

---

## Introdução à Arquitetura de Computadores

# Os Blocos Combinatórios Básicos Circuitos Sequenciais e Máquinas de Estados

Pedro M. Lavrador

Departamento de Electrónica, Telecomunicações e Informática  
Universidade de Aveiro  
plavrador@ua.pt

## Índice

---

- Os Blocos Combinatórios Básicos.
  - *Multiplexers*
  - Descodificadores
  - Somadores
- Circuitos Sequenciais
  - Latch R-S e Flip-Flops tipo D
  - Registos e Memórias
  - Máquinas e diagramas de estado

## Índice

- Os Blocos Combinatórios Básicos.
  - *Multiplexers*
  - Descodificadores
  - Somadores
- Circuitos Sequenciais
  - Latch R-S e Flip-Flops tipo D
  - Registos e Memórias
  - Máquinas e diagramas de estado

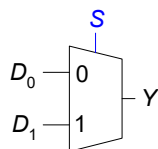
PML – IAC - 2023

3

Blocos Combinatórios Básicos

## Multiplexer

- Um Multiplexer (multiplexador) ou Mux é um circuito com N entradas de dados e 1 saída, e com  $\log_2 N$  entradas de seleção que especificam qual das entradas liga à saída.
- Um Mux 2:1 tem 2 entradas de dados, uma entrada de seleção e uma saída.



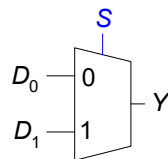
S	Y
0	$D_0$
1	$D_1$

PML – IAC - 2023

4

## Multiplexer

- Exercício:
  - Escrever a tabela de verdade do Mux 2:1, simplificá-la e representar uma implementação do circuito usando portas lógicas.



PML – IAC - 2023

5

## Multiplexer

- O multiplexer pode também ser implementado usando portas *tristate*.
- Uma porta *tristate*, pode ser descrita de modo simplista como uma porta de admissão um terceiro estado isto é ela pode ter à saída o estado lógico '0', '1' ou 'alta impedância'.

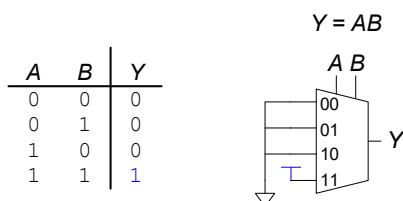


PML – IAC - 2023

6

## Funções Lógicas com Multiplexers

- O multiplexer pode ser usado para implementar funções lógicas, servindo como implementação direta de uma tabela de verdade:
  - As variáveis lógicas são usadas como entradas de seleção;
  - As entradas do multiplexer são as constantes '0' ou '1' que implementam a função pretendida.
- Por exemplo: Implementação de uma AND com mux.



PML – IAC - 2023

7

## Funções Lógicas com Multiplexers

- Exercício:
  - Usar um multiplexer 8:1 para implementar a seguinte função:
  - $Y = A.\bar{B} + \bar{B}.\bar{C} + \bar{A}.B.C$
  - É possível implementar a função anterior usando um multiplexer 4:1?

PML – IAC - 2023

8

## Índice

- Os Blocos Combinatórios Básicos.
  - *Multiplexers*
  - Descodificadores
  - Somadores
- Circuitos Sequenciais
  - Latch R-S e FlipFlops tipo D
  - Registos e Memórias
  - Máquinas e diagramas de estado

PML – IAC - 2023

9

Blocos Combinatórios Básicos

## Descodificador

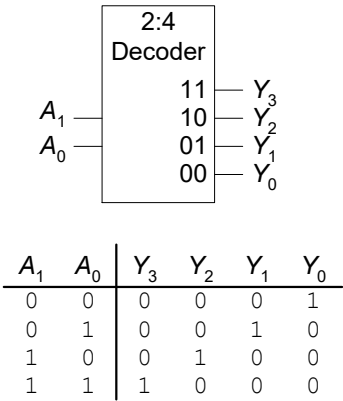
- Um decodificador é um circuito com  $N$  entradas e  $2^N$  saídas.
- Em cada instante está ativa a saída correspondente ao número que está representado na entrada.
  - Só está ativa uma saída em cada instante;
- O decodificador é um circuito necessário por exemplo para aceder a memórias:
  - permite seleccionar a célula a partir do seu endereço.

PML – IAC - 2023

10

### Descodificador

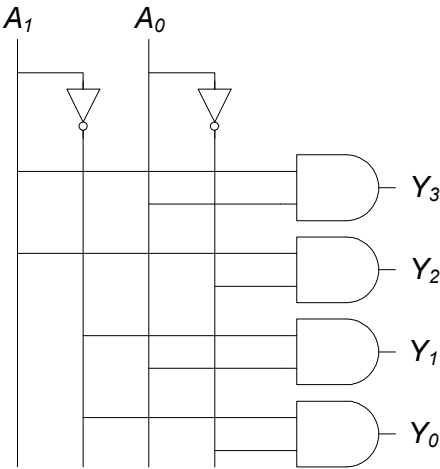
- Um descodificador de 2 entradas tem 4 saídas.



$$Y_0 = \overline{A_1} \cdot \overline{A_0} = m_0$$
$$Y_1 = \overline{A_1} \cdot A_0 = m_1$$
$$Y_2 = A_1 \cdot \overline{A_0} = m_2$$
$$Y_3 = A_1 \cdot A_0 = m_3$$

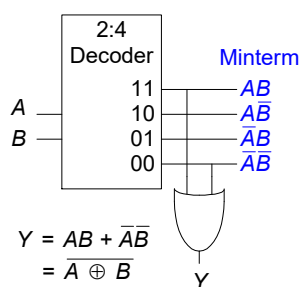
### Descodificador

- A equação da saída  $Y_x$  corresponde ao mintermo  $x$ .



## Descodificador

- O descodificador pode ser visto como um “gerador de mintermos” e portanto usado como uma base para a implementação de equações lógicas escritas na forma de soma de produtos.
- Por exemplo o operador XNOR ( $Y = AB + \bar{A}\bar{B}$ ) pode ser implementado usando um descodificador como:



PML – IAC - 2023

13

## Índice

- Os Blocos Combinatórios Básicos.
  - Multiplexers
  - Descodificadores
  - Somadores
- Circuitos Sequenciais
  - Latch R-S e FlipFlops tipo D
  - Registos
  - Máquinas e diagramas de estado

PML – IAC - 2023

14

## Somador Completo de um bit

- Quando fazemos uma adição em binário, de dois números: A mais B:

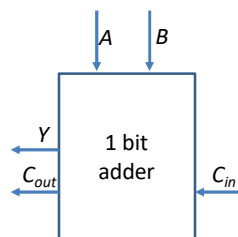
$$\begin{array}{r}
 1110 \\
 0010 \\
 +1111 \\
 \hline
 10001
 \end{array}$$

$$\begin{array}{r}
 C_i3 C_i2 C_i1 0 \\
 A_3 A_2 A_1 A_0 \\
 + B_3 B_2 B_1 B_0 \\
 \hline
 Y_3 Y_2 Y_1 Y_0 \\
 \dots C_{o1} C_{o0}
 \end{array}$$

- Na coluna  $i$  temos que considerar os bits  $A_i$ ,  $B_i$ , o  $Carry In_i$  e calcular as saídas  $Y_i$  e  $Carry Out_i$ .

## Somador Completo de um bit

- Exercício:
  - Escreva a tabela de verdade do somador completo, considerando:
    - como entradas  $A$ ,  $B$  e  $C_{in}$
    - como saídas  $Y$  e  $C_{out}$
  - Represente uma implementação possível do somador completo.





## Exercício: Somador completo de 1 bit

A	B	C <sub>in</sub>	Y	C <sub>out</sub>
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

Y

BCin

	00	01	11	10
A	0	1	0	1
1	1	0	1	0

$$Y = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

$$Y = A \oplus B \oplus C_{in}$$

C<sub>out</sub>

BCin

	00	01	11	10
A			1	
1		1	1	1

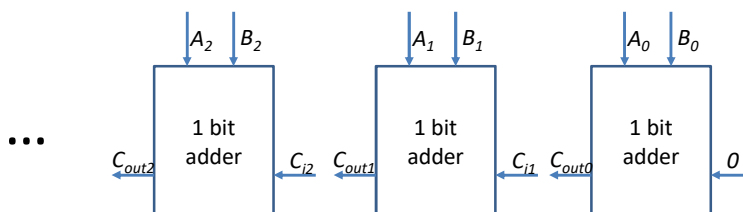
$$C_{out} = BC_{in} + AC_{in} + AB$$

PML – IAC - 2023

17

## Somador Completo

- Somador de N bits:
  - Uma possível implementação de um somador de N bits é cascatear vários somadores



- Esta solução tem um tempo de atraso muito grande porque a soma do último bit depende do resultado do primeiro.
  - Os tempos de atraso de cada somador propagam-se.
  - Existem melhores implementações.

PML – IAC - 2023

18

## Outros Circuitos Básicos

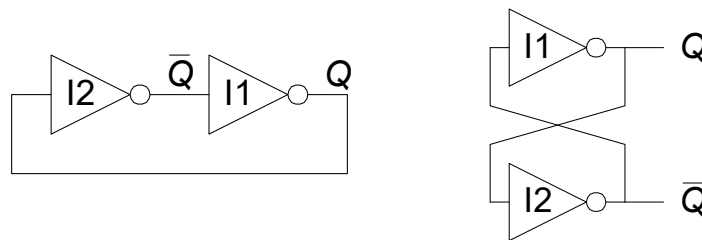
- Além dos Multiplexers, Descodificadores e Somadores são ainda blocos fundamentais:
  - Comparadores
  - Unidades aritméticas e lógicas
    - Realizam uma de várias operações possíveis.
    - A saída é escolhida com um multiplexer.
  - e Shifters

## Índice

- Os Blocos Combinatórios Básicos.
  - Multiplexers
  - Descodificadores
  - Somadores
- Circuitos Sequenciais
  - Latch R-S e FlipFlops tipo D
  - Registos e Memórias
  - Máquinas e diagramas de estado

## Circuitos Sequenciais

- Circuitos sequenciais são aqueles em que a saída depende do valor atual e do valor passado das entradas.
- Mas, como pode um circuito manter informação sobre o valor passado das entradas?
- O circuito Bi-estável:

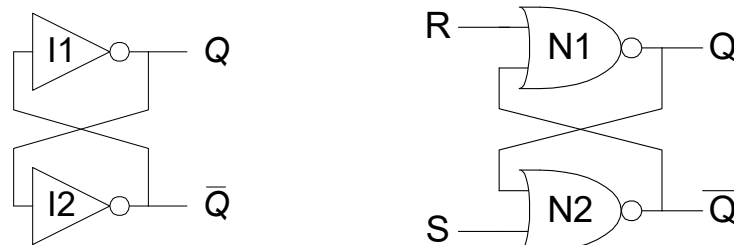


PML – IAC - 2023

21

## A latch SR

- O circuito bi-estável guarda um bit, mas não tem entrada que permita controlar o seu estado.
- Substituindo os NOT's por NOR's criamos duas entradas no circuito:
  - O SET e o RESET.

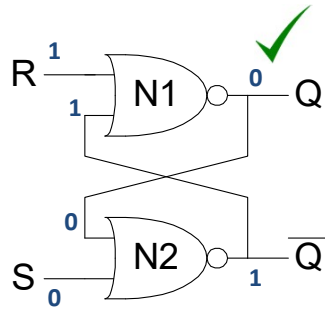


PML – IAC - 2023

22

## A latch SR

- Comportamento quando  $R=1$  e  $S=0$

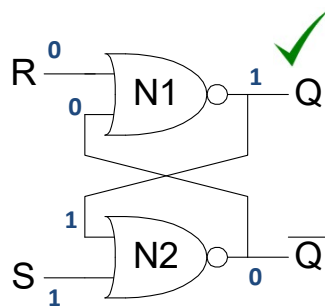


PML – IAC - 2023

23

## A latch SR

- Comportamento quando  $R=0$  e  $S=1$

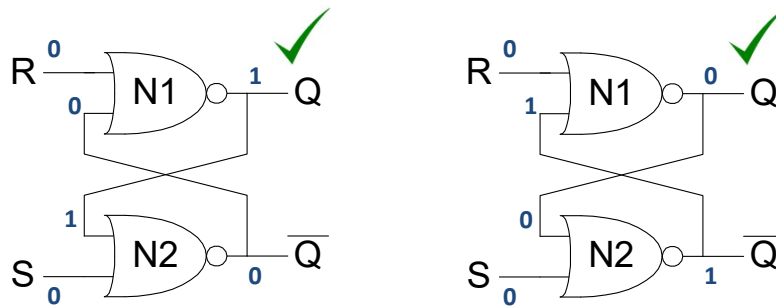


PML – IAC - 2023

24

## A latch SR

- Comportamento quando  $R=0$  e  $S=0$ 
  - Mantem o estado anterior!

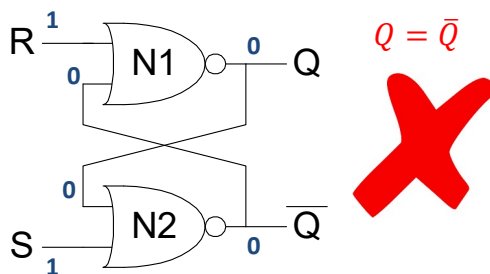


PML – IAC - 2023

25

## A latch SR

- Comportamento quando  $R=1$  e  $S=1$



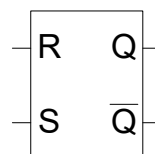
PML – IAC - 2023

26

## A latch SR

- A latch SR permite guardar um bit de estado (Q)
- O valor armazenado pode ser controlado usando as entradas Set ou Reset
- Tem um estado inválido. ( $S=1$  e  $R=1$ )
- Precisa de ser melhorada para evitar esse estado proibido.

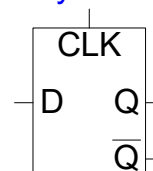
SR Latch  
Symbol



## A latch D

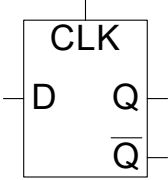
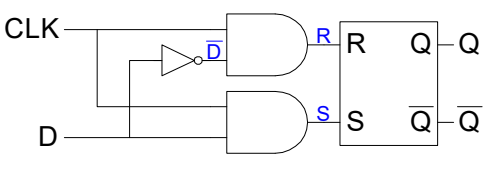
- A latch D tem duas entradas D e CLK
  - D são os dados a escrever na latch
  - CLK determina QUANDO os dados são escritos
- Funcionamento:
  - Quando CLK está a 1 é transparente para D ( $Q=D$ )
  - Quando CLK está a 0 guarda o valor anterior (é opaco)
- Evita o estado inválido.

D Latch  
Symbol



Circuitos Sequenciais

### A latch D: implementação



CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	$\overline{X}$	0	0	$Q_{prev}$	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

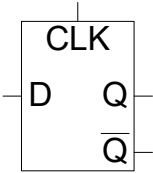
PML – IAC - 202329

Circuitos Sequenciais

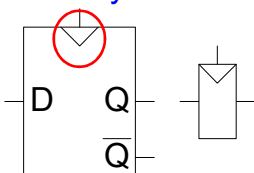
### O Flip-Flop D

- O Flip-Flop D funciona como a Latch D exceto que a escrita ocorre apenas quando o relógio transita de 0 para 1.
- O flip-flop é ativo na transição (*edge-triggered*)
  - Há apenas um instante em que a escrita é efetuada.

D Latch Symbol

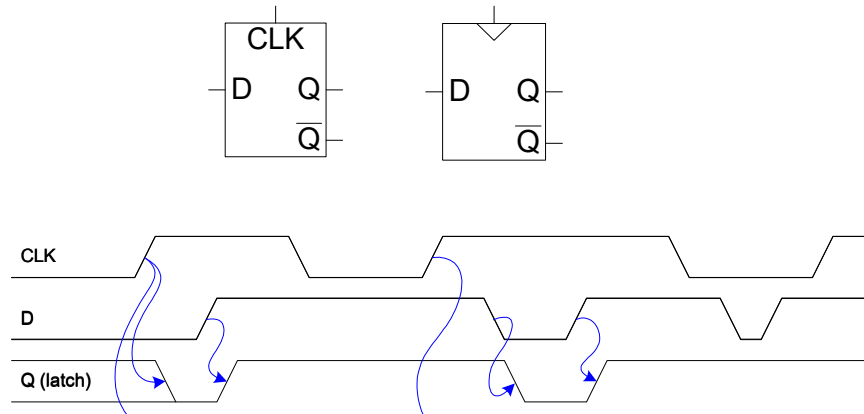


D Flip-Flop Symbols



PML – IAC - 202330

## O funcionamento do Flip-Flop D vs Latch D



PML – IAC - 2023

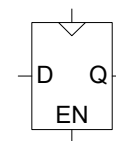
31

## O Flip-Flop D

### • Variantes do Flip-Flop D:

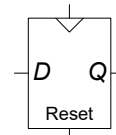
#### – Com enable

- controla quando são escritos dados
  - EN=0, não há escrita (mesmo que haja CLK).



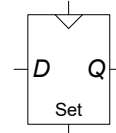
#### – Com reset

- Reset = 1, faz Q = 0, independentemente de D.



#### – Com Set

- Set = 1, faz Q = 1, independentemente de D.



PML – IAC - 2023

32



## Índice

- Os Blocos Combinatórios Básicos.
  - *Multiplexers*
  - Descodificadores
  - Somadores
- Circuitos Sequenciais
  - Latch R-S e FlipFlops tipo D
  - Registos e Memórias
  - Máquinas e diagramas de estado

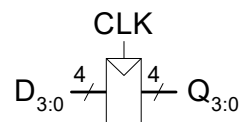
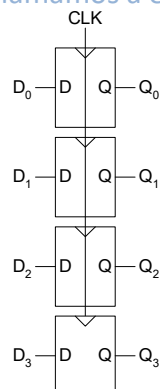
PML – IAC - 2023

33

Circuitos Sequenciais

## Registos

- Um Flip-Flop D, consegue armazenar um bit.
- Para armazenar N bits podemos agrupar N flip-flops D.
  - Chamamos a essa estrutura um registo de N bits.

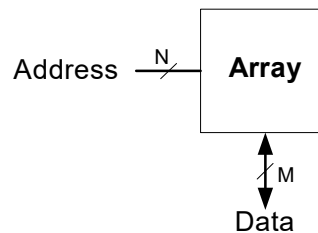


PML – IAC - 2023

34

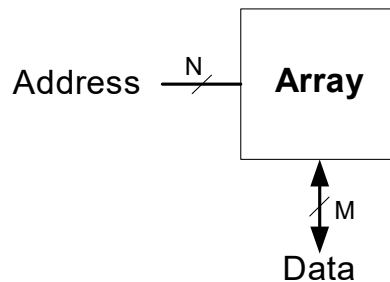
## Memórias

- Armazenam de modo eficiente grandes quantidades de dados.
- Os três tipos mais comuns são:
  - DRAM: Dynamic Random Access Memory
  - SRAM: Static Random Access Memory
  - ROM: Read Only Memory
- As memórias RAM são **voláteis**.
- As memórias ROM são **não voláteis**.



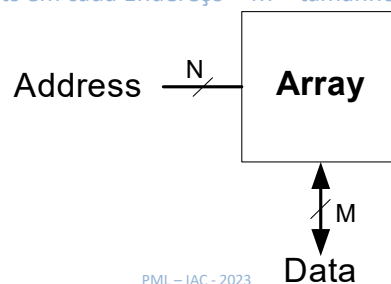
## Arrays de Memória

- A memória organiza-se como um *array* bidimensional de células de 1 bit.
- Com N bits de endereço e M bits de dados, a memória tem:
  - Número de Palavras (Depth):  $2^N$  linhas
  - Número de bits de cada palavra (Width): M colunas
  - Tamanho: Depth x Width =  $2^N \times M$



## Arrays de Memória

- Espaço de endereçamento vs Endereçabilidade
- Espaço de endereçamento
  - Conjunto de endereços possíveis = 0 a  $2^N - 1$
  - Dimensão do Espaço de endereçamento = número de palavras.
- Endereçabilidade
  - Número de bits em cada Endereço =  $M$  = tamanho da palavra.

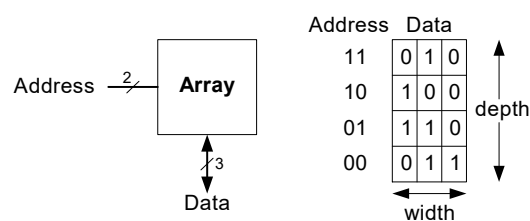


PML – IAC - 2023

37

## Exemplo de um Array de memória

- Um *array* de memória com 2 bits de endereço e 3 bits de dados.
  - Número de Palavras (Depth):  $2^2 = 4$
  - Tamanho da palavra: 3 bits
  - É um array de  $2^2 \times 3$  bits.
- A palavra armazenada no endereço 01 é 110.



PML – IAC - 2023

38

## Memória: Endereço e Espaço de Endereçamento

- **Endereço** (*address*) – um número (único) que identifica cada posição de memória. Os endereços são contados sequencialmente, começando em 0
- **Espaço de endereçamento** (*address space*) – a gama total de endereços que o CPU consegue referenciar (depende da dimensão do barramento de endereços).
  - Exemplo: um CPU com um barramento de endereços de 16 bits pode gerar endereços na gama: 0x0000 a 0xFFFF (i.e., 0 a  $2^{16}-1$ )
- Seja uma memória com  $2^k \times m$  bits
  - O endereço é um identificador de localização com  $k$  bits.
  - O espaço de endereçamento é o conjunto de endereços entre 0 e  $2^k-1$ .
  - O conteúdo da memória são as palavras de  $m$  bits guardadas em cada endereço.

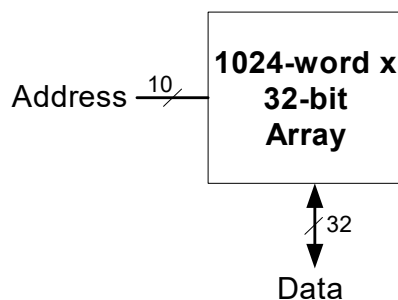
PML – IAC - 2023

39

Memórias

## Exemplo de um Array de memória

- Um array de memória com 10 bits de endereço e 32 bits de dados.
  - Número de Palavras (Número de Endereços):  $2^{10} = 1024$
  - Tamanho da palavra: 32 bits
  - É um array de  $1024 \times 32$  bits =  $1k \times 32$ .



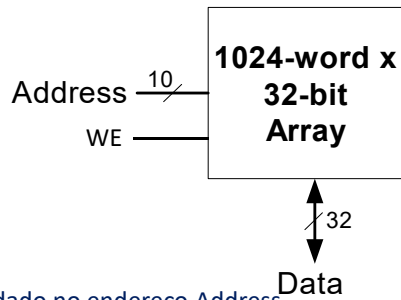
PML – IAC - 2023

40

## Array de memória

- Operações de leitura vs Operações de Escrita.

- Como distinguir?
  - WE = Write Enable

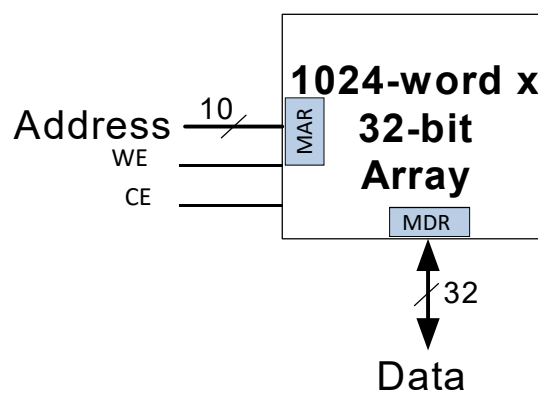


- Duas operações possíveis:
  - WE = 1 -> Escrita
    - O valor das linhas Data é guardado no endereço Address
  - WE = 0 -> Leitura
    - O valor guardado no endereço Address é colocado nas linhas Data.

\* Se o WE for *Active Low* funciona ao contrário

## Ciclo Básico de Acesso à Memória

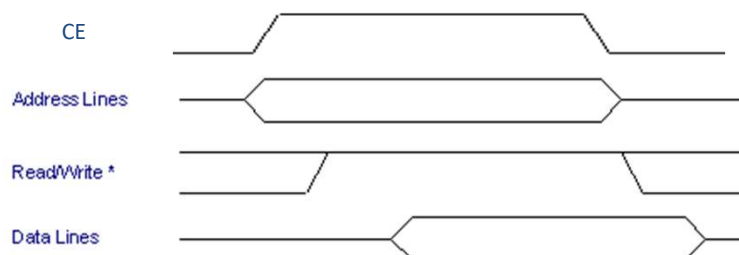
- Como se processa o acesso para leitura ou escrita na memória?



## Ciclo Básico de Acesso à Memória

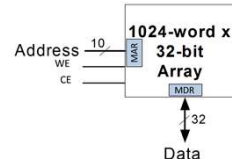
- Para uma Leitura (LOAD) A CPU:

- ativa o sinal de CE (seleciona a memória)
- Coloca o endereço a ler no barramento de endereços.
- Envia o sinal de Read (não escrita)
- Lê o conteúdo da memória no barramento de dados.



PML – IAC - 2023

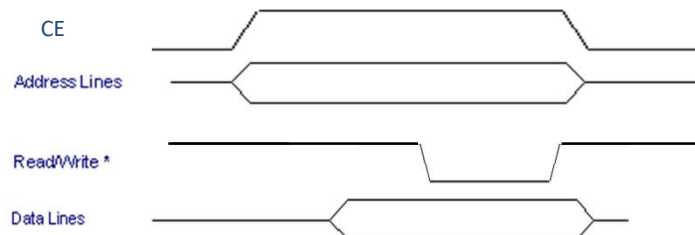
43



## Ciclo Básico de Acesso à Memória

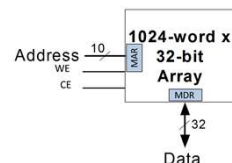
- Para uma escrita (STORE) A CPU:

- Ativa o sinal de CE (seleciona a memória)
- Coloca o endereço a ser escrito no barramento de endereços
- Coloca o valor a escrever no barramento de dados
- Ativa o sinal de Write



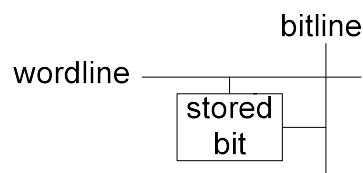
PML – IAC - 2023

44



## Células de memória

- Uma célula de memória é a unidade básica capaz de armazenar um bit
  - A célula de memória tem pelo menos dois interfaces
    - Um para saber quando é endereçada
    - Um para receber/transmitir a informação
  - Como a memória está organizada em “palavras” o endereçamento é comum a todos os bits de cada palavra: **wordline**.
  - **wordline** funciona como um enable, i.e., seleciona a linha a ser acedida.
    - **Só pode haver uma wordline ativa de cada vez.**

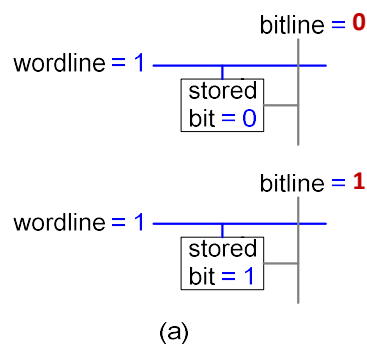


PML – IAC - 2023

45

## Funcionamento da Célula de memória

- Quando a *wordline* está ativa o conteúdo da célula passa para a *bitline*.
- Quando a *wordline* está inativa a célula não interfere com a *bitline*.
  - Está em alta impedância ou *tristate*.

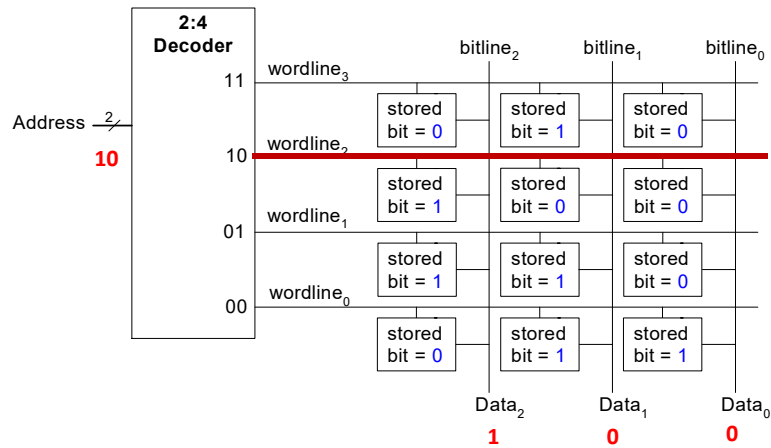


PML – IAC - 2023

46

## Organização da memória

- Um array de memória com dois bits de endereço e 3 bits de dados, pode esquematizar-se como:



PML – IAC - 2023

47

## (Nota Histórica) Um visionário: Robert Dennard

- Inventou a DRAM em 1966 na IBM
- Enfrentou o ceticismo dos colegas que não acreditavam que o processo funcionaria.
- Por volta de 1975 a DRAM estava em “todos” os computadores.



PML – IAC - 2023

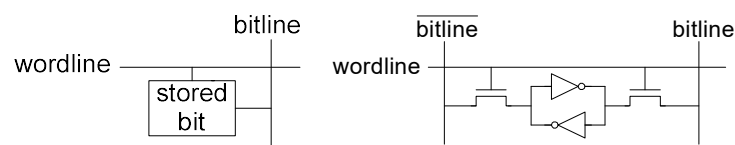
48



## RAM estática vs Dinâmica

- SRAM: Static RAM

- O bit é guardado em dois inversores acoplados.
- O valor é mantido enquanto o circuito estiver alimentado, por isso se chama **estática**.
- Cada célula de memória precisa de um circuito complexo.



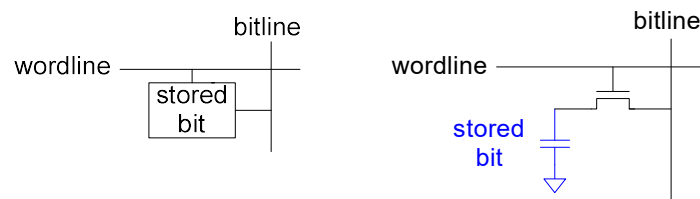
PML – IAC - 2023

49

## RAM estática vs Dinâmica

- DRAM: Dynamic RAM

- O bit é guardado como uma carga num condensador.
- O condensador tem perdas de carga (ao fim de algum tempo é impossível distinguir um 0 de um 1);
- A leitura da memória é destrutiva.
- Chama-se **dinâmica** porque o valor precisa de ser re-escrito periodicamente e também sempre que é lido.



PML – IAC - 2023

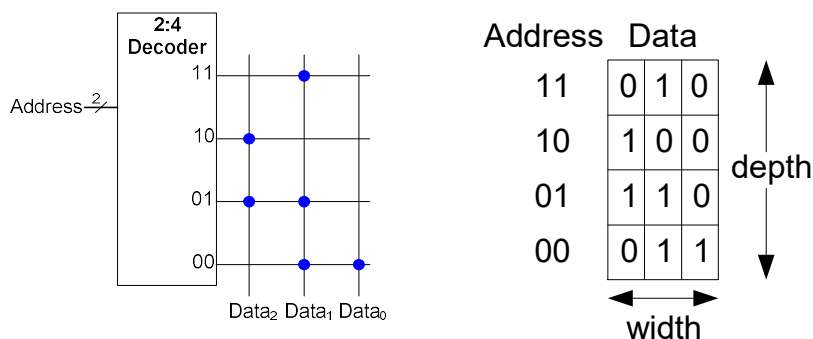
50

## Tipos de Memórias

- **RAM: Random Access Memory**
  - Volátil: perde a informação quando é desligada.
  - Pode ser lida e escrita rapidamente.
    - Chama-se Random Access Memory, porque se pode aceder com igual facilidade a qualquer posição de memória, ao contrário das memórias de acesso sequencial como as cassetes de fita.
- **ROM: Read Only Memory**
  - Não volátil: mantém a informação mesmo quando é desligada.
  - O acesso para leitura é rápido.
  - Acesso para escrita é impossível ou lento.
    - Chama-se Read Only Memory, porque as primeiras memórias de facto eram programadas ainda em fábrica ou escritas por um processo destrutivo.
    - Atualmente isto já não é verdade com a tecnologia Flash Memory.

## Memória ROM

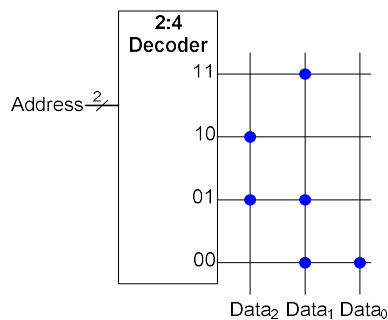
- **Notação Pontual:**
  - O decodificador opera como um gerador de mintermos.
  - Cada coluna faz o OR dos mintermos seleccionados assinalando-os com os pontos.



## Implementações Lógicas com ROM

- As saídas  $Data_2...Data_0$  implementam as seguintes funções lógicas:

$$\begin{aligned}
 - Data_2 &= (A_1 \cdot \overline{A_0} + \overline{A_1} \cdot A_0) = A_1 \oplus A_0 \\
 - Data_1 &= (A_1 A_0 + \overline{A_1} A_0 + \overline{A_1} \cdot \overline{A_0}) = \overline{A_1} + A_0 \\
 - Data_0 &= \overline{A_1} \cdot \overline{A_0}
 \end{aligned}$$



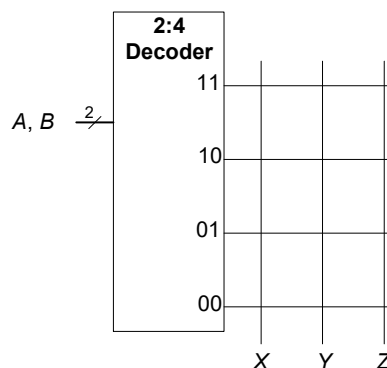
Address	Data			
11	0	1	0	depth ↑ ↓
10	1	0	0	
01	1	1	0	
00	0	1	1	
	width ←→			

PML – IAC - 2023

53

## Implementações Lógicas com ROM

- Exercício:
- Implementar as funções lógicas seguintes usando um array de memória  $2^2 \times 3$ :
  - $X = AB$
  - $Y = A + B$
  - $Z = A\overline{B}$



PML – IAC - 2023

54

### (Nota Histórica) **Outro Visionário: Fujio Masuoka**

- Trabalhou na Toshiba em circuitos rápidos e memórias.
- De forma não autorizada conduziu um projeto à noite e aos fins de semana no final da década de 70.
- O processo de limpar a memória pareceu-lhe semelhante ao flash de uma máquina.
- A Toshiba demorou a comercializar.
- A Intel lançou a memória flash no mercado em 1988.
- Atualmente é um negócio de 25 mil milhões de dólares por ano.



PML – IAC - 2023

55

## **Índice**

- Os Blocos Combinatórios Básicos.
  - *Multiplexers*
  - Descodificadores
  - Somadores
- Circuitos Sequenciais
  - Latch R-S e FlipFlops tipo D
  - Registos e Memórias
  - Máquinas e diagramas de estado

PML – IAC - 2023

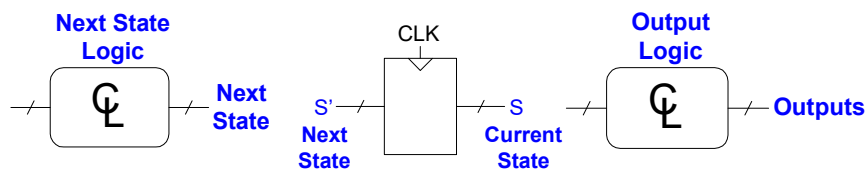
56

## Máquinas e Diagramas de Estado

- Uma máquina de estados é um circuito que:
  - Tem vários estados possíveis;
  - O estado atual é armazenado num registo (conjunto de flip-flops);
  - O estado muda na transição do relógio;
    - O sistema é sincronizado pelo relógio
- Regras de composição de circuitos sequenciais:
  - Cada elemento do circuito é um registo ou um circuito combinatório
  - Pelo menos um elemento do circuito é um registo
  - Todos os registos partilham o mesmo relógio
  - Todos os caminhos cíclicos têm de conter pelo menos um registo

## Máquinas e Diagramas de Estado

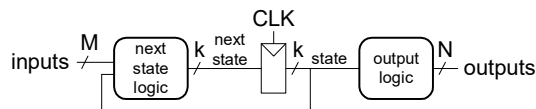
- Um registo de estado:
  - Guarda o estado atual
  - Carrega o próximo estado na transição do relógio
- Lógica combinatória:
  - Calcula o próximo estado
  - Calcula as saídas



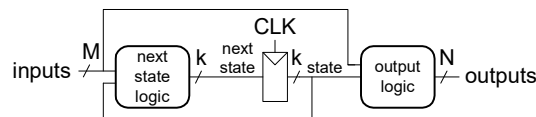
## Máquinas e Diagramas de Estado

- Máquinas de estado síncronas (Máquinas de Moore):
  - A saída depende apenas do estado atual.
  - i.e., só há mudanças na saída quando há transições de estado.
- Máquinas de estado assíncronas (Máquinas de Mealy):
  - A saída depende do estado e das entradas.

Moore FSM



Mealy FSM



PML – IAC - 2023

59

## Máquinas e Diagramas de Estado

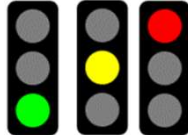
- Procedimento de projeto de Máquinas de Estados (Moore)
  1. Identificar entradas e saídas.
  2. Desenhar o diagrama de estados e transições.
  3. Escrever a tabela de transição de estados.
    - Em função do estado atual e das entradas.
  4. Escrever a tabela das saídas (em função do estado).
  5. Escrever (e simplificar) as equações booleanas:
    - da lógica do próximo estado
    - da lógica de saída.
  6. Desenhar o esquema do circuito.

PML – IAC - 2023

60

## Exercício: Semáforo

- Implementar uma máquina de estados que controle as luzes de um semáforo.
  - Deve implementar a sequência Verde, Amarelo, Vermelho.

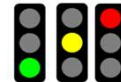


- Assume-se que os tempos de ativação de cada cor são iguais.
- Para já assume-se que não existe nenhuma intervenção dos utilizadores.

PML – IAC - 2023

61

## Exercício: Semáforo

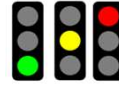


- 1º passo: Identificação de Entradas e Saídas
  - Entradas: Não tem. (não existe nenhuma intervenção dos utilizadores)
  - Saídas: G,Y, R (as cores do semáforo)
    - 1 = cor acesa
    - 0 = cor apagada

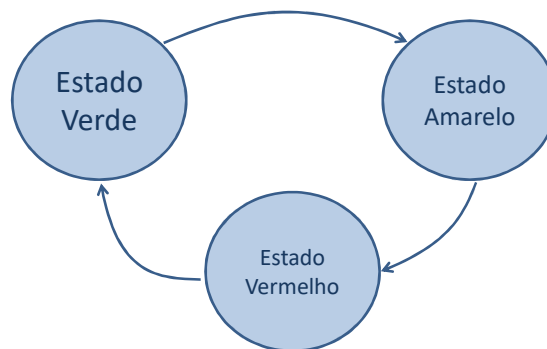
PML – IAC - 2023

62

## Exercício: Semáforo



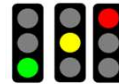
- 2º passo: Desenhar o Diagrama de Estados e transições:
  - O Sistema terá três estados: Verde, Amarelo e Vermelho
  - As transições entre estados são também indicadas.



PML – IAC - 2023

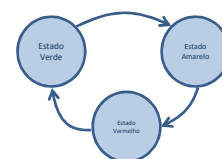
63

## Exercício: Semáforo



- 3º passo: Escrever a tabela de transições entre estados (em função do estado atual e das entradas):
  - Neste caso não há entrada!
- Primeira etapa é arranjar (escolher) a codificação de cada estado.
- Temos 3 estados -> precisamos de 2 bits para codificar cada estado.
  - Chamamos aos bits de estado  $S_1$  e  $S_0$

Estado	Codificação $S_1 S_0$
Estado Verde	0 0
Estado Amarelo	0 1
Estado Vermelho	1 0

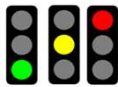


PML – IAC - 2023

64



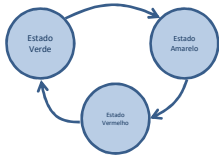
Exercício: Semáforo



- 3º passo: Escrever a tabela de transições entre estados (em função do estado atual e das entradas):
  - Neste caso não há entrada!
- 2ª etapa é a tabela de transições
  - $S_1^+$  e  $S_0^+$  representam o próximo valor de  $S_1 S_0$

Estado	Codificação $S_1 S_0$
Estado Verde	0 0
Estado Amarelo	0 1
Estado Vermelho	1 0

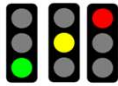
Estado Atual $S_1 S_0$	Estado Seguinte $S_1^+ S_0^+$
0 0	0 1
0 1	1 0
1 0	0 0
1 1	X X



PML – IAC - 2023

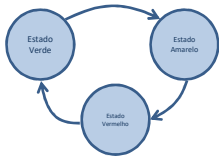
65

Exercício: Semáforo



- 4º passo: Escrever a tabela das saídas (em função do estado).

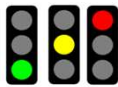
Estado Atual $S_1 S_0$	Saídas		
	G	Y	R
0 0	1	0	0
0 1	0	1	0
1 0	0	0	1
1 1	X	X	X



PML – IAC - 2023

66

Exercício: Semáforo



- 5º passo: Escrever e simplificar as equações da lógica do estado seguinte e das saídas.

Estado Atual $S_1S_0$	Estado Seguinte $S_1^+ S_0^+$
0 0	0 1
0 1	1 0
1 0	0 0
1 1	X X

$S_1^+ = \overline{S_1}S_0$

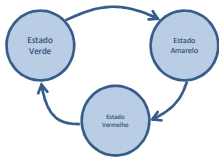
$S_0^+ = \overline{S_1}\overline{S_0}$

Estado Atual $S_1S_0$	Saídas		
	G	Y	R
00	1	0	0
01	0	1	0
10	0	0	1
11	X	X	X

$G = \overline{S_1}\overline{S_0}$

$Y = \overline{S_1}S_0$

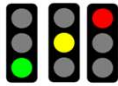
$R = S_1\overline{S_0}$



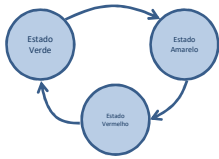
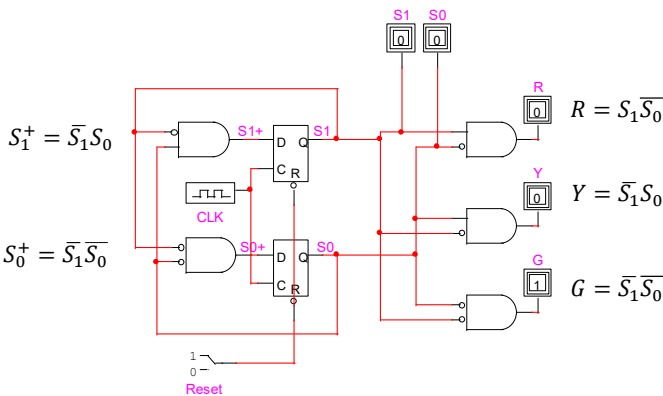
PML – IAC - 2023

67

Exercício: Semáforo



- 6º passo: Desenhar o esquema do circuito.

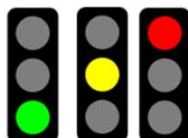


PML – IAC - 2023

68

### Exercício: Semáforo (com entrada de peões)

- Implementar uma máquina de estados que controle as luzes de um semáforo.
  - Deve implementar a sequência Verde, Amarelo, Vermelho.



- Assume-se que os tempos de ativação de cada cor são iguais.
- Se o peão pressionar o botão o próximo estado é **Vermelho** independentemente do estado atual.

PML – IAC - 2023

69

### Exercício: Semáforo (com entrada de peões)

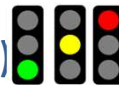


- 1º passo: Identificação de Entradas e Saídas
  - Entradas: Peão (P)
  - Saídas: G,Y, R (as cores do semáforo)
    - 1 = cor acesa
    - 0 = cor apagada

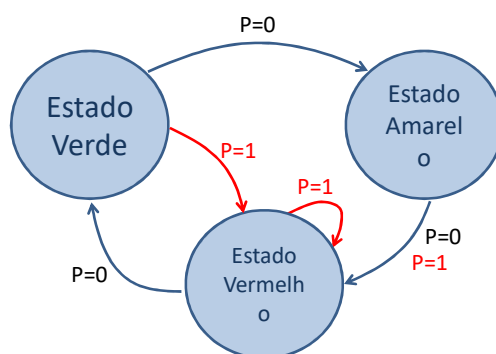
PML – IAC - 2023

70

## Exercício: Semáforo (com entrada de peões)



- 2º passo: Desenhar o Diagrama de Estados e transições:
  - O Sistema terá três estados: Verde, Amarelo e Vermelho
  - As transições entre estados são também indicadas em função da entrada P.



PML – IAC - 2023

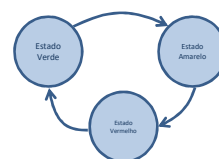
71

## Exercício: Semáforo (com entrada de peões)



- 3º passo: Escrever a tabela de transições entre estados (em função do estado atual e das entradas):
  - Neste caso há entrada, mas mantém-se os três estados.

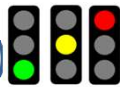
Estado	Codificação $S_1S_0$
Estado Verde	00
Estado Amarelo	01
Estado Vermelho	10



PML – IAC - 2023

72

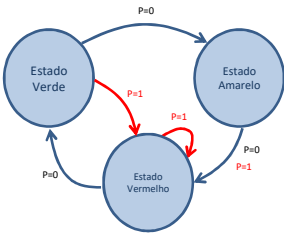
Exercício: Semáforo (com entrada de peões)



- 3º passo: Escrever a tabela de transições entre estados (em função do estado atual e das entradas):
  - Neste caso há entrada P.
- A tabela de transições agora depende do estado atual e da entrada P.

Estado	Codificação $S_1S_0$
Estado Verde	00
Estado Amarelo	01
Estado Vermelho	10

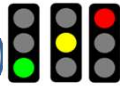
Estado Atual $S_1S_0$	Entrada P	Estado Seguinte $S_1^+S_0^+$
00	0	01
00	1	10
01	0	10
01	1	10
10	0	00
10	1	10
11	0	XX
11	1	XX



PML – IAC - 2023

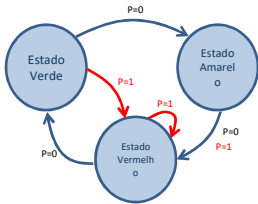
73

Exercício: Semáforo (com entrada de peões)



- 4º passo: Escrever a tabela das saídas (em função do estado).
- Igual ao exemplo anterior:
  - A saída só depende do estado atual.

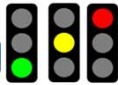
Estado Atual $S_1S_0$	G	Y	R
00	1	0	0
01	0	1	0
10	0	0	1
11	X	X	X



PML – IAC - 2023

74

Exercício: Semáforo (com entrada de peões)



- 5º passo: Escrever e simplificar as equações da lógica do estado seguinte e das saídas.
- As equações das saídas G,Y e R mantêm-se.

Estado Atual $S_1 S_0$	Entrada $P$	Estado Seguinte $S_1^+ S_0^+$
00	0	01
00	1	10
01	0	10
01	1	10
10	0	00
10	1	10
11	0	XX
11	1	XX

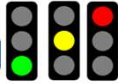
$S_1 S_0$		$S_1^+$			
P		00	01	11	10
0	0	0	1	X	0
1	1	1	1	X	1

$S_1^+ = P + S_0$

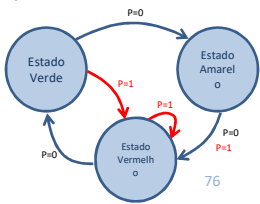
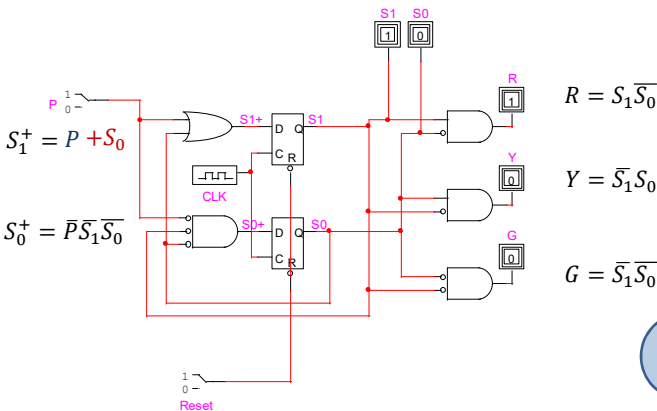
$S_1 S_0$		$S_0^+$			
P		00	01	11	10
0	1	0	X	X	0
1	0	0	X	X	0

$S_0^+ = \bar{P}\bar{S}_1\bar{S}_0$

Exercício: Semáforo (com entrada de peões)

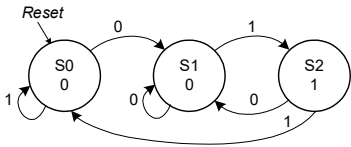


- 6º passo: Desenhar o esquema do circuito.



Exercício 2: Detetor de sequências

- Exercício:
- A Alice quer implementar um detetor de sequências que recebe como entrada 0's e 1's e ativa a saída sempre que detetar a sequência 01.
- 1º identificar entradas e saídas:
  - Entradas: o bit da sequência
  - Saída: 1 se as entradas anteriores foram 01, 0 nos outros casos.
- 2º desenhar o diagrama de estados e transições



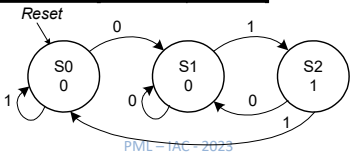
State	Encoding
S0	00
S1	01
S2	10

Exercício: Detetor de sequências

- 3º escrever a tabela de transições de estados:

Current State		Inputs	Next State	
$S_1$	$S_0$		$S'_1$	$S'_0$
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

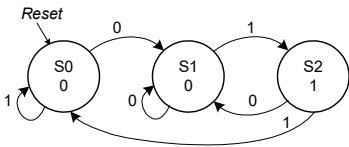
State	Encoding
S0	00
S1	01
S2	10



Exercício: Detetor de sequências

- 4º escrever a tabela das saídas em função do estado:

Current State		Output
$S_1$	$S_0$	$Y$
0	0	
0	1	
1	0	



Exercício: Detetor de sequências

- 5º escrever as equações booleanas da lógica de saída e da lógica do próximo estado:
  - $S_0' = \dots$
  - $S_1' = \dots$
  - $Y = \dots$

Current State		Inputs	Next State	
$S_1$	$S_0$		$S_1'$	$S_0'$
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

$S_1 S_0$		$S_1'$			
A		00	01	11	10
0	0	0	X	0	
1	0	1	X	0	

$S_1' = AS_0$

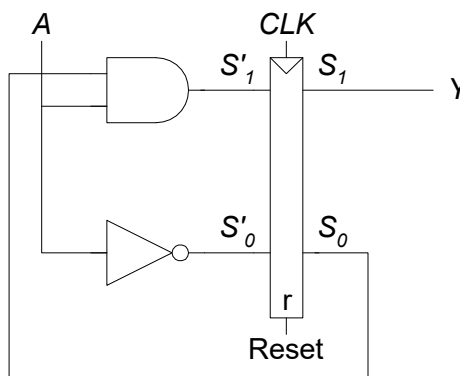
$S_1 S_0$		$S_0'$			
A		00	01	11	10
0	0	1	1	X	1
1	0	0	X	0	

$S_0' = \bar{A}$



## Exercício: Detetor de seqüências

- 5º escrever as equações booleanas da lógica de saída e da lógica do próximo estado:
  - $S_1' = AS_0$
  - $S_0' = \bar{A}$
  - $Y = S_1$
- 6º desenhar o circuito



PML – IAC - 2023

81

## Máquinas e Diagramas de Estado

- Exercício:
- Projetar uma máquina de estados que implemente um contador módulo 4.
- A seqüência de contagem é 0, 1, 2, 3, 0, 1, 2, 3, 0, ...
- Esta máquina de estados tem duas particularidades:
  - A saída atual é igual ao estado atual
  - O próximo estado apenas depende do estado atual

PML – IAC - 2023

82