Advanced Computer Architecture
(0630561)

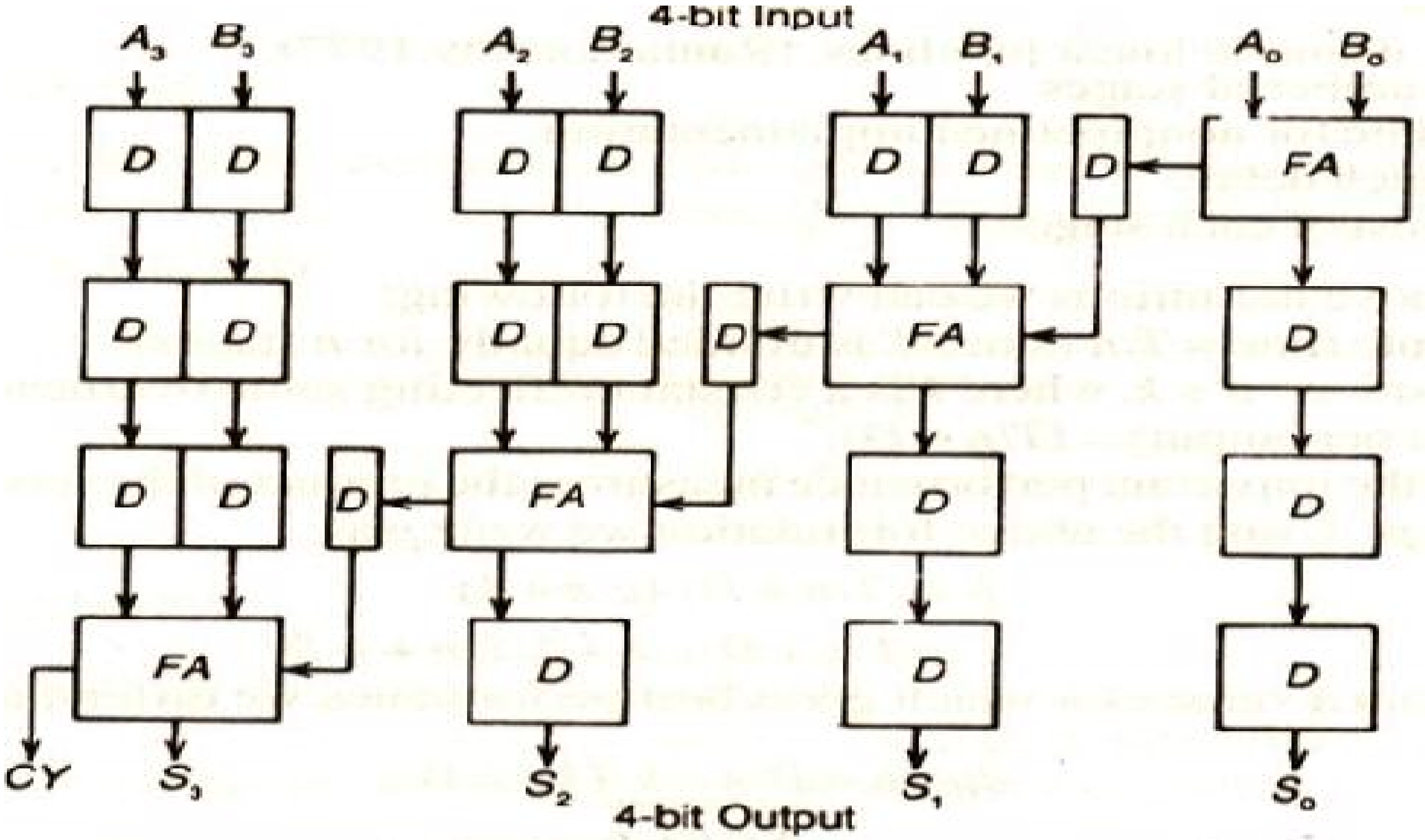
Lecture 5

Arithmetic Pipelining

Prof. Kasim M. Al-Aubidy
Computer Eng. Dept.

Arithmetic Pipelines:

Fixed Point Addition Pipeline:



D = D flip-flops, which are all clocked by a common clock
FA = Full Adder

Arithmetic Pipeline

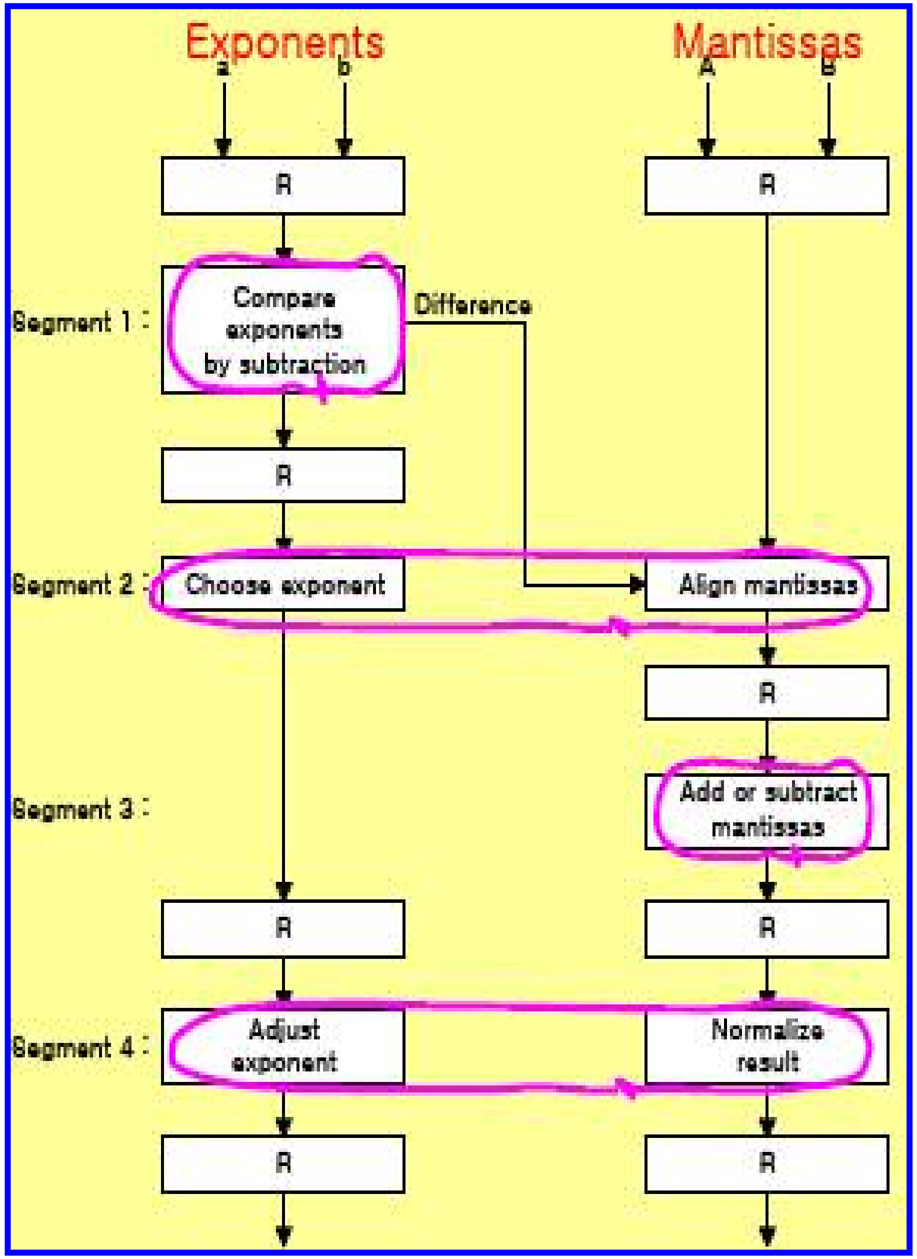
Floating-point Adder Pipeline Example:

Add / Subtract two normalized fp binary number

» $X = A \times 2^a = 0.9504 \times 10^3$

» $Y = B \times 2^b = 0.8200 \times 10^2$

- 4 segments suboperations
  - » 1) Compare exponents by subtraction:
    - 3 - 2 = 1
    - ▪ $X = 0.9504 \times 10^3$
    - ▪ $Y = 0.8200 \times 10^2$
  - » 2) Align mantissas
    - ▪ $X = 0.9504 \times 10^3$
    - ▪ $Y = 0.08200 \times 10^3$
  - » 3) Add mantissas
    - ▪ $Z = 1.0324 \times 10^3$
  - » 4) Normalize result
    - ▪ $Z = 0.1324 \times 10^4$

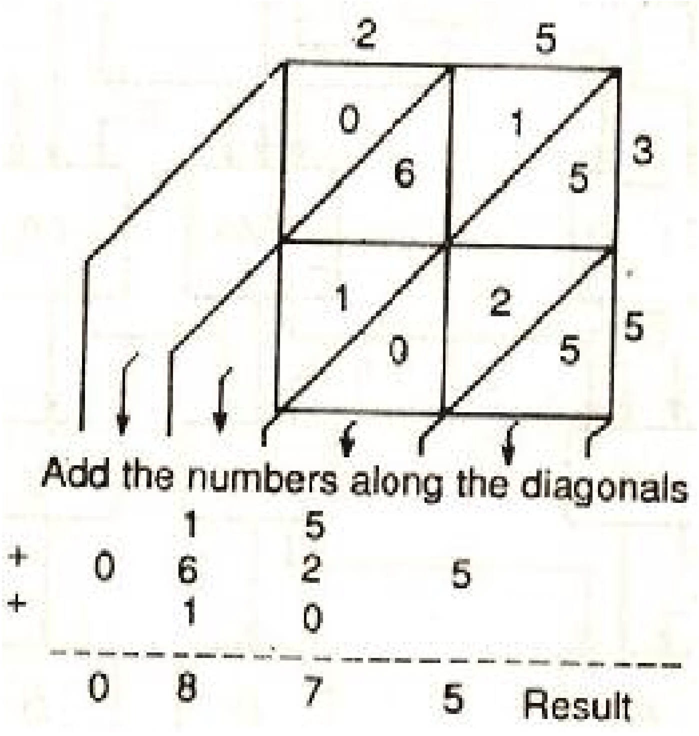


Fixed Point Multiplication Pipeline:

- A pipelined multiplier based on the digit products can be designed using digit product generation logic and the digit adders.

Example:

25 * 35 = 875



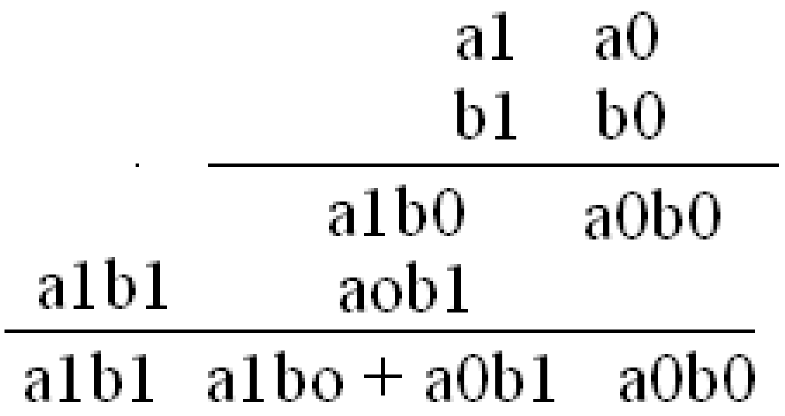
Add the numbers along the diagonals

```
        1    5
+   0   6    2      5
+       1    0
-------------------------
    0   8    7      5   Result
```

Now for binary multiplication:

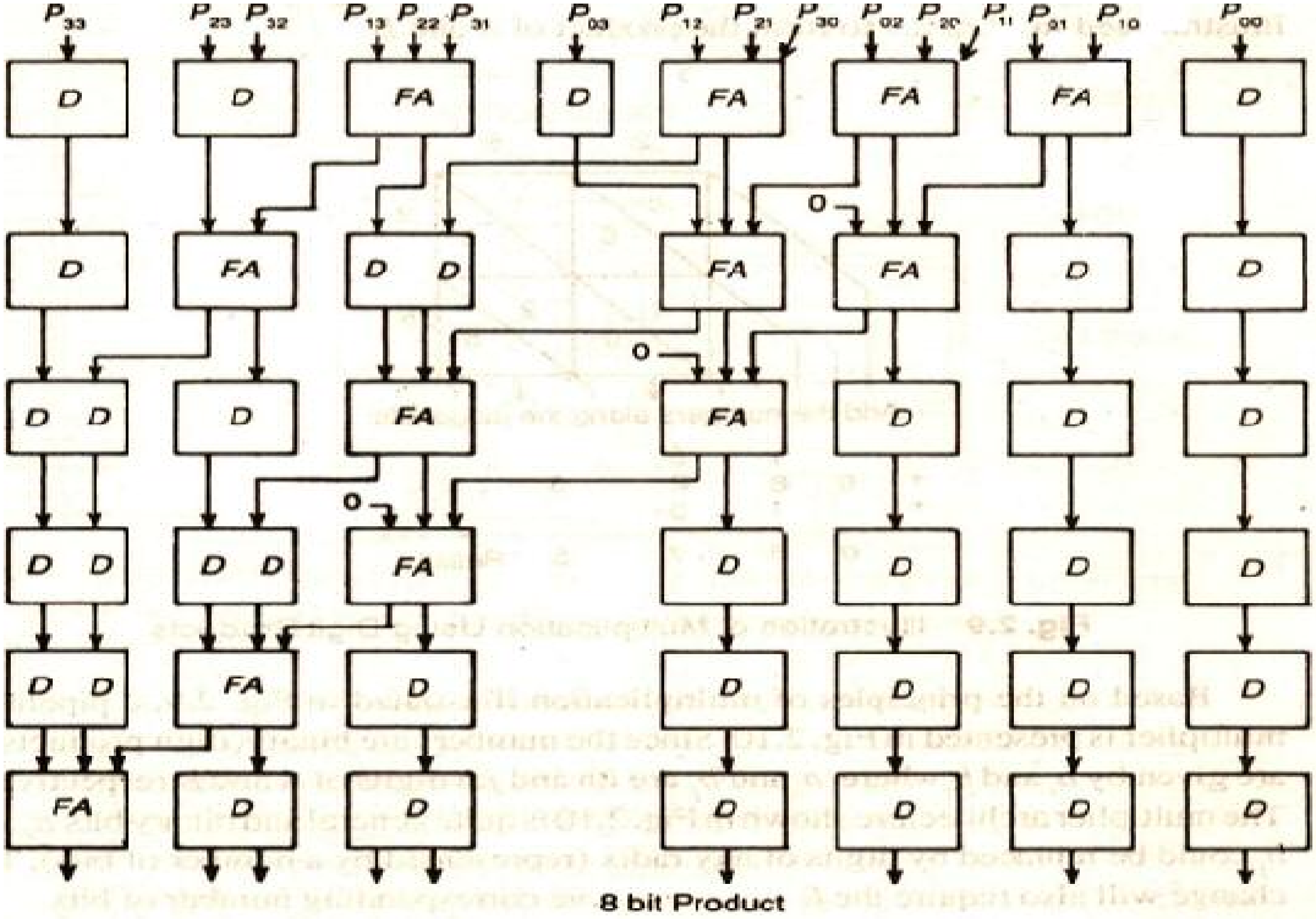
A = a1 a0

B = b1 b0



```
                 a1      a0
                 b1      b0
           ----------------------
             a1b0        a0b0
 a1b1        aob1
----------------------------------
 a1b1    a1bo + a0b1     a0b0
```

Multiplier Based on Digit Products:



Floating Point Multiplication Pipeline:

FP multiplication involves the following three major steps:

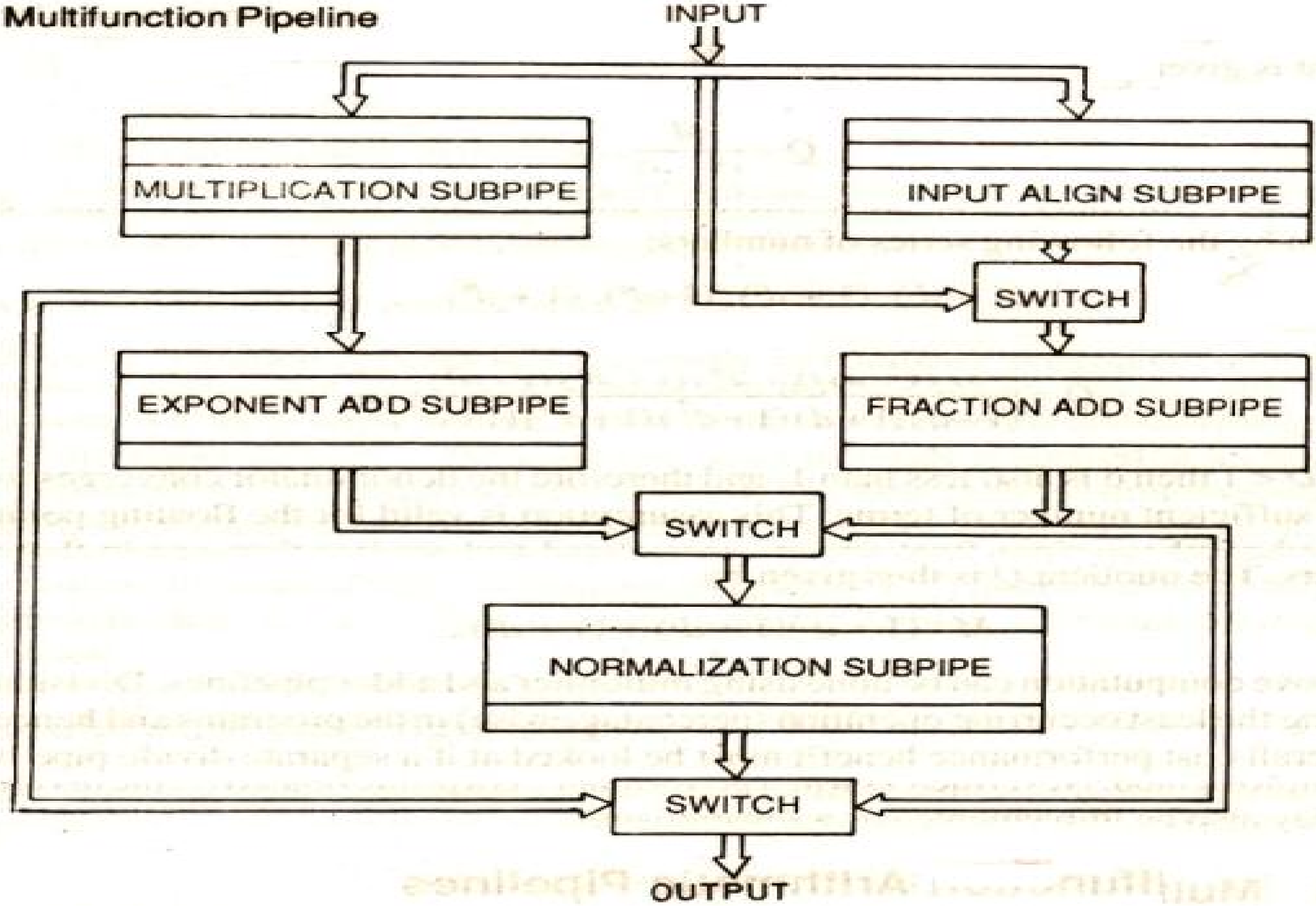
1. Multiplication of fractions.
2. Addition of exponents.
3. Normalization of the result.

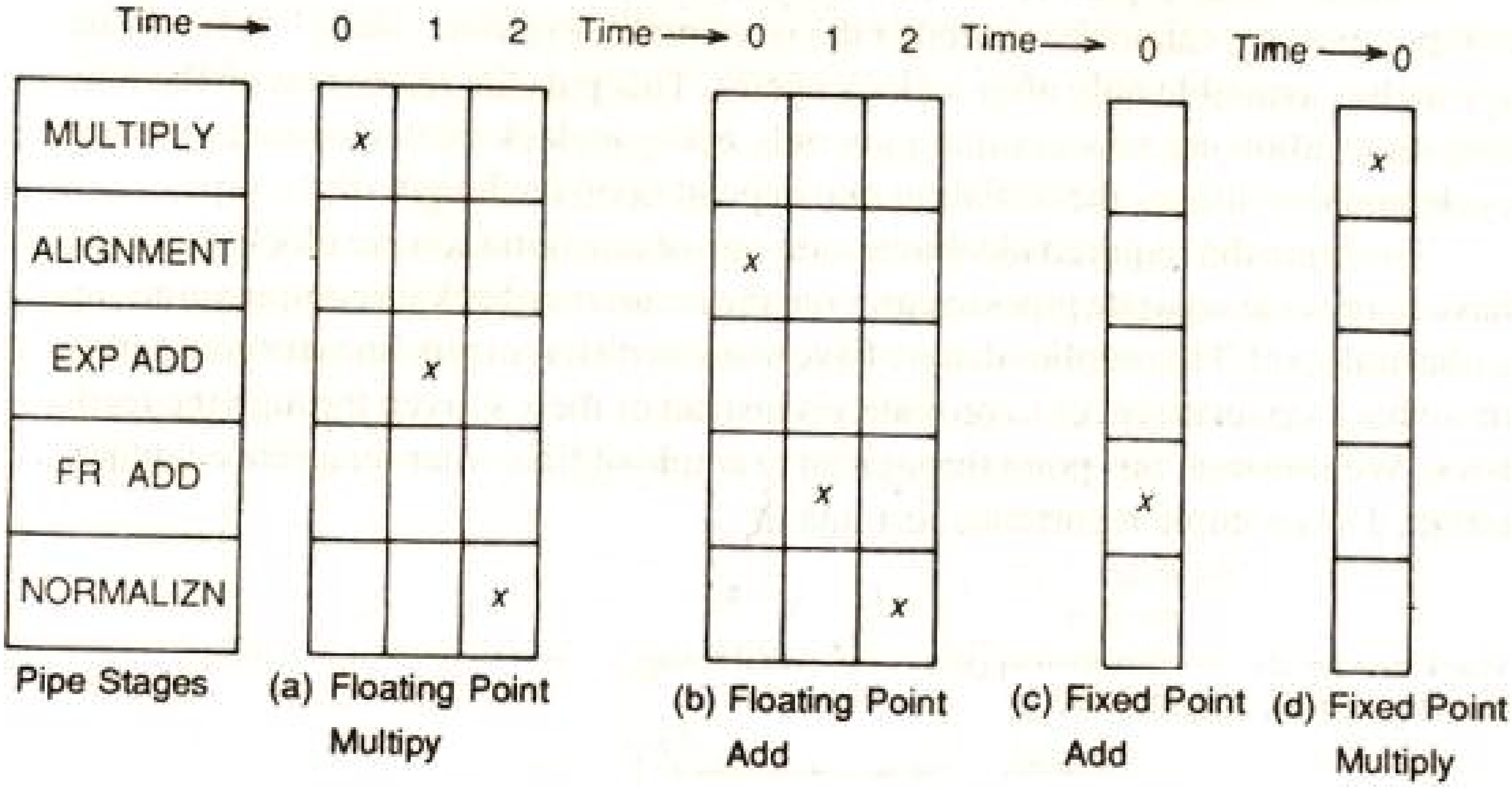
- Since fractions and exponents are fixed-point numbers, the steps 1 & 2 can be implemented using the principles discussed before. Normalization step can be implemented as given in the floating point addition.

Floating Point Division Pipeline:

- Division operation appears less frequently in computer programs compared to addition subtraction and multiplication and hence separate pipeline unit for the division is seldom implemented. It is common to schedule the division using adder and multiplier pipelines.

Multifunction Pipeline





(a) Floating Point Multipy

Pipe Stages	0	1	2
MULTIPLY	x		
ALIGNMENT			
EXP ADD		x	
FR ADD			
NORMALIZN			x

(b) Floating Point Add

Pipe Stages	0	1	2
MULTIPLY			
ALIGNMENT	x		
EXP ADD			
FR ADD		x	
NORMALIZN			x

(c) Fixed Point Add

Pipe Stages	0
MULTIPLY	
ALIGNMENT	
EXP ADD	
FR ADD	x
NORMALIZN	

(d) Fixed Point Multiply

Pipe Stages	0
MULTIPLY	x
ALIGNMENT	
EXP ADD	
FR ADD	
NORMALIZN	

Figure shows four reservation tables for four operations.

Recurrence Computations:

- The recurrence formula expresses how to compute a sequence of numbers (Vector X) from another sequence of numbers (Vector A). A pipeline to compute the vector X should ideally take one element of A and produce one element of X, as illustrated in this example:

$$X_i = a_i + X_{i-1}$$

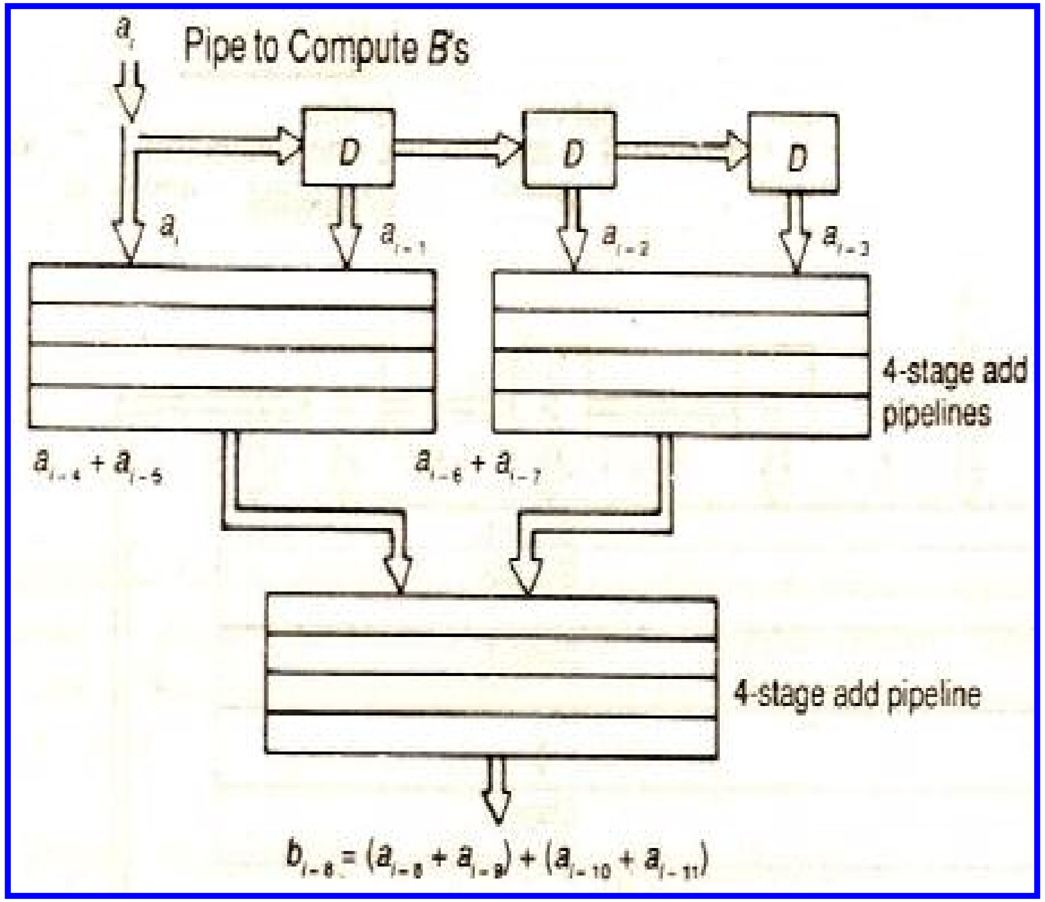
$$X_{i-1} = a_{i-1} + X_{i-2}$$

$$X_i = a_i + a_{i-1} + X_{i-2}$$

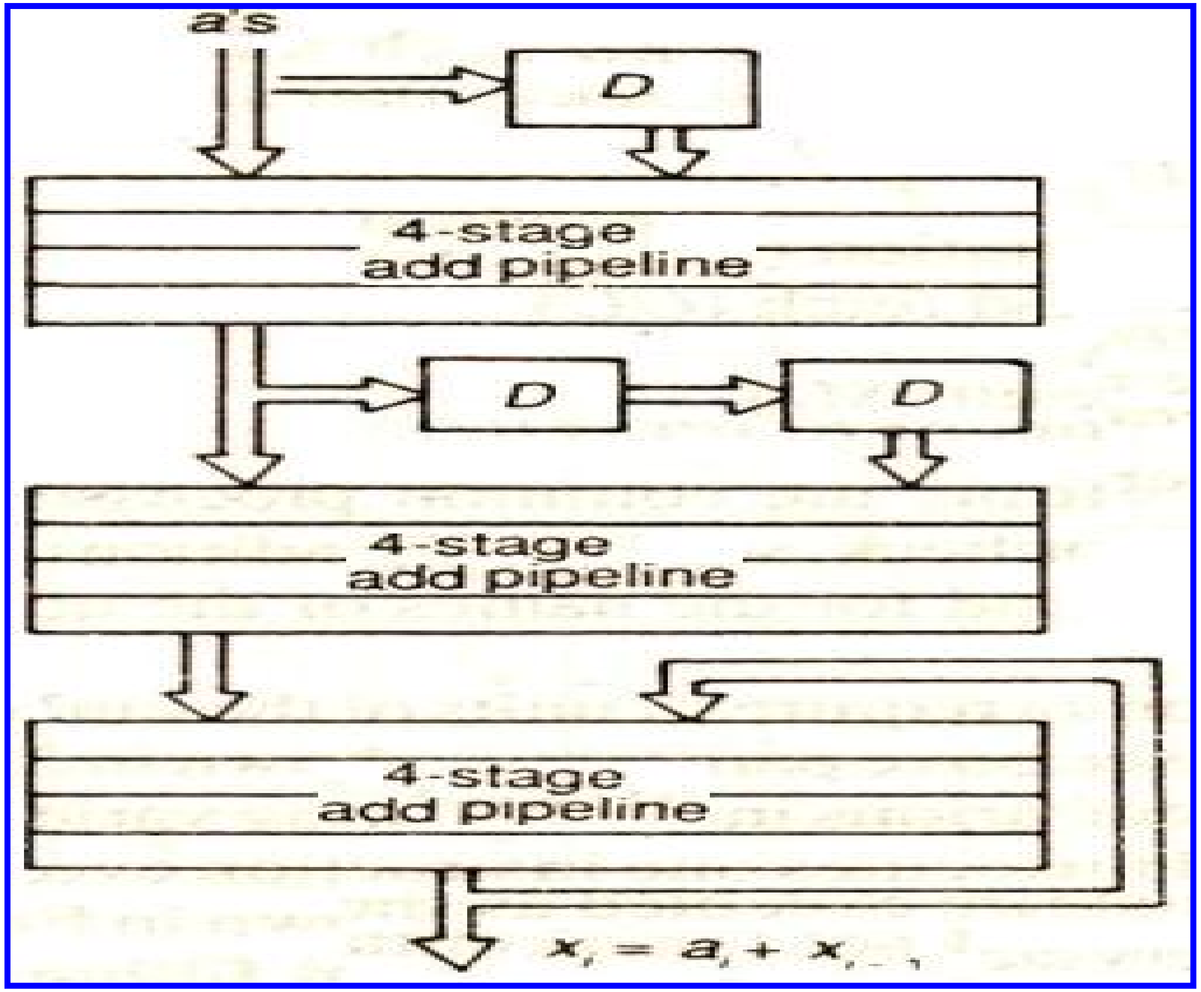
$$X_i = a_i + a_{i-1} + a_{i-2} + a_{i-3} + X_{i-4}$$

$$X_i = b_i + X_{i-4}$$

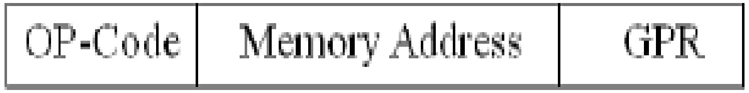
$$b_i = a_i + a_{i-1} + a_{i-2} + a_{i-3}$$



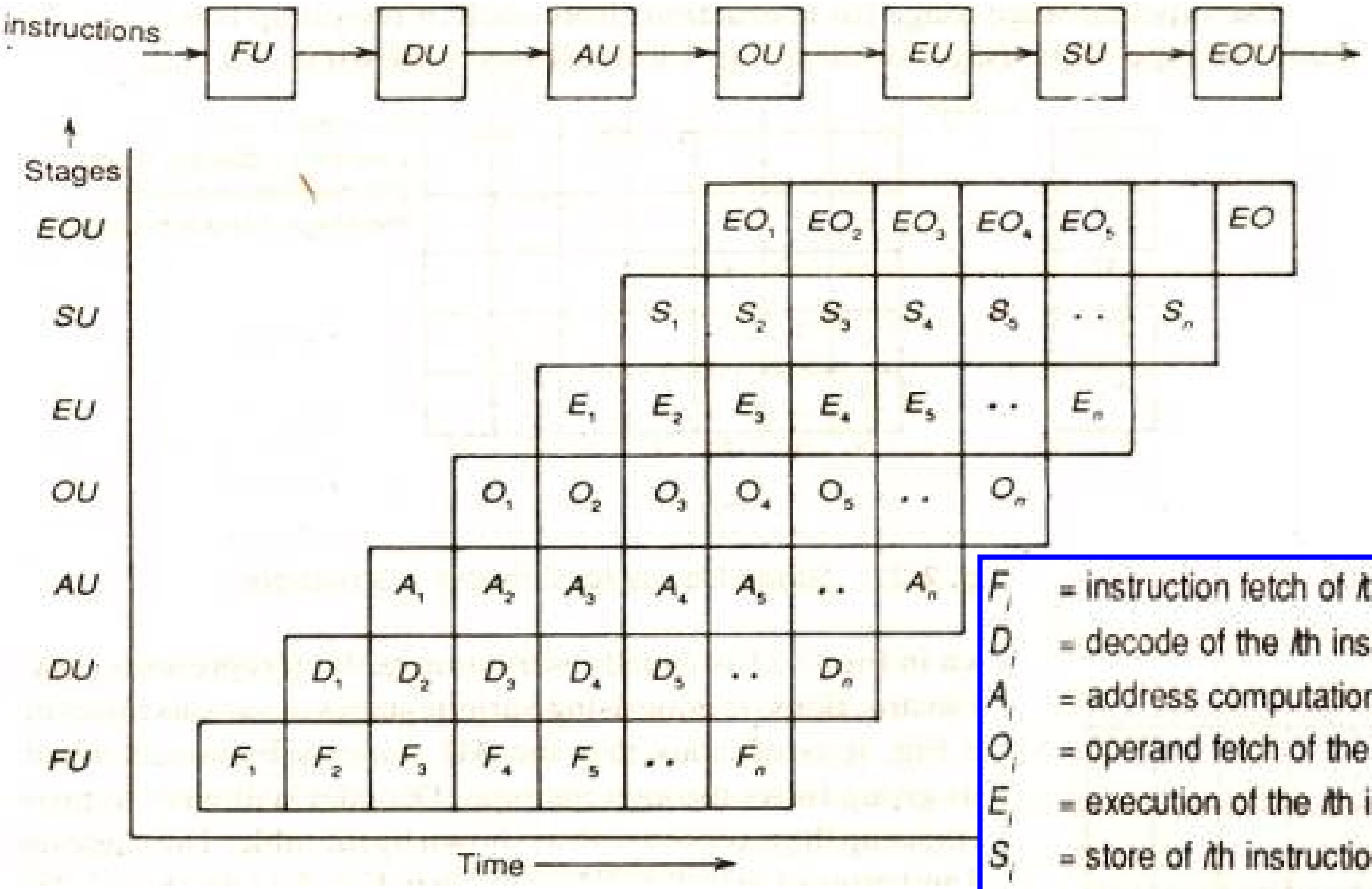
- This pipeline computes b’s using only TWO pipes and third pipe computes the X’s.
- This pipeline has 12 stages and computes one X every clock cycle with each new value provided on the input



Example: the instruction cycle can be implemented as a sequence of a basic steps;



OP-Code	Memory Address	GPR



Stages													
EOU							EO_1	EO_2	EO_3	EO_4	EO_5		EO
SU						S_1	S_2	S_3	S_4	S_5	..	S_n	
EU					E_1	E_2	E_3	E_4	E_5	..	E_n		
OU				O_1	O_2	O_3	O_4	O_5	..	O_n			
AU			A_1	A_2	A_3	A_4	A_5	..	A_n				
DU		D_1	D_2	D_3	D_4	D_5	..	D_n					
FU	F_1	F_2	F_3	F_4	F_5	..	F_n						

F_i = instruction fetch of ith instruction

D_i = decode of the ith instruction

A_i = address computation of ith instruction

O_i = operand fetch of the ith instruction

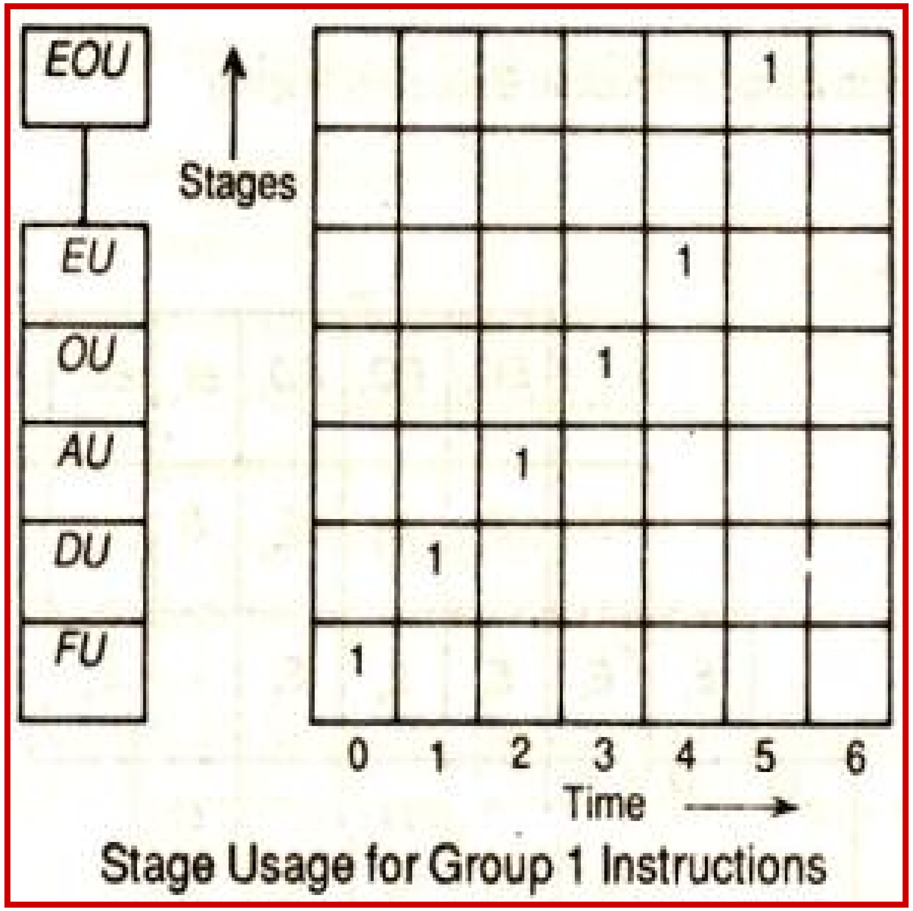
E_i = execution of the ith instruction

S_i = store of ith instruction

EO_i = EOP for ith instruction.

- The instructions may be classified into groups such that a group represents the similarity in the use of stages by all the instructions in the group. It is easy to characterize the groups from the overall function they present. The following groups are:

Group 1: add like instructions.

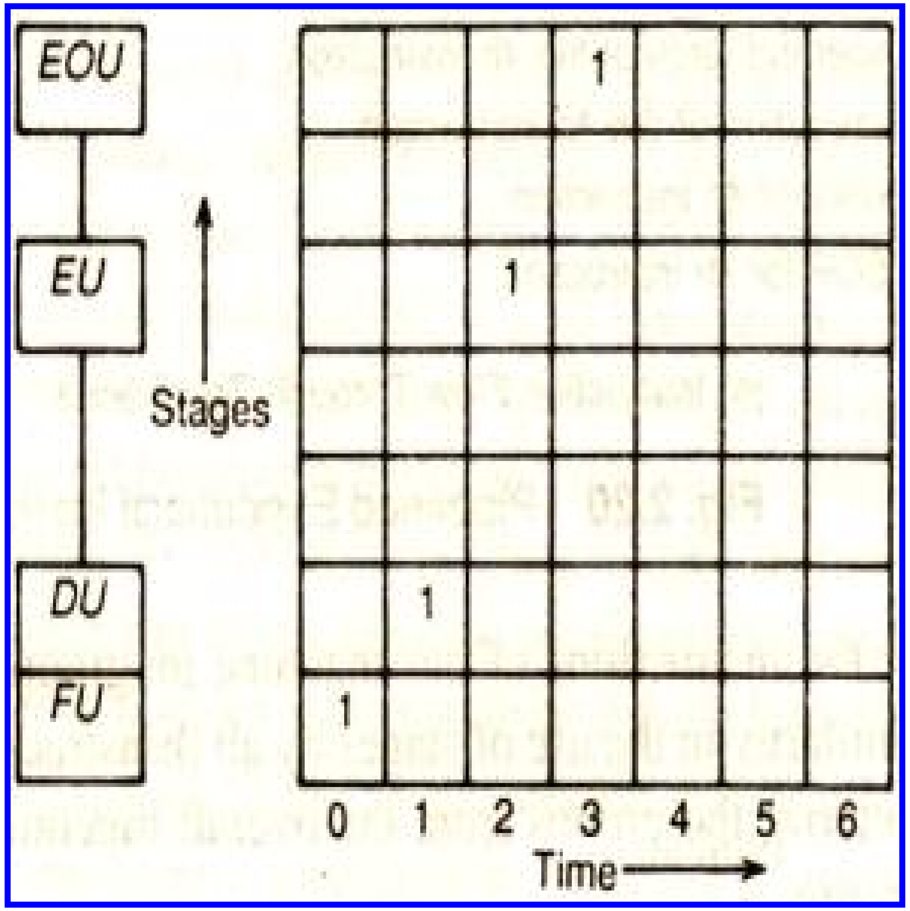


Stages	0	1	2	3	4	5	6
EOU						1	
EU					1		
OU				1			
AU			1				
DU		1					
FU	1						

Stages
Time

Stage Usage for Group 1 Instructions

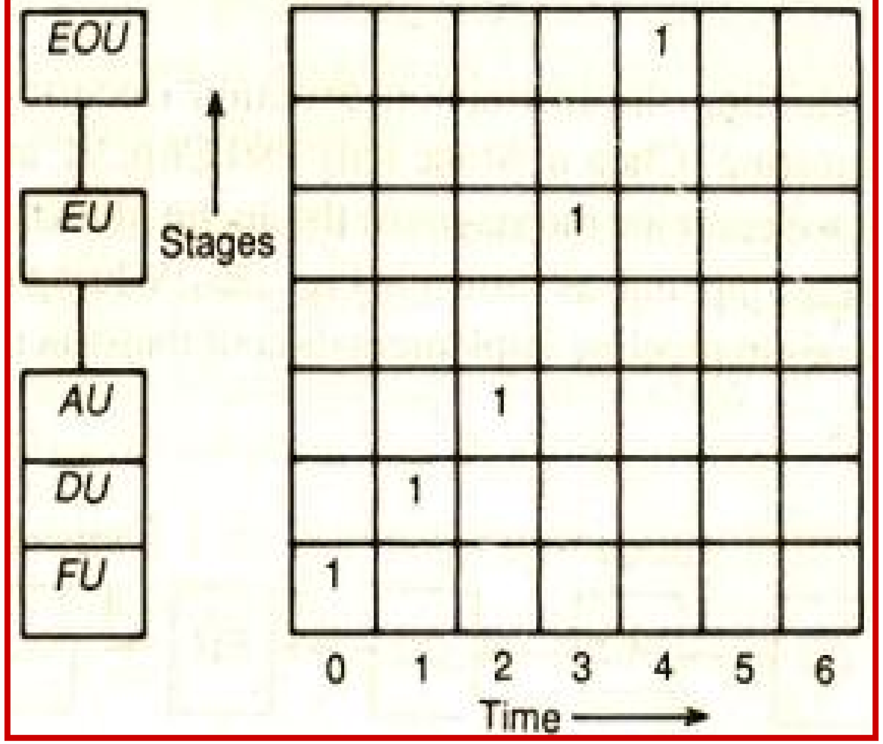
Group 2: branch instructions.



Stages	0	1	2	3	4	5	6
EOU				1			
EU			1				
DU		1					
FU	1						

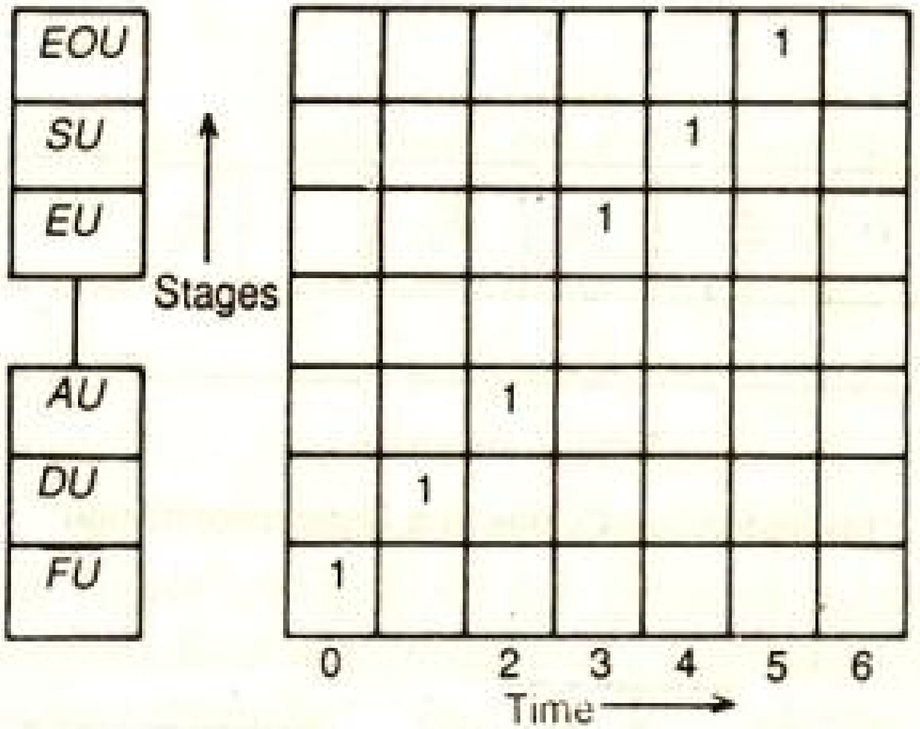
Stages
Time

Group 3: register to register instructions.

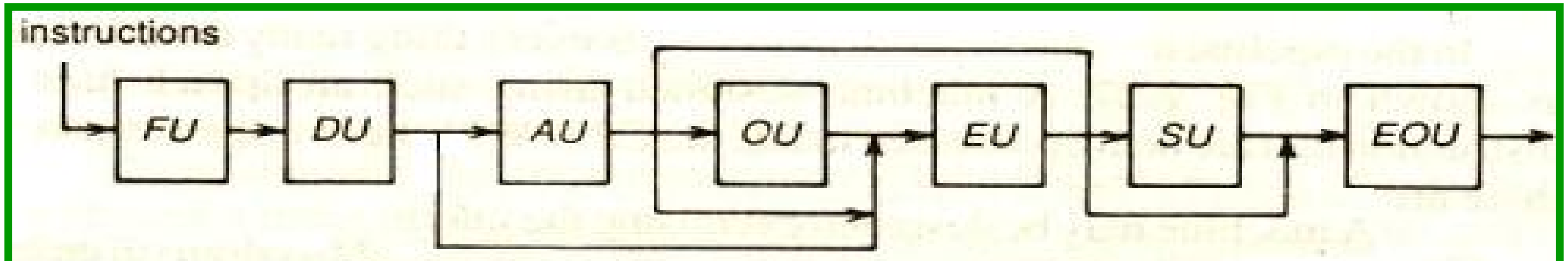

Stages
Time

	0	1	2	3	4	5	6
EOU					1		
EU				1			
AU			1				
DU		1					
FU	1						

Group 4: store instructions.

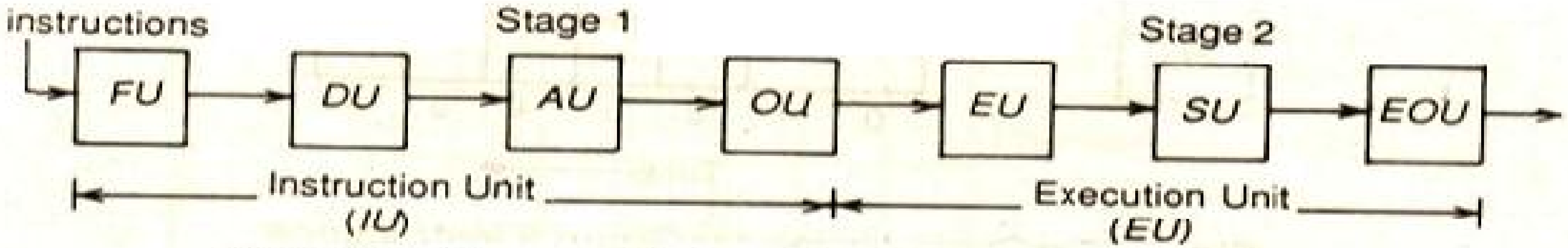

Stages
Time

	0		2	3	4	5	6
EOU						1	
SU					1		
EU				1			
AU			1				
DU		1					
FU	1						

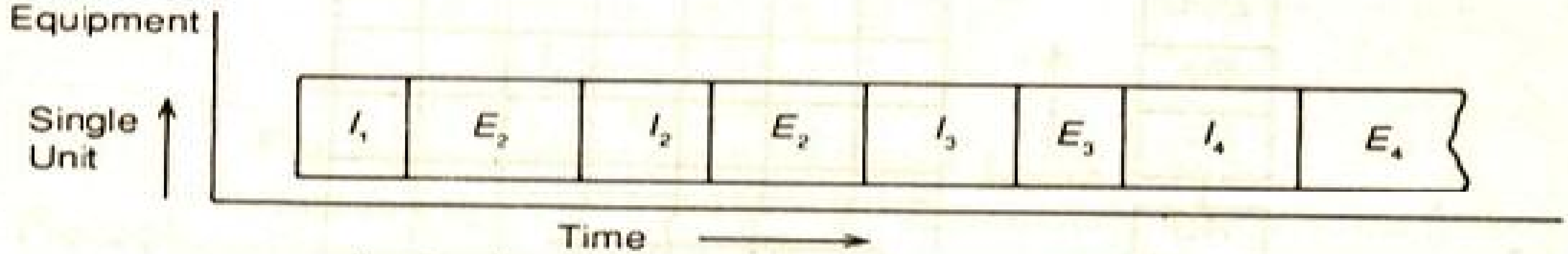


Seven Stage Instruction Execution Pipeline

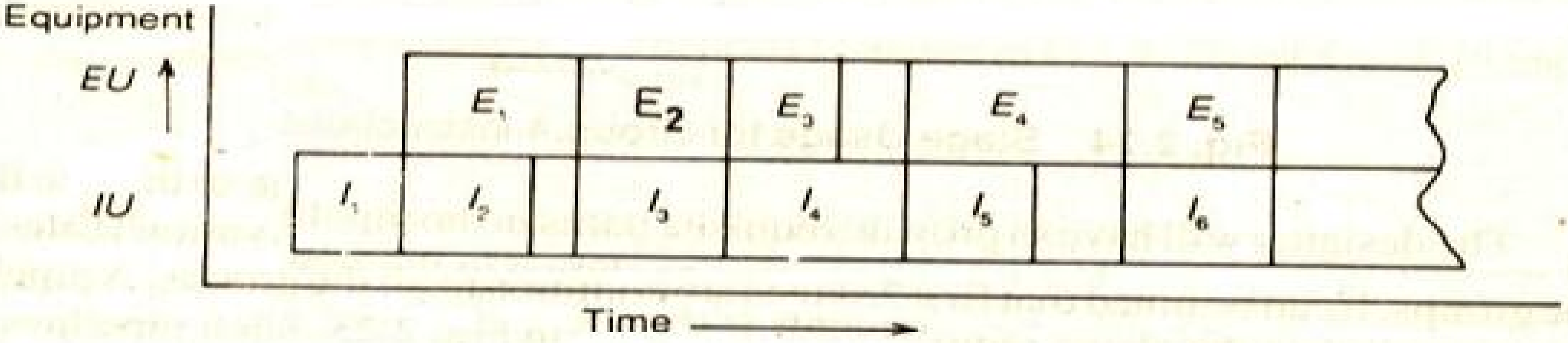
Pipeline Instruction Processing: Some Issues



Main Two Stages of Instruction Execution Pipe



Instruction Cycles in a Conventional Design



Instruction cycles in pipelined Design