Aquisição e envio de informação

Hardware

Porta paralela

Porta Série

Portas USB

Placas de aquisição de dados

- **Apenas digitais**
- **Apenas Analógicas**
- **Digitais / Analógicas**

Afinal os computadores também "falam" com o meio exterior

Acesso directo aos registos físicos

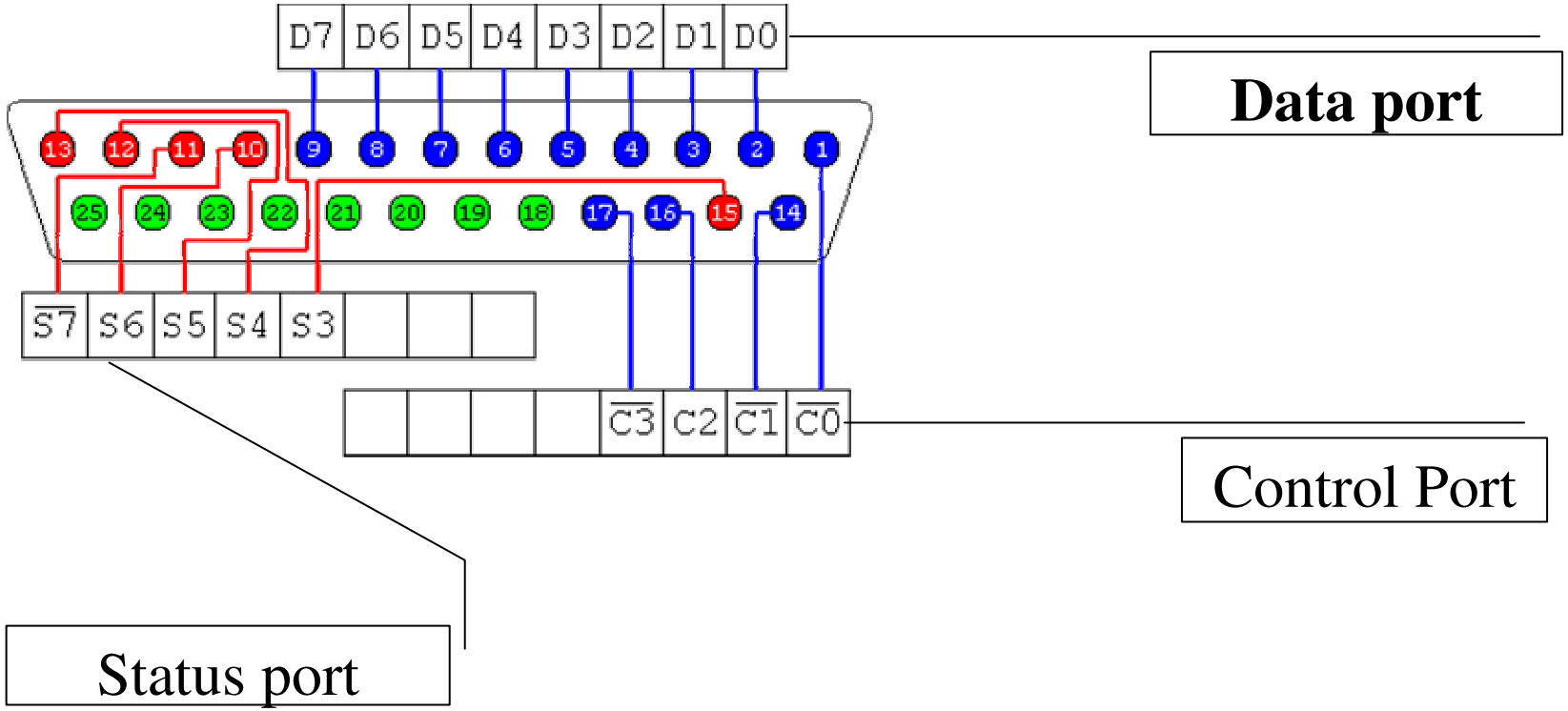
Porta Paralela

A porta paralela original (IBM-PC's) tem:

3 portos de 8 bits cada:

Data port
Status port
Control port

Identificação dos portos e dos bits na porta paralela



Importante perceber que existem alguns bits invertidos

- **As portas paralelas têm evoluído**
- **Versões mais avançadas têm sido introduzidas ao longo dos anos:**

  - » **Bi-direccional (PS/2)**
  - » **Enhanced Parallel Port (EPP)**
  - » **Extended Capability Port (ECP)**

- **A porta original é vulgarmente designada por Standard Parallel Port (SPP)**

Endereços normais dos portos da porta paralela

	Endereço LPT1	**Endereço LPT2**
Registo dados	378h	278h
Registo de Estado	379h	279h
Registo de controlo	37Ah	27Ah

Note-se que estes endereços podem ser diferentes, em especial em máquinas mais antigas

Lembre-se que uma Porta paralela não é um equipamento de potência, e que portanto funciona com níveis de tensão reduzidos e de corrente ainda mais reduzidos...

Soluções???

Software

• Tipicamente, quando pretendemos interagir directamente com o HW, utiliza-se SW codificado em assembly (não é estritamente necessário…)

• Em Ada, podemos incluir código assembly directamente nos nossos programas (*inline assembler*), utilizando as facilidades do pacote *System.Machine_Code*

• Existem algumas diferenças entre o assembly da Intel e aquele que é utilizado pelo GNAT

IO_Ports.ads

```
with Interfaces;

package IO_Ports is

   -- Interrupt control
   procedure Enable_Interrupts;
   procedure Disable_Interrupts;

   -- Writing IO ports
   procedure Write_IO_Port (Address : in Interfaces.Unsigned_16;
                            Value   : in Interfaces.Unsigned_8);
   procedure Write_IO_Port (Address : in Interfaces.Unsigned_16;
                            Value   : in Interfaces.Unsigned_16);
   procedure Write_IO_Port (Address : in Interfaces.Unsigned_16;
                            Value   : in Interfaces.Unsigned_32);

   -- Reading IO ports
   procedure Read_IO_Port (Address : in     Interfaces.Unsigned_16;
                           Value   :    out Interfaces.Unsigned_8);
   procedure Read_IO_Port (Address : in     Interfaces.Unsigned_16;
                           Value   :    out Interfaces.Unsigned_16);
   procedure Read_IO_Port (Address : in     Interfaces.Unsigned_16;
                           Value   :    out Interfaces.Unsigned_32);

 private

   pragma Inline (Read_IO_Port);
   pragma Inline (Write_IO_Port);
   pragma Inline (Enable_Interrupts);
   pragma Inline (Disable_Interrupts);

end IO_Ports;
```

Pragma???

→ Instrução ao compilador para efectuar uma determinada acção em tempo de compilação

Pragma Inline???

→ Inline é um pragma que indica ao complidaor que o código correspondente aos subprogramas passados como argumento deve ser expandido em cada chamada.

Vantagens???

→Existem já que o inline (especialmente para porções de código pequenos) pode diminuir o "overhead" provocado pelo chamamento a subprogramas.

→ Quando o "overhead" é uma parte importante do tempo de execução do subprograma, a utilização do pragma inline pode reduzir significativamente o tempo de execução

procedure Asm (Template : String; Outputs : Asm_Output_Operand_List;

Inputs : Asm_Input_Operand_List; Clobber : String := "";
Volatile : Boolean := False);

Informar o compilador sobre os registos que o código assembly está a utilizar → evitar "confusões" entre o assembly e a linguagem compilada

O compilador tenta fazer optimizações o que na presença de assembly inline pode, por vezes, levar a efeitos indesejados → para evitar estas optimizações → Volatile => true

IO_Ports.adb

```
with System.Machine_Code; use System.Machine_Code;

package body IO_Ports is

   use Interfaces;

-------------------------
INTERRUPT CONTROL
-------------------------

   procedure Enable_Interrupts is
   begin
      Asm ("STI");
   end Enable_Interrupts;

   procedure Disable_Interrupts is
   begin
      Asm ("CLI");
   end Disable_Interrupts;
```

Pacote que, entre outros aspectos contém os tipos "sem sinal" (unsigned)

Asm é um procedimento definido do pacote System.Machine_Code. Pode ser interpretado como um procedimento que, em tempo de compilação converte as "strings" fornecidas como parâmetros em código máquina executável

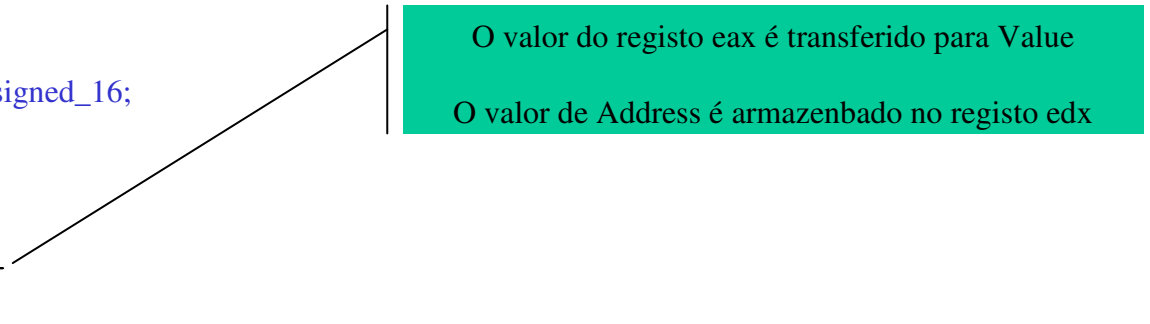
"Instruction Set" 80x86

STI (Set Interrupt Flag) → Habilita os interrupts

CLI (Clear Interrupt Flag) → Inibe os interrupts

READING IO PORTS

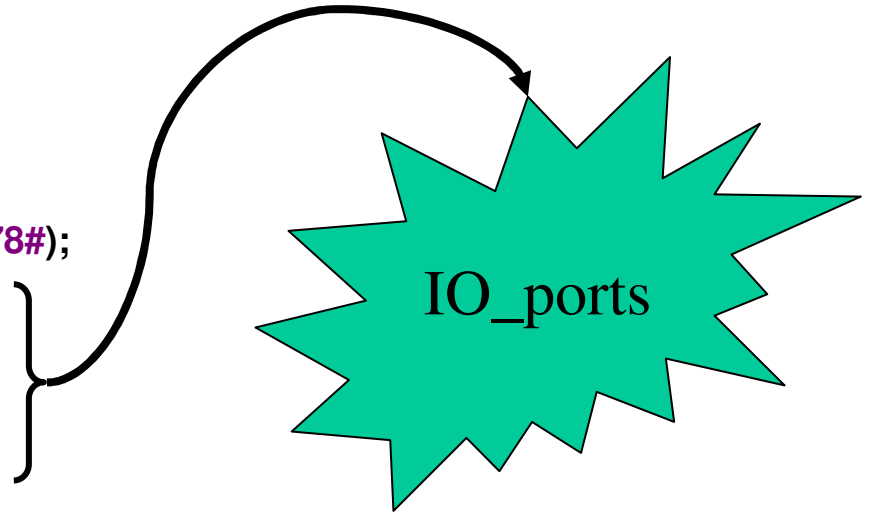
```
procedure Read_IO_Port (Address : in     Unsigned_16;
                        Value   :    out Unsigned_8) is
begin
   ASM ("INB %%DX",
     Unsigned_8'Asm_Output ("=a", Value),
     Unsigned_16'Asm_Input ("d", Address),
     Volatile => True);
end Read_IO_Port;

procedure Read_IO_Port (Address : in     Unsigned_16;
                        Value   :    out Unsigned_16) is
begin
   ASM ("INW %%DX",
     Unsigned_16'Asm_Output ("=a", Value),
     Unsigned_16'Asm_Input  ("d", Address),
     Volatile => True);
end Read_IO_Port;

procedure Read_IO_Port (Address : in     Unsigned_16;
                        Value   :    out Unsigned_32) is
begin
   ASM ("INL %%DX",
     Unsigned_32'Asm_Output ("=a", Value),
     Unsigned_16'Asm_Input  ("d", Address),
     Volatile => True);
end Read_IO_Port;
```



```
----------------------
WRITING IO PORTS
-----------------------

   procedure Write_IO_Port (Address : in Unsigned_16;
                            Value   : in Unsigned_8) is
   begin
      ASM ("OUTB %%DX",
        No_Output_Operands,
        (Unsigned_8'Asm_Input  ("a", Value),
         Unsigned_16'Asm_Input ("d", Address)));
   end Write_IO_Port;

   procedure Write_IO_Port (Address : in Unsigned_16;
                            Value   : in Unsigned_16) is
   begin
      ASM ("OUTW %%DX",
        No_Output_Operands,
        (Unsigned_16'Asm_Input ("a", Value),
         Unsigned_16'Asm_Input ("d", Address)));
   end Write_IO_Port;

   procedure Write_IO_Port (Address : in Unsigned_16;
                            Value   : in Unsigned_32) is
   begin
      ASM ("OUTL %%DX",
        No_Output_Operands,
        (Unsigned_32'Asm_Input ("a", Value),
         Unsigned_16'Asm_Input ("d", Address)));
   end Write_IO_Port;

end IO_Ports;
```

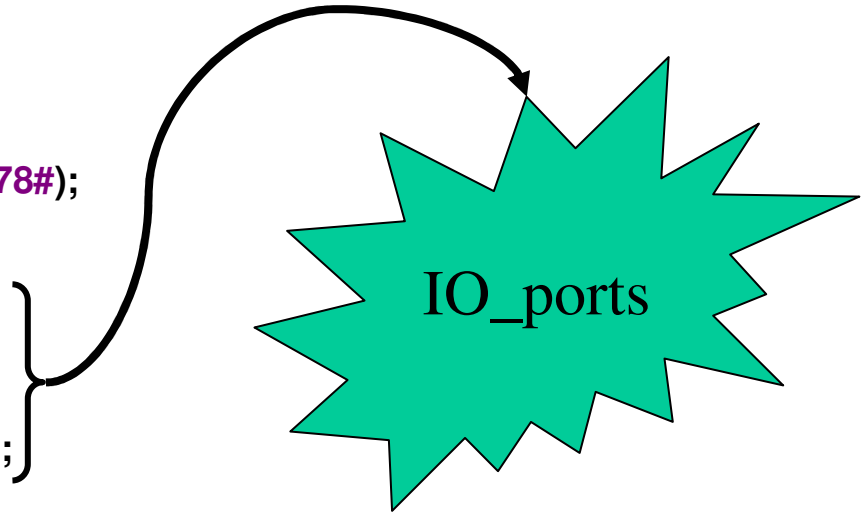
Ler Dados

```
with Ada.Integer_Text_Io, Ada.Text_Io, Io_Ports, Interfaces;
use Ada.Integer_Text_Io, Ada.Text_Io, Io_Ports, Interfaces;

procedure Ler is
   A:Unsigned_8:=0;
   B:Integer;
   Register:Unsigned_16;
begin
Register:=Unsigned_16(16#378#);
     Disable_Interrupts;
     Read_Io_Port(Register,A);
     Enable_Interrupts;
     Put(Integer(A));
end Ler;
```



Escrever Dados

```
with Ada.Integer_Text_Io, Ada.Text_Io, Io_Ports, Interfaces;
use Ada.Integer_Text_Io, Ada.Text_Io, Io_Ports, Interfaces;

procedure Escrever is
   B,C:Integer;
   Register:Unsigned_16;
   Data: Unsigned_8;
begin
      Register:=Unsigned_16(16#378#);
      Get (C);
      Data:=Unsigned_8(C);
      Disable_Interrupts;
      Write_Io_Port(Register, Data);
      Enable_Interrupts;
end Escrever;
```



Utilização placa de aquisição de dados série DAS1600

Algumas características mais importantes da DAS:

16 entradas analógicas "single-ended" (não diferenciais) ou oito entradas diferenciais.

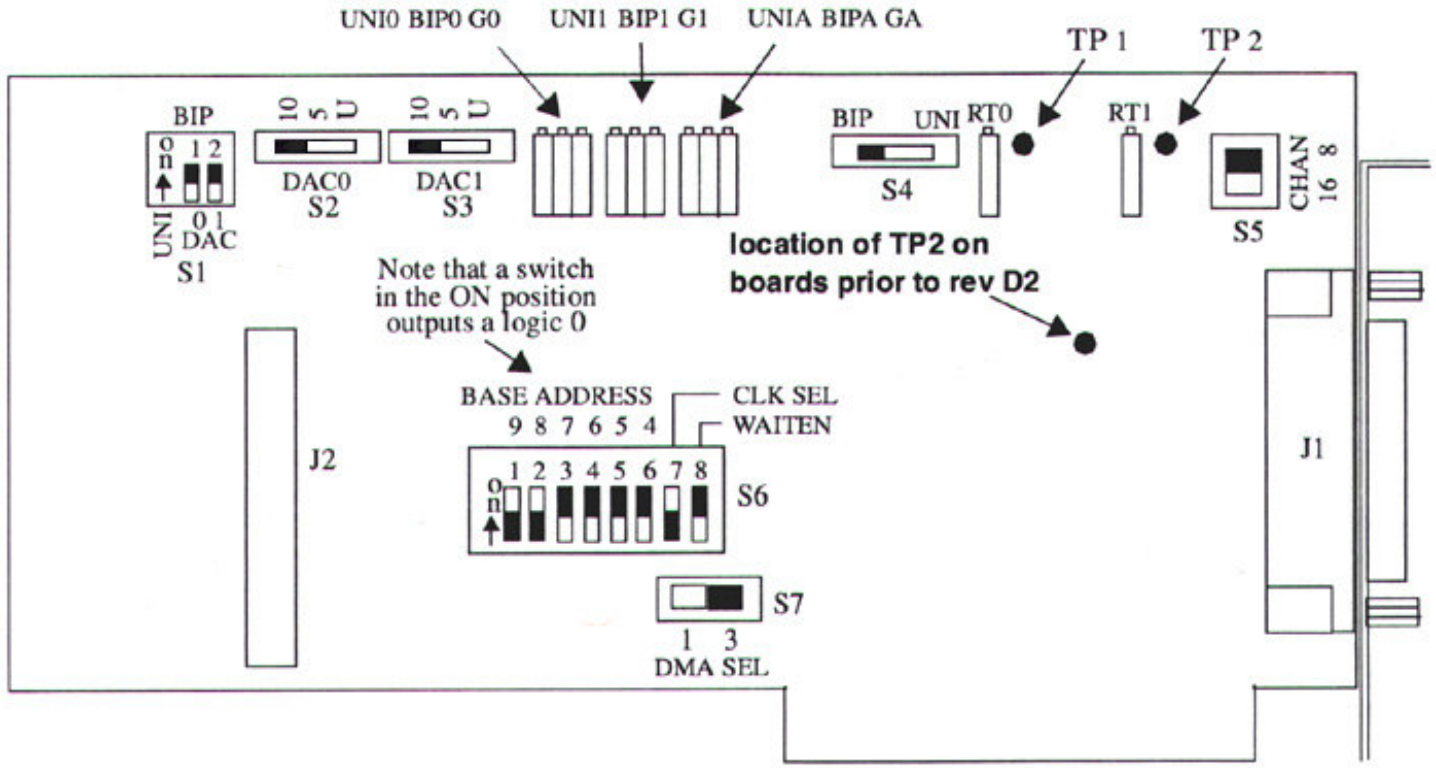
As entradas analógicas podem ser configuradas para unipolares (0-10V) ou bipolares (±10V).

As entradas analógicas são amostradas no máximo a uma taxa de 100K amostras por segundo.

Dois canais de saída aos quais se associam dois conversores DAC de 12 Bits, sendo estas saídas configuradas para gamas de 0-5V, 0-10V, ±5V e ±10V.

Pode ser aplicada uma referencia externa às saídas analógicas permitindo obter outras gamas de valores.

Configurações da placa



Pinout da placa

Rear View

LL GND	Pin 19	Pin 37	Ch0 HI IN
Ch0 LO IN / Ch8 HI IN	Pin 18	Pin 36	Ch1 HI IN
Ch1 LO IN / Ch9 HI IN	Pin 17	Pin 35	Ch2 HI IN
Ch2 LO IN / Ch10 HI IN	Pin 16	Pin 34	Ch3 HI IN
Ch3 LO IN / Ch11 HI IN	Pin 15	Pin 33	Ch4 HI IN
Ch4 LO IN / Ch12 HI IN	Pin 14	Pin 32	Ch5 HI IN
Ch5 LO IN / Ch13 HI IN	Pin 13	Pin 31	Ch6 HI IN
Ch6 LO IN / Ch14 HI IN	Pin 12	Pin 30	Ch7 HI IN
Ch7 LO IN / Ch15 HI IN	Pin 11	Pin 29	LL GND
D/A 0 REF IN*	Pin 10	Pin 28	LL GND
D/A 0 OUT*	Pin 9	Pin 27	D/A 1 OUT*
VREF (-5)*	Pin 8	Pin 26	D/A 1 REF IN*
POWER GND	Pin 7	Pin 25	IP 0 / TRIG 0 / XPCLK
IP 1 / XTRIG	Pin 6	Pin 24	IP 2 / CTR 0 GATE
IP 3	Pin 5	Pin 23	OP 0
OP 1	Pin 4	Pin 22	OP 2
OP 3	Pin 3	Pin 21	CTR 0 CLOCK IN
CTR 0 OUT	Pin 2	Pin 20	CTR 2 OUT
+5 V PWR	Pin 1		

* Not connected in DAS-1400 Series

Registos do ADC

Os registos 300H, 301H e 302H são usados na conversão dos dados de entrada A/D.

Os quatro bits menos significativos do registo 300H, permitem endereçar o canal do qual se pretende fazer a aquisição de dados. Nos restantes bits são guardados os quatro bits menos significativos dos dados lidos.

7	6	5	4	3	2	1	0
CD3	CD2	CD1	CD0	MA3	MA2	MA1	MA0

MA0 a MA3 = n.º do canal a converter
CD0 a CD3 = Quatro bits menos significativos dos dados convertidos do canal seleccionado.

O registo 301H contém os 8 bits mais significativos dos dados convertidos.

7	6	5	4	3	2	1	0
CD11	CD10	CD9	CD8	CD7	CD6	CD5	CD4

CD4 a CD11 = Oito bits mais significativos.

Resolução

$$\frac{Gama}{2^{12}}$$

Os dados são guardados numa palavra de 12 bits divididos pelos 2 registos.

O registo 302H é um registo de escrita/leitura que controla o número dos canais a partir dos quais se faz a aquisição.

Os 4 bits menos significativos referem-se ao canal inferior e os restantes 4 bits referem-se ao canal superior.

Para se realizar conversões num único canal os 4 bits mais significativos devem ser iguais aos 4 bits menos significativos, que é a situação que se adoptou. Por exemplo para se ler apenas a partir do canal número 1 o registo teria (2#00010001#).

7	6	5	4	3	2	1	0
AH3	AH2	AH1	AH0	AL3	AL2	AL1	AL0

AL0 a AL3 = Inicio do scan address
AH0 a AH3 = Fim do scan address

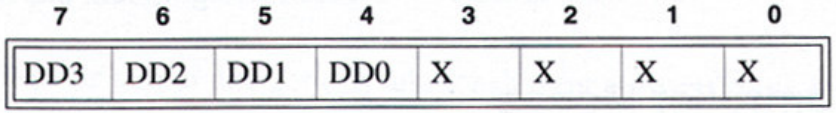
Registos de envio

Os registos 304H a 307H são usados na conversão dos dados de saída D/A.

O 304H e o 305H correspondem ao DAC 0 (canal de saída 0), ou seja ao conversor digital analógico 0.

O 305H e o 306H correspondem ao DAC 1.

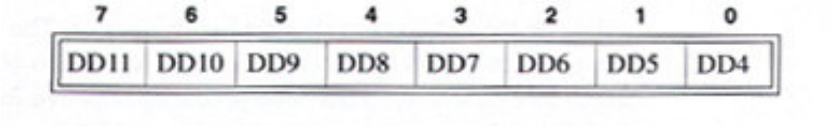
Tal como nos registos de leitura os dados a enviar são repartidos por dois registos e terão um comprimento máximo de 12 bits.



304 h

X = Bits insignificantes

DD0 a DD3 = Quatro bits menos significativos do DAC 0.



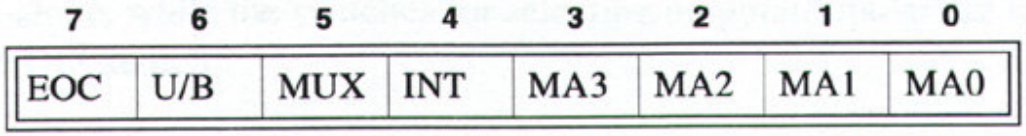
305 h

DD4 a DD11 = Oito bits mais significativos do DAC 0.

Registo de estado

O registo 308H é um registo só de leitura que fornece informações sobre a operação e a configuração da placa.

A atribuição dos bits deste registo é a seguinte:



7	6	5	4	3	2	1	0
EOC	U/B	MUX	INT	MA3	MA2	MA1	MA0

MA0 a MA3 = Fornece o estado corrente do multiplexer de selecção de canal, ou seja o canal que está a ser lido.

INT = Estado do interrupt, se este bit estiver no estado alto (1) significa que uma conversão ADC teve lugar, podendo ser feita outra conversão, leitura

MUX = Informação do estado em que está o switch de configuração para single-ended ou para diferencial. Toma o valor 1 para 16 canais (single-ended) e 0 para 8 canais (diferencial).

U/B = Informação acerca do modo em que o ADC está a operar, unipolar ou bipolar. Se o U/B estiver a 1 unipolar, se 0 bipolar.

EOC = estado do conversor ADC, se estiver a 1 significa que o conversor está ocupado, se estiver a 0, significa que o conversor está pronto para efectuar outra conversão.

Software para acesso a placa de aquisição de dados DAS 1600

Placa_IO.ads

```
with Io_ports, Interfaces;
use  Io_ports, Interfaces;

package Placa_IO is

   type Tensao is range -10 .. 10;

   function Ler (Canal : Unsigned_8) return float;

   Procedure Escrever (Tensao :  float ;
                       Canal : Unsigned_16);

end Placa_IO;
```

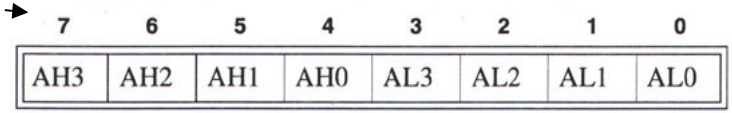
Placa_IO.adb

```
package body  Placa_IO is

   function Ler (Canal : Unsigned_8) return FLOAT is

   Register: constant Unsigned_16:=16#300#;
   -- Registo de leitura com bits LSB

   Register1: constant Unsigned_16:=16#301#;
   -- Registo de leitura com bits MSB

   Register2: constant Unsigned_16:=16#302#;
   -- Registo que permite selecionar a faixa de canais

   Register8: constant Unsigned_16:=16#308#;
   -- Registo de flags

   F_Canais : Unsigned_8;

   Zero : constant Unsigned_8 := 2#00000000#;

   a,b,d : Unsigned_8:=0;

   c : INTEGER;

   Valor: FLOAT;
```

```
begin

    Disable_Interrupts;

      F_Canais := Canal + Canal*16 ; -- Selecção do canal no registo 302H
      write_IO_Port(Register2, F_Canais);
      write_IO_Port(Register1, Zero);
      write_IO_Port(Register, Canal);
      Read_Io_Port(Register8,d); -- Lê para (d) o valor do registo das flags
      d:=d and 2#00010000#;  -- Mascara a flag de fim de conversão do
ADC
      if d=2#00010000# then  -- Se a conversão já foi feita..
        Read_Io_Port(Register,a);
        Read_Io_Port(Register1,b);
      end if;

    Enable_Interrupts;

      a:=a and 2#11110000#;   -- Mascara os bits mais significativos do Reg
#300#
      a:=Interfaces.Shift_Right(a,4);  -- Faz o shift dos 4 bits para a direita
      c:=integer(a)+integer(b)*16;  -- Somar os valores dos dois registos
      if c <= 2048 then  -- Conversão dos valores inteiros para valores de
tensão de -10..10v
        valor:=-10.0+float(c)*10.0/2048.0;
          else
            valor:=(float(c)-2047.0)*10.0/2048.0;
      end if;

    return valor;

  end Ler;
```

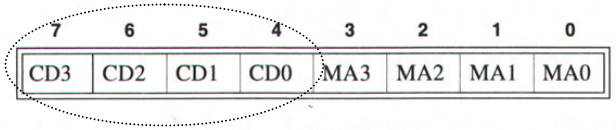
Canal 1 → F_Canais = 17 (10) = 0001 0001 (2)

Canal 4 → F_Canais = 68 (10) = 0100 0100 (2)

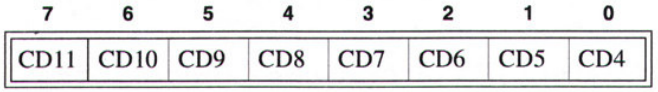


16#302#

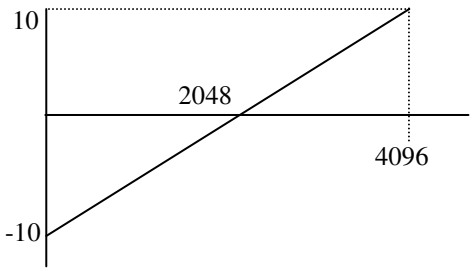
16#300#



16#301#



12 bits = 2^{12} = 4096



```
Procedure Escrever (Tensao :  float ; Canal : Unsigned_16) is

   Register4: Constant Unsigned_16:=772+2*Canal;
   Register5: Constant Unsigned_16:=773+2*Canal;
   Data: Constant Unsigned_8:=2#00000000#;
   Data1: Constant Unsigned_8:=2#10000000#;
   valor: integer;
   ld: Unsigned_8;
   d,hd:unsigned_16;

   begin

      Disable_Interrupts;

         write_IO_Port(Register4, Data);
         write_IO_Port(Register5, Data1);

      Enable_Interrupts;

         valor:=integer(((tensao+10.0)*2047.5)/10.0);
         d:=unsigned_16(valor);
         ld:=unsigned_8(d and 15);          15 (10) = 0000 1111(2)
         ld:=Interfaces.Shift_Left(ld,4);
         hd:=d/16;

      Disable_Interrupts;

          write_IO_Port(Register4,ld);
          write_Io_Port(Register5,hd);

       Enable_Interrupts;

   end Escrever;

end Placa_IO;
```

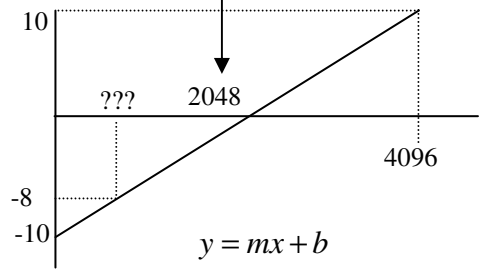
DAC 0:	Register4 = 772 (10) = 304 (16) Register5 = 773 (10) = 305 (16)
DAC 1:	Register4 = 774 (10) = 306 (16) Register5 = 775 (10) = 307 (16)

7	6	5	4	3	2	1	0
DD11	DD10	DD9	DD8	DD7	DD6	DD5	DD4

7	6	5	4	3	2	1	0
DD3	DD2	DD1	DD0	X	X	X	X

305 h 304 h

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0



$$y = mx + b$$

$$m = \frac{10-(-10)}{4096-0} = 0{,}0048828$$

$$y = 0{,}0048828x - 10$$

$$x = \frac{(y+10)}{0{,}0048828} = (y+10)\times 204{,}8 = \frac{(y+10)\times 2048}{10}$$

Ex: - 8 (V) → valor 410 →

d=410 (10) → 110011010 (2)

305 h 304 h

0 0 0 1 1 0 0 1 1 0 1 0 0 0 0 0