Digital Design and Computer Organization
Module 5

3rd SEM

B-sec

Topics

❖ Arithmetic

❑ Addition and Subtraction of signed Numbers.

❑ Design of Fast Adders

❑ Multiplication of Positive Numbers

❑ Integer Division

❖ Basic Processing Unit

❑ Some fundamental concepts

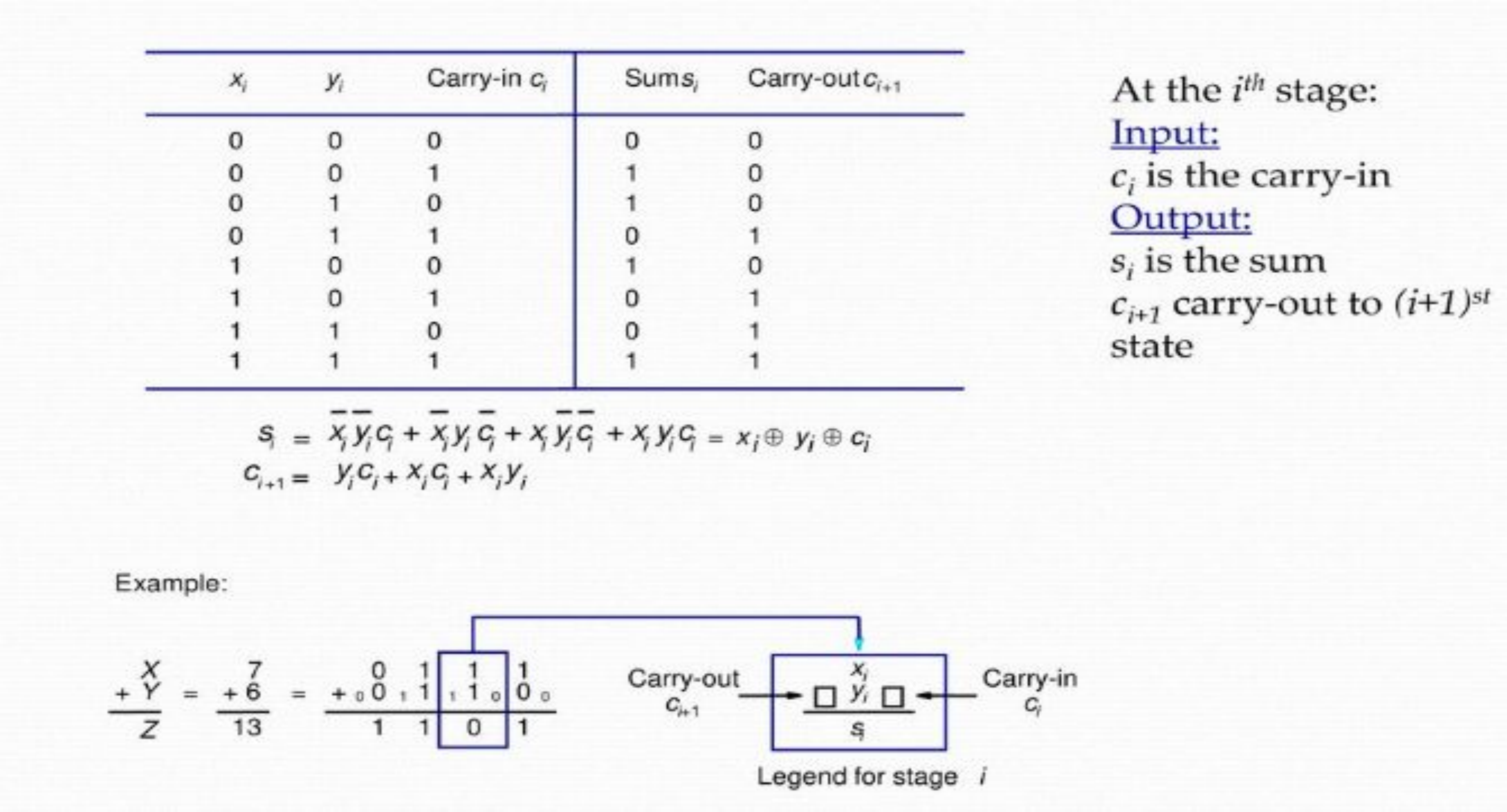
❑ Execution of a Complete Instructions

❑ Multiple Bus Organizations

Arithmetic

- A basic operation in all digital computers is the addition or subtraction of 2 numbers.
- Arithmetic operations occur at the machine level instructions. They are implemented , with basic logic functions such as AND, OR, NOT and XOR in ALU subsystem of the Processor.
- We present the logic circuits to implement the arithmetic operations.
- Multiply and divide operations , which require more complex circuitry than either addition of subtraction.

Addition / subtraction of Signed numbers



x_i	y_i	Carry-in c_i	Sum s_i	Carry-out c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$s_i = \bar{x}_i \bar{y}_i c_i + \bar{x}_i y_i \bar{c}_i + x_i \bar{y}_i \bar{c}_i + x_i y_i c_i = x_i \oplus y_i \oplus c_i$$

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

Example:

$$\begin{array}{r} X \\ +\,Y \\ \hline Z \end{array} = \begin{array}{r} 7 \\ +\,6 \\ \hline 13 \end{array} = \begin{array}{rrrr} 0 & 1 & 1 & 1 \\ +\,_0 0 & _1 1 & _1 1 & _0 0\ _0 \\ \hline 1 & 1 & \boxed{0} & 1 \end{array}$$

Carry-out c_{i+1} → □ $\begin{array}{c} x_i \\ y_i \\ \hline s_i \end{array}$ □ ← Carry-in c_i

Legend for stage i

At the i^{th} stage:

Input:

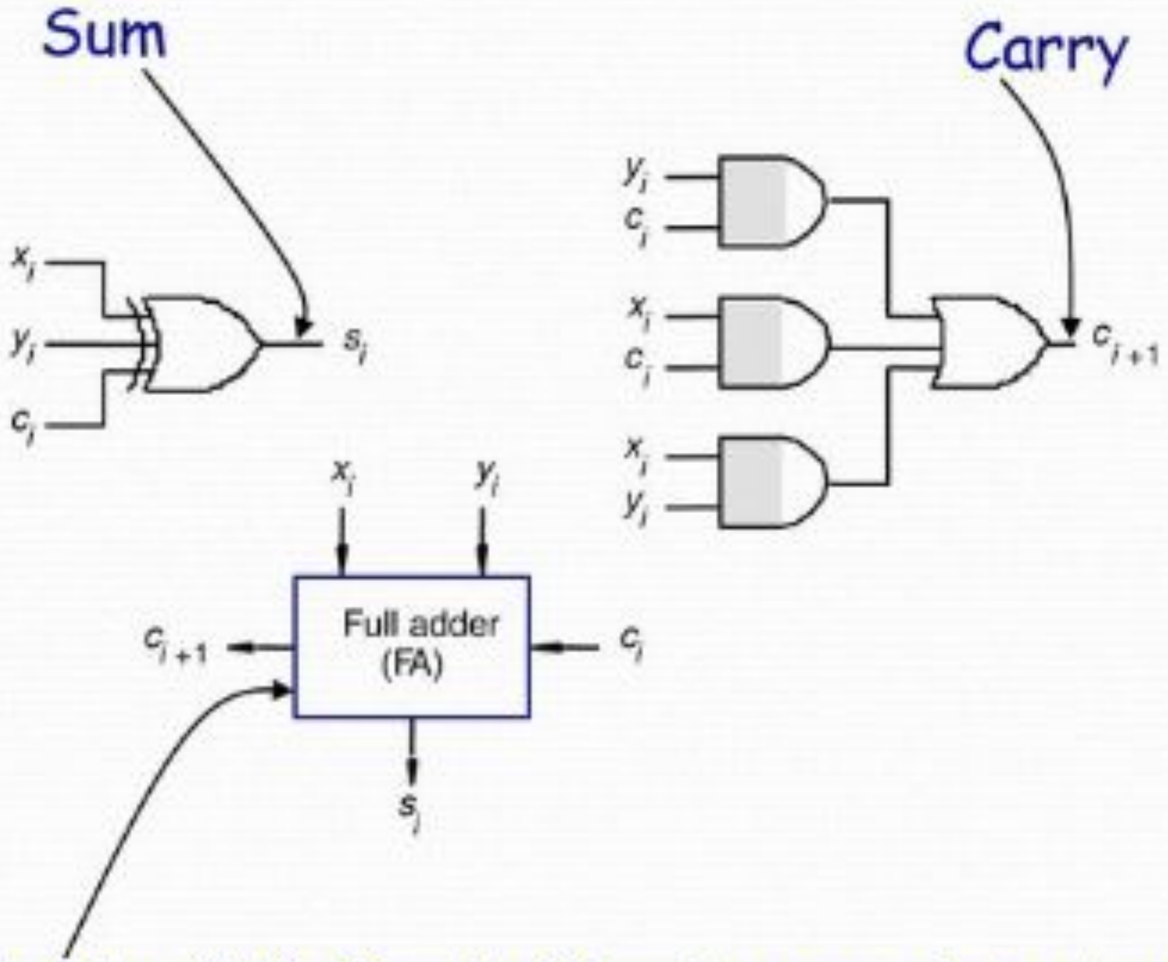
c_i is the carry-in

Output:

s_i is the sum

c_{i+1} carry-out to $(i+1)^{st}$ state

Addition logic for a single stage



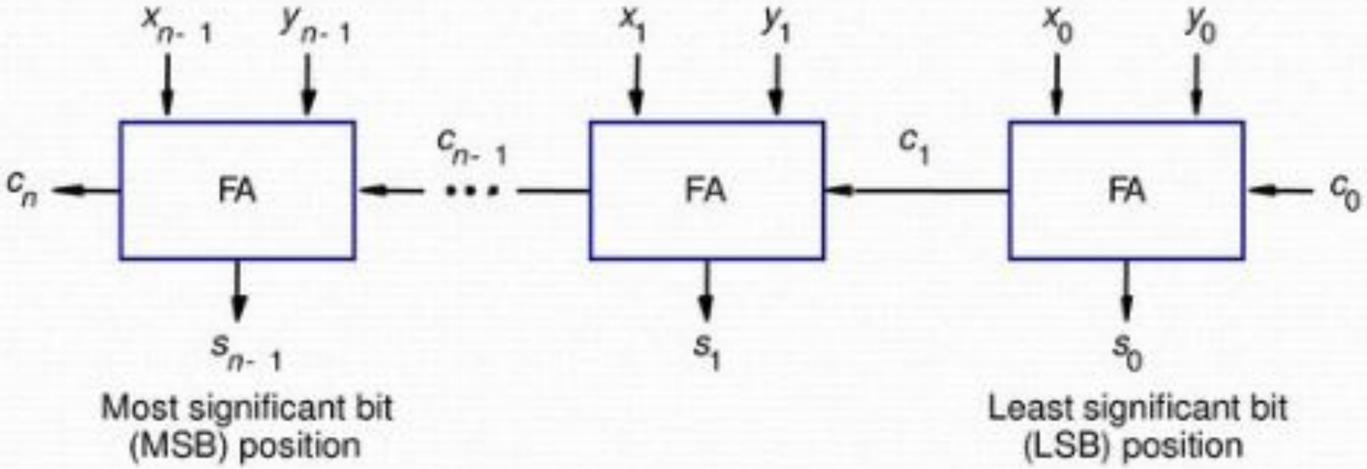
Sum

Carry

Full Adder (FA): Symbol for the complete circuit
for a single stage of addition.

n-bit adder

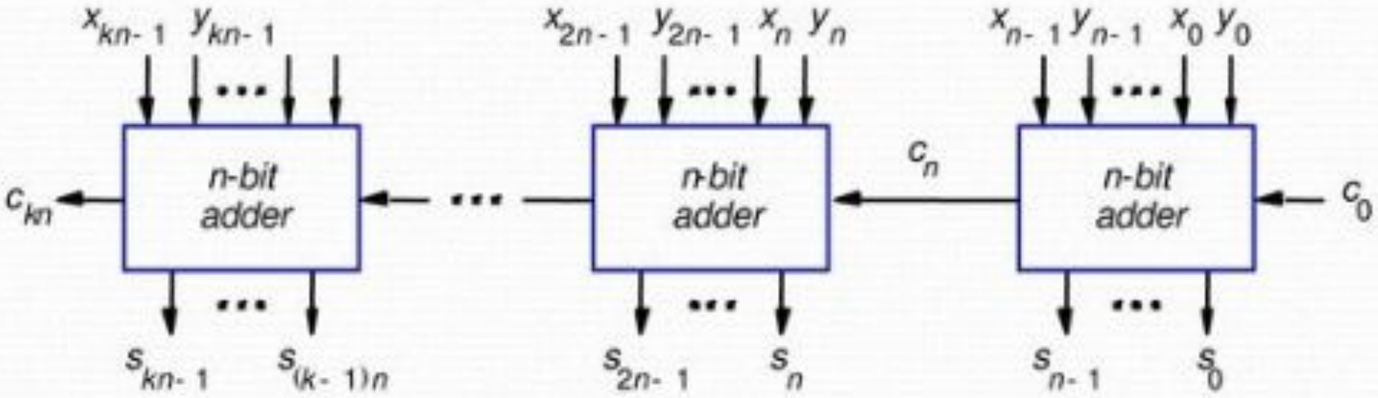
- Cascade *n* full adder (FA) blocks to form a *n*-bit adder.
- Carries propagate or ripple through this cascade, *n*-bit ripple carry adder.



Carry-in c_0 into the LSB position provides a convenient way to perform subtraction.

K n-bit adder

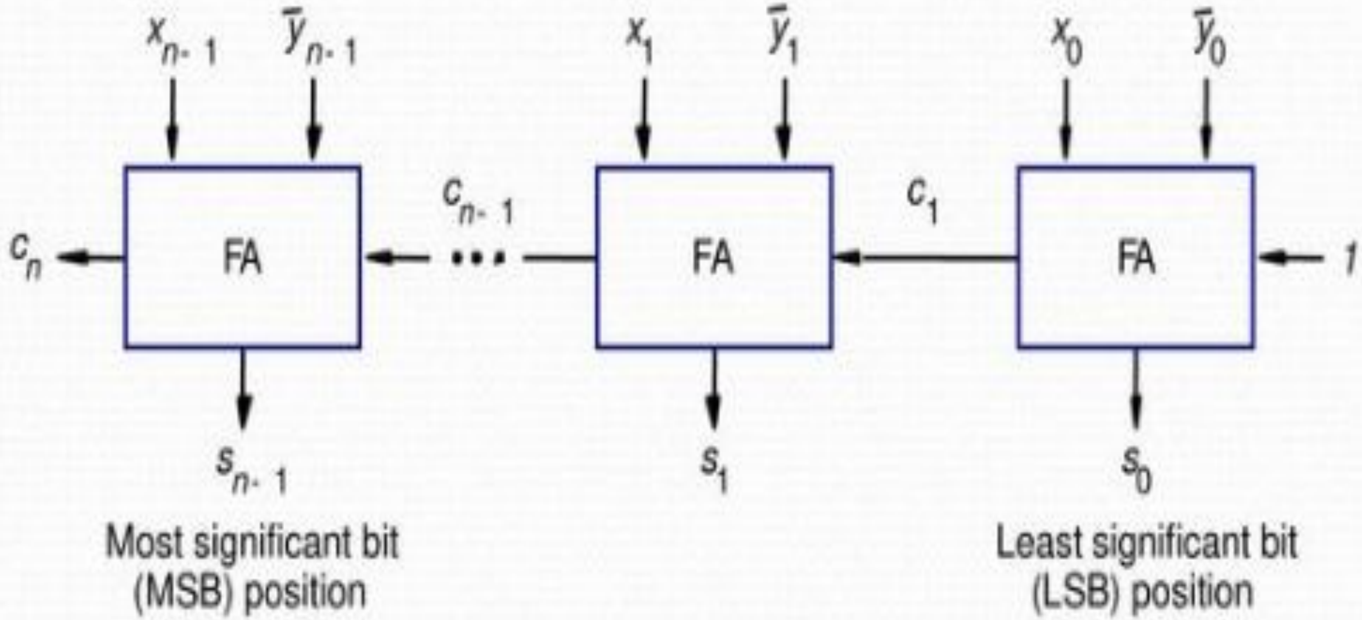
K n-bit numbers can be added by cascading *k n*-bit adders.



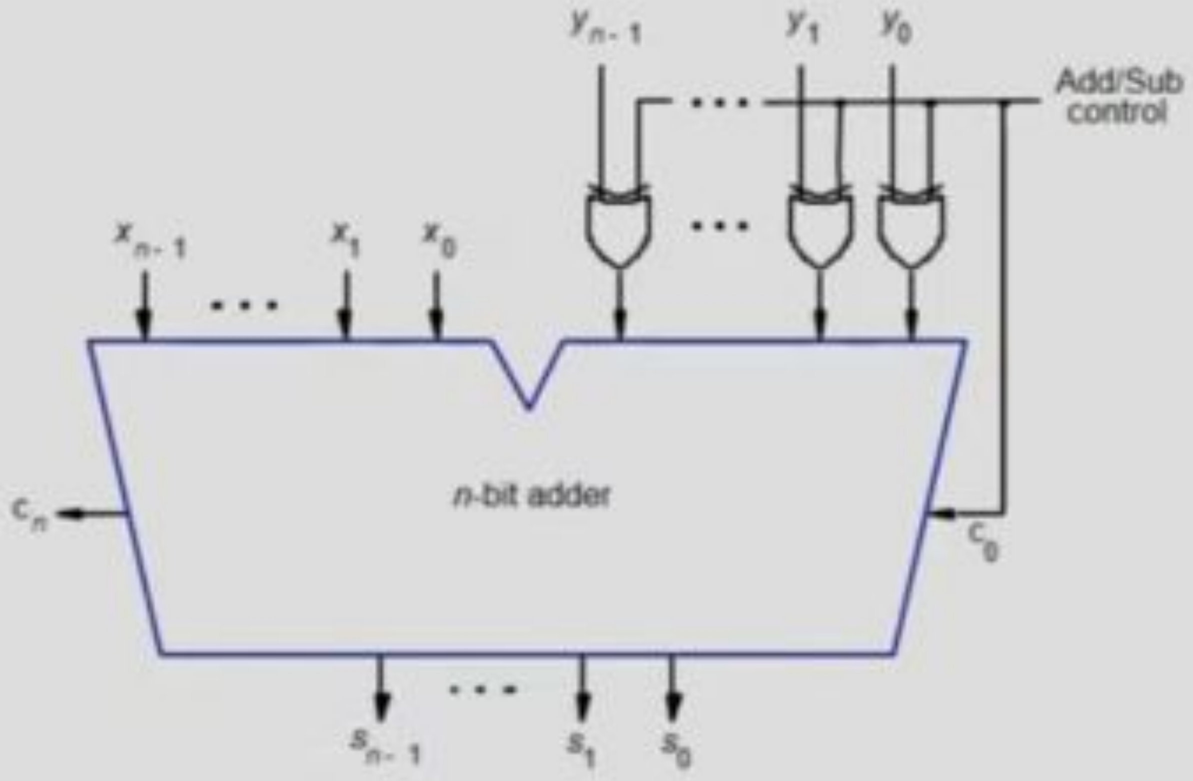
Each *n*-bit adder forms a block, so this is cascading of blocks.
Carries ripple or propagate through blocks, Blocked Ripple Carry Adder

n-bit subtractor

- Recall $X - Y$ is equivalent to adding 2′s complement of Y to X.
- 2′s complement is equivalent to 1′s complement + 1.
- $\mathrm{X} - \mathrm{Y} = \mathrm{X} + \overline{\mathrm{Y}} + 1$
- 2′s complement of positive and negative numbers is computed similarly.



n-bit subtractor continued..



- Add/sub control = 0, addition.
- Add/sub control = 1, subtraction.

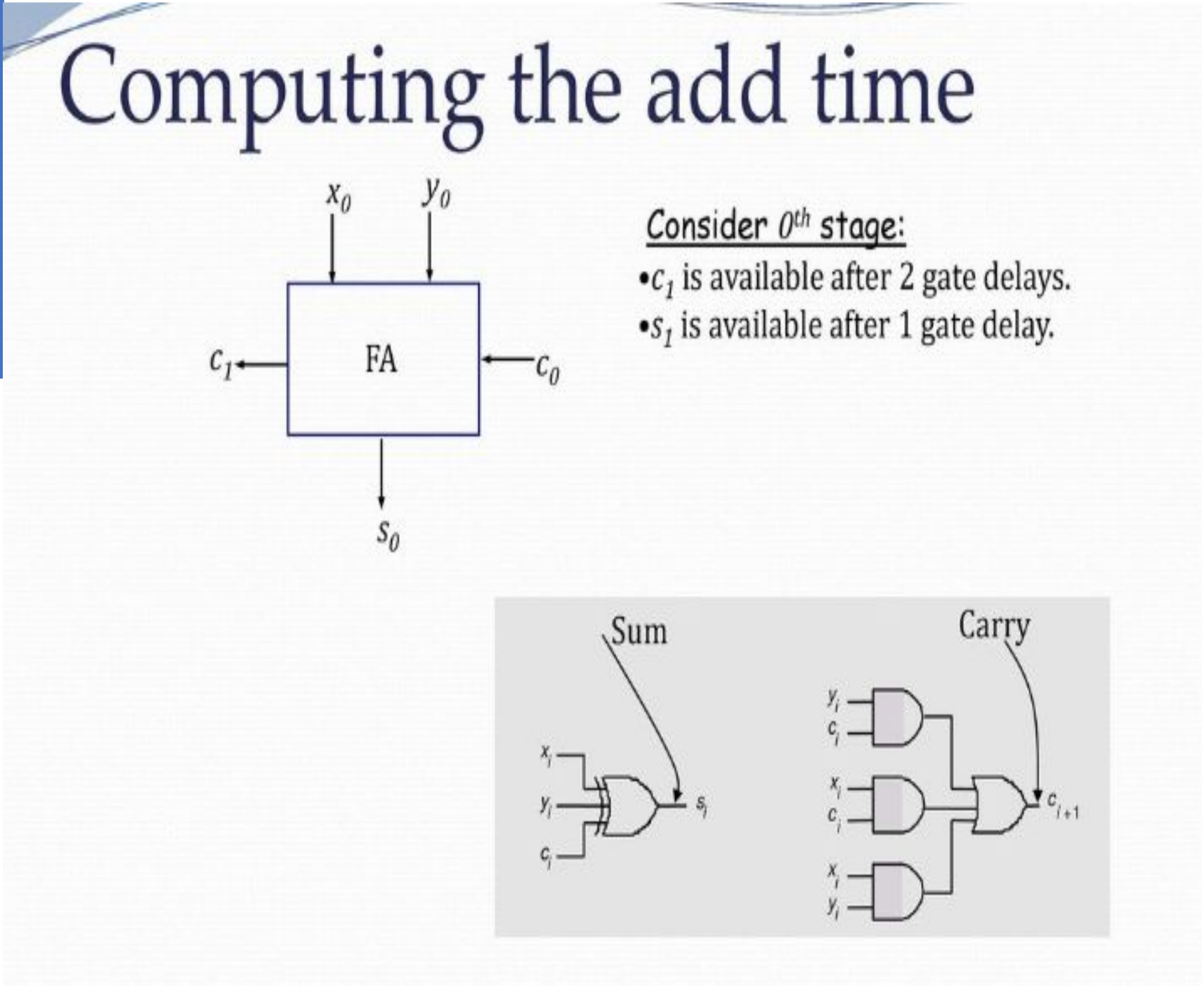
Sub =1

x-y

7 - 3

x	0111
y	0011
	0100

Design of Fast Adder

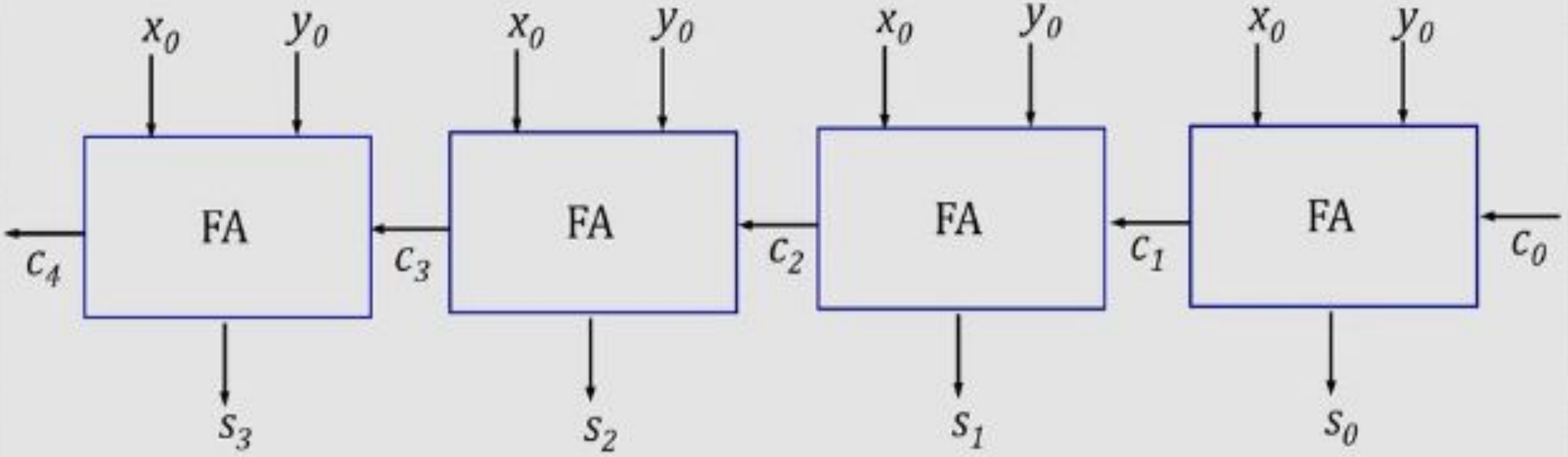


Computing the add time

Consider 0^{th} stage:

- c_1 is available after 2 gate delays.
- s_1 is available after 1 gate delay.

Cascade of 4 Full Adders, or a 4-bit adder



- s_0 available after 1 gate delays, c_1 available after 2 gate delays.
- s_1 available after 3 gate delays, c_2 available after 4 gate delays.
- s_2 available after 5 gate delays, c_3 available after 6 gate delays.
- s_3 available after 7 gate delays, c_4 available after 8 gate delays.

For an n-bit adder, s_{n-1} is available after $2n-1$ gate delays
c_n is available after $2n$ gate delays.

Fast addition

Recall the equations:

$$s_i = x_i \oplus y_i \oplus c_i$$

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

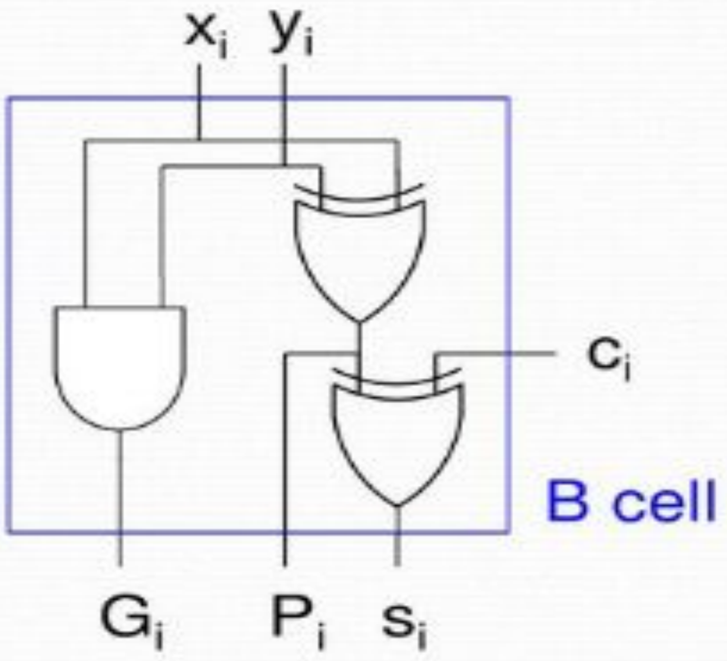
Second equation can be written as:

$$c_{i+1} = x_i y_i + (x_i + y_i) c_i$$

We can write:

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

$$where\ G_i = x_i y_i\ and\ P_i = x_i + y_i$$



- G_i is called generate function and P_i is called propagate function
- G_i and P_i are computed only from x_i and y_i and not c_i, thus they can be computed in one gate delay after X and Y are applied to the inputs of an n-bit adder.
- Ci+1 = 1 only when Generate= 1 and Propagate = 0 or 1
- So we can modify the Pi

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

$$where\ G_i = x_i y_i\ and\ P_i = x_i \oplus y_i$$

Carry lookahead

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

$$c_i = G_{i-1} + P_{i-1} c_{i-1}$$

$$\Rightarrow c_{i+1} = G_i + P_i (G_{i-1} + P_{i-1} c_{i-1})$$

continuing

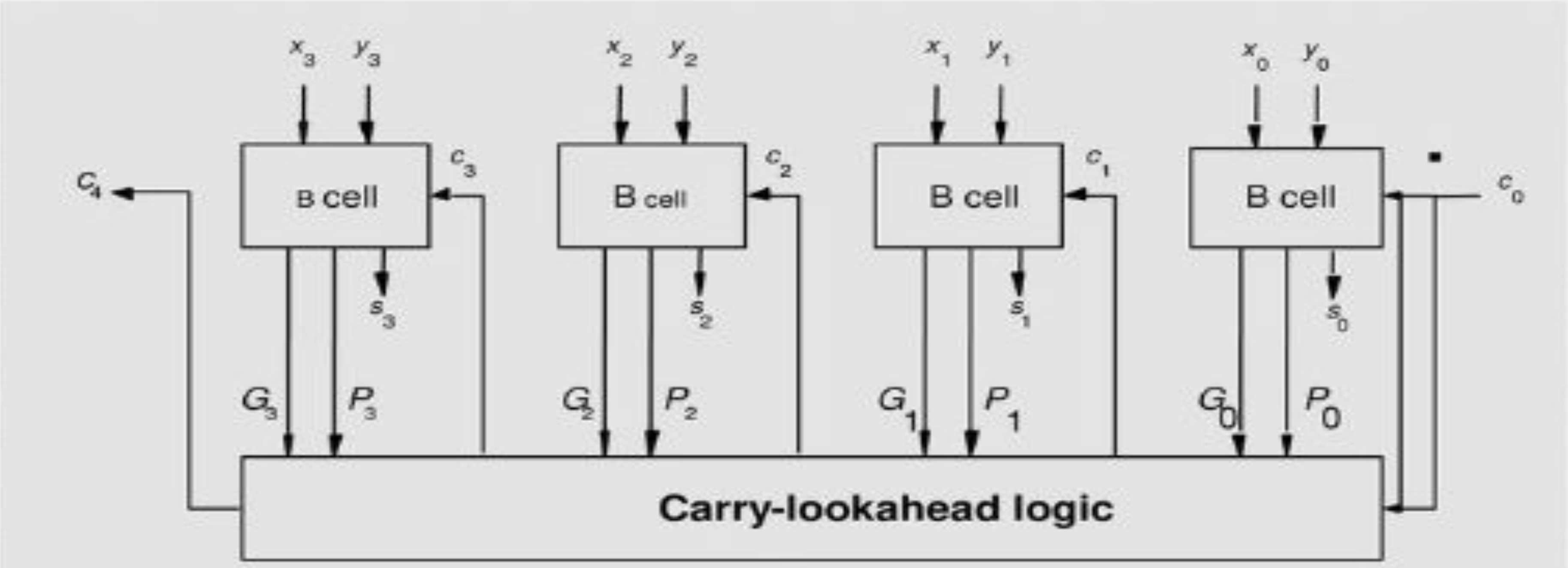
$$\Rightarrow c_{i+1} = G_i + P_i (G_{i-1} + P_{i-1} (G_{i-2} + P_{i-2} c_{i-2}))$$

until

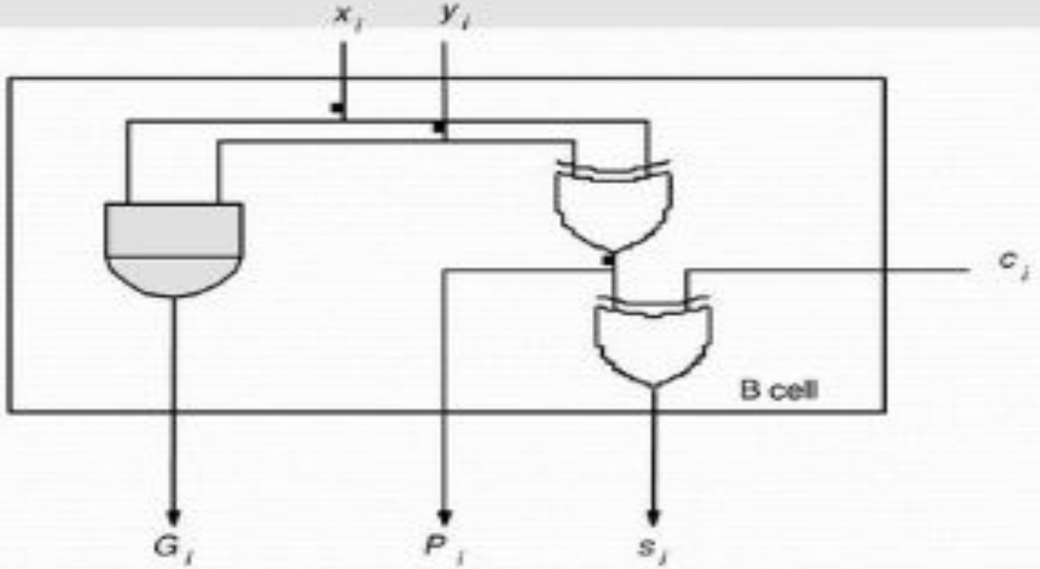
$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + .. + P_i P_{i-1} .. P_1 G_0 + P_i P_{i-1} ... P_0 c_0$$

- All carries can be obtained 3 gate delays after *X, Y* and c_0 are applied.
    - -One gate delay for P_i and G_i
    - -Two gate delays in the AND-OR circuit for c_{i+1}
- All sums can be obtained 1 gate delay after the carries are computed.
- Independent of *n*, *n*-bit addition requires only 4 gate delays.
- This is called Carry Lookahead adder.

Carry-lookahead adder



4-bit carry-lookahead adder



B-cell for a single stage

Blocked Carry-Lookahead adder

Carry-out from a 4-bit block can be given as:

$$c_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$$

Rewrite this as:

$$P_0^I = P_3P_2P_1P_0$$

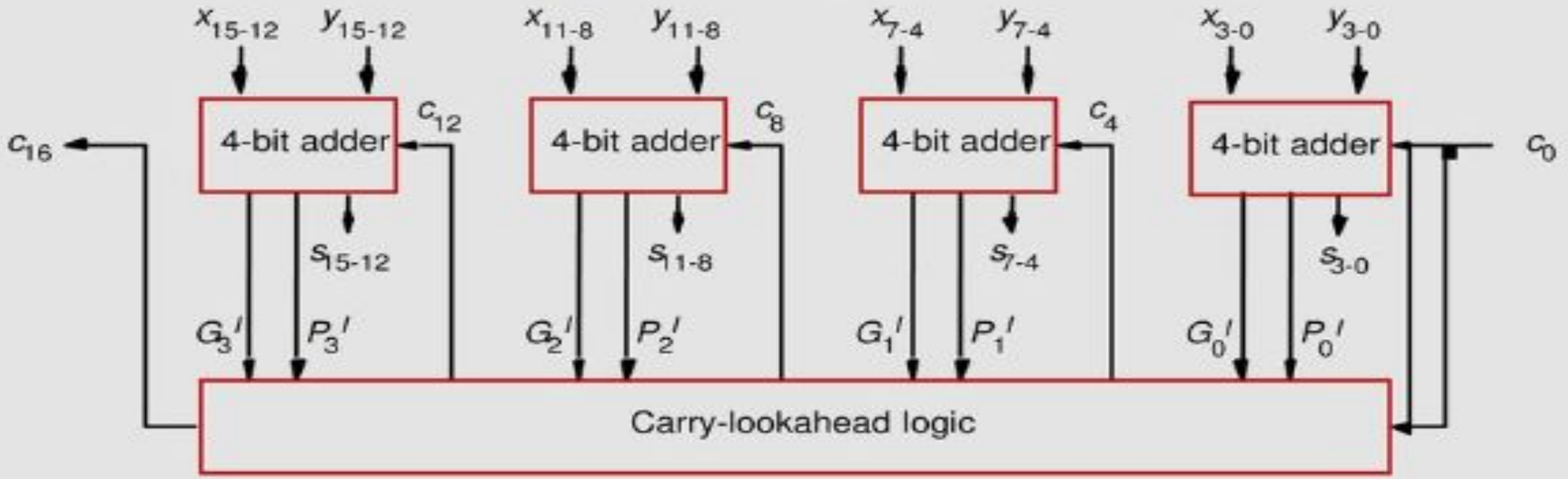
$$G_0^I = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$

Subscript I denotes the blocked carry lookahead and identifies the block.

Cascade 4 4-bit adders, c_{16} can be expressed as:

$$c_{16} = G_3^I + P_3^IG_2^I + P_3^IP_2^IG_1^I + P_3^IP_2^IP_1^0G_0^I + P_3^IP_2^IP_1^0P_0^0c_0$$

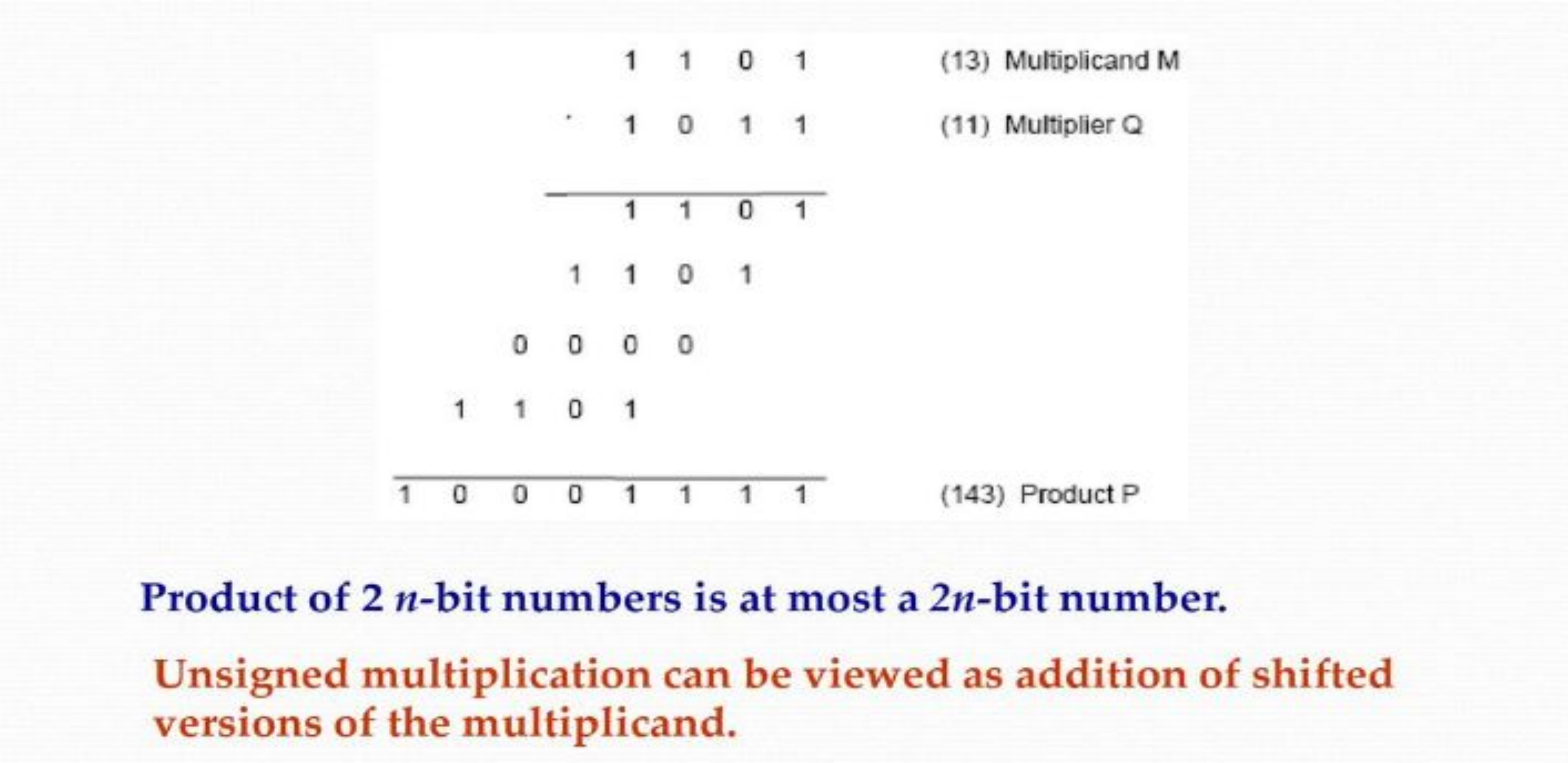
Blocked Carry-Lookahead adder



After x_i, y_i and c_0 are applied as inputs:

- G_i and P_i for each stage are available after 1 gate delay.
- P^I is available after 2 and G^I after 3 gate delays.
- All carries are available after 5 gate delays.
- c_{16} is available after 5 gate delays.
- s_{15} which depends on c_{12} is available after 8 (5+3) gate delays (Recall that for a 4-bit carry lookahead adder, the last sum bit is available 3 gate delays after all inputs are available)

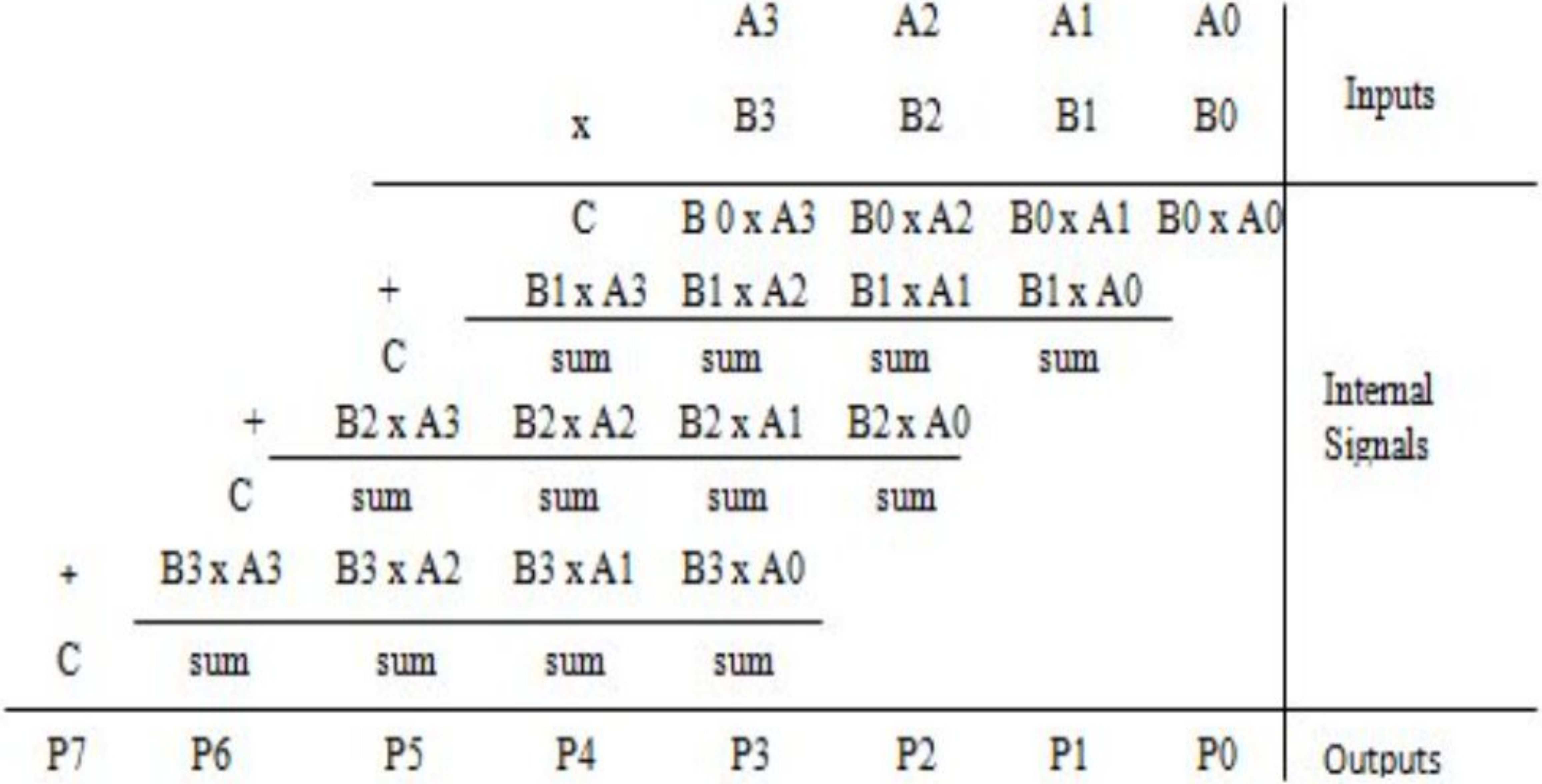
Multiplication of Positive Numbers



```
                1   1   0   1        (13)  Multiplicand M
                1   0   1   1        (11)  Multiplier Q
            -----------------
                1   1   0   1
            1   1   0   1
        0   0   0   0
    1   1   0   1
-----------------------------
1   0   0   0   1   1   1   1        (143)  Product P
```

Product of 2 *n*-bit numbers is at most a 2*n*-bit number.

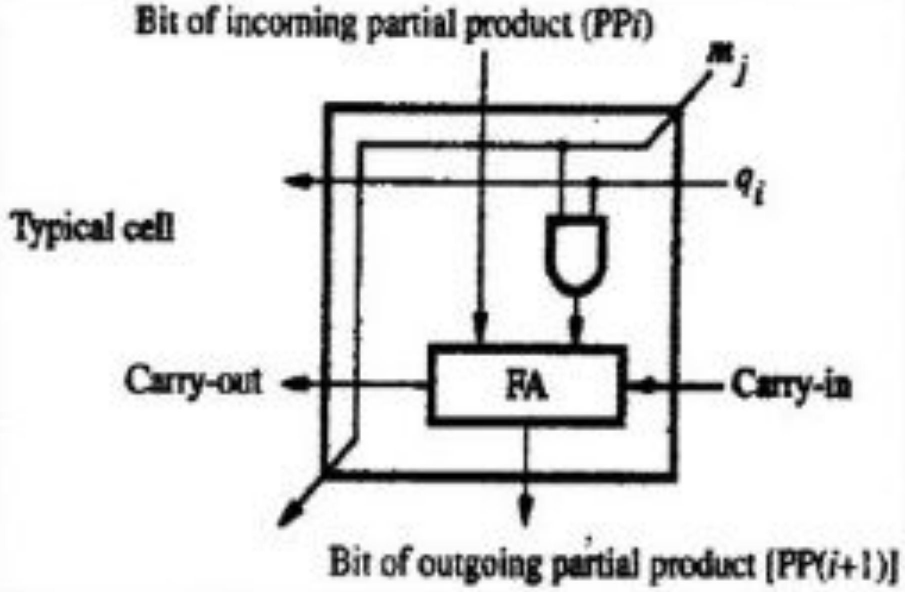
Unsigned multiplication can be viewed as addition of shifted versions of the multiplicand.

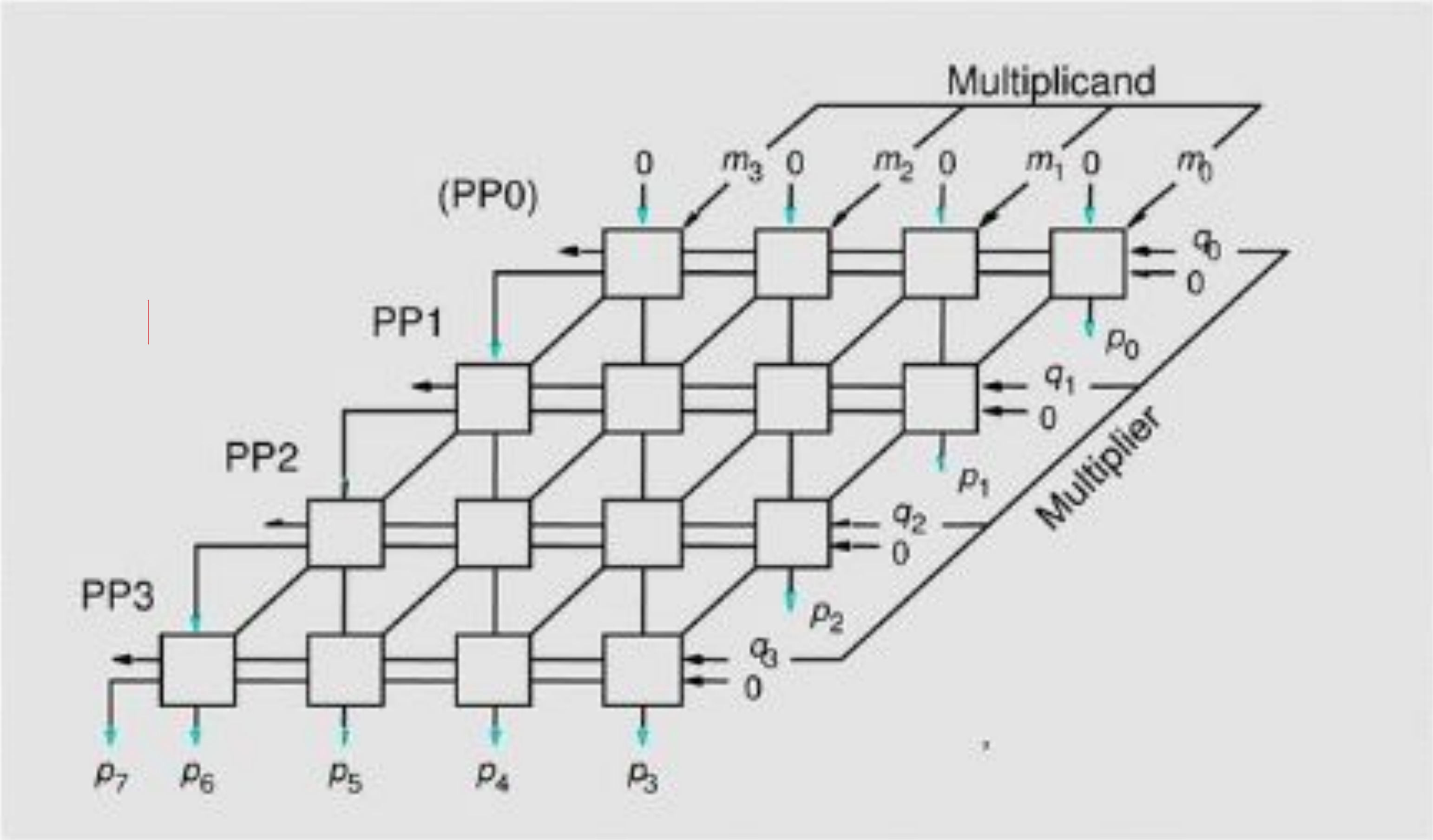


				A3	A2	A1	A0	Inputs
			x	B3	B2	B1	B0	
			C	B 0 x A3	B0 x A2	B0 x A1	B0 x A0	
		+	B1 x A3	B1 x A2	B1 x A1	B1 x A0		
		C	sum	sum	sum	sum		Internal Signals
	+	B2 x A3	B2 x A2	B2 x A1	B2 x A0			
	C	sum	sum	sum	sum			
+	B3 x A3	B3 x A2	B3 x A1	B3 x A0				
C	sum	sum	sum	sum				
P7	P6	P5	P4	P3	P2	P1	P0	Outputs

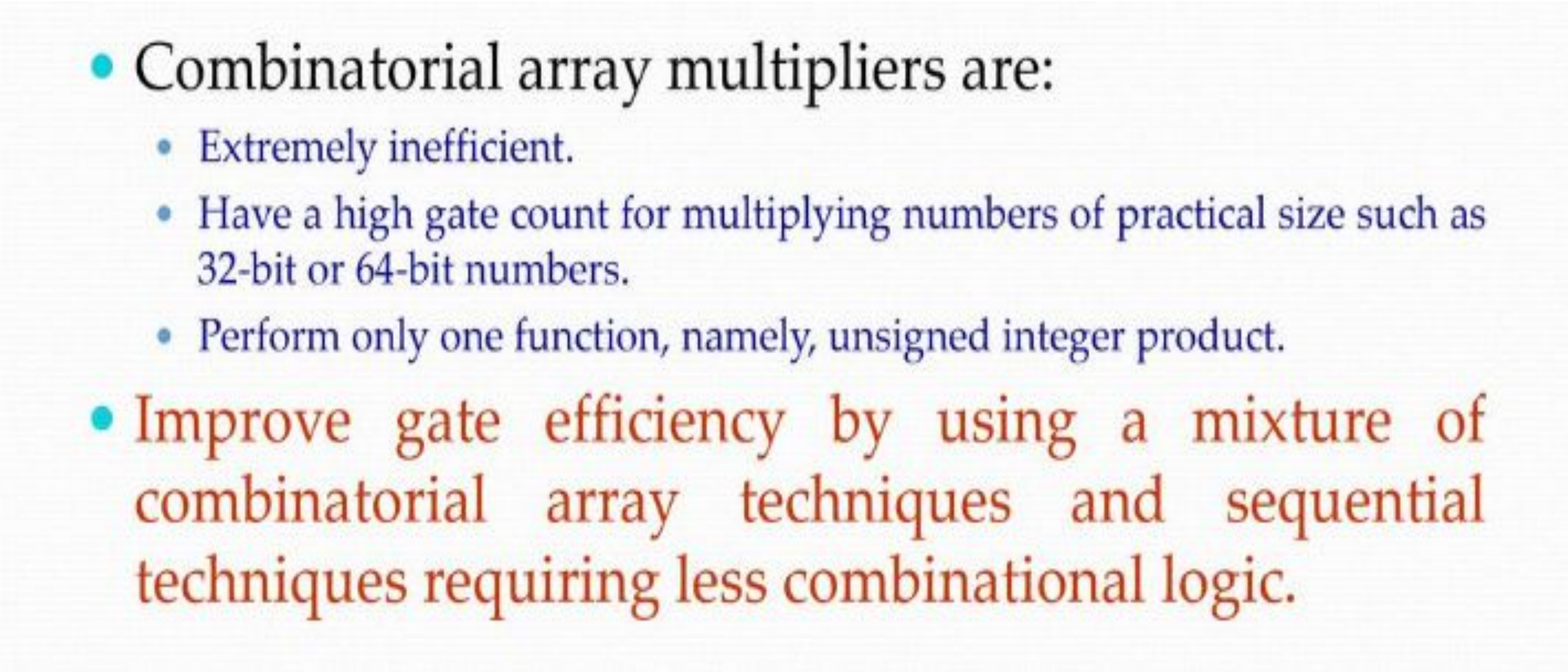
Combinatorial array multiplier

Combinatorial array multiplier


Bit of incoming partial product (PPi)
m_j
q_i
Typical cell
Carry-out
FA
Carry-in
Bit of outgoing partial product [PP(i+1)]

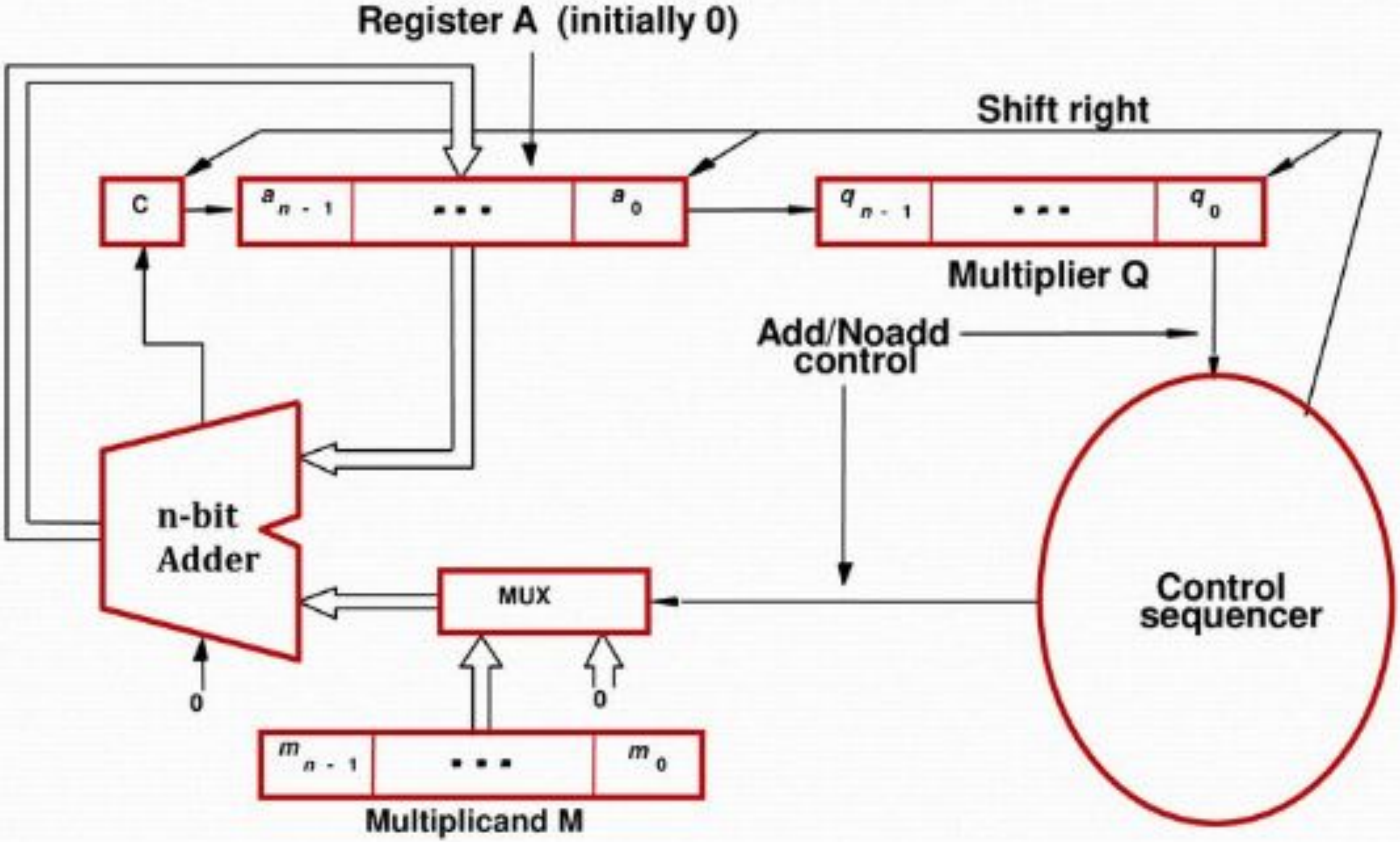


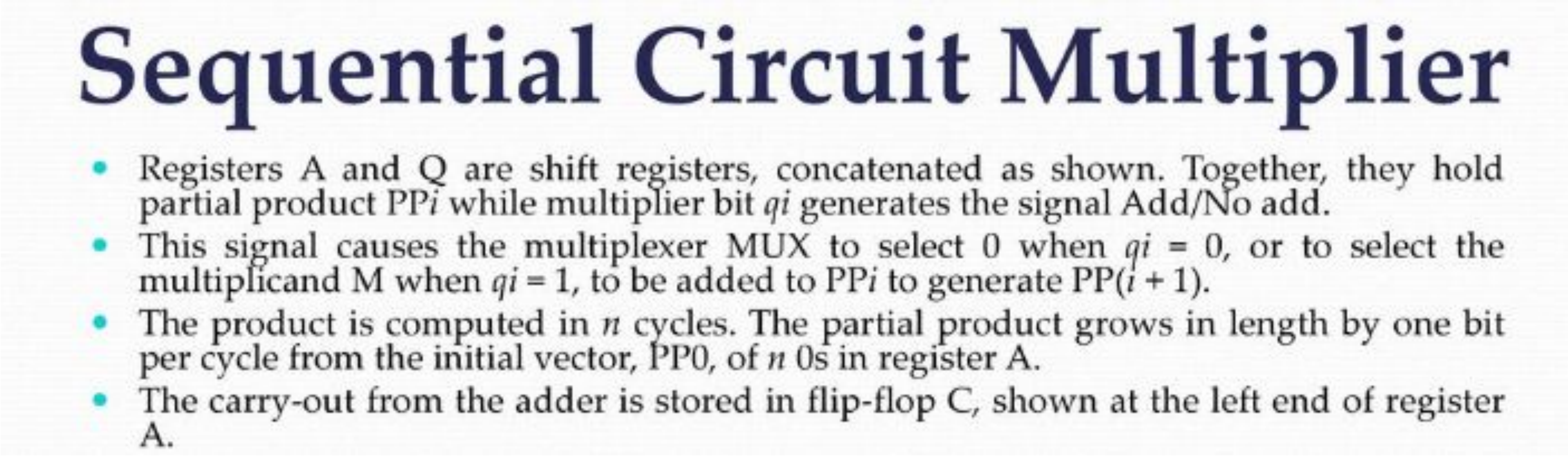
Product is: $p_7, p_6, ..p_0$



- Combinatorial array multipliers are:
  - Extremely inefficient.
  - Have a high gate count for multiplying numbers of practical size such as 32-bit or 64-bit numbers.
  - Perform only one function, namely, unsigned integer product.
- Improve gate efficiency by using a mixture of combinatorial array techniques and sequential techniques requiring less combinational logic.

Sequential Circuit Multiplier





Sequential Circuit Multiplier

- Registers A and Q are shift registers, concatenated as shown. Together, they hold partial product PP*i* while multiplier bit *qi* generates the signal Add/No add.
- This signal causes the multiplexer MUX to select 0 when *qi* = 0, or to select the multiplicand M when *qi* = 1, to be added to PP*i* to generate PP(*i* + 1).
- The product is computed in *n* cycles. The partial product grows in length by one bit per cycle from the initial vector, PP0, of *n* 0s in register A.
- The carry-out from the adder is stored in flip-flop C, shown at the left end of register A.

Sequential Multiplication Algorithm

1. Initialize 2n bit Registers for two n bit unsigned multiplication register M with multiplicand , register Q with multiplier, register A with Zero (0) and carry flag with Zero (0).
2. If LSB of register Q i.e., q0 = 1, then ADD A = A + M and retain Q otherwise if q0 = 0 then NO ADD.
3. Right Shift 1 bit from Carry Flag (C) to Register Q.
4. Repeat Step 2 and 3 n times.

- Refer textbook and class notes for Multiplication Problems and Division Problems

Basic Processing Unit

- In this chapter we will study about,

□ How a processor executes instructions.

□ Internal functional units of a processor and how they are interconnected

Some Fundamental concepts

- To execute a program , the processor fetches one instruction at a time and performs the operations specified.
- Instructions are fetched from successive memory locations until a branch or a jump instructions is encountered.
- The processor keeps track of the address of the memory location containing the next instruction to be fetched using the program counter, PC.
- After fetching an instruction, the contents of the PC are updated to point to the next instruction in the sequence.
- A branch instruction load a different value into the PC.

- To execute an instructions , the processor has to perform the following three steps:

1. Fetch the contents of memory location pointed to by the PC. The contents of this location are interpreted as an instructions to be executed. Hence, they are loaded into IR.

   IR ← [[PC]]

2. Assuming that the memory is byte addressable, increment the contents of the PC by 4 i.e,

   PC ← [PC] + 4

3. Carry out the actions specified by the instructions in the IR.

Step1 and step2 are Fetch Phase and Step 3 is of execution phase.

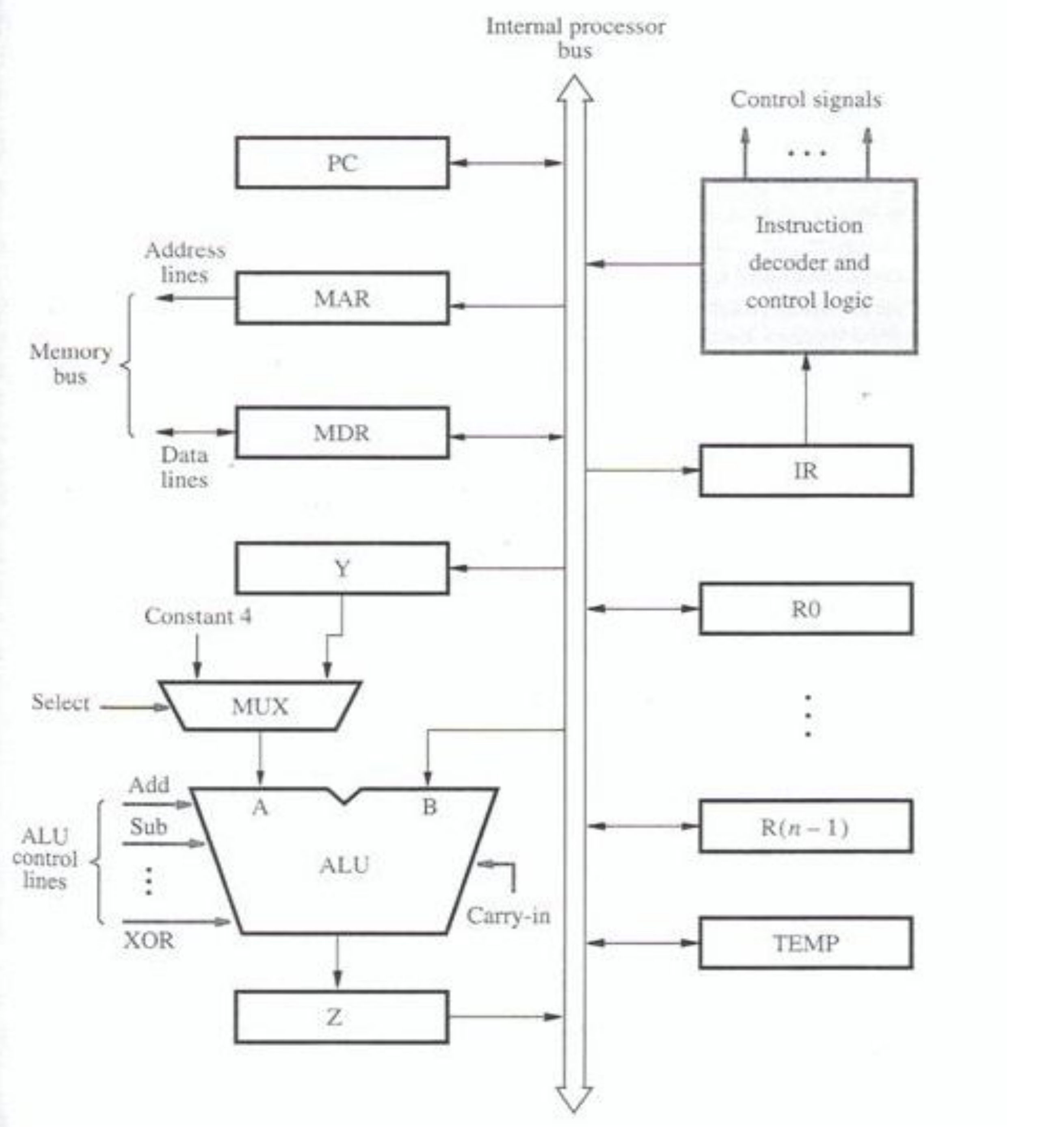
- The operations specified by an instruction can be carried out by performing one or more of the following actions.

1. Read the contents of a given memory location and load them into a register.
2. Read data from one or more registers.
3. Perform an arithmetic or logic operation and place the result into a register.
4. Store data from a register into a given memory location

Single Bus Organizations

- ALU and all the registers are interconnected via a single Common bus.
- This bus is internal to the processor.
- Data & address lines of the external memory bus is connected to the internal processor bus via MDR and MAR.
- MDR has 2 inputs and 2 outputs. Data may be loaded

1. Either from memory bus(external).
2. From processor – bus (internal)



- MAR’s input is connected to internal bus and MAR’s output is connected to external bus.
- Instruction Decoder and Control unit is responsible for issuing the control signals to all the units inside the processor.
- Implementing the actions specified by the instruction (loaded in the IR).
- Register R0 through R(n-1) are the processor registers.
- The programmer can access these registers for general purpose use.
- Only processor can access 3 registers, Y, Z & Temp for temporary storage during program execution. Programmer cannot access these 3 registers.

1. ALU
2. MUX

In ALU,

1. A input gets the operand from the output of the multiplexer(MUX).
2. B input gets the operand directly from the processor bus

MUX is used to select one of the 2 inputs.

MUX selects either

- Output of Y or
- Constant value 4 (which is used to increment PC content).

Execution of Instruction

- Transfer a word of data from one processor register to another or to the ALU.
- Perform an arithmetic or a logic operation and store the result in a processor register.
- Fetch the contents of a given memory location and load them into a processor register.
- Store a word of data from a processor register into a given memory location.

Register Transfers

- Instruction execution involves a sequence of steps in which data are transferred from one register to another.
- For each register , two control-signals are used:

Rin=1 -- data on the bus is loaded into the register Ri.

Rout=1 Contents of Ri is placed on the bus.

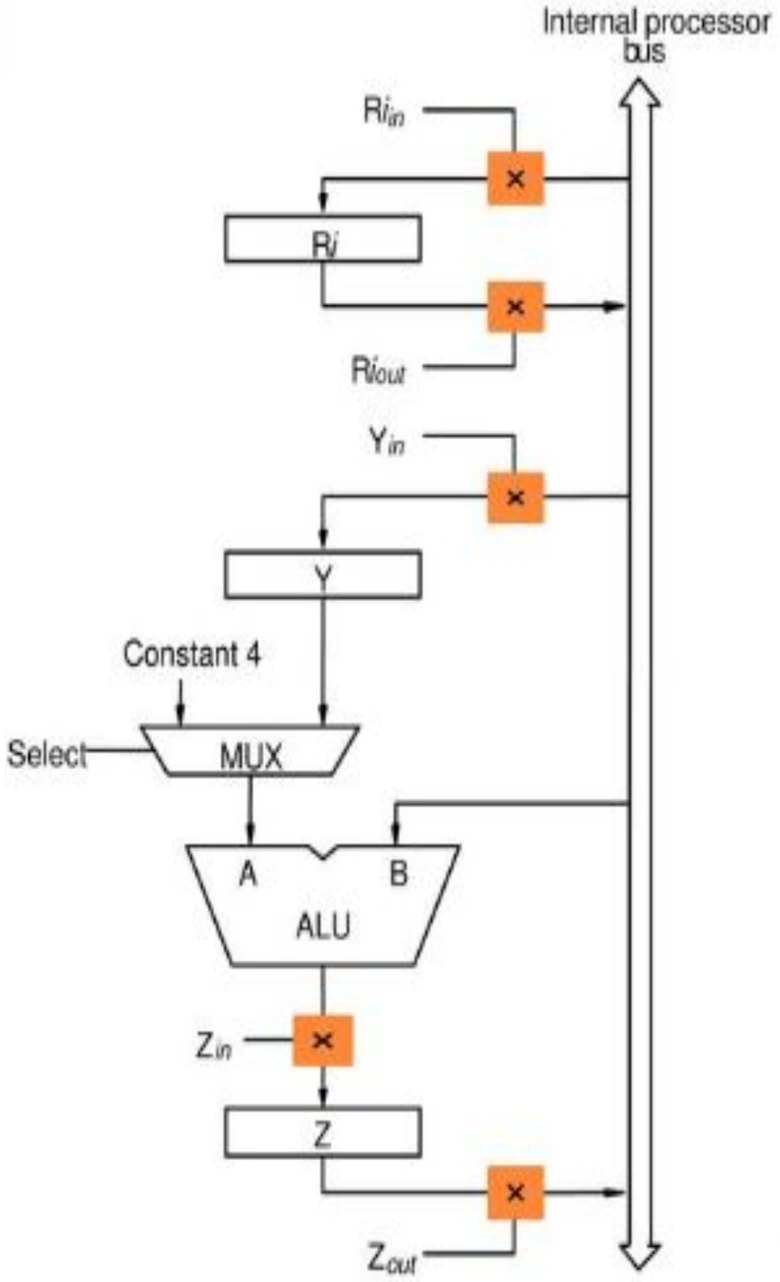
Example:

MOV R1,R4

This transfer the contents of register R1 to register R4.

Enables the output register R1 by setting R1out to 1, this place contents of R1 on the processor bus.

Enable the input of register R4 by setting R4in to 1. this loads the data from the processor bus into register R4



Performing an Arithmetic or Logic Operation

- ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z.
- First operand is always send to Register Y and Second is directly send to point B of ALU
- Consider an example , ADD R1,R2,R3

Sequence of operations to add the contents of register R1 to R2 and store in R3.

1. R1out, Yin
2. R2out, Select Y, ADD, Zin
3. Zout, R3in

Step 1: Output of the register R1 and input of the register Y are enabled, causing the contents of R1 to be transferred to Y.

Step 2: The multiplexer’s select signal is set to select Y causing the multiplexer to gate the contents of register Y to input A of the ALU.

Step 3: The contents of Z are transferred to the destination register R3.

Fetching a word from memory

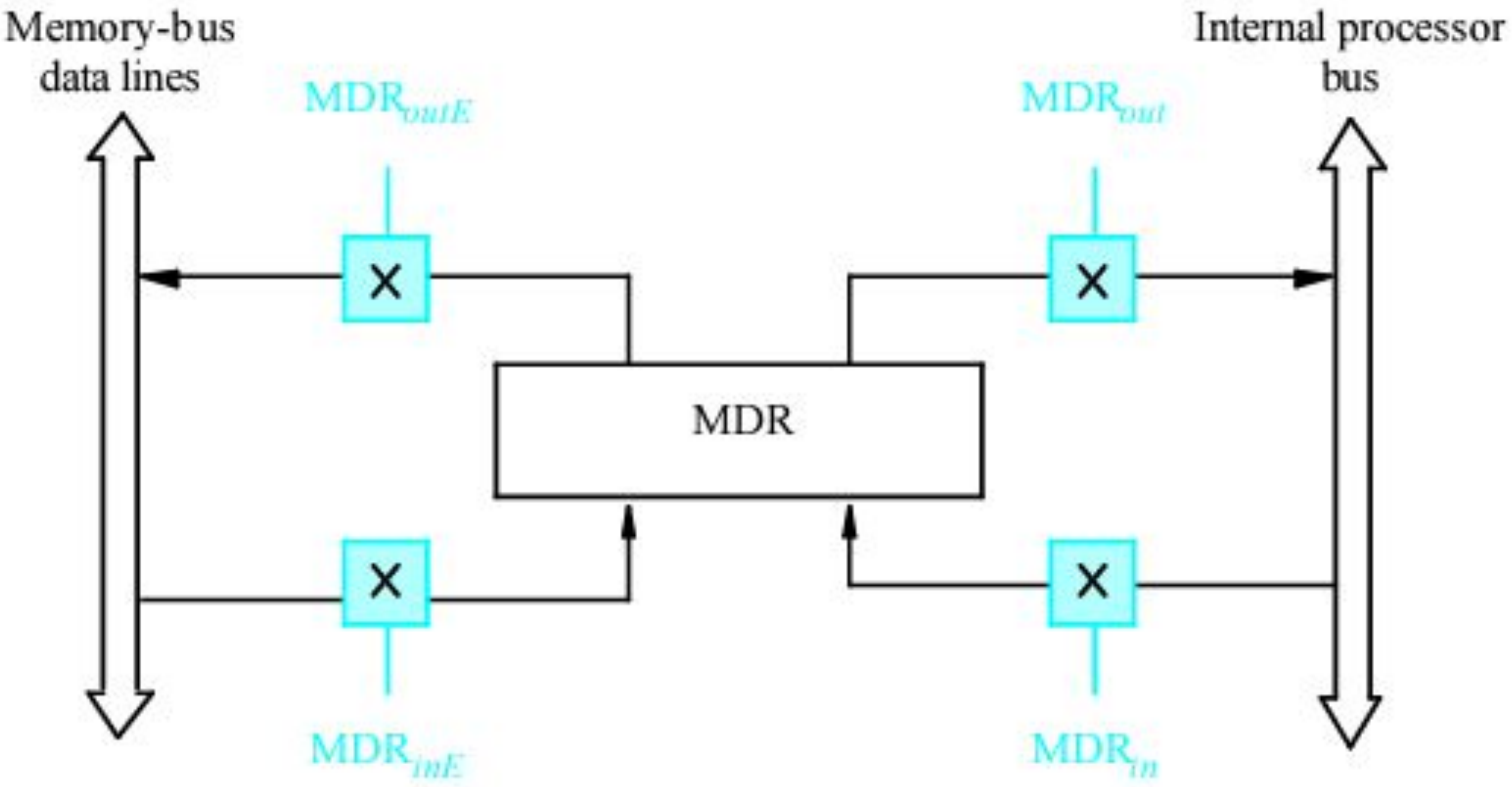
- To fetch a word of information from memory , the processor has to specify the address of the memory location where this information is stored and request a Read operation.

- The processor transfers the required address to the MAR, whose output is connected to the address lines of the memory bus.

- The processor uses the control lines of the memory bus to indicate that Read operation is needed.

- When the requested data are received from the memory they are stored in register MDR, from where they can be transferred to other registers in the processor.

The connection for register MDR are illustrated as



- The response time of each memory access varies (cache miss, memory-mapped I/O,…).
- To accommodate this, the processor waits until it receives an indication that the requested operation has been completed (Memory-Function-Completed, MFC).
- The addressed device sets this signal to 1 to indicate that the contents of the specified location have been read and are available on the data lines of the memory bus.

- Consider the instruction Move (R1),R2.
- The action needed to execute this instruction are

1. MAR <------ [R1]
2. Start a Read operation on the memory bus
3. Wait for the MFC response from the memory
4. Load MDR from the Memory bus
5. R2 < ------ [MDR]

The Actions can be carried out as separate steps, but some can be combined into a single steps, each actions can be completed in one clock cycle , except action 3 which requires one or more clock cycles, depending on speed of addressed devices.

- Memory read operation requires 3 steps , which can be described by the signals being activated as follows:

1. R1out, MARin, Read
2. MRDinE, WMFC
3. MDRout, R2in

Where, WMFC is the control signal that causes the processor’s control circuitry to wait for the arrival of the MFC signals.

Storing a Word in Memory

- Writing a word into a memory location follows a similar procedure.
- The desired address is loaded into MAR. then the data to be loaded into MDR, and a write command is issued.
- Consider an example

Move R2, (R1)

The execution of above instruction requires the following sequence

1. R1out, MARin
2. R2out, MDRin, Write
3. MDRoutE, WMFC

Execution of a Complete Instruction

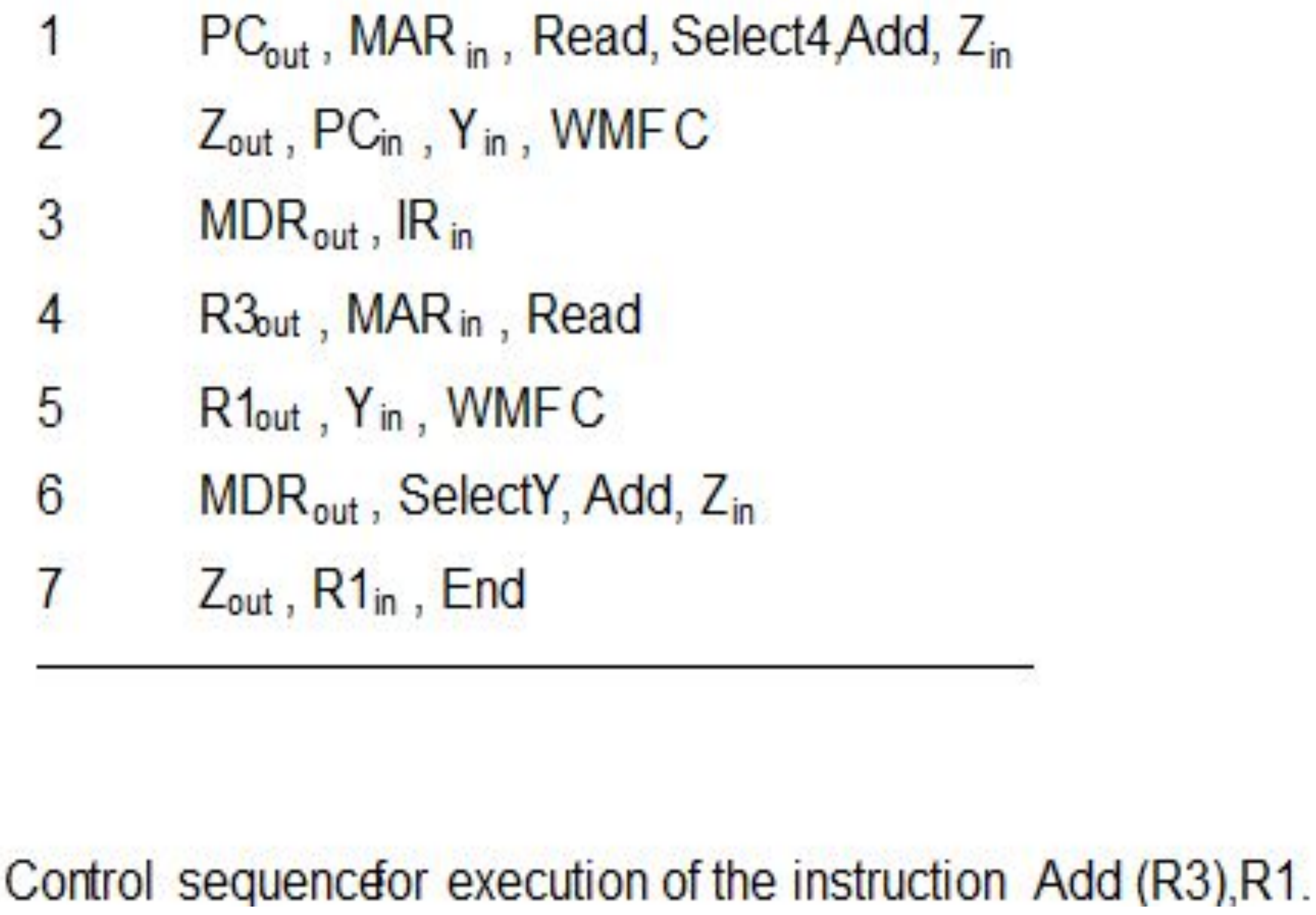
Consider the instruction

Add (R3), R1

This instruction adds the contents of the memory location pointed by R3 to R1. this requires the following actions :

- Fetch the instruction.
- Fetch the first operand (the contents of the memory location Stored at R3)
- Perform the addition.
- Load the result into R1.

Add (R3), R1



1 PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in}

2 Z_{out} , PC_{in} , Y_{in} , WMFC

3 MDR_{out} , IR_{in}

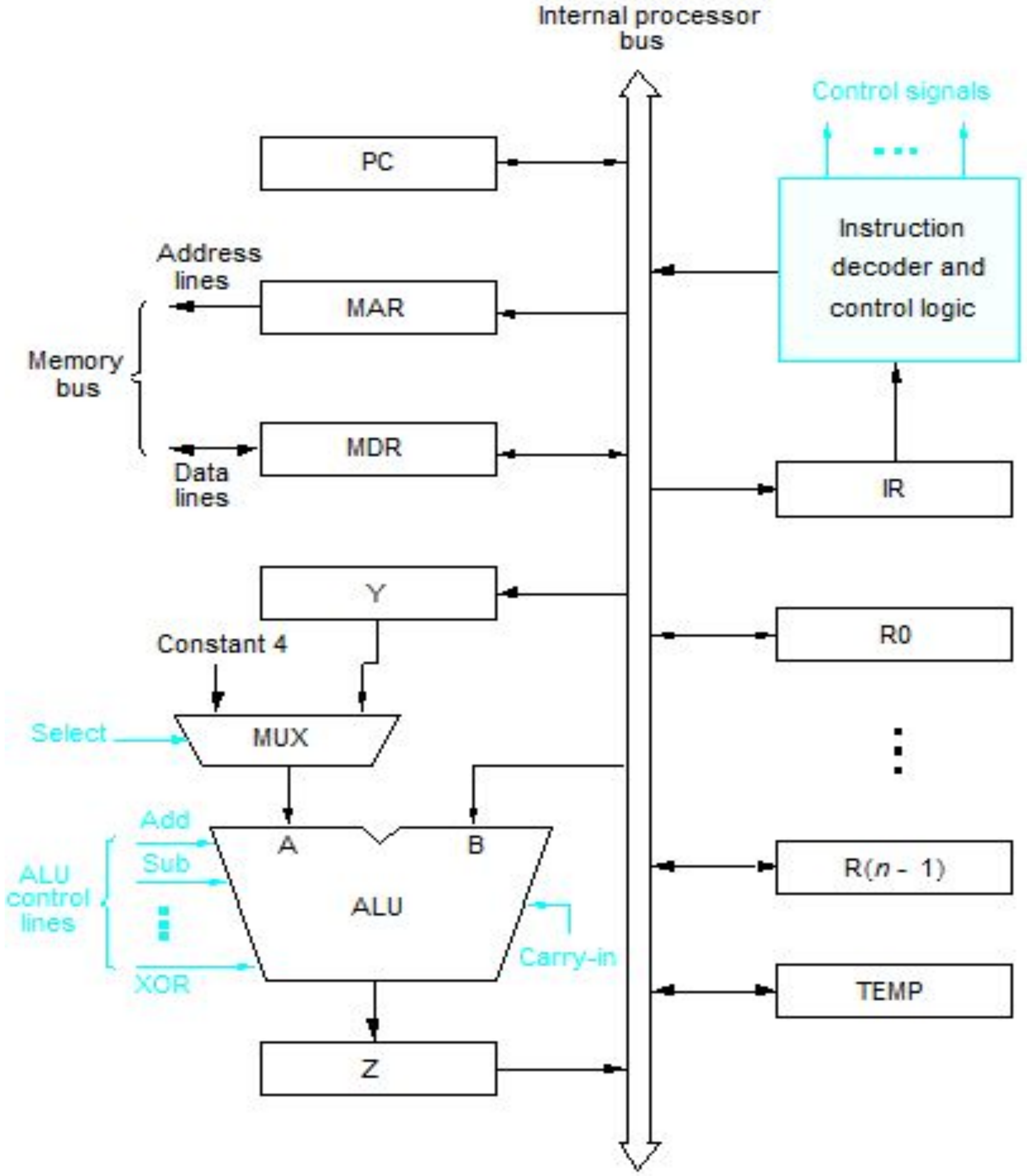
4 $R3_{out}$, MAR_{in} , Read

5 $R1_{out}$, Y_{in} , WMFC

6 MDR_{out} , SelectY, Add, Z_{in}

7 Z_{out} , $R1_{in}$, End

Control sequencefor execution of the instruction Add (R3),R1.



Execution of Branch Instructions

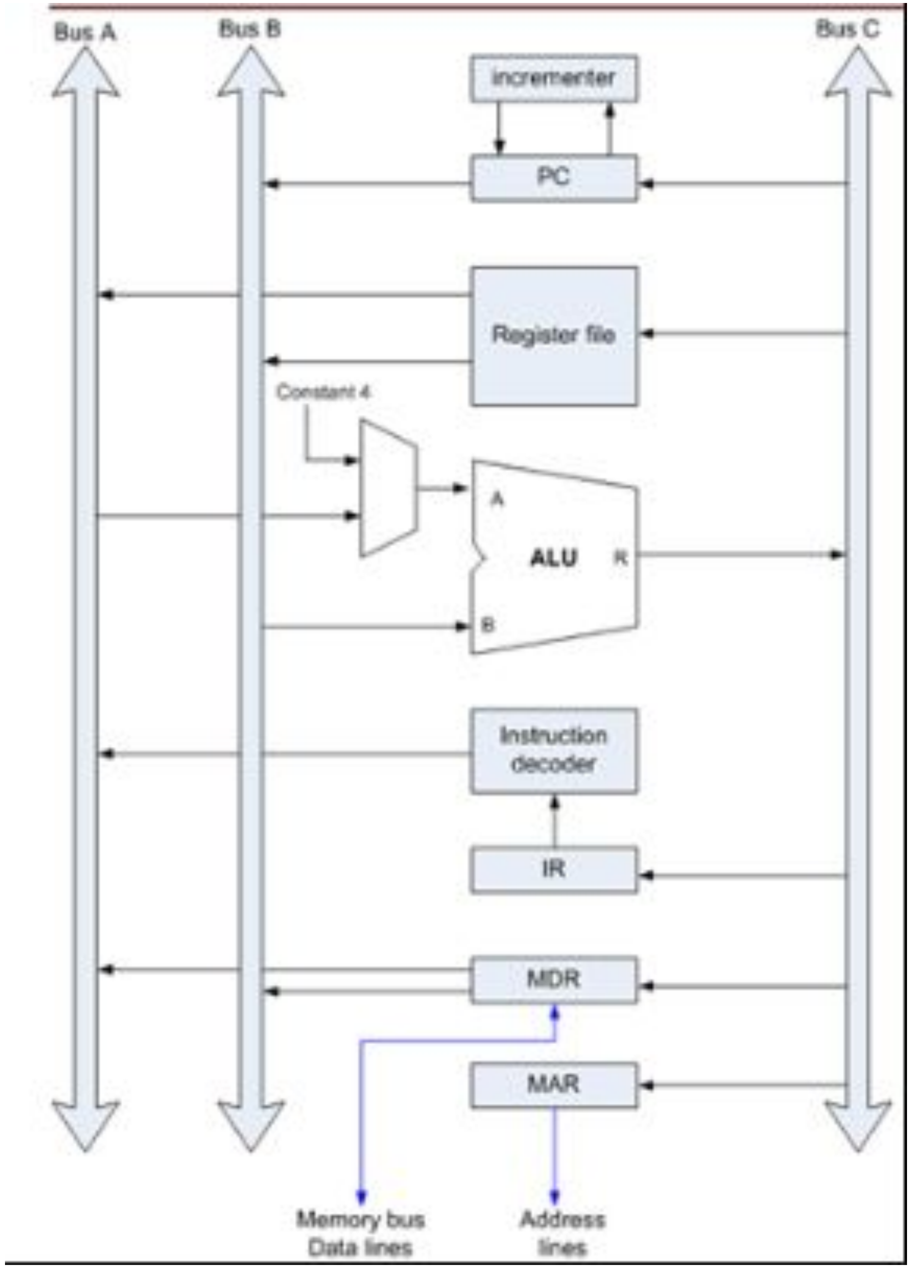
- A branch instruction replaces the contents of PC with the branch target address, which is usually obtained by adding an offset X given in the branch instruction.
- The offset X is usually the difference between the branch target address and the address immediately following the branch instruction.

Execution of Branch Instructions

Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in}, End

Figure Control sequence for an unconditional branch instruction.

Multiple-Bus Organization



Multiple-Bus Organization

- Add R1, R2, R3

Step	**Action**
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R1_{outA}$, $R2_{outB}$, SelectA, Add, $R3_{in}$, End

Figure:Control sequence for the instruction. Add R1,R2,R3,
for the three-bus organization