ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

DEPARTAMENTO DE ELECTRONICA Y CONTROL

TESIS DE GRADO PREVIA A LA OBTENCION

DEL TITULO DE INGENIERO EN

ELECTRONICA Y CONTROL

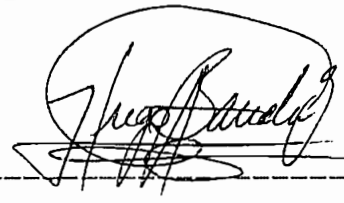
DE LA ESCUELA POLITECNICA NACIONAL

"APLICACION DEL PROTOCOLO MIDI A LA SINTESIS DIGITAL DEL SONIDO"

ADRIAN NARANJO PUENTE

QUITO, MARZO DE 1993

Certifico que la presente Tesis
ha sido completamente realizada
por el señor Adrian Naranjo Puente.



Dr. Hugo Banda Gamboa

DIRECTOR DE TESIS

DEDICATORIA

A mis Padres y hermano por sus esfuerzos y sacrificios.

A Ana Cristina por su apoyo y compañia.

AGRADECIMIENTO

- Al Dr. Hugo Banda G. por su muy acertada dirección.
- Al Ing. Gonzalo Jurado por su valiosa colaboración.
- A todos mis maestros de la Escuela Politécnica Nacional.

INDICE

PAGINA

PAGINA

PAGINA

INDICE DE FIGURAS

PREFACIO

Desde los albores de la humanidad el hombre ha tratado de crear sonidos que en un inicio fueron usados para lograr comunicarse con otros individuos, al golpear dos piedras entre si, descubrió el fascinante mundo de la creación del sonido. Desde este instante se empezó ha crear cada vez más medios de generación de sonidos, especialmente debido a su necesidad de comunicación y al inherente ritmo que marca la vida misma del ser humano.

Durante siglos los compositores han intentado romper las ataduras impuestas a su arte. Un estudio de partituras musicales demuestra como una y otra vez los músicos han intentado apartarse de los convencionalismos de su época en busca de nuevos modelos de expresión. El uso de estructuras armónicas nuevas y la invención de nuevos instrumentos fueron el resultado de esta búsqueda. Por lo tanto no es de sorprenderse el hecho de que el descubrimiento de las leyes de la electricidad y la tecnología misma abriera puertas a la generación de sonidos de una manera diferente a la acústica.

Es por eso que nuestra sociedad moderna, lejos de olvidar la música, ofrece sofisticados medios de composición, registro y difusión de ella, gracias a la electrónica.

En particular, existen hoy en día instrumentos electrónicos de generación de sonido

extremadamente interesantes. La difusión de estos instrumentos no se hace solo entre profesionales, sino también, en una medida cada vez mayor, entre el gran público.

Es así, que en la actualidad las computadoras, irónicamente, están disolviendo los límites entre las artes y las ciencias.

Hoy en día que se tiene disponibilidad de microcomputadoras y poderosos lenguajes de programación, existe la posibilidad de crear sistemas, que nos permitan crear funciones que solo hace una década eran imposibles.

Este es el objetivo fundamental del presente trabajo, demostrar que la tecnología con la que contamos hoy día no está desligada de las actividades creativas del ser humano.

Aún el propio Albert Einstein, quien solía interpretar el violín, decía que muchas de sus revelaciones científicas tenían mas de inspiración emocional o intuición artística que el resultado de largos cálculos matemáticos.

I N T R O D U C C I O N

INTRODUCCION

El campo de aplicación de la electrónica se ha extendido prácticamente a todo actividad humana, hoy en día se puede decir que no existe campo del conocimiento donde no estén presentes de una u otra manera equipos o elementos electrónicos que permiten realizar cualquier tarea de una manera mas rápida, eficiente y segura.

En nuestro país esta situación no es ajena, a pesar del papel pasivo que tenemos frente al desarrollo de los países industrializados, se observa que la influencia de la electrónica y la informática en nuestro diario convivir es creciente.

Es en este marco que se aborda la presente tesis, estudiando un tópico que no ha sido tratado con anterioridad. La síntesis del sonido es un campo en el cual la aplicación de la electrónica ha permitido obtener sistemas altamente sofisticados que tienen un rango de aplicación muy amplio, el cual va desde su uso para aplicaciones musicales hasta sistemas de reconocimiento de voz para automatizar aún más los sistemas existentes.

Esta verdadera revolución en la creación del sonido obligó a los diferentes fabricantes de equipo de síntesis de sonido, a establecer un sistema mediante el

cual, se permita la conexión de equipos de diferentes marcas. Es así que en Julio de 1984, 16 fabricantes de todo el mundo adoptaron un nuevo estándar de comunicación que se lo denominó MIDI, que es el acrónimo de las siglas inglesas de MUSICAL INSTRUMENTS DIGITAL INTERFACE.

El propósito fundamental de la presente Tesis es establecer las bases que nos permitan comprender de una manera clara los diferentes principios que gobiernan la generación artificial del sonido, así como el realizar una detallada descripción del protocolo MIDI y visualizar sus posibilidades a través de la implementación de programas escritos en un lenguaje apropiado para esta clase de aplicaciones.

Desde los inicios de la relación electrónica sonido, la investigación fue orientada especialmente a la aplicación musical y así a la creación de equipos que permitan sustituir los aparatos de ejecución tradicionales tales como de percusión (piano), cuerdas vibrantes (violines), viento (flautas), etc. Es por esta razón que si bien, la sintetización del sonido abarca la generación de cualquier sensación acústica que perciban nuestros oídos, en la presente tesis abordaremos principalmente la generación de sonidos aplicados a la música.

Para lograr los fines anteriormente anotados, se presenta en el Capitulo I una descripción

histórica de la evolución que ha tenido la síntesis del sonido, partiendo de las etapas iniciales de experimentación en el siglo pasado, se estudia los primeros aparatos desarrollados a inicios de este siglo, revisaremos los pianos eléctricos y órganos electrónicos para llegar a la era de los sintetizadores analógicos de sonido, finalmente entraremos a la era de la síntesis digital del sonido para llegar a analizar uno de los más recientes desarrollos tecnológicos, la creación de los sistemas MIDI y el aparecimiento de lo que se ha denominado, música asistida por computador.

Posteriormente en el Capítulo II, se presenta una descripción de los diferentes métodos que se han desarrollado para sintetizar el sonido. Para abordar este tema se hace necesario realizar previamente un estudio de las diferentes cualidades que caracterizan al sonido, por lo que se presenta un breve estudio de las mismas. Se estudia los métodos de síntesis tales como la síntesis por adición y por eliminación. Se revisa además las funciones de algunos de los aparatos que se han desarrollado en los últimos años para la síntesis digital de sonido, elementos tales como: sintetizadores con y sin teclado, muestreadores, secuenciadores digitales, procesadores MIDI, etc.

En el Capitulo III se presenta una descripción detallada del protocolo MIDI, definiéndolo

desde el punto de vista musical e informático. Se presentan las características de este sistema así como los circuitos MIDI y el hardware asociado con este tipo de sistemas. Para obtener una clara visión de las posibilidades de un sistema MIDI se estudian los mensajes MIDI correspondientes al estándar MIDI-0.1

En el Capítulo IV se implementan programas que permiten visualizar varias de las posibilidades al trabajar con un sistema MIDI. Cabe mencionar que para el desarrollo de los programas se ha usado el siguiente equipo: un computador IBM compatible equipado con la tarjeta V-4000 la que está constituida por
un Interfaz MIDI que es 100% compatible con la Unidad de Procesamiento MIDI MPU - 401 de Roland, sintetizador digital de sonido HT - 700 de CASIO equipado con terminales de comunicación MIDI.

Estos programas se implementan en lenguaje "C", y permiten la comunicación entre el computador y el sintetizador de sonido de tal manera que se pueda analizar los mensajes MIDI que se transmiten entre el sintetizador y el computador, además de la implementación de un secuenciador de un canal (grabación / ejecución) basado en software. Estos programas requieren la implementación de subrutinas escritas en lenguaje Assembly para el microprocesador del computador, subrutinas que sirven para la entrada y salida de los datos MIDI.

Finalmente, en el Capítulo V se realizan pruebas al programa desarrollado y se plantean algunos posibles cambios al mismo para realizar un secuenciador multicanal, se dan además los lineamientos para desarrollar un programa que permita sintetizar los sonidos de un sintetizador a partir del computador. Se presentan también conclusiones y recomendaciones al presente trabajo.

C A P I T U L O I

EVOLUCION HISTORICA DE LA SINTESIS ELECTRICA DEL SONIDO

EVOLUCION HISTORICA DE LA SINTESIS ELECTRICA DEL SONIDO

La tecnología para la creación del sonido por medios sintéticos, ha tenido unas historia larga e intrigante. Los primeros intentos rudimentarios para la generación eléctrica del sonido se llevaron a cabo aún antes de la invención del bombillo eléctrico. Los principios de la electricidad no pudieron entenderse hasta antes de finales del siglo pasado e inicialmente los fenómenos acústicos se usaron para demostrar la naturaleza oscilatoria de las ondas eléctricas.

El desarrollo de sistemas para la generación del sonido puede dividirse en cuatro épocas. La época anterior al aparecimiento de la grabación en cinta en 1948 está designada como la etapa inicial. La segunda etapa, la era del estudio de sonido se inicia a finales de la década de los 40, cuando se hizo realidad el sueño de almacenar los sonidos y manipularlos electrónicamente.

La era del sintetizador analógico de sonido empezó con la invención del sintetizador controlado por voltaje por Robert Moog en 1964. Moog y sus contemporáneos revolucionaron el diseño de equipo de síntesis de sonido al usar tecnología integrada para crear el primer sintetizador de sonido.

La tecnológica etapa más reciente incluye la

síntesis digital del sonido, cuando la aplicación de sistemas computarizados y sofisticados medios de procesamiento de información musical digitalizada, han permitido crear lo que se ha denominado música asistida por computador.(1)

1.1 ETAPA INICIAL

Muchos de los primeros aparatos para producir sonido electrónicamente fueron el resultado de accidentes experimentales. El primer instrumento musical eléctrico fue inventado en 1874 por el americano Elisha Grey, quien descubrió que al conectar algunos curiosos circuitos eléctricos con una lámina vibrante se producía un tono audible, basándose en este descubrimiento desarrolló un pequeño aparato con un teclado al que lo llamó telégrafo musical.

A finales del siglo pasado el físico inglés William Duddell realizó trabajos para eliminar el molesto ruido que se producía en las lámparas de arco de carbón, descubriendo que podía controlar estos irregulares tonos al conectar un circuito secundario al arco, esto permitía modular y controlar las oscilaciones del arco. Duddell adaptó un pequeño teclado a este aparato creando lo que sería una versión primitiva de modulación de frecuencia usando un circuito de control. Más tarde, Thaddeus Cahill fue el primer hombre que tuvo una visión verdadera de lo

que sería la generación eléctrica del sonido. Cahill construyó el primer sintetizador de sonido, aparato que era capaz de generar y alterar las ondas eléctricas, siendo conocido con el nombre de telharmónio. Cahill fue además el que adoptó el término "sintetización", refiriéndose a la técnica de combinar tonos individuales para crear tonos compuestos.(1) La complejidad del telharmonio era sorprendente. El peso del aparato era de 200 toneladas y el mecanismo generador de tonos constaba de 12 ejes sobre los cuales estaban montados una serie de alternadores, que cuando eran rotados rápidamente al girar los ejes hacían contacto con el circuito eléctrico, la frecuencia de este contacto creaba una oscilación eléctrica de una frecuencia dada. Cada uno de los 12 ejes correspondía a una nota de la escala cromática, usando en total 408 alternadores. Los tonos generados eran combinados en un mezclador de tonos, donde los componentes individuales del sonido total podían ser controlados y balanceados, de esta manera se podía imitar sonidos orquestales. Estas señales eran entonces transmitidas a unos receptores telefónicos equipados con grandes bocinas para amplificar los sonidos.

Al iniciar los años 20, gracias al desarrollo del tubo de vacío y la radio - electrónica, se inventaron nuevos y curiosos aparatos. La nueva corriente musical de compositores tales como Bela Bartok o Edgard Varèse animó a desarrollar aparatos que a diferencia del telharmonio no requerían partes móviles para generar el

sonido sino éste se generaba por medios puramente electrónicos.

Uno de los inventos que marcaron esta época fue desarrollado por el ruso León Theremin, quien creó lo que se denominó etherófono. El aspecto más interesante y extraño de este instrumento es que se lo interpretaba sin contacto físico entre el intérprete y el aparato, simplemente se ejecutaba batiendo las manos en la vecindad de dos antenas, lo que añadía un aire místico a la interpretación, pero a la vez requería una gran preparación para ejecutarlo. El etherófono basaba su funcionamiento en un principio de la modulación llamado heterodino. Se mezclaban dos señales que eran casi iguales en frecuencia, esta combinación daba una tercera señal que era igual a la de las dos señales originales. Las oscilaciones de radiofrecuencia estaban por sobre el rango auditivo del oído humano, pero la diferencia entre las dos señales era audible. La frecuencia de uno de los radio osciladores era fija, mientras la frecuencia del otro oscilador podía ser variada al mover las manos en la vecindad de una antena vertical. Mientras la mano entraba en el campo electromagnético de la antena la frecuencia del oscilador variaba lo que resultaba en un cambio en la frecuencia de la tercera señal que podía ser escuchada por el oído humano. Además existía otra antena, colocada horizontalmente, con la que se podía controlar la intensidad del sonido. El sonido así generado era casi

puramente sinusoidal, pero tenía algunos armónicos que añadían profundidad a este sonido. Existía además la posibilidad de añadir ciertos efectos tales como vibrato con simples movimientos de la mano.

En 1926 el inventor alemán Jörg Mager desarrolló un aparato capaz de liberarse del rango tonal de la escala cromática, a éste lo denominó sparófono que a diferencia del piano clásico permitía una división microtonal. El principio de funcionamiento era bastante parecido al etherófono, pero su característica microtonal fue bastante futurista para su época por lo que pronto entró en desuso.

Otro aparato que se desarrolló en años posteriores fue el denominado ondas maternot, el cual fue inventado en 1928 por el francés Maurice Martenot. Este aparato monofónico, básicamente funcionaba bajo el mismo principio del theremin, siendo su característica fundamental el hecho de que la altura de las notas se controlaba por el movimiento lateral de una cinta asegurada a un anillo en un dedo de la mano, cuyo extremo estaba asegurado a un capacitor variable, lo que permitía variar la frecuencia en un rango de siete octavas.

A inicios de la década de los treinta fue atentado en Alemania un aparato llamado trautonomio, el mismo que se basaba en el mismo principio del ondas

martenot, pero usaba un oscilador de tubo de neón, lo que producía un sonido muy diferente a sus predecesores. Su sonido estaba compuesto por ondas con forma de diente de sierra que podían ser filtradas usando los controles del aparato.

En la década de los treinta la mayoría de aparatos musicales tendían a reemplazar al tradicional órgano de tubos, para tener así un aparato que ocupe menos espacio que el convencional órgano de iglesia. El primer órgano electrónico fue creado por el americano Laurens Hammond, quien usó tubos de vacío, amplificadores y parlantes, aunque el método para generar las ondas era electromecánico y bastante parecido al usado en el telharmonio. El órgano de Hammond replicaba muchas de las funciones del órgano de iglesia. Este aparato usaba doce osciladores de triodos de tubo de vacío, para generar las frecuencias altas, en tanto que requería una complicada red adicional para dividir estas frecuencias y generar así los sonidos en las frecuencias inferiores, desafortunadamente, sus características se adelantaron a las capacidades tecnológicas de la época y debido a su complejo circuito electrónico y a la naturaleza inestable de cientos de tubos de vacío, no era muy confiable y no permaneció en el mercado por mucho tiempo.

Los primeros pianos eléctricos se desarrollaron en la década de los treinta, pero a

diferencia de los órganos electrónicos de ese entonces, en los primeros pianos eléctricos el sonido era generado de manera tradicional, es decir con el golpe de pequeños martillos en diferentes cuerdas. Este sonido era entonces filtrado y amplificado, lo que variaba profundamente el sonido tradicional del piano.

El siguiente período no estuvo marcado con la creación de ningún instrumento en particular, sino por el aparecimiento de un medio para registrar el sonido, el cual expandió drásticamente las posibilidades técnicas en el procesamiento electrónico del mismo.

A partir de 1948 el campo del procesamiento electrónico de sonido empezó a tecnificarse en gran medida y popularizó los principios de la acústica que gobiernan la generación eléctrica del sonido.

Los primeros estudios de sonido fueron instalados a partir de 1951. Inicialmente los sonidos no eran electrónicamente generados, sino mas bien se usaba sonidos naturales los cuales eran mezclados y filtrados para generar sonidos de diferentes características.

En 1955 se creó el sintetizador denominado RCA Mark II, el mismo que era un sistema modular que podía: generar, modificar, procesar y grabar complejas sonoridades. El Mark II, de manera similar a las

computadoras de la época era un aparato muy grande que usaba electrónica de tubos de vacío, siendo éste el primero en incluir la idea del control por computadora para el procesamiento del sonido, de esta manera, se podía instruir a la máquina para que controle los parámetros básicos del mismo. Este control se lo hacía ingresando los datos en una cinta perforada de papel. Los osciladores del Mark II, permitían generar los doce tonos de la escala cromática y partiendo de ondas de diente de sierra, personal muy experto podía generar complejas sonoridades.

El sistema de introducción de información al sintetizador, se basaba en un código binario de treinta y seis columnas, que generaba sólo dos sonidos a la vez. Este código designaba parámetros tales como: timbre, frecuencia y duración de las notas. Otra característica importante de este sintetizador era que incluía una registradora y reproductora de cinta magnética que permitía crear una secuencia de sonidos y luego combinarla con otra secuencia en una cinta nueva.

1.2 ETAPA DE LA SINTESIS ANALOGICA

La gran revolución en el diseño de aparatos electrónicos para generar sonido, se inició a mediados de la década de los sesenta. En parte esta revolución fue causada por la llegada del transistor. Varios de los precedentes para el aparecimiento del moderno sintetizador

fueron establecidos por el Mark II de RCA. Muchas de las falencias asociadas con este sintetizador fueron superadas con el aparecimiento de elementos controlados por voltaje, este trabajo fue realizado en 1964 por Robert Moog y Donald Buchla en los Estados Unidos. Es en estos años cuando se empezó a asociar el término sintetizador con la generación electrónica del sonido.

Es importante definir qué es un sintetizador. Se puede definirlo como un sistema electrónico para generar, modificar y organizar el sonido en tiempo real. Cuando un sintetizador se combina con equipo para amplificar, mezclar y grabar, se obtiene un moderno estudio de sonido.

Las características generales del moderno sintetizador pueden resumirse de la siguiente manera:

- Un sintetizador tiene un amplio espectro para generar sonidos sean estos musicales o no. Puede ser programable y con preajustes de cualquier sonido o secuencia predeterminada.

- Los osciladores, amplificadores y filtros del sintetizador son controlados y activados por medio de señales de voltaje. Generalmente incluyen un teclado, por medio del cual se envía una señal de voltaje correspondiente a la tecla presionada, para que los componentes generadores de sonido sean activados y

produzcan el sonido. Debido a que la misma señal de voltaje puede ser usada para activar más de un generador de sonido, el control por voltaje permite un disparo automático de los diferentes controles del sonido.

- Un sintetizador es una unidad modular y puede ser fácilmente expandido, gracias a la adición de otros elementos controlados por voltaje, tales como unidades de efectos y otros.

- Los sintetizadores pueden producir sonidos en tiempo real.

- Los componentes primarios del sintetizador moderno son: osciladores para generar las formas de onda de audio, elementos para alterar el sonido tales como: filtros pasa banda y moduladores, amplificadores, unidades de reverberación, componentes controlados por voltaje y generalmente teclados.

En 1964, Robert Moog construyó un prototipo de un sintetizador controlado por voltaje. Entre los elementos de este sintetizador se menciona:

- Un teclado de cinco octavas para el control del voltaje, el cual podía ser seleccionado para operar ya sea con la escala cromática o con escala microtonal.

- Sus generadores de sonido eran solamente monofónicos, es decir producía un solo sonido a la vez.

- Los osciladores controlados por voltaje, tenían un rango de frecuencia bastante alto, desde 0.1 Hz hasta 40 KHz. Debido a que el rango auditivo del ser humano va solo desde 20 Hz hasta 20 KHz, el sintetizador de Moog permitía el uso de frecuencias subsónicas y ultrasónicas, para usarse en funciones de modulación. La generación de sonidos estaba basada en dos osciladores controlados por voltaje (VCO). Cada VCO podía ser seleccionado para que genere ya sea ondas sinusoidales, dientes de sierra, triangulares u ondas cuadradas.

- Los amplificadores eran controlados por voltaje (VCA), de igual manera los filtros eran también controlados por voltaje.

- Contaba con un generador de envolvente que permitía controlar el tiempo inicial de crecimiento, tiempo inicial de decaimiento y tiempo final de decaimiento de la señal de salida.

- Permitía además la conexión de otros módulos, tales como: reverberación, generadores de ruido blanco y rosa, vocoders (procesadores analógicos de voz), desviadores de frecuencia, etc.

En 1975 Moog introdujo el primer sintetizador polifónico, este instrumento fue el primero en usar tecnología de circuitos integrados para el control del hardware generador de sonido. Este aparato permitió reemplazar el alambrado de los sintetizadores anteriores, además de incluir preajustes de sonidos programados.

Otro de los populares sintetizadores de la década de los 60 fue el desarrollado en 1966 por, Donald Buchla, quien creó un aparato analógico controlado por voltaje, al que llamó Sistema Sintetizador Modular Buchla. El sintetizador Buchla constaba de: osciladores controlados por voltaje, amplificadores y filtros, todos creados con elementos de estado sólido. Aunque las capacidades de este sintetizador eran bastante similares a las de Moog, la diferencia entre los dos radicaba esencialmente en que el sintetizador de Buchla incluía un secuenciador, para la programación de una serie de voltajes de control. Esta innovación, realmente fue la introducción de un elemento de control automático, que es común a los sintetizadores de la actualidad. Otra gran diferencia entre los dos sintetizadores radicaba en que el sintetizador de Buchla no incluía un teclado comó elemento de control, en lugar de éste, tenía 16 platinas sensibles al tacto, las cuales eran manualmente programadas conectando pequeños alambres en el panel de control de manera similar a los computadores analógicos de la época.

En la década de los setenta, uno de los grandes fabricantes de sintetizadores fue Arp Instruments, de Estados Unidos, cuyo primer sintetizador de estudio fue creado en 1970 bajo el nombre de Arp 2500, era portable e incluía preajustes para varias de las conexiones que anteriormente se hacían con alambres. El Arp 2600 incluía además un amplificador y parlantes para monitoreo, lo que era poco usual en un sintetizador portable de la época. Una deficiencia del mismo era el bajo rango de frecuencia (0.03 Hz - 20 KHz), el que no permitía el rico espectro de ondas ultrasónicas que se tenía en los sintetizadores de Buchla y Moog.

1.3 ETAPA DE LA SINTESIS COMPUTARIZADA

En la década de los 80, las capacidades de los sintetizadores analógicos ha sido completamente superada al aparecer sistemas de síntesis digital basados en computador. Este fenómeno está estrechamente relacionado con la popularidad y desarrollo alcanzado por sistemas computarizados que prácticamente han eliminado a los sistemas analógicos anteriores. Es importante señalar que inicialmente aparecieron sistemas de síntesis híbridos, en los que se combinaba un control computarizado con componentes de síntesis analógicos.

Los sistemas analógicos basaban su funcionamiento en la generación de vibraciones eléctricas,

las que son usadas para actuar fisicamente sobre sistemas de parlantes. Así, el máximo desarrollo en la síntesis analógica fue el control por voltaje de sus componentes, esto permitía un alto grado de control sobre la producción de secuencias analógicas de voltajes para la producción de sonido.

La aplicación de sistemas digitales a la creación de información sonora ha tenido varias etapas, las mismas que se describen a continuación.

- En composición musical, en la cual la computadora ayuda al compositor a producir una partitura, la misma que puede ser ejecutada con cualquier instrumento. Esta práctica fue usada ya desde 1955 por Leonard Isaacson. Un gran computador creaba secuencias de notas que eran entonces transformadas en partituras. El aparecimiento de herramientas más poderosas para el procesamiento de información digital ha permitido en la actualidad crear lo que se ha denominado música asistida por computador.

- En síntesis computarizada de sonido, donde la computadora procesa códigos digitales para crear sonidos directamente. Esto se logra usando conversores D / A, que transforman dichos códigos en señales analógicas.

- Control computarizado sobre sintetizadores analógicos, en este caso los sonidos son producidos por los componentes normales de un sintetizador analógico, pero un computador genera los voltajes de control para los diferentes elementos controlados por voltaje. Este campo de aplicación tuvo su auge a finales de la década de los setenta, gracias al aparecimiento de los microprocesadores. Este tipo de sistemas híbridos permitió el aparecimiento de los secuenciadores, que asisten al sintetizador a ejecutar sus funciones.

A pesar del gran potencial de los sintetizadores analógicos controlados por voltaje, estos tenían limitaciones en varios aspectos. Primero, la creación y estructuración de cada sonido era una operación manual, que involucraba el ajuste de potenciómetros y conexiones de cables, las mismas que requerían experiencia por parte del usuario. Además, si se necesitaba repetir algún preajuste, el proceso se tornaba muy laborioso debido a la falta de un sistema de memoria del sintetizador.

La aplicación de sistemas digitales al procesamiento de sonido ha facilitado grandemente el procesamiento de información musical, trátese ésta de sonidos mismos o de secuencias de voltajes para controlar el sintetizador. Al poder almacenar esta información y tener un acceso aleatorio a la misma, la manipulación de ésta es mucho más fácil que en los sistemas de grabadoras y

mezcladoras de cinta magnética.

El primer sintetizador de sonido completamente digital, fue introducido en 1975 por John Appleton. Este sintetizador llamado sinclavier, permitía el almacenamiento de pistas o tracks de sonido que podían interpretarse mezclándose con interpretaciones en tiempo real. Desde entonces la aplicación de sistemas computarizados a la síntesis del sonido ha tenido tres áreas básicas:

- Aparatos portables, orientados al consumo general.

- Aparatos que se comportan como periféricos, a un sistema personal de computación.

- Sistema integrados, que contienen un computador. Estos están generalmente orientados a grandes estudios de sonido.

En 1979 la corporación CASIO de New Jersey, descubrió una nueva aplicación para los microprocesadores: la síntesis de sonido. En 1980 CASIO introdujo en el mercado su primer sintetizador portable de sonido. El secreto tras de este pequeño aparato, era su tecnología completamente computarizada, en la que los sonidos eran generados por un microprocesador. El éxito alcanzado por este aparato motivó a CASIO a crear una línea completa de

sintetizadores portables de sonido, en los cuales la generación de sonidos está completamente computarizada. Para finales de la década anterior, aparecieron en el mercado de sintetizadores computarizados portables, aparatos desarrollados por compañías como YAMAHA y ROLAND.

En la década pasada la disponibilidad de microprocesadores cada vez más rápidos y poderosos permitió el uso de computadores personales en aplicaciones de síntesis de sonido. Una de las compañías pioneras en el desarrollo de software para aplicaciones musicales fue: Syntauri Corporation de California. La misma que en 1980 presentó un sistema llamado Alpha Syntauri, el cual estaba diseñado para un computador Apple II. Entre otras opciones, este sistema permitía diseñar gráficamente las formas de onda de sonido, seleccionar formas de onda preajustadas, almacenar cualquier información y ejecutar cualquier sonido por medio del teclado. En la figura 1.1 se presenta el diagrama de Alpha Syntauri.

Una posibilidad más económica de crear un sistema sintetizador basado en un computador personal, fue desarrollado por ROLAND. El Compu Music CMU - 880R, consistía en un aparato externo al computador que contenía seis generadores digitales de tonos, controles de envolvente y controles de volumen. Este aparato se conectaba al computador vía un circuito interfaz.

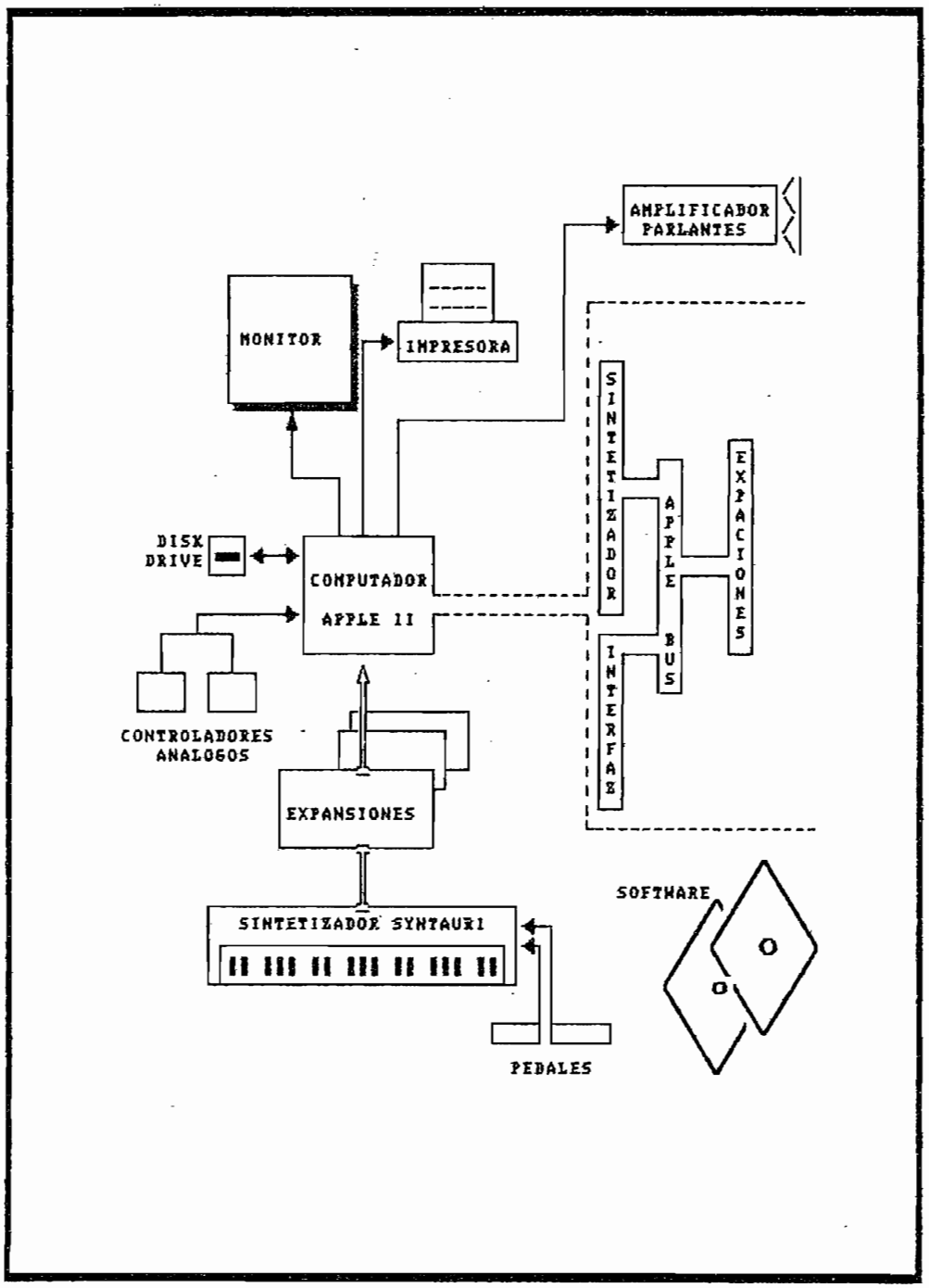


DIAGRAMA DE CONFIGURACION DEL SISTEMA SINTETIZADOR ALPHA SYNTAURI

FIGURA 1.1

A mediados de la década pasada aparecieron en el mercado sofisticados sistemas computarizados diseñados expresamente para aplicaciones en síntesis de sonido. Por sus características, estos sistemas estaban sólo al alcance de grandes estudios de sonido; el más común de ellos fue el CMI, desarrollado por Fairlight. Este permite un control completo para el diseño de formas de onda gracias a un computador que viene incluido en esta unidad sintetizadora. El CMI, incluía capacidades de secuenciador, así como un banco de 400 sonidos preajustados y la posibilidad de crear cualquier escala tonal.

En el año 1984, se dio el paso más importante tendiente a estandarizar los sistemas de síntesis de sonido creados por diferentes industrias. En el marco anteriormente señalado se creó un protocolo de comunicación digital, que pronto se convertiría en una poderosísima herramienta para satisfacer cualquier requerimiento de un sistema sintetizador de sonido. Este protocolo recibió el nombre de MIDI, que es la abreviatura inglesa de: Musical Instruments Digital Interface.

En el capítulo III, se presenta una descripción muy detallada del protocolo de comunicación MIDI.

C A P I T U L O II

METODOS DE SINTESIS DEL SONIDO

METODOS DE SINTESIS DE SONIDO

Desde el aparecimiento de métodos para generar sonidos eléctricamente, los investigadores tropezaron con problemas técnicos de orden práctico, es así, que en principio estos mecanismos podían generar un sonido, pero a diferencia de los instrumentos naturales, los sonidos sintéticos tenían deficiencias especialmente en el nivel de ataque, ausencia de armónicos, niveles de restitución, etc

El paso de los años ha tendido ha eliminar estas limitaciones técnicas, es así que en la actualidad los sonidos sintéticos son exactamente iguales a los que podrían obtenerse con cualquier instrumento tradicional, siendo prácticamente imposible diferenciar los dos sonidos aún para oídos muy experimentados. En muchos casos la riqueza armónica y melódica es superada por los sonidos sintéticos.

Los mecanismos de síntesis actuales no han permitido sólo reemplazar a los medios tradicionales, sino han permitido crear o "inventar" sonidos que no han sido escuchados nunca antes y que no pueden crearse por medios naturales.

Debido a que cualquier método de síntesis, está basado en el principio fundamental de que cualquier

sonido se construye a partir de sus principios básicos, es necesario antes de afrontar el problema de la síntesis del sonido, establecer ciertas bases teóricas de acústica, así como analizar brevemente como el oído humano reacciona ante las variaciones de los diferentes parámetros del sonido.

2.1 COMPONENTES DEL SONIDO

Se puede definir al sonido como la sensación que experimenta el oído cuando reacciona a una cierta gama de vibraciones. Desde otro punto de vista, estas mismas vibraciones pueden definirse como sonido.

Uno de los principales investigadores, fue Hermann von Helmotz, quien en 1862 publicó su trabajo llamado "Sensación del tono", demostrando que los sonidos podían ser analizados partiendo de unos pocos principios físicos.

El primer investigador en establecer la relación entre la física del sonido y la creación de sonidos musicales fue John Cage, quien en 1937 desarrolló investigaciones en las que, concluyó que el sonido era el resultado de la manipulación de cuatro funciones básicas: amplitud o intensidad, frecuencia o tono, timbre y envolvente.

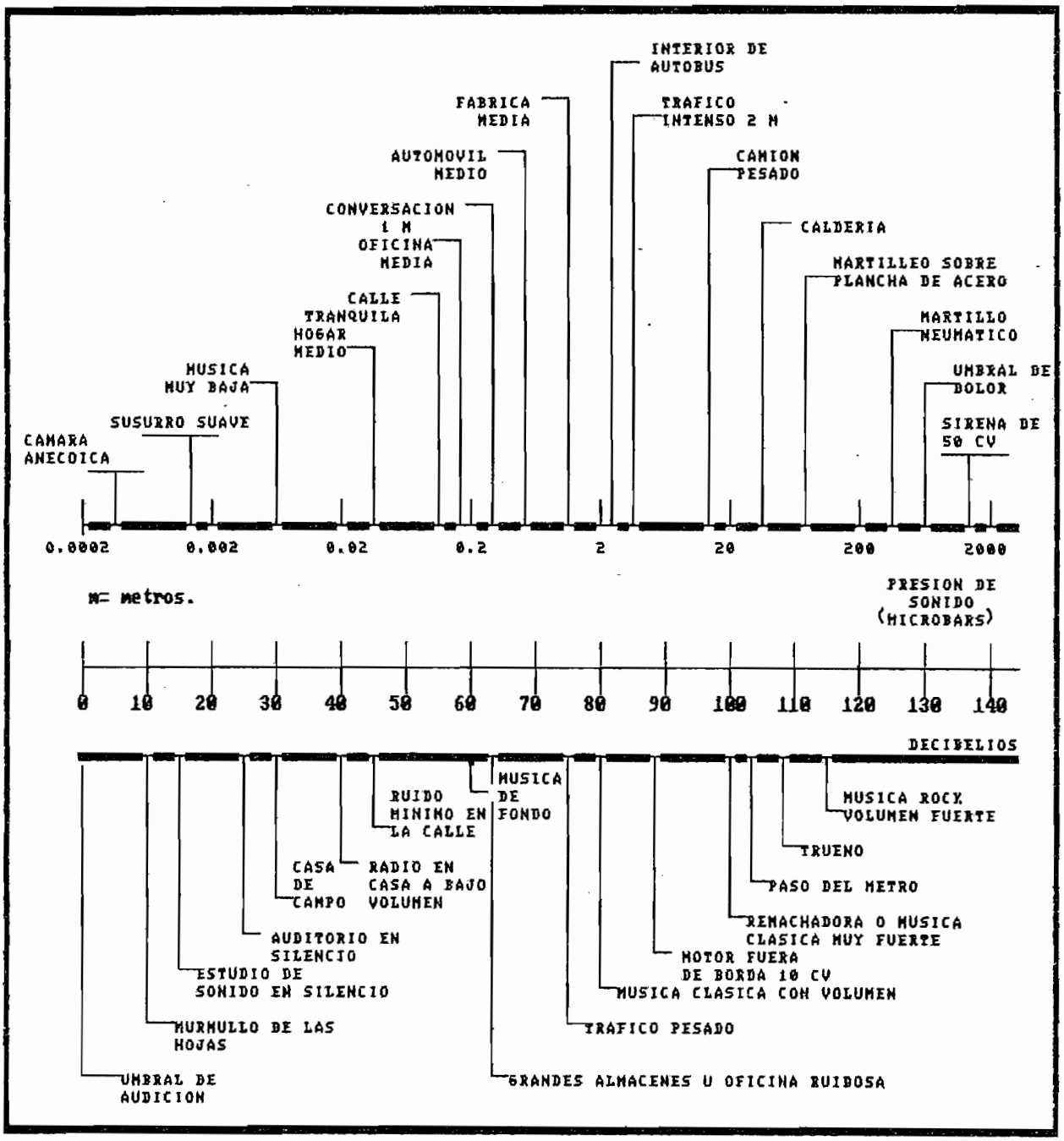
El primer parámetro que identifica a un

sonido es el conocido cómo intensidad. A primera vista parece que la intensidad como elemento del sonido no es tan importante como los otros elementos, pero esto no es así. Intensidad es la característica que hace que un sonido sea percibido más o menos fuerte por el oído. (2)

La intensidad del sonido es proporcional a la energía vibratoria que capta el oído, siendo ésta proporcional al cuadrado de la amplitud de la misma. Se ha determinado que para una persona el umbral de audición es 0.0002 µbar y el umbral de dolor es aproximadamente 1.0 µbar. En la figura 2.1 se indica los niveles de presión acústica de algunos sonidos comunes.

Otra característica del sonido es la llamada frecuencia, tono o altura. Esta nos permite diferenciar un sonido grave de uno agudo. El tono viene determinado por el número de vibraciones que se producen en un determinado período de tiempo. A mayores variaciones por período de tiempo, más agudo es el sonido.

Los umbrales de respuesta del oído humano a la frecuencia reciben el nombre de espectro de frecuencias audibles, y para el hombre normal, joven, va aproximadamente desde 20 Hz hasta 17000 Hz. El oído no puede apreciar fielmente vibraciones con frecuencias menores a 30 Hz, pero bastan muy pocos armónicos para guiarlo. Para un oído normalmente constituido, a partir de



NIVELES DE INTENSIDAD DE SONIDOS COMUNES

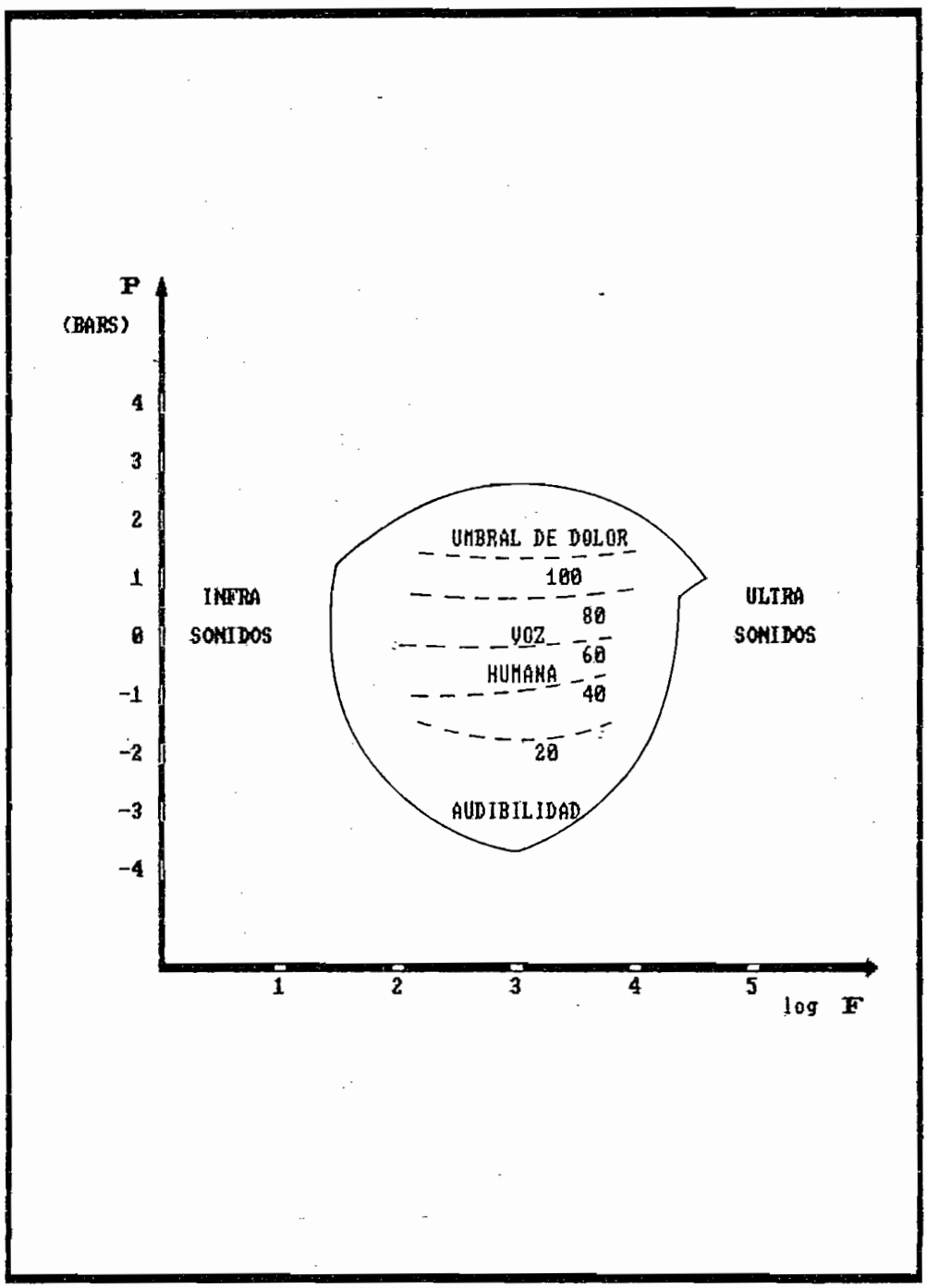
FIGURA 2.1

frecuencias superiores a 12000 Hz, sus armónicos caen dentro de los ultrasonidos.

Podemos deducir, que en el caso de un instrumento monódico, que emite sólo una sucesión de notas individuales, será totalmente diferente del de los instrumentos polifónicos que realizan acordes. Supongamos una sala donde se emiten dos sonidos de frecuencias f_1 y f_2. Además de estas vibraciones aparecen resultantes denominadas diferenciales, dadas por $f_1 - f_2$ (o $f_2 - f_1$). Por tanto, si se tienen más de dos frecuencias aumenta el número de combinaciones posibles y por consiguiente también el de diferenciales. En realidad es el mismo oído el que crea en parte este fenómeno, pues si se envía cada una de las dos frecuencias por un auricular, colocado cada uno en un oído, es muy difícil para el oyente percibir las frecuencias diferenciales.

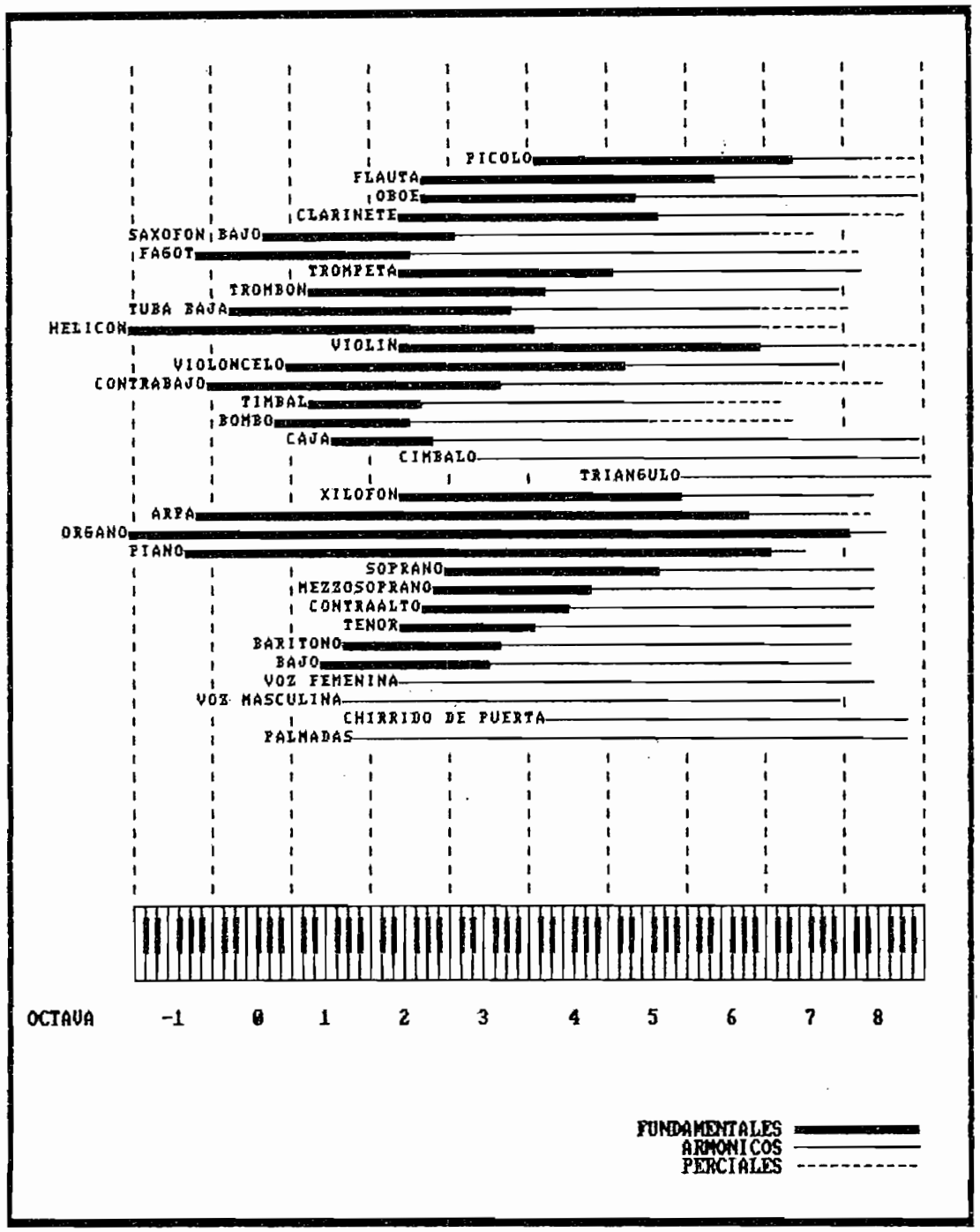
En la figura 2.2, se indica gráficamente la curva de respuesta de frecuencia del oído humano. Este gráfico visualiza la relación entre el logaritmo de la frecuencia versus la presión de intensidad acústica.

Es interesante comparar los espectros de frecuencia cubiertos por los diferentes instrumentos conocidos. La figura 2.3 presenta una clara visualización del rango de varios sonidos. Se observa que el órgano es el instrumento que abarca el espectro más amplio de



RESPUESTA DE FRECUENCIA DEL OIDO HUMANO

FIGURA 2.2



RANGOS DE FRECUENCIA FUNDAMENTALES, ARMONICOS Y PARCIALES DE VARIOS INSTRUMENTOS

FIGURA 2.3

frecuencia. Con los métodos de síntesis modernos, el espectro de frecuencias se amplía grandemente, incluso hasta el campo de los ultrasonidos.

El valor universal de referencia de frecuencia es de 440 Hz (la_3). A título indicativo se presenta en la tabla 2.1 una recapitulación de las frecuencias de todas las notas de la escala temperada. El valor más agudo en esta escala corresponde a 15804.25 Hz, valor elevado para la respuesta de frecuencia del oído humano, inversamente, el valor más bajo corresponde a 16.35 Hz, la misma que también está en el límite de audición del oído humano.

Otro de los parámetros del sonido es el timbre o color tonal, el cual es aquel factor del sonido que nos permite distinguir entre dos fuentes sonoras que producen una misma nota sostenida.

Las ondas sonoras son el resultado de una vibración. Las vibraciones de la mayor parte de sistemas vibratorios tienden a ser bastante complejas ya que vibran a varias frecuencias simultáneamente. Es la combinación e interacción de estas frecuencias, llamadas armónicos o sobretonos, las que dan al sonido la calidad que conocemos como timbre.

El timbre generalmente se ve muy afectado

N.OCTAVA ⇒ / NOTA ⇓	-1	0	1	2	3	4	5	6	7	8
DO	16.35	32.70	65.40	130.80	261.62	523.25	1046.5	2093.0	4186.0	6372.0
DO #	17.32	34.64	69.29	138.59	277.18	554.36	1108.7	2217.4	4434.9	8869.8
RE	18.35	36.70	73.41	146.83	293.66	587.32	1174.6	2349.3	4698.6	9397.2
RE #	19.44	38.89	77.78	155.56	311.12	622.25	1244.5	2489.0	4978.0	9956.0
MI	20.60	41.20	82.40	164.81	329.62	659.25	1318.5	2637.0	5274.0	10548.0
FA	21.82	43.65	87.30	174.61	349.22	698.45	1396.9	2793.8	5587.6	11175.2
FA #	23.12	46.24	92.49	184.99	369.99	739.98	1479.9	2959.9	5919.9	11839.8
SOL	24.49	48.99	97.99	195.99	391.99	783.99	1567.9	3135.9	6270.9	12541.8
SOL#	25.95	51.91	103.82	207.65	415.30	830.60	1661.2	3322.4	6644.8	13288.7
LA	27.50	55.00	110.0	220.0	440	880.0	1760.0	3520.0	7040.0	14080.0
LA #	29.13	58.27	116.54	233.08	466.16	932.32	1864.6	3729.3	7568.6	15177.2
SI	30.86	61.73	123.47	248.94	493.88	987.76	1975.5	3951.0	7902.1	15804.2

TABLA DE FRECUENCIAS (Hz) DE LAS NOTAS DE LA ESCALA TEMPERADAS

TABLA 2.1

por el tono y la envolvente, es por eso que cuando se sintetizan sonidos, es generalmente preferible dejar el timbre para la etapa final.

La forma de onda, es el factor que tiene más influencia en las características del timbre de un sonido; es decir, la forma de onda determina la familia del instrumento básico a la cual corresponde el sonido.

Para sintetizar sonidos se requiere una gran cantidad de formas de onda, pero éstas pueden sintetizarse a partir de algunas ondas básicas: sinusoidal, triangular, diente de sierra y cuadrada.

Otra fuente básica para generar formas de onda para sintetizar sonidos, pero que no exhibe la rigidez de las ondas anteriores, son las llamada señales de ruido, sean éstos ruido blanco o ruido rosa. Todos los circuitos electrónicos generan una cierta cantidad de ruido, en la mayoría de los casos el ruido es un factor indeseable. Sin embargo en síntesis de sonido, frecuentemente el ruido es el punto de partida para el timbre de sonidos sin tono definido o de elementos no tonales en un sonido.

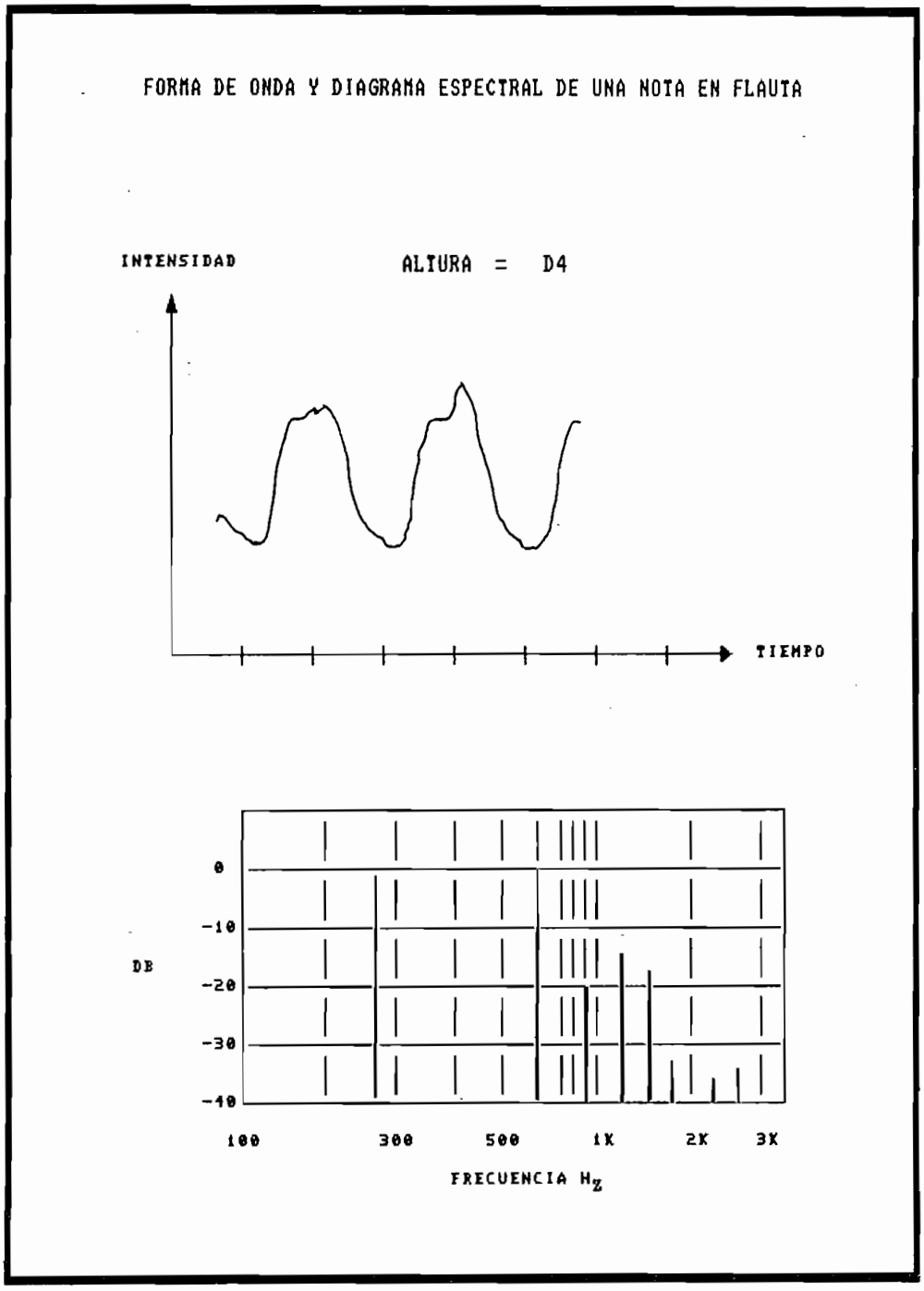
El ruido blanco ocurre cuando todas las características de frecuencia y amplitud de un sonido ocurren aleatoriamente dentro de un amplio rango del espectro de frecuencia. Más precisamente, el ruido blanco

contiene todas las frecuencias audibles entre 18 Hz y 22000 Hz, en tanto el ruido rosa, es aquel que contiene cantidades iguales de energía en cada octava. Como cada octava tiene igual energía, este tipo de ruido suena a nuestros oídos como si tuviera una cantidad igual de todas las frecuencias y puede ser filtrado y refinado para sonar de manera similar al viento, en el océano. Este ruido tiene las frecuencias contenidas entre 18 Hz y 1000 Hz. (2)

Las formas de onda de los diferentes instrumentos son en realidad mucho más complejas que las ondas básicas mencionadas. Para obtenerlas se usa diferentes tipos de filtros (síntesis analógica), para así añadir los armónicos requeridos. En las figuras 2.4, 2.5 y 2.6, se presentan los oscilogramas y los diagramas espectrales para tres instrumentos musicales; se observa el importante número de armónicos contenidos en las sonoridades de una flauta, trompeta y clarinete respectivamente.

Al tiempo durante el cual un sonido es audible se conoce como duración. Los métodos de síntesis electrónica introducen la posibilidad de sostener un sonido indefinidamente.

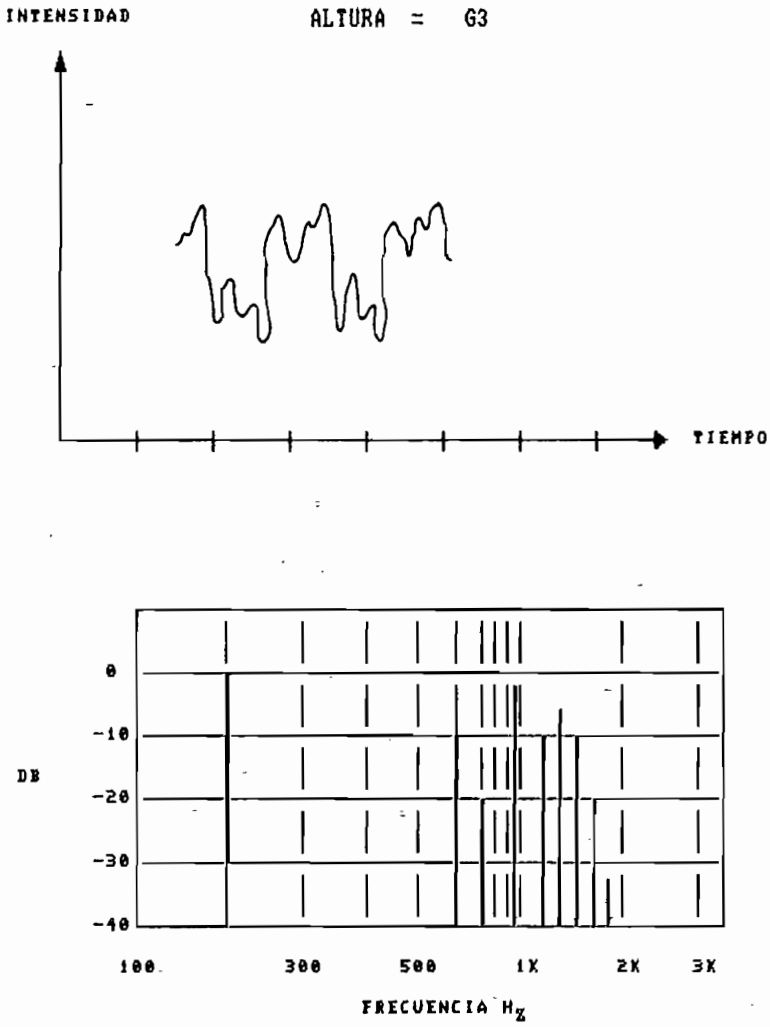
Los parámetros: intensidad, tono, timbre y duración, si bien definen un sonido de manera correcta, lo hacen insuficientemente. Para definir un sonido de manera



OSCILOGRAMA Y DIAGRAMA ESPECTRAL DE UN SONIDO EN FLAUTA

FIGURA 2.4

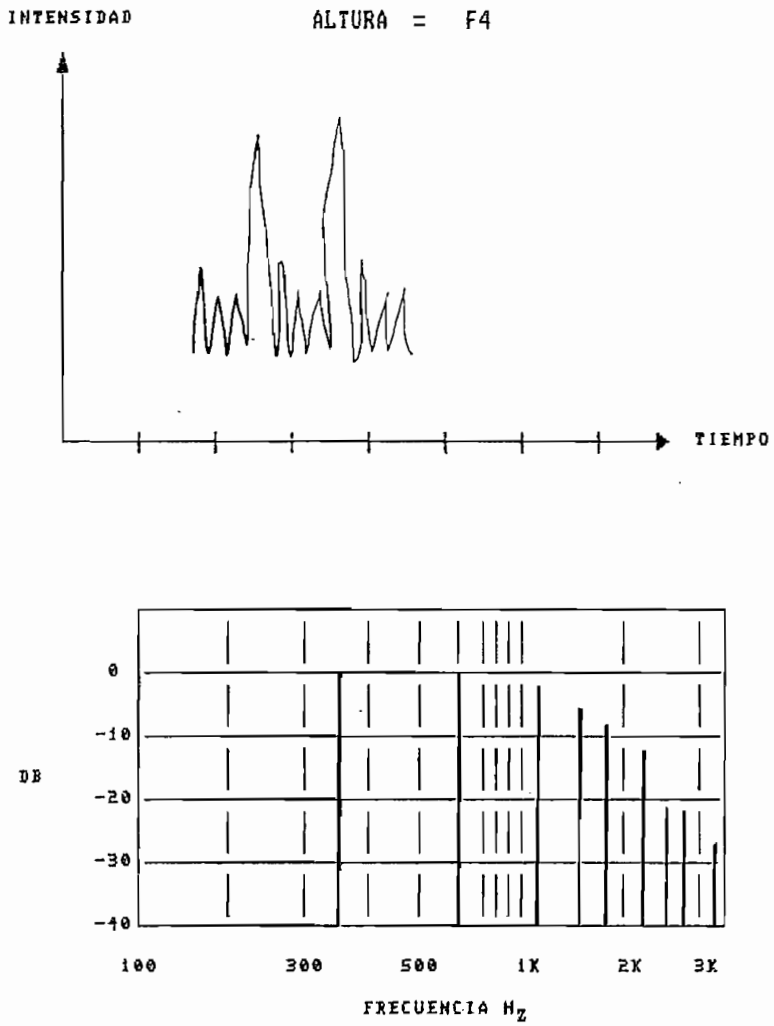
FORMA DE ONDA Y DIAGRAMA ESPECTRAL DE UNA NOTA EN CLARINETE



OSCILOGRAMA Y DIAGRAMA ESPECTRAL DE UN SONIDO EN CLARINETE

FIGURA 2.5

FORMA DE ONDA Y DIAGRAMA ESPECTRAL DE UNA NOTA EN TROMPETA



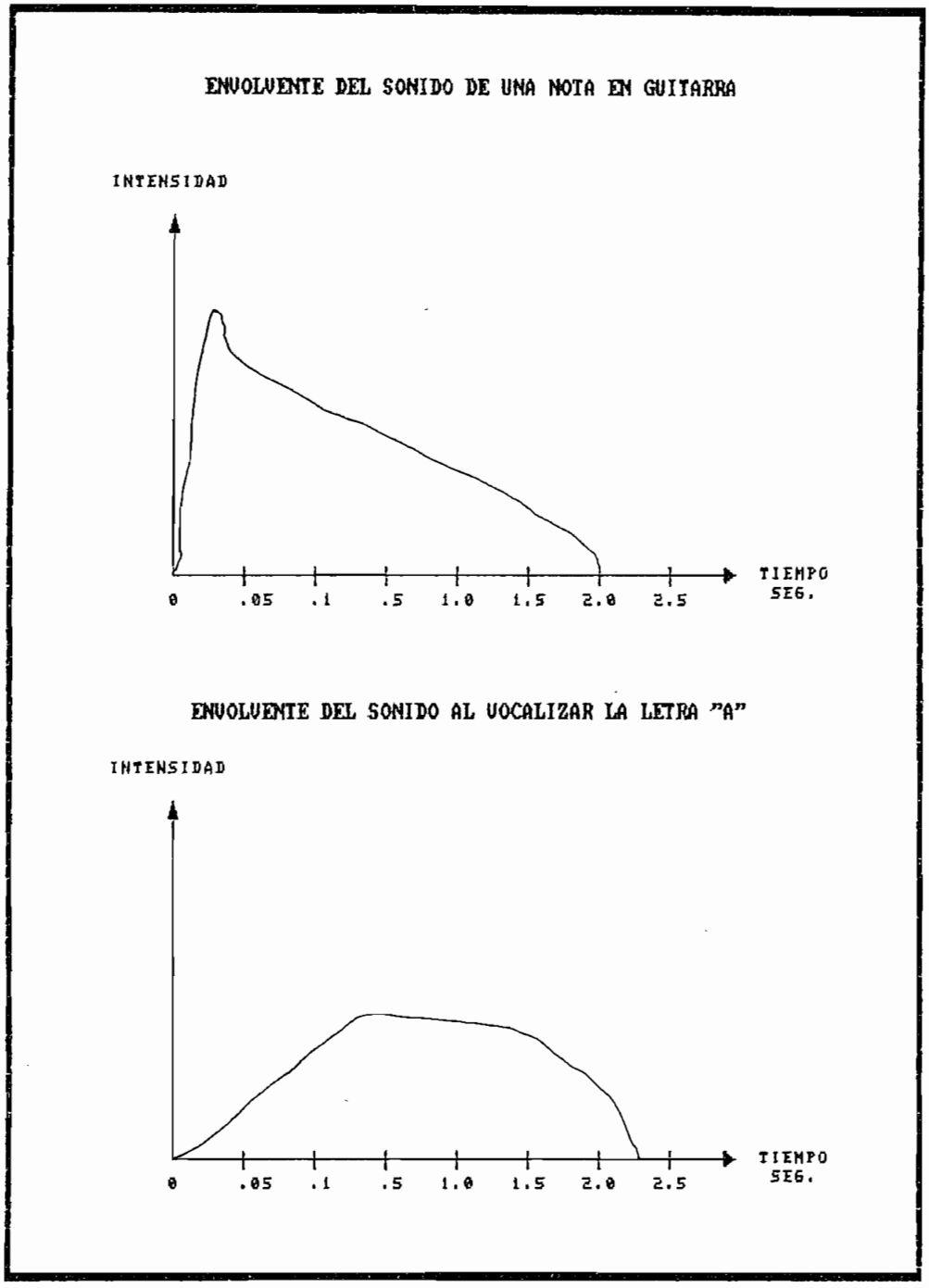
OSCILOGRAMA Y DIAGRAMA ESPECTRAL DE UN SONIDO EN TROMPETA

FIGURA 2.6

completa es necesario usar parámetros dinámicos.

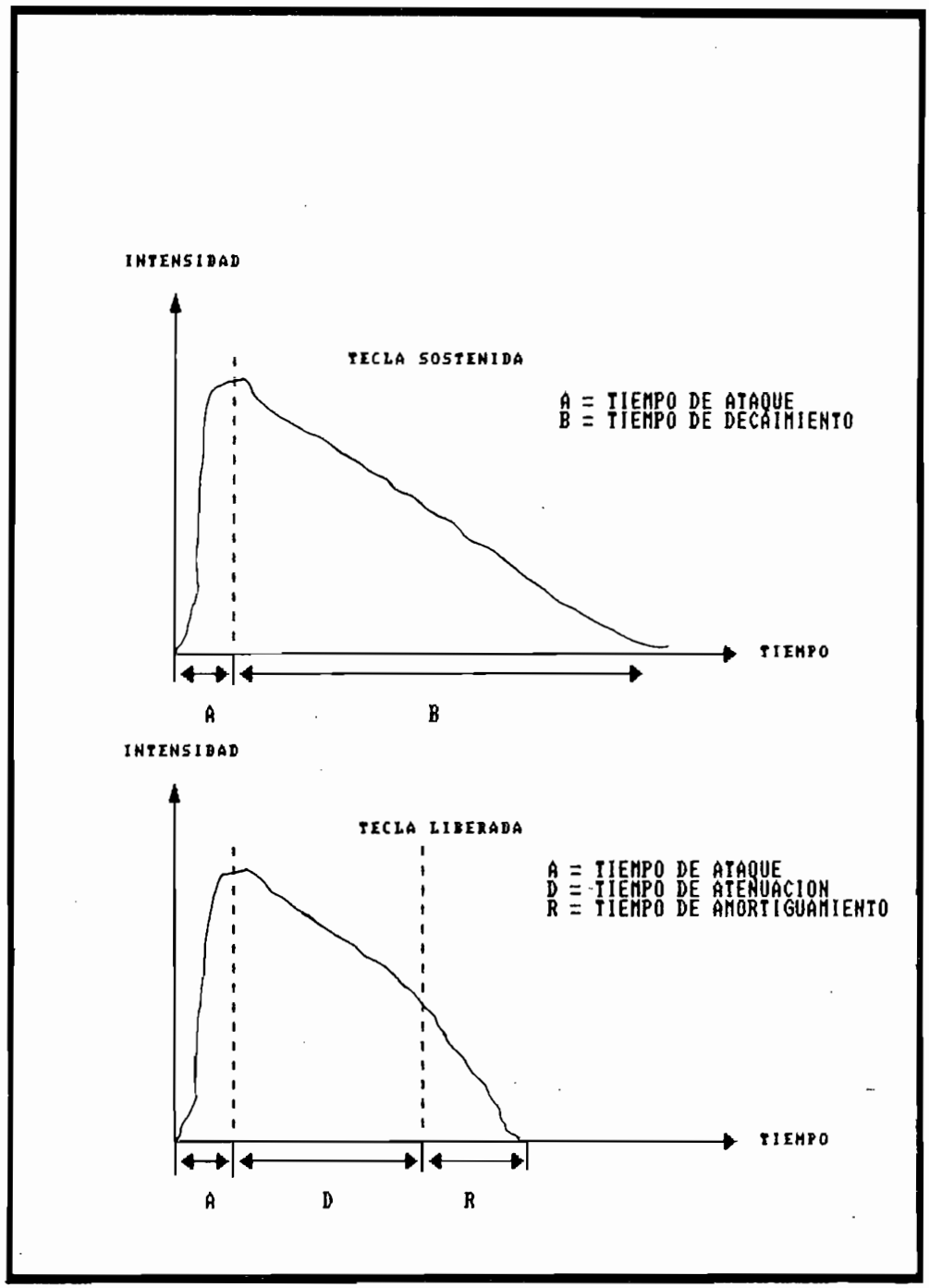
Un parámetro muy relacionado con la intensidad y la duración es el conocido como morfología o envolvente. La envolvente de un sonido hace que su intensidad varíe durante su período de duración. En la figura 2.7 se indica los modelos de envolventes de una nota de guitarra y del sonido al pronunciar la letra "a". En la primera, tan pronto como se suelta la cuerda, el sonido crece con rapidez a su valor máximo y después se atenúa con lentitud. El nivel máximo que se alcanza y el tiempo que se necesita para que el sonido se extinga dependerá de la fuerza aplicada al tañir la cuerda. En el caso de la envolvente de la letra "a", ésta podrá variar grandemente dependiendo de la manera como sea vocalizada.

La envolvente determina las características de ataque y decaimiento de un sonido, es decir la manera como éste crece y termina. En la figura 2.8 se presenta la envolvente de una nota tocada al piano, en ésta se puede apreciar sus diferentes características. El tiempo necesario para que el sonido llegue hasta su máximo intensidad se conoce como tiempo de ataque. El tiempo que precisa un sonido para su extinción se conoce como tiempo de decaimiento. Si se suelta la tecla del piano antes que se haya extinguido el sonido, éste se amortiguará rápidamente. Esto se llama tiempo de amortiguación.



CURVAS ENVOLVENTES DE DOS DIFERENTES SONIDOS

FIGURA 2.7



ENVOLVENTE DE NOTAS EN PIANO (TECLA SOSTENIDA Y TECLA LIBERADA)

FIGURA 2.8

Gracias al carácter experimental de los modernos medios de síntesis, éstos han permitido modelar los sonidos de una manera más precisa y variada que los instrumentos tradicionales. Estudiando la manera como interaccionan las variaciones de éstos y el oído humano, es posible definir características que no se las conocía con anterioridad.

2.2 METODOS DE SINTESIS ANALOGICA

Los métodos de procesamiento electrónico de tipo analógico, fueron los primeros en aplicarse para generar sonidos sintéticos. Por su naturaleza, éstos requerían grandes circuitos para procesar las diferentes señales. Los métodos de síntesis analógicos más importantes son los de adición y substracción de armónicos.

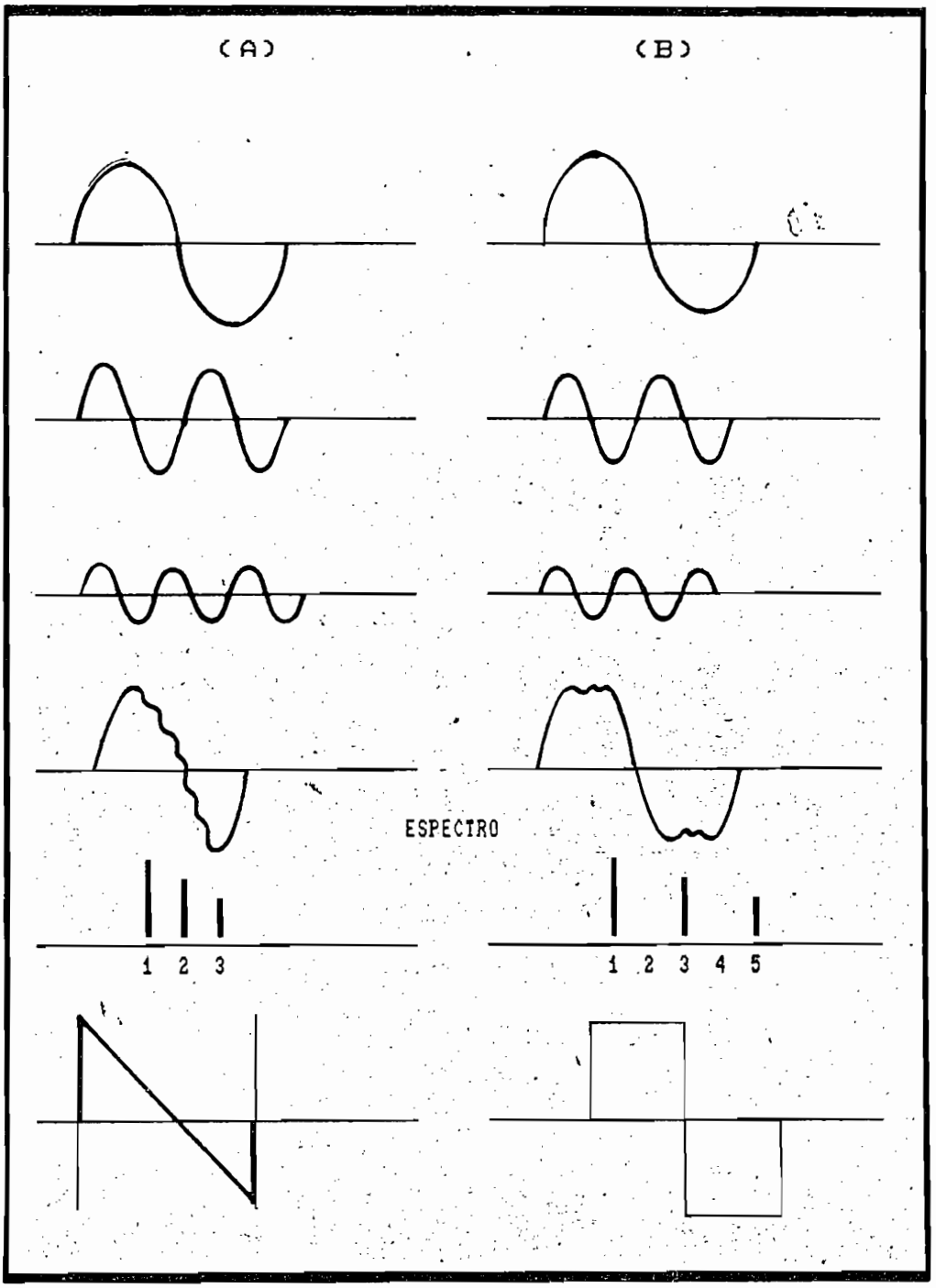
2.2.1 Síntesis por Adición

Este método de síntesis analógico, es uno de los más complicados métodos de síntesis disponibles en la actualidad. Su fundamento se basa en la adición de cada armónico a la señal fundamental. Cada sobretono que se añade, cambia sus características, así, a mayor cantidad de armónicos añadidos, el sonido es más brillante y parecido a un sonido producido por un instrumento real, en tanto que si los armónicos añadidos son pocos y cercanos a la fundamental, el sonido tiende a ser más artificial.

En la síntesis por adición partimos de ondas sinusoidales de diferentes frecuencias y las sumamos en sus proporciones correctas para producir un sonido con el contenido armónico adecuado. Este método requiere generalmente un número bastante grande de generadores de ondas sinusoidales (uno para cada armónico requerido). El grado de intensidad de cada sobretono, afecta directamente a las características del sonido, es por esta razón que los sintetizadores que usan este método de síntesis, permiten controlar la intensidad de cada sobretono separadamente, lo que significa que se puede controlar los más finos elementos de las características de timbre del sonido. Además, el contenido armónico de muchos sonidos varía durante su emisión, por lo tanto el control exacto de la proporción de cada sinusoide individualmente resulta ser algo muy complejo. Por estos motivos, la síntesis por adición es poco usada desde el punto de vista práctico. (3)

A pesar de lo anteriormente indicado, el método de síntesis por adición, es uno de los más flexibles, ya que involucra la manipulación de cada sobretono individualmente, así para sintetizar un sonido de un instrumento natural se requieren por lo menos la adición de 60 sobretonos.

En la figura 2.9 se presenta gráficamente el resultado de añadir dos armónicos de diferente magnitud a una señal sinusoidal fundamental, en ésta se observa la



SINTESIS ADITIVA - ADICION DE ARMONICOS

FIGURA 2.9

obtención de ondas de diferentes características, en este caso onda cuadrada y diente de sierra.

2.2.2 Síntesis por Eliminación

Este método de síntesis, fue popularizado con los sintetizadores de Moog y en la actualidad es uno de los más comunes, es por esta razón que se lo estudiará con mayor detalle que al método anterior.

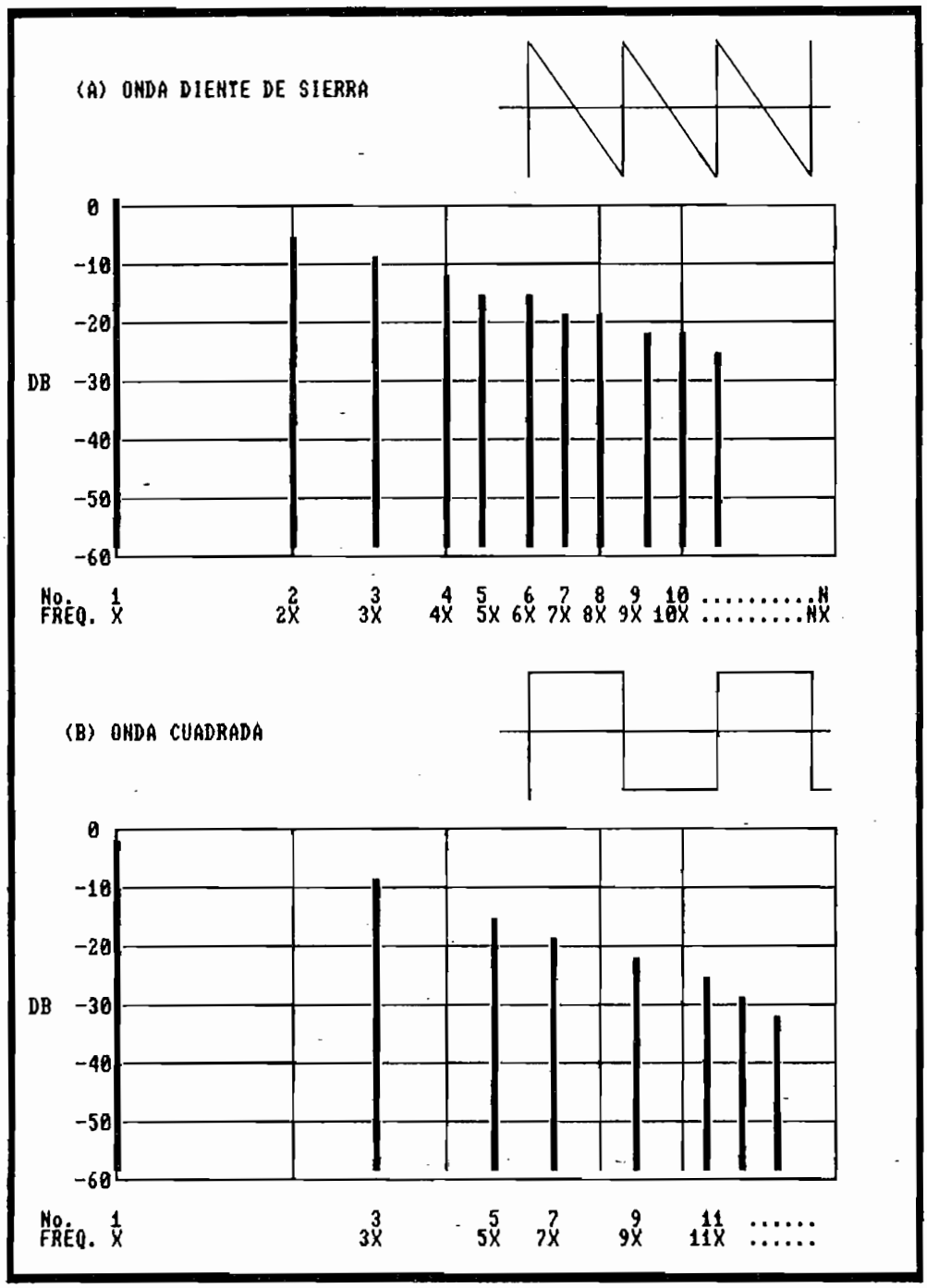
Inicialmente, este método fue usado en sintetizadores analógicos, con osciladores y amplificadores controlados por voltaje (VCO's y VCA's), pero el advenimiento de sistemas digitales ha permitido reemplazar el control por voltaje a un control digital de los osciladores y amplificadores, así en la actualidad los VCO's y VCA's han sido reemplazados por DCO's (Digitally Controlled Oscillator) y DCA (Digitally Controlled Amplifier). En esencia, el fundamento de este método de síntesis es el mismo de un sintetizador analógico o digital.

Su principio se basa en la eliminación de sobretonos, partiendo de un tipo de onda rica en armónicos. La onda primaria la obtenemos de un oscilador, que se ajusta para generar una onda rica en sobretonos. La salida del oscilador se conecta a un filtro que modifica el timbre. En la síntesis por eliminación se utiliza

generalmente como ondas primarias, ondas en forma de diente de sierra y cuadrada. En la figura 2.10 se indica el contenido armónico de estas ondas. La estructura de éstas, hace que sean muy fáciles de ser generadas electrónicamente y es relativamente sencillo controlar con precisión la frecuencia de las mismas, ésta es una consideración muy importante en la generación de los tonos musicales. (2)

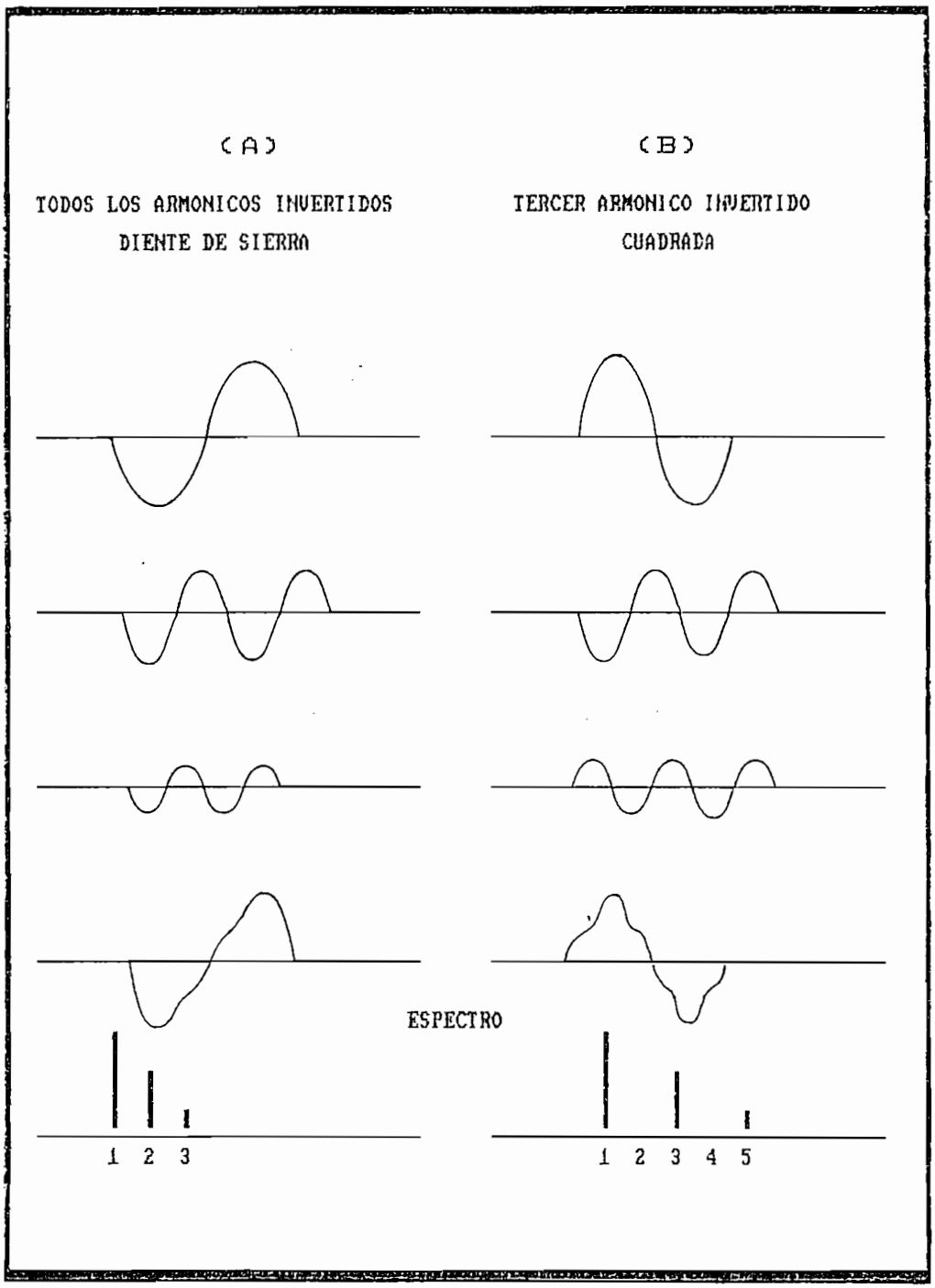
En la figura 2.11 (a) se presenta lo que sucede si sumamos tres ondas sinusoidales invertidas de frecuencias X, 2X y 3X. Como el sonido de una onda sinusoidal no varía cuando ésta se invierte, al sumar ondas invertidas o sin invertir, se obtienen exactamente los mismos sonidos. Por este motivo no tiene importancia si un sintetizador genera rampas ascendentes o descendentes como fuente de ondas diente de sierra.

En la figura 2.11 (b), observamos lo que sucede cuando intentamos sintetizar una onda cuadrada con armónicos impares, pero invertimos uno de ellos, en este caso el tercer armónico, de forma que este queda defasado respecto a los anteriores. El punto importante en este caso es que, aunque la onda resultante no se parece a una onda cuadrada, contiene los mismos armónicos de la onda cuadrada de la figura 2.9 (b), por lo tanto sonará de la misma forma. De esto se deduce claramente, que el timbre de una fuente sonora, dependerá del contenido armónico del sonido y generalmente no guardará relación con las fases de



ESPECTRO DE ONDA CUADRADA Y ONDA DIENTE DE SIERRA

FIGURA 2.10



INVERSION DE ARMONICOS

FIGURA 2.11

estos armónicos excepto en aquellas fuentes sonoras donde la fase cambia continuamente.

Para lograr una clara descripción de este método de síntesis, nos basaremos en el control analógico de sus diferentes elementos, (es decir control por voltaje), sin dejar de visualizar que este mismo control puede ser logrado digitalmente.

2.2.2.1 Síntesis de Tono

Como hemos visto el tono del sonido se encuentra muy relacionado con la frecuencia del mismo. En el sintetizador por eliminación la fuente más importante de señales de diferentes frecuencias o tonos es el oscilador controlado por voltaje (VCO).

Para el VCO una de las fuentes más comunes de voltaje es el teclado musical, el mismo que no es más que un divisor de voltaje; el nivel de voltaje de salida depende de la tecla que se pulsa. Si se aplica el voltaje de control a un VCO convenientemente calibrado, este generará un tono que estará en relación directa con la tecla pulsada.

En la tabla 2.1 se observa que la relación de frecuencias de las escalas musicales, no es lineal, esta relación corresponde a una progresión exponencial. Es por

esto que para establecer la relación voltaje de control de los osciladores y frecuencia de los sonidos generados existen dos maneras, las mismas que se describen a continuación.

En el VCO lineal, la frecuencia de salida del VCO está en relación directamente proporcional con el voltaje de control aplicada a los osciladores. Por ejemplo, tal VCO puede tener la siguiente relación voltaje - frecuencia:

Entrada Voltaje de control		Salida Frecuencia del VCO	
	1 voltio	= 110 Hz (la_2)	
PROGRESION	2 voltios	= 220 Hz (la_3)	PROGRESION
DE TENSION	3 voltios	= 330 Hz (mi_4)	TONAL
LINEAL	4 voltios	= 440 Hz (la_4)	EXPONENCIAL

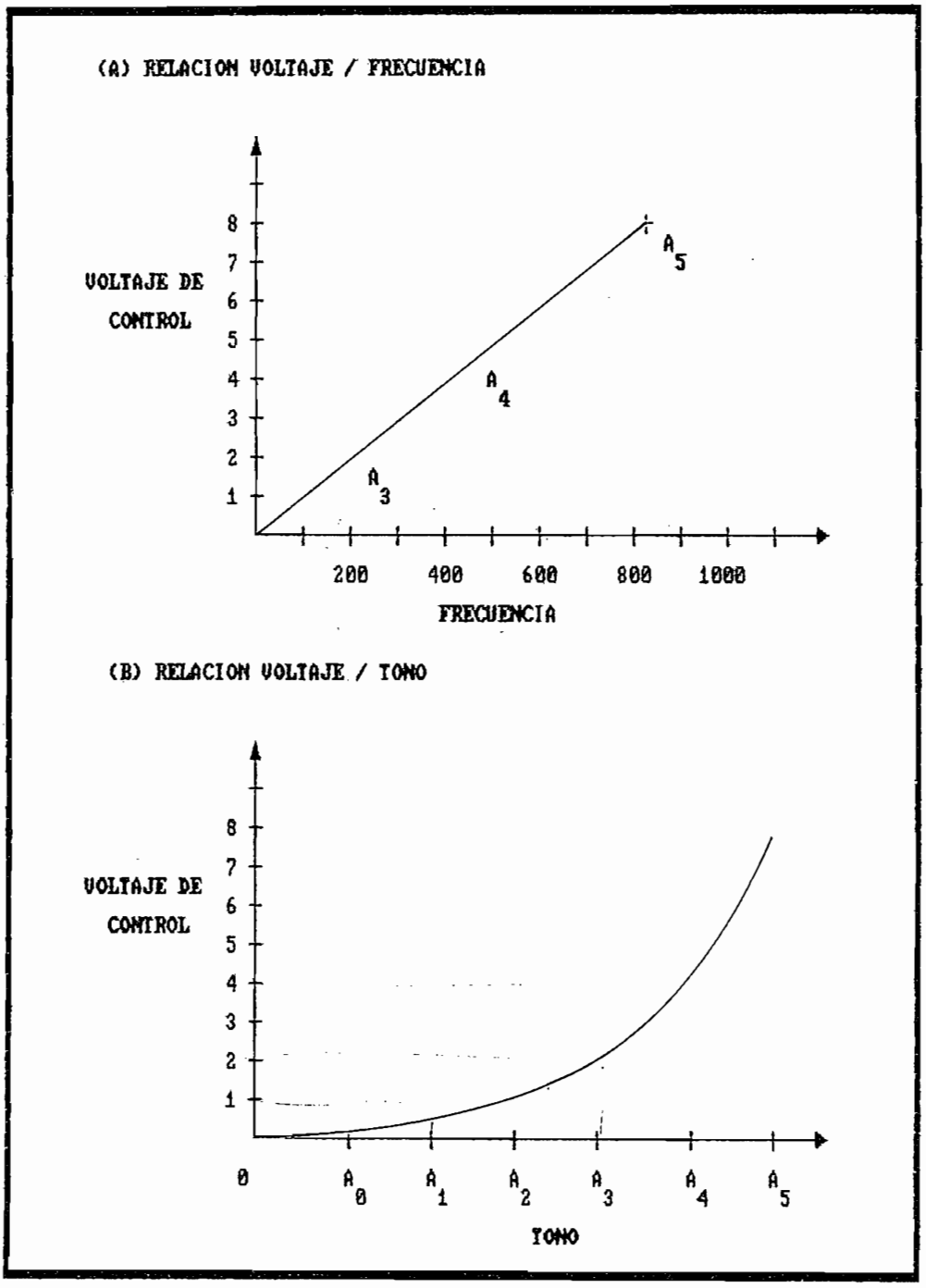
O para producir las octavas:

Entrada Voltaje de control		Salida Frecuencia del VCO	
	1 voltio	= 110 Hz (la_2)	
PROGRESION	2 voltios	= 220 Hz (la_3)	PROGRESION
DE TENSION	4 voltios	= 440 Hz (la_4)	TONAL
EXPONENCIAL	8 voltios	= 880 Hz (la_5)	LINEAL

etc...

Por lo tanto para producir una serie ascendente de octavas se requiere una fuente de voltaje que aumente exponencialmente.

En la figura 2.12 (a), se presenta una gráfica voltaje - frecuencia, para el VCO lineal anterior. En esta figura se justifica el término "lineal". En la figura 2.12 (b) se indica la gráfica voltaje - tono para el mismo VCO, la misma que es de tipo exponencial. Para producir una progresión de tonos, tal como octavas (como se representa en el ejemplo) o una escala cromática, se requerirán variaciones exponenciales en la fuente señales de voltaje. Este hecho plantea problemas a nivel de afinación y transposición. Si tenemos una fuente de voltaje que produce una secuencia exponencial de tensiones, como por ejemplo 1V, 2V, 4V, 8V y alimentamos esta secuencia al VCO lineal del ejemplo, este generará tonos que producirán saltos de una octava empezando en 110 Hz. (la_2) y terminando en 880 Hz (la_5), tal como se muestra en la figura 2.12 (b). Para transponer esta secuencia de notas a una octava superior, de forma que inicie en 220 Hz (la_3) y termine en 1760 Hz (la_6), sería necesario que multiplicáramos por 2 el voltaje de entrada, por lo tanto, el VCO lineal requiere la adición de un elemento multiplicador a su entrada para poder transponer libremente. También se necesita un teclado exponencial, es decir un teclado que genere pasos de voltaje que aumenten exponencialmente cuando se toca una escala cromática. Como



RELACIONES EN UN VCO LINEAL

FIGURA 2.12

el teclado a veces se utiliza para otras funciones del sintetizador, se deberían acomodar otros parámetros del sistema para satisfacer esta relación exponencial.

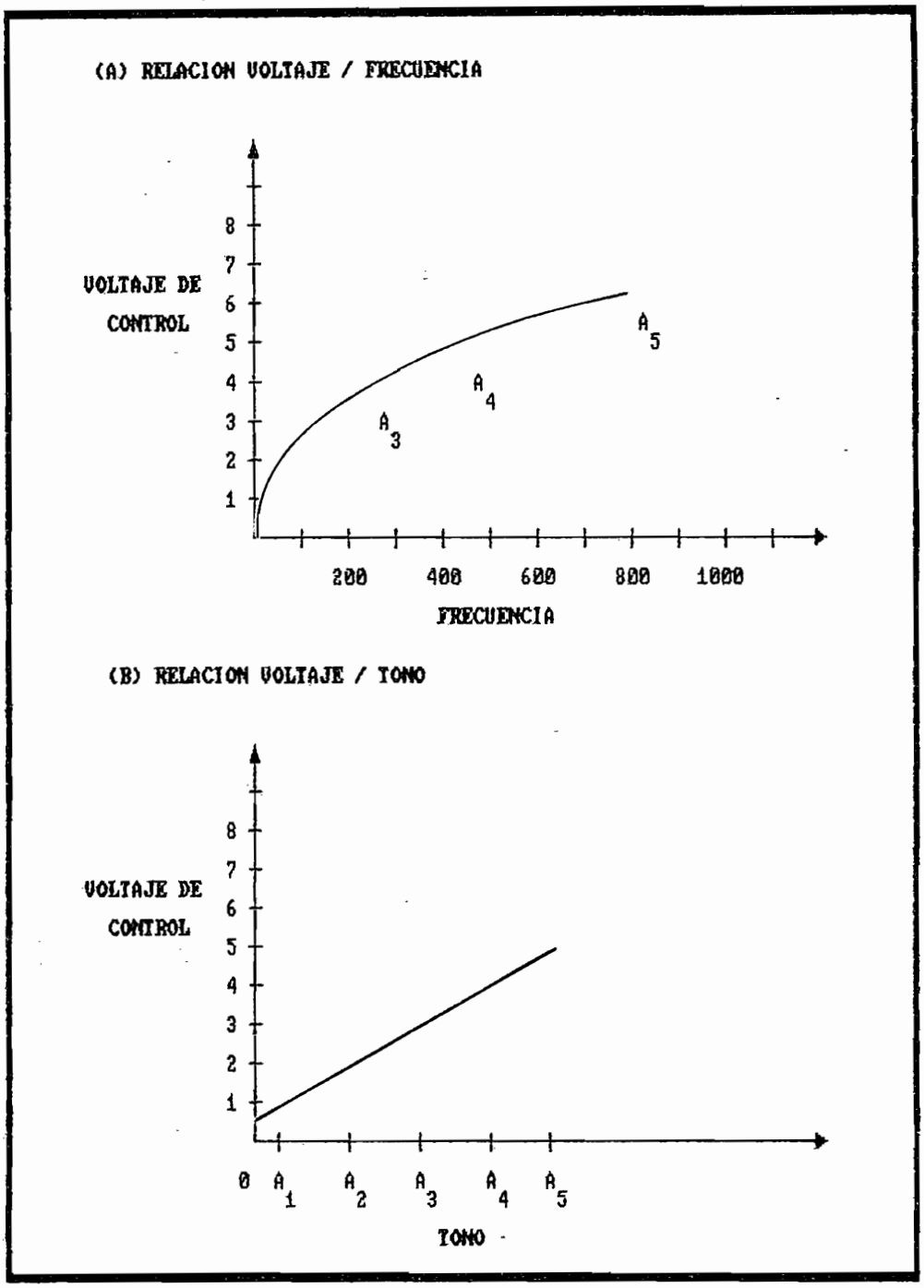
La segunda posibilidad, consiste en usar un VCO exponencial. En éste tipo, la salida de tonos es directamente proporcional a el voltaje de control aplicada. Por ejemplo este VCO puede tener las siguientes relaciones voltaje frecuencia:

Entrada Voltaje de control			Salida Frecuencia del VCO	
	1 voltio	=	110 Hz (la_2)	
PROGRESION	2 voltios	=	220 Hz (la_3)	PROGRESION
DE TENSION	3 voltios	=	440 Hz (la_4)	TONAL
LINEAL	4 voltios	=	880 Hz (la_5)	LINEAL

etc...

En este sistema variaciones lineales de voltaje producen una progresión de octavas.

En la figura 2.13 (a) se indica la gráfica voltaje - frecuencia en el VCO exponencial. En la figura 2.13 (b) se presenta la gráfica voltaje - tono del mismo VCO. De aqùi se puede deducir que el nombre aplicado a cada uno de estos VCO's depende simplemente del punto de vista. El VCO exponencial produce una relación lineal voltaje - tono, y el VCO lineal produce una relación



RELACIONES EN UN VCO EXPONENCIAL

FIGURA 2.13

exponencial voltaje - tono. Estos nombres han sido dados desde el punto de vista técnico, ya que en ingeniería se habla mas bien en términos de frecuencias que en términos de tonos musicales.

Debido a que para aplicaciones musicales es preferible tener linealizada la relación voltaje de control - tono que la relación voltaje de control - frecuencia, se elige como alternativa más lógica a un VCO exponencial. Cualquier fuente lineal puede ser utilizada para producir escalas musicales y las fuentes lineales de voltaje son más fáciles de acomodar a otros parámetros del sistema. Además operaciones como transposición y afinación, son fáciles de realizarse, así para transponer un VCO exponencial sólo se necesita aportar un voltaje fijo al nivel propio para que se sume el voltaje de control de tono.

Como el VCO exponencial utiliza entradas exponenciales para obtener variaciones exponenciales de frecuencia, éste precisa un convertidor lineal - exponencial. Este generador exponencial, así denominado, es un elemento integrante de todos los VCO exponenciales. Electrónicamente hablando, los generadores exponenciales no son difíciles de diseñar. La mayor dificultad estriba en que son muy sensibles a la temperatura. En los primeros VCO exponenciales, esta dependencia de las variaciones de temperatura significaba que los VCO tenían que ser afinados

continuamente. Actualmente esta dependencia ha sido casi eliminada, ya que los modernos VCO tienen compensaciones de temperatura, por lo que éstos son muy estables a variaciones térmicas.

El patrón voltaje de control / tono más común es: 1 Voltio / octava, tal como se ha presentado en los ejemplos anteriores. Esto significa que si la entrada de voltaje de control varía en 1 voltio, el tono del VCO cambiará en una octava. Para obtener la variación de un semitono se precisa una variación de 1/12 de voltio en el voltaje de control, ya que existen 12 semitonos en una octava. Estos valores se adoptaron a partir de los valores de polarización de los elementos electrónicos, además de que es muy fácil obtener variaciones de voltaje que se mantengan perfectamente lineales entre 0 y unos 12 voltios.

Con un patrón de 1 Voltio / octava y utilizando + 10 voltios como límite superior, se obtendrá una gama de 10 octavas que cubre todas las frecuencias audibles, si requerimos una precisión de +/- 10 milivoltios, esto dará una precisión de 0.01 % para los tonos.

En la figura 2.14 se presenta el diagrama de bloques de un VCO exponencial. En la parte inferior del diagrama hay tres entradas controladas por voltaje (CV). La tecla CV (del teclado), tiene el interruptor paro / marcha, de forma que la salida del VCO puede independizarse

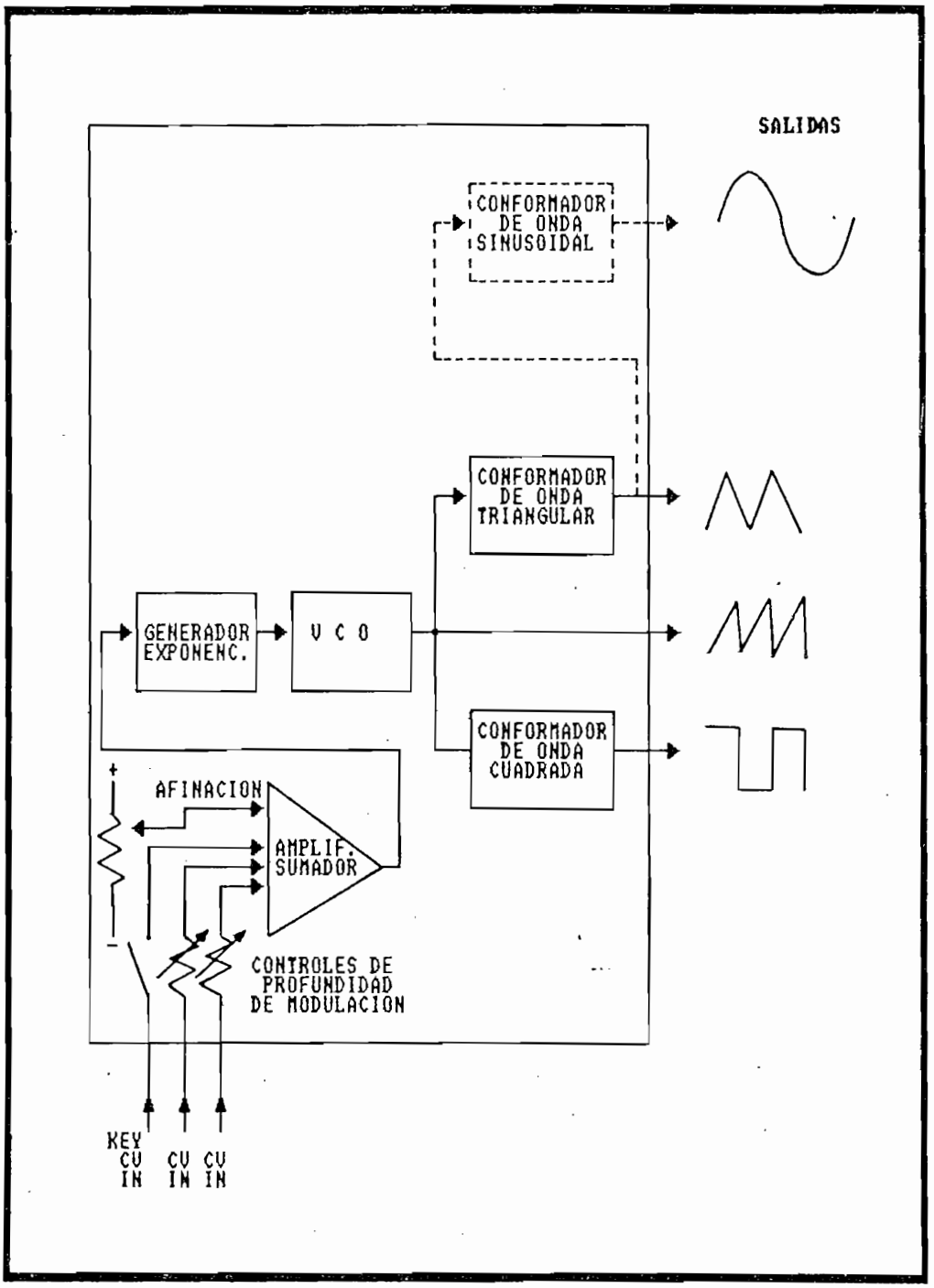


DIAGRAMA DE BLOQUES DE UN VCO EXPONENCIAL

FIGURA 2.14

del teclado. Se observa además dos entradas de voltaje de control, las mismas que pueden atenuarse con los respectivos potenciómetros. El control de afinación, es una fuente variable de voltaje dentro del VCO. Estas señales son sumadas para así generar el nivel adecuado que produzca el tono deseado en el VCO. Por ejemplo, si las cuatro señales de entradas de voltaje son respectivamente: + 1.5 V, -0.8 V, + 1.32 V y 0.0 V, la suma sería 2.0 V. Así, este voltaje podría producir el tono de Do central, el tono real dependerá del diseño del VCO. A continuación de este sumador está el convertidor exponencial que transforma las variaciones lineales de el voltaje de entrada en variaciones exponenciales que controlan el oscilador propiamente dicho. Un tipo común de oscilador genera una onda de diente de sierra, si se desea otros tipos de ondas se deben incluir los circuitos conformadores de onda, tal como se indica en la figura 2.14.

Generalmente se suele incluir un elemento conocido como LFO (Low Frecuency Oscillator), el cual es usado para generar frecuencias justo por encima del límite de audibilidad (25, 30 Hz), hasta muy por debajo del límite de audibilidad. Un LFO típico puede alcanzar 0.1 Hz. Así como los VCO's, los LFO's a menudo producen varios tipos de ondas siendo la más frecuente la sinusoidal. Si se aplica una onda sinusoidal de baja frecuencia a la entrada de voltaje de control del VCO, el resultado será una oscilación de la frecuencia del VCO a la frecuencia del

LFO. Este proceso se usa para añadir varios efectos especiales, tales como el "vibrato" a la salida del VCO.

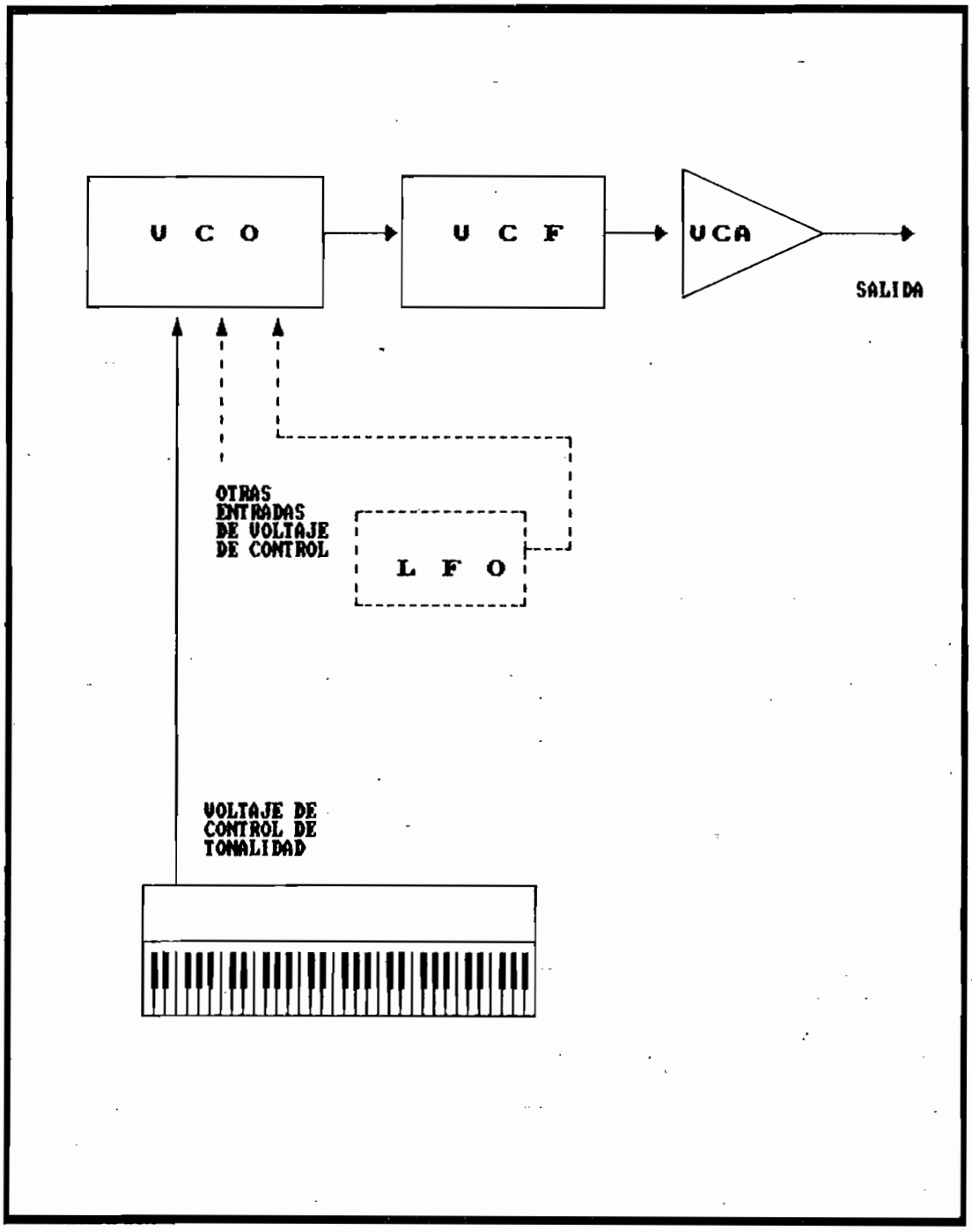
En la figura 2.15 se presenta la interconexión básica de un sintetizador, con la adición de entradas de modulación LFO.

2.2.2.2 Síntesis de Intensidad

Este parámetro del sonido es muy importante, ya que el timbre del mismo es afectado no sólo por la intensidad total de cada armónico, sino por la relación de intensidades de cada armónico respecto a los demás. Esto puede demostrarse analizando los espectros de la onda cuadrada y la onda triangular. El espectro de la onda triangular tiene exactamente los mismos armónicos que la onda cuadrada, pero en el caso de la onda triangular la intensidad de los armónicos existentes es tan baja que ésta tiene un sonido parecido al de la onda sinusoidal, muy diferente de la onda cuadrada.

El principal elemento usado en síntesis de sonido para controlar la intensidad del mismo es el VCA (Voltaje Controlled Amplifier), de tal manera que el nivel de salida de este amplificador está controlado por el voltaje de control aplicado externamente.

Como se ha visto, uno de los factores que



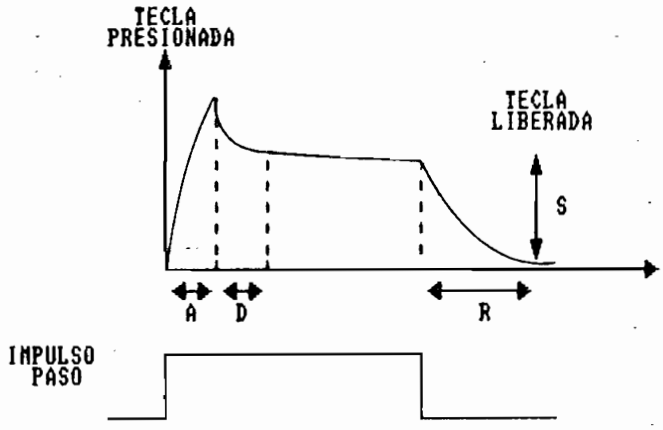
INTERCONEXION BASICA DEL VCO Y LFO EN EL SINTETIZADOR

FIGURA 2.15

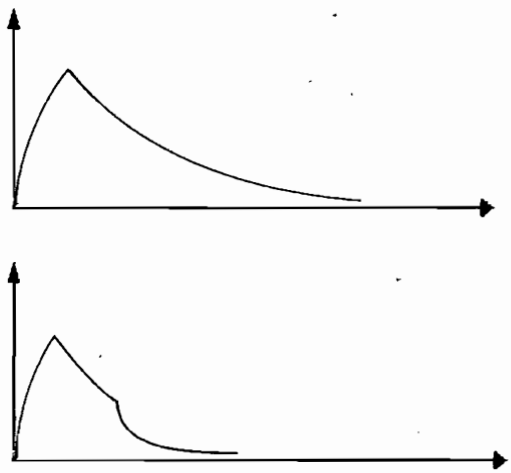
más inciden en el carácter de un sonido es su envolvente. En la síntesis de sonido, para generar adecuadamente las diferentes envolventes, a los sonidos se usa un generador de envolventes, el cual sirve para realizar el control por voltaje del VCA. Este generador de envolventes recibe el nombre de ADSR.

En la figura 2.16 (a) se presenta una envolvente de sintetizador, en la misma que se indica sus cuatro elementos: ataque (A), atenuación (D), sostenido (S) y amortiguación (R). El instante de pulsar la tecla, una señal paso permite que se desencadene el inicio del tiempo de ataque de forma que el nivel de salida del VCA empieza a crecer. Cuando el sonido alcanza su nivel máximo, al final del tiempo de ataque, se inicia la atenuación y el nivel desciende hasta el nivel prefijado de sostenido, manteniéndose así hasta que se libere la tecla, entonces se suspende la señal paso y se desencadena el tiempo de amortiguación, tiempo durante el cual el sonido se extingue definitivamente. En las figuras 2.16 (b), se presentan dos envolventes de piano sintetizadas. Obsérvese la diferencia entre éstas y las presentadas en la figura 2.7. Experimentalmente se ha demostrado que las curvas de ataque, atenuación y amortiguación exponenciales suenan naturales, y que añadir las pequeñas fluctuaciones inherentes al instrumento natural no añade nada útil al sonido tal como éste es percibido por el oído.

(A) CURVA ENVOLVENTE DE SINTETIZADOR



(B) ENVOLVENTES DE PIANO OBTENIDAS CON SINTETIZADOR



CURVAS ENVOLVENTES OBTENIDAS CON SINTETIZADOR

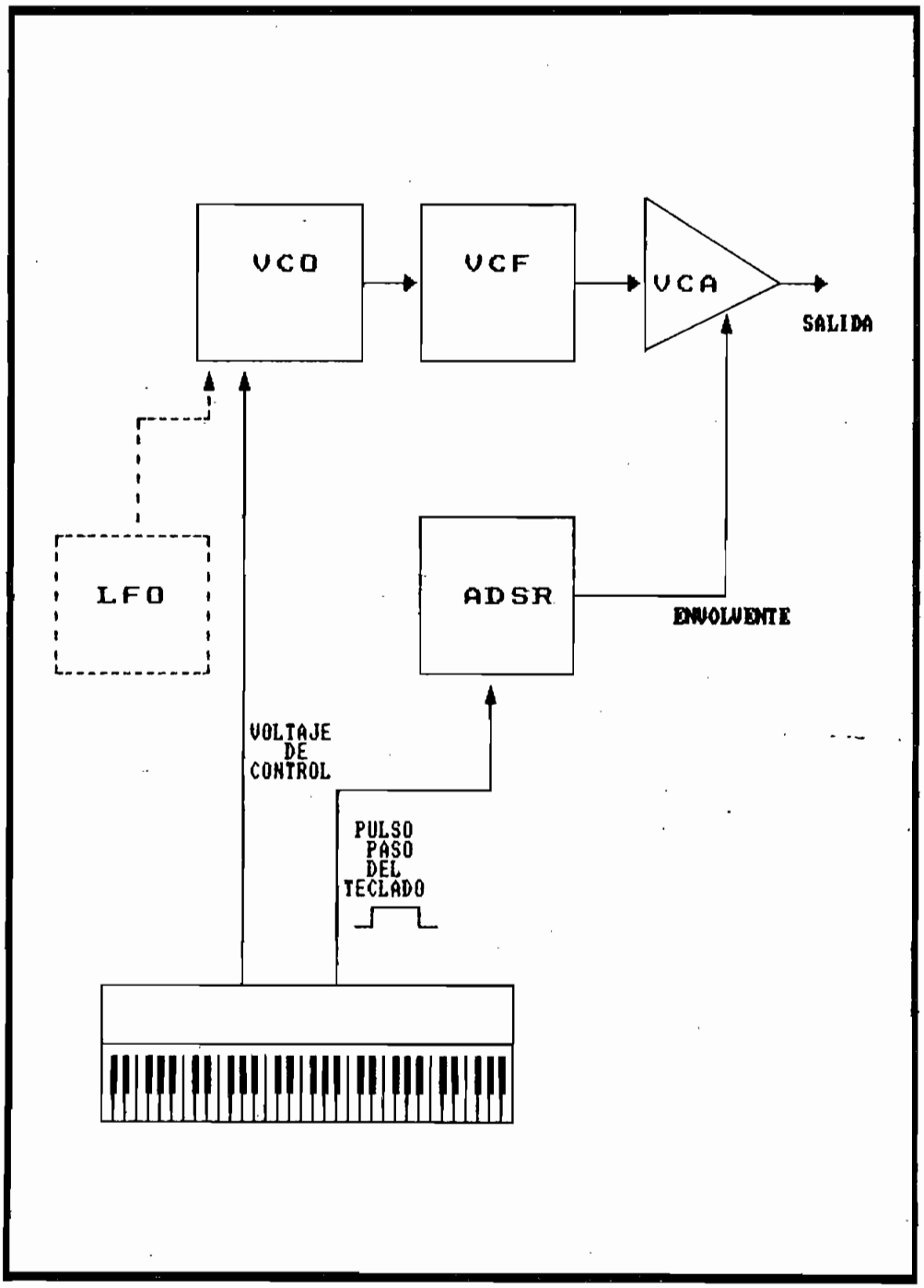
FIGURA 2.16

Una consideración muy importante sobre el VCA es su respuesta en relación con el voltaje de control de entrada. En la mayoría de VCA la respuesta es lineal. Por ejemplo si la entrada de control es de 2 V, la salida será también de 2 V, en la salida se tiene una reproducción exacta de la onda de entrada.

Algunos sintetizadores dan la opción de seleccionar entre respuesta lineal y exponencial. En el caso de un VCO lineal, la salida es modulada por la señal de control entregada por el generador de envolventes ADSR. En el caso de un VCO exponencial, la salida se atenúa muy rápidamente, esto es muy útil si se desea sintetizar instrumentos percusivos.

La mayoría de VCA proporcionan un control de ganancia inicial (llamado a veces control de bloqueo), el que sirve para establecer las condiciones iniciales del mismo. La acción del LFO sobre el VCA, puede ser usada para generar efectos como el "trémolo", el mismo que es la oscilación de la amplitud de una onda de sonido modulada por otra forma de onda.

En la figura 2.17 se presenta la interconexión básica añadiendo el VCA y su modulación por medio del ADSR.



INTERCONEXION BASICA DEL VCA EN EL SINTETIZADOR

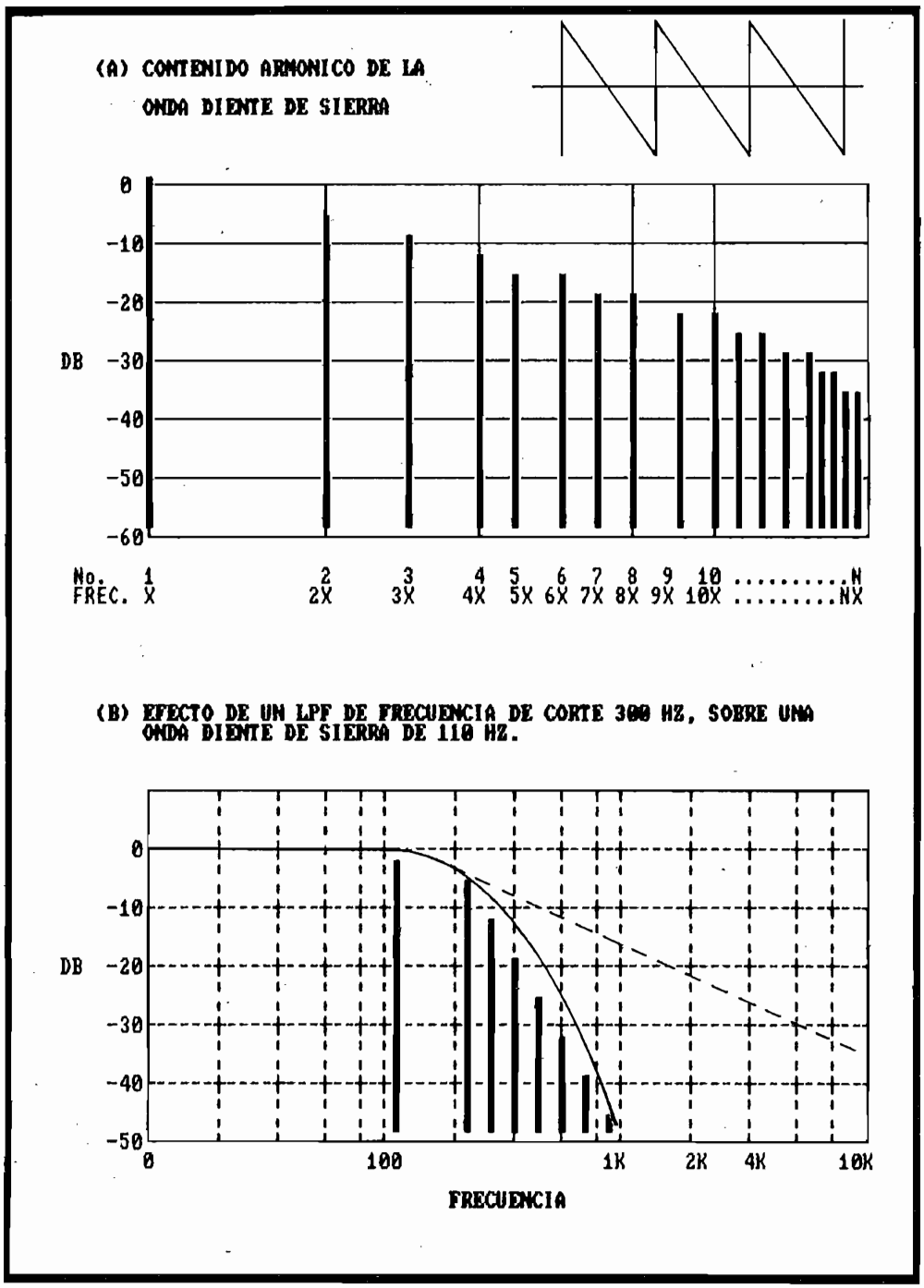
FIGURA 2_17

2.2.2.3 Síntesis por eliminación del Timbre

La principal fuente de tonos en un sintetizador por eliminación es el VCO, si se trata de sonidos con tono; y, el generador de ruido, si se trata de sonidos sin tono. La principal fuente de control para añadir timbre a los tonos, son los filtros y entre éstos los más comunes son los VCF (Voltaje Controlled Filter).

Existen dos tipos fundamentales de filtros que son usados en los sintetizadores por eliminación: Filtro pasa bajos (LPF), filtro pasa alto (HPF). El tipo más común de VCF es el filtro pasa bajo controlado por voltaje. Este tipo es tan común, que cuando se sintetiza el timbre por medio de un LFO, es casi seguro que se trata de un filtro pasa bajo. Sistemas más sofisticados incluyen también filtros pasa alto, pasa banda y filtros de eliminación de banda controlados por señales de voltaje.

Es conocido el efecto de un filtro pasa bajos sobre una señal fundamental, pero en síntesis de sonido nos interesa el efecto de este filtro sobre una señal rica en armónicos. En la figura 2.18 (a) se presenta una onda diente de sierra de frecuencia correspondiente a la nota la_3, que es aplicada a un filtro pasa bajo de frecuencia de corte de 330 Hz. Los resultados se pueden obtener superponiendo el espectro de la onda diente de sierra y la curva de respuesta de frecuencia tal como se



EFECTO DE UN LPF SOBRE LA ONDA DIENTE DE SIERRA

FIGURA 2.18

indica en la figura 2.18 (b). Como comparación, las líneas a trazos muestran los niveles que los armónicos alcanzarían sin filtraje. Como se puede ver el primer armónico no resulta afectado, pero todos los demás son más o menos atenuados. Se observa que el nivel normal del cuarto armónico es de -12 db con relación al fundamental. De acuerdo a la curva de respuesta del filtro, el nivel de salida de una onda sinusoidal de 440 Hz (frecuencia del cuarto armónico) sería de -6 db y su nivel será de - 18 db. Los niveles de los otros armónicos se ven afectados en forma similar. Como los niveles de los armónicos en la onda resultante son diferentes a los originales, el timbre y la forma de onda, es decir su sonido, también variará.

De la figura 2.18 (b) se puede observar que en tanto se mantiene fija la frecuencia de corte del filtro, el contenido armónico del sonido de salida sólo se mantendrá fijo si el tono de la onda diente de sierra se mantiene fijo. Cambiando el tono de la onda diente de sierra desplazará el espectro hacia la derecha o a la izquierda en relación con la curva de respuesta del filtro y por lo tanto cambia el contenido armónico del sonido de salida. Es decir si se toca una escala cromática se producirán notas teniendo cada una de ellas un timbre distinto, lo que obviamente es inadecuado en un instrumento musical. Si tocamos la nota la_5 (880 Hz), el nivel del fundamental habrá bajado más de 15 db. Así, a mayor frecuencia se reducirá considerablemente el nivel del

sonido de salida. El control de la frecuencia de corte del VCF fija el punto de corte inicial que a continuación puede ser alterado con voltajes de control aplicados externamente. Si usamos la salida de voltaje del teclado para controlar la frecuencia de corte del VCF, es fácil conseguir que la frecuencia de corte del filtro siga al tono. En el ejemplo dado, la frecuencia de corte del filtro se situó sobre la frecuencia del tercer armónico de la onda diente de sierra, 110 Hz (la_2). Aplicando el voltaje de control de nota al VCF, es posible que la frecuencia de corte se desplace con el tono y por lo tanto puede permanecer en el punto aproximado que representa el tercer armónico de cualquier tono que se pulse. De esto lógicamente se deduce que sería esencial que el VCF tenga la misma relación voltaje - frecuencia de corte que el VCO. Si una variación de 1 V, hace variar el tono del VCO en una octava, también debe variar la frecuencia de corte del VCF en una octava. Con esta relación el VCF puede seguir exactamente cualquier variación de tono.

El timbre de la mayoría de instrumentos varía con el tono. Generalmente los tonos agudos son más brillantes que los tonos graves. Como los armónicos agudos de alto nivel producen un timbre brillante, esto parece implicar que si se toca la escala cromática ascendente en el teclado, en lugar de seguir el tono de la música, la frecuencia de corte del VCF realmente tendría que

anticiparse al tono adelantándose más de lo normal, de forma que permita que pasen más armónicos. En lugar de 1V - octava es posible que una variación de dos octavas en la frecuencia de corte por 1 Voltio de variación en el voltaje de control sería el adecuado. En la práctica, debido a las peculiaridades de nuestra audición esto no es necesario. Experimentalmente se observa que si se atenúa el voltaje de control de tono aproximadamente en la mitad, el timbre del sonido de salida parecerá a nuestros oídos que tiene un contenido armónico constante. Si no se atenúa el voltaje de control de tono, el timbre parecerá que se hace más brillante a medida que los tonos sean más agudos. Por lo tanto sigue válida una relación de 1 voltio - 1 octava.

El timbre de muchos instrumentos, particularmente los instrumentos de viento y las cuerdas pulsadas, varían durante la emisión de cada nota. Esto puede ser imitado usando un generador de envolvente para hacer que varíe la frecuencia de corte del VCF. A menudo la misma envolvente que se utiliza para controlar el VCA, también puede ser utilizada para controlar el VCF. En otros casos, es ventajoso utilizar un segundo generador de envolvente de forma que el barrido de los armónicos sea independiente de la envolvente de intensidad. Algunos sintetizadores permiten la inversión de la envolvente de forma que los armónicos pueden barrer en el sentido contrario de lo que normalmente cabe esperar.

En la figura 2.19 se presenta la interconexión básica de todo el sintetizador por eliminación controlado por voltaje.

2.2.3 Síntesis Frecuencia Modulada

Este método de síntesis de sonido, es uno de los más populares en la actualidad. Fue introducido inicialmente por YAMAHA en 1983, y actualmente esta compañía tiene los derechos exclusivos en esta tecnología.

En términos simples, la síntesis de Frecuencia Modulada FM, tiene como fundamento la modulación de ondas sinusoidales para generar así nuevas ondas.

La síntesis FM trabaja de manera similar a la síntesis aditiva, pero sus resultados finales son diferentes. En esta técnica de síntesis, también se añaden ondas sinusoidales simples, para crear una forma de onda más compleja, pero en lugar de sumar una a otra, la síntesis FM modula una onda base con las ondas sinusoidales adicionales sobre ésta.

Cada onda sinusoidal, en esta configuración, es un bloque del diagrama de bloques y recibe el nombre de "operador". El operador base recibe el nombre de portador", y su función es modelar el armónico fundamental de la onda de sonido. Los otros operadores reciben el

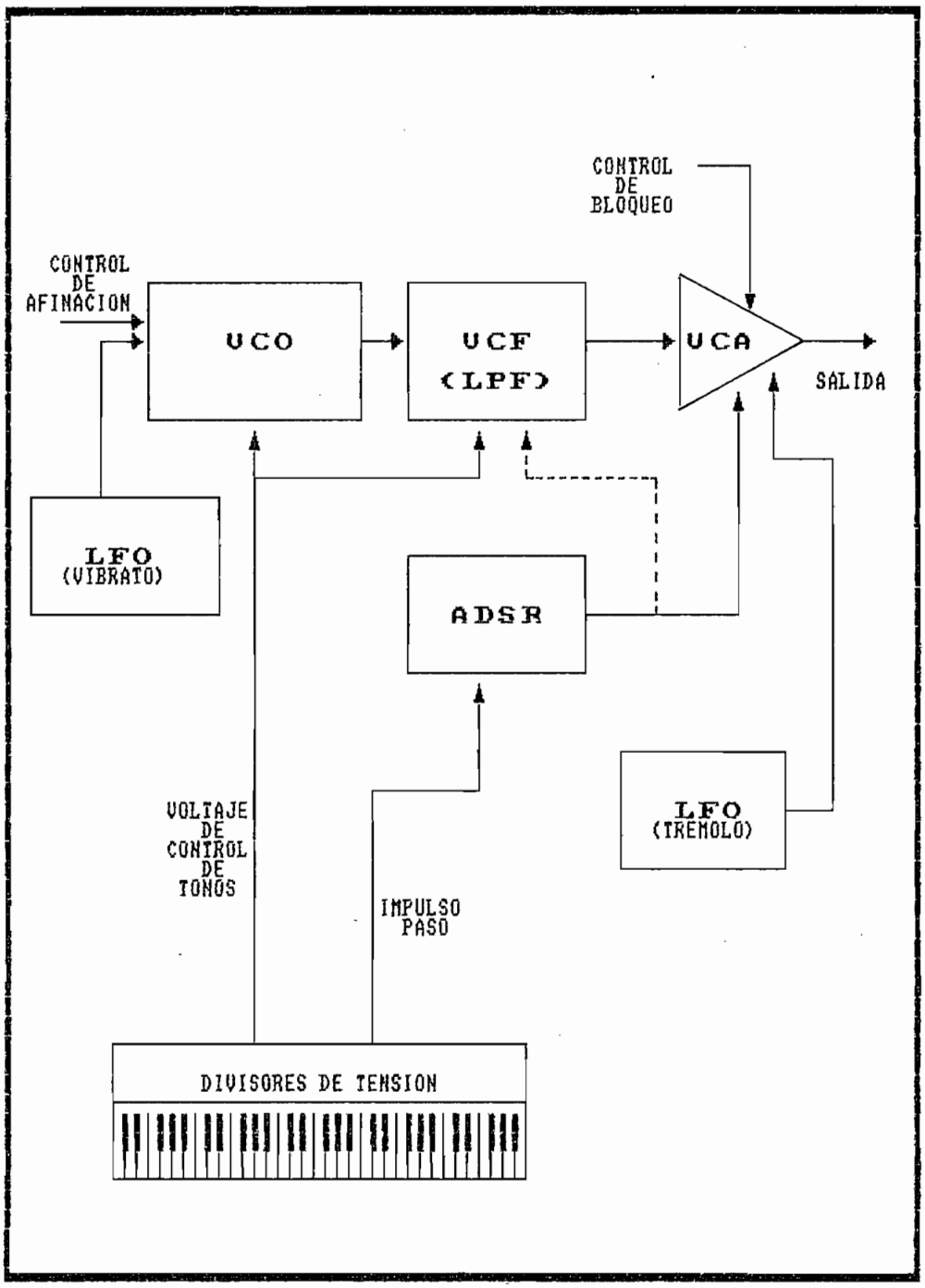


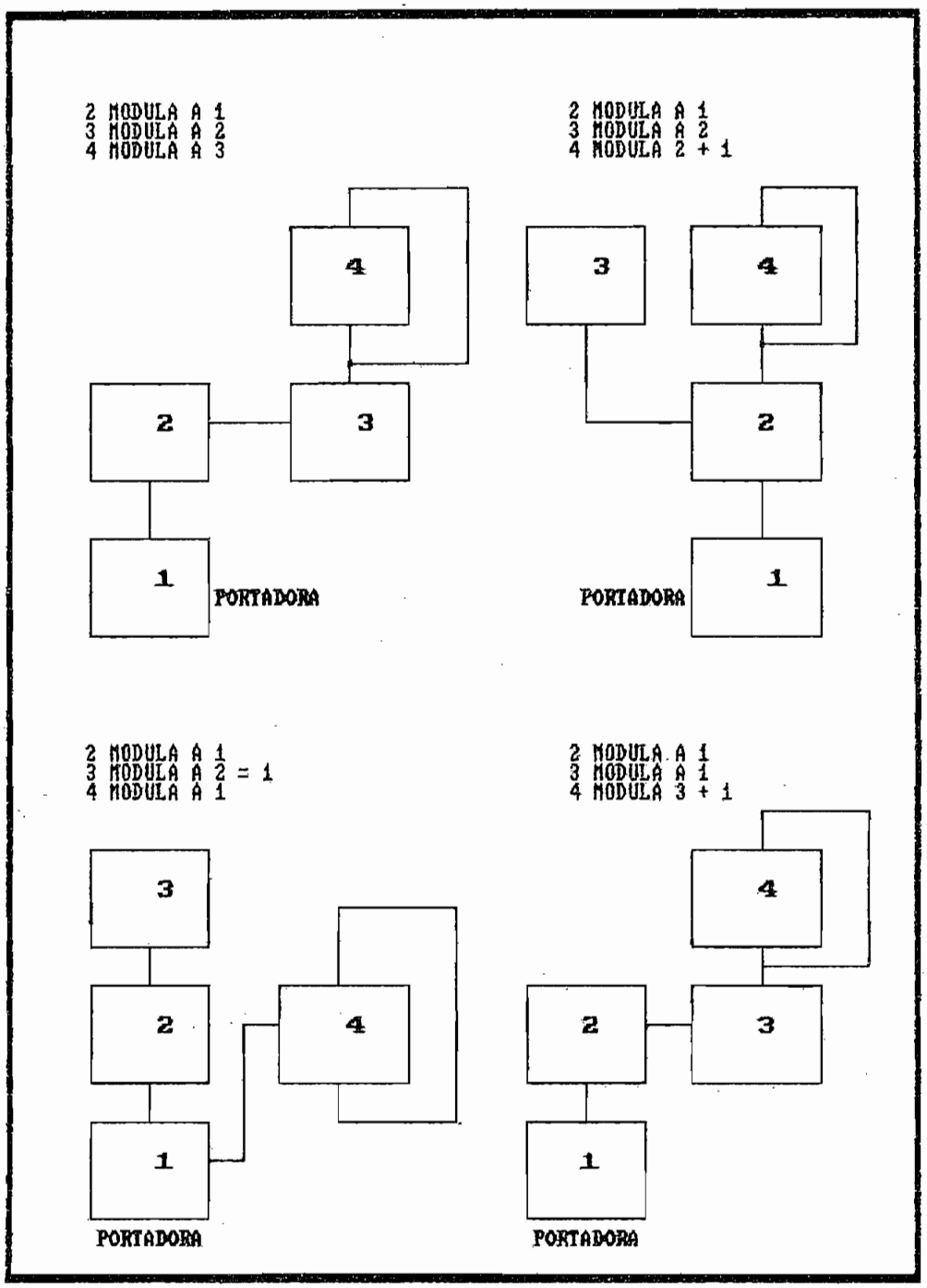
DIAGRAMA DE BLOQUES DE LA INTERCONEXION BASICA DEL SINTETIZADOR POR ELIMINACION

FIGURA 2.19

nombre de "moduladores", los cuales sirven para añadir las características de timbre al sonido. Existen muchos arreglos, en los cuales los moduladores pueden actuar sobre la señal portadora, cada arreglo posible recibe el nombre de "algoritmo". Con este número de algoritmos se puede crear un número casi ilimitado de sonidos.

Debido a que cada operador, añade más complejidad al sonido final, es importante el número de operadores con los cuales se desee realizar el proceso de síntesis. La técnica desarrollada por YAMAHA, parte de sistemas de 4 y 6 operadores.

Los sistemas de 4 operadores fueron inicialmente desarrollados para su uso en sintetizadores de uso popular, en tanto que los sistemas de 6 operadores fueron diseñados para su uso en sintetizadores profesionales. Los sistemas de 4 operadores permiten sintetizar sonidos basándose en sus algoritmos, 4 de los cuales se presentan en la figura 2.20, cada uno de los mismos permite sintetizar sonidos de características completamente diferentes. En los sistemas de 6 operadores, el número de sonidos posibles es superior al de cualquier otro modelo de sintetizador, siendo este de hasta 15000 sonidos.



ALGUNOS ALGORÍTMOS DE CUATRO OPERADORES DEL SINTETIZADOR FM TX81Z DE YAMAHA

FIGURA 2.20

2.3 METODO DE SINTESIS DIGITAL

Este método de síntesis es uno de los más recientes. Combina la técnica de síntesis por eliminación con la técnica de muestreo digital. El resultado de este proceso permite sintetizar sonidos de muy alta calidad. Por su similitud con los sistemas de síntesis por eliminación este sistema de síntesis es fácil de ser usado.

Los sintetizadores de este tipo usan una técnica de muestreo digital, partiendo de sonidos de instrumentos reales, éstos, una vez digitalizados se entregan al usuario como una librería de sonidos fundamentales (Patch library). A partir de estas ondas el usuario puede usar el método de síntesis por eliminación, pero a diferencia de éste, no se parte de ondas cuadradas o dientes de sierra, sino de cualquier onda de la librería de sonidos. Este proceso simplifica significativamente el proceso de síntesis, a la vez que los sonidos obtenidos tienen características muchas veces superiores a los sonidos generados con cualquier instrumento natural.

La técnica de muestreo de los sonidos naturales usa la modulación por impulsos codificados (PCM), la cual fue desarrollada ya en 1937 por Alec Reeves de Laboratrorios ITT y aplicada por primera vez a la síntesis de sonido por ROLAND que patentó este método de síntesis.

2.4 DESCRIPCION DE EQUIPOS DE SINTESIS DIGITAL EXISTENTES

Actualmente, la variedad de equipo para síntesis de sonido existente es extremadamente amplia, partiendo de sistemas pequeños hasta llegar a equipos de síntesis profesionales basados en hardware y software.

- Sintetizadores digitales.- Los sintetizadores digitales son actualmente una de las más comunes herramientas disponibles para realizar tareas de síntesis de sonido. Algunos son simplemente el equivalente digital de los sintetizadores analógicos anteriores, aunque la mayoría son programables permitiendo almacenar en su memoria los sonidos sintetizados por el usuario. Debido a su naturaleza digital, últimamente se han desarrollado sintetizadores sin teclado, los cuales pueden actuar como maestros para controlar así una serie de sintetizadores esclavos. En 1988, aparecieron los sintetizadores multiumbral, los cuales generan simultáneamente hasta 16 sonidos, permitiendo así reemplazar un grupo de sintetizadores por uno sólo. (4). También se han desarrollado sintetizadores que pueden ser controlados por medio de cuerdas, viento y percusión.

- Muestreadores.- Estos elementos permiten registrar digitalmente sonidos acústicos. Existen muestreadores con frecuencias de muestreo de hasta 100 KHz. El requerimiento de memoria de la información muestreada es muy alto.

C A P I T U L O III

PROTOCOLO DE COMUNICACION MIDI

PROTOCOLO DE COMUNICACION MIDI

3.1 INTRODUCCION

3.1.1 Los primeros pasos

Los primeros intentos, tendientes a crear un sistema de comunicación digital entre sistemas de síntesis de sonido datan a inicios de la década de los 80. Es así, que en el año 1981, los fabricantes ROLAND, SEQUENTIAL CIRCUITS y OBERHEIM se reunieron para desarrollar una forma de comunicación común a todos sus productos de síntesis. La meta fue crear un interfaz basado en hardware y un lenguaje de computación que permitiera dicha comunicación. SEQUENCIAL PRODUCTS, propuso un estándar al que se llamó USI siglas de "Universal Standard Interface", el cual fue introducido en 1981 en la convención de la AES (Audio Engineers Society). (4)

A partir de este año, los principales fabricantes, convinieron en aceptar al USI e incluirlo en sus aparatos en lugar de seguir desarrollando sus propios proyectos para crear un interfaz individualizado. Para esta época varios fabricantes habían desarrollado sistemas de comunicación que permitían la conexión de sintetizadores de la misma marca pero no con marcas de otros fabricantes.

Originalmente, el estándar fue diseñado para

satisfacer dos metas específicas. La primera, era permitir que desde un teclado de sintetizador se pudiese controlar la salida tonal de otro. La segunda, fue tratar de sincronizar diferentes máquinas que funcionen con control de tiempo, tales como máquinas de ritmo y secuenciadores.

A inicios de 1983, tuvo lugar el primer gran encuentro de fabricantes de sintetizadores, esta vez cada fabricante aportó con su contribución al estándar inicial. Estos esfuerzos se combinaron, y dieron lugar a la primera versión del estándar universal. Así, se reemplazó al "Universal standard Interface" por el "Musical Instrument Digital Interface", MIDI. Para 1984, 16 fabricantes de todo el mundo habían adoptado este sistema de estandarización.

Una vez que se establecieron los lineamientos generales del nuevo estándar, los fabricantes empezaron a incluir este sistema sólo a su línea de sintetizadores profesionales. La primera demostración pública del sistema MIDI, tuvo lugar en 1985 en la convención de la NAMM (National Association of Music Merchants), en la cual se conectaron exitosamente dos sintetizadores de diferentes fabricantes, demostrándose que se podía controlar los sonidos de los dos sintetizadores desde el teclado de cualquiera. Esta simple conexión maestro - esclavo, es el arreglo fundamental de un sistema MIDI.

[illegible] actualmente se ha convertido en una herramienta de posibilidades ilimitadas.

3.1.2 Definiciones

Es necesario primeramente definir, que es el sistema MIDI. Para lograr este objetivo se plantea la respuesta desde dos puntos de vista.

Desde un punto de vista musical, el sistema MIDI permite la comunicación musical de dos sintetizadores, en donde uno de ellos indica al otro que tecla fue presionada, cuan fuerte fue presionada, que presión fue aplicada luego de la presión inicial, así como revelar los más pequeños efectos que pueden aplicarse al sonido durante la ejecución de un instrumento, de tal manera que enviando esta información digital desde el terminal MIDI OUT del un sintetizador al terminal MIDI IN del otro sintetizador, este puede reproducir con enorme precisión la ejecución del sintetizador maestro.

Desde un punto de vista más técnico, se puede definir al sistema MIDI simplemente como un protocolo de comunicación digital. El mismo que transmite serialmente mensajes de varios bytes, dependiendo de la información que se esté transmitiendo.

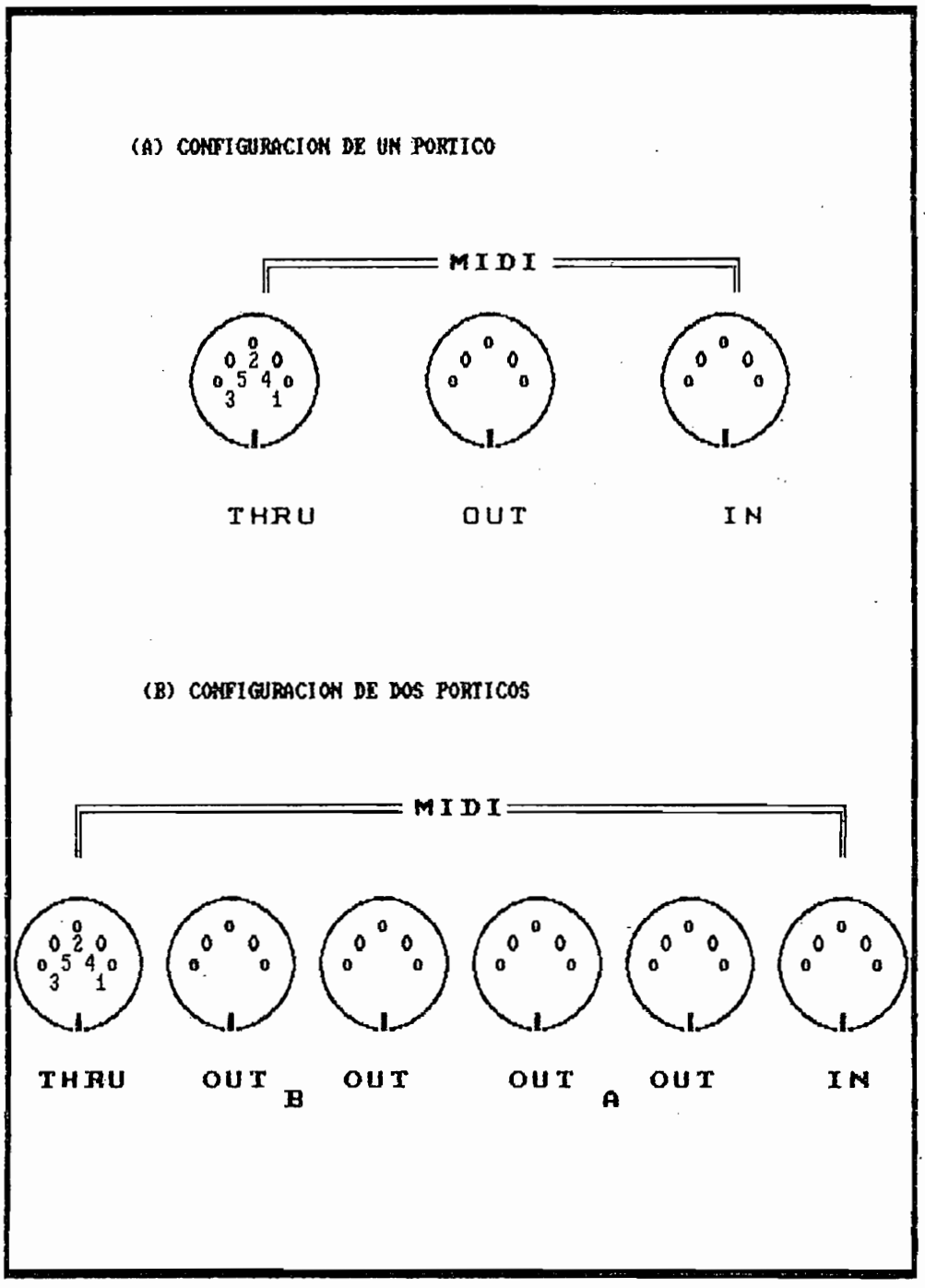
3.1.3 Terminales de comunicación MIDI

Los terminales de comunicación MIDI, pueden tener varios tipos de configuraciones, dependiendo del

sintetizador. La especificación MIDI, establece que cada conector debe ser del tipo DIN de 5 pines. En la figura 3.1 se presenta dos configuraciones comunes en sintetizadores actuales. Generalmente, se encuentra un pórtico de comunicación MIDI con tres terminales, etiquetados con los siguientes nombres:

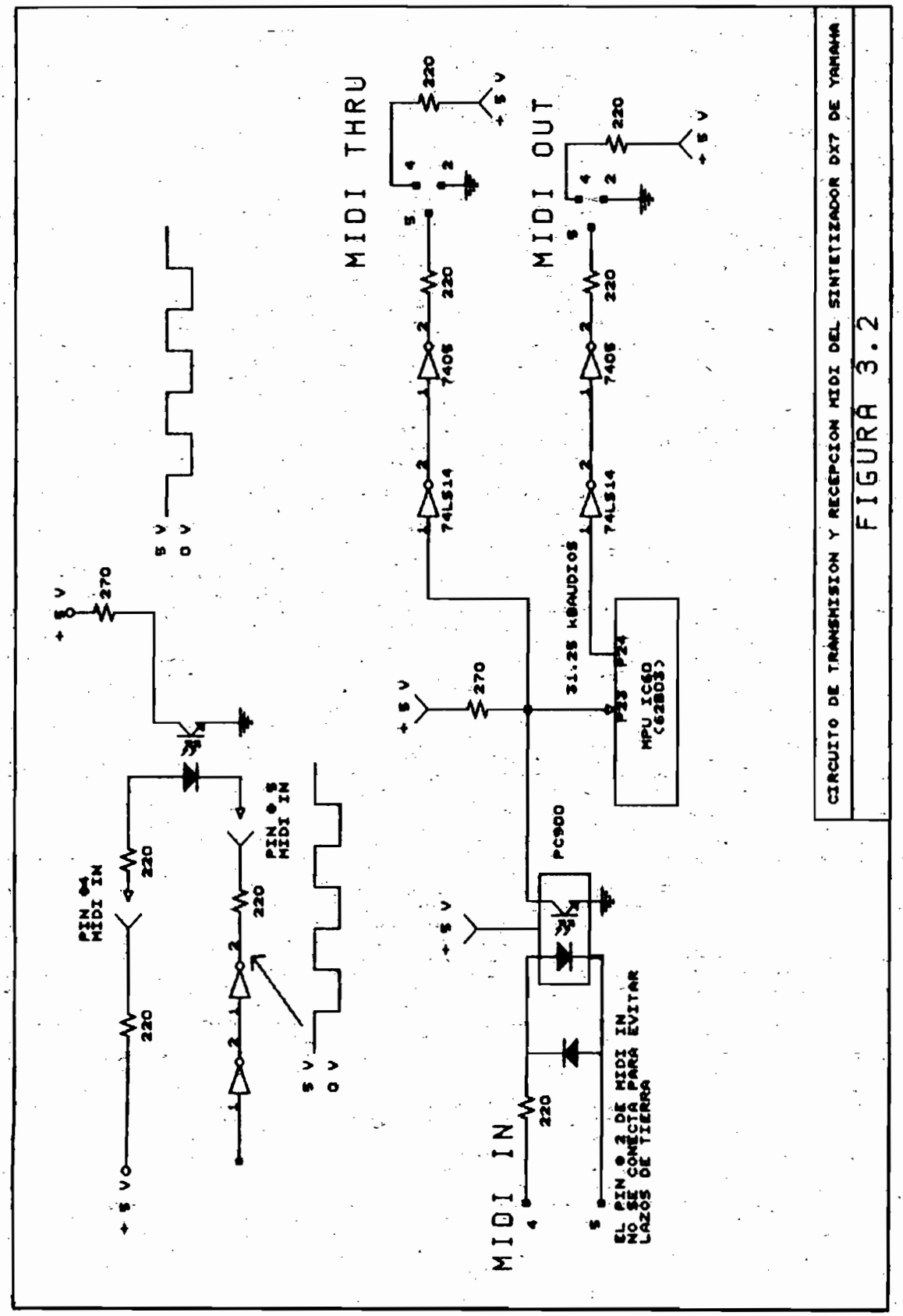
- MIDI OUT: terminal de salida de la información MIDI
- MIDI IN: terminal de entrada de información MIDI
- MIDI THRU: terminal de salida, por el que se transmite exactamente la misma información que fue recibida por el terminal MIDI IN.

Cualquier conexión entre estos terminales, debe hacerse con cables con conectores DIN, los que no deben tener una longitud mayor a 15 m. De los 5 pines de cada conector DIN, se usan solamente 3 pines. En la figura 3.2, se presenta el diagrama del circuito de transmisión y recepción de información MIDI del sintetizador DX7 de YAMAHA. En esta se observa que en la etapa de transmisión (MIDI OUT), se usan solamente los pines 2, 4 y 5, en tanto que en la etapa receptora (MIDI IN) se usan solamente los pines 4 y 5. En la misma figura se observa que la información recibida en MIDI IN es aplicada a un fotodiodo de tal manera que este induce una corriente de base en el fototransistor, creando así un fotoacoplamiento entre el lado transmisor y receptor, lo que aísla eléctricamente los sintetizadores.



TERMINALES DE COMUNICACION MIDI

FIGURA 3.1



CIRCUITO DE TRANSMISION Y RECEPCION MIDI DEL SINTETIZADOR DX7 DE YAMAHA

FIGURA 3.2

El circuito de transmisión - recepción indicado, usa señales digitales TTL, de tal manera que cuando en el pin # 5 del conector MIDI IN se aplica una señal de 0 V, el voltaje en el pin # 4, induce una corriente de 7.5 mA, por lo que el fototransistor se enciende y su colector se pone a tierra. En caso contrario, cuando en el terminal # 5 del conector DIN, existe un voltaje positivo, deja de fluir corriente en el fotodiodo, con lo que el fototransistor se apaga y el voltaje en colector se pone a 5 V. Esta señal es aplicada a un circuito de procesamiento MIDI (MPU IC60 - 63B03) y directamente a los terminales de salida MIDI THRU.

3.2 DESCRIPCION DEL PROTOCOLO MIDI

Como cualquier protocolo de comunicación el sistema MIDI, requiere el estudio de sus características en función de sus diferentes códigos y significados. En la presente tesis, se estudiará el protocolo MIDI, tal como esta estandarizado en la especificación MIDI - 0.1.

3.2.1 Características Generales

El protocolo de comunicación MIDI transmite y recibe mensajes serialmente en palabras de 8 bits. Cada byte transmitido lleva tanto un bit de inicio como un bit de parada. El bit de inicio corresponde a un cero lógico, en tanto que el bit de parada corresponde a 1 lógico.

El interfaz que permite obtener esta comunicación es el ASCI (Asynchronous Serial Communication Interface), el cual está incluido en la MPU (MIDI Processing Unit), que tiene todo aparato que transmita información MIDI. Este elemento permite codificar datos de 8 bits recibidos en forma paralela y transmitirlos en forma serial con los correspondientes bits de inicio y parada. En la figura 3.3 se observa, en diagrama de bloques, la configuración básica de una unidad de procesamiento MIDI. (4)

Una vez que una MPU, recibe los 10 bits correspondientes a un byte de información más el bit de inicio y parada, estos son decodificados, de tal manera que se procesan solamente los 8 bits útiles, por lo tanto en los posterior nos referiremos solamente a la información MIDI útil, es decir los 8 bits.

La MPU, trabaja con un circuito UART, (Universal Asynchronous Receiver Transmitter), el cual permite transmitir y recibir información digital serialmente. Estas dos unidades forman lo que se conoce como interfaz MIDI.

El protocolo establece además una velocidad de transmisión de información serial de 31.25 kbaudios, la misma que será a través de alambres blindados conectados entre sí por medio de terminales tipo DIN de 5 pines.

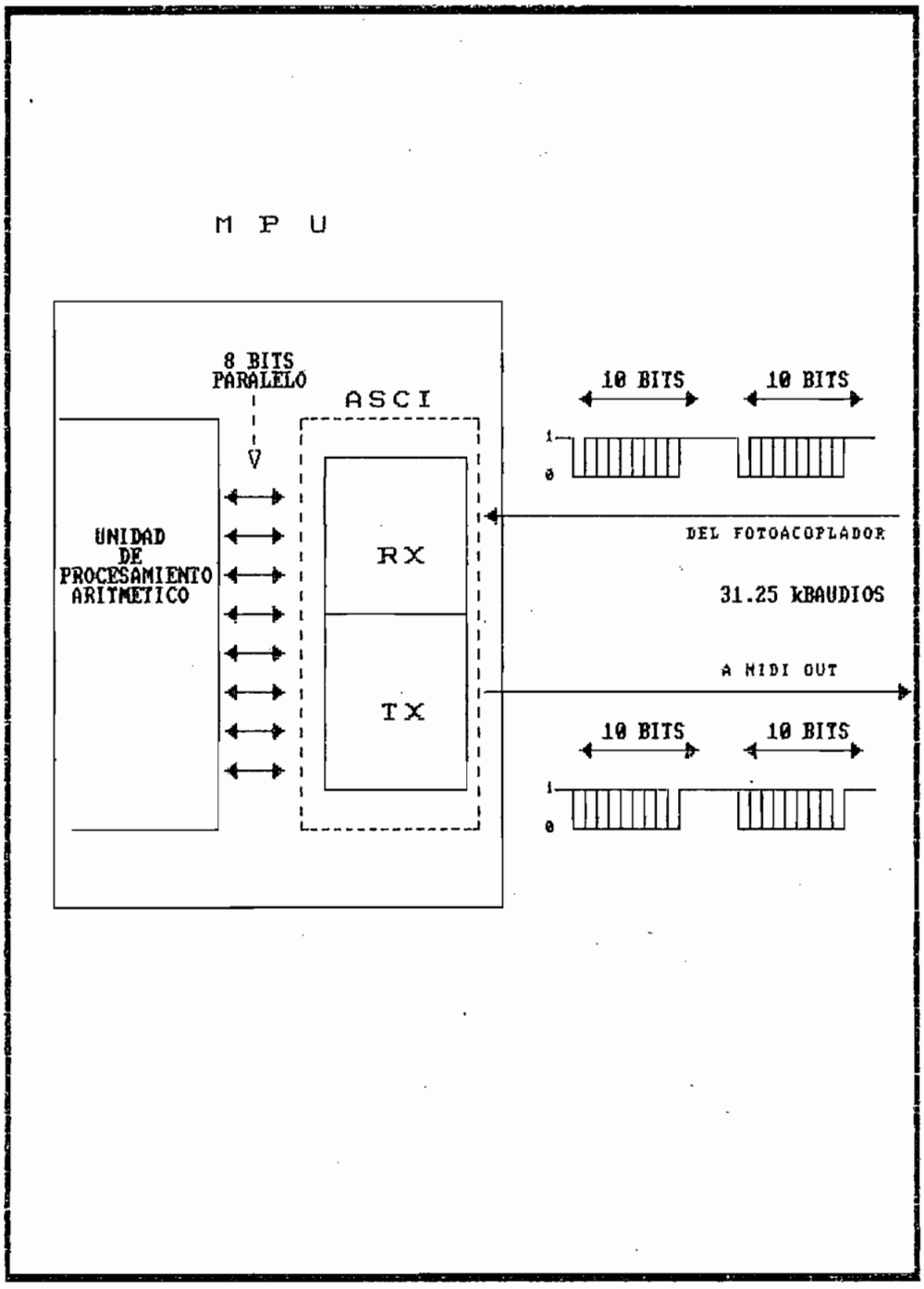


DIAGRAMA DE BLOQUES DE UNA UNIDAD DE PROCESAMIENTO MIDI (MPU)

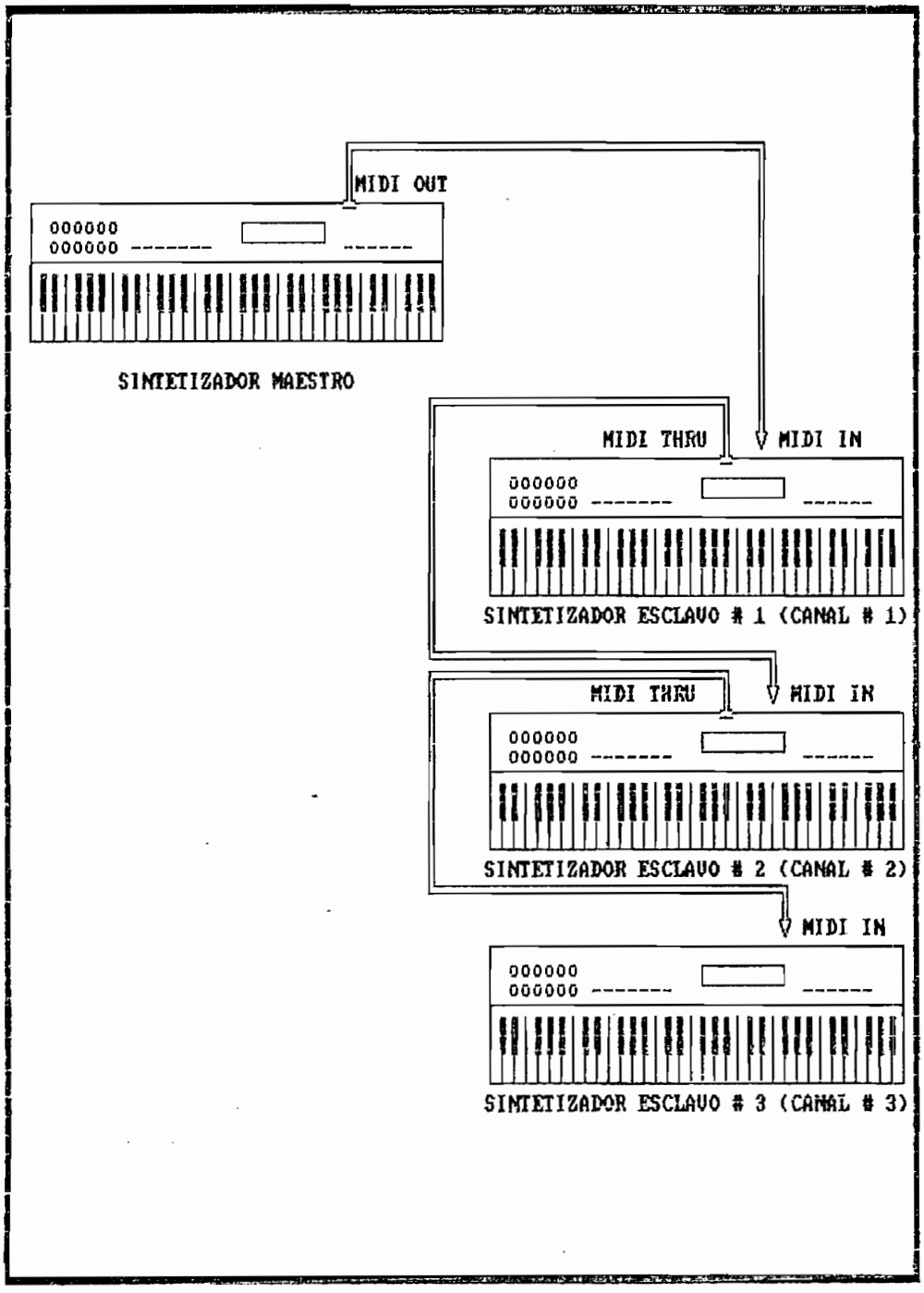
FIGURA 3.3

3.2.2 Canales MIDI

La información MIDI, es transmitida a través de 16 canales independientes, esto significa que es posible transmitir información correspondiente a 16 sintetizadores por el mismo cable. Por ejemplo si un sintetizador maestro está enviando mensajes MIDI por medio del canal 1, esta información puede ser recibida por un sintetizador esclavo solo si este se encuentra asignado al canal 1, en caso contrario no recibirá esta información. Supongamos ahora que el sintetizador maestro esta transmitiendo información por los canales 1, 2 y 3, es posible en este caso conectar tres sintetizadores, tal que cada uno de ellos se encuentre asignado a los canales 1, 2 y 3 respectivamente. El resultado en este caso será que cada sintetizador responderá a su propio canal, y se obtendrá la reproducción de toda la información que esté transmitiendo el sintetizador maestro. Esta conexión se la realiza tomando las señales de los terminales MIDI THRU, y se la conoce como conexión "daisy chain". En la figura 3.4 se presenta un diagrama que describe gráficamente esta situación.

3.2.3 Formato Básico de Transmisión de datos MIDI

En la figura 3.5 se presenta en diagrama de bloques el formato básico de transmisión de datos MIDI. En ésta se observa que cualquier mensaje MIDI, está formado por lo menos de 1 byte de información, pueden haber



SINTETIZADORES EN CONEXION DAISY CHAIN

FIGURA 3.4

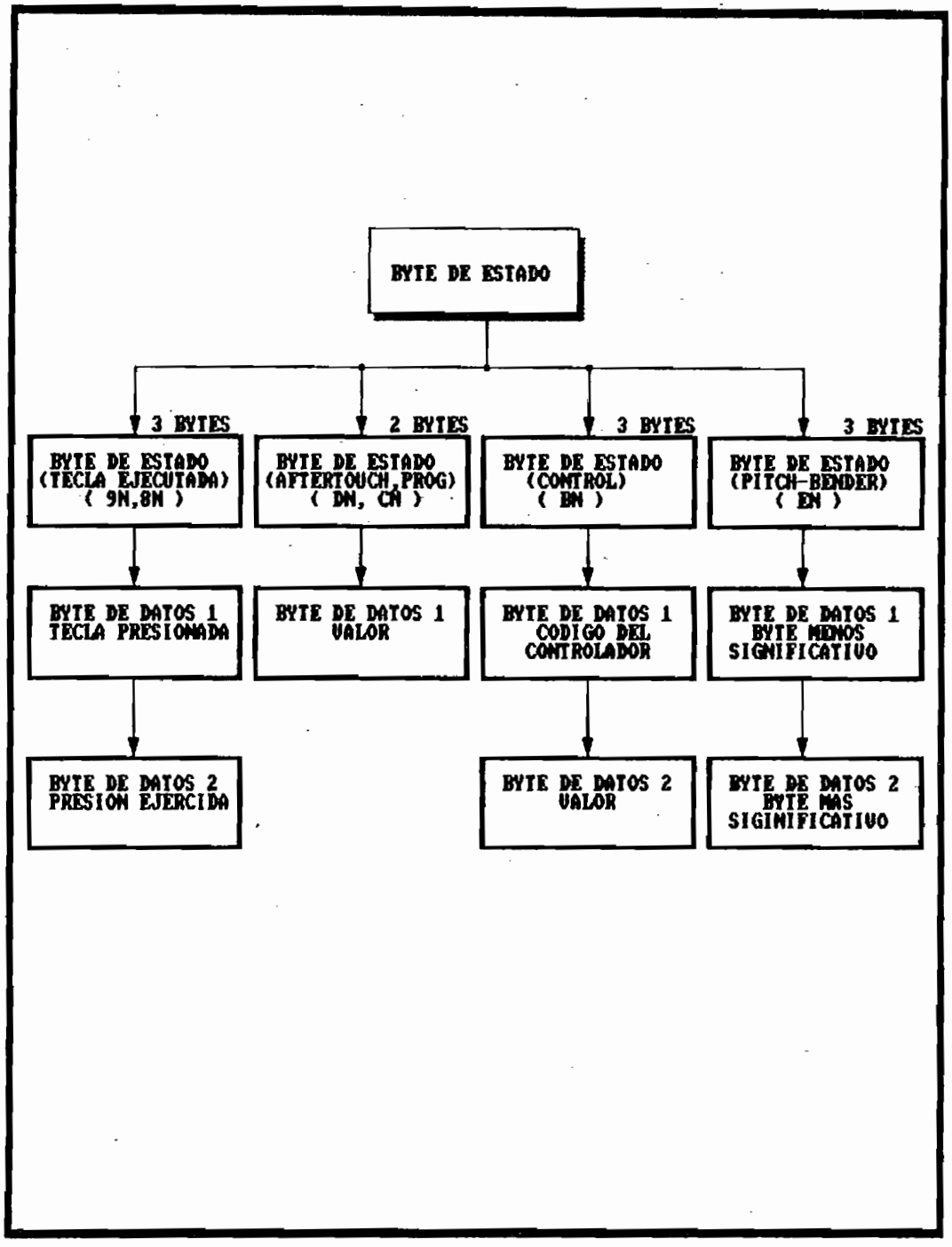


DIAGRAMA DE BLOQUES DE UN MENSAJE MIDI BASICO

FIGURA 3.5

mensajes de 2 y 3 bytes de información. El primer byte recibe el nombre de byte de estado, el cual define que tipo de información será transmitida en los siguientes bytes. El byte de estado siempre tiene su dígito más significativo igual a 1 lógico. En tanto que los bytes que siguen a un byte de estado, siempre tienen como dígito más significativo a un 0 lógico, éstos se usan para transmitir datos.

La información básica que puede ser transmitida por un sistema MIDI, es acerca de la ejecución de una nota musical en un sintetizador. Esta información contiene tres bytes, donde el primero representa que una nota se ha presionado en el teclado y que ésta es transmitida en el canal N. El segundo byte, indica la nota K que ha sido presionada y el tercero indica la presión inicial V que se ejerció sobre esta tecla. Esta información es de la siguiente forma:

INFORMACION	BYTE	COMENTARIO
1000nnnn	Byte de estado	Tecla presionada, se transmite en canal N
0kkkkkkk	Byte de datos	Fue presionada la tecla K
0vvvvvvv	Byte de datos	Fue presionada con presión inicial K.

Se observa, que los 8 bits del byte de estado se usan para representar dos tipos de informaciones. Los 4 bits más significativos anuncian el tipo de dato ha ser transmitido por los siguientes bytes, en tanto que los cuatro bits menos significativos representan el canal por el cual se va a realizar esta transmisión. Debido a que el bit más significativo del byte de estado es siempre igual a 1 lógico, solamente quedan 3 bits para definir el tipo de dato ha ser transmitido, por lo que es posible definir 8 categorías de datos. En tanto, los 4 bits menos significativos del byte de estado sirven para definir el canal de transmisión MIDI, por lo que se pueden tener hasta 16 canales de transmisión y recepción.

El protocolo MIDI define la numeración de canales de comunicación MIDI, de la siguiente manera:

TABLA DE CANALES MIDI			
Ch1 0000	Ch5 0100	Ch9 1000	Ch13 1100
Ch2 0001	Ch6 0101	Ch10 1001	Ch14 1101
Ch3 0010	Ch7 0110	Ch11 1010	Ch15 1110
Ch4 0011	Ch8 0111	Ch12 1011	Ch16 1111

Se observa que la numeración de canales parte desde Ch # 1 para el nibble 0000 y llega a Ch # 16 para el nibble 1111.

3.2.4 Tipos de mensajes MIDI

Los mensajes MIDI, se pueden clasificar en dos categorías:

- mensajes de datos de canal.
- mensaje de datos de sistema.

Debido a que el byte de estado determina el tipo de mensaje a transmitirse, existen también dos categorías de bytes de estado, que son; bytes de estado de datos de canal y bytes de estado de datos de sistema.

Los mensajes de datos de canal siempre se transmiten y/o reciben a través de uno o más de los 16 canales MIDI, en tanto que los mensajes de datos de sistema se envían sin designación de canal.

De manera general, los mensajes de datos de canal llevan información respecto a teclas presionadas, cambios de programas (programa en términos MIDI es equivalente a sonidos), acciones en los pedales de efectos, etc. Los mensajes de datos de sistema transmiten información general del mismo como: señales en tiempo real del reloj MIDI y de sincronización, mensajes comunes del sistema y mensajes exclusivos del sistema.

Existen dos tipos de mensajes de datos de canal, estos son:

- mensajes de datos de canal de voz.

- mensajes de datos de canal de modo.

Los primeros son los más comunes mensajes MIDI, estos transmiten 7 tipos de información diferentes, la que controla la generación de sonido en el sintetizador esclavo, éstas son: nota de tecla presionada, nota de tecla liberada, modulador de frecuencia (pitch bender), velocidad, presión posterior (aftertouch), cambios de programa y cambios de control. En tanto, los mensajes de datos de canal de modo transmiten información de la manera como un sintetizador esclavo debe recibir la información. En la siguiente tabla se representa, la clasificación de los mensajes MIDI.

<table>
<tr><th colspan="3">MENSAJES MIDI</th></tr>
<tr><th colspan="2">MENSAJES DE DATOS DE CANAL
(transmitidos por
canal n)</th><th>MENSAJES DE DATOS
DE SISTEMA
(no se transmiten
por ningún canal</th></tr>
<tr><th>CANAL DE VOZ</th><th>CANAL DE MODO</th><td rowspan="2">- Reloj MIDI
- Comunes del sistema
- Exclusivos del sistema</td></tr>
<tr><td>- tecla presionada
- tecla liberada
- velocidad
- presión posterior
- cambios de control
- "pitch bend"
- cambios de programa.</td><td>Modos de recepción</td></tr>
</table>

3.2.4.1 Mensajes de datos de canal de voz

Estos mensajes se transmiten por cualquiera de los 16 canales MIDI y son recibidos por un esclavo que se encuentre asignado al mismo canal. Tienen por lo menos 2 bytes de información, donde el primer byte corresponde siempre al byte de estado y los siguientes a bytes de datos

Como se indicó anteriormente, este tipo de mensajes pueden ser de 7 tipos diferentes, los que son descritos en detalle a continuación.

- Mensaje de nota y velocidad de tecla presionada:

Este mensaje sirve para transmitir información de la tecla ejecutada en un sintetizador. Esta formado por 3 bytes de información, los que son: byte de estado, byte de número de nota y byte de velocidad.

El byte de estado correspondiente a esta información es del tipo:

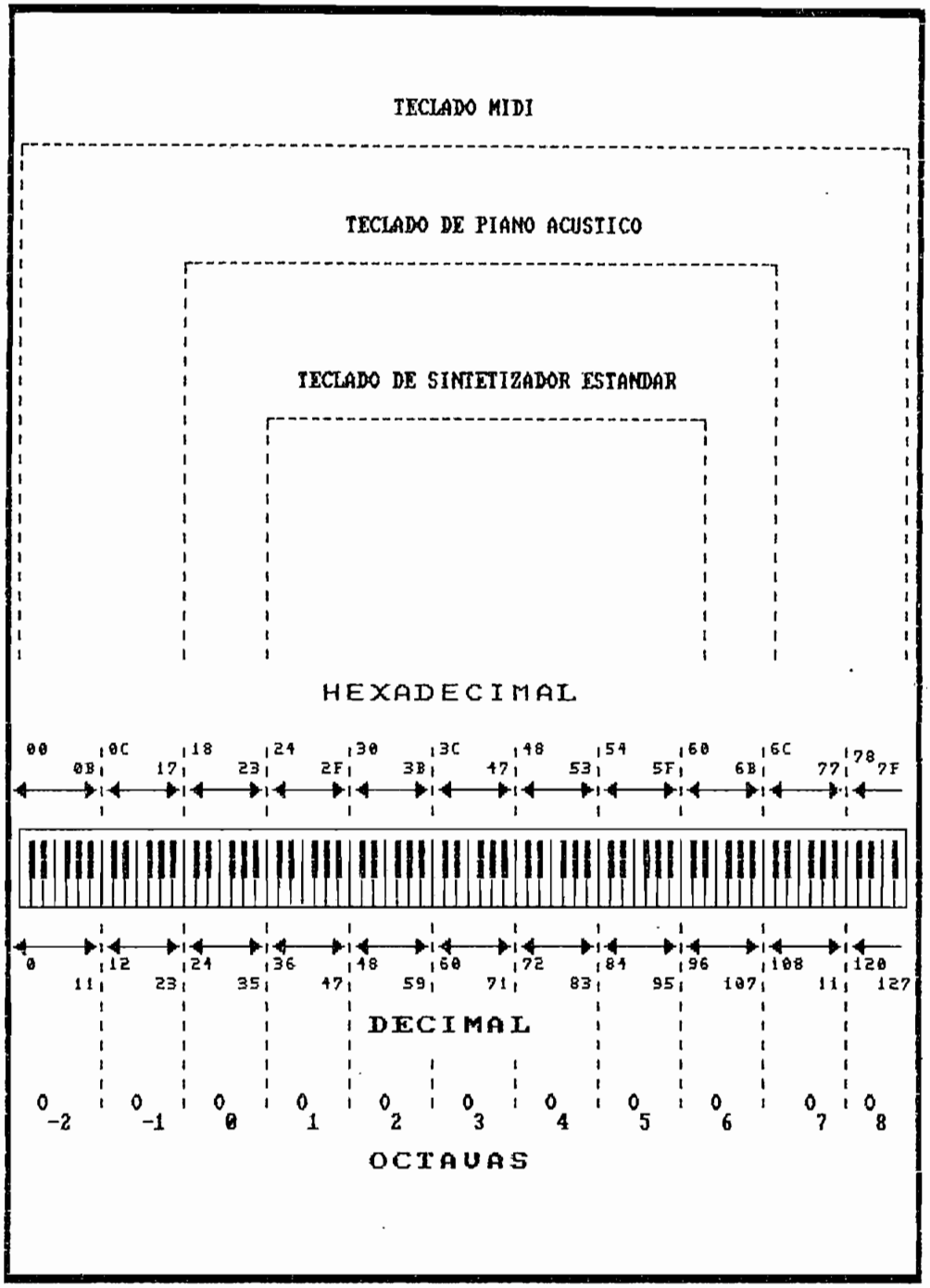
1001nnnn

Como se mencionó anteriormente, este byte siempre inicia con 1 lógico y en numeración hexadecimal corresponde a 9Nh. Donde N (4 bits menos significativos) corresponden al canal de transmisión. Por ejemplo, si el byte de estado correspondiera al byte: 10010001 (91h), esto

indicaría que se ha presionado una tecla y que su información va a ser transmitida en el canal 2.

El protocolo MIDI, define las notas musicales, partiendo desde el Do de la octava -2 hasta el Sol de la octava 8. Es decir se tiene 128 notas musicales, lo que permite sobrepasar el alcance de frecuencia del piano acústico. Estas notas van numeradas desde la nota # 0 (Do_{-2}) hasta lo nota # 127 (Sol_{8}). El teclado MIDI estándar tiene solamente 60 teclas desde Do_{1} hasta Do_{6}, aunque existen teclados que tienen una octava extra. En la figura 3.6 se representa el teclado MIDI estándar con la asignación de notas del protocolo tanto en valor decimal como en exadecimal. Es más común definir las notas en valores exadecimales, lo que facilita mucho su representación binaria. Las notas van numeradas desde la nota # 24h (36_{10}) hasta la nota 60h (96_{10}), correspondiendo el Do central (262 Hz) a la nota # 3Ch (60_{10}).

Generalmente los sintetizadores MIDI, pueden responder a informaciones de número de nota que estén fuera del rango de su propio teclado. Esto se debe a que sus circuitos de síntesis de frecuencia pueden generar rangos mayores a los que determinan las octavas del mismo. Por esta razón todo teclado MIDI, tiene un teclado de ejecución (teclado real) y un teclado virtual (dado por el rango de generación de frecuencia de su circuitería interna).



CODIGOS DECIMALES Y HEXADECIMALES DEL TECLADO DEFINIDO POR EL PROTOCOLO MIDI

FIGURA 3.6

El byte de número de nota se define con 7 bits (tiene el bit más significativo en 0 lógico). Estos 7 bits definen una de las 128 teclas. Por ejemplo, si el byte de número de nota correspondiese a: 00111100 (3Ch), esto significaría que se ha presionado la tecla correspondiente al Do central.

El tercer byte, corresponde a la información de la velocidad con la que se presionó la tecla en el sintetizador, esto es equivalente al volumen con el cual sonó la tecla presionada. Esta información es transmitida por todo sintetizador MIDI, pero actúa solamente en sintetizadores sensibles al tacto (touch - sensitive). De manera similar al byte de número de nota, el rango de este byte de datos va desde 0 hasta 127. La equivalencia de este valor con la notación musical tradicional es:

CARTA DE DINAMICA DE VELOCIDAD		
FUERZA	VALOR DE VELOCIDAD MIDI	
ppp	20	14h
pp	40	28h
p	50	32h
mp	64	40h
mf	70	46h
f	80	50h
ff	90	5Ah
fff	115	73h

Por ejemplo, si se transmiten los siguientes bytes la información que envían es:

10010000 = 90h estado: Tecla presionada, ch # 1.
00111100 = 3Ch nota: Do_3.
01000000 = 40h fuerza: mp.

De manera similar a este mensaje MIDI, existe el mensaje de nota liberada, el cual se define de manera muy similar al mensaje de nota presionada. La diferencia radica en que el byte de estado es 8Nh, por ejemplo:

10000000 = 80h estado: Tecla liberada, ch # 1.
00111100 = 3Ch nota: Do_3.
01000000 = 40h fuerza: mp.

Existe otro formato de información de tecla liberada, éste consiste en enviar datos de tecla presionada pero con velocidad igual a cero. Por ejemplo:

10010000 = 90h estado: tecla presionada, ch # 1.
00111100 = 3Ch nota: Do_3.
00000000 = 00h velocidad: tecla liberada.

- <u>Mensaje de presión posterior (Aftertouch)</u>:

Se define como presión posterior a la fuerza

que es aplicada a una tecla una vez que ha finalizado el movimiento de la misma. Este tipo de respuesta generalmente es transmitida solo por sintetizadores profesionales. Este efecto puede darle varias características al sonido emitido, por ejemplo, el sonido puede aumentar su volumen, adquirir más brillo o tomar cierto efecto de vibrato, esto depende del sintetizador que reciba esta información.

El número de bytes correspondiente a este tipo de mensaje es 2. El byte de estado que informa este mensaje es: 1101nnnn (DNh), donde N corresponde al canal de transmisión. (Si el efecto de presión posterior es polifónico el byte de estado tiene el formato: ANh).

El byte de datos tiene un rango desde 0 hasta 127 para lo cual usa los 7 bits menos significativos. Un ejemplo de bytes correspondientes a este mensaje es:

11010001 = D0h estado: presión posterior, ch # 2.
01010010 = 52h presión posterior: 82.

Generalmente, la presión de una tecla transmite cientos de mensajes de presión posterior. Si esta información va a ser procesada por un secuenciador, no es conveniente hacerlo desde el punto de vista de espacio en memoria. Además, debido a que el efecto de presión posterior no es observable en todo sonido y a que no todo

sintetizador responde a esta información, es práctica común eliminar la transmisión del mismo.

- Mensajes de cambios de control:

Esta clase de mensajes transmiten información referentes a acciones realizada sobre los controladores de teclado. Un controlador de teclado es un elemento que sirve para añadir efectos adicionales a los sonidos emitidos por el sintetizador.

El protocolo MIDI establece 128 tipos diferentes de controladores de teclado; en la actualidad este número es mucho menor, por lo que la mayoría de éstos están indefinidos, pero dejan lugar para futuros desarrollos en este campo tecnológico.

En la siguiente tabla se presenta la carta de controladores con sus respectivos números de asignación.

CARTA DE CAMBIOS DE CONTROL

ASIGNACION	CONTROLADOR
0	Indefinido
1	Rueda de modulación
2	Controlador de viento
4	Controlador de pedal
5	Tiempo de portamento

6	Entrada de datos
7	Volumen
8	Balance
9	Indefinido
10	Posición estereo
11	Controlado de expresión
12 - 15	Indefinido
16 - 19	Controladores de propósito general
20 - 31	Indefinido
32 - 63	bms (controladores # 0 - # 31)
64	Pedal de sostenido
65	Portamento
66	Sostenuto
67	Pedal de suavización
68	Indefinido
69	Sotenido 2
70 - 79	Indefinido
80 - 83	Propósito general (# 5 - # 8)
84 - 91	Indefinido
92	Profundidad de trémolo
93	Profundidad de coro
94	Profundidad de celeste
95	Profundidad de defase
96	Incremento de datos
97	Decremento de datos
98	Parámetro no registrado (bms)
99	Parámetro no registrado (BMS)
100	Parámetro registrado (bms)

101	Parámetro registrado (BMS)
102 - 121	Indefinido
122 - 127	Mensajes de canal de modo

El número de controladores disponibles por un sintetizador se encuentra indicado en la carta de implementación MIDI del mismo. Actualmente los controladores más comunes son: rueda de modulación, filtro de modulación, pedal de expresión y pedal de sostenido. En sintetizadores profesionales el número de controladores disponibles es mayor.

De manera breve se explicará la función de varios controladores.

- Modulación (CC # 1): Toda la información emitida por la rueda de modulación de amplitud (joystick en algunos sintetizadores), es enviada por medio del controlador # 1. Este controla la modulación de amplitud (vibrato), su rango va desde 0 a 127.

- Controlador de viento (CC # 2): Este controlador fue introducido por YAMAHA, en la actualidad existe un terminal para conectar este controlador en sintetizadores de otras marcas. Este permite el control de varios parámetros del sonido como: ataque e intensidad. Su rango va desde 0 a 127.

- Volumen (CC # 7): La información de cambios de volumen. Los cambios de volumen son transmitidos a partir de variaciones continuas en el pedal de expresión. Su rango va desde 0 a 127.

- Pedal de sostenido: Esta información es transmitida desde el pedal de sostenido. Este tiene solamente dos estados: conectado o desconectado.

Una de las características más importantes del protocolo MIDI, es que permite la asignación indistinta de cualquier controlador a cualquier parámetro que se desee controlar. Es decir, el usuario puede asignar un controlador a la variación del parámetro de su conveniencia. Para realizar esta asignación, se debe considerar la manera como dicho parámetro responde a las variaciones del controlador.

Los controladores tienen varios tipos de acciones, las que pueden ser continuas o discretas. Su asignación depende de la aplicación. Desde este punto de vista se los agrupa de la siguiente manera:

- Primer Grupo (desde CC # 0 al CC # 31):
  Este tipo de controladores son útiles cuando se les asigna a parámetros de variación continua tales como: modulación, tiempo de portamento, localización estereo, volumen, etc. Entre estos controladores

tenemos: pedales continuos, potenciómetros y joysticks.

- Segundo grupo (desde CC # 32 al CC # 63):
  Este grupo trabaja en conjunción con el grupo anterior. En estos controladores, se indica el byte menos significativo (bms) de los controladores # 0 al # 31.

- Tercer grupo (desde CC # 64 al CC # 83):
  Este grupo de controladores son del tipo de dos posiciones. Controlan efectos tales como sostenido. Si se les asigna un controlador continuo, una variación bajo el punto medio se considera conectado, en tanto que una variación sobre el punto medio se considera desconectado. Sirven además para efectos que son controlados por medio de switches.

- Cuarto Grupo (desde CC # 92 al CC # 95):
  Estos controladores son también de dos posiciones. Pertenecen específicamente al control de profundidad de los efectos.

- Quinto Grupo (CC # 98 al CC # 101):
  La MMA (MIDI Manufacturers Association) y la JMSC (Japanese MIDI Standard Committee), registran y asignan estos controladores específicamente a: sensitividad el "pitch bender", calibración tonal

gruesa y calibración tonal fina. Estos controladores se usan conjuntamente con los controladores: # 6 (Entrada de datos), # 96 (Incremento de datos) y # 97 (decremento de datos). Además de éstos, el protocolo MIDI permite también el uso de controladores específicos no registrados para cada fabricante (CC # 98, CC # 99). Estos son útiles cuando se desea desarrollar procesos de control implementados con software.

Estos mensajes están constituidos por tres bytes de información. El primer byte es el de estado (BNh), el segundo es el número de código del controlador y en el tercer byte se representa la variación que se ejecuta con el respectivo controlador. Por ejemplo:

a) 10110000 = B0h estado: cambio de control, Ch # 1
00000111 = 07h controlador: volumen (CC # 7)
01001000 = 48h valor: 72

b) 10110001 = B1h estado: cambio de control, Ch # 2
01000000 = 40h controlador: sostenido (CC # 64)
00101000 = 28h valor: 40

c) 10110011 = B1h estado: cambio de control, Ch # 4
00000101 = 05h controlador: tiempo de portamento (CC # 5)
01101000 = 68h valor: 104

d) 10111001 = B1h estado: cambio de control, Ch # 10

01000001 = 41h controlador: portamento (CC # 65)

00000000 = 00h valor: 00, portamento apagado.

- Mensaje de Pitch Bend:

Este tipo de mensaje transmite acciones realizadas sobre el "pitch - bender". Este controlador fue el primero en ser incluido en los sintetizadores. Fue inicialmente introducido en el sintetizador de Moog en la década de los 60. Es por esta razón que las primeras versiones del protocolo MIDI, lo consideraron separadamente aunque en la actualidad se lo considera dentro de los controladores de teclado. Es por esto que puede ser transmitido de dos maneras: la primera como un controlador más y la segunda con su propio byte de estado (ENh) más dos bytes de datos, el primero contiene el byte menos significativo y el segundo el byte más significativo. Por ejemplo:

11100001 = E1h estado: acción en pitch bender, Ch # 2

00100001 = 21h valor: bms

00010100 = 14h valor: BMS

- Mensajes de Cambios de Programa:

Se define como programa a una serie de ajustes (patches) en los elementos de síntesis, éstos

generan los diferentes sonidos del sintetizador. El cambio de sonidos en el sintetizador genera mensajes de cambios de programa que contienen 2 bytes de información. El byte de estado correspondiente a este mensaje es: CNh, donde N es el número de canal. El segundo byte corresponde al número de sonido. Este valor tiene un rango desde 1 hasta 128. Debido a que cada fabricante tiene una codificación diferente para sus programas internos, es necesario realizar una transformación para obtener el número de programa que establece el protocolo MIDI. Los siguientes son ejemplos de este tipo de mensajes:

a) 11000000 = C0h estado: cambio de programa, Ch # 1
   00000001 = 01h valor: programa # 2

b) 11000010 = C2H estado: cambio de programa, Ch # 3
   00010001 = 11h valor: programa # 17

3.2.4.2 Mensajes de datos de canal de modo

Este tipo de mensajes son transmitidos por cualquier canal MIDI. Llevan información referente a la manera como un sintetizador esclavo debe responder a la misma. Esta puede ser de 4 tipos:

- OMNI ON / POLY: OMNI ON representa que la información puede ser recibida por los 16 canales. Pero cada canal tiene asignado un mismo sonido. POLY representa

que los sonidos generados pueden ser polifónicos (acordes).

- OMNI OFF / POLY: Varios sintetizadores vienen inicializados con este modo cuando salen de fábrica. En este modo el sintetizador puede responder sólo al canal que le asigne el usuario, aunque la información llegue por todos los canales. La respuesta es de tipo polifónico.

- OMNI ON / MONO: En este modo cada sonido es asignado a un canal diferente y cada sonido de cada canal suena monofónicamente. Este modo es seleccionado cuando se usa como controlador maestro a una guitarra MIDI. En este caso, cada cuerda de la guitarra genera sólo sonidos monofónicos asignándolo un canal a cada una, pudiendo asignar a cada canal un sonido diferente.

- OMNI OFF / MONO: En este modo el sintetizador esclavo puede recibir solo por un canal y generar solo un sonido a la vez. Este es muy raramente usado.

Los mensajes de canal de modo se transmiten como si fuesen mensajes de cambio de control. Están formados por 3 bytes de información, donde el byte de estado corresponde al mismo byte de estado de los mensajes de cambio de control (BNh), N es el canal de transmisión.

El segundo byte corresponde al número de controlador. Para enviar los mensajes de modo, el protocolo MIDI designa los canales 122 al 127. Este código representa a los 4 posibles modos de transmisión.

Por ejemplo, para que el sintetizador reciba 16 canales a la vez (OMNI ON), se deben transmitir 3 bytes y para que responda polifónicamente (POLY) se deben transmitir 3 bytes adicionales. Nótese que el tercer byte de los mensajes OMNI ON, OMNI OFF y POLY ON es siempre 00h. En tanto, los 7 bits menos significativos del mensaje MONO ON representan el número de notas que serán recibidas cada una por un canal independiente.

- OMNI ON

```
10110000 = B0h      estado: cambio de control, Ch # 1
01111101 = 7Dh      controlador # 125, Modo de canal,
00000000 = 00h      código de OMNI ON
```

- OMNI OFF

```
10110000 = B0h      estado: cambio de control, Ch # 1
01111100 = 7Ch      controlador # 124, Modo de canal,
00000000 = 00h      código de OMNI OFF
```

- POLY ON

```
10110000 = B0h      estado: cambio de control, Ch # 1
01111111 = 7Fh      controlador # 127, Modo de canal,
0mmmmmmm = 00h      código de POLY ON
```

- POLY OFF

10110000 = B0h estado: cambio de control, Ch # 1

01111110 = 7Eh controlador # 126, Modo de canal,

0mmmmmmm = 00h código de POLY OFF

3.2.4.3 Mensajes de datos de Sistema

Este tipo de mensajes no llevan asignación de ningún canal MIDI. Sirven para transmitir información especial. Si un sintetizador esclavo recibe esta información, éste podrá o no responder a la misma; esto depende de si el sintetizador está o no equipado para recibir información MIDI exclusiva.

Los mensajes de datos de sistema son de tres tipos: mensajes de reloj MIDI, mensajes comunes del sistema y mensajes exclusivos del sistema.

- Reloj MIDI

Debido a que toda acción llevada a cabo por un sistema MIDI es en tiempo real, es necesario contar con un reloj. Este se usa para realizar todos los controles de sincronización del sistema. Esta señal es muy importante en aplicaciones de secuenciador, grabadoras MIDI y máquinas de ritmo. El reloj MIDI del sintetizador maestro sincroniza su funcionamiento con los sintetizadores esclavos. Todo elemento MIDI permite la selección de un

reloj interno o externo. De tal manera que si en el sintetizador maestro se selecciona al reloj interno, en el sintetizador esclavo debe seleccionarse al reloj externo.

Los mensajes del reloj MIDI, están todos formados por 1 byte de información. El mensaje de reloj fundamental es: 11111110 y se transmite cada 50 ms. La siguiente lista contiene todos los mensajes de reloj que transmite un sistema MIDI.

1) 11111000 = F8h estado: reloj musical del sistema.
2) 11111001 = F9h estado: todavía no definido.
3) 11111010 = FAh estado: inicio (se transmite al inicio de una secuencia).
4) 11111011 = FBh estado: luego del inicio.
5) 11111100 = FCh estado: final (se transmite cuando finaliza las secuencia).
6) 11111101 = FDh estado: todavía no definido.
7) 11111110 = FEh estado: sensor activo (determina la desconexión de un circuito).
8) 11111111 = FFh estado: reseteo del sistema.

- <u>Mensajes comunes del sistema</u>:

Esta clase de mensaje se usa para enviar información referente al control de tiempo musical así como información para afinar un sintetizador analógico. Sus posibles mensajes son:

1) 11110001 = F1h estado: todavía no definido.

2) 11110010 = F2h estado: puntero de posición
   00000000 = 00h musical.
   01111111 = 7Fh

3) 11110011 = F3h estado: selección de canciones.
   0kkkkkkk valor: # de la canción seleccionada.
   (0 - 127)

4) 11110100 = F4h estado: todavía no definido.

5) 11110101 = F5h estado: todavía no definido.

6) 11110110 = F6h estado: afinación de sintetizador
   analógico (tune request).

7) 11110111 = F7h estado: bandera de finalización de un
   mensaje exclusivo.

- Mensajes Exclusivos del sistema:

Estos mensajes llevan información exclusiva para ser transmitida a un sintetizador de la misma marca y modelo. Esta información es solamente recibida por un sintetizador idéntico al transmisor, esto se debe a que cada modelo de sintetizador tiene su propio sistema operativo. Este código incluye un valor ID que es asignado por la MMA (MIDI Manufacturers Association) a cada fabricante de sintetizadores MIDI, si el receptor no reconoce este código, simplemente ignora estos mensajes exclusivos.

Esta clase de mensajes sirven principalmente, para transmitir valores de parámetros de los sonidos almacenados en la memoria del sintetizador. Esto es de enorme utilidad, ya que gracias a ello se puede transferir los parámetros de los sonidos internos a un computador, de tal manera que estos sean editados y almacenados en la memoria del computador, para ser posteriormente devueltos al sintetizador. (5)

Los mensajes exclusivos del sistema pueden incluir cualquier número de bytes, y terminan siempre con la transmisión de un mensaje de finalización exclusivo (ROX) o con un byte de estado (Mensaje común de sistema: F7h).

Esta clase de mensaje depende del modelo de sintetizador, como ejemplo citamos la transmisión de mensajes exclusivos del Sintetizador DX7 de YAMAHA, el cual puede transmitir información sobre parámetros de sus 32 sonidos internos (bulk data).

11110000 = F0h estado: mensaje exclusivo.
01000011 = 43H código: Código asignado por MMA a YAMAHA.
..... Bytes intermedios determinados por el fabricante.
11110111 = F7h código: bandera de finalización de mensaje exclusivo.

En la actualidad, los sintetizadores disponibles en el mercado no procesan toda la información que permite el protocolo MIDI, sin embargo cada día las posibilidades de los modernos sintetizador son mayores. La información de estas posibilidades se encuentran detalladas en la carta de implementación MIDI, cuyo formato esta estandarizado por la MMA y es entregada al usuario con cada equipo MIDI. En el Anexo # 1 se presenta un modelo de la misma.

3.3 CARACTERISTICAS DEL SISTEMA MIDI.

En la década pasada, el sistema de comunicación MIDI, era incluido solo en sintetizadores profesionales. En los últimos años se han desarrollado una gran cantidad de equipos procesadores de información MIDI. Entre estos tenemos controladores de teclado, controladores de guitarra, sistemas de percusión, controladores de viento, módulos de sonido, secuenciadores y computadores personales. Es gracias a estos nuevos aparatos que las características de un sistema MIDI, no se restringen a las del sistema MIDI básico (maestro - esclavo).

Una de las principales características del sistema MIDI, radica en su modularidad. Gracias a que los terminales de comunicación no son solo de entrada y salida es posible la conexión de cierto número de equipo MIDI. El protocolo MIDI, establece que si se usan los terminales

MIDI THRU, para colocar sintetizadores en serie, el número de estos no debe exceder a 4 sintetizadores, ya que de lo contrario se degradarían las señales digitales. Para evitar este inconveniente se han desarrollado Expansores MIDI, los cuales tienen un terminal MIDI IN y 8 terminales de salida MIDI OUT, esto permite la conexión de 8 sintetizadores en serie. De manera similar existen lo que se conoce como mezcladores MIDI, estos reciben señales de varios terminales MIDI IN y presentan a su salida un solo terminal MIDI OUT.

Esta modularidad ha permitido el desarrollo de aparatos MIDI en los que se han individualizado el procesamiento de información y generación de sonidos. Es así que, actualmente se encuentran en el mercado Módulos de Sonido, estos solo pueden recibir información que comanda su sistema de generación de sonidos. Estos no incluyen teclados y se comportan solo como elementos esclavos.

Otra característica muy importante del sistema MIDI, radica en el tipo de información que este puede transmitir. Hemos visto que los mensajes MIDI pueden ser de canal y de sistema. Los mensajes de sistema permiten transferir datos internos del sintetizador. La mayoría de elementos MIDI, aceptan un catridge de memoria para almacenar información de sus registros de sonido. Esta información, previamente transformada en mensajes MIDI exclusivos, puede ser transmitida a un computador personal

convenientemente equipado. Esto implica que un sintetizador MIDI puede comportarse como un periférico más para un computador personal. En esta situación, el computador se transforma en un poderoso controlador maestro.

Para que un computador personal pueda comunicarse con una red MIDI, este debe incluir por lo menos un pórtico de comunicación MIDI. Generalmente este pórtico de comunicación es un elemento externo al computador y se lo conoce como interfaz MIDI. Básicamente existen dos tipos de interfaz MIDI. El primer tipo es un interfaz inteligente, en el cual existe un microprocesador MPU (MIDI Processing Unit) que realiza todas las rutinas de recepción, transmisión y sincronización de información sin la ayuda del microprocesador del computador. El CPU del computador simplemente se encarga de enviar la información al MPU del interfaz MIDI. El segundo tipo de interfaz, no incluye un MPU y requiere que el CPU del computador lleve el control de datos y de tiempo de la información transferida. En el mercado existen varios tipos de interfaz MIDI, pero el primer interfaz compatible con IBM que fue aceptado por MMA, corresponde al MPU - 401 de ROLAND, desde entonces este se ha convertido en el interfaz estandarizado para comunicación MIDI.

3.4 POSIBILIDADES DEL SISTEMA MIDI.

El sistema MIDI pertenece a los más

recientes desarrollos tecnológicos, es por eso que varias de las posibilidades que se mencionará a continuación han sido introducidas en los dos primeros años de esta década.

- Código de temporización MIDI:

En los últimos años el sistema MIDI ha sido adoptado para la generación de efectos de sonido en estudios de audio y video. El código de sincronización usado en la industria de audio y video es el SMPTE (Society of Motion Pictures and TV Engineers), el cual permite la sincronización de aparatos de audio y video. Este código es compatible con el código de temporización MIDI, de tal manera que se puede realizar una sincronización exacta entre la generación de sonidos del sintetizador y las señales de audio o video.

- Redes de comunicación MIDI:

La conexión de varios sistemas MIDI requiere el uso de redes de computo más poderosas. Una de las tecnologías para conectar una serie de equipos de computo es la conocida como LAN (Local Area network), pero esta no ha podido ser aplicada al sistema MIDI, debido a que la información MIDI es en tiempo real, y el protocolo LAN no permite la transmisión de información en tiempo real, es decir, este trata la información en paquetes dando comandos de prioridad, lo que desde el punto de vista de información

musical no es aceptable.

En 1992 se desarrolló un nuevo protocolo que permite que un sistema MIDI use redes LAN. Esta nueva red recibió el nombre de "MIDItap", la que combina el código MIDI con el código "Media Link". El sistema MIDItap permite controlar un sistema de equipos MIDI. Dentro del sistema se puede definir a un "elemento", el cual puede ser un sintetizador o un grupo de sintetizadores. Una vez que este ha sido definido, puede ser direccionado cada vez que se lo necesite. Esto posibilita la selección de elementos de destino y de origen y así decidir cual elemento actúa como esclavo o cual como controlador master. Este aplicación esta en la etapa de investigación pero es muy probable que sea el siguiente paso de la tecnología MIDI.

- <u>MIDI Genérico</u>:

Si bien el sistema MIDI ha estandarizado la etapa de comunicación entre sintetizadores, actualmente cada fabricante asigna los programas de cada sintetizador de manera diferente. Por ejemplo, si se ha usado un secuenciador para crear una secuencia musical con el sintetizador del fabricante A y se la quiere ejecutar en un sintetizador del fabricante B, en el mejor de los casos se obtendrá una reproducción alterada de la información original. Para ser reproducida correctamente, primero la información debe ser editada para adecuarse al sintetizador

B. Este proceso se evitaría si el protocolo MIDI, incluiría además un patrón de estandarización de las voces internas de todo sintetizador. Este variación al protocolo MIDI está siendo considerada bajo lo que se denominará MIDI Genérico.

En el Capítulo IV, visualizaremos de manera más clara dos de las posibilidades de un sistema MIDI, estas son un analizador de datos MIDI y un secuenciador MIDI basado en software.

C A P I T U L O IV

IMPLEMENTACION DE LOS PROGRAMAS

IMPLEMENTACION DEL PROGRAMA

4.1 ESPECIFICACIONES Y ALCANCE DEL PROGRAMA

4.1.1 Justificación del Lenguaje de Programación

Actualmente, las computadoras son uno de los elementos más importantes en un sistema MIDI. Es por está razón que en el presente Capítulo se presenta el diseño de un paquete de programas que permite visualizar la aplicación de las computadoras a los sistemas de síntesis del sonido.

Antes de enfrentar el problema del diseño de dicho programa, es importante, encontrar un lenguaje de programación que permita lograr nuestro objetivo de una manera satisfactoria. Debido a que el número de lenguajes de programación existentes en la actualidad es bastante amplio, para escoger uno de estos debemos realizar un análisis de las posibilidades de los lenguajes más comunes y así determinar cual cumple con nuestros requerimientos.

El lenguaje de programación conocido con el nombre de Assembly, es quizá el lenguaje ideal desde el punto de vista de velocidad de ejecución, pero debido a que sus instrucciones son a nivel de bits, requiere un gran conocimiento del mismo para lograr realizar nuestra objetivo de una manera satisfactoria. Otro posible

lenguaje de programación es el Basic, pero debido a su baja velocidad de ejecución y a su sintaxis no estructurada, no es útil para aplicaciones MIDI. El lenguaje Fortran, aunque es muy poderoso para aplicaciones matemáticas, no es adecuado para programación de sistemas MIDI. Existen otras posibilidades dentro de los lenguajes de medio y alto nivel (Lisp, Prolog, Modula, Ada), pero estos presentan inconvenientes en uno u otro sentido, lo que los hace no aptos para nuestro propósito.

Una característica fundamental, que debe ser tomada en cuenta al escoger un lenguaje de programación para sistemas MIDI, es que este tipo de sistemas procesan información en tiempo real, es decir que el factor velocidad de procesamiento debe ser el parámetro fundamental que nos guíe a escoger un lenguaje de programación. Entre los lenguajes que satisfacen nuestros requerimientos están el lenguaje conocido como "C" , lenguaje Forth y Pascal, estos son más fáciles de escribir que el Assembly, pero el costo frente a este es un tiempo de ejecución mas largo y un tamaño de programa mayor. De estos tres lenguajes escogemos el lenguaje "C". Este lenguaje de programación es en la actualidad uno de los más populares. La mayoría de programas comerciales para aplicaciones MIDI se encuentran escritos en una combinación de lenguaje "C" y lenguaje Assembly.

El lenguaje C, es ideal para escribir

programas para aplicaciones MIDI. Este cuenta con operadores a nivel de bit, lo que lo hace muy útil para codificar y decodificar los datos MIDI. Otra característica de este lenguaje de programación radica en la facilidad de realizar interfaces con funciones escritas en lenguaje Assembly. Esto nos permitirá realizar las subrutinas de comunicación en lenguaje Assembly las que se comportan como extensiones del lenguaje C. El costo en velocidad que se sacrifica al usar un lenguaje de medio nivel, se lo recupera al realizar las subrutinas de comunicación en lenguaje de máquina. Además, debido a la naturaleza modular del lenguaje C, este permite desarrollar los programas en módulos separadas, facilitando el desarrollo de los mismos.

4.1.2 Hardware requerido

Para su funcionamiento, un sistema MIDI, requiere varios elementos básicos, entre estos mencionamos:

- Por lo menos un sintetizador de sonido equipado con terminales de comunicación MIDI. (MIDI IN y MIDI OUT).

- Computador personal, 100 % compatible con IBM PC.

- Interfaz de comunicación MIDI.

El sistema que usaremos para demostrar

varias de las posibilidades de un sistema MIDI, consta de los siguientes elementos, los que se describe con sus respectivas características.

- Sintetizador MIDI: El sintetizador que se usará pertenece al modelo HT-700 de CASIO. Este sintetizador digital de sonido, usa como técnica de síntesis al método por eliminación, parte de 32 ondas básicas para sonidos de melodía y 16 para sonidos de acompañamiento; las ondas básicas han sido muestreadas con técnica PCM. Presenta 20 programas de sonido almacenados en ROM interna, 20 programas sintetizados por el usuario almacenados en RAM interna; y, 20 programas sintetizados por el usuario almacenados en RAM externa. Cuenta además, con diez secuencias de ritmo: bajo, percusión, melodía almacenados en ROM interna, diez secuencias de ritmo programables almacenados en RAM interna y diez secuencias de ritmo programables almacenados en RAM externa. Incluye además un secuenciador de ritmo en hardware (memoria de acordes / operación). Esta permite registrar hasta 1280 acordes y 396 cambios de programa en memoria RAM interna; y, 1280 acordes y 396 cambios de programa en memoria RAM externa. Este sintetizador estereo tiene además tres controladores de pedal de sostenido, coro y rueda de modulación (Pitch / Bender). Cuenta además con un mecanismo de transposición de hasta una octava. En cuanto a sus capacidades MIDI, el HT - 700 presenta: terminales de comunicación MIDI IN, MIDI OUT, 13 canales de comunicación

MIDI, selección de reloj interno o externo y 3 modos de comunicación básicos:

Modo A: Envía y recibe mensajes MIDI como un teclado simple (hasta 8 notas polifónicas). En la figura 4.1 (a) se presenta gráficamente este modo.

Modo B: Divide al teclado en 3 secciones, una sección de 4 notas polifónicas (melodía superior), una sección de 3 notas polifónicas (acordes) y una sección monofónica para utilizar como fuente de sonido múltiple. Esta información se transmite a un secuenciador, seteado en reloj interno. Los canales MIDI para esta transmisión son:

Ch n = melodía

Ch n+1 acordes

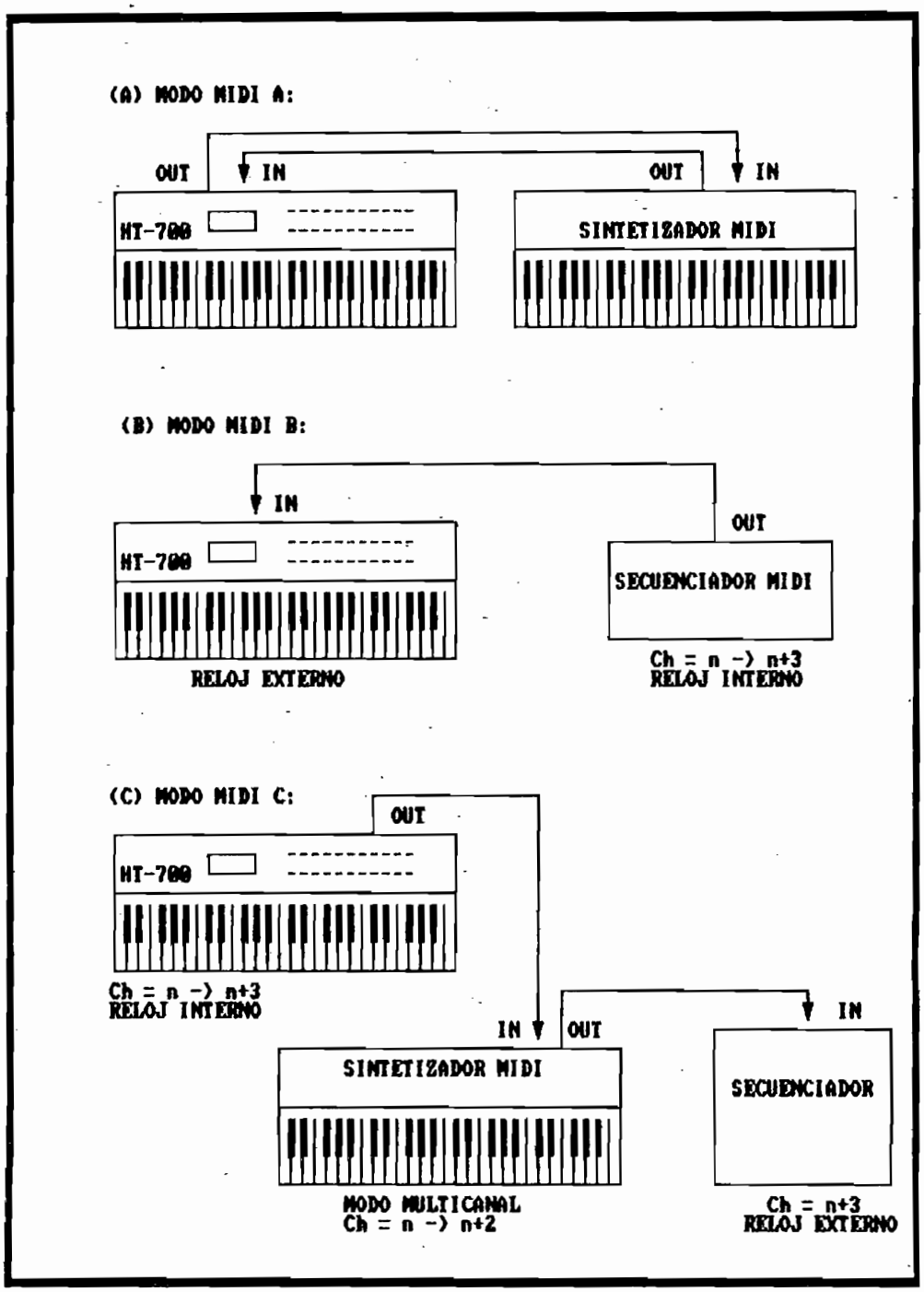
Ch n+2 bajo

Ch n+3 ritmo

En la figura 4.1 (b) se presenta el diagrama de este modo.

Modo C: Este modo envía múltiples mensajes MIDI para la ejecución de melodías y patrón de acompañamiento. (Sirve para ser usado como un secuenciador de reserva mediante la memoria de acordes / operación). En la figura 4.1 (c) se presenta el diagrama de este modo.

- Computador Personal: El computador a usarse corresponde a un computador equipado con el microprocesador 80286 100%



DIAGRAMAS DE MODOS DE COMUNICACION MIDI DEL SINETATIZADOR HT-700 DE CASIO

FIGURA 4.1

compatible con IBM PC - AT.

- Interfaz de comunicación MIDI: El interfaz que se usará para realizar comunicación entre el sintetizador HT-700 y el computador personal corresponde al modelo V-4000 de VOYETRA, el cual es 100 % compatible con el estándar ROLAND MPU-401. Este es un interfaz inteligente ya que incluye un microprocesador ROLAND 6801 y un controlador LSI hand - shake. Estos realizan todas las operaciones de sincronización en tiempo real y reconocimiento de información MIDI, lo que permite que el microprocesador del computador, realice operaciones como manejo de pantalla, manejo de memoria e interfaces al usuario, esto mientras simultáneamente el MPU-401 está procesando información MIDI. El interfaz V-4000 incluye además una salida de audio que entrega señales de 1 Vp-p para ser usadas como metrónomo musical. La salida de reloj corresponde a 0.5 Voltios de colector abierto. Otra característica importante de este interfaz es que permite la selección por del usuario del nivel de interrupciones del microprocesador del computador huésped (IRQ 2, 3, 4 y 7). Los pórticos de comunicación corresponden a :

  Pórtico de datos: 330H

  Pórtico de estado: 331H

Se debe mencionar además que, este interfaz permite expandir los pórticos de comunicación MIDI al número de 4 esto usando la unidad expansora EB-4 también de VOYETRA. Es importante mencionar además que para la implementación

de los programas se cuenta con el manual técnico del interfaz ROLAND MPU-401. Este manual incluye todos los comandos que permiten controlar al interfaz MPU-401 desde el computador.

4.1.3 Especificaciones del Programa

En el Capítulo anterior se ha observado que las posibilidades de un sistema MIDI son muy numerosas. Estas abarcan todo un campo tecnológico que se conoce como Informática Musical. Es por esto que tratar de visualizar todas las posibilidades de un sistema de este tipo está fuera de los alcances de la presente Tesis.

En este Capítulo, presentaremos dos de las múltiples posibilidades de un sistema MIDI. La primera aplicación que visualizaremos corresponde a un analizador de información MIDI. Este nos permitirá visualizar en pantalla, de tal manera que la información recibida desde el sintetizador será desplegada en sus códigos exadecimales correspondientes así como decodificada en sus equivalencias musicales. El programa también permitirá enviar mensajes exadecimales de información MIDI hacia el MPU-401 de tal manera que se observe la respuesta del sintetizador a estos mensajes.

La importancia de un Analizador de datos MIDI, radica en el hecho de que el desarrollo de cualquier

aplicación MIDI, requiere un dominio de este protocolo de comunicación. La mejor manera de lograr lo anterior es visualizando directamente la información transmitida y decodificándola para observar su significado musical de una manera clara.

El segundo programa que se implementará corresponde a una de las más comunes aplicaciones de las computadoras en los sistemas MIDI, este es un Secuenciador basado en software. Se conoce como secuenciador a un sistema que permite grabar en tiempo real información MIDI enviada por un controlador maestro como un sintetizador, procesar esta información y enviarla al sintetizador. Esta información puede ser grabada de manera similar a las grabadoras multitrack análogas, es decir, la información MIDI es registrada en tracks polifónicos independientes, pero a diferencia de las grabadoras multitrack análogas, las que como máximo constan de 32 tracks, un secuenciador MIDI basado en software puede tener cualquier número de tracks dependiendo de la capacidad del computador (generalmente 1000 tracks). (4) Una vez almacenada en el computador, la información MIDI puede ser procesada de varias formas, entre las que se menciona: manejo de la duración de la secuencia musical haciéndola más larga o mas corta, transposición de altura tonal, adición de efectos, reasignación de cualquier parte de la secuencia a cualquier sonido, etc. (5)

Otra característica muy importante de un secuenciador MIDI es el hecho de que el espacio requerido en memoria por la información procesada es extremadamente compacto. Como comparación, una secuencia musical bastante elaborada creada con un secuenciador MIDI, no requerirá un espacio mayor a 30 Kbytes de memoria, en tanto que el espacio ocupado por la misma información procesada en formato de Audio Digital (PCM) de alta calidad está en el orden de las unidades de Mbytes. (4)

El diseño de un secuenciador MIDI con todas las posibilidades de edición es un gran proyecto de Ingeniería de Software. El secuenciador que se diseña en la presente tesis presenta las bases de desarrollo de cualquier secuenciador basado en software. Este corresponde a un secuenciador MIDI prototipo, el mismo que permitirá registrar en tiempo real la información enviada desde el sintetizador, esta podrá ser posteriormente ejecutada por el sintetizador, permitiendo asignar a la secuencia el compás musical que se desee. Como ayuda para marcar el tiempo musical el programa permitirá el control de número y duración de compases musicales, esto por medio del metrónomo que se encuentra incluido en el interfaz MPU-401. El secuenciador permitirá registrar la información en un track polifónico.

Los programas ha diseñarse basan su funcionamiento en la comunicación entre el sintetizador y

el computador. Debido a que la transmisión de información MIDI se realiza a 32.25 Kbaudios, estas requieren un tiempo de procesamiento bastante rápido. La opción más adecuada para lograr este propósito es la escritura de las subrutinas de comunicación en lenguaje de bajo nivel, de tal manera que sean procesadas por el microprocesador del computador en el menor tiempo posible. El lenguaje que se adapta de mejor manera a nuestro objetivo es el Assembly, que para nuestro caso corresponde al Assembly de la familia de microprocesadores 8086.

Para establecer la comunicación computador - sintetizador, el interfaz MPU-401 requiere enviar o recibir tres tipos de mensajes los que son: recibir dato, recibir comando y enviar dato. Estas funciones al ser implementadas en lenguaje de máquina pueden ser ensambladas, de esta manera pueden ser usadas como funciones por los programas escritos en lenguaje "C".

4.2 DISEÑO DE LOS PROGRAMAS

4.2.1 Diseño de las Subrutinas de Comunicación

Uno de los elementos de software básicos en un sistema MIDI, son los drivers que permiten establecer comunicación entre el computador y el sintetizador.

Por razones de velocidad, el lenguaje ideal

editor de texto que entregue código ASCII, se escribirá el texto de las subrutinas en código fuente (.ASM), posteriormente usando un ensamblador como por ejemplo el Macro Assembly de MicroSoft, se transformaran las subrutinas en código fuente a código objeto (.OBJ). Una vez que estas están en código objeto, pueden ser unidas con los programas escritos en lenguaje "C", el que considera a las subrutinas en Assembly como simples funciones.

Para el diseño de estas tres subrutinas, se considera tres módulos separados, los que son escritos con el editor de texto que incluye el paquete de programación Turbo C de Borland. Posteriormente estas tres subrutinas son incluidas en un solo programa en Assembly, el que incluye todos el código necesario para realizar el manejo de memoria del computador desde el lenguaje Assembly. Este programa lo denominaremos: MPUCOM.ASM.

4.2.1.1 Diseño de la subrutina para leer datos desde el MPU:

Esta subrutina nos permitirá leer un byte de dato enviado desde el sintetizador al MPU. En el Capítulo Tercero se estudió el tipo de información que es transmitida desde el sintetizador. Estos datos de información MIDI contienen paquetes de 10 bits, de los cuales la información musical útil es solamente de 8 bits, el bit de inicio y de parada corresponden a códigos que

permiten al MPU diferenciar un byte de mensaje del siguiente. Esta situación no tiene importancia al implementar los drivers del MPU. Esto se debe a que ésta Unidad de Procesamiento MIDI decodifica ésta información y presenta en su pórtico de datos solamente la información MIDI útil, es decir los 8 bits de información.

La información que contiene el MPU es presentada al computador en el pórtico de datos número 330H. (4) El código que presenta el MPU para informar al computador de la presencia de un dato en el pórtico de datos corresponde al comando: "Data Send Ready" (DSR), cuyo equivalente es (01111111) (0 lógico en el bit # 7), este código es enviado por el pórtico de estado signado con el número 331H. Es decir que si se comprueba en el pórtico número 331H, un byte con su bit más significativo en 0 lógico, el MPU tiene un byte de dato en el pórtico número 330H, listo para ser enviado al computador. Esta subrutina debe prever además un lapso de tiempo para leer el dato en el pórtico de datos, esto es útil ya que no siempre puede darse la condición de que el MPU esté listo a enviar un dato el instante que el computador solicita el mismo. Para este fin se incluye un lazo que se repite un cierto número de veces, de tal manera que en este tiempo aparezca un comando de "Data send Ready" en el pórtico 331H. Si no se incluyera este lazo y si por ejemplo se hubiese desconectado algún cable entre el computador y el sintetizador, el computador se quedaría esperando el dato

indefinidamente. Si en el intervalo de tiempo dado por este lazo, el MPU no responde, la subrutina devuelve en lugar de dato un mensaje de error que es definido con: -1. Esto informa al programa escrito en lenguaje "C", que no existe dato en el pórtico de datos del MPU o que existe algún problema de conexiones entre el computador y el sintetizador.

Esta subrutina recibe el nombre de "LEERDATO". El diagrama de flujo correspondiente a la misma se encuentra en la Figura 4.2.

4.2.1.2 Diseño de la subrutina para enviar dato al MPU:

Esta subrutina es muy similar a la rutina LEERDATO. Antes de transmitir el byte de dato desde el computador al MPU, se debe verificar el código presente en el pórtico de estado 331H. Cuando el MPU está listo para recibir un dato presenta un comando "Data Receive Ready" (DRR), este corresponde a un byte donde el bit # 6 corresponde a un 0 lógico (10111111). Es decir antes de transmitir el byte de datos se debe leer el valor en el pórtico de estado, si este presenta el bit # 6 en 0 lógico, se puede transmitir el dato al pórtico de datos 330H

De manera similar a la subrutina anterior, se debe incluir un lazo para dar un lapso de tiempo a que el MPU responda con el código DRR, si por alguna razón,

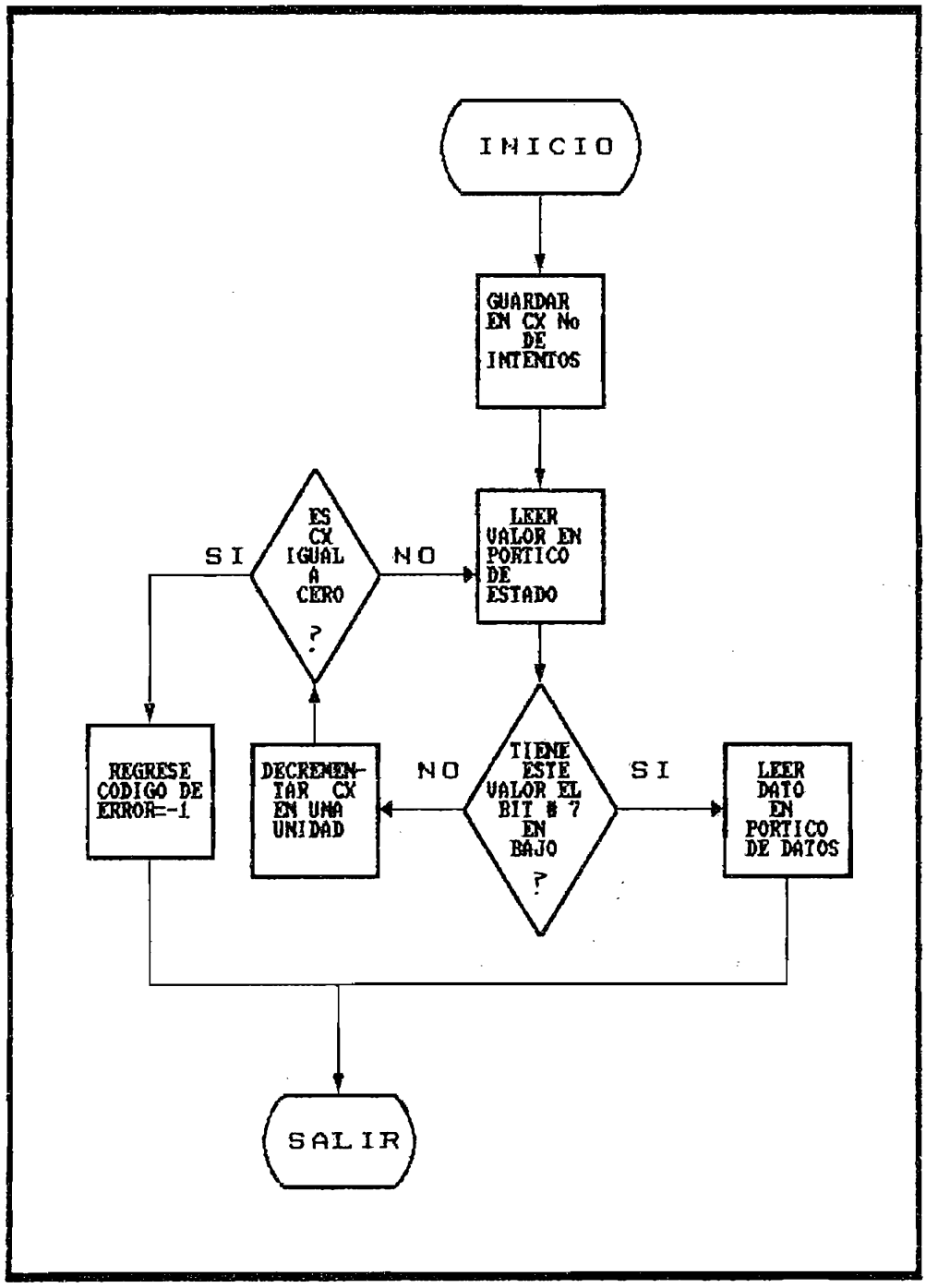


DIAGRAMA DE FLUJO DE LA RUTINA EN ASSEMBLER " LEERDATO "

FIGURA 4.2

este no responde, la subrutina entrega un código de error: -1, de tal manera que el programa principal no se quede indefinidamente esperando la respuesta del MPU. Una vez que el MPU envía el código DRR, la subrutina recupera el dato que ha sido enviado por el programa principal y lo envía al pórtico de datos 330H. El proceso termina regresando el control al programa principal.

Esta subrutina recibe el nombre de "ESCRIBIRDATO". El diagrama de flujo de la misma se presenta en la figura 4.3.

4.2.1.3 Diseño de la subrutina para enviar comando al MPU:

Esta subrutina es diferente a las dos anteriores debido a que cuando el MPU recibe un comando en su pórtico de estado, este responde enviando un código al computador para informarle que el código enviado desde el computador ha sido recibido adecuadamente. Este código recibe el nombre de "Acknowledge" (ACK) y corresponde al valor exadecimal FEH.

La subrutina se inicia con un lazo para verificar que en el pórtico de datos exista el código "Data Receive Ready" (DRR), si el valor en el pórtico de estado corresponde a un byte con el bit # 6 en 0 lógico, la subrutina recupera el comando enviado por el programa principal y lo envía al pórtico de estado 331H. Si el lazo

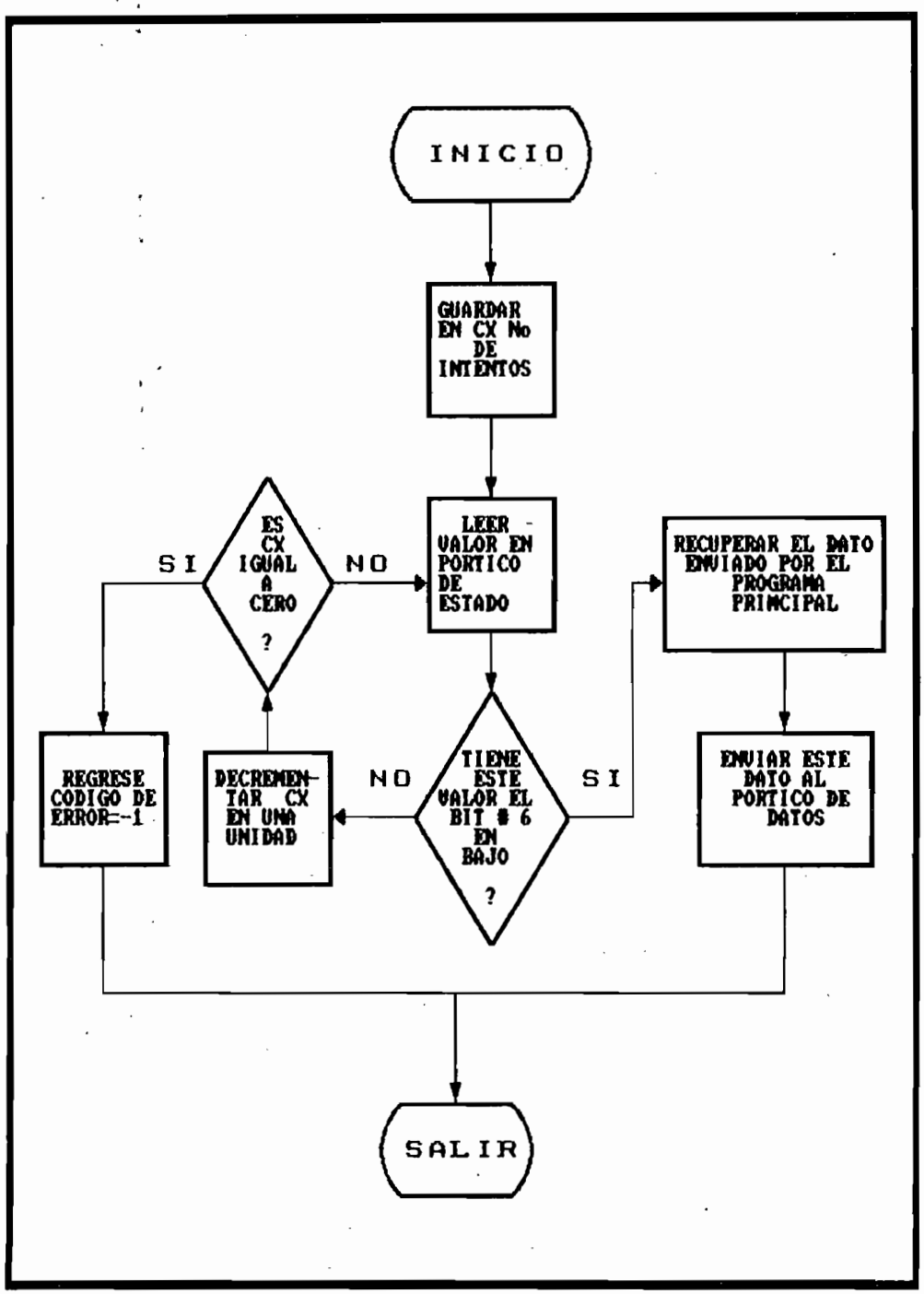


DIAGRAMA DE FLUJO DE LA RUTINA EN ASSEMBLER " ESCRIBIRDATO "

FIGURA 4.3

termina y el MPU no presenta el código DRR, la subrutina devuelve el control al programa principal enviando el mensaje de error: -1.

Una vez que el comando es enviado al pórtico de estado, el MPU responde con el código ACK. Antes de leer este código, la subrutina debe verificar la presencia del código DSR en el pórtico de estado. De manera similar, se incluye un lazo para dar un tiempo para que el MPU responda con el código DSR. Si el lazo termina y el código DSR no es recibido, la subrutina termina enviando el código de error: -1. Si se verifica el código DSR en el pórtico de estado, se debe tomar el dato presente en el pórtico de datos, el cual se compara con el código ACK. Si este valor corresponde a dicho código, el programa envía el valor ACK y devuelve el control al programa principal, en caso contrario, envía el código de error: -1.

Esta subrutina recibe el nombre de "COMANDO". El diagrama de flujo correspondiente a esta, se presenta en la figura 4.4.

4. 2. 2 Diseño del programa analizador de datos MIDI

Un analizador de datos MIDI, es una herramienta muy útil para desarrollar cualquier aplicación MIDI. En cierto sentido este programa nos permitirá abrir una ventana al protocolo MIDI, permitiendo que el

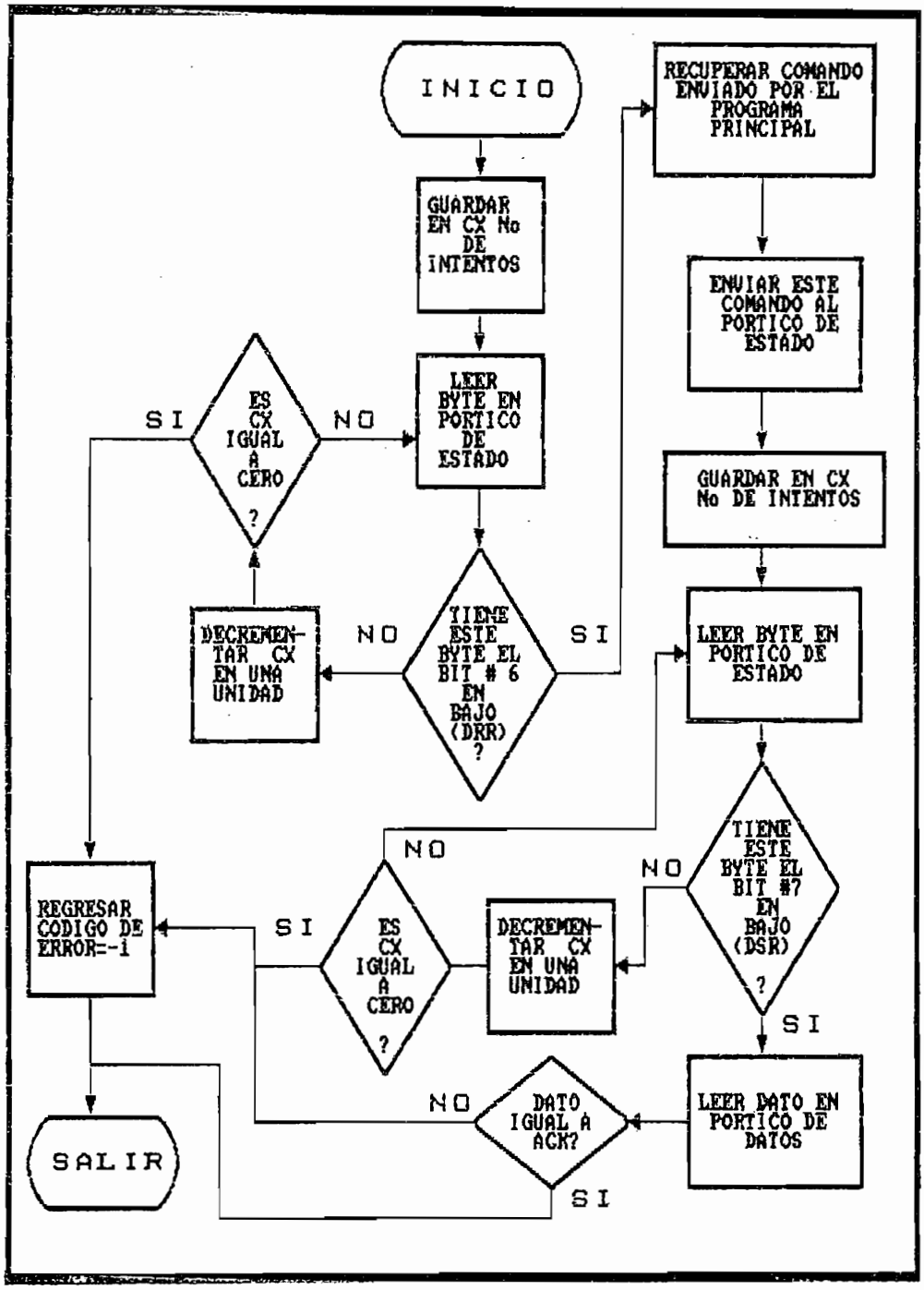


DIAGRAMA DE FLUJO DE LA RUTINA EN ASSEMBLER " COMANDO "

FIGURA 4.4

computador despliegue los códigos exadecimales así como sus equivalencias musicales.

El diseño de este programa se basa en el uso de las subrutinas de comunicación en lenguaje de máquina diseñadas anteriormente. La información MIDI puede ser recibida y transmitida desde y hacia el MPU-401. Los códigos de información recibida deberán además ser decodificados en sus respectivos equivalentes musicales.

La lógica para recibir y transmitir la información exadecimal se describe en el diagrama de flujo que se indica en la figura 4.5. En este se observa que el programa presenta un lazo infinito dentro del cual se realiza la recepción, transmisión y decodificación de la información.

Debido a que la cantidad de bytes que pueden recibirse es bastante alta e imposible de ser decodificada de manera inmediata por el computador, debe preverse un mecanismo de almacenamiento de esta información en un buffer temporal de datos, de tal manera que el computador vaya tomando los datos byte por byte para realizar entonces el procesamiento respectivo.

El envío de mensajes al MPU, se basa en el uso de la función ESCRIBIR_DATO(). El programa deberá leer un byte formado por dos números exadecimales, una vez que

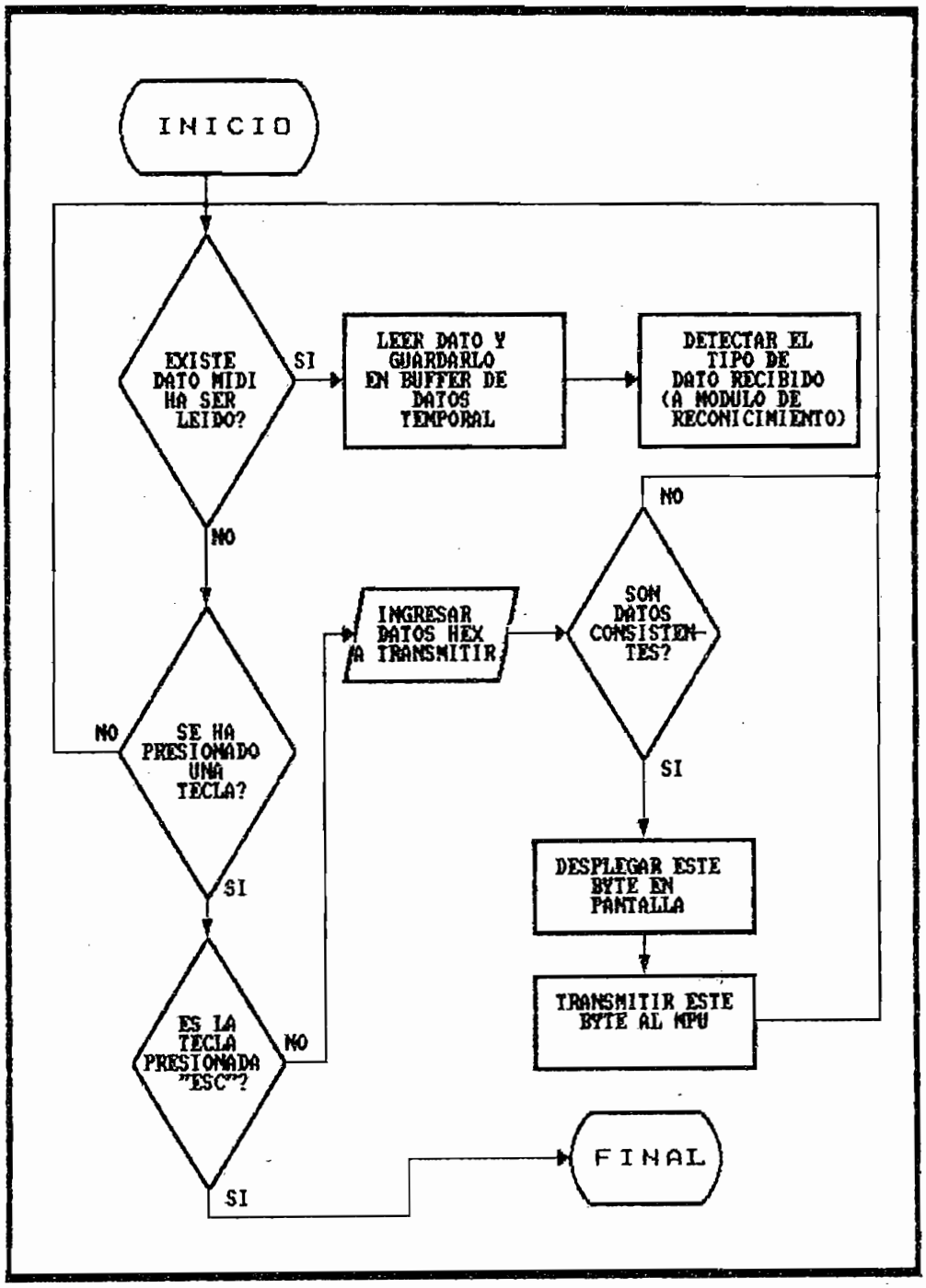


DIAGRAMA DE FLUJO DEL PROGRAMA ANALIZADOR (PROGRAMA PRINCIPAL)

FIGURA 4.5

se ha comprobado la consistencia de el byte ha transmitirse este debe ser enviado al MPU.

El procesamiento de la información recibida, se fundamenta en el uso de la función en lenguaje Assembly LEER_DATO(). Este procesamiento es mas complejo que el de la información transmitida. En este caso debe decodificarse en los diferentes mensajes MIDI, así, debe determinarse si el byte recibido corresponde a un byte de estado o a un byte de datos. Si se trata de un byte de estado se debe determinar que byte de estado representa, para así procesar o no los bytes de datos que siguen al mensaje de estado. De este byte se deberá determinar además el canal MIDI de transmisión. El programa deberá desglosar los diferentes bytes de datos que siguen a un byte de estado. Debido a la modularidad del lenguaje de programación escogido, se crearán diferentes módulos para la detección de cada posible byte de datos.

En la figura 4.6 se presenta el diagrama de flujo del módulo de reconocimiento de bytes recibidos. Este determina si el byte corresponde a un byte de estado, de datos o de reloj. Si el modulo detecta a un byte de estado, este determina el tipo de byte de estado, el canal de transmisión así como el número de bytes de datos correspondientes, para desglosar a los mismos en sus equivalentes musicales. Esta determinación se realizará con la ayuda de módulos extras para cada tipo de byte.

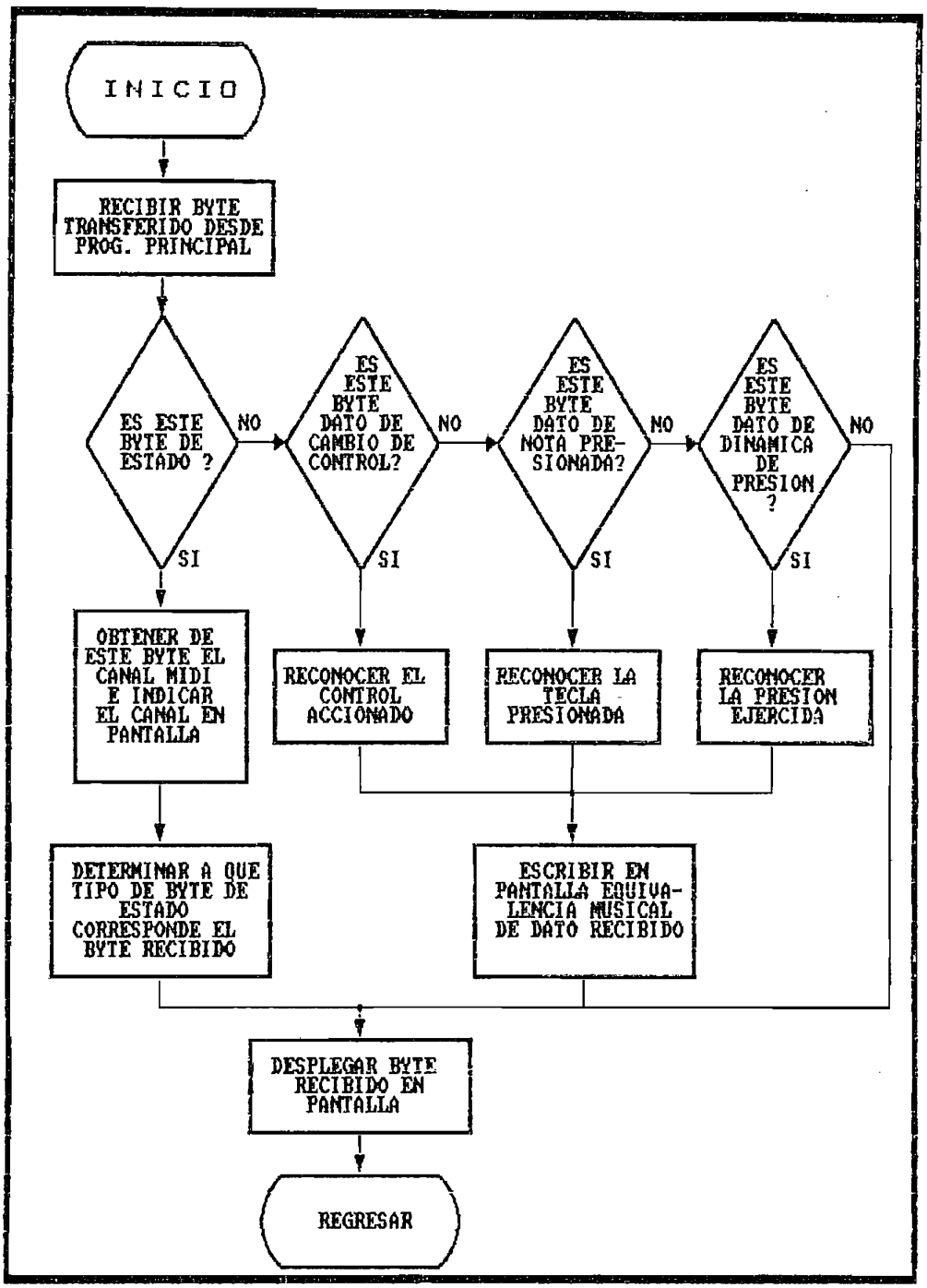


DIAGRAMA DE FLUJO DE MODULO DE DETECCION DE BYTE RECIBIDO

FIGURA 4.6

Por ejemplo, si se trata de un mensaje de tecla presionada, deberá determinar con el segundo byte la tecla presionada y con el tercero la dinámica de fuerza ejercida sobre la misma. Si se trata de un cambio de control se deberá determinar sobre cual de los 128 controladores MIDI se ejerció esta acción.

El programa analizador de datos MIDI, deberá presentar un adecuado interfaz con ayudas al usuario. En este debe presentarse a manera de ventanas, el canal de transmisión, datos transmitidos, datos recibidos y su correspondiente significado musical así como un bloque de mensajes que indiquen el estado de procesamiento del programa en cada instante, además de los posibles errores que se presenten durante el procesamiento del mismo.

El programa analizador deberá cumplir los requerimientos de diseño de software fundamentales, es decir, presentará un código legible y claro, deberá ser portátil y permitir el uso de cualquier monitor de video además de escribirse de la manera mas modular posible.

4.2.3 Diseño del programa Secuenciador

La creación de secuenciadores basados en software es una de las más populares aplicaciones de los sistemas computarizados a los sistemas MIDI de síntesis de sonido. Para afrontar el diseño del programa secuenciador,

es necesario tener conocimiento de la manera como el interfaz MPU-401 recibe y transmite la información MIDI e n tiempo real.

4.2.3.1 Código de Temporización del MPU-401

Debido a que este tipo de aplicación es de tiempo real, la información no solo debe ser recibida y transmitida, sino que cualquier acción de recepción y transmisión debe ser realizada en el instante preciso. El manejo de la temporización de la información MIDI por el programa ha diseñar, resultaría extremadamente complejo si el computador tendría que llevar el control de la misma. Esta situación se facilita enormemente al usar un interfaz inteligente. El MPU-401 incluye el microprocesador ROLAND 6801 CPU, el cual maneja toda la lógica temporizadora, lo cual simplifica la programación de secuenciadores basados en este interfaz.

El MPU-401 tiene un sistema de lógica de codificación de señales de reloj, de tal manera que su reloj interno realiza contajes de 0 a 240 "ticks". El MPU-401, tiene además una salida de audio de metrónomo, la que permite obtener una señal audible. En el modo de inicio esta señal está ajustada a 100 "beats" por minuto. Cada beat del metrónomo corresponde a un cuarto de nota musical. Cada cuarto de nota tiene 120 ticks, de tal manera que cada dos cuartos de nota (dos beeps audibles) el MPU-401 ha

contado desde 0 a 240 ticks, reseteándose automáticamente a 0 ticks.

Este reloj actúa independiente de si el MPU-401 está grabando o no información. Al recibir datos MIDI, el MPU-401 la transmite junto con el código de tiempo correspondiente al instante de la recepción. Este código consta de un byte de tiempo, de tal manera que el computador no tiene que calcular cuando recibió el dato, solamente debe guardar registro del byte de tiempo.

Para que trabaje este sistema de codificación temporal, el MPU-401 envía además bytes que indican al computador el instante que se alcanza los 240 ticks del reloj interno del MPU-401, este mensaje recibe el nombre de TIME OVERFLOW (F8h). También se envía mensajes de final de compás MEASURE END con los bytes temporizadores adecuados, esto permite que el computador guarde registro de todos los controles de tiempo necesarios en el procesamiento de la información MIDI. (6)

Durante la ejecución, el computador debe enviar al MPU-401 los bytes de tiempo y datos en el mismo formato que fueron recibidos. Este puede ejecutar 8 tracks independientemente, para mantener el control de cada track, el MPU-401 requiere contadores de 8 eventos simultáneamente. Se define como evento a un mensaje MIDI. Cada vez que el MPU-401 espera la recepción de un dato en

un track, este envía un byte de TRACK DATA REQUEST (Fnh, donde n = # de track). El proceso de solicitud de datos para cualquier track, se repite hasta que el MPU-401 recibe un mensaje DATA END (FCh), el cual apaga el track respectivo, pero siguen activos los tracks que no han recibido este mensaje aún.

Es importante diferenciar los tracks internos del MPU-401 con los canales de transmisión MIDI. Los tracks tienen significado solo para lógica interna del MPU-401, pueden ser considerados como el equivalente digital de los tracks de una grabadora multitrack analógica. Los canales MIDI son convensiones del protocolo de comunicación para que se puedan transmitir multiples sonidos de instrumentos por un mismo cable. Para lograr mantener registro de cada track es conveniente asignar un canal MIDI con un instrumento a un track del MPU-401, esto es útil siempre y cuando se use no más de 8 tracks. Para un número mayor de tracks e instrumentos se debe permitir que más de un canal MIDI coexista en un track del MPU-401.

Es importante señalar que para tener un idea más clara de estos mensajes de temporización se ha desarrollado una sencilla subrutina escrita en lenguaje C, y que usa las subrutinas de comunicación escritas en Assembly. El módulo de procesamiento de este programa es muy similar al presentado en el programa analizador, ya que la etapa de recepción de datos es la misma.

recibidos, esto permite que se indiquen en pantalla estos, comandos.

4.2.3.2 ESTRUCTURA DEL SECUENCIADOR

De manera similar al programa ANALIZER, el secuenciador basado en software que desarrollaremos consta de varios módulos cada uno de los cuales ejecutará una función especifica.

Una de las características más importantes de un programa de esta naturaleza es la presentación de un interfaz adecuado al usuario. Para nuestro caso se desea desarrollar un interfaz de tal manera que semeje las funciones de una grabadora analógica. Las opciones del programa se presentaran a manera de menú desplegable de tal manera que la opción ha escoger se presente en video reverso.

Las funciones que presenta el MPU-401 son varias, de estas el programa ha desarrollar escoge las funciones de temporización del metrónomo. Estas son el número de beats por minuto, el compás musical y la activación o desactivación del metrónomo. Debido a que estas funciones involucran el ingreso de valores, el programa debe prever un mecanismo de detección de consistencia de los valores ingresados, de tal manera que estos sean enviados al MPU-401 e indicados en pantalla sólo

después del chequeo de consistencia respectivo, esto para reducir las posibilidades de error al mínimo.

Las opciones más importantes de este programa son las de grabación y ejecución de la información MIDI. El secuenciador prototipo permitirá registrar un track polifónico de información MIDI. El registro debe ser diseñado de tal manera que puedan ser detenido por el usuario en cualquier instante. El almacenamiento de la información recibida en memoria debe de ser en paquetes de bytes donde cada paquete es un evento MIDI de longitud variable, esto se debe realizar debido a que para manejar más de un track del MPU-401 se tiene que enviar un evento a cada track, en tanto que si la información se almacena sin guardar registro de cada evento, el procesamiento para desarrollar un secuenciador multitrack es más complicado.

La ejecución de la información registrada debe ser temporizada con la ayuda de las funciones del metrónomo. El programa deberá permitir además el listado de los archivos de secuencias registradas con el secuenciador. Es importante que el programa presente también una ventana de estado donde se reporte al usuario el estado de procesamiento actual y errores del mismo. Para esto se debe prever todos los posibles errores que pueden presentarse al leer la información del MPU-401 y durante el proceso de manejo de memoria del computador.

En la figura 4.7 se presenta el diagrama de flujo del programa principal. En este se destaca la naturaleza modular del programa secuenciador, donde cada función será implementada con un módulo independiente.

Un factor importante que debe considerarse al implementar el programa es la posibilidad de que se presente un choque de información entre el computador y el MPU-401, es decir el caso en el cual el computador esté enviando un comando al MPU-401 y al mismo tiempo esté enviando un mensaje de DATA REQUEST. En esta situación se debe prever que el comando enviado no se pierda en el procesamiento. Este problema se elimina usando el código ACKNOWLEDGE del MPU-401, de tal manera que si el computador envía un comando, el secuenciador debe verificar primero el código ACK y si este no aparece, se debe almacenar los datos recibidos en un buffer temporal hasta que se reciba el mensaje ACK y solo entonces enviar los datos correspondientes al comando transmitido por el computador.

En la figura 4.8, se presenta el diagrama de flujo correspondiente a la función de grabación. En este se observa el proceso que debe seguirse para enviar a memoria la información codificada por eventos MIDI. Para esto, se debe realizar comparaciones entre los datos recibidos y los diferentes mensajes MIDI y así determinar el número de bytes de cada evento y no tener un número fijo de bytes para cada uno, lo que usaría memoria inútilmente.

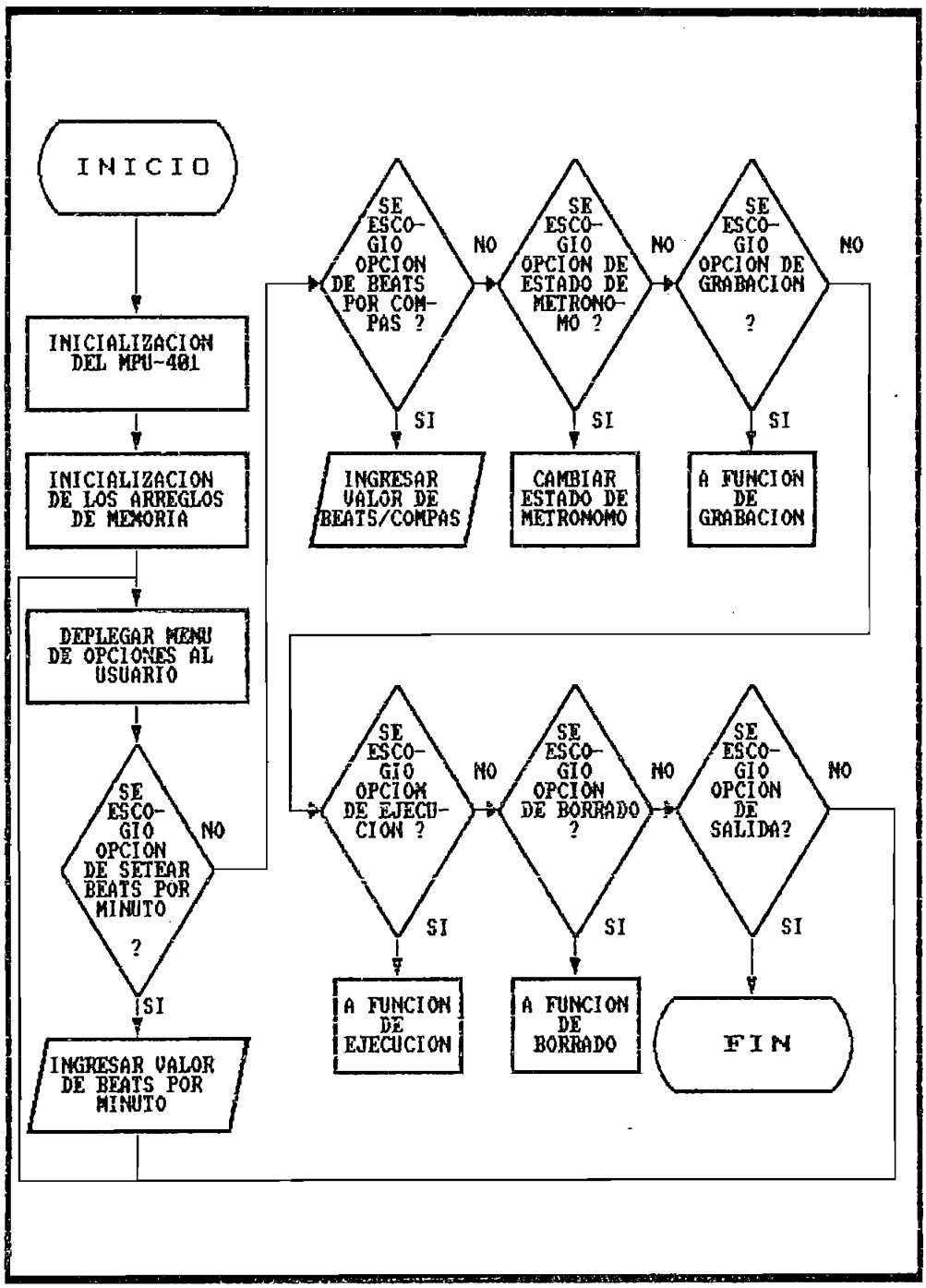


DIAGRAMA FLUJO DEL MODULO PRINCIPAL DEL PROGRAMA SECUENCIADOR

FIGURA 4.7

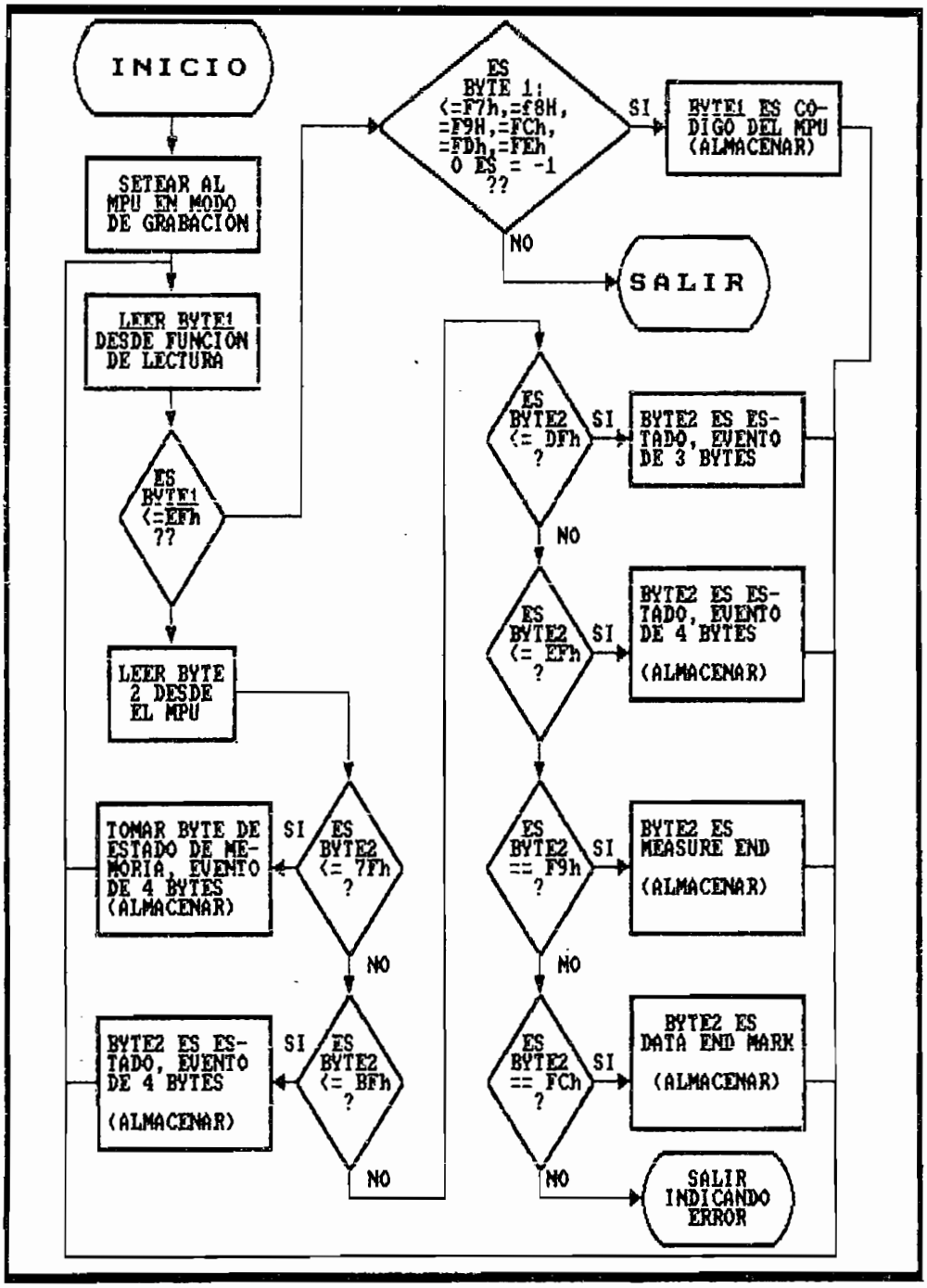


DIAGRAMA DE FLUJO DE LA FUNCION DE GRABACION DEL PROGRAMA SECUENCIADOR

FIGURA 4.8

4.3 DESCRIPCION DE SUBRUTINAS

4.3.1 Descripción de Subrutinas de Comunicación

En el numeral anterior hemos diseñado las tres subrutinas correspondientes a los drivers de comunicación. La implementación de estas subrutinas las realizaremos en lenguaje Assembly de la familia de microprocesadores 8086.

El programa de comunicación se inicia con asignaciones que corresponden a los pórticos que el MPU usa para su comunicación: Pórtico de datos (330H) y pórtico de estado (331H). Se asignan además el código correspondiente al comando Data Receive Ready (DRR), este valor es útil para verificar en el pórtico de estado un byte con su bit # 6 igual a 0 lógico. La siguiente asignación es útil para verificar en el pórtico de estado el código ACK. Finalmente, se asigna un valor de 256, el mismo que sirve para realizar los lazos de tiempo para verificar un byte en el pórtico de estado del MPU.

El programa define además como públicas a las tres subrutinas en Assembly, de tal manera que sean reconocidas por los módulos en lenguaje C. (7)

Cuando un módulo de programa escrito en lenguaje de medio nivel, entrega el control a un módulo

escrito en lenguaje de máquina, es necesario antes de realizar cualquier procesamiento, almacenar en el stack varios registros del micropocesador para ser recuperados el instante que el módulo en Assembly devuelve el control al módulo en lenguaje C. (8)

A continuación dentro del segmento de código se encuentran las tres subrutinas de comunicación Estas en lenguaje Assembly se conocen como procesos. Cada subrutina se encuentra limitada por las directivas: PROC y ENDP y termina con la directiva RET que transfiere el control al punto del programa que la llamó. (7)

La subrutina _COMANDO, inicia guardando en el stack los registros índices. Posteriormente se lee el byte presente en el pórtico de estado. Con la instrucción TEST AD DRR, se prueba si el bit # 6 de este byte es 0 lógico, si esto es verdad prosigue a enviar el comando al MPU. Si el bit # 6 es 1 lógico, reintenta leer el byte en el pórtico de estado. Este lazo lo puede realizar hasta por 256 veces, si en este lazo no encuentra el mensaje DRR, la subrutina termina regresando el código de error: -1.

Para enviar el comando al MPU, la subrutina recupera el valor a enviarse desde el stack, ya que cuando una subrutina en Assembly es llamada desde un módulo en C, el argumento que se transfiere desde el lenguaje C al Assembly, se almacena en la dirección dada por el puntero

del stack. Con el uso de la instrucción MOV DX,AL se transmite al pórtico de estado el byte menos significativo de AX, es decir el comando del MPU. Una vez recibido el comando el MPU debe devolver un mensaje ACK por el pórtico de datos, previo al cual debe presentar un mensaje DSR en el pórtico de estado. Para verificar la presencia del mensaje DSR (bit # 7 en bajo), usamos la instrucción de rotación ROL AL,1, insertando un 1 lógico en el bit menos significativo y enviando el bit más significativo al carry. Al comprobar el bit de carry se determina si corresponde o no al mensaje DSR. En caso negativo el programa intenta por 256 veces reconocer el mensaje DRR, si el lazo termina y no reconoce este mensaje la subrutina devuelve el control al lenguaje C regresando el código de error: -1. Si se establece que el MPU tiene dato en el pórtico de datos, el programa toma este valor y verifica si corresponde o no al mensaje ACK. Si se trata de este mensaje, la subrutina devuelve por medio del registro AL el valor ACK, en caso contrario devuelve el código de error: -1.

Las subrutinas _ESCRIBIR_DATO y _LEER_DATO se implementan de manera similar a la subrutina anterior, la diferencia radica en que en transferencia de datos, el MPU no responde con ningún comando, por tanto no se verifica el código ACK.

El listado del programa de las subrutinas de comunicación se encuentra en el Anexo Nº 3.

4.3.2 Descripción de Funciones del Programa Analizador de Datos MIDI

La implementación de las diferentes funciones de este programa se realizarán en lenguaje C. Debido a la naturaleza modular de este lenguaje de programación, se ha diseñado al programa basándose en diferentes funciones cada una de las cuales realiza un procesamiento diferente de la información MIDI.

Se diseña inicialmente un programa principal, el cual controla el procesamiento general, llamando a las diferentes funciones del mismo. El flujo de procesamiento se implementa basándose en el diagrama de flujo de la figura 4.6.

4.3.2.1 Descripción del Programa Principal

El programa se inicia incluyendo dos headers diseñados para este programa estos son: notas.h y control.h. Estos contienen información referente a los nombres de las teclas de un sintetizador MIDI de 5 octavas y de los 128 controladores que define el protocolo MIDI.

Posteriormente se realizan varias asignaciones de valores útiles. Estos son: UART (3FH), este sirve para asignar al MPU al modo UART (Universal Asynchronous Receive and Transmit), el cual apaga el modo

inteligente del MPU y deja pasar los datos sin alteración desde y hacia el sintetizador. El otro comando que se asigna es el correspondiente a SYSRESET (FFH), este apaga el modo UART e inicializa al MPU en su modo inteligente normal. Otra asignación corresponde a crear localidades temporales de memoria (BUFFER = 1000), las que se usan para almacenar los bytes recibidos durante el tiempo que el computador realiza el procesamiento de los mismos. (6)

Posteriormente se realizan las declaraciones de variables del módulo principal. Entre estas se incluyen dos variables para almacenar el número de bytes requeridos para almacenar las imágenes en pantalla. Se declara también un puntero (*m), el cual se usará para la lectura de los bytes a transmitir al MPU. El interfaz que el programa presenta al usuario se encuentra realizado en modo gráfico, para esto se autodetecta la tarjeta gráfica y valores máximos de coordenadas. Para escribir caracteres y variables formateadas en modo gráfico se desarrolla una función a la que se le denomina OUTFMTTEXT(). (8) Antes de ingresar al lazo de procesamiento de la información, se setea al MPU en modo UART y se asigna el modo de recepción (recepción()).

El procesamiento de la información se realiza dentro de un lazo que es interrumpible solo por el usuario. Este lazo incluye dentro del mismo dos lazos condicionales. Estos son: un lazo para lectura de datos de

número de caracteres transformados, de tal manera que si su número es mayor a 2, se indica al usuario con un mensaje de error. Estas funciones basan su funcionamiento en la transformación de los caracteres ASCII que representan números exadecimales a valores numéricos mediante la adición y substracción de valores fijos.

El programa principal incluye además dos funciones RECEPCION() y TRANSMISION(), las que sirven para indicar en video reverso el estado de procesamiento del programa.

En el listado del Anexo Nº4 se presenta el código correspondiente al módulo principal del programa analizador.

4.3.2.2 Descripción del Módulo de Reconocimiento de bytes recibidos

Este módulo es usado por el programa principal para decodificar los datos MIDI recibidos desde el MPU. En cada llamada al módulo reconocer(), el programa principal transfiere como argumento al byte recibido desde el MPU.

La declaración de variables asigna el valor 0 a las variables enteras (n, p, c), estas son útiles para decodificar los segundos y terceros bytes de los mensajes

MIDI de: nota presionada, presión ejercida y cambio de control.

Para discriminar entre un byte de estado y un byte de datos se usan las variables enteras: n, p, c, de tal manera que si estas tienen un valor diferente de cero, el dato recibido corresponde a un byte de datos. Debido a que un byte de datos sigue siempre a un byte de estado, al determinarse un byte de estado este asigna el valor 1, ya sea a la variable n, p ó c dependiendo del tipo de byte de estado que se trate. Estos saltos condicionales se realizan verificando el valor en las variables anteriores. Si no se cumplen las condiciones anteriores la función acepta como byte de estado al byte recibido.

Este byte de estado contiene información acerca del canal de transmisión en el nibble menos significativo y del tipo de datos que seguirán en el nibble más significativo. Para decodificar estos valores asignamos a la variable canal el producto lógico AND entre el byte recibido y el valor 0FH. En tanto que para determinar el tipo de byte de estado se asigna a la variable 'aux' el producto lógico AND entre el byte recibido y el valor F0H.

Para determinar el tipo de byte de estado realizamos lazos condicionales para todos los posibles tipos de mensajes. Si se cumple una de estas condiciones y

si el mensaje requiere más de un byte; dentro del lazo condicional respectivo se asigna el valor 1 a la variable n, p ó c, dependiendo de si se trate mensaje de nota presionada, presión ejercida o cambio de control.

Una vez que se determina el tipo de byte de estado, se despliega la información en pantalla tanto del tipo de byte como del canal de recepción. Debido a que con la decodificación del byte de estado el programa sabe de antemano el byte que será recibido posteriormente, el próximo byte es enrutado ya sea a los módulos de detección de nota presionada TONAL(), presión ejercida PRESION() o controlador accionado DET_CONT().

El módulo de detección de nota presionada TONAL(), realiza un salto condicional comparando el byte recibido con los valores que contiene el header NOTAS.H. En este constan en dos arreglos, los códigos y nombres de las diferentes teclas de un sintetizador MIDI estándar (5 octavas).

El módulo de detección de dinámica de presión ejercida al accionar las teclas del sintetizador PRESION(), se basa en varios saltos condicionales, comparando el byte recibido con los diferentes valores que asigna la carta de dinámica de velocidad del protocolo MIDI. Cuando se ha determinado la presión ejercida en términos musicales, este equivalente se lo representa en

pantalla.

Para determinar el equivalente musical de un byte de dato de cambio de control, se realiza comparaciones entre el byte recibido y los valores que se encuentran en los arreglos que contiene el header CONTROL.H. En este se encuentran adecuadamente asignados en dos arreglos los códigos y nombres de los 128 controladores que asigna el protocolo MIDI en la carta de cambios de control.

Una vez que el módulo DETECTAR(), realiza la detección de todos los tipos de datos, se asigna el valor cero a las variables: n, p y c, esto sirve para indicar que el siguiente byte a recibir ya no corresponde a un byte de datos y se espera un byte de estado.

El proceso de detección incluye también a los mensajes de reloj MIDI, pero en este caso no se realiza la detección del canal de transmisión. Esto se debe a que estos mensajes se transmiten sin designación de canal.

Para desplegar los bytes recibidos y transmitidos en pantalla, se requiere desarrollar dos funciones para presentar en el interfaz al usuario ventanas de datos transmitidos y recibidos. Estas funciones reciben el nombre de DESPLEGAR() y DATA_TRANS() y crean dos ventanas de 12 columnas de bytes por 4 filas. Estas ventanas se borran cada vez que se llenan con 48 bytes.

En el listado del Anexo Nº 5 se presentan los códigos de las diferentes funciones que incluye el módulo de detección.

4.3.3 Descripción de Subrutinas del Programa Secuenciador

El programa secuenciador basa su funcionamiento en una serie de módulos independientes donde cada uno realiza una función específica. El interfaz que el programa presenta al usuario se encuentra desarrollado en modo de texto, a diferencia del programa analizador donde el interfaz se lo implementó en modo gráfico.

4.3.3.1 Subrutinas en Lenguaje de Máquina

Para desarrollar un interfaz que presente un menú desplegable con la opción a escoger en video reverso, es necesario desarrollar previamente una serie de funciones escritas en lenguaje de máquina, las mismas que al ser ensambladas se comportan como cualquier otra función escrita en lenguaje C.

En el Anexo Nº 6, se presenta los listados de estas subrutinas, las mismas que se encuentran escritas en lenguaje Assembly. Este programa (BIOS.ASM), realiza inicialmente el manejo de memoria de manera idéntica al programa MPUCOM.ASM. El funcionamiento se basa en llamar a

las diferentes funciones de servicios de interrupciones que permite el BIOS del computador IBM-PC. Así para implementar cada función se debe almacenar en los registros respectivos los valores adecuados: AH (número de servicio del BIOS), DH, DL (coordenadas de pantalla), BH (página de memoria de video). De esta manera al efectuar la función : INT BIOS 10, se ejecuta los comandos con los valores que se encuentren almacenados en los registros. Las funciones implementadas en Assembly son:

home(): coloca el cursor en las coordenadas 0,0 de la pantalla de texto.

writchr(c,n): escribe el carácter "c", "n" veces en la posición actual del cursor sin realizar cambio de atributo en la escritura del carácter.

writbw(c,n): escribe el carácter "c" con atributo de video normal, "n" veces en la posición actual el cursor.

writul(c,n): escribe el carácter "c" subrayado, "n" veces en la posición actual del cursor.

writrv(c,n): escribe el carácter "c" en video reverso, "n" veces en la posición actual del cursor.

writblr(c,n): escribe el carácter "c" en video reverso

titilante, "n" veces en la posición actual del cursor.

4.3.3.2 Manejo de Imágenes en Pantalla

En el programa secuenciador se presenta un sistema diferente de manejo de imágenes en pantalla. Generalmente la imagen del interfaz al usuario, se encuentra implementada dentro del programa principal, esta alternativa no es la más adecuada debido a que el proceso de compilación es más largo, además de que la función PRINTF del lenguaje C, no permite el manejo de caracteres gráficos. En tanto que si la imagen se almacena en un archivo separado, es posible crear directamente imágenes con un editor de texto del tipo WYSIWYG (What you see is what you get), lo que permite usar los caracteres gráficos directamente.

La imagen creada puede ser almacenada directamente en memoria como una matriz de 25 x 80. Este método usaría memoria inútilmente, ya que no toda la pantalla estaría con caracteres. Para usar la memoria indispensable se requiere, un método que permita accesar solo la cantidad de memoria exacta por cada línea de pantalla. Esto se puede realizar usando el método de definición dinámica de memoria. En lugar de definir un arreglo fijo de memoria durante la compilación, el método de definición dinámica de memoria espera que el programa se

ejecute para definir solo la memoria necesaria.

El proceso anterior, se realiza con el uso de las funciones: 'malloc(sizeof())'. Estas definen el espacio de memoria necesaria para cada línea de la imagen, mientras esta es leída desde su archivo. Para mantener registro de cada línea se usa un método de programación conocido como "listas enlazadas", en este, se define las líneas en secuencia desde la primera hasta la última. Así, se crea una cadena donde sus lazos son punteros que contienen las direcciones de memoria donde se almacenaran los caracteres de la siguiente línea de la imagen. Esto escrito como estructura en lenguaje C, es:

```
struct cadena {
    char*línea;
    struct cadena *próximo;};
```

El primer puntero, apunta a las direcciones de la línea actual que se almacena, en tanto que el siguiente puntero apunta a las direcciones en que se almacenará la siguiente línea. Al llegar a la última línea, el nodo final es asignado a NULL, lo que puede ser usado para finalizar la definición de memoria.

Para implementar este método de lectura de imágenes, se implementa un programa en lenguaje C, (IMAGEN.C), el mismo que permite leer cualquier archivo y presentarlo en pantalla, definiendo solo la memoria estrictamente necesaria. Para implementar el código de

estos programas de una manera clara, se ha escrito un Header (STANDARD.H), el cual se encuentra listado en el Anexo Nº 6. En este constan varias definiciones correspondientes a los códigos ASCII del IBM-PC.

El programa IMAGEN.C requiere la definición de las estructuras que permiten el manejo de memoria para almacenar las imágenes a ser leídas. Esto se realiza en el Header IMAGEN.H. La primera función del programa IMAGEN.C, es INPCHAIN(), la cual lee el contenido de un archivo y lo almacena en memoria, para esto se debe indicar el nombre del archivo y la longitud de la línea de máxima longitud. Esta función inicialmente define un buffer temporal de memoria, de longitud suficiente para almacenar la cadena de caracteres más larga. Posteriormente se lee cada línea desde el archivo abierto, hacia el buffer temporal, para esto se usa la función: fgets(). Así cada línea de caracteres es añadida al final de la cadena. Al llegar al final del archivo se indica guardando en el puntero de la próxima línea un carácter nulo: NULL. Esta función, cierra el archivo abierto y libera la memoria que se ha usado en el procesamiento.

La definición de memoria es realizada con la funciones: CHAINALLOC() y STRSAVE(). Estas permiten definir y almacenar dos punteros durante cada nodo, uno de ellos apuntando a la siguiente línea y el otro a la línea actual. Para presentar la imagen en pantalla se implementa

la función: DISPCHAIN(), la cual salta de nodo a nodo, imprimiendo las cadenas de caracteres de cada línea, esta se detiene cuando encuentra un puntero NULL al final del archivo. La función DECHAIN(), libera la memoria que ha sido usada por la lectura del archivo. El listado correspondiente a este módulo se incluye en el Anexo Nº 6.

4.3.3.3 Implementación de Menús en Video Reverso

Una vez que se tiene en pantalla el menú primario, se desea escoger las diferentes opciones con el uso de las teclas de cursor y <RET>. Para esto se debe escribir el bloque a seleccionar en video reverso. Con el uso de las funciones en Assembly podemos escribir solo un carácter con cualquier atributo y en cualquier posición. Para escribir más de un carácter, se ha desarrollado una subrutina, la que recibe como argumentos: el atributo del carácter a escribir, el carácter a escribir y las coordenadas del carácter a escribir. A esta función se la ha denominado: WRITWORD(), la misma que está incluida dentro del programa: MENU.C.

Para crear un menú de opciones se ha desarrollado la función OPCION(). Esta permite un manejo lógico de las teclas de cursor sobre el menú implementado. La lógica de selección podría realizarse con una serie de funciones IF(), las que tendrían que considerar absolutamente todas las posibilidades en los movimientos de

las teclas de cursor. Sin embargo, este método de selección, en el caso de un menú de varias opciones, utilizaría decenas de sentencias IF(). En lugar de este método se desarrolla un sistema que generaliza el movimiento del cursor y reduce el código necesario para implementar este sistema de menú.

Se crea una estructura del siguiente tipo:

```
struct menu{
    int xpos, int ypos,
    char content[15];
    int nup; int ndown; int nleft; int nright;
    int key;};
```

En esta, los dos primeros elementos, son las posiciones x e y donde se localizará el bloque a escoger. El tercer elemento es una cadena de caracteres donde se almacena el mensaje que aparecerá en el bloque a escoger. De esta manera el programa MENU.C, sabe donde y que escribir en video reverso y en video normal. Los siguientes cuatro elementos de la estructura menú, corresponden al número de bloque al cual debe ir el cursor, dado que se presione la tecla de cursor hacia arriba, abajo, izquierda y derecha respectivamente. Esta sistema presenta un método compacto de almacenar la información correspondiente al menú a realizar. El módulo principal del programa MENU.C es la función OPCION(). Esta recibe tres argumentos de entrada. El primer argumento corresponde al nombre del arreglo de estructuras de tipo menú que guarda el menú a implementar.

El segundo y tercer argumento corresponde a los valores que indican la primera y última opción del menú que se debe escribir en video reverso. La función OPCION(), llama entonces a la función WRITWORD(), para escribir en video reverso el primer elemento del menú, los argumentos que se envía a la función writword(), se toman del elemento correspondiente del arreglo de estructuras del menú a desarrollar. Posteriormente se incluye un lazo infinito que permite procesar las acciones del usuario. En el caso de que exista acción del usuario sobre las teclas de cursor, la nueva posición del cursor del menú se determina a partir del arreglo de estructuras del menú, variando el valor de la variable "k". Al final del lazo, se escribe el bloque anterior con video normal, en tanto que el nuevo bloque es escrito en video reverso. Luego de presionar <RET>, esta función regresa el valor "k", el cual corresponde a la opción que se encuentra en video reverso. Si el usuario toma acción sobre las teclas (?) ó (ESC), el programa regresa los códigos -2 y -3 respectivamente, en tanto que si se presiona una tecla que no corresponde a las mencionadas, la función regresa el código de error -1.

La función MENU.C, incluye además un módulo (WRITERR()), el que permite escribir mensajes en cualquier posición de pantalla, de tal manera que el mensaje se borra el instante que existe acción del usuario. Esta función es muy útil para indicar al usuario cualquier error que se produce durante el procesamiento del programa.

El listado correspondiente a los módulos de las función MENU.C se encuentran en el Anexo Nº 6.

El programa secuenciador, presenta entre sus opciones la posibilidad de ingresar valores enteros que controlarán los parámetros de operación del MPU-401. Antes de transmitirse, se debe realizar un chequeo de consistencia de los valores ingresados. Para lograr este objetivo, se implementa una función de ingreso de datos enteros y chequeo de consistencia (INPUT.C). Esta función recibe como argumentos de entrada: el número de línea donde se imprimirá el mensaje de lectura del dato, el mensaje a indicar, la variable en la que se almacenará este valor y los valores máximo y mínimo que puede tomar la variable a cambiar. Si el usuario no ingresa un valor y presiona <RET>, el programa regresa el código 1 y la variable a leer sin cambio. Si se ingresa un valor, se chequea consistencia y si cumple esta condición, se asigna el valor a la variable respectiva y se regresa el código 1. Si la prueba de consistencia no es satisfecha, se indica al usuario un mensaje de error y solicita el valor nuevamente.

El listado de este programa se encuentra en el Anexo Nº 6.

4.3.3.4 Módulo Principal del Programa Secuenciador

Uno de los aspectos más importantes a ser

tomado en cuenta en la implementación del secuenciador, es el lograr un adecuado manejo de la memoria que será usada para almacenar la información MIDI recibida. No basta crear un arreglo de bytes, donde se almacenen los bytes que se reciban, sino que estos deben almacenarse guardando registro del inicio y final de cada evento MIDI. Esto se debe a que si bien el secuenciador que implementaremos, usa solo un track del MPU-401, la estructura de datos debe hacerse de manera flexible, tal que sea posible la implementación de un secuenciador multitrack, a partir del secuenciador prototipo que diseñaremos.

Cada evento MIDI, tiene una longitud entre 1 y 4 bytes, el problema radica en que antes de almacenar la información se debe determinar de que longitud será el evento a ser recibido. Esto se logra realizando una decodificación de los datos recibidos, para definir así, antes de almacenarse la longitud de cada evento, dato que también será almacenado en memoria.

El interfaz MPU-401, tiene muchos comandos. Para referirnos a los mismos de una manera clara y no por sus equivalentes exadecimales, se ha escrito un header, el cual presenta las definiciones de los comandos del MPU-401 así como una concisa definición de los mismos. A este header se lo ha llamado MPU401.H y se encuentra listado en el Anexo Nº 6.

El módulo principal del programa secuenciador se inicia definiendo un arreglo de estructuras para crear la presentación del menú del programa. Como se explicó anteriormente, en este arreglo consta toda la información referente a las posiciones y movimientos que debe presentar el menú. Para el funcionamiento de este sistema de lectura de imagen, se debe escribir en otro archivo la imagen principal del programa, esta se presenta el archivo SEC.SCR, cuyo contenido es el siguiente:

```
   ESCUELA POLITECNICA NACIONAL
   FACULTAD DE INGENIERIA ELECTRICA
   DEPARTAMENTO DE ELECTRONICA Y CONTROL
   TESIS DE GRADO: ADRIAN NARANJO PUENTE
   DIRECTOR:       DR. HUGO BANDA GAMBOA
   PROGRAMA: SECUENCIADOR MIDI DE UN CANAL
             PARA EL INTERFAZ MPU-401 ROLAND
```

BPM	Ajustar el numero de beats del metrónomo	:	
TIEMPO	Ajustar el # de beats por compás.	:	/4
METRONOMO	Apagar / encender el metrónomo	:	
GRABAR	Inicio de grabación		
EJECUTAR	Inicio de ejecución		
LISTAR	Lista archivos .MID		
SALIR	Regresar al DOS		

La definición de las variables del programa incluye los valores iniciales del sistema de reloj del MPU-401, así los punteros *metrate y *meter se inicializan con contenidos de 120 y 4 respectivamente. El programa inicializa al MPU-401, enviando el comando RESET. El archivo de imagen es leído con la función INPCHAIN(), en la cual se indica el archivo de imagen a leer y la máxima

longitud de las líneas del mismo. Antes de procesar el menú del programa se envían al MPU-401 los códigos de inicialización del metrónomo.

En este punto se incluye un lazo infinito, dentro del cual se presentarán en pantalla el estado del reloj del sistema así como el manejo del menú y la selección de las diferentes opciones por parte del usuario. Para presentar la imagen del menú se usa la función DISPCHAIN(). Los valores de inicialización del reloj MIDI, deben ser enviados al menú, cada vez que se presente una variación en los mismos. Estos valores deben enviarse al menú como caracteres, para convertirlos desde sus valores enteros se usa la función "itoa()". El metrónomo se activa o desactiva dependiendo del estado de la variable "meton". El escogitamiento de la opción de menú se realiza con el uso de la función OPCION(), la cual regresa el código correspondiente a la opción de menú escogida, y con el uso de la sentencia "switch()", es posible direccionar a efectuar el procesamiento escogido. Si se ha escogido una opción que involucra el ingreso de datos del usuario (case (0) y case (1)), se usa la función INPUT(), la cual regresa el código 1, si se ingresó un nuevo valor, en este caso el programa procede a transmitir el comando con su dato respectivo, entrando nuevamente a determinar otra acción del usuario. El estado del metrónomo, se varía de manera que si este está encendido, se apaga y viceversa. (case(2)).

Las opciones del secuenciador se encuentran implementadas en funciones independientes: GRABAR(), EJECUTAR() y LISTAR().

En el caso de que la función OPCION(), regrese códigos de acciones sobre la tecla <ESC> o opción salir, el programa termina, devolviendo el control al sistema operativo.

El programa SEC.C, incluye los módulos necesarios para grabar y ejecutar la información recibida. La primera función que se implementa corresponde a GETNEXT(), esta función es útil en el caso de que se produzca un choque de información entre el computador y el MPU-401, así, se define un buffer temporal de comandos, de tal manera que si existe dato en el buffer, la función regresa el dato almacenado, si no existe dato en el buffer de comandos, la función regresa un dato leído directamente desde el MPU-401. La función que almacena los datos en colisión en el buffer temporal de comandos es UNGETNEXT(). La función GET401() permite el manejo del caso en el que el programa no recibe respuesta del MPU-401. Esta función incluye dentro de un lazo infinito a la función GETNEXT(), de tal manera que ésta termina si se lee un byte o si el usuario presiona una tecla, regresando ya sea el byte leído o el código de error (-1) respectivamente.

La función de grabación GRABAR(), se basa en

el diagrama de flujo de la figura 4.8. Con el uso de la función COMANDO(START_REC), se inicializa al MPU-401 en el modo de grabación. El lazo de grabación de eventos es infinito, el mismo que termina solo si existe acción por parte del usuario. La lógica para determinar la longitud de cada evento es la siguiente: con la función GET401(), se lee el primer dato transmitido por el MPU-401, dependiendo del valor de este byte se toman dos alternativas. Si el byte leído es menor que EFh (239), significa que corresponde a un byte de tiempo (ningún dato MIDI es menor a EFh). En este caso se debe leer el byte siguiente, dependiendo de el valor del segundo byte se determina el número de bytes que tendrá el mensaje. Así, si el segundo byte es menor que 7Fh, significa que este byte es de datos. En este caso se debe recuperar el byte de estado de memoria, ya que el MPU-401 no transmite en cada mensaje el byte de estado, este se transmite solo si el mensaje a cambiado de byte de estado. El byte de estado ausente en el mensaje, corresponde al byte almacenado en la variable "estado", en la cual se almacena todo byte de estado que registre la función. Para este caso se debe tomar el tercer dato con GET401(), y se almacenan en memoria con la función GUARDAR(), de la siguiente manera: inicialmente se almacena el byte 11h, el cual corresponde a un indicador de que las secuencias registradas con este programa corresponden al estándar MIDI 0.1, posteriormente se almacena el número de bytes del evento y los bytes correspondientes al mismo. La comparación del segundo byte

con los diferentes mensajes MIDI, se usa para determinar la longitud de todos los posibles eventos.

En el caso de que el primer byte leído no corresponda a un valor menor a EFh, se deduce que este byte corresponde a un código del MPU-401, por lo tanto el evento corresponde solo a un byte, siendo almacenado solo un byte por eventos con el uso de la función GUARDAR(). En el caso de que no se cumpla las condiciones indicada, el proceso de grabación se detiene.

Este método de decodificación de la información MIDI, es posible gracias a la manera como ha sido definido el protocolo. Así, los bytes más comunes tienen siempre valores menores, lo que permite usar una sencilla lógica de decodificación de los mismos.

La función GUARDAR(), recibe como argumento los datos de cada evento, definiendo un archivo cuyo nombre es ingresado por el usuario. A este archivo la función le añade la extensión .MID para su respectivo almacenamiento. Cuando termina la secuencia, asigna 4 bytes de valor 00h, lo que es útil para determinar el fin de una secuencia. Este sistema de almacenamiento de información, es flexible ya que el manejo de eventos permite asignarlos a diferentes tracks, con lo que se facilita la implementación de un sistema multitrack.

Otra de las opciones del secuenciador, es la de ejecución. Este proceso es implementado en la función EJECUTAR(). Esta función abre el archivo que indique el usuario, ejecutando la secuencia solo si el primer byte corresponde a 11h, de lo contrario indica un mensaje de error. Comparada con la función de grabación, la función de ejecución es bastante simple. Se inicia por activar el modo de ejecución, borrando los contadores de programa e iniciando el modo de ejecución (CLEAR-PCOUNT y START_PLAY). Se incluye un lazo infinito el mismo que termina en tres circunstancias: acción del usuario, término de la cantidad de datos de la secuencia o sí el MPU envía un mensaje de ALL_END. Cualquiera de las situaciones mencionadas transmite al MPU-401 el comando STOP_PLAY y CLEAR_MAP, lo que borra el mapa de ejecución. El envío de eventos al MPU-401, se realiza cada vez que este transmite al computador un código REQ_T1 a REQ_T8, los bytes correspondiente a los mensajes MIDI son transmitidos al MPU-401 decodificando los valores de la secuencia.

La opción de listado de las secuencias almacenadas en disco, se la realiza con la función LIST(), la que permite listar todos los archivos con la extensión .MID, que se encuentren en el mismo directorio desde donde se ejecuta el programa principal.

El listado del este módulo principal y de sus respectivos módulos se presenta en el Anexo Nº 7.

C A P I T U L O V

ANALISIS DE RESULTADOS

ANALISIS DE RESULTADOS

5.1 PRUEBAS

Los programas implementados en la presente Tesis, han sido diseñados de manera tal que su funcionamiento sea lo más portable posible. Esto es posible debido a que la creación del interfaz al usuario en los programas analizador y secuenciador, se basan en funciones genéricas que son independientes del tipo de tarjeta gráfica usada.

Además, debido a que las subrutinas de comunicación están escritas directamente en lenguaje de máquina de la familia de microprocesadores 8086, la transmisión y recepción de los diferentes datos MIDI es compatible con cualquier computador IBM PC.

Es importante señalar que el procesamiento de la información MIDI recibida, usa funciones que permiten la decodificación de absolutamente todos los mensajes del protocolo MIDI 0.i. Esto hace posible el uso de los programas con cualquier sintetizador de sonido compatible con este estándar.

Es así que el programa analizador, si bien ha sido desarrollado basándose en los mensajes MIDI transmitidos por el sintetizador HT-700 de CASIO, ha sido

probado de manera completamente satisfactoria con el sintetizador DX-7 de YAMAHA, el mismo que presenta varias diferencias en la transmisión de datos de nota liberada y pitch bender.

De igual manera, el secuenciador basado en software implementado, ha sido ejecutado tanto en el sintetizador HT-700 de CASIO y DX-7 de YAMAHA, comprobándose que el registro y reproducción de las secuencias de información MIDI es exactamente igual para los dos sintetizadores.

5.2 CONCLUSIONES

Al finalizar el desarrollo de la presente Tesis, se presentan las siguientes conclusiones:

- Hemos revisado, el largo camino por el cual, desde sus orígenes hasta los más recientes desarrollos tecnológicos, ha transitado el ingenio del ser humano, para así, satisfacer sus necesidades de creación de sonidos, de esta forma verificamos que el desarrollo tecnológico lejos de abrir brechas entre las artes y las ciencias ha permitido la unión de dos líneas aparentemente divergentes del conocimiento humano.

- Se ha demostrado el enorme potencial de los sistemas computarizados en la creación sintética del sonido.

- Hemos comprobado el gran poder y versatilidad del lenguaje de programación conocido como C. Verificando que el uso del mismo permite implementar prácticamente cualquier algoritmo de una manera estructurada y ante todo con una muy rápida velocidad de ejecución, lo que es fundamental en aplicaciones en tiempo real.

- Es importante señalar que la implementación de las diferentes subrutinas de los programas diseñados, nos ha permitido entender y manejar varios conceptos básicos que deben ser considerados dentro del campo tecnológico conocido como Ingeniería de Software.

5.3 RECOMENDACIONES

Debido a que las posibilidades del protocolo de comunicación MIDI son bastante extensas, los programas diseñados nos han permitido visualizar sólo algunas de estas posibilidades. Como recomendación se presentan las posibles alteraciones a los mismos:

- Debido a la existencia de mensajes MIDI exclusivos, es posible la transmisión de información de parámetros de sonidos almacenados en la memoria del sintetizador. El programa analizador, puede ser usado como una etapa inicial de un sistema que permita decodificar estos datos y almacenarlos en memoria, para así realizar alteraciones a los mismos y así poder transmitirlos al

sintetizador. De esta manera es posible la síntesis directa sobre de los sonidos del sintetizador.

- La implementación de un secuenciador multitrack, se facilita debido a que el sistema de almacenamiento de datos MIDI es a nivel de eventos. De esta manera, si a cada evento se le añade un byte donde se indique el track al cual está asignado, es posible enrutar cada evento a tracks diferentes. Esta alteración ampliaría enormemente las posibilidades del secuenciador.

- Los mensajes de fin de compás que transmite el MPU, pueden ser usados de tal manera de marcar en pantalla los tiempos del compás elegido al iniciar el metrónomo

- Al ser detenida la ejecución de una secuencia en el sintetizador, el programa secuenciador regresa automáticamente al inicio de secuencia. Esta situación puede ser alterada si se mantiene un registro del número de eventos y sus extensiones, de esta manera es posible emular las funciones de avance rápido y rebobinado de grabadoras análogas. Esto permitiría ejecutar una secuencia en cualquier punto de la misma.

- Existen varias alternativas para editar la información musical incluida en una secuencia. Por ejemplo, se recomienda la implementación de un sistema de transposición de altura tonal. Esto se podría lograr

guardando registro de todos los eventos MIDI que corresponden a nota presionada y liberada, de tal manera que se sume o reste al segundo byte del mensaje una cantidad ingresada por el usuario, la que debe ser correspondiente al valor de la transposición.

- En general para realizar la edición de cualquiera de los posibles eventos MIDI, se debe guardar registro de todos los eventos de un mismo tipo, para así modificar los bytes de datos respectivos.

A N E X O S

ANEXO Nº 1

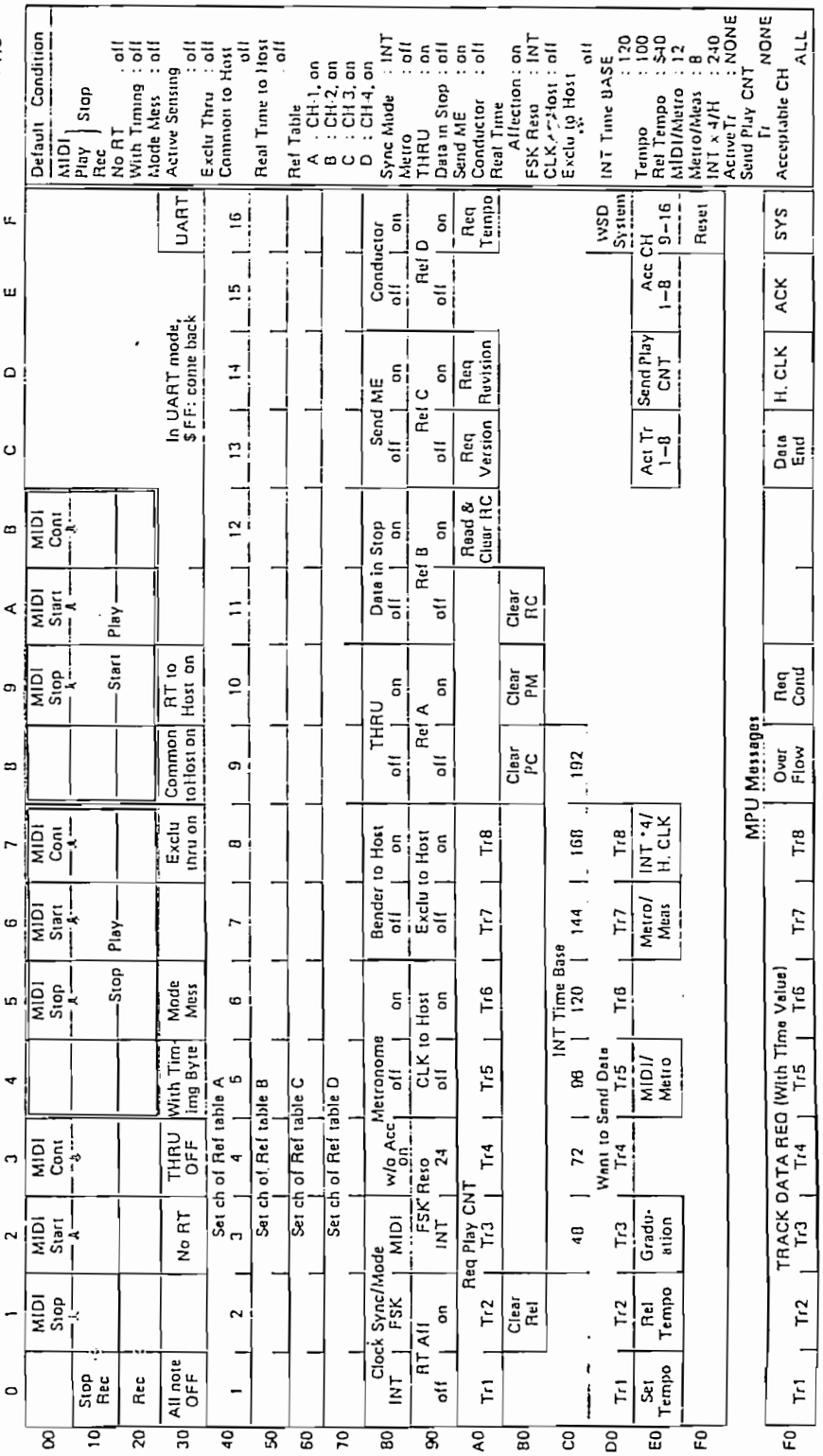
CARTA DE IMPLEMENTACION MIDI.

Function ...		Transmisrted	Recognized	Remarks
Basic Channel	Default	1—16	1—16	Memorized
	Changed	1—16	1—16	
Mode	Default		×	
	Messages	OMNI ON/OFF·POLY·MONO	×	created by even edit
	Altered	•••••		
Note Number	: True voice	0—127	0—127	
		•••••	0—127	
Velocity	Note ON	O	O•	
	Note OFF	×9n V = 0	×	
After Touch	Key's	O	×	ECHO only
	Ch's	O	O•	
Pitch Bender		O	O•	
Control Change	0 ~ 63	O	O•	
	64 ~ 121		O•	
Prog Change	: True #	O	O•	
		•••••	0—127	
SystemExclusive		O	O•	
System Common	: Song Pos	O	O	
	: Song Sel	O	O	
	: Tune	×	×	
System Real Time	: Clock	O	O••	
	: Commands	O	O••	
Aux Messages	: Local ON/OFF	O	×	Created by even edit
	: All Notes OFF	×	O (123~127)	
	: Active Sense	×	×	
	: Reset	×	×	
Notes		• : Can be set to O or × memorized. •• : Recognized only when clock is set to MIDI		

Mode 1 : OMNI ON, POLY Mode 2 : OMNI ON, MONO O : Y

ANEXO Nº 2

TABLA DE COMANDOS DEL INTERFAZ MIDI ROLAND MPU-401.



ANEXO Nº 3

LISTADO DE SUBRUTINAS DE COMUNICACION (LENGUAJE ASSEMBLY).

```
;INICIO
;-------------------------------------------------------------------
;ASIGNACIONES

DATAPORT  EQU  330H       ;PORTICO DE DATOS DEL MPU-401
STATPORT  EQU  331H       ;PORTICO DE ESTADO DEL MPU-401
DRR       EQU  040H       ;VALOR  PARA COMPROBAR SI EL BIT #  6 ES 1L
MPUACK    EQU  0FEH       ;CODIGO DEL MPU (ACKNOWLEDGE)
INTENTOS  EQU  0FFH       ;MAXIMO NUMERO DE INTENTOS PARA OBTENER
                          ;RESPUESTA DEL MPU
;-------------------------------------------------------------------;

;A LAS SIGUIENTES FUNCIONES LAS HACEMOS PUBLICAS, DE  TAL MANERA QUE
;SEAN ENTENDIDAS POR EL PROGRAMA EN LENGUAJE C
;
PUBLIC    _leer_dato, _escribir_dato, _comando
;
;-------------------------------------------------------------------
;DEFINIMOS DOS MACROS  QUE SE USAN ANTES  Y DESPUES DE CADA  FUNCION
;EN ASSEMBLY


GUARDAR_STK MACRO         ;MACRO  PARA  GUARDAR  REGISTROS  DE  STACK
      PUSH BP
      MOV  BP,SP
      PUSH DI
      PUSH SI
      ENDM


RECUPERAR_STK  MACRO      ;MACRO PARA RECUPERAR REGISTROS  DE STACK
      POP  SI
      POP  DI
      MOV  SP,BP
      POP  BP
      ENDM

;-------------------------------------------------------------------
;MANEJO DE SEGMENTOS DE MEMORIA

STACK     SEGMENT para stack 'STACK'   ;SEGMENTO PARA STACK
STACK     ENDS

_DATA     SEGMENT word public 'DATA'   ;SEGMENTO PARA DATOS
_DATA     ENDS
```

```
;TEXTO DEL PROGRAMA

_TEXT     SEGMENT byte public 'CODE'   ;SEGMENTO PARA PROGRAMA

;DIRECTIVA DE REGISTROS PARA DIRECCIONES DE ETIQUETAS

     ASSUME CS:_TEXT, DS:_DATA, SS:STACK, ES:NOTHING

;*************************************************************
;*********************     FUNCIONES     ***********************
;*************************************************************

;******   _COMANDO(n)    ******
;     ENTREGA UN COMANDO AL MPU, CHEQUEA EL RECIBIMIENTO DEL  MISMO
;     VERIFICANDO EL CODIGO DE ACKNOWLEDGE DEL MPU-401.

_comando  PROC

     GUARDAR_STK
     MOV  DX,STATPORT     ;GUARDAR  EN  DX  #  DE  PORTICO  DE ESTADO
     MOV  CX,INTENTOS     ;GUARDAR EN CX EL NUMERO DE INTENTOS

LBL1:
     IN   AL,DX       ;LEER ESTADO
     TEST AL,DRR      ;CHEQUEA SI BIT # 6 ES OL
     JZ   LBL2        ;SI ES OL, SIGUE A LABEL 2
     DEC  CX          ;DECREMENTO DEL CONTADOR DE INTENTOS
     CMP  CX,1        ;SE HAN HECHO TODOS LOS INTENTOS?
     JGE  LBL1        ;REALIZAR EL INTENTO UNA VEZ MAS
     MOV  AX,-1
     JMP  SALIR1      ;SALIR MANDANDO EL CODIGO DE ERROR = -1

LBL2:
     CLI              ;DESHABILITAR INTERUPCIONES
     MOV  AX,[BP+4]   ;ESCRIBE EL DATO EN AX
     OUT  DX,AL       ;ESCRIBE EL DATO EN PORTICO DE ESTADO

;REVISAR SI DATO FUE RECIBIDO CORRECTAMENTE

     MOV  CX,INTENTOS      ;GUARDAR EN CX # DE INTENTOS
LBL3:
     IN   AL,DX            ;LEER BYTE EN PORTICO DE ESTADO
     ROL  AL,1             ;GUARDAR BIT # 7 EN CARRY
     JNB  LBL4             ;SI CARRY = 0 IR A LABEL 4
     DEC  CX               ;DECREMENTAR CONTADOR DE INTENTOS
     CMP  CX,1             ;SE HAN HECHO TODOS LSO INTENTOS?
     JGE  LBL3             ;INTENTAR UNA VEZ MAS
     MOV  AX,-1            ;SALIR ENVIANDO CODIGO DE ERROR
     JMP  SALIR1
```

```
LBL4:
      MOV   DX,DATAPORT       ;GUARDAR EN DX # DE PORTICO DE DATOS
      IN    AL,DX             ;LEER DATO DE PORTICO DE DATOS
      CMP   AL,MPUACK         ;COMPARAR CON CODIGO ACK
      JZ    SALIR1            ;SI ES CODIGO ACK, SALIR
      MOV   AX,-1             ;SALIR ENVIANDO CODIGO DE ERROR
SALIR1:
      STI                     ;HABILITAR INTERRUPCIONES
      RECUPERAR_STK           ;RECUPERAR REGISTROS DEL STACK
      RET
_comando  ENDP
;-------------------------------------------------------------------

;******    LEER_DATO(n)    ******
;    LEER UN DATO DESDE EL MPU401

_leer_dato PROC
      GUARDAR_STK             ;GUARDAR REGISTROS EN EL STACK
      MOV   DX,STATPORT       ;GUARDAR EN DX, EL # DE PORTICO DE ESTADO
      MOV   CX,INTENTOS       ;EN CX GUARDA EL # DE INTENTOS
LBL6:
      IN    AL,DX             ;LEER EL VALOR EN EL PORTICO DE ESTADO
      ROL   AL,1              ;CHEQUEAR SI EL BIT # 7 ES OL
      JNB   LBL7              ;SI, ES OL VAYA A LABEL 7
      DEC   CX                ;DECREMENTAR EL CONTADOR DE INTENTOS
      CMP   CX,1              ;SE HAN HECHO TODOS LOS INTENTOS
      JGE   LBL6              ;HACER UN INTENTO MAS
      MOV   AX,-1             ;SALIR ENVIANDO CODIGO DE ERROR
      JMP   SALIR2
LBL7:
      MOV   DX,DATAPORT       ;GUARDAR  EN   DX  EL   PORTICO  DE   DATO
      MOV   AH,0              ;ENCERA AX (BMS)
      IN    AL,DX              ;LER EN AX (bms) EL DATO DEL PORTICO DE
                              ;DATOS
SALIR2:
      RECUPERAR_STK           ;RECUPERAR REGISTROS DEL STACK
      RET                     ;REGRESAR
_leer_dato ENDP

;-------------------------------------------------------------------
;******    ESCRIBIR_DATO(n)    ******
;     ENVIA UN DATO AL MPU - 401

_escribir_dato PROC
      GUARDAR_STK             ;GUARDAR REGISTROS EN EL STACK
      MOV   DX,STATPORT       ;GUARDAR EN DX EL PORTICO DE ESTADO
      MOV   CX,INTENTOS       ;GUARDAR EN CX EL # DE INTENTOS
LBL8:
      IN    AL,DX             ;LEER   EL  VALOR  EN  PORTICO  DE  ESTAD
      TEST  AL,DRR            ;PRUEBA SI EL BIT 6 ES OL
      JZ    LBL9              ;SI NO ES OL, VAYA A LABEL 8
      DEC   CX                      ;DECREMEMNTA EL CONTADOR  DE INTENTO
      CMP   CX,1              ;SE HAN HECHO TODOS LOS INTENTOS?
      JGE   LBL8              ;INTENTAR UNA VEZ MAS
      MOV   AX,-1             ;REGRESAR EL CODIGO DE ERROR
      JMP   SALIR3            ;SALIR
```

```
LBL9:
     MOV  DX,DATAPORT      ;GUARDAR EN DX EL # DE PORTICO DE DATOS
     MOV  AX,[BP+4]        ;TRAER EL DATO DESE EL STACK
     OUT  DX,AL            ;PONER EL DATO EN PORTICO DE DATOS
SALIR3:
     RECUPERAR_STK         ;RECUPERAR REGISTROS DEL STACK
     RET                   ;REGRESAR
_escribir_dato ENDP

;-------------------------------------------------------------------------
_TEXT     ENDS              ;FIN DE SEGMENTO DE TEXTO
     END
```

ANEXO Nº 4

LISTADO DEL MODULO PRINCIPAL DEL PROGRAMA ANALIZADOR.

```
/**
*** ESCUELA POLITECNICA NACIONAL
*** FACULTAD DE INGENIERIA ELECTRICA
*** DEPARTAMENTO DE ELECTRONICA Y CONTROL
***
*** TESIS DE GRADO:
***
***                    ADRIAN NARANJO PUENTE
***
*** DIRECTOR:
***                    DR. HUGO BANDA GAMBOA
***
*** ANALIZADOR DE DATOS MIDI   (ANALIZER.EXE)
***
*/

#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <graphics.h>
#include "notas.h"
#include "control.h"
#include <alloc.h>
                                /* Asignaciones */
#define UART      0x3F          /* Comando para  setear al MPU-401 en
                                   modo UART*/
#define RESET     0xFF          /* Comando  de reseteo  del MPU-401 */
#define BUFFER    1000          /* Tamaño del buffer para almacenar
los datos temporalmente */
#define ESC       27            /* Código ASCII para tecla ESC */
#define a         65            /* Código ASCII para letra a */
main()
{
/* Declaración de variables */

void *imagen1;                   /*Variables para almacenar
                                   temporalmente*/
void *imagen2;                   /*La imagen grafica en pantalla*/
Char str[2],d;                   /*Variable de caracteres para leer
                                   datos a transmitir*/

int i, j, k,kk, p, n, *m;
static int xm;                   /*Valores de rango máximo en x e y*/
static int ym;
int x,y;
static int mdata[BUFFER];        /*Arreglo  para almacenar los datos
                                   MIDI recibidos desde el MPU*/

int gd=DETECT, gm=0;
m=&p;
```

```
initgraph( &gd, &gm, "");      /*INICIALIZACION MODO GRAFICO*/

xm=getmaxx();           /*Detección  de valores máximos  en x e y*/
ym=getmaxy();


                    /*Definición de los tamaños en memoria para
                      almacenar  temporalmente las imagenes
                                                  g r á f i c a s * /
imageni=malloc(imagesize(0,0,xm,0.7184*ym));

imagen2=malloc(imagesize(0.3056*xm,0.6609*ym,0.7222*xm,0.8046*ym));

  setlinestyle (0,0,3);      /*Dibujo de los diferentes bloques en
                               pantalla*/
  setcolor(3);
  line(0.0694*xm,0.1006*ym,0.9583*xm,0.1006*ym);
  line(0.9583*xm,0.1006*ym,0.9583*xm,0.9626*ym);
  line(0.0694*xm,0.1149*ym,0.0694*xm,0.1006*ym);
  line(0.9583*xm,0.9626*ym,0.9514*xm,0.9626*ym);
  rectangle(0.0625*xm,0.1149*ym,0.9514*xm,0.8621*ym);
  rectangle(0.0625*xm,0.8621*ym,0.9514*xm,0.9770*ym);

/* Impresión de mensajes en pantalla */

     x=0.0694*xm;
     y=0.1293*ym;

     outfmttext(x,y,"ESCUELA POLITECNICA NACIONAL ");
     y=0.1580*ym;
     outfmttext(x,y,"FACULTAD DE INGENIERIA ELECTRICA ");
     y=0.1868*ym;
     outfmttext(x,y,"DEPARTAMENTO DE ELECTRONICA Y CONTROL");
     y=0.2155*ym;
     outfmttext(x,y,"TESIS DE GRADO: ADRIAN NARANJO PUENTE");
     y=0.2443*ym;
     outfmttext(x,y,"DIRECTOR:       DR. HUGO BANDA GAMBOA");


     setfillstyle(SOLID_FILL,3);
     bar(0.25000*xm,0.2845*ym,0.78890*xm,0.3563*ym);
     setcolor(0);
     settextstyle(0,0,2);

     x=0.2529*xm;
     y=0.2902*ym;
     outfmttext(x,y,"ANALIZADOR DE DATOS MIDI");
     setcolor(3);
     settextstyle(0,0,1);
     setlinestyle(0,0,1);
        /*Dibujo de bloque de recepción*/
     line(0.0694*xm,0.3736*ym,0.50280*xm,0.3736*ym);
     line(0.5028*xm,0.3736*ym,0.5028*xm,0.8477*ym);
     line(0.5028*xm,0.84770*ym,0.0694*xm,0.84770*ym);
     line(0.0694*xm,0.84770*ym,0.0694*xm,0.3736*ym);
     line(0.0694*xm,0.4195*ym,0.50280*xm,0.4195*ym);
```

```
   /*Dibujo de bloque de transmisión*/
line(0.5097*xm,0.3736*ym,0.94440*xm,0.3736*ym);
line(0.94440*xm,0.3736*ym,0.94440*xm,0.8477*ym);
line(0.94440*xm,0.8477*ym,0.50970*xm,0.8477*ym);
line(0.50970*xm,0.8477*ym,0.50970*xm,0.3736*ym);
line(0.50970*xm,0.4195*ym,0.94440*xm,0.4195*ym);

   /*Dibujo de bloque de mensajes*/
line (0.0694*xm,0.8764*ym,0.55560*xm,0.87640*ym);
line (0.55560*xm,0.87640*ym,0.55560*xm,0.9626*ym);
line (0.55560*xm,0.9626*ym,0.0694*xm,0.96260*ym);
line (0.0694*xm,0.9626*ym,0.0694*xm,0.8764*ym);

   /*Dibujo de bloque de comandos*/
line (0.56250*xm,0.8764*ym,0.94440*xm,0.8764*ym);
line (0.94440*xm,0.8764*ym,0.94440*xm,0.9626*ym);
line (0.94440*xm,0.9626*ym,0.56250*xm,0.9626*ym);
line (0.56250*xm,0.9626*ym,0.56250*xm,0.8764*ym);

   /*Dibujo de bloque de byte recibido*/
line (0.0764*xm,0.7184*ym,0.49580*xm,0.7184*ym);
line (0.49580*xm,0.7184*ym,0.49580*xm,0.8362*ym);
line (0.49580*xm,0.8362*ym,0.0764*xm,0.8362*ym);
line (0.0764*xm,0.8362*ym,0.0764*xm,0.7184*ym);
rectangle(0.0764*xm,0.6897*ym,0.49580*xm,0.7184*ym);

x=0.0819*xm;
y=0.6948*ym;
outfmttext(x,y,"BYTES RECIBIDOS:");

   /*Dibujo de bloque de byte transmitido*/
line (0.51670*xm,0.7184*ym,0.93750*xm,0.7184*ym);
line (0.93750*xm,0.7184*ym,0.93750*xm,0.8362*ym);
line (0.93750*xm,0.8362*ym,0.51670*xm,0.8362*ym);
line (0.51670*xm,0.8362*ym,0.51670*xm,0.7184*ym);
rectangle(0.51670*xm,0.6897*ym,0.93750*xm,0.7184*ym);

x=0.5222*xm;
y=0.6948*ym;
outfmttext(x,y,"BYTES TRANSMITIDOS:");

x=0.5694*xm;
y=0.8908*ym;
outfmttext(x,y,"ESC:     Salir");

x=0.5694*xm;
y=0.9253*ym;
outfmttext(x,y,"<RET>:  Transmitir");

x=0.0764*xm;
y=0.8966*ym;
outfmttext(x,y,"# de bytes en buffer = 0");

recepcion();          /*Iniciar en modo de recepción*/

COMANDO(UART);        /*Enviar al MPU para ponerlo en modo UART*/
```

```
  /*Lazo de duración indefinida, interrumpible solo por el
    usuario*/

    while (1)
    {
    INICIO:
          j = 0;    /*Inicialización del contador de datos del
                      buffer*/
       /*Lazo de lectura de datos desde el MPU*/
          while ((i = LEER_DATO()) != -1)
          {
             if (j==0)
             {
               setfillstyle(SOLID_FILL,0);
               bar(0.0764*xm,0.88510*ym,0.5380*xm,0.9195*ym);
               recepcion();
               x=0.0833*xm;
               y=0.9310*ym;

outfmttext(x,y,"\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
               x=0.305*xm;

outfmttext(x,y,"\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
             }
             x=0.0764*xm;
             y=0.8966*ym;
             outfmttext(x,y,"# de bytes en buffer =");

             /* Lée todos los datos MIDI del buffer */
             mdata[j++] = i;
             setfillstyle(SOLID_FILL,0);
             bar(0.32640*xm,0.8851*ym,0.5380*xm,0.9195*ym);
             x=0.3306*xm;
             y=0.8966*ym;
             outfmttext(x,y,"%d",j);

             /*Determinación de exceso de datos en buffer*/
             if (j > BUFFER - 1)
             {
                   setfillstyle (SOLID_FILL,0);
                   bar(0.0764*xm,0.8851*ym,0.46530*xm,0.9195*ym);
                   setcolor(3);
                   printf("\a");

                   x=0.0764*xm;
                   y=0.8966*ym;
                   outfmttext(x,y,"ERROR: Buffer de datos lleno");

                   for (k=0; k<5000; k++){}
                   setfillstyle(SOLID_FILL,0);
                   bar(0.0764*xm,0.8851*ym,0.50690*xm,0.9138*ym);
                   setcolor(3);

                   x=0.0764*xm;
                   y=0.8966*ym;
```

```
            outfmttext(x,y,"# de bytes en buffer =  1000");
            break;
         }
         setfillstyle(SOLID_FILL,0);
         bar(0.0764*xm,0.9310*ym,0.54170*xm,0.9598*ym);
         setcolor(3);
      }

      if (j) /*Imprime en pantalla todos los  datos, si
               estos fueron enviados por el sintetizador**/
      {
            kk=j;
            for (k = 0; k < j; k++)
      {
      if (mdata[k] != 0xFE)  /*Imprime en pantalla solo
                               los datos MIDI */
      recepcion();

      /*Llama a función que detecta el tipo de dato
        MIDI recibido*/

      reconocer(mdata[k]);
      kk=kk-1;
      setfillstyle(SOLID_FILL,0);
      bar(0.2630*xm,0.8851*ym,0.52780*xm,0.9195*ym);

      x=0.3306*xm;
      y=0.8966*ym;
      outfmttext(x,y,"%d",kk);
   }
   setfillstyle (SOLID_FILL,0);
   bar(0.0764*xm,0.8851*ym,0.53470*xm,0.9195*ym);

   x=0.0764*xm;
   y=0.8966*ym;
   outfmttext(x,y,"#  de bytes en buffer = 0");
   setcolor(3);
}

if (kbhit())      /*Procesa cualquier acción del usuario*/
{
      n = getch();     /*Leer el código ASCII de la tecla
                         presionada*/
      if (toupper(n)==a) /*Ayuda*/
         {
            ayuda();
            goto INICIO;
         }

      if  (n == ESC)  /*Si la tecla presionada corresponde  a
                        ESC salir del programa*/
      {
                     /*Guardar  imagen en memoria*/
getimage( 0.3056*xm, 0.6609*ym, 0.7222*xm,  0.8046*ym,imagen2);
            setfillstyle(SOLID_FILL,0);
            bar(0.3056*xm,0.6609*ym,0.7222*xm,0.8046*ym);
```

```
        setlinestyle(0,0,3);
        rectangle(0.3083*xm,0.6667*ym,0.7194*xm,0.7989*ym);
        setlinestyle(0,0,2);
        rectangle(0.3139*xm,0.6753*ym,0.7139*xm,0.7902*ym);
        setcolor(3);
        setfillstyle(SOLID_FILL,3);
        bar(0.3681*xm,0.7126*ym,0.6667*xm,0.7471*ym);
        setcolor(0);
        x=0.4028*xm;
        y=0.7213*ym;
        outfmttext(x,y,"SALIR:     SI   /   NO");
        setcolor(1);
LABEL1:
        j=getch();
        if (toupper(j)==115)        /*Determinación de tecla
                                      presionada es S ó N*/
        goto SALIR;

        if (toupper(j)==110)
        goto LABEL2;

        goto LABEL1;

 SALIR:
        restorecrtmode();
        break;

LABEL2:
        putimage(0.3056*xm,0.6609*ym,imagen2,0);
        goto INICIO;
    }

    /*Si no se trata de  salir, se desea transmitir*/
    transmision();
    setfillstyle(SOLID_FILL,0);
    bar(0.52640*xm,0.5029*ym,0.83330*xm,0.52598*ym);
    setcolor(3);
    setfillstyle (SOLID_FILL,0);
    bar(0.0764*xm,0.88510*ym,0.46530*xm,0.9195*ym);

    x=0.5264*xm;
    y=0.4741*ym;
    outfmttext(x,y,"Datos hex a transmitir ?");

    getimage(0,0,xm,0.7184*ym,imagen1);
    gets(str);    /* Lee como texto los dos digitos
                     exadecimales */

    putimage(0,0,imagen1,0);

    setfillstyle(SOLID_FILL,0);
    bar(0.52640*xm,0.5029*ym,0.83330*xm,0.5259*ym);
    setcolor(3);
    setfillstyle(SOLID_FILL,0);
    bar(0.52640*xm,0.4741*ym,0.83330*xm,0.4971*ym);
    setcolor(3);
```

```
            n = transf(str, m);   /*Llama a la función que
                                    transforma el texto a
                                    exadecimal, n sirve
                                    para verifir  que se hayan
                                    digitado  solo 2 números
                                    exadecimales*/
            if (n == 2)    /*Envia los datos a MPU, solo si se
                             escribieron 2 digitos exadecimales*/
            {
               ESCRIBIR_DATO(*m);   /*Envía un byte al MPU*/
               data_trans(*m,xm,ym); /*Llama a función  que imprime
                                       en modo gráfico los datos
                                       transmitidos*/
            }
            else    /*Si se digitaron mas de 2 digitos, o
                      caracteres no exadecimales, imprime
                      mensaje de error*/
            {
                setfillstyle (SOLID_FILL,0);
                bar(0.0764*xm,0.88510*ym,0.50690*xm,0.9195*ym);
                setcolor(3);
                x=0.0764*xm;
                y=0.8966*ym;
                outfmttext(x,y,"ERROR:  Use 2 digitos:  0-9,A-F");
            }
          }

        }
  COMANDO(RESET);          /*Resetea el MPU a su modo inteligente
                             normal*/
 }
```

- Función que transforma caracteres que representan números exadecimales a enteros exadecimales. TRANSF().

```
transf(str, byte)

char str[4];
int *byte;

{
    int n, i;
    *byte = i = 0;

    do{           /*Lazo para transformar números
                    decimales en números hexadecimales*/

     if ((n = hextran(str[i])) < 0)  /*Si el  caracter no
                                       corresponde a un número
                                       exadecimal, la función
                                       termina*/
     break;
```

```
      else
        *byte = (*byte * 16) + n;        /*Transformación a
                                           exadecimal*/

      }while (++i < 4);

      return(i);              /*Regresa el número de caracteres
                                transformados*/
}
```

- Función que transforma un caracter que representa un número exadecimal en decimal. HEXTRAN().

```
hextran(hex)    /*Función para convertir un caracter que representa
                  un número exadecimal en decimal*/

char hex;

{
     hex  = toupper(hex);                /*Transformamos a mayuscula al
                                           caracter recibido*/

     if (hex >= '0' && hex <= '9')      /*Transformamos  a decimal, si
                                           el caracter es  un número de 0
                                           a 9*/
          return(hex - '0');

      else
        if  (hex >= 'A' && hex <= 'F')    /*Transformación a decimal,
                                            si el caracter es una
                                            letra de A a F*/
          return(hex - 'A' + 10);

        else                           /*Si no corresponde a un número
                                         exadecimal regresa un valor
                                         menor que 0*/
          return(-1);
}
```

- Función que presenta "RECEPCION" en video reverso.

```
recepcion()
{
  int xm,ym,x,y;
  xm=getmaxx();
  ym=getmaxy();

  setfillstyle(SOLID_FILL,3);
  bar(0.0722*xm,0.3793*ym,0.50000*xm,0.4138*ym);
  setcolor(0);
```

```
   x=0.1944*xm;
   y=0.3879*ym;
   outfmttext(x,y,"RECEPCION:");

   setfillstyle(SOLID_FILL,0);
   bar(0.51250*xm,0.3793*ym,0.94170*xm,0.4138*ym);
   setcolor(3);

   x=0.6514*xm;
   y=0.3879*ym;
   outfmttext(x,y,"TRANSMISION:");
}
```

- Función que presenta "TRANSMISION" en video reverso.

```
transmision()
{

    int xm,ym,x,y;
    xm=getmaxx();
    ym=getmaxy();
    setfillstyle(SOLID_FILL,3);
    bar(0.51250*xm,0.3793*ym,0.94170*xm,0.4138*ym);
    setcolor(0);

    x=0.6514*xm;
    y=0.3879*ym;
    outfmttext(x,y,"TRANSMISION:");

    setfillstyle(SOLID_FILL,0);
    bar(0.0722*xm,0.3793*ym,0.50000*xm,0.4138*ym);
    setcolor(3);

    x=0.1944*xm;
    y=0.3879*ym;
    outfmttext(x,y,"RECEPCION:");
}
```

- FUNCION PARA ESCRITURA DE VARIABLES CON FORMATO EN MODO GRAFICO
  OUTFMTTEXT()

```
#include <conio.h>
#include <graphics.h>
#include <stdarg.h>
#include <stdio.h>

void outfmttext(int x, int y, char *fmt, ...)
```

```
{
  va_list arg_ptr;
  char t[255]; /*formateo de 255 carateres máximo*/
  va_start (arg_ptr,fmt); /*Apunta a argumentos opcionales*/
  vsprintf (t,fmt,arg_ptr); /*formatea el string*/
  va_end(arg_ptr);
  outtextxy (x,y,t);
}
```

ANEXO Nº 5

LISTADO DE FUNCIONES DEL PROGRAMA ANALIZADOR.

```
#include <stdarg.h>
#include <graphics.h>

reconocer(data)
int data;

{                    /*Definición de variables*/
  static int n=0; /*Variable de  detección de bytes de mensaje de
                     notas*/
  static int  p=0; /*Variable de detección de bytes de mensaje de
                      programa*/
  static int c=0; /*Variable de detección de bytes de mensaje de
                     control*/
  static int x,y;
  int xm,ym;
  int aux,canal;
  setcolor(3);
  xm=getmaxx(); /*Detección de valores máximos en x e y*/
  ym=getmaxy();
        /*Detección de  segundos y terceros bytes de los mensajes
          MIDI*/
   if(p==1)  /*Determina el  2do. byte de mensaje de cambio de
               programa*/
   {
     goto FINSUB_REC;
   }

   if(c==1)  /*Determina el  2do. byte de mensaje de cambio de
               control*/
   {
     c=0;
     det_cont(data,xm,ym); /*Llama a función que detecta el
                             controlador MIDI accionado*/
     goto FINSUB_REC;
   }

   if (n==1)  /*Determina el 2do. byte de mensaje de nota
                presionada*/
   {
     bloque1();
     setcolor(0);
     x=0.1767*xm;
     y=0.4943*ym;
     outfmttext(x,y,"Tecla presionada:");
     setcolor(3);
     tonal(data,xm,ym); /*Llama a función que detecta la tecla
                          presionada*/
     n=2;
     goto ESTADO;
   }
```

```
   if (n==2)  /*Determina el 3er. byte de mensaje de nota
                presionada*/
   {
      presion(data,xm,ym); /*Llama a  función que detecta la presión
                              ejercida sobre la tecla presionada*/
      p=0;
      c=0;
      n=1;
   }

/*Si el dato entra a esta etiqueta, se trata de un byte de estado*/

ESTADO:  /*Detección del tipo de byte de estado recibido*/

  aux=data &  0xf0; /*Tomamos solo el nibble más significativo del
                      byte de estado (tipo de byte de estado)*/
  canal= data & 0x0f; /*Tomamos solo el nibble menos significativo
                        (información de canal de transmisión)*/
  if (aux==0x90)    /*Detecta si se trata de byte de estado de nota
                      presionada*/
  {
     bloque1();  /*Llama a función que borra linea donde se indica
                   el byte de estado*/
     setcolor(0);
     x=0.1767*xm;
     y=0.4943*ym;
     outfmttext(x,y,"Tecla Presionada:");
     setcolor(3);
     n=1; /*Variable  usada para indicar que se espera un byte de
            datos*/
     p=0;
     c=0;
     goto CANAL; /*Va a imprimir el canal de transmisión*/
  }

  if (aux==0x80)  /*Detecta si se trata de byte de estado de nota
                    liberada*/
  {
     bloque1();
     setcolor(0);
     x=0.1667*xm;
     y=0.4943*ym;
     outfmttext(x,y,"Tecla Liberada:");
     setcolor(3);
     p=0;
     n=0;
     c=0;
     goto CANAL;  /*Va a imprimir el canal de transmisión*/
  }

  if (aux==0xd0)      /*Detecta si se trata de byte de estado  de
                        mensaje de Aftertouch (monofónico)*/
  {
     bloque1();
     setcolor(0);
     x=0.1667*xm;
```

```
    y=0.4943*ym;
    outfmttext(x,y,"After touch:");
    setcolor(3);
    p=0;
    n=0;
    c=0;
    goto CANAL;   /*Va a imprimir el canal de transmisión*/
}

if (aux==0xa0)      /*Detecta si se trata de byte de estado de
                      mensaje de Aftertouch (polifónico)*/
{
    bloque1();
    setcolor(0);
    x=0.1667*xm;
    y=0.4942*ym;
    outfmttext(x,y,"After touch:(Pol)");
    setcolor(3);
    p=0;
    n=0;
    c=0;
    goto CANAL;   /*Va a imprimir el canal de transmisión*/
}

if (aux==0xe0)   /*Detecta si se trata de byte de estado de
                   mensaje de pitchbender*/
{
    bloque3();
    bloque1();
    bloque2();
    x=0.1867*xm;
    y=0.4943*ym;
    outfmttext(x,y,"Pitch Bender:");
    setcolor(3);
    p=0;
    n=0;
    c=0;
    goto CANAL;   /*Va a imprimir el canal de transmisión*/
}

if (aux==0xb0)    /*Detecta si se trata de byte de estado  de
                    mensaje de cambio de control*/
{
    bloque3();
    bloque1();
    bloque2();
    x=0.1767*xm;
    y=0.4943*ym;
    outfmttext(x,y,"Cambio de control:");
    setcolor(3);
    p=0;
    n=0;
    c=1; /*Variable usada para indicar  que se espera un byte de
           datos*/
    goto CANAL;   /*Va a imprimir el canal de transmisión*/
```

```
if (aux==0xf0)      /*Detecta si se trata de byte de estado de
                      mensaje de reloj MIDI*/
{

   bloque3();
   bloque1();
   bloque2();
   x=0.20*xm;
   y=0.4943*ym;
   outfmttext(x,y,"Reloj MIDI:");
   n=1;
   p=0;
   c=0;
   setfillstyle(SOLID_FILL,0);
   bar(0.08*xm,0.5460*ym,0.4861*xm,0.5805*ym);
   setcolor(3);
   goto FINSUB_REC;
}


if (aux==0xc0)    /*Detecta si se trata  de byte de estado de
                    mensaje de cambio de programa*/
{

   bloque3();
   bloque1();
   bloque2();
   x=0.1667*xm;
   y=0.4943*ym;
   outfmttext(x,y,"Cambio de Programa:");
   setcolor(3);
   n=0;
   p=1;  /*Variable usada para indicar que se espera un  byte de
           datos*/
   c=0;
   goto CANAL;  /*Va a imprimir el canal de transmisión*/
}

goto FINSUB_REC;


CANAL:   /*Imprimir número de canal de transmisión*/

   setfillstyle(SOLID_FILL,0);
   bar(0.2083*xm,0.5460*ym,0.4861*xm,0.5805*ym);
   setcolor(3);
   x=0.0972*xm;
   y=0.5603*ym;
   outfmttext (x,y,"Canal MIDI: %d",canal+1);
   p=0;


FINSUB_REC:
      desplegar(data,xm,ym);   /*Llama a función que despliega  los
                                 datos recibidos*/
}
```

```
bloque1()                    /*Función para borrar la linea de
                               tipo de byte de estado*/


{
      int xm,ym;

      xm=getmaxx();
      ym=getmaxy();
      setfillstyle(SOLID_FILL,3);
      bar(0.1597*xm,0.4885*ym,0.3819*xm,0.5172*ym);
      setcolor(0);
}
```

- FUNCION PARA DESPOLEGAR EN MODO GRAFICO DATOS TRANSMITIDOS

DESPLEGAR()

```
desplegar(data,xm,ym)        /*Función para desplegar en modo gráfico
                               los datos recibidos*/
int data,xm,ym;
{
      static int col=0;
      static int fil=0;
      int x,y;

      x=0.0900*xm+col*0.0333*xm;
      y=0.7241*ym+fil*0.0287*ym;
      col++;

      if (col==12)   /*Imprimir 12 columnas*/
      {
            fil++;
            col=0;
            goto END;
      }

      if(fil==4)    /*Imprimir 4 filas*/
      {
            x=0.0900*xm;
            y=0.7241*ym;
            fil=0;
            col=1;
            setfillstyle(SOLID_FILL,0);
            bar(0.0792*xm,0.7241*ym,0.4790*xm,0.8305*ym);
      }

      END:

      outfmttext(x,y,"%02x",data);
}
```

- FUNCION PARA DESPLEGAR DATOS TRANSMITIDOS EN MODO GRAFICO

  DATA_TRANS()

```
data_trans(data,xm,ym)          /*Función para desplegar en modo
                                  gráfico los datos transmitidos*/


int data,xm,ym;
{

      int x,y;
      static int col=0;
      static int fil=0;
      x=0.5336*xm+col*0.0333*xm;
      y=0.7241*ym+fil*0.0287*ym;
      col++;

      if (col>11)       /*Imprimir 12 columnas*/
      {
            fil++;
            col=0;
            goto END;
      }

      if(fil>3)         /*Imprimir 4 filas*/
      {
            x=0.5336*xm;
            y=0.7241*ym;
            fil=0;
            col=1;
            setfillstyle(SOLID_FILL,0);
            bar(0.5194*xm,0.7213*ym,0.92*xm,0.8333*ym);
      }

      END:
      outfmttext(x,y,"%02x",data);
}
```

- FUNCION DE RECOCIMIENTO DE TECLA PRESIONADA
  (TONAL()).

  (Segundo byte de mensaje de tecla presionada)

```
#include <stdarg.h>
#include <graphics.h>
#include "notas.h"   /*El nombre y código de las teclas  de un
                       sintetizador MIDI se encuentra en el  Header:
                       NOTAS.H*/

tonal(data,xm,ym)
int data,xm,ym;
{
```

```
      int num;
      int x,y;
      setcolor(3);

      for(num=0;num<=62;num++)
      {
            if(data==num_not[num]) /*Detecta tecla presionada*/
                goto DETECTADO;
      }

      DETECTADO:  /*Imprime nombre de tecla presionada*/

            bloque3(); /*Borra linea anterior de nombre de
                         tecla presionada*/
            x=0.0972*xm;
            y=0.6034*ym;
            outfmttext(x,y,"Nota:   %s",nota[num]);
}

bloque3() /*Función para borrar  linea de indicación de nota
            presionada*/

 {
   int xm,ym;
   xm=getmaxx();
   ym=getmaxy();
   setfillstyle(SOLID_FILL,0);
   bar(0.0833*xm,0.5977*ym,0.4861*xm,0.6236*ym);
}
```

- FUNCION DE RECONOCIMIENTO DE PRESION EJERCIDA (PRESION()).

(Tercer byte de mensaje de nota presionada)

```
# include <stdarg.h>
# include <graphics.h>

presion(data,xm,ym)
int data,xm,ym;
{
  int x,y;
                          /*Detección de fuerza ejercida*/

      if (data  == 0x00)  /*Si el dato es 00H, se ha liberado la
                            tecla*/
      {
            bloque1(); /*Borra la linea de byte de estado*/
            setcolor(0);
            bloque2(); /*Borra la linea de fuerza ejercida*/
            x=0.1944*xm;
            y=0.4943*ym;
            outfmttext(x,y,"Tecla liberada:");
            setcolor(3);
            goto FINSUB_FUERZA;
      }
```

```
if (data >> 0 && data << 20)
{
      setcolor(3);
      x=0.0972*xm;
      y=0.6466*ym;
      outfmttext(x,y,"Fuerza: ppp");
      goto FINSUB_FUERZA;
}


if (data >> 20 && data << 40)
{
      setcolor(3);
      x=0.0972*xm;
      y=0.6466*ym;
      outfmttext(x,y,"Fuerza: pp");
      goto FINSUB_FUERZA;
}


if (data >> 40 && data << 50)
{
      setcolor(3);
      x=0.0972*xm;
      y=0.6466*ym;
      outfmttext(x,y,"Fuerza: p");
      goto FINSUB_FUERZA;
}


if (data >> 50 && data << 64)
{
      setcolor(3);
      x=0.0972*xm;
      y=0.6466*ym;
      outfmttext(x,y,"Fuerza: mp");
      goto FINSUB_FUERZA;
}

if (data >> 64 && data << 70)
{
      setcolor(3);
      x=0.0972*xm;
      y=0.6466*ym;
      outfmttext(x,y,"Fuerza: mf");
      goto FINSUB_FUERZA;
}

if (data >> 70 && data << 80)
{
      setcolor(3);
      x=0.0972*xm;
      y=0.6466*ym;
      outfmttext(x,y,"Fuerza: f");
      goto FINSUB_FUERZA;
}
```

```
      if (data >> 80 && data << 90)
      {
            setcolor(3);
            x=0.0972*xm;
            y=0.6466*ym;
            outfmttext(x,y,"Fuerza: ff");
            goto FINSUB_FUERZA;
      }

      if (data >> 115)
      {
            setcolor(3);
            x=0.0972*xm;
            y=0.6466*ym;
            outfmttext(x,y,"Fuerza: fff");
            goto FINSUB_FUERZA;
      }

  FINSUB_FUERZA:
}


bloque2()       /*Función para borrar linea de fuerza ejercida*/

{
  int xm,ym;

  xm=getmaxx();
  ym=getmaxy();

  setfillstyle(SOLID_FILL,0);
  bar(0.0972*xm,0.6466*ym,0.5*xm,0.6753*ym);
}
```

- FUNCION DE RECONOCIMIENTO DE CONTROLADORES MIDI

(DET_CONT()).

(Segundo byte de mensaje de cambio de control).

```
#include <stdarg.h>
#include <graphics.h>
#include "control.h" /*El nombre  y código de los controladores MI
                       se encuentra en en el Header: CONTROL.H*/

det_cont(data,xm,ym)
int data,xm,ym;

{
      int num,x,y;

      setcolor(3);
```

```
      for(num=0;num<=128;num++)
      {
            if(data==num_control[num]) /*Detecta el controlador*/
               goto DETECTADO;
      }

      DETECTADO: /*Imprime en pantalla en nombre del controlador*/

            bloque3();
            x=0.0972*xm;
            y=0.6034*ym;
            outfmttext(x,y,"%s:",controlador[num]);
}
```

- FUNCION DE MANEJO DE AYUDAS MIDI

```
/*FUNCION DE AYUDAS*/

#include <stdarg.h>
#include <graphics.h>
#include <alloc.h>
#include "ayuda1.h"     /*Headers que contienen información sobre*/
#include "ayuda2.h"     /*el protocolo MIDI*/
#include "ayuda3.h"     /*Estos headers de  ayuda se encuentran*/
#include "ayuda4.h"     /*listados en el ANEXO # 6*/
#include "control.h"
#include "notas.h"

ayuda()
{
      int num,num1,aux;
      int xm,ym;
      int  x,y,h;
      void *ayudas;
      xm=getmaxx();
      ym=getmaxy();
      /*Guarda  imagen en  memoria, para  crear  ventana de  ayudas*
ayudas=malloc(imagesize(0.1667*xm,0.2874*ym,0.8333*xm,0.8621*ym));
getimage(0.1667*xm,0.2874*ym,0.8333*xm,0.8621*ym,ayudas);
      setfillstyle(SOLID_FILL,0);
      bar(0.1667*xm,0.2874*ym,0.8333*xm,0.8621*ym);
      setlinestyle(0,0,3);
      rectangle(0.1694*xm,0.2931*ym,0.8292*xm,0.8563*ym);
      setlinestyle(0,0,0);
      rectangle(0.1736*xm,0.3017*ym,0.8250*xm,0.8477*ym);
      setfillstyle(SOLID_FILL,1);
      bar(0.1778*xm,0.3103*ym,0.8208*xm,0.3391*ym);
      setcolor(0);
      x=0.3056*xm;
      y=0.3161*ym;
      outfmttext(x,y,"* * * * A Y U D A S  M I  D I * * * *");
      setfillstyle(SOLID_FILL,1);
      bar(0.1778*xm,0.8103*ym,0.8208*xm,0.8391*ym);
      setcolor(0);
```

```
      x=0.1875*xm;
      y=0.8161*ym;
      outfmttext(x,y,"Q: SALIR DE AYUDA");
      x=0.6389*xm;
      y=0.8161*ym;
      outfmttext(x,y,"<RET>: OTRA PAG.");
      setcolor(1);
      rectangle(0.1778*xm,0.3448*ym,0.8208*xm,0.8046*ym);

/*************************************************************/
/*Primera página de ayudas (mensajes básicos)*/

      x=0.1875*xm;
      y=0.3736*ym;
      for(num=0;num<=14;num++)
      {
            outfmttext(x,y,"%s",lineas[num]);
            y=y+0.0287*ym;
      }
      h=getch();
      if (toupper(h)==81)
      goto FINAL;

/*************************************************************/
/*Paginas 2-5 de ayudas (codigos de notas)*/

      setfillstyle(SOLID_FILL,0);
      bar(0.1778*xm,0.3448*ym,0.8208*xm,0.8046*ym);
      x=0.2153*xm;
      y=0.3736*ym;
      outfmttext(x,y,"CODIGO DE NOTAS MIDI                    # CODIGO
                      HEX");          x=0.2153*xm;
      y=0.4023*ym;

      outfmttext(x,y,"-----------------------------------");
      aux=0;
      for(num1=0;num1<=4;num1++)
      {
            y=0.4310*ym;
            for(num=aux;num<=aux+12;num++)
            {
               x=0.2153*xm;
               outfmttext(x,y,"%s",nota[num]);
               x=0.6528*xm;
               if(num_control[num]<61)
                 outfmttext(x,y,"%x",num_not[num]);
               y=y+0.0287*ym;
            }
            h=getch();
            if (toupper(h)==81)
               goto FINAL;
            setfillstyle(SOLID_FILL,0);
            bar(0.1778*xm,0.4310*ym,0.8208*xm,0.8046*ym);
            aux=aux+13;
      }
```

```
/*****************************************************************/
/*Páginas 6-16 de ayudas (controladores MIDI)*/

      setfillstyle(SOLID_FILL,0);
      bar(0.1778*xm,0.3448*ym,0.8208*xm,0.8046*ym);
      x=0.2153*xm;
      y=0.3736*ym;
      outfmttext(x,y,"CONTROLADORES MIDI                          # CODIGO")
      x=0.2153*xm;
      y=0.4023*ym;
      outfmttext(x,y,"-----------------------------------------");
      aux=0;
      for(num1=0;num1<=9;num1++)
      {
            y=0.4310*ym;
            for(num=aux;num<=aux+12;num++)
            {
               x=0.2153*xm;
               outfmttext(x,y,"%s",controlador[num]);
               x=0.6528*xm;
               if(num_control[num]<128)
                 outfmttext(x,y,"%d",num_control[num]);
               y=y+0.0287*ym;
            }
            h=getch();
            if (toupper(h)==81)
               goto FINAL;
            setfillstyle(SOLID_FILL,0);
            bar(0.1778*xm,0.4310*ym,0.8208*xm,0.8046*ym);
            aux=aux+13;
      }

/*****************************************************************/
/*Página 17 de ayuda (modos de transmisión)*/

      setfillstyle(SOLID_FILL,0);
      bar(0.1778*xm,0.3448*ym,0.8208*xm,0.8046*ym);
      x=0.1875*xm;
      y=0.3736*ym;
      for(num=0;num<=14;num++)
      {
            outfmttext(x,y,"%s",modo[num]);
            y=y+0.0287*ym;
      }
      h=getch();

      if (toupper(h)==81)
            goto FINAL;

/*****************************************************************/
/*Página 18 de ayuda (mensajes exclusivos)*/

      setfillstyle(SOLID_FILL,0);
      bar(0.1778*xm,0.3448*ym,0.8208*xm,0.8046*ym);
      x=0.1875*xm;
      y=0.3736*ym;
```

```
      for(num=0;num<=14;num++)
      {
            outfmttext(x,y,"%s",excl[num]);
            y=y+0.0287*ym;
      }
      h=getch();
      if (toupper(h)==81)
            goto FINAL;

/*************************************************************/
/*Página 19 de ayuda (dinámica de velocidad)*/

      setfillstyle(SOLID_FILL,0);
      bar(0.1778*xm,0.3448*ym,0.8208*xm,0.8046*ym);
      x=0.1875*xm;
      y=0.3736*ym;
      for(num=0;num<=14;num++)
      {
            outfmttext(x,y,"%s",fuerza[num]);
            y=y+0.0287*ym;
      }
      getch();

FINAL: /*Recupera imagen antes de entrar a ayudas*/
      putimage(0.1667*xm,0.2874*ym,ayudas,0);
}
```

ANEXO Nº 6

FUNCIONES EN ASSEMBLY PARA MANEJO DE VIDEO.

```
;*************************************************************************
;*************************************************************************
;**   ESCUELA POLITECNICA NACIONAL                                      **
;**   FACULTAD DE INGENIERIA ELECTRICA                                  **
;**   DEPARTAMENTO DE ELECTRONICA  Y CONTROL                            **
;**                                                                     **
;**   ADRIAN NARANJO PUENTE                                             **
;**                                                                     **
;**   PROGRAMA:                                                         **
;**                                                                     **
;**   FUNCIONES EM  LENGUAJE ASSEMBLY (IBM PC) PARA MANIPULACION        **
;**   DE CARACTERES EN PANTALLA                                         **
;**                                                                     **
;**   NOMBRE:  BIOS.ASM                                                 **
;**                                                                     **
;**   FUNCIONES:                                                        **
;**                                                                     **
;**   _HOME()                                                           **
;**   _WRITCHR(C,N)                                                     **
;**   _WRITBW(C,N)                                                      **
;**   _WRITUL(C,N)                                                      **
;**   _WRITRV(C,N)                                                      **
;**   _WRITBLR(C,N)                                                     **
;**                                                                     **
;*************************************************************************
;*************************************************************************
;
;INICIO
;
;-----------------------------------------------------------------------
;ASIGNACIONES

BIOS10    EQU  10H  ;INTERRUPCION BIOS No.10 (HEX)
MAXCOL    EQU  79D  ;No. DE COLUMNAS DE PANTALLA 79 (DEC)
;-----------------------------------------------------------------------
;FUNCIONES PUBLICAS DE TAL MANERA QUE SEAN ENTENDIDAS POR EL
;PROGRAMA EN LENGUAJE C
;
PUBLIC  _home, _writchr, _writbw, _writul
PUBLIC  _writrv, _writblr
;-----------------------------------------------------------------------;
;DEFINICION DE DOS MACROS NECESARIOS ANTES Y DESPUES DE
;ENTREGAR EL CONTROL AL LENGUAJE ASSEMBLY
SAVSTK    MACRO          ;MACRO  PARA  GUARDAR  REGISTROS  EN  STACK
     PUSH BP
     MOV  BP,SP
     PUSH DI
     PUSH SI
     ENDM
```

```
RCLSTK    MACRO           ;MACRO  PARA RECUPERAR  REGISTROS DE  STAC
      POP   SI
      POP   DI
      MOV   SP,BP
      POP   BP
      ENDM

;-------------------------------------------------------------------
;MANEJO DE SEGMENTOS DE MEMORIA
;
STACK     SEGMENT para stack 'STACK'    ;SEGMENTO PARA STACK
STACK     ENDS

_DATA     SEGMENT word public 'DATA'    ;SEGMENTO PARA DATOS
_DATA     ENDS

;TEXTO DEL PROGRAMA

_TEXT     SEGMENT byte public 'CODE'    ;SEGMENTO PARA PROGRAMA

;DIRECTIVA DE REGISTROS PARA DIRECCIONES DE ETIQUETAS

      ASSUME CS:_TEXT, DS:_DATA, SS:STACK, ES:NOTHING

;*******************************************************************
;************************    FUNCIONES    ************************
;*******************************************************************
;-------------------------------------------------------------------

;******      _HOME()     ******
;  COLOCA EL CURSOR EN LA ESQUINA SUPERIOR IZQUIERDA DE LA PANTALLA

_home     PROC
      SAVSTK            ;NECESARIO ANTES DE REALIZAR CUALQUIER
                        ;PROCESO
      MOV   DX,0        ;SE GUARDA 0 EN DH Y DL (COORDENADAS DEL ORIGEN)
      MOV   BH,0        ;NUMERO DE PAGINA MEMORIA DE VIDEO
      MOV   AH,02H      ;NUMERO DE SERVICIO DEL BIOS PARA SETEAR LA
                        ;POSICION DEL CURSOR
      INT   BIOS10      ;SE LLAMA A LA INTERRUPCION  No. 10 PARA QUE
                        ;SE EJECUTE LA INFORMACION ALMACENADA EN LOS
                        ;REGISTROS AH, BH, DH Y DL.
      RCLSTK            ;NECESARIO ANTES DE ENTREGAR EL CONTROL AL
                        ;LENGUAJE C
      RET
_home     ENDP

;-------------------------------------------------------------------
;******      _WRITCHR(C,N)     ******
;  ESCRIBE EL CARACTER "C" N VECES EN PANTALLA EN LA POSICION
;  ACTUAL DEL CURSOR SIN CAMBIAR LOS ATRIBUTOS DEL TEXTO
_writchr PROC
      SAVSTK
      MOV   AX,[BP+4]   ;SE RECUPERA DEL STACK EL CODIGO ASCII
                        ;DEL CARACTER A ESCRIBIR
      MOV   AH,0AH      ;NUMERO DE SERVICIO DEL BIOS PARA
```

```
                        ;ESCRIBIR CARACTER SIN CAMBIO DE ATIBUTO
      MOV   CX,[BP+6]   ;SE RECUPERA DEL STACK EL VALOR DE "N"
      MOV   BX,0        ;NUMERO DE PAGINA DE MEMORIA DE VIDEO
      INT   BIOS10      ;SE EJECUTA INTERRUPCION No. 10
      CALL  INCCSR      ;USA LA FUNCION LOCAL INCSCR, PARA
                        ;INCREMENTAR EL CURSOR (EL SERVICIO No.
                        ;NO REALIZA ESTE PROCESO)
      RCLSTK
      RET
_writchr ENDP

;-------------------------------------------------------------------------;
;*****      _WRITBW(C,N)     *****
;   ESCRIBE EL CARACTER "C" N VECES EN PANTALLA EN LA POSICION
;   ACTUAL DEL CURSOR. LA ESCRITURA ES CON ATRIBUTOS BLANCO Y NEGRO
;   (CARACTER EN BLANCO, BACKGROUND NEGRO)

_writbw   PROC
      SAVSTK
      MOV   AX,[BP+4]   ;SE RECUPERA EL CARACTER A ESCRIBIR
      MOV   AH,09H      ;NUMERO DE SERVICIO PARA ESCRIBIR
                        ;CARACTER CON ATRIBUTO
      MOV   CX,[BP+6]   ;SE RECUPERA DE STACK VALOR DE "N"
      MOV   BX,0007H    ;BLANCO Y NEGRO EN PAGINA 0
      INT   BIOS10      ;EJECUTAR INTERRUPCION
      CALL  INCCSR      ;AUMENTAR CURSOR
      RCLSTK
      RET
_writbw   ENDP

;-------------------------------------------------------------------------;
;*****      _WRITUL(C,N)     *****
;   ESCRIBE EL CARACTER "C" N VECES EN PANTALLA EN LA POSICION
;   ACTUAL DEL CURSOR. LA ESCRITURA ES CON ATRIBUTOS DE SUBRAYADO

_writul   PROC
      SAVSTK
      MOV   AX,[BP+4]   ;SE RECUPERA EL CARACTER DEL STACK
      MOV   AH,09H      ;NUMERO DE SERVICIO PARA ESCRIBIR
                        ;CARACTER CON ATRIBUTO
      MOV   CX,[BP+6]   ;SE RECUPERA EL VALOR DE "N" DEL STACK
      MOV   BX,0001H    ;SUBRAYADO EN PAGINA 0 DE MEMORIA DE VIDE
      INT   BIOS10      ;EJECUTAR INTERRUPCION No. 10
      CALL  INCCSR      ;AUMENTAR CURSOR
      RCLSTK
      RET
_writul   ENDP

;-------------------------------------------------------------------------;
;*****      _WRITRV(C,N)     *****
;   ESCRIBE EL CARACTER "C" N VECES EN PANTALLA EN LA POSICION
;   ACTUAL DEL CURSOR. LA ESCRITURA ES CON VIDEO REVERSO

_writrv   PROC
      SAVSTK
      MOV   AX,[BP+4]   ;RECUPERAR EL CARACTER DEL STACK
```

```
      MOV   AH,09H      ;NUMERO DE SERVICIO PARA ESCRIBIR
                        ;CARACTER CON ATRIBUTO
      MOV   CX,[BP+6]   ;RECUPERAR EL VALOR DE "N" DEL STACK
      MOV   BX,0070H    ;MODO DE VIDEO REVERSO EN PAGINA 0
      INT   BIOS10      ;EJECUTAR INTERRUPCION No. 10
      CALL  INCCSR      ;AUMENTAR CURSOR
      RCLSTK
      RET
_writrv   ENDP

;-------------------------------------------------------------------;
;******      _WRITBLR(C,N)     ******
;  ESCRIBE EL CARACTER "C" N VECES EN PANTALLA EN LA POSICION
;  ACTUAL DEL CURSOR. LA ESCRITURA ES CON VIDEO REVERSO TITILANTE

_writblr  PROC
      SAVSTK
      MOV   AX,[BP+4]   ;RECUPERAR EL CARACTER DEL STACK
      MOV   AH,09H      ;NUMERO DE SERVICIO PARA ESCRIBIR
                        ;CARACTER CON ATRIBUTO
      MOV   CX,[BP+6]   ;RECUPERAR EL VALOR DE "N" DEL STACK
      MOV   BX,00F0H    ;MODO DE VIDEO REVERSO TITILANTE EN
                        ;PAGINA 0
      INT   BIOS10      ;EJECUTAR INTERRUPCION 10
      CALL  INCCSR      ;AUMENTAR CURSOR
      RCLSTK
      RET
_writblr  ENDP


;-------------------------------------------------------------------
;SUBRUTINA EN ASSEMBLER PARA INCREMENTAR EL CURSOR (ES SUBRUTINA
;SOLO LOCAL)

INCCSR    PROC near
      MOV   AH,03H      ;NUMERO DE SERVICIO PARA LEER POSICION
                        ;ACTUAL DEL CURSOR
      MOV   BH,0        ;PAGINA 0 DE MEMORIA DE VIDEO
      INT   BIOS10      ;EJECUTAR LA LECTURA DE LA POSICION
                        ;(EL RESULTADO SE GUARDA EN DX)
      CMP   DL,MAXCOL   ;COMPARA CON NUMERO MAXIMO DE COLUMNAS
      JGE   FILA        ;SI ES MAYOR QUE EL NUMERO DE COLUMNAS
                        ;SALTA A SIGUIENTE FILA
      INC   DL          ;INCREMENTAR COLUMNA
FILA:     MOV   AH,02H          ;NUMERO DE SERVICIO PARA LOCALIZAR
CURSOR    INT   BIOS10          ;EJECUTAR INTERRUPCION 10 PARA
                                ;LOCALIZAR CURSOR
      RET
INCCSR    ENDP

;-------------------------------------------------------------------
_TEXT     ENDS                            ;FIN DE SEGMENTO DE TEXTO
      END               ;FINAL DE PROGRAMA
```

- FUNCION PARA LEER IMAGENES ALMACENADAS EN ARCHIVOS

```
/*IMAGEN.C*/
#include <alloc.h>
#include <stdio.h>
#include "standard.h"     /*Header de codigós ASCII del IBM-PC*/
#include "imagen.h"       /*definición de estructuras*/

struct imagen *inpchain(file, maxlong) /*lee el archivo de imagen*/
char *file;

int maxlong;
{
     FILE *stream;      /*Puntero del archivo a leer*/
     struct imagen *p, *raiz, *anterior_p;
     char *strbuf;
     strbuf = (char *) malloc(maxlong + 1);    /*define espacio de
                                                 memoria temporal*/
     stream = fopen(file, "r");/*abrir el archivo para lectura*/
     p = raiz = chainalloc();  /*definir espacio de memoria para
                                 la  lectura de líneas del archivo*/
                               /*Leer línea a línea el archivo*/
     while (fgets(strbuf, maxlong, stream) != NULL)
     {p->linea = strsave(strbuf); /*Puntero de la línea que se lee*/
      p->prox = chainalloc();
         anterior_p = p;
      p = p->prox;}                 /*Puntero de la siguiente linea*/

     anterior_p ->prox = NULL;     /*NULL = final*/
     free(p);                  /*elimina los nodos no usados*/
     fclose(stream);           /*Cerrar el archivo leido*/
     free(strbuf);             /*Liberar buffer de memoria*/
     return(raiz);
}


dechain(p)                    /*Función para liberar memoria*/
struct imagen *p;             /*usada para leer el archivo*/
{
     struct imagen *q;
     while (p != NULL)
     {q = p->prox;
      free(p->linea); /*Libera memoria línea por línea*/
      free(p);
      p = q;}
}

dispchain(p)          /*Escribe líneas leidas en pantalla(stdout)*/
struct strchain *p;
{     while (p != NULL)
     {fputs(p->linea, stdout);
      p = p->prox;}
}
```

```
struct imagen *chainalloc() /*define memoria para  siguiente linea*/
{    return((struct imagen *) malloc(sizeof(struct imagen)));
}

char *strsave(s)     /*guarda lineas en memoria*/
char *s;
{
    char *p;
    if ((p = (char *) malloc(strlen(s) + 1)) != NULL)
        strcpy(p,s);
    return(p);
}

/* imagen.h   header para imagen.c */

struct imagen {
    char *linea;
    struct imagen *prox;};
struct imagen *inpchain(), *chain(), *chainalloc();
char *strsave();
```

— MENU.C FUNCION PARA CREAR MENUS DE PROGRAMAS

```
#include <stdio.h>
#include <conio.h>

#include "standard.h"    /*Headers de códigos del IBM PC
                           (listado en Anexo # 8)*/
#include "menu.h"    /*Headers de estructuras para almecenar
                       información del menú a implementar*/

opcion(pantalla, primero, ultimo)

struct menu pantalla[];
int primero,ultimo;       /*primer y  último elemento del menú*/

{
    int c, k, anteriork;
    k = primero;

    /*escribir en video el primer elemento del menú*/
    writword(RVC,pantalla[k].content,pantalla[k].xpos,
             pantalla[k].ypos);

    while (1)
   {
     anteriork = k;
     while (!kbhit()) ;  /*procesa acción del usuario*/
LABEL:
     c = getch();
     if (!c)
     c = getch();    /*borra los caracteres nulos y toma
                       el siguiente*/
```

```
        switch (c)      /*detecta que tecla se presionó*/
        {
        case KUP:
             k = pantalla[k].nup;
             break;
        case KDOWN:
             k = pantalla[k].ndown;
             break;
        case KLEFT:
        case BACKSP:
             k = pantalla[k].nleft;
             break;
        case KRIGHT:
        case TAB:
             k = pantalla[k].nright;
             break;
        case KHOME:
        case KPGUP:
             k = 0;
             break;
        case KEND:
        case KPGDN:
             k = ultimo;
             break;
        case ESC:          /*si se presionó ESC, regresa el código -2*
             return(-2);
        case CR:
        case KPLUS:
             return(k);    /*regresa el valor de la opción que se
                             escogió*/
        case '?':
             return(-3);   /*si se presionó "?" se regresa el
                             código  -3*/
        default:
               goto LABEL;  /*no se presionó ninguna de las teclas
                              mencionadas*/
        }


     /*Escribe en video normal opción anterior*/
     writword(BWC,pantalla[anteriork].content,
           pantalla[anteriork].xpos,pantalla[anteriork].ypos);

      /*Escribe en video reverso opción escogida*/
     writword(RVC,pantalla[k].content,pantalla[k].xpos,
           pantalla[k].ypos);
      }
}


- FUNCION PARA ESCRIBIR EN MODO DE TEXTO CARACTERES  CON CUALQUIE
  ATRIBUTO Y EN CUALQUIER POSICION DE PANTALLA

writword(modo,string,x,y)    /*se debe  enviar el atributo, los
                               caracteres  a imprimir y la posició
```

```
int modo,x,y;
char *string;
{
     char c;
     x=x+1;
     y=y+1;
     gotoxy(x,y);              /*localiza el cursor en posición
                                 indicada*/
     while ((c = *string++) != '\0'){
      switch (modo)
      {
            case (SAMEC):        /*se imprime sin cambio  de atributo*/
                 writchr(c,1);
            break;
            case (BWC):          /*modo de video normal*/
                 writbw(c,1);
            break;
            case (ULC):          /*subrayado*/
               writul(c,1);
            break;
            case (RVC):          /*video reverso*/
                 writrv(c,1);
            break;
            case (BRVC):         /*video reverso titilante*/
                 writblr(c,1);
            break;
            default:
                 writchr(c,1);
      }
     }
}

- FUNCION PARA ESCRIBIR MENSAJES DE ERROR EN PANTALLA

writerr(s)      /*escribe mensaje de error en pantalla
                  se debe enviar el mensaje a imprimir*/
char *s;
{
     int x,y;

     while (kbhit())
     getch();
     gotoxy(5,SCRNTALL-4); /*localiza el cursor el posición
                             indicada*/
     fputs(s,stderr);
     fputs(" <RET>",stderr);

     while (!kbhit()) ;              /*espera acción del usuario*/
     getch();
     gotoxy(5,SCRNTALL-4);

     writchr(' ',50);    /*borra mensaje de error al recibirse acción
                          del usuario*/
     return(1);
}
```

- menu.h: Header para menu.c

```
struct menu {
     int xpos;
     int ypos;
     char content[15];
     int nup;
     int ndown;
     int nleft;
     int nright;
     int key;
};
```

- INPUT.C FUNCION PARA INGRESO DE NUMEROS ENTEROS

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include "standard.h"  /*Códigos estandares del IBM-PC*/

getint(lineno,string,valor,min,max)
/*  El programa que solicita esta función debe enviar:
     el número de línea donde se debe imprimir el  mensaje de lectura
     del  dato a leer, el mensage a escribir, la variable en la que
     se guarda el valor y el rango  min. y máx. del valor a leer.
*/
int *valor, min, max, lineno;
char *string;
{
     static char input[20], minbuf[20], maxbuf[20], mensage[80];
     int tempval;

     while (kbhit())    /*Procesa acción del usuario*/
     getch();

     while (1)
     {
      writword(BWC,string,4,lineno);    /*Se llama a  la función que*/
                                        /*escribe caracteres*/
      fgets(input,19,stdin);
       /*Procesa cualquier acción del usuario*/
       /*al ingresar los datos. Si no se ingresa*/
       /*dato se regresa el valor "0"*/
      if (input[0] == '\n' || input[0] == '\0' || input[0] == ESC)
          return(0);
          tempval = atoi(input);

      if (tempval > max || tempval < min)  /*Si el valor esta  fuera
                                             del rango, indica  al
                                             usuario*/
      {
          itoa(min,minbuf,10);
          itoa(max,maxbuf,10);
          strcpy(mensage,"Valor debe estar entre ");
          strcat(mensage,minbuf);
```

```
            strcat(mensage," y ");
            strcat(mensage,maxbuf);
            writerr(mensage);
        }
        else
        {
            *valor = tempval;
            return(1);      /*Si se ingresa un valor adecuado se
                              regresa el código 1*/
        }
      }
}
```

- Header de comandos del MPU-401

```
/* marcas del mpu */

#define   NOP          0xf8       /* no operación */
#define   MES_END      0xf9       /*  fin de compás */
#define   DATA_END     0xfc       /*  fin de datos */

/* mpu messages */

#define   REQ_T1           0xf0 /*  track  data request,  track  1
#define   REQ_T2           0xf1
#define   REQ_T3           0xf2
#define   REQ_T4           0xf3
#define   REQ_T5           0xf4
#define   REQ_T6           0xf5
#define   REQ_T7           0xf6
#define   REQ_T8           0xf7 /* track data request, track 8*/
#define   TIME_OUT         0xf8 /* sobreflujo de tiempo */
#define   CONDUCT          0xf9 /* conductor data request */
#define   ALL_END          0xfc /* terminación */
#define   CLOCK_OUT        0xfd /* reloj del huesped */
#define   ACK              0xfe /* acknowledge */
#define   SYS_MES          0xff /* mensaje del sistema */

#define   STOP_PLAY        0x05
#define   START_PLAY       0x0a
#define   STOP_REC         0x11
#define   START_REC        0x22
#define   STOP_OVDUB       0x15
#define   START_OVDUB      0x2A

#define   NO_ALL_OFF       0x30 /* apagar todas las notas */
#define   NO_RTIME         0x32 /* no tiempo real */
#define   THRU_OFF_CHAN    0x33 /* thru : off en todos los canales *
#define   WITH_TIME        0x34 /* con byte de tiempo : on */
#define   MODE_THRU        0x35 /* modo mes : on */
#define   EXCL_THRU        0x36 /* exclusive thru : on */
#define   COM_THRU         0x38 /* común al huesped : on */
#define   REAL_THRU        0x39 /* tiempo real al huesped : on */
#define   UART             0x3f /* modo UART */
#define   INT_CLOCK        0x80 /* reloj interno */
#define   FSK_CLOCK        0x81 /* reloj FSK */
```

```
#define    MIDI_CLOCK      0x82 /* reloj MIDI */
#define    MET_ON_WOUT     0x83 /*  metronome :  on -  sin acentos  *
#define    MET_OFF         0x84 /* metronome : off */
#define    MET_ON_WITH     0x85 /*  metronome :  on -  con acentos  *
#define    BEND_OFF        0x86 /* bender : off */
#define    BEND_ON         0x87 /* bender : on */
#define    THRU_OFF        0x88 /* midi thru : off */
#define    THRU_ON         0x89 /* midi thru : on */
#define    DSTOP_OFF       0x8a /* datos en mode stop: off */
#define    DSTOP_ON        0x8b /* datos en modo stop: on */
#define    MEAS_OFF        0x8c /* enviar fin de compás: off */
#define    MEAS_ON         0x8d /* enviar fin de compás: on */
#define    COND_OFF        0x8e /* conductor : off */
#define    COND_ON         0x8f /* conductor : on */
#define    REAL_OFF        0x90
#define    REAL_ON         0x91
#define    FSK_INT         0x92 /* FSK interno */
#define    FSK_MIDI        0x93 /* FSK MIDI */
#define    CLOCK_OFF       0x94 /* reloj al huesped: off */
#define    CLOCK_ON        0x95 /* reloj al huesped: on */
#define    EXCL_OFF        0x96 /* exclusivo al huesped: off */
#define    EXCL_ON         0x97 /* exclusivo al huesped: on */

#define    CHANA_OFF       0x98 /* referencia: tabla a : off */
#define    CHANA_ON        0x99 /* a : on */
#define    CHANB_OFF       0x9a /* b : off */
#define    CHANB_ON        0x9b /* b : on */
#define    CHANC_OFF       0x9c /* c : off */
#define    CHANC_ON        0x9d /* c : on */
#define    CHAND_OFF       0x9e /* d : off */
#define    CHAND_ON        0x9f /* d : on */

/* lectura de datos */

#define    REQ_CNT0  0xa0      /*  request play counter de track 1 *
#define    REQ_CNT1  0xa1
#define    REQ_CNT2  0xa2
#define    REQ_CNT3  0xa3
#define    REQ_CNT4  0xa4
#define    REQ_CNT5  0xa5
#define    REQ_CNT6  0xa6
#define    REQ_CNT7  0xa7

#define    REQ_REC_CNT     0xab      /*  pedido de  record counter  *
#define    REQ_VER         0xac      /* pedido de versión */
#define    REQ_REV         0xad      /* pedido de revisión */
#define    REQ_TEMPO       0xaf      /* pedido de tiempo */

#define    RES_RTEMPO      0xb1 /* resets tiempo relativo */
#define    CLEAR_PCOUNT    0xb8 /*  borra los  contadores  de  ejec.*
#define    CLEAR_PMAP      0xb9 /* todas las notas off */
#define    CLEAR_REC       0xba /* borra el contador de grabación */

#define    TB_48           0xc2      /* 48 base de tiempo */
#define    TB_72           0xc3
#define    TB_96           0xc4
```

```
#define   TB_120          0xc5
#define   TB_144          0xc6
#define   TB_168          0xc7
#define   TB_192          0xc8

#define   WSD0      0xd0 /*  desea  enviar  datos   en  track  1
#define   WSD1      0xd1
#define   WSD2      0xd2
#define   WSD3      0xd3
#define   WSD4      0xd4
#define   WSD5      0xd0
#define   WSD6      0xd6
#define   WSD7      0xd7
#define   WS_SYS    0xdf /* desea enviar un mensaje de sistema */

/* ajuste de condiciones y valores (seguido por un byte de dato)
#define   SET_TEMPO       0xe0      /* ajuste de tiempo */
#define   SET_RTEMPO      0xe1      /* tiempo relativo */
#define   SET_GRAD        0xe2      /* graduación */
#define   MIDI_METRO      0xe4      /* beats por beep del metr.*/
#define   METRO_MEAS      0xe6      /* compás */
#define   INT_HOST        0xe7      /* interrupción al huesped */
#define   ACT_TRACK       0xec      /* tracks activos on/off */
#define   SEND_PCOUNT     0xed      /* enviar contador de ejec.*/
#define   CHAN_ON1  0xee    /* canales aceptables 1 - 8 on/off */
#define   CHAN_ON2  0xef    /* canales aceptables 9 - 16 on/off*/
#define   RESET     0xff    /* reset */
```

— Header de definciones estandar para el IBM PC

```
#define ESC     27
#define BACKSP  8     /* CODIGOS DE TECLAS */
#define CTLC    3
#define TAB     9
#define KUP     72
#define KDOWN   80
#define KLEFT   75
#define KRIGHT  77
#define KHOME   71
#define KEND    79
#define KPGUP   73
#define KPGDN   81
#define KDEL    83
#define KINS    82
#define KPLUS   78

#define SCRNWIDE 80 /* LONGITUDES DE PANTALLAS */
#define SCRNTALL 24
                /* CODIGOS PARA ATRIBUTOS EB PANTALLA */

#define SAMEC 0           /* IGUAL ATRIBUTO */
#define BWC   1           /* NORMAL */
#define ULC   2           /* SUBRAYADO */
#define RVC   3           /* VIDEO REVERSO */
#define BRVC  4           /* VIDEO REVERSO TITILANTE */
```

ANEXO Nº 7

LISTADO DEL MODULO PRINCIPAL DEL PROGRAMA SECUENCIADOR.

```
   /*PROGRAMA PRINCIPAL*/
   /*SECUENCIADOR MIDI PARA EL ROLAND MPU-401  */


#include <stdio.h>  /*headers*/
#include <conio.h>
#include <string.h>
#include <alloc.h>

#include "imagen.h"          /*header para lectura de archivo de
                              imagen en pantalla*/
#include "menu.h"            /*header para manejo de menus en video
                              reverso*/
#include "standard.h"       /*header de codigos del IBM PC*/
#include "mpu401.h"         /*header de codigos del MPU-401*/
#define NPARAM      7       /*definición  de número de opciones del
                              menú*/

struct menu secuenciador[NPARAM] = {
     {4, 10, "BPM",         NPARAM-1,   1, 0, 0, 0},
     {4, 11, "TIEMPO",      0,          2, 1, 1, 1},
     {4, 12, "METRONOMO",   1,          3, 2, 2, 2},
     {4, 13, "GRABAR",      2,          4, 3, 3, 3},
     {4, 14, "EJECUTAR",    3,          5, 4, 4, 4},
     {4, 15, "LISTAR",      4,          6, 5, 5, 5},
     {4, 17, "SALIR",       5,          0, 6, 6, 6},
};

void ejecutar();
void grabar();


                    /*PROGRAMA PRINCIPAL*/
main()
{
     struct imagen *schain;
     char buf[20];
     int i, respues, pick, lastpick, meton;
     int *metrate, *meter;
     metrate = (int *)malloc(sizeof(int));
     meter = (int *)malloc(sizeof(int));
     meton = 0;
     *metrate = 120;        /*Valores     de     ajustes     iniciales*/
                            /*del MPU-401*/
     *meter = 4;            /*120 bpm, compás 4/4*/
     clrscr();
     comando(RESET);              /*inicializar el MPU-401 */
     puts("Cargando imagen.");
     schain = inpchain("sec.scr",81);  /*lectura  del archivo que
                                         contiene imageneS*/
```

```
sendcmd(BEND_ON);       /*eliminar el filtro de datos  de
                          pitch-bend*/
sendcmd(SET_TEMPO);     /*Setear ajustes iniciales del MPU*/
escribir_dato(*metrate/2);
sendcmd(METRO_MEAS);
escribir_dato(*meter);

pick = 0;
clrscr();
dispchain(schain);              /*Imprimir imagen del programa*/
while (1)
{
 itoa(*metrate,buf,10);
 writword(BWC,buf,67,10);       /*escribir en pantalla valores*/
 strcpy(buf," ");               /*de ajustes iniciales*/
 itoa(*meter,buf,10);
 writword(BWC,buf,67,11);
 if (meton)                     /*escribir estado del metronómo*/
     writword(BWC,"ENCENDIDO",67,12);
 else
     writword(BWC,"APAGADO  ",67,12);

 while (kbhit())          /*Procesa acción del usuario*/
 getch();
 lastpick = pick;
                            /*llama a función de selección de
                              menú*/
 pick = opcion(secuenciador,pick,NPARAM - 1);
 switch (pick)
 {
 while (kbhit())          /*Procesa acción del usuario*/
 getch();
 case (0):       /*Se escogió opción BPM */
                 /*Se llama a función para leer BPM*/
 respues = getint(SCRNTALL-5,"Ingrese No. de beats/min ->",
                  metrate,8,200);
         if (respues == 1)    /*Se varió el valor de BPM*/
         {
            writword(BWC,"   ",67,10);
            escribir_dato(*metrate/2);
         }
         writword(BWC,"                              ",4,19);
         break;
 case (1):       /*Se escogió opción de Tiempo*/
                 /*Se llama función para leer el No. de
                   beats/compás*/
 respues = getint(SCRNTALL-5,"Ingrese No. de beats/compás ->",
                   meter,1,20);
         if (respues == 1)       /*Se varió el valor de tiempo*/
         {
             sendcmd(METRO_MEAS);
             escribir_dato(*meter);
             writword(BWC,"  ",67,11);
         }
         writword(BWC,"                         4,19);
         break;
```

```
      case (2):       /*Se escogió variar el estado del metrónomo*/
            if (meton)            /*Se apaga el metrónomo*/
            {
               sendcmd(MET_OFF);
               meton = 0;
            }
            else                  /*Se enciende el metrónomo*/
            {
               sendcmd(MET_ON_WOUT);
               meton = 1;
            }
            break;
       case (3):            /*Se escoge opción de grabación*/
            writword(BRVC," GRABANDO ",67,19);
            writword(BWC,"<RET> para detener grabación",4,
                     SCRNTALL-5);
          /*Se llama a función de grabación*/
            grabar();
            writword(BWC,"                          ",4,19);
            break;
       case (4):            /*Se escogió opción de ejecución*/
            sendcmd(ACT_TRACK);
            escribir_dato(1);
            ejecutar();           /*Se llama a función de ejecución*/
            break;
      case (5):       /*Se escogió opción de listar*/
            listar();
            writerr("LISTAR FILES.  ");
            break;
      case (-2):            /*ESC*/

      case (NPARAM-1):      /*se escogió la opción de salida*/
            clrscr();
            exit();
            break;
      default:
            pick = lastpick;
       }
     }
}

/*****************************************************************************/

#define BUFSIZE           100
int cmdbuf[BUFSIZE];      /*buffer para comandos al  MPU pendientes*/
int cmdbufp=0;

getnext() /*lee los posibles comandos pendientes desde el MPU
            ó el cmdbuf */
{
    return((cmdbufp > 0) ? cmdbuf[--cmdbufp] : leer_dato());
}
```

```
ungetnext(n)
int n;
{
    if (cmdbufp > BUFSIZE)
     printf("\nungetnext sin espacio en buffer.");
    else
     cmdbuf[cmdbufp++] = n;
}
get401()        /*saca el  proximo byte desde el MPU (o buffer
                  pendiente)*/
{
    int i;

    while (1)  /*Lazo infinito interrumpible solo por el usuario*/
    {
     i = getnext();
     if (kbhit()) return(-1);
     if (i != -1) return(i);
    }
}

/*********************************************************************/

sendcmd(n)    /*envia un  comando, chequea el ACK  si no guarda  los
datos MIDI*/ int n ;           /*hasta que estos ya no se envian*/
{
    int resp;

    resp = comando(n);
    if (resp == ACK)
     return;
    else if (resp != -1)
    {
     ungetnext(resp);          /*guarda datos  pendientes en  stack*/
     while (1)
     {                    /*chequéa si vienen más datos*/
         resp = leer_dato();
         if (resp == ACK || resp == -1)
          return;
         else
          ungetnext(resp);
     }
    }
}

/*********************************************************************/

guardar(nb,b1,b2,b3,b4,d)  /*función de almacenamiento de
                             información*/
int nb,b1,b2,b3,b4,d;
{
FILE *f;
static int num =1;
static int  datos[15000]={11,0,0}; /*tamaño máximo del archivo de
                                     datos*/
```

```
int t,a,cero;
char nombre[30];
int arr[5];

    while (kbhit())
    getch();                        /*Procesa acción del usuario*/
arr[0]=nb;arr[1]=b1;arr[2]=b2;arr[3]=b3;arr[4]=b4;
datos[0]=11;

for(t = 0;t < (nb+1);t++)
{
     datos[num]=arr[t];
     num++;
  if(num>=15000)
  {
     writword(BWC,"             ",67,19);
     writword(BRVC,"No se pudo definir más memoria!  <RET>",4,19);
     break;
  }
}

if (d==1)
{
    writword(BWC,"             ",67,19);

    while (kbhit())
    getch();                    /*procesa acción del usuario*/
NOMBRE:
     gotoxy(5,20);
     printf("Archivo a grabar ? (q:salir sin grabar)  ");
     gets(nombre);
     if  (nombre[0] == '\n' || nombre[0] == '\0' || nombre[0] ==
         ESC)
     goto NOMBRE;

     if (nombre [0] == 'q')
         goto LABEL;

     if(strchr(nombre,'.')== NULL)  /*aumenta extensión .MID*/
         strcat(nombre,".MID");
     printf("\n\n");
     f=fopen(nombre,"wb");
     for  (t=0;t<15000;t++)
     {
           putc(datos[t],f);
           if (t>20)
           {
                    cero=datos[t]+datos[t-1]+datos[t-2]+datos[t-3
                    if (cero== 0)
                         goto LABEL;
           }
     }
LABEL:
     for (t=0;t<20000;t++)  /*inicializa el  arreglo de datos
                              temporales*/
         datos[t]=0;
```

```
      num=0;
      rewind(f);
      fclose(f);
      }
}

/*****************************************************************/

void ejecutar()           /*función de ejecución*/
{
int i, comando,a,t,n,b,c;
static int song[15000]={0,0};  /*arreglo de lectura de  la
                                 secuencia*/
static g=0;
int nume = 0;
static  char name[20];
FILE *f2;

while (kbhit())
    getch();                      /*Pocesa acción del usuario*/
if (g==1)
{
     gotoxy(5,20);
     printf("Ejecutar archivo anterior (S=> si) ? ");
     a=getch();
     gotoxy(5,20);
     printf("                                          ");
     if (toupper(a) == 'S')
          goto REP;
}

NOMBRE:
gotoxy(5,20);
printf("Ingrese nombre del archivo a ejecutar ? ");
gets(name);

if (name[0]  == '\0' || name[0] == '\n')
         goto NOMBRE;

if(strchr(name,'.')== NULL)
         strcat(name,".MID");

gotoxy(5,20);
printf("                                                  ");
if((f2=fopen(name,"rb"))==NULL) /*regresa error si no puede abrir
                                  archivo*/
{
        gotoxy(5,20);
        printf(" El archivo [ %s ] no existe !          ",name);
        getch();
        gotoxy(5,20);
        printf("                                        ");
        return;
}
rewind(f2);
gotoxy(4,20);
```

```
printf("Leyendo el archivo: [ %s ]",name);

for(t=0;t<15000;t++)  /*transfiere el archivo a un arreglo de
                        datos*/
    song[t]=getc(f2);
gotoxy(4,20);
printf("                                    ",name);
g=1;
fclose(f2);  /*cierra el archivo de ejecución*/

REP:
writword(BRVC,"EJECUTANDO",67,19);
gotoxy(5,20);
printf("<RET>: Para detener ejecución de [ %s ]          ",name);


    while (kbhit())
    getch();                  /*procesa acción del usuario*/
    sendcmd(CLEAR_PCOUNT);    /*borrar los  contadores de
                                ejecución*/
    sendcmd(START_PLAY);
    a=song[0];
    if (a!=11)  /*si el primer byte es diferente de 11h, no
                  ejecuta*/
    {
      gotoxy(5,20);
      printf("                                                    ");
      writword(BRVC,"NO COMPATIBLE CON MIDI V 1.1       ",5,19);
      getch();
      writword(RVC,"                                        ",5,19);
      goto FIN;
    }
    nume=1;
    while (1)
    {
     comando = get401();
     if (comando == -1)
           goto FIN;
     if (comando >= REQ_T1 && comando <= REQ_T8)
     {
           c=0;
           a=song[nume];
           for  (t=0;t<a;t++)
           {
                nume++;
                b=song[nume];
                c=c+b;
                escribir_dato(b);
            }
           nume++;
           if (c==0)
                goto FIN;
      }
      if (comando == ALL_END)
           goto FIN;
    }
```

```
FIN:
writword(BWC,"
",4,19); writword(BWC,"            ",67,19);
sendcmd(STOP_PLAY);
sendcmd(CLEAR_PMAP);
}

/*****************************************************************/

listar() /*opción de listar*/
{
        list(2);
}
/*****************************************************************/

void grabar()  /*graba un track, detiene grabación
                 si se presiona cualquier tecla*/
{
    int primero, segundo, tercero, cuarto, mestado;
    int t;
    mestado = 0;
    while (kbhit())
    getch();                        /*Pocesa acción del usuario*/
    sendcmd(START_REC);
    while (1)
    {
    INIC:

     primero = get401();
     if (primero == -1)
     {                  /*existió presión de una tecla*/
         while (kbhit())
         getch();
         sendcmd(STOP_REC);          /*termina la grabación*/
         primero = get401();
         guardar(4,0,0,0,0,1);
         return;
     }
     if (primero <= 0xEF)
     {                    /*byte de tiempo*/
         segundo = get401();
         if (segundo <= 0x7F)
         {                          /*Se asume byte de estado*/
          tercero = get401();
          guardar(4,primero,mestado,segundo,tercero,0);
         }
         else if (segundo <= 0xBF)
         {            /*mensaje MIDI  de nota presionada /  liberada
          mestado = segundo;  /*after  touch ó  cambio de  control
          tercero = get401();
          cuarto = get401();
          guardar(4,primero,segundo,tercero,cuarto,0);
         }
         else if (segundo <= 0xDF)
         {               /*cambiode programa o after touch*/
          mestado = segundo;
```

```
            tercero = get401();
            guardar(3,primero,segundo,tercero,0,0);
           }
           else if (segundo <= 0xEF)
           {              /*pitch bender*/
            mestado = segundo;
            tercero = get401();
            cuarto = get401();
            guardar(4,primero,segundo,tercero,cuarto,0);
           }
           else if (segundo == 0xF9)
           {               /*byte de fin de compás*/
            guardar(2,primero,segundo,0,0,0);
            goto INIC;
           }
           else if (segundo == 0xFC)
           {              /*final de datos para el track*/
            guardar(2,primero,segundo,0,0,0);
            goto INIC; /*se graba solo un track*/
           }
           else
           {
            printf("\ndato no reconocido %x %x",primero,segundo);
          }
       }
       else if (primero <= 0xF7)      /*pedido de datos de tracks
                                        data track request*/
           printf("\nSe espera Conductor data.");
       else if (primero == 0xFC)
           goto INIC;
       else if (primero == 0xFD)      /*señal de reloj del MPU*/
           printf("\nGot a clock out signal (%x)",primero);
       else if (primero == 0xFE)
       {          /*ignore el acknoledge */
       }
       else
       {
           writerr("Grabación detenida.");
           break;
       }
     }
}
```

- FUNCION PARA LISTAR ARCHIVOS .MID DEL DIRECTORIO

```
#include <stdio.h>
#include <dir.h>
#include "standard.h"

int list(int ac)
```

```
{
    struct ffblk ffblk;
    int n, done, min, hour, day, month, year,q;
    char *files, fdate[9],  ftime[6], drive[MAXDRIVE], dir[MAXDIR],
    file[MAXFILE], ext[MAXEXT];

    n = 1;
    putchar('\n');
    do
    {
      if (ac == 1)
        files = "*.*";
      else
      {
        files ="*.MID";
      }
      done = findfirst(files,&ffblk,0);
      while (!done)
      {
            day = ffblk.ff_fdate&0x001f;
            month = (ffblk.ff_fdate >> 5) & 0x000f;
            year = ((ffblk.ff_fdate >> 9) & 0x007f) + 80;
            sprintf(fdate, "%02d/%02d/%2d", day, month, year);
            min = (ffblk.ff_ftime >> 5) & 0x003f;
            hour = (ffblk.ff_ftime >> 11) & 0x001f;
            sprintf(ftime, "%02d:%02d", hour, min);
            fnsplit(ffblk.ff_name, drive, dir, file, ext);
            gotoxy(5,20);
            printf("%-8s.%-4s %6ld bytes %9s%7s  Q:  Regresar", file,
            ext+1, ffblk.ff_fsize, fdate, ftime);
            q=getch();
            if(toupper(q)==81)
            {
                 gotoxy(5,20);
                 printf("                                                ");
                 goto FIN;
            }
            gotoxy(5,20);
            printf("                                                ");
            done = findnext(&ffblk);
      }
      n++;
    }while (n < ac);
FIN:
}
```

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

BIBLIOGRAFIA

- Banda, Hugo y Stewart, Brian. PROGRAMMING IN C. Dundee University, 1991.

- BORLAND. TURBO C USER'S GUIDE

- CASIO Corporation. CASIO HT-700 DIGITAL SYNTHESIZER USER'S GUIDE.

- Conger, Jim. C PROGRAMMING FOR MIDI. 1ra. ed. NY. M&T Publishing, Inc. 1990.

- Chamberlain, H. MUSICAL APPLICATIONS OF MICROPROCESSORS. NY. Hayden Books. 1985.

- Deutsh, Herbert. SOUND SYNTHESIS. NY. Tab Books, 1983.

- Holmes, Thomas. ELECTRONIC AND EXPERIMENTAL MUSIC. NY. Charles Scribener's Sons, 1987.

- Latraublon, G. MUSICA ELECTRONICA. 2da. ed. Madrid. Editorial Paraninfo, 1982.

- Morgan , Christopher. INTRODUCCION AL MICROPROCESADOR 8086 / 8088. Mexico. Mc Graw Hill, 1986.

- Mathews, Max. THE TECHNOLOGY OF COMPUTER MUSIC. NY. Harmony Books, 1984.

- Mosish, Donna y Shamma, Namir. ADVANCED TURBOC C PROGRAMMER'S GUIDE. NY. John Willey & Son. Inc, 1989.

- Norton, Peter. ASSEMBLY LANGUAGE BOOK FOR THE IBM PC. NY. Prentice Hall. 1986.

- ROLAND Corporation. FUNDAMENTOS DE MUSICA ELECTRONICA CON SINTETIZADORES. Osaka. Japón. 1983.

- ROLAND Corporation. MIDI PROCESSING UNIT MPU-401 TECHNICAL REFERENCE MANUAL.

- Scanlon, Leo. 8086 / 8088 / 80286 ASSEMBLY LANGUAGE. NY. Simon & Schuster, 1988.

- VOYETRA Technologies. V-400X USER'S GUIDE.

- Walker, Don. HOW MIDI WORKS. CA. Peter Alexander Publishing. Inc. 1992.

- YAMAHA Corporation. WHAT'S MIDI?.

REFERENCIAS BIBLIOGRAFICAS

REFERENCIAS BIBLIOGRAFICAS

1.- Holmes, Thomas. ELECTRONIC AND EXPERIMENTAL MUSIC. NY. Charles Scribener's Sons, 1987.

2.- ROLAND Corporation. FUNDAMENTOS DE MUSICA ELECTRONICA CON SINTETIZADORES. Osaka. Japón. 1983.

3.- Deutsh, Herbert. SOUND SYNTHESIS. NY. Tab Books, 1983.

4.- Walker, Don. HOW MIDI WORKS. CA. Peter Alexander Publishing. Inc. 1992.

5.- YAMAHA Corporation. WHAT'S MIDI?.

6.- ROLAND Corporation. MIDI PROCESSING UNIT MPU-401 TECHNICAL REFERENCE MANUAL.

7.- Morgan , Christopher. INTRODUCCION AL MICROPROCESADOR 8086 / 8088. Mexico. Mc Graw Hill, 1986.

8.- BORLAND. TURBO C USER'S GUIDE