ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

"PROGRAMA DIDACTICO PARA LA VISUALIZACION GRAFICA DE LA EJECUCION DE LAS INSTRUCCIONES DEL MICROCONTROLADOR MCS-52"

TESIS PREVIA A LA OBTENCION DEL TITULO DE INGENIERO EN ELECTRONICA Y TELECOMUNICACIONES.

HAMILTON SHENS MESIAS LOPEZ

QUITO, NOVIEMBRE, 1996

Certifico que el presente trabajo ha sido realizado por el Sr. Hamilton Mesías López bajo mi dirección.

Ing. Jaime Velarde

Agradecimientos

Al Ingeniero Jaime Velarde por su valioso tiempo y ayuda dedicados durante la elaboración de esta tesis

A todos mis amigos que de una u otra manera me han brindado su ayuda.

INDICE

CAPITULO 1

INTRODUCCION

1.1 INTRODUCCION GENERAL

En la actualidad la difusión de Software Educativo ha convertido a los computadores personales en una herramienta de enseñanza poderosa, que complementa parcial o totalmente los conocimientos impartidos en un aula. Resulta cómodo aprender en la pantalla del computador, ya que èste repetirá un tema una y otra vez sin cansarse hasta que el usuario lo crea conveniente, a la vez resulta entretenido el aprendizaje ya que se procura que estos programas sean lo mas didácticos posible.

El estudio del Microcontrolador MCS-52 en la Facultad es de suma importancia puesto que es la base para el desarrollo de diferentes aplicaciones,de las cuales se puede ir tomando conciencia en la misma materia de Sistemas Microprocesados donde los ejemplos de aplicación son varios, posteriormente las materias de Sistemas Analógico - Digitales y Comunicación Digital trabajan con módulos didácticos construidos en base al microcontrolador

Por lo dicho el estudiante debe tener sus conocimientos bien claros respecto al microcontrolador y mas aún que cualquier aplicación con èste requiere la elaboración de programas basados en el set de instrucciones. Entonces resulta indispensable que se tenga bien claro que es lo que implica la ejecución de las instrucciones para pasar a la elaboración de los programas.

El VGI MCS-52 (Programa para la visualización gráfica de las instrucciones del Microcontrolador MCS-52) espera ser la ayuda extra a lo enseñado en las aulas respecto al set de instrucciones, el programa ha sido elaborado en Visual Basic 3.0 procurando ser lo más didáctico posible con el fin de contribuir al aprendizaje del manejo del set de instrucciones del MCS-52.

Esta tesis abarca los conocimientos que se han requerido para lograr implementar el VGI MCS-52, así como también los resultados de la elaboración del mismo, por lo que de una manera general se podría describir los contenidos de esta tesis de la siguiente manera:

En el Capítulo 1 a más de esta pequeña introducción seguidamente se intenta explicar la manera en que opera Visual Basic para la elaboración de aplicaciones, lo cual se lo ha hecho en vista de que al menos en nuestro medio la programación con este paquete no es muy difundida, no intenta ser un manual de Visual Basic sino simplemente una orientación respecto a éste.

En el Capítulo 2 se describe de una manera clara lo mas importante respecto al MCS-52 donde se incluyen temas como su arquitectura, memoria y set de instrucciones.

El Capítulo 3 se refiere exclusivamente al camino de solución para la elaboración del VGI MCS-52., al cual se añade un Manual del usuario.

Finalmente el Capítulo 4 ha recogido los resultados y conclusiones que se han ido acumulando a través del tiempo en que se desarrolló esta tesis.

1.2 ACERCA DE VISUAL BASIC

Visual Basic es un lenguaje de programación de alto nivel que permite desarrollar aplicaciones para Windows,lo cual en los actuales momentos es de gran ayuda ya que la mayoría de Software se lo desarrolla para ambiente Windows.

Desde la aparición de Visual Basic versión 1.0 en el año 1991 éste ha ido mejorando y presentando sus nuevas versiones, así la versión 2.0 era más rápido, la versión 3.0 añade una forma sencilla de manejar bases de datos, y en los actuales momentos ya existe la versión 4.0.

1.2.1 REQUERIMIENTOS

Para ejecutar Visual Basic se requiere:

* Un computador 80386 o superior
* 13 MBytes de espacio en el disco duro
* Un ratón (mouse)
* Tarjeta gráfica EGA o superior
* Windows 3.0 o superior
* 4 MBytes de RAM como mínimo

Cumpliendo estas condiciones mínimas se dirá que el trabajar con Visual Basic resultará cómodo

1.2.2 CONCEPTOS

A diferencia de aplicaciones desarrolladas en otros lenguajes de programación que siguen un flujo de programa de arriba hacia abajo, las aplicaciones de Visual Basic ejecutan sus acciones dependiendo de los eventos que se le han indicado que reconozca, dichos eventos pueden ser activados por el usuario o por condiciones ligadas a los controles de la aplicación.

Para desarrollar una aplicación en Visual Basic se deben seguir 3 pasos:

* Crear la interface
* Especificar las propiedades
* Escribir el código

1.2.2.1 CREAR LA INTERFACE

Cuando se ingresa a Visual Basic automáticamente el usuario dispone de un formulario (form), sobre el cual uno puede colocar todos los controles que desee mostrar cuando la aplicación esté corriendo, dichos controles se los escoge de la

ventana de herramientas tomándolos por medio del mouse y colocándolos en el sitio que uno desee, cabe señalar que todo control tendrá varias propiedades que determinarán la apariencia y comportamiento de éstos.

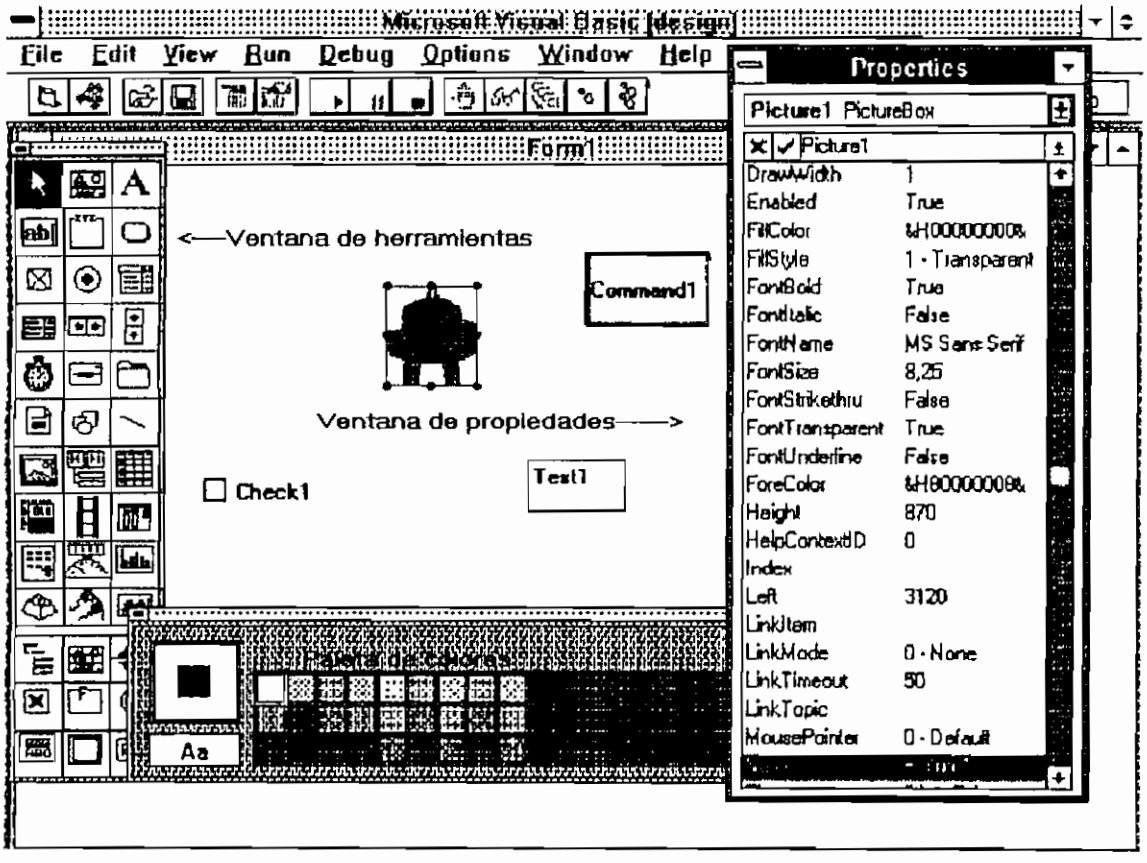


Figura 1.1 Ambiente de trabajo para Visual Basic 3.0

Visual Basic también nos da la posibilidad de crear menús lo cual se lo realiza fácilmente a través de la opción Window - Menu design

Es decir la Interface consiste en establecer todos los elementos que va a ver el usuario

Nota: Una aplicación puede tener mas de un formulario, para lo cual hay que escoger la opción File - Add New

1.2.2.2 ESPECIFICAR LAS PROPIEDADES

Una vez escogidos todos los controles se debe especificar las propiedades de los mismos las cuales determinarán su apariencia y comportamiento a lo largo de la aplicación.

Cabe señalar que si bien estas propiedades generalmente se especifican cuando se está construyendo la aplicación, éstas también pueden ser especificadas al momento en que la aplicación está corriendo.

Las propiedades especifican características tan variadas como tamaño, color, ubicación, etc. y con respecto al comportamiento definen principalmente si el objeto está habilitado para responder a los diferentes eventos que se pueden generar

1.2.2.3 ESCRIBIR EL CODIGO

Una vez que hemos escogido todos los controles y les hemos asignado sus propiedades, queda por definir las acciones que la aplicación ejecutará, lo cual dependerá del código es decir la programación que para Visual Basic sigue mas o menos la línea de Quick Basic

Por ejemplo sentencias como IF then, For next,Select case son también válidas en Visual Basic, pero además disponemos de métodos como Additem, Move,Setfocus que son líneas de comando que realizan una acción determinada.

El código se escribe para cada evento que se quiere que un objeto reconozca, estos eventos pueden ser

* Acciones activadas por el usuario (Por ejemplo haciendo un Click con el ratón sobre un control)

* Acciones activadas por la condición de un control (Por ejemplo Evento Change el cual se activa cuando el contenido de una etiqueta cambia)

* Acciones activadas por el sistema (Por ejemplo cuando una aplicación es cargada se activa el Evento Load del formulario activo)

CAPITULO 2

DESCRIPCION DEL MCS-52

La familia MCS-52 está formada por los microcontroladores de 8 bits que se muestran en la tabla 2.1.

ELEMENTO	TECNOLOGIA	MEMORIA INTERNA DE PROGRAMA	MEMORIA INTERNA DE DATOS
8052	HMOS II	8 K (ROM)	256 BYTES
8032	HMOS II	–	256 BYTES
8752	HMOS	8 K (EPROM)	256 BYTES

Tabla 2.1

Como se puede notar las diferencias de estos dispositivos radican en la memoria de programa, de allí se puede decir que son similares, por lo cual a lo largo de esta tesis se referirá al Microcontrolador MCS-52 que estará abarcando a todos sus dispositivos.

2.1 ARQUITECTURA DEL MCS-52

El Microcontrolador MCS-52 está basado en la arquitectura mostrada en la figura 2.1

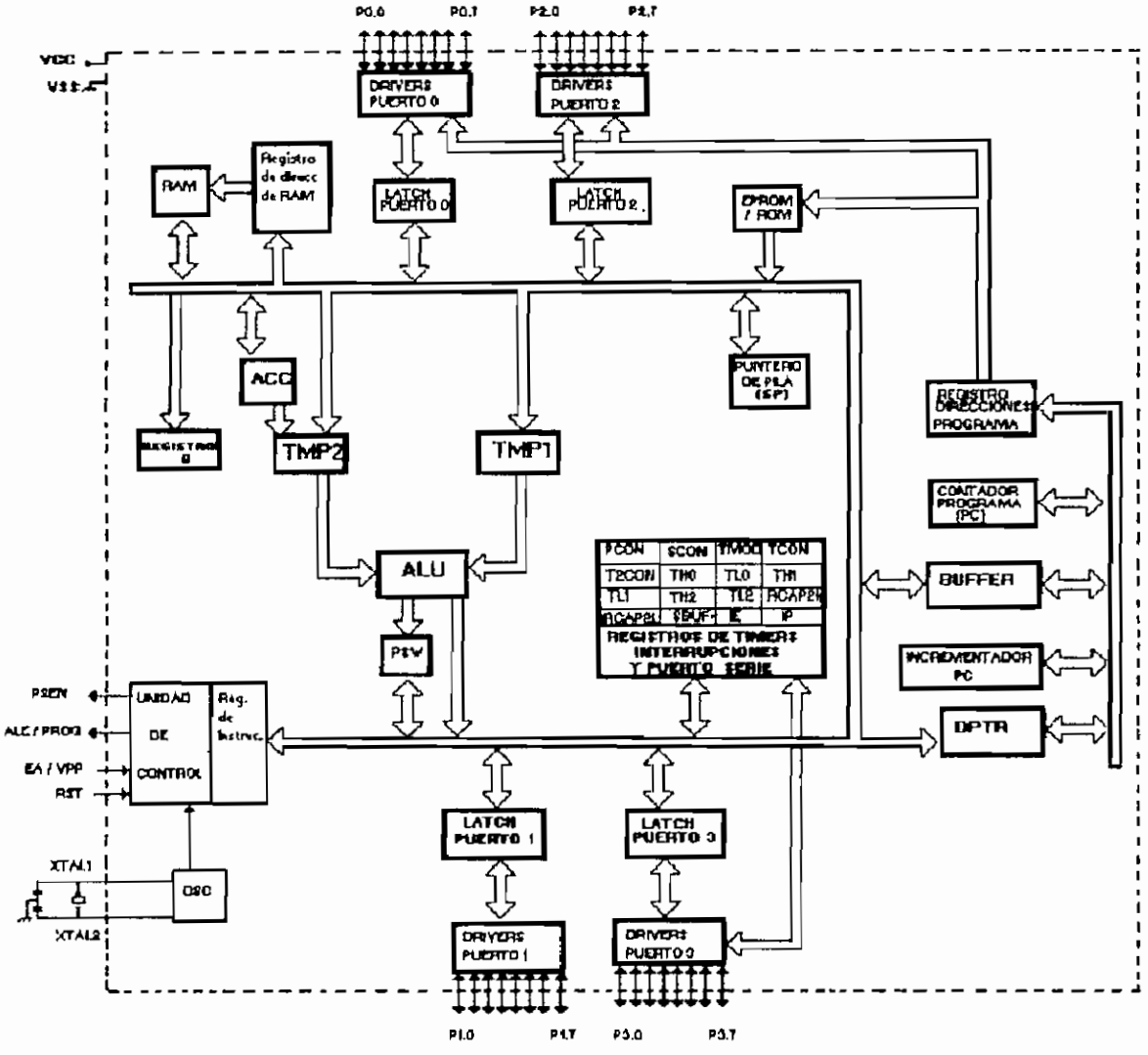


FIGURA 2.1 Arquitectura del MCS-52

Sus principales características se indican a continuación.

* CPU de 8 bits.
* Procesador Booleano.
* 4 Puertos de 8 bits.
* Capacidad de manejo de memoria externa de programa de hasta 64 Kbytes.
* Capacidad de manejo de memoria externa de datos de hasta 64 Kbytes.
* 3 Temporizadores/Contadores de 16 bits.
* 6 fuentes de interrupciones con 2 niveles de prioridad.

* Comunicación asíncrona full-duplex

* Oscilador interno

2.1.1 DESCRIPCION DE TERMINALES

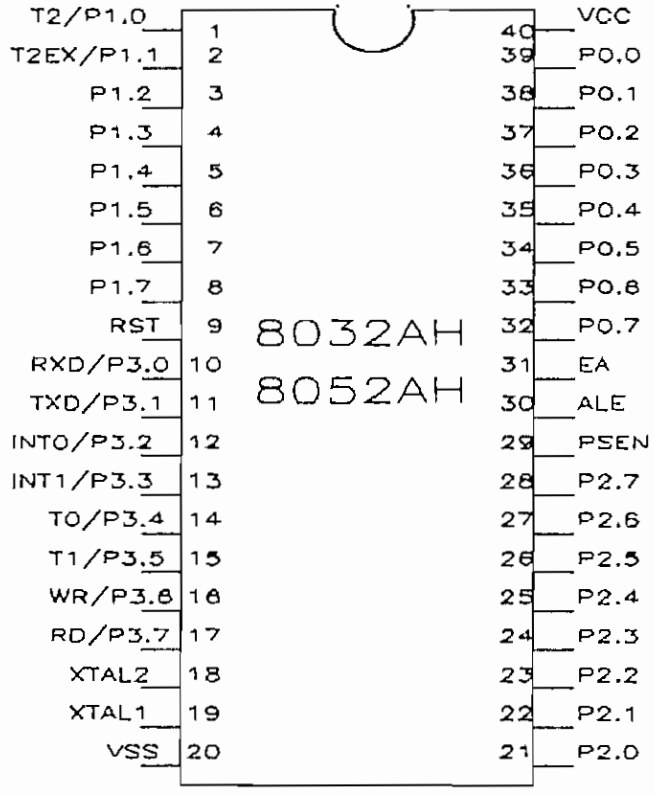


FIGURA 2.2

VCC Voltaje de alimentación (+5 voltios)

VSS Conexión a tierra

P0.0 - P0.7 Terminales del Puerto 0, el cual es un puerto bidireccional de 8 bits con canal abierto.

Entre las funciones de este puerto tenemos :

* Multiplexa en el tiempo la parte baja del bus de direcciones y el bus de datos durante el acceso a memoria externa.

* Recibe los bytes de código durante la programación de la memoria EPROM y emite los bytes de código durante la verificación de la memoria EPROM o ROM.

P1.0 - P1.7 Terminales del Puerto 1, el cual es un puerto bidireccional de 8 bits con resistencia interna de "pull up", entre sus funciones están:

* Recibe la parte baja de direcciones durante la programación y verificación de la memoria EPROM.

* Los bits P1.0 y P1.1 tienen las funciones alternativas de T2 y T2EX respectivamente. T2 es la entrada externa del Timer 2 y T2EX es la entrada a interrupción externa 2.

P2.0 - P2.7 Terminales del Puerto 2, el cual es un puerto bidireccional de 8 bits con resistencia interna de " pull up ", entre sus funciones tenemos :

* Emite la parte alta del bus de direcciones durante el acceso a la memoria externa cuando utilizan 16 bits de dirección.

* Recibe la parte alta de la dirección durante la programación y verificación de la memoria EPROM

P3.0 - P3.7 Terminales del Puerto 3, el cual es un puerto bidireccional de 8 bits con resistencia interna, los terminales de este puerto tienen ciertas funciones alternativas. las cuales se indican en la tabla 2.2.

TERMINAL	FUNCION ALTERNATIVA
P3.0	RXD (Entrada del puerto serie)
P3.1	TXD (Salida del puerto serie)
P3.2	INT0 (Interrupción Externa 0)
P3.3	INT1 (Interrupción Externa 1)
P3.4	T0 (Entrada externa Timer 0)
P3.5	T1 (Entrada externa Timer 1)
P3.6	WR (Permiso de escritura en memoria de datos externa)
P3.7	RD (Permiso de lectura en memoria de datos externa)

Tabla 2.2

RESET Un nivel alto en este terminal por dos ciclos de máquina produce una reiniciación del dispositivo.

ALE/PROG ALE (Address Latch Enable) es un pulso que emite el Microcontrolador para retener el byte bajo del bus de direcciones durante el acceso a la memoria externa. ALE se emite con una frecuencia de 1/6 de la frecuencia de reloj

PROG es el terminal de entrada para los pulsos de programación de la memoria EPROM.

PSEN (Program Store Enable), es la señal para leer de la memoria externa de programa. Esta señal no se activa cuando se está ejecutando el programa de la ROM o EPROM interna.

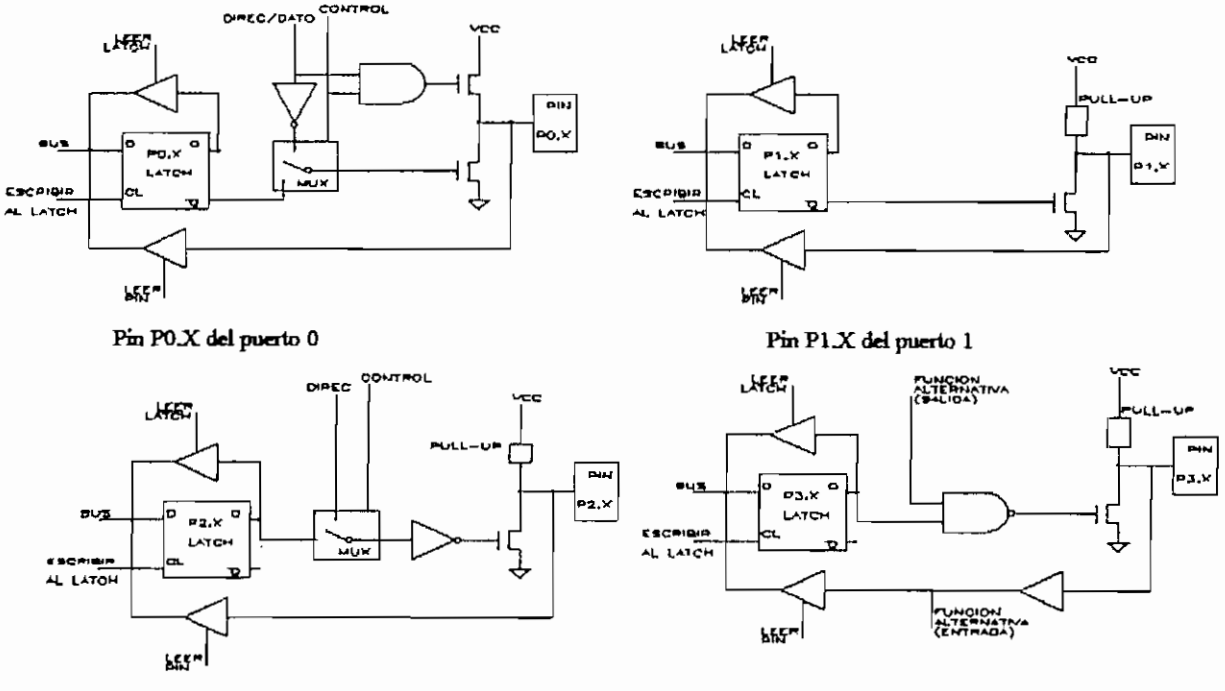
EA/VPP (External Access), cuando EA está a nivel alto se ejecuta solo el programa de la memoria interna, a no ser que el contador de programa exceda de 1FFFH. En cambio si EA es puesto a nivel bajo siempre ejecuta el programa de la memoria externa.

VPP es el terminal para la tensión de programación de la memoria EPROM

XTAL1 y XTAL2 Son la entrada y salida respectivamente de un amplificador inversor, el cual puede ser configurado como oscilador.

2.1.2 ESTRUCTURA DE PUERTOS Y OPERACION

Los 4 puertos del MCS-52 son bidireccionales. Cada terminal está conformado por un latch, un driver de salida y un buffer de entrada. Adicionalmente los Puertos 0 y 2 poseen un conmutador que permite seleccionar ya sea la operación como el bus de direcciones/datos del microcontrolador o la entrada/salida del periférico. Lo dicho se puede representar con el siguiente diagrama funcional



Pin P0.X del puerto 0

Pin P1.X del puerto 1

Pin P2.X del puerto 2

Pin P3.X del puerto 3

FIGURA 2.3 Estructura de Puertos

De este diagrama se puede establecer que:

El valor del latch tomará el valor del bus interno solo cuando la señal Escribir al Latch sea activada, esta señal al igual que leer latch y leer pin serán emitidas por la CPU y dependerá de la instrucción ejecutada

Entre las instrucciones que leen los puertos se debe considerar si leen de los latch o leen de los pines, así se tiene que :

Existen algunas instrucciones que leen y reescriben en el latch, las cuales se indican a continuación.

```
ANL  P1,A
ORL  P2,A
XRL  P3,A
JBC  P1.1,rel
CPL  P3.0
INC  P2
DEC  P2
DJNZ P3,REL
MOV  PX.Y,C
CLR  PX.Y
SETB PX.Y
```

En cambio instrucciones que tienen como operando origen algún byte o bit de cualquier puerto, leen el valor del pin : Ejemplos

```
MOV  A,P1
ADD  A,P3
CJNE A,P2,REL
MOV  C,PX.Y
```

Si el latch de cualquiera de los puertos 1, 2 o 3 contiene un 1L entonces el driver FET estaría desactivado y por lo tanto el nivel del pin es llevado a alto debido a la resistencia interna de "pull up ", pero este nivel puede ser cambiado por alguna fuente externa (poner a Cero), es decir actúa como entrada.

Para el puerto 0 ya que este no tiene resistencia interna el efecto de poner un 1 en el latch sería el que ambos drivers quedarían desactivados y como consecuencia el pin

quedaría flotando, entonces estos pines podrían ser usados como entradas de alta impedancia.

Si los latch contienen un 0, el nivel de los pines sería también cero, en cuyo caso los pines solo podrían actuar como salidas.

2.1.3 ACCESO A MEMORIA EXTERNA

El acceso a la memoria externa puede ser de dos clases : acceso a la memoria externa de programa y acceso a la memoria externa de datos.

El acceso a la memoria externa de programa se produce bajo dos condiciones:

1.- Si el terminal EA=0 o

2) Si el terminal EA=1 y el contador de programa (PC) apunta una dirección superior a 1FFFH

En este acceso el puerto P0 es dedicado a la parte baja del bus de direcciones y al bus de datos multiplexados en el tiempo, y el puerto P2 está dedicado a la parte alta del bus de direcciones. La operación de acceso se lo realiza de la siguiente manera

El puerto P0 emite el byte bajo del contador de programa (PCL) y la señal ALE se encarga de introducir esta dirección dentro de un Latch, al mismo tiempo que se *emite* PCL el byte alto del contador de programa (PCH) se emite por el puerto 2, completando la dirección de 16 bits que será apuntada en la memoria externa de programas, ahora la señal PSEN que en este caso se emite dos veces cada ciclo de maquina da el permiso de lectura en la memoria y el dato leído es ingresado por el puerto P0.

El acceso a la memoria externa de datos permite direcciones de 8 o 16 bits, para este

acceso la señal PSEN no es emitida en su lugar se utilizan las señales RD y WR que son los permisos de lectura y escritura en RAM externa, dichas señales se emiten automáticamente cuando es la instrucción MOVX. De manera similar al acceso a memoria externa de programa el puerto P0 multiplexa en el tiempo PCL y el bus de datos, pero el puerto P2 emite el PCH solo si el acceso es de 16 bits, para el caso de 8 bits P2 emite el contenido del latch.

2.1.4 TEMPORIZADORES-CONTADORES

El MCS-52 tiene 3 registros temporizadores - contadores de 16 bits, los cuales son: Timer 0, Timer 1, Timer 2. ,los cuales se pueden configurar para que trabajen como temporizadores o como contadores.

La función de temporizador equivale a que los registros de cuenta se incrementen dependiendo de la frecuencia del oscilador (generalmente la frecuencia de incremento será 1/12 frecuencia del oscilador), en cambio la función como contador equivale a que los registros de cuenta se incrementen con cada flanco descendente aplicado a los terminales T0, T1 o T2.

El Timer 0 y 1 tienen 4 modos de operación, en cambio el Timer 2 tiene 3 modos de operación, a continuación se explican los diferentes modos de trabajo para estos 3 timers.

2.1.4.1 TIMER 0 (T0) Y TIMER 1 (T1)

Una representación de estos Timers se lo logra por medio del circuito de la figura 2.4

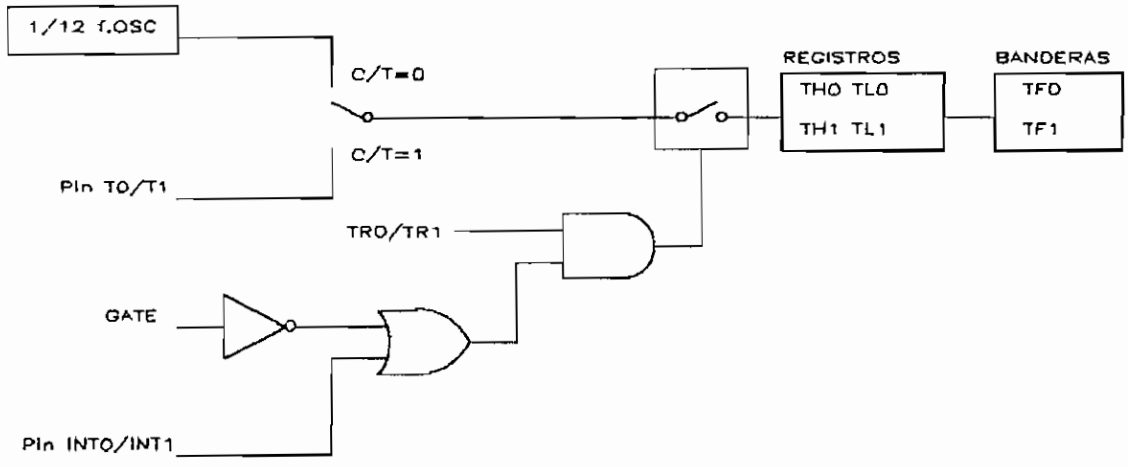


FIGURA 2.4 Representación de Timers 0 y 1

donde se puede observar que:

La selección del tipo de función del timer se lo realiza a través del bit C/T, el cual si es cero actúa como temporizador, caso contrario funcionaría como contador de impulsos

Para que se produzca el arranque o habilitación de los timers, es decir provocar que los registros de cuenta (TL0 / TH0, TL1 / TH1) se incrementen se requiere que la salida de la compuerta AND sea igual a 1 lo cual se lo puede lograr con cualquiera de las siguientes combinaciones.

Bit TRi	Bit GATE	TERMINAL INTi	TIMERi
1	0	x	Activado
1	1	1	Activado
0	x	x	Desactivado

Tabla 2.3

donde i = 0,1

El bloque al final del diagrama representado como Banderas, será puesto a 1

siempre que se produzca desbordamiento de los registros de cuenta. Estas banderas TF0 y TF1 al igual que TR1 y TR0 son bits pertenecientes al registro TCON

TCON

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Los bits GATE y C/T al igual que M1 y M0 son propios de cada timer y pertenecen al registro TMOD, estos dos últimos bits son los que determinan el modo de operación de cada timer y son indicados en la tabla 2.4..

TMOD

GATE	C/T	M1	M0	GATE	C/T	M1	M0
<——————TIMER 1 ——————>				<——————TIMER 0——————>			

M1	M0	MODO	SIGNIFICADO
0	0	0	Temporizador / Contador de 13 bits
0	1	1	Temporizador/Contador de 16 bits
1	0	2	Temporizador/Contador de 8 bits con Auto - recarga
1	1	3	Doble temporizador / contador de 8 bits

Tabla 2.4

A continuación se explican cada uno de estos modos

2.1.4.1.1 MODO 0

Este modo indica que el Timer 0 o 1 trabajan como temporizadores/contadores de 13 bits, es decir los registros de cuenta se repartirán de la siguiente manera:

Los 8 bits mas significativos para TH0 o TH1, y los 5 bits de menor peso para TL0 o TL1, (los 3 bits de mas peso de TL0 o TL1 son indeterminados y no deben tomarse en cuenta). Es decir el registro TH se incrementará 1 luego de que el registro TL haya tenido 32 incrementos. Con respecto a la activación de la bandera TF0 o TF1 se tiene que èsta se activará cuando se produzca la siguiente transición:

THI	TLI	TFI
1111 1111	XXX1 1111	0
0000 0000	XXX0 0000	1

Tabla 2.5

i=0,1

2.1.4.1.2 MODO 1

El funcionamiento en modo1 es similar que el modo 0 excepto que los registros de cuenta TL0 / TL1, TH0 / TH1 son cada uno de 8 bits, lo cual permite que trabaje como temporizador / contador de 16 bits.

La activación de la bandera se producirá cuando se detecte la siguiente transición

THI	TLI	TFI
1111 1111	1111 1111	0
0000 0000	0000 0000	1

Tabla 2.6

i=0,1

2.1.4.1.3 MODO 2

Para este modo solo el registro TL0 o TL1 se incrementa, es decir actúa como temporizador/contador de 8 bits, mientras que el registro TH0 o TH1 determinan el valor que será cargado al registro TL cuando se produzca el desbordamiento, es decir para la próxima cuenta los registros TL empezarán desde el valor de TH., como en los otros modos al producirse la transición la bandera TF0 o TF1 será puesta a 1

2.1.4.1.4 MODO 3

Para el Timer 0 el modo 3 significa tener 2 contadores independientes :

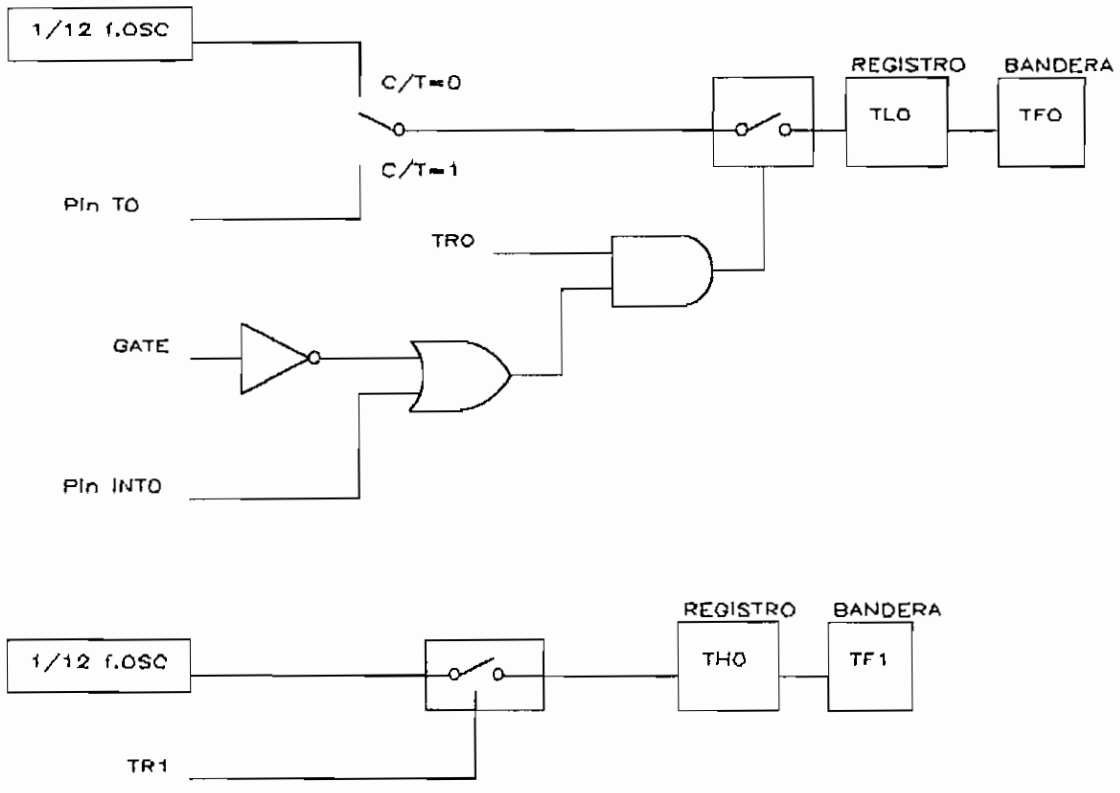


FIGURA 2.5 Timer0 en modo 3

El primer contador será equivalente al del modo 2 pero sin recarga, es decir el registro TL0 es el registro de cuenta, activará la bandera TF0 al producirse la transición de 255 a 0, pero como no hay recarga, para la próxima cuenta empezará desde cero.

El segundo Contador será el registro TH0 el cual siempre operará como temporizador y la habilitación de este dependerá del bit de control TR1 el cual si es 1 habilita este contador caso contrario no, además como bandera de desbordamiento tendrá asignada la bandera TF1.

Para el timer 1 el modo 3 es sinónimo de inhabilitado ya que en este modo los registros de cuenta TL1 y TH1 no sufren alteración alguna

2.1.4.2 TIMER 2 (T2)

Al igual que los timers 0 y 1 el timer 2 puede trabajar como temporizador o como contador de flancos descendentes ingresados a través del terminal T2. La configuración del Timer 2 se lo realiza a través del registro T2CON el cual se describe segidamente.

T2CON

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2

C/T2=0 Permite que trabaje como temporizador, es decir se incrementa con una frecuencia igual a 1/12 de la frecuencia del reloj para los modos autorrecarga y captura, en cambio para el modo generador de baudios se incrementará a razón de 1/2 de la frecuencia de reloj

C/T2=1 Permite que opere como contador de flancos descendentes a través del terminal T2

TR2 0 Deshabilita el Timer2

1 Arranca el Timer2

Los bits RCLK, TCLK, CP/RL2 y TR2 determinan el modo de operación del timer 2 , como se indica en la tabla 2.7.

RCLK or TCLK	CP/RL2	TR2	MODO
0	0	1	Temporizador / Contador de 16 bits con autorrecarga
0	1	1	Temporizador / Contador de16 blts con captura
1	X	1	Generador de baudios
X	X	0	DESACTIVADO

Tabla 2.7

Los bits TF2 y EXF2 son banderas del timer 2 y como son activadas se lo explica seguidamente para cada modo.

2.1.4.2.1 MODO AUTORRECARGA

Se presenta el siguiente diagrama que ayuda a la compresión del modo autorrecarga.

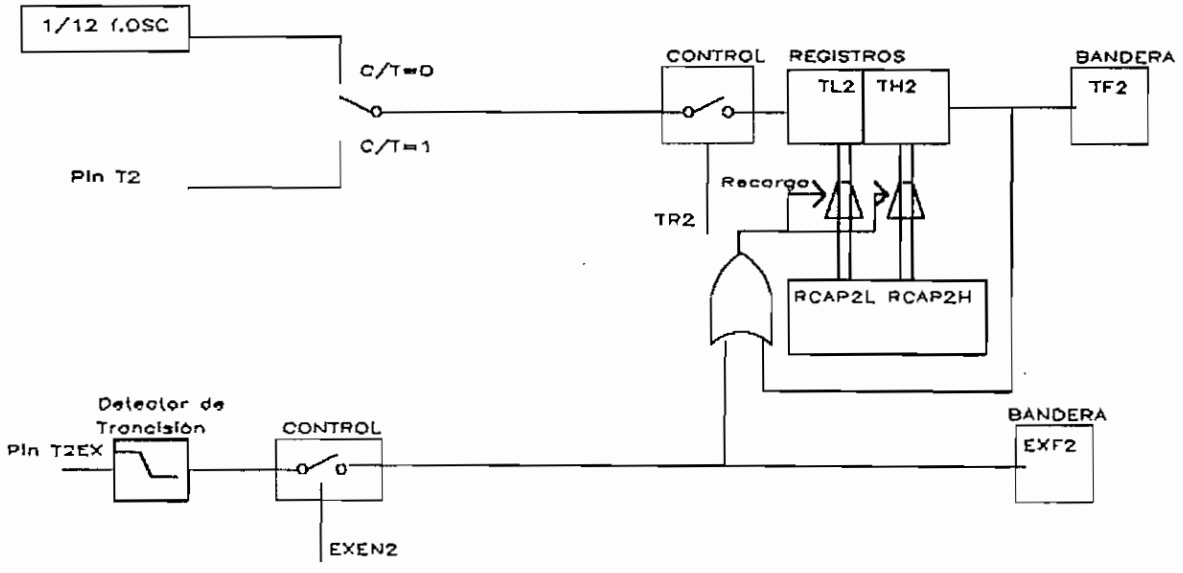


FIGURA 2.6 Timer 2 en modo autorrecarga

Como se puede notar del diagrama, si el bit EXEN2 es 0, el timer 2 puede operar como temporizador o contador de 16 bits activando la bandera TF2 si se produce desbordamiento, momento en el cual produce que los registros TL2 y TH2 sean cargados con los valores de los registros RCAP2L y RCAP2H respectivamente.

Si el bit EXEN2 es 1 puede hacer todo lo dicho anteriormente pero además si un flanco descendente se detecta en el terminal T2EX en ese momento los registros TL2 y TH2 son cargados con los valores de los registros de captura RCAP2L y RCAP2H respectivamente, y además la bandera EXF2 es puesta a 1.

2.1.4.2.2 MODO CAPTURA

Se presenta el siguiente diagrama que ayuda a la compresión del modo captura.

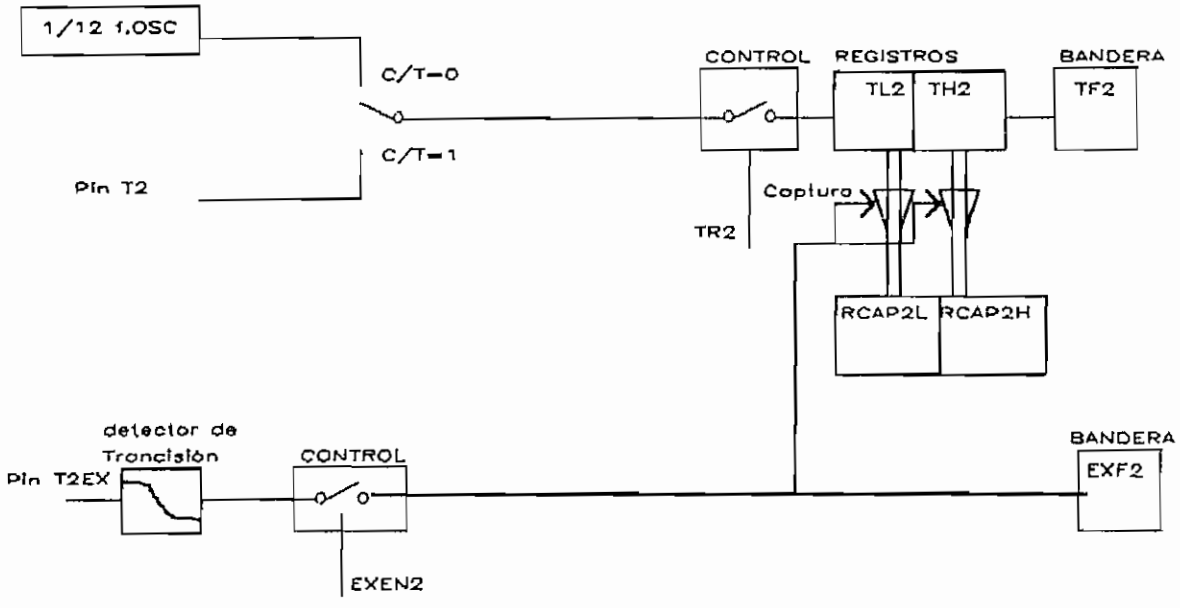


FIGURA 2.7 Timer 2 en modo captura

Como se puede notar del diagrama, si el bit EXEN2 es 0, el timer 2 puede operar como temporizador o contador de 16 bits activando la bandera TF2 si se produce desbordamiento.

Si el bit EXEN2 es 1 puede hacer todo lo dicho anteriormente pero además si un flanco descendente se detecta en el terminal T2EX en ese momento los valores de los registros TL2 y TH2 son capturados en los registros RCAP2L y RCAP2H respectivamente, y además la bandera EXF2 es puesta a 1.

2.1.4.2.3 GENERADOR DE BAUDIOS

Se presenta el siguiente diagrama que ayuda a la compresión de la generación de baudios

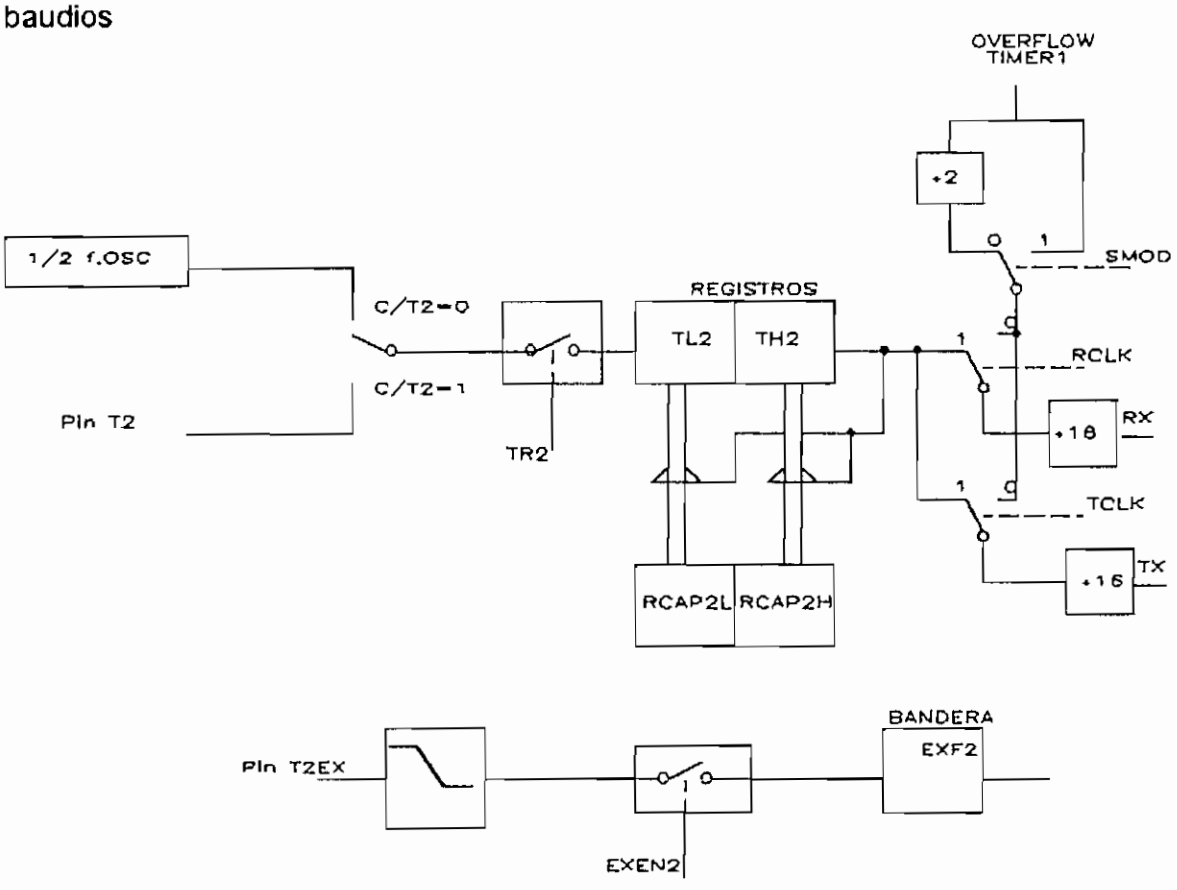


FIGURA 2.8 Timer 2 como generador de baudios

Del diagrama se nota que :

El timer 2 trabaja como Temporizador / Contador de 16 bits con autorrecarga pero a 1/2 de la frecuencia del oscilador, además si se produce desbordamiento no se activa la bandera TF2 como ocurría en otros modos.

Generalmente se configura el timer 2 como temporizador con lo que las velocidades en baudios para la transmisión o recepción estarían dados por

$$\text{Baudios} = \frac{\text{Relación de desbordamiento del timer 2}}{16}$$

pero:

$$\text{Relación de desbordamiento del timer 2} = \frac{\text{frecuencia del oscilador}}{2 * (65536 - \text{RCAP2L,RCAP2H})}$$

entonces:

$$\text{Baudios} = \frac{\text{frecuencia del oscilador}}{32 * (65536 - \text{RCAP2H,RCAP2L})}$$

donde RCAP2H,RCAP2L es un número de 2 bytes compuesto por los contenidos de estos registros

Nota: Si EXEN2=1 y se detecta un flanco descendente en el terminal T2EX se activa la bandera EXF2

2.1.5 PUERTO SERIAL

El puerto serie del Microcontrolador trabaja en el modo full-duplex es decir que puede transmitir y recibir simultáneamente, Para el caso de la transmisión se debe cargar el byte a transmitir en el registro SBUF del SFR, y para la recepción se accede al byte transmitido desde el exterior leyendo el registro SBUF.

El puerto serie puede operar en 4 modos lo cual se establece a través de los bits SM1 y SM0 del registro SCON.

SCON

SM0	**SM1**	SM2	REN	TB8	RB8	TI	RI

SM0	**SM1**	**MODO**	**VELOCIDAD**
0	0	0 (Registro de desplazamiento)	f.osc/2
0	1	1 (UART de 8 bits)	Variable
1	0	2 (UART de 9 bits)	f.osc/64 o f.osc/32
1	1	3 (UART de 9 bits)	Variable

Tabla 2.8

UART= Universal Asynchronous Receiver and Transmiter

2.1.5.1 MODO 0

A través del terminal RXD se transmiten o reciben los datos y por el terminal TXD se presenta los impulsos de desplazamiento del reloj. Los datos son transmitidos o recibidos utilizando un formato de 8 bits, siendo el primer bit tanto para la transmisión y recepción el bit menos significativo. En este modo la velocidad con la que se puede establecer la comunicación es igual a 1/12 de la frecuencia del reloj del microcontrolador.

La transmisión se inicia cuando el dato a transmitir es escrito en el registro SBUF y termina cuando la bandera TI es activada (TI=1)

La recepción se inicia cuando REN=1 y RI=0 y termina cuando la bandera RI es activada (RI=1)

2.1.5.2 MODO 1

Utiliza el terminal TXD para la transmisión y el terminal RXD para la recepción. Los datos transmitidos o recibidos utilizan un formato de 10 bits repartidos de la siguiente manera

Bit Start=0	8 Bits de Datos LSB MSB X X X X X X X X	Bit Stop = 1

La velocidad con la que se realiza la comunicación está dada por :

Cuando se utiliza el timer 1 como generador de baudios:

Se puede configurar al timer 1 en modo 2 (8 bits con autorrecarga) entonces:

$$\text{Baudios} = \frac{2^{SMOD}}{64} * \frac{\text{frecuencia del osilador}}{12 * (256 - TH1)}$$

donde SMOD es el bit 7 del registro PCON..

Cuando se utiliza el timer 2 como generador de baudios se tiene que:

$$\text{Baudios} = \frac{\text{frecuencia del osilador}}{32 * (65536 - (RCAP2H, RCAP2L))}$$

La transmisión se inicia cuando el dato a transmitir es escrito en el registro SBUF y termina cuando la bandera TI es activada (TI=1)

La recepción se inicia cuando REN=1 y se detecta un flanco descendente (bit start en RXD) y termina cuando la bandera RI es activada (RI=1)

2.1.5.3 MODO 2

Utiliza el terminal TXD para la transmisión y el terminal RXD para la recepción. Los datos transmitidos o recibidos utilizan un formato de 11 bits repartidos de la siguiente manera

Bit Start =0	8 Bits de Datos LSB MSB X X X X X X X X	Noneno bit programable	Bit Stop = 1

El noveno bit puede ser 0 o 1 y se refleja en el bit TB8 durante la transmisión o en el bit RB8 durante la recepción, ambos bits pertenecen al registro SCON.

La velocidad con la que se puede establecer la comunicación está dada por :

$$\text{Baudios} = \frac{2^{SMOD}}{64} * \text{frecuencia del oscilador}$$

donde SMOD es el bit 7 del registro PCON, es decir en este modo se pueden tener velocidades de 1/32 o 1/64 de la frecuencia del oscilador.

La transmisión se inicia cuando el dato a transmitir es escrito en el registro SBUF y termina cuando la bandera TI es activada (TI=1)

La recepción se inicia cuando REN=1 y se detecta un flanco descendente (bit start en RXD). y termina cuando la bandera RI es activada (RI=1)

2.1.5.4 MODO 3

Utiliza el terminal TXD para la transmisión y el terminal RXD para la recepción. Los datos transmitidos o recibidos utilizan un formato de 11 bits repartidos de la siguiente manera

Bit Start =0	8 Bits de Datos LSB MSB X X X X X X X X	Noneno bit programable	Bit Stop = 1

La velocidad con la que se puede establecer la comunicación está dada por :

Si se utiliza el timer1 como generador de baudios:

Se puede configurar al timer 1 en modo 2 (8 bits con autorrecarga) entonces

$$\text{Baudios} = \frac{2^{SMOD}}{64} * \frac{\text{frecuencia del osilador}}{12 * (256 - TH1)}$$

donde SMOD es el bit 7 del registro PCON..

Si se utiliza el timer 2 como generador de baudios como ya se explicó en el numeral 2.1.4.2.3 (Generador de baudios).

$$\text{Baudios} = \frac{\text{frecuencia del osilador}}{32 * (65536 - (RCAP2H, RCAP2L))}$$

La transmisión se inicia cuando el dato a transmitir es escrito en el registro SBUF y termina cuando la bandera TI es activada (TI=1)

La recepción se inicia cuando REN=1 y se detecta un flanco descendente (bit start en RXD) y termina cuando la bandera RI es activada (RI=1)

2.1.6 INTERRUPCIONES

El Microcontrolador MCS-52 posee 6 fuentes de interrupción, las cuales de ser solicitadas generan la ejecución de un subprograma específico cuya dirección de inicio se establece en una tabla de vectorización.

La petición del servicio de interrupción se lo realiza a través de las banderas de interrupción propios de cada fuente, la activación de dichas banderas se indica a continuación.

Fuente de Interrupción	**Bandera**	**Activación de la Bandera**
Externa 0	IE0	Se detecta un flanco descendente en el terminal INT0, o el terminal INT0 está en nivel bajo, la selección de cualquiera de estas 2 formas se la realiza a través del bit 1 del registro TCON
Timer 0	TF0	Cuando se produce el desbordamiento de los registros de cuenta del Timer 0
Externa 1	IE1	Se detecta un flanco descendente en el terminal INT1, o el terminal INT1 está en nivel bajo, la selección de cualquiera de estas 2 formas se la realiza a través del bit 2 del registro TCON
Timer 1	TF1	Cuando se produce el desbordamiento de los registros de cuenta del Timer 1
Puerto	RI	Al finalizar la recepción de datos

Serie	TI	Al finalizar la transmisión de datos
Timer 2	TF2	Cuando se produce el desbordamiento de los registros de cuenta del Timer 2
	EXF2	Si el bit EXEN2 del registro T2CON es puesto a 1 y se detecta un flanco descendente en el terminal T2EX

Tabla 2.10

TCON

TF1	TR1	TF0	TR0	**IE1**	**IT1**	**IE0**	**IT0**

ITi=0 La bandera de interrupción IEi se activa por nivel bajo del terminal INTi

ITi=1 La bandera de interrupción IEi se activa por flanco de bajada en el terminal INTi

IEi= Bandera de interrupción externa i

donde i= 0,1

Pero la petición del servicio de interrupción no es suficiente para que ésta sea atendida, ya que es necesario que la interrupción esté habilitada y se debe también tomar en cuenta el nivel de prioridad, lo cual se establece a través de los registros IE e IP del SFR.

REGISTRO IE (INTERRUPT ENABLE)

b7	b6	b5	b4	b3	b2	b1	b0
EA	X	ET2	ES	ET1	EX1	ET0	EX0
	X	Timer 2	Puerto Serie	Timer 1	Externa 1	Timer 0	Externa 0

Colocando un Uno en cualquiera de los bits 0 al 5 se habilita individualmente cada una de las interrupciones indicadas, en cambio poniendo un cero deshabilita las interrupciones.

El bit 6 es un bit reservado

El bit 7, (EA) es el encargado de habilitar a todas las interrupciones que tienen puesto 1 en su bit de habilitación individual, esto se consigue con EA=1. En cambio si EA=0 ninguna interrupción es reconocida

Ahora para el caso en que mas de una fuente de interrupción estè habilitada se debe preveer la condición de que un servicio de interrupción sea interrumpido por otra fuente de interrupción o que se solicite el servicio de 2 fuentes o mas de interrupción al mismo tiempo, en este caso se debería establecer un nivel de prioridades lo cual se establece a travès del registro IP

REGISTRO IP (INTERRUPT PRIORITY)

b7	b6	b5	b4	b3	b2	b1	b0
X	X	PT2	PS	PT1	PX1	PT0	PX0
		Timer 2	Puerto Serie	Timer 1	Externa 1	Timer 0	Externa 0

Entonces con un Uno en el bit de prioridad de la respectiva fuente de interrupción se establece un nivel alto, y un cero establece un nivel de prioridad bajo, con lo que :

Una interrupción de nivel de prioridad bajo puede ser interrumpida por una de nivel de prioridad alto, pero no por otra de nivel de prioridad bajo. Una interrupción de prioridad alta no puede ser interrumpida por otra fuente de interrupción salvo que se trate de la interrupción externa RESET.

Ahora si dos servicios de interrupción de distintos niveles de prioridad son solicitados al mismo tiempo, la interrupción de nivel de prioridad alto es servida, pero si los niveles de prioridad son los mismos entonces se establece un segundo nivel de prioridades el cual se indica en la tabla 2.11

Nivel de prioridades para fuentes de interrupción con el mismo nivel

Fuente de interrupción	Prioridad
Externa 0	1 (más alta)
Timer 0	2
Externa 1	3
Timer 1	4
Puerto Serie	5
Timer 2	6 (más baja)

Tabla 2.11

Una vez indicado todo lo que respecta a la manera de acceder al servicio de interrupción se procede a explicar las características de este servicio.

El servicio de interrupción como ya se dijo consiste en la ejecución de un subprograma escrito a partir de la localidad asignada de acuerdo a la tabla de vectorización,, es decir cuando se accede a este servicio se generaría un LCALL al vector de interrupción .

TABLA DE VECTORIZACIONES

Fuente de interrupción	Dirección
Externa 0	0003H
Timer 0	000BH
Externa 1	0013H
Timer 1	001BH
Puerto Serie	0023H
Timer 2	002BH

Tabla 2.12

El espacio disponible para escribir la rutina de interrupción es de 8 localidades de memoria. De no ser suficientes estas 8 localidades de memoria lo que se puede hacer es desviar la rutina de interrupción a una zona mas amplia de la memoria de programa mediante una instrucción de salto.

Esta rutina deberá siempre terminar con la instrucción RETI la cual permitirá el regreso al programa principal.

Con respecto a las banderas de interrupción hay que señalar que algunas de èstas serán borradas automáticamente por hardware (el propio microcontrolador) y otras deberán ser borradas por software es decir con alguna instrucción que ponga a cero la bandera. Estos aspectos se indican en la tabla 2.13.

Bandera	**Tipo de borrado**
IE0	Hardware cuando se activa por flanco Cuando se activa por nivel la bandera es borrada al cambiar de nivel
TF0	Hardware
IE1	Hardware cuando se activa por flanco Cuando se activa por nivel la bandera es borrada al cambiar de nivel
TF1	Hardware
RI	Software
TI	Software
TF2	Software
EXF2	Software

Tabla 2.13

2.2 ORGANIZACION DE LA MEMORIA EN EL MCS-52

El microcontrolador MCS-52 tiene 3 tipos de espacios de direcciones para la memoria los cuales son:

* 64 Kbytes de memoria de programa
* 64 Kbytes de memoria de datos externa
* 384 bytes de memoria de datos interna

2.2.1 MEMORIA DE PROGRAMA

Los 64 Kbytes de memoria de programa puede estar compuesta de la siguiente manera:

Si EA=1 esta memoria está compuesta por 8K de memoria interna y el resto de memoria externa ya que en esta condición el contador de programa busca direcciones de memoria interna para direcciones menores a 2000H, pero si el contador apunta direcciones mayores o iguales a 2000H buscará en la memoria externa.

Si EA=0 la memoria de programas será completamente externa ya que el contador de programa independiente de la dirección buscará en la memoria externa.

Cabe señalar que las localidades 0000H a 002BH son utilizadas para las rutinas de servicio de interrupción.

2.2.2 MEMORIA DE DATOS

La memoria de datos puede ser externa o interna, pudiendo acceder a la memoria de datos externa sólo cuando la instrucción MOVX es ejecutada.

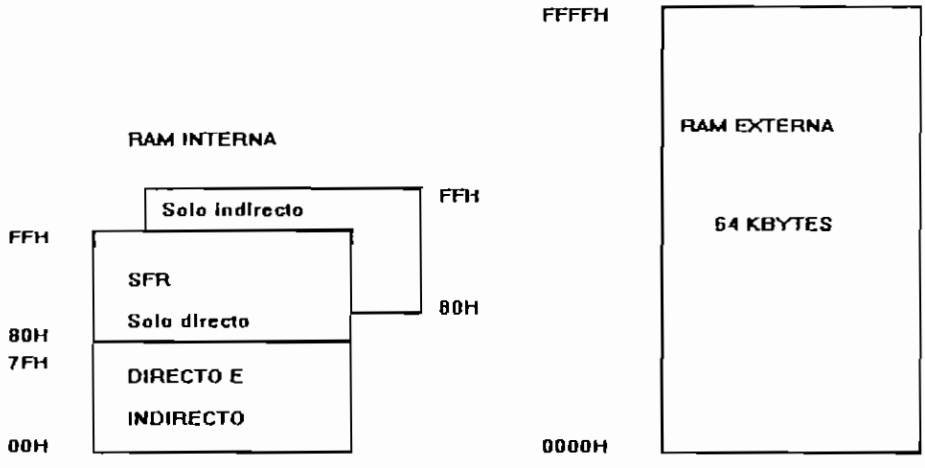


FIGURA 2.9 Memoria de Datos Interna y Externa

La memoria de datos interna está dividida en 3 áreas

* Area de direccionamiento indirecto y directo
* Area de direccionamiento directo y SFR
* Area de direccionamiento sólo indirecto

2.2.2.1 AREA DE DIRECCIONAMIENTO DIRECTO E INDIRECTO

Esta área corresponde al espacio de direcciones 00H a 7FH (128 bytes), a la cual se puede acceder con los modos de direccionamiento directo, indirecto, por registro, inmediato, e implícito y como se puede ver en la figura 2.10 está repartido de la siguiente manera:

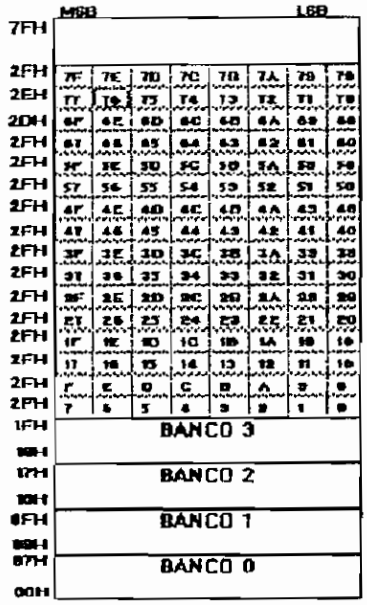


FIGURA 2.10 Distribución del área de direccionamiento directo

* Dentro del área 00H a 1FH se encuentran los 4 bancos de registros (banco 0, 1, 2 y 3), cada banco está formado por 8 registros denominados R0, R1, R2, R3, R4, R5, R6 y R7, La selección del banco de registros activo se lo realiza a través de los bits RS1 y RS0 del registro PSW del SFR que será explicado mas adelante, después de un reset el banco activo es el banco 0

* El espacio de direcciones de 20H a 2FH corresponde al área direccionable bit a bit, es decir se puede acceder a un total de 128 bits, los cuales son direccionados como 00H hasta 7FH. Cabe señalar que también se puede acceder a los bytes (Ver figura 2.10)

* El espacio de direcciones 30H a 7FH corresponde al área Scratch Pad que es un espacio de memoria donde puede trabajar el usuario.

2.2.2.2 AREA DE DIRECCIONAMIENTO SOLO DIRECTO y SFR

Dentro de esta área se encuentran los registros de función especial que utiliza el MCS-52, los cuales son indicados en la siguiente tabla

REGISTRO	NOMBRE	DIRECCION
*B	Registro B	F0H
*ACC	Acumulador	E0H
*PSW	Palabra de Estado de Programa	D0H
TH2	Byte alto (Timer 2)	CDH
TL2	Byte bajo (Timer 2)	CCH
RCAP2H	Byte alto del registro de captura	CBH
RCAP2L	Byte bajo del registro de captura	CAH
*T2CON	Control (Timer 2)	C8H
*IP	Prioridad de interrupciones	B8H
*P3	Puerto 3	B0H
*IE	Habilitación de Interrupciones	A8H
*P2	Puerto 2	A0H
SBUF	Buffer de datos del puerto serie	99H
*SCON	Control del puerto serie	98H
*P1	Puerto 1	90H
TH1	Byte alto (timer 1)	8DH
TH0	Byte alto (timer 0)	8CH
TL1	Byte bajo (timer 1)	8BH
TL0	Byte bajo (timer 0)	8AH
TMOD	Control Modo temporizador / contador	89H
*TCON	Control temporizador / contador	88H
PCON	Control consumo de potencia	87H
DPH	Byte alto del DPTR	83H
DPL	Byte bajo del DPTR	82H
SP	Puntero de la memoria de pila	81H
*P0	Puerto 0	80H

Tabla 2.14 Registros de Función Especial (SFR)

Todos los registros marcados con * son direccionables bit a bit

El registro PSW contiene las banderas C, AC, OV y otros bits que a continuación se explican.

PSW Este registro contiene la información de estado de la CPU en cada ciclo de instrucción, y es direccionable bit a bit.

PSW

C	AC	F0	RS1 RS0	OV	-	P
Carry	Acarreo Auxiliar	Bandera de propósito general	Seleccionan el banco de registros activo	Overflow	Definible por el usuario	bit de paridad

P=1 Indica que el número de unos del acumulador es impar,

P=0 Indica que el número de unos del acumulador es par.

RS1	RS0	BANCO ACTIVO
0	0	Banco 0 (00-07H)
0	1	Banco 1 (08-0FH)
1	0	Banco 2 (10-17H)
1	1	Banco 3 (18-1FH)

Tabla 2.15 Selección del Banco Activo de Registros

2.2.2.3 AREA DE DIRECCIONAMIENTO SOLO INDIRECTO

Esta zona ocupa el espacio de direcciones de 80H a FFH, que como se puede apreciar en la figura 2.9 son las mismas direcciones que corresponden a la Zona de los SFR, pero la diferencia radica en el camino de acceso a cada zona, por ahora se dirá que para esta zona sólo se puede acceder con modo de direccionamiento indirecto.

2.2.3 MODOS DE DIRECCIONAMIENTO

Hasta el momento se ha indicado la manera como está organizada la memoria para el MCS-52, pero no se ha dicho nada de la manera de accesar a estos espacios de direcciones, lo cual se logra a través de los modos de direccionamiento.

2.2.3.1 DIRECCIONAMIENTO INMEDIATO

Se refiere a que el operando está después del código de operación.

Ejemplo: MOV A,#8

CJNE A,#3,5

2.2.3.2 DIRECCIONAMIENTO A LOS REGISTROS

Dentro del código de la operación se especifica el registro con el cual se opera.

Ejemplo: MOV A,R4

ADD A,R1

2.2.3.3 DIRECCIONAMIENTO DIRECTO

El operando se especifica en la instrucción por un campo de dirección de 8 bits. Es el único modo con el que se puede acceder al SFR.

Ejemplo: MOV A,40H

MOV TCON,23H

2.2.3.4 DIRECCIONAMIENTO INDIRECTO

En el código de la instrucción se especifica un registro que contiene la dirección del operando.

Ejemplo: ADD A,@R0

MOV A,@R1

2.2.3.5 DIRECCIONAMIENTO INDEXADO

Permite acceder a la memoria de programa (sólo lectura), cuya dirección se establece mediante la suma de un registro base de 16 bits (DPTR o PC) y el Acumulador.

Ejemplo: MOVC A,@A+DPTR

MOVC A,@A+PC

2.2.3.6 DIRECCIONAMIENTO IMPLICITO

Corresponde a aquellas instrucciones que no tienen segundo operando, es decir el operando a ser operado está implícito en el código de operación.

Ejemplo: INC A

INC DPTR

2.3 SET DE INSTRUCCIONES

El set de instrucciones para el MCS-52 posee 111 instrucciones, las cuales respetan el siguiente formato:

Instrucción= Operación campo de operandos

Donde :

Operación : Indica la operación a realizar (ADD, MOV, CJNE, etc.)

Campo de operandos: Es aquí donde se especifican el tipo de datos y el modo de direccionamiento utilizado, a excepción de las instrucciones NOP, RET y RETI todas presentan como mínimo un operando y como máximo 3 los cuales van separados por comas.

Dependiendo del tipo de operación que realizan las instrucciones se las puede clasificar en los 5 grupos siguientes:

* Aritméticas
* Lógicas
* Transferencia
* Booleanas
* Salto

Las banderas Carry (C), Carry Auxiliar (AC) y Overflow (OV) pueden ser afectadas su valor de 2 maneras :

a) Indirectamente : que ocurre cuando la bandera sin ser especificada en los operandos resulta afectada. Ejemplos

ADD A,#255

MUL AB

b) Directamente : que ocurrre cuando la bandera o el registro PSW es el operando destino de alguna instrucción. Ejemplos

MOV PSW,#255

MOV C,7FH

Por lo tanto se dará una explicación de la alteración de las banderas sólo cuando ocurra de manera indirecta caso contario se dirá que las banderas sólo pueden ser afectadas de manera directa.

2.3.1 INSTRUCCIONES ARITMETICAS

Con estas instrucciones se pueden realizar 4 operaciones básicas: Suma, Resta, Multiplicación y División, a continuación se explican todas aquellas que caen en esta clasificación.

2.3.1.1 ADD

Especifica la operación Suma y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones

ADD A,Rn Realiza la suma entre el contenido del Acumulador y el registro Rn.

ADD A,@RI Realiza la suma entre el contenido del Acumulador y el contenido de la dirección apuntada por Ri.

ADD A,Direccion Realiza la suma entre el contenido del Acumulador y el contenido de Dirección.

ADD A,#Dato Realiza la suma entre el contenido del Acumulador y dato.

2.3.1.1.1 ALTERACION DE BANDERAS PARA ADD

Con el fin de lograr una mejor explicación la instrucción de suma se puede generalizar de la siguiente manera:

ADD A,origen (A)=(A)+(origen)

donde :

(A) Representa el contenido del Acumulador

(origen) Representa el contenido de Rn ó el contenido de la dirección apuntada por Ri ó el contenido de Dirección, y para el caso de #dato representa a dato.

CARRY (C)

Si (A)+(origen) > 255 entonces C=1

caso contrario C=0

CARRY AUXILIAR (AC)

si... $(A_{3-0})+(origen_{3-0}) > 15$ entonces $AC=1$

caso contrario $AC=0$

donde $(_{3-0})$ representa los 3 bits menos significativos

OVERFLOW (OV)

La alteración del valor de esta bandera se entiende mejor si se considera que la suma se realiza entre números con signo, es decir el octavo bit es el bit del signo.

(A)	(origen)	(A)+(origen)	OV
Positivo	Positivo	Negativo	1
Negativo	Negativo	Positivo	1

Tabla 2.16

Para cualquier otro caso el OV es puesto a Cero.

2.3.1.2 ADDC

Especifica la operación Suma con llevo y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones

ADDC A,Rn Realiza la suma entre el contenido del Acumulador, el registro Rn, y el Carry

ADDC A,@RI Realiza la suma entre el contenido del Acumulador, el contenido de la dirección apuntada por Ri y el Carry

ADDC A,Direccion Realiza la suma entre el contenido del Acumulador, el contenido de Dirección y el Carry

ADDC A,#Dato Realiza la suma entre el contenido del Acumulador, dato y el Carry

2.3.1.2.1 ALTERACION DE BANDERAS PARA ADDC

Con el fin de lograr una mejor explicación la instrucción de Suma con Carry se puede generalizar de la siguiente manera:

ADDC A,origen (A)=(A)+(origen)+C

donde :

(A) Representa el contenido del Acumulador

(origen) Representa el contenido de Rn ó el contenido de la dirección apuntada por Ri el contenido de Dirección, y para el caso de #dato representa a dato.

C El valor del carry

CARRY (C)

Si (A)+(origen)+C > 255 entonces C=1

caso contrario C=0

CARRY AUXILIAR (AC)

si... $(A_{3-0}) + (origen_{3-0}) + C > 15$ entonces $AC = 1$

caso contrario $AC = 0$

donde ($_{3-0}$) representa los 3 bits menos significativos

OVERFLOW (OV)

La alteración del valor de esta bandera se entiende mejor si se considera que la suma se realiza entre números con signo, es decir el octavo bit es el bit del signo, con lo que se tiene la siguiente tabla:

(A)	(origen)	(A)+(origen)+C	OV
Positivo	Positivo	Negativo	1
Negativo	Negativo	Positivo	1

Tabla 2.17

Para cualquier otro caso el OV es puesto a Cero.

2.3.1.3 SUBB

Especifica la operación Resta con debo y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones.

SUBB A,Rn Al contenido del Acumulador, le resta el contenido del registro Rn, y el Carry

SUBB A,@Ri Al contenido del Acumulador, le resta el contenido de la dirección apuntada por Ri, y el Carry

SUBB A,Direccion Al contenido del Acumulador, le resta el contenido de dirección y el Carry

SUBB A,#Dato Al contenido del Acumulador, le resta dato y el Carry

2.3.1.3.1 ALTERACION DE BANDERAS PARA SUBB

Con el fin de lograr una mejor explicación la instrucción de Resta con Carry se puede generalizar de la siguiente manera:

SUBB A,origen (A)=(A)-(origen)-C

donde :

(A) Representa el contenido del Acumulador

(origen) Representa el contenido de Rn ó el contenido de la dirección apuntada por Ri el contenido de Dirección, y para el caso de #dato representa a dato.

C El valor del carry

CARRY (C)

Si (A)-(origen)-C <0 entonces C=1

caso contrario C=0

CARRY AUXILIAR (AC)

si... $(A_{3-0}) - (origen_{3-0}) - C < 0$ entonces $AC = 1$

caso contrario $AC = 0$

donde $(_{3-0})$ representa los 3 bits menos significativos

OVERFLOW (OV)

La alteración del valor de esta bandera se entiende mejor si se considera que la resta se realiza entre números con signo, es decir el octavo bit es el bit del signo, con lo que se tiene la siguiente tabla:

(A)	(origen)	(A)-(origen)-C	OV
Positivo	Negativo	Negativo	1
Negativo	Positivo	Positivo	1

Tabla 2.18

Para cualquier otro caso el OV es puesto a Cero.

2.3.1.4 INC

Especifica la operación Incrementar y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones

INC A Al contenido del Acumulador se suma Uno

INC Rn Al contenido del registro Rn se suma Uno

INC Direccion Al contenido de Dirección se suma Uno

INC @RI Al contenido de la dirección apuntada por Ri se suma Uno

INC DPTR Al contenido del DPTR se suma Uno

Las Banderas sólo pueden ser afectadas directamente.

2.3.1.5 DEC

Especifica la operación Decrementar y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones

DEC A Al contenido del Acumulador se resta Uno

DEC Rn Al contenido del registro Rn se resta Uno

DEC Direccion Al contenido de Dirección se resta Uno

DEC @RI Al contenido de la dirección apuntada por Ri se resta Uno

Las Banderas sólo pueden ser afectadas directamente.

2.3.1.6 MUL

Especifica la operación Multiplicación y la única instrucción con este fin es:

MUL AB Multiplica el contenido del Acumulador con el contenido del Registro B, y el resultado que se presenta en formato de 16 bits se almacena de la siguiente manera:

En el Acumulador se guarda el byte menos significativo y en el Registro B el byte mas significativo

2.3.1.6.1 ALTERACION DE BANDERAS PARA MUL

CARRY (C)

Siempre lo pone a Cero

CARRY AUXILIAR (AC)

OVERFLOW(OV)

(A)*(B) < 256 entonces OV=1

Caso contrario OV=0

2.3.1.7 DIV

Especifica la operación División y la única instrucción con este fin es :

DIV AB Divide el contenido del Acumulador con el contenido del Registro B, y el resultado se almacena de la siguiente manera:

En el Acumulador se guarda el cociente y en el Registro B el residuo.

2.3.1.7.1 ALTERACION DE BANDERAS PARA DIV

CARRY (C)

Siempre lo pone a Cero

CARRY AUXILIAR (AC)

No sufre alteración

OVERFLOW(OV)

Si se puede realizar la división el OV es puesto a Cero

Si se realiza una división para Cero el OV es puesto a Uno

2.3.1.8 DA

Especifica la operación Ajuste Decimal la única instrucción con este fin es:

DA A Empaqueta en BCD el resultado de una operación de suma (ADD ó ADDC) que ha sido realizada con operandos BCD:, el resultado del ajuste decimal se lo guarda en el Acumulador.

Una representación analítica de la ejecución de esta instrucción consiste en :

$si...(A_{3-0}) > 9...OR...(AC) = 1$

$entonces...(A_{3-0}) = (A_{3-0}) + 6$

$y...si...(A_{7-4}) > 9...OR...(C) = 1$

$entonces...(A_{7-4}) = (A_{7-4}) + 6$

donde $(_{3-0})$ son los 3 bits menos significativos

y $(_{7-4})$ son los 3 bits más significativos

2.3.1.8.1 ALTERACION DE BANDERAS PARA DA

CARRY (C)

Si el contenido del Acumulador es mayor a 99 antes de realizar el ajuste decimal, el Carry es puesto a Uno

CARRY AUXILIAR (AC)

No sufre alteración

OVERFLOW (OV)

No sufre alteración

2.3.2 INSTRUCCIONES LOGICAS

Con estas instrucciones se pueden realizar operaciones como : AND, OR, OR Exclusivo, Complemento, Borrado, Rotación. El resultado de ejecutar estas instrucciones será guardado en el operando destino (primer operando)

2.3.2.1 ANL

Especifica la operación AND y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones

ANL A,Rn Realiza la operación AND entre el contenido del Acumulador y el registro Rn

ANL A,@RI Realiza la operación AND entre el contenido del Acumulador y el contenido de la dirección apuntada por Ri

ANL A,Direccion Realiza la operación AND entre el contenido del Acumulador y el contenido de Dirección

ANL A,#Dato Realiza la operación AND entre el contenido del Acumulador y dato

ANL Direccion,A Realiza la operación AND entre el contenido de Dirección y el contenido del Acumulador.

ANL Direccion,#Dato Realiza la operación AND entre el contenido de Dirección y dato

Las banderas solo pueden ser afectadas directamente

2.3.2.2 ORL

Especifica la operación OR y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones

ORL A,Rn Realiza la operación OR entre el contenido del Acumulador y el registro Rn

ORL A,@RI Realiza la operación OR entre el contenido del Acumulador y el contenido de la dirección apuntada por Ri

ORL A,Direccion Realiza la operación OR entre el contenido del Acumulador y el contenido de Dirección

ORL A,#Dato Realiza la operación OR entre el contenido del Acumulador y dato

ORL Direccion,A Realiza la operación OR entre el contenido de Dirección y el contenido del Acumulador.

ORL Direccion,#Dato Realiza la operación OR entre el contenido de Dirección y dato

Las Banderas sólo pueden ser afectadas directamente.

2.3.2.3 XRL

Especifica la operación XOR y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones

XRL A,Rn Realiza la operación OR exclusivo (XOR) entre el contenido del Acumulador y el registro Rn

XRL A,@RI Realiza la operación OR exclusivo (XOR) entre el contenido del Acumulador y el contenido de la dirección apuntada por Ri

XRL A,Direccion Realiza la operación OR exclusivo (XOR) entre el contenido del Acumulador y el contenido de Dirección

XRL A,#Dato Realiza la operación OR exclusivo (XOR) entre el contenido del Acumulador y dato

XRL Direccion,A Realiza la operación OR exclusivo (XOR) entre el contenido de Dirección y el contenido del Acumulador.

XRL Direccion,#Dato Realiza la operación OR exclusivo (XOR)entre el contenido de Dirección y dato

Las Banderas sólo pueden ser afectadas directamente.

2.3.2.4 CLR

Especifica la operación Borrar y la única instrucción con este fin orientada al byte es :

CLR A El contenido del Acumulador es puesto a Cero

2.3.2.5 CPL

Especifica la operación Complementar y la única instrucción con este fin orientada al byte es:

CPL A El contenido del Acumulador es complementado

2.3.2.6 RL

Especifica la operación Rotación a la izquierda y la única instrucción con este fin es :

RL A Los bits del Acumulador son desplazados hacia la izquierda un puesto y para el caso del bit 0 pasa a ocupar el lugar del bit 7.

2.3.2.7 RLC

Especifica la operación Rotación a la izquierda con Carry y la única instrucción con este fin es :

RLC A Los bits del Acumulador son desplazados hacia la izquierda un puesto y para el caso del bit 0 pasa a ocupar el lugar del Carry, y el Carry pasa a ocupar el lugar del bit 7.

Resulta claro que esta instrucción produce alteración del Carry (C), cuyo valor pasa a ser como se indicó el del bit 0 del Acumulador antes de ejecutar la instrucción.

2.3.2.8 RR

Especifica la operación Rotación a la derecha y la única instrucción con este fin es :

RR A Los bits del Acumulador son desplazados hacia la derecha un puesto y para el caso del bit 7 pasa a ocupar el lugar del bit 0

2.3.2.9 RRC

Especifica la operación Rotación a la derecha con Carry y la única instrucción con este fin es :

RRC A Los bits del Acumulador son desplazados hacia la derecha un puesto y para el caso del bit 7 pasa a ocupar el lugar del Carry, y el Carry pasa a ocupar el lugar del bit 0

Resulta evidente que esta instrucción produce alteración del Carry (C), cuyo valor pasa a ser como se indicó el del bit 7 del Acumulador antes de ejecutar la instrucción

2.3.2.10 SWAP

Especifica la operación Intercambio de nibbles y la única instrucción con este fin es :

SWAP A El nibble bajo del Acumulador es reemplazado por el nibble alto del Acumulador y viceversa.

2.3.3 INSTRUCCIONES DE TRANSFERENCIA

Estas instrucciones permiten realizar movimientos de datos de una localidad de memoria a otra. Las transferencias de datos pueden realizarse sobre :

* RAM Interna
* RAM Externa
* ROM

A continuación se explican cada una de las instrucciones que para este fin existen.

2.3.3.1 MOV

Especifica el movimiento de datos para localidades de RAM Interna y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones

MOV A,Rn Reemplaza el contenido del Acumulador por el contenido del registro Rn

MOV A,Direccion Reemplaza el contenido del Acumulador por el contenido de dirección

MOV A,@RI Reemplaza el contenido del Acumulador por el contenido de la dirección apuntada por Ri

MOV A,#Dato Dato reemplaza el contenido del Acumulador

MOV Rn,A Reemplaza el contenido del Registro Rn por el contenido Acumulador

MOV Rn,Direccion Reemplaza el contenido del Registro Rn por el contenido de dirección

MOV Rn,#Dato Dato reemplaza el contenido del Registro Rn

MOV Dirección,A Reemplaza el contenido de Dirección por el contenido del Acumulador

MOV Dirección,Rn Reemplaza el contenido de Dirección por el contenido del registro Rn

MOV Direccion,Direccion Reemplaza el contenido de la Dirección destino (primer operando) por el contenido de la dirección origen (segundo operando).

MOV Direccion,@Ri Reemplaza el contenido de Dirección por el contenido de la dirección apuntada por Ri

MOV Direccion,#Dato Dato reemplaza el contenido de Dirección

MOV @Ri,A Reemplaza el contenido de la dirección apuntada por Ri por el contenido del Acumulador

MOV @Ri,Direccion Reemplaza el contenido de la dirección apuntada por Ri por el contenido de dirección

MOV @Ri,#Dato Dato reemplaza el contenido de la dirección apuntada por Ri.

MOV DPTR,#Dato16 El dato de 16 bits (Dato16) reemplaza el contenido del DPTR.

Las Banderas sólo pueden ser afectadas directamente

2.3.3.2 MOVC

Especifica el traslado de datos pertenecientes a localidades de ROM hacia RAM interna, teniendo como único destino el Acumulador. Es decir permite leer datos de la memoria de programa. Para este tipo de transferencia sólo existen 2 instrucciones las cuales son:

MOVC A,@A+DPTR La suma de los contenidos del Acumulador y el DPTR determinan la dirección del dato a ser leído de la ROM, el cual será almacenado en el Acumulador.

MOVC A,@A+PC La suma de los contenidos del Acumulador y el PC (Contador de Programa) determinan la dirección del dato a ser leído de la ROM, el cual será almacenado en el Acumulador. Puesto que esta instrucción implica el Contador de Programa. no está por demás indicar que cuando esta instrucción es apuntada para su ejecución el contador de programa se incrementa 1, entonces es este valor incrementado el que se suma al contenido del Acumulador para apuntar la dirección del dato a ser leído de la ROM

MOVC A,@A+PC

Secuencia de operaciones

PC<—PC+1

(A)<——((A)+(PC))

2.3.3.3 MOVX

Especifica el traslado de datos pertenecientes a localidades de RAM externa hacia RAM interna, y viceversa. Estas instrucciones permiten leer de RAM externa y

escribir en la RAM externa y el único de modo de direccionamiento permitido es el de tipo indirecto. A continuación se explican dichas instrucciones.

MOVX A,@RI Realiza la lectura de RAM Externa, siendo la dirección del dato a ser leído la apuntada por Ri y que será almacenado en el Acumulador

MOVX A,@DPTR Lee el dato de RAM Externa, cuya dirección es igual al contenido del DPTR y lo almacena en el Acumulador.

MOVX @RI,A Transfiere el contenido del Acumulador a la dirección en RAM Externa apuntada por Ri. (Escritura en RAM Externa).

MOVX @DPTR,A Transfiere el contenido del Acumulador a la dirección en RAM Externa apuntada por el DPTR. (Escritura en RAM Externa).

2.3.3.4 PUSH

Especifica el traslado de datos hacia la pila (Stack), y la única instrucción existente para este fin es:

PUSH Direccion Coloca en la dirección de la pila dada por el contenido del SP aumentado Uno el contenido de Dirección. Dicho de otra manera la ejecución de esta instrucción implica la realización de las siguientes operaciones:

```
INC  SP
MOV  @SP,Direccion
```

2.3.3.5 POP

Especifica la extracción de datos de la pila (Stack), para ser colocados en alguna dirección de RAM Interna. La única instrucción existente para este fin es;

POP Direccion El SP apunta el dato de la pila (Stack) que será llevado a Dirección, adicionalmente hecho ésto decrementa el contenido del SP. Dicho de otra manera la ejecución de esta instrucción implica la realización de las siguientes operaciones:

```
MOV  Direccion,@SP
DEC  SP
```

2.3.3.6 XCH

Especifica la operación intercambio de datos y dependiendo del modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones

XCH A,Rn El contenido del Acumulador es reemplazado por el contenido del Registro Rn y viceversa.

XCH A,Direccion El contenido del Acumulador es reemplazado por el contenido de Dirección y viceversa.

XCH A,@Ri El contenido del Acumulador es reemplazado por el contenido de la dirección apuntada por Ri y viceversa.

Las Banderas sólo pueden ser afectadas directamente

2.3.3.7 XCHD

Especifica la operación intercambio de nibbles bajos y debido a que sólo soporta direccionamiento indirecto tenemos la instrucción

XCHD A,@RI El nibble bajo del Acumulador es reemplazado por el nibble bajo del contenido de la dirección apuntada por Ri y viceversa.

2.3.4 INSTRUCCIONES BOOLEANAS

Dentro de esta categoría tenemos todas las instrucciones orientadas al bit, es decir las operaciones que se realizan se las hace sobre bits. Así como para las instrucciones orientadas al byte el Acumulador representa un registro de propósito general, el Carry se puede decir representa un bit de propósito general. Las operaciones que se pueden realizar sobre bits pueden ser : lógicas, de transferencia y de salto.. Resulta evidente que el valor de las banderas C, AC, OV pueden ser alterados directamente.

A continuación se explican todas las Instrucciones Booleanas disponibles

2.3.4.1 ANL

Especifica la operación AND y puesto que el único operando de destino disponible es el Carry (C) obviamente resultará alterada esta bandera y puesta al valor que resulte de realizar esta operación.

ANL C,bit Ejecuta la operación AND entre el valor de bit y el Carry

ANL C,/bit Ejecuta la operación AND entre el valor de bit complementado y el Carry

2.3.4.2 ORL

Especifica la operación OR y puesto que el único operando de destino disponible es el Carry (C) obviamente resultará alterada esta bandera y puesta al valor que resulte de realizar esta operación.

ORL C,bit Ejecuta la operación OR entre el valor de bit y el Carry

ORL C,/bit Ejecuta la operación OR entre el valor de bit complementado y el Carry

2.3.4.3 MOV

Especifica el movimiento de bits, y con este fin se disponen de las siguientes instrucciones

MOV C,bit Reemplaza el valor del Carry con el valor del bit direccionado

MOV bit,C Reemplaza el valor del bit direccionado con el valor del Carry.

2.3.4.4 CLR

Especifica la operación de borrado, y con este fin se disponen de las siguientes instrucciones

CLR C El valor del Carry es puesto a Cero.

CLR bit El valor del bit direccionado es puesto a Cero

2.3.4.5 SETB

Especifica la operación Set, y con este fin se disponen de las siguientes instrucciones

SETB C El valor del Carry es puesto a Uno

SETB bit El valor del bit direccionado es puesto a Uno

2.3.4.6 CPL

Especifica la operación Complementar, y con este fin se disponen de las siguientes instrucciones

CPL C El valor del Carry es complementado.

CPL bit El valor del bit direccionado es complementado.

2.3.4.7 JC

Especifica la operación salto condicional, utilizando como condicionante el Carry, es decir si se cumple cierta condición se produce un salto, y con este fin se dispone de la instrucción

JC rel Al contador de programa se suma rel si el Carry es Uno

Secuencia de operaciones :

(PC)<———(PC)+2

Si (C)=1 entonces

(PC)<———(PC)+rel

2.3.4.8 JNC

Especifica la operación salto condicional, utilizando como condicionante el Carry, es decir si se cumple cierta condición se produce un salto, y con este fin se dispone de la instrucción

JNC rel Al contador de programa se suma rel si el Carry es Cero

Secuencia de operaciones :

(PC)<———(PC)+2

Si (C)=0 entonces

(PC)<———(PC)+rel

2.3.4.9 JB

Especifica la operación salto condicional, utilizando como condicionante un bit direccionado de forma directa, y con este fin se dispone de la instrucción

JB bit,rel Al contador de programa se suma rel si el valor del bit direccionado es Uno

Secuencia de operaciones :

(PC)<———(PC)+3

Si (bit)=1 entonces

(PC)<———(PC)+rel

2.3.4.10 JNB

Especifica la operación salto condicional, utilizando como condicionante un bit direccionado de forma directa, y con este fin se dispone de la instrucción

JNB bit,rel Al contador de programa se suma rel si el valor del bit direccionado es Cero

Secuencia de operaciones :

(PC)<———(PC)+3

Si (bit)=0 entonces

(PC)<———(PC)+rel

2.3.4.11 JBC

Especifica la operación salto condicional, utilizando como condicionante un bit direccionado de forma directa, y con este fin se dispone de la instrucción

JBC bit,rel Si el valor del bit direccionado es Uno, al contador de programa se suma rel y el valor del bit es puesto a Cero

Secuencia de operaciones :

(PC)<———(PC)+3

Si (bit)=1 entonces

(bit)<———0

(PC)<———(PC)+rel

2.3.5 INSTRUCCIONES DE SALTO

En esta categoría constan todas aquellas instrucciones orientadas al byte que pueden producir alguna alteración en el valor del Contador de Programa. Este tipo de instrucciones pueden ser incondicionales y condicionales.

2.3.5.1 INSTRUCCIONES DE SALTO INCONDICIONALES

Se les da el nombre de saltos incondicionales porque basta la ejecución de estas instrucciones para que el salto se produzca. Las instrucciones que trabajan de esta

manera son: ACALL, LCALL, RET, RETI, AJMP, LJMP, SJMP, JMP, a continuación se explican cada una

2.3.5.1.1 ACALL

Esta instrucción produce un salto máximo de 2Kbytes, y adicionalmente guarda el contenido del contador de programa antes de ejecutar el salto en la pila (Stack), su formato es

ACALL **direc11** Guarda el PC en la pila y procede a reemplazar los 11 bits menos significativos del contenido del PC por el valor direc11

Secuencia de operaciones

$(PC) \leftarrow (PC) + 2$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{7-0})$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{15-8})$
$(PC_{10-0}) \leftarrow Direc11$

2.3.5.1.2 LCALL

Esta instrucción puede producir saltos de hasta 64Kbytes, y adicionalmente guarda el contenido del contador de programa antes de ejecutar el salto en la pila (Stack), su formato es

LCALL **direc16** Guarda el PC en la pila y procede a reemplazar el contenido del PC por el valor direc16

Secuencia de operaciones

$(PC) \leftarrow (PC) + 3$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{7-0})$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{15-8})$
$(PC) \leftarrow Direc16$

2.3.5.1.3 RET

Esta instrucción es utilizada para regresar al programa principal luego de haber terminado de ejecutar una subrutina.

RET Recupera el byte alto y el byte bajo del PC en ese orden que se encuentra guardado en la pila

Secuencia de operaciones

$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

2.3.5.1.4 RETI

Esta instrucción es utilizada para regresar de las rutinas del servicio de interrupción.

RETI Recupera el byte alto y el byte bajo del PC en ese orden que se encuentra guardado en la pila

Secuencia de operaciones

$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

2.3.5.1.5 AJMP

Esta instrucción simplemente produce un salto máximo de 2Kbytes

AJMP direc11 Reemplaza los 11 bits menos significativos del contenido del PC por el valor direc11

Secuencia de Operaciones
$(PC) \leftarrow (PC) + 2$
$(PC_{10-0}) \leftarrow Direc11$

2.3.5.1.6 LJMP

Esta instrucción simplemente produce un salto como máximo de 64Kbytes

LJMP direc16 Reemplaza el contenido del PC por el valor direc16

Secuencia de operaciones
$(PC) \leftarrow Direc16$

2.3.5.1.7 SJMP

El salto se especifica a través de un número con signo, es decir puede saltar para adelante o para atrás.

SJMP rel Al contador de programa PC se suma rel

Secuencia de operaciones
$(PC) \leftarrow (PC) + 2$
Si $(A) = 0$ entonces
$(PC) \leftarrow (PC) + rel$

2.3.5.2.2 JNZ

El Acumulador es utilizado como condicionante para efectuar el salto

JNZ rel Si el contenido del Acumulador es diferente de cero al contador de programa (PC) se suma rel

Secuencia de operaciones
$(PC) \leftarrow (PC) + 2$
Si $(A) \neq 0$ entonces
$(PC) \leftarrow (PC) + rel$

2.3.5.2.3 CJNE

Los dos primeros operandos son los condicionantes para efectuar el salto, de acuerdo al modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones.

CJNE A,direc,rel Si el contenido del Acumulador es diferente del contenido de dirección al contador de programa (PC) se suma rel

CJNE A,#dato,rel Si el contenido del Acumulador es diferente de dato al contador de programa (PC) se suma rel

CJNE Rn,#dato,rel Si el contenido del Registro Rn es diferente de dato al contador de programa (PC) se suma rel

CJNE @RI,#dato,rel Si el contenido de la dirección apuntada por Ri es diferente de dato al contador de programa (PC) se suma rel

La ejecución de estas instrucciones produce alteración del Carry lo cual se explica a continuación.

Con el fin de lograr una mejor explicación estas instrucciones se pueden generalizar de la siguiente manera:

CJNE op1,op2,rel

donde :

(op1) Representa el contenido del Acumulador ó el contenido del registro Rn ó el contenido de la dirección apuntada por Ri

(op2) Representa el contenido de direc y para el caso de #dato representa a dato.

rel Representa rel (número con signo)

Secuencia de operaciones
(PC) ← (PC) + 3
Si (op1) ≠ (op2) entonces
(PC) ← (PC) + rel
Si (op1) < (op2) entonces
(C) ← 1
caso contrario
(C) ← 0

2.3.5.2.4 DJNZ

La condición del primer operando determina si se produce el salto. De acuerdo al modo de direccionamiento utilizado se pueden presentar las siguientes instrucciones.

DJNZ Rn,rel Decrementa el contenido del Registro Rn y si como resultado de realizar esta operación el nuevo contenido de Rn es diferente de Cero suma rel al contador de programa (PC)

DJNZ Direccion,rel Decrementa el contenido de Dirección y si como resultado de realizar esta operación el nuevo contenido de Dirección es diferente de Cero suma rel al contador de programa (PC)

Generalizando esta instrucción tenemos:

DJNZ (op1),rel

donde

(op1) Representa el contenido del Registro Rn ó el contenido de Dirección

entonces :

Secuencia de operaciones
$(PC) \leftarrow (PC) + 2$
Si $(op1) \leftarrow (op1) - 1$
Si $(op1) \neq 0$ entonces
$(PC) \leftarrow (PC) + rel$

2.3.6 NOP

Esta instrucción no ejecuta ninguna operación, por lo cual suele ser usada para crear rutinas de retardo.

CAPITULO 3

DESARROLLO DEL SOFTWARE

Para la implementación de este Software se tuvo que tomar ciertas consideraciones respecto a la configuración y funcionamiento del Microcontrolador MCS-52.

- EL MCS-52 está configurado como Microprocesador operando al máximo de capacidad en el manejo de memoria externa, es decir se puede direccionar 64K de memoria externa de programa, y 64K de memoria de datos externa, es por eso que se asume que los puertos 0 y 2 están siendo usados como buses de datos y direcciones, por lo cual los pines correspondientes a estos puertos no están disponibles.

- Cada vez que se ejecuta una instrucción, el programa asume que el código de máquina correspondiente a esta instrucción estará escrito a partir de la dirección apuntada por el contador de programa (PC)

- De activarse el servicio a una interrupción esta será ejecutada de una forma generalizada.

Hechas estas consideraciones se presenta el proceso de análisis que llevó a la implementación del VGI MCS-52.

3.1 DIAGRAMAS DE FLUJO

A continuación se muestran los diagramas de flujo que indican de una manera general la forma en que el programa fue implementado.

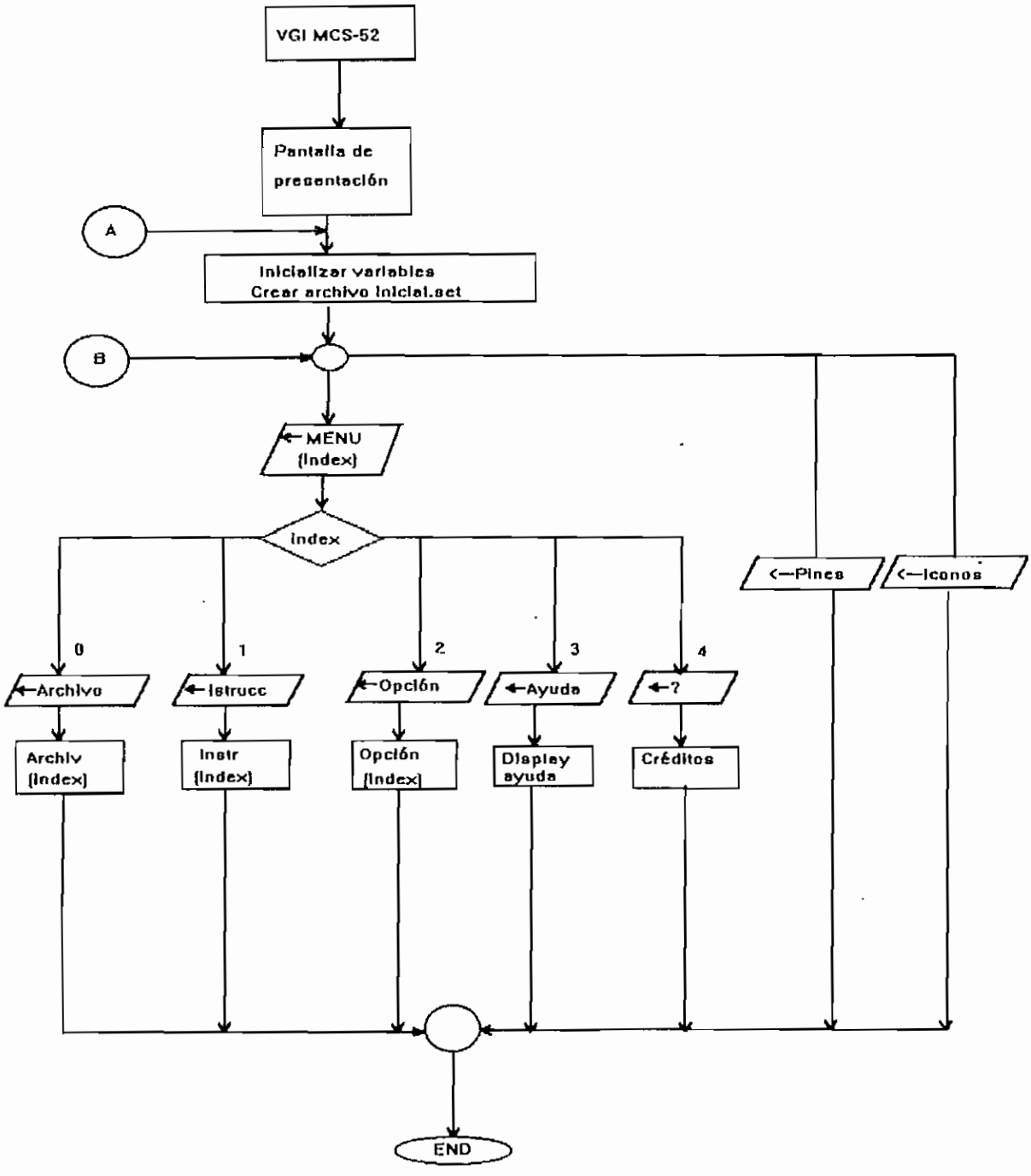


Figura 3.1 Diagrama de flujo (Iniciación del programa)

En este diagrama se describe el proceso que se sigue cuando recién se ingresa al programa que consiste en inicializar las variables y crear el archivo inicial.set una vez que este proceso ha concluido el usuario tendrá la posibilidad de escoger entre la barra de menús, los pines o los íconos.

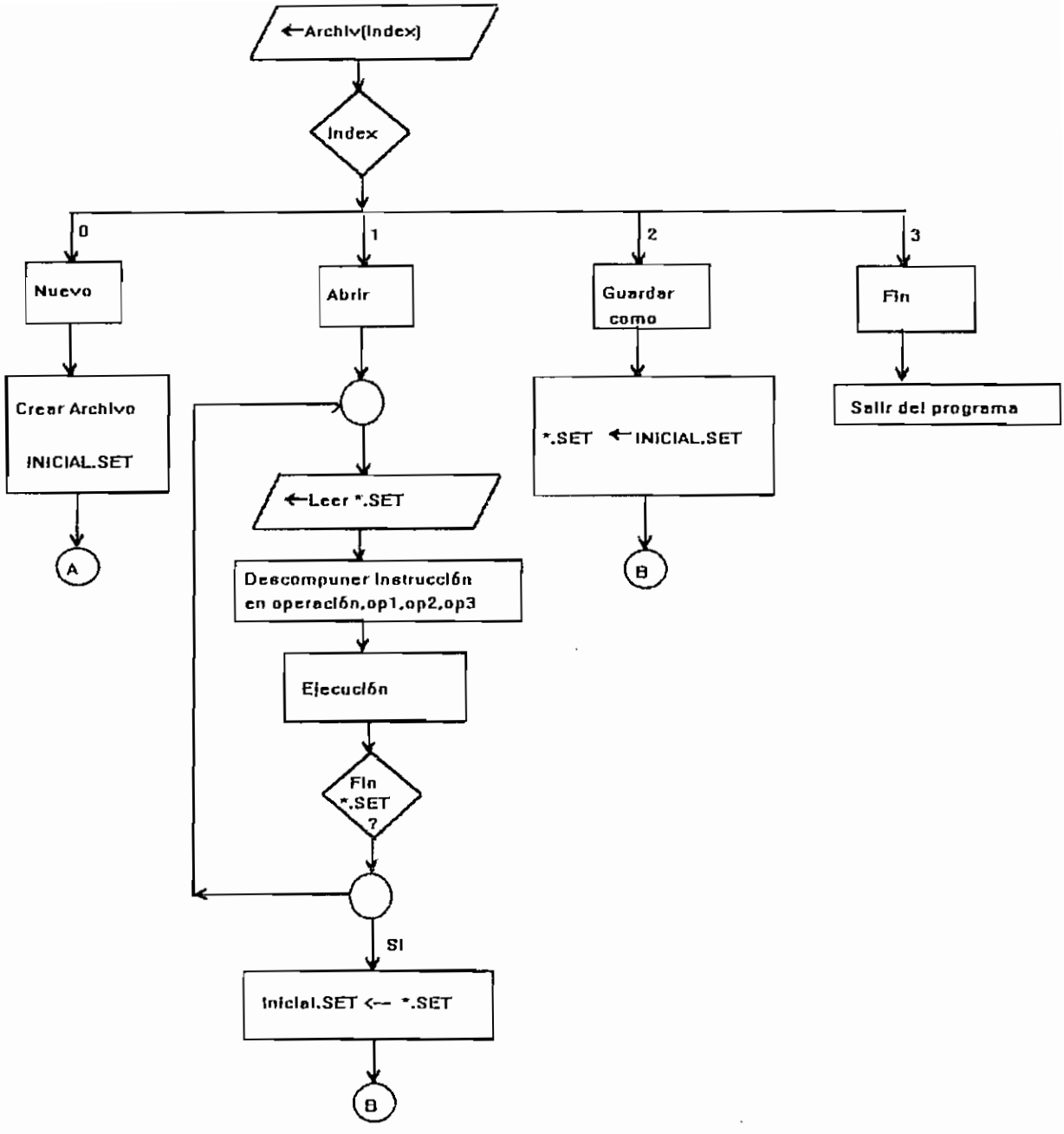


Figura 3.2 Diagrama de flujo (Selección del Menú Archivo)

El *menú archivo* ha sido seleccionado, entonces se podrá escoger entre las opciones Nuevo, Abrir, Guardar como y Fin

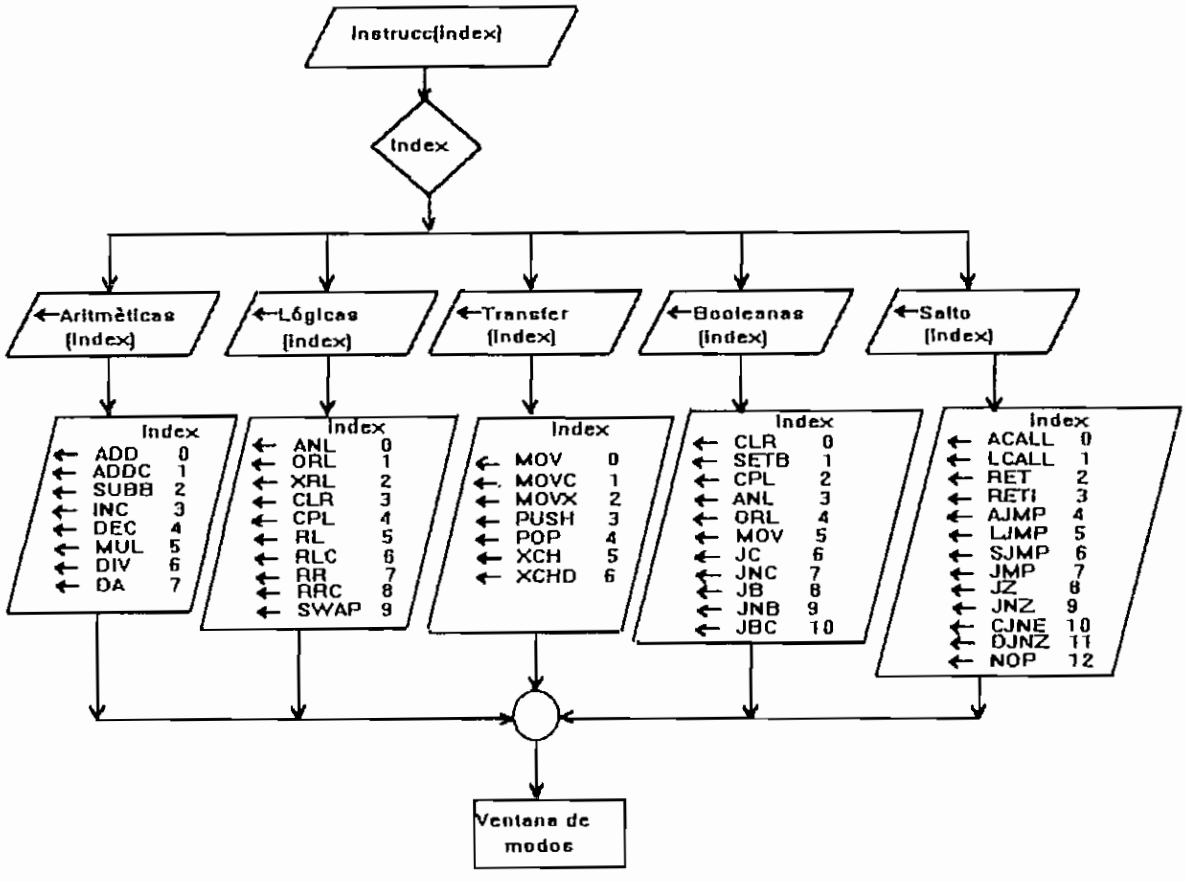


Figura 3.3 Diagrama de flujo (Selección del Menú Instrucciones)

El menú instrucciones ha sido seleccionado, entonces se tendrá la posibilidad de acceder a cualquiera de las instrucciones del MCS-52 a través de submenús, hecho èsto el control del programa pasa a la Ventana de Modos (Figura 3.5).

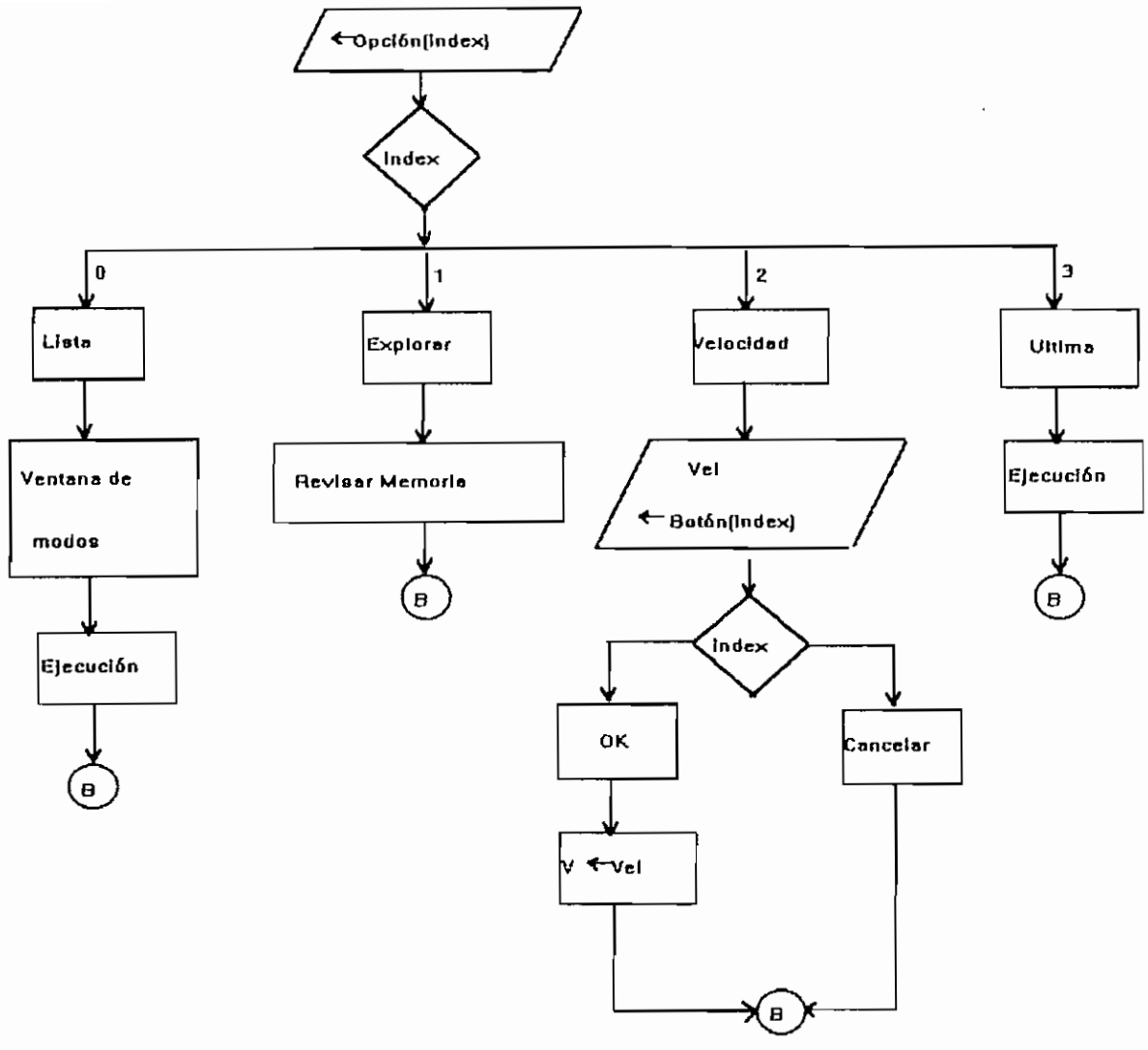


Figura 3.4 Diagrama de flujo (Selección del Menú Opción)

El menú opción ha sido seleccionado, entonces se podrá escoger entre las opciones Lista, Explorar, Velocidad y Ultima, cada uno de los cuales al terminar de ejecutar sus acciones permite regresar al punto B (figura 3.1)

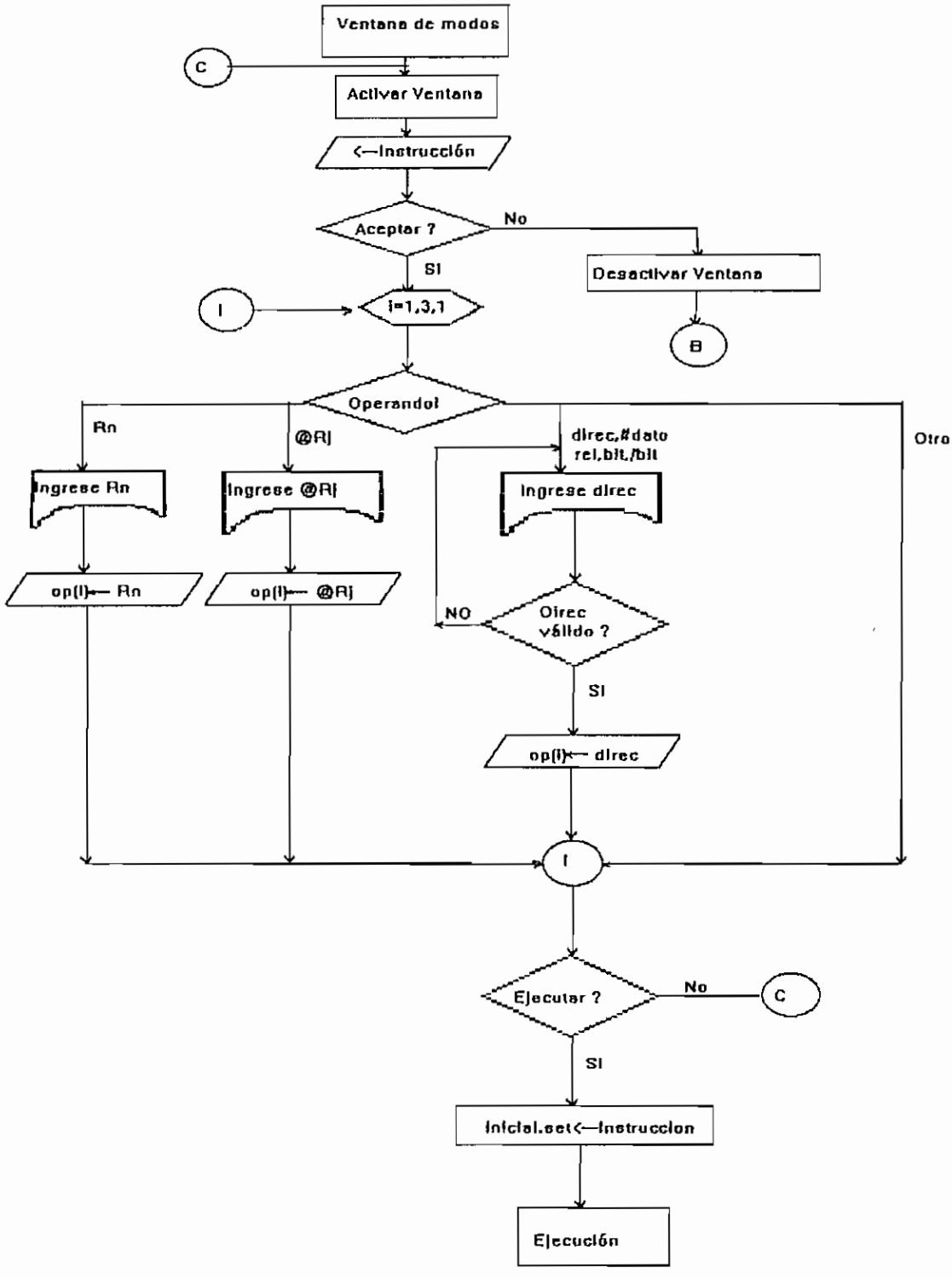


Figura 3.5 Diagrama de flujo (Ventana de modos)

La ventana de modos es desplegada como resultado de haber solicitado alguna instrucción, a través de esta ventana se podrá seleccionar la instrucción definitiva con los operandos que el usuario asigne a fin de que pueda ser ejecutada y guardada en el archivo inicial.set, también tendrá la posibilidad de cancelar.

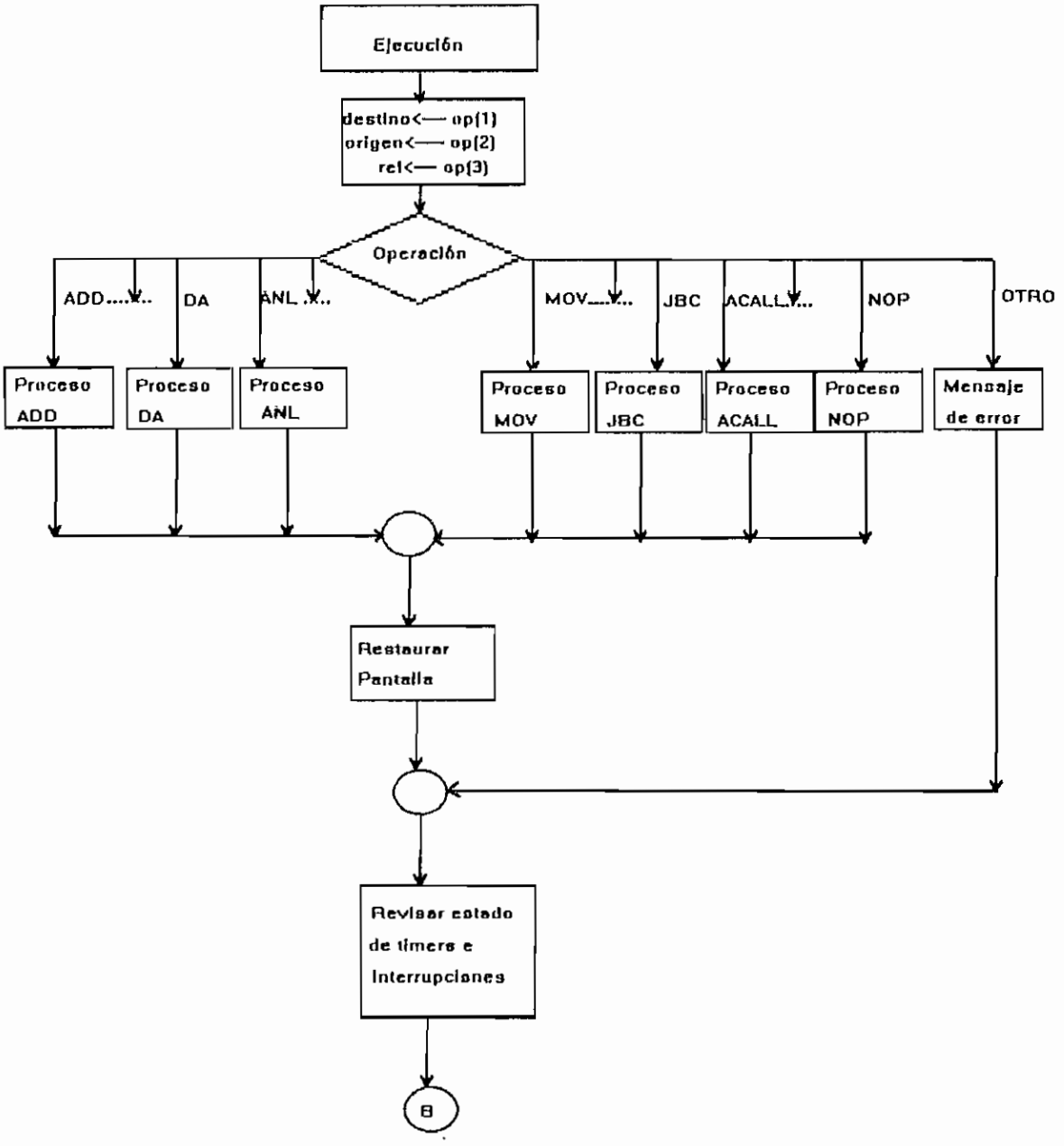


Figura 3.6 Diagrama de flujo (Ejecución)

Dependiendo de la instrucción se ejecuta un determinado proceso (Figura 3.7), el cual contendrá la ejecución analítica y gráfica, una vez terminadas èstas acciones se realiza una revisión del estado de timers e interrupciones, finalmente el usuario podrá acceder otra vez al menú, los pines o los íconos (punto B, Figura 3.1)

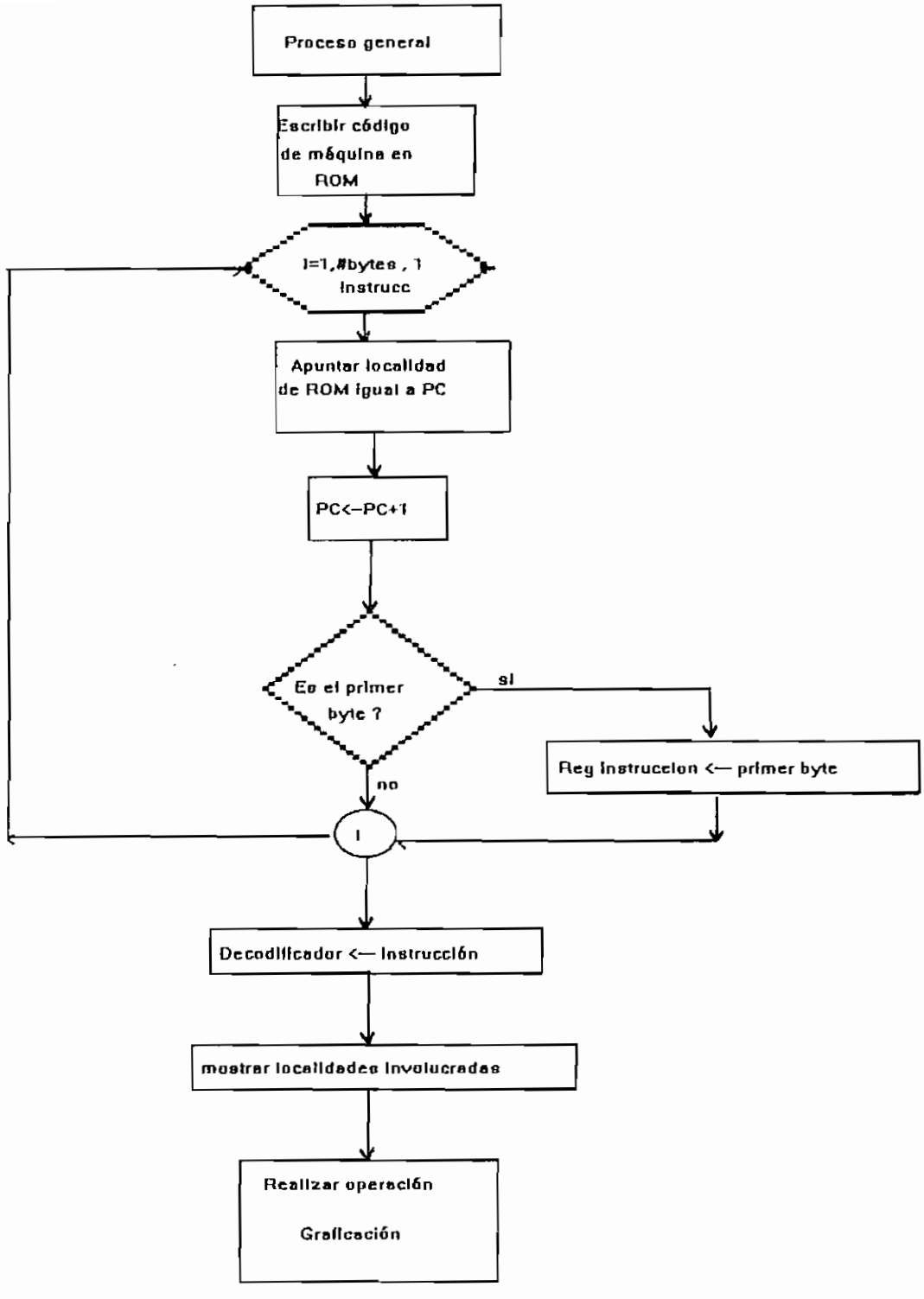


Figura 3.7 Diagrama de flujo (Proceso general)

Proceso general representa las acciones que deben ejecutarse para cada una de las instrucciones, que mas o menos siguen el camino indicado en èste diagrama. Sus acciones consisten en escribir el código de máquina en ROM, apuntar estos códigos, para finalmente, realizar la operación analítica y la graficación de la instrucción

3.2 PROGRAMA PRINCIPAL, PROCEDIMIENTOS

El programa está compuesto por 3 módulos cada uno de los cuales se basa en la implementación de procedimientos que permiten obtener una organización y optimización al máximo del programa total. Cabe señalar que cada módulo siempre tiene un procedimiento llamado Declaration en donde se definen variables que serán usadas en cualquier nivel del módulo.

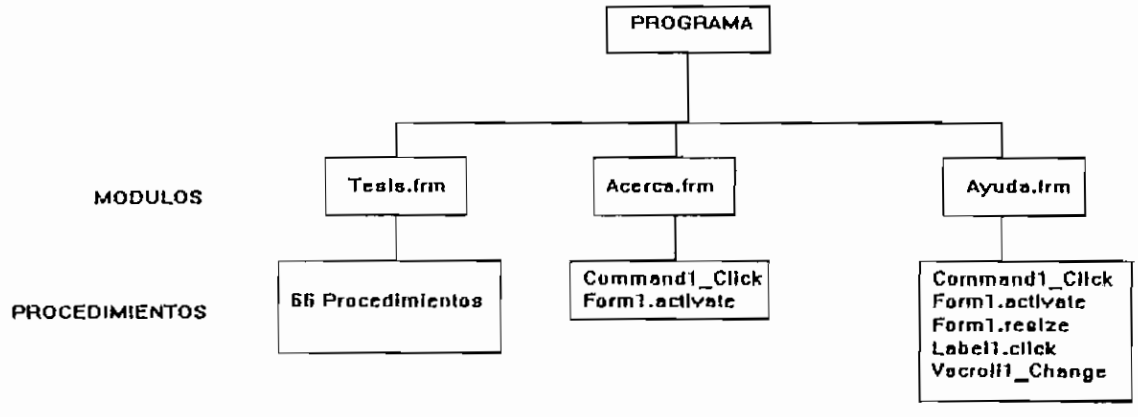


Figura 3.8 Estructura del programa implementado

El módulo Ayuda.frm es el encargado de mostrar la información de ayuda cuando es solicitada, posee 4 procedimientos los cuales ejecutan operaciones relativamente simples como determinar que tema ha sido solicitado, leer el archivo de texto hep4.dat o cambiar el tamaño de la pantalla, por lo cual una explicación mas profunda sería innecesaria.

El módulo Acerca.frm es el encargado de mostrar la pantalla de créditos, de ahí que solo posea 2 procedimientos encargados de hacer visible o invisible dicha pantalla

El módulo Tesis.frm es el más importante de todos ya que es aquí donde se realiza el proceso de visualización de las instrucciones, por lo cual sus procedimientos serán

explicados dada la importancia que tienen. En total en este módulo se implementó 66 procedimientos los cuales de acuerdo a la tarea que realizan se los puede clasificar en los siguientes grupos:

-PROCESOS

-LOCALIDADES

-DATOS

-VISUALIZACION

-DISPLAY

-EVENTOS

3.2.1 PROCESOS

Dentro de este grupo he agrupado todas aquellos procedimientos que son propios de una o un grupo de instrucciones. Estos procedimientos son llamados cuando se llega al punto de la ejecución analítica de la instrucción. y es esta ejecución analítica la parte exclusiva de todos estos PROCESOS ya que en general su estructura está formada por otros procedimientos. Cabe señalar que la ejecución analítica implica también la actualización de banderas.

Los datos de entrada que requieren estos procedimientos son generalmente especificados en un procedimiento llamado MODOS.

Cabe señalar además que muchos de estos procedimientos requieren el ingreso de la variable aumento, la cual es usada para determinar el primer byte del código de instrucción

A continuación se explican cada uno de estos PROCESOS y en general se pondrá énfasis en la manera como se realiza la ejecución analítica.

3.2.1.1 AJUSTE

Procedimiento correspondiente a la instrucción ajuste decimal (DA A). Toma el valor de la variable fuente1, obtiene de ésta el nibble bajo y el nibble alto para analizarlos junto con el carry y el AC y determinar si se realiza el ajuste decimal o no.

3.2.1.2 ARITLOG

Dentro de este procedimiento se lleva a cabo los procesos correspondientes a las instrucciones ADD, ADDC, SUBB, ANL, ORL, XRL.

Este procedimiento en primer lugar determina el modo de direccionamiento que se está utilizando, luego de realizar funciones que tienen en común estas instrucciones mencionadas como la escritura del código de máquina, la presentación de localidades involucradas, determina el tipo de operación a realizar esto es : Suma, Suma con Carry, Resta, etc. ; hecho ésto nuevamente se requieren funciones comunes como es la visualización, actualización de localidades.

3.2.1.3 CEROUNO

Dentro de este procedimiento se lleva a cabo los procesos correspondientes a las instrucciones, CPL, CLR, SETB.

Determina la instrucción a ejecutar, hecho ésto se llama a los procedimientos para escribir el código de máquina, presentación de localidades, ahora se ejecuta la operación, la cual para el caso de CLR y SETB es bastante simple basta con asignar 0 o 1 a la variable RAMi(destino) ; para el caso de CPL que implica complementar el valor de fuente1 se lo realiza bit a bit

3.2.1.4 CJNE

Procedimiento para la ejecución de la instrucción CJNE

Primero se determina el modo de direccionamiento, se escribe el código de máquina entonces se determina si se produce el salto, y también que valor toma el Carry,

3.2.1.5 DJNZ

Procedimiento para la ejecución de la instrucción DJNZ

Se determina el modo de direccionamiento utilizado, escribo el código de máquina. Ahora a la variable RAMi(destino) se asigna el valor igual a fuente1 -1, Se controla si este valor es menor que cero se realiza la corrección correspondiente es decir a RAMi(destino) se asignaría el valor de 255, hecho ésto mediante se determina si se produce el salto

3.2.1.6 INCDEC

Este procedimiento es usado para la ejecución de las instrucciones INC, DEC requiere como dato de entrada el valor de signo,

Para la ejecución de esta instrucción simplemente se suma o resta UNO al valor de fuente 1. El que se sume o reste depende de la variable signo si es 1 suma, y si es -1 resta. Un punto importante en esta instrucción es que si al incrementar el valor es mayor que 255, o que al decrementar resulta menor que cero se debe realizar la corrección correspondiente. Para el caso de INC DPTR es en esencia lo mismo ya que el control se realiza a través de la variable DPTR, es decir se debe corregir si el DPTR excede a 65535

3.2.1.7 MOVER

Procedimiento utilizado para las instrucciones MOV y MOVC

De manera similar a las anteriores ejecuto los procedimientos comunes, se determina el modo de direccionamiento. La ejecución consiste en asignar el valor de fuente2 al valor RAMi(destino)

3.2.1.8 MOVERX

Procedimiento utilizado para la instrucción MOVX

Se realiza una operación de asignación de la siguiente manera

RAMx(fila,columna)=fuente2 Para el caso de escribir en RAM Externa

RAMi(destino)=RAMx(fila,columna) Para el caso de lectura en RAM Externa

3.2.1.9 MULDIV

Procedimiento utilizado para las instrucciones MUL y DIV

Aquí se ejecuta las operaciones multiplicación y división, para determinar que operación se realiza se utiliza un Select Case con la variable MD La actualización de las banderas C y OV se lo realiza mediante el uso de sentencias condicionales

3.2.1.10 PUSHPOP

Procedimiento utilizado para las instrucciones PUSH y POP

Su ejecución analítica está basada completamente en el llamado a los procedimientos INCDEC y MOV.

3.2.1.11 RETORNO

Procedimiento utilizado para las instrucciones RET Y RETI

Su ejecución analítica está basada en el llamado a los procedimientos MOV e INCDEC

3.2.1.12 ROTAR

Procedimiento utilizado para las instrucciones RL, RLC, RR, RRC, SWAP

Su ejecución analítica consiste en: Obtener el equivalente binario de fuente1, almacenado en el vector NUMx(), se requiere un segundo vector NUMy() a cuyos elementos se asignará los bits de NUMX() pero con la posición que implica realizar la rotación o Swap. Por ejemplo para el caso RLC se haría lo siguiente :

```
Carry=Numx(7) :  NUMy(0)=Carry
for i= 1 to 6
        NUMy(i)=NUMx(i-1)
next i
```

Realizada la operación bit a bit se procede a obtener el equivalente decimal del vector NUMy() y almacenarlo en RAMi(destino)

3.2.1.13 SALTAR

Procedimiento utilizado para las instrucciones ACALL, AJMP, LCALL, LJMP, SJMP, JMP, JZ, JNZ

La ejecución de estas instrucciones implica operaciones de asignación, comparación, y realizar una suma con la variable relativo, dicha variable representa un número con signo por lo cual se recurre a sentencias condicionales para determinar si se realiza una suma o una resta.

3.2.1.14 SALTOBIT

Procedimiento utilizado para las instrucciones JC, JNC, JB, JNB, JBC

Su ejecución implica realizar comparaciones para determinar si se produce el salto.

3.2.1.15 XCH

Procedimiento utilizado para las instrucciones XCH y XCHD

Para el caso de XCH se realiza una operación de asignación tal que:

RAMi(destino)=fuente2

RAMi(origen)=fuente1

Para XCHD primero se obtienen los bytes bajos de fuente1 y fuente2 para luego ser intercambiados

3.2.2 LOCALIDADES

A este grupo pertenecen los procedimientos : Previo, Previo2, Previo3, y en términos generales se puede decir que estos procedimientos son los encargados de actualizar la presentación de los datos es decir los contenidos de las celdas de las grillas que se muestran en la pantalla de ejecución, dichas grillas pueden representar los bloques de memoria, o los cuadros con valores de timers, banderas, etc.

A continuación se explica la función de cada uno

3.2.2.1 PREVIO

Este procedimiento es utilizado cuando se presenta el bloque de localidades involucradas de RAM Interna, y su ejecución se basa en el llamado al procedimiento PREVIO2, que seguidamente se explica Las variables requeridas son los datos que sobre las celdas de las grillas se mostrarán.

3.2.2.2 PREVIO2

Este procedimiento requiere como datos de entrada el número de la grilla, las celdas que serán afectadas las cuales se especifican a través de los valores de columna inicial (SSC), columna final (SEC), fila inicial (SSR), fila final (SER); y los datos que serán puestos en estas celdas los cuales se almacenan en la variable mens$

Como resultado de ejecutar este procedimiento se visualizará en las grillas los valores que uno desee que muestre.

3.2.2.3 PREVIO3

Este procedimiento es similar a Previo2 con la diferencia de que el dato especificado a través de la variable mens afecta una sola celda de la grilla dada por el valor de Indice, la celda que será afectada se especifica a través de los valores de columna inicial (SSC), fila inicial (SSR)

3.2.3 DATOS

Dentro de esta categoría están procedimientos que transforman datos de un formato a otro o interpretan ciertos datos leídos. En este grupo tenemos los siguientes:

3.2.3.1 BIN

Permite la transformación de un número decimal a binario,

Esta función requiere como datos de ingreso el número decimal el cual es almacenado en la variable Numero, y el nombre del vector donde se almacenará el equivalente binario. el cual dentro del procedimiento será almacenado en el vector cod()

3.2.3.2 BUSCAR

Este procedimiento es el encargado de encontar la fila y columna dentro de las matrices definidas como RAM externa y ROM externa, equivalente a la dirección de memoria a la que se quiere acceder.

3.2.3.3 DATOS2

Es un procedimiento de tipo función y permite representar a cualquier número decimal menor o igual a 255 como un número hexadecimal de 1 byte Por Ejemplo

datos2(5)——resultado———> 05

3.2.3.4 DATOS4

Permite representar a cualquier número decimal menor o igual a 65535 como un número hexadecimal de 2 bytes. Por Ejemplo

datos4(2)——resultado———> 0002

3.2.3.5 DECI

Permite obtener el equivalente decimal de un número binario

Como datos requiere el vector que contiene el número binario y el nombre de la variable donde se almacenará el equivalente decimal,

3.2.3.6 DECIBOOL

Aquí se determina si el formato de los operandos es del tipo bit o /bit. La variable oper123$ es cargada con el operando a analizar ; en la variable clase se carga el valor del byte al que pertenece dicho bit y en fuentes se carga la posición del bit dentro del byte. Dentro de este procedimiento también se carga el valor de la variable cmbyte que representa el código de máquina para dicho bit

3.2.3.7 DECIFRAR

Aquí se determina si el formato de los operandos es decimal, etiqueta o hexadecimal. La variable oper123$ es cargada con el operando a analizar ; en clase se carga el valor del operando en formato decimal, y en fuentes se carga el valor RAMi(clase) solo si clase es menor a 256 caso contrario es puesto a cero.

3.2.3.8 EJECUCION

Este procedimiento realiza la función de enviar cada instrucción al proceso que le corresponda (numeral 3.2.1 procesos) o tambièn puede darse el caso que si se lee de un archivo una instrucción incorrecta, entonces despliega un mensaje de error

3.2.3.9 HEXDEC

Permite obtener el equivalente decimal de un número hexadecimal

Como datos requiere el número hexadecimal (Numh) que será transformado a decimal (Numd)

3.2.3.10 LSB

Este procedimiento es del tipo función y permite obtener el nibble bajo de un byte

Ejemplo:

LSB(46H)———Resultado———>6

3.2.3.11 MODOS

Aquí se analizan uno a uno los operandos de la instrucción. dando como resultado la obtención de la variable MD, origen, destino, rel, fuente1, fuente2, fuenter.. Este procedimiento constituye un filtro ya que de encontrarse un operando no valido envía un mensaje de error

3.2.3.12 MUESTREO

Este procedimiento se ejecuta luego de haber visualizado la instrucción y su función es revisar el estado de los timers e interrupciones para que en el caso de que èstos estuvieran activados ejecutar las acciones como actualización de ventana de timers o ventana de interrupciones.

3.2.3.13 PAUSA

Este procedimiento se encarga de temporizar todos los eventos que tienen que ver con el movimiento de flechas o los diferentes efectos que se observan en la visualización, lo cual lo realiza a través de un lazo como el que se indica a continuación.

```
while timer1.interval=1000*limite
wend
```

Entonces saldrá del lazo solo cuando el intervalo del timer se consuma

El valor de la variable limite expresa en segundos el tiempo que consumirá este procedimiento

3.2.3.14 PCS

Este procedimiento cambia la apariencia del valor del PC, que consiste en insertar 2 espacios en blanco entre cada cifra del PC

3.2.3.15 RESETEO

Su función es cargar los valores correspondientes a un RESET

3.2.3.16 SEPARAR

Su función consiste en dividir la instrucción en operación, operando1, operando2, operando3 cargados en las variables operacion, op(1), op(2), op(3) respectivamente

3.2.4 VISUALIZACION

En este grupo están todos los procedimientos que tienen que ver con la visualización gráfica de la ejecución de cualquier instrucción. Estos son : Códigos, EjecVisual, Flechas, Moverdato, Movi, Movi2

A continuación se explica cada uno de èstos

3.2.4.1 CODIGOS

Este procedimiento es el encargado de escribir el código de instrucción en ROM, luego procede a apuntar las localidades que contienen dicho código al mismo tiempo que carga el registro de instrucción, decodificador e incrementa el contador de programa PC. Como datos de entrada requiere la variable total la cual indica el # de bytes que tiene el código de instrucción, lo cual determina el límite del lazo de repetición que se requiere para las acciones indicadas anteriormente

3.2.4.2 EJECVISUAL

Es este procedimiento el que está encargado de dirigir las acciones que requiere la visualización de la instrucción, aquí se ordena que se ejecute los diferentes tipos de movimiento que se observan cuando el programa está ejecutando. Como dato de entrada requiere el valor de la variable index sirve para seleccionar los movimientos que requiere cada instrucción.

3.2.4.3 FLECHAS

Este proceso se encarga de hacer visible la flecha que se requiera con su forma correcta. Los datos de entrada que requiere son:

flecha: identifica la flecha a utilizar

direc: indica en que dirección se quiere que apunte la flecha

letra: es el mensaje que lleva la flecha durante su movimiento

3.2.4.4 MOVERDATO

Este proceso es requerido cuando se quiere trasladar algún dato desde la ROM a la RAM Interna, por ser muy frecuente su uso se lo implementó cuyo funcionamiento se

basa en llamar a los procedimientos Movi y Flechas. Como datos de entrada requiere las coordenadas de la posición de origen y la posición final de la flecha especificadas a través de L1,L2,T1,T2, ; y la variable labell que indica el mensaje que llevará la flecha durante su movimiento.

3.2.4.5 MOVI

Este proceso genera el movimiento de una flecha.. Los datos de entrada requeridos son :

Inicio y fin : indican los puntos entre los cuales se va a desplazar la flecha

subir: si es cero la flecha se mueve horizontalmente y si es uno se mueve verticalmente

indic: identifica cual flecha se va a mover

3.2.4.6 MOVI2

Este proceso genera el movimiento simultáneo de 2 flechas.. Los datos de entrada requeridos son :

Inicio y fin : indican los puntos entre los cuales se van a desplazar las 2 flechas

subir: si es cero las flechas se mueven horizontalmente y si es uno se mueven verticalmente

indic1 e indic2 : identifican las flechas que se van a mover

3.2.4.7 REGRESO

Su función es llevar al fondo normal de pantalla, es decir ocultar los elementos gráficos que no sean parte del fondo normal de pantalla

3.2.5 DISPLAY

En este grupo constan los procedimientos que realizan el display de algún tipo de dato al usuario, pertenecientes a este grupo tenemos los procedimientos: Banderas, Ingresos, Mensajes, Modo1

A continuación se procede a describirlos

3.2.5.1 BANDERAS

Es el encargado de actualizar los valores del Carry, OV, AC, P que se muestran en la caja de Banderas

3.2.5.2 INGRESOS

Es el encargado de controlar que ventana se hace visible cuando se requiere el ingreso de algún operando

3.2.5.3 MENSAJES

Su función es colocar los comentarios que se observan durante la ejecución en la caja de mensajes, lo cual lo realiza a través de la variable lugar que indica la posición dentro del archivo de datos Mensajes.dat el comentario que requiere determinada instrucción.

3.2.5.4 MODO1

Este proceso hace el display de las instrucciones con todos los modos de direccionamiento disponibles, para lo cual requiere el ingreso de las variables saltos y total que determinan el bloque de datos a ser leído desde el archivo ARIT.DAT ; nombre$ identifica uno de los títulos dentro de la ventana en los cual aparecen estos

3.2.6 EVENTOS

En este grupo tenemos aquellos procedimientos que son invocados automáticamente ya sea en respuesta a un evento generado por el usuario, a algún código del programa o que son activados por el sistema.

En el programa se han implementado 6 tipos de eventos:

Evento Click Son aquellos que ocurren cuando el usuario presiona y suelta el mouse sobre un objeto ; o cuando el valor de un control ha cambiado

Evento load Ocurre cuando un formulario (form) es cargado

Evento Keypress Ocurre cuando se presiona una tecla

Evento mouse move Ocurre cuando el puntero del mouse es movido sobre algún control

Evento Change Ocurre cuando el valor del control ha sido cambiado

Evento Timer Ocurre cuando se ha consumido el intervalo de tiempo asignado a un Timer

Con lo dicho a continuación se explican dichos eventos

3.2.6.1 ARCHIVO_CLICK

Todos los ítems del menú Archivo forman parte de un arreglo de controles los cuales están identificados por el siguiente nombre Archivo(index), donde index puede ser un número entero, entonces cuando el evento click se genera el tipo de acciones a ejecutarse para cada ítem se lo determina a través de un select case y la variable index

Las acciones desarrolladas para cada ítem son:

Item Nuevo: Produce un llamado al procedimiento reseteo, y abre el archivo de inicio donde se guardarán todas las instrucciones ejecutadas.

Item Abrir Muestra un caja de diálogo donde el usuario escoge el archivo que desee, este archivo se procede a abrirlo y leerlo para que a través de un lazo tipo While Wend ejecute todas las instrucciones leídas, dicho lazo será finalizado al encontrarse el fin de archivo o cuando el usuario pulse la tecla "ESC"

Item Guardar como Despliega una caja de diálogo donde el usuario puede digitar el nombre con el que desee guardar su sesión de trabajo, una vez guardado el archivo el usuario puede continuar trabajando con el programa y todas las instrucciones que se ejecuten desde este momento tendrán la posibilidad de almacenarse en el mismo archivo.

Item fin Ejecuta la sentencia END la cual ocasiona el salir del programa.

3.2.6.2 ARIT_CLICK

Todos los ítems del submenú aritméticas están identificados por el nombre Arit(index), y al ejecutarse este evento genera un llamado al procedimiento MODO1 seteado con sus respectivas variables dependiendo del ítem escogido.

3.2.6.3 BARRA_CLICK

Con el nombre barra(index) están identificados los nombres de la barra del menú principal èstos son : Archivo, Instrucciones, Opciones, Ayuda, ?. Al ser activado este evento se producen acciones que solo tienen que ver con Ayuda y ?, las cuales consisten en activar los módulos donde están editadas la ayuda y los créditos del programa. Esta activación se la realiza utilizando la sentencia Show

3.2.6.4 BOOLEANAS_CLICK

A este evento pertenecen los Items del submenú Booleanas los cuales están identificados como Booleanas(index), y las acciones que se ejecutan están basadas en el llamado al procedimiento MODO1.

3.2.6.5 COMBO2_CLICK

Su acción consiste en la habilitación del botón Aceptar

3.2.6.6 COMMAND1_CLICK

A este evento corresponden las acciones ejecutadas para el arreglo de controles Command1(index), entonces cuando este evento se activa se determina el botón pulsado provocando la ejecución de las siguientes acciones.

Command1(0)

Corresponde al botón Aceptar y sus acciones consisten en: Almacena el dato escogido de la caja de selección de modos en la variable A$, seguidamente llama a los procedimientos SEPARAR e INGRESOS

Command1(1)

Corresponde al botón Cancelar y sus acciones consisten en: Deshabilitar la ventana de modos quedando en posibilidad de escoger otra opción del menú, activar los Iconos o escribir en los pines

Command1(2)

Corresponde al botón Ejecutar y sus acciones consisten en: Oculta la ventana de modos, escribe la instrucción a ejecutar en el archivo inicial.set y finalmente llama a los procedimientos Ejecución y Muestreo

3.2.6.7 COMMAND2_CLICK

Los botones OK y Cancel de la ventana de velocidad son parte del arreglo de controles Command2(index), cuyos index son 0 y 1 respectivamente. Cuando este evento se activa solo el botón OK tiene acción la cual consiste en cargar la variable paso con el valor de la barra de velocidad, y la variable espera determinada a partir de paso

3.2.6.8 COMMAND3D1_CLICK

Dentro del arreglo de controles command3d1 están contenidos los Iconos que aparecen permanentemente en la pantalla cuyas acciones consisten en mostrar las ventanas de los timers, interrupciones y visualización del contenido de un archivo.

3.2.6.9 COMMAND4_CLICK

Dentro de este arreglo de controles identificados como command4(index) se indica a continuación las acciones ejecutadas para cada uno

Command4(0)—>Command4(7)

Corresponde a los botones R0 a R7 respectivamente de la ventana Ingrese Rn, su acción consiste en cargar la caja que indica el Rn escogido pero no definitivo

Command4(8)

Corresponde al botón OK de la ventana de Ingrese Rn y su acción consiste en leer el Rn escogido definitivamente, cargar este Rn en la variable op(i) que corresponda y llamar al procedimiento INGRESOS

Command4(9), Command4(10)

Corresponde a los botones @Ro y @R1 de la ventana Ingrese @Ri, su acción consiste en cargar la caja que indica el @Ri escogido pero no definitivo

Command4(11)

Corresponde al botón OK de la ventana de Ingrese @Ri y su acción consiste en leer el @Ri escogido definitivamente, cargar este @Ri en la variable op(i) que corresponda y llamar al procedimiento INGRESOS

Command4(12)

Corresponde al botón OK de la ventana de Ingresos manuales de datos y su acción consiste en leer el dato escrito por el usuario, determinar si es un dato válido con la ayuda de los procedimientos Decifrar y Decibool, de ser válido cargar este dato en el operando op(i) que corresponda y llamar al procedimiento ingresos, caso contrario enviar un mensaje de error y esperar hasta que el dato ingresado sea válido o se escoja el botón Cancele.

3.2.6.10 FORM_KEYPRESS

Se ejecuta al pulsar la tecla **ESC** permitiendo que cuando se ha usado la opción Abrir del menú Archivo, el proceso de lectura de las instrucciones desde el archivo abierto concluya.

3.2.6.11 FORM_LOAD

Se ejecuta al iniciar el programa y en este evento se fijan las propiedades de los diferentes controles usados, todas las variables utilizadas son puestas a sus valores por defecto, crear el archivo inicial,set y llama al procedimiento reseteo

3.2.6.12 GRID2_CLICK

Se encarga de poner al frente el bloque de memoria que reciba el click

3.2.6.13 GRID2_MOUSEMOVE

Este evento permite que el formato de presentación de las grillas correspondientes a RAM interna no sea alterado por algún movimiento no deseado del mouse sobre dichas grillas

3.2.6.14 LOGICAS_CLICK

Todos los ítems del submenú Lógicas están identificados por el nombre Logicas(index), y al ejecutarse este evento genera un llamado al procedimiento MODO1 seteado con sus respectivas variables dependiendo del ítem escogido.

3.2.6.15 OPCIONES_CLICK

Todos los ítems del menú Opciones están identificados por el nombre Opciones(index). A continuación se indican las acciones que genera este evento dependiendo del ítem escogido.

Opciones_click(0)

Equivale al ítem Lista, y sus acciones consisten: en cargar en la caja de modos todo el Set de Instrucciones lo cual lo realiza a través de la lectura del archivo arit.dat, hecho èsto muestra la ventana de modos permitiendo así que el control pase a cargo de los botones de esta ventana

Opciones_click(1)

Equivale al ítem Explorar, y su acción consiste: en activar las barras de desplazamiento para poder revisar los contenidos de los bloques de memoria.

Opciones_click(2)

Equivale al ítem velocidad, y su acción consiste: en mostrar la ventana de velocidad.

Opciones_click(3)

Equivale al ítem Ultima, y su acciones consisten: en cargar la variable A$ con el valor de instrucciion$ para seguir con el llamado a los procedimientos SEPARAR y EJECUCION

3.2.6.16 PINES_CLICK

A este evento corresponden las acciones ejecutadas para el arreglo de controles pines(index)

Estas acciones consisten en determinar si se producen flancos en pines que pueden activar interrupciones o habilitar timers o si dichos flancos representan pulsos que incrementan los registros de cuenta. Y un caso especial que es el del pin(9) que corresponde al reset el cual simula un pulso high-low-high y carga los valores correspondientes a un Reset.

3.2.6.17 SALTO_CLICK

Todos los ítems del submenú salto están identificados por el nombre salto(index), y al ejecutarse este evento genera un llamado al procedimiento MODO1 seteado con sus respectivas variables dependiendo del ítem escogido.

3.2.6.18 TIMER1_TIMER

Este evento es generado cuando el tiempo del intervalo asignado al timer1 ha sido consumido y la acción que se genera es deshabilitar el timer1

3.2.6.19 TIPOS_CLICK

Todos los ítems del menú Instrucciones están identificados por el nombre tipos(index), y su acción consiste en escribir en la ventana de modos el tipo de instrucción seleccionada.

3.2.6.20 TRANS_CLICK

Todos los ítems del submenú Transferencia están identificados por el nombre Trans(index), y al ejecutarse este evento genera un llamado al procedimiento MODO1 seteado con sus respectivas variables dependiendo del ítem escogido.

3.2.6.21 VSCROLL1_CHANGE

A este evento corresponden las acciones ejecutadas para el arreglo de controles Vscroll1(index)

Su acción consiste en ir mostrando las localidades de memoria que correspondan al valor de las barras de desplazamiento

3.3 MANUAL DEL USUARIO

VGI MCS-52 versión 1.0 fue desarrollado en el lenguaje VISUAL BASIC versión 3.0, y para ser utilizado se recomienda instalarlo de la siguiente manera:

Crear el directorio C:\VGI

Copiar a este directorio todos los archivos de extensión .dat y el archivo VGI.EXE.(estos archivos requerirán alrededor de 200Kbytes en el disco duro). En caso de no tener instalado Visual Basic 3.0 en su computador deberá copiar los archivos de extensión VBX a la ruta C:\windows\system. (Los archivos VBX requieren 453KBytes en el disco duro).

Realizadas estas operaciones podrá correr VGI.EXE o instalarlo como una aplicación mas de Windows en cuyo caso tendrá el siguiente ícono asignado.

Figura 3.9 Icono

Con respecto al equipo de trabajo se recomienda:

Un computador personal 80386 o superior

Un ratón

Sistema Windows mínimo 3.1

Tarjeta gráfica VGA preferiblemente

Mientras el programa está siendo cargado se verá una pantalla de presentación, la cual desaparecerá automáticamente cuando el programa estè disponible al usuario.

Una vez que se ha ingresado al ambiente de VGI MCS-52 la pantalla constará de una barra de menús, los cuales están compuestos de varios submenús, 3 íconos, una caja de texto la cual desplegará diferentes mensajes y ocupando casi en su totalidad la

pantalla donde se ejecutarán las instrucciones. A continuación se describen los elementos indicados.

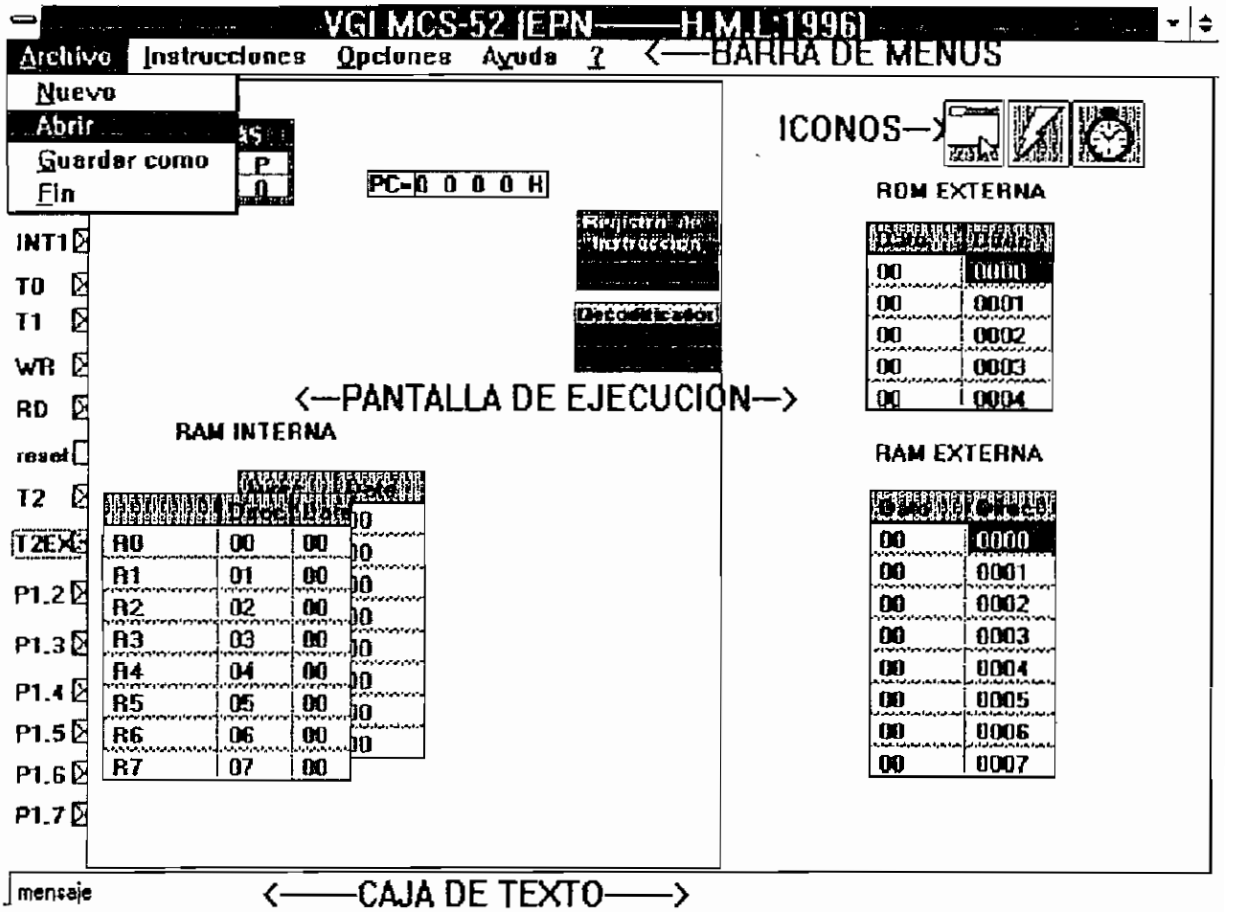


FIGURA 3.10 Elementos de la Pantalla de VGI MCS-52

La barra de menús

Esta barra consta de los menús : Archivo, Instrucciones, Opciones, Ayuda, ? ; a los cuales se puede acceder como es habitual por medio del mouse o a través de las teclas alt + la letra que aparece subrayada. Por ejemplo para acceder al menú Archivo, se hace un click en èste o se presiona ALT A

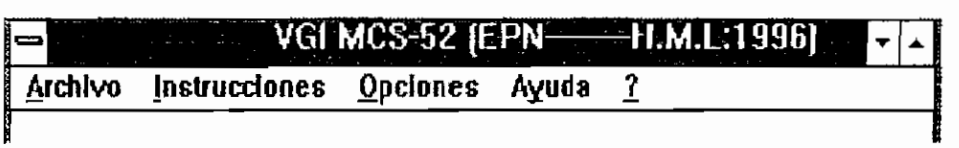


Figura 3.11 Barra de Menús

A continuación se describe cada uno de los menús mencionados

Menú Archivo

Este menú consta de 4 elementos, los cuales se describen seguidamente

Nuevo: Inicia una nueva sesión de trabajo con los valores correspondientes a un RESET

Abrir: Permite ejecutar un grupo de instrucciones, las cuales son leídas desde un archivo con formato ASCII. Este archivo puede ser seleccionado por medio de una caja de diálogo que asoma al escoger esta opción. Por defecto se buscan los archivos de extensión. SET. Una vez que se estén ejecutando las instrucciones leídas desde el archivo se puede salir del proceso pulsando la tecla "ESC".

Cabe señalar que si durante la lectura del archivo se detecta una instrucción incorrecta un mensaje de error como el siguiente es desplegado.

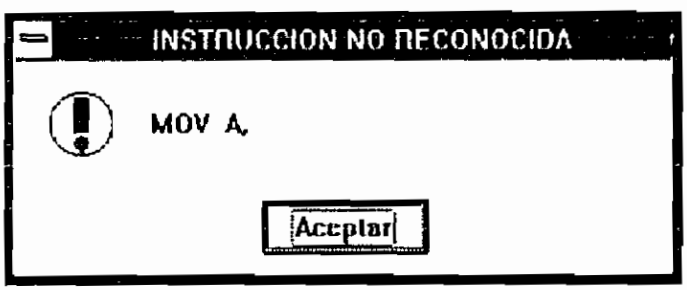


Figura 3.12 Mensaje de Error

Guardar como: Al igual que en abrir despliega un cuadro de diálogo que permite guardar toda la sesión de trabajo con un nombre que asigne el usuario. Los archivos generalmente se guardan con extensión. SET

Fin: Abandona el programa VGI MCS-52

Menú Instrucciones

Este será sin duda alguna el menú que mas se requiere, ya que éste nos proporciona el acceso al Set de Instrucciones de una manera ordenada y amigable.

En primer lugar al accesar a este menú aparece un submenú con los nombres de los tipos de instrucciones disponibles esto es: Aritméticas, Lógicas, Transferencia, Boolenas y Salto. Ahora escogiendo estos ítems se accede a otros submenús los cuales contienen los nombres de las operaciones que ejecuta cada instrucción. Por ejemplo si escogemos el ítem Transferencia, el nuevo submenú constará de los siguientes elementos : MOV, MOVC, MOVX, PUSH, POP, XCH, XCHD.

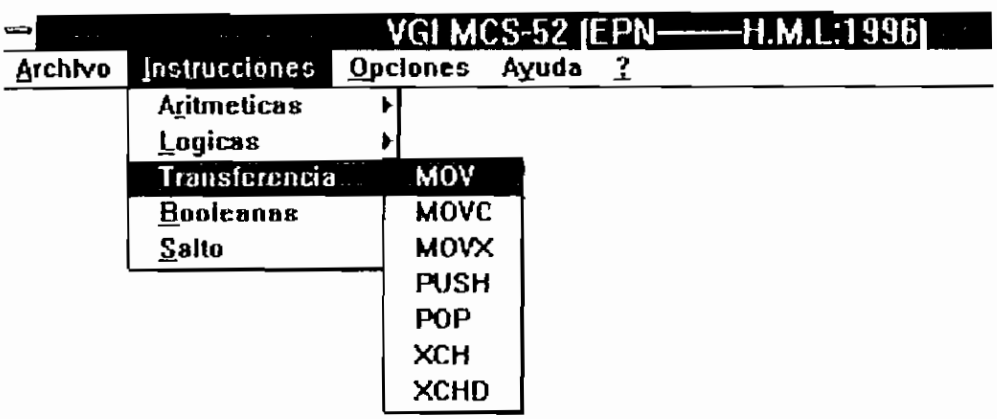


Figura 3.13

, finalmente escogiendo cualquiera de estos ítems se despliega la ventana de modos donde constará el tipo de instrucción, el nombre de la instrucción, una caja de selección con los modos de direccionamiento disponibles, de la cual se deberá escoger la instrucción que se requiera haciendo un click ; esta caja de selección dispondrá si es necesario de una barra de desplazamiento. Escogida la instrucción se deberá presionar ACEPTAR, con lo cual de ser necesario aparecerán subventanas de donde se escogerán o ingresarán los operandos. Una vez escogido o ingresado los operandos estará disponible el botón Ejecutar el cual dará la orden de ejecución. Siempre se tiene la opción de cancelar la cual desaparecerá la ventana de instrucción. De darse el caso que ya no se desee la instrucción que se escogió sino otra es suficiente volver hacer un click en la caja de selección y pulsar otra vez ACEPTAR.

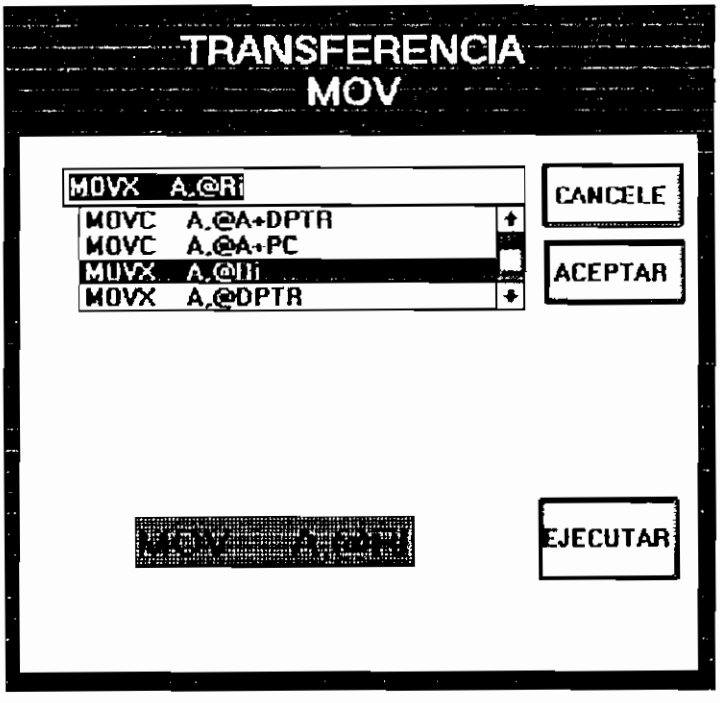


Figura 3.14 Ventana de modos

Seguidamente se muestran las ventanas para el ingreso de datos.

Esta ventana tendrá la misma forma para el caso de el ingreso de #dato,dirección origen, dirección destino, Relativo, Bit, /Bit.. Lo único que cambiará será los títulos de la ventana. El valor debe ser ingresado de forma manual, respetando los formatos de datos soportados por VGI MCS-52

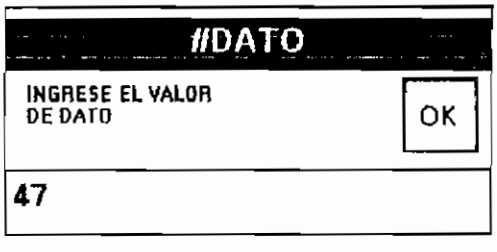


Figura 3.15 Ventana para ingreso de #dato

Para estas 2 ventanas basta con hacer un click en el registro que se desee y luego pulsar OK

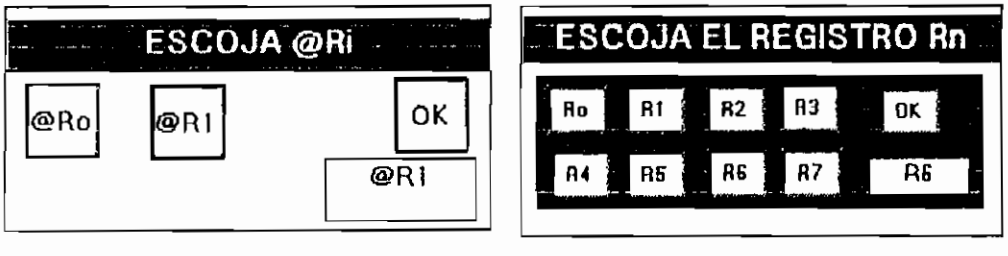


Figura 3.16 Ventanas para escoger @Ri y Rn

Menú Opciones

Dispone de los siguientes elementos.

Lista: Despliega un cuadro donde constan todo el Set de instrucciones. Seleccionada la instrucción se debe seguir el mismo proceso que el indicado anteriormente. Esta opción esta dirigida a un usuario con un mayor conocimiento respecto a la clase de instrucciones que se dispone.

Explorar: De querer indagar el contenido de las memorias no hay mas que escoger esta opción la cual despliega barras de desplazamiento pegadas a los bloques de memoria, su uso resulta evidente, simplemente hacer un click en estas barras buscando la dirección cuyo contenido se desee averiguar.

Velocidad: Nos permite alterar la rapidez con que se visualizan la ejecución de las instrucciones, para lo cual aparece una ventana en la cual se selecciona la velocidad haciendo click en la barra de velocidad, fijada la velocidad se debe presionar el botón OK, o de querer anular la operación el botón CANCEL

Figura 3.17 Ventana para selección de velocidad

Ultima: Repite la ejecución de la última instrucción seleccionada. Esta opción es muy útil ya que se supone que el usuario está en un proceso de aprendizaje, con lo cual podrá revisar la ejecución de una instrucción cualquiera una y otra vez simplemente escogiendo esta opción.

Menú Ayuda

Despliega una ventana que da acceso a información sobre varios tópicos a los que se puede acceder a través del botón temas.

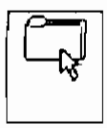
Menú ?

Simplemente despliega una ventana con los créditos del programa, para que èsta desaparezca basta escoger el botón OK.

Iconos

Haciendo un click en èstos se accede a las respectivas funciones que realizan

A continuación se describen individualmente



Permite revisar el contenido del archivo que está siendo leído cuando se ha usado la opción Abrir del menú Archivo.

Despliega un cuadro con información acerca de todas las fuentes de interrupción, sus banderas (FLAGS), con sus respectivos valores.

Despliega un cuadro con información acerca del los timers 0, 1 y 2. Muestra el contenido de los registros de cuenta (TH, TL), las banderas (TF0, TF1, TF2), y EXF2 solo para el Timer 2.

Si ya no requiriera el uso de las funciones de estos íconos, sólo debe hacer un nuevo click en èstos.

PINES

Los pines a los que se puede acceder son los correspondientes a los Puertos 3 y 1 (viendo de arriba hacia abajo). De tener algún nombre especial el Pin aparecerá dicho nombre, de no ser así se presentará como Puerto.#Pin. Ej P1.7.

Si el pin se muestra con una "X" equivale a un 1L, por el contrario si el pin está vacío representa un 0L.

Para cambiar el valor de algún pin no hay mas que hacer "click" en el que se desee

Ademas se tiene el pin correspondiente al RESET, el cual está claramente indicado

FORMATO PARA INGRESAR DATOS

La entrada de datos puede soportar los siguientes formatos :

DECIMALES ó HEXADECIMALES

En cuyo caso se deberá respetar el siguiente formato

DECIMAL: 1, 4, 63000, etc.

HEXADECIMAL: 10H, FAH, BA11H, etc.

ETIQUETAS

* Para el caso de los registros del SFR se puede ingresar el nombre del registro

Ej A, TCON, DPL, DPH, etc.

* Para el caso de instrucciones BOOLEANAS, se puede ingresar de la siguiente manera:

Ej: 20H o 32 equivale al bit 0 del byte 24

43H o 67 equivale al bit 3 del byte 40

* Para los registros del SFR direccionables bit a bit

Ej: A.4 equivale al bit 4 del acumulador

TCON.1 equivale al bit 1 del registro TCON

* Para el caso de bits que tienen algún nombre especifico

Ej: TF0 es el bit 5 del registro TCON

De no respetarse este formato de datos un mensaje de error como el siguiente es desplegado

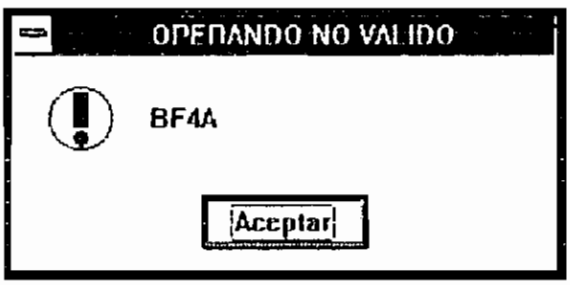


Figura 3.18 Mensaje de error

CAPITULO 4

RESULTADOS Y CONCLUSIONES

RESULTADOS

El programa disponible al usuario trabajará de la siguiente manera:

Permanentemente se tendrá una pantalla como la indicada en la figura 4.1

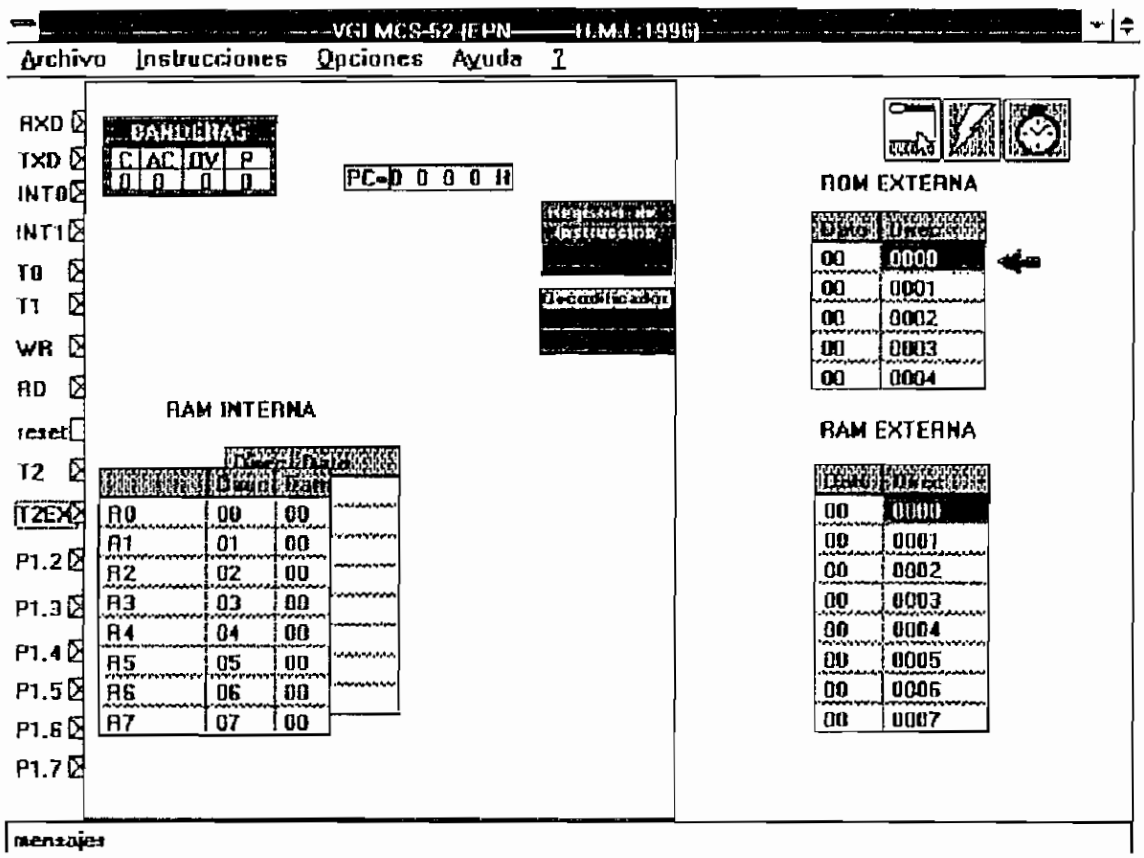


Figura 4.1

donde se observa:

El MCS-52 con los pines correspondientes a los Puertos 1 y 3, en su interior constan : Caja de banderas (C, AC, OV, P), contador de programa (PC), registro de Instrucción, decodificador, el bloque de memoria correspondiente a RAM Interna., y un bloque puesto detrás de èste que representa el área de direccionamiento indirecto. Exterior al MCS-52 constan : el bus de direcciones representado como una flecha apuntando ROM Externa, los bloques de memoria correspondientes a ROM Externa y

RAM externa (64K para cada uno) y ocupando la parte inferior una caja de mensajes. Cuando una instrucción se ejecuta su visualización gráfica consiste en lo siguiente :

* En la ROM Externa asoma escrito a partir de la localidad apuntada por el contador de programa (PC) el código de máquina correspondiente a la instrucción a ejecutar.

* El Bus de direcciones procede a apuntar una a una las localidades que contienen el código de instrucción con los siguientes eventos asociados: al apuntar el primer código, se ve como èste es llevado al registro de instrucción por medio de una flecha móvil, si es necesario al apuntar el segundo código èste puede ser llevado a alguna localidad de RAM interna o al ALU, y de existir un tercer código puede darse lo mismo que para el segundo código.

* Cuando el Bus de direcciones a apuntado todos los códigos, en el decodificador asoma la instrucción a ejecutar.

Hasta aquí todos estos pasos siempre se darán para cualquier instrucción.

Con excepción de las instrucciones AJMP, LJMP, SJMP, NOP todas las demás mostrarán un bloque con las localidades de RAM Interna involucradas este bloque será sobrepuesto al bloque de RAM interna normal. y constará de 2 o 3 columnas

a) b)

Figura 4.2 Bloque de localidades involucradas para

a) Instrucciones No Booleanas

b) Instrucciones Booleanas

Ahora la ejecución dependerá de la instrucción así:

Para instrucciones que involucren el ALU, se verá como los datos son llevados a ésta por medio de flechas móviles y al llegar asomará un mensaje con la operación que se ejecuta (suma, resta, multiplicación, división, and, or, xor, ajuste decimal), seguidamente se verá como el resultado sale del ALU y se dirige al destino que le corresponda y finalmente se regresará al fondo permanente de pantalla (figura 4.1) pero con sus valores actualizados.

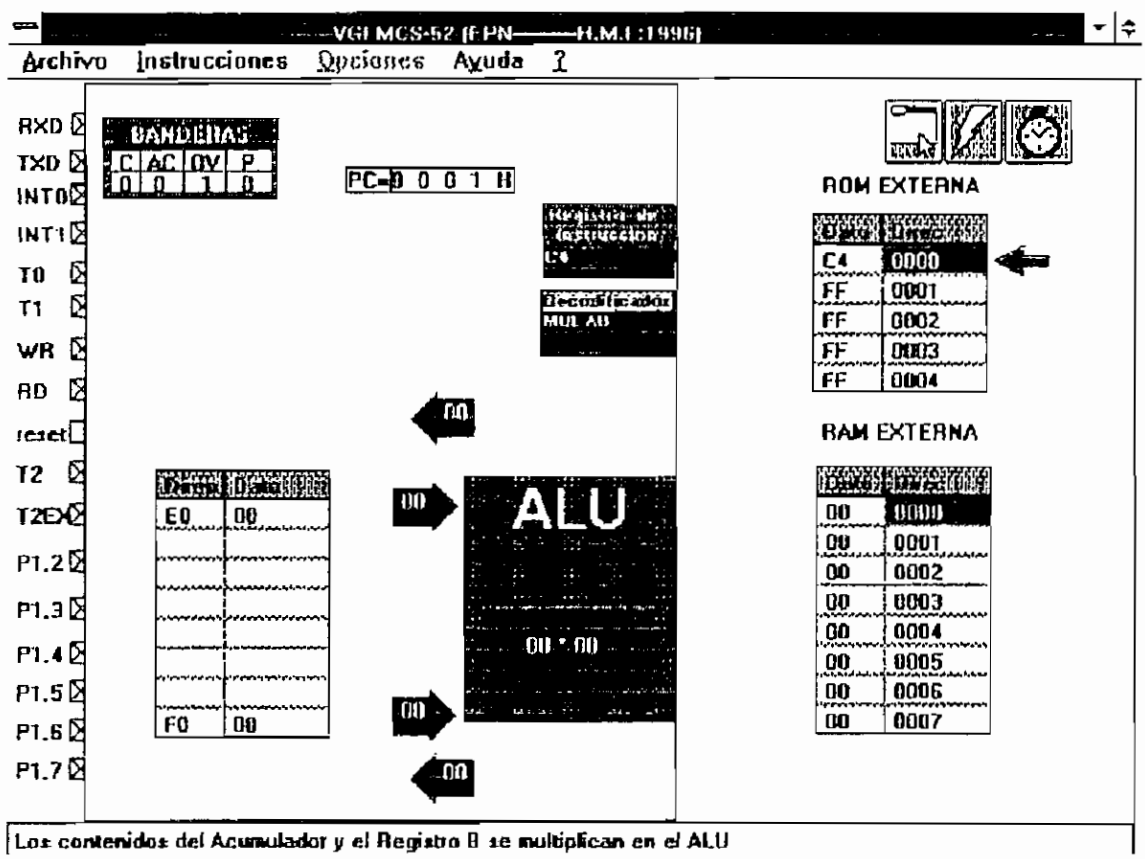


Figura 4.3 Ejecución de MUL AB

Para el caso de instrucciones de rotación, SWAP, CPL ocurrirá lo mismo que lo dicho anteriormente pero el mensaje que mostrará el ALU será reemplazado por el valor del dato en formato binario.

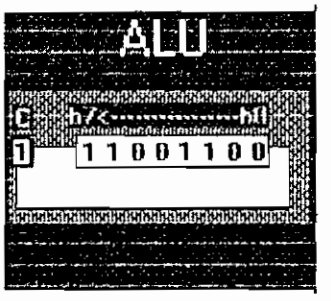


Figura 4.4 ALU con mensaje en formato binario

Para las instrucciones LCALL, ACALL, LJMP, AJMP, asomará un registro temporal donde será almacenado la dirección de salto ; puede asomar en formato hexadecimal o binario.

Para instrucciones de transferencia se verá como el dato es llevado del origen al destino

Si algún timer fuera habilitado se puede ver la variación de los contenidos de sus registros y banderas seleccionando el ícono reloj

De ser habilitada alguna interrupción, se simulará un servicio de interrupción que ejecutará las siguientes acciones:

LCALL Vector de interrupción

Al ejecutar la subrutina de interrupción el bus de direcciones apuntará las 8 localidades que puede tener una rutina de interrupción, y el contenido de estas localidades será simplemente un mensaje como "RUTINA DE INTERRUPCION PARA INT0", ver figura 4.5, hecho èsto automáticamente se ejecutará un RETI, cabe señalar que el programa se encargará de quitar las condiciones que generaron la interrupción (borrado de banderas).

ROM EXTERNA

Dato	Dir.
RUTINA	0003
DE	0004
INTERRUPCION	0005
PARA	0006
INT0	0007

Figura 4.5 Contenido de Rutina de interrupción para INT0

CONCLUSIONES

El programa elaborado ha tratado de representar de una forma gráfica lo más clara posible el proceso que implica la ejecución de una instrucción, es decir intentar dar una nueva forma de explicación, personalmente creo que se lo ha conseguido en la mayoría de instrucciones, pero la última palabra la tendrá el usuario.

El VGI MCS-52 está dirigido a todo aquel que esté iniciándose en el estudio del Microcontrolador MCS-52 ya que en este momento es cuando se necesitan la mayor cantidad de herramientas didácticas que ayuden a su comprensión. Pero cuando se ha llegado a un nivel en el que lo mas importante es probar el funcionamiento de un programa para controlar el Microcontrolador entonces ya no es útil este Software, recomendándose en este caso el uso del Simulador.

En lo que se refiere a :

Los timers ; para èstos se ha previsto todos sus modos de funcionamiento, pero solo hasta cierto punto se ha logrado dar una idea de que es lo que sucede cuando un timer se habilita por Ej se puede ver la variación de sus registros, y estado de las banderas pero un timer es mas que eso por lo cual se recomienda otros materiales de consulta para su entendimiento.

Los servicios de interrupción creo que se ha logrado una representación bastante buena de este proceso, pero se recomienda un estudio más profundo de este tópico., ya que aspectos como nivel de prioridades, apagado de banderas, etc. no es posible apreciarlo claramente.

El puerto serie, puesto que este tópico es tan extenso y cuyo funcionamiento podía ensombrecer al objetivo de esta tesis que es el Set de Instrucciones, se recomienda recurrir a otros materiales de consulta para su entendimiento.

Durante la elaboración del programa fue de mucha ayuda el uso del simulador existente, y en èste se detectó una falla en lo que se refiere a la ejecución de la instrucción XCH A,Pi, donde Pi es cualquiera de los puertos. La falla consiste en que el byte que reemplaza al que existía en el puerto es escrito en los pines mientras que el Latch permanece sin variación lo cual no es posible ya que la estructura de los puertos establece que no existe un camino a través del cual se pueda escribir en los pines sin pasar por el LATCH (Exceptuando cuando los puertos P0 y P2 se usan como bus de datos y direcciones).

Table 2. Instruction Opcodes In Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr,A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0, data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,/ bit	AND complement of direct bit to Carry	2	2
ORL	C,bit	OR direct bit to Carry flag	2	2
ORL	C,/ bit	OR complement of direct bit to Carry	2	2
MOV	C,bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp. immed. to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp. immed. to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp. immed. to ind. & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

- Rn — Working register R0-R7
- direct — 128 internal RAM locations, any I/O port, control or status register
- @Ri — Indirect internal RAM location addressed by register R0 or R1
- #data — 8-bit constant included in instruction
- #data16 — 16-bit constant included as bytes 2 & 3 of instruction
- bit — 128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

- addr16 — Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
- Addr11 — Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction
- rel — SJMP and all conditional jumps include an 8-bit offset byte. Range is +127-128 bytes relative to first byte of the following instruction

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

```
'MODULO PRINCIPAL (TESIS.FRM)
Option Compare Text
'DEFINICION DE MEMORIA Y SFR
Dim rami(383) As Integer, RAMx(2, 21900) As Integer
Dim ROMIX(2, 21900) As Integer
Dim CC, CARRY, AC, OV, N, pc, binario, P, BINARIO2
Dim acc, B, psw, SP, DPL, dpH, P0, P1, P2, p3, IP, IE, TMOD, TCON, T2CON
Dim th0, tl0, th1, tl1, TH2, TL2, rcap2h, rcap2l, scon, SBUF, PCON
Dim r0, r1, r2, r3, r4, r5, r6, R7, dptr

Dim FUENTE1, FUENTE2, FUENTER, ORIGEN, DESTINO, REL
Dim OPERACION$, MODO$, INSTRUCCION$, OPERANDO$
Dim A$, ii, RES, MD, tipo, cont, EXTRA
Dim COD(12), CODIGO(4), K(383), OP(4) As String
Dim NUMX(16), numy(16), numz(16)

Dim FINTE(2) As String, pulso(1) As Integer
Dim INDI%, EXTRAS, EXTRAS2, ini2
Dim paso%, esperal, tiempo, FILA, COLUMNA, SEGUIR$
Dim msgalu, PSW2, cmbyte
Dim pulso2%, t2exf%, CM%, poner$, p3aux%, p1aux%, posarch%
Dim disco$

Sub AJUSTE ()
'**************REALIZA EL AJUSTE DECIMAL***********
If MD = "A" Then
   msgalu = ""
   CM = 1
   LLEVOX = 0
   CODIGO(1) = &HD4
   Call CODIGOS(1)
   poner = "NO"
   Call PREVIO(datos2(DESTINO), datos2(FUENTE1), "", "")
   Call MENSAJES(24)
   picture9.Visible = True  '*****ALU***********
   aux = datos2(rami(DESTINO))
   A30 = Right$(aux, 1) 'NIBBLE BAJO
   A74 = Left$(aux, 1)  'NIBBLE ALTO
   Call hexdec(A30, ORIGEN)
   Call hexdec(A74, FUENTE2)
   If CInt(ORIGEN) > 9 Or CInt(AC) = 1 Then
            ORIGEN = ORIGEN + 6
            LLEVOX = CInt(Left$(datos2(ORIGEN), 1))
            msgalu = "6"
   Else
            msgalu = "0"
   End If
   If CInt(FUENTE2) + LLEVOX > 9 Or CInt(CARRY) = 1 Then
            A74 = CInt(FUENTE2) + 6 + LLEVOX
            msgalu = "6" & msgalu
            If CInt(A74) - 16 >= 0 Then
               CARRY = 1
            End If
   Else
            msgalu = "0" & msgalu
            A74 = CInt(FUENTE2) + LLEVOX
   End If
   Call hexdec(Right$(datos2(A74), 1) + Right$(datos2(ORIGEN), 1), rami(DESTINO))
   msgalu = datos2(FUENTE1) & "+" & msgalu
   Call EJECVISUAL(5)
   Call previo3(5, 1, 1, datos2(rami(DESTINO)))
   Call pausa(espera)
   '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
   Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO))) 'ACTUALIZA EL ACUMULADOR
   GRID2(3).TopRow = DESTINO + 1
Else
   MsgBox "NO EXISTE ESA INSTRUCCION"
End If
End Sub

Sub archivo_click (index As Integer)
On Error GoTo cancelar '
Dim archbien$
```

```
posarch = False: archbien = "si"
Select Case index
Case 0 '**********NUEVO*******
   screen.MousePointer = 11'puntero de mouse en forma de reloj
   Close #3
   Kill disco + "inicial.set"
   Open disco + "inicial.set" For Output Access Write As #3
   Call reseteo
   screen.MousePointer = 0' mouse NORMAL
Case 1 '*********ABRIR*****
   Close #3
   cmdialog1.Action = 1
   HH = 0
   If archbien = "si" Then
         barra(0).Enabled = False
         barra(1).Enabled = False
         barra(3).Enabled = False
         barra(4).Enabled = False
         opciones(0).Enabled = False
         opciones(1).Enabled = False
         opciones(3).Enabled = False
         screen.MousePointer = 11
         list1.Clear
         command3d1(2).Enabled = True
         label11.Caption = cmdialog1.Filename
         Call reseteo
         Open cmdialog1.Filename For Input Access Read As #7
         While posarch + EOF(7) = False
            DoEvents
            Line Input #7, A$
            list1.AddItem A$ 'SI ABRO UN Vacio
            list1.TopIndex = HH'list1.ListCount - 1
            Call SEPARAR
            If OP(3) <> "" Then
                  coma = ",": coma2 = ","
            Else
                  If OP(2) <> "" Then
                     coma = "": coma2 = ","
                  End If
            End If
            INSTRUCCION$ = OPERACION$ & " " & OP(1) & coma2 & OP(2) & coma & OP(3)
            Call EJECUCION
            Call MUESTREO
            HH = HH + 1
         Wend
         screen.MousePointer = 0
         Close #7
         FileCopy cmdialog1.Filename, disco + "inicial.set"
         Open disco + "inicial.set" For Append Access Write As #3
         barra(0).Enabled = True
         barra(1).Enabled = True
         barra(3).Enabled = True
         barra(4).Enabled = True
         opciones(0).Enabled = True
         opciones(1).Enabled = True
         opciones(3).Enabled = True
   Else
         Open disco + "inicial.set" For Append Access Write As #3
   End If
Case 2 '********GUARDAR COMO**********
   Close #3
   cmdialog1.Action = 2
   If archbien = "si" Then
         FileCopy disco + "inicial.set", cmdialog1.Filename
   End If
   Open disco + "inicial.set" For Append Access Write As #3
Case 3 '***************FIN*******
   Close #3
   kj = MsgBox("Desea guardar la sesion de trabajo", 35, "VI")'
   Select Case kj
   Case 6 'yes
         cmdialog1.Action = 2
         If archbien = "si" Then
```

```
            FileCopy disco + "inicial.set", cmdialog1.Filename
            Kill disco + "inicial.set"
            End
          Else
            Open disco + "inicial.set" For Append Access Write As #3
            Exit Sub
          End If

  Case 7 'no
          Kill disco + "inicial.set"
          End
  Case 2 'cancel
          Open disco + "inicial.set" For Append Access Write As #3
          Exit Sub
  End Select
End Select
Exit Sub
cancelar:
  archbien = "no"
Resume Next
End Sub

Sub ARIT_Click (index As Integer)
barra(0).Enabled = False
barra(1).Enabled = False
barra(2).Enabled = False
Select Case index
Case 1
  Call MODO1(1, 4, "ADD")
Case 2
  Call MODO1(5, 8, "ADDC")
Case 3
  Call MODO1(9, 12, "SUBB")
Case 4
  Call MODO1(13, 17, "INC")
Case 5
  Call MODO1(18, 21, "DEC")
Case 6
  Call MODO1(22, 22, "MUL")
Case 7
  Call MODO1(23, 23, "DIV")
Case 8
  Call MODO1(24, 24, "DA")
End Select
End Sub

Sub ARITLOG (aumento%)
Dim m1$, m2$, m3$
CM = 1: SITIO = 8
Select Case MD
Case "A,Rn"
  CODIGO(1) = &H28 + tipo + N
  Call CODIGOS(1): INDI = 1
  Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
  Call MENSAJES(1 + aumento)
  GoSub DECIDA
Case "A,DIREC"
  CODIGO(1) = &H25 + tipo: CODIGO(2) = ORIGEN
  Call CODIGOS(2): INDI = 1
  Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
  Call MENSAJES(2 + aumento)
  GoSub DECIDA
Case "A,@Ri"
  CODIGO(1) = &H26 + tipo + N
  Call CODIGOS(1): INDI = 0
  Call PREVIO(datos2(EXTRAS), datos2(rami(EXTRAS)), datos2(ORIGEN), datos2(FUENTE2))
  Call MENSAJES(3 + aumento)
  GoSub DECIDA
Case "A,#DIREC"
  SITIO = 1
  CODIGO(1) = &H24 + tipo: CODIGO(2) = (ORIGEN)
  Call CODIGOS(2): INDI = 2
  poner = "NO"
```

```
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    Call MENSAJES(4 + aumento)
    GoSub DECIDA
  Case "L A,Rn"
    CODIGO(1) = &H28 + tipo + N
    Call CODIGOS(1): INDI = 1
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(25 + aumento)
    GoSub DECIDA
  Case "L A,DIREC"
    CODIGO(1) = &H25 + tipo: CODIGO(2) = ORIGEN
    Call CODIGOS(2): INDI = 1
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(26 + aumento)
    GoSub DECIDA
  Case "L A,@Ri"
    CODIGO(1) = &H26 + tipo + N
    Call CODIGOS(1): INDI = 0
    Call PREVIO(datos2(EXTRAS), datos2(rami(EXTRAS)), datos2(ORIGEN), datos2(FUENTE2))
    Call MENSAJES(27 + aumento)
    GoSub DECIDA
  Case "L A,#DIREC"
    SITIO = 1
    CODIGO(1) = &H24 + tipo: CODIGO(2) = (ORIGEN)
    Call CODIGOS(2): INDI = 2
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    Call MENSAJES(28 + aumento)
    GoSub DECIDA
  Case "L DIREC,A"
    CODIGO(1) = &H52 + tipo: CODIGO(2) = DESTINO
    Call CODIGOS(2): INDI = 1
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(29 + aumento)
    GoSub DECIDA
  Case "L DIREC,#DIREC"
    CM = 2
    SITIO = 1
    CODIGO(1) = &H53 + tipo
    CODIGO(2) = DESTINO: CODIGO(3) = ORIGEN
    Call CODIGOS(3): INDI = 2
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    Call MENSAJES(30 + aumento)
    GoSub DECIDA
  Case "L CBIT"
    auxiliar = rami(ORIGEN)
    CM = 2
    raya$ = ""
    CODIGO(1) = &H52 + tipo
    CODIGO(2) = cmbyte: Call CODIGOS(2)
    If CInt(ORIGEN) = &H90 Then 'puerto1
            rami(ORIGEN) = 0
            For mm = 0 To 7
              rami(ORIGEN) = rami(ORIGEN) + (2 ^ mm) * pines(9 + mm).Value
            Next mm
    Else
            If CInt(ORIGEN) = &HB0 Then 'puerto  3
              rami(ORIGEN) = 0
              For mm = 0 To 7
                    rami(ORIGEN) = rami(ORIGEN) + (2 ^ mm) * pines(mm).Value
              Next mm
            End If
    End If
    Call BIN(rami(ORIGEN), mumy())
    FUENTE3 = mumy(FUENTE2)
    Call MENSAJES(84 + aumento * 2 / 6)
    GoSub DECIDABIT
    rami(ORIGEN) = auxiliar
  Case "L C/BIT"
    auxiliar = rami(ORIGEN)
    CM = 2
    raya$ = "/"
```

```
    CODIGO(1) = &H80 + tipo
    CODIGO(2) = cmbyte: Call CODIGOS(2)
    If CInt(ORIGEN) = &H90 Then 'puerto1
            rami(ORIGEN) = 0
            For mm = 0 To 7
               rami(ORIGEN) = rami(ORIGEN) + (2 ^ mm) * pines(9 + mm).Value
            Next mm
    Else
            If CInt(ORIGEN) = &HB0 Then 'puerto  3
               rami(ORIGEN) = 0
               For mm = 0 To 7
                     rami(ORIGEN) = rami(ORIGEN) + (2 ^ mm) * pines(mm).Value
               Next mm
            End If
    End If

    Call BIN(rami(ORIGEN), numy())
    FUENTE3 = numy(FUENTE2)
    numy(FUENTE2) = 1 Xor numy(FUENTE2)'saco el complemento

    Call MENSAJES(85 + aumento * 2 / 6)
    GoSub DECIDABIT
    rami(ORIGEN) = auxiliar
Case Else
    MsgBox "INSTRUCCION NO RECONOCIDA"
End Select
GoTo FIN2

DECIDABIT:
    FUENTE1 = CARRY 'VALOR ANTES DE SER OPERADO
    GRID2(5).Cols = 3
    GRID2(5).FixedRows = 0
    Call previo2(5, 0, 2, 0, 0, "Direc" & Chr(9) & "Dato" & Chr(9) & "Bit")
    GRID2(5).FixedRows = 1
    For I = 0 To 2
            GRID2(5).ColWidth(I) = 500
    Next I
    poner = "NO"
    Call PREVIO(datos2(ORIGEN), datos2(rami(ORIGEN)), " ", " ")
    Call BIN(rami(DESTINO), NUMX())
    Call previo3(5, 0, 8, datos2(DESTINO))
    Call previo3(5, 1, 8, datos2(rami(DESTINO)))
    Call previo3(5, 2, 1, FUENTE3)
    Call previo3(5, 2, 8, CARRY)

    Select Case RES
    Case 3
            CARRY = CARRY And numy(FUENTE2)
            NUMX(7) = CARRY  'ACTUALIZO BYTE
    Case 4
            CARRY = CARRY Or numy(FUENTE2)
            NUMX(7) = CARRY  'ACTUALIZO BYTE
    End Select
    Call DECI(NUMX(), rami(DESTINO))
    Call BIN(rami(DESTINO), numy())
    '******GRAFICACION*****************
    SIGNO$ = Choose(RES, " + ", " - ", " AND ", " OR ", " XOR ", " + ")
    FUENTE2 = FUENTE3
    mensaje$ = FUENTE1 & SIGNO$ & raya$ & FUENTE2
    msgalu = mensaje$
    picture9.Visible = True '***APARECE ALU******
    Call EJECVISUAL(14) '***LLAMO**PANTALLA*********
    Call previo3(5, 1, 8, datos2(rami(DESTINO)))
    Call previo3(5, 2, 8, CARRY)
    Call pausa(espera)
    '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
    Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO)))
    GRID2(3).TopRow = DESTINO + 1

Return

DECIDA:
    Select Case RES
```

```
    Case 6, 1
            GoSub SUMA
    Case 2
            GoSub RESTA
    Case 3
            rami(DESTINO) = FUENTE1 And FUENTE2 'ejecuta AND
    Case 4
            rami(DESTINO) = FUENTE1 Or FUENTE2 'ejecuta  OR
    Case 5
            rami(DESTINO) = FUENTE1 Xor FUENTE2 'ejecuta XOR
    End Select
    '******GRAFICACION*****************
    SIGNO$ = Choose(RES, " + ", " - ", " AND ", " OR ", " XOR ", " + ")
    mensaje$ = datos2(FUENTE1) & SIGNO$ & datos2(FUENTE2) & "H"
    MENSAJE2$ = mensaje$ & "  " & SIGNO$ & "CARRY"
    msgalu = Choose(CC + 1, mensaje$, MENSAJE2$)
    picture9.Visible = True '***APARECE ALU******
    Call EJECVISUAL(INDI) '*****PANTALLA*********
    'regreso
    Call previo3(5, 1, SITIO, datos2(rami(DESTINO)))
    Call pausa(espera)
    '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
    Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO)))
    GRID2(3).TopRow = DESTINO + 1
Return
SUMA:
    '*****ESTADO DE BANDERAS*********************
    If LSB(FUENTE1) + LSB(FUENTE2) + CC * CARRY > 15 Then
            AC = 1
    Else
            AC = 0 '
    End If
    rami(acc) = FUENTE1 + FUENTE2 + CC * CARRY'EJECUTO LA SUMA
    If rami(acc) > 255 Then
      rami(acc) = rami(acc) - 256
      CARRY = 1
    Else
            CARRY = 0
    End If

    If CInt(FUENTE1) < 128 And CInt(FUENTE2) < 128 Then 'POSITIVOS
            If CInt(rami(acc)) > 127 Then  'RESULTADO NEGATIVO
              OV = 1
            Else  'RESUL POSITIVO
              OV = 0
            End If
    Else
            If CInt(FUENTE1) > 127 And CInt(FUENTE2) > 127 Then 'NEGATIVOS
              If CInt(rami(acc)) < 128 Then 'RESULTADO POSITIVO
                    OV = 1
              Else  'resultado negativo
                    OV = 0
              End If
            Else  'signo diferente
              OV = 0
            End If
    End If
Return
RESTA:
    '*****ESTADO DE BANDERAS*********************
    If CInt(LSB(FUENTE1) < CInt(LSB(FUENTE2) + CARRY * CC)) Then
            AC = 1
    Else
            AC = 0
    End If
    ACCAUX% = FUENTE1 - FUENTE2 - CC * CARRY'ejecuta la operacion add
    If ACCAUX% < 0 Then
            rami(acc) = 256 + ACCAUX%
            CARRY = 1
    Else
            rami(acc) = ACCAUX%
            CARRY = 0
    End If
```

```
   '***DETERMINO OV***************
   If CInt(FUENTE1) < 128 And CInt(FUENTE2) > 127 Then '+,-
          If CInt(rami(acc)) > 127 Then  'RESULTADO NEGATIVO
             OV = 1
          Else  'RESUL POSITIVO
             OV = 0
          End If
   Else
          If CInt(FUENTE1) > 127 And CInt(FUENTE2) < 128 Then '-,+
             If CInt(rami(acc)) < 128 Then 'RESULTADO POSITIVO
                    OV = 1
             Else
                    OV = 0
             End If
          Else
             OV = 0
          End If
   End If
Return
FIN2:
End Sub

Sub banderas ()
'*************ACTUALIZACION DE BANDERAS*****************
ReDim puerto13(16), paux(16)
'********Obtengo el bit de paridad P*********
CONTAR = 0
Call BIN(rami(acc), puerto13())
For I = 0 To 7
   If CInt(puerto13(I)) = 1 Then
          CONTAR = CONTAR + 1
   End If
Next I
If CInt(CONTAR - 2 * Fix(CONTAR / 2)) <> 0 Then
   P = 1
Else
   P = 0
End If
If CInt(PSW2) = CInt(rami(psw)) Then 'ocurre cuando las
   Call BIN(rami(psw), puerto13())     'banderas han
   puerto13(7) = CARRY                 ' cambiado
   puerto13(6) = AC                   'independientemente
   puerto13(2) = OV
   puerto13(0) = P
   Call DECI(puerto13(), rami(psw))
   Call previo3(3, 2, psw + 1, datos2(rami(psw)))
   PSW2 = rami(psw)
Else  'ocurre cuando el PSW ha sido cambiado
   Call BIN(rami(psw), puerto13())
   CARRY = puerto13(7) 'actualizo las banderas
   AC = puerto13(6)
   OV = puerto13(2)
   If CInt(puerto13(0)) <> CInt(P) Then
          puerto13(0) = P
          Call DECI(puerto13(), rami(psw))
   End If
   Call previo3(3, 2, psw + 1, datos2(rami(psw)))
End If
label3(0).Caption = " " & CARRY & "  " & AC & "    " & OV & "    " & P
LABEL6.Caption = datos2(rami(psw)) & "  " & datos2(PSW2)
'*********Actualizo pines de p1*******
Call BIN(rami(P1), puerto13())
Call BIN(p1aux, paux())
For pp = 0 To 7
   If CInt(puerto13(pp)) = 0 Then 'x-->0
          pines(pp + 9).Value = 0
          pines(pp + 9).Enabled = False
   Else
          If paux(pp) = 0 Then   '0-->1
             pines(pp + 9).Value = 1
             pines(pp + 9).Enabled = True
          Else '1-->1
             pines(pp + 9).Enabled = True
```

```
            End If
    End If
Next pp
'***************Actualizo pines de p3***************
Call BIN(rami(p3), puerto13())
Call BIN(p3aux, paux())
For pp = 0 To 7
    If CInt(puerto13(pp)) = 0 Then 'x-->0
            pines(pp).Value = 0
            pines(pp).Enabled = False
    Else
            If paux(pp) = 0 Then   '0-->1
              pines(pp).Value = 1
              pines(pp).Enabled = True
            Else '1-->1
              pines(pp).Enabled = True
            End If
    End If
Next pp
End Sub

Sub barra_Click (index As Integer)
Select Case index
Case 3
    AYUDA.Show
Case 4
    acerca.Command1.Visible = True
    acerca.Show
End Select
End Sub

Sub BIN (NUMERO, COD())
'******transformacion de un numero decimal a binario*****
NUMERO2 = (NUMERO)
For I = 0 To 15
    COD(I) = 0
Next I
I = 0
While CLng(NUMERO2) > 0
    NUM1 = Fix(NUMERO2 / 2)
    COD(I) = NUMERO2 - NUM1 * 2
    NUMERO2 = NUM1
    I = I + 1
Wend
binario = COD(7) & COD(6) & COD(5) & COD(4) & COD(3) & COD(2) & COD(1) & COD(0)
BINARIO2 = COD(15) & COD(14) & COD(13) & COD(12) & COD(11) & COD(10) & COD(9) & COD(8)
End Sub

Sub BOOLEANAS_Click (index As Integer)
barra(0).Enabled = False
barra(1).Enabled = False
barra(2).Enabled = False
Select Case index
Case 0
    Call MODO1(78, 79, "CLR")
Case 1
    Call MODO1(80, 81, "SETB")
Case 2
    Call MODO1(82, 83, "CPL")
Case 3
   Call MODO1(84, 85, "ANL")
Case 4
   Call MODO1(86, 87, "ORL")
Case 5
   Call MODO1(88, 89, "MOV")
Case 6
   Call MODO1(90, 90, "JC")
Case 7
   Call MODO1(91, 91, "JNC")
Case 8
   Call MODO1(92, 92, "JB")
Case 9
   Call MODO1(93, 93, "JNB")
```

```
Case 10
  Call MODO1(94, 94, "JBC")
End Select

End Sub

Sub BUSCAR (direccion)
'PERMITE ENCONTRAR EL EQUIVALENTE DE LA LOCALIDAD DENTRO
'DE LA MATRIZ DEFINIDA COMO MEMORIA 3FILAS,21901COLUMNAS
For I = 0 To 2
   COLUMNA = direccion - I * 21901
   If COLUMNA <= 21900 Then
         FILA = I: Exit For
   End If
Next I
End Sub

Sub CEROUNO ()
CM = 1
CUADRO(11).Visible = False
GRID2(6).HighLight = True
Select Case MD
Case "A"  'CLR A
   CODIGO(1) = &HE4
   Call CODIGOS(1)
   '******RESULTADO AL DESTINO**********
   CUADRO(0).Move 1350, 1500 'MUESTRO REGISTRO
   LABEL1(17).Caption = "00" 'TEMPORAL
   CUADRO(0).Visible = True
   poner = "NO"
   Call PREVIO(datos2(DESTINO), datos2(rami(DESTINO)), " ", " ")
   Call MENSAJES(43)
   GRID2(3).TopRow = acc + 1
   rami(acc) = 0 'EJECUTO
   Call FLECHAS(0, 3, "00")'MUEVO EL CERO HACIA
   Call movi(1500, 2430, 1, 0)   'EL ACUMULADOR
   Call previo3(5, 1, 1, datos2(rami(acc)))
   Call pausa(espera)
   Call previo3(3, 2, DESTINO + 1, datos2(rami(acc)))
  'FALTA LAS BANDERAS

Case "C"
   CERUN = 0: MENSI = 0
   DESTINO = psw: FUENTE2 = 7
   CARRY = 0
   CODIGO(1) = &HC3
   Call CODIGOS(1)
   GoSub CLRSETB
Case "BIT", "direc1"
   CERUN = 0: MENSI = 1
   FUENTE2 = FUENTE1
   CODIGO(1) = &HC2: CODIGO(2) = cmbyte
   Call CODIGOS(2)
   GoSub CLRSETB
Case "CSET"
   CERUN = 1: MENSI = 2
   DESTINO = psw: FUENTE2 = 7
   CARRY = 1
   CODIGO(1) = &HD3
   Call CODIGOS(1)
   GoSub CLRSETB
Case "BITSET"
   CERUN = 1: MENSI = 3
   FUENTE2 = FUENTE1
   CODIGO(1) = &HD2: CODIGO(2) = cmbyte
   Call CODIGOS(2)
   GoSub CLRSETB
Case "CPL A"
   CODIGO(1) = &HF4
   Call CODIGOS(1)
   GRID2(3).TopRow = acc + 1
   poner = "NO"
   Call PREVIO(datos2(DESTINO), datos2(rami(DESTINO)), " ", " ")
```

```
    Call MENSAJES(44)
    Call BIN(rami(acc), NUMX())'PASO A BINARIO EL ACUMULADOR
    For I = 7 To 0 Step -1
            Call previo3(6, I, 0, CStr(NUMX(7 - I)))'VISUALIZA EL CONTENIDO DEL
            NUMX(7 - I) = 1 Xor NUMX(7 - I)'***SACO EL COMPLEMENTO**
    Next I
    Call DECI(NUMX(), rami(acc))
    Call pausa(espera)
    GoSub COMUN2
Case "CPL C"
    Call BIN(rami(DESTINO), NUMX())
    CERUN = 1 Xor NUMX(FUENTE1)
    CARRY = 1 Xor NUMX(FUENTE1)
    DESTINO = psw: FUENTE2 = 7
    CODIGO(1) = &HB3
    Call CODIGOS(1)
    Call MENSAJES(82)
    GoSub BORRABIT
Case "CPL BIT"
    Call BIN(rami(DESTINO), NUMX())
    CERUN = 1 Xor NUMX(FUENTE1): FUENTE2 = FUENTE1
    CODIGO(1) = &HB2: CODIGO(2) = cmbyte
    Call CODIGOS(2)
    Call MENSAJES(83)
    GoSub BORRABIT
Case Else
    MsgBox "INSTRUCCION NO RECONOCIDA"
End Select
GoTo FINAL7

CLRSETB:
    CUADRO(0).Move 1350, 1500
    LABEL1(17).Caption = CStr(CERUN)
    CUADRO(0).Visible = True
    Call MENSAJES(78 + MENSI)
    Call BIN(rami(DESTINO), NUMX())
    GRID2(5).Cols = 3 '
    GRID2(5).FixedRows = 0
    Call previo2(5, 0, 2, 0, 0, "Direc" & Chr(9) & "Dato" & Chr(9) & "Bit")
    GRID2(5).FixedRows = 1
    For I = 0 To 2
            GRID2(5).ColWidth(I) = 500
    Next I
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(rami(DESTINO)), " ", " ")
    Call previo3(5, 2, 1, NUMX(FUENTE2))
    GRID2(3).TopRow = DESTINO + 1
    NUMX(FUENTE2) = CERUN   'EJECUTO BIT
    Call DECI(NUMX(), rami(DESTINO))'ACTUALIZO BYTE
    '******RESULTADO AL DESTINO**********
    Call FLECHAS(0, 3, CStr(CERUN))'******
    Call movi(1500, 2430, 1, 0)
    'ACTUALIZO LOCALIDADES
    Call previo3(5, 1, 1, datos2(rami(DESTINO)))
    Call previo3(5, 2, 1, NUMX(FUENTE2))
    Call pausa(espera)
    '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
    Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO)))
    GRID2(3).TopRow = DESTINO + 1
Return
BORRABIT:
    Call BIN(rami(DESTINO), NUMX())
    FUENTE1 = NUMX(FUENTE2)'valor antes de ser operado
    GRID2(5).Cols = 3
    GRID2(5).FixedRows = 0
    Call previo2(5, 0, 2, 0, 0, "Direc" & Chr(9) & "Dato" & Chr(9) & "Bit")
    GRID2(5).FixedRows = 1
    For I = 0 To 2
            GRID2(5).ColWidth(I) = 500
    Next I
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(rami(DESTINO)), " ", " ")
    Call previo3(5, 2, 1, NUMX(FUENTE2))
```

```
    GRID2(3).TopRow = DESTINO + 1
    NUMX(FUENTE2) = CERUN 'ejecuto BIT
    Call DECI(NUMX(), rami(DESTINO))'ACTUALIZO BYTE
    '******GRAFICACION*****************
    msgalu = mensaje$
    picture9.Visible = True '***APARECE ALU******
    Call EJECVISUAL(18) '***LLAMO**PANTALLA*********
    Call previo3(5, 1, 1, datos2(rami(DESTINO)))
    Call previo3(5, 2, 1, NUMX(FUENTE2))
    Call pausa(espera)
    '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
    Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO)))
    GRID2(3).TopRow = DESTINO + 1

Return
COMUN2:
    '****VISUALIZACION DE CLR****************
    CUADRO(1).BorderStyle = 0
    CUADRO(1).Visible = True
    CUADRO(1).Left = 2145
    CUADRO(1).Top = 3345
    CUADRO(1).Width = 2880
    CUADRO(1).Height = 2100
    INDI = 3
    CUADRO(1).ZOrder 0
    SendKeys "{break}", True: wait = False
    GRID2(5).ZOrder 0
    PICTURE2(28).Visible = True
    Call pausa(espera)
    CUADRO(2).Visible = True
    Call pausa(espera)
    DATO$ = ""
    For I = 7 To 0 Step -1
            DATO$ = DATO$ + CStr(NUMX(I)) + Chr(9)
    Next I
    Call previo2(6, 0, 7, 0, 0, DATO$)
    LABEL1(8).Caption = "    "
    '****TRASLADO DEL RESULTADO AL DESTINO**********
    Call EJECVISUAL(INDI)
    Call previo3(5, 1, 1, datos2(rami(DESTINO)))
    Call pausa(espera)
    Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO)))
Return
FINAL7:
    GRID2(6).HighLight = False
End Sub

Sub CINE ()
CM = 2
PC2 = DESTINO
Select Case MD
Case "A,DIREC,DIREC"
    '****CODIGOS**********
    OPERACION$ = "hghjgjh"
    CODIGO(1) = &HB5: CODIGO(2) = ORIGEN: CODIGO(3) = REL
    Call CODIGOS(3)
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call EJECVISUAL(16)
Case "A,#DIREC,DIREC"
    '****CODIGOS**********
    CODIGO(1) = &HB4: CODIGO(2) = ORIGEN: CODIGO(3) = REL
    Call CODIGOS(3)
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    ini2 = 0
    Call EJECVISUAL(16)
Case "Ru,#DIREC,DIREC"
    '****CODIGOS*********
    CODIGO(1) = &HB8 + N: CODIGO(2) = ORIGEN: CODIGO(3) = REL
    Call CODIGOS(3)
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    ini2 = 1
```

```
   Call EJECVISUAL(16)
Case "@Ri,#DIREC,DIREC"
   '****CODIGOS**********
   CODIGO(1) = &HB6 + N: CODIGO(2) = ORIGEN: CODIGO(3) = REL
   Call CODIGOS(3)
   Call PREVIO(datos2(EXTRAS), datos2(DESTINO), " ", " ")
   Call EJECVISUAL(16)
Case Else
    MsgBox "INSTRUCCION NO RECONOCIDA"
End Select
End Sub

Sub CODIGOS (TOTAL)
BD = pc 'CARGO EL BUS DE DIRECCIONES
If CLng(pc + TOTAL - 1) > 65535 Then
   MSG1$ = "ESTA INSTRUCCION NO AVANZA A SER ESCRITA EN LA ROM A PARTIR DE LA LOCALIDAD APUNTADA POR
EL PC"
   MSG2$ = " POR LO CUAL SE ASUME QUE ESTARA ESCRITA  A PARTIR DE LA LOCALIDAD 0000H"
   MsgBox MSG1$ + MSG2$
   pc = 0
   BD = pc
End If
For I = 1 To 20  'TOTAL=# DE DATOS A ESCRIBIR EN ROM
   Call BUSCAR(BD + I - 1)
   If CInt(I) <= CInt(TOTAL) Then
          ROMIX(FILA, COLUMNA) = CODIGO(I)'ESCRIBO EN ROM EL CODIGO DE INSTRUCCION
   Else
          ROMIX(FILA, COLUMNA) = 255'ESCRIBO EN ROM
   End If
Next I
For I = 0 To 19  'TOTAL=# DE DATOS A ESCRIBIR EN ROM
   Call BUSCAR(BD + I)
   If CLng(BD + I) < 65536 Then
          GRID2(2).AddItem datos2(ROMIX(FILA, COLUMNA)) & Chr(9) & DATOS4(BD + I), I + 1
   Else
          GRID2(2).AddItem " ", I + 1
   End If
Next I
For I = 1 To 10
   GRID2(2).RemoveItem 1
Next I
GRID2(2).TopRow = 1

'****VISUALIZACION DE CONTENIDOS EN ROM EXTERNA***************
For J = 1 To TOTAL
   BD = pc
   If CLng(BD) < 65536 Then
          Call previo3(2, 1, J, CStr(DATOS4(BD)))
   End If
   Picture1(0).Visible = True  'FLECHA DE ROMX
   GRID2(2).TopRow = J
   pc = pc + 1
   If CLng(pc) > 65535 Then
          pc = 0'8192
   End If
   LABEL1(10).Caption = PCS(DATOS4(pc)) 'PC
   Beep
   SendKeys "{break}", True: wait = False
   GoSub ELEGIR
   Call pausa(espera)
Next J
For J = 1 To 10  'TOTAL=# DE DATOS A ESCRIBIR EN ROM
   GRID2(2).RemoveItem 1
Next J
GRID2(2).TopRow = TOTAL
'**********APUNTAR DIRECCION EN ROM EXT************
LABEL1(0).Visible = True 'decodificador
LABEL1(0).Caption = INSTRUCCION$
'Beep
GoTo FINXY
ELEGIR:
Select Case J
Case 1
```

```
    GoSub REGINS
    Select Case OPERACION$
    Case "ACALL", "AJMP"
            CUADRO(19).Visible = True
            LABEL1(21).Caption = NUMX(10) & NUMX(9) & NUMX(8)
    End Select
Case 2
    Select Case OPERACION$
    Case "ACALL", "AJMP"
            CUADRO(19).Visible = True
            LABEL1(21).Caption = NUMX(10) & NUMX(9) & NUMX(8) & binario
    Case "LCALL", "LJMP"
            CUADRO(19).Visible = True
            LABEL1(21).Caption = "  " & datos2(CODIGO(2))
    Case "MOVEDPTR#"
            FUENTE2 = CODIGO(2)
            LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 3675
            Call MOVERDATO(LL1, LL2, TT1, TT2, datos2(FUENTE2))
            LABEL1(19).Caption = datos2(CODIGO(2)) & datos2(rami(DPL))'*******
            CUADRO(15).Visible = False
            PICTURE4(5).Visible = False
    Case "cjne"
            picture9.Visible = True
            LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 3675
            Call MOVERDATO(LL1, LL2, TT1, TT2, datos2(CODIGO(2)))
    End Select
Case 3
    Select Case OPERACION$
    Case "LCALL", "LJMP"
            LABEL1(21).Caption = PCS(DATOS4(EXTRAS2))
    Case "MOVEDPTR#"
            FUENTE2 = CODIGO(3)
            Call MOVERDATO(LL1, LL2, TT1, TT2, datos2(FUENTE2))
            LABEL1(19).Caption = datos2(CODIGO(2)) & datos2(CODIGO(3))'*******
            CUADRO(15).Visible = False
            PICTURE4(5).Visible = False
    End Select
End Select
Return
REGINS: 'codigo se lleva al registro de instruccion
    CUADRO(15).ZOrder 1
    LABEL1(6).Caption = datos2(CODIGO(1))
    PICTURE4(5).Visible = True
    SendKeys "{break}", True: wait = False
    CUADRO(15).Move 6210, 1590
    CUADRO(15).Visible = True
    Call movi(6210, 5685, 0, 15)
    'Beep
    LABEL1(4).Caption = datos2(CODIGO(1))
    CUADRO(15).Visible = False
    PICTURE4(5).Visible = False
Return
FINXY:
End Sub

Sub COMBO2_Click ()
Command1(0).Enabled = True'boton aceptar
Command1(0).SetFocus
End Sub

Sub Command1_Click (index As Integer)
Select Case index
Case 0  '*******BOTON ACEPTAR*****************
    CUADRO(6).Visible = False  'cuadro Rn
    CUADRO(7).Visible = False  'cuadro @Ri
    CUADRO(8).Visible = False  'cuadro direccion
    OPERACION$ = label2(1).Caption   'operacion a ejecutar
    A$ = COMBO2.List(COMBO2.ListIndex) 'leo MD ESCOJIDO
    'los operandos estan generalizados Rn,@Ri,direcll
    Call SEPARAR 'separo los operandos
    ii = 1 'inicializo lazo de lectura de operandos
    Call ingresos
    'CONTINUA EN COMMAND4() DE SER NECESARIO
```

```
Case 1 '*******BOTON CANCELAR*******
   PICTURE12.ZOrder 0
   COMBO2.Clear
   'label2(4).Caption = ""
   'label2(5).Caption = ""
   label2(2).Visible = False
   label2(2).Caption = ""
   text1.Text = ""
   FRAME1.Visible = False
   CUADRO(6).Visible = False  'cuadro Rn
   CUADRO(7).Visible = False  'cuadro @Ri
   CUADRO(8).Visible = False  'cuadro direccion
   Command1(0).Enabled = False
   Command1(2).Enabled = False
   barra(0).Enabled = True
   barra(1).Enabled = True
   barra(2).Enabled = True
Case 2  '*******BOTON EJECUTAR*******
   FRAME1.Visible = False
   PICTURE12.ZOrder 0
   SendKeys "{break}", True: wait = False
   INSTRUCCION$ = label2(2).Caption 'instruccion a ejecutar
   Print #3, INSTRUCCION$
   label2(2).Visible = False
   COMBO2.Clear
   Call EJECUCION
   Call MUESTREO
   Command1(0).Enabled = False'aceptar
   Command1(2).Enabled = False 'ejecutar
   CUADRO(6).Visible = False  'cuadro Rn
   CUADRO(7).Visible = False  'cuadro @Ri
   CUADRO(8).Visible = False  'cuadro direccion
   barra(0).Enabled = True
   barra(1).Enabled = True
   barra(3).Enabled = True
   barra(4).Enabled = True
   opciones(0).Enabled = True
   opciones(1).Enabled = True
   opciones(3).Enabled = True
End Select
End Sub

Sub Command2_Click (index As Integer)
'botones para control de velocidad
Select Case index
Case 0
   paso = HSCROLL1.Value
   espera = (-paso / 160) + 3 'con min=1 y max=400
Case 1
End Select
PICTURE10.Visible = False

End Sub

Sub Command3D1_Click (index As Integer)
Select Case index
Case 0  'TIMERS
   'RAMi(T2CON) = 29
   'Call previo3(3, 2, T2CON + 1, datos2(RAMi(T2CON)))
   GRID2(1).Visible = Not GRID2(1).Visible
   GRID2(1).ZOrder 0
Case 1  'INTERRUPCIONES
   GRID2(8).Visible = Not GRID2(8).Visible
   GRID2(8).ZOrder 0
   'RAMi(TCON) = &H20  '1'INT0 FLANCO
   'RAMi(IE) = &HFF
   'RAMi(IP) = &HFF
   'Call previo3(3, 2, TCON + 1, datos2(RAMi(TCON)))
   'Call previo3(3, 2, IE + 1, datos2(RAMi(IE)))
   'Call previo3(3, 2, IP + 1, datos2(RAMi(IP)))
Case 2 'ver archivo

   picture3.Visible = Not picture3.Visible
```

```
   picture3.ZOrder 0
End Select

End Sub

Sub Command4_Click (index As Integer)
Dim MAXIMO&
'***PERMITE ESCOJER Rn,@Ri,BIT E INGRESAR DIREC,#DATO,REL,ETC
COMMAND4(7).TabStop = False
Select Case index
   Case 0 To 7        '**BOTONES PARA escojer Rn******
          label2(4) = "R" + CStr(index)
          COMMAND4(8).SetFocus
   Case 8             '***BOTON OK DE Rn******
          OP(ii) = label2(4).Caption
          If Trim$(label2(4).Caption) = "Rn" Then
          Else
             ii = ii + 1
             CUADRO(6).Visible = False
          End If
          Call ingresos
   Case 9 To 10       '**BOTONES PARA ESCOJER @Ri**********
          label2(5) = "@R" + CStr(index - 9)
          COMMAND4(11).SetFocus
   Case 11            '****OK PARA @Ri*****
          OP(ii) = label2(5).Caption
          If Trim$(label2(5).Caption) = "@Ri" Then
          Else
             ii = ii + 1
             CUADRO(7).Visible = False
          End If
          Call ingresos 'HACE QUE SE REPITA EL LAZO
   Case 12  '**OK PARA *dato,direccion1 y2,relativo**********
   If Trim$(text1.Text) = "" Then
          Call ingresos
          Else
          Select Case OPERANDO$
          Case "#DATO", "DIRECCION", "rel" 'ESTOS OPERANDOS
             MAXIMO = 255   'MAXIMO PUEDEN VALER 255
             GoSub FILTRO
          Case "DIREC11"    '***************
             MAXIMO = 2047
             GoSub FILTRO
          Case "DIREC16", "#DATO16"   '***************
             MAXIMO = 65535
             GoSub FILTRO
          Case "BIT", "/bit"   '**********
             operx$ = Trim$(text1.Text)'LEO EL BIT
             Call DECIBOOL(operx$, Orig, FUEN)
             If SEGUIR = "NO" Then
                    kj = MsgBox("OPERANDO NO VALIDO")
             Else
                    COMMAND4(12).Visible = False
                    If Mid$(label2(6).Caption, 1, 1) = "/" Then
                       OP(ii) = "/" + Trim$(text1.Text)
                    Else
                       OP(ii) = Trim$(text1.Text)
                    End If
                    text1.Text = ""
                    ii = ii + 1  'CONTROL LAZO DE LECTURA
                    CUADRO(8).Visible = False
             End If
             Call ingresos 'LAZO DE LECTURA
          End Select
   End If
End Select
GoTo FINF
FILTRO: 'SE ENCARGA DE VERIFICAR SI EL OPERANDO INGRESADO ES VALIDO
   operx$ = Trim$(text1.Text)  'lee valor ingresado
   Call DECIFRAR(operx$, Orig, FUEN)
   If SEGUIR = "NO" Or CDbl(Orig) > MAXIMO Then
          kj = MsgBox("OPERANDO NO VALIDO")
   Else  'EL OPERANDO ES VALIDO
```

```
            COMMAND4(12).Visible = False
            If Mid$(label2(6).Caption, 1, 1) = "#" Then
               OP(ii) = "#" + Trim$(text1.Text)
            Else
               OP(ii) = Trim$(text1.Text)
            End If
            text1.Text = ""
            ii = ii + 1 'CONTROL LAZO DE LECTURA
            CUADRO(8).Visible = False
   End If
   Call ingresos 'REPITE EL LAZO DE LECTURA
Return
FINF:
'**********continua en command3 (BOTON EJECUTAR)*****
End Sub

Function datos2 (NOMBRE)
If CInt(NOMBRE) < 16 Then
   datos2 = CStr("0" + Hex$(NOMBRE))
Else
   datos2 = Hex$(NOMBRE)
End If
End Function

Function DATOS4 (NOMBRE)
Select Case CLng(NOMBRE)
Case Is < 16
   DATOS4 = CStr("000" + Hex$(NOMBRE))
Case 16 To 255
   DATOS4 = CStr("00" + Hex$(NOMBRE))
Case 256 To 4095
   DATOS4 = CStr("0" + Hex$(NOMBRE))
Case Is > 4095
   DATOS4 = Hex$(NOMBRE)
End Select
End Function

Sub DECI (COD(), NUMERO)
'******transformacion de un numero  binario A DECIMAL*****
NUMERO = 0
For I = 0 To 15
   NUMERO = NUMERO + COD(I) * 2 ^ I
Next I
End Sub

Sub DECIBOOL (OPER123$, clase, fuentes)
SEGUIR = "NO"
SITIO = InStr(OPER123$, ".") 'FORMATO BIT.i
If SITIO > 1 Then            'EJ: CLASE.FUENTES = 33.7
   OPER4$ = OPER123$: OPER5$ = Left$(OPER123$, SITIO - 1)
   RESUL1 = OPER5$
   If IsNumeric(OPER5$) Then 'PERTENECE AL AREA 20H.2FH
         If CInt(OPER5$) >= 32 And CInt(OPER5$) <= 47 Then
            clase = CInt(OPER5$)'DIRECCION DEL BYTE
            MD = MD + "BIT"
            fuentes = CVar(Right(OPER123$, 1)) 'POSICION bi
            SEGUIR = Switch(CLng(fuentes) > 7, "NO", CLng(fuentes) < 8, "SI")
            cmbyte = CInt(clase) + CInt(fuentes)
         Else
            SEGUIR = "NO"
         End If
   Else 'ES DEL SFR DIRECCIONABLE BIT A BIT ; P1.6,B.4
         Open disco + "REGISTRO.DAT" For Input Access Read As #1
         For I = 1 To 82
            Line Input #1, DATO
            If CInt(I) >= 56 And CInt(I) <= 67 Then
                  NOMBRE$ = Left$(DATO, 6)
                  numer = CVar(Trim$((Mid$(DATO, 9, 3))))
                  resul = RESUL1 Like Trim$(NOMBRE$)
                     If resul = True Then
                           clase = numer
                           MD = MD + "BIT"
                           fuentes = CVar(Right(OPER4$, 1))
```

```
                                SEGUIR = Switch(CLng(fuentes) > 7, "NO", CLng(fuentes) < 8, "SI")
                                cmbyte = CInt(clase) + CInt(fuentes)
                                Exit For
                        Else : SEGUIR = "NO"
                        End If
                End If
            Next I
            Close #1
    End If
Else 'se ingreso nombre del bit Ej: TF0,TF1,ETC
    Open disco + "REGISTRO.DAT" For Input Access Read As #1
    For I = 1 To 113
            Line Input #1, DATO
            If CInt(I) >= 68 And CInt(I) <= 113 Then
                NOMBRE$ = Trim$(Left$(DATO, 8))
                resul = Trim$(OPER123$) Like NOMBRE$
                        If resul = True Then
                            clase = CVar(Trim((Mid(DATO, 9, 3))))
                            MD = MD + "BIT"
                            fuentes = CVar(Right(DATO, 1))
                            cmbyte = CInt(clase) + CInt(fuentes)
                            SEGUIR = "SI"
                            Exit For
                        Else : SEGUIR = "NO"
                        End If
            End If
    Next I
    Close #1
'EL BIT ESTA DIRECCIONADO DE LA FORMA 0,7FH
    If IsNumeric(OPER123$) Then 'FORMATO DECIMAL
            clase = CLng(OPER123$)
            If clase < 128 Then 'ESTA CORRECTO
                cmbyte = CInt(clase)   'codigo de maquina
                aux = Fix(clase / 8)
                fuentes = clase - 8 * aux
                clase = 32 + aux   'byte ; clase.fuentes
                MD = MD + "BIT"
                SEGUIR = "SI"
            End If
    Else 'PUEDE TENER FORMATO HEXADECIMAL
            ACHE$ = Right$(OPER123$, 1) '
            If ACHE$ = "H" Then
            DATO1$ = Left$(OPER123$, Len(OPER123$) - 1)
            Call hexdec(DATO1$, NUMERO) 'transf a decimal
            If NUMERO = "NO" Then 'DATO NO VALIDO
                SEGUIR = "NO"'ENVIAR MENSAJE DE DATO NO VALIDO
            Else  'DATO VALIDO
                clase = NUMERO
                cmbyte = CInt(clase)   'codigo de maquina
                aux = Fix(clase / 8)
                fuentes = clase - 8 * aux
                clase = 32 + aux   'byte ; clase.fuentes
                MD = MD + "BIT"
                SEGUIR = "SI"
            End If
    End If
End If
End If
End Sub

Sub DECIFRAR (OPER123$, clase, fuentes)
SITIO2 = InStr(OPER123$, ".")'descarto que se trate
sitio3 = InStr(OPER123$, "/") 'de bit o /bit
If SITIO2 = 0 And sitio3 = 0 Then
SEGUIR = "NO"
If IsNumeric(OPER123$) Then  'decimal
    Select Case CLng(OPER123$)
    Case Is < 256
            clase = CInt(OPER123$):
            fuentes = rami(clase)
            SEGUIR = "SI"
            MD = MD + "direc"
    Case 256 To 2047
```

```
            clase = (OPER123$): fuentes = 0
            SEGUIR = "SI"
            MD = MD + "direc11"
    Case 2048 To 65535
            clase = (OPER123$): fuentes = 0
            SEGUIR = "SI"
            MD = MD + "direc16"
    Case Else
            SEGUIR = "NO"
    End Select
Else                'etiqueta
    For I = 0 To 255
            resul = OPER123$ Like Trim$(K(I))
            If resul = True Then  'ojooooo
              clase = I
              fuentes = rami(clase)
              MD = MD + "DIREC"
              SEGUIR = "SI": Exit For
            Else
              SEGUIR = "NO"
            End If
    Next I
End If
If SEGUIR = "NO" Then   'VEO SI EL DATO ES HEXADECIMAL
    simbol$ = Right$(OPER123$, 1)
    DATO1$ = Left$(OPER123$, Len(OPER123$) - 1)
    Call hexdec(DATO1$, NUMERO) 'transf a decimal
    If NUMERO = "NO" Then
            SEGUIR = "NO"'ENVIAR MENSAJE DE DATO NO VALIDO
    Else
            Select Case simbol$
            Case "H"    'HEXADECIMAL
              Select Case CLng(NUMERO)
              Case Is < 256
                      clase = CInt(NUMERO)
                      fuentes = rami(clase)
                      SEGUIR = "SI"
                      MD = MD + "direc"
              Case 256 To 2047
                      clase = NUMERO: fuentes = 0
                      SEGUIR = "SI"
                      MD = MD + "direc11"
              Case 2048 To 65535
                      clase = NUMERO: fuentes = 0
                      SEGUIR = "SI"
                      MD = MD + "direc16"
              Case Else
                      SEGUIR = "NO"
              End Select
            Case "B"
            End Select
    End If
    End If
Else
    SEGUIR = "NO"
End If

findec:

End Sub

Sub DJNZ ()
CM = 2: PC2 = DESTINO
Select Case MD
Case "Rn,DIREC"
    '****CODIGOS**********
    CODIGO(1) = &HD8 + N: CODIGO(2) = ORIGEN
    Call CODIGOS(2)
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    Call MENSAJES(109)
    rami(DESTINO) = FUENTE1 - 1
    If rami(DESTINO) < 0 Then
```

```
            rami(DESTINO) = 255
    End If
    Call EJECVISUAL(17)
Case "DIREC,DIREC"
    '****CODIGOS*********
    CODIGO(1) = &HD5: CODIGO(2) = DESTINO: CODIGO(3) = ORIGEN
    Call CODIGOS(3)
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    Call MENSAJES(110)
    rami(DESTINO) = FUENTE1 - 1
    If rami(DESTINO) < 0 Then
            rami(DESTINO) = 255
    End If
    Call EJECVISUAL(17)
Case Else
    MsgBox "INSTRUCCION NO RECONOCIDA"
End Select

End Sub

Sub EJECUCION ()
barra(0).Enabled = False
barra(1).Enabled = False
barra(2).Enabled = True
barra(3).Enabled = False
barra(4).Enabled = False
opciones(0).Enabled = False
opciones(1).Enabled = False
opciones(3).Enabled = False
p3aux = rami(p3): p1aux = rami(P1)
frame2.Enabled = False
VSCROLL1(0).Visible = False
VSCROLL1(2).Visible = False
VSCROLL1(3).Visible = False
VSCROLL1(4).Visible = False
PSW2 = rami(psw)
GRID2(4).ZOrder 1
'***DE ACUERDO AL TIPO DE OPERACION DECIDE LO QUE HACE****
CC = 0:
Call hexdec(datos2(rami(dpH)) + datos2(rami(DPL)), dptr)

MODO$ = OP(1) & OP(2) & OP(3)
Call MODOS
Select Case OPERACION$
  Case "Add"
          CC = 0: RES = 6: tipo = &H0
          Call ARITLOG(0)
  Case "ADDC"
          CC = 1: RES = 1: tipo = &H10
          Call ARITLOG(4)
  Case "SUBB"
          CC = 1: RES = 2: tipo = &H70
          Call ARITLOG(8)
  Case "INC"
          Call INCDEC(1)
  Case "DEC"
          Call INCDEC(-1)
  Case "MUL"
          Call MULDIV(&H20)
  Case "DIV"
          Call MULDIV(0)
  Case "DA"
          Call AJUSTE
  Case "ANL"
          RES = 3: tipo = &H30
          MD = "L " + MD
          Call ARITLOG(0)
  Case "ORL"
          RES = 4: tipo = &H20
          MD = "L " + MD
          Call ARITLOG(6)
  Case "XRL"
```

```
        RES = 5: tipo = &H40
        MD = "L " + MD
        Call ARITLOG(12)
Case "CLR"
        Call CEROUNO
Case "SETB"
        MD = MD + "SET"
        Call CEROUNO
Case "CPL"
        MD = "CPL " + MD
        Call CEROUNO
Case "RL"
        Call ROTAR(&H20)
Case "RLC"
        Call ROTAR(&H30)
Case "RR"
        Call ROTAR(&H0)
Case "RRC"
        Call ROTAR(&H10)
Case "SWAP"
        Call ROTAR(&HC1)
Case "MOV"
        tipo = &HC0
        Call MOVER
Case "MOVC"
        Call MOVER
Case "MOVX"
        Call MOVERX
Case "PUSH"
        Call PUSHPOP(0)
Case "POP"
        Call PUSHPOP(16)
Case "XCH": tipo = &HA0
        Call XCH
Case "XCHD"
        tipo = &HB0
        MD = "XCHD" + MD
        Call XCH
Case "ACALL", "LCALL"
        Call SALTAR
Case "AJMP", "LJMP", "SJMP"
        Call SALTAR
Case "RET"
        Call RETORNO(0)
Case "RETI"
        Call RETORNO(&H10)
Case "jmp"
        Call SALTAR
Case "JZ", "JNZ"
        Call SALTAR
Case "CJNE"
        Call CJNE
Case "DJNZ"
        Call DJNZ
Case "JC", "JNC", "JB", "JNB", "JBC"
        Call SALTOBIT
Case "NOP"
        Call MENSAJES(111)
        If MD = "NADA" Then
        CODIGO(1) = 0: Call CODIGOS(1)
        End If
Case Else
        kj = MsgBox("No existe esa instruccion " & INSTRUCCION$)
TEXT2.Text = " "
DESTINO = 0: ORIGEN = 0: FUENTE1 = 0: FUENTE2 = 0
CODIGO(1) = 0: CODIGO(2) = 0: CODIGO(3) = 0
Call REGRESO(True)
LABEL1(21).Caption = " "
VSCROLL1(0).Visible = opciones(1).Checked
VSCROLL1(2).Visible = opciones(1).Checked
VSCROLL1(3).Visible = opciones(1).Checked
VSCROLL1(4).Visible = opciones(1).Checked
frame2.Enabled = True
```

```
    Exit Sub
End Select
    Call banderas
    TEXT2.Text = " "
    DESTINO = 0: ORIGEN = 0: FUENTE1 = 0: FUENTE2 = 0
    CODIGO(1) = 0: CODIGO(2) = 0: CODIGO(3) = 0
    Call REGRESO(True)
    LABEL1(21).Caption = " "
    VSCROLL1(0).Visible = opciones(1).Checked
    VSCROLL1(2).Visible = opciones(1).Checked
    VSCROLL1(3).Visible = opciones(1).Checked
    VSCROLL1(4).Visible = opciones(1).Checked
    frame2.Enabled = True
    TEXT2.Text = " "
End Sub

Sub EJECVISUAL (index As Integer)
Dim L1, L2, L3, L4, T1, T2, T3, T4, T5
L1 = 1710: T1 = 3975: T2 = 4980: L2 = 2250: L3 = 3555
T3 = 4550: T4 = 5505: L4 = 1200: T5 = 3195: EXTRA = 0
NOMBRE$ = "Ri"
LABEL1(15).Caption = datos2(FUENTE2): LABEL1(16).Caption = datos2(FUENTE1)
'********PANTALLAS PARA VISUALIZAR EJECUCION********
Select Case index
Case 0  '***DESTINO,@Ri**************
    TOPES = 4140: APUNTADO = ORIGEN
    GoSub PUNTERO
    GoSub ODALU

    GoSub RHDA
Case 1    '*******DESTINO,ORIGEN*************************
    T1 = T5
    GoSub ODALU
    GoSub RHDA
Case 2   '*************DESTINO,#dato*********************
    GoSub DHALU
    GoSub DATOHALU
    GoSub RHDUP
Case 3   '*******caso RL traslado del resultado**********************
    GoSub RHDUP
Case 4  '***@Ri COMO ORIGEN PARA MOV*************
    TOPES = 4140: APUNTADO = ORIGEN
    GoSub PUNTERO
Case 5    '*******PARA INC*************************
    GoSub DHALU
    LABEL1(8).Caption = msgalu
    LABEL1(8).Visible = True
    SendKeys "{break}", True: wait = False
    GoSub RHDUP
Case 6  '***@Ri PARA INC*todo*************
    TOPES = 5115: APUNTADO = DESTINO: T5 = T2
    GoSub PUNTERO
    GoSub DHALU
    LABEL1(8).Caption = msgalu
    LABEL1(8).Visible = True
    SendKeys "{break}", True: wait = False
    GoSub RHDA
Case 7  '***@Ri COMO DESTINO PARA MOV*************
    TOPES = 5115: APUNTADO = DESTINO
    GoSub PUNTERO
Case 8 '******MUL AB******
    T1 = 3195: T2 = T2
    GoSub ODALU
    Call banderas
    L1 = 3615: T1 = 3385: T2 = 4550: T3 = 2430: T4 = 5505: L2 = 1350
    CUADRO(0).Move L1, T1: CUADRO(3).Move L1, T2
    Call FLECHAS(0, 1, datos2(rami(acc)))'*****
    Call FLECHAS(3, 3, datos2(rami(B)))'*****
    For I = 0 To 955 Step paso
            CUADRO(0).Move L1, T1 - I
            CUADRO(3).Move L1, T2 + I
            Call pausa(tiempo)
    Next I
```

```
    CUADRO(0).Move L1, T1 - 955: CUADRO(3).Move L1, T2 + 955
    Call FLECHAS(0, 2, datos2(rami(acc)))'*****
    Call FLECHAS(3, 2, datos2(rami(B)))'*****
    Call movi2(L1, L2, 0, 0, 3) '***HORIZONTAL
    Call FLECHAS(0, 3, datos2(rami(acc)))'*****
    Call FLECHAS(3, 1, datos2(rami(B)))'*****
    LABEL1(8).Caption = msgalu
    LABEL1(8).Visible = True '**MENSAJE DE ALU*****
    SendKeys "{break}", True: wait = False
  Case 9  '***@SP COMO ORIGEN **********
    TOPES = 4140: APUNTADO = ORIGEN: NOMBRE$ = "SP"
    GoSub PUNTERO
  Case 10  '***@SP COMO DESTINO*************
    TOPES = 5115: APUNTADO = DESTINO: NOMBRE$ = "SP"
    GoSub PUNTERO
  Case 11  '*************@A+DPTR******************
    GoSub ADPTRALU
    ETIQ = "A+DPTR"
    GoSub APUNTAROMix
    CUADRO(18).Visible = False
    picture9.Visible = False
    CUADRO(16).Visible = False
    PICTURE4(4).Visible = False
  Case 12  '*************@A+PC******************
    picture9.Left = 2700
    picture9.Visible = True  '*****ALU************
    L2 = 1725:
    GoSub DHALU  '******ACC AL ALU*****
    msgalu = datos2(rami(acc)) & "+" & DATOS4(pc)
    GoSub PCALU
    LABEL1(8).Caption = msgalu
    LABEL1(8).Visible = True
    Call pausa(1)
    SendKeys "{break}", True: wait = False
    ETIQ = "A+PC"
    GoSub APUNTAROMix
    picture9.Visible = False
    CUADRO(17).Visible = False: PICTURE4(10).Visible = False
    CUADRO(17).Width = 1005
    CUADRO(21).Visible = False
    CUADRO(16).Visible = False: PICTURE4(4).Visible = False
  Case 13 'PC+REL
            FUENTE2 = REL
            picture9.Left = 2700
            picture9.Visible = True
            LABEL4.Caption = "ALU"
            msgalu = DATOS4(pc) & "+" & datos2(REL)
            GoSub PCALU
            GoSub RELHALU
            pc = pc + Switch(REL < 128, REL, REL > 127, REL - 256)
            If CLng(pc) > 65535 Then
              pc = pc - 65536
            Else
              If CLng(pc) < 0 Then
                    pc = 65536 + pc
              End If
            End If
            GoSub RAPC
  Case 14  'C,BIT*************************
    T1 = T5
    LABEL1(15).Caption = FUENTE2: LABEL1(16).Caption = FUENTE1
    GoSub ODALU
    CUADRO(17).Height = 700
    LABEL1(17).Caption = CARRY
    GoSub RHDABIT
  Case 15 'JMP @A+DPTR
    GoSub ADPTRALU
    pc = DESTINO
    GoSub RAPC
    CUADRO(18).Visible = False
  Case 16 'CJNE a,direc,rel
    T1 = T5
    picture9.Visible = True 'alu
```

```
    Select Case MD
    Case "A,DIREC,direc"
            msgalu = " "
            GoSub ODALU '****** A Y DIREC AL ALU
            Call MENSAJES(105)
    Case "A,#Direc,direc", "Rn,#DATO,REL"
            GoSub DHALU
            Call MENSAJES(106 + ini2)
    Case "@Ri,#Direc,direc"
            TOPES = 5115: APUNTADO = DESTINO
            GoSub PUNTERO
            T5 = T2
            GoSub DHALU
            Call MENSAJES(108)
    End Select
    CARRY = Switch(CInt(FUENTE1) < CInt(FUENTE2), 1, CInt(FUENTE1) >= CInt(FUENTE2), 0)
    If CInt(FUENTE1) <> CInt(FUENTE2) Then 'salto
            msgalu = "SALTO"
            If CInt(FUENTE1) > CInt(FUENTE2) Then
               label5.Caption = datos2(FUENTE1) & ">" & datos2(FUENTE2)
            Else
               label5.Caption = datos2(FUENTE1) & "<" & datos2(FUENTE2)
            End If
    CUADRO(17).Height = 900
    LABEL1(8).Caption = msgalu
    LABEL1(8).Visible = True'**MENSAJE DE ALU****
    label5.Visible = True  'MENSAJE DE SALTOS
    Call pausa(espera)
    picture9.Left = 2700
    CUADRO(15).Visible = False
    CUADRO(17).Visible = False
    CUADRO(16).Visible = False
    label5.Visible = False
    LABEL1(8).Visible = False
    FUENTE2 = REL
    picture9.Left = 2700: picture9.Visible = True
    msgalu = DATOS4(pc) & "+" & datos2(REL)
    GoSub PCALU
    GoSub RELHALU
    pc = pc + Switch(REL < 128, REL, REL > 127, REL - 256)
            If CLng(pc) > 65535 Then
               pc = pc - 65536
            Else
               If CLng(pc) < 0 Then
                     pc = 65536 + pc
               End If
            End If

    GoSub RAPC
    Else     'son iguales
            msgalu = "NO HAY SALTO"
            label5.Caption = datos2(FUENTE1) & "=" & datos2(FUENTE2)
            LABEL1(8).Caption = msgalu
            LABEL1(8).Visible = True
            label5.Visible = True
            Call pausa(espera)
            label5.Visible = False
            LABEL1(8).Visible = False
    End If
Case 17 'DJNZ direc,rel
    picture9.Visible = True
    msgalu = datos2(FUENTE1) & "-1"
    GoSub DHALU
    LABEL1(8).Caption = msgalu
    LABEL1(8).Visible = True
    GoSub RHDUP
    Call banderas
    Call previo3(5, 1, 1, datos2(rami(DESTINO)))
    Call pausa(espera)
    GRID2(3).TopRow = DESTINO + 1
    Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO)))
    If CInt(rami(DESTINO)) <> 0 Then
            CUADRO(17).Height = 900
```

```
            label5.Caption = datos2(rami(DESTINO)) & "<> 0"'MENSAJE DE SALTOS
            label5.Visible = True
            LABEL1(8).Caption = "SALTAR"
            Call pausa(espera)
            picture9.Left = 2700
            CUADRO(0).Visible = False
            CUADRO(15).Visible = False
            CUADRO(17).Visible = False
            CUADRO(16).Visible = False
            label5.Visible = False
            LABEL1(8).Visible = False
            REL = ORIGEN: FUENTE2 = REL
            picture9.Left = 2700: picture9.Visible = True
            msgalu = DATOS4(pc) & "+" & datos2(REL)
            GoSub PCALU
            GoSub RELHALU
            pc = pc + Switch(REL < 128, REL, REL > 127, REL - 256)
            If CLng(pc) > 65535 Then
               pc = pc - 65536
            Else
               If CLng(pc) < 0 Then
                     pc = 65536 + pc
               End If
            End If

            GoSub RAPC
      Else
            label5.Caption = datos2(rami(DESTINO)) & "= 0"'MENSAJE DE SALTOS
            label5.Visible = True
            LABEL1(8).Caption = "NO HAY SALTO"
            LABEL1(8).Visible = True
            Call pausa(espera)
      End If
   Case 18  '***********clr bit*******************
      '****DESTINO HACIA ALU***
      LABEL1(15).Caption = (FUENTE1)'*******
      CUADRO(16).Move L1, T5: PICTURE4(4).Visible = True
      CUADRO(16).Visible = True
      Call movi(L1, L2, 0, 16)
      LABEL1(8).Caption = FUENTE1
      LABEL1(8).Visible = True
      Call pausa(1)'espera)
      LABEL1(8).Caption = NUMX(FUENTE2)
      '******RESULTADO AL DESTINO**********
      CUADRO(0).Move 3615, 3100
      Call FLECHAS(0, 1, NUMX(FUENTE2))'******
      Call movi(3100, 2430, 1, 0)
      Call FLECHAS(0, 2, NUMX(FUENTE2))'******
      Call movi(3615, 1350, 0, 0)
      Call FLECHAS(0, 3, NUMX(FUENTE2))'******
   End Select
   CUADRO(17).Height = 900
   GoTo FINAL8
   PUNTERO:
      '***APARECE EL PUNTERO Ri******************
      LABEL1(7).Caption = NOMBRE$
      PICTURE7(1).Move Left, TOPES
      LABEL1(5).Caption = datos2(APUNTADO)
      PICTURE7(1).Visible = True 'PUNTERO
      SendKeys "{break}", True: wait = False
   Return
   ODALU:
      '******ORIGEN Y DESTINO AL ALU**********
      CUADRO(16).Move L1, T1: CUADRO(17).Move L1, T2
      Call FLECHAS(16, 4, LABEL1(15))'*****
      Call FLECHAS(17, 4, LABEL1(16))'*****
      Call movi2(L1, L2, 0, 16, 17) '*OR. Y DEST. AL ALU***
      LABEL1(8).Caption = msgalu
      LABEL1(8).Visible = True '**MENSAJE DE ALU*****
      SendKeys "{break}", True: wait = False
      CUADRO(17).Height = 700
      Call pausa(espera)
   Return
```

```
DHALU:  'POR ARRIBA
   '****DESTINO HACIA ALU***
   LABEL1(15).Caption = datos2(FUENTE1)'*******
   CUADRO(16).Move L1, T5: PICTURE4(4).Visible = True
   CUADRO(16).Visible = True
   Call movi(L1, L2, 0, 16)
Return
DATOHALU:
   '*****#DATO HACIA ALU********
   LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 3675
   Call MOVERDATO(LL1, LL2, TT1, TT2, datos2(FUENTE2))
   LABEL1(8).Caption = msgalu
   LABEL1(8).Visible = True'**MENSAJE DE ALU****
   SendKeys "{break}", True: wait = False
Return
RHDUP:
   '******RESULTADO AL DESTINO***********
   CUADRO(0).Move 3615, 3100
   Call FLECHAS(0, 1, datos2(rami(DESTINO)))'******
   Call movi(3100, 2430, 1, 0)
   Call FLECHAS(0, 2, datos2(rami(DESTINO)))'******
   Call movi(3615, 1350, 0, 0)
   Call FLECHAS(0, 3, datos2(rami(DESTINO)))'******
Return
RHDA:
   '***RESULTADO HACIA DESTINO**RHDA**
   CUADRO(16).Height = 700
   LABEL1(17) = datos2(rami(DESTINO))
RHDABIT:
   CUADRO(0).Move L3, T3: PICTURE4(14).Visible = True
   CUADRO(0).Visible = True
   Call movi(T3, T4, 1, 0)
   PICTURE4(14).Visible = False: PICTURE4(13).Visible = True
   Call movi(L3, L4 + 200, 0, 0)'***HACIA DESTINO****
   PICTURE4(13).Visible = False: PICTURE4(12).Visible = True
   CUADRO(16).Height = 900
Return
PCALU:
   '****PC AL ALU**********
   CUADRO(17).Move 2400, 500
   Call FLECHAS(17, 3, "PC")'******
   Call movi(500, 2500, 1, 17)
Return
RAPC:
   '*********RESULTADO AL PC***********
   CUADRO(17).Width = 720
   LABEL1(24).Caption = "PC"
   CUADRO(21).Move 3200, 3400
   PICTURE4(20).Visible = True
   CUADRO(21).Visible = True
   Call movi(3400, 900, 1, 21)
   LABEL1(10).Caption = PCS(DATOS4(pc))
   Call pausa(espera)
Return
ADPTRALU:
   picture9.Visible = True  '*****ALU***********
   GoSub DHALU '****ACC AL ALU
   msgalu = datos2(rami(acc)) & "+" & DATOS4(dptr)
   '****************DPTR AL ALU*******
   LABEL1(19).Caption = DATOS4(dptr) '*******
   CUADRO(18).Move 1800, 4980: PICTURE6(2).Visible = True
   CUADRO(18).Visible = True
   Call movi(1800, 2100, 0, 18)
   LABEL1(8).Caption = msgalu
   LABEL1(8).Visible = True
   Call pausa(1)
Return
APUNTAROMix:
   '*************APUNTAR ROMIX*************
   PICTURE7(0).Move 8460, 1695  'PUNTERO
   LABEL1(13).Caption = ETIQ
   LABEL1(14).Caption = DATOS4(ORIGEN) & "H"
   PICTURE7(0).Visible = True
```

```
    SendKeys "{break}", True: wait = False
    BD = ORIGEN 'CARGO EL BUS DE DIRECCIONES
    For I = 1 To 20  'TOTAL=# DE DATOS A ESCRIBIR EN ROM
            Call BUSCAR(BD + I - 1)
            If CLng(BD + I - 1) <= 65535 Then
               GRID2(2).AddItem datos2(ROMIX(FILA, COLUMNA)) & Chr(9) & DATOS4(BD + I - 1), I 10 + i
            Else
               GRID2(2).AddItem " " & Chr(9) & " ", I 10 + I
            End If
    Next I
    For I = 1 To 20  'TOTAL=# DE DATOS A ESCRIBIR EN ROM
            GRID2(2).RemoveItem 11
    Next I
    GRID2(2).TopRow = 1
Return
RELHALU:
    '*****RELATIVO HACIA ALU********
    LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 3675
    Call MOVERDATO(LL1, LL2, TT1, TT2, datos2(FUENTE2))
    CUADRO(15).ZOrder 0
    Call movi(5685, 5300, 0, 15)
    LABEL1(8).Caption = msgalu
    LABEL1(8).Visible = True'***MENSAJE DE ALU****
    SendKeys "{break}", True: wait = False
    Call pausa(espera)
Return
FINAL8:
End Sub

Sub FLECHAS (FLECHA, direccion, LETRA)
Select Case FLECHA
Case 3
    LABEL1(18).Caption = LETRA
    INDIC = Choose(direccion, 18, 17, 19, 16)
    For I = 1 To 4
            INDIC2 = Choose(I, 18, 17, 19, 16)
            If CInt(INDIC2) <> CInt(INDIC) Then
               PICTURE4(INDIC2).Visible = False
            End If
    Next I
    CUADRO(3).Visible = True
    PICTURE4(INDIC).Visible = True
    SendKeys "{break}", True: wait = False

Case 0
    LABEL1(17).Caption = LETRA
    INDIC = Choose(direccion, 12, 13, 14, 15)
    For I = 1 To 4
            INDIC2 = Choose(I, 12, 13, 14, 15)
            If CInt(INDIC2) <> CInt(INDIC) Then
               PICTURE4(INDIC2).Visible = False
            End If
    Next I
    CUADRO(0).Visible = True
    PICTURE4(INDIC).Visible = True
    SendKeys "{break}", True: wait = False

Case 15
    LABEL1(6).Caption = LETRA
    INDIC = Choose(direccion, 0, 5, 6, 7)
    For I = 1 To 4
            INDIC2 = Choose(I, 0, 5, 6, 7)
            If CInt(INDIC2) <> CInt(INDIC) Then
               PICTURE4(INDIC2).Visible = False
            End If
    Next I
    CUADRO(15).Visible = True
    PICTURE4(INDIC).Visible = True
    SendKeys "{break}", True: wait = False
Case 16
    LABEL1(15).Caption = LETRA
    INDIC = Choose(direccion, 1, 2, 3, 4)
    For I = 1 To 4
```

```
            INDIC2 = Choose(I, 1, 2, 3, 4)
            If CInt(INDIC2) <> CInt(INDIC) Then
               PICTURE4(INDIC2).Visible = False
            End If
   Next I
   CUADRO(16).Visible = True
   PICTURE4(INDIC).Visible = True
   SendKeys "{break}", True: wait = False
Case 17
   LABEL1(16).Caption = LETRA
   INDIC = Choose(direccion, 8, 9, 10, 11)
   For I = 1 To 4
            INDIC2 = Choose(I, 8, 9, 10, 11)
            If CInt(INDIC2) <> CInt(INDIC) Then
               PICTURE4(INDIC2).Visible = False
            End If
   Next I
   CUADRO(17).Visible = True
   PICTURE4(INDIC).Visible = True
   SendKeys "{break}", True: wait = False
End Select

End Sub

Sub Form_KeyPress (keyascii As Integer)
If keyascii = 27 Then
   posarch = True
End If
End Sub

Sub Form_Load ()
Close
'*********asignacion dedirecciones DE RAM INTERNA*******************
acc = &HE0: B = &HF0: psw = &HD0: SP = &H81: DPL = &H82
dpH = &H83: P0 = &H80: P1 = &H90: P2 = &HA0: p3 = &HB0
IP = &HB8: IE = &HA8: TMOD = &H89: TCON = &H88: T2CON = &HC8
th0 = &H8C: tl0 = &H8A: th1 = &H8D: tl1 = &H8B: TH2 = &HCD
TL2 = &HCC: rcap2h = &HCB: rcap2l = &HCA: scon = &H98
SBUF = &H99: PCON = &H87
r0 = 0: r1 = 1: r2 = 2: r3 = 3: r4 = 4: r5 = 5: r6 = 6: R7 = 7
poner = "si"
FINTE(0) = "NO": FINTE(1) = "NO"
disco = "c:\VGI\"
Open disco + "REGISTRO.DAT" For Input Access Read As #1
For I = 0 To 383
   K(I) = " "CONTIENE LAS ETIQUETAS DE RAM INTERNA
Next I
For I = 1 To 58
   Line Input #1, DATO
   numer = CVar(Mid$(DATO, 9, 3))
   K(numer) = Mid$(DATO, 1, 6)
Next I
Close #1

acerca.Show 'Muestra pantalla de creditos
screen.MousePointer = 11'puntero de mouse en forma de reloj
SendKeys "{break}", True: wait = False
TIMER1.Interval = 1000
FORM1.Cls
For I = 1 To 16
   pines(I).Left = 10 'ubica los pines en la pantalla
Next I
PICTURE12.Left = 7440
PICTURE12.Top = 180
PICTURE12.BorderStyle = 0
FRAME1.Left = 2500
FRAME1.Top = 30
FRAME1.Height = 5295
FRAME1.Width = 5385
FORM1.WindowState = 2 'maximizo pantalla
frame2.Left = -120
frame2.Top = -435
frame2.Height = 7050
```

```
frame2.Width = 9690
CUADRO(4).Left = 765
CUADRO(4).Top = 465
CUADRO(4).Width = 5040
CUADRO(4).Height = 6300'6465
picture9.Left = 3210
picture9.Top = 3345
picture9.Width = 1800
picture9.Height = 2100
CUADRO(15).BorderStyle = 0
CUADRO(16).BorderStyle = 0
CUADRO(3).BorderStyle = 0
CUADRO(17).BorderStyle = 0
CUADRO(0).BorderStyle = 0
CUADRO(19).BorderStyle = 0
CUADRO(20).BorderStyle = 0
CUADRO(21).BorderStyle = 0
'**********Dimensionamiento de Grillas*********************
GRID2(0).AddItem "Dato" & Chr(9) & "Direc", 0
GRID2(2).AddItem "Dato" & Chr(9) & "Direc", 0
GRID2(3).AddItem " " & Chr(9) & "Direc" & Chr(9) & "Dato", 0
GRID2(4).AddItem "Direc" & Chr(9) & "Dato", 0
Call previo2(5, 0, 1, 0, 0, "Direc" & Chr(9) & "  Dato")
For I = 0 To 5
   GRID2(I).FixedRows = 1
Next I
GRID2(1).FixedRows = 0
For I = 0 To 255
   Call previo2(3, 0, 1, I + 1, I + 1, CStr(K(I)) & Chr$(9) & datos2(I))
Next I
For I = 0 To 127
   Call previo2(4, 0, 0, I + 1, I + 1, datos2(128 + I))
Next I

For J = 0 To 5
   For I = 0 To 1
         GRID2(J).ColWidth(I) = 750' 550 + 950 * I
   Next I
Next J
GRID2(5).ColWidth(0) = 750
GRID2(5).ColWidth(1) = 830
GRID2(3).ColWidth(0) = 900
GRID2(3).ColWidth(1) = 550
GRID2(3).ColWidth(2) = 550
GRID2(1).ColWidth(0) = 700
GRID2(1).ColWidth(1) = 400
GRID2(1).ColWidth(2) = 400
GRID2(1).ColWidth(3) = 400
For I = 0 To 7
   GRID2(6).ColWidth(I) = 182
   GRID2(6).ColAlignment(I) = 2
   GRID2(7).ColWidth(I) = 182
   GRID2(7).ColAlignment(I) = 2
Next I
For I = 7 To 0 Step -1
   Call previo3(6, I, 0, " ")'
   Call previo3(7, I, 0, " ")'
Next I
For I = 0 To 4
   GRID2(I).ColAlignment(1) = 0
Next I
GRID2(5).ColAlignment(1) = 2

Open disco + "inicial.set" For Output Access Write As #3
Call reseteo
'*************PANTALLA DE TIMERS*****
Call previo2(1, 0, 0, 1, 4, "TH" & Chr(13) & "TL" & Chr(13) & "TF" & Chr(13) & "EXF2")
Call previo2(1, 0, 3, 0, 0, "TIMER" & Chr(9) & "0" & Chr(9) & "1" & Chr(9) & "2")
Call BIN(rami(TCON), NUMX())
Call previo2(1, 1, 1, 1, 4, datos2(rami(th0)) & Chr(13) & datos2(rami(tl0)) & Chr(13) & NUMX(5))
Call previo2(1, 2, 2, 1, 4, datos2(rami(th1)) & Chr(13) & datos2(rami(tl1)) & Chr(13) & NUMX(7))
Call BIN(rami(T2CON), NUMX())
```

```
Call previo2(1, 3, 3, 1, 4, datos2(rami(TH2)) & Chr(13) & datos2(rami(TL2)) & Chr(13) & CStr(NUMX(7)) & Chr(13) &
CStr(NUMX(6)))
GRID2(1).FixedRows = 1
GRID2(1).FixedCols = 1
'**********ACTUALIZO PATALLA DE INTERRUPCIONES*********
TIT1$ = "FUENTE" & Chr(13) & "INT0" & Chr(13) & "TIMER0" & Chr(13) & "INT1" & Chr(13) & "TIMER1" & Chr(13) & "PS" &
Chr(13) & " " & Chr(13) & "TIMER2" & Chr(13) & " "
tit2$ = "FLAG" & Chr(13) & "IE0" & Chr(13) & "TF0" & Chr(13) & "IE1" & Chr(13) & "TF1" & Chr(13) & "RI" & Chr(13) & "TI" &
Chr(13) & "TF2" & Chr(13) & "EXF2"
Call BIN(rami(TCON), NUMX())
Call BIN(rami(T2CON), numy())
Call BIN(rami(scon), numz())
tit3$ = " " & Chr(13) & NUMX(1) & Chr(13) & NUMX(5) & Chr(13) & NUMX(3) & Chr(13) & NUMX(7) & Chr(13) & NUMX(0) &
Chr(13) & numz(1) & Chr(13) & numy(7) & Chr(13) & numy(6)
Call previo2(8, 0, 0, 0, 8, TIT1$)
Call previo2(8, 1, 1, 0, 8, tit2$)
Call previo2(8, 2, 2, 0, 8, tit3$)
GRID2(8).FixedRows = 1
GRID2(8).ColWidth(0) = 780
acerca.Hide 'oculto pantalla de creditos
screen.MousePointer = 0  'mouse normal
End Sub

Sub Grid2_Click (index As Integer)
Select Case index
Case 3 'RAM INTERNA CON SFR
   GRID2(3).ZOrder 0
   VSCROLL1(3).ZOrder 0
   LABEL1(27).Caption = "RAM INTERNA"
Case 4 'RAM INTERNA CON DIRECCIONAMIENTO INDIRECTO
   GRID2(4).ZOrder 0
   VSCROLL1(4).ZOrder 0
   LABEL1(27).Caption = "AREA DIRECCIONAMIENTO INDIRECTO"
End Select
End Sub

Sub Grid2_MouseMove (index As Integer, Button As Integer, Shift As Integer, x As Single, Y As Single)
Select Case index
Case 3
   GRID2(3).LeftCol = 0

Case 4
   GRID2(4).LeftCol = 0
End Select
End Sub

Sub hexdec (NUMM, NUMD)
'****TRANSFORMACION DE HEXADECIMAL A decimal*******
NUMH$ = CStr(NUMM)
NUMD = 0
If NUMH$ <> "0" Then
   ABC$ = "ABCDEF": NUMD = 0
   For I = 1 To Len(NUMH$)
            If InStr(ABC$, Mid$(NUMH$, I, 1)) <> 0 Then
               EX = 9 + InStr(ABC$, Mid$(NUMH$, I, 1))
            Else
               If IsNumeric(Mid$(NUMH$, I, 1)) Then
                        EX = CVar(Mid$(NUMH$, I, 1))
               Else
                        NUMD = "NO"
                        Exit For
               End If
            End If
            NUMD = (16 ^ (Len(NUMH$) - I)) * EX + NUMD
   Next I
Else
   NUMD = 0
End If
End Sub

Sub INCDEC (SIGNO)
CM = 1
mensaje$ = Switch(SIGNO = 1, "+1", SIGNO = -1, "-1")
```

```
aumento = Switch(SIGNO = 1, 0, SIGNO = -1, &H10)
Select Case MD
  Case "A"
        CODIGO(1) = &H4 + aumento
        Call CODIGOS(1): INDI = 5
        poner = "NO"
        Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
        Call MENSAJES(13 + aumento * INDI / 16)
        GoSub ejecutar4
  Case "Rn"
        CODIGO(1) = &H8 + N + aumento
        Call CODIGOS(1): INDI = 5
        poner = "NO"
        Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
        Call MENSAJES(14 + aumento * INDI / 16)
        GoSub ejecutar4
  Case "@Ri"
        CODIGO(1) = &H6 + N + aumento
        Call CODIGOS(1): INDI = 6
        Call PREVIO(datos2(EXTRAS), datos2(DESTINO), " ", " ")
        Call MENSAJES(16 + aumento * 5 / 16)
        GoSub ejecutar45
  Case "DPTR"
        CM = 2
        CODIGO(1) = &HA3
        Call CODIGOS(1)
        poner = "NO"
        Call PREVIO("DPH", datos2(rami(dpH)), " ", " ")
        Call previo3(5, 0, 2, "DPL")
        Call previo3(5, 1, 2, datos2(rami(DPL)))
        Call MENSAJES(17)
        dptr = dptr + 1
        If CLng(dptr) > 65535 Then
          dptr = 0
        End If
        Call hexdec(Left$(DATOS4(dptr), 2), rami(dpH))
        Call hexdec(Right$(DATOS4(dptr), 2), rami(DPL))
        GRID2(3).TopRow = DPL + 1
        GoSub INDPTR
  Case "DIREC"
        CODIGO(1) = &H5 + aumento: CODIGO(2) = DESTINO
        Call CODIGOS(2): INDI = 5
        poner = "NO"
        Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
        Call MENSAJES(15 + aumento * INDI / 16)
        GoSub ejecutar4
  Case "SP"
        DESTINO = SP: FUENTE1 = rami(SP)
        poner = "NO"
        Call PREVIO("SP", datos2(FUENTE1), " ", " ")
        GRID2(3).TopRow = DESTINO + 1
        picture9.Visible = True '*****ALU**********
        rami(DESTINO) = FUENTE1 + 1 * SIGNO'***INC O DEC 1****
        GoSub desborde
        msgalu = datos2(FUENTE1) & mensaje$
        Call EJECVISUAL(5)
        Call previo3(5, 1, 1, datos2(rami(DESTINO))) 'ACTUALIZA EL ACUMULADOR
        Call banderas
        Call pausa(espera)
        '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
        Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO))) 'ACTUALIZA EL ACUMULADOR
        Call REGRESO(False)
Case Else
   MsgBox "INSTRUCCION NO RECONOCIDA"
End Select
GoTo FIN5
ejecutar4:
  picture9.Visible = True '*****ALU**********
  rami(DESTINO) = FUENTE1 + 1 * SIGNO 'EJECUTO*INC O DEC 1****
  GoSub desborde
  msgalu = datos2(FUENTE1) & mensaje$
  Call EJECVISUAL(INDI)
  Call previo3(5, 1, 1, datos2(rami(DESTINO)))
```

```
   Call banderas
   Call pausa(espera)
   '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
   Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO))) 'ACTUALIZA EL ACUMULADOR
   GRID2(3).TopRow = DESTINO + 1
Return
ejecutar45:   'INC @Ri
   GRID2(3).Visible = False
   picture9.Visible = True
   GRILLA = Switch(DESTINO > 127, 4, DESTINO < 128, 3)
   CELDA = Switch(DESTINO > 127, DESTINO - 127, DESTINO < 128, DESTINO + 1)
   DESTINO = Switch(DESTINO > 127, 128 + DESTINO, DESTINO < 128, DESTINO)
   rami(DESTINO) = FUENTE1 + 1 * SIGNO'INC O DEC 1
   GoSub desborde
   msgalu = datos2(FUENTE1) & mensaje$
   Call EJECVISUAL(INDI) '*****PANTALLA****
   Call previo3(5, 1, 8, datos2(rami(DESTINO)))
   Call banderas
   Call pausa(espera)
   '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
   Call previo3(GRILLA, -GRILLA + 5, CELDA, datos2(rami(DESTINO)))'ACTUALIZA EL ACUMULADOR
   GRID2(GRILLA).TopRow = CELDA
   GRID2(GRILLA).ZOrder 0
   Return
INDPTR:
   picture9.Visible = True  '*****ALU***********
   msgalu = DATOS4(dptr - 1) & mensaje$
   LABEL1(19).Caption = DATOS4(dptr - 1)'*******
   CUADRO(18).Move 1800, 3465: PICTURE6(2).Visible = True
   CUADRO(18).Visible = True
   Call movi(1800, 3170, 0, 18)
   LABEL1(8).Caption = msgalu
   LABEL1(8).Visible = True
   Call pausa(espera)
   LABEL1(19).Caption = DATOS4(dptr) '*******
   CUADRO(18).Move 3170, 3465: PICTURE6(2).Visible = False
   PICTURE6(1).Visible = True
   CUADRO(18).Visible = True
   Call movi(3170, 2050, 0, 18)
   Call previo3(5, 1, 1, datos2(rami(dpH)))
   Call previo3(5, 1, 2, datos2(rami(DPL)))
   Call pausa(espera)
   '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
   Call previo3(3, 2, dpH + 1, datos2(rami(dpH))) 'ACTUALIZA EL ACUMULADOR
   Call previo3(3, 2, DPL + 1, datos2(rami(DPL))) 'ACTUALIZA EL ACUMULADOR
   GRID2(3).TopRow = DPL + 1
   Call pausa(espera)
   CUADRO(18).Visible = False: PICTURE6(1).Visible = False

Return
desborde:
   If rami(DESTINO) > 255 Then
         rami(DESTINO) = 0
   End If
   If rami(DESTINO) < 0 Then
         rami(DESTINO) = 255
   End If
Return
FIN5:
End Sub

Sub ingresos ()
'PERMITE AL USUARIO ESCOJER EL Rn,@Ri o
'INGRESAR DIREC,#DATO,RELATIVO,bit,/bit
REPITA:
fuente$ = Choose(ii, "DIRECCION DESTINO", "DIRECCION ORIGEN", " ", " ")
If ii < 4 Then 'este lazo se repite hasta leer como maximo 3 operandos
   OPERANDO$ = OP(ii)
   Select Case OPERANDO$
         Case "Rn"          '
            CUADRO(6).Visible = True
         Case "@Ri"     '*******
            CUADRO(7).Visible = True
```

```
            Case "DIRECCION"    '***************
               COMMAND4(12).Visible = True
               COMMAND4(12).Default = True
               CUADRO(8).Visible = True
               text1.SetFocus
               label2(6).Caption = fuente$
            Case "#DATO", "rel", "BIT", "/BIT", "DIREC11", "DIREC16", "#DATO16"
               COMMAND4(12).Visible = True
               COMMAND4(12).Default = True'SetFocus
               CUADRO(8).Visible = True
               text1.SetFocus
               label2(6).Caption = OPERANDO$
            Case Else 'el operando es una etiqueta Ej. A,DPTR
               ii = ii + 1
               GoTo REPITA
   'excepto case else continua en command4
   End Select
Else 'se han leido todos los operandos (ej. Rn=R0,#dato=12)
   If OP(3) <> "" Then
            coma = ",": coma2 = ","
   Else
            If OP(2) <> "" Then
               coma = "": coma2 = ","
            End If
   End If
   label2(2).Caption = OPERACION$ & " " & OP(1) & coma2 & OP(2) & coma & OP(3)
   label2(2).Visible = True'se muestra la instruccion a ejecutar
   Command1(2).Visible = True
   Command1(2).Enabled = True
   Command1(2).SetFocus
End If
End Sub

Sub LOGICAS_Click (index As Integer)
barra(0).Enabled = False
barra(1).Enabled = False
barra(2).Enabled = False
Select Case index
Case 0
   Call MODO1(25, 30, "ANL")
Case 1
   Call MODO1(31, 36, "ORL")
Case 2
   Call MODO1(37, 42, "XRL")
Case 3
  Call MODO1(43, 43, "CLR")
Case 4
  Call MODO1(44, 44, "CPL")
Case 5
  Call MODO1(45, 45, "RL")
Case 6
  Call MODO1(46, 46, "RLC")
Case 7
  Call MODO1(47, 47, "RR")
Case 8
  Call MODO1(48, 48, "RRC")
Case 9
   Call MODO1(49, 49, "SWAP")
End Select
End Sub

Function LSB (num3)
LSB = num3 - (Fix(num3 / 16)) * 16 'BYTE MENOS SIGNIFICATIVO EN datosADECIMAL
End Function

Sub MENSAJES (lugar%)
'******MUESTRA LOS MENSAJES DURANTE LA EJECUCION **********
   TEXT2.Enabled = True
   Open disco + "mensajes.DAT" For Input Access Read As #4
   For I = 1 To lugar + 1
            Line Input #4, DATO
            If I = lugar Then
               TEXT2.Text = "Mensaje:" & DATO
```

```
            End If
    Next I
    Close #4
    TEXT2.Enabled = False
    Call pausa(espera * 2)
    SendKeys "{break}", True: wait = False
End Sub

Sub MODO1 (saltos, TOTAL, NOMBRE$)
label2(1).Caption = NOMBRE$
'******HACE UN DISPLAY DE LOS MD DISPONIBLES
Open disco + "ARIT.DAT" For Input Access Read As #2
For I = 1 To TOTAL
    Line Input #2, IN
            If I >= saltos Then
               COMBO2.AddItem IN
            End If
Next I
Close #2
FRAME1.Visible = True
PICTURE12.ZOrder 1
SendKeys "{break}", True: wait = False
'*****CONTINUA EN COMMAND1 (ACEPTAR)
End Sub

Sub MODOS ()
Dim BIT$, DATO$, banco%
Call BIN(rami(psw), NUMX())
banco = 8 * (CInt(NUMX(4)) * 2 + CInt(NUMX(3)))'determino
                                'banco de registros activo RS1,RS0
N = 0: MD = ""

If Len(OP(1)) > 0 Then
Select Case OP(1)
Case "R0", "R1", "R2", "R3", "R4", "R5", "R6", "R7"
            N = CVar(Mid$(OP(1), 2, 1))
            DESTINO = banco + Choose(N + 1, r0, r1, r2, r3, r4, r5, r6, R7)
            FUENTE1 = rami(DESTINO)
            MD = "Rn"
Case "@R0", "@R1"
            N = CVar(Mid$(OP(1), 3, 1))
            EXTRAS = banco + Choose(N + 1, r0, r1)
            DESTINO = rami(EXTRAS): EXTRAS2 = DESTINO
            FUENTE1 = Switch(DESTINO > 127, rami(128 + DESTINO), DESTINO < 128, rami(DESTINO))
            MD = "@Ri"
Case "A"
   DESTINO = acc: FUENTE1 = rami(DESTINO)
   MD = "A"
Case "AB"
   MD = "AB"
Case "DPTR"
   DESTINO = dptr: ' FUENTE1 = DPTR
   MD = "DPTR"
Case "@DPTR"
   DESTINO = dptr: EXTRAS2 = dptr
   MD = "@DPTR"
Case "C"
   DESTINO = psw: FUENTE1 = rami(psw)
   MD = "C"
Case "@A+DPTR"    'SOLO PARA JMP
   DESTINO = rami(acc) + dptr:
   If CLng(DESTINO) > 65535 Then
            DESTINO = DESTINO - 65536
   End If
   MD = "@A+DPTR"
Case OP(1)  'DIRECCION1
   Select Case OPERACION$
   Case "CLR", "SETB", "CPL", "JB", "JNB", "JBC"
            Call DECIBOOL(OP(1), DESTINO, FUENTE1)
   Case "MOV"
            If OP(2) = "C" Then
               Call DECIBOOL(OP(1), DESTINO, FUENTE1)
            Else
```

```
            Call DECIFRAR(OP(1), DESTINO, FUENTE1)
        End If
    Case Else 'INSTRUCCION NO BOOLEANA
        Call DECIFRAR(OP(1), DESTINO, FUENTE1)
    End Select
End Select
Else
    MD = "NADA"
End If
If Len(OP(2)) > 0 Then
BIT = Mid$(OP(2), 2, Len(OP(2)) - 1) 'CUANDO SE TRATA DE /BIT
DATO = Mid$(OP(2), 2, Len(OP(2)) - 1)
Select Case OP(2)
Case "R0", "R1", "R2", "R3", "R4", "R5", "R6", "R7"
        N = CVar(Mid$(OP(2), 2, 1))
        ORIGEN = banco + Choose(N + 1, r0, r1, r2, r3, r4, r5, r6, R7)
        FUENTE2 = rami(ORIGEN)
        MD = MD + ",Rn"
Case "@R0", "@R1"
        N = CVar(Mid$(OP(2), 3, 1))
        EXTRAS = banco + Choose(N + 1, r0, r1)
        ORIGEN = rami(EXTRAS): EXTRAS2 = ORIGEN
        FUENTE2 = Switch(ORIGEN > 127, rami(128 + ORIGEN), ORIGEN < 128, rami(ORIGEN))
        MD = MD + ",@Ri"
Case "A"
    ORIGEN = acc: FUENTE2 = rami(ORIGEN)
    MD = MD + ",A"
Case "C"
    ORIGEN = psw: FUENTE2 = 7
    MD = MD + "C"
Case "#" & DATO
    MD = MD + ",#"
    Call DECIFRAR(DATO, ORIGEN, FUENTE2)
    FUENTE2 = ORIGEN
Case "@DPTR"
    ORIGEN = dptr: EXTRAS2 = dptr
    MD = MD + ",@DPTR"
Case "@A+DPTR"
    ORIGEN = rami(acc) + dptr:
    If CLng(ORIGEN) > 65535 Then
        ORIGEN = ORIGEN - 65536
    End If
    Call BUSCAR(ORIGEN)
    FUENTE2 = ROMIX(FILA, COLUMNA)
    MD = "MOVC@A+DPTR"
Case "@A+PC"
    ORIGEN = rami(acc) + pc + 1:
    If CLng(ORIGEN) > 65535 Then
        ORIGEN = ORIGEN - 65536
    End If
    Call BUSCAR(ORIGEN)
    FUENTE2 = ROMIX(FILA, COLUMNA)
    MD = "MOVC@A+PC"
Case "/" & BIT
        MD = MD + "/"
        Call DECIBOOL(BIT, ORIGEN, FUENTE2)
        If SEGUIR = "NO" Then 'NO EXISTE ESE OPERANDO
            'MANDAR MENSAJE
        End If

Case OP(2)  'DIRECCION2
  If OP(1) = "C" Then 'ABARCA ANL,ORL,MOV  PERO BOOLEANAS
        Call DECIBOOL(OP(2), ORIGEN, FUENTE2)
  Else  'ES INSTRUCCION NO BOOLEANA CON EXEPCION DE JC,JNC
        MD = MD + ","
        Call DECIFRAR(OP(2), ORIGEN, FUENTE2)
        If CInt(ORIGEN) = &H90 Then 'puerto1
            FUENTE2 = 0
            For mm = 0 To 7
                FUENTE2 = FUENTE2 + (2 ^ mm) * pines(9 + mm).Value
            Next mm
        Else
            If CInt(ORIGEN) = &HB0 Then 'puerto  3
```

```
                    FUENTE2 = 0
                    For mm = 0 To 7
                      FUENTE2 = FUENTE2 + (2 ^ mm) * pines(mm).Value
                    Next mm
                End If
            End If

    End If
End Select
End If
If Len(OP(3)) > 0 Then
    MD = MD + ","
    Call DECIFRAR(OP(3), REL, FUENTER)
    If SEGUIR = "NO" Then 'VEO SI ES BIT
            'MANDAR MENSAJE
    Else
    End If

End If

FINY:
End Sub

Sub MOVER ()
Dim L1, L2, L3, T1, T2
CM = 1
L1 = 1710: L2 = 2250: L3 = 1950: T1 = 3195: T2 = 4980
DONDE2 = 8
GRILLA = 3: CELDA = DESTINO + 1
Select Case MD
Case "A,Rn"
    CODIGO(1) = &HE8 + N
    Call CODIGOS(1)
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(50)
    GoSub MOVIMIENTO
Case "A,DIREC"
    CODIGO(1) = &HE5: CODIGO(2) = ORIGEN
    Call CODIGOS(2)
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(51)
    GoSub MOVIMIENTO
Case "A,@Ri"
    CODIGO(1) = &HE6 + N
    Call CODIGOS(1)
    Call PREVIO(datos2(EXTRAS), datos2(rami(EXTRAS)), datos2(ORIGEN), datos2(FUENTE2))
    Call MENSAJES(52)
    Call EJECVISUAL(4)
    T1 = 3975
    GoSub MOVIMIENTO
Case "A,#DIREC"
    CODIGO(1) = &H74: CODIGO(2) = ORIGEN
    Call CODIGOS(2)
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    Call MENSAJES(53)
    donde = 1
    LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 3675: TT3 = 3195
    GoSub MOVIMIENTO2
Case "Rn,A"
    CODIGO(1) = &HF8 + N
    Call CODIGOS(1)
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(54)
    GoSub MOVIMIENTO
Case "Rn,direc"
    CM = 2
    CODIGO(1) = &HA8 + N: CODIGO(2) = ORIGEN
    Call CODIGOS(2)
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(55)
    GoSub MOVIMIENTO
Case "Rn,#DIREC"
```

```
    CODIGO(1) = &H78 + N: CODIGO(2) = ORIGEN
    Call CODIGOS(2)
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    Call MENSAJES(56)
    donde = 1
    LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 3675: TT3 = 3195
    GoSub MOVIMIENTO2
Case "DIREC,A"
    CODIGO(1) = &HF5: CODIGO(2) = DESTINO
    Call CODIGOS(2)
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(57)
    GoSub MOVIMIENTO
Case "direc,Rn"
    CM = 2
    CODIGO(1) = &H88 + N: CODIGO(2) = DESTINO
    Call CODIGOS(2)
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(58)
    GoSub MOVIMIENTO
Case "direc,direc"
    CM = 2
    CODIGO(1) = &H85: CODIGO(2) = ORIGEN: CODIGO(3) = DESTINO
    Call CODIGOS(3)
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(59)
    GoSub MOVIMIENTO
Case "DIREC,@Ri"
    CM = 2
    CODIGO(1) = &H87 + N: CODIGO(2) = DESTINO
    Call CODIGOS(2)
    Call PREVIO(datos2(EXTRAS), datos2(rami(EXTRAS)), datos2(ORIGEN), datos2(FUENTE2))
    Call MENSAJES(60)
    Call EJECVISUAL(4) 'PUNTERO
    T1 = 3975
    GoSub MOVIMIENTO
Case "DIREC,#DIREC"
    CM = 2
    CODIGO(1) = &H75: CODIGO(2) = DESTINO: CODIGO(3) = ORIGEN
    Call CODIGOS(3)
    poner = "NO"
    Call PREVIO(datos2(DESTINO), datos2(FUENTE1), " ", " ")
    Call MENSAJES(61)
    donde = 1
    LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 3675: TT3 = 3195
    GoSub MOVIMIENTO2
Case "@Ri,A"
    CODIGO(1) = &HF6 + N
    Call CODIGOS(1)
    Call PREVIO(datos2(EXTRAS), datos2(DESTINO), datos2(ORIGEN), datos2(FUENTE2))
    Call MENSAJES(62)
    Call EJECVISUAL(7) 'PUNTERO
    T1 = 3975
    GoSub ARROBA
    GoSub MOVIMIENTO
Case "@Ri,DIREC"
    CM = 2
    CODIGO(1) = &HA6 + N: CODIGO(2) = ORIGEN
    Call CODIGOS(2)
    Call PREVIO(datos2(EXTRAS), datos2(DESTINO), datos2(ORIGEN), datos2(FUENTE2))
    Call MENSAJES(63)
    Call EJECVISUAL(7) 'PUNTERO
    T1 = 3975
    GoSub ARROBA
    GoSub MOVIMIENTO
Case "@Ri,#DIREC"
    CODIGO(1) = &H76 + N: CODIGO(2) = ORIGEN
    Call CODIGOS(2)
    poner = "NO"
    Call PREVIO(CStr(datos2(EXTRAS)), datos2(DESTINO), " ", " ")
    Call previo3(5, 0, 8, datos2(DESTINO))
    Call previo3(5, 1, 8, datos2(FUENTE1))
```

```
    donde = 8
    LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 5460: TT3 = 4980
    Call MENSAJES(64)
    Call EJECVISUAL(7)          'PUNTERO
    GoSub ARROBA
    GoSub MOVIMIENTO2
Case "DPTR,#DIREC", "DPTR,#DIREC11", "DPTR,#DIREC16"
    CM = 2
    Call BIN(ORIGEN, NUMX())
    CUADRO(18).Move 4215, 3465:
    CUADRO(18).Visible = True
    LABEL1(19).Caption = DATOS4(dptr)'*******
    OPERACION$ = "MOVEDPTR#"
    '*************** CODIGOS************
    CODIGO(1) = &H90: CODIGO(2) = 0: CODIGO(3) = 0
    For I = 0 To 7
            CODIGO(2) = CODIGO(2) + NUMX(I + 8) * (2 ^ I)
            CODIGO(3) = CODIGO(3) + NUMX(7 - I) * (2 ^ (7 - I))
    Next I
    Call CODIGOS(3)
    '*****LOCALIDADES************
    poner = "NO"
    Call PREVIO("DPH", datos2(rami(dpH)), " ", " ")
    Call MENSAJES(65)
    Call previo3(5, 0, 2, "DPL")
    Call previo3(5, 1, 2, datos2(rami(DPL)))
    Call MENSAJES(65)
    '******TRASLADO DE #DATA16 A DPH Y DPL
    PICTURE6(1).Visible = True 'PUNTA IZQ DEL DPTR
    Call movi(4215, 2160, 0, 18)'MOV DEL DPTR
    rami(dpH) = CODIGO(2): rami(DPL) = CODIGO(3)
    Call previo3(5, 1, 1, datos2(CODIGO(2)))
    Call previo3(5, 1, 2, datos2(CODIGO(3)))
    Call pausa(espera)
    '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
    Call previo3(3, 2, dpH + 1, datos2(rami(dpH))) 'ACTUALIZA EL ACUMULADOR
    Call previo3(3, 2, DPL + 1, datos2(rami(DPL))) 'ACTUALIZA EL ACUMULADOR
    GRID2(3).TopRow = DPL + 1
    Call pausa(espera)
    CUADRO(18).Visible = False: PICTURE6(1).Visible = False

Case "@SP,PC7-0"
    Call PREVIO("SP", datos2(rami(SP)), " ", " ")
    Call EJECVISUAL(10)  'PUNTERO
    L1 = INDI: T1 = 600: L2 = L1: L3 = 1950: T2 = 4980
    GoSub ARROBA
    GoSub MOVIMIENTO
    Call REGRESO(False)
Case "@SP,DIREC"
    Call PREVIO("SP", datos2(rami(SP)), datos2(ORIGEN), datos2(FUENTE2))
    Call EJECVISUAL(10) 'PUNTERO
    Call MENSAJES(72)
    T1 = 3975
    GoSub ARROBA
    GoSub MOVIMIENTO
Case "DIREC,@SP"
    Call PREVIO("SP", datos2(rami(SP)), datos2(ORIGEN), datos2(FUENTE2))
    Call EJECVISUAL(9)
    Call MENSAJES(73)
    T1 = 3975
    GoSub MOVIMIENTO
Case "MOVC@A+DPTR"
    CM = 2
    CODIGO(1) = &H93
    Call CODIGOS(1)
    '******LOCALIDADES*****
    poner = "NO"
    Call PREVIO("ACC", datos2(rami(acc)), " ", " ")
    Call MENSAJES(66)
    Call previo3(5, 0, 8, "DPL")
    Call previo3(5, 1, 8, datos2(rami(DPL)))
    Call previo3(5, 0, 7, "DPH")
    Call previo3(5, 1, 7, datos2(rami(dpH)))
```

```
    'MOVIMIENTO ACC Y DPTR AL ALU
    Call MENSAJES(66)
    Call EJECVISUAL(11)
    donde = 1
            LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 3675: TT3 = 3195
            GoSub MOVIMIENTO2 'LLEVA DATO DE ROMX AL ACC
Case "movc@A+PC"
    CM = 2
    CODIGO(1) = &H83: Call CODIGOS(1)
    poner = "NO"
    Call PREVIO("ACC", datos2(rami(acc)), " ", " ")
    Call MENSAJES(67)
    'MOVIMIENTO ACC Y PC AL ALU
    Call EJECVISUAL(12)
    donde = 1
    'TRASLADO DEL DATO DE ROM AL ACC
    LL1 = 6210: LL2 = 5685: TT1 = 1545: TT2 = 3675: TT3 = 3195
    GoSub MOVIMIENTO2 'LLEVA DATO DE ROMX AL ACC
Case "CBIT"
    auxiliar = rami(ORIGEN)
    CODIGO(1) = &HA2: CODIGO(2) = cmbyte
    Call CODIGOS(2)
    FUENTE1 = 7:
    Call MENSAJES(88)
    If CInt(ORIGEN) = &H90 Then 'puerto1
            rami(ORIGEN) = 0
            For mm = 0 To 7
               rami(ORIGEN) = rami(ORIGEN) + (2 ^ mm) * pines(9 + mm).Value
            Next mm
    Else
            If CInt(ORIGEN) = &HB0 Then 'puerto  3
               rami(ORIGEN) = 0
               For mm = 0 To 7
                        rami(ORIGEN) = rami(ORIGEN) + (2 ^ mm) * pines(mm).Value
               Next mm
            End If
    End If
    GoSub SOLOBITS
    rami(ORIGEN) = auxiliar
Case "BITC"
    CM = 2
    CODIGO(1) = &H92: CODIGO(2) = cmbyte
    Call CODIGOS(2)
    FUENTE2 = 7:
    Call MENSAJES(89)
    GoSub SOLOBITS
Case Else
    MsgBox "INSTRUCCION NO RECONOCIDA"
End Select
GoTo FINAL
ARROBA:
    GRILLA = Switch(DESTINO > 127, 4, DESTINO < 128, 3)
    CELDA = Switch(DESTINO > 127, DESTINO - 127, DESTINO < 128, DESTINO + 1)
    DESTINO = Switch(DESTINO > 127, 128 + DESTINO, DESTINO < 128, DESTINO)
Return
SOLOBITS:
    GRID2(5).Cols = 3
    GRID2(5).FixedRows = 0
    Call previo2(5, 0, 2, 0, 0, "Direc" & Chr(9) & "Dato" & Chr(9) & "Bit")
    GRID2(5).FixedRows = 1
    For I = 0 To 2
            GRID2(5).ColWidth(I) = 500
    Next I
    poner = "NO"

    Call PREVIO(datos2(ORIGEN), datos2(rami(ORIGEN)), " ", " ")
    Call BIN(rami(DESTINO), NUMX())
    Call BIN(rami(ORIGEN), numy())'

    Call previo3(5, 0, 8, datos2(DESTINO))
    Call previo3(5, 1, 8, datos2(rami(DESTINO)))

    Call previo3(5, 2, 1, numy(FUENTE2))'
```

```
    Call previo3(5, 2, 8, NUMX(FUENTE1))
    'OPERACION
    NUMX(FUENTE1) = numy(FUENTE2)   'EJECUTO
    Call DECI(NUMX(), rami(DESTINO))'ACTUALIZO BYTE DESTINO
    '******GRAFICACION*****************
    CUADRO(16).Move L1, T1
    Call FLECHAS(16, 4, numy(FUENTE2))
    Call movi(L1, L2, 0, 16)
    Call FLECHAS(16, 3, numy(FUENTE2))
    Call movi(T1, T2, 1, 16)
    Call FLECHAS(16, 2, numy(FUENTE2))
    Call movi(L2, L3, 0, 16)
    Call previo3(5, 2, 8, NUMX(FUENTE1))
    Call previo3(5, 1, 8, datos2(rami(DESTINO)))
    Call pausa(espera)
    '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
    Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO)))
    GRID2(3).TopRow = DESTINO + 1
    LABEL1(8).Caption = " "
    Call pausa(espera)
Return

MOVIMIENTO:
    rami(DESTINO) = FUENTE2   '******ejecuto move********
    CUADRO(16).Move L1, T1
    Call FLECHAS(16, 4, datos2(FUENTE2))
    Call movi(L1, L2, 0, 16)
    Call FLECHAS(16, 3, datos2(FUENTE2))
    Call movi(T1, T2, 1, 16)
    Call FLECHAS(16, 2, datos2(FUENTE2))
    Call movi(L2, L3, 0, 16)
    Call previo3(5, 1, DONDE2, datos2(rami(DESTINO)))
    Call banderas
    LABEL1(8).Caption = " "
    Call pausa(espera)
    GoSub COMUN3
Return

MOVIMIENTO2: '#DATO*************
    rami(DESTINO) = FUENTE2   '******ejecuto move********
    Call MOVERDATO(LL1, LL2, TT1, TT2, datos2(FUENTE2))
    CUADRO(15).ZOrder 0
    Call movi(LL2, 3780, 0, 15)
    CUADRO(16).Move 3000, TT3
    Call FLECHAS(16, 2, datos2(FUENTE2))
    CUADRO(15).Visible = False
    Call movi(3000, 1920, 0, 16)
    Call previo3(5, 1, donde, datos2(rami(DESTINO)))
    Call banderas
    LABEL1(8).Caption = " "
    Call pausa(espera)
    GoSub COMUN3

Return
COMUN3:
    '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
    Call previo3(GRILLA, -GRILLA + 5, CELDA, datos2(rami(DESTINO)))'ACTUALIZA EL DESTINO
    GRID2(GRILLA).TopRow = CELDA
    Call pausa(espera)
    GRID2(GRILLA).ZOrder 0
Return
FINAL:
End Sub

Sub MOVERDATO (L1, L2, T1, T2, LABELL)
    CUADRO(15).ZOrder 1: CUADRO(15).Move L1, T1
    Call FLECHAS(15, 2, LABELL)
    Call movi(L1, L2, 0, 15)
    Call FLECHAS(15, 3, LABELL)
    Call movi(T1, T2, 1, 15)
    Call FLECHAS(15, 2, LABELL)
'***LLEGO DATO AL ALU ************************
```

```
End Sub

Sub MOVERX ()
CM = 2
Select Case MD
Case "A,@Ri"
   CODIGO(1) = &HE2 + N: Call CODIGOS(1)
   GoSub comunx
   Call MENSAJES(68)
   EXTRAS2 = rami(EXTRAS)
   Call BUSCAR(EXTRAS2)
   Call previo3(5, 0, 8, CStr(datos2(EXTRAS)))
   Call previo3(5, 1, 8, CStr(datos2(rami(EXTRAS))))
   GRID2(5).Visible = True: GRID2(5).ZOrder 0
   GRID2(0).TopRow = 9: GRID2(5).TopRow = 1
   Call pausa(espera)
   PICTURE7(0).Move 8460, 3780
   LABEL1(13).Caption = "Ri"
   LABEL1(14).Caption = datos2(ORIGEN) & "H"
   PICTURE7(0).Visible = True
   Call pausa(espera)
   rami(DESTINO) = RAMx(FILA, COLUMNA)'*****EJECUTO******
   GoSub MOVIMIENTOX
Case "A,@DPTR"
   CODIGO(1) = &HE0: Call CODIGOS(1)
   GoSub comunx
   Call MENSAJES(69)
   Call previo3(5, 0, 7, CStr(datos2(dpH)))
   Call previo3(5, 1, 7, CStr(datos2(rami(dpH))))
   Call previo3(5, 0, 8, CStr(datos2(DPL)))
   Call previo3(5, 1, 8, CStr(datos2(rami(DPL))))
   GRID2(5).TopRow = 1
   GRID2(5).Visible = True: GRID2(5).ZOrder 0
   GRID2(0).TopRow = 9
   Call pausa(espera)
   PICTURE7(0).Move 8460, 3780
   LABEL1(13).Caption = "DPTR"
   LABEL1(14).Caption = DATOS4(dptr) & "H"
   PICTURE7(0).Visible = True
   Call pausa(espera)
   rami(DESTINO) = RAMx(FILA, COLUMNA)'*****EJECUTO******
   GoSub MOVIMIENTOX
Case "@Ri,A"
   CODIGO(1) = &HF2 + N: Call CODIGOS(1)
   GoSub comunx
   Call MENSAJES(70)
   Call previo3(5, 0, 8, CStr(datos2(EXTRAS)))
   Call previo3(5, 1, 8, CStr(datos2(rami(EXTRAS))))
   GRID2(5).Visible = True: GRID2(5).ZOrder 0
   GRID2(5).TopRow = 1
   EXTRAS2 = rami(EXTRAS)
   Call BUSCAR(EXTRAS2)
   GRID2(0).TopRow = 9
   Call pausa(espera)
   PICTURE7(0).Move 8460, 3780
   LABEL1(13).Caption = "Ri"
   LABEL1(14).Caption = datos2(DESTINO) & "H"
   PICTURE7(0).Visible = True
   Call pausa(espera)
   RAMx(FILA, COLUMNA) = rami(acc)'*****EJECUTO******
   GoSub MOVIMIENTO2X

Case "@DPTR,A"  'ESCRIBIR EN RAM EXTERNA
   CODIGO(1) = &HF0: Call CODIGOS(1)
   GoSub comunx
   Call MENSAJES(71)
   Call previo3(5, 0, 7, CStr(datos2(dpH)))
   Call previo3(5, 1, 7, CStr(datos2(rami(dpH))))
   Call previo3(5, 0, 8, CStr(datos2(DPL)))
   Call previo3(5, 1, 8, CStr(datos2(rami(DPL))))
   GRID2(5).TopRow = 1
   GRID2(5).Visible = True: GRID2(5).ZOrder 0
   GRID2(0).TopRow = 9
```

```
    Call pausa(espera)
    PICTURE7(0).Move 8460, 3780
    LABEL1(13).Caption = "DPTR"
    LABEL1(14).Caption = DATOS4(dptr) & "H"
    PICTURE7(0).Visible = True
    Call pausa(espera)
    RAMx(FILA, COLUMNA) = rami(acc) '***ejecuto*****
    GoSub MOVIMIENTO2X
  Case Else
     MsgBox "INSTRUCCION NO RECONOCIDA"
  End Select
  GoTo FINALX
  comunx:
  GRID2(3).TopRow = acc + 1
  EXTRAS2 = EXTRAS2 - 1
  For J = 0 To 7
    EXTRAS2 = EXTRAS2 + 1
    Call BUSCAR(EXTRAS2)
    GRID2(0).AddItem datos2(RAMx(FILA, COLUMNA)) & Chr$(9) & DATOS4(EXTRAS2), J + 9
    If CLng(EXTRAS2) > 65535 Then
            GRID2(0).AddItem " " & Chr$(9) & " ", J + 9
    End If
  Next J
  EXTRAS2 = EXTRAS2 - 7
  Call BUSCAR(EXTRAS2)
  poner = "NO"
  Call PREVIO(datos2(acc), datos2(rami(acc)), " ", " ")
  Return
  MOVIMIENTOX:
    CUADRO(15).Move 6270, 3675
    Call FLECHAS(15, 2, datos2(rami(DESTINO)))
    CUADRO(15).ZOrder 1
    Call movi(6270, 5870, 0, 15)
    CUADRO(15).ZOrder 0
    Call movi(5870, 3780, 0, 15)
    CUADRO(16).Move 3000, 3195
    Call FLECHAS(16, 2, datos2(rami(DESTINO)))
    CUADRO(15).Visible = False
    Call movi(3000, 1900, 0, 16) '***LLEGO EL DATO
    GoSub COMUN3X
  Return
  MOVIMIENTO2X:
    CUADRO(16).Move 1710, 3195
    Call FLECHAS(16, 4, datos2(rami(acc)))
    Call movi(1710, 3000, 0, 16)
    CUADRO(15).Move 3780, 3675
    Call FLECHAS(15, 4, datos2(rami(acc)))
    CUADRO(15).ZOrder 0
    CUADRO(16).Visible = False
    picture11(1).ZOrder 0
    Call movi(3780, 6000, 0, 15)
    CUADRO(15).ZOrder 1
    Call previo3(0, 0, 9, datos2(rami(acc))) 'ACTUALIZA EL DESTINO
    SendKeys "{break}", True: wait = False
    Call pausa(espera)
    For I = 0 To 7
            GRID2(0).RemoveItem 1
    Next I
  Return
  COMUN3X:
    Call previo3(5, 1, 1, datos2(rami(DESTINO)))
    Call banderas
    LABEL1(8).Caption = " "
    Call pausa(espera)
    '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
    Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO))) 'ACTUALIZA EL DESTINO
    GRID2(3).TopRow = DESTINO + 1
    Call pausa(espera)
    For I = 0 To 7
            GRID2(0).RemoveItem 1
    Next I
  Return
  FINALX:
```

```
End Sub

Sub movi (INICIO, FIN, SUBIR, INDIC)
'******PERMITE MOVER UNA FLECHA HORIZONTAL O VERTICAL*****
If CInt(INICIO) > CInt(FIN) Then
   paso = -paso
End If
For cont = INICIO To FIN Step paso
If SUBIR = 0 Then
   CUADRO(INDIC).Move cont
Else 'LAS FLECHAS SE MUEVEN HORIZONTALMENTE
   CUADRO(INDIC).Move CUADRO(INDIC).Left, cont
End If
   Call pausa(tiempo)
Next cont
If SUBIR = 0 Then
   CUADRO(INDIC).Move FIN
Else
   CUADRO(INDIC).Move CUADRO(INDIC).Left, FIN
End If
paso = Abs(paso)
End Sub

Sub movi2 (INICIO, FIN, SUBIR, indic1, INDIC2)
If CInt(INICIO) > CInt(FIN) Then
   paso = -paso
End If
For cont = INICIO To FIN Step paso
   If SUBIR = 0 Then
            CUADRO(indic1).Move cont
            CUADRO(INDIC2).Move cont
   Else
            CUADRO(indic1).Move CUADRO(indic1).Left, cont
            CUADRO(INDIC2).Move CUADRO(INDIC2).Left, cont
   End If
   Call pausa(tiempo)
Next cont
If SUBIR = 0 Then
   CUADRO(indic1).Move FIN
   CUADRO(INDIC2).Move FIN
Else
   CUADRO(indic1).Move CUADRO(indic1).Left, FIN
   CUADRO(INDIC2).Move CUADRO(indic1).Left, FIN
End If
paso = Abs(paso)
End Sub

Sub MUESTREO ()
DoEvents
Dim cuenta%, BI%, TF2%, EXF2%
Dim MODO3$, habil$
ReDim FTCON(16), ft2con(16), MSGINT(1) As String
ReDim sai(5) As String, ipx(15)
MODO3 = "NO"

'*************************TIMER0**********************
Call BIN(rami(TMOD), NUMX())
Call BIN(rami(TCON), numy())
habil = CStr(numy(4)) + CStr(NUMX(3)) + CStr(pines(2).Value)
'habil me indica si el timer esta habilitado
M1M0 = CStr(NUMX(1)) + CStr(NUMX(0))'determino el modo
Select Case habil & M1M0
   Case "10000", "10100", "11100"'modo 0 ;13 bits
            cuenta = 32
            GoSub contador
   Case "10001", "10101", "11101" 'modo 1 ; 16 bits
            cuenta = 256
            GoSub contador
   Case "10010", "10110", "11110" 'modo 2  8 bits con autorecarga
            If CInt(NUMX(2)) = 0 Then ' c/t=0 TEMPORIZADOR
               rami(tl0) = rami(tl0) + CM+ 1
               TEXT2.Text = "TIMER0 HABILITADO COMO TEMPORIZADOR"
            Else                    'c/t=1    'CONTADOR
```

```
            rami(tl0) = rami(tl0) + pulso(0)
            pulso(0) = 0
            TEXT2.Text = "TIMER0 HABILITADO COMO CONTADOR"
          End If
          If CInt(rami(tl0)) >= 256 Then
            rami(tl0) = rami(th0)
            BI = 5:  'bandera TF0
            GoSub banderaT
          End If
    Case Else 'MODO3
          If M1M0 = "11" Then  'MODO 3
          If CInt(numy(6)) = 1 Then  'TR1
            TEXT2.Text = "TH0 HABILITADO EN MODO3"
            MODO3 = "SI"
            rami(th0) = rami(th0) + CM'+ 1
            If CInt(rami(th0)) >= 256 Then   'SOBREPASA
                  rami(th0) = rami(th0) - 256
                  BI = 7: ' TF1 = 1  bandera
                  GoSub banderaT
            End If
          End If
          If habil = "100" Or habil = "101" Or habil = "111" Then
            MODO3 = "SI"
            If CInt(NUMX(2)) = 0 Then ' c/t=0 TEMPORIZADOR
                  TEXT2.Text = "TL0 HABILITADO COMO TEMPORIZADOR EN MODO3"
                  rami(tl0) = rami(tl0) + CM'+ 1
            Else                  'c/t=1     'CONTADOR
                  TEXT2.Text = "TL0 HABILITADO COMO CONTADOR EN MODO3"
                  rami(tl0) = rami(tl0) + pulso(0)
                  pulso(0) = 0
            End If
            If CInt(rami(tl0)) >= 256 Then
                  rami(tl0) = rami(tl0) - 256
                  BI = 5: 'bandera TF0
                  GoSub banderaT
            End If
          End If
    End If
End Select
Call DECI(numy(), rami(TCON))
'************************TIMER1 **********************
Call BIN(rami(TMOD), NUMX())
Call BIN(rami(TCON), numy())
habil = CStr(numy(6)) + CStr(NUMX(7)) + CStr(pines(3).Value)
M1M0 = CStr(NUMX(5)) + CStr(NUMX(4))'para timer1
LABEL6.Caption = TMOD & timerx & M1M0
If habil = "100" Or habil = "101" Or habil = "111" Then
    Select Case M1M0
    Case "00" 'modo 0 ;13 bits
          cuenta = 32
          GoSub CONTADOR1
    Case "01" 'modo 1 ; 16 bits
          cuenta = 256
          GoSub CONTADOR1
    Case "10" 'modo 2  8 bits con autorecarga
            If CInt(NUMX(6)) = 0 Then 'c/t=0 temporizador
                  TEXT2.Text = "TIMER1 HABILITADO COMO TEMPORIZADOR"
                  rami(tl1) = rami(tl1) + CM'+ 1
            Else            'contador
                  TEXT2.Text = "TIMER1 HABILITADO COMO CONTADOR"
                  rami(tl1) = rami(tl1) + pulso(1)
                  pulso(1) = 0
            End If
            If CInt(rami(tl1)) >= 256 Then
                  rami(tl1) = rami(th1)
                  If MODO3 = "NO" Then 'solo habilita tf1 si el
                    BI = 7: 'TF1 = 1  timer0 no esta en modo3
                    GoSub banderaT
                  End If
            End If
          Case Else '"11" 'modo 3

  End Select
```

```
End If
Call DECI(numy(), rami(TCON))
'**********************timer2*************************
Call BIN(rami(T2CON), NUMX())
If CInt(NUMX(2)) = 1 Then 'SOLO SI TR2=1 SE ACTIVA
   MODO2 = CStr(NUMX(5) Or NUMX(4)) & CStr(NUMX(0)) & CStr(NUMX(2))
   'MODO2= (RCLK OR TCLK)+(CP/RL2)+(TR2)
   toc2 = CStr(NUMX(3)) & CStr(NUMX(1))'temp o cont del timer2
   'T0C2=EXEN2+C/T2
   LABEL6.Caption = TMOD & timerx & M1M0'ooooppppppppppp
   Select Case MODO2
   Case "011" '16 bits con CAPTURA
         Select Case toc2
         Case "00", "01" 'temp Y CONT
            GoSub contador2
         Case "10", "11" 'temp Y CONT con autORECARGA
            GoSub contador2
            If t2exf = 1 Then
                  t2exf = 0
                  rami(rcap2l) = rami(TL2)
                  rami(rcap2h) = rami(TH2)
                  BI = 6
                  GoSub banderat2
                  kj = MsgBox("CAPTURA")
            End If
         End Select
   Case "001" '16 bits con autorecarga
         Select Case toc2
         Case "00", "01" 'temp Y CONT
            GoSub contador2
         Case "10", "11" 'temp Y CONT con autORECARGA
            GoSub contador2
            If t2exf = 1 Then 'detector de transicion con T2EX
                  t2exf = 0
                  rami(TL2) = rami(rcap2l)
                  rami(TH2) = rami(rcap2h)
                  BI = 6
                  GoSub banderat2
                  kj = MsgBox("autorecarga")
            End If
         End Select
   Case "101", "111"GENERADOR DE BAUDIOS
         If CInt(NUMX(1)) = 0 Then 'C/T2
            rami(TL2) = rami(TL2) + 6 * CM'+ 6  '(f/2) CADA CM AUMENTA 6
         Else
            rami(TL2) = rami(TL2) + pulso2: pulso2 = 0
         End If
         cuenta = 256
         If CInt(rami(TL2)) >= cuenta Then
            rami(TL2) = rami(TL2) - cuenta
            rami(TH2) = rami(TH2) + 1
            If CInt(rami(TH2)) > 255 Then  'SOBREPASA
                  rami(TH2) = rami(TH2) - 256
                  rami(TL2) = rami(rcap2l)
                  rami(TH2) = rami(rcap2h)
            End If
         End If
         If t2exf = 1 And CInt(NUMX(3)) = 1 Then 'EXEN2 Y detector de transicion POR T2EX
                  t2exf = 0
                  BI = 6  'ACTIVO EXF2
                  GoSub banderat2
         End If
   Case Else
   End Select
End If
'**********************INTERRUPCIONES*****************
Call BIN(rami(TCON), NUMX())
Call BIN(rami(p3), numy())
'****DETECTOR DE NIVEL Y FLANCO PARA INT0 E INT1 ***********
If NUMX(0) = 0 And pines(2) = 0 Then TT0 Y INT0  POR NIVEL BAJO
   NUMX(1) = 1 'FLAG IE0
   MSGINT(0) = "INTERRUPCION EXTERNA INT0 HA SIDO ACTIVADA POR NIVEL BAJO"
   kj = MsgBox("bajo")
```

```
End If
If NUMX(0) = 1 And FINTE(0) = "SI" Then TT0 Y FLANCO INT0
   FINTE(0) = "NO"
   NUMX(1) = 1 'FLAG IE0
   MSGINT(0) = "INTERRUPCION EXTERNA INT0 HA SIDO ACTIVADA POR FLANCO"
   kj = MsgBox("FLANCO")
End If
If NUMX(2) = 0 And pines(3) = 0 Then TT1 e INT1  POR NIVEL BAJO
   NUMX(3) = 1 'FLAG IE0
   MSGINT(1) = "INTERRUPCION EXTERNA INT1 HA SIDO ACTIVADA POR NIVEL BAJO"
   kj = MsgBox("bajo")
End If
If NUMX(2) = 1 And FINTE(1) = "SI" Then TT0 Y FLANCO INT0
   FINTE(1) = "NO"
   NUMX(3) = 1 'FLAG IE0
   MSGINT(1) = "INTERRUPCION EXTERNA INT1 HA SIDO ACTIVADA POR FLANCO"
   kj = MsgBox("FLANCO")
End If
Call DECI(NUMX(), rami(TCON)) 'ACTUALIZO TCON
Call BIN(rami(IE), numz()) 'REVISO IE
Call BIN(rami(IP), ipx())  'REVISO IP
If CInt(numz(7)) = 1 Then 'SI EA=1 ESTAN HABILITADAS
   sai(0) = "INT0 " + CStr(NUMX(1))  'ieo FLAG
   sai(1) = "T0 " + CStr(NUMX(5))    'tfo
   sai(2) = "INT1 " + CStr(NUMX(3))  'ie1
   sai(3) = "T1 " + CStr(NUMX(7))    'tf1
   sai(4) = "PS " + CStr(Ri + TI)
   sai(5) = "T2 " + CStr(TF2 + EXF2)
   For J = 1 To 0 Step -1 '1=PRIORIDAD ALTA
   For I = 0 To 5
         Select Case sai(I) + CStr(numz(I)) + CStr(ipx(I)) + CStr(J)
         Case "INT0 1111", "INT0 1100"
            Call BIN(rami(TCON), NUMX())
            NUMX(1) = 0   'apago bandera de interrupcion ieo
            Call DECI(NUMX(), rami(TCON))'ACTUALIZO TCON
            vector = 3
            fuente$ = "INT0": FUENTED$ = "TIMER0"
            TEXT2.Text = "int0 activada"'MSGINT(0)'MENSAJE DE INTERR ACTIVADA
            Call pausa(6)
            Call previo2(8, 2, 2, 1, 1, NUMX(1))
            GoSub INTERRUP
         Case "T0 1111", "T0 1100" 'TIMER0
            NUMX(5) = 0   'apago bandera de interrupcion TF0
            Call DECI(NUMX(), rami(TCON))
            vector = 11
            fuente$ = "TIMER0": FUENTED$ = "INT1"
            TEXT2.Text = "INTERRUPCION DEL TIMER0 HA SIDO ACTIVADA " 'MENSAJE DE INTERR ACTIVADA
            Call pausa(6)
            Call previo2(8, 2, 2, 2, 2, NUMX(5))
            GoSub INTERRUP
         Case "INT1 1111", "INT1 1100" 'INT1
            NUMX(3) = 0   'apago bandera de interrupcion ie1
            Call DECI(NUMX(), rami(TCON))
            vector = 19
            fuente$ = "INT1": FUENTED$ = "TIMER1"
            TEXT2.Text = "int1 activada"'MSGINT(1)'MENSAJE DE INTERR ACTIVADA
            Call pausa(6)
            Call previo2(8, 2, 2, 3, 3, NUMX(3))
            GoSub INTERRUP
         Case "T1 1111", "T1 1100"
            NUMX(7) = 0   'apago bandera de interrupcion TF1
            Call DECI(NUMX(), rami(TCON))
            vector = 27
            fuente$ = "TIMER1": FUENTED$ = "PUERTO SERIE"
            TEXT2.Text = "INTERRUPCION DEL TIMER1 HA SIDO ACTIVADA " 'MENSAJE DE INTERR ACTIVADA
            Call pausa(6)
            Call previo2(8, 2, 2, 4, 4, NUMX(7))
            GoSub INTERRUP
         Case "PS 1111", "PS 1100"

         Case "T2 1111", "T2 1100"

         End Select
```

```
            Call BIN(rami(TCON), NUMX())
    Next I
    Next J
End If

GoTo FINT

contador2:
    If NUMX(1) = 0 Then
            rami(TL2) = rami(TL2) + CM'+ 1
    Else
            rami(TL2) = rami(TL2) + pulso2: pulso2 = 0
    End If
    cuenta = 256
    If CInt(rami(TL2)) >= cuenta Then
            rami(TL2) = rami(TL2) - cuenta
            rami(TH2) = rami(TH2) + 1
            If CInt(rami(TH2)) > 255 Then  'SOBREPASA
               rami(TH2) = rami(TH2) - 256
               BI = 7  'TF2 = 1 bandera
               GoSub banderat2
               If MODO2 = "001" Then   'RECARGA
                        rami(TL2) = rami(rcap2l)
                        rami(TH2) = rami(rcap2h)
               End If
            End If
    End If
Return
banderat2:
            TF2 = 1 'bandera
            Call BIN(rami(T2CON), fl2con())
            fl2con(BI) = 1
            Call DECI(fl2con(), rami(T2CON))
Return
CONTADOR1:
    If CInt(NUMX(6)) = 0 Then 'c/t=0 tempotizador
            TEXT2.Text = "TIMER1 HABILITADO COMO TEMPORIZADOR"
            rami(tl1) = rami(tl1) + CM'+ 1
    Else                        'CONTADOR
            TEXT2.Text = "TIMER1 HABILITADO COMO CONTADOR"
            rami(tl1) = rami(tl1) + pulso(1)
            pulso(1) = 0
    End If
    If CInt(rami(tl1)) >= cuenta Then
            rami(tl1) = rami(tl1) - cuenta'32
            rami(th1) = rami(th1) + 1
            If CInt(rami(th1)) > 255 Then
               rami(th1) = rami(th1) - 256
               If MODO3 = "NO" Then  'solo activa tfl  si el
                        BI = 7 'TF1 = 1 timer0 no esta en modo3
                        GoSub banderaT
               End If
            End If
    End If
Return

contador:
    If CInt(NUMX(2)) = 0 Then ' c/t=0 TEMPORIZADOR
            TEXT2.Text = "TIMER0 HABILITADO COMO TEMPORIZADOR"
            rami(tl0) = rami(tl0) + CM'+ 1
    Else                    'c/t=1     'CONTADOR
            TEXT2.Text = "TIMER0 HABILITADO COMO CONTADOR"
            rami(tl0) = rami(tl0) + pulso(0)
            pulso(0) = 0
    End If
    If CInt(rami(tl0)) >= cuenta Then
            rami(tl0) = rami(tl0) - cuenta'32
            rami(th0) = rami(th0) + 1
            If CInt(rami(th0)) > 255 Then
               rami(th0) = rami(th0) - 256
               BI = 5  'TF0 = 1 bandera
               GoSub banderaT
            End If
```

```
    End If
Return
banderaT:
        Call BIN(rami(TCON), FTCON())
        FTCON(BI) = 1
        Call DECI(FTCON(), rami(TCON))
Return
INTERRUP:
    FILAx = GRID2(2).TopRow
    DESTINO = vector: OPERACION$ = "INTERRUPCION": MD = ""
    Call MENSAJES(116)
    Call SALTAR
    Call MENSAJES(117)
    BD = pc 'CARGO EN EL BUS DE DIRECCIONES
    picture11(0).Enabled = True
    picture11(0).Left = 6255: picture11(0).Width = 2190:
    picture11(0).Enabled = False
    GRID2(2).Width = 2190
    GRID2(2).ColWidth(0) = 1500: GRID2(2).ColWidth(1) = 1000
    GRID2(2).AddItem "RUTINA" & Chr(9) & DATOS4(BD), 1
    GRID2(2).AddItem "DE" & Chr(9) & DATOS4(BD + 1), 2
    GRID2(2).AddItem "INTERRUPCION" & Chr(9) & DATOS4(BD + 2), 3
    GRID2(2).AddItem "PARA" & Chr(9) & DATOS4(BD + 3), 4
    GRID2(2).AddItem fuente$ & Chr(9) & DATOS4(BD + 4), 5
    GRID2(2).AddItem "-" & Chr(9) & DATOS4(BD + 5), 6
    GRID2(2).AddItem "-" & Chr(9) & DATOS4(BD + 6), 7
    GRID2(2).AddItem " " & Chr(9) & DATOS4(BD + 7), 8
    GRID2(2).AddItem "RUTINA" & Chr(9) & DATOS4(BD + 8), 9
    GRID2(2).AddItem "PARA" & Chr(9) & DATOS4(BD + 9), 10
    GRID2(2).AddItem FUENTED$ & Chr(9) & DATOS4(BD + 10), 11
    GRID2(2).AddItem "-" & Chr(9) & DATOS4(BD + 11), 12
    '****VISUALIZACION DE CONTENIDOS EN ROM EXTERNA***************
    For JF = 1 To 8
        BD = pc
        GRID2(2).TopRow = JF
        pc = pc + 1
        LABEL1(10).Caption = PCS(DATOS4(pc))
        SendKeys "{break}", True: wait = False
        CUADRO(15).Visible = False
        PICTURE4(5).Visible = False
        Call pausa(espera)
    Next JF
    Call MENSAJES(118)
    TEXT2.Text = "EJECUCION DE RETI"
    Call pausa(4)
    MD = "nada"
    Call RETORNO(0)
    picture11(0).Enabled = True
    picture11(0).Left = 6945: picture11(0).Width = 1500
    picture11(0).Enabled = False
    GRID2(2).Width = 1500
    GRID2(2).ColWidth(0) = 550: GRID2(2).ColWidth(1) = 950
    For II = 1 To 12  'TOTAL=# DE DATOS A ESCRIBIR EN ROM
        GRID2(2).RemoveItem 1
    Next II
    GRID2(2).TopRow = FILAx
Return
FINT:
CM = 0
'*************actualizacion de bloque de ram*********
Call previo3(3, 2, tl0 + 1, datos2(rami(tl0)))
Call previo3(3, 2, th0 + 1, datos2(rami(th0)))
Call previo3(3, 2, tl1 + 1, datos2(rami(tl1)))
Call previo3(3, 2, th1 + 1, datos2(rami(th1)))
Call previo3(3, 2, TL2 + 1, datos2(rami(TL2)))
Call previo3(3, 2, TH2 + 1, datos2(rami(TH2)))
Call previo3(3, 2, TCON + 1, datos2(rami(TCON)))
Call previo3(3, 2, T2CON + 1, datos2(rami(T2CON)))
Call previo3(3, 2, rcap2l + 1, datos2(rami(rcap2l)))
Call previo3(3, 2, rcap2h + 1, datos2(rami(rcap2h)))
'***************ACTUALIZO PANTALLA DE TIMERS************
Call BIN(rami(TCON), NUMX())
Call previo2(1, 1, 1, 1, 4, datos2(rami(th0)) & Chr(13) & datos2(rami(tl0)) & Chr(13) & NUMX(5))
```

```
Call previo2(1, 2, 2, 1, 4, datos2(rami(th1)) & Chr(13) & datos2(rami(tl1)) & Chr(13) & NUMX(7))
Call BIN(rami(T2CON), NUMX())
Call previo2(1, 3, 3, 1, 4, datos2(rami(TH2)) & Chr(13) & datos2(rami(TL2)) & Chr(13) & CStr(NUMX(7)) & Chr(13) &
CStr(NUMX(6)))
'**********ACTUALIZO PATALLA DE INTERRUPCIONES*********
Call BIN(rami(TCON), NUMX())
Call BIN(rami(T2CON), numy())
Call BIN(rami(scon), numz())
tit3$ = NUMX(1) & Chr(13) & NUMX(5) & Chr(13) & NUMX(3) & Chr(13) & NUMX(7) & Chr(13) & numz(0) & Chr(13) & numz(1)
& Chr(13) & numy(7) & Chr(13) & numy(6)
Call previo2(8, 2, 2, 1, 8, tit3$)
End Sub

Sub MULDIV (aumento)
Dim FUENTE4, DESTINO2
CM = 4: CARRY = 0
ORIGEN = acc: FUENTE2 = rami(acc)
DESTINO = B: FUENTE1 = rami(B)
'*****************************CODIGOS*****
CODIGO(1) = &HA4 + aumento
Call CODIGOS(1)
'*****************************LOCALIDADES**************
Call PREVIO(datos2(acc), datos2(rami(acc)), " ", " ")
Select Case OPERACION$ + MD
Case "MULAB"
   Call MENSAJES(22)
   msgalu = datos2(rami(acc)) & " * " & datos2(FUENTE1)
   '*****EJECUCION**************************
   FUENTE4 = rami(acc) * CLng(rami(B))
   DESTINO2 = Right(DATOS4(FUENTE4), 2) 'BYTE BAJO
   Call hexdec(DESTINO2, rami(acc))
   DESTINO2 = Left(DATOS4(FUENTE4), 2) 'BYTE BAJO
   Call hexdec(DESTINO2, rami(B))'BYTE ALTO

   If CLng(FUENTE4) < 256 Then
         OV = 1
   Else
         OV = 0
   End If
   GoSub REGRESO6
Case "DIVAB"
   Call MENSAJES(23)
   msgalu = datos2(rami(acc)) & " / " & datos2(FUENTE1)
   If rami(B) = 0 Then
         OV = 1
         msgalu = " Division por 0"
   Else
         OV = 0
         rami(acc) = Fix(rami(acc) / rami(B))
         rami(B) = FUENTE2 - rami(acc) * rami(B)
   End If
   GoSub REGRESO6
Case Else
   MsgBox "INSTRUCCION NO RECONOCIDA"
End Select
GoTo FIN9
REGRESO6:
   '*****PANTALLA******************
   picture9.Visible = True '****ALU*****
   Call EJECVISUAL(8)
   Call previo3(5, 1, 1, datos2(rami(acc)))
   Call previo3(5, 1, 8, datos2(rami(B)))
   Call pausa(espera)
   '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
   Call previo3(3, 2, acc + 1, datos2(rami(acc))) 'ACTUALIZA EL ACUMULADOR
   GRID2(3).TopRow = acc + 1
   Call previo3(3, 2, B + 1, datos2(rami(B))) 'ACTUALIZA EL ACUMULADOR
   GRID2(3).TopRow = B + 1
Return
FIN9:
End Sub

Sub OPCIONES_Click (index As Integer)
```

```
Select Case index
Case 0  'LISTA  todas las instrucciones
   barra(0).Enabled = False
   barra(1).Enabled = False
   barra(2).Enabled = False
   label2(0).Caption = "SET"
   label2(1).Caption = "DE INSTRUCCIONES"
   '******HACE UN DISPLAY DE LOS MD DISPONIBLES
   Open disco + "ARIT.DAT" For Input Access Read As #2
   For I = 1 To 111
            Line Input #2, IN
            COMBO2.AddItem IN
   Next I
   Close #2
   FRAME1.Visible = True
   PICTURE12.ZOrder 1
   SendKeys "{break}", True: wait = False
   '******CONTINUA EN COMMAND1 (ACEPTAR)
Case 1 'explorar
   opciones(1).Checked = Not opciones(1).Checked 'visto
   If opciones(1).Checked = True Then
            VSCROLL1(3).Visible = True
            VSCROLL1(4).Visible = True
            For I = 0 To 2 Step 2
               VSCROLL1(I).Visible = True
               GRID2(I).SelStartCol = 1
               GRID2(I).SelStartRow = 1
               GRID2(I).SelEndRow = 1
               GRID2(I).SelEndCol = 1
               dirmemh = GRID2(I).Clip  'obtengo la direccion actual
               Call hexdec(dirmemh, dirmem)
               ubic = Fix(dirmem / (8 - (I * 1.5)))
               VSCROLL1(I).Value = ubic
            Next I
            VSCROLL1(3).Value = GRID2(3).TopRow - 1
            VSCROLL1(4).Value = GRID2(4).TopRow - 1
   Else
            VSCROLL1(0).Visible = False
            VSCROLL1(2).Visible = False
            VSCROLL1(3).Visible = False
            VSCROLL1(4).Visible = False
   End If
Case 2 'velocidad
   PICTURE10.Left = 3345: PICTURE10.Top = 600
   PICTURE10.Visible = True
Case 3  'ULTIMA
   A$ = INSTRUCCION$
   Call SEPARAR
   Call EJECUCION
   barra(0).Enabled = True
   barra(1).Enabled = True
   barra(3).Enabled = True
   barra(4).Enabled = True
   opciones(0).Enabled = True
   opciones(1).Enabled = True
   opciones(3).Enabled = True

End Select
End Sub

Sub pausa (LIMITE)
TIMER1.Interval = 1000 * LIMITE
While TIMER1.Interval = 1000 * LIMITE
   SendKeys "{BREAK}", True: wait = False
Wend
End Sub

Function PCS (NOMBRE)
PAD = Space$(2)
NOMBRE1 = Mid$(NOMBRE, 1, 1)
NOMBRE2 = PAD + Mid$(NOMBRE, 2, 1)
NOMBRE3 = PAD + Mid$(NOMBRE, 3, 1)
NOMBRE4 = PAD + Mid$(NOMBRE, 4, 1)
```

```
PCS = NOMBRE1 + NOMBRE2 + NOMBRE3 + NOMBRE4 + PAD + "H"
End Function

Sub pines_Click (index As Integer)
Dim valorp%
ReDim puerto(16)
'****LECTURA DE PINES************************
valorp = pines(index).ValuePUEDE SER 0=NO ESCOJIDO 1 =ESCOJIDO 2 = DESHABILITADO
Select Case index
Case 2, 3  'int0,int1
   If pines(index).Value = 0 And pines(index).Enabled = True Then
            FINTE(index - 2) = "SI"
            Call MUESTREO
            FINTE(index - 2) = "NO"
   Else
            FINTE(index - 2) = "NO"
   End If
Case 4, 5   't0,t1
   If pines(index).Value = 0 And pines(index).Enabled = True Then 't0
            pulso(index - 4) = 1
            Call MUESTREO
            pulso(index - 4) = 0
   End If
Case 9    't2
   If pines(9).Value = 0 And pines(9).Enabled = True Then  't2
            pulso2 = 1
            Call MUESTREO
            pulso2 = 0
   End If
Case 10  't2ex
   If pines(10).Value = 0 And pines(10).Value = True Then  't2ex
            t2exf = 1 'detector de transicion
            Call MUESTREO
            t2exf = 0
   End If
Case 8 'RESET
   If pines(8).Value = 1 Then
            screen.MousePointer = 11
            For I = 0 To 3
               barra(I).Enabled = False
            Next I
            Call reseteo
            For I = 0 To 3
               barra(I).Enabled = True
            Next I
            screen.MousePointer = 0
            pines(8).Value = 0
   End If
End Select
End Sub

Sub PREVIO (extra1, extra2, extra3, extra4)
col1$ = extra1 & Chr$(13) & "" & Chr(13) & "" & Chr(13) & extra3
col1$ = col1$ & Chr$(13) & "" & Chr(13) & "" & Chr(13) & ""
col2$ = extra2 & Chr$(13) & "" & Chr(13) & "" & Chr(13) & extra4
col2$ = col2$ & Chr$(13) & "" & Chr(13) & "" & Chr(13) & ""
Call previo2(5, 0, 0, 1, 7, col1$)
Call previo2(5, 1, 1, 1, 7, col2$)
If poner = "SI" Then
   Call previo2(5, 0, 1, 8, 8, datos2(DESTINO) & Chr$(9) & datos2(FUENTE1))
Else
   Call previo2(5, 0, 1, 8, 8, "" & Chr$(9) & "")
End If
GRID2(5).TopRow = 1
GRID2(4).Visible = False
GRID2(3).Visible = False
LABEL1(27).Visible = False
SendKeys "{break}", True: wait = False
GRID2(5).Visible = True: GRID2(5).ZOrder 0
SendKeys "{break}", True: wait = False
End Sub

Sub previo2 (GRILLA, SSC, SEC, SSR, SER, MENS)
```

```
GRID2(GRILLA).SelStartCol = SSC          'permite que la celda
GRID2(GRILLA).SelStartRow = SSR          'seleccionada
GRID2(GRILLA).SelEndRow = SER            'se muestre
GRID2(GRILLA).SelEndCol = SEC            'resaltada
GRID2(GRILLA).Clip = CStr(MENS)          'EN BINARIO
SendKeys "{break}", True: wait = False


End Sub

Sub previo3 (indice, SSC, SSR, MENS)
GRID2(indice).SelStartCol = SSC          'permite que la celda
GRID2(indice).SelStartRow = SSR          'seleccionada
GRID2(indice).SelEndRow = SSR            'se muestre
GRID2(indice).SelEndCol = SSC            'resaltada
GRID2(indice).Clip = CStr(MENS)          'EN BINARIO
SendKeys "{break}", True: wait = False
End Sub

Sub PUSHPOP (aumento)
CM = 2
CODIGO(1) = &HC0 + aumento: CODIGO(2) = DESTINO
Select Case OPERACION$ + MD
   Case "PUSHDIREC"
          Call CODIGOS(2): INDI = 5
          mensaje$ = Switch(aumento = 0, "+1", aumento = 16, "-1")
          Orig = DESTINO: FUEN2 = rami(DESTINO)
          MD = "SP"
          Call INCDEC(1)
          EXTRAS = SP
          DESTINO = rami(SP): EXTRAS2 = DESTINO
          FUENTE1 = Switch(DESTINO > 127, rami(128 + DESTINO), DESTINO < 128, rami(DESTINO))
          MD = "@SP,DIREC"
          ORIGEN = Orig: FUENTE2 = rami(ORIGEN)
          Call MOVER
   Case "POPDIREC"
          Call CODIGOS(2): INDI = 5
          mensaje$ = Switch(aumento = 0, "+1", aumento = 16, "-1")
          ORIGEN = rami(SP)
          FUENTE2 = Switch(ORIGEN > 127, rami(128 + ORIGEN), ORIGEN < 128, rami(ORIGEN))
          MD = "DIREC,@SP"
          Call MOVER
          Call REGRESO(True)
          MD = "SP"
          Call INCDEC(-1)
Case Else
   MsgBox "INSTRUCCION NO RECONOCIDA"
End Select
GoTo FIN55
FIN55:
End Sub

Sub REGRESO (ver)
CUADRO(0).Visible = False
CUADRO(16).Visible = False
CUADRO(3).Visible = False
CUADRO(15).Visible = False
CUADRO(17).Visible = False
CUADRO(21).Visible = False
picture9.Visible = False 'ALU
GRID2(5).Visible = False
GRID2(3).Visible = ver '
GRID2(4).Visible = ver '
LABEL1(27).Visible = ver
CUADRO(1).Visible = False'ALU PARA ROTAR
PICTURE2(28).Visible = False'flecha en rotar
CUADRO(2).Visible = False
CUADRO(12).Visible = False
GRID2(7).Visible = False
PICTURE7(0).Visible = False
PICTURE7(1).Visible = False
LABEL1(8).Visible = False '****DESAPARECE MENSAJE DE ALU***
LABEL1(8).Caption = ""
```

```
CUADRO(14).Visible = False
LABEL4.Caption = "ALU"

For I = 0 To 20
   PICTURE4(I).Visible = False 'FLECHAS
Next I
For I = 1 To 8
   GRID2(5).RemoveItem I
   GRID2(5).AddItem " ", I
Next I
CUADRO(17).Width = 1005
picture9.Left = 3210
PICTURE6(2).Visible = False
PICTURE6(1).Visible = False

GRID2(5).Cols = 2
GRID2(5).FixedRows = 0
Call previo2(5, 0, 1, 0, 0, "Direc" & Chr(9) & "  Dato")
GRID2(5).FixedRows = 1
GRID2(5).ColWidth(0) = 750
GRID2(5).ColWidth(1) = 830
GRID2(5).ColAlignment(1) = 2
label9.Visible = False'carry en rotar
'text2.Text = " "
CUADRO(20).Left = 2200
CUADRO(11).Visible = True'

poner = "SI"'**************************
MODO$ = ""

End Sub

Sub reseteo ()
p1aux = 0: p3aux = 0
'**************VALORES DEL RESET***********
Erase rami
rami(SP) = &H7
rami(P0) = &HFF
rami(P1) = &HFF
rami(P2) = &HFF
rami(p3) = 255
pc = 0
LABEL1(10).Caption = PCS(DATOS4(pc))

MODO$ = "" 'AGRUPA LOS OPERANDOS DE LA INSTRUCCION
paso = HSCROLL1.Value: tiempo = .1
espera = (-paso / 160) + 3 'con min=1 y max=400
For I = 0 To 9
   Call BUSCAR(I)
   'ROMIX(FILA, COLUMNA) = 255
   GRID2(2).AddItem datos2(ROMIX(FILA, COLUMNA)) & Chr$(9) & DATOS4(I), I + 1
Next I
GRID2(2).TopRow = 1
For I = 0 To 7
   GRID2(0).AddItem datos2(RAMx(0, I)) & Chr$(9) & DATOS4(I), I + 1
Next I
For I = 0 To 255
   Call previo2(3, 2, 2, I + 1, 1 + 1, datos2(rami(I)))
Next I
For I = 0 To 127
   Call previo2(4, 1, 1, I + 1, I + 1, datos2(rami(256 + I)))
Next I
'***ACTUALIZACION DE BANDERAS******
CARRY = 0
AC = 0
OV = 0
P = 0
label3(0).Caption = " " & CARRY & "   " & AC & "    " & OV & "   " & P
FUENTE1 = 0: FUENTE2 = 0
Call banderas
End Sub

Sub RETORNO (aumento)
```

```
CM = 2
If MD = "nada" Then
   If OPERACION$ <> "interrupcion" Then
          '****CODIGOS**********
          CODIGO(1) = &H22 + aumento
          Call CODIGOS(1)
   End If
   Call MENSAJES(97)
   '**** MOV PC15-8,@SP**********
          INDI = 2300
          Call hexdec(Right$(DATOS4(pc), 2), ORIGEN)'PC15-8
          DESTINO = rami(SP)
          'revisar*************************************
          FUENTE1 = Switch(DESTINO > 127, rami(128 + DESTINO), DESTINO < 128, rami(DESTINO))
          PC3 = FUENTE1
          PC4$ = datos2(FUENTE1) & datos2(ORIGEN)
          Call hexdec(PC4$, PC2)
          LABEL1(22).Caption = Left$(PCS(DATOS4(PC2)), 5)
          GoSub MOVERR
          LABEL1(10).Caption = PCS(DATOS4(PC2))
          LABEL1(22).Visible = False
          '****DEC SP******
          MD = "SP"
          Call INCDEC(-1)
          '**** MOV PC7-0,@SP**********
          INDI = 2800
          Call hexdec(Left$(DATOS4(pc), 2), ORIGEN)'PC7-0
          DESTINO = rami(SP)
          FUENTE1 = Switch(DESTINO > 127, rami(128 + DESTINO), DESTINO < 128, rami(DESTINO))
          PC4$ = datos2(PC3) & datos2(FUENTE1)
          Call hexdec(PC4$, PC2)
          LABEL1(22).Caption = PCS(DATOS4(PC2))
          GoSub MOVERR
          '****DEC SP**********
          MD = "SP"
          Call INCDEC(-1)
          GRID2(3).TopRow = rami(SP) + 1
          GRID2(3).Visible = True
          pc = (PC2)
          LABEL1(10) = PCS(DATOS4(pc))
Else
   kj = MsgBox("Instruccion no reconocida")
End If
GoTo FINR
MOVERR:
   Call PREVIO("SP", datos2(rami(SP)), " ", " ")
   Call EJECVISUAL(10)  'PUNTERO
   L1 = 1710: T1 = 4980: L2 = INDI: L3 = L2: T2 = 900
   '******ejecuto move********
   CUADRO(16).Move L1, T1
   CUADRO(16).Visible = True
   LABEL1(15).Caption = datos2(FUENTE1)
   PICTURE4(4).Visible = True
   SendKeys "{break}", True: wait = False

   Call movi(L1, L2, 0, 16)
   PICTURE4(4).Visible = False
   PICTURE4(1).Visible = True

   Call movi(T1, T2, 1, 16)
   LABEL1(22).Visible = True
   Call pausa(espera)
   Call REGRESO(False)

Return
FINR:
   LABEL1(22).Visible = False
End Sub

Sub ROTAR (aumento)
Select Case OPERACION$
Case "RL"
   GoSub COMUNROT
```

```
    Call MENSAJES(45)
    CUADRO(12).Move 510, 330
    numy(0) = NUMX(7):
    rami(acc) = numy(0)  'BIT A BIT
    'ACTUALIZO EL ACUMULADOR
    For I = 0 To 6
            numy(I + 1) = NUMX(I)'ROTAN LOS BITS
            rami(acc) = rami(acc) + numy(I + 1) * (2 ^ (I + 1))
            Call previo3(6, 7 - I, 0, CStr(NUMX(I))) 'VISUALIZA EL CONTENIDO DEL
    Next I
    '****VISUALIZACION DE RL A**************
            GoSub PROPIEDADES
            Call pausa(espera)
            PICTURE2(28).Visible = True
            Call pausa(espera)
            CUADRO(2).Visible = True: LABEL1(26).Caption = " ": CUADRO(12).Visible = True
            LABEL1(26).Caption = NUMX(7)
            SendKeys "{break}", True: wait = False
            '****ROTAR*********
            Call movi(510, 15, 0, 12)
            Call movi(330, 530, 1, 12)
            Call previo3(6, 1, 0, " ")
            Call previo3(6, 0, 0, CStr(NUMX(6)))
            Call movi(530, 735, 1, 12)
            For I = 1 To 5
               Call movi(15 + 400 * (I - 1), 15 + 400 * I, 0, 12)
               Call previo3(6, I + 1, 0, " ")
               Call previo3(6, I, 0, CStr(NUMX(6 - I)))
            Next I
            Call movi(735, 535, 1, 12)
            Call previo3(6, 7, 0, " ")
            Call previo3(6, 6, 0, CStr(NUMX(0)))
            Call movi(535, 330, 1, 12)
    CUADRO(12).Move 1800, 330
    GoSub COMUN
Case "RLC"
    GoSub COMUNROT
    Call MENSAJES(46)
    label9.Visible = True  'ETIQUETA CARRY
    CUADRO(12).Move 15, 330
    LABEL1(25).Caption = " "
    CUADRO(14).Visible = True
    Call BIN(0, numy())
    numy(0) = CARRY: CARRY = NUMX(7)
    Call previo3(6, 0, 0, CStr(NUMX(7))) 'VISUALIZA EL CONTENIDO DEL
    For I = 0 To 6
            numy(I + 1) = NUMX(I)'EL NUEVO VALOR DEL ACUMULADOR
            Call previo3(6, 7 - I, 0, CStr(NUMX(I))) 'VISUALIZA EL CONTENIDO DEL
    Next I
    Call DECI(numy(), rami(acc))
    '****VISUALIZACION DE RLC A**************
    GoSub PROPIEDADES
    Call pausa(espera)
    PICTURE2(28).Visible = TrueFLECHA
    Call pausa(espera)
    LABEL1(26).Caption = numy(0)'**CARRY VIEJO****
    CUADRO(12).Visible = True
    LABEL1(25).Caption = " "
    SendKeys "{break}", True: wait = False
    CUADRO(2).Visible = True: 'CONTAINER
    Call pausa(espera)
    '****COMUN PARA RL Y RLC

    Call movi(330, 735, 1, 12)
    Call previo3(6, 0, 0, " ")
    CUADRO(14).Visible = True: LABEL1(25).Caption = NUMX(7)
    SendKeys "{break}", True: wait = False
    For I = 0 To 5
            Call movi(15 + 330 * I, 15 + 330 * (I + 1), 0, 12)
            Call previo3(6, I + 1, 0, " ")
            Call previo3(6, I, 0, CStr(NUMX(6 - I)))
    Next I
    Call movi(735, 535, 1, 12)
```

```
    Call previo3(6, 7, 0, " ")
    Call previo3(6, 6, 0, CStr(NUMX(0)))
    Call movi(535, 330, 1, 12)
    CUADRO(12).Move 1800, 330'1755
    GoSub COMUN

Case "RR"
    GoSub COMUNROT
    Call MENSAJES(47)
    CUADRO(12).Move 1755, 330
    numy(7) = NUMX(0)
    rami(acc) = numy(7) * (2 ^ (7))  'BIT A BIT
    For I = 0 To 6
            numy(I) = NUMX(I + 1)'EL NUEVO VALOR DEL ACUMULADOR
            rami(acc) = rami(acc) + numy(I) * (2 ^ (I))
            Call previo3(6, 6 - I, 0, CStr(NUMX(I + 1)))'VISUALIZA EL CONTENIDO DEL
    Next I
    '****VISUALIZACION DE RR A*****************
    GoSub PROPIEDADES
    Call pausa(espera)
    PICTURE2(28).Visible = True
    Call pausa(espera)
    CUADRO(2).Visible = True: LABEL1(26).Caption = " ": CUADRO(12).Visible = True
    LABEL1(26).Caption = NUMX(0)'*****REEMPLAZAR CON CARRY****
    SendKeys "{break}", True: wait = False
    Call movi(1755, 2040, 0, 12)
    '****COMUN PARA RL Y RLC
    Call movi(330, 530, 1, 12)
    Call previo3(6, 6, 0, " ")
    Call previo3(6, 7, 0, CStr(NUMX(1)))
    Call movi(530, 735, 1, 12)
    For I = 5 To 0 Step -1
            Call movi(2040 - 335 * (5 - I), 2040 - 335 * (6 - I), 0, 12)
            Call previo3(6, I, 0, " ")
            Call previo3(6, I + I, 0, CStr(NUMX(7 - I)))
    Next I
    CUADRO(12).Move 15
    Call movi(735, 535, 1, 12)
    Call previo3(6, 0, 0, " ")
    Call previo3(6, 1, 0, CStr(NUMX(7)))
    Call movi(535, 330, 1, 12)
    CUADRO(12).Move 510, 330
    GoSub COMUN
Case "RRC"
    GoSub COMUNROT
    Call BIN(0, numy())
    Call MENSAJES(48)
    label9.Visible = True
    CUADRO(12).Move 1755, 330
    LABEL1(25).Caption = CARRY: CUADRO(14).Visible = True
    SendKeys "{break}", True: wait = False
    numy(7) = CARRY: CARRY = NUMX(0)'EN NUMY SE ALMACENA
    Call DECI(numy(), rami(acc))
    Call previo3(6, 7, 0, CStr(NUMX(0)))  'VISUALIZA EL CONTENIDO DEL
    For I = 0 To 6
            numy(I) = NUMX(I + 1)'EL NUEVO VALOR DEL ACUMULADOR
            rami(acc) = rami(acc) + numy(I) * (2 ^ (I))
            Call previo3(6, 6 - I, 0, CStr(NUMX(I + 1)))'VISUALIZA EL CONTENIDO DEL
    Next I
    '****VISUALIZACION DE RR A*****************
    GoSub PROPIEDADES
    SendKeys "{break}", True: wait = False
    Call pausa(espera)
    PICTURE2(28).Visible = True
    Call pausa(espera)
    CUADRO(2).Visible = True:
    Call pausa(espera)
    Call previo3(6, 7, 0, " ")

    LABEL1(26).Caption = NUMX(0)'*****REEMPLAZAR CON CARRY****
    CUADRO(12).Visible = True
    SendKeys "{break}", True: wait = False
    Call movi(1755, 2040, 0, 12)
```

```
    '****COMUN PARA RL Y RLC
    Call movi(330, 530, 1, 12)
    Call previo3(6, 6, 0, " ")
    Call previo3(6, 7, 0, CStr(NUMX(1)))
    Call movi(530, 735, 1, 12)
    For I = 6 To 1 Step -1
            Call movi(2040 - 335 * (6 - I), 2040 - 335 * (7 - I), 0, 12)
            Call previo3(6, I - 1, 0, " ")
            Call previo3(6, I, 0, CStr(NUMX(8 - I)))
    Next I
    CUADRO(12).Move 15
    Call movi(735, 535, 1, 12)
              LABEL1(25).Caption = " ": CUADRO(14).Visible = True
              Call previo3(6, 0, 0, CStr(numy(7)))
    Call movi(535, 330, 1, 12)
    GoSub COMUN
Case "SWAP"
            GoSub COMUNROT
            Call MENSAJES(49)
            CUADRO(11).Visible = False
            GRID2(7).Visible = True
            GoSub PROPIEDADES
            Call pausa(espera)
            PICTURE2(28).Visible = True '*****FLECHA*****
            Call pausa(espera)
            CUADRO(2).Visible = True
            rami(acc) = 0

            For I = 7 To 0 Step -1
               Call previo3(6, I, 0, CStr(NUMX(7 - I)))'VISUALIZA EL CONTENIDO DEL ACC
            Next I
            For I = 3 To 0 Step -1
               numy(I) = NUMX(4 + I)'****EJECUTO SWAP***
               numy(4 + I) = NUMX(I)'****EJECUTO SWAP***
               rami(acc) = rami(acc) + numy(I) * (2 ^ I) + numy(4 + I) * (2 ^ (4 + I))
            Next I

            DATO1$ = "": DATO2$ = ""
            For I = 7 To 4 Step -1
               DATO2$ = DATO2$ + CStr(NUMX(I)) + Chr(9)
               DATO1$ = DATO1$ + CStr(NUMX(I - 4)) + Chr(9)
            Next I
            DATO3$ = " " + Chr(9) + " " + Chr(9) + " " + Chr(9) + " " + Chr(9) + DATO1$
            DATO4$ = DATO2$ + Chr(9) + " " + Chr(9) + " " + Chr(9) + " " + Chr(9) + " "
            Call previo2(6, 0, 7, 0, 0, DATO3$)
            Call previo2(7, 0, 7, 0, 0, DATO4$)
            Call pausa(espera)
            DATO4$ = "" + Chr(9) + "" + Chr(9) + "" + Chr(9) + "" + Chr(9) + DATO2$

            Call previo2(6, 0, 7, 0, 0, DATO1$)
            Call previo2(7, 0, 7, 0, 0, DATO4$)
            Call pausa(espera)
            Call previo2(6, 0, 7, 0, 0, DATO1$ + DATO2$)
            Call previo2(7, 0, 7, 0, 0, " ")
            Call pausa(espera)
            GoSub COMUN
Case Else
   MsgBox "INSTRUCCION NO RECONOCIDA"
End Select
GoTo FINAL6
COMUNROT:
   CM = 1
   '*****CODIGOS*****
   CODIGO(1) = &H3 + aumento
   Call CODIGOS(1)
   poner = "NO"
   Call PREVIO(datos2(DESTINO), datos2(rami(DESTINO)), " ", " ")
   GRID2(3).TopRow = acc + 1
   Call BIN(rami(acc), NUMX())'PASO A BINARIO EL ACUMULADOR
   For I = 7 To 0 Step -1
            Call previo3(6, I, 0, " ")' GRID2(6) MUESTRA EL ACC
   Next I
Return
```

```
PROPIEDADES:
   CUADRO(1).ZOrder 0
   CUADRO(1).BorderStyle = 0
   CUADRO(1).Visible = True
   CUADRO(1).Left = 2145
   CUADRO(1).Top = 3345
   CUADRO(1).Width = 2880
   CUADRO(1).Height = 2100
   INDI = 3
   SendKeys "{break}", True: wait = False
   GRID2(5).ZOrder 0
Return
COMUN:
   '****TRASLADO DEL RESULTADO AL ACUMULADOR**********
   LABEL1(8).Caption = " "
   Call EJECVISUAL(3)
   Call previo3(5, 1, 1, datos2(rami(acc)))
   Call pausa(espera)
   Call previo3(3, 2, acc + 1, datos2(rami(acc)))
Return
FINAL6:
CUADRO(11).Visible = True
End Sub

Sub SALTAR ()
Dim cod1%
LABEL1(21).Caption = ""
CODIGO(1) = 0: CODIGO(2) = 0: CODIGO(3) = 0
PC2 = DESTINO
Select Case OPERACION$ + MD
   Case "ACALLDIREC", "ACALLDIREC11"
            cod1 = 17
            GoSub codigos11
            Call MENSAJES(95)
            GoSub OPERACIONES
            Call MENSAJES(112)
            GoSub pedirec11
   Case "AJMPDIREC", "AJMPDIREC11"
            cod1 = 1
            GoSub codigos11
            Call MENSAJES(99)
            GoSub pedirec11
   Case "LCALLDIREC", "LCALLDIREC11", "LCALLDIREC16"
            cod1 = 18
            GoSub codigos16
            Call MENSAJES(96)
            GoSub OPERACIONES
            Call MENSAJES(113)
            GoSub pedirec16

   Case "LJMPDIREC", "LJMPDIREC11", "LJMPDIREC16"
               cod1 = 2
               GoSub codigos16
               Call MENSAJES(100)
               GoSub pedirec16

Case "SJMPDIREC" 'PC=PC+2+REL
   REL = DESTINO
   '****CODIGOS**********
   Call BIN(DESTINO, NUMX())
   CODIGO(1) = &H80
   CODIGO(2) = DESTINO
   Call CODIGOS(2)
   Call MENSAJES(101)
   '********PC+RELATIVO********
   GRID2(4).Visible = False
   Call EJECVISUAL(13)
   GRID2(4).Visible = True

Case "JMP@A+DPTR"
            '****CODIGOS**********
            CODIGO(1) = &H75
            Call CODIGOS(1)
```

```
            '******LOCALIDADES*****
            poner = "NO"
            Call PREVIO("ACC", datos2(rami(acc)), " ", " ")
            Call previo3(5, 0, 8, "DPL")
            Call previo3(5, 1, 8, datos2(rami(DPL)))
            Call previo3(5, 0, 7, "DPH")
            Call previo3(5, 1, 7, datos2(rami(dpH)))
            Call MENSAJES(102)
            Call EJECVISUAL(15)
 Case "JZDIREC" 'PC=PC+2+REL
    '****CODIGOS*********
    CODIGO(1) = &H60
    CODIGO(2) = DESTINO
    Call CODIGOS(2)
    REL = DESTINO
    '********PC+RELATIVO********
    poner = "NO"
    Call PREVIO("ACC", datos2(rami(acc)), " ", " ")
    Call MENSAJES(103)
    picture9.Visible = True
    CUADRO(16).Move 1710, 3195
    Call FLECHAS(16, 4, datos2(rami(acc)))'*****
    Call movi(1710, 3200, 0, 16) 'DEST. AL ALU****

    If CInt(rami(acc)) = 0 Then
            LABEL4.Caption = "A" & "=0"
            msgalu = "SALTO"
            LABEL1(8).Caption = msgalu
            LABEL1(8).Visible = True
            Call pausa(2)
            CUADRO(16).Visible = False
            Call EJECVISUAL(13)
    Else
            picture9.Visible = True
            LABEL4.Caption = "A" & Chr$(216) & "0"
            msgalu = "NO HAY SALTO"
            LABEL1(8).Caption = msgalu
            LABEL1(8).Visible = True
            Call pausa(2)
    End If
Case "JNZDIREC" 'PC=PC+2+REL
    '****CODIGOS*********
    CODIGO(1) = &H70
    CODIGO(2) = DESTINO
    Call CODIGOS(2)
    REL = DESTINO
    '********PC+RELATIVO********
    poner = "NO"
    Call PREVIO("ACC", datos2(rami(acc)), " ", " ")
    Call MENSAJES(104)
    picture9.Visible = True
    CUADRO(16).Move 1710, 3195
    Call FLECHAS(16, 4, datos2(rami(acc)))'*****
    Call movi(1710, 3200, 0, 16) 'DEST. AL ALU****

    If CInt(rami(acc)) <> 0 Then
            LABEL4.Caption = "A" & "<>0"
            msgalu = "SALTO"
            LABEL1(8).Caption = msgalu
            LABEL1(8).Visible = True
            Call pausa(2)
            CUADRO(16).Visible = False
            Call EJECVISUAL(13)
    Else
            picture9.Visible = True
            LABEL4.Caption = "A=0"
            msgalu = "NO HAY SALTO"
            LABEL1(8).Caption = msgalu
            LABEL1(8).Visible = True
            Call pausa(espera)
    End If

Case "INTERRUPCION"
```

```
        CUADRO(19).Visible = True  'reg temporal para direc11
        EXTRAS2 = DESTINO 'VECTOR DE INTERRUPCION
        LABEL1(21).Caption = PCS(DATOS4(EXTRAS2))'REG TEMPORAL
        Call BIN(DESTINO, NUMX())'0j0i000000000000
        Call MENSAJES(96)
        GoSub OPERACIONES
        '********MOV PC,DIRECCION*********
        Call MENSAJES(119)
        CUADRO(19).Move 3750
        L1 = CUADRO(20).Left + LABEL1(23).Width
        Call movi(3750, L1, 0, 19)
        Call pausa(espera)
        pc = PC2 'PC CAMBIA AL VALOR DE VECTOR DE INTERRUPCION
        Call pausa(espera)
        CUADRO(19).Visible = False
        LABEL1(10).Caption = PCS(DATOS4(pc))
        CUADRO(19).Move 3750
        CUADRO(19).Visible = False  'reg temporal para direc11
Case Else
   MsgBox "INSTRUCCION NO RECONOCIDA"
End Select
GoTo FINS
OPERACIONES:
        '****INC SP******
        PC2 = DESTINO
        MD = "SP"
        Call INCDEC(1)
        '**** MOV @SP,PC7-0***********
        INDI = 2800
        Call hexdec(Right$(DATOS4(pc), 2), ORIGEN)'PC7-0
        FUENTE2 = ORIGEN
        DESTINO = rami(SP)
        FUENTE1 = Switch(DESTINO > 127, rami(128 + DESTINO), DESTINO < 128, rami(DESTINO))
        MD = "@SP,PC7-0"
        Call MOVER
        '****INC SP***********
        MD = "SP"
        Call INCDEC(1)
        '**** MOV @SP,PC15-8***********
        INDI = 2200
        Call hexdec(Left$(DATOS4(pc), 2), ORIGEN)'PC15-8
        FUENTE2 = ORIGEN
        DESTINO = rami(SP)
        FUENTE1 = Switch(DESTINO > 127, rami(128 + DESTINO), DESTINO < 128, rami(DESTINO))
        MD = "@SP,PC7-0"
        Call MOVER
        GRID2(3).TopRow = rami(SP) + 1
        GRID2(3).Visible = True
        GRID2(4).Visible = True
Return
codigos11:
        '****CODIGOS**********
        Call BIN(DESTINO, NUMX()) 'paso direc11 a binario
        Call BIN(cod1, numy())
        For I = 5 To 7
          numy(I) = NUMX(I + 3)
          NUMX(I + 3) = 0
        Next I
        Call DECI(numy(), CODIGO(1))
        Call DECI(NUMX(), CODIGO(2))
        Call BIN(DESTINO, NUMX()) 'paso direc11 a binario
        Call CODIGOS(2)
Return
pcdirec11:
        '********MOV PC,DIRECCION*********
        Call BIN(pc, numy())  'PC = NUMY() EN BINARIO
        CUADRO(20).Move 1575
        LABEL1(10).Caption = CStr(BINARIO2) & CStr(binario)
        CUADRO(19).Move 3750
        Call movi(3750, 2490, 0, 19)
        Call pausa(espera)
        '*******ACTUALIZO EL PC*****
        For I = 11 To 15     'ADDR11=NUMX() EN BINARIO
```

```
            NUMX(I) = numy(I) 'EL PC =b15+..b11+direc11
          Next I
          Call DECI(NUMX(), pc)
          Call pausa(espera)
          CUADRO(20).Move 2200'1770
          CUADRO(19).Visible = False
          LABEL1(10).Caption = PCS(DATOS4(pc))
          CUADRO(19).Move 3750
Return
codigos16:
          '****CODIGOS**********
          EXTRAS2 = DESTINO '=DIREC16
          Call BIN(DESTINO, NUMX())'paso direc16 a binario
          CODIGO(1) = cod1
          For I = 0 To 7
            CODIGO(2) = CODIGO(2) + NUMX(I + 8) * (2 ^ I)
            CODIGO(3) = CODIGO(3) + NUMX(7 - I) * (2 ^ (7 - I))
          Next I
          Call CODIGOS(3)
Return
pcdirec16:
          '********MOV PC,DIRECCION*********
          CUADRO(19).Move 3750
          L1 = CUADRO(20).Left + LABEL1(23).Width
          Call movi(3750, L1, 0, 19)
          Call pausa(espera)
          pc = PC2 'EL PC CAMBIA AL VALOR DE ADDR16 INGRESADO
          Call pausa(espera)
          CUADRO(19).Visible = False
          LABEL1(10).Caption = PCS(DATOS4(pc))
          CUADRO(19).Move 3750
Return
FINS:
CM = 2
End Sub

Sub SALTO_Click (index As Integer)
barra(0).Enabled = False
barra(1).Enabled = False
barra(2).Enabled = False
Select Case index
Case 0
  Call MODO1(95, 95, "ACALL")
Case 1
  Call MODO1(96, 96, "LCALL")
Case 2
  Call MODO1(97, 97, "RET")
Case 3
  Call MODO1(98, 98, "RETI")
Case 4
  Call MODO1(99, 99, "AJMP")
Case 5
  Call MODO1(100, 100, "LJMP")
Case 6
  Call MODO1(101, 101, "SJMP")

Case 7
  Call MODO1(102, 102, "JMP")
Case 8
  Call MODO1(103, 103, "JZ")
Case 9
  Call MODO1(104, 104, "JNZ")
Case 10
  Call MODO1(105, 108, "CINE")
Case 11
  Call MODO1(109, 110, "DJNZ")
Case 12
  Call MODO1(111, 111, "NOP")
End Select
End Sub

Sub SALTOBIT ()
CM = 2
```

```
MD = OPERACION$ + MD
PC2 = DESTINO
Select Case MD
Case "JCDIREC" 'PC=PC+REL
   '****CODIGOS**********
   CODIGO(1) = &H40: CODIGO(2) = DESTINO
   Call CODIGOS(2)
   REL = DESTINO
   DESTINO = psw: FUENTE1 = 7
   '********PC+RELATIVO********
   GoSub LOCALIDADES
   Call MENSAJES(90)
   GoSub UNO
Case "JNCDIREC" 'PC=PC+REL
   '****CODIGOS**********
   CODIGO(1) = &H50: CODIGO(2) = DESTINO
   Call CODIGOS(2)
   REL = DESTINO
   DESTINO = psw: FUENTE1 = 7
   '********PC+RELATIVO********
   GoSub LOCALIDADES
   Call MENSAJES(91)
   GoSub CERO
Case "JBBIT,DIREC"
   '****CODIGOS**********
   CODIGO(1) = &H20: CODIGO(2) = cmbyte: CODIGO(3) = ORIGEN
   Call CODIGOS(3)
   REL = ORIGEN
   '********PC+RELATIVO********
   GoSub LOCALIDADES
   Call MENSAJES(92)
   GoSub UNO
Case "JNBBIT,DIREC"
   '****CODIGOS**********
   CODIGO(1) = &H30: CODIGO(2) = cmbyte: CODIGO(3) = ORIGEN
   Call CODIGOS(3)
   REL = ORIGEN
   '********PC+RELATIVO********
   GoSub LOCALIDADES
   Call MENSAJES(93)
   GoSub CERO
Case "JBCBIT,DIREC"
   '****CODIGOS**********
   CODIGO(1) = &H10: CODIGO(2) = cmbyte: CODIGO(3) = ORIGEN
   Call CODIGOS(3)
   REL = ORIGEN
   '********PC+RELATIVO********
   Call MENSAJES(94)
   GoSub LOCALIDADES
   GoSub UNO
Case Else
   MsgBox "INSTRUCCION NO RECONOCIDA"

End Select
GoTo FINBIT
LOCALIDADES:
   GRID2(5).Cols = 3
   GRID2(5).FixedRows = 0
   Call previo2(5, 0, 2, 0, 0, "Direc" & Chr(9) & "Dato" & Chr(9) & "Bit")
   GRID2(5).FixedRows = 1
   For I = 0 To 2
         GRID2(5).ColWidth(I) = 500
   Next I
   Call BIN(rami(DESTINO), NUMX())
   poner = "NO"
   Call PREVIO(datos2(DESTINO), datos2(rami(DESTINO)), " ", " ")
   Call previo3(5, 2, 1, NUMX(FUENTE1))
   picture9.Visible = True
   CUADRO(16).Move 1710, 3195
   Call FLECHAS(16, 4, NUMX(FUENTE1))'*****
   Call movi(1710, 3200, 0, 16) 'DEST. AL ALU****
Return
UNO:
```

```
   If CInt(NUMX(FUENTE1)) = 1 Then
            LABEL4.Caption = NUMX(FUENTE1) & "=" & "1"
            msgalu = "SALTO"
            LABEL1(8).Caption = msgalu
            LABEL1(8).Visible = True
            Call pausa(espera)
            If MD = "JBCBIT,DIREC" Then
               NUMX(FUENTE1) = 0
               Call FLECHAS(16, 2, 0)'*****
               Call movi(3200, 1900, 0, 16) 'DEST. AL ALU****
               Call DECI(NUMX(), rami(DESTINO))
               Call previo3(5, 2, 1, NUMX(FUENTE1))
               Call previo3(3, 2, DESTINO + 1, datos2(rami(DESTINO)))
               GRID2(3).TopRow = DESTINO + 1
               LABEL1(8).Caption = " " & msgalu
            End If
            Call pausa(espera)
            CUADRO(16).Visible = False
            Call EJECVISUAL(13)  'PC+REL
   Else
            picture9.Visible = True
            LABEL4.Caption = NUMX(FUENTE1) & Chr$(216) & "1"
            msgalu = "NO HAY SALTO"
            LABEL1(8).Caption = msgalu
            LABEL1(8).Visible = True
            Call pausa(espera)
   End If
Return
CERO:
   If CInt(NUMX(FUENTE1)) = 0 Then
            LABEL4.Caption = NUMX(FUENTE1) & "= 0"
            msgalu = "SALTO"
            LABEL1(8).Caption = msgalu
            LABEL1(8).Visible = True
            Call pausa(espera)
            CUADRO(16).Visible = False
            Call EJECVISUAL(13)
   Else
            picture9.Visible = True
            LABEL4.Caption = NUMX(FUENTE1) & Chr$(216) & "0"
            msgalu = "NO HAY SALTO"
            LABEL1(8).Caption = msgalu
            LABEL1(8).Visible = True
            Call pausa(espera)
   End If
Return
FINBIT:
End Sub

Sub SEPARAR ()
'instruccion=operacion   op(1)  ,  op(2)  ,  op(3)
'               pos1     pos2      pos3
'**PROCESO PARA SEPARAR LA INSTRUCCION
OP(1) = "": OP(2) = "": OP(3) = "" 'inicializo operandos
resul$ = Trim$(A$)'quito espacios en blanco de la instruccion (extremos)
pos1 = InStr(resul$, " ")'fin de operacion
pos2 = InStr(resul$, ",") 'fin del primer operando
pos3 = InStr(pos2 + 1, resul$, ",")'fin del segundo operando
If pos1 <> 0 Then 'puede exsistir el primer operando
   OPERACION$ = Left$(resul$, pos1 - 1)'obtengo operacion
   If pos2 <> 0 Then 'el primer operando existe
            OP(1) = Mid$(resul$, pos1, pos2 - pos1)'obtengo el primer operando
            If pos3 <> 0 Then 'el segundo y tercer operando existen
               OP(2) = Mid$(resul$, pos2 + 1, pos3 - pos2 - 1) 'obtengo los operandos
               OP(3) = Mid$(resul$, pos3 + 1, Len(resul$) - pos3) '  2 y 3
            Else 'no tiene tercer operando
               OP(2) = Mid$(resul$, pos2 + 1, Len(resul$) - pos2)
               OP(3) = ""
            End If
   Else 'solo tiene el primer operando (ej. DA A)
            OP(1) = Mid$(resul$, pos1, Len(resul$) - pos1 + 1)'
            OP(2) = ""
            OP(3) = ""
```

```
    End If
    OP(1) = Trim$(OP(1))'elimino
    OP(2) = Trim$(OP(2))'espacios
    OP(3) = Trim$(OP(3))'en blanco
Else 'la instruccion no tiene operandos (ej. nop)
    OPERACION$ = resul$
End If
End Sub

Sub Timer1_Timer ()
TIMER1.Interval = 0 'deshabilito
End Sub

Sub tipos_Click (index As Integer)
Select Case index
Case 0
    label2(0).Caption = "ARITMETICAS"
Case 1
    label2(0).Caption = "LOGICAS"
Case 2
    label2(0).Caption = "TRANSFERENCIA"
Case 3
    label2(0).Caption = "BOOLEANAS"
Case 4
    label2(0).Caption = "SALTO"
End Select
End Sub

Sub TRANS_Click (index As Integer)
barra(0).Enabled = False
barra(1).Enabled = False
barra(2).Enabled = False
TRANSFERENCIA
Select Case index
Case 0
    Call MODO1(50, 65, "MOV")
Case 1
    Call MODO1(66, 67, "MOVC")
Case 2
    Call MODO1(68, 71, "MOVX")
Case 3
   Call MODO1(72, 72, "PUSH")
Case 4
   Call MODO1(73, 73, "POP")
Case 5
   Call MODO1(74, 76, "XCH")
Case 6
   Call MODO1(77, 77, "XCHD")
End Select
End Sub

Sub VScroll1_Change (index As Integer)
Select Case index
Case 0   'ram externa
    direccion = 8 * VSCROLL1(0).Value'leo la direccion buscada
    For I = 0 To 7 'muestro las direcciones exploradas
            Call BUSCAR(direccion + I)
            GRID2(0).AddItem datos2(RAMx(FILA, COLUMNA)) & Chr$(9) & DATOS4(direccion + I), 9 + I
    Next I
    For I = 0 To 7
            GRID2(0).RemoveItem 1
    Next I
Case 2   'rom
    direccion = 5 * VSCROLL1(2).Value
    For I = 0 To 4
            Call BUSCAR(direccion + I)
            If CLng(direccion + I) < 65536 Then
               GRID2(2).AddItem datos2(ROMIX(FILA, COLUMNA)) & Chr$(9) & DATOS4(direccion + I), 6 + I

            '   Call previo2(2, 0, 1, i + 1, i + 1, datos2(ROMIX(FILA, COLUMNA)) & Chr$(9) & DATOS4(direccion + i))
            Else
               GRID2(2).AddItem " " & Chr$(9) & " ", 6 + I
               'Call previo2(2, 0, 1, i + 1, i + 1, " " & Chr$(9) & " ")
```

```
            End If
    Next I
    For I = 0 To 4
            GRID2(2).RemoveItem 1
    Next I
Case 3
    VSCROLL1(3).Visible = True
    GRID2(3).TopRow = VSCROLL1(3).Value + 1
Case 4
    VSCROLL1(4).Visible = True
    GRID2(4).TopRow = VSCROLL1(4).Value + 1
End Select
End Sub

Sub XCH ()
Dim AUX1, AUX2, AUX3, AUX4, T1, T2
CM = 1
AUX3 = " ": AUX4 = " ": T1 = 3195: T2 = 4980: donde = 1
GRILLA = 3: CELDA = ORIGEN + 1
rami(DESTINO) = FUENTE2 '*******EJECUCION*****
Select Case MD
Case "A,Rn"
    CODIGO(1) = &HC8 + N
    Call CODIGOS(1)
    rami(ORIGEN) = FUENTE1  '***********
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(74)
    GoSub MOVIMIENTOC
Case "A,DIREC"
    CODIGO(1) = &HC5: CODIGO(2) = ORIGEN
    Call CODIGOS(2)
    rami(ORIGEN) = FUENTE1  '*EJECUCION
    Call PREVIO(datos2(ORIGEN), datos2(FUENTE2), " ", " ")
    Call MENSAJES(75)
    GoSub MOVIMIENTOC
Case "A,@Ri"
    CODIGO(1) = &HC6 + N
    Call CODIGOS(1)
    Call PREVIO(datos2(EXTRAS), datos2(rami(EXTRAS)), datos2(ORIGEN), datos2(FUENTE2))
    INDI = 4
    Call EJECVISUAL(INDI) 'puntero
    Call MENSAJES(76)
    GoSub ARROBA2
    T1 = 3975: donde = 4
    GoSub MOVIMIENTOC
Case "XCHDA,@Ri"
    GoSub XCHD
Case Else
    MsgBox "INSTRUCCION NO RECONOCIDA"
End Select

GoTo FINALC
ARROBA2:
    GRILLA = Switch(ORIGEN > 127, 4, ORIGEN < 128, 3)
    CELDA = Switch(ORIGEN > 127, ORIGEN - 127, ORIGEN < 128, ORIGEN + 1)
    ORIGEN = Switch(ORIGEN > 127, 128 + ORIGEN, ORIGEN < 128, ORIGEN)
    rami(ORIGEN) = FUENTE1 'ejecucion con direcc.indirecto
Return
XCHD:
    CODIGO(1) = &HD6 + N
    Call CODIGOS(1)
    Call PREVIO(datos2(EXTRAS), datos2(rami(EXTRAS)), datos2(ORIGEN), datos2(FUENTE2))

    Call EJECVISUAL(4) 'PUNTERO
    '***************EJECUTO XCHD*****************
    Call MENSAJES(77)
    GoSub ARROBA2
    rami(DESTINO) = FUENTE1 And &HF0
    AUX3 = datos2(rami(DESTINO))
    Call hexdec(Right$(datos2(FUENTE2), 1), AUX1)
    rami(DESTINO) = rami(DESTINO) Or AUX1
    rami(ORIGEN) = FUENTE2 And &HF0
    AUX4 = datos2(rami(ORIGEN))
```

```
    Call hexdec(Right$(datos2(FUENTE1), 1), AUX2)
    rami(ORIGEN) = rami(ORIGEN) Or AUX2
    FUENTE1 = AUX2: FUENTE2 = AUX1
    '*******************PANTALLA**********************
    T1 = 3975: donde = 4
    GoSub MOVIMIENTOC
Return
MOVIMIENTOC:
    Call pausa(espera)
    CUADRO(17).Move 1710, T1: CUADRO(16).Move 1710, T2
    Call FLECHAS(16, 4, datos2(FUENTE1))
    Call FLECHAS(17, 4, datos2(FUENTE2))
    Call movi2(1710, 2985, 0, 16, 17)
    Call FLECHAS(16, 1, datos2(FUENTE1))
    Call FLECHAS(17, 3, datos2(FUENTE2))
    CUADRO(16).Height = 615
    For cont = T2 To T1 Step -paso
            CUADRO(16).Move 2985, cont
            CUADRO(17).Move 2985, T1 + (T2 - cont)
            Call pausa(tiempo)
    Next cont
    CUADRO(17).Move 2985, T2: CUADRO(16).Move 2985, T1
    CUADRO(16).Height = 900
    Call FLECHAS(16, 2, datos2(FUENTE1))
    Call FLECHAS(17, 2, datos2(FUENTE2))
    Call movi2(2985, 1900, 0, 16, 17)
    GoSub COMUN3C
Return

COMUN3C:
    Call previo3(5, 1, 8, datos2(FUENTE2))
    Call previo3(5, 1, donde, datos2(FUENTE1))'rami(ORIGEN)))
    LABEL1(8).Caption = " "  'BANDERAS
    Call pausa(espera)
    '**ACTUALIZACION DE LAS LOCALIDADES DE MEMORIA INVOLUCRADAS
    Call previo3(3, 2, DESTINO + 1, datos2(FUENTE2))'rami(destino))) 'ACTUALIZA EL DESTINO
    Call previo3(GRILLA, GRILLA + 5, CELDA, datos2(FUENTE1))'rami(ORIGEN)))'ACTUALIZA EL DESTINO
    GRID2(3).TopRow = DESTINO + 1
    Call pausa(espera)
Return
FINALC:
End Sub

'MODULO PARA AYUDA (AYUDA.FRM)

Dim ACTIVO%

Sub Command1_Click (Index As Integer)
Select Case Index
Case 0 'TEMAS
    PANEL3D1.Visible = True'CONTAINER DE TEMAS
Case 1 '>>
    If ACTIVO < CInt(LABEL1(0).Tag) Then
        LABEL1_CLICK (ACTIVO + 1)
    Else
        LABEL1_CLICK (0)
    End If
Case 2 '<<
    If ACTIVO > 0 Then
        LABEL1_CLICK (ACTIVO - 1)
    Else
        LABEL1_CLICK (CInt(LABEL1(0).Tag))
    End If
Case 3 'SALIR
    AYUDA.Hide
    FORM1.Show
End Select
End Sub

Sub Form_Load ()
vscroll1.Left = picture3.Left + list1.Width '- 200
vscroll1.Height = list1.Height
End Sub
```

```
Sub Form_Resize ()
picture3.Width = AYUDA.Width
picture3.Height = AYUDA.Height - 2460
list1.Width = picture3.Width - 100
list1.Height = picture3.Height
vscroll1.Top = picture3.Top
vscroll1.Left = picture3.Left + list1.Width - 220
vscroll1.Height = list1.Height

End Sub

Sub LABEL1_CLICK (Index As Integer)
Dim ARCH$, LINEA$, INICIO%, FIN%
disco = "c:\VGA"
PICTURE4.Visible = False
picture3.Visible = True
vscroll1.Visible = True
ACTIVO = Index
'LABEL1(INDEX).Enabled = False
list1.Clear
PANEL3D1.Visible = False
ARCH = disco + "HELP4.dat"
Select Case Index
Case 0   'ARCHIVO
   INICIO = 1: FIN = 21
Case 1   'INSTRUCCIONES
   INICIO = 148: FIN = 166
Case 2   'OPCIONES
   INICIO = 22: FIN = 43
Case 3   'DATOS
   INICIO = 62: FIN = 99
Case 4 'MCS-52
   INICIO = 100: FIN = 119
Case 5 'PINES
   INICIO = 44: FIN = 61
Case 6 'CONOS
   INICIO = 121: FIN = 146
   PICTURE4.Visible = True
   PICTURE4.Enabled = False
End Select
   LABEL2.Caption = LABEL1(Index).Caption
   Open ARCH For Input Access Read As #1
   For I = 1 To INICIO - 1
      Line Input #1, LINEA
   Next I
   For I = INICIO To FIN
      Line Input #1, LINEA
      list1.AddItem LINEA
   Next I
   Close #1
   vscroll1.Max = list1.ListCount - 1
End Sub

Sub VScroll1_Change ()
list1.TopIndex = vscroll1.Value
End Sub

'PANTALLA DE CREDITOS (MODULO ACERCA.FRM)

Sub Command1_Click ()
ACERCA.Hide
End Sub

Sub Form_Activate ()
COMMAND1.Visible = True
End Sub
```

BIBLIOGRAFIA

1.- Introducción a los Microcontroladores, González Vázquez José, McGraw Hill, 1992

2.- Manual Intel de Microprocesadores y Microcontroladores

3.- Fundamentos de los Microprocesadores, Tokheim Roger L., Serie Schaun, 1985

4.- Apuntes de Sistemas Microprocesados

5.- Intel MCS BASIC - 52 User's Guide

6.- Visual Basic para Windows, Bravo Vicente C., E.P.N