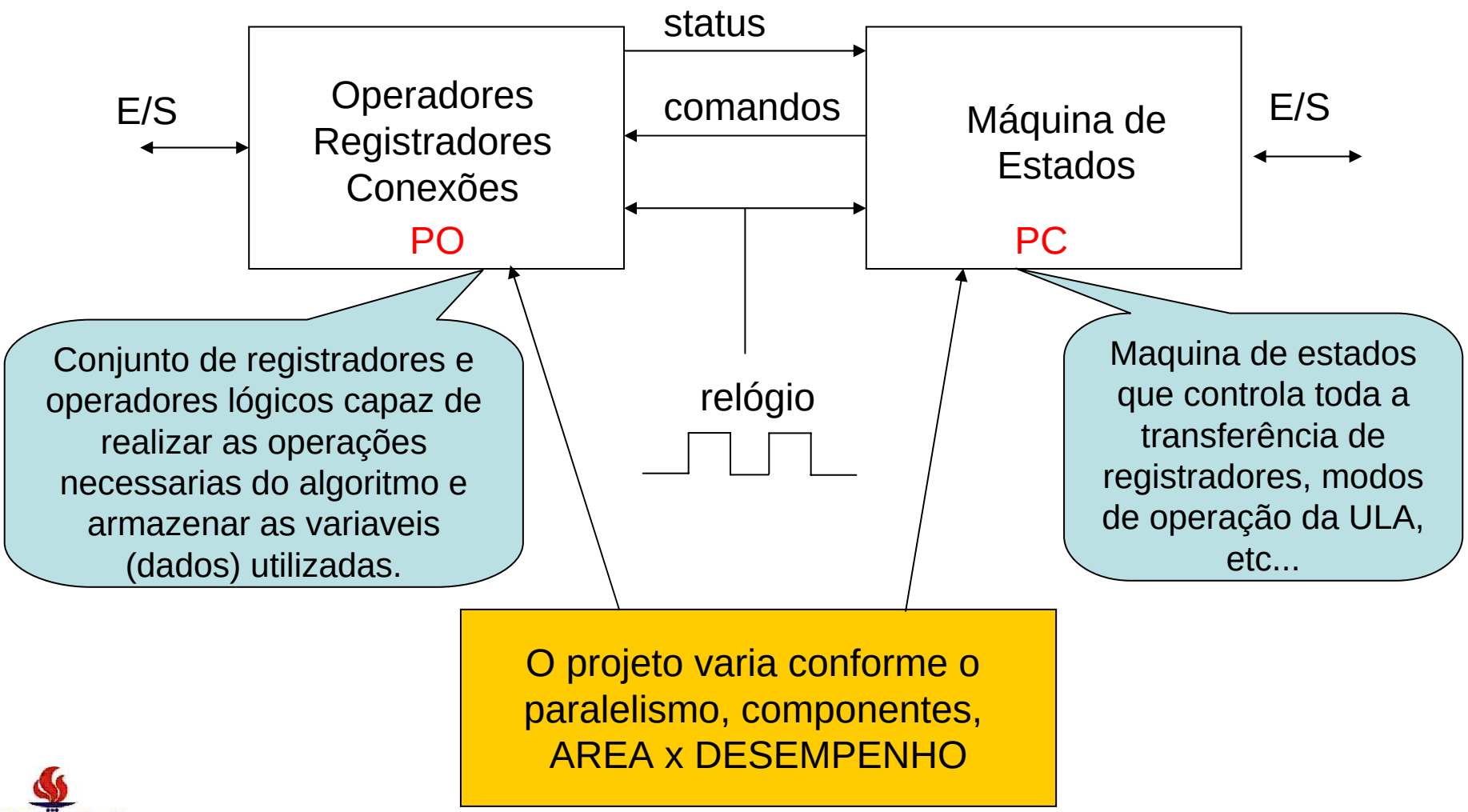
CMP - 238

- Serial x Paralelo
- Pipeline

UFRGS

Parte Operativa – Parte de Controle

Descrição a nivel de transferencia entre registradores (RTL)



Exemplo 1

Projete um bloco de controle (represente na forma de diagrama de estados) que realize a seguinte operação no datapath a seguir:

$S = A.X^2 + B.X + C$

.

```
inicio: X <= novo valor
        Start =1;
        Wait until done=1
        go to inicio;
```

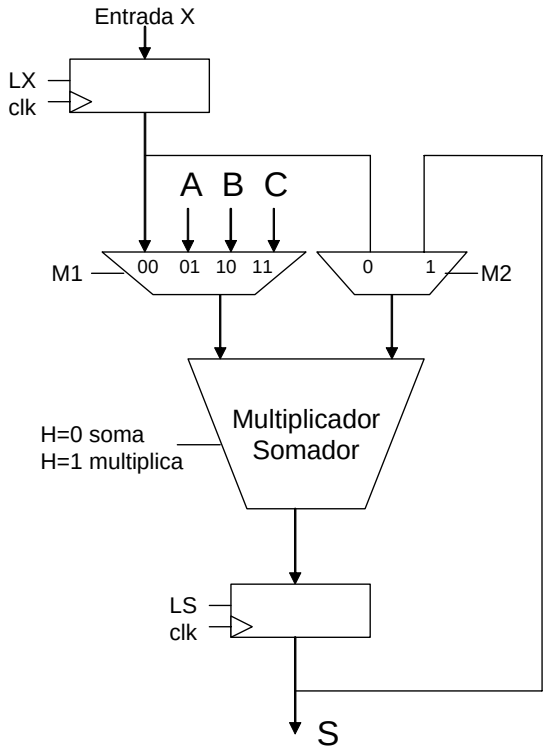
Implementação PARALELA : cada operação tem o seu operador, ou seja, precisamos de 3 multiplicadores e 2 somadores.

Implementação SERIAL : um único operador para somar e multiplicar, ou seja, uma operação por ciclo de relógio, precisa ter registrador para armazenar os estados intermediarios

Exemplo 1

VERSÃO SERIAL

$S = A.X^2 + B.X + C$



A parte operativa tem carater acumulativo, ou seja, multiplica ou soma e acumula no registrador da saida do operador lógico.

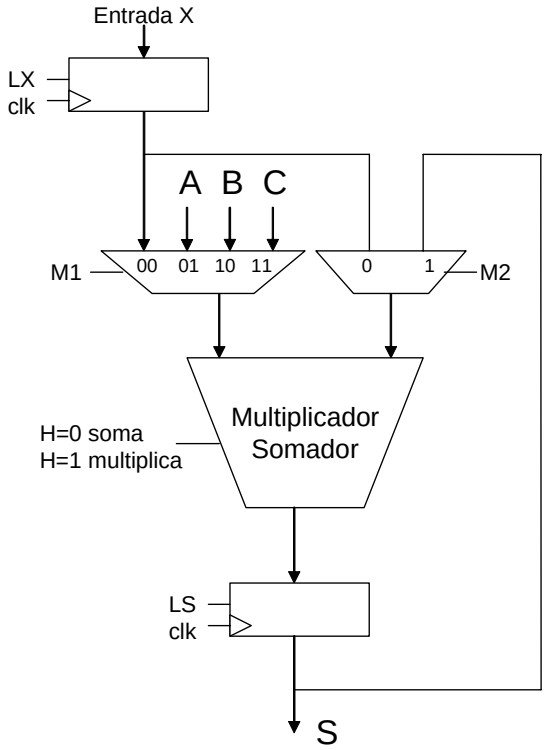
Modificamos a equação

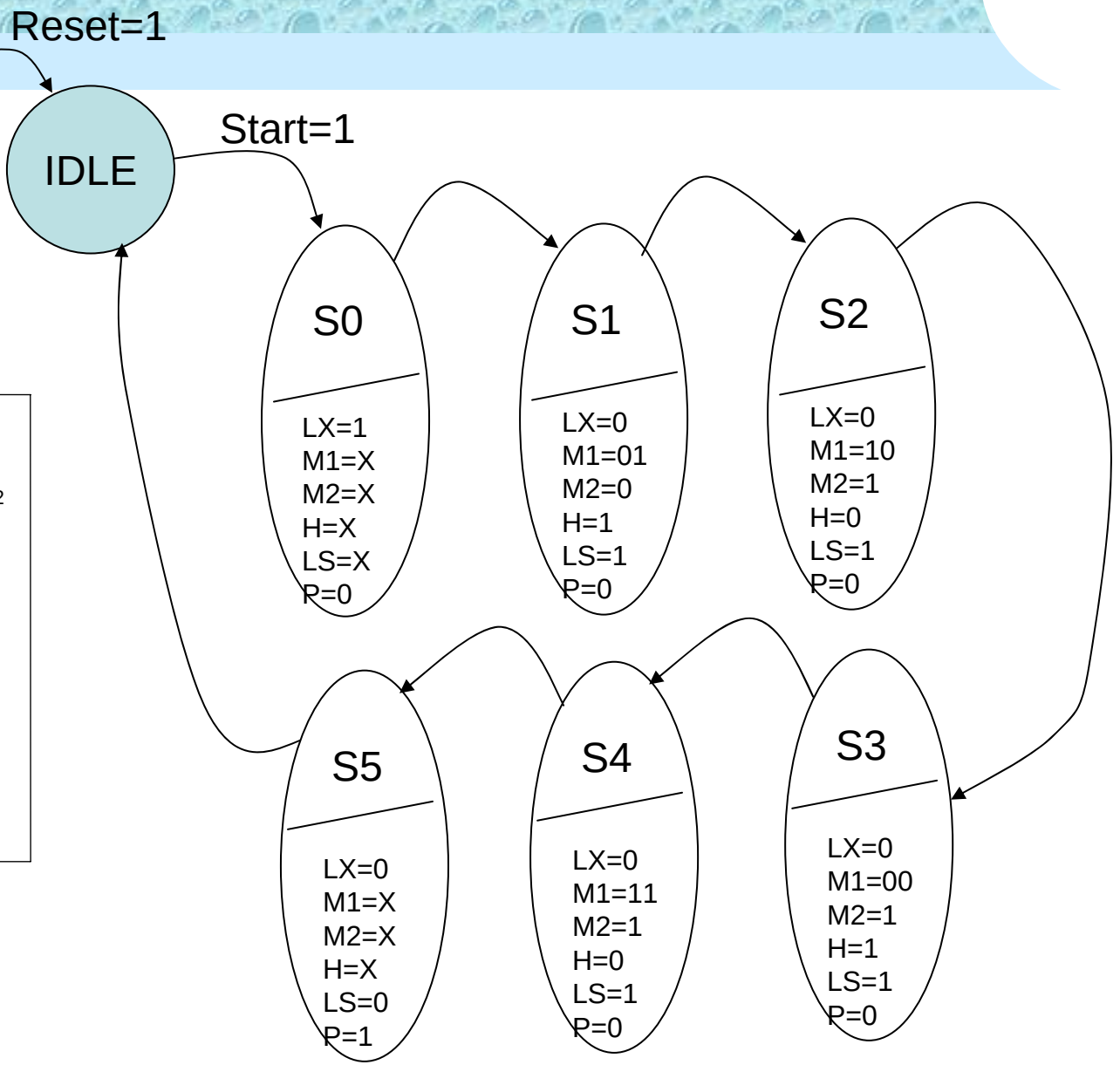
$$S = X (A.X + B) + C$$

Assim, para calcular S temos A que multiplica por X que soma com B que multiplica por X e soma com C.

Exemplo 1

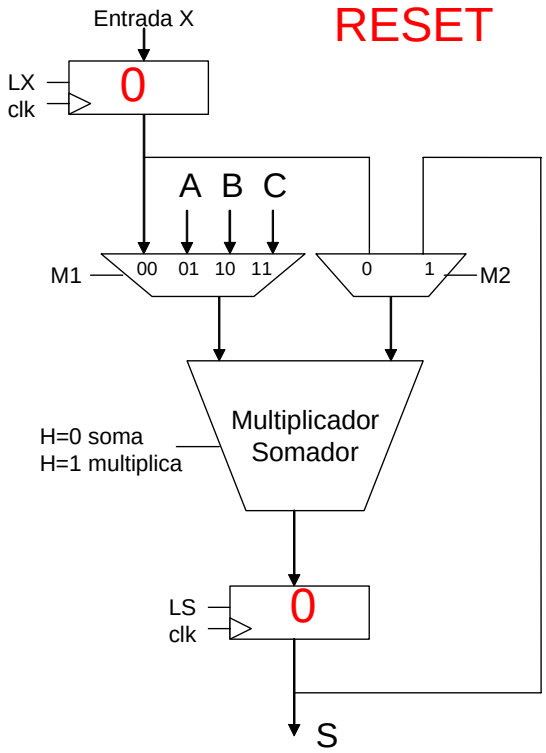
$S = X\,(A.X + B) + C$

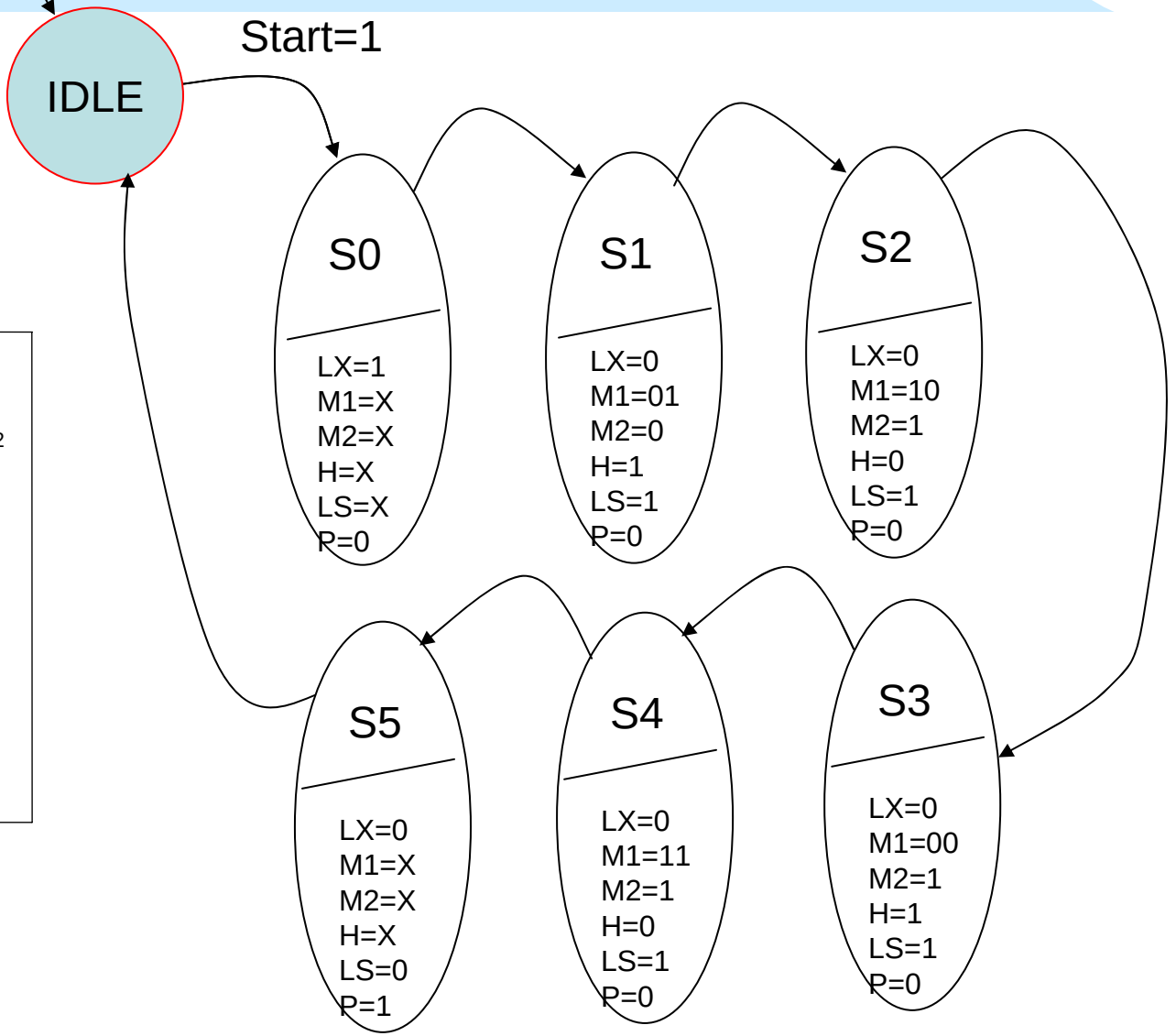




Exemplo 1

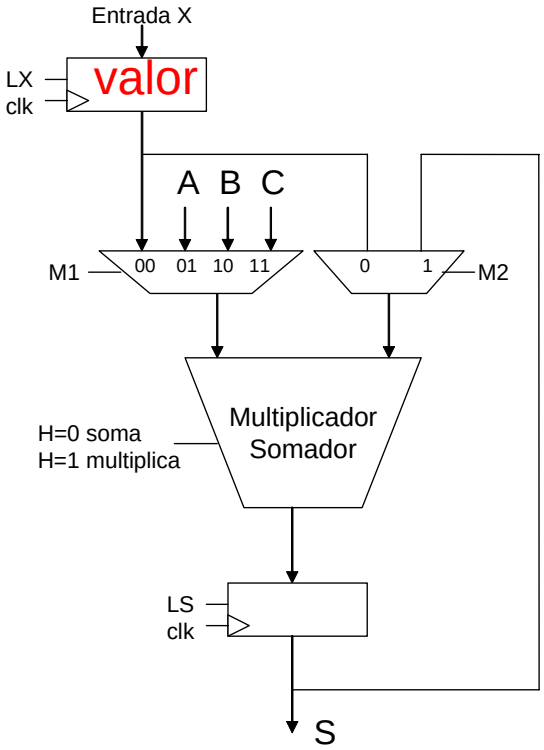
S= X (A.X + B) + C

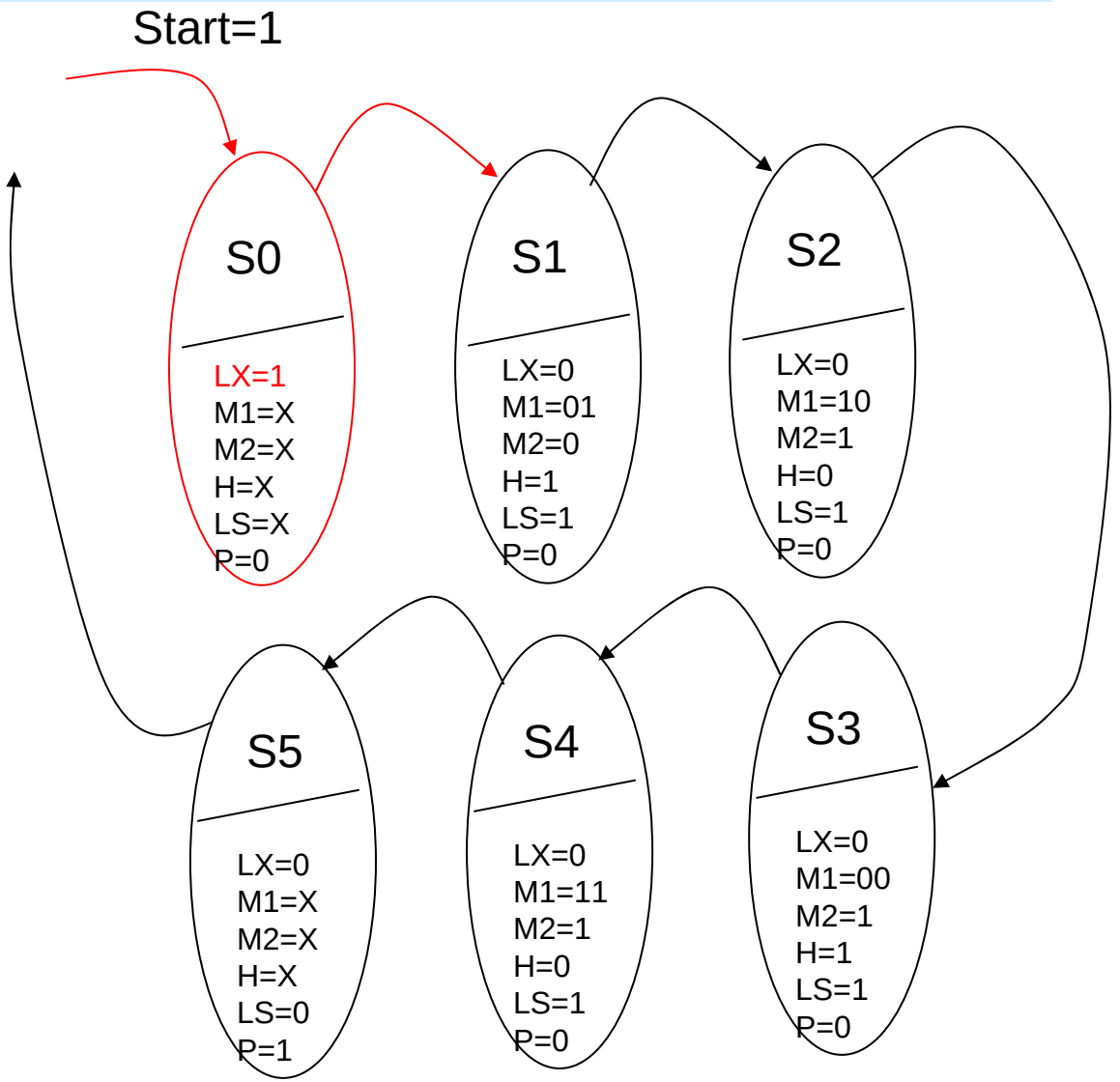




Exemplo 1

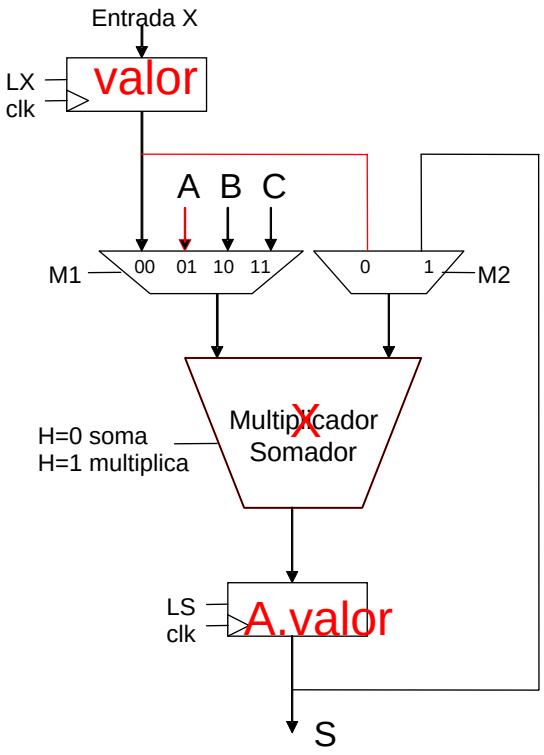
$$S = X\,(A.X + B) + C$$

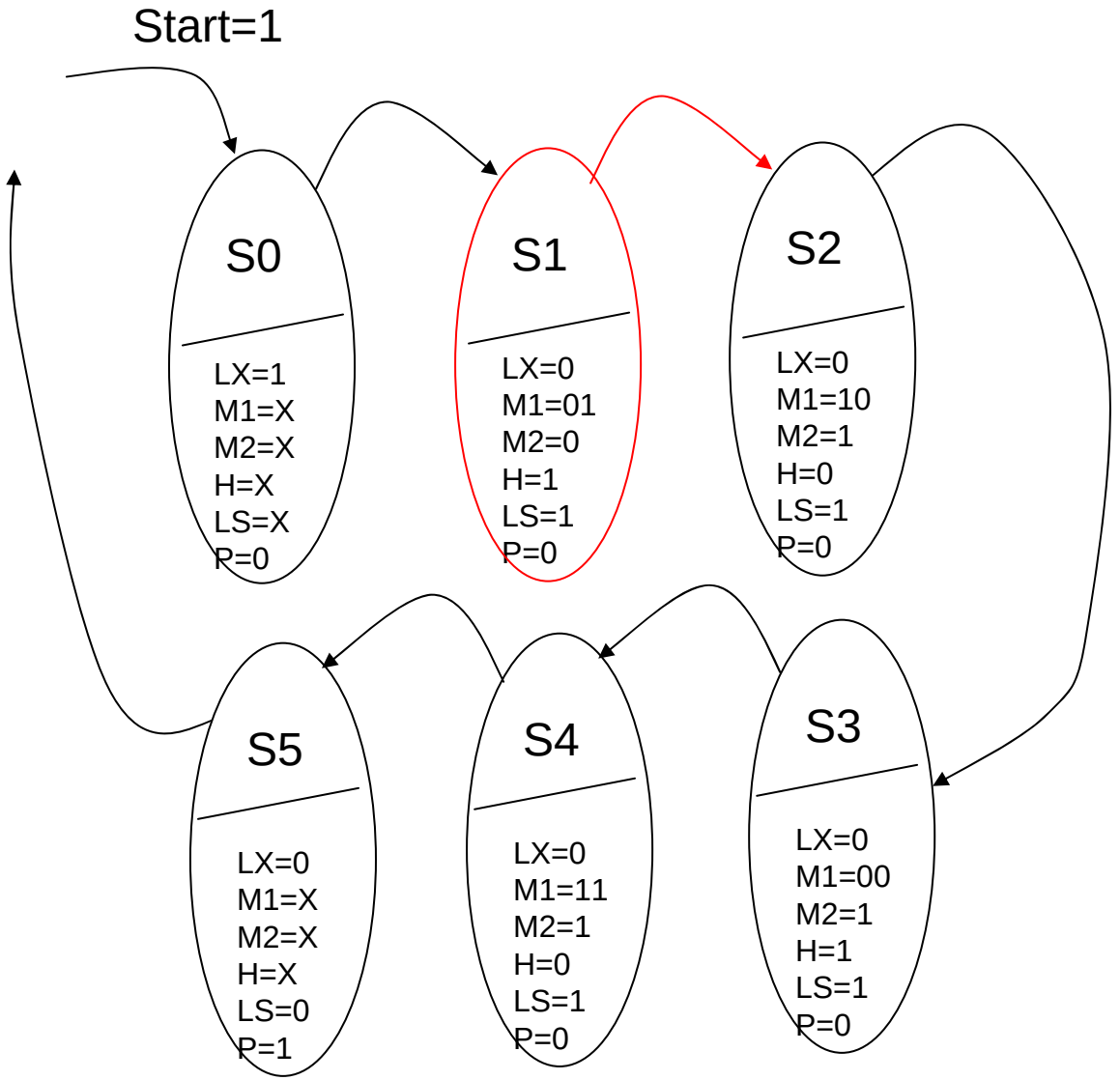




Exemplo 1

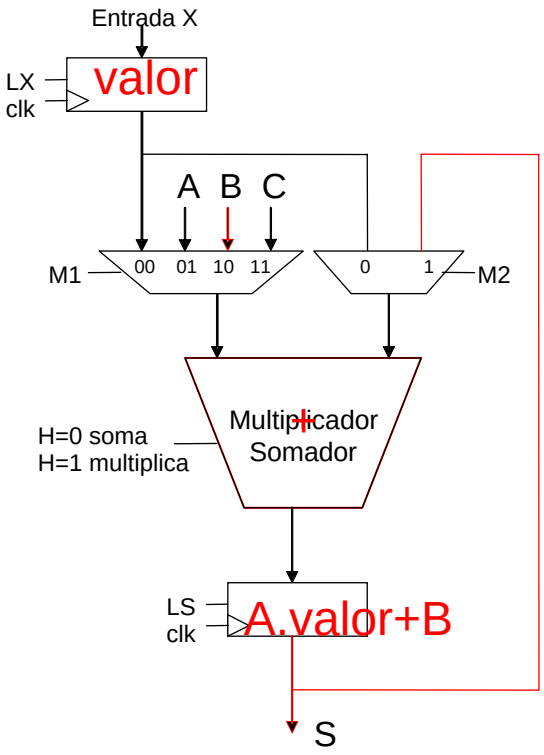
S= X (A.X + B) + C

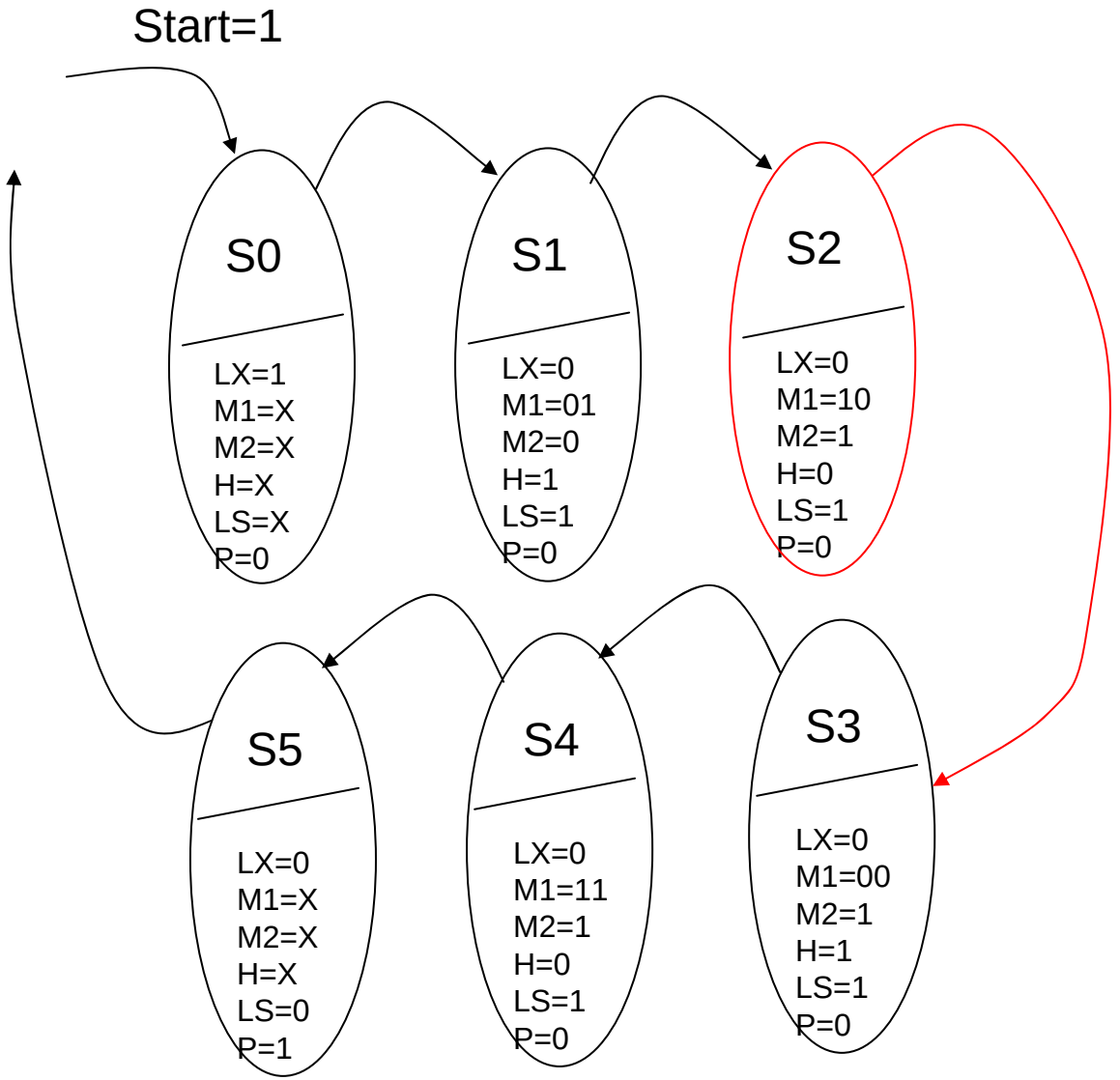




Exemplo 1

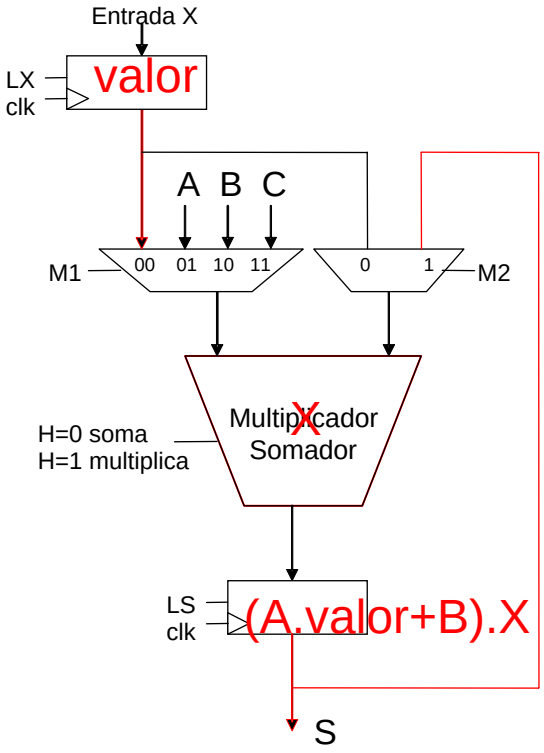
S= X (A.X + B) + C

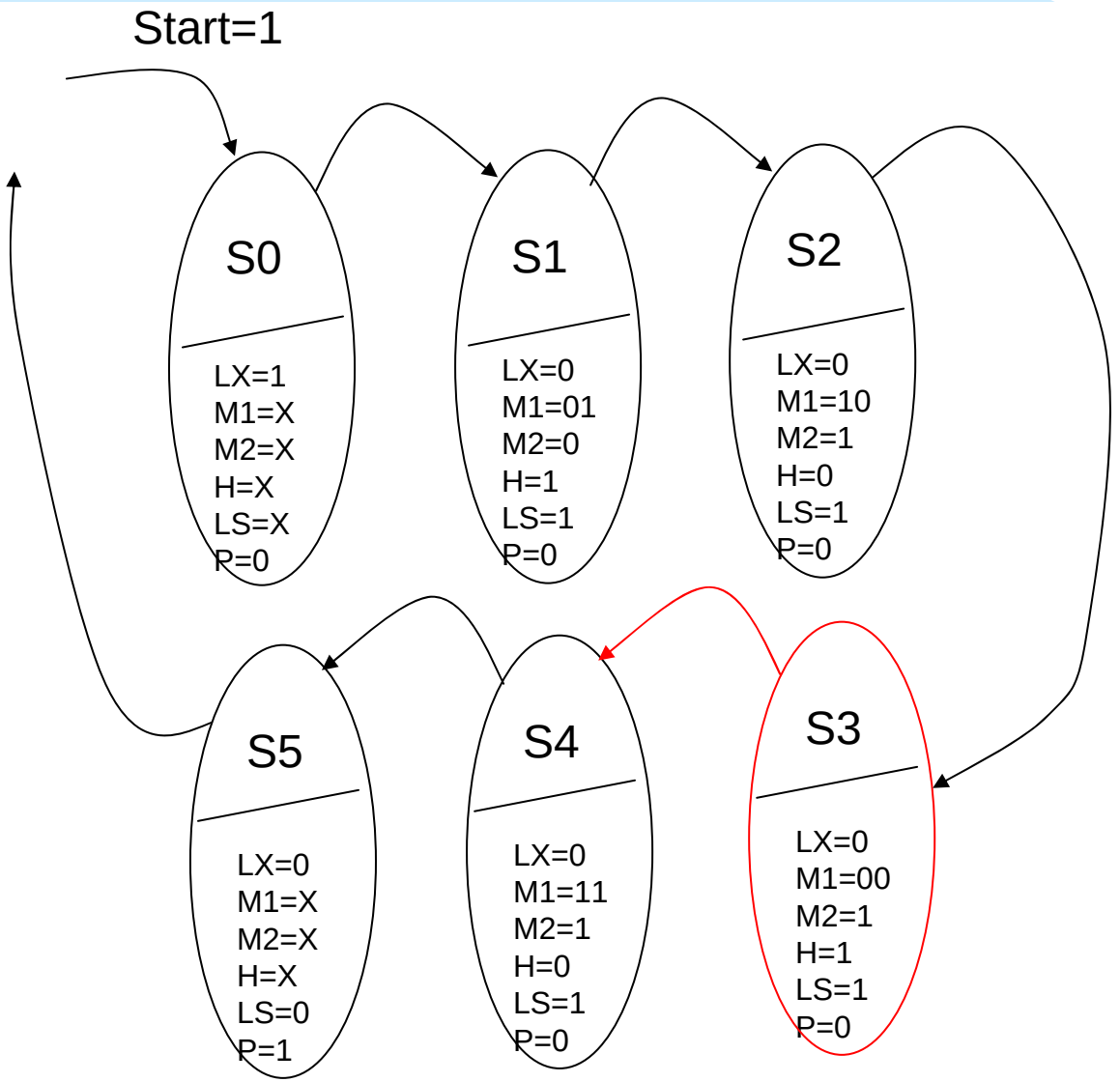




Exemplo 1

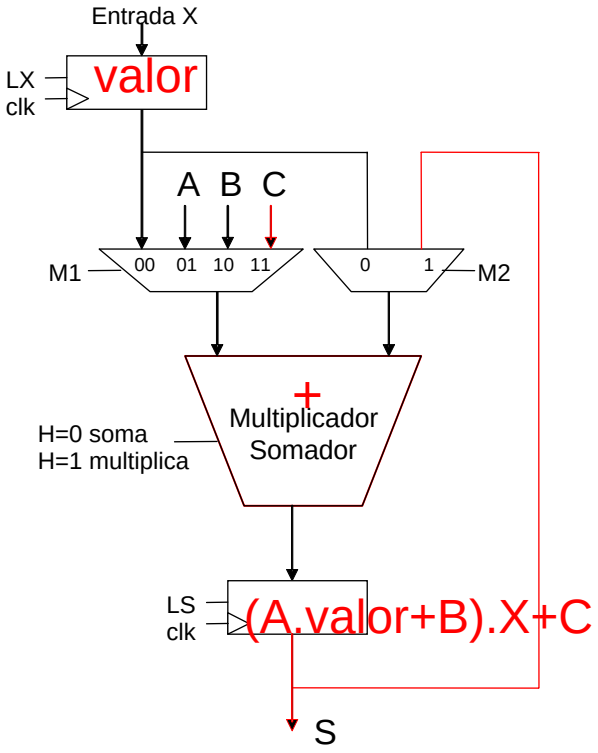
S= X (A.X + B) + C

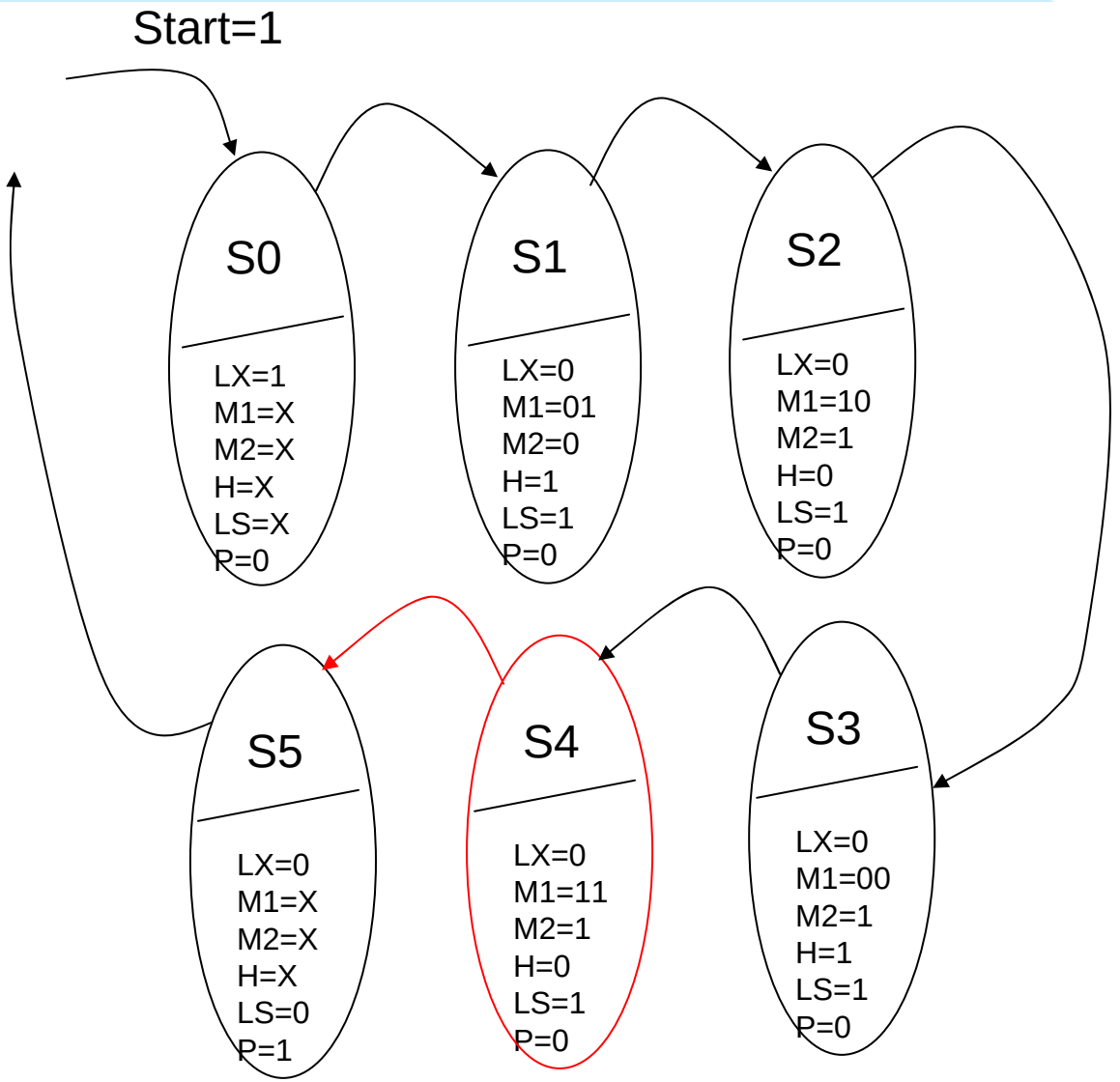




Exemplo 1

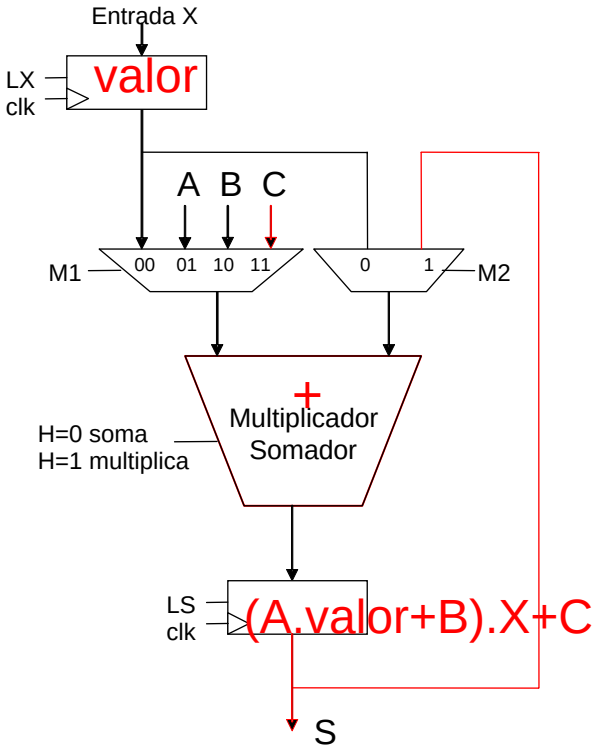
S= X (A.X + B) + C



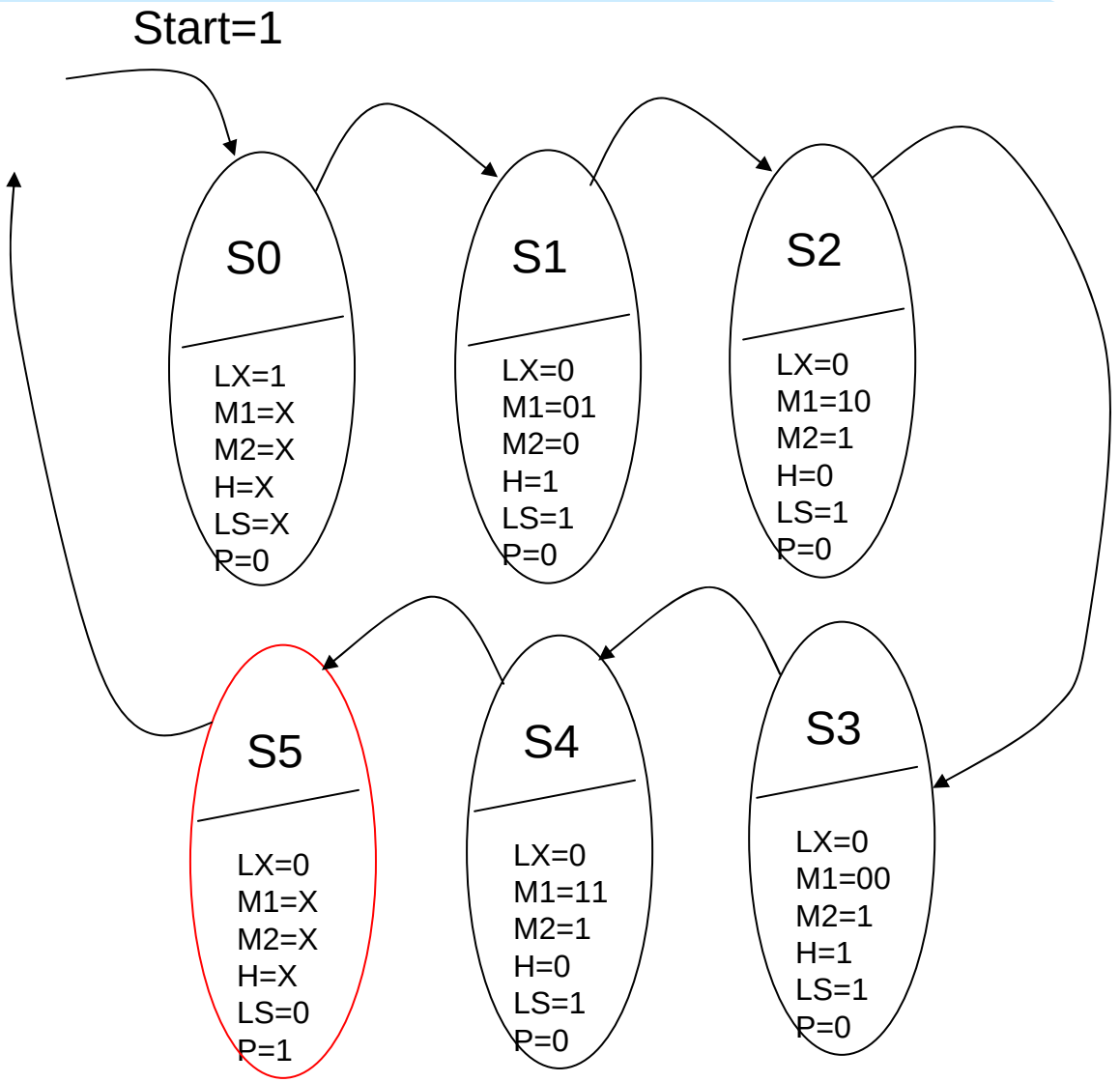


Exemplo 1

S= X (A.X + B) + C



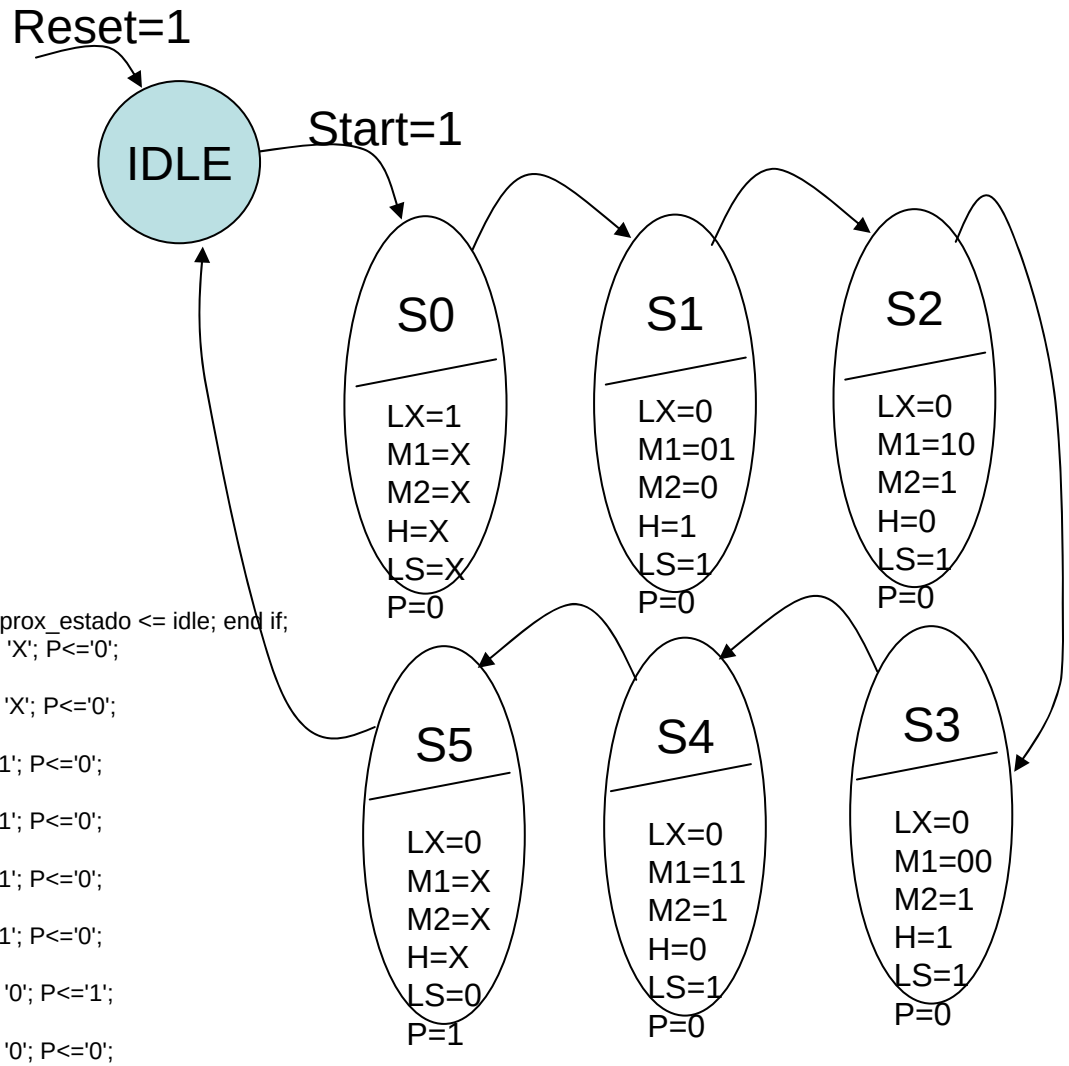
P=1 (FIM)



VHDL: Exemplo 1

RTL – VERSAO SERIAL

```
entity pc_funcao1 is
    Port ( reset : in  STD_LOGIC;
           start : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           LX : out  STD_LOGIC;
           LS : out  STD_LOGIC;
           M1 : out  STD_LOGIC_VECTOR(1 downto 0);
           M2 : out  STD_LOGIC;
           P : out  STD_LOGIC;
           H : out  STD_LOGIC);
end pc_funcao1;


architecture Behavioral of pc_funcao1 is
type tstate is (idle, S0, S1, S2, S3, S4, S5);
signal estado, prox_estado : tstate;
begin

process(reset, clk)
begin
if (reset='1') then
   estado <= idle;
elsif (clk'event and clk='1') then
   estado <= prox_estado;
end if;
end process;
process(estado, start)
begin
CASE estado IS
WHEN idle => if start='1' then prox_estado <= S0; else prox_estado <= idle; end if;
          LX<= 'X'; M1<="XX"; M2 <= 'X'; H <= 'X'; LS <= 'X'; P<='0';
WHEN S0 => prox_estado <= S1;
          LX<= '1'; M1<="XX"; M2 <= 'X'; H <= 'X'; LS <= 'X'; P<='0';
WHEN S1 => prox_estado <= S2;
          LX<= '0'; M1<="01"; M2 <= '0'; H <= '1'; LS <= '1'; P<='0';
WHEN S2 => prox_estado <= S3;
          LX<= '0'; M1<="10"; M2 <= '1'; H <= '0'; LS <= '1'; P<='0';
WHEN S3 => prox_estado <= S4;
          LX<= '0'; M1<="00"; M2 <= '1'; H <= '1'; LS <= '1'; P<='0';
WHEN S4 => prox_estado <= S5;
          LX<= '0'; M1<="11"; M2 <= '1'; H <= '0'; LS <= '1'; P<='0';
WHEN S5 => prox_estado <= idle;
          LX<= '0'; M1<="XX"; M2 <= 'X'; H <= 'X'; LS <= '0'; P<='1';
WHEN others =>  prox_estado <= idle;
          LX<= '0'; M1<="XX"; M2 <= 'X'; H <= 'X'; LS <= '0'; P<='0';
END CASE;
end process;
```



VHDL

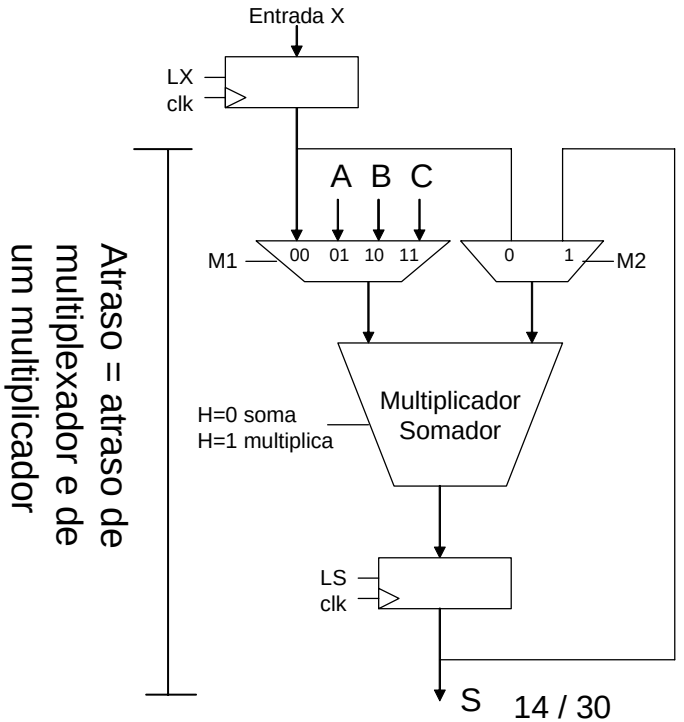
RTL

```
entity PO_funcao1 is
    Port ( reset : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           LX : in  STD_LOGIC;
           M1 : in  STD_LOGIC_VECTOR(1 downto 0);
           M2 : in  STD_LOGIC;
           LS : in  STD_LOGIC;
           H : in  STD_LOGIC;
           dado : in  STD_LOGIC_VECTOR (7 downto 0);
           A : in  STD_LOGIC_VECTOR (7 downto 0);
           B : in  STD_LOGIC_VECTOR (7 downto 0);
           C : in  STD_LOGIC_VECTOR (7 downto 0);
           Saida_funcao : out  STD_LOGIC_VECTOR (15 downto 0));
end PO_funcao1;
```

```
architecture Behavioral of PO_funcao1 is
signal dado_16, A_16, B_16, C_16, ula, mux1, mux2, regx, regs : std_logic_vector(15 downto 0);
begin

process(clk, reset)
begin
if reset='1' then
   regx <= "0000000000000000";
elsif (clk'event and clk='1') then
   if LX ='1' then  regx <= dado_16;
else regx <= regx;
end if; end if;
end process;

process(clk, reset)
begin
if reset='1' then
   regs <= "0000000000000000";
elsif (clk'event and clk='1') then
   if LS ='1' then    regs <= ula;
else regs <= regs;
end if; end if;
end process;

process(mux1, mux2, H)
begin
if H='0' then    ula <= mux1 + mux2;
else     ula <= mux1 * mux2;
end if;
end process;
```

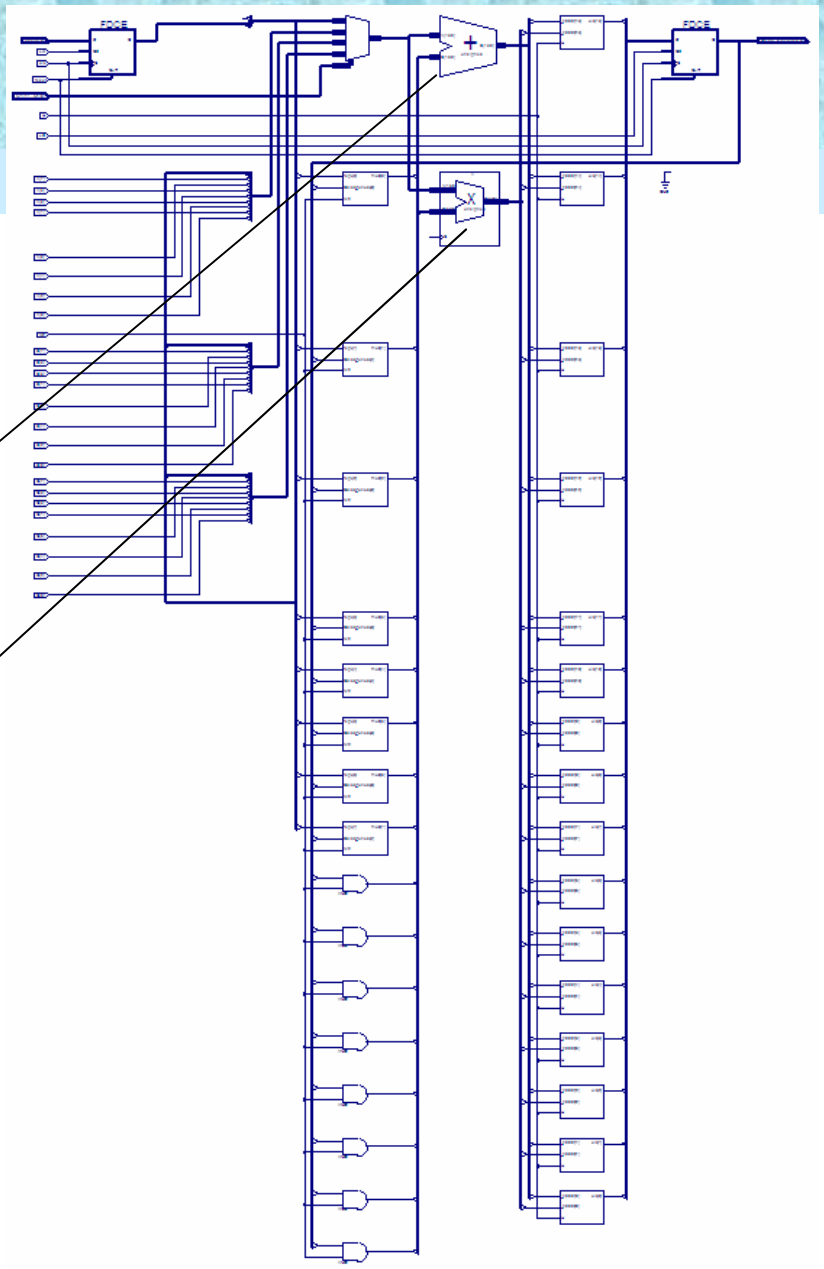
```
process(A_16, B_16, C_16, regx, M1)
begin
CASE M1 IS
WHEN "00" => mux1 <= regx;
WHEN "01" => mux1 <= A_16;
WHEN "10" => mux1 <= B_16;
WHEN others => mux1 <= C_16;
END CASE;
end process;

process(regx, regs, M2)
begin
if M2 = '1' then    mux2 <= regs;
else  mux2 <= regx;
end if;
end process;

saida_funcao <= regs;
A_16 <= "00000000"&A;
B_16 <= "00000000"&B;
C_16 <= "00000000"&C;
dado_16 <= "00000000"&dado;
end Behavioral;
```

Implementação SERIAL



Parte Operativa

XC2v80 (VirtexII)



Alocação de recursos:

Processamento serial

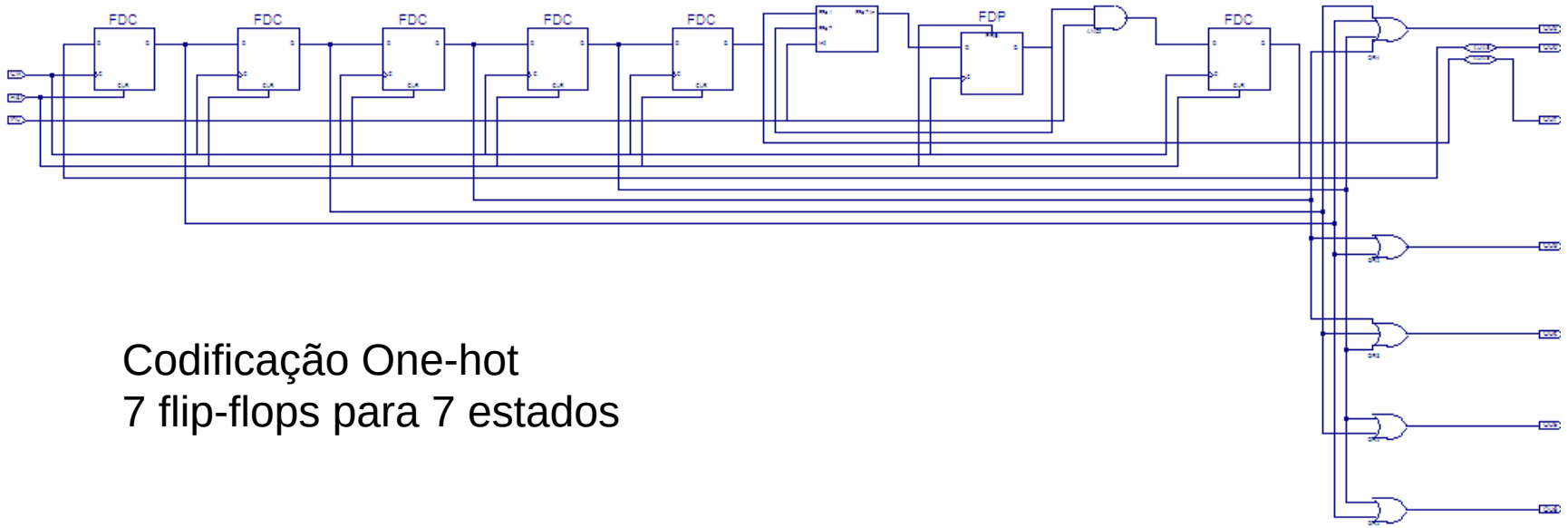
Somador/subtrator

Multiplicador

Parte de Controle

XC2v80 (VirtexII)

Estados: Idle, S0, S1, S2, S3, S4, S5



Codificação One-hot
7 flip-flops para 7 estados

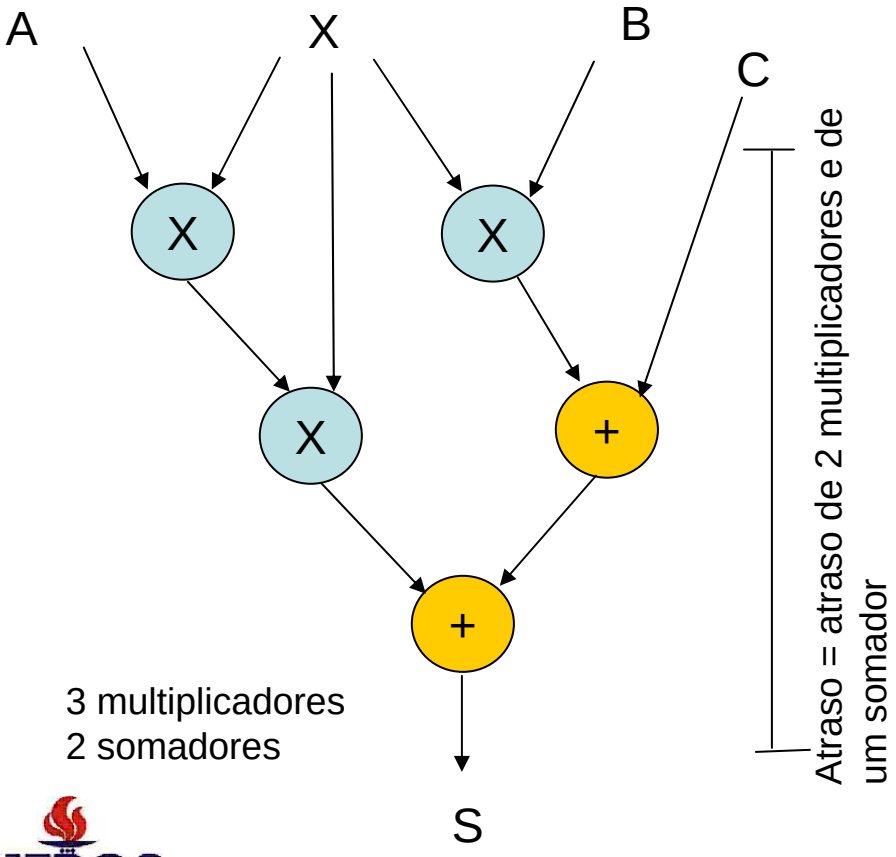
Periodo mínimo de clk = 2ns

Descrição em VHDL: Exemplo 1

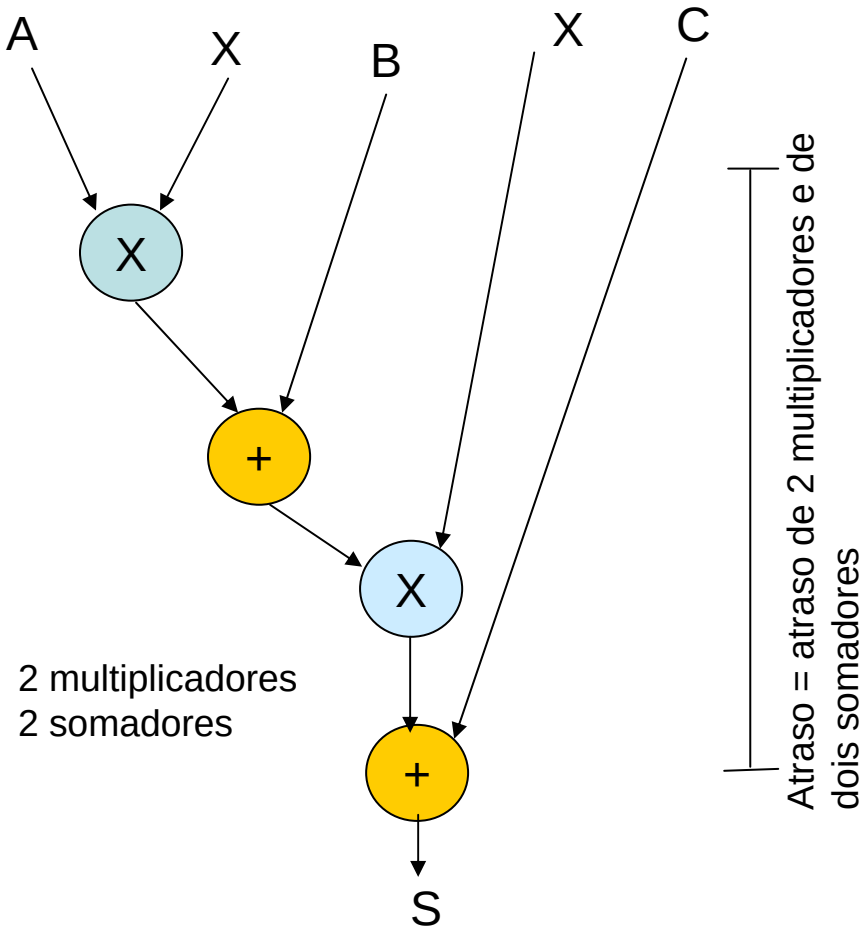
VERSÃO PURAMENTE COMBINACIONAL

- Paralelismo máximo

$S = A.X^2 + B.X + C$

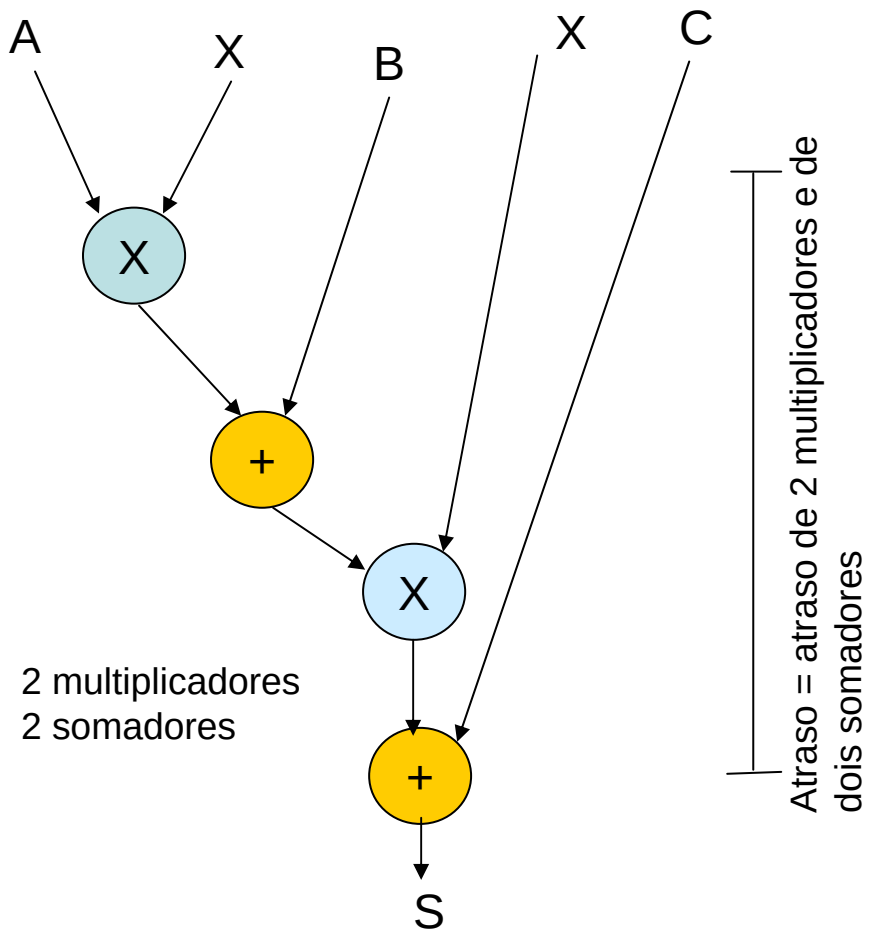


$S = X.(A.X + B) + C$



```
entity funcao1_comb is
    Port ( dado : in STD_LOGIC_VECTOR(7 downto 0);
           clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
           start : in STD_LOGIC;
           a : in  STD_LOGIC_VECTOR(7 downto 0);
           b : in  STD_LOGIC_VECTOR(7 downto 0);
           c : in  STD_LOGIC_VECTOR(7 downto 0);
           saida_funcao : out STD_LOGIC_VECTOR(15 downto 0));
end funcao1_comb;

architecture Behavioral of funcao1_comb is

signal dado_16, A_16, B_16, C_16, regx, regs : std_logic_vector(15 downto 0);

begin

saida_funcao <= regs;
A_16 <= "00000000"&A;
B_16 <= "00000000"&B;
C_16 <= "00000000"&C;
dado_16 <= "00000000"&dado;

process(clk, reset)
begin
if reset='1' then
   regx <= "0000000000000000";
elsif (clk'event and clk='1') then
   if start ='1' then
   regx <= dado_16;
 else  regx<=regx;
end if; end if;
end process;

process(clk, reset)
begin
if reset='1' then
   regs <= "0000000000000000";
elsif (clk'event and clk='1') then
   if start='0' then
   regs <= (((A_16 * regx) + B_16)* regx ) + C_16;
else
    regs <= regs;
end if;
end if;
end process;

end Behavioral;
```

$$S = X.(A.X + B) + C$$



Descrição em VHDL: Exemplo 1

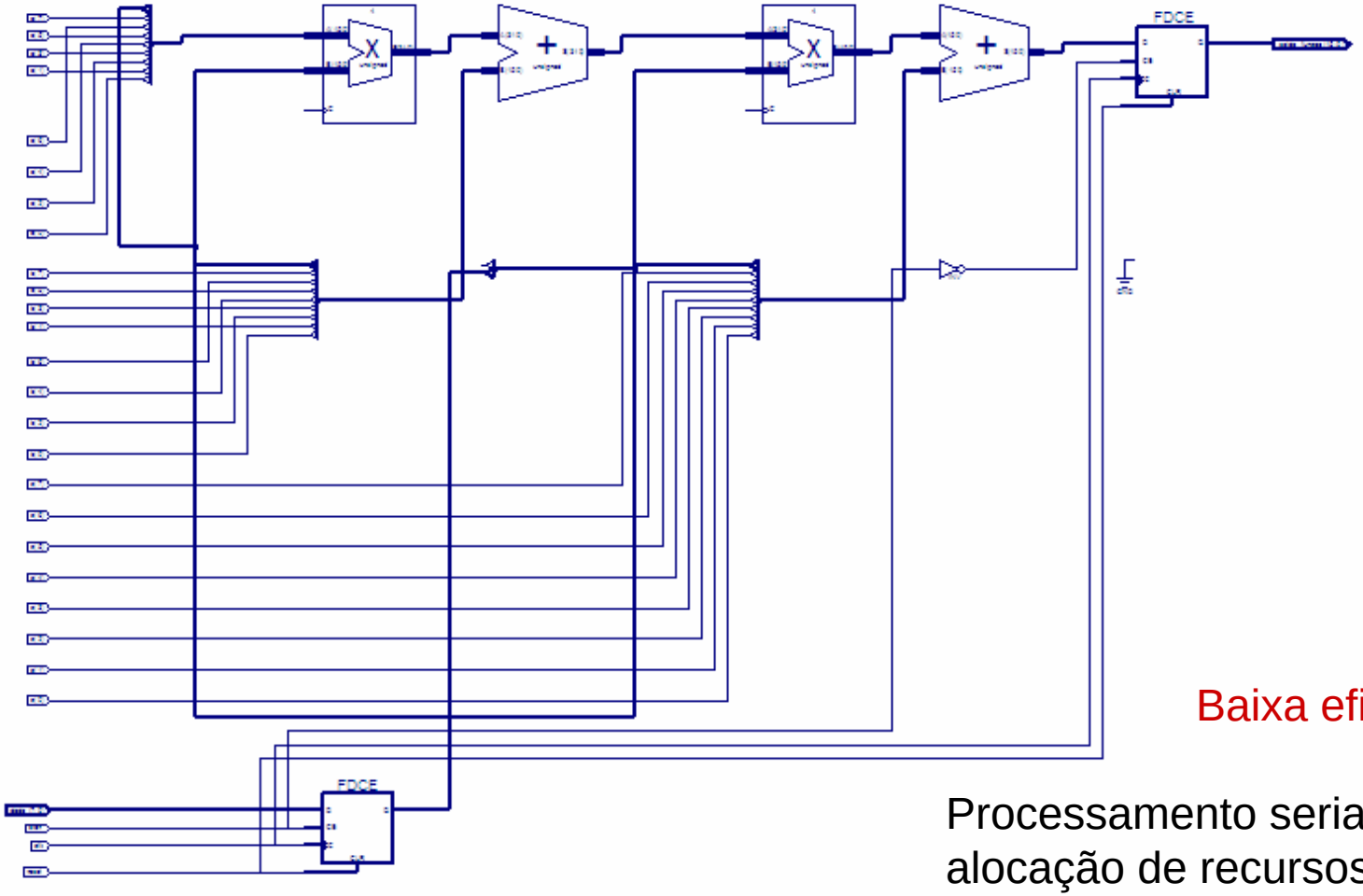
RTL – versão 2

```
entity funcao1_altonivel is
    Port ( reset : in  STD_LOGIC;
           clk : in  STD_LOGIC;
                                                    start : in std_logic;
           dado : in  STD_LOGIC_VECTOR(7 downto 0);
           A, B, C : in  STD_LOGIC_VECTOR(7 downto 0);
           saida_funcao : out  STD_LOGIC_VECTOR(15 downto 0));
end funcao1_altonivel;

architecture Behavioral of funcao1_altonivel is

signal dado_16, A_16, B_16, C_16, regx, regs : std_logic_vector(15 downto 0);
signal cont : std_logic_vector(1 downto 0);
begin


saida_funcao <= regs;
A_16 <= "00000000"&A;
B_16 <= "00000000"&B;
C_16 <= "00000000"&C;
dado_16 <= "00000000"&dado;

process(clk, reset)
begin
if reset='1' then
   regx <= "0000000000000000";
elsif (clk'event and clk='1') then
   if start ='1' then
   regx <= dado_16;
                        else
                        regx<=regx;
                        end if;
end if;
end process;
```

```
process(reset, clk)
begin
if (reset='1' or start='1') then
  cont <= "00";
elsif clk'event and clk='1' then
 if start='0' then
  cont <= cont +1;
  end if;
end if;
end process;

process(clk, reset)
begin
if reset='1' then
   regs <= "0000000000000000";
elsif (clk'event and clk='1') then
   if start='0' then
                        CASE CONT IS
                         WHEN "01" => regs <=  A_16 * regx;
                         WHEN "10" =>  regs <= regs + B_16;
                         WHEN  "11" => regs <= regs * regx;
                         WHEN others => regs <= regs + C_16;
                        END CASE;
else
regs <= regs;
end if;
end if;
end process;

end Behavioral;
```

Descrição em VHDL: Exemplo 1

RTL – versão 2



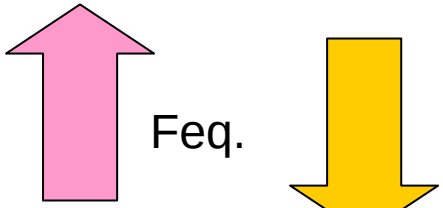
Baixa eficiência

Processamento serial mas alocação de recursos paralelo

Comparação de área e desempenho

XC2v80 (VirtexII)

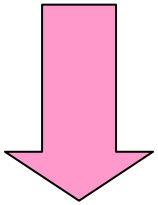
Versão RTL (serial)
PC => 7 LUTS e 7 Flip-flops
PO => 78 LUTs e 24 flip-flops e 1 MULT18x18
=> periodo mínimo 6.8ns

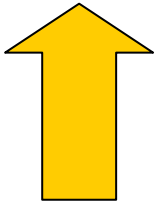


Resposta lenta

6x 6.8ns=40ns

Versão COMB e paralela
=> 30 LUTS, 24 flip-flops e 2 MULT18x18
=> periodo mínimo 12.4 ns





Resposta rápida

12.4ns

Devido ao caminho crítico
E ao paralelismo x serial

Exemplo 2

Implementar um hardware para realizar as seguintes operações:

$F(x) = (A.\ x^2 + B)/4 + C$

Máximo desempenho
(comprometimento em área)

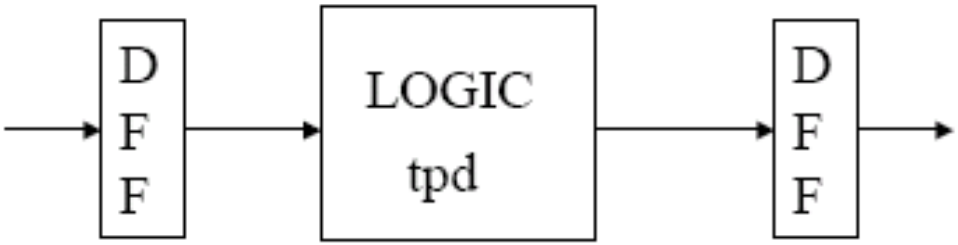
```
inicio: X <= novo valor
        Start =1;
        Wait until done=1
       go to inicio;
```

Mínima área
(comprometimento em desempenho)

Aumentando mais o desempenho

- The register-to-register delay is usually the delay path that sets the maximum clock rate
- From a design point of view, can only affect the combinational logic between the registers
  - Need to shorten the maximum combinational delay path
  - Setup/Hold time of registers are fixed
- Can shorten the delay by placing a register in the combinational logic to break longest delay path
  - This technique is called *pipelining*
  - Adds *latency* to the output (the number of clocks between an input value and its corresponding output result)

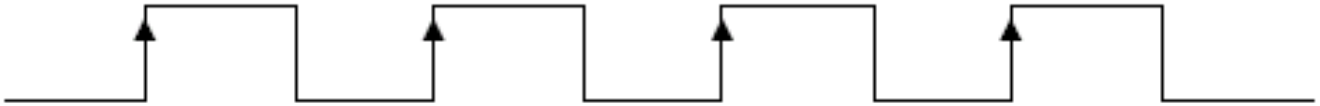
Registered Datapath

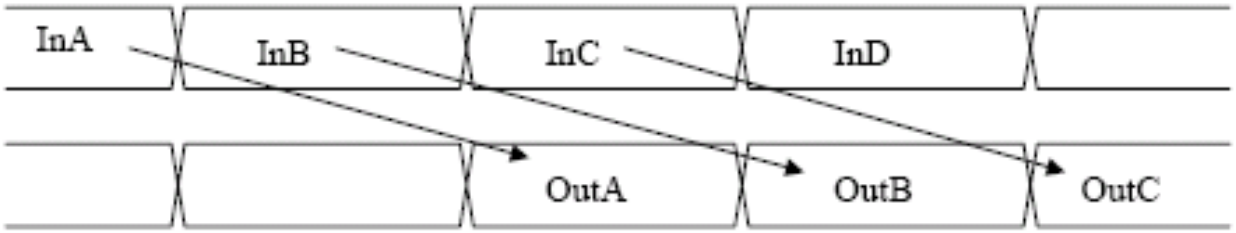


DFFs are rising edge triggered

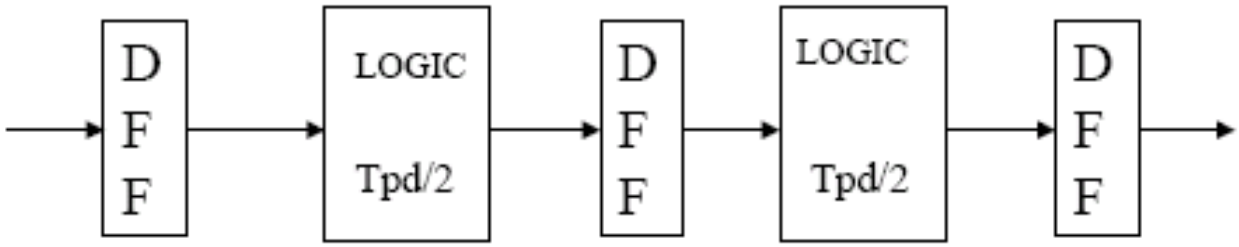
Clk Freq = 1/ (Tclk2q + Tpd + Tsu)

Latency = 1 clk



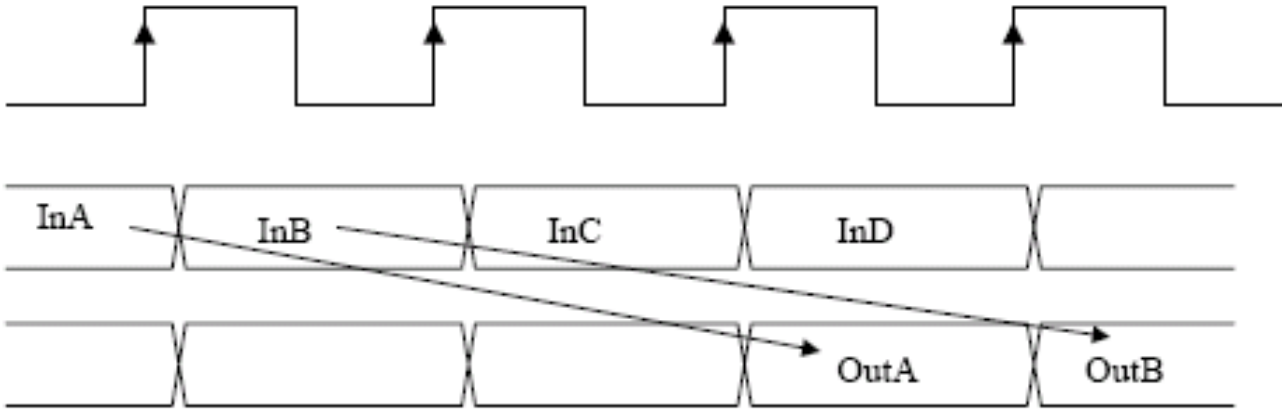


Add a pipeline stage



Clk Freq = 1/ (Tclk2q + Tpd/2 + Tsu)

Latency = 2 clks



Definitions

Initiation Rate - Rate at which new input values are accepted after first output value

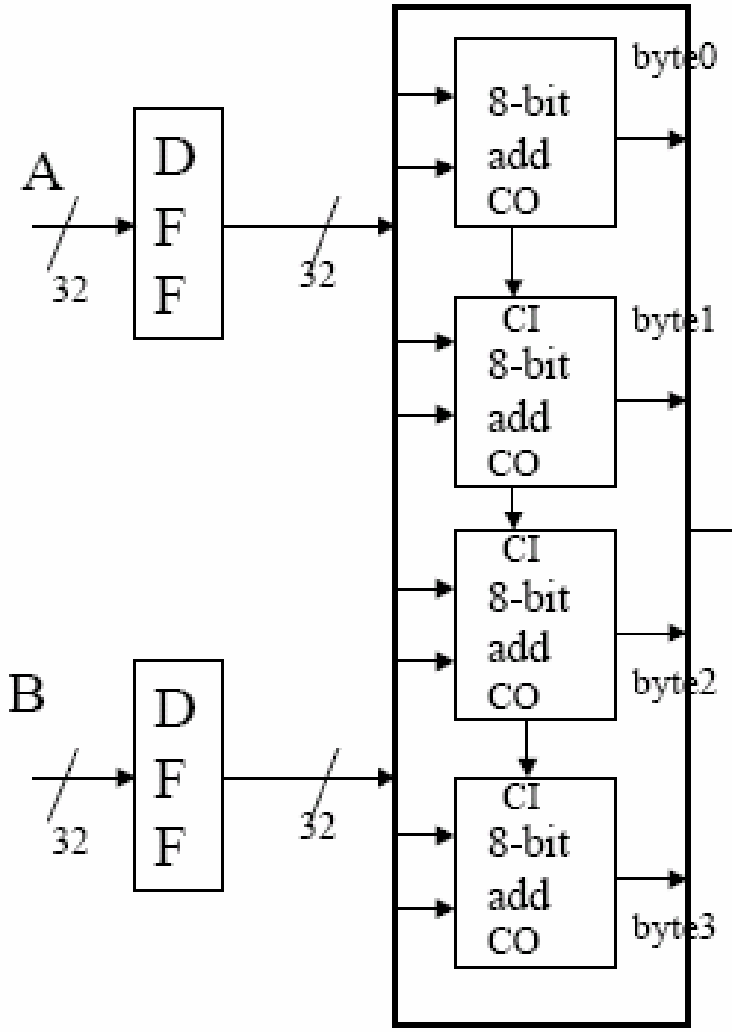
- Rate at which new computations are initiated
- Maximum initiation rate = 1

Latency

- Number of clock cycles between input value and output value
- Adding pipeline stages always increases latency

At some point, adding more pipeline stages does not increase clock frequency because Tclk2q and Tsu dominate delay.

Pipeline Example



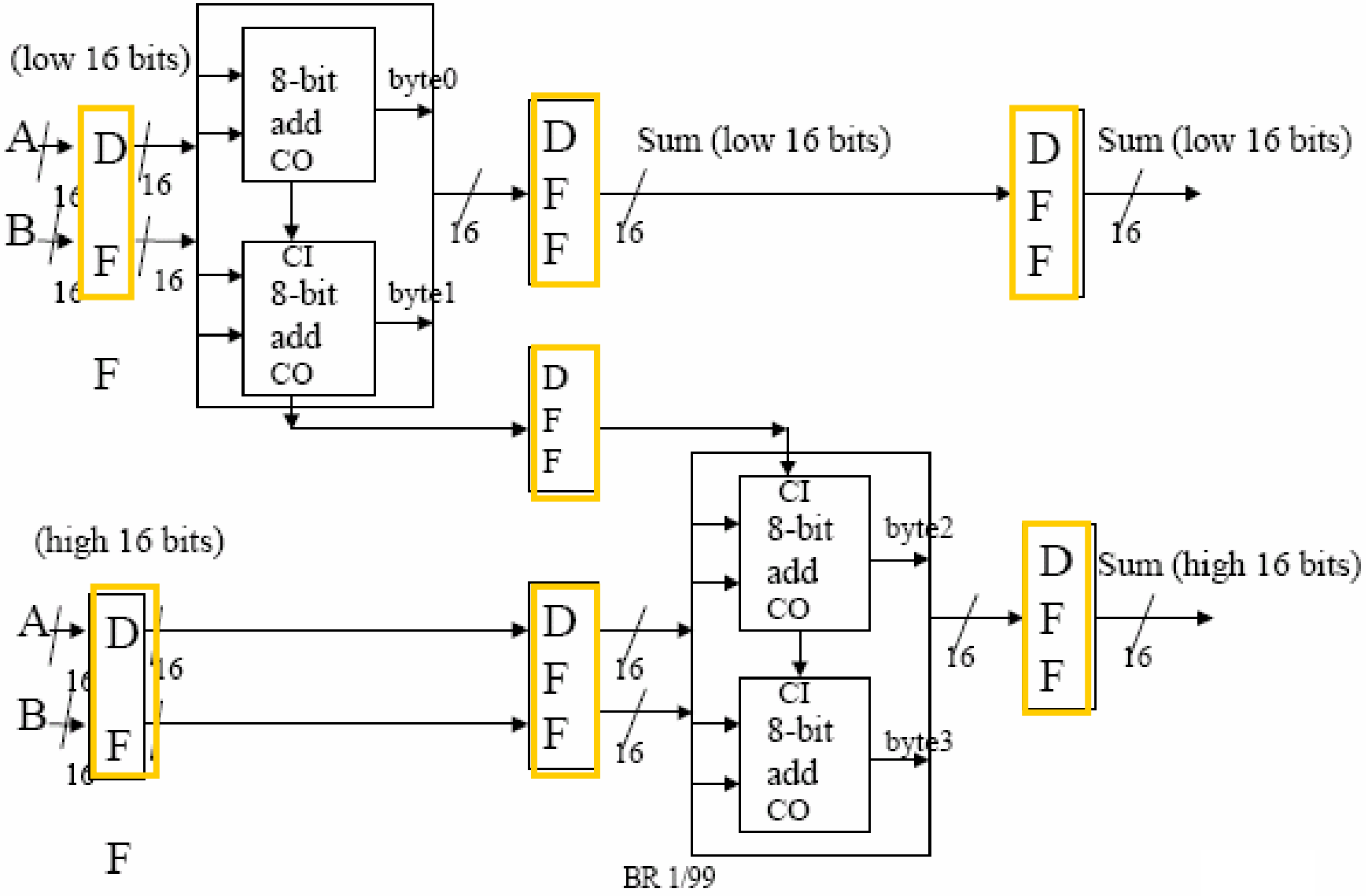
32-bit ripple carry adder.

Longest delay path in carry chain from byte0 to byte3

Can I use pipelining to speed this up?

BR 1/99

Insert pipeline stage between byte1 and byte2.



- Note that the pipeline stage broke the carry chain into two equal paths
- Each pipeline stage should have approximately the same combinational delay
- Clock speed will be set by the delay of the slowest pipeline stage

- If I inserted 2 pipeline stages, I would need to break the carry chain delay into equal thirds
- Could insert a pipeline stage between each BIT in order to get maximum clock speed
  - Called *'bit pipelining'*

Latency tolerance

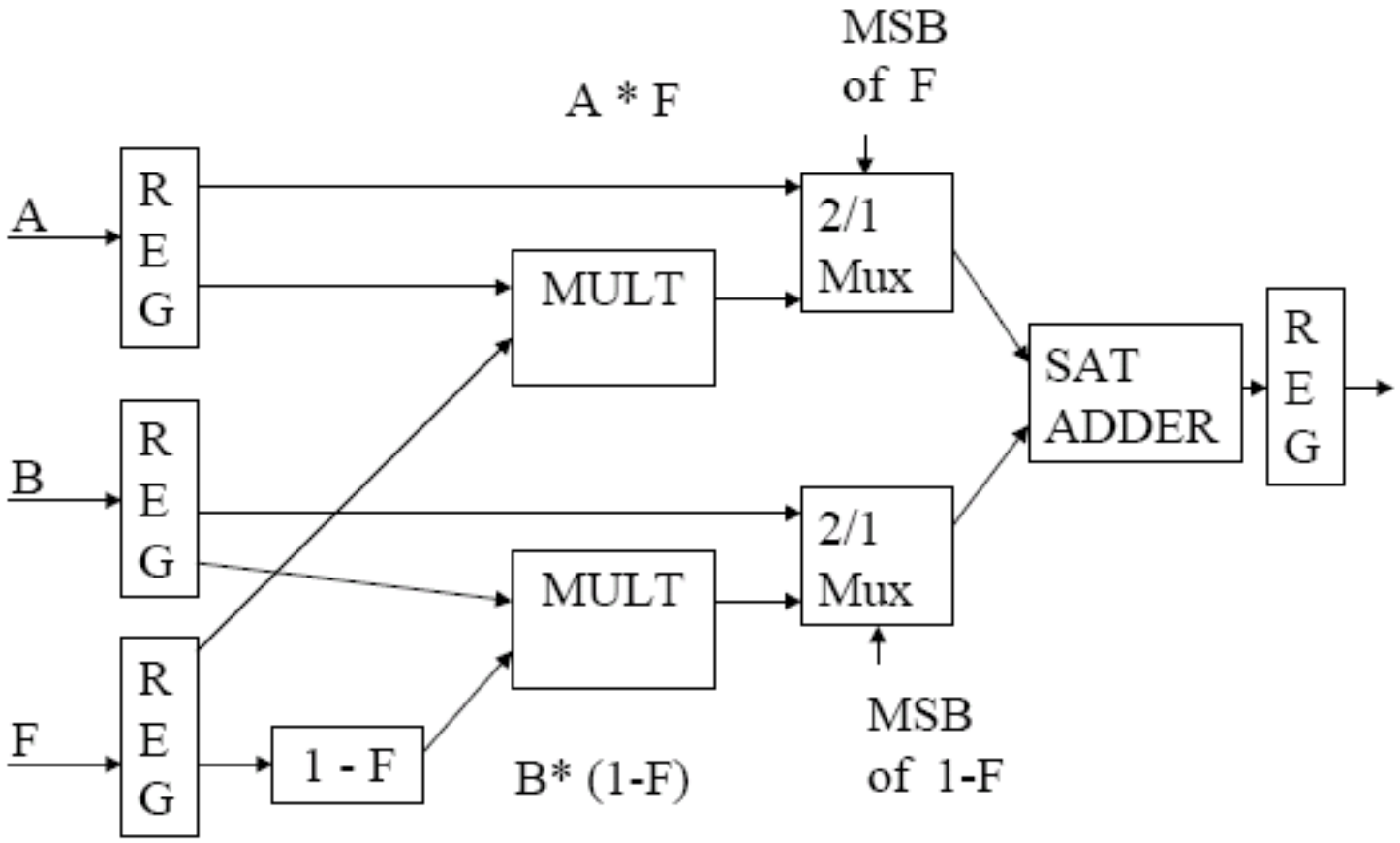
- Latency tolerance is dependent upon each application
- Frequent flushing of a pipeline (discarding partial results within the pipeline and restarting the pipeline with a new value) wastes time and makes an application latency intolerant.
- Flushing of a pipeline introduces clock cycles in which the results coming out of the pipeline are ignored -- these are wasted clock cycles.
- High Latency tolerance means that you can have many pipeline stages, whatever the number you need to meet the clock rate specification.

Two Examples

- Graphics hardware for processing pixels is extremely latency tolerant - not unusual to find pipelines that have 10's of stages.
  - Graphics pipelines are never flushed
  - High clock rate is EXTREMELY important because of large number of pixels (> 1 Million) that have to be supplied every screen, at > 30 updates per second
- Microprocessor instruction pipelines are not very latency tolerant - most CPU pipelines are only about 5-10 stages.
  - Branch instructions can cause pipeline to be flushed.
  - By the time you determine direction of branch, may have started processing instructions that should not be in the pipeline. These are flushed and the pipeline restarted.

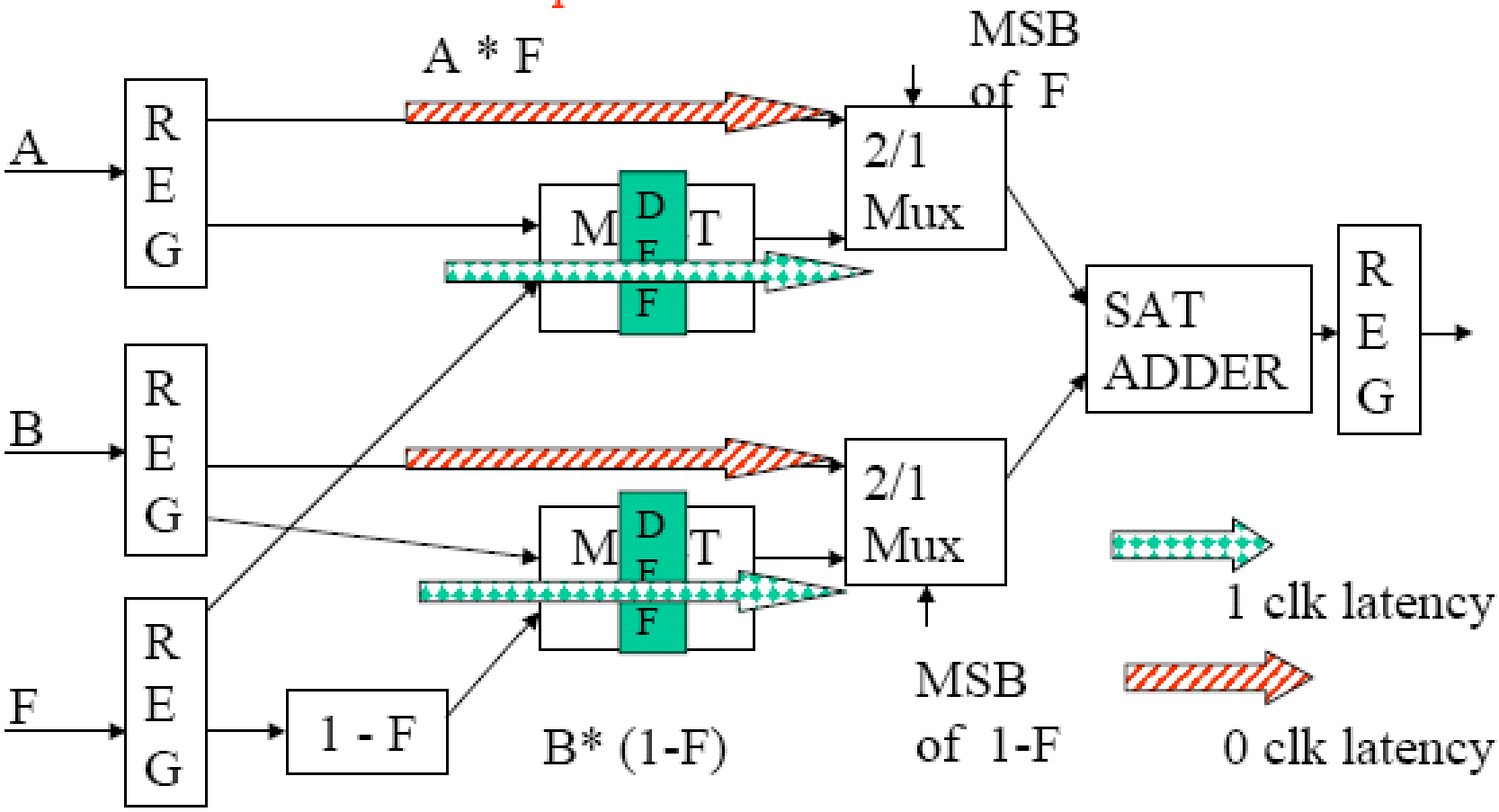
Datapath Pipeline

BLEND Datapath without Pipelining. MULT is combinational.

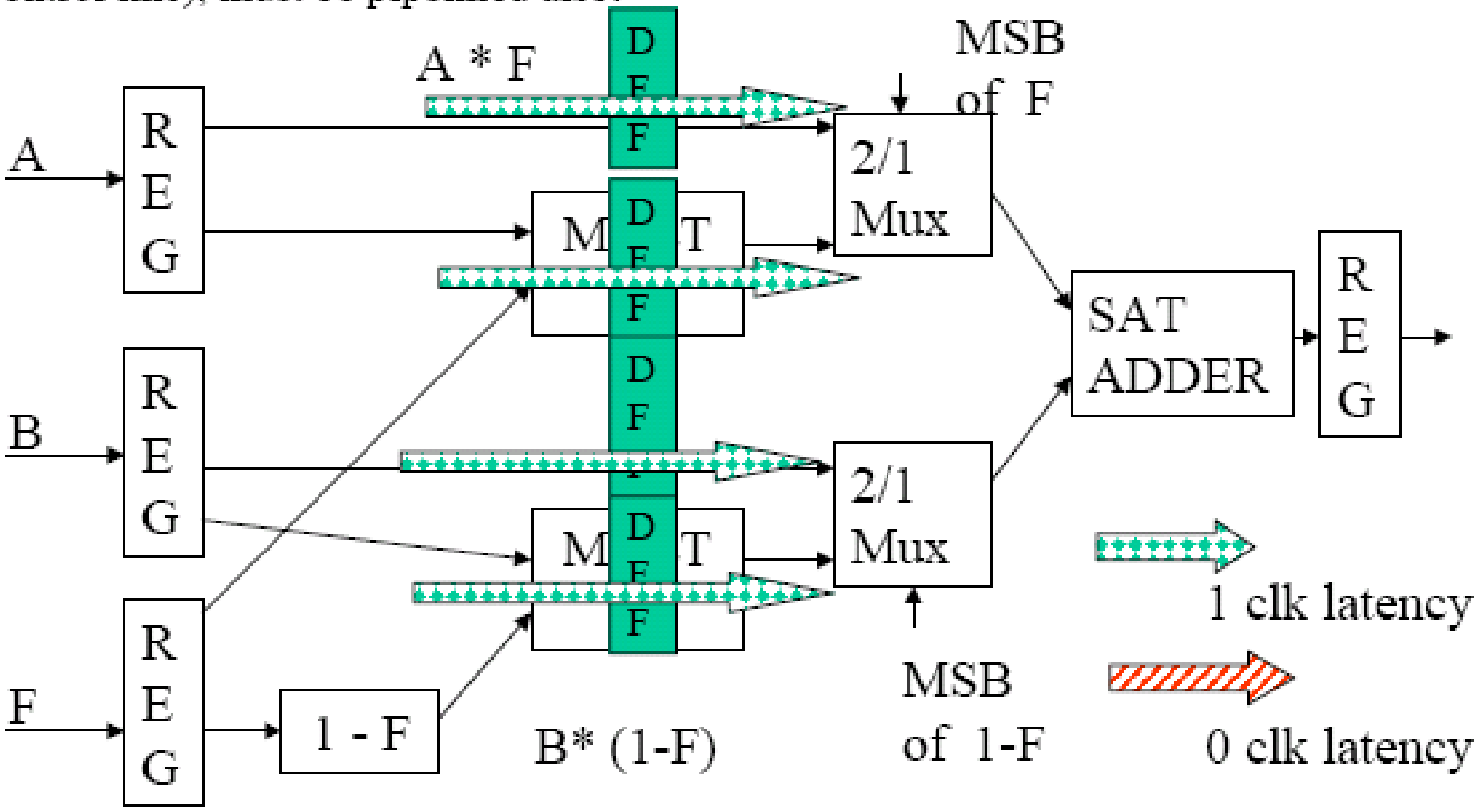


BR 1/99

LPM_Mult can be pipelined via LPM_PIPELINE parameter. Add one pipeline stage to LPM Mult. DFFs inserted automatically in LPM_MULT. We now have a LATENCY mismatch within our datapath!!!!

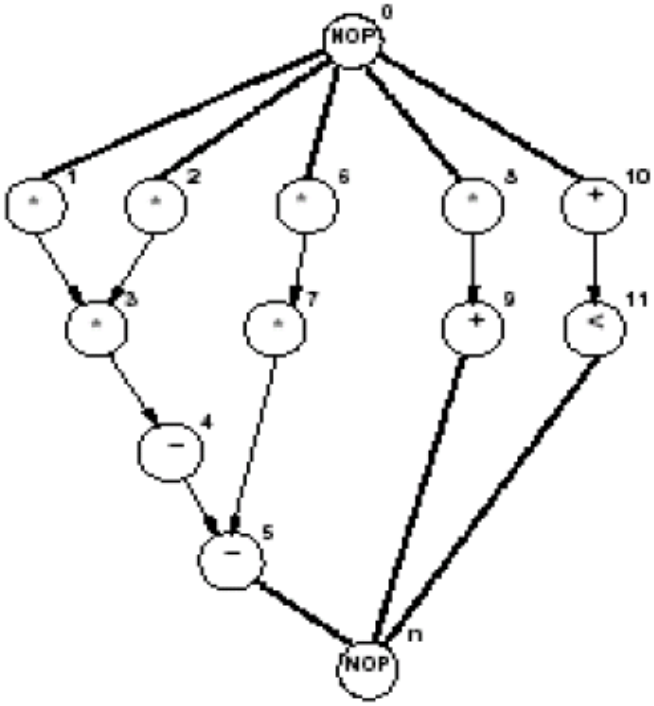


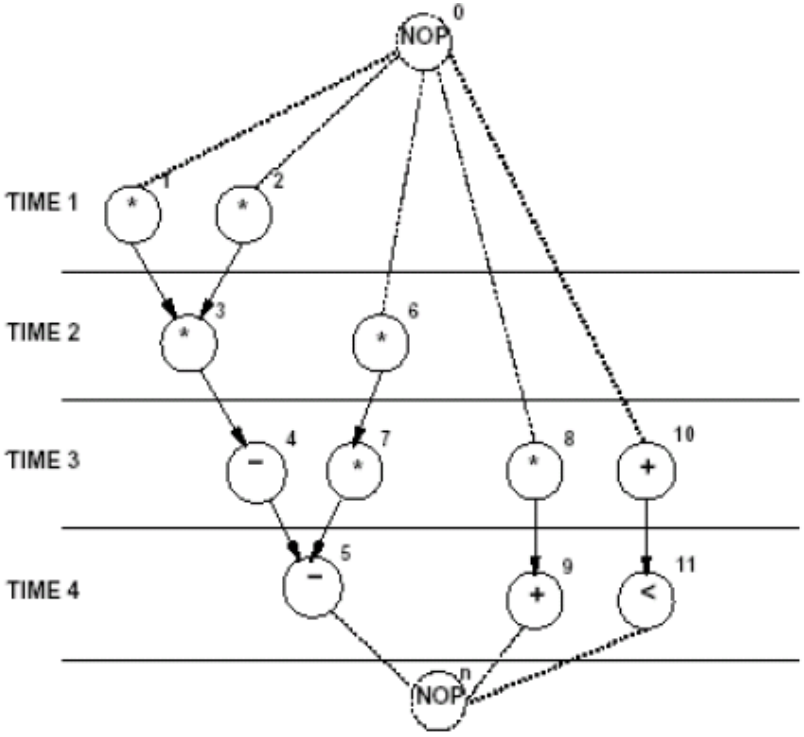
Correct the latency mismatch by adding DFFs in other path as well. May have to break delay paths in other places or add additional pipeline stages to LPM_mult to meet clock frequency target. Control lines (like MUX control line), must be pipelined also.



Exemplo de Scheduling ALAP

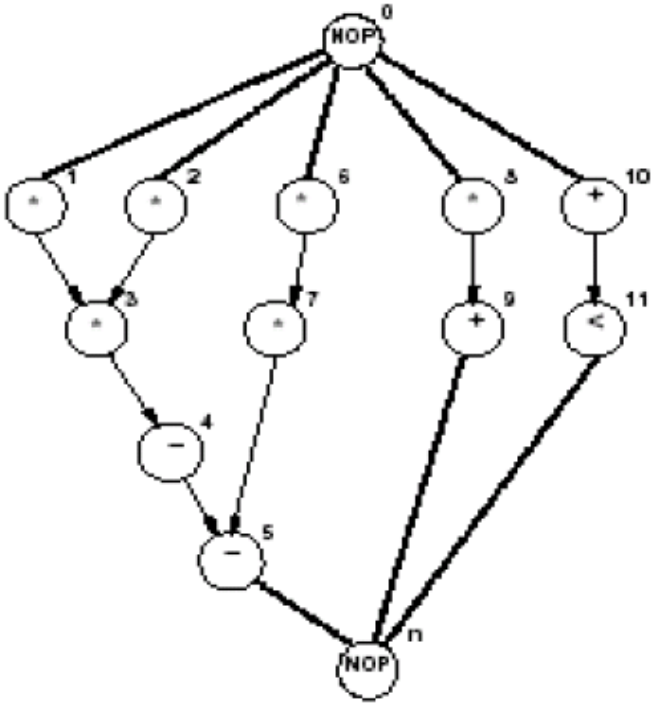
Control/Data Flow Graph (CDFG)

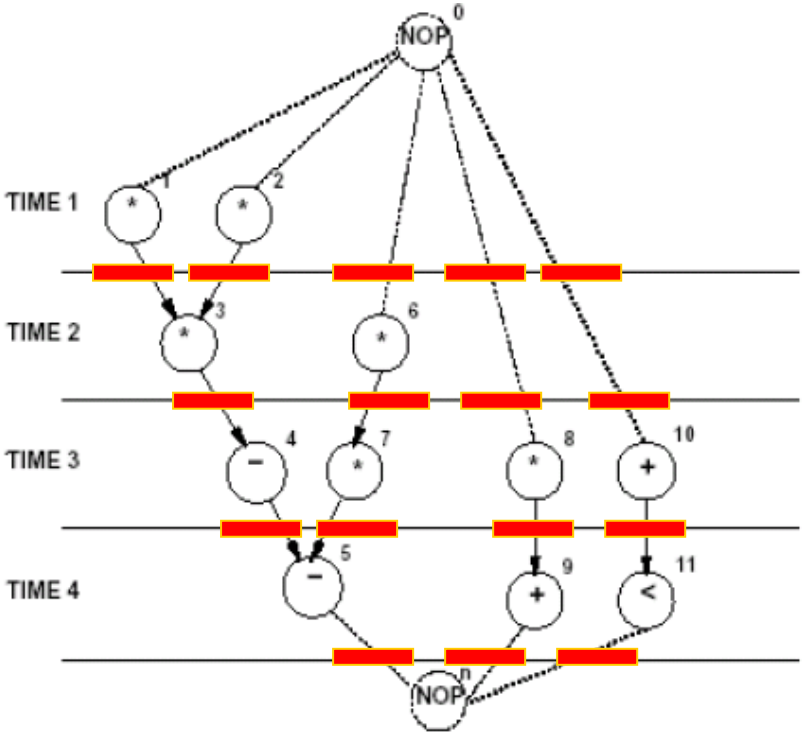




Exemplo de Scheduling ALAP

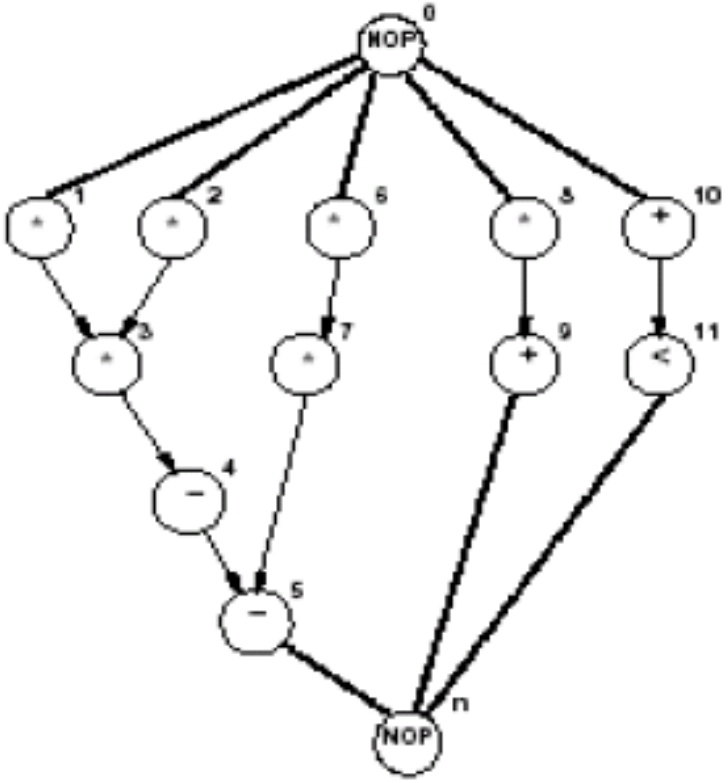
Control/Data Flow Graph (CDFG)

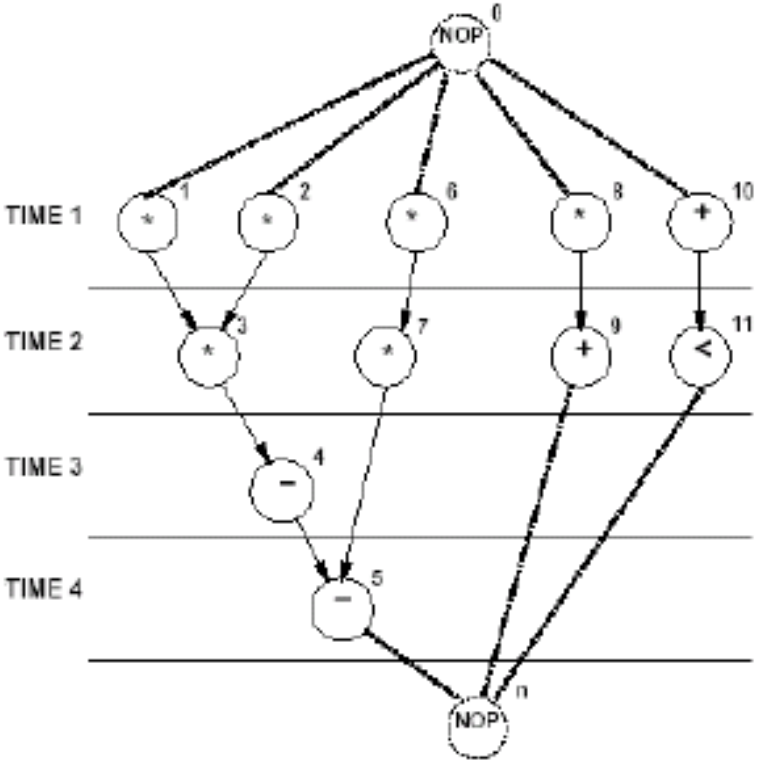




Exemplo de Scheduling ASAP

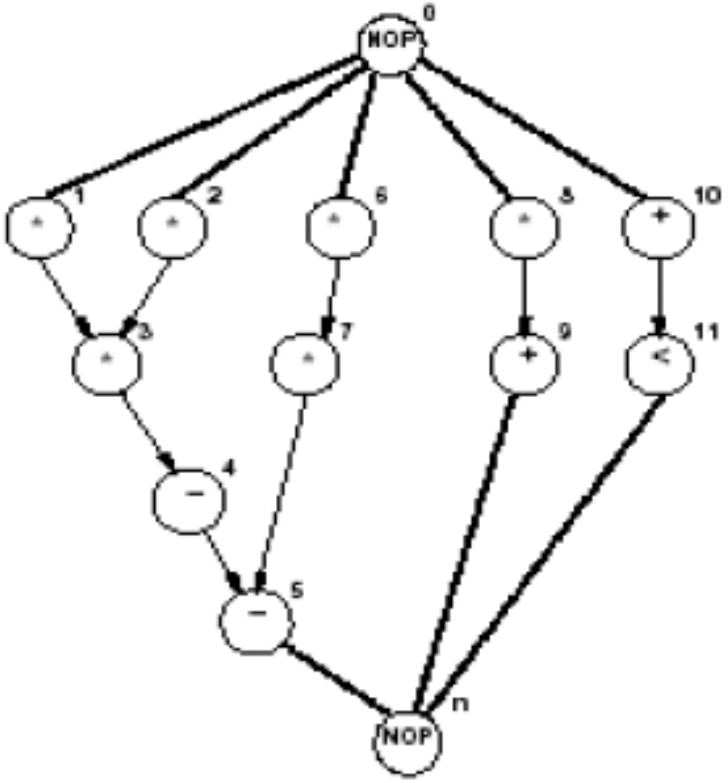
Control/Data Flow Graph (CDFG)

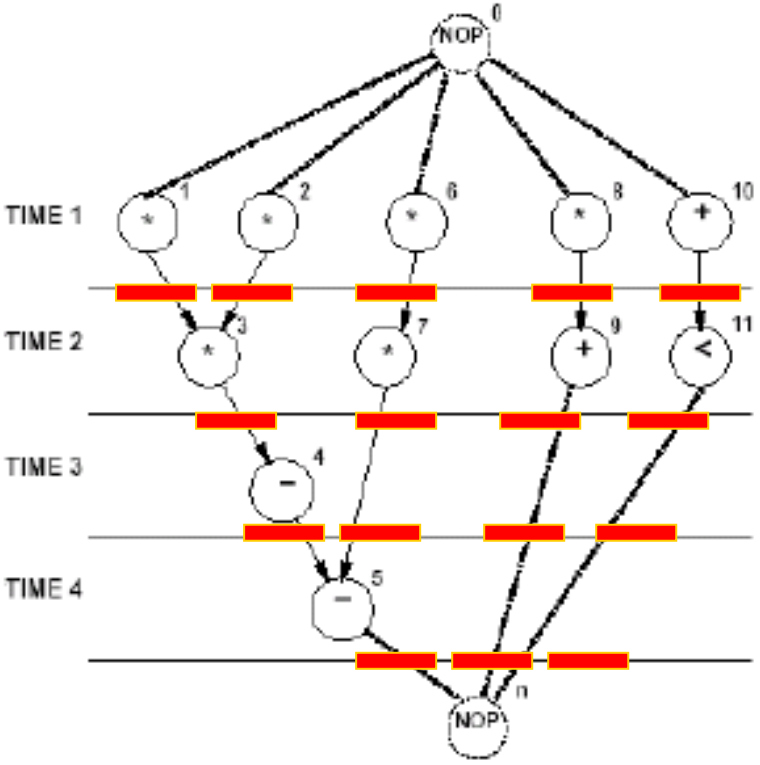




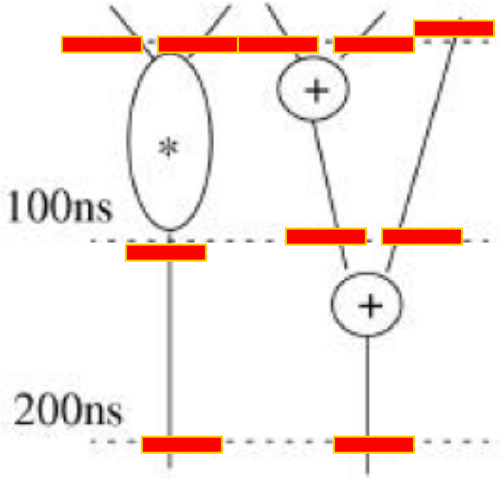
Exemplo de Scheduling ASAP

Control/Data Flow Graph (CDFG)

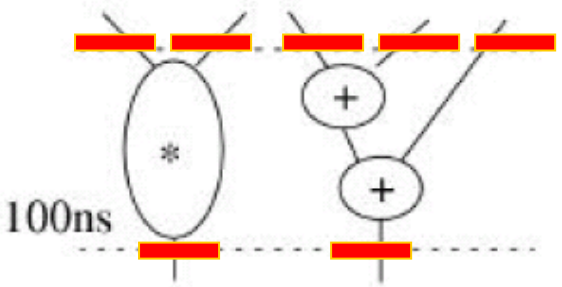




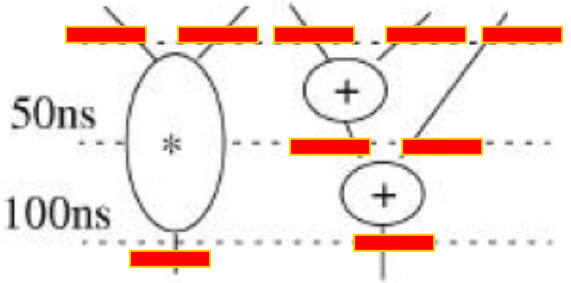
Scheduling Avançado



(a) No chaining or multicycling



(b) Two chained additions



(c) A multicycle multiplication

Estudo de Caso

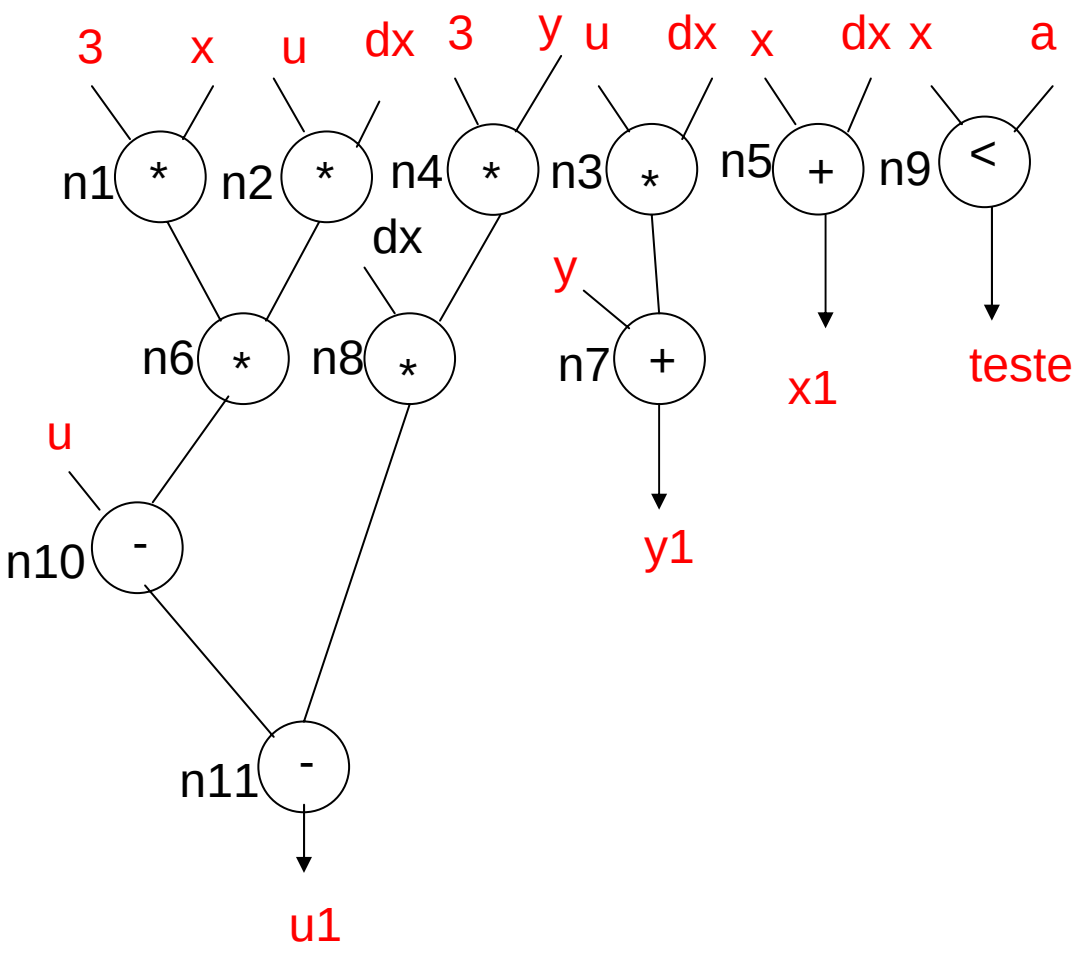
Algoritmo:

```
int diffeq(int x, int y, int u, int dx, int a) {
int x1, u1, y1;
while ( x < a ) {
x1 = x + dx;
u1 = u - (3 * x * u * dx) - (3 * y * dx);
y1 = y + u * dx;
x = x1;
u = u1;
y = y1;
}
return y;
}
```

a é uma constante

Pseudo-codigo e dependência de dados


```
Enquanto (x<a){
    x1= x +dx;
    u1= u – 3*x*u*dx – 3*y*dx;
    y1= y + u*dx;
    x = x1;
    y = y1;
    u = u1;
}
```



Paralelismo inerente ao processo da aplicação

Analise de Recursos de Hardware

- Depende do paralelismo implementado:
  - Implementação puramente combinacional
  - Implementação sequencial (certo número de ciclos de relogio para completar o algoritmo) - Escalonamento
  - Inserção de pipeline no fluxo sequencial de operação.

```
Enquanto (x<a){
    x1= x +dx;
    u1= u – 3*x*u*dx – 3*y*dx;
    y1= y + u*dx;
    x = x1;
    y = y1;
    u = u1;
}
```

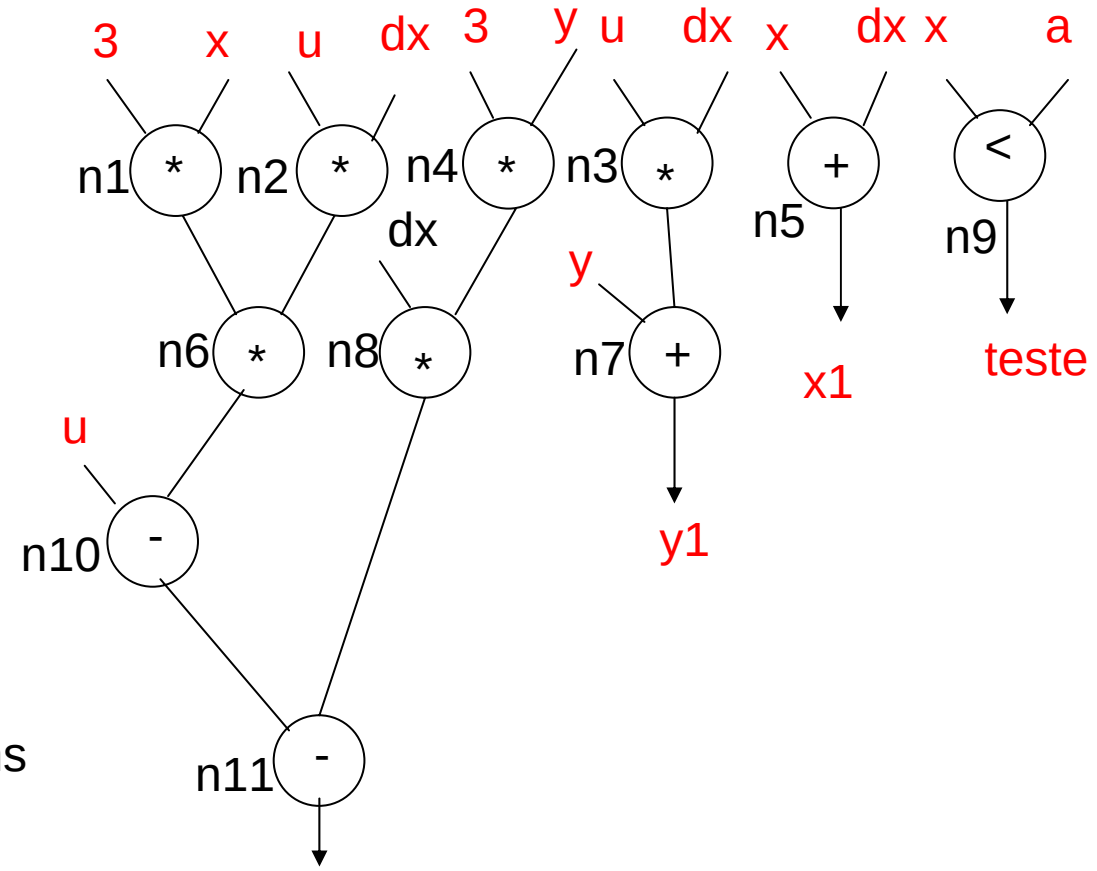
Paralelismo dos Operadores

Puramente Combinacional

- Definir a quantidade de hardware necessária para a execução da aplicação.

Opção 1:
Tudo combinacional

- 6 multiplicadores
- 2 somadores
- 3 subtradores
- 1 comparador



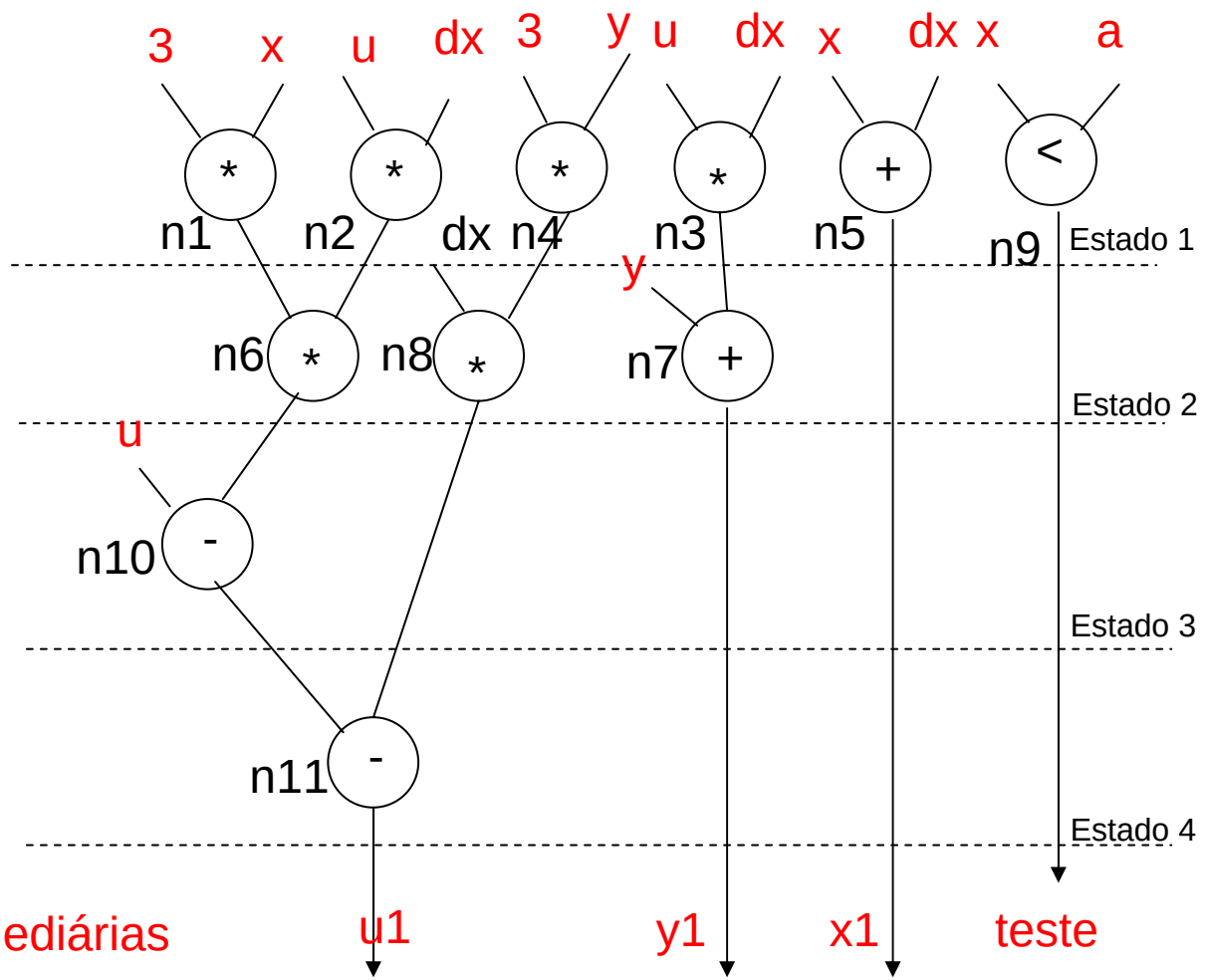
Atraso caminho critico:
2 multiplicadores + 2 subtratores

Ex: 100ns + 100ns + 30ns + 30 ns
= 260 ns

Paralelismo dos Operadores

Sequencial (sem limite de recursos)

```
enquanto (x<a){
    x1= x +dx;
    u1= u – 3*x*u*dx – 3*y*dx;
    y1= y + u*dx;
    x = x1;
    y = y1;
    u = u1;
}
```



Opção 2:
Execução por estados!!!

Necessita de variáveis intermediárias

Paralelismo dos Operadores

Sequencial (sem limite de recursos)

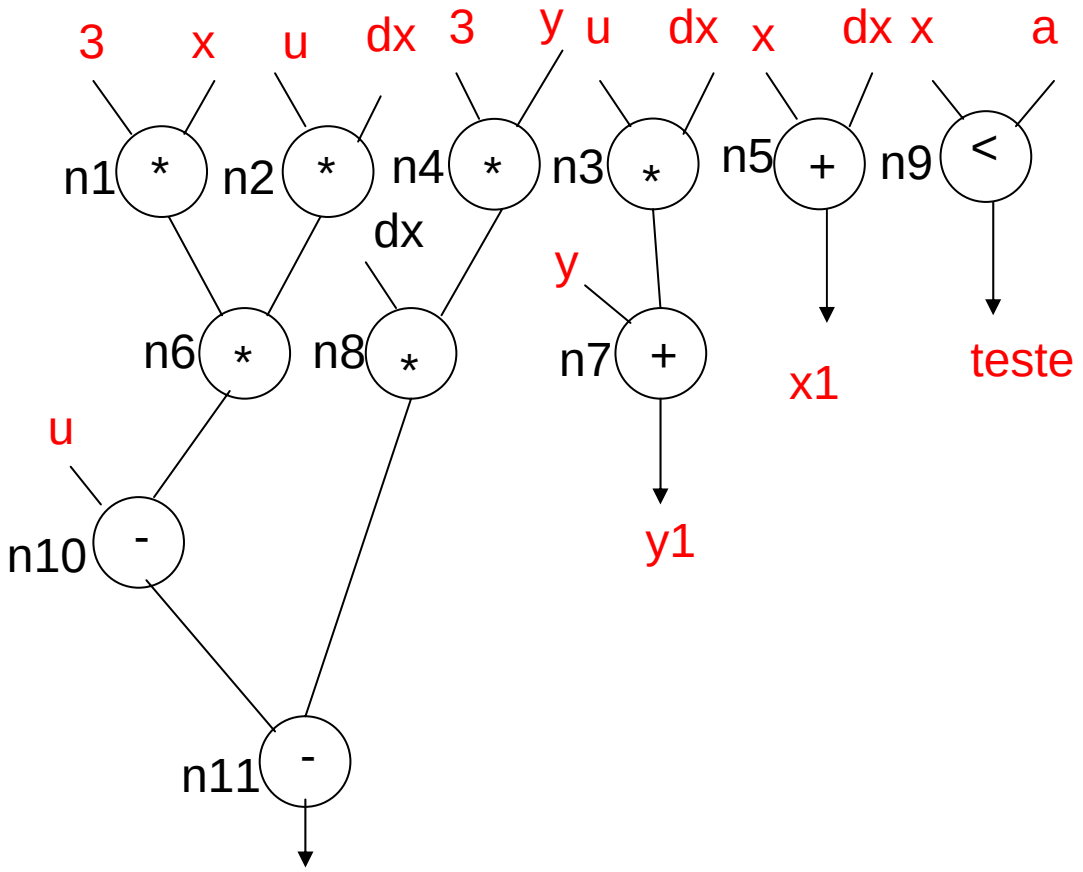
- O numero de ciclos de relógio neste caso depende da dependencia natural da aplicação.

Opção 2:
Execução por estados

- 4 multiplicadores
- 1 somadores
- 1 subtrador
- 1 comparador

Atraso caminho critico:
1 multiplicador
Tempo de execução: 4 ciclos

Ex: 4 x 100ns = 400ns



Paralelismo dos Operadores

Sequencial (Recurso limitado)

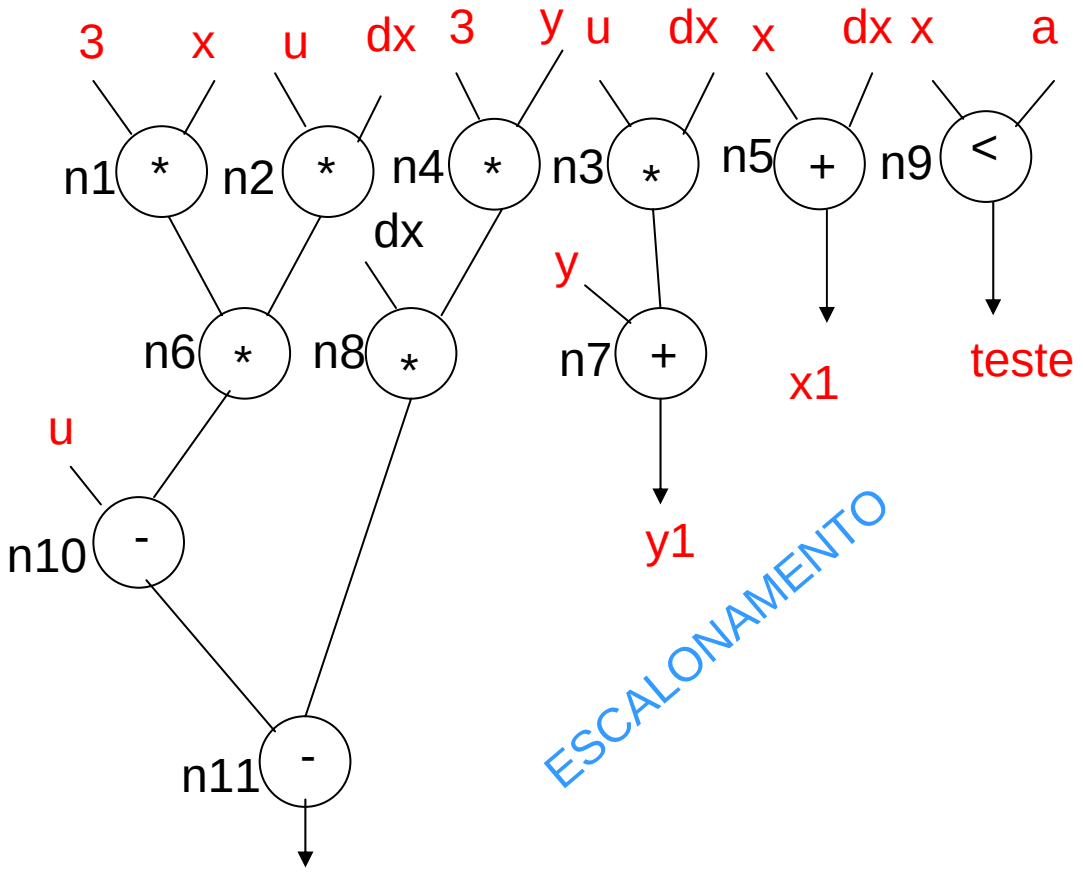
- Definir a quantidade de hardware necessária para a execução da aplicação.

Opção 3:
Execução por estados

- 1 multiplicadores
- 1 somadores
- 1 subtrador
- 1 comparador

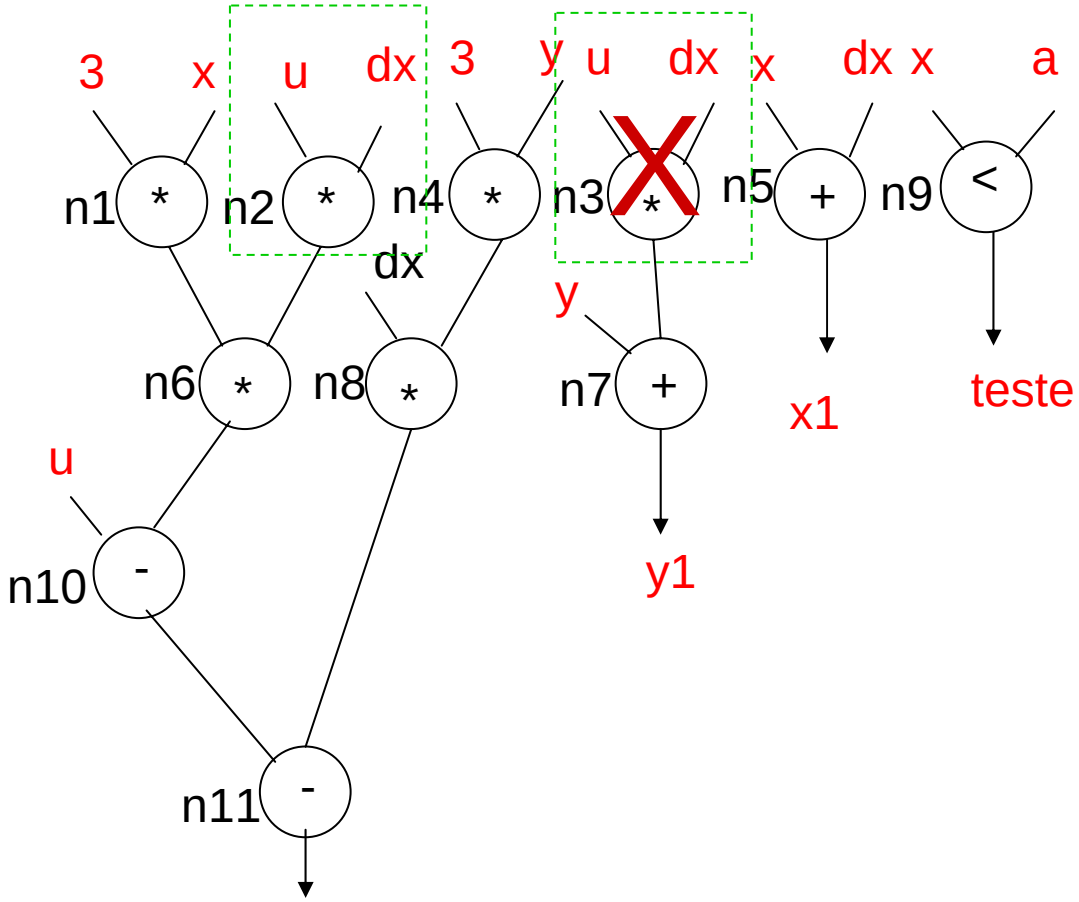
Atraso caminho critico:
1 multiplicador
Tempo de execução: X ciclos

Ex: ? x 100ns = ? ns



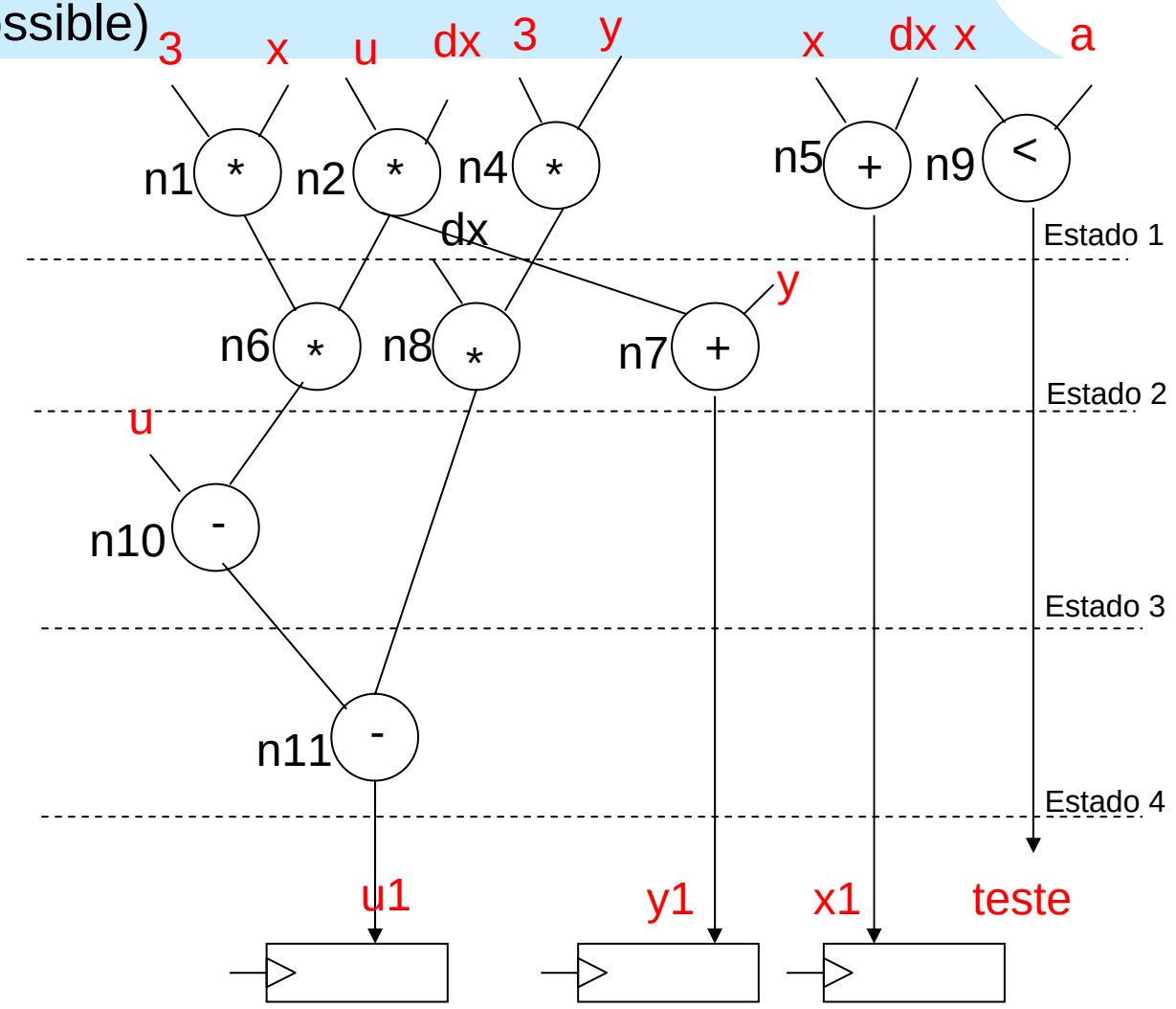
Otimizações

1 – Otimizar temos em comum



Algoritmos de Otimização

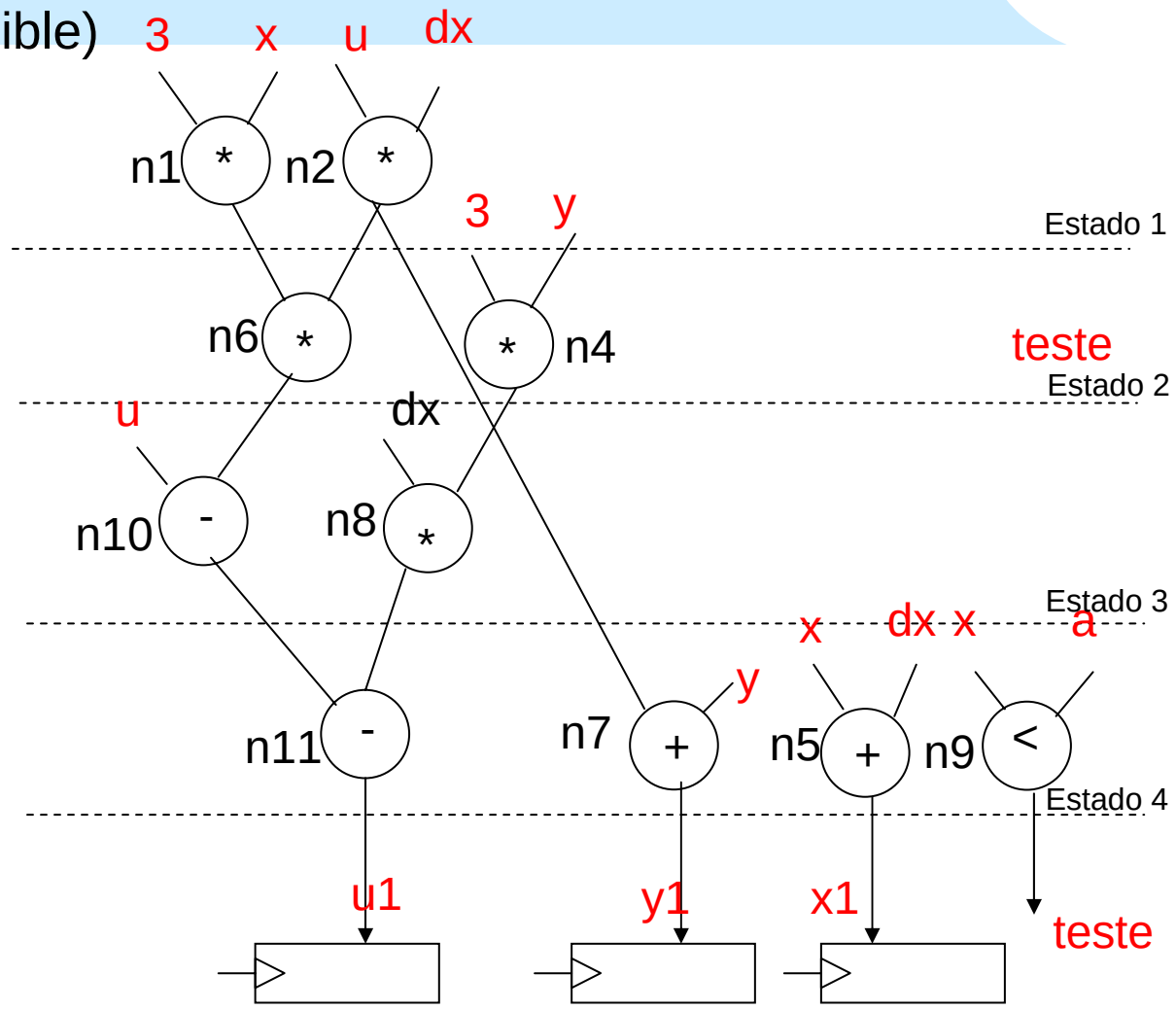
- ASAP (as soon as possible)



Estado1: 3 multiplicadores, 1 somador e 1 comparador
Estado 2: 2 multiplicadores, 1 somador
Estado 3: 1 subtrator
Estado 4: 1 subtrator

Algoritmos de Otimização

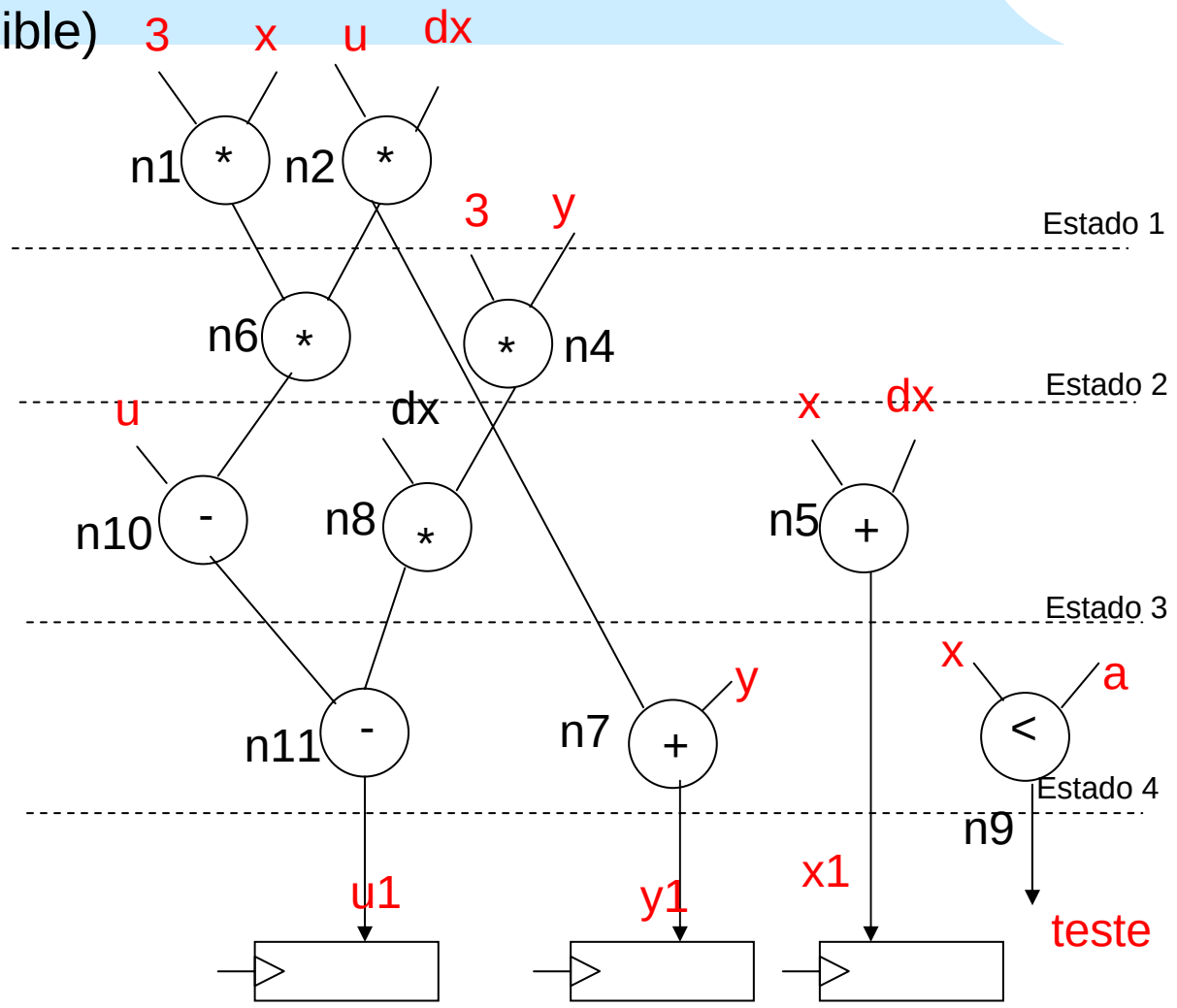
- ALAP (as late as possible)



Estado1: 2 multiplicadores,
Estado 2: 2 multiplicadores,
Estado 3: 1 multiplicador e 1 subtrator
Estado 4: 1 subtrator, 2 somadores, 1 comparador

Algoritmos de Otimização

- ALAP (as late as possible) **melhorado**



Estado1: 2 multiplicadores,
Estado 2: 2 multiplicadores,
Estado 3: 1 multiplicador e 1 subtrator e 1 somador
Estado 4: 1 subtrator, 1 somador, 1 comparador

Paralelismo dos Operadores

Puramente Combinacional

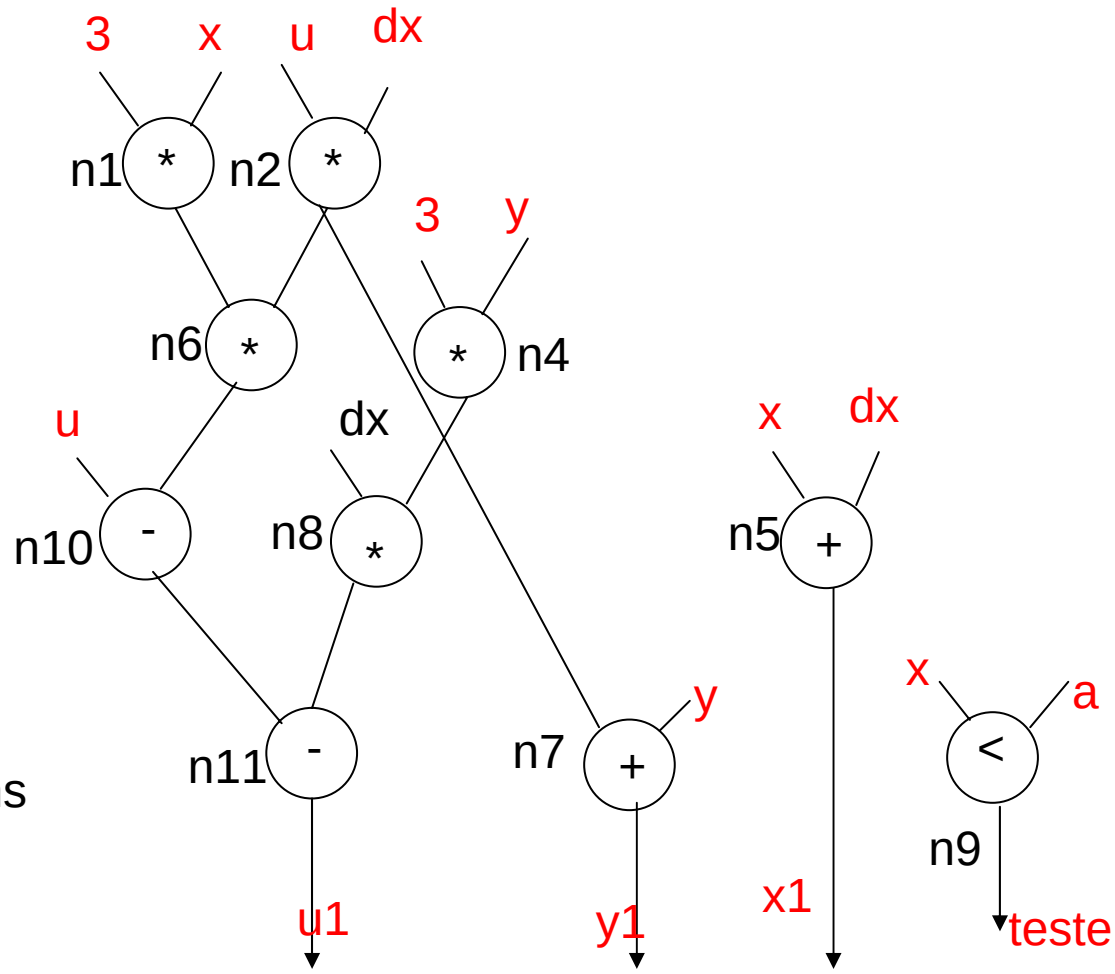
- Definir a quantidade de hardware necessária para a execução da aplicação.

Opção 1:
Tudo combinacional

- **5 multiplicadores**
- 2 somadores
- 3 subtradores
- 1 comparador

Atraso caminho critico:
2 multiplicadores + 2 subtratores

Ex: 100ns + 100ns + 30ns + 30 ns
= 260 ns



Paralelismo dos Operadores

Sequencial (sem limite de recursos)

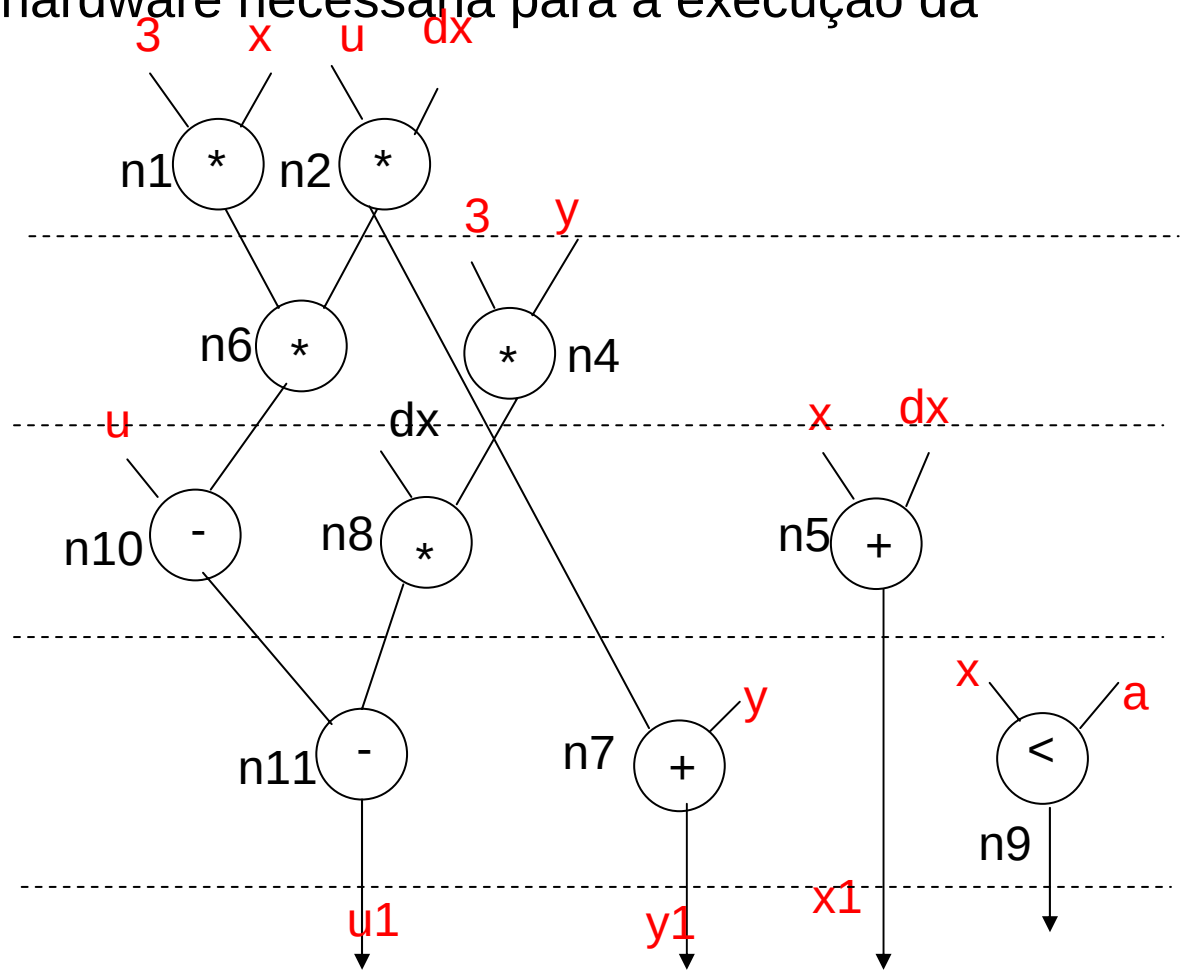
- Definir a quantidade de hardware necessária para a execução da aplicação.

Opção 2:
Execução por estados

- **2 multiplicadores**
- 1 somadores
- 1 subtrador
- 1 comparador

Atraso caminho critico:
1 multiplicador
Tempo de execução: 4 ciclos

Ex: 4 x 100ns = 400ns

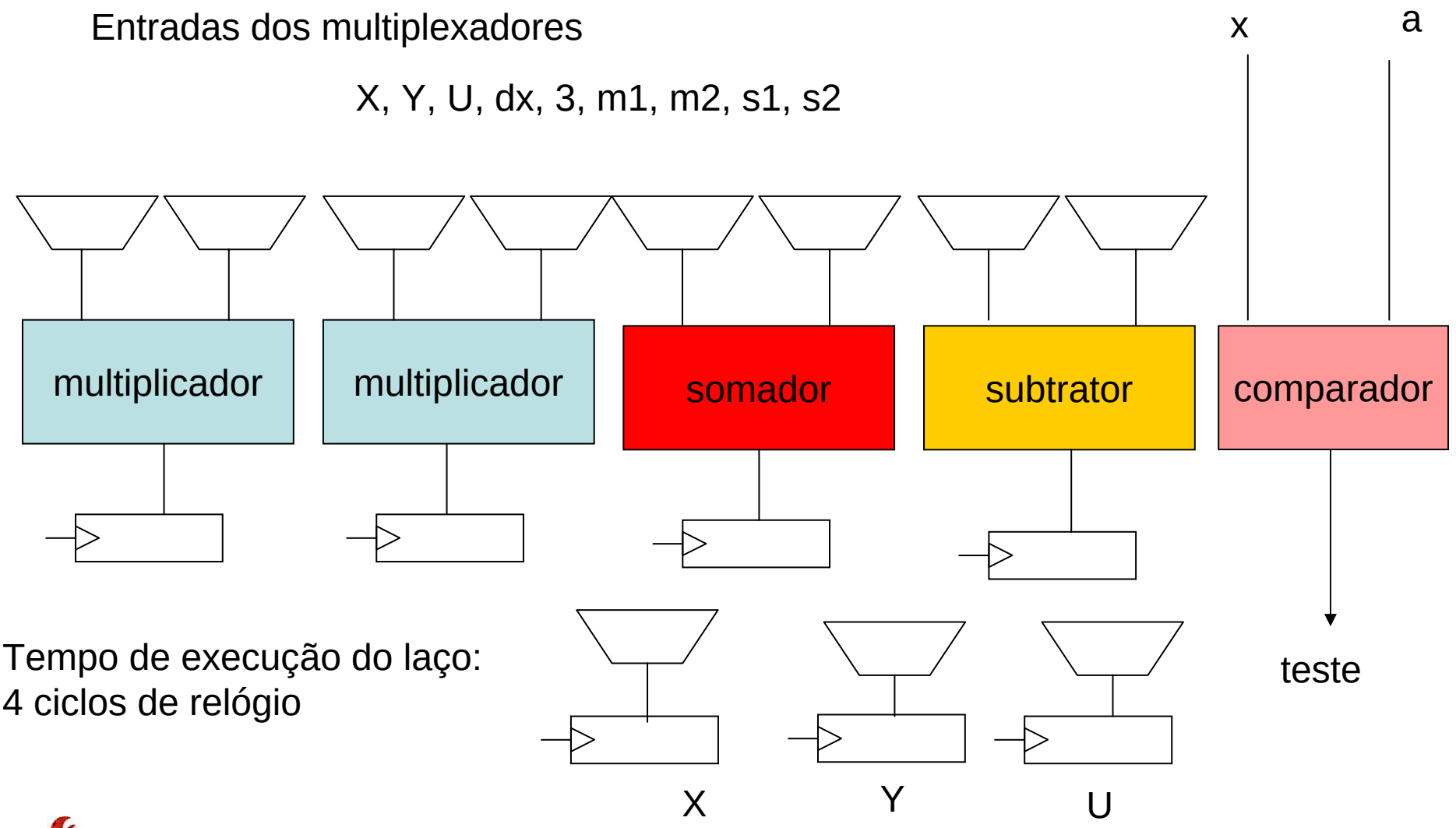


Hardware necessário

Sequencial (sem limite de recursos)

Entradas dos multiplexadores

X, Y, U, dx, 3, m1, m2, s1, s2



Tempo de execução do laço:
4 ciclos de relógio

Uso de Pipeline

Pipeline

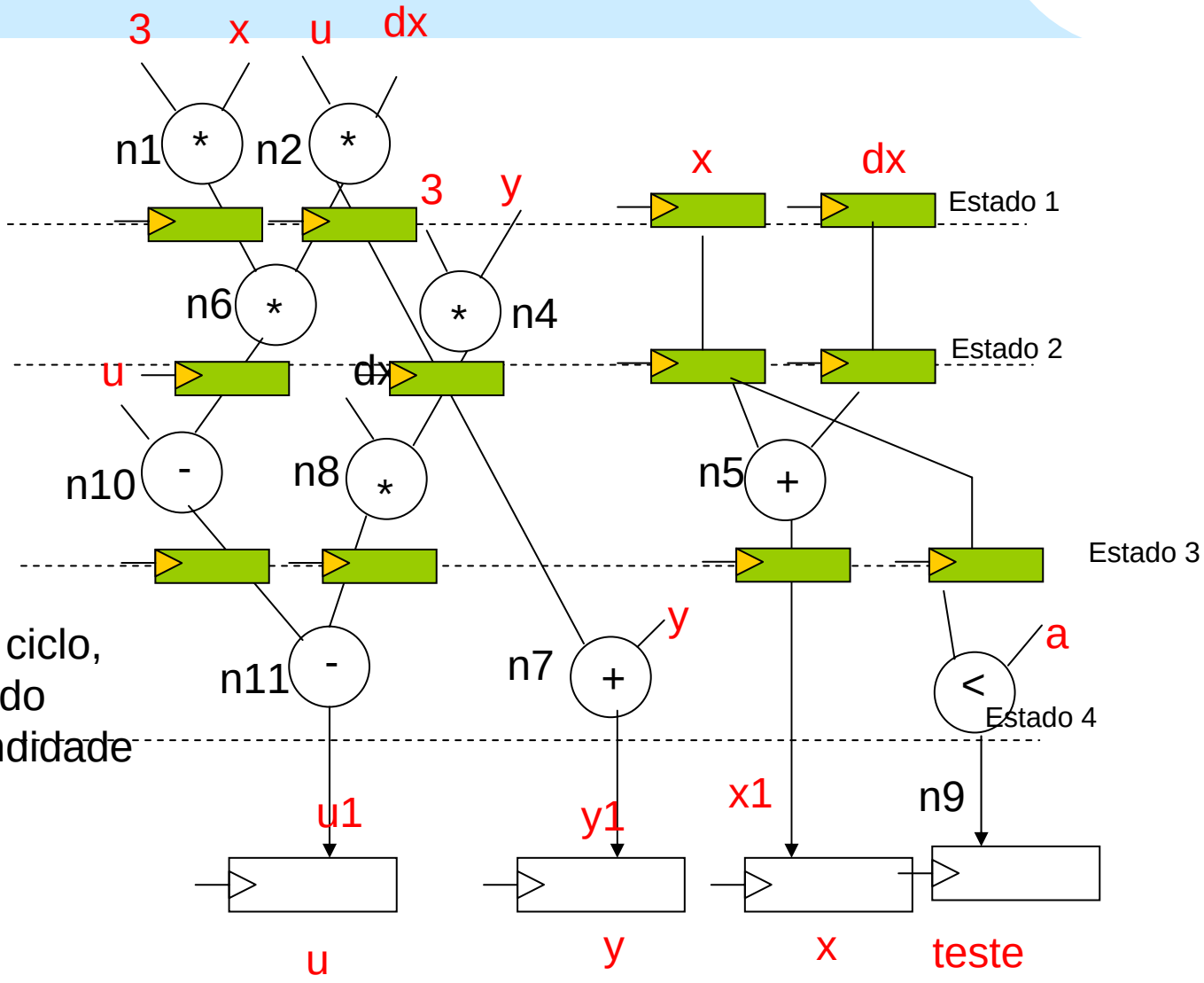
Opção 3:
Pipeline

- **5 multiplicadores**
- 2 somadores
- 3 subtradores
- 1 comparador

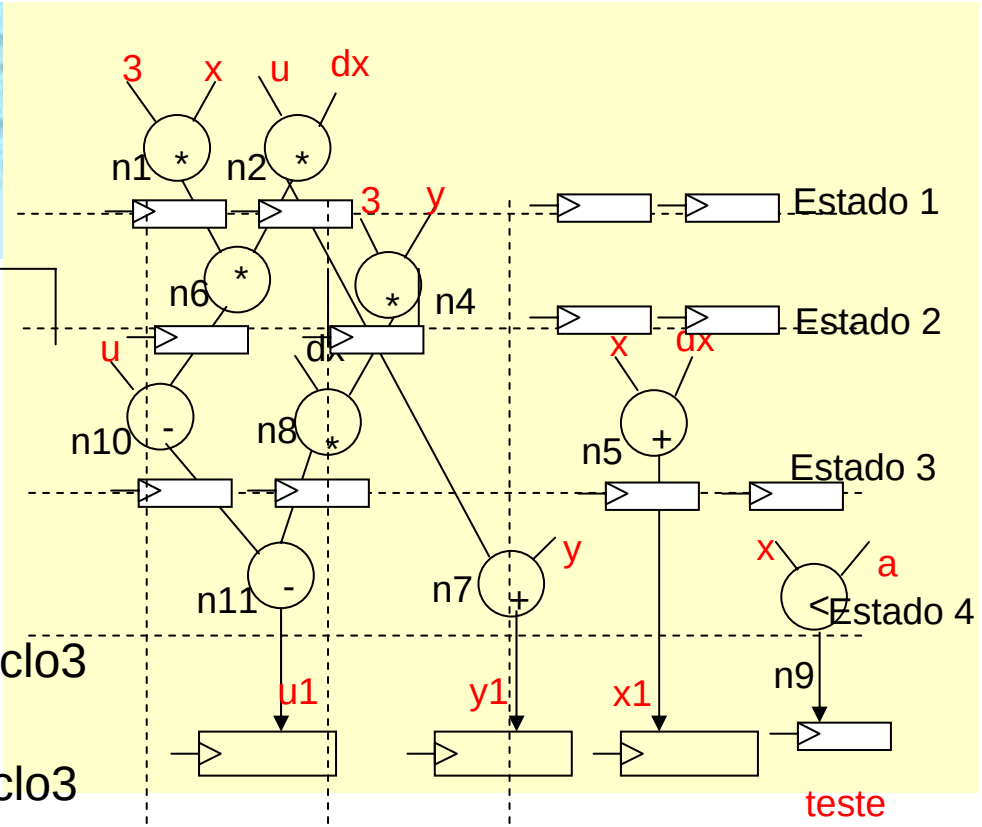
Atraso caminho critico:
1 multiplicador

Tempo de execução: 1 ciclo,
Após o preenchimento do pipeline que tem profundidade 4.

Ex: 1x100ns = 100 ns



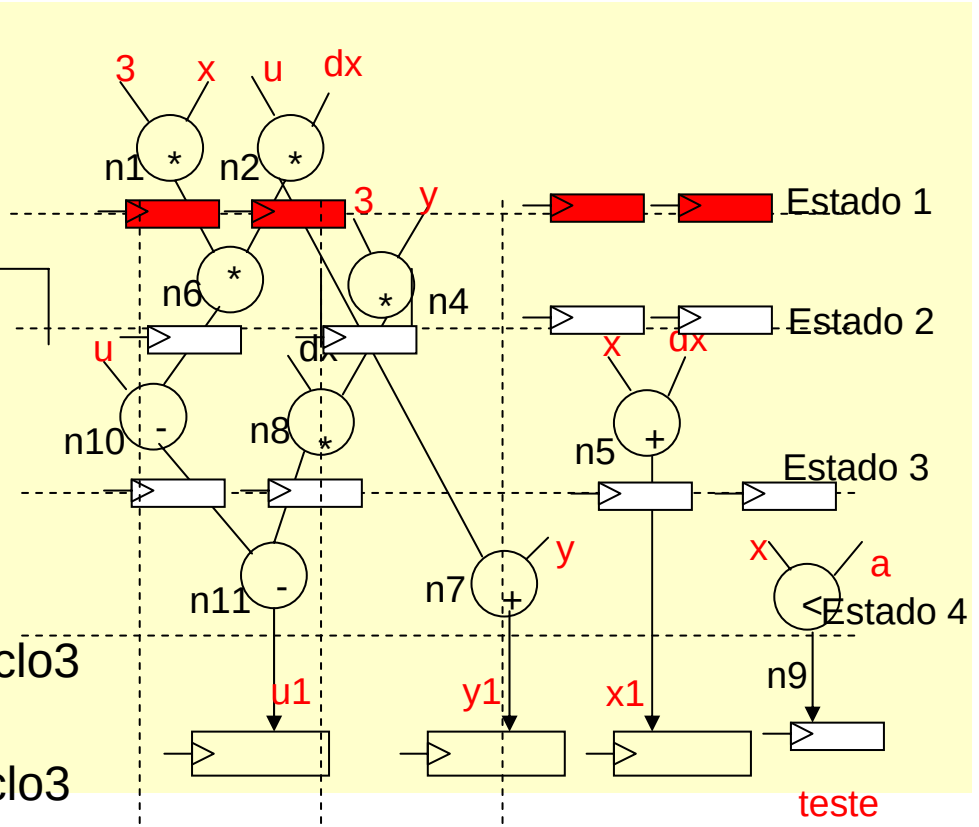
Pipeline



clk

n1	xxxxxxxxciclo1	ciclo2	ciclo3					
n2	xxxxxxxxciclo1	ciclo2	ciclo3					
n6	xxxxxxxxxxxxxxxciclo1	ciclo2	ciclo3					
n4	xxxxxxxxxxxxxxxciclo1	ciclo2	ciclo3					
n11	xxxxxxxxxxxxxxxxxxxxxxxxciclo1	ciclo2	ciclo3	ciclo4	ciclo5 …			
n8	xxxxxxxxxxxxxxxxxxxxxxxxciclo1	ciclo2	ciclo3	ciclo4	ciclo5 ….			
U1	valorc0	ciclo1	ciclo2	ciclo3	ciclo4 …..			
Y1	valorc0	ciclo1	ciclo2	ciclo3	ciclo4 …..			
X1	valorc0	ciclo1	ciclo2	ciclo3	ciclo4 …..			

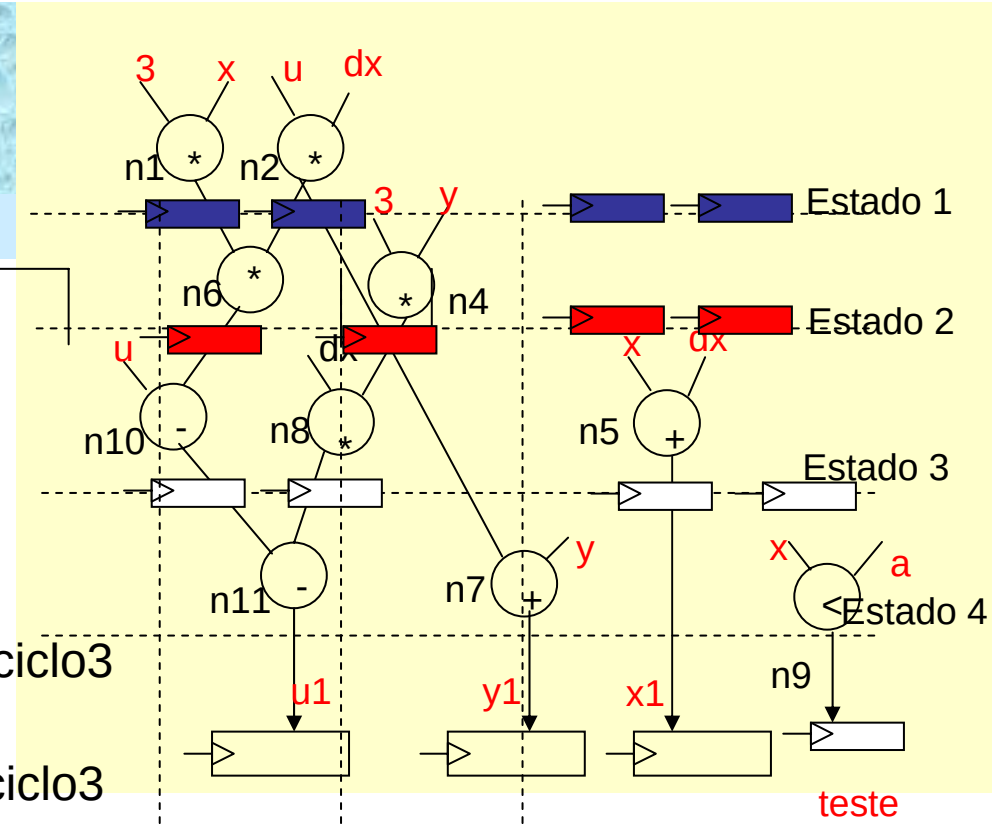
Pipeline



clk

n1	xxxxxxxx**ciclo1**	ciclo2	ciclo3					
n2	xxxxxxxx**ciclo1**	ciclo2	ciclo3					
n6	xxxxxxxxxxxxxxxx	ciclo1	ciclo2	ciclo3				
n4	xxxxxxxxxxxxxxxx	ciclo1	ciclo2	ciclo3				
n11	xxxxxxxxxxxxxxxxxxxxxxxx	ciclo1	ciclo2	ciclo3	ciclo4	ciclo5 …		
n8	xxxxxxxxxxxxxxxxxxxxxxxx	ciclo1	ciclo2	ciclo3	ciclo4	ciclo5 ….		
U1	valorc0		ciclo1	ciclo2	ciclo3	ciclo4 …..		
Y1	valorc0		ciclo1	ciclo2	ciclo3	ciclo4 …..		
X1	valorc0		ciclo1	ciclo2	ciclo3	ciclo4 …..		

Pipeline

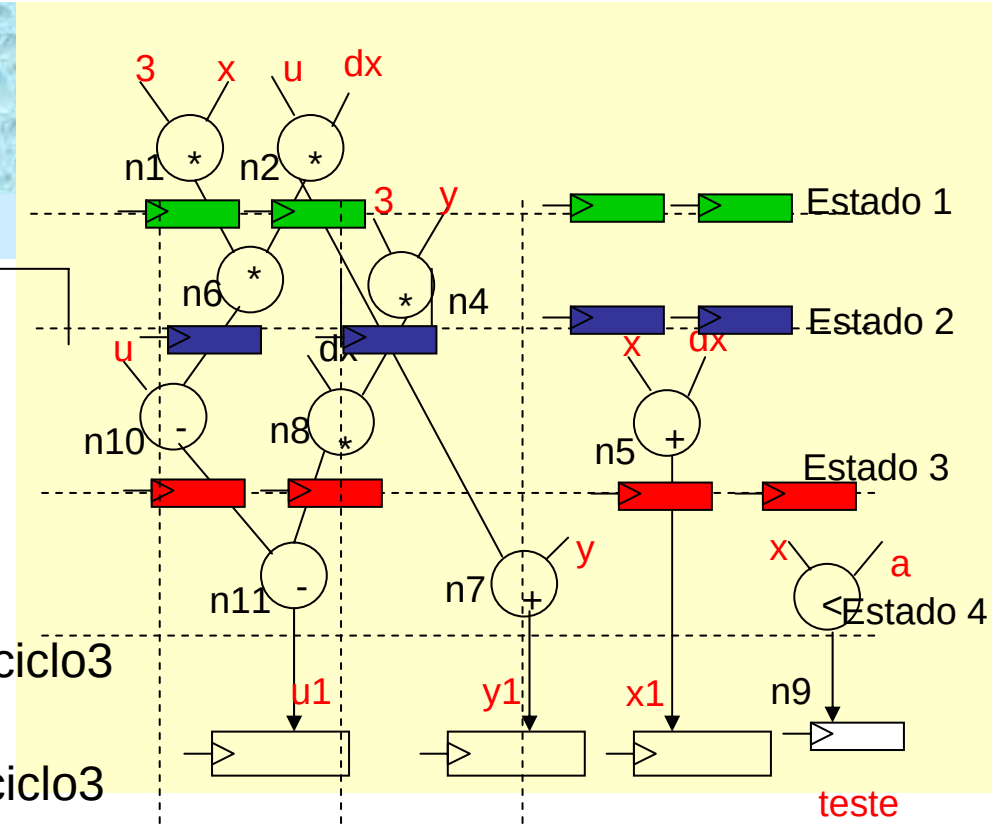


clk

n1	xxxxxxxxciclo1	**ciclo2**	ciclo3			
n2	xxxxxxxxciclo1	**ciclo2**	ciclo3			
n6	xxxxxxxxxxxxxxxx	**ciclo1**	ciclo2	ciclo3		
n4	xxxxxxxxxxxxxxxx	**ciclo1**	ciclo2	ciclo3		
n11	xxxxxxxxxxxxxxxxxxxxxxxx	ciclo1	ciclo2	ciclo3	ciclo4	ciclo5 …
n8	xxxxxxxxxxxxxxxxxxxxxxxx	ciclo1	ciclo2	ciclo3	ciclo4	ciclo5 ….

U1	valorc0	ciclo1	ciclo2	ciclo3	ciclo4 …..
Y1	valorc0	ciclo1	ciclo2	ciclo3	ciclo4 …..
X1	valorc0	ciclo1	ciclo2	ciclo3	ciclo4 …..

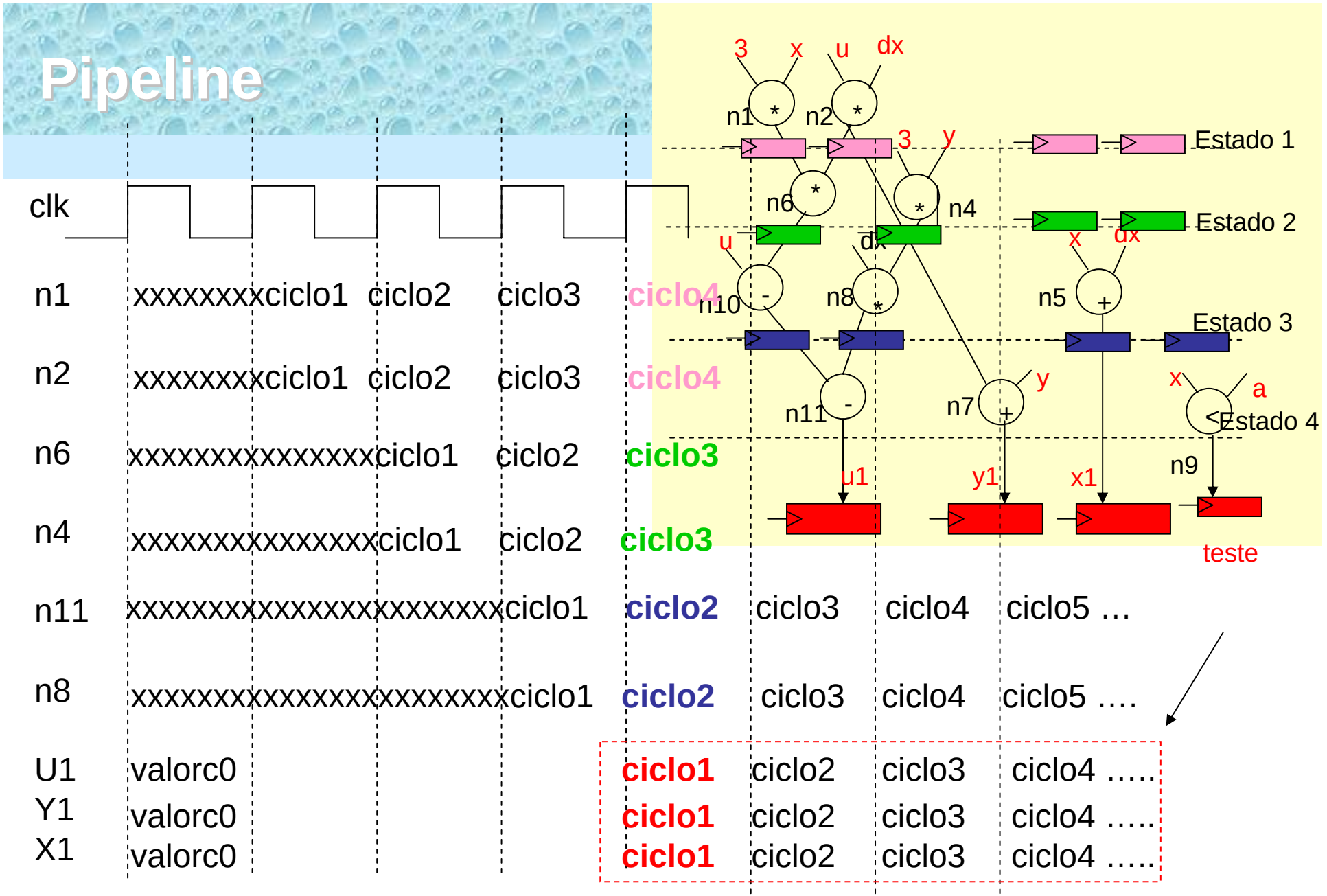
Pipeline



clk

n1	xxxxxxxxciclo1	ciclo2	**ciclo3**			
n2	xxxxxxxxciclo1	ciclo2	**ciclo3**			
n6	xxxxxxxxxxxxxxxciclo1	**ciclo2**	ciclo3			
n4	xxxxxxxxxxxxxxxciclo1	**ciclo2**	ciclo3			
n11	xxxxxxxxxxxxxxxxxxxxxxxx	**ciclo1**	ciclo2	ciclo3	ciclo4	ciclo5 …
n8	xxxxxxxxxxxxxxxxxxxxxxxx	**ciclo1**	ciclo2	ciclo3	ciclo4	ciclo5 ….
U1	valorc0		ciclo1	ciclo2	ciclo3	ciclo4 …..
Y1	valorc0		ciclo1	ciclo2	ciclo3	ciclo4 …..
X1	valorc0		ciclo1	ciclo2	ciclo3	ciclo4 …..

Pipeline

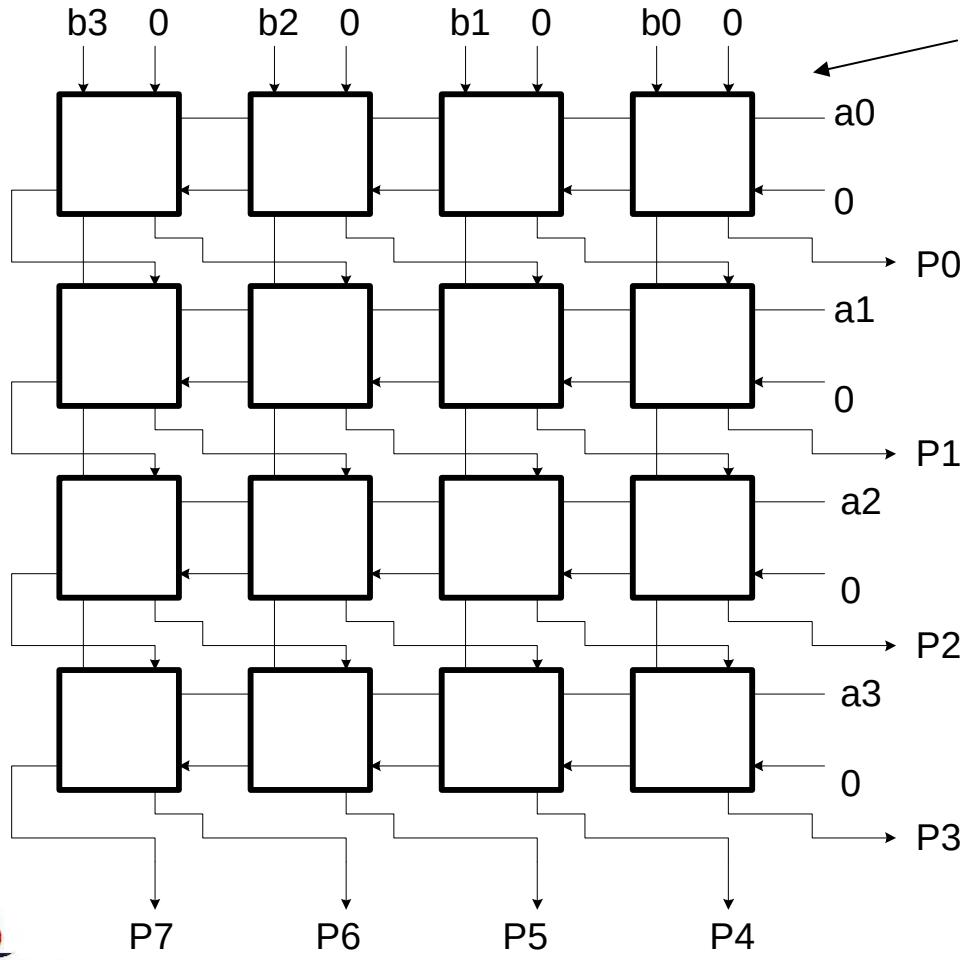

clk
n1 xxxxxxxxciclo1 ciclo2 ciclo3 ciclo4
n2 xxxxxxxxciclo1 ciclo2 ciclo3 ciclo4
n6 xxxxxxxxxxxxxxxciclo1 ciclo2 ciclo3
n4 xxxxxxxxxxxxxxxciclo1 ciclo2 ciclo3
n11 xxxxxxxxxxxxxxxxxxxxxxxxciclo1 ciclo2 ciclo3 ciclo4 ciclo5 …
n8 xxxxxxxxxxxxxxxxxxxxxxxxciclo1 ciclo2 ciclo3 ciclo4 ciclo5 ….
U1 valorc0 ciclo1 ciclo2 ciclo3 ciclo4 …..
Y1 valorc0 ciclo1 ciclo2 ciclo3 ciclo4 …..
X1 valorc0 ciclo1 ciclo2 ciclo3 ciclo4 …..
3 x u dx
n1 n2 n6 n4 n10 n8 n5 n11 n7 n9
Estado 1
Estado 2
Estado 3
Estado 4
u1 y1 x1
teste

Comparação

	Combinacional	4 estados (PC-PO)	Pipeline
Area	**5 multiplicadores** 2 somadores 3 subtradores 1 comparador	**2 multiplicadores** 1 somadores 1 subtrador 1 comparador	**5 multiplicadores** 2 somadores 3 subtradores 1 comparador
Desempenho	Atraso de 2 multiplicadores + 2 subtratores	4 ciclos de Atraso 1 multiplicador	1 ciclo de Atraso de 1 multiplicador Latencia = 4.

Exemplo

- Descrever em VHDL um multiplicador 4x4 com pipeline. E compare o desempenho sem e com pipeline.



Each row: n-bit adder with AND gates