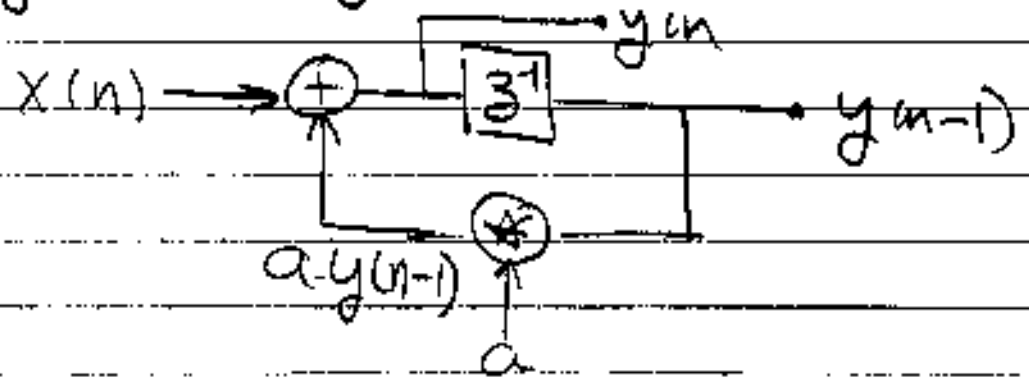


# SPEEDING UP RECURSIVE SYSTEMS

(1)

$$y(n) = a \cdot y(n-1) + x(n)$$



$$T_{\text{ADDER}} = 1$$

$$T_{\text{MULT}} = 3$$

$$T_{\text{IPB}} = \frac{T_{\text{ADDER}} + T_{\text{MULT}}}{1}$$

## LOOK AHEAD

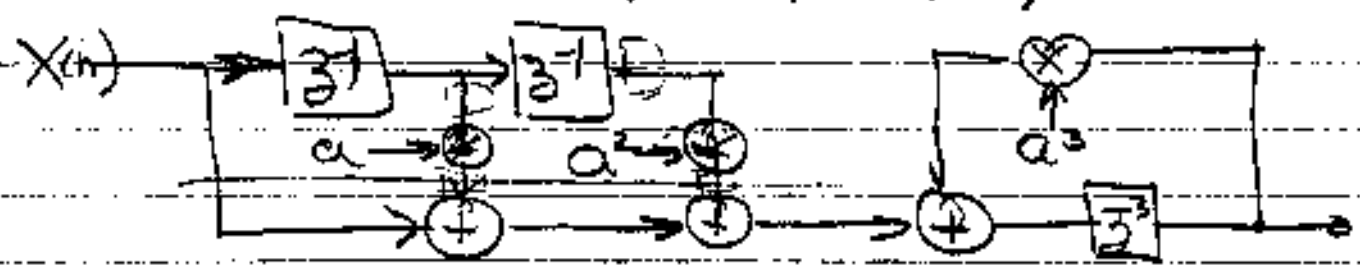
- ITERATE THE EQUATIONS

$$y(n) = a(n-1) + x(n)$$

$$y(n-1) = a(n-2) + x(n-1)$$

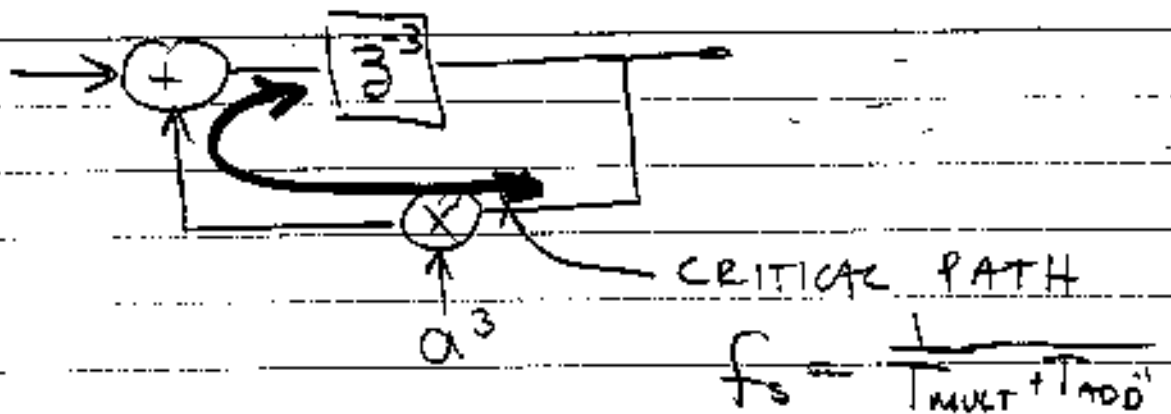
$$y(n-2) = a(n-3) + x(n-2)$$

$$y(n) = a^3 y(n-3) + a^2 x(n-2) + a x(n-1) + x(n)$$

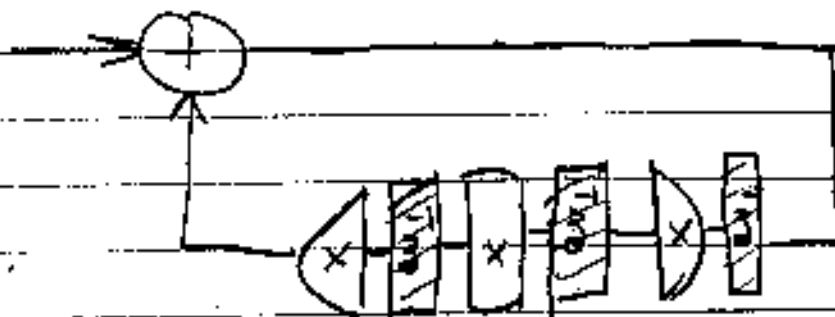


How TO MAKE IT FASTER?

ADD PIPELINE REGISTERS TO THE FORWARD PATH AS NECESSARY.



MOVE THE DELAYS AROUND  
"RETIMING"



ie. PIPELINE THE MULTIPLIER TO A DEPTH OF 3

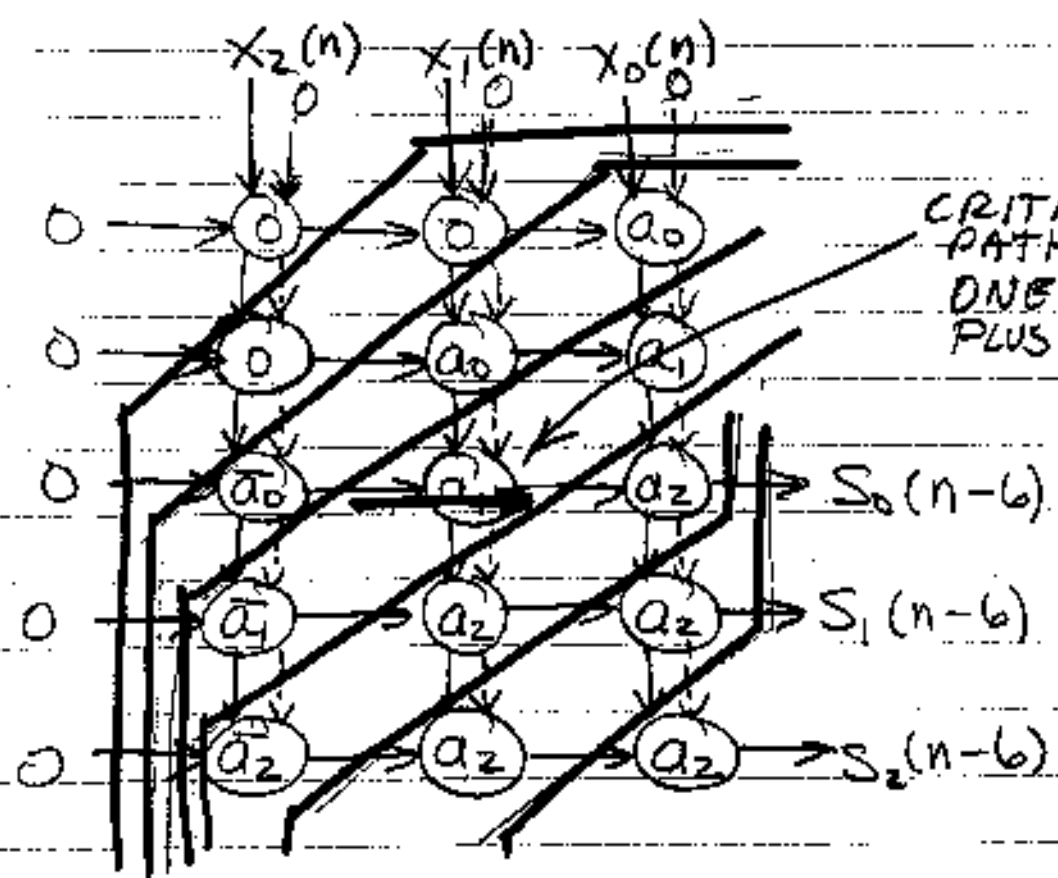
Now WE ACHIEVE THE REGULATION

# BIT LEVEL PIPELINING -

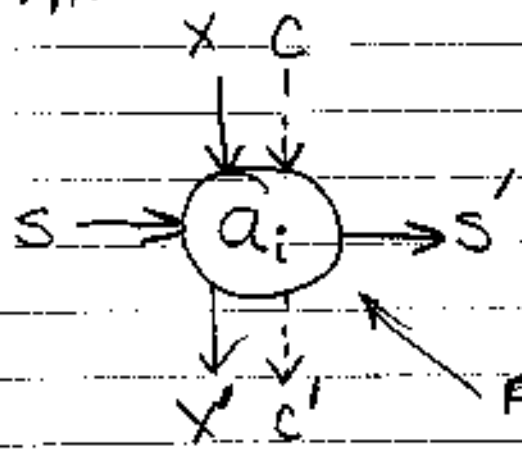
## 3 BIT MULTIPLIER

$$a = a_2 a_1 a_0$$

$$x = x_2 x_1 x_0$$



CRITICAL PATH IS NOW ONE ADDER PLUS REGISTERS



$$x' = x$$

$$s' = \text{SUM}(s, a_i \cdot x, c)$$

$$c' = \text{CARRY}(s, a_i \cdot x, c)$$

FULL ADDER WITH AND ( $a_i \cdot x$ )

# ANOTHER STRATEGY

(SCATTERED LOOKAHEAD)

START WITH  $H(z)$

$$H(z) = \frac{(z - z_1)(z - z_2) \dots}{(z - p_1)(z - p_2) \dots}$$

USE THE IDENTITY:

$$\frac{1}{z - p_i} = \frac{z^{L-1} + p_i z^{L-2} + p_i^2 z^{L-3} + \dots + p_i^{L-1}}{z^L - p_i^L}$$

$$\Rightarrow \frac{N(z)}{D(z^D)}$$



NON-RECURSIVE

RECURSIVE WITH

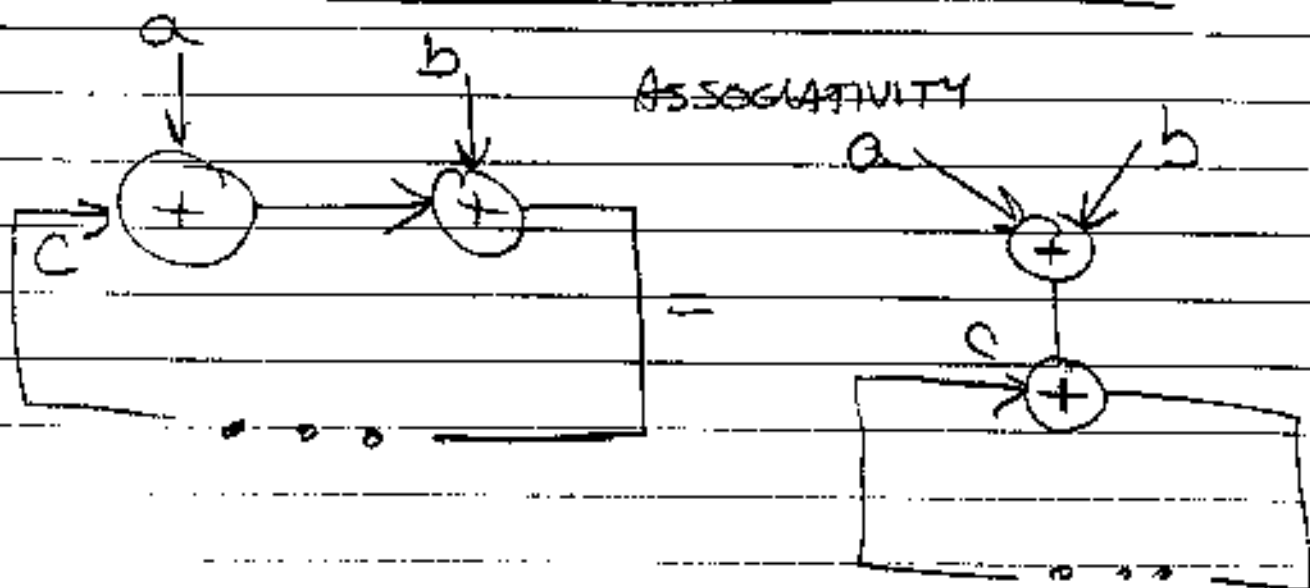
ALL LOOPS WITH

AT LEAST  $D$  DELAYS

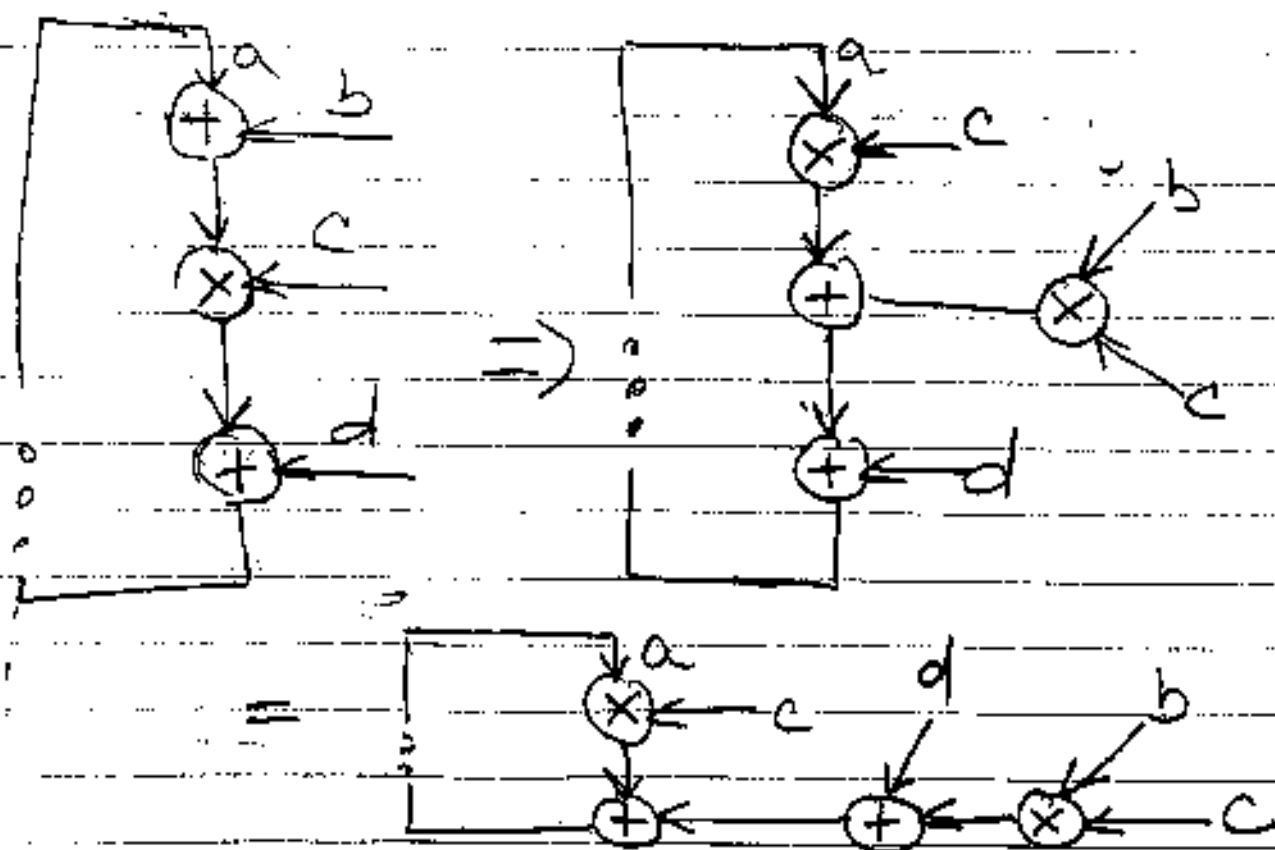
TO USE FOR PIPELINING

# YET ANOTHER METHOD FOR REDUCING ITERATION BOUND

## LOOP SHRINKING TRANSFORMATIONS



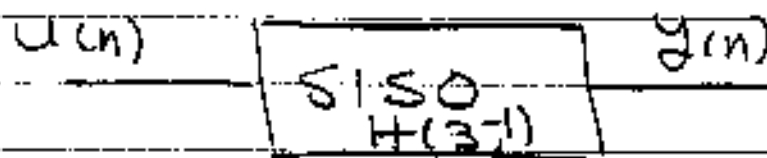
## DISTRIBUTIVITY



# FILTER TRANSFORMATIONS

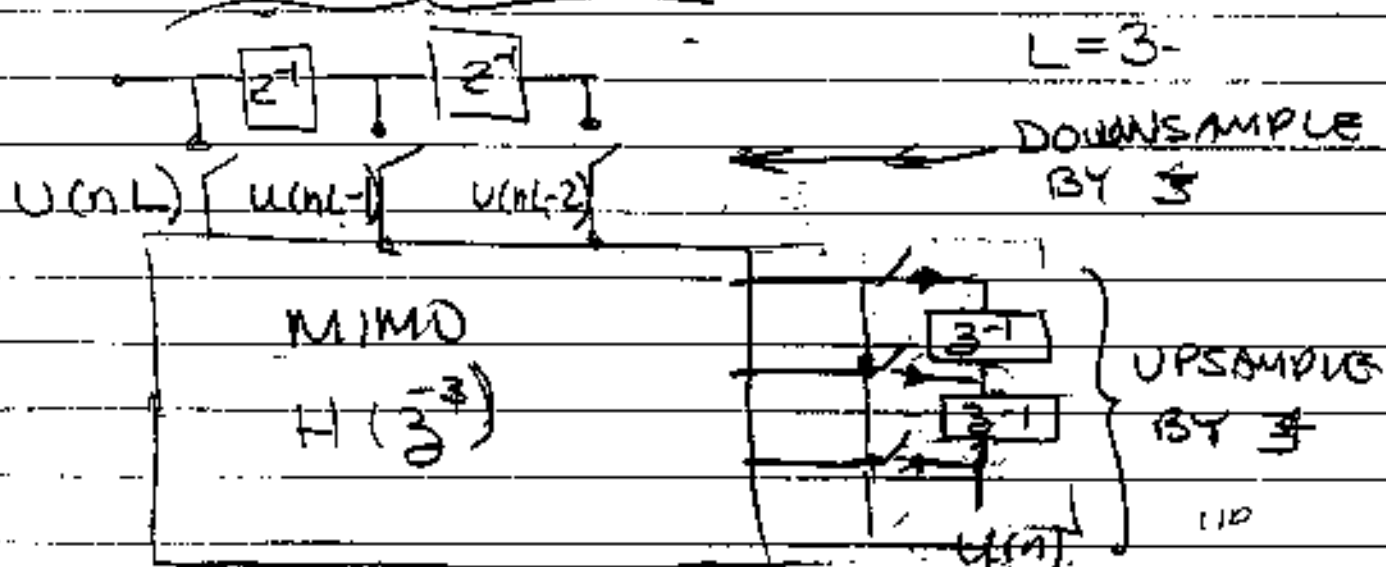
(MESSERSCHMITT - BREAKING THE RECURSIVE BOTTLENECK)

SISO  $\Rightarrow$  SINGLE INPUT SINGLE OUTPUT



CONVERSION TO MULTIPLE INPUT MULTIPLE OUTPUT (MIMO)

SERIAL TO PARALLEL CONVERTER

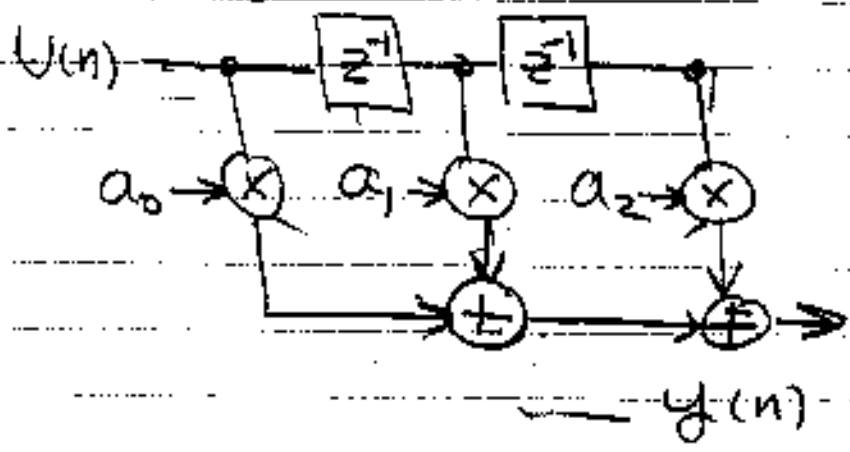


$\frac{1}{3}$  INPUTS AT ONCE ( $z^{-3}$  TIME)

3 OUTPUTS " " " " " "

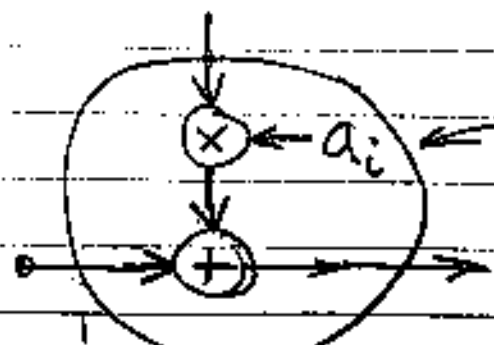
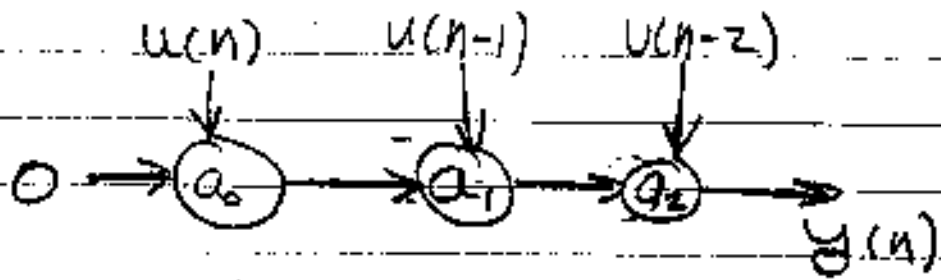
# FIR FILTER TRANSFORMATIONS

DIRECT FORM:



$$y(n) = a_0 u(n) + a_1 u(n-1) + a_2 u(n-2)$$

PE FORM OF ABOVE



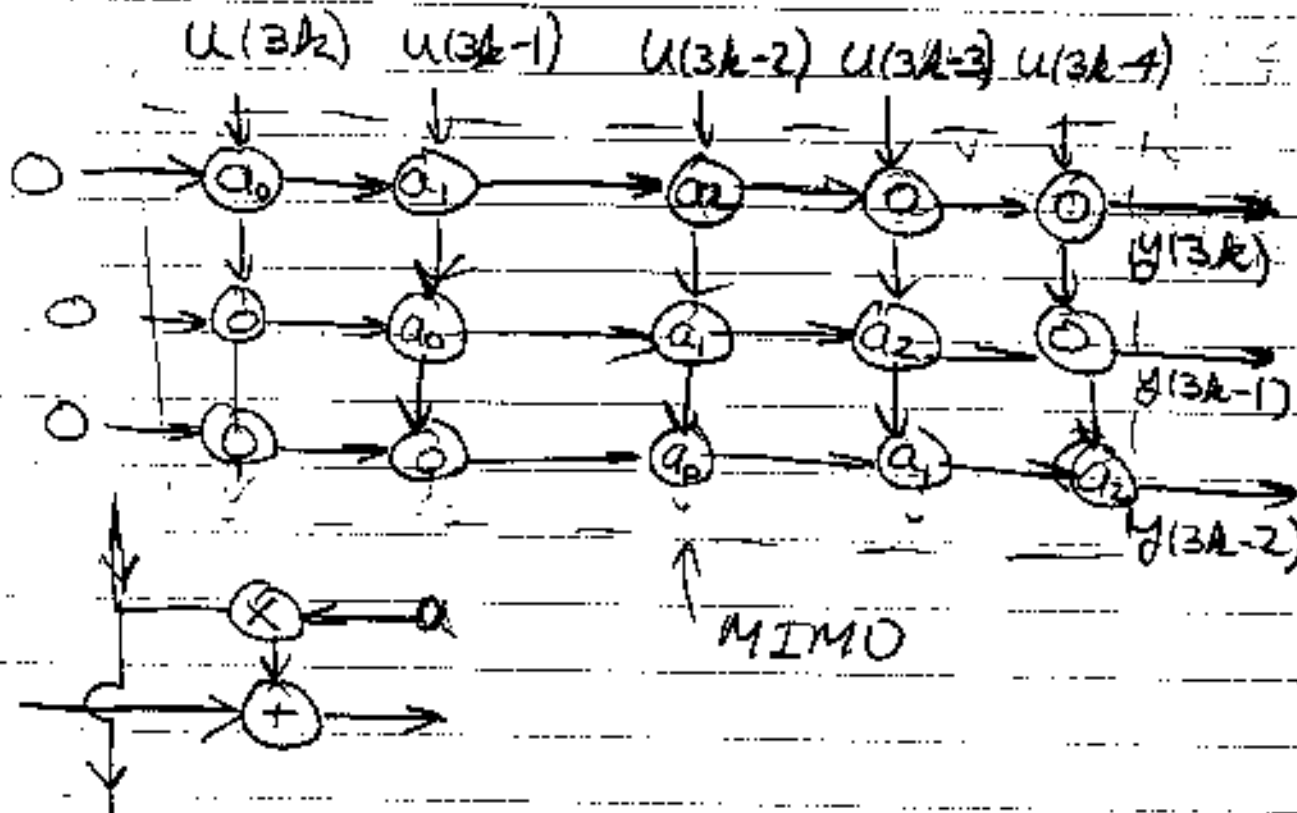
A PE (PROCESSOR ELEMENT)

L = 3      3 TIMES SPEEDUP

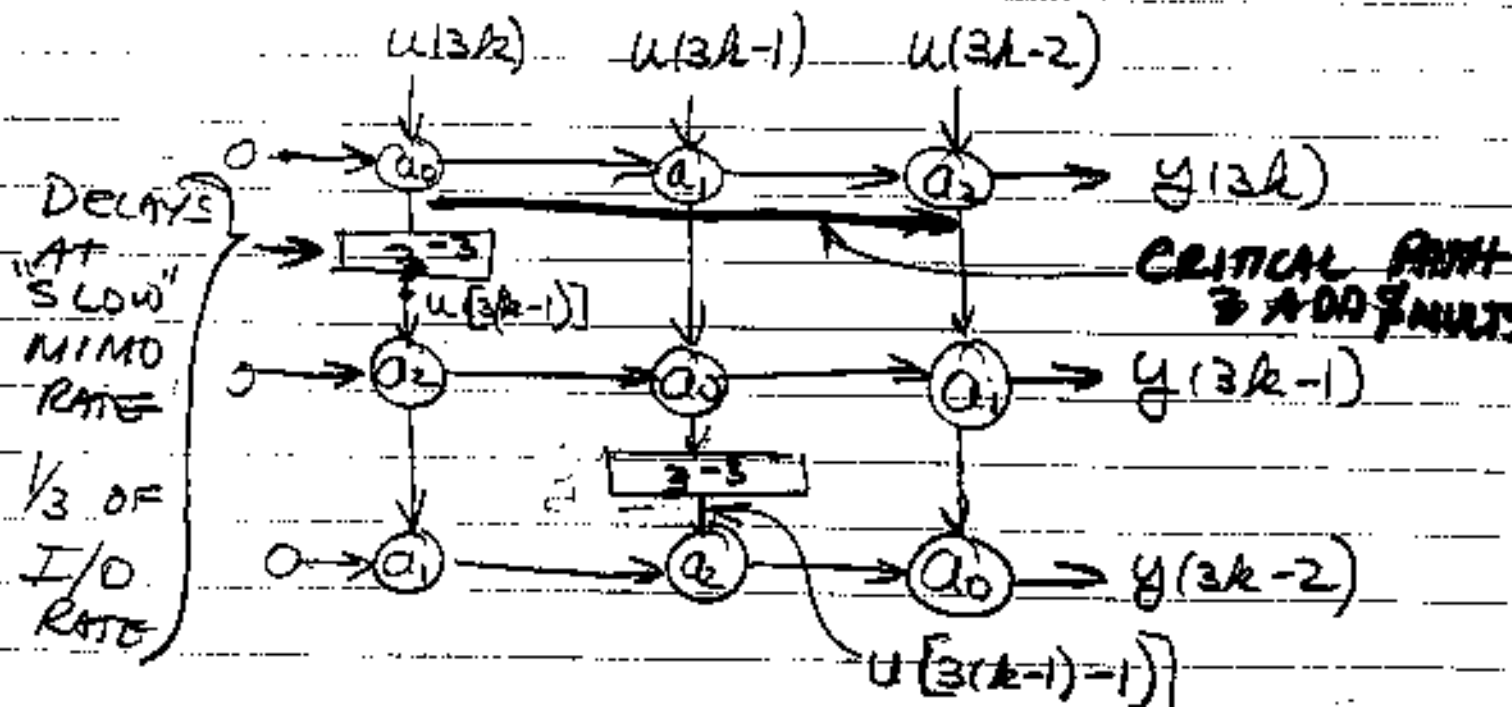
⑤  
8

ARRAY IMPLEMENTATION OF

$$y(n) = a_0 u(n) + a_1 u(n-1) + a_2 u(n-2)$$

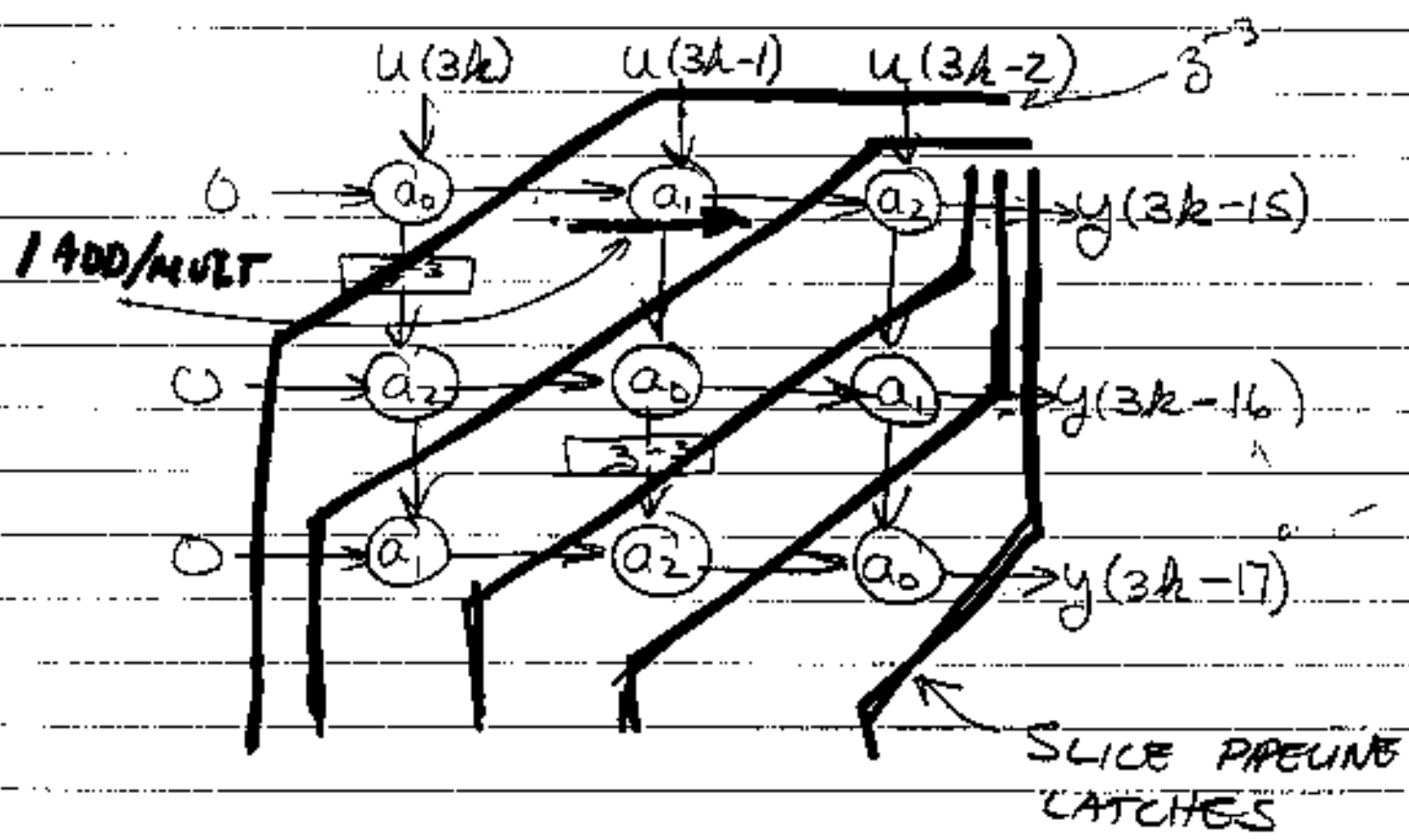


"FOLDING" THIS :



# ADDING PIPELINE REGISTERS TO A FEEDFORWARD CUTSET

ie. MUST ADD THE SAME NUMBER OF DELAYS TO EACH FORWARD PATH BETWEEN INPUT AND OUTPUT



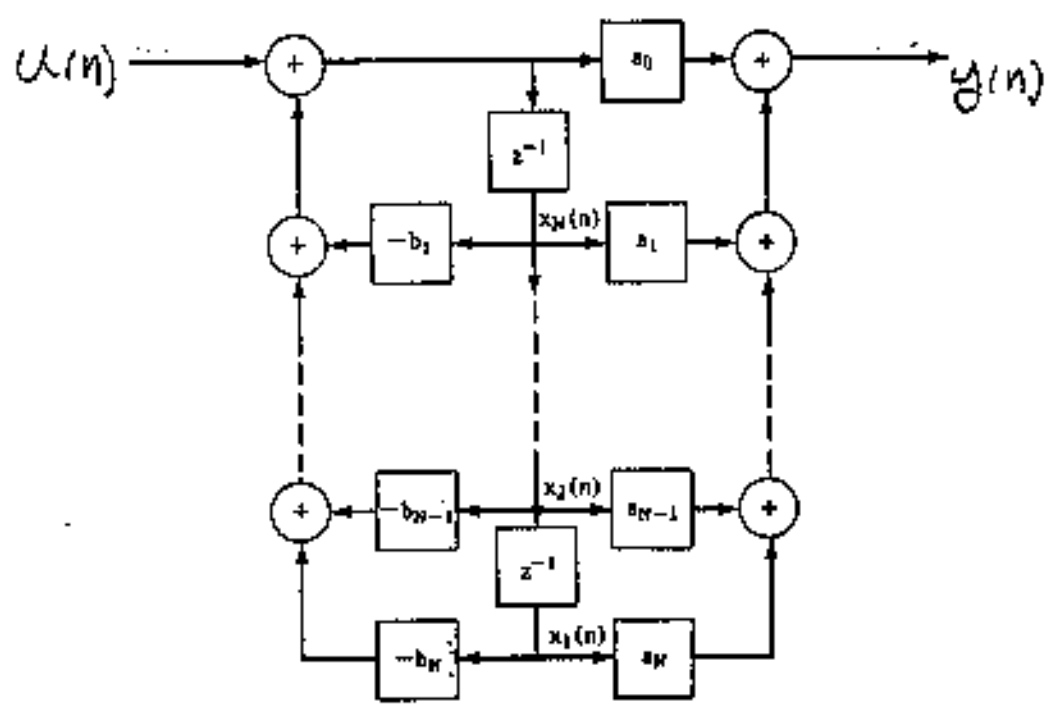
ADDS A LATENCY OF 15

$$y(3k-1) \rightarrow y(3k-16)$$

THIS IS WORD LEVEL PIPELINING

# HIGHER ORDER SYSTEMS

## GENERAL $N^{\text{TH}}$ ORDER SYSTEM



$$y(n) = \sum_{k=0}^N a_k x(n-k) - \sum_{k=1}^N b_k y(n-k) \quad (2.97)$$

where we have chosen  $N = M$  for convenience sake.

The general Direct Form II realization of equation (2.97) is illustrated in Figure 2.13 for the case of  $M = N$ . If we choose as state variables the outputs of the delay elements then from the block diagram we can write

$$\begin{aligned} x_1(n+1) &= x_2(n) \\ x_2(n+1) &= x_3(n) \\ &\vdots \\ x_{N-1}(n+1) &= x_N(n) \\ x_N(n+1) &= -b_N x_1(n) - b_{N-1} x_2(n) - \dots - b_1 x_N(n) + u(n) \end{aligned} \quad (2.98)$$

and

$$\begin{aligned} y(n) &= a_N x_1(n) + a_{N-1} x_2(n) + \dots + a_1 x_N(n) + a_0 x_N(n+1) \\ &= (a_N - a_0 b_N) x_1(n) + (a_{N-1} - a_0 b_{N-1}) x_2(n) + \dots \\ &\quad + (a_1 - a_0 b_1) x_N(n) + a_0 u(n) \end{aligned} \quad (2.99)$$

# STATE SPACE REPRESENTATION

From equations (2.98) and (2.99) we can write the matrix state equation:

$$\begin{bmatrix} x_1(n+1) \\ x_2(n+1) \\ \vdots \\ x_{N-1}(n+1) \\ x_N(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ & & & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ -b_N & -b_{N-1} & \dots & -b_2 & -b_1 \end{bmatrix} \begin{bmatrix} x_1(n) \\ x_2(n) \\ \vdots \\ x_N(n) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} u(n) \quad (2.100)$$

and the output equation

$$y(n) = [(a_N - a_0 b_N), (a_{N-1} - a_0 b_{N-1}), \dots, (a_1 - a_0 b_1)] \begin{bmatrix} x_1(n) \\ x_2(n) \\ \vdots \\ x_N(n) \end{bmatrix} + a_0 u(n) \quad (2.101)$$

Equations (2.100) and (2.101) provide a state-space representation of the filter.

IN MATRIX NOTATION

$$\vec{x}(n+1) = \vec{A} \vec{x}(n) + \vec{b} u(n)$$

$$y(n) = \vec{c}^T \vec{x}(n) + a_0 u(n)$$

$\vec{A} \equiv$  STATE MATRIX

$$H(z) = \vec{c}^T \underbrace{(z\vec{I} - \vec{A})^{-1}}_{\text{POLES OF } H(z)} \vec{b} + a_0$$

THIS CAN BE CONVERTED TO A MIMO SYSTEM BY DEFINING AN INPUT VECTOR  $\vec{u}^{(L)}(kL)$  OR  $L$  INPUTS AND  $L$  OUTPUTS  $\vec{y}^{(L)}(kL)$

WHERE  $\vec{u}^{(L)}(kL) = \begin{pmatrix} u(kL) \\ u(kL+1) \\ \vdots \\ u[kL+(L-1)] \end{pmatrix}$  (SAME FOR  $\vec{y}^{(L)}(kL)$ )

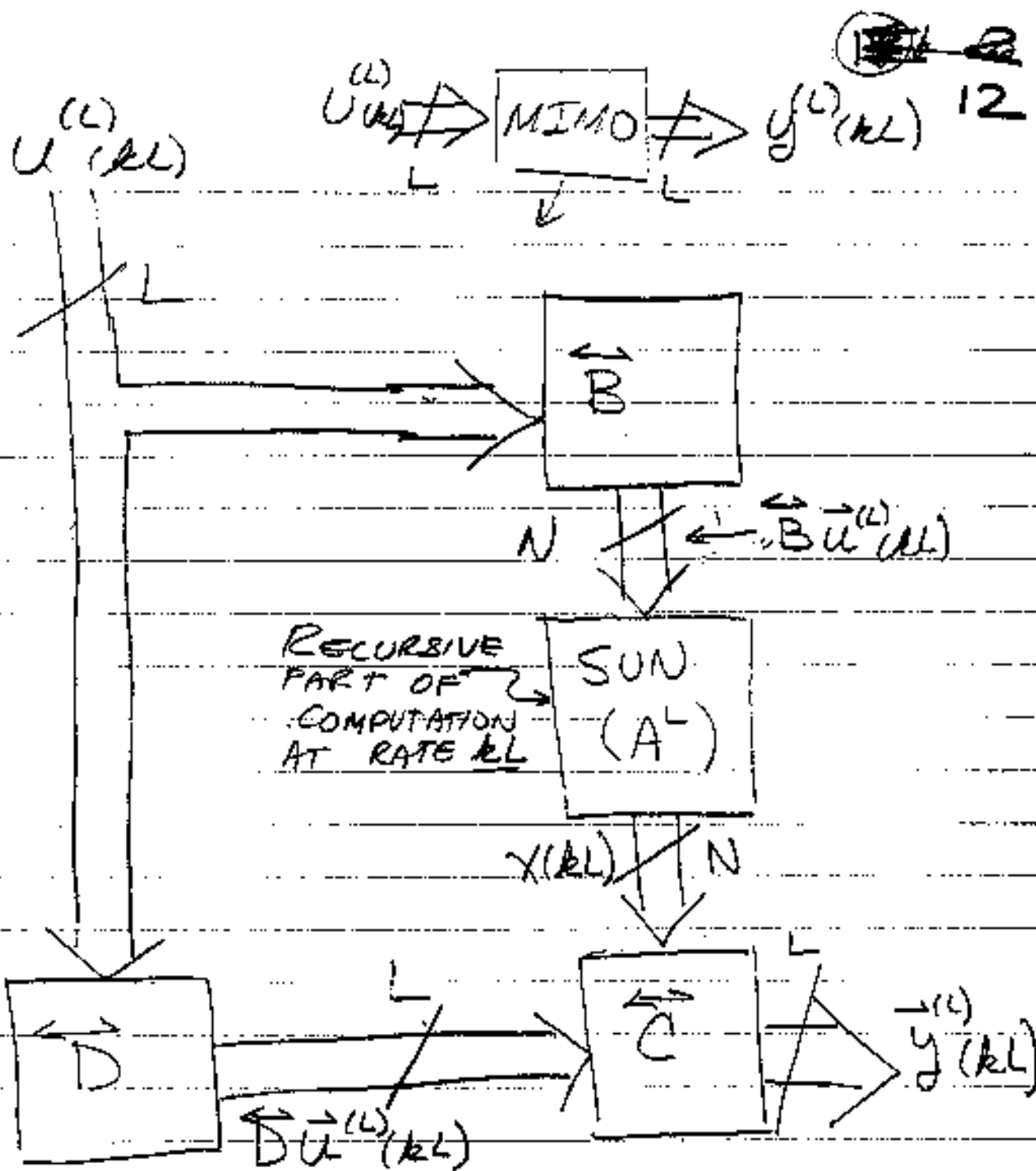
THE  $L$  INPUT/OUTPUT SYSTEM

- (1)  $\vec{x}((k+1)L) = \vec{A}^L \vec{x}(kL) + \vec{B} \vec{u}^{(L)}(kL)$ 
(2)  $\vec{y}^{(L)}(kL) = \vec{C} \vec{x}(kL) + \vec{D} \vec{u}^{(L)}(kL)$

EQ. (1)  $\Rightarrow$  STATE UPDATE NETWORK EQUATION (SUN)

$\vec{A}^L, \vec{B}, \vec{C}$  &  $\vec{D}$  CAN ALL BE PRE-COMPUTED

NOTE THAT THE SUN IS UPDATED AT THE SLOW RATE  $kL$



GENERAL FORM OF BLOCK STATE REALIZATION:

$B$ ,  $C$  &  $D$  BLOCKS ARE SIMPLY MATRIX-VECTOR MULTIPLIERS OPERATING USE LOG-ARITHM IN THE SUN TO

# TIME VARYING CASE

eg. ADAPTIVE FILTERS

$$X(n) = a(n) X(n-1) + u(n)$$

↑  
TIME VARYING

USING LOOK-AHEAD WITH M-1 ITERATIONS:

$$X(n) = C(n, M) X(n-M) + D(n, M)$$

WHERE:

$$C(n, j+1) = a(n-j) C(n, j)$$

$$C(n, 0) = 1$$

$$D(n, j+1) = D(n, j) + u(n-j) C(n, j)$$

$$D(n, 0) = 0$$

