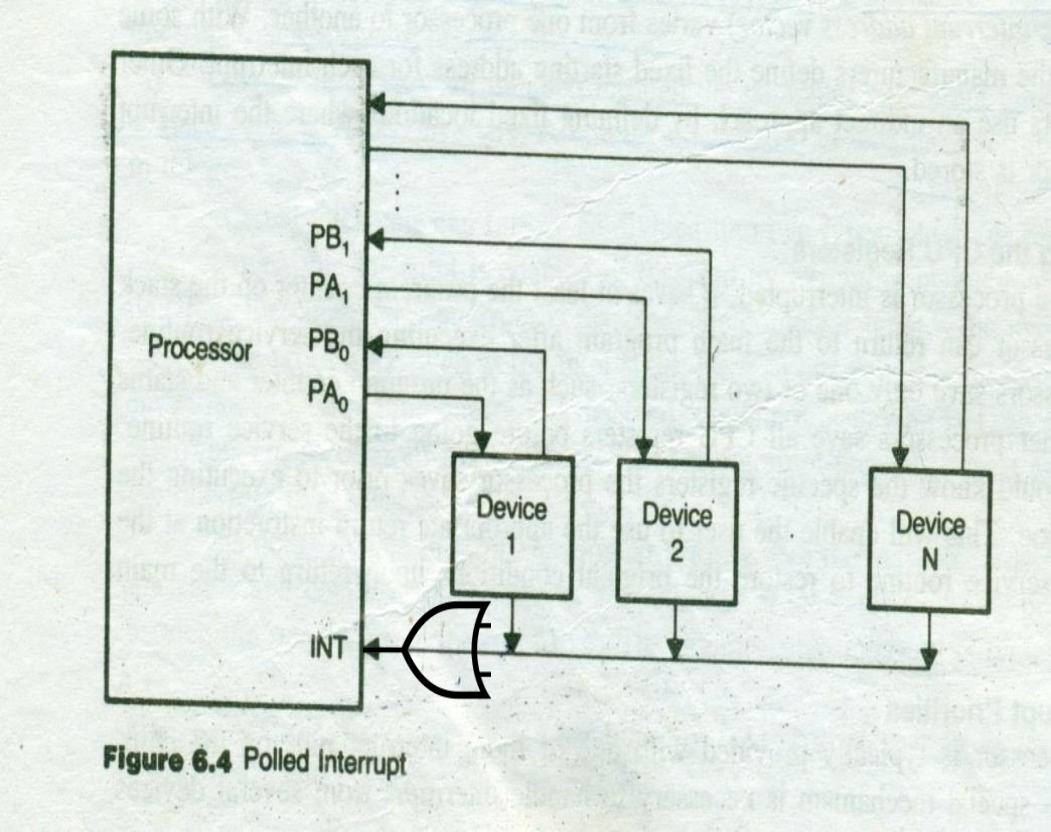


Figure 6.4 Polled Interrupt

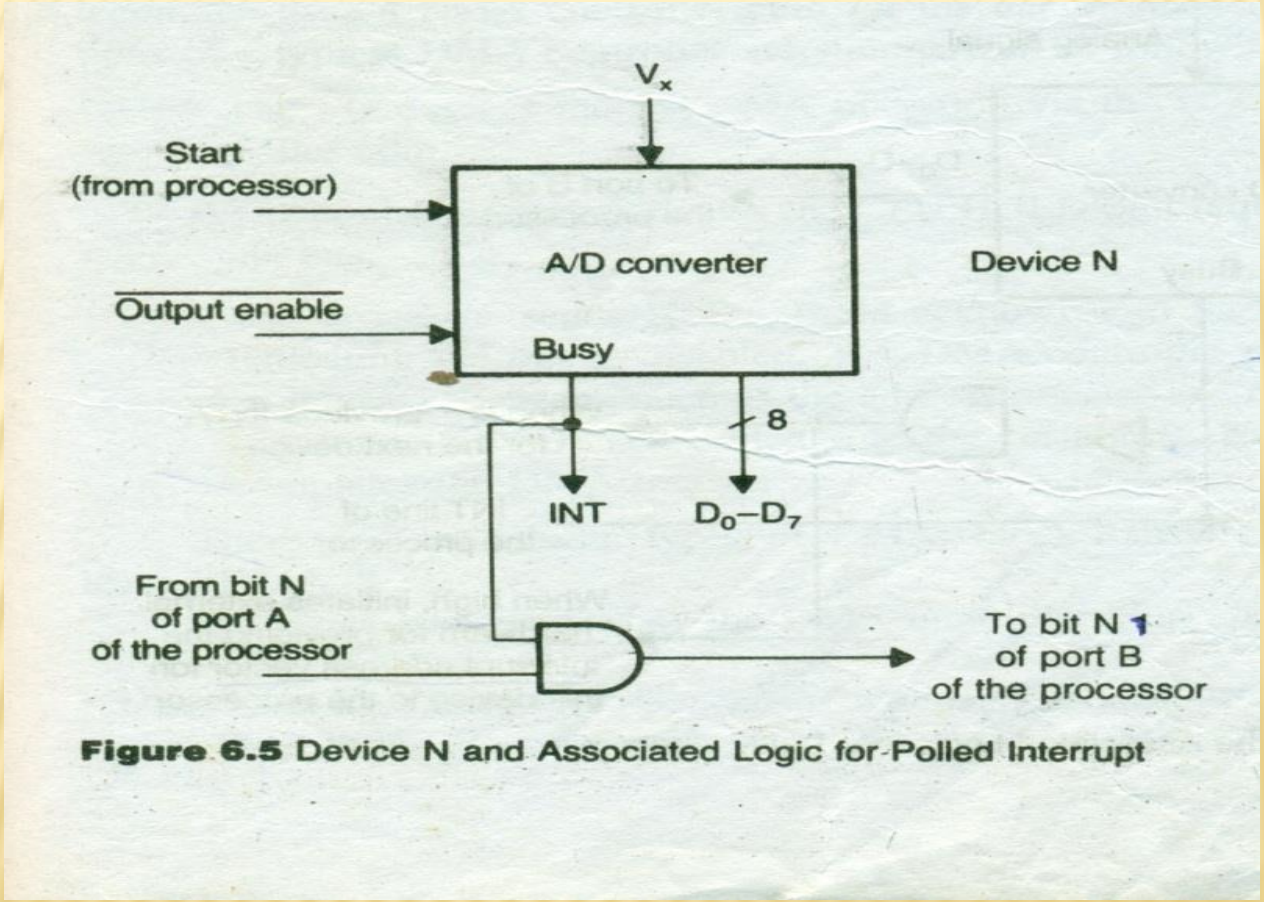


Figure 6.5 Device N and Associated Logic for Polled Interrupt

DAISY CHAIN TECHNIQUE

- Devices are connected in Daisy chain fashion.
- Whenever devices sends an interrupt, processor sends $\overline{INTA}$ to the device with the highest priority.
- If this device has sent the interrupt, it accepts it and send the signal to generate the interrupt address vector which is sent to the processor.
- If it has not sent the interrupt, $\overline{INTA}$ is passed to the device with the next priority.

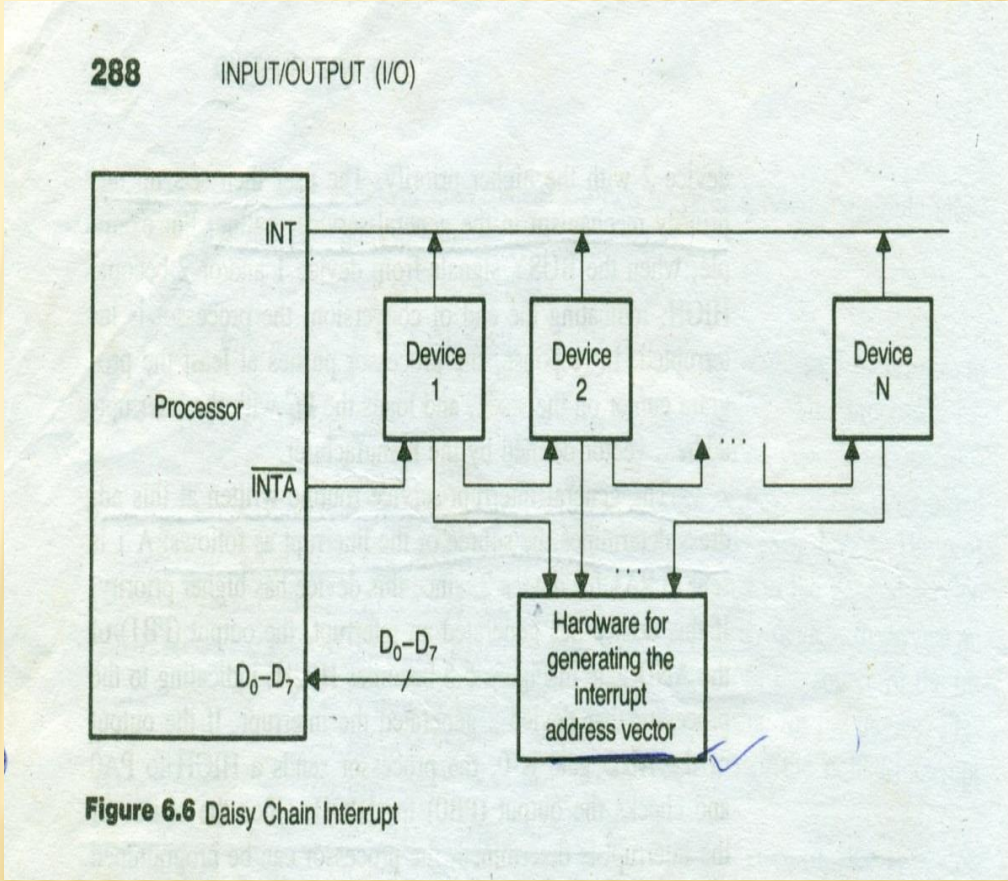


Figure 6.6 Daisy Chain Interrupt

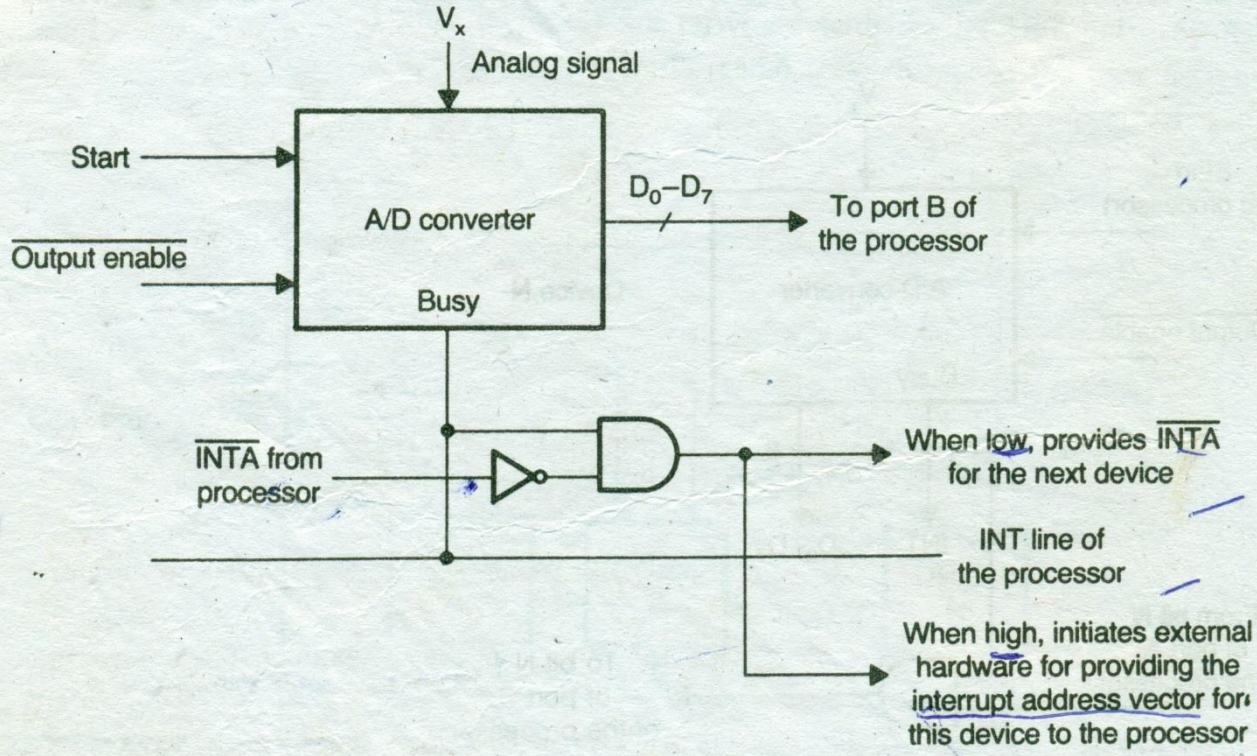


Figure 6.7 Each device and the Associated Logic in a Daisy Chain

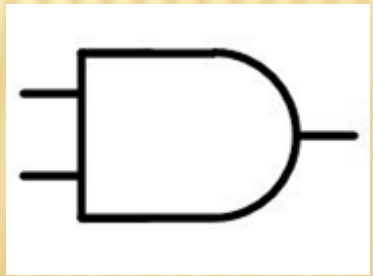
DIRECT MEMORY ACCESS(DMA)

There are three basic types of DMA:

- Cycle stealing
- Interleaved
- Block transfer

Cycle stealing

- Data transfer between memory and I/O device occurs on word by word basis.
- DMA takes away or steals a cycle without CPU recognition.
- DMA controller controls $\overline{INHIBIT}$ signal.
- When DMA operation is desired, it makes $\overline{INHIBIT}$ signal low for one clock cycle.

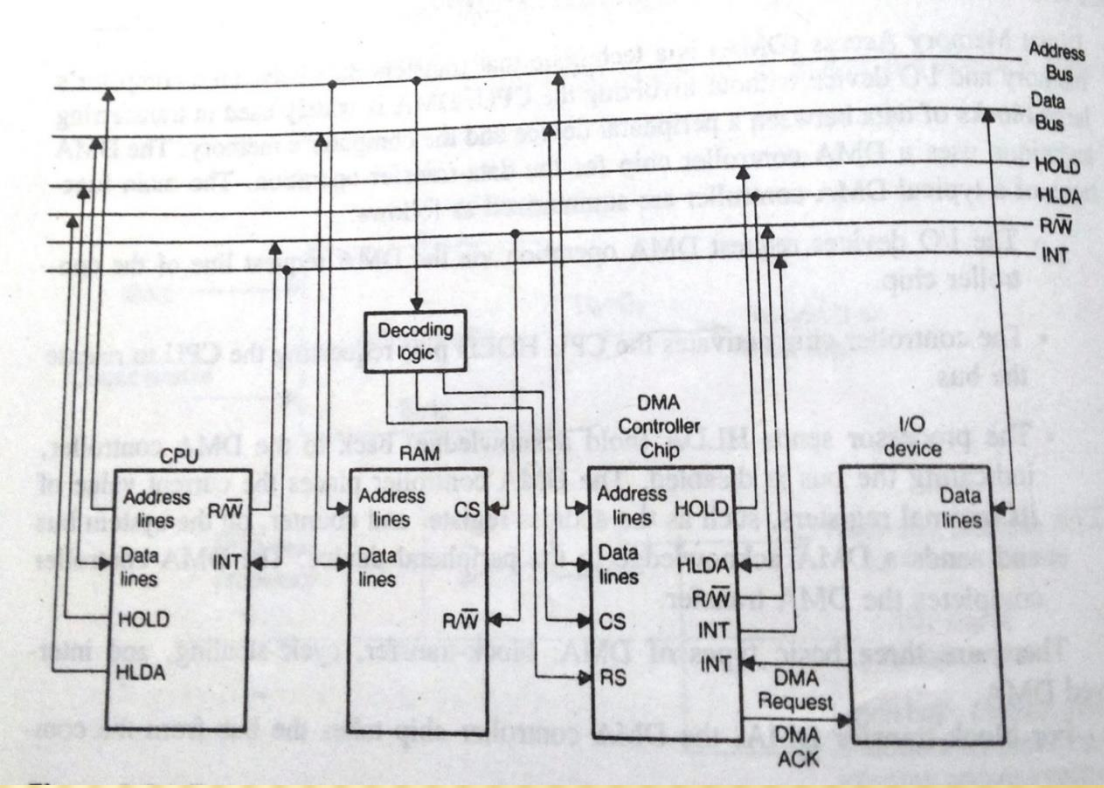


Interleaved DMA

- DMA controller takes over the bus when the CPU is not using it.
- For example: When CPU is performing ALU operation or incrementing the pointer.
- Data transfer takes place for a period of time.

BLOCK TRANSFER DMA

- I/O device requests for the DMA using DMA request line.
- DMA controller chip then sends a HOLD signal to the CPU.
- Waits for HLDA signal from the CPU.
- When DMA controller receives HLDA, it sends DMA ACK to I/O device.
- On completion of the data transfer, controller sends INT to the CPU and returns the bus to CPU.



DMA controller chip has three registers:

- Address register
- Terminal count register
- Status register