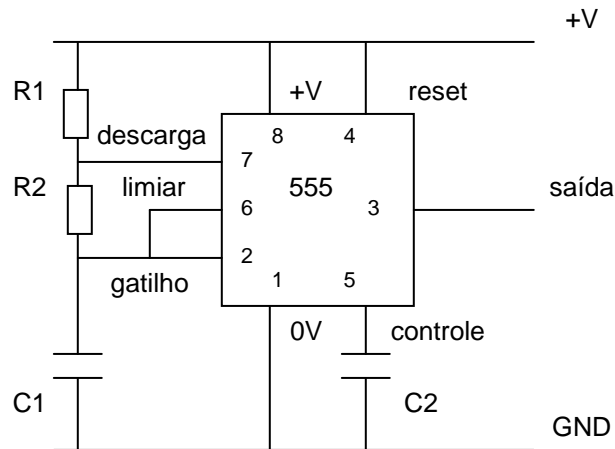


### Controle de temporização de circuitos

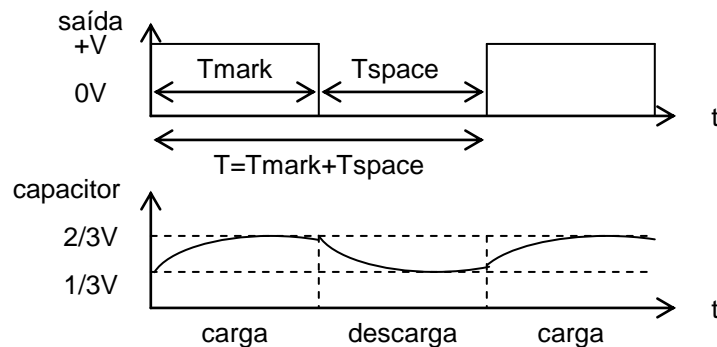
Um dos circuitos integrados mais versáteis para controle de tempo é o 555, capaz de funcionar em três modos:

- **monoestável** modo no qual o circuito produz um único disparo  
aplicações: temporização, chaveamento sem ressaltos, divisores de frequência, modulação de largura de pulso (PWM) etc.
- **astável** modo no qual o circuito opera como um oscilador, capaz de alternar regularmente entre estados altos e baixos  
aplicações: acionamentos de LEDs, geradores de tons, alarmes, modulação de posição de pulso, **clocks** etc.
- **biestável** modo no qual o circuito opera como um **flip-flop**, capaz de permanecer em um de dois estados indefinidamente  
aplicações: chaveamento sem ressaltos (**bouncefree latched**) e registradores (memória)

O diagrama abaixo representa a configuração típica para um oscilador em modo astável:



O capacitor C1 é carregado pela corrente que passa por R1 e R2. Quando a carga alcança  $2/3$  da tensão de alimentação (+V), o limiar é atingido, a saída vai para nível baixo e o pino de descarga é conectado a 0V. Quando a descarga da corrente que passa por R2 atinge  $1/3$  da tensão de alimentação, a saída vai para nível alto e cessa a descarga permitindo a recarga do capacitor. O ciclo se repetirá continuamente até que o pino de **reset** seja conectado a 0V.



Um ciclo de trabalho (carga e recarga) ocorre durante o período (T) da onda quadrada, o qual inclui o tempo de marcação (Tm) e o tempo de espaçamento (Ts):

$$T = T_m + T_s = [0,7 \times (R_1 + R_2) \times C_1] + [0,7 \times R_2 \times C_1] = 0,7 \times (R_1 + 2R_2)$$

onde

T – período [s]  
 Tm – tempo de marcação [s]  
 Ts – tempo de espaçamento [s]  
 R1 – resistor [ohms]  
 R2 – resistor [ohms]  
 C1 – capacitor [F]

A frequência de oscilação [Hz] é o número de ciclos de trabalho por segundo:

$$f = \frac{1}{T} = \frac{1,4}{(R_1 + 2R_2) \times C_1}$$

Para que o circuito funcione no modo astável, o tempo de marcação (Tm) deverá ser praticamente igual ao tempo de espaçamento (Ts). Isso acontecerá se o valor de R2 for muito maior que R1, nesse caso o valor da frequência será dado por:

$$f = \frac{0,7}{R_2 \times C_1}$$

Exemplo:

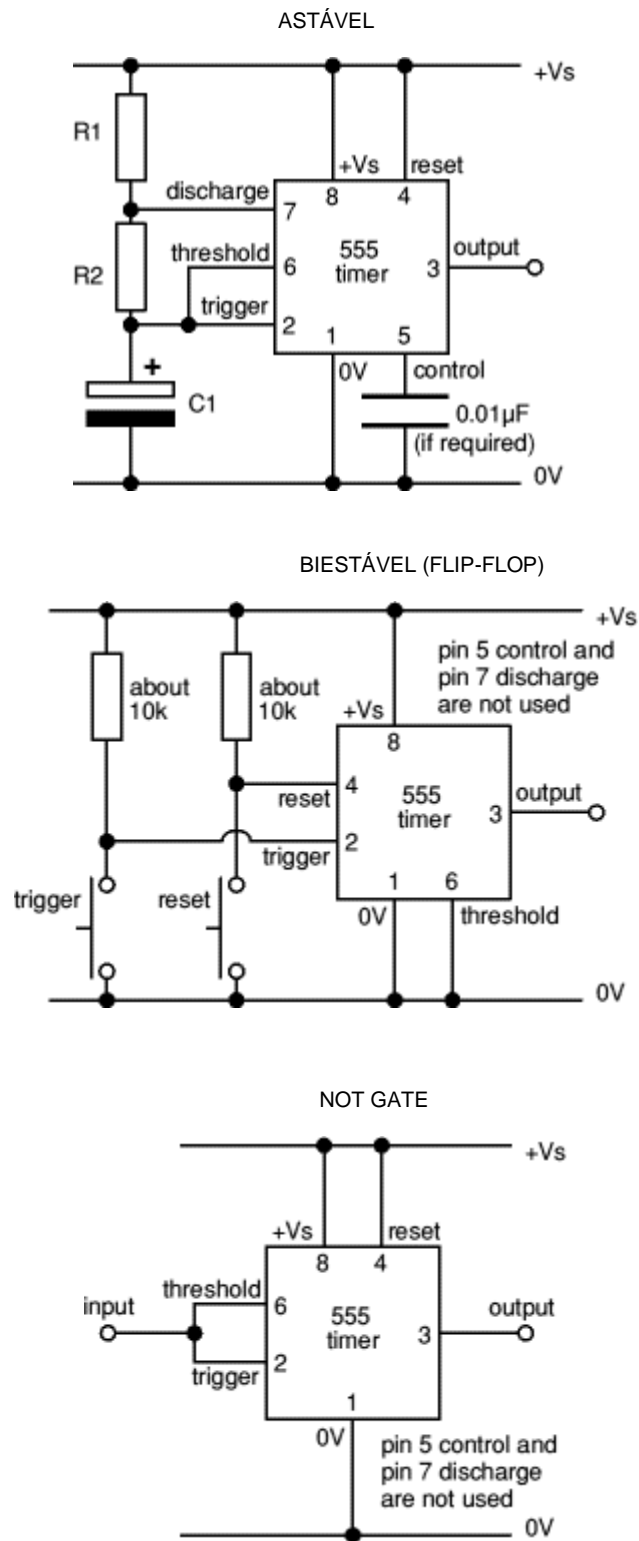
Com os valores dados abaixo:

R1 = 1 KΩ  
 R2 = 68 KΩ  
 C1 = 10 μF  
 C2 = 0,1 μF (para estabilização)

A frequência será de

$$f = 0,7 / (68 \times 10^3 \times 10 \times 10^{-6}) \approx 1 \text{ Hz}$$

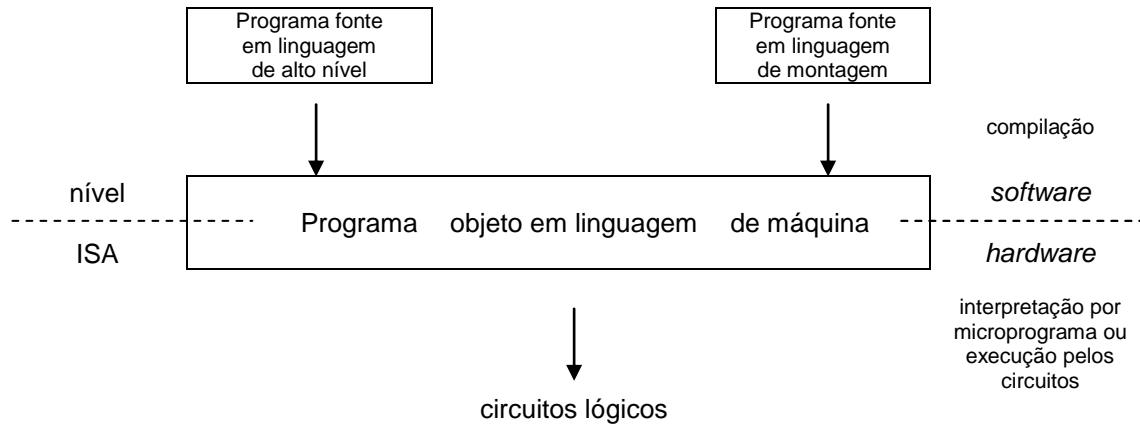
## Exemplos de usos do temporizador



### Interface software-hardware (ISA – *Instruction Set Architecture*)

O nível ISA (*Instruction Set Architecture*) define como uma máquina se apresentará ao programador:

- quais as instruções em linguagem de máquina
- qual o modelo de memória (quantidade de bits, alinhamento etc.)
- quantidade e tipos dos registradores (uso geral, apontadores, pilha, de estado)
- tipos de dados disponíveis (numéricos e não-numéricos: lógicos, cadeias etc.)



Em termos práticos, cada nível possui uma definição adequada à sua expressão:

language de alto nível:	A = 10;
linguagem de montagem:	LDA 0ah
linguagem de máquina:	00110011 00001010

### Formatos de instruções

O formato das instruções está relacionado ao modelo de memória, à quantidade de processadores, ao tempo de decodificação, ao tempo de execução (busca de operandos) e à quantidade de memória endereçável.

Os formatos podem ser constituídos por códigos de instrução (opcode) e operandos/endereços.

Os formatos podem ser fixos ou variados, com 0, 1, 2 ou mais operandos/endereços, dependendo da arquitetura e dos modos de endereçamento (memória e registradores).

código (opcode)			
código (opcode)	operando / endereço		
código (opcode)	endereço1	endereço2	
código (opcode)	endereço1	endereço2	endereço3

Tipos de formatos de instrução

## Modos de endereçamento

## 1. Implícito

- instrução traz o próprio o endereçamento (não há operando)

								instrução
--	--	--	--	--	--	--	--	-----------

## Exemplo:

No Intel 8080:

STC – Set Carry Flag

RAL – Rotate Accumulator Left

RLC – Rotate Accumulator through Carry

DAA – Decimal Adjust Accumulator

## 2. Imediato

- instrução traz o próprio operando (dado/constante)

								instrução
X	X	X	X	X	X	X	X	operando

								instrução
A	A	A	A	A	A	A	A	operando/endereço
A	A	A	A	A	A	A	A	

## Exemplo:

No Intel 8080:

ADI data – Add Immediate to accumulator

LDI data – Load Immediate to accumulator

JMI address – Jump Immediate to address

CPI data – Compare Immediate with accumulator

## Uso:

- valor constante
- valor inicial para contador
- armazenar endereço (ponteiro) em registrador
- indicar quantidade de posições em deslocamento de bits

## Vantagens:

- uso com valores constantes
- operando obtido durante o ciclo de busca (apenas 1 acesso)

## Desvantagens:

- tamanho do dado limitado à quantidade de bits
- não há flexibilidade para se alterar dados que variem a cada execução do programa

## 3. Direto

- instrução traz o endereço do operando (dado) na memória

								instrução
A	A	A	A	A	A	A	A	operando/endereço
A	A	A	A	A	A	A	A	

Exemplo:

No Intel 8080:

LDA address – Load accumulator with Address content

JMP address – Jump to address

Uso:

- indicar posição em memória

Vantagens:

- referência direta à memória

Desvantagens:

- tamanho do endereço limitado à quantidade de bits
- mais lento que o modo imediato (mais ciclos para busca do operando durante execução)

## 4. Indireto via registrador

- instrução indica o(s) registrador(es) que contém o endereço (apontador) do dado na memória

				R	R	r	r	instrução/registorador(es)
--	--	--	--	---	---	---	---	----------------------------

Exemplo:

No Intel 8080:

ADD M – Add Memory addressed by register pair (HL) to accumulator

MOV M, register – Move register to Memory addressed by register pair (HL)

Uso:

- indicação do dado

Vantagens:

- referência indireta à memória (estruturas de dados mais complexas)

Desvantagens:

- tamanho de registradores limitado
- mais lento que o modo imediato (mais ciclos para busca do operando durante execução)

## 5. Direto via registrador

- instrução indica o registrador que contém o dado



Exemplo:

No Intel 8080:

ADD register – Add register to accumulator

DCR register – Decrement register

Uso:

- contador

Vantagens:

- tamanho da instrução pequeno
- referência direta a registrador (não faz acesso à memória)

Desvantagens:

- quantidade registradores limitada
- mais lento que o modo imediato (mais ciclos para busca do operando durante execução)

## 6. Indexado

- instrução opera sobre o endereço obtido pela soma do operando a um registrador (índice)

Exemplo:

No Intel 8086:

LDX register, operand – Load register with memory addressed by (register+operand)

ADX register, operand – Add register with memory addressed by (register+operand)

Uso:

- para acessar dado em arranjo

Vantagens:

- referência rápida para acesso a dados contíguos na memória

Desvantagens:

- tamanho

## 7. Modo (base+deslocamento)

- instrução opera sobre o endereço obtido pela soma do operando (deslocamento) ao endereço contido em um registrador (base)

Exemplo:

ADD [base+index\_register+offset], register – Add register to memory address

Uso:

- para segmentação e para realocação de dados/programas na memória

Vantagens:

- referência rápida para acesso a porções contíguas na memória

Desvantagens:

- tamanho limitado

## 8. Combinado

- instruções que combinam modos de endereçamento: direto (ou imediato) + indireto (via pilha)

## Paralelismo

### Tipos de paralelismo

Há quatro tipos básicos de paralelismo:

- ao nível de bits: o qual se obtém pelo aumento da palavra do processador, permitindo-se executar operações com dados maiores que o comprimento da mesma;
- ao nível de instrução: o qual se obtém otimizando os passos básicos para o tratamento de uma instrução: busca (IF – **instruction fetch**), decodificação (ID – **instruction decode**), execução (EX – **execute**), acesso à memória (MEM) e reescrita (WB – **writeback**).

#### Subescalar

IF	ID	EX	MEM	WB					
					IF	ID	EX	MEM	WB

#### Superescalar

IF	ID	EX	MEM	WB					
	IF	ID	EX	MEM	WB				
		IF	ID	EX	MEM	WB			
			IF	ID	EX	MEM	WB		
				IF	ID	EX	MEM	WB	

- de dados: o qual se obtém distribuindo-se dados entre diversos nodos computacionais para que sejam processados em paralelo;
- de tarefas: o qual se obtém executando-se processamentos diferentes sobre um mesmo conjunto de dados ou sobre diferentes conjuntos de dados.

### Tipos de computadores paralelos

Os tipos mais comuns de computadores paralelos são: com vários núcleos (**multicore**); simétricos (vários processadores com uma mesma memória compartilhada); distribuídos (conectados por rede: **cluster**, massivamente paralelos; **grid**); e os especializados (reconfiguráveis, de uso geral, específicos para aplicações e vetoriais).



## Taxonomia de Flynn

SISD	SIMD
<b>Single Instruction Single Data</b>	<b>Single Instruction Multiple Data</b>
MISD	MIMD
<b>Multiple Instruction Single Data</b>	<b>Multiple Instruction Multiple Data</b>

- **Single Instruction – Single Data**

**Single instruction:** apenas uma sequência de instruções pode ser executada por uma unidade de controle durante um ciclo de **clock**

**Single data:** apenas uma sequência de dados pode ser usada como entrada durante um ciclo de **clock**

Tipo de problema: uso geral

Tipo de execução: determinística

Tipo de computador: mainframes, minicomputadores e a maioria dos computadores atuais

- **Single Instruction – Multiple Data**

**Single instruction:** apenas uma sequência de instruções pode ser executada em todas as unidades de controle durante o mesmo ciclo de **clock**

**Multiple data:** apenas uma sequência de dados pode ser usada como entrada durante o mesmo ciclo de **clock**

Tipo de problema: com grande regularidade tais como os de processamento de imagens

Tipo de execução: síncrona (**lockstep**) e determinística

Tipo de computador: arranjos de processadores, **pipelines** vetoriais, processadores com GPU's

- **Multiple Instruction – Multiple Data**

**Multiple instruction:** sequências independentes de instruções podem ser executadas em todas as unidades de controle durante o mesmo ciclo de **clock**

**Single data:** apenas uma sequência de dados pode ser usada como entrada durante um ciclo de **clock**

Tipo de problema: filtros de frequência e decifrar mensagem por vários algoritmos

Tipo de execução: assíncrona

Tipo de computador: Carnegie-Mellon C.mmp Computer (1971)

- **Multiple Instruction – Multiple Data**

**Multiple instruction:** sequências independentes de instruções podem ser executadas em todas as unidades de controle durante o mesmo ciclo de **clock**

**Single data:** sequências independentes de dados podem ser usadas como entrada durante um ciclo de **clock**

Tipo de problema: filtros de frequência e decifrar mensagem por vários algoritmos

Tipo de execução: síncrona e assíncrona, determinística ou não-determinística

Tipo de computador: supercomputadores, **clusters**, **grids**, computadores com múltiplos núcleos

## Arquiteturas de memória para computadores paralelos

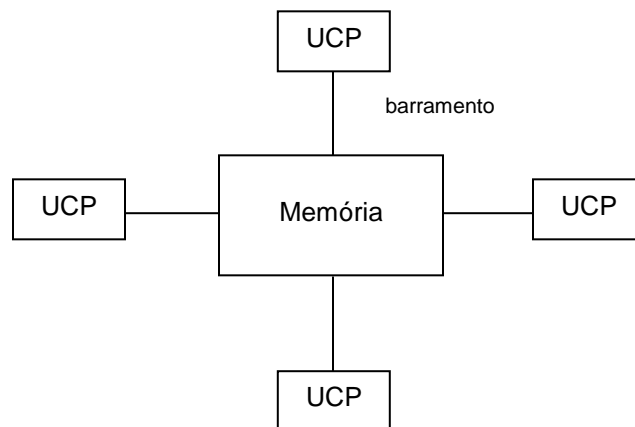
- Memória compartilhada (**shared memory**)

Vantagens: espaço de endereçamento global facilitado para o usuário  
compartilhamento rápido e uniforme pelos processadores  
coerência de cache mantida por **hardware**

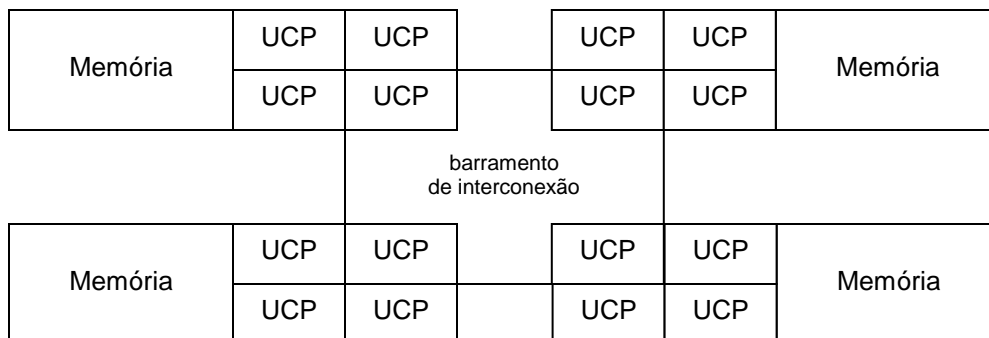
Desvantagens: falta de escalabilidade  
(aumentar processadores implica maior tráfego no barramento)  
maior responsabilidade para o programador para garantir acesso “correto”  
(controle de sincronização)  
maior custo  
(para projetar e produzir memórias com múltiplos acessos)

- Modelo UMA (**Uniform Memory Access**)

- Tipo de memória: a mesma com tempo de acesso igual para todos os processadores
- Tipo de computador: multiprocessadores simétricos (SMP)

- Modelo NUMA (**Non-Uniform Memory Access**)

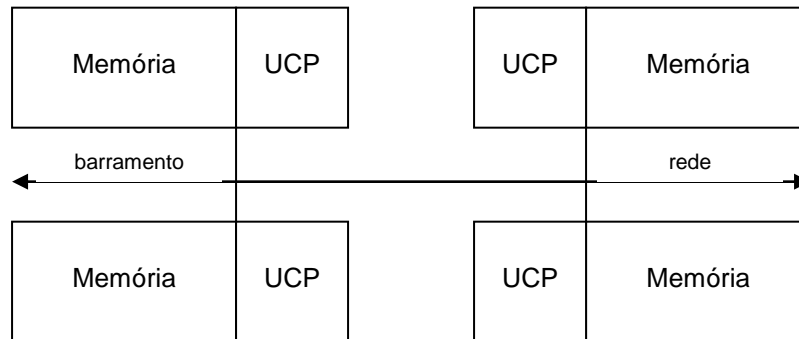
- Tipo de memória: a mesma com tempo de acesso diferente entre processadores
- Tipo de computador: conexões entre multiprocessadores simétricos (SMP)



- Memória distribuída (***distributed memory***)

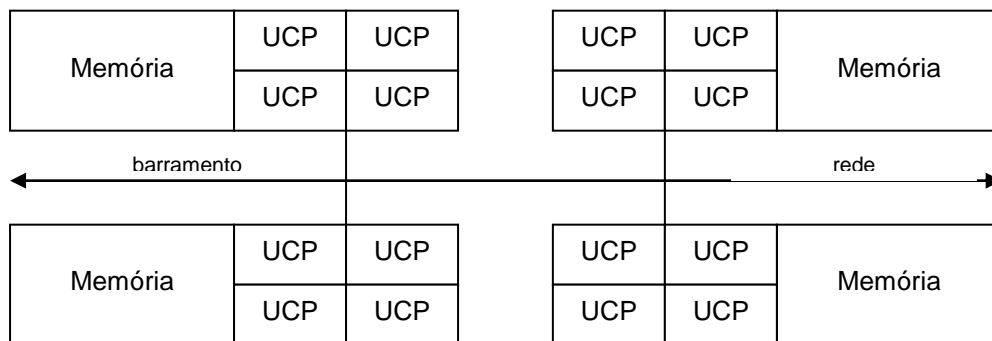
Vantagens: escalabilidade  
 acesso rápido à memória local pelo processador  
 não há trabalho extra (***overhead***) para manter coerência  
 custo compatível com processadores comuns e recursos de rede

Desvantagens: maior responsabilidade para o programador  
 para garantir comunicação de dados entre processadores  
 dificuldade em manter estruturas de dados  
 baseadas em memória global  
 tempo de acesso não uniforme



- Memória híbrida (***hybrid distributed-shared memory***)

Modelo comum à maioria dos processadores simétricos (SMP) atuais.  
 Os requisitos de acesso à memória dependem da infraestrutura de rede.



## Modelos de programação paralela

### - Memória compartilhada

Tarefas compartilham um espaço de endereçamento comum no qual podem ler e escrever assincronamente. Mecanismos como travas (**locks**) e semáforos (**semaphores**) podem ser usados para o controle de acesso à memória compartilhada.

Vantagens: simplicidade no desenvolvimento de programas  
não há necessidade de um controle explícito da comunicação entre tarefas

Desvantagens: dificuldade em gerenciar a localidade dos dados  
(manter dados locais conserva os acessos à memória, evita refrescamento de cache e tráfego no barramento, mas isso pode ser dificultado pelo uso de vários processadores ao mesmo tempo)

### - **Threads**

Um processo simples pode ter múltiplas sequências de execução concorrentes, cada um com acesso a dados locais, mas compartilhando recursos de um mesmo programa, como o espaço de memória global, através do qual se comunicam usando mecanismos de sincronização.

### - Troca de mensagens

Um conjunto de tarefas que trabalham sobre suas próprias memórias locais podem se comunicar através de troca de mensagens, e assim agir de forma cooperativa.

### - Dados paralelos

Um conjunto de tarefas trabalha sobre o mesmo conjunto de dados, mas cada uma delas lida com uma porção específica da estrutura de dados.

### - Híbridos

Combinações de dois ou mais modelos de programação paralela, como por exemplo, troca de mensagens com **threads** ou memória compartilhada. Outro modelo pode combinar dados paralelos com troca de mensagens.

### - SPMD (**Single Program Multiple Data**)

Um mesmo programa é executado por todas as tarefas simultaneamente, mas não exatamente as mesmas instruções ao mesmo tempo, uma vez que cada tarefa lidará com uma porção diferente do conjunto de dados.

### - MPMD (**Multiple Program Multiple Data**)

Vários programas podem estar ativos, trabalhando em paralelo.

## Prova de teoremas

As relações descritas podem ser utilizadas em Lógica Formal para provar teoremas.

Exemplo 1:

Provar ( s' ) dados:

1. t - premissa
2.  $t \rightarrow q'$  - premissa
3.  $q' \rightarrow s'$  - premissa
4.  $q'$  - *modus ponens* entre 1 e 2
5.  $s'$  - *modus ponens* entre 3 e 4 (c.q.d.)

Exemplo 2:

Provar ( a ) dados :

1.  $a' \rightarrow b$  - premissa
2.  $b \rightarrow c$  - premissa
3.  $c'$  - premissa
4.  $b'$  - *modus tolens* entre 2 e 3
5.  $(a')'$  - *modus tolens* entre 1 e 4
6. a - dupla negação (c.q.d.)

## Quantificadores

A Lógica de Predicados, ou Lógica de Primeira Ordem, associa o uso de quantificadores.

## Quantificador Universal

$(\forall x \in S) P(x)$  - **para todo** x em S, P(x) é verdadeiro

## Quantificador Existencial

$(\exists x \in S) P(x)$  - **para algum** x em S, P(x) é verdadeiro

Os quantificadores admitem complementação :

$$\text{não } \left( (\forall x) P(x) \right) = (\exists x) (\text{não } P(x))$$

$$\text{não } \left( (\exists x) P(x) \right) = (\forall x) (\text{não } P(x))$$

Exemplo :

Supor o predicado abaixo, devidamente quantificado :

$$(\forall x \in R) (x^2 + 2x - 1 > 0)$$

$$\text{não } \left( (\forall x \in R) (x^2 + 2x - 1 > 0) \right) \rightarrow (\exists x \in R) (x^2 + 2x - 1 \leq 0)$$

o que pode ser verificado quando  $x = 0$  !

**Exercícios propostos**

1. Provar, pela álgebra, que :

- a)  $(p \cdot q') + p' = p' + q'$
- b)  $p \cdot (p' + q) = p \cdot q$
- c)  $p \cdot q + (p' + q')' = (p' + q')'$
- d)  $(p \cdot q + r) \cdot ((p \cdot q)' \cdot r') = 0$
- e)  $(p \cdot q + r') \cdot (p \cdot q \cdot r' + p \cdot r') = p \cdot r'$

2. Fazer as tabelas e os circuitos do exercício anterior.

3. Demonstrar as relações abaixo através de tabelas:

- a)  $x + x \cdot y = x$  (absorção)
- b)  $x \cdot (x + y) = x$  (absorção)
- c)  $(x + y) \cdot (x + z) = x + y \cdot z$
- d)  $x + x' \cdot y = x + y$
- e)  $x \cdot y + y \cdot z + y' \cdot z = x \cdot y + z$

4. Verificar o resultado das seguintes relações :

- a)  $(x \rightarrow y) \cdot x \rightarrow y$  ("modus ponens")
- b)  $(x \rightarrow y) \cdot y' \rightarrow x'$  ("modus tolens")
- c)  $(x + y) \cdot x' \rightarrow y$  (silogismo disjuntivo)
- d)  $(x \rightarrow y) \cdot (y \rightarrow z) \rightarrow (x \rightarrow z)$  (silogismo hipotético)
- e)  $((x \rightarrow y) \cdot (w \rightarrow z)) \cdot (x+w) \rightarrow (y+z)$  (dilema)
- f)  $(x \rightarrow y) \rightarrow (x \rightarrow x \cdot y)$  (absorção)
- g)  $x \cdot y \rightarrow x$  (simplificação)
- h)  $x \rightarrow x + y$  (adição)
- i)  $(x' \rightarrow x) \rightarrow x$  (contradição)
- j)  $(x' \rightarrow y) \rightarrow ((x' \rightarrow y') \rightarrow x)$  (contradição)
- h)  $y \cdot (y \cdot x' \rightarrow z) \cdot (y \cdot x' \rightarrow z') \rightarrow x$  (contradição)
- i)  $(x \rightarrow y) = (x \cdot y') \rightarrow (y \cdot y')$  (redução ao absurdo)

## 5. Verificar o resultado das seguintes relações

- a)  $x \cdot (y \cdot z) \Leftrightarrow (x \cdot y) \cdot z$  (associatividade)
- b)  $x + (y + z) \Leftrightarrow (x + y) + z$  (associatividade)
- c)  $x \cdot y \Leftrightarrow y \cdot x$  (comutatividade)
- d)  $x + y \Leftrightarrow y + x$  (comutatividade)
- e)  $x \cdot (y + z) \Leftrightarrow (x \cdot y) + (x \cdot z)$  (distributividade)
- f)  $x + (y \cdot z) \Leftrightarrow (x + y) \cdot (x + z)$  (distributividade)
- g)  $(x \cdot y)' \Leftrightarrow x' + y'$  (De Morgan)
- h)  $(x + y)' \Leftrightarrow x' \cdot y'$  (De Morgan)
- i)  $(x')' \Leftrightarrow x$  (involução)
- j)  $x \cdot x \Leftrightarrow x$  (idempotência)
- k)  $x + x \Leftrightarrow x$  (idempotência)
- l)  $x \rightarrow y \Leftrightarrow x' + y$  (implicação)
- m)  $(x \Leftrightarrow y) \Leftrightarrow (x \rightarrow y) \cdot (y \rightarrow x)$  (equivalência)
- n)  $x \rightarrow y \Leftrightarrow x' \rightarrow y'$  (contraposição)
- o)  $(x \cdot y) \rightarrow z \Leftrightarrow x \rightarrow (y \rightarrow z)$  (exportação)

## 6. Transformar as afirmativas abaixo em proposições :

- a) A soma de dois inteiros é maior que o primeiro.
- b) A soma de dois inteiros é maior que a diferença entre eles.
- c) O produto de um inteiro por zero é menor que outro inteiro.
- d) Se um de dois números são nulos, o produto deles é zero.
- e) Se dois números são iguais a um terceiro, então são iguais.

## 7. Construir um circuito para identificar se dois bits são iguais.

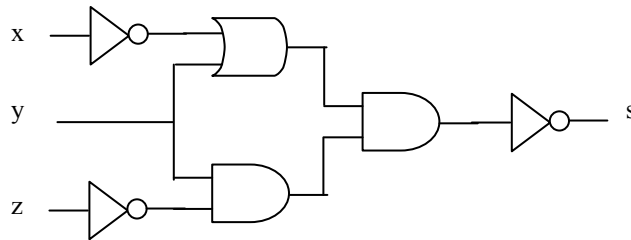
## 8. Construir um circuito capaz de fornecer a tabela abaixo :

x y z	f (x,y,z)
0 0 0	1
0 0 1	0
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	1

9. Simplificar a expressão lógica abaixo :

$$(x > 0 \text{ e } y = 0) \text{ ou não } (x < 0 \text{ ou } z > 0)$$

10. Identificar a equação do circuito abaixo pela soma de produtos (SoP):



11. Identificar a equação do circuito anterior pelo produto das somas (PoS).

12. Construir um circuito equivalente ao da questão (10) usando apenas portas NAND.

13. Construir um circuito equivalente ao da questão (10) usando apenas portas NOR.

14. Montar um diagrama de um somador completo de 2 palavras de 2 bits cada.

15. Construir um circuito equivalente ao da questão (14) usando apenas portas NAND.

16. Construir um circuito equivalente ao da questão (14) usando apenas portas NOR.

17. Simplificar por mapa de Karnaugh:  $a'b'c'd' + a'b'c'd + a'b'c'd' + a'b'c'd + a'b'c'd' + a'b'c'd + a'b'c'd' + a'b'c'd$

18. Demonstrar a validade do seguinte argumento:

(1)  $x < 6$

(2)  $y > 7 \text{ ou } x = y \rightarrow (y = 4 \text{ e } x < y)$

(3)  $y = 4 \rightarrow x < 6$

(4)  $x < 6 \rightarrow x < y$

$\therefore x = y$

19. Verificar se as proposições são falsas ou verdadeiras:

a)  $(\forall n \in \mathbb{N}) (n+4 > 3)$

b)  $(\forall n \in \mathbb{N}) (n+3 > 7)$

c)  $(\exists n \in \mathbb{N}) (n+4 < 7)$

d)  $(\exists n \in \mathbb{N}) (n+3 < 2)$

20. Sendo  $A = \{1, 2, 3, 4, 5\}$  determinar a negação de:

a)  $(\exists n \in A) (x+4 = 10)$

b)  $(\forall n \in \mathbb{N}) (x+4 < 10)$