

UNIVERSIDAD POLITÉCNICA DE VALENCIA



ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA INDUSTRIAL
DE VALENCIA

PROYECTO:
VERIFICACIÓN DE CIRCUITOS ELECTRÓNICOS ASISTIDA
POR COMPUTADOR

1.- MEMORIA

PROYECTO FIN DE CARRERA
REALIZADO POR : D. EMILIO SAHUQUILLO DOBÓN
DIRIGIDO POR : D. EDUARDO GARCÍA BREIJÓ
ESPECIALIDAD : ELÉCTRICA
SECCIÓN : ELECTRÓNICA
Valencia a 31 de Mayo de 1992

ÍNDICE (MEMORIA)

ÍNDICES.

1.- ENUNCIADO Y OBJETO DEL PROYECTO	1
2.- NECESIDADES DEL PROYECTO	1
2.1.- Necesidad académica	1
2.2.- Necesidad Técnica	2
3.- INTRODUCCIÓN	4
4.- ADQUISICIÓN DE LOS DATOS REALES	6
4.2.- CARACTERÍSTICAS DE LA TARJETA DE ADQUISICIÓN :	
PCLAB-812	8
4.2.1.- Entradas Analógicas (Convertidor A/D)	8
4.2.2.- Salidas Analógicas (Convertidor D/A)	9
4.2.3.- Entrada Digital	9
4.2.4.- Salida Digital	10
4.2.5.- Temporizador/Contador programable	10
4.2.6.- Canal de interrupciones	10
4.2.7.- Canal de acceso directo a memoria (ADM)	11
4.3.- TIPOS DE CONVERSIÓN	11

5.3.2.- Identificadores	43
5.3.3.- Tipos de datos	44
Byte	45
Integer	45
LongInt	45
Real	45
Char	45
String	45
Single	46
Double	46
Extended	46
Comp	46
Registro	46
Array	46
5.3.4.- Estructuras de control y decisión	47
If-Then	47
If-Then-Else	47
Case	47
For-Do	48
Repeat-Until	48
While-Do	48
Goto	49
5.3.5.- Lista de instrucciones	51
ABS	51

ADDR	51
APPEND	51
ARC {Unidad Graph}	51
ARCTAN	52
ASSIGN	52
ASSIGNCRT {Unidad CRT}	52
BAR {Unidad GRAPH}	52
BAR3D {Unidad Graph}	52
BLOCKREAD	53
BLOCKWRITE	53
CHDIR	53
CHR	53
CIRCLE {Unidad GRAPH}	53
CLEARDEVICE {Unidad GRAPH}	53
CLEARVIEWPORT {Unidad GRAPH}	54
CLOSE	54
CLOSEGRAPH {Unidad GRAPH}	54
CLREOL {Unidad CRT}	54
CLRSCR {Unidad CRT}	54
CONCAT	54
COPY	54
COS	54
CSEG	54
DEC	54

DELAY	55
DELETE	55
DISKSIZE {Unidad DOS}	55
DISPOSE	55
DOSVERSION {Unidad DOS}	55
EOF	55
EOLN	55
EXEC {Unidad DOS}	56
EXIT	56
FILESIZE	56
FILLCHAR	56
FINDFIRST {Unidad DOS}	56
FINDNEXT	57
FREEMEM	57
GETCOLOR {Unidad GRAPH}	57
GETDATE {Unidad DOS}	57
GETDIR	57
GETGRAPHMODE {Unidad GRAPH}	58
GETIMAGE {Unidad GRAPH}	58
GETMAXMODE {Unidad GRAPH}	58
GETMAXX {Unidad GRAPH}	58
GETMAXY {Unidad GRAPH}	58
GOTOXY {Unidad CRT}	58
GRAPHRESULT {Unidad GRAPH}	59

HALT	59
IMAGESIZE {Unidad GRAPH}	59
INITGRAPH {Unidad GRAPH}	59
INSERT	59
INTR {Unidad DOS}	60
IORESULT	60
KEEP {Unidad DOS}	60
LENGTH	60
LINE {Unidad GRAPH}	60
LINEREL {Unidad GRAPH}	60
LN	60
LOWVIDEO {Unidad CRT}	61
MARK	61
MKDIR	61
MOVE	61
MSDOS {Unidad DOS}	61
NEW	61
NORMVIDEO {Unidad CRT}	61
NOSOUND {Unidad CRT}	62
PI	62
PIESLICE {Unidad GRAPH}	62
POS	62
PTR	62
PUTIMAGE {Unidad GRAPH}	62

PUTPIXEL {Unidad GRAPH}	63
READ (READLN)	63
READKEY {Unidad CRT}	63
RECTANGLE {Unidad GRAPH}	63
RELEASE	63
RESET	64
RESTORECRTMODE {Unidad CRT}	64
REWRITE	64
ROUND	64
SEEK	64
SETCOLOR y SETBKCOLOR {Unidad GRAPH}	64
SETGRAPHMODE {Unidad GRAPH}	65
SETLINESTYLE {Unidad GRAPH}	65
SETTEXTSTYLE {Unidad GRAPH}	65
SIN	65
SIZEOF	65
SOUND	65
SQR	65
SQRT	66
STR	66
SWAPVECTORS {Unidad DOS}	66
TEXTBACKGROUND {Unidad CRT}	66
TEXTCOLOR {Unidad CRT}	66
TEXTMODE {Unidad CRT}	66

UPCASE	66
VAL	67
WHEREX {Unidad CRT}	67
WHEREY {Unidad CRT}	67
WINDOW {Unidad CRT}	67
WRITE(LN)	67
5.3.6.- Unidades	73
5.3.7.- Ficheros Include	74
5.3.8.- Directivas de compilación	74
\$A+/- : Alineamiento por palabra	75
\$B+/- : Evaluación booleana	76
\$D+/- : Generar información para depurado	76
\$E+/- : Emulación de coma flotante	76
\$F+/- : Forzar llamadas lejanas	77
\$I+/- : Detección de errores de E/S	77
\$I<nombre de fichero> : incluir fichero	78
\$L<nombre de fichero> : Enlace de fichero objeto	78
\$M<tamaño de la pila>,<mínimo del heap (montón)>,<máximo del heap (montón)> : Asignación de memoria	78
\$N+/- : Procesador matemático	79
\$O+/- : Activación de compilación de unidades como <i>overlays</i>	79
\$O<nombre de unidad> : Especificación de nombre de unidad <i>overlay</i>	80

\$R+/- : Comprobación de error de rango en un índice	80
\$S+/- : Comprobación del desbordamiento de la pila	80
\$V+/- : Comprobación de la longitud de cadenas pasadas por referencia	81
5.4.- DIFERENCIAS ENTRE LA PROGRAMACIÓN ORIENTADA A GRÁFICOS Y LA PROGRAMACIÓN TEXTO	81
5.5.- INTERRUPCIONES DOS Y BIOS	84
5.6.- PROGRAMACIÓN ORIENTADA A OBJETO (POO) Y TURBO VISIÓN	86
Código y datos	86
Herencia	86
Encapsulación	87
6.- ADQUISICIÓN DE LOS DATOS TEÓRICOS	88
6.1.- INTRODUCCIÓN	88
6.2.- SIMULACIÓN MIXTA PSPICE	89
6.2.1.- Análisis	89
6.2.1.1.- Tipos de Análisis	90
1.- <i>Análisis en continua</i>	91
2.- <i>Análisis Transitorio</i>	91
3.- <i>Análisis en alterna</i>	91
4.- <i>Análisis de Montecarlo y peor caso</i>	91
6.2.1.2.- Simulación mixta (analógica/digital)	91
6.3.- FICHERO DE SALIDA RESULTADO DE LA SIMULACIÓN	92

6.4.- FORMATO DEL FICHERO .TXT	93
7.- DEFINICIONES DE UNA MAGNITUD PERIÓDICA	97
1) Valor eficaz (RMS, Root Mears Square)	97
2) Valor medio (AV)	97
3) Factor de forma (f_F)	97
4) Factor de rizado (r)	97
8.- SOLUCIÓN AL PROBLEMA PLANTEADO	98
9.- BIBLIOGRAFÍA	103

1.- ENUNCIADO Y OBJETO DEL PROYECTO

El proyecto realizado tiene por título : "Verificación de circuitos electrónicos asistida por computador", y pretende la realización de un programa informático que sea capaz de analizar el funcionamiento (correcto o incorrecto) de cualquier circuito electrónico en régimen de tensiones, permitiendo controlar de forma fiable la calidad de dicho circuito o conjunto de circuitos.

2.- NECESIDADES DEL PROYECTO

Las necesidades que se derivan de la realización práctica de proyecto enunciado son las siguientes :

2.1.- Necesidad académica.

La necesidad académica viene impuesta por la prioridad de la obtención del Título de Ingeniería Técnica Industrial Eléctrica (Sección Electrónica) en la persona de su autor.

El director del proyecto es el profesor titular D. Eduardo García Breijó, perteneciente al Departamento de Ingeniería Electrónica de la Escuela Universitaria de Ingeniería Técnica Industrial de Valencia (Universidad Politécnica de Valencia).

2.2.- Necesidad Técnica.

La necesidad técnica viene impuesta por el compromiso a que debe llegar el Ingeniero en el diseño y fabricación de circuitos electrónicos. Precisamos, pues, un sistema que verifique de forma fiable el funcionamiento de un circuito electrónico, mediante simulación y captura de los datos provenientes de dicho circuito.

El conjunto a desarrollar debe ser económico y de fácil utilización del entorno a desarrollar, para que su adaptabilidad al medio de adquisición sea máxima.

Precisaremos por tanto de un sistema que se constituya por los siguientes elementos :

* Un soporte físico formado por :

- Ordenador personal, o red conectada a cualquier terminal (IBM AX, XT o compatible).
- Un sistema de adquisición de datos fiable, de gran potencia en la conversión analógica/digital y que reúna las condiciones de economía y rapidez en las capturas de datos del circuito a analizar (Tarjeta de adquisición de datos PCLABCARD-812).

* Un soporte lógico formado por :

- Un programa de simulación de circuitos electrónicos que nos

proporcione los datos teóricos adecuados para realizar la comparación de muestras (PSPICE^(R) en su versión educativa o en su versión comercial).

- Un entorno operativo adecuado al funcionamiento del programa de simulación de circuitos electrónicos antes descrito, que se adapte convenientemente al proyecto de aplicación informática a desarrollar (MS-DOS cualquier versión).

Hay que señalar que con realización del proyecto que aquí se presenta, se establece un nuevo modelo a seguir en el diseño de circuitos electrónicos, donde el diseñador (empresa de electrónica, profesional de laboratorio...) consigue una importante ayuda para verificar que lo que se construye funciona de la forma que debe funcionar, controlando de forma adecuada los márgenes de funcionamiento y el control de calidad del circuito en su verificación.

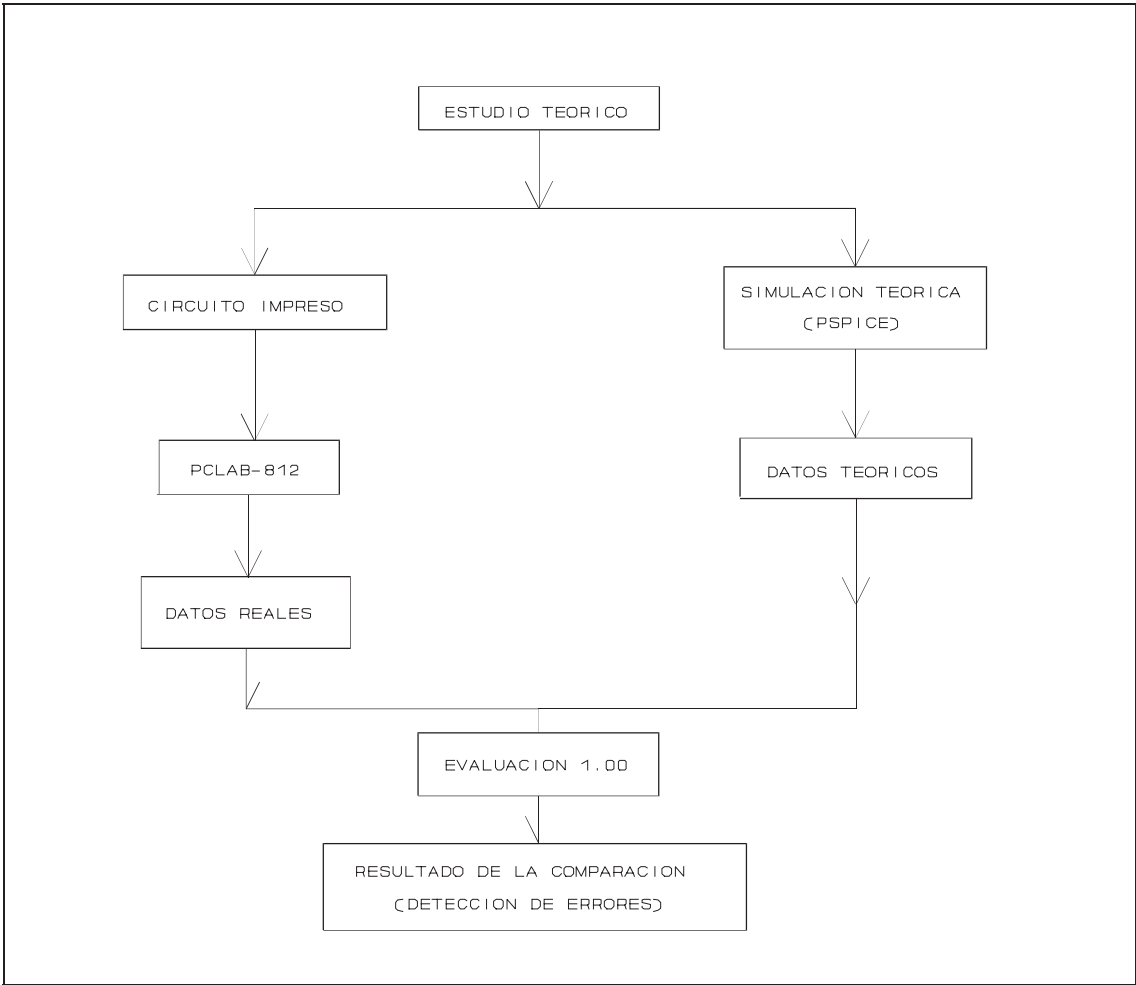
3.- INTRODUCCIÓN

El proyecto de aplicación informática que se desarrolla a continuación trata de llenar el vacío existente en la verificación de circuitos electrónicos. Esto se consigue comparando los datos teóricos de la simulación del circuito (por medio del programa PSPICE^(R)) con los datos reales (obtenidos mediante una tarjeta de adquisición de datos).

El resultado de la comparación especificará el correcto funcionamiento del circuito simulado.

El esquema siguiente nos muestra de forma sencilla el proceso de evaluación de cualquier circuito electrónico.

Tras el estudio teórico, comparamos los datos reales con los teóricos, obteniendo el resultado de error de dicha comparación. Nótese que el programa de comparación ha sido denominado EVALUACIÓN 1.00 y se encarga de comparar y presentar los resultados obtenidos.



Esquema de simulación y verificación de un circuito electrónico

4.- ADQUISICIÓN DE LOS DATOS REALES

Una forma con que el ordenador se comunica con el exterior es mediante las tarjetas de adquisición de datos (entre otras muchas formas).

La tarjeta de adquisición se desarrolla como un sistema capaz de servir de interfase entre el ordenador y el exterior. El aspecto básico es la conversión analógica/digital. El ordenador únicamente puede tratar señales digitales, la transformación a señales digitales se hace obvia.

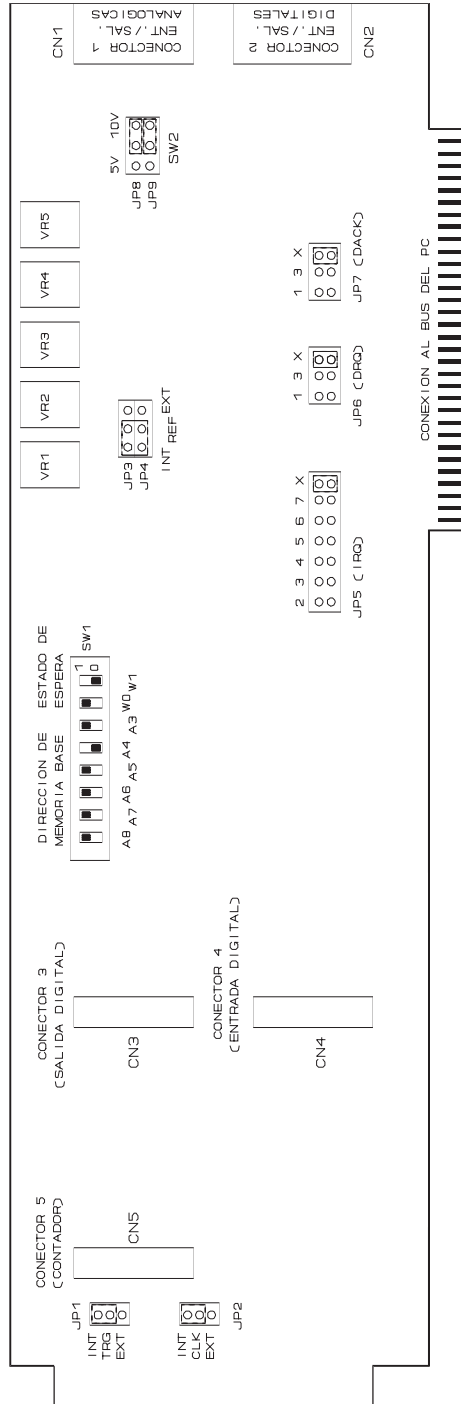
Dentro de esta dinámica se incluye la tarjeta de adquisición de datos PCLAB-812 que, direccionada en posiciones de memoria dinámica, permite al usuario de la misma trabajar el tratamiento de señales en el control de procesos. Su sencillo manejo a través de los programas que se incluyen o por los creados por el propio usuario en cualquier lenguaje (Pascal, Basic, Algol, C...), aunado a la potencia y rapidez de conversión hace de ella una herramienta potente de lectura, transformación y emisión de señales fácilmente tratables.

Si ha todo esto incluimos el carácter económico que se desprende de su gran expansión en el mercado, obtenemos una tarjeta de adquisición fiable y con un gran número de utilidades, tanto en el mundo profesional como en el privado.

La representación esquemática de la tarjeta de adquisición de datos PCLAB-812 se muestra en la figura siguiente.

Se recomienda al lector su consulta para determinar la posición de los distintos elementos

esenciales que la configuran.



Tarjeta de adquisición de datos **PCLAB-812**

4.2.- CARACTERÍSTICAS DE LA TARJETA DE ADQUISICIÓN : PCLAB-812.

Las características de la tarjeta de adquisición de datos PCLAB-812 se distribuyen según el tipo de uso que hagamos de la misma. Son las siguientes :

4.2.1.- Entradas Analógicas (Convertidor A/D) :

Canales :	16 con masa común.
Resolución :	12 bits.
Rango de entrada :	Bipolar: ± 10 , ± 5 , ± 2 , ± 1 Voltios.
Tensión límite :	± 30 V. máxima continua.
Tipo de conversión :	Aproximaciones sucesivas.
Convertidor :	HADO574Z.
Velocidad de conversión :	30KHz.
Precisión :	0.015% de la lectura (± 1 bit).
Modos de disparo:	Disparo por programa. Disparo utilizando el temporizador de la misma. Disparo por señal externa.
Transferencia de los datos:	Por programa. Por interrupciones. Por acceso directo a memoria.

4.2.2.- Salidas Analógicas (Convertidor D/A) :

Canales :	2 canales.
Resolución :	12 bits.
Rango de salida :	De 0 a 5V. por referencia interna. ±10V. por referencia interna.
Tensión de referencia :	Interna: -5V. (± 0.05 V). Externa: ± 10 V máxima(DC o AC).
Tipo de conversión :	Multiplicador monolítico (R-2R).
Convertidor :	DAC7541AKN o equivalente.
Linealidad :	$\pm 1/2$ bit.
Carga de salida :	± 5 mA máxima.
Tiempo de establecimiento :	30 microsegundos.

4.2.3.- Entrada Digital :

Canal :	16 bits.
Nivel :	Compatible TTL.
Tensión de entrada :	Nivel bajo: 0.8 V máxima. Nivel alto: 2.0 V máxima.
Carga de entrada :	Nivel bajo: 0.40 mA máxima. Nivel alto: 0.05 mA máxima.

4.2.4.- Salida Digital :

Canal : 16 bits.
Nivel : Compatible TTL.
Tensión salida : Nivel L: 8 mA a 0.5 V máxima.
Nivel H: 0.4 mA a 2.4 V mínima.

4.2.5.- Temporizador/Contador programable :

Dispositivo : INTEL 8253-5.
Contadores : 3 canales, 16 bits.
2: generación de disparos.
1: libre para el usuario.
Entrada : Compatible TTL/DTL/CMOS.
Base de tiempos : 2 MHz.
Salida/disparo : 35 minutos/pulso a 0.5 MHz.

4.2.6.- Canal de interrupciones :

Nivel : IRQ: 2 a 7.
Habilitación : S0, S1 y S2 (reg. control).

4.2.7.- Canal de acceso directo a memoria (ADM) :

Nivel : 1 ò 3.

Habilitación : S0, S1 y S2 (reg. control).

4.3.- TIPOS DE CONVERSIÓN.

La conversión de los datos que realiza la tarjeta de adquisición dependerá del uso a que esté destinada. Utilizaremos la conversión Analógica/Digital (A/D) siempre que deseemos transmitir valores desde el exterior hasta el ordenador. Por el contrario utilizaremos la conversión Digital/Analógica (D/A) para transmitir información o controlar procesos desde el ordenador hacia el exterior.

A continuación veremos de forma introductoria estos tipos de conversión.

4.3.1.- CONVERSIÓN ANALÓGICA/DIGITAL (A/D).

La conversión A/D se caracteriza por su utilización como lectura de datos por el canal de entrada de la tarjeta (uno de los 16 canales de entrada) hacia el ordenador. La tarjeta de adquisición se encarga pues de transformar el dato (tensión) analógico, en dato digital, que se almacenará en posiciones determinadas de memoria.

Este tipo de conversión es el utilizado en el proyecto de programación

presentado, pues los requerimientos de función se limitaban a la lectura de tensiones exteriores pertenecientes al circuito a verificar.

La tarjeta de adquisición que se ha utilizado posee una resolución de 12 bits que pertenecen a dos registros almacenados en posiciones consecutivas de memoria.

Esto hace que la tarjeta haya sido direccionada de una manera determinada por el usuario de la misma. Este direccionamiento se modifica según un conmutador existente en la placa (Ver figura esquemática de la misma, conector SW1); las posiciones que se pueden modificar son las siguientes (Ver cuadro adjunto) :

DIRECCIÓN E/S (HEXA.)	CONMUTADOR SW1						
	A9	A8	A7	A6	A5	A4	A3
200-20F	1	0	0	0	0	0	X
210-21F	1	0	0	0	0	1	X
220-22F	1	0	0	0	1	0	X
300-3FF	1	1	0	0	0	0	X
3F0-3FF	1	1	1	1	1	1	X

Generalmente el posicionamiento de la tarjeta se realiza a partir de la dirección de memoria \$200.

De esta manera los registros en que almacena la información resultante de la conversión A/D se encuentran en \$204 (parte baja) y \$205 (parte alta). Cabe señalar que el registro donde se almacena la parte alta quedan desaprovechados los 4 bits de mayor peso, por tratarse de un tarjeta con resolución de 12 bits; utilizando los 8 bits del registro \$204 y los 4 bits de menor peso del registro \$205.

De forma esquemática, para comprender el direccionamiento sucesivo de las 16 posiciones de memoria que ocupa la tarjeta de adquisición supondremos que comienza su dirección en una posición denominada **BASE**; las sucesivas posiciones que caracterizan su colocación en el ordenador son las siguiente (Ver cuadro) :

El número del canal de entrada sobre el que se realizan las lecturas viene determinado al escribir en la posición de memoria \$210 (que equivale a la dirección **BASE+10**).

Dirección E/S	Lectura	Escritura
BASE+0	Contador 0	Contador 0
BASE+1	Contador 1	Contador 1
BASE+2	Contador 2	Contador 2
BASE+3	No utilizado	Control contador
BASE+4	A/D byte bajo	CH1 D/A bajo
BASE+5	A/D byte alto	CH1 D/A alto
BASE+6	D/I byte bajo	CH2 D/A bajo
BASE+7	D/I byte alto	CH2 D/A alto
BASE+8	-----	-----
BASE+9	No utilizado	No utilizado
BASE+10	No utilizado	R. Multiplexor
BASE+11	No utilizado	R. Control
BASE+12	No utilizado	Disparosoft A/D
BASE+13	No utilizado	D/O byte bajo
BASE+14	No utilizado	D/O byte alto
BASE+15	No utilizado	No utilizado

D/I = Entrada digital.

D/O = Salida digital.

La conversión Analógica/Digital puede ser iniciada según una de las siguientes formas de disparo :

- 1) Disparo por programa.
- 2) Disparo por temporizador (disparo interno).
- 3) Disparo externo (mediante señal externa).

La forma de lectura o transferencia de los datos obtenidos en la conversión puede llevarse a cabo de las siguientes formas :

- 1) Transferencias por programa.
- 2) Transferencias por interrupciones.
- 3) Transferencias por acceso directo a memoria.

Para realizar de forma adecuada las conversiones A/D hay que tener en cuenta los siguientes aspectos :

a)Escribir el **canal de entrada** sobre el que se van a efectuar las lecturas. Esta escritura se realizará en el registro BASE+10, indicando de 0 a 15.

b)Indicar el **rango de tensión** máximo que puede leer la tarjeta de adquisición. Este rango (bipolar) puede variar entre los valores estándar que presenta la tarjeta. Según las posiciones del conector **SW2** (ver posición en esquema general) los rangos de tensión se dividen en :

CONMUTADOR SW2					RANGO BIPOLAR
1	2	3	4	5	
ON	OFF	ON	OFF	X	± 10 V.
OFF	ON	ON	OFF	X	± 5 V.
ON	OFF	OFF	ON	X	± 2 V.
OFF	ON	OFF	ON	X	± 1 V.

c) Y por último deberemos **indicar el modo de disparo** que hemos elegido para el tipo de conversión definida. Este modo de disparo se selecciona escribiendo la combinación adecuada en el los 3 bits de menor peso (**0, 1, 2**) del registro de control BASE+11 (\$211 generalmente). Ver cuadro siguiente :

BYTE DEL REGISTRO DE CONTROL (BASE+11)							
7	6	5	4	3	2	1	0
X	X	X	X	X	S2	S1	S0

Los tipos de disparo (y transferencia de los datos) más utilizados son los siguientes :

4.3.1.1.- TIPOS DE DISPARO.

4.3.1.1.1.- Disparo por programa y transferencia por programa.

Escribiendo en el registro de control (BASE+11) la combinación 0-0-1 se consigue este tipo de disparo.

S2 S1 S0

0 0 1

Una vez habilitado, el disparo se consigue escribiendo cualquier valor en el registro BASE+12.

En la transferencia de datos por programa, las lecturas debe realizarlas el ordenador comprobando el bit **DRDY** (bit 4 de 0..7) del registro BASE+5 (\$205). Cuando el bit toma el valor 0, el dato muestreado es válido, y se puede transferir a la memoria del ordenador.

Se incluyen dos programas que realizan este tipo de disparo y transferencia. El primero utiliza un único canal (elegido por el usuario), y el segundo es para un conjunto de canales especificados (de 0 a 15).

PROGRAMA 1 :

(*DISPARO POR PROGRAMA,TRANSFERENCIA POR PROGRAMA, UN CANAL*)
(*CON TRANSFORMACIÓN DE LOS DATOS A N° REALES*)

(*TRANSFERENCIA DE DATOS A MEMORIA DINÁMICA*)

```

PROGRAM ADQUISICIÓN_DE_DATOS;
USES DOS, CRT;
CONST
  BASE=$200;
  REGCONT1=BASE+1;
  REGCONT2=BASE+2;
  CONTCONTR=BASE+3;
  DATOADL=BASE+4;
  DATOADH=BASE+5;
  DATODA1L=BASE+4;
  DATODA1H=BASE+5;
  DATODA2L=BASE+6;
  DATODA2H=BASE+7;
  CLEARINT=BASE+8;
  CANALMUX=BASE+10;
  REGCONTROL=BASE+11;
  REGDISPARO=BASE+12;

TYPE
  PUNTEROV=^V;

  V=RECORD
    Vmuestras:REAL;
    CANAL:INTEGER;
    DATO:INTEGER;
    SIG:PUNTEROV;
  END;

VAR
  FIN_CONVERSIÓN,DATO_BAJO,DATO_ALTO:BYTE;
  I,J,MAX,CAN,TENS,MUESTRA:INTEGER;
  PRIMERO,ANTERIOR,ACTUAL:PUNTEROV;
  MUEST:REAL;
  TECLA:CHAR;

(*---Escribe en memoria dinámica las muestras con su N° de CANAL---*)

PROCEDURE OTRAMUESTRA(C,D:INTEGER;V:REAL);
BEGIN
  CLRSCR;
  V:=TENS+(2*TENS/4095)*(V-4095);
  IF PRIMERO=NIL THEN
    BEGIN
      NEW(ACTUAL);
      ACTUAL^.CANAL:=C;
      ACTUAL^.Vmuestras:=V;
      ACTUAL^.DATO:=D;
      PRIMERO:=ACTUAL;
      ACTUAL^.SIG:=NIL
    END
  ELSE
    BEGIN
      ANTERIOR:=ACTUAL;

```

```

NEW(ACTUAL);
ACTUAL^.CANAL:=C;
ACTUAL^.Vmuestras:=V;
ACTUAL^.DATO:=D;
ANTERIOR^.SIG:=ACTUAL;
ACTUAL^.SIG:=NIL
END
END;

(*-----Lee las muestras de memoria dinámica-----*)

PROCEDURE LEERMUESTRAS;
VAR
  MTRA:PUNTEROV;

BEGIN
  MTRA:=PRIMERO;
  WHILE MTRA<>NIL DO
    BEGIN
      WITH MTRA ^ DO
        WRITELN('CANAL N° ',CANAL,' ; MUESTRA N°',DATO,' = ',Vmuestras);
        MTRA:=MTRA^.SIG
      END
    END;

(*-----Programa principal-----*)

BEGIN
REPEAT
  PRIMERO:=NIL; (*-----Inicializa el puntero-----*)
  CLRSCR;
  PORT[REGCONTROL]:=0;
  WRITELN
('INTRODUCE EL N° DE CANAL AL CUAL HAS CONECTADO LA PCLAB(0 A 15)= ');
  READ(CAN);
  WRITELN('TENSION DE LA PCLAB=');
  READ(TENS);
  WRITELN('INTRODUCE EL NUMERO DE MUESTRAS A EFECTUAR= ');
  READ(MAX);
  PORT[CANALMUX]:=CAN;
  PORT[REGCONTROL]:=1;
  FOR I:=0 TO MAX-1 DO
    BEGIN
      PORT[REGDISPARO]:=0;
      REPEAT FIN_CONVERSION:=PORT[DATOADH];
      UNTIL FIN_CONVERSION<=$F;
      DATO_BAJO:=PORT[DATOADL];
      DATO_ALTO:=PORT[DATOADH] MOD 16;
      MUEST:=DATO_ALTO*256+DATO_BAJO;
      OTRAMUESTRA(J,I,MUEST)
    END;
  PORT[REGCONTROL]:=0;
  LEERMUESTRAS;
  WRITELN(' ');
  WRITELN('PULSAR CUALQUIER TECLA PARA REPETIR, ESCAPE PARA SALIR');

```

```
TECLA:=READKEY
UNTIL TECLA=#27
END.
```

PROGRAMA 2 :

(*DISPARO POR PROGRAMA,TRANSFERENCIA POR PROGRAMA, VARIOS CANALES*)
(*CON TRANSFORMACION DE LOS DATOS A N° REALES*)
(*TRANSFERENCIA DE DATOS A MEMORIA DINAMICA*)

```
PROGRAM ADQUISICION_DE_DATOS;
USES DOS, CRT;
CONST
  BASE=$200;
  REGCONT1=BASE+1;
  REGCONT2=BASE+2;
  CONTCONTR=BASE+3;
  DATOADL=BASE+4;
  DATOADH=BASE+5;
  DATODAIL=BASE+4;
  DATODAIH=BASE+5;
  DATODA2L=BASE+6;
  DATODA2H=BASE+7;
  CLEARINT=BASE+8;
  CANALMUX=BASE+10;
  REGCONTROL=BASE+11;
  REGDISPARO=BASE+12;

TYPE
  PUNTEROV=^V;

  V=RECORD
    Vmuestras:REAL;
    CANAL:INTEGER;
    DATO:INTEGER;
    SIG:PUNTEROV
  END;

VAR
  FIN_CONVERSION,DATO_BAJO,DATO_ALTO:BYTE;
  I,J,MAX,CAN,TENS,MUESTRA:INTEGER;
  PRIMERO,ANTERIOR,ACTUAL:PUNTEROV;
  MUEST:REAL;
  TECLA:CHAR;
```

(*--Escribe en memoria dinámica las muestras con su N° de CANAL--*)

```
PROCEDURE OTRAMUESTRA(C,D:INTEGER;V:REAL);
```

```

BEGIN
CLRSCR;
V:=TENS+(2*TENS/4095)*(V-4095);
IF PRIMERO=NIL THEN
  BEGIN
  NEW(ACTUAL);
  ACTUAL^.CANAL:=C;
  ACTUAL^.Vmuestras:=V;
  ACTUAL^.DATO:=D;
  PRIMERO:=ACTUAL;
  ACTUAL^.SIG:=NIL
  END
ELSE
  BEGIN
  ANTERIOR:=ACTUAL;
  NEW(ACTUAL);
  ACTUAL^.CANAL:=C;
  ACTUAL^.Vmuestras:=V;
  ACTUAL^.DATO:=D;
  ANTERIOR^.SIG:=ACTUAL;
  ACTUAL^.SIG:=NIL
  END
END;

(*-----Lee las muestras de memoria dinámica-----*)

PROCEDURE LEERMUESTRAS;
VAR
  MTRA:PUNTEROV;

BEGIN
  MTRA:=PRIMERO;
  WHILE MTRA<>NIL DO
    BEGIN
      WITH MTRA^ DO
        WRITELN('CANAL N° ',CANAL,' ; MUESTRA N°',DATO,' = ',Vmuestras);
        MTRA:=MTRA^.SIG
      END
    END;

(*-----Programa principal-----*)

BEGIN
REPEAT
  PRIMERO:=NIL; (*-----Inicializa el puntero-----*)
  CLRSCR;
  PORT[REGCONTROL]:=0;
  WRITELN
('INTRODUCE EL N° DE CANALES CONECTADOS A LA PCLAB(0 A 15)= ');
  READ(CAN);
  WRITELN('TENSION DE LA PCLAB=');
  READ(TENS);
  WRITELN('INTRODUCE EL NUMERO DE MUESTRAS A EFECTUAR= ');
  READ(MAX);
  PORT[REGCONTROL]:=1;

```

```
FOR J:=0 TO CAN DO
  FOR I:=0 TO MAX-1 DO
    BEGIN
      PORT[CANALMUX]:=J;
      PORT[REGDISPARO]:=0;
      REPEAT FIN_CONVERSION:=PORT[DATOADH];
      UNTIL FIN_CONVERSION<=$F;
      DATO_BAJO:=PORT[DATOADL];
      DATO_ALTO:=PORT[DATOADH] MOD 16;
      MUEST:=DATO_ALTO*256+DATO_BAJO;
      OTRAMUESTRA(J,I,MUEST)
    END;
  PORT[REGCONTROL]:=0;
  LEERMUESTRAS;
  WRITELN(' ');
  WRITELN('PULSAR CUALQUIER TECLA PARA REPETIR, ESCAPE PARA SALIR');
  TECLA:=READKEY
UNTIL TECLA=#27
END.
```

4.3.1.1.2.- Disparo interno y transferencia por programa.

El disparo interno se caracteriza por muestrear los datos de los canales seleccionados a una frecuencia elegida por el usuario programador. Teniéndose en cuenta que a frecuencias muy elevadas (>10 KHz.) la fiabilidad de los datos obtenidos disminuye notablemente.

Este disparo se realiza por el temporizador que posee la tarjeta de adquisición (temporizador/contador 8253 de INTEL), escribiendo en el registro de control :

S2 S1 S0

1 1 0

La frecuencia a la que se realizan las medidas se calculará de la siguiente forma :

$$F = \frac{\text{Frecuencia Base}}{(HB1 \times 256 + LB1) \times (HB2 \times 256 + LB2)}$$

Donde los elementos de la ecuación son los siguientes :

HB1 = parte alta del contenido del contador 1.

LB1 = parte baja del contenido del contador 1.

HB2 = parte alta del contenido del contador 2.

LB2 = parte baja del contenido del contador 2.

El contenido de cada contador deberá ser mayor o igual a 2.

Como ejemplos de sencillos programas de aplicación de este modo de disparo se incluyen los siguientes programas :

PROGRAMA 3 :

(*DISPARO INTERNO, TRANSFERENCIA POR PROGRAMA, UN CANAL*)
(*CON TRANSFORMACION DE LOS DATOS A N° REALES*)

(*TRANSFERENCIA DE DATOS A MEMORIA DINAMICA*)

```

PROGRAM ADQUISICION_DE_DATOS;
USES DOS, CRT;
CONST
  BASE=$200;
  REGCONT1=BASE+1;
  REGCONT2=BASE+2;
  CONTCONTR=BASE+3;
  DATOADL=BASE+4;
  DATOADH=BASE+5;
  DATODA1L=BASE+4;
  DATODA1H=BASE+5;
  DATODA2L=BASE+6;
  DATODA2H=BASE+7;
  CLEARINT=BASE+8;
  CANALMUX=BASE+10;
  REGCONTROL=BASE+11;
  REGDISPARO=BASE+12;

TYPE
  PUNTEROV=^V;

  V=RECORD
    Vmuestras:REAL;
    CANAL:INTEGER;
    DATO:INTEGER;
    SIG:PUNTEROV
  END;

VAR
  FIN_CONVERSION,DATO_BAJO,DATO_ALTO:BYTE;
  FRE,I,MAX,CAN,TENS,MUESTRA:INTEGER;
  PRIMERO,ANTERIOR,ACTUAL:PUNTEROV;
  MUEST:REAL;
  TECLA:CHAR;

```

(*---Escribe en memoria dinámica las muestras con su N° de CANAL---*)

```

PROCEDURE OTRAMUESTRA(C,M:INTEGER;V:REAL);
(*Transferencia de las muestras a memoria dinámica*)
BEGIN
  CLRSCR;
  V:=TENS+(2*TENS/4095)*(V-4095);
  IF PRIMERO=NIL THEN
    BEGIN
      NEW(ACTUAL);
      ACTUAL^.CANAL:=C;
      ACTUAL^.Vmuestras:=V;
      ACTUAL^.DATO:=M;
      PRIMERO:=ACTUAL;
      ACTUAL^.SIG:=NIL
    END
  ELSE
    BEGIN

```

```

ANTERIOR:=ACTUAL;
NEW(ACTUAL);
ACTUAL^.CANAL:=C;
ACTUAL^.Vmuestras:=V;
ACTUAL^.DATO:=M;
ANTERIOR^.SIG:=ACTUAL;
ACTUAL^.SIG:=NIL
END
END;

(*-----Lee las muestras de memoria dinámica-----*)

PROCEDURE LEERMUESTRAS;

VAR
  MTRA:PUNTEROV;

BEGIN
  MTRA:=PRIMERO;
  WHILE MTRA<>NIL DO
    BEGIN
      WITH MTRA^ DO
        WRITELN('CANAL N° ',CANAL,', MUESTRA N° ',DATO,',',Vmuestras);
      MTRA:=MTRA^.SIG
    END
  END;

END;

(*-----Programación de la Frecuencia-----*)

PROCEDURE FRECUENCIA (FRECUENCIA:INTEGER);
VAR D1,D2,AUX:LONGINT;
BEGIN
  AUX:=200000 DIV FRECUENCIA;
  D1:=ROUND(SQRT(AUX));
  D2:=D1;
  PORT[CONTR]:=$74;
  PORT[REGCONT1]:=D1 MOD 256;
  PORT[REGCONT1]:=D1 DIV 256;
  PORT[CONTR]:=$B4;
  PORT[REGCONT2]:=D2 MOD 256;
  PORT[REGCONT2]:=D2 DIV 256
END;

(*-----Programa principal-----*)

BEGIN
REPEAT
  PRIMERO:=NIL; (*-----Inicializa el puntero-----*)
  CLRSCR;
  WRITELN('INTRODUCE LA FRECUENCIA DE MUESTREO: ');
  READ(FRE);
  PORT[REGCONTROL]:=0;
  WRITELN
('INTRODUCE EL N° DE CANAL AL CUAL HAS CONECTADO LA PCLAB (0 A 15)= ');
  READLN(CAN);

```

```
WRITELN('INTRODUCE EL N° DE MUESTRAS: ');
READ(MAX);
WRITELN('INTRODUCE EL VALOR DE LA TENSION MAXIMA EN LA PCLAB');
READLN(TENS);
PORT[CANALMUX]:=CAN;
PORT[REGCONTROL]:=6;
FRECUENCIA(FRE);
  FOR I:=0 TO MAX-1 DO
    BEGIN
      REPEAT FIN_CONVERSION:=PORT[DATOADH];
        UNTIL FIN_CONVERSION<=$F;
      DATO_BAJO:=PORT[DATOADL];
      DATO_ALTO:=PORT[DATOADH] MOD 16;
      MUEST:=DATO_ALTO*256+DATO_BAJO;
      OTRAMUESTRA(CAN,I,MUEST);
      REPEAT FIN_CONVERSION:=PORT[DATOADH];
        UNTIL FIN_CONVERSION>$F
      END;
    PORT[REGCONTROL]:=0;
    LEERMUESTRAS;
    WRITELN(' ');
    WRITELN('PULSAR CUALQUIER TECLA PARA REPETIR, ESCAPE PARA SALIR');
    TECLA:=READKEY
  UNTIL TECLA=#27;
  PORT[REGCONTROL]:=0
END.
```

PROGRAMA 4 :

(*DISPARO INTERNO, TRANSFERENCIA POR PROGRAMA, VARIOS CANALES*)
(*CON TRANSFORMACION DE LOS DATOS A N° REALES*)
(*TRANSFERENCIA DE DATOS A MEMORIA DINAMICA*)

```
PROGRAM ADQUISICION_DE_DATOS;
USES DOS, CRT;
CONST
  BASE=$200;
  REGCONT1=BASE+1;
  REGCONT2=BASE+2;
  CONTCONTR=BASE+3;
  DATOADL=BASE+4;
  DATOADH=BASE+5;
  DATODA1L=BASE+4;
  DATODA1H=BASE+5;
  DATODA2L=BASE+6;
  DATODA2H=BASE+7;
  CLEARINT=BASE+8;
```

```

CANALMUX=BASE+10;
REGCONTROL=BASE+11;
REGDISPARO=BASE+12;

```

```

TYPE

```

```

  PUNTEROV=^V;

```

```

  V=RECORD

```

```

    Vmuestras:REAL;
    CANAL:INTEGER;
    DATO:INTEGER;
    SIG:PUNTEROV
  END;

```

```

VAR

```

```

  FIN_CONVERSION,DATO_BAJO,DATO_ALTO:BYTE;
  FRE,I,J,MAX,CAN,TENS,MUESTRA:INTEGER;
  PRIMERO,ANTERIOR,ACTUAL:PUNTEROV;
  MUEST:REAL;
  TECLA:CHAR;

```

(*---Escribe en memoria dinámica las muestras con su N° de CANAL---*)

```

PROCEDURE OTRAMUESTRA(C,M:INTEGER;V:REAL);

```

(*Transferencia de las muestras a memoria dinámica*)

```

BEGIN

```

```

  CLRSCR;

```

```

  V:=TENS+(2*TENS/4095)*(V-4095);

```

```

  IF PRIMERO=NIL THEN

```

```

    BEGIN

```

```

      NEW(ACTUAL);

```

```

      ACTUAL^.CANAL:=C;

```

```

      ACTUAL^.Vmuestras:=V;

```

```

      ACTUAL^.DATO:=M;

```

```

      PRIMERO:=ACTUAL;

```

```

      ACTUAL^.SIG:=NIL

```

```

    END

```

```

  ELSE

```

```

    BEGIN

```

```

      ANTERIOR:=ACTUAL;

```

```

      NEW(ACTUAL);

```

```

      ACTUAL^.CANAL:=C;

```

```

      ACTUAL^.Vmuestras:=V;

```

```

      ACTUAL^.DATO:=M;

```

```

      ANTERIOR^.SIG:=ACTUAL;

```

```

      ACTUAL^.SIG:=NIL

```

```

    END

```

```

END;

```

(*-----Lee las muestras de memoria dinámica-----*)

```

PROCEDURE LEERMUESTRAS;

```

```

VAR

```

```

  MTRA:PUNTEROV;

```

```

BEGIN
  MTRA:=PRIMERO;
  WHILE MTRA<>NIL DO
    BEGIN
      WITH MTRA ^ DO
        WRITELN('CANAL N° ',CANAL,'; MUESTRA N° ',DATO,','=','Vmuestras);
        MTRA:=MTRA^.SIG
      END
    END;

```

(*-----Programación de la Frecuencia-----*)

```

PROCEDURE FRECUENCIA (FRECUENCIA:INTEGER);
VAR D1,D2,AUX:LONGINT;
BEGIN
  AUX:=2000000 DIV FRECUENCIA;
  D1:=ROUND(SQRT(AUX));
  D2:=D1;
  PORT[CONTR]:=$74;
  PORT[REGCONT1]:=D1 MOD 256;
  PORT[REGCONT1]:=D1 DIV 256;
  PORT[CONTR]:=$B4;
  PORT[REGCONT2]:=D2 MOD 256;
  PORT[REGCONT2]:=D2 DIV 256
END;

```

(*-----Programa principal-----*)

```

BEGIN
REPEAT
  PRIMERO:=NIL; (*-----Inicializa el puntero-----*)
  CLRSCR;
  WRITELN('INTRODUCE LA FRECUENCIA DE MUESTREO: ');
  READ(FRE);
  PORT[REGCONTROL]:=0;
  WRITELN
('INTRODUCE EL N° DE CANALES CONECTADOS A LA PCLAB(0 A 15)= ');
  READLN(CAN);
  WRITELN('INTRODUCE EL N° DE MUESTRAS: ');
  READ(MAX);
  WRITELN('INTRODUCE EL VALOR DE LA TENSION MAXIMA EN LA PCLAB');
  READLN(TENS);
  PORT[REGCONTROL]:=6;
  FRECUENCIA(FRE);
  FOR J:=0 TO CAN DO
    FOR I:=0 TO MAX-1 DO
      BEGIN
        PORT[CANALMUX]:=J;
        REPEAT FIN_CONVERSION:=PORT[DATOADH];
          UNTIL FIN_CONVERSION<=$F;
        DATO_BAJO:=PORT[DATOADL];
        DATO_ALTO:=PORT[DATOADH] MOD 16;
        MUEST:=DATO_ALTO*256+DATO_BAJO;
        OTRAMUESTRA(J,I,MUEST);
        REPEAT FIN_CONVERSION:=PORT[DATOADH];

```

```
        UNTIL FIN_CONVERSION>$F
        END;
    PORT[REGCONTROL]:=0;
    LEERMUESTRAS;
    Writeln(' ');
    Writeln('PULSAR CUALQUIER TECLA PARA REPETIR, ESCAPE PARA SALIR');
    TECLA:=READKEY
UNTIL TECLA=#27;
PORT[REGCONTROL]:=0
END.
```

PROGRAMA 5 :

(*DISPARO INTERNO, TRANSFERENCIA POR PROGRAMA, VARIOS CANALES*)
(*CON TRANSFORMACION DE LOS DATOS A N° REALES*)
(*TRANSFERENCIA DE DATOS A MEMORIA DINAMICA*)
(*POSIBILIDAD DE TRANSFERIR LOS DATOS A DISCO*)

```
PROGRAM ADQUISICION_DE_DATOS;
USES DOS, CRT;
CONST
    BASE=$200;
    REGCONT1=BASE+1;
    REGCONT2=BASE+2;
    CONTCONTR=BASE+3;
    DATOADL=BASE+4;
    DATOADH=BASE+5;
    DATODAI1L=BASE+4;
    DATODAI1H=BASE+5;
    DATODA2L=BASE+6;
    DATODA2H=BASE+7;
    CLEARINT=BASE+8;
    CANALMUX=BASE+10;
    REGCONTROL=BASE+11;
    REGDISPARO=BASE+12;

TYPE
    PUNTEROV=^V;

    V=RECORD
        Vmuestras:REAL;
        CANAL:INTEGER;
        DATO:INTEGER;
        SIG:PUNTEROV
    END;

VAR
    FIN_CONVERSION,DATO_BAJO,DATO_ALTO:BYTE;
    FRE,I,J,MAX,CAN,TENS,MUESTRA:INTEGER;
    PRIMERO,ANTERIOR,ACTUAL:PUNTEROV;
```

```

MUEST:REAL;
TECLA,T:CHAR;
ARCHIVO:FILE OF V;

```

```
(*-----Lectura de muestras del disco-----*)
```

```
PROCEDURE LEER_MUESTRAS_DEL_DISCO;
```

```
VAR
```

```

F:FILE OF V;
N:STRING;
V1:V;

```

```
BEGIN
```

```

CLRSCR;
WRITELN('ESCRIBE EL NOMBRE DEL ARCHIVO A LEER...');
READ(N);
ASSIGN(F,N);
CLRSCR;
RESET(F);
WHILE NOT EOF(F) DO
  BEGIN
    READ(F,V1);
    WITH V1 DO
      BEGIN
        WRITELN('CANAL N° ',CANAL,', MUESTRA N° ',DATO,'=',Vmuestras)
      END
    END;
  CLOSE(F);
  WRITELN;
  WRITE('PULSE INTRO...');
  READLN;
  READLN
END;

```

```
(*---Escribe en memoria dinámica las muestras con su N° de CANAL---*)
```

```
PROCEDURE OTRAMUESTRA(C,M:INTEGER;V:REAL);
```

```
BEGIN
```

```

CLRSCR;
V:=TENS+(2*TENS/4095)*(V-4095);
IF PRIMERO=NIL THEN
  BEGIN
    NEW(ACTUAL);
    ACTUAL^.CANAL:=C;
    ACTUAL^.Vmuestras:=V;
    ACTUAL^.DATO:=M;
    PRIMERO:=ACTUAL;
    ACTUAL^.SIG:=NIL
  END
ELSE
  BEGIN
    ANTERIOR:=ACTUAL;
    NEW(ACTUAL);
    ACTUAL^.CANAL:=C;

```

```

ACTUAL^.Vmuestras:=V;
ACTUAL^.DATO:=M;
ANTERIOR^.SIG:=ACTUAL;
ACTUAL^.SIG:=NIL
END
END;

(*-----Lee las muestras de memoria dinámica-----*)

PROCEDURE LEERMUESTRAS;

VAR
  MTRA:PUNTEROV;
  D:CHAR;
  NOM:STRING;

BEGIN
  WRITELN('DESEAS GUARDAR LOS DATOS OBTENIDOS EN DISCO (S/N)');
  D:=READKEY;
  CLRSCR;
  IF D='S' THEN
    BEGIN
      WRITELN('ESCRIBE RUTA Y NOMBRE DE FICHERO PARA LOS DATOS');
      READLN(NOM);
      ASSIGN(ARCHIVO,NOM);
      REWRITE(ARCHIVO);
      MTRA:=PRIMERO;
      WHILE MTRA<>NIL DO
        BEGIN
          WRITE(ARCHIVO,MTRA^);
          MTRA:=MTRA^.SIG
        END;
      CLOSE(ARCHIVO)
      END;

      MTRA:=PRIMERO;
      WHILE MTRA<>NIL DO
        BEGIN
          WITH MTRA^ DO
            WRITELN('CANAL N° ',CANAL,', MUESTRA N° ',DATO,',Vmuestras);
          MTRA:=MTRA^.SIG
        END
      END;
    END;
  END;

```

(*-----Programación de la Frecuencia-----*)

```

PROCEDURE FRECUENCIA (FRECUENCIA:INTEGER);
VAR D1,D2,AUX:LONGINT;
BEGIN
  AUX:=200000 DIV FRECUENCIA;
  D1:=ROUND(SQRT(AUX));
  D2:=D1;
  PORT[CONTCONTR]:=74;
  PORT[REGCONT1]:=D1 MOD 256;
  PORT[REGCONT1]:=D1 DIV 256;

```

```

PORT[CONTR]:=$B4;
PORT[REGCONT2]:=D2 MOD 256;
PORT[REGCONT2]:=D2 DIV 256
END;

(*-----Programa principal-----*)

BEGIN
CLRSCR;
Writeln('DESEAS VER ALGUN FICHERO CON DATOS DE LA PCLAB (S/N)');
T:=READKEY;
IF T='S' THEN
  REPEAT
  BEGIN
  LEER_MUESTRAS_DEL_DISCO;
  Writeln('Leer otro fichero C)continuar --*PULSA LA OPCION DESEADA');
  T:=READKEY
  END
  UNTIL T='C';

REPEAT
PRIMERO:=NIL; (*-----Inicializa el Puntero-----*)
CLRSCR;
Writeln('INTRODUCE LA FRECUENCIA DE MUESTREO: ');
READ(FRE);
PORT[REGCONTROL]:=0;
Writeln
('INTRODUCE EL N° DE CANALES CONECTADOS A LA PCLAB(0 A 15)= ');
READLN(CAN);
Writeln('INTRODUCE EL N° DE MUESTRAS: ');
READ(MAX);
Writeln('INTRODUCE EL VALOR DE LA TENSION MAXIMA EN LA PCLAB');
READLN(TENS);
PORT[REGCONTROL]:=6;
FRECUENCIA(FRE);
  FOR J:=0 TO CAN DO
    FOR I:=0 TO MAX-1 DO
      BEGIN
        PORT[CANALMUX]:=J;
        REPEAT FIN_CONVERSION:=PORT[DATOADH];
        UNTIL FIN_CONVERSION<=$F;
        DATO_BAJO:=PORT[DATOADL];
        DATO_ALTO:=PORT[DATOADH] MOD 16;
        MUEST:=DATO_ALTO*256+DATO_BAJO;
        OTRAMUESTRA(J,I,MUEST);
        REPEAT FIN_CONVERSION:=PORT[DATOADH];
        UNTIL FIN_CONVERSION>$F
      END;
    PORT[REGCONTROL]:=0;
    LEERMUESTRAS;
    Writeln(' ');
    Writeln('PULSAR CUALQUIER TECLA PARA REPETIR, ESCAPE PARA SALIR');
    TECLA:=READKEY
  UNTIL TECLA=#27;
  PORT[REGCONTROL]:=0

```

END.

4.3.1.1.3.- Disparo externo.

Por último debemos tener en cuenta una tercera forma de disparo : El disparo mediante una señal externa conectada al pin 5 del conector 1 de la placa.

Para habilitar este modo de disparo, hay que colocar el jumper JP1 en el modo TRG EXT (disparo externo. Ver conjunto del principio). Posteriormente elegiremos el modo de transferencia de los datos (por el registro de control), y operaremos de forma adecuada.

4.3.2.- CONVERSIÓN DIGITAL/ANALÓGICA.

Mediante este tipo de conversión obtenemos una señal analógica de salida variable según la elección del usuario programador. Dicha tensión analógica se consigue por uno de los dos canales de salida preparados para tal efecto. Previamente, habremos escrito la tensión digital (sobre 12 bits efectivos) que deseamos convertir en uno de las dos parejas de registros de datos de solo escritura que tiene asociado cada canal. Escribiremos el byte menos significativo primero, y luego el byte más significativo.

Las tensiones analógicas que podemos obtener varían según la tensión de referencia elegida. Así Para tensión de referencia de -5V obtenemos una tensión de salida

entre $\pm 5V$; y con tensión de referencia de $-10V$ obtenemos tensión de salida variable entre el rango de $\pm 10V$.

5.- DESCRIPCIÓN Y UTILIZACIÓN DE TURBO PASCAL

5.1.- INTRODUCCIÓN HISTÓRICA.

Resumir de forma breve los comienzos de la programación *PASCAL*, es introducirnos en el cambio producido por el paso a los lenguajes de alto nivel. Antes de la llegada de la programación estructurada, los lenguajes de bajo nivel (código máquina) e intermedios (Basic) colmaban las delicias de los programadores, quienes sin disponer de una herramienta potente con que dirigir sus programas, se enfrascaban delante de pequeños ordenadores consiguiendo sacar todo el provecho a pequeñas unidades de memoria.

En ese momento la programación orientada a interminables e indescifrables líneas de código llenaba por completo el quehacer de los profesionales en la programación. La programación así entendida era privilegio de unos pocos que, entendidos en la materia, se afanaban en defender las ventajas del lenguaje de bajo nivel.

Los lenguajes así conocidos estaban llegando al fin de su vida operativa; no porque no tuvieran ventajas, que si las tienen y muchas, si no porque con la extensión de la memoria en la aparición de potentes ordenadores personales (IBM PC, y compatibles) el mercado demandaba la necesidad de crear programas intermedios para el entendimiento entre el usuario y su ordenador, aún a costa de sacrificar espacio de memoria y velocidad de ejecución.

La historia nos remonta pues a 1971 cuando Niklaus Wirth escribió la definición de lenguaje *PASCAL*, en su libro *Pascal: User Manual and Report*. Este libro contenía una descripción detallada de lo que un lenguaje debe hacer.

Hay que establecer aquí la diferencia entre la definición de un lenguaje y la versión de dicho lenguaje.

La definición debe versar sobre lo que tiene que hacer dicho lenguaje introduciendo fórmulas eficientes para llevar a la práctica al mismo. La versión debe llegar más allá, combinando la definición y llenando las lagunas que la práctica puede encontrar en su camino de programación.

Sin embargo cuando Wirth escribió su definición no existía en ese momento ningún programa compilador real que lea un fichero fuente y produzca como resultado, a partir de él, un fichero de código ejecutable.

Desde ese momento aparecieron programas, primero para *mainframes* (como el CDC 6600), y luego para microordenadores.

Estos programas compiladores materializaban funcionalmente la definición del lenguaje Pascal tal como la expresó Wirth. Pero tal definición constaba de muchos puntos débiles que la condenaban a tener una aplicación limitada (no trataba el manejo de cadenas, los ficheros de acceso aleatorio, las llamadas al sistema operativo...).

Es entonces donde la realización de la versión del lenguaje propiamente dicha debe operar de forma autónoma, y muchos de estos programas comenzaron a poner limitaciones a los programadores que la definición de Wirth no tenía en cuenta.

Desde entonces hasta hoy en día la programación de alto nivel (Pascal, y demás lenguajes que conjugan instrucciones complejas con llamadas simples) no ha parado de variar y las primeras versiones de *TURBO PASCAL* pronto se quedaron obsoletas dando paso a nuevas versiones más actualizadas, más potentes y capaces de aprovechar al máximo todas las propiedades de los ordenadores personales.

Es importante recordar que Turbo Pascal tiene de Pascal lo que la programación estructurada hace de nuestros programas. La diferencia viene dada por los detalles. Y en informática, los detalles son importantes.

5.2.- LA PROGRAMACIÓN ESTRUCTURADA.

Pascal, por su diseño es un lenguaje estructurado.

A diferencia de otros lenguajes, impone a sus programas una estructura. Cada sentencia debe ir encadenada de una manera determinada, y el compilador se encargará de que ello haya sido dispuesto por el programador. Un programa debe ser codificado en ciertas partes. Cada una ha de ir en el lugar que le corresponde. Algunas cosas no pueden colocarse juntas, por el contrario, otras deben ir siempre asociadas.

El objetivo de esta insólita estructuración es potenciar cierta forma de programar. Esta forma de programar no es más que la materialización de las ideas de Wirth sobre la creación de programas que puedan ser comprendidos de forma sencilla por cualquier programador, sin necesidad de complicados y extensos diagramas de flujo. Esto es la programación estructurada; y aunque se puede utilizar en muchos lenguajes (Basic incluido), Pascal es uno de los pocos que exige su utilización.

La estructuración del Pascal es una programación que utiliza palabras sencillas inglesas (y aún falta mucho para que podamos programar en castellano) formadas a su vez por caracteres ASCII. Hay un número pequeño de palabras reservadas (palabras con significado especial en la programación Pascal : ABSOLUTE, AND, ARRAY...), un número mayor de palabras denominadas identificadores estándar y un número ilimitado de identificadores creados por el programador.

5.3.- ESTRUCTURA BÁSICA DE LA PROGRAMACIÓN EN TURBO PASCAL.

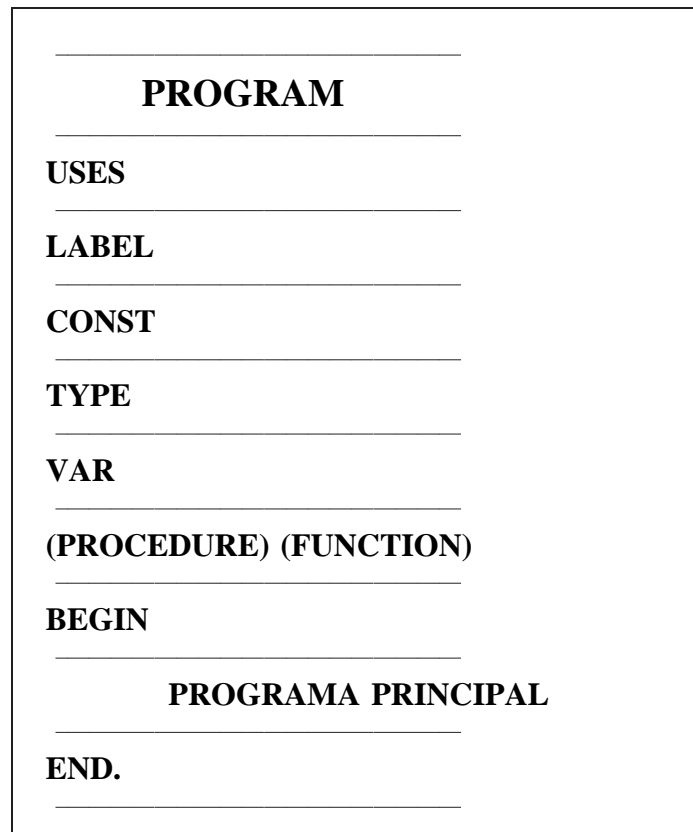
5.3.1.- Estructura de un programa Pascal.

Todo programa en Pascal, y por ende, todo programa en Turbo Pascal, debe de estar constituido por una serie de elementos o instrucciones básicas que nos definen la realización del mismo. En su estructuración está el secreto de la definición, y por tanto, será el formato de programación el que nos determine la utilidad del mismo.

En este apartado vamos a descubrir como debe ser el contenido de un programa para que cada cosa se sitúe en el sitio que le corresponde; para ello el programa contendrá todas las sentencias, expresiones y declaraciones que explicaremos en apartados posteriores, sin embargo conviene tener claro la estructura antes de introducirnos en este campo.

El programa debe construirse a partir de elementos sencillos de una forma particular. Esta planificación se hace para facilitar la comprensión de los programas de una forma clara y estructurada.

La figura siguiente contiene un diagrama de lo que acontece al conjunto de un programa. No es imprescindible que su programa conste de todos los apartados que se mencionan para su correcto funcionamiento, pero la utilización de los mismos facilita la elaboración y comprensión del mismo.



Estructura de un programa Turbo Pascal

A continuación desarrollaremos una explicación de cada uno de los elementos que lo componen, con el propósito de complementar el diagrama.

5.3.1.2.- La sentencia *PROGRAM*.

Asocia un nombre al programa y avisa al compilador que su trabajo comienza aquí.

5.3.1.2.- La declaración de unidades (sentencia *USES*).

Turbo Pascal difiere del Pascal estándar en que permite la compilación independiente. Con ella podemos desarrollar el trabajo de nuestro programa principal utilizando subprogramas (unidades), mediante procedimientos y funciones compilados anteriormente.

Su utilización puede omitirse si no se desea utilizar ningún tipo de unidad, pero si se declara, debe ser lo inmediato después de la sentencia *program*.

5.3.1.3.- La declaración de etiquetas (sentencia *LABEL*).

A continuación viene la declaración de etiquetas del programa. Mediante la palabra reservada **LABEL**, seguida por las etiquetas convenientes, declaramos los nombres de los lugares a los que una vez situadas en el programa podemos realizar un salto de ejecución por las sentencias **GOTO**.

Si un programa no contiene sentencias **GOTO**, no precisa de declaración de etiquetas. Hay que hacer incapié en la necesidad de no alimentar el programa con muchas sentencias **GOTO** que oscurecen su desarrollo y comprensión; pero habrá veces en que su inclusión será inevitable.

5.3.1.4.- La declaración de constantes (*CONST*).

Si se van a utilizar constantes en un programa, el compilador espera

encontrarlas a continuación de la declaración de etiquetas. Las constantes son valores que se definen durante el tiempo de compilación y permanecen invariables durante la ejecución del programa. Declarar una constante consiste en especificar el identificador de esa constante y su valor, separados por el signo "=".

5.3.1.5.- La declaración de los tipos (*TYPE*).

A continuación de la declaración de constantes del programa aparece la declaración de tipos.

Esta declaración la realiza el programador a su conveniencia para la creación de variables formadas por variables estándar, o por variables elegidas y creadas por el propio programador.

5.3.1.6.- La declaración de variables (*VAR*).

Las variables determinan elementos que en la ejecución del programa pueden cambiar su valor de forma arbitraria o controlada por el propio programa.

5.3.1.7.- Definición de procedimientos y funciones (*PROCEDURE, FUNCTION*).

Después de la definición de las variables que va a utilizar el programa, se han de definir los procedimientos y funciones que no se incorporen en las unidades declaradas

mediante la sentencia **USES**.

Los procedimientos y las funciones son subrutinas en Pascal que pueden ser llamadas en cualquier momento de la ejecución del programa realizando una operación que acorta el tamaño del fichero a compilar y hace transparente la comprensión del programa.

La declaración de los procedimientos y funciones debe seguir un orden bien establecido para poder realizar sus respectivas llamadas. Un procedimiento (o función) puede llamar a otro que se haya declarado antes, pero no al revés. Esto implica que la posición de un procedimiento (o función) respecto a otro establece una jerarquía de llamadas.

5.3.1.8.- El programa principal.

Encerrado entre las sentencias **BEGIN-END** se halla el cuerpo del programa principal.

Cuando el ordenador empieza a ejecutar el programa compilado, no comienza la ejecución por el principio del fichero fuente, sino por el **BEGIN** que marca el comienzo del programa principal. Cuando encuentra el **END** correspondiente, abandona el programa y devuelve el control al sistema operativo. Mientras a realizado todo el trabajo que tenía a su cargo.

5.3.2.- Identificadores.

El ordenador se utiliza para crear programas. Los programas son colecciones de cosas (datos) e instrucciones que indican como almacenar, modificar y presentar estas cosas (sentencias). El ordenador accede a todas esas cosas mediante su dirección en memoria. Los nombres en lenguaje común que damos a los datos, programas, funciones y procedimientos. Es el conjunto de identificadores.

Los identificadores aparecen en el fichero fuente pero en ningún caso en el fichero de código.

Los identificadores deben cumplir las siguientes reglas:

1. Deben ser secuencias de 127 caracteres de longitud máxima.
2. Los caracteres legales incluyen letras, dígitos y subrayados. No es permitido espacios y símbolos (\$,&!,*,%,¡...).
3. Los identificadores deben comenzar con una letra, nunca con un dígito (0..9).
4. La diferencia entre mayúsculas y minúsculas es ignorada.
5. Los subrayados son legales y significativos, y la longitud del indicativo es siempre tomada en cuenta por el compilador.

5.3.3.- Tipos de datos :

Con Turbo Pascal el conjunto de los tipos de datos es muy grande, y cada uno de los cuales sirven para un propósito específico.

Una variable de tipo **Byte** ocupa un byte, y es un valor numérico sin signo que abarca un rango que va desde 0 hasta 255.

Al igual que la variables byte, las variables **Integer** albergan valores numéricos sin signo. Pero al tener dos bytes de longitud pueden variar su valor desde -32.768 a 32767.

Las variables **LongInt** ocupan cuatro bytes en memoria y tienen un rango de -2.147.483.648 a 2.147.483.647.

Para números con parte decimal o con magnitudes que excedan el valor máximo positivo del LongInt, Turbo Pascal proporciona el tipo de datos **Real**, conocido como tipo de coma flotante. Una variable Real requiere seis bytes de almacenamiento, y su rango puede variar de $2,9 \cdot 10^{-39}$ hasta $1,7 \cdot 10^{38}$.

Al igual que el tipo Byte, el tipo de datos **Char** (carácter) ocupa un byte de memoria. Sin embargo este tipo se utiliza para comparar texto.

El tipo de datos **String** guarda información de texto. Puede tener una longitud de 1 a 255 caracteres, y por tanto la cantidad de memoria de almacenamiento que utiliza dependerá de la extensión que decida el programador.

Además de los datos anteriores, Turbo Pascal soporta otros cuatro tipos de datos numéricos : **Single**, **Double**, **Extended** y **Comp**.

El Single es un Real corto (4 bytes).

El Double es un Real largo (8 bytes).

Y el Extended es el mayor de todos (10 bytes).

El Comp utiliza 8 bytes, y no es de coma flotante.

Estos últimos cuatro tipos son de base especial para el coprocesador de la familia 87; pero no debe preocuparse si no posee alguno, porque Turbo Pascal permite el acceso a todos ellos con el modo de Emulación.

El **Registro** (Record) es una combinación de otros tipos de datos en un nuevo tipo de datos, que se accede mediante campos.

Por último trataremos de forma breve el **Array** (matriz). Un Array es una variable que repite un tipo de datos un número especificado de veces; el formato que utiliza es el siguiente :

Array[Limitinferior..LimitSuperior] Of Tipo de Datos;

Aquí el Array es unidimensional, pero puede tener tantas dimensiones como memoria disponible poseamos para asignarle.

5.3.4.- Estructuras de control y decisión :

La forma más sencilla de bifurcación condicional es la sentencia **If-Then** (Si .. Entonces), que hace que se ejecute una instrucción o bloque de programa si se cumple cierta condición.

Lo primero que realiza esta sentencia es evaluar la información que se le ha proporcionado en forma de sentencia booleana. La evaluación produce uno de los dos posibles resultados (cierto-falso). Si la sentencia es cierta, el programa ejecuta el bloque de código que está inmerso después de la sentencia **If-Then**. Si el resultado es falso, el programa se salta dicho bloque.

Una variación de la anterior es la estructura **If-Then-Else** (Si .. Entonces .. Si No ..). La metodología de funcionamiento es la misma, pero incluye la propiedad de que si la evaluación booleana a resultado falsa, se ejecutará el bloque de código que es precedido por la sentencia **Else**.

Si se utilizan a menudo tipos simples de datos en los programa, se puede utilizar la sentencia **Case** en lugar de **If-Then**. **Case** proporciona una estructura lógica clara para bifurcación múltiple.

La sentencia también realiza una evaluación booleana, pero a diferencia, permite que las bifurcaciones de los bloques de código sean ilimitados.

En un programa tenemos que tener en cuenta las estructuras de control repetitivas.

El bucle **For-Do** es un ejemplo de ello. Al realizar un conjunto de instrucciones que abarque el bucle, debe de especificarse un punto inicial, y un punto de comienzo; el conjunto se ejecutará el número de veces indicado por inicio y comienzo.

Por ejemplo :

For i:=1 To 100 Do

Begin

Sentencias

End;

El bucle **Repeat-Until** es una notable mejora del anterior. Aquí el conjunto se ejecuta hasta que una condición booleana sea cierta.

Por ejemplo :

Repeat

Sentencias

Until (condición booleana);

El bucle **While-Do** es similar al Repeat-Until, exceptuando que el While-Do examina Una condición booleana antes de ejecutar cualquier sentencia o bloque de sentencias.

Por ejemplo :

While (condición booleana) Do

```
Begin  
Sentencias  
End;
```

Por último trataremos la bifurcación no condicionada. Se trata de los saltos **Goto** hacia etiquetas (**Label**).

La sentencia *Goto* realiza el salto de programa al punto indicado por una etiqueta declarada en la sentencia *Label*.

Por ejemplo :

```
...  
Label 1;  
...  
Sentencias  
...  
1 :  
Sentencias  
...  
Goto 1;  
...  
Sentencias  
...  
End.
```

5.3.5.- Lista de instrucciones.

En este apartado veremos los distintos tipos de instrucciones que configuran la utilización óptima de Turbo Pascal.

Únicamente nombraremos las instrucciones por orden alfabético, profundizando de forma limitada en aquellas que pueden considerarse más interesantes. Será a partir de aquí cuando el lector recurrirá a la bibliografía recomendada para conseguir más información al respecto.

Instrucciones :

ABS

Función que devuelve el valor absoluto del parámetro que se le pasa.
 sintaxis : Function Abs(r:Real) : Real;
 Function Abs(i:Integer) : Integer;

ADDR

Addr devuelve la dirección de una variable, constante con tipo, o procedimiento. El resultado es un puntero (**Pointer**).
 Sintaxis : Addr(Var Variable) : Pointer

APPEND

Append abre un archivo de texto para escritura colocando el puntero del archivo al final del mismo. Con la instrucción escribiremos en el archivo sin perder el contenido anterior.
 sintaxis : Procedure Append(Var F : Text);

ARC {Unidad Graph}

Arc dibuja un círculo alrededor de unas coordenadas, con radio determinado, y dibujándose desde un ángulo inicial hasta un ángulo final. (En el sentido de las agujas del reloj).

Sintaxis : Procedure Arc (X,Y : integer; AngIni,AngFin, Radio : Word);

ARCTAN : Función que devuelve el valor del arcotangente que se le pasa como parámetro (generalmente Real).

ASSIGN

Assing enlaza una variable de archivo **F** al archivo indicado mediante **Nombre**. Es preciso utilizar éste procedimiento para leer o escribir sobre un archivo de disco, ya que en el programa se enlazará un archivo lógico a uno físico.

Sintaxis : Procedure Assign(Var F : File; Nombre : String);

ASSIGNCRT {Unidad CRT} : Procedimiento que permite al usuario enviar información al monitor de vídeo mediante la escritura en el archivo **F**. (Var F : Text);

BAR {Unidad GRAPH}

En pantalla gráfica, el procedimiento nos crea un rectángulo relleno, con el patrón actual de relleno, según las coordenadas de vértice superior izquierdo e inferior derecho.

Sintaxis : Procedure Bar(X1,Y1,X2,Y2);

BAR3D {Unidad Graph} : Dibuja en pantalla un área rectangular de coordenadas determinadas, con profundidad y altura escogida.

BLOCKREAD

Procedimiento de lectura de un número determinado de registros de un archivo sin tipo en el buffer. **RegistrosLeídos** indica el número de registros que se han leído en realidad.

Sintaxis : Procedure BlockRead (Var F : File; Var B : Tipo; NumeroRegistros : Integer; Var RegistrosLeídos : Integer);

BLOCKWRITE

Procedimiento que graba un número de registros del Buffer **B** en el archivo sin tipo **F**. Es utilizado, junto con el procedimiento anterior, para copiar archivos entre sí.

Sintaxis : Procedure BlockWrite (Var F : File; Var B : Tipo; NumRegs : Integer);

CHDIR

ChDir cambia el directorio actual de trabajo por el indicado en una variable string.
Sintaxis : ChDir(Direc : String);

CHR : Nos devuelve el carácter ASCII correspondiente al entero (Integer) que se le pasa.

CIRCLE {Unidad GRAPH} : Nos dibuja un círculo en las coordenadas determinadas, de radio expuesto.

CLEARDEVICE {Unidad GRAPH} : Como procedimiento borra la pantalla actual de gráficos.

CLEARVIEWPORT {Unidad GRAPH} : Como procedimiento borra la ventana gráfica actual.

CLOSE

Close vacía el buffer del archivo liberando el manejador de archivo que se estaba utilizando; a su vez cierra el archivo correspondiente.
Sintaxis : Procedure Close(Var F : File);

CLOSEGRAPH {Unidad GRAPH} : Procedimiento que restaura el monitor de vídeo al modo en que se encontraba antes de realizar la llamada a gráficos.

CLREOL {Unidad CRT} : Nos borra la línea actual de la pantalla desde la posición del cursor hasta el extremo derecho de la pantalla.

CLRSCR {Unidad CRT} : Procedimiento que borra la pantalla y sitúa el cursor en la posición inicial de la misma. (1,1).

CONCAT : Función que concatena un número cualquiera de cadenas y las devuelve en una sola. Si la longitud de la cadena concatenada es mayor que 255, se generará un error en tiempo de ejecución.

COPY : Devuelve una parte de una cadena contando a partir de cierto número, determinados caracteres. Function Copy(S : String; P,L : Integer) : String;

COS : Cos nos devuelve el coseno de el real (Real) que se le pasa.

CSEG : Devuelve la dirección del segmento del código del programa (mediante variable de tipo Word).

DEC : Procedimiento que decrementa una variable escalar (Scalar) en N valores (LongInt).

DELAY : Procedimiento de retardo de milisegundos en la ejecución del programa.

DELETE

Delete elimina **N** caracteres de la cadena **C** comenzando por el carácter **P**.
Sintaxis : Procedure Delete(C : String; N,P : Integer);

DISKSIZE {Unidad DOS}

DiskSize es una función que nos devuelve el tamaño, expresado en bytes, de la unidad (1=A, 2=B, 0=Unidad por defecto).
Sintaxis : Function DiskSize(Unidad : word) : LongInt;

DISPOSE

Dispose nos libera la memoria del montón asignada a una variable de tipo puntero (Pointer). Se usa en conjunto con New.
Sintaxis : Procedure Dispose(P : Pointer);

DOSVERSION {Unidad DOS} : Devuelve el código de salida de un subprocesso (0=terminación normal, 1=terminación por Ctrl-C, 2=terminación por error en dispositivo, 3=terminación mediante el procedimiento **Keep**).

EOF

Como función, devuelve TRUE cuando el puntero alcanza el final del archivo.
Sintaxis : Function Eof(F : File) : Boolean;

EOLN

Eoln devuelve TRUE cuando el puntero del archivo F alcanza el fin de línea o el fin de archivo.

Sintaxis : Function Eoln(F : File) : Boolean;

EXEC {Unidad DOS}

Este procedimiento ejecuta el archivo especificado en el encaminamiento, con los parámetros de línea que se le quieran pasar.

Sintaxis : Procedure Exec(Encaminamiento, Línea de órdenes : String);

EXIT

Procedimiento que hace que el programa salga del bloque que se está ejecutando en ese momento.

Sintaxis : Procedure Exit;

FILESIZE

FileSize devuelve el número de registros que actualmente contiene el archivo.

Sintaxis : FileSize(F : File) : Integer;

FILLCHAR

Procedimiento que rellena un número determinado de bytes de memoria con el valor escalar comenzando en la dirección elegida.

Sintaxis : Procedure FillChar(Variable : Tipo; I, Código : Scalar);

FINDFIRST {Unidad DOS}

FindFirst devuelve información del primer archivo encontrado en el directorio señalado cuyos atributos coinciden con **Attrib**.

Sintaxis : Procedure FindFirst (Encaminamiento : String; Attrib : Word; Var S : SerchRec);

FINDNEXT

FindNext devuelve información del siguiente archivo encontrado en el directorio especificado, definido en **FindFirst** cuyos atributos coinciden con los de **FindFirst**. Si la búsqueda se produce con éxito, el valor de DosError será cero.

Sintaxis : Procedure FindNext(Var S : SearchRec);

FREEMEM

Procedimiento que libera I bytes del montón de memoria asociado con una variable de tipo puntero, que se debe haber asignado con GetMem.

Sintaxis : Procedure FreeMem(Var P : Pointer; I : Integer);

GETCOLOR {Unidad GRAPH} : Devuelve el color del dibujo actual en el modo gráfico.

GETDATE {Unidad DOS} : Devuelve la fecha determinada mediante el reloj del sistema, sobre unas variables (año, mes, día, díasemana) de tipo Word.

GETDIR : Obtiene el directorio de la unidad especificado por U. El directorio se devuelve en una variable cadena. Si U es cero, GetDir busca en la unidad por defecto.
GetDir(U : Byte; Var S : String);

GETGRAPHMODE {Unidad GRAPH}

Función que devuelve el modo gráfico actual. El valor numérico del modo gráfico se deberá interpretar conjuntamente con la información sobre el controlador de gráficos usado.

Sintaxis : Function GetGraphMode : Integer;

GETIMAGE {Unidad GRAPH}

GetImage guarda en una variable una porción rectangular de una pantalla gráfica, delimitada por sus coordenadas.

Sintaxis : Procedure GetImage(X1,Y1,X2,Y2 : Integer; Var MapaBits);

GETMAXMODE {Unidad GRAPH}

Nos devuelve un valor que indica el modo gráfico de mayor resolución de la tarjeta gráfica que se encuentra instalada.

Sintaxis : Function GetMaxMode : Word;

GETMAXX {Unidad GRAPH}

GetMaxX devuelve la máxima coordenada horizontal del modo gráfico actual.

Sintaxis : Function GetMaxX : Integer;

GETMAXY {Unidad GRAPH}

Devuelve la máxima coordenada vertical del modo gráfico actual.

Sintaxis : Function GetMaxY : Integer;

GOTOXY {Unidad CRT}

Es el procedimiento que ,nos sitúa el cursor en las coordenada de la pantalla señaladas.

Sintaxis : Procedure GotoXY(X,Y : Integer);

GRAPHRESULT {Unidad GRAPH}

Devuelve un código de error relativo al último procedimiento de gráficos utilizado.

Sintaxis : Function GraphResult : Integer;

HALT

Halt termina la ejecución de un programa.

Sintaxis : Procedure Halt;

IMAGESIZE {Unidad GRAPH}

Es la función que devuelve el número de bytes requerido para almacenar el mapa de bits de la porción de la pantalla definida por sus coordenadas.

Sintaxis : Function ImageSize(X1,Y1,X2,Y2 : Integer);

INITGRAPH {Unidad GRAPH}

Para inicializar la entrada al modo gráfico utilizaremos este procedimiento, que busca, según un encaminamiento, los ficheros .BGI de los controladores gráficos. Si controlador es cero, el procedimiento detecta automáticamente el adaptador de gráficos disponible, ajustando la máxima resolución.

Sintaxis : Procedure InitGraph (Var Controlador : Integer; ModoGráfico : Integer; Encaminamiento : String);

INSERT : Inserta la cadena fuente en la posición determinada de la cadena destino.
Procedure Insert(Fuente : String; Var Destino : String; Indice : Integer).

INTR {Unidad DOS} : Llamada a la interrupción del BIOS.

IORESULT

= phantom 5 F_RMS over F_AV^CX59w

Nos informará del código de error ocurrido al realizar una operación r0