Prototipação de Sistemas Digitais

Síntese RTL

Cristiano Araújo

Síntese RTL

- **Síntese RTL**
  - **A Síntese de Alto-Nível tipicamente fornece uma descrição inicial no nível RT do Caminho de dados(Data Path) e do Controlador(Controller).**
  - **A síntese RTL pode ser caraterizada como um refinamento da síntese de Alto-Nível. Neste nível de projeto, onde a sincronização é explicitamente feita por um ou mais sinais de relógio(clock), as características físicas dos bloco lógicos são conhecidas em maior detalhe que no nível algoritmo, permitindo assim etapas de otimização baseados em modelos mais realistas que àqueles da Síntese de Alto-Nível.**
  - **Descrições a nível RT representam a entrada para síntese a nível lógico do sistema, para qual o sistema é especificado por blocos de lógica combinacional e elementos de armazenamento.**

Síntese RTL

- Modelos de entrada para síntese RTL vão desde FSMDs completas à arquiteturas completamente especificadas.

Synthesis task	Synchronization points	Input design model
Scheduling	I/O events	CFG
Resource allocation	Clock	FSMD
Resource binding	Clock	FSMD with resources
Logic synthesis	Clock	FSM + DP
Layout synthesis	Wire value change	Gate netlist

Behavioral synthesis

RTL synthesis

Síntese Algorítmica

Descrição inicial:
- Data Parth (Caminho de dados)
- Controller (Controlador)

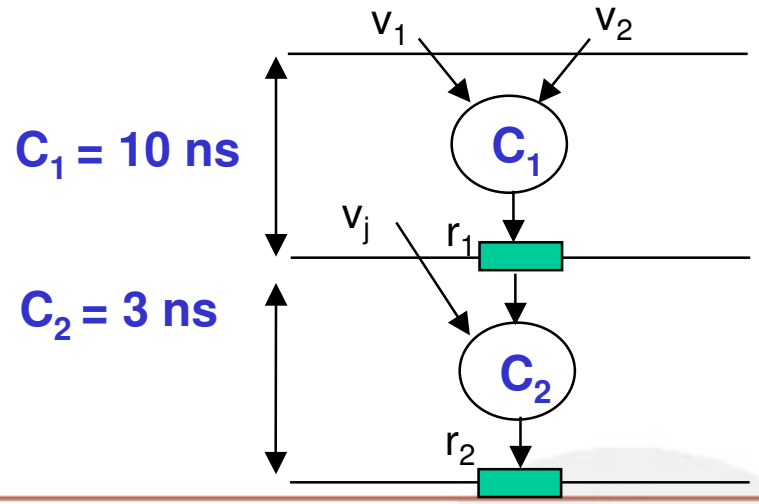
Síntese RTL

Síntese do Data Path
Síntese do controlador (FSM)

Descrição resultante:
- Blocos de lógica combinacional
- Elementos de armazenagem

Síntese Lógica

- Técnicas de Otimização
  - **Suponha um Scheduling tendo como resultado a seguinte descrição a nível RT:**
    - control-step 1: r_1 <- $c_1(v_i, ...)$
    - control-step 2: r_2 <- $c_2(r_1, v_j, ...)$
  - **Onde** r_1 e r_2 são registradores, c_1 e c_2 blocos computacionais e v_i e v_j variáveis. Baseado na resposta em tempo do sistema, teríamos que a freqüência máxima de operação do circuito seria dado por:
    - **f_{max} = 1/max{delay(c_1), delay(c_2)}**



C_1 = 10 ns

C_2 = 3 ns

delay(c_1) > delay(c_2)

$fmax_{operação}$ = 1/10 = 100 MHz

- **Supondo que o bloco** c_1 determina a freqüência do circuito, isso é delay(c_1) > delay(c_2), existem várias possibilidades de aumentarmos o relógio do sistema (retiming) sem mudar a estrutura global do mesmo.
  - **1ª** Substituir blocos lentos por blocos combinacionais mais rápidos (ex: ripple carry adder por um carry-look ahead adder)
  - **2ª** Reestruturar o circuito e mudar as posições de registradores.
- **As possíveis modificações acima implicam em negociação entre área e tempo. Por exemplo, maior área, maior velocidade, maior custo, etc.**

- Reestruturar o circuito
  - Reestruturar o circuito e
  - Mudar posição de registradores (retiming)

- **Suponha c_1 pode ser reestuturado, particionado-o em dois novos circuitos**

$$c_1^{novo} = g(f_1(v_i, \ldots), f_2(v_i, \ldots))$$

- onde f_1 e f_2 têm menor complexidade que c_1;
- assim seus atrasos seriam assumidos menor que c_1 e a nova descrição RT pode ser resintetizada (re-síntese), resultando em:

control-step 1: $r_1 \leftarrow f_1(v_i, \ldots); r_2 \leftarrow f_2(v_i, \ldots)$

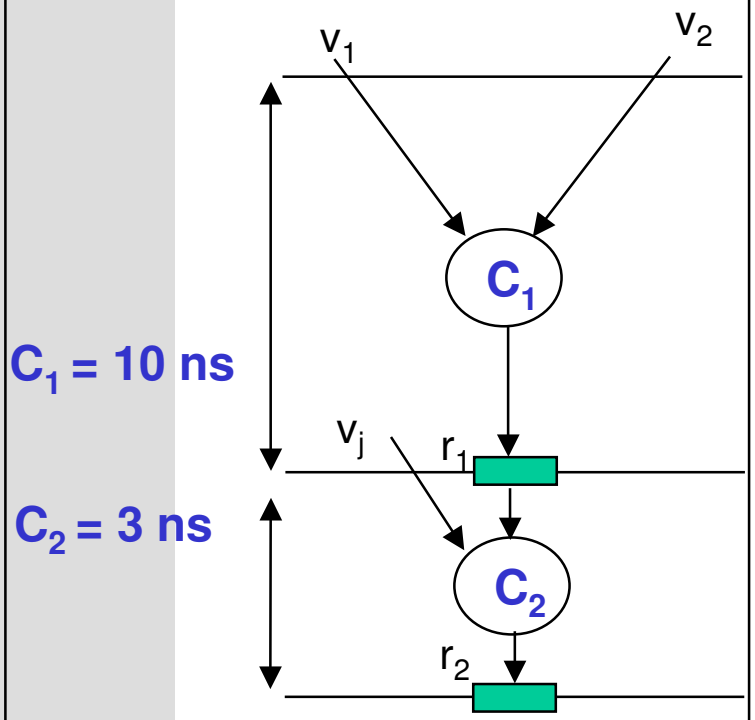
control-step 2: $r_3 \leftarrow c_2(g(r_1, r_2), v_j, \ldots)$

-A solução é mais rápida que a original assumindo que

- **$delay(g)+delay(c_2) < delay(c_1)$**

Re-estruturando o circuito

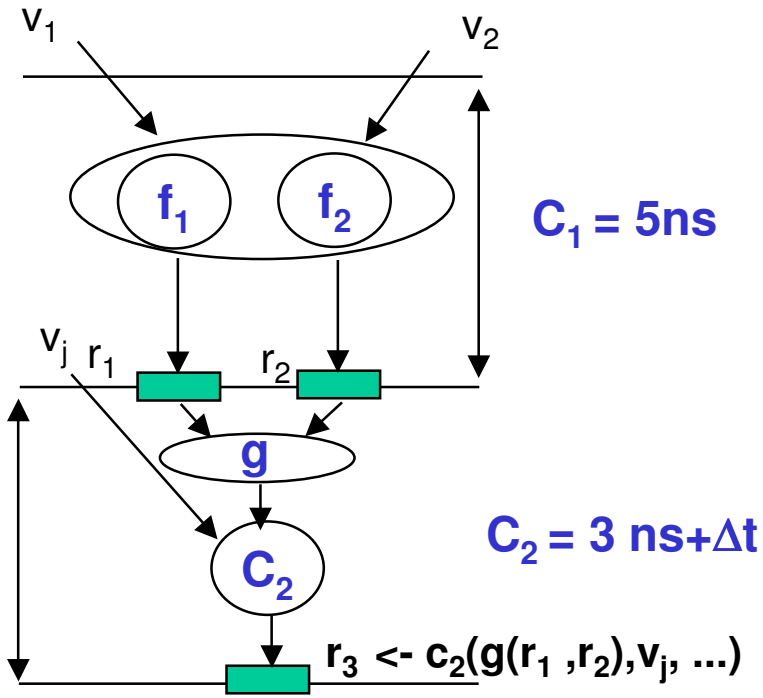
- **Circuito Original (exemplo)**



$delay(c_1) > delay(c_2)$

$F_{operação}$ = 1/10 = 100 MHz

- **Reestruturando o circuito (re-síntese)**



$c_1^{novo} = g(f_1(v_i, ...), f_2(v_i, ...)) = 4$ ns

$F_{operação}$ = 1/4 = 250 MHz

$delay(g)+delay(c_2) < delay(c_1)$

- Retiming
  - É uma transformação de circuitos seqüenciais no nível RT, onde registradores podem ser deslocados adequadamente entre blocos de lógica combinacional, os quais podem ser substituídos, visando minimizar o relógio do circuito ou número de registradores. Dependendo ao algoritmo de otimização o desempenho pode alcançar a 50% (as vezes).

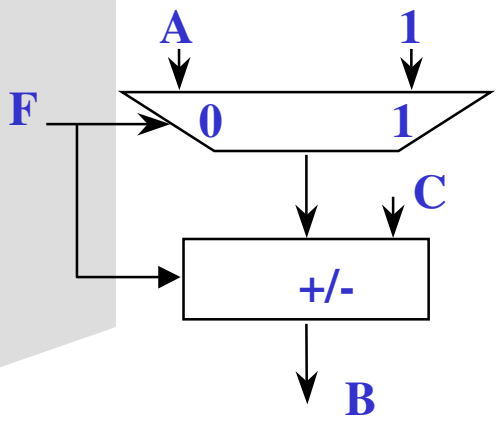
- Características gerais:
  - A função do circuito (funcionalidade) não é modificada.
  - Otimizar período de relógio tendo como restrições o número de registradores.
  - Otimizar o número dos registradores com restrições de relógio

Síntese RT - Síntese do Caminho de Dados

- Otimização de Hardware
  - if F=0 then B:= A+C
    else B:= C-1;

Primeira solução

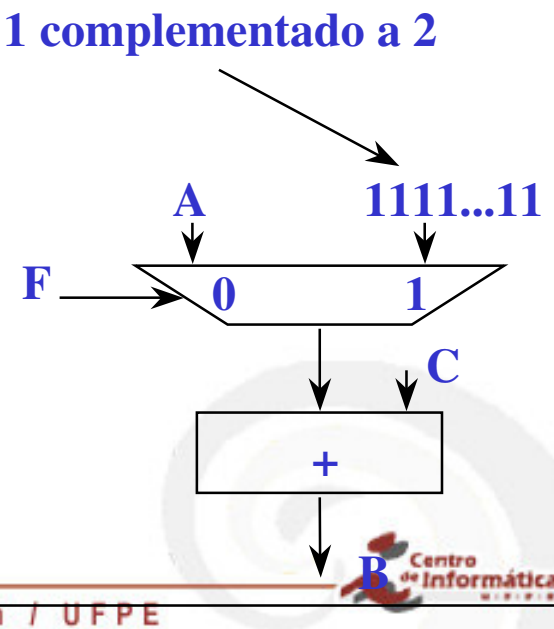
Uso de um Somador e um decrementador



Segunda solução

Como existe dependência de variáveis, as duas operações não podem se executadas simultaneamente, e uma unidade funcional seria suficiente para executá-las.

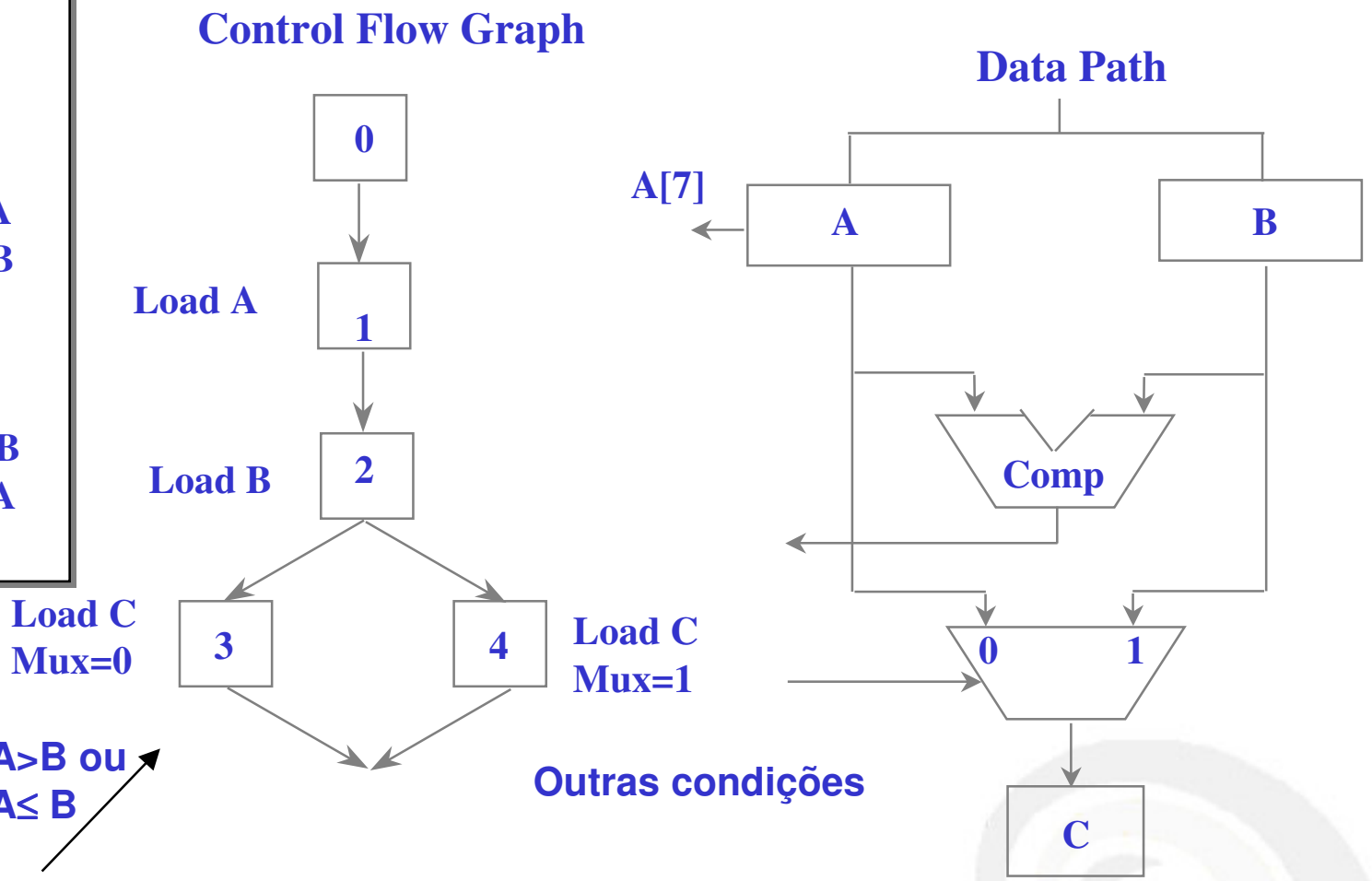
A operação de subtração do número '1' poderia ser substituída pela soma de seu inverso.



Síntese RT - Síntese do Caminho de dados

-**Redução da complexidade do controlador pelo deslocamento de parte do hardware do controle para o caminho de dados (datapath)**

```
read_port (A);
read_port (B);
if A[7]=0 then
   begin
      if A>B then C:=A
             else  C:= B
      end
else
   begin
      if A>B  then C:=B
             else C:= A
     end;
```

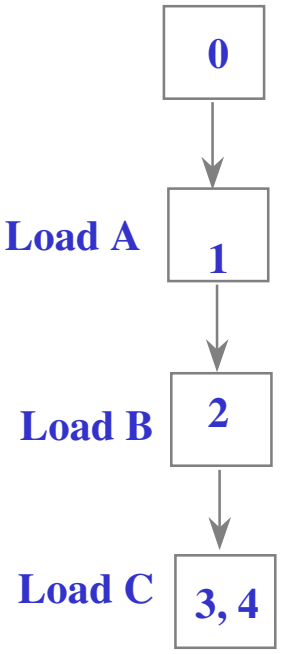


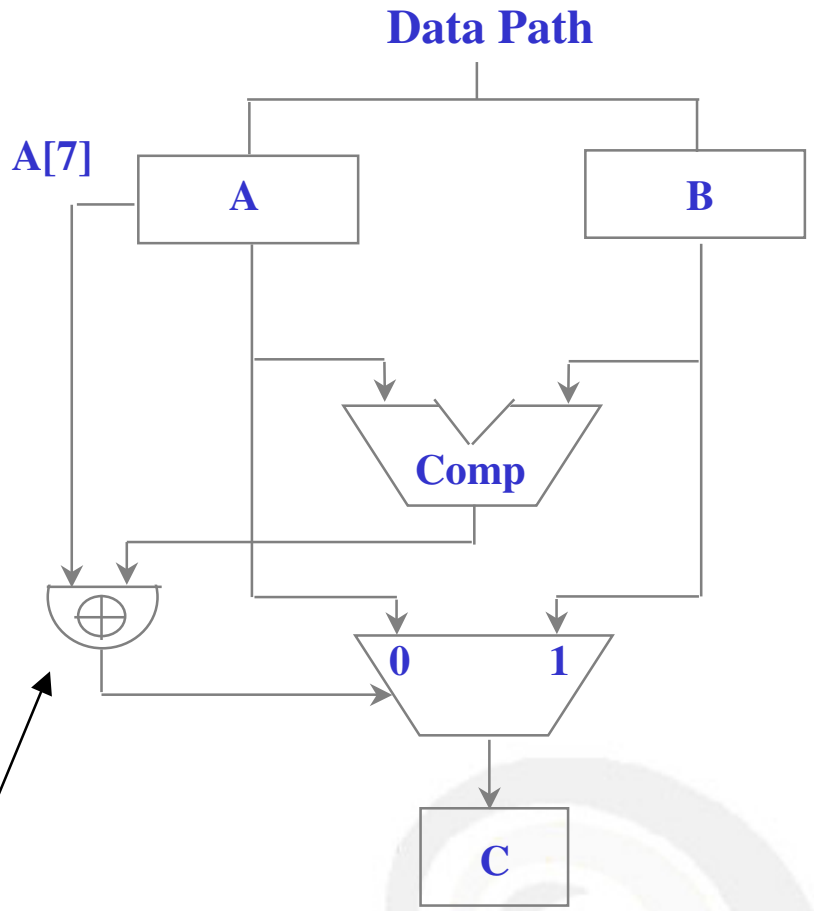
Máquina com 4 estados (Máquina de Mealy)

Síntese RT - Síntese do Caminho de dados

- Simplificação é possível através do deslocamento da síntese inicialmente voltada para controle para uma síntese voltada para dados

```
read_port (A);
read_port (B);
if A[7]=0 then
   begin
      if A>B then C:=A
             else  C:= B
      end
else
   begin
      if A>B  then C:=B
              else C:= A
     end;
```





Máquina com 3 estados (Máquina de Moore)

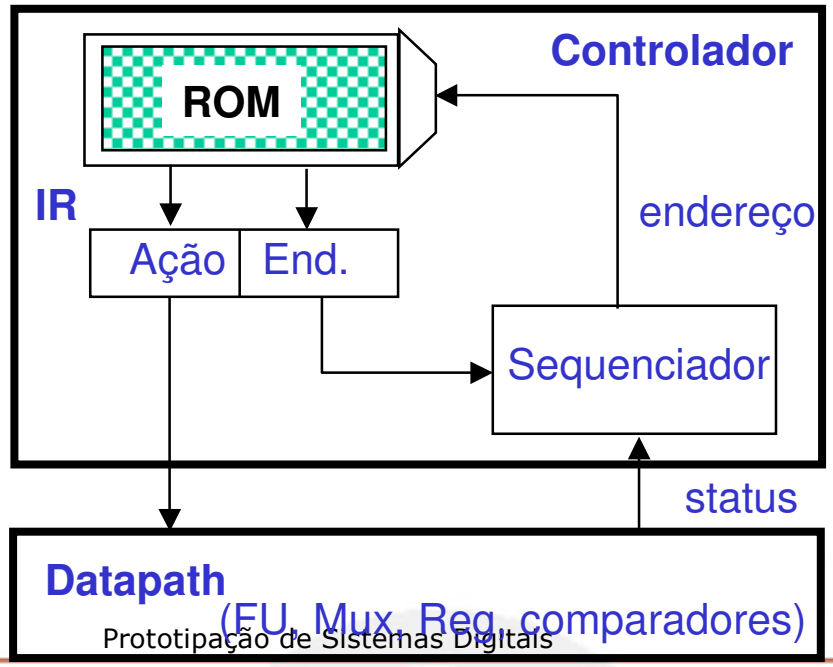
Adição de um EXOR

Síntese RT - Síntese do Controlador

- **Controlador**
  - **A tarefa do controlador é o de controlar o caminho de dados (data path), carregando registradores, selecionando multiplexadopres ou selecionando o código de operações em unidades multi-funcionais.**
  - **Implementação do Scheduling em hardware.**

RT-Síntese de controle

- Arquitetura de controladores
  - **Controladores microprogramáveis**
    - Cada comando do microprograma corresponde a um ou várias palavras da ROM e cada bit controla uma parte específica do caminho de dados (data path) em um certo ciclo de relógio definido pelo seqüência estabelecida dentro do controlador.



RT-Síntese de controle

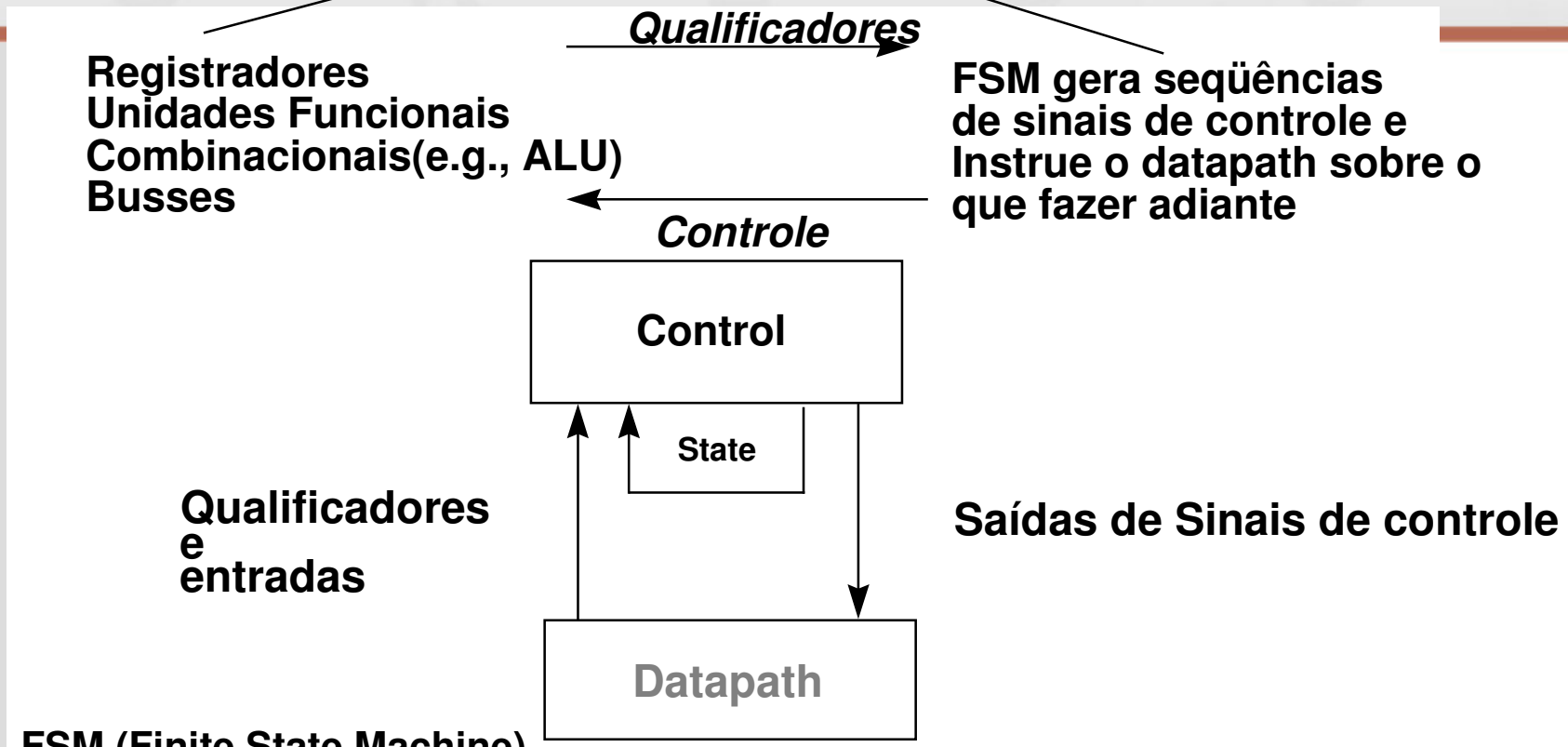
- **Controladores microprogramáveis**
  - **Modelo Geral é composto pro três blocos:**
    - Uma memória ROM contendo o código de controle ou vetores de controle ou microcódigo;
    - Um registrador de Instruções (IR) contendo os sinais de controle, o próximo endereço de instrução e o próximo flag de modo de cálculo de endereço.
    - O seqüenciador dá o próximo endereço conhecendo o conteúdo do IR e o resultado do processamento do caminho de dados (datapath).

RT-Síntese de controle

- **Máquinas de controle baseadas em FSM**
  - Um caminho para se implementar um controle é através de uma *FSM*, usando *Mealy* ou *Moore* e sintetizar está *FSM* em portas lógicas ou em uma *Programmble Logic Array (PLA)*.
  - Este caminho deve permitir descrever o comportamento do controle em uma linguagem de hardware sintetizável (VHDL).
- **Características:**
  - Redução do números de estados para reduzir tamanho do hardware
  - *One-hot encoding* com um Flip-Flop para cada estado (simplicidade)
  - PLA - Lógica dois níveis como parte combinacional + FF tipo D para armazenamento
  - FSM implementada em lógica multi-nível como parte combinacional + FF tipo D para armazenamento.

Máquina de Estados

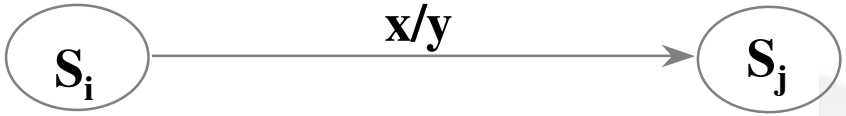
Computer Hardware = Datapath + Control



FSM (Finite State Machine)

Função de saída é dada por:

$y(t) = foutput(S(t),x(t))$



Equação de próximo estado:

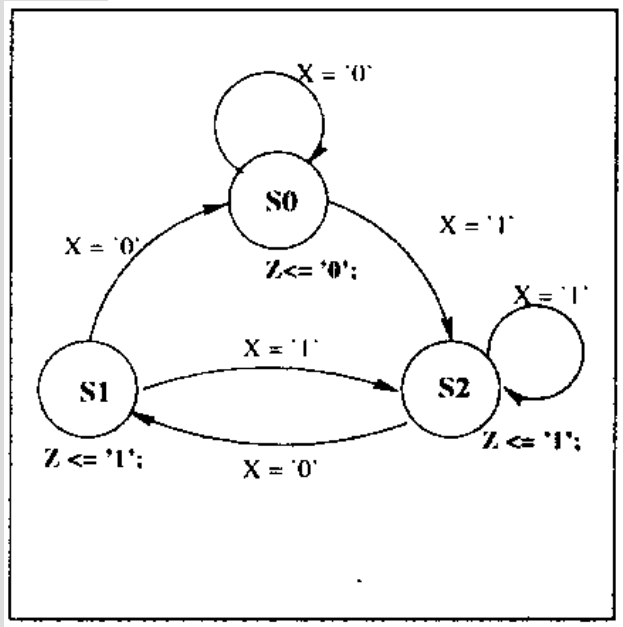
$S(t+dt) = fnextstate(S(t),x(t))$

Onde:

S(t) - estado presente; S(t+dt) - próximo estado); x(t) - entrada

RT-Síntese de controle

Descrição de Uma FSM em VHDL
Modelo de Mealy




```
ENTITY moore IS
 PORT (
  clk : IN bit;
  X : IN bit;
  Z : OUT bit
 );
END moore;

ARCHITECTURE FSM OF moore IS

 TYPE state_type IS ( S0, S2, S1);
 SIGNAL current_state, next_state : state_type;

BEGIN

 registers : PROCESS ( clk )
 BEGIN
 transitions : PROCESS ( current_state, X )
 BEGIN
  CASE current_state IS
   WHEN S0 =>
    Z <= '0';
    IF ( X = '0' ) THEN
     next_state <= S0;
    ELSIF ( X = '1' ) THEN
     next_state <= S2;
    END IF;
   WHEN S1 =>
    Z <= '1';
    IF ( X = '0' ) THEN
     next_state <= S0;
    ELSIF ( X = '1' ) THEN
     next_state <= S2;
    END IF;
   WHEN S2 =>
```