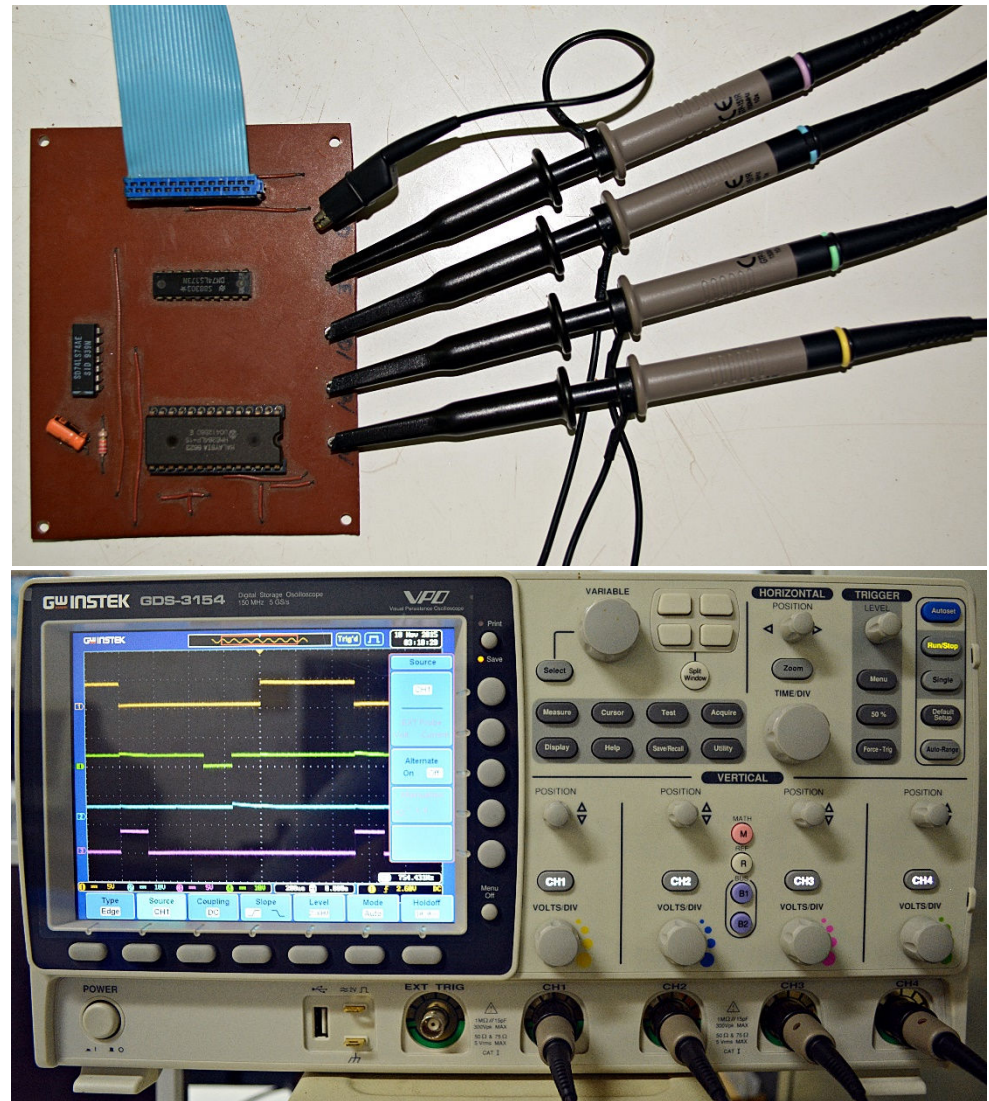


Barramento de Microprocessador Multiplexado no Tempo

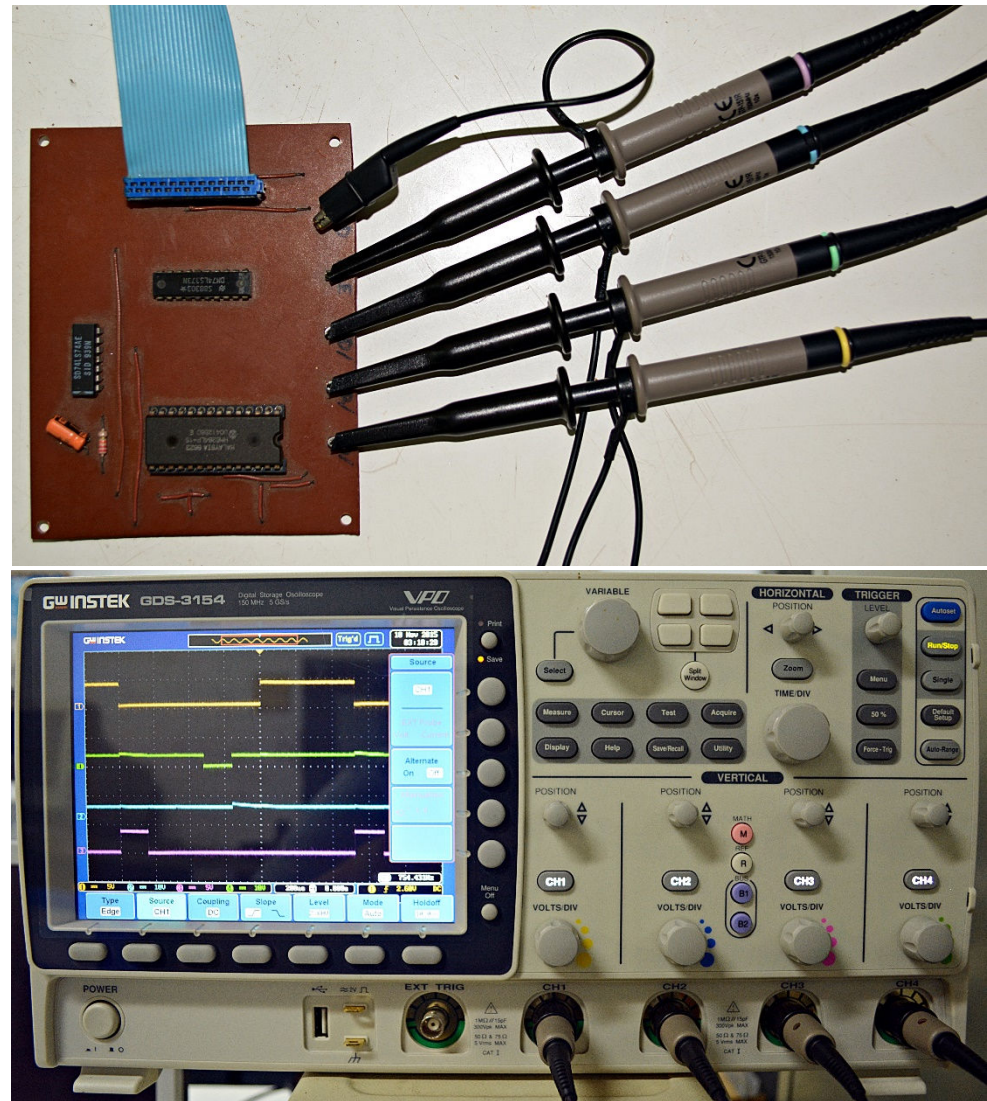
Dr. Jorge L. Aching Samatelo
jlasm001@gmail.com



❑ OBJETIVOS

➤ Estudar:

- ❖ o funcionamento de um barramento de microprocessador multiplexado no tempo (dados e endereços).
- ❖ os ciclos básicos de leitura e escrita em uma memória estática - SRAM.



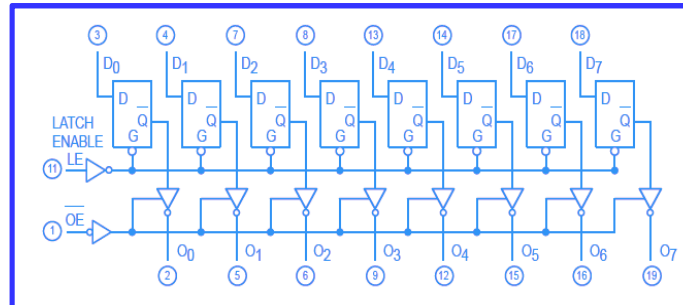
Índice

- ☐ Placa de Interface Paralela.
- ☐ Laboratório

Placa de Interface Paralela

Placa de Interface Paralela

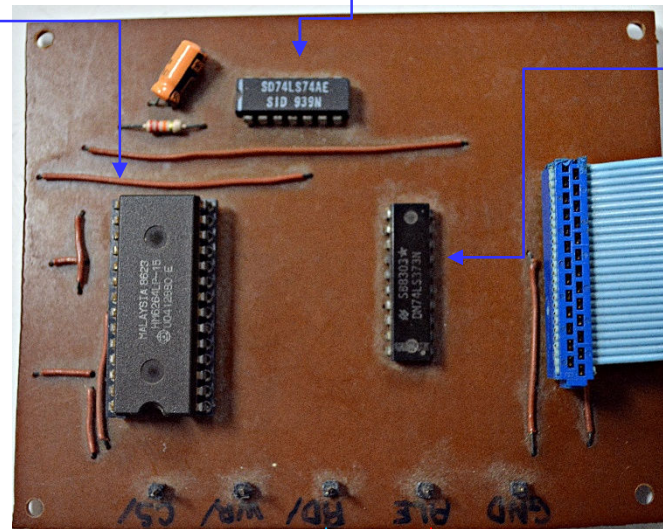
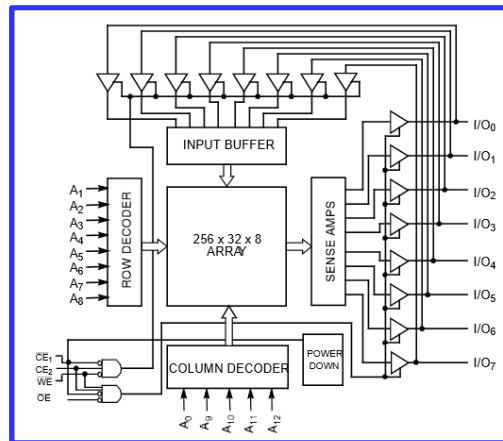
O **74LS373** faz a demultiplexação do barramento de endereços.



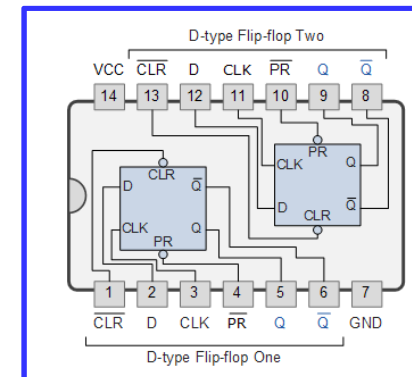
Flip-Flop Tri-State (**74LS373**)

O circuito **74LS373** garante que a memória não seja selecionada quando o computador é ligado e os níveis dos sinais de saída da interface paralela são aleatórios.

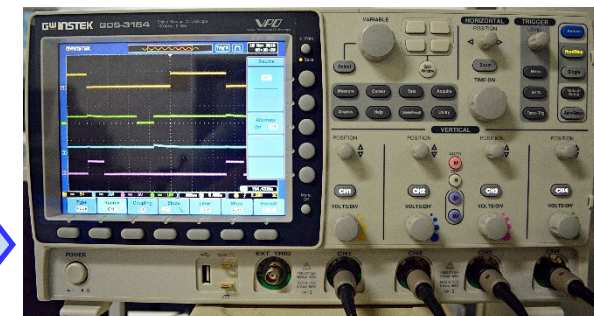
Memoria Estatica
(SRAM)8Kx8
(**6264**)



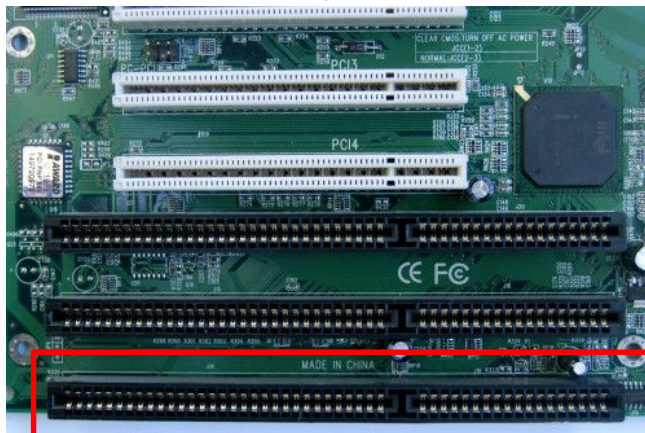
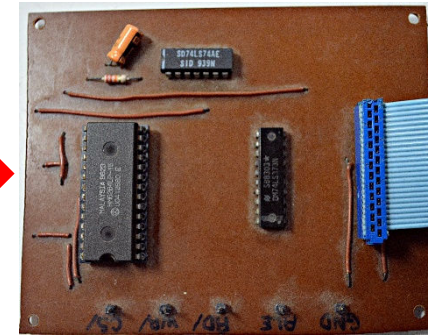
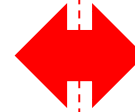
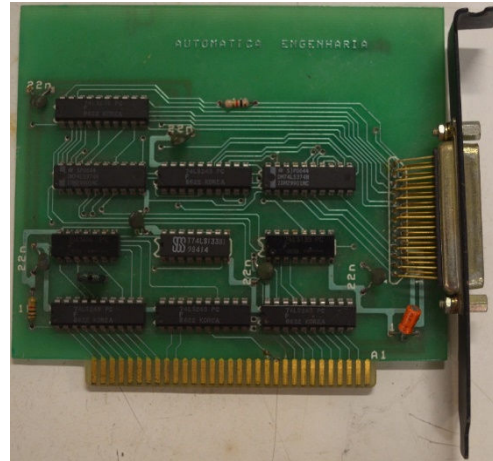
Flipflop Tipo D
(**74LS74**)



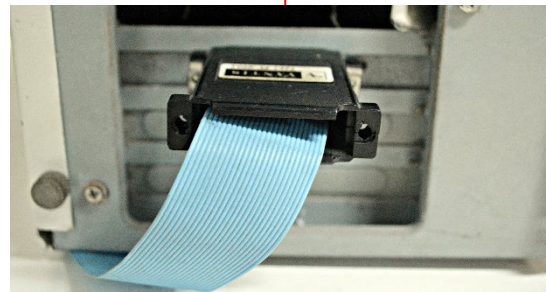
O **74LS373** tem: (a) 13 linhas para endereços e 8 para dados ($2^{13} \times 8 = 2^3 \times 2^{10} \times 8 = 8K \times 8$) (b) 2 linhas para habilitação (**CS1/** e **CS2/**) e uma linha para ordenar a operação de leitura (**RE/**) e outra para escrita (**WE/**).



Placa de Comunicação com a PC

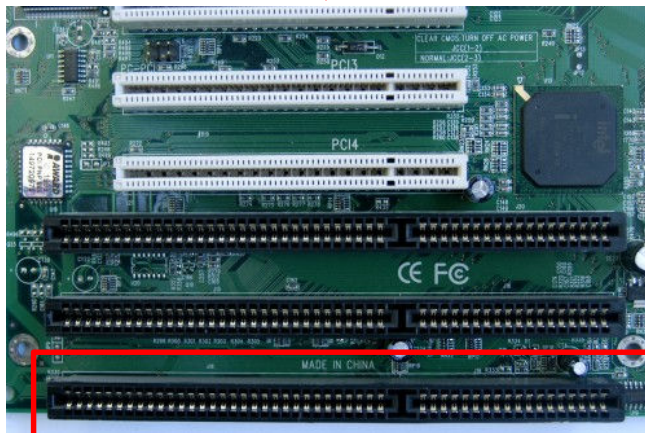
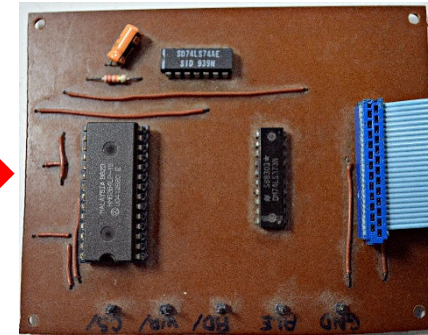
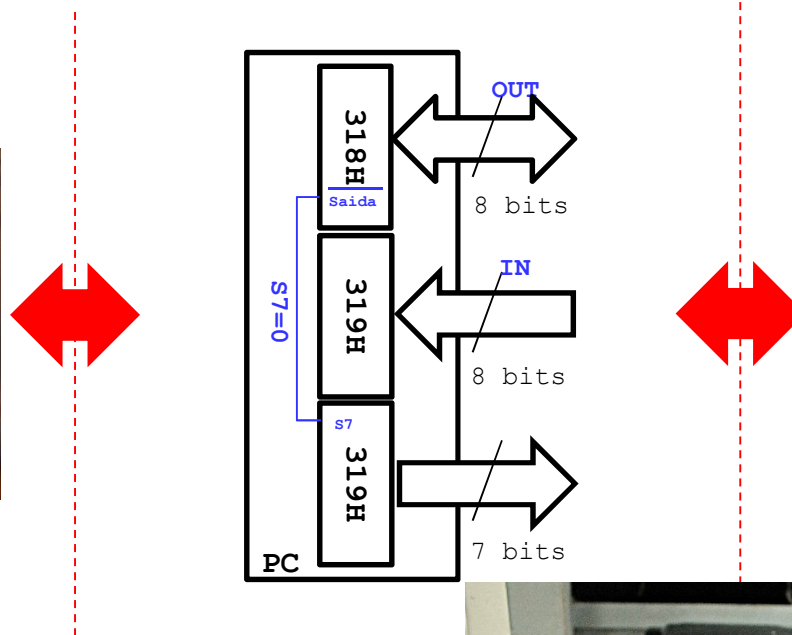


Barramento ISA de 16 bits

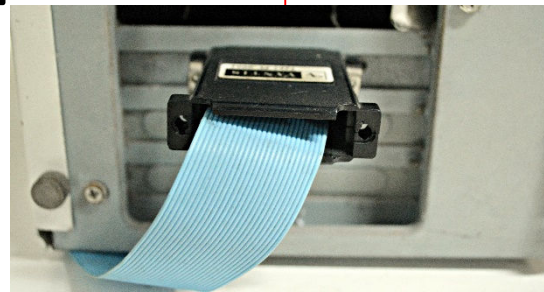


Cabo de 25 pinos

Placa de Comunicação com a PC

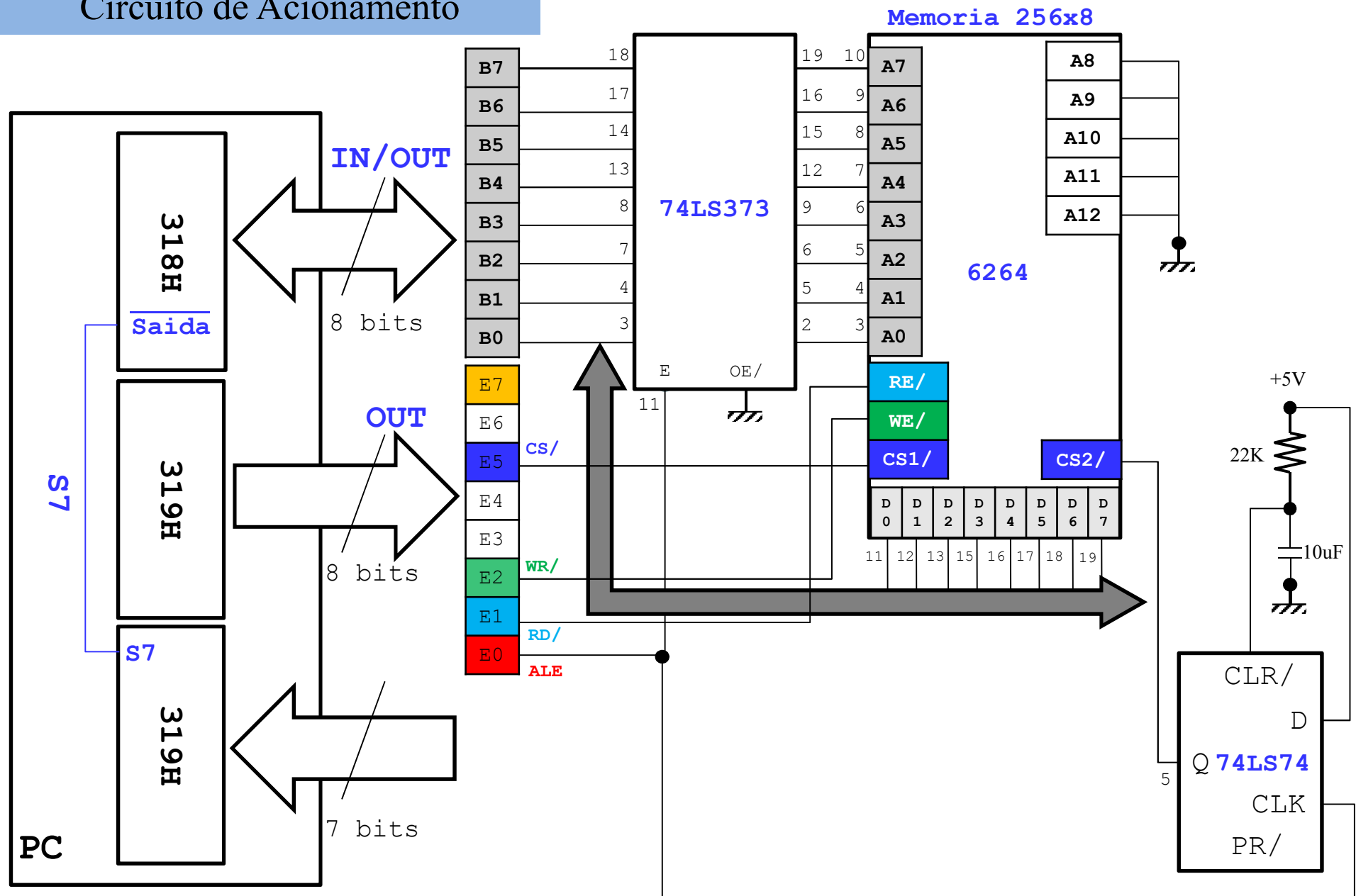


Barramento ISA de 16 bits



Cabo de 25 pinos

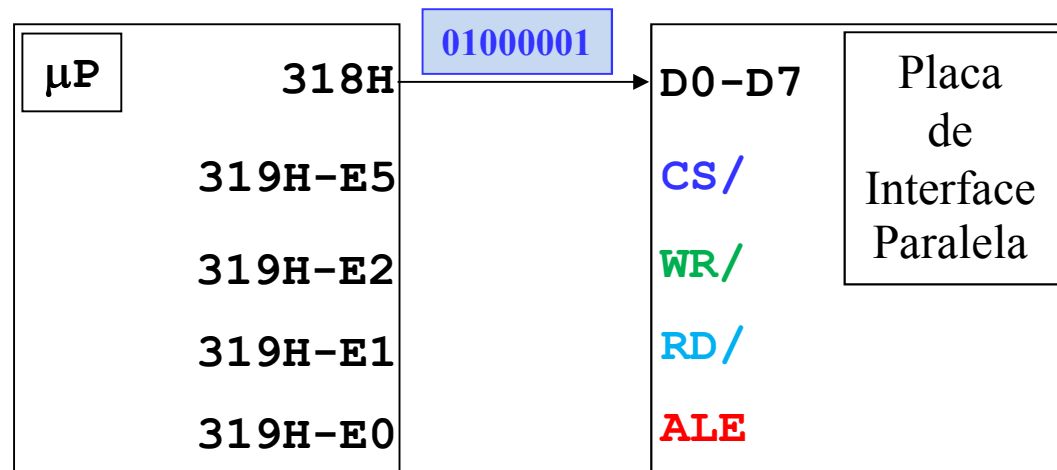
Circuito de Acionamento



Ciclos de Operação

Sinais de Barramento

- ❑ A figura abaixo mostra os sinais do barramento de 8 bits a ser implementado. O detalhamento da função de cada um deles é dado a seguir:

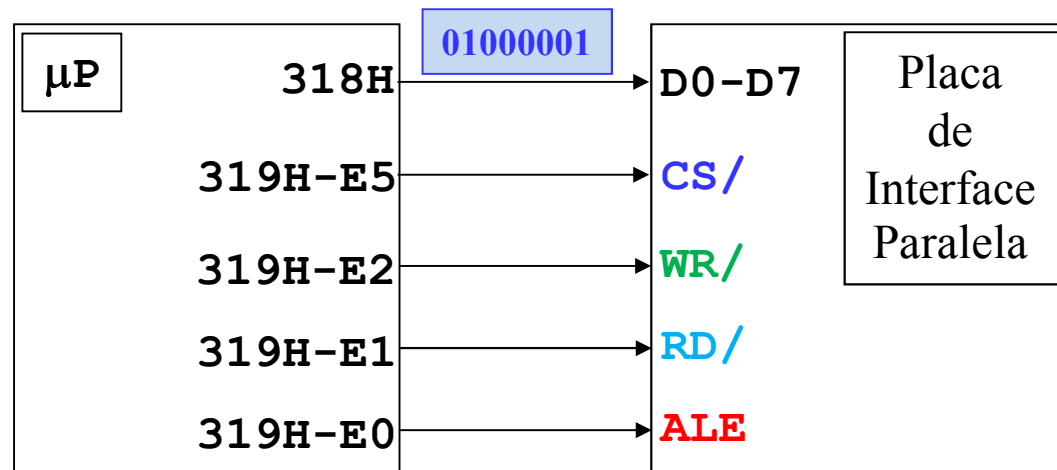


- ❑ **D0-D7 - Barramento de dados e endereços multiplexado** (endereçamento máximo da memória é de 256 posições)
 - Com este barramento de 8 bits multiplexado, serão implementados a demultiplexação do barramento e os ciclos para leitura e escrita de uma memória 6264 (RAM).

Ciclos de Operação

Sinais de Barramento

- ❑ A figura abaixo mostra os sinais do barramento de 8 bits a ser implementado. O detalhamento da função de cada um deles é dado a seguir:



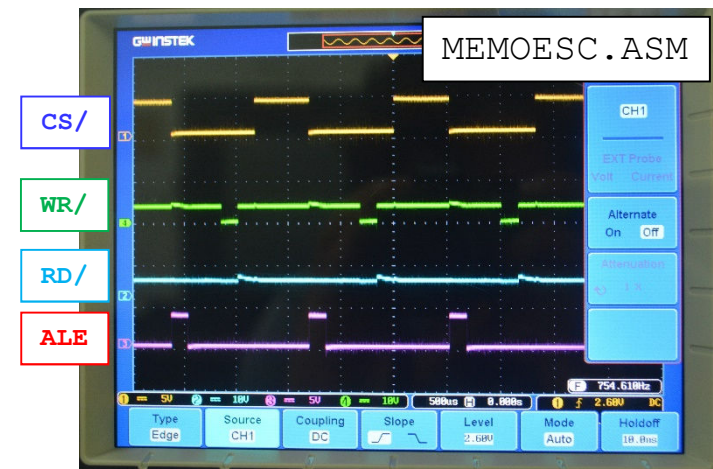
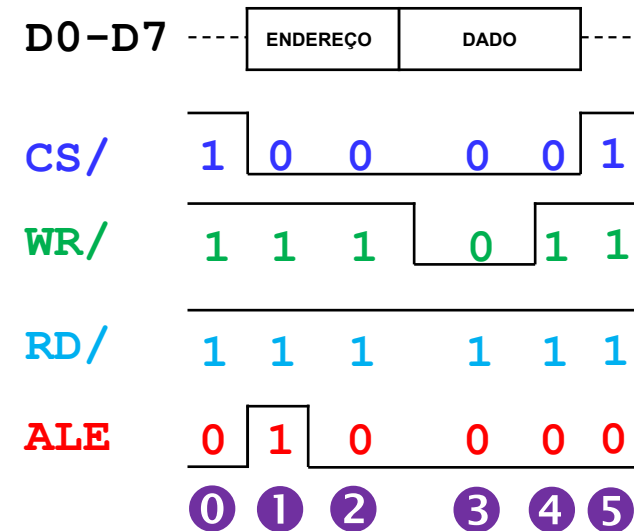
- ❑ **ALE** - *Address latch enable* - Transição negativa do ALE indica **endereços estáveis** no barramento AD 0-7.
- ❑ **RD**- *Read* - Sinal normalmente em nível 1. Quando em nível 0 indica que um dado da memória deve ser colocado no barramento AD 0-7.
- ❑ **WR**- *Write*- Sinal normalmente em 1. Quando em nível 0 indica que o dado do barramento AD 0-7 deve ser escrito na memória.
- ❑ **CS** - *Chip-select* - Quando em 0 deve habilitar a memória para leitura ou escrita. Quando em 1 a memória deve estar desabilitada.

Ciclos de Operação

Ciclo de Escrita

- Temporização das sinais para os ciclo de escrita na memória.

- 0 Inicializa o ciclo de Escrita:
S7-CS/-WR/-RD/-ALE \leftarrow 11110
- Envia endereço
- 1 e 2 Envia um flanco negativo no ALE
- Envia o dado
- 3 Ativa a escrita de dados: WR/ \leftarrow 0
- 4 Desativa a escrita de dados : WR/ \leftarrow 1



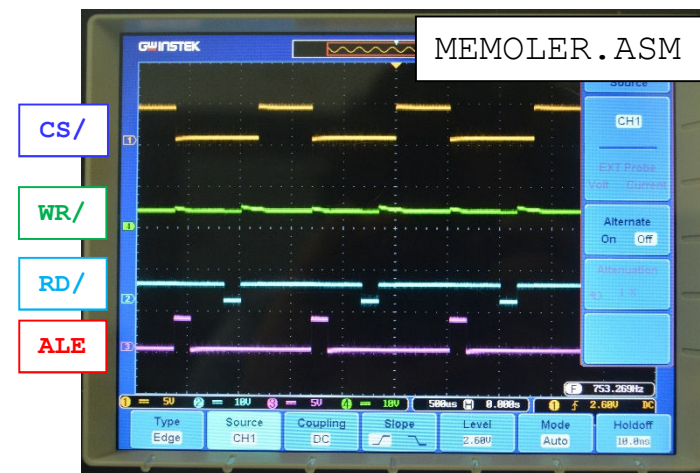
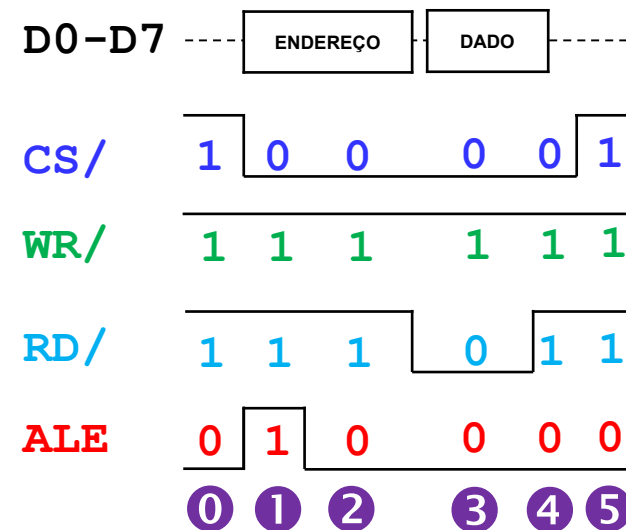
Ciclos de Operação

Ciclo de Leitura

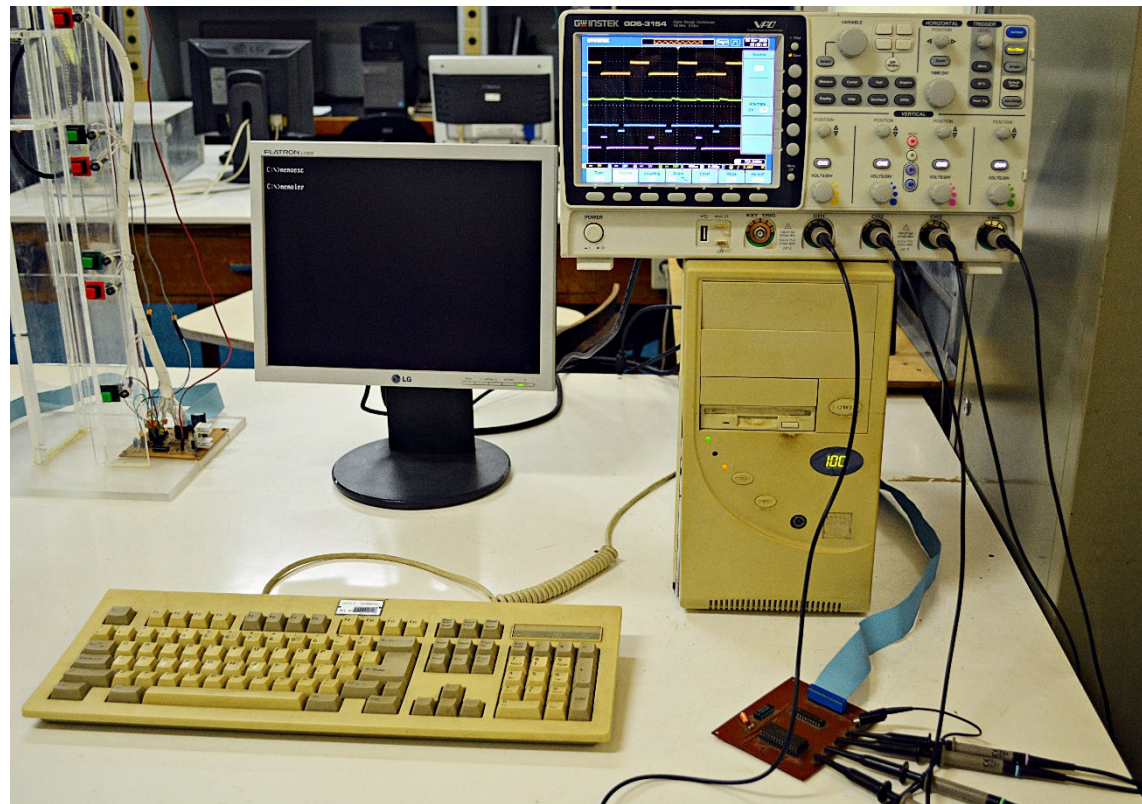
- ❑ Temporização das sinais para os ciclo de leitura na memória.

- 0 Inicializa o ciclo de Leitura:
 $S7\text{-CS/-WR/-RD/-ALE} \leftarrow 11110$
Envia endereço
- 1 e 2 Envia um flanco negativo no **ALE**
Configuramos a porta **318H** como entrada
- 3 Um dado da memoria será colocado no barramento: $RD/ \leftarrow 0$
Envia o dado
- 4 No barramento não tem dado da memoria:
 $RD/ \leftarrow 1$

Lembre-se que a direção do barramento AD 0-7 é dada pelo pino 7 da porta de saída 319H. **QUANDO ELE É 0 O BARRAMENTO É SAÍDA, QUANDO 1 O BARRAMENTO É ENTRADA.** Só mude a direção do barramento para saída quando necessário, voltando para entrada assim que possível.



Laboratório



Laboratório

Atenção

❑ Lembrar que:

- Comando para montar a pasta de trabalho no DOSBox (c:\sistemb1\frsm)

```
mount c c:\sistemb1\frsm\ ↵  
c: ↵
```

- Comando para gerar o arquivo .OBJ: `nasm nome_arquivo ↵`

- Comando para gerar o arquivo .EXE: `freelink nome_arquivo ↵`

- Depurar o arquivo .EXE: `debug nome_arquivo.exe ↵`

❖ Para depurar linha por linha é o comando `t`.

- Executar o .EXE: `nome_arquivo.exe ↵`

Laboratório

Atenção

❑ Com relação ao uso dos disquetes:

- O comando DOS para copiar um arquivo do disquete ao disco C da PC onde está o módulo é:

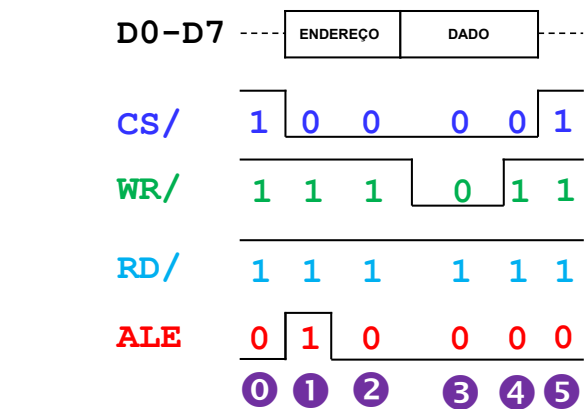
```
C:\>copy a:nome_arquivo.asm
```

- Quando copiar um arquivo da PC onde está trabalhando para o disquete, é necessário desmarcar a opção "arquivo morto" do arquivo em questão, senão o PC onde está o módulo não consegue ler.
 - ❖ Sobre o arquivo, pressionar botão direito do mouse, ir em propriedades e desmarcar a opção "arquivo morto".

Laboratório

1.

- ☐ Estudar os programas `MEMOESC.ASM` e `MEMOLER.ASM`.
-



0 Inicializa o ciclo de Escrita:
S7-CS/-WR/-RD/-ALE ← 1**1110**

```
mov al, 10100110b
mov dx, 319h
out dx, al
```

Envia endereço

```
mov al, [endereco]
mov dx, 318h
out dx, al
```

1 Envia um flanco negativo no **ALE**

e

2

```
mov al, 00000111b
mov dx, 319h
out dx, al
call delay
mov al, 00000110b
out dx, al
```

Envia o dado

```
mov al, [dadomen]
mov dx, 318h
out dx, al
```

3 Ativa a escritura de dados: **WR/** ← 0

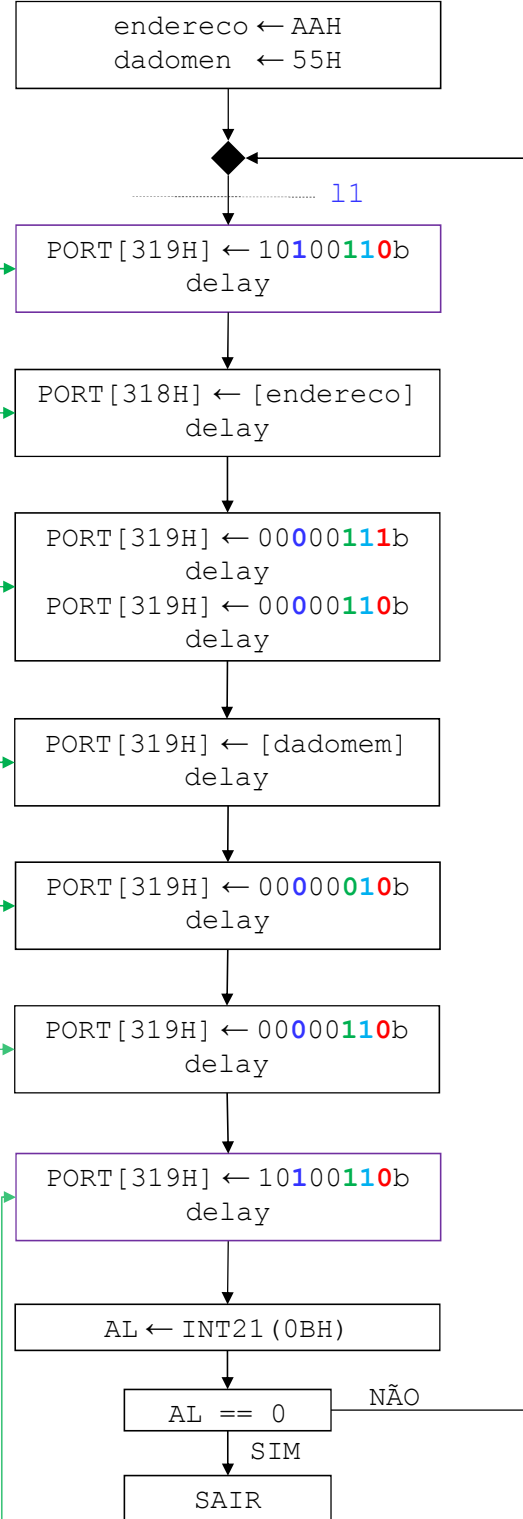
```
mov al, 00000010b
mov dx, 319h
out dx, al
```

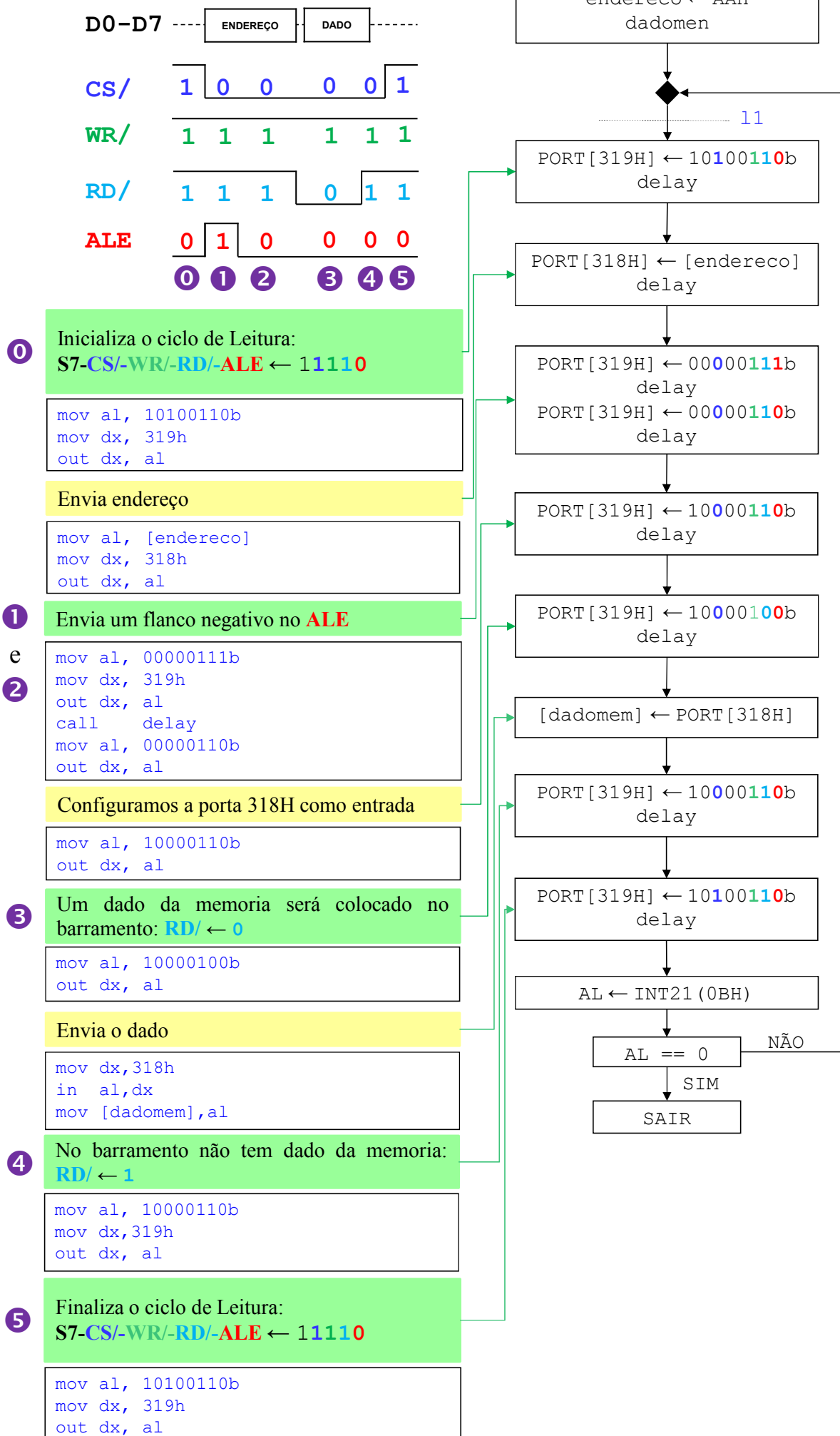
4 Desativa a escritura de dados : **WR/** ← 1

```
mov al, 00000110b
out dx, al
```

5 Finaliza o ciclo de Escrita:
S7-CS/-WR/-RD/-ALE ← 1**1110**

```
mov al, 10100110b
mov dx, 319h
out dx, al
```





Laboratório

2.

- ❑ Tomando em conta os programas `MEMOESC.ASM` e `MEMOLER.ASM`, implementar:
 - a) Um programa que escreva uma sequência de 16 bytes na memória externa nos endereços 0 a 15.
 - b) Um programa que leia uma sequência de 16 bytes da memória externa (endereços 0 a 15) e imprima o resultado na tela.
-

❑ Dica:

- Os executáveis dos programas a implementar são: `MEMOEM.EXE` e `MEMOLM.EXE`.

- ❖ Primeiro, implemente o programa de escrita de mensagem, e teste com o programa `MEMOLM.EXE`.

```
>>MeuProgramaEM↵  
>>MEMOLM↵
```

- ❖ Segundo, implemente o programa de leitura de mensagem, e teste com o programa `MEMOEM.EXE`.

```
>>MEMOEM↵  
>>MeuProgramaLM↵
```

Anexos

Anexos

- ☐ Definição de Variáveis.
- ☐ Modos de Endereçamento de Dados.
- ☐ Comparação de Dados.
- ☐ Saltos Condicionais.
- ☐ Instruções para acionar portas de E/S.

Definição de Variáveis

Definição de arrays

□ Arrays

- Conjunto de BYTES ou WORDS em sequência na memória.

□ Estrutura

<Nome> <Pseudo-instrução> <valor>

□ Exemplo: *Arrays* numéricos

BARRAY	DB	0x10, 0x20, 0x30, 0x01
WARRAY	DW	0x1000, 0x123, 0x0, 0xFFFF

	DS:000B	FF
	DS:000A	FF
	DS:0009	00
	DS:0008	00
	DS:0007	01
	DS:0006	23
	DS:0005	10
WARRAY →	DS:0004	00
	DS:0003	01
	DS:0002	30
	DS:0001	20
BARRAY →	DS:0000	10

Definição de Variáveis

Definição de arrays

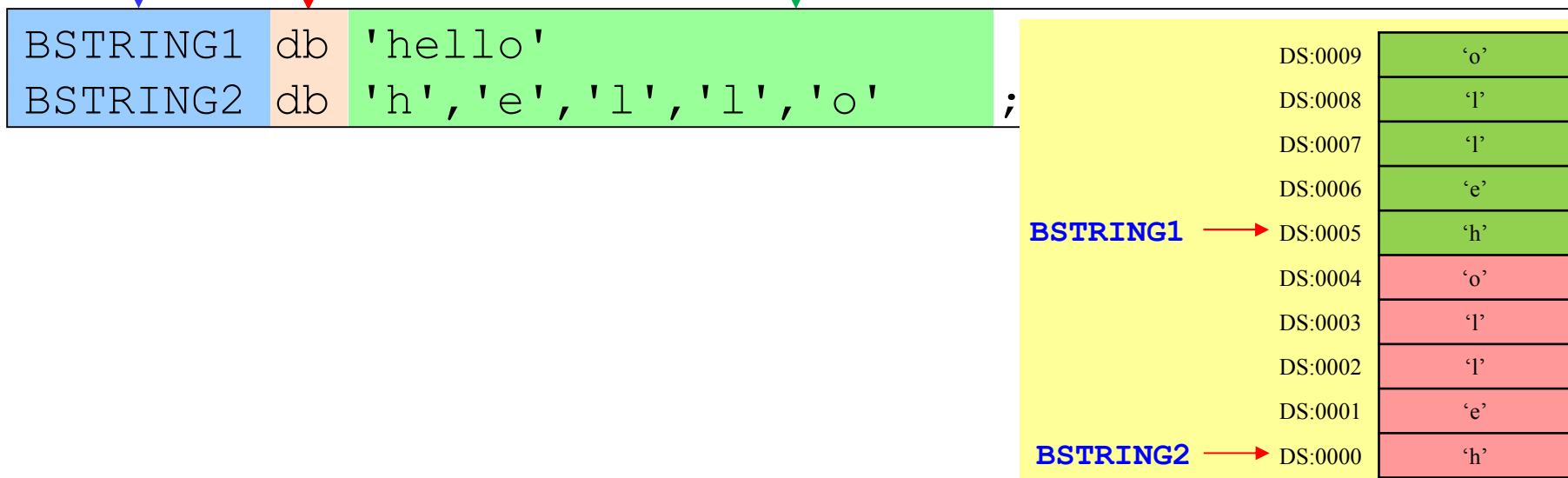
□ Arrays

- Conjunto de BYTES ou WORDS em sequência na memória.

□ Estrutura

<Nome> <Pseudo-instrução> <valor>

□ Exemplo: *Arrays* de caracteres (*string*)



Definição de Variáveis

Definição de arrays

□ Arrays

- Conjunto de BYTES ou WORDS em sequência na memória.

□ Estrutura

<Nome> <Pseudo-instrução> <valor>

□ Exemplo: *Arrays* de caracteres (*string*)

```
STR1 DB "Hola mundo", "$"  
STR2 DB 'Oi', '$'
```

- Quando uma *string* é impressa usando a interrupção `int 0x21`, o `$` indica o final de linha.

```
segment code  
mov dx, STR1  
mov ah, 09  
int 21h
```

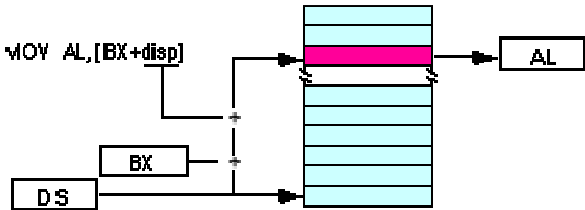
	DS:000D	'\$'
	DS:000C	'i'
	DS:000B	'O'
STR2 →	DS:000A	'\$'
	DS:0009	'o'
	DS:0008	'd'
	DS:0007	'n'
	DS:0006	'u'
	DS:0005	'm'
	DS:0004	
	DS:0003	'a'
	DS:0002	'l'
	DS:0001	'o'
STR1 →	DS:0000	'h'

Modos de Endereçamento de Dados

Endereçamento Indexado (*Indexed Addressing*)

- ❑ Os deslocamentos gerados por estes modos de endereçamento são a soma da constante e do registrador especificado. Como padrão, os modos de endereçamento que envolvem BX, SI e DI usam o segmento de dados e o modo de endereçamento disp[BP] usa o segmento da pilha.

Segmento usado por defeito	Definição do segmento a usar
MOV AL, disp[BX] ; (DS)	MOV AL, SS:disp[BX] ; (SS)
MOV AL, disp[BP] ; (SS)	MOV AL, ES:disp[BP] ; (ES)
MOV AL, disp[SI] ; (DS)	MOV AL, CS:disp[SI] ; (CS)
MOV AL, disp[DI] ; (DS)	MOV AL, SS:disp[DI] ; (SS)

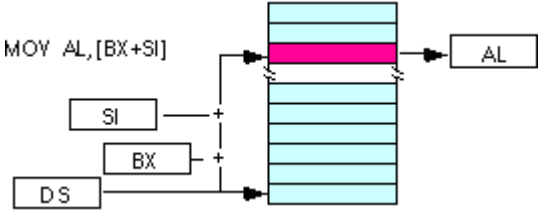
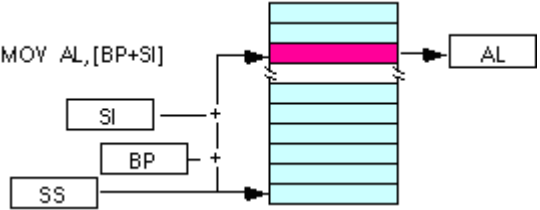


The diagram illustrates the Indexed Addressing mode. It shows a memory stack with a highlighted pink cell. The address of this cell is calculated as the sum of the displacement (disp) and the contents of the BX register. The DS register points to the base of the stack. The result of the addition is used to access the memory cell, which then provides the value to the AL register.

Modos de Endereçamento de Dados

Endereçamento Indexado mais base (*Based Indexed*)

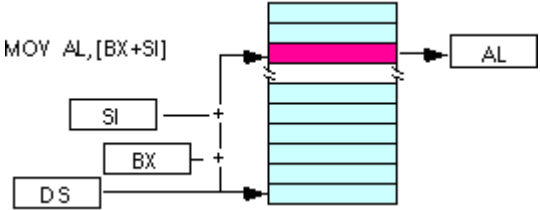
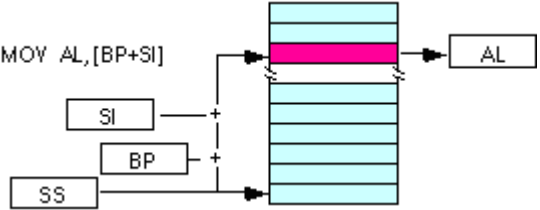
- ❑ Este modos formam o deslocamento adicionando o registrador base (BX ou BP) ao registrador índice (SI ou DI).

Segmento usado por defeito	
MOV AL, [BX][SI]	
MOV AL, [BX][DI]	
MOV AL, [BP][SI]	
MOV AL, [BP][DI]	
 <p>Diagram illustrating the instruction MOV AL, [BX+SI] using the DS segment. The SI register and the BX register are added together to form an effective address. This address is then used to access a memory location within the DS segment, which is represented by a stack of memory cells. The data from this location is loaded into the AL register.</p>	 <p>Diagram illustrating the instruction MOV AL, [BP+SI] using the SS segment. The SI register and the BP register are added together to form an effective address. This address is then used to access a memory location within the SS segment, which is represented by a stack of memory cells. The data from this location is loaded into the AL register.</p>

Modos de Endereçamento de Dados

Endereçamento Indexado mais base mais deslocamento

- ❑ Este modos formam o deslocamento adicionando o registrador base (BX ou BP) ao registrador índice (SI ou DI) e um *offset* `disp`.

Segmento usado por defeito	
MOV AL, <code>disp[BX][SI]</code>	
MOV AL, <code>disp[BX+DI]</code>	
MOV AL, <code>[BP+SI+disp]</code>	
MOV AL, <code>[BP][DI][disp]</code>	
 <p>Diagram illustrating the address calculation for the instruction <code>MOV AL, [BX+SI]</code> using the Data Segment (DS). The base register BX and the index register SI are added together to form the effective address. This address is then used to access a memory location within the DS segment, which is represented by a stack of memory cells. The data from the selected memory cell is loaded into the AL register.</p>	 <p>Diagram illustrating the address calculation for the instruction <code>MOV AL, [BP+SI]</code> using the Stack Segment (SS). The base register BP and the index register SI are added together to form the effective address. This address is then used to access a memory location within the SS segment, which is represented by a stack of memory cells. The data from the selected memory cell is loaded into the AL register.</p>

Comparação de Dados

Instrução **CMP**

❑ Que faz:

- Efetua a **comparação** de inteiros e caracteres.
- A comparação é feita através da subtração entre os dois operandos, porém o resultado não é armazenado em nenhum registrador.

❑ Sintaxe:

`CMP <destino>, <fonte>`

`Resultado = destino - fonte`

- Resultados modificam os bits do registrador `FLAGS`.
 - ❖ *Overflow* (OF),
 - ❖ *Sign* (SF),
 - ❖ *Zero* (ZF),
 - ❖ *Carry* (CF)

Comparação de Dados

Instrução **CMP**

❑ Que faz:

- Efetua a **comparação** de inteiros e caracteres.
- A comparação é feita através da subtração entre os dois operandos, porém o resultado não é armazenado em nenhum registrador.

❑ Sintaxe:

`CMP <destino>, <fonte>`

`Resultado = destino - fonte`

- A mudança nos *flags* variam se estamos tratando de números com ou sem sinal

❖ Comparação de **NÚMEROS SEM SINAL (0,1,2,...)**:

Resultados do CMP	ZF	CF
<code>destino < fonte</code>	0	1
<code>destino > fonte</code>	0	0
<code>destino = fonte</code>	1	0

Comparação de Dados

Instrução **CMP**

❑ Que faz:

- Efetua a **comparação** de inteiros e caracteres.
- A comparação é feita através da subtração entre os dois operandos, porém o resultado não é armazenado em nenhum registrador.

❑ Sintaxe:

`CMP <destino>, <fonte>`

`Resultado = destino - fonte`

- A mudança nos *flags* variam se estamos tratando de números com ou sem sinal

❖ Comparação de **NÚMEROS COM SINAL (...,-2, -1, 0, 1, 2, ...)**:

Resultados do CMP	<i>flags</i>
<code>destino < fonte</code>	<code>SF ≠ OF</code>
<code>destino > fonte</code>	<code>SF = OF</code>
<code>destino = fonte</code>	<code>ZF = 1</code>

Altera os *flag*:

- ❑ *Sign* (**SF**)
- ❑ *Overflow* (**OF**)
- ❑ *Zero* (**ZF**)

Comparação de Dados

Instrução **CMP**

❑ Que faz:

- Efetua a **comparação** de inteiros e caracteres.
- A comparação é feita através da subtração entre os dois operandos, porém o resultado não é armazenado em nenhum registrador.

❑ Sintaxe:

CMP <destino>, <fonte>

❑ Exemplo:

MOV	AX,	30	
CMP	AX,	-40	; Destino (30) > fonte (-40) ⇒ SF == OF

Comparação de Dados

Instrução **CMP**

❑ Que faz:

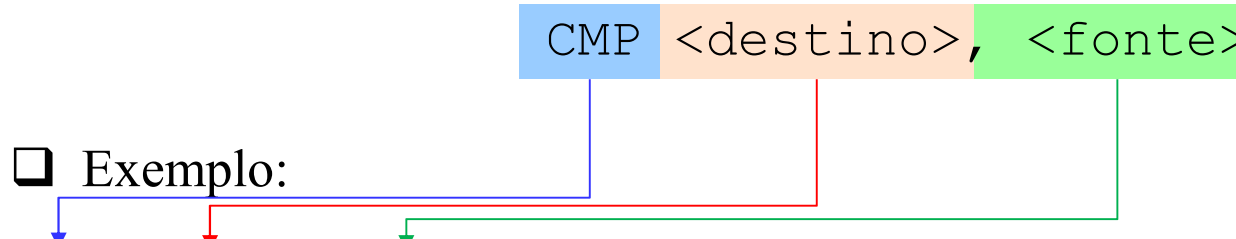
- Efetua a **comparação** de inteiros e caracteres.
- A comparação é feita através da subtração entre os dois operandos, porém o resultado não é armazenado em nenhum registrador.

❑ Sintaxe:

CMP <destino>, <fonte>

❑ Exemplo:

MOV	BX,	-30	
CMP	BX,	10	; SF != OF



Saltos Condicionais

Saltos de *Flag* Simples

- ❑ Considerando que são usadas depois de uma **instrução de comparação** (ex. **CMP**) ou uma **expressão aritmética** (ex. **INC**, **DEC**).

<i>Símbolos</i>	<i>Descrição</i>	<i>Condições</i>
JE ou JZ	<i>Jump if Equal ou Jump if Zero</i>	ZF = 1
JNE ou JNZ	<i>Jump if Not Equal ou Jump if Not Zero</i>	ZF = 0
JC	<i>Jump if Carry</i>	CF = 1
JNC	<i>Jump if Not Carry</i>	CF = 0
JO	<i>Jump if Overflow</i>	OF = 1
JNO	<i>Jump if Not Overflow</i>	OF = 0
JS	<i>Jump if Sign</i>	SF = 1
JNS	<i>Jump if Not Sign</i>	SF = 0
JP ou JPE	<i>Jump if Parity ou</i>	PF = 1
JNP ou JPO	<i>Jump if Not Parity ou</i>	PF = 0

Instruções para acionar portas de E/S

Instrução **IN** e **OUT**

- ❑ Em termos gerais, a comunicação do 8086/8088 com periféricos ocorre por meio de duas instruções:
 - A instrução **IN**, a qual realiza a **LEITURA DE DADOS**;
 - A instrução **OUT**, responsável pelo **ESCRITURA DE DADOS**.

Instruções para acionar portas de E/S

Instrução **IN** (*read port*)

❑ Que faz:

- copia um BYTE ou WORD de uma porta especificada para o acumulador.

❑ Sintaxe:

IN <destino>, <endereço>

Onde:

❖ <destino>: registrador de destino do dado (AL ou AX).

❖ <endereço>: endereço da PORTA DE ENTRADA do dado (uma constante ou DX).

- ❑ Utilizando-se o registrador AL como destino, apenas um BYTE será LIDO do endereço fornecido.

- Caso se utilize IN AL, <endereço>, o endereço poderá variar de 00H até FFH.
- Caso se utilize IN AL, DX, os endereços poderão variar de 0000H até FFFFH.

- ❑ Utilizando-se o registrador AX como destino,

- o valor lido no endereço fornecido será carregado em AL e
- o valor lido no endereço subsequente será carregado em AH.

Instruções para acionar portas de E/S

Instrução **IN** (*read port*)

❑ Que faz:

- copia um BYTE ou WORD de uma porta especificada para o acumulador.

❑ Sintaxe:

```
IN <destino>, <endereço>
```

❑ Exemplo:

```
MOV DX, 0x318 ; AL <= 318H  
IN AL, DX ; AL <= PORT[0318H]
```


Instruções para acionar portas de E/S

Instrução **OUT** (*write port*)

❑ Que faz:

- copia um BYTE ou WORD do acumulador para uma porta especificada.

❑ Sintaxe:

OUT <endereço>, <origem>

Onde:

❖ <origem>: registrador de origem do dado (AL ou AX).

❖ <endereço>: endereço da PORTA DE SAÍDA do dado (uma constante ou DX).

- ❑ Utilizando-se o registrador AL como origem do dado, apenas um BYTE será ESCRITO no endereço fornecido.

- Caso se utilize OUT <endereço>, AL, o valor de AL será transferido para o endereço fornecido (que pode variar de 00H até FFH).
- Caso se utilize OUT AL, DX, o valor de AL será transferido para o endereço fornecido por DX (que pode variar de 0000H até FFFFH).

- ❑ Utilizando-se o registrador AX como origem do dado,

- o valor de AL será transferido para o endereço fornecido.
- o valor de AH será transferido para o endereço fornecido acrescido de uma₄₄

Instruções para acionar portas de E/S

Instrução **OUT** (*write port*)

❑ Que faz:

- copia um BYTE ou WORD do acumulador para uma porta especificada.

❑ Sintaxe:

OUT <endereço>, <origem>

❑ Exemplo:

```
MOV DX, 0x319    ; DX <= 319H
MOV AL, 0x0       ; AL <= 0H
OUT DX, AL        ; PORT[0319H] <= AL
```