2 ej. 30

Universidad Nacional Autónoma de México

Facultad de Ciencias

"Diseño de un Sistema Microprocesador para Transformada Rápida de Fourier"

T E S I S

Que para obtener el Grado de:

F I S I C O

P r e s e n t a :

LORENZO JUAREZ PALAFOX

México, D. F. Julio de 1986

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

I N D I C E

APENDICES

BIBLIOGRAFIAS

R E C O N O C I M I E N T O S

La labor de desarrollar un trabajo es un esfuerzo que no solo es merito del que lo desarrolla, sino tambien de las personas que de alguna forma ya se directa o indirecta aportan ideas o critican para la realización de este, por lo que deseo expresar mi gratitud a:

Dr. David Romeu cacajuana, director de la presente tesis, por su apoyo y orientación.

Dr. Miguel Jose Yacaman, director del instituto de fisica, por darme todas las facilidades para desarrollar este trabajo.

Dr. Alfredo Gomez Rodriguez, por sus criticas constructivas y gran apoyo.

Dr. Jose Gpe. Pere Ramirez, por la gran aportación de ideas durante la realización de mi trabajo.

Fis. Raul Espejel Paz, por sus grandes comentarios y sugerencias para darle la presentación final a la tesis.

Por otra parte quiero agradecerle muy especialmente al fisico Fernando Angeles del Instituto de Astronomia por su explicaciones y aclaraciones del lenguaje pascal y sistemas digitales.

De igual forma quiero agradecer a las siguientes pesonas Dr. pablo schabes por sus comentarios y al Maestro Jose Reyes Gasga y a los fisicos Alberto fuentes Maya y Roberto Hernandez Reyes por sus sugerencias.

CAPITULO I INTRODUCCION

I.1 Que es una imagen digital?

Siempe que se habla de una imagen monocromatica o simplememte de imagen nos estamos refiriendo a una función f(x,y) bidimensional de la intensidad de luz, donde x e y son coordenadas espaciales y el valor de f en algun punto (x,y) es proporcional a la brillantez (o nivel de gris) de la imagen en el punto. Ya que esta función es una función continua se podra intuir facilmente que una IMAGEN DIGITAL es una función f(x,y) la cual ha sido discretizada en sus coordenadas espaciales y en brillantez.

Podemos considerar una imagen digital como una MATRIZ cuyos renglones y columnas identifiquen un punto en la imagen y el valor del correspondiente elemento de la matriz identifique un nivel de gris del punto. Los elementos de tal arreglo matricial son llamados elementos de imagen, elementos del cuadro o escena, pixeles o pels. Estás dos ultimás son nombres abreviados de los terminos ingleses (picture elements).

Un sistema básico para hacer tratamiento de imagenes digitales de proposito general, contendrá los elementos mostrados en la figura #1. La operación de todo el sistema puede ser dividida en tres partes principales: digitalización, procesado y desplegado, cuyas cracteristicas son:

a).- DIGITALIZADORES

la función de un digitalizador es la de convertir una imagen f(x,y) continua en una representación numerica que automaticaemnte se convierte en la entrada ideal para una computadora digital. Para lograr esto un digitalizador tener cinco elementos. El primero es una abertura de muestreo, la cual permite al digitalizador tener acceso solamente a un pixel ignorando el resto de la imagen. El segundo elemento de un digitalizador es un dispositivo para muestrear la imagen, este proceso consiste en mover la abertura de muestreo sobre un patrón pedeterminado. El muestreador permite a la abertura de muestreo poder direccionar cada pixel.

El tercer elemento es un detector el cual puede medir el brillo de la imagen en cada pixel. El detector es comunmente un transductor que convierte intensidad de luz en voltaje o corriente eléctrica. El cuarto elemento es un cuantizador en un circuito electrónico llamado convertidor analógico a digital, esta unidad produce un número que es proporcional al nivel de voltaje o corriente de entrada.

El quinto elemento de un digitalizador de imágenes es el medio de salida, los valores del nivel de gris producidos por el cuantizador deben ser almacenados en un formato apropiado para su procesamiento computacional. El medio de salida puede ser un disco o cinta magnética, un banco de

memoria digital, etc.. Algunos ejemplos de estos sistemas pueden ser microdensitrometros, camaras de TV digitalizadoras etc.

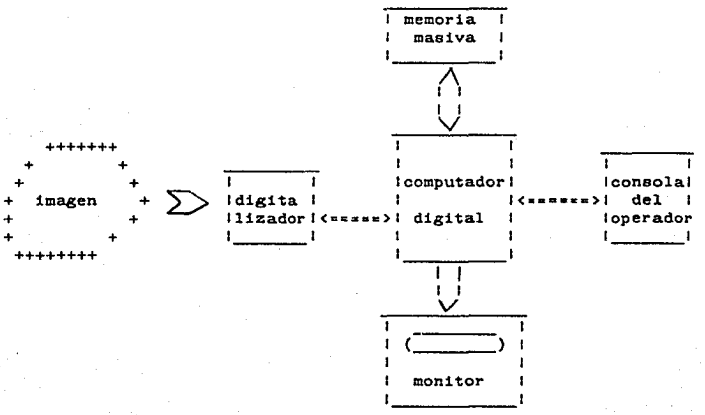


figura #1 Elementos de un sistema de procesado de imagenes digital.

b).- PROCESADOR DE IAMGEN

Los sistemas usados para procesar la imagen digitalizada se pueden clasificar desde dispositivos microprocesadores para aplicaciones de proposito general hasta grandes sistemas de

computación capaces de realizar una gran variedad de funciones sobre arreglos de imagen de alta resolución.

c).- DISPOSITIVOS DE DESPLIEGUE

La función de la unidad de despliegue en un sistema de procesamiento de imagenes, es la de convertir el arreglo numerico almacenado en la computadora en una forma entendible para la interpretación humana.

En sistemas de CRT las posiciones horizontal y vertical de cada elemento en el arreglo de la imagen digital son convertidas en voltajes, los cuales son usados para deflectar el rayo de electrones del CRT y la intensidad del rayo es proporcional al valor del punto en el arreglo número.

Como hicimos notar al principio, la función f(x,y) denota la brillantez de la imagen en un punto y siendo que la luz es una forma de energia, la función f(x,y) no debe ser menor que cero y además debe ser acotada, esto es;

$$\emptyset < f(x,y) < Fmax$$

Por otra parte para hacer comprensible una imagen a una computadora digital, una función f(x,y) imagen puede ser digitalizada espacialmente y en amplitud. A la digita lización de las

coordenadas espaciales (x,y) se le llama muestreo de la imagen, mientras que al digitalización de la amplitud se le conoce como cuantización del nivel de gris.

Si una imagen continua es aproximada por un muestreo equidistante en la forma de un arreglo N X N como se muestra en la ecuacin #1, el lado derecho de esta ecuación representa lo que es comunmente llamada una imagen digital, donde cada elemento del arreglo es llamado elemento de la imagen, elemento de cuadro o escena, pixel o pel.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(N-1,0) & f(N-1,1) & \cdots & f(n-1,N-1) \end{bmatrix}$$

ec. (1)

El proceso de digitalización descrito arriba requiere de tomar una decisión sobre el valor de N, así como sobre el número de niveles de gris elegido para cada pixel. Una practica común en procesado de imagenes digitalizadas es hacer que estas cantidades sean potencias enteras de 2, esto es;

$$N = 2^{n} \qquad \text{ec. (2)}$$

y

$$G = 2^{m} \qquad \text{ec. (3)}$$

donde G representa el número de niveles de gris, y N el número de muestras por renglon y por columna del arreglo. Usando las ecuaciones (2) y (3) se puede calcular el número de bits (b) requeridos para almacenar una imagen digitalizada, esto es dado por la siguiente expresión;

$$b = N \times N \times m \qquad \text{ec. (4)}$$

por ejemplo, una imagen de 128 X 128 elementos con 64 niveles de gris requiere 98304 bits de almacemamiento. La tabla #1 resume los valores de b para algunos valores tipicos de N y m, y la tabla #2 da los correspondientes números de 8 bits o bytes. La resolución (el rango de detalle distinguible) de una imagen es dependiente de ambos parametros N y m.

N\m	1	2	3	4	5	6	7	8
32	1024	2048	3072	4096	5120	6144	7168	8192
64	4096	8192	12288	16384	20480	24576	28672	32768
128	16384	32768	49152	65536	81920	98304	114688	131072
256	65536	131072	196608	262144	327680	393216	458752	524288
512	262144	524288	786432	1048576	1310720	1572864	1835008	2097152

tabla #1 número de bits de almacenamiento para varios valores de N y m

N\m	1	2	3	4	5	6	7	8
32	128	256	384	512	640	768	896	1,024
64	512	1,024	1,536	2,048	2,560	3,072	3,584	4,096
128	2,048	4,096	6,144	8,192	10,240	12,288	14,336	16,384
256	8,192	12,536	24,476	32,768	40,960	49,172	57,344	65,536
512	32,768	65,536	98,304	131,072	163,940	196,608	229,376	262,144

tabla #2 número de bytes de almacenamiento para varios valores de N y m

I.2 Procesamiento de imágenes

Después de que se ha obtenido la imagen digitalizada por algún medio, el siguiente paso es procesarla , en todas las aplicaciones de procesamiento de imágenes el objetivo es extraer información de la imagen. Obtener la información deseada puede requerir filtraje, transformación, análisis interactivo u otros metodos.

Podemos considerar casi todos los trabajos de procesamiento de imágenes en una de las siguientes categorias:

1).-Mejoramiento de la imagen.- Este metodo sirve simplemente para mejorar la imagen a ser vista por sistemas de interpretación visual (hombres o maquinas). Los tipos de mejoramieto de la imagen incluyen operaciones del tipo de ajuste de contraste, filtraje para aminorar el ruido, aplicaciones de pseudocolor y muchas otras.

2).-Recuperación de la imagen.- Esta tecnica esta dirigida a eliminar degradaciones de la imagen producidas por el proceso de formación. Por ejemplo eliminar trazo borroso de uma imagen causado por movimientos de la cámara, imagen fuera de foco o turbulencias atmósfericas, todos estos son problemas de recuperación de imagen.

3).- Extracción de caracteristicas y reconocimiento.- En muchos trabajos de procesamiento de imágenes es importante caracterizar

objetos de una escena o identificarlos, esto se logra utilizando patrones aproximados de reconocimiento de objetos especificos, como un ejemplo donde es útil esta tecnica es en la identificación de celulas mutantes en una muestra biológica.

4).- Codificación de la imagen.- Esta tecnica envuelve la compresión de la imagen en una forma alternativa (codificada) para almacenamiento y/o requerimentos de transmisión.

Para los tres primeros incisos la transformada rápida de FOURIER "FFT" resulta indispensable, ya que se pueden aplicar una gran variedad de tecnicas como son: filtrado de altas y bajas frecuencias, pseudocoloración, etc.. Y dado que el proceso es muy lento por la cantidad de operaciones que se requieren, es necesario implementarla de manera rápida.

Es el objetivo de este trabajo diseñar un sistema que extraiga la transformada de FOURIER rápidamente (lenguaje de maquina), que sea aplicable a cualquier sistema de cómputo y que no distraiga a la unidad central mientras se realiza este cálculo.

I.3 Separabilidad de las transformadas de FOURIER bidimensionales

Una de las propiedades más importantes de las transformadas de FOURIER bidimensionales es la separabilidad. Esto es, si tomamos el muestreo de una imagen en un arreglo cuadrado de N X N entonces podemos escribir la transformada discreta bidimensional como sigue:

$$F(u,v) = 1/N\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} (f(x,y)\ \exp[-2i(ux + vy)/N]) \qquad \text{ec. (5)}$$

y para u,v = 0, 1, . . . , N-1

$$f(x,y) = 1/N\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} (F(u,v)\ \exp[2i(ux + vy)/N]) \qquad \text{ec. (6)}$$

Estas ecuaciones se pueden expresar en la forma separable

$$F(u,v) = 1/N\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} (\exp[-2iux/N]f(x,y)\exp[-2ivy/N]) \qquad \text{ec. (7)}$$

y para u,v = 0, 1, . . . , N-1

$$f(x,y) = 1/N\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} (\exp[2iux/N]F(u,v)\exp[2ivy/N]) \qquad \text{ec. (8)}$$

Para nuestro propósito, la principal ventaja de la propiedad de separabilidad es que F(u,v) o f(x,y) pueden ser obtenidas en dos pasos por aplicaciones sucesivas de la transformada de FOURIER unidimencional o su inversa. Esto es evidente si identificamos la ecuación (7) expresandola en la forma siguiente

$$F(u,v) = 1/N \sum_{x=0}^{N-1} (F(x,v)\ \exp[-2iux/N]) \qquad \text{ec. (9)}$$

donde

$$F(x,v) = N[\ 1/N \sum_{y=0}^{N-1} (f(x,y)\ \exp[-2ivy/N])\ ] \qquad \text{ec. (10)}$$

Para cada valor de x, la expresión dentro de los corchetes en la ecuación (10) es una transformada unidimencional con valores de la frecuencia v = 0, 1, . . . , N-1. Por lo tanto la

función bidimencional F(x,v) es obtenida por la ejecución de una transformada a lo largo de cada renglon de f(x,y) y multiplicando el resultado por N. El resultado deseado F(u,v) es entonces obtenido por la ejecución de una transformada a lo largo de cada columna de F(x,v), como es indicado en la ecuación (9). El proceso es ilustrado gaficamente en la figura #2. Como se puede notar, obtiene el mismo resultado si aplicamos una transformada a lo largo de cada columna de f(x,y) y despues a lo largo de cada renglon del resultado obtenido. Esto es facilmente demostrable reinvirtiendo el orden de las sumatorias de la ecuación (7), y haciendo lo mismo para la ecuación (8).

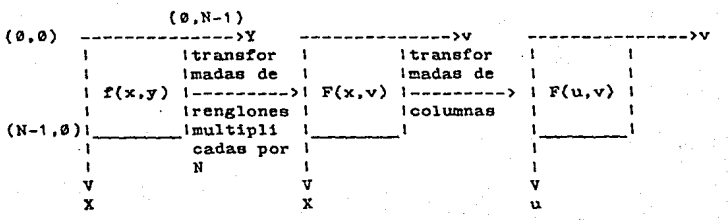


figura #2 aplicación de la transformada unidimencional sobre un arreglo bidimencional

CAPITULO II DESCRIPCION DEL SISTEMA.

II.1 REQUERIMENTOS DEL SISTEMA

El sistema a diseñar es el enmarcado dentro de la linea punteada de la fig.#3, como se puede observar se necesita diseñar un computador dedicado que realice lo siguiente:

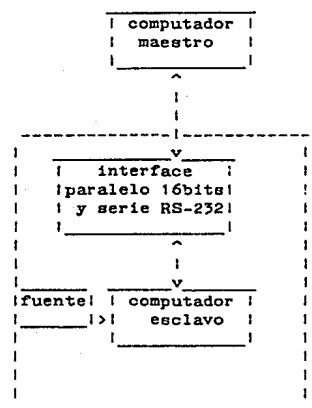


fig. #3

1.- Una interfaz que realice el protocolo de comunicación entre el computador maestro y el computador esclavo.

2.- El computador debe ser capaz de recibir dos matrices de datos de 512 X 512 elementos real e imaginaria lo que obliga al sistema a poseer una cantidad de memoria RAM de 524KB más la cantidad de memoria para ejecutar todas las subrutinas y programas de inicialización del sistema.

3.- por medio de un programa principal se debe de obtener la transformada rapida de fourier de la matriz y devolver el resultado al computador maestro.

Para realizar este diseño se eligió un microprocesador de Motorola MC68000 el cual tiene una arquitectura de propósito general, y tiene las siguientes caracteristicas:

1.- 8 registros de datos de 32 bits (Dn).
2.- 8 registros de direcciones de 32 bits (An).
3.- Un rango de 16 Mbytes de direccionamiento directo.
4.- Contador de programa (PC) de 32 bits.
5.- Registro de estados (SR) de 16 bits.
6.- 56 tipos de instrucciones.
7.- Operaciones en cinco tipos de datos.
8.- Memoria mapeada de entrada/salida.
9.- 14 modos de direccionamiento.

Hay cuatro versiones que difieren en la frecuencia de trabajo (4,6,8,10MHz), las cuales tienen periodos de reloj de 250, 167, 125 y 100ns. La instrucción más rapida (copiar un registro en otro) se ejecuta en cuatro periodos de reloj o 500ns a 8MHz. La instrucción más lenta (una división signada de un número de 32 bits entre un número de 16 bits) puede llevarse hasta 170 periodos de reloj, o 21.25us a 8MHz.

De los 8 registros de direcciones el A7 es el "STACK" del sistema que tiene doble funcionamiento. El apuntador de stack de usuario, el cual salva las direcciones de "return" durante llamadas a subrutinas, está activo cuando el CPU esta en el estado usuario. El apuntador de stack supervisor, el cual "salva" las

direcciones de "return" y el contenido del registro de estado durante rutinas de interupción y de trampa (trap), está activo en el estado supervisor.

Los dos comparten el mismo registro puesto que no se pueden accesar simultáneamente. Todo esto lo hace un microprocesador muy poderoso y completamente compatible con la microcomputadora CBM AMIGA (la cual tambien posee un microprocesador MC68000) y es la mejor maquina que hay hasta el momento para diseño grafico.

Para mayores detalles del microprocesador MC68000 se da un explicación de su funcionamiento en el apendic A.

II.2 DESCRIPCION DE SUBSISTEMAS.

La informacion desde o hacia la CPU y otros subsistemas puede ser de tres tipos: direcciones, datos y control; por lo que fundamentalmente se encontrarán tres tipos de buses en el microcomputador, aunque no esten fisicamente siempre separados, ya que pueden estar multiplexados en el tiempo sobre lines comunes. La interconexión de la CPU con el sistema de memoria y E/S se realiza mediante estos buses, tal como se muestra en la fig.#4.

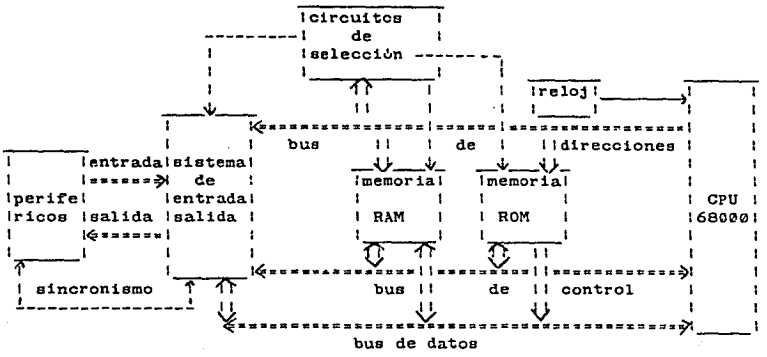


fig.#7.

Los distintos grupos funcionales que se pueden resumir en los siguientes incisos:

a).- Un sistema de entrada y salida para establecer la comunicación con el mundo exterior (recepción de órdenes y datos, y entrega de resultados), que será realizada por medio de una ACIA MC6850 (asynchronous communications interface adapter), este dispositivo servira para realizar la interface serie RS-232. Mientras que para realizar la interface paralelo se utilizaran dos circuitos PIA MC6821 (parallel interface adapter) las fig. #5 y #6 muestran dos diagramas a bloques de estas interfaces.

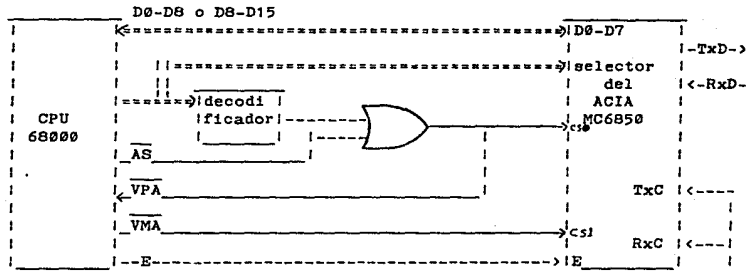

D0-D8 o D8-D15
D0-D7
-TxD->
selector
del
ACIA
MC6850
<-RxD-
CPU
68000
decodi
ficador
cs0
AS
VPA
TxC
VMA
cs1
RxC
E
baud rate
generador

fig #5

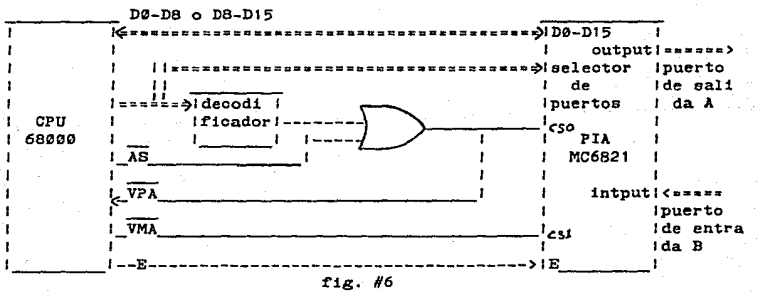
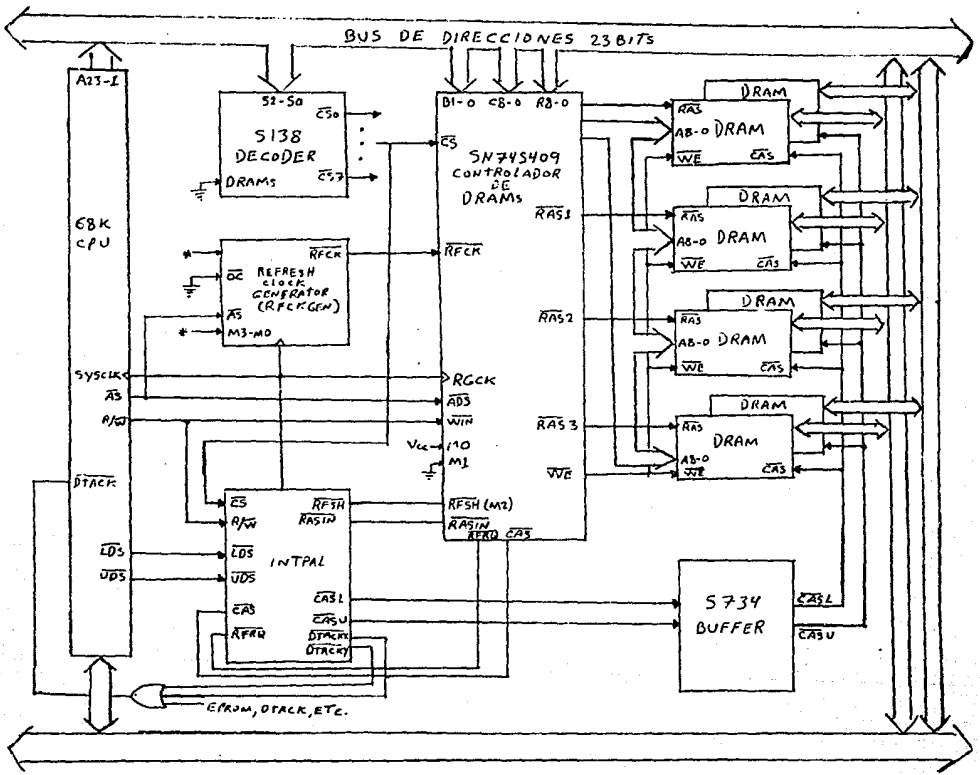

D0-D8 o D8-D15
D0-D15
output
selector
de
puertos
PIA
MC6821
intput
puerto
de sali
da A
puerto
de entra
da B
CPU
68000
decodi
ficador
cs0
AS
VPA
VMA
cs1
E

fig. #6

b).- Un sistema de memoria para la obtención de instrucciones (programas) y almacemamiento de información (datos).

El diseño del banco de memoria RAM se realizo a partir de un controlador 74S409-2 (Dynamic RAM controller/driver) para un tiempo de acceso de 150ns. La interface se implementó usando dispositivos PAL (programmable array logic) como se muestra en la fig. #7 . El 74S409-2 opera en modo de acceso automatico (modo 5) con refresco oculto, el sistema basico incluye un CPU MC68000 a 8MHz, RAM dinamica, EPROM y logica asociada.

La interface al banco de RAM dinamica incluye el circuito decodificador, el cual da el RESET del sistema, un reloj generador del refresco (RFCKGEN), el cual divide el reloj del sistema para proveer un reloj para el refresco oculto y refresco forzado, y un circuito el cual provee el control para el 74S409-2 y tambien señales de "handshake" para el CPU (INTPAL). La generación del refresco, control del 74S409-2 y "handshake" con el CPU fueron diseñados en dos dispositivos PAL, un diagrama equivalente de ambos circuitos es mostrado en la fig. #8 .



*conectar a Vcc o GND dependiendo del periodo de refresco elegido
fig #7

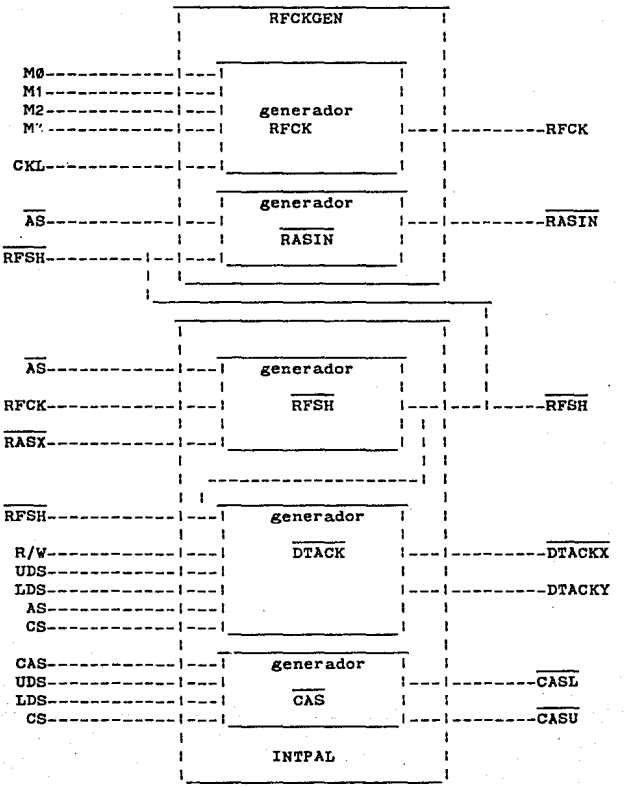

RFCKGEN
M0
M1
M2
M3
CKL
generador
RFCK
RFCK
AS
RASIN
RASIN
RFSH
AS
RFCK
RASX
generador
RFSH
RFSH
RFSH
R/W
UDS
LDS
AS
CS
generador
DTACK
DTACKX
DTACKY
CAS
UDS
LDS
CS
generador
CAS
CASL
CASU
INTPAL

fig. #8

CAPITULO III DESCRIPCION DEL HARDWARE Y DEL SOFTWARE

III.1 EXPLICACION DE LAS SEÑALES UTILIZADAS

GENERACIÓN DE RASIN. Rasin es activada por una de las dos AS o la señal $\overline{\text{RFSH}}$ venida de fuera del INTPAL.

RELOJ DE REFRESCO (RFCK). El tiempo RFCK LOW puede ser seleccionado a permitir el largo del intervalo necesitado para el refresco forzado. El intervalo de refresco forzado puede incluir el tiempo necesario para el sistema a completar un ciclo de acceso, el tiempo necesitado por el dispositivo PAL para conmutar desde el modo 5 (acceso) al modo 1 (refresco), el tiempo necesitado por el actual ciclo de refresco, y el tiempo de precambio RAS.

GENERACION DE $\overline{\text{RFSH}}$. Cuando RFSH va a HIGH y no hubo refresco oculto preformado durante el tiempo RFCK LOW, el 74S409-2 genera una petición de refresco ($\overline{\text{RFRQ}}$) o señal al sistema que un refresco es necesario. En otro tiempo $\overline{\text{AS}}$ es HIGH la señal $\overline{\text{RFSH}}$ pasa a LOW y conmuta el 74S409-2 en su modo de operación desde el acceso al refresco. La señal $\overline{\text{RASIN}}$ es habilitada despues de un retraso de tiempo. La señal interna SIG es generada cuando ambas $\overline{\text{RAS}}$ y $\overline{\text{RFSH}}$ son LOW y es usada a extender la duración de $\overline{\text{RFSH}}$ LOW para un ciclo despues $\overline{\text{RASIN}}$ sea HIGH, por lo tanto retrasando un acceso pendiente para un mayor ciclo en orden a satisfacer los requerimentos de precambio de $\overline{\text{RAS}}$.

GENERACION DE $\overline{CAS}$. Siendo que el CPU 68000 tiene una capacidad de acceso de byte individual, la señal $\overline{CAS}$ comienza fuera del 74S409-2 pudiendo ser dividida en dos señales; $\overline{CASU}$ y $\overline{CASL}$. $\overline{CASU}$ y $\overline{CASL}$ son habilitadas por sus respectivos habilitadores de datos $\overline{UDS}$ y $\overline{LDS}$.

GENERACION DE $\overline{DTACK}$. Dos señales viniendo de fuera del INTPAL; $\overline{DTACKX}$ y $\overline{DTACKY}$. $\overline{DTACKX}$ reconoce un ciclo de lectura y es generado asi un ciclo de espera (dos estados de espera) es insertada en el ciclo de memoria. $\overline{DTACKY}$ reconoce un ciclo de escritura y no permite estados de espera en operación. Las dos $\overline{DTACK}$s pueden ser pasadas por una compuerta AND con las otras $\overline{DTACK}$s para crear la señal $\overline{DTACK}$ para el CPU 68000.

TIEMPO DEL CICLO DE LECTURA. El cilo de lectura comienza cuando $\overline{AS}$ es puesta LOW. Si el dato es obtenible no antes de "tdicl" despues de la transición negativa del pulso de reloj S6, entonces la operación de estados de no espera puede ser realizada habilitando $\overline{DTACK}$ no antes que tasi antes de la transición negativa del pulso de reloj S4. El $\overline{AS}$ hasta el tiempo de dato para varias frecuencias de reloj puede ser calculado de las ecuaciones (ver la tabla #3);

$\overline{AS}$ de dato es valido = 2.5T-tdicl-tchsLn
(para un ciclo de lectura con estados de no espera)

donde: tchsLn = reloj HIGH hasta $\overline{AS}$ LOW (max)

T = tiempo de ciclo del reloj del sistema

tdicl = entrada de dato hasta reloj LOW

FRECUENCIA	AS LOW hasta dato (min)
4 MHz	515 ns
6 MHz	332.5 ns
8 MHz	237.5 ns
10 MHz	185 ns
12.5 MHz	135 ns

TABLA #3

Cuando el ciclo de memoria no puede ser completado con el $\overline{AS}$ LOW hasta la entrada de dato en intervalo valido, estados de espera son introducidos a crear el tiempo de acceso deseable en incrementos de T (ciclo de reloj) correspondiendo a minimos incrementos de dos estados de espera. La introducción de estados de espera es realizada reteniendo $\overline{DTACK}$ HIGH hasta despues de la transición negativa del pulso de reloj S4.

TIEMPO DEL CICLO DE ESCRITURA. Durante el ciclo de escritura, los datos llegan a ser disponibles desde el 68000 al;

1.- "trldo" despues que $\overline{R}$/W va a LOW

2.- "tdosl" antes que $\overline{UDS}$, o $\overline{LDS}$, o ambos vayan a LOW

3.- "tcldo" despues de la transición negativa del pulso S2

4.- T+tcldo-tchsl despues que $\overline{AS}$ va a LOW

El dato esta obtenible a "tchdo" despues que de la transición positiva del pulso S7 y tshdo despues que $\overline{AS}$ y $\overline{DS}$ activado va a HIGH. El dato es obtenible por lo tanto solamente 3T-tchsLx+tchdo desde que $\overline{AS}$ va a LOW (ver la tabla #4).

FRECUENCIA	$\overline{AS}$ LOW HASTA SALIDA DE DATO VALIDO		RISE LOW HASTA SALIDA DE DATO VALIDO	SALIDA DE DATO HASTA $\overline{DS}$ LOW	SALIDA DE DATO SOSTENIDO DE $\overline{DS}$ HASTA $\overline{AS}$ HIGH	SALIDA DE DATO SOSTENIDO DESDE $\overline{AS}$ LOW	SALIDA DE DATO VALIDO
	MIN	MAX	MIN	MAX	MIN	MIN	MIN
4 MHz	260ns	340ns	55 ns	55 ns	60 ns	670 ns	330ns
6 MHz	177ns	247ns	35 ns	35 ns	40 ns	431 ns	184ns
8 MHz	135ns	195ns	30 ns	30 ns	30 ns	315 ns	120ns
10 MHz	100ns	155ns	20 ns	20 ns	20 ns	245 ns	90ns
12.5 MHz	80ns	135ns	10 ns	15 ns	15 ns	185 ns	50ns

TABLA #4

III.2 TIEMPOS DEL ARREGLO DE MEMORIA DINAMICA.

TIEMPO DE ACCESO. Cuando calculamos el tiempo necesario para el arreglo de memoria dinamica a completar un ciclo de acceso, uno puede sumar los retardos producidos por los buffers entre el sistema y el 68000, por los dispositivos PAL que puertean las señales al controlador de RAM, por el controlador de RAM dinamica, y por el generador de CAS el cual incluye un PAL y un

74S734 MOS driver. La figura #9 ilustra los retardos involucrados en un ciclo de escritura y un ciclo de lectura para un arreglo de RAMs dinamicas de tiempo de acceso de 150ns. El analisis de tiempos es dado para Hitachi's HM256-15 dynamic RAM, pero muchos parametros criticos son iguales o muy parecidos para otras memorias dinamicas de tiempo de acceso de 150ns.

Para nuestro circuito, el tiempo desde $\overline{AS}$ LOW hasta $\overline{RAS}$ LOW es de 68ns, y el tiempo desde $\overline{AS}$ LOW hasta $\overline{CAS}$ LOW es de 215ns, y el de $\overline{AS}$ LOW hasta la entrada del dato valido (lectura) es de 283ns. Con un CPU de frecuencia de 8MHz.

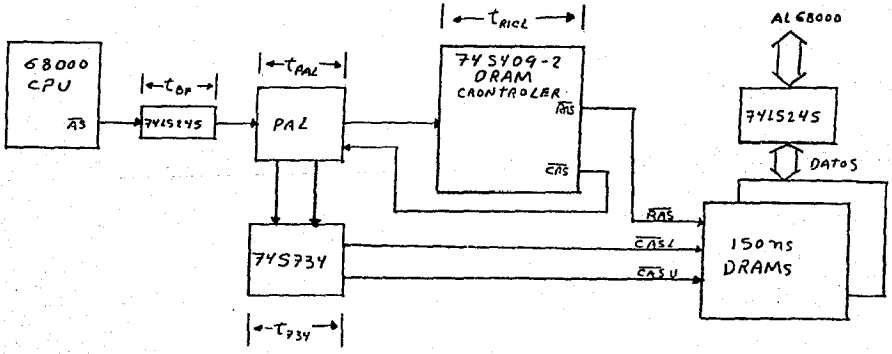


fig #9.

TIEMPO DE CAMBIO. Las DRAMs requieren que ambas $\overline{RAS}$ y $\overline{CAS}$ esten desactivas (HIGH) para duraciones especificas de tiempo en orden a conceder al circuito interno de la DRAM a "cambiar". El tiempo de cambio de $\overline{RAS}$ es igual al tiempo de cambio de $\overline{CAS}$ y requiere especial atención. Tiempos tipicos de cambio de $\overline{RAS}$ y $\overline{CAS}$ son mostrados en la tabla #5 de abajo.

TIEMPO DE ACCESO EN ns	100	120	150	200
tiempo de cambio de RAS (RAS HIGH a RAS LOW)	80	90	100	120
tiempo de cambio de CAS a RAS (CAS HIGH a RAS LOW)	10	10	10	10

TABLA #5

Cuando la salida $\overline{AS}$ del 68000 es conectada atraves de un buffer y un dispositivo PAL en la entrada $\overline{RASIN}$ del controlador de RAM dinamica 74S409-2, el tiempo de cambio obtenible para una RAM dinamica puede ser calculado de la siguiente ecuación.

tiempo de $\overline{RAS}$ HIGH = tsh(min)-Max($\overline{AS}$ LOW hasta el retardo $\overline{RAS}$ LOW)+Min(AS HIGH hasta el retardo RAS HIGH)

Donde "tsh" es el tiempo $\overline{AS}$ HIGH y el $\overline{AS}$ hasta el retardo $\overline{RAS}$ consiste de el retrdo del buffer, el reterdo del dispositivo PAL, y el retardo del 74S409-2.

III.3 DESCRIPCION DE LAS INTERFACES A MEMORIA.

Un diagrama de la conexión de las señales del bus de datos (ver fig #10 al final dl capitulo), del bus de direcciones, $\overline{DTACK}$, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, R/$\overline{W}$, $\overline{RESET}$, $\overline{HALT}$ y CLK al MC68000L8 por medio de BUS driver 74LS245 las cuales sirven de interface entre el CPU y el sistema controlador de memoria.

Las señales $\overline{BGACK}$, $\overline{BR}$, $\overline{VPA}$, $\overline{BERR}$ y $\overline{IPL}$0-2 todas van conectadas a Vcc por medio de una resistencia de 10K.

El diagrama de la memoria ROM implementado con un par de decodificadores 74LS138, un PAL20R4A y un par de compuertas 74LS32, asi como un par de memorias EPROM 2732. Las cuales me generan 8KB de ROM, (figura #11 al final del capitulo).

El Diagrama de la conexión del controlador de memoria RAM a los circuitos PAL20X8, PAL20R6 y al banco de memorias RAM, (figura #12 al final del capitulo).

Para una mayor descripción de los circuitos PAL consultar el anexo B.

III.4 DESCRIPCION DE LA INTERFACE SERIE.

La interface serie trabaja a una frecuencia de 2MHz, pudiendo transmitir información con un BAUD RATE desde 110 hasta 9600, el cual es seleccionado por medio de un switch , el formato de la transmisión es programable, pero esta programación se realiza por medio del programa de inicialización del sistema, la interface tiene unos BUS DRIVER para convertir la señal de TTL a RS-232C, como se ve de la figura # 13 al final del capitulo, para mayor detalle del ACIA consultar el apendice C.

III.5 DESCRIPCION DE LA INTERFAZ PARALELO

La interface paralelo fué diseñada utilizando dos PIAs MC6821 (Peripheral Interface Adapter), que son dispositivos de 40 patas que tienen dos puerto de 8 bits cada uno. Los cuales pueden ser configurados como entradas o salidas individualemte.

Operación asincrona, el diagrama de la interface MC68000/MC6821 aparece en la figura #14 al final del capitulo. La decodificación de la direcciones se realizo de la siguiente manera: Las lineas de dirección de A5--A9 son pasadas atraves de una compuerta NOR, al igual que las lineas de direcciones de A10--A14 que son pasadas atraves de la compuerta NOR, y las salidas de estas dos compuertas son pasadas atraves de una compuerta AND junto con la linea de dirección A15. Para poder direccionar el dispositivo que trabaja a una frecuencia de 2MHz

en forma asincrona, generando la señal DTACK por medio de dos FLIP-FLOPs los cuales habilitan al sistema y realizan el protocolo entre el dispositivo y el microprocesador.

III.6 EXPLICACION DEL SOFTWARE DEL SISTEMA Y SUBRUTINAS

Los siguientes diagramas de flujo muestran la forma en que son ejecutados los programas. Tanto el programa principal, como el programa de la subrutina que ejecuta la TRANSFORMADA RAPIDA DE FOURIER FFT.

El diagrama de flujo de la fig. #15 muestra la forma en que inicializado el sistema al ser encendido, o al terminar de procesar una imagen. El sistema pone el nivel de interupción más alto (nivel 7), para evitar otras interupciones de menor nivel y limpia la memoria de basura, para evitar errores de lectura de datos, llamados "bus error". Inmediatamente despues programa los perifericos, para que puedan realizar la comunicación con el computador maestro, e inicializa los apuntadores de buffer para guardar los datos a recibir asi como los contadores a utilizar durante la recepción de las dos matrices a trnasformar. Una vez hecho todo esto habilita las interrupciones y espera la interrupción por alguno de los dos puertos, ya sea el serie RS-232C o el paralelo, cuando recibe la interrupción de alguno de estos dos puertos (de nivel 3 y de nivel 4 respectivamente) carga la subrutina que le da servicio a este puerto, de la siguiente forma, cargando los registros de control del puerto que

mando la interrupción para monitoriarlo y realizar una busqueda sobre su registro de datos para ver si hay datos a recibir e incrementando un registro contador y checar si fuerón todos los datos que se tenian que recibir. Posteriormente ejecuta la subrutina FFT que se detalla más adelante, y regresa la información por el mismo puerto que la recibio, para despues volver a la condición de inicio (para mayor detalle del programa escrito en lenguaje ensamblador para el microprocesador MC68000 consultar el apendice D).

Por otra parte el diagrama de flujo de la figura #16, muestra la forma en que son procesadas las matrices real e imaginaria de la información recibida. Al ser llamada la subrutina FFT, primeramente se cargan los primeros vectores renglón de las dos matrices y se manda llamar a una subrutina llamada IMAGEN, que realiza la transformada de FOURIER de un vector (parte real y parte imaginaria, dando como resultado tambien una parte real y una parte imagiraria de la transformada del vector). Despues de haber procesado todos los renglones de ambas matrices, de la misma forma que se procesaron los primeros renglones, se transponen las matrices resultantes real e imaginaria, y se vuelve a ejecutar la misma subrutina IMAGEN sobre cada uno de los renglones, que en realidad son las columnas de las matrices; obteniendose asi dos matrices que representan la parte real e imaginaria de la imagen, posteriormente se obtiene el modulo de éstas matrices (que representa la brillantes o nivel de gris de cada

punto de la imagen obtenida) y se divide en cuatro cuadrantes para hacer el intercambio de los cuadrantes, uno por el tercero y viceversa, y el segundo por el cuarto y viceversa, obteniendose asi el resultado final (esto es por que la subrutina que realiza la transformada de FOURIER intercambia los cuadrantes del resultado). Que es la información final que entrega el sistema (para mayor detalle del programa escrito en lenguaje ensamblador para el microprocesador MC68000 y un progrma en lenguaje PASCAL escrito para el compilador TURBOPASCAL para las computadoras COLUMBIA PC, este programa sirvio para simular el algoritmo utilizado en el calculo de transformada de FOURIER consultar el apendice D).

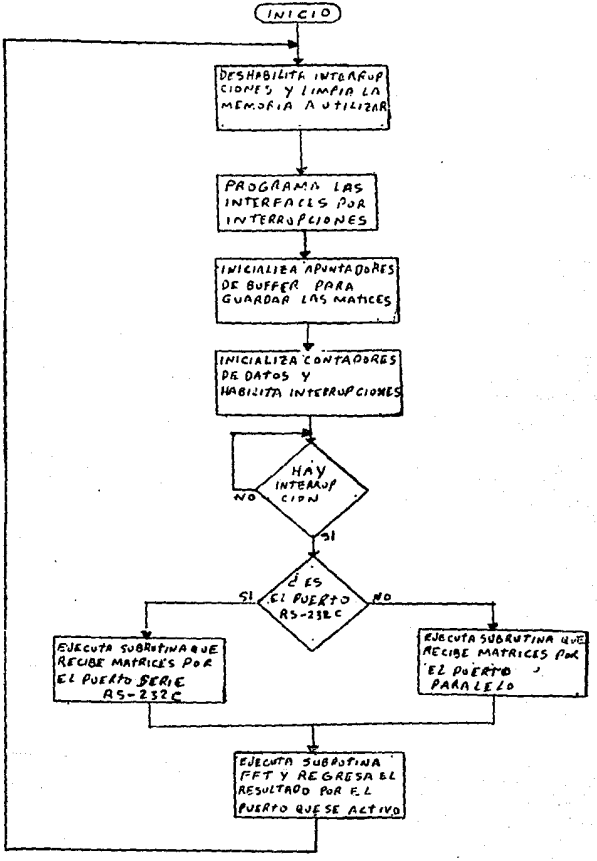


fig #15 diagrama de flujo del programa monitor.

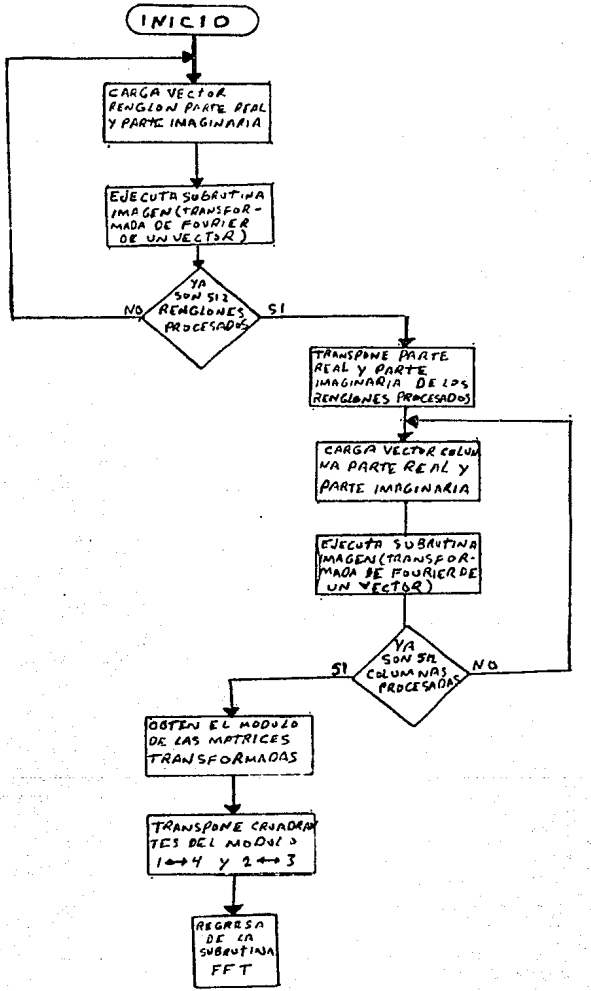


fig #16 diagrama de flujo del programa de la subrutina FFT.

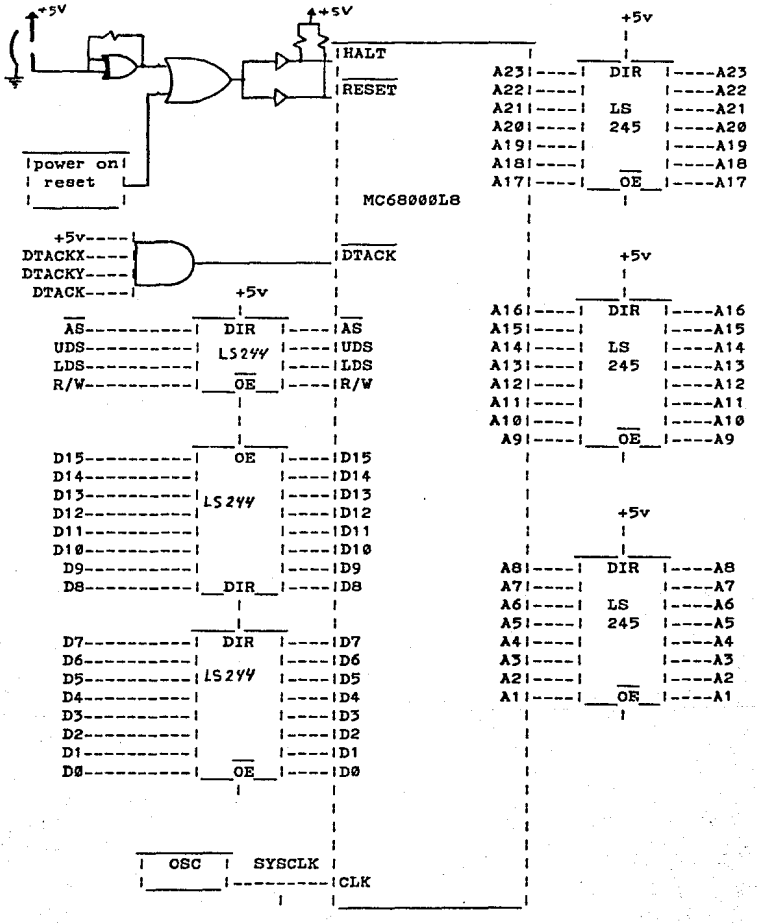

+5V
+5V
+5v
HALT
RESET
power on reset
MC68000L8
+5v----
DTACKX----
DTACKY----
DTACK----
DTACK
+5v
AS----------
UDS----------
LDS----------
R/W----------
DIR
LS244
OE
AS
UDS
LDS
R/W
D15----------
D14----------
D13----------
D12----------
D11----------
D10----------
D9----------
D8----------
D7----------
D6----------
D5----------
D4----------
D3----------
D2----------
D1----------
D0----------
OE
LS244
DIR
LS244
DIR
OE
D15
D14
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
A23----
A22----
A21----
A20----
A19----
A18----
A17----
DIR
LS 245
OE
A23
A22
A21
A20
A19
A18
A17
A16----
A15----
A14----
A13----
A12----
A11----
A10----
A9----
A8----
A7----
A6----
A5----
A4----
A3----
A2----
A1----
DIR
LS 245
OE
A16
A15
A14
A13
A12
A11
A10
A9
DIR
LS 245
OE
A8
A7
A6
A5
A4
A3
A2
A1
OSC
SYSCLK
CLK
SYSCLK

fig #10

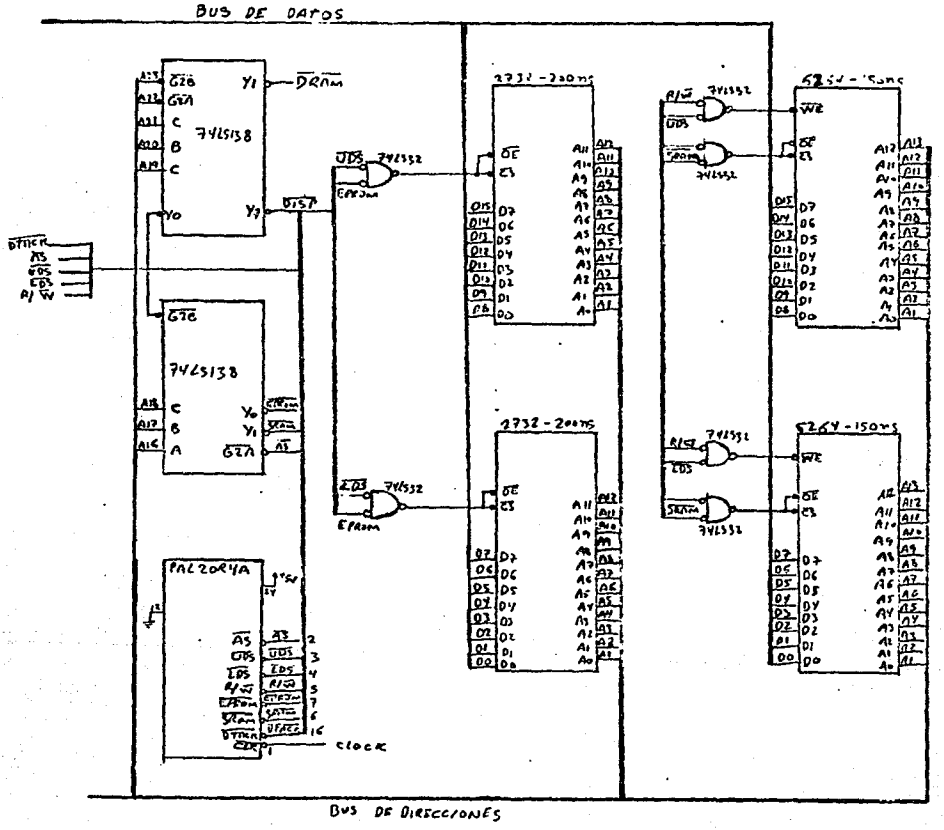


fig #11 diagrama de la memoria ROM.

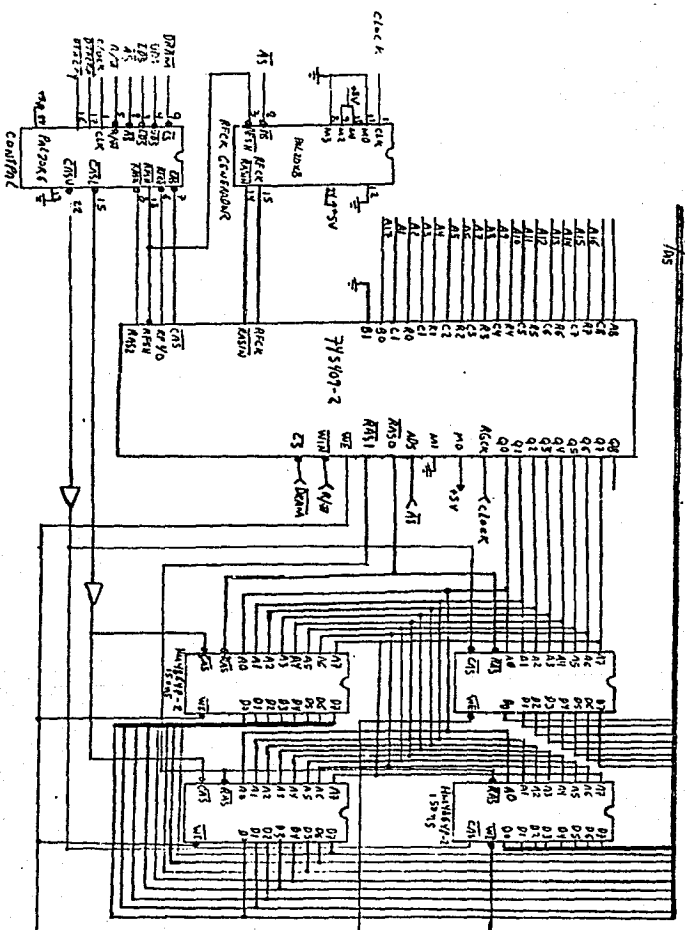


fig #12 diagrama de la interface entre el controlador y la memoria RAM.

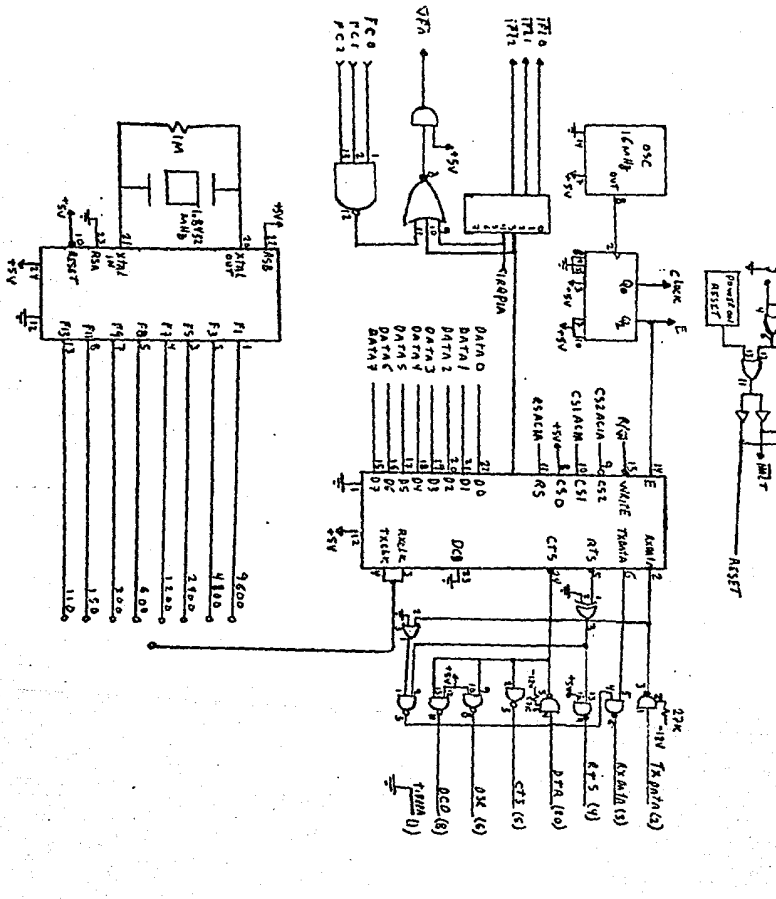


fig #13 diagrama de la inteface serie RS-232C.

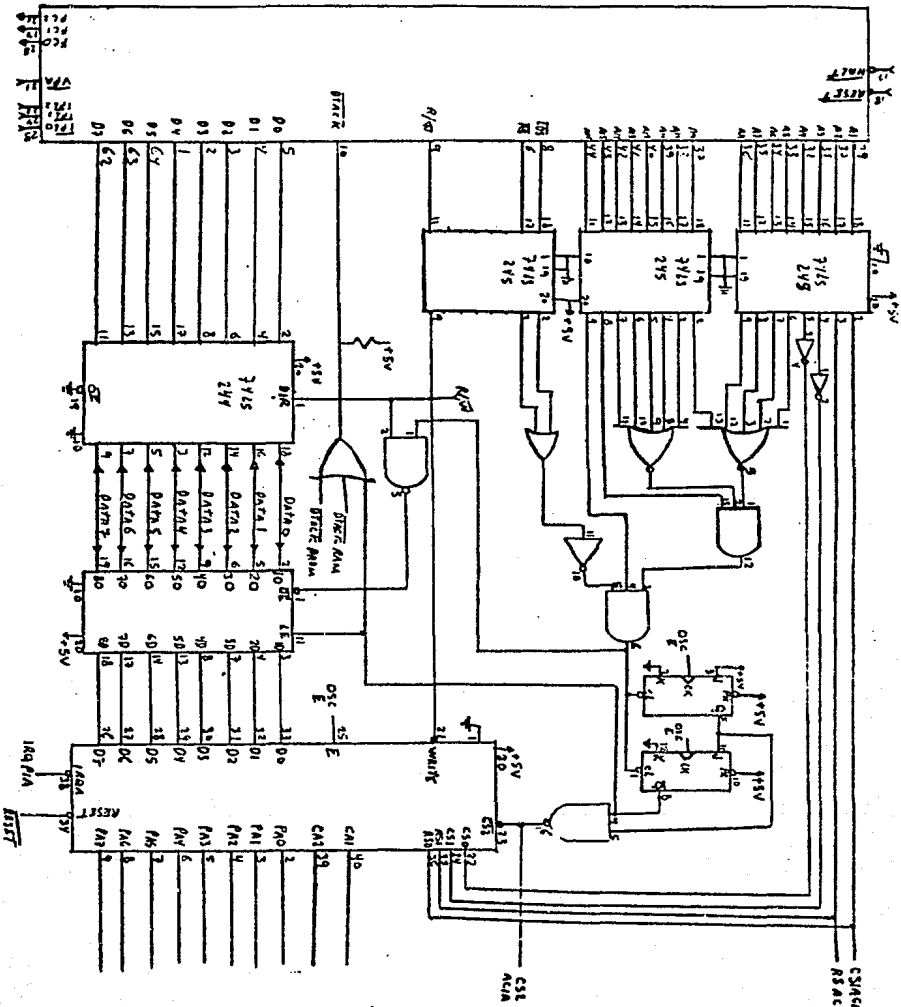


fig #14 diagrama de la interface paralelo.

CAPITULO IV PARTICULAS MULTIPLEMETE GEMELADAS

CONCLUCIONES

IV.1 APLICACION DEL PROGRAMA SIMULADOR A UN PROBLEMA EN FISICA

En la industria petroquimica la utilización de catalizadores es de vital importancia, y en especial los catalizadores metalicos, que son elaborados a partir de metales caros como el oro, paladio, radio y otros. Ya que estos catalizadores son utilizados en forma de particulas pequeñas de estos metales, es de gran interés estudiar este tipo de particulas pequeñas.

Las particulas pequeñas con simetria cinco ("five-fold symmetry") como lo muestra la figura #17, tales como decaedros e icosaedros, presentan una estructura que ha originado un gran interés en la comunidad cientifica, la cual no ha podido determinar con toda exactitud este tipo de estructura. Para explicarse este tipo de estructura se han propuesto dos hipótesis que tratan de describir el tipo de estructura de estas particulas pequeñas, las cuales son:

a).- Que las particulas pequeñas forman una estructura del tipo BCO (ortorombica centrada en el cuerpo), ó sea que se forman a partir de una estructura cubica homogeneamente deformada.

b).- Que la estructura es FCC (cubica centrada en las caras) deformada inhomogeneamente.

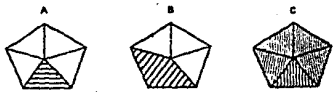


Fig. 4. Electron diffraction pattern and real space image contrasts of a decahedral particle in the "fivefold" orientation.

fig #17.

Por otro lado se sabe que las particulas de elememtos como el oro tienen una estructura FCC formada por tetraedros regulares como lo muestra la figura #18.

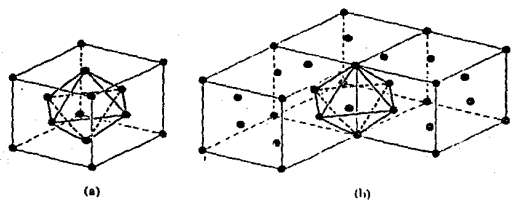


Fig. 26-8 Huecos octaédricos en estructuras cfc. (a) Cuerpo centrado. (b) Arista centrada.

fig #18

Si quisieramos unir cinco de estos tetraedros para formar un decaedro regular como lo muestra la figura #19, se forma una pequeña separación llamada "HENDIDURA DE VOLTERRA" que tiene un valor de 7 grad 20 min, sin embargo se han tomado fotografias en microscopia de particulas pequeñas que tienen tal estructura decaedral (YANG, JOSE YACAMAN y HEINEMMAN, 1979) como lo muestra la fotografia de la figura #20.

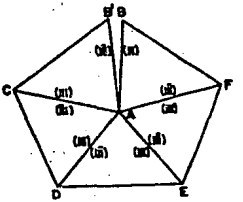


fig #19

FIG. 9. (111) lattice image of decahedral particle in face orientation. (a) The whole particle is shown. (b) Magnification of the boundary region.

fig #2C.

De todo esto uno se pregunta, ?Cuál es la forma de unir estos cinco tetraedros para formar un decaedro sin que aparezca tal separación?, ?Cuál de las dos hipótesis para explicar la estructura de estas particulas es la correcta?. Una forma rápida de tratar de indagar sobre estas interrogantes y saber si es una buena linea de investigación o no, ya que este tipo de trabajo diria si por medio de la microscopia electrónia tiene o no solución dichop problema, es la de simular patrones de difracción por computadora utilizando las posiciones de los átomos difractantes predichas por las dos hipótesis antes mencionadas.

Estos patrones se muestran en las figuras #21 y #22, y para BCO y DE WITT respectivamente y mediante la transformada de Fourier se pueden obtener sus correspondientes imagenes figuras #23 y #24 para después compararlas con las imagenes del microscopio de una de estas particulas y poder discriminar diferencias que indiquen, cual de las dos hipótesis describe acertadamente la solución al problema de la estructura de las particulas pequeñas con forma decaedral.

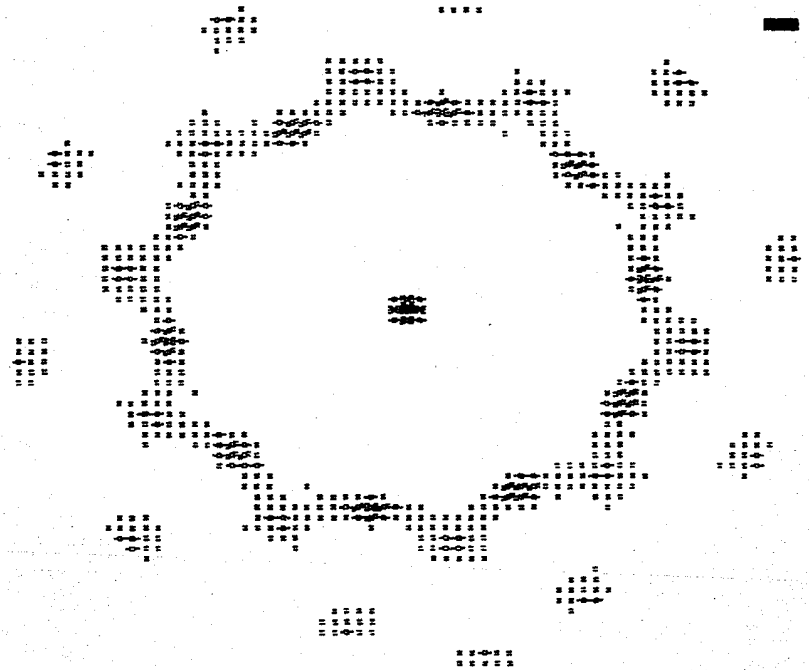


fig #21 patron de difracción para una particula del tipo BCO

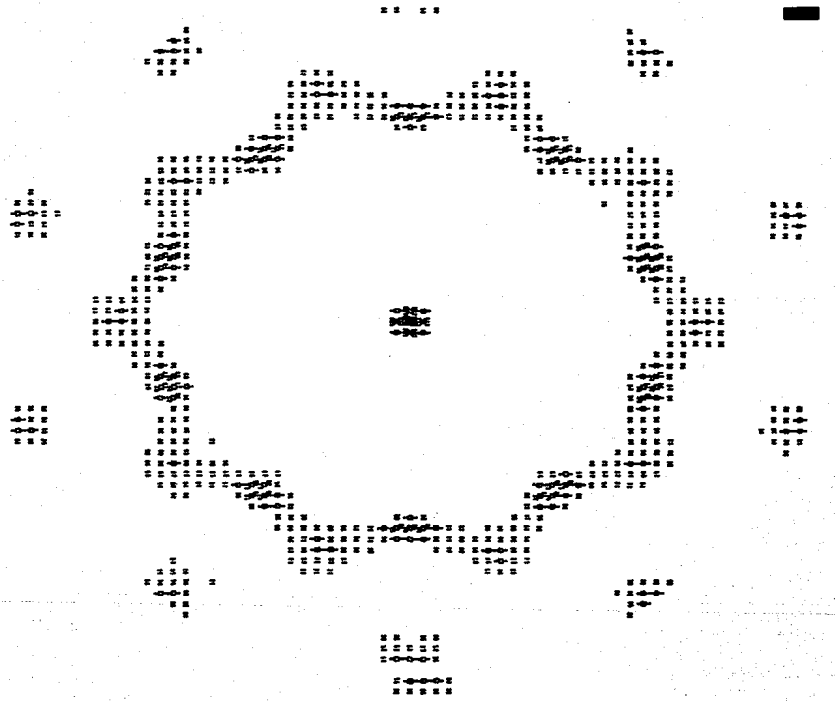


fig #22 patron de difracción para una particula del tipo DE WITT

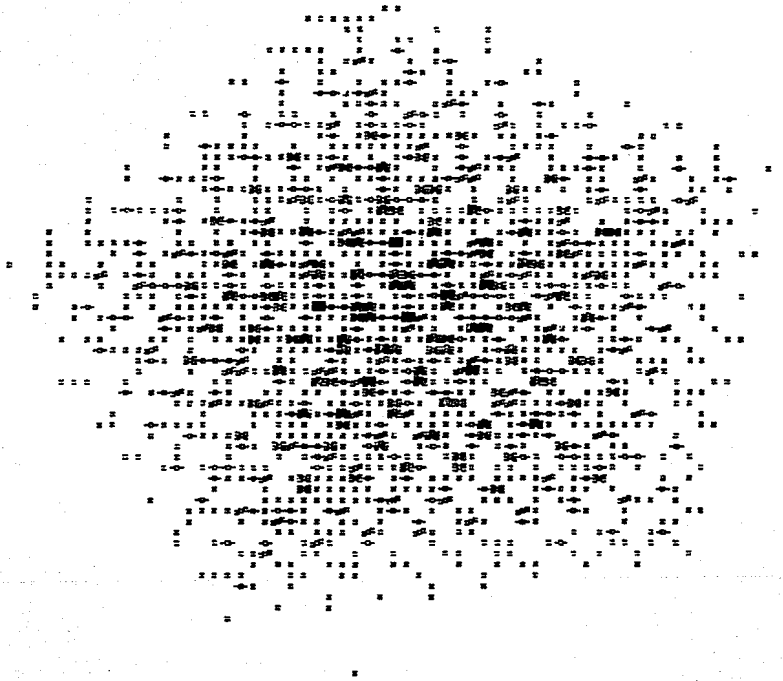
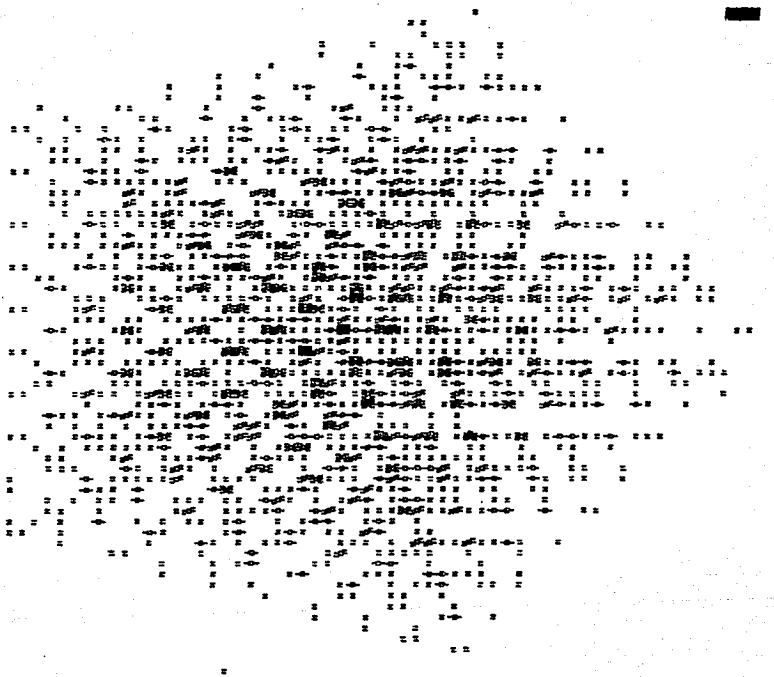


fig #23 imagen de una particula del tipo BCO



. fig #24 imagen de una particula del tipo DE WITT

IV.2 CONCLUSIONES

La simulación por computadora de los patrones de difracción se realizo por medio de un modelo matematico de las posiciones de los atomos en cristales ideales de acuerdo a las dos hipotesis propuestas, y la difracción se obtuvo haciendo incidir un haz de ondas planas sobre los cristales.

Esto equivale a realizar un experimento de difracción con un microscopio ideal, esto es que no tenga problemas de aberraciones, además de que la muestra no presentaria ningun tipo de dislocaciones o impurezas que pudieran alterar el patrón de difración y la imagen de ésta.

Por otra parte como se observa en las transformadas obtenidas de los patrones de difracción para las dos hipotesis propuestas, no se aprecian diferencias concluyentes que indiquen cual de las dos hipotesis es la correcta o se apegue más a la realidad.

De todo esto se puede concluir que si realizando el experimento ideal no se obtubo un resultado satisfactorio, para resolver el problema, entonces la aplicación de las tecnicas de microscopia no son el camino adecuado para la comprobación de las dos hipotesis propuestas al principio de este capitulo, por lo que se tendran que buscar nuevas tecnicas que arrojen luz sobre este problema.

Con respecto al objetivo principal de esta tesis, se tuvieron las siguientes conclusiones. Como el sistema nada más se diseñó y no se construyó solo se pudo evaluar las subrutinas, pero esto no implica que no se puedan hacer algunas consideraciones respecto a este, ya que se tuvo la oportunidad de hacer una emulación en un sistema de desarrollo HP-64000 que emulaba entre otros microprocesadores al MC68000, y que fue el que se escogio para realizar el diseño del sistema.

De ahi que se pueda decir lo siguiente;

1.- El sistema tiene diseñada una interface RS-232C que recibe información con una velocidad de 9600 bits por segundo, con un formato de diez bits donde el primero es el bit de inicio y el ultimo es el bit de alto. Por lo que si el sistema recibiera por este puerto la información, que serian 524288 elementos de las dos matrices de la información y como cada elemento se representa por un medio de un número de 16 bits, esto daria un total de 10485760 bits a recibir o transmitir tomando ya en cuenta los bits de inicio y final, que tomarian aproximadamente 18 minutos en recibir o transmitir la información. Lo que es muy tardado y molesto, pero este tipo de interface hace el sistema compatible con una gran cantidad de computadoras que tienen este tipo de comunicación.

2.- La otra interface paralelo, es una interface de 16 bits que trabaja a 2MHz por lo que el tiempo de recepción o transmisión de

datos seria de 5.5 segundos, lo que la hace una interface bastante rapida.

3.- Con respecto a la memoria dinámica el problema del refresco no es problema ya que se utilizó un controlador de memoria dinámica que tiene la posibilidad de realizar el refresco sin interrumpir al microprocesador, esto es llamado refresco oculto, lo que le da todo el tiempo al microprocesador para emplearlo en sus tareas normales de calculo y manejo de datos.

4.- Por ultimo se puede hacer una estimación del tiempo que tardara el sistema en realizar la transformada rapida de FOURIER, si tomamos en cuenta que son del orden de mil millones de instrucciones las que tendrá que ejecutar el microprocesador con un promedio de un ciclo de maquina por instrucción, y utilizando un microprocesador de 8MHz el tiempo seria de aproximadamente 2 minutos.

Todos estos calculos se estimaron sin tomar en cuenta los retrasos de las compuertas discretas y sin considerar que hay operaciones que dependen si se realizan o no de otros resultados, por lo que los tiempos pueden aumentar o disminuir, pero se espera que estos cambios no sean muy datrasticos.

APENDICE A DESCRIPCION DEL MC68000.

DISTRIBUCION DE TERMINALES Y SEÑALES

El MC68000 viene en un encapsulado "dual-inline" de 64 patas, algunos nombres de las patas se le coloca una barra encima, con el objeto de distinguir cuales de las señales son activas bajas (esto quiere decir que se activan con un nivel de voltaje de 0 volts), como lo muestra la fig.#25.

ALIMENTACION, TIERRAS Y RELOJ.- El MC68000 opera con +5v conectados a dos patas, etiquetadas Vcc; igualmente tiene dos patas etiquetadas con GND. La entrada de reloj (CLK) es la señal de nivel TTL que puede tener una frecuencia desde 4MHz hasta 10MHz, dependiendo de tipo de procesador usado.

BUS DE DATOS Y BUS DE DIRECCIONES.- Al MC68000 se le llama un procesador de 16 bits debido, a que su unidad basica de información, la palabra, es de 16 bits. Esto signbifica que a un tiempo, solamente 16 bits se pueden transferir de, o a la memoria y a dispositivos de entrada/salida en paralelo. Para transferir más de 16 bits se requerirá de mas ciclos de transferencia. Toda transferencia se lleva a cabo por el bus de datos (D0-D15).El bus de direcciones (A1-A23) 23 lineas,es el bus que usa el CPU para accesar el dispositivo al cual se quiere transmitir o recibir información. Dos señales adicionales, $\overline{\text{UDS}}$ (upper data strobe) y $\overline{\text{LDS}}$ (lower data strobe), sirven al CPU para seleccionar bytes dentro de una palabra. Con la señal $\overline{\text{AS}}$ (adress strobe), el CPU

notifica al sistema que hay una dirección valida en el bus de direcciones.

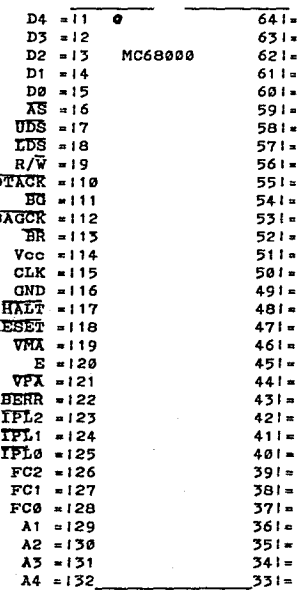


fig. #25.

SEÑALES DE FUNCION CODIFICADA.- Estas señales, etiquetadas como FC0, FC1 y FC2, le sirven al CPU para informar al sistema cuando está direccionando datos u operandos de programa y cuándo

está en estado usuario o estado supervisor y cuándo está atendiendo interrupciones.

Cuando el contador de programa es la fuente de dirección o cuando se está alimentando vectores de reset, las señales de función codificada indican que se está accesando un espacio de programa. Cuando el contador de programa no es la fuente de dirección o se están alimentando otros vectores que no sean los de reset, o cuando todos los operandos ya se escribieron, las señales de función codificada indican que se está accesando espacio de datos.

La tabla #6, muestra las diferntes combinaciones de estas tres señales. Nótese que la señal de función codificada más significativa FC2, refleja el estado del bit supervisor (S) en el registro de estado.

FUNCION CODIFICADA			CLASIFICACION	ESTADO DE PRIVILEGIO
FC2	FC1	FC0		
0	0	0	(reservada)	usuario
0	0	1	espacio de datos	usuario
0	1	0	espacio de programa	usuario
0	1	1	(reservada)	usuario
1	0	0	(reservada)	supervisor
1	0	1	espacio de datos	supervisor
1	1	0	espacio de programa	supervisor
1	1	1	interrup ackowldege	supervisor

tabla #6

LINEAS DE CONTROL.- el MC68000 tiene la ventaja de que puede conectarse a dispositivos sincronos o asincronos, sin necesidad de interface especial, ya que cuenta con lineas de control para cada dispositivo, especialmente para la familia del MC6800.

Los bytes individuales se accesan por medio del bus de direcciones y dos señales de control; $\overline{UDS}$ (upper data strobe) y $\overline{LDS}$ (lower data strobe). Cuando $\overline{UDS}$ se activa (0 logico) la información se transfiere en las 8 lineas altas del bus de datos (D8-D15). Cuando $\overline{LDS}$ se activa (0 logico) la información se transfiere por la parte baja del bus de datos (D0-D7).

La señal R/W le sirve al CPU para informar al sistema si quiere leer datos (1 logico) o si quiere escribir datos (0 logico).

La señal DTACK (data transfer acknowldege) la usa el sistema para informar al CPU que se ha puesto o tomado un dato en el bus de datos. Cuando el CPU sensa esta señal, durante un ciclo de lectura, amarra dato y da por terminado el ciclo de lectura.
La fig.#26, muestra todas las señales involucradas en el direccionamiento de memoria asincrona. El circuito timer tipo "Watchdog" tiene la función de esperar un determinado tiempo, entre la activación de la señal $\overline{AS}$ y la recepción de la señal $\overline{DTACK}$. Si los circuitos de memoria regresan la información adecuada de $\overline{DTACK}$ par y $\overline{DTACK}$ impar en ese tiempo, la señal $\overline{DTACK}$ es sensada por el CPU, de otro modo el timer accesa una señal de error BERR, lo cual ocasiona que el CPU inicie una excepción de

microprocesamiento. De esta manera el timer previene la posibilidad que un dispositivo fallo, mantenga al sistema en espera.

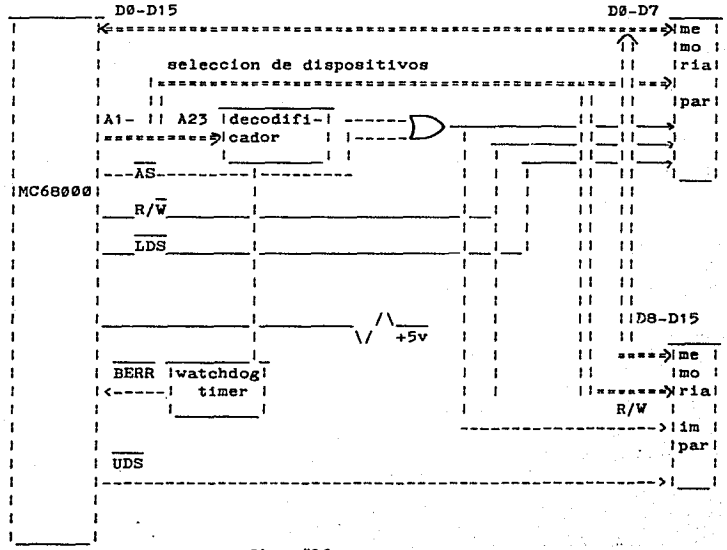


fig. #26.

SEÑALES DE CONTROL SINCRONO.- En el MC68000 existen tres señales para control sincrono, que son: "Ensamble" (E), "Valid peripherial adress" ($\overline{VPA}$) y "Valid memory adress" ($\overline{VMA}$).

La señal "Enable" (E) es una señal de reloj que usan los sistemas de 8 bits para sincronizar la transferencia de datos. La señal de reloj E, tiene una frecuencia de 1/10 de la entrada de reloj del MC68000 y tiene un ciclo de trabajo de 60/40; es 0 lógico para 6 ciclos de reloj y es 1 lógico para 4 ciclos de reloj. La señal $\overline{VPA}$, es una señal de entrada que notifica al MC68000 cuando se accesa un dispositivo de la familia MC6800 y que la operación de transferencia de datos debe ser sincronizada con el reloj E. Normalmente la señal $\overline{VPA}$ se deriva de una dirección decodificada y la señal $\overline{AS}$ (address strobe). Se puede decir que $\overline{VPA}$ es el equivalente sincrono de la señal asincrona $\overline{DTACK}$.

Cuando el MC68000 recibe $\overline{VPA}$ y $\overline{AS}$ está activa, el CPU responde activando la señal VMA (valid memory address), la cual se usa para completar la selección del dispositivo del sistema que se quiere accesar. La figura #27 muestra las señales que usaremos para conectar el MC68000 con perifericos del MC6800.

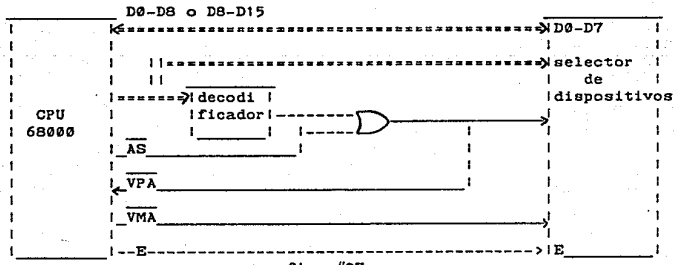


fig. #27.

BUS DE SEÑALES DE ARBITRAJE.- Estas señales se usan en aplicaciones de acceso directo a memoria y multiprocesamiento para transferir el control del bus del sistema a otros dispositivos exteriores. Para estas aplicaciones, los dispositivos externos que desean controlar las líneas del sistema (buses), informan al CPU su petición activando la señal de entrada BR (bus request). La prioridad del MC68000 sobre el bus está al ultimo comparando con dispositivos externos, por lo tanto el CPU cederá el control del bus después de completar un ciclo de bus normal. Mientras tanto, al sensar la señal $\overline{BR}$, el CPU se sincroniza internamente y luego indica su aceptación de la petición activando la señal $\overline{BG}$. (bus grant). En el caso de varios dispositivos que pudieran requerir el control del sistema de líneas, se requeriría de circuiteria externa para dar prioridades.

Despues de recibir la señal $\overline{BG}$ (bus grant), el dispositivo petitorio espera que CPU termine su ciclo normal y luego envia una señal de regreso al CPU $\overline{BGACK}$ (bus grant ackwoldege) para informarle que ya tiene el control de los buses. Luego el dispositivo petitorio avisa al CPU que ya termino su petición, desactivando la señal $\overline{BR}$, igualmente el CPU desactiva $\overline{BG}$ y espera a que el dispositivo externo termine su operación, o sea cuando éste desactive la señal $\overline{BGACK}$ y el CPU reasumirá su operación normal de nuevo.

SEÑALES DE CONTROL DEL SISTEMA.- Existen tres señales de control en el MC68000. Una de estas es solo de entrada y las otras dos son bidireccionales:

RESET ($\overline{\text{RESET}}$) es una señal bidireccional que permite al CPU o a un dispositivo externo reiniciar el sistema. Un reset generado por el CPU (inclusive por una instrucción de reset), activa la señal de $\overline{\text{RESET}}$ por un periodo de 124 ciclos de reloj, dandole asi tiempo al sistema para reinicializarse, sin afectar el estado interno del MC68000. En una falla total, el sistema completo (CPU y perifericos) se pueden reinicializar, si ambas señales $\overline{\text{RESET}}$ y $\overline{\text{HALT}}$ (señal de control bidireccional) son activadas al MC68000 por más de 100ms. Esto hace que el CPU inicie una secuencia de encendido (Power-on reset), durante la cual el CPU entra en el estado supervisor y salta a una rutina de reset desde un vector en la parte más baja de memoria.

De cualquier forma, la señal $\overline{\text{HALT}}$ no necesariamente necesita ir acompañada de la señal $\overline{\text{RESET}}$. Por si sola se puede usar como una señal de entrada al CPU, para que el CPU ejecute instrucción por intrucción (single step) para propositos de depuración de programas. El MC68000 puede tambien activar $\overline{\text{HALT}}$ para señal de salida. Esto ocurre cuando el CPU ha terminado de ejecutar instrucciones debido a una condición de excepción llamada "double bus fault", que se discutira después. La otra señal de control de sistema, que solo es de entrada al CPU, es $\overline{\text{BERR}}$. Esta señal sirve para informar al CPU que hay un problema en algun sitio del sistema. Dicho de otra manera, $\overline{\text{BERR}}$ indica o señala la ocurrencia de un evento no anticipado (como una interrupción espurea, o un requerimento de acceso ilegal a memoria) o la no ocurrencia de un evento anticipado (por ejemplo cuando un periferico ha fallado en

regresar la señal $\overline{\text{DTACK}}$ o $\overline{\text{VPA}}$). Cuando el MC68000 sensa la señal $\overline{\text{BERR}}$ bien puede iniciar una secuencia de excepción de error o tratar de correr el ciclo de bus de nuevo. Esto ultimo lo haria si la señal HALT se activa externamente cuando está recibiendo la señal $\overline{\text{BERR}}$. Cuando la lógica del sistema desactiva o niega $\overline{\text{HALT}}$ y BERR, entonces el CPU correrá los ciclos previos de bus.

LINEAS DE CONTROL DE INTERRUPCIONES.- Las lineas $\overline{\text{IPL0}}$, $\overline{\text{IPL1}}$ e $\overline{\text{IPL2}}$, debidamente decodifcadas, son las que usa el sistema para pedir una dirección. La longitud de palabra, en general, es la misma que la del microprocesador y bus de datos.

APENDICE B DESCRIPCION DE LOS CIRCUITOS PAL.

Descripción del circuito PAL20R4A (fig #28). Este PAL produce 0 estados de espera si uno de los dos una RAM escribe o una ROM lee, El PAL tambien tiene una entrada OR para DTACK externo (retrazado 25ns).

Este PAL genera la señal DTACK a partir de la siguiente tabla de verdad.

CKL	OE	CNTR	AS	UDS	LDS	R/W	DRAM	EPROM	EXTDTACK	DTACK
C	H	Z	X	X	X	X	H	H	L	L
C	H	Z	X	X	X	X	H	H	H	H
C	L	X	X	X	X	X	H	H	L	L
C	L	X	X	X	X	X	H	H	H	H
C	L	H	X	X	X	X	H	X	H	H
C	L	X	L	L	L	H	H	L	H	L
C	L	X	L	L	H	H	H	L	H	L
C	L	X	L	H	L	H	H	L	H	L
C	L	X	L	H	H	H	H	L	H	H
C	L	X	X	X	X	L	H	L	H	H
C	L	X	H	H	H	X	H	L	H	H
C	L	X	X	L	L	H	L	H	H	L
C	L	X	X	L	H	H	L	H	H	L
C	L	X	X	H	L	H	L	H	H	L
C	L	X	X	H	H	H	L	H	H	H
C	L	H	X	X	X	L	L	H	H	H
C	L	X	H	X	X	L	L	H	H	H
C	L	X	L	L	L	L	L	H	H	L
C	L	X	L	L	H	L	L	H	H	L
C	L	X	L	H	L	L	L	H	H	L
C	L	X	L	H	H	L	L	H	H	H

cuya ecuación logica es la siguiente;

CNTR = AS*UDS*LDS*$\overline{\text{CNTR}}$+UDS*CNTR+LDS*$\overline{\text{CNTR}}$

DTACK = EXTDTACK
+CNTR*AS*UDS*R/W*EPROM
+CNTR*AS*LDS*R/W*EPROM
+CNTR*AS*UDS*R/W*DRAM
+CNTR*AS*LDS*R/W*DRAM
+DRAM*R/W*UDS
+DRAM*R/W*LDS

$\overline{\text{CLK}}$=1	24=Vcc
$\overline{\text{AS}}$=2	23=NC
$\overline{\text{UDS}}$=3	22=NC
$\overline{\text{LDS}}$=4	21=NC
R/$\overline{\text{W}}$=5	20=NC
$\overline{\text{DRAM}}$=6	19=NC
$\overline{\text{EPROM}}$=7	18=NC
$\overline{\text{EXTDTACK}}$=8	17=$\overline{\text{CNTR}}$
NC=9	16=$\overline{\text{DTACK}}$
NC=10	15=NC
NC=11	14=NC
GND=12	13=OE

fig #28 asignacion de patas del PAL20R4A

Descripcion del circuito PAL20X8 (fig #29). Este PAL genera dos señales, la primera es $\overline{\text{RASIN}}$. Esta salida va al S409 y es usada como la entrada RAS. La segunda señal cuya duración es 128 ciclos de reloj del sistema. El tiempo de RFCK empezando LOW es determinado por las entradas M0-M3. el tiempo HIGH es obvio.

Las ecuaciones que describen el funcionamiento de este PAL son las siguientes;

RASIN = AS*$\overline{\text{RFSH}}$

$\overline{\text{RFCK}}$ = M3*Q6*Q5+M2*Q6*Q5*Q4
+M1*Q6*Q5*Q4*Q3+M0*Q6*Q5*Q4*Q3*Q2.

TIEMPO HIGH = TIMPO DEL TOTAL DE CICLOS-TIEMPO LOW

= 128 CICLOS-TIEMPO LOW.

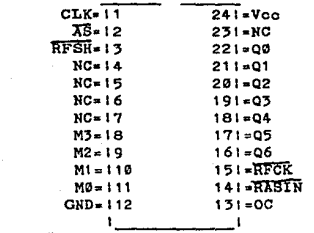


fig #29 asignacion de patas
del PAL20X8

Descripcion del circuito PAL20R6 (fig #30). Este PAL genera cinco señales basicas;

$\overline{\text{RFSH}}$ Es una señal que le llega al S409 para pedir el refresco forzado. Esta señal es generada por otras dos señales $\overline{\text{SIG}}$ y $\overline{\text{PULSE}}$.

$\overline{\text{CASL}}$ Esta es el habilitador de la dirección de columna del byte bajo. Esta puede ser buffereada externamente por los S734 antes de ir a las DRAMs.

$\overline{\text{CASU}}$ Esta es el habilitador de la direcciòn de columna del byte alto. Esta puede ser buffereada externamente ppor los S734 antes de ir a las DRAMs.

$\overline{\text{DTACKY}}$ Esta es la señal de terminaciòn de transferncia de dato para una escritura de DRAM. Esta permite escribir ciclos de cero estados de espera. Esta sëal puede ser pasada atraves de una compuerta OR para producir la señal DTACK final que llegara al 68000.

$\overline{\text{DTACKX}}$ Esta es la señal de terminaciòn de transferencia de dato para una lectura de DRAM. Esta permite ciclos de un estado de espera en ciclos de lectura. Esta señal puede ser pasada atraves de una compuerta OR para producir la señal DTACK final que llegara al 68000.

Las ecuaciones que definen estas señales son;

CASL = CAS * LDS

CASU = CAS * UDS

RFSH = $\overline{\text{AS}}$ * RFRQ + RF * RFSH + SIG * RFSH

SIG = RASX * RFSH

RF = RFRQ

DTACKX = X * AS * R/W * CS * LDS
+ X * AS * R/W * CS * UDS

$$X = \overline{RFSH} * AS * R/W * CS * LDS + RFSH * AS * R/W * CS * UDS$$

$$DTACKY = \overline{RFSH} * AS * R/W * LDS + RFSH * AS * R/W * UDS$$

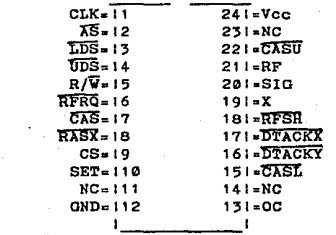


fig #30 asignacion de patas del PAL20R6

APENDICE C DETALLES DE PROGRAMACION DEL ACIA MC6850.

OPERACION DEL DISPOSITIVO; el ACIA opera como dos localidades de memoria direccionables. Internamente, tiene cuatro registros: dos registros de lectura solamente y dos de escritura solamente. Los registros de lectura solamente son el de estado y el receptor de datos; los registros de escritura solamente son el de control y el transmisor de datos. La inteface serie consiste de lineas de entrada en serie y lineas de salida en serie con relojes independientes, y tres lineas de control periferico/modem.

ENCENDIDO/RESET MAESTRO; El reset maestro (CR0, CR1) aplicado durante la inicialización del sistema asegura la condición de reset y prepara al ACIA para ser programado.

REGISTRO DE CONTROL; El registro de control del ACIA consiste de un buffer de 8 bits de escritura solamente que es seleccionado cuando RS y R/$\overline{W}$ son LOW. Este registro controla la función de el receptor, transmisor, interruptor, habilitador y el control de salida periferico/modem. Estos 8 bits se programan de la siguiente forma.

BITS SELECTORES DEL DIVISOR DEL CONTADOR (CR0 y CR1); Estos bits determinan, la frecuencia de reloj utilizada con repecto al reloj del sistema como se muestra en la siguiente tabla #7 .

CR1	CR0	FUNCION
0	0	% 1
0	1	% 16
1	0	% 64
1	1	reset maestro

TABLA #7.

BITS SELECTORES DE PALABRA (CR4, CR3 y CR2); Estos bits son usados a seleccionar longitud de palabra, paridad y el numenro de bits de STOP. El formato se elige de la siguiente forma, como lo muestra la tabla #8 .

CR4	CR3	CR2	FUNCION
0	0	0	7 bits+paridad par+2 bits de stop
0	0	1	7 bits+paridad impar+2 bits de stop
0	1	0	7 bits+paridad par+1 bit de stop
0	1	1	7 bits+paridad impar+1 bit de stop
1	0	0	8 bits+2 bits de stop
1	0	1	8 bits+1 bit de stop
1	1	0	8 bits+paridad par+1 bit de stop
1	1	1	8 bits+paridad impar+1 bit de stop

TABLA #8.

BITS DE CONTROL DE TRANSMISION (CR6 y CR5); Estos bits controlan la interrupción desde la condición del registro transmisor de datos vacio, La petición a enviar salida ($\overline{RTS}$), y la transmisión

de un nivel de interrupción. Programandolos de la siguiente manera, tabla #9.

CR6	CR5	FUNCION
Ø	Ø	RTS=LOW, interruptor de transmision deshablitado
Ø	1	RTS=LOW, interruptor de transmision habilitado
1	Ø	RTS=HIGH, interruptor de transmision deshablitado
1	1	RTS=LOW, transmitiendo un nivel de interupcion sobre el transmisor de datos de salida. interruptor de transmision deshablitado

TABLA #9.

BIT HABILITADOR DEL RECERPTOR DE INTERRUPCION (CR7); Las siguientes interrupciones seran habilitadas por un nivel HIGH en el bit 7 del registro de control (CR7): Registro receptor de datos lleno, sobrecorrimiento. o una transición LOW a HIGH sobre la linea de señal del detector de acarreo de datos ($\overline{DCD}$).

REGISTRO DE ESTADO; La infromación sobre el estado del ACIA es obtenible para el CPU leyendo el registro de estado del ACIA. Este registro de solo lectura es seleccionado cuando RS es LOW y R/$\overline{W}$ es HIGH. La información almacenada en el registro indica el estado del Registro transmisor de dato, el registro receptor de datos y error logico, y el estado de las entradas periferico/modem del ACIA.

REGISTRO RECEPTOR DE DATOS LLENO (RDRF), bit Ø; Este bit indica que el dato recibido a sido transferido al registro receptor de datos. RDRF es limpiado despues de una lectura del CPU del

registro receptor de datos o por un reset maestro. El estado limpiado o vacio indica que el contenido del registro receptor de datos no es comun. El detector de acarreo de datos estando HIGH tambien causa RDRF a indicar vacio.

REGISTRO TRANSMISOR DE DATOS VACIO (TDRE), bit 1; Cuando este bit es puesto HIGH indica que el contenido del registro transmisor de datos a sido transferido y que un nuevo dato puede ser metido. El estado LOW indica que el registro esta lleno y que la transmisión

de un nuevo caracter no comenzará sino despues del último comando de escritura de dato.

DETECTOR DE ACARREO DE DATO (DCD), bit 2; el detector de acarreo de dato sera puesto HIGH cuando la entrada $\overline{DCD}$ desde un modem a sido HIGH a indicar que el acarreo no está presente. Cuando este bit pasa a HIGH generá una petición de interrupción cuando el habilitador de recepción de interrupcioPH'n es puesto.

LIMPIAR o ENVIAR ($\overline{CTS}$), bit 3; Un estado LOW de este bit indica que hay un envio o limpia desde el modem. En el estado HIGH, el registro transmisor de datos vacio es inhibido.

ERROR DE AJUSTE (FE), bit 4; este indica que el caracter recibido esta impropiamente ajustado por un bit de inicio y un bit de final por la ausencia del primer bit de stop. Este error indica

un error de sincronización.

RECEPTOR DE SOBRECORRIMIENTO (OVRN), bit 5; este bit indica que uno o mas caracteres en el dato corriente fueron perdidos.

ERRROR DE PARIDAD (PE), bit 6; este bit indica que el numero de unos en el caracter no concuerda con la paridad seleccionada.

PETICION DE INTERRUPCION ($\overline{IRQ}$), bit 7; este bit indica la petición de una interrupción.

La distribución de terminales (fig #31) del dispositivo es la siguiente:

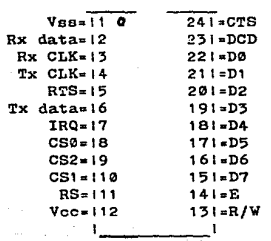


fig #31 ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER (ACIA) MC 6850.

APENDICE D PROGRAMA EN ENSAMBLADOR DEL SISTEMA Y PROGRAMA PARA SIMULACION EN LENGUAJE PASCAL.

PROGRAMA PRINCIPAL

```
        ENSAMBLADOR                COMENTARIOS
VEM   MOVE.W  #$2700,SR      deshabilita las interrupciones
      MOVE.L  #$1000,A0      inicializa apuntador de memoria RAM
      MOVE.L  #$300004,A1    inicializa contador de localidades de memoria
DNE   MOVE.L   #0,(A0)+      limpia la memoria a usar
      CMP.L    A0,A1         checa si ya se limpio toda la memoria
      BNE       DNE          si no sigue limpindola
      CLR.L     D0           limpia registro de apoyo
      CLR.L     D1             "       "     "   "
      CLR.L     D2             "       "     "   "
      CLR.L     D3             "       "     "   "
      CLR.L     D4             "       "     "   "
      CLR.L     D5             "       "     "   "
      CLR.L     D6             "       "     "   "
      CLR.L     D7             "       "     "   "
      CLR.L     A0             "       "     "   "
      CLR.L     A1             "       "     "   "
      CLR.L     A2             "       "     "   "
      CLR.L     A3             "       "     "   "
      CLR.L     A4             "       "     "   "
      CLR.L     A5             "       "     "   "
      CLR.L     A6             "       "     "   "
      MOVE.L  #$F3FF21,A0      carga la dirección del registro de control
      MOVE.L  #$F3FF23,A1      y estados, y del registro de recepción y
                               transmición del ACIA respectivamente
      MOVE.L  #$074,A3         carga subrutina para interrupción del ACIA
      MOVE.B   #$8D,(A0)       programa el ACIA para interrupciones
      MOVE.L  IPA,(A3)         manda a atender la interrupción
      MOVE.L  #$F3FF40,A1      carga la dirección de los registros de datos
      MOVE.L  #$F3FF44,A2      y control de los PIAs
      MOVE.W  #$0828,(A2)      programa el registro de control
      CLR.W     (A1)           programa los lados A como entradas
      OR.W     #$0404,(A2)     programa los PIAs para interrupciones
      MOVE.L  #$06C,A4         carga subrutina para interrupción del PIA
      MOVE.L    IPP            manda a atender la interrupción
ESP   MOVE.W  #$2000,SR        programa al CPU para recibir interrupciones
      BRA       ESP            espera la interrupción
IPA   MOVE.L  #$F3FF21,A0      carga la dirección del registro de control
      MOVE.L  #$F3FF23,A2      y estados, y del registro de recepción y
                               transmición del ACIA respectivamente
      MOVE.L   #$2C24,A1       inicializa apuntador de memoria para guardar
                               la matriz de la imagen a procesar
      MOVE.W    #0,D5          inicializa contador de datos
      MOVE.B  #$0D,(A0)        programa el ACIA para busqueda
```

```
OTR   MOVE.B   #Ø,D1          inicializa contador de digitos
      CLR.L     D2           limpia registro de apoyo
      CLR.L     D4              "       "     "   "
      CLR.L     D3              "       "     "   "
SCE   CLR.L     DØ              "       "     "   "
      MOVE.B  (AØ),DØ          salva el registro de estados del ACIA
      AND.B   #1,DØ            deja prendido solo en bit #Ø
      CMP.B   #1,DØ            checa si esta prendido ese bit
      BNE      SCE             sigue esperando a que llegue el digito
      MOVE.B  (A2),D2          salva el digito en el registro D2
      ADD.B   #1,D1            incrementa el contador de digitos
      CMP.B   #1,D1            checa si es el primer digito
      BEQ      SD1             ve a salvar el primer digito
      CMP.B   #2,D1            checa si es el segundo digito
      BEQ      SD2             ve a salvar el segundo digito
      CMP.B   #3,D1            checa si es el tercer digito
      BEQ      SD3             ve a salvar el tercer digito
SD1   MOVE.B  D2,D4            salva el primer digito
      BRA      OTR             ve por otro digito
SD2   MOVE.B  D2,D3            salva segundo digito
      MULU    #10,D3           incrementa la posición del digito
      ADD.W   D3,D4            suma los dos digitos
      CLR.L    D3              limpia registro de apoyo
      BRA      OTR             ve por otro digito
SD3   MOVE.B  D2,D3            salva el tercer digito
      MULU   #100,D3           incrementa la posición del digito
      ADD.W   D3,D4            suma el tercer digito
      CLR.L    D3              limpia registro de apoyo
      MOVE.L  D4,(A1)+         salva el dato en memoria
      ADD.W   #1,D5            incrementa el contador de datos
      CMP.W   #512,D5          checa si ya se recibieron todos los datos
      BGT      OTR             si no ve por otro dato
      JSR      FFT             ve a realizar la subrutina FFT
      MOVE.L  #$2C24,AØ        inicializa apuntador para devolver datos
      MOVE.L  #$204C24,A1      apuntador de exponentes de datos
      MOVE.W   #Ø,DØ           inicializa contador de datos
DDØ   MOVE.B   #Ø,D1           inicializa contador de digitos
MDA   MOVE.B (AØ)+,(A2)        manda primeros digitos
      ADD.B   #1,D1            incrementa contador de digitos
      CMP.B   #4,D1            checa si ya se mandaron cuatro bytes
      BNE      MDA             si no sigue mandando bytes
      MOVE.B  (A1)+,(A2)       manda el exponente del dato transmitido
      ADD.W   #1,DØ            incrementa el contador de datos
      CMP.W   #512,DØ          checa si son todos los datos
      BGT      DDØ             sigue mandando datos
      BRA      VEM             vuelve a esperar otra matriz
IPP   MOVE.L #$F3FF40,A1       carga la dirección de los registros
                               de datos
      MOVE.L  #$F3FF44,A2      y control de los PIAs
      MOVE.W  #$Ø828,(A2)      programa el registro de control
      CLR.W     (A1)           programa los lados A como entradas
      MOVE.L  #$2C24,A3        inicializa apuntador para recibir datos
      MOVE.W  #$Ø404,(A2)      programa los PIAs para busqueda
```

```
        MOVE.W  #0,D0           inicializa contador de datos
MDT     CLR.L     D1            limpia registro de apoyo
        MOVE.W  (A1),D1         salva el dato a almacenar
        MOVE.L  D1,(A3)+        salva el dato en memoria
        ADD.W   #1,D0           incrementa el contador de datos
        CMP.W   #512,D0         checa si son todos los datos
        BGT      MDT            si no ve por más datos
        JSR      FFT            ve a ejecutar la subrutina FFT
        MOVE.L  #$F3FF40,A1     carga la dirección de los registros
                                de datos
        MOVE.L  #$F3FF44,A2     y control de los PIAs
        MOVE.W  #$0828,(A2)     programa el registro de control
        MOVE.W  #$FF,(A1)       programa los lados B como salidas
        MOVE.W   #$0404,(A2)    programa los PIAs para dialogo
        MOVE.L  #$2C24,A3       inicializa apuntador de datos en memoria
        MOVE.L  #$204C24,A4     inicializa apuntador de exponentes
        MOVE.W  #0,D0           inicializa contador de datos
MMD     MOVE.L  (A3)+,(A1)      manda los datos del resultado
        MOVE.B  (A4)+,(A1)      manda el exponente del dato
        ADD.W   #1,D0           incrementa el contador de datos
        CMP.W   #512,D0         checa si ya son todos los datos
        BGT      MMD            si no sigue mandando datos
        BRA      VEM            ve a esperar otra matriz de datos
```

SUBRUTINA PARA REALIZAR FFT

```
          ENSAMBLADOR                COMENTARIOS
FFT     MOVE.L #$1200,A4        inicializa apuntador de buffer de apoyo
        MOVE.L #$1A00,A5             "          "       "    "    "
        MOVE.L #$2200,A6             "          "       "    "    "
        MOVE.L #$27512,D7            "          "       "    "    "
        MOVE.L #$2C24,A0             "          "     para datos reales
        MOVE.L #$102C24,A1           "          "      "    "   imagimarios
        MOVE.L #$202C24,A2           "          "    de exponentes de reales
        MOVE.L #$204C24,A3           "          "    "    "  de imaginarios
CPR     MOVE.W  #512,D6         FOR I:=1 TO N DO
WWP     MOVE.W  #512,D5         FOR J:=1 TO N DO
CVT     MOVE.L (A0)+,(A4)+      carga los valores reales
        MOVE.L (A1)+,(A5)+        "    "      "    imaginarios
        MOVE.B (A2)+,(A6)+        "    "      "  de exponentes reales
        MOVE.B (A3)+,(D7)+        "    "      "    "   imaginarios
        SUB.W    #1,D5          decrementa el bucle for J
        CMP.W    #0,D5          checa si ya se realizo todo el bucle
        BLT      CVT            END; (FOR J)
        MOVEM.L A0-A3,-(A7)     salva registros apuntadores
        JSR       IMA           realiza subrutina imagen
        MOVEM.L (A7)+,A0-A3     recupera registros apuntadores
        MOVEM.L A0-A3,-(A7)     salva registros apuntadores
        MOVE.W  #512,D5         FOR J:=1 TO N DO
```

```
REC  MOVE.L -(A4),-(A0)       recupera datos reales transformados
     MOVE.L -(A5),-(A1)          "        "  imaginarios    "
     MOVE.B -(A6),-(A2)          "    exponentes reales
     MOVE.B -(D7),-(A3)          "          "     imaginarios
     SUB.W   #1,D5             decrementa el bucle for J
     CMP.W   #0,D5             checa si ya se realizo todo el bucle
     BLT      REC              END; (for J)
     MOVEM.L (A7)+,A0-A3       recupera apuntadores
     SUB.W    #1,D6            decrementa bucle for I
     CMP.W    #0,D6            checa si ya se realizo todo el bucle
     BLT       WWP             END; (for I)
     MOVE.L  #$2C24,A0         inicializa apuntador de buffer para reales
     MOVE.L #$102C24,A1         "         "       "      "   imaginarios
     MOVE.L #$202C24,A2            "          "       " exponentes reales
     MOVE.L #$242C24,A3            "          "       "      "  imaginarios
     MOVE.W  #512,D6           FOR I:=1 TO N DO
SA3  MOVE.L #$1200,A4          inicializa apuntador para la parte real
     MOVE.L #$1A00,A5             "          "       "     "   " imaginaria
     MOVE.L #$2200,A6             "          "   de exponentes reales
     MOVE.L #$2400,D7             "          "   "       " imaginarios
     MOVE.W #512,D5.           FOR J:=1 TO N DO
SN4  MOVE.L 200(A0)+,(A4)+     recupera elementos de columna reales
     MOVE.L 200(A1)+,(A5)+        "          "       "     " imaginarios
     MOVE.B 200(A2)+,(A6)+        "    los exponentes reales
     MOVE.B 200(A3)+,(D7)+        "     "       "  imginarios
     SUB.W    #1,D5            decrementa el bucle for J
     CMP.W    #0,D5            checa si ya se ejecuto todo el bucle
     BLT      SN4              END; (for J)
     MOVEM.L A0-A3,-(A7)       salva registros de apoyo
     JSR      IMA              ejecuta subrutina imagen
     MOVE.W  #512,D5           FOR J:=1 TO N DO
MGD  MOVE.L -(A4),-200(A0)     regresa columna real procesada
     MOVE.L -(A5),-200(A1)        "        " imaginario  "
     MOVE.B -(A6),-200(A2)        "   exponentes reales
     MOVE.B -(D7),-200(A3)        "        "  imaginarios
     SUB.W    #1,D5            decrementa el bucle for J
     CMP.W    #0,D5            checa si ya se ejecuto todo el bucle
     BLT      MGD              END; (for J)
     MOVEM.L (A7)+,A0-A3       recupera registros de apoyo
     SUB.W     #1,D6           decrementa el bucle for I
     CMP.W     #0,D6           checa si ya se ejecuto todo el bucle
     BLT       SA3             END; (for I)
MDM  MOVE.L #$2C24,A0          inicializa apuntador para guardar reales
     MOVE.L #$102C24,A1          "         "       "     " imaginarios
     MOVE.L #$202C24,A2           "          "  de exponentes reales
     MOVE.L #$242C24,A3           "          "   "       " imaginarios
     MOVE.L    A0,A4           copia A0 en A4
     MOVE.L    A1,A5             "   A1   " A5
     MOVE.L    A2,A6             "   A2   " A6
     MOVE.L    A3,D7             "   A3   " D7
     MOVE.W   #512,D6          FOR I:=1 TO N DO
GHR  MOVE.L (A0)+,$1000        prepara parte real para elevarla
                               al cuadrado
```

```
MOVE.B (A2)+,$1004      prepara su exponente
MOVE.L (A4)+,$1006      prepara parte real para elevarla
MOVE.B (A6)+,$100A      prepara su exponente
JSR     MULU 32         ejecuta subrutina de multiplicación
MOVE.L $1000,-(A4)      recupera el cuadrado de la parte real
MOVE.B $1004,-(A6)          "     su exponente
MOVE.L (A1)+,$1000      ahora prepara la parte imaginaria
MOVE.B (A3)+,$1004        "       "     su exponente
MOVE.L (A5)+,$1006        "       "     la parte imaginaria
MOVE.B (D7)+,$100A        "       "     su exponente
JSR     MULU 32         ejecuta subrutina de multiplicación
MOVE.L (A4)+,$1006      prepara la parte real para la suma con
MOVE.B (A6)+,$100A      la parte imaginaria
JSR     SUMA 32         ejecuta la suma de los cuadrados
MOVE.L $1000,-(A4)      guarda el primer elemento del modulo
MOVE.B $1004,-(A6)      de la matriz transformada
MOVE.L   A0,A4          recupera  el valor del apuntador A4
MOVE.L   A2,A6              "     "   "    "       "     A6
SUB.W    #1,D6          decrementa el bucle for I
CMP.W    #0,D6          checa si ya se ejecuto todo el bucle
BLT      GHR            END; (for I)
RTS                     regresa de la subutina FFT
```

SUBRUTINA IMAGEN

```
     ENSAMBLADOR                COMENTARIOS
IMA  MOVE.L #$12A1B0A1,$1100    memoriza el valor de Pi
     MOVE.B #$F8,$1104          memoriza el exponente de Pi
     MOVE.W #$100,$1106         almacena el valor de n2=256
     MOVE.B   #8,$1166          inicializa N1=8
     MOVE.W #$0,$1108           limpia K=0
     MOVE.B  #9,$1164           FOR I:=1 TO N0 DO
     MOVE.L #$1200,A0           inicializa apuntador para guardar reales
     MOVE.L #$2200,A2                "          "  de exponentes reales
     MOVE.L #$1A00,A1               "          "     " imaginarios
     MOVE.L #$2400,A3                "    "     "     " imaginarios
     MOVE.L #$1200,A4           inicializa apuntador para la parte real
     MOVE.L #$1A00,A5            "          "     "    " imaginaria
     MOVE.L #$2200,A6             "          "   de exponentes reales
     MOVE.L #$2400,D7             "          "   "       " imaginarios
SE4  CMP.W $1108,#512           WHILE   K < N
     BGT       WER              ejecuta lo que sigue mientras K < N
     MOVE.B   $1106,D0          FOR I:=1 TO N2 DO
KJY  MOVE.L    #2,D1            las diez  instruciones proximas lo que
     MOVE.L   D1,$1000          hacen es obtener el valor de J y
     MOVE.B   #0,$1004          almacenarlo en localidad reservada para
     MOVE.B $1166,$100C       esta variable que es la dirección $110A
```

```
JSR     POT 32              el valor de J se calcula por medio de la
MOVE.W $1000,D1             subrutina POT 32 la cual eleva un numero
CLR.L     D2                a una potencia indicada utilizando
MOVE.W $1108,D2             J=int(K/2 elevado a la N1)
DIVU      D1,D2
MOVE.W  D2,$110A
JSR       ANA               ejecuta la subrutina analiza
CLR.L   $1110               localida reservada para la variable A1
CLR.B   $1114                   "          "       "  su exponente
MOVE.L $1100,$1110        las proximas cuatro instruciones obtienen
MOVE.B $1104,$1114        la igualdad A1= 2Pi, sumando dos veces la
ADD.L   $1100,$1110         magnitud de Pi y dos veces su exponente
ADD.B   $1104,$1114
CLR.L     D1              las proximas siete instrucciones obtienen
MOVE.W   $110E,D1           el valor del producto 2P1P por medio de
MOVE.L   D1,$1000           la subrutina MULU 32
MOVE.B   #0,$1004
MOVE.L $1110,$1006
MOVE.B $1114,$100A
JSR       MULU 32
MOVE.L $1000,$1006        las proximas nueve instrucciones obtienen
MOVE.B $1004,$100A          el valor final de la variable A1=2P1P/N
CLR.L     D1
MOVE.W   #512,D1
MOVE.L    D1,$1000
MOVE.B    #0,$1004
JSR       DIVU 32
MOVE.L $1000,$1110
MOVE.B $1004,$1114
JSR     COS 32              obten el valor  C=cos(A1)
MOVE.L $1000,$1116          salva el valor de C en la dirección
MOVE.B $1004,$111A          $1116 y su exponente en $111A
MOVE.L $1110,$1000          las siguientes cinco instrucciones
MOVE.B $1114,$1004          calculan el valor de S=sen(A1) y salvan
JSR       SEN 32            el valror del seno  calculado en las
MOVE.L $1000,$111C          direcciones $111C para el valor y en la
MOVE.B $1004,$1120    direccion $1120 para el valor de su exponente
CLR.L     D1                limpia registro de apoyo
MOVE.W $1108,D1           las siguientes tres instrucciones calculan
ADD.W     #1,D1           la igualdad K1 = K + 1 almacenando el valor
MOVE.W   D1,$112            de K1 en la localidad $1122
ADD.W $1106,D1            estas dos instruccines realizan la igualdad
MOVE.W   D1,$1124               k2 = k1 + n2
MOVE.L #$1200,A0            inicializa apuntador para guardar reales
MOVE.L #$2200,A2                 "          "  de exponentes reales
MOVE.L #$1A00,A1                 "          "       " imaginarios
MOVE.L #$2400,A3                 "          "       " imaginarios
ADD.L   D1,A0             las proximas ocho instrucciones localizan
ADD.L   D1,A1             en que lugar del buffer se encuentran los
ADD.L   D1,A2               valores X[k2], Y[K2] y sus respectivos
ADD.L   D1,A3               exponentes, memorizando estos lugares en
MOVE.L A0,$1144             localidades reservadas par tal fin
MOVE.L A1,$1148
```

```
MOVE.L A2,$115C
MOVE.L A3,$1160
MOVE.L (A0),$1128        estas dos instrucciones obtienen los
MOVE.B (A2),$112A       valores de X[K2] y el de su exponente
MOVE.L (A1),$112C         estas dos instrucciones obtienen los
MOVE.B (A3),$1130       valores  de Y[K2] y el de su exponente
MOVE.L (A0),$1000      las siguientes siete instrucciones calculan
MOVE.B (A2),$1004         el producto X[K2]*C y lo almacenan el la
MOVE.L $1116,$1006        localidad reservada para la variable T1
MOVE.B $111A,$100A
JSR    MULU 32
MOVE.L $1000,$1132
MOVE.B $1004,$1136
MOVE.L (A1),$1000      las siguientes diez  instrucciones calculan
MOVE.B (A3),$1004         el producto Y[K2]*S y se lo suman a  la
MOVE.L $111C,$1006        localidad reservada para la variable T1
MOVE.B $1120,$100A        obteniendo la igualdad
JSR    MULU 32            T1 = X[K2]*C + Y[K2]*S
MOVE.L $1132,$1006
MOVE.B $1136,$100A
JSR    SUMA 32
MOVE.L $1000,$1132
MOVE.B $1004,$1136
MOVE.L (A1),$1000      las siguientes siete instrucciones calculan
MOVE.B (A3),$1004         el producto Y[K2]*C y lo almacenan el la
MOVE.L $1116,$1006        localidad reservada para la variable T2
MOVE.B $111A,$100A
JSR    MULU 32
MOVE.L $1000,$1138
MOVE.B $1004,$113C
MOVE.L (A0),$1000       las siguientes once instrucciones calculan
MOVE.B (A2),$1004         el producto X[K2]*S y se lo suman a  la
MOVE.L $111C,$1006        localidad reservada para la variable T2
MOVE.B $1120,$100A        obteniendo la igualdad
JSR    MULU 32            T2 = Y[K2]*C - X[K2]*S
MOVE.L $1132,$1006
MOVE.B $1136,$100A
NEG.L  $1000
JSR    SUMA 32
MOVE.L $1000,$1138
MOVE.B $1004,$113C
MOVE.L #$1200,A0          inicializa apuntador para guardar reales
MOVE.L #$2200,A2               "          "  de exponentes reales
MOVE.L #$1A00,A1              "          "     " imaginarios
MOVE.L #$2400,A3               "          "    " imaginarios
CLR.L    D1               limpia registro de apoyo
MOVE.W  $1122,D1          carga la direcciòn de K1 en D1
ADD.L  D1,A0             las proximas ocho instrucciones localizan
ADD.L  D1,A1             en que lugar del buffer se encuentran los
ADD.L  D1,A2              valores X[k1], Y[K1] y sus respectivos
ADD.L  D1,A3              exponentes, memorizando estos lugares en
MOVE.L (A0),$1000      estas ocho instrucciones obtienen los valores
MOVE.B  (A2),$1004    de X[K1] y el de su exponente y recuperando el
```

```
        MOVE.L $1132,$1006        valor de la variable T1 se realiza la
        MOVE.B $1136,$100A        operación aritmetica
        NEG.L  $1006              X[K2] = X[K1] - T1
        JSR    SUMA 32
        MOVE.L $1000,$1128
        MOVE.B $1004,$112A
        MOVE.L (A1),$1000       las siguientes ocho  instrucciones calculan
        MOVE.B (A3),$1004         la operación aritmetica siguiente
        MOVE.L $113B,$1006        Y[K2] = Y[K1] - T2
        MOVE.B $118C,$100A
        NEG.L  $1006
        JSR    SUMA 32
        MOVE.L $1000,$112C
        MOVE.B $1004,$1130
        MOVE.L (A0),$1000         estas nueve instrucciones siguientes
        MOVE.B (A2),$1004         realizan la operación aritmetica
        MOVE.L $1132,$1006        siguiente
        MOVE.B $1136,$100A        X[k1] = X[k1] + T1
        JSR    SUMA 32
        MOVE.L $1000,$113E
        MOVE.B $1004,$1142
        MOVE.L $1000,(A0)
        MOVE.B $1004,(A2)
        MOVE.L (A1),$1000         las nueve instrucciones siguientes
        MOVE.B (A3),$1004         calculan la operación aritmetica
        MOVE.L $113B,$1006        siguiente
        MOVE.B $113C,$100A        Y[k1] = Y[k1] + T2
        JSR    SUMA 32
        MOVE.L $1000,$1144
        MOVE.B $1004,$1148
        MOVE.L $1000,(A1)
        MOVE.B $1004,(A3)
        ADD.W   #1,$1108            K = K + 1
        SUB.W   #1,D0             decrementa el bucle for I
        CMP.W   #0,D0             checa si ya se realiza todo el bucle
        BLT     KJY               END; (for I)
        ADD.W  $1106,$1108         K = K + n2
        BRA     SE4               ve a checar si el bucle WHILE
WER     CLR.W   $1108              K = 0
        SUB.W   #1,$1166           n1 = n1 -1
        CLR.L    D1               limpia registro de apoyo
        MOVE.W  $1106,D1           recupera el valor de n2
        DIVU    #2,D1              n2 = n2 div 2
        MOVE.W  D1,$1106           salva el nuevo valor de n2
        SUB.B   #1,$1164           decrementa el bucle for I
        CMP.B   #0,$1164           checa si ya se ralizo todo
        BLT      SE4               END; (for I)
        MOVE.W  #512,D0            FOR K:=1 TO N DO
SRT     MOVE.W $1108,D1          estas tres instrucciones proximas realizan
        SUB.W   #1,D1              la operación aritmetica siguiente
        MOVE.W  D1,$1114           J = K - 1
        JSR      ANA               ejecuta la subrutina analiza
        MOVE.W  $110E,D1           i = P
```

```
ADD.W   #1,D1              i = P + 1
CMP.W    D1,$1108          IF i > K THEN
BLE       SQT               sino se cumple la condición brinca
CLR.L     D2                   limpia registro de apoyo
MOVE.W  1108,D2             carga el valor de K en D2
MOVE.L #$1200,A0          inicializa apuntador para guardar reales
MOVE.L #$2200,A2                "          "  de exponentes reales
MOVE.L #$1A00,A1              "          "       "     " imaginarios
MOVE.L #$2400,A3                "          "    "      " imaginarios
ADD.L   D2,A0             las proximas ocho instrucciones localizan
ADD.L   D2,A1             en que lugar del buffer se encuentran los
ADD.L   D2,A2               valores X[k], Y[K] y sus respectivos
ADD.L   D2,A3              exponentes, memorizando estos lugares en
MOVE.L A0,$1174             localidades reservadas par tal fin
MOVE.L A1,$1178
MOVE.L A2,$117C
MOVE.L A3,$1180
MOVE.L (A0),$1168      estas dos instrucciones obtienen los valores
MOVE.B (A2),$116C           de T1 = X[K] y el de su exponente
MOVE.L (A1),$116E      estas dos instrucciones obtienen los valores
MOVE.B (A3),$1172           de T2 = Y[K] y el de su exponente
MOVE.L #$1200,A0          inicializa apuntador para guardar reales
MOVE.L #$2200,A2                "          "  de exponentes reales
MOVE.L #$1A00,A1              "          "       "     " imaginarios
MOVE.L #$2400,A3                "          "    "      " imaginarios
ADD.L   D1,A0             las proximas ocho instrucciones localizan
ADD.L   D1,A1             en que lugar del buffer se encuentran los
ADD.L   D1,A2               valores X[i], Y[i] y sus respectivos
ADD.L   D1,A3              exponentes, memorizando estos lugares en
MOVE.L A0,$1190             localidades reservadas par tal fin
MOVE.L A1,$1194
MOVE.L A2,$1198
MOVE.L A3,$119C
MOVE.L (A0),$1184      estas dos instrucciones obtienen los valores
MOVE.B (A2),$1188           de X[i] y el de su exponente
MOVE.L (A1),$118A      estas dos instrucciones obtienen los valores
MOVE.B (A3),$118E           de Y[i] y el de su exponente
MOVE.L #$1174,A0          inicializa apuntador para guardar reales
MOVE.L #$1178,A2                "          "  de exponentes reales
MOVE.L #$117C,A1              "          "       "     " imaginarios
MOVE.L #$1180,A3                "          "    "      " imaginarios
MOVE.L  $1184,(A0)         estas dos instrucciones realiza la opera
MOVE.B  $1188,(A2)          ción  X[K] = X[i]
MOVE.L  $118A,(A1)         estas dos instrucciones realiza la opera
MOVE.B  $118E,(A3)          ción  Y[K] = Y[i]
MOVE.L #$1190,A0          inicializa apuntador para guardar reales
MOVE.L #$1194,A2                "          "  de exponentes reales
MOVE.L #$1198,A1              "          "       "     " imaginarios
MOVE.L #$119C,A3                "          "    "      " imaginarios
MOVE.L  $1168,(A0)         estas dos instrucciones realiza la opera
MOVE.B  $116C,(A2)          ción  X[i] = T1
MOVE.L  $116E,(A1)         estas dos instrucciones realiza la opera
MOVE.B  $1172,(A3)          ción  Y[i] = T2
```

```
SQT   SUB.W    #1,D0          decrementa el bucle for K
      CMP.W    #0,D0          checa si ya se realizo todo el bucle
      BLT      SRT            END; (for K)
      RTS                     regresa de la subrutina IMAGEN
```

SUBRUTINA ANALIZA

```
        ENSAMBLADOR              COMENTARIOS
ANA   MOVE.W  #0,$110C         J  = 0
      MOVE.W  $10A,$110C       J1 = J
      MOVE.W    #0,$110E       P = 0
      MOVE.B   #9,D1           FOR H:=1 TO N0 DO
UYR   CLR.L     D2                 limpia registro de apoyo
      CLR.L     D3                 limpia registro de apoyo
      MOVE.W  $110C.D2       estas proximas tres instrucciones calculan
      DIVU     #2,D2           la expresión siguiente
      MOVE.W  D2,$1110         J2 = J1 DIV 2
      MOVE.W  $110E,D2       las proximas ocho instrucciones calculan
      ADD.W   D2,$110E         la expresión algebraica siguiente
      ADD.W   $110C,$110E      P = 2*P + J1 - 2*J2
      MOVE.W  $1110,D2
      MOVE.W    D2,D3
      ADD.W     D2,D3
      SUB.W    D3,$110E
      MOVE.W  $1110,$110C
      SUB.W   #1,D1            decrememta el bucle for H
      CMP.W    #0,D1           checa si ya se realizo todo el bucle
      BLT        UYR           END; (for H)
      RTS                      regresa de la subrutina analiza
```

BIBLIOTECA DE FUNCIONES

Surutina para realizar la suma de dos numeros de 32 bits signados "SUMA 32", el valor del primer sumando debe ser colocado en la direcciòn de memoria $1000 y su exponente en la direcciòn $1004 mientras que el segundo sumando debe ser colocado en la direcciòn $1006 y su exponente en la direcciòn $100A, el resultado de la suma se regresa en la direcciòn del primer sumando.

```
        ENSAMBLADOR              COMENTARIOS
      MOVEM.L D0-D5,-(A7)    salva registros de apoyo
      CLR.L     D0           limpia registro de apoyo
      CLR.L     D1             "       "     "   "
```

```
        CLR.L     D2              "       "     "   "
        CLR.L     D3              "       "     "   "
        CLR.L     D4              "       "     "   "
        CLR.L     D5              "       "     "   "
        MOVE.L  $1000,D0        carga el primer sumando en D0
        MOVE.B  $1004,D1        carga el valor de su exponente
        MOVE.L  $1006,D2        carga el segundo sumando en D2
        MOVE.B  $100A,D3        carga el valor de su exponente
        CMP.B     #0,D1         compara si es negativo el exponente de S1
        BGT       SUM2          brinca a comparar el exponente de S2
        CMP.B     #0,D3         compara si es negativo el exponente de S2
        BGT       SUM3          brinca a compara cual de los dos es mayor
        CMP.B     D1,D3         compara si son iguales
        BNE       COM1          si no brinca a comparar cual es mayor
        ADD.L  $1006,$1000      realiza la suma directamente
        MOVEM.L (A7)+,D0-D5     recupera registros de apoyo
        RTS                     regresa de la subrutina
COM1    MOVE.B    D3,D4         salva el valor del segundo exponente en D4
        SUB.B     D1,D3         compara cual es mayor
        MOVE.W    SR,D5         copia el SR en el registro D5
        AND.W     #8,D5         deja solo el bit N
        CMP.W     #8,D5         checa si esta encendido
        BNE       IGU1          brinca a igualar los exponentes
        MOVE.B    D1,D5         salva el primer exponente en D5
        MOVE.B    D4,D3         recupera el segundo exponente
        SUB.B     D3,D1         calcula la diferencia de tamaños
        ADD.B     D1,D5         iguala exponentes
        MULU      #$A,D1        calcula el numero de recorrimintos de S1
        DIVS      D1,D0         ejecuta los recorrimientos
        ADD.L     D0,D2         realiza la suma
        MOVE.L  D2,$1000        regresa el valor de la suma
        MOVE.B  D5,$1004        regresa el valor del exponente
        MOVEM.L (A7)+,D0,D5     recupera registros de apoyo
        RTS                     regresa de la subrutina
IGU1    MOVE.B    D1,D5         salva el primer exponente en D5
        MOVE.B    D4,D3         recupera el segundo exponente
        SUB.B     D1,D3         compara cual es mayor
        ADD.B     D3,D4         iguala los exp[onentes
        MULU      #$A,D3        calcula el numero de recorrimientos de S2
        DIVS      D3,D2         ejecuta el numero de recorrimientos
        ADD.L     D0,D2         realiza la suma
        MOVE.L  D2,$1000        regresa el resultado
        MOVE.B  D4,$1004        regresa el exponente
        MOVEM,L (A7)+,D0-D5     recupera registros de apoyo
        RTS                     regresa de la subrutina
SUM2    CMP.B     #0,D3         compara si es negativo el exponente
        BGT       SUM4          brinca a comparar cual es mayor
        MOVE.B    D3,D4         salva el segundo exponente en D4
        SUB.B     D1,D3         iguala los exponentes
        MULU      #$A,D3        calcula el numero de recorrimientos de S2
        DIVS      D3,D2         iguala los sumandos
        ADD.L     D0,D2         realiza la suma
        MOVE.L  D2,$1000        regrea el valor de la suma
```

```
      MOVE.B  D4,$1004        regresa el valor del exponente
      MOVEM.L (A7)+,D0-D5     recupera registros de apoyo
      RTS                     regresa de la subrutina
SUM3  MOVE.B    D1,D5         salva el primer exponente en D5
      SUB.B     D3,D1         iguala los exponentes
      MULU      #$A,D1        calcula el numero de recorrimientos de S1
      DIVS      D1,D0         iguala los sumandos
      ADD.L     D0,D2         realiza la suma
      MOVE.L  D2,$1000        regresa el valor de la suma
      MOVE.B  D5,$1004        regresa el valor del exponente
      MOVEM.L (A7)+,D0-D5     recupera registros de apoyo
      RTS                     regresa de la subrutina
SUM4  MOVE.B      D3,D4       salva el segundo exponente en D4
      SUB.B       D1,D3       compara cual de los exponentes es mayor
      MOVE.W      SR,D5       salva el SR en D5
      AND.W       #8,SR       deja el bit N
      CMP.W       #8,D5       prueba si esta encendido
      BNE         IGU2        ve a igualar los exponentes
      MOVE.B      D1,D5       salva el primer exponente en D5
      SUB.B       D3,D1       iguala los exponentes
      MULU        #$A,D1      calcula recorrimiento de S1
      DIVS        D1,D0       iguala los sumandos
      ADD.L       D0,D2       realiza la suma
      MOVE.L    D2,$1000      regresa el valor de la suma
      MOVE.B    D5,$1004      regresa el valor del exponente
      MOVEM.L (A7)+,D0-D5     recupera rgristos de apoyo
      RTS                     regresa de la subrutina
IGU2  MOVE.B      D4,D3       recupera exponente de S2 en  D3
      SUB.B       D1,D3       iguala los exponentes
      MULU        #$A,D3      calcula el recorrimiento de S2
      DIVS        D3,D2       iguala los sumandos
      ADD.L       D0,D2       realiza la suma
      MOVE.L    D2,$1000      regresa el valor de la suma
      MOVE.B    D4,$1004      regresa el valor del exponente
      MOVEM.L (A7)+,D0-D5     recupera registros de apoyo
      RTS                     regresa de la subrutina
```

Subrutina para la multiplicacion de dos numeros de 32 bits signados "mulu 32" , el valor del multiplicando se debe colocar en la direcciòn $1000 y su exponente en la direcciòn $1004, mientras que el valor del multip[licador debe sr colocado en la direcciòn $1006 y el de su exponente en la direcciòn $100A, y el resultado de la multiplicaciòn se regresa en la misma direcciòn del multiplicando.

```
     ENSAMBLADOR              COMENTARIOS
    MOVEM.L D1-D6,-(A7)  salva registros de apoyo
    MOVE.L   $1000,D1    carga el multiplicando en D1
    MOVE.L   $1006,D2    carga el multiplicador en D2
    CLR.B     D6         limpia D6
```

```
        TST.L    D1          multiplicando negativo
        BPL      CHD2        brinca a CHD2
        NEG.L    D1          convierte lo a positivo
        NOT.B    D6          cambia contador de signo
CHD2    TST.L    D2          multiplicador negativo
        BPL      MULT        brinca a MULT
        NEG.L    D2          convierte lo a positivo
        NOT.B    D6          cambia contador de signo
        MOVE.L   D1,D3       copia multiplicando en D3
        MOVE.L   D1,D4       copia multiplicador en D4
        SWAP     D4          prepara multiplicación parcial
        MOVE.L   D2,D5       copia multiplicador en D5
        SWAP     D5          prepara multiplicación parcial
        MULU     D2,D1       producto parcial #1
        MULU     D4,D2       producto parcial #2
        MULU     D5,D3       producto parcial #3
        MULU     D5,D4       producto parcial #4
        SWAP     D1          prepara suma parcial #1
        ADD.W    D2,D1       suma 1=p.p.2(bajo)+p.p1(alto)
        CLR.L    D5          registro propagador de "carry"
        ADDX.L   D5,D4       propaga "carry" a D4
        ADD.W    D3,D1       suma 2=suma 1+p.p.3(bajo)
        ADDX.L   D5,D4       propaga "carry" a D4
        SWAP     D1          prepara producto parcial #1
        CLR.W    D2          elimina parte tomada de p.p.2
        SWAP     D2          prepara suma parcial #3
        CLR.W    D3          prepara suma parcial #3
        ADD.L    D3,D2       suma 3=p.p.2(alto)+p.p.3(alto)
        ADD.L    D4,D2       suma 4=suma 3+p.p.4
        MOVE.L   D2,D3       copia los bits más significativos en D3
        SWAP     D3          invierte el orden de las dos palabras
        ROL.W    #8,D3       invierte el orde de los dos ultimos byts
        CMP.B    #0,D3       compara si el byte #8 es cero
        BEQ      ROL1        si SI brinca a comparar el byte #7
        AND.B    #$F,D3      limpia los 4 bits más significativos
        CMP.B    #0,D3       comparalos si son cero
        BNE      RE28        brinca a hacer 28 recorrimientos
        MOVE.B   #$32,D4     carga el numero de recorrimientos en D4
        BRA      EJR         ejecuta recorrimientos indicados en D4
RE28    MOVE.B   #$28,D4     carga el numero de recorrimientos en D4
        BRA      EJR         ejecuta recorrimientos indicados en D4
ROL1    ROL.W    #8,D3       reordena el orden de los ultimos byts
        CMP.B    #0,D3       compara si el byte #7 es cero
        BEQ      ROL2        si SI brinca a comparar el byte #6
        AND.B    #$F,D3      limpia los cuatro bits más significativos
        CPM.B    #0,D3       compara si son cero
        BNE      RE20        brinca a hacer 20 recorrimientos
        MOVE.B   #$24,D4     carga el numero de recorrimientos en D4
        BRA      EJR         ejecuta recorrimientos indicados en D4
RE20    MOVE.B   #$20,D4     carga numero de recorrimientos en D4
        BRA      EJR         ejecuta recorrimientos indicados en D4
ROL2    SWAP     D3          reordena el orden de las palabras
        ROL.W    #8,D3       invierte el orden de los byts #6 y #5
```

```
        CMP.B   #0,D3          compara si el byte #6 es cero
        BEQ     ROL3           si SI brinca a comparar el byte #5
        AND.B   #$F,D3         limpia los cuator bits más significativos
        CMP.B   #0,D3          compara si son cero
        BNE     RE12           brinca a hacer 12 recorrimientos
        MOVE.B  #$16,D4        carga el numero de recorrimienots en D4
        BRA     EJR            brinca a ejecutar recorrimientos
RE12    MOVE.B  #$12,D4        carga el numero de recorrimientos en D4
        BRA     EJR            ejecuta recorrimientos
ROL3    ROL.W   #8,D3          reordena los byts #6 y #5
        CMP.B   #0,D3          compara si el byte #5 es cero
        BEQ     ROL4           si SI brinca a realizar cero recorrimientos
        AND.B   #$F,D3         limpia los cuatro bits más significativos
        CMP.B   #0,D3          compara si son cero
        BNE     RE4            brinca a realizar 4 recorrimientos
        MOVE.B  #8,D4          carga el numero de recorrimientos en D4
        BRA     EJR            ejecuta recorrimientos
RE4     MOVE.B  #4,D4          carga el numero de recorrimientos en D4
        BRA     EJR            ejecuta el recorrimiento
ROL4    MOVE.B  #0,D4          carga el numero de recorrimientos en D4
EJR     MOVE.B  D4,D5          carga el numero de recorrimientos en D5
        DIVU    #4,D5          obten el exponente del resultado
ROX     ROXR.L  #1,D2          saca el bit menos significativo hacia el bit X
        ROXL.L  #1,D1          metelo en el bit más significativo de D1
        SUB.B   #1,D4          decrementa el numero de recorrrimientos
        CMP.B   #0,D4          checa si ya se realizaron todos
        BLT     ROX            si no ejecuta otro corrimiento
        ADD.B   $100A,D5       suma el exponente del recorrimiento con los
        ADD.B   D5,$1004       exponentes de los multiplicandos
        TST.B     D6           checa el signo
        BEQ       FUE          si es positivo brinca a FUE
        NEG.L     D1           complementa a dos el producto
FUE     MOVE.L  D1,$1000       regresa el resultado
        MOVEM.L (A7)+,D1-D6    recupera registro de apoyo
        RTS                    regresa de la subrutina de multiplicación
```

Subrutina para realilzar la división de dos numeros de 32 bits (DIVU 32), enla dirección $1000 se debe de colocar el divisor y su exponente en la dirección $1004, mientras que en la dirección $1006 se debe de colocar el dividendo y en la direcció$100A y el resultado se obtendra en la dirección del divisor junto con su exponente.

```
    ENSAMBLADOR              COMENTARIOS
MOVEM.L D0-D5,-(A7)    salva registros de apoyo
MOVE.L  $1000,D0       carga divisor en D0
MOVE.L  $1006,D1       carga dividendo en D1
 CLR.B    D5          limpia D5
 TST.L    D1          dividendo negativo
```

```
      BPL       CHD2          brinca a CHD2
      NEG.L     D1            convierteLo a positivo
      NOT.B     D5            cambia contador de signo
CHD2  TST.L     D0            divisor negativo
      BPL       DIV           brinca a DIV
      NEG.L     D0            convierteLo a positivo
      NOT.B     D5            cambia contador de signo
DIV   MOVE.B    #2,D4          carga contador en D4
      ADD.B   #4,$1004         actualiza  exponente del divisor
ASL1  ASL.L     #8,D0          recorre el divisor a 16 bits
      SUB.B     #1,D4          decrementa el contador de corrrimiento
      CMP.B     #0,D4          compara el contador de corrimiento
      BLT       ASL1           brinca a recorrer otra vez el divisor
      SUB.B $1004,$100A        resta exponentes
      MOVE.B $100A,$1004       reordena los exponentes
      CLR.W     D3             limpia la parte baja de D3
      DIVU      D0,D1          efectua división
      BVC       FOR            brinca a formatear el resultado
      MOVE.W    D1,D2          copia residuo en D2
      CLR.W     D1             limpia parte baja de D1
      SWAP      D1             invierte D1
      DIVU      D0,D1          ejecuta división
      MOVE.W    D1,D3          salva resultado de división en D3
      MOVE.W    D2,D1          salva el residuo en D1
      DIVU      D0,D1          ejecuta la división
FOR   MOVE.L    D1,D0          salva el resultado de la división
      SWAP      D1             invierte D1
      MOVE.W    D3,D1          salva el cociente en D1
      SWAP      D1             invierte D1
      TST.B     D5           checa el signo
       BEQ      FUE           si es positivo brinca a FUE
       NEG.L    D1            complementa a dos el producto
FUE   MOVE.L  D1,$1000          regresa resultado
      MOVEM.L (A7)+,D0-D5       recupera registros de apoyo
      RTS                       regresa de subrutina
```

Subrutina para obtener una potencia N de un numero de 32 bits "POT 32" signado, para utilizar esta subrutina se carga el numero en la localidad de memoria $1000 y en la dirección $1004 se coloca el exponente mientras que en la dirección $100C el numero de la potencia que se desea obtener, y el resultado se obtiene en la mismas direcciones donde se cargaron los numeros.

```
      ENSAMBLADOR                COMENTARIOS
     MOVE.L  $1000,$1006  copia el numero en $1006
     MOVE.B  $1004,$100A  copia el exponente en $100A
COM  SUB.B    #1,$100C     actualiza el contador de potencias
     CMP.B    #0,$100C verifica si ya se realizaron todas las potencias
```

```
        BEQ     FIN             si SI termina la subrutina
        JSR     MULU 32         si NO brinca a subrutina de multiplicación
        BRA     COM             brinca a comparar de nuevo
FIN     RTS                     regresa de subrutina
```

Subrutina para obtener la función seno de un numero de 32 bits "SEN 32" signado y con exponente por medio de una aproximación, evaluando los primeros 5 terminos de un desarrollo de Taylor de de la función seno. El numero del cual se desea obtener el resultado de la función seno se debera cargar en al dirección $1000 y su exponente en la dirección $1004. Obteniendose el resultado en las mismas direcciones.

```
      ENSAMBLADOR                COMENTARIOS
     MOVE.L   $1000,$1006  copia el numero en $1006
     MOVE.B   $1004,$100A  copia el exponente en $100A
     MOVE.L   $1000,$100C  guarda el primer sumando
     MOVE.B   $1004,$1010  guarda el exponente del sumando
     JSR        MULU 32    eleva al cuadrado el numero
     MOVE.L   $1000,$1006  guarda el cuadrado del numero
     MOVE.B   $1004,$100A  guarda su exponente
     MOVE.L   $100C,$1000  recupera el numero
     MOVE.B   $1010,$1004  recupera su exponente
     JSR        MULU 32    eleva al cubo el numero
     MOVE.L   $1000,$1012  guarda el segundo sumando
     MOVE.B   $1004,$1016  guarda el exponente del numero
     JSR        MULU 32    eleva a la quinta potencia el numero
     MOVE.L   $1000,$1018  guarda el tercer sumando
     MOVE.B   $1004,$101C  guarda su exponente
     JSR        MULU 32    eleva a la septima potencia el numero
     MOVE.L   $1000,$101E  guarda el cuarto sumando
     MOVE.B   $1004,$1022  guarda su exponente
     JSR        MULU 32    eleva a la novena potencia el numero
     MOVE.L #$58980,$1006  carga el valor de 9!
     MOVE.B     #00,$100A  carga el exponente de 9!
     JSR         DIVU 32   divide el quinto sumando entre 9!
     MOVE.L   $1000,$1024  actualiza el quinto sumando
     MOVE.B   $1004,$1028  actualiza su exponente
     MOVE.L   $101E,$1000  recupera el cuarto sumando
     MOVE.B   $1022,$1004  recupera su exponente
     MOVE.L #$1B80,$1006   carga el valor de 7!
     MOVE.B     #00,$100A  carga el exponente de 7!
     JSR         DIVU 32   divide el cuarto sumando entre 7!
     MOVE.L   $1000,$101E  actualiza el cuarto sumando
     MOVE.B   $1004,$1022  actualiza su exponente
     MOVE.L   $1018,$1000  recupera el tercer sumando
     MOVE.B   $101C,$1004  recupera su exponente
     MOVE.L    #$78,$1006  carga el valor de 5!
     MOVE.B      #0,$100A  carga el exponente de 5!
     JSR         DIVU 32   divide el tercer sumando entre 5!
     MOVE.L   $1000,$1018  actualiza el tercer sumando
```

```
MOVE.B  $1004,$101C   actualiza su exponente
MOVE.L  $1012,$1000   recupera el segundo sumando
MOVE.B  $1016,$1004   recupera su exponente
MOVE.L    #$6,$1006   carga el valor de 3!
MOVE.B     #0,$100A   carga el exponente de 3!
JSR        DIVU 32    divide el segundo sumando entre 3!
NEG.L     $1000       cambiale el signo al segundo sumando
MOVE.L  $100C,$1006   recupera primer sumando
MOVE.B  $1010,$100A   recupera su exponente
JSR        SUMA 32    realiza la primer suma
MOVE.L  $1018,$1006   recupera el tercer sumando
MOVE.L  $101C,$100A   recupera su exponente
JSR        SUMA 32    realiza la segunda suma
MOVE.L  $101E,$1006   recupera el cuarto sumando
MOVE.B  $1022,$100A   recupera su exponente
NEG.L      $101E      cambiale el signo al cuarto sumando
JSR        SUMA 32    realiza la tercer suma
MOVE.L  $1024,$1006   recupera el quinto sumando
MOVE.B  $1028,$100A   recupera su exponente
JSR        SUMA 32    realiza la cuarta suma
RTS                   regresa de subrutina.
```

Subrutina para obtener la función coseno de un numero de 32 bits "COS 32" signado y con exponente por medio de una aproximación, evaluando los primeros 5 terminos de un desarrollo de Taylor de de la función coseno. El numero del cual se desea obtener el resultado de la función seno se debera cargar en al dirección $1000 y su exponente en la dirección $1004. Obteniendose el resultado en las mismas direcciones.

```
 ENSAMBLADOR                COMENTARIOS
MOVE.L  $1000,$1006   copia el numero en $1006
MOVE.B  $1004,$100A   copia el exponente en $100A
JSR        MULU 32    eleva al cuadrado el numero
MOVE.L  $1000,$1012   guarda el segundo sumando
MOVE.B  $1004,$1016   guarda su exponente
MOVE.L  $1012,$1006   recupera el numero
MOVE.B  $1016,$100A   recupera su exponente
JSR        MULU 32    eleva a la cuarta potencia el numero
MOVE.L  $1000,$1018   guarda el tercer sumando
MOVE.B  $1004,$101C   guarda su exponente
JSR        MULU 32    eleva a la sexta potencia el numero
MOVE.L  $1000,$101E   guarda el cuarto sumando
MOVE.B  $1004,$1022   guarda su exponente
JSR        MULU 32    eleva a la octava potencia el numero
MOVE.L  $1000,$1024   guarda el quinto sumando
MOVE.B  $1004,$1028   guarda su exponente
MOVE.L  #$D900,$1006  carga el valor de 8!
MOVE.B     #00,$100A  carga el exponente de 8!
JSR        DIVU 32    divide el quinto sumando entre 8!
MOVE.L  $1000,$102A   actualiza el quinto sumando
```

```
MOVE.B  $1004,$1028  actualiza su exponente
MOVE.L  $101E,$1000  recupera el cuarto sumando
MOVE.B  $1022,$1004  recupera su exponente
MOVE.L   #$2D0,$1006 carga el valor de 6!
MOVE.B     #00,$100A carga el exponente de 6!
JSR       DIVU 32    divide el cuarto sumando entre 6!
MOVE.L  $1000,$101E  actualiza el cuarto sumando
MOVE.B  $1004,$1022  actualiza su exponente
MOVE.L  $1018,$1000  recupera el tercer sumando
MOVE.B  $101C,$1004  recupera su exponente
MOVE.L   #$18,$1006  carga el valor de 4!
MOVE.B     #0,$100A  carga el exponente de 4!
JSR       DIVU 32    divide el tercer sumando entre 4!
MOVE.L  $1000,$1018  actualiza el tercer sumando
MOVE.B  $1004,$101C  actualiza su exponente
MOVE.L  $1012,$1000  recupera el segundo sumando
MOVE.B  $1016,$1004  recupera su exponente
MOVE.L    #$2,$1006  carga el valor de 2!
MOVE.B     #0,$100A  carga el exponente de 2!
JSR       DIVU 32    divide el segundo sumando entre 2!
NEG.L     $1000      cambiale el signo al segundo sumando
MOVE.L  $100C,$1006  recupera primer sumando
MOVE.B  $1010,$100A  recupera su exponente
JSR       SUMA 32    realiza la primer suma
MOVE.L  $1018,$1006  recupera el tercer sumando
MOVE.L  $101C,$100A  recupera su exponente
JSR       SUMA 32    realiza la segunda suma
MOVE.L  $101E,$1006  recupera el cuarto sumando
MOVE.B  $1022,$100A  recupera su exponente
NEG.L     $101E      cambiale el signo al cuarto sumando
JSR       SUMA 32    realiza la tercer suma
MOVE.L  $1024,$1006  recupera el quinto sumando
MOVE.B  $1028,$100A  recupera su exponente
JSR       SUMA 32    realiza la cuarta suma
RTS                  regresa de subrutina.
```

Subrutina para la raiz cuadrada de un nunero de 32 bits, el numero se debe colocar el al direcciòn $1000 y su exponente en la direcciòn $1004, y el resultado se obtendra en las mismas localidades.

```
  ENSAMBLADOR              COMENTARIOS
MOVEM.L D0-D3,-(A7)    salva registros de apoyo
MOVE.L  $1000,D0       carga numero en D0


ASR.L   #8,D0          prepara numero para la subrutina
ADD.B   #2,$1004       actualiza el exponente del numero
```

```
     MOVE.L  D0,D2          copia el numero en D2
     DIVU    #200,D2        divide entre 200 para la primera aproximación
     ADD.B    #2,D2         sumale 2 a la aproximación
REP  MOVE.L  D0,D1          recupera el valor del numero
     DIVU    D2,D1          vuelve a aproximar el valor
     MOVE.L  D1,D3          guarda la aproximación en D3
     SUB.W   D2,D3          prueba si son iguales las aproximaciones
     BEQ      FIN           si SI termina la subrutina
     CMPI    #1,D3          compara si difieren en 1
     BEQ     FIN            si SI termina la subrutina
     CMPI    #-1,D3         compara si es negativa la diferencia
     BEQ     FIN            si SI termina la subrutina
     ADD.L    D1,D2         suma las dos ultimas aproximaciones
     LSR.W    #1,D2         recorre la suma una unidad
     BRA      REP           vuelve a repetir la aproximación
FIN  MOVE.B  $1004,D2       recuepera valor del exponente
     AND.B     #1,D2        prueba si es impar el exponente
     BEQ      REO           brinca a reordenar el resultado
     CLR.B    D4            limpia D4
     TST.B    D2            exponente negativo
     BPL      DIV           brinca a DIV
     NEG.B    D2            conviertelo a positivo
     NOT.B    D4            cambia contador de signo
DIV  MOVE.L  #0,D3          limpia D3
     MOVE.B  D2,D3          carga el exponente en D3
     DIVU    #2,D3          divide el exponente a al mitad
     TST.B    D4            checa el signo
     BEQ      TER           si es positivo brinca a TER
     NEG.B     D3           complementa la división
TER  MOVE.B  #2,D0          carga contador en D0
     SUB.B   #4,D3          actualiza el exponente
REC  ASL.L   #8,D1          recorre el resultado
     SUB.B   #1,D0          restale 1 al contador
     CPM.B   #0,D0          compara el contador con 0
     BLT      REC           vuelve a ejecutar el recorrimiento
     MOVE.L  D1,$1000       regresa el resultado
     MOVE.B  D3,$1004       regresa su exponente
     MOVEM.L (A7)+,D0-D4    recupera registros de apoyo
     RTS                    regresa de la subrutina
REO  CLR.B    D4            limpia D4
     TST.B    D2            exponente negativo
     BPL      DIB           brinca a DIV
     NEG.B    D2            conviertelo a positivo
     NOT.B    D4            cambia contador de signo
DIB  MOVE.L  #0,D3          limpia D3
     ADD.B   #1,D2          convierte a par el exponente
     MOVE.B  D2,D3          carga el exponente en D3
     DIVU    #2,D3          divide el exponente a al mitad
     TST.B    D4            checa el signo
     BEQ      TER           si es positivo brinca a TRM
     NEG.B     D3           complementa la división
TRM  SUB.B   #2,D3          actualiza el exponente
     ASL.L   #8,D1          recorre el resultado
```

```
MOVE.L  D1,$1000        regresa el resultado
MOVE.B  D3,$1004        regresa su exponente
MOVEM.L (A7)+,D0-D4     recupera registros de apoyo
RTS                     regresa de la subrutina
```

PROGRAMA PARA SIMULAR EL SISTEMA ESCRITO EN LENGUAJE PASCAL

```
program fft (intput,output);
   const n=512; {define la matriz de datos a procesar nXn}
         n0=9;{n0 se define como n0=ln n/ln 2)
         pi=3.141592654;
   type lineas=array[1..n] of real;
        arch=file of lineas;
        cadena=string[12];
   var x,y,b,x1,y1,z1:lineas;
       indice,n1,n2,k,k1,k2,i,m,o,q,r,l,p,j,j1,j2,h,a,d,e,f,g:integer;
       a1,c,s,t1,t2,temp:real;
       res:char;
       arch1,archr,archr1,arch2,arch1,archi1:arch;
   procedure analiza; {subrutina utilizada por la transformada de}
       begin {analiza)(FOURIER de un renglon)
             j1:=0;
             j1:=j;
             p:=0;
             for h:=1 to n0 do
                 begin {for h)
                       j2:=j1 div 2;
                       p:=2*p+j1-2*j2;
                       j1:=j2;
                 end; {for h)
        end; {analiza)
   procedure fft (var x,y:lineas);{subrutina que obtiene la transformada
       begin (fft){de FOURIER de un renglon}
             n2:=n div 2;
             n1:=n0-1;
             k:=0;
             for l:=1 to n0 do
                 begin {for l)
                       while k<n do
                             begin (while)
                                   for i:=1 to n2 do
                                       begin (for i)
                                             j:=trunc(k/exp(n1*ln(2)));
                                             analiza;
                                             a1:=2*pi*p/n;
                                             c:=cos(a1);
                                             s:=sin(a1);
                                             k1:=k+1;
                                             k2:=k1+n2;
```

```
                                        t1:=x[k2]*c+y[k2]*s;
                                        t2:=y[k2]*c-x[k2]*s;
                                        x[k2]:=x[k1]-t1;
                                        y[k2]:=y[k1]-t2;
                                        x[k1]:=x[k1]+t1;
                                        y[k1]:=y[k1]+t2;
                                        k:=k+1;
                                    end; (for i)
                                k:=k+n2;
                            end;(while)
                        k:=0;
                        n1:=n1-1;
                        n2:=n2 div 2;
                end; (for i)
            for k:=1 to n do
                begin (for k)
                        j:=k-1;
                        analiza;
                        i:=p+1;
                        if i>k then
                           begin (if)
                                t1:=x[k];
                                t2:=y[k];
                                x[k]:=x[i];
                                y[k]:=y[i];
                                x[i]:=t1;
                                y[i]:=t2;
                           end; (if)
                end; (for k)
    end; (imagen)
procedure leetc;(subrutina para leer del teclado una matriz y)
    var  nom1:string[8];(asignarsela a un archivo)
         nom2:string[12];
    begin (leetc)
          writeln;
          write('COMO SE VA A LLAMAR TU ARCHIVO ? ');
          readln(nom1);
          nom2:=nom1+'.dat ';
          assign(arch1,nom2);
          rewrite(arch1);
          for a:=1 to n do
              begin (for a)
                    writeln('renglon ',a);
                    for d:=1 to n do
                        begin (for d)
                              write('DATO (',1:3,d:4,')= ');
                              readln(b[d]);
                        end; (for d)
        write('QUIERES CORREGIR ALGUN DATO DEL RENGLON ',a,' s/n ? ');
                    readln(res);
                    while res='s' do
                         begin (while res)
                               write('QUE NUMERO DE DATO ? ');
```

```
                                    readln(k);
                                    write('DATO (',a:3,k:4,')= ');
                                    readln(b[k]);
                                  write('QUIERES CORREGIR OTRO s/n ? ');
                                    readln(res);
                               end; {while res}
                         writeln;
                         write(arch1,b);
                    end; {for a}
               close(arch1);
               writeln('DATOS GUARDADOS EN EL ARCHIVO ',NOM2);
     end; {leetc}
 procedure leearch;{subrutina para leer de un archivo una matriz}
     var  nom1:string[8];
          nom2:string[12];
     begin {leearch}
          writeln;
          write('COMO SE VA A LLAMAR TU ARCHIVO ? ');
          readln(nom1);
          nom2:=nom1+'.dat ';
          assign(arch1,nom2);
     end;
procedure invierte(archinv:cadena;{subrutina para invertir una matriz}
                   name:cadena;{leida de un archivo dado y almace-}
                 var archn:arch);{narla en otro archivo}
     var  e,f:integer;
          arch2:arch;
          linea1,linea2:lineas;
     begin {invierte}
          assign(archn,name);
          assign(arch2,archinv);
          rewrite(arch2);
          for e:=1 to n do
              begin {for e}
                    reset(archn);
                    for f:=1 to n do
                        begin {for f}
                              read(archn,linea1);
                              linea2[f]:=linea1[e];
                        end; {end for f}
                    write(arch2,linea2);
              end; {for e}
          close(archn);
          close(arch2);
   end; {invierte}
 procedure voltea;
     begin {voltea}
     reset(arch1);
          assign(archr,'archr1.dat ');
          rewrite(archr);
          for indice:=1 to n do
              begin {for indice}
                    read(arch1,x);
```

```
                    for i:=1 to n div 2 do
                        begin (for i)
                              temp:=x[i];
                              x[i]:=x[n div 2 +i];
                              x[n div 2 +i]:=temp;
                        end; (for i)
                    write(archr,x);
                end; (for indice)
            close(arch1);
            close(archr);
    end; (voltea)
procedure reflex;(subrutina para reflejar una matriz sobre sus dos)
    begin (reflex)(ejes, leida de un archivo y almacenarla en otro)
            voltea;
            invierte('archinv.dat','archreal.dat',arch1);
            assign(arch1,'archinv.dat ');
            voltea;
            assign(arch2,'archr1.dat ');
            reset(arch2);
            for indice:=1 to n do
                begin (for indice)
                      read(arch2,x);
                      for i:=1 to n do
                          begin (for i)
                                write(x[i]:12:4)
                          end; (for i)
                      writeln;
                end; (for indice)
          close(arch2);
    end; (reflex)
begin (program)(empieza el programa)
      write('VAS A LEER DEL TECLADO ');
      read(res)
      if res='S'then leetc(ejecuta la subrutina de leer teclado)
      else
      leearch;(ejecuta la subrutina de leer un archivo)
      reset(arch1);
      assign(archr,'archr1.dat ');
      assign(arch1,'arch11.dat ');
      rewrite(archr);
      rewrite(arch1);
      for indice:=1 to n do
          begin (for)
                read(arch1,x);
                for g:=1 to n do(despliega en pantalla la matriz)
                    begin (for g) (de datos a procesar)
                          write(x[g]:8:0);
                          y[g]:=0;(condicion inicial matriz de datos)
                    end; (for g) (parte imaginaria cero)
                writeln;
                fft(x,y);(ejecuta la subrutina de fft sobre los
                (renglones de la matriz de datos)
                write(archr,x);(guarda la parte real)
```

```
                write(archi,y);(guarda la parte imaginaria)
            end; (for)
        close(arch1);
        close(archr);
        close(archi);
        writeln;
        writeln;
        invierte('archinv1.dat','archr1.dat',arch1);
        (ejecuta la subrutina de transposicion de matrices sobre
       la matriz de valores reales, transponiedo columnas por renglones)
        invierte('archinv2.dat','archi1.dat',arch1);
        (ejecuta la subrutina de transposicion de matrices sobre
        la matriz de valores imaginarios, transponiedo columnas
        por renglones )
        assign(arch1,'archinv1.dat ');
        assign(arch2,'archinv2.dat ');
        assign(archr,'archreal.dat ');
        reset(arch1);
        reset(arch2);
        rewrite(archr);
        for m:=1 to n do
            begin (for m)
                read(arch1,x);
                read(arch2,y);
                fft(x,y);(ejecuta la subrutina de fft sobre las
                columnas de las matrices real e imaginaria )
                for j:=1 to n do
                   begin (for j)
                        x1[j]:=sqr(x[j]);
                        y1[j]:=sqr(y[j]);
                        z1[j]:=sqrt(x1[j]+y1[j]);
                   end; (for j)
                write(archr,z1);
            end; (for m)
       close(arch1);
       close(arch2);
       close(archr);
       assign(arch1,'archreal.dat ');
       reflex;(se ejecuta la subrutina de reflexion sobre los dos ejes)
       (de la matriz modulo y se despliega en pantalla)
end. (programa)
```

B I B L I O G R A F I A

1.- R. E. TWOGOOD & R. J. SHERWOOD, AN OVERVIEW OF DIGITAL IMAGE PROCESSING, LAWRECE LIVERMORE NATIONAL LABORATORY, LIERMORE CALIFORNIA.

2.- H. C. ANDREWS & D. R. HUNT, DIGITAL IMAGE RESTORATION NEW JERSY, PRETICE HALL.

3.- CASTLEMAN KENNETH R., DIGITAL IMGE PROCESSING, ENGLEWOOD CLIFS N. J., PRENTICE HALL.

4.- DISEÑO DE SISTEMAS DIGITALES CON MICROPROCESADORES, E. MANDADO y E. TASSIS, PUBLICACIONES MARCOMBO.

5.- INTERCONEXION DE PERIFERICOS A MICROPROCESADORES, SERIE MUNDO ELECTROICO, PUBLICACIONES MARCOMBO.

6.- 16-BIT MICROPROCESSOR MC68000, USER'S MANUAL, MOTOROLA.

7.- PROCESADORES PROGLAMABLES EL MICROPROCESADOR, E. MANDADO, PUBLICACIONES MARCOMBO.

8.- CIRCUITOS DIGITALES Y MICROPROCESADORES, HEBERT TAUB, MC GRAW HILL.

9.- MICROCOMPUTER-BASED DESIGN, JOHN B. PEATMAN, INTERNATIONAL STUDENT EDITION, MC GRAW HILL.

10.-THE 68000 PRINCIPLES AND PROGRAMMING, LEO J. SCALON, HOWARD W. SAMS & CO. INC.

11.-68000 MICROPROCESSORS HANDBOOK, G. KANE, OSBORNE/MC GRAW HILL.

12.-68000 ASSEMBLY LENGUAGE PROGRAMMING, G. KANE, D. HAWKINS and L. A. LEVENTHAL, OSBORNE/MC GRAW HILL.

13.-M68000 16/32 BIT MICROPROCESSOR PROGRAMMERS REFERENCE MANUAL (FOURT EDITION) MOTOROLA SEMICONDUCTORS.

14.-THE FAST FOURIER TRANSFORM, E. ORAN BRIGHAM,PRENTICE ALL.

R E F E R E N C I A S

1.- ADVANCE INFORMATION MC68000L4, MC68000L6 MC68000L8, MOTOROLA SEMICONDUCTORS.

2.- INTERFACING M6800 PERIPHERIAL DEVICES TO THE MC68000, APPLIATION NOTE 808, MOTOROLA SEMICONDUCTORS.

3.- DUAL 16-BIT PORTS FOR THE MC68000 USING TWO MC6821, APPLIATION NOTE 810, MOTOROLA SEMICONDUCTORS.

4.- PRIORITIZED INDIVIDUALLY VECTORED INTERRUPS FOR MULTIPLE PERIPHERIAL SYSTEMS WITH THE 68000, APPLIATION NOTE 819, MOTOROLA SEMICONDUCTORS.

5.- A DISCUSSION OF INTERRUPTS FOR THE MC68000, ENGINEERIG BULLETIN 97 MOTOROLA SEMICONDUCTORS.

6.- FISICOQUIMICA, GILBERT W. CASTELLAN, FONDO EDUCATIVO INTERAMERICANO.

7.- TESIS DOCTORAL, VICTOR MANUEL CASTAÑO MENESES.

8.- MICRODIFRACTION AND LATTICE RESOLUTION STUDIES OF FIVEFOLD SYMMETRY GOLD PARTICLES, P. SCHABES RETCHKIMAN, A. GOMEZ, G. VAZQUEZ POLO & M. JOSE YACAMAN, J. VAC. SCI. TECHOL. A2(1), JAN-MAR 1984.