Universidad Tecnológica Nacional
Facultad Regional Mendoza

TRABAJO FINAL DE LA CARRERA INGENIERIA EN ELECTRONICA

Titulo: Z80 Development System Kit

Autores: Gabriel Rosso (Legajo: 13883-3)
Rafael Meza (Legajo: 11513-5)

Cátedra: Proyecto Final
Año Lectivo: 1995
Fecha de Finalización: Agosto de 1996
Comisión Evaluadora: Ing. Gustavo Mercado (Prof. Técnicas Digitales II)
Ing. Jorge Abraham (Prof. Técnicas Digitales II)

Colaboraciones:

El proyecto fue realizado con la colaboración técnica y económica del IIACE. El proyecto se realizó, en su mayor parte, el los laboratorios del IIACE usando los instrumentos y la infraestructura del mismo. Los integrantes del proyecto renuncian a percibir cualquier emolumento, salario o beca por parte del IIACE por la realización de presente proyecto. El IIACE recibe, como recompensa a su contribución, toda la información y documentación necesaria para la utilización y reproducción del sistema.

El IIACE permite que el departamento de Electrónica de la UTN FRM reproduzca el sistema, pero sólo con fines didácticos y NO para fines comerciales. Si existiera una finalidad comercial el IIACE, la UTN FRM y los autores deberán discutir la implementación de tal opción. El IIACE permite la divulgación del sistema a través de presentación a congresos, publicaciones y cursos, siempre que se reconozca la contribución del IIACE con una leyenda tal como: TRABAJO SOSTENIDO CON EL APOYO FINANCIERO DE PROYECTO BID-CONICET II NRO. 173 BARREDOR MULTIESPECTRAL AEROTRANSPORTADO - IIACE -.

Elementos físicos que lo componen y distribuyen:

Para el IIACE:

- 1 (una) copia del Manual de Diseño.
- 1 (una) copia del Manual del Usuario.
- 2 (dos) placas de desarrollo con componentes.
- 2 (dos) cables para la interconexión serie con la PC.
- 2 (dos) cables para la alimentación de las placas.
- 2 (dos) juegos de placas para la expansión del sistema con sus cables de interconexión.
- 1 (un) diskette con los programas necesarios para instalar el software de PC.
- 1 (un) diskette con el software de placa.
- 1 (un) diskette con los programas utilizados en la fase de diseño de los softwares de PC y de placa.
- 1 (un) diskette con la versión prototipo del sistema.

Para la UTN FRM:

- 1 (una) copia del Manual de Diseño.
- 1 (una) copia del Manual del Usuario.
- 2 (dos) placas de desarrollo con componentes.
- 2 (dos) cables para la interconexión serie con la PC.
- 2 (dos) cables para la alimentación de las placas.
- 2 (dos) juegos de placas para la expansión del sistema con sus cables de interconexión.
- 1 (un) diskette con los programas necesarios para instalar el software de PC.
- 1 (un) diskette con el software de placa.
- 1 (un) diskette con los programas utilizados en la fase de diseño de los softwares de PC y de placa.
- 1 (un) diskette con la versión prototipo del sistema.

Para los diseñadores del sistema:

- 1 (una) copia del Manual de Diseño.
- 1 (una) copia del Manual del Usuario.

Agradecimientos

Queremos agradecer a todas aquellas personas que hicieron posible que este proyecto salga adelante y muy especialmente al Ing. GUSTAVO MERCADO quien fue nuestro guía en este desafío. A Osvaldo Peinado, Juan Yelós, Ernesto Betman, Daniel Vicare, Jorge Garay, Arturo Maidana, María Edith Mateu, Alejandro Grasso, Héctor Raimondo, Jorge Martínez, Nestor Zandomeni, Carlos Zrimsek y Ricardo Borzotta; en fin, a todo el equipo del IIACE. Gracias!

Z80 DSK
Manual de diseño

Gabriel Rosso
Rafael Meza

Mendoza, Agosto de 1996

1. Indice

1.1. General

1.2. Diagramas

1.3. Figuras

1.4. Tablas

2. Introducción

El proyecto consiste en el diseño e implementación de un sistema para el Desarrollo de Microprocesadores el cual posibilita y facilita el análisis, desarrollo, implementación y verificación de programas y sistemas basados en el microprocesador Z80 y periféricos de la línea Zilog.

El Z80 Development System Kit está conectado a una PC a través del canal de acceso serie, con lo cual se permite el manejo remoto del sistema.

Se pretendió construir un sistema para la enseñanza y práctica de la Arquitectura de Computadoras, disciplina fundamental de la Cátedra Técnicas Digitales II, pero que además sirva para emprender y facilitar diseños basados en microprocesador Z80 por parte de quien así lo desee.

3. Composición

El Z80 Development System Kit consta de dos partes que interactúan entre sí. Una parte está compuesta por la PC la cual alberga un software de PC que permite el control de todo el sistema y por una parte compuesta por una placa de desarrollo que tiene un software de placa para controlar la actividad de la misma y la comunicación con la PC.

3.1. Parte compuesta por la placa de desarrollo

3.1.1. Hardware

3.1.1.1.Introducción

Para el diseño de la placa de desarrollo se tuvo en cuenta que el objetivo de la misma era permitir el aprendizaje de la disciplina Arquitectura de Computadoras por parte del estudiantado de la Cátedra Técnicas Digitales II, por lo que se pretendió darle una distribución funcional a los componentes.

3.1.1.2.Diseño

Para realizar la placa se utilizó el CAD Protel. Este CAD permite el diseño de hardware de una manera muy amigable. Primeramente se desarrollaron los diagramas de conexiones (esquemático) de los componentes que tendría la placa (utilitario Editor de Esquemáticos de Protel). Todos los componentes fueron extraídos de la librería que posee el CAD, no siendo necesario crear ninguno a través del SLM (Schematic Library Manager) de Protel. Una vez que se tuvo el esquemático de la placa deseada, se generaron los archivos que contienen el listado de conexiones de los componentes (archivo con extensión .NET) , la lista de materiales (archivo con extensión .BOM) y la lista con los errores que se cometieron al realizar la interconexión de componentes (archivo con extensión .REP) a través del uso del utilitario POST de Protel.

Con estos archivos se comenzó el diseño de la placa (PCB) con el uso del utilitario PCB de Protel. Previamente a la distribución que actualmente se puede ver en ella, se intentaron otras distribuciones que no prosperaron debido a la densidad de líneas que tiene. Con el PCB diseñado en su totalidad, se ejecutó el comando DRC del utilitario Protel con el objetivo de corregir errores cometidos en el diseño.

Se imprimió una copia del diseño con el uso de una impresora láser y se envió para que se realizase un fotolito y una placa. En esta placa se experimentó con un software prototipo y se hicieron las modificaciones correspondientes al esquemático y al PCB para llegar a la placa que se tiene actualmente.

3.1.1.3.Composición

La placa está compuesta por los siguientes componentes:

```
--------------------------------------------------------------------------------
Bill of Material For dsk-z80   17:58 5-JUN-1996  Page : 1
--------------------------------------------------------------------------------
DESCRIPTION          QUAN.  COMPONENT NAME(S)

                      14    L1         S1         W1         W2         W3
                            W4         W5         W6         W7         W8
                            W9         W10        W11        W12

.1mF                  16    C6         C7         C8         C9         C10
                            C11        C12        C13        C14        C15
                            C16        C17        C18        C19        C20
                            C21

1N4148                 1    D1

4PIN                   3    P1         P2         P3

10K                    3    R1         R2         R3

10mF                   4    C2         C3         C4         C5

47uF                   1    C1

50PIN                  2    J1         J2

74HC08                 1    U16

74HC14                 1    U1

74HC138                2    U7         U14

74HC245                4    U2         U9         U10        U17

2764                   1    U3

6164                   2    U5         U6

DB9                    1    J3

MAX232                 1    U15

OSC                    2    XTAL1      XTAL2

ROM RAM 64             1    U4

Z80ACPU                1    U8

Z80CTC                 1    U13

Z80PIO                 1    U11

Z80SIO0                1    U12
```

Además de los componentes antes mencionados la placa contiene cinco tacos de suspensión para darle rigidez mecánica cuando sea apoyada sobre una mesa o tablero.

3.1.1.3.1. Circuitos Esquemáticos

En el Apéndice C se puede ver los circuitos esquemáticos y PCB layer de la placa de desarrollo.

3.1.1.3.2. Distribución y función de los componentes

3.1.1.3.2.1. Buffers

En la figura del top overlayer de la placa se puede ver la distribución esquemática de los componentes de la placa de desarrollo en su totalidad. Se puede observar que a la izquierda de la misma se encuentra el microprocesador separado del resto por una “ barrera” de buffers, cuya función es la de permitir que el microprocesador funcione en forma correcta

independientemente de lo que se le encuentre "colgado". Estos integrados son bidireccionales, dependiendo del estado de una de las patas (pata 1), por lo que son muy útiles para atender la función del bus de datos del microprocesador.

Tres de los cuatro buffers (U2, U9 y U17) son unidireccionales por lo que no se requiere lógica para su control, pero el buffer U10 (buffer del bus de datos) requiere de la implementación de una lógica para lograr el control del sentido del flujo de la información. Esta lógica está basada en el estudio de las distintas señales que se encuentran involucradas en las diversas operaciones del microprocesador; de todas ellas, se utilizan las siguientes:

- M1
- RD
- WR

ya que son las comunes a todas las operaciones.

El estado de estas señales en el sentido microprocesador - resto de la placa es la siguiente:

Operación	M1	RD	WR
Write	1	1	0
Output	1	1	0

Tabla 1: Estado de las señales micro - placa

y el estado de estas señales en el sentido resto de la placa - microprocesador es la siguiente:

Operación	M1	RD	WR
Instruction Fetch	0	0	1
Read	1	0	1
Input	1	0	1
Interrupt Acknowledge	0	1	1

Tabla 2: Estado de las señales placa - micro

De las dos tablas anteriores se puede deducir que para lograr el sentido de dirección basta combinar en una compuerta AND las señales M1 y RD; y para lograr la habilitación del integrado es necesario combinar en otra compuerta AND la señal WR y salida de la compuerta AND anteriormente detallada.

3.1.1.3.2.2.Mapa de memoria

En la parte superior de la placa se puede observar el banco de memoria, que fija la capacidad del sistema a 32 Kbytes (si no se usa el bus de expansión de memoria) juntamente con el integrado que permite organizar el mapa de memoria.

El mapa de memoria es el siguiente:

0000 - 1FFF	ROM 0
2000 - 3FFF	ROM 1 - RAM 0
4000 - 5FFF	RAM 1
6000 - 7FFF	RAM 2
8000 - 9FFF	Externo
A000 - BFFF	Externo
C000 - DFFF	Externo
E000 - FFFF	Externo

Tabla 3: Mapa de Memoria

La memoria está diseñada para que pueda elegirse a voluntad la cantidad de memoria no volátil que se desea que tenga la placa. Esto puede elegirse a través del jumper W5 que conmuta entre VCC y la señal WR, permitiendo así la selección.

La conformación de este mapa de memoria se logra con el decodificador U7, el cual posee en su entrada los tres bits más altos del bus de direcciones.

3.1.1.3.2.3.Mapa de entrada/salida

En la parte inferior de la placa de desarrollo se encuentran los integrados que la comunican con el exterior, o sea los puertos, tanto paralelo (Z80 PIO) como serie (Z80 SIO). A ellos se les une el integrado que facilita la temporización del puerto serie a la velocidad de comunicación que se desee (Z80 CTC).

El mapa de entrada salida es el siguiente:

00 - 0F	SIO
10 - 1F	CTC
20 - 2F	PIO
30 - 3F	Externo
40 - 4F	Externo
50 - 5F	Externo
60 - 6F	Externo
70 - 7F	Externo
80 - FF	No implementado

Tabla 4: Mapa de Entrada/Salida

De este mapa se puede observar que se encuentran implementados solamente la mitad de los puertos que el microprocesador es capaz de manejar. Si se desea ampliar este numero se deberá implementar externamente la lógica para su conformación.

La conformación de este mapa de entrada/salida está implementada a través del decodificador U14 que utiliza como entradas los cuatro bits más altos de la primera mitad del bus de direcciones.

3.1.1.3.2.4.Extensión del sistema

Se pretendió que la placa fuese lo suficientemente flexible como para que a través de su uso se pudiese analizar e implementar sistemas puntuales diseñados por el usuario. Para lograrlo se implementaron un par de buses de expansión de 50 pines cada uno, lo que permite que el usuario utilice toda la potencialidad del sistema a través del uso de todas las señales involucradas en la placa de desarrollo, para que elija la que más se adecue a sus necesidades; y se distribuyeron un conjunto de jumpers con el objetivo de seleccionar el modo de operación de la placa (extender y/o el mapa de memoria, extender el mapa de puertos, exteriorizar o interiorizar señales de control del microprocesador).

3.1.1.3.2.5.Esquema de interrupciones

Del esquema anterior se puede ver que el SIO es el periférico de más alta prioridad y que luego del PIO se puede conformar un esquema propio de interrupciones según las necesidades de una aplicación.

3.1.1.3.2.6.Interface serie

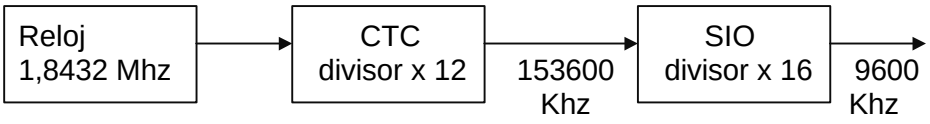
Para lograr la comunicación serie de la placa con la PC se utilizó el canal A del SIO, configurándolo para que opere a una velocidad de transmisión de 9600 bps, para lo cual se utilizó un oscilador a cristal de 1,8432 Mhz en conjunto con un temporizador programable (CTC). Se utilizó un cristal de esta frecuencia ya que permite, a través de una división por un número entero, obtener las distintas frecuencias de transmisión.

El SIO se lo utilizó en el modo X16 (esto quiere decir que transmite a una frecuencia correspondiente a la 16ava parte de la frecuencia del reloj que se le coloque. La ventaja de usar este modo radica en que no se necesita implementar una sincronización para la comunicación.

El CTC se lo utiliza en el modo contador, utilizando una constante de operación igual a 12 (decimal).

Con estos tres valores se consigue la frecuencia de transmisión de 9600 bps.

Todo lo anterior se puede ver en el siguiente gráfico:



3.1.2. Software de la placa

La placa de desarrollo del Z80 Development System Kit tiene en su parte baja de memoria (0-1FFFH) un software de placa que permite que la misma se comunique con la PC y realice las operaciones que esta le determine. Este software está escrito en lenguaje ensamblador y se utilizó para su desarrollo el editor de texto del lenguaje de programación Quick Basic.

Se pretendió que el diseño del mismo fuese lo más sencillo y corto posible, delegando las tareas complejas al software de PC.

Antes de explicar el programa en sí mismo se debe dar una reseña de los objetivos pretendidos al momento de su desarrollo.

3.1.2.1.Objetivos del software de placa

El objetivo del software de placa del Z80 Development System Kit es el de administrar la operación de la placa de desarrollo. Dentro de esta administración podemos citar que se debe encargar de:

- inicializar los componentes que utilice el sistema en la placa.
- limpiar el contenido de memoria que es aleatorio al momento de dar tensión al sistema.
- atender la comunicación entre la placa y la PC a través del puerto serie.
- llevar a cabo los requerimientos que la PC le solicite.

3.1.2.2.Diseño

Previo al comienzo del desarrollo se debió establecer la forma en que se comunicaría la placa con la PC, o sea determinar el protocolo de comunicación. Para diseñar el protocolo de comunicación se fijaron las operaciones básicas relacionadas con el sistema. Las operaciones básicas que se pautaron son las siguientes:

- Llenar un bloque de memoria con datos
- Enviar el contenido de un bloque de memoria
- Programar un puerto
- Enviar el contenido de los puertos
- Inicializar el contenido de los registros
- Enviar el contenido de los registros
- Ejecutar un programa

- Indicar si se produjo un error en la comunicación
- Indicar si se pulsó el switch de reset

Con estas operaciones básicas y con la intención de simplificar el software escrito en lenguaje ensamblador se llegó a la conclusión de que lo más conveniente sería establecer un protocolo para comunicación de datos en binario, dado que si se hacía con los datos representados a través de su código ASCII, el software de placa se debería encargar de la conversión a binario.

El protocolo de comunicación se respeta en la comunicación PC - placa, ya que cuando se trata de la comunicación placa - PC, la placa se limita a enviar los datos solicitados. El protocolo de comunicación es el siguiente:

Cantidad Bytes + Comando + Cantidad Datos + Posición Memoria Inicial + Posición Memoria Final + Posición Memoria Nueva Posición + Datos + Suma Chequeo

Significado de los parámetros:

- **Cantidad Bytes** le indica a la placa la cantidad de bytes que vienen detrás de el para que se prepare a recibirlos.
- **Comando** le indica la operación que debe realizar.
- **Cantidad Datos** le indica si existe información en el campo "Datos" y su cantidad.
- **Posición Memoria Inicial** le indica el valor de posición inicial de memoria pretendida para la ejecución del comando.
- **Posición Memoria Final** le indica el valor de posición final de memoria pretendida para la ejecución del comando.
- **Posición Memoria Nueva Posición** le indica el valor de la nueva posición de memoria pretendida para la ejecución del comando.
- **Datos** contiene los datos que se utilizaran en el comando
- **Suma Chequeo** contiene el complemento a dos de la suma de los bytes anteriores para chequear la integridad de lo que se le ha comunicado.

Este protocolo se respeta siempre, sin importar el comando del que se trata, ya sea porque ocupa ciertos, todos o ningún parámetro.

La comunicación serie se realiza de manera asíncrona, con 8 bits de datos, dos bits de stop, paridad par y con una velocidad de 9600 bps.

3.1.2.3.Filosofía del diseño

Una vez definidos los dos puntos anteriores (comandos y protocolo de comunicación) se comenzó con la confección del programa propiamente dicho. El primer punto a tratar fue establecer la forma en que operaria dinámicamente la placa de desarrollo a través de su software. Esto se puede ver a través del siguiente diagrama de flujo:

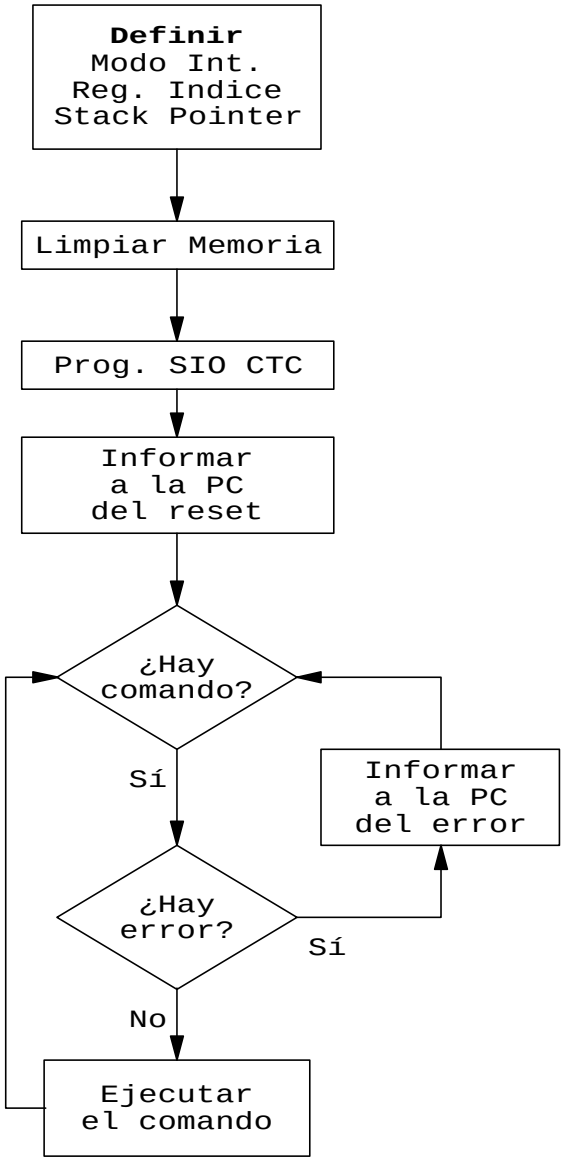


Diagrama 1: Funcionamiento global del software de placa

Básicamente la operación que realiza el software de placa es la siguiente:

El software de placa inicializa la placa de desarrollo y se queda en un ciclo ocioso (HALT) esperando que se reciba un carácter por el puerto serie. El puerto serie está programado para que interrumpa con el primer carácter que reciba. Ese carácter recibido indica (de acuerdo al protocolo) la cantidad de caracteres que vienen detrás de él. Con este dato se almacenan en una tabla todos los bytes indicados, para luego extraer de la misma el comando solicitado por la PC y los parámetros del mismo. Se ejecuta el comando y se retorna al ciclo ocioso (HALT) en espera de un nuevo comando.

Dada esta filosofía de funcionamiento, se confeccionó el programa total en forma modular, donde cada módulo esta relacionado con la función que cumple dentro del software completo. Así podemos ver que el software de placa Z80-DSK.HEX se halla compuesto por los siguientes módulos:

- COMIENZO: inicialización del sistema.
- COMANDOS: rutinas de todos los comandos del software de placa.
- DATOS: definición de las constantes y variables utilizadas.

- ERRORES: atención de los errores que suceden en la placa.
- MJO-INT: rutina que atiende las interrupciones que le hacen al SIO los comandos que llegan desde la PC.
- PROG-SIO: rutina con las distintas programaciones que se le hace al SIO.
- RX-CMD: recepción de los comandos provenientes de la PC y ejecución de los mismos.
- RETCONTR: almacenar el contenido de los registros de la CPU para regresar al control de la placa por el software de PC.

Los ocho módulos anteriores se encadenan para generar un archivo con el nombre Z80-DSK.HEX, el cual contiene el software de placa en forma completa y en formato hexadecimal, listo para ingresarlo a una memoria EPROM.

NOTA: el encadenamiento de los 8 módulos está realizado de acuerdo al orden dado en la explicación anterior.

3.1.2.4.Descripción de los módulos

A continuación se dará una explicación tendiente a que se comprenda la filosofía del diseño de cada uno de los módulos. Se intentará que la explicación sea lo más transparente posible y sin ahondar en detalles, ya que creemos que los mismos se pueden ver en el módulo propiamente dicho. La explicación de cada módulo irá acompañada de su diagrama de flujo si el mismo contribuye a su mejor comprensión y de la porción del programa que lo compone. Los listados completos se pueden ver utilizando el diskette que se provee con los módulos del diseño.

3.1.2.4.1.COMIENZO

Este módulo es el primer módulo que se escribió, ya que contiene la inicialización del sistema. La inicialización consiste en definir los valores que tendrán los registros elementales para el funcionamiento del software: la elección del modo de interrupción que se utilizará, la limpieza de la memoria asignada para el uso del usuario ya que posee valores aleatorios y la elección del modo de operación de los periféricos que utiliza el sistema para su funcionamiento (SIO y CTC).

A continuación se describe a través de un diagrama de flujo la estructura de este módulo.

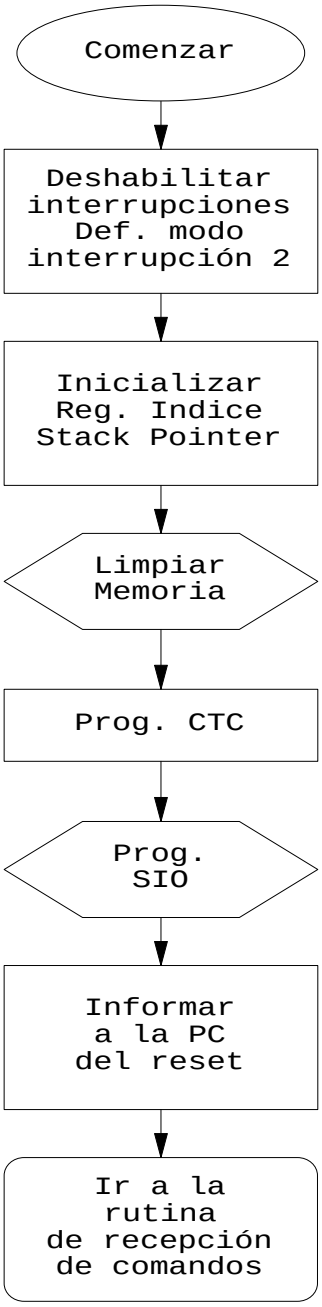


Diagrama 2: Comienzo

y el programa que compone este módulo es el siguiente:

```
                    ORG 0

COMIENZO        DI                      ;deshabilitacion de interr.
                IM 2                    ;eleccion del modo de int.
                LD A, CERO              ;registro I para modo de
                LD I, A                 ;interrupcion 2
                LD IX, PUNTERO_DE_PILA
                LD SP, IX
                JP INICIALIZACION

                ORG 90

                DW BUFFER_TX_VACIO
                DW CAMBIO_ESTADO_EXTERNO
                DW CARACTER_DISPONIBLE
                DW CONDICION_RX_ESPECIAL
```

```
                ORG 100
  ;inicializacion de los perifericos: CTC, SIO.

INICIALIZACION  CALL LIMPIAR_MEMORIA          ;limpiamos el contenido de
                                              ;la memoria RAM

                LD A, CONTROL_1_CTC           ;interrupcion deshabilitada
                                              ;modo contador, flanco
                                              ;ascendente, sigue constante
                                              ;de tiempo, operacion
                                              ;continua, palabra de control
                OUT (PUERTO_CTC_CH0_C),A      ;canal 0
                LD A, CONSTANTE_TIEMPO_CTC    ;valor de la constante
                                              ;de tiempo para Tx y Rx
                                              ;en 9600 baudios
                OUT (PUERTO_CTC_CH0_C),A
                CALL INICIALIZACION_SIO       ;inicializamos el chip que con-
                                              ;trola la comunicacion serie
                LD A, CMD_RESET
                CALL ENVIAR_BYTE              ;indicamos a la PC que la placa
                                              ;se encuentra lista para traba-
                                              ;jar
                JP RECEPCION_COMANDOS         ;esperamos un comando desde la
                                ;PC
```

Este módulo contiene además el procedimiento Limpiar Memoria, cuya función es la inicializar el contenido del bloque de memoria asignado para uso del usuario.

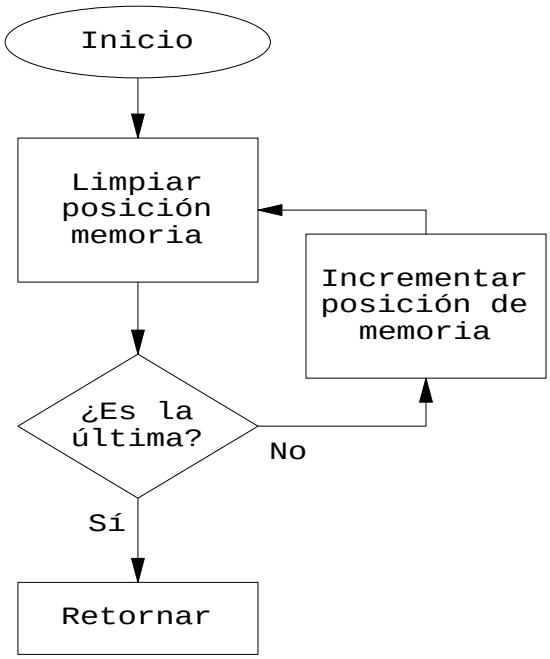


Diagrama 3: Limpiar Memoria

```
LIMPIAR_MEMORIA LD DE, COMIENZO_MEMORIA_USUARIO
LOOP_1          LD HL, FINAL_MEMORIA_USUARIO
                LD A, CERO                    ;futuro contenido de la memoria
                LD (DE), A                    ;inicializamos la posicion de
                                              ;memoria
                INC DE                        ;posicion de memoria siguiente
                AND A                         ;carry reset
                SBC HL, DE                    ;llego al final?
                JP NZ, LOOP_1                 ;si no llego, inicializar otra
                                              ;posicion de memoria
                RET
```

3.1.2.4.2.COMANDOS

Este módulo esta compuesto por: las rutinas que atienden la ejecución de los comandos ordenados por el software de PC, por una rutina que se encarga de obtener los valores que vienen en el protocolo de comunicación y por la rutina que permite el envío de datos byte a byte por el puerto serie. Las rutinas son las siguientes:

- **Llenar bloque**: este comando se utiliza para llenar un bloque de memoria con el contenido que establezca la PC. Utiliza dos parámetros, uno es la posición de memoria a partir de la cual debe comenzar a llenar con el contenido establecido y el otro son los datos o contenido pretendido para el bloque de memoria. Para llevar a cabo el comando se utiliza un ciclo de llenado en el cual cada vez que se llena una posición de memoria se comprueba si hay mas datos para cargar en la memoria, de cuyo resultado dependerá que se siga llenando la memoria o que se finalice la operación.

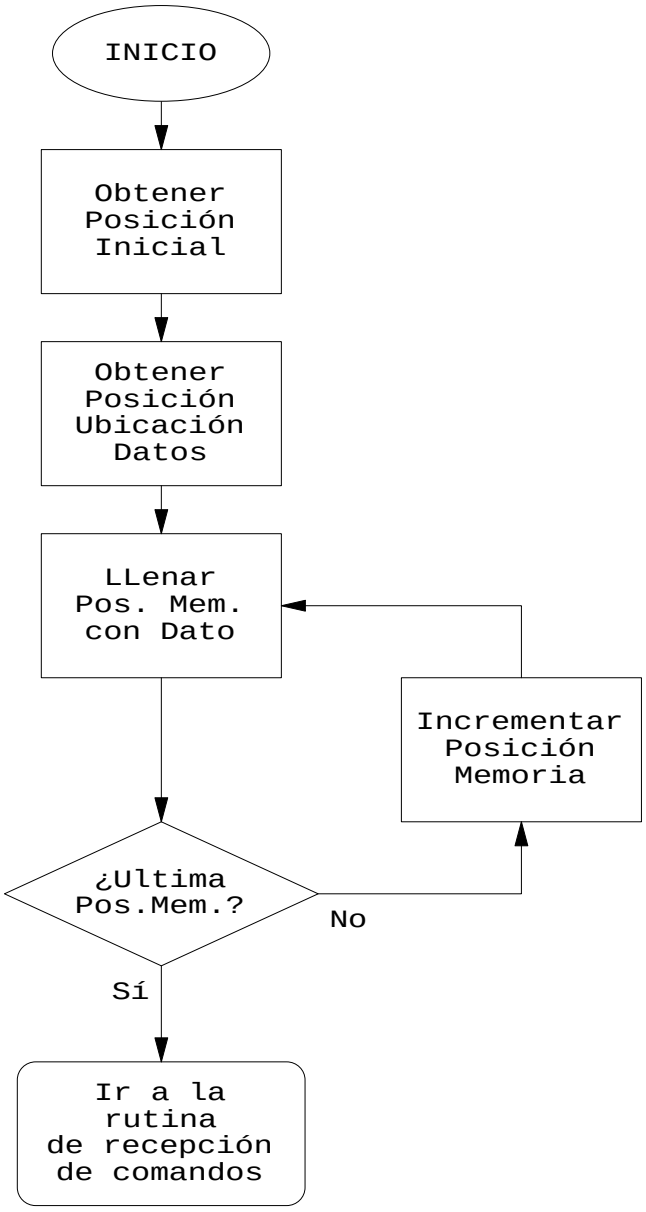


Diagrama 4: Llenar bloque de memoria

```
LLENAR_BLOQUE    LD IX, (POSICION_MEMORIA_INICIAL)
                  ;almacenamos la posicion inicial a partir de la cual
                  ;llenaremos la memoria
                 LD IY, (POSICION_MEMORIA_UBICACION_DATOS)
                  ;almacenamos la posicion inicial a partir de la cual
                  ;se encuentran los datos con que llenaremos la memoria
```

```
                LD A, (CANTIDAD_DATOS)
                LD B, A
                 ;cantidad de datos a guardar
LOOP_1          LD A, (IY)
                 ;byte a transferir
                LD (IX), A
                 ;almacenamiento del byte en el destino
                INC IX
                INC IY
                 ;incrementar las posiciones de memoria fuente y destino
                DJNZ LOOP_1
                 ;repetir hasta que la cantidad de datos sea cero
                LD A, CMD_EJECUCION_OK
                CALL ENVIAR_BYTE
                JP RECEPCION_COMANDOS
```

- **Enviar bloque**: este comando se utiliza para enviar el contenido de un bloque de memoria a la PC. Utiliza dos parámetros, uno es la posición de memoria inicial y el otro es la posición de memoria final del bloque a enviar. Para llevar a cabo este comando se envía el contenido de la posición de memoria inicial del bloque a enviar y se continua enviando los contenidos de las subsiguientes posiciones de memoria hasta alcanzar la posición de memoria final del bloque deseado.

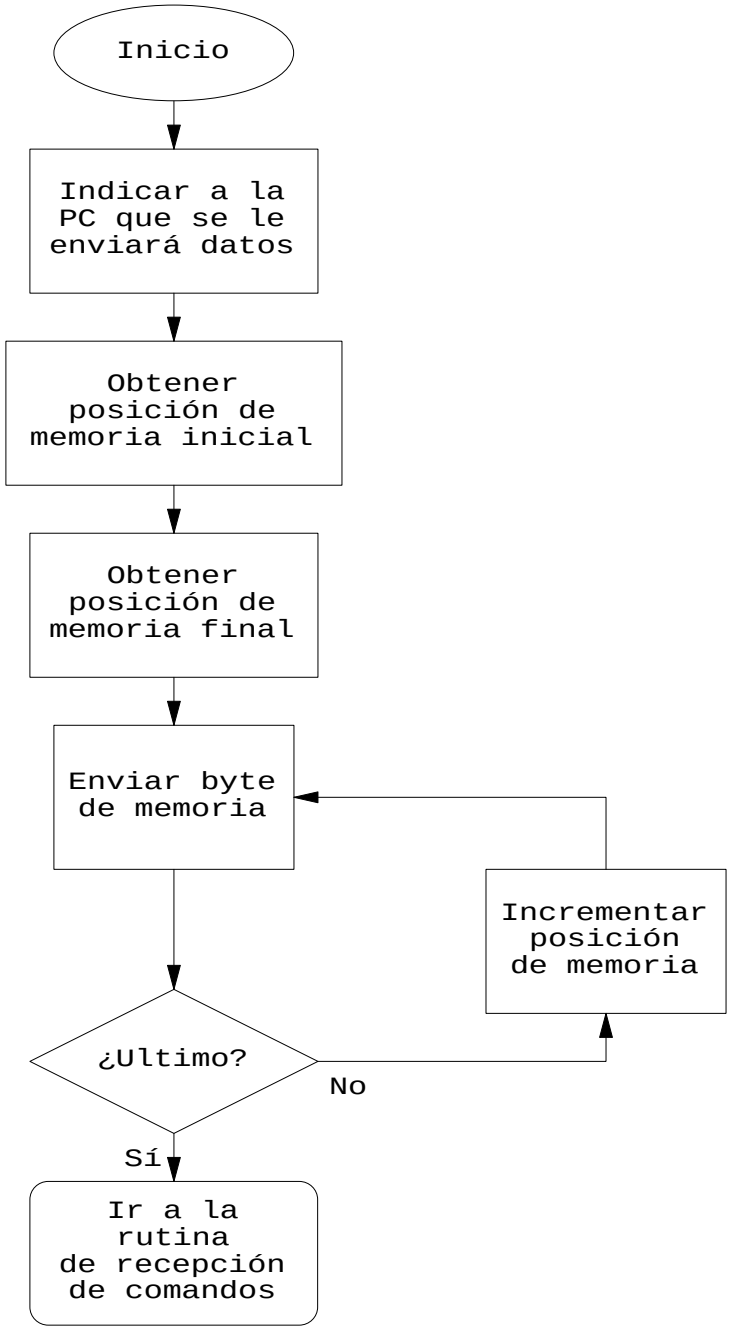


Diagrama 5: Enviar bloque de memoria

```
ENVIAR_BLOQUE    LD A, CMD_DATOS        ;indicar a la PC que se enviara datos
                 CALL ENVIAR_BYTE
                 LD BC, (POSICION_MEMORIA_INICIAL)
                  ;principio de la memoria fuente
LOOP_2           LD HL, (POSICION_MEMORIA_FINAL)
                  ;fin de la memoria fuente
                 LD A, (BC)
                  ;byte de la memoria fuente
                 CALL ENVIAR_BYTE
                  ;envio del byte por el puerto serie
                 AND A
                  ;carry reset
                 SBC HL, BC
                  ;fin de la memoria fuente - posicion en la memoria fuente
                 JP Z, FIN_1
                  ;si llego al final de la memoria fuente retornar
                 INC BC
```

```
                ;incrementar la posicion en la memoria fuente
               JP LOOP_2
                ;enviar otro byte
FIN_1          JP RECEPCION_COMANDOS
```

- **Inicializar registros**: este comando se utiliza para llenar las variables registros con los valores establecidos por el usuario. Utiliza un parámetro; son los Datos, que se utilizan para programar las variables registros una por una.

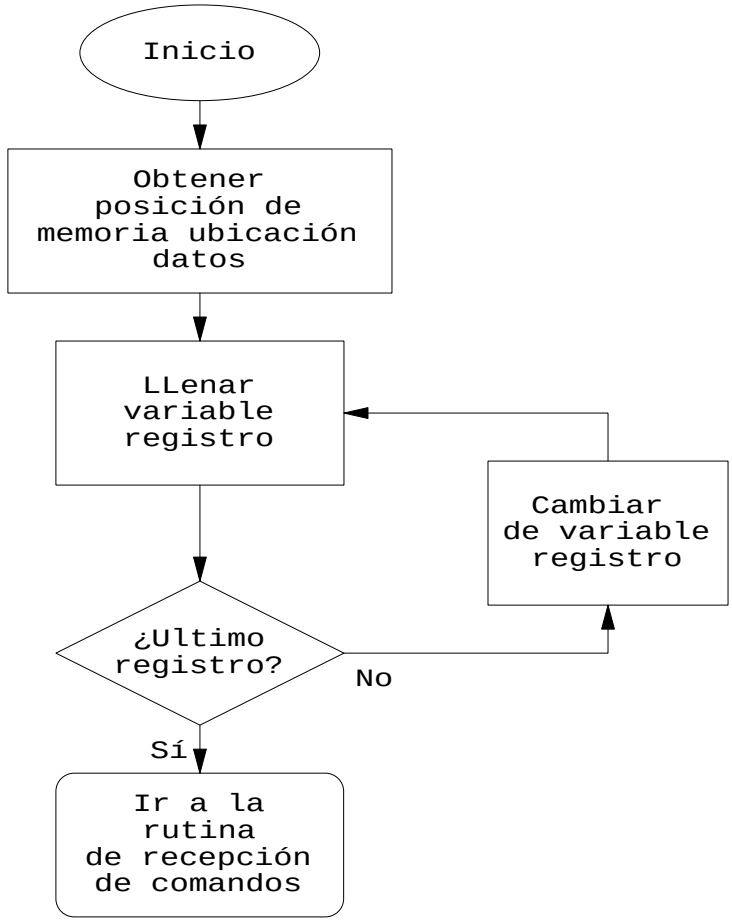


Diagrama 6: Inicializar Registros

```
INICIALIZAR_REGISTROS   LD IX, (POSICION_MEMORIA_UBICACION_DATOS)
                         ;posicion de memoria donde comienza la tabla
                         ;de datos provenientes de la PC
                        LD A, (IX)
                        LD (REGISTRO_A), A
                         ;contenido para el Acumulador
                        INC IX
                         ;siguiente byte
                        LD A, (IX)
                        LD (REGISTRO_B), A
                         ;contenido para el registro B
                        INC IX
                         ;siguiente byte
                        LD A, (IX)
                        LD (REGISTRO_C), A
                         ;contenido para el registro C
                        INC IX
                         ;siguiente byte
                        LD A, (IX)
                        LD (REGISTRO_D), A
                         ;contenido para el registro D
                        INC IX
                         ;siguiente byte
                        LD A, (IX)
                        LD (REGISTRO_E), A
```

```
 ;contenido para el registro E
INC IX
 ;siguiente byte
LD A, (IX)
LD (REGISTRO_F), A
 ;contenido para las banderas
INC IX
 ;siguiente byte
LD A, (IX)
LD (REGISTRO_H), A
 ;contenido para el registro H
INC IX
 ;siguiente byte
LD A, (IX)
LD (REGISTRO_L), A
 ;contenido para el registro L
INC IX
 ;siguiente byte
LD A, (IX)
LD (REGISTRO_I), A
 ;contenido para el registro I
INC IX
 ;siguiente byte
LD A, (IX)
LD (REGISTRO_R), A
 ;contenido para el registro R
INC IX
 ;siguiente byte
LD A, (IX)
LD (REGISTRO_IX), A
INC IX
LD A, (IX)
LD (REGISTRO_IX+1),A
 ;contenido para el registro IX
INC IX
 ;siguiente byte
LD A, (IX)
LD (REGISTRO_IY), A
INC IX
LD A, (IX)
LD (REGISTRO_IY+1),A
 ;contenido para el registro IY
INC IX
 ;siguiente byte
LD A, (IX)
LD (REGISTRO_SP), A
INC IX
LD A, (IX)
LD (REGISTRO_SP+1),A
 ;contenido para el puntero de pila
INC IX
 ;siguiente byte
LD A, (IX)
LD (REGISTRO_PC), A
INC IX
LD A, (IX)
LD (REGISTRO_PC+1),A
 ;contenido para el contador de programa
LD A, CMD_EJECUCION_OK
CALL ENVIAR_BYTE
JP RECEPCION_COMANDOS
```

- **Examinar registros**: este comando se utiliza para enviar el contenido de las variables registros a la PC. Este comando no utiliza ningún parámetro.

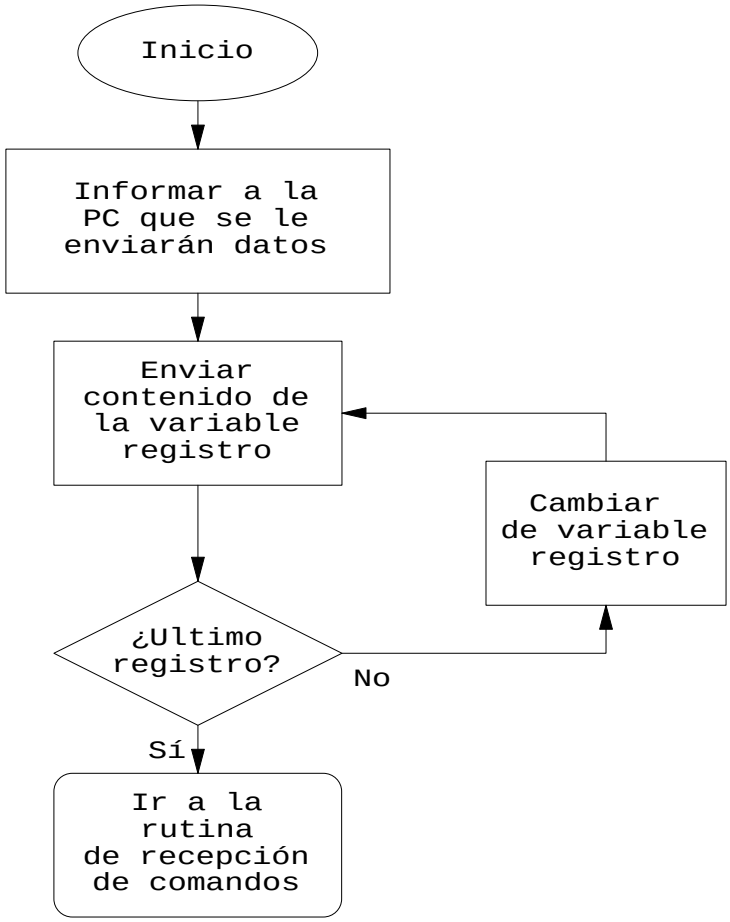


Diagrama 7: Examinar Registros

```
EXAMINAR_REGISTROS      LD A, CMD_DATOS         ;indicar a la PC que se
                        CALL ENVIAR_BYTE        ;enviaran datos
                        LD A, (REGISTRO_A)
                        CALL ENVIAR_BYTE
                        LD A, (REGISTRO_B)
                        CALL ENVIAR_BYTE
                        LD A, (REGISTRO_C)
                        CALL ENVIAR_BYTE
                        LD A, (REGISTRO_D)
                        CALL ENVIAR_BYTE
                        LD A, (REGISTRO_E)
                        CALL ENVIAR_BYTE
                        LD A, (REGISTRO_F)
                        CALL ENVIAR_BYTE
                        LD A, (REGISTRO_H)
                        CALL ENVIAR_BYTE
                        LD A, (REGISTRO_L)
                        CALL ENVIAR_BYTE
                        LD A, (REGISTRO_I)
                        CALL ENVIAR_BYTE
                        LD A, (REGISTRO_R)
                        CALL ENVIAR_BYTE
                        LD BC, (REGISTRO_IX)
                        LD A, B
                        CALL ENVIAR_BYTE
                        LD A, C
                        CALL ENVIAR_BYTE
                        LD BC, (REGISTRO_IY)
                        LD A, B
                        CALL ENVIAR_BYTE
                        LD A, C
                        CALL ENVIAR_BYTE
                        LD BC, (REGISTRO_SP)
                        LD A, B
                        CALL ENVIAR_BYTE
```

```
                    LD A, C
                    CALL ENVIAR_BYTE
                    LD BC, (REGISTRO_PC)
                    LD A, B
                    CALL ENVIAR_BYTE
                    LD A, C
                    CALL ENVIAR_BYTE
                    JP RECEPCION_COMANDOS
```

- **Llenar puertos**: este comando se utiliza para programar el puerto señalado por el usuario con el valor que éste determine. Utiliza un parámetro; son los Datos, de donde se extrae el puerto a programar y el contenido que se desea que tenga.

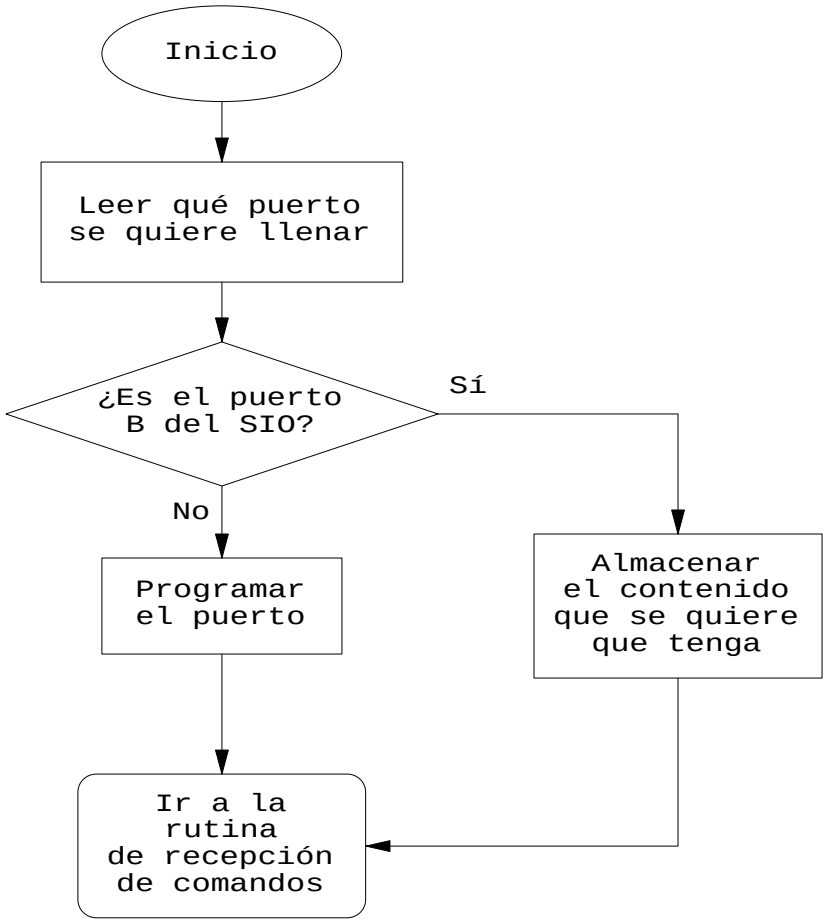


Diagrama 8: Llenar Puertos

```
LLENAR_PUERTOS      LD IY, (POSICION_MEMORIA_UBICACION_DATOS)
                     ;posicion donde se encuentran los datos para
                     ;proceder a llenar el puerto deseado.
                     ;el primer byte corresponde al numero del puerto
                     ;que se desea escribir, el segundo byte corresponde
                     ;al contenido que se quiere que tenga el puerto
                    LD A, (IY)
                    CP PUERTO_SIO_CB
                     ;verificar si se esta escribiendo al puerto B del SIO
                    JP Z, SALTAR
                    LD C, (IY)
                    LD A, (IY+1)
                    OUT (C), A
                     ;escribir el puerto
                    JP FIN_3
SALTAR              LD A, (IY+1)
                    LD (CONTENIDO_DEL_PUERTO), A
                     ;almacenar el contenido con el que se quiere llenar
                     ;el puerto deseado
FIN_3               LD A, CMD_EJECUCION_OK
                    CALL ENVIAR_BYTE
```

```
JP RECEPCION_COMANDOS
```

Este comando programa directamente el puerto solicitado a menos que el mismo sea el puerto que utiliza el sistema para la comunicación serie (puerto B), en cuyo caso se almacena temporariamente el contenido establecido para este puerto, para luego utilizarlo en el comando Enviar Puertos. Esto se debe a que no se permite la programación de este puerto, solamente se permite la lectura de sus registros internos.

- **Enviar puertos**: este comando se utiliza para enviar el contenido de los puertos a la PC. Utiliza un parámetro; son los Datos, de donde extrae la cantidad de puertos que se debe enviar.

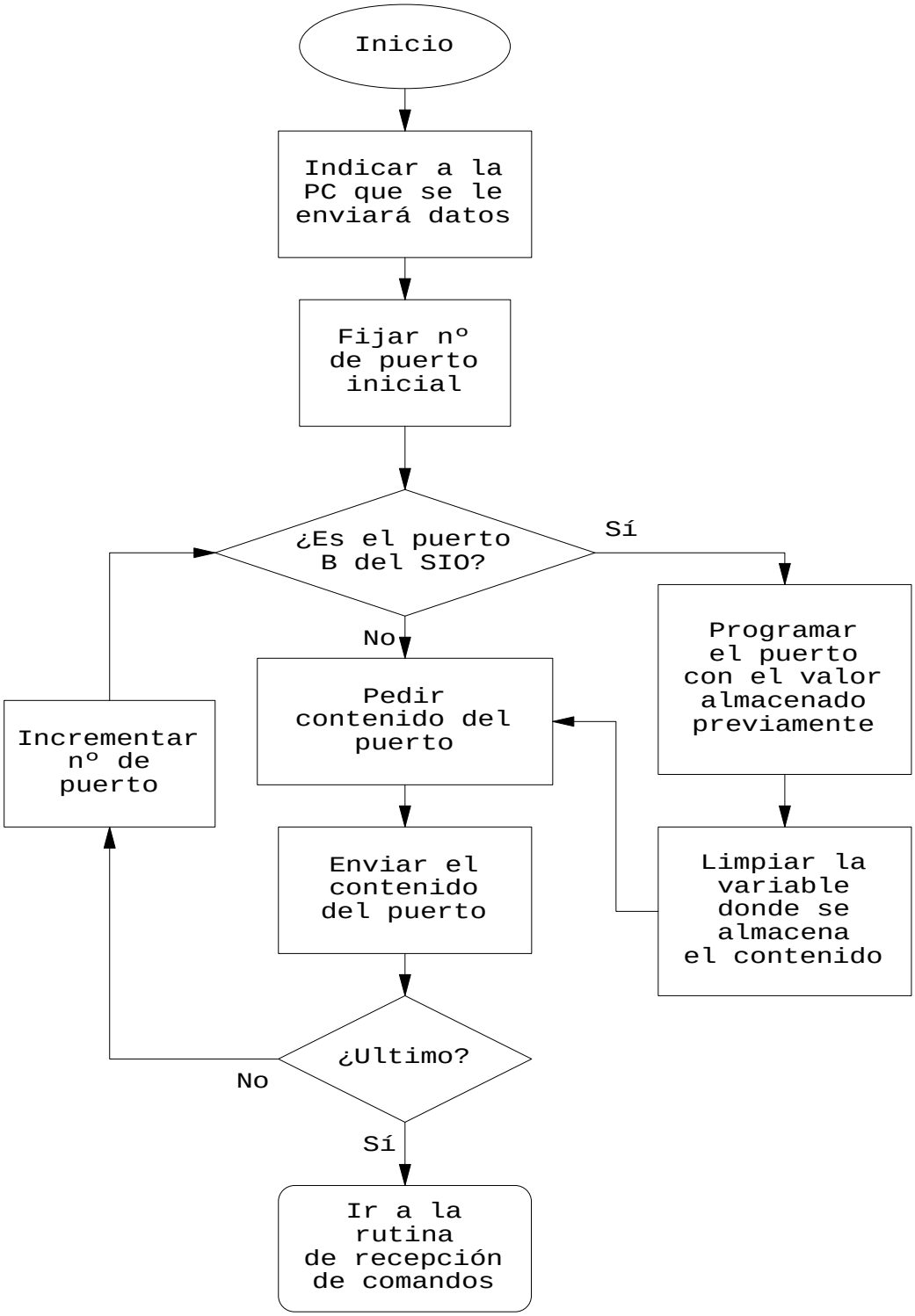


Diagrama 9: Enviar Puertos

```
ENVIAR_PUERTOS          LD A, CMD_DATOS         ;indicar a la PC que se
                        CALL ENVIAR_BYTE        ;enviaran datos
                        LD IY, (POSICION_MEMORIA_UBICACION_DATOS)
                    LD A, (IY)
                           ;cantidad de puertos a enviar
                        ADD A, 1
                        LD B, A
                         ;cantidad de puertos a leer.
                         ;se agrega uno mas por la forma de realizar el
                         ;lazo de envio.
```

```
                    LD D, CERO
                     ;numero de puerto a leer y enviar.
LOOP_4              LD A, PUERTO_SIO_CB
                    CP D
                     ;verificar si se esta por leer el canal B del SIO
                     ;(control)
                    JP NZ, SEGUIR
                     ;indicar al SIO que se realizara una operacion
                     ;de lectura de sus registros internos
                    LD A, (CONTENIDO_DEL_PUERTO)
                    OUT (PUERTO_SIO_CB), A
                     ;enviar el puntero al SIO
                    LD A, CERO
                    LD (CONTENIDO_DEL_PUERTO), A
SEGUIR              LD C, D
                    IN A, (C)
                    CALL ENVIAR_BYTE
                    INC D
                    DJNZ LOOP_4
                    JP RECEPCION_COMANDOS
```

Este comando envía directamente el contenido de los puertos a menos que el puerto que se este por enviar sea el que utiliza el sistema para la comunicación serie (puerto B), en cuyo caso lo programa previamente con el valor almacenado en el comando Llenar Puertos para luego enviar su contenido. Esta operación permite la lectura del los registros RR0, RR1 y RR2 del puerto B del SIO.

- **Ejecutar**: este comando se utiliza para la ejecución de los programas a partir de la posición de memoria que desee el usuario. No utiliza ningún tipo de parámetro.

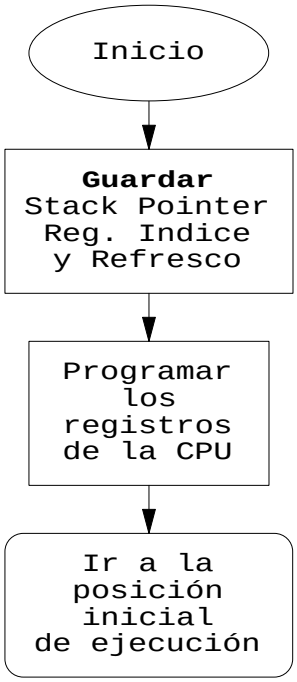


Diagrama 10: Ejecutar

```
EJECUTAR            LD (VALOR_PREVIO_SP), SP
                     ;almacenar el valor que tiene el registro SP
                    LD A, I
                    LD (VALOR_PREVIO_I), A
                     ;almacenar el valor que tiene el registro I
                    LD A, R
                    LD (VALOR_PREVIO_R), A
                     ;almacenar el valor que tiene el registro R
                    ;se almacenan estos registros para que el sistema
                    ;pueda seguir operando sin problemas.
                    ;inicializar el contenido de los registros
                    ;antes de ejecutar
```

```
                    LD A, (REGISTRO_I)
                    LD I, A
                     ;inicializar el registro I
                    LD A, (REGISTRO_R)
                    LD R, A
                     ;inicializar el registro R
                    ;inicializar el acumulador y las banderas
                    LD A, (REGISTRO_A)
                    LD B, A
                    LD A, (REGISTRO_F)
                    LD C, A
                    PUSH BC
                    POP AF
                    ;inicializar es Stack Pointer
                    LD SP, (REGISTRO_SP)
                    ;guardar en el SP la direccion de ejecucion
                    LD BC, (REGISTRO_PC)
                    PUSH BC
                    ;inicializar los demas registros
                    LD A, (REGISTRO_B)
                    LD B, A
                    LD A, (REGISTRO_C)
                    LD C, A
                    LD A, (REGISTRO_D)
                    LD D, A
                    LD A, (REGISTRO_E)
                    LD E, A
                    LD A, (REGISTRO_H)
                    LD H, A
                    LD A, (REGISTRO_L)
                    LD L, A
                    LD IX, (REGISTRO_IX)
                    LD IY, (REGISTRO_IY)
                    LD A, (REGISTRO_A)
                    ;poner en el contador de programa el contenido del
                    ;REGISTRO_PC
                    RET
```

Este comando almacena los valores que tienen los registros SP, I y R, para que el software de PC sea capaz de recuperar el control de la placa, luego programa los registros de la CPU con los valores de las variables registros y finalmente bifurca a la posición de memoria establecida por el contenido de la variable registro PC.

- **Insertar bifurcación**: este comando se utiliza para ensamblar en la memoria la dirección a la cual debe bifurcar el programa que usa el usuario para permitir que el software de PC recupere el control de la placa. Este comando se usa en la Ejecución Paso A Paso del programa del usuario. Utiliza el parámetro que determina la posición de memoria en la que se quiere ensamblar la dirección de retorno controlado.

```
INSERTAR_BIFURCACION    LD HL, RETORNO_CONTROLADO
                         ;obtener la posicion de memoria donde se
                         ;encuentra el comienzo de la rutina que
                         ;permite que la PC tenga de nuevo el control
                         ;de la placa
                        LD IX, (POSICION_MEMORIA_INICIAL)
                         ;recuperar la posicion de memoria a partir de
                         ;dende se ubicara la instruccion
                         ;CALL RETORNO_CONTROLADO que permite el retorno
                         ;controlado hacia la PC
                        LD (IX+1), L
                        LD (IX+2), H
                         ;insertar la constante en la memoria
                         ;en la posicion indicada por (IX) va el codigo de
                         ;la instruccion CALL NN o sea CD
                        LD A, CMD_EJECUCION_OK
                        CALL ENVIAR_BYTE
                        JP RECEPCION_COMANDOS
```

- **Obtener Valores**: esta rutina se utiliza para almacenar en variables los parámetros del protocolo de comunicación que solicita la ejecución de un determinado comando.

```
OBTENER_VALORES LD A, (TABLA_DE_ALMACENAMIENTO+1)
                LD (CANTIDAD_DATOS), A
                LD IX, (TABLA_DE_ALMACENAMIENTO+2)
```

```
LD (POSICION_MEMORIA_INICIAL), IX
LD IX, (TABLA_DE_ALMACENAMIENTO+4)
LD (POSICION_MEMORIA_FINAL), IX
LD IX, (TABLA_DE_ALMACENAMIENTO+6)
LD (POSICION_MEMORIA_NUEVA_POSICION), IX
LD IX, TABLA_DE_ALMACENAMIENTO+8
LD (POSICION_MEMORIA_UBICACION_DATOS), IX
RET
```

- **Enviar Byte**: esta rutina se utiliza para enviar datos por el puerto serie del sistema.

```
ENVIAR_BYTE             OUT (PUERTO_SIO_DB), A  ;enviar byte
LOOP_5                  IN A, (PUERTO_SIO_CB)   ;fijarse en RR0 si lo envio
                        BIT 2, A
                        JP Z, LOOP_5            ;si no lo envio, esperar
                        RET                     ;retornar
```

Se puede ver que esta rutina no retorna hasta que se haya enviado el byte por el puerto serie. Esto se realiza para evitar pérdida de información por sobreescritura. Recordar que el puerto serie trabaja a una velocidad de 9600 baudios y que el microprocesador trabaja con una frecuencia de reloj de 4 Mhz.

3.1.2.4.3.DATOS

Este módulo esta compuesto por las declaraciones de las variables y constantes utilizadas por los módulos que componen el archivo Z80-DSK.HEX

El listado completo de este módulo se puede ver al final de este documento.

3.1.2.4.4.ERRORES

Este módulo esta compuesto por la rutina que atiende los errores que se pueden producir en la comunicación con la PC.

Durante la comunicación con la PC en la dirección PC-placa puede ocurrir que los datos recibidos tengan errores. Esta rutina genera una pérdida de tiempo durante la cual se espera que terminen de llegar los posibles datos faltantes y avisa a la PC de la ocurrencia del error.

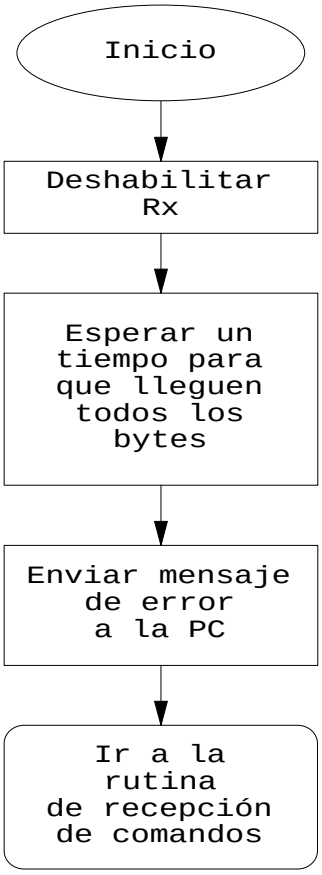


Diagrama 11: Errores

```
ERROR_DE_Tx     CALL DESHABILITACION_DE_Rx ;no se reciben mas caracteres
CONTINUAR       LD DE, CERO                ;generar un tiempo ocioso para
LOOP_2          LD HL, 0FFFF               ;que terminen de arribar los
                INC DE                     ;bytes al puerto serie antes de
                AND A                      ;habilitar el sistema nuevamente
                SBC HL, DE
                JP NZ, LOOP_2
                LD A, CMD_ERROR_EN_TX      ;informar a la PC de que se produ-
                CALL ENVIAR_BYTE           ;jo un error
                CALL HABILITACION_DE_RX    ;habilitar nuevamente la recepcion
                JP RECEPCION_COMANDOS      ;ir a recibir nuevos comandos
```

3.1.2.4.5.MJO-INT

Este módulo contiene la rutina a la que accede el microprocesador del sistema cuando ocurre una interrupción en el SIO por la llegada de un byte.

A la llegada de un carácter a la placa de desarrollo se ejecuta esta rutina, la cual inicializa un puntero de la memoria que indica la posición inicial de almacenamiento de los bytes que llegarán por el puerto serie. El primer byte que llega indica la cantidad de bytes que vienen detrás de el, cuyo valor se utiliza para recibir los demás bytes. Si la cantidad indicada por este byte es menor que la cantidad de bytes que arriban al puerto serie, se sale de esta rutina cuando se alcance esta cantidad; pero, si la cantidad arribada es menor que la cantidad indicada por el primer byte, la rutina queda esperando la llegada de la totalidad de los bytes indicados.

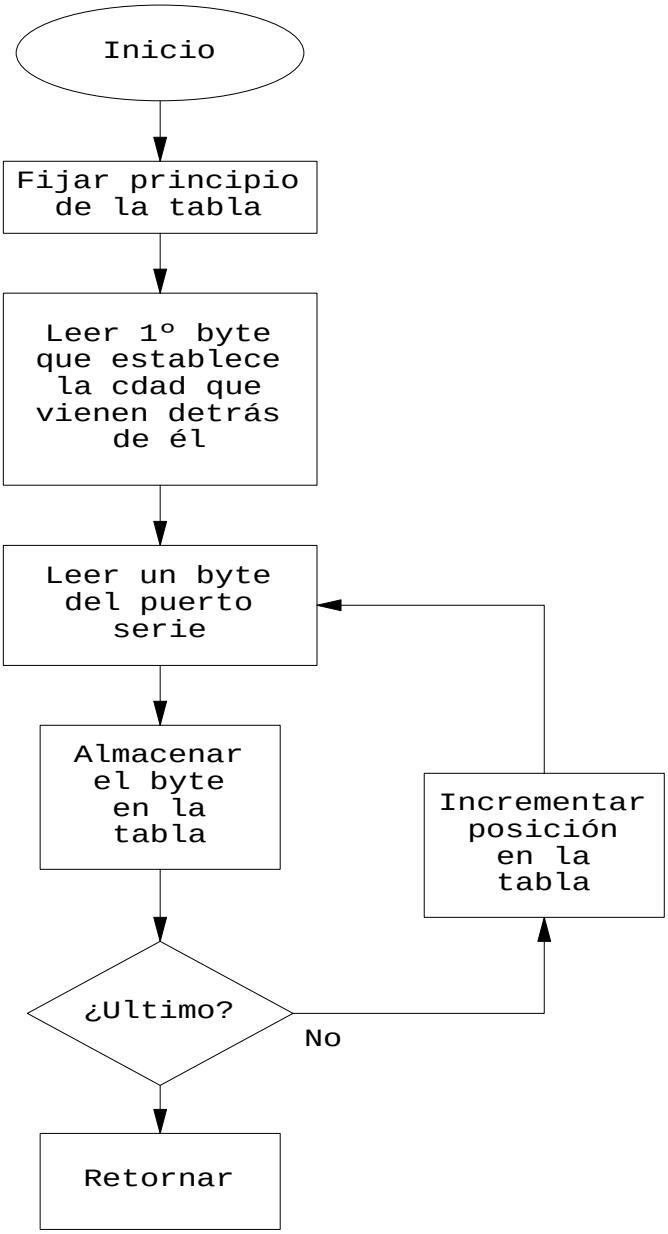


Diagrama 12: Manejo de las interrupciones

```
BUFFER_TX_VACIO         RETI
CAMBIO_ESTADO_EXTERNO   RETI
CONDICION_RX_ESPECIAL   RETI
CARACTER_DISPONIBLE     LD IX, TABLA_DE_ALMACENAMIENTO
                        IN A, (PUERTO_SIO_DB)   ;ingresamos el primer byte
                                                ;recibido, el que genero la
                                                ;interrupcion
                        LD B, A                 ;cantidad de bytes que debemos
                                                ;recibir
                        LD (CANTIDAD_BYTES), A  ;almacenamos la cantidad de
                                                ;bytes que debemos recibir
LOOP_1                  CALL INGRESAR_BYTE      ;ingresamos un byte
                        DJNZ LOOP_1             ;recibir otro byte
FIN                     RETI                    ;volver
```

3.1.2.4.6.PROG-SIO

Este módulo contiene las distintas rutinas de programación que se le efectúa al SIO del sistema.

```
INICIALIZACION_SIO      CALL RESETEAR_SIO
                        CALL FIJAR_VECTOR_DE_INTERRUPCION_SIO
                        CALL FIJAR_CONDICIONES_DE_OPERACION_SIO
                        CALL HABILITACION_DE_INTERRRUPCION_CON_1_CARACTER
                        CALL HABILITACION_DE_Rx
                        CALL HABILITACION_DE_Tx
                        RET

RESETEAR_SIO            LD A, CHANNEL_RESET_SIO
                        OUT (PUERTO_SIO_CB), A
                        RET

FIJAR_VECTOR_DE_INTERRUPCION_SIO        LD A, POINTER_2_SIO
                                        OUT (PUERTO_SIO_CB), A
                                        LD A, VECTOR_INTERRUPCION_SIO
                                        OUT (PUERTO_SIO_CB), A
                                        RET

FIJAR_CONDICIONES_DE_OPERACION_SIO      LD A, POINTER_4_SIO
                                        OUT (PUERTO_SIO_CB), A
                                        LD A, CONTROL_4_SIO
                                        OUT (PUERTO_SIO_CB), A
                                        RET

HABILITACION_DE_Tx      LD A, POINTER_5_SIO
                        OUT (PUERTO_SIO_CB), A
                        LD A, CONTROL_1_5_SIO
                        OUT (PUERTO_SIO_CB), A
                        RET

HABILITACION_DE_Rx      LD A, POINTER_3_SIO
                        OUT (PUERTO_SIO_CB), A
                        LD A, CONTROL_1_3_SIO
                        OUT (PUERTO_SIO_CB), A
                        RET

DESHABILITACION_DE_Tx     LD A, POINTER_5_SIO
                          OUT (PUERTO_SIO_CB), A
                          LD A, CONTROL_2_5_SIO
                          OUT (PUERTO_SIO_CB), A
                          RET

DESHABILITACION_DE_Rx     LD A, POINTER_3_SIO
                          OUT (PUERTO_SIO_CB), A
                          LD A, CONTROL_2_3_SIO
                          OUT (PUERTO_SIO_CB), A
                          RET

RESETEAR_ERRORES_SIO    LD A, CONTROL_1_0_SIO
                        OUT (PUERTO_SIO_CB), A
                        RET

HABILITACION_DE_INTERRRUPCION_CON_1_CARACTER    LD A, POINTER_1_SIO
                                                OUT (PUERTO_SIO_CB), A
                                                LD A, CONTROL_4_1_SIO
                                                OUT (PUERTO_SIO_CB), A
                                                RET
```

3.1.2.4.7.RX-CMD

Este módulo contiene la rutina que permite la recepción de los comandos y la bifurcación hacia la rutina que ejecuta el comando solicitado.

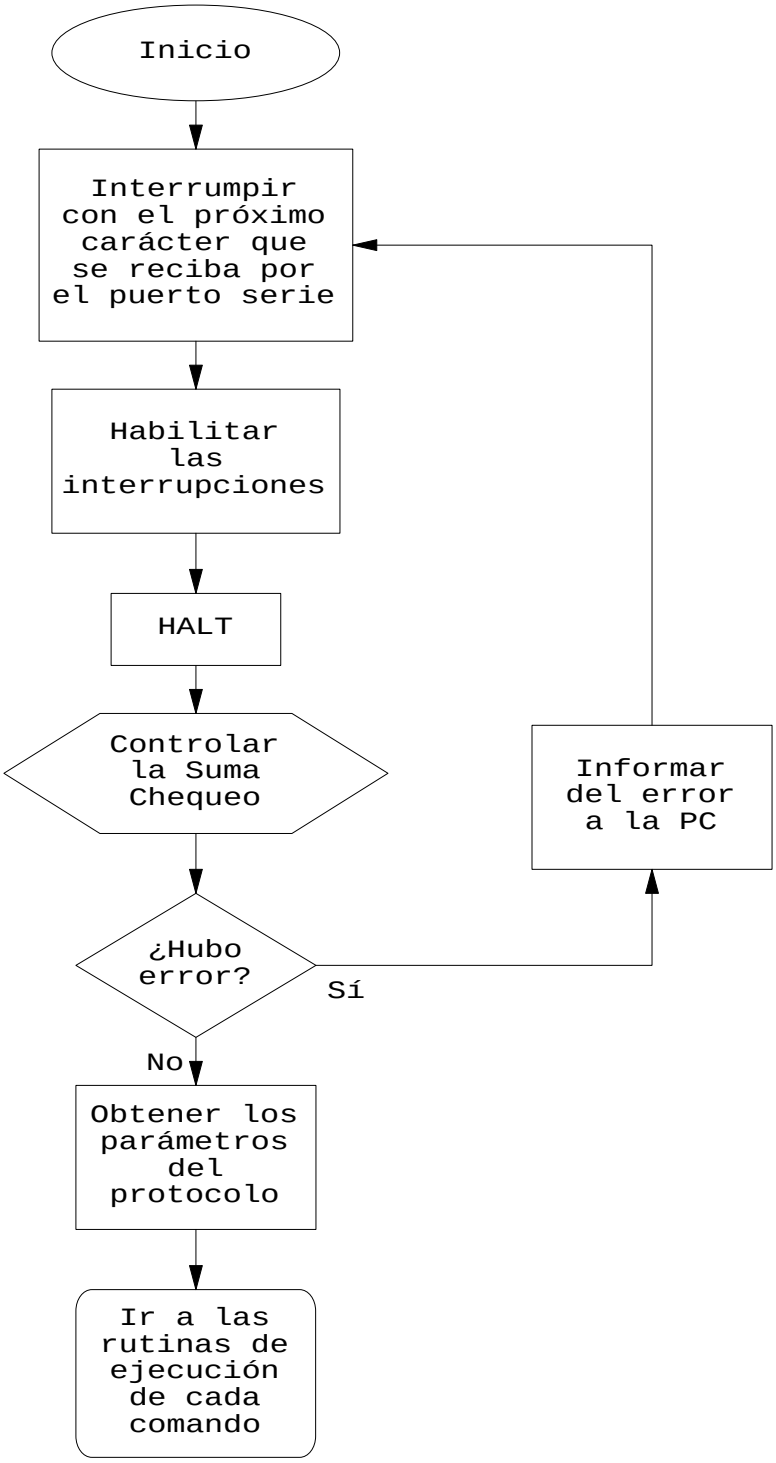


Diagrama 13: Recepción de los comandos

```
RECEPCION_COMANDOS      LD A, CONTROL_3_0_SIO           ;interrumpir en el
                        OUT (PUERTO_SIO_CB), A          ;siguiente caracter
                                                        ;que se reciba
                        EI                              ;habilitacion de las
                                                        ;interrupciones
                         ;interrumpir con el primer que se reciba
                        HALT                            ;esperamos la llegada
                                                        ;del primer caracter
                                                        ;desde la placa
                        LD C, CERO
                        CALL SUMA_CHEQUEO_ENTRADA       ;realizar la suma che-
```

```
                                        ;queo de los bytes
                                        ;de la tabla
                LD A, C
                CP UNO                  ;hubo un error de
                                        ;suma chequeo?
                JP Z, ERROR_DE_Tx
                CALL OBTENER_VALORES
                LD IX, TABLA_DE_ALMACENAMIENTO  ;principio de la tabla
                LD A, (IX)              ;primer byte (comando)
;comparar el primer byte de la tabla de almacenamiento con los
;valores de los posibles comandos. Ir a la rutina que corresponda
                CP CMD_LLENAR_BLOQUE
                JP Z, LLENAR_BLOQUE
                CP CMD_ENVIAR_BLOQUE
                JP Z, ENVIAR_BLOQUE
                CP CMD_INICIALIZAR_REGISTROS
                JP Z, INICIALIZAR_REGISTROS
                CP CMD_EXAMINAR_REGISTROS
                JP Z, EXAMINAR_REGISTROS
                CP CMD_LLENAR_PUERTOS
                JP Z, LLENAR_PUERTOS
                CP CMD_ENVIAR_PUERTOS
                JP Z, ENVIAR_PUERTOS
                CP CMD_EJECUTAR
                JP Z, EJECUTAR
                CP CMD_INSERTAR_BIFURCACION
                JP Z, INSERTAR_BIFURCACION
```

Se puede observar que el sistema normalmente se encuentra en la instrucción de HALT esperando ser interrumpido por un carácter proveniente del puerto serie. Una vez que este carácter interrumpe se ingresan los demás caracteres como se vio en el módulo MJO-INT y posteriormente se analiza la existencia de errores. Si no se produjeron errores, el programa bifurca a la rutina que ejecuta el comando solicitado.

También esta la rutina que efectúa la Suma Chequeo de los datos arribados por el puerto serie.

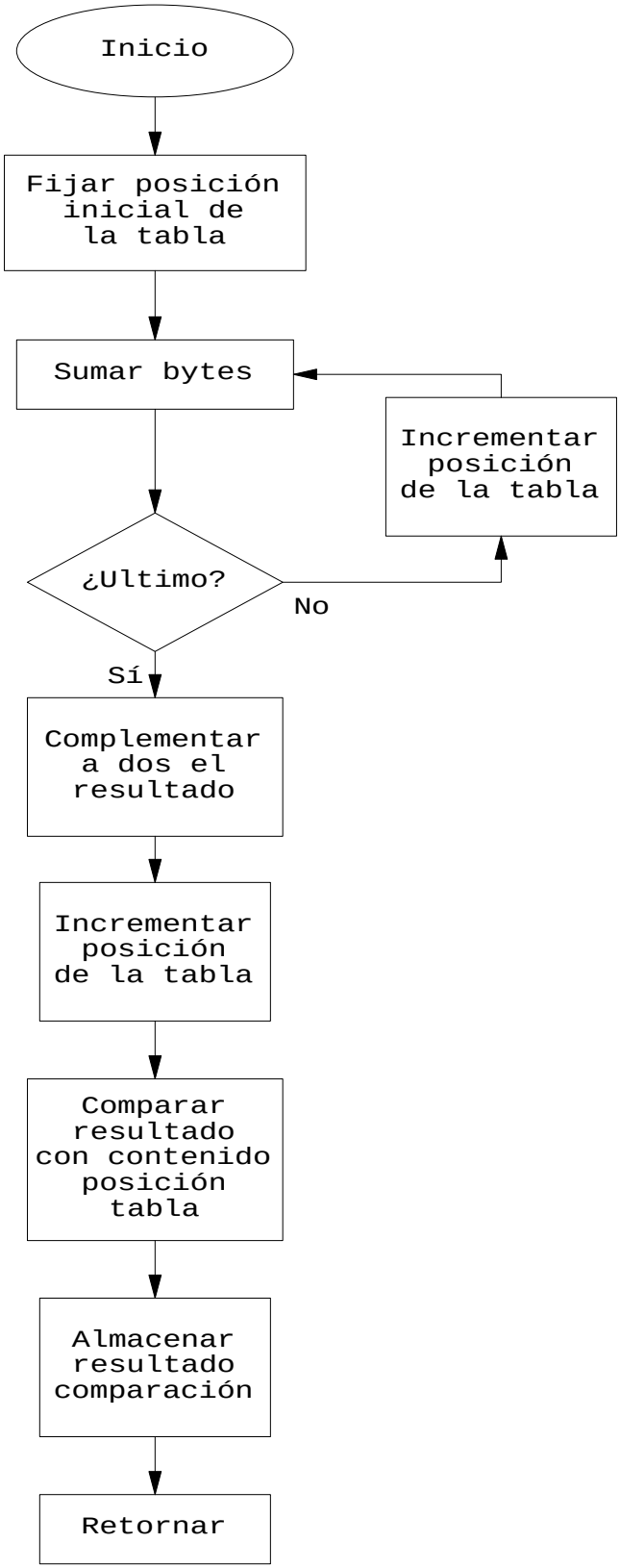


Diagrama 14: Suma chequeo

```
SUMA_CHEQUEO_ENTRADA    LD IX, TABLA_DE_ALMACENAMIENTO ;principio de la tabla
                                                       ;donde se encuentran
                                                       ;los bytes recibidos
                        LD A, (CANTIDAD_BYTES)         ;cantidad de bytes a
```

```
                                        ;leer
                    SUB 2               ;restarle 2 por la
                                        ;forma de realizar la
                                        ;suma chequeo
                    LD B, A
                    LD A, (IX)          ;contenido de la tabla
LOOP_3              INC IX              ;posicion siguiente en
                                        ;la tabla
                    ADD A, (IX)         ;sumar los bytes
                    DJNZ LOOP_3         ;repetir hasta CANTI-
                                        ;DAD_BYTES - 1
                    CPL                 ;complemento a uno
                    ADD A, UNO          ;complemento a dos
                    INC IX              ;siguiente byte
                    CP (IX)             ;comparar resultado
                                        ;con suma chequeo reci-
                                        ;bida desde la PC
                    JP Z, FINALIZAR     ;si coincide retornar
                    LD C, 1             ;marcar el error
FINALIZAR           RET                 ;retornar
```

La suma chequeo se realiza sumando todos los bytes arribados excepto el primero y el último (el primero porque solamente indica la cantidad de bytes que vienen detrás de él, y el último porque trae la suma chequeo realizada por la PC) para luego realizar el complemento a dos del resultado y compararlo con el valor suministrado por la PC, cuyo resultado se devuelve por medio del registro C de la CPU.

3.1.2.4.8.RETCONTR

Este módulo contiene la rutina a la que accede el software de placa por medio de la inserción de una bifurcación controlada para permitir almacenar el contenido de los registros de la CPU en las variables registros antes de devolver el control de la placa al software de PC.

```
RETORNO_CONTROLADO  PUSH AF
                    ;guardar el valor del acumulador y de la banderas
                    LD A, B
                    LD (REGISTRO_B), A
                    LD A, C
                    LD (REGISTRO_C), A
                    POP BC
                     ;recuperar el valor de AF almacenado en la pila,
                     ;en el registro BC
                    LD A, B
                    LD (REGISTRO_A), A
                    LD A, C
                    LD (REGISTRO_F), A
                    LD A, D
                    LD (REGISTRO_D), A
                    LD A, E
                    LD (REGISTRO_E), A
                    LD A, H
                    LD (REGISTRO_H), A
                    LD A, L
                    LD (REGISTRO_L), A
                    LD (REGISTRO_IX), IX
                    LD (REGISTRO_IY), IY
                    POP BC
                     ;recuperar el valor del PC
                     ;decrementar en 3 el valor del mismo porque el
                     ;valor que tiene corresponde a la posicion de
                     ;memoria siguiente a la instruccion de bifurcacion
                     ;controlada
                    DEC BC
                    DEC BC
                    DEC BC
                     ;la instruccion CALL NN ocupa tres bytes
                    LD (REGISTRO_PC), BC
                    LD (REGISTRO_SP), SP
                    LD A, I
                    LD (REGISTRO_I), A
                    LD A, R
                    LD (REGISTRO_R), A
                    ;devolver el valor que tenia originalmente el SP
                    LD IX, (VALOR_PREVIO_SP)
                    LD SP, IX
                    LD A, (VALOR_PREVIO_I)
```

```
LD I, A
LD A, (VALOR_PREVIO_R)
LD R, A
LD A, CMD_EJECUCION_OK
CALL ENVIAR_BYTE
JP RECEPCION_COMANDOS
```

3.2. Parte compuesta por la PC

3.2.1. Introducción

El software de PC del Z80 Development System Kit es un programa que permite, como su nombre lo indica, administrar todos los recursos de la placa de desarrollo del sistema. Este software está diseñado en el lenguaje de programación Visual Basic version 3.0 con el objeto de ofrecer una interface entre la placa de desarrollo y el usuario en entorno Windows.

3.2.2. Características:

El software de PC del Z80 Development System Kit está diseñado para realizar la interface Usuario - Placa y su implementación permite al usuario disponer de las funciones necesarias para lograr un completo manejo de la placa de desarrollo. Las operaciones que pueden realizarse con la placa de desarrollo a través de este software son las siguientes:

- Ingresar un archivo con formato hexadecimal (*.HEX) a la memoria.
- Guardar un bloque de memoria en un archivo con formato hexadecimal.
- Imprimir el contenido de un bloque de memoria, de los puertos de la placa y de los registros de la CPU.
- Ver el contenido de un bloque de memoria.
- Llenar un bloque de memoria.
- Comparar los contenidos de dos bloques de memoria.
- Trasladar el contenido de un bloque de memoria hacia otro sector de la misma.
- Ver y programar el contenido de los registros de la CPU.
- Ejecutar un programa situado en memoria en tiempo real, tiempo virtual, paso a paso o con break point.
- Ensamblar instrucciones en memoria.
- Ver el contenido de los puertos de la placa.
- Programar los puertos de la placa.
- Trabajar con dos extensiones del mapa de memoria y del mapa de puertos.

Además de estas operaciones, se pretende que el software de PC sea lo mas compatible posible con las características de otras aplicaciones Windows. Para ello se agrega, como operaciones a realizar, las siguientes:

- Permitir la edición de texto, o sea que permita las operaciones de Cortar, Pegar, Copiar y Borrar texto.
- Permitir la organización de las ventanas.
- Dar información del sistema.

  Estas operaciones constituyen la base del desarrollo del programa.

3.2.3. Objetivos:

El objetivo del software de PC es el de administrar los recursos de la placa de desarrollo a través del puerto serie RS232C utilizando la máxima potencialidad de este lenguaje de programación, tratando de simplificar al mínimo el software de placa que se encuentra en la placa de desarrollo y que se escribe en lenguaje ensamblador.

3.2.4. Composición:

El software de PC (llamado proyecto a partir de ahora) está diseñado de forma modular, donde los módulos que lo componen son los siguientes:

- RUTGRALS.BAS: módulo de código con los procedimientos y funciones generales del proyecto.
- CONSTANT.TXT: módulo de texto donde se definen constantes de uso general.
- DESENSAM.BAS: módulo de código con los procedimiento y funciones necesarias para permitir el desensamblado de un bloque de datos.
- COMPLACA.BAS: módulo de código con los procedimientos y funciones necesarias para lograr la comunicación con la placa.
- UTILES.BAS: módulo de código con los procedimientos y funciones útiles para todo el proyecto.
- EJECUTAR.BAS: módulo de código con los procedimientos y funciones necesarias para llevar a cabo los distintos tipos de ejecución de un programa.
- Z80.FRM: formulario padre. Ventana principal del proyecto.
- FMM_PS.FRM: formulario utilizado para la visualización del programa del sistema que se encuentra en la memoria.
- FA_GUARD.FRM: formulario que gestiona el guardado de archivos.
- FMM_PU.FRM: formulario utilizado para la visualización del programa del usuario que se encuentra en la memoria.
- FMM_D.FRM: formulario utilizado para la definición de los comienzos de los bloques de memoria a visualizar.
- FMM_DA.FRM: formulario utilizado para la visualización de los datos que se encuentran en la memoria.
- FS_E.FRM: formulario que gestiona el ensamblado línea a línea de instrucciones en la memoria.
- FO_PS.FRM: formulario utilizado para la definición del puerto serie a utilizar por el usuario.
- FS_REGIS.FRM: formulario encargado de la gestión de los registros.
- FA_IMPR.FRM: formulario encargado de la gestión de impresión.
- F_GENERA.FRM: formulario que contiene la herramienta para la gestión de archivos.
- FP_VER.FRM: formulario utilizado para la visualización del contenido de los puertos.
- FP_ESCR.FRM: formulario encargado de la gestión de programación de los puertos.
- FM_LCT.FRM: formulario encargado de la gestión de llenar, comparar y trasladar bloques de memoria.
- FS_D.FRM: formulario utilizado para la definición de los parámetros de ejecución.
- ACERCADE.FRM: formulario utilizado para la visualización de la ventana Acerca De... del menú Ayuda como en toda aplicación Windows.

Estos módulos se encuentran linkeados en el archivo Z80-DSK.MAK, el cual se utiliza para dar origen al archivo Z80-DSK.EXE que es el que finalmente se distribuye.

NOTA: MAK es la extensión usada para los proyectos.
FRM es la extensión usada para los formularios.
BAS es la extensión usada para los módulos de código.
TXT es la extensión usada para los módulos de texto.

3.2.5. Diseño

3.2.5.1.Introducción

Creemos que la forma más conveniente de explicar el diseño del software de PC es mediante la estructuración de la explicación. Se intentara, como en el caso del software de placa, de ser lo más claros posibles, por lo que la explicación se hara de manera funcional y siguiendo la cronología del desarrollo.

3.2.5.2.Lineamientos

Para el diseño del software de PC se siguieron varios lineamientos, que son la estructura de toda la programación, a saber:

- permitir que el usuario decida libremente el puerto serie a utilizar.
- indicar todos los errores que sucedan y las causas de los mismos.
- ofrecer un entorno lo mas “Windows” posible, ofreciéndole al usuario una pequeña ayuda sobre lo que haga cada elemento que vea en pantalla.
- intensificar el uso de procedimientos y funciones durante la programación para permitir un mejor seguimiento del programa.

3.2.5.3.Secuencia

Lo primero que se desarrolla al diseñar una aplicación en Visual Basic son las ventanas que tendrá el proyecto, posteriormente se escribe el código para cada procedimiento de suceso de cada objeto que tenga el formulario en su interior. Una vez que se tiene todo lo anterior, se escribe el código que permite un desenvolvimiento dinámico y coordenado entre distintas funciones del software.

3.2.5.4.Desarrollo

3.2.5.4.1.Ventanas

Las ventanas son la interface entre el programa y el usuario. Permite que el usuario ingrese parámetros para las distintas funciones y logra que el usuario visualice información que necesita.

Las ventanas son llamadas “formularios” en el lenguaje de programación Visual Basic. Estos formularios pueden contener “objetos” en su interior cuya función es la de permitir la interacción con el usuario. Cualquier aplicación Windows contiene objetos dentro de las ventanas, por ejemplo se pueden ver botones de órdenes (famoso botón para Aceptar o Cancelar una operación dentro de una ventana), cajas de texto (donde se definen la cantidad de hojas a imprimir), etiquetas (leyenda que indica que en esa caja de texto se debe indicar la cantidad de hojas a imprimir), etc.

Estos objetos son suceptibles a percibir que procedimiento de suceso esta ocurriendo en él. En base a ello se colocan los distintos objetos que se desea que tenga el formulario ya que el programador tiene el control sobre la forma en que desea que el usuario acceda a las distintas funciones del software.

Como ejemplo de procedimientos de suceso podemos citar los siguientes:

- hacer click con el botón del mouse.
- hacer doble click con el botón del mouse.
- mostrar una ventana.
- cambiar el tamaño de una ventana.
- minimizar, maximizar o cerrar una ventana.
- escribir texto.
- etcétera.

A continuación se hará una descripción de todos los formularios que posee el software de PC.

3.2.5.4.1.1.Formulario principal (padre)

El diseño del software comenzó con la creación de la ventana principal del sistema. En ella se colocaron, a través del uso de la barra de menús, los diversos comandos que constituyen y permiten administrar todo el sistema. La ventana que se diseñó es la siguiente:

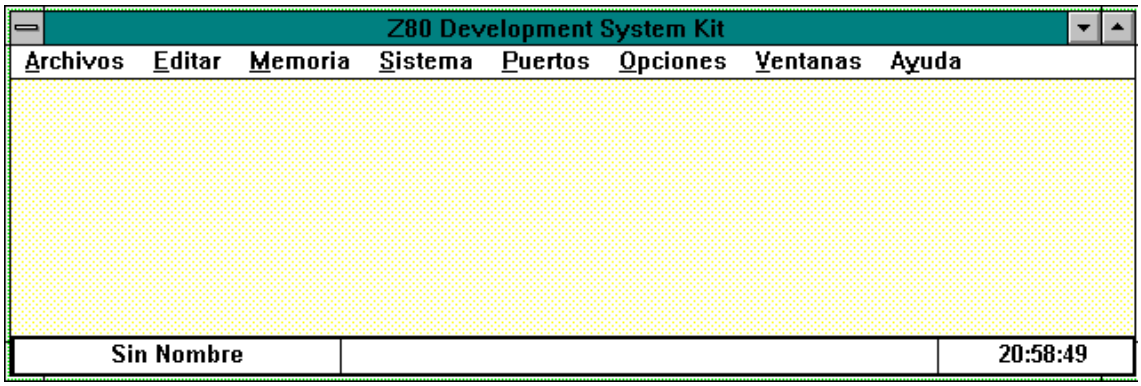


Figura 1: Formulario padre

cuyas propiedades son:

```
Begin MDIForm Z80
  Caption        =  "Z80 Development System Kit"   'leyenda del formulario
  ClientHeight   =  2640                           'altura del interior del formulario
  ClientLeft     =  60                             'posición izquierda del interior
  ClientTop      =  1485                           'posición superior del interior
  ClientWidth    =  9480                           'ancho del interior
  Height         =  3330                           'altura del formulario
  Icon           =  Z80.FRX:0000                   'ícono de minimización
  Left           =  0                              'posicion izquierda del formulario
  LinkTopic      =  "MDIForm1"
  Tag            =  "Z80DSK"                       'identificación extra
  Top            =  855                            'posición superior
  Width          =  9600                           'ancho
  WindowState    =  2  'Maximized                  'estado inicial del formulario
End
```

En ella se puede ver la existencia de ocho menús. El diseño de esta barra de menús correspondió al agrupamiento funcional de los distintos comandos del sistema. Así, tenemos que dentro del menú Archivos se encuentran los comandos:

Figura 2: Submenú Archivos

cuya representación a través del código es la siguiente:

```
Begin Menu menuArchivos
   Caption        =  "&Archivos"
   Begin Menu ArchivosNuevo
     Caption      =  "&Nuevo"
   End
   Begin Menu ArchivosIngresar
     Caption      =  "Abr&ir"
   End
```

```
  Begin Menu ArchivosGuardar
    Caption        =   "&Guardar"
  End
  Begin Menu ArchivosGuardarComo
    Caption        =   "Guardar &Como"
  End
  Begin Menu ArchivosLinea0
    Caption        =   "-"
  End
  Begin Menu ArchivosImprimir
    Caption        =   "Im&primir"
  End
  Begin Menu ArchivosLinea1
    Caption        =   "-"
  End
  Begin Menu ArchivosSalir
    Caption        =   "&Salir"
  End
End
```

en el menú Editar están los comandos:

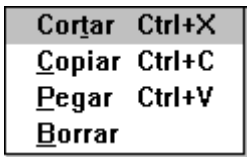


Figura 3: Submenú Editar

cuya representación a través del código es la siguiente:

```
Begin Menu menuEditar
    Caption        =   "&Editar"
    Begin Menu EditarItem
      Caption        =   "Cor&tar"
      Index          =   0
      Shortcut       =   ^X
    End
    Begin Menu EditarItem
      Caption        =   "&Copiar"
      Index          =   1
      Shortcut       =   ^C
    End
    Begin Menu EditarItem
      Caption        =   "&Pegar"
      Index          =   2
      Shortcut       =   ^V
    End
    Begin Menu EditarItem
      Caption        =   "&Borrar"
      Index          =   3
    End
  End
```

en el menú Memoria están los comandos:

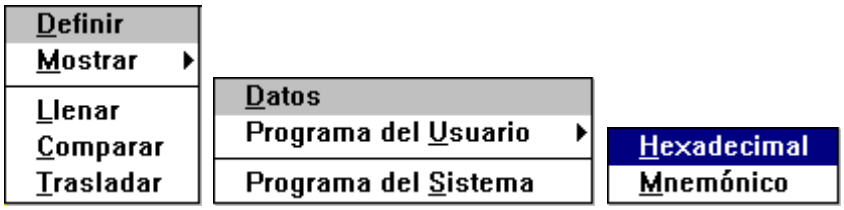


Figura 4: Submenú Memoria

en el menú Sistema están los comandos:

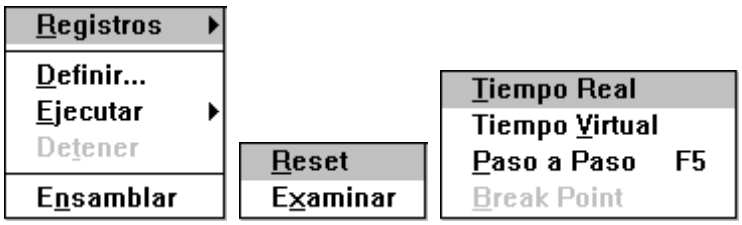


Figura 5: Submenú Sistema

en el menú Puertos están los comandos:

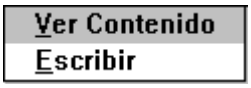


Figura 6: Submenú Puertos

en el menú Opciones están los comandos:

Figura 7: Submenú Opciones

en el menú Ventanas están los comandos:

Figura 8: Submenú Ventanas

cuya representación a través del código es la siguiente:

```
Begin Menu menuVentanas
    Caption       =  "&Ventanas"
    WindowList    =  -1  'True          'se hace un listado de las ventanas que estén abiertas
                                        'al final del submenú anterior
End
```

y en el menu Ayuda están los comandos:

Figura 9: Submenú Ayuda

NOTA: el código completo para la representación de los distintos comandos se puede encontrar al final de este documento.

En el submenú de cada menú de este formulario principal se encuentran los distintos comandos que posee el software para la administración de la placa. Cuando un item del submenú tiene a su derecha una flecha indica que si se hace click en el se despliega un nuevo submenú con más opciones.

En los distintos menús se pueden ver que una letra se encuentra subrayada. Esta letra en conjunto con la tecla ALT dan la combinación para el acceso rápido al submenú. Por ejemplo si se utiliza la combinación ALT + A se accederá al submenú Archivos. También se puede ver que algunas letras de los comandos que se encuentran en los distintos submenús están subrayadas, su función es similar a la anteriormente descripta, con la salvedad de que no se utiliza la tecla ALT.

El formulario principal del proyecto es el que alberga en su interior a todas las ventanas que se abran; permite organizarlas, distribuirlas, maximizarlas y minimizarlas pero siempre dentro de sus límites. Este formulario principal contiene, en su parte inferior, tres Etiquetas cuya función es la de mostrarle al usuario la hora, el nombre y ruta del archivo que esté abierto y una ayuda en línea sobre lo que realiza cada objeto dentro de un formulario. Las propiedades de estas etiquetas son las siguientes:

```
Begin Label Aclaraciones
   Alignment      =  2  'Center          'alineación del texto en el interior.
   BackStyle      =  0  'Transparent     'estilo del fondo.
   BorderStyle    =  1  'Fixed Single    'estilo del borde.
   FontBold       =  0   'False          'texto en negrita.
   FontItalic     =  0   'False          'texto en cursiva.
   FontName       =  "Arial"             'tipo de letra.
   FontSize       =  8.25                ´tamaño de letra.
   FontStrikethru =  0   'False          'texto tachado.
   FontUnderline  =  0   'False          'texto subrayado.
   Height         =  375
   Left           =  2760
   TabIndex       =  3
   Top            =  0
   Width          =  5055
End
Begin Label ArchivoEnUso
   Alignment      =  2  'Center
   BackStyle      =  0  'Transparent
   BorderStyle    =  1  'Fixed Single
   FontBold       =  -1  'True
   FontItalic     =  0   'False
   FontName       =  "Arial"
   FontSize       =  9.75
   FontStrikethru =  0   'False
   FontUnderline  =  0   'False
   Height         =  375
   Left           =  0
   TabIndex       =  2
   Top            =  0
   Width          =  2775
End
Begin Label EtiquetaHora
```

```
    Alignment      =   2  'Center
    BackStyle      =   0  'Transparent
    BorderStyle    =   1  'Fixed Single
    FontBold       =   -1  'True
    FontItalic     =   0   'False
    FontName       =   "Arial"
    FontSize       =   9.75
    FontStrikethru =   0   'False
    FontUnderline  =   0   'False
    Height         =   315
    Left           =   7800
    TabIndex       =   1
    Top            =   0
    Width          =   1815
  End
```

Además de todos estos objetos que posee el formulario, se encuentran otros tres que no se ven en el gráfico del formulario; uno es un objeto que administra la comunicación serie de una aplicación Visual Basic, otro es un temporizador utilizado para actualizar el valor de la hora y el tercero es una caja de imágenes que se utiliza para albergar las etiquetas. Esto se hace porque de esta forma no se cubren las etiquetas con las ventanas que se abran dentro del formulario principal.

```
Begin MSComm PuertoSerie
    Interval       =   1000
    Left           =   2160
    RThreshold     =   1                ‘ejecutar el procedimiento de suceso con cada
                                        ‘caracter
                                        ‘que llegue
    RTSEnable      =   -1  'True
    Settings       =   "9600,e,8,2"     ‘configuracion del puerto serie
    Top            =   0
End
Begin Timer Hora
    Interval       =   1000             ‘esta dado en microsegundos
    Left           =   9000
    Top            =   0
End
Begin PictureBox CajaDeAclaraciones
    Align          =   2  'Align Bottom ‘lugar de ubicacion
    AutoRedraw     =   -1  'True
    Enabled        =   0   'False
    Height         =   330
    Left           =   0
    ScaleHeight    =   300
    ScaleWidth     =   9450
    TabIndex       =   0
    Top            =   2310
    Width          =   9480
End
```

NOTA: Se habrá visto en los listados de las propiedades que muchas veces se utiliza el símbolo “&”, este es el símbolo que hay que poner antes de la letra que se quiere que aparezca subrayada.

3.2.5.4.1.2.Formulario Guardar

Este formulario aparece cuando se desea que el usuario defina los límites del bloque de memoria que desea que se almacene en un archivo con formato hexadecimal.

El formulario es el siguiente:

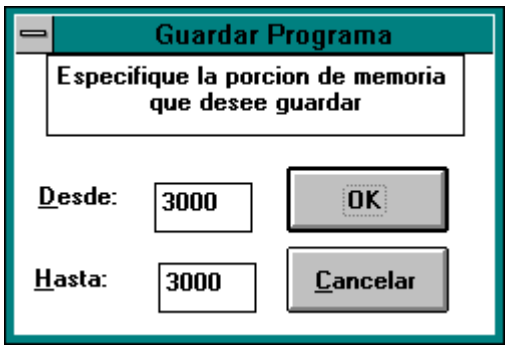


Figura 10: Formulario Guardar Como...

las propiedades de este formulario son:

```
Begin Form F_Archivos_Guardar
  BorderStyle    =  3  'Fixed Double        ‘borde doble y fijo
  Caption        =  "Guardar Programa"
  ClientHeight   =  2100
  ClientLeft     =  3375
  ClientTop      =  4905
  ClientWidth    =  3600
  Height         =  2505
  Left           =  3315
  LinkTopic      =  "Form1"
  ScaleHeight    =  2100
  ScaleWidth     =  3600
  Top            =  4560
  Width          =  3720
End
```

Este formulario se encuentra en el archivo FA_GUARD.FRM.

Dentro de este formulario se pueden ver dos etiquetas, tres cajas de texto y dos botones de órdenes cuyas propiedades son las siguientes:

```
Begin TextBox Descripcion
    Alignment      =  2  'Center              ‘ubicación del texto
    Height         =  615
    Left           =  240
    MultiLine      =  -1  'True               ‘ubicación del texto en múltiples líneas
    TabIndex       =  1                       ‘índice de tabulación.
    TabStop        =  0  'False
    Text           =  "Especifique la porcion de memoria que desee guardar"
    Top            =  0
    Width          =  3135
End
Begin Label Etiquetas
    Alignment      =  2  'Center
    BackStyle      =  0  'Transparent
    Caption        =  "&Desde:"
    Height         =  375
    Index          =  0
    Left           =  120
```

```
    TabIndex      =  2
    Top           =  960
    Width         =  735
End
Begin TextBox Direcciones
    Alignment     =  2  'Center
    Height        =  375
    Index         =  0
    Left          =  1050
    MaxLength     =  4                  ‘longitud máxima 4 caracteres.
    TabIndex      =  3
    Top           =  960
    Width         =  735
End
Begin Label Etiquetas
    Alignment     =  2  'Center
    BackStyle     =  0  'Transparent
    Caption       =  "&Hasta:"
    Height        =  375
    Index         =  1
    Left          =  135
    TabIndex      =  4
    Top           =  1575
    Width         =  615
End
Begin TextBox Direcciones
    Alignment     =  2  'Center
    Height        =  375
    Index         =  1
    Left          =  1080
    MaxLength     =  4
    TabIndex      =  5
    Top           =  1560
    Width         =  735
End
Begin CommandButton OK
    Caption       =  "OK"
    Default       =  -1  'True          ‘reacciona con la tecla ENTER
    Height        =  495
    Left          =  2040
    TabIndex      =  0
    Top           =  840
    Width         =  1215
End
Begin CommandButton Cancelar
    Cancel        =  -1  'True          ‘reacciona con la tecla ESC
    Caption       =  "&Cancelar"
    Height        =  495
    Left          =  2040
    TabIndex      =  6
    Top           =  1440
    Width         =  1215
End
```

De los listados anteriores se puede ver las propiedades mas interesantes, que son: TabIndex (índice en que irá circulando el foco por los distintos objetos del formulario con la tecla TAB), TabStop (permite habilitar o deshabilitar el objeto para recibir el foco a través de la tecla TAB), MaxLength (longitud máxima para los datos mostrados en una caja de texto), Default (permite definir qué botón de órdenes ejecutará el procedimiento de suceso click

cuando se pulse la tecla ENTER) y Cancel (permite definir que botón de órdenes ejecutará el procedimiento de suceso click cuando se pulse la tecla ESC).

3.2.5.4.1.3.Formulario Imprimir

Este formulario aparece cuando se desea que el usuario defina que es lo que pretende tener impreso en una hoja de papel. Puede elegir la impresión del contenido de un bloque de memoria, el contenido de los puertos del sistema, el contenido de los registros de la CPU o cualquier texto que se encuentre en el portapapeles.

El formulario es el siguiente:

Figura 11: Formulario Imprimir

cuyas propiedades son las siguientes:

```
Begin Form F_Archivos_Imprimir
  Caption        =   "Administrador de Impresion"
  ClientHeight   =   2625
  ClientLeft     =   1620
  ClientTop      =   4485
  ClientWidth    =   3585
  Height         =   3030
  Left           =   1560
  LinkTopic      =   "Form2"
  MaxButton      =   0   'False
  ScaleHeight    =   2625
  ScaleWidth     =   3585
  Top            =   4140
  Width          =   3705
End
```

Este formulario se encuentra en el archivo FA_IMPR.FRM

En este formulario se pueden ver dos botones de órdenes, cuatro botones de opciones independientes que indican cuáles son las opciones que tiene el usuario para imprimir, una caja contenedora de objetos que tiene en su interior tres botones de opciones que permiten ver las opciones con las que cuenta el usuario para imprimir el contenido de un bloque de memoria.

```
Begin OptionButton BotonQueImprimir
    Caption        =   "Memo&ria"
    Enabled        =   0   'False
    Height         =   375
    Index          =   0
    Left           =   120
```

```
    TabIndex      =  2
    Top           =  120
    Width         =  1215
End
Begin Frame TipoMemoria                    ‘caja contenedora de objetos
    Height        =  1335
    Left          =  1440
    TabIndex      =  9
    Top           =  120
    Width         =  2055
    Begin OptionButton BotonQueImprimirMemoria
      Caption       =  "Datos"
      Enabled       =  0   'False
      FontBold      =  0   'False
      FontItalic    =  0   'False
      FontName      =  "MS Sans Serif"
      FontSize      =  8.25
      FontStrikethru =  0   'False
      FontUnderline =  0   'False
      Height        =  255
      Index         =  0
      Left          =  150
      TabIndex      =  3
      Top           =  120
      Width         =  1095
    End
End
```

3.2.5.4.1.4.Formulario Definir Memoria

Este formulario aparece cuando el usuario desea indicar la posicion de memoria inicial a partir de la cual desea ver el contenido de la memoria.

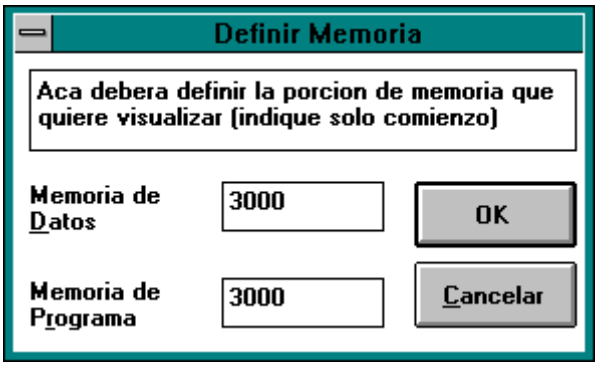


Figura 12: Formulario Definir Memoria

Se puede ver en el formulario que se puede definir el comienzo del bloque de memoria que muestra los datos y del bloque del programa del usuario. No se diseñó para poder definir el bloque de memoria donde se ubica el programa del sistema porque se consideró que la manera más correcta para seguirlo es desde el principio.

Las caracteristicas de este formulario son:

```
Begin Form F_Mem_Mostrar_Definir
  BorderStyle    =  3  'Fixed Double
  Caption        =  "Definir Memoria"
  ClientHeight   =  2235
```

```
  ClientLeft     =   1770
  ClientTop      =   3330
  ClientWidth    =   4275
  Height         =   2640
  Left           =   1710
  LinkTopic      =   "Form1"
  ScaleHeight    =   2235
  ScaleWidth     =   4275
  Top            =   2985
  Width          =   4395
End
```

Este formulario se encuentra en el archivo FMM_D.FRM

Dentro de este formulario se encuentran tres cajas de texto, dos etiquetas y dos botones de órdenes. Una caja de texto sirve para mostrar un mensaje, mientras que las otras dos se utilizan para que el usuario pueda ingresar los valores iniciales de los bloques de memoria que quiera ver.

NOTA: los nombres de los archivos que contienen los formularios se forman con la ruta para llegar a ellos mediante la barra de menus. Por ejemplo: FMM_D, la F es formulario, luego Memoria, posteriormente Mostrar y finalmente Definir.

3.2.5.4.1.5. Formulario Mostrar Datos

Este formulario se activa cuando el usuario desea visualizar el contenido de la memoria donde se albergan los datos del programa que se está ejecutando o escribiendo.

Es necesario ofrecerle al usuario la posibilidad de incrementar los límites del bloque de memoria mostrado, que lo decremente y que pueda actualizar el contenido mostrado.

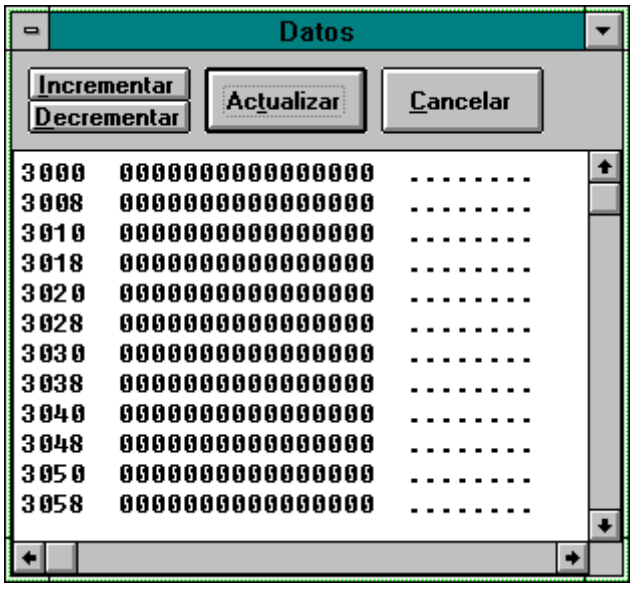


Figura 13: Formulario Datos

Las propiedades de este formulario son:

```
Begin Form F_Mem_Mostrar_Datos
  Caption        =   "Datos"
  ClientHeight   =   2715
  ClientLeft     =   1695
  ClientTop      =   3630
```

```
  ClientWidth    =  4080
  FontBold       =  0  'False
  FontItalic     =  0  'False
  FontName       =  "MS Sans Serif"
  FontSize       =  8.25
  FontStrikethru =  0  'False
  FontUnderline  =  0  'False
  Height         =  3120
  Left           =  1635
  LinkTopic      =  "Form2"
  MaxButton      =  0  'False
  MDIChild       =  -1 'True              ‘formulario hijo
  ScaleHeight    =  2715
  ScaleWidth     =  4080
  Tag            =  "MemoriaDatos"        ‘identificacion extra
  Top            =  3285
  Width          =  4200
End
```

Este formulario se encuentra en el archivo FMM_DA.FRM

Dentro de este formulario se encuentran cuatro botones de órdenes, una caja de texto y una caja de imágenes que alberga en su interior los botones de órdenes (cumple una funcion estetica solamente).

```
Begin PictureBox CajaDeComandos      ‘caja de imágenes.
   Align          =  1  'Align Top   ‘ubicación de la caja de imágenes.
   BackColor      =  &H00C0C0C0&     ‘color de fondo.
   BorderStyle    =  0  'None        ‘tipo de borde.
   Height         =  735
   Left           =  0
   ScaleHeight    =  735
   ScaleWidth     =  4080
   TabIndex       =  4
   TabStop        =  0  'False
   Top            =  0
   Width          =  4080
End
```

3.2.5.4.1.6.Formulario Mostrar Programa Usuario

Este formulario se activa cuando el usuario desea visualizar el contenido de la memoria donde se alberga el programa que se está ejecutando o escribiendo.

Es necesario ofrecerle al usuario la posibilidad de incrementar los límites del bloque de memoria mostrado, que lo decremente y que pueda actualizar el contenido mostrado.

Figura 14: Formulario Programa del Usuario

Las propiedades de este formulario son:

```
Begin Form F_Mem_Mostrar_PU
  Caption        =  "Programa del Usuario"
  ClientHeight   =  2730
  ClientLeft     =  1320
  ClientTop      =  2550
  ClientWidth    =  4230
  Height         =  3135
  Left           =  1260
  LinkTopic      =  "Form1"
  MaxButton      =  0   'False
  MDIChild       =  -1  'True
  ScaleHeight    =  11.375
  ScaleMode      =  4  'Character
  ScaleWidth     =  35.25
  Tag            =  "MemoriaPU"
  Top            =  2205
  Width          =  4350
End
```

Este formulario se encuentra en el archivo FMM_PU.FRM

Dentro de este formulario se encuentran cuatro botones de órdenes, una caja de texto y una caja de imágenes que alberga en su interior los botones de órdenes (cumple una funcion estetica solamente).

3.2.5.4.1.7.Formulario Mostrar Programa Sistema

Este formulario se activa cuando el usuario desea visualizar el contenido de la memoria donde se alberga el programa que se encuentra en la memoria EPROM de la placa de desarrollo y que contiene el software de placa del sistema.

Es necesario ofrecerle al usuario la posibilidad de incrementar los límites del bloque de memoria mostrado, que lo decremente y que pueda actualizar el contenido mostrado.

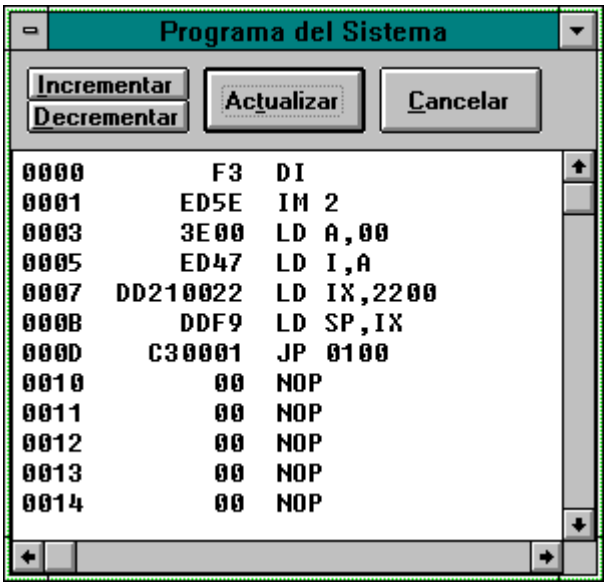


Figura 15: Formulario Programa del Sistema

Las propiedades de este formulario son:

```
Begin Form F_Mem_Mostrar_PS
  Caption        =  "Programa del Sistema"
  ClientHeight   =  2730
  ClientLeft     =  2070
  ClientTop      =  3900
  ClientWidth    =  4080
  Height         =  3135
  Left           =  2010
  LinkTopic      =  "Form3"
  MaxButton      =  0  'False
  MDIChild       =  -1  'True
  ScaleHeight    =  11.375
  ScaleMode      =  4  'Character
  ScaleWidth     =  34
  Tag            =  "MemoriaPS"
  Top            =  3555
  Width          =  4200
End
```

Este formulario se encuentra en el archivo FMM_PS.FRM

Dentro de este formulario se encuentran cuatro botones de órdenes, una caja de texto y una caja de imágenes que alberga en su interior los botones de órdenes (cumple una funcion estetica solamente).

3.2.5.4.1.8.Formulario para llenar, comparar y trasladar bloques de memoria

Este formulario se muestra cuando el usuario desee llenar, comparar o trasladar bloques de memoria. A través de este formulario el usuario puede definir los parámetros para realizar dichos comandos.

Debido a que los parámetros a definir son similares para los tres comandos, el formulario que se diseñó es el mismo para los tres; la diferencia se establece a través del cambio los títulos según corresponda al comando elegido. El formulario que se muestra a continuación vale para el comando Llenar, mientras que el que se encuentra a su derecha vale para los comandos Comparar y Trasladar.

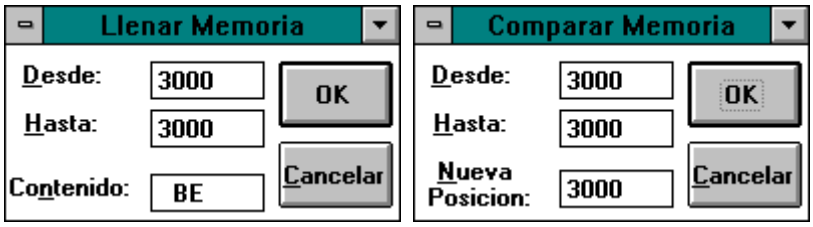


Figura 16: Formularios Llenar y Comparar

Las propiedades de estos formularios son:

```
Begin Form F_Mem_LCT
  BorderStyle    =  1  'Fixed Single
  ClientHeight   =  1305
  ClientLeft     =  1905
  ClientTop      =  4815
  ClientWidth    =  2940
  Height         =  1710
  Left           =  1845
  LinkTopic      =  "Form1"
  MaxButton      =  0   'False
  MDIChild       =  -1  'True
  ScaleHeight    =  1305
  ScaleWidth     =  2940
  Top            =  4470
  Width          =  3060
End
```

Nota: este formulario no tiene definida la leyenda CAPTION ya que se irá cambiando por código de acuerdo al comando solicitado.

Este formulario se encuentra en el archivo FM_LCT.FRM

Dentro de este formulario se encuentran cuatro etiquetas, cuatro cajas de texto y dos botones de órdenes. Las etiquetas y las cajas de texto se utilizan para que el usuario pueda definir los parametros que se le solicitan para la ejecucion “personalizada” del comando.

3.2.5.4.1.9.Formulario Registros

Este formulario se activa cuando el usuario desee visualizar y poder alterar el contenido de los registros de la CPU.

A través de este formulario el usuario goza de la posibilidad de programar el contenido de los registros de la CPU y también puede actualizar el contenido de los registros que se está viendo. El formulario es el siguiente:

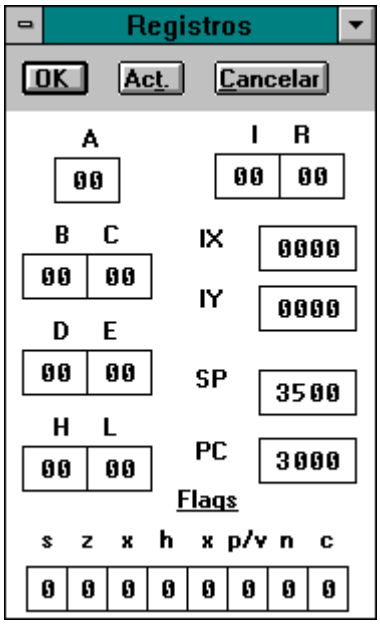


Figura 17: Formulario Registros

Este formulario fue diseñado con el propósito de mostrar los contenidos de los registros de una manera muy funcional. Se puede ver que los registros pares fueron agrupados para lograr una mejor interpretación de su funcionamiento; también se pretendió lograr una adecuada visualización de los valores y nombres que poseen las banderas utilizadas por la CPU para el control del flujo de las instrucciones.

Las propiedades de este formulario son:

```
Begin Form F_Sistema_Registros
  BorderStyle    =  1  'Fixed Single          'borde fijo y simple
  Caption        =  "Registros"
  ClientHeight   =  4290
  ClientLeft     =  1425
  ClientTop      =  1680
  ClientWidth    =  2760
  Height         =  4695
  Left           =  1365
  LinkTopic      =  "Form1"
  MaxButton      =  0  'False
  MDIChild       =  -1  'True
  ScaleHeight    =  4290
  ScaleWidth     =  2760
  Tag            =  "Registros"
  Top            =  1335
  Width          =  2880
End
```

Este formulario se encuentra en el archivo FS_REGIS.FRM

Dentro de este formulario se encuentran 22 etiquetas y 22 cajas de texto (21 son visibles y una es invisible), tambien hay tres botones de órdenes y una caja de imágenes para albergar los botones de órdenes (finalidad estetica).

Las propiedade de los distintos objetos que se encuentran dentro del formulario se darán de acuerdo a su función:

- Registros

```
Begin Label EtiquetaRegistros
```

```
    Alignment      =   2  'Center
    BackStyle      =   0  'Transparent
    Caption        =   "PC"
    Height         =   255
    Index          =   13
    Left           =   1350
    TabIndex       =   43
    Tag            =   "PC:"
    Top            =   2940
    Width          =   375
End
Begin TextBox Registros
    Alignment      =   2  'Center
    FontBold       =   0   'False
    FontItalic     =   0   'False
    FontName       =   "Fixedsys"
    FontSize       =   9
    FontStrikethru =   0   'False
    FontUnderline  =   0   'False
    Height         =   340
    Index          =   13
    Left           =   1890
    MaxLength      =   4
    MultiLine      =   -1  'True
    TabIndex       =   44
    Top            =   2940
    Width          =   735
End
```

- Banderas

```
Begin Label EtiquetasBitsFlags
    Alignment      =   2  'Center
    BackStyle      =   0  'Transparent
    Caption        =   "s"
    Height         =   315
    Index          =   7
    Left           =   150
    TabIndex       =   16
    Top            =   3600
    Width          =   315
End
Begin TextBox BitsFlags
    Alignment      =   2  'Center
    FontBold       =   0   'False
    FontItalic     =   0   'False
    FontName       =   "Fixedsys"
    FontSize       =   9
    FontStrikethru =   0   'False
    FontUnderline  =   0   'False
    Height         =   340
    Index          =   7
    Left           =   150
    MaxLength      =   1
    MultiLine      =   -1  'True
    TabIndex       =   17
    Top            =   3900
    Width          =   315
End
```

- Botones de órdenes

```
Begin PictureBox CajaDeComandos
    Align          =   1  'Align Top
    AutoRedraw     =   -1  'True
    BackColor      =   &H00C0C0C0&
    BorderStyle    =   0  'None
    Height         =   495
    Left           =   0
    ScaleHeight    =   495
    ScaleWidth     =   2760
    TabIndex       =   47
    TabStop        =   0   'False
    Top            =   0
    Width          =   2760
End
Begin CommandButton OK
    Caption        =   "OK"
    Default        =   -1  'True
    Height         =   255
    Left           =   120
    TabIndex       =   45
    Top            =   120
    Width          =   495
End
Begin CommandButton Actualizar
    Caption        =   "Ac&t."
    Height         =   255
    Left           =   840
    TabIndex       =   46
    Top            =   120
    Width          =   495
End
Begin CommandButton Cancelar
    Cancel         =   -1  'True
    Caption        =   "&Cancelar"
    Height         =   255
    Left           =   1560
    TabIndex       =   0
    Top            =   120
    Width          =   855
End
```

3.2.5.4.1.10.Formulario Definir Parametros Ejecucion

Este formulario se visualiza cuando el usuario desea definir los parámetros que utilizará el comando Ejecutar cuando lo haga a través del uso de un Break Point.

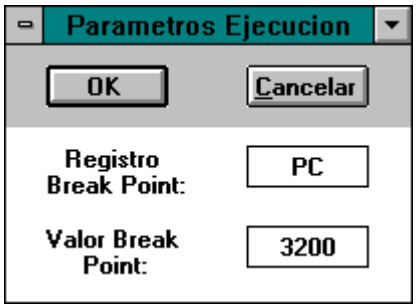


Figura 18: Formulario Definir Ejecución

Las propiedades de este formulario son:

```
Begin Form F_Sistema_Definir
  BorderStyle    =   1  'Fixed Single
  Caption        =   "Parametros Ejecucion"
  ClientHeight   =   1920
  ClientLeft     =   2460
  ClientTop      =   3045
  ClientWidth    =   3030
  Height         =   2325
  Left           =   2400
  LinkTopic      =   "Form1"
  MaxButton      =   0   'False
  MDIChild       =   -1  'True
  ScaleHeight    =   1920
  ScaleWidth     =   3030
  Top            =   2700
  Width          =   3150
End
```

Este formulario se encuentra en el archivo FS_D.FRM

Con este formulario el usuario puede introducir los parámetros necesarios para ejecutar el comando Ejecutar con Break Point. Estos parámetros se introducen a través de dos cajas de texto, con las cuales se ingresa el nombre del registro que hará de Break Point y el valor que tiene que alcanzar dicho registro para que se detenga el flujo del programa.

Además de estas dos cajas de texto, el formulario contiene dos etiquetas, dos botones de órdenes y una caja de figuras para albergar los botones de órdenes.

3.2.5.4.1.11.Formulario Ensamblar

Este formulario se visualiza cada vez que el usuario desee ensamblar en la memoria RAM asignada para su uso (3000H - FFFFH) instrucciones individuales.

A través de este formulario el usuario puede introducir los parámetros solicitados para llevar a cabo el comando. El formulario es el siguiente:

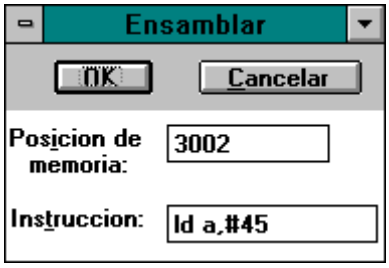


Figura 19: Formulario Ensamblar

cuyas propiedades son:

```
Begin Form F_Sistema_Ensamblar
  BorderStyle    =  1  'Fixed Single
  Caption        =  "Ensamblar"
  ClientHeight   =  1620
  ClientLeft     =  2415
  ClientTop      =  3510
  ClientWidth    =  2820
  Height         =  2025
  Left           =  2355
  LinkTopic      =  "Form1"
  MaxButton      =  0   'False
  MDIChild       =  -1  'True
  ScaleHeight    =  6.75
  ScaleMode      =  4  'Character
  ScaleWidth     =  23.5
  Top            =  3165
  Width          =  2940
End
```

Este formulario se encuentra dentro del archivo FS_E.FRM. Los parámetros que solicita son: la posición de memoria a partir de la cual se ensamblará la instrucción y la instrucción propiamente dicha. Dentro de este formulario se pueden ver dos etiquetas, dos cajas de texto, dos botones de órdenes y una caja de imágenes que alberga los botones de órdenes (finalidad estética).

3.2.5.4.1.12.Formulario Ver Contenido Puertos

Este formulario se activa cuando se desee ver el contenido de los puertos del sistema. El formulario es el siguiente:

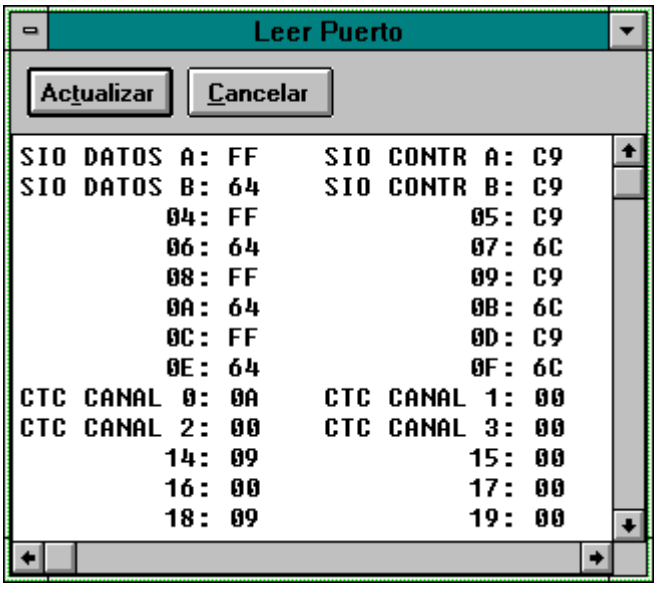


Figura 20: Formulario Leer Puertos

y sus propiedades son:

```
Begin Form F_Puertos_VerContenido
```

```
  Caption        =   "Leer Puerto"
  ClientHeight   =   3210
  ClientLeft     =   2385
  ClientTop      =   1635
  ClientWidth    =   2595
  Height         =   3615
  Left           =   2325
  LinkTopic      =   "Form1"
  MaxButton      =   0   'False
  MDIChild       =   -1  'True
  ScaleHeight    =   3210
  ScaleWidth     =   2595
  Tag            =   "Puertos"
  Top            =   1290
  Width          =   2715
End
```

Este formulario se encuentra en el archivo FP_VER.FRM

Dentro de este formulario hay un botón de órdenes que permite la actualización de los contenidos de los puertos que se están visualizando. Este formulario contiene además otro botón de órdenes, una caja de texto que permite la visualización de los contenidos de los puertos y una caja de imágenes que alberga los botones de órdenes.

3.2.5.4.1.13.Formulario Escribir Puertos

Este formulario se activa cuando se desee escribir (programar) algún puerto del sistema.

Figura 21: Formulario Escribir Puerto

Las propiedades de este formulario son:

```
Begin Form F_Puertos_Escribir
  BorderStyle    =   1  'Fixed Single
  Caption        =   "Llenar Puerto"
  ClientHeight   =   1785
  ClientLeft     =   1110
  ClientTop      =   3765
  ClientWidth    =   2505
  Height         =   2190
  Left           =   1050
  LinkTopic      =   "Form1"
  MaxButton      =   0   'False
  MDIChild       =   -1  'True
  ScaleHeight    =   1785
  ScaleWidth     =   2505
  Top            =   3420
  Width          =   2625
End
```

Este formulario se encuentra en el archivo FP_ESCR.FRM

Dentro de este formulario se puede definir el puerto que se desea programar y el contenido con el que se quiere programar a través de dos cajas de texto colocadas para tal fin. Se puede ver que dentro de él se encuentran ademas dos etiquetas, dos botones de órdenes y una caja de imágenes para albergar los botones de órdenes.

3.2.5.4.1.14.Formulario Informacion del Sistema

Este formulario se activa cada vez que se desee ver la informacion del sistema, a la cual se puede acceder a través del menú Ayuda y el submenú Acerca de...

El formulario es el siguiente:

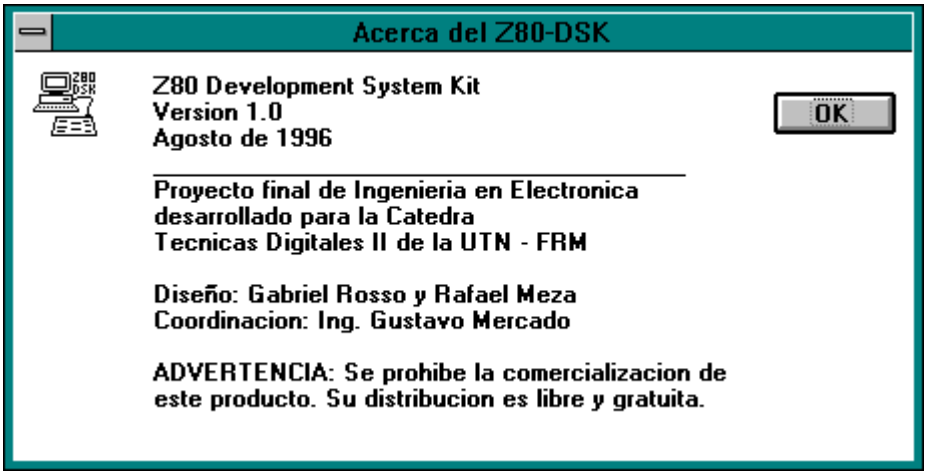


Figura 22: Formulario Información del Sistema

y sus propiedades son las siguientes:

```
Begin Form AcercaDelZ80DSK
  BorderStyle    =  3  'Fixed Double
  Caption        =  "Acerca del Z80-DSK"
  ClientHeight   =  3060
  ClientLeft     =  1725
  ClientTop      =  1785
  ClientWidth    =  6750
  Height         =  3465
  Left           =  1665
  LinkTopic      =  "Form1"
  MaxButton      =  0   'False
  MinButton      =  0   'False
  ScaleHeight    =  3060
  ScaleWidth     =  6750
  Top            =  1440
  Width          =  6870
End
```

Este formulario se encuentra en el archivo ACERCADE.FRM

Dentro de este formulario se puede ver el ícono del Z80 Development System Kit, una etiqueta donde se consigna la información del sistema y un botón de órdenes que se utiliza para finalizar con la visualización del formulario.

Las propiedades de estos objetos son:

```
Begin PictureBox IconoProyecto          'caja donde se ubica el ícono.
```

```
    AutoSize      =  -1  'True                ‘se ajusta automáticamente al tamaño del ícono.
    BorderStyle   =  0  'None
    Height        =  480
    Left          =  150
    Picture       =  ACERCADE.FRX:0000        ‘nombre del ícono.
    ScaleHeight   =  480
    ScaleWidth    =  480
    TabIndex      =  2
    Top           =  150
    Width         =  480
End
```

NOTA: se habrá podido observar que dentro del listado de las propiedades de los mismos objetos hay algunas propiedades que figuran en unos mientras que en otros no aparecen. Esto se debe a que Visual Basic lista solamente las propiedades cuyos valores hayas sido modificados respecto al que traen por defecto.

3.2.5.4.2.Comandos

3.2.5.4.2.1.Introducción

Una vez que se tienen los formularios diseñados, se pasa a la etapa de escribir el código (programa) para cada suceso, o sea escribir el código para cada procedimiento de suceso.

Para aclarar un poco más este concepto, veamos la siguiente ventana que se utiliza cuando se diseña la aplicación.

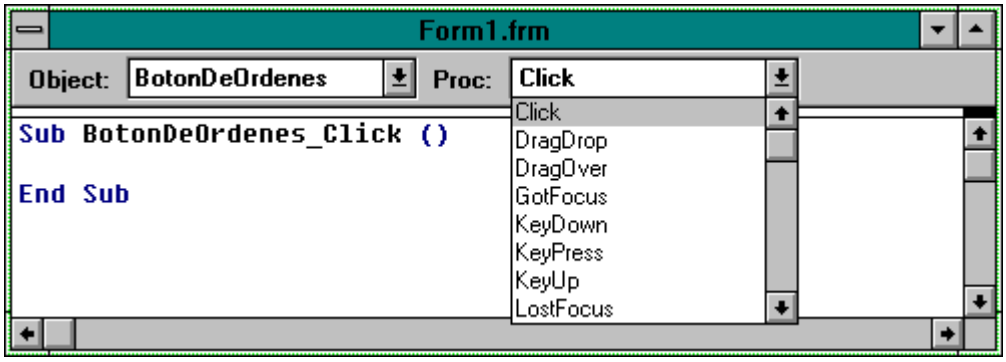


Figura 23: Procedimientos de suceso

Se puede ver que en la parte superior de color gris se encuentra la leyenda Object, aquí es donde aparecen listados los objetos que tiene el formulario, mientras donde se lee Proc: (Procedimientos) se listan los sucesos a los cuales responde este Objeto. Dentro de estos procedimientos podemos ver que este objeto, que es un botón de órdenes, reconoce los sucesos:

- Click: se hace click con el botón izquierdo del mouse sobre él.
- GotFocus: se ha enfocado a través del uso de la tecla TAB el objeto.
- KeyDown: se ha pulsado una tecla.
- KeyPress: se ha pulsado una tecla. La diferencia con el anterior es que en el anterior se ejecuta el código del procedimiento antes de enviar la tecla pulsada.
- KeyUp: se ha soltado una tecla que estaba siendo pulsada y retenida.
- LostFocus: se acaba de desenfocar el objeto a través del uso de la tecla TAB.

Dentro de la ventana anterior se encuentra escrito lo siguiente:

Sub BotonDeOrdenes_Click ()
'aqui va el código para este procedimiento de suceso.
End Sub

Las partes marcadas en negrita indican el comienzo y el final del procedimiento de suceso click para el objeto BotonDeOrdenes. Dentro de estos límites se escribe el código para este procedimiento, si se desea que este suceso tenga influencia sobre el programa general.

En base a los sucesos que reconocen los objetos introducidos en el formulario, se escribe el código para aquellos que satisfagan nuestras expectativas sobre lo que debe realizar el usuario para ejecutar un comando determinado. No se describirá el código propiamente dicho asociado con cada comando, sino que se verá la forma en que fueron implementados los códigos para realizarlo, o sea, la filosofía del diseño de cada uno de ellos.

3.2.5.4.2.2.Comando Nuevo

Este comando se implementa a través de la utilización del código necesario para inicializar las variables de tipo global que utiliza el sistema para la ejecución de los distintos comandos y para visualizar si hay un archivo abierto.

3.2.5.4.2.3.Comando Ingresar

Este comando se lo elige cuando se desea ingresar el contenido de un archivo con formato hexadecimal (*.HEX) a la memoria RAM de la placa de desarrollo asignada para uso del usuario (3000H - FFFFH). Para realizar el ingreso el usuario debe ir al menú Archivos, desplegarlo y hacer click en el submenu ABRIR. Una vez que se hace esto, se abre una ventana de gestion de archivos en el modo Ingresar Archivo, cuya forma es la siguiente:

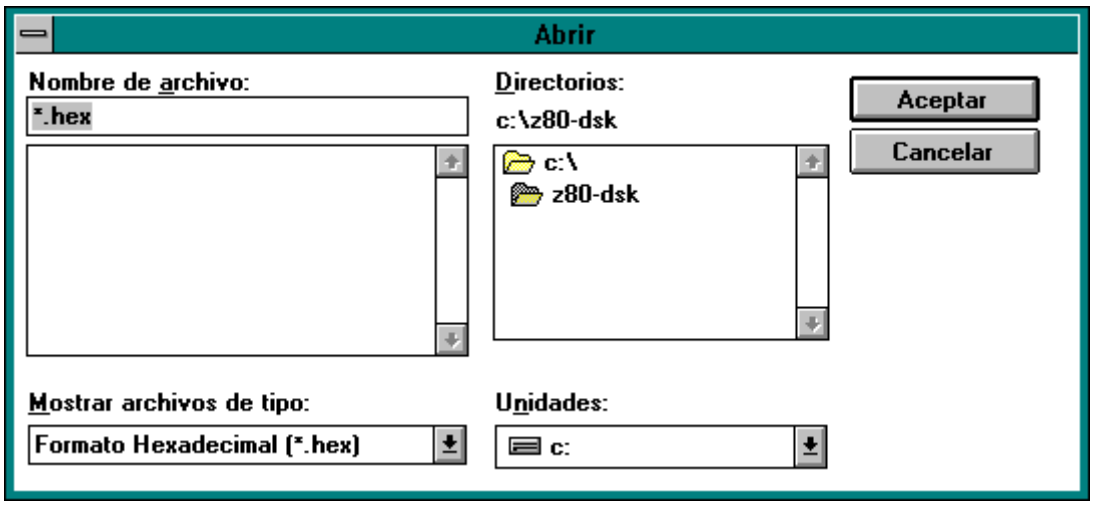


Figura 24: Formulario Abrir Archivo

Esta es una ventana general para el ingreso de archivos. Permite la seleccion de un archivo como asi tambien la ruta para encontrarlo. Esta ventana tiene una particularidad, si se elige el botón Aceptar devuelve el nombre y ruta completa del archivo seleccionado en MAYUSCULAS, mientras que lo hace en minusculas si se elige el botón Cancelar. En base a ello se llama al procedimiento AbrirArchivo utilizando como parametro el nombre del archivo provisto por la ventana anterior.

Elegido el archivo a traves de la ventana anterior, el codigo esta diseñado para realizar lo siguiente:

- controlar que el nombre del archivo exista (la primera vez que se abre la ventana de gestion de archivos y se pulsa Cancelar la ventana devuelve la cadena vacía) , que el mismo esté

en mayúsculas (se eligió Aceptar en la ventana de gestión de archivos) y que posea la extensión de los archivos con formato hexadecimal (*.HEX). Si se cumple con esto entonces:

- el archivo se abre y recupera.
- se controla que la posicion inicial de memoria de ensamblado se encuentre dentro de los límites asignados para uso del usuario.
    - se controla el formato del archivo y si tiene el formato usado se extraen los datos hexadecimales de él.
    - controlar que los datos quepan en memoria.
    - enviarlos a memoria.
    - mostrar el contenido de la memoria recién llenada.
    - colocar el nombre del archivo en la parte inferior izquierda del formulario padre.

3.2.5.4.2.4.Características de un archivo con formato hexadecimal (*.HEX)

Un archivo con formato hexadecimal posee la siguiente estructura:

```
:143000000022334455667788990000998877665544332211D3
:00000001FF
```

donde todas las lineas comienzan con dos puntos y la linea final tiene la misma estructura que la anterior.

Cada linea responde al siguiente formato:

Dos puntos + Cantidad Bytes Datos (en hexadecimal) + Posicion Inicial de Memoria para ubicacion de los datos + Código Datos + Datos (en nibbles) + Suma Chequeo

Para comprender mejor lo anterior veamos el siguiente ejemplo basado en la primera linea mostrada anteriormente:

- Cantidad Bytes Datos: 14 (en hexadecimal)
- Posicion Inicial Memoria: 3000
- Código Datos: 00 (indica que esa linea posee datos).
- Datos: 00223344556677889900009988776655443322 11
- Suma Chequeo: D3 (Calculada como el complemento a dos de la suma de todos los caracteres)

El Código de Datos tambien puede valer 01, indicando de esta forma que es la linea final de los datos. Normalmente esta linea no posee datos.

Una caracteristica de los programas con formato hexadecimal es que los espacios vacios en el linkeo de los modulos son rellenados con ceros (NOPs), por lo que la posicion de memoria a partir de la cual se encontrara el programa es la indicada por la primera linea del archivo. Otra caracteristica es que por linea se colocan como maximo 20H bytes de datos, y que la ultima línea nunca contiene datos y además está formada por los caracteres de la última línea del ejemplo anterior.

3.2.5.4.2.5.Comando Guardar

Este comando se utiliza cuando el usuario desea actualizar el contenido de un archivo que se encuentra abierto al momento de la ejecución del comando con el contenido de un bloque de memoria cuyos límites están dados por los límites del archivo abierto.

El código para la implementación de este comando está diseñado para realizar lo siguiente:

- pedir el contenido del bloque de memoria a la placa.
- configurar el contenido del bloque de memoria a formato hexadecimal.

- guardar el contenido con formato hexadecimal con el nombre del archivo que se encuentra abierto al momento de ejecutar este comando (**sobreescribir el archivo**).

3.2.5.4.2.6.Comando Guardar Como...

Este comando se utiliza cuando el usuario desea guardar el contenido de un bloque de memoria, cuyos límites quiere definir, con un nombre y una ruta que también quiere definir.

El código para la implementación de este comando está diseñado para realizar lo siguiente:

- mostrar el formulario FA_GUARD para que el usuario pueda definir los límites del bloque de memoria cuyo contenido desea guardar en un archivo con formato hexadecimal.
    - controlar que los valores colocados sean hexadecimales y que se encuentren dentro de los límites permitidos de la memoria RAM disponible para uso del usuario (depende de la selección que se haya hecho con el comando que permite elegir la extensión del mapa de memoria).
- mostrar una ventana de gestión de archivos en el modo Guardar Archivos (ver figura siguiente) para que se pueda seleccionar el nombre y la ruta del archivo.
    - controlar que el nombre del archivo exista (la primera vez que se abre la ventana de gestion de archivos y se pulsa Cancelar la ventana devuelve la cadena vacía) , que el mismo esté en mayúsculas (se eligió Aceptar en la ventana de gestión de archivos) y que posea la extensión de los archivos con formato hexadecimal (*.HEX), si no la tiene se colocara automáticamente. Si se cumple con esto entonces:
        - pedir el contenido del bloque de memoria a la placa.
        - configurar el contenido del bloque de memoria a formato hexadecimal.
        - guardar el contenido con formato hexadecimal en el archivo seleccionado.

3.2.5.4.2.7.Comando Imprimir

Este comando se elige cuando se desea tener una copia impresa del contenido del bloque de memoria, o de los puertos, o de los registros, que se están viendo en pantalla; o se desea imprimir el contenido del portapapeles. El código para la implementación de este comando está diseñado para realizar lo siguiente:

- detectar cuales formularios se encuentran abiertos para realizar la habilitación de las opciones correspondientes en el formulario FA_IMPR.
- mostrar el formulario FA_IMPR para que el usuario pueda elegir entre las opciones habilitadas.
- de acuerdo a la opción elegida:
    - obtener los datos a imprimirse
    - calcular la cantidad de páginas que se imprimirán con esos datos.
    - imprimir los datos, teniendo en cuenta que en cada página se debe imprimir un encabezamiento y el contenido correspondiente dejando un margen a la izquierda.

NOTA: el sistema utiliza para la impresión el controlador de impresión de Windows, por lo que si se desea realizar alguna modificación especial, como por ejemplo reestablecer el tamaño de la página, se deberá hacer a través del Administrador de Impresión que se encuentra dentro del grupo Principal.

3.2.5.4.2.8.Comando Salir

Este comando se utiliza para salir del software de PC del Z80-DSK. La instrucción de Visual Basic que lo realiza es la instrucción END.

3.2.5.4.2.9.Comando Definir Bloques de Memoria a Mostrar

El código necesario para llevar a cabo este comando está diseñado para realizar lo siguiente:

- mostrar el formulario FMM_D para que el usuario pueda definir los valores iniciales de los bloques de memoria de datos y del programa del usuario.
    - controlar que los valores ingresados sean hexadecimales y que se encuentren dentro de los límites permitidos.
        - asignar los valores establecidos en variables globales para que los puedan utilizar las distintas partes del proyecto.

3.2.5.4.2.10.Comando Mostrar Datos

Este comando le permite al usuario visualizar el contenido de un bloque de memoria destinado para el almacenamiento de datos a través de su valor hexadecimal.

Para la implementacion de este comando, el código está diseñado para realizar lo siguiente:

- pedir un bloque de memoria de 100H posiciones de memoria, o menor si la diferencia entre la posición final de memoria y el valor a partir del cual se pide el bloque es menor que 100H.
- mostrar el contenido del bloque de la siguiente manera:
    - columna izquierda: posiciones de memoria.
    - columna central: contenido en hexadecimal.
    - columna derecha: contenido en ASCII (si los mismos no son de control).
    - en cada fila se mostrarán 8H bytes.

3.2.5.4.2.10.1.Características:

Este comando debe permitir actualizar el contenido del bloque de memoria que se esté mostrando, ver el contenido del bloque inmediato inferior e inmediato superior y mostrar en forma cíclica, esto quiere decir que se debe:

- almacenar el primer valor que se muestra por si se desea actualizar el contenido que se muestra.
- almacenar el último valor que se muestra por si se desea visualizar el contenido del bloque inmediato superior al que se esta mostrando.
- controlar si se ha llegado al final de la memoria asignada para volver a mostrar desde el principio si se desea ver el contenido del bloque inmediato superior.
- controlar si se está en el comienzo de la memoria asignada para poder ir al final de la misma si se pretende ver el contenido del bloque inmediato inferior.

3.2.5.4.2.11.Comando Mostrar Programa del Usuario

Este comando permite visualizar el contenido de un bloque de la memoria asignada para uso del usuario ya sea en forma hexadecimal o a través de los mnemónicos correspondientes. Este comando posee las mismas características que el comando anterior.

Para implementar este comando se realizan los mismos pasos que los descriptos antes, con la salvedad de que si se pretende ver el contenido a través de los mnemónicos, el código deberá mostrarlos de la siguiente forma:

- columna izquierda: posiciones de memoria.
- columna central: código de la instrucción.
- columna derecha: mnemónico de la instruccion.

3.2.5.4.2.12.Comando Mostrar Programa del Sistema

Este comando se implementa de manera análoga al antes descripto, teniendo en cuenta que solamente se muestra el contenido de esta porción de memoria a través de los mnemónicos.

3.2.5.4.2.13.Comando Llenar Bloque Memoria

Para la implementación de este comando, el código está diseñado para realizar lo siguiente:

- mostrar el formulario FM_LCT en el modo Llenar Memoria (asignar la leyenda correspondiente a la parte superior del formulario) para que el usuario defina los límites del bloque de memoria que desea llenar y el contenido que desea que tenga ese bloque.
- controlar que los valores asignados sean hexadecimales y que se encuentren dentro de los valores permitidos.
    - formar una cadena, cuya longitud está dada por la diferencia entre los límites del bloque a llenar, con el contenido que se desea que tenga, repetido tantas veces como dicho contenido quepa en el bloque deseado.
    - llenar el bloque de la memoria de la placa con el contenido deseado.

3.2.5.4.2.14.Comando Comparar Bloques de Memoria

Para la implementación de este comando, el código está diseñado para realizar lo siguiente:

- mostrar el formulario FM_LCT en el modo Comparar Memoria (asignar la leyenda correspondiene a la parte superior del formulario) para que el usuario pueda definir los límites de los bloques cuyos contenidos desea comparar.
- controlar que los valores asignados sean hexadecimales, que se encuentren dentro de los valores permitidos y que la longitud de los bloques a comparar sean iguales (se utiliza como límite superior máximo del bloque nº 2 el valor final de memoria permitido para uso del usuario).
    - pedir el contenido del bloque nº 1 a la placa.
    - pedir el contenido del bloque nº 2 a la placa.
    - comparar sus valores.
    - mostrar el resultado obtenido.

3.2.5.4.2.15.Comando Trasladar Bloque de Memoria

Para la implementación de este comando, el código está diseñado para realizar lo siguiente:

- mostrar el formulario FM_LCT en el modo Trasladar Memoria (asignar la leyenda correspondiene a la parte superior del formulario) para que el usuario pueda definir los límites del bloque cuyo contenido desea trasladar a partir de una posicion de memoria que también puede definir (bloque nº2) .
- controlar que los valores asignados sean hexadecimales, que se encuentren dentro de los valores permitidos.
    - pedir el bloque nº 1 a la placa.
    - controlar si la longitud del contenido del bloque nº 1 es menor que la longitud del bloque nº 2 (teniendo en cuenta como principio el valor definido por el usuario para este bloque y como valor final la última posición de memoria que puede usar el usuario). Si es menor, se deberá recortar la parte final del contenido del bloque nº 1 hasta que quepa.
    - enviar el nuevo contenido del bloque de memoria nº 2 a la placa.

3.2.5.4.2.16.Comando Resetear Registros

Para la implementación de este comando, el código está diseñado para realizar lo siguiente:

- asignar los valores por defecto establecidos para los registros a las variables registros correspondientes.
- enviar los nuevos valores de los registros a la placa.
- actualizar los contenidos mostrados si se encuentra el FS_REGIS abierta.

3.2.5.4.2.17.Comando Examinar Registros

Este comando permite que el usuario visualice el contenido de los registros de la CPU, y que también pueda programarlos con determinados valores.

El codigo necesario para la implementación de este comando está diseñado para realizar lo siguiente:

- mostrar el formulario FS_REGIS para que el usuario vea el contenido que tienen los registros de la CPU, para lo cual se debe pedir sus valores a la placa.
- permitir que se cambien los valores que tienen los registros.
    - controlar que los nuevos valores sean hexadecimales y que sean válidos.
        - enviar los nuevos contenidos de los registros a la placa.
- permitir actualizar los contenidos mostrados de los registros.
    - pedir el contenido de los registros a la placa.
    - asignar sus valores a las cajas de texto correspondientes.
        - separar bit por bit el contenido del registro banderas.

3.2.5.4.2.18.Comando Definir Parámetros de Ejecución

Este comando permite que el usuario defina los parámetros necesarios para poder realizar la ejecución de un programa con break point.

El código necesario para la implementación de este comando está diseñado para realizar lo siguiente:

- mostrar el formulario FS_D para que el usuario pueda definir el registro que hará de break point y el valor que deberá alcanzar este registro para que se detenga la ejecución del programa.
    - controlar que el nombre asignado para el registro sea valido, esto quiere decir que sea uno de los registros del Z80, y que el valor asignado sea hexadecimal y permitido.
        - asignar los parámetros a las variables correspondientes.

3.2.5.4.2.19.Comando Ejecutar en Tiempo Real

Este código de este comando permite que el usuario ejecute un programa que se encuentra en la memoria de la placa en tiempo real. El programa se ejecuta a partir de la posición de memoria cuyo valor es extraído del contenido de la variable registro PC, por lo que la única función de este comando es la de enviar a la placa la orden de ejecución. A partir de ese momento, el software de PC pierde el control de la placa y queda en un estado ocioso. Si se desea que el software de PC vuelva a tener el control, se deberá incluir al final del programa que se desea ejecutar una instrucción de salto incondicional como se especifica en el Manual del Usuario.

3.2.5.4.2.20.Comando Ejecutar en Tiempo Virtual

Este comando permite que se ejecute un programa paso a paso en forma ininterrumpida utilizando mientras no se elija el comando Detener.

El codigo necesario para la implementación de este comando está diseñado para realizar lo siguiente:

- colocar en VERDADERO una bandera utilizada para indicar que se ha elegido esta forma de ejecución.
- ejecutar una instrucción a través del comando Paso a Paso.
- controlar si se ejecutó el comando Detener. Si se ejecutó, detener la ejecución colocando en FALSO la bandera correspondiente; si no, volver al punto anterior.

Se utiliza una bandera para indicar si se eligió este comando porque la ejecución ininterrumpida de las instrucciones a través del comando paso a paso se implementa con la función DoEvents () que la provee Visual Basic. Esta función se ejecuta cada vez que la aplicación se encuentra en un estado ocioso, lo cual es beneficioso para este comando debido a las razones que se expondrán a continuación:

Una de las características de Visual Basic es que mientras se está ejecutando el código de un procedimiento, el sistema no responde de manera visible a los sucesos que se estén

produciendo en ese momento; por el contrario, almacena en una tabla todos estos sucesos en el orden que están apareciendo, y, una vez que el sistema vuelve a un estado ocioso, los atiende en el orden en que aparecieron.

La implementación de este comando está pensada para que el sistema ejecute una instrucción, luego atienda los sucesos que llegaron mientras se ejecutaba esa instrucción, y una vez que finalice con los sucesos pendientes, ejecute otra instrucción. Así hasta que se detenga la ejecución con el comando Detener.

3.2.5.4.2.21.Comando Ejecutar Paso a Paso

Una de las partes más desafiantes al desarrollar este proyecto fue la implementación de un comando que permitiera la ejecución de un programa instrucción por instrucción o como suele denominarse, paso a paso.

Cuando se le da la orden a la placa para que ejecute un programa a partir de la posición de memoria indicada por el contenido de la variable del registro PC, el software de PC pierde el control de la placa a menos que el usuario disponga lo contrario. La manera de que el software de PC recupere el control, después de ejecutar una instrucción, es insertando una bifurcación a continuación de la instrucción que se quiere ejecutar, alterando de esta manera el flujo del programa del usuario. La bifurcación que se introduce en el programa del usuario debe apuntar a una rutina que se encuentre dentro del software de placa que le permita al software de PC recuperar el control de la placa. Hay muchas instrucciones de bifuración que pueden ser ensambladas en la posición en la que debe bifurcar el programa, pero la que se consideró que más se adecuaba para este propósito fue la instrucción de bifurcación CALL, ya que la misma permite almacenar el estado del contador de programa. Esta instrucción se la utiliza de manera incondicional, es decir que a continuación se debe colocar la dirección de bifurcación, la cual se obtiene de la dirección de comienzo de la rutina que devolverá el control de la placa al software de PC. Pero no todas las instrucciones deben llevar una instrucción para el retorno controlado (IRC) a continuación de su código, ya que en las instrucciones de salto, sean relativos o no, condicionales o no, la siguiente instrucción a ejecutarse no siempre es la instrucción que se encuentra a continuación, por lo que se debió realizar un análisis de las mismas, cuyo resultado, traducido en la potencialidad del software de PC de seguir un programa paso a paso, se encuentra en el código que se diseñó para la implementación del comando.

El codigo necesario para la implementación de este comando está diseñado para realizar lo siguiente:

- pedir un bloque de memoria de 7 bytes (4 corresponden al codigo de instrucción más largo y 3 al codigo de la IRC) a partir de la posición de memoria determinada por el contenido de la variable del registro PC.
- desensamblar contenido del bloque de memoria de 7 bytes para encontrar la instrucción correspondiente.
- analizar la instrucción.
- si la instrucción no pertenece al grupo de instrucciones cuyo mnemónico comienza con JP, JR o DJNZ ensamblar la IRC a continuación de la instrucción.
    - si la instrucción pertenece al grupo anterior, entonces:
    - ensamblar la IRC en la dirección determinada por la constante de la instrucción si es una instrucción de bifurcación incondicional y si la IRC no sobreescribe algún byte de la instrucción a ejecutar.
        - si hay conflicto para ensamblar la IRC, ensamblarla a continuación de la instrucción a ejecutar y avisar de lo sucedido.
    - si es una instrucción de bifurcación condicional, ensamblar una IRC a continuación de la instrucción y otra IRC en la dirección de bifurcación, siempre y cuando no sobreescriba ni la instrucción a ejecutar ni la IRC que se encuentra a continuación.
        - si hay conflicto para ensamblar la IRC en la dirección de bifurcación, no ensamblarla y avisar de lo sucedido.

- ejecutar la instrucción en Tiempo Real
- pedir el contenido de los registros a la placa para actualizar sus contenidos.
- pedir el contenido de los puertos a la placa para actualizar sus contenidos.
- restaurar el programa del usuario llenando el bloque de memoria modificado con el bloque pedido de 7 bytes.

NOTA: la instrucción para el retorno controlado (IRC) es ensamblada en la posición de memoria adecuada si se trata de una instrucción JP, JR o DJNZ siempre que no cree conflicto, ya que si lo hay siempre tiene prioridad la IRC que se coloca a continuación de la instrucción a ejecutarse individualmente.

El set de instrucciones contiene otras instrucciones de bifurcación además de las enmarcadas en la explicación, pero el software de PC no realiza ningún tipo de control sobre ellas, por lo que se deben tomar las precauciones necesarias para que no se produzca ningún efecto indeseado en la ejecución de un programa a través del uso de este comando.

3.2.5.4.2.22.Comando Ejecutar con Break Point

Este comando permite la ejecución de un programa de manera ininterrumpida mientras el contenido del registro que actúa como Break Point no alcance el valor establecido con el comando Definir Parámetros de Ejecución, o mientras no se ejecute el comando Detener.

La implementación de este comando es de manera análoga al comando Ejecutar en Tiempo Virtual, con la salvedad de que se incluye otra condición para la detención de la ejecución.

Nota: tanto este comando como el comando Ejecutar en Tiempo Virtual habilitan el comando Detener cuando son ejecutados.

3.2.5.4.2.23.Comando Detener

Este comando se utiliza para detener la ejecución de programas que se están ejecutando en Tiempo Virtual o a través del uso de Break Points.

El código de este comando coloca en FALSO la bandera que utiliza la función DoEvents () (ver comando Ejecutar en Tiempo Virtual) para realizar la ejecución de una nueva instrucción del programa.

3.2.5.4.2.24.Comando Ensamblar

El codigo necesario para la implementación de este comando está diseñado para realizar lo siguiente:

- mostrar el formulario FS_E para permitir que el usuario de defina la instrucción y la posición de memoria a partir de la cual quiere ensamblarla.
    - controlar que el valor asignado como dirección de memoria sea hexadecimal y que se encuentre dentro de los valores permitidos.
    - controlar que la instrucción que se pretende ensamblar pertenezca al set de instrucciones del microprocesador.
        - obtener el código de la instrucción.
        - controlar si la instrucción cabe en memoria.
        - ensamblar la instrucción en memoria a partir de la posición de memoria establecida.

3.2.5.4.2.25.Comando Ver Contenido Puertos

El codigo necesario para la implementación de este comando está diseñado para realizar lo siguiente:

- mostrar el formulario FP_VER para que el usuario pueda ver el contenido de los puertos.

- pedir el contenido de todos los puertos que permite utilizar el sistema de acuerdo con el mapa de entrada/salida establecido con el comando Mapa de Puertos Extendido.
- mostrar los puertos y sus contenidos de la siguiente forma:
    - si el número del puerto que se va a mostrar pertenece a alguno de los puertos que utiliza el sistema, en vez de mostrar el número del puerto, mostrar el nombre que tiene el puerto.
    - colocar el símbolo dos puntos a continuación del número o nombre del puerto.
    - colocar el contenido del puerto después de los dos puntos.
    - mostrar en dos columnas.

3.2.5.4.2.26.Comando Escribir Puertos

El codigo necesario para la implementación de este comando está diseñado para realizar lo siguiente:

- mostrar el formulario FP_ESCR para que el usuario pueda definir el puerto y el contenido con el que lo quiere escribir (programar).
    - controlar que los valores sean hexadecimales.
    - controlar que el puerto a escribir no sea el puerto utilizado por el temporizador de la comunicación serie (puerto 0 del CTC).
        - controlar si el puerto a escribir es el puerto del SIO utilizado por el sistema. Si lo es controlar que el valor con el que se lo quiere programar no sea mayor que 3 (si es mayor que 3 se está intentando reprogramar el puerto serie) y que no se produzcan dos escrituras seguidas a este puerto sin una lectura intermedia (para leer el contenido del RR0, RR1 o RR2 del puerto).
        - enviar la orden a la placa para que se programe el puerto elegido con el valor deseado.

3.2.5.4.2.27.Comando Mapa de Memoria Extendido

Este comando permite elegir el límite superior del mapa de memoria. Se utiliza para ello una variable global, cuyo valor es modificado por este comando para dar los dos mapas de memoria posibles.

3.2.5.4.2.28.Comando Mapa de Puertos Extendido

Este comando permite elegir el límite superior del mapa de entrada/salida. Se utiliza para ello una variable global, cuyo valor es modificado por este comando para dar los dos mapas de entrada/salida posibles.

3.2.5.4.3.Ensamblador / Desensamblador

Otro de los desafíos al desarrollar este proyecto fue la implementación, dentro del software de PC, de un ensamblador/desensamblador de línea. Este tipo de ensamblador/desensamblador es muy útil en estas herramientas de desarrollo, ya que permite corregir los pequeños errores que se cometan en la lógica de un programa que se está probando, reduciendo así los tiempos de diseño y puesta a punto de un sistema.

Para implementar este sistema se realizó un análisis del set de instrucciones del microprocesador teniendo en cuenta las características que presentan las mismas desde dos puntos de vista, uno a partir de los códigos de las instrucciones y otro a partir de los mnemónicos. Los resultados obtenidos derivaron en la confección de dos tablas paralelas, donde en una se consignaron los códigos de las instrucciones y en la otra el mnemónico correspondiente (se dice que son paralelas porque los contenidos de las tablas se corresponden) de todo el set de instrucciones del microprocesador y que se puede ver en el listado general del código, dentro del procedimiento Equivalencias que se encuentra en el módulo de código DESENSAM.BAS. De estas tablas se puede extraer que el ordenamiento lógico de las instrucciones es el siguiente:

1. instrucciones con código de operación de un byte.

2. instrucciones con código de operación de dos bytes.
3. instrucciones con código de operación de un byte + una constante de un byte.
4. instrucciones con código de operación de un byte + una constante de dos bytes.
5. instrucciones con código de operación de dos bytes + una constante de dos bytes.
6. instrucciones con código de operación de dos bytes + un índice de un byte.
7. instrucciones con código de operación de tres bytes + una constante de un byte.
8. instrucciones de bifurcación incondicional.
9. la instrucción OUT constante, A
10. instrucciones con código de operación de dos bytes + constante de un byte + un índice de un byte.
11. instrucciones de bifurcación a direcciones relativas.

Para llegar a este tipo de ordenamiento lógico de las instrucciones, primero se hizo una selección de las instrucciones por la cantidad de bytes que posee su código de instrucción. De esta primera selección se extrajeron aquellas cuyo código de operación coincide con su código de instrucción, o sea las instrucciones directas (término usado por nosotros porque estas instrucciones se ensamblan y desensamblan por comparación directa solamente). Luego se seleccionaron aquellas que necesitan de una constante para completar el código de instrucción, después se separaron las que utilizan un índice, y finalmente se colocaron aquellas que eran combinación de las anteriores. La separación final en los distintos grupos como se ve actualmente en el procedimiento Equivalencias, estuvo determinada por problemas ocurridos al ensamblar o desensamblar durante la fase de diseño.

La idea de tener este tipo de estructuración de los datos radica en la forma en que se procederá a ensamblar y desensamblar instrucciones, que es por comparación.

3.2.5.4.3.1.Ensamblador

Si se analiza la tabla correspondiente a los mnemónicos se advertirá, por comparación con el listado del set de instrucciones, que:

- las instrucciones directas se encuentran de forma completa.
- las instrucciones que usan una constante, tienen un signo # en lugar de la constante (si la misma no se encuentra entre paréntesis ya que sino se encontrará el paréntesis solamente).
- las instrucciones indexadas no tienen el índice.

Para poder comparar la instrucción introducida por el usuario se la debe “desmenuzar”, esto quiere decir que:

- si hay un símbolo de numeral (#) extraer la constante cuya longitud depende del grupo de la tabla que se esté analizando.
- si dentro de paréntesis no hay numeral ni simbolos de indexación extraer la constante.
- si dentro de paréntesis hay símbolo de indexación extraer el índice.

  y así dejar la instrucción de forma parecida a como se encuentra en la tabla para realizar la comparación. Si al realizar la comparación de la instrucción introducida por el usuario, de forma “desmenuzada”, con las de la tabla se encuentra la equivalencia, entonces se procede a extraer el código de la tabla paralela e insertarle la constante o el índice que le corresponde de la manera correcta (si le corresponde).

NOTA: para más detalles sobre la manera precisa de cómo se implementa el ensamblador, se puede seguir el código escrito para tal efecto, ya que el mismo se ha tratado de hacer lo más claro posible, y con los conceptos básicos detallados antes, creemos que no tendrá inconvenientes en seguirlo y comprenderlo con profundidad.

3.2.5.4.3.2.Desensamblador

El desensamblador funciona de manera análoga al ensamblador ya que la base de comparación es la misma, una tabla con todos los códigos que tienen las instrucciones.

La forma de implementar esta función es seccionando en forma secuencial las partes adecuadas de los datos para realizar la comparación con los distintos grupos de códigos que se encuentran en la tabla de Códigos.

NOTA: El código escrito para realizar esta función está estructurado de forma que sea muy fácil su estudio y seguimiento. Para su comprensión basta tener a mano el set de instrucciones del microprocesador y el listado de las tablas, tanto códigos como mnemónicos.

3.2.5.4.4.Rutina para el comienzo de la aplicación

El software de PC comienza con la ejecución del procedimiento Sub Main que se ubica dentro del módulo de código RUTGRAL.BAS. A partir de allí, el código permite que el usuario defina el puerto serie que utilizará la PC con el sistema y hace que el sistema se quede esperando la llegada de la información de la placa que le indica que acaba de ser reseteada. Mientras este comando no llegue, el código continuará su espera, permitiéndole al usuario que redefina el puerto por si el puerto elegido es el incorrecto. Una vez que le llega el reset de la placa, inicializa el sistema dandole valores por defecto a las variables correspondientes, envía la orden a la placa de inicializar las variables registros del software de placa y muestra el formulario principal.

3.2.5.4.5.Comunicación con la placa

La comunicación serie de una aplicación Visual Basic está administrada por una herramienta provista por Visual Basic para tal efecto. El software de PC delega esta responsabilidad a dicha herramienta y centra su atención en la forma en que desea comunicarse con la placa. Todos los comandos, tarde o temprano, desembocan en las rutinas de comunicación con la placa. De todas las rutinas que se pueden ver dentro del módulo de código COMPLACA.BAS, las que realmente entablan la comunicación con la placa son los procedimientos TxCadena y TxCadenaRxDatos. Dentro de los comandos empleados para la comunicación con la placa, hay algunos que solamente dan órdenes sin esperar resultados, mientras que otros esperan de la placa el envío de datos. El procedimiento TxCadena es el encargado de enviar la órden a la placa para que ejecute un comando determinado pero no espera más respuesta de ella que el código correspondiente a que realizó correctamente la petición. El otro procedimiento determina la cantidad de bytes que espera recibir de la placa, luego de enviarle el comando correspondiente, y se queda esperando el arribo de esos bytes.

El sistema de comunicación está pensado de manera que una vez que se entabla la comunicación con la placa, se activa un temporizador que evita que el sistema se "plante" por algún inconveniente con la comunicación. Una falla común puede ser que el software de PC se quede esperando la contestación de la placa, que por motivos desconocidos no responde, entonces el temporizador hace que se aborte la espera, indica de la falla y devuelve el control al programa. El sistema de comunicación está también diseñado para que se detecte un reset de la placa de desarrollo y se ejecute el código asociado a este comando externo, y para que informe sobre posibles fallas en la comunicación.

3.2.5.4.5.1.Protocolo

Dentro del módulo de código COMPLACA.BAS se encuentran las rutinas que conforman la cadena que se transmitirá a la placa para que la misma ejecute el comando que se le está solicitando y de esta forma poder administrarla. Las cadenas conformadas por estas rutinas responden al protocolo de comunicación ya visto y los parámetros del protocolo que no son suministrados por la fuente son automáticamente completados con "ceros". Una vez que se tiene la cadena en formato caracter lista para la transmisión, se la convierte a formato binario para enviarla a la placa.

3.2.5.4.6.Comunicación entre formularios

Los comandos del software de PC no son independientes, sino que por el contrario, interactúan para darle flexibilidad al sistema. El código común para todos los formularios está diseñado para implementar lo siguiente:

- el software de PC detecta automáticamente un reset de la placa, a excepción de que esté comunicándose con ella.
- cuando se corre por primera vez el programa o se ejecuta el comando Nuevo, se inicializan todas las variables a sus valores por defecto.
- cuando se abre un archivo, los límites del archivo se asignan a los límites del comando Guardar Como... y se abre el formulario FMM_PU para mostrar el contenido del archivo abierto.
- cuando se resetean los registros, si la ventana de los mismos se encuentra abierta, se actualizan sus valores.
- cuando se ejecuta (a excepción de la ejecución en Tiempo Real) un programa se actualizan los valores de los puertos y de los registros si las ventanas correspondientes se encuentran abiertas.

4. Bibliografía y Documentación

- Z80 CPU Central Processing Unit Technical Manual de Zilog.
- Z80 SIO Technical Manual.
- Z80 PIO Technical Manual.
- Z80 CTC Counter/Timer Circuit Technical Manual.
- PROICE-Z80/PC Iwasaki Electronics Co. Operation Manual.
- PROASMII 4/8 bit General Cross (Macro) Assembler User's Manual de Iwasaki Electronics Co.
- AVCESSORIES, Utility Programs, User's Manual de AVOCET Systems Inc.
- MOS Memory Data Book. Texas Instruments.
- MAXIM 1994 New Releases Data Book Volume III.
- ECG Semiconductors.
- ECG TTL Data Book.
- DE GIUSTI, Armando E., "DESCRIPCION Y VALIDACION DE HARDWARE", De. Preliminar, V Escuela Brasileña Argentina de Informática, 1991, Río de Janeiro, Brasil.
- TANENBAUM, A. S., "ORGANIZACION DE ORDENADORES, UN ENFOQUE ESTRUCTURADO", Ed. 2da, Prentice Hall Hispanoamericana, 1985, ISBN 0-13-854489-1.
- TANENBAUM, A. S., "ORGANIZACIÓN DE ORDENADORES, UN ENFOQUE ESTRUCTURADO", De. 3ra, Prentice Hall Hispanoamericana, 1992, ISBN 0-13-854662-2.
- TANENBAUM, A. S., "SISTEMAS OPERATIVOS DISEÑO E IMPLEMENTACIÓN", Prentice Hall Hispanoamericana, 1988, México, ISBN 968-880-153-4.
- GODFREY, J. Terry, "LENGUAJE ENSAMBLADOR PARA MICROCOMPUTADORAS IBM, PARA PRINCIPIANTES Y AVANZADOS", Prentice Hall Hispanoamericana, 1991.
- Microsoft VISUAL BASIC - Versión 3.0 - Programmer´s Guide - 1993.
- Microsoft VISUAL BASIC - Versión 3.0 - Language Reference - 1993.

5. Apéndice A: La programación en Visual Basic

5.1. Por qué Windows y por qué Visual Basic? [1]

Las interfaces gráficos de usuario, o GUI (*Graphical User Interface*) han revolucionado la industria de las microcomputadoras. Han demostrado que el proverbio "vale mas una imagen que mil palabras" no ha perdido su validez. En lugar del critico carácter de aviso de ordenes C:> que los usuarios del DOS utilizan (y muchos han temido), encuentran un escritorio poblado de iconos. Todo esto da una imagen gráfica de lo que puede ofrecer la computadora. La siguiente figura muestra un ejemplo típico de un escritorio Windows

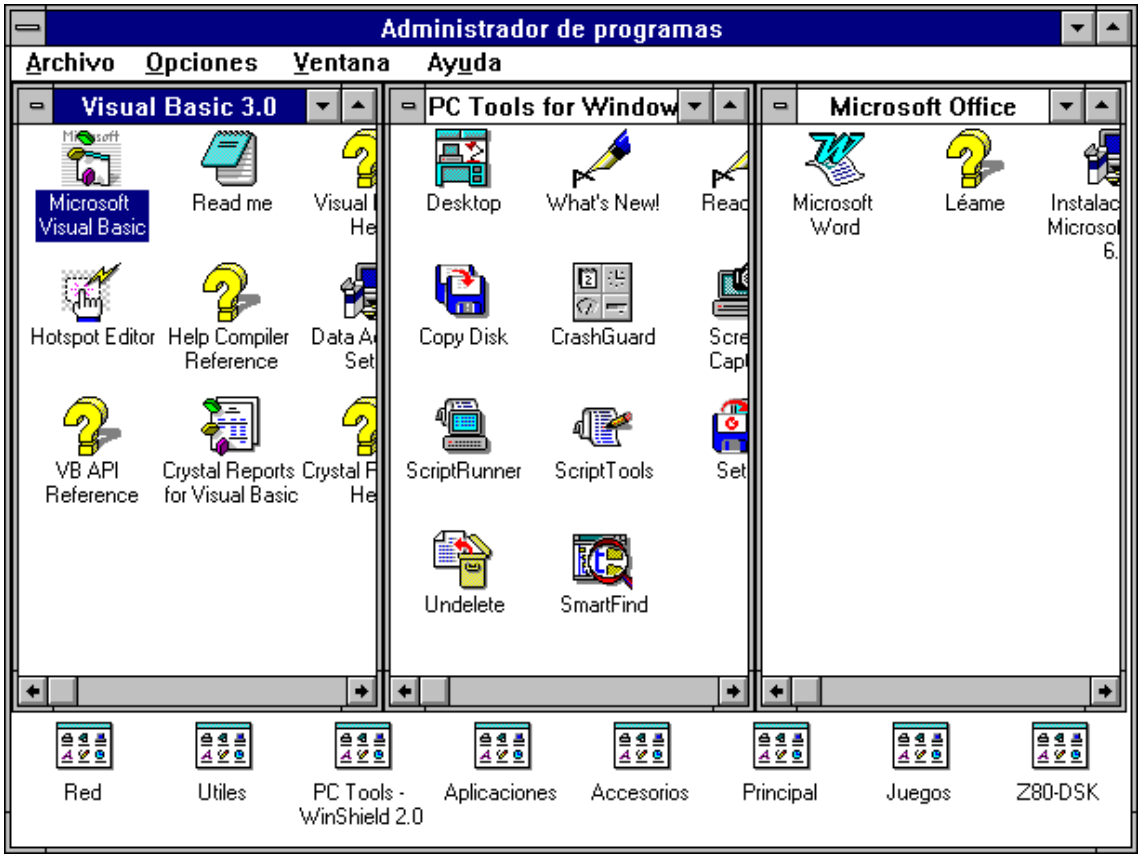


Figura 25: Administrador de programas

Quizá mas importante a largo plazo que el aspecto de las aplicaciones para Windows es la *sensación* que proporcionan. Las aplicaciones para Windows tienen generalmente una interfaz de usuario consistente. Esto significa que los usuarios disponen de mas tiempo para dominar la aplicación sin tener que preocuparse de que teclas deben pulsarse, dentro de los menús y cuadros de dialogo.

Todo esto tiene un precio; antes de la existencia de Visual Basic, el desarrollo de aplicaciones para Windows era mucho mas complicado que desarrollar aplicaciones para DOS. Los programadores tenían que preocuparse mas de lo que estaba haciendo el ratón, de donde estaba el usuario dentro de un menú y de si estaba realizando un click o un doble click en un punto de terminado. Desarrollar un aplicación para Windows requería expertos programadores en C, e incluso estos tenían problemas.

Visual Basic ha cambiado esta situación. Se pueden desarrollar aplicaciones para Windows en una fracción del tiempo que se necesitaba anteriormente. Los errores de

[1] Manual de Visual Basic 3 para Windows. Gary Cornell. Editorial Osborne Mc. Graw Hill. Pag. 1 a 3.

programaciones (gazapos, *bugs*) no se generan tan frecuentemente, y si lo hacen son mucho mas sencillos de detectar y solventar. Programar para Windows se ha convertido en algo divertido (por lo menos la mayor parte del tiempo).

No se pierde mucho en características: las aplicaciones desarrolladas en Visual Basic se ejecutan rápidamente. Esto no significa que se pueda eliminar el C o el lenguaje ensamblador en la programación para Windows; la extensión del Visual Basic todavía requiere herramientas disponibles, de momento, solo en estos dos lenguajes.

5.2. Programación en Visual Basic frente a la programación tradicional [2]

Lo que hace que Visual Basic sea diferente de cualquier otra herramienta de programación es la facilidad con la que se puede diseñar una pantalla. Se puede dibujar literalmente la interfaz de usuario, parecido a la forma de utilizar un programa de dibujo. Además, una vez se ha terminado de dibujar la interfaz, los botones de ordenes, cajas de texto y otros controles que se han colocado en una ventana en blanco reconocerán automáticamente acciones del usuario, tales como el movimiento del ratón y los click de los botones.

Tan solo después de diseñar la interfaz de usuario es cuando se empieza a hacer algo que se parezca a programar. Los objetos en Visual Basic *reconocerán* sucesos como los clicks del ratón; la forma en que los objetos respondan dependerá del código que se escriba. Se necesitara escribir código para que los controles respondan a los sucesos. Todo esto hace que Visual Basic sea diferente de la programación convencional.

Los programas en los lenguajes de programación convencionales se ejecutan de arriba abajo. En los antiguos lenguajes de programación la ejecución comienza en la primera línea y se desplaza con el flujo del programa a las distintas partes según se necesite. Un programa en Visual Basic funciona de un modo totalmente diferente. El núcleo de un programa en Visual Basic es un conjunto de diferentes partes de código que son *activadas* por, y que solamente responden a, los sucesos que se les ha indicado que reconozcan. Esto es un avance fundamental. Ahora, en lugar de diseñar un programa que haga lo que el programador piense que debe hacer, el usuario tiene el control.

La mayor parte del código de programación en Visual Basic indica al programador el modo de responder a determinados sucesos, como el click del ratón, en lo que en Visual Basic se denominan *procedimientos de suceso*. Esencialmente, cualquier cosa ejecutable en Visual Basic es, o bien un procedimiento de suceso, o es utilizado por un procedimento de suceso para ayudar al procedimiento a realizar su trabajo. De hecho, para enfatizar que Visual Basic es diferente de los lenguajes normales de programación, la documentación utiliza el termino *proyecto*, en lugar de *programa*, para referirse a la combinación de código de programación e interfaz de usuario que hacen que sea posible una aplicación en Visual Basic.

5.3. Características de los proyectos

Se pretende dar una breve descripción a la programación por eventos y a los objetos que en ella intervienen para que se pueda comprender el código escrito para el desarrollo del software de PC del Z80 Development System Kit.

5.3.1. Formularios

Los proyectos que se desarrollan a través del lenguaje de programación Visual Basic constan de *formularios*, los cuales responden de acuerdo a la posición jerárquica que ocupen dentro del proyecto. Existen tres tipos de formularios: formularios *padres*, formularios *hijos* y formularios independientes. Prácticamente, un formulario es una ventana de la aplicación.

[2] Manual de Visual Basic 3 para Windows. Gary Cornell. Editorial Osborne Mc. Graw Hill. Pag. 3-4.

5.3.1.1.Formularios padres (MDI Form) [3]:

MDI significa *Multiple Document Interface*, (Interfaz múltiple de documentos), que es el termino utilizado por Microsoft para un entorno de ventanas como en muchos procesadores de texto y hojas de calculo en donde una ventana, denominada *contenedor* MDI o *padre* MDI, contiene múltiples ventanas, denominadas normalmente formularios *hijos*. Por ejemplo, se puede utilizar un contenedor MDI para permitir al usuario trabajara en dos ventanas separadas en la misma aplicación.

El formulario contenedor MDI ocupa toda la pantalla; el usuario no puede modificar el tamaño del formulario contenedor, aunque si puede minimizarlo a un icono. solo se puede tener un formulario MDI en un proyecto, y dicho formulario deber ser el formulario inicial (el primero que se carga cuando se corre una aplicación).

Una de las características de los formularios MDI de Visual Basic es que los menús del formulario contenedor cambian al cambiar el foco entre los formularios hijos. Lo que permite trabajar con menús específicos para cada formulario hijo.

5.3.1.2.Formularios hijos e independientes:

Los formularios son todos independientes a menos que se indique dentro de las propiedades del formulario que dicho formulario es un formulario hijo. Que un formulario sea independiente implica que el mismo no se encontrara enmarcado ni sujeto al formulario MDI o padre. Esta ventaja es muchas veces útil cuando se desea mostrar una ventana en la cual se piden parámetros para realizar una determinada acción dentro del proyecto, como por ejemplo determinar que es lo que se quiere imprimir. Normalmente esta ventana no permanece en pantalla hasta que se cierra la aplicación, sino que se debe cerrar la misma para seguir con el curso de la aplicación.

Las características de los formularios hijos ya fueron abordadas en el punto anterior (ver formularios padres).

5.3.1.3.Objetos

Se denominan *Objetos* a los elementos que puede contener un formulario en su interior y se utilizan para comunicar el código del proyecto con el usuario. La elección de un determinado objeto dependerá de lo que se espere del usuario para realizar una determinada acción. Si por ejemplo se ve el código de un formulario mientras se desarrolla una aplicación Visual Basic se vera los siguiente:

[3] Manual de Visual Basic 3 para Windows. Gary Cornell. Editorial Osborne Mc. Graw Hill. Pag. 373.

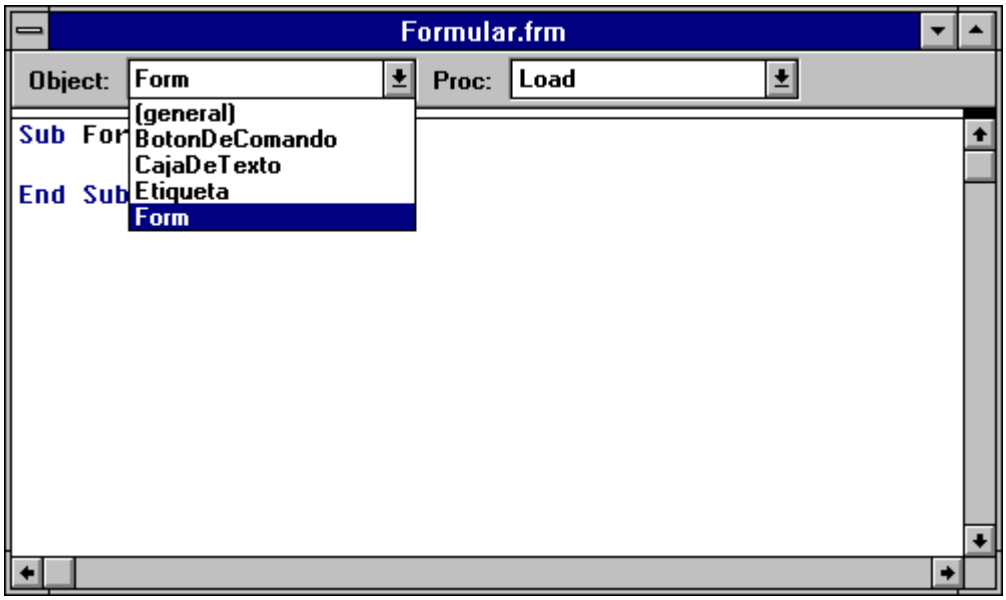


Figura 26: Objetos de un formulario

Se puede ver en la lista OBJECT el listado de los objetos que componen el formulario.

5.3.1.4.Procedimientos

Cada objeto tiene la facultad de responder a una lista de *procedimientos* o eventos que en el ocurran. Por ejemplo en la siguiente figura se da el listado parcial de los eventos a los que responde el objeto Botón de Comando.

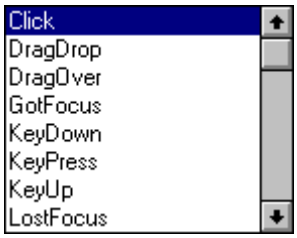


Figura 27: Procedimientos de suceso para un objeto

5.3.1.5.Características comunes de los formularios:

Una característica común que tienen los formularios es el Objeto *General*. Este objeto contiene las declaraciones de las variables que se desea que sean únicas en el ámbito del formulario, ya que las variables y constantes que se utilicen dentro de un procedimiento es independiente de las demás, ya sea de otros procedimientos u objetos.

Dentro de los procedimientos del objeto general se ubican todos los procedimientos o funciones que el programador diseñe. El ámbito de aplicación este procedimiento o función se limita al formulario.

5.3.2. Módulos de código

En todo proyecto se desea que ciertas variables, constantes, procedimientos y funciones sean utilizadas por distintos formularios del proyecto. Para que ello suceda se utilizan los módulos de código. En Objeto General se definen las variables y constantes comunes que utilizaran los formularios del proyecto; y dentro de los procedimientos figuran los procedimientos y funciones que son utilizadas en distintas partes del proyecto.

6. Apéndice B: Prototipo del Z80 Development System Kit

Previo al desarrollo actual de los softwares de operación y administración del sistema, se escribió un software prototipo para residir en la EPROM de la placa de desarrollo, y desde allí permitir la operación y administración del sistema. Se puede decir que, por sus características, esta es una versión para DOS del Z80 DSK. Este software está íntegramente diseñado en lenguaje ensamblador y a continuación se lo describe.

6.1. Detalle de la confección del software de PC prototipo del Z80 Development System Kit

6.1.1. Introducción

El sistema Z80 DSK prototipo se basa en una placa tipo general conteniendo un Z80 CPU junto con 32 Kb de memoria (8 Kb ROM y 24 Kb RAM) y todos sus periféricos (idem a la versión final).

Esta placa estará interconectada con una PC a través del puerto serie (RS232C), la cual trabajará como terminal DUMMY, o sea que todo lo que reciba por el teclado lo enviará al puerto serie, como asimismo todo lo que reciba por el puerto serie lo enviará a la pantalla. El software encargado de que la PC realice la actividad antes mencionada es el AVTERM de Avocet Systems Inc.. Cabe mencionar que todos los datos que procese este emulador de terminal tendrán formato ASCII, por lo que será imperioso convertirlos a hexadecimal cuando éstos ingresen a nuestro sistema, y viceversa cuando sean enviados a la PC.

Para la confección del programa se utilizó el editor de texto de Quick Basic. El programa se subdividió en módulos para facilitar la programación y su posterior depuración. Estos módulos fueron confeccionados de acuerdo a la aplicación particular que tendrían dentro del programa. En base a ello se podrán encontrar módulos que por ejemplo atiendan a la programación del SIO, módulos que permitan la conversión de datos ASCII a hexadecimal y viceversa, etc. Los módulos fueron posteriormente ensamblados individualmente para luego ser linkeados con el fin de obtener el programa final.

6.1.2. Conceptos Generales

6.1.2.1.Comandos

Una vez arrancado el sistema, éste espera que se pulse cualquier tecla para comenzar la ejecución del programa. Esta condición se hace imperiosa usarla para sincronizar el emulador de terminal con el programa. No se envía cartel indicador porque éste se vería solamente en muy pocas ocasiones, ya que para poder correr el emulador de terminal la placa debe encontrarse en funcionamiento, y dada la alta velocidad de operación de la misma, no daría tiempo al usuario para ejecutar el emulador y posteriormente ver el cartel indicador. Luego de la pulsación de una tecla, se muestra una presentación y luego un prompt consistente en la palabra COMANDO. A partir de este momento el sistema se encuentra en un estado de espera hasta que se ingrese un comando. Una vez recibido el comando, si éste es correcto (se analiza el primer carácter recibido), el sistema pasa a analizarlo carácter por carácter de atrás para adelante, por lo que si se ingresó el comando M1234 se analizará primero el cuatro, luego el tres y así siguiendo hasta la M. Este método de análisis se emplea en todos los comandos.

6.1.2.2.Etiquetas

Dentro de cada módulo se emplean etiquetas internas llamadas LOOP seguidas por un número; éstas etiquetas indican bucles repetitivos dentro de la estructura del programa.

También se ha empleado etiquetas para el uso de constantes, como asimismo de variables. Se ha intentado dar un nombre obvio a cada una de ellas.

6.1.2.3.Mensajes

Los mensajes, tanto de indicación de error como de algún caso en particular, se realiza a través de interrupciones, detectando un carácter especial dentro de la tira para finalizar la transmisión.

6.1.2.4.Banderas

Las banderas (posiciones de memoria) son usadas para registrar y trasladar de una parte a otra del programa distintos acontecimientos que ocurren durante la ejecución del programa.

6.1.2.5.Registros

Este es el detalle de los registros más importantes y su función en el programa.

- RESULTADO: contiene el resultado de la conversión de ASCII a hexadecimal. Su contenido proviene de la subrutina CONVERSIÓN.
- JUMP_1: contiene el resultado de la conversión de ASCII a hexadecimal. La diferencia con el anterior no es muy radical, pero se emplea este esquema ya que con este registro se ataca directamente a la ejecución de la instrucción, previniendo de esta forma posibles acarreos de errores y su costosa localización en la puesta a punto del sistema. Su contenido proviene de las subrutinas ANALISIS y ANALISIS_2.
- POSICION_DE_MEMORIA_PUENTE: se utiliza cuando no existe una instrucción específica dentro de la CPU para realizar intercambio de valores de 16 bits. Por ejemplo la instrucción LD HL,IX no existe, por lo que se debe utilizar un registro que haga de puente entre ellos.

6.1.2.6.Constantes

Este es el detalle de las constantes más importantes.

- FIN_DE_MEMORIA: indica cuál es la última posición de memoria disponible del sistema.
- PRINCIPIO_DE_TABLA: indica cuál es la posición de memoria inicial de almacenamiento de caracteres recibidos desde el teclado.
- EQUIVALENTE: indica cuál es la posición de memoria inicial donde se encuentra la tabla de equivalencias entre ASCII y hexadecimal.
- PRINCIPIO_DE_MEMORIA: indica cuál es la posición inicial de la memoria asignada al usuario.
- TABLA_ALMACENAMIENTO_ARCHIVO: indica cuál es la posición de memoria inicial donde se irá almacenando temporalmente el archivo que se ingrese.

6.1.3. Descripción modular

A continuación se pretende dar una explicación pormenorizada de lo que cada módulo realiza dentro del programa, tratando de ser lo más explícitos posible pero sin ahondar en muchos detalles técnicos de confección que pueden ser perfectamente extraíbles del programa mismo.

6.1.3.1.Programación del canal de acceso serie

6.1.3.1.1.Programación del CTC

Se usa un CTC para la programación de la velocidad de transmisión y recepción. El CTC se lo programa para tener una velocidad de 9600 baudios.

6.1.3.1.2.Programación del SIO

El SIO se utiliza para generar todo el acceso serie a y desde la placa. El módulo PROG-SIO.src contiene las distintas opciones de uso del SIO en la administración de la placa, ellas son:

•programación general del SIO. Esta se lleva a cabo al inicio del programa completo y sirve para dar una inicialización al mismo. (Subprograma INICIALIZACION_SIO)

•programación particular del SIO. Está compuesta por varias componentes de acuerdo a la operación a realizar:

- •RESETEAR_SIO
- •FIJAR_VECTOR_INTERRUPCION_SIO
- •FIJAR_CONDICIONES_DE_OPERACION_SIO: define que se usará paridad par, dos bits de stop y 7 bits de datos.
- •HABILITACION_DE_INTERRUPCION_POR_Tx_Y_Rx: permite que ocurran interrupciones cada vez que el buffer de transmisión quede vacío, como también cada vez que se reciba un carácter.
- •DESHABILITACION_DE_INTERRUPCION_POR_Tx_Y_Rx: no permite que ocurran interrupciones cada vez que el buffer de transmisión quede vacío, como también cada vez que se reciba un carácter.
- •HABILITACION_DE_INTERRUPCION_POR_Tx: permite que ocurran interrupciones cada vez que el buffer de transmisión quede vacío, pero no cuando se reciba un carácter.
- •HABILITACION_DE_INTERRUPCION_POR_Rx: permite que ocurran interrupciones cada vez que se reciba un carácter, perro no cuando el buffer de transmisión quede vacío.
- •HABILITACION_DE_Tx
- •HABILITACION_DE_Rx
- •DESHABILITACION_DE_Tx
- •DESHABILITACION_DE_Rx
- •RESETEAR_ERRORES_SIO: permite resetear los registros indicadores que un error ha ocurrido durante la transmisión.
- •RESETEAR_INTERRUPCIONES_POR_Tx_PENDIENTES: esta subrutina se invoca cada vez que se finaliza de enviar una cadena de caracteres a través de interrupciones para prevenir posibles anormalidades, surgidas por interrupciones, en el funcionamiento del sistema.
- •HABILITACION_RTS: esta subrutina se invoca para permitir la habilitación de la pata RTS del integrado, indicando de este modo que el sistema se encuentra apto para recibir datos.
- •DESHABILITACION_RTS: esta subrutina se invoca para deshabilitar la pata RTS del integrado, indicando de este modo que el sistema no se encuentra apto para recibir datos.

6.1.3.2.Recepción de los comandos

Cada vez que aparece el prompt COMANDO el sistema se encuentra a la espera de que se le ingresen datos desde el teclado de la PC. Estos datos son almacenados en la memoria a partir de la posición PRINCIPIO_DE_TABLA en forma ascendente y consecutiva hasta detectar que se ha pulsado la tecla <Enter> (se usa el registro CARACTER_FINAL para almacenar temporalmente la última tecla pulsada y comprobar la condición antes mencionada). La posición de memoria donde se encuentra el último carácter del comando se almacena en un registro FINAL_DEL_COMANDO. A partir de esta posición se comienza el análisis del comando, o sea de atrás para adelante, previamente habiendo detectado que la primera letra del comando sea un indicador de la sintaxis convenida para el ingreso al sistema a través de sus instrucciones. Por ej. si se tipea A1234<Enter> el sistema comprueba que la primera letra del comando no corresponde a ninguna instrucción por lo que se envía un mensaje de error de sintaxis. Si se hubiera tipeado E1234<Enter> el sistema comprobaría que la primera letra corresponde a la instrucción Ejecutar por lo que prosigue el análisis. En este último ejemplo la letra E se encontraría en la posición de memoria PRINCIPIO_DE_TABLA mientras que en la posición de memoria FINAL_DEL_COMANDO se encontraría la dirección donde se encuentra almacenado el cuatro. La rutina de recepción de las teclas pulsadas se lleva a cabo en RECEPCION_DE_CARACTERES que se encuentra en el módulo R-E-CAR.src. Dentro de este

módulo se encuentran rutinas para el tratamiento de eventuales errores de paridad, de overrun y para gestionar el accionar de la tecla de retroceso (tener en cuenta que cuando se pulsa la tecla de retroceso el emulador de terminal sólo envía el ASCII, debiéndose encargar el sistema de su accionar).

6.1.3.3.Análisis de los comandos

Los comandos se analizan de atrás para adelante y además se debe convertir los datos ingresados con formato ASCII a hexadecimal.

6.1.3.3.1.Conversión de ASCII a hexadecimal

Esta conversión se explicará a través de un ejemplo.

Supongamos que se ingresa el comando M29, esto quiere decir que se desea ver la memoria desde la posición 29 hex. hasta la 129 hex. Si viéramos lo que ha ingresado a la memoria veríamos:

posición de memoria:	PRINCIPIO_DE_TABLA	ASCII (M)
	PRINCIPIO_DE_TABLA+1	ASCII (2)
	PRINCIPIO_DE_TABLA+2	ASCII (9)

Obviamente esto no es utilizable en primera instancia, por lo que se debería convertir los ASCIIs de la posición de memoria deseada en el valor que usaría el sistema. La forma de lograr esto es usando la propiedad de que los ASCIIs utilizados se encuentran codificados en forma consecutiva. En el ejemplo anterior el ASCII(9) es 39, por lo que si le restamos el valor 30 nos daría por resultado lo esperado, el valor 9; o bien se puede usar el valor ASCII como puntero para ingresar a un tabla de equivalencias (este es el método que se utiliza), donde la posición de memoria inicial de esa tabla es la posición de memoria EQUIVALENTE. A continuación se procede a hacer lo mismo con el ASCII(2), pero al resultado se lo debe desplazar un nibble hacia la izquierda para que luego sumado con el resultado obtenido previamente de la conversión del ASCII(9) se obtenga el valor 29, que en definitiva es lo que se utiliza en el programa. Este formato de conversión es el que se utiliza en todos los análisis de comandos y es llevado a cabo por la subrutina CONVERSIÓN.

6.1.3.3.2.Análisis de instrucciones más complejas

En el ejemplo del punto 3.1 se analizó una instrucción de un sólo parámetro. En el caso de las funciones con más de un parámetro el análisis se lleva a cabo, salvo por ligeras modificaciones, básicamente del mismo modo.

La multiparametrización de las instrucciones se implementa de forma que los parámetros se encuentren uno a continuación del otro separados por "comas". Por ejemplo una instrucción con tres parámetros se escribe de la siguiente forma: Lxxxx,yyyy,zz

donde xxxx
yyyy
zz son los parámetros.

El análisis consiste en ir haciendo la conversión de los ASCIIs en forma ordenada e ir almacenando los resultados de esas conversiones en los registros correspondientes. Las comas se utilizan para determinar en el análisis hasta donde llegar con el análisis en particular de un parámetro para pasar a analizar el siguiente. Hay una subrutina que se emplea en el análisis de las instrucciones multiparametrizadas: ANALISIS_2. Esta subrutina permite analizar parámetros de 2 y de 4 caracteres (según el valor del registro D de la CPU) y detectar una condición de fin de parámetro (según el contenido del registro C de la CPU) para finalizar el análisis.

6.1.3.3.3.Errores

Los comandos están sujetos a errores de tipo humano. Estos errores si se detectan en el momento del tipeado del comando se pueden subsanar a través de la tecla de retroceso, pero en caso contrario es el sistema el que debe detectarlos.

Los distintos tipos de errores pueden ser:

- errores de sintaxis: se producen cuando no se escribe correctamente el comando.
- errores en la asignación de parámetros: se producen cuando a pesar de haberse tipeado en forma correcta el comando, sus parámetros no se encuentran dentro de los límites impuestos por los diseñadores del sistema.
- error en el formato de archivo a ingresar a la memoria.

6.1.3.4.Análisis de los módulos

A continuación se expondrá la forma de funcionamiento de cada módulo, los parámetros para su acceso, los registros que utiliza y los datos de salida.

6.1.3.4.1.INICIO.SRC

Este módulo se encuentra en la posición de memoria cero de la ROM por lo que será el programa que se ejecutará siempre que se encienda el DSK o se lo resetee.

Contiene las inicializaciones principales del sistema:

- ubicación del Stack Pointer.
- funcionamiento CTC.
- funcionamiento SIO.

y tiene un espacio reservado de memoria hasta la posición 100 Hex para confeccionar una tabla de punteros para acceder a rutinas de interrupción.

6.1.3.4.2.RECEPCIO.SRC

Este módulo atiende la recepción de comandos desde el teclado de la PC, verifica que se trate de una instrucción utilizada por el sistema y si esa verificación resulta correcta bifurca a la rutina que atienda dicha instrucción.

El registro BANDERA_7 se usa para indicar si el comando es VERIFICAR o TRASLADAR, conteniendo en cada caso el valor CERO o UNO respectivamente.

6.1.3.4.3.EJECUCIO.SRC

Este módulo ejecuta la instrucción Ejecutar. Utiliza el contenido del registro JUMP_1 como indicador de la posición inicial de ejecución.

La sintaxis de la instrucción es: Exxxx

donde xxxx es la posición inicial de memoria de ejecución.

6.1.3.4.4.MOSTRAR.SRC

Este módulo ejecuta la instrucción Mostrar.

La sintaxis de la instrucción es: Mxxxx

donde xxxx es la posición inicial de memoria que se desea visualizar. En caso de omisión del parámetro se muestra a partir de la posición de memoria siguiente a la última que se mostró en una ejecución anterior del comando Mostrar.

El sistema cuando recibe esta instrucción envía a la pantalla el contenido de 100 hex. posiciones de memoria a partir de la posición de memoria elegida en el parámetro. Estas 100 hex. posiciones de memoria se muestran de la siguiente manera:

- Primero se envía la posición de memoria elegida seguida de un espacio, luego se envía el contenido de las F posiciones de memoria siguientes separadas por un espacio cada una y a continuación un retorno de carro y un line feed para acceder a la línea siguiente de la pantalla. Se incrementa el valor de la posición de memoria en 10 Hex. y se repite el procedimiento anterior hasta completar la muestra de 100 Hex. posiciones de memoria.
- Se almacena la última posición de memoria mostrada para que en caso de recibir el comando sin parámetros se muestre a partir de la posición de memoria siguiente a la última que se mostró.

Registros

- BANDERA_MOSTRAR indica si se ha especificado la posición de memoria cero como parámetro del comando. Esto lo hace si su contenido es el valor UNO.
- VALOR_ANTERIOR_DE_MUESTRA contiene la última posición de memoria mostrada.
- VALOR_PARCIAL_DE_IX se utiliza para almacenar temporalmente el contenido del registro IX.
- TABLA_DE_CONVERSION es la posición de memoria inicial de una tabla de conversión de hexadecimal a ASCII.

Hay que tener presente para el análisis de esta instrucción que el sistema muestra transparencia, en el uso de memoria, al usuario, con esto queremos decir que el usuario verá que la memoria asignada para su uso comienza a partir de lo posición cero, cuando en realidad la posición inicial de memoria asignada al usuario comienza en PRINCIPIO_DE_MEMORIA.

6.1.3.4.5.R-E-CAR.SRC

Este módulo atiende la recepción y el envío de caracteres por el puerto serie a través del uso de interrupciones. También se encuentra la rutina para el envío de caracteres por el puerto serie por el uso de polling.

6.1.3.4.5.1.Recepción

Cuando se recibe un carácter el sistema interrumpe y se accede a esta rutina, la cual ingresa el dato para su almacenamiento y además lo rebota para visualizarlo en la pantalla, con esto logramos ver lo que estamos escribiendo. Si la tecla pulsada no es la tecla <Enter> se procede a analizar si la tecla pulsada fue la de retroceso, que en caso de ser afirmativo, se deberá atender a la función que debe realizar dicha tecla. Asimismo se cuenta la cantidad de caracteres que se han ingresado desde el teclado y si esa cantidad sobrepasa el valor CANTIDAD_TOTAL_DE_CARACTERES se envía un mensaje de error.

Esta rutina almacena los valores recibidos en la posición de memoria indicada por el registro IX en forma ascendente, o sea que una vez almacenado un valor se incrementa el valor del registro IX para que apunte a otra posición de memoria. La rutina que ejecuta esta opción es RECEPCION_DE_CARACTERES.

6.1.3.4.5.2.Envío

1.Por interrupción: se utiliza para enviar tiras de caracteres. Esto se consigue colocando en el registro IX la posición de memoria donde se encuentra el primer carácter a enviar y luego invocar la subrutina VISUALIZACION. El envío de la tira concluye cuando se detecta el carácter de fin de tira (ASCII 40). La rutina que ejecuta esta opción es ENVIO_DE_CARACTERES.

2.Por polling: se utiliza para enviar caracteres individualmente. En este método se envía un carácter y se espera a que el buffer quede vacío antes de permitir el retorno de la subrutina. La subrutina que ejecuta esta opción es ENVIO y se debe cargar en el acumulador el valor a enviar antes de invocarla.

6.1.3.4.6.LLENAR.SRC

Este módulo ejecuta la instrucción de llenado de memoria. La sintaxis de la instrucción es la siguiente: Lxxxx,yyyy,zz donde xxxx indica la posición inicial de memoria a llenar. Se utiliza el registro POSICION_INICIAL_DE_MEMORIA.

yyyy indica la posición final de memoria a llenar. Se utiliza el registro POSICION_FINAL_DE_MEMORIA.

zz es el valor con el que se quiere llenar la memoria. Se utiliza el registro CONTENIDO_DE_MEMORIA.

El registro POSICION_DE_MEMORIA se utiliza indica la posición actual de llenado de memoria cuando se realiza el llenado de la memoria.

6.1.3.4.7.VER-TRAS.SRC

Este modulo ejecuta la instrucción de verificación y de traslado de dos bloques de memoria. La sintaxis de la instrucción es la siguiente: Vxxxx,yyyy,zzzz o Txxxx,yyyy,zzzz

donde xxxx indica la posición inicial de memoria a comparar o trasladar. Se utiliza el registro COMIENZO_DE_MEMORIA_1.

yyyy indica la posición final de memoria a comparar o trasladar. Se utiliza el registro FIN_DE_MEMORIA_1.

zzzz indica la posición inicial de memoria del segundo bloque para comparar o trasladar. Se utiliza el registro COMIENZO_DE_MEMORIA_2.

El registro BANDERA_7 se usa para determinar si la instrucción es de verificar o de trasladar.

6.1.3.4.8.ANALISIS.SRC

Este módulo contiene las subrutinas usadas para analizar los parámetros de los comandos.

- Subrutina Analisis: analiza y convierte el parámetro de una instrucción de parámetro único. Esto lo realiza de atrás para adelante y utiliza como detector de fin de parámetro la letra y la posición de memoria donde se encuentra el identificador de comando. El resultado lo almacena en la posición de memoria JUMP_1.
- Subrutina Analisis_2: analiza y convierte los parámetros de una instrucción multiparametrizada. Esto lo realiza de atrás para adelante y utiliza como detector de fin de parámetro el contenido del registro C. Se utiliza el registro D para indicar si se quiere el análisis sobre 2 caracteres o 4 caracteres como máximo. El resultado lo almacena en la posición de memoria JUMP_1.
- Subrutina Conversión: convierte el contenido de memoria indicado por el registro IX en un valor hexadecimal. El valor hexadecimal lo entrega en el registro RESULTADO.

6.1.3.4.9.VISUALIZ.SRC

Este módulo se utiliza para el envío de tiras de caracteres a través de interrupciones. Envía por el puerto serie el contenido de la posición de memoria apuntada por el registro IX y espera que ocurra la interrupción. Cuando ésta ocurre un bucle recursivo se encarga de transmitir toda la tira hasta que detecta el carácter de fin de tira (ASCII 40).

6.1.3.4.10.PUERTO.SRC

Este módulo ejecuta la instrucción de lectura y de escritura de los puertos.

6.1.3.4.10.1.Lectura

La sintaxis para la lectura de los puertos es la siguiente:

PExx donde xx es el número de puerto (se utiliza el registro VALOR_DEL_PUERTO_DE_ENTRADA). Ante este tipo de instrucción el sistema devuelve el contenido del puerto deseado colocando en la pantalla el número de puerto deseado seguido de dos puntos y un espacio, para luego mostrar el contenido de dicho puerto.

6.1.3.4.10.2.Escritura

La sintaxis para la escritura de los puertos es la siguiente: PSxx,yy donde: xx es el número de puerto deseado. Se utiliza el registro VALOR_DEL_PUERTO_DE_SALIDA.

yy es el contenido con el que se quiere llenar el puerto deseado. Se utiliza el registro CONTENIDO_DEL_PUERTO.

El sistema no permitirá la escritura del puerto que utiliza el CTC para establecer la velocidad de transmisión y de recepción del canal serie, como tampoco permitirá que se quiera escribir el WR0, WR3..WR7 del canal que se utiliza para la comunicación serie del SIO ya que será considerado como una posible alteración del sistema. Tampoco se permitirá dos escrituras consecutivas al SIO en el canal que utiliza para la comunicación serie.

6.1.3.4.11.ING-ARCH.SRC

Este módulo se utiliza para ingresar archivos con formato .HEX

El procedimiento para el ingreso de archivos es el siguiente:

- Se utiliza el comando I<Enter>.
- El sistema muestra un cartel indicando el procedimiento a realizar para utilizar una de las funciones del emulador de terminal, que es la de enviar archivos por el puerto de acceso serie. Este envío lo realiza enviando carácter por carácter del archivo, por lo que si se analiza un archivo .HEX se verá el protocolo usado en su estructuración.
- Se tipea el nombre del archivo y se pulsa la tecla <Enter>.
- Se debe pulsar la tecla S para poder salir de la subrutina.

Los archivos con formato .HEX tienen el siguiente formato en cada fila: :xxyyyyzzaaaaaaaaakk

donde: los dos puntos son un identificador de fila.

xx indica la cantidad de datos que hay en la fila.

yyyy indica la posición de memoria donde se debe comenzar a depositar los datos de la fila.

zz indica el tipo de datos.

aa son los datos

kk es la suma chequeo realizada sumando todos los bytes ingresados y luego realizandole el complemento a dos al total de la suma.

Se debe tener la precaución de que al ensamblar los programas, el usuario los haya organizado previamente a partir de la posición PRINCIPIO_DE_MEMORIA como mínimo, porque sino se enviará un cartel de error ya que la memoria del usuario comienza a partir de esta posición.

6.1.3.4.12.PROG-SIO.SRC

Este módulo contiene todas las programaciones que se realizan en el SIO. Están estructuradas en forma de subrutinas para permitir su fácil acceso. Las distintas subrutinas fueron explicadas anteriormente.

6.1.3.4.13.DATOS.SRC

Este módulo contiene la definición de todas las etiquetas que tienen un valor significativo. Dentro de él están definidas las constantes, las variables, los carteles indicadores y las tablas de equivalencias.

6.1.3.4.14.ERROR.SRC

Este módulo contiene los distintos mensajes de error utilizados por el sistema.

Una vez enviado el mensaje de error, el sistema vuelve casi al principio del programa general. Este lugar es donde se muestra el prompt COMANDO, indicando que se debe ingresar nuevamente un comando.

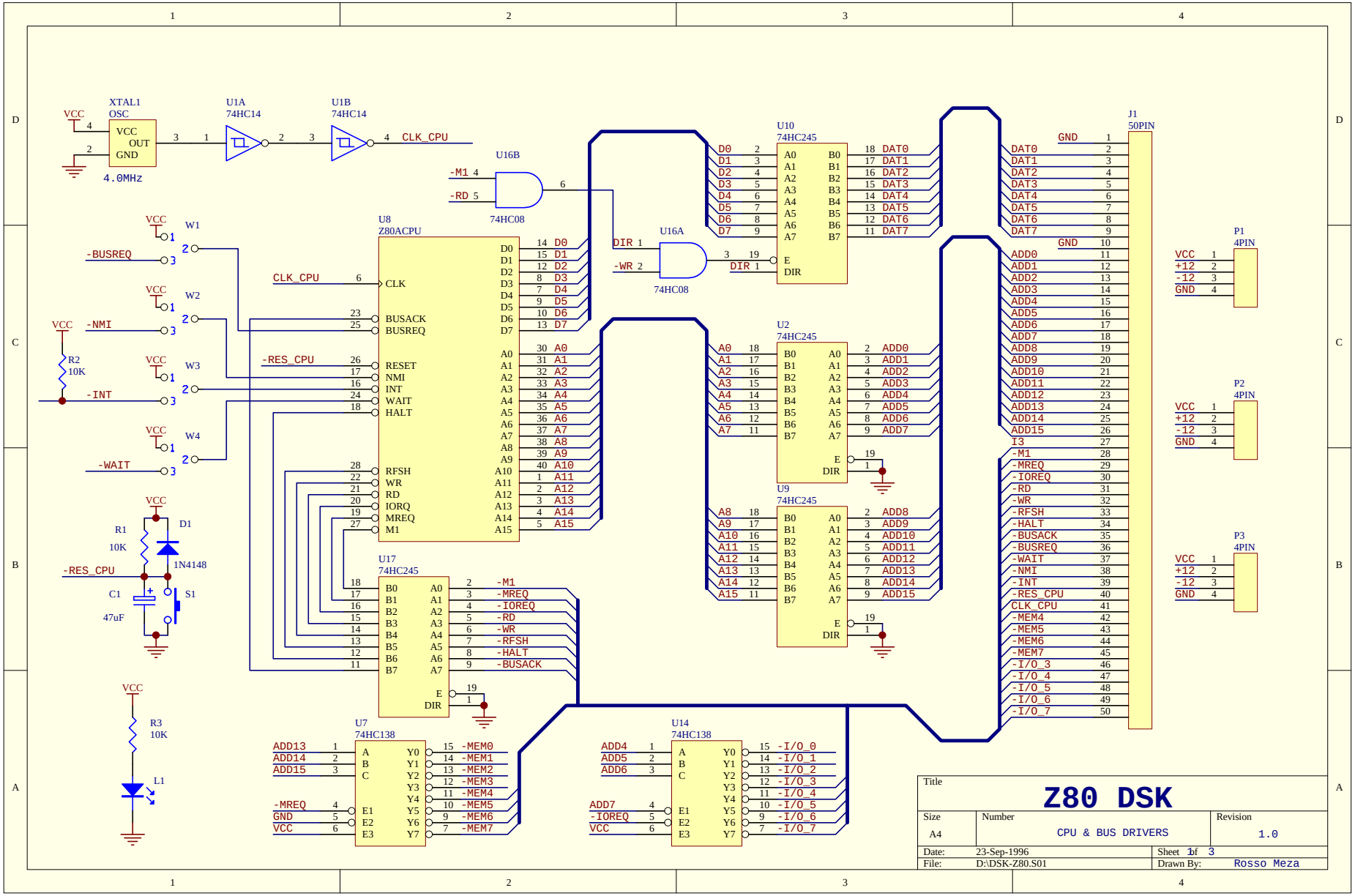
6.1.4. Set de Instrucciones

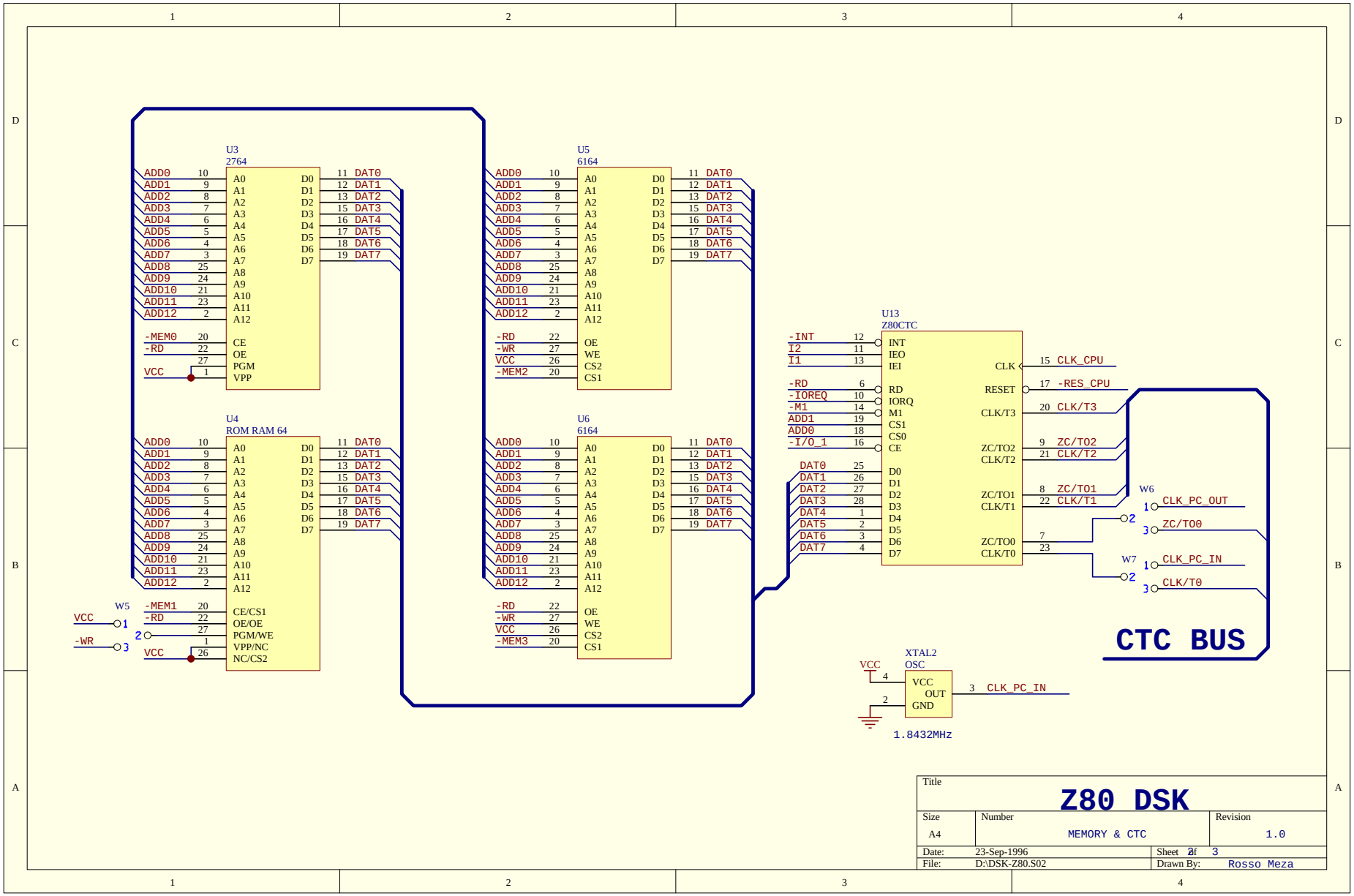
Instrucción	Operación que realiza	Parámetros que utiliza	Sintaxis
Ejecutar	Transfiere el valor del parámetro al contador de programa	Posición de memoria de comienzo de ejecución. (0-4FFF)	Exxxx
Mostrar	Muestra el contenido de	Posición inicial de	Mxxxx

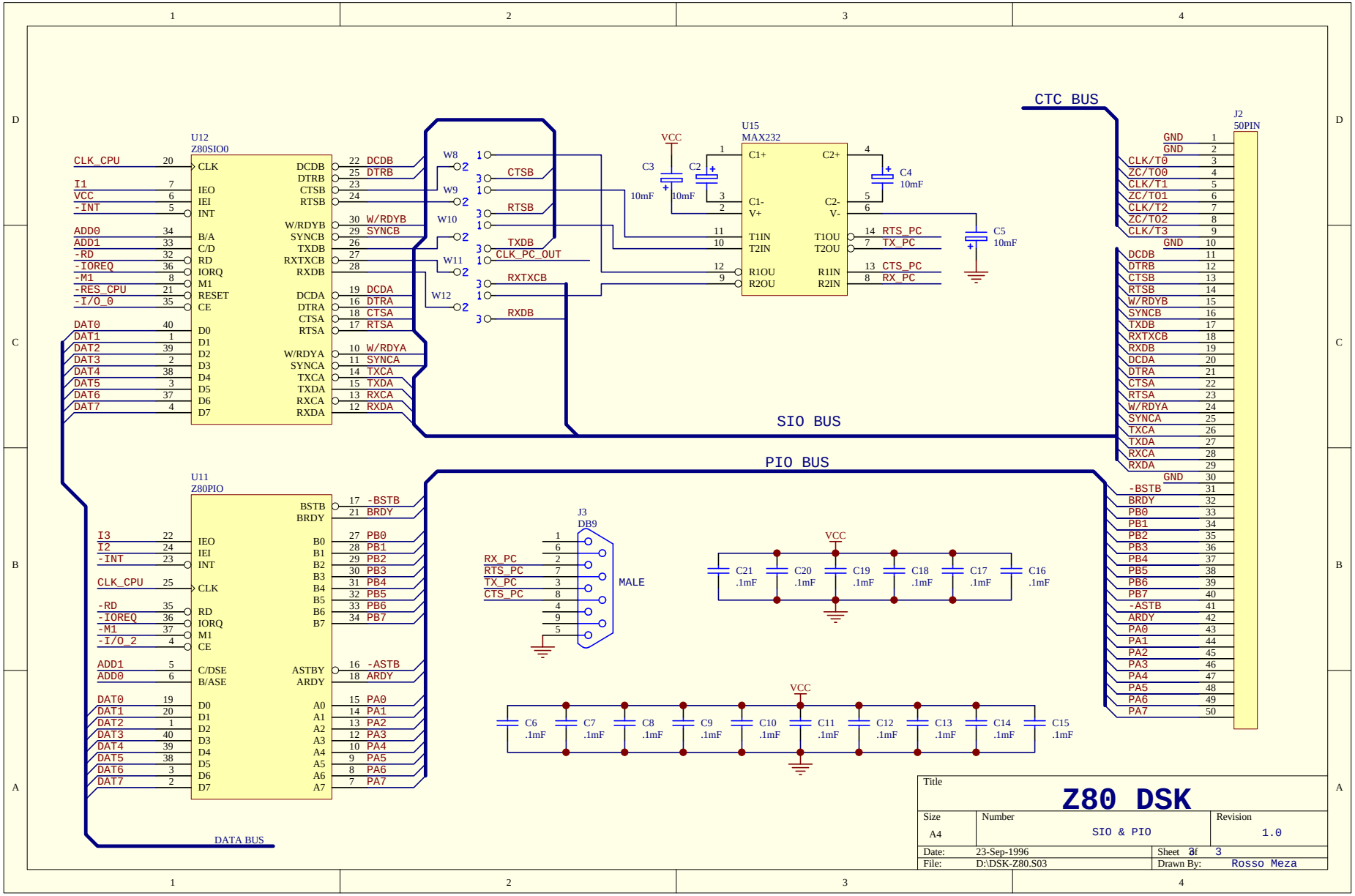
	100 posiciones de memoria a partir del parámetro	muestreo (0-4FFF)	
Llenar	Llena la memoria desde un valor inicial hasta un valor final con un valor deseado	Valor Inicial Valor Final Contenido (VI<VF) (0-4FFF)	Lxxxx,yyyy,zz donde xxxx=VI yyyy=VF zz=Contenido
Verificar	Comprueba si dos bloques de memoria son iguales	Valor Inicial Valor Final Nueva Posición (VI<VF) (0-4FFF)	Vxxxx,yyyy,zzzz donde xxxx=VI yyyy=VF zzzz=NP
Trasladar	Mueve un bloque de memoria a otra posición	Valor Inicial Valor Final Nueva Posición (VI<VF) (0-4FFF)	Txxxx,yyyy,zzzz donde xxxx=VI yyyy=VF zzzz=NP
Ingresar Archivo	Ingresa un archivo con formato .HEX		I
Puerto Salida	Envia al puerto seleccionado el valor deseado	Puerto deseado Valor deseado (0-FF)	PSxx,yy donde xx=PD yy=VD
Puerto Entrada	Devuelve el contenido del puerto seleccionado	Puerto deseado (0-FF)	PExx donde xx=Puerto deseado

7. Apéndice C: Circuitos Esquemáticos y PCB Layout

Los circuitos que a continuación se detallan son los siguientes:


XTAL1
OSC
4.0MHz
U1A
74HC14
U1B
74HC14
CLK_CPU
U16B
74HC08
U16A
74HC08
U8
Z80ACPU
CLK
BUSACK
BUSREQ
RESET
NMI
INT
WAIT
HALT
RFSH
WR
RD
IORQ
MREQ
M1
D0
D1
D2
D3
D4
D5
D6
D7
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
U10
74HC245
U2
74HC245
U9
74HC245
U17
74HC245
U7
74HC138
U14
74HC138
J1
50PIN
P1
4PIN
P2
4PIN
P3
4PIN
W1
W2
W3
W4
-BUSREQ
-NMI
-INT
-WAIT
-RES_CPU
R1
10K
R2
10K
R3
10K
D1
1N4148
C1
47uF
S1
L1
VCC
GND
DAT0
DAT1
DAT2
DAT3
DAT4
DAT5
DAT6
DAT7
ADD0
ADD1
ADD2
ADD3
ADD4
ADD5
ADD6
ADD7
ADD8
ADD9
ADD10
ADD11
ADD12
ADD13
ADD14
ADD15
-M1
-MREQ
-IOREQ
-RD
-WR
-RFSH
-HALT
-BUSACK
-WAIT
-NMI
-INT
-RES_CPU
CLK_CPU
-MEM0
-MEM1
-MEM2
-MEM3
-MEM4
-MEM5
-MEM6
-MEM7
-I/O_0
-I/O_1
-I/O_2
-I/O_3
-I/O_4
-I/O_5
-I/O_6
-I/O_7
Title
Z80 DSK
Size
A4
Number
CPU & BUS DRIVERS
Revision
1.0
Date:
23-Sep-1996
Sheet 1 of 3
File:
D:\DSK-Z80.S01
Drawn By:
Rosso Meza


U3
2764
U4
ROM RAM 64
U5
6164
U6
6164
U13
Z80CTC
XTAL2
OSC
1.8432MHz
W5
W6
W7
CTC BUS
Title
Z80 DSK
Size
A4
Number
MEMORY & CTC
Revision
1.0
Date:
23-Sep-1996
Sheet 2 of 3
File:
D:\DSK-Z80.S02
Drawn By:
Rosso Meza


CTC BUS
SIO BUS
PIO BUS
DATA BUS
U12
Z80SIO0
U11
Z80PIO
U15
MAX232
J2
50PIN
J3
DB9
MALE
VCC
W8
W9
W10
W11
W12
CTSB
RTSB
TXDB
CLK_PC_OUT
RXTXCB
RXDB
RTS_PC
TX_PC
CTS_PC
RX_PC
C1 .1mF
C2 .1mF
C3 .1mF
C4 .1mF
C5 .1mF
C6 .1mF
C7 .1mF
C8 .1mF
C9 .1mF
C10 .1mF
C11 .1mF
C12 .1mF
C13 .1mF
C14 .1mF
C15 .1mF
C16 .1mF
C17 .1mF
C18 .1mF
C19 .1mF
C20 .1mF
C21 .1mF
Title
Z80 DSK
Size
A4
Number
SIO & PIO
Revision
1.0
Date:
23-Sep-1996
Sheet 3 of 3
File:
D:\DSK-Z80.S03
Drawn By:
Rosso Meza